Video SnapCut: Robust Video Object Cutout Using Localized Classifiers

Xue Bai[1] Jue Wang[2] David Simons[2] Guillermo Sapiro[1]

[1]University of Minnesota [2]Adobe Systems

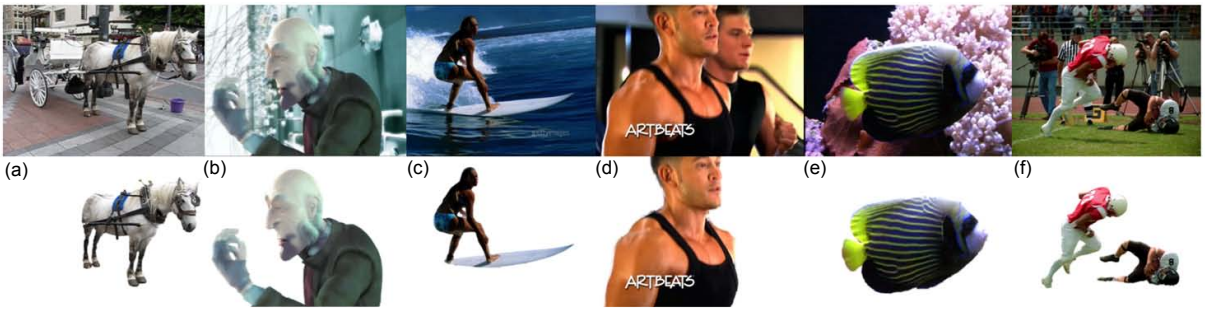

Figure 1: *We propose a video object cutout system that performs consistently well on a variety of examples which are difficult for previous approaches. See Section 6 and the accompanying video for more comparisons and results. Original videos in (d) and (f) courtesy of Artbeats.*

Abstract

Although tremendous success has been achieved for interactive object cutout in still images, accurately extracting dynamic objects in video remains a very challenging problem. Previous video cutout systems present two major limitations: (1) reliance on global statistics, thus lacking the ability to deal with complex and diverse scenes; and (2) treating segmentation as a global optimization, thus lacking a practical workflow that can guarantee the convergence of the systems to the desired results.

We present *Video SnapCut*, a robust video object cutout system that significantly advances the state-of-the-art. In our system segmentation is achieved by the collaboration of a set of local classifiers, each adaptively integrating multiple local image features. We show how this segmentation paradigm naturally supports local user editing and propagates them across time. The object cutout system is completed with a novel coherent video matting technique. A comprehensive evaluation and comparison is presented, demonstrating the effectiveness of the proposed system at achieving high quality results, as well as the robustness of the system against various types of inputs.

1 Introduction

Interactive object segmentation and matting in still images and video is a critical and challenging task that has received significant attention in recent years. While the state-of-the-art for still images has significantly advanced and recent developments are already being incorporated in commercial software packages, video cutout systems sufficiently sophisticated to be applied to real production are lacking.

In this paper we present *Video SnapCut*, a video cutout system that significantly advances the state-of-the-art and gets us closer to the incorporation of such tools into production. At the core of the system lies a new localized segmentation paradigm, where segmentation is achieved by the collaboration of a set of local classifiers. Each classifier adaptively integrates multiple local features such as color, edge, and on-line learned shape prior. Our system also naturally supports user-guided spatio-temporal local refinements.

The paper is organized as follows. After presenting a summarization of related work and an overview of the proposed approach in this section, we introduce the local classifiers used for segmentation and the one-frame propagation algorithm in Section 2. The extension to multi-frame propagation is presented in Section 3. A coherent video matting approach is presented in Section 4, and how interactive refinement is achieved in the system is described in Section 5. Experimental results and a comprehensive comparison are provided in Section 6. Finally, limitations of the current system is discussed in Section 7, and the paper concludes in Section 8.

1.1 Background

Tremendous progress has been achieved on interactive, or user-guided, object segmentation and matting of still images in recent years, as summarized in [Wang and Cohen 2007a]. In such a system, the user guides the segmentation either by specifying strokes indicating the foreground and background components (e.g., [Bai and Sapiro 2007; Boykov et al. 2001; Li et al. 2004; Rother et al. 2004]), or coarsely tracing the object's boundary (e.g., see [Mortensen and Barrett 1995; Wang et al. 2007]), and then relies on the underlying algorithms to optimize the final alpha matte. High quality results can be achieved for fairly complex images.

Although the same problem in video sequences has also been extensively studied, it remains unsolved and quite challenging. A variety of directions have been explored, such as extending 2D graph cut segmentation directly to 3D video volumes [Armstrong et al. 2007; Li et al. 2005; Wang et al. 2005], using optical flow to propagate the segmentation across time [Blake and Isard 1998; Chuang et al. 2002], tracking and moving spatially-weighted color models [Yu et al. 2007], or interpolating partial boundary splines using rotoscoping techniques [Agarwala et al. 2004]. Although various successful examples have been shown, these systems have not reached a point that can be easily adopted into a practical system and expected to perform reliably well on a variety of input sequences one

may encounter in video production. The application range of previous techniques is severely limited by the following major limitations:

Limited classification power. Most existing video cutout systems (e.g., [Bai and Sapiro 2007; Li et al. 2005; Wang et al. 2005]) use color as the primary feature and build global foreground and background color models as the major classifier, thus are not able to distinguish foreground objects from the background in complex scenes where their global color statistics are un-separable (see Figure 13(b)). Although the object cut-and-paste system [Li et al. 2005] uses local color models for local boundary refinement, which is somewhat similar to our local classifiers, color is still the only feature used there, thus it will not work for locally overlapping color distributions.

Un-controllable workflow. Many previous approaches (e.g., [Armstrong et al. 2007; Li et al. 2005; Wang et al. 2005]) treat interactive video cutout as a global optimization problem, thus any changes the user makes in a local region of one frame may affect the existing good segmentation in other un-related regions on the same and/or other frames. As a result, the convergence of these systems is mainly example-dependent and is often uncontrollable by the user. From our observations we found that these systems are very hard to converge on difficult examples, since new errors are constantly introduced when the user is fixing existing errors.

Limitations on topology issues, computational efficiency, and the user interface. Previous approaches present various other limitations that preclude them from practical use. Many global optimization systems need to access all video frames at the same time, thus require significant amounts of memory. Some of them (e.g., [Wang et al. 2004]) contain a computationally expensive preprocessing component. Compared with global optimization techniques, Rotoscoping [Agarwala et al. 2004; Stewart 2003] is a more localized technique for tracing partial foreground boundaries, however it cannot deal with rapid topology changes and often requires a significant amount of user interaction. In contrast to more standard keyframe-based user interfaces, treating video as a 3D volume and allowing the user to directly manipulate the spatio-temporal 3D video cube has gained considerable attention recently [Wang et al. 2005; Waschbsch et al. 2006]. However, in practice users find this UI unnatural and hard to use. When given a new video, it is extremely hard for the user to mentally map what he/she sees in the rotated, sliced, and deformed video cube to the desired foreground objects.

1.2 Video SnapCut Overview

Our goal is to develop a practical video cutout system that can perform reliably well on complicated video footage that previous systems cannot deal with. Motivated by studies on human visual systems [Wandell 1995; Komogortsev and Khan 2004], we have found two important principles that can guide us to develop a better cutout system: (1) Multiple cues should be used for extracting the foreground, such as color, texture, shape, and motion. Among these, shape should play an important role for maintaining a temporally-coherent recognition; and (2) Multiple cues should be evaluated and integrated *both locally and globally*, rather than *just globally*, to maximize their discriminant powers.

