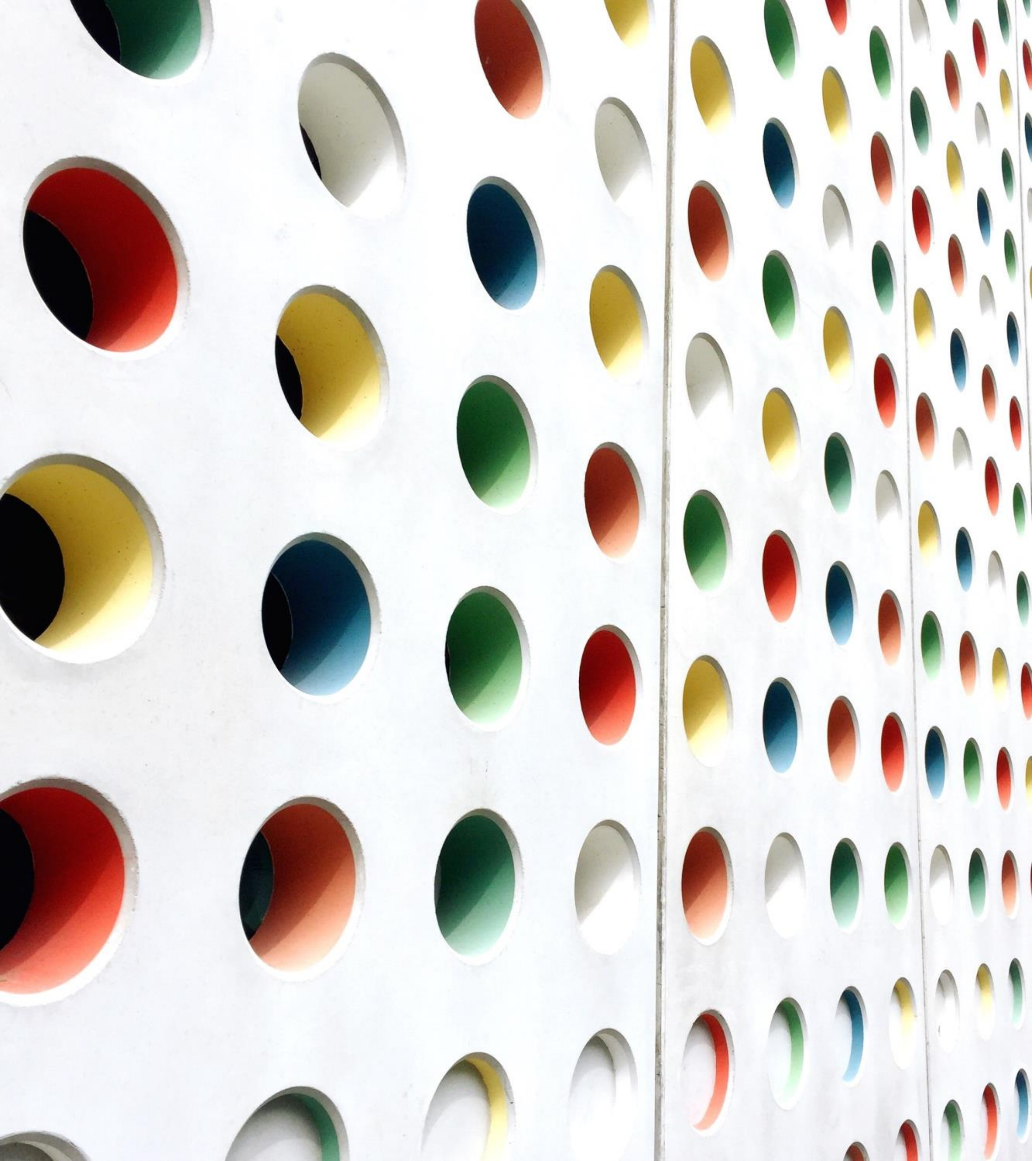


Computer Vision Homework 5

Matt Ferguson

August 10^{th}, 2022

Code Walkthrough 1

We iterate through each file set, and we iterate within each set by pairing images left to right. We load a 'left image' and 'right image' into memory. Between iterations of files within a file set our left image is replaced with the stitched output. The next image to be input to our collage replaces the right image. When a new set of files is loaded, we reset our working images. With this iteration structure in place, we declare a SIFT object, then locate keypoints and descriptors in our left and right images. We use brute force k-nearest neighbors to match keypoints and apply the ratio test to obtain good matches. We input our good matches into openCV's findHomography method and then we warp the right image into the plane of the left image using openCV's warpPerspective method. We superimpose the left image on top of a black image to add padding as needed. Our right image has the necessary black space as it is represented in the co-ordinate system of the left image.

```
#Computer Vision Homework 5 Matt Ferguson
import cv2
import numpy as np

files=['rio', 'blacksburg', 'diamondhead']
for file in files:

    # Loading
    img_1=cv2.imread(r'C:\Users\Matt\OneDrive\Virginia Tech\CV\Stitching\\'+file+'-00.png',0)
    img_2=cv2.imread(r'C:\Users\Matt\OneDrive\Virginia Tech\CV\Stitching\\'+file+'-01.png',0)
    img_3=cv2.imread(r'C:\Users\Matt\OneDrive\Virginia Tech\CV\Stitching\\'+file+'-02.png',0)

    imgs=[img_1,img_2,img_3]
    img_left=img_1

    for i in range(2):

        img_left_original_width=img_left.shape[1]
        img_right=imgs[i+1]

        # Feature Mapping
        sift_object=cv2.SIFT_create(500)
        kp_1, desc_1=sift_object.detectAndCompute(img_left, None)
        kp_2, desc_2=sift_object.detectAndCompute(img_right, None)
        bf=cv2.BFMatcher()
        bf_matches=bf.knnMatch(desc_1,desc_2, k=2)
        good = []
        for m,n in bf_matches:
            if m.distance < 0.75*n.distance:
                good.append(m)

        # Homography
        dst_pts=np.float32([ kp_1[m.queryIdx].pt for m in good ]).reshape(-1,1,2)
        src_pts = np.float32([ kp_2[m.trainIdx].pt for m in good ]).reshape(-1,1,2)
        M, mask = cv2.findHomography(src_pts,dst_pts,cv2.RANSAC,5.0)

