UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MARCOS HENRIQUE BACKES

A PatchMatch-based Approach for Matte Propagation in Videos

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Manuel Menezes de Oliveira Neto

Porto Alegre
December 2019

CIP — CATALOGING-IN-PUBLICATION

Backes, Marcos Henrique

A PatchMatch-based Approach for
Matte Propagation in Videos / Marcos Henrique Backes. – Porto Alegre: PPGC da UFRGS, 2019.

69 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2019. Advisor: Manuel Menezes de Oliveira Neto.

1. Image/Video Editing. 2. Alpha Matting. 3. Video Matting. 4. Video Compositing. I. Oliveira Neto, Manuel Menezes de. II. Título.

“Beware of bugs in the above code;
I have only proved it correct, not tried it.“

— Donald E. Knuth

ACKNOWLEDGEMENTS

First, I would like to thank my advisor Manuel for pushing me to my limit in order get the best out of me. I'm also thankful to all professors and staff of PPGC and UFRGS, for all the lessons learned not only in computer science, but also for life.

I would like to acknowledge, with gratitude, the support and love of my family – my parents Lucia and Paulo, my brother Tiago, and my canine friends Totó and Buddy. Besides of giving me strength and motivation for pursuing this degree, their teachings and experiences lived together are the result of what I am today. Also, a special thanks to all my relatives and dearest friends. The happiness they give me is what makes life beautiful and worth living.

ABSTRACT

Despite considerable advances in natural image matting over the last decades, video matting still remains a difficult problem. The main challenges faced by existing methods are the large amount of user input required, and temporal inconsistencies in mattes between pairs of adjacent frames. We present a temporally-coherent matte-propagation method for videos based on PatchMatch and edge-aware filtering. Given an input video and trimaps for a few frames, including the first and last, our approach generates alpha mattes for all frames of the video sequence. We also present a user scribble-based interface for video matting that takes advantage of the efficiency of our method to interactively refine the matte results. We demonstrate the effectiveness of our approach by using it to generate temporally-coherent mattes for several natural video sequences. We perform quantitative comparisons against the state-of-the-art sparse-input video matting techniques and show that our method produces significantly better results according to three different metrics. We also perform qualitative comparisons against the state-of-the-art dense-input video matting techniques and show that our approach produces similar quality results while requiring only about 7% of the amount of user input required by such techniques. These results show that our method is both effective and user-friendly, outperforming state-of-the-art solutions.

Keywords: Image/Video Editing. Alpha Matting. Video Matting. Video Compositing.

Uma abordagem baseada em PatchMatch para a Propagação de Alpha Matte em Videos

RESUMO

Apesar de avanços consideráveis em *alpha matting* de imagens naturais nas últimas décadas, *video matting* ainda continua sendo um problema difícil. Os desafios principais encarados pelos métodos existentes são a grande quantidade de interação do usuário requerida e inconsistências temporais em *mattes* entre pares de quadros adjacentes. Nesse trabalho, é apresentado um método temporalmente coerente de propagação do canal alfa em vídeos baseado em PatchMatch e filtros de suavização com preservação de arestas. Dado um vídeo de entrada e *trimaps* para alguns frames, incluindo o primeiro e o último, nossa abordagem gera *alpha mattes* para todos os quadros da sequência de vídeo. É apresentado, também, uma interface baseada em rabiscos para *video matting* que se aproveita da eficiência do nosso método para interativamente refinar os resultados. É demonstrada a eficácia da nossa abordagem usando-a para gerar *mattes* temporalmente coerentes para várias sequências de vídeo. Foi realizada uma comparação quantitativa com o estado da arte em técnicas de *video matting* que mostra que nosso método produz resultados significativamente melhores de acordo com três métricas diferentes. Além disso, foi realizada uma análise qualitativa contra técnicas de *video matting* que requerem um *trimap* por quadro que mostra que nossa técnica produz resultados similares usando apenas 7% da quantidade de interação do usuário requerida pelas outras técnicas. Esses resultados mostram que a técnica proposta é efetiva e fácil de usar superando o estado da arte.

Palavras-chave: Edição de Imagens/Vídeos, Alpha Matting, Video Matting, Composição de Vídeos.

LIST OF ABBREVIATIONS AND ACRONYMS

AE	Adobe After Effects
ANNF	Approximate Nearest-Neighbor Field
CCL	Connected Components Labeling
CF	Closed Form Matting
CPU	Central Processing Unit
DM	Deep Matting
GPU	Graphics Processing Unit
HD	High Definition
KNN	K-Nearest-Neighbor
LAB	Lightness-A-B color model
LSH	Locality Sensitive Hashing
MAKNN	Motion-Aware KNN Matting
NNF	Nearest-Neighbor Field
RGB	Red-Green-Blue color model
SAM	Self-Adaptive Matting
SIFT	Scale-Invariant Feature Transform
SLR	Sparse Low-Rank Mattting
SM	Shared Matting

LIST OF FIGURES

Figure 1.1 Given an input video sequence (top) and user-defined trimaps for the first and last frames, our method is able to efficiently interpolate a temporally coherent alpha matte (bottom) for the video. 14

Figure 3.1 A pair of frames (a) and (b). Optical-flow (c) computed from (a) to (b), for every pixel of (a). The colors in (c) represent the motion vector associated with each pixel, such that hue represents the direction and saturation the length of the vector as shown in (d). Black pixels in (c) represent occlusions. 22

Figure 3.2 Example of NNF mapping. Patches in A are mapped to its most similar patch in B. Note that this example only shows a few maps in the NNF from A to B. The maps are computed for every possible patch in A. 24

Figure 3.3 Illustration of the steps performed by the PatchMatch algorithm. (a) All patches are initialized with a random offset. (b) The blue patch checks the solution of its local neighbors to try to improve its solution. In this case, matching next to the red patch solution reduces the cost of the match. (c) The blue patch searches randomly for solutions around its current best match. 26

Figure 3.4 Example of 1D edge-aware filtering using the Domain Transform. Input signal I (a), x-coordinates $ct(u)$ computed for I (b), transformed signal I using $ct(u)$ as x-coordinates (c), transformed signal filtered with a Gaussian filter (d) and filtered signal mapped to the original domain. 28

Figure 3.5 Example of colorization using the Domain Transform. Input image (a), input scribbles (b), scribble mask (c) obtained from (b). Applying the recursive filter to (b) and (c) results in (d) and (e). Performing elementwise division of (d) by (e) results in (f) (normalization). Finally, the colorized result (g) is obtained combining the lightness of (a) with the chrominance of (f). 30

Figure 3.6 Source frame t (a). A sparse set of feature matches between frames t and $t+1$ is used to estimate the initial solution (b). (b) is spatially and temporally filtered using the Domain Transform to produce (c). 31

Figure 3.7 Temporal filter iteration. The current motion vector computed for each pixel in an iteration i is used to estimate the trajectory path of a pixel (left). These paths can be viewed as 1D signals (right). The temporal filtering is the result of applying the Domain Transform filter on these 1D signals. 31

Figure 4.1 Visual comparison of matte propagation for source frame (a) using optical-flow (c) and our version of PatchMatch (d). Ground truth matte of (a) is shown in (b). In this example, both (c) and (d) were obtained propagating one frame, starting from a ground truth matte on a key frame, using our propagation technique (Section 4.2), with different optical-flow initializations. 34

Figure 4.2 Visual comparison of propagation with and without the refinement step. ...37

Figure 4.3 Visual comparison of propagation with and without the removal of disconnected components step. Keyframe (a) and its precomputed matte (c). In the next frame (b), a region with colors similar to the foreground is revealed. Optical-flow errors cause the matte to be incorrectly propagated as foreground (d). Result after removing the false foreground components (Section 4.4) (e). 39

Figure 5.1 Our interactive video matting system interface. Scribbles on the keyframes indicate the foreground (white), background (black), and unknown (gray) regions. The extracted foreground object is instantly updated on the right window. Our system then propagates the extracted mattes for the unconstrained frames. Users can inspect the matte of any frame and interactively refine it with additional scribbles. The resulting changes are propagated forward and backwards to other frames. Please refer to the video illustrating the use of our system, in the supplementary material. 42

Figure 5.2 The main components of our video-matting editing interface: *Tools*, *Scribble Panel*, *Result Panel*, and *Video Frame Selection* 42

Figure 5.3 Overview of our interactive video-matting interface. 43

Figure 6.1 Representative frames from three videos with ground truth [Erofeev et al. 2015] used for quantitative comparisons involving sparse-input video matting techniques. 47

Figure 6.2 Comparison of the matte propagation methods under the SSDA error metric. Smaller values are better. AE - *After Effects Rotobrush Tool* [Adobe Inc. 2019, Bai et al. 2009], MAKNN - *Motion-aware KNN Matting* [Li, Chen and Tang 2013], SLR - *Sparse Low-Rank Representation Ratting* [Zou et al. 2019], OURS+CF - Our method using *Closed-form Matting* [Levin, Lischinski and Weiss 2008] initialization and OURS+SM - Our method using *Shared Matting* [Gastal and Oliveira 2010] initialization. 48

Figure 6.3 Comparison of the matte propagation methods under the dtSSD error metric. Smaller values are better. AE - *After Effects Rotobrush Tool* [Adobe Inc. 2019, Bai et al. 2009], MAKNN - *Motion-aware KNN Matting* [Li, Chen and Tang 2013], SLR - *Sparse Low-Rank Representation Ratting* [Zou et al. 2019], OURS+CF - Our method using *Closed-form Matting* [Levin, Lischinski and Weiss 2008] initialization and OURS+SM - Our method using *Shared Matting* [Gastal and Oliveira 2010] initialization. 49

Figure 6.4 Comparison of the matte propagation methods under the MESSDdt error metric. Smaller values are better. AE - *After Effects Rotobrush Tool* [Adobe Inc. 2019, Bai et al. 2009], MAKNN - *Motion-aware KNN Matting* [Li, Chen and Tang 2013], SLR - *Sparse Low-Rank Representation Ratting* [Zou et al. 2019], OURS+CF - Our method using *Closed-form Matting* [Levin, Lischinski and Weiss 2008] initialization and OURS+SM - Our method using *Shared Matting* [Gastal and Oliveira 2010] initialization. 50

Figure 6.5 First frame of videos from *the video matting benchmark* [Erofeev et al. 2015] used in the qualitative comparison. 52

Figure 6.6 Qualitative comparison of results produced by techniques that require one trimap per frame against results produced by our method. DM - *Deep Matting* [Xu et al. 2017], SAM - *Self-Adaptive Matting* [Cao et al. 2019], LB - *Learning Based Matting* [Zheng and Kambhamettu 2009] and SpSM - *Sparse Sampling Matting* [Karacan, Erdem and Erdem 2017]. OURS+DM, OURS+SAM, OURS+LB and OURS+SpSM stand for our method initialized by these respective matting methods every 15 frames. 53

Figure 6.7 Comparison of temporal coherence between techniques. In these examples, *Deep Matting* (DM) [Xu et al. 2017] and *Sparse Sampling Matting* (SpSM) [Karacan, Erdem and Erdem 2017] present sudden changes in the alpha channel between consecutive frames, which results in temporal jittering. Our technique is able to generate more temporally-coherent results when using these same techniques to initialize one out of fifteen keyframes. Please refer to the supplementary material for video results. 55
Figure 6.8 Frames from videos showing foreground elements partially occluding others (first and third rows). Corresponding extract mattes for the boy (second row), and for one horse and policeman (fourth row). These were obtained by propagating mattes of adjacent frames extracted using our interactive matting interface. 56
Figure 6.9 Due to fast motions and foreground/background color ambiguities, using evenly distributed trimaps our technique could not avoid these undesired artifacts in the *rain* video sequence. Such artifacts can be avoided by repositioning the keyframes, or adding new ones. 58

Figure B.1 An example of parallel reduction iteration using the sum operator. Sum operations are grouped in pairs which reduces in half the number of elements for the next iteration. 68

LIST OF TABLES

Table 5.1 Runtime of each step of our method using an NVIDIA GTX 1070 graphics card and video with 1920x1080 resolution. 44

Table 6.1 Mean error metrics computed for the three video sequences using nine keyframes. AE - Adobe After Effects Rotobrush Tool, MAKNN - Motion-aware KNN matting, SLR - Sparse Low-Rank matting, OURS+CF - Ours with Closed-form Matting, and OURS+SM - Ours with Shared Matting. Please refer to text for details. 51

CONTENTS

1 INTRODUCTION **13**
1.1 Thesis Statement **14**
2 RELATED WORKS **16**
2.1 Alpha matting **16**
2.2 Video matting **16**
2.2.1 Dense-input video matting 16
2.2.2 Sparse-input video matting 17
2.2.3 Domain Transform Filters 19
2.3 Summary **19**
3 BACKGROUND **21**
3.1 Optical-flow **21**
3.1.1 PatchMatch 23
3.1.1.1 Approximate nearest-neighbor field algorithm 25
3.1.1.2 GPU Implementation 27
3.2 Domain Transform **27**
3.2.1 Domain Transform Application: Image colorization 29
3.2.2 Practical Temporal Consistency using the Domain Transform 30
3.3 Summary **31**
4 MATTE PROPAGATION **33**
4.1 Computing Forward and Backward optical-flow **33**
4.2 Propagation **35**
4.3 Refining the Propagated Matte **37**
4.4 Discarding False Foreground Components **38**
4.5 Summary **40**
5 INTERACTIVE VIDEO MATTING **41**
5.1 Interface **41**
5.1.1 Preprocessing 41
5.1.2 Defining Initial Keyframes 43
5.1.3 Matte Propagation 43
5.1.4 Refining Propagated Keyframes 44
5.2 Performance **44**
5.3 Summary **45**
6 RESULTS **46**
6.1 Quantitative Evaluation **46**
6.2 Qualitative Evaluation **51**
6.3 Videos with Complex Occlusion Patterns **54**
6.4 Limitations **57**
6.5 Summary **57**
7 CONCLUSION **59**
REFERENCES **60**
APPENDIX A — SUPPLEMENTARY MATERIAL **65**
APPENDIX B — DETECTING THE LARGEST CONNECTED COMPONENT IN GPU **66**
B.1 Finding Connected Components **66**
B.2 Parallel Reduction **67**

1 INTRODUCTION

Object detection, extraction, and compositing are important tasks in image and video processing. Natural image/video matting refers to the process of accurately extracting foreground objects from natural images/video frames based on the compositing equation

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

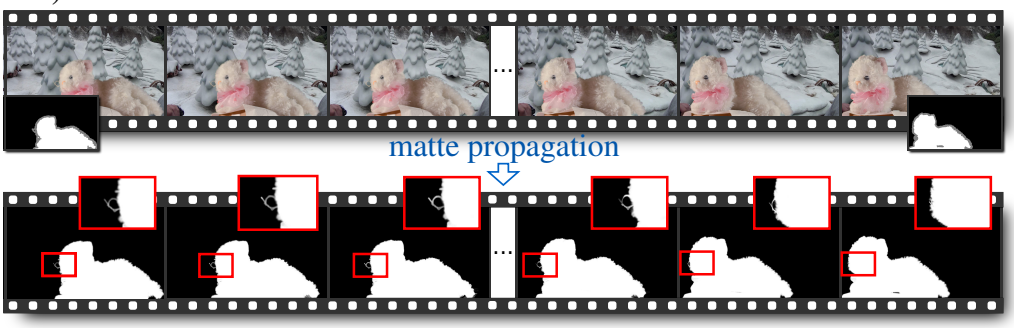
where for any pixel p, its color I_p can be described as a linear combination of a foreground color F_p and a background color B_p, according to some opacity value $\alpha_p \in [0, 1]$.

The goal of a matting algorithm is to determine the foreground and background colors, and the alpha channel for each pixel in the image/frame. Alpha matting is, however, an ill-posed problem, as all variables on the right side of Equation (1.1) are unknown. Thus, matting techniques require additional information, often presented in the form of trimaps or scribbles specifying three sets of pixels belonging, respectively, to the foreground, to the background, and to unknown regions.

Although image matting is a well studied problem and recent works can produce high-quality results [Levin, Lischinski and Weiss 2008, Rhemann et al. 2009, Gastal and Oliveira 2010, He et al. 2011, Chen, Li and Tang 2013, Aksoy, Aydin and Pollefeys 2017, Xu et al. 2017], video matting still presents several challenges. As in most video applications, there are difficulties associated with fast motions, lighting changes, occlusion and disocclusion, and processing of large amounts of data. In addition, video-matting algorithms have two special requirements: they are expected to operate under sparse user input, and achieve temporal coherence [Johnson, Cholakkal and Rajan 2017].

Video matting involves processing a large amount of data and most existing techniques use one trimap per frame. To reduce the user burden, some techniques generate the required trimaps [Wang et al. 2005, Bai, Wang and Simons 2011, Johnson, Rajan and Cholakkal 2015]; others use a sparse set of trimaps [Li, Chen and Tang 2013, Zou et al. 2019]. Although these methods can reduce the amount of user-provided input, they are not sufficiently fast for interactive use. The ability to interactively compute and refine mattes considerably reduces the amount of time involved in video matting tasks.

Figure 1.1: Given an input video sequence (top) and user-defined trimaps for the first and last frames, our method is able to efficiently interpolate a temporally coherent alpha matte (bottom) for the video.



Source: The authors and Erofeev et al. (2015).

1.1 Thesis Statement

We propose to perform interactive, semi-automatic video matting by propagating mattes obtained at a few keyframes. The central idea of this research can be summarized as:

It is possible to achieve interactive video matting by propagating mattes obtained at a few video keyframes. This can be done by computing the optical flow for the input video and using such flow to propagate the opacity values along the temporal domain in a way analogous to color propagation performed by scribble-based methods for image colorization performed in the spatial domain.

