**Assignment 3: Texture Synthesis**

**Due:** At the **start of class**, Friday, March 1, 2019.

The purpose of this assignment is to implement and understand the texture synthesis approach of Efros and Leung.

**The assignment**

1. First, you should make sure you understand the approach of Efros and Leung from the course notes and textbook. You should also read the original paper, [**“Texture Synthesis by Non-parametric Sampling,” (PDF)**](https://www.cs.ubc.ca/~lsigal/425_2018W2/efrosLeung99.pdf) published in 1999. You also may wish to visit Efros’ [**web site**](http://graphics.cs.cmu.edu/people/efros/research/EfrosLeung.html) that lists more recent research on this topic.
2. To simplify your task of implementing this method we have prepared some Python code that does the more difficult portions, so that you only have to fill in some of the basic matching routines. This code and the sample images are available in a zip file, [**hw3.zip**](https://www.cs.ubc.ca/~lsigal/425_2018W2/hw3.zip). You should download this file to your directory and unzip it with the command
4. unzip hw3.zip

1. The assign directory contains a sample image of a donkey in the file [**donkey.jpg**](https://www.cs.ubc.ca/~lsigal/425_2018W2/images/donkey.jpg). There is also an image showing the region to be filled (shown in black) and the sample texture region (shown as a white rectangle) in the file [**donkey2.jpg**](https://www.cs.ubc.ca/~lsigal/425_2018W2/images/donkey2.jpg).

|  |  |
| --- | --- |
| Donkey | Donkey2 |
| Original image | Region to be texture filled |

1. You can try running the provided skeleton program in Python by typing
3. python Holefill.py

6. at the command prompt. (This runs the script file Holefill.py). The script will pop up two figures: one containing the orginal image and the other showing the region to be filled and the texture region. You will be asked, Are you happy with this choice of fillRegion and textureIm? Type Yes to proceeed. Finally, the script will pop up a figure of the image with the texture filled in. As this skeleton program is missing some critical routines, the texture will initially be filled in as all black.
7. (8 points)

You need to write a function ComputeSSD that computes the sum squared difference (SSD) between an image patch and the texture image, for each possible location of the patch within the texture image. It must ignore empty pixels that have a value of 1 in the given mask image. Skeleton code for this function is provided in Holefill.py.

HINTS: The image patch is called TODOPatch, and it is a square patch of size [2 \* patchL + 1, 2 \* patchL + 1, 3]. Note that the final dimension of 3 means that there are 3 colour values for each pixel. The texture image, textureIm, is of size [texImRows, texImCols, 3]. There is also a mask (an image of 1s and 0s) that specifies which elements in TODOPatch are empty and waiting to be filled in. The mask is called TODOMask and it contains a 1 for each empty pixel and a 0 for each pixel that has a useful value. Its first 2 dimensions are the same as TODOPatch, but it does not have the third dimension. You must ignore the empty pixels when computing the result. Note that the result, ssdIm, will have size [texImRows - 2 \* patchL, texImCols - 2 \* patchL] because it is only defined where the patch completely overlaps the texture. Further note that, as given in Holefill.py, the TODOPatch and textureIm arguments to ComputeSSD have data type unit8. This is not a suitable data type for computing the required SSD. A simple Python trick to coerce a number to floating point is to multiply it by 1.0. Do this, in ComputeSSD, as needed.

1. (6 points)

The next section of Holefill.py takes the SSD image created above and chooses randomly amongst the best matching patches to decide which patch to paste into the texture image at that point. Next you need to write a functon CopyPatch which copies this selected patch into the final image. Remember again that you should only copy pixel values into the hole section of the image. Existing pixel values should not be overwritten. Skeleton code for CopyPatch is provided in Holefill.py, along with comments explaining the arguments.

Note that this technique of copying a whole patch is much faster than copying just the center pixel as suggested in the original Efros and Leung paper. However, the results are not quite as good. We are also ignoring the use of a Gaussian weighted window as described in their paper.

Hand in a printed copy of the donkey image after texture synthesis has been used to remove the donkey.

1. (5 points)

Try running this texture synthesis method on some new images of your choosing. You will need to indicate the area of the removed region in the code. You can load your own image by altering the line that reads donkey.jpg in Holefill.py. You will need to select small regions to avoid long run times.

You can specify the fill and texture regions yourself by typing

python polyselect.py

at the command prompt. The donkey image will be displayed. You draw a polygon by selecting each successive vertex with a mouse click. Close the display window to complete and save the polygon. To try another image, you will have to alter the variable imname around line 90. Regions you specify yourself will be saved for subsequent use as the files fill\_region.pkl and texture\_region.pkl, overwriting existing files with the same name. Note: Be sure to save copies of the original fill\_region.pkland texture\_region.pkl files.

Take a look at the code which randomly selects from the best matching patches. This takes an argument randomPatchSD which is the standard deviation of the number of the patch that gets chosen. If this value is set to 0, then the optimal patch (minimum of the ssd image) is always chosen. If this value is large, then the patch choice will be more random. Experiment with this value and with patchL which defines the size of the synthesis patches.

Hand in texture synthesis results for 2 new images, where one shows the algorithm performing well and the other shows it performing poorly. For each example, show both the original image and the modified one. You do not need to print in colour. Briefly describe why the method failed in the case in which it performed poorly.

1. (6 points)

Provide an explanation for the effects of the randomPatchSD and patchLparameters. What results can be expected if these values are too small or too large, and why do these results happen?

**Deliverables**

Hand your functions (i.e., \*.py files) or a Jupyter Notebook. These must have sufficient comments for others to easily understand the code. You will lose marks for insufficient or unclear comments. In addition, hand in the images and answers requested in a single PDF file. Specifically PDF file should include the following:

* 1. the donkey image after texture synthesis has been used to remove the donkey
  2. the texture synthesis examples for the 2 images of your choosing. (For each example, include both the original image and the modified one.)

Both Python files and PDF should be submitted through Canvas.