Greg Matthews
CS 583 – Section 900
May 25th, 2017

Computer Vision
Project 3: Structure from Motion

Building Matrices W, R, and S:

Objective:

- Build a mean-centered matrix called W which will house the centered image points
- From W, use singular value decomposition to decompose W into U, D, and V matrices, and manipulate them to create R and S

Steps:

1. Given original image correspondents points, obtain the mean-centered image points using the factorization approach
2. Form the matrix into shape $2 * Frames\ x\ Correspondences$, and decompose using SVD to get R and S, being the camera rotation, and 3D scene points, respectively

Challenges:

This problem wasn't particularly challenging, mostly dealing with basic algebra and matrix operations to compute the necessary W, R and S matrices. The one issue that was tough in the beginning, is pinpointing exactly what is known, and what is not known. Solving for what is not known is especially challenging when you're unfamiliar with what can be computed, such as the solving for the center of mass of each 2D in each frame.

Solving for Q:

Objective:

- Solve for matrix Q that will satisfy the unit-vector and orthogonality constraints that must be assumed for image points ih and jh.

Steps:

1. Given the image points ih and jh which are extracted from matrix R, we must build matrixes A and b given the predefined constraints.
2. Use matrix A and b to solve for matrix c using least squares algorithm
3. Using c, form matrix C and then perform Cholesky to obtain Q

Challenges:

Again a very straightforward function, not much trouble was had here, especially because of the kindly distributed PDF file that goes into detail on how to solve for Q.

Structure from Motion:

Objective:

- Perform SfM factorization on a given set of corresponding image points.

Steps:

1. Use the previously discussed computations to solve for matrices W, R, and S, as well as matrix Q
2. Given Q, refine the matrices R and S by apply Q and Q transpose to them, respectively.
3. Build an orthonormal matrix that rotates the first rotation matrix from R into the identity matrix
4. Apply the computed rotation matrix to all matrices in R and S

Challenges:

Building matrices W, R, and S were intuitive processes and resulted in little to no issues with computation. Applying Q to both R and S also posed no significant challenge, however building the orthonormal matrix to rotate the first rotation matrix into the identity matrix, was most assuredly an unintuitive endeavor. The computation was confusing as a result of each rotation matrix in R being of the shape 2x3, and as a result of it not being symmetric, one cannot invert the matrix to solve the orthonormal matrix with a basic linear equation. As a result, the 2x3 matrix had to be considered as 2 orthogonal vectors, with the 3rd one needed to be computed by taking the cross product of the first two. Given this, a 3x3 matrix can be used to linearly solve for the orthonormal matrix by using a 3x3 identity matrix and the first rotation matrix.

Texture Generation:

Objective:

- Given a set number of frames and region points that correspond to 4 points of a quadrilateral planar region, extract the texture from the region averaged over all images.

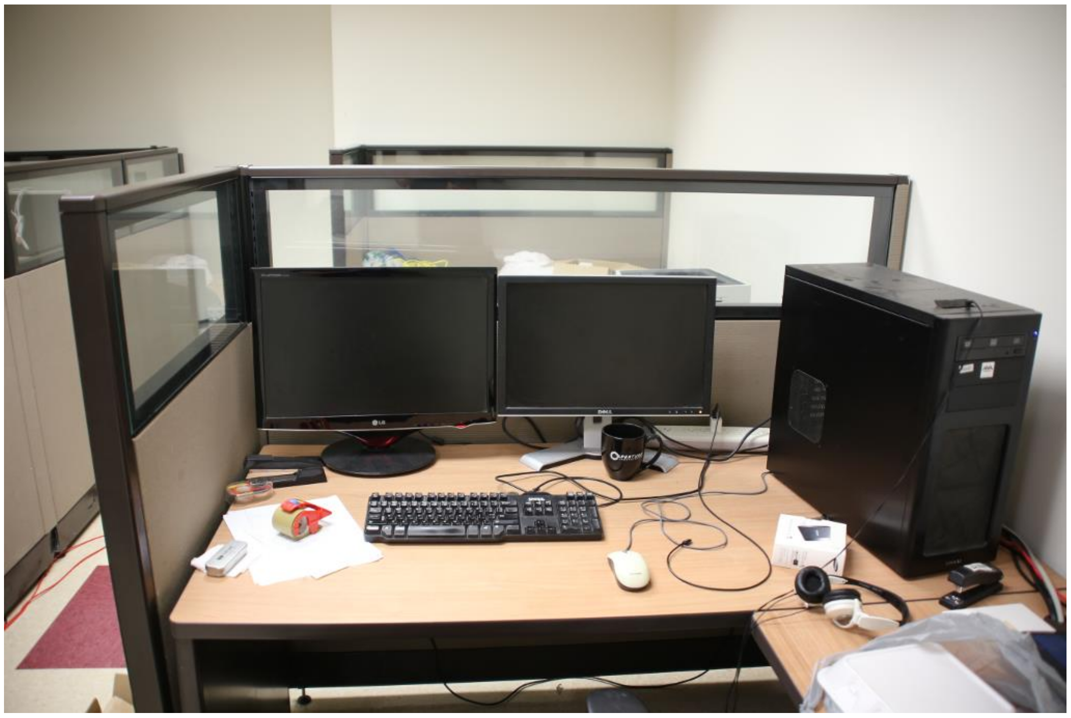
Steps:

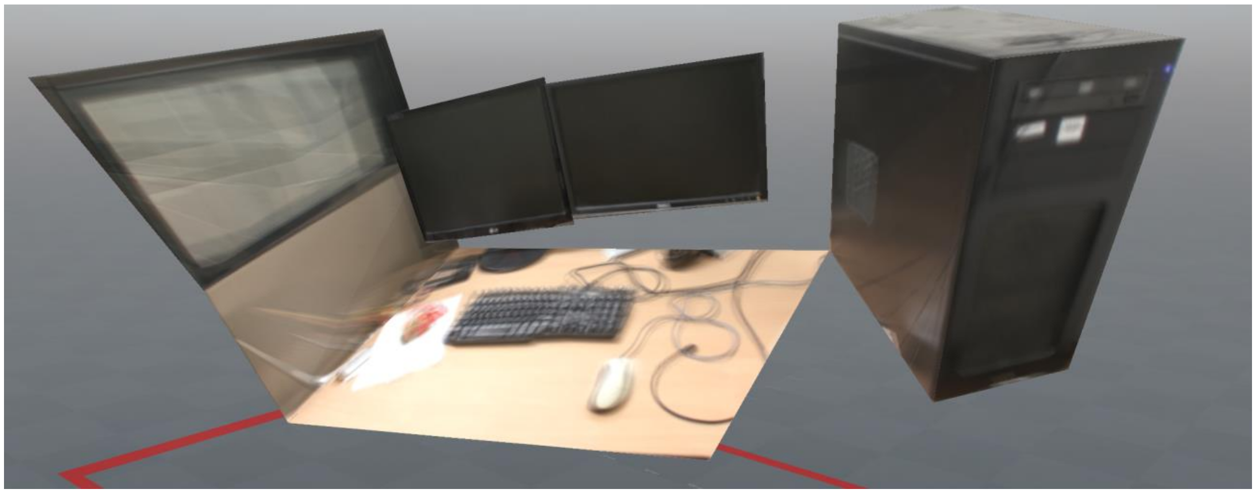
1. Build the shape of the planar texture region.
2. Find the Homography that warps the region points of the current frame, into the texture planar region.
3. Warp the image with the precomputed Homography matrix, and obtain the given texture from the current frame
4. Average all the textures from each frame, and continue the process for each texture region

