Interactive Video Cutout

Jue Wang[1] Pravin Bhat[1] R. Alex Colburn[2] Maneesh Agrawala[2] Michael F. Cohen[2]

[1]University of Washington [2]Microsoft Research

Figure 1: Our interactive video cutout system makes it easy to extract the foreground objects in these videos of an elephant and a skatebaorder. We then composite these objects onto a third background video to form a new video in which the skateboarder skates on the elephant.

Abstract

We present an interactive system for efficiently extracting foreground objects from a video. We extend previous min-cut based image segmentation techniques to the domain of video with four new contributions. We provide a novel painting-based user interface that allows users to easily indicate the foreground object across space and time. We introduce a hierarchical mean-shift preprocess in order to minimize the number of nodes that min-cut must operate on. Within the min-cut we also define new local cost functions to augment the global costs defined in earlier work. Finally, we extend 2D alpha matting methods designed for images to work with 3D video volumes. We demonstrate that our matting approach preserves smoothness across both space and time. Our interactive video cutout system allows users to quickly extract foreground objects from video sequences for use in a variety of applications including compositing onto new backgrounds and NPR cartoon style rendering.

Keywords: Interactive video processing, min-cut, graph-cut, mean-shift segmentation, alpha matting

1 Introduction

We have recently seen new innovations for interactively extracting foreground objects from an image [Rother et al. 2004; Li et al. 2004]. The user guides these systems by painting strokes that coarsely indicate foreground and background regions in the image. Inspired by these techniques, we have developed an interactive system to cutout dynamic foreground objects from a video sequence.

Operating on each frame of a video individually using the previous techniques would fail in two ways. Painting foreground and background strokes on every frame would be tedious for the user. In addition slight differences in the extraction from frame to frame would lead to a lack of temporal coherence. Another difference between video and a single image is that within a single shot, each video frame is usually highly correlated with nearby frames. We will take advantage of this temporal coherence in constructing our video cutout algorithm.

Building an interactive interface for extracting a dynamic foreground object from a video presents two primary challenges. First, the algorithms must be designed to efficiently process the large number of pixels in a video sequence. In particular, the algorithms must fast enough so that the interface appears responsive and the user remains engaged in the extraction task. The second challenge is to provide an intuitive interface which allows users to quickly and easily indicate foreground regions as these regions move and deform through time. Our system addresses both of these challenges and offers four main contributions over previous work:

Volumetric painting interface. We present a novel interface that allows users to indicate foreground and background regions by painting on surfaces within the spatio-temporal video volume.

Hierarchical segmentation algorithm. We introduce a hierarchical mean-shift segmentation preprocess that allows us to solve a global min-cut optimization in interactive time – 10 to 15 seconds for 100 to 200 frame sequences at 640x480 or 720x480 resolution.

Local color and edge costs. We define new spatially local color and edge models within a min-cut framework to leverage the advantages video offers. We show how to combine the local models with the global models from previous work.

Spatio-temporal alpha matting. We extend the 2D alpha matting presented in [Rother et al. 2004] to work with 3D spatio-temporal video objects while preserving both spatial and temporal smoothness of the alpha matte.

We demonstrate that our interactive video cutout system allows users to quickly extract dynamic foreground objects from relatively long video sequences. Once we have extracted the objects we can composite them onto new background (see Figure 1) or render them in a cartoon style (see Figure 11).

2 Related Work

The problem of extracting foreground objects from images and video has been studied extensively over the last 20 years. We focus on several strands of previous work that are directly related to our interactive video cutout system.

2.1 Segmentation-Based Cutout

The simplest image cutout techniques segment the image by selecting all pixels that match a user-specified image feature such as color. Photoshop's magic wand [INCORP. 2002] takes this approach. It lets users select pixels that match a range of colors.

Automatic image segmentation is a well-studied problem in computer vision. Techniques such as mean-shift segmentation [Comaniciu et al. 2001] and min-cut optimization [Boykov et al. 2001] use a variety of image features such as gradients, edges and texture to divide the image into similar looking regions. Without user input, however, it is difficult for these techniques to differentiate between background and foreground regions. The challenge is to develop algorithms that can accurately identify the foreground regions with minimal user guidance.

Recently, we have seen segmentation-based image cutout systems that address this challenge. Intelligent paint [Reese and Barrett 2002] first oversegments the image and then lets the user select the regions that form the foreground object. LazySnapping [Li et al. 2004] and GrabCut [Rother et al. 2004] provide interactive min-cut-based cutout solutions. In both systems users coarsely indicate foreground and background regions with a few strokes of the mouse and the system determines the ideal boundary for segmenting the image. Our work extends these techniques to the more challenging problem of video cutout.

Mean-shift segmentation has been used in the context of video segmentation. The Video Tooning system [Wang et al. 2004] uses a mean-shift method to divide the video into 3D regions and then asks the user to manually merge the regions together into semantically meaningful regions. We replace the low-level manual region merging with user guided min-cut optimization. The Video Paintbox system [Collomosse et al. 2003] applies mean-shift to each 2D frame individually and then runs a heuristic matching algorithm based on color, area, overlap and shape to associate 2D regions across adjacent frames. While we borrow the idea of applying mean-shift to 2D frames individually, we then apply mean-shift again to the 2D regions to produce 3D spatio-temporal regions. Our hierarchical mean-shift is also similar to the approach of DeMenthon and Megret [2002]. However, we do not rely on motion vectors as part of the feature space.

2.2 Boundary-Based Cutout

Another way to extract a foreground object is to fit a curve to its boundary. However, precisely drawing this boundary curve by hand can be extremely difficult. Boundary tracing techniques such as the classic active contour models [Kass et al. 1987], intelligent scissors [Mortensen and Barrett 1995], image snapping [Gleicher 1995], jetstream [Prez et al. 2001] and Photoshop's magnetic lasso use curve optimization methods to significantly reduce the accuracy required to trace a boundary.

The process of tracking the boundary curve of an object through a video sequence is called *rotoscoping*. User-guided rotoscoping techniques [Hall et al. 1997; Luo and Eleftheriadis 1999] allow users to trace curves every few frames and automatically interpolate between them. More recently Agarwala et al. [2004b] have combined an optimization technique [Blake and Isard 1998] with user guidance to significantly reduce user effort. However, this technique requires that the video contain strong edges between foreground and background and has difficulty with fast moving foreground objects.

2.3 Matting

Pixels on the edge of a hard segmentation boundary often contain a mixture of the foreground and background. To seamlessly composite such "mixed pixels" requires estimating an opacity (alpha) value and foreground color for each pixel. Ruzon and Tomasi [Ruzon and Tomasi 2000] show how to estimate the alpha matte and foreground color using statistical methods. Chuang et al. [Chuang et al. 2001; Chuang et al. 2002] extend this approach by using Bayesian statistics to accurately estimate alpha for partially transparent foreground objects in both images and video. The GrabCut system [Rother et al. 2004] describes *border matting* which assumes a strong prior model on alpha to quickly estimate a smooth matte. Our work provides a fast interactive means to determine the *trimap* needed for Bayesian Matting. We also extend the border matting method in [Rother et al. 2004] to the 3D video volume.

