Video Object Cut and Paste

Yin Li   Jian Sun   Heung-Yeung Shum

Microsoft Research Asia

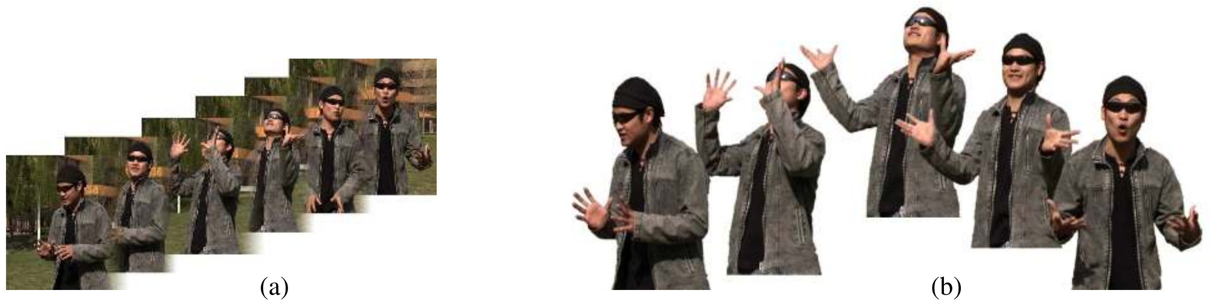

Figure 1: Given a video (a) containing an opaque video object on a complicated background, our system can generate clear alpha mattes and foreground colors (b) for the video object, which can be pasted onto another background.

Abstract

In this paper, we present a system for cutting a moving object out from a video clip. The cutout object sequence can be pasted onto another video or a background image. To achieve this, we first apply a new 3D graph cut based segmentation approach on the spatial-temporal video volume. Our algorithm partitions watershed pre-segmentation regions into foreground and background while preserving temporal coherence. Then, the initial segmentation result is refined locally. Given two frames in the video sequence, we specify two respective windows of interest which are then tracked using a bi-directional feature tracking algorithm. For each frame in between these two given frames, the segmentation in each tracked window is refined using a 2D graph cut that utilizes a local color model. Moreover, we provide brush tools for the user to control the object boundary precisely wherever needed. Based on the accurate binary segmentation result, we apply coherent matting to extract the alpha mattes and foreground colors of the object.

Keywords: Video Segmentation, Matting, Tracking, Graph Cut

1 Introduction

Cut and paste of moving objects in a video sequence has many applications in video processing. Typically, this is performed by chroma keying, which is also referred to as blue screen matting. In chroma keying, foreground objects are video recorded in front of a solid-colored background, usually blue or green, and then are separated from the background using matting techniques such as [Smith and Blinn 1996] that take advantage of the known background color. The simplicity of these techniques enables rapid foreground separation. For example, using the Ultimate® system, chroma keying can be computed in real time. However, these methods are limited to simple backgrounds of a solid color. Errors often occur when foreground objects contain colors similar to the background.

Previous approaches for video object cutout involve silhouette tracking, such as in [Kass et al. 1987; Blake and Isard 1998; Agarwala et al. 2004; Drori et al. 2004]. Although these methods can be applied to general backgrounds, object boundaries are imprecisely represented by smooth curves for greater robustness in the tracking process. Since a coarse boundary descriptor cannot capture the fine details of a silhouette, these techniques are inadequate for cut and paste applications. Rough boundaries could be interactively refined by auto keying [Mitsunaga et al. 1995], which provides a user interface for detailed boundary adjustment by spline editing. However, since each video frame must be individually modified by the user, a prohibitive amount of manual work would be required to properly delineate the boundary details.

Recently, video matting techniques (e.g., [Chuang et al. 2002; Apostoloff and Fitzgibbon 2004]) have relaxed the solid color background requirement to allow smooth color changes. The success of video matting depends on how accurately the trimaps can be propagated and how well Bayesian matting [Chuang et al. 2001] performs in each individual frame. Thus, video matting has two main difficulties for general video sequences. First, many videos contain fast motions, deforming silhouettes, and often-changing topologies, which are very challenging for the state-of-art optical flow algorithm [Black and Ananda 1996] to bidirectionally propagate trimaps as shown in Figure 2(c). Second, even if accurate trimaps can be obtained with considerable user interaction, the Bayesian matting technique often produces unsatisfactory results when the foreground/background contains complex textures or the foreground colors are similar to the background colors. An example of this problem is shown in Figure 2(e).

In this paper, we propose a practical system for video object cut and paste from general backgrounds. We obtain a binary segmentation of the video objects using a two-step approach: a novel 3D graph cut based segmentation followed by a new tracking-based local refinement. Then we adopt coherent matting [Shum et al. 2004] which uses the binary segmentation as a prior to produce the alpha matte of the object.

