SLIDING-WINDOW RIBBON CARVING FOR VIDEO CONDENSATION

Huan-Yu Wu

Dec 11, 2009

Boston University

Department of Electrical and Computer Engineering

Technical report No. ECE-2009-03

SLIDING-WINDOW RIBBON CARVING FOR VIDEO CONDENSATION

Huan-Yu Wu

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

Dec 11, 2009

Technical Report No. ECE-2009-03

This report is submitted in partial requirement for Master of Science Degree under the supervision of Prof. Janusz Konrad and Prof. Prakash Ishwar.

Summary

The objective of this project is to extend the current ribbon carving algorithm for video condensation to handle streaming video. Based on image seam carving, video ribbon carving is a novel approach developed at Boston University. This method allows us to shorten the length of surveillance video while permitting graceful activity loss. We adopt a sliding-window approach to process very long video sequences, in principle infinite-length ones. We have successfully tested the developed algorithm on several video sequences captured in an urban environment, including pedestrians, bikers, and motor vehicles.

Contents

List of Figures

List of Tables

1 Introduction

Surveillance cameras have been densely deployed in populous areas and are producing extremely huge video data. However, viewing the entire video to find activity is inefficient as most segments are not important. Therefore, methods are needed to manage data overload.

Several different approaches have been proposed to condense video. Uniform frame-dropping, or fast forwarding, can achieve high condensation ratio, yet activity is quite distorted. Non-uniform frame-dropping keeps all activity frames thus prevents distortion. Nevertheless, its performance is rather limited.

Ribbon carving we use here is a novel approach to condense video. Due to the limitation of memory, however, processing an extremely long video is prohibited. In this project, we use sliding-window to condense streaming video. We also apply different background subtraction methods as our costs and use different stopping criteria to examine the performance.

The report is organized as follows. In Section 2, prior works of image seam carving and video ribbon carving are explained as background materials. In Section 3, we illustrate how we apply ribbon carving to streaming video. In Section 4, we show some experimental results and analyze the data obtained. Finally, in Section 5 we come to the conclusion and propose some further works.

2 Preliminaries

The concept of 3-D video ribbon carving was inspired by 2-D image seam carving described in [1]. We will first describe the idea of image seam carving, and then extend this to video ribbon carving.

2.1 Image seam carving

The idea of image re-sizing by seam carving is removing pixels that are less relevant while preserving pixels that contain important objects. The question here is how we define the cost of removing each pixel. Apparently, objects in an image often imply that there will be relatively significant change in color at edges. Therefore, one general approach is to utilize the magnitude of derivative as out cost function:

$$C(I) = \left| \frac{\partial}{\partial x} I \right| + \left| \frac{\partial}{\partial y} I \right|, \tag{1}$$

where I is the luminance of the image. This means that when we have higher changes of luminance compared to the neighbor of a pixel, we have higher cost at the pixel. After we compute the cost of each pixel, we have a cost array whose size is the same as the image. Then we can define the cost of a seam as the summation of cost at the corresponding locations of the seam.

Next, we decide to remove which pixels by utilizing the cost array. In order to maintain the rectangular shape of an image, a straightforward way is to delete the pixels which have the minimum value of cost array in each row or each column, depending on reducing the column or row in an image respectively. However, this may destroy the image content by producing a zigzag effect (Fig. 2(e)) as the removed pixels are not connected. Therefore, the locations of the pixels to be removed have to be restricted as a connected seam. Formally, a vertical seam in an $H \times W$ image is a set of pixels $(x(y), y)$, $y = 1,\ldots,H$, s.t. $\forall y$, $|\, x(y) - x(y-1)\,| \le \phi$, where $1 \le x(y) \le W$. The flex parameter ϕ here is a non-negative integer, which controls the maximum deviation from connectivity (Fig. 1). If $\phi = 0$ then the seam coincides with a vertical line. Similarly, a horizontal seam is a set of pixels $(x, y(x))$, $x = 1,\ldots,W$, s.t. $\forall x$, $|\, y(x) - y(x-1)\,| \le \phi$, where $1 \le y(x) \le H$.

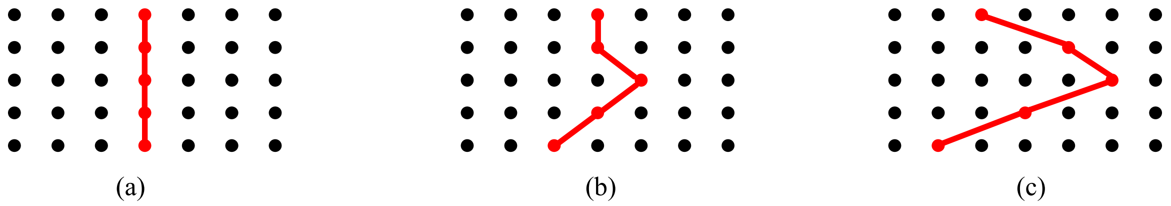


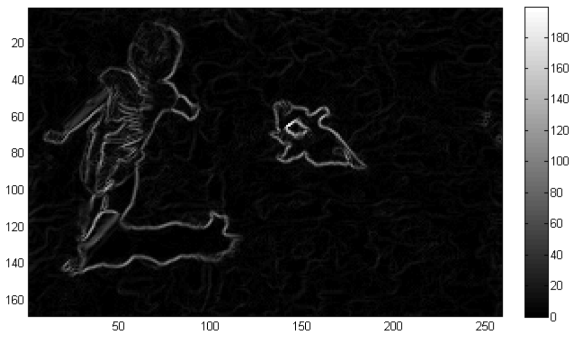
Fig. 1. Examples of vertical seams with different flex parameters: (a) $\phi = 0$, (b) $\phi = 1$, (c) $\phi = 2$

The problem now becomes how we find the seam with the minimum cost. We use dynamic programming (appendix 6.1), which guarantees that the seam with minimum cost will be discovered. To find the vertical seam with minimum cost, first we traverse the cost array from the second row to the last row and compute the cumulative minimum cost M:

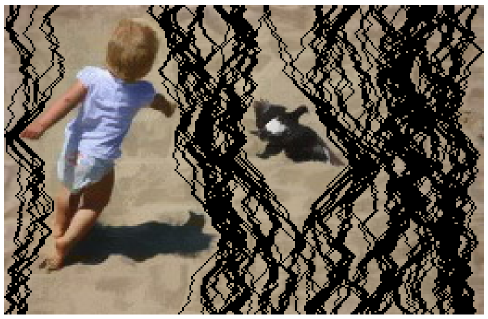
$$M(i,j) = C(i,j) + \min(M(i-1, j-\phi), M(i-1, j-\phi+1), \ldots, M(i-1, j+\phi)).$$

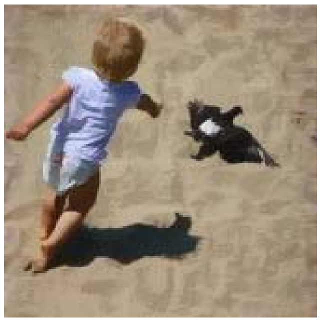
At the end of this process, the location of the minimum value of the last row in M will be the end of the vertical seam with minimum cost. Then we can trace back from this pixel to find the path of the optimal seam. The procedure for finding a horizontal seam is similar. After a seam is removed, we re-compute the costs and then find the least-cost seam again. We perform these steps recursively until some stopping criterion is met.

(a)



(b)


(c)


(d)

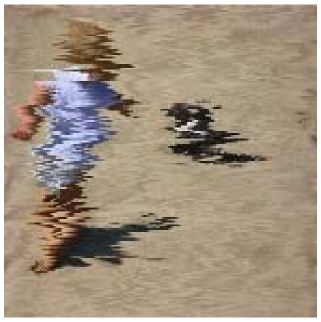

(e)

Fig. 2. Re-sizing a picture from rectangular size to square size: (a) original image, (b) corresponding costs computed using equation (1), (c) original image with seams of $\phi = 1$, (d) re-sized image using seam carving, and (e) re-sized image by removing least-cost pixels in each row.

