15-418 Project Milestone Report

Matthew Yu, Emily Ho

November 2025

1 Work Summary

These are the work completed so far for the project:

1. Implemented a serial version of the pipeline on Google Colab.
2. Migrated part of the code from Colab to Repository.
3. Configure environment on the GHC machines to allow for boilerplate/preprocessing code to be written in Python, and parallel modules to be written in CUDA/C++.
4. Experimented with many ways of computing the SVD (averaging multiple 3*3 SVDs to compute 4*4 SVD, cuSOLVER, Jacobi method).
5. Parallel triangulation kernel completed (see below section for more details).
6. Certain components of the RANSAC loop (finding epipolar distance, eight point algorithm).
7. Tests implemented for multiple parts of the pipeline to ensure correctness & check performance.

2 Progress on Goals and Deliverables

A Parallel triangulation kernel has been implemented and performs better than a CPU implementation, showing roughly 20x speedup on both random inputs with a similar size to our test images as well as the test images themselves (correspondent points was computed using the serial pipeline).

While our RANSAC implementation for finding image correspondence is currently nonfunctional, once that part of the pipeline works, we will almost have the whole pipeline completed and have an MVP. Scaling the program to take in more datapoints will not be too difficult as we would just have to loop the pipeline to handle multiple images so we are still on track for our target final product.

3 Preliminary Results

Performance of triangulate.cu (GPU implemention) vs triangulate.py (CPU Implementation)

# Correspondent Points	CPU Time (ms)	GPU Time(ms)	GPU Speedup
1	0.082	0.180	0.46
10	0.448	0.192	2.33
100	4.155	0.207	20.03
1000	37.997	0.342	110.98
10000	379.908	1.720	220.91
100000	3965.583	15.707	252.48
1000000	38552.759	168.694	228.54

The GPU implementation of triangulate sees considerable speedup over the CPU version, especially as the number of correspondent points increases. We computed correspondent points from a sample image using the CPU pipeline, passing it into the GPU implementation of triangulate, and the accuracy was consistent with the CPU pipeline and was also around 20x faster than the CPU version as it had roughly 350 correspondent points.

4 Concerning Issues

A big commitment during the project involved implementing an efficient SVD solver on GPU as it is used in both finding correspondences and triangulation 3D points. Nvidia has a library that has an efficient SVD solver called cuSOLVER. When we eventually got the solver working, we noticed that it was actually performing slower than the CPU implementation using numpy's SVD solver, and was only 8x faster on 1 million points, which was way more than the number of points that will be computed in this project. We then switched to a one-sided Jacobi method that saw significant improvement over the CPU version even on smaller inputs, though the time spent exploring cuSOLVER has significantly delayed the development schedule.

Both the SIFT and Bundle Adjustment algorithms are serial as a lot of time was spent implementing an efficient GPU SVD solver. This can be a bottleneck in the program if they are not optimized and may prevent a real-time 3D construction program due to the slowdown.

5 Updated Schedule & Deliverables

5.1 Schedule

We are slightly altering our schedule since we don't have a working parallel implementation yet (which was the goal for week 2). We are extending this by half a week, and our goal is to get a working MVP of the pipeline and then start

optimizing.

We will try to scale our pipeline up to use multiple pairs of images from a video instead of just using 2 images only, as mentioned in one of our "Hope to Achieves".

We are also going to leave one part of the pipeline serial for now (the SIFT computation) since although it's very parallelizable if we do it by point, the algorithm is extremely complicated and would require significant effort to implement (Sample SIFT Implementation: https://github.com/rmislam/PythonSIFT/blob/master/pysift.py).

1. 12/1 (Emily): Finish RANSAC parallel implementation.
2. 12/1-12/3 (Matt): Parse video into multiple frames / images so we can process many pairs of images instead of just 1. Utilize bundle adjustment to minimize reprojection error.
3. 12/2-12/4 (