UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MARCOS HENRIQUE BACKES

A Graph-Cut-based Trimap Propagation Method for Video Matting

Work presented in partial fulfillment
of the requirements for the degree of
Bachelor in Computer Science

Advisor: Prof. Dr. Manuel Menezes de Oliveira Neto

Porto Alegre
December 2017

ABSTRACT

Alpha matting is an important task in image processing. It allows a user to extract foreground objects from natural images and compose them with different backgrounds, achieving subpixel-level precision. Due to its ill-posed nature, alpha matting requires additional inputs: for a single image, the user should specify the foreground, background, and unknown regions of the image using a *trimap*. However, when performing alpha matting on a video sequence, providing a trimap for each individual frame becomes tedious and time-consuming. We propose a novel method for propagating trimaps between frames of a video sequence. While most previous techniques require a binary segmentation for each frame to produce trimaps, our approach requires the user to define a trimap for only a few keyframes. To obtain trimaps for the remaining frames, we use the alpha channel (opacity map) computed for the previous frame to estimate the trimap for the next one, using a graph-cut-based approach. Results show that our algorithm is able to significantly reduce the amount of user effort for simple videos, while still resulting in considerable savings when applied to videos containing little temporal coherence.

Keywords: Image processing. Video segmentation. Alpha matting. Video matting. Graph-cuts.

Um Método de Propagação de Trimaps baseado em Graph-Cuts para Video Matting

RESUMO

Alpha matting é uma tarefa importante na área de processamento de imagens. Permite ao usuário extrair objetos de imagens naturais e compô-los com planos de fundo diferentes. Por não ser um problema bem-posto, *alpha matting* requer informações adicionais: para uma imagem, o usuário deve especificar o primeiro plano, o plano de fundo e a região desconhecida da imagem usando um *trimap*. No entanto, ao realizar *alpha matting* em uma sequência de vídeo, criar um *trimap* para cada quadro torna-se uma tarefa tediosa e demorada. Nesse trabalho, é apresentada um novo método para propagar *trimaps* entre quadros de uma sequência de vídeo. Enquanto a maioria das técnicas anteriores requer uma segmentação binária para cada quadro para produzir *trimaps*, nossa técnica requer um *trimap* para apenas alguns quadros. Para obter *trimaps* para os quadros restantes, nós utilizamos o canal alfa (opacidade) computado do frame anterior para estimar o *trimap* para o próximo, usando uma técnica baseada em *graph-cuts*. Os resultados mostram que nosso algoritmo consegue reduzir significativamente a quantidade de esforço realizado pelo usuário para vídeos simples, enquanto que, mesmo quando aplicado em vídeos com pouca coerência temporal, resulta em uma economia considerável.

Palavras-chave: Processamento de vídeos. Segmentação de vídeos. *Alpha matting*. *Video matting*. *Graph cut*.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

Max-flow	Maximum flow
Min-cut	Minimum cut
MPC	Modified PatchMatch Cost
NNF	Nearest Neighbor Field
RGB	Red, Green, Blue

CONTENTS

1 INTRODUTION

Object detection, extraction, and compositing are important tasks in image and video processing. They allow a user, for instance, to extract an object from an image or from video sequence and composite it on different backgrounds. Since digital images/videos are discrete representations, some pixels at the borders between foreground and background elements might combine colors from both. A naïve cutout and compositing of an object into another picture/movie will often exhibit objectionable artifacts, as some of its border pixels might carry color information from the original background. The task of separating foreground and background colors from a image at a subpixel level or, equivalently, computing an opacity map for foreground elements, is called *alpha matting*. When this process is applied to a video sequence, we will refer to it as *video matting*.

Given an input image I, the goal of *alpha matting* is to estimate a foreground color F_p, a background color B_p, and an opacity value α_p for each pixel p in I. This process corresponds to reversing the compositing equation

$$I_p = \alpha_p F_p + (1 - \alpha_p) B_p, \tag{1.1}$$

where I_p is the color of pixel p. The equation uses each opacity value α_p as an interpolation factor between F_p and B_p. Once all relevant F_p and α_p have been determined, one can use this information to composite the extracted elements seamlessly with another background.

Alpha matting is an ill-posed problem. Given an RGB image I, one must compute seven parameters for each pixel p (R, G, and B values for both F_p and B_p, plus α_p). Hence, it is usually required that the user provides some additional input by informing some image/frame regions that are known to belong to the background, as well as some regions that are known to belong to the foreground. This is often done using a *trimap* or a few *scribbles*. A trimap segments the image in three distinct classes of pixels: *foreground*, *background*, and *unknown* (Figure 1.1). Thus, an alpha-matting technique only needs to compute the result for the pixels in the unknown region. This simplifies the problem, since the technique can analyze both known regions to estimate the opacity of each pixel based on color similarity (affinity) in the neighborhood. Ideally, the unknown regions should be as small as possible, containing only pixels mixing foreground and background elements.

Creating a *trimap* may be a tedious task. For this reason, some applications are able to generate a trimap given only a few user scribbles (GASTAL; OLIVEIRA, 2010).

Figure 1.1: Example of alpha-matting: given an input image (a) and a trimap (b), produce the alpha channel (c). Note that in (b) known foreground and known background correspond, respectively, to white and black regions, while the unknown region is grey.

(a) Input image (b) Input Trimap (c) Output alpha channel

However, applying alpha matting to a video sequence would require too much effort if a trimap needs to created for each individual frame. To minimize this effort, some applications try to propagate trimaps between frames, requiring the user to provide trimaps for only a few key frames (CHUANG et al., 2002; JOHNSON; RAJAN; CHOLAKKAL, 2015).

While applying alpha matting to a single image is a difficult task, automatically applying it to a video is even more challenging. Like alpha matting, video matting is usually performed as two-step process. The first one consists of generating a *trimap* for every frame, while the second applies alpha-matting to each individual frame. The techniques should be able to handle problems like (WANG; COHEN et al., 2008):

- *Large data size*: algorithms should perform efficiently, since videos have a large number of pixels. Interactivity is very important since on high interactive systems, the user can quickly correct mistakes made by the program;
- *Temporal coherence*: it is necessary that the trimaps and the computed opacity maps be temporally coherent, as incoherence and jittering may result in poor quality videos;
- *Fast motions*: for videos containing fast object and/or camera motions it is often hard to maintain temporal coherence.

We propose a more user-friendly method for video matting based on trimap propagation between frames. Our approach is based on the Graph Cut algorithm (BOYKOV; VEKSLER; ZABIH, 2001), and uses the alpha matte (opacity map) computed for the pre-

vious video frame to estimate a trimap for the next one. The transfer of an alpha matte from a frame to the next is performed using a modified version of the PatchMatch algorithm (BARNES et al., 2009). While previous techniques perform trimap generation and alpha matting separately, we believe our integrated approach is able to better track foreground objects. Our experiments show that our method is able to produce high-quality results on simple video sequences, requiring very little user interaction. On scenes with fast motions, foreground/background ambiguity and occlusions, trimaps for additional key frames might be required.

The remaining of this document is organized as follows. Chapter 2 discusses some related work on video segmentation and video matting. Chapter 3 provides a brief review of the use of the graph-cut algorithm for image segmentation. Chapter 4 presents the details of our graph-cut-base trimap propagation method for video matting, while Chapter 5 discusses some results obtained with our technique. Chapter 6 closes this thesis with some conclusions and directions for future exploration.

2 RELATED WORK

Video segmentation and matting is an important problem in computer vision with many applications such as video editing and compositing. It is, therefore, used in TV and movie production to create a variety of scenes and effects. High-quality methods are usually obtained using chroma keying (AKSOY et al., 2016), which simplifies the problem significatively. However, when processing natural background videos, a high understanding of the scene is required, since defining which object must be classified as foreground depends on the user's goal. Hence, this task is usually interactive, where the user informs which objects must be segmented by completely labeling some keyframes or by drawing some scribbles. This task is called semi-supervised segmentation and, although successful approaches have been proposed for single image segmentation over the last decades (MORTENSEN; BARRETT, 1995; BOYKOV; FUNKA-LEA, 2006; GRADY, 2006), natural-video segmentation is still considered a challenge due to difficulties such as background/foreground color ambiguity, motion blur, illumination changes, fast motions, object deformation, occlusions, and camera motion.

2.1 Video Segmentation

Many video segmentation techniques proposed over the last years are based on graph cuts (WANG et al., 2005; LI; SUN; SHUM, 2005; GRUNDMANN et al., 2010; MAERKI et al., 2016). The main idea is to generate a graph for the whole video sequence where pixels from one frame are also connected to pixels in adjacent frames. One of the main problems of this approach is that, for long video sequences, the cost of representing the whole graph tends to be prohibitive. In order to reduce such cost, some techniques reduce the graph complexity by assigning nodes to frame regions or superpixels (WANG et al., 2005; LI; SUN; SHUM, 2005; GRUNDMANN et al., 2010). Recently, the work by Maerki et al. (MAERKI et al., 2016) has shown great results and efficiency by representing the video volume using the bilateral-grid data structure (CHEN; PARIS; DURAND, 2007).

Another possible approach to the problem is to transfer the segmentation mask between adjacent frames (BAI et al., 2009; ZHONG et al., 2012; FAN et al., 2015). The great advantage, in terms of time efficiency, is that there is no need to process the video as a whole; on the other hand, the algorithm might not be able to treat foreground

occlusions for example. On Video SnapCut (BAI et al., 2009), correlations between pixels in neighboring frames are estimated using optical flow (LUCAS; KANADE et al., 1981). After that, pixels are classified according to local classifiers based on color similarity and position around the border between foreground and background, which is propagated via optical flow.

The work presented by Fan et al. (FAN et al., 2015) is able to transfer the mask between temporally distant frames, resulting in an interactive and easy-to-use tool. To do so, it estimates pixel correlations using PatchMatch (BARNES et al., 2009), which has been demonstrated to produce more accurate results than optical flow for this task. After that, a final correction is applied to the current frame using an extension of the level-set technique known as Geodesic Active Contour (CASELLES; KIMMEL; SAPIRO, 1997).

2.2 Video Matting

Video-matting methods usually can be split into two main stages. The first one is *trimap propagation*, where the objective is, given trimaps for keyframes, to estimate the foreground, background and unknown regions for the remaining frames. After obtaining a trimap for each frame, the final stage is to create a temporally-coherent matting.

2.2.1 Trimap propagation

Although many techniques are able to produce high-quality trimaps with little or no user interaction, they require additional information about the scene, such as pixel depth (WANG et al., 2007), multiple cameras with different focal planes (MCGUIRE et al., 2005), or camera arrays (JOSHI; MATUSIK; AVIDAN, 2006).

Early works on natural video matting (WANG et al., 2005; LI; SUN; SHUM, 2005; BAI et al., 2009) rely on first generating a binary segmentation for each frame and then obtaining trimaps by dilating and eroding the segmented frame. The uniform width of the resulting trimap causes the unknown regions to often be too thin (resulting in loss of information), or too thick (which may introduce errors during the matting stage).

