CS 498: Assignment 2: Multi-view Geometry

Due by 11:59pm Friday, March 4th 2022

February 11, 2022

Submission

In this assignment you will code your own structure from motion and stereo matching algorithm, to convert the 2D observations to 3D structures. The starter code consists of 4 python files you will modify along with a folder of some images and saved data. Please put together a single PDF with your answers and figures for each problem, and submit it to Gradescope (Course Code: JBXJVZ). If your code produces figures they should go in this pdf, along with a description of what you did to solve the problem. We recommend you add your answers to the latex template files we provided. For code submission, make sure you use the provided ".py" files with your modification and the dumped ".npz" file. The graders will check both your PDF submission and your code submission if needed.

Keypoint Matching

Question 1 (Putative Matches)[2 pts]: Here you will be modifying "correspondence.py". Your task is to select putative matches between detected SIFT keypoints found in two images. Detecting keypoints will be done for you by cv2. The matches should be selected based on the Euclidean distance between the pairwise descriptors. In your implementation, make sure to filter your keypoint matches using Lowe's ratio test (link). For this question, you should implement your solution without using third-party functions e.g., cv2 knnmatch, which can significantly trivialize the problem. Meanwhile, please avoid solutions using for-loop which slows down the calculations. **Hint**: To avoid using for-loop, check out `scipy.spatial.distance.cdist (X,Y,'sqeuclidean')`.

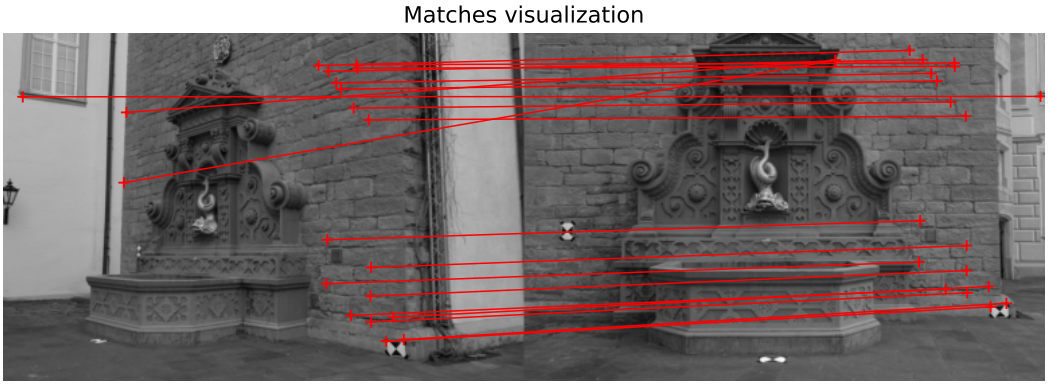

Figure 1: Lowe's Ratio Test Result

Fundamental Matrix Estimation

Question 2 (Eight-point Estimation) [3 pts]: Here you will be modifying "fundamental.py". For this question, your task is to implement the unnormalized eight-point algorithm (covered in class lecture) and normalized eight-point algorithm (link) to find out the fundamental matrix between two cameras. We provide code to compute the quality of your matrices based on the average geometric distance, i.e. the distance between each projected keypoint from one image to its corresponding epipolar line in the other image. Please report this distance in your pdf.

```
F_with_normalization average geo distance:  0.01657657398906439
F_without_normalization average geo distance:  0.023203213949373653
```

Question 3 (RANSAC) [3 pts]: Here you will be modifying "fundamental.py". Your task is to implement RANSAC to find the fundamental matrix between two cameras. Please report the average geometric distance based on your estimated fundamental matrix, given 1, 100, and 10000 iterations of RANSAC. Please also visualize the inliers with your best estimated fundamental matrix in your solution for both images (we provide a visualization function). In your PDF, please also explain why we do not perform SVD or do a least-square over all the matched key points.

