Programming Project #2: Image Quilting

CS445: Computational Photography - Fall 2020

In [1]:

```
# from google.colab import drive
# drive.mount('/content/drive')


###NOTICE
##In part 3, "else" condition used another, "better" method to apply
# the mask to the picture, which is not seen in the first 3 conditions
#further optimization would be:
## apply "else" method to "row" and "col" part
```

In [2]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import os
from random import random
import random
import time
import utils

# modify to where you store your project data including utils.py
# datadir = "/content/drive/My Drive/cs445_projects/proj2/"

# utilfn = datadir + "utils.py"
# !cp "$utilfn" .
# samplesfn = datadir + "samples"
# !cp -r "$samplesfn" .
# import utils
```

In [3]:

```
from utils import cut # default cut function for seam finding section
```

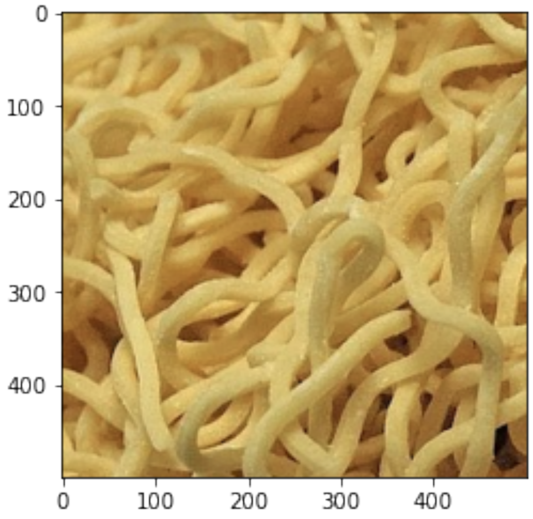
Part I: Randomly Sampled Texture (10 pts)

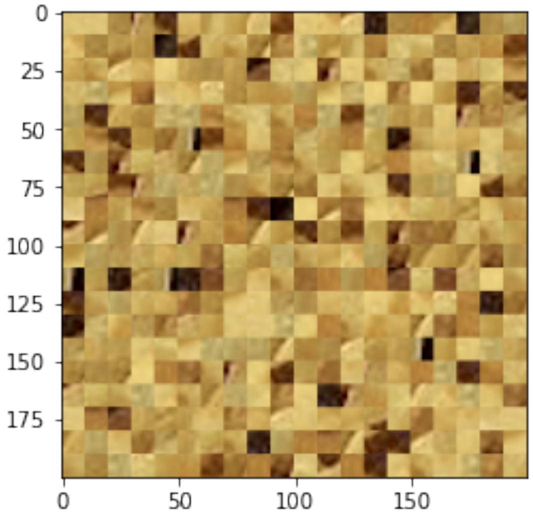
In [4]:

```
def quilt_random(sample, out_size, patch_size):
    """
    Randomly samples square patches of size patchsize from sample in o
    to create an output image of size outsize.

    :param sample: numpy.ndarray   The image you read from sample dire
    :param out_size: int            The width of the square output ima
    :param patch_size: int          The width of the square sample pat
    :return: numpy.ndarray
    """
    output = np.zeros((out_size,out_size,3), dtype=np.float32)
    number = out_size // patch_size
    r_max = sample.shape[0] - patch_size
#     print(sample.shape[0])
    r_min = 0
#     print(r_max, r_min)
#     print(rand)
#     print("numbter is,", number) #num is 5
    for i in range(number):
        for j in range(number):
            rand_start = (int)(random.randrange(r_min,r_max))
            new_x1 = rand_start
            new_x2 = rand_start + patch_size
            new_y1 = new_x1
            new_y2 = new_x2
            test_block = np.array(sample[new_y1:new_y2, new_x1:new_x2]
#             print(test_block.shape)
            #put the test_block in to its position
            x1 = j *patch_size
            y1 = i *patch_size
            x2= x1 + patch_size
            y2= y1 + patch_size
            #notice, confused about xy position. Check it later
            output[y1:y2, x1:x2] = test_block
#             print("x1, y1", x1, y1)
#             print("x2, y2", x2, y2)
    return output
```

In [5]:

```
sample_img_fn = 'samples/noodle_500.jpg'
sample_img = cv2.cvtColor(cv2.imread(sample_img_fn), cv2.COLOR_BGR2RGB
plt.imshow(sample_img) #(192, 192, 3)
plt.show()

out_size = 200  # change these parameters as needed
patch_size = 10
res = quilt_random(sample_img, out_size, patch_size)

#the dtype is wrong. fix it by changing the dtype to int32(originally
#https://stackoverflow.com/questions/49643907/clipping-input-data-to-t
res = np.array(res, np.int32)
plt.imshow(res)
plt.show()
# if res is not None:
#     plt.imshow(res)
```





Part II: Overlapping Patches (30 pts)

In [6]:

```
def getOnePatch(sample_img, patch_size):
    r_max = sample_img.shape[0] - patch_size
    r_min = 0
    rand_start = (int)(random.randrange(r_min,r_max))
    new_x1 = rand_start
    new_x2 = rand_start + patch_size
    new_y1 = new_x1
    new_y2 = new_x2
    rand_patch = np.array(sample_img[new_y1:new_y2, new_x1:new_x2])
    return rand_patch
```

In [7]:

```
#fix this section
# ssd_cost = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel
# + cv2.filter2D(I ** 2, ddepth=-1, kernel=M)

#  Each pixel of the ssd_cost gives you the cost for sampling a patch
#input sample picture,

# ssd_patch performs template matching with the overlapping region, co
# of sampling each patch, based on the sum of squared differences (SSD
# regions of the existing and sampled patch

def ssd_patch(I, T, M, patch_size):
#     I = np.array(I, np.float32)
# I, T, M, should all be in float32.
#     print(I.dtype, T.dtype, M.dtype)
    ssd_cost_raw = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, k
    sample_size = I.shape[0]
    ssd_cost_result = np.zeros((sample_size,sample_size), dtype=np.flo
    start_index = (patch_size) // 2

    end_index = sample_size - start_index
    for i in range(start_index, end_index + 1):
        for j in range(start_index, end_index + 1):
#             print(ssd_cost_raw[i, j])
            ssd_cost_result[i,j] = ssd_cost_raw[i, j].sum()
#             print(ssd_cost_raw[i, j].sum())
#             print(ssd_cost_raw.dtype)


#     print("start_index, end_index", start_index, end_index)
#     return ssd_cost_raw
#     print(ssd_cost_result.dtype)
    return ssd_cost_result
#ssd_cost_result contains a matrix, size of sample, one channel. each
#sum of surronding ssd. THE FIRST AND LAST ROW && COL IS 0, SIZE OF PA
```

In [8]:

```
import heapq

# choose_sample should take as input a cost image
# (each pixel's value is the cost of selecting the patch centered
#  at that pixel) and select a randomly sampled patch with low cost.
# It's recommended to sort the costs and choose of of the tol smallest
# So if tol=1, the lowest cost will always be chosen (this is a good w
# but mainly copies the input texture). If tol=3, one of the three low
# will be chosen.
def choose_sample(sample, patch_size, cost_image, tol):
    #input:
    #output: choose one KEY from list of 'tol' number of smallest (key
    #value is (i,j), center of balabala
    min_cost = float("inf")
    store_tol_pairs = []
    result_list = []
#     cost_image.flatten()
#     print(cost_image.shape)

    start_index = patch_size // 2
    sample_size = sample.shape[0]
    end_index = sample_size - start_index

#     print(start_index,end_index )
    #maintain a PQ, from small val to large val. Then pop the largest
    #key is the cost_value, and value is [i, j] position
    #will occupy a huge memory. probably
    for i in range (start_index,end_index+1):
        for j in range (start_index,end_index+1):
            heapq.heappush(store_tol_pairs, (cost_image[i,j], (i,j)))
#             if cost_image[i,j] < min_cost and cost_image[i,j] != 0:
#                 min_cost = cost_image[i,j]

    for x in range (tol):
        result_list.append(heapq.heappop(store_tol_pairs))

    pick_random_number = (int)(random.randrange(0,tol))

    return result_list[pick_random_number][1]#randomly return one posi
```

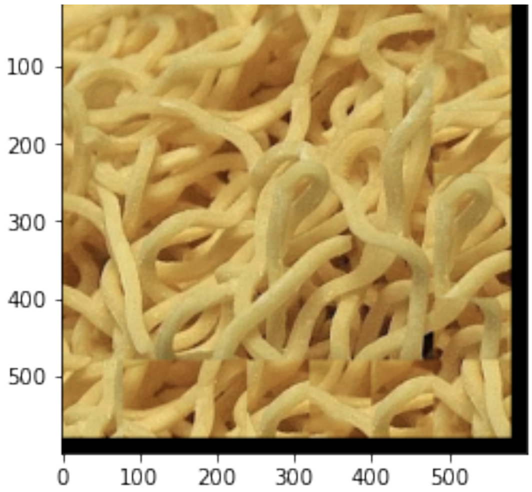
In [9]:

```
def quilt_simple(sample, out_size, patch_size, overlap, tol):
    """
    Randomly samples square patches of size patchsize from sample in o
    Feel free to add function parameters
    :param sample: numpy.ndarray
    :param out_size: int
    :param patch_size: int
    :param overlap: int
    :param tol: int
    :return: numpy.ndarray
    """

    sample = np.array(sample, np.float32)#cast from unit8 to float32
    output = np.zeros((out_size,out_size,3), dtype=np.float32)


    loop_number = (out_size - patch_size) // (patch_size - overlap) +
    #need to loop this much on cols and rows
    #you can move len(patch_size - overlap) each time