Inspired in part by these principles, we propose a new video segmentation model of *overlapping localized classifiers*. It consists of a group of overlapping windows around the foreground object boundary, each associated with a local classifier which only segments a local piece of the foreground boundary. The spatial locations of these classifiers are pushed along with motion vectors estimated across frames, and the segmentation of a new frame is achieved by aggregating local classification results together. Furthermore, each local classifier carries local image features such as color, shape, and motion, and adaptively integrates them together to generate the optimal classification. Compared with previous systems, our approach has the following major characteristics and advantages:

The ability to handle complicated scenes. Since each classifier is adaptively integrating multiple cues for segmenting only a local region, aggregating all the classifiers together gives much more discriminative power than methods based on global statistics. Topological changes are naturally handled as well.

A natural workflow. Our system adopts a keyframe-based forward-propagation workflow which is already familiar to users. Unlike previous approaches, our system naturally supports local spatio-temporal editing. Once additional user input has been made in a local region, only nearby classifiers are affected. Furthermore, since the classifiers are tracked over time, the local editing can be easily propagated across frames. The user can easily control how large the impact region should be for each local correction, both spatially and temporally. This is also beneficial for computational efficiency since the common problem of re-running the segmentation on a whole video for every local correction is avoided.

2 Segmenting with Localized Classifiers

Given an input video sequence, the segmentation process starts by having the user provide a relatively accurate mask for the desired object on the first (*key*) frame, using image-based object selection approaches. We found that for example the lazysnapping system [Li et al. 2004] and the Quick Selection tool in Photoshop [Adobe Systems 2008] are both good candidates for this task.

Once the initial mask is created, a group of local classifiers are constructed around the foreground boundary, which are then propagated onto successive frames segmenting the object. In this section we describe how the classifiers are initialized and propagated to the next frame for segmentation, and in Section 3 we detail how this is extended to a series of frames to achieve coherent segmentation for blocks of video frames.

2.1 Overlapping Local Classifiers

As shown in Figure 2, given the initial mask $L^t(x)$ on a keyframe I_t, we uniformly sample a set of overlapping windows $W_1^t, ..., W_n^t$ along its contour C_t (for simplicity we assume a single contour for now, multiple contours can be handled in the same way and examples are presented in the experimental results). The size of the windows can vary according to the size of the object, and it is usually 30×30 to 80×80 pixels. Each window defines the application range of a local classifier, and the classifier will assign to every pixel inside the window a foreground (object) probability, based on the local statistics it gathers. Neighboring windows overlap for about 1/3rd of the window size. We will explain why the windows need to overlap in Section 2.5.

Each classifier inside the window W_k^t consists of a local color model M_c, a color model confidence value f_c, and a local shape model M_s. Similar to previous approaches, we build Gaussian Mixture Models (GMMs) for the local foreground ($\mathcal{F}$) and background ($\mathcal{B}$) regions, in the Lab color space.[1] To avoid possible sampling errors, we only use pixels whose spatial distance to the segmented boundary is larger than a threshold (5 pixels in our system) as the training data for the GMMs. The number of components in each GMM is set to 3 as we expect simple color distributions in a small

[1] We should emphasize that the proposed video cutout framework is not limited just to the use of color features. See also Section 7 for more discussions on the selection of color space.

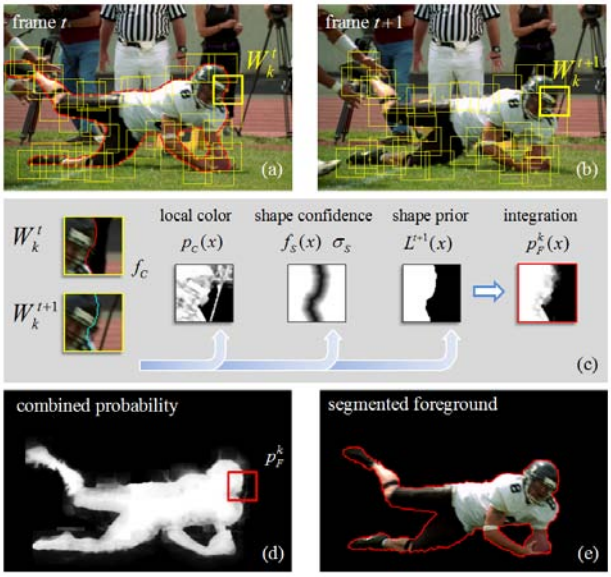

Figure 2: *Illustration of local classifiers. (a) Overlapping classifiers (yellow squares) are initialized along the object boundary (red curve) on frame t. (b) These classifiers are then propagated onto the next frame by motion estimation. (c) Each classifier contains a local color model and shape model, they are initialized on frame t and updated on frame t+1. (d) Local classification results are then combined to generate a global foreground probability map. (e) The final segmented foreground object on frame t + 1. Original video courtesy of Artbeats.*

local region. For a pixel x in the window,[2] its foreground probability generated from the color model is computed as

$$p_c(x) = p_c(x|\mathcal{F}) / \left(p_c(x|\mathcal{F}) + p_c(x|\mathcal{B})\right), \quad (1)$$

where $p_c(x|\mathcal{F})$ and $p_c(x|\mathcal{B})$ are the corresponding probabilities computed from the two GMMs.

The local color model confidence f_c is used to describe how separable the local foreground is against the local background using just the color model. It allows us to apply the constructed color models conservatively, as done for image segmentation, but not for video, in [Li et al. 2008; Protiere and Sapiro 2007]. Let $L^t(x)$ be the known segmentation label ($\mathcal{F} = 1$ and $\mathcal{B} = 0$) of pixel x for the current frame, f_c is computed as

$$f_c = 1 - \frac{\int_{W_k} |L^t(x) - p_c(x)| \cdot \omega_c(x) dx}{\int_{W_k} \omega_c(x) dx}. \quad (2)$$

The weighting function $\omega_c(x)$ is computed as $\omega_c(x) = exp(-d^2(x)/\sigma_c^2)$, where $d(x)$ is the spatial distance between x and the foreground boundary, computed using a distance transform. σ_c is fixed as half of the window size in our system. $\omega_c(x)$ is higher when x is closer to the boundary, meaning that we require the color model to work well near the foreground boundary, thus an accurate segmentation could be generated.

The local shape model M_s contains the existing segmentation mask $L^t(x)$, and a shape confidence mask computed as

$$f_s(x) = 1 - exp(-d^2(x)/\sigma_s^2), \quad (3)$$

[2] With a slight abuse of notation, we use x to denote both the pixel location and its color values.

where as before $d(x)$ stands for the distance to the foreground boundary and σ_s is a parameter. An example of the shape confidence map is later shown in Figure 5(d). A large σ_s means that the shape confidence is low around the foreground boundary, while a small σ_s means a high confidence on the segmentation mask $L^t(x)$. In fact, σ_s is a very important parameter in our approach, we will show later how σ_s is adaptively and automatically adjusted to achieve accurate local segmentation.

Once initialized, the classifiers, along with their associated windows, propagate onto the next frame according to motion estimations (Section 2.2). The color and shape models in each classifiers are then updated (Section 2.3), and integrated according to new local image statistics (Section 2.4). Finally the outputs of all the local classifiers are aggregated to generate the segmentation for the new frame (Section 2.5), as shown in Figure 2(d),(e).

One may find that local image patches have been used for video segmentation in [Kohli et al. 2007]. However, the small patches (4×4) defined in that work are just used to compute textons as describing features. This is in sharp contrast with the use of windows in our work, which are tracked across frames, and defined as the application range of local classifiers that adaptively integrate multiple local features.

