Real-Time Object Tracking for Augmented Reality Combining Graph Cuts and Optical Flow

Jonathan Mooser* Suya You† Ulrich Neumann‡

CGIT Lab
University of Southern California

Figure 1: Tracking and annotating an object using graph cut segmentation. A building sign on the USC campus is first detected using simple recognition, after which no additional information is needed. As the camera moves, we segment and track the sign through significant scale and orientation changes, rendering an annotation above it. This example illustrates a possible navigation tool for exploring buildings on campus. In this case, the annotation tells the user that the building is Powell Hall, and that the offices inside include the CGIT Lab, the IMSC offices, and the EE department.

Abstract

We present an efficient and accurate object tracking algorithm based on the concept of graph cut segmentation. The ability to track visible objects in real-time provides an invaluable tool for the implementation of markerless Augmented Reality. Once an object has been detected, it's location in future frames can be used to position virtual content, and thus annotate the environment.

Unlike many object tracking algorithms, our approach does not rely on a preexisting 3D model or any other information about the object or its environment. It takes, as input, a set of pixels representing an object in an initial frame and uses a combination of optical flow and graph cut segmentation to determine the corresponding pixels in each future frame. Experiments show that our algorithm robustly tracks objects of disparate shapes and sizes over hundreds of frames, and can even handle difficult cases where an object contains many of the same colors as its background. We further show how this technology can be applied to practical AR applications.

Index Terms: I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Tracking; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Augmented Reality;

1 Introduction

As interest in Augmented Reality has grown over the past several years, so has the demand for visual tracking methods that do not rely on artificial markers or fiducials. In a large scale environment it is often impractical to prepare the entire physical space with markers. When possible it is far preferable to track elements of the environment itself.

Tracking natural features is often challenging, typically requiring a complex model of the environment. 3D models may not be not readily available, especially for intricately shaped objects. 2D features and color distributions are easier to produce, but may not work in low contrast cases. One of our goals is to handle these difficult cases while relying on little or no preexisting information.

In the case of the present work, tracking is performed at the level of individual objects. Given an initial set of image pixels roughly corresponding to the visible area of an object (foreground pixels), we track it over subsequent frames, even as the viewpoint changes and different parts of the object come into and out of view.

We thus treat tracking as a binary foreground/background segmentation problem. Because the visible area of an object is very similar between consecutive frames, an accurate or nearly accurate segmentation in one frame can be used to initialize the next. This, as we demonstrate, offers substantial advantages in terms of both accuracy and speed.

At each frame, I_t, at time t, we find an optical flow vector for keypoints falling inside the current foreground segment. These vectors are then used to estimate a set of foreground pixels in the next frame, I_{t+1}. Typically this initial estimate is very good and only needs to be adjusted by a few pixels. The goal of the adjustment is to make the segmentation boundary match edges detected in image I_{t+1} while adding or removing as few pixels as possible. This adjustment step is the job of the graph cut algorithm, as laid out in section 4.

At no time during the tracking process do we rely on a 3D pose estimate of the object. This is a key advantage, as it allows tracking to proceed even in cases where the visible features are insufficient to compute a 3D pose.

In our experiments, we initialize the segmentation of the first

*e-mail: mooser@graphics.usc.edu
†suyay@graphics.usc.edu
‡uneumann@graphics.usc.edu

frame, I_0, either manually or using a simple Histogram of Gradients (HOG) descriptor, similar to the one described in [11]. We search for a few learned objects, and when one of them is recognized, use a stored silhouette as an initial estimate of the foreground. This detection process is intentionally kept simple. Our focus is the tracking algorithm applied to subsequent frames; after that first frame, the HOG descriptor is no longer used.

In the context of AR, a tracked object can be used to position and render virtual content, such as repair instructions or navigational guides. As an object moves within the camera frame, associated annotations will move with it, thus creating an immersive experience for the user. We demonstrate some simple examples in which virtual annotations are displayed alongside a target object.

2 Related Work

Natural feature tracking has been a key focus of AR research for some time. Alleviating the need to prepare an environment with artificial markers offers far more flexibility than traditional fiducial tracking. Many natural feature trackers, like the one described here, use optical flow as a basis for motion estimation. In [21], for example, optical flow is used to track individual point features. The present work, by contrast, focuses on tracking entire objects. Tracked 2D features may also be used as a basis for camera pose estimation[14]. Other trackers target on specific classes of features [1] such as buildings silhouettes [10] or planar structures [26].

Most object tracking algorithms rely on an underlying model of an object to estimate its positions at each frame. In many cases this involves a 3D representation that can be matched to detected edges or surfaces in an image[12, 23]. A 3D model may not be readily available, however, and is thus not feasible in all cases. Other methods, such as those based on mean shift, track statistical distributions of 2D features [8, 9, 22]. While effective in many situations, these methods may not work when the colors and features inside an object are similar to those of the background. It is also worth noting that trackers based on mean shift generally fit a rectangular or circular region around the tracked object. Our goal is to estimate an arbitrary set of pixels, even for oddly shaped targets.

An exception to these model-based trackers is [2]. Much like our work, they use a model for initial detection which is then set aside during subsequent feature tracking. The focus there, however, is on camera pose estimation. After many frames the features being tracked may not even belong to the original object. Our method focuses on tracking the object itself and determining which pixels make up its foreground for as long as the object is visible.