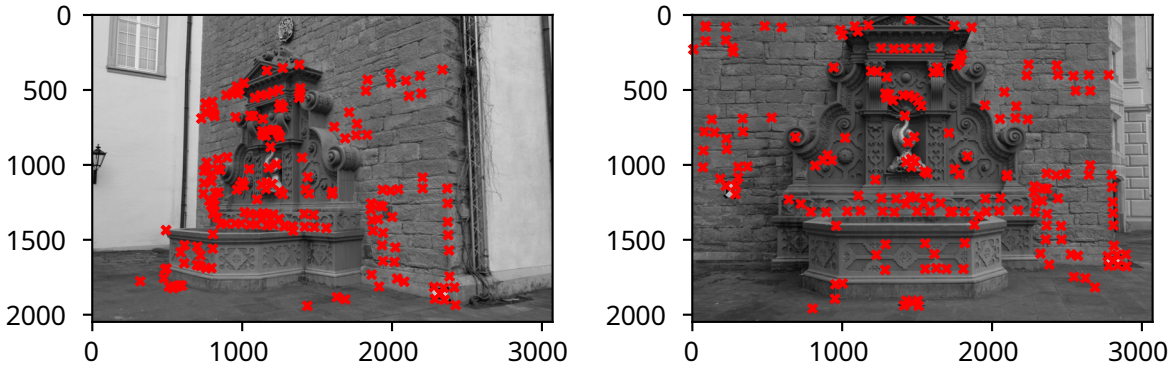

Figure 2: RANSAC Inliers with 1 Iteration
Best Average Geometric Distance: 0.1877

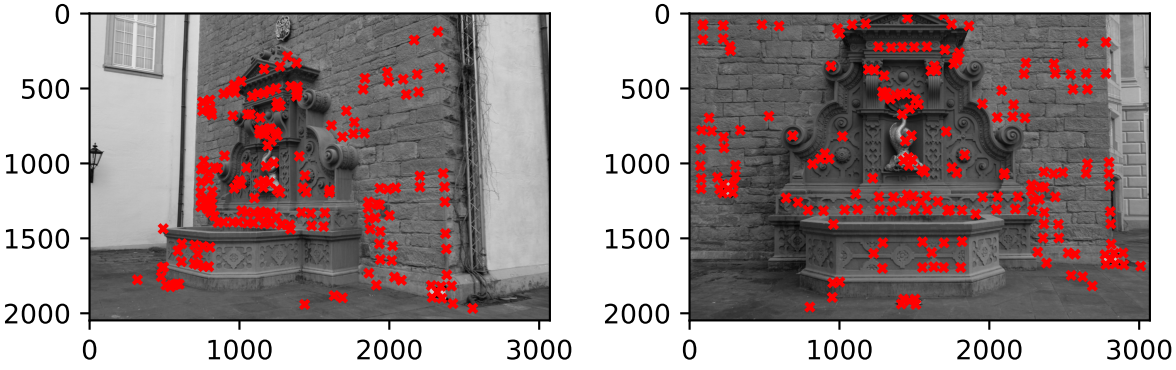

Figure 3: RANSAC Inliers with 100 Iterations
Best Average Geometric Distance: 0.0253

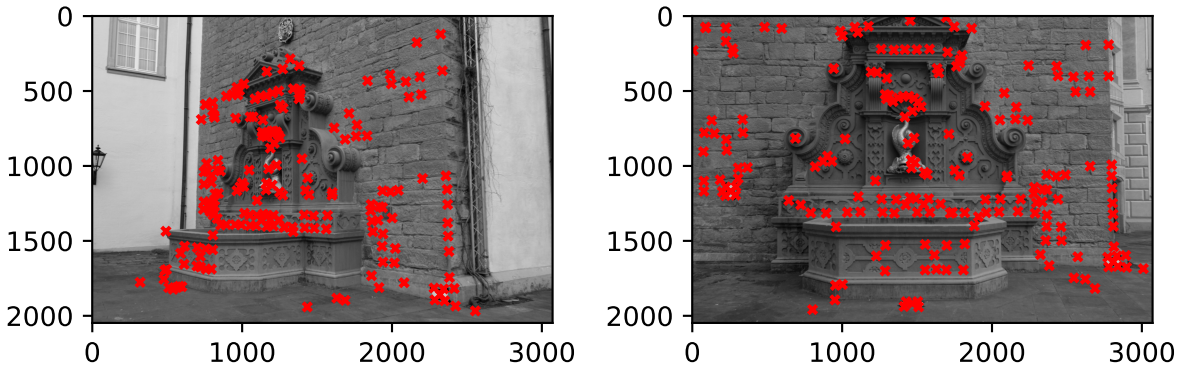

Figure 4: RANSAC Inliers with 10000 Iterations
Best Average Geometric Distance: 0.0202

Ordinary least-square is prune to noise and outlier. Because of the uncertainties in key point detection and matching, the matches could be very noise and contain some outliers. Performing SVD directly could produce an estimation that is heavily influenced by outliers.

