2009-10-01

LIVEcut: Learning-based Interactive Video Segmentation by Evaluation of Multiple Propagated Cues

Bryan S. Morse
morse@byu.edu

Brian L. Price

Scott Cohen

Original Publication Citation

B. Price, B. Morse, and S. Cohen, "LIVEcut: Learning-based interactive video segmentation by evaluation of multiple propagated cues," in IEEE International Conference on Computer Vision (ICCV), pp. 779-786, October 29.

BYU ScholarsArchive Citation

Morse, Bryan S.; Price, Brian L.; and Cohen, Scott, "LIVEcut: Learning-based Interactive Video Segmentation by Evaluation of Multiple Propagated Cues" (2009). *Faculty Publications*. 121.
https://scholarsarchive.byu.edu/facpub/121

LIVEcut: Learning-based Interactive Video Segmentation by Evaluation of Multiple Propagated Cues

Brian L. Price
Brigham Young University
bprice@rivit.cs.byu.edu

Bryan S. Morse
Brigham Young University
morse@byu.edu

Scott Cohen
Adobe Systems
scohen@adobe.com

Abstract

Video sequences contain many cues that may be used to segment objects in them, such as color, gradient, color adjacency, shape, temporal coherence, camera and object motion, and easily-trackable points. This paper introduces LIVEcut, a novel method for interactively selecting objects in video sequences by extracting and leveraging as much of this information as possible. Using a graph-cut optimization framework, LIVEcut propagates the selection forward frame by frame, allowing the user to correct any mistakes along the way if needed. Enhanced methods of extracting many of the features are provided. In order to use the most accurate information from the various potentially-conflicting features, each feature is automatically weighted locally based on its estimated accuracy using the previous implicitly-validated frame. Feature weights are further updated by learning from the user corrections required in the previous frame. The effectiveness of LIVEcut is shown through timing comparisons to other interactive methods, accuracy comparisons to unsupervised methods, and qualitatively through selections on various video sequences.

1. Introduction

Video segmentation is an essential process in many video applications. It is required for video editing and special effects whenever objects must be moved, deleted, individually edited, or layered. It is also used in object recognition, 3D reconstruction from video, and compression. Despite recent research in the area, industry still largely relies on chroma keying and manual rotoscoping, emphasizing the need for an effective, easy-to-use video segmentation tool.

This need remains due to the surprising difficulty of the problem. Video segmentation shares the difficulties of image segmentation, such as overlapping color distributions, weak edges, complex textures, and compression artifacts. In addition to these challenges, video may contain erratic camera and/or object movement, motion blur, and occlusions. Objects may move enough that there is no overlap between successive frames. Other moving objects may cause confusion. Lighting changes and shadows alter the color distributions, and movements in 3D space may greatly change an object's 2D projected boundary. A given video sequence can easily exhibit many of these challenges.

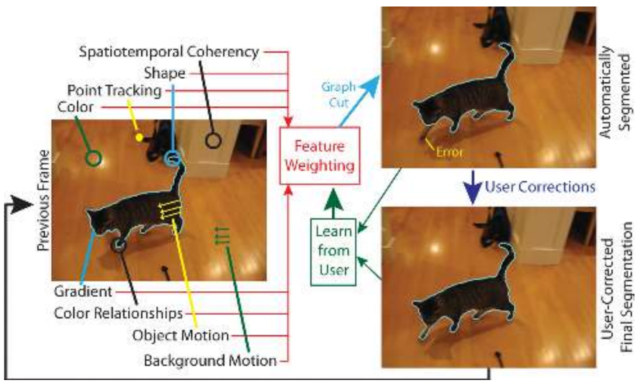


Figure 1. From an initial segmented frame, a variety of features are extracted. These are automatically locally weighted based on estimated correctness and used to segment the next frame. If errors occur, the user may correct them, and the system learns which features are providing good information. The corrected frame is used to continue propagating the segmentation.

Many different kinds of information can be gleaned from successive video frames to aid object selection. Such features include color, gradient, adjacent color relationships, shape, spatiotemporal coherence, camera motion, object motion, and trackable points. The relative importance of the cues differs depending on the sequence, the frame, and even the location in the frame. For example, in Figure 1 a color model can easily distinguish the cat from the light brown floor but would struggle separating the tail from the similarly-colored bag. A shape feature, however, could separate the tail and bag. An algorithm that intelligently applies all of these cues based on specific circumstances will perform better than one relying only on a subset of these cues or on a static combination of all of them.

Despite the importance of each kind of information,

2009 IEEE 12th International Conference on Computer Vision (ICCV)
978-1-4244-4419-9/09/$25.00 ©2009 IEEE

most current algorithms do not use all these features. Algorithms that segment the video as a spatiotemporal volume [2, 3, 5, 22] can generally only extract information from the pixels under the user strokes to model the foreground and background. These methods have no information about some of these features such as shape or boundary information, and have limited knowledge of other features such as foreground and background color. By allowing the user to segment one frame and then propagating this information to other frames, these features can be used.