When re-sizing an image, we have to decide the order of removing a vertical seam and a horizontal seam. If we would like to change the size of a given image from $H \times W$ to $H \times W'$ or from $H \times W$ to $H' \times W$, the choices of removing vertical seams or horizontal seams are obvious; if we retarget an image from $H \times W$ to $H' \times W'$, however, one approach to achieve the global minimum cost is to build a transport map indicating the cost of the optimal sequence of vertical and horizontal seam removal operations as well as a 1-bit map indicating which of the two options was chosen [1]. Nevertheless, this is time-consuming and impractical for further use of video condensation. Thus the alternative approach we utilize is to use greedy algorithm, which makes the locally optimal choice at each stage.

2.2 Video ribbon carving

In video condensation, we would like to reduce the length of video while preserving the activities that appear in video. Thus we extend the idea of a seam to a ribbon; the approach of video condensation becomes removing the ribbon with minimum cost.

Suppose that we have a segment of video with N frames, where the size of each frame is W pixels wide and H pixels tall. We define a ribbon as a connected surface that for each (x, y) position, where $x = 1,\dots,W$, $y = 1,\dots,H$, there is exactly one corresponding pixel in time domain. In particular, a ribbon is a set of pixels $(x, y, t(x, y))$, where $t(x, y)$ is a function with range $1,\dots,N$, and $| t(x, y) - t(x', y') | \leq \phi$ for all (x, y) and (x', y') for which $| x - x' | \leq 1$ and $| y - y' | \leq 1$. The flex parameter ϕ is again a non-negative integer, which controls the maximum deviation from connectivity. If $\phi = 0$ then the ribbon coincides with a frame.

Next we further define two types of ribbon: a vertical ribbon and a horizontal ribbon (Fig. 3). A vertical ribbon is a set of pixels $(x, y, t(y))$ in which $t(y)$ do not depend on x. As a result, the vertical ribbon becomes a curve line when we view it from side. Similarly, a horizontal ribbon is a set of pixels $(x, y, t(x))$ in which $t(x)$ do not depend on y. As a result, the horizontal ribbon becomes a curve line when we view it from top. This formalization is useful to transform the 3-D problem into 2-D problem.

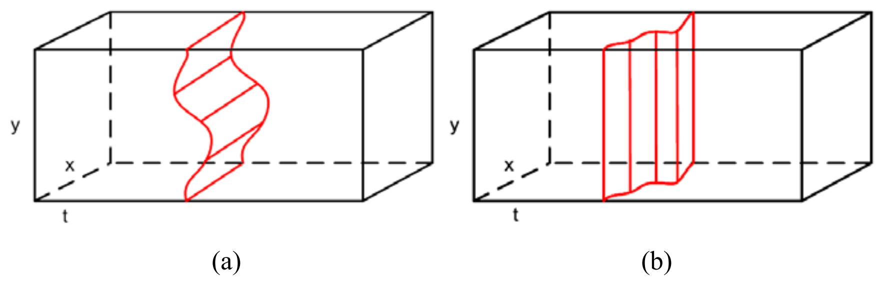


Fig. 3. Illustration of (a) vertical ribbon, and (b) horizontal ribbon in video cubes.

To delete a ribbon from a video segment, we need to find the ribbon with minimum cost. We define the cost of a ribbon as the summation of cost at the individual pixels that constitute the ribbon, that is,

$$C(R) = \sum_{(x,y,t)\in R} C(x,y,t),$$

where R denotes a ribbon. Therefore, from a given video, we can find both least-cost vertical and horizontal ribbons. Then we compare the minimum cost and remove the ribbon which has smaller cost. We recursively carve out the ribbon with minimum cost until some stopping criterion is met (sec. 3.2). Note that here we do not need to re-compute the cost after every ribbon deletion because the cost of motion-related activity will be preserved in the condensed video [2]. After condensation, we have N' ($\le N$) frames, where the size of each frame remains the same.

3 Condensation for streaming video

Ribbon carving allows us to condense a segment of video at a time. However, when we need to condense a video with very long sequence, there are two main reasons why we cannot read all video frames simultaneously to process:

1) Computation time of finding a least-cost ribbon will be extremely long,

2) Due to limitation of memory, reading in whole sequence of video is prohibited.

Therefore, we utilize sliding-window to solve these problems.

Our whole algorithm is discussed in this section, including background subtraction, stopping criteria, and sliding-window approach.

3.1 Background subtraction

In order to preserve the motion of objects, the costs should indicate motion-related activity reliably. Here we use background subtraction to obtain binary costs, where 0 denoting background and 1 denoting foreground.

3.1.1 Simple background subtraction

One simple method is comparing the current frame and the background model. If the absolute value of difference is greater than some threshold *θ*, we assume that there is motion at the pixel and thus the cost is assigned to 1; otherwise the cost is assigned to 0:

$$C(x,y,t)=\begin{cases}1 & if\ \frac{1}{3}\begin{pmatrix}\left|I_r(x,y,t)-B_r(x,y,t-1)\right|+\\ \left|I_g(x,y,t)-B_g(x,y,t-1)\right|+\\ \left|I_b(x,y,t)-B_b(x,y,t-1)\right|\end{pmatrix}>\theta\\ 0 & otherwise\end{cases}.$$

where *I* is a video frame with subscript *r*, *g*, and *b* meaning red, green, and blue values, respectively. The background model *B* is recursively updated at each frame. For initialization, we pick *n* frames from the first (1 + *nm*) frames of video to compute the background using temporal median filter:

$$B_r(x,y,0)=\text{median}\{I_r(x,y,1+m),I_r(x,y,1+2m),...,I_r(x,y,1+nm)\}.$$

In our experiments we use $n = 20$ and $m = 10$. The same operation is applied to green and blue color of background. For the following frames, the background is equivalent to the linear combination of previous background and the current frame:

$$B(x,y,t)=(1-\alpha)B(x,y,t-1)+\alpha I(x,y,t),$$

where *α* is a small value between zero and one. The whole process can be done from the first frame to the last frame. At the end, we have a 3-D binary cost array.

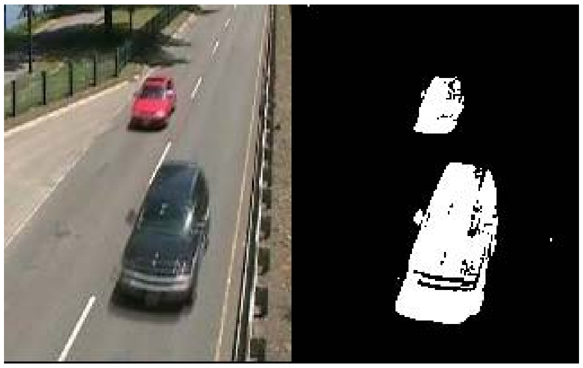


Fig. 4. Activity-based binary labels (due to object motion): (a) video frame, and (b) activity frame (by Simple background subtraction with $\theta = 30$, $\alpha = 0.001$)

3.1.2 Adaptive background subtraction

In our experiments, we utilize the algorithm developed in [3] with foreground model disabled and Markov random field enabled. Since the notations used in the results of this report are the same, we refer the reader to this paper.

3.2 Stopping criteria

After the cost is computed, we can now start to find the least-cost ribbon. We recursively carve out the ribbon until some stopping criterion is met. In image re-sizing, we can specify the new size of image and then do seam-carving recursively until we have the image in specified size. In video ribbon carving, however, specifying new length (number of frames) is not appropriate because some video segments may have more activities while others have less. Therefore, instead of specifying new length, we recursively remove least-cost ribbons until there are no more ribbons with cost less than ε. Here ε is a user-defined small value compared to the total number of pixels in a ribbon (or equivalently in a frame).

The selection of ε is related to the following factors:

- False positives: The color of background may not be stable all the time. For example, change of sunlight, water surface or video noise may cause some false positives. A non-zero ε allows the least-cost ribbon pass through some false positives.

- False negatives: There may be some misses in moving objects after imperfect background subtraction. Therefore, setting a high value of ε may lead a least-cost ribbon to pass through these objects.
- The size of moving objects: If the moving objects which we are interested in are small, ε should be restricted in lower value (say at least ten times lower than the number of pixels covered by objects), or the moving objects will be carved out by ribbons thus some visible distortions will happen on the objects. Otherwise, if the moving objects which we are interested in are large, ε can be released to a higher value. In addition, if there are some unimportant moving objects that are relatively small (small birds, distant objects, etc.) compared to the frame size, slightly increasing ε will let ribbons carve out the small objects and hence increase condensation ratio.

If ε increases, the condensation ratio will also increase as we allow ribbons with higher cost to be removed. However, some visible distortions of moving objects start to appear when ε is greater than some value. When we increase ε further, distortions of moving objects become more serious (Fig. 11).