Chuang et al. (CHUANG et al., 2002) proposed a technique to propagate trimaps across video frames. It uses optical flow to estimate pixel movement between consecutive frames. A trimap is generated for each frame interpolating trimaps between key frames

using forward and backward optical flow. However, optical flow is known to be erroneous in regions with low contrast and object boundaries, and is also susceptible to errors when objects become occluded.

The method proposed by Bai et al. (BAI; WANG; SIMONS, 2011) is able to adjust the width of the trimap contour according to how fuzzy the border is, but requires an accurate segmentation for each frame. It also requires the colors of the object and background to be sufficiently distinct. Johnson et al. (JOHNSON; RAJAN; CHOLAKKAL, 2015) improved this method's coherence by tracking the object's shape across the frames.

2.2.2 Temporally-coherent alpha-matting

Although many alpha-matting techniques are able to produce high-quality results when applied to images (LEVIN; LISCHINSKI; WEISS, 2008; GASTAL; OLIVEIRA, 2010; XU et al., 2017), applying them to videos usually results in jittering because they do not preserve temporal coherence. Bai et al. (BAI; WANG; SIMONS, 2011) tries to solve this problem by smoothing transitions between obtained mattes. To do so, one represents the matting of a frame using level sets. Temporal coherence is achieved interpolating level sets between frames. Although the method successfully reduces jittering, it may excessively blur small structures.

The work by Sindeev et al. (SINDEEV; KONUSHIN; ROTHER, 2012) estimates optical flow based on the alpha-channel value (*Alpha Flow*). Alpha matte for video is then obtained using Closed Form matting (LEVIN; LISCHINSKI; WEISS, 2008), where the neighborhood of a pixel contains not only intraframe pixels but also interframe pixels estimated with *Alpha Flow*. The work shows that improved results can be obtained when the alpha channel is propagated instead of a binary segmentation.

3 GRAPH-CUT-BASED IMAGE SEGMENTATION

Image segmentation is a classic problem and perhaps one of the most important applications of image processing. Different from alpha matting, the goal is to generate a binary segmentation (foreground or background). Many applications depend on this task, including analysis of medical images, face recognition, surveillance and machine-vision. One of the algorithms which has shown great results over the last decades is based on graph cuts (BOYKOV; VEKSLER; ZABIH, 2001) and will be presented in this section.

3.1 Graph

A *graph* is a data structure which defines a set of elements and the relationships (connections) among them. A *directed graph* can be defined as an ordered pair $G = (V, E)$, where V is a set of nodes or vertices, and E is a set of edges or arcs (CORMEN, 2009). Each edge $(u, v) \in E$ represents a directed link (ordered pair) between nodes u and v, with $u, v \in V$.

A *weighted directed graph* is a graph which has a weight assigned to each of its edges. Therefore, it may be defined as an ordered tuple $G = (V, E, w)$, where $w : E \mapsto \mathbb{R}$ is a function which maps an edge to a real value. Figure 3.1a shows an example of a weighted directed graph.

3.2 Maximum-Flow Problem

The *Maximum Flow* problem (also called *max-flow*) consists of finding the maximum amount of flow that can be sent from a source node $s \in V$ to a sink node $t \in V$ in a weighted directed graph G, while respecting the maximum capacity (or weight) of each edge in G. Figure 3.1b illustrates the max-flow problem.

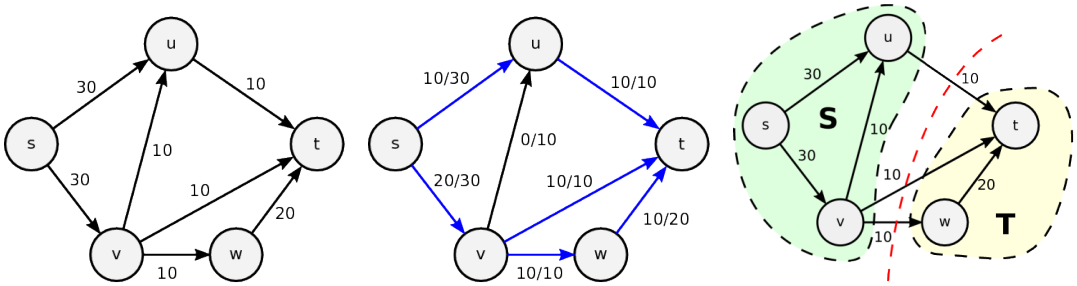
Formally, the single-source single-sink Max-flow may be defined as a linear program for a graph $G = (V, E, w)$ and its source and sink nodes s and t as

$$\begin{array}{lll} \text{maximize} & J(f) = \sum_{e \in \delta^+(s)} f_e & \\ \text{subject to} & \sum_{e \in \delta^+(v)} f_e = \sum_{e \in \delta^-(v)} f_e, & \forall v \in V \\ & 0 \leq f_e \leq w_e, & \forall e \in E \end{array} \tag{3.1}$$

where f_e denotes the amount of flux passing through edge e, $\delta^+(v)$ and $\delta^-(v)$ denote, respectively, the set of outcoming and incoming edges of node v.

Max-flow is an extensively studied problem. Several approaches have been proposed over the last 60 years presenting different strategies and time complexities, including Ford-Fulkerson ($O(VC)$, where C is an upper bound for the maximum flow in the graph) (FORD; FULKERSON, 1956), Edmonds-Karp ($O(VE^2)$) (EDMONDS; KARP, 1972), Push-Relabel ($O(V^2E)$) (GOLDBERG; TARJAN, 1988) and, most recently, Orlin et al. ($O(VE)$) (ORLIN, 2013).

Figure 3.1: Example of a weighted directed graph G with five nodes and seven edges (a). Optimal max-flow for G using s as source and t as sink (b), where the values on the edges represent "flux/capacity". Optimal min-cut for G, dividing it into twos subsets S and T (c). Note that the cut (in red) passes right through the saturated edges in (b).



(a) Graph (b) Max-flow (c) Min-cut

Source: The Authors

3.3 Minimum-Cut Problem

The *minumum-cut problem* (also called *min-cut*) consists of finding, for a graph $G = (V, E, w)$ with a source node s and a sink node t, a partition $S \cup T = V$ where $s \in S$, $t \in T$ and $S \cap T = \varnothing$ such that the sum of the capacities of edges between S and T is minimal. Figure 3.1c illustrates the min-cut problem.

Min-cut may be defined as the linear program

$$\begin{aligned} \text{minimize} \quad & J(S,T) = \textstyle\sum_{e \in E} w_e d_e \\ \text{subject to} \quad & d_{s,t} = 1, \\ & S \cap T = \varnothing, \end{aligned} \tag{3.2}$$

where w_e is the weight associated to edge $e = (u, v)$, and function $d_{u,v}$ evaluates to 0 if

$u, v \in S$ or $u, v \in T$; otherwise, it evaluates to 1.

As proven in the *max-flow min-cut theorem* (CORMEN, 2009), if $G = (V, E, w)$ is a graph with sink s and source t then the value of the max-flow equals the value of the min-cut in G. Furthermore, the min-cut optimal solution can be obtained by partitioning G through the edges saturated by max-flow.

3.4 Image Segmentation

The first work based on graph energy minimization (max-flow/min-cut) in image processing was presented by Greig et al. (GREIG; PORTEOUS; SEHEULT, 1989). Originally, the method was applied to image restoration, but since then many adaptations have proposed for different applications (BOYKOV; KOLMOGOROV, 2004), including stereo correspondence (ROY; COX, 1998), shape reconstruction (SNOW; VIOLA; ZABIH, 2000), and image segmentation (BOYKOV; FUNKA-LEA, 2006).

Consider an image I and the problem of classifying each pixel $p \in I$ as foreground $\mathcal{F}$ or background $\mathcal{B}$. A feasible solution may be represented as a partition $\mathcal{F} \cup \mathcal{B} = I$ such that $\mathcal{F} \cap \mathcal{B} = \varnothing$. Usually, the goal of graph-based image processing algorithms is to minimize a defined energy function. For binary image segmentation, this function can be represented as (BOYKOV; KOLMOGOROV, 2004)

$$J(\mathcal{F}, \mathcal{B}) = \lambda \sum_{p \in I} D_p + \sum_{(p,q) \in \mathcal{F} \times \mathcal{B} | p \in N(q)} V_{p,q}, \tag{3.3}$$

where D_p is called the data term, $V_{p,q}$ the smoothness term, $N(q)$ is the set of neighboring pixels of q, and λ is a constant that defines the relative importance of the data term versus the smoothness term.

The data term D_p represents the penalty of assigning a pixel p to foreground or background. It may be defined as

$$D_p = \begin{cases} P_{\mathcal{F}}(p), & \text{if } p \in \mathcal{B}; \\ P_{\mathcal{B}}(p), & \text{if } p \in \mathcal{F}, \end{cases} \tag{3.4}$$

where $P_{\mathcal{F}}(p)$ represents the likelihood of p belonging to the foreground, and is typically defined based on its color/intensity or proximity to other foreground pixels. It can also be interpreted as the probability of p belonging to $\mathcal{F}$, for $P_{\mathcal{F}}(p) + P_{\mathcal{B}}(p) = 1$. The case of

$P_{\mathcal{B}}(p)$ is analogous.

The smoothness term $V_{p,q}$ represents the penalty of assigning neighbouring pixels p and q different labels. In this sense, it should produce a big penalty if p and q have similar color intensities and a small penalty otherwise. It is typically defined as

$$V_{p,q} = \frac{e^{-\frac{\|I_p - I_q\|^2}{2\sigma^2}}}{d_{pq}}, \tag{3.5}$$

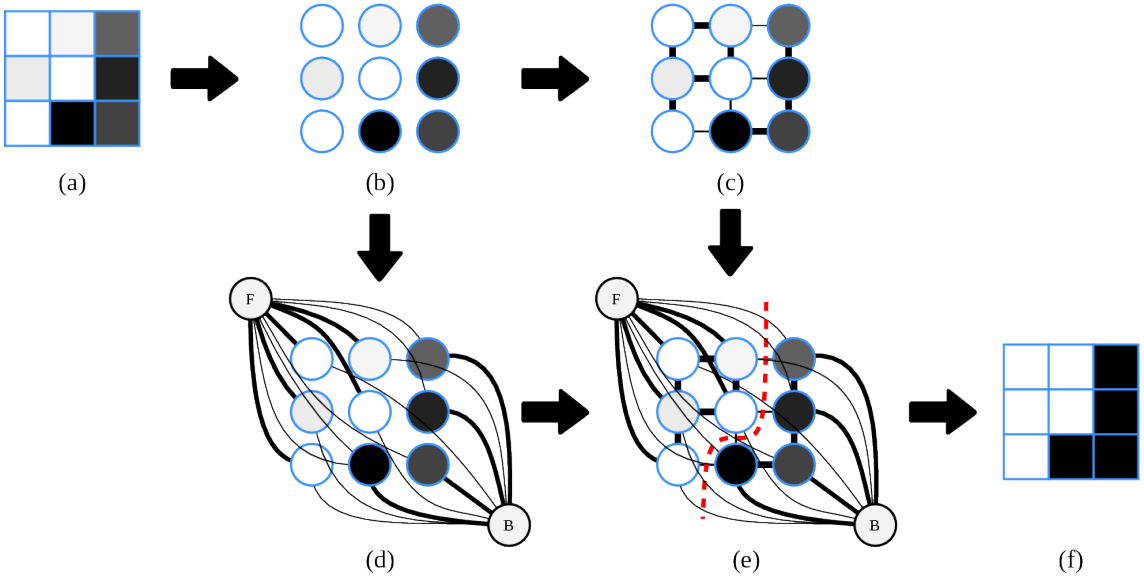
where I_p and I_q are the colors/intensities of neighbor pixels p and q, σ is the standard deviation of a Gaussian function, and d_{pq} is the distance between p and q in pixel space. Note that $V_{p,q}$ evaluates to a high number when $|I_p - I_q| < \sigma$, resulting in a high penalty when separating similar pixels. In our method, we used $\sigma = 1$, since, empirically, it generates good results for the tested video sequences.