Image segmentation based on graph cuts has existed in various forms for many years [24, 27]. The most common approach, and one that has gained particular attention in recent years, reduces segmentation to a max-flow/min-cut problem [4, 5, 6, 13, 17, 19]. It can be proven that for binary segmentation, this reduction finds a globally optimal segmentation in polynomial time [16]. The present work follows this general strategy, as laid out in sections 3 and 4.

While segmentation is generally applied to still images, we are not the first to use graph cut segmentation on video. In fact, [4] includes examples of video segmentation, but only by passing a complete set of frames as input to their algorithm and treating the entire sequence as a 3D grid of pixels. For the purposes of AR, we need to incorporate new frames as they are acquired in real-time. Graph cuts are also applied to video in [15] and [17], where the focus is on improving computational efficiency. Our focus is on combining optical flow and edge detection to reliably track the segmentation from one frame to the next.

3 Overview of Graph Cut Segmentation

A binary foreground/background segmentation assigns a label $x_i \in \{\text{“bkg”}, \text{“obj”}\}$ to each pixel p_i in image I. Our goal is to find a complete set of labels $\mathbf{X} = (x_1, x_2, ..., x_N)$ minimizing the cost

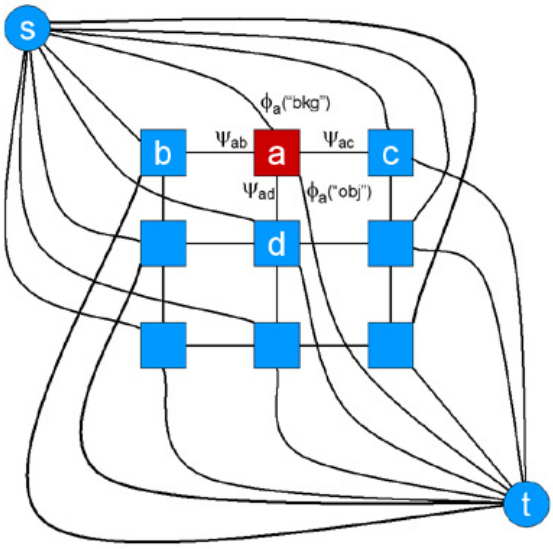

Figure 2: A graph used to compute an optimal segmentation on a hypothetical 3x3 image. The edges connected to node “a” are labeled with their respective weights.

function

$$C(\mathbf{X}) = \sum_{p_i \in I} \phi_i(x_i) + \sum_{(p_i,p_j)\in\mathcal{N}} \psi_{ij} \cdot \delta(x_i, x_j) \qquad (1)$$

where

$$\delta(x_i, x_j) = \begin{cases} 0 & \text{if } x_i = x_j \\ 1 & \text{if } x_i \neq x_j \end{cases}$$

and $\mathcal{N}$ is the set of all pairs of neighboring pixels. In our implementation, we consider two pixels neighbors if they are directly adjacent vertically or horizontally, but in theory any neighborhood system can be used. The term $\phi_i(x_i)$ can be interpreted as the cost of assigning label x_i to pixel p_i without respect to the rest of the image. The term ψ_{ij} is the cost of assigning non-matching labels to neighboring pixels p_i and p_j, and thus serves to enforce smoothness.

Clearly the optimal segmentation depends on the values of ϕ_i and ψ_{ij} at each pixel. Some implementations, for example, set the value of ϕ_i based on how closely p_i matches a preestablished color distribution for foreground and background [4]. The values of ψ_{ij} may be set to a constant, enforcing uniform smoothness throughout the image, or may vary based on the color disparity between p_i and p_j [5, 15, 17]. One of the main contributions of this paper is the novel method by which we set these values using the segmentation from the previous frame and the observed edges in the current frame, as elaborated in section 4.

Although the complete search space of all possible values of $\mathbf{X}$ is intractably large, it turns out that a global optimum can be found in low order polynomial time by constructing a weighted graph and reducing the problem to a minimum-cut problem.

The reduction proceeds as follows. Let weighted graph G have a node v_i for every pixel p_i and an edge e_{ij} connecting every pair of neighboring pixels $(p_i, p_j) \in \mathcal{N}$. In addition, add an object terminal, s, and a background terminal, t, connected to all v_i by edges e_{is} and e_{it} respectively. The edge weights are determined by values used in equation (1). We assign a weight of ψ_{ij} to all e_{ij} edges, a weight of $\phi_i(\text{“bkg”})$ to all e_{is} edges and a weight of $\phi_i(\text{“obj”})$ to all e_{it} edges. An example is illustrated in Fig. 2.

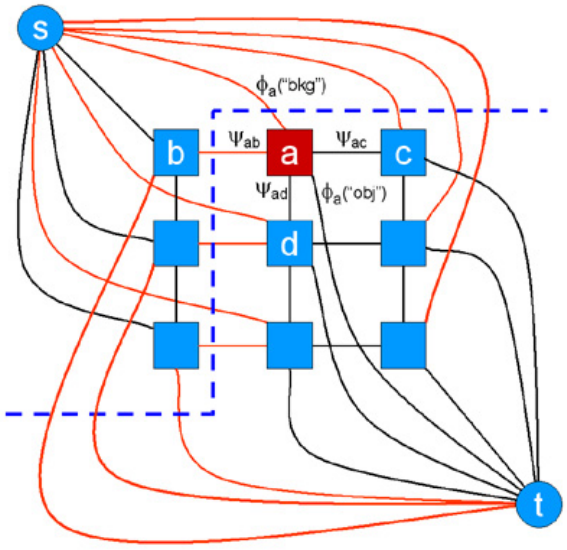

Figure 3: An example of an *s-t* cut with the edges intersected by the cut shown in red. The cost of the cut would be the sum of the weights of those edges.

