CS 378: 3D Reconstruction and Computer Vision

Sai Avala
Siena McFetridge
Jack Carlson

September 21, 2014

1 Introduction

This project required us to build a image stitcher that builds a panorama out of set of images. At the very minimum, our image stitcher was required to do the following:

- locate corresponding points between a pair of images
- use corresponding points to fit a homography (2D projective transform) that maps one into the space of the other
- use the homography to warp images into a common target frame, resizing and cropping as necessary
- composite several images in a common frame into a panorama

In order to complete the above requirements we implemented three functions *homography(), warp_image(), and create_mosaic()*

2 homography()

The main premise of our homography function is to return a 2D projective transformation matrix of one image onto another. The steps that we took to get this done are by reading in the images as grayscale images, identifying the key points using SIFT, and finally, establishing matches using the KNN algorithm. In order to find the best homography, we used RANSAC as our method to estimate the homgraphy between two images.

3 warp_image()

warp_image() takes in the homography as well as an image to transform and returns the warped image as well as the upper left origin point of the warped image. The first step step was to calculate the four corners *p1, p2, p3, and p4.* Next we took the dot product of each point against the homography matrix to calculate *p1', p2', p3', and p4'*. Given the new points, we calculated the minimum of the points in order to compute the new origin point. After completing that calculation, we used the new origin points as the x and y translation offsets and multiplied it against the homography matrix to get a new homography matrix. We then used the new adjusted homography matrix in order to warp the input image into the space given by the homography.

4 create_mosaic()

create_mosaic() is the final function that we had to implement for the panorama image stitcher. This function took in a list of images, calculated the final panorama image size by using the new origins, and then warping the images to scale at the panorama image size. The reason as to why we did this is because once each image was now the same size, it's easy to overlay them on top of each other. During the overlay process, if there was an intersecting region between two images, we just had the second image write over the pixels of the first image. At this point we would continue with the process until each image had been writen to the final image frame and was stitched as one final panorama.

5 Discussion

To truly test our panorama image stitcher, we took three pictures of a room in the GDC (Gates Dell Complex), wrote a script to find the homographies, warp the images, and finally create the mosaic image. The script is my_panos_stitcher.py. The script writes the final image as panorama.png.

We ran the script to produce two final images, a mean based blending approach and an image without any blending. There were some issues that arose when we attempted to use a blending approach for our set of images. The issue was that the light panel at the top of the image seems to be duplicated. However, the portion that was duplicated looks "washed" out, but the edges where each image overlaps looks fine, which is what the blender is supposed to accomplish. The final image that doesn't take blending into account of course shows the differences in color once all the images were stitched on top of eachother.