Multi-Scale Video Cropping

Hazem El-Alfy
Department of Computer Science
University of Maryland
A.V. Williams Building
College Park, MD 20742
helalfy@cs.umd.edu

David Jacobs
Department of Computer Science
University of Maryland
A.V. Williams Building
College Park, MD 20742
djacobs@cs.umd.edu

Larry Davis
Department of Computer Science
University of Maryland
A.V. Williams Building
College Park, MD 20742
lsd@cs.umd.edu

ABSTRACT

We consider the problem of *cropping* surveillance videos. This process chooses a trajectory that a small sub-window can take through the video, selecting the most important parts of the video for display on a smaller monitor. We model the information content of the video simply, by whether the image changes at each pixel. Then we show that we can find the globally optimal trajectory for a cropping window by using a shortest path algorithm. In practice, we can speed up this process without affecting the results, by stitching together trajectories computed over short intervals. This also reduces system latency. We then show that we can use a second shortest path formulation to find good cuts from one trajectory to another, improving coverage of interesting events in the video. We describe additional techniques to improve the quality and efficiency of the algorithm, and show results on surveillance videos.

Categories and Subject Descriptors

I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis

General Terms

Algorithms

Keywords

Video cropping, Shortest path algorithm, Surveillance

1. INTRODUCTION

The use of video surveillance systems has been rising over the past decade. Most recently, the need to improve public safety and the concerns about terrorist activity have contributed to a dramatic increase in the demand for surveillance systems. The presence of these systems is very common in airports, subways, metropolitan areas, seaports, and in areas with large crowds. Modern video surveillance systems consist of networks of cameras connected to a control room that includes a collection of monitors. Typical control rooms have a much smaller number of monitors than cameras and far fewer operators than monitors. The monitors either cycle automatically through the cameras, or operators can manually choose any camera from the network and display it on a selected monitor.

We are interested in the design of future control rooms, and envision an architecture consisting of a large display wall which acts as a single entity, as opposed to matrices of independent monitors, or display regions with pre-specified monitoring tasks, as in current state of the art control rooms. The display wall assigns variable areas and locations to a subset of the available videos from surveillance cameras. The problems of determining *what* video to display, *where* to display it and *how* to do that, are addressed as follows. First, videos are "scored" according to their level of *interest* to human operators. Then, the scores are used along with other constraints (for example on minimum display duration once a camera is selected for display), to dynamically control the mapping of videos to the display space. Intuitively, higher score videos will be assigned both larger and more prominent parts of the display wall. Finally, we note that assigning a video to its display area typically requires resizing it. If the assigned space is small, simply reducing the resolution of the original video might render its contents to be illegible. Reducing the resolution is often needed also to save bandwidth in transferring the video or saving it for archiving purposes. In either situation, *cropping* the video before resizing it results in videos that should be easier for humans to interpret. An overview of this system's architecture is illustrated in figure 1. This paper describes only the video cropping components of the system, leaving other components for future papers.

In surveillance applications, video cropping helps to focus the attention of operators on specific parts of the scene. Activity occurring in the background or at corners of the display area might pass unnoticed by operators, due to other activity in more prominent areas of the scene. The tradeoff here is between the size of the cropped video and the information loss. In scenes with several regions of simultaneous activity, allowing the *cropping window* to jump occasionally between these regions supports coverage of multiple activities, while keeping the resulting cropped video small. In addition, it is crucial in surveillance applications to process video online −as it becomes available− and cannot require

MM'07, September 23–28, 2007, Augsburg, Bavaria, Germany.

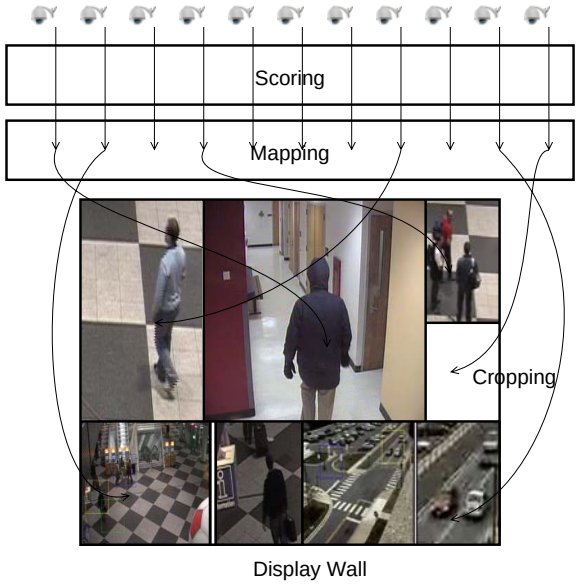

Figure 1: Diagram of the proposed future control room

the entire video to be available beforehand. These points are amplified in the body of the paper.

We define the video cropping problem to be the determination of a smooth path of a cropping window that captures "salient" foreground throughout the video. The window can have variable size, which introduces virtual zoom-in and zoom-out effects. It is allowed occasional jumps through the video, similar to scene cuts in filmmaking. We use motion energy as a measure of the "saliency" of a cropping window trajectory. Unlike previous approaches, we optimize the saliency globally for the whole trajectory, rather than for individual frames or shots, as follows.