#     mask[:,0:overlap] = 1.0 #左边一条
#     mask[0:overlap,:] = 1.0 #上边一条


# ###     ssd_patch(I, T, M, patch_size)
    for i in range(loop_number): #i related to y
        for j in range(loop_number): #j related to x
            print("running")
            mask = np.zeros((patch_size,patch_size,3), dtype=np.float3
            if(i == 0 and j == 0): #top left, randomly assign a patch
#                 print("top left")
                first_patch = getOnePatch(sample_img, patch_size)
                first_patch = np.array(first_patch, np.float32)
                output[0:patch_size,0:patch_size] = first_patch
# output[0:patch_size,0:patch_size] = sample[a:a+patch_size,b:b+patch_
#                 show_first_patch = np.array(first_patch, np.int32)
#                 show_output = np.array(output, np.int32)
#                 plt.imshow(show_output)
#                 plt.show()
            elif(i == 0):#first row, | shape mask
#                 print("first row", (i, j))
                mask[:,0:overlap] = 1.0
            elif(j == 0): #first col, - shape mask
#                 print("first col", (i, j))
                mask[0:overlap,:] = 1.0
            else:
                mask[0:overlap,:] = 1.0
                mask[:,0:overlap] = 1.0
#                 print("other")

            #make template
            each_hop = patch_size - overlap
            current_y = each_hop * i
            current_x = each_hop * j
#             print("current_x,current_y", (current_x,current_y))
            template = output[current_y:current_y + patch_size, curren

     #######apply mask, template, image to SSD#######
#             ssd_patch(I, T, M, patch_size)
            cost_SSD = ssd_patch(sample, template, mask, patch_size)

    ########get the random position########
    ########get the random position########
#             choose_sample(sample, patch_size, cost_image, tol)
            chosen_position = choose_sample(sample, patch_size, cost_S
#             print("chosen_position", chosen_position)

     ########put it in the OUTPUT picture########
            chosen_y = chosen_position[0]
            chosen_x = chosen_position[1]
            half_ps = patch_size // 2
#             print(half_ps)
            output[current_y:current_y + patch_size, current_x:current
            sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x-half_


#             output[current_y:current_y + patch_size, current_x:curre
#             sample[chosen_x-half_ps : chosen_x+half_ps, chosen_y-hal

#             output[current_x:current_x + patch_size, current_y:curre
#             sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x-hal

#             print(chosen_x-half_ps, chosen_x+half_ps)
#             print(current_x,current_x + patch_size)
    return output
```

In [10]:

```
sample_img_fn = 'samples/noodle_500.jpg'
sample_img = cv2.cvtColor(cv2.imread(sample_img_fn), cv2.COLOR_BGR2RGB
# plt.imshow(sample_img)
# plt.show()
overlap = 20
tol = 3
out_size = 600  # change these parameters as needed
patch_size = 100
# print(sample_img.shape)(192, 192, 3)

# rand_block = getOnePatch(sample_img, patch_size)

# 三维矩阵y[i,j,m]的参数理解:
# 第一维 i: 确定是哪一个二维矩阵
# 第二维 j: 每一个二维矩阵的行
# 第三维 m: 每一个二维矩阵的列
# test = np.array([[[1,1,1]], [[1,1,1]]])#(2, 1, 3)
# print(test[1][0].sum()) #3

res = quilt_simple(sample_img, out_size, patch_size, overlap, tol) #fe
res_pic = np.array(res, np.int32)
print(res_pic.shape)
plt.imshow(res_pic)

# plt.savefig('./output_res/p2_brick2.jpg')
plt.show()
# if res is not None:
#     plt.figure(figsize=(10,10))
#     plt.imshow(res)
#     plt.imshow(res)
```

```
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
running
(600, 600, 3)
```



Part III: Seam Finding (20 pts)

In [11]:

```python
# optional or use cut(err_patch) directly
def customized_cut(bndcost):
    pass
```

In [12]:

```
def calculate_ssd(previous, current, patch_size, overlap, overlap_type
#To find a vertical path, you can apply cut to the transposed patch, e
    """
    previous: previous patchnew_one
    current: the new patch added to the graph

    return: the ssd of two patches
    """
    res_x = previous.shape[1]
    res_y = previous.shape[0]
    if (overlap_type == "rows"):
        right_of_p = previous[0:patch_size, patch_size-overlap:patch_s
        left_of_c = current[0:patch_size, 0:overlap]
        result_diff = np.zeros((patch_size,overlap),dtype=np.float32)
        err_diff = np.square(right_of_p - left_of_c)
        for i in range(patch_size):
            for j in range(overlap):
                result_diff[i,j] = err_diff[i][j].sum()


    elif(overlap_type == "cols"):
#         print("in the function, cols")
        bottom_of_p = previous[patch_size-overlap:patch_size, 0:patch_
        top_of_c = current[0:overlap, 0:patch_size]
        result_diff = np.zeros((overlap,patch_size),dtype=np.float32)
        err_diff = np.square(bottom_of_p - top_of_c)
        for i in range(overlap):
            for j in range(patch_size):
                result_diff[i,j] = err_diff[i][j].sum()
        result_diff = result_diff.T


    return result_diff
    #the result feed cut function
```

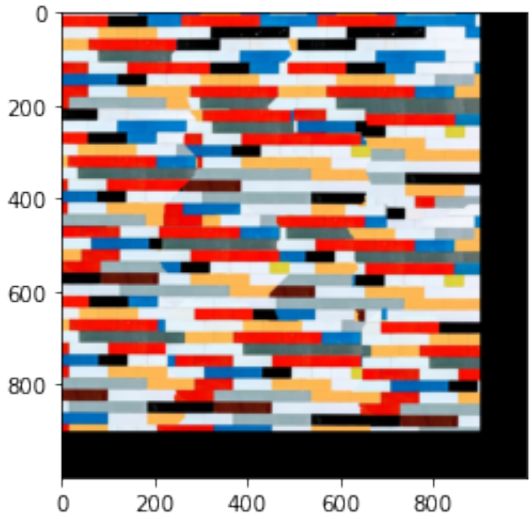
In [13]:

```
def quilt_cut(sample, out_size, patch_size, overlap, tol):
    """
    Samples square patches of size patchsize from sample using seam fi
    Feel free to add function parameters
    :param sample: numpy.ndarray
    :param out_size: int
    :param patch_size: int
    :param overlap: int
    :param tol: float
    :return: numpy.ndarray
    """
    sample = np.array(sample, np.float32)#cast from unit8 to float32
    output = np.zeros((out_size,out_size,3), dtype=np.float32)
    output = np.zeros((out_size,out_size,3), dtype=np.float32)
    loop_number = (out_size - patch_size) // (patch_size - overlap) +

#     mask[:,0:overlap] = 1.0 #左边一条
#     mask[0:overlap,:] = 1.0 #上边一条

    for i in range(loop_number): #i related to y
        for j in range(loop_number): #j related to x
            mask = np.zeros((patch_size,patch_size,3), dtype=np.float3

            if(i == 0 and j == 0): #top left, randomly assign a patch
                first_patch = getOnePatch(sample_img, patch_size)
                first_patch = np.array(first_patch, np.float32)
                output[0:patch_size,0:patch_size] = first_patch
                continue
            elif(i == 0):#first row, | shape mask
                mask[:,0:overlap] = 1.0
            elif(j == 0): #first col, - shape mask
                mask[0:overlap,:] = 1.0
            else:
                mask[0:overlap,:] = 1.0
                mask[:,0:overlap] = 1.0
            each_hop = patch_size - overlap
            current_y = each_hop * i
            current_x = each_hop * j
            template = output[current_y:current_y + patch_size, curren
     ########apply mask, template, image to SSD########
            cost_SSD = ssd_patch(sample, template, mask, patch_size)
    ########get the random position########
            chosen_position = choose_sample(sample, patch_size, cost_S
     ########put it in the OUTPUT picture########
            chosen_x = chosen_position[1]
            chosen_y = chosen_position[0]
            half_ps = patch_size // 2
            chosen_new_patch = sample[chosen_y-half_ps : chosen_y+half
            ##################################
    ##########deal with overlap cost##########
    ##################################
            if  i == 0 and j  == 0:
                continue
            elif i == 0:
                print("row, continue")
                old_y = each_hop * i
                old_x = each_hop * (j -1)
#                 chosen_new_patch = sample[chosen_y-half_ps : chosen_
                old_patch = output[old_y:old_y + patch_size, old_x:old
                right_of_p = old_patch[0:patch_size, patch_size-overla
                left_of_c = chosen_new_patch[0:patch_size, 0:overlap]
                patch_ssd = calculate_ssd(old_patch, chosen_new_patch,


                use_cut_result = cut(patch_ssd.T).T


                left_mask = np.zeros((patch_size, overlap, 3), dtype =
                left_mask = np.zeros((patch_size, overlap, 3), dtype
                for a in range(patch_size):
                    for b in range(overlap):
                        if (use_cut_result[a][b] == 0):
                            for c in range (3):
                                left_mask[a][b][c] = 1.0
                left_right_mix = right_of_p*left_mask + left_of_c*(1 -

                output[current_y:current_y + patch_size, current_x:cur
                sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x-h
                ##overlap for right_left
                output[current_y:current_y + patch_size,current_x:curr
            elif j == 0:
                print("col, continue")
                old_y = each_hop * (i-1)
                old_x = each_hop * j
#                 chosen_new_patch = sample[chosen_y-half_ps : chosen_
                old_patch = output[old_y:old_y + patch_size, old_x:old
                bottom_of_p = old_patch[patch_size - overlap:patch_siz
                top_of_c = chosen_new_patch[0:overlap, 0:patch_size]

                patch_ssd = calculate_ssd(old_patch, chosen_new_patch,
                use_cut_result = cut(patch_ssd.T)#should be a "-" shap
#                 print("patch_ssd," patch_ssd.shape)
#                 print("in col")

                top_mask = np.zeros((overlap, patch_size, 3), dtype =

                for a in range(overlap):
                    for b in range(patch_size):
                        if (use_cut_result[a][b] == 1):
                            for c in range (3):
                                top_mask[a][b][c] = 1.0
# #                   print("left_mask,", left_mask.shape)
# #                   right_mask = np.zeros((patch_size, patch_size, 3),
                bottom_top_mix = bottom_of_p*(1-top_mask) + top_of_c*(


                output[current_y:current_y + patch_size, current_x:cur
                sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x-h

