Programming Project #2: Image Quilting

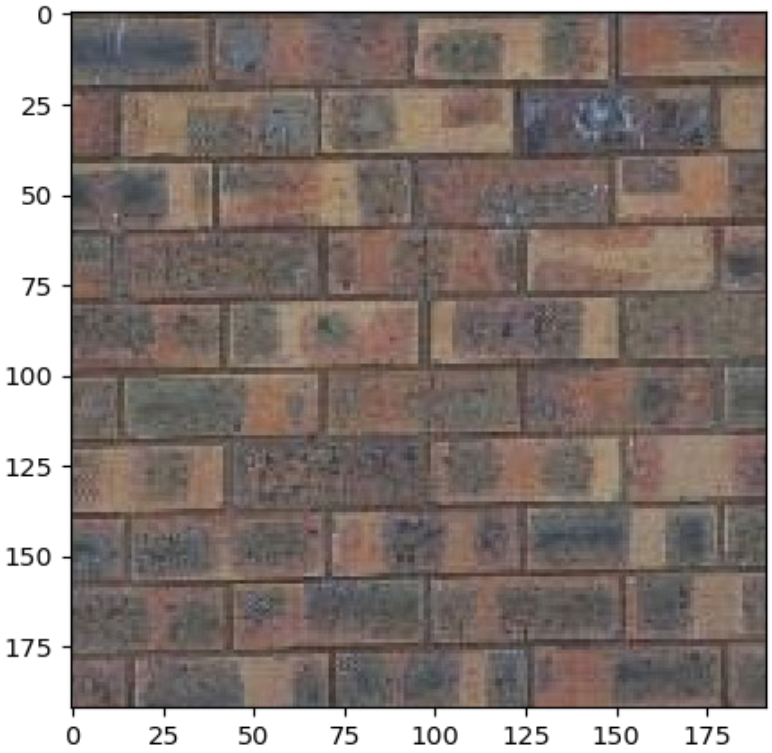
CS445: Computational Photography - Fall 2019

In [1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib notebook
import utils
import os
from random import randint
import math
from pprint import pprint
```

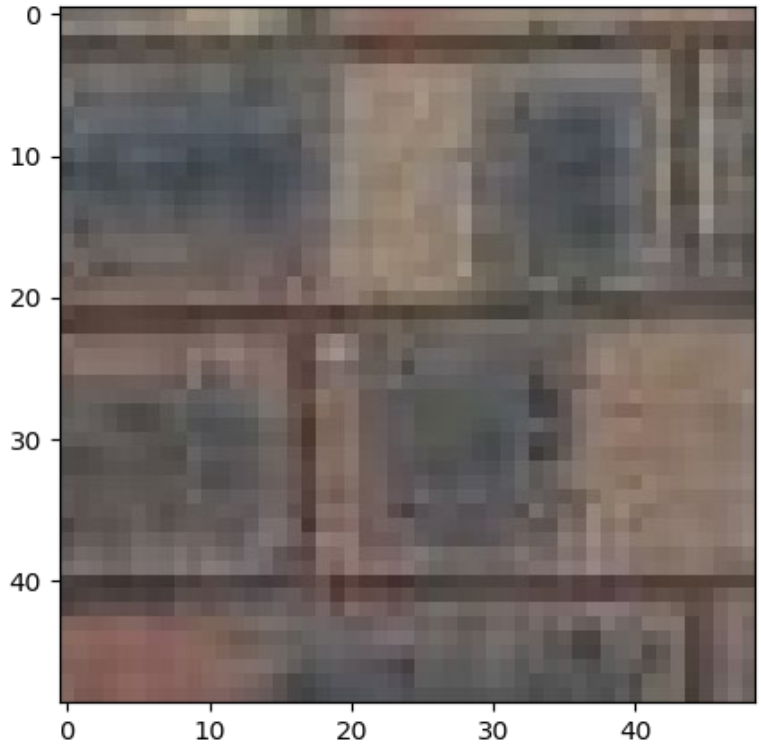
In [2]:
```
from utils import cut # default cut function for seam finding section
```

Part I: Randomly Sampled Texture (10 pts)

In [3]:
```
#https://images-na.ssl-images-amazon.com/images/I/81p-%2Bkp6%2BRL._AC_SL1500_.jpg bricks
#https://images.theconversation.com/files/133922/original/image-20160812-16339-v2g90o.png?ixlib=rb-1.1.0&q=45&auto=format&w=496&fit=clip cool patters
sample_img_dir = 'samples/bricks_small.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
    fig, axes = plt.subplots(1, 1)
    axes.imshow(sample_img)
```



In [4]:
```
def sample_image(width,sample_img):
    row = sample_img.shape[0]
    column = sample_img.shape[1]
    random_row = randint(0,row - width)
    random_column = randint(0,column - width)
    new = sample_img[random_row:random_row + width,random_column:random_co
lumn + width,:]
    return new


fig, axes = plt.subplots(1, 1)
axes.imshow(sample_image(49,sample_img))
```

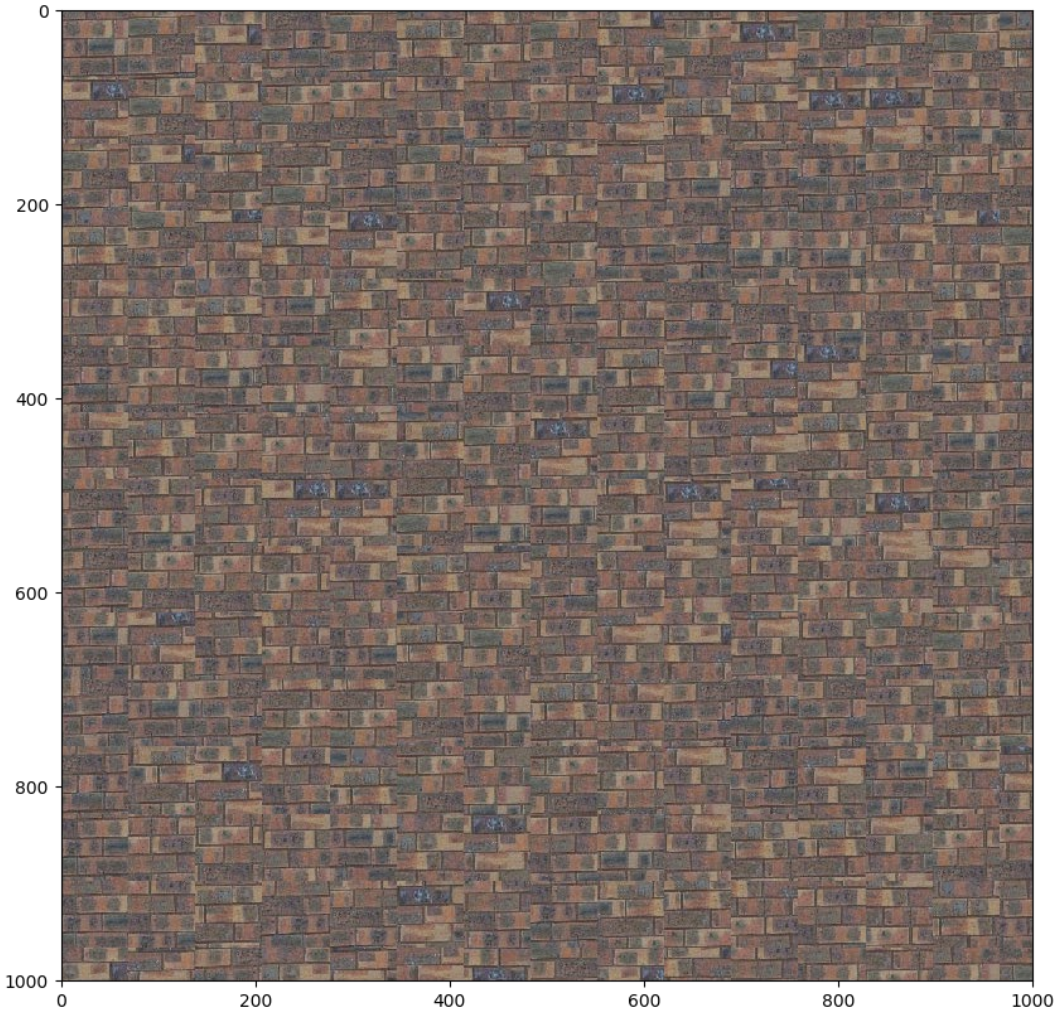


Out[4]: <matplotlib.image.AxesImage at 0x18eeb7fd438>

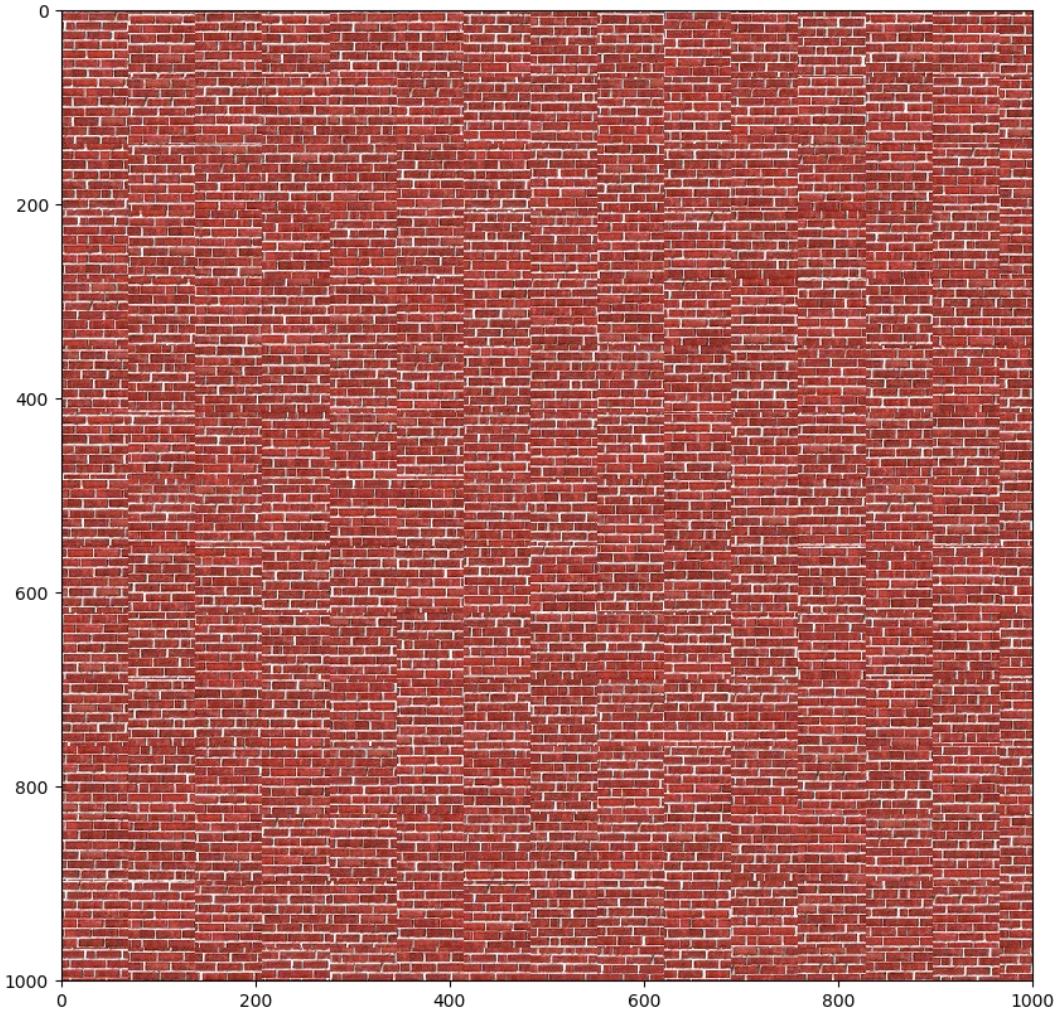
In [6]:
```
def quilt_random(sample, out_size, patch_size):
    """
    Randomly samples square patches of size patchsize from sample in order
 to create an output image of size outsize.

    :param sample: numpy.ndarray   The image you read from sample director
y
    :param out_size: int            The width of the square output image
    :param patch_size: int          The width of the square sample patch
    :return: numpy.ndarray
    """
    size = out_size
    output = np.zeros(shape=(size,size,3))
    new = sample_image(patch_size,sample_img)
    y = 0
    size_patch = new.shape[0]
    while y < size:
        x = 0
        while x < size:
            if x + size_patch > size and y + size_patch > size:
                new = sample_image(patch_size,sample_img)
                output[y:size,x:size] = new[0:size-y,0:size-x]
                break
            if x + size_patch > size:
                new = sample_image(patch_size,sample_img)
                output[y:size_patch+y,x:size] = new[:,0:size-x]
```