First, the video is modeled as a graph of windows, with its edge weights reflecting saliency captured by windows, efficiently computed using integral images [16]. Then, a shortest path algorithm finds the window trajectory that captures the overall maximum motion energy. The resulting trajectory is smoothed to remove jiggles and staircase-like appearance. This procedure is repeated several times on the remaining parts of the video to capture the remaining saliency. This results in obtaining a set of disjoint smooth paths of cropping windows that capture as much saliency as possible. A secondary optimization procedure produces the final path by alternately jumping between the paths computed earlier, selecting which one to follow at which time, so as to maximize both captured saliency and covered regions of the original video. Long videos are processed by breaking them into manageable sub-videos, while allowing overlap between consecutive subvideos, to produce smooth transitions. Our algorithm is applied to a collection of real surveillance videos. Several experiments are performed to determine the optimal choices regarding issues such as where to cut a video into sub-videos, the amount of overlap between them and how long a segment should be displayed before jumping to another. Some display configurations are compared, such as cropped video alone, side by side with original video, or video-in-video (like commercially available picture-in-picture).

We are mainly interested in surveillance applications. Typically, much more video than operators can observe is available. In addition, this video is unedited, and more importantly, not focused on any agent in the scene. This has made our approach different from earlier approaches that dealt with edited videos, such as movies, news reports, or classroom video. In particular, our method offers the following contributions:

- a variable size cropping window which results in a smooth zoom in/out effect,
- multiple cropping windows to cover more agents in the scene,
- only a relatively short video segment needs to be processed at a time −not the complete video− which makes the algorithm an online algorithm, and
- we show empirically that by stitching together results from short segments of a video, we get a result identical to the globally optimal one, given the entire video.

The rest of this paper is organized as follows: Section 2 reviews related work. Then, the video cropping problem is formally defined in section 3. In section 4, we present our approach to solve the problem, while section 5 presents the results. Finally, closing remarks and conclusions can be found in section 6.

2. RELATED WORK

Research has been performed in the area of visual attention to detect salient areas in images and video from low-level features. Itti et al. [11] use orientation filters in addition to color and intensity to detect salient parts of images. Later, Itti and Baldi extend this method to work with videos, using a statistical model for time [10]. A probabilistic approach is used by Kadir and Brady as a measure of local saliency in images [12]. Their method is generalized by Hung and Gong by including the time variable to quantify spatio-temporal saliency in videos [9].

Many methods for cropping still images automatically have been published. For example, Suh et al. [15] use Itti's saliency model, along with face detection, to crop informative parts of images before reducing them to thumbnails. The same model is also used by Chen et al. [2], with the addition of text detection, to find regions of interest in images for adaptation to small displays. Xie et al. [18] study the statistics of users' interaction with images on small displays to determine regions of maximum user interest.

Much less work has been done in the area of *video cropping*, or detecting interesting space-time regions of a video. Fan et al. [6] determine areas of interest in individual frames, then combine them smoothly. Wang et al. [17] split surveillance video into interesting and non-interesting sequences using a threshold on their motion content. Non-interesting sequences are zoomed out and transmitted/displayed at reduced frame rates, while interesting ones are displayed at full frame rate and zoomed in to clusters of high motion energy. Both of these methods are optimized locally and need not produce videos that are globally optimal, in terms of their saliency content. More recently, Kang et al. used a space-time saliency measure to cut out informative portions

of a video and pack them into a video of smaller resolution and shorter duration [13]. This approach models videos and processes them as a whole, making them unsuitable for relatively longer videos, or for continuous surveillance video.

Liu and Gleicher [14] edit videos using a fixed size cropping window. Since they focus mainly on editing feature films, the window moves are restricted to pans and cuts whose parameters are optimized over individual shots. To keep the original structure of the film, the authors introduce a set of heuristic penalties that limit the motion of the cropping window. Although this approach works well with professionally captured films, it may not generalize well to unedited raw surveillance video, which generally have wider fields of view, are not focused on a main subject, and consist of a single shot. Another recent example of automatic editing of specific types of videos is that of classroom video editing. Heck et al. [8] find an optimal shot sequence, from a set a virtual shots, that have been selected based on prior knowledge of the scene and video content. These methods also are not well suited to raw video.

3. PROBLEM DEFINITION

Given a video sequence, our goal is to determine a smooth trajectory for a variable size window through the video, that maximizes the captured saliency over all such trajectories and window sizes. Occasional jumps are allowed to include as much saliency as possible. More formally, we consider the problem of optimizing a single trajectory. Assume the input video segment has T frames. Each frame t can be covered by a set of n variable size overlapping windows. These windows are labeled $W_{i,t}$, with i being the window number, selected from an index set $\mathcal{I} = \{1, 2, \ldots, n\}$. Define the cross product set $\mathbb{I} = \mathcal{I} \times \mathcal{I} \times \cdots \times \mathcal{I} = \mathcal{I}^T$. Then, we want to solve the problem:

$$\arg\max_{\mathcal{Q}} \sum_t S(W_{i,t}) \tag{1}$$

where $S(\cdot)$ is a saliency measure, $\mathcal{Q} \in \mathbb{I}$ is the window sequence that maximizes the saliency, and $i \in \mathcal{I}$. It is more desirable to minimize functions, thus, the saliency function $S(\cdot)$ can be replaced by a cost function $C(\cdot)$ that decays with increasing saliency. Model (1) doesn't enforce spatial smoothness. To guarantee a smooth path, windows in two consecutive frames are restricted to be close to one another and with little area variation. The problem is thus formulated as:

$$\arg\min_{\mathcal{Q}} \sum_t C(W_{i,t}) \tag{2}$$

$$\text{such that:} \quad \begin{aligned} & d(W_{i',t-1}, W_{i,t}) < d_{max} \\ & |\mathcal{A}(W_{i',t-1}) - \mathcal{A}(W_{i,t})| < \mathcal{A}_{max} \end{aligned}$$

where $d(\cdot,\cdot)$ is a distance measure and $\mathcal{A}(W)$ is the area of window W.

