Streaming Video Condensation by Ribbon Carving

Wei Liu

Dec 20, 2008
Boston University
Department of Electrical and Computer Engineering
Technical report No. ECE-2008-07

BOSTON UNIVERSITY

Streaming Video Condensation by Ribbon Carving

Wei Liu

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

Dec 20, 2008

Technical Report No. ECE-2008-07

This report describes research carried out as a partial requirement for the degree of Master of Science at the Department of Electrical and Computer Engineering, Boston University under the supervision of Profs. Janusz Konrad and Prakash Ishwar

Summary

The goal of this project is to develop a method to summarize streaming video based on ribbon carving recently developed at Boston University. Standard scaling of video in time, by means of regular or irregular sub-sampling, is not sufficient since it is oblivious to video content. Recently, seam carving has been demonstrated to change image size by gracefully removing or inserting pixels in different parts of the image. The idea of ribbon carving is an extension of seam carving to the case of video resizing in the temporal direction. The original implementation of video condensation was somewhat complex computationally and was not geared towards a real-time implementation at video frame rates. In this project, we reduce the computational complexity of ribbon carving by replacing the relatively complex non-parametric kernel model in background subtraction with a sliding-window temporal median filter. We also implement a dual-buffer (intensity and labels) sliding-window processing to handle streaming (endless) video sequence. Our implementation affords faster processing and is more efficient in terms of memory allocation than the original implementation.

Contents

List of Figures

List of Tables

1 Introduction

In surveillance video of bank or parking, there is no event or activity of interest in most of the frames. It's a waste of time to entirely browse such a long video sequence, and a more efficient technique is needed. Video summarization aims at extracting only interesting segments from surveillance video. It's especially effective for videos with long periods of inactivity.

Different from frame skipping, video condensation [2] is novel in the way information is removed from the original video, and is conceptually simple and relatively easy to implement. Video condensation method is based on content-aware resizing: seam carving and ribbon carving. Seam carving [1,3] has been developed for image resizing by means of gracefully removing or inserting pixels in different parts of the image.

The idea of ribbon carving, recently proposed in the literature [2], is an extension of seam carving to the case of video resizing in the direction of time. The content-aware resizing method has better performance in video summarization by improving the condensation rate without losing temporal order.

Video condensation by ribbon carving was developed at Boston University, but the original implementation has two limitations:

- it cannot handle streaming (endless) video, e.g., coming from a surveillance camera,
- is computationally complex due to the use of non-parametric background model used in detection of activities.

In this project, the research goal is to develop a more efficient video condensation algorithm for streaming video. We apply a sliding-window concept to process endless video by repeatedly reading original video frames into a buffer, condensing all frames in the buffer, writing out a block of video frames of the streaming video, and again reading new frames. In order to reduce computational complexity of motion detection we implement a sliding termporal-median filter as a background model for background subtraction.

2 Background Material

The traditional way to reduce the size of an image includes down-sampling, cropping, etc. However, these methods are all content oblivious.

Seam carving is a novel technique to resize image by reducing the horizontal/vertical dimension by one column/row at a time. Seams are associated with cost function which can reflect the content of the image, larger cost indicating more important content which is less desired to be removed. Desired resize ratio can be achieved by recursively removing least-cost seams from the original image. The recursive procedure together with the cost function tries to minimize the amount of content degradation caused by the deletion of seams.

Seam carving gives inspiration to ribbon carving in video, an extension to 3D case. In video condensation, the spatial size of the video frames is kept and only the temporal dimension of the video is shrunk, while events and their relative timings are preserved.

2.1 Seam Carving

Assume the size of original image is H*W, where H is height and W is width. A seam can be vertical or horizontal. In the following, vertical seam will be discussed in detail; horizontal seam has similar definition and properties.

2.1.1 Definition of vertical seam in an image

A vertical seam consists of H pixels that

1. extend from the top to the bottom of the image,
2. have different vertical coordinates, and
3. are path-connected as defined by flex parameter s.

i.e., a seam consists of one pixel in each row, but those pixels are not necessarily in a straight column, (i.e., they don't have, in genaral, the same horizontal coordinates). Pixels in a vertical seam form a vertically-oriented curve, with a flex parameter to decide how curved the seam can be. A vertical seam can be formally defined as a set of pixels (x(y), y), y = 1, . . . ,H, where x(y) is a function with range 1, . . . ,W, and the property that | x(y + 1) − x(y) | < s for all y = 1, . . . , (H − 1). Thus, x(y) describes the graph of a vertically-oriented curve which defines a vertical seam as shown in Fig. 1(a). The width an image is reduced by 1 when a vertical seam is removed from the original image.

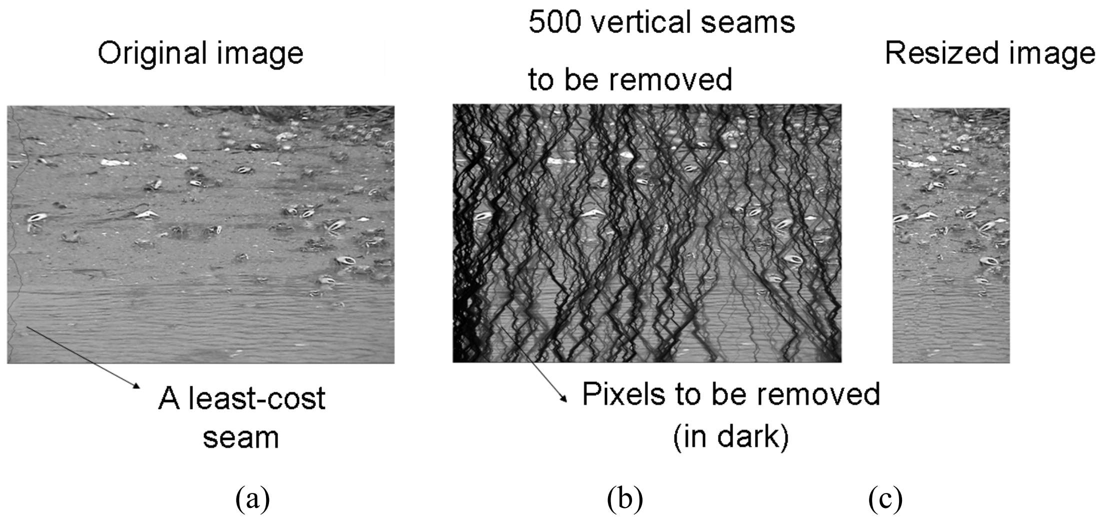

Original image
500 vertical seams to be removed
Resized image
A least-cost seam
Pixels to be removed (in dark)
(a)
(b)
(c)

Fig. 1 Illustration of seam carving: (a) original image a least-cost seam, (b) original image with seams-to-be-removed superimposed; (c) image after removal of seams.

In the figure below, vertical seams with flex parameter s=0, s=1, s=2, respectively, are shown, for a small 5x7 image.

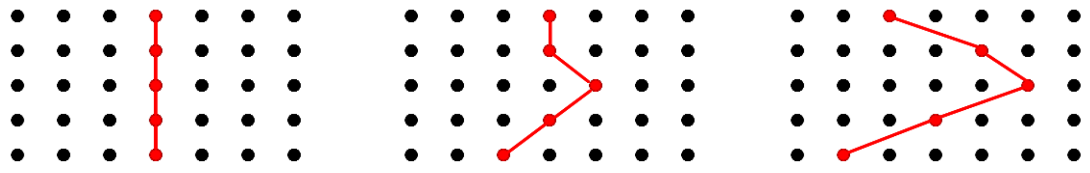

(a) s=0 (b) s=1 (c) s=2

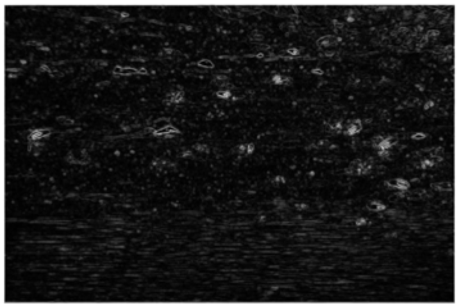
Fig. 2 Illustration of seam shape for different values of flex parameter s

A seam with s = 0 is a column and its removal is simply removal of a column from an image, either by cropping or by horizontal sub-sampling. A larger flex parameter s permits more flexible seams and thus suggests an ability to handle more complex image structures, and potentially greater re-sizing factors.