```
                break
            if y + size_patch > size:
                new = sample_image(patch_size,sample_img)
                output[y:size,x:size_patch+x] = new[0:size-y,:]
                x += size_patch
            else:
                new = sample_image(patch_size,sample_img)
                output[y:size_patch+y,x:size_patch+x] = new
                x += size_patch
        y += size_patch
    output = output.astype('uint8')
    return output
```

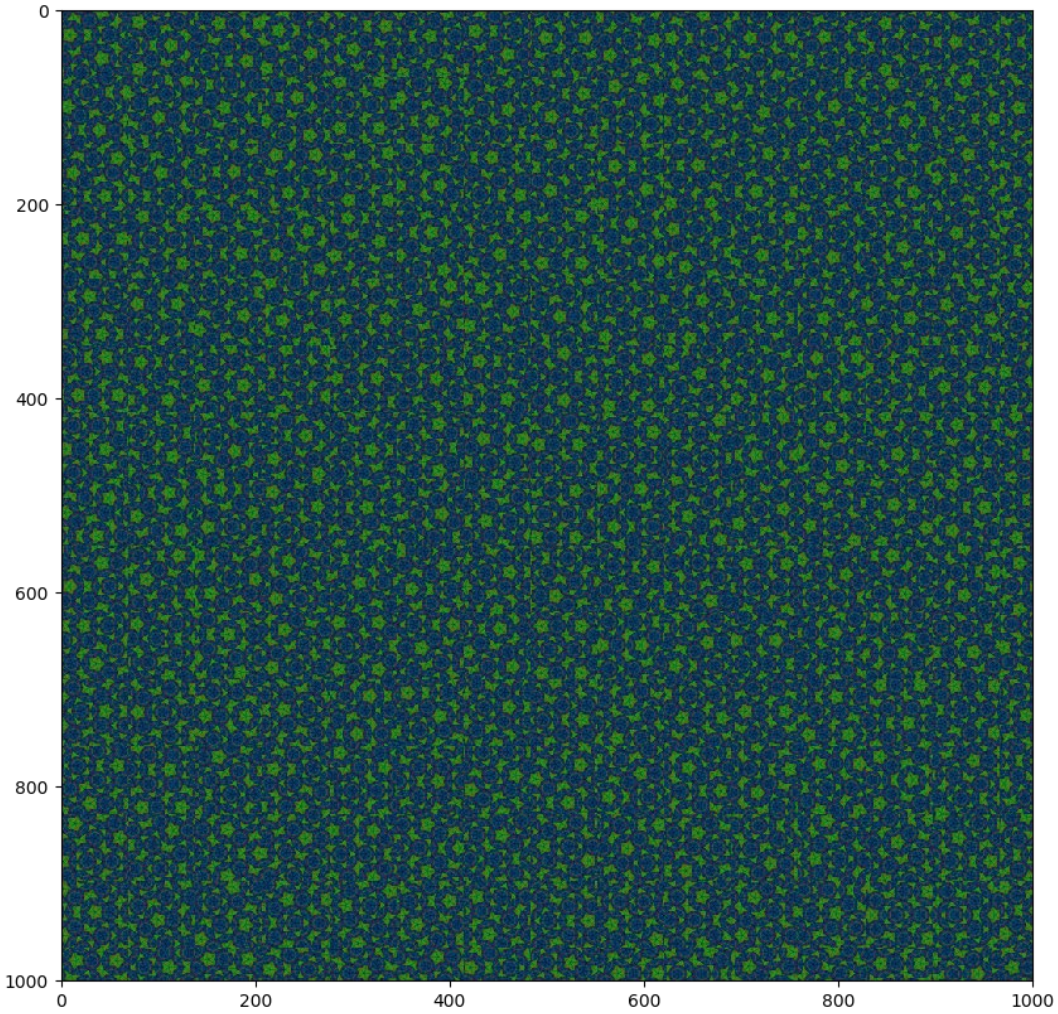
In [6]:
```
out_size = 1000 # feel free to change to debug
patch_size = 69 # feel free to change to debug
res = quilt_random(sample_img, out_size, patch_size)
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(res)
plt.savefig('bricks_small_random.png')
```



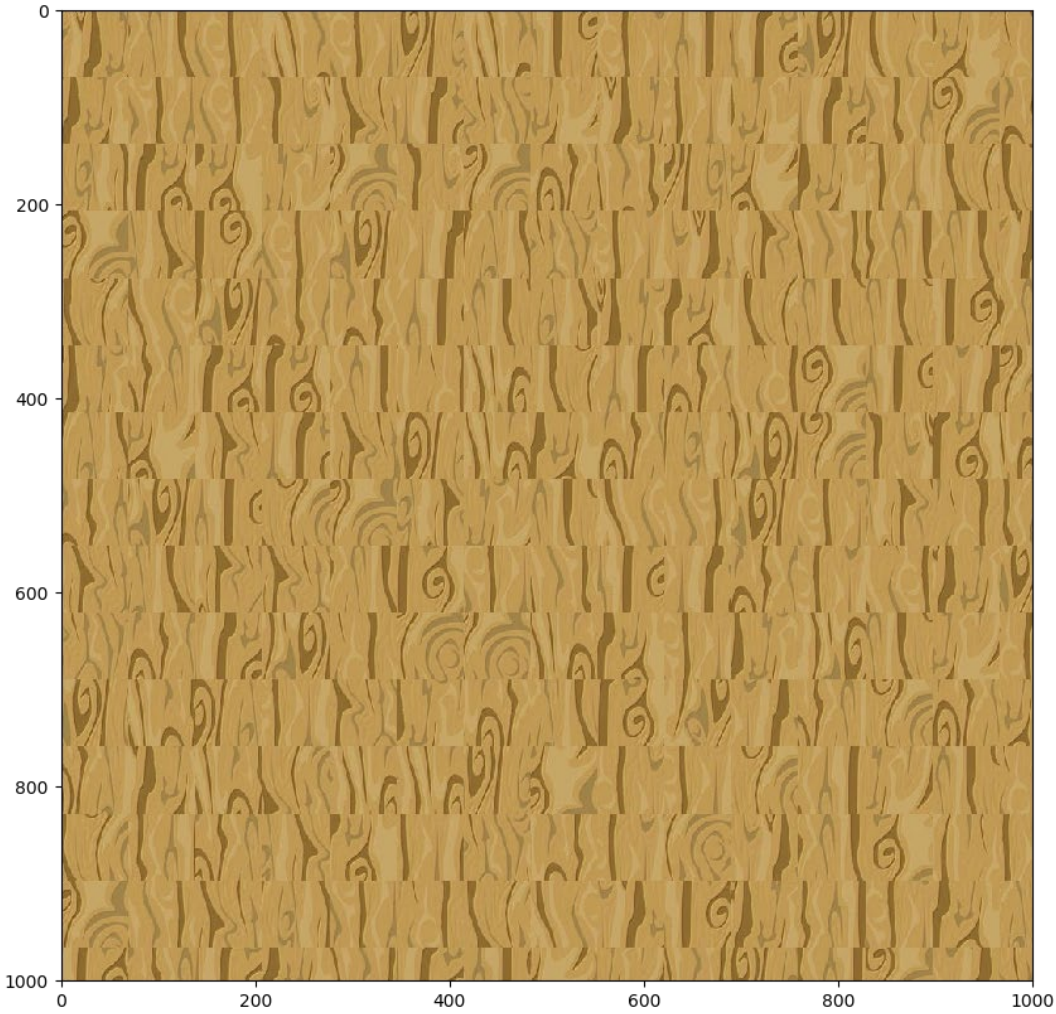
In [21]:
```
sample_img_dir = 'samples/bricks.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
out_size = 1000 # feel free to change to debug
patch_size = 69 # feel free to change to debug
res = quilt_random(sample_img, out_size, patch_size)
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(res)
plt.savefig('bricks_random.png')
```



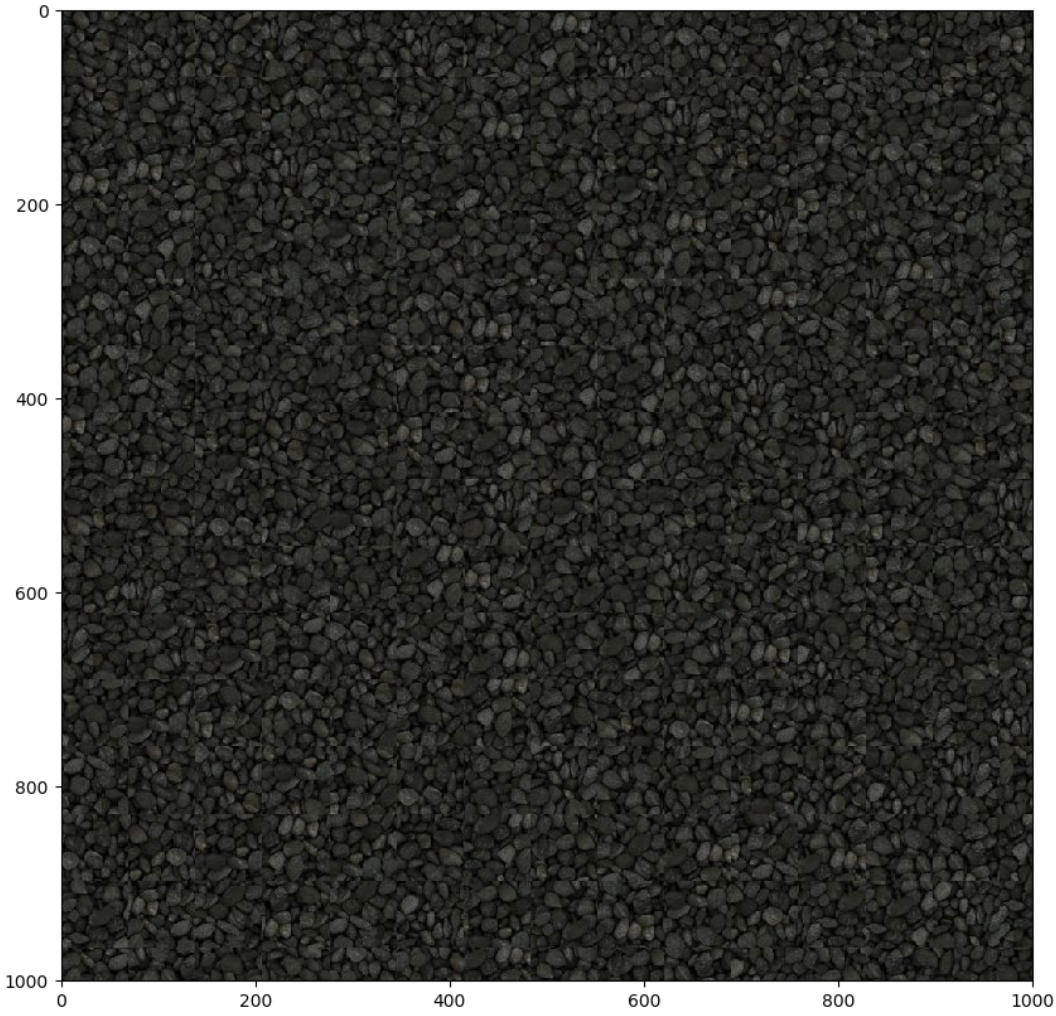
In [38]:
```
sample_img_dir = 'samples/cool_pattern.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
out_size = 1000 # feel free to change to debug
patch_size = 69 # feel free to change to debug
res = quilt_random(sample_img, out_size, patch_size)
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(res)
plt.savefig('cool_pattern_random.png')
```



In [41]:
```
sample_img_dir = 'samples/wood.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
out_size = 1000 # feel free to change to debug
patch_size = 69 # feel free to change to debug
res = quilt_random(sample_img, out_size, patch_size)
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(res)
plt.savefig('wood_random.png')
```



In [67]:
```
sample_img_dir = 'samples/rocks.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
out_size = 1000 # feel free to change to debug
patch_size = 69 # feel free to change to debug
res = quilt_random(sample_img, out_size, patch_size)
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(res)
plt.savefig('rocks_random.png')
```



Part II: Overlapping Patches (30 pts)

In [5]:
```
def ssd_patch_left(overlap,left_patch,patch_size,sample):
    M = np.zeros(shape=(patch_size,patch_size,3))
    M[:,:overlap,:] = 1
    T = np.zeros(shape=(patch_size,patch_size,3))
    T[:,:overlap,:] = left_patch[:, patch_size-overlap:patch_size,:] #temp
late overlap real values
    T/255.0
    I = sample/255.0
    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [6]:
```
def ssd_patch_up(overlap,up_patch,patch_size,sample):
    M = np.zeros(shape=(patch_size,patch_size,3))
    M[:overlap,:,:] = 1
```

```
    T = np.zeros(shape=(patch_size,patch_size,3))
    test = up_patch[patch_size-overlap:patch_size, :,:]
    test_2 = T[:overlap,:,:]
    T[:overlap,:,:] = up_patch[patch_size-overlap:patch_size, :,:] #templa
te overlap real values
    T/255.0
    I = sample/255.0
    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [7]:
```
def ssd_patch_both(overlap,up_patch,left_patch,patch_size,sample):
    size_x = up_patch.shape[1]
    size_y = left_patch.shape[0] + overlap
    M = np.zeros(shape=(size_y,size_x,3))
    M[:overlap,:overlap,:] = 1
    T = np.zeros(shape=(size_y,size_x,3))
    T[:overlap,:,:] = up_patch #template overlap real values
    T[overlap:,:overlap,:] = left_patch
    T/255.0
    I = sample/255.0
    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [8]:
```
def choose_sample(k,ssd,overlap,patch_size,input_image):
    input_image_size = input_image.shape[0]
    #print(input_image_size)
    row = []
    col = []
    z = []
    ssd_test = ssd[:,:,0]
    while len(row) < k:
        row_1, col_1 = np.where(ssd_test == np.amin(ssd_test))
        if (patch_size-1)/2<row_1[0] < input_image_size-((patch_size-1)/2)
 and (patch_size-1)/2<col_1[0] < input_image_size-((patch_size-1)/2):
            row.append(row_1[0])
            col.append(col_1[0])
        ssd_test[row_1,col_1] = 10000000000000
    #print (row, col, z)
    rand_int = randint(0,k-1)
    return input_image[row[rand_int]-int(((patch_size-1)/2)):(row[rand_int
]-int((patch_size-1)/2))+patch_size,col[rand_int]-int((patch_size-1)/2):(c
ol[rand_int]-int((patch_size-1)/2))+patch_size,:]
```

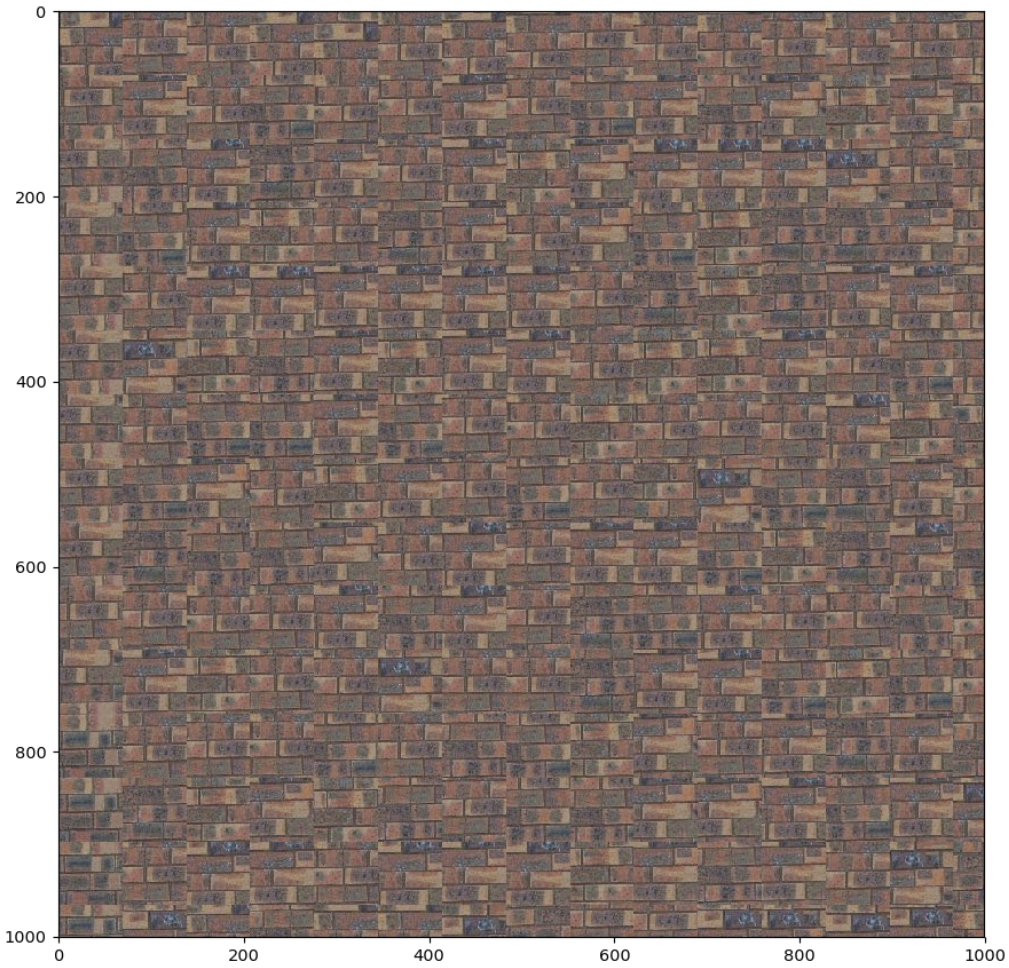
In [9]:
```
def quilt_simple(sample, out_size, patch_size, overlap, k):
    """
    Randomly samples square patches of size patchsize from sample in order
 to create an output image of size outsize.
    Feel free to add function parameters
    :param sample: numpy.ndarray
    :param out_size: int
    :param patch_size: int
    :param overlap: int
    :param k: int how many random samples to choose from smallest ssd valu
es.
```

```
    :return: numpy.ndarray
    """
    output = np.zeros(shape=(out_size,out_size,3))
    y = 0
    t = 0
    print(t)
    while y < out_size:
        x = 0
        while x < out_size:
            if y == 0 and x == 0: #1
                random_sample = sample_image(patch_size,sample)
                output[:patch_size,:patch_size,:] = random_sample
                x += patch_size
                #print(y,x)
            elif y ==0 and x + patch_size <= out_size: #2
                left_patch = output[y:patch_size,x-patch_size:x,:]
                ssd = ssd_patch_left(overlap,left_patch,patch_size,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)
                output[y:patch_size,x:x+patch_size,:] = out
                x += patch_size
                #print(y,x)
            elif y ==0 and x + patch_size > out_size: #3
                left_patch = output[y:patch_size,x-patch_size:x,:]
                ssd = ssd_patch_left(overlap,left_patch,patch_size,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)
                output[y:patch_size,x:out_size,:] = out[:,:out_size-x,:]
                y += patch_size
                x = out_size + 1
                #print(y,x)
            elif y + patch_size <= out_size and x == 0:#4
                up_patch = output[y-patch_size:y,x:x+patch_size,:]
                ssd = ssd_patch_up(overlap,up_patch,patch_size,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)
                output[y:y+patch_size,x:x+patch_size,:] = out
                x += patch_size
                #print(y,x)
            elif y + patch_size <= out_size and x + patch_size <= out_size
: #5
                up_patch = output[y-overlap:y,x-overlap:x+patch_size,:]
                left_patch = output[y:y+patch_size,x-overlap:x,:]
                ssd = ssd_patch_both(overlap,up_patch,left_patch,patch_siz
e,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)
                output[y:y+patch_size,x:x+patch_size,:] = out
                x += patch_size
                #print(y,x)
            elif y + patch_size <= out_size and x + patch_size > out_size:
 #6
                up_patch = output[y-overlap:y,x-overlap:out_size,:]
                left_patch = output[y:y+patch_size,x-overlap:x,:]
                ssd = ssd_patch_both(overlap,up_patch,left_patch,patch_siz
e,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)
                output[y:y+patch_size,x:out_size,:] = out[:,:out_size-x,:]
                x = out_size + 1
                y += patch_size
                #print(y,x)
```

```
            elif y + patch_size > out_size and x == 0: #7
                up_patch = output[y-patch_size:y,x:x+patch_size,:]
                ssd = ssd_patch_up(overlap,up_patch,patch_size,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)
                output[y:out_size,x:x+patch_size,:] = out[:out_size-y,:,:]
                x += patch_size
                #print(y,x)
            elif y + patch_size > out_size and x + patch_size <= out_size:
                up_patch = output[y-overlap:y,x-overlap:x+patch_size,:]
                left_patch = output[y:out_size,x-overlap:x,:]
                ssd = ssd_patch_both(overlap,up_patch,left_patch,patch_siz
e,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)
                output[y:out_size,x:x+patch_size,:] = out[:out_size-y,:,:]
                x += patch_size
            else:
                up_patch = output[y-overlap:y,x-overlap:out_size,:]
                left_patch = output[y:out_size,x-overlap:x,:]
                ssd = ssd_patch_both(overlap,up_patch,left_patch,patch_siz
e,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)
                output[y:out_size,x:out_size,:] = out[:out_size-y,:out_siz
e-x,:]
                return output
```