We demonstrate this thesis by presenting an efficient temporally-coherent matte-propagation method for videos. Our technique uses a sparse set of trimaps, requiring a relatively small amount of user input. We exploit the parallelism of modern GPUs and the use of linear-time edge-aware filters [Gastal and Oliveira 2011] to process high-resolution videos (*e.g.*, full HD or higher) in just a few milliseconds per frame, allowing for interactive editing and propagation of the computed mattes on-the-fly. Such interactivity improves productivity and the quality of the extracted mattes. Figure 1.1 illustrates the use of our technique to extract mattes for a video sequence.

In summary, our video matting technique begins with the user providing trimaps for, usually, the first and last frames of a video sequence, but more trimaps can be added if needed. We make use of existing alpha matting methods to obtain accurate mattes in these keyframes. Then we initiate our propagation procedure, which consists of four steps. First, we quickly estimate the optical flow for each frame of the video using PatchMatch [Barnes et al. 2009]. After that, we propagate the mattes across the entire video using the

Domain Transform recursive filter [Gastal and Oliveira 2011]. Finally, we post-process the obtained mattes, removing false foreground objects and, finally, applying a per-frame matte optimization [Gastal and Oliveira 2010, Levin, Lischinski and Weiss 2008].

The **contributions** of our work include:

- An efficient temporally-coherent matte-propagation method for videos (Chapter 4). It takes a video sequence and a sparse set of trimaps and propagates the computed mattes to the entire video sequence. Unlike previous approaches that can only handle a few frames at a time, ours processes an entire video sequence at once, naturally enforcing temporal coherence;
- A system for performing interactive video matting that improves productivity and the quality of the extracted mattes (Chapter 5). The efficiency of our method, which takes just a few milliseconds per frame, allows the users to interactively refine and propagate the computed mattes with instant feedback.

2 RELATED WORKS

2.1 Alpha matting

Alpha matting techniques are usually classified as *sampling-based* and *affinity-based*. Sampling-based methods [Gastal and Oliveira 2010, He et al. 2011, Karacan, Erdem and Erdem 2017] assume that the true foreground and background colors of pixels in the unknown region can be estimated by analyzing nearby foreground and background pixels. Affinity-based methods [Levin, Lischinski and Weiss 2008, Aksoy, Aydin and Pollefeys 2017, Chen, Li and Tang 2013] minimize a cost function defined by similarity metrics such as pixel color and spatial proximity.

Recently, machine-learning methods have been successfully applied to the alpha matting problem. Deep neural networks have been used to automatically generate trimaps [Shen et al. 2016] and perform matting of specific portrait images [Shen et al. 2016]. Xu et al. also used deep learning to extract high-quality mattes for given pairs of images and trimaps [Xu et al. 2017].

2.2 Video matting

Video matting algorithms can be classified according to the amount of required user input as *dense-input* and *sparse-input*. The first group consists of techniques requiring per-frame user input, while the second only requires input for a few frames of the video. Since our approach uses sparse input, we will briefly cover the dense-input techniques, focusing our discussion on the sparse-input ones.

2.2.1 Dense-input video matting

Most video-matting solutions handle the individual video frames independently, requiring a trimap per frame, and often compromising temporal coherence. Even when high-quality image-matting techniques are used, the resulting videos tend to exhibit temporal jittering and inconsistencies across frames [Erofeev et al. 2015]. Some recent video-matting techniques [Shahrian et al. 2014, Karacan, Erdem and Erdem 2017, Cao et al. 2019] are able to find interframe pixel relationships to produce temporally-coherent

mattes. Such techniques typically only consider relations between pixels in up to five frames of distance from the current frame.

2.2.2 Sparse-input video matting

The first work on sparse-input video matting [Chuang et al. 2002] proposed a temporal trimap propagation based on optical flow. A cumulative error map was used to select between forward and backward propagated trimaps. Later works relied on an initial binary segmentation of the foreground objects [Wang et al. 2005, Bai et al. 2009, Bai and Sapiro 2009, Bai, Wang and Simons 2011]. Trimaps were then generated from the segmentations to produce the alpha mattes. Wang et al. [Wang et al. 2005] used a sparse set of user input scribbles to perform graph-cut segmentation over the 3D video volume. A hierarchical representation of the video was used to reduce the number of nodes in the graph, making this solution feasible. Finally, a uniform trimap was generated for extracting the matte. Unfortunately, this hierarchical representation results in temporal flickering on the resulting mask [Sindeev, Konushin and Rother 2013]. Segmentation performance was improved by decreasing the size of the graph using downscaling [Tong, Zhang and Ding 2011] and clustering [Zhang, Tang and Cheng 2015], thus enhancing user interaction and supporting larger video sizes.

Video SnapCut [Bai et al. 2009] uses a set of local classifiers created from overlapping windows placed along the initial frame segmentation border. Each classifier takes into account the colors of foreground and background components in the current window and the shape of the object, which is warped to the next frame using optical flow in order to propagate the mask along the video. An adaptation of the matting Laplacian [Levin, Lischinski and Weiss 2008] which uses the matte of the previous frame as constraints for the current one is used to ensure temporal coherency. Geodesic Matting [Bai and Sapiro 2009] uses an efficient fast marching algorithm to perform the initial segmentation. Trimaps are generated by dilating an unknown region on the object's border as needed. This method, however, does not take into account object movement (*i.e.*, optical flow). Bai et al. [Bai, Wang and Simons 2011] noticed the importance of temporal coherency not only in the matte, but also on the generated trimaps. They use optical flow between user-defined trimaps on keyframes to obtain temporally-coherent trimaps. Then, a level-set technique was used to ensure temporal coherency in the produced alpha mattes. Johnson et al. [Johnson, Rajan and Cholakkal 2015, Johnson et al. 2016] propagate

trimaps to the entire video using optical flow and performing corrections based on the object shape.

More recent methods directly propagate the obtained alpha channel [Tang et al. 2012, Sindeev, Konushin and Rother 2013, Li, Chen and Tang 2013, Zou et al. 2019]. Tang et al. [Tang et al. 2012] propagate the matte to the next frame using a variation of the matting Laplacian [Levin, Lischinski and Weiss 2008] to create temporal constraints based on optical flow. A trimap is then created from a graph-cut segmentation of the mask, and another propagation is performed (involving only pixels in the unknown region) resulting in the final matte. Alpha-Flow [Sindeev, Konushin and Rother 2013] propagates the alpha matte through the video using an iterative method. In the first iteration, optical flow is computed based on pixel colors. It is used then to segment the video into temporal chains to be used as temporal restrictions in the 3D matting Laplacian created for the entire video volume to compute an alpha channel for each frame. Finally, the mattes are refined using the Guided Filter [He, Sun and Tang 2013]. This approach requires solving large inter-frame linear equation systems, and is only feasible for very short low-resolution videos.

Some recent works rely on nonlocal affinities, largely used in image denoising, which have shown to be useful for treating complex textures in the input image. Li et al. [Li, Chen and Tang 2013] extended the ideas of KNN Matting [Chen, Li and Tang 2013] to ensure temporal coherence. Nonlocal affinities are produced by finding the motion-aware K-nearest neighbors for each pixel, when creating the matting Laplacian. Temporal coherence is achieved using the previous frame's alpha values as soft constraints for the current frame. Zou et al. [Zou et al. 2019] use principles of sparse coding to create such affinities. The objective is to produce a sparse dictionary for the video and a representation for each pixel. The dictionary is created using foreground and background pixels only (marked by the user) in such a way that the representation (for translucent pixels) is a combination of the foreground with the background. In the end, non-local affinities are generated from the obtained representation, and temporal coherence is also obtained using the result of the previous frame as a constraint to the current frame on the matting Laplacian.

The techniques discussed in this section either use a frame-by-frame propagation strategy [Bai et al. 2009, Bai, Wang and Simons 2011, Johnson, Rajan and Cholakkal 2015, Tang et al. 2012, Li, Chen and Tang 2013, Zou et al. 2019], or process the video as a whole [Wang et al. 2005, Sindeev, Konushin and Rother 2013]. The first group

can only propagate the matte one way. Thus, whenever an error occurs it is propagated forward, resulting in temporal inconsistencies as the next keyframe is reached. Although, theoretically, by using the entire sequence the second group should be able to overcome this issue, in practice the presented solutions are temporally unstable [Wang et al. 2005] or do not scale to current video resolutions [Sindeev, Konushin and Rother 2013].

In contrast to previous approaches, our technique propagates matte information both ways and can handle entire video sequences, thus producing more temporally-coherent results.

2.2.3 Domain Transform Filters

The Domain Transform is an efficient framework for performing edge-aware filtering [Gastal and Oliveira 2011]. Lang et al. [Lang et al. 2012] use Domain Transform filters to approximate solutions for energy minimization problems composed of a *data* term and a *smoothness* term. The authors show that their method is ideal for processing large amounts of data, producing good results for many video applications where temporal coherence is crucial such as optical flow, disparity estimation, and scribble propagation.

In our work, we adapt the concept presented in [Lang et al. 2012] to efficiently propagate the matte across the input video and ensure temporal coherence. We use the Domain Transform's recursive filter [Gastal and Oliveira 2011] to smoothly propagate the matte information and user edits across the video sequence, allowing users to interactively refine mattes with instant feedback. We also propose a novel method for detecting and removing false foreground components during this propagation step. Finally, we optimize the propagated matte with an adaptation of the Laplacian-based matte optimization [Gastal and Oliveira 2010], using propagated foreground and background colors as the initial confidence values.

2.3 Summary

Video matting techniques can be classified as *dense-input* or *sparse-input*. The first class requires user input on every frame of the video, while the second one requires it only on a sparse set of frames and the missing data is deduced or propagated from the defined keyframes. We focus on sparse-input video matting. Although many video

matting techniques have been proposed over the last decade, they either use a frame-by-frame propagation or do not scale to current video resolutions. This thesis proposes a novel video matting approach, which makes use of the Domain Transform filter [Gastal and Oliveira 2011] and the ideas of the work by Lang et al. [Lang et al. 2012] to efficiently propagate the mattes accross multiple frames of the video.

3 BACKGROUND

In this chapter we review some fundamental concepts of image filtering and video processing, which are key for understanding some ideas and algorithms used in this thesis. Section 3.1 presents the optical-flow problem and its applications in video processing. In Section 3.2 we present the Domain Transform algorithm for real-time edge-aware filtering.

3.1 Optical-flow

Optical-flow consists of the apparent motion of objects, surfaces, and edges perceived by a human brain in a scene or environment [Szeliski]. In computer vision, the notion of optical-flow becomes important when processing a video volume, especially in scenes with high object movement and background displacement. Hence, it is one of the most important applications in video processing. Also, many applications such as video summarization, video stabilization and video compression make use of optical-flow algorithms in order to find correlations between pixels in different frames.

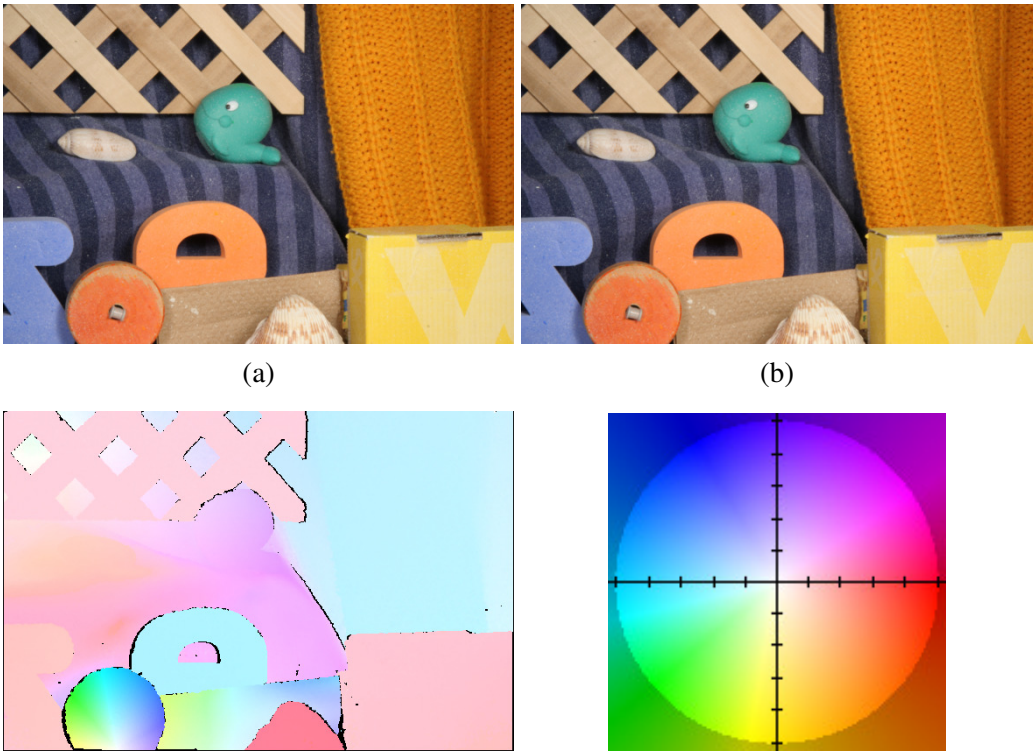
Formally, the dense optical-flow problem stands for, given a pair of frames in a video I spaced apart by Δt units of time, finding for each pixel $I^t_{x,y}$ with coordinates (x,y) in frame t a horizontal and a vertical motion vectors Δx and Δy, respectively, such that the brightness constancy (or color constancy in case of colored videos) is preserved

$$I^t_{x,y} \approx I^{t+\Delta t}_{x+\Delta x, y+\Delta y}. \tag{3.1}$$

Figure 3.1 shows an example of computation of optical-flow.

Optical-flow is a well studied problem. There exist, in the literature, a large amount of solutions and variations for estimating video motion, as well as different evaluation benchmarks [Baker et al. 2011, Geiger, Lenz and Urtasun 2012, Butler et al. 2012]. Early works on optical-flow algorithms [Lucas and Kanade 1981, Horn and Schunck 1981] model the problem as the minimization of an energy function based on equation 3.1. The work by Sun, Roth and Black (2010) presented an experimental study to find out which heuristics, energy functions, and post processing methods can be used to improve the accuracy of the computed flow. Recently, many works have shown satisfactory results by using deep convolutional neural networks to solve this problem [Ilg et al. 2016, Hui, Tang

Figure 3.1: A pair of frames (a) and (b). Optical-flow (c) computed from (a) to (b), for every pixel of (a). The colors in (c) represent the motion vector associated with each pixel, such that hue represents the direction and saturation the length of the vector as shown in (d). Black pixels in (c) represent occlusions.



(a) (b)

(c) (d)

Source: [Baker et al. 2011]

and Loy 2018].

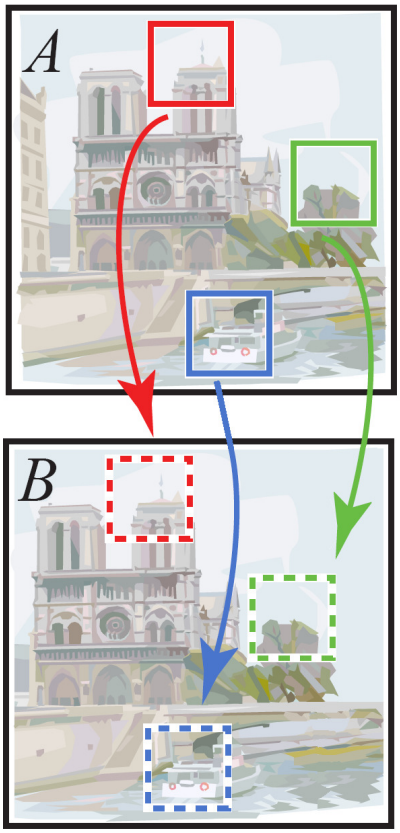
3.1.1 PatchMatch

Many image and video processing applications rely on searching for intensity/color similarities to find correspondences between pixels or regions of an image. Examples of these applications are *image retargeting* [Rubinstein, Shamir and Avidan 2008, Wang et al. 2008], where users are able to rescale an image while maintaining the aspect ratio of some objects; *image completion* [Criminisi, Perez and Toyama 2004, Komodakis 2006], in which users can erase components of an image and the completion tool automatically fills the region with synthesized content which matches the context of the image; and *image reshuffling* [Simakov et al. 2008,Cho, Avidan and Freeman 2010], which allows users to move objects of an image while the computer synthesizes the rest, so that it resembles the original one.

Barnes et al. (2009) noticed that these applications have a common factor: they rely on a problem of searching for the best match between patches of images. In other words, given a pair of RGB images A and B, for every overlapping squared patch of apothem a in A, the goal is to find its match in B under a distance metric d (originally L_2 distance). See Figure 3.2 for a visual example. Such correspondences can be described by a mapping called the *nearest-neighbor field* (NNF). We can define an NNF as a function $f : \mathbb{N}^2 \rightarrow \mathbb{N}^2$ which maps the coordinates of the center pixel in every patch in A to the center pixel in a patch in B. Finding such neighbors would usually require a quadratic number of comparisons, but the work by Barnes et al. (2009) proposed an algorithm called PatchMatch, which efficiently finds an *approximate nearest-neighbor field* (ANNF) solution using a randomized search. This solution is possible due to neighbor patch coherence assumptions found in natural images.

