# Video Browsing Using Edges and Motion

Ramin Zabih Justin Miller Kevin Mai

Computer Science Department
Cornell University
Ithaca, NY 14853
rdz@cs.cornell.edu

**Abstract**

*Automatic video browsing requires algorithms for detecting a variety of events, including production effects (e.g., scene breaks and captions) and moving objects. We present new methods that use edges and motion for detecting production effects and computing motion segmentation. Production effects, such as cuts, dissolves, wipes and captions, can be detected by looking for new edges that are far from previous edges. A global motion computation is used to register consecutive images. We have also developed a method for motion segmentation, which does not require computing local optical flow. Our methods run at several frames per second on a Sparc workstation, and tolerate compression artifacts.*

## 1 Introduction

With modern compression techniques and the plummeting costs of secondary storage, it is becoming feasible to have thousands of hours of digital image sequences available on-line. Several recent research projects have investigated tools for querying such large image databases [3, 4]. Automatic browsing of image sequences requires algorithms for detecting a variety of events, such as production effects (such as scene breaks and captions) and moving objects.

Automated browsing is a fertile area for developing and applying computer vision algorithms. Compression artifacts are a major challenge, since lossy compression is commonplace in such applications. In this paper, we describe new methods for detecting production effects and moving objects. By using edges and motion, we have designed solutions that withstand compression artifacts.

We begin by reviewing existing work on automatic video browsing, which has focused on scene break detection and (to a lesser extent) motion. In section 3 we present our solutions for detecting scene breaks, and in section 4 we supply experimental evidence that our methods perform well. In section 5 we extend these methods to handle other production effects (eg. captions and subtitles) and to perform motion segmentation.

## 2 Related work

Most existing work has focused on browsing using scene break detection, although there has also been some work on motion-based browsing. Scene break detection and motion segmentation are also important for a number of multimedia applications besides video browsing, such as compression and automatic keyframing. Motion-based compression algorithms like MPEG can obtain higher compression rates without sacrificing quality when the locations of scene breaks are known. Knowledge about scene breaks can be used to look for higher-level structures (such as a sequence of cuts between cameras), or to ensure that keyframes come from different scenes.

### 2.1 Scene break detection

Scene breaks mark the transition from one sequence of consecutive images (or scene) to another. A cut is an instantaneous transition from one scene to the next. A fade is a gradual transition between a scene and a constant image (*fade out*) or between a constant image and a scene (*fade in*). During a fade, images have their intensities multiplied by some value $\alpha$, that varies between 0.0 and 1.0. A dissolve is a gradual transition from one scene to another, in which the first scene fades out and the second scene fades in. Another common scene break is a wipe, in which a line moves across the screen, with the new scene appearing behind the line.

Scene breaks are detected by computing and thresholding a similarity measure between consecutive images. Existing work has relied directly on intensity data, using such techniques as image differencing and intensity histogramming. Most approaches are based on intensity histograms, and concentrate on cuts [6, 7]

These methods have difficulty with "busy" scenes, in which intensities change substantially from frame to frame. Such changes often result from camera or object motion.

Cuts usually result in a dramatic change in image intensities, so they can be detected much of the time. However, a dissolve is a gradual change of all the intensities, and cannot be easily distinguished from motion. A dissolve can even occur between two scenes each containing motion. Thus, dissolves are more difficult to detect than cuts, especially if the scenes involve motion. Increasing the detection threshold can reduce false positives due to motion, but at the risk of missing gradual scene transitions.

Hampapur, Jain and Weymouth [4] use an explicit model of the video production process to detect a variety of scene breaks. While their approach is intensity-based, it does not involve histogramming. Instead, they compute a chromatic image from a pair of consecutive images. Its value at each pixel is the change in intensity between the two images divided by the intensity in the later image. Ideally, the chromatic image should be uniform and non-zero during a fade.