2.4 Video as 3D Object

The user interface we describe relies on the idea of treating video at a 3D volume of data. Kwatra et al. [Kwatra et al. 2003] have applied graph cut methods directly to a video volume for texture synthesis. Fels et al. [Fels and Mase 1999] and Klein et al. [Klein et al. 2002] present approaches for interactively viewing video in the form of a 3D volume. The Proscenium [Bennett and McMillan 2003] system uses the video cube paradigm for video editing. Users can manipulate the cube by slicing it with a cutting plane, and distort or warp the video to facilitate alignment. We leverage the idea of manipulating the video cube and allow users to indicate foreground objects by painting on arbitrary spatio-temporal surfaces in the video volume.

3 System Overview

A block diagram of our system is shown in Figure 2. We assume that the input video has been stabilized either by using software stabilization tools [Lucas and Kanade 1981], or by shooting the video with the aid of a tripod. The input video is processed in three majors stages; automatic preprocessing, interactive segmentation and automatic post-processing.

Automatic preprocessing. To ensure efficient updates during the interactive segmentation stage we precompute a hierarchical decomposition of the input video by iteratively applying a mean-shift clustering algorithm. We also compute neighborhood information between the clusters in the decomposition and local statistics over the entire video sequence. We have worked with clips at (720x480) or (640x480) resolution and of 100 to 200 frames. The preprocess takes from 10 to 30 minutes.

Interactive segmentation. In the interactive segmentation stage users paint pixels to indicate that they are foreground or background. Users can interactively rotate, slice and cutaway parts of the video volume to paint on pixels inside it. At any point in the interaction, users can run a min-cut optimization over the entire video to compute a segmentation based on the current set of painted hints. We have designed a new, efficient, version of the min-cut algorithm that runs on the hierarchical representation created in our preprocessing stage. Ten to fifteen seconds of computation provides a segmentation result to which the user can add more paint strokes as necessary to refine the segmentation.

Automatic post-processing. In the final post-processing stage we first perform a pixel-based min-cut optimization constrained to a narrow region around the coarse segmentation to refine the foreground boundary. We then extract a spatio-temporally coherent alpha matte so that the foreground object can be seamlessly composited onto other backgrounds.

4 Preprocessing

Even short video sequences contain a large number of pixels (e.g., 100M pixels in a 10 second shot). The scale of the problem makes

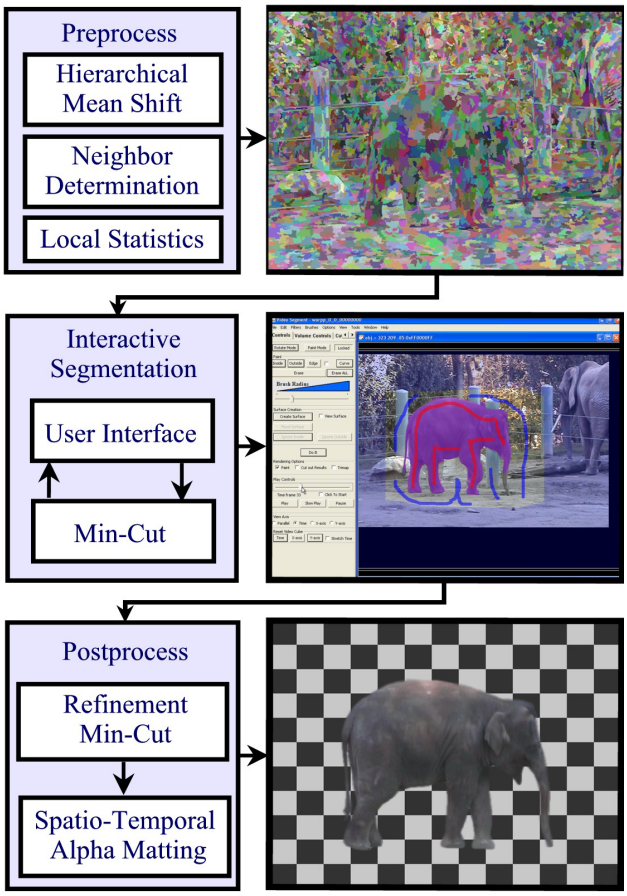

Figure 2: The input video is processed in three stages to extract the foreground object. The preprocess over-segments the video and precomputes local statistics to improve the efficiency of the interactive stage. In the interactive segmentation stage the user works with the system to extract the foreground object. The postprocessing stage automatically refines the boundary of the foreground object.

it computationally infeasible to run a segmentation algorithm at the pixel level. Our preprocessing stage pre-segments the video into a hierarchical decomposition and computes local statistics. This preprocess allows our interactive segmentation stage to operate efficiently and respond quickly to users as they paint new strokes in the video volume.

4.1 Hierarchical Mean-Shift Segmentation

We use the mean-shift clustering algorithm to build the hierarchical decomposition of the input video. The mean-shift algorithm clusters pixels that lie near one another in some *feature space*; typically the cross product of spatio-temporal position and color. Our hierarchical algorithm first applies mean-shift segmentation to create small 2D regions (on average 100 pixels in size) on each frame of the input video. The complete collection of 2D regions, parameterized by their mean positions and colors, are then clustered into 3D spatio-temporal regions (see Figure 4). On average, about 20 2D regions are merged into each 3D regions, thus a typical 3D region contains about 2000 pixels. Figure 2 shows a pseudo-colored slice through the 3D regions on a video depicting an elephant.

This procedure generates a strict hierarchy of regions so that each pixel belongs to a single 2D region and each 2D region belongs to a single 3D region. The exact coverage ensures that each pixel is given exactly one label in the interactive min-cut optimization.

By constructing the hierarchy in this two-step manner we avoid the prohibitive expense of running mean-shift directly on all pixels of the video. In comparison to the "overnight" running time of the full 3D mean-shift procedure in [Wang et al. 2004], the first step of hierarchical approach requires about 2 seconds per frame at 720x480 resolution to perform the segmentation. The second step then operates on approximately 1% of the number of nodes as pixels and thus completes in only 5 seconds per frame or about 10 to 15 minutes on a 150 frame sequence.

4.2 Neighbor Determination

In the preprocessing stage we also compute neighborhood relationships between all pairs of adjacent regions in the mean-shift hierarchy. At the pixel level, we consider each pixel to have 26 *links* to its neighbors, the 8 pixels surrounding it in its own frame, and the 9 adjacent neighbors in the frames immediately preceding and following it. The neighboring pixel links are implicitly encoded by the pixel ordering.

At higher levels in the hierarchy, we establish the neighborhood relationships between all 2D and 3D mean-shift regions. Two regions are neighbors if any pixel in one region is adjacent to a pixel in the other region. For each region we record pointers to all neighboring regions. We also record the number of pixels in each region, and the number of pixel-to-pixel links between each pair of neighboring regions.