In this paper, we introduce LIVEcut, a frame-by-frame interactive video segmentation method designed to maximize the information propagated from one frame to the next. As shown in Figure 1, LIVEcut extracts various features, locally weights them based on likely effectiveness, and resolves them using graph-cut optimization. LIVEcut also learns automatically from user corrections how well each cue performed and weights their importance accordingly. Our local weighting allows LIVEcut to selectively apply the cues that will most effectively segment the object. Contributions are also made in the extraction of many of the individual cues. These include full foreground and background local color models, color adjacency models, separate foreground and background motion models, point tracking information, and a new shape prior.

2. Related Work

Many approaches have been taken in interactive video segmentation. Some approaches focus on either boundary or region information only. Agarwala et al. [1] performs boundary tracking using splines that follow object boundaries between keyframes using both boundary color and shape-preserving terms. Bai and Sapiro [3] use region color to compute a geodesic distance to each pixel to form a selection. These approaches perform well when a single type of cue is sufficient for selecting the desired object.

Many current techniques use graph cut to segment the video as a spatiotemporal volume. Graph cut, as formulated in [5], solves for a segmentation by minimizing an energy function over a combination of both region and boundary terms. It has been shown to be effective in the segmentation of images [11, 15] and volumes [2].

Boykov and Jolly [5] introduced a basic approach to segmenting video as a spatiotemporal volume. Their graph connects pixels in a volume, which implicitly includes spatiotemporal coherence information. Graph cut is applied using a region term based on a color model of the pixels under the user strokes and a boundary term based on gradient.

Wang et al. [22] builds on this approach by allowing users to segment video by drawing strokes on arbitrary slices of the spatiotemporal volume. While this permits a user to mark several frames at once, it requires a steep learning curve to know how to carve the volume so that the right pixels are visible along the slice. The method uses a global color model based on the user strokes as well as a local color model for static backgrounds in addition to gradient values.

In Li et al. [10], users segment every tenth frame, and graph cut computes the selection between the frames using global color models from the key-frames, gradient, and coherence as its primary cues. The user may also manually indicate areas to which local color models are applied. While this method performs well, it requires the manual segmentation of many frames in addition to corrections.

In methods where the video is treated as a spatiotemporal volume [2, 3, 5, 22], the only information known for certain about the object and background are in the user-marked pixels. This provides very limited knowledge about the object interior and no knowledge about the boundary. While [10] is an exception to this, it requires the user to manually segment many frames. These methods contrast our own, where frame-by-frame propagation allows for the computation of complete features.

In parallel with our own work, Yin and Collins [26] proposed an automated video segmentation system that includes color, gradient, color adjacency, and shape information in a graph cut framework. They dynamically reweight these terms from frame to frame, but do so on a global basis without regard to user corrections.

Some unsupervised video segmentation methods have also combined various cues [6, 19, 25]. While unsupervised techniques generally perform well at roughly separating motion layers, they do not produce the high-quality results required for many applications. The object of interest may also not correspond to a motion layer, leaving these methods incapable of generating the desired result.

3. LIVEcut Video Segmentation

While the methods described in Section 2 provide good means of segmenting video, each relies only on a few cues to make decisions. LIVEcut extracts much more information about the sequence and uses this to improve the segmentation. The user marks the object in the first frame of the sequence using the stroke-based method employed by most graph-cut methods, and LIVEcut propagates various cues taken from the full frame to the next frame as described in this section. These cues are automatically weighted locally and resolved using graph-cut optimization (Section 4). As the user proceeds through the sequence, the implicit verification of the previous frame allows LIVEcut to use the entire previous frame once again to segment the current frame.

3.1. Graph cut framework

Before explaining the specific features we propagate from frame to frame, we present the overall framework in which the features are resolved. For this, we use minimum graph-cut optimization. Graph cut computes a segmentation over a set of pixels P by minimizing the equation

$$E(\mathcal{L}) = \sum_{x_i \in P} R(x_i, \mathcal{L}_i) + \lambda \sum_{(x_i,x_j)\in N} B(x_i, x_j)\ |\mathcal{L}_i - \mathcal{L}_j| \quad (1)$$

where $\mathcal{L} = (\mathcal{L}_i)$ is a binary vector of labels and $\mathcal{L}_i$ is the label (0 for background, 1 for foreground) for pixel x_i, $R(x_i, l)$ is a region cost term based on the label l, $B(x_i, x_j)$ is a boundary cost term, λ is a relative weighting of R and B, and N is the set of pairs of neighboring pixels.

Our region term $R(x_i, l)$ is the sum of all cues that apply to an individual pixel. Given a set of unary cues U,

$$R(x_i, l) = s(x_i, l) + \sum_{u \in U} \alpha_u(x_i)\ w_u(x_i, l) \qquad (2)$$

where $w_u(x_i, l)$ is the cost of labeling pixel x_i with label l according to cue u, $\alpha_u(x_i)$ is a scalar giving the certainty of w_u at x_i, and $s(x_i, l) = 0$ if the pixel was labeled l by a user stroke and ∞ if labeled $\bar{l}$ (the other label).