2.2 Single-Frame Local Window Propagation

The goal of window propagation is to move local windows along with the moving object so that its boundary on the next frame is still fully covered by these windows, although not yet accurately. To achieve this, rough motion estimation is necessary.

Our motion estimation operates in a hierarchical fashion. From two successive frames I_t and I_{t+1}, we first estimate a global affine transform from matching SIFT feature points, [Lowe 2004], between the two frames (only SIFT features inside the foreground object in I_t are used for matching), and use this transform to align I_t to I_{t+1}, resulting in a new image I'_{t+1}. This initial shape alignment usually captures and compensates for large rigid motions of the foreground object.

In the second step, optical flow is estimated between I'_{t+1} and I_{t+1}, in order to capture the local deformations of the object. In our case, we are only interested in flow vectors for pixels inside the object in I'_{t+1}. It is well known that optical flow is unreliable, especially near boundaries where occlusions occur, thus in our system we use a local flow averaging approach which generates significantly more robust results. As shown in Figure 3, let x' be a point on the affinely-transformed contour C'_{t+1}, Ω the inside region of C'_{t+1}, and N'_x a window centered at x'. The new position of x is then computed as $x^* = x' + \overline{v}_x$, where $\overline{v}_x$ is the average flow vector inside the region $\Omega \cap N'_x$. Since the motion of the object is locally smooth, the local average flow gives significantly more robust contour prediction.

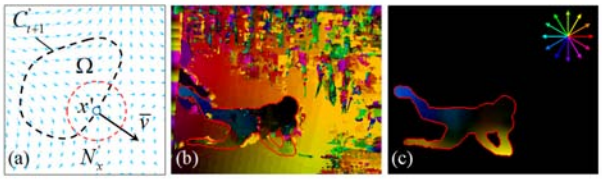

Figure 3: *(a) A window center x' is moving along the regional average optical flow $\overline{v}_x$ inside the contour. (b) Pixel-wise optical flow (color represents flow direction (see (c)) and intensity represents magnitude.). (c) Average flow inside the contour, which is locally smooth while the deformation of the object is maintained.*

Note that we do not assume and require accurate motion estimation.

The motion vectors are only used to shift local windows/classifiers, so that the contour of the moving object can be covered by them. The relatively large local windows are tolerant to moderate misalignments caused by non-accurate motion vectors.

2.3 Updating Local Models

Both the color model and the shape model of the local classifier need to be updated once the window is propagated onto the next frame. The shape model M_s is naturally updated by warping the local foreground mask $L^t(x)$ according to the motion estimation described in the previous section (see next section for more on the shape confidence $f_s(x)$). Although motion estimation contains errors, the warped shape model is usually good enough as the initial update. We later show in Section 2.5 how the segmentation is further improved in an automatic iterative fashion.

Updating the color model M_c is essential for achieving good segmentation. Since the foreground and the background usually have different motions, new background colors may appear in the local region (window). To update the color model, we build a new foreground GMM $G_{\mathcal{F}}^{t+1}$ by sampling all pixels in the warped window whose foreground confidence computed from the updated shape model is greater than a high threshold $T_{\mathcal{F}}^s$ (see Equation 3). A new background GMM $G_{\mathcal{B}}^{t+1}$ is constructed in a similar way using a low threshold $T_{\mathcal{B}}^s$. The thresholds $T_{\mathcal{F}}^s$ and $T_{\mathcal{B}}^s$ are set conservatively (0.75 and 0.25 in our system), so that the sampled pixels are relatively far from the initially estimated boundary.

Now, for the local classifier we have two possible sets of color models for the frame being processed: (1) History color model $M_c^h := \{G_{\mathcal{F}}^t, G_{\mathcal{B}}^t\}$, which is carried on from the previous frame; and (2) Updated color model $M_c^u := \{G_{\mathcal{F}}^t, G_{\mathcal{F}}^{t+1}, G_{\mathcal{B}}^t, G_{\mathcal{B}}^{t+1}\}$, where newly estimated color models are added to the history model.

An adaptive decision has to be made regarding which model to use for segmenting the current image window. Ideally, if the motion estimation is perfect, then we should always use M_c^u, since it captures the new $\mathcal{F}$ and $\mathcal{B}$ colors that just appeared in the local region. However, since motion estimation may contain errors, in particular for large motions, $G_{\mathcal{F}}^{t+1}$ and $G_{\mathcal{B}}^{t+1}$ could be inaccurate due to miss-sampled pixel colors.

To deal with potential sampling errors, we assume that for a local classifier, the local $\mathcal{F}$ region generally remains consistent across consecutive frames, since the local windows are tracked with the foreground, while the local $\mathcal{B}$ region may present dramatic changes (see Figure 4). Under this assumption $G_{\mathcal{F}}^t$ is similar to $G_{\mathcal{F}}^{t+1}$, while $G_{\mathcal{B}}^t$ can be significantly different than $G_{\mathcal{B}}^{t+1}$, giving a richer background model which can reduce the classification error of the foreground. We then test the two possible models M_c^h and M_c^u in the local window, resulting in two classification maps p_c^h (using M_c^h) and p_c^u (using M_c^u), as shown in Figure 4. If p_c^u has a larger foreground region (white pixels) than p_c^h, which means that in M_c^u the foreground suddenly has a lot more colors, we then choose to use M_c^h, since M_c^u might be mis-constructed given the foreground consistency assumption. Otherwise if p_c^u has a smaller foreground region we then trust the updated color model M_c^u.

For the color confidence value f_c, if M_c^h is chosen as the color model, then it remains unchanged. Otherwise we recompute f_c, following Equation 2, now including all the samples used to estimate $G_{\mathcal{F}}^{t+1}$ and $G_{\mathcal{B}}^{t+1}$.

While the foreground consistency assumption could be violated by luminance variance (this can be addressed by illumination-invariant features instead of Lab), foreground self-occlusion, or object rotation, we found this assumption to hold true for most common scenarios in natural videos, and it helps to significantly improve the performance of our system in general. For special cases, the user always has the possibility to locally adjust how the color models should be updated, as we will explain in detail in Section 5.

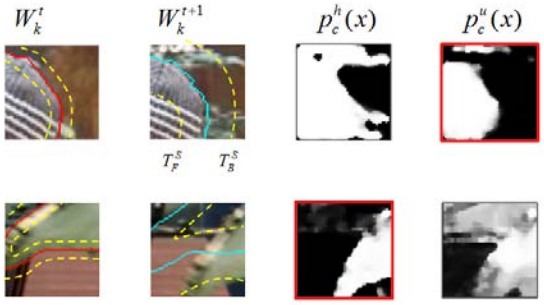

Figure 4: *First row: adding new color samples in W_k^{t+1} generally makes a better color model for classification in frame $t+1$. Second row: sometimes when motion estimation fails locally, $\{G_{\mathcal{F}}^{t+1}; G_{\mathcal{B}}^{t+1}\}$ are mistakenly constructed and lead to an erroneous foreground probability map. Our model selection scheme will choose the right color model to use in both cases (red border).*

2.4 Adaptive Color and Shape Integration

The local classifier adaptively integrates the updated shape model M_s and color model M_c for local segmentation. The underlying idea is conceptually simple: when the foreground and background colors are separable, then the color model M_c should be trusted more. Otherwise the shape model M_s should play a bigger role for temporally consistent segmentation.