        img_right=cv2.warpPerspective(img_right,M,((img_left.shape[1]+img_right.shape[1]),img_left.shape[0]))

        black_img=np.zeros(img_right.shape)
        black_img[0:img_left.shape[0],0:img_left.shape[1]]=img_left
        img_left=black_img

        # Locate right image first non zero column for blending
        c=0
        x=0
        y=int(img_right.shape[0]/2)
        while c == 0:
            x+=1
            if img_right[y,x]!=0:
                c=1
```

```
print(M)

[[  0.919    0.008 468.797]
 [ -0.032    0.975 -10.417]
 [ -0.       0.       1.   ]]
[[   0.876    0.027 1088.72 ]
 [  -0.066    1.021   -1.684]
 [  -0.       0.       1.   ]]
[[  1.       0.     466.   ]
 [  0.       1.     137.997]
 [  0.       0.       1.   ]]
[[  0.999    0.002 950.997]
 [ -0.       1.001  38.976]
 [ -0.       0.       1.   ]]
[[  0.889   -0.057 350.878]
 [  0.013    0.959 -17.678]
 [ -0.      -0.       1.   ]]
[[  0.738   -0.099 856.679]
 [  0.033    0.989 -95.193]
 [ -0.       0.       1.   ]]
```

Code Walkthrough 2

We then locate the seems of the left and right images by iterating from the black page edges until we find the first nonzero column at the vertical center (half height). We could have used co-ordinates from the homography matrix to more easily locate seams potentially. We use these seam points for blending down the seams. We weight a pixel value as a distance normed average between the left image value and the right image value for a location. Our weighting is 90% distance based and so a minimum 5% of each seam is contributed to a pixel value. We only apply this distance weighting between seams. In cases for a location where one image is black and the other nonblack, we use the value from the nonblack image at that output location. In cases where image values are equal, we use this equal value (rare). This scheme works well because outside the intersection between seams there is only a single nonblack image value. We tested extending our distance weighting some percentage out from the seam location, but this gave an inferior result. We finish by modest trimming of black columns to the right of the center of the final image and then writing our images. This trimming is for convenience of display and by no means comprehensive. Additional cropping could be performed to yield a professional result.