Our main contribution in this paper is that we optimize globally, over the whole video, rather than locally for individual frames, that are later combined. This approach maximizes the *total* captured saliency and provides smoother results.

4. VIDEO CROPPING APPROACH

Solving problem (2) by trying all possible paths is prohibitively expensive. Instead, we employ a dynamic programming approach, using the following procedure. First, motion energy is extracted through frame differences. Then, we build a weighted directed graph, with the cropping windows as its vertices, and edge weights measuring the motion energy. A shortest path algorithm through the graph selects the first optimal trajectory, which minimizes equation (2). This trajectory is then smoothed, and the motion it contains is "wiped out" from the original frames. This procedure is repeated to capture some of the remaining energy. A second graph is built out of the resulting trajectories with shortest path run once again to determine the optimal combination of paths. In the final cropped video, one trajectory is followed at a time, with occasional jumps between these trajectories. For long videos, video segments are chosen to overlap with their immediately preceding ones, and are processed similarly. This allows smooth transitions between the respective trajectories in each segment. The whole process is summarized in figure 2. In the remainder of this section, the above steps are presented in more detail.

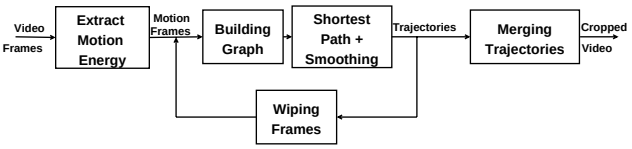

Figure 2: Overview of our approach to crop a video segment.

4.1 Extracting motion energy

In our implementation, we use motion energy as a measure of saliency. Earlier approaches [6, 14] used centered frame saliency maps, face and text detectors in addition to motion contrast, to crop movies and news. These detectors are too specific to be used with surveillance video.

Motion energy is efficiently computed in real time, and captures important activity in surveillance video. We compute frame differences and threshold them to detect motion, then apply morphological operations to the resulting motion frames. In particular, the *opening* operation is applied to remove detected small noisy areas, then a *closing* operation connects nearby fragments. The motion energy is computed to be the number of 1's in the resulting binary images. The remainder of the algorithm works on these preprocessed difference frames.

4.2 Building the graph

The video is modeled as a weighted directed graph as follows. Let $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ be the graph. Each frame is sampled by n overlapping cropping windows of various sizes. Then, each window is represented by a vertex $v \in \mathbf{V}$. This makes the total number of vertices in the graph $|\mathbf{V}| = nT$ for a video segment of T sample frames. The restrictions in the problem formulation (2) are implemented by allowing a window to move only to neighboring window positions, or to grow or shrink by no more than one size step between two consecutive frames. This translates to adding an edge to $\mathbf{E}$ only if it connects a pair of vertices that represents a pair of windows between which a step is allowed. For a certain ordering of the vertices $\mathbf{V}$, the adjacency matrix of $\mathbf{G}$ is banded. This is a sparse matrix that can be efficiently stored using an adjacency list graph representation, to store just the bands. With $b \ll nT$ neighbors allowed per window, we

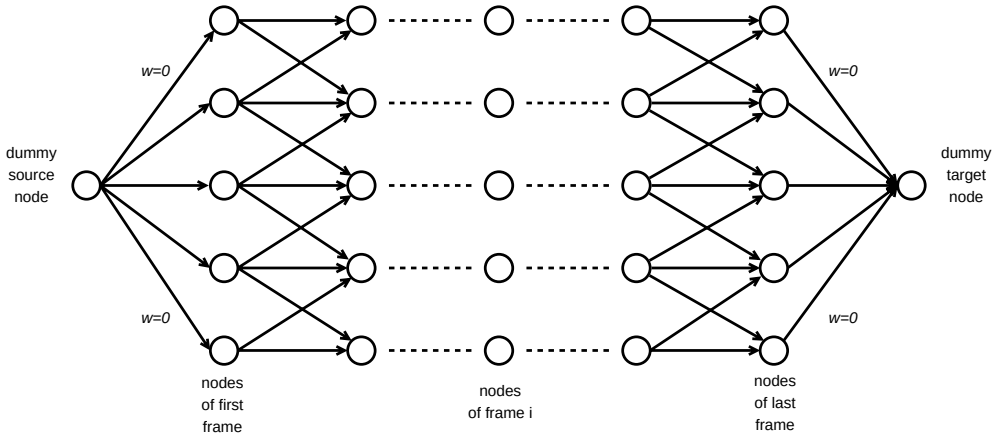

Figure 3: Graph modeling the video.