Our boundary term $B(x_i, x_j)$ is given by

$$B(x_i, x_j) = w_a(x_i, x_j)\ w_g(x_i, x_j). \qquad (3)$$

and encourages selection boundaries in the current frame to occur at image edges with similar color profiles to the selection boundaries in the previous frame. The unary terms (color w_c, spatiotemporal coherency w_h, shape w_s, and point tracking w_p) and binary terms (gradient w_g and color adjacency w_a) are defined in Sections 3.3-3.8.

In order to increase the speed of the algorithm, we apply our algorithm to an oversegmentation of the image produced using [20]. The segmentation is then refined on the pixel level similar to [11]. While the following terms are defined according to pixels, they can all be directly extended to oversegmented regions.

3.2. Object and Background Motion

Motion is an important cue in video segmentation. By considering the motion of the object, more precise local information may be used to segment it. By removing camera motion, better local information can be used for the background where it is static.

Many methods account for the camera motion by aligning the frames in a preprocessing step [10, 22]. However, since the foreground object will often exhibit different motion patterns than the background, aligning the background will not correctly align the foreground.

Since we know the segmentation of the previous frame, we can align the foreground and background separately. The background is aligned by locating good points to track [17], then computing and applying a homography.

While the foreground can be tracked in the same manner, problems can occur if the foreground does not have enough trackable points to generate a good homography due to large movements or little texture. To account for these cases, we use a novel method to roughly align the foreground.

We use an iterative closest point-style algorithm [4] to match pixels x_i in the selection M on the current frame I to pixels y_j in the next frame I_{next} with one affine transformation A. The iteration alternates between (a) finding the best matches $\{(x_i, y_{m(i)})\}$ for a given A, and (b) finding the best A to align matches $\{(x_i, y_{m(i)})\}$. In (a), we match points in (xy position $\times$ RGB color) space so that points in M are matched to points in I_{next} that are similar in color and position after applying A. For each $x_i \in M$, we solve a nearest neighbor problem $y_{m(i)} = \arg\min_{y_j} ||(Ax_i, \gamma I(x_i)) - (y_j, \gamma I_{\text{next}}(y_j))||_2^2$ ([14]), where RGB values are in $[0, 1]$ and γ is the sum of the frame width and height. For (b), we solve $A = \arg\min_A \sum_{i=1}^{n} ||Ax_i - y_{m(i)}||_2^2$ ([16]).

The object and background motions are not included as a term in graph cut. Rather, they are used to spatially transform the locality information of the other cues. While this transformation does not completely capture non-rigid motion, it improves the locality of the foreground information and works well in practice.

3.3. Gradient

Image gradients are important for encouraging selection boundaries to fall on image edges. As in [11], we use color difference as a boundary term:

$$w_g(x_i, x_j) = \frac{1}{||C(x_i) - C(x_j)||^2 + 1}. \qquad (4)$$

where $C(x_i) \in [0, 255]^3$ is the color at x_i. Gradient boundary terms are standard practice in graph-cut segmentation.

3.4. Color

A color-model region term encourages pixels to be labeled according to the color distribution of the model. Because most graph-cut algorithms [5, 11, 22] do not have access to a full segmentation of a frame, only the pixels under the user strokes are used to create the model. This limited sample does not always accurately represent the color properties of the image. These algorithms must also by necessity use a global color model, which does not differentiate colors located in different regions of the image. While [10] can use a local color model, it only does so over a small window if manually indicated by the user.

A contribution of LIVEcut is that it uses a *local* color model generated from the *entire* previous frame, which can distinguish between colors in different regions of the image. Such a color model is shown in Figure 2a, where the cat is likely foreground while the similarly-colored backpack and rope are not. The local color model is generated by creating a (l, u, v, x, y) vector p_i for each pixel x_i in the

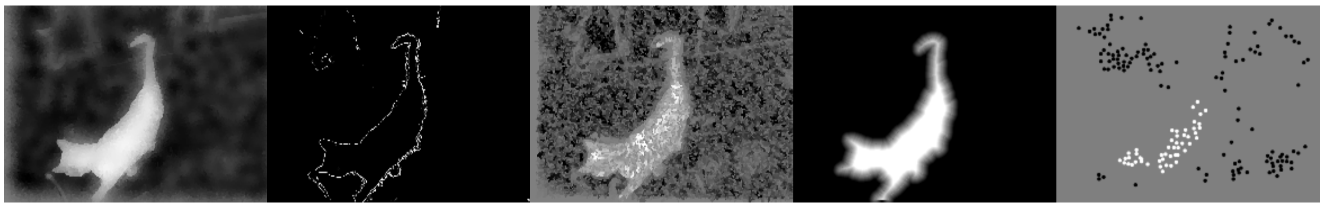


Figure 2. Visualization of the graph-cut terms for the frame with a cat from Figure 6. White indicates foreground likelihood, black background, and mid-gray neutral, except for the color adjacency, where white indicates an object boundary and black indicates no boundary.

previous frame (where (l, u, v) is the color and (x, y) is the motion-adjusted location). The probability of the pixel being foreground is then computed by a 5D Fast Gauss Transform [24]. The probability is assigned to the cost term by

$$w_c(x_i, l) = P(p_i|\bar{l}) \tag{5}$$

where $P(p_i|\bar{l})$ is the normalized probability of the location and color of x_i given the label $\bar{l}$.