Question 4 (Epipolar Line Visualization) [1 pt]: Here you will be modifying "fundamental.py". Please visualize the epipolar line for both images for your estimated F in Q2 and Q3. Here we do not provide you visualization code. To draw on images, cv2.line, cv2.circle are useful for plotting lines and circles. Check our Lecture 4, Epipolar Geometry, to learn more about equation of epipolar line. This link also gives a thorough review of epipolar geometry.

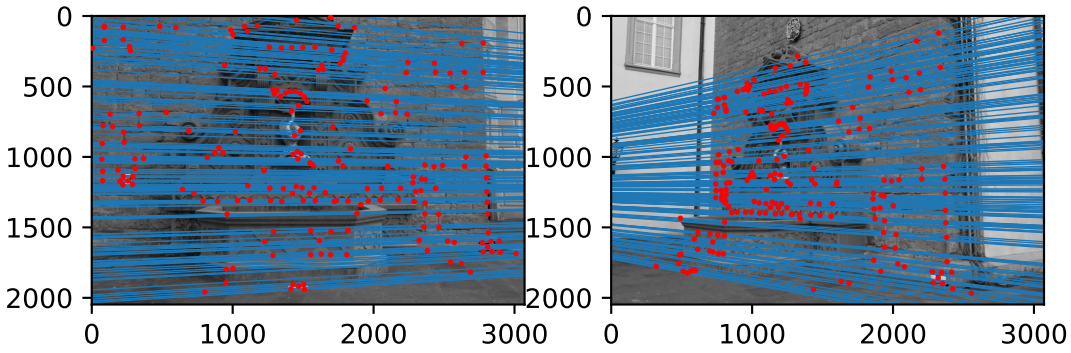

Figure 5: Q2 Eight-point Estimation w/o Normalization

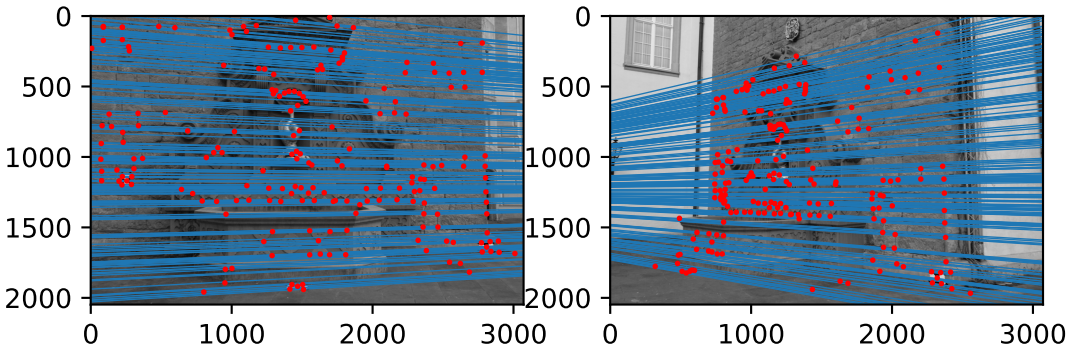

Figure 6: Q2 Eight-point Estimation with Normalization

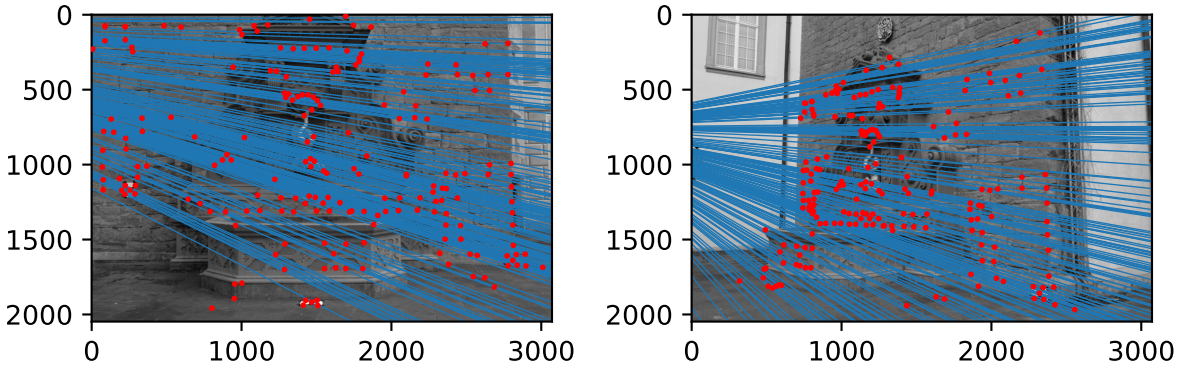