Such adaptive integration is achieved by using the (updated) color model confidence f_c and the shape confidence map $f_s(x)$ (see Section 2.1). Formally, the parameter σ_s in Equation 3, which controls the influence of the shape model via $f_s(x)$, is computed as

$$\sigma_s = \begin{cases} \sigma_{min} + a(f_c - f_{cutoff})^r & f_{cutoff} < f_c \leq 1, \\ \sigma_{min} & 0 \leq f_c \leq f_{cutoff}, \end{cases} \quad (4)$$

where $a = (\sigma_{max} - \sigma_{min})/(1 - f_{cutoff})^r$. In our system we fix $f_{cutoff} = 0.85$, $\sigma_{min} = 2$, $r = 2$, and σ_{max} to be the size of the local window. The profiles of σ_s and $f_s(x)$ are shown in Figure 5(a),(b). It is clear that a high f_c will result in a large σ_s, which produces a loose shape constraint, and vice versa.

The foreground probability $p_{\mathcal{F}}^k(x)$, corresponding to the window W_k in the current frame, becomes a linear combination of the updated color probability $p_c(x)$ and the updated (warped) binary shape mask $L^{t+1}(x)$, using the shape confidence map $f_s(x)$ with the just computed σ_s as the interpolation coefficients:

$$p_{\mathcal{F}}^k(x) = f_s(x)L^{t+1}(x) + (1 - f_s(x))p_c(x). \quad (5)$$

Examples of $p_c(x)$, $f_s(x)$, and $p_{\mathcal{F}}^k(x)$, are shown in Figure 5(c),(d),(e).

Figure 6 shows examples of integrating color and shape cues to achieve good classification in different scenarios. Note how our local classifier achieves good results even when the foreground and background have highly overlapping color distributions.

2.5 Object Cutout and Iterative Refinement

Let us now look at how the local classifiers work together to generate the coherent segmentation of the whole object in the current

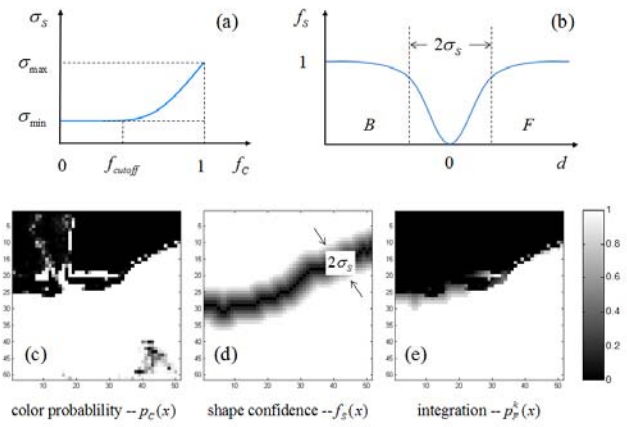

Figure 5: *(a) As color confidence f_c increases (more separable foreground and background color distributions), the value of σ_s increases, giving less weight to the shape prior. (b) Profile of the shape prior $f_s(x)$. An example of (c) color probability $p_c(x)$, (d) shape confidence $f_s(x)$ with parameter σ_s, and (e) the integrated probability $p^k_{\mathcal{F}}(x)$.*

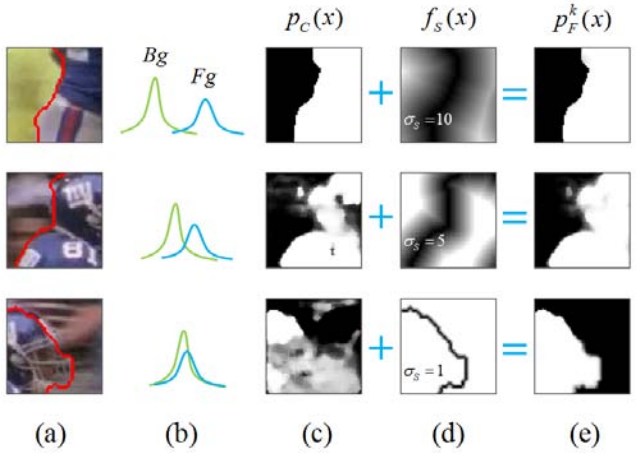

Figure 6: *Depending on the color complexity in the local window, the color and shape priors are integrated with adaptive weights. The columns correspond to (a) local window W^t_k, (b) foreground and background histograms (blue and green respectively), (c) color probability $p_c(x)$, (d) shape confidence $f_s(x)$, and (e) integrated probability $p^k_{\mathcal{F}}(x)$.*

frame. First note that since each local window moves along its own (averaged) motion vector, the distances between updated neighboring windows may slightly vary. This is one of the main reasons to use overlapping windows in the keyframe, so that after propagation, the foreground boundary is still fully covered by the windows.

In this overlapping window design, a pixel is often covered by multiple adjacent windows, and its foreground probability is a weighted linear combination of the probabilities computed from each window it belongs to,

$$p_{\mathcal{F}}(x) = \frac{\sum_k p^k_{\mathcal{F}}(x)(|x - c_k| + \epsilon)^{-1}}{\sum_k (|x - c_k| + \epsilon)^{-1}}, \qquad (6)$$

where k is the index of local windows (the sum ranges over all the k-s such that the updated window W^{t+1}_k covers the pixel), ϵ is a small constant (0.1 in the system), and c_k is the center of the window ($|x - c_k|$ is the distance from the pixel x to the center).

A full-frame foreground probability map can be constructed in this way, as shown in Figure 2(d). Although local classifiers only deal with the boundary region, the internal area of the foreground is automatically filled-in due to the whole-frame motion estimation described in Section 2.2. We then apply a pixel-level Graph Cut segmentation [Boykov et al. 2001] to generate a new foreground mask $L^{t+1}(x)$.

For large motions, this process can be iterated to generate more accurate segmentation. As detailed in previous sections, both the color and shape models of a local window are updated when propagating, through motion compensation, the initial foreground mask $L^t(x)$ to frame $t+1$ as $L^{t+1}(x)$. Denoting this initial mask as $L^{t+1}_0(x)$, a new mask $L^{t+1}_1(x)$ is obtained after segmentation, which is usually more accurate than $L^{t+1}_0(x)$. We then use $L^{t+1}_1(x)$ to update the color and shape models again, and re-compute a new segmentation $L^{t+1}_2(x)$. This process can iterate until $L^{t+1}_q(x)$ has converged. Figure 7 shows the results of this iterative segmentation process for one frame of an image region with large motion, the foreground boundary gradually becomes more accurate.

The iterative refinement process is carried out in each local window, thus different local windows can have different converging rates. Once a window has converged, we fix its foreground mask and do not update it anymore. The Graph Cut segmentation is only applied in the active windows, using converged windows as boundary constraints. In practice we found that over 90% of local windows converge after 3 iterations, thus the iterative process is computationally very efficient.

Figure 7: *Segmented foreground contour (red curve) before (just global motion plus optical flow) (a) and after 1st (b), 4th (c) and 6th (d) iteration. The contour evolves to snap to the object's boundary. Original video courtesy of Artbeats.*

3 Multi-frame Propagation

Directly applying the propagation and segmentation method described in Section 2 sequentially from frame to frame may encounter some new challenges. In this section we identify them and propose their solutions. Figure 8 shows the flow chart of the multi-frame propagation workflow.

3.1 Stabilizing the Color Models

The segmentation generated by the single-frame propagation system described above is not necessarily perfect and may contain errors. These errors affect the updated color models, which might lead to more errors in the next frame if a simple frame-by-frame propagation is adopted. If no constraints are applied, the color models might gradually drift away from capturing the correct $\mathcal{F}$ and $\mathcal{B}$ colors, resulting in amplified errors in the propagation process.

