Exercise 2

Deadline: 01.05.2013, 16.00

As the 1st of May is a public holiday, mail me your submission by the above date. Please hand in your printed copies by Thursday, May 2nd. My office is in the HCI, 2nd floor, H 2.13.

1 Patch-based texture synthesis (20pt)

In the lecture, we have discussed the paper "Image Quilting for Texture Synthesis and Transfer" by A. Efros and W. Freeman (SIGGRAPH, 2001). In this exercise, we will implement their patch-based texture synthesis method.

Skeleton code can be found on the lecture's webpage in the exercises section. There are two files:

- *textureSynthesis_helpers.template.py*
  Copy this file as `textureSynthesis_helpers.py` into the same directory as `textureSynthesis.py`. Your task is to implement all the stub functions in this file. Once you have done so...
- *textureSynthesis.py*
  ...you can run this file to synthesize textures.

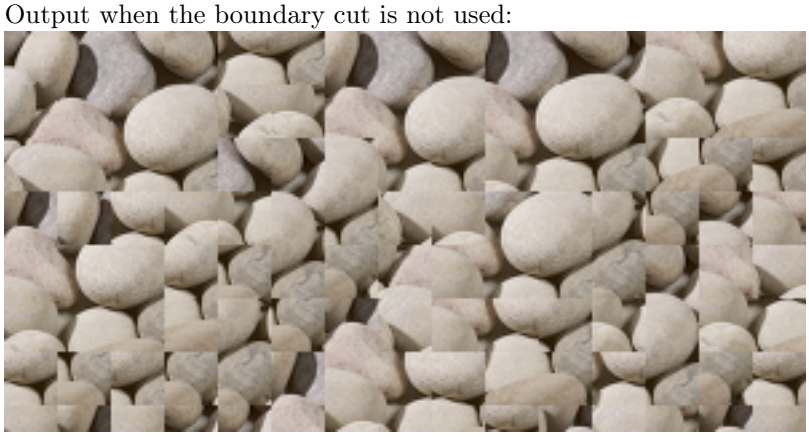
Example output

Input:

Intermediate output (set `debug=True`):

Output:

Output when the boundary cut is not used:



Tasks

- `verticalPathsCosts` (**9pt**)
  Implement the function.
  Plot the result for the example cost map given in `test_paths()`.
  Useful functions: `matplotlib.pyplot.pcolor`.
- `verticalPath_backtracking` (**4pt**)
  Implement the function.
  Plot the resulting path for the example cost map given in `test_paths()`.
- `mkCostMap` (**2pt**)
  Implement the function. Choose two suitable images and plot them together with the associated cost map.
- Run your method on five different textures. Choose representative examples, such that you can demonstrate where the method succeeds and where it fails. You may want to change the patch size and overlap. Briefly comment each output. What property of the method is particularly visible? (**5pt**)

  Implementation hints: Be careful with image sizes. For the example here I've used a 100×100 RGB image as source to generate a 300×150 new texture, which took about 15 seconds with debugging enabled. To speed things up, turn of debugging and work with grayscale images.

Below is a copy of the `textureSynthesis_helpers.template.py` file. Instructions are given in detail in the docstring of each functions that is to be implemented.

```
import numpy

def verticalPathsCosts(costMap):
    """ You are given a 2D costMap with shape costMap.shape
        In the picture below, the cost map's pixels are shown
        with 'x'; it has shape 5x4.

           s
        / | | \
        x x x x
        x x x x
        x x x x
        x x x x
        x x x x
        \ | | /
           t

        As output, we want to have -- for each pixel --
        the cost of the cheapest allowed path from the source 's'
        to the sink 't'.
        Allowed paths are directed paths starting from the source,
        where each extension of an existing path can only go to the
        pixel below, or to the pixels (below, left) or (below,right).

        This is the same setup as in the lecture.

        As transition cost from pixel (i,j) to pixel (k,l), we use
        costMap[k,l].

        Note that the edges connecting s and the image as well as the
        edges connecting the image and t have 0 weight.

        New functions to learn which could be useful:
        numpy.ones, numpy.zeros
    """

    pathCosts = None #initialize array here
    # IMPLEMENT ME HERE
    assert pathCosts.shape == costMap.shape
    return pathCosts

def verticalPath_backtracking(cumulativeCosts):
    """ Given the cost for allowed paths from 's' to each pixel
        in the array cumulativeCosts, find the
        minimum costs path from 't' to 's'.

        To do this, use backtracking as described in the lecture.

        Return an array with data type dtype=numpy.uint8
        where each pixel that is traversed by the minimum cost path
        is set to 1, while all other pixels are set to zero.

        New functions to learn which could be useful:
        numpy.argmin, numpy.argmax
    """

    pathImg = None #initialize array here
    # IMPLEMENT ME HERE
    assert pathImg.shape == cumulativeCosts.shape
    assert pathImg.dtype == numpy.uint8
    return pathImg
def mkCostMap(img1, img2):
    """Given two images, compute the pixel-wise L2 norm of the
       difference image.

       Note: this is a one-liner using numpy!
```

```
    New functions to learn which could be useful:
    http://docs.scipy.org/doc/numpy/reference/routines.math.html
"""

assert img1.shape == img2.shape
assert img1.dtype == numpy.float32
assert img2.dtype == numpy.float32
assert img1.ndim == 3 and img2.ndim == 3

pass #IMPLEMENT ME (in one line, please)
```