CS 378: 3D Reconstruction with Computer Vision

Sai Avala	EID: ssa766
Rene Garcia	EID: rrg696
Rohan Ramchand	EID: rsr898

October 13, 2014

1 Introduction

For this project, we were required to reconstruct a 3D scene from a stereo image ppairs. The steps for achieving this is as follows:

- Start off with two images
- Compute the Fundamental Matrix, and both homographies for both images by rectifying the pairs
- Once you've rectified each of the images, warp both of the images using their respective homographies
- Compute the disparity given both of the warped images and return the disparity
- Pass in the disparity, the left image, and a focal length into the *point_cloud()* function to return a string that writes a *.ply* file
- The *.ply* file is the 3D scene that is computed from the stereo image pairs and can be viewed in Meshlab

To complete the above steps, we implemented the following functions:
rectify_pair(), disparity_map(), point_cloud(). They are located in *stereo.py*. In addition, we have a script that computes the *.ply* file given a pair of stereo images - it's called *stereo_script.py*. You can run it using *run_stereo_script.sh*.

2 rectify_pair()

This function takes in two arguments: image_left and image_right. Both of the images that are passed in are 3-channel images, which make up a stereo pair. The function computes the pair's fundamental matrix and both rectifying homographies.

Essentially what we did to finish writing this function is by using cv2.SIFT() in order to compute the key points between both of the images. Then we used a FlannBasedMatcher using knn in order to store all the best matches according to Lowe's ratio test. Given this, we then ran cv2.findFundamentalMat() with the RANSAC algorithm in order to compute F, the fundamental matrix. Once we did that, we used cv2.stereoRectifyUncalibrated to return two homographies, one for each respectify stereo image that we passed in. Finally, we returned the fundamental matrix, a "homography left", and a "homography right".

3 disparity_map()

This function computes the disparity between two rectified images, image_left and image_right, and returns a single-channel image containing disparities between both of the images. In order to compute the disparity, we went used cv2.StereoSGBM(). We found that in order to achieve the best disparity that still passes the unit tests, we let the minimum disparity equal 16, the number of disparities be a function of the minimum disparities, and uniqueness ratio equal 10. Furthermore, we let the speckle window size equal 100 and the range equal 32. Once we computed the disparity, we had to divide that by 16 an then set the type of the disparity to be "uint8".

4 point_cloud()

This function creates a point cloud from the disparity image, image left, and focal length, and returns a string which contains the PLY point cloud of the3D locations of the pixels, with colors sampled from the left image. Essentially what we did was that we used cv2.reprojectImageto3D using the disparity image and a transformation matrix in order to compute the total set of points, and we used cv2.cvtColor() to compute the total set of colors. By comparing the disparity image and the minimum value of it, we stored that value in a mask variable, and passed that into the points and colors arrays we computed. We then generated a PLY header, used a StringIO object to write the data too, and returned the contents of that.

5 Discussion

While working on this project, we noticed a few essential items to be discussed. While testing out the project on our own set of images given two different camera points of view, we noticed that because of the rotation, the disparity function would not detect everything properly. This is most likely because of the fact that when the images are warped, there is the "black" empty data around it, which is being used in the computation of the disparity_map(). What would most likely help fix this issue is if given the warped images, compute an area inside the images where the objects overlap, which lets us compute a more accurate disparity. In addition, what we also noticed was that if we tried to tune the parameters in StereoSGBM as

much as possible, then we would fail the unit tests. Furthermore, while testing out our implementations, we noticed that in order to get the best possible PLY file, we would have to make sure that the background had lots of texture and the objects in the stereo images were bold, but not a shiny color.