In order to use this graph, we define an "*s-t* cut" as a division of all nodes in G into two disjoint sets, A and B, such that $s \in A$ and $t \in B$. We further define the cost of an *s-t* cut, $c(A, B)$, as the sum of the weights of edges having one end in A and the other in B.

$$c(A, B) = \sum_{e_{ij} | v_i \in A, v_j \in B} w_{ij} \tag{2}$$

where w_{ij} is the weight assigned to edge e_{ij} (see Fig. 3). Any *s-t* cut implies an image segmentation assigning a label of "obj" to all pixels whose nodes are in A and a label of "bkg" to all pixels whose nodes are in B. It is fairly straightforward to prove that $c(A, B)$ is exactly equal to the cost of the corresponding segmentation as defined by equation (1) [13, 16]. Thus, we can find the optimal solution by finding a minimal *s-t* cut.

There exists a rich history of algorithms that solve minimum-cut problems efficiently, beginning with the classic work of Ford and Fulkerson [16]. Many newer variations are specifically optimized for segmentation problems like the one discussed here [15, 17, 28]. We employ the algorithm described in [5], using a freely available open source implementation [18].

4 Our Approach

Given a segmentation $\mathbf{X}_t$ of frame I_t at time t, we will attempt to find an updated segmentation $\mathbf{X}_{t+1}$ of frame I_{t+1} to reflect the changes in the visible area of the object being tracked. The complete process at each frame involves three main phases: optical flow estimation, edge detection, and graph cut segmentation. These steps are illustrated in Fig. 5. In this case, a stapler is tracked on top of a cluttered desktop.

Note that a majority of the stapler's surface is mirrored, and will thus take on the colors and textures of its environment. A tracking algorithm that models the object foreground by color or feature histograms would face significant difficulties in this case. Our algorithm, by contrast, uses the visible edges at the object's boundary. As explained below, these edges do need to completely surround the object in all frames.

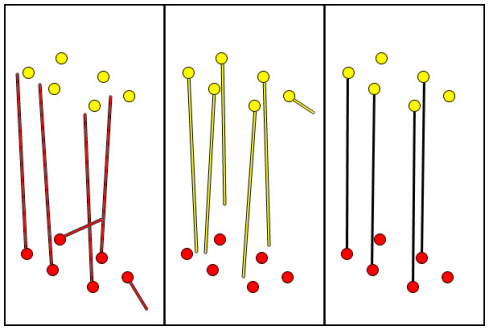

Figure 4: Illustration of matching for bidirectional optical flow. **(a)** and **(b):** Features and optical flow vectors for the current frame (red) and the previous frame (yellow). **(c):** Stable matching after applying the Gale-Shapley algorithm and removing extreme outliers having $\sigma_t(i, j) > \sigma_{\max}$.

4.1 Optical Flow

Our optical flow computations begin by detecting a set of reliable points features to track. Our features are based on a gradient covariation matrix, T_i, about each pixel, p_i

$$T_i = \sum_{p_j \in W(p_i)} \begin{bmatrix} I_x(p_j)^2 & I_x(p_j)I_y(p_j) \\ I_x(p_j)I_y(p_j) & I_y(p_j)^2 \end{bmatrix} \tag{3}$$

where $W(p_i)$ is a 9x9 window around pixel p_i and $I_x(p_j)$ and $I_y(p_j)$ are the local gradients at pixel p_j in the x and y directions. We select those points for which the smallest eigenvalue of T is a local maximum. Features based on eigenvalues of the covariation matrix have been shown to track exceptionally well [25].

Because we are only concerned with the motion of the object being tracked, we discard all features not on or near the object itself. The K remaining features at frame t form a set $F_t = \{f_{t,1}, f_{t,2}, ..., f_{t,K}\}$ of points whose motion we wish to estimate. We find a motion vector for each feature using a variation of the Lucas-Kanade optical flow tracker [20] based on image pyramids [3]. The pyramidal feature tracker takes, as input, the current image, the previous image, and the set of points selected by the feature detector. It returns an estimate of the location of each point in the previous image. By subtracting its previous location from its current location, we derive a motion vector, $\mathbf{v}_{i,t}$, for each feature.

While generally quite robust, the pyramidal feature tracker will occasionally generate significant outliers. To remove these, we could potentially use a randomized model fitting algorithm such as RANSAC. We opt instead to use a non-iterative approach which we call "bidirectional optical flow."

