Hieu Nguyen
GTID: 903185448
Email: hieu@gatech.edu
11/16/15

CS-6475 Computational Photography Assignment 11

Video textures are slices of video, which seemingly loop forever. Good video textures are seamless and make it difficult to determine where the video ends and the loop begins. This assignment takes in an image sequence and computes the pixel differences between frames. Small dynamics are taken into account and a (hopefully) long, smooth loop is returned using a scoring metric.

This assignment required a lot of trial and error, but I found that the key to getting good video textures is to have a good video volume input. The input should be short in duration, lower resolution, and contain a scene that appears to loop already. This code does not perform well with overly dynamic scenes. And longer durations and higher resolutions lead to significantly longer execution times. The other key to a good video texture is to tune the control parameters. The alpha value can dramatically change the loop start and end frames, so it is important to find the proper value for the particular input. Although I did not experiment with it, adjusting the transition difference matrix with different weighting functions and kernels would also have an effect on the calculated output.

Candle Flame Example

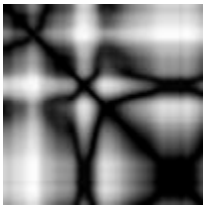
Input (selected frames below): http://imgur.com/p4XD8WC

Here, I tune the alpha parameter from the default value to maximize loop length and improve smoothness. With an alpha=2.5E6, I can get a video texture that is 2 frames longer than the default alpha. With an alpha=14E4, I get a much shorter (and less dynamic) loop, but the flame appears to have smoother motion.

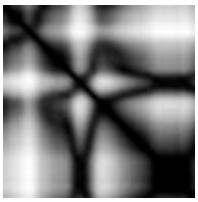
Trial	Alpha Value	Start Frame #	End Frame #	Loop Length (frames)
1	1.5 E6 (default)	37	89	53
2	2.5 E6	41	95	55
3	14.0 E4	77	89	13

Candle difference matrices:

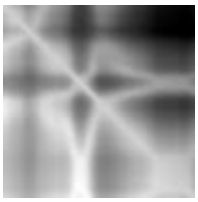
alpha=1.5E6
diff1



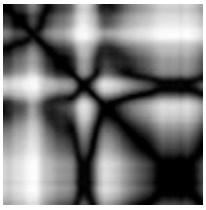
diff2



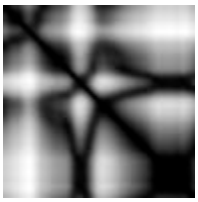
diff3



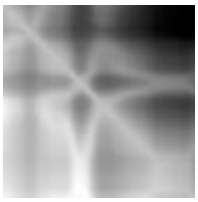
alpha=2.5E6
diff1



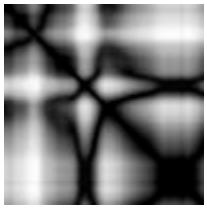
diff2



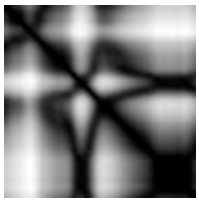
diff3



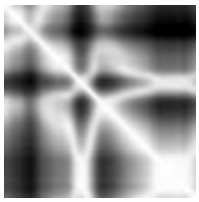
alpha=14E4
diff1



diff3



diff3



Output:

Link to gif album:	http://imgur.com/a/twvqY
alpha=1.5E6:	http://imgur.com/Heoorle
alpha=2.5E6:	http://imgur.com/nUpHHhr
alpha=14E4:	http://imgur.com/XkNOXhh

Slow Motion Face Example

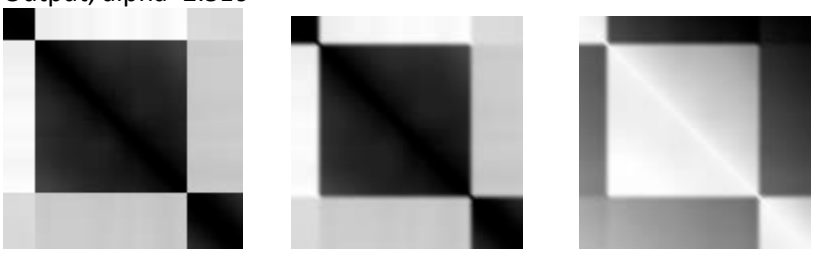
For my own video texture, I decided to try making a seamless gif of a person's face moving in slow motion. I found some source videos on YouTube (https://youtu.be/8iQvZueIBrA and https://youtu.be/UFEtPODcL9Y). I cut them down to have shorter durations, but with enough dynamic content to ensure the video texture code was working properly. This is very evident in the difference matrices. I spend a lot of time with trial and error, and the final output turned out *okay*. It is not as seamless as I had hoped. I believe this is due to limitations of the transitionDifference() function, as large dynamics are properly filtered (like scene changes), but small dynamics (like mouth movements) are not detected very well. I played around with alpha values in order to cut the looped scene properly.