3.3 Sliding-window approach

3.3.1 Maximum extension of a ribbon

Due to the fact that a ribbon is made up of connected pixels, the number of frames that a ribbon can cover is limited. This depends on width W, height H, and flex parameter ϕ. For a vertical ribbon, the maximum extension is $(\phi H - \phi + 1)$ frames; for a horizontal ribbon, the maximum extension is $(\phi W - \phi + 1)$ frames (Fig. 5). Therefore, a ribbon with flex parameter ϕ cannot span more than $M_\phi \equiv (\max(\phi H - \phi + 1, \phi W - \phi + 1)) = (\phi\max(H, W) - \phi + 1)$ frames.

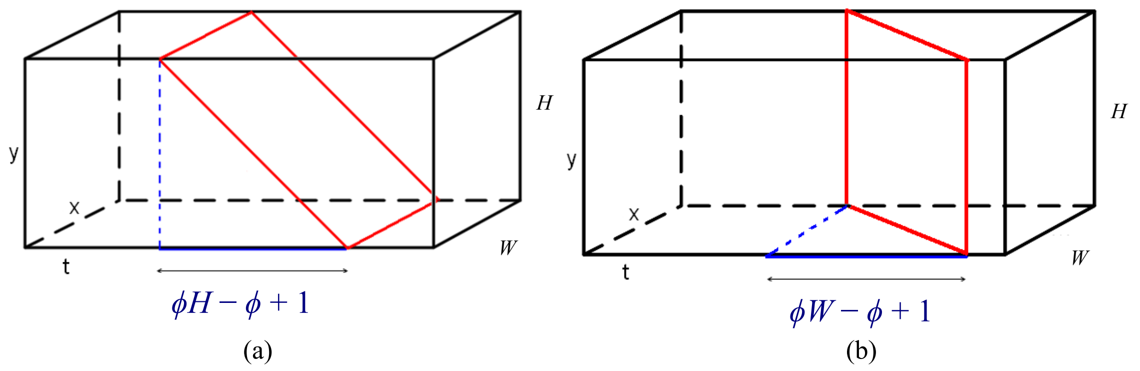


Fig. 5. Maximum extension of (a) a vertical ribbon, and (b) a horizontal ribbon along time axis

3.3.2 Processing streaming video

To apply ribbon carving algorithm to streaming video efficiently and overcome the limitation of memory, we adopt a sliding-window approach. The basic idea is that we use a sliding-window buffer to process a segment of video at a time before it can save some frames if appropriate and read in new frames to process. This way we do not need to process the whole video at a time.

The observation described in Section 3.3.1 is useful to implement sliding-window ribbon carving. Suppose that we have a sliding-window buffer filled with N frames. After ribbon carving, we have N' frames. Then we compare N' with M_ϕ. If $N' > M_\phi$, then we save the first $N' - M_\phi$ frames, push the remaining M_ϕ frames to the front of the buffer, and read in new frames from the video to fill up the buffer. The reason why we can save the first $N' - M_\phi$ frames is that no ribbon from the new frames beyond N' can reach this set of frames. Ribbons passing through the first $N' - M_\phi$ frames will all have cost greater than ε. Thus saving these frames will not affect the result of finding the least-cost ribbon. If $N' \leq M_\phi$, then no frames can be saved. Hence in this case we simply read in new frames from the video to fill up the buffer. After the buffer is filled with N frames, we again do ribbon carving until the stopping criterion is met. The above procedure is implemented until the end of video.

Since greater ϕ may potentially introduce event anachronism [2], we start with $\phi = 0$ and increase it by one each time until $\phi_{\max}$. The overall procedure can be described as follows:

1) Given a video, compute the costs.
2) Do ribbon carving for $\phi = 0$. After this step we have a condensed video.
3) Increase ϕ by one. Use sliding-window to do ribbon carving until end of video.
4) If $\phi < \phi_{\max}$, go to step 3. Otherwise if $\phi = \phi_{\max}$, then we stop.

The flowchart of the overall procedure is illustrated in Fig. 6. While implementing, some details should be noticed:

- Video and cost array should be carved at the same time since we find least-cost ribbon in cost and deal with the video at the corresponding position. When we carve out a ribbon from video, the ribbon of the same position should be carved out in the corresponding cost array as well. When using sliding-window, we should assign the same size of buffer to both video and cost array.
- When we do ribbon carving for $\phi = 0$, we simply check the cost of each frame and save the frames with cost greater than ε. We do this frame by frame thus no sliding window procedure is needed.
- Theoretically, the buffer length N can be any number greater than M_ϕ. Here we choose $N = 2M_\phi$ for implementation efficiency.
- In the ribbon carving algorithm, we compare minimum cost of vertical ribbon C_v and minimum cost of horizontal ribbon C_h and then remove the ribbon with lower cost. However, C_v may be equal to C_h. Feasible ways can be choosing between them randomly or by default. In our implementation we choose between them randomly.
- When we come to the end of video, we may not be able to fill the buffer with N frames. Therefore the input of ribbon carving algorithm may not necessarily be N frames only for this case.

