A Progressive Tri-level Segmentation Approach for Topology-Change-Aware Video Matting

Jinlong Ju\textsuperscript{1,2} Jue Wang\textsuperscript{3} Yebin Liu\textsuperscript{1} Haoqian Wang\textsuperscript{2} Qionghai Dai\textsuperscript{1}

\textsuperscript{1}Department of Automation, Tsinghua University, China
\textsuperscript{2}Graduate School at Shenzhen, Tsinghua University, China
\textsuperscript{3}Adobe Research, USA

Abstract

Previous video matting approaches mostly adopt the “binary segmentation + matting” strategy, i.e., first segment each frame into foreground and background regions, then extract the fine details of the foreground boundary using matting techniques. This framework has several limitations due to the fact that binary segmentation is employed. In this paper, we propose a new supervised video matting approach. Instead of applying binary segmentation, we explicitly model segmentation uncertainty in a novel tri-level segmentation procedure. The segmentation is done progressively, enabling us to handle difficult cases such as large topology changes, which are challenging to previous approaches. The tri-level segmentation results can be naturally fed into matting techniques to generate the final alpha mattes. Experimental results show that our system can generate high quality results with less user inputs than the state-of-the-art methods.

1. Introduction

Interactive, or supervised video matting refers to the problem of extracting accurate alpha mattes of foreground objects from video sequences, guided by sparse user inputs such as scribbles. This problem has been extensively studied, and many advanced systems have been proposed for achieving high quality matting results with minimal user input [CAC\textsuperscript{02}, WBC\textsuperscript{05}, YLS\textsuperscript{05}, BWSS\textsuperscript{09}]. Most previous systems share a common strategy of first applying binary segmentation to segment each frame into foreground (\(F\)) and background (\(B\)) regions, and then employing matting techniques to refine the foreground boundary. We call this strategy “binary segmentation + matting”, which works well in many cases shown in previous work. Nevertheless, it has several major limitations, as stated below.

First, segmentation and matting is loosely coupled in this framework. Without considering the following matting procedure, binary segmentation is often too aggressive to try to label every pixel as either \(F\) or \(B\), resulting in segmentation errors that are hard to remove at the matting stage.

Secondly, since matting requires trimaps as input [WC\textsuperscript{07a}], previous systems often convert a binary segmentation into a trimap using simple heuristic operations, for instance, eroding and dilating the \(F\) region to create a narrow unknown region for matting. This is certainly not optimal when the foreground object contains both near-solid boundary parts as well as large semi-transparent regions, such as long hair strands.

Finally, recent video matting systems [BWSS\textsuperscript{09}, BWSS\textsuperscript{10}] employ local classifiers to improve the temporal segmentation stability. Localized segmentation works well when the topology of the foreground object remains consistent across frames. However it is insufficient to handle large foreground topology changes, which is common for dynamic foreground objects.

We propose a new video matting approach that addresses these limitations. The key component of our system is a novel tri-level segmentation procedure, which handles segmentation ambiguities by introducing an additional uncertain region \(U\) for each frame. The tri-level segmentation is done in a progressive refinement manner to carefully avoid introducing segmentation errors. Compared with the traditional strategy, our approach has the following advantages:

1. Improved segmentation accuracy. Due to the new tri-level
segmentation scheme, segmentation ambiguities are well handled, leading to less segmentation errors (thus less user corrections) than previous binary segmentation approaches.

2. Integrated segmentation and matting. The tri-level segmentation results can be directly fed into matting algorithms as accurate trimaps, eliminating the need to create trimaps from binary masks using heuristic rules.

3. The ability to handle large topology changes. The tri-level segmentation method is able to detect large foreground topology changes and handle them correctly. This is difficult to achieve in previous systems.

Experimental results on a wide variety of examples show that our system outperforms state-of-the-art systems by generating more accurate results with less user inputs.

2. Related Work

Video segmentation and matting has been extensively studied. In this section we discuss a few representative work what is most related to ours. A more detailed survey on this topic can be found in [WC07a].

Binary video segmentation for matting. Previous video matting systems first segment each frame into $F$ and $B$ regions, either automatically or interactively. For certain types of videos such as monocular sequences, automatic segmentation could be achieved by using bilayer segmentation methods [CCBK06, YCEW07, ZJHB11].

