Computational Photography HW4

contents:

0. introduction

In this assignment, we will be applying our computational photography magics 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 as follows:

part0-2.py - contain the code you need to finish in order to complete the homework assignment. Each file also contains test functions for you to debug your code.
run.py - contains code in order to take a video, find the largest loop in that video, and render the loop separately.
submit.py - this file evaluates your code and reports your score to coursera
split.py - convert a video file into a folder of images (details in appendix)
merge.py - convert a folder or images into a video file (details in appendix)

1. part0 - video_volume and SSD

video_volume

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.

ssd

In this function, you will find an image distance between every pair of frames in the video, as discussed in lecture. SSD stands for sum of square distances, which as the name suggests, requires you to take the pixel-wise difference of the two frames, square it, and sum them all together.

The documentation string in the file contain further detail. When you're finished with your code, the function run.py will save a visualization of the difference matrix generated by this function in the video/out folder.

2. part1 - diff2

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,

$$diff2[i,j] \;=\; \sum_{k=-2\rightarrow 2} w_k * ssd(i+k, j+k)$$

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

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, run.py will save a visualization of this matrix in the video/out folder.

3. part0 - find_biggest_loop and synthesize_loop

find_biggest_loop

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:

$$score(s,f) = \alpha * (f - s) - diff2[f,s]$$

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 wizardry 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 find_biggest_loop function has to compute this score for every choice of s and f, and return the s and f that correspond to the largest score.

The documentation string has further details. As with previous score matrixes, run.py will visualize this when you’re finished with the programming assignment.

synthesize_loop

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).

The documentation string contains details. Once you're finished with this, give yourself a pat on the back (or a hug). You're done with the final homework assignment!

4. Appendix - video

As was mentioned in lecture, 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.

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:

```
python split.py path_to_video.avi
```

To merge a folder of images into video, run:

```
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:

http://opencv.willowgarage.com/wiki/documentation/cpp/highgui/VideoWriter
http://opencv.willowgarage.com/wiki/VideoCodecs
http://www.opencv.org.cn/opencvdoc/2.3.2/html/modules/highgui/doc/reading_and_writing_images_and_video.html?videowriter-videowriter#videowriter

animated gif (gifsicle)

Instead of video, another option is to create an animated gif. Many products, like photoshop, are capable of doing this. In this section we will discuss gifsicle, available here:

http://www.lcdf.org/gifsicle/

This is a tool for creating animated gifs. Unfortunately, gifsicle is only compatible with gifs, and opencv is not, so you will have to convert the frame images from png format to gif format. Again, there are lots of tools for doing this in batch, like imagemagick:

http://www.imagemagick.org/script/index.php

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

ffmpeg (avconv)

ffmpeg, available here:

http://www.ffmpeg.org/

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.ext
```

5. references

numpy and scipy - link
numpy user guide - link
opencv - link
scipy.signal.convolve2d - link
binomial filters - link
video textures - link
very important link - link