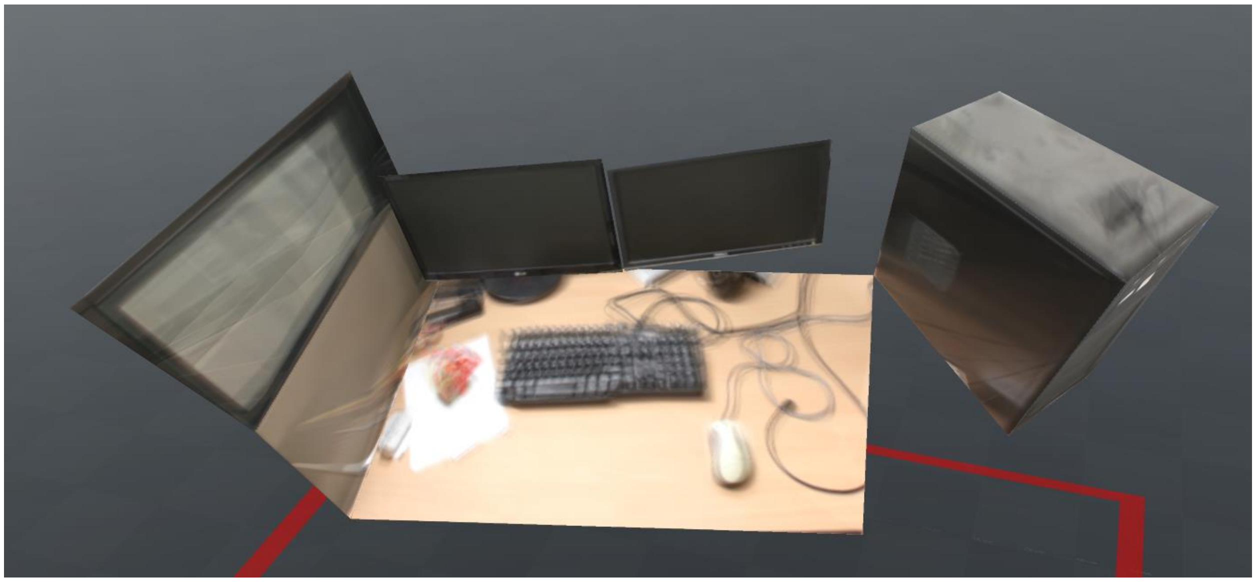
Challenges:

This function took some time to understand what exactly is being performed, and looking up information pertaining to the $findHomography$ and $warpPerspective$ functions was necessary in understanding their capabilities and limitations. After some research, the function was quite simple, however I continued to get significant issues with my generated textures, and nothing in my implementation seemed incorrect. It turned out that my region points were producing negative coordinate values because I was passing the points by reference in my sfm function and altering the contents of them when computing the mean centered points. A simple copy of the image points corrected the issue with the textures.

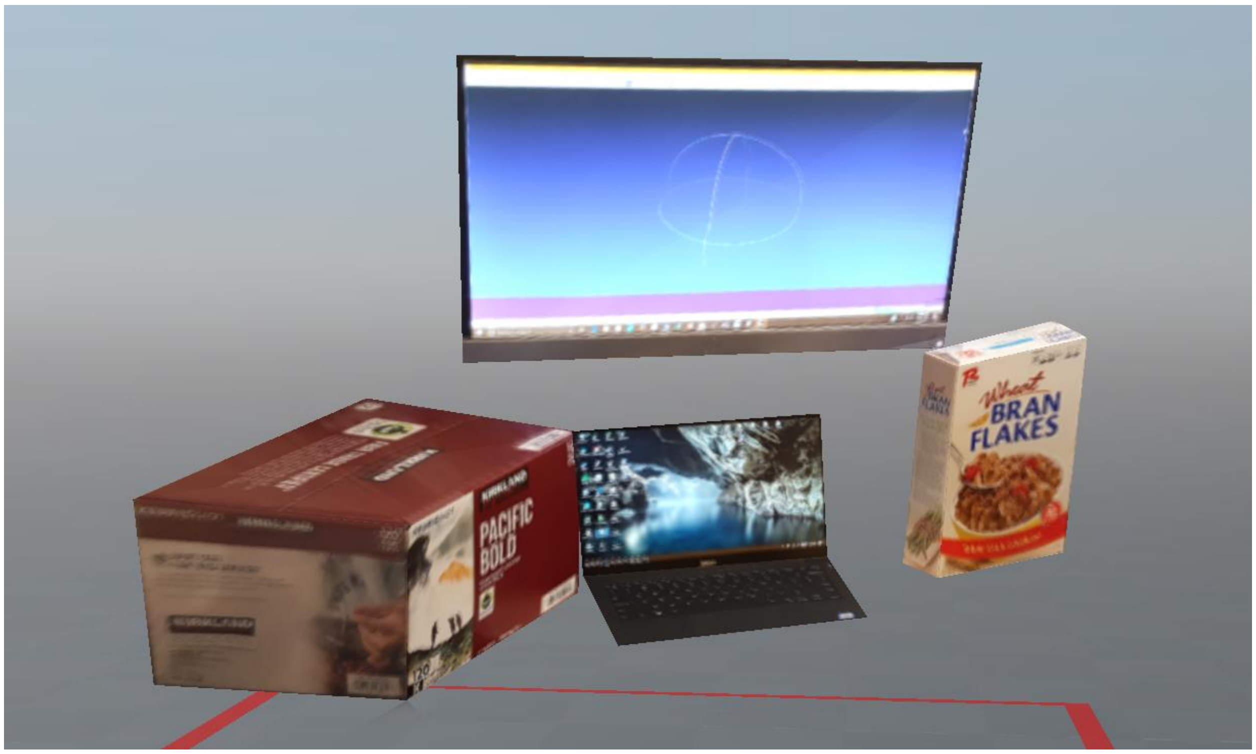
Structure from Motion Experiment #1: [Test Data]