Figure 4: Using window energy density as a measure of motion energy favors smaller windows, cropping away small parts of objects, such as heads in humans (left: original frame, showing the location of the cropping window; right: cropped frame).

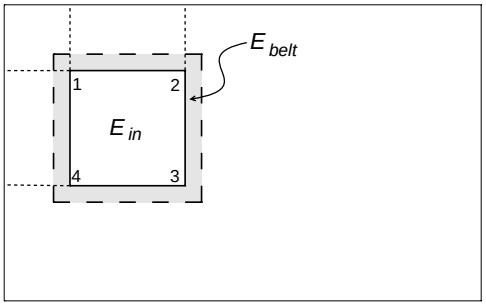

Figure 5: Cropping window configuration, with surrounding belt area. The numbers represent the labels of the inner (cropping) window's corners.

have a graph of $\mathcal{O}(nT)$ edges. An additional "source" node and a "target" node are added to the beginning and end of the graph. A model graph is shown in figure 3.

To find a trajectory that maximizes the captured motion energy, we need to define a window energy measure. Using the total energy enclosed in a window always favors larger windows, which makes the cropping useless. On the other hand, using the energy density as a measure favors smaller windows, which can be more densely filled, with little empty space. This results in cropping away smaller parts of objects, such as heads in humans, as is illustrated in figure 4. This is definitely an undesirable result.

To balance both effects, we use an energy density measure, penalized by the total energy in a thin surrounding belt. This is similar to center-surround representations (e.g. Itti et al. [11]). It prevents the window from cropping away parts of objects, making it large enough to fit a single "cluster" of objects, while leaving distant ones for subsequent trajectories to capture. The *multi-scale energy function* of a window W is defined as:

$$E(W) = \frac{E_{in}}{A_{in}} - \frac{1}{K} E_{belt}, \tag{3}$$

where E_{in} is the motion energy (number of 1's) in window W, A_{in} is the area of window W in pixels (i.e., the number of pixels in W), E_{belt} is the motion energy in the surrounding belt, and K is an empirically chosen constant, that determines how much penalty to assign to cropped away parts. The diagram in figure 5 illustrates these parameters. Next, the energy measure (3) is put in a form that can be minimized, to fit as a cost function $C(\cdot)$ in equation (2). In addition, to suit the shortest path algorithm used, the edge weights must be normalized to a non-negative integer range. We choose this range to be from 0 to 100. If a transition is allowed from vertex i to vertex j, an edge is added to the graph with weight $w(i,j)$ computed as:

$$w(i,j) = \left\lfloor 100 - 100 \max\left(\frac{E_j}{A_j} - \frac{1}{K} E_{belt_j}, 0\right) + 0.5 \right\rfloor \tag{4}$$

where E_j, A_j and E_{belt_j} are all related to the window represented by vertex j. The function $\lfloor x + 0.5 \rfloor$ rounds x to the nearest integer. E_j and E_{belt_j} are computed by summing the motion pixels (1's) for each window in each preprocessed difference frame. This is a time consuming operation if performed in a straightforward manner. Instead, we use integral images in a manner similar to Viola and Jones in image analysis [16] based on an idea by Crow for texture mapping [3]. Given an image $i(x,y)$, the integral image ii accumulates pixels to the top and left of each pixel, as defined by:

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y').$$

The integral image is computed in one pass using the two recurrences:

$$\begin{array}{rcll} s(x,y) & = & s(x,y-1)+i(x,y) & \text{(row sum)} \\ ii(x,y) & = & ii(x-1,y)+s(x,y) & \text{(integral image)} \\ s(x,-1)=0 & ; & ii(-1,y)=0 & \end{array}$$

The integral image is computed once for every frame, then every window sum, E_j, is computed using only 4 operations. Thus, we avoid computing the cumulative sum for each window, which includes a lot of redundancy due to the overlap between windows. Given a window with corners $\overrightarrow{x_1}$, $\overrightarrow{x_2}$, $\overrightarrow{x_3}$ and $\overrightarrow{x_4}$ in clockwise direction, starting from the top left (see figure 5), the cumulative sum in that window can be computed as:

$$ii(\overrightarrow{x_3}) - ii(\overrightarrow{x_2}) - ii(\overrightarrow{x_4}) + ii(\overrightarrow{x_1}).$$

4.3 Shortest path

The shortest path from the source node to the target node is computed using Dijkstra's algorithm [5]. Benefiting from the special structure of this graph, some modifications are introduced into the algorithm to improve performance. Early termination can be achieved by halting the search prematurely when the first vertex (window) in the last video frame is reached, rather than waiting until all the vertices in the graph are labeled.

Dijkstra's algorithm is mainly slowed down by the search for the closest vertex to the source, in a list of temporary labeled nodes, at each iteration. With N vertices in the graph, the asymptotic running time of the algorithm is $\mathcal{O}(N^2)$ or $\mathcal{O}(N\log(N))$ depending on the data structure used to implement the list of temporary labeled nodes. This running time has been reduced in our implementation in two ways. The multiplying factor is directly affected by the size of the list of temporary labeled nodes. In our runs, we note that the maximum number of nodes in that list, over all iterations, is just around 1% of the total number of vertices in the graph.