The difficulties caused by motion and by dissolves are well-known. For example, Hampapur, Jain and Weymouth note in their discussion of dissolves that their measure "is applicable if the change due to the editing dominates the change due to motion"[4, page 11], and describe both object and camera motion as causes of false positives for their method. Another recent paper [14] describes motion as a major limitation of histogram-based methods.

Zhang, Kankanhalli and Smoliar [14] have extended conventional histogram-based approaches to handle dissolves and to deal with certain camera motions. They use a dual threshold on the change in the intensity histogram to detect dissolves. In addition, they have a method for avoiding the false positives that result from certain classes of camera motion, such as pans and zooms. They propose to detect such camera motion and suppress the output of their scene-break measure during camera motion.

Their method does not handle false positives that arise from more complex camera motions or from object motion. Nor does their method handle false negatives that occur in dissolves between scenes involving motion. In section 4 we will provide an empirical comparison of our method with histogram-based techniques and with chromatic scaling.

### 2.2 Motion-based browsing

A number of groups are currently investigating the use of motion for automatic browsing of videos. For example, researchers are exploring motion-based extensions to QBIC [3] and to Photobook [8]. However, little work on motion-based browsing of videos has been published.

There has, of course, been a great deal of work on related problems, such as computing motion or motion segmentation. Most work on motion segmentation involves some kind of local computation of motion (often called optical flow). For example, Adiv [1] partitions the flow field into connected segments whose motion is consistent with a planar surface moving rigidly. Another approach [10, 2] focuses on layered representations of multiple motions, which computes optical flow and performs a segmentation into regions.

## 3 An Edge-Based Approach

During a cut or a dissolve, new intensity edges appear far from the locations of old edges and old edges disappear far from the locations of new edges. These simple observations allow us to detect and classify scene breaks. We define an edge pixel that appears far from an existing edge pixel as an *incoming* edge pixel, and an edge pixel that disappears far from an existing edge pixel as an *outgoing* edge pixel. By counting the incoming and outgoing edge pixels, we can detect and classify cuts, fades and dissolves. By analyzing the spatial distribution of incoming and outgoing edge pixels, we can detect and classify wipes.

Our method can be easily extended in order to handle motion. We can use any registration technique to compute a global motion between frames. We can then apply this global motion to align the frames before detecting incoming or outgoing edge pixels.

The algorithm we propose takes as input two consecutive images $I$ and $I'$. We first perform an edge detection step, resulting in two binary images $E$ and $E'$. Let $\rho_{in}$ denote the fraction of edge pixels in $E'$ which are more than a fixed distance $r$ from the closest edge pixel in $E$. $\rho_{in}$ measures the proportion of incoming edge pixels. It should assume a high value during a fade in, or a cut, or at the end of a dissolve.[1]

Similarly, let $\rho_{out}$ be the fraction of edge pixels in $E$ which are farther than $r$ away from the closest edge pixel in $E'$. $\rho_{out}$ measures the proportion of outgoing edge pixels. It should assume a high value during a fade out, or a cut, or at the beginning of a dissolve.

Our basic measure of dissimilarity is

$$\rho = \max(\rho_{in}, \rho_{out}). \tag{1}$$

This represents the fraction of changed edges; this fraction of the edges have entered or exited. Scene

[1] Due to the quantization of intensities, new edges will generally not show up until the end of the dissolve.

breaks can be detected by looking for peaks in $\rho$, which we term the *edge change fraction*. In our experiments, we have used a Canny-style edge detector. We smooth the image with a Gaussian of width $\sigma$, threshold the gradient magnitude at a value of $\tau$, and perform non-maximum suppression.

### 3.1 Motion compensation

Our algorithm handles small motions through the use of dilated edges. Edges which move no farther than the dilation radius $r$ will not result in changing pixels (i.e., incoming or outgoing edge pixels). Even if every edge moves by $r$ pixels, there will still be no changing pixels. A slow camera pan, for example, will be handled in this manner. Other examples that give rise to small motions include rotations, zooms and non-rigid object motions.