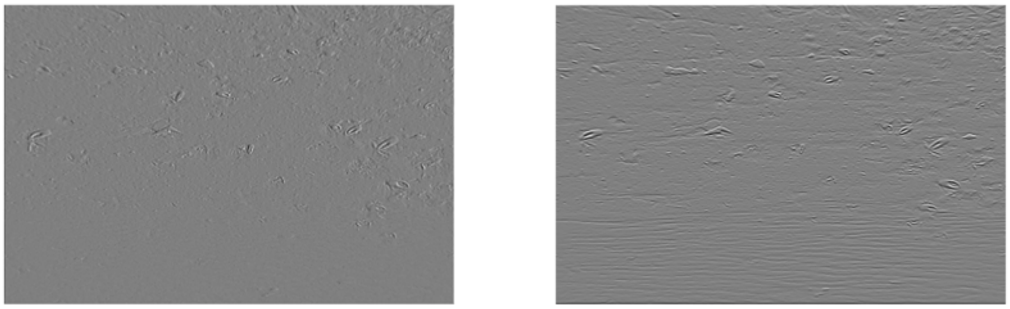
Since seam carving removes unimportant pixels from the original image, the resized image will contain a strict subset of pixels from the original image, but rearranged in some way.

2.1.2 Content-aware cost function

Seam carving associates content-aware costs with seams, which is the key idea of this novel method. When reducing image size, objects should be maintained as much as possible. To accomplish this, a seam should not carve through objects. In the cost image, the least-cost seam is selected to be removed and then the cost image is updated. Typical content-aware costs are based on weighted intensity gradients, which can be computed using standard 2D FIR filters. For example, this kind of cost function is suitable for an image in which objects have some texture while background is mainly smooth. As a result, the cost image will have large value where there is an object in the original image and small value otherwise. The least-cost seam will hardly carve any object.



(a) cost = length of gradient (using both horizontal and vertical gradients)



(b) cost = only horizontal gradient (c) cost = only vertical gradient

Fig. 3 Cost displayed as brightness value (bright = high cost) for the image from Fig. 1

It can be seen in Fig. 3 that horizontal gradient cost function detects mainly vertical edges but tends to miss horizontal ones, while vertical gradient cost function has the contrary result. Clearly, if looking for vertical seams in a cost image one needs to compute horizontal gradient, but then it is possible that the selected “optimal” vertical seam will carve horizontal edges in the original image. Similarly, horizontal seams might carve vertical edges if using vertical gradient as the cost function. Therefore, vertical seams should be found and carved in the cost image computed from vertical gradients, and horizontal seams - from horizontal gradients.

2.1.3 Finding least-cost seam using dynamic programming

An additive cost function together with the structure of a seam make it possible to formulate the search for a least-cost seam in terms of a dynamic programming which is guaranteed to find a least-cost seam and has a reasonable computational complexity.

After removing a seam, the cost image needs to be updated. For the reason that cost image is calculated by 2D filtering of the original image, re-calculating the cost image can be performed only along a narrow band around the removed seam, instead of the entire image. This can reduce the computational complexity, especially when implemented in C (less computational savings in Matlab).

2.1.4 Stopping criterion

There are two ways to stop the recursive procedure of carving out seams:

1. Pre-define the number of seams which need to be removed from the original image. In this case, the resize ratio is fixed and there might be some distortion in the resulting image, depending on how large the ratio is. In other words, it is a "lossy" carving, i.e., the removed seams might carve objects.

2. Pre-define a threshold such that only seams with lower costs than a threshold are removed. As a result, the resize ratio is not fixed but depends on the threshold. This is a "lossless" carving, since all the removed seams have costs lower than the threshold.

2.2 Ribbon Carving

Ribbon carving is an extension of 2D seam carving to 3D case. Depending on the orientation of ribbons, a video can be condensed in spatial dimension (vertically or horizontally) [3] or in time dimension. In this project, we focus on reducing the temporal length of the original video without changing the spatial size of video frames.

Therefore, a seam becomes a connected surface in 3D, which partitions the video into "past" and "future" regions. No two pixels in the surface have the same spatial coordinates. This property ensures that deleting the pixels belonging to this surface reduces the temporal dimension of the video block by exactly one.

Assume the size of video frame is H*W, where H is height and W is width. The buffer length is K, i.e., a block of K frames can be loaded into the buffer and get condensed. A ribbon can be vertical or horizontal. In the following, vertical ribbon will be discussed in detail; horizontal ribbons have similar definition and properties.

2.2.1 Definition of a vertical ribbon in space-time video volume

Consider a video sequence to be a space-time video volume with size: H*W*K. If the plane "y-t" is viewed as an image, a vertical seam of this "image" can be defined. A vertical ribbon consists of such vertical seams in all the planes "y-t" with different coordinates only in x axis. Formally, a vertical ribbon is a set of pixels $(x, y, t(y))$, $x = 1, . . . ,W$, $y = 1, . . . ,H$, where $t(y)$ is a function of only y with range $1, . . . ,K$, and the property that $| t(y +1) - t(y)| < s$ for all $y = 1, . . . , (H - 1)$.

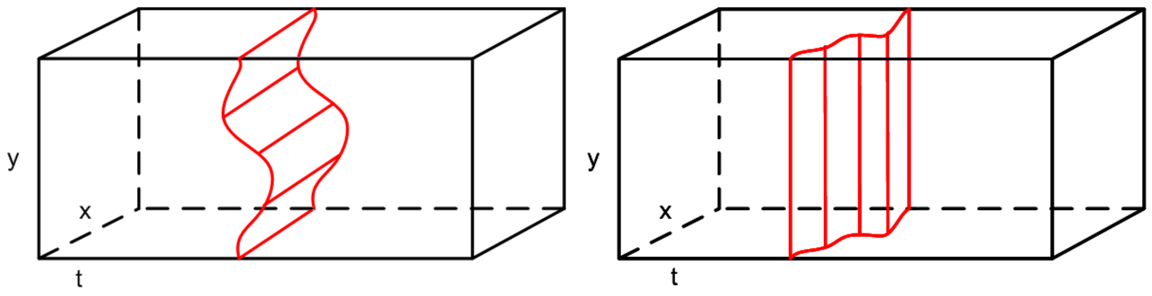


Fig. 4 A vertical ribbon and a horizontal ribbon

Similarly, a ribbon also has a flex parameter s to define how curved the ribbon can be in time dimension.

2.2.2 Cost function

When condensing video in time, usually maintaining events or moving objects is desirable while empty frames should be dropped. An activity-based or motion-based cost function should be used in this case to preserve moving object. Thus, we apply background subtraction to the original video to obtain a sequence of binary labels (0 – no activity, e.g., background, 1 – detected activity, e.g., movement). For example:

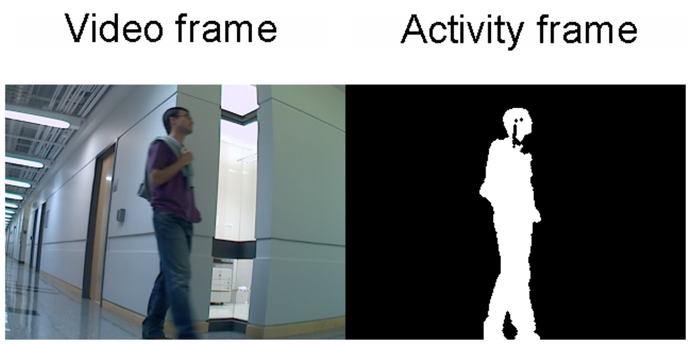


Fig. 5 Original frame and label frame as cost based on activity

Since a ribbon consists of several seams at different coordinates x, the cost of a ribbon can be computed as a sum of all the costs of the seams in this ribbon. With these observations, finding a least-cost vertical ribbon is equivalent to finding a least-cost vertical seam in the plane “y-t”. As shown in Fig. 6, first a summation needs to be performed along x axis to get a “y-t” plane “image”. Thus, the value of each pixel at position (y,t) in this “image” is the sum of binary values of all the pixels (x,y,t), where x = 1, . . . ,W. Secondly, a search for the least-cost vertical seam in this “image” needs to be performed, exactly the same procedure as in 2D case. When such a seam is found, one needs to repeat it in each “y-t” image with different x coordinates to form the least-cost vertical ribbon.