The running time is also reduced an order of magnitude, from quadratic to linear in the number of vertices, using Dial's implementation [4]. In the original algorithm, Dial stores temporarily labeled nodes in *buckets*, indexed by the nodes' distances. This makes the search for the minimum distance node $\mathcal{O}(1)$. With C being the maximum edge weight (100 in our graph), Dial's original algorithm maintains $NC+1$ buckets, which may be prohibitively large. A remark made by Ahuja [1] reduces the space requirements to only $C+1$ buckets. During each iteration, the difference between the maximum and minimum finitely labeled nodes cannot exceed C. Hence, temporarily labeled nodes can be hashed by their distance labels into just $C+1$ buckets.

4.4 Smoothing

The shortest path resulting from the previous stage has a noisy appearance, which can be best compared to a jittery or shaking cameraman, from the point of view of the cropped video. Two levels of smoothing are applied to the trajectory. First, a moving average smoother is applied to the trajectory. With $y(t)$ being the original data (raw trajectory) at time t and $y_s(t)$ the smoothed one, the difference equation is:

$$y_s(i) = \frac{1}{2N+1}\left(y(i+N)+y(i+N-1)+\cdots+y(i-N)\right),$$

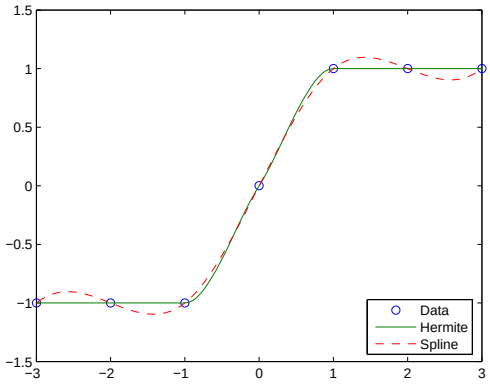

Figure 6: Piecewise cubic Hermite interpolation performs better than cubic spline when the original data is staircase.

where N is the "radius" of the span interval. This has the effect of a lowpass filter, reducing the shaky appearance of the trajectory. Truncating the result completely removes any such artifacts, but results in a staircase-like appearance. A second level of smoothing is done by interpolating a piecewise cubic Hermite polynomial to a sub-sampled version of the data. This polynomial interpolates the data points and has a continuous first derivative. However, unlike cubic splines, the second derivative need not be continuous. This property is more suitable for our staircase data, since it avoids excessive oscillations. Figure 6 illustrates our situation where Hermite interpolation produce more stable results.

4.5 "Wiping out" captured motion

A single moving cropping window might not capture all motion energy in the video. In situations where several activities occur simultaneously in separated regions of the scene, panning back and forth between these regions produces a blurry video that covers the mostly empty area between them. To solve this problem, we compute several independent window trajectories, each covering a different activity in the same video. Later, we show how to control jumps between these paths. The window trajectories are determined by repeating the above procedure of modeling the video using a graph, computing the shortest path and smoothing. Between two consecutive computations of window trajectories, all captured objects that overlap with the first are removed from the motion frames. This "wiping" procedure guarantees that an ensuing window path will not cover parts of previously captured objects, thus avoiding two similar trajectories.

4.6 Merging trajectories

Once enough window paths are computed, we merge them into a single path that captures as much motion as possible, and covers as many regions of the original video as possible. Figure 7 illustrates an example solution to this problem. After following a segment of a path for some period of time, a jump to a segment of another path results in covering another activity region, without panning through the scene. This appears in the final cropped video as a cut.

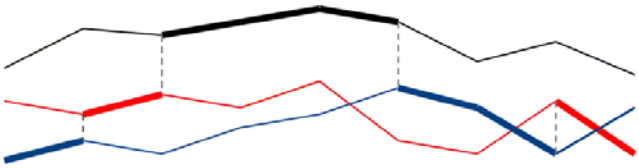

Figure 7: Example of merging three windows trajectories. The horizontal axis represents time; the thick lines are for the segments of the trajectories picked at that time; and the dashed lines are for times where jumps occur.

To compute the final merged trajectory, we solve a second optimization that uses a shortest path algorithm through the trajectories. A second graph $\mathbf{G}' = (\mathbf{V}', \mathbf{E}')$ is built, with the list of vertices $\mathbf{V}'$ formed by concatenating nodes from all computed trajectories. This list of nodes is duplicated k times, with the i^{th} copy of a node from path p representing that this path has been followed for i steps (frames/nodes). The number of frames after which a switch between trajectories is allowed without penalty is k. This allows us to keep track of how long (in frames) a single trajectory has been followed.

There always exists an edge $e \in \mathbf{E}'$ from every node to the next node in the same path, and edges to next frame nodes in the other paths if a "cut" is allowed at that time. The weight function w' is the cost associated with the representative window in the original graph $\mathbf{G}$, computed as in (4), if the edge connects two nodes in the same path. However, if a path switch occurs, a penalty function is added to that weight. Intuitively, a higher penalty is associated with switches to closer trajectories, to favor more coverage of the original video, and to avoid "jumps". Another penalty is added to the window cost, that decays with the time that has been spent following a certain trajectory. This latter penalty inhibits frequent successive cuts, which might distract the operator. These penalties are fractions of the window cost, to make them comparable to the window cost penalty (2) in the global optimization. They are determined empirically, based on runs conducted on several videos. We also noticed that these penalties are consistent with filmmakers' heuristics, who might use frequent cuts purposely only when special "pacing effects" are required. Similarly, filmmakers would avoid a cut to a nearby location, which is known in cinematography as a "jump cut". These rules have been followed in feature film editing [14] when creating virtual cuts.

