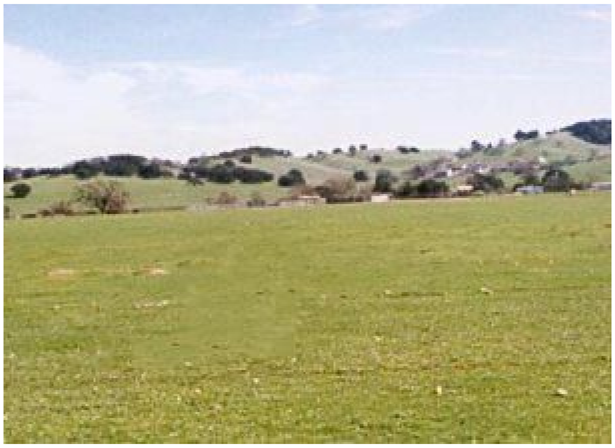
Assignment 3 Texture Synthesis

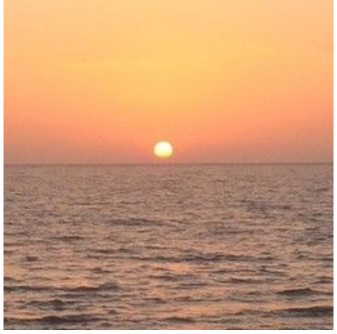
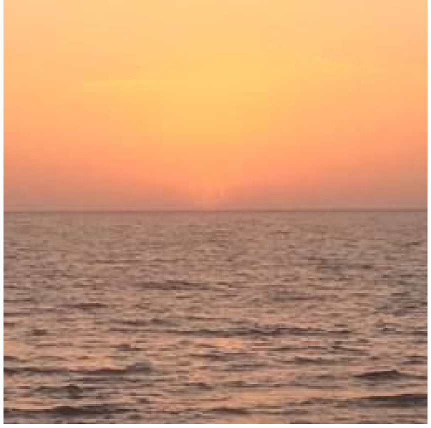
Q4 & Q5

Result image:

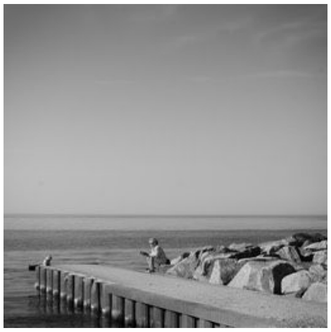
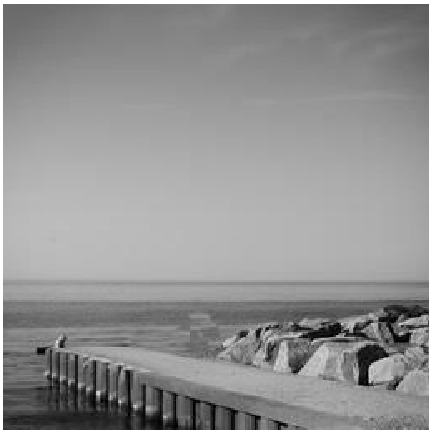


Q6

Perform well example:

Perform badly: texture chosen is too small, so lack of good matches that found in the texture to fill the empty region