4.3 Local Statistics

As we will discuss in section 5.2.2, we have developed new local cost models for the min-cut process. To efficiently compute these local costs, we precompute statistics on *pixel-spans* and *link-spans* in the video. A pixel-span is defined as the set of T pixels at some spatial location (x,y) through time (where T is the number of frames). Similarly, a link-span is the set of links that connect two directly linked pixel-spans. Note that the two pixel-spans may be the same, thus such a link-span contains only the direct temporal links between pixels in a single pixel-span. We compute the mean and variance of the gradients across the links in a link-span in each of the RGB color channels. We also compute and store local link costs as will be described in section 5.2.2.

5 Interactive Segmentation

The interactive segmentation loops between two stages. In the first stage, the user paints strokes to coarsely indicate foreground and background. Then, a min-cut optimization is invoked to compute a segmentation based on the strokes. The min-cut takes about 10 to 15 seconds on a 720x480x100 pixel video. The user then examines the solution, adds more strokes if needed and repeats the process until satisfied.

5.1 User Interface

With 2D images all the data is visible on the image plane. Users can directly access each pixel to indicate foreground and background. With 3D spatio-temporal video volumes, pixels in front occlude those in back and therefore we must provide tools for interactively accessing pixels that lie inside the volume.

As shown in Figure 3, our interface allows users to view and access pixels in a variety of ways. We treat the video volume as a 3D cube of data where the X- and Y-axes represent the normal horizontal/vertical axes of a single frame, while the Z-axis represents time. Users can rotate the cube to view it from any angle, cut through the cube with cutting planes at any orientation and slice through parts

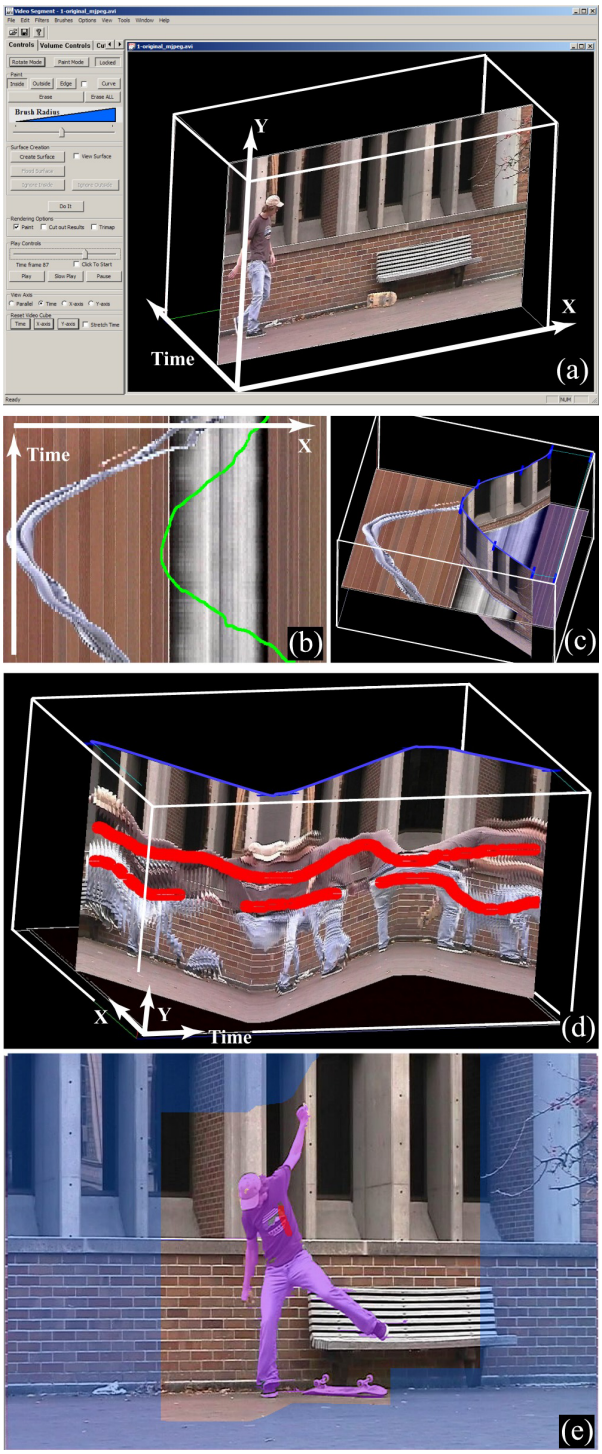

Figure 3: (a) The user rotates a video volume of skateboarder. (b) An XZ-view of the video volume. The user draws a green curve to extrude a surface through the volume. (c) The extruded surface allows the user to fill the right side of the volume with background paint (blue tint on XZ-plane). (d) Another curve drawn along the skateboarder's path creates an extruded surface that allows the user to quickly mark the skateboarder with red foreground paint across multiple frames. (e) An intermediate segmentation result (purple tint indicates foreground object). The user touches up this result with a little more red foreground paint on the skateboarder's shirt.

of the cube by drawing arbitrary surfaces through it. These manipulations make it easy to mark foreground (red paint) and background (blue paint) pixels within the volume using simple paint strokes as well as larger-scale volume fill operations.

Figure 3(a) is a screen-shot of our user interface showing a video sequence of a skateboarder in the 3D video cube representation. The user has rotated the video cube representation with an arcball interface and the viewing plane is oriented parallel to the standard XY-plane. The user can reorient the cutting plane with an arcball interface, or switch to any orthogonal axis-aligned view, like the top-down view shown in Figure 3(b). In this case the user is looking at an XZ-plane passing through the volume. Interactive controls allow sweeping any such cutting plane through the volume.

Spatio-temporal cutting planes are very useful for seeing the motion of objects over the entire video sequence in a single image. In Figure 3(b), the blue-gray C-shaped curve shows the motion of the skateboarder over time. The user can see that the skateboarder never appears on the right side of this view, and can confirm this hypothesis by scrolling the XZ-plane through the volume. To mark the entire right side of the video volume as background the user simply draws a curve (shown in green) across this XZ-plane. As shown in Figure 3(c), the system extrudes this curve through the volume. The user then fills the large portion of the volume to the right of the surface with background paint (the blue tint on XZ-plane indicates that it is marked as background). This operation is equivalent to setting a tight bound on the right side of the skateboarder in every frame. These large background regions are excluded from the min-cut optimization to significantly speed up the optimization.

In Figure 3(d) we see another extruded surface. This time the curve was drawn through skateboarder in the top-down view. As a result he is spread out over much of the surface. By applying red, foreground paint over the skateboarder on this curved surface, the user quickly indicates foreground pixels across many parts of the spatio-temporal volume. Cutting the volume with such surfaces is especially useful for marking thin moving structures that may be difficult to paint in a standard video frame. Moreover, because min-cut optimization is designed to regularize the shape of regions, such thin structure usually require more red foreground paint to be extracted properly.