A single larger motion requires a motion compensation step before computing the edge change fraction. In our experience, it has been sufficient to restrict the motion to be purely translational. While it is possible to handle affine or projective motions, they incur significant additional overhead, and do not seem to result in better performance.

We have explored two algorithms for computing motion, both of which have given satisfactory results. The algorithms are based on correlation, but also tolerate the presence of multiple motions. In our initial experiments we used the Hausdorff distance [5], which operates on edge images. More recently, we have used correlation based on non-parametric local transforms, an approach described in [13].

Note that the simple motion compensation scheme described does not handle multiple, distinctive motions. For example, if the camera pans to follow a fast-moving object, motion compensation will stabilize either the object or the background. Edges undergoing the secondary motion will appear to move; if the relative motion is larger than $r$, this will result in incoming and outgoing pixels. Handling this case requires motion segmentation, which we will discuss in section 5.2.

### 3.2 Peak detection and classification

There are three basic steps to our algorithm: motion compensate $I$ and $I'$; perform edge detection; and compute $\rho_{in}$ and $\rho_{out}$ (and thus, $\rho$). Once this is done, scene breaks can be detected by looking for peaks in $\rho$. A detailed description of the methods used to classify different types of scene breaks is beyond the scope of this paper; a full specification is given in [12]. A summary of the classification methods for cuts, dissolves, fades, and wipes is given here.

Cuts occur between a single set of frames and typically have the most intense difference measure, all the other transitions occur over a sequence of frames. Fades are characterized by a transition from or to a constant image, therefore either $\rho_{in}$ or $\rho_{out}$ will be near-zero for the duration of the transition. Wipes are characterized by the spatial distribution of changing pixels. In a wipe, the region of change moves consistently across the frame for the duration of the transition, while in the other scene breaks the distribution of changing pixels is random. Scene transitions that do not display the characteristics of a cut, fade, or wipe are classified as dissolves.

### 3.3 The Hausdorff distance

Our similarity measure is related to the Hausdorff distance, which has been used to search for the best match for a model in an image [5]. The Hausdorff distance, which originates in point set topology, is a metric for comparing point sets. The distance from the point set $A$ to the point set $B$ is defined as

$$h(A,B) \quad = \quad \max_{a \in A} \min_{b \in B} \|a - b\|. \qquad (2)$$

If $h(E',E) \leq r$ then every edge pixel in $E'$ is within $r$ of the closest edge pixel in $E$, there are no incoming edge pixels, and so $\rho_{in} = 0$. Similarly, if $h(E,E') \leq r$ then there are no outgoing edge pixels and $\rho_{out} = 0$.

Most applications of the Hausdorff distance use a generalization called the partial Hausdorff distance, which is

$$h_K(A,B) \quad = \quad K^{th}_{a \in A} \min_{b \in B} \|a - b\|. \qquad (3)$$

This selects the $K^{th}$ ranked distance from a point in $A$ to its closest point in $B$. If we select the largest such distance, we have the original Hausdorff distance defined in equation 2.

Applications which use the partial Hausdorff distance for matching can provide a fixed fraction $f = K/|A|$, which is $1 - \rho$. This specifies what fraction of the points in $A$ should be close to their nearest neighbor in $B$ at the best match. Alternatively, a fixed distance can be supplied, and the fraction of points in $A$ within this distance of their nearest neighbor in $B$ can be minimized.

### 3.4 Algorithm parameters

Because our algorithm is based on edges, it relies on the performance of the edge detector. Clearly our solution will not work on image sequences with minimal contrast, or in circumstances in which edge detection fails.

Our algorithm has several parameters that control its performance:

- the edge detector's smoothing width $\sigma$ and threshold $\tau$,
- the expansion distance $r$,

We have gotten good performance from a single set of parameters across all the image sequences we have tested. These parameters are $\sigma = 1.2$ and $\tau = 24$, for the edge detector, and $r = 6$, and were used to generate the data shown in this paper.