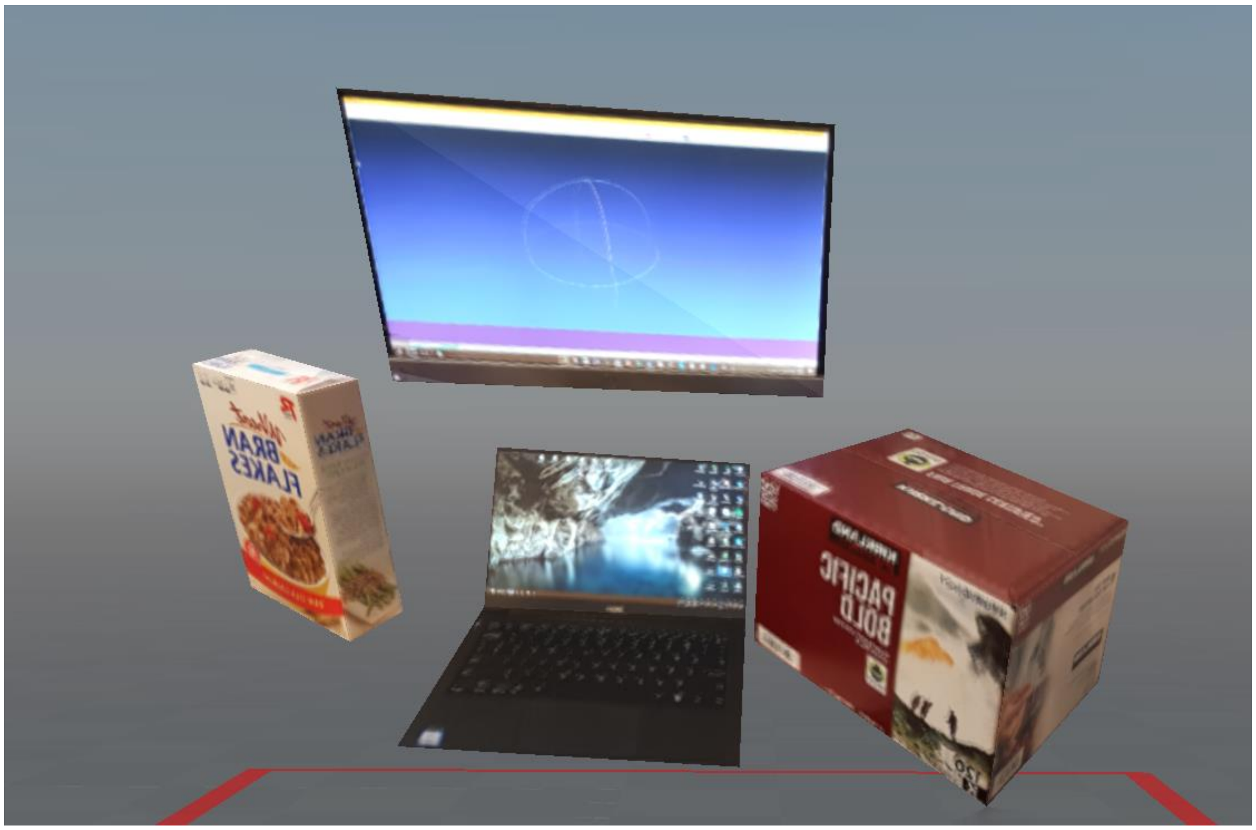




Structure from Motion Experiment #2: [My Data]



Structure from Motion Experiment #2: [My Data - Inverted]



Structure from Motion Experiment #3: [My Data]