The user can invoke the min-cut optimization at any time by clicking a button in the interface. After several seconds of computation the resulting segmentation is shown with the foreground element tinted purple. Figure 3(e) shows an intermediate segmentation result for the skateboarder (note that here again the large background regions are tinted blue, while the segmented skateboarder is tinted purple). The user can scroll through the entire video to visually check how well the foreground object is extracted, and add more paint as necessary. In this example, the user spots a hole on the skateboarder's shirt and adds red foreground paint so that the min-cut optimization will properly include the entire shirt in the next iteration.

5.2 Hierarchical Min-cut

We invoke a min-cut optimization algorithm [Boykov et al. 2001] on the mean-shift hierarchy to assign each pixel in the video to be either foreground (F) or background (B). To use min-cut for video segmentation, we need to design two aspects of the problem; the graph topology and the cost function to minimize.

Graph topology. The graph topology is defined by a set of nodes and bidirectional links between the nodes. Based on the mean-shift results, a node can be a pixel, a 2D spatial region within a frame, or a 3D spatio-temporal region. The full collection of nodes in the graph should exactly cover the original video with no overlapping nodes. In other words, if a 3D (or 2D) region is selected for inclusion in the graph, none of its children should be selected. Similarly

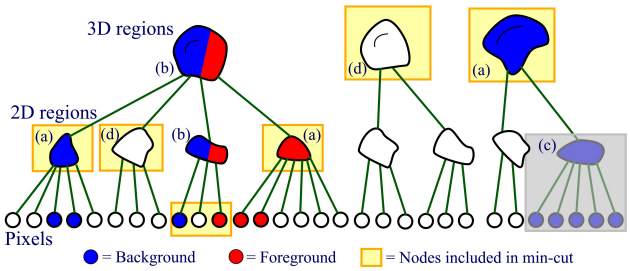

Figure 4: The hierarchical mean-shift structure of a video. Pixels are first clustered into 2D regions which are then clustered into 3D regions. As users mark foreground (red) or background (blue) pixels the labels are propagated up the trees. (a) Nodes that contain only red or blue paint are fully constrained. (b) Mixed nodes are left out of the graph due to conflicting paint strokes in their children. (c) If an entire subtree is marked as background (highlighted in gray) it is removed from further consideration. (d) The min-cut algorithm must assign a label to each white node in the graph. Although not in this diagram, these white nodes are typically the most numerous. To apply min-cut we construct a graph containing the highest level nodes of a single color (highlighted in yellow).

if a pixel (or 2D) node is selected, none of its ancestors should be included. Links are created between all pairs of neighboring nodes.

Cost functions. The min-cut optimization minimizes a cost function over the graph. Our cost function consists of two types of costs, *data costs* for assigning each node to either label, F or B, and *link costs* for each link. During min-cut the link cost is incurred only if the nodes connected by the link are assigned different labels.

Given the graph topology and the data and links costs, the min-cut algorithm chooses a set of labels (F or B) for each node that minimizes the total labeling cost on the whole graph. Once nodes are labeled, the labels are propagated down to pixels for the final segmentation result.

5.2.1 Graph Topology

Figure 4 illustrates the hierarchy resulting from the mean-shift presegmentation. All pixels are initially unlabeled; indicated as white in the diagram. As the user paints pixels as either foreground (red) or background (blue) these labels are propagated up the trees. Regions above the pixel level can take on both labels if their children are marked with conflicting labels.

At each invocation of the min-cut procedure a new graph is built dynamically. Each tree of regions from the mean-shift procedure is processed top down. When a region is found that has no conflicting labels it is added as a node in the graph so that only the highest level regions with a single color are added to the graph. These regions (highlighted in yellow in Figure 4) are guaranteed to be a minimal set of consistently marked nodes that exactly covers the video volume. In addition, to improve efficiency, subtrees that are fully marked as background are simply removed from further consideration (highlighted in gray in Figure 4).

All pairs of nodes in the graph are then tested to see if they are neighbors. If so, a link is added to the graph connecting the pair. In practice almost all nodes are the top level 3D nodes. Users only need to break the top level 3D nodes by painting stroked when adjacent foreground and background regions are similar in appearance. This process assures that the user can override any errors in the initial mean shift process.

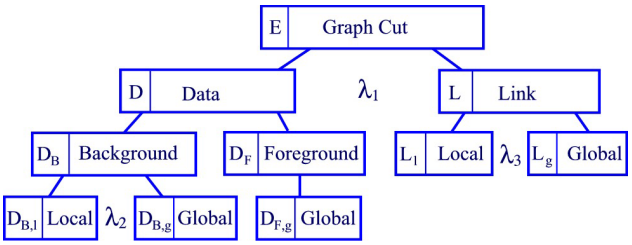

Figure 5: Min-cut cost structure. Each symbol to the left of a box represents an individual cost function. They are defined in Section 5.2.2. The λ's represent weighting parameters between the different cost functions.

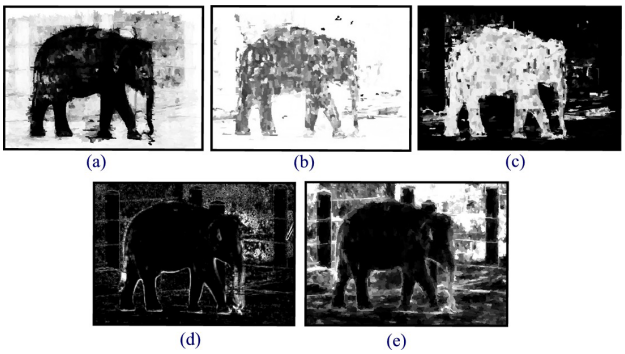

Figure 6: Visualizations of the costs on a single frame; (a) local background data cost ($D_{B,l}$), (b) global background data cost ($D_{B,g}$), (c) global foreground data cost ($D_{F,g}$),(d) local link cost (L_l), and (e) global link cost (L_g). White represents low cost indicating good places to assign label B in (a) and (b), to assign label F in (c), and to cut links in (d) and (e). No single cost function is accurate everywhere, but taken together they allow min-cut to extract the elephant

.

5.2.2 Data and Link Costs

We define a cost function E on our graph as a combination of global cost functions described in previous work [Rother et al. 2004; Li et al. 2004], and new local cost functions. Figure 5 illustrates the structure of our cost function.

The total cost is represented as a Gibbs energy

$$E(X,Z,\Gamma) = \sum_i D(x_i,z_i,\gamma_i) + \lambda_1 \sum_{nghbrs(i,j)} L(x_i,x_j,z_i,z_j) \qquad (1)$$

where D is the data term which assigns a cost to label node i with label x_i, that has color z_i and has the user assigned label (if any) γ_i. In our case x_i equals either background, B, or foreground, F, and the user labels are either foreground F (red paint) or background B (blue paint). Unpainted nodes are marked as $\emptyset$. The link term L assigns a cost to each link between neighboring nodes i and j. This cost is 0 if the labels $x_i = x_j$. Otherwise, the link cost is related to the color gradient, $\bigtriangledown$, between nodes i and j. Note that henceforth for brevity, we will leave out all the arguments of these functions unless they take on a specific value.