In [10]:
```
output = quilt_simple(sample_img, 1000, 69, 10, 400)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('text_small_overlapping_patches.png')
```

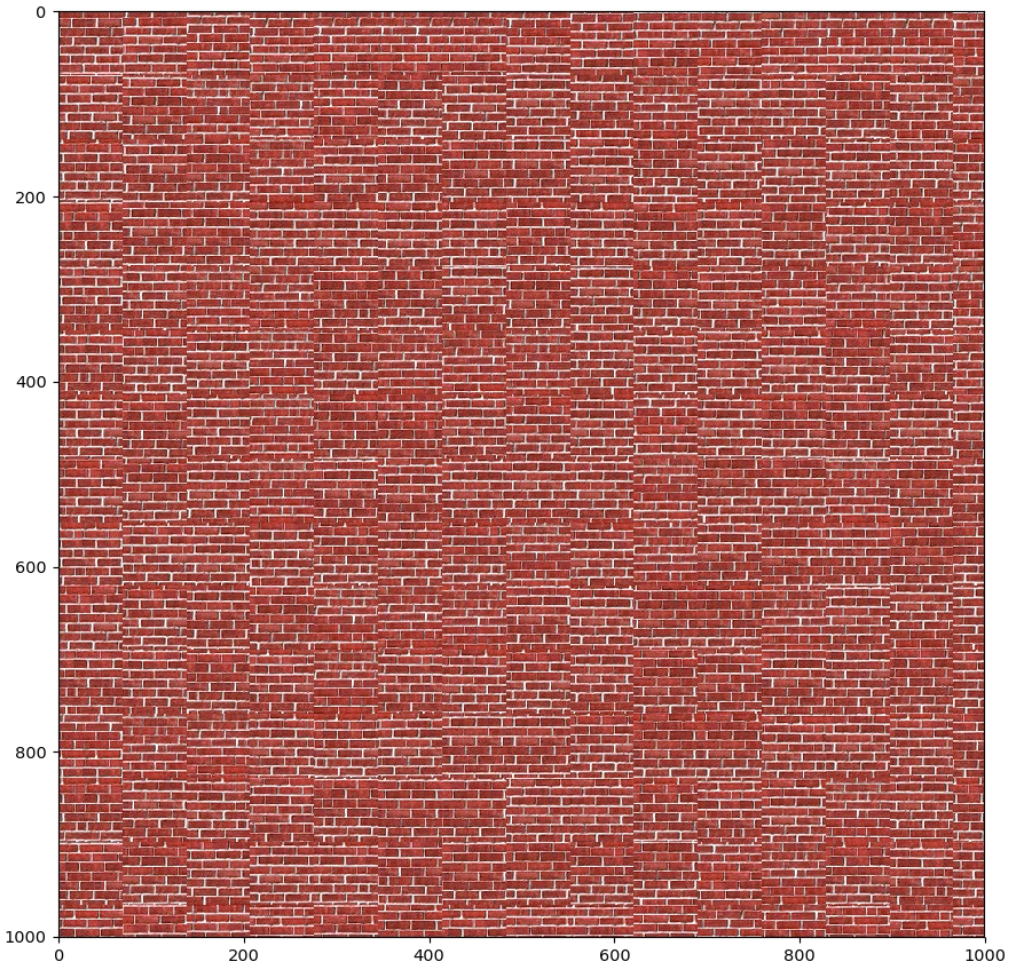
0



Out[10]: <matplotlib.image.AxesImage at 0x18eec042e80>

In [11]:
```
sample_img_dir = 'samples/bricks.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
output = quilt_simple(sample_img, 1000, 69, 10, 400)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('bricks_overlapping_patches.png')
```

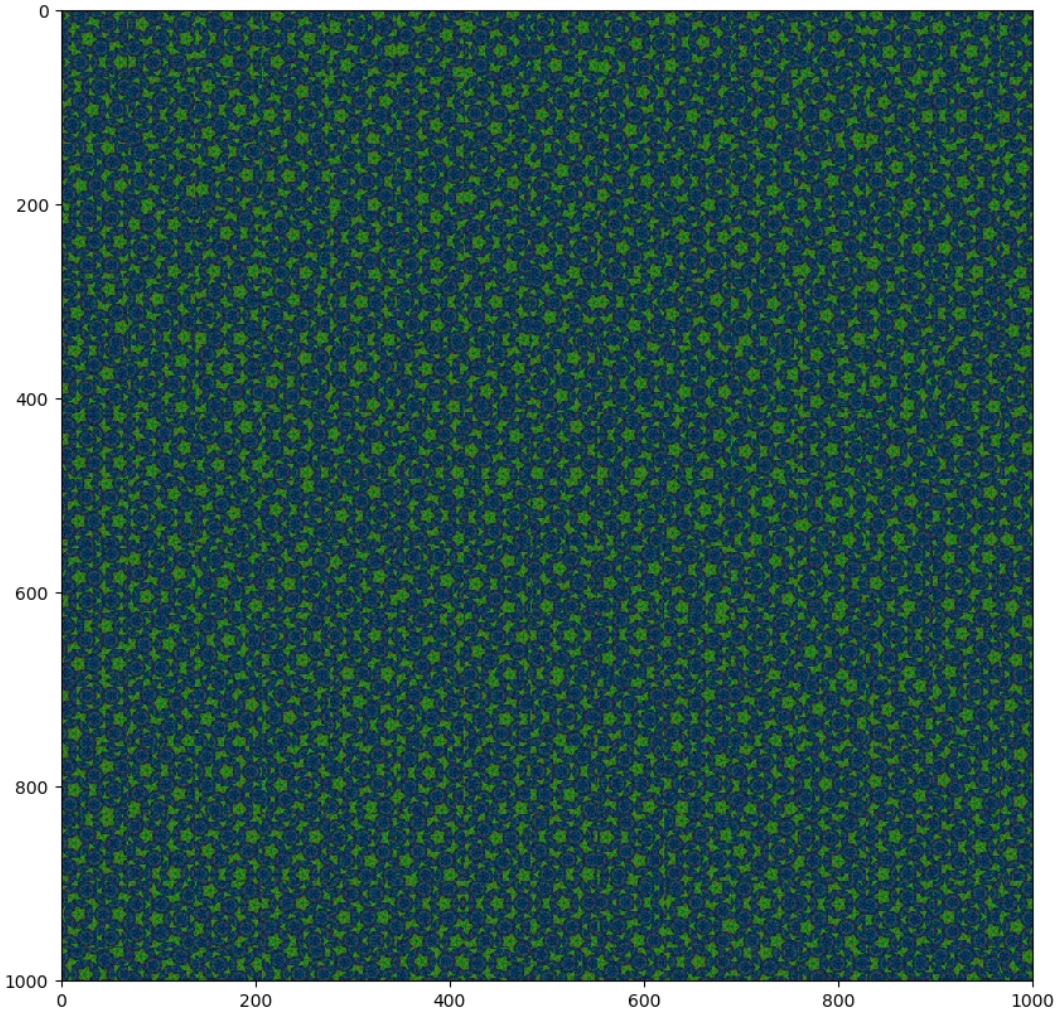
0



Out[11]: <matplotlib.image.AxesImage at 0x18eecf229b0>

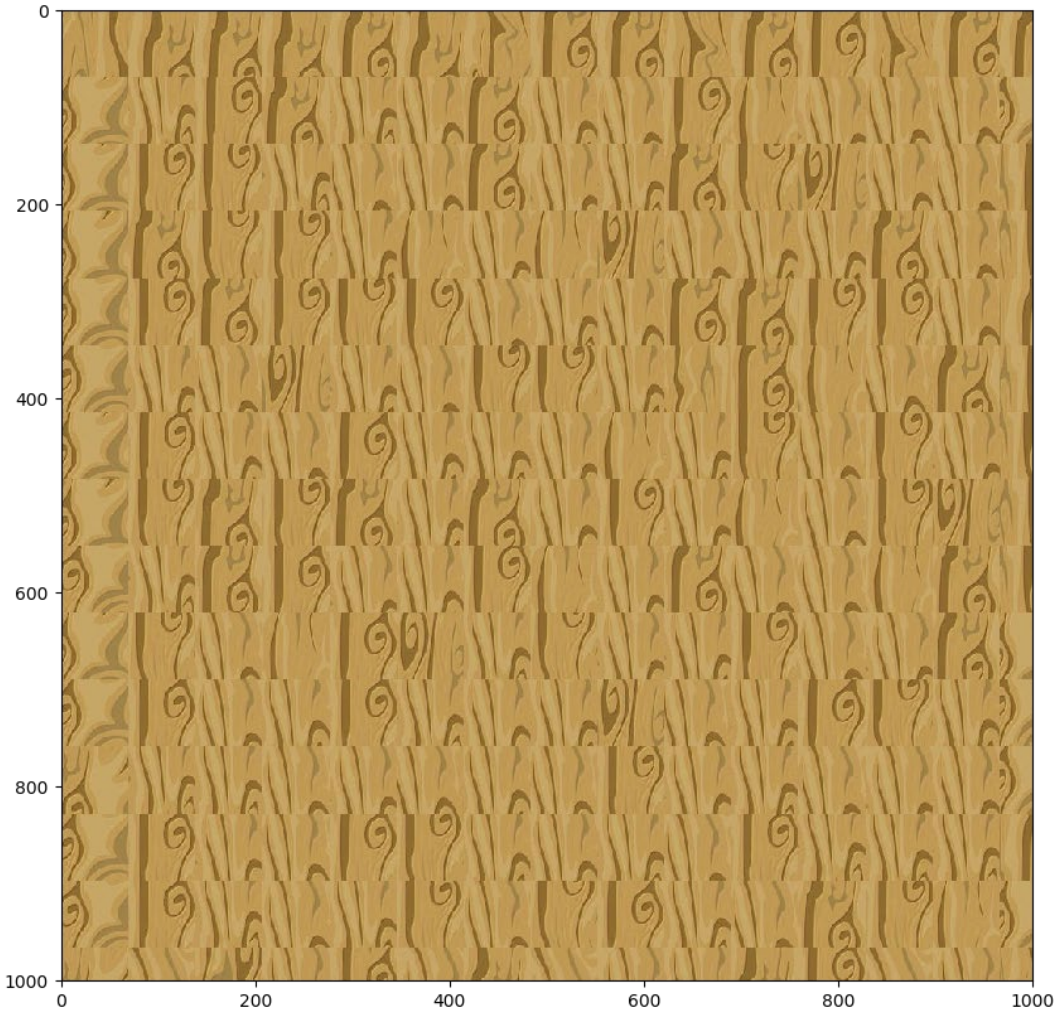
In [39]:
```
sample_img_dir = 'samples/cool_pattern.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
output = quilt_simple(sample_img, 1000, 69, 10, 400)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('cool_pattern_overlapping_patches.png')
```

