A PatchMatch-based Approach for Matte Propagation in Videos

Marcos H. Backes and Manuel M. Oliveira

Instituto de Informática – UFRGS, Brazil

Figure 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.

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.

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

\[ I_p = \alpha_p F_p + (1 - \alpha_p) B_p, \]  

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 \( \alpha_p \).

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) 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 foreground, to the background, and to unknown regions.

Although image matting is a well studied problem and recent works can produce high-quality results \cite{LLW08, RRW09, GO10, HRR+11, CLT13, AAP17, XPCH17}, 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 \cite{JCR17}.

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 \cite{WBC05, BWS11, JRC15}; others use a sparse set of trimaps \cite{LCT13, ZCCW19}. 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.

We present 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 \cite{GO11} 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 illustrates the use of our technique to extract mattes for a video sequence. The user provides trimaps for the first and last frames; the mattes computed for these frames are propagated to the entire video sequence.

The contributions of our work include:

- An efficient temporally-coherent matte-propagation method for videos (Section 3). 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 (Section 4). 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 work

2.1. Alpha matting

Alpha matting techniques are usually classified as sampling-based and affinity-based. Sampling-based methods \cite{GO10, HRR+11, KEE17} 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 \cite{LLW08, AAP17, CLT13} 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 \cite{STG*05} and perform matting of specific portrait images \cite{SHJ+16}. Xu et al. also used deep learning to extract high-quality mattes for given pairs of images and trimaps \cite{XPCH17}.

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 \cite{EGV15}. Although some recent video-matting techniques \cite{SPCR14, KEE17, CLCH19} are able to find interframe pixel relationships to produce temporally-coherent mattes, such techniques typically only handle up to five frames at a time.

2.2.2. Sparse-input video matting

The first work on sparse-input video matting \cite{CAC02} 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 object \cite{WBC05, BWS09, BS09, BWS11}. Trimaps were then generated from the segmentations to produce the alpha mattes. Wang et al. \cite{WBC05} 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 \cite{SKR13}. Segmentation performance was improved by decreasing the size of the graph using downsampling \cite{TZD11} and clustering \cite{ZTC15}, thus enhancing user interaction and supporting larger video sizes.

Video SnapCut \cite{BWS09} 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 \cite{LLW08} which uses the matte of the previous frame as constraints for the current one is used to ensure temporal coherency. Geodesic Matting \cite{BS09} uses an efficient fast marching algorithm to perform the initial segmentation. Trimaps are generated by dilating an unknown region on the object border as needed. This method, however, does not take into account object
movement (i.e., optical flow). Bai et al. [BWS11] 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. [JRC15, JVR16] 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 [TMWZ12, SKR13, LCT13, ZCCW19]. Tang et al. [TMWZ12] propagate the matte to the next frame using a variation of the matting Laplacian [LLW08] 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. AlphaFlow [SKR13] 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 [HST13]. 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 has shown to be useful for treating complex textures in the input image. Li et al. [LCT13] extended the ideas of KNN Matting [CLT13] to ensure temporal coherency. 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. [ZCCW19] 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 [BWSS09, BWS11, JRC15, TMWZ12, LCT13, ZCCW19], or process the video as a whole [WBC05, SKR13]. 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 [WBC05] or do not scale to current video resolutions [SKR13].

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

2.3. Domain Transform Filters

The Domain Transform is an efficient framework for performing edge-aware filtering [GO11]. Lang et al. [LWA*12] 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 [LWA*12] to efficiently propagate the matte across the input video and ensure temporal coherence. We use the domain transform’s recursive filter [GO11] 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 [GO10], using propagated foreground and background colors as the initial confidence values.

3. 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 3.1); (ii) Using the computed optical flows to propagate alpha values, as well as foreground and background colors from keyframes to unconstrained ones using a temporal version of the domain transform’s recursive filter (Section 3.2); and (iii) Refining the computed alpha values to obtain locally-smooth mattes (Section 3.3).

3.1. Computing Forward and Backward Optical Flow

We use PatchMatch [BSFG09] 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 radius r in A, PatchMatch looks for its nearest neighbor in B under a distance metric d (originally L2 distance). Finding such neighbors would usually require a quadratic number of comparisons, but PatchMatch efficiently finds an approximate solution using a randomized search. PatchMatch was originally designed for interactive image editing tasks, but has also been used for video related tasks such as video stylization [BZL*15], video segmentation [FZL*15], and optical flow [BYJ14, BRFK14, CCLX18]. For optical flow, one sets A = f and B = f+1 to compute the forward optical flow, or A = f+1 and B = f for the backward optical flow, where f is the current frame and f+1 is the next one.