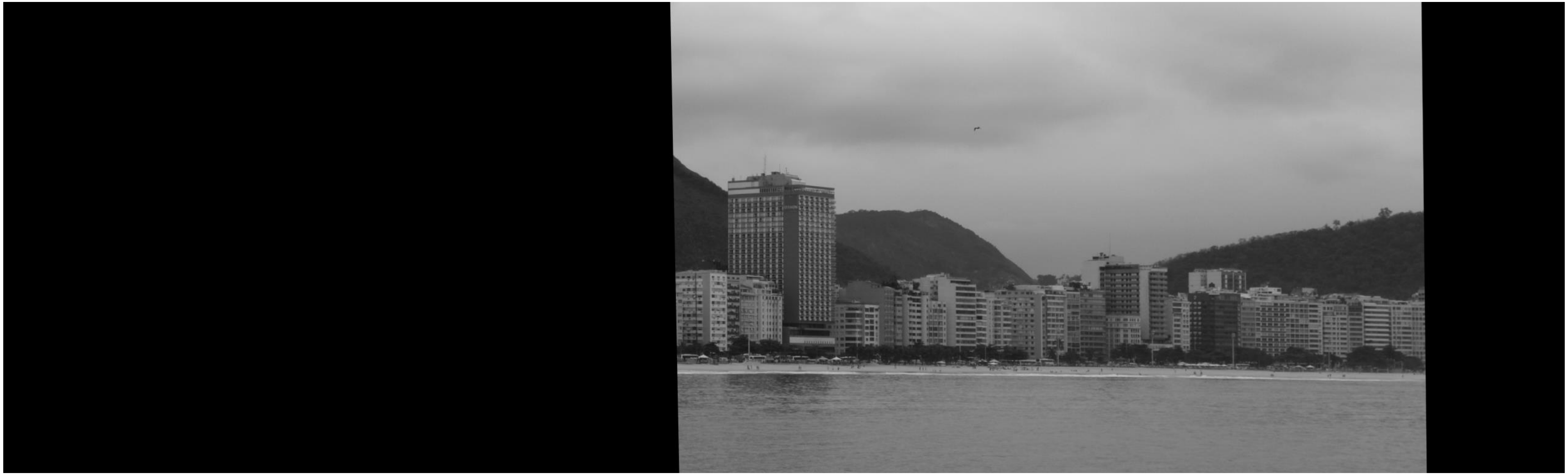
```
left_seam=x-10
# Locate left image first non zero column for blending
c=0
x=img_left.shape[1]
y=int(img_left.shape[0]/2)
while c==0:
    x-=1
    if img_left[y,x]!=0:
        c=1
right_seam=x+10

# Blend between first non-zero in right image (left seam) and left image original width (right seam)
output=img_right.copy()
w1=img_left.shape[1]
h1=img_left.shape[0]

output=cv2.GaussianBlur(output,(5,5),0)
for j in range(0,w1):
    for k in range(0,h1):
        if j>left_seam and j<right_seam and img_left[k,j]!=0 and img_right[k,j]!=0:
           output[k,j]= 0.9*((j-left_seam)/(right_seam-left_seam)*img_right[k,j]+(right_seam-j)/ \
                        (right_seam-left_seam)*img_left[k,j]) + 0.05*img_left[k,j]+0.05*img_right[k,j]
        else:
            if img_left[k,j] == img_right[k,j]:
                output[k,j]=img_left[k,j]
            if img_left[k,j] != img_right[k,j]:

                if img_left[k,j]!=0 and img_right[k,j]!=0:
                    output[k,j]=(img_left[k,j]+img_right[k,j])/2
                elif img_left[k,j] == 0:
                    output[k,j]=img_right[k,j]
                elif img_right[k,j] == 0:
                    output[k,j]=img_left[k,j]

# Locate Output right edge for trimming
c=0
x=output.shape[1]
y=int(output.shape[0]/2)
while c==0:
    x-=1
    if output[y,x]!=0:
        c=1

output=output[:,0:(x)]
output=cv2.GaussianBlur(output,(3,3),0)

cv2.imwrite(r'C:\Users\Matt\Desktop\Results\\'+file+'right'+str(i)+'.png',img_right)
cv2.imwrite(r'C:\Users\Matt\Desktop\Results\\'+file+'left'+str(i)+'.png',img_left)
cv2.imwrite(r'C:\Users\Matt\Desktop\Results\\'+file+'output'+str(i)+'.png',output)
img_left=output
```