PatchMatch was the first algorithm to efficiently approximate a solution for the task of finding NNFs. More recently, some works proposed more time efficient methods such as replacing the random search using Locality Sensitive Hashing (LSH) [Korman and Avidan 2016, Datar et al. 2004]. Different from other hashing schemes, this kind of hashing produces collisions when patches are similar. To compute the nearest neighbor, patches are placed into bins according to this hash function. As a result, patches in the same bin are good candidates for a match on the search step. The work by He and Sun (2012) stores patches in a KD-Tree. Good candidates for the NNF are found by searching

Figure 3.2: Example of NNF mapping. Patches in A are mapped to its most similar patch in B. Note that this example only shows a few maps in the NNF from A to B. The maps are computed for every possible patch in A.



Source: Barnes et al. (2009).

for similar patches in the data structure. Although these two methods are faster, they only work when using the L_2 distance metric, while the original PatchMatch works with any metric, as long as it respects the neighbor patch coherence assumption.

In the context of optical-flow and motion estimation, many works [Bao, Yang and Jin 2014, Besse et al. 2014, Chen et al. 2019, Fan et al. 2015] use PatchMatch as an initial guess or coarse solution. In this case, one sets $A = I^t$ and $B = I^{t+1}$ to compute the forward optical-flow, or $A = I^{t+1}$ and $B = I^t$ for the backward optical-flow, where I^t is the current frame and I^{t+1} is the next one. We will be referring again to this in Chapter 4.

3.1.1.1 Approximate nearest-neighbor field algorithm

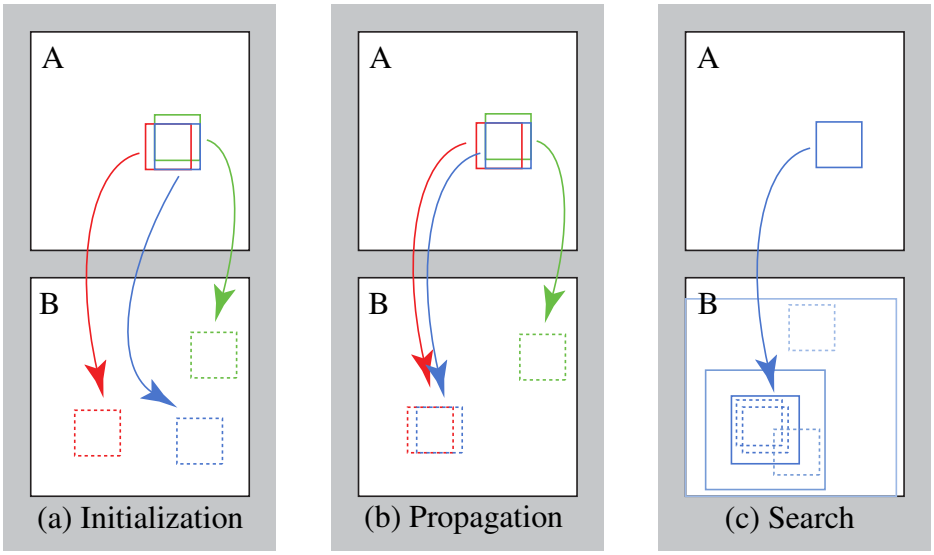
As mentioned before, a brute-force algorithm for computing an NNF would be too slow, since it would be necessary to compare each squared patch of apothem a of image A with every patch in image B, resulting in a $O(n^2a^2)$ algorithm, where n is the number of possible patches in A (assuming A and B have the same size). Instead, Patchmatch [Barnes et al. 2009] algorithm quickly finds an approximate solution (ANNF). This method is a randomized iterative algorithm where each iteration has $O(n\log(n)a^2)$ complexity and, usually, five iterations should produce a good-enough solution.

The Patchmatch algorithm is composed of three main components: random initialization, propagation, and random search. Initially, the nearest-neighbor field is filled with random offsets. Next, an iterative update process is applied to the NNF, where good matches are propagated to adjacent pixels, followed by a random search in the neighborhood of the best offset found so far. These steps are depicted in Figure 3.3.

Random Initialization. Let $f(x,y)$ be the nearest-neighbor field function at the squared patch of width a of A centered at the pixel with coordinates (x,y). It is important to note that f is not injective or, in other words, different patches in A are allowed to map to the same patch in B. We initialize $f(x,y)$ with random offsets uniformly sampled over image B. For each patch in A we also compute the distance (error) metric denoted as $D(x,y)$ which is equal to the patch distance between patch centered at (x,y) of A and patch centered at $f(x,y)$ in B. Since the cost of computing the patch distance between two patches is $O(a^2)$ and we need to compute this cost for each random initialized NNF in A, the resulting complexity for this step is $O(na^2)$.

Iteration. After initializing the NNF, we attempt to improve the current solution using an iterative process. At each iteration we process each patch in A separately, in scan order (from top to bottom, left to right). For each patch we apply two steps called

Figure 3.3: Illustration of the steps performed by the PatchMatch algorithm. (a) All patches are initialized with a random offset. (b) The blue patch checks the solution of its local neighbors to try to improve its solution. In this case, matching next to the red patch solution reduces the cost of the match. (c) The blue patch searches randomly for solutions around its current best match.



Source: Barnes et al. (2009).

propagation and *random search*.

Propagation. In the propagation step, we attempt to propagate good solutions found so far for a patch to its neighbors. In other words, we attempt to improve $f(x,y)$ using the solutions of already processed patches in this iteration $f(x-1,y)$ and $f(x,y-1)$. The main idea is that if $f(x-1,y)$ is a good match for patch $(x-1,y)$, then $f(x-1,y)+(1,0)$ should be a good candidate for (x,y). Therefore, we propagate the matches $f(x-1,y)$ and $f(x,y-1)$ in order to find a better match for $f(x,y)$, resulting in the new value for $f(x,y)$ to be the match of minimum cost between $f(x,y)$, $f(x-1,y)+(1,0)$ and $f(x,y-1)+(0,1)$. This propagation produces the effect that if $f(x,y)$ is a good mapping, it will be propagated to all subsequent pixels to the right and below. In order to be able to propagate to the left and upwards, we use the method mentioned above on *odd* iterations, and on *even* iterations the image is processed in reverse scanline order using candidates $f(x+1,y)$ and $f(x,y+1)$. Finally, the costs of matches $f(x-1,y)+(1,0)$ and $f(x,y-1)+(0,1)$ can be computed in $O(a)$ instead of $O(a^2)$ if we consider that the overlapping with $f(x,y)$ produces redundant terms. Therefore, the resulting complexity of this stage is $O(na)$.

Random Search. In order to be able to escape from local minima, we attempt to

improve $f(x,y)$ by testing a sequence of random candidate offsets $\boldsymbol{u}_i$ at an exponentially decreasing distance from $f(x,y)$.

$$\boldsymbol{u}_i = f(x,y) + w2^{-i}R_i \tag{3.2}$$

where w is the maximum search window (set to max(width, height) of A), R_i is a uniform random coordinate in $[-1,1] \times [-1,1]$. Candidates are generated until $w2^{-i} < 1$ and $f(x,y)$ is updated whenever the cost of a candidate is better than the current solution. Since we use about $log(n)$ candidates for each patch in A and the time cost of evaluating each candidate is $O(a^2)$, the complexity of the random search step is $O(n\log(n)a^2)$.

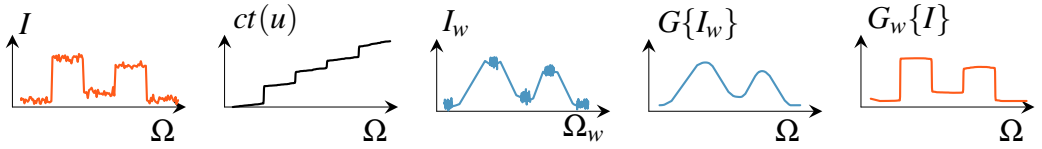
3.1.1.2 GPU Implementation

Parallelizing the Patchmatch algorithm is very straight forward since most of the operations depend only on the patch currently being processed. Therefore, it can be implemented by assigning to each thread a single patch in each of the main steps of the algorithm. One problem to be noticed is that there is a serial dependency between pixels and, thus, the parallelization should not be much effective. To solve this the original Patchmatch implementation uses a propagation scheme similar to *jump flooding* [Barnes et al. 2009, Rong and Tan 2006]. Instead of using only $f(x-1,y)$ and $f(x,y-1)$ as propagation candidates, when using jump flooding, we use candidates in four directions with jump steps of base 2: $f(x \pm 2^i, y)$ and $f(x, y \pm 2^i)$. In practice, Barnes et al. have shown that the results produced are similar to the sequential algorithm when $i \in [0,3]$. This approach eliminates the serial dependency between pixels, Unfortunately, the assumption used in Section 3.1.1.1 is not valid anymore and the cost of the candidates must be computed in $O(a^2)$ resulting in $O(na^2)$ complexity for the propagation on the GPU.

3.2 Domain Transform

The Domain Transform is an efficient framework for performing edge-aware filtering [Gastal and Oliveira 2011]. The fundamental idea of this work is to transform the original signal, according to neighboring pixel intensity similarities, in such a way that the distance between similar pixels is small and different ones become spaced apart. After this transformation, a simple filter (such as a Box or Gaussian filter) is able to perform edge-aware filtering. It is important to notice that the complexity of this approach

Figure 3.4: Example of 1D edge-aware filtering using the Domain Transform. Input signal I (a), x-coordinates $ct(u)$ computed for I (b), transformed signal I using $ct(u)$ as x-coordinates (c), transformed signal filtered with a Gaussian filter (d) and filtered signal mapped to the original domain.



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

Source: Gastal and Oliveira (2011).

is invariant to the sizes of σ_r (color smoothing) and σ_s (spatial smoothing) parameters, resulting in an $O(n)$ filter, where n is the number of pixels in the image. The Domain Transform for a 1D signal I is defined as

$$ct(u) = \int_0^u 1 + \frac{\sigma_r}{\sigma_s} \sum_{k=1}^{c} \left|I_k'(x)\right| dx, \tag{3.3}$$

where I_k is the k-th channel of I. Figure 3.4 shows an example of edge-aware filtering using the Domain Transform.

The Domain Transform cannot be computed for higher dimensional signals. In other words, 2D signals (i.e. images) cannot be mapped by the Domain Transform, since isometric transforms do not exist in bidimentional spaces, in general. In order to perform edge-aware filtering on 2D signals a series of T iterations over 1D signals alternating between horizontal and vertical passes are required.

Gastal and Oliveira (2011) also presented three edge-aware filters: *normalized convolution*, *interpolated convolution* and *recursive filter*. These filters are efficient approximations of the previous edge-aware filters Bilateral Filter [Tomasi and Manduchi 1998], Anisotropic Difusion [Perona and Malik 1990] and Weighted Least Squares [Farbman et al. 2008], respectively. For our work, the most important one is the recursive filter, which is defined with the recursive equation

$$J[n] = (1 - a^d)I[n] + a^d J[n-1], \tag{3.4}$$

where I is the input 1D signal and J is the filtered one. The feedback coefficients a and d are defined as

$$a = e^{-\sqrt{2/\sigma_{H_i}}} \tag{3.5}$$

$$d = ct(x_n) - ct(x_{n-1}) \tag{3.6}$$

where σ_{H_i} varies at every iteration i

$$\sigma_{H_i} = \sigma_s \sqrt{3} \frac{2^{T-i}}{\sqrt{4^T - 1}}. \tag{3.7}$$

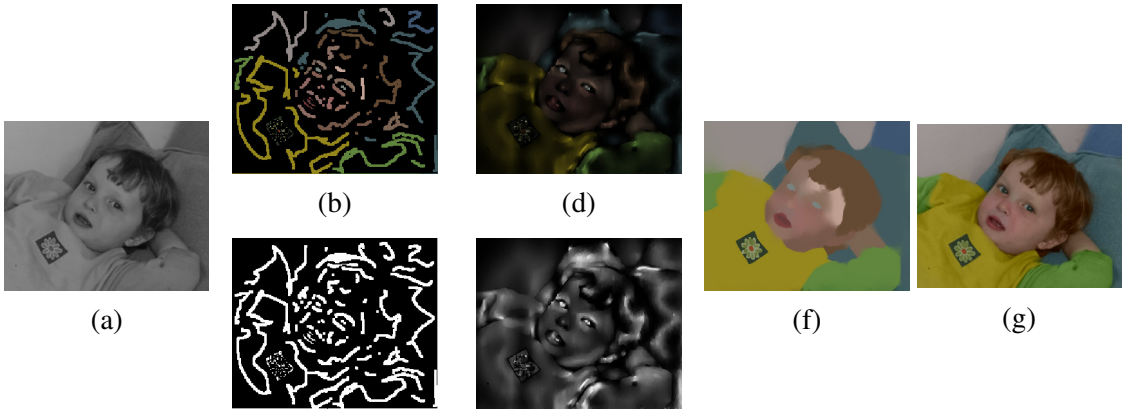
To filter a 1D signal, two passes are necessary: a left-to-right pass and a right-to-left one, since the filter is not symmetric. Similarly, to filter a 2D signal, T iterations consisting of of alternating left-to-right, right-to-left, top-to-bottom, bottom-to-top passes are required.

3.2.1 Domain Transform Application: Image colorization

One application of the Domain Transform is image colorization [Levin, Lischinski and Weiss 2004], which is similar to the alpha matting problem. Given an input grayscale image I (Figure 3.5a) and a set of user colored scribbles, the objective is to generate a colored image C (Figure 3.5g) from a small set of scribbles. We define a new image S (Figure 3.5b) where each pixel is either zero (*color black*) if there is no user input on that pixel or an RGB color correspoding to the same color as the scribble.

An efficient solution is produced using the previously defined recursive filter in Equation (3.4). To do this, we perform the recursive filter to smoothen S using the edge information of I, resulting in S' (Figure 3.5d). In image processing, such operation is called *joint filtering*. As the recursive filter interpolates pixels with scribble data and pixels with no data (zero), resulting in a visually darkened image, it is necessary to normalize the result. To keep track of how much color propagates to every pixel, a normalization channel N (Figure 3.5c) is used. N is defined as 1 where there is user input and 0 otherwise. N is then filtered also using the edge information of I, resulting in N'. Then, S' is normalized by performing an elementwise division of S' by N' (Figure 3.5f). Finally, the colored image is obtained by mixing the lightness information of I with the chrominance of S'/N'. This can be achieved converting these images to a lightness/chrominance color space such as LAB and creating a colored image C using the L channel of I and the A and B channels of S'/N'. This method is a fast approximation of the previous method by Levin, Lischinski and Weiss (2004).

Figure 3.5: Example of colorization using the Domain Transform. Input image (a), input scribbles (b), scribble mask (c) obtained from (b). Applying the recursive filter to (b) and (c) results in (d) and (e). Performing elementwise division of (d) by (e) results in (f) (normalization). Finally, the colorized result (g) is obtained combining the lightness of (a) with the chrominance of (f).



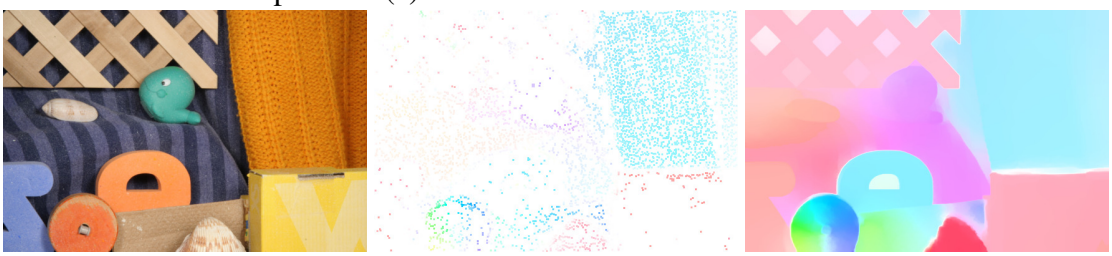
Source: The Authors and Gastal and Oliveira (2011).

3.2.2 Practical Temporal Consistency using the Domain Transform

The work by Lang et al. (2012) presented a fast and scalable solution to the optical-flow problem based on the Domain Transform edge-aware filtering [Gastal and Oliveira 2011]. Previous approaches to this problem were able to achieve accurate results, but only processing one pair or a few frames at a time [Hosni et al. 2012, Zimmer, Bruhn and Weickert 2011, Volz et al. 2011]. Due to the high computational complexity of these techniques, processing more frames at once is unfeasible. Using the Domain Transform to approximate a solution to optical-flow trades accuracy for shorter running times and smaller memory footprint, allowing the processing of multiple frames or even an entire video sequence at once. The main advantage of this strategy is that it supports long temporal relations, producing more temporally stable flow results, at the expense of lower per-frame precision.

To compute the optical-flow for a video, first, the result is initialized with sparse feature correspondences using SIFT [Lowe 2004] and Lucas-Kanade [Lucas and Kanade 1981] features, as depicted in Figure 3.6. The goal is to propagate these sparse data similarly to user defined scribbles in Section 3.2.1, using also a normalization channel. In order to do this, iterative passes of the Domain Transform are performed not only horizontaly and verticaly but also temporaly, as shown in Figure 3.7. A confidence factor is associated to the normalization channel of each pixel to take into account possible

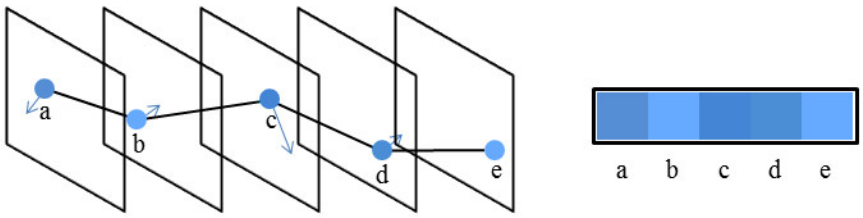
Figure 3.6: Source frame t (a). A sparse set of feature matches between frames t and $t+1$ is used to estimate the initial solution (b). (b) is spatially and temporally filtered using the Domain Transform to produce (c).