The use of the edge-preserving matching cost function described by Bao et al. [BYJ14] (Equation (2)) produces more accurate matching around object borders when compared to traditional optical flow approaches [SRB10] (see Figure 2). 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 (2) 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 $\omega$. 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:

\[
d(a, b) = \frac{1}{W} \sum_{\Delta x, \Delta y \leq r} \omega(a, b, \Delta) ||A(a + \Delta) - B(b + \Delta)||^2,
\]

\[
\omega(a, b, \Delta) = \exp \left( -\frac{||A(a + \Delta) - A(a)||^2}{\sigma_f^2} \right) \exp \left( -\frac{||B(b + \Delta) - B(b)||^2}{\sigma_g^2} \right) \exp \left( -\frac{||\Delta||^2}{\sigma_r^2} \right),
\]

where $a$ and $b$ 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 $\omega$. For all examples shown in the paper, the radius of the patch is $r = 3$ (resulting in a squared patch of width 7) and the constants $\sigma_f = 0.5r$ and $\sigma_g = 0.1$. Each patch in $A$ is limited to search for matches in $B$ inside a square region of side 128 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 3.4.

3.2. 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 [LWA+12, GO11].

The alpha values, foreground and background colors for the keyframes are obtained with the use of some matting technique (e.g., [GO10], [LLW08], [XPCH17], [AAP17]). 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 $D_p^t$:

\[
D_p^t = [a_p, F_p, B_p, n_p],
\]

where $a_p$ is the pixel’s opacity value, $F_p$ and $B_p$ are, respectively, its foreground and background RGB colors, and $n_p$ is a normalization factor. For the keyframes, $a_0^t$, $F_0^t$, and $B_0^t$ are initialized by the image matting technique, and $n_0^t = 1$. 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 $F_p^t$. The propagated foreground and background colors are also used in the refinement phase (Section 3.3) to evaluate the confidence of the obtained opacity values.

During forward propagation, for every pixel $p$ in frame $t'$ we use the backward optical flow to find $p'$s temporal neighbor $q$ in the previous frame $t'^{-1}$. Likewise, during backward propagation, we use the forward optical flow to find $p$’s temporal neighbor $r$ in the next frame $t'^{+1}$. Matte and color forward propagation is then achieved using the following recursive equation for $J^t = D^t$:

\[
J^t = (1 - a^t)D^t + a^tJ^{t-1},
\]

while backward propagation uses $J^t' = D^t'$:

\[
J^t' = (1 - a^t)D^t' + a^tJ^{t+1},
\]

where $t \in \{1, 2, \ldots, n - 1\}$, $D^t$ contains the data to be propagated, $J^t$ is the filtered signal, $a \in \{0, 1\}$ is the feedback coefficient computed as $a = \exp \left( -\sqrt{2/\sigma_i} \right)$, and $d$ is the geodesic distance between neighbor samples along the temporal 1D paths defined by the optical flow. For forward propagation, $d = 1 + |F_p^t - F_q^{t-1}|$. For backward propagation, $d = 1 + |F_p^t - F_q^{t+1}|$. The values of the keyframes are not modified by the filtering process. Experimentally, we found

\[
\sigma_i = 2^{-i-1} \sqrt{3}/\sqrt{2^i - 1}.
\]

† 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 $\sigma_i = 2^{-i-1} \sqrt{3}/\sqrt{2^i - 1}$. 

Figure 2: Visual comparison of matte propagation using optical flow (d) and our version of PatchMatch.
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:

$$
\hat{c}'_p = \frac{c'_p}{n'_p}, \quad \hat{F}'_p = \frac{F'_p}{n'_p}, \quad \hat{B}'_p = \frac{B'_p}{n'_p},
$$

where $\hat{c}'_p$, $\hat{F}'_p$, and $\hat{B}'_p$ are the normalized attributes for pixel $p$ of frame $t$, and $D'_p = (\alpha'_p, \hat{F}'_p, \hat{B}'_p, n'_p)$ is the vector with the propagated data to pixel $p$ at frame $t$.