Higher level nodes are composed of many individual pixels. We set the data costs to be the sum of individual pixel costs for a given label. Thus, $D(i) = \sum_{p \in i} D(p)$ and the higher level nodes incur data costs proportional to their volume. We set the link costs to the sum of individual link costs at the pixel level over all links between the nodes. Thus, $L(i,j) = \sum_{l \in (i,j)} L(l)$. The link costs are independent of user input and are computed and stored at preprocess time. The λ_1 constant balances the data and link costs.

Data Costs. As described in section 5.1 the user paints within the video volume to indicate specific pixels are either foreground or background. These values are then propagated up the mean-shift hierarchy as shown in Figure 4. If node i has been painted as foreground $\gamma_i = F$, or background $\gamma_i = B$, then

$$\begin{aligned} D(x_i = B, z_i, \gamma_i = B) &= 0 \\ D(x_i = F, z_i, \gamma_i = B) &= \infty \\ D(x_i = B, z_i, \gamma_i = F) &= \infty \\ D(x_i = F, z_i, \gamma_i = F) &= 0 \end{aligned} \tag{2}$$

In other words, assigning a label that matches the user's paint strokes is free, and violating the user's paint strokes incurs infinite cost. Thus, the user's painted hints are guaranteed to be respected by the min-cut.

The vast majority of nodes remain unpainted even after propagating the paint up the mean-shift hierarchy (i.e., $\gamma_i = \emptyset$). In this case, individual data costs, D_B and D_F, are determined for assigning a node to background or foreground. These are then normalized as in previous work

$$D(x_i = B) = \frac{D_B}{D_B + D_F} \tag{3}$$

$$D(x_i = F) = \frac{D_F}{D_B + D_F} \tag{4}$$

As in [Rother et al. 2004] we use the pixels painted by the user to build two Gaussian mixture models (GMM), one for the colors found in the background and the other for colors found in the foreground. Our models use 5 Gaussians. A color of a node, z_i, (the mean color in the case of a higher level node) is tested against each GMM to return a likelihood of that color belonging to either the background or foreground. The global data costs, $D_{B,g}$ and $D_{F,g}$ are taken as the complements of the likelihoods.

$$D_{B,g}(x_i = B) = 1 - \sum_{k=1}^{5} w_k e^{(-(z_i-\mu_k)^T \Sigma_k^{-1}(z_i-\mu_k)/2)} \tag{5}$$

where the w_k are the weights corresponding to the fraction of samples closest to the k^{th} component of the GMM, and μ_k and Σ_k are the mean color and covariance of the k^{th} component of the GMM. The same equation applies to the global foreground model, $D_{F,g}$, except that the GMM is computed from the samples marked as foreground by the user.

Unlike previous work, we also include a local background color model in our data cost. If a video was captured from a tripod, or has been stabilized, the background color will often vary little from frame to frame at any specific location. We extract a static "clean plate" from the video by applying temporal filters [Agarwala et al. 2004a] or using a simple median filter across all frames. However, for most sequences, even after video stabilization, the background is not exactly stationary due to camera shake, moving background objects, and camera noise. Therefore, we consider the pixels from the static clean plate as well as pixels marked by the user as background to build an additional local background color model. For a pixel, p, our local model is given by:

$$D_{B,l}(x_p = B) = 1 - e^{(-d(z_p)/(2\eta_{cp}^2))} \tag{6}$$

where the distance function $d(z_p)$ is taken as the minimum color difference between the pixel color, z_p, and the pixel in the clean plate or any pixel in it's pixel span with user label B. The term η_{cp} models noise and is set a priori. We have found that a value of 5 (in a 0-255 color space) to work well in practice. Since our local background color model cannot be evaluated directly for higher level nodes, we compute it as the sum over all child pixels. For efficiency we pre-compute the local background term at each pixel after generating the static clean plate.

The complete background data term is a linear combination,

$$D_B(x_i = B) = \lambda_2 D_{B,l} + (1 - \lambda_2) D_{B,g} \tag{7}$$

where λ_2 reflects the confidence in the local background model. It is set locally based on the number, N, of B marked pixels in the span

$$\lambda_2 = 1 - e^{(-(N+1)\cdot 100/NumFrames)}. \tag{8}$$

Link Costs. Link costs are non-zero only when the nodes across the link are set to opposite labels, B and F. The link costs are designed to keep the segmentation coherent across space and time.

Previous min-cut based segmentation approaches make an a priori observation that the transitions between foreground and background will usually exhibit higher gradients than the average gradient between any two pixels. Therefore, they assign a static exponential function based on the color difference, or gradient $\bigtriangledown$, between the two nodes the link connects. We adopt this approach for our global link cost

$$L_g = e^{(-\bigtriangledown^2/(2\eta_{link}^2))}. \tag{9}$$

Here η_{link} represents the variance of the gradients across all links in the video. In practice we set η_{link} to 20.

In complex video sequences, both the foreground and the background may have complex patterns and strong edges which will encourage the segmentation to cut along edges inside the foreground object as well as within the background. We have developed developed a local link model to better handle such cases. We utilize the mean $\mu_{\bigtriangledown}$ and variance $\sigma_{\bigtriangledown}$ of the link gradients in each link-span. Our model is independent of the user's input and is computed once at preprocess time. We encourage a link to be cut if it has both a higher than average gradient in its span, and is an unusual occurrence in the span. Thus the local link cost is computed as

$$L_l(x_i \neq x_j) = \begin{cases} e^{((\bigtriangledown_{ij} - \mu_{\bigtriangledown ij})^2 / 2\sigma_{\bigtriangledown ij}^2)}, & \text{if } \bigtriangledown_{\text{ij}} > \mu_{\bigtriangledown \text{ij}} \\ 1, & \text{otherwise} \end{cases} \tag{10}$$

The global and local link costs are also taken as a weighted average $L(x_1 \neq x_2) = \lambda_3 \cdot L_g + (1 - \lambda_3) L_l$. In practice we set λ_3 to 0.3.

Figure 6 shows visualizations of the various costs on a frame of an elephant video. Note how the local costs, (a) and (d) help to isolate the foreground and help determine the best links across which to cut.

6 Post-processing

As shown in Figure 7(a), the interactive segmentation stage of our system usually produces a foreground object with boundary edges that are noisy in both space and time. The automatic post-processing stage is designed to refine the hard foreground boundary edges and produce a smooth spatio-temporal alpha matte.

6.1 Refinement Min-cut

To refine the foreground boundary produced by the interactive segmentation we run a pixel-level min-cut optimization within a narrow band of the initial boundary. We first erode and dilate the initial boundary by 3 pixels to remove small holes and thin, thread-like structures. We then build a pixel-level graph containing all pixels within a narrow spatio-temporal band of the boundary, as shown in Figure 7(b). The width of this boundary band is set to 10 pixels spatially and 1 frame temporally.