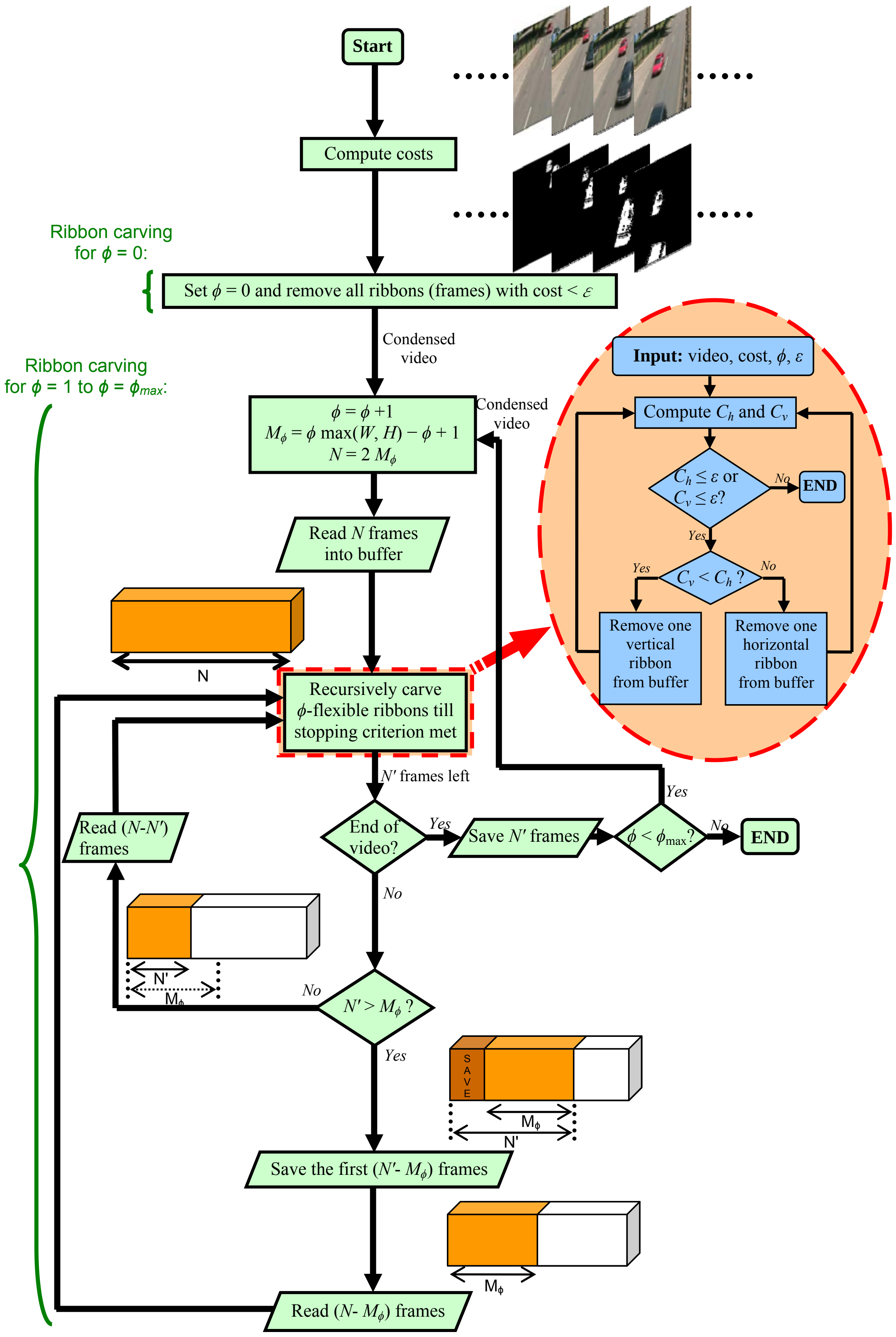

Start
Compute costs
Ribbon carving for $\phi = 0$:
Set $\phi = 0$ and remove all ribbons (frames) with cost < ε
Condensed video
Ribbon carving for $\phi = 1$ to $\phi = \phi_{max}$:
$\phi = \phi + 1$
$M_\phi = \phi \max(W, H) - \phi + 1$
$N = 2 M_\phi$
Condensed video
Read N frames into buffer
N
Recursively carve ϕ-flexible ribbons till stopping criterion met
N' frames left
End of video?
Yes
Save N' frames
$\phi < \phi_{max}$?
Yes
No
END
No
Read (N-N') frames
N'
M_ϕ
No
$N' > M_\phi$?
Yes
SAVE
M_ϕ
N'
Save the first (N'- M_ϕ) frames
M_ϕ
Read (N- M_ϕ) frames
Input: video, cost, ϕ, ε
Compute C_h and C_v
$C_h \le \varepsilon$ or $C_v \le \varepsilon$?
No
END
Yes
$C_v < C_h$?
Yes
No
Remove one vertical ribbon from buffer
Remove one horizontal ribbon from buffer

Fig. 6. Flowchart of ribbon carving video condensation using sliding-window procedure

4 Experimental results

The video condensation algorithm developed in this project has been tested on four video sequences captured on Boston University campus. The parameters and condensation ratios are listed in Table 1. Two different background subtraction algorithms described in Sec. 3.1 are used.

Sequence (30 frames per second)	**Parkway** ($W = 160$, $H = 200$, $\varepsilon/WH = 0.1\%$)		**Plaza** ($W = 270$, $H = 162$, $\varepsilon/WH = 0$)		**Sidewalk** ($W = 240$, $H = 208$, $\varepsilon/WH = 0$)		**Highway** ($W = 320$, $H = 80$, $\varepsilon/WH = 0$)	
Background subtraction algorithm	"Simple" $(\theta, \alpha) = (30, 0.001)$		"Adaptive" $(N, \theta, 1/\gamma, \sigma^2) =$ (100, 0.3, 10, 3)		"Adaptive" $(N, \theta, 1/\gamma, \sigma^2) =$ (50, 1, 10, 20)		"Adaptive" $(N, \theta, 1/\gamma, \sigma^2) =$ (50, 0.3, 10, 3)	
Length (number of frames) & Cumulative condensation ratio (length of original video / length of condensed video)								
Original video	7,000	1:1	9,800	1:1	7,950	1:1	24,000	1:1
Condensed video ($\phi = 0$)	6,402	1.09:1	5,163	1.90:1	3,570	2.23:1	7,819	3.07:1
Condensed video ($\phi = 1$)	4,172	1.68:1	3,788	2.59:1	2,888	2.75:1	2,843	8.44:1
Condensed video ($\phi = 2$)	3,746	1.87:1	3,426	2.86:1	2,888	2.75:1	2,805	8.56:1
Condensed video ($\phi = 3$)	3,671	1.91:1	3,312	2.96:1	2,888	2.75:1	2,792	8.60:1

Table 1. Condensation ratios for the four different video sequences

Table 1 shows the length of the condensed videos without visible distortion after completion of different flex parameters. First, we can see that the high condensation ratios happen in the Highway video after $\phi = 0$. This is because the objects are moving in the same direction and the speeds of them are similar. This allows us to find many ribbons with low cost in gaps of moving objects. Second, the reduced frames tend to decrease when ϕ increases and become marginal when ϕ goes from two to three. One exception is in Parkway video: the reduced frames for $\phi = 1$ are more than those for $\phi = 0$. This is because the frames that do not contain moving objects are very few, hence ribbon carving for $\phi = 0$ can only remove very few frames. Therefore, we can expect that after $\phi = 3$ the condensation ratio will not improve much. However, if a video segment contains moving objects that have the following properties: a) moving very fast (say more than four pixels/frame), b) moving unidirectionally, and c) densely distributed in frames, we can expect that the condensation ratio will further improve much after $\phi = 3$ (Fig. 7).

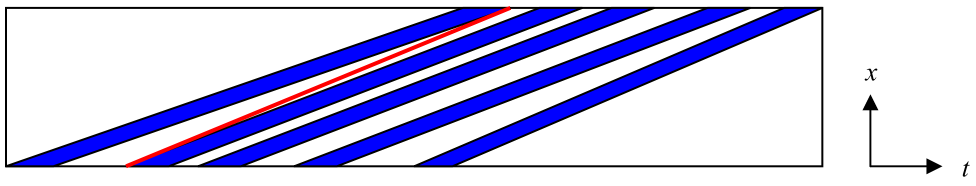


Fig. 7. Illustration of video condensation involving fast moving objects. Object tunnels (blue) and a possible ribbon (red) here show that only ribbons with high flex parameter can further condense the video segment. Note that the x axis can also change to y axis.

Sequence	**Parkway**	**Plaza**	**Sidewalk**	**Highway**
Vertical ribbons ($\phi = 1$)	1,031	484	60	572
Horizontal ribbons ($\phi = 1$)	1,199	891	622	4,404
Vertical ribbons ($\phi = 2$)	1,213	653	60	575
Horizontal ribbons ($\phi = 2$)	1,443	1,084	622	4,439
Vertical ribbons ($\phi = 3$)	1,245	666	60	578
Horizontal ribbons ($\phi = 3$)	1,486	1,185	622	4,449

Table 2. Cumulative number of carved-out ribbons in the four sequences of Table 1.

In Table 2 we count the two different ribbons carved out in these four sequences. In Sidewalk and Highway videos, objects predominantly have horizontal motion thus the carved-out horizontal ribbons are much more than vertical ribbons. In Parkway and Plaza videos, numbers of vertical and horizontal ribbons are much balanced since objects tend to move diagonally. In Plaza video, however, some objects have simply horizontal motion while no objects have simply vertical motion, thus horizontal ribbons somewhat carve out more. Fig. 12 further shows the number of vertical and horizontal ribbons in Parkway video regarding different stopping criteria *ε*.

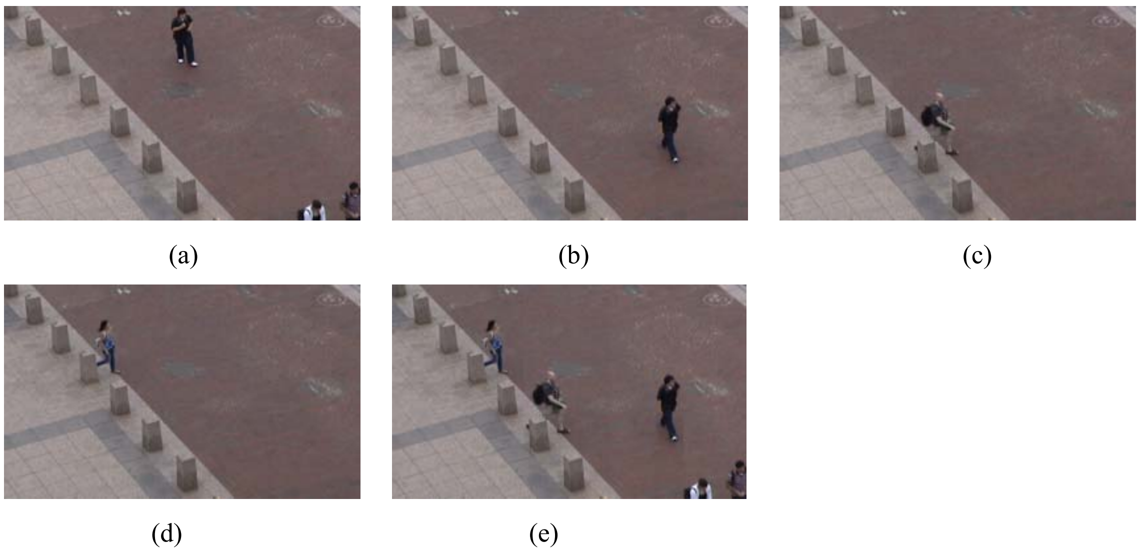


Fig. 8. Sample frames from Plaza video. (a) Original frame #7302, (b) Original frame #7417, (c) Original frame #7871, (d) Original frame #8655, (e) Condensed frame #2890.