For complex videos with dynamic scenes and fast moving objects, accurate automatic segmentation is generally difficult. Many interactive segmentation techniques thus have been proposed for segmenting these videos, which require the user to provide sparse inputs such as scribbles to guide the segmentation. Some approaches [WBC*05, YLS05, GKH010] treat the whole sequence as a 3D volume and solve a global graph-based optimization for segmentation. However, as shown in [BWSS09], this workflow has convergence problems and in practice is not easy to use for end users. Other approaches use the frame-by-frame propagation strategy, i.e., ask the user to create accurate segmentations on some keyframes, then propagate the results from keyframes to in-between frames [CAC*02, BWSS09, BWSS10], facilitated by motion estimation and object tracking. We adopt the frame-by-frame propagation framework as it is shown to be efficient and easy-to-use for users.

Trimap generation in video. Early video matting systems [CAC*02] try to directly propagating trimaps from keyframes to intermediate frames using bi-directional optical flow, which is prone to motion estimation errors. Recent approaches [YLS05, BWSS09, BWSS10] first apply binary segmentation to all frames, then erode and dilate the $F$ region on each frame to create a uniform band for matting. This method only works well for near-solid foreground boundaries, but is not optimal when the foreground boundary has both near-solid parts (e.g., the body part of a person) and large semi-transparent regions (e.g., the hair region of a person), which requires spatially-varying trimap bandwidth. In contrast, our tri-level segmentation can generate trimaps with complex shapes, and can be directly fed into matting algorithms.

Video alpha matting. Once trimaps are generated, image matting techniques can be directly applied on a frame-by-frame basis. Representative image matting techniques that have been used in various video matting systems include Coherent matting [YLS05], Closed-form matting [LLW08], Robust matting [WC07b], and Shared matting [GO10]. However, frame-by-frame matting usually leads to temporally incoherent results. To further improve the temporal coherency, Bai et al. [BWSS09] warp the matte in the previous frame using optical flow and treat it as a strong prior for the current frame. Choi et al. introduce a multi-frame nonlocal matting Laplacian [CLT12]. Brosch et al. [BHRG12] extend the Guided filtering from images to video to postprocessing the alpha mattes. A Level-set-based filter is proposed by Bai et al. [BWSS11] for the same purpose. After tri-level segmentation is done, these advanced matting methods can be applied in our system to create high quality alpha mattes.

3. System Overview

Similar to previous approaches, our system starts by asking the user to provide an accurate alpha matte of the target object on a keyframe. This could be done efficiently using image segmentation and matting techniques. In our system we use a graph-cuts-based segmenter [BVZ01] and the learning-based matting approach [ZK09] for this purpose.

Our system then tries to propagate the alpha matte forward. On each frame, the system first generates an initial, coarse tri-level segmentation, using a per-pixel probability calculation with a carefully-designed error-avoiding strategy, as described in Sect. 4. The segmentation is further refined using a local-window-based cross-frame refinement procedure, as described in Sect. 5. Finally, the tri-level segmentation is directly fed into the learning-based matting approach as trimap for computing the final alpha matte, which is served as input to the next frame. When segmentation errors occur, the user can provide additional scribbles to refine the tri-level segmentation results, which can effectively improve the quality of the final alpha matte.

4. Tri-level Segmentation Initialization

Assuming an accurate alpha matte is generated on frame $t-1$, in this section we show how to propagate it onto frame $t$ to generate the initial tri-level segmentation result.
Figure 1: Comparing GMM color models and our sampling-based color model for initial tri-level segmentation. (a) Frame $t$. (b) Zoomed-in region of the left hand. (c) Corresponding region in frame $t-1$. (d)-(f) Segmentation results using GMM with 5, 10 and 20 components, respectively. (g) Segmentation result using our color model. (h) Zoomed-in region of the right hand. (i) Corresponding region in frame $t-1$. (j)-(l) Segmentation results using GMM with 5, 10 and 20 components, respectively. (m) Segmentation result using our color model.