Our approach has the following advantages. First, we generate an accurate binary segmentation before we apply the coherent matting. Therefore, coherent matting can generate better results than Bayesian matting because it fully exploits the information in the binary segmentation with a regularization term for the alpha matte, as shown in Figure 2(f). Moreover, to obtain a binary video segmentation, our system provides more accurate results and an easier-to-use

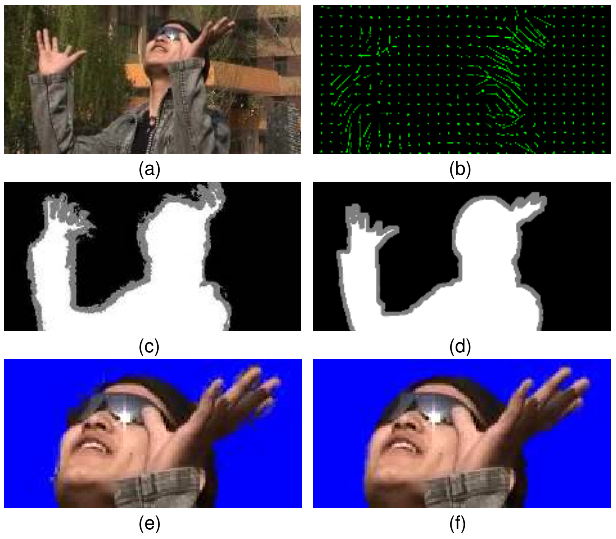

Figure 2: Coherent matting vs. Bayesian matting.

(a) The 29th frame in clip #4 from the accompanying video.

(b) The optical flow from the 28th frame. (each vector is multiplied by 2 for better visualization.)

(c) The trimap generated by the optical flows from two accurate trimaps in the 28th and 30th frames by following the approach in [Chung et al. 2002], which appears too coarse for matting.

(d) The accurate trimap obtained from accurate binary segmentation.

(e) Even with the accurate trimap (d), Bayesian matting produces a fuzzy result because of the low contrast boundary (such as the black colors around the head) and complicated background textures (such as the trees in background). Even worse, these artifacts may cause flickering across frames.

(f) The result produced by our approach and coherent matting shows a clearer and more stable boundary.

UI for refinement than contour tracking or trimap propagation. Recent interactive 2D image segmentation methods [Li et al. 2004; Rother et al. 2004] have demonstrated that accurate object boundaries can be easily obtained using simple user interaction and the graph cut algorithm [Boykov and Jolly 2001]. In this paper, we further extend the pixel-level 3D graph cut proposed by [Boykov and Jolly 2001] to the region-level 3D graph cut to handle video objects (Section 3), and we also provide a local refinement method using tracking (Section 4).

2 Overview

The framework of our system is illustrated in Figure 3. The user first selects a few key frames in the video sequence and provides their precise foreground/background segmentation using any existing image snapping tool, such as from [Li et al. 2004]. Key frames are typically sampled at ten-frame intervals, but the sampling rate may vary according to object motion. For slower moving or deforming objects, a lower sampling rate may be used.

Between each pair of successive key frames, a 3D graph is built on atomic regions (obtained with pre-segmentation) instead of individual pixels [Boykov and Jolly 2001; Kwatra et al. 2003]. A novel 3D graph cut based segmentation is then performed by considering the color consistency of each region with the foreground/background color distribution in key frames, and then maximizing the color differences between regions across the object boundary. In addition, it embeds temporal coherence of the video object in the optimization. Much of the object silhouette can be accurately located by this 3D graph cut segmentation.

To correct errors caused by the global nature of the color models used in the above 3D segmentation, our system allows the user

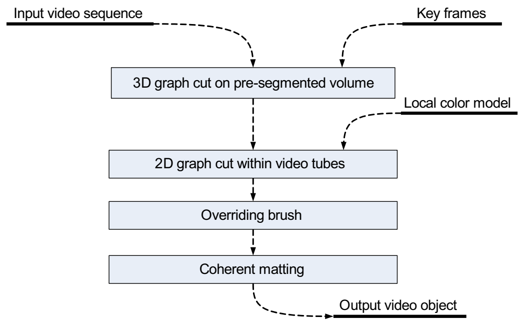

Figure 3: The framework of our system

to refine the segmentation results in local windows across frames, which we refer to as *video tubes*. These tubes are extracted by bi-directional feature tracking of windows positioned by the user. The segmentation of sections of these tubes is recomputed using local color models and 2D graph cut.

For those regions that are very difficult for automatic segmentation, e.g., when color changes are subtle or edges are ambiguous, our system allows the user to override the segmentation mask by brushing over the foreground and background.

Finally, the video object is cut out by applying coherent matting within a trimap that is generated by dilating the binary segmentation boundary. The alpha matte as well as the foreground colors are produced for the cut-out object sequence, which can be directly pasted onto another video or image background.