Plaza is an interesting video, showing pedestrians coming from different positions of video and moving in different directions. Result shows that our algorithm works well since the video is condensed while all important activities are preserved. Fig. 8 shows that distance between pedestrians becomes closer. The pedestrian wearing white shoes and the two people in front are moving in the same direction. Their distance can be shortened by removing horizontal ribbons and vertical ribbons (Fig. 9).

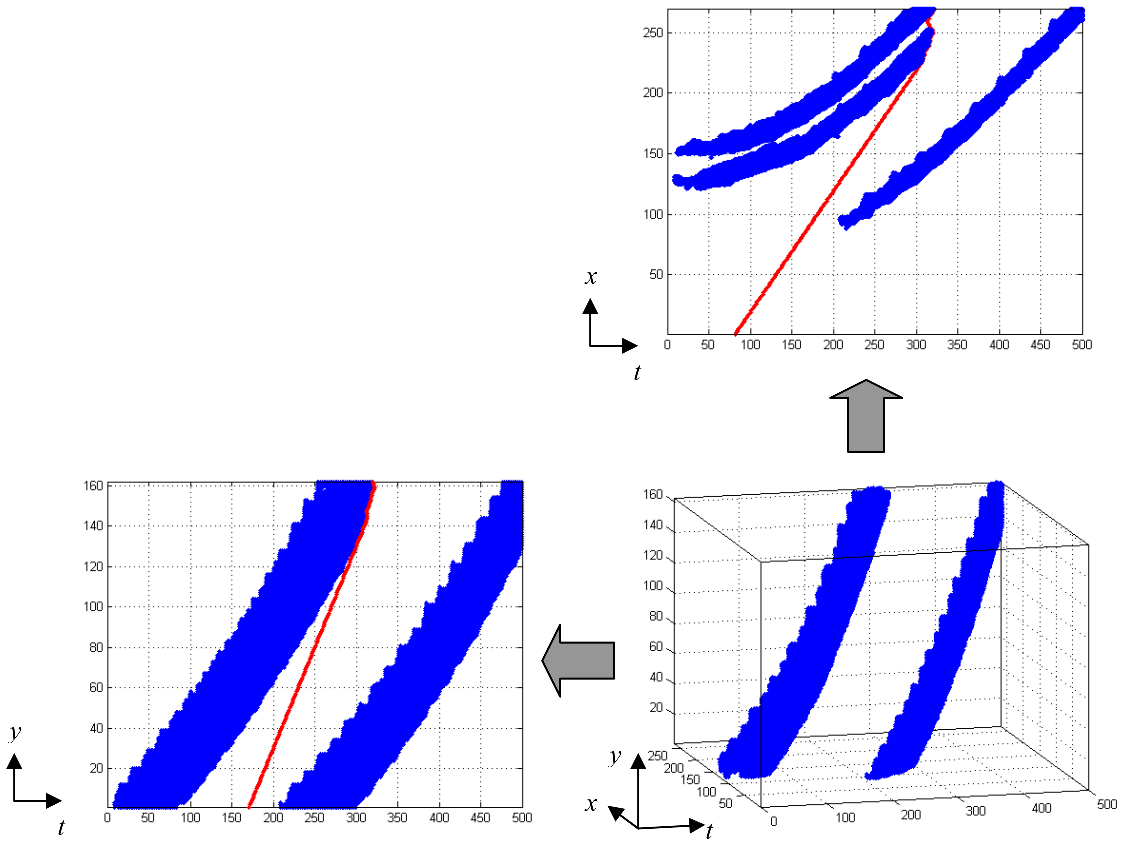


Fig. 9. Object tunnels (blue) and least-cost ribbons of $\phi = 1$ (red) obtained from the Plaza sequence: perspective view (bottom right), top view (top) and side view (left). Horizontal ribbons are perceived in top view while vertical ribbons are perceived in side view. This figure shows that video with objects moving in the same direction, if not horizontal or vertical, can be condensed by removing horizontal and vertical ribbons.

Fig. 11 shows that how stopping criterion ε affects the performance in Parkway video. When ε increases, the number of carved-out ribbons also increases hence the condensation ratio becomes higher. For $\varepsilon = 0$, few ribbons with cost = 0 can be found since moderate false positives are allowed here in Simple background subtraction. When ε increases a little, the number of carved-out ribbon can significantly increase. We find the highest condensation ratio with no visible distortion when $\varepsilon/(WH) = 0.1\%$. However, some visible distortions start to occur from $\varepsilon/(WH) = 0.15\%$. Distortions become more and more noticeable and when $\varepsilon/(WH)$ comes to 1% the moving objects are severely distorted.

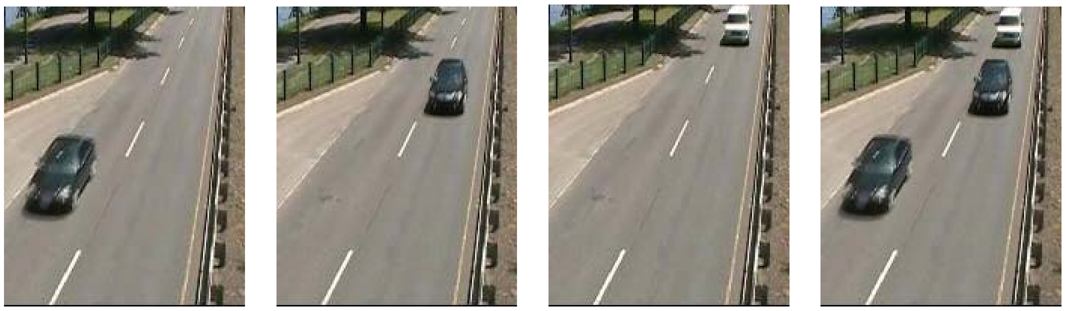


Fig. 10. Sample frames from Parkway video. (a) Original frame #182, (b) Original frame #255, (c) Original frame #344, (d) Condensed frame #124.

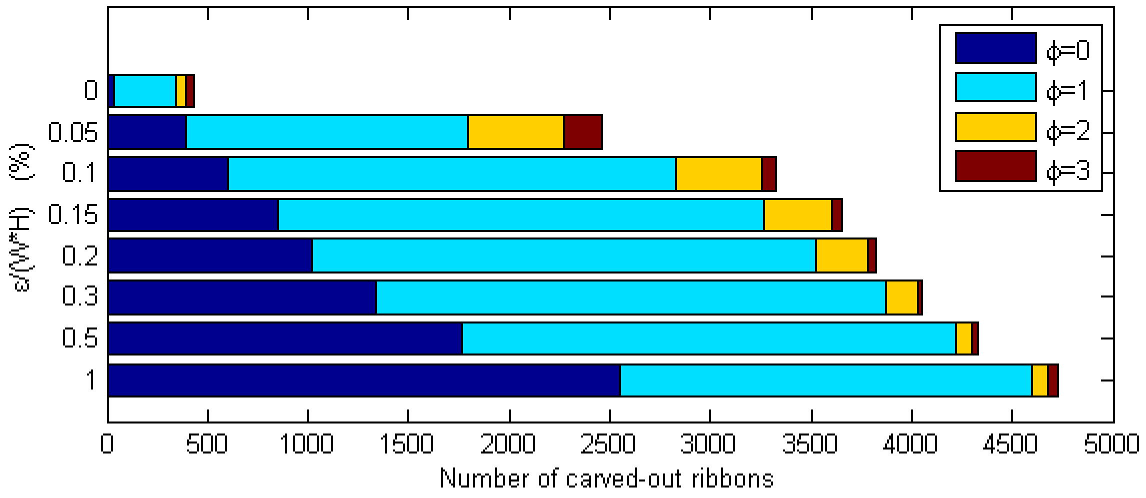


Fig. 11. Number of carved-out ribbons in Parkway video using different values of stopping criterion ε.