Bidirectional optical flow compares motion vectors of the current feature set, F_t, to the *reverse* motion vectors of the previous feature set, F_{t-1}. The most reliable features are selected by finding a stable matching between the two sets according to a matching cost:

$$\sigma_t(i, j) = [(f_{i,t} - \mathbf{v}_{i,t}) - f_{j,t-1}]^2 + [(f_{j,t-1} + \mathbf{v}_{j,t-1}) - f_{i,t}]^2 \tag{4}$$

In other words, feature $f_{i,t}$, in the current frame is considered a good match for feature, $f_{j,t-1}$, in the previous frame if $\mathbf{v}_{i,t}$ predicts that $f_{i,t}$ was near $f_{j,t-1}$ at time $t-1$ and $\mathbf{v}_{j,t-1}$ predicts that $f_{j,t-1}$ will be near $f_{i,t}$ at time t. Smaller values of $\sigma_t(i, j)$ are thus considered better matches.

Using equation 4, we look for a matching with the property that if $f_{i,t}$ is matched to $f_{j,t-1}$ and $f_{k,t}$ is matched to $f_{m,t-1}$ then $\sigma_t(i, m)$ can never be smaller than both $\sigma_t(i, j)$ and $\sigma_t(k, m)$. A matching with this property is an example of a "stable matching."

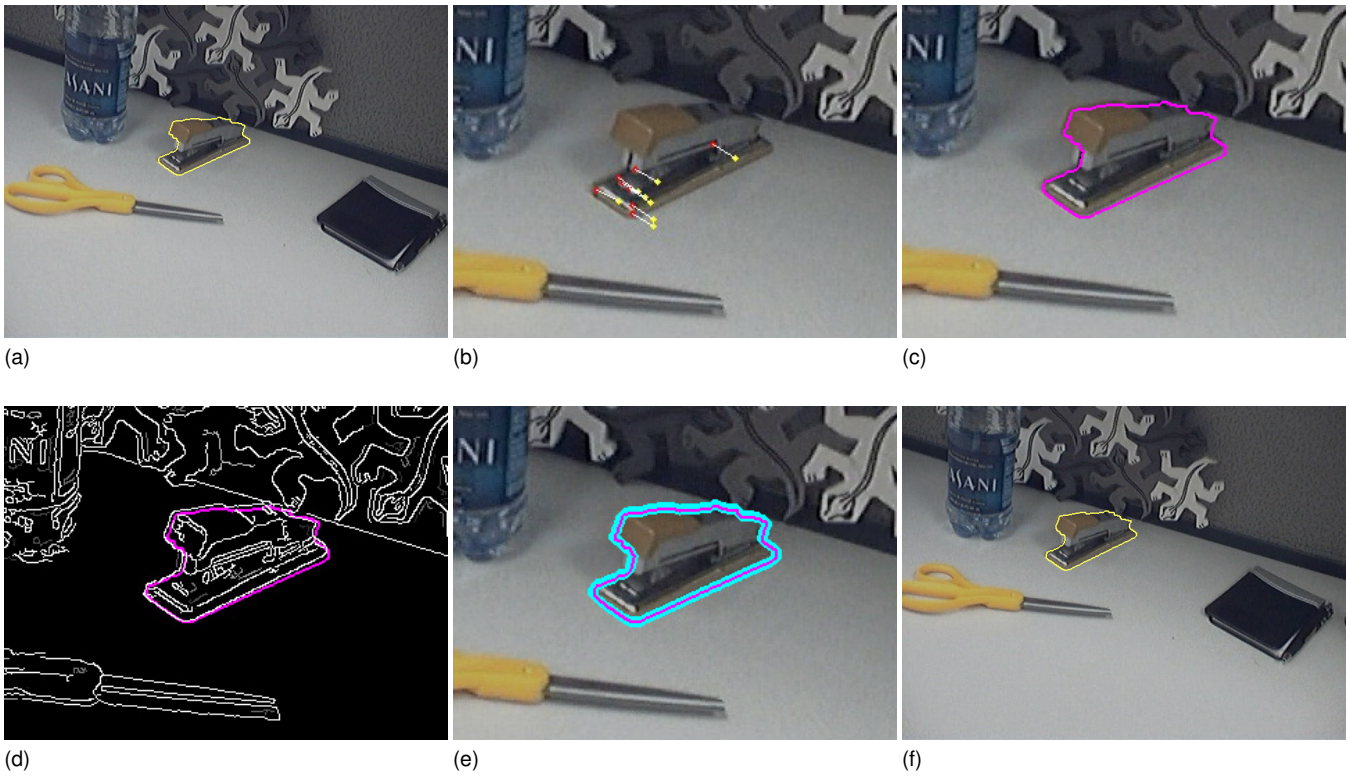

Figure 5: The steps to update the segmentation at each frame. **(a):** The segmentation, X_{t-1}, at time $t-1$. **(b):** A detail of the next frame at time $t+1$ showing the results of bidirectional optical flow. Features in the current frame are shown in red with their matches from the previous frame shown in yellow. **(c):** The segmentation estimate $\hat{\mathrm{X}}_t$ derived by translating X_{t-1} **(d):** $\hat{\mathrm{X}}_t$ superimposed over the edge chains returned by the edge salience detector, with more salient edges are rendered more brightly. **(e):** The band of pixels, B_t, for which graph nodes will be created. B_t is defined by the difference between the dilated and eroded segmentation estimates, $\hat{\mathrm{X}}_t^+$ and $\hat{\mathrm{X}}_t^-$. **(f):** The final segmentation, X_t, derived from the optimal graph cut.