3 3D graph cut segmentation

Our 3D graph cut segmentation algorithm is applied on the spatial-temporal volume of the video. To make the optimization process tractable, we pre-segment each frame in the video into a number of atomic regions using the watershed algorithm [Vincent and Soille 1991] and build the 3D graph based on these atomic regions. An alternative pre-segmentation algorithm is tobogganing [Mortensen and Barrett 1999]. The novelty of our approach lies in the way we form temporal connections that preserve a set of candidates and therefore embed temporal consistency without explicit motion estimation.

3.1 3D graph construction

The video object segmentation problem can be viewed as a labeling problem, where each region in the video is assigned a unique label, $x \in \{1(\text{foreground}), 0(\text{background})\}$. The regions in key frames already have labels, while regions in other frames are to be assigned labels by the 3D graph cut segmentation algorithm.

We construct a 3D graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{A} \rangle$ on a 3D volume bounded by two successive key frames. The node set $\mathcal{V}$ contains atomic regions generated by the watershed pre-segmentation algorithm. The arc set $\mathcal{A}$ contains two kinds of arcs: intra-frame arcs $\mathcal{A}_I$ connecting nodes within one frame, and inter-frame arcs $\mathcal{A}_T$ connecting nodes across adjacent frames.

To construct the intra-frame arcs $\mathcal{A}_I$, we simply connect each region r_t to each of the adjacent regions in the same frame I_t. To construct the inter-frame arcs $\mathcal{A}_T$, we connect each region r_t to each region in the adjacent frame $I_{t\pm1}$ that lies within a given radius[1] (typically 15 pixels), excluding obviously unrelated regions whose mean color differs from that of region r_t by more than a threshold T_c (typically 30). We keep a set of candidate connections for possible correspondences on adjacent frames, and let graph cut optimization decide which should be cut off. This strategy leads to

[1]To handle regions with various shapes, such as an "L" shape or thin and long regions, the adjacency between regions is computed by morphological dilation instead of Euclidean distance between region centers.

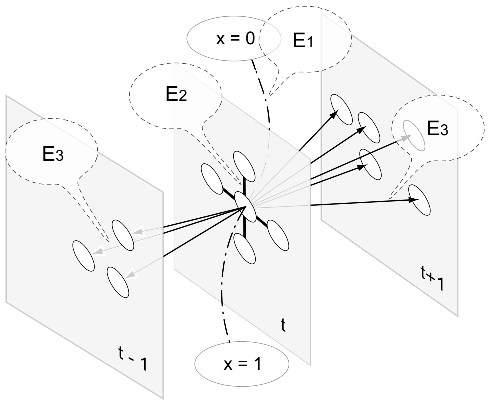

Figure 4: 3D graph cut construction. For a region r, it contributes to 3D graph construction in three ways. First, it connects to the foreground and background virtual nodes according to an energy term E_1. Second, it connects to neighboring regions within a frame with term E_2. Last, it connects to candidate regions on adjacent frames with term E_3.

greater robustness than traditional tracking methods, which determine only one correspondence.

3.2 3D graph cut optimization

The 3D graph cut algorithm solves the labeling problem by minimizing the following energy function defined on the 3D graph $\mathcal{G}$:

$$E(X) = \sum_{r\in\mathcal{V}} E_1(x_r) + \lambda_1 \sum_{(r,s)\in\mathcal{A}_I} E_2(x_r,x_s) + \lambda_2 \sum_{(r,s)\in\mathcal{A}_T} E_3(x_r,x_s) \tag{1}$$

where x_r is the foreground/background label of region r, and $X = \{x_r : \forall r\}$. The first term E_1 measures the conformity of the color of region r to the foreground/background color model built from the color information in the key frames. The second term E_2 measure color differences between two adjacent regions in the same frame, and encourage two similar adjacent regions to be both within the foreground or in the background. The third term E_3 measures color differences between two adjacent regions in two adjacent frames, and embeds temporal coherence in the graph cut optimization process through intra-frame arcs $\mathcal{A}_T$.

Likelihood energy E_1 The foreground/background color models for E_1 are built by sampling the colors in these key frames. Gaussian mixture models (GMMs) are used to describe the foreground/background color distributions. The mth component of the foreground GMMs is denoted as $(w^f_m, \mu^f_m, \Sigma^f_m)$, representing the weight, the mean color and the covariance matrix. We use M components to describe the foreground or background colors, hence $m \in [1, M]$. Typically $M = 6$.