(a) (b) (c)

Source: Lang et al. (2012).

Figure 3.7: Temporal filter iteration. The current motion vector computed for each pixel in an iteration i is used to estimate the trajectory path of a pixel (left). These paths can be viewed as 1D signals (right). The temporal filtering is the result of applying the Domain Transform filter on these 1D signals.



Source: The Authors and Lang et al. (2012).

feature mismatches in the initialization and also produce results robust to occlusions.

The generated flow has several applications such as colorization, scribble propagation, depth upsampling, and visual saliency which are computed using the same filtering strategy. Although this work is not appropriate for video matting, since the resulting flow is not accurate along object edges, we use their scribble propagation ideas to efficiently propagate an alpha matte across a video volume.

3.3 Summary

This chapter presented three key concepts which are important for understanding the main ideas and algorithms used to demonstrate this thesis:

- **Optical-flow**: the apparent motion of pixels from one frame of the video to another;
- **PatchMatch**: an efficient algorithm for approximating correlations between pairs of images and is often used to obtain an optical-flow estimate;
- **Domain Transform filter**: an edge-aware filter, with interesting applications such

as missing data interpolation which can be used for image colorization and other video-related applications.

4 MATTE PROPAGATION

Our matte propagation technique for videos has three major steps: (i) Computing both forward and backward optical-flows along the temporal dimension with PatchMatch (Section 4.1); (ii) Using the computed optical-flows to propagate alpha values, foreground, and background colors from keyframes to unconstrained ones, using a temporal version of the domain transform's recursive filter (Section 4.2); and (iii) Refining the computed alpha values to obtain locally-smooth mattes (Section 4.3).

4.1 Computing Forward and Backward optical-flow

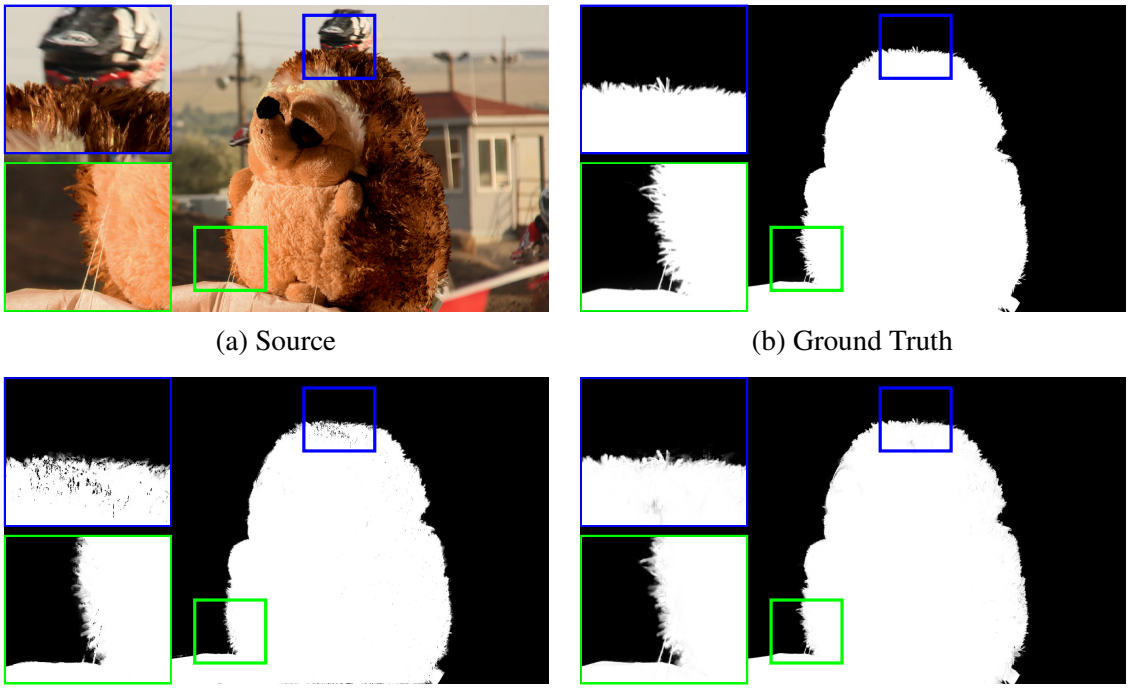
We use PatchMatch [Barnes et al. 2009] (Section 3.1.1) to find correspondences between pairs of pixels across adjacent frames. Given a pair of RGB images A and B, for every overlapping square patch of apothem a in A, PatchMatch looks for its nearest neighbor in B under a distance metric d (originally L_2 distance).

The use of the edge-preserving matching cost function described by Bao et al. [Bao, Yang and Jin 2014] (Equation (4.1)) produces more accurate matching around object borders when compared to traditional optical-flow approaches [Sun, Roth and Black 2010] (see Figure 4.1). According to our experience, it produces better results for our application than all tested alternatives. One should note, however, that the optical-flow obtained with PatchMatch has no sub-pixel accuracy, and that the use of more precise flow around the edges of the foreground objects should lead to more accurate matte propagation.

Equation (4.1) uses a variation of the L_2 metric where the distances between pairs of corresponding pixels in two patches p_A and p_B are weighted by a function ω. Such function takes into account the distance of each pixel to the center of its patch, as well as how similar it is to such central pixel:

$$d(\boldsymbol{a},\boldsymbol{b}) = \frac{1}{W} \sum_{\substack{\Delta(\Delta x,\Delta y): \\ |\Delta x|<=a, \\ |\Delta y|<=a}} \omega(\boldsymbol{a},\boldsymbol{b},\Delta) \|A(\boldsymbol{a}+\Delta) - B(\boldsymbol{b}+\Delta)\|^2, \tag{4.1}$$

Figure 4.1: Visual comparison of matte propagation for source frame (a) using optical-flow (c) and our version of PatchMatch (d). Ground truth matte of (a) is shown in (b). In this example, both (c) and (d) were obtained propagating one frame, starting from a ground truth matte on a key frame, using our propagation technique (Section 4.2), with different optical-flow initializations.



(a) Source (b) Ground Truth

(c) Optical Flow [Sun, Roth and Black 2010] (d) Our PatchMatch

Source: The authors and Erofeev et al. (2015).

$$\omega(\boldsymbol{a},\boldsymbol{b},\Delta) = \exp\left(-\frac{\|A(\boldsymbol{a}+\Delta)-A(\boldsymbol{a})\|^2}{\sigma_r^2}\right) \exp\left(-\frac{\|B(\boldsymbol{b}+\Delta)-B(\boldsymbol{b})\|^2}{\sigma_r^2}\right) \exp\left(-\frac{\|\Delta\|^2}{\sigma_s^2}\right), \tag{4.2}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are the centers of the patches p_A and p_B, respectively, in images A and B, and W is the sum of all weights ω. For all examples shown in this thesis, the apothem of the patch is $a = 3$ (resulting in a squared patch of width 7) and the constants $\sigma_s = 0.5a$ and $\sigma_r = 0.1$. Each patch in A is limited to search for matches in B inside a square region of side 128 pixels centered at its position.

As a preprocessing step, *we compute a forward and backward flow for each frame*, which will be used in the propagation stage. Optical-flow mismatches, usually caused by color ambiguities and fast motions of objects in the video, can cause two types of errors

in the propagated alpha matte: *holes* and *false foreground components*. Holes in the matte cannot be distinguished from actual holes in the foreground elements, but they can be easily fixed with simple user feedback (*e.g.*, using a scribble). Handling false foreground components is discussed in Section 4.4.

4.2 Propagation

The forward and backward optical-flows computed with PatchMatch guide the propagation of alpha values, and of foreground and background colors throughout the unconstrained frames between pairs of keyframes. We use the domain transform recursive filter to propagate these values in linear time with respect to the number of pixels in the video [Gastal and Oliveira 2011, Lang et al. 2012] (Sections 3.2 and 3.2.2).

The alpha values, foreground and background colors for the keyframes are obtained with the use of some matting technique (*e.g.*, [Gastal and Oliveira 2010], [Levin, Lischinski and Weiss 2008], [Xu et al. 2017], [Aksoy, Aydin and Pollefeys 2017]). This makes matte propagation orthogonal to the choice of the matte computation algorithm applied to the keyframes. Since every matting technique has its own strengths and weaknesses, the user can select the one that works best for the type of video at hand.

The data propagated by the recursive filter from a pixel p in frame t corresponds to an 8-dimensional vector D_p^t:

$$D_p^t = [\alpha_p^t, F_p^t, B_p^t, n_p^t], \tag{4.3}$$

where α_p^t is the pixel's opacity value, F_p^t and B_p^t are, respectively, its foreground and background RGB colors, and n_p^t is a normalization factor. For the keyframes, α_p^t, F_p^t, and B_p^t are initialized by the image matting technique, and $n_p^t = 1$. For the unconstrained frames, all these variables are initialized with 0 (zero). In order to propagate the matte data from the keyframes to the unconstrained pixels, we perform a 1-D joint filter of D_p^t using the colors of the temporal neighbor pixels in the input video I along the optical-flow path through I_p^t. The propagated foreground and background colors are also used in the refinement phase (Section 4.3) to evaluate the confidence of the obtained opacity values.

Initially, we intended to apply the recursive filter in three directions of the video volume: horizontal, vertical and temporal, in a similar fashion to the work by Lang et

al. [Lang et al. 2012]. But in our tests, we obtained our best results without the application of horizontal and vertical passes and using a per-frame optimization instead (see Section 4.3). Thus, the propagation step consists only of forward and backward passes of Domain Transform's recursive filter.

During forward propagation, for every pixel p in frame I^t we use the backward optical-flow to find p's temporal neighbor q in the previous frame I^{t-1}. By propagating this way, we ensure that every pixel in the current frame has a neighbor in the previous frame and thus, the matte information will be propagated to all pixels in the current frame. One could also propagate forward using the forward optical-flow, but in this case it is not guaranteed that every pixel in the current frame corresponds to a pixel in the previous one, resulting in "holes" in the propagation. Likewise, during backward propagation, we use the forward optical flow to find p's temporal neighbor r in the next frame I^{t+1}.

Matte and color propagation is then achieved using successive forward and backward recursive propagation steps. Assuming frames varying from 1 to n, the forward propagation is defined by the following recursive equation, for $t \in \{2,3,...,n\}$:

$$J^t = (1-a^d)D^t + a^d J^{t-1}, \tag{4.4}$$

with the initial frame as $J^1 = D^1$. While the backward propagation, for $t \in \{n-1, n-2, ..., 1\}$, is defined as:

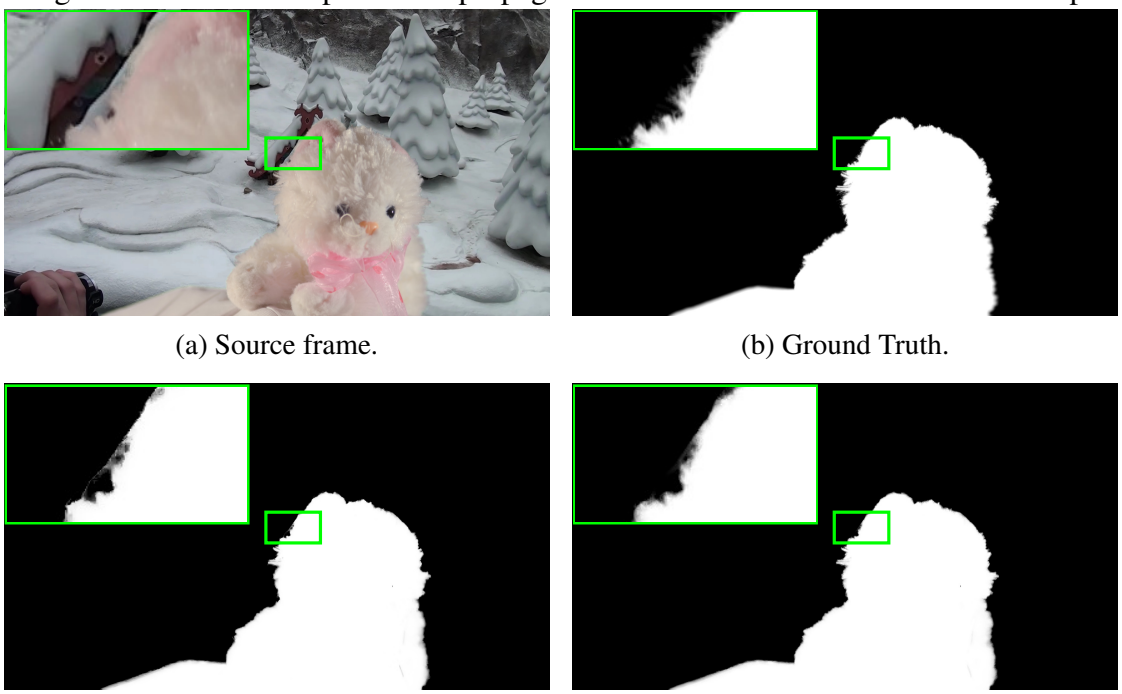
$$J^t = (1-a^d)D^t + a^d J^{t+1}, \tag{4.5}$$

where D^k contains the data to be propagated (the input matte of the iteration), J^k is the filtered signal (the output matte of the iteration), $a \in [0,1]$ is the feedback coefficient computed as $a = \exp\left(-\sqrt{2/\sigma_i}\right)$[1], and d is the geodesic distance between neighbor samples along the temporal 1D paths defined by the optical flow. For forward propagation, $d = 1 + \frac{\sigma_s}{\sigma_r}\|I_p^t - I_q^{t-1}\|_1$. Analogously, for backward propagation, $d = 1 + \frac{\sigma_s}{\sigma_r}\|I_p^t - I_r^{t+1}\|_1$.

The values of the keyframes are not modified by the filtering process since it would only degrade the quality of these keyframe mattes. More precisely, J^t is not updated by Equations (4.4) and (4.5) if t is a keyframe. Instead, it is set as $J^t = D^t$ in every iteration of the propagation. Also, by freezing input mattes, we make the propagation of video sections between keyframes independent of each other, allowing the processing of all

[1]The recursive filter performs three iterations. Each one propagates matte and color data both forward and backwards. For the i-th iteration, the value of $\sigma_i = 2^{3-i}\sqrt{3}/\sqrt{4^3-1}$.

Figure 4.2: Visual comparison of propagation with and without the refinement step.



(a) Source frame. (b) Ground Truth.

(c) Propagated opacity (Section 4.2). (d) Refined opacity (Section 4.3).

Source: The authors and Erofeev et al. (2015).

sections in parallel. Experimentally, we found that the standard deviation values for the spatial and range components of the recursive Domain Transform filter $\sigma_s = 2 \times 10^3$ and $\sigma_r = 0.1$ produce satisfactory results for all tested videos.

After the application of the recursive filter, the values propagated to pixel p of an unconstrained frame t are normalized as:

$$\hat{\alpha}_p^t = \alpha_p^t / n_p^t, \qquad \hat{F}_p^t = F_p^t / n_p^t, \qquad \hat{B}_p^t = B_p^t / n_p^t, \tag{4.6}$$

where $\hat{\alpha}_p^t$, $\hat{F}_p^t$, and $\hat{B}_p^t$ are the normalized attributes for pixel p of frame t, and $D_p^t = [\alpha_p^t, F_p^t, B_p^t, n_p^t]$ is the vector with the propagated data to pixel p at frame t.

4.3 Refining the Propagated Matte

The normalized alpha values $\hat{\alpha}_p^t$ might be noisy. To refine the matte of the propagated frames only, we use the scheme presented by Gastal and Oliveira [Gastal and Oliveira 2010] for optimizing the alpha channel based on the matting Laplacian L [Levin, Lischinski and Weiss 2008] and on the confidence c_p^t of the obtained alpha values, which

is computed as

$$c_p^t = \exp\left(-\delta \| I_p^t - (\hat{\alpha}_p^t \hat{F}_p^t + (1-\hat{\alpha}_p^t)\hat{B}_p^t) \|\right), \tag{4.7}$$

where $\delta = 10$. In this equation, the closer the input color of the pixel is to the composition of foreground and background colors using the alpha matting equation (Equation (1.1)), the higher the confidence. The refined matte is obtained minimizing the following energy function for each frame t:

$$E_\alpha = \alpha^T L \alpha + \lambda(\alpha - \hat{\alpha})^T K(\alpha - \hat{\alpha}) + \gamma(\alpha - \hat{\alpha})^T \Gamma(\alpha - \hat{\alpha}) \tag{4.8}$$

where L is the matting Laplacian, Γ is a diagonal matrix where each diagonal element is defined as c_p^t, K is a diagonal matrix whose elements are 1 if $\hat{\alpha}_p^t$ is either 1 or 0, $\lambda = 100$ and $\gamma = 0.1$. α and $\hat{\alpha}$ are vectorized versions of, respectively, the refined and normalized alpha values for all pixels in frame t. The resulting energy function enforces smoothness of the input matte. The higher the confidence on a pixel, the less likely the pixel is to be changed by the optimization. The optimal solution can be obtained by solving a system of linear equations of size n, where n is the number of pixels in each frame. In our GPU implementation it is solved using the CUSP [CUSP] implementation of the *biconjugate gradient stabilized method*. Figure 4.2 illustrates the use of the refinement step.

