Creating a Structure from Motion Pipeline

Allen Wu
Stanford University
allenzwu@stanford.edu

Abstract

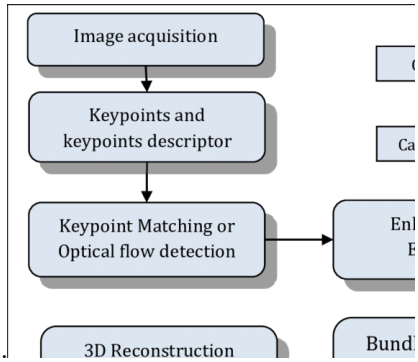
This report details the process of the author's approach of implementing a structure from motion pipeline in python. Structure from motion is a well-established technique in computer vision that pertains to recreating three-dimensional structures from two-dimensional image sequences. The approach in this report is rather limited in scope, focusing primarily on a single cube modeled in the 3d modelling software Blender. Regardless, the final result seems to be usable, though it should be noted that the approach is heavily tailored to the dataset.

1. Background/related work

Structure from Motion is a well-established technique in computer vision that attempts to model the shape of a 3d object from a series of 2d pictures, generally from a video. Fundamentally, Structure from Motion is based off of the concept of parallax, where objects closer to the camera or viewpoint move more than objects farther away. Through this, depth information can be extracted from a series of images, and through such, depth information can be used to generate an accurate 3d model of the object in question. Structure from Motion generally appears in many commercial software such as blender and photoshop, as well as appearing in various libraries across different languages such OpenCV in python, as well as in MATLAB and Mathematica. There are also various technical and research applications in oceanography and archeology, though these approaches typically require specialized equipment and only work in extremely specialized settings.

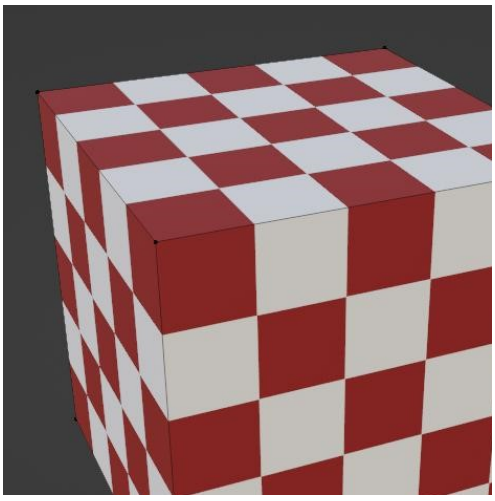
2. Approach

This report's structure from motion pipeline is a highly simplified version of the process listed in the graph below. The report does not cover adequate image acquisition, as the camera is simulated, and texture mapping is also skipped, in favor of simply using a 3d point cloud as the final model. Regardless, the remaining sections are attempted from python code and generally follows a similar pipeline as the OpenCV library in python.
Pipeline:



2.1 Camera calibration and image acquisition.

The images were acquired from the 3d model of a box created in python. The box was textured in a checkerboarded pattern, for the main purpose of making it easier to discover point correspondences. The checkerboard was a lso added for the purpose of camera calibration, though it was found that using the given camera parameters in blender and was more accurate. Example of data used:

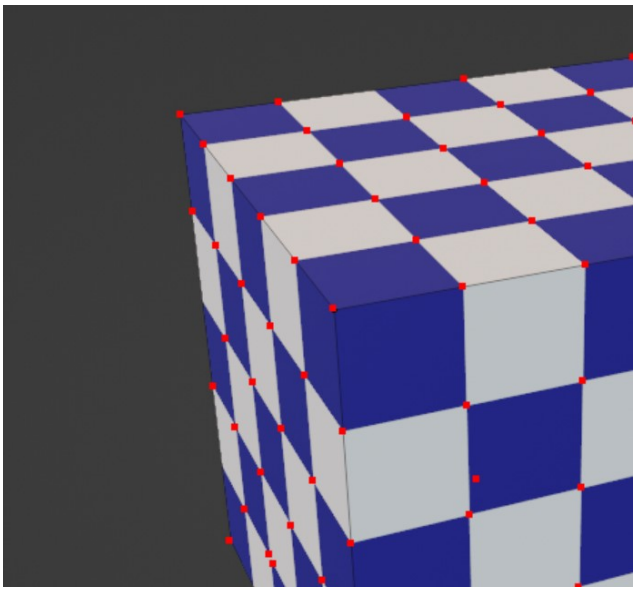


2.1. Keypoint Matching

The keypoint matching was done using a 2-step process Harris corner detector. The first step was the implementation of the Harris corner detector. In the Harris corner detector, we maximize a function

$$E(u,v) = \sum \quad w(x,y) \quad [I(x+u,y$$

for corner detection. Various attributes were adjusted in accordance with the given data set so that all corners were accounted for on the checkerboarded patterns on the square. Then a parallel line detection algorithm was implemented to ensure that the point correspondences could be set up properly. Each set of points was used to calculate slopes and all irrelevant slopes were discarded until only the slopes of the lines matching the checkerboarded patters on the given data was left. Then, the points at the edges of these parallel lines, representative of the corners of the checkerboard in the picture were isolated. Since the 3 faces of the cube remained consistent across all of the given pictures, the point correspondences were appropriately set up between the different pictures. Corners detected:



2.2. Algebraic Structure from Motion

The next step was the main bulk of the structure from motion. The images were normalized using the given camera matrix. Then, the essential matrix was calculated from the normalized image coordinates, fundamental matrices, and intrinsic matrices. Then, we use established factorization methods to determine valid rotation matrices and translation vectors. Using the correct R, t pair, we are able to find the triangulated point

3. Experiment

As mentioned earlier, the experiments were only performed on a single dataset of 3d boxes generated in blender. The parallel line detection method mentioned earlier would only work on such boxes, as anything otherwise would be fairly pointless. The following is the result of only using a small amount of corner correspondences, so it was extremely clean, and the representation of the cube is easily visible. First result:

l

The problem is that this initial model was only formed from 2 images, so it seems that the angles between the 3 planes were slightly off, and the angles are not exactly perpendicular.