We construct the graph by treating each pixel within the band as a node, and each node is connected to its immediate spatial/temporal

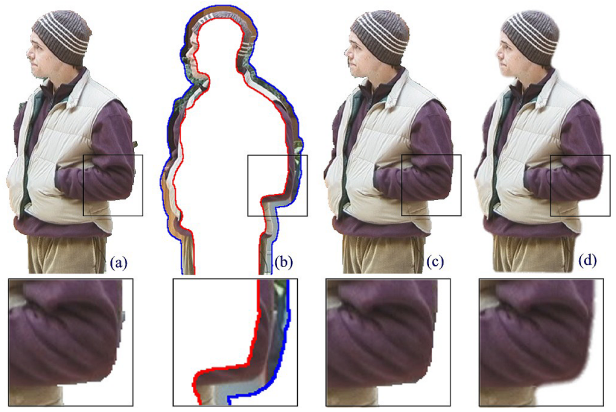

Figure 7: (a) Initial segmentation result of frame. (b) We limit our pixel-based refinement min-cut segmentation to the boundary of the initial segmentation and constrain the edges to be foreground (red) and background (blue). (c) Refined foreground boundary. (d) Final alpha matting result.

neighbors. If a pixel is on the inner boundary of the band it is set to $z = F$, and likewise outer boundary pixels are set to $z = B$ (shown as red and blue pixels in figure 7(b)). Otherwise, we set the data cost of each unknown pixel based on its foreground and background likelihoods computed locally, using the Bayesian image matting approach of Chuang et al. [2001]. We sample the 16 closest known foreground pixels from a local spatio-temporal neighborhood, and use these pixels to estimate a Gaussian foreground color model to compute the foreground data cost. We do the same thing to estimate a background color model and data cost for each pixel.

The link cost is set to a constant for all links in the graph. This encourages the min-cut optimization to cut through a small number of links to separate the unknown region into a foreground subregion and a background subregion as shown in figure 7(c). We use the refined boundaries for alpha matting.

6.2 Spatio-Temporal Matting

To extract a spatio-temporally coherent matte based on the refined boundaries, we first parameterize the boundary and build a consistent 3D contour mesh across the entire video volume. We then run a new spatio-temporal border matting algorithm to generate the smooth alpha matte.

6.2.1 3D Contour Mesh Construction

We begin by separately parameterizing the foreground boundary for each frame. The closed boundary for frame t is parameterized into a set of contour points $\{c_t^1, \ldots c_t^n\}$, where the number of vertices n is fixed for all frames. For illustration, we assume here that the foreground object is one connected component without holes.

Next, we build correspondences between contour points across adjacent frames using both shape and local color information. For c_t^i (the i^{th} contour point on frame t) and c_{t+1}^j, we compute a correspondence cost as:

$$g(c_t^i, c_{t+1}^j) = g_s(c_t^i, c_{t+1}^j) + w \cdot g_c(c_t^i, c_{t+1}^j) \qquad (11)$$

where g_s is the shape distance and g_c is a local "color distance" between the two points. The shape distance g_s is computed as the the magnitude of the difference of the *shape context* vectors [Belongie et al. 2002] of each point. If $S(c_t^i)$ and $S(c_{t+1}^j)$ are the shape context vectors for each point, then $g_s(c_t^i, c_{t+1}^j) = \|S(c_t^i) - S(c_{t+1}^j)\|$.

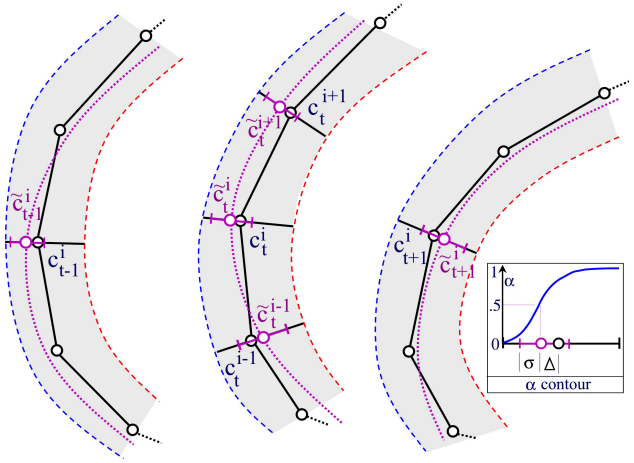

Figure 8: Three corresponding portions of trimaps and contours at three successive frames, $t-1$, t, and $t+1$. Inset: α profile.

To increase the accuracy of the correspondence we also include a local color distance term g_c in the correspondence cost. Foreground colors can be expected to be more stable than background colors in close proximity of the contour. For each contour point c_t^i, we select the 16 nearest foreground pixels and compute their mean color $M(c_t^i)$. The color distance $g_c(c_t^i, c_{t+1}^j)$ is computed as $\|M(c_t^i) - M(c_{t+1}^j)\|$. We normalize g_s and g_c to the range $[0,1]$ and have experimentally found that setting the weight w to 4 produces good results.

We use dynamic programming to find the match that minimizes the total correspondence cost; the point-wise correspondence cost summed over all contour points, on all frames. For later use, we also record the frame-to-frame correspondence cost for each frame, $G(t, t+1)$; the sum in each frame over all its contour points. Thus, we obtain a consistent 3D surface mesh over the foreground object.

6.2.2 Spatio-Temporal Border Matting

Image matting applied frame-by-frame to a video sequence produces temporally noisy results because the small errors generated on each individual frame are incoherent. The human visual system is sensitive to such temporal incoherence. Therefore we extend the *border matting* approach introduced in the GrabCut system [Rother et al. 2004] to generate spatio-temporally smooth mattes.

We begin by repeating the approach we used in generating the boundary band for the refinement min-cut to create a trimap. We mark all pixels within 10 pixels spatially and 1 pixel temporally of the refined foreground boundary as unknown (i.e., $\alpha \in [0,1]$). The pixels inside the inner boundary of this band are marked as foreground ($\alpha = 1$) and and those outside the outer boundary are marked as background ($\alpha = 0$).

Figure 8 illustrates the intuition behind our spatio-temporal border matting algorithm. In the diagram the contours from the refinement step are shown in black. Border matting defines an α profile along each normal to the contour. This contour has $\alpha = 0$ along the outside edge, and $\alpha = 1$ along the inside edge and varies smoothly in between as defined by a profile function (see inset of Figure 8). The parameters of the α profile are given by an offset Δ (the distance between the pink circles, $\tilde{c}$, and the contour points, c) and a width σ (shown as the pink error bars). The pink dotted line passing through the $\tilde{c}$'s represents where α takes on value 0.5.