For a given color c, its distance to the foreground GMMs is defined as,

$$d^f(c) = \min_{m\in[1,M]} \left[\hat{D}(w^f_m, \Sigma^f_m) + \bar{D}(c, \mu^f_m, \Sigma^f_m)\right], \tag{2}$$

where

$$\hat{D}(w, \Sigma) = -\log w + \frac{1}{2}\log\det\Sigma, \tag{3}$$

and

$$\bar{D}(c, \mu, \Sigma) = \frac{1}{2}(c-\mu)^T\Sigma^{-1}(c-\mu). \tag{4}$$

For a region r, its distance to the foreground GMMs is defined as the expectation of the distance of all pixels inside the region, denoted as $\langle d^f\rangle_r$. The distance $\langle d^b\rangle_r$ to the background color is defined similarly. Then, the likelihood energy $E_1(x_r)$ is defined as:

	$r \in \{F\}$	$r \in \{B\}$	$r \notin \{F\}\cup\{B\}$
$E_1(x_r = 1)$	0	∞	$\langle d^f\rangle_r$
$E_1(x_r = 0)$	∞	0	$\langle d^b\rangle_r$

$\{F\}$ and $\{B\}$ are sets of foreground regions and background regions, respectively, in key frames, whose labels are inputs. Assignments of 0 and ∞ to E_1 enforce these hard constraints in the optimization.

Prior energies E_2 **and** E_3 These two energies are defined with respect to color similarity between two regions r and s as follows:

$$E(x_r, x_s) = |x_r - x_s| \cdot e^{-\beta\|c_r - c_s\|^2}, \tag{5}$$

where $\|c_r - c_s\|$ is the L_2 norm of the RGB color difference. β is a robust parameter that weights the color contrast, and can be set to $\beta = \left(2\langle\|c_r - c_s\|^2\rangle\right)^{-1}$ [Blake et al. 2004], where $\langle\cdot\rangle$ is the expectation operator. β is computed separately for E_2 and E_3. Note that the factor $|x_r - x_s|$ allows this energy to be considered only for connections across the segmentation boundary. The prior energy E_2 and E_3 are penalty terms when adjacent nodes are assigned with different labels.

The objective function of Equation (1) can be globally minimized by an efficient graph cut algorithm ([Boykov and Jolly 2001]) and the resulting labels for each node determine a segmentation in the video volume. The construction of the 3D graph is illustrated in Figure 4. Note that in the 3D graph construction, the edge cost of the arc to virtual foreground (background) node in the graph is $E_1(0)$ ($E_1(1)$), and the edge cost of the intra-frame or inter-frame arc is $e^{-\beta\|c_r - c_s\|^2}$. The arc between nodes that have similar colors (c_r and c_s) should have high cost.

The default parameters are fixed to $\lambda_1 = 24, \lambda_2 = 12$ in all of our experiments. The 3D graph cut segmentation algorithm can compute the video object boundary well at a reasonable speed.

4 Local refinement by tracking

Since the foreground/background color distributions are built globally from the key frames, the 3D graph cut segmentation result can be poor in areas where the foreground color matches the background color of a different part of the video, and vice versa. In this section, we introduce a tool which allows the user to specify short and localized video tubes where only local color models are used in graph cut segmentation. By isolating local colors, the segmentation boundary can be improved significantly.

A video tube consists of rectangular windows $\{W_t\}^T_{t=1}$ across T frames. To specify a video tube, the user only needs to place two key windows W_1 and W_T. The remaining windows are automatically located by a bi-directional feature tracking algorithm. There are two requirements for specifying a video tube: 1) at least one key frame is in between W_1 and W_T such that local foreground/background color models can be obtained for refinement, 2) the tube boundary must be correct at the segmentation borders, since the intersection points provide hard constraints in the optimization.

After tracking is performed, a constrained 2D pixel-level graph cut segmentation is applied to each window individually using the local foreground and background color models constructed from the windows in the key frames. Finally, the refined segmentation result in each window is seamlessly connected to the existing boundary outside the window.

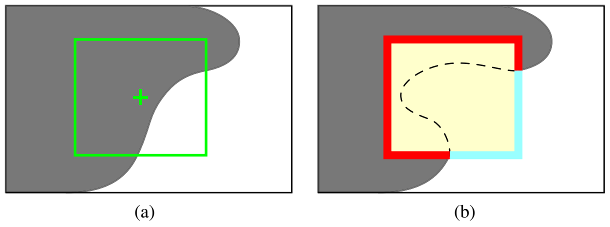

Figure 5: (a) A window of a video tube placed on a boundary of an existing segmentation result. (b) A 2D graph cut segmentation is constructed. The outermost pixels are labeled as foreground/background hard constraints according to the existing segmentation result, and all inside pixels are uncertain. The graph cut segmentation result (shown as a dashed line) is used to replace previous segmentation boundary.

4.1 Bi-directional feature tracking

Given two key windows W_1 and W_T, our algorithm tracks the position of the window in the intermediate frames. The sizes of W_1 and W_T can be different and adjusted by the user. Before tracking, the windows in between are linearly interpolated (both position and size) from W_1 and W_T.