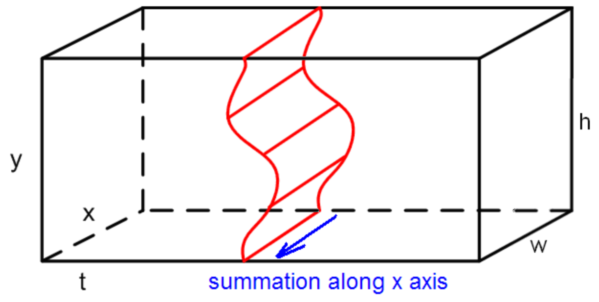


Fig. 6 Finding the least-cost vertical ribbon by converting to 2D case

2.2.3 Stopping criterion

Thus, one needs to find and remove the least-cost ribbon (which can be a vertical or horizontal ribbon) in the label sequence to reduce the length of video by 1 frame and, simultaneously, remove the corresponding ribbon from the original video. One needs to recursively carve out the least-cost ribbon until some stopping criterion is reached, i.e., no more ribbons can be removed from this block of video without violating the stopping criterion. The stopping criterion for ribbon carving is similar to that for seam carving.

2.2.4 Multi-level condensation using different flex parameter s

For a block of video frames, ribbon carving can be done in multi-level fashion, i.e., first take s = 0 and perform ribbon carving until the stopping criterion is met. Then, take s = 1 and apply ribbon carving to the resulting video, and so on, until the desired condensation ratio is reached.

3 Computationally-efficient ribbon carving of streaming video

3.1 Background subtraction via median filtering

3.1.1 Temporal median filter

Usually, during m consecutive frames, a pixel is occupied by background during most of the time, and by foreground only when the objects of interest move across. Thus, the background can be crudely estimated by median of the intensity values of the pixels in previous m frames. The value of background image at position (x,y) is then given by:

$$B(x,y;t_1,t_m) = median\{f(x,y,t_1), f(x,y,t_2),\dots, f(x,y,t_m)\}$$

where $f(x,y,t)$ is the value in the original video at position (x,y,t), i.e., in the t-th frame.

An activity frame is computed by thresholding the absolute value of difference between the estimated background and a new video frame.

$$L(x,y,t) = \begin{cases} 1 & |f(x,y,t) - B(x,y;t_1,t_m)| \ge \theta \\ 0 & |f(x,y,t) - B(x,y;t_1,t_m)| < \theta \end{cases} \qquad (t \ge t_m)$$

where $L(x,y,t)$ is the binary label frame and θ is a user-defined threshold.

3.1.2 Sliding median filter

To account for brightness changes in natural video due to illumination and camera gain variations, we update the background every m/2 frames. In consequence, consecutive background images are computed from m/2 shared frames and m/2 new frames, as shown in formula and figure below:

$$L(x,y,t) = \begin{cases} 1 & |f(x,y,t) - B(x,y;t_0-m-1,t_0-1)| \ge \theta \\ 0 & |f(x,y,t) - B(x,y;t_0-m-1,t_0-1)| < \theta \end{cases} \qquad t \in [t_0, t_0+m/2]$$

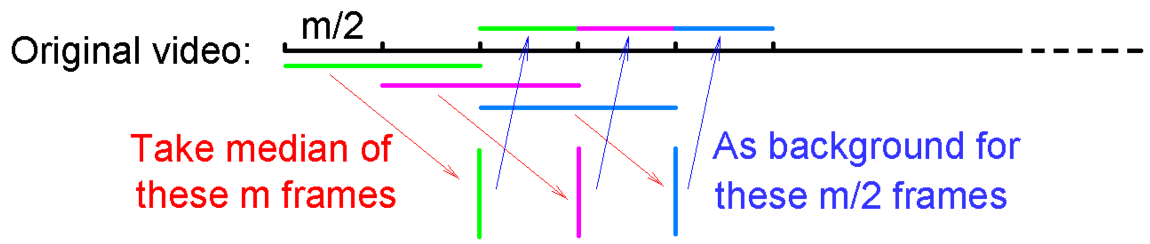

Original video:
m/2
Take median of these m frames
As background for these m/2 frames

Fig. 7 Sliding-median filter to compute background

3.1.3 Morphological operation

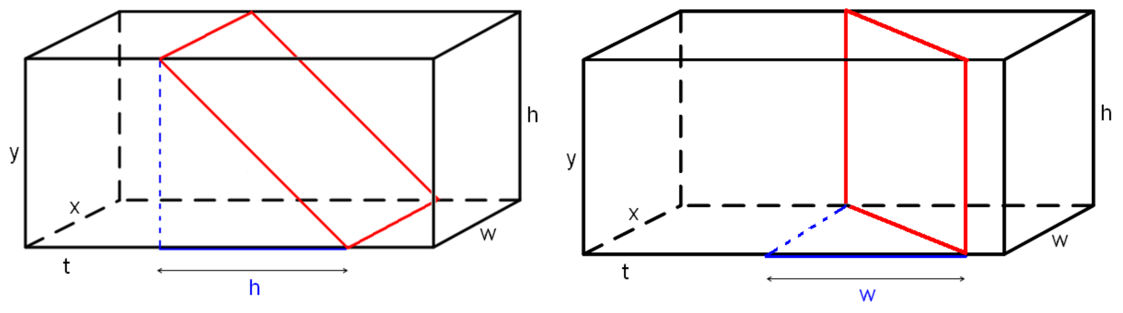
After background subtraction, we apply morphological operators to the labeled video in order to depress false positives and retrieve missing foreground. We erode a binary image, to get more background pixels and fewer foreground pixels, by sliding a disk, radius of which is defined by the user, over all background pixels and setting all pixels in the disk area to 0 (background). We dilate a binary image, a converse operation, to enlarge the area of foreground. In Matlab, first we use the function imerode to supress noise in the background, but this might cause more missing pixels in objects. Thus, we then use the function imdilate to retrieve those foreground pixels removed in the first step and perhaps further fill-in the holes in objects. In fact, the holes inside of objects are not of much concern since a ribbon won't carve them (an thus the object) as long as the holes are fully inside the objects. That's also the reason why first erode and then dilate.

3.2 Sliding-window processing of streaming video

Since the streaming video is basically endless while the amount of memory available for our program is relatively small, a technique for processing endless video is needed.

3.2.1 Important observation: the maximum extension of a ribbon in time

Due to the connection of pixels in a ribbon, it has a limited extension along time axis, which depends on the flex parameter s. A vertical ribbon can cover $s * (H – 1) + 1 = s * H – s + 1$ frames at most, and a horizontal ribbon can cover $s * (W – 1) + 1 = s * W – s + 1$ frames at most. The case of $s = 1$ is shown in the Fig. 8 below.



(a) vertical (b) horizontal

Fig. 8 Maximum extension of a ribbon with flex parameter s=1

With this observation, we know that the number of frames affected by a ribbon passing through a specific frame is limited. Therefore, as long as the condensed video block is longer than the maximum extension of a ribbon, there will be some frames unaffected by any ribbon, even if new frames are put together with the condensed ones. This allows us to write the unaffected frames out to a file, and it permits us to repeatedly read in, condense and write out, without buffer overflow.

3.2.2 Sliding-window processing of streaming video

It is the limited extension of a ribbon that makes the sliding-window algorithm possible to process streaming video. The basic idea is: load a block of frames from the original video into the buffer and condense the buffer until the stopping criterion is met, which will free some space in the buffer. As discussed above, because ribbons have limited extension along time axis, it is possible that there are some frames in the condensed video which won't be affected by any new ribbons. Therefore, these frames can be written out to a file and then some space in the buffer is freed up. Read new frames from the original video into the buffer until it is full. Repeat the procedure to condense a long video as desired.

Crucial steps are:

1. Initialize original video buffer and activity buffer.
2. Remove the lowest-cost ribbons from both the video and activity buffer
3. Write those video frames that won't be affected by new ribbons to an output file; drop the same activity frames.
4. Push all the remaining frames to the front of each buffer.
5. Read new video and activity frames into respective buffers until they are full.
6. Go back to step 1.