Rio Warp 1

$$\text{Homographic Matrix} = \begin{bmatrix} 0.919 & 0.008 & 468.8 \\ -0.032 & 0.975 & -10.42 \\ -0. & 0. & 1. \end{bmatrix}$$

Rio Warp 2

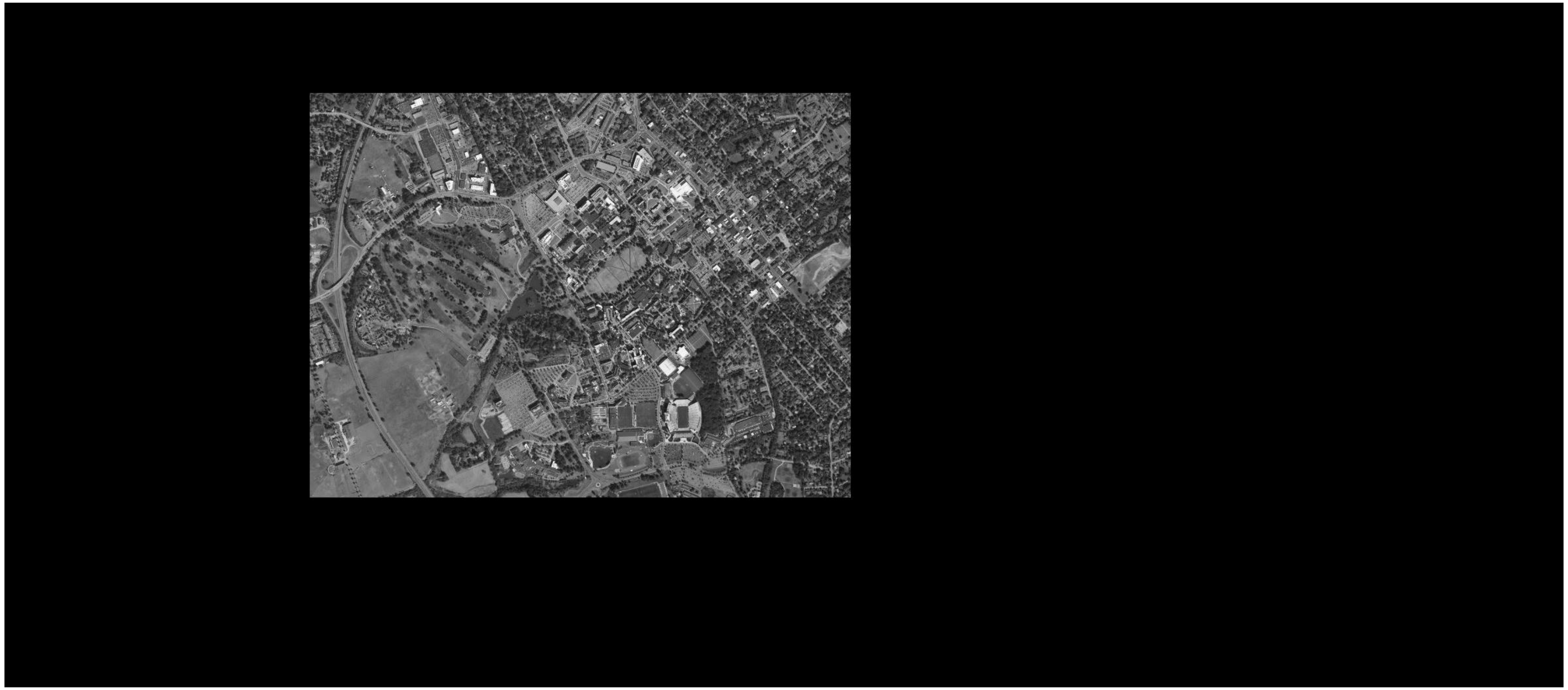
$$\text{Homographic Matrix} = \begin{matrix} [[& 0.876 & 0.027 & 1088.72\ ] \\ [& -0.066 & 1.02 & -1.68] \\ [& -0. & 0. & 1.\ ]] \end{matrix}$$