Once a refined alpha matte has been obtained for frame t, we also refine the corresponding foreground and background colors. This is achieved by minimizing, over all pixels of each frame t, the chromatic distortion resulting from the refined alpha values α_p^t, the pixel color I_p^t, and the estimated foreground F_p^t, and background B_p^t colors, while enforcing smoothness on matte edges [Levin, Lischinski and Weiss 2008]:

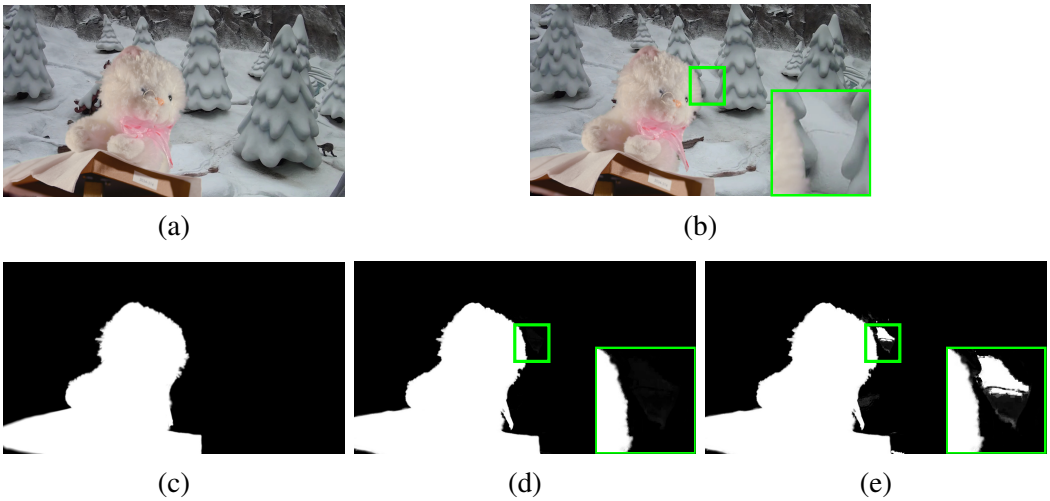
$$\begin{aligned} E_{F,B}^t = & \sum_p \| \alpha_p^t F_p^t + (1-\alpha_p^t) B_p^t - I_p^t \|^2 \\ & + \left|\alpha_{p_x}^t\right| \left(\|F_{p_x}^t\|^2 + \|B_{p_x}^t\|^2 \right) + \left|\alpha_{p_y}^t\right| \left(\|F_{p_y}^t\|^2 + \|B_{p_y}^t\|^2 \right), \end{aligned} \tag{4.9}$$

where $\alpha_{p_x}^t$, $\alpha_{p_y}^t$, $F_{p_x}^t$, $F_{p_y}^t$, $B_{p_x}^t$ and $B_{p_y}^t$ are, respectively, the horizontal and vertical derivatives of α_p^t, F_p^t and B_p^t.

4.4 Discarding False Foreground Components

A video sequence may contain multiple foreground objects, or one foreground object may appear as multiple connected components (*e.g.*, a sequence showing only the

Figure 4.3: Visual comparison of propagation with and without the removal of disconnected components step. Keyframe (a) and its precomputed matte (c). In the next frame (b), a region with colors similar to the foreground is revealed. Optical-flow errors cause the matte to be incorrectly propagated as foreground (d). Result after removing the false foreground components (Section 4.4) (e).



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

Source: The authors and Erofeev et al. (2015).

torso and hands of a character). Occasionally, optical-flow mismatches may lead to incorrect classification of background pixels as foreground ones. This might happen, for instance, when some previously occluded portion of the background becomes visible from behind a foreground object having similar colors. Due to their affinity, this cluster of background pixels is likely to be interpreted as belonging to the foreground object. As these two elements move away from each other and split, the background element will appear as a false foreground component. To minimize the occurrence of such events, users can specify the maximum number of foreground components present in a sequence. In this case, for each frame we use a flood filling strategy to detect connected pixel regions with $\alpha > 0$ and keep at most a user-specified number of the largest ones. The remaining are treated as background pixels (*i.e.*, have their opacity values set to zero and the normalization factor set to one). Note that some of these background pixel clusters may take a few frames to disconnect from the foreground object, when they become detectable. The removal of disconnected components is performed inbetween iterations of the recursive filter propagation (Section 4.2), allowing the propagation of corrections in order to remove errors that take more than one frame to disconnect from the original foreground object. Figure 4.3 illustrates the process of removing false foreground components. In our GPU implementation we use a parallel connected-components algorithm [Šťava and Beneš 2011] and a parallel reduction sum [Sanders and Kandrot 2010] (Appendix B) to

count the number of pixels in the connected components and find the largest ones.

4.5 Summary

This chapter presented a novel approach for the propagation of alpha mattes across the frames of a video. Our technique consists of four major steps:

- **Computing Forward and Backward optical-flow**. We use a version of PatchMatch with a modified cost function by Bao et al. [Bao, Yang and Jin 2014], which produces more accurate results around the objects' borders;
- **Propagation**. The alpha channel, foreground and background colors of the keyframes are propagated along the optical-flow estimated trajectories using the Domain Transform recursive filter;
- **Refining the Propagated Matte**. Two optimization techiques are applied to increase the quality of the propagated mattes: one for optimizing the alpha channel and the other for foreground and background colors. Both are solved using linear systems;
- **Discarding False Foreground Components**. Errors caused by ambiguities in optical-flow estimation are removed by removing smaller foreground components.

5 INTERACTIVE VIDEO MATTING

In this chapter, we present an interface for interactive video matting. It takes advantage of the time efficiency of the GPU implementation of our propagation method for easy and accurate extraction of objects from videos and their compositing onto other sequences. The main concept is that, given an input video, users are able to create/edit keyframes individually with instant feedback and, once they are satisfied with the results they can propagate the obtained mattes to the remaining frames quickly. Hence, a video editing session consists of an iterative process of propagating mattes and refining keyframes. An overview of the interface is shown in Figure 5.1. We refer the reader to a video demonstrating the use of our interactive video matting system, which is available in the supplementary material [1].

5.1 Interface

The main components of the interface are presented in Figure 5.2. The *Tools* panel contains the main controls of the interface, allowing the execution of tasks such as definition of keyframes, running the propagation, and also specifing scribble attributes. User input is performed via scribbles on the *Scribble Panel* and the results are shown instantly in the *Result Panel*. At any time, the user is able to switch between displaying the alpha channel or the final result composited with custom background colors or frames from another video. Finally, the current frame can be selected at the *Video Frame Selection* bar at the bottom. There, the keyframes are highlighted in green.

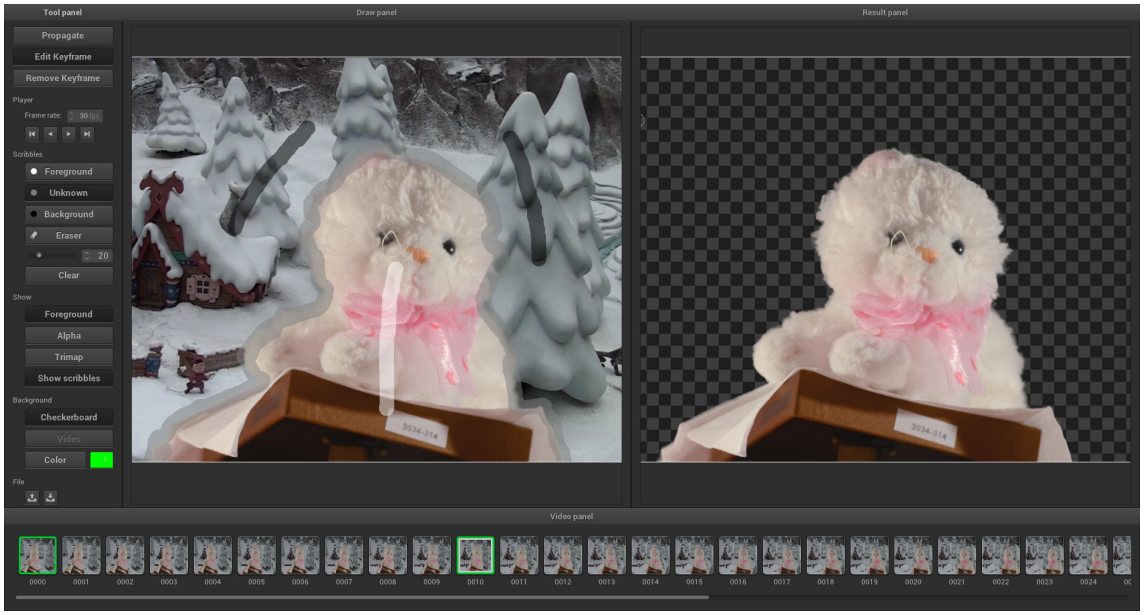
The basic workflow of our interactive interface consists of four major steps: (1) *Preprocessing*, (2) *Defining Initial Keyframes*, (3) *Matte Propagation*, and (4) *Refining Propagated Keyframes* as shown in Figure 5.3.

5.1.1 Preprocessing

Given an input video, we start by preprocessing it by computing the dense forward and backward optical flows for each frame using our PatchMatch approach, as described

[1] The supplementary material is available here: <http://www.inf.ufrgs.br/~oliveira/pubs_files/VM/SM/index.html>

Figure 5.1: Our interactive video matting system interface. Scribbles on the keyframes indicate the foreground (white), background (black), and unknown (gray) regions. The extracted foreground object is instantly updated on the right window. Our system then propagates the extracted mattes for the unconstrained frames. Users can inspect the matte of any frame and interactively refine it with additional scribbles. The resulting changes are propagated forward and backwards to other frames. Please refer to the video illustrating the use of our system, in the supplementary material.

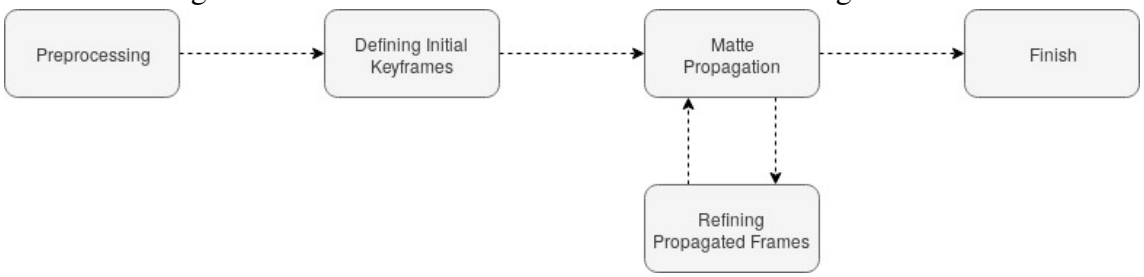


Source: The authors.

Figure 5.2: The main components of our video-matting editing interface: *Tools*, *Scribble Panel*, *Result Panel*, and *Video Frame Selection*

Source: The authors.

Figure 5.3: Overview of our interactive video-matting interface.



Source: The authors.

in Section 4.1. This operation is performed before user interactivity is enabled because it is the most time-consuming (see Section 5.2). Also, since the optical flow does not change as the user creates the mattes, there is no need for updating it.

5.1.2 Defining Initial Keyframes

Once the preprocessing is done and the optical flow is computed, one can begin defining the initial keyframes for the matte propagation. Since our technique propagates mattes both forward and backwards, the user is expected to define keyframes at the first and last video frames. Additional keyframes can be defined in between those.

A simple procedure is used to perform the definition of trimaps on keyframes. First, the user is required to define the unknown region around the borders of the foreground object (shown as grey scribbles). Then, the user can define foreground (white scribble) and background (black scribble) regions with one scribble each, since the regions are filled with a flood-fill algorithm. The alpha, foreground, and background colors are obtained on keyframes using *Shared Matting* [Gastal and Oliveira 2010], as the user draws the trimap. Once the initial trimap is completed, it can be refined with additional scribbles until one is satisfied with the result.

5.1.3 Matte Propagation

Once the initial keyframes are defined, the user can click the propagation button and the matte propagation procedure described in Chapter 4 will begin. Such procedure generates a matte for every intermediate frame not defined as a keyframe. Alternatively, one can also click the optimization button, which will preform the matte optimization (Section 4.3) on each frame.

Table 5.1: Runtime of each step of our method using an NVIDIA GTX 1070 graphics card and video with 1920x1080 resolution.

Step	Runtime (per frame)
Optical-flow (Section 4.1)	1.24s
Propagation (Section 4.2)	67ms
Optimization (Section 4.3)	720ms

5.1.4 Refining Propagated Keyframes

Propagation errors are expected to happen in a video editing session, due to the ambiguities related to the video matting problem. To resolve these ambiguities, the user can define a new trimap in the frames containing errors. In this step we offer two options. The first one is removing the propagated matte from the frame, allowing the user to re-define the trimap from scratch, as presented in Section 5.1.2. The second option is using the propagated alpha channel to create an initial trimap, which can be edited. For this, we simply create an unknown region by dilating regions formed by semi-transparent pixels.

5.2 Performance

The recursive filter in the propagation step can be trivially parallelized by processing different rows and columns of frames, as well as different temporal chains on separate threads. To exploit such parallelism, we implemented the described interface using CUDA/C++ and NanoGUI [NanoGUI 2019]. Considering a 1080p video and an NVIDIA GTX 1070 graphics card, the average running time for each step of our algorithm is shown in Table 5.1. Since the user only has to wait for the propagation step to obtain some visual feedback, our technique provides instant feedback, as opposed to other state-of-art sparse-input video matting methods [Li, Chen and Tang 2013, Zou et al. 2019].

Since GPUs have relatively small memory sizes, only (relatively) short videos can be loaded at once. On a GPU with 8 GB of memory, our system can process about 70 frames at 1080p resolution at once. For long videos, the first and last frames in each video segment loaded into the GPU memory should be keyframes. Each such segment can be safely processed independently from the remaining ones. If a shot change happens in the middle of such a segment, the last frame of the current shot and the first frame of the next one should also be marked as keyframes. In an interactive interface, automatically

splitting video segments as the user draws keyframes and processing them separatedly in the GPU can be implemented in a straightforward way.

5.3 Summary

This chapter described an interface for video matting. Users are able to interactively add and edit keyframes until they are satisfied with the obtained mattes. The main advantage is that modifications in keyframes are propagated in both directions, resulting in temporally coherent mattes. Using a GPU implementation of our method, we obtained the following average runtimes on Nvidia GTX 1070: optical-flow estimation takes about 1.24 s per frame, the propagation step takes about 67 ms per frame, and the optimization takes 720 ms per frame. Since the optical flow is computed offline, a user must wait only a few milliseconds to obtain interactive feedback.

6 RESULTS

To evaluate our method, we perform both quantitative and qualitative evaluations against the state-of-the-art video matting techniques, on various types of videos. More specifically, we perform quantitative evaluations against two techniques that, like ours, do not require the specification of one trimap per frame [Li, Chen and Tang 2013, Zou et al. 2019], as well as against Adobe After Effects Rotobrush Tool (AE). We also perform qualitative comparisons against AE plus top four ranked techniques by the video matting benchmark [Erofeev et al. 2015]: *Deep Matting* [Xu et al. 2017], *Self-Adaptive Matting* [Cao et al. 2019], *Learning Based Matting* [Zheng and Kambhamettu 2009], and *Sparse Sampling Matting* [Karacan, Erdem and Erdem 2017]. These techniques require one trimap per frame, and we show that our method produces similar results even using a much smaller number of keyframes.

6.1 Quantitative Evaluation

For the quantitative evaluation, we used three training videos from the video matting benchmark [Erofeev et al. 2015] for which the ground truth mattes are available: *Alex* (150 frames), *castle* (285 frames), and *Dmitriy* (150 frames). Figure 6.1 shows a representative frame from each of these videos. We compare our method against the two state-of-the-art video matting techniques which, like ours, only require sparse user input: *Motion-aware KNN matting* (MAKNN) [Li, Chen and Tang 2013] and *Sparse Low-Rank matting* (SLR) [Zou et al. 2019], as well as against Adobe After Effects Rotobrush Tool [Adobe Inc. 2019]. Since no source code was publicly available for MAKNN and SLR, we used our own implementations, reproducing them as faithfully as possible. For simplicity, we implemented these methods in MATLAB. For a full-HD video (*i.e.*, 1920×1080), our MATLAB implementations of MAKNN and SLR take approximately 10 and 30 minutes per frame, respectively. Thus, a run-time comparison against the CUDA implementation of our method (which takes a few milliseconds per frame) is not provided.

We compare results produced by all methods using the first and last frames as keyframes; then adding a keyframe in the middle; then using 5, and 9 keyframes equally spaced across the video. The techniques of Li et al. [Li, Chen and Tang 2013] and Zou et al. [Zou et al. 2019] do not make use of the trimap on the last frame, since they are not

Figure 6.1: Representative frames from three videos with ground truth [Erofeev et al. 2015] used for quantitative comparisons involving sparse-input video matting techniques.

Alex castle Dmitriy

Source: The authors and Erofeev et al. (2015).

able to propagate the matte backwards; ours, instead, makes full use of every trimap.