These steps are illustrated in Fig. 9 with the following notation:

- N = number of condensed frames;
- K - N = number of ribbons removed;
- P = N - (s * max{h , w} - s + 1) = number of frames written to a file;
- K - (N - P) = number of empty frames after writing;

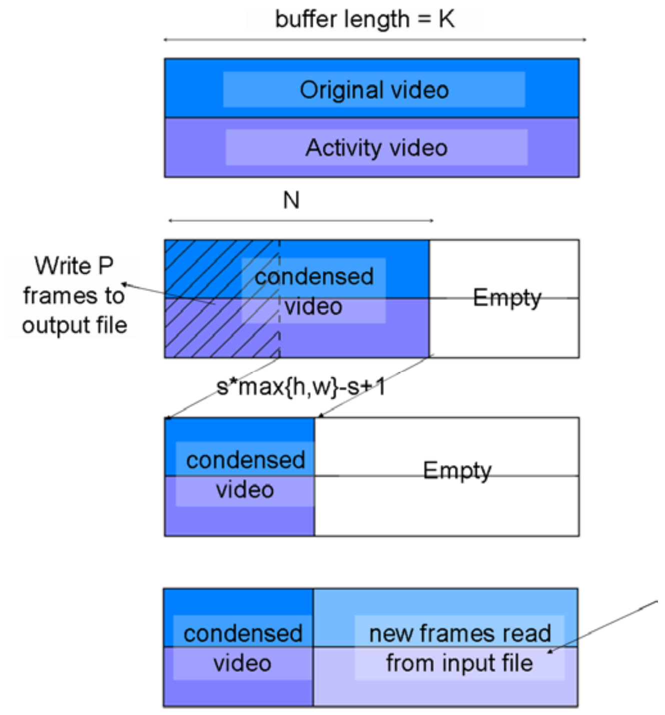


Fig. 9 Procedure of sliding-window algorithm for processing streaming video

3.2.3 Matlab functions

Below are listed Matlab functions developed.

`[E,Vseamindex,minE]=getVseamS(s,E,remainedvN)`

1) Function: to find the least-cost vertical seam
2) Input parameters:
   s: flex parameter, s = 1,2,…, but s < W/2; when s = 0, column deleting but not seam carving
   E: cost image (will be updated in the function)
   remainedvN: the number of columns left in the image after removing some seams
3) Output variables:
   E: cost image (updated in the function)
   Vseamindex: vector with length = H, to store the horizontal coordinates of pixels in the vertical seam
   minE: the cost of the least-cost vertical seam, i.e., the summation of the cost of all the pixels in the vertical seam
4) Program structure:

```
initialize: cost = E, which is to compute accumulative cost
for i = top to bottom
    for j = left to right
        decide search arrange [front, rear] which is depending on j (at left
boundary,           middle, or right boundary);
        find minimum cost(i - 1,k);
        record k;
        update cost(i, j) = cost(i, j) + minimum cost(i - 1,k);
    end
end
```

`[newI,tranI]=RemoveAndStackVseam(newI,tranI,Vseamindex,remainedvN)`

1) Function: to remove the least-cost vertical seam from image to get newI, and stack in order the removed seam in tranI

2) Input parameters:
   newI: “original” image ready to be remove a vertical seam (will be updated in the function)
   tranI: “image” consists of seams already removed (will be updated in the function)
   Vseamindex: (the same as above)
   remainedvN: (the same as above)
3) Output variables:
   newI: resized image (updated in the function)
   tranI: image” consists of seams already removed (updated in the function)

`E=getVcost (newI,filt1,filt2,E,remainedvN)`

1) Function: to re-compute cost only along the seam just removed and then get new cost image
2) Input parameters:
   newI: resized image
   filt1: horizontal filter
   filt2: vertical filter
   E: cost image (will be updated in the function)
   remainedvN: (the same as above)
3) Output variables:
   E: cost image (updated in the function)

Functions: `getHseamS, RemoveAndStackHseam, and getHcost` are similar to those functions above, with the only difference being the horizontal orientation.

Seam carving for image

1) Input parameters:
   I: original image
   s: (the same as above)
   Dx, Dy: horizontal and vertical filter
   VN, HN: user-defined number of vertical or horizontal seams to be removed

2) Output variables:

   newI: resized image

   tranI: image" consists of seams already removed

   labelI: label the removed seams in the original image

3) Program structure:

```
Initialize: newI = I;
E = cost of entire I;
for p = 1 to (VN - 1)
   [E,Vseamindex,notre]=getVseamS(s,E,W-p+1);
   [newI,tranI]=RemoveAndStackVseam (newI,tranI,Vseamindex,W-p+1);
   E=getVcost (newI,Dx,Dy,E,W-p+1);
end
p=VN;
[E,Vseamindex,notre]=getVseamS(s,E,W-p+1);
[newI,tranI]=RemoveAndStackVseam(newI,tranI,Vseamindex,W-p+1);
generate labelI;
```

Ribbon carving for streaming video

1) Input parameters:

   ovname: file name of the original avi file

   s: flex parameter, s = 1,2,…, but s < W/2; when s = 0, frame dropping but not ribbon carving

   tnforbg: the user-defined number of frames for temporal median filter for background subtraction

   thre: the user-defined threshold for background subtraction

   thre0: the user-defined threshold for s=0 ribbon carving, i.e., frame dropping

   thre1: the user-defined threshold for s=1 ribbon carving

   se1, se2: the user-defined disk size for morphological operation

2) Output:

   labelV2.avi: a binary video with the same length of the original video, in which 0 denotes background and 1 denotes foreground; it's the result

of background subtraction and morphological operation
condenseV.avi: the final result video after condensation of the original video
condenselabelV.avi: the corresponding result video after condensation of the labeled video
2condenseV: combine the condenseV.avi and condenselabelV.avi together, in order to see the condensed original video and condensed labeled video at the same time

3) Program structure:

```
load tnforbg frames of original video into buffer 0 (buffer for computing background);
compute median of buffer0 to get background image: bg;
load K frames ([ tnforbg + 1 ~ tnforbg + 1 + K ]) of original video into buffer 1 (buffer for original video);
maximum extension of ribbon: adps = s * max(H,W) - s + 1;
initialize: pointer of frame in original video: p = tnforbg + 1;
        number of frames in buffer 1: ninbuf =0;
        number of frames haven't been read in: ninvideo = total frame number;
while (ninvideo > 0)
    while (ninbuf < K)
        number of frames will be read in: nread = (K - ninbuf);
        read nread frames of original video in;
        ninvideo=ninvideo-nread;
        t = 1; k = ninbuf + 1; q = 0;
        while (t < nread)
            if (mod(p - 1, tnforbg / 2) == 0)
                update bg;
            end
            background subtraction and morphological operation, to get labeled frame;
            write to labeledV.avi;
            if ( above thre0)
                load this frame into buffer2 (buffer for labeled video);
                k++;
            else
```

```
                delete the corresponding frame from buffer1;
                q++;
            end
            t++; p++;
        end
        ninbuf = ninbuf + nread - q;
    end
    remainedtN = K;
    summation along x axis to get “y-t” image;
    summation along y axis to get “x-t” image;
    [vrImage,vrIndex,miniVcost]=getVseamS(s,vrImage,remainedtN);
    [hrImage,hrIndex,miniHcost]=getHseamS(s,hrImage,remainedtN);
    while ((miniVcost <= thre1) || (miniHcost <= thre1))
        if (miniVcost <= miniHcost)
            remove the vertical ribbon in buffer1 and buffer2;
            remainedtN - - ;
            summation along y axis to get “x-t” image;
        else
            remove the horizontal ribbon in buffer1 and buffer2;
            remainedtN - - ;
            summation along x axis to get “y-t” image;
        end
        [vrImage,vrIndex,miniVcost]=getVseamS(s,vrImage,remainedtN);
        [hrImage,hrIndex,miniHcost]=getHseamS(s,hrImage,remainedtN);
    end
    number of frames to be written out: nwrite = remainedtN - adps;
    if (nwrite > 0)
        write nwrite frames in buffer1 and buffer2 out to file;
        push the remaining frames in buffer1 and buffer2 to the front;
    end
    ninbuf = minimum of {adps, remainedtN};
end
write all the remaining frames in buffer1 and buffer2 to file;
```