A stable matching always exists and can be found using the algorithm of Gale and Shapley[16]. We use the Gale-Shapley algorithm to match features from the current frame to those in the previous frame, after which we discard those matches having $\sigma_t(i,j)$ greater than a constant $\sigma_{\max}$. Fig. 4 shows an example of two sets of features along with their estimated optical flow vectors and the resulting matches.

Using the set of features matches, we next make an estimate, $\hat{\mathrm{X}}_t = \{\hat{x}_1, \hat{x}_2, ..., \hat{x}_N\}$, of the segment labels at time t. We compute an average displacement

$$\bar{\mathbf{v}}_t = \frac{\sum_{(i,j)\in M} f_{i,t} - f_{j,t-1}}{|M|} \tag{5}$$

where M is the set of matches returned by the bidirectional optical flow procedure. $\hat{\mathrm{X}}_t$ is simply the previous segmentation X_{t-1} with the foreground pixels translated by $\bar{\mathbf{v}}_t$. While we could fit a more complex model, such as an affine transformation, we have found this simple approach effective. Generally, consecutive video frames are very similar, so $\hat{\mathrm{X}}_t$ will be sufficiently accurate even in cases of scaling and rotation. $\hat{\mathrm{X}}_t$ is only an estimation which subsequent steps will refine to produce a final segmentation X_t. Fig. 5(c) and (d) show an example of a segmentation estimate, which almost, but does not quite match the the object.

4.2 Edge Salience

When we build the graph G_t for frame t, the neighbor weights, ψ_{ij} will be set to reflect whether pixels p_i and p_j are on opposite sides of a salient edge. In this case, we do not just measure a local intensity gradient, but look for long, continuous chains of edges for which the average gradient response is significant. The rationale behind this process is that most visible objects are primarily surrounded by continuous chains of strong edges, and it is along those edges that we would like the segmentation boundary to fall. It thus makes sense to ignore small isolated edges, most likely brought about by image noise or other artifacts.

We use a Sobel edge detector to find the gradient, $I_x(p_i)$ and $I_x(p_i)$, in the x and y directions for pixel p_i. From this, we can calculate gradient magnitudes and directions for all pixels.

$$\begin{aligned} |\nabla I|(p_i) &= I_x(p_i)^2 + I_y(p_i)^2 \\ \nabla_\theta I(p_i) &= \arctan(I_y(p_i)/I_x(p_i)) \end{aligned} \tag{6}$$

To find the most salient edges we perform non-maximal suppression with hysteresis in a manner similar to a Canny edge detector [7]. We scan each image for pixels for which $|\nabla I|(p_i)$ is above a large threshold, ∇_{hi}. For each such pixel that has not already been added to a chain, we initialize a new edge chain, ϵ. The chain is followed along a direction perpendicular to the gradient direction, adding pixels to ϵ reaching some pixel, p_j, for which $\nabla_\theta I(p_j)$ falls below a low threshold ∇_{lo}. At no time does p_i initialize or continue a chain unless $|\nabla I|(p_i)$ is a local maximum along the direction of $\nabla_\theta I(p_i)$.

After all edge chains have been identified, each is assigned a salience value, $S(\epsilon)$, to reflect its length and the gradient magnitudes of its pixels.

$$S(\epsilon) = \sum_{p_i \in \epsilon} (\alpha |\nabla I|(p_i) + \beta) \tag{7}$$

The constant α favors chains whose pixels have a strong gradient magnitude while the constant β favors longer chains regardless of gradient strength. Note that one salience value is applied to all pixels in a given chain, so a pixel with a smaller gradient magnitude may show a high salience if it is a member of a chain with strong characteristics. Fig. 5(d) shows the results of edge detection and salience computations, with more salient edge chains rendered more brightly.

4.3 Graph Initialization and Minimum-Cut Computation

Having built the segmentation estimate, $\hat{\mathrm{X}}_t$, and computed the salience of all edge chains, we are now ready to construct the graph that will produce a final segmentation. We work from the assumption that $\hat{\mathrm{X}}_t$ will only differ from X_t by a few pixels at any place along its boundary. With this in mind, we dilate and erode the foreground region of $\hat{\mathrm{X}}_t$ by 4 pixels to produce a larger set $\hat{\mathrm{X}}_t^+$ and a smaller set $\hat{\mathrm{X}}_t^-$. All pixels for which $\hat{x}_i^- =$ "obj" are automatically labeled $x_i =$ "obj" and those for which $\hat{x}_i^+ =$ "bkg" labeled $x_i =$ "bkg". Only the pixels in the intervening band, B_t, will be considered by the graph cut process. The blue area of Fig. 5(e) is an example of a band built around the segmentation estimate.

This type of "banded" segmentation is similar to the graph cut active contours in [28] or the multi-level bands used in [19]. In the case of the present work, it provides two essential advantages. The first is one of performance. The graph will only contain nodes for pixels in B_t, as all of the other pixels have already been labeled. Because B_t typically covers less that 1% of the the entire image, the resulting graph has proportionally fewer nodes and edges. The algorithms that compute minimum cuts are super-linear, so the reduction in size makes this phase of the process faster by several orders of magnitude.