To evaluate temporal coherence for the obtained alpha matte, previous techniques [Li, Chen and Tang 2013, Zou et al. 2019, Johnson, Cholakkal and Rajan 2017] used a metric proposed by Lee et al. [Lee, Yoon and Lee 2010]. Instead, we use the spatial accuracy (SSDA) and temporal coherency (dtSSD, MESSDdt) metrics by Erofeev et al., as they better describe the perceptual quality of the matte [Erofeev et al. 2015]. For a frame t, these metrics are given by:

$$\text{SSDA} = \sqrt{\sum_p (\alpha_p^t - \bar{\alpha}_p^t)^2}, \tag{6.1}$$

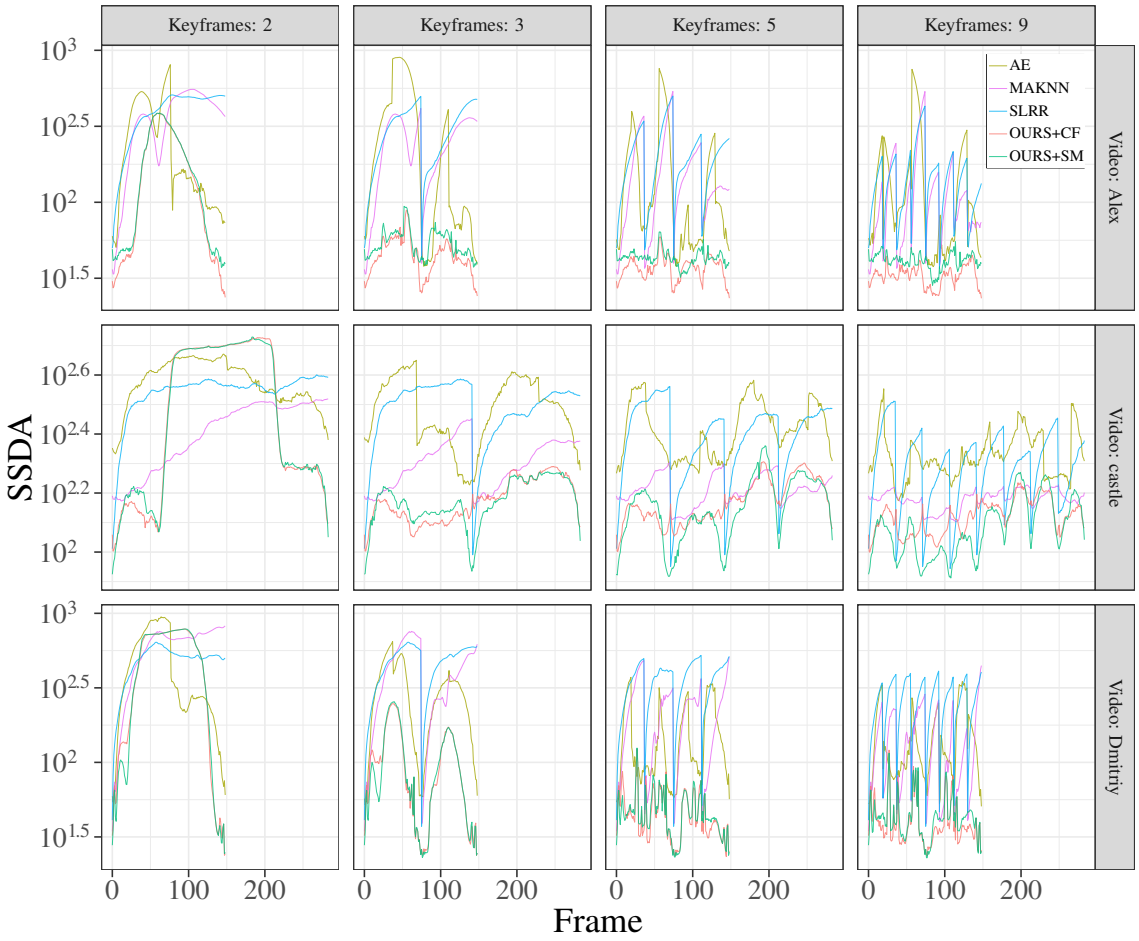
$$\text{dtSSD} = \sqrt{\sum_p \left(\frac{\mathrm{d}\alpha_p^t}{\mathrm{d}t} - \frac{\mathrm{d}\bar{\alpha}_p^t}{\mathrm{d}t} \right)^2}, \tag{6.2}$$

$$\text{MESSDdt} = \sum_p \left| \left(\alpha_p^t - \bar{\alpha}_p^t\right)^2 - \left(\alpha_{p+v_p^t}^{t+1} - \bar{\alpha}_{p+v_p^t}^{t+1}\right)^2 \right|, \tag{6.3}$$

where α_p^t and $\bar{\alpha}_p^t$ denote, respectively, the computed and the ground truth alpha values for pixel p of frame t. Likewise, $\frac{\mathrm{d}\alpha_p^t}{\mathrm{d}t}$ and $\frac{\mathrm{d}\bar{\alpha}_p^t}{\mathrm{d}t}$ are their corresponding derivatives considering the values of alpha at pixel p in frames t and $(t-1)$. v_p^t denotes the motion vector at pixel p, frame t, computed using the optical flow method by Sun et. al [Sun, Roth and Black 2010].

We compare two variants of our technique, each using a traditional alpha matting solution to initialize the matte data on the keyframes: *Shared Matting* (with optimization step) - SM [Gastal and Oliveira 2010] and *Closed-form Matting* - CF [Levin, Lischinski and Weiss 2008]. Figures 6.2 to 6.4 show the three error metrics for the results obtained using the four solutions for all frames of the three videos. Note that with three or more keyframes both variants of our technique perform significantly better than the competing

Figure 6.2: Comparison of the matte propagation methods under the SSDA error metric. Smaller values are better. AE - *After Effects Rotobrush Tool* [Adobe Inc. 2019, Bai et al. 2009], MAKNN - *Motion-aware KNN Matting* [Li, Chen and Tang 2013], SLR - *Sparse Low-Rank Representation Ratting* [Zou et al. 2019], OURS+CF - Our method using *Closed-form Matting* [Levin, Lischinski and Weiss 2008] initialization and OURS+SM - Our method using *Shared Matting* [Gastal and Oliveira 2010] initialization.



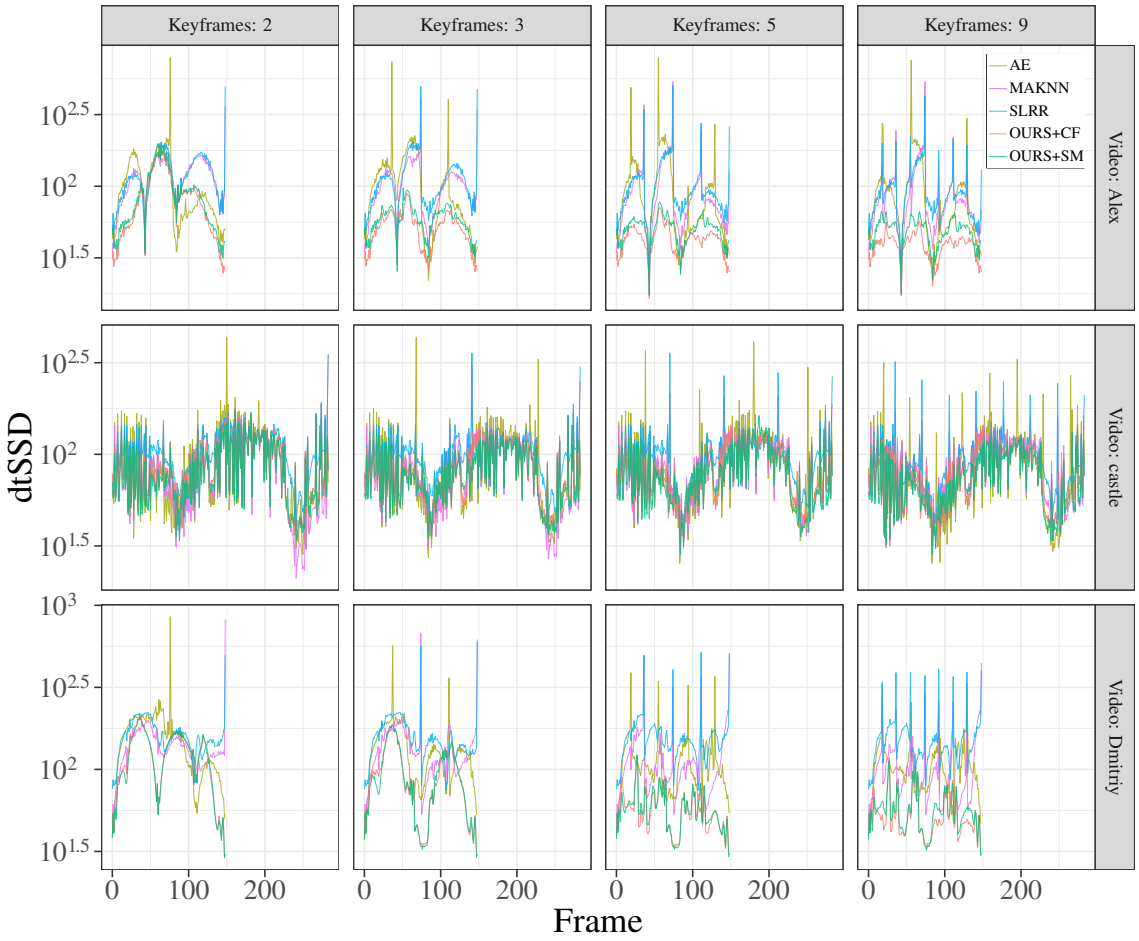
Source: The authors.

ones. The error decreases as more keyframes are provided.

We have found that providing one trimap about every 30 frames is enough for obtaining good results for most videos. The total number of keyframes can be reduced if, instead of uniformly distributing them over the video, as done for all examples shown in the thesis and supplementary materials, one selects the keyframes based on events such as disocclusions, lighting changes, fast object motion, etc.

The graphs in Figures 6.2 to 6.4 also show that the competing techniques produce error spikes whenever a keyframe is reached. This happens because these algorithms only propagate the matte forward, favoring the accumulation of errors. Our method, on the other hand, propagates the mattes to unconstrained video frames in both directions,

Figure 6.3: Comparison of the matte propagation methods under the dtSSD error metric. Smaller values are better. AE - *After Effects Rotobrush Tool* [Adobe Inc. 2019, Bai et al. 2009], MAKNN - *Motion-aware KNN Matting* [Li, Chen and Tang 2013], SLR - *Sparse Low-Rank Representation Ratting* [Zou et al. 2019], OURS+CF - Our method using *Closed-form Matting* [Levin, Lischinski and Weiss 2008] initialization and OURS+SM - Our method using *Shared Matting* [Gastal and Oliveira 2010] initialization.

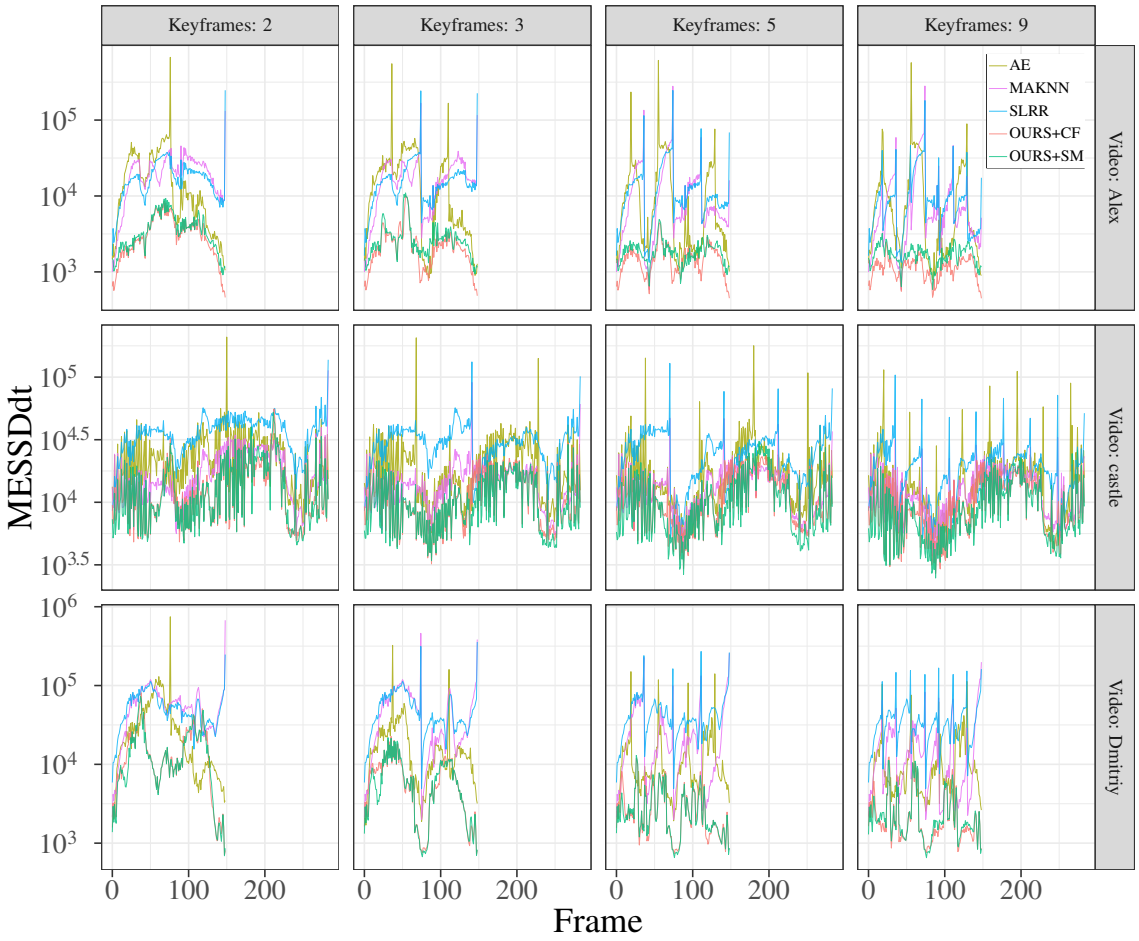


Source: The authors.

producing more temporally-consistent results.

Table 6.1 summarizes the results of the quantitative evaluation. It shows the average per-frame error computed considering the three error metrics for each video sequence, using nine keyframes. It confirms the results observed by inspecting the graphs. The two tested variants of our technique (Ours + SM) and (Ours + CF) performed significantly better than MAKNN, SLR, and AE in the three metrics for all tested videos. For the videos Alex and Dmitriy, our SSDA and MESSDdt results are one order of magnitude better than the other approaches. The last row on Table 6.1 (Total) shows the average per-frame error considering all frames in the three videos. Overall, the results of our technique were 45% more accurate (SSDA), 31% more temporally coherent (dtSSD), and

Figure 6.4: Comparison of the matte propagation methods under the MESSDdt error metric. Smaller values are better. AE - *After Effects Rotobrush Tool* [Adobe Inc. 2019, Bai et al. 2009], MAKNN - *Motion-aware KNN Matting* [Li, Chen and Tang 2013], SLR - *Sparse Low-Rank Representation Ratting* [Zou et al. 2019], OURS+CF - Our method using *Closed-form Matting* [Levin, Lischinski and Weiss 2008] initialization and OURS+SM - Our method using *Shared Matting* [Gastal and Oliveira 2010] initialization.



Source: The authors.

64% more temporally coherent considering motion estimation (MESSDdt).

The *castle* video is a challenging test for video matting techniques. The noisy plots for such video result from the difficulty to distinguish between the dark hair and some dark elements in the background. Such ambiguities introduce errors in the obtained mattes. Since dtSSD is based on the sum of the magnitudes of the temporal per-pixel derivatives, the value of dtSSD increases with such errors. The errors tend to persist for longer periods in the mattes generated by MAKNN and SLR, in this case making the magnitude of their temporal derivatives small and, consequently, their corresponding plots less noisy, despite the bigger errors. The explanation for the MESSDdt plots is similar. Our method has, on average, the lowest error and also the best accuracy (*i.e.*,

Table 6.1: Mean error metrics computed for the three video sequences using nine keyframes. AE - Adobe After Effects Rotobrush Tool, MAKNN - Motion-aware KNN matting, SLR - Sparse Low-Rank matting, OURS+CF - Ours with Closed-form Matting, and OURS+SM - Ours with Shared Matting. Please refer to text for details.

		Technique				
Video	Metric	AE	MAKNN	SLR	OURS+CF	OURS+SM
Alex	SSDA	147.83	121.80	147.66	**33.18**	41.44
	dtSSD	90.76	85.49	93.18	**38.58**	48.48
	MESSDdt	16,076.97	12,419.27	11,965.11	**1,166.11**	1,705.46
castle	SSDA	214.65	150.93	200.01	125.83	**111.91**
	dtSSD	85.84	84.90	92.76	74.81	**70.61**
	MESSDdt	15,692.79	12,540.72	19,274.85	9,206.04	**7,911.91**
Dmitriy	SSDA	144.66	140.65	258.17	**43.59**	45.51
	dtSSD	104.97	102.90	154.34	**55.51**	57.38
	MESSDdt	11,853.67	19,001.01	38,768.78	2,851.13	**2,769.59**
Total	SSDA	178.71	140.12	200.46	80.64	**76.48**
	dtSSD	91.53	89.20	108.08	**60.28**	61.24
	MESSDdt	14,732.34	14,090.91	22,279.33	5,492.45	**4,980.80**

lowest SSDA). Please see the accompanying videos in the supplementary material for a side-by-side comparison of these results.