### 3.5 Compression tolerance

Most video will undergo some form of compression during its existence, and most compression methods are lossy. It is therefore important that our algorithm degrade gracefully in the presence of compression-induced artifacts. While edge detection is affected by lossy compression, especially at high compression ratios, we do not rely on the precise location of edge pixels. We only wish to know if another edge pixel is with $r$ of an edge. As a consequence, the precise location of edge pixels can be changed by image compression without seriously degrading our algorithm's performance. The experimental evidence we present in the next section comes from images that were highly compressed with the lossy JPEG compression scheme.

Figure 1 shows the results from an Eric Clapton MPEG that has been further compressed by JPEG-compressing each frame with a quality factor of 3. Our algorithm performs correctly even though the compression artifacts are so enormous as to make the sequence almost unviewable. Figure 1 also shows frame #40 at this compression rate.

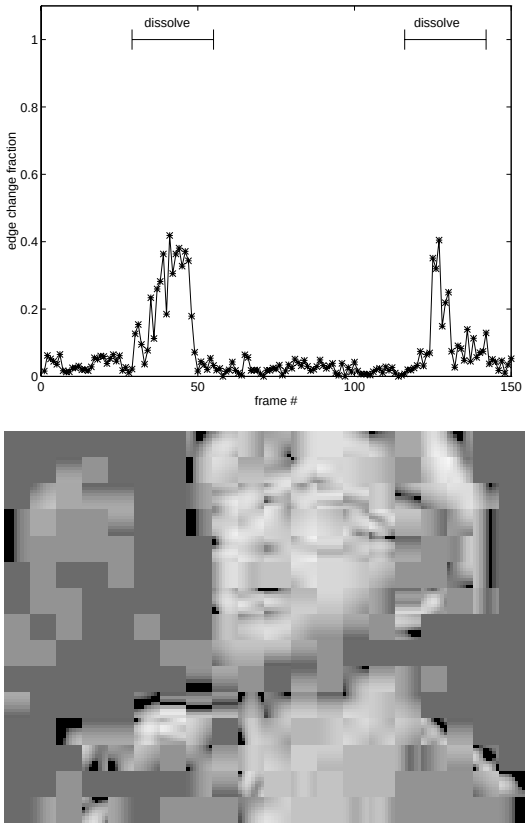



Figure 1: Clapton music video sequence results at .35 bits/pixel

## 4 Experimental Results

We have tested our algorithm on a number of image sequences, containing various scene breaks. To provide a comparison, we have also implemented two other intensity-based measures used to detect scene breaks. The first measure is the intensity histogram difference, which is used with slight variations in most work on scene breaks [6, 7, 14]. The second measure is the chromatic scaling method of Hampapur, Jain and Weymouth [4], a recent method for classifying scene breaks. We implemented a histogramming variant used by Zhang, Kankanhalli and Smoliar. For each of the 3 color channels we used the 2 most significant bits, for a total of $N = 64$ bins in the histogram; histograms are compared using the $L_1$ distance.

### 4.1 Sources of data

The image sequences used for testing are both MPEG and QuickTime movies. Our web page `http://www.cs.cornell.edu/Info/People/rdz/dissolve.html` contains links to the sites from which samples were obtained. Segment genres include music videos, television advertisements, NASA recordings and motion picture trailers. We found motion picture trailers to be particularly useful for their frequent and rich scene transitions. The sequences are highly compressed: MPEG movies typically have 0.80 bits per pixel and the QuickTime movies typically have 2.4 bits per pixel (including the audio track). All these sequences are color, so the compression ratios (from 24-bit color images) range from 10:1 to 69:1. Movies available on the web are also characterized by small frame sizes, most were $160 \times 120$ and the movie trailers were $156 \times 84$. These high compression ratios and small images are the result of using videos available on the World-Wide Web, which places a premium on compression to minimize bandwidth and storage costs. However, this makes our data set representative of the kind of video that is widely available today.

