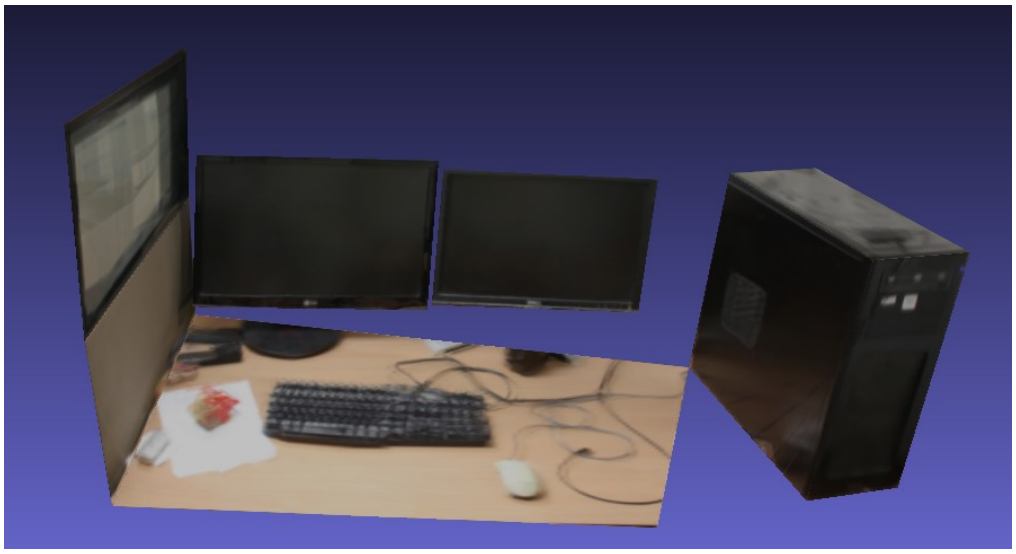
Project 3 : Structure from Motion



This project uses matrix factorization to reconstruct a set of 3D points from correspondences in a set of images. The final result is a texture-mapped set of quadrilateral planar regions.

Project Overview

In this project you will take a set of images of an object, identify correspondence points that are visible in all images, and use these correspondences to reconstruct the 3D structure of the object. You will also identify quadrilateral planar regions in the correspondences for texture mapping.

Skeleton

Your grade will be based on code that you add to complete the skeleton file linked below. Though it should not be necessary, you are free to add additional functions as you see fit to complete the assignment. You are not, however, allowed to import additional libraries or submit separate code files. Everything you will need has been included in the skeleton. The included `sfm.py` file can be used to run the skeleton on a set of images. Usage is described below.

All image parameters for skeleton functions are passed as 3D numpy arrays with shape (height, width, 3) for color images. **You should return images in the same format.**

You can run the the sfm wrapper to test your implementation like this:

```
python sfm.py test_data/tracked_points.txt test_data/test_quads.txt
```

The result will be a 3D model, saved in `mesh.obj`, a directory called `reprojections` containing the 3D points projected back into each image (can be useful for debugging), and a `textures` directory containing textures for each of the quadrilateral planar regions identified in the mesh. The mesh.obj file together with the textures directory can be viewed in any program that can open OBJ-format 3D models. Meshlab is one such program.

Tracking Points

The SfM wrapper requires a set of correspondences and a set of correspondence indices which form planar 3D regions. For the test data, both of these are provided for you. When you use your own images, you will need to track many points through each image in the set. **The points you choose to track must be visible in**

all of the images. The provided `track_points.py` file is invaluable for this task, as you would otherwise need to load all of the images in an editor and find pixel coordinates for each correspondence in every image manually. The point tracker requires the OpenCV Python bindings which are installed on Tux. You can run the point tracker like this:

```
python track_points.py N_POINTS_TO_TRACK IMAGE_1 IMAGE_2 ... IMAGE_N
```

You will be shown the sequence of images N_POINTS_TO_TRACK times. Each time you go through the images, you will be tracking a single point through all images. Clicking on a point in the image will advance to the next. When you have tracked the point in all frames, it will appear in the image with an index # next to it. This signals that you should start tracking another point. Make sure you can actually identify N_POINTS_TO_TRACK correspondence points in each image. Output is saved to `tracked_points.txt` in the same directory as the point tracker.

Once you have identified the correspondences, you will need to provide a list of correspondence indices for quadrilateral planar regions. The format for this list is 4 correspondence numbers per line, separated by spaces. Each line should contain 4 points on a planar region. To help you find these planar points, you can look at the newly-saved correspondence images in the same folder as the input images. Identify planar regions composed of 4 points and write the 4 point indices into a file. Use the numbers shown in the images when you make the file. The test data contains an example in `test_data/test_quads.txt`

Q Derivation Tips

You will need to build a number of matrices to complete the skeleton. Three of them (W, R, S) are defined in the slides and you may simply follow the derivations on the slides to build these matrices. One of the matrices, Q, is not given in the slides. You will need to derive the solution for Q and implement it in the skeleton.

Here are some tips for where to start with the derivation.

- The constraint equations are quadratic in Q, so you can't directly solve for Q.
- QQ^T can be replaced with another variable to linearize the constraints.
- You can solve for the new variable directly and convert it to Q.
- The Cholesky decomposition of a matrix A is: $A = LL^T$.

For a detailed explanation, see this PDF.

For extra credit, you may instead solve the non-linear constraint equations using the non-linear least-squares solver algorithm found in `scipy.optimize`. This will require you to formulate the constraints as a non-linear optimization problem, compute the Jacobian matrix of the problem, provide the proper input to the chosen `scipy.optimize.leastsq` algorithm, then return the solution. **In order to get full extra credit, you must build the Jacobian matrix (even though the algorithm will approximate it for you if it is missing).**

Downloads

- Main SfM Wrapper: sfm.py
- Point Tracking Tool: track_points.py
- Skeleton: p3_skeleton.py
- Test Data: test_data.zip

What to Submit (Drexel Learn)

Zip or tar.gz containing the following:

- **Code**: Include your completed p3_skeleton.py file.
- **Data**: **You must demonstrate your code on at least one set of images you have captured on your own.** Include the images and the resulting textures/mesh in your submission.
- **Report (PDF)**: Writeup about your experience (what was difficult, etc.). Describe your experiments if you did any (e.g., changing parameters and showing their effects on the results), and tell us what you did for extra credit. Include lots of pictures.

Grading

- **Code** [55pts]
  - Build W,R,S Matrices [20pts]
    - [10 pts] Build W, the matrix of centered image points.
    - [10 pts] Compute R and S from W.
  - Solve for Q [20pts]
    - [10 pts] Build linear constraint system.
    - [10 pts] Solve system and recover Q.
  - Structure from Motion [10pts]
    - Use R, S, and Q to compute the scene structure and camera motion.
  - Texture Mapping [5pts]
    - Use the correspondence points and planar point IDs to compute a homography and warp the images into planar surface textures.
- **Data** [5pts]
  - Demonstrate your code on **at least one image set of your choosing.** Submit all files used to generate images in your report (images, correspondences, and region point indices).
- **Report** [10pts]
  - Submit a detailed report that explains how you implemented the project and your overall experience. Show what went well and wrong, discuss what the causes may be, experiment with different ideas and demonstrate the pros and cons.
- **Extra Credit** [30+pts]
  - [15 pts]: Solve for Q using non-linear least-squares. You must derive the Jacobian matrix for full extra credit, see note above.
  - [15 pts]: Automatically track correspondences using a feature detector + descriptor.
  - [? pts]: Do something really interesting.

  You must show the results of your code for each extra credit portion.