The optimal solution for the cost function in Equation 3.3 can be obtained via max-flow/min-cut algorithms. Figure 3.2 depicts the whole segmentation process. First, we must create a weighted graph $G = (V, E, w)$. Each pixel in I corresponds to a node in V (Fig. 3.2a). There are two additional terminal nodes: F and B, representing, respectively, foreground and background.

The set of edges E consists of two types: *n_link* (neighborhood links) and *t_link* (terminal links). Each pair of neighboring pixels (p, q) has an *n_link* edge with weight defined by $V_{p,q}$ (Fig. 3.2c). Also, each pixel p is connected to both terminal nodes F and B via *t_links* defined by $P_{\mathcal{F}}(p)$ and $P_{\mathcal{B}}(p)$, respectively (Fig. 3.2d).

The optimal min-cut solution results in a partition $\mathcal{F} \cup \mathcal{B}$ (Fig. 3.2e) which yields the optimal segmentation for image I. Boykov et al. (BOYKOV; VEKSLER; ZABIH, 2001) present an efficient max-flow/min-cut implementation for graphs based on images.

Figure 3.2: Example of image segmentation using graph-cuts: Input 3x3 monochromatic image (a); image representation as graph nodes (b); *n_link* edges (c), the thicker the edge, the bigger the weight; *t_link* edges (d), assuming color white as foreground; optimal min-cut as dashed red line (e); resulting binary segmentation of the input image (f).



Source: The Authors

4 GRAPH-CUT-BASED TRIMAP PROPAGATION FOR VIDEO MATTING

The proposed method for semi-supervised trimap generation for video sequences is presented in this chapter. It assumes that the user provides a video with a trimap and alpha matte for the first frame of the video sequence, although some methods can be used to generate the initial trimap and alpha matte from a set of scribbles (GASTAL; OLIVEIRA, 2010). Figure 4.1 summarizes our method. For each frame, given a trimap and alpha matte for frame I_t, the trimap is propagated to frame I_{t+1} according to the following steps:

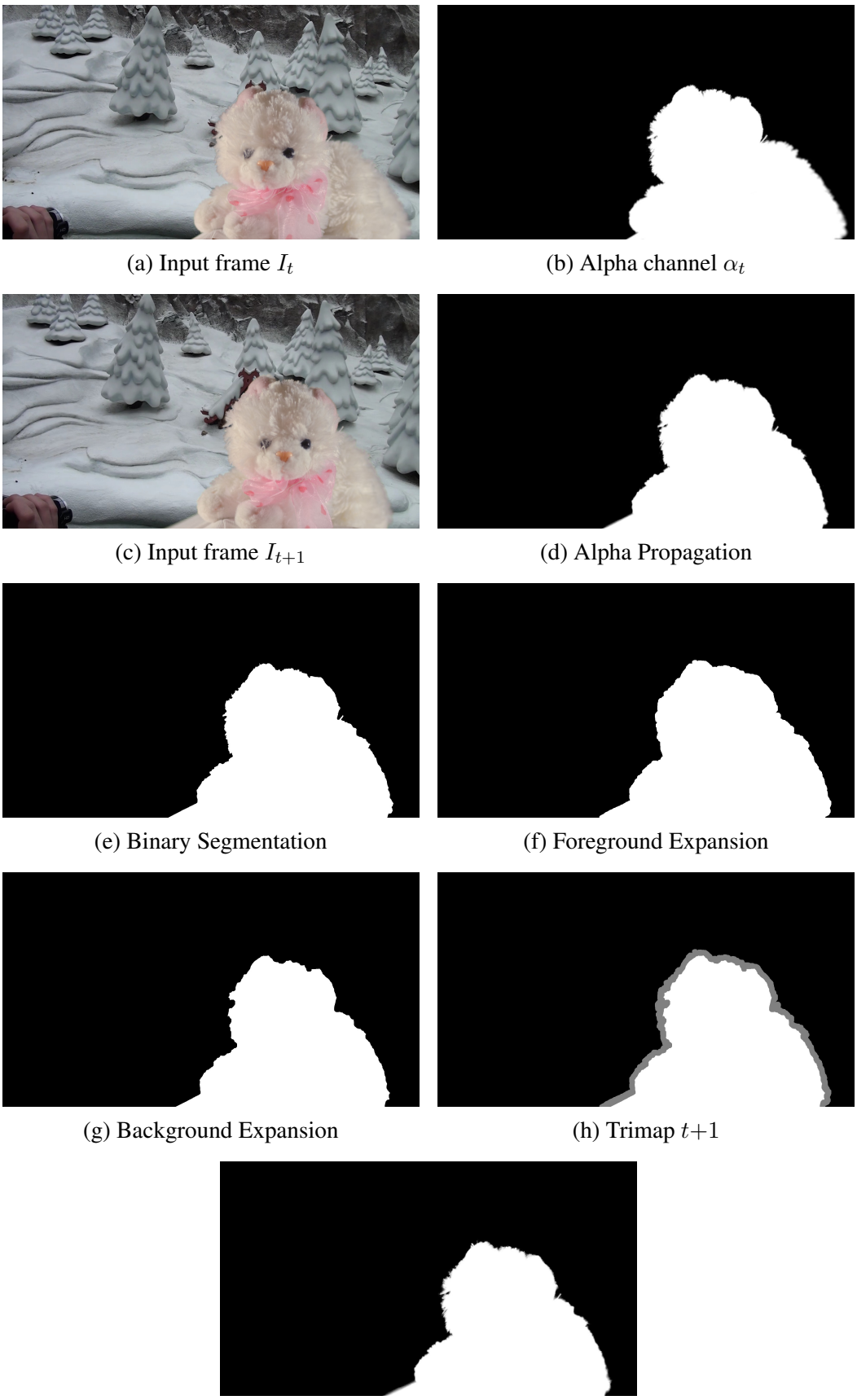
1. *Alpha Propagation* – the alpha channel of frame I_t is maped to frame I_{t+1} using PatchMatch (BARNES et al., 2009), taking into account camera and object motion. The result is an estimation of alpha for each pixel in frame I_{t+1} (Figure 4.1d).
2. *Binary Segmentation* – using the estimated alpha for each pixel as the value of the *t-link* edges, frame I_{t+1} is segmented using graph-cut binary segmentation (Figure 4.1e).
3. *Foreground/Background Expansion* – the trimap for frame I_{t+1} is generated by expanding both the foreground (Figure 4.1f) and background regions (Figure 4.1g). The trimap's unknown region results from the intersection of the expanded regions (Figure 4.1h). Different from simple trimap propagation methods, the width of the unknown region is not uniform. It varies for each pixel according to its estimated alpha value and neighborhood color similarity, as explained in Section 4.3.
4. *Alpha Matting* – finally, given the computed trimap, alpha matting is performed using Shared Matting (GASTAL; OLIVEIRA, 2010) (Figure 4.1i). Note that any alpha matting solution (e.g., (LEVIN; LISCHINSKI; WEISS, 2008)) could be used at this stage.

The details of these steps are presented in the following sections.

4.1 Alpha Propagation

To generate a trimap for frame I_{t+1} from frame I_t, first we obtain a good estimate of the alpha value for each pixel in I_{t+1}. To do so, we compute the pixel displacements between frames I_t and I_{t+1}. However, we use Patchmatch (BARNES et al., 2009), which has been demonstrated to achieve more accurate results than optical flow (FAN et al.,

Figure 4.1: Overview of the proposed method.



(a) Input frame I_t

(b) Alpha channel α_t

(c) Input frame I_{t+1}

(d) Alpha Propagation

(e) Binary Segmentation

(f) Foreground Expansion

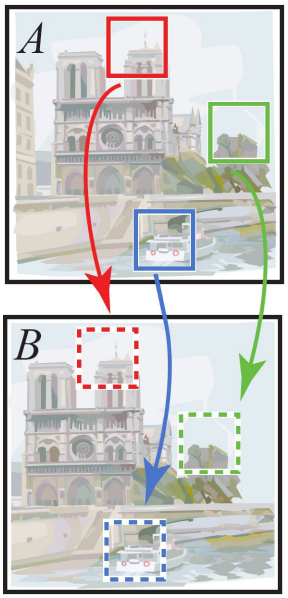
(g) Background Expansion

(h) Trimap $t+1$

(i) Alpha matting $t+1$

Source: The Authors

Figure 4.2: Nearest Neighbor Field (NNF) problem: given two images A and B, find for every patch in A the best matching patch in B.



Source: Barnes et al. (2009)

2015). Patchmatch computes the nearest neighbor field (NNF) between two images. Thus, given two images A and B, the goal is to find for each patch P_A of size M in A the patch P_B (of same size) in B which is the most similar to P_A. Figure 4.2 illustrates the idea. Patchmatch is able to obtain a good approximation for the solution using an efficient randomized method.

The foreground and background regions in a video sequence tend to displace in different ways (FAN et al., 2015). Usually, the background moves according to the motion of the camera with respect to the scene, while the foreground object moves in a different direction. Thus, similar to (FAN et al., 2015), we estimate two NNFs from frame I_{t+1} to I_t: one for the foreground and other for the background.

To obtain more meaningful NNFs, we independently align the background and the foreground of frames I_t and I_{t+1}. To do so, we first perform feature matching between both frames using SURF (BAY et al., 2008), taking into account only features in the background (foreground) of I_t. The pixels of frame I_t are then translated by the (rounded) mean displacement between the matched features to generate an image I_t^b (I_t^f), which tries to align its background (foreground) to the one of frame I_{t+1}.

Given I_t^b and I_t^f, we compute two NNFs using a modified version of PatchMatch: one from I_{t+1} to I_t^b and another from I_{t+1} to I_t^f. To exploit the benefits of the translated images to compute more meaningful NNFs, we modified the original PatchMatch algo-

rithm to restrict its search to a local neighborhood around the coordinates of the source pixel position. Thus, since the images have been pre-aligned, we initialize each patch with a zero offset, as opposed to using a random offset as done in the original version of the algorithm. Also, we limit the search to a squared window whose side corresponds to one third of the frame's diagonal. Finally, we modified the PatchMatch cost function to include an Euclidian distance term between the centers of the patches. Such a term favors local matches.