3.2.4 Sliding-window algorithm

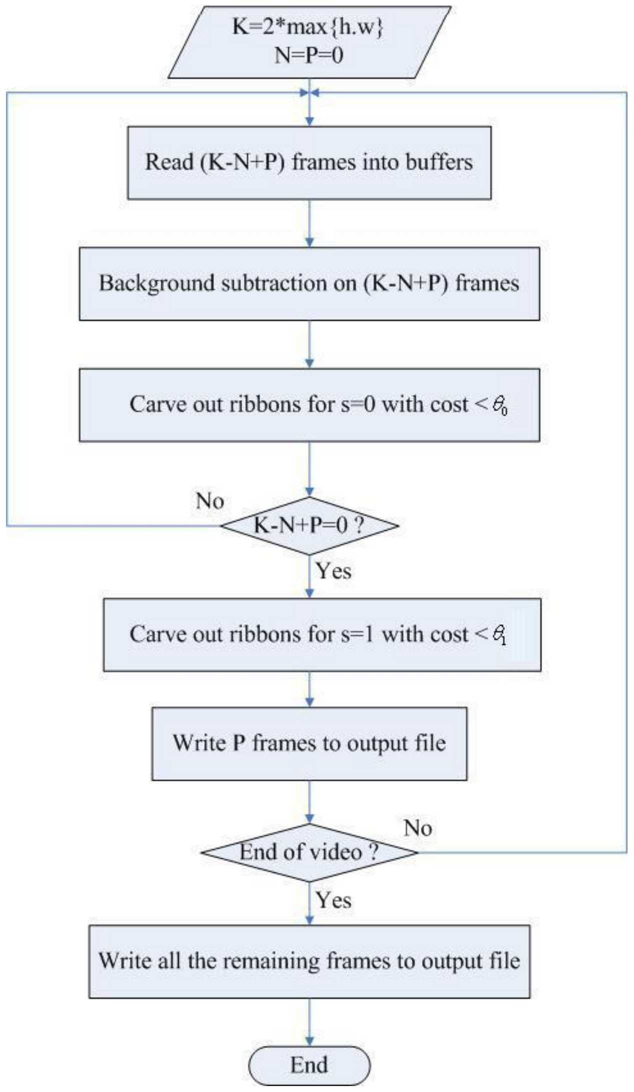

K=2*max{h.w}
N=P=0
Read (K-N+P) frames into buffers
Background subtraction on (K-N+P) frames
Carve out ribbons for s=0 with cost < θ_0
No
K-N+P=0 ?
Yes
Carve out ribbons for s=1 with cost < θ_1
Write P frames to output file
No
End of video ?
Yes
Write all the remaining frames to output file
End

Fig. 10 Flowchart of the sliding-window algorithm

4 Experimental Results

The sliding-window video condensation algorithm developed in this project has been tested on two video sequences with parameters listed in Table 1 and sample frames shown in Figs. 11 and 13.

Table 1 Original video sequence specifications

Video	Spatial resolution	Time duration	Total number of frames
Indoor traffic	240*320	5’10’’	9000
Outdoor traffic	208*240	4’25’’	7991

4.1 Condensation performance

Two frames from the original *Indoor traffic* video and one condensed frame are shown in Fig. 11 below.

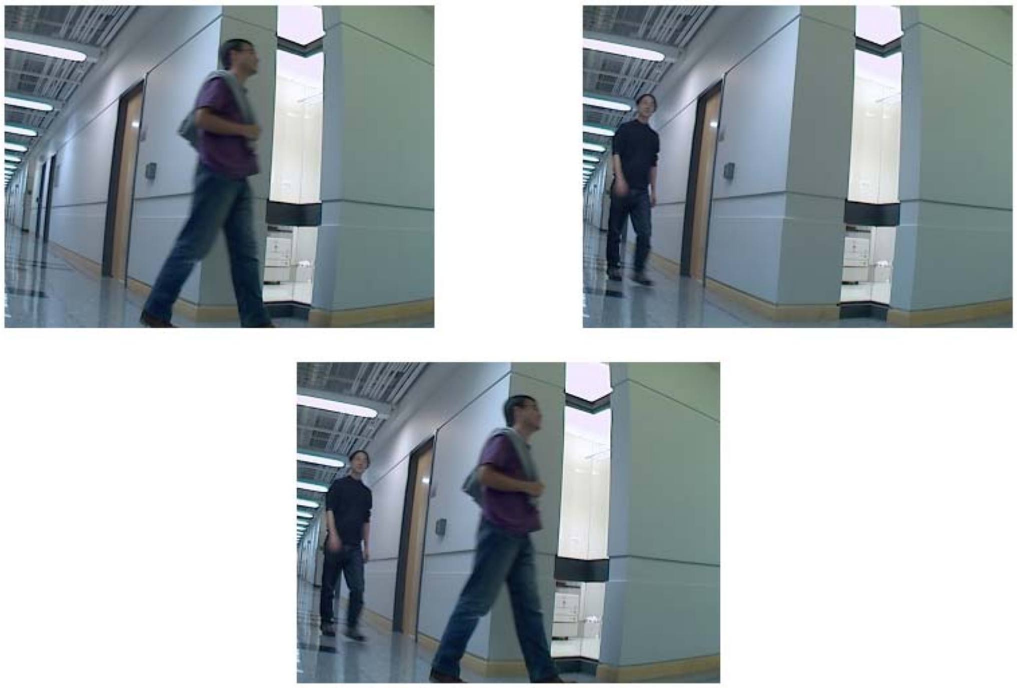
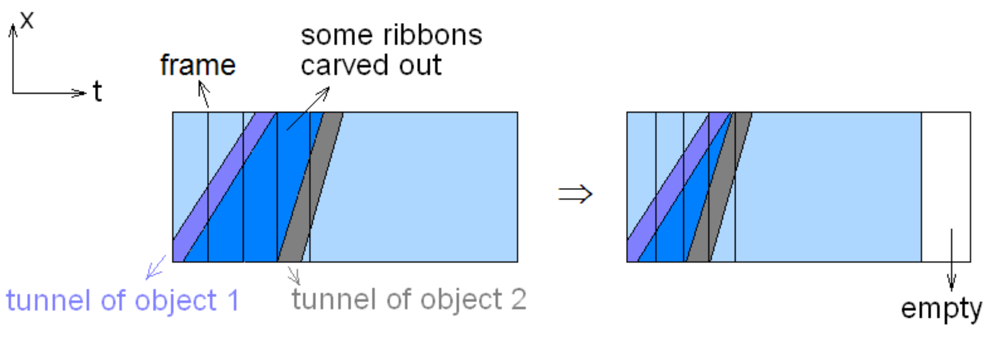


Fig. 11 Condensed *Indoor traffic* video: two original frames (top) and one condensed frame (bottom)

In the condensed video, objects from different frames appear at the same frame. That’s because the volume between the two object moving tunnels is carved out, as shown in Fig. 12 below.



(a) before condensation (b) after condensation

Fig. 12 Object tunnels

It can be easily seen that before condensation each frame only intersects with one object tunnel; in every frame only one person appears. When watching the original video, we'll see that the first person walked in, and then after he left the scene, the second person walked in. However, in the condensed video, there are several frames which intersect with both of the object tunnels. Thus, the two persons will appear in the same frame.

The same effect is seen in the *Outdoor traffic* video in Fig. 13. Four pedestrians observed in separate frames in the original video, occur in one frame in the condensed video.

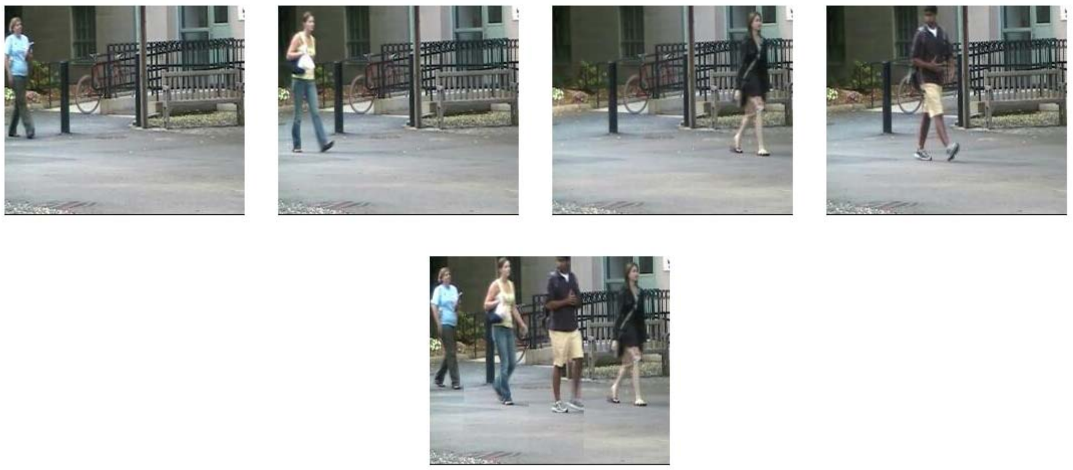


Fig. 13 Condensed *Outdoor traffic* video: four original frames (top) and one condensed frame (bottom)

4.2 Condensation ratio

Condensation ratio is defined as the ratio of the length of the original video to the length of condensed video. It depends on several factors, such as how frequently events happen in the original video, the thresholds of background subtraction and ribbon carving, and the flex parameter of ribbons. Parameters used in Table 1 are defined as follows:

- m = number of frames to compute the background
- θ = acceptable cost threshold for generating labeled video
- θ_0 = acceptable cost threshold for ribbon carving with s=0
- θ_1 = acceptable cost threshold for ribbon carving with s=1

Table 2 Comparison of condensation ratios attained

Video	M	θ	θ_0	θ_1	Condensation ratio
Inner-building	200	30	100	3	1.94
Inner-building	200	30	100	100	2.00
Pedestrian traffic	100	30	100	3	2.82

Condensation ratios attained are about 2~3 and are dependent on the original video. If there are frequent events, the condensation ratio is not high.

