CSE47201: Introduction to Computer Vision

Assignment #3

Due on Wednesday, December 1st, 2021, before the class

Objective

The goal of this assignment is to understand 2-view geometry. You will also learn how to compute a disparity map and 3D reconstruction (point cloud). **If you have any issues or don't understand anything, contact the instructor for advice.**

Instructions (READ CAREFULLY)

- Complete individual steps and turn in your outputs (see **Task #**) to Blackboard. There are a total of **5 Tasks** in this assignment.
- A single PDF file with all solutions and a discussion of your solution.
- Along with the PDF file you should provide a link or a zip for your results and code (e.g., GitHub).
- No extension. Please start your assignment as soon as possible. Do NOT wait until the deadline.
- Reference: Multiple view geometry in computer vision by Richard Hartley
- Pick any two consecutive images for your assignment

1. Disparity map

Our goal is to implement a disparity map after image rectification. We will break down the steps we studied into several tasks below. Note that camera parameters and essential/fundamental matrix are unknown/missing in many cases so need to be estimated by calibration and 8-point algorithm. In this assignment, camera parameters are given for easy computation and simplicity.

Task 1. **Camera center in the world coordinate system:** The first step is to compute the respective camera centers from the left and right images. You are given both extrinsic and intrinsic camera parameters (see lec 11). Specifically, you have the following extrinsic parameter matrix that combines projection and scene pose transformations:

$$[\mathbf{I} \quad \mathbf{0}] \begin{bmatrix} R^T & -R^T C \\ \mathbf{0}^T & 1 \end{bmatrix} = [R^T \quad -R^T C]$$

The extrinsic matrix is a 3-by-4 matrix, and you will see this matrix in the "cam-poses" folder. You can easily decompose this form into rotation and translation. Provide a solution to compute the camera center using the extrinsic matrix. In your document, you should report how to solve this along with your implementation.

Task 2. Image rectification: Once you know the camera centers, your next step is to compute the image rectification for the horizontal alignment for an image pair. The intrinsic parameter matrix is common for all the images. You can find it in "Calibration.txt". Implement the image rectification and show your results **before and after** image rectification (see Fig.1 or the lecture slides 35-36 in lec13 for example illustration – draw horizontal lines). You will need "projective" transformation to warp images as you did for homography.

Hint: Make sure the two images have the same size for the visualization (you can use crop if needed).

Fig 1. Horizontal lines after image rectification for an example image pair.

Task 3. Disparity map: After image rectification, you will need to compute a disparity map for your image pair. As studied, use window-based matching with NCC or SSD. Define the window size and matching metrics by yourself. Depending on the window size, you will see different results like the lecture slide 13 in lec 12. Illustrate your disparity map. You will observe disparity computation often fails in some of regions. Explain where these regions are and why this happens.

2. 3D point reconstruction

Our goal is to recover 3D point coordinates from an image pair. Once you have a disparity map, you can compute 3D coordinates using disparity information.

Task 4. **Triangulation for 3D coordinate reconstruction**: Compute 3D coordinates from your disparity map. You can assume that focal length f is the average of f_x and f_y in the intrinsic parameter matrix, i.e., $f = \frac{(f_x+f_y)}{2}$. You may want to discard some pixels if their disparity is incorrect to generate a less noisy point cloud.

Task 5. **Point cloud visualization**: Your last step is to visualize the reconstructed points. For each pixel, take its RGB value and reconstructed coordinate. You then supply this information to create a point cloud. You will need to implement a simple PLY format writer. Here is an example specification:

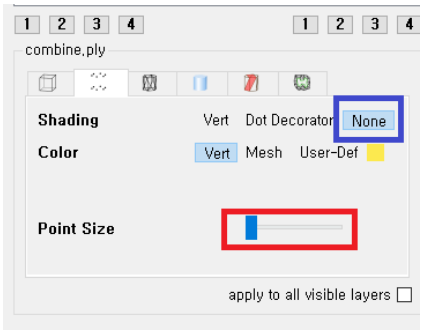
```
ply
format ascii 1.0
element vertex 131283
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
end_header
-4.931166 3.563993 -13.043296 34 27 29
-4.911234 3.563701 -13.042568 32 25 27
…
```

You will need a header followed by 3D coordinates (blue, x y z) and RGB channels (0-255, green) – space separated line. Just copy this template and replace # of points (red) you obtained from Task 4 – see bold and underlines in the box, which you will replace in your implementation. You may want to find more information about PLY at Wikipedia: https://en.wikipedia.org/wiki/PLY_(file_format). Once your result is created as a PLY format, visualize your result using MeshLab (https://www.meshlab.net/). If your point cloud is too big, you can either reduce image size by n or computer 3D coordinates for every n^{nd} pixel (n=2,3,4, …). Submit your result (ply).

Tips:

1) By default, MeshLab has 60 degrees of FoV, which means you will see large distortion if your objects are close to your current camera view. Use "shift+wheel (scroll down)" to make "orthogonal projection". If you see "FoV: Ortho" in the MeshLab status, you have correct visualization.

2) By default, the point size is too big and too dark in MeshLab. To reduce point size, you can find the configuration tab in the right-hand side in MeshLab (see the red box). Also, set "shading" is "none" (see the blue box) to have brighter RGB colors.