4.7 Processing long videos

Real-time surveillance applications require online processing of video. Additionally, available processing resources constrain the longest video that can be processed as a whole. Our approach allows us to process long videos by breaking them into segments, with some overlap. Each segment is processed separately, resulting in consecutive overlapping trajectories. By piecing together corresponding paths from each segment and removing the overlap, continuous smooth trajectories result. This is further discussed in the section on results, where experiments are performed to determine appropriate locations to break the video into segments and the amount of overlap needed for smooth transitions.

4.8 Video display

Several display methods for the resulting cropped video are considered. The most obvious is just displaying the cropped region. This is the most space efficient but may result in seeing the cropped video out of its context. To keep both context and content, we display both the original and cropped videos side by side, with the original one shrunk to fit the display space. This display method is not space efficient, though.

To display context, while keeping a reasonable video size, we suggest a video-in-video display style. The familiar style is to display the "sub"-video (cropped video here) in a corner of the "full" video. In our approach, we find the optimal display location automatically while computing the shortest path. It is chosen to be largest border window, that covers the minimum activity in the video. In terms of our implementation, this is the largest window on the border that contains the least motion energy. The video-in-video window is of fixed size and location, but can be allowed occasional changes over longer periods of time.

5. EXPERIMENTAL RESULTS

We have applied our approach to several surveillance videos. Typical cropped frames from an airport surveillance video (three-window) and from a traffic intersection video (two-window) are shown for fixed size cropping, in figures 8 and 9 respectively. For now, the number of cropping windows is manually selected for each video to cover all people who appear in the scene, occasionally jumping between them. The figures illustrate single frames in which only one cropping window is displayed. Variable-size single-path windows, video-in-video and multi-camera display are shown in figures 10, 11 and 12.

We have timed the performance of our system on the airport surveillance video. This is a high quality 720×480, 30 fps video. Excluding background subtraction and disk operations, which are done at independent stages, we were able to process a 66 second video segment (2000 frames), as a single segment, in about 400 seconds, for a fixed-size, single-trajectory window. This is equivalent to processing about 5 frames per second. Lower resolution videos (320×240) can be processed at almost real-time rates. Experiments are done on a 2.8 GHz dual processor CPU with 1 GB of memory. Using a variable size window with 6 size steps multiplies the size of the problem by a factor of 6, thus dramatically reducing the processing speed to about 0.88 fps. Computing more trajectories reduces the processing speed even further.

A key contribution of our system over previous approaches is that it is able to process segments of video sequentially, eventually allowing for processing flows of input video online. This makes our system able to process surveillance video online, without the need to have the whole video available completely before processing. This feature, along with the processing rates for the basic fixed-size window low-resolution video cropper, makes it useful for real-time surveillance systems. We hope that more efficient implementations and faster hardware will allow the full feature variable-size multi-window cropper to work in real-time in the future.

When splitting a long video into segments, we compare the solution, i.e. cropping window path, to that obtained if the complete video is processed as a single block. Experi-

Figure 8: Four typical frames from an airport surveillance video, with three trajectories and fixed window size. In each frame, the original video frame, showing the locations of the cropping windows, is to the left and the cropped frame is to the right, resized to the original's height.

Figure 9: Two typical frames from a traffic intersection surveillance video, with two trajectories and fixed window size. In each frame, the original video frame, showing the locations of the cropping windows, is to the left and the cropped frame is to the right, resized to the original's height.

Figure 10: Variable size single trajectory results. Compared to the fixed size results, less empty area is retained around small cropped objects.

Figure 11: Video-in-video results: the cropped video location is selected automatically

ments carried out on several videos show that window paths obtained by processing a sequence of subvideos are identical to those obtained by processing the longest video that could fit in memory as a whole.

Processing video segments individually results in small jumps when piecing the resulting cropping window paths together. To obtain smooth transitions between consecutive segments, the locations to split the video should be carefully selected. In addition, some amount of overlap is required between segments, to obtain a transition at the closest point between window paths. Two sets of experiments are described in the following subsections. The first one determines the best locations to split a video into segments, and the second one determines the amount of overlap required between segments to obtain smooth transitions.

5.1 Splitting a long video into segments

The first problem to be solved is to find *where* to split the video into segments. The rule of thumb here is to avoid splitting the video at a point where little or no activity occurs. This might result in a large discontinuity of the window path at the split point. Regardless of the overlap between the consecutive segments (see next section), choosing the split point inappropriately might still cause discontinuities. Our experiments for fixed-size cropping windows show that a split at any time where significant motion is detected will result in smooth transitions. Figure 13 illustrates a situation where a video is split into two segments during a period of very little motion energy. This results in a large jump at the split location, despite the long overlap region between the two segments. The x-coordinate of the center of the cropping window is shown on the vertical axis, along with the motion energy content, normalized to a scale of 300, for visualization. The solid line represents the result of processing a one-minute-video segment as a whole, and the dashed lines are for the results of splitting in two sub-segments.