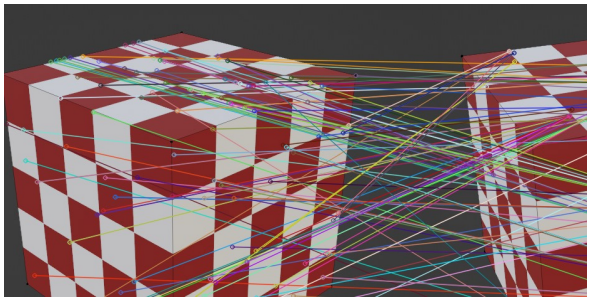
Here is the final result from the culimation of 5 pictures and entire algorithm:

The results were actually fairly good all things considered. Some of the point correspondences had to be removed manually as otherwise it wouldn't match properly and the entire result would be ruined. The main problem is the appearance of some noise as well as the fact that one of the faces seem to be curving inward for some reason. This is probalby due to not taking bundle adjustment into account, or improperly handled camera matricies, as it is not certain whether the camera matricies obtained from blender's data were accurate or not. It may also be the issue of rounding, as the corner detection was forced into integers to be more clear.

3.1 Ground Truth

To obtain a ground truth, the pictures were also fed into blender in order to establish a good baseline, or a good target to reach for. Perhaps it was the fact that the images were also generated in blender, but the baseline in blender was excellent, and far exceeds any of the results this project was able to accomplish. The following details the results of the SfM from blender. A large number of points were removed for reasons of visibility. Blender result:

The result from blender was basically perfect, with the cube being represented extremely accurately. This is most likely due to the usage of some advanced keypoint matching algorithms, though it is odd as when those same algorithms were used in python through openCV, the correspondences were extremely messy. Here is the result of running the ORB algorithm from opencv:



Here, the point correspondences were just a huge mess, and it was pretty much impossible to detect anything. Perhaps the mistake was that the results here were the result of 2 images with a rather large change in angle, while the results from blender were results of a more gradual change in angle.

4. Conclusion

There are several important lessons that can be learned from this project. First is that while the concept behind Structure from Motion is relatively understandable, the implementation is actually quite nuanced, and requires a decent understanding of the dataset. This is likely the main reason that there is no widespread implementation of Structure from Motion available, and most 3D reconstruction applications are relegated to industrial settings where there can be far better camera angles, controlled settings, and image quality and consistency. Another lesson is that the processing of data often takes most of the time in projects like this, as the algorithm itself isn't new and doesn't need much development. However, getting the data to fit that algorithm may be challenging, and there are many ways for the algorithm may fail that may not be obvious from a cursory glance or from even observing the algorithm on a different dataset. If this project was to continue in the future, the most obvious development would be to expand the project to a different dataset, and preferably one from specific images. Dealing with background noise shouldn't be that big of an issue, and a checkerboarded box in a physical setting can probably be modeled similarly. In this case, the camera calibration would need to be properly worked out, as the patch job solution in this project may no longer be sufficient. Another obvious development is a better keypoint matching algorithm. This section of the pipeline became the bottleneck of the entire project, as it is difficult to weed out wrong correspondences, which are extremely important because a single wrong point correspondence may throw off the entire model. Perhaps a better representation of these point correspondences could be made to have better error tolerances.

5. References

[1] Tony Jebara, Ali Azarbayejani and Alex Pentland, 3D structure from 2d Motion, MIT Media Laboratory, Cambridge MA. http://www.cs.columbia.edu/~jebara/papers/sfm.pdf

[2] Wu, Changchang. "VisualSFM: A visual structure from motion system." (2011). http://ccwu.me/vsfm/doc.html

[3] Bianco, S., Ciocca, G., & Marelli, D. (2018). Evaluating the performance of structure from motion pipelines. *Journal of Imaging*, *4*(8), 98. https://www.mdpi.com/2313-433X/4/8/98

[4] Exploring Structure from Motion Using OpenCV https://hub.packtpub.com/exploring-structure-motion-using-opencv/

[5] Structure from motion overview Mathworks https://www.mathworks.com/help/vision/ug/structure-from-motion.html

[6] Structure from motion OpenCV https://docs.opencv.org/3.4/de/d7c/tutorial_table_of_content_sfm.html

[7] CS231A Course Notes 4: Stereo Systems and Structure from Motion http://web.stanford.edu/class/cs231a/course_notes/04-stereo-systems.pdf

[8] Blender Structure from Motion https://blenderartists.org/t/blendersfm-structure-from-motion/675474

[9] Structure from Motion in Python https://harish-vnkt.github.io/blog/sfm/

[10] SFM: Structure from Motion https://openmvg.readthedocs.io/en/latest/software/SfM/SfM

[11] Structure from Motion and 3D reconstruction on the easy in OpenCV 2.3 https://www.morethantechnical.com/2012/02/07/structure-from-motion-and-3d-reconstruction-on-the-easy-in-opencv-2-3-w-code/