# Hieu Nguyen

# GTID: 903185448

# Email: [hieu@gatech.edu](mailto:hieu@gatech.edu)

# 10/19/15

**CS-6475 Computational Photography Assignment 8**

For this panorama assignment, I utilized

Part 1: Programming the core of panoramas

1. getImageCorners()

This function returns the 4 corners of an input image. The output is a numpy.ndarray of shape (4, 1, 2), which contains the corners in (X, Y) format.

def getImageCorners(image):

corners = np.zeros((4, 1, 2), dtype=np.float32)

corners[1] = [0, image.shape[0]]

corners[2] = [image.shape[1], 0]

corners[3] = [image.shape[1], image.shape[0]]

return corners

1. findMatchesBetweenImages()

This function returns the top list of matches between two input images. It takes in two grayscale images and a desired number of matches as arguments, then outputs the corresponding image keypoints and a list of the matches. For this function, I just recycled my code from Assignment 7 on Feature Detection. It essentially uses ORB to get keypoints and descriptors, then performs a brute force technique to find matches.

def findMatchesBetweenImages(image\_1, image\_2, num\_matches):

orb = cv2.ORB()

image\_1\_kp, image\_1\_desc = orb.detectAndCompute(image\_1, None)

image\_2\_kp, image\_2\_desc = orb.detectAndCompute(image\_2, None)

bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)

matches = bf.match(image\_1\_desc, image\_2\_desc)

matches = sorted(matches, key=lambda x:x.distance)

return image\_1\_kp, image\_2\_kp, matches

1. findHomography()

This function returns the homography between the keypoints of image 1, image 2, and its matches. I iterate through each match to find the (X, Y) location of the keypoint for each match. Here, image 1 is the query image and image 2 is the train image. The cv2.findHomography() function is then used to generate a 3x3 homography matrix. This function uses RANSAC to eliminate outlier matches.

def findHomography(image\_1\_kp, image\_2\_kp, matches):

image\_1\_points = np.zeros((len(matches), 1, 2), dtype=np.float32)

image\_2\_points = np.zeros((len(matches), 1, 2), dtype=np.float32)

i = 0

for mat in matches:

image\_1\_points[i] = np.float32(image\_1\_kp[mat.queryIdx].pt)

image\_2\_points[i] = np.float32(image\_2\_kp[mat.trainIdx].pt)

i += 1

M\_hom, mask = cv2.findHomography(image\_1\_points, image\_2\_points, \

method=cv2.RANSAC, ransacReprojThreshold=5.0)

return M\_hom

1. warpImagePair()

This function warps image 1 so it can be blended with image 2 (stitched). First, the corners of both images are found. Next, the corners of image 1 are transformed using the calculated homography matrix. The min and max values for X and Y are calculated based off the transformed image 1 corners and the image 2 corners. A translation matrix is then created using (-1\*x\_min) and (-1\*y\_min). These values are multiplied by negative 1 because… The function cv2.warpPerspective() is then called using the the dot product of the homography and the translation matrix. The resulting warped image and image 2 are then stitched/blended together by calling blendImagePair().

def warpImagePair(image\_1, image\_2, homography):

image\_1\_corners = getImageCorners(image\_1)

image\_2\_corners = getImageCorners(image\_2)

image\_1\_corners\_t = cv2.perspectiveTransform(image\_1\_corners, homography)

join\_corners = np.append(image\_1\_corners\_t, image\_2\_corners, axis=0)

x\_min = np.amin(join\_corners[:,:,0])

x\_max = np.amax(join\_corners[:,:,0])

y\_min = np.amin(join\_corners[:,:,1])

y\_max = np.amax(join\_corners[:,:,1])

trans\_mat = np.array([[1, 0, -1 \* x\_min], [0, 1, -1 \* y\_min], [0, 0, 1]])

trans\_hom\_mat = np.dot(trans\_mat, homography)

warped\_image = cv2.warpPerspective(image\_1, trans\_hom\_mat, (x\_max-x\_min, y\_max-y\_min))

output\_image = blendImagePair(warped\_image, image\_2, (-1\*x\_min, -1\*y\_min))

return output\_image

Part 2: Get creative with programming and photography

1. blendImagePair()

This function takes an image that has been warped and an images that needs to be inserted into the warped image, using a point.

def blendImagePair(warped\_image, image\_2, point):

output\_image = np.copy(warped\_image)

output\_image[point[1]:point[1] + image\_2.shape[0],

point[0]:point[0] + image\_2.shape[1]] = image\_2

return output\_image

* rotational panorama
* planar panorama
* Brenizer method panorama (wedding photography)

This is what we want you to modify. We want you to write a function that blends the warped image with the second image at a given point. The simplest implementation is to average pixels where the images overlap. If you do this, try to incorporate a weight function from the start of the overlap point so that you get a 'fade' effect and properly transition from image 1 to image 2. You cannot modify the arguments of the function.

We want you to be creative here. Good blending may be difficult and time-consuming, and we do not expect you to implement a universally perfect and seamless blend. We want to see good effort and creative solutions. Feel free to compare and discuss different high-level approaches with other students.

This is what we want you to do for the PDF.

1. Demonstrate your 3+ Input Panorama images
2. Demonstrate the Panorama result.
3. Explain how you implemented your blendImagePair. We don't need you to explain the other functions but feel free to do so.

Answer the following questions:

* What effect did increasing the number of matched features have on your output? You can modify it either in the test file or on the bottom of assignment8.py where we provided some commented out code. Just make sure it is comment it out before submitting if you use that code.
* What type of panorama did you take?
* Were you happy with your result? If so / If not, why?