The original border matting algorithm optimizes over the Δ's and σ's (equation (12) in [Rother et al. 2004]) to create an α matte that best explains each pixel's color from nearby foreground and background pixels and the local α value. It simultaneously tries

to minimize the variation in the $\triangle$ values, and σ values along the contour to maintain spatial coherence.

To achieve temporal smoothness we add a temporal term to equation (12) in [Rother et al. 2004] and optimize over all contours across all frames. Our temporal term is defined as:

$$\sum_t w(t,t+1) \sum_i \|(\triangle\kappa)\|^2 + \lambda(\triangle\sigma)^2 \qquad (12)$$

$$\begin{aligned} \triangle\kappa &= (\tilde{c}_t^{i-1} - 2\tilde{c}_t^i + \tilde{c}_t^{i+1}) - (\tilde{c}_{t+1}^{i-1} - 2\tilde{c}_{t+1}^i + \tilde{c}_{t+1}^{i+1}) \\ \triangle\sigma &= \sigma_t^i - \sigma_{t+1}^i \end{aligned}$$

Here $\triangle\kappa$ is a shape term similar to that defined in the rotoscoping system [Agarwala et al. 2004b]. It measures the change in the curvature of the contour over time. The $\triangle\sigma$ term is used to encourage σ to change smoothly. λ is a weight which we set to 100.

$w(t,t+1)$ penalizes correspondence errors based on the frame-to-frame correspondence cost, $G(t,t+1)$, we computed earlier (see section 6.2.1). $w(t,t+1)$ is defined as

$$w(t,t+1) = \begin{cases} 0 & : \quad G(t,t+1) \geq T_G \\ 1 - \frac{G(t,t+1)}{T_G} & : \quad G(t,t+1) < T_G \end{cases} \qquad (13)$$

where T_G is a predefined threshold. If the foreground object changes topology, the correspondence cost will be high and the weight will decrease to zero. In this way we only apply temporal smoothness constraints within blocks of video frames where the foreground motion and appearance is homogeneous, and do not apply smoothness constraints across frames where frame correspondences are erroneous or less reliable. The approach allows our system to produce good results even when the video sequence contains intensive motions.

So far we have computed alpha values to create a smooth spatio-temporal matte. We still need to estimate foreground colors for each foreground pixel. We estimate the foreground colors using the "pixel stealing" approach described in the GrabCut system.

7 Failure Modes and Solutions

We have exercised our system on many difficult videos. The type of automation we provide is quite powerful, but will fail in some cases. However, previously published work can be very helpful to solve such problems. Here we consider several classes of failure modes and solutions.

Video will not stabilize – Optical flow: If there is significant camera translation in the input video, parallax will prevent it from being stabilized. Optical flow algorithms can return the apparent motion of each pixel from one frame to the next. The original pixel lattice can then by modified by connecting each pixel through its flow vector to a new pixel in the next frame. The system runs as before except that the local data and link costs are not used. This was used for the "Man in Cap" sequence.

Foreground is similar to background – Rotoscoping: The min-cut based approach has difficulties if the differences between foreground colors and background colors are small. An example is shown in figure 9(a), where the girl's white shirt is ambiguous with the white wall in the color space. In this case we can call on rotoscoping methods [Agarwala et al. 2004b] constrained by our partial results. As shown in figure 9(b), we first treat the girl's skin and hair regions as foreground and segment them from the sequence using the min-cut based system. The user then indicates a rotoscoping curve around the body, including the shirt. The red vertices are attached to the boundaries of the pre-segmented regions. The red vertices are automatically propagated across time using our 3D contour mesh construction described in section 6.2.1. The green vertices are "unknown" ones corresponding to the shirt region, they

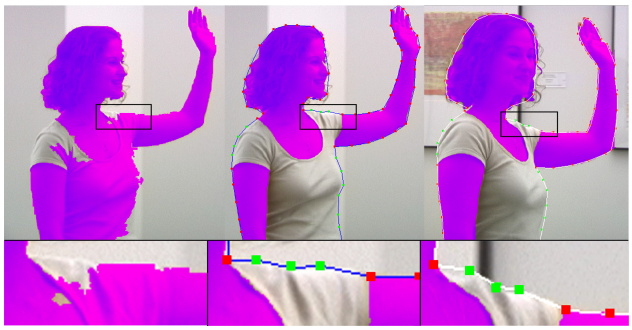

Figure 9: Amira (2): (a) Initial segmentation result on one frame. Min-cut segmentation alone cannot robustly segment the girl's shirt due to color ambiguity. (b) The user segments the skin and hair regions out, and adds a rotoscoping curve around the boundary. (c) The rotoscoping curve has been robustly propagated to the 32nd frame by optimizing the green vertices only based on Agarwala et al. [2004].

are the only vertices we need to optimize, thus the optimization can be performed more robustly, as shown in figure 9(c).

Foreground still to similar to background – Recurse: In the ballet sequence (see Figure 10) the shoe color matches the floor so well they were very hard to segment. In this case one can first mark off all but the small portion of the video containing the shoes. Operating on this subset of the video is more successful since the foreground and background color models can much more tightly focus on the shoe and floor colors. Once done, the shoes are accepted as foreground, effectively marking them as F . The rest of the video is brought into consideration and the remainder of the foreground object is segmented.

All else fails – Trust user: The beauty of an interactive system such as the one described, is that in the worst case, the user is still in control. In some cases when the foreground includes very thin structures in which almost every pixel is some combination of the foreground and background colors no system we know of will work robustly. The tail of the elephant presented such a challenge and required a few thin strokes on about every fifth frame to be included in the foreground object.

8 Results

As described in Table 8, we have run our interactive video cutout system on a number of videos, that were between 80 and 175 frames in length. Preprocessing to compute the mean-shift hierarchy and neighborhood relationships took 30 minutes or less. Post-processing time including refinement min-cut and matting took about 15 seconds per frame, or less than an hour per example.

User time depends on the complexity of the scene; more complexity, thin structures, and fast motion may require more paint strokes to guide the system. The user also has to wait for the min-cut computation between iterations. Due to the efficiencies from the hierarchical mean-shift, our min-cut computation took only 7 to 17 seconds, and thus fit nicely within our interactive framework. As sessions progressed, the min-cut time increases slightly due to the increased number of samples used to compute the color models.

The video, Amira (1) was reported to have taken about 40 minutes of rotoscoping in [Agarwala et al. 2004b]. Because of the strong edges and almost constant colors in this sequence, we were able to fully extract the foreground by painting only a few strokes on a single frame of the video.

More challenging examples are shown in Figure 10. In the skateboarder example there are many strong edges in the background which move slightly in the video despite its being captured with

Figure 10: Left: Original frame from four example video sequences. In order from top to bottom the sequences are named Ballet, Skateboarder, Man in Cap and Stairs. Right: Three frames from each sequence showing just the extracted foreground objects.