The banded approach is also considerably more robust over many frames than applying graph cut segmentation to the entire image. Normally, the segmentation has a tendency to suddenly grow or shrink dramatically if the edges surrounding the object become briefly diffuse. Using our method, the segmentation can never grow or shrink more than the width of the band in any one frame. As long as clear edges reappear within a few frames, the segmentation generally corrects itself.

For each pixel $p_i \in B_t$ we create a node v_i. The neighbor weights, ψ_{ij}, are set as follows.

$$\psi_{ij} = \begin{cases} \psi_{\max} & \text{if } p_i, p_j \text{ not on opposite sides of an edge chain} \\ \max(\psi_{\min}, \psi_{\max} - S(\epsilon)) & \text{if } p_i, p_j \text{ on opposite sides of edge chain } \epsilon \end{cases} \tag{8}$$

where $\psi_{\max}$ and $\psi_{\min}$ are predetermined constants representing the largest and smallest possible values for ψ_{ij}. This means that if two pixels are not on opposite sides of an edge chain they are very likely to receive the same label. If they are on opposite sides of an edge chain, that likelihood decreases in proportion to the edge chain's salience as determined by equation 7.

Two factors determine the values of $\phi_i(\text{"obj"})$ and $\phi_i(\text{"bkg"})$. We first consider whether or not p_i is inside the segmentation estimate, $\hat{\mathrm{X}}_t$, then add $\psi_{\max}$ for every neighbor of p_i that is part of either the dilated or eroded segmentations, $\hat{\mathrm{X}}_t^+$ and $\hat{\mathrm{X}}_t^-$

$$\phi_i(\text{"bkg"}) = \begin{cases} \phi_0 + \psi_{\max} \cdot (|N_o(i)|) & \text{if } \hat{x}_i = \text{"obj"} \\ 0 & \text{if } \hat{x}_i = \text{"bkg"} \end{cases}$$

$$\phi_i(\text{"obj"}) = \begin{cases} 0 & \text{if } \hat{x}_i = \text{"obj"} \\ \psi_{\max} \cdot (|N_b(i)|) & \text{if } \hat{x}_i = \text{"bkg"} \end{cases} \tag{9}$$

where $|N_o(i)|$ is the number of neighbors of p_i for which $\hat{x}_i^-$=("obj") and $|N_b(i)|$ is the number of neighbors of p_i for which $\hat{x}_i^+$=("bkg"). ϕ_o is a constant. Equation 9 means there is no cost associated with assigning a label of "bkg" to a pixel for which the estimate $\hat{x}_i =$"bkg." If, however, $\hat{x}_i =$"obj," a label of "bkg" incurs a cost of at least ϕ_0. Any of the pixel's neighbors that are inside the eroded segmentation will automatically receive a label of "obj." We thus add an additional cost of $\psi_{\max}$ for each, because two neighboring pixels will now receive opposite labels. The cost of $\phi_i(\text{"obj"})$ is determined analogously, but without the ϕ_o term.

The form of $\phi_i(\text{"obj"})$ and $\phi_i(\text{"bkg"})$ differs from traditional graph cut segmentation algorithms, such as those described in [13] or [4]. In most cases, the cost of assigning a particular label to a pixel depends on the pixel's color and how it compares to some known color distribution for foreground and background. That is not the case here. We use only the optical flow field and the detected edges to segment each image. This "colorless" segmentation allows us to handle low contrast cases like the one show in Fig. 6.

We set the constants $\psi_{\max}$, $\psi_{\min}$, and ϕ_0 to 250, 5, and 2 respectively. We want the edges to be the primary factor determining the segmentation, thus the ψ constants dominate. Only the ratios, and not the absolute values, of these constants are significant.

Having set all of the terms in equation 1, we can construct the graph and compute its minimum *s-t* cut. Once the minimum cut has been found, all pixels on the *s*-side are labeled "obj" and all of the pixels on the *t*-side are labeled "bkg," thus completing the final segmentation, X_t, at time t.

5 Results

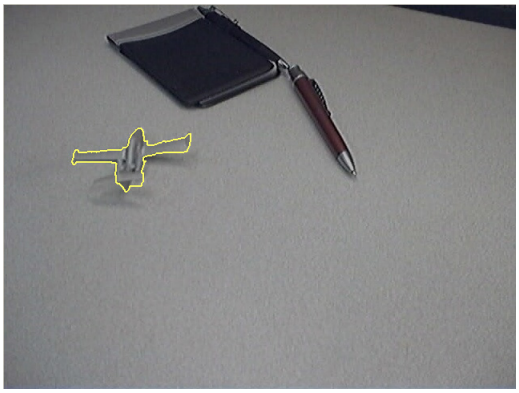

Figure 6: A difficult case handled by our algorithm. We track a gray model airplane on a gray table. Tracking remains robust even though the object and background contain nearly identical colors.

We tested our tracking algorithm on a number of different types of objects in both indoor and outdoor environments. It has proven effective even on traditionally difficult cases like the ones in Figs. 6, 7, and 8. Fig. 6 demonstrates tracking of a small model plane against an almost identically colored background. Any tracking method that relies on color or texture features to separate object

Figure 7: Another difficult case, a statue with an intricate, curved 3D structure. Although the statue is made of a uniform material, a high degree of specular reflection makes it unidentifiable by color intensities.