Smaller m is used for pedestrian traffic video, because there is more severe brightness change, which requires more frequent updating of the background. Increasing θ_0 and θ_1 encourages higher condensation ratios. However, as a trade-off, objects might get partially or fully carved out or in the worst case, there will be visible time distortion.

4.3 Processing rate

In Matlab, processing rate is about 1 frame per second. A further speed-up would be possible through an implementation in C. We have tested this on seam carving (2-D case). For example, to remove 500 vertical seams from a 480*720 image:

- Matlab implementation requires 597 sec,
- C implementation requires 10.5 sec,

which is a significant improvement.

5 Conclusions

5.1 Performance

From these result, we can see that ribbon carving is effective and efficient on indoor and outdoor pedestrian traffic videos. With first s=0 and then s=1 ribbon carving, original video can be condensed by the factor of about 2~3. The condensation preserves all the important events and their relative timings with little degradation of the quality of the perceived video. By using the sliding-window algorithm, the up to date implementation of ribbon carving can handle streaming video. The computational complexity is further reduced by the means of sliding-median filter for background subtraction.

5.2 Encountered Problems

A problem with the current implementation is a visible moving line in the condensed video caused by brightness change in the original video over time, as shown in Fig. 14.

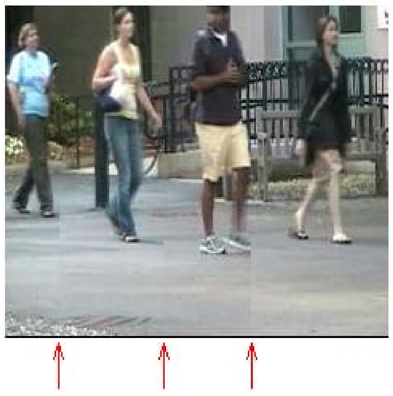


Fig. 14 Discontinuity due to brightness change

It is quite common that brightness of a scene changes gradually and is captured by video. Thus, when frames without events are dropped and far-apart frames are pushed together, there will be a difference of brightness. As the condensed video is playing, the visible line is moving forward as objects move. The problem might be solved by temporal filter only on background but not on foreground objects.

5.3 Future work

To improve the algorithm of ribbon carving for streaming video, the future work should focus on the following points:

1) an effective way to deal with brightness change which causes visible line between objects pairs;
2) real-time implementation of video condensation in C/C++;

use of threading on multi-core architecture.

References

[1] S. Avidan and A. Shamir. Seam carving for content-aware image resizing. *ACM Trans. Graph. 26, 3 (Jul. 2007), 10*
.

[2] Zhuang Li, Prakash Ishwar, and Janusz Konrad. Video Condensation by Ribbon Carving. *IEEE Trans. Image Process., Oct. 2008 (submitted).*

[3] M. Rubinstein, A. Shamir, and S. Avidan. Improved seam carving for video retargetting. ACM Trans. Graph., vol. 27, no. 3, 2008.

Appendix

Below is listed Matlab source code developed for this project.

1. Matlab Functions:

```
function [E,Vseamindex,minE]=getVseamS(s,E,remainedvN)
[m,n]=size(E);% m & n should be global variance in C++
cost=E(:,1:remainedvN);
flag=zeros(m,remainedvN);

for i=2:m
    for j=1:remainedvN
        if(j<=s)
            front=1;rear=j+s;
        end
        if((j>s)&&((j+s)<=remainedvN))
            front=j-s;rear=j+s;
        end
        if((j+s)>remainedvN)
            front=j-s;rear=remainedvN;
        end
        minvalue=cost(i-1,front);location=front;
        for k=front:j
            if(cost(i-1,k)<=minvalue)
                minvalue=cost(i-1,k);
                location=k;
            end
        end
        for k=(j+1):rear
            if(cost(i-1,k)<minvalue)
                minvalue=cost(i-1,k);
                location=k;
            end
            if((cost(i-1,k)==minvalue)&&(abs(k-j)<abs(location-j)))
                minvalue=cost(i-1,k);
                location=k;
            end
        end
        flag(i,j)=location;% record path
        cost(i,j)=cost(i,j)+minvalue;% update
    end
end

[minE IND]=min(cost(m,1:remainedvN));
Vseamindex=zeros(m,1);
Vseamindex(m)=IND;
for i=m-1:-1:1
    Vseamindex(i)=flag(i+1,Vseamindex(i+1));
end

% update E
for i=1:m
```

```
    for j=Vseamindex(i):(remainedvN-1)
        E(i,j)=E(i,j+1);
    end
    E(i,remainedvN)=Vseamindex(i);
end


function [E,Hseamindex,minE]=getHseamS(s,E,remainedhN)
[m,n]=size(E);% m & n should be global variance in C++
cost=E(1:remainedhN,:);
flag=zeros(remainedhN,n);

for j=2:n
    for i=1:remainedhN
        if(i<=s)
            front=1;rear=i+s;
        end
        if((i>s)&&((i+s)<=remainedhN))
            front=i-s;rear=i+s;
        end
        if((i+s)>remainedhN)
            front=i-s;rear=remainedhN;
        end
        minvalue=cost(front,j-1);location=front;
        for k=front:i
            if(cost(k,j-1)<=minvalue)
                minvalue=cost(k,j-1);
                location=k;
            end
        end
        for k=(i+1):rear
            if(cost(k,j-1)<minvalue)
                minvalue=cost(k,j-1);
                location=k;
            end
            if((cost(k,j-1)==minvalue)&&(abs(k-i)<abs(location-i)))
                minvalue=cost(k,j-1);
                location=k;
            end
        end
        flag(i,j)=location;% record path
        cost(i,j)=cost(i,j)+minvalue;% update
    end
end

[minE IND]=min(cost(1:remainedhN,n));
Hseamindex=zeros(1,n);
Hseamindex(n)=IND;
for j=n-1:-1:1
    Hseamindex(j)=flag(Hseamindex(j+1),j+1);
end

% update E
for j=1:n
    for i=Hseamindex(j):(remainedhN-1)
```

```
        E(i,j)=E(i+1,j);
    end
    E(remainedhN,j)=Hseamindex(j);
end


function [newI,tranI]=RemoveAndListVseam2(newI,tranI,index,remainedvN)
[m,n]=size(newI);% m & n should be global variance in C++
k=n-remainedvN+1;
for i=1:m
    tranI(i,k)=newI(i,index(i));
    for j=index(i)+1:remainedvN
        newI(i,j-1)=newI(i,j);
    end
end


function [newI,tranI]=RemoveAndListHseam2(newI,tranI,index,remainedhN)
[m,n]=size(newI);% m & n should be global variance in C++
k=m-remainedhN+1;
for j=1:n
    tranI(k,j)=newI(index(j),j);
    for i=index(j)+1:remainedhN
        newI(i-1,j)=newI(i,j);
    end
end


function Cost=getcost33(newI,filt1,filt2,Cost,remainedvN)
    [m,n]=size(Cost);% m & n should be global variance in C++
    k=n-remainedvN+1;
    Vseamindex=Cost(:,remainedvN);
    costarray1=Vseamcost(newI,Vseamindex-2,filt1,filt2,k);
    costarray2=Vseamcost(newI,Vseamindex-1,filt1,filt2,k);
    costarray3=Vseamcost(newI,Vseamindex+1-1,filt1,filt2,k);
    costarray4=Vseamcost(newI,Vseamindex+2-1,filt1,filt2,k);

    for i=1:m
        if(((Vseamindex(i)-2)>0)&&((Vseamindex(i)-2)<remainedvN))
            Cost(i,Vseamindex(i)-2)=costarray1(i);
        end
        if(((Vseamindex(i)-1)>0)&&((Vseamindex(i)-1)<remainedvN))
            Cost(i,Vseamindex(i)-1)=costarray2(i);
        end
        if(((Vseamindex(i)+1-1)>0)&&((Vseamindex(i)+1-1)<remainedvN))
            Cost(i,Vseamindex(i)+1-1)=costarray3(i);
        end
        if(((Vseamindex(i)+2-1)>0)&&((Vseamindex(i)+2-1)<remainedvN))
            Cost(i,Vseamindex(i)+2-1)=costarray4(i);
        end
    end


function Cost=getcost99(newI,filt1,filt2,Cost,remainedhN)
    [m,n]=size(Cost);% m & n should be global variance in C++
    k=m-remainedhN+1;
    Hseamindex=Cost(remainedhN,:);
    costarray1=Hseamcost(newI,Hseamindex-2,filt1,filt2,k);
    costarray2=Hseamcost(newI,Hseamindex-1,filt1,filt2,k);
    costarray3=Hseamcost(newI,Hseamindex+1-1,filt1,filt2,k);
    costarray4=Hseamcost(newI,Hseamindex+2-1,filt1,filt2,k);

    for j=1:n
        if((Hseamindex(j)-2)>0)&&((Hseamindex(j)-2)<remainedhN)
            Cost(Hseamindex(j)-2,j)=costarray1(j);
        end
        if((Hseamindex(j)-1)>0)&&((Hseamindex(j)-1)<remainedhN)
            Cost(Hseamindex(j)-1,j)=costarray2(j);
        end
        if((Hseamindex(j)+1-1)>0)&&((Hseamindex(j)+1-1)<remainedhN)
            Cost(Hseamindex(j)+1-1,j)=costarray3(j);
        end
        if((Hseamindex(j)+2-1)>0)&&((Hseamindex(j)+2-1)<remainedhN)
            Cost(Hseamindex(j)+2-1,j)=costarray4(j);
        end

    end


function costarray=Vseamcost2(I,Vseamindex,filt0,time)
[mI,nI]=size(I);
[mf,nf]=size(filt0);
k=nI-time;
xf=floor(nf/2);yf=floor(mf/2);
mIex=mI+2*yf;
nIex=k+2*xf;

Iex=padarray(I(:,1:k),[yf,xf]);% zeros--what "conv2" does with boudaries

costarray=zeros(mI,1);


for k=1:mI
    for i=-yf:1:yf
        for j=-xf:1:xf
            if((Vseamindex(k)+j+yf)>0)&&((Vseamindex(k)+j+yf)<nIex+1)

costarray(k)=costarray(k)+filt0(i+ceil(mf/2),j+ceil(nf/2))*Iex(k+i+xf,V
seamindex(k)+j+yf);
            end
        end
    end
    costarray(k)=abs(costarray(k));
end


function costarray=Hseamcost2(I,Hseamindex,filt0,time)
[mI,nI]=size(I);
[mf,nf]=size(filt0);
k=mI-time;
```

```
xf=floor(nf/2);yf=floor(mf/2);
mIex=k+2*yf;
nIex=nI+2*xf;

Iex=padarray(I(1:k,:),[yf,xf]);% zeros--what "conv2" does with boudaries

costarray=zeros(1,nI);

for k=1:nI
   for i=-yf:1:yf
      for j=-xf:1:xf
         if((Hseamindex(k)+i+xf)>0)&&((Hseamindex(k)+i+xf)<mIex+1)

costarray(k)=costarray(k)+filt0(i+ceil(mf/2),j+ceil(nf/2))*Iex(Hseamind
ex(k)+i+xf,k+j+yf);
         end
      end
   end
   costarray(k)=abs(costarray(k));
end
```