Figure 7: Q3 RANSAC Estimation with 1 Iteration

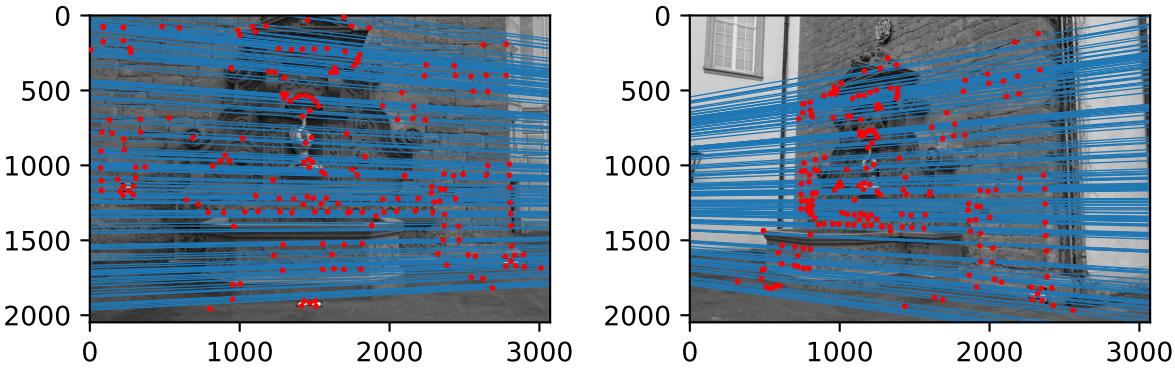

Figure 8: Q3 RANSAC Estimation with 100 Iteration

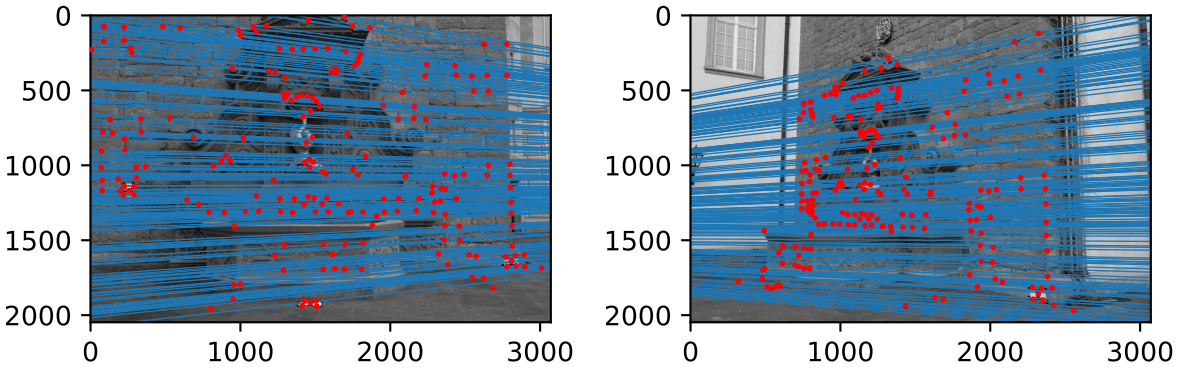

Figure 9: Q3 RANSAC Estimation with 10000 Iteration

Unprojection

After solving relative poses between our two cameras, now we will move from 2d to 3d space.

Question 5 (Triangulation) [3 pts]: Here you will be modifying "triangulation.py". You are given keypoint matching between two images, together with the camera intrinsic and extrinsic matrix. Your task is to perform triangulation to restore the 3D coordinates of the key points. In your PDF, please visualize the 3d points and camera poses in 3D from three different viewing perspectives.

