Exercise 4

Deadline: 13. January, 2021

Please upload your solutions to to your personal **Submissions** folder in OLAT.
Use the **Latex template** provided in Exercise_0 for the report.
Hand-written results will not be accepted!

Theory

The main challenge of dense 3D reconstruction is to find correct correspondences among nearly all pixels between the different image views. As explained in the lecture, if the epipolar geometry is known, the correspondence search reduces to a 1-D search problem. In a stereo setup (two images only), image rectification may be used to further simplify the search for correspondences.

(a) What is the advantage of a rectified image pair regarding correspondence search?

(b) What is the advantage of a rectified image pair regarding triangulation?

(c) Is image rectification also a good approach in case of multi-view dense reconstruction? Why? (*Hint: Consider a scenario where several images were taken around a statue.*)

Practical Part

Goal The goal of this exercise is to perform dense matching on rectified images and reconstruct the original 3D scene.

Requirements For the remaining tasks, the following packages are required:

- numpy, python-opencv, scikit-learn
- MeshLab
  - Ubuntu: `sudo apt-get install meshlab`
  - Windows: download binaries from www.meshlab.net

Data Sets In this exercise, we have prepared two datasets. Each dataset consists of a pair of rectified images with known intrinsic calibration and baseline distance. The first pair comes from a synthetic data set rendered with *Blender*[1] and the second pair is taken from the *KITTI*[2] data set. The depth is obtained by

$$Z = \frac{b \cdot f}{d},$$

where b is the baseline, f the focal length and d the disparity. For more details, check the explanation in the provided source code and the README files in the data folder.

[1] `https://www.blender.org`
[2] `http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=stereo`

Implementation Complete the following implementation tasks for both the *KITTI* and *medieval port* data sets. There is a third data set of an *office scene* given as an optional task. As mentioned in the template code, save results of each data set in **separate folders**.

1. Pixel-Based Dense Matching

(a) For each pixel in the left image, find the corresponding pixel on the right image using a **pixel-based** cost function, i.e. the difference between individual left and right pixels.

(b) Compute the disparity map and save it as a gray scale image named `pixelDisparity.jpg`.

(c) Is the disparity map a good representation of the original scene? Why? (The answer should be included in your report.)

2. Window-Based Dense Matching

(a) For each pixel in the left image, find the corresponding pixel on the right image using a **window-based** *NCC* (Normalized Cross Correlation) or *SSD* (Sum of Square Differences) cost functions. Therefore, use a window size of 7.

(b) Compute the disparity maps and save them as a gray scale images named `nccDisparity7x7.jpg` and `ssdDisparity7x7.jpg`.

(c) Experiment with different window sizes: 3×3, 5×5, 7×7, 9×9, 11×11 pixels, etc. and save them to `nccDisparity3x3.jpg`, `nccDisparity5x5.jpg` and so on. See if they can give better results compared to window size of 7.

(d) Comparing the window-based results to that obtained with the pixel-based cost function, has the quality of the disparity map improved? Why? (The answer should be included in your report.)

(e) Compute the 3D coordinates of all correspondences by triangulation, in order to reconstruct the original 3D scene.

(f) Save at least one screenshot of MeshLab with the reconstructed point cloud and add it to your report.

3. Outlier Filtering

In image regions with good texture, for each pixel we will find very few (ideally one) corresponding pixels with high similarity score. The rest of the candidates will have low similarity scores. Unfortunately, in textureless areas matching becomes harder as many candidate pixels will similar scores. If a pixel has a large number of candidate pixels with high similarity score, taking the one with maximum similarity will most like lead to a wrong match. Since having wrong matches leads to wrong 3D reconstruction we filter out such points (those with large number of candidates with high score).

(a) **SSD Filtering** For a given pixel in the left image, if N or more pixels in the right image have SSD scores less than $1.5 \times min\_ssd\_score$, consider it as an outlier and reject the pixel. This will serve as a simple outlier filtering scheme. Try different outlier filtering thresholds $N = 2, 3, 4$ and report the best number (you have to make a qualitative evaluation by looking at the resulting 3D reconstruction). Include your results (screenshots from Meshlab) for at least two different thresholds.

(b) **NCC Filtering** Similarly, for a given pixel in the left image, if N or more pixels in the right image have NCC scores greater than $0.8\times$ *max_ncc_score*, consider it as an outlier and reject the pixel. Again, try different outlier filtering thresholds $N = 2, 3, 4$ and report the best number. Include your results(screenshots from Meshlab) for at least two different thresholds.

(c) Now, based on your results from both *medieval port* and *KITTI* data sets: What is the influence of the window size on the reconstruction quality? What is the influence of the size of window on the processing time?

Remark:
Make sure your code executes the tasks above sequentially by simply calling `python main.py` (include `data.mat` alongside `main.py` in your `.zip` file).

Good Luck!