                ##overlap for bottom_top
                output[current_y:current_y + overlap,current_x:current

            else:
                print("else, running")

                #left right old patch
                old_y = each_hop * i
                old_x = each_hop * (j -1)
#                 chosen_new_patch = sample[chosen_y-half_ps : chosen_
                old_patch_l_r = output[old_y:old_y + patch_size, old_x
                right_of_p = old_patch_l_r[0:patch_size, patch_size-ov
                left_of_c = chosen_new_patch[0:patch_size, 0:overlap]
                patch_ssd_left_right = calculate_ssd(old_patch_l_r, ch
                use_cut_result = cut(patch_ssd_left_right.T).T
                left_mask = np.zeros((patch_size, overlap, 3), dtype =
                for a in range(patch_size):
                    for b in range(overlap):
                        if (use_cut_result[a][b] == 0):
                            for c in range (3):
                                left_mask[a][b][c] = 1.0
#                 left_right_mix = right_of_p*left_mask + left_of_c*(1


                ########################
                ###########OVERLAP SHOWCASE###########
#                 show_output = np.array(left_right_mix, np.int32)
#                 plt.imshow(show_output)
#                 plt.show()
#                 plt.imshow(left_mask)
#                 plt.show()


#                 output[current_y:current_y + patch_size, current_x:c
#                 sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x

#                 output[current_y:current_y + patch_size,current_x:cu

                #bottom top old patch
                old_y_2 = each_hop * (i-1)
                old_x_2 = each_hop * j

                old_patch_t_b = output[old_y_2:old_y_2 + patch_size, o
                bottom_of_p = old_patch_t_b[patch_size - overlap:patch
                top_of_c = chosen_new_patch[0:overlap, 0:patch_size]
                patch_ssd_bottom_top = calculate_ssd(old_patch_t_b, ch
                use_cut_result_2 = cut(patch_ssd_bottom_top.T)#should
# #                 print("patch_ssd," patch_ssd.shape)
# #                 print("in col")

                top_mask = np.zeros((overlap, patch_size, 3), dtype =

                for a in range(overlap):
                    for b in range(patch_size):
                        if (use_cut_result_2[a][b] == 1):
                            for c in range (3):
                                top_mask[a][b][c] = 1.0

# # #                 right_mask = np.zeros((patch_size, patch_size, 3
#                 bottom_top_mix = bottom_of_p*(1-top_mask) + top_of_c

                two_mask_together = np.zeros((patch_size, patch_size,

                #mark non-overlap region 1
                for a in range(overlap, patch_size):
                    for b in range(overlap, patch_size):
                        for c in range(3):
                            two_mask_together[a][b][c] = 1
                for a in range(0, overlap):
                    for b in range(overlap, patch_size):
                        for c in range(3):
                            if (top_mask[a,b,c] == 1):
                                two_mask_together[a][b][c] = 1

                for a in range(overlap, patch_size):
                    for b in range(0, overlap):
                        for c in range(3):
                            if (left_mask[a,b,c] == 0):
                                two_mask_together[a][b][c] = 1
                #mark overlaped-overlap region one. Use bitwise and(&)
                for a in range(0, overlap):
                    for b in range(0, overlap):
                        for c in range(3):
                            if (left_mask[a,b,c] == 0 and top_mask[a,b
                                two_mask_together[a][b][c] = 1
#                 print(two_mask_together.shape)
#                 two_mask_together = left_mask & top_mask

#                 plt.imshow(left_mask)
#                 plt.show()
# #                 show_output2 = np.array(complement_patch, np.int32
# #                 plt.imshow(show_output2)
# #                 plt.show()

#                 plt.imshow(top_mask)
#                 plt.show()
# #                 show_output2 = np.array(complement_patch, np.int32
# #                 plt.imshow(show_output2)
# #                 plt.show()



#                 plt.imshow(two_mask_together)
#                 plt.show()




                two_mask_together_patch = two_mask_together * chosen_n

#                 show_output2 = np.array(two_mask_together_patch, np.
#                 plt.imshow(show_output2)
#                 plt.show()
                ########################
                ############OVERLAP SHOWCASE############
#                 show_output2 = np.array(bottom_top_mix, np.int32)
#                 plt.imshow(show_output2)
#                 plt.show()
#                 plt.imshow(top_mask)
#                 plt.show()


#                 output[current_y:current_y + patch_size, current_x:c
#                 sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x


                complement_patch = (1 - two_mask_together) * output[cu

                output[current_y:current_y + patch_size, current_x:cur
                two_mask_together_patch + complement_patch





# 三维矩阵y[i,j,m]的参数理解:
# 第一维 i: 确定是哪一个二维矩阵
# 第二维 j: 每一个二维矩阵的行
# 第三维 m: 每一个二维矩阵的列


    return output
```

In [133]:

```
sample_img_fn = 'samples/pattern3_500.jpg'
sample_img = cv2.cvtColor(cv2.imread(sample_img_fn), cv2.COLOR_BGR2RGB
# plt.imshow(sample_img)
# plt.show()

out_size = 1000  # change these parameters as needed
patch_size = 300
overlap = 100
tol = 3

res = quilt_cut(sample_img, out_size, patch_size, overlap, tol)
show_output = np.array(res, np.int32)
plt.imshow(show_output)
# plt.savefig('./output_res/p3_pattern2_2.jpg')
plt.show()
```

```
row, continue
row, continue
row, continue
col, continue
else, running
else, running
else, running
col, continue
else, running
else, running
else, running
col, continue
else, running
else, running
else, running
```



part IV: Texture Transfer (30 pts)

In [18]:

```
def ssd_transfer(input_sample, guidance_patch, patch_size):
#     SSD_guidance is the SSD between the input sample and
#     the patch in the guidance/correspondence image at the
#     same position as the output patch.