6.2 Qualitative Evaluation

We also compared our technique with methods that require one trimap per frame. For this evaluation, we used ten test videos (ground truth not available) from the video matting benchmark [Erofeev et al. 2015] (Figure 6.5). Such videos contain challenging elements for video matting, such as fur and long hair (*Artem*, *juneau*, *Slava*, *Vitaliy* and *woods*), semi-transparent objects (*flowers*), lighting changes (*city*), object deformations (*concert*), fast motions (*rain*), and moving background (all of them).

The comparison was peformed against the top four ranked techniques in the video matting benchmark [Erofeev et al. 2015]: *Deep Matting* (DM) [Xu et al. 2017], *Self-Adaptive Matting* (SAM) [Cao et al. 2019], *Learning Based Matting* (LB) [Zheng and Kambhamettu 2009], and *Sparse Sampling Matting* (SpSM) [Karacan, Erdem and Erdem 2017]. These methods require one trimap per frame. For each direct comparison, we initialize the keyframe mattes for our technique using the results of the corresponding method taken at every fifteen frames. Nevertheless, our technique is able to produce similar results. Figure 6.6 compares mattes generated by these methods and mattes propagated by our technique for a frame halfway between two keyframes.

The goal of our method is to propagate the matte data to the entire video using a

Figure 6.5: First frame of videos from *the video matting benchmark* [Erofeev et al. 2015] used in the qualitative comparison.

Artem city concert

flowers juneau rain

Slava snow Vitali y

woods

Source: Erofeev et al. (2015).

Figure 6.6: Qualitative comparison of results produced by techniques that require one trimap per frame against results produced by our method. DM - *Deep Matting* [Xu et al. 2017], SAM - *Self-Adaptive Matting* [Cao et al. 2019], LB - *Learning Based Matting* [Zheng and Kambhamettu 2009] and SpSM - *Sparse Sampling Matting* [Karacan, Erdem and Erdem 2017]. OURS+DM, OURS+SAM, OURS+LB and OURS+SpSM stand for our method initialized by these respective matting methods every 15 frames.

Source: The authors and Erofeev et al. (2015).

small number of keyframes and, possibly, a few additional user edits (scribbles). Since dense methods require one trimap per frame, a quantitative comparison with our technique would not be appropriate as, among other things, it would not allow the user to perform interactive edits. Note that, in the limit, one could use our technique with one keyframe per frame, but this would defeat the purpose of our method.

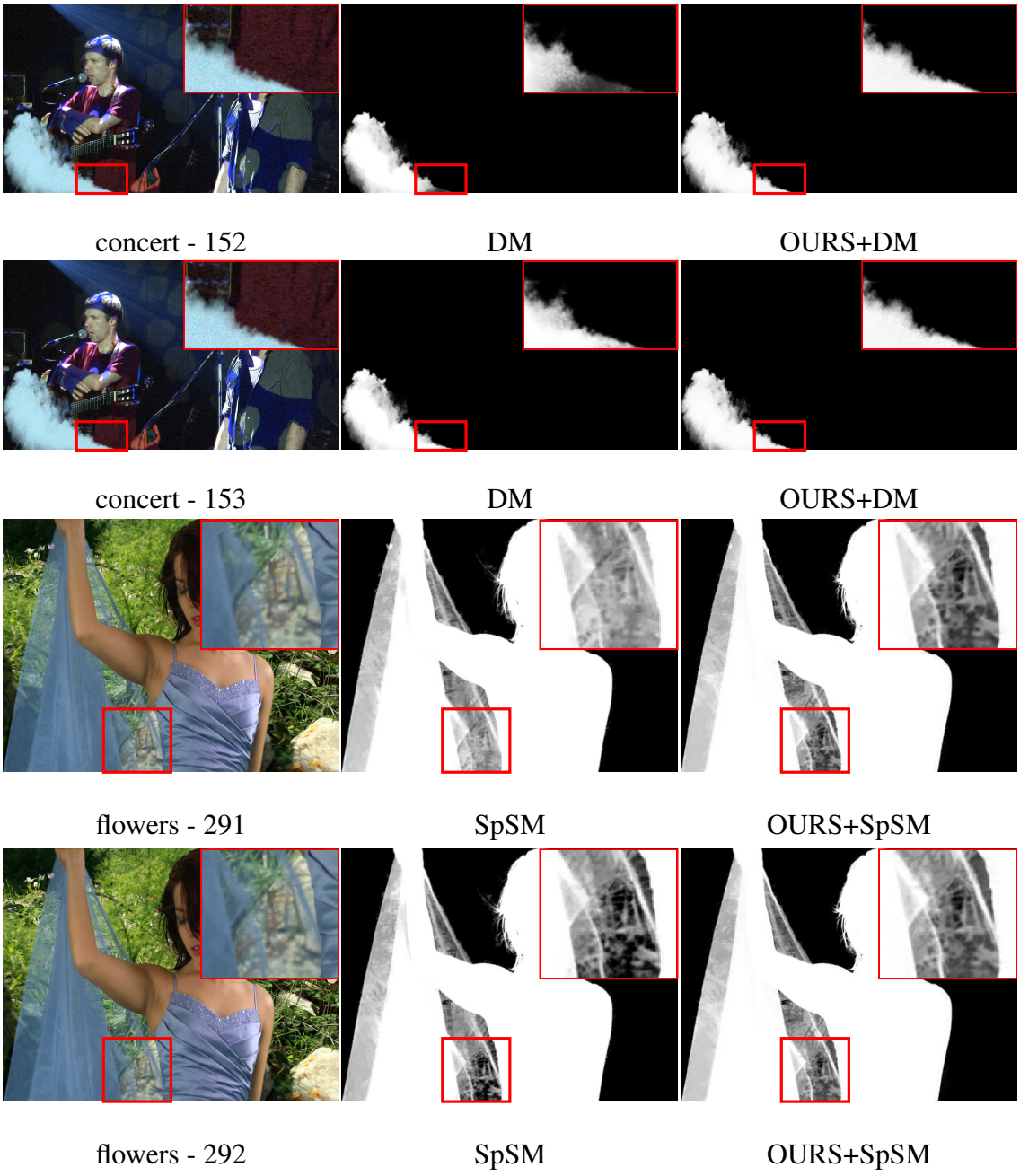
We also performed a similar qualitative comparison against Adobe After Effects Rotobrush Tool [Adobe Inc. 2019, Bai et al. 2009]. For this, we initialized the same keyframes used with our method (*i.e.*, one keyframe every 15 frames), trying to produce the best result with user-defined scribbles. The Rotobrush Tool is very effective in terms of editing individual frames, allowing one to correct an obtained frame matte by drawing a few strokes. Its main limitation, however, is the fact that, unlike our method, it only propagates the matte either forwards or backwards and, therefore, is unable to make smooth transitions between keyframes. Hence, it requires the user to constantly correct the matte as the video progresses. Figure 6.6 (AE) and a series of videos in the supplementary materials show qualitative comparisons between the mattes obtained with our technique and the Rotobrush Tool for the ten test videos. The video *flowers* contains large portions of semitranslucent materials, making this example particularly hard even for dense-input video matting techniques. When initialized with a matte generated by *Deep Matting* (DM), our technique prograte matte results that are qualitative better than the ones obtained by the other three dense-input methods (SAM, LB, and SpSM), and clearly better than the one produced by the Rotobrush Tool (Figure 6.6).

Figure 6.7 depicts examples of temporal jittering produced by Deep Matting [Xu et al. 2017] and by Sparse Sampling Matting [Karacan, Erdem and Erdem 2017]. By performing both forward and backward matte propagation, our technique is able to generate more temporally-coherent results when using these same techniques to initialize one keyframe at every fifteen frames.

6.3 Videos with Complex Occlusion Patterns

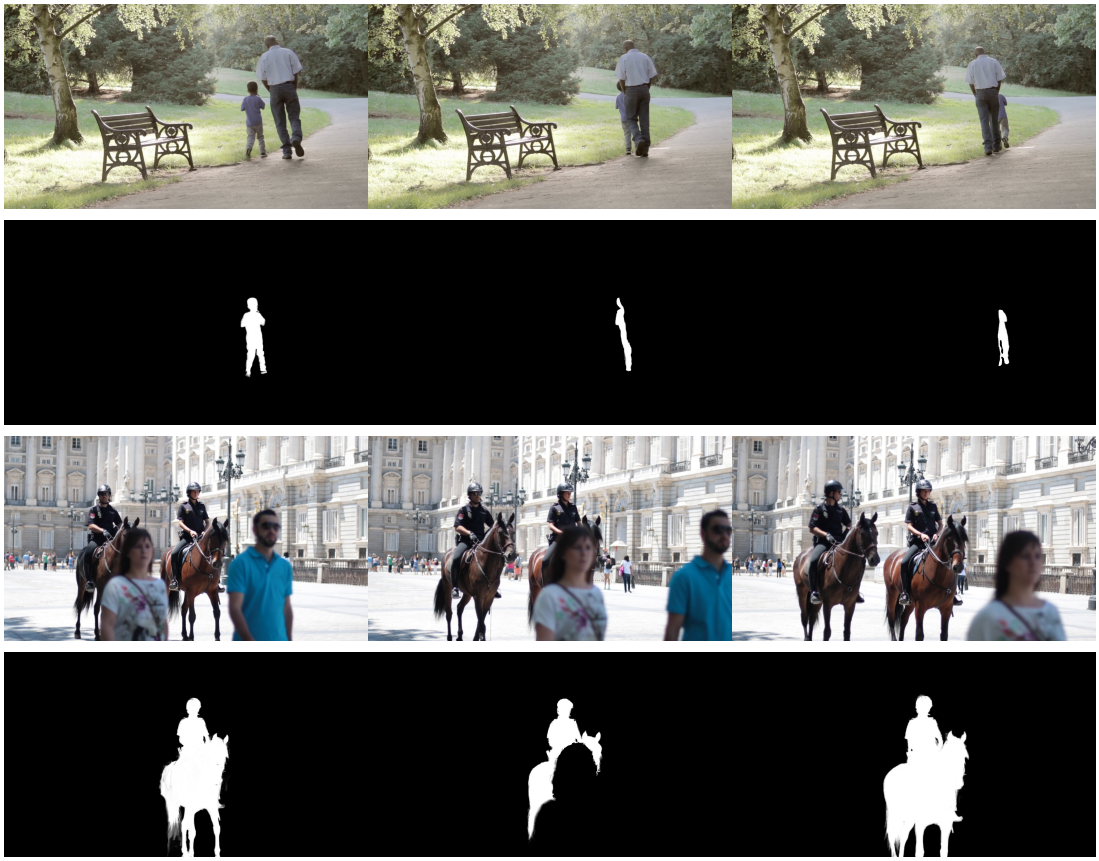
Since our method depends on optical flow estimation for finding correlations between pixels in different frames, optical flow mismatches caused by occlusions may produce matte propagation errors. Our method is able to handle both partial and complete occlusions of foreground objects, given keyframes before the object disappears and after it reappears. Figure 6.8 shows frames from two video sequences exhibiting foreground

Figure 6.7: Comparison of temporal coherence between techniques. In these examples, *Deep Matting* (DM) [Xu et al. 2017] and *Sparse Sampling Matting* (SpSM) [Karacan, Erdem and Erdem 2017] present sudden changes in the alpha channel between consecutive frames, which results in temporal jittering. Our technique is able to generate more temporally-coherent results when using these same techniques to initialize one out of fifteen keyframes. Please refer to the supplementary material for video results.



Source: The authors and Erofeev et al. (2015).

Figure 6.8: Frames from videos showing foreground elements partially occluding others (first and third rows). Corresponding extract mattes for the boy (second row), and for one horse and policeman (fourth row). These were obtained by propagating mattes of adjacent frames extracted using our interactive matting interface.



Source: The authors.

elements partially occluded others, and the corresponding mattes for the occluded elements. These were obtained by propagating mattes of adjacent frames extracted using our interactive matting interface. Handling occlusions requires, in general, user intervention two or three frames around the frames where the occlusion happens. Please refer to supplementary material for video examples.

6.4 Limitations

The recursive filter used to propagate the matte across the video has an exponential decay. In a machine with infinite floating point precision, this would not be a problem, but, since in our implementation we use 32-bit floating points, at some point in long propagations, values will not be representable anymore, stopping the propagation. We experimented using 64-bit floating points, but this only increases the propagation by a few frames. To avoid this problem, our technique requires one trimap at approximately every 15 frames. Reducing the number of needed keyframes is an important direction for future exploration.

Our matte propagation may produce incorrect results due to fast motions and foreground-background color ambiguities, as shown in Figure 6.9 for the actress' arm in the *rain* sequence. In the *city* sequence, background lighting changes and also foreground-background ambiguities cause our method to misclassify some foreground pixels. Both types of error can be fixed with additional user input (*i.e.*, more trimaps). Finally, our method relies on the quality of the techniques used to extract the mattes for the keyframes. In the videos *juneau* and *woods*, since the initialization mattes for hair produced by the used techniques already contain background artifacts, our method propagates them.

6.5 Summary

This chapter compared our method against state-of-art video-matting techniques. When compared with sparse-input methods, we showed that our method produces significantly better results according to three different metrics: one for per-frame accuracy (SSDA), and two for temporal coherency (dtSSD and MESSDdt). Considering dense-input methods, we also showed that our technique requires only about 7% of the amount of user input required by them to produce similar quality results. Finally, we showed that

Figure 6.9: Due to fast motions and foreground/background color ambiguities, using evenly distributed trimaps our technique could not avoid these undesired artifacts in the *rain* video sequence. Such artifacts can be avoided by repositioning the keyframes, or adding new ones.

rain - Frame 16 DM OURS+DM

Source: The authors and Erofeev et al. (2015).

our technique can handle videos containing non-trivial foreground occlusions.

7 CONCLUSION

We presented an efficient temporally-coherent matte-propagation method for videos. Our technique uses a sparse set of trimaps, requiring a relatively small amount of user input. Our solution performs both forward and backward matte propagation, lending to better temporal coherence. It is also orthogonal to the choice of alpha matte technique applied to the keyframes, allowing us to select the one that works best for the type of video at hand.

We demonstrated the effectiveness of our technique by performing quantitative and qualitative evaluations against the state-of-the-art methods for video matting. Compared to approaches that only require sparse input, our solution performs significantly better with respect to three error metrics. When compared to techniques that require one trimap per frame, ours produces similar-quality results while using 15 times less user input.

Given its computational efficiency, our technique provides instant feedback, allowing the development of interactive video matting systems for accurate matte extraction and compositing. Since the user only has to wait for the recursive filter to obtain some visual feedback, our technique provides instant feedback, as opposed to other state-of-art sparse-input video matting methods [Li, Chen and Tang 2013, Zou et al. 2019]. Finally, our method is able to interpolate the matte propagated between two keyframes, which results in less user input required than commercial tool Rotobrush [Adobe Inc. 2019], which only propagates the matte in one direction.

REFERENCES

Adobe Inc. **Adobe After Effects**. 2019. Available from Internet: <https://www.adobe.com/products/aftereffects.html>.

AKSOY, Y.; AYDIN, T. O.; POLLEFEYS, M. Designing effective inter-pixel information flow for natural image matting. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2017. p. 228–236.

BAI, X.; SAPIRO, G. Geodesic Matting: A Framework for Fast Interactive Image and Video Segmentation and Matting. **Int. J. Comput. Vision**, Kluwer Academic Publishers, Hingham, MA, USA, v. 82, n. 2, p. 113–132, abr. 2009.

BAI, X.; WANG, J.; SIMONS, D. Towards Temporally-coherent Video Matting. In: **Proceedings of the 5th International Conference on Computer Vision/Computer Graphics Collaboration Techniques**. Berlin, Heidelberg: Springer-Verlag, 2011. (MIRAGE'11), p. 63–74.

BAI, X. et al. Video SnapCut: Robust Video Object Cutout Using Localized Classifiers. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 28, n. 3, p. 70:1–70:11, jul. 2009.

BAKER, S. et al. A Database and Evaluation Methodology for Optical Flow. **Int. J. Comput. Vision**, Kluwer Academic Publishers, Hingham, MA, USA, v. 92, n. 1, p. 1–31, mar. 2011.

BAO, L.; YANG, Q.; JIN, H. Fast Edge-Preserving PatchMatch for Large Displacement Optical Flow. **IEEE Transactions on Image Processing**, v. 23, n. 12, p. 4996–5006, Dec 2014.

BARNES, C. et al. PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 28, n. 3, p. 24:1–24:11, jul. 2009.

BESSE, F. et al. PMBP: PatchMatch Belief Propagation for Correspondence Field Estimation. **International Journal of Computer Vision**, v. 110, n. 1, p. 2–13, Oct 2014.

BUTLER, D. J. et al. A naturalistic open source movie for optical flow evaluation. In: **Proceedings of the 12th European Conference on Computer Vision - Volume Part VI**. Berlin, Heidelberg: Springer-Verlag, 2012. (ECCV'12), p. 611–625.

CAO, G. et al. Patch-Based Self-Adaptive Matting for High-Resolution Image and Video. **The Visual Computer**, v. 35, n. 1, p. 133–147, Jan 2019. ISSN 1432-2315.

CHEN, J. et al. Efficient Segmentation-Based PatchMatch for Large Displacement Optical Flow Estimation. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 29, n. 12, p. 3595–3607, Dec 2019.

CHEN, Q.; LI, D.; TANG, C. KNN Matting. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 35, n. 9, p. 2175–2188, Sep. 2013.

CHO, T. S.; AVIDAN, S.; FREEMAN, W. T. The Patch Transform. In: . Washington, DC, USA: IEEE Computer Society, 2010. v. 32, n. 8, p. 1489–1501.