3.5. Color adjacency

Not only are the colors indicative of the objects, but the relationship of adjacent colors is as well. Certain color pairs may only exist within the object (background), while others only cross the object boundary. For example, the ballerina in Figure 6 contains a strong red-to-black edge in her clothing that only exists within her interior and never across her boundary. Ideally, a method should distinguish which transitions exist along the boundary and which do not.

While some methods have modeled the color profile of the object edge, such as [13], they do not handle strong gradients within objects where a cut could occur. Cui et al. [7] modifies gradient strength based on color relationships but requires the color to be heavily quantized and does not specify exactly how the locality of edges is implemented.

We introduce a new color-adjacency model to weight the importance of image gradients. The model is computed using a Fast Gauss Transform [24], similar to the color model. Adjacent pixels are represented by an 8D vector $e_{ij} = (l_i, u_i, v_i, l_j, u_j, v_j, x, y)$ where (l_i, u_i, v_i) is the color of pixel x_i, (l_j, u_j, v_j) the color of the x_j, and (x, y) their motion-adjusted location. A model I is generated for all edges that are in the interior of either the foreground or background, and another model B is generated for all edges along the boundary. These probabilities are combined into a boundary reweighting factor by

$$w_a(x_i, x_j) = \left(1 + \left|\frac{P(e_{ij}|I) - P(e_{ij}|B)}{P(e_{ij}|I) + P(e_{ij}|B)}\right|\right)^{2\eta} \tag{6}$$

where $P(e_{ij}|l)$ is the probability of e_{ij} given the label l. η gives the sign of the numerator: 1 if $P(e_{ij}|I) \geq P(e_{ij}|B)$ and -1 otherwise. Equation 6 creates a scalar ranging from 0.25 if the model indicates a pure boundary ($P(e_{ij}|I) = 0, P(e_{ij}|B) = 1$) to 4 for a pure interior edge ($P(e_{ij}|I) = 1, P(e_{ij}|B) = 0$), with a factor $w_a = 1$ for equal interior and boundary probabilities ($P(e_{ij}|I) = P(e_{ij}|B)$). Figure 2b shows the effect of the color adjacency model. The cat's outline is clearly highlighted as the desired boundary, while other edges are suppressed.

3.6. Spatiotemporal Coherency

Videos usually exhibit a high amount of coherency between frames. Spatiotemporal-volume approaches [10, 22] implicitly capture this coherency through edges across frames. With our frame-by-frame approach, coherency between frames can be included without explicitly representing the labeled pixels from the previous frame. Rather, we assign a high region cost to label x_i as l if there is a nearby pixel (after motion adjustment) in the previous frame labeled $\bar{l}$ that has a similar color:

$$w_h(x_i, l) = \sum_{y_j \in N_{\bar{l}}(x_i)} \frac{1}{||C(x_i) - C(y_j)||^2 + 1} \tag{7}$$

where $N_{\bar{l}}(x_i)$ is the set of all neighbors of x_i from the previous frame that are labeled $\bar{l}$. Figure 2c shows the cost map for the spatiotemporal coherency where the cat is likely foreground since it overlaps with the previous frame. The blockiness is due to the oversegmentation regions.

3.7. Shape

When an object passes over a similarly colored background, no edge exists upon which to place the boundary. In these cases, the shape of the object is vital. Including a shape term in the features can handle such cases.

Recently there has been interest in including shape priors into graph cut [8, 9, 18, 21].The common approach is to align the shape to the image by user interaction and/or automated means, and then include a term in the cost function based on distance to the shape or a mismatch score.

In LIVEcut, because we have tracked the object motion forward, we already have an estimate of the motion-adjusted object shape Φ (where $\Phi(x_i) = 1$ if x_i is in the object mask and 0 otherwise) and its boundary Ω (where

$\Omega = \partial\Phi$). We compute the distance from each pixel to the boundary after adjusting for object motion using

$$d_\Omega(x_i) = \min_{p \in \Omega}(||p - x_i||). \quad (8)$$

Our shape term is an extension to [21] but takes distance into account:

$$w_s(x_i, l) = |l - \Phi(x_i)| \min(d_\Omega(x_i)/M, 1) \quad (9)$$

where M is the maximum allowable distance (we use $M = 10$). If the estimated shape mask does not match the labeling of a pixel, this term penalizes the labeling based on the pixel's distance to the predicted shape boundary up to a threshold M. Using a small M, if the boundary is only off by a few pixels, it will have a minimal cost added. This cost function combined with the estimation of the object motion comprise a novel shape prior for graph cut. The resulting costs produced by the shape cue are shown in Figure 2d.