We denote p_t as the center position of each window W_t in the video tube. We also define a search range S_t for the position of each window. All positions $\{p_t\}_{t=2}^{T-1}$ of windows can be solved by minimizing the following objective function:

$$\{p_t^*\} = \arg\min_{\{p_t\}} \sum_{t=2}^{T-1} \min(D(p_t, p_1), D(p_t, p_T)) +$$
$$\sum_{t=2}^{T} \{\eta_1 \|(p_t - p_{t-1}) - (\widehat{p}_t - \widehat{p}_{t-1})\| + \eta_2 D(p_t, p_{t-1})\}, \quad (6)$$

where $D(p_{t1}, p_{t2})$ is the sum of squared color distances between two windows W_{t1} and W_{t2} in their overlapping region when their centers p_{t1} and p_{t2} are aligned. $\widehat{p}_{t-1}$ and $\widehat{p}_t$ are the positions of windows W_{t-1} and W_t before optimization, which is computed by linear interpolation. $\eta_1 = 0.1$ and $\eta_2 = 1$ are used in all our experiments.

The first term in equation (6) is designed to optimize the color consistency of the window with respect to the key windows. We choose the best matching key window to compute this cost, to allow for feature changes over time. The second term enforces the smoothness of the video tube. The third term is for minimizing the color differences between adjacent windows. Note that the positions of key windows are fixed in the optimization process, since they have been placed by the user. We refer to this tracking method as "bi-directional" tracking because each window receives information from two key windows in two directions.

This objective function can be optimized using the dynamic programming (DP) algorithm [Bellman 1957]. In our system, a multiscale method is used for the optimization. First, a Gaussian pyramid is built for each frame in the video, and each higher level has half the frame size of its immediate lower level. The window's position and size are scaled accordingly. We perform optimization at each level beginning from the top of the pyramid, within the search range S_t centered at the optimized location in the preceding level. For the top level, the initial position of W_t is linearly interpolated from the key windows. Typically, for an NTSC video (720×480) there are $L = 4$ levels and S_t is a 7×7 square window at each level for our experiments.

To view this tracking process, please refer to the accompanying video. Although the appearances within some windows may change over time, our optimization algorithm performs well and locates the windows that comply with the requirements of the local refinement process.

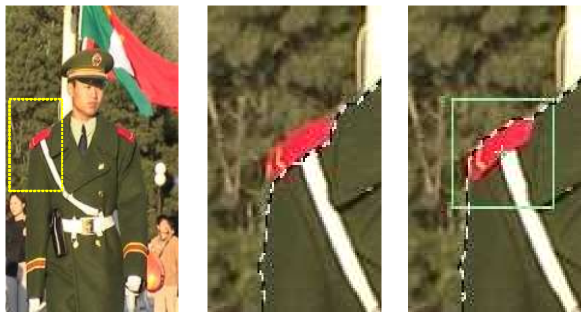

Figure 6: Local refinement by local color model. (a) One frame from video clip #3. (b) The 3D graph cut segmentation result. Notice that the error pixels have colors similar to the color of the red flag in the background. (c) The green rectangle is one window of a video tube. With a local color model that excludes irrelevant global color information, the boundary is precisely refined.

4.2 Constrained 2D graph cut segmentation

Once a video tube is located, a 2D graph cut segmentation is performed within each window to refine the existing segmentation boundaries. The 2D graph is constructed at the pixel level:

$$E(X) = \sum_{i \in \mathcal{V}'} E_1(x_i) + \lambda_1' \sum_{(i,j) \in \mathcal{A}'_I} E_2(x_i, x_j) \quad (7)$$

where x_i is the label of the pixel i, $\mathcal{V}'$ are all pixels in the tracker, and $\mathcal{A}'_I$ is the eight-neighboring relationship between pixels. E_1 and E_2 have similar definitions as in Equation (1) except that regions are replaced by pixels. The value of λ_1' is typically set to 10.

In order to seamlessly embed the refinement into the existing segmentation, a foreground and background hard constraint is automatically generated according to the existing segmentation result. As shown in Figure 5, the labels of all pixels inside the window are solved by the 2D graph cut algorithm, except for the pixels on the window's boundary. These pixels are marked as foreground hard constraints if it is in the foreground of the existing segmentation. Otherwise, they are marked as background hard constraints. Because of these hard constraints, the 2D graph cut segmentation inside the window must produce a result that is seamlessly connected to existing boundaries outside of the window, as shown in Figure 5 (b).

There must be at least one key frame inside a video tube. The pixels inside the window in the key frames are collected to compute the foreground/background GMM models for this video tube for the E_1 term above. Compared to the global color models in 3D graph cut segmentation, this local 2D graph cut segmentation uses more accurate color models in local windows and leads to significantly improved results. Figures 6(b) and 6(c) show the segmentation results before and after local refinement, respectively. This refinement method does not require accurate user interactions, because the user needs only to place the key windows to exclude irrelevant colors.

5 Postprocessing

Overriding operations When there are ambiguous edges around the boundary or the contrast of the border is very low, the graph cut algorithm may not be able to produce a correct object boundary. Moreover, it usually performs poorly for very thin structures, such as fingers.