Given both NNFs, the final NNF from I_{t+1} to I_t is obtained by selecting for each pixel the match with smaller modified PatchMatch cost (MPC)

$$MPC = D(P_{t+1}(p), P_t(p')) + ||p - p'||, \tag{4.1}$$

where $D(P_{t+1}(p), P_t(p'))$ is the original PatchMatch cost function computed as the sum of absolute differences of the colors/intensities of the corresponding pixels in the patches centered at pixel p in frame I_{t+1} and at pixel p' in frame I_t. $||p - p'||$ is the Euclidean distance between the centers of the corresponding patches. Finally, the alpha channel value estimated for each pixel p in frame $t+1$ is the alpha value from pixel p' in frame t.

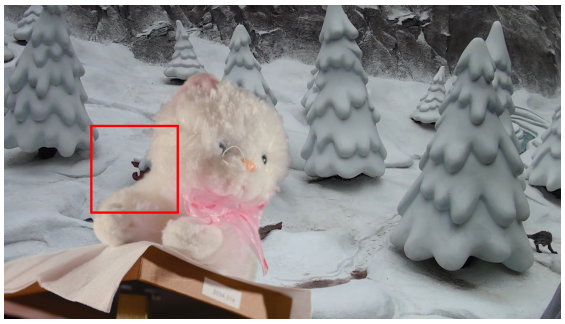
Equation 4.1 needs to be evaluated for each pixel in frame I_{t+1}. High matching costs are associated with sudden changes in the neighborhood of pixel p from one frame to another, such as the appearance of previously occluded objects, as shown in Figure 4.3. While one could ask the user to classify these new elements, since most often they result from disocclusions, whenever the cost of a pixel is higher than a defined threshold, it is reasonable to classify it as a background pixel. Furthermore, if one assumes the existence of a single foreground object, the foreground region should be connected and small isolated components that might have been possibly incorrectly considered as foreground can be confidently classified as background.

4.2 Binary Segmentation

After obtaining an estimate for the alpha channel of each pixel in frame $t+1$, we can perform a binary segmentation. To do so, we create a graph for I_{t+1} as showed in Section 3.4 using, for each pixel p, α_p as the weight of edge (p, F) and $(1 - \alpha_p)$ as the weight of (p, B). Segmented image L_{t+1} is computed using graph cut (BOYKOV; VEKSLER; ZABIH, 2001).

Figure 4.3: High matching cost due to disocclusion of background elements from frame I_t (a) to frame I_{t+1} (b), highlighted by the red square. Such situations often cause PatchMatch to incorrectly estimate some alpha values (c). Corrected alpha values after treating high-cost matchings and isolated small components as part of the background (d).

(a) Input frame I_t



(b) Input frame I_{t+1}

(c) Incorrectly estimated alpha values.

(d) Corrected alpha.

Source: The Authors

4.3 Foreground and Background Expansions

Here we explain how to generate a trimap from the binary segmentation, in such a way that the width of the unknown region varies according to the fuzziness of the border. The Graph Cut algorithm produces binary segmentations. In order to obtain a trimap from such a segmentation, we expand both the foreground and the background regions by incorporating fuzzy pixels from the complementary regions. The intersection between those expanded areas defines the trimap unknown region. Figure 4.4 illustrates the concept.

Recall the graph-cut cost function from Section 3.4:

$$J(\mathcal{F},\mathcal{B}) = \lambda \sum_{p \in I} D_p + \sum_{(p,q)\in \mathcal{F}\times\mathcal{B} | p \in N(q)} V_{p,q} \tag{4.2}$$

As an example, let I be a 1-dimensional image, as shown in Figure 4.5a, whose optimal segmentation is a cut at position b. Let the target trimap region be the region between a and c. A plot of the cost of the cut versus its position is shown in Figure 4.5b. The cost in minimum at position b and increases towards both a and c. Therefore, the intuition behind our method is to obtain a trimap by, starting from a graph-cut optimal solution, incrementally advancing the foreground (background) frontier by incorporating pixels with similar colors or low probability of belonging to the foreground (background) in a controlled way, until a certain cost is reached.

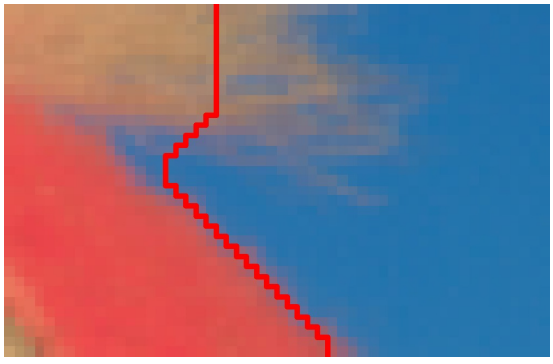
We define the cost of changing the label of a pixel p from background to foreground as

$$C_{\mathcal{F}}(p) = \lambda * (P_{\mathcal{B}}(p) - P_{\mathcal{F}}(p)) + \sum_{q \in N_{\mathcal{B}}(p)} V_{p,q} - \sum_{q \in N_{\mathcal{F}}(p)} V_{p,q}, \tag{4.3}$$

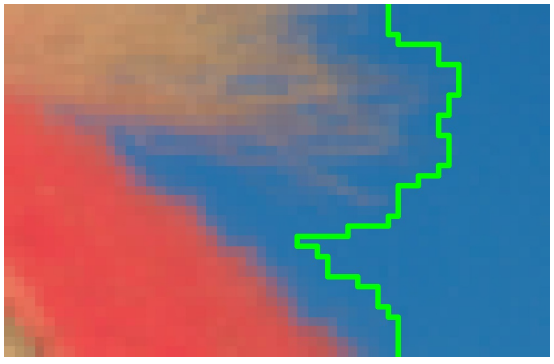
where $N_{\mathcal{F}}(p)$ and $N_{\mathcal{B}}(p)$ are the sets of neighbor pixels of p labeled as foreground and background, respectively. This function determines how much the cost of the cut increases when a pixel is switched from background to foreground. An intuition about this cost function is presented in Figure 4.6. Suppose we want to know how much the cost varies when switching the highlighted pixel p (green) to the foreground in Figure 4.6a. The neighborhood of p is represented by the graph shown in Figure 4.6b. Note that currently $J(\mathcal{F},\mathcal{B}) = P_{\mathcal{F}}(p) + V_{p,s}$, since the cost of the cut is the sum of the weights of the edges it passes through. Once we move p to the foreground (Figure 4.6c), the cut no longer passes through edges (p, F) and (p, s). Hence, to compute $C_{\mathcal{F}}(p)$ we must subtract the weight

Figure 4.4: Optimal segmentation lies inside trimap's unknown region. Therefore, the trimap can be obtained by expanding the foreground and background regions.

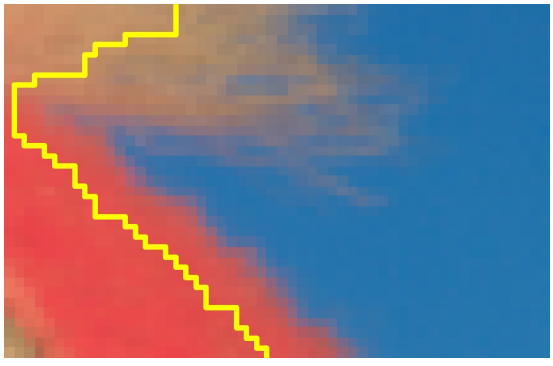
(a) Input image I



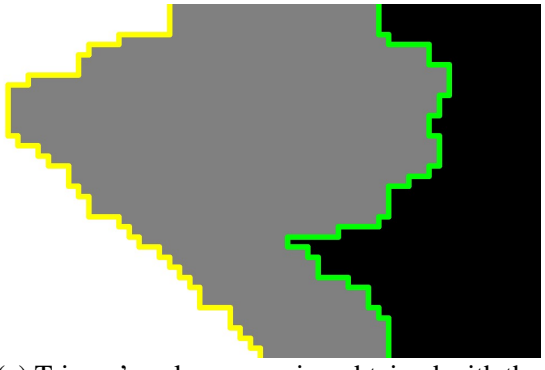
(b) Optimal segmentation (red line) in highlighted region of (a)



(c) Expanded foreground (green line)



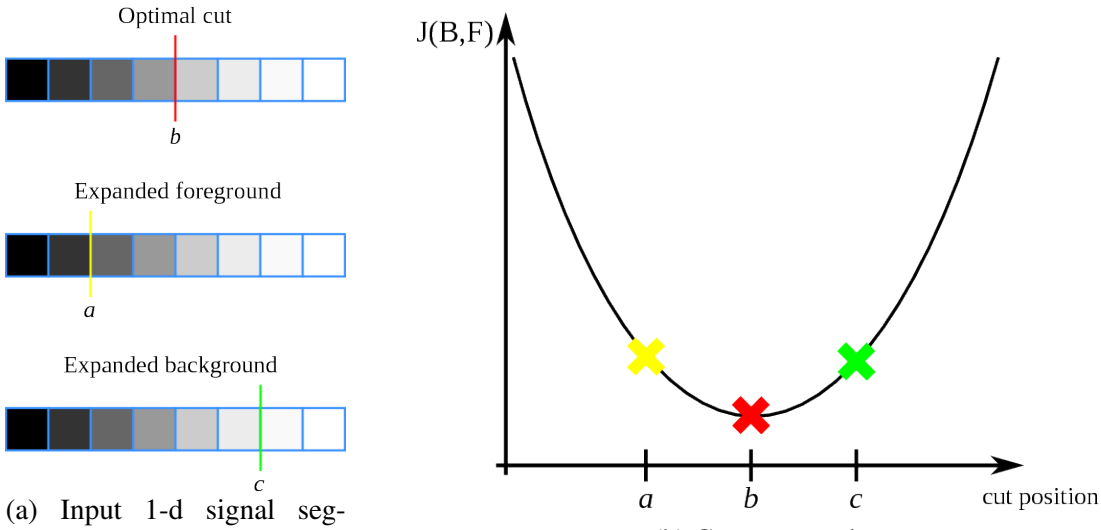
(d) Expanded background (yellow line).



(e) Trimap's unknown region obtained with the intersection of expanded regions.

Source: The Authors

Figure 4.5: Variation of graph-cut cost when signal is cut at different positions.



(a) Input 1-d signal segmented in different positions

(b) Cut cost graph

Source: The Authors

of those edges. Finally, we must sum $P_{\mathcal{B}}(p)$ and the weight of the edges between p and its neighbors in the background, which results in Equation 4.3. Note that this cost is lower for pixels which have a low probability of belonging to the background or are similar to their neighbouring pixels in the foreground.