Rio Output

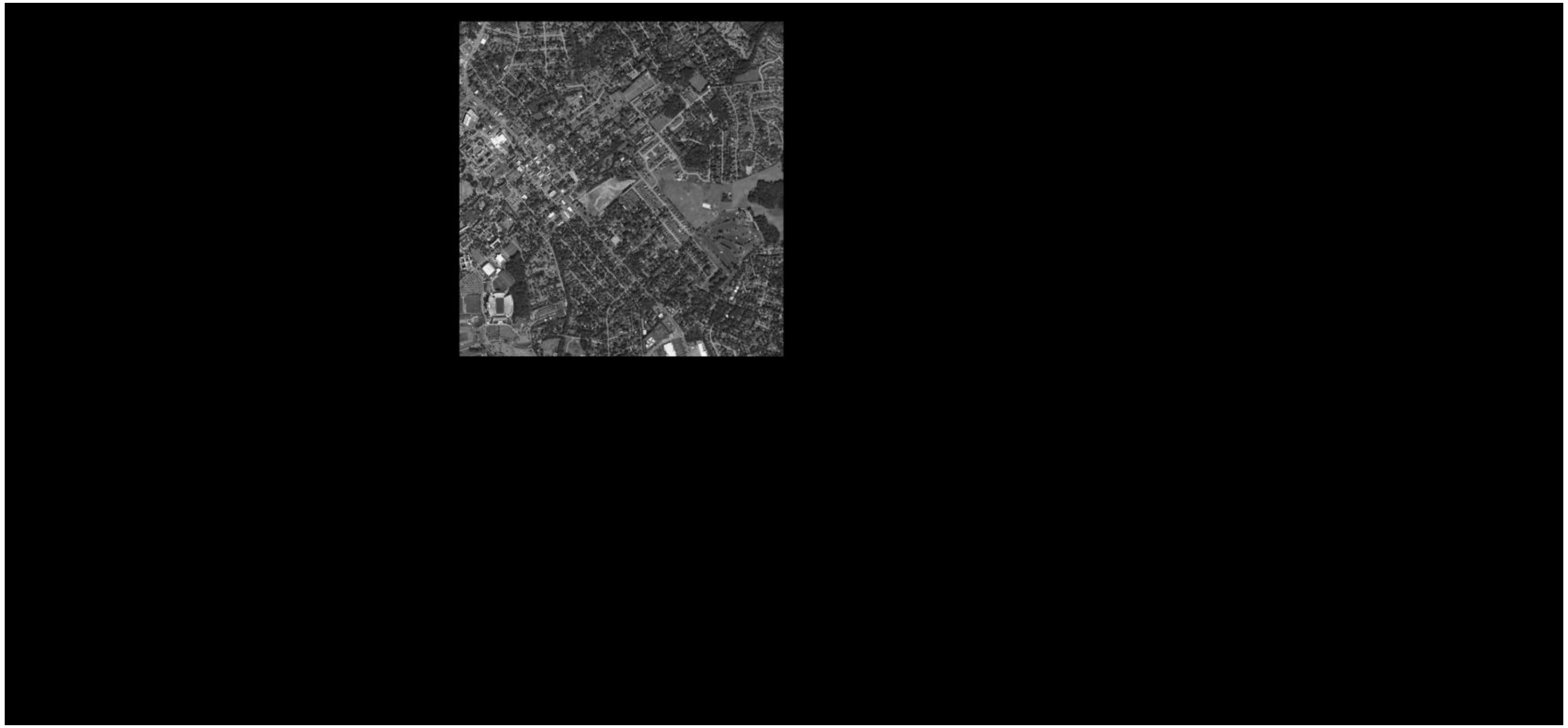
Blacksburg Warp 1

$$\text{Homographic Matrix} = \begin{bmatrix} 1. & 0. & 466. \\ 0. & 1. & 138. \\ 0. & 0. & 1. \end{bmatrix}$$



Blacksburg Warp 2

```
                              [[  1.     0.002   951.  ]
Homographic Matrix  =          [ -0.     1.    38.98]
                               [ -0.     0.     1.  ]]
```



Blacksburg Output

Diamondhead Warp 1

Homographic Matrix =
[[0.889 -0.057 350.88]
 [0.013   0.959 -17.68]
 [-0.   -0.   1.]]

Diamondhead Warp 2

Homographic Matrix =

```
[[  0.738  -0.099  856.68]
 [  0.033   0.989 -95.19]
 [ -0.     0.     1.  ]]
```

Diamondhead Output