Figure 8: Partial occlusion. A sign is partially occluded by surrounding foliage while the un-occluded parts remain correctly segmented.

from background is likely to fail. Because we find a few salient edges that delineate the object boundary, we can estimate a fairly accurate segmentation at each frame. The object boundaries do not need to be detected at all times. So long as most of the edges in appear in most of the frames, tracking will remain robust.

Fig. 7 shows our ability to track complex 3D objects, such as the "Tommy Trojan" statue, with no 3D model. To produce a model of the statue would be extremely difficult, as it has an intricate structure made mostly of curved surfaces. Moreover, it has a high degree of specular reflectance, so its image contains a broad range of color intensities. Despite these difficulties, we are able to track the statue over a wide range of viewpoints.

Fig. 8 is an example of robustness to some amount of occlusion. The bushes cover part of the sign, briefly breaking the segmentation. Once the occlusion is removed, the segmentation boundary naturally expands to meet the edges of the sign.

One of the primary goals of this work is to produce a system capable of operating in real-time, an essential feature in the context of AR. The application needs to process frames as they are acquired to provide a smooth, immersive experience. We measured the average processing time per frame on a Pentium IV processor running Windows XP. Testing on a number of different object types, the results ranged from 87 to 112 milliseconds. In all cases, the input images had a resolution of 640x480 pixels. The average times for each phase of the process are summarized in table 1.

In certain cases, the tracking process may fail. For example,

Figure 9: Substantial occlusion can break the segmentation, especially when the occluding object "sweeps across" the object being tracked.

when the camera moves too fast, the optical flow algorithm can lose track of the point features, and thus be unable to build a reliable segmentation estimate for the next frame. Fortunately, we have found that normal motions do not generally cause a problem, even in the presence of motion blur. Maintaining a good frame rate is helpful; less time between frames means smaller motions and thus a greater chance of successful tracking. As hardware becomes faster and provides better frame rates, we can expect improved robustness to fast camera motions.

Another potential point of failure are large occlusions, especially those that completely cover an entire part of the object boundary. In such cases, the system may confuse the edges of the tracked object with the edges of the occluding object and be unable to recover. Fig. 9 shows an example where a lamppost "sweeps across" the building being tracked and loses the original edge.

All of these failure cases can be handled by occasionally reinitializing the recognition process. Although object recognition algorithms are often slow, in this case it would only need to work once every few hundred frames.

Function	**Average Processing Time (milliseconds)**
Optical Flow	30.90
Edge Detection	56.72
Graph Cut	15.43
Total	**103.04**

Table 1: Performance results broken down by functionality. "Optical Flow" includes identifying features, estimating their velocities, and matching. "Edge Detection" includes computing image gradients, building edge chains and computing their saliences. "Graph Cut" includes constructing the graph for each frame and finding its minimum *s-t* cut.

6 Applications

In our experiments we used a simplified version of the Histogram of Oriented Gradients (HOG) descriptor [11] to recognize objects and initialize the first segmentation. The HOG descriptor stores a histogram of gradient directions for each cell of a grid covering the object. In our case, we use eight orientation bins. Thus, if there are N cells, the HOG descriptor is effectively an 8xN element vector. Two such vectors can be compared by Euclidean distance.

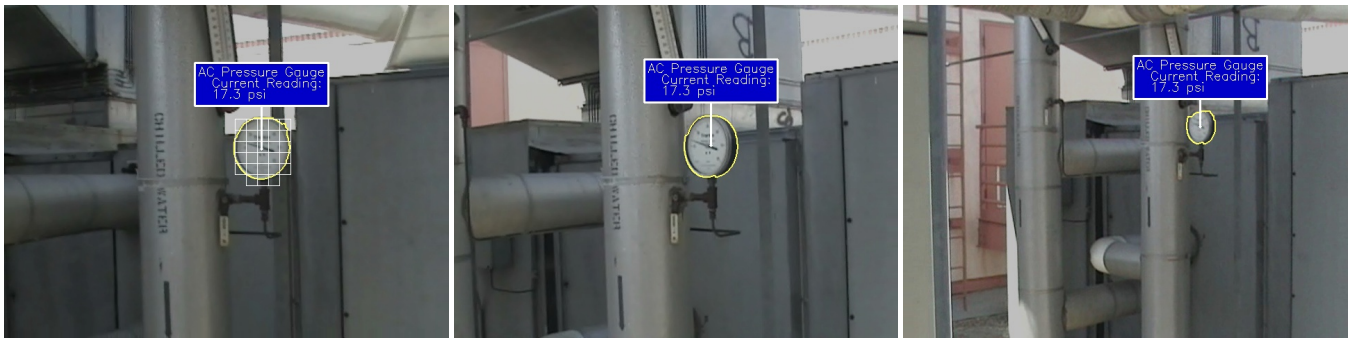

Figure 10: Tracking in an industrial setting. To assist with maintenance and repair, a pressure gauge is annotated with a real time reading. As the camera moves, the equipment is tracked and content rendered in the appropriate position.

The first images of Figs. 1 and 10 show the HOG grid drawn at the location and scale where the object was detected along with the initial segmentation and the associated annotation. In principle any object recognition could be used for initialization as long as it provides an estimate of the object boundary. We intentionally chose a simple approach here to demonstrate that after the first frame the tracker proceeds on its own.