0



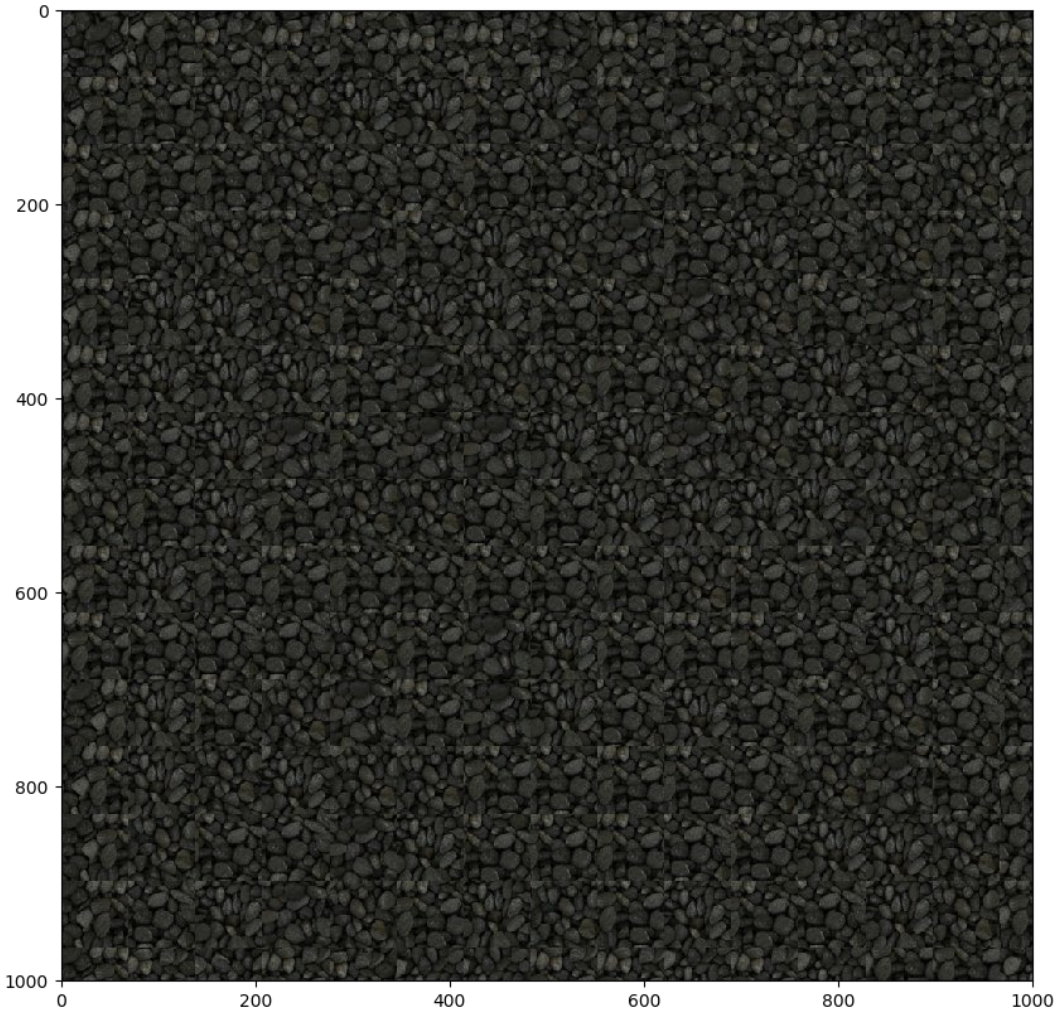
In [42]:
```
sample_img_dir = 'samples/wood.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
output = quilt_simple(sample_img, 1000, 69, 10, 400)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
plt.savefig('wood_overlapping_patches.png')
```

0



In [85]:
```
sample_img_dir = 'samples/rocks.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
output = quilt_simple(sample_img, 1000, 69, 10, 400)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
plt.savefig('rocks_overlapping_patches.jpg')
```

0



Part III: Seam Finding (20 pts)

In [12]:
```
def sample_image(width,sample_img):
    row = sample_img.shape[0]
    column = sample_img.shape[1]
    random_row = randint(0,row - width)
    random_column = randint(0,column - width)
    new = sample_img[random_row:random_row + width,random_column:random_co
lumn + width,:]
    return new
```

In [13]:
```
def ssd_patch_left(overlap,left_patch,patch_size,sample):
    M = np.zeros(shape=(patch_size,patch_size,3))
    M[:,:overlap,:] = 1
    T = np.zeros(shape=(patch_size,patch_size,3))
    T[:,:overlap,:] = left_patch[:, patch_size-overlap:patch_size,:] #temp
late overlap real values
```

```
    T/255.0
    I = sample/255.0
    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [14]:
```
def ssd_patch_up(overlap,up_patch,patch_size,sample):
    M = np.zeros(shape=(patch_size,patch_size,3))
    M[:overlap,:,:] = 1
    T = np.zeros(shape=(patch_size,patch_size,3))
    test = up_patch[patch_size-overlap:patch_size, :,:]
    test_2 = T[:overlap,:,:]
    T[:overlap,:,:] = up_patch[patch_size-overlap:patch_size, :,:] #templa
te overlap real values
    T/255.0
    I = sample/255.0
    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [15]:
```
def ssd_patch_both(overlap,up_patch,left_patch,patch_size,sample):
    size_x = up_patch.shape[1]
    size_y = left_patch.shape[0] + overlap
    M = np.zeros(shape=(size_y,size_x,3))
    M[:overlap,:overlap,:] = 1
    T = np.zeros(shape=(size_y,size_x,3))
    T[:overlap,:,:] = up_patch #template overlap real values
    T[overlap:,:overlap,:] = left_patch
    T/255.0
    I = sample/255.0
    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [16]:
```
def overlap_patch_left(A,B,overlap,patch_size):
    #A is the left patch
    #B is the new patch
    row_size = A.shape[0]
    col_size = A.shape[1]
    s = (A[:,:,:]-B[:,:,:])**2
    s = s[:,:,0] + s[:,:,1] + s[:,:,2]
    s =  s.transpose()
    mask = cut(s)
    mask = mask.transpose()
    output_in = np.zeros(shape=(row_size,overlap,3))
    row = 0
    col = 0
    for x in mask:
        col = 0
        for y in x:
            if y == 0:
                output_in[row,col,:] = A[row,col,:]
            else:
                output_in[row,col,:] = B[row,col,:]
            #print(row,col)
            col += 1
```

```
        row += 1
    return output_in
```

In [17]:
```
def overlap_patch_up(A,B,overlap,patch_size):
    #A is the up patch
    #B is the new patch
    row_size = A.shape[0]
    col_size = A.shape[1]
    s = (A[:,:,:]-B[:,:,:])**2
    s = s[:,:,0] + s[:,:,1] + s[:,:,2]
    mask = cut(s)
    output_in = np.zeros(shape=(overlap,col_size,3))
    row = 0
    col = 0
    for x in mask:
        col = 0
        for y in x:
            if y == 0:
                output_in[row,col,:] = A[row,col,:]
            else:
                output_in[row,col,:] = B[row,col,:]
            #print(row,col)
            col += 1
        row += 1
    return output_in
```

In [18]:
```
def choose_sample(k,ssd,overlap,patch_size,input_image):
    input_image_size = input_image.shape[0]
    #print(input_image_size)
    row = []
    col = []
    z = []
    ssd_test = ssd[:,:,0] + ssd[:,:,1] + ssd[:,:,2]
    while len(row) < k:
        row_1, col_1 = np.where(ssd_test == np.amin(ssd_test))
        if (patch_size-1)/2<row_1[0] < input_image_size-((patch_size-1)/2)
 and (patch_size-1)/2<col_1[0] < input_image_size-((patch_size-1)/2):
            row.append(row_1[0])
            col.append(col_1[0])
        ssd_test[row_1,col_1] = 10000000000000
    #print (row, col, z)
    rand_int = randint(0,k-1)
    return input_image[row[rand_int]-int(((patch_size-1)/2)):(row[rand_int
]-int((patch_size-1)/2))+patch_size,col[rand_int]-int((patch_size-1)/2):(c
ol[rand_int]-int((patch_size-1)/2))+patch_size,:]
```

In [19]:
```
def quilt_cut(sample, out_size, patch_size, overlap, k):
    """
    Samples square patches of size patchsize from sample using seam findin
g in order to create an output image of size outsize.
    Feel free to add function parameters
    :param sample: numpy.ndarray
    :param out_size: int
    :param patch_size: int
    :param overlap: int
    :param tol: float
```

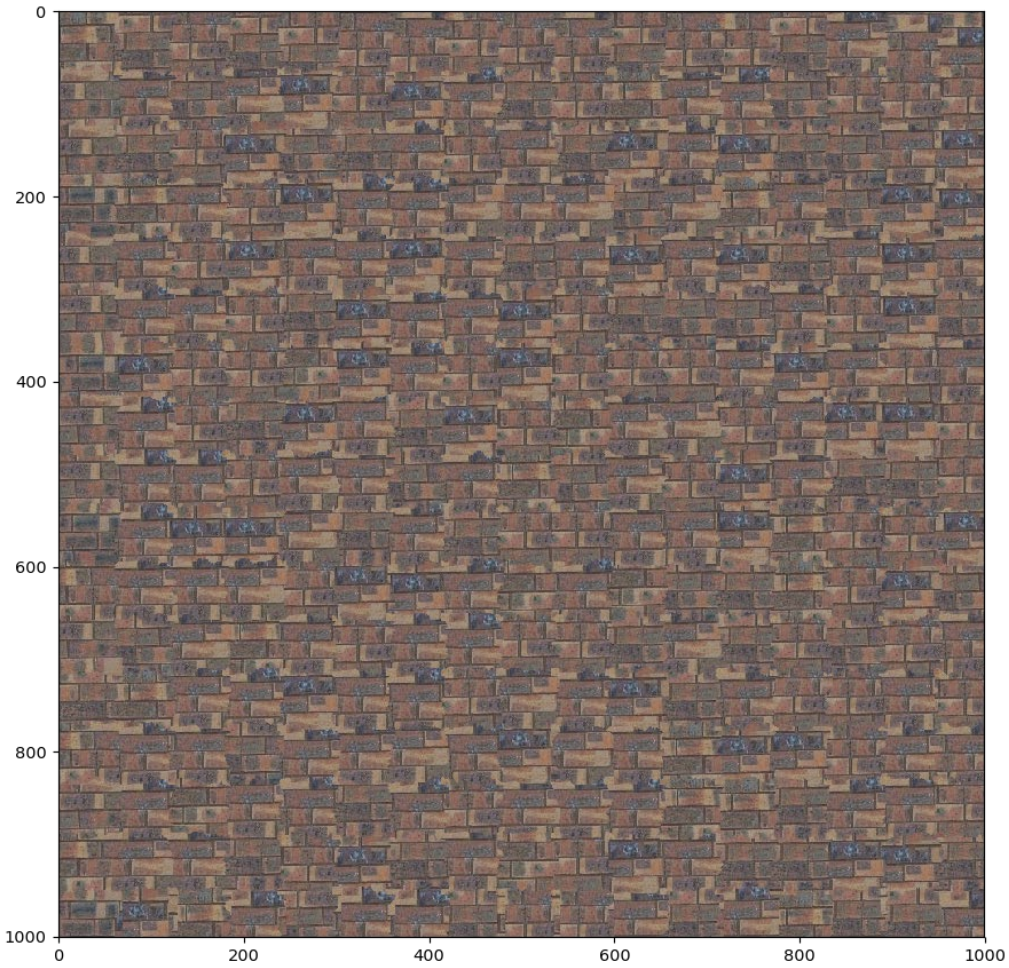
```
    :return: numpy.ndarray
    """
    output = np.zeros(shape=(out_size,out_size,3))
    t = 0
    y = 0 #the current row value
    while y < out_size:
        x = 0 #current column value
        while x < out_size:
            if y == 0 and x == 0: #1
                random_sample = sample_image(patch_size,sample)
                output[:patch_size,:patch_size,:] = random_sample
                x += patch_size
                #print(y,x)
            elif y ==0 and x + patch_size - overlap <= out_size: #2
                left_patch = output[y:patch_size,x-patch_size:x,:]
                ssd = ssd_patch_left(overlap,left_patch,patch_size,sample)

                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #left
                A = left_patch[:,patch_size-overlap:,:]
                B = out[:,:overlap,:]
                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:patch_size,x-overlap:x,:] = left

                #left over
                output[y:patch_size,x:x+patch_size-overlap,:] = out[:,over
lap:,:]
                x = x + patch_size - overlap
                #print(y,x)
            elif y ==0 and x + patch_size > out_size: #3
                left_patch = output[y:patch_size,x-patch_size:x,:]
                ssd = ssd_patch_left(overlap,left_patch,patch_size,sample)

                out = choose_sample(k,ssd,overlap,patch_size,sample)
                #left
                A = left_patch[:,patch_size-overlap:,:]
                B = out[:,:overlap,:]
                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:patch_size,x-overlap:x,:] = left

                #left over
                output[y:patch_size,x:out_size,:] = out[:,overlap:out_size
-x+overlap,:]
                x = out_size + 1
                y = y + patch_size
                #print(y,x)
            elif y + patch_size <= out_size and x == 0:#4
                up_patch = output[y-patch_size:y,x:x+patch_size,:]
                ssd = ssd_patch_up(overlap,up_patch,patch_size,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #up
                A = up_patch[patch_size-overlap:,:,:]
                B = out[:overlap,:,:]
                up = overlap_patch_up(A,B,overlap,patch_size)
                output[y-overlap:y,x:patch_size,:] = up
```

```
                #left over
                output[y:y+patch_size-overlap,x:patch_size,:] = out[overla
p:,:,:]
                x = x + patch_size
                #print(y,x)
            elif y + patch_size <= out_size and x + patch_size <= out_size
: #5
                up_patch = output[y-overlap:y,x-overlap:x+patch_size-overl
ap,:]
                left_patch = output[y:y+patch_size-overlap,x-overlap:x,:]

                ssd = ssd_patch_both(overlap,up_patch,left_patch,patch_siz
e,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)


                #up
                A = up_patch
                B = out[:overlap,:,:]
                up = overlap_patch_up(A,B,overlap,patch_size)
                output[y-overlap:y,x-overlap:x+patch_size-overlap,:] = up

                #left
                A = left_patch
                B = out[overlap:,:overlap,:]


                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:y+patch_size-overlap,x-overlap:x,:] = left

                #leftover
                output[y:y+patch_size-overlap,x:x+patch_size-overlap,:] =
out[overlap:,overlap:,:]
                x = x + patch_size - overlap
                #print(y,x)

            elif y + patch_size <= out_size and x + patch_size > out_size:
 #6
                up_patch = output[y-overlap:y,x-overlap:out_size,:]
                left_patch = output[y:y+patch_size-overlap,x-overlap:x,:]

                ssd = ssd_patch_both(overlap,up_patch,left_patch,patch_siz
e,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #UP
                A = up_patch
                B = out[:overlap,:out_size-x+overlap]
                up = overlap_patch_up(A,B,overlap,patch_size)
                output[y-overlap:y,x-overlap:out_size,:] = up

                #left
                A = left_patch
                B = out[overlap:,:overlap,:]
```

```
                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:y+patch_size-overlap,x-overlap:x,:] = left

                #leftover
                output[y:y+patch_size-overlap,x:out_size,:] = out[overlap:
,overlap:out_size-x+overlap,:]
                x = out_size + 1
                y = y + patch_size - overlap

                #print(y,x)
            elif y + patch_size > out_size and x == 0: #7
                up_patch = output[y-patch_size:y,x:x+patch_size,:]
                ssd = ssd_patch_up(overlap,up_patch,patch_size,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #UP
                A = up_patch[patch_size-overlap:,:,:]
                B = out[:overlap,:,:]
                up = overlap_patch_up(A,B,overlap,patch_size)
                output[y-overlap:y,x:patch_size,:] = up

                #left over
                output[y:out_size,x:x+patch_size,:] = out[overlap:out_size
-y+overlap,:,:]
                x = x + patch_size
                #print(y,x)

            elif y + patch_size > out_size and x + patch_size <= out_size:
 #8
                up_patch = output[y-overlap:y,x-overlap:x+patch_size-overl
ap,:]
                left_patch = output[y:out_size,x-overlap:x,:]
                ssd = ssd_patch_both(overlap,up_patch,left_patch,patch_siz
e,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #up
                A = up_patch
                B = out[:overlap,:,:]
                up = overlap_patch_up(A,B,overlap,patch_size)
                output[y-overlap:y,x-overlap:x+patch_size-overlap,:] = up

                #left
                A = left_patch
                B = out[:out_size - y ,:overlap,:]
                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:out_size,x-overlap:x,:] = left