Sequence	Size	Preprocessing	Min-cut Runtime	Artist Time	Post-processing
Skateboarder	720*480*175	30 min	12.50 sec	20 min	40 min
Elephant	720*480*100	20 min	9.10 sec	40 min	30 min
Man in Cap	640*480*150	30 min	16.5 sec	20 min	35 min
Ballet	640*480*150	25 min	11.5 sec	50 min (twice)	30 min
Amira (1)	640*480*100	15 min	7.05 sec	2 min	50 min (Trimap, Bayesian matting)
Amira (2 - roto)	640*480*80	12 min	5.00 sec	5 min + roto(10 min)	35 min (Bayesian matting)
Stairs	640*480*100	20 min	8.50 sec	20 min	30 min

Table 1: Video sequences and timings for each stage of the algorithm.

a tripod. The foreground character moves very rapidly across the screen and back. Nevertheless, a successful matte is extracted with about 20 minutes of user time. The elephant seen in Figure 1 presented special difficulty due to the similarity of the elephant's color with it's background, and the presence of the relatively thin trunk and tail structures. The high frequencies in the background coupled with the uneven stabilization of the video also make this a challenging example. Approximately 40 minutes of user time was needed to extract the elephant. The ballet sequence presented additional problems due to fast motion and the fact that the feet almost matched the floor color. We completed this sequence in two passes, extracting the feet first and then the rest of the dancer while constraining the feet to be foreground. Each pass took about 50 minutes.

Once an alpha matted foreground object is extracted from a video, there are a number of possible applications. The most straightforward application is to composite the extracted foreground onto a new background video. Figure 1 shows an example of compositing a skateboarder and an elephant onto a background video taken outside the library at the University of Washington. As shown in Figure 11, the extracted foreground can also be stylized into a cartoon appearance by applying the stylization approach described in the Video Tooning system [Wang et al. 2004].

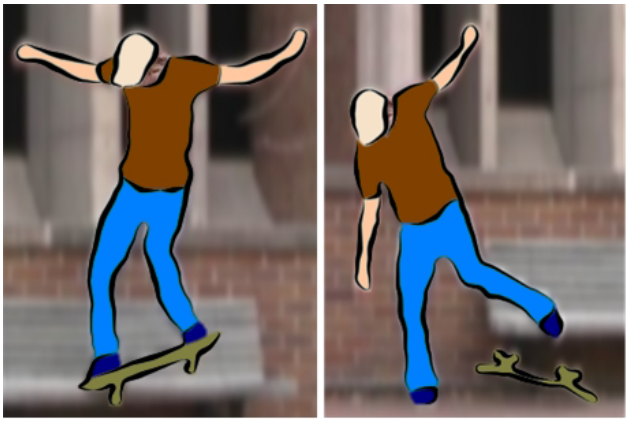

Figure 11: The skateboarder is recursively segmented into foreground sub-objects. These are then stylized using the Video Tooning system of Wang et al. [2004].

9 Conclusion

We have demonstrated an interactive system for quickly extracting alpha matted foreground objects from videos. We have also touched on a couple of applications for the extracted objects.

Although our system works well, as we mention, previous work also is helpful in special circumstances. No single solution is likely to work on all inputs. Thus an integrated set of tools will someday need to be made to fully exploit all the ideas that have been presented.

There are other problems as well we have not tackled. One may want to extract an object that passes behind thin structures. In this case, one would need to extract these blocking objects and fill in the occluded pixels with, for example, texture synthesis.

Despite the inherent difficulties of working with video, we be-

lieve our interactive video cutout system demonstrates an example of how a well designed user interface and efficient algorithms opens new possibilities for video editing.

Acknowledgements

The authors would like to thank Ke (Colin) Zheng for help with stabilizing the hand held videos, and Aseem Agarwala for sharing his videos and expertise. One of the authors of this work is supported by a grant from Microsoft Research.

References

AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. In *Proceedings of ACM SIGGRAPH*, 294–302.

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

BELONGIE, S., MALIK, J., AND PUZICHA, J. 2002. Shape matching and object recognition using shape contexts. *IEEE Trans. on Pattern Analysis and Machine Intelligence 24*, 4, 509–522.

BENNETT, E. P., AND MCMILLAN, L. 2003. Proscenium: A framework for spatio-temporal video editing. In *Proceedings of ACM Multimedia*, 177–183.

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.

CHUANG, Y.-Y., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2001. A bayesian approach to digital matting. In *Proceedings of IEEE CVPR 2001*, vol. 2, 264–271.

CHUANG, Y.-Y., AGARWALA, A., CURLESS, B., SALESIN, D. H., AND SZELISKI, R. 2002. Video matting of complex scenes. *ACM Transactions on Graphics 21*, 3, 243–248.

COLLOMOSSE, J. P., ROWNTREE, D., AND HALL, P. M. 2003. Stroke surfaces: A spatio-temporal framework for temporally coherent non-photorealistic animations. *University of Bath, Technical Report CSBU 2003-01 (June 2003)*.

COMANICIU, D., RAMESH, V., AND MEER, P. 2001. The variable bandwidth mean shift and data-driven scale selection. In *Proc. IEEE 8th Int. Conf. on Computer Vision*.

DEMENTHON, D., AND MEGRET, R. 2002. Spatio-temporal segmentation of video by hierarchical mean shift analysis. In *University of Maryland Technical Report LAMP-TR-090, CAR-TR-978, CS-TR-4388, UMIACS-TR-2002-68*.

FELS, S. S., AND MASE, K. 1999. Interactive video cubism. In *Proceedings of the Workshop on New Paradigms for Interactive Visualization and Manipulation (NPIVM)*, 78–82.

GLEICHER, M. 1995. Image snapping. In *Proceedings of SIGGRAPH 95*, 183–190.

HALL, J., GREENHILL, D., AND JONES, G. 1997. Segmenting film sequences using active surfaces. In *International Conference on Image Processing (ICIP)*, 751–754.

INCORP., A. S. 2002. Adobe photoshop user guide.

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

KLEIN, A. W., SLOAN, P.-P. J., FINKELSTEIN, A., AND COHEN, M. F. 2002. Stylized video cubes. In *Proceedings of SCA 2002*.

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

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

LUCAS, B. D., AND KANADE, T. 1981. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, 674–679.

LUO, H., AND ELEFTHERIADIS, A. 1999. Spatial temporal active contour interpolation for semi-automatic video object generation. In *International Conference on Image Processing (ICIP)*, 944–948.

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

PREZ, P., BLAKE, A., AND GANGNET, M. 2001. Jetstream: Probabilistic contour extraction with particles. In *Proc. Int. Conf. on Computer Vision*, vol. II, 524–531.

REESE, L. J., AND BARRETT, W. A. 2002. Image editing with intelligent paint. *Proceedings of Eurographics 21*, 3, 714–724.

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

RUZON, M., AND TOMASI, C. 2000. Alpha estimation in natural images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. I, 18–25.

WANG, J., XU, Y.-Q., SHUM, H.-Y., AND COHEN, M. F. 2004. Video tooning. In *Proceedings of ACM SIGGRAPH*, 574–583.