4.1. Coarse Matching and The Uncertainty Map

To account for large object motion, we detect SIFT feature points [Low99] in the current frame $I'$, and track them in $I'_{t-1}$ using a standard KLT tracker [LK81]. We then use the tracked points to roughly align the two frames, and compute optical flow between two aligned frames. In this way for each pixel $x$, we can generate its cross-frame motion vector $s_x$, and the corresponding point in $I'_{t-1}$ is denoted as $x' = x + s_x$.

Next, we search in a $N \times N$ window centered at $x'$ in $I'_{t-1}$ and select $m$ foreground pixels $F = \{f_1, f_2, \ldots, f_m\}$ whose color values are the closest to $l(x)$, the color value of $x$ in RGB space. Background color sample set $B = \{b_1, b_2, \ldots, b_n\}$ can be obtained in the same way. The foreground color probability of $x$ is defined as:

$$p_f(x) = \exp \left( -\frac{\sum_{i=1}^{m} \|l(x) - l(f_i)\|^2}{m \cdot \delta} \right),$$

where $\delta$ is the color sensitivity parameter which we fix at 0.015 (assuming $l(\cdot) \in [0, 1]$), and $m$ is set to be 10 in the system. Similarly, the background color probability of $x$ is:

$$p_b(x) = \exp \left( -\frac{\sum_{i=1}^{m} \|l(x) - l(b_i)\|^2}{m \cdot \delta} \right).$$

The initial normalized foreground color probability $p(x)$ is then defined as:

$$p(x) = \frac{p_f(x)}{p_f(x) + p_b(x)}.$$  

It is also worth noting that many previous approaches [WBC05, YLS05, BWSS09] use parametric models, such as Gaussian Mixture Models (GMMs) to describe local color distributions. While in our system, for measuring color probability, we measure the color distance from the target pixel to a small number of closest color samples, as shown in Eqn. 1 and 2. This is because when the foreground and background color distributions are complex, simple Gaussians are insufficient to model the higher order statistics of the color distributions. In practice we found that a small number of samples can estimate the foreground probability for an unknown pixel better, as shown in previous sampling-based image matting research [WC07a]. Such an example is shown in Fig. 1.

With the computed initial foreground color probability $f(x)$, we assign $x$ to one of the four labels as:

$$u(x) = \begin{cases} UM, & p_f(x) < \varepsilon \& p_b(x) < \varepsilon \\ \mathcal{F}, & \text{else if } p(x) > t_h \\ \mathcal{B}, & \text{else if } p(x) < t_l \\ \mathcal{U}, & \text{else} \end{cases}$$

In Eqn. 4, the four labels $UM, \mathcal{F}, \mathcal{B}$ and $\mathcal{U}$ refer to unmatched, foreground, background and uncertain pixels, respectively. We set $\varepsilon = 0.05$, $t_h = 0.9$ and $t_l = 0.1$ in our system. It is important to note that, if the color similarity of a pixel to the foreground and background samples are both smaller than $\varepsilon$, meaning that we could not find good foreground and background colors to match with the current pixel, we will classify it as $UM$ instead of forcing to compute a probability, which may be erroneous. In many cases, these unmatched pixels are caused by internal topology-changes of a fast moving object. For example, in Fig. 2, the red region in image $I'$ is a newly appeared background area.
Pixels in this region cannot be matched to any nearby foreground pixels in \( I^{t-1} \), making \( p_f(x) \) extremely low. Such topology change is therefore successfully captured by our method. We will further label these unmatched pixels in the next step of the system described in Sect. 4.2.

Our segmentation strategy significantly differs from recent local-classifier-based systems [BWSS09, BWSS10], where the foreground contour computed in the previous frame is warped to the current frame using optical flow, and only pixels around the warped contour are considered for classification. However, when topology changes happen inside the foreground object, segmentation errors will occur due to the fact that the narrow segmentation band does not cover those pixels (see Fig. 2). In our system all pixels on the target frame are classified, thus topology changes inside the foreground or background regions can be detected.