CHUANG, Y.-Y. et al. Video Matting of Complex Scenes. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 21, n. 3, p. 243–248, jul. 2002. ISSN 0730-0301.

CRIMINISI, A.; PEREZ, P.; TOYAMA, K. Region Filling and Object Removal by Exemplar-based Image Inpainting. **IEEE Transactions on Image Processing**, v. 13, n. 9, p. 1200–1212, Sep. 2004.

CUSP. Available from Internet: <https://cusplibrary.github.io>.

DATAR, M. et al. Locality-sensitive Hashing Scheme Based on P-stable Distributions. In: **Proceedings of the Twentieth Annual Symposium on Computational Geometry**. New York, NY, USA: ACM, 2004. (SCG '04), p. 253–262.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. **Commun. ACM**, ACM, New York, NY, USA, v. 51, n. 1, p. 107–113, jan. 2008.

EROFEEV, M. et al. Perceptually Motivated Benchmark for Video Matting. In: **Proceedings of the British Machine Vision Conference (BMVC)**. [S.l.]: BMVA Press, 2015. p. 99.1–99.12.

FAN, Q. et al. JumpCut: Non-successive Mask Transfer and Interpolation for Video Cutout. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 34, n. 6, p. 195:1–195:10, oct. 2015.

FARBMAN, Z. et al. Edge-preserving Decompositions for Multi-scale Tone and Detail Manipulation. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 27, n. 3, p. 67:1–67:10, aug. 2008.

GASTAL, E. S. L.; OLIVEIRA, M. M. Shared Sampling for Real-Time Alpha Matting. **Computer Graphics Forum**, v. 29, n. 2, p. 575–584, May 2010.

GASTAL, E. S. L.; OLIVEIRA, M. M. Domain transform for edge-aware image and video processing. **ACM Trans. Graph.**, v. 30, n. 4, p. 69:1–69:12, 2011.

GEIGER, A.; LENZ, P.; URTASUN, R. Are we ready for autonomous driving? The KITTI vision benchmark suite. In: **2012 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2012. p. 3354–3361.

HARALICK, R. M.; SHAPIRO, L. G. Image segmentation techniques. **Computer Vision, Graphics, and Image Processing**, v. 29, n. 1, p. 100–132, 1985.

HARRIS, M. **Optimizing Parallel Reduction in CUDA**. 2007.

HE, K. et al. A global sampling method for alpha matting. In: **Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition**. Washington, DC, USA: IEEE Computer Society, 2011. (CVPR '11), p. 2049–2056.

HE, K.; SUN, J. Computing nearest-neighbor fields via Propagation-Assisted KD-Trees. In: **Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. Washington, DC, USA: IEEE Computer Society, 2012. (CVPR '12), p. 111–118.

HE, K.; SUN, J.; TANG, X. Guided Image Filtering. **IEEE Trans. Pattern Anal. Mach. Intell.**, IEEE Computer Society, Washington, DC, USA, v. 35, n. 6, p. 1397–1409, jun. 2013.

HORN, B. K. P.; SCHUNCK, B. G. Determining Optical Flow. **Artif. Intell.**, Elsevier Science Publishers Ltd., Essex, UK, v. 17, n. 1-3, p. 185–203, aug. 1981.

HOSNI, A. et al. Temporally Consistent Disparity and Optical Flow via Efficient Spatio-temporal Filtering. In: **Proceedings of the 5th Pacific Rim Conference on Advances in Image and Video Technology - Volume Part I**. Berlin, Heidelberg: Springer-Verlag, 2012. (PSIVT'11), p. 165–177.

HUI, T.; TANG, X.; LOY, C. C. LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation. In: **2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2018. p. 8981–8989.

ILG, E. et al. FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks. **arXiv:1612.01925 [cs]**, 2016.

JOHNSON, J.; CHOLAKKAL, H.; RAJAN, D. L1-regularized Reconstruction Error as Alpha Matte. **IEEE Signal Processing Letters**, PP, p. 1–1, 02 2017.

JOHNSON, J.; RAJAN, D.; CHOLAKKAL, H. Temporal Trimap Propagation using Motion-Assisted Shape Blending. In: **2015 Visual Communications and Image Processing (VCIP)**. [S.l.: s.n.], 2015. p. 1–4.

JOHNSON, J. et al. Sparse Coding for Alpha Matting. **IEEE Transactions on Image Processing**, v. 25, n. 7, p. 3032–3043, July 2016.

KARACAN, L.; ERDEM, A.; ERDEM, E. Alpha Matting With KL-Divergence-Based Sparse Sampling. **IEEE Transactions on Image Processing**, v. 26, n. 9, p. 4523–4536, Sep. 2017.

KOMODAKIS, N. Image Completion Using Global Optimization. In: **Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1**. Washington, DC, USA: IEEE Computer Society, 2006. (CVPR '06), p. 442–452.

KORMAN, S.; AVIDAN, S. Coherency Sensitive Hashing. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 38, n. 6, p. 1099–1112, June 2016.

LANG, M. et al. Practical Temporal Consistency for Image-based Graphics Applications. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 31, n. 4, p. 34:1–34:8, jul. 2012.

LEE, S.-Y.; YOON, J.-C.; LEE, I.-K. Temporally Coherent Video Matting. **Graph. Models**, Academic Press Professional, Inc., San Diego, CA, USA, v. 72, n. 3, p. 25–33, may 2010.

LEVIN, A.; LISCHINSKI, D.; WEISS, Y. Colorization Using Optimization. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 23, n. 3, p. 689–694, aug. 2004.

LEVIN, A.; LISCHINSKI, D.; WEISS, Y. A Closed-Form Solution to Natural Image Matting. **IEEE Trans. Pattern Anal. Mach. Intell.**, IEEE Computer Society, Washington, DC, USA, v. 30, n. 2, p. 228–242, feb. 2008.

LI, D.; CHEN, Q.; TANG, C.-K. Motion-Aware KNN Laplacian for Video Matting. In: **Proceedings of the 2013 IEEE International Conference on Computer Vision**. Washington, DC, USA: IEEE Computer Society, 2013. (ICCV '13), p. 3599–3606.

LOWE, D. G. Distinctive Image Features from Scale-Invariant Keypoints. **Int. J. Comput. Vision**, Kluwer Academic Publishers, Norwell, MA, USA, v. 60, n. 2, p. 91–110, nov. 2004.

LUCAS, B. D.; KANADE, T. An Iterative Image Registration Technique with an Application to Stereo Vision. In: **Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1981. (IJCAI'81), p. 674–679.

NanoGUI. 2019. Available from Internet: <https://github.com/wjakob/nanogui>.

PERONA, P.; MALIK, J. Scale-Space and Edge Detection Using Anisotropic Diffusion. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 12, n. 7, p. 629–639, July 1990.

RHEMANN, C. et al. A perceptually motivated online benchmark for image matting. In: **2009 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2009.

RONG, G.; TAN, T.-S. Jump Flooding in GPU with Applications to Voronoi Diagram and Distance Transform. In: **Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games**. New York, NY, USA: ACM, 2006. (I3D '06), p. 109–116.

RUBINSTEIN, M.; SHAMIR, A.; AVIDAN, S. Improved Seam Carving for Video Retargeting. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 27, n. 3, p. 16:1–16:9, aug. 2008.

SANDERS, J.; KANDROT, E. **CUDA by Example: An Introduction to General-Purpose GPU Programming**. 1st. ed. [S.l.: s.n.], 2010.

SHAHRIAN, E. et al. Temporally Coherent and Spatially Accurate Video Matting. **Computer Graphics Forum**, The Eurographs Association; John Wiley; Sons, Ltd., Chichester, UK, v. 33, n. 2, p. 381–390, may 2014.

SHEN, X. et al. Automatic Portrait Segmentation for Image Stylization. In: **Proceedings of the 37th Annual Conference of the European Association for Computer Graphics**. Goslar Germany, Germany: Eurographics Association, 2016. (EG '16), p. 93–102.

SHEN, X. et al. Deep Automatic Portrait Matting. In: LEIBE, B. et al. (Ed.). **Computer Vision – ECCV 2016**. Cham: Springer International Publishing, 2016. p. 92–107.

SIMAKOV, D. et al. Summarizing visual data using bidirectional similarity. In: **2008 IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2008. p. 1–8.

SINDEEV, M.; KONUSHIN, A.; ROTHER, C. Alpha-Flow for Video Matting. In: **Proceedings of the 11th Asian Conference on Computer Vision - Volume Part III**. Berlin, Heidelberg: Springer-Verlag, 2013. (ACCV'12), p. 438–452.

ŠT'AVA, O.; BENEŠ, B. Chapter 35 - Connected Component Labeling in CUDA. In: HWU, W.-m. W. (Ed.). **GPU Computing Gems Emerald Edition**. [S.l.]: Elsevier, 2011. p. 569–581.

SUN, D.; ROTH, S.; BLACK, M. J. Secrets of Optical Flow Estimation and Their Principles. In: **IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)**. [S.l.]: IEEE, 2010. p. 2432–2439.

SZELISKI, R. **Computer Vision: Algorithms and Applications**. 1st. ed. [S.l.]: Springer Science & Business Media.

TANG, Z. et al. Video Matting via Opacity Propagation. **Vis. Comput.**, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 28, n. 1, p. 47–61, jan. 2012.

TOMASI, C.; MANDUCHI, R. Bilateral Filtering for Gray and Color Images. In: **Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)**. [S.l.: s.n.], 1998. p. 839–846.

TONG, R.-F.; ZHANG, Y.; DING, M. Video Brush: A Novel Interface for Efficient Video Cutout. **Computer Graphics Forum**, v. 30, n. 7, p. 2049–2057, 2011.

VOLZ, S. et al. Modeling Temporal Coherence for Optical Flow. In: **Proceedings of the 2011 International Conference on Computer Vision**. Washington, DC, USA: IEEE Computer Society, 2011. (ICCV '11), p. 1116–1123.

WANG, J. et al. Interactive video cutout. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 24, n. 3, p. 585–594, jul. 2005.

WANG, Y.-S. et al. Optimized Scale-and-stretch for Image Resizing. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 27, n. 5, p. 118:1–118:8, dec. 2008.

XU, N. et al. Deep Image Matting. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2017. p. 2970–2979.

ZHANG, Y.; TANG, Y.-L.; CHENG, K.-L. Efficient Video Cutout by Paint Selection. **Journal of Computer Science and Technology**, v. 30, n. 3, p. 467–477, May 2015.

ZHENG, Y.; KAMBHAMETTU, C. Learning Based Digital Matting. In: **2009 IEEE 12th International Conference on Computer Vision**. [S.l.: s.n.], 2009. p. 889–896.

ZIMMER, H.; BRUHN, A.; WEICKERT, J. Optic Flow in Harmony. **Int. J. Comput. Vision**, Kluwer Academic Publishers, Hingham, MA, USA, v. 93, n. 3, p. 368–388, jul. 2011.

ZOU, D. et al. Unsupervised Video Matting via Sparse and Low-Rank Representation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, p. 1–1, 2019.

APPENDIX A — SUPPLEMENTARY MATERIAL

The supplementary material of this dissertation is available at <http://www.inf.ufrgs.br/~oliveira/pubs_files/VM/SM>.

This material contains:

- An example of usage of our video matting interface (Chapter 5).
- Results of our quantitative comparisons (Section 6.1), including interactive graphs and videos.
- Results of our qualitative comparisons (Section 6.2).
- The matte extraction of a video with complex foreground occlusion patterns (Section 6.3).

APPENDIX B — DETECTING THE LARGEST CONNECTED COMPONENT IN GPU

In this appendix chapter we describe the procedures used for detecting the largest connected component in an image (required in Section 4.4) using a GPU implementation, since it is not a trivial operation to parallelize. The algorithm consists of two major steps. The first one is labeling the connected components which is performed by a parallel union-find-based algorithm. In order to identify which of the components is the largest a parallel reduction is performed in order to compute the size of the largest component in the second step.

B.1 Finding Connected Components

Connected Components Labeling (CCL), in the context of image processing, is a task fof finding connected regions in the input image. It is a required component for many computer vision algorithms [Šťava and Beneš 2011, Haralick and Shapiro 1985], being usually related to image segmentation. In this sense, the objective of a CCL algorithm is to label with different *ids* connected components in pixel space, accortding to a connectivity criterium. Criteria can be defined in different ways such as intensity similarity (i.e. two neighbor pixels are connected if their color intensity distance is below a certain threshold) or using an edge detector such as Canny (i.e. two neighbor pixels are connected if they are on the same side of an edge), depending on the application.

The work by Šťava and Beneš (2011) proposed, a parallelizable algorithm for finding connected components which is especially designed for the GPU. The main idea is to slit the input image into blocks and perform a union-find-based CCL for each block separately, in such a way that blocks can be processed in parallel.

Algorithm 1 depicts the first step of the algorithm. The function *BlockCCL* defines a kernel, which will execute for every thread in every block of the input image. It takes as input a two dimesional array containing segmentation information for every pixel. For example: pixels can have value 0 or 1 and neighboring pixels are connected only if they have the same segmentation value. This algorithm description can be easily adapted to other forms of connection representation.

Each thread is associated with a pixel and they will execute this code in parallel.

In order to cooperate and share results within threads in a block, the kernel uses shared memory variables. Also, the procedure $syncThreads()$ is used to syncronize their work. Once the block connected component algorithm is done, the local label assigned to each pixel is converted into a unique global label ($globalLabel()$). Finally, another kernel function is used iteratively to merge the labels between neighbor blocks using the previously generated global label, resulting in the final connected component labeling.

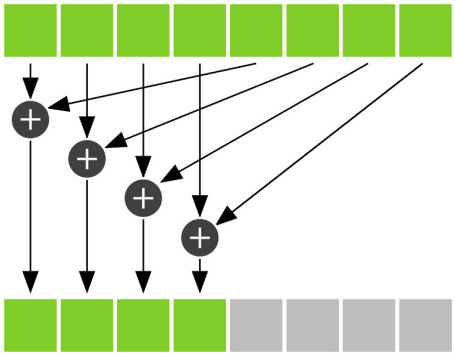
Algorithm 1 Block CCL kernel [Št'ava and Beneš 2011], for performing the Connected Compoents Algorithm in CUDA.

procedure BLOCKCCL(**in**: dSegData, **out**: dLabelData)
  shared $sSegs[]$, $sLabels[]$, $sChanged \leftarrow$ true ▷ Variables shared inside block
  $index \leftarrow threadid$
  $label \leftarrow index$
  $sSegs[index] \leftarrow dSegData[thread.globalPosition]$
  while $sChanged$ **do**
    $syncThreads()$
    $sChanged \leftarrow$ false
    $sLabels[index] \leftarrow label$
    $syncThreads()$
    $minLabel \leftarrow FindMinimumLabelConnectedAmongNeighbors(sLabels, sSegs)$
    $syncThreads()$
    if $minLabel < label$ **then**
      $atomicMin(sLabels[label], minLabel)$ ▷ Atomic is used to avoid race condition.
      $sChanged \leftarrow$ true
    end if
    $syncThreads()$
    $label \leftarrow findRoot(sLabels, label)$
  end while
  $dLabelData[thread.globalPosition] = globalLabel(block.id, label)$
end procedure

B.2 Parallel Reduction

The `reduce` operation is one of the most common and most important parallel data primitive. It is largely used to effiently process large amounts of data in databases [Dean and Ghemawat 2008] and also a building block for many parallel algorithms, especially for GPU implementations [Sanders and Kandrot 2010]. Formally, the objective of `reduce` is to apply an associative binary operator *op* such as $+$, $\times$, min or max over a collection of data. A common example would be summing all the elements of a vector,

Figure B.1: An example of parallel reduction iteration using the sum operator. Sum operations are grouped in pairs which reduces in half the number of elements for the next iteration.



Source: Sanders and Kandrot (2010).

for instance.

The idea behind the algorithm is to take advantage of the associative property of *op* to rearrange the operator in such a way that the number of operations that can be performed in parallel is maximized. For example, if we wished to sum all elements from a to h

$$a+b+c+d+e+f+g+h, \tag{B.1}$$

the equation could be rearranged to

$$(a+b)+(c+d)+(e+f)+(g+h) \tag{B.2}$$

so that the operations $a+b$, $c+d$, $e+f$, and $g+h$ could be performed in parallel, without any data dependency. Therefore, to optimize the number of operations performed in parallel, the objective is to iteratively reduce the number of elements by half using *op* until we have only one element left (the final result). See Figure B.1 for an illustration of a reduction iteration.

Let A be an array of size n which is a power of 2. The algorithm for the parallel reduction with the binary operator *op* can be expressed as shown in Algorithm 2. By assuming that all statements inside the *parfor* construction are performed in parallel, we are able to perform a task that would take $O(n)$ operations in only $O(log(n))$ iterations of parallel operations. This algorithm can be easily adapted to an array of any size n by padding the array with zeros until n is a power of 2 or taking extra care to ignore memory acesses out of bounds.

Practical code optimization problems occur when implementing this method on a GPU. Complications such as implementing the parallelization scheme as threads/blocks

Algorithm 2 Parallel reduction algorithm.

```
procedure REDUCE(in: A, in: op )
    for i ∈ [1, log2(n)] do
        parfor j ∈ [0, n/2^i − 1] do
            A[2j] ← op(A[2j], A[2j+1])
        end parfor
    end for
    return A[0]
end procedure
```

and coordinating *shared memory* access can be easily adapted to a CUDA implementation. Other optimization related issues such as avoiding branch divergence, avoiding memory bank conflicts and unrolling loops for more efficiency were addressed by the work of Harris [Harris 2007].