                #leftover
                output[y:out_size,x:x+patch_size-overlap,:] = out[overlap:
out_size-y+overlap,overlap:,:]
                x = x + patch_size - overlap
                #print(y,x)

            else:
                up_patch = output[y-overlap:y,x-overlap:out_size,:]
                left_patch = output[y:out_size,x-overlap:x,:]
```

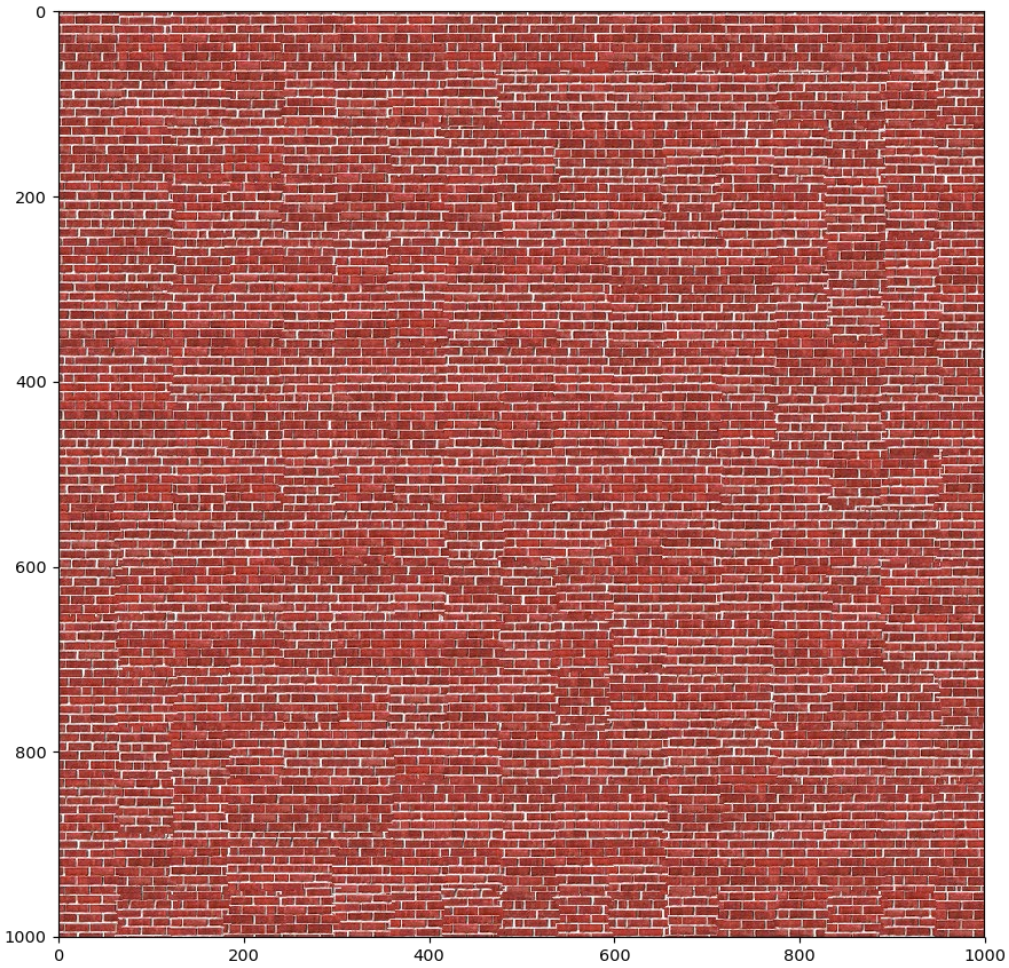
```
                ssd = ssd_patch_both(overlap,up_patch,left_patch,patch_siz
e,sample)
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #up
                A = up_patch
                B = out[:overlap,:out_size-x+overlap]
                up = overlap_patch_up(A,B,overlap,patch_size)
                output[y-overlap:y,x-overlap:out_size,:] = up

                #left
                A = left_patch
                B = out[overlap:out_size - y + overlap,:overlap,:]
                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:y+patch_size-overlap,x-overlap:x,:] = left

                #leftover
                output[y:out_size,x:out_size,:] = out[overlap:out_size-y+o
verlap,overlap:out_size-x+overlap,:]
                return output
```

In [20]:
```
sample_img_dir = 'samples/bricks_small.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
output = quilt_cut(sample_img, 1000, 69, 10, 400)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('bricks_small_seam_finding.png')
```



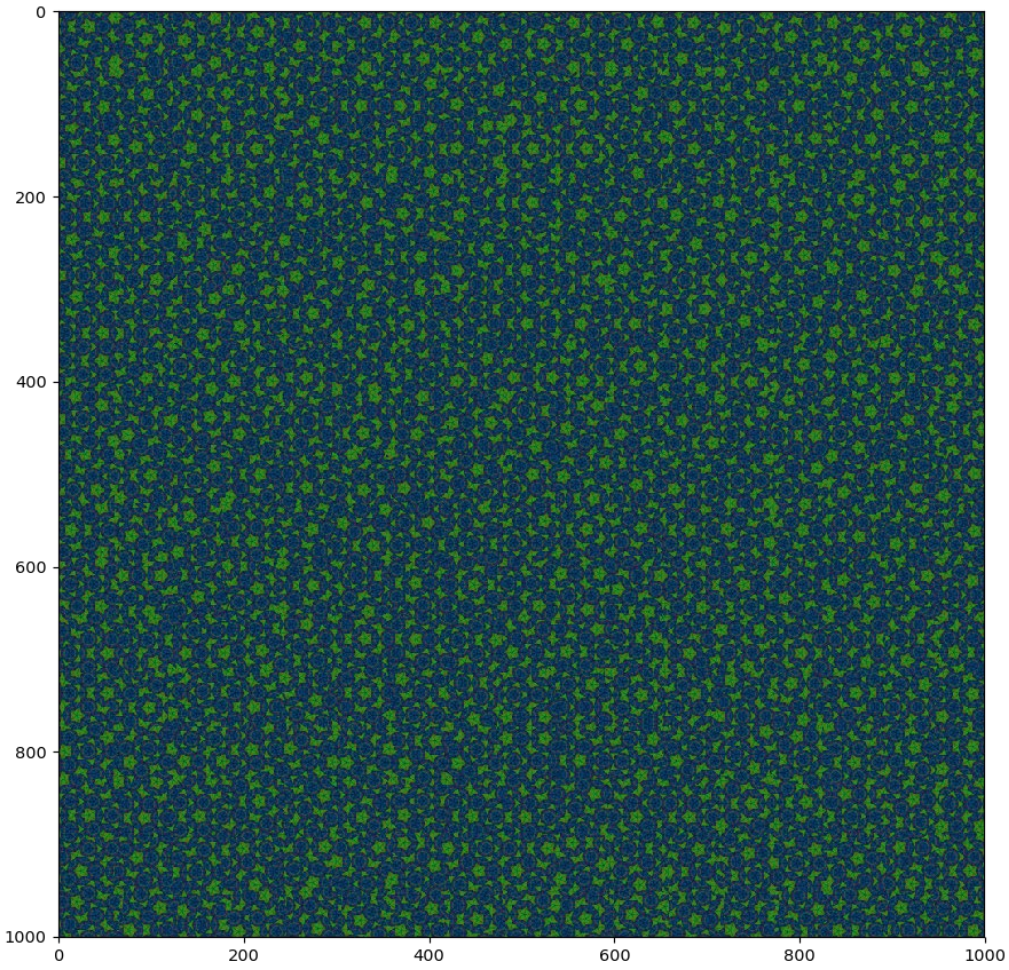
Out[20]: <matplotlib.image.AxesImage at 0x18eed6812b0>

In [21]:
```
sample_img_dir = 'samples/bricks.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
output = quilt_cut(sample_img, 1000, 69, 10, 400)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('bricks_seam_finding.png')
```



Out[21]: <matplotlib.image.AxesImage at 0x18eed689978>

In [22]:
```
sample_img_dir = 'samples/cool_pattern.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
output = quilt_cut(sample_img, 1000, 69, 10, 400)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('cool_pattern_seam_finding.png')
```

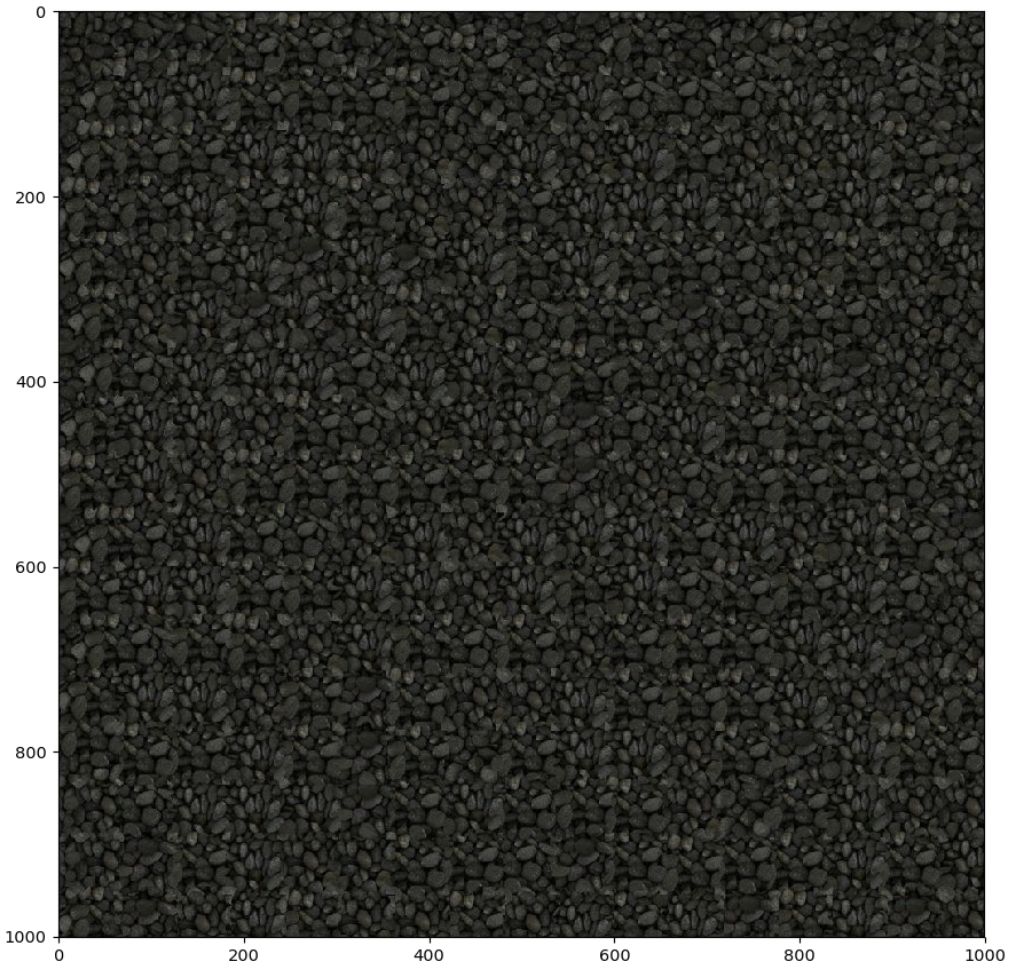


Out[22]: <matplotlib.image.AxesImage at 0x18ef02ddc50>

In [23]:
```
sample_img_dir = 'samples/wood.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
output = quilt_cut(sample_img, 1000, 69, 10, 400)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('wood_seam_finding.png')
```

Out[23]: <matplotlib.image.AxesImage at 0x18ef0ed7dd8>

In [24]:
```
sample_img_dir = 'samples/rocks.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
output = quilt_cut(sample_img, 1000, 69, 10, 400)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('rocks_seam_finding.png')
```



Out[24]: <matplotlib.image.AxesImage at 0x18ef1ae9160>

part IV: Texture Transfer (30 pts)

In [25]:
```
def ssd_patch_left(overlap,left_patch,patch_size,sample):
    M = np.zeros(shape=(patch_size,patch_size,3))
    M[:,:overlap,:] = 1
    T = np.zeros(shape=(patch_size,patch_size,3))
    T[:,:overlap,:] = left_patch[:, patch_size-overlap:patch_size,:] #temp
late overlap real values
    T/255.0
    I = sample/255.0
    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [26]:
```
def ssd_patch_up(overlap,up_patch,patch_size,sample):
```

```
    M = np.zeros(shape=(patch_size,patch_size,3))
    M[:overlap,:,:] = 1
    T = np.zeros(shape=(patch_size,patch_size,3))
    T[:overlap,:,:] = up_patch[patch_size-overlap:patch_size, :,:] #templa
te overlap real values
    T/255.0
    I = sample/255.0
    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [27]:
```
def ssd_patch_both(overlap,up_patch,left_patch,patch_size,sample):
    size_x = up_patch.shape[1]
    size_y = left_patch.shape[0] + overlap
    M = np.zeros(shape=(size_y,size_x,3))
    M[:overlap,:overlap,:] = 1
    T = np.zeros(shape=(size_y,size_x,3))
    T[:overlap,:,:] = up_patch #template overlap real values
    T[overlap:,:overlap,:] = left_patch
    T/255.0
    I = sample/255.0
    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [28]:
```
def overlap_patch_left(A,B,overlap,patch_size):
    #A is the left patch
    #B is the new patch
    row_size_1 = A.shape[0]
    col_size_1 = A.shape[1]
    s = (A[:,:,:]-B[:,:,:])**2
    s = s[:,:,0] + s[:,:,1] + s[:,:,2]
    s =  s.transpose()
    mask = cut(s)
    mask = mask.transpose()
    output_in = np.zeros(shape=(row_size_1,overlap,3))
    row = 0
    col = 0
    for x in mask:
        col = 0
        for y in x:
            if y == 0:
                output_in[row,col,:] = A[row,col,:]
            else:
                output_in[row,col,:] = B[row,col,:]
            #print(row,col)
            col += 1
        row += 1
    return output_in
```

In [29]:
```
def overlap_patch_up(A,B,overlap,patch_size):
    #A is the up patch
    #B is the new patch
    row_size_2 = A.shape[0]
    col_size_2 = A.shape[1]
    s = (A[:,:,:]-B[:,:,:])**2
```

```
    s = s[:,:,0] + s[:,:,1] + s[:,:,2]
    mask = cut(s)
    output_in = np.zeros(shape=(overlap,col_size_2,3))
    row = 0
    col = 0
    for x in mask:
        col = 0
        for y in x:
            if y == 0:
                output_in[row,col,:] = A[row,col,:]
            else:
                output_in[row,col,:] = B[row,col,:]
            #print(row,col)
            col += 1
        row += 1
    return output_in
```

In [30]:
```
def choose_sample(k,ssd,overlap,patch_size,input_image):
    input_image_size = input_image.shape[0]
    #print(input_image_size)
    row = []
    col = []
    z = []
    ssd_test = ssd
    while len(row) < k:
        row_1, col_1 = np.where(ssd_test == np.amin(ssd_test))
        if (patch_size-1)/2<row_1[0] < input_image_size-((patch_size-1)/2)
 and (patch_size-1)/2<col_1[0] < input_image_size-((patch_size-1)/2):
            row.append(row_1[0])
            col.append(col_1[0])
        ssd_test[row_1,col_1] = 10000000000000
    #print (row, col, z)
    rand_int = randint(0,k-1)
    return input_image[row[rand_int]-int(((patch_size-1)/2)):(row[rand_int
]-int((patch_size-1)/2))+patch_size,col[rand_int]-int((patch_size-1)/2):(c
ol[rand_int]-int((patch_size-1)/2))+patch_size,:]
```