4.2. Dealing with Unmatched Pixels

Unmatched pixels in \( \mathcal{UM} \) occur in the initial segmentation when motion estimation fails to track the object, or the foreground object changes its topology at the current frame. To further evaluate the foreground probability for these pixels, we return to color modeling in a larger sampling area, by jointly considering pixels in a narrow band (the width is set to be 10 pixels in our system) around the current foreground boundary, and pixels in the original \( N \times N \) window. GMM is adopted to calculate the foreground and background likelihood \( p(x|F) \) and \( p(x|B) \). The foreground probability of \( x \) is then re-estimated as:

\[
p(x) = \frac{p(x|F)}{p(x|F) + p(x|B)}. \tag{5}
\]

We assign \( x \) to \( F \) if \( p(x) > t_f \), to \( B \) if \( p(x) < t_l \), or \( U \) otherwise. Note that here we use GMM model instead color sampling, as in this step some color samples are spatially far away from \( x \) and thus are not reliable. By relabeling every pixel in \( \mathcal{UM} \), we remove the entire \( \mathcal{UM} \) region and produce a complete tri-level segmentation, as shown in the example in Fig. 2.

4.3. Local Correction

Till now, each pixel is labeled independently, which may lead to noise in the obtained initial segmentation. We therefore propose a local correction strategy to remove noise and small errors based on pixel clustering. To identify coherent image regions, the mean-shift algorithm [CM02] is applied on the input image \( I^t \) to merge nearby pixels with very similar colors into small segments. Since the clustering is conservative because segments are very small, it is safe to assume no foreground and background pixels are merged together to form a small segment. With this assumption, we refine the labels of pixels in each segment using its spatial neighborhood defined by the segment.

Since we assume that pixels in the same segment should belong to the same object, errors are detected if there are conflict foreground and background labels in one segment. For instance, the blue region in Fig. 3(b) illustrates such a conflict. The region marked with red circle in Fig. 3(c) is a segment where error is detected. We found that the inconsistency is mostly caused by the fact that different pixels are using different color sample sets in Eqs. 1 and 2 to compute their probabilities. To improve segmentation in this case, we re-label the pixels in an erroneous segment by:

1. creating a large foreground and background sample set, by combining the small sample sets for individual pixels in the segment together;
2. for each pixel \( x \), recalculating its foreground probability \( p(x) \) from the new sample sets using Eqs. 1, 2, and 3;
3. re-labeling \( u(x) = F \) if \( p(x) > t_f \), or \( u(x) = B \) if \( p(x) < t_l \), or otherwise \( u(x) = U \).

In the local correction process, since all pixels are using the same set of color samples, the segmentation consistency is greatly improved. Furthermore, we set \( t_f = 0.95 \) and \( t_l = 0.05 \) in Step 3, which are tighter constraints to meet than the thresholds in Eqs. 4, making the relabeling to be more conservative. It simply classifies more pixels as \( U \), thus can effectively avoid making mistakes in ambiguous cases, as shown in Fig. 3(d).

5. Tri-level Segmentation Refinement

After the above steps, we obtain a tri-level segmentation map which contains \( F, B \) and \( U \) regions. In simple cases where the local foreground and background color distributions are well-separable, the initial segmentation is already accurate enough to be used as the trimap for image matting. However in more complicated cases, there will still be a substantial number of pixels that are labeled as \( U \), which need further process before image matting, as a loosely-defined trimap
will inevitably lower the performance of existing image matting techniques (see the hair and foot region in Fig. 4(a)). In this section we show how to incorporate local shape and color priors for achieving it.

5.1. Cross-frame Window Matching

Inspired by the local classifiers proposed by Bai et al. [BWSS10], given the initial tri-level map, we create a set of local windows \( \{W_1, W_2, ..., W_n\} \) to cover all pixels in the \( \mathcal{U} \) region, as shown in Fig. 4(a). Each pixel in \( \mathcal{U} \) is covered by at least one window.