5.2 Overlap in video segments

Once the locations at which to split the video are chosen, the amount of overlap between these segments has to be determined. A compromise has to be made between smoothness, requiring longer overlap, and efficiency, requiring shorter overlap. Many experiments were conducted to determine average and shortest overlaps for which smooth transitions can still be obtained. We first start with the fixed size window case. For all the videos considered, a maximum of 20 overlap frames was required. By tracing the shortest path algorithm, we record for each node the last frame (largest time index) that was processed to determine the node's distance from the source node (depth in graph). In Dijkstra's shortest path algorithm, once a node's distance is permanent, a shortest path passing through it is not changed any further (although the path itself is not determined until the backward pass). This property indicates how much video needs to be processed ahead to compute the final trajectory. We found that no more than the nodes of 18 frames ahead are needed to compute the vertices' permanent distances, hence, the shortest path. Experiments carried out for the variable window size case show that longer overlaps are required. In some situations, we were able to obtain smooth transitions for 50 frame overlaps, but more experiments still need to be done to determine requirements for a smooth result.

To further illustrate the point of segment overlap, figure 14 shows a fixed-size-window shortest path computed for a 1000 frame video (solid), plotted against shortest paths for sub-videos (dotted), starting at the same frame (source node) but ending much before the original video. The discrepancies are found in the range of 8-12 frames. Figure 15 shows a 1000 frame video that serves as ground truth. It is split into five 200-frame-segments with 40 frame overlap on each side. The cropping window trajectories obtained using each method are plotted against each other. Again, differences between the two trajectories are found not to exceed 20 frames around split points. Elsewhere, the two solutions are identical. This shows that our method tends to compute trajectories that are globally optimal, even though we are processing video segments individually.

6. CONCLUSIONS

Modern surveillance systems contain so many high resolution videos that it is not possible for operators to view them all. Automatic systems that select the most relevant portions of the video will make it possible to display important parts of the video at higher resolution. To do this, we have created a system that crops videos to retain the regions of greatest interest, while also cutting from one region

Figure 12: Multi-camera display issues should be addressed in later work. Original video to the left; cropped video to the right.

of the video to another, to provide coverage of all activities of interest.

At the core of our system is the use of a shortest path algorithm to find the optimal trajectory of a sub-window through a video, to capture its most interesting portions. This relies on a measure of interestingness using a center-surround operator on the video's motion energy. By normalizing this operator for scale, we produce trajectories that alter the zoom of the sub-window, allowing us to smoothly track objects as their apparent size changes. These methods allow us to find a globally optimal trajectory, given access to the entire video. In practice, we show that we can process the video a few seconds at a time and obtain results identical to the globally optimal solution, but with much less processing time and latency. In addition, we smooth the trajectory of the cropping video, to reduce unpleasant artifacts. We also show that we can compute a number of independent trajectories, and use a shortest path algorithm to find the best way of cutting between these. This produces a cropped video in which we pan across the original video, following objects of interest, with occasional cuts to display different objects.

We plan additional work to improve upon these results in the future. First, in this paper we have focused on finding algorithms that can maximize the amount of information extracted from videos by cropping and cutting. We plan to perform user studies to determine the extent to which this can assist an operator in performing a real surveillance task (some relevant user studies of related systems can be found

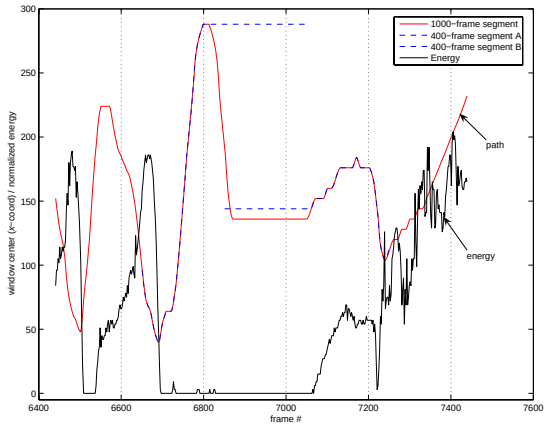

Figure 13: Effect of splitting a video at a time of little activity

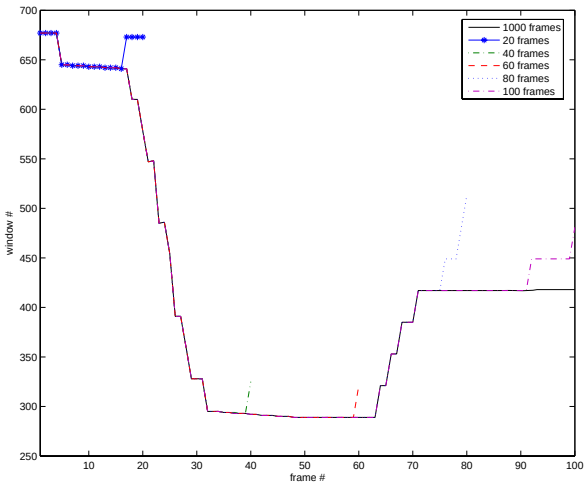

Figure 14: Shortest paths of video prefixes.