As shown in Figure 8, we stabilize the color models by forcing the system to always consider the original foreground models in the starting keyframe $G^0_{\mathcal{F}}$, no matter which frame is currently being

segmented. In other words, $G_{\mathcal{F}}^t$ is replaced by $G_{\mathcal{F}}^0$ in Section 2.3, no matter which frame t we are currently working on. In this way we maintain a consistent foreground color model across frames, and in each frame the local classifier will jointly consider the original foreground color model generated from the starting keyframe and the new colors presented in the current frame $t+1$. In case the foreground colors dramatically change, the user can start a new keyframe and the old models are discarded.

Figure 8 shows a single window which is tracked for several consecutive frames, and how the color and shape models are adaptively integrated in each frame to produce a consistent segmentation.

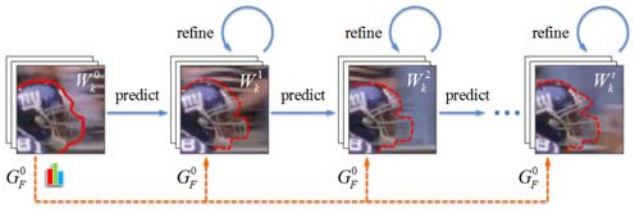

Figure 8: *System flow chart for multi-frame-propagation. The original foreground color model $G_{\mathcal{F}}^0$ is being applied to the segmentation of each of the successive frames.*

3.2 Dealing with Topology Changes

Dynamic foreground objects often present topology changes, e.g., legs crossing for a walker. In our system, we can use multiple contours to describe a foreground object, thus the number of contours, as well as the lengths of each contour, may vary according to the motion of the object. A robust cutout system needs to be able to handle these topological changes, a challenge that has not been properly addressed before for this application.[3]

In order to understand how the proposed cutout system can handle topological changes, it is important to note that our segmentation paradigm does not have any strict constraint on the foreground topology. In other words, in each window the segmented foreground region can have any arbitrary topology. Topology changes can be naturally dealt with as long as the local windows fully cover the foreground boundaries. To ensure this coverage, we dynamically adjust the number of windows and their locations after segmenting each frame, as explained next.

Assume that we have n local windows/classifiers on frame t, denoted as W_k^t, $k = 1, .., n$, and they are propagated onto frame $t+1$ and used to generate a new binary segmentation L^{t+1}. We then compute the displacement of the center of each window to its closest new boundary points, and move each window along the locally-averaged (for the purpose of robustness) displacement so that it will again center on the new boundary, as in Figure 9. A window layout adjustment step is then carried out to: (1) remove windows if the local density of windows is too high, i.e., every boundary pixel is covered by more than two windows; and (2) add new windows in places where boundary pixels have not yet been covered by existing (re-centered) windows. In this way we make sure the new foreground contours are fully covered by a minimal set of well distributed windows.

For newly-added windows, their corresponding color and shape models can be naturally constructed by looking at the segmentation

[3]While level-set approaches for object tracking have been proposed in the literature, and can handle topological changes, they do not achieve the level of accuracy expected from a production-level video cutout system and suffer from the same problems mentioned in the introduction that are common to previous approaches.

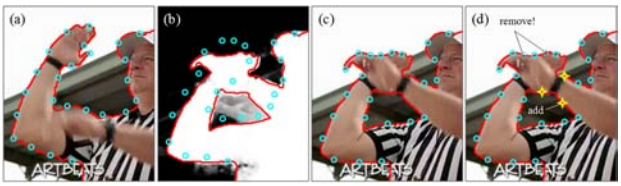

Figure 9: *(a) Local window centers (cyan circles) on frame t. (b) These windows are propagated to frame $t+1$ to generate a new segmentation with topology changes. (c) The window centers are then projected onto the new contours. (d) Window layout adjustment is carried out for re-distributing the local windows. Original video courtesy of Artbeats.*

labels inside them. The color model confidence value is also initialized. The current frame thus becomes the birth frame of these windows, and their succeeders will carry on the foreground color models created at this frame. Now the construction of a whole new set of local classifiers is completed, and they are ready to be propagate forward for segmenting the following frame. Figure 9 shows an example of how our system naturally deals with topology changes. Additional examples are presented in the experiments section and the accompanying video.

Using exactly the same window adjustment method, our system also naturally deals with occlusions. Figure 10 shows such an example. Note how the system handles complicated occlusions without any user assistant.

3.3 Bi-directional Propagation

So far we have only described a forward propagation workflow, where the user sets an initial starting keyframe, and lets the segmentation propagate forward. Alternatively, and similar to some previous approaches [Agarwala et al. 2004; Chuang et al. 2002], we can also define two keyframes at time t^1 and t^2 (we assume t^1 comes before t^2 in the normal video playing order), and segment all intermediate frames using both constraints. Our system thus has the flexibility of switching between different workflows based on the user's choice.

To achieve this we carry out two individual propagation processes: a forward propagation from t^1 to t^2, and a backward propagation from t^2 to t^1. For an intermediate frame t, the two processes merge. Formally, the foreground probability map on frame t is computed as $p_{\mathcal{F}}^t(x) = \frac{(t-t^1)\cdot p_{\mathcal{F}}^2(x)+(t^2-t)\cdot p_{\mathcal{F}}^1(x)}{t^2-t^1}$, where $p_{\mathcal{F}}^1(x)$ and $p_{\mathcal{F}}^2(x)$ are the corresponding probability maps for frame t computed from the forward and backward processes, respectively. The iterative refinement process (Section 2.5) is then applied, initialized with $p_{\mathcal{F}}^t(x)$, to generate the desired segmentation.

4 Coherent Matting

For applications such as compositing, the binary segmentation results need to be further processed to create soft alpha mattes, process known as alpha matting. Recently, significant advances on image matting have been achieved, as summarized in [Wang and Cohen 2007a]. In the proposed system we extend the state-of-the-art image matting algorithms to video.

Given the binary mask $L^t(x)$ on frame t, we first erode and dilate it to create a trimap with a uniform bandwidth set by the user, as shown in Figure 11. The bandwidth of the unknown region can be also locally varied, as we will describe in Section 5. A matting

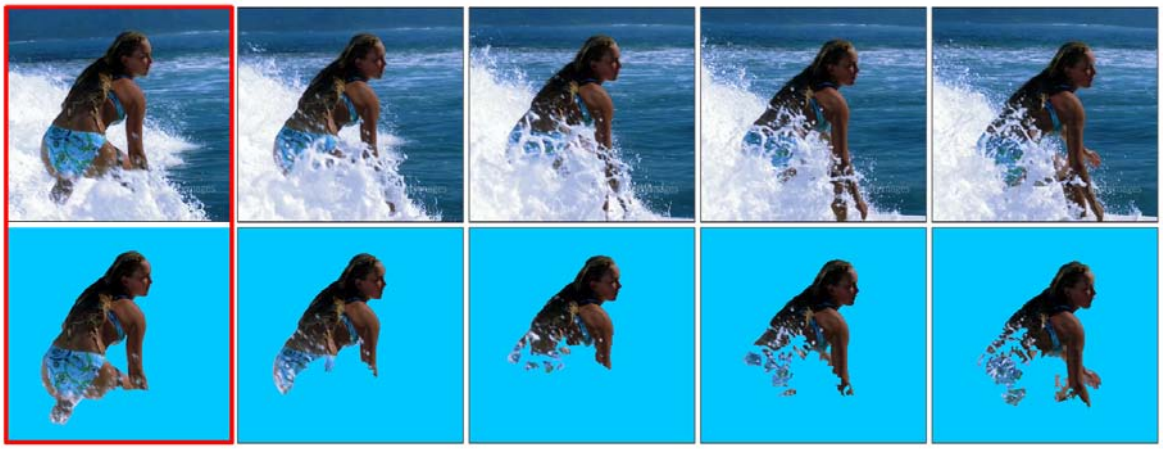

Figure 10: *Our proposed system automatically handles complex occlusions presented in this example. The first frame is the keyframe.*