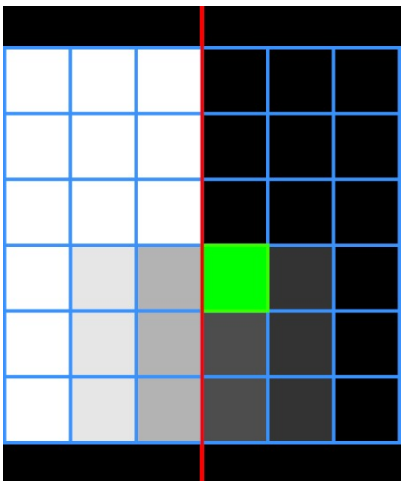
Similarly, the cost of changing a pixel from foreground to background is defined as

$$C_{\mathcal{B}}(p) = \lambda * (P_{\mathcal{F}}(p) - P_{\mathcal{B}}(p)) + \sum_{q \in N_{\mathcal{F}}(p)} V_{p,q} - \sum_{q \in N_{\mathcal{B}}(p)} V_{p,q}. \tag{4.4}$$

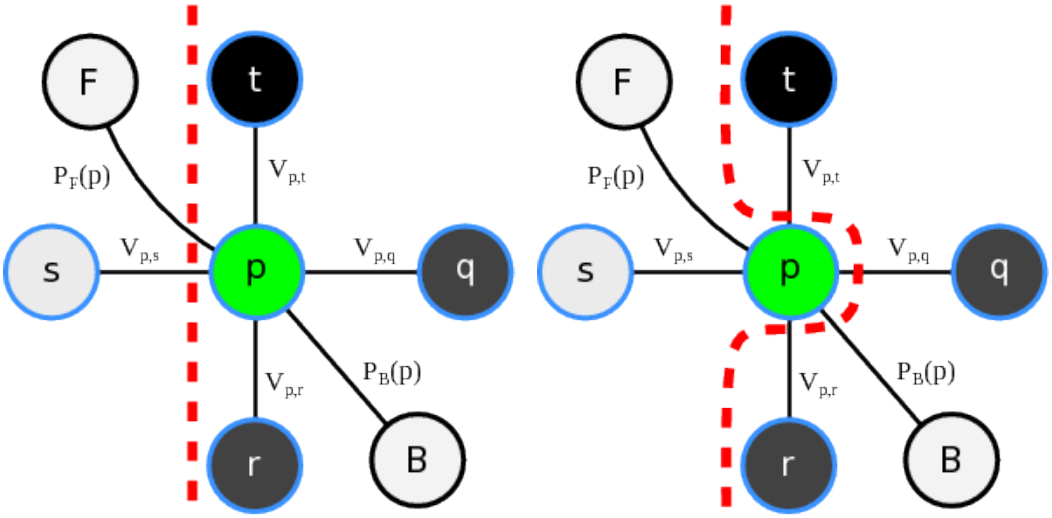
To expand the foreground region, we consider only pixels located on the border between foreground and background. At each iteration, we calculate $C_{\mathcal{F}}(p)$ for every pixel p labeled as background which are located on the border with the foreground. We then select the pixel with the lowest cost and add it to the foreground. We continue expanding the frontier until the cost of assigning a pixel is higher than a threshold $\delta < \lambda$. Note that if we implement this method using a strategy similar to Prim's algorithm (CORMEN, 2009) (using a priority queue), the time complexity is $O(N \log(N))$, where N is the number of pixels in the video frame. In practice, however, the average complexity is lower, since the number of visited pixels is proportional to the size of the unknown region of the trimap, which is usually much smaller than the size of the frame.

The background expansion is obtained similarly. As mentioned before, once we have the expanded foreground and background, the unknown region of the trimap is ob-

Figure 4.6: Cost function intuition



(a) Segmented image



(b) Neighborhood of background pixel p (c) p changed to foreground

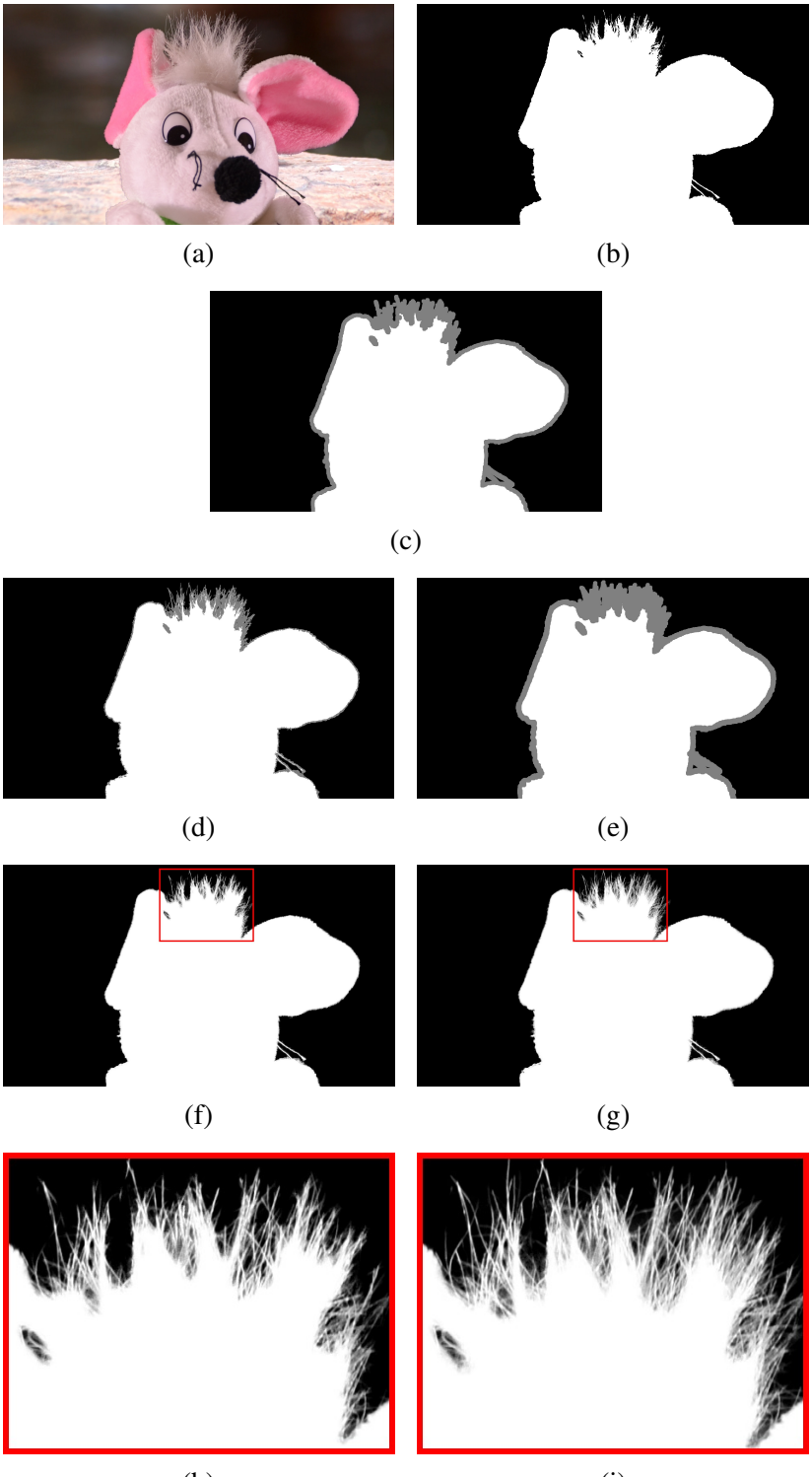
Source: The Authors

tained as the intersection of these expanded regions. Finally, it is usually a good idea to slightly dilate the obtained unknown region to make sure it covers the whole object. Figure 4.7 illustrates the various stages of our trimap generation and alpha channel estimation.

4.4 Alpha Matting

Once the trimap is generated, we can finally compute the alpha matte for frame I_{t+1} (Figure 4.7 (f)). We use Shared Matting (GASTAL; OLIVEIRA, 2010), since it is very fast and produces good results. Note, however, that our method can be used

Figure 4.7: Automatic trimap and alpha-channel computation. (a) Input image. (b) Binary segmentation using graph-cut (Section 4.2). (c) Simple trimap obtained by dilating (b) (10px width). (d) Unknown region obtained by intersecting the expanded background/foreground from (b). (e) Actual trimap obtained by dilating (d) (10px width). (f) Alpha matte for (a) obtained applying alpha matting (GASTAL; OLIVEIRA, 2010) to the trimap in (c). (g) Alpha matte obtained applying alpha matting (GASTAL; OLIVEIRA, 2010) to the trimap (e). (h) Highlighted region from (f). (i) Highlighted region from (g). Note that (h) misses some details of the hair, while (i) is more accurate.



Source: The Authors

with any alpha-matting strategy. It is important to mention that the more accurate the matting technique, the better is the trimap propagation, as errors in the matting stage might propagate to subsequent frames.

4.5 Propagating between Multiple Keyframes

So far, we have explained how to propagate trimaps forward, given that the user has provided an accurate trimap for the first frame. However, for situations involving fast object or camera motions the user might need to provide trimaps for multiple keyframes. In such situations, forward propagation only may result in temporal discontinuities in the trimap sequence. Therefore, we must interpolate the trimaps between keyframes to make sure that all transitions are as smooth as possible.

To propagate a trimap between keyframes t and $t+k$ we first propagate the trimaps forward, while also keeping track on the amount of error propagated in each frame, that is the accumulated PatchMatch cost. As described in Section 4.1, we use PatchMatch to compute correlations between pixels in frames t and $t+1$, and, then, we use it to map the alpha channel of frame t to frame $t+1$. We also use the correlations obtained by PatchMatch to calculate an accumulated error estimate. Whenever a pixel p in I_{t+1} is matched with a pixel q in I_t, the accumulated error for pixel p is equal to the modified PatchMatch cost plus the accumulated error for q.

Once we have propagated the trimaps forward, we perform a second pass propagating trimaps backwards starting at frame $t+k$, while also keeping track of the cummulative error. On this pass, whenever we warp the alpha channel, we select, for each pixel, between the forward and backward propagation, the propagation with the smallest error.

5 RESULTS

Our method was implemented in Matlab with C++ bindings for the most time-intensive routines. Segmentation, PatchMatch, and alpha-matting are performed using the authors' source code (BOYKOV; VEKSLER; ZABIH, 2001; BARNES et al., 2009; GASTAL; OLIVEIRA, 2010). The tests were performed on a Intel i7-4970k quad-core processor (4.0Ghz) with 8Gb RAM on Ubuntu 16.04 operating system. On average, for 1080p videos, our approach takes about fifteen seconds to transfer a trimap between two frames, using the multithreaded version of PatchMatch. The current version of the code has not been optimized and the only stages that cannot be paralelized are the segmentation and the foreground/background expansion. Thus, we would expect a significant speedup from an optimized GPU implementation.

For the results shown in this chapter, we perceived that the best set of parameters for our method is $\lambda = 100$, $\delta = 0.9\lambda$ and $\sigma = 1$, since these, in general, generated the best results. Therefore, we set these parameters as default for our method.

We compared our approach with the work by Chuang et al. (CHUANG et al., 2002). We generated trimaps for the video sequences provided by the author and compared the number of keyframes necessary to generate an accurate trimap for the entire sequence. Since the keyframes of the previous work were not available, we drew trimaps as necessary. Table 5.1 shows the comparison. See Apendix A for visualization of results.

Table 5.1: Comparison between number of key frames needed by Chuang et al. and by our method.

Video	#Frames	(CHUANG et al., 2002)	Ours
amira	91	10	6
kim	101	11	6

A comparison with Bai et al. (BAI; WANG; SIMONS, 2011) and Jonson et al. (JOHNSON; RAJAN; CHOLAKKAL, 2015) would be ideal. However, since neither the source code nor the video benchmark used by the authors is publicly available, this comparison was not possible.