3.8. Point tracking

For most pixels in a typical video sequence, it is difficult to precisely determine the corresponding point in the next frame. However, easily-trackable points give nearly certain information about their labeling (see Figure 2e). While many algorithms make use of such points, video segmentation methods based on graph cut currently do not. We use [12, 17] to track these points and assign a penalty to labeling x_i as l if x_i is within a distance D (we use $D = 5$) of a tracked point that was labeled $\bar{l}$ in the previous frame:

$$w_p(x_i, l) = \begin{cases} 1 & \text{if } d_{\Theta_{\bar{l}}}(x_i) \le D \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

where $\Theta_{\bar{l}}$ is the set of tracked points labeled $\bar{l}$. Any points that were not reliably tracked are removed from Θ_l. We also filter out any points too close to the object boundary (within 10 pixels), because points near the boundary may potentially spill over onto the other side.

4. Automatically Weighting Cues

While a variety of cues can be used for video segmentation, some features will perform more reliably than others given a specific sequence, frame, or even location within the frame. In order to best leverage the various cues, we evaluate and learn from their performance.

We automatically weight the region terms in graph cut on a local basis, as shown in Equation 2 by the α_u factors. In this manner, the most effective cues will have a stronger effect. Each α_u is a combination of an automatic scaling β_u based on the estimated effectiveness of that term locally (Section 4.1) and a weighting ρ_u that is learned through user corrections (Section 4.2):

$$\alpha_u(x_i) = \beta_u(p_i)\, \rho_u(x_i). \quad (11)$$

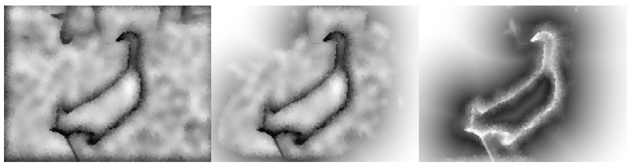


Figure 3. Visualization of the estimated accuracy of the (a) color, (b) spatial coherency, and (c) shape terms. The grayscale value of a pixel indicates the local weight β for that pixel and cue.

4.1. Setting estimated effectiveness

For each region term, LIVEcut assesses its own performance on the previous frame to estimate the accuracy of that feature for each pixel. This ensures that each feature is weighted strongly in the areas where it is most effective.

For the color term, the accuracy can be estimated by applying the model to the frame that generated it (i.e. the previous frame) by

$$\beta_c(x_i) = P^{prev}(p_i | \mathcal{L}_i^{prev}) \quad (12)$$

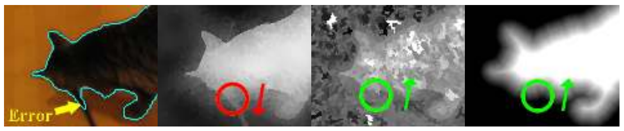
where the superscript $prev$ indicates that the probability and label are from the previous frame and p_i is the (l, u, v, x, y) color and location vector for pixel x_i. Figure 3a shows the estimated effectiveness for one frame of the cat sequence. Note that while the weighting is generally high (shown by brighter pixels), it is lower near where the cat crosses the rope due to the overlapping color models.

The coherency term in Equation 7 returns a large value when a pixel is similar in color to a neighboring pixel of the opposite label in the previous frame. This works well except when near object boundaries where the foreground and background colors are similar, because the costs for each label are then similar. This can be detected by a low probability in the color model near the boundary. We weight the coherency term accordingly:

$$\beta_h(x_i) = \max(P^{prev}(p_i | \mathcal{L}_i^{prev}), \min(d^2_\Omega(x_i)/D^2, 1)) \quad (13)$$

where D is a distance threshold (we use 0.25(image width + height)). This equation includes both a color term and a distance term. Near the boundary, the distance term is small, so the color term dominates, and the weight is high if the foreground and background colors are dissimilar. Far from the boundary, the distance term dominates. This is illustrated in Figure 3b, where near the object the weighting looks similar to that of the color model, while away from the object it resembles a distance map.

The shape term is most important for localizing boundaries where the similarity of foreground and background colors weakens the effectiveness of the color, spatiotemporal, gradient, and color adjacency terms. It is also effective far from the boundary where the labeling is more certain.



(a) Error (b) Color (c) Coherency (d) Shape

Figure 4. (a) An error occurred in the propagated segmentation. Since the (b) color cue was incorrect, its weight is decreased. The (c) coherency and (d) shape cues were correct, so they are increased. The grayscale value in (b-d) visualizes the label the cue suggests, with white indicating foreground and black background.

Based on these ideas, we weight the shape term using

$$\beta_s(x_i) = \max(1-P^{prev}(p_i|\mathcal{L}_i^{prev}), \min(d_\Omega^2(x_i)/D^2, 1)). \quad (14)$$

Note the similarity to $\beta_h(x_i)$ in Equation 13, except that the color probability has been subtracted from 1. This can be seen in Figure 3c, where the shape term looks like the coherency term except that it is inverted near the object. The shape prior will thereby be more heavily weighted in areas where color does *not* effectively identify the boundary, such as the cat and rope overlap in Figure 3.

For the point-tracking term, we have already removed points that were not reliably tracked, so we have high confidence in the remaining points. We therefore weight all points equally: $\beta_p(x_i) = 1$.