Source: cut YouTube videos from https://youtu.be/8iQvZueIBrA and https://youtu.be/UFEtPODcL9Y

Slow_face1

Input:

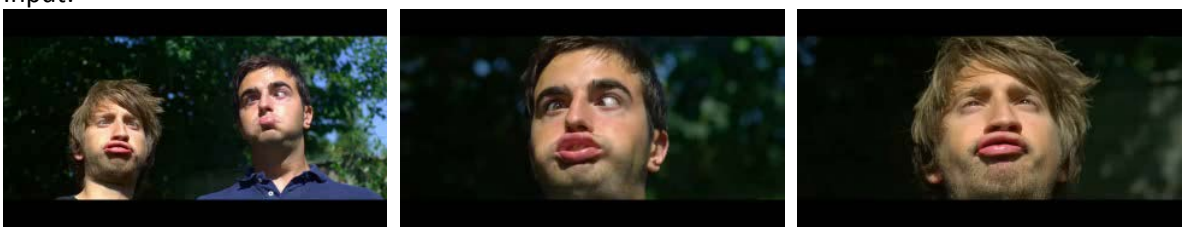
Output, alpha=2.5E6



Gif: http://imgur.com/hbkTQ8k

Slow_face2

Input:

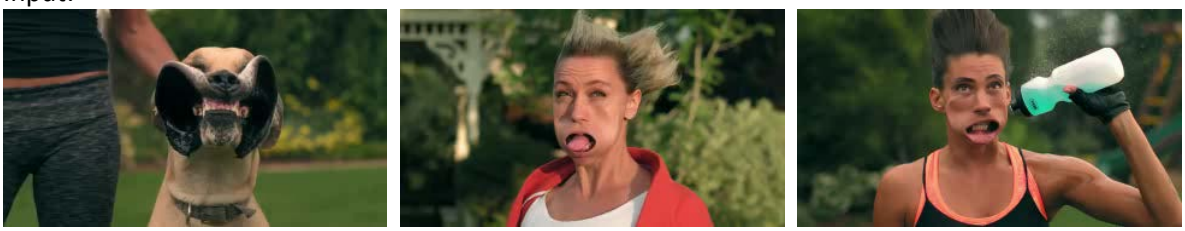


Output, alpha=1.5E6

Gif: http://imgur.com/qpkGRqf

Slow_face3

Input:



Output, alpha=1.5E6

Gif: http://imgur.com/Ddk25j3

Functions

The following functions were implemented for this assignment.

1. videoVolume()

   This function creates a video volume from an input image list. The output is a 4D numpy array with dimensions (num_frames, rows, cols, 3).

```
def videoVolume(images):
    output = np.zeros((len(images), images[0].shape[0], images[0].shape[1],
                       images[0].shape[2]), dtype=np.uint8)
    output = np.array(images)
    return output
```

2. sumSquaredDifferences ()

   This function computes the sum of squared differences for each pair of frames in the input video volume. Because the output matrix is symmetrical (computing the ssd at I,j is identical to computing the ssd at j,i), I can improve performance by avoiding repeat calculations. This effectively speeds up the function by 2. The arrays are converted to dtype=float from uint8 to account for overflow.

```
def sumSquaredDifferences(video_volume):
    output = np.zeros((len(video_volume), len(video_volume)), dtype=np.float)
    # WRITE YOUR CODE HERE.
    for i in xrange(len(video_volume)):
        cur_frame = video_volume[i]
        for j in xrange(len(video_volume)):
            if (i<j):   # Avoid repitition in symmetrical matrix
                comparison_frame = video_volume[j]
                output[i][j] = np.sum((cur_frame.astype(float)
                                       - comparison_frame.astype(float))**2)
                output[j][i] = output[i][j]
    # END OF FUNCTION.
    return output
```

3. transitionDifferencee()

   This function computes the transition cost between frames, taking dynamics into account. It takes in a difference matrix created by ssd, and updates it with dynamic information by considering the preceding and following frames. This can be done efficiently using a 2D convolution with a binomial filter-weighted kernel.

```
def transitionDifference(ssd_difference):
    output = np.zeros((ssd_difference.shape[0] - 4,
                       ssd_difference.shape[1] - 4), dtype=ssd_difference.dtype)

    # Construct 5x5 kernel with binomial filter weights along diagonal
    kernel = np.diag(binomialFilter5())

    # Perform 2D convolution to calculate difference matrix
    output = scipy.signal.convolve2d(ssd_difference, kernel, mode='valid')

    return output
```

4. findBiggestLoop ()

   This function finds the longest and smoothest loop for the video texture, given the difference matrix. It utilizes a scoring metric to tradeoff between loop size and smoothness. A factor, alpha, is used to control this tradeoff; large alphas prefer large loop sizes, and small alphas prefer smoother transitions. To improve performance and repeat calculations, I restrict the column loop variable j to range between i+1 and the image width. This, however, assumes that the input

image has more columns than rows (wider). This shortcut probably wouldn't work effectively on vertical videos.

```
def findBiggestLoop(transition_diff, alpha):
    start = 0
    end = 0
    largest_score = 0
    for i in xrange(transition_diff.shape[0]):
        for j in xrange(i+1, transition_diff.shape[1]):
            score = alpha*(j - i) - transition_diff[j][i]
            if score > largest_score:
                start, end = i, j
                largest_score = score
    return start, end
```

5. synthesizeLoop ()
   This function pulls out the given loop frim the input video volume, given the start and end frame numbers.

```
def synthesizeLoop(video_volume, start, end):
    output = []
    for i in xrange(start, end+1):
        output.append(video_volume[i])
    return output
```