To overcome these difficulties, our system allows the user to directly control the object boundary with great precision using two override brushes for identifying definite foreground and definite background regions, respectively. All overriding operations are recorded in an override layer, as shown in Figure 7(b), Figure 8(b), and Figure 9(b).

Coherent matting To extract the video object for pasting, we adopted coherent matting [Shum et al. 2004] to compute a fractional alpha matte for the object boundary. The coherent matting algorithm improves Bayesian matting by introducing a regularization term for the alpha. Hence, it produces an alpha matte that complies with the prior binary segmentation boundaries, and performs well even when foreground/background colors are similar.

The uncertain regions in matting are computed by dilating the binary object boundary, typically by 10 pixels. For small holes or thin gaps in the foreground, this dilation may result in no background colors to be sampled nearby. In this case, we instead sample background colors from neighboring frames.

6 Experiments

All experiments were performed on a 3.1GHz PC. The source videos were taken with a DV camera in progressive scan mode at a 12.5 frames/sec rate. Each clip was split into about 30 frames per segment, and each segment was loaded and processed individually. The key frames were usually sampled at every ten frames, while some clips needed denser samples due to fast motion or shadow changes.

The processing time was about half an hour for each segment of video. About 20% was for preprocessing and other computation, 40% for video tube tracking and adjustment, and another 40% for overriding operations.

Table 1 shows the complexity and processing time of the four video clips that appear in this paper. Pre-processing is performed only once for each segment and the watershed results and 3D graph can be saved and reused if needed. Time for building the 3D graph is needed only when the parameters are changed, which is rare. In our experiments, all parameters are fixed to the default values mentioned in previous sections.

Figures 7, 8, and 9 show some frames from our experiments. In these figures, (a) shows the 3D graph cut results as overlaid dashed lines. (b) shows the local refinements in video tubes and override layers. Dashed lines indicate the boundaries after both processes. The white pixels record the actions of foreground brushing and the black pixels for background brushing. (c) shows the coherent matting results pasted on a blue background. More experiments can be found in our accompanying video, including more video clips, complete video tube and overriding layers, and video object cut and paste results.

7 Conclusion

In this paper, we have proposed a video object cut-out system, which separates a video object from a complicated background, and preserves the details of the boundaries. Using a novel 3D graph cut based segmentation approach, our system can capture complex shape deformations with the input of only a few key frame mattes. Moreover, using local color models, the boundaries are well located even when colors are ambiguous. A bi-directional feature tracking algorithm is designed to track the regions of local color models. The resulting object sequence is ready to be composed onto other backgrounds.

In the future, we also plan to extend our system to light fields. Another interesting problem for future work is how to simultaneously cut out several moving objects from a video sequence.

Clip	#1		#2		#3		#4		
Width, Height	444,504		720,540		406,534		620, 550		
Number of frames	88		101		67		61		
Number of segments	2		2		2		3		
Number of key frames	19		12		12		12		
Pre-processing (sec.)	≈ 200		≈ 250		≈ 160		≈ 250		
Build 3D graph (sec.)	≈ 60		≈ 80		≈ 40		≈ 75		
Solve 3D graph cut (sec.)	≈ 3		≈ 3.6		≈ 3		≈ 7.5		
Number of video tubes	9	7	4	4	15	19	5	4	12
Solve all video tubes (sec.)	2.6	2.7	4.3	1.1	5.2	5.8	3.4	3.3	3.8

Table 1: Complexity and processing time.

Acknowledgment We would like to thank the anonymous reviewers for their constructive critiques. Many thanks to Yingqing Xu and Beijing Dancing Academy for help on video sources. Many thanks to Steve Lin for his great help in video production and proofreading, and to Zhengyou Zhang for his comments.

References

AGARWALA, A., HERTZMANN, A., SEITZ, S., AND SALESIN, D. H. 2004. Keyframe-based tracking for rotoscoping and animation. In *Proceedings of ACM SIGGRAPH 2004*, 584–591.

APOSTOLOFF, N. E., AND FITZGIBBON, A. W. 2004. Bayesian video matting using learnt image priors. In *Proceedings of CVPR 2004*, I: 407–414.

BELLMAN, R. E. 1957. *Dynamic Programming*. Princeton University Press, Princeton, NJ.

BLACK, M. J., AND ANANDA, P. 1996. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. In *Computer Vision and Image Understanding*, vol. 63, 75–104.

BLAKE, A., AND ISARD, M. 1998. Active contours. In *Springer Verlag, London.*

BLAKE, A., ROTHER, C., BROWN, M., P.PEREZ, AND P.TORR. 2004. Interactive image segmentation using an adaptive gmmrf model. In *Proceedings of ECCV*, I: 428–441.

BOYKOV, Y., AND JOLLY, M. P. 2001. Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *Proceedings of ICCV 2001*, I: 105–112.

CHUANG, Y.-Y., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2001. A bayesian approach to digital matting. In *Proceedings of CVPR 2001*, II: 264–271.