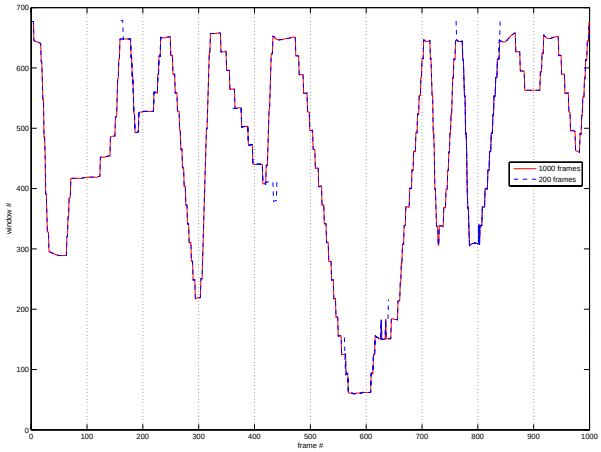

Figure 15: Splitting a video into 5 segments.

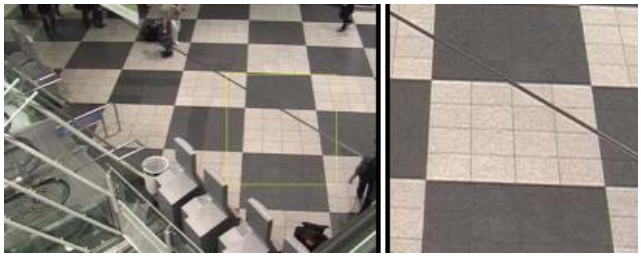

Figure 16: A cropped frame resulting from panning across a largely empty scene (left: original frame, showing the location of the cropping window; right: cropped frame).

in [6, 7, 17]). User studies will also allow us to assess the importance of parameters that affect video quality, such as the frequency of cuts or the smoothness of trajectories. We also plan to explore the use of temporal cutting, in which we eliminate frames in which not much is happening (see figure 16). This will be particularly important in large scale systems that cycle through many videos.

Our work is based on an overall vision for control rooms of the future, in which a system determines which portions of which videos are most relevant to a surveillance task, and maps these to large displays. Such a system will combine automatic processing and user input to help operators focus on the most relevant visual nuggets from a huge sea of visual information. This paper contributes to that vision by showing how to use tools from optimization to find the most important portions of a video.

7. ACKNOWLEDGMENTS

The research described in this paper was supported in part by the US Government's VACE program.

8. REFERENCES

[1] R. K. Ahuja, K. Mehlhorn, J. Orlin, and R. E. Tarjan. Faster algorithms for the shortest path problem. *J. ACM*, 37(2):213–223, 1990.

[2] L.-Q. Chen, X. Xie, X. Fan, W.-Y. Ma, H.-J. Zhang, and H.-Q. Zhou. A visual attention model for adapting images on small displays. *Multimedia Systems*, 9(4):353–364, Oct 2003.

[3] F. C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques (SIGGRAPH'84)*, volume 18, pages 207–212, 1984.

[4] R. B. Dial. Algorithm 360: Shortest-path forest with topological ordering [H]. *Commun. ACM*, 12(11):632–633, 1969.

[5] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] X. Fan, X. Xie, H.-Q. Zhou, and W.-Y. Ma. Looking into video frames on small displays. In *ACM international conference on Multimedia (MM'03)*, pages 247–250, 2003.

[7] A. Girgensohn, J. Adcock, M. Cooper, J. Foote, and L. Wilcox. Simplifying the management of large photo collections. In *Human-Computer Interaction (INTERACT'03)*, pages 196–203, 2003.

[8] R. Heck, M. Wallick, and M. Gleicher. Virtual videography. *ACM Transactions on Multimedia Computing, Communications and Applications (TOMCCAP)*, 3(1):4, Feb 2007.

[9] H. Hung and S. Gong. Quantifying temporal saliency. *British Machine Vision Conference (BMVC '04)*, pages 727–736, Sept. 2004.

[10] L. Itti and P. Baldi. A principled approach to detecting surprising events in video. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1:631–637, June 2005.

[11] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 20(11):1254–1259, Nov. 1998.

[12] T. Kadir and M. Brady. Saliency, scale and image description. *International Journal of Computer Vision (IJCV)*, 45(2):83–105, Nov. 2001.

[13] H.-W. Kang, X.-Q. Chen, Y. Matsushita, and X. Tang. Space-time video montage. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2:1331–1338, June 2006.

[14] F. Liu and M. Gleicher. Video retargeting: automating pan and scan. In *ACM international conference on Multimedia (MM'06)*, pages 241–250, 2006.

[15] B. Suh, H. Ling, B. B. Bederson, and D. W. Jacobs. Automatic thumbnail cropping and its effectiveness. *ACM Symposium on User Interface Software and Technology (UIST '03)*, pages 95–104, Nov. 2003.

[16] P. Viola and M. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–157, May 2004.

[17] J. Wang, M. J. Reinders, R. L. Lagendijk, J. Lindenberg, and M. S. Kankanhalli. Video content representation on tiny devices. *IEEE International Conference on Multimedia and Expo (ICME'04)*, 3:1711– 1714, June 2004.

[18] X. Xie, H. Liu, S. Goumaz, and W.-Y. Ma. Learning user interest for image browsing on small-form-factor devices. In *SIGCHI Conference on Human Factors in Computing Systems (CHI'05)*, pages 671–680, 2005.