Fig. 12 further shows which type of ribbons are removed in this case. We can see that before visible distortion occurs, horizontal ribbons tend to carve out more while vertical ribbons tend to be removed more after visible distortion occurs. Fig. 13 shows the summation of costs projected to side and top of the cube. In our algorithm, we use these two matrices to find the horizontal and vertical least-cost ribbon respectively. Thus the number of ribbons we can remove depends on how many ribbons with cost < ε we can find in the two matrices. First, we can see that in top view (Fig. 13 (a)) the cars do not move across the whole x axis. Some cars only move in a segment along x axis. This provides free paths in some upper and lower parts of this matrix for the least-cost ribbon. Thus, when ε is set to low values, more ribbons can be found in this matrix. Second, the values in tunnels are lower in side view (Fig. 13 (b)) than that in top view. Therefore, when ε is set to high values, ribbons start to pass through the tunnels in the side view matrix and then some visible distortions occur as a consequence.

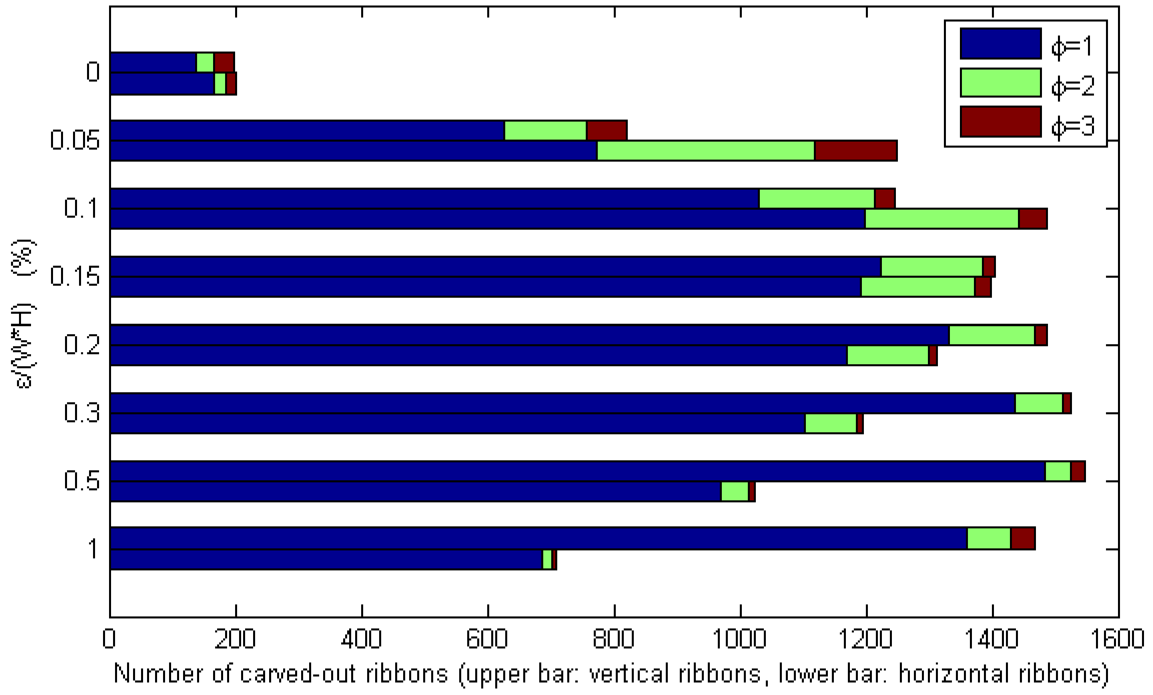


Fig. 12. Number of carved-out vertical and horizontal ribbons counted from $\phi = 1$ in Parkway video using different values of stopping criterion ε.

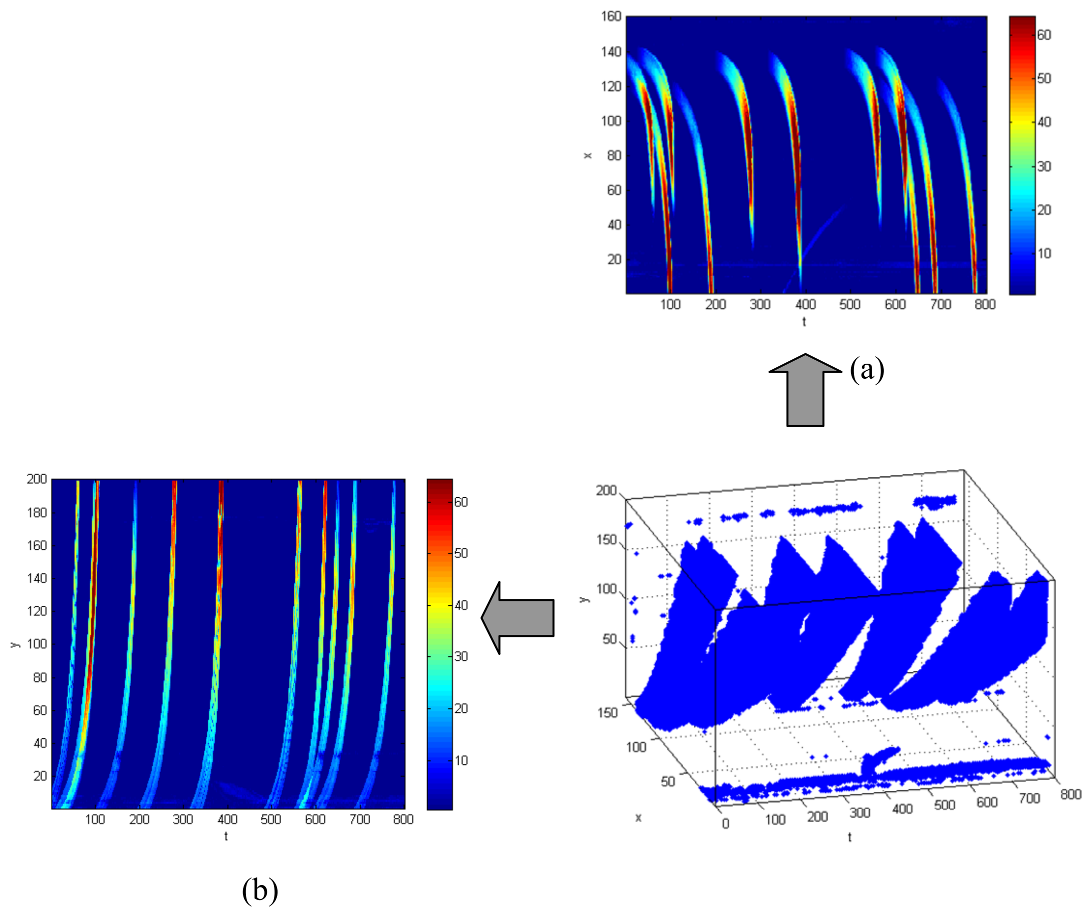
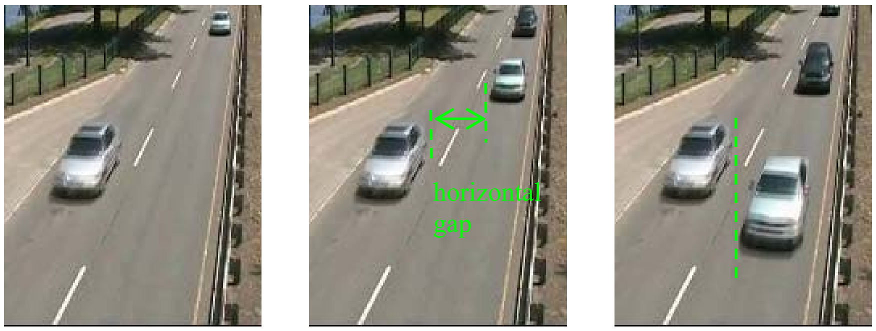


Fig. 13. Projection of costs from the first 800 frames of Parkway video viewed from (a) top and (b) side of the cube. Higher values indicate tunnel of moving objects. The ribbons have to go from the first row to the last row in both matrices. Note that the cars move not only vertically but also horizontally at the same time.

Fig. 14 shows little event anachronism on some sample frames from the original video and condensed videos. The car in the right lane is originally behind the car in the left lane. However, in condensed video the car in the right lane comes to the front of the car in the left lane when ϕ comes to two. This is because there is still some space that horizontal ribbons can fit in between these two moving objects after $\phi = 1$ ribbon carving is done. Although little event anachronism occurs here, no visible distortions of objects are introduced.