4.2. Learning from user corrections

While the automatic weighting usually works well, it may be incorrect at times and require further user correction. LIVEcut handles this automatically by learning from the corrections. The user corrects any mistakes by marking them with strokes before proceeding to the next frame. Since each region term gives a value in favor of the foreground $\mathcal{F}$ and the background $\mathcal{B}$, each suggests a label for each pixel. More precisely, if $w_u(x_i, \mathcal{F}) - w_u(x_i, \mathcal{B}) > 0$, then the term w_u would label x_i as background on its own, and vice versa. By comparing the initial propagated selection to the selection after corrections, we can determine which features were correct at each pixel and use that to weight their future performance. In Figure 4, since the color weight for foreground $w_c(x_i, \mathcal{F})$ was greater than the background weight $w_c(x_i, \mathcal{B})$, the future weight of those color terms are weakened. The coherency and shape terms suggested the correct labeling and are strengthened.

We initialize ρ_u to a constant value for all x_i. Let S_u^i be the initial propagated segmentation suggested by term w_u alone, and let S^f be the final segmentation after the user has corrected any mistakes. If $S_u^i(x_i) \neq S^f(x_i)$, w_u suggested an incorrect labeling for x_i and its weight should be discounted by ρ_u in the next frame,

$$\rho_u^{next}(x_i) = \begin{cases} \rho_u(x_i) + \delta_0 & \text{if } S_u^i(x_i) = S^f(x_i) \\ \rho_u(x_i) - \delta_1 & \text{if } S_u^i(x_i) \neq S^f(x_i) \end{cases} \quad (15)$$

Video	Size	Graph Cut Time	User Time
Bass Guitar	960×540×72	0.055 / 3.53 sec	38 min
Cat	640×480×56	0.038 / 1.88 sec	5 min
Flamingo	960×540×76	0.055 / 2.82 sec	30 min
Footballer	720×576×19	0.053 / 1.99 sec	5 min
Lemurs	960×540×86	0.047 / 3.11 sec	36 min

Table 1. Timing results from several sequences. The graph-cut time first gives the time to process an interaction on one frame, and then the time to propagate information to the next frame. "Footballer" is courtesy of Artbeats (www.artbeats.com).

where all segmentations S are from the current frame. δ_0 and δ_1 are constant increments for ρ_u. We use $\delta_0 = 0.4$ and $\delta_1 = 0.8$, and ρ is initialized to 1.

5. Results

For an interactive segmentation system, the real measure of success is the amount of user time required to perform a selection. Accordingly, we report the time required to select objects in several sequences. The segmentations can be judged qualitatively by the examples shown in Figure 6. In order to better evaluate the accuracy of LIVEcut, we also compare it to automatic segmentation techniques, despite the disadvantage this gives to an algorithm designed for interactive use.

5.1. Timing and Qualitative Results

Table 1 gives timing results over several challenging video sequences. The "footballer" sequence exhibits large motions, a drastically changing object shape, and a partial occlusion from another moving object. "Bass guitar" and "lemurs" both contain overlapping color models, boundaries where there is no gradient information, and motion blur. While much of the body of the "flamingo" is easy to segment, the legs are narrow, exhibit large movements, are often heavily blurred, and have a similar color to the background. Using LIVEcut, a user is able to segment the objects without excessive interaction. The selection from several frames of these sequences can be seen in Figure 6. We apply the robust matter [23] to our output to account for mixed pixels on boundaries.

We compare LIVEcut to [22] using videos from this paper in Table 2. The user time to acquire binary segmentation results similar in quality to these techniques is comparable or less in these examples. The time the user must wait between each interaction for the selection to update is also less, providing a better interactive experience. Our algorithm also does not need the large preprocessing time that [22] requires. We were able to segment "amira" with LIVEcut, while [22] required the help of [1] to do so. We also were able to segment the "ballerina" as one object, while [22] required one pass for the feet and another for

		Other Techniques					LIVEcut	
Video	Size	Method	Pre-process	Graph Cut Time	User Time	Post-process	Graph Cut Time	User Time
amira	640×480×35(80)*	[22]+[1]	12 min	5 sec	15 min	35 min	0.054 / 1.62 sec	7 min
ballerina	640×480×150	[22]	25 min	11.5 sec	140 min	30 min	0.051 / 1.76 sec	74 min
elephant	720×480×100	[22]	20 min	9.1 sec	40 min	30 min	0.036 / 2.39 sec	38 min
manincap	640×480×150	[22]	30 min	16.5 sec	20 min	35 min	0.034 / 1.86 sec	23 min
stairs	640×480×63(100)*	[22]	20 min	8.5 sec	20 min	30 min	0.028 / 1.71 sec	13 min

Table 2. Comparison of LIVEcut to [22]. The graph-cut time for LIVEcut lists first the time to process an interaction on one frame, and then the time to propagate the selection to the next frame. The '*' indicates that the video we obtained differed in length to that reported in [22] (shown in parentheses). The postprocess time for [22] consists of pixel-level refinement, but also includes matting, which is not reported for the our method. LIVEcut does not need any pre-processing time.

Sequence	41	43	50	51	54
LIVEcut Error %	2.30	5.93	1.07	1.18	0.45
[25] Error %	0.80	0.02	1.31	1.06	0.33
Sequence	56	58	60	IU	JM
LIVEcut Error %	2.47	0.24	14.96	2.96	31.37
[25] Error %	0.93	0.79	6.33	2.56	0.27

Table 3. Comparison of [25] to LIVEcut using automatic segmentation (i.e. *without* allowing user corrections).