procedure is then followed to estimate an alpha matte α^t as

$$\alpha^t = \arg\min_{\alpha^t} \sum_{x \in U} \left[\lambda_x^T(\alpha_x^t - \hat{\alpha}_x^{t-1})^2 + \lambda_x^C(\alpha_x^t - \tilde{\alpha}_x^t)^2\right] + \lambda^S \alpha^{tT} L_m \alpha^t, \quad (7)$$

where U is the unknown region of the trimap. The first two terms are data terms, or alpha priors, assigned to each individual pixel, while the third term is the neighborhood term which constraints the smoothness of the alpha matte. Specifically, the first data term is the temporal coherence term, where $\hat{\alpha}_x^{t-1}$ is the temporal alpha prior for x computed from frame $t-1$, which will be described in detail later. The second data term is the color coherence term, similarly to the data term proposed in the RobustMatting approach [Wang and Cohen 2007b]. For the smoothness term we use the matting Laplacian proposed in [Levin et al. 2008], which has been shown to be more accurate than other matting affinities [Wang and Cohen 2007a], and L_m is the matting Laplacian matrix defined in Equation 5 in [Levin et al. 2008]. λ^S is a weight to balance the data term and smoothness term, which is typically set to be 20 in our experiments, and can be locally changed by the user.

The temporal alpha prior $\hat{\alpha}_x^{t-1}$ is important for maintaining the temporal coherence of the matte. It is computed as $\hat{\alpha}_x^{t-1} = \frac{\sum_k \beta_{W_k} \cdot \alpha^{t-1}(x_{W_k}^{t-1})}{\sum_k \beta_{W_k}}$, where W_k indicates the k-s window that covers x, and $x_{W_k}^{t-1}$ is the pixel corresponding to x on frame $t-1$, having the same relative coordinates inside window W_k on frame $t-1$ as x on frame t. β_{W_k} is the temporal coherence weight for window W_k, computed based on how well the two segmentations inside W_k match on both frames. Denoting the binary segmentation map inside W_k on frame $t-1$ as $L_{W_k}^{t-1}$, and the corresponding segmentation on frame t as $L_{W_k}^t$, the weight is computed as

$$\beta_{W_k} = \exp\left(-\frac{1}{|W_k|} \sum_{i \in W_k} |L_{W_k}^t(i) - L_{W_k}^{t-1}(i)| / \sigma_w\right), \quad (8)$$

where $|W_k|$ is the size of the window and σ_w is a constant in the system, set to 0.1 in our experiments. Intuitively, if the segmentation inside W_k maintains the same shape across frames, then β_{W_k} is high, indicating the the two local segmentation patches have a strong correlation, thus stronger local temporal coherence constraint is needed. On the other hand, if the segmentation results change dramatically inside the same window across frames, for instance, local topology change occurs in these two frames, then a weaker temporal coherence constraint is enforced, since no correct registration can be easily achieved. The pixel-wise weight λ_x^T in Equation 7 is simply computed as $\sum_k \beta_{W_k} / \sum_k$, the average temporal coherence weight of all windows that cover it.

The alpha prior $\tilde{\alpha}_x^t$ is built based on the local known foreground and background color statistics, as used in many image matting approaches. The details of how to compute $\tilde{\alpha}_x^t$ and the pixel-wise confidence value λ_x^C are described in Section 4.1 in [Wang and Cohen 2007b] (see equations 2 and 6 in [Wang and Cohen 2007b] for $\tilde{\alpha}_x^t$ and λ_x^C, respectively).

Since all the terms in Equation 7 are defined as quadratic functions, the optimal alpha matte can be estimated by solving a linear system. Figure 11 shows how the temporal coherence prior can greatly stabilize the matting results across frames.

Finally, true foreground colors for pixels with fractional alpha values need to be estimated in order to achieve seamless composition. The optimization method proposed in [Levin et al. 2008] (Section 7) is used in our system for this purpose.

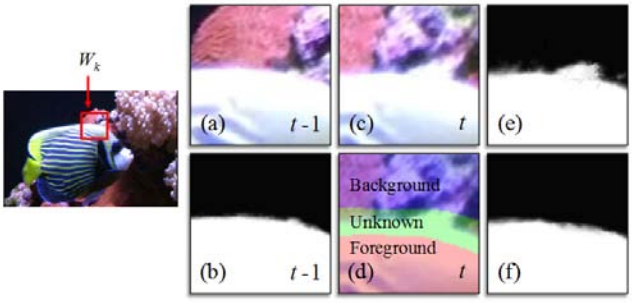

Figure 11: *(a) Local window W_k on frame $t-1$. (b) Matte on frame $t-1$. (c) W_k on frame t. (d) Trimap generated on frame t for W_k. (e) Matte computed without the temporal alpha prior $\hat{\alpha}_x^{t-1}$. (f) Matte computed with the proposed temporal alpha prior.*

5 Interactive Local Refinement

Sometimes the user needs to provide additional input to correct small segmentation errors, especially in local regions where the foreground boundary is extremely weak. Our system provides a natural infrastructure for spatio-temporal local editing.

A set of interactive tools are provided in our system for local boundary editing, such as foreground and background brushes, and spline-based local rotoscoping tools. In case the local brushes are used, a Graph Cut or Geodesic image segmentation is applied only in the affected local windows. Once the foreground boundary is updated, we select all the local windows that are affected by this local editing, make the modified frame as the starting keyframe for these windows, and automatically re-compute local segmentation in them in the following frames. In this way the local correction can be propagated in time and the same errors are removed in later frames, as shown in the example in Figure 12.

Figure 12: *Local correction. First row: automatic segmentation results in four consecutive frames with mistakes (the lamp base is misclassified as part of the foreground). Second row: The user correction updates a small number of windows and then the correction quickly propagates to multiple frames.*

In addition to directly correcting the segmentation errors, the user can also set different parameters locally so that segmentation and matting can be done differently in different local regions, thanks to our localized segmentation paradigm. Locally-variable parameters include the variance σ_c in Equation 2, f_{cutoff}, σ_{min} and r in Equation 4, the trimap bandwidth for matting, λ^S in Equation 7, and σ_w in Equation 8. Changing these parameters sometimes can produce better local segmentation and matting results, although we found that their default values work consistently well across a wide range of examples we have tested.

6 Comparisons and Results

We have tested the proposed system on a wide range of example videos, as shown in Figure 1. The dataset includes film, videos shot by professional HD cameras, and home video clips. The complete segmentation results of these videos are shown in the accompanying video. Note that our test examples contain highly complex color distributions, dynamic backgrounds, and rapid topology changes.

Figure 13(a) shows one frame of the "horse" video. Note that the foreground object and the background scene both contain exactly the same white colors, which will confuse any classifiers built upon global color statistics, as shown in Figure 13(b). Any video segmentation system that uses global color classifiers (see Section 1.1) will fail to generate good segmentation results on this example. In contrast, our system employs local color models which have much stronger discriminant power than global models, as shown in Figure 13(c). Furthermore, by adaptively integrating color and shape models together, our system produces a high quality foreground probability map which leads to a good segmentation, as visualized in Figure 13(d).