(a) (b) (c)

Fig. 14. Illustration of little event anachronism in Parkway video ($\varepsilon/(WH) = 0.1\%$): (a) Original frame #6898, (b) Condensed ($\phi = 1$) frame #4113, (c) Condensed ($\phi = 2$) frame #3710.

Since the condensation ratio depends not only on how objects move in video but also on the performance of background subtraction, here we discuss the parameters selection in both used algorithm. The Simple background subtraction compares the current frame and the background, so reducing θ will have less false negatives while introducing more false positives. The learning parameter α controls the learning rate of background. When objects move slowly, lower value is desired as this will not contaminate the background. Otherwise, higher value is feasible because it will learn background quickly thus preventing false positives caused by change of sunlight. Note that when using this background subtraction algorithm, some scattered false positives are unavoidable thus usually we allow ε to be nonzero when doing ribbon carving.

The Adaptive background subtraction has more parameters to deal with. Usually using $N = 50$ to compute background model is enough. When there are moving objects staying in the same position for a period of time, using higher value can prevent background from being contaminated by the objects. Both the threshold θ and the variance σ^2 of the Gaussian kernel control number of false positives. Higher θ and/or lower σ^2 lead to more false positives. The penalty $1/\gamma$ in Markov random field (MRF) varies the threshold. Higher value of $1/\gamma$ can strengthen the influence of MRF model. This algorithm utilizes MRF thus preventing scattered false positives, hence usually we use $\varepsilon = 0$ as our stopping criterion.

5 Conclusion and future work

Sliding-window ribbon carving has been implemented successfully for video condensation. We have demonstrated that the algorithm can condense a streaming video with static background while preserving major activities. Condensation ratios of 1.91:1 to 8.60:1 are achieved in our experiments.

So far the algorithm has been developed in MATLAB and the computation time will be an issue when dealing with an extremely long sequence. Therefore, it would be interesting to implement this algorithm in C/C++ or even on multi-core CPUs or GPUs so that real-time application can be realized. In addition, exploring different types of ribbon (e.g., directional) may further increase condensation ratio as the shape of ribbon becomes more flexible. These are potential directions for future research.

Acknowledgment

We would like to thank Dr. P.-M. Jodoin of the University of Sherbrooke, Canada, for providing the idea and MATLAB code of Simple background subtraction. We thank Mr. Kai Guo of Boston University for providing MATLAB code and helpful comments of Adaptive background subtraction. We also thank Mr. Chia-Chiunn Ho for providing his photo work so that the idea of image seam carving can be clearly exhibited.

6 Appendix

6.1 Dynamic programming

Example: Cost array *e*

1	9	12
1	5	4
12	13	0

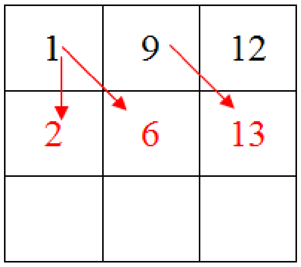
Steps (top-down):

1. Copy the first row from the cost matrix *e* to matrix *M*:

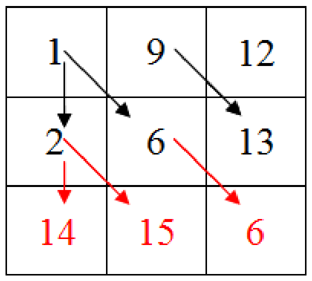
1	9	12

2. For the second row, compute the cumulative minimum cost *M* for all possible connected seams for each entry (*i, j*) and record the direction:

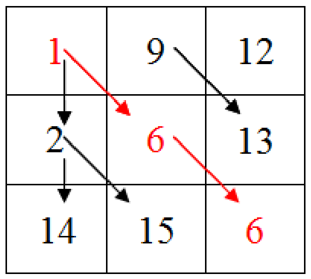
$M(i,j) = e(i,j) + \min(M(i-1,j-1), M(i-1,j), M(i-1,j+1))$