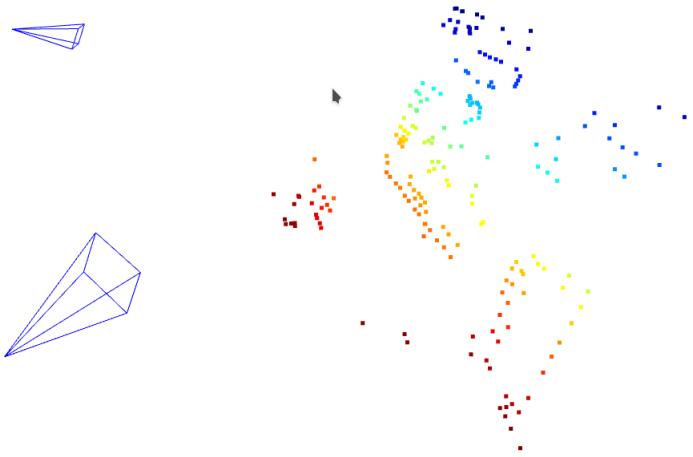

Figure 10: Triangulated Point Cloud View 1

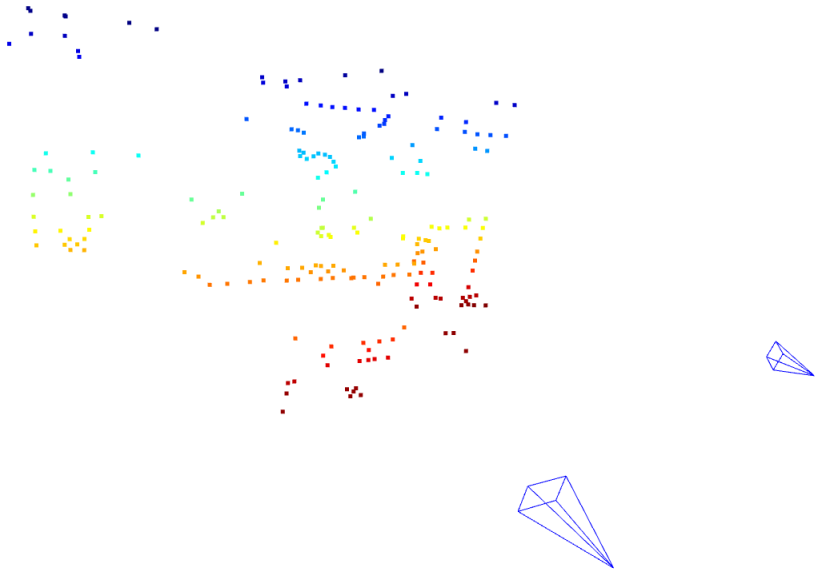

Figure 11: Triangulated Point Cloud View 2

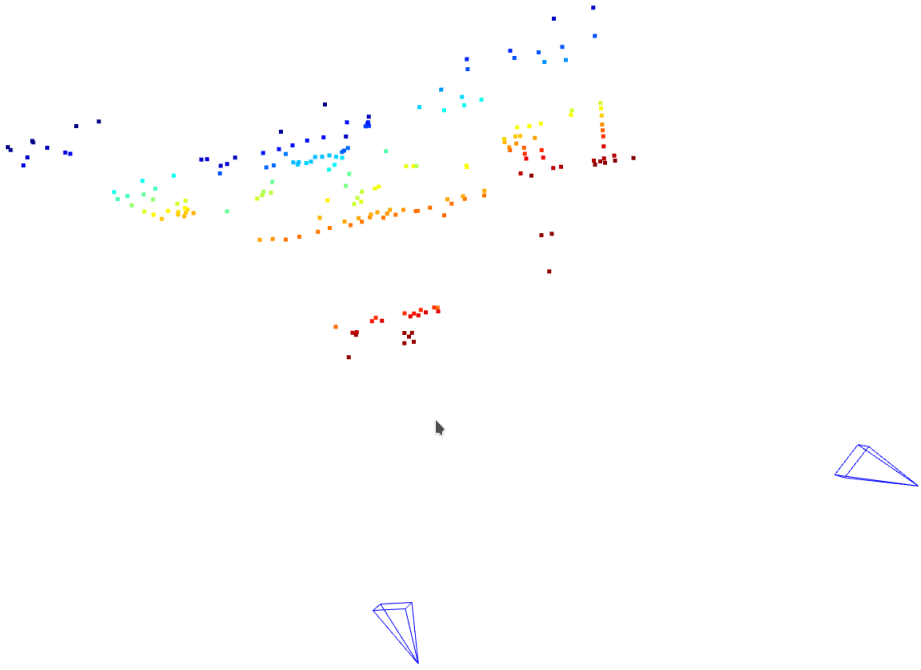

Figure 12: Triangulated Point Cloud View 3

Stereo Estimation

Now given two camera poses, we could get the relative camera pose and a sparse set of 3D point based on sparse keypoint matching. Could we go further and recover the dense geometry? The answer is yes. In question 6, you will reason a dense point cloud from a pair of images taken at the different views.

Question 6 (Stereo) [3 pts + 1 bonus pt]: Given two images $(\mathbf{I}_1, \mathbf{I}_2)$ with ground-truth intrinsic matrices $\mathbf{K}_1, \mathbf{K}_2$, extrinsic matrices $\mathbf{R}_1, \mathbf{t}_1, \mathbf{R}_2, \mathbf{t}_1$. Our goal is to recover a dense correspondence between the two images using stereo matching.

Computing the dense matching is computationally expensive. We down-sample both images by s times to reduce the computational burden. Could you describe what will be the updated intrinsic matrices? Please report the intrinsics on your write-up and fill in the code to update $\mathbf{K}_1, \mathbf{K}_2$.

The updated intrinsic matrix is the old one divided by the scale s.