#     I = np.array(I, np.float32)
# I, T, M, should all be in float32.
#     print(I.dtype, T.dtype, M.dtype)
    sample_size = I.shape[0]
    ssd_cost_result = np.zeros((patch_size,patch_size), dtype=np.float

    for i in range(0, patch_size):
        for j in range(0, patch_size):
            ssd_cost_result[i,j] = input_sample[i][j] - guidance_patch
            ssd_cost_result[i,j] = ssd_cost_result[i,j].sum()

#     print("start_index, end_index", start_index, end_index)
#     return ssd_cost_raw
#     print(ssd_cost_result.dtype)
    return ssd_cost_result
#ssd_cost_result contains a matrix, size of sample, one channel. each
#sum of surronding ssd. THE FIRST AND LAST ROW && COL IS 0, SIZE OF PA
```

In [63]:

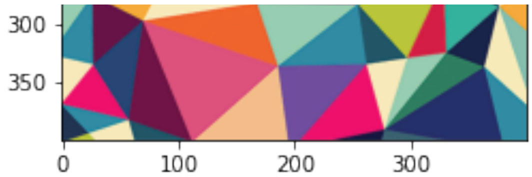
```
def texture_transfer(sample, patch_size, overlap, tol, guidance_im, al
# def quilt_cut(sample, out_size, patch_size, overlap, tol):
    """
    Samples square patches of size patchsize from sample using seam fi
    Feel free to modify function parameters
    :param sample: numpy.ndarray
    :param patch_size: int
    :param overlap: int
    :param tol: float
    :param guidance_im: target overall appearance for the output
    :param alpha: float 0-1 for strength of target
    :return: numpy.ndarray
    """
    sample = np.array(sample, np.float32)#cast from unit8 to float32
    out_size = guidance_im.shape[0]
    print("out_size", out_size)
    output = np.zeros((out_size,out_size,3), dtype=np.float32)
    loop_number = (out_size - patch_size) // (patch_size - overlap) +
    print("loop_number", loop_number)
#     mask[:,0:overlap] = 1.0 #左边一条
#     mask[0:overlap,:] = 1.0 #上边一条

    for i in range(loop_number): #i related to y
        for j in range(loop_number): #j related to x
            mask = np.zeros((patch_size,patch_size,3), dtype=np.float3

            if(i == 0 and j == 0): #top left, randomly assign a patch
```

```
                first_patch = getOnePatch(sample_img, patch_size)
                first_patch = np.array(first_patch, np.float32)
                output[0:patch_size,0:patch_size] = first_patch
                continue
            elif(i == 0):#first row, | shape mask
                mask[:,0:overlap] = 1.0
            elif(j == 0): #first col, - shape mask
                mask[0:overlap,:] = 1.0
            else:
                mask[0:overlap,:] = 1.0
                mask[:,0:overlap] = 1.0
            each_hop = patch_size - overlap
            current_y = each_hop * i
            current_x = each_hop * j
            template = output[current_y:current_y + patch_size, curren
     #######apply mask, template, image to SSD########
    ##改这里 alpha*ssd_overlap + (1-alpha)*ssd_transfer###
    ##############################################
    ##############################################


#             cost_SSD = ssd_patch(sample, template, mask, patch_size)

            cost_SSD_overlap = ssd_patch(sample, template, mask, patch

    #template不一样了 变成脸图里面对应的位置了
    #mask呢?
            template2 = guidance_im[current_y:current_y + patch_size,
#             template2 =
#             cost_SSD_transfer =ssd_transfer(input_sample, guidance_p
            mask2 = np.zeros((patch_size,patch_size,3), dtype=np.float
            for m in range(patch_size):
                for n in range(patch_size):
                    mask2[m][n] = 1
            cost_SSD_transfer = ssd_patch(sample, template2, mask2, pa
#          guidance_im: target overall appearance for the output