Finally, we run tests using the Video Matting Benchmark (EROFEEV et al., 2015). This benchmark consists on a series of challenging video sequences, with complex scenes containing difficult problems related to video matting such as fast motions, color ambiguity, and foreground with a large number of transparent pixels. The benchmark provides a trimap for each frame of the video sequences. We selected the mininal amount of uniformly distributed trimaps as key frames that our method requires to produce trimaps for

the sequence. Table 5.2 shows the number of keyframes needed to generate a trimap for each frame. Video frames are shown in Appendix A.

Table 5.2: Number of keyframes needed by our method to recover trimaps for the entire video sequences from the Video Matting Benchmark (EROFEEV et al., 2015).

Video	#Frames	#Key frames
Alex	150	16
concert	200	2
Dmitriy	150	31
Slava	150	31
snow	160	17
Vitaliy	149	16

We conclude that for simple scenes, such as videos from (CHUANG et al., 2002) our method is able to accurately generate good trimaps with little user interaction. However, as the scene complexity increases (i. e., fast motions, background and foreground color ambiguity) such as found in the *Dmitriy* sequence, one trimap may be required every five frames and, still, there might be some errors in the extracted matte.

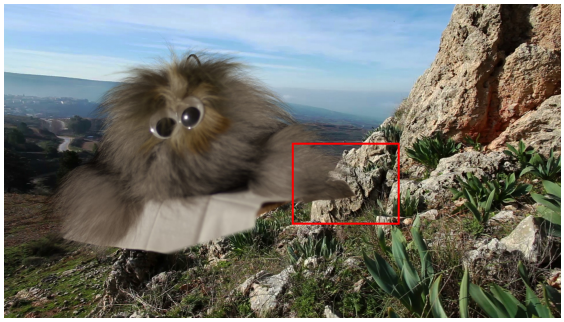
The results we have presented were obtained using trimaps every k key frames, since the trimaps were provided by the benchmark. Alternatively, we foresee an interactive application where the user initially provides two trimaps (for the first and last frames). After our technique computes the alpha mattes for the intermediate frames, the user can use scribbles to interactively correct errors. Such corrections would then be automatically incorporated and propagated to other frames, minimizing the need for user intervention.

5.1 Limitations and Future Work

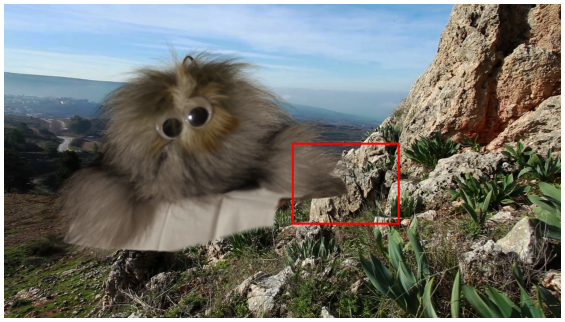
Although our method is able to significantly reduce the amount of user input required for simple scenes, several challenges still need to be overcome. First, our algorithm is highly dependent on the quality of the alpha matte provided by the alpha matting algorithm. Thus, inaccuracies in the computed matte (e.g., a high alpha for a background pixel) might be propagated to the subsequent frames, degrading the quality of the results over time (Figure 5.1). This is the reason why we were unable to generate trimaps for some of the video sequences in the Video Matting Benchmark. A possible solution could be trying to automatically detect and correct these problems or using another matting technique.

Although PatchMatch (BARNES et al., 2009) tends to produce better results than

Figure 5.1: Example of error generated by Shared Matting due to a very complex scene. A small error is generated in frame 0 and is increased when propagated to frame 5.

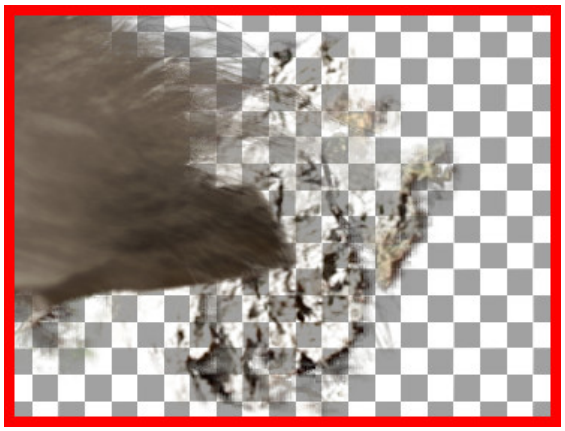


(a) *Artem* frame 0



(b) *Artem* frame 5

(c) Alpha matte of highlighted region in (a)

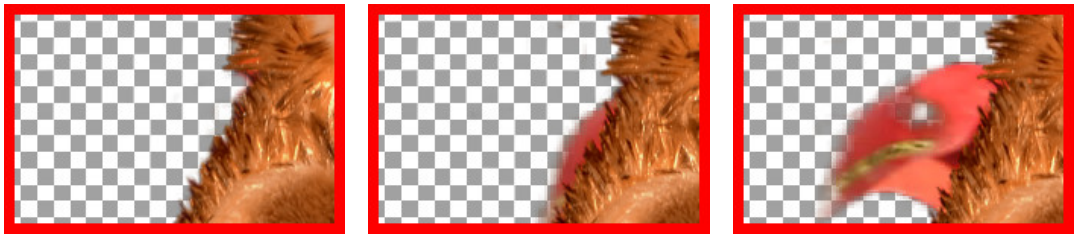


(d) Alpha matte of highlighted region in (b)

Source: The Authors

Figure 5.2: Example of error generated by PatchMatch on a video sequence with fast motions and color ambiguity. Note that the fast moving motorcycle is classified wrong due to fast motions and background occlusion.

(a) *Dmitriy* frame 139 (b) *Dmitriy* frame 140 (c) *Dmitriy* frame 141



(d) Alpha matte of highlighted region in (a) (e) Alpha matte of highlighted region in (b) (f) Alpha matte of highlighted region in (c)

Source: The Authors

optical flow methods, our technique still requires a considerable amount of user input to perform well on scenes with fast motions and ambiguity between background and foreground colors (Figure 5.2). An accurate method for movement estimation could reduce the number of keyframes required for complex scenes. Furthermore, on many video sequences of the Video Matting Benchmark, only part of the foreground object is shown in the beginning and, as the video continues, more of the object is revealed. In some of these situations, our method may fail to correctly classify parts of these initially occluded objects.

Shared Matting was designed for alpha matting of images. As such, it does not consider temporal coherence when computing the matte, which may result in jittering. Applying a temporally-coherent matting algorithm on the obtained trimaps should improve the results.

Finally, as already mentioned, our implementation has not been optimized. Currently, it was not practical to create an interactive editing application due to the relatively long execution time for each frame. A faster implementation would result in more responsiveness, enabling the user to quickly correct any misclassification produced by our method.

6 CONCLUSION

We presented a novel method for trimap propagation in video sequences. The user only needs to specify trimaps for a few keyframes to obtain trimaps for the entire sequence. Different from other approaches which first segment the video and then generate a trimap, our method generates trimaps based on the alpha matte obtained for the previous frame. If the matte is correct, it should provide more relevant information for pixel classification than a binary segmentation.

We use an adaptation of the modified PatchMatch algorithm (BARNES et al., 2009) proposed by Fan et al. (FAN et al., 2015) to propagate the alpha matte of a given video frame to the next one. Given the propagated alpha channel, we obtain a binary segmentation for the current video frame using graph cut (BOYKOV; VEKSLER; ZABIH, 2001). We then proposed a method for expanding the unknown region of the trimap based on the dynamic update of the graph-cut segmentation cost function. This expansion gives priority to *fuzzy* pixels, resulting in an adaptively-defined trimap. Together with the method for video matting, these extensions to PatchMatch and graph cuts are the main contributions of our work.

Our results show that our method is able to handle simple scenes requiring little user interaction. We have compared our method to the technique proposed by (CHUANG et al., 2002) and obtained similar results using approximately 60% of the number of keyframes used in their work. For complex scenes, our method may require a keyframe about every five frames. Improvements could be achieved by using more accurate methods for estimating and propagating the alpha matte. Also, the use of a temporally-coherent alpha-matting technique should reduce the occurrence of jittering.

REFERENCES

AKSOY, Y. et al. Interactive high-quality green-screen keying via color unmixing. **ACM Transactions on Graphics (TOG)**, ACM, v. 35, n. 5, p. 152, 2016.

BAI, X.; WANG, J.; SIMONS, D. Towards temporally-coherent video matting. In: SPRINGER. **MIRAGE**. [S.l.], 2011. v. 11, p. 63–74.

BAI, X. et al. Video snapcut: robust video object cutout using localized classifiers. In: ACM. **ACM Transactions on Graphics (ToG)**. [S.l.], 2009. v. 28, n. 3, p. 70.

BARNES, C. et al. PatchMatch: A randomized correspondence algorithm for structural image editing. **ACM Transactions on Graphics (Proc. SIGGRAPH)**, v. 28, n. 3, ago. 2009.

BAY, H. et al. Speeded-up robust features (surf). **Computer vision and image understanding**, Elsevier, v. 110, n. 3, p. 346–359, 2008.

BOYKOV, Y.; FUNKA-LEA, G. Graph cuts and efficient nd image segmentation. **International journal of computer vision**, Springer, v. 70, n. 2, p. 109–131, 2006.

BOYKOV, Y.; KOLMOGOROV, V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 26, n. 9, p. 1124–1137, 2004.

BOYKOV, Y.; VEKSLER, O.; ZABIH, R. Fast approximate energy minimization via graph cuts. **IEEE Transactions on pattern analysis and machine intelligence**, IEEE, v. 23, n. 11, p. 1222–1239, 2001.

CASELLES, V.; KIMMEL, R.; SAPIRO, G. Geodesic active contours. **International journal of computer vision**, Springer, v. 22, n. 1, p. 61–79, 1997.

CHEN, J.; PARIS, S.; DURAND, F. Real-time edge-aware image processing with the bilateral grid. **ACM Transactions on Graphics (TOG)**, ACM, v. 26, n. 3, p. 103, 2007.

CHUANG, Y.-Y. et al. Video matting of complex scenes. **ACM Transactions on Graphics**, v. 21, n. 3, p. 243–248, July 2002. Sepcial Issue of the SIGGRAPH 2002 Proceedings.

CORMEN, T. H. **Introduction to algorithms**. [S.l.: s.n.], 2009.

EDMONDS, J.; KARP, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. **Journal of the ACM (JACM)**, ACM, v. 19, n. 2, p. 248–264, 1972.

EROFEEV, M. et al. Perceptually motivated benchmark for video matting. In: **BMVC**. [S.l.: s.n.], 2015. p. 99–1.

FAN, Q. et al. Jumpcut: non-successive mask transfer and interpolation for video cutout. **ACM Trans. Graph.**, v. 34, n. 6, p. 195–1, 2015.

FORD, L. R.; FULKERSON, D. R. Maximal flow through a network. **Canadian journal of Mathematics**, v. 8, n. 3, p. 399–404, 1956.