A real world implementation would most likely use a recognition system specifically geared toward the characteristics of the objects being tracked. One of the key strengths of our approach is its ability to work in conjunction with any type of recognition. The initial estimate need only be approximate, because the segmentation boundary will generally converge onto the object boundary within a few frames.

Fig. 1 demonstrates sign tracking for an interactive navigation tool on the USC campus. As a user approaches, annotations appear that provide additional information about the building in question. Our tracker successfully segments the sign and positions the annotations through significant scale and orientation changes, thus providing an immersive experience.

Fig. 10 shows how this technology might be applied in an industrial context, automatically annotating a pressure gauge. Although the piece of equipment being tracked contains many of the same colors as its surroundings, the segmentation remains intact. A tracker based on 2D features might be able to identify the dial from frame to frame, but it is generally difficult to make 2D feature recognition robust to a wide range of viewpoints. In our case, recognition only needs to take place at one scale. Even after the initially detected features have become hidden or unrecognizable, the segmentation still works.

7 Conclusion and Future Work

In this paper we have demonstrated a robust object tracking system based on graph cuts. The strength of our algorithm lies in its versatility. Because it relies only on edges for segmentation, it can track almost any object in any environment so long as the object is at least mostly surrounded by detectable edges. The lack of reliance on any 3D or 2D feature model means that new objects do not require any special training or preprocessing to be tracked. In our experiments we use recognition only to initialize each sequence. Scale and orientation changes may obscure recognizable features in future frames, yet tracking remains reliable.

Future versions may incorporate object recognition throughout the process, identifying recognizable features whenever possible and using graph cut segmentation to keep track of the object through the intervening frames. Such a system may be more robust, for example, to occlusion and could handle cases where the object temporarily moves outside of the camera frame.

Another possible direction for future work is 3D pose estimation. Because we can track a 3-dimensional object over multiple frames, point correspondences could be used to compute a 6-DOF pose. In an AR context this would allow us to render 3D virtual media in addition to the 2D labels shown here.

The ability to track objects without artificial markers is beneficial to almost any AR application. Our system is able to do that in a wide range of contexts, and is thus of great potential value.

Acknowledgements

This study was funded by the Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies (CiSoft); CiSoft is a joint University of Southern California-Chevron initiative.

This work made use of Integrated Media Systems Center Shared Facilities supported by the National Science Foundation under Cooperative Agreement No. EEC-9529152. Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

References

[1] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *IEEE Comput. Graph. Appl.*, 21(6):34–47, 2001.

[2] G. Bleser, H. Wuest, and D. Stricker. Online camera pose estimation in partially known and dynamic scenes. In *International Symposium on Mixed and Augmented Reality*, pages 56 – 65, 2006.

[3] J. Y. Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. (http://sourceforge.net/projects/opencvlibrary), 2001.

[4] Y. Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *ICCV*, volume 01, pages 105 – 112, 2001.

[5] Y. Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, 2004.

[6] Y. Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. In *ICCV*, volume 01, pages 377–384, 1999.

[7] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679 – 698, 1986.

[8] R. T. Collins. Mean-shift blob tracking through scale space. In *CVPR*, pages 234–241, 2003.

[9] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *CVPR*, pages 2142–2150, 2000.

[10] V. Coors, T. Huch, , and U. Kretschmer. Matching buildings: Pose estimation in an urban environment. In *ISAR*, pages 89–92, 2000.

[11] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, volume 1, pages 886–893, 2005.

[12] J. Deutscher and I. Reid. Articulated body motion capture by stochastic search. *Int. J. Comput. Vision*, 61(2):185–205, 2005.

[13] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B*, 51(2):271–279, 1989.

[14] B. Jiang, S. You, , and U. Neumann. Camera tracking for aumented reality media. In *ICME*, pages 1637 – 1640, Jul 2000.

[15] O. Juan and Y. Boykov. Active graph cuts. In *CVPR*, volume 1, pages 1023–1029, 2006.

[16] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[17] P. Kohli and P. H. S. Torr. Effciently solving dynamic markov random fields using graph cuts. In *ICCV*, pages 922–929, 2005.

[18] V. Kolmogorov. (http://www.adastral.ucl.ac.uk/ vladkolm), 2007.

[19] H. Lombaert, Y. Sun, L. Grady, and C. Xu. A multilevel banded graph cuts method for fast image segmentation. In *ICCV*, volume 1, pages 259–265, 2005.

[20] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[21] U. Neumann and S. You. Natural feature tracking for augmented reality. *IEEE Transactions on Multimedia*, 1(1):53–64, 1999.

[22] V. Parameswaran, V. Ramesh, and I. Zoghlami. Tunable kernels for tracking. In *CVPR*, pages 2179–2186, 2006.

[23] J. Platonov, H. Heibel, P. Meier, and B. Grollmann. A mobile markerless AR system for maintenance and repair. In *International Symposium on Mixed and Augmented Reality*, pages 105 – 108, 2006.

[24] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888 – 905, 2000.

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

[26] G. Simon, A. Fitzgibbon, and A. Zisserman. Markerless tracking using planar structures in the scene. In *ISAR*, pages 120–128, 2000.

[27] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.

[28] N. Xu, R. Bansal, and N. Ahuja. Object segmentation using graph cuts based active contours. In *CVPR*, volume 2, pages 46–53, 2003.