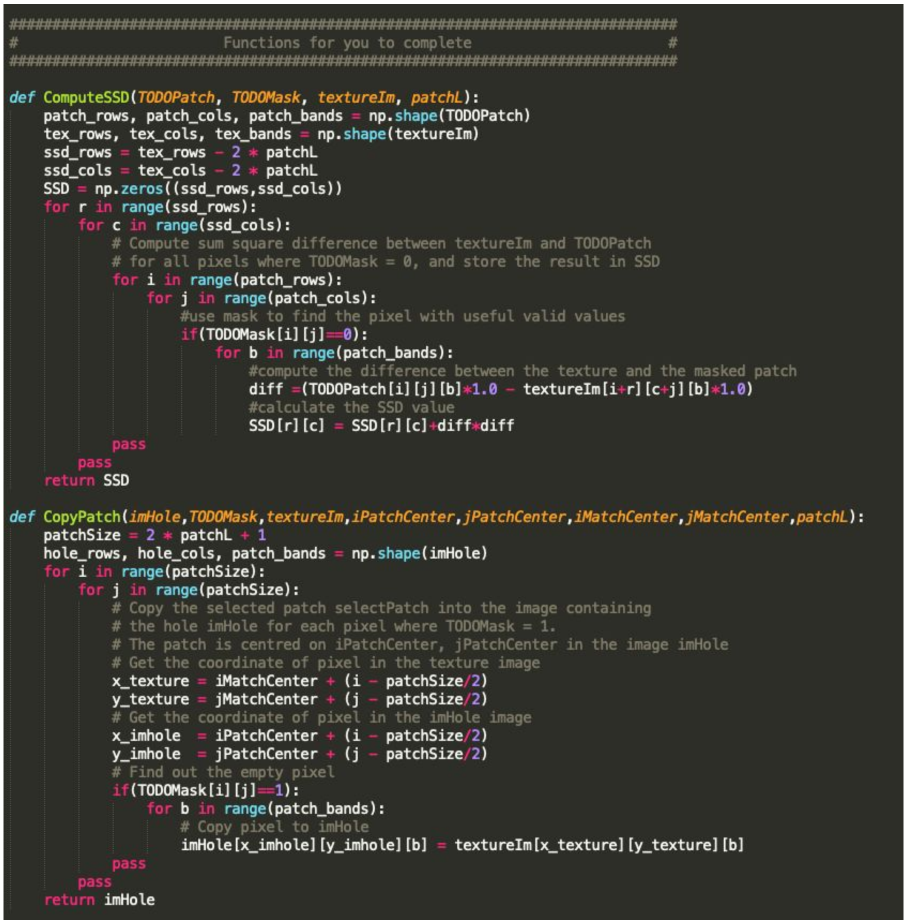



Q7

randomPatchSD is used to randomly select the patch that would be used to do the filling. By increased the value of the randomPatchSD, we are increasing the degree of randomization for selecting patches, which means that the accuracy of selected patches from the sample texture may decrease, and would possibly lead to unrealistic textures. On the other hand, if we decrease the ramdomPatchSD, we are decreasing the randomness of selecting patches. Which means that the patch that selected my be a very uniform or unnatural fill.

patchL is the parameter that used to define the patch size. Bigger patch means there will be more matches and also bigger possibility to get the complete or expected pattern. But if the patchL gets too big, we may probably would have too many matches, so when it comes to selection, the result we get may not be as good as when patchL is about the right size. If the patchL becomes too small, it's also hard for us to get good matches since the texture image may not big enough to cover the whole pattern.

Full version of code added



```
################################################################################
#                          Functions for you to complete                       #
################################################################################

def ComputeSSD(TODOPatch, TODOMask, textureIm, patchL):
    patch_rows, patch_cols, patch_bands = np.shape(TODOPatch)
    tex_rows, tex_cols, tex_bands = np.shape(textureIm)
    ssd_rows = tex_rows - 2 * patchL
    ssd_cols = tex_cols - 2 * patchL
    SSD = np.zeros((ssd_rows,ssd_cols))
    for r in range(ssd_rows):
        for c in range(ssd_cols):
            # Compute sum square difference between textureIm and TODOPatch
            # for all pixels where TODOMask = 0, and store the result in SSD
            for i in range(patch_rows):
                for j in range(patch_cols):
                    #use mask to find the pixel with useful valid values
                    if(TODOMask[i][j]==0):
                        for b in range(patch_bands):
                            #compute the difference between the texture and the masked patch
                            diff =(TODOPatch[i][j][b]*1.0 - textureIm[i+r][c+j][b]*1.0)
                            #calculate the SSD value
                            SSD[r][c] = SSD[r][c]+diff*diff
            pass
        pass
    return SSD

def CopyPatch(imHole,TODOMask,textureIm,iPatchCenter,jPatchCenter,iMatchCenter,jMatchCenter,patchL):
    patchSize = 2 * patchL + 1
    hole_rows, hole_cols, patch_bands = np.shape(imHole)
    for i in range(patchSize):
        for j in range(patchSize):
            # Copy the selected patch selectPatch into the image containing
            # the hole imHole for each pixel where TODOMask = 1.
            # The patch is centred on iPatchCenter, jPatchCenter in the image imHole
            # Get the coordinate of pixel in the texture image
            x_texture = iMatchCenter + (i - patchSize/2)
            y_texture = jMatchCenter + (j - patchSize/2)
            # Get the coordinate of pixel in the imHole image
            x_imhole  = iPatchCenter + (i - patchSize/2)
            y_imhole  = jPatchCenter + (j - patchSize/2)
            # Find out the empty pixel
            if(TODOMask[i][j]==1):
                for b in range(patch_bands):
                    # Copy pixel to imHole
                    imHole[x_imhole][y_imhole][b] = textureIm[x_texture][y_texture][b]
            pass
        pass
    return imHole
```