The image sequences we have collected fall into three classes. Several image sequences had easy scene breaks, which could be detected by all the methods we tried. For example, there may only be cuts, or there

may be minimal motion. Another class of image sequences caused errors for conventional intensity-based methods, but were handled correctly by our feature-based method. Examples include sequences with motion, and especially ones with both motion and dissolves. Finally, certain image sequences yielded incorrect answers, no matter which method we used. Examples include commercials with very rapid changes in lighting and fast-moving objects passing right in front of the camera with velocities as high as 50 pixels per frame.

### 4.2 Comparative results

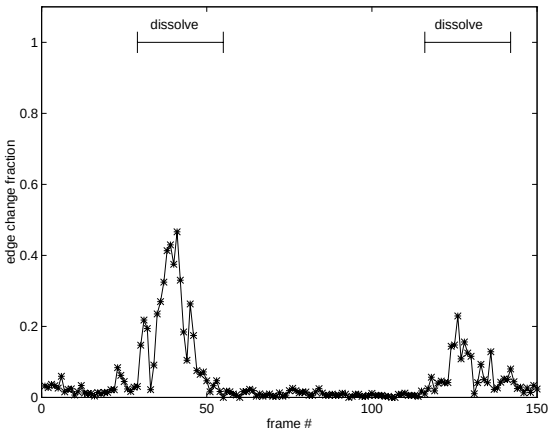
The Clapton video that we used to demonstrate compression tolerance is an example of a movie for which the other methods we implemented failed and has been used to benchmark other algorithms (e.g., [4]) . The Clapton sequence is difficult because of large object motion (the singer) and the long duration (over 30 frames) of the dissolves. Figure 2 compares all three methods on the Clapton video, all methods are intended to produce distinctive peaks at cuts and dissolves. The data used here is the original decoded MPEG, not the further JPEG-compressed data illustrated earlier.

The intensity histogram difference, shown in figure 2(b), shows a rise during the first dissolve, and it is possible that the dual threshold scheme of [14] would detect this (depending on the exact thresholds used). However, the second dissolve appears to be indistinguishable from the noise. Their method for handling motion would not help here, since the problem is a false negative rather than a false positive.

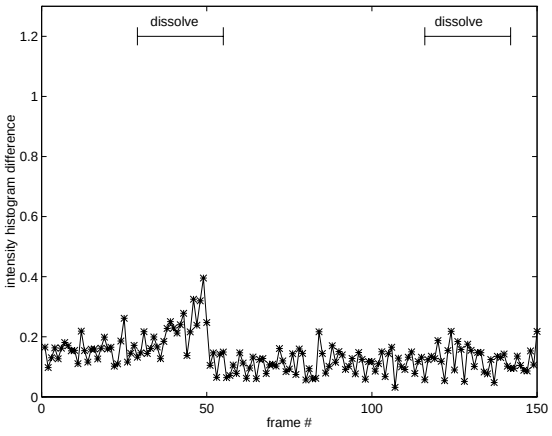
The chromatic scaling feature of [4] is shown in figure 2(c). As the authors state, their method has difficulty with dissolves involving motion.

### 4.3 Results on our data set

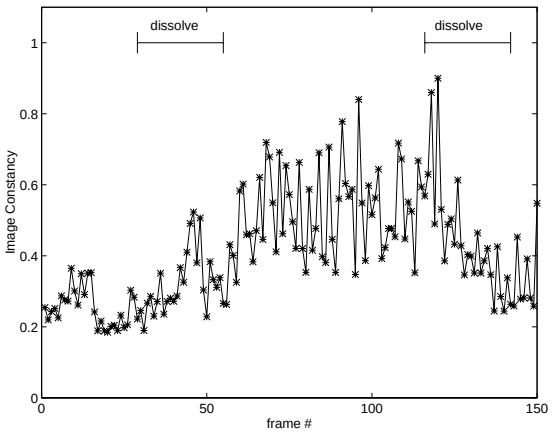
Our algorithm has been tested on an initial dataset of over 50 sequences mentioned in section 4.1. This data set consists of over 10,000 frames of video with a total of 162 cuts 28 dissolves and 24 fades. Results for our method and for the dual-thresholding color-histogramming method described in [14] are given. A single set of parameters is used with each method for all the sequences. The parameters for the color-histogramming method were obtained by searching for the parameters that minimized the number of errors (false negatives and false positives). The number of false negatives could not be reduced for color-histogramming without introducing a much larger number of false positives. (Note that the histogram method does not classify scene breaks. All the false