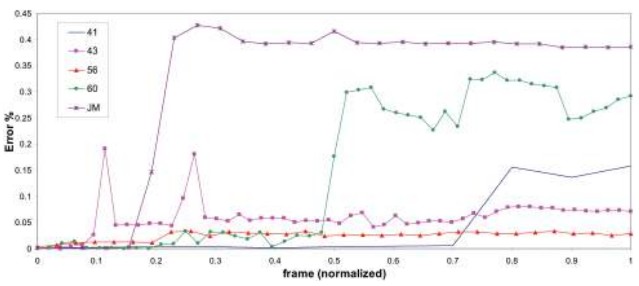


Figure 5. Accuracy of several sequences from Table 3.

the body. Finally, our user interaction is simpler, requiring only drawing strokes on individual frames and allowing sequential processing of the video, while [22] also requires rotating and slicing through a spatiotemporal volume.

5.2. Accuracy and Stability

For interactive segmentation systems, accuracy is difficult to measure since a user can always achieve perfect accuracy given enough time. To demonstrate accuracy, we perform automatic segmentations and compare to the unsupervised method from [25] on their database. In doing so, LIVEcut faces a large disadvantage. LIVEcut was designed to assume that the previous frame was correctly segmented by the user, and proceeds under that assumption. Furthermore, LIVEcut receives no user training, while [25] is trained on similar data. While this test neutralizes many of the strengths of LIVEcut, it allows us to show the algorithm's accuracy and stability.

For this test, we segmented the first frame of ten sequences, each of size 320×240 with an average length of over 350 frames. We then computed the segmentation *without* additional user interaction and compare to the results from [25] in Table 3. For several of the videos (50, 51, 54, 58, IU), we have comparable or better results. For the others, the accuracy over time is shown in Figure 5. Our segmentation error in each case was very low until an abrupt increase due to a change in the scene. For three of the cases, the error is low until the subject moves his hand in front of his body. In these cases, LIVEcut assumes that the hand is an occluding object that it should not segment and does not recover the entire object once the hand leaves. In the other cases, a rapid motion confuses our algorithm. Note that in each case, the error is quite stable after the initial mistake because LIVEcut accurately tracks what it assumes is the new state of the object.

Sequence	41	43	56	60	JM
LIVEcut Error %	0.55	1.41	1.37	3.17	2.26
# frames corrected	1	5	1	2	7

Table 4. Accuracy of LIVEcut on sequences from Table 3 after corrections on the number of frames shown.

To better show the accuracy and stability on these sequences, we resegmented the video allowing corrections only on or near the frames where large errors occur. Table 4 shows that the accuracy is now similar to or less than [25] while allowing very few corrections. While LIVEcut can achieve similar results to unsupervised methods with little or no corrections, these methods could not produce the high quality results from LIVEcut shown in Figure 6.

6. Conclusion

We have presented a new method for interactively segmenting video sequences by propagating multiple cues from one frame to another. These cues are automatically weighted according to their predicted importance on the specific video sequence being segmented, and are further weighted based on learning from user corrections. Many of the cues also include novel improvements in the context of video segmentation using graph cut.

While propagating multiple weighted cues is effective in segmenting video, further improvements can be made. LIVEcut only uses cues from the previous frame together