Next, we aim to find a window \( \tilde{W}_i \) in frame \( f' \) that best matches with a window \( W_i \) (\( i = 1, 2, ..., n \)) in frame \( f \). For stability as well as efficiency, we constrain the searching range to be local for each window. Since we have a flow vector \( s_t \) from \( f \) to \( f' \) for each uncertain pixel \( x \) in \( f \), we average the flow values of pixels in \( W_i \) to get the average flow vector \( v_i \). Suppose \( c_j \) is the center position of the window \( W_i \), then a searching window \( \bar{W}_i \) centered at position \( c_j = c_i + v_i \) on frame \( f' \) is constructed, as shown in Fig. 4(c). Due to the fact that fast moving objects may introduce higher uncertainty for matching, we let the edge length of window \( \bar{W}_i \) grow linearly with the intensity of the estimated motion, as:

\[
L_{\bar{W}_i} = \sqrt{s_t} \cdot (v_i + L_{W_i}).
\]

We search in \( \bar{W}_i \) using a sliding window with the same size as \( W_i \), and select the window \( \bar{W}_i \) that minimizes a designed objective function. Considering that the object is moving against the background, the similarity of background colors in \( W_i \) and \( \bar{W}_i \) are usually low. We thus measure the distance between two windows by minimizing the following objective function which only involves the \( \mathcal{F} \) regions in the two windows:

\[
f(\bar{W}_i) = \text{EMD} \left( H_{W_i} \otimes M_{f'}(W_i), H_{\bar{W}_i} \otimes (M_{f'}(W_i) \odot M_{f}(\bar{W}_i)) \right),
\]

where function \( M_{f'} \) defines the foreground mask for an input window, so that \( M_{f'}(x) = 1 \) if \( x \) is labeled as \( \mathcal{F} \), and 0 otherwise. Examples of \( M_{f'}(W_i) \) and \( M_{f'}(\bar{W}_i) \) are shown in Fig. 4(d) and 4(e), respectively. \( H \) is the histogram operation, and "\( \odot \)" refers to the Hadamard product of two matrices. The Hadamard product produces a matrix shares with the same size with the two original matrices, and with each element equals to the product of the two corresponding elements in the original matrices. \( \text{EMD} \) is a function measuring the similarity of two histograms using earth mover's distance (EMD) [RTG98], which reflects the minimal amount of work that must be performed to transform one distribution into the other. Intuitively, if foreground regions in \( W_i \) and \( \bar{W}_i \) have roughly the same shape, size and position, then \( f(\bar{W}_i) \) will be low, indicating a good match. Such an example is shown in Fig. 4(b).

5.2. Local Window Refinement

With the help of the matched window \( \tilde{W}_i \), we further define local classifiers to refine the segmentation in window \( W_i \). We first define a foreground shape similarity metric between \( \tilde{W}_i \) and \( W_i \), which measures if the local object shape is consistent across the two frames. Denote the current foreground probability map in \( W_i \) as \( p_f(W_i) \), we normalize it using the sum of all pixel values in \( p_f(W_i) \) to treat it as a probability distribution \( p(W_i) \). Similarly, we generate \( p(\tilde{W}_i) \) for \( \tilde{W}_i \), as its alpha matte is known. The shape similarity between the two windows is computed as:

\[
f_{\text{shape}}(i) = B_d \left( p(W_i), p(\tilde{W}_i) \right),
\]

where \( B_d \) stands for the Bhattacharyya distance [Bha43], which is commonly used for measuring the distance between two discrete probability distributions \( p \) and \( q \) over the same domain \( \chi \):

\[
B_d(p_1, p_2) = -\ln \left( \sum_{x \in \chi} \sqrt{p_1(x)p_2(x)} \right).
\]

A high \( f_{\text{shape}} \) value indicates that the local contour shape is consistent across the two frames, thus we can treat \( M_{f'}(\tilde{W}_i) \), the foreground shape in the previous frame as a strong prior for segmenting \( W_i \). On the other hand, if \( f_{\text{shape}} \) is low, we do not trust the shape prior \( M_{f'}(\tilde{W}_i) \), and rely on known foreground and background colors in \( W_i \) to segment \( W_i \).