$$\mathbf{K}_1 = \mathbf{K}_2 = \begin{bmatrix} 344.93500 & 0 & 190.08625 \\ 0 & 345.52000 & 125.85125 \\ 0 & 0 & 0.12500 \end{bmatrix}$$

We do notice that the pair of images are not perfectly fronto-parallel. To make matching easier, we will rectify the pair of images such that its epipolar lines are always horizontal. To achieve this, we simply call `cv2.StereoRectify` function. The updated projection matrices are also returned. Please go through the code and understand what each step does. You do not need to write any code here.

Finally, we will do stereo matching. Specifically, given a pixel at (i, j) on the left image, we will compute a winner-take-all disparity value that minimizes the sum of square distance (SSD) between a pair of left and right image patches, centered at (i, j) and $(i, j - d)$:

$$d[i, j] = \arg\min_d \sum_{m=-w}^{w} \sum_{n=-w}^{w} (I_1[i + m, j + n] - I_2[i + m, j + n - d])^2$$

where $w * 2 + 1$ is the block size and w is an integer. Please do this in a sliding window manner to get the dense disparity map. Please visualize the disparity map you get and include it in your report.

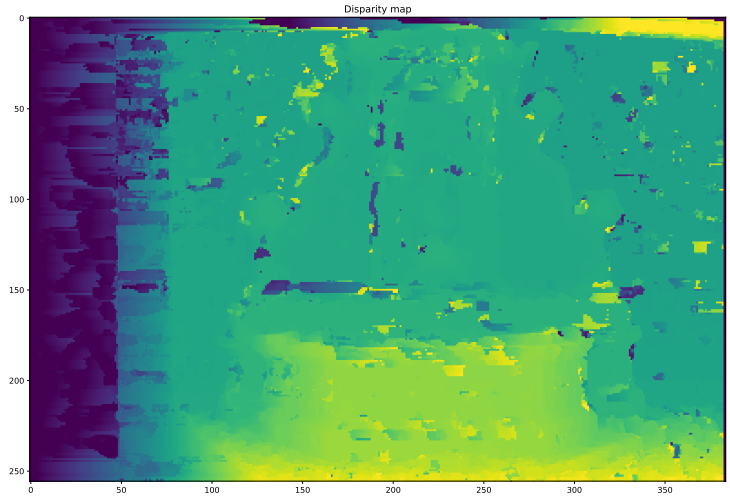

Figure 13: SSD Disparity Map

Do you think the disparity you get is satisfactory? You could compare your results against stereo matching results implemented by an off-the-shelf method `cv2.stereoBM` (we already provide the code).