CHUANG, Y.-Y., AGARWALA, A., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2002. Video matting of complex scenes. In *Proceedings of ACM SIGGRAPH 2002*, 243–248.

DRORI, I., LEYVAND, T., COHEN-OR, D., AND YESHURUN, H. 2004. Interactive object segmentation in video by fitting splines to graph cuts. In *ACM SIGGRAPH 2004 Posters Session*.

KASS, M., WITKIN, A., AND TERZOPOULOS, D. 1987. Snakes: Active contour models. *International Journal on Computer Vision 1*, 4, 321–331.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. Graphcut textures: Image and video synthesis using graph cuts. In *Proceedings of ACM SIGGRAPH 2003*, 277–286.

LI, Y., SUN, J., TANG, C. K., AND SHUM, H. Y. 2004. Lazy snapping. In *Proceedings of ACM SIGGRAPH 2004*, 303–308.

MITSUNAGA, T., YOKOYAMA, T., AND TOTSUKA, T. 1995. Autokey: Human assisted key extraction. In *Proceedings of ACM SIGGRAPH'95*, 265–272.

MORTENSEN, E. N., AND BARRETT, W. A. 1999. Toboggan-based intelligent scissors with a four parameter edge model. In *Proceedings of CVPR 1999*, II: 452–458.

ROTHER, C., BLAKE, A., AND KOLMOGOROV, V. 2004. Grabcut - interactive foreground extraction using iterated graph cuts. In *Proceedings of ACM SIGGRAPH 2004*, 309–314.

SHUM, H., SUN, J., YAMAZAKI, S., LI, Y., AND TANG, C. 2004. Pop-up light field: An interactive image-based modeling and rendering system. *ACM Transaction of Graphics 23*, 2, 143–162.

SMITH, A. R., AND BLINN, J. F. 1996. Blue screen matting. In *Proceedings of ACM SIGGRAPH 1996*, 259–268.

VINCENT, L., AND SOILLE, P. 1991. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Tran. on PAMI 13*, 6, 583–598.

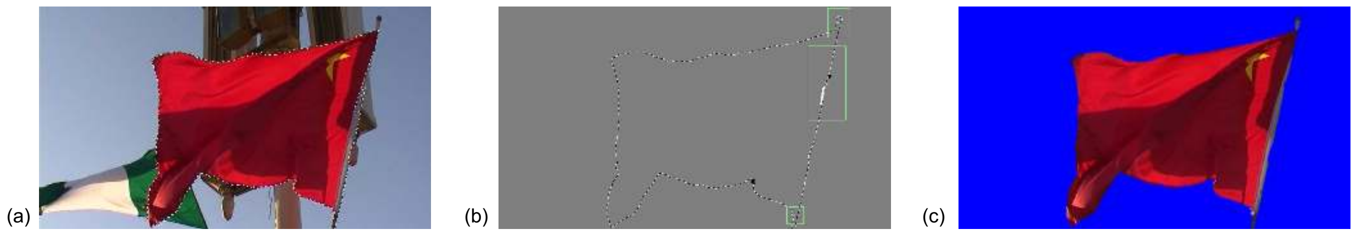

Figure 7: Clip #2, frame 27. (a) 3D graph cut result is shown by the overlaid dashed line. The flag is a rapidly deforming object, but 3D graph cut can capture the shape very well. (b) Dashed lines indicate the boundaries after both the local refinement and overriding operations. The white pixels record the actions of foreground brushing and the black pixels for background brushing. (c) coherence matting result pasted on a blue screen.

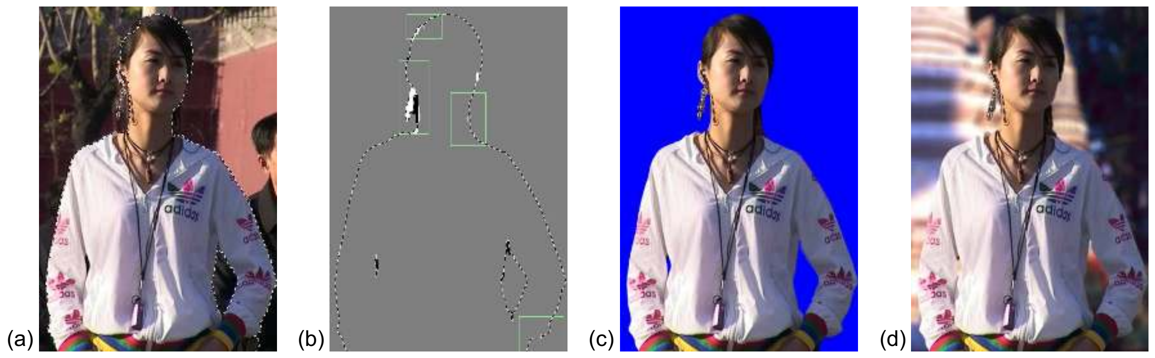

Figure 8: Clip #1, frame 84. (a) 3D graph cut result. Notice that the low contrast edges between the hair and the tree shadows, and the ambiguous edges around the earring are difficult for global optimization of 3D graph cut. (b) Local refinement in the video tube windows can correct most errors, but some fine details, especially thin structures, need manual override by the user. (c) Coherent matting result pasted on a blue screen. (d) Coherent matting result pasted on another background with bright colors in contrast to the original dark one. Notice that the video object is extracted well with clear boundaries.