(a) Edge change fraction (`clapton`)




(b) Intensity histogram difference (`clapton`)




(c) Chromatic scaling feature (`clapton`)

Figure 2: Results from the `clapton` sequence

positives listed here exceed the single frame cut-detection threshold.)

| Method | False Negatives | | False Positives | |
|---|---|---|---|---|
| | cut | fade/dissolve | cut | fade/dissolve |
| Edge Chg. | 0 | 0 | 1 | 9 |
| Histogram | 1 | 7 | 7 | na |

### 4.4 Performance

The initial implementation of our algorithm is optimized for simplicity rather than for speed. However, its performance is still reasonable. Most of the processing time comes from the global motion computation. We have implemented our algorithm on a Sparc workstation with a single 50-MHz processor, and with 4 50-MHz processors. Because it is so easy to perform the motion compensation in parallel, we have obtained near linear speedups.

The table below shows our implementation's performance with motion compensation disabled, when running on the table tennis sequence.

| Image dimensions | 1 processor | 4 processors |
|---|---|---|
| $88 \times 60$ | 11.03 Hz | 44.13 Hz |
| $176 \times 120$ | 2.91 Hz | 11.63 Hz |
| $352 \times 240$ | .62 Hz | 2.48 Hz |

The next table shows the performance when using a simple Hausdorff-distance based motion compensation scheme. The running time is linear in the number of disparities considered. Data is shown for a range of disparities which has been adequate for images of these sizes.

| Image dimensions | 1 processor | 4 processors |
|---|---|---|
| $88 \times 60$ | 9.14 Hz | 36.57 Hz |
| $176 \times 120$ | 1.49 Hz | 5.95 Hz |
| $352 \times 240$ | .15 Hz | .6 Hz |

The performance on our corpus of MPEG movies was typically around 2 frames per second on a single processor.

### 4.5 Limitations

In our experience, our algorithm's failures involve false positives which result from very rapid motion of large objects. We are developing methods to help eliminate false positives, including the motion segmentation scheme described in section 5.2. These methods are not yet fully implemented and were not used when generating the above results.


Figure 3: Example of caption localization

## 5 Extensions

We also address two problems in video browsing beyond scene break detection. First, incoming and outgoing pixels can be used to identify other production effects, such as textual overlays. Second, our motion estimation methods can be extended to perform motion segmentation without local optical flow computation.

### 5.1 Production effects

We have extended our work on scene break detection to handle other production effects involving overlays. Many broadcast videos also provide some kind of textual overlay. For example, news broadcasts often begin a story by overlaying the location of the reporter. Similarly, a number of movies and television shows overlay the opening credits on top of a scene.

These textual overlays contain significantly more information than scene breaks, but they share some similarities. Textual overlays can suddenly appear and disappear (like a cut), or gradually fade in and fade out (like a dissolve). The pixels in the textual overlay should show up as incoming pixels when they appear, and as outgoing pixels when they disappear.

An example is shown in figure 3. The title caption is overlayed on a "busy" sequence of a waterfall and appears instantaneously, in a manner similiar to a cut. There are significant edges in the background, which causes some degradation in the output; however, the captions clearly show up as incoming edges.