In [31]:
```
def choose_sample(k,ssd,overlap,patch_size,input_image):
    input_image_size = input_image.shape[0]
    #print(input_image_size)
    row = []
    col = []
    z = []
    ssd_test = ssd
    while len(row) < k:
        row_1, col_1 = np.where(ssd_test == np.amin(ssd_test))
        if (patch_size-1)/2<row_1[0] < input_image_size-((patch_size-1)/2)
 and (patch_size-1)/2<col_1[0] < input_image_size-((patch_size-1)/2):
            row.append(row_1[0])
            col.append(col_1[0])
        ssd_test[row_1,col_1] = 10000000000000
    #print (row, col, z)
    rand_int = randint(0,k-1)
    return input_image[row[rand_int]-int(((patch_size-1)/2)):(row[rand_int
]-int((patch_size-1)/2))+patch_size,col[rand_int]-int((patch_size-1)/2):(c
ol[rand_int]-int((patch_size-1)/2))+patch_size,:]
```

In [32]:
```
def ssd_transfer_left(sample, target, patch_size, overlap, x, y,edge):
    sample_gray = cv2.cvtColor(sample, cv2.COLOR_RGB2GRAY)
    target_gray = cv2.cvtColor(target, cv2.COLOR_RGB2GRAY)
    sample_blur = cv2.GaussianBlur(sample_gray, ksize = (5,5), sigmaX = 3)
    target_blur = cv2.GaussianBlur(target_gray, ksize = (5,5),  sigmaX = 3
)

    col_target = target.shape[1]

    I = sample_blur/255.0
    if edge:
        T = target_blur[y:y+patch_size, x-overlap:col_target]/255.0
        M = np.ones(shape = (patch_size, col_target - x + overlap))
    elif x == 0 and y == 0:
        T = target_blur[y:y+patch_size, x:x+patch_size]/255.0
        M = np.ones(shape = (patch_size, patch_size))
    else:
        T = target_blur[y : y +patch_size, x - overlap : x + patch_size -
overlap] / 255.0
        #M = np.zeros(shape = (patch_size, patch_size))
        #M[:, :overlap] = 1
        M = np.ones(shape = (patch_size, patch_size))

    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [33]:
```
def ssd_transfer_up(sample, target, patch_size, overlap, x, y,edge):
    sample_gray = cv2.cvtColor(sample, cv2.COLOR_RGB2GRAY)
    target_gray = cv2.cvtColor(target, cv2.COLOR_RGB2GRAY)
    sample_blur = cv2.GaussianBlur(sample_gray, ksize = (5,5), sigmaX = 3)
    target_blur = cv2.GaussianBlur(target_gray, ksize = (5,5),  sigmaX = 3
)

    row_target = target.shape[0]

    I = sample_blur/255.0
    if edge:
        T = target_blur[y-overlap:row_target, x:x+ patch_size]/255.0
        M = np.ones(shape = (row_target - y + overlap, patch_size))
    else:
        T = target_blur[y-overlap:y + patch_size - overlap, x:x+ patch_siz
e]/255.0
        M = np.ones(shape = (patch_size, patch_size))

    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [34]:
```
def ssd_transfer_both(sample, target, patch_size, overlap, x, y,edge):
```

```
    sample_gray = cv2.cvtColor(sample, cv2.COLOR_RGB2GRAY)
    target_gray = cv2.cvtColor(target, cv2.COLOR_RGB2GRAY)
    sample_blur = cv2.GaussianBlur(sample_gray, ksize = (5,5), sigmaX = 3)
    target_blur = cv2.GaussianBlur(target_gray, ksize = (5,5),  sigmaX = 3
)

    row_target = target.shape[0]
    col_target = target.shape[1]

    I = sample_blur/255.0
    if edge == 0:
        T = target_blur[y-overlap:y + patch_size - overlap, x-overlap:x+ p
atch_size-overlap]/255.0
        M = np.ones(shape = (patch_size, patch_size))
    elif edge == 1:
        T = target_blur[y-overlap:y+patch_size -overlap, x-overlap:col_tar
get]/255.0
        M = np.ones(shape = (patch_size, col_target - x + overlap))
    elif edge == 2:
        T = target_blur[y-overlap:row_size, x-overlap:x+ patch_size-overla
p]/255.0
        M = np.ones(shape = (row_size -y + overlap, patch_size))
    elif edge == 3:
        T = target_blur[y:row_target, x:col_target]/255.0
        M = np.ones(shape = (row_target -y, col_target - x))



    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [37]:

```
def texture_transfer(sample,target,patch_size,overlap,alpha,k):
    """
    Feel free to add function parameters
    """
    row_size = target.shape[0]
    col_size = target.shape[1]
    output = np.zeros(shape=(row_size,col_size,3))


    y = 0
    while y < row_size:
        x = 0
        while x < col_size:
            if y == 0 and x == 0: #1
                #print(y,x,1)
                edge = False
                ssd = ssd_transfer_left(sample, target, patch_size, overla
p, x, y,edge)
                out = choose_sample(k,ssd,overlap,patch_size,sample)
                output[:patch_size,:patch_size,:] = out
                x = x + patch_size

            elif y ==0 and x + patch_size - overlap <= col_size: #2
                #print(y,x,2)
```

```
                edge = False
                #transfer ssd
                ssd_transfer = ssd_transfer_left(sample, target, patch_siz
e, overlap, x, y,edge)

                #overlap ssd
                left_patch = output[y:patch_size,x-patch_size:x,:]
                ssd_overlap = ssd_patch_left(overlap,left_patch,patch_size
,sample)
                ssd_overlap = ssd_overlap[:,:,0] + ssd_overlap[:,:,1] + ss
d_overlap[:,:,2]

                #total ssd
                ssd = (1-alpha)*(ssd_transfer) + (alpha) * ssd_overlap

                #sample patch
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #left overlap
                A = left_patch[:,patch_size-overlap:,:]
                B = out[:,:overlap,:]
                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:patch_size,x-overlap:x,:] = left

                #left over
                output[y:patch_size,x:x+patch_size-overlap,:] = out[:,over
lap:,:]
                x = x + patch_size - overlap

            elif y ==0 and x + patch_size - overlap > col_size: #3
                #print(y,x,3)
                edge = True

                #transfer ssd
                ssd_transfer = ssd_transfer_left(sample, target, patch_siz
e, overlap, x, y,edge)

                #overlap ssd
                left_patch = output[y:patch_size,x-patch_size:x,:]
                ssd_overlap = ssd_patch_left(overlap,left_patch,patch_size
,sample)
                ssd_overlap = ssd_overlap[:,:,0] + ssd_overlap[:,:,1] + ss
d_overlap[:,:,2]

                #total ssd
                ssd = (1-alpha)*(ssd_transfer) + (alpha) * ssd_overlap

                #sample patch
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #left overlap
                A = left_patch[:,patch_size-overlap:,:]
                B = out[:,:overlap,:]
                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:patch_size,x-overlap:x,:] = left

                #left over
```

```
                output[y:patch_size,x:col_size,:] = out[:,overlap:col_size
-x+overlap,:]
                x = col_size + 1
                y = y + patch_size


            elif y + patch_size - overlap <= row_size and x == 0:#4
                #print(y,x,4)
                edge = False
                #transfer ssd
                ssd_transfer = ssd_transfer_up(sample, target, patch_size,
 overlap, x, y,edge)

                #overlap ssd
                up_patch = output[y-patch_size:y,x:x+patch_size,:]
                ssd_overlap = ssd_patch_up(overlap,up_patch,patch_size,sam
ple)
                ssd_overlap = ssd_overlap[:,:,0] + ssd_overlap[:,:,1] + ss
d_overlap[:,:,2]

                #total ssd
                ssd = (1-alpha)*(ssd_transfer) + (alpha) * ssd_overlap

                #sample patch
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #up
                A = up_patch[patch_size-overlap:,:,:]
                B = out[:overlap,:,:]
                up = overlap_patch_up(A,B,overlap,patch_size)
                output[y-overlap:y,x:patch_size,:] = up

                #left over
                output[y:y+patch_size-overlap,x:patch_size,:] = out[overla
p:,:,:]
                x = x + patch_size


            elif y + patch_size - overlap <= row_size and x + patch_size -
 overlap <= col_size: #5
                #print(y,x,5)
                edge = 0

                #transfer ssd
                ssd_transfer = ssd_transfer_both(sample, target, patch_siz
e, overlap, x, y,edge)

                #overlap ssd
                up_patch = output[y-overlap:y,x-overlap:x+patch_size-overl
ap,:]
                left_patch = output[y:y+patch_size-overlap,x-overlap:x,:]
                ssd_overlap = ssd_patch_both(overlap,up_patch,left_patch,p
atch_size,sample)
                ssd_overlap = ssd_overlap[:,:,0] + ssd_overlap[:,:,1] + ss
d_overlap[:,:,2]

                #total ssd
```

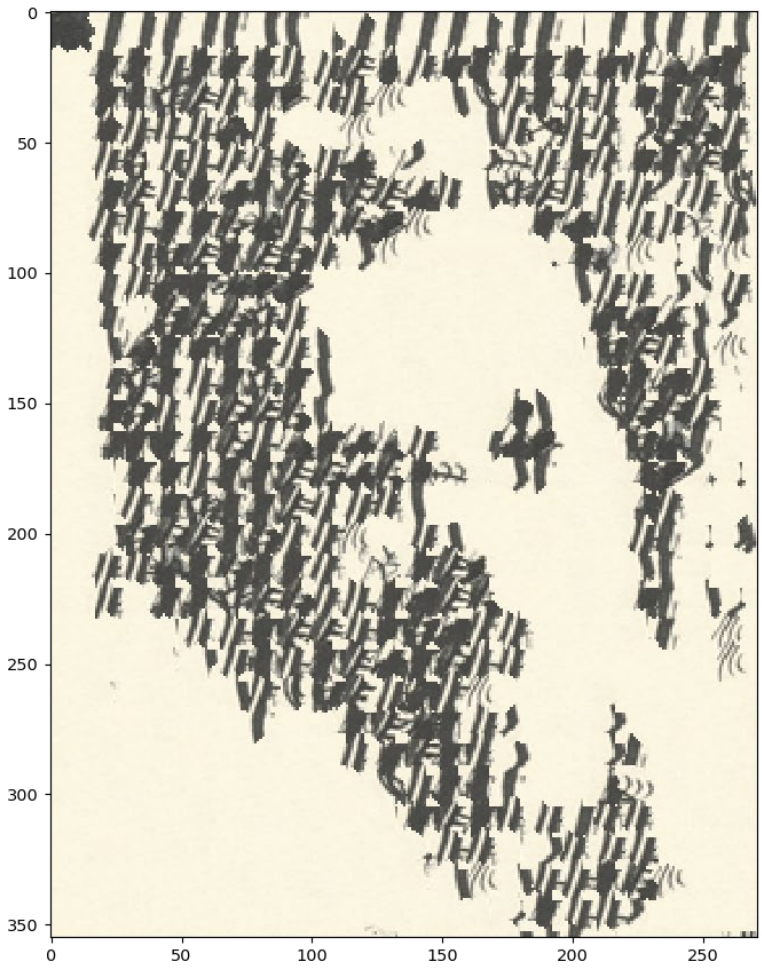
```
                ssd = (1-alpha)*(ssd_transfer) + (alpha) * ssd_overlap

                #sample patch
                out = choose_sample(k,ssd,overlap,patch_size,sample)


                #up
                A = up_patch
                B = out[:overlap,:,:]

                size_a_row = A.shape[0]
                size_a_col = A.shape[1]
                size_b_row = A.shape[0]
                size_b_col = A.shape[1]

                if size_a_col != size_b_col:
                    B_1 = np.zeros(shape=(size_a_row,size_a_col))
                    B_1[:size_b_row,:size_b_col] = B
                    B = B_1

                try:
                    up = overlap_patch_up(A,B,overlap,patch_size)
                except:
                    pass
                output[y-overlap:y,x-overlap:x+patch_size-overlap,:] = up

                #left
                A = left_patch
                B = out[overlap:,:overlap,:]


                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:y+patch_size-overlap,x-overlap:x,:] = left

                #leftover
                try:
                    output[y:y+patch_size-overlap,x:x+patch_size-overlap,:
] = out[overlap:,overlap:,:]
                except:
                    try:
                        rows_100 = output[y:y+patch_size-overlap,x:x+patch
_size-overlap,:].shape[0]
                        cols_100 = output[y:y+patch_size-overlap,x:x+patch
_size-overlap,:].shape[1]
                        rows_101 = out[overlap:,overlap:,:].shape[0]
                        cols_101 = out[overlap:,overlap:,:].shape[1]

                        out_1 = np.zeros(shape=(rows_100,cols_100))
                        out_1[:rows_101,:cols_101,:] = out[overlap:,overla
p:,:]
                        output[y:y+patch_size-overlap,x:x+patch_size-overl
ap,:] = out_1
                    except:
                        pass
```

```
                x = x + patch_size - overlap

            elif y + patch_size - overlap <= row_size and x + patch_size -
 overlap > col_size: #6
                #print(y,x,6)
                edge = 1

                #transfer ssd
                ssd_transfer = ssd_transfer_both(sample, target, patch_siz
e, overlap, x, y,edge)

                #overlap ssd
                up_patch = output[y-overlap:y,x-overlap:col_size,:]
                left_patch = output[y:y+patch_size-overlap,x-overlap:x,:]
                ssd_overlap = ssd_patch_both(overlap,up_patch,left_patch,p
atch_size,sample)
                ssd_overlap = ssd_overlap[:,:,0] + ssd_overlap[:,:,1] + ss
d_overlap[:,:,2]

                #total ssd
                ssd = (1-alpha)*(ssd_transfer) + (alpha) * ssd_overlap

                #sample patch
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #UP
                A = up_patch
                B = out[:overlap,:col_size-x+overlap]
                up = overlap_patch_up(A,B,overlap,patch_size)
                output[y-overlap:y,x-overlap:col_size,:] = up

                #left
                A = left_patch
                B = out[overlap:,:overlap,:]


                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:y+patch_size-overlap,x-overlap:x,:] = left

                #leftover
                output[y:y+patch_size-overlap,x:col_size,:] = out[overlap:
,overlap:col_size-x+overlap,:]
                x = col_size + 1
                y = y + patch_size - overlap

            elif y + patch_size - overlap > row_size and x == 0: #7
                edge = True
                #print(y,x,7)

                #transfer ssd
                ssd_transfer = ssd_transfer_up(sample, target, patch_size,
 overlap, x, y,edge)


                #overlap ssd
                up_patch = output[y-patch_size:y,x:x+patch_size,:]
                ssd_overlap = ssd_patch_up(overlap,up_patch,patch_size,sam
```

```
ple)
                ssd_overlap = ssd_overlap[:,:,0] + ssd_overlap[:,:,1] + ss
d_overlap[:,:,2]

                #total ssd
                ssd = (1-alpha)*(ssd_transfer) + (alpha) * ssd_overlap

                #sample patch
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                #UP
                A = up_patch[patch_size-overlap:,:,:]
                B = out[:overlap,:,:]
                size_a_row = A.shape[0]
                size_a_col = A.shape[1]
                size_b_row = A.shape[0]
                size_b_col = A.shape[1]

                if size_a_col != size_b_col:
                    B_1 = np.zeros(shape=(size_a_row,size_a_col))
                    B_1[:size_b_row,:size_b_col] = B
                    B = B_1


                up = overlap_patch_up(A,B,overlap,patch_size)
                output[y-overlap:y,x:patch_size,:] = up