To better demonstrate the advantages of the proposed system, in

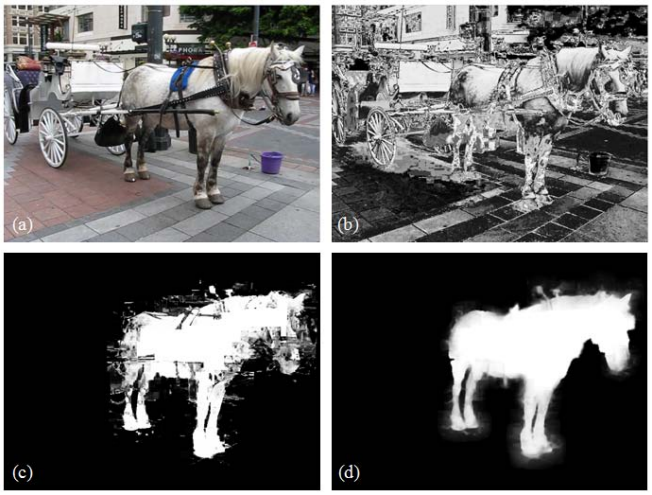

Figure 13: *(a) One frame of the "horse" video. (b) The foreground probability map generated by global GMM color models (color samples are drawn from the previous manually-segmented frame). White means high foreground probability. (c) Our localized color models greatly improve the classification result. (d) After integrating the shape priors, an accurate pixel-wise feature map is achieved in our system.*

Figure 14 we directly compare our system with the interactive Cutout system [Wang et al. 2005][4] and the geodesic video segmentation system [Bai and Sapiro 2007], on two different 9-frame examples. For each system, all the user inputs are visualized for a fair comparison. These examples clearly show that both previous systems present large segmentation errors on these complicated examples, even with relatively dense user-scribbles. In comparison, our system is able to achieve high quality segmentation results with significantly less user input. These examples also give a good hint as to how much total user interaction is needed for segmenting each example video with our system.

Figure 15: *Comparison of segmentation with (bottom row) and without (top row) using the learned shape priors. First frame is the keyframe provided by the user. Original video courtesy of Artbeats.*

The object cut-and-paste system [Li et al. 2005] employs a tracking-based local correction/segmentation tool which is somewhat similar to our local classifiers. However since only color features are used there, it cannot achieve high quality local segmentation on examples with confusing color distributions. Figure 15 shows a comparison of segmenting such a video sequence with and without using

[4] Only 2D foreground and background strokes are used, since the comparison is focused on the segmentation quality.

Figure 14: *Comparison of the geodesic segmentation system, interactive cutout system, and the proposed system on two different 9-frame examples (Original videos courtesy of Mike Wilbur (above) and Artbeats (below)). All user inputs are visualized. For our system the user-provided keyframe masks are highlighted in red border. Zoom-in into the image for a better appreciation of the segmentation differences and the significant improvements obtained with the proposed system.*

the shape priors. It clearly demonstrates that learned shape priors are very important for achieving a consistent segmentation when color distributions are complicated.

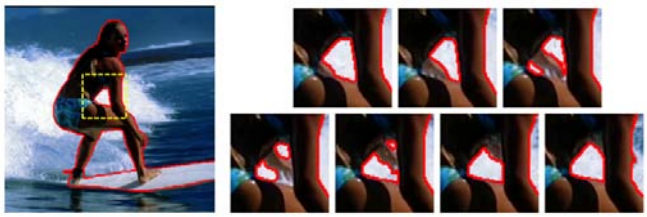

Figure 16: *Segmentation results (red curves) of a local region of the "surfing" video from frame* 20 *to frame* 26 *with no user interaction. Note how the topology changes are correctly handled.*

Figure 16 shows a local region of the "surfing" example where rapid topology changes occur, our system handles this without any user assistance. In contrast, rotoscoping systems (i.e., [Agarwala et al. 2004]) need a lot of user input to handle such topology changes.

Finally, Figure 17 shows four additional sets of examples of video cutout, together with some special editing effects.

Our system is computationally efficient and operates in an interactive fashion for binary segmentation. The average computing time of each step of the one-frame propagation and segmentation process, using our non-optimized research system on a Macbook Pro laptop machine with an Intel Core 2 Duo CPU at 2.8GHz and 2G memory, is: SIFT matching 0.20s, Optical flow 0.22s, feature integration 0.57s, and Graph Cut segmentation 0.09s, for a total of 1.08s for a 720×576 frame. The system does not involve any pre-processing, and the current computational cost in SIFT matching and optical flow calculation can be potentially shifted into an automatic pre-processing step. Our system is also efficient in terms of memory consumption since segmentation is achieved progressively without any global optimization. For the automatic matting step, our non-optimized implementation runs at about 2 seconds per frame.

7 Limitations and Discussions

Being an interactive system, our approach usually achieves good results even for difficult examples when enough user assistance has been provided. However, we consider our proposed system to "fail" if too much user correction is needed. For example, the "foreground" object being segmented might become background relative to an object even closer to the camera (e.g. when the object passes behind a telephone booth), and the system currently does not try to detect/correct for this. Also the foreground color model assumption does not really expect the new colors introduced at the edge of an object that is spinning on its own axis. The system does not deal with objects without a clear shape such as smoke or fire, for which the use of shape prior will not help but only hurt the segmentation.

Severe motion blur or defocus blur can cause weak foreground boundaries, thus the Graph Cut segmentation is more likely to snap to a nearby strong image edge (although the shape prior can partially eliminate this error). In this case more user assistance is often required to get accurate segmentation. Figure 18 shows such an example where severe motion blur, defocus blur, and occlusion oc-