2. Seam carving for image

```
clear;close all;clc;
tic;

I=double(rgb2gray(imread('crab.tif')));
figure;imshow(I,[0 255]);title('Image 1: Original image');% original image

Dx=[-1 0 1;-2 0 2;-1 0 1];
Dy=[1 2 1;0 0 0;-1 -2 -1];
s=1;

 [m,n]=size(I);
newI=I; % newI: smaller image with Vseamindex

Gx=conv2(I,Dx,'same');
Gy=conv2(I,Dy,'same');
E=sqrt(Gx.^2+Gy.^2);
% % remove VN columns
VN=50;
tranI=zeros(m,VN);
for p=1:VN-1
    [E,Vseamindex,notre]=getVseamS(s,E,n-p+1);
    [newI,tranI]=RemoveAndListVseam2(newI,tranI,Vseamindex,n-p+1);
    E=getcost33(newI,Dx,Dy,E,n-p+1);
end
p=VN;
 [E,Vseamindex,notre]=getVseamS(s,E,n-p+1);
 [newI,tranI]=RemoveAndListVseam2(newI,tranI,Vseamindex,n-p+1);

figure;imshow(newI(:,1:(n-VN)),[0 255]);title(['Image 2: ',int2str(VN),'
Vseams removed from Image 1']);% the new, smaller image
```

```
maskI=[ones(m,n-VN) zeros(m,VN)];
for p=1:VN
    k=n-VN+p-1;
    for i=1:m
        q=E(i,k+1);
        for j=k:-1:q
            maskI(i,j+1)=maskI(i,j);
        end
        maskI(i,q)=(n-k)/(VN+1);
    end
end
% maskI
labelI=floor(maskI.*I);
figure;imshow(maskI);title('Mask for Image 1');% mask as an image
figure;imshow(labelI,[0 255]);title('Labeled Image 1');% original image
with removed Vseams labeled
figure;imshow(tranI,[0 255]);title(['List ',int2str(VN),' Vseams from
Image 1']);

clear I;
I=newI(:,1:(n-VN));
figure;imshow(I,[0 255]);title('Image 2, again');

%
%
 [m,n]=size(I);
newI=I;
Gx=conv2(I,Dx,'same');
Gy=conv2(I,Dy,'same');
E=sqrt(Gx.^2+Gy.^2);

% remove HN rows
%
HN=30;
tranI=zeros(HN,n);
for p=1:HN-1
    [E,Hseamindex,notre]=getHseamS(s,E,m-p+1);
    [newI,tranI]=RemoveAndListHseam2(newI,tranI,Hseamindex,m-p+1);
    E=getcost99(newI,Dx,Dy,E,m-p+1);
end
p=HN;
 [E,Hseamindex,notre]=getHseamS(s,E,m-p+1);
[newI,tranI]=RemoveAndListHseam2(newI,tranI,Hseamindex,m-p+1);

figure;imshow(newI(1:(m-HN),:),[0 255]);title(['Image 3: ',int2str(HN),'
Hseams removed from Image 2']);% the new, smaller image

maskI=[ones(m-HN,n);zeros(HN,n)];
for p=1:HN
    k=m-HN+p-1;
    for j=1:n
        q=E(k+1,j);
        for i=k:-1:q
            maskI(i+1,j)=maskI(i,j);
        end
        maskI(q,j)=(m-k)/(HN+1);
    end
end
labelI=floor(maskI.*I);
figure;imshow(maskI);title('Mask for Image 2');% mask as an image
figure;imshow(labelI,[0 255]);title('Labeled Image 2');% original image
with removed Hseams labeled
figure;imshow(tranI,[0 255]);title(['List ',int2str(HN),' Hseams from
Image 2']);

t=toc
```