                #left over
                output[y:row_size,x:x+patch_size,:] = out[overlap:row_size
-y+overlap,:,:]
                x = x + patch_size

            elif y + patch_size -overlap > row_size and x + patch_size <=
col_size: #8
                edge = 2
                #print(y,x,8)
                #transfer ssd
                ssd_transfer = ssd_transfer_up(sample, target, patch_size,
 overlap, x, y,edge)

                #overlap ssd
                up_patch = output[y-overlap:y,x-overlap:x+patch_size-overl
ap,:]
                left_patch = output[y:row_size,x-overlap:x,:]
                ssd_overlap = ssd_patch_both(overlap,up_patch,left_patch,p
atch_size,sample)
                ssd_overlap = ssd_overlap[:,:,0] + ssd_overlap[:,:,1] + ss
d_overlap[:,:,2]

                #total ssd
                ssd = (1-alpha)*(ssd_transfer) + (alpha) * ssd_overlap

                #sample patch
                out = choose_sample(k,ssd,overlap,patch_size,sample)
                #up
                A = up_patch
                B = out[:overlap,:,:]
```

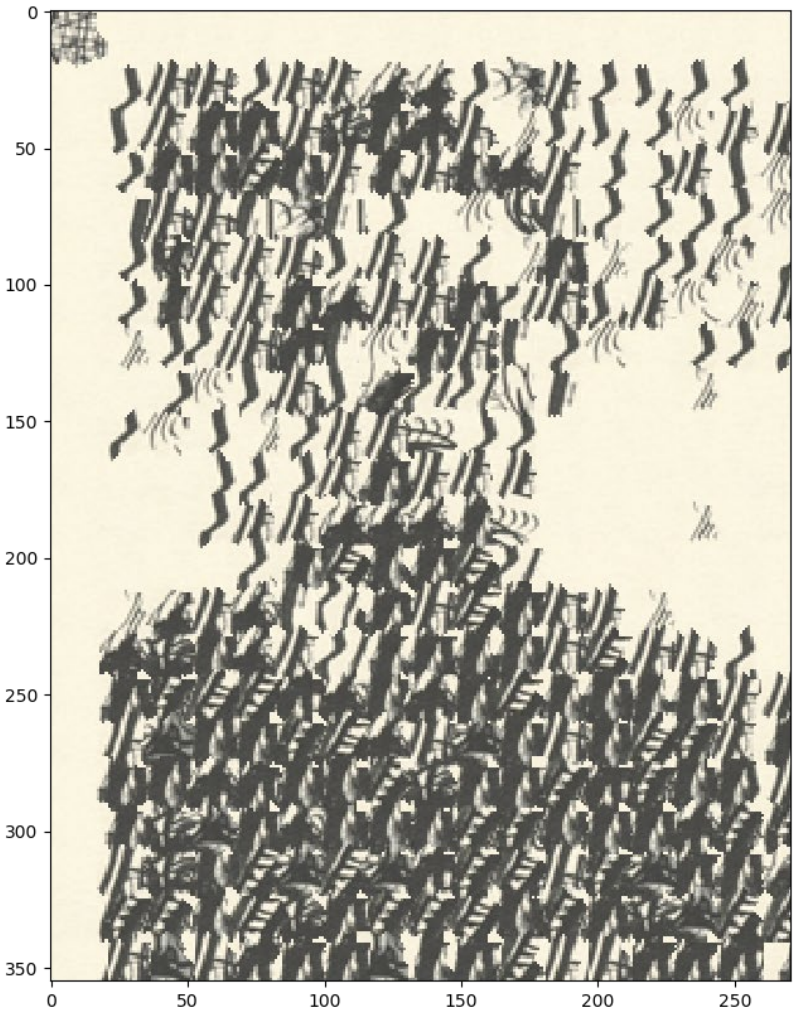
```
                size_a_row = A.shape[0]
                size_a_col = A.shape[1]
                size_b_row = A.shape[0]
                size_b_col = A.shape[1]

                if size_a_col != size_b_col:
                    B_1 = np.zeros(shape=(size_a_row,size_a_col))
                    B_1[:size_b_row,:size_b_col] = B
                    B = B_1


                up = overlap_patch_up(A,B,overlap,patch_size)
                output[y-overlap:y,x-overlap:x+patch_size-overlap,:] = up

                #left
                A = left_patch
                B = out[:row_size - y,:overlap,:]


                left = overlap_patch_left(A,B,overlap,patch_size)
                output[y:row_size,x-overlap:x,:] = left

                #left over
                output[y:row_size,x:x+patch_size-overlap,:] = out[overlap:
row_size-y+overlap,overlap:,:]
                x = x + patch_size - overlap




            else:
                #print(y,x,9)
                edge = 3
                #transfer ssd
                ssd_transfer = ssd_transfer_both(sample, target, patch_siz
e, overlap, x, y,edge)

                #overlap ssd
                up_patch = output[y-overlap:y,x-overlap:col_size,:]
                left_patch = output[y:row_size,x-overlap:x,:]
                ssd_overlap = ssd_patch_both(overlap,up_patch,left_patch,p
atch_size,sample)
                ssd_overlap = ssd_overlap[:,:,0] + ssd_overlap[:,:,1] + ss
d_overlap[:,:,2]

                #total ssd
                ssd = (1-alpha)*(ssd_transfer) + (alpha) * ssd_overlap

                #sample patch
                out = choose_sample(k,ssd,overlap,patch_size,sample)

                output[y:row_size,x:col_size,:] = out[:row_size - y, :col_
size - x,:]

                return output
```

In [38]:
```
target_dir = 'samples/feynman.tiff'
sample_dir = 'samples/sketch.tiff'
if os.path.exists(sample_dir):
    sample = cv2.imread(sample_dir)
    sample = cv2.cvtColor(sample,cv2.COLOR_BGR2RGB)
    target = cv2.imread(target_dir)
    target = cv2.cvtColor(target,cv2.COLOR_BGR2RGB)

output = texture_transfer(sample, target,17,5,.1,50)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('face_texture.png')
```



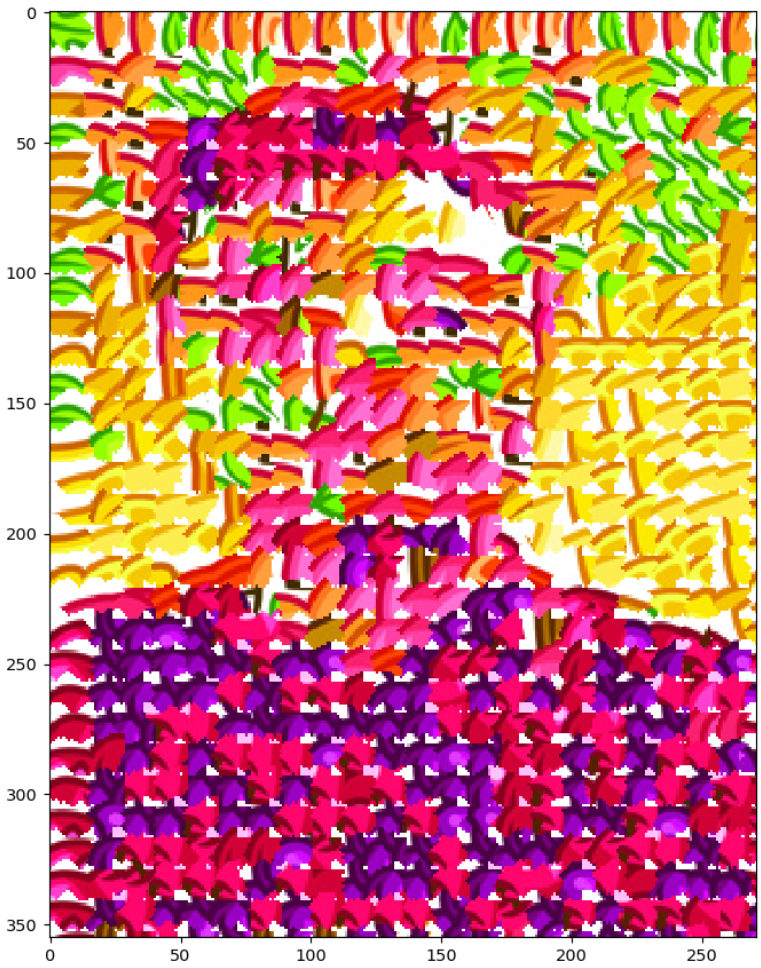
Out[38]: <matplotlib.image.AxesImage at 0x18eecbe5e80>

In [20]:
```
target_dir = 'samples/tarik_small.jpg'
sample_dir = 'samples/sketch.tiff'
if os.path.exists(sample_dir):
    sample = cv2.imread(sample_dir)
    sample = cv2.cvtColor(sample,cv2.COLOR_BGR2RGB)
    target = cv2.imread(target_dir)
    target = cv2.cvtColor(target,cv2.COLOR_BGR2RGB)

output = texture_transfer(sample, target,21,5,.1,50)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('tarik_texture.png')
```



In [39]:
```
target_dir = 'samples/tarik_small.jpg'
sample_dir = 'samples/fruit.jpg'
if os.path.exists(sample_dir):
```

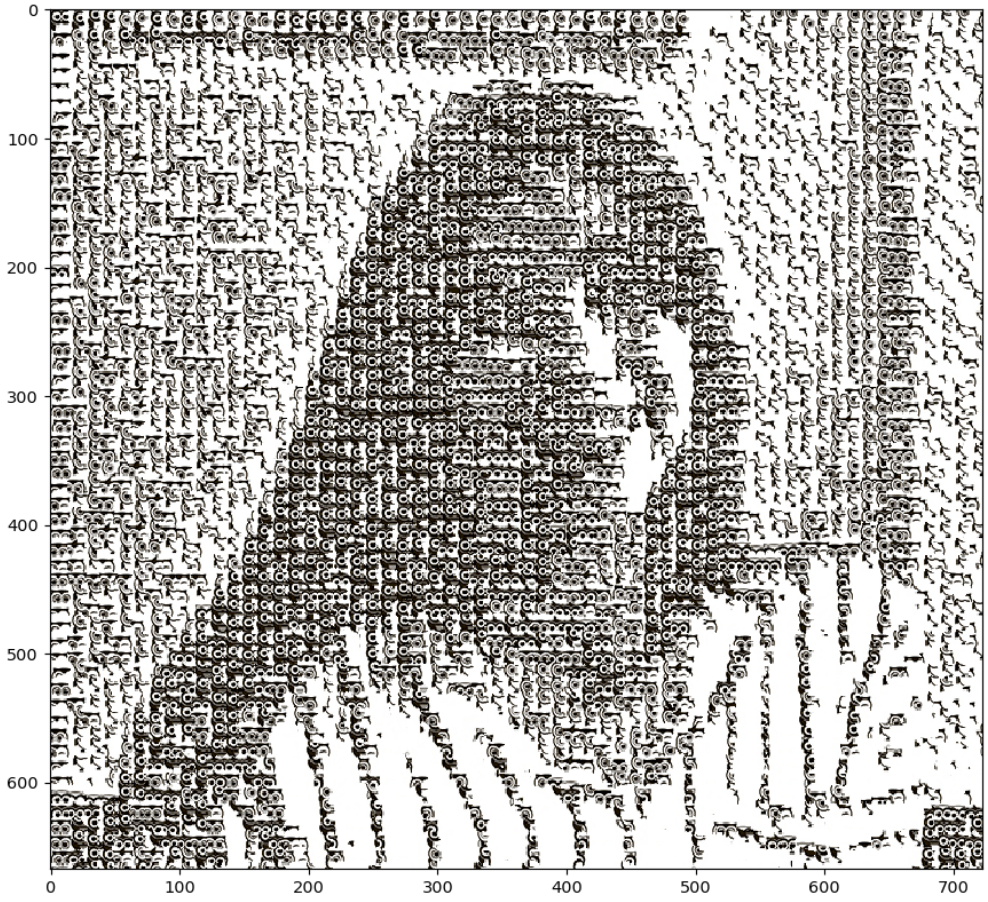
```
    sample = cv2.imread(sample_dir)
    sample = cv2.cvtColor(sample,cv2.COLOR_BGR2RGB)
    target = cv2.imread(target_dir)
    target = cv2.cvtColor(target,cv2.COLOR_BGR2RGB)

output = texture_transfer(sample, target,17,5,.1,50)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('tarik_texture_3.png')
```



Out[39]: <matplotlib.image.AxesImage at 0x18eeccb50b8>

In [40]:
```
target_dir = 'samples/samra.jpg'
sample_dir = 'samples/owls.jpg'
if os.path.exists(sample_dir):
    sample = cv2.imread(sample_dir)
    sample = cv2.cvtColor(sample,cv2.COLOR_BGR2RGB)
```

```
    target = cv2.imread(target_dir)
    target = cv2.cvtColor(target,cv2.COLOR_BGR2RGB)

output = texture_transfer(sample, target,17,5,.1,50)
output = output.astype('uint8')
fig, axes = plt.subplots(1, 1)
fig.set_size_inches(10, 10)
axes.imshow(output)
#plt.savefig('samra_owls.png')
```

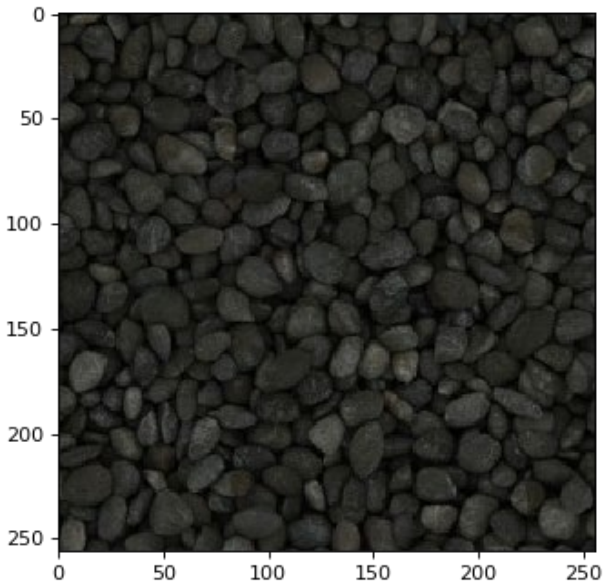


Out[40]: <matplotlib.image.AxesImage at 0x18eeceb6198>

Part V

In [8]:
```
#https://images-na.ssl-images-amazon.com/images/I/81p-%2Bkp6%2BRL._AC_SL15
00_.jpg bricks
#https://images.theconversation.com/files/133922/original/image-20160812-1
```