GASTAL, E. S.; OLIVEIRA, M. M. Shared sampling for real-time alpha matting. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2010. v. 29, n. 2, p. 575–584.

GOLDBERG, A. V.; TARJAN, R. E. A new approach to the maximum-flow problem. **Journal of the ACM (JACM)**, ACM, v. 35, n. 4, p. 921–940, 1988.

GRADY, L. Random walks for image segmentation. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 28, n. 11, p. 1768–1783, 2006.

GREIG, D. M.; PORTEOUS, B. T.; SEHEULT, A. H. Exact maximum a posteriori estimation for binary images. **Journal of the Royal Statistical Society. Series B (Methodological)**, JSTOR, p. 271–279, 1989.

GRUNDMANN, M. et al. Efficient hierarchical graph-based video segmentation. In: IEEE. **Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on**. [S.l.], 2010. p. 2141–2148.

JOHNSON, J.; RAJAN, D.; CHOLAKKAL, H. Temporal trimap propagation using motion-assisted shape blending. In: IEEE. **Visual Communications and Image Processing (VCIP), 2015**. [S.l.], 2015. p. 1–4.

JOSHI, N.; MATUSIK, W.; AVIDAN, S. Natural video matting using camera arrays. In: ACM. **ACM Transactions on Graphics (TOG)**. [S.l.], 2006. v. 25, n. 3, p. 779–786.

LEVIN, A.; LISCHINSKI, D.; WEISS, Y. A closed-form solution to natural image matting. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 30, n. 2, p. 228–242, 2008.

LI, Y.; SUN, J.; SHUM, H.-Y. Video object cut and paste. In: ACM. **ACM Transactions on Graphics (ToG)**. [S.l.], 2005. v. 24, n. 3, p. 595–600.

LUCAS, B. D.; KANADE, T. et al. An iterative image registration technique with an application to stereo vision. Vancouver, BC, Canada, 1981.

MAERKI, N. et al. Bilateral space video segmentation. In: **The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2016.

MCGUIRE, M. et al. Defocus video matting. In: ACM. **ACM Transactions on Graphics (ToG)**. [S.l.], 2005. v. 24, n. 3, p. 567–576.

MORTENSEN, E. N.; BARRETT, W. A. Intelligent scissors for image composition. In: ACM. **Proceedings of the 22nd annual conference on Computer graphics and interactive techniques**. [S.l.], 1995. p. 191–198.

ORLIN, J. B. Max flows in o (nm) time, or better. In: ACM. **Proceedings of the forty-fifth annual ACM symposium on Theory of computing**. [S.l.], 2013. p. 765–774.

ROY, S.; COX, I. J. A maximum-flow formulation of the n-camera stereo correspondence problem. In: IEEE. **Computer Vision, 1998. Sixth International Conference on**. [S.l.], 1998. p. 492–499.

SINDEEV, M.; KONUSHIN, A.; ROTHER, C. Alpha-flow for video matting. In: SPRINGER. **Asian Conference on Computer Vision**. [S.l.], 2012. p. 438–452.

SNOW, D.; VIOLA, P.; ZABIH, R. Exact voxel occupancy with graph cuts. In: IEEE. **Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on**. [S.l.], 2000. v. 1, p. 345–352.

WANG, J. et al. Interactive video cutout. In: ACM. **ACM Transactions on Graphics (ToG)**. [S.l.], 2005. v. 24, n. 3, p. 585–594.

WANG, J.; COHEN, M. F. et al. Image and video matting: a survey. **Foundations and Trends® in Computer Graphics and Vision**, Now Publishers, Inc., v. 3, n. 2, p. 97–175, 2008.

WANG, O. et al. Automatic natural video matting with depth. In: IEEE. **Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference On**. [S.l.], 2007. p. 469–472.

XU, N. et al. Deep image matting. **arXiv preprint arXiv:1703.03872**, 2017.

ZHONG, F. et al. Discontinuity-aware video object cutout. **ACM Transactions on Graphics (TOG)**, ACM, v. 31, n. 6, p. 175, 2012.

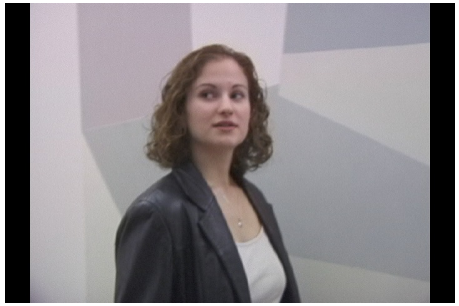
APPENDIX A — RESULT FIGURES

This appendix presents results obtained for tested video sequences. Figures A.1 and A.2 show videos from (CHUANG et al., 2002) and Figures A.3-A.7 (EROFEEV et al., 2015). For each video we show input frames, input trimaps (first and last), output trimaps and the resulting alpha channel for some frames. The full video sequences are available online for download[1]. We strongly recommend the reader to watch the videos for a better view of the results.

[1] <https://inf.ufrgs.br/~mhbackes/video-matting/results.zip>

Figure A.1: *amira* video sequence

(a) *amira* frame 0 - source



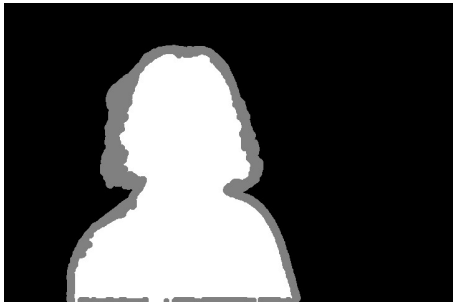
(b) *amira* frame 35 - source

(c) *amira* frame 65 - source

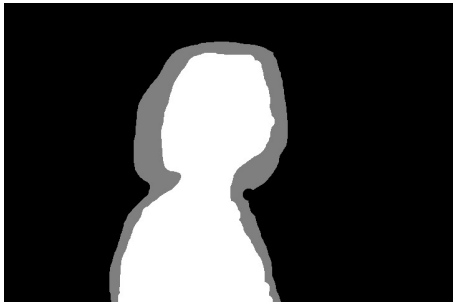
(d) *amira* frame 90 - source

(e) *amira* frame 0 - input trimap

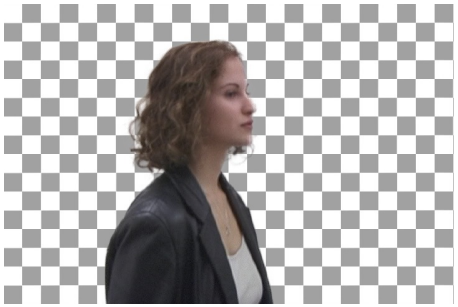
(f) *amira* frame 35 - result trimap



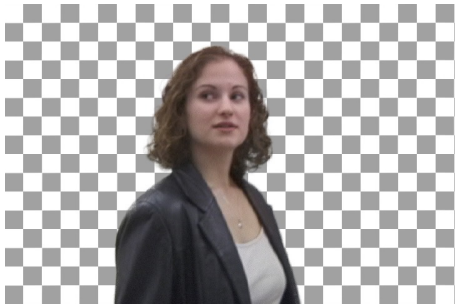
(g) *amira* frame 65 - result trimap



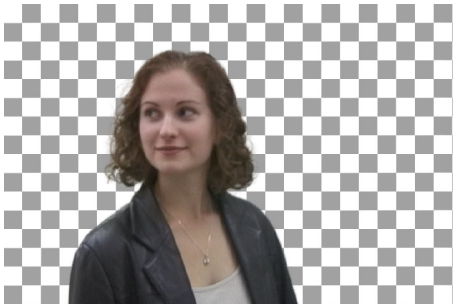
(h) *amira* frame 90 - input trimap



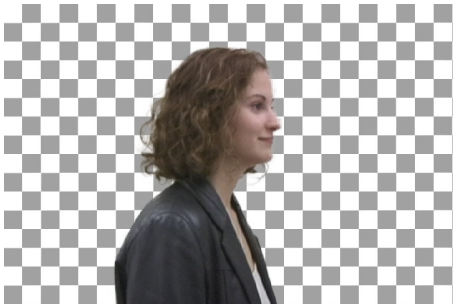
(i) *amira* frame 0 - result alpha



(j) *amira* frame 35 - result alpha



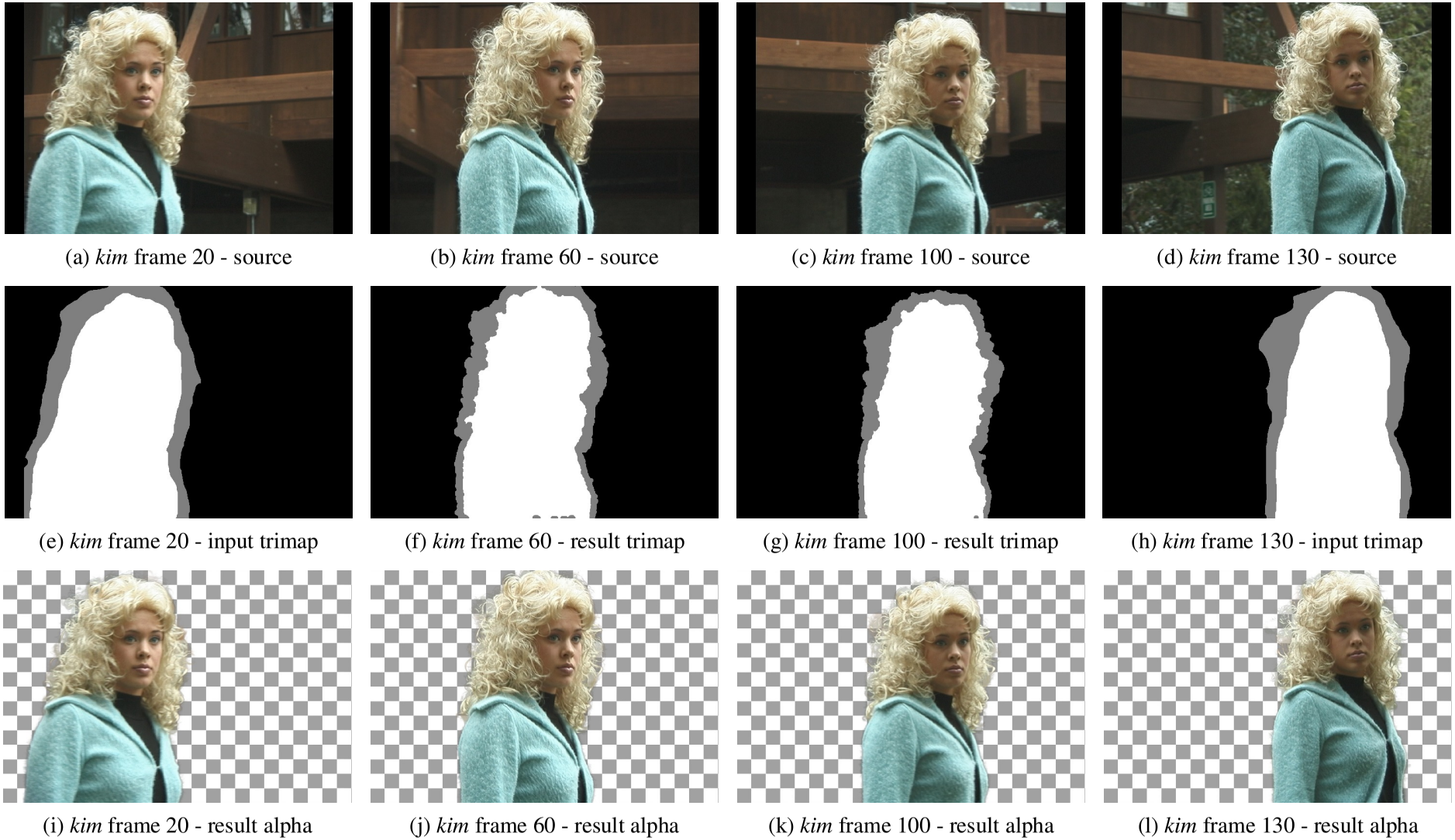
(k) *amira* frame 65 - result alpha



(l) *amira* frame 90 - result alpha

Source: The Authors

Figure A.2: *kim* video sequence



(a) *kim* frame 20 - source
(b) *kim* frame 60 - source
(c) *kim* frame 100 - source
(d) *kim* frame 130 - source
(e) *kim* frame 20 - input trimap
(f) *kim* frame 60 - result trimap
(g) *kim* frame 100 - result trimap
(h) *kim* frame 130 - input trimap
(i) *kim* frame 20 - result alpha
(j) *kim* frame 60 - result alpha
(k) *kim* frame 100 - result alpha
(l) *kim* frame 130 - result alpha