3. Ribbon carving for streaming video:

```
clc;clear;tic;
% % Part 1: Read in original video
mov1 = avifile('labelV.avi','compression','none','fps',30);
mov1_do = avifile('labelV2.avi','compression','none','fps',30);
mov2 = avifile('2condenseV.avi','compression','none','fps',30);
mov3 = avifile('condenseV.avi','compression','none','fps',30);
mov4 = avifile('condenselabelV.avi','compression','none','fps',30);

tnforbg=100;     % # of frames used to compute background

% get infomation from the original video
start_time_b=0;
end_time_b=1/30;
video=mmread('group_original.avi',[],[start_time_b end_time_b]);
tmp=video.frames;
[M,N,color]=size(tmp.cdata); % color==3;

% initialization
s=1;adps=s*max(M,N);% adaptive parameter
K=2*adps; % how many frames should be read in and processed at one time
          % length of buffer

bg=zeros(M,N);
temp2=zeros(M,N,3);
temp3=zeros(M,2*N,3);
bgVideo=uint8(zeros(M,N,tnforbg));
originalVideo=uint8(zeros(M,N,K));
originalVideoR=uint8(zeros(M,N,K));
originalVideoG=uint8(zeros(M,N,K));
originalVideoB=uint8(zeros(M,N,K));
labelVideo=logical(zeros(M,N,K));% labeled video: white as foreground &
black as background -- should be boolean value

thre1=30;
thre2=100;
se1 = strel('disk',2);
se2 = strel('disk',1);

flag=logical(1);
```

```
% initialization of buffer for computing background (half full)
start_time_b=0;
end_time_b=start_time_b+(tnforbg/2)/30;
video=mmread('group_original.avi',[],[start_time_b end_time_b]);
tmp=video.frames;
[st,nbg]=size(tmp);   % st==1
for k=1:min(tnforbg/2,nbg)
    bgVideo(:,:,k)=rgb2gray(tmp(k).cdata);
end

ninbuf=0;
end_time=tnforbg/30;
ninvideo=video.nrFramesTotal; % # of frames which haven't be read in
p=tnforbg+1;

while(ninvideo>0)
    while(ninbuf<K)
        disp('read');
        nread=K-ninbuf;
        start_time=end_time;
        end_time=min(start_time+nread/30,video.totalDuration);
        video=mmread('group_original.avi',[],[start_time end_time]);
        tmp=video.frames;
        [st,nread]=size(tmp);
        ninvideo=ninvideo-nread;
        tem=min(nread,K-ninbuf);
        nread=tem;
        for k=1:nread
            originalVideo(:,:,k+ninbuf)=rgb2gray(tmp(k).cdata);
            originalVideoR(:,:,k+ninbuf)=tmp(k).cdata(:,:,1);
            originalVideoG(:,:,k+ninbuf)=tmp(k).cdata(:,:,2);
            originalVideoB(:,:,k+ninbuf)=tmp(k).cdata(:,:,3);
        end
        q=0;
        k=ninbuf+1;t=1;
        while(t<=nread)
            if(mod(p-1,tnforbg/2)==0)
    %           disp('update bg');
                start_time_b=end_time_b;
                end_time_b=start_time_b+(tnforbg/2)/30;
                video=mmread('group_original.avi',[],[start_time_b
end_time_b]);
                tmp=video.frames;
                [st,nbg]=size(tmp);   % st==1

                for
bgk=(1+flag*tnforbg/2):(min(tnforbg/2,nbg)+flag*tnforbg/2)

bgVideo(:,:,bgk)=rgb2gray(tmp(bgk-flag*tnforbg/2).cdata);
                end
                flag=~flag;
                % background
                for i=1:M
                    for j=1:N
                        bg(i,j)=median(double(bgVideo(i,j,:)));
                    end
                end
    %               figure;imshow(bg,[]);
            end
            temp=(abs(double(originalVideo(:,:,k))-bg))>thre1; % temp is
boolean

            % write temp to file
            temp2(:,:,1)=255*temp;
            temp2(:,:,2)=temp2(:,:,1);
            temp2(:,:,3)=temp2(:,:,1);
            temp2=uint8(temp2);
            mov1=addframe(mov1,temp2);

            temp_d=imerode(temp, se1);
            temp_do=imdilate(temp_d, se2);

            temp2(:,:,1)=255*temp_do;
            temp2(:,:,2)=temp2(:,:,1);
            temp2(:,:,3)=temp2(:,:,1);
            temp2=uint8(temp2);
            mov1_do=addframe(mov1_do,temp2);

            if(sum(sum(temp_do))>thre2)
                labelVideo(:,:,k)=temp_do;
                k=k+1;
            else
                for v=k:(ninbuf+nread-1-q)
                    originalVideo(:,:,v)=originalVideo(:,:,v+1);
                    originalVideoR(:,:,v)=originalVideoR(:,:,v+1);
                    originalVideoG(:,:,v)=originalVideoG(:,:,v+1);
                    originalVideoB(:,:,v)=originalVideoB(:,:,v+1);
                end
                q=q+1;
            end
            t=t+1;
            p=p+1;
        end
        ninbuf=ninbuf+nread-q;
    end

    remainedtN=K;

    % % Part 3: Ribbon carving
    s=1;
    vrImage=zeros(M,remainedtN);
    hrImage=zeros(remainedtN,N);
    % get vrImage
    for k=1:remainedtN
        for i=1:M
            vrImage(i,k)=sum(labelVideo(i,:,k));
        end
    end

    % get hrImage
```

```
   for k=1:remainedtN
      for j=1:N
         hrImage(k,j)=sum(labelVideo(:,j,k));
      end
   end

   [vrImage,vrIndex,miniVcost]=getVseamS(s,vrImage,remainedtN);
   [hrImage,hrIndex,miniHcost]=getHseamS(s,hrImage,remainedtN);

   while((miniVcost<=3)||(miniHcost<=3))
      if(miniVcost<=miniHcost) % Vertical
         disp('vertical');
         for i=1:M
            for k=vrIndex(i)+1:remainedtN
                  originalVideo(i,:,k-1)=originalVideo(i,:,k);    %
                  labelVideo(i,:,k-1)=labelVideo(i,:,k);         %
removeVribbon
                  originalVideoR(i,:,k-1)=originalVideoR(i,:,k);
                  originalVideoG(i,:,k-1)=originalVideoG(i,:,k);
                  originalVideoB(i,:,k-1)=originalVideoB(i,:,k);
            end
         end
         remainedtN=remainedtN-1;
         % get hrImage
         for k=1:remainedtN
            for j=1:N
               hrImage(k,j)=sum(labelVideo(:,j,k));
            end
         end

      else % Horizontal
         disp('horizontal');
         for j=1:N
               for k=hrIndex(j)+1:remainedtN
                  originalVideo(:,j,k-1)=originalVideo(:,j,k);    %
                  labelVideo(:,j,k-1)=labelVideo(:,j,k);         %
removeHribbon
                  originalVideoR(:,j,k-1)=originalVideoR(:,j,k);
                  originalVideoG(:,j,k-1)=originalVideoG(:,j,k);
                  originalVideoB(:,j,k-1)=originalVideoB(:,j,k);
               end
         end
         remainedtN=remainedtN-1;
         % get vrImage
         for k=1:remainedtN
            for i=1:M
               vrImage(i,k)=sum(labelVideo(i,:,k));
            end
         end

      end % of: if else
      [vrImage,vrIndex,miniVcost]=getVseamS(s,vrImage,remainedtN);
      [hrImage,hrIndex,miniHcost]=getHseamS(s,hrImage,remainedtN);
   end % of: while
```

```
    % write some processed frames to file
    nwrite=remainedtN-adps;
    if(nwrite>0)
       disp('write');
       % write out originalVideo(:,:,1:nwrite) and
labelVideo(:,:,1:nwrite)
       for k=1:nwrite
           temp3(:,1:N,1)=originalVideoR(:,:,k);
           temp3(:,1:N,2)=originalVideoG(:,:,k);
           temp3(:,1:N,3)=originalVideoB(:,:,k);

           temp2=255*labelVideo(:,:,k);
           temp3(:,(N+1):(2*N),1)=temp2;
           temp3(:,(N+1):(2*N),2)=temp2;
           temp3(:,(N+1):(2*N),3)=temp2;
           temp3=uint8(temp3);
           mov2=addframe(mov2,temp3);
           mov3=addframe(mov3,temp3(:,1:N,:));
           mov4=addframe(mov4,temp3(:,(N+1):(2*N),:));
       end
       for k=1:adps
           originalVideo(:,:,k)=originalVideo(:,:,k+nwrite);
           labelVideo(:,:,k)=labelVideo(:,:,k+nwrite);
           originalVideoR(:,:,k)=originalVideoR(:,:,k+nwrite);
           originalVideoG(:,:,k)=originalVideoG(:,:,k+nwrite);
           originalVideoB(:,:,k)=originalVideoB(:,:,k+nwrite);
       end
    end
    ninbuf=min(adps,remainedtN);
end % of while

disp('finally write');
for k=1:ninbuf
           temp3(:,1:N,1)=originalVideoR(:,:,k);
           temp3(:,1:N,2)=originalVideoG(:,:,k);
           temp3(:,1:N,3)=originalVideoB(:,:,k);

           temp2=255*labelVideo(:,:,k);
           temp3(:,(N+1):(2*N),1)=temp2;
           temp3(:,(N+1):(2*N),2)=temp2;
           temp3(:,(N+1):(2*N),3)=temp2;
           temp3=uint8(temp3);
           mov2=addframe(mov2,temp3);
           mov3=addframe(mov3,temp3(:,1:N,:));
           mov4=addframe(mov4,temp3(:,(N+1):(2*N),:));
end

mov1=close(mov1);
mov1_do=close(mov1_do);
mov2=close(mov2);
mov3=close(mov3);
mov4=close(mov4);
% fclose(fid);
t=toc
```