Specifically, similar to Section 4.1, to compute the foreground probability of \( x \) in \( W_i \), we sample a set of known foreground and background colors \( p(W_i) \), denoted as \( \{a_1, a_2, ..., a_s\} \) and \( \{b_1, b_2, ..., b_b\} \), respectively. In our implementation we set \( n = 10 \). Note that the samples are on previous frame \( t-1 \), not on the current frame. To select the best samples, we define a similarity measure between \( x \) and a sample pixel \( y \) by jointly considering their spatial and color

\[
\text{sim}(x, y) = \frac{\text{EMD}(h(x), h(y))}{\text{max}(h(x), h(y))},
\]

where \( h(x) \) is the color histogram of pixel \( x \), and \( h(y) \) is the color histogram of pixel \( y \). The EMD is a function measuring the similarity of two histograms using earth mover's distance (EMD) [RTG98], which reflects the minimal amount of work that must be performed to transform one distribution into the other. Intuitively, if foreground regions in \( W_i \) and \( \tilde{W}_i \) have roughly the same shape, size and position, then \( f(\tilde{W}_i) \) will be low, indicating a good match. Such an example is shown in Fig. 4(b).
distances as:

\[ f_{xy} = f_{shape}^F \cdot d_f(x, y) + (1 - f_{shape}^F) \cdot d_c(x, y), \]  

(10)

where \( d_c \) is based on the spatial distance between \( x \) and \( y \), defined as:

\[ d_c(x, y) = 1 - \|x - y\|_2, \]  

(11)

and

\[ d_f(x, y) = \exp(-\frac{\|l(x) - l(y)\|_2}{\delta}). \]  

(12)

Here, \( \vec{x} \) is the relative position of \( x \) respect to the center of the window \( W_i \), normalized into \([-0.5, 0.5]\) by the width of \( W_i \) as computed in Eqn. 6. Similarly, \( \vec{y} \) is the normalized relative position of \( y \) to the center of \( W_i \). Intuitively, if \( f_{shape}^F \) is close to 1, then \( d_f(x, y) \) plays the major role on determining the similarity. That is, if \( x \) is spatially close to an existing foreground pixel, it has a high probability of being foreground, given that the two windows are aligned. On the other hand, if \( f_{shape}^F \) is low, we then mainly rely on using the pixel colors for computing the similarity, without considering the spatial locations of \( x \) and \( y \). Thus, the shape prior is encoded in \( d_f(x, y) \).

To select a small set of best foreground examples, we examine all foreground pixels in \( W_i \) and select the top \( n \) samples which have the largest \( f_{xy} \) values. Background samples are selected in the same way. We then can use these samples to compute a foreground similarity and background similarity measure as:

\[ p_f(x) = \frac{1}{n} \sum_{j=1}^{n} f_s a_j, \quad p_b(x) = \frac{1}{n} \sum_{j=1}^{n} f_s b_j. \]  

(13)

One more thing we need to consider is that the background samples \( b_j \), which are selected from \( W_i \) on frame \( t - 1 \), may not well represent the true background colors in \( W_i \) on frame \( t \). This is due to the fact that in video segmentation, we often assume local foreground colors to be consistent over time as we only track the foreground object. The background colors may change quickly for fast moving objects. So to find out whether there is a significant background color change between \( W_i \) and \( W_t \), we define a background color distance measure as:

\[ d^B_{\text{color}} = \text{EMD}\left(H_{W_i \oplus M_b(W_i)}(W_t), H_{W_i \oplus M_b(W_t)}(W_t)\right). \]  

(14)

Intuitively, in both windows, we build a background color histogram of pixels which are currently labeled as background, and compute the distance of two histograms using earth mover’s distance. If the background colors in these two windows are similar, then the distance \( d^B_{\text{color}} \) is low.

Using the background color distance measure, the new foreground probability of pixel \( x \) is computed as:

\[ p(x) = \left\{ \begin{array}{ll} \frac{p_f(x)}{p_f(x) + p_b(x)} & d^B_{\text{color}} < 0.5 \\ p_f(x) & d^B_{\text{color}} \geq 0.5 \end{array} \right. \]  

(15)

Basically, when the background color change is detected \( (d^B_{\text{color}} \geq 0.5) \), we only trust foreground samples and do not use background samples selected from the previous frame. The tri-level segmentation label \( u(x) \) is also updated accordingly. The refined tri-level segmentation results are then fed into the learning-based matting algorithm [ZK09] to generate the final alpha mattes. Fig. 5 illustrates the performance of local window refinement.