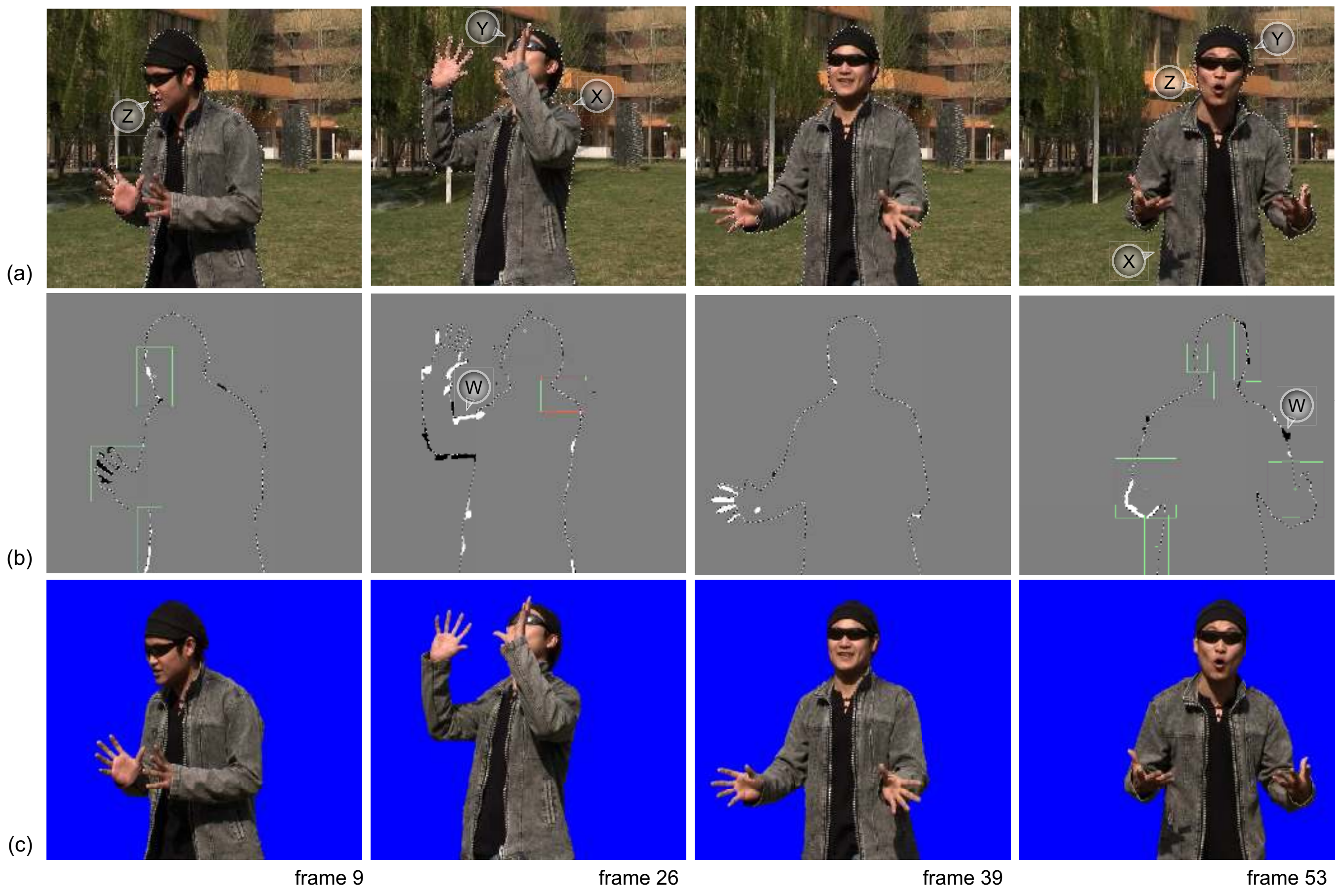

Figure 9: Clip #4. (a) 3D graph cut results. Please note that some subtle artifacts (X) are hardly visible in the still image, but they appear clearly in the video as flickering artifacts. (b) Local video tubes are usually used to refine the low contrast regions (Y) and ambiguous edges (Z). Overriding operations are usually necessary to eliminate artifacts caused by accidental shadow or texture changes (W), which do not appear in neighboring key frames. (c) Coherent matting results pasted on a blue screen.