#             print("cost_SSD_overlap", cost_SSD_overlap.shape)
#             print("cost_SSD_transfer", cost_SSD_transfer.shape)

            cost_SSD = alpha * cost_SSD_overlap+(1-alpha) * cost_SSD_t
    ########get the random position########
            chosen_position = choose_sample(sample, patch_size, cost_S
     #######put it in the OUTPUT picture########
            chosen_x = chosen_position[1]
            chosen_y = chosen_position[0]
            half_ps = patch_size // 2


            if  i == 0 and j  == 0:
                continue
            elif i == 0:
                old_y = each_hop * i
                old_x = each_hop * (j -1)
                chosen_new_patch = sample[chosen_y-half_ps : chosen_y+
                old_patch = output[old_y:old_y + patch_size, old_x:old
                right_of_p = old_patch[0:patch_size, patch_size-overla
                left_of_c = chosen_new_patch[0:patch_size, 0:overlap]
                patch_ssd = calculate_ssd(old_patch, chosen_new_patch,


                use_cut_result = cut(patch_ssd.T).T


                left_mask = np.zeros((patch_size, overlap, 3), dtype =
                for a in range(patch_size):
                    for b in range(overlap):
                        if (use_cut_result[a][b] == 0):
                            for c in range (3):
                                left_mask[a][b][c] = 1.0
                left_right_mix = right_of_p*left_mask + left_of_c*(1 -

                output[current_y:current_y + patch_size, current_x:cur
                sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x-h
                ##overlap for right_left
                output[current_y:current_y + patch_size,current_x:curr
            elif j == 0:
                old_y = each_hop * (i-1)
                old_x = each_hop * j
                chosen_new_patch = sample[chosen_y-half_ps : chosen_y+
                old_patch = output[old_y:old_y + patch_size, old_x:old
                bottom_of_p = old_patch[patch_size - overlap:patch_siz
                top_of_c = chosen_new_patch[0:overlap, 0:patch_size]

                patch_ssd = calculate_ssd(old_patch, chosen_new_patch,
                use_cut_result = cut(patch_ssd.T)#should be a "-" shap
#                 print("patch_ssd," patch_ssd.shape)
#                 print("in col")

                top_mask = np.zeros((overlap, patch_size, 3), dtype =

                for a in range(overlap):
                    for b in range(patch_size):
                        if (use_cut_result[a][b] == 0):
                            for c in range (3):
                                top_mask[a][b][c] = 1.0
# #                 print("left_mask,", left_mask.shape)
# #                 right_mask = np.zeros((patch_size, patch_size, 3),
                bottom_top_mix = bottom_of_p*top_mask + top_of_c*(1 -


                output[current_y:current_y + patch_size, current_x:cur
                sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x-h

                ##overlap for bottom_top
                output[current_y:current_y + overlap,current_x:current

            else:
                print("running, loopnumber i, j:",i,j)
                #left right old patch
                old_y = each_hop * i
                old_x = each_hop * (j -1)
                chosen_new_patch = sample[chosen_y-half_ps : chosen_y+
                old_patch_l_r = output[old_y:old_y + patch_size, old_x
                right_of_p = old_patch_l_r[0:patch_size, patch_size-ov
                left_of_c = chosen_new_patch[0:patch_size, 0:overlap]
                patch_ssd_left_right = calculate_ssd(old_patch_l_r, ch
                use_cut_result = cut(patch_ssd_left_right.T).T
                left_mask = np.zeros((patch_size, overlap, 3), dtype =
                for a in range(patch_size):
                    for b in range(overlap):
                        if (use_cut_result[a][b] == 0):
                            for c in range (3):
                                left_mask[a][b][c] = 1.0
                left_right_mix = right_of_p*left_mask + left_of_c*(1 -
                output[current_y:current_y + patch_size, current_x:cur
                sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x-h

                output[current_y:current_y + patch_size,current_x:curr
                #bottom top old patch

                old_y_2 = each_hop * (i-1)
                old_x_2 = each_hop * j

                old_patch_t_b = output[old_y_2:old_y_2 + patch_size, o
                bottom_of_p = old_patch_t_b[patch_size - overlap:patch
                top_of_c = chosen_new_patch[0:overlap, 0:patch_size]
                patch_ssd_bottom_top = calculate_ssd(old_patch_t_b, ch
                use_cut_result_2 = cut(patch_ssd_bottom_top.T)#should
# #                 print("patch_ssd," patch_ssd.shape)
# #                 print("in col")

                top_mask = np.zeros((overlap, patch_size, 3), dtype =

                for a in range(overlap):
                    for b in range(patch_size):
                        if (use_cut_result_2[a][b] == 0):
                            for c in range (3):
                                top_mask[a][b][c] = 1.0
# # #                 print("left_mask,", left_mask.shape)
# # #                 right_mask = np.zeros((patch_size, patch_size, 3
                bottom_top_mix = bottom_of_p*top_mask + top_of_c*(1 -
#                 output[current_y:current_y + patch_size, current_x:c
#                 sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x

                output[current_y:current_y + overlap,current_x:current




#             output[current_y:current_y + patch_size, current_x:curre
#             sample[chosen_y-half_ps : chosen_y+half_ps, chosen_x-hal
# 三维矩阵y[i,j,m]的参数理解:
# 第一维 i: 确定是哪一个二维矩阵
# 第二维 j: 每一个二维矩阵的行
# 第三维 m: 每一个二维矩阵的列


    return output
```

In [67]:

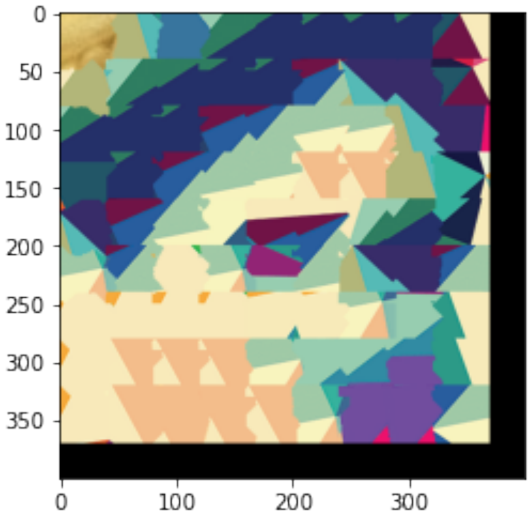
```
# load/process appropriate input texture and guidance images