Source: The Authors

Figure A.3: *Alex* video sequence

(a) *Alex* frame 0 - source

(b) *Alex* frame 45 - source

(c) *Alex* frame 105 - source

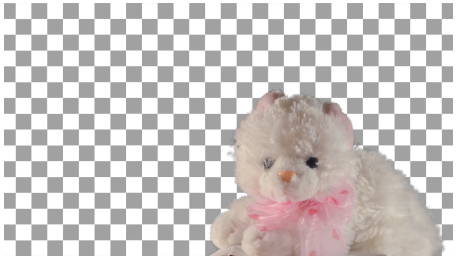
(d) *Alex* frame 149 - source

(e) *Alex* frame 0 - input trimap

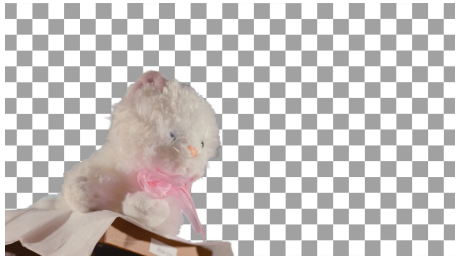
(f) *Alex* frame 45 - result trimap

(g) *Alex* frame 105 - result trimap

(h) *Alex* frame 149 - input trimap

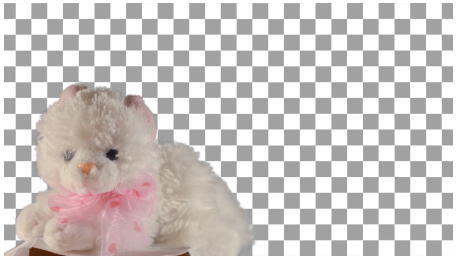


(i) *Alex* frame 0 - result alpha



(j) *Alex* frame 45 - result alpha

(k) *Alex* frame 105 - result alpha



(l) *Alex* frame 149 - result alpha

Source: The Authors

Figure A.4: *Dmitriy* video sequence

(a) *Dmitriy* frame 0 - source

(b) *Dmitriy* frame 48 - source

(c) *Dmitriy* frame 103 - source

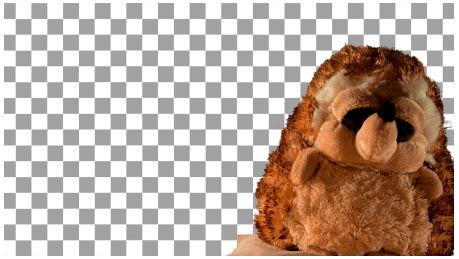
(d) *Dmitriy* frame 149 - source

(e) *Dmitriy* frame 0 - input trimap

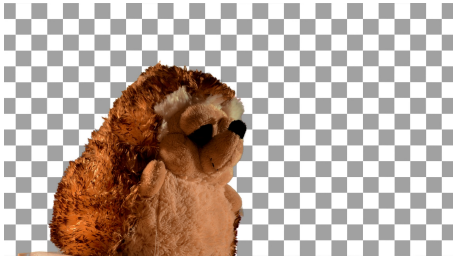
(f) *Dmitriy* frame 48 - result trimap

(g) *Dmitriy* frame 103 - result trimap

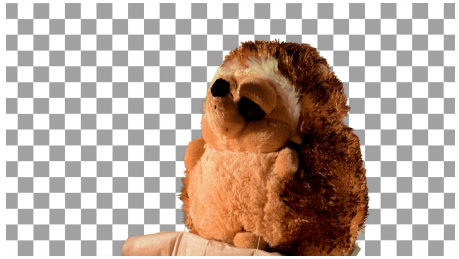
(h) *Dmitriy* frame 149 - input trimap



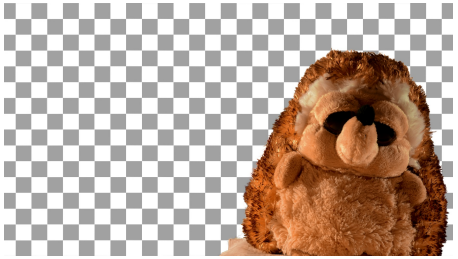
(i) *Dmitriy* frame 0 - result alpha



(j) *Dmitriy* frame 48 - result alpha



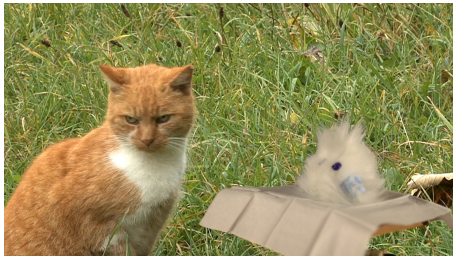
(k) *Dmitriy* frame 103 - result alpha



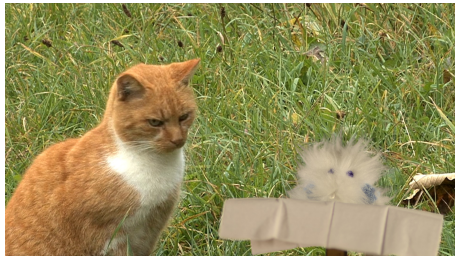
(l) *Dmitriy* frame 149 - result alpha

Source: The Authors

Figure A.5: *Slava* video sequence



(a) *Slava* frame 0 - source



(b) *Slava* frame 48 - source

(c) *Slava* frame 103 - source

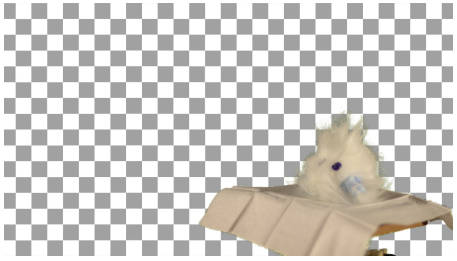
(d) *Slava* frame 149 - source

(e) *Slava* frame 0 - input trimap

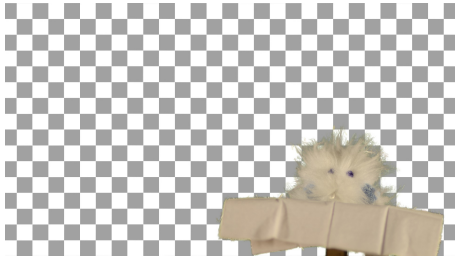
(f) *Slava* frame 48 - result trimap

(g) *Slava* frame 103 - result trimap

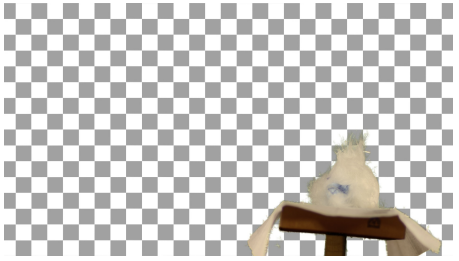
(h) *Slava* frame 149 - input trimap



(i) *Slava* frame 0 - result alpha



(j) *Slava* frame 48 - result alpha



(k) *Slava* frame 103 - result alpha

(l) *Slava* frame 149 - result alpha

Source: The Authors

Figure A.6: *snow* video sequence

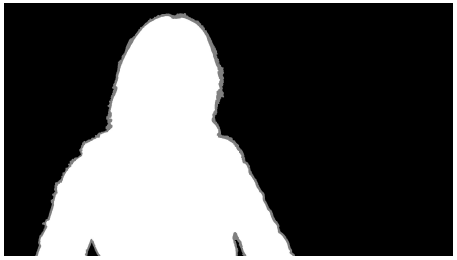
(a) *snow* frame 0 - source

(b) *snow* frame 48 - source

(c) *snow* frame 103 - source

(d) *snow* frame 159 - source

(e) *snow* frame 0 - input trimap



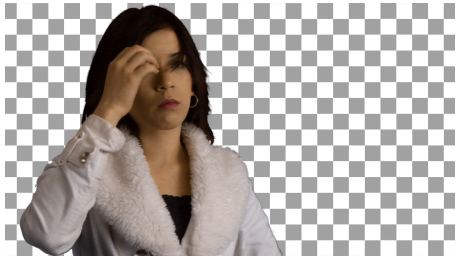
(f) *snow* frame 48 - result trimap

(g) *snow* frame 103 - result trimap

(h) *snow* frame 159 - input trimap

(i) *snow* frame 0 - result alpha

(j) *snow* frame 48 - result alpha



(k) *snow* frame 103 - result alpha

(l) *snow* frame 159 - result alpha

Source: The Authors

Figure A.7: *Vitaliy* video sequence

(a) *Vitaliy* frame 0 - source

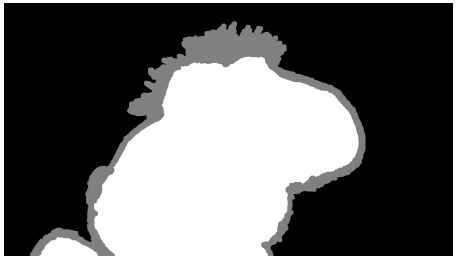
(b) *Vitaliy* frame 48 - source

(c) *Vitaliy* frame 103 - source

(d) *Vitaliy* frame 148 - source

(e) *Vitaliy* frame 0 - input trimap

(f) *Vitaliy* frame 48 - result trimap



(g) *Vitaliy* frame 103 - result trimap

(h) *Vitaliy* frame 148 - input trimap

(i) *Vitaliy* frame 0 - result alpha

(j) *Vitaliy* frame 48 - result alpha

(k) *Vitaliy* frame 103 - result alpha

(l) *Vitaliy* frame 148 - result alpha

Source: The Authors