6. Experimental Results

6.1. Implementation Details

We implemented our method using C++. Our testing environment is a PC running MS Windows 7 64bit version with Intel Core i5 CPU and 8GB RAM, however we only use a single CPU thread. For a 960 × 540 frame, segmentation initialization (Sec. 4) costs 0.9 to 2.5 seconds depending on the size of the foreground object. The computational time for segmentation refinement (Sec. 5) also depends on the complexity of the scene, but it is well under one second. The mean-shift package we use costs 3 seconds per-frame, but it is done as preprocessing. We have tested the system on a number of videos, some are shown in this section, and more are shown in the supplementary material.

![Figure 5: Examples of segmentation refinement.](image-url)
6.2. Qualitative Comparisons

In order to demonstrate the advantages of the system on dealing with complicated color distributions and topology changes, we compare our system with the recent video SnapCut system [BWSS09], as shown in Fig. 6. For each method, we show the final results as well as user inputs. These examples clearly show that the SnapCut system often requires additional user input when the foreground topology changes. In comparison, our system achieves higher quality results with much less user efforts.

To illustrate how our system handles large semi-transparent regions, Fig. 8 shows two successive frames of the “hair” video. Note that the foreground object has very complicated motion, as the girl shakes her head and her hair waves irregularly. Methods based on binary segmentation are not able to generate good results, as there are a large number of semi-transparent pixels in the hair region. This is not a problem for our system as those pixels are naturally labeled as uncertain in the tri-level segmentation, and correctly handled by the matting algorithm, as shown in Fig. 8(b).

6.3. Quantitative Comparisons

For quantitatively comparing the matting accuracy, we collect a few sequences and manually segment them by hand. We then treat the first frame as the only keyframe, and run our system and the SnapCut system automatically on other frames without any user interaction. We define the error rate as the mean number of misclassified pixels of all frames, di-
Table 1: Comparisons of error rates of different methods.

<table>
<thead>
<tr>
<th></th>
<th>Ours</th>
<th>SnapCut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Basketball</td>
<td>2.17%</td>
<td>4.34%</td>
</tr>
<tr>
<td>Akko</td>
<td>1.12%</td>
<td>2.63%</td>
</tr>
<tr>
<td>Dancer</td>
<td>0.69%</td>
<td>1.71%</td>
</tr>
<tr>
<td>baseball</td>
<td>1.32%</td>
<td>2.87%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No refinement</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

vided by the number of foreground pixels. The results are shown in Table 1, and the “dancer” example is shown in Fig. 7. It suggests that our system produces less segmentation errors than the previous system.

Furthermore, we compare our complete system with a simplified version by disabling the segmentation refinement procedure described in Sec. 5 in Table 1, where the results of the simplified version are shown in the column named as “no refinement”. It demonstrates that the segmentation refinement is very important for achieving accurate results.

7. Conclusion, Limitation and Future Work

We present a new progressive tri-level segmentation procedure for video matting. By introducing a new uncertain label in the segmentation procedure and a progressive segmentation scheme, our approach generates less errors compared to binary segmentation, and can be naturally combined with alpha matting techniques for generating high quality mattes. We build an interactive video matting system based on the new segmentation method and demonstrate its effectiveness through both qualitative and quantitative comparisons.

As for limitations, similar to most previous methods, we assume that the color statistics of the foreground object is stable across time. Sudden illumination changes can lead to errors in multiple steps of the system. Figure 9 shows such an example. Although this can be handled by additional user input, we would like to explore automatic approaches to robustly handle illumination changes.

With the default parameter setting, the alpha mattes generated by our current system tend to be slightly sharper, or more binarized than desired when dealing with foreground objects with large semi-transparent regions, which can be seen from Fig. 8. This is because a single set of thresholds (e.g. \( t_h \) and \( t_l \) in Eqn. 4, \( t_h' \) and \( t_l' \) in Sec. 4.3) cannot handle diversity examples very well. In the future we plan to add an additional parameter on the UI to allow the user to control the smoothness of the alpha mattes.

Finally, our system relies on existing image matting techniques to generate the final results, thus it is limited by these methods. Particularly, when the foreground and/or background regions are highly-textured, existing matting methods often fail to produce satisfactory results. Better matting techniques are desired to generate good results in these cases.

Acknowledgements

This work was supported by The National High Technology Research and Development Program of China (863 Program) No.2013AA01A604 and the Project of NSFC (No.60932007 and No.U0935001).

References