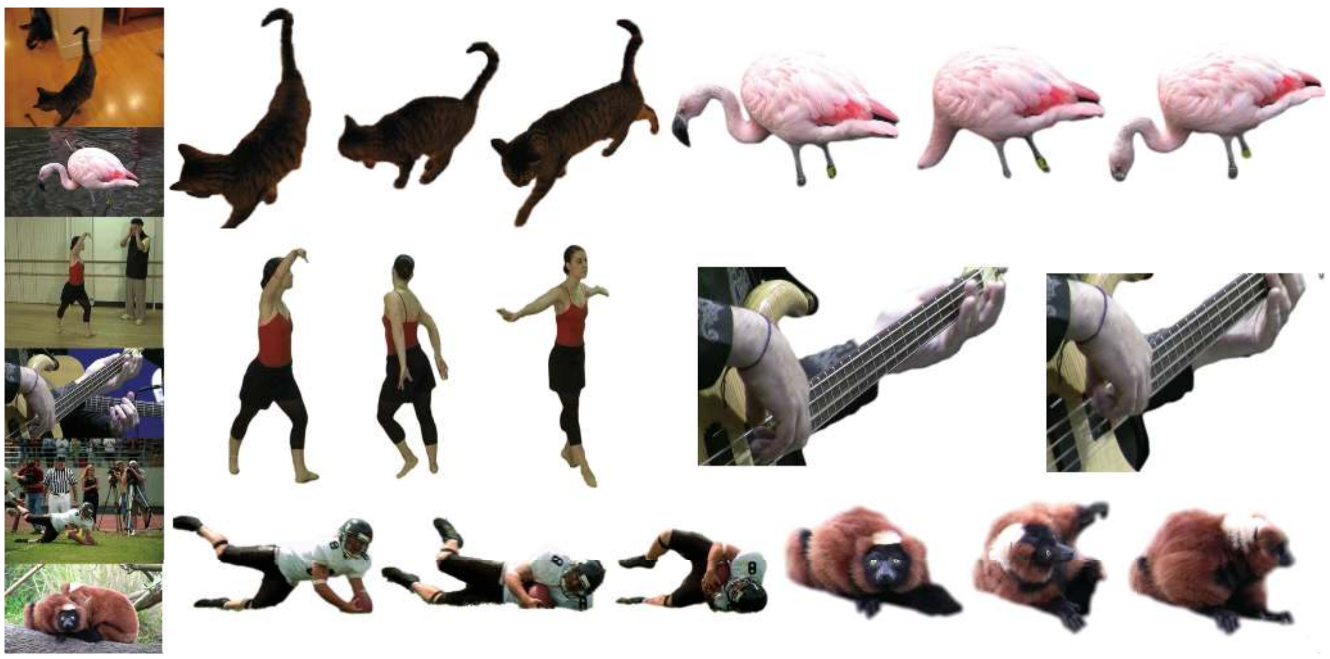


Figure 6. Several examples of object selections using LIVEcut.

with the accumulated learning. However, more global information about the entire video sequence may assist the segmentation. Improved learning techniques may better weight the graph-cut terms.

Acknowledgements

This work was funded by Adobe Systems. We thank William Barrett for his early ideas for this project, and David Simons for his suggestions.

References

[1] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz. Keyframe-based tracking for rotoscoping and animation. *SIGGRAPH*, 23(3):584–591, 2004.

[2] C. Armstrong, B. Price, and W. Barrett. Interactive segmentation of image volumes with live surface. *Computers and Graphics*, 31(2), April 2007.

[3] X. Bai and G. Sapiro. A geodesic framework for fast interactive image and video segmentation and matting. *ICCV*, pages 1–8, 2007.

[4] P. Besl and N. McKay. A method for registration of 3-D shapes. *PAMI*, 14(2):239–256, Feb. 1992.

[5] Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In *IEEE ICCV*, pages 105–112, 2001.

[6] A. Criminisi, G. Cross, A. Blake, and V. Kolmogorov. Bilayer segmentation of live video. *IEEE CVPR 2006*, pages 53–60, 2006.

[7] J. Cui, Q. Yang, F. Wen, Q. Wu, C. Zhang, L. Van Gool, and X. Tang. Transductive object cutout. *IEEE CVPR*, pages 1–8, 2008.

[8] D. Freedman and T. Zhang. Interactive graph cut based segmentation with shape priors. In *IEEE CVPR 2005*, pages 755–762, 2005.

[9] M. Kumar, P. Ton, and A. Zisserman. Obj cut. In *IEEE CVPR*, pages 18–25, 2005.

[10] Y. Li, J. Sun, and H.-Y. Shum. Video object cut and paste. *ACM Trans. Graph.*, 24(3):595–600, 2005.

[11] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum. Lazy snapping. In *ACM SIGGRAPH 2004*, pages 303–308, 2004.

[12] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Int'l Joint Conf. on Artificial Intelligence*, pages 674–679, 1981.

[13] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. *SIGGRAPH*, pages 191–198, 1995.

[14] D. M. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching. http://www.cs.umd.edu/˜mount/ANN/.

[15] C. Rother, V. Kolmogorov, and A. Blake. Grabcut - interactive foreground extraction using iterated graph cuts. In *ACM SIGGRAPH 2004*, pages 309–314, 2004.

[16] S. Schaefer, T. McPhail, and J. Warren. Image deformation using moving least squares. *SIGGRAPH*, 25(3):533–540, 2006.

[17] J. Shi and C. Tomasi. Good features to track. In *IEEE CVPR 1994*, pages 593–600, 1994.

[18] P. Tang and L. Gao. Video object segmentation based on graph cut with dynamic shape prior constraint. In *ICPR08*, pages 1–4, 2008.

[19] A. Thayananthan, M. Iwasaki, and R. Cipolla. Principled fusion of high-level model and low-level cues for motion segmentation. *IEEE CVPR 2008.*, pages 1–8, 2008.

[20] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(6):583–598, 1991.

[21] N. Vu and B. Manjunath. Shape prior segmentation of multiple objects with graph cuts. *CVPR*, pages 1–8, 2008.

[22] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala, and M. F. Cohen. Interactive video cutout. *ACM Trans. Graph.*, 24(3):585–594, 2005.

[23] J. Wang and M. Cohen. Optimized color sampling for robust matting. *IEEE CVPR 2007*, pages 1–8, 2007.

[24] C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis. Improved fast Gauss transform and efficient kernel density estimation. In *IEEE ICCV*, pages 464–471, 2003.

[25] P. Yin, A. Criminisi, J. Winn, and I. Essa. Tree-based classifiers for bilayer video segmentation. *IEEE CVPR*, pages 1–8, 2007.

[26] Z. Yin and R. Collins. Shape constrained figure-ground segmentation and tracking. In *IEEE CVPR*, 2007.