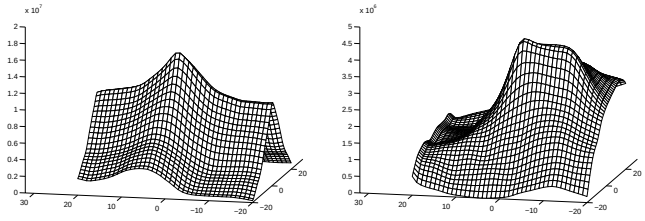
### 5.2 Motion segmentation

We have developed a simple scheme for motion segmentation that does not require estimating local optical flow. It is based on the motion compensation scheme described in section 3.1, which performs a single estimate of global image motion. The advantage of our approach is that global image motion can be estimated much more efficiently than local motion. We address two related problems: first, computing the secondary motion; and secondly, segmenting the image into pixels undergoing different motions. As an

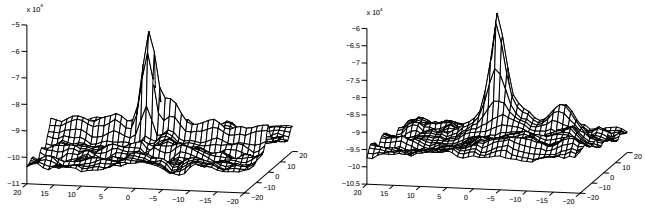
example, we will analyze a sequence where a camera pans to track a fast-moving ice skater.

### 5.2.1 Computing the secondary motion

Correlation-based methods for computing motion compute the shift (displacement) that makes the two images most similar. The similarity of the two images is a scalar function of the displacement; this similarity forms an *error surface*. The motion estimate is the displacement with the highest peak in the error surface.[2] Figure 4 shows the error surfaces for two motion algorithms, for a scenes with one and two motions. Besides the standard sum of squared differences ($L_2$) error measure, we also show the results from using the census transform [13] to compute motion. Note that the standard ($L_2$) correlation method yields an error surface which does not show the presence of the second motion. The census transform, on the other hand, shows the second motion clearly as a second bump.



(a) $L_2$ correlation



(b) Census transform correlation

Figure 4: Error surfaces for motion algorithms, for scenes with one motion (left) and two motions (right).

[2] For ease of viewing, we have displayed the similarity between the images. But we will use the "error surface" terminology, since it is standard.

We observed that in sequences with two motions there was always a clear secondary peak in the error surface. This suggested a simple method for computing the second motion, namely finding the second highest isolated peak. Note that such dual-peak detection methods have been used with local motion estimation before, to compute discontinuities [9] and to segment objects [11]. Results from this approach are shown in figure 5. The $x$ displacements are displaced for the primary (shown as a line) and secondary (shown with asterixes) motions. Note that around frame 60 the primary motion and the secondary motion switch. This occurs because the skater is moving towards the camera, and becomes larger than the background.

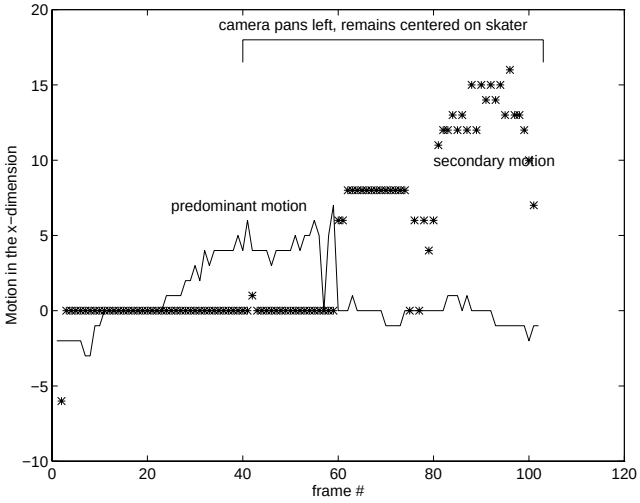



Figure 5: Primary and secondary motions in ice skater sequence.

### 5.2.2 Segmenting the image

Once the primary and secondary motion have been computed, we next attempt to classify the individual pixels. Our solution to this problem involves the individual pixel error scores when the images are shifted by the primary motion, and when they are shifted by the secondary motion.