3. Similar procedure to step 2 until the entries in the last row are computed.



4. Choose the minimum value of the last row and backtrack from this entry along the recorded direction.



6.2 MATLAB code

Below is listed Matlab source code developed for this project.

```
%sliding window script
%input file:'whole_video.avi'
%output files (number depends on flexmax):
%      'whole_cost.avi',
%      'flex0_video.avi','flex0_cost.avi',
%      'flex1_video.avi','flex1_cost.avi',
%      'flex2_video.avi','flex2_cost.avi',
%      'flex3_video.avi','flex3_cost.avi',(...)

%Huan-Yu Wu
%

eps=0;                          %stopping criterion
flexmax=3;
colormap=zeros(2,3);              %set up colormap for output cost file
colormap(2,:)=ones(1,3);
```

```
Xinfo=aviinfo('whole_video.avi');
H=Xinfo.Height;
W=Xinfo.Width;



%compute Costs first
%input file: whole_video.avi (with RGB values)
%output file: whole_cost.avi
tic
N=50;
C_avi=avifile('whole_cost.avi','colormap',colormap,'compression','None'
  ,'fps',30);
%compute the cost array here
%set the first N frames to all ones
for frame=1:N
   C_avi=addframe(C_avi,uint8(true(H,W)));
end
[C_avi it]=BkgSub_avioutput('whole_video.avi',C_avi,Xinfo.NumFrames,
  N,0.3,10,3,1);     %(seq,output,T,N,th,pen,var,ratio)
C_avi=close(C_avi);
toc



%flex==0
%input file: whole_video.avi, whole_cost.avi
%output file: flex0_video.avi, flex0_cost.avi
h=waitbar(0,'Computing flex=0...');
tic
X_avi=avifile('flex0_video.avi','compression','None','fps',30);
C_avi=avifile('flex0_cost.avi','colormap',colormap,'compression','None'
  ,'fps',30);
for frame=1:Xinfo.NumFrames
   waitbar(frame/Xinfo.NumFrames)
   X=aviread('whole_video.avi',frame);
   X=X.cdata;
   C=aviread('whole_cost.avi',frame);
   C=C.cdata;
   %check the cost of each frame
   if sum(sum(C))>eps
      X_avi=addframe(X_avi,X);
      C_avi=addframe(C_avi,C);
   end
end
X_avi=close(X_avi);
C_avi=close(C_avi);
toc
close(h);

%flex>=1
%input file: flex0_video.avi, flex0_cost.avi
%output file: flex1_video.avi, flex1_cost.avi, etc.

%the following four vectors are for statistics only
vertRibbonCount=zeros(flexmax,1);
horRibbonCount=zeros(flexmax,1);
way1Count=zeros(flexmax,1);
```

```
way2Count=zeros(flexmax,1);

for flex=1:flexmax
    h=waitbar(0,['Computing flex=',num2str(flex),'...']);
    tic
    X_avi=avifile(['flex',num2str(flex),'_video.avi'],'compression',
      'None','fps',30);
    C_avi=avifile(['flex',num2str(flex),'_cost.avi'],'colormap',colormap,
      'compression','None','fps',30);
    %initialization
    Mphi=flex*max(W,H)-flex+1;
    N=fix(2*Mphi);
    startframe=1;      %start reading from this frame
    endframe=N;        %stop reading until this frame
    bufferlength=0;    %the current length of the buffer
    Xinfo=aviinfo(['flex',num2str(flex-1),'_video.avi']);

    while startframe<=Xinfo.NumFrames
        waitbar(startframe/Xinfo.NumFrames)
        if endframe<Xinfo.NumFrames
            tempX=aviread(['flex',num2str(flex-1),'_video.avi'],
              startframe:endframe);
            X(:,:,:,bufferlength+1:N)=cat(4,tempX.cdata);
            clear tempX
            tempC=aviread(['flex',num2str(flex-1),'_cost.avi'],
              startframe:endframe);
            C(:,:,bufferlength+1:N)=cat(3,tempC.cdata);
            clear tempC
            [X C vertRibbonCount(flex) horRibbonCount(flex)]=
              ribboncarvemain(X,C,flex,eps,vertRibbonCount(flex),
              horRibbonCount(flex));      %do ribbon carving

            Np=size(X,4);
            if Np > Mphi
                Npp=Np-Mphi;
                %save the first Npp frames
                for frame=1:Npp
                    X_avi=addframe(X_avi,X(:,:,:,frame));
                    C_avi=addframe(C_avi,C(:,:,frame));
                end
                %push the remaining output to the front
                X(:,:,:,1:Mphi)=X(:,:,:,Npp+1:Np);
                C(:,:,1:Mphi)=C(:,:,Npp+1:Np);
                bufferlength=Mphi;
                %set up the next reading frames
                startframe=endframe+1;
                endframe=startframe+N-Mphi-1;
                way1Count(flex)=way1Count(flex)+1;
            else
                %no frame is saved
                bufferlength=Np;
                startframe=endframe+1;
                endframe=startframe+N-Np-1;
                way2Count(flex)=way2Count(flex)+1;
            end
        else
            %processing the last video chunck
            tempX=aviread(['flex',num2str(flex-1),'_video.avi'],
              startframe:Xinfo.NumFrames);
            X(:,:,:,bufferlength+1:bufferlength+Xinfo.NumFrames-
              startframe+1)=cat(4,tempX.cdata);
            clear tempX
            X=X(:,:,:,1:bufferlength+Xinfo.NumFrames-startframe+1);
            tempC=aviread(['flex',num2str(flex-1),'_cost.avi'],
              startframe:Xinfo.NumFrames);
            C(:,:,bufferlength+1:bufferlength+Xinfo.NumFrames-
              startframe+1)=cat(3,tempC.cdata);
            clear tempC
            C=C(:,:,1:bufferlength+Xinfo.NumFrames-startframe+1);
            [X C vertRibbonCount(flex) horRibbonCount(flex)]=
              ribboncarvemain(X,C,flex,eps,vertRibbonCount(flex),
              horRibbonCount(flex));

            Np=size(X,4);
            %save the entire buffer
            for frame=1:Np
                X_avi=addframe(X_avi,X(:,:,:,frame));
                C_avi=addframe(C_avi,C(:,:,frame));
            end
            clear X C
            break
        end
    end
    %the avi files of video and cost are eventually produced here
    X_avi=close(X_avi);
    C_avi=close(C_avi);
    toc
    close(h);
end
```

```
function [X C vertRibbonCount horRibbonCount]=ribboncarvemain(X,C,flex,eps,
  vertRibbonCount,horRibbonCount)
%The main function for ribbon carving
%
%input:
%X: input video (4-D array)
%C: input cost (3-D array)
%flex: flex parameter
%eps: stopping criterion
%
%output:
%X: output video (4-D array)
%C: output cost (3-D array)
%
%the number of carved-out vertical ribbons in this function will be
%vertRibbonCount(output) - vertRibbonCount(input).
%the number of carved-out horizontal ribbons in this function will be
%horRibbonCount(output) - horRibbonCount(input).
```

```
[H W rgb N]=size(X);

My=cummincost(C,flex,'vert');   %compute the cumulative minimum cost
Cv=min(My(H,:,1));              %the cost of removing a vertical ribbon
Mx=cummincost(C,flex,'hor');
Ch=min(Mx(W,:,1));              %the cost of removing a horizontal ribbon
Ch
Cv
while Ch<=eps || Cv<=eps
   %remove a vertical ribbon
   if Cv < Ch
      [X C]=carve(X,C,My,'vert');
      vertRibbonCount=vertRibbonCount+1;

   %remove a horizontal ribbon
   elseif Ch < Cv
      [X C]=carve(X,C,Mx,'hor');
      horRibbonCount=horRibbonCount+1;

   %when the cost of removing a vertical and horizontal ribbon are equal
   else
      if rand < .5
         [X C]=carve(X,C,My,'vert');
         vertRibbonCount=vertRibbonCount+1;
      else
         [X C]=carve(X,C,Mx,'hor');
         horRibbonCount=horRibbonCount+1;
      end
   end
   My=cummincost(C,flex,'vert');  %compute the cumulative minimum cost
   Cv=min(My(H,:,1));             %the cost of removing a vertical ribbon
   Mx=cummincost(C,flex,'hor');
   Ch=min(Mx(W,:,1));             %the cost of removing a horizontal ribbon
   Ch
   Cv
end
```

```
function M=cummincost(C,flex,direction)
%compute the cumulative minimum cost for all possible connected ribbons
%(using dynamic programming)
%
%input:
%C: cost array
%flex: flex parameter (non-negative integer)
%direction: 'vert': remove vertical ribbon
%           'hor': remove horizontal ribbon
%
%output:
%M(:,:,1): cumulative minimum cost (2-D array)
%M(:,:,2): direction of backtracking (2-D array)

[H W N]=size(C);

switch direction
```

```
    case 'vert'
        M=zeros([H,N,2],'single');
        %sum up the cost along the 2nd dimension, and then rearrange it to
        %2-D array
        sumC=permute(sum(C,2),[1 3 2]);
        M(1,:,1)=sumC(1,:);
        for i=2:H
            for j=flex+1:N-flex
                [a b]=min(M(i-1,j-flex:j+flex,1));
                M(i,j,1)=sumC(i,j)+a;
                M(i,j,2)=b-flex-1;
            end

            %boundary condition
            for j=1:flex
                [a b]=min(M(i-1,1:j+flex,1));
                M(i,j,1)=sumC(i,j)+a;
                M(i,j,2)=b-j;
            end

            %boundary condition
            for j=N-flex+1:N
                [a b]=min(M(i-1,j-flex:N,1));
                M(i,j,1)=sumC(i,j)+a;
                M(i,j,2)=b-flex-1;
            end
        end

    case 'hor'
        M=zeros([W,N,2],'single');
        %sum up the cost along the 1st dimension, and then rearrange it to
        %2-D array
        sumC=permute(sum(C,1),[2 3 1]);
        M(1,:,1)=sumC(1,:);
        for i=2:W
            for j=flex+1:N-flex
                [a b]=min(M(i-1,j-flex:j+flex,1));
                M(i,j,1)=sumC(i,j)+a;
                M(i,j,2)=b-flex-1;
            end

            %boundary condition
            for j=1:flex
                [a b]=min(M(i-1,1:j+flex,1));
                M(i,j,1)=sumC(i,j)+a;
                M(i,j,2)=b-j;
            end

            %boundary condition
            for j=N-flex+1:N
                [a b]=min(M(i-1,j-flex:N,1));
                M(i,j,1)=sumC(i,j)+a;
                M(i,j,2)=b-flex-1;
            end
        end
end
```

```
function [X C]=carve(X,C,M,direction)
%remove one ribbon using dynamic programming
%
%input:
%X: input video (4-D array)
%C: input cost (3-D array)
%M: cumulative minimum cost
%direction: 'vert': remove vertical ribbon
%          'hor': remove horizontal ribbon
%
%output:
%X: output video (4-D array)
%C: output cost (3-D array)

H=size(X,1);
W=size(X,2);
N=size(X,4);
switch direction
   case 'vert'
      [a p]=min(M(H,:,1));
      X(H,:,:,p:N-1)=X(H,:,:,p+1:N);
      C(H,:,p:N-1)=C(H,:,p+1:N);
      for i=H-1:-1:1
         p=p+M(i+1,p,2);
         X(i,:,:,p:N-1)=X(i,:,:,p+1:N);
         C(i,:,p:N-1)=C(i,:,p+1:N);
      end

   case 'hor'
      [a p]=min(M(W,:,1));
      X(:,W,:,p:N-1)=X(:,W,:,p+1:N);
      C(:,W,p:N-1)=C(:,W,p+1:N);
      for i=W-1:-1:1
         p=p+M(i+1,p,2);
         X(:,i,:,p:N-1)=X(:,i,:,p+1:N);
         C(:,i,p:N-1)=C(:,i,p+1:N);
      end

end
clear M a p direction H W N
X(:,:,:,end)=[];
C(:,:,end)=[];
```

7 References

[1] S. Avidan and A. Shamir, “Seam carving for content-aware image resizing,” *ACM Trans. Graph.*, vol. 26, no. 3, 2007

[2] Z. Li, P. Ishwar, and J. Konrad, “Video condensation by ribbon carving,” *IEEE Trans. Image Process.*, vol. 18, no. 11, Nov. 2009, pp.2572-2583

[3] J. McHugh, J. Konrad, V. Saligrama, and P.-M. Jodoin, “Foreground-adaptive background subtraction,” *IEEE Signal Process. Lett.*, vol. 16, pp. 390-393, May 2009.