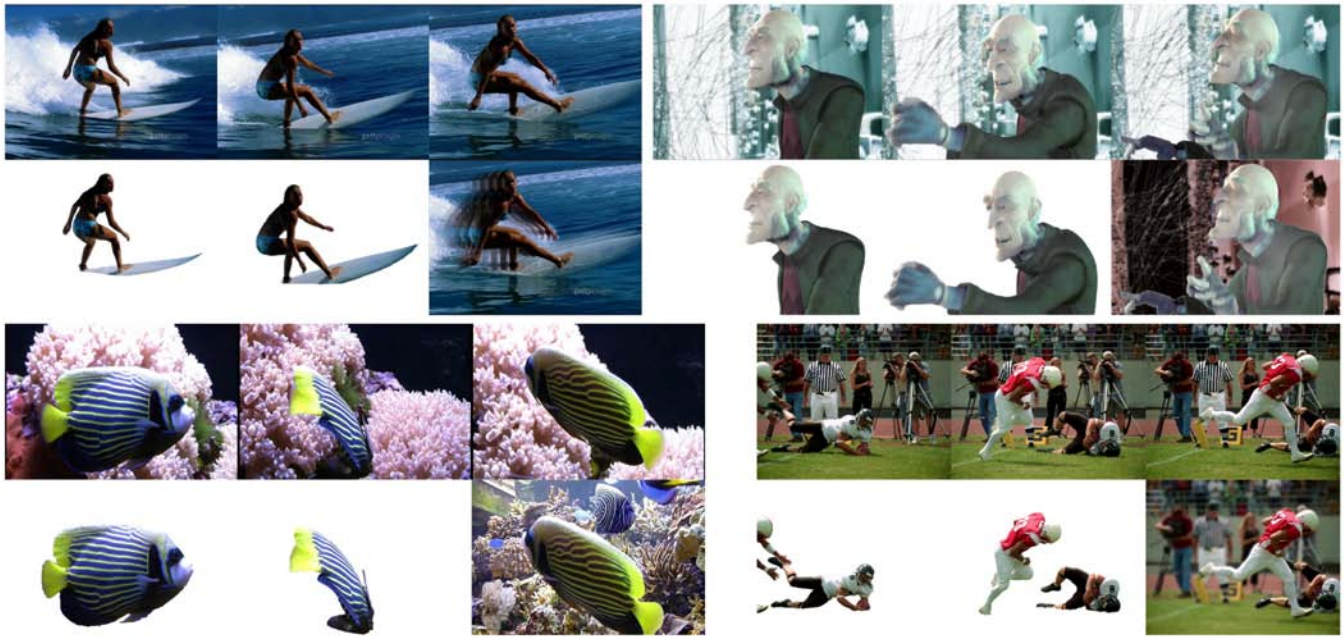

Figure 17: *Video SnapCut examples and applications. For each of the four examples, the sub-figures in the top row correspond to three original frames, followed in the bottom row by two corresponding white background composites and a different special effect. The special effects are, clockwise from top-left, delayed-fading foreground, inverted background, new background, and blurred background (original video courtesy of Artbeats). See also the accompanying video.*

Figure 18: *A difficult example with severe blur. From left to right: frames 2, 16, 18, 19 and 20, along with limited user edits and binary segmentation boundaries generated by our system, followed by zoom-in view of less accurately segmented defocus blur (green) and motion blur (purple).*

cur simultaneously.[5] With more user input, our system is still able to generate a relatively accurate binary segmentation, but one can clearly see how the foreground boundaries are affected by nearby background textures. A better solution is to replace the Graph Cut segmentation with an alpha matting component, thus the soft edges caused by blur can be correctly recognized and handled. Also, rough motion estimation across frames can help determining where the motion blur is most likely to happen.

Sudden illumination changes can cause the historic color models M_c^h described in Section 2.3 to be inaccurate, thus the user has to either start a new keyframe, or provide more local correction strokes to force the system pick up new colors. Figure 19 shows such an example.Without any user assistance, the system is not be able to recognize the illumination change automatically, which leads to severe segmentation errors. Although in the current system the problem can be solved with additional user input (see Figure 19 bottom), more advanced color modeling and feature extraction techniques, for instance, the recently proposed illumination-invariant color space [Chong et al. 2008], can potentially be employed in the system to make it more robust against such changes.

[5]Original videos shown in Figure 18 and Figure 19, along with our segmentation results and user edits, are available as supplemental material to this paper in the ACM Digital Library. For the availability of other videos shown in the paper, please contact the authors.

8 Conclusion

We propose a novel video cutout system which, unlike previous systems, uses adaptive local classifiers for segmenting dynamic video objects. By localizing the classifiers, our system achieves significantly better results than previous systems for complicated videos, including complex color distributions, dynamic backgrounds and non-rigid foreground deformations.

The system can be further improved by adding features such as texture, computed via multiscale/orientation filters. For static backgrounds, background substraction and global camera motion can help as well. More advanced coherent matting methods are needed to deal with dynamic fuzzy foreground boundaries such as hair blowing in the wind. The incorporation of such enhancements is part of our current efforts.

Acknowledgements

We thank the anonymous reviewers for helping to improve this paper. We thank Artbeats (www.artbeats.com), Mike Wilbur, Jon Goldman, and Jiawen Chen for permitting us to use various video footage shown in the paper and the supplemental material. Xue Bai and Guillermo Sapiro are partially supported by NSF, NGA, ONR, ARO, and DARPA.

Figure 19: *An example with sudden illumination changes. Top: if only one keyframe (frame 15) is provided before the illumination change, the segmentation results rapidly deteriorate during the change. Bottom: A small amount of additional user input can help the system correctly deal with the illumination change.*

References

ADOBE SYSTEMS. 2008. Adobe Photoshop CS4 User Guide.

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

ARMSTRONG, C. J., PRICE, B. L., AND BARRETT, W. A. 2007. Interactive segmentation of image volumes with live surface. *Computers and Graphics 31*, 2, 212–229.

BAI, X., AND SAPIRO, G. 2007. A geodesic framework for fast interactive image and video segmentation and matting. In *Proc. of IEEE ICCV*.

BLAKE, A., AND ISARD, M. 1998. *Active Contours*. Springer-Verlag.

BOYKOV, Y., VEKSLER, O., AND ZABIH, R. 2001. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Analysis and Machine Intelligence 23*, 11, 1222–1239.

CHONG, H., GORTLER, S. J., AND ZICKLER, T. 2008. A perception-based color space for illumination-invariant image processing. In *Proc. of ACM SIGGRAPH*.

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

KOHLI, P., KUMAR, M. P., AND TORR, P. H. S. 2007. P3 & beyond: solving energies with higher order cliques. In *Proc. of IEEE CVPR*.

KOMOGORTSEV, O., AND KHAN, J. 2004. Predictive perceptual compression for real time video communication. In *Proc. of the 12th Annual ACM Int. Conf. on Multimedia*, 220–227.

LEVIN, A., LISCHINSKI, D., AND WEISS, Y. 2008. A closed-form solution to natural image matting. *IEEE Trans. Pattern Analysis and Machine Intelligence 30*, 2, 228–242.

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

LI, Y., SUN, J., AND SHUM, H. 2005. Video object cut and paste. In *Proc. ACM SIGGRAPH*, 595–600.

LI, Y., ADELSON, E., AND AGARWALA, A. 2008. Scribbleboost: Adding classification to edge-aware interpolation of local image and video adjustments. In *Proc. of EGSR*, 1255–1264.

LOWE, D. G. 2004. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision 60*, 2, 91–110.

MORTENSEN, E., AND BARRETT, W. 1995. Intelligent scissors for image composition. In *Proc. of ACM SIGGRAPH*, 191–198.

PROTIERE, A., AND SAPIRO, G. 2007. Interactive image segmentation via adaptive weighted distances. *IEEE Trans. Image Processing 16*, 1046–1057.

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

STEWART, S., 2003. Confessions of a roto artist: Three rules for better mattes. http://www.pinnaclesys.com/SupportFiles/Rotoscoping.pdf.

WANDELL, B. 1995. *Foundations of Vision*. Sinauer Associates.

WANG, J., AND COHEN, M. 2007. Image and video matting: A survey. *Foundations and Trends in Computer Graphics and Vision 3*, 2, 97–175.

WANG, J., AND COHEN, M. 2007. Optimized color sampling for robust matting. In *Proc. of IEEE CVPR*.

WANG, J., XU, Y., SHUM, H., AND COHEN, M. 2004. Video tooning. In *Proc. of ACM SIGGRAPH*.

WANG, J., BHAT, P., COLBURN, A., AGRAWALA, M., AND COHEN, M. 2005. Interactive video cutout. In *Proc. of ACM SIGGRAPH*.

WANG, J., AGRAWALA, M., AND COHEN, M. 2007. Soft scissors: an interactive tool for realtime high quality matting. In *Proc. of ACM SIGGRAPH*.

WASCHBSCH, M., WRMLIN, S., AND GROSS, M. 2006. Interactive 3d video editing. *The Visual Computer 22*, 9-11, 631–641.

YU, T., ZHANG, C., COHEN, M., RUI, Y., AND WU, Y. 2007. Monocular video foreground/background segmentation by tracking spatial-color Gaussian mixture models. In *Proc. of WMVC*.