Visualize both methods side-by-side and provide insights into further improving your results. Given disparity map estimation, could you get the dense point cloud? Try to get the point cloud and visualize your point cloud. (Bonus 1 pt)

No, the result is not satisfactory. The result I got from SSD is too coarse. I should further finetune the window size and try a smaller one. Normalized correlation could also be used as matching cost to account for the difference in lighting.]

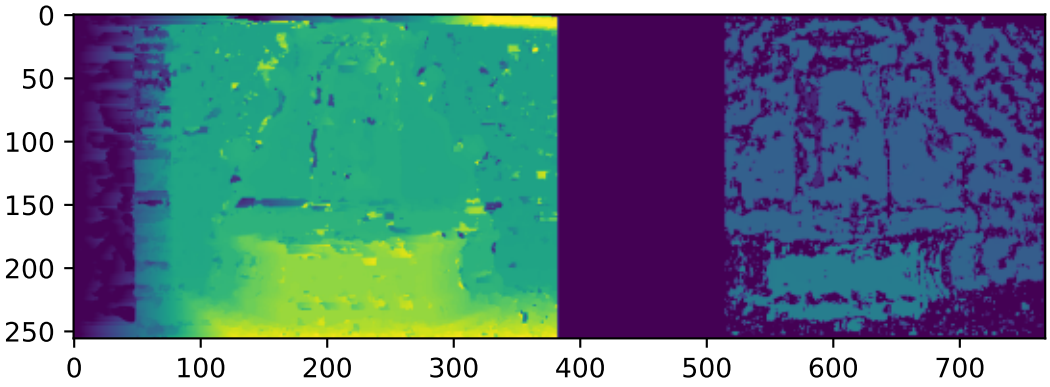

Figure 14: SSD Disparity Map (Left) Compared with OpenCV (Right)

Below is the point cloud I got from the depth map.

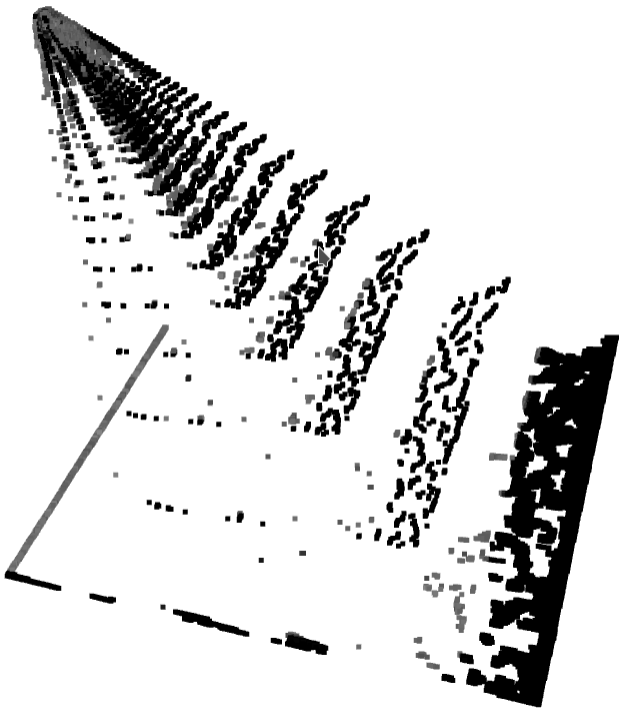

Figure 15: Point Cloud from Depth Map

Question 7 (Structure-from-Motion) [bonus 3 pts]: We believe you have a thorough understanding of multi-view geometry now. This bonus question will expand the two-view structure-from-motion and stereo estimation problem into multiple views. You have two directions to win the three bonus points: 1) Write your own mini bundle adjustment solver and run it on the fountain dataset. You could leverage Ceres, g2o, GTSAM libraries as your non-linear least square solver, but you need to derive the Jacobian of the energy terms and write your own code to initialize camera poses and 3D points, based on two-view geometry; 2) Collect your own data and utilize Colmap (https://demuc.de/colmap/) to run the full MVS pipeline. This includes correspondences matching, initialization, bundle adjustment, multi-view stereo matching, and depth fusion. Visualize your camera trajectories, sparse point cloud from SFM, and the final dense point cloud from multi-view stereo. Try your best to get the most creative and visually appealing results. We will choose the top solutions in each direction and announce them in the class.