We can now classify pixels based on their error scores at the primary and secondary displacements. Indistinct pixels (for example, those from areas with low texture) will have similar error scores at both displacements. Pixels with small errors at the primary displacement but large errors at the secondary displacement will be classified as undergoing the primary displacement. Similarly, pixels in the opposite situ-

ation will be classified as undergoing the secondary displacement.

The results of this approach can be improved by first box-filtering the error scores. Figure 6 shows some results on the ice skater sequence. Pixels determined to be undergoing the primary motion are displayed at right, while other pixels are shown in black. Note the two small black patches on the skater's torso; these are regions with minimal texture.

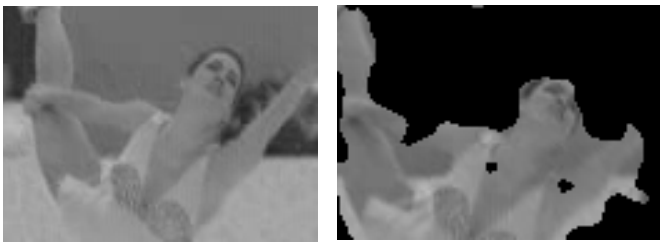


Figure 6: Original image, and pixels segmented as undergoing primary motion.

## Conclusions

We have shown how by using edges and motion it is possible to detect scene breaks, find captions, and perform motion segmentation. There are a number of obvious extensions to this work. For example, it is possible that the output of our caption detector can be piped into an OCR system. This would provide an automatic source of textual information about video without relying on manual annotation or speech recognition.


### Acknowledgements

Justin Miller is supported by a grant from the GTE Research Foundation.


## References


[1] Gilad Adiv. Determining three-dimensional motion and structure from optical flow generated by several moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(4):384–401, July 1985.

[2] Serge Ayer and Harpreet Sawhney. Layered representation of motion video using robust maximum-likelihood estimation of mixture models and MDL encoding. In *5th International Conference on Computer Vision, Cambridge, MA*, pages 777–784, 1995.

[3] M. Flickner *et al.* Query by image and video content: The QBIC system. *IEEE Computer*, 28(9):23–32, September 1995.

[4] Arun Hampapur, Ramesh Jain, and Terry Weymouth. Production model based digital video segmentation. *Journal of Multimedia Tools and Applications*, 1:1–38, March 1995.

[5] Daniel Huttenlocher, Greg Klanderman, and William Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[6] Akio Nagasaka and Yuzuru Tanaka. Automatic video indexing and full-video search for object appearances. In *2nd Working Conference on Visual Database Systems*, October 1991.

[7] K. Otsuji and Y. Tonomura. Projection-detecting filter for video cut detection. *Multimedia Systems*, 1:205–210, 1994.

[8] Alex Pentland, Rosalind Picard, Glorianna Davenport, and Ken Haase. Video and image semantics: Advanced tools for telecommunications. *IEEE Multimedia*, 1(2):73–75, 1994. Also appears as MIT Media Lab technical report 283.

[9] Anselm Spoerri and Shimon Ullman. The early detection of motion boundaries. In *International Conference on Computer Vision*, pages 209–218, 1987.

[10] John Wang and Edward Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing*, 3(5):625–638, September 1994.

[11] John Woodfill. *Motion Vision and Tracking for Robots in Dynamic, Unstructured Environments.* PhD thesis, Stanford University, August 1992.

[12] Ramin Zabih, Justin Miller, and Kevin Mai. A feature-based algorithm for detecting and classifying scene breaks. In *ACM Multimedia Conference*, 1995.

[13] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In Jan-Olof Eklundh, editor, *3rd European Conference on Computer Vision*, number 801 in LNCS, pages 151–158. Springer-Verlag, 1994.

[14] HongJiang Zhang, Atreyi Kankanhalli, and Stephen William Smoliar. Automatic partitioning of full-motion video. *Multimedia Systems*, 1:10–28, 1993.