**Assignment 11 - Video Textures**

**Introduction**

In this assignment we will be applying our computational photography magic to video, with the purpose of creating video textures, or infinitely looping pieces of video.

The input and output for the homework assignment is provided as a folder of images. The reasoning behind this, and suggestions for moving between different formats of video are given in the video Appendix in section 4.

The files in this homework are available under Resources > Assignments > Assignment 11 and contain the following files:

* **assignment11.py** - Contains the code you need to finish in order to complete the coding section of the homework assignment.
* **assignment11\_test.py**- Contains code that can test the assignment, and also run sample output.
* **split.py**- Convert a video file into a folder of images (details in appendix)
* **merge.py**- Convert a folder of images into a video file (details in appendix)
* **output (folder)**- Contains sample images for you to run your code on.
* **readme.html**- Contains these instructions.

**Part 0 - videoVolume and sumSquaredDifferences**

**videoVolume**

Hopefully, while implementing this function you will get more familiar with the 4d coordinate system of a video volume (time x row x column x channel). Your task is to take a list containing image numpy arrays, and turn them into a single array which contains the entire video volume.

The lectures, as well as the documentation string in the file contains further detail.

**sumSquaredDifferences**

In this function, you will find an image distance between every pair of frames in the video, as discussed in the lectures. SSD stands for sum of square distances, which as the name suggests, requires you to take the pixelwise difference of the two frames, square it, and sum them all together. Remember when dealing with the values to account for overflow (convert to float arrays first).

**Part 1 - transitionDifference**

**transitionDifference**

This function takes in a difference matrix created by ssd, and updates it with dynamic information.

The intuition behind this is as follows when considering the transition cost from frame i to j, we should not only look at the frames themselves, but also consider the preceding and following frames.

So,

http://www.dcastro.me/omscs6475/assignment11/transitionDiff.png

In other words, we are going to take a weighting function, and sum across the ssd outputs of the preceding and following frames weighed by our function in order to update the transition costs with dynamic information of prior and future frames.

For this assignment, you will be using a binomial filter, which is given to you in the code. For more details about this filter, consult the video textures paper and further references at the end of this document.

The documentation string, and the lectures have further information. When you are finished with the assignment you'll be able to see a visualization of this matrix.

**Part 2 - findBiggestLoop and synthesizeLoop**

**findBiggestLoop**

Now that we have the costs of transitioning from one frame to another, we should find a suitable loop for our video texture. Simply taking the smallest transition distance here might not be desirable what if the resulting loop is only 1 frame long?

In order to state within the code that loop size matters, we will use a trick that is frequent in engineering disciplines.

We are going to find the transition which is optimal under a metric:

http://www.dcastro.me/omscs6475/assignment11/scoreFunction.png

Note the two terms of the metric. The first is the difference between the final and starting frame of our loop. This term is large when the loop is large. The second term is the output of our diff2 function, which tells us the cost of transitioning from finish to start. Subtracting this term turns it into a ‘smoothness’ parameter. It is larger when the transition is less noticeable.

The last bit of compuytation is the alpha parameter. Because the size of the loop and the transition cost are likely to be in very different units, we introduce a new parameter to make them comparable. We can manipulate alpha to control the tradeoff between loop size and smoothness. Large alphas prefer large loop sizes, and small alphas prefer smoother transitions.

Your findBiggestLoop function has to compute this score for every choice of start and end, and return the start and end that corresponds to the largest score.

The documentation string has further details. As with previous score matrixes we will show you a visualization of this matrix after all functions run.

**synthesizeLoop**

The finishing step is to take our video volume, and turn it back into a series of images, now cropping it to only contain the loop we found. This function does just that. It is pretty much the inverse of the video\_volume function you implemented earlier, except for this time you’re starting with a full video volume, and you are returning a list of only the image frames between start and finish (inclusive of the ending index!).

The documentation string contains some more details but this should be fairly straightforward to you at this point. Once you’re finished with this, give yourself a pat on the back (or a hug!). You’re done with the homework assignments!

**Part 3 - Writeup**

This is what we want you to do for the PDF.

1. Demonstrate the output you got on the input images we gave you. Don't include all the images, just a link to the GIF output, and the start and ending index that your function got. Feel free to include your difference matrices too.
2. Demonstrate an example output of your own. Figure out a way to convert it to a GIF and host it on Imgur (just provide the link) or wherever you want to host it as long as we can see it.
3. Discuss which alpha values worked for the video you chose and why. Remember to pick a video that has a repetitive pattern (feel free to record a very short one) so the output makes sense.

**What to turn in:**

Please turn in the following files.**Keep the size limit of each file to 6MB**. If you need to compress your PDF before submitting, you can use <http://smallpdf.com/compress-pdf>:

* **assignment11.py** - Your code.
* **assignment11.pdf** - See above for the writeup, include the images in the PDF (only in the PDF)!

**Appendix - video**

Working with video is still not very user friendly. This is especially so when using free tools, it is very difficult to guarantee that a particular video codec will work on a given system. In order to avoid such issues, this assignments inputs and outputs are sequences of numbered images.

There are tools, discussed in this appendix, that will allow you to split your videos into frames, and put them back together into videos. We cannot guarantee they will work due to versioning systems, etc but hopefully it can help some of you.

**python**

OpenCV for python does support dealing with video, however this support is inconsistent across methods of installation and operating systems. If you have an installation capable of dealing with video, or you want to invest the time to set one up, this week’s assignment comes with two files, ‘split.py’ and ‘merge.py’ that handle the conversion between frame images and video, as their names suggest.

To split a video into frames, run this in terminal:

*python split.py path/to/video.avi*

To merge a folder of images into video, run this in terminal:

*python merge.py path/to/image/folder/*

You may have to change video formats and the output fourcc codec in order to make this work on your system. For details, see:

[OpenCV - Reading and Writing Images and Video](http://www.opencv.org.cn/opencvdoc/2.3.2/html/modules/highgui/doc/reading_and_writing_images_and_video.html?videowriter-videowriter#videowriter)

We are hoping that if you are motivated you will be able to use the install and support pages to use these programs.

**ffmpeg (avconv)**

ffmpeg is available [here](http://www.ffmpeg.org/).

avconv is available [here](https://libav.org/avconv.html).

Some of these calls may need to be updated depending on your versions, etc.

This is a free and very widely used software for dealing with video and audio. Once you have it installed, you can use this command to split your video into frames:

*ffmpeg -i video.ext -r 1 -f image2 image\_directory/%04d.png*

And this command to put them back together:

*ffmpeg -i image\_directory/%04d.png out\_video.gif*

**References**

* [scipy](http://docs.scipy.org/doc/)
* [numpy](http://www.numpy.org/)
* [opencv](http://docs.opencv.org/)
* [scipy.signal.convolve2d](http://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve2d.html)
* [binomial filters](http://www.cse.yorku.ca/~kosta/CompVis_Notes/binomial_filters.pdf)
* [video textures](http://www.cc.gatech.edu/cpl/projects/videotexture/)
* [very important link](http://www.youtube.com/watch?v=O0eEuCxGiAE&feature=youtu.be)