```
6339-v2g90o.png?ixlib=rb-1.1.0&q=45&auto=format&w=496&fit=clip cool patter
s
sample_img_dir = 'samples/rocks.jpg' # feel free to change
sample_img = None
if os.path.exists(sample_img_dir):
    sample_img = cv2.imread(sample_img_dir)
    sample_img = cv2.cvtColor(sample_img,cv2.COLOR_BGR2RGB)
    fig, axes = plt.subplots(1, 1)
    axes.imshow(sample_img)
```



In [10]:
```
def sample_image(width,sample_img):
    row = sample_img.shape[0]
    column = sample_img.shape[1]
    random_row = randint(0,row - width)
    random_column = randint(0,column - width)
    new = sample_img[random_row:random_row + width,random_column:random_co
lumn + width,:]
    return new
```

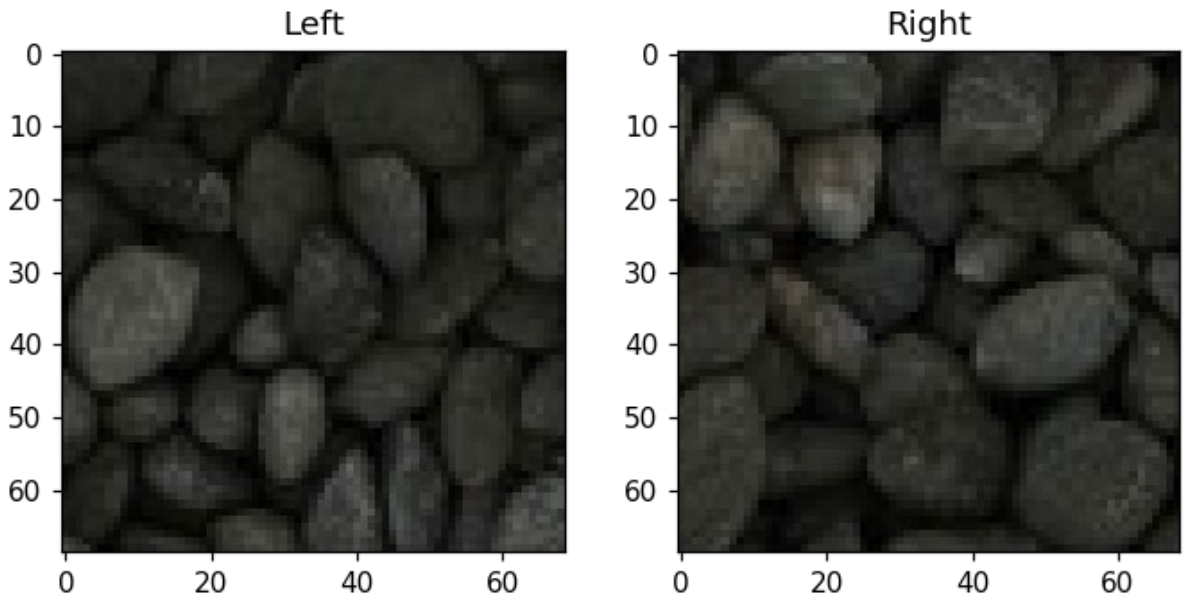
In [11]:
```
def ssd_patch_left(overlap,left_patch,patch_size,sample):
    M = np.zeros(shape=(patch_size,patch_size,3))
    M[:,:overlap,:] = 1
    T = np.zeros(shape=(patch_size,patch_size,3))
    T[:,:overlap,:] = left_patch[:, patch_size-overlap:patch_size,:] #temp
late overlap real values
    T/255.0
    I = sample/255.0
    ssd = ((M*T)**2).sum() - 2 * cv2.filter2D(I, ddepth=-1, kernel = M*T)
+ cv2.filter2D(I ** 2, ddepth=-1, kernel=M)
    return ssd
```

In [15]:
```
def choose_sample(k,ssd,overlap,patch_size,input_image):
    input_image_size = input_image.shape[0]
    #print(input_image_size)
```

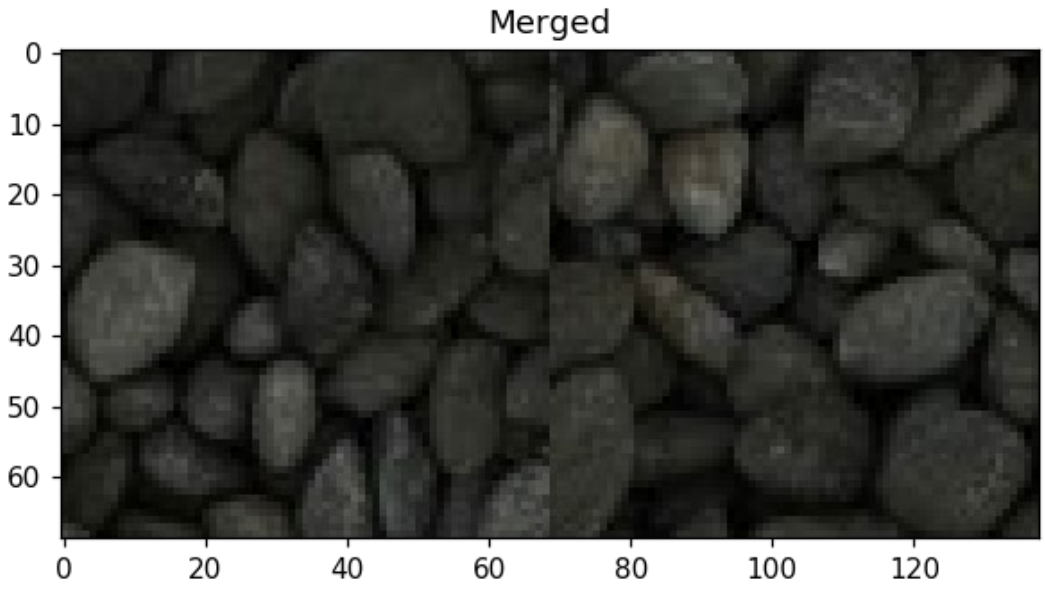
```
    row = []
    col = []
    z = []
    ssd_test = ssd[:,:,0]
    while len(row) < k:
        row_1, col_1 = np.where(ssd_test == np.amin(ssd_test))
        if (patch_size-1)/2<row_1[0] < input_image_size-((patch_size-1)/2)
 and (patch_size-1)/2<col_1[0] < input_image_size-((patch_size-1)/2):
            row.append(row_1[0])
            col.append(col_1[0])
        ssd_test[row_1,col_1] = 10000000000000
    #print (row, col, z)
    rand_int = randint(0,k-1)
    return input_image[row[rand_int]-int(((patch_size-1)/2)):(row[rand_int
]-int((patch_size-1)/2))+patch_size,col[rand_int]-int((patch_size-1)/2):(c
ol[rand_int]-int((patch_size-1)/2))+patch_size,:]
```

In [18]:
```
left_patch = sample_image(69,sample_img)
ssd = ssd_patch_left(10,left_patch,69,sample_img)
right_patch = choose_sample(400,ssd,10,69,sample_img)
```

In [19]:
```
fig, axes = plt.subplots(1,2)
fig.tight_layout()
axes[0].imshow(left_patch)
axes[1].imshow(right_patch)
#axes[0].axis('off')
#axes[1].axis('off')
#axes[2].axis('off')
axes[0].title.set_text('Left')
axes[1].title.set_text('Right')
```



In [22]:
```
merged = np.zeros(shape = (69,69*2,3))
merged[:,0:69,:] = left_patch
merged[:,69:,:] = right_patch
merged = merged.astype('uint8')
fig, axes = plt.subplots(1, 1)
axes.title.set_text('Merged')
axes.imshow(merged)
```



Out[22]: <matplotlib.image.AxesImage at 0x1fa52eba2e8>

In [29]:
```
def overlap_patch_left(A,B,overlap,patch_size):
    #A is the left patch
    #B is the new patch
    row_size_1 = A.shape[0]
    col_size_1 = A.shape[1]
    s = (A[:,:,:]-B[:,:,:])**2
    s = s[:,:,0] + s[:,:,1] + s[:,:,2]
    s =  s.transpose()
    mask = cut(s)
    mask = mask.transpose()
    output_in = np.zeros(shape=(row_size_1,overlap,3))
    row = 0
    col = 0
    for x in mask:
        col = 0
        for y in x:
            if y == 0:
                output_in[row,col,:] = A[row,col,:]
            else:
                output_in[row,col,:] = B[row,col,:]
            #print(row,col)
            col += 1
        row += 1
    return output_in,mask
```

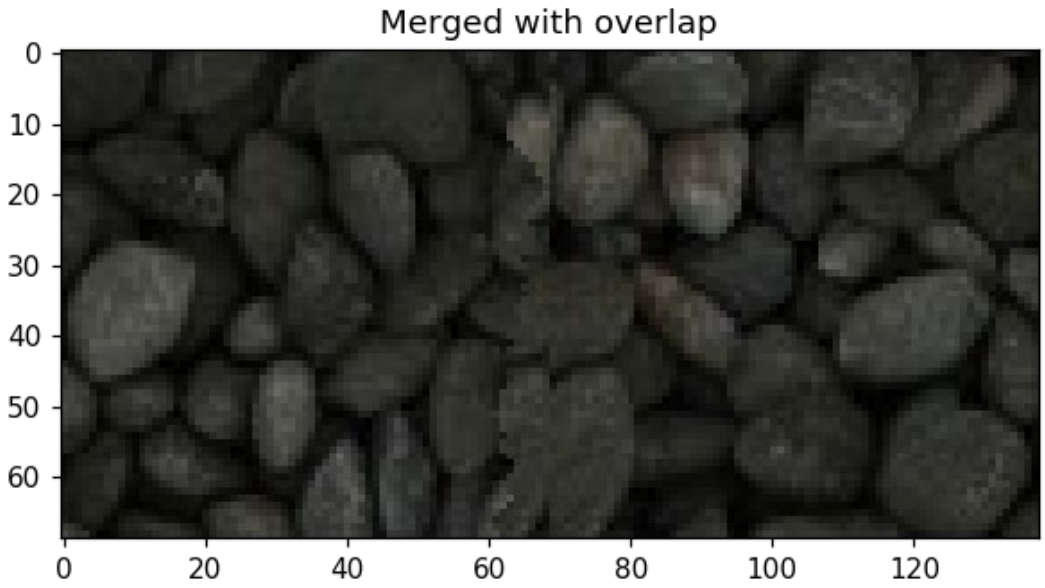
In [35]:
```
#plot over lap region
```

```
left_patch_intersect = left_patch[:,69-10:,:]
right_patch_intersect = right_patch[:,:10,:]
overlaped, mask = overlap_patch_left(left_patch_intersect,right_patch_inte
rsect,10,69)
mask
```

Out[35]:
```
array([[0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
       [0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
       [0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
```

```
[0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
[0, 0, 0, 0, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 1, 1, 1, 1, 1, 1, 1, 1],
[0, 0, 0, 1, 1, 1, 1, 1, 1, 1]])
```

In [36]:
```
merged[:,59:69,:] = overlaped
merged = merged.astype('uint8')
fig, axes = plt.subplots(1, 1)
axes.title.set_text('Merged with overlap')
axes.imshow(merged)
```

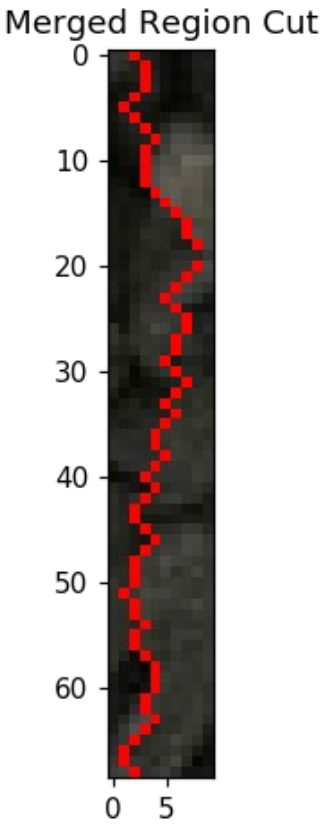


Out[36]: <matplotlib.image.AxesImage at 0x1fa52ee2fd0>

In [32]:
```
for i in range(mask.shape[0]):
```

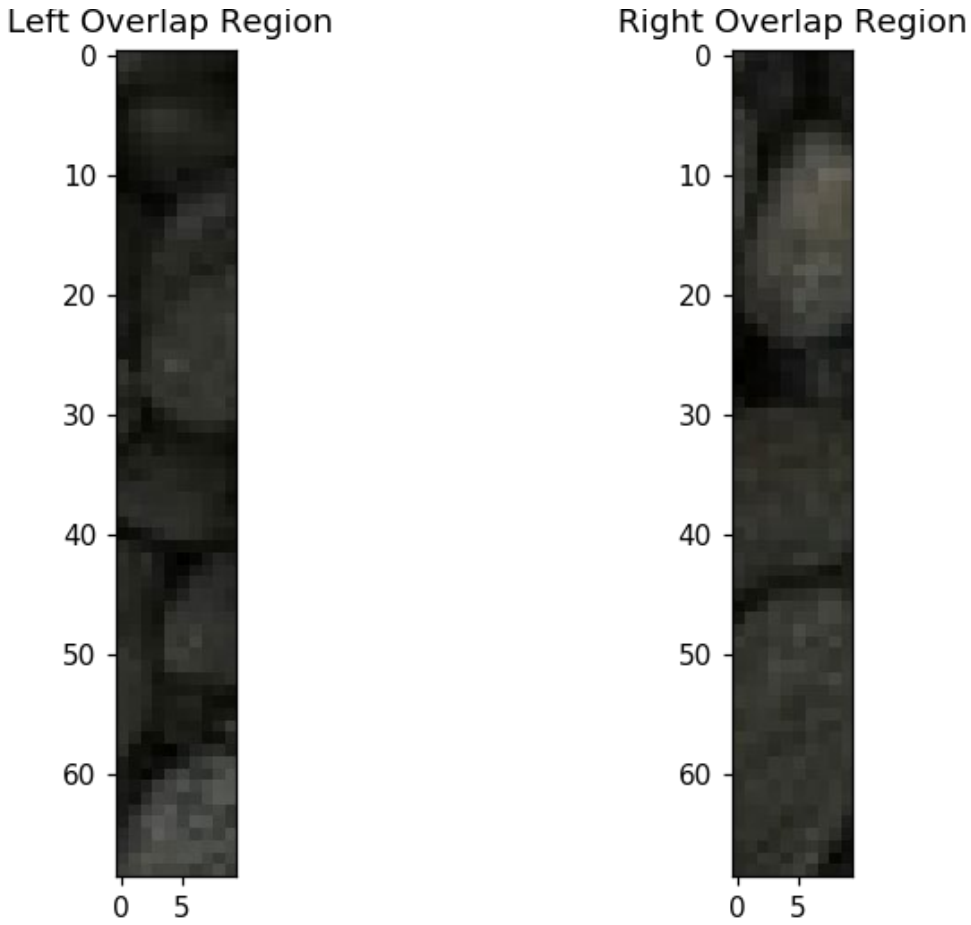
```
    for k in range(mask.shape[1]):
        if mask[i][k] == 1:
            overlaped[i, k-1, 0] = 255
            overlaped[i, k-1, 1] = 0
            overlaped[i, k-1, 2] = 0
            break
```

In [33]:
```
overlaped = overlaped.astype('uint8')
fig, axes = plt.subplots(1, 1)
axes.title.set_text('Merged Region Cut')
axes.imshow(overlaped)
```



Out[33]: <matplotlib.image.AxesImage at 0x1fa5455ee10>

In [34]:
```
fig, axes = plt.subplots(1,2)
fig.tight_layout()
axes[0].imshow(left_patch_intersect)
axes[1].imshow(right_patch_intersect)
#axes[0].axis('off')
#axes[1].axis('off')
#axes[2].axis('off')
axes[0].title.set_text('Left Overlap Region ')
axes[1].title.set_text('Right Overlap Region')
```



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

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

(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.

(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.).