# def quilt_cut(sample, out_size, patch_size, overlap, tol):

sample_img_fn = 'samples/pattern2.png'
sample = cv2.cvtColor(cv2.imread(sample_img_fn), cv2.COLOR_BGR2RGB)
target_img_fn = 'samples/hf_400.jpg'
target_img = cv2.cvtColor(cv2.imread(target_img_fn), cv2.COLOR_BGR2RGB

# print(sample.shape, target_img.shape)



plt.imshow(sample)
# plt.savefig('./output_res/p3_bricks.jpg')
plt.show()
# plt.imshow(target_img)
# plt.show()

patch_size = 50
overlap = 10
tol = 3
alpha = 0.5
res = texture_transfer(sample, patch_size, overlap, tol, target_img, a

# # # plt.figure(figsize=(15,15))
show_output = np.array(res, np.int32)
plt.imshow(show_output)
plt.show()
```

0
50
100
150
200
250



```
out_size 400
loop_number 9
running, loopnumber i, j: 1 1
running, loopnumber i, j: 1 2
running, loopnumber i, j: 1 3
running, loopnumber i, j: 1 4
running, loopnumber i, j: 1 5
running, loopnumber i, j: 1 6
running, loopnumber i, j: 1 7
running, loopnumber i, j: 1 8
running, loopnumber i, j: 2 1
running, loopnumber i, j: 2 2
running, loopnumber i, j: 2 3
running, loopnumber i, j: 2 4
running, loopnumber i, j: 2 5
running, loopnumber i, j: 2 6
running, loopnumber i, j: 2 7
running, loopnumber i, j: 2 8
running, loopnumber i, j: 3 1
running, loopnumber i, j: 3 2
running, loopnumber i, j: 3 3
running, loopnumber i, j: 3 4
running, loopnumber i, j: 3 5
running, loopnumber i, j: 3 6
running, loopnumber i, j: 3 7
running, loopnumber i, j: 3 8
running, loopnumber i, j: 4 1
running, loopnumber i, j: 4 2
running, loopnumber i, j: 4 3
running, loopnumber i, j: 4 4
running, loopnumber i, j: 4 5
running, loopnumber i, j: 4 6
running, loopnumber i, j: 4 7
running, loopnumber i, j: 4 8
running, loopnumber i, j: 5 1
running, loopnumber i, j: 5 2
running, loopnumber i, j: 5 3
running, loopnumber i, j: 5 4
running, loopnumber i, j: 5 5
running, loopnumber i, j: 5 6
running, loopnumber i, j: 5 7
running, loopnumber i, j: 5 8
running, loopnumber i, j: 6 1
running, loopnumber i, j: 6 2
running, loopnumber i, j: 6 3
running, loopnumber i, j: 6 4
running, loopnumber i, j: 6 5
running, loopnumber i, j: 6 6
```

```
running, loopnumber i, j: 6 7
running, loopnumber i, j: 6 8
running, loopnumber i, j: 7 1
running, loopnumber i, j: 7 2
running, loopnumber i, j: 7 3
running, loopnumber i, j: 7 4
running, loopnumber i, j: 7 5
running, loopnumber i, j: 7 6
running, loopnumber i, j: 7 7
running, loopnumber i, j: 7 8
running, loopnumber i, j: 8 1
running, loopnumber i, j: 8 2
running, loopnumber i, j: 8 3
running, loopnumber i, j: 8 4
running, loopnumber i, j: 8 5
running, loopnumber i, j: 8 6
running, loopnumber i, j: 8 7
running, loopnumber i, j: 8 8
```



Bells & Whistles

(10 pts) Create and use your own version of cut.m. To get these points, you should create your own implementation without basing it directly on the provided function (you're on the honor code for this one).

You can simply copy your customized_cut(bndcost) into the box below so that it is easier for us to grade

In [ ]:

```

```

(15 pts) Implement the iterative texture transfer method described in the paper. Compare to the non-iterative method for two examples.

```
In [ ]:
```

(up to 20 pts) Use a combination of texture transfer and blending to create a face-in-toast image like the one on top. To get full points, you must use some type of blending, such as feathering or Laplacian pyramid blending.

```
In [ ]:
```

(up to 40 pts) Extend your method to fill holes of arbitrary shape for image completion. In this case, patches are drawn from other parts of the target image. For the full 40 pts, you should implement a smart priority function (e.g., similar to Criminisi et al.).

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```