Extra credits (this is optional)

Disparity map refinement: If your implementation works, you will immediately observe that your point cloud suffers from stairstep artifacts, which is expected. This is mainly because your disparity map is discrete. There are several approaches to address this issue: 1) local extrema estimation of disparity (see [1]), 2) filtering of depth map (median, graph-cut, etc.) or 3) you can investigate other published works. See Fig. 2 for example before and after the refinement.

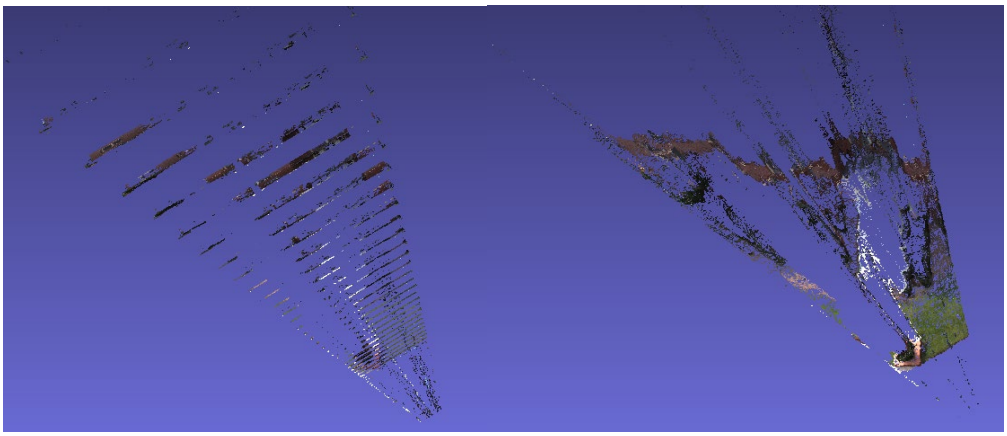


Fig 2. Refined disparity map by applying local extrema estimation [1] followed by median filtering.

Tips: Here is a basic idea of [1]: Given an initial disparity i_0, we denote $i_{-1} = i_0 - 1$ and $i_{-1} = i_0 + 1$. Let c_0, c_{-1}, and c_1 be NCCs (or negative SSDs) for i_0, i_{-1}, and i_1, respectively. Then, the method can estimate more accurate disparity i_{pk} by using a pyramid interpolation.

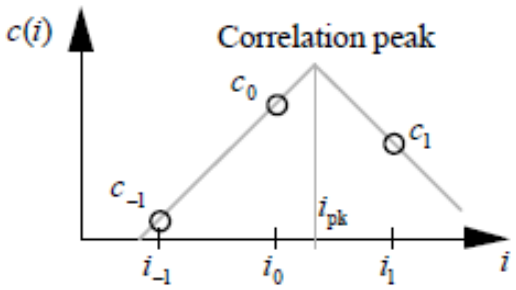


Fig 3. Disparity estimation using a pyramid approximation.

This can be written as follows:

$$i_{pk} = i_0 + \frac{(c_1 - c_{-1})}{2(c_0 - \min(c_1, c_{-1}))}.$$

You can easily refine your disparity using the formula above. If you prove the formula, you will be also given extra credits.

Scene reconstruction: You can reconstruct multiple images in a single 3D scene since more than one pair of images are provided in this assignment. You can compute an individual disparity map per pair for 3D coordinates and combine them into a single 3D space. You will need to consider how to align camera orientations; all you need is to transformation between two camera positions (rotation and translation). If you fully understand the idea of image rectification and world-to-camera coordinate systems, finding the transformation between two images won't be hard. Once disparity maps are improved and all point clouds merged into a single scene, you can expect full reconstruction as shown in Fig. 4.

Tips: You may want to remove some points from your scene if their depth is too large.

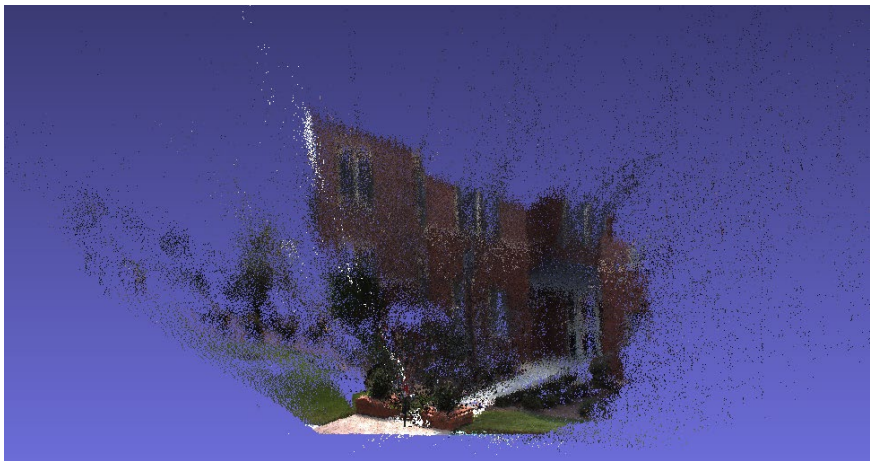


Fig 4. Reconstructed 3D point cloud of 24 image sequences.

References

[1] D.G. Bailey and T.H. Lill, "Image Registration Methods for Resolution Improvement," Proceedings of Image and Vision Computing NZ Conference, 91-96 (1999).