3.3. Refining the Propagated Matte

The normalized alpha values $\hat{c}'_p$ might be noisy. To refine the matte, we use the scheme presented by Gastal and Oliveira [GO10] for optimizing the alpha channel based on the matting Laplacian $L$ [LLW08] and on the confidence $c'_p$ of the obtained alpha values, which is computed as

$$
c'_p = \exp(-\delta |\alpha'_p - (\alpha'_p F'_p + (1 - \alpha'_p) B'_p)|),
$$

where $\delta = 10$. The refined matte is obtained minimizing the following energy function for each frame $t$:

$$
E_a = \alpha^T L \alpha + \lambda (\alpha - \hat{\alpha})^T K (\alpha - \hat{\alpha}) + \gamma (\alpha - \hat{\alpha})^T \Gamma (\alpha - \hat{\alpha}),
$$

where $L$ is the matting Laplacian, $\Gamma$ is a diagonal matrix where each diagonal element is defined as $c'_p$, $K$ is a diagonal matrix whose elements are 1 if $\hat{c}'_p$ is either 1 or 0, $\lambda = 100$ and $\gamma = 0.1$. $\alpha$ and $\hat{\alpha}$ are vectorized versions of, respectively, the refined and normalized alpha values for all pixels in frame $t$. Figure 3 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 $\hat{c}'_p$, the pixel color $F'_p$, and the estimated foreground $F'_p$, and background $B'_p$, colors, while enforcing smoothness on matte edges [LLW08]:

$$
E_{FB} = \sum_p \|c'_p F'_p + (1 - c'_p) B'_p - F'_p\|^2 + \|c'_p\| \left(\|F'_p\|^2 + \|B'_p\|^2\right) + \|c'_p\| \left(\|F'_p\|^2 + \|B'_p\|^2\right),
$$

where $\alpha_p$, $\hat{c}_p$, $F'_p$, $F'_p$, $B'_p$, and $B'_p$ are, respectively, the horizontal and vertical derivatives of $\alpha'_p$, $F'_p$ and $B'_p$.

3.4. 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 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. Once detected, our backward matte propagation process corrects the mattes in the previous frames. Figure 4 illustrates the process of removing false foreground components. In our GPU implementation we use a parallel connected components algorithm [SB11] and a parallel reduction sum [SK10] to count the number of pixels in the connected components and find the largest ones.

4. Interactive Video Matting

We now describe an interface for interactive video matting that allows for easy and accurate extraction of objects from videos and their compositing onto other sequences. Given an input video, it is preprocessed to compute dense forward and backward optical flows using PatchMatch, as described in Section 3.1. Then, the user specifies trimaps (through a few scribbles) for some keyframes of the video. 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 these. For the keyframes, the alpha, foreground, and background colors are obtained using Shared Matting [GO10], which allows the intermediate results to be shown as the user draws the trimap.

Once mattes have been generated for the keyframes, we use the method presented in Section 3 to propagate the matte for the remaining frames of the video. The user can visually inspect the resulting mattes and interactively refine them using additional scribbles, until a satisfactory result is obtained. Figure 5 shows the interface of our system. We refer the reader to a video demonstrating the use of our interactive video matting system, which can be found in the supplementary material.\(^\dagger\)

\(^\dagger\) The supplementary material is available here.

\(^\dagger\) The supplementary material is available here.
M. H. Backes & M. M. Oliveira / A PatchMatch-based Approach for Matte Propagation in Videos

Figure 3: Visual comparison of propagation with and without the refinement step.

(a) Propagated opacity (Section 3.2).
(b) Refined opacity (Section 3.3).
(c) Ground Truth.

Figure 4: Visual comparison of propagation with and without the removal of disconnected components step. Keyframe (a) and its pre-computed matte (c). In 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 3.4) (e).

Since GPUs have relatively small memory sizes, only (relatively) short videos can be loaded at once. On a GPU with 8 GiB 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. Automatically dispatching video segments to the GPU can be implemented in a straightforward way.

5. 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 [LCT13, ZCCW19], 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 [EGV*15]: Deep Matting [XPCH17], Self-Adaptive Matting [CLCH19], Learning Based Matting [ZK09], and Sparse Sampling Matting [KEE17]. These techniques require one trimap per frame, and we show that our method produces similar results even using a much smaller number of keyframes.

5.1. Quantitative Evaluation

For the quantitative evaluation, we used three training videos from the video matting benchmark [EGV*15] for which the ground truth mattes are available: Alex (150 frames), castle (285 frames), and Dmitriy (150 frames). Figure 6 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, do not require user input: Motion-aware KNN matting (MAKNN) [LCT13] and Sparse Low-Rank matting (SLR) [ZCCW19], as well as against Adobe After Effects Rotobrush Tool. 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.

Figure 5: 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.
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. [LCT13] and Zou et al. [ZCCW19] do not make use of the trimap on the last frame, since they are not 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 [LCT13, ZCCW19, JCR17] used a metric proposed by Lee et al. [LYL10]. Instead, we use the spatial accuracy (SSDA) and temporal coherence (dtSSD, MESSDdt) metrics by Erofeev et al., as they better describe the perceptual quality of the matte [EGV+15]. For a frame \( t \), these metrics are given by:

\[
SSDA = \sqrt{\sum_p (\alpha_p^t - \bar{\alpha}_p^t)^2},
\]

\[
dtSSD = \sqrt{\sum_p \left( \frac{d\alpha_p^t}{dt} - \frac{d\bar{\alpha}_p^t}{dt} \right)^2},
\]

\[
MESSDdt = \sum_p \left| (\alpha_p^t - \bar{\alpha}_p^t)^2 - \left( \alpha_{p+1}^t - \bar{\alpha}_{p+1}^t \right)^2 \right|,
\]

where \( \alpha_p^t \) and \( \bar{\alpha}_p^t \) denote, respectively, the computed and the ground truth alpha values for pixel \( p \) of frame \( t \). Likewise, \( \frac{d\alpha_p^t}{dt} \) and \( \frac{d\bar{\alpha}_p^t}{dt} \) are their corresponding derivatives considering the values of alpha at pixel \( p \) in frames \( t \) and \( (t-1) \). \( v_r^p \) denotes the motion vector at pixel \( p \), frame \( t \), computed using the optical flow method by Sun et. al [SRB10].

We compare two variants of our technique, each using a traditional alpha matting solution to initialize the matte data on the keyframes: Shared Matting (SM) [GO10] and Closed-form Matting (CF) [LLW08]. Figure 7 shows 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 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 paper and supplementary materials, one selects the keyframes based on events such as disocclusions, lighting changes, fast object motion, etc.

The graphs in Figure 7 also show that the competing techniques produce error spikes whenever a keyframe is reached. This happens because these algorithms only propagate the matte forward, leading to error accumulation. Our method, in contrast, propagates the mattes to unconstrained video frames in both directions, producing more temporally-consistent results.

Table 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 Dmitry, our SSA and MESSDdt results are one order of magnitude better than the other approaches. The last column on Table 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 (SDDA), 31% more temporally coherent (dtSSD), and 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., lowest SSDA). Please see the accompanying videos in the supplementary material for a side-by-side comparison of these results.

5.2. 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 [EGV+15] (Figure 8). 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 performed against the top four ranked techniques in the video matting benchmark [EGV+15]: Deep Matting (DM) [XPC17], Self-Adaptive Matting (SAM) [CLCH19], Learning Based Matting (LB) [ZK09], and Sparse Sampling Matting (SpSM) [KEE17]. 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
Figure 7: Comparison of the matte propagation methods under three error metrics SSDA, dtSSD and MESSDdt. Smaller values are better. AE - After Effects Rotobrush Tool [Ado19, BWSS09], MAKNN - Motion-aware KNN Matting [LCT13], SLR - Sparse Low-Rank Representation Ratting [ZCCW19], OURS+CF - Our method using Closed-form Matting [LLW08] initialization and OURS+SM - Our method using Shared Matting [GO10] initialization. Please refer to Section 5.1 for more details.
matches 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 12 shows frames from two video sequences exhibiting foreground 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.

5.4. Limitations

Since the recursive filter used to propagate the matte across the video has exponential decay, 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 11 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. Conclusion

We presented an efficient temporally-coherent matte-propagation method for videos. Our technique uses a sparse set of trimaps, re-

5.3. Videos with Complex Occlusion Patterns

Since our method depends on optical flow estimation for finding correlations between pixels in different frames, optical flow mis-

Table 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.

| Video Metric | SSDA | Alex dSSD | MESSDdt | SSDA | castle dSSD | MESSDdt | SSDA | Dmitry dSSD | MESSDdt | Total SSDA | Alex dSSD | MESSDdt |
|--------------|------|-----------|---------|------|-------------|---------|------|-------------|---------|-----------|----------|---------|
| AE           | 147.83 | 90.76 | 16.076.97 | 214.65 | 85.84 | 15.692.79 | 144.66 | 104.97 | 11.853.67 | 178.71 | 91.53 | 14.732.34 |
| MAKNN        | 121.80 | 85.49 | 12.419.27 | 150.93 | 84.90 | 12.540.72 | 140.65 | 102.90 | 19.001.01 | 140.12 | 89.20 | 14.090.91 |
| SLR          | 147.66 | 93.18 | 11.965.11 | 200.01 | 92.76 | 19.274.85 | 258.17 | 154.34 | 38.768.78 | 200.46 | 108.08 | 22.279.33 |
| OURS+CF      | 33.18 | 38.58 | 1.166.11 | 125.83 | 74.81 | 9.206.04 | 43.59 | 55.51 | 2.851.13 | 80.64 | 60.28 | 5.492.45 |
| OURS+SM      | 41.44 | 48.48 | 1.705.46 | 111.91 | 70.61 | 7.911.91 | 45.51 | 57.38 | 2.769.59 | 76.48 | 61.24 | 4.980.80 |

Figure 8: First frame of videos from the video matting benchmark [EGV*15] used in the qualitative comparison.

Figure 9: Comparing mattes generated by these methods and mattes propagated by our technique for a frame halfway between two keyframes.

Figure 10 depicts examples of temporal jittering produced by Deep Matting [XPCH17] and by Sparse Sampling Matting [KEE17]. 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. Please see the accompanying videos in the supplementary material.

The goal of our method is to propagate the matte data to the entire video using a 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 [Ado19, BWS809]. 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 9 (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 propagates 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 9).

Figure 10 depicts examples of temporal jittering produced by Deep Matting [XPCH17] and by Sparse Sampling Matting [KEE17]. 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. Please see the accompanying videos in the supplementary material.

5.3. Videos with Complex Occlusion Patterns

Since our method depends on optical flow estimation for finding correlations between pixels in different frames, optical flow mis-

© 2019 The Author(s)
Computer Graphics Forum © 2019 The Eurographics Association and John Wiley & Sons Ltd.
Figure 9: Qualitative comparison of results produced by our technique and by methods that require one trimap per frame as well as by Adobe After Effects Rotobrush Tool (AE) [Ado19, BWSS09]. DM - Deep Matting [XPCH17], SAM - Self-Adaptive Matting [CLCH19], LB - Learning Based Matting [ZK09] and SpSM - Sparse Sampling Matting [KEE17]. OURS+DM, OURS+SAM, OURS+LB and OURS+SpSM stand for our method initialized by these respective matting methods every 15 frames. AE used keyframes every 15 frames, like in our technique. These results show mattes halfway between two keyframes.
Figure 10: Comparison of temporal coherence between techniques. In these examples, Deep Matting (DM) [XPCH17] and Sparse Sampling Matting (SpSM) [KEE17] 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 supplementary material for video results.

Figure 11: Due to fast motions and foreground/background color ambiguities, the use of evenly-distributed trimaps could lead to matte artifacts. These can be avoided by repositioning the keyframes, or adding new ones.

Figure 12: 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.

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.

Acknowledgements
This work was sponsored by CNPq-Brazil (fellowships and grants 131965/2018-2, 312975/2018-0, 423673/2016-5).

References
[AAPI7] Aksoy Y., Aydin T. O., Pollefeys M.: Designing Effective Inter-Pixel Information Flow for Natural Image Matting. In Proc. CVPR (2017).
[Ado19] Adobe Inc.: Adobe After Effects, 2019. URL: https://www.adobe.com/products/aftereffects.html.
[BRFK14] Besse F., Rother C., Fitzgibbon A., Kautz J.: PMBP: PatchMatch Belief Propagation for Correspondence Field Estimation. Int. J. Comput. Vision (2014), 2–13.
[BS09] Bai X., Jeyaraj V., Jeyaraj S.: Geodesic Matting: A Framework for Fast Interactive Image and Video Segmentation and Matting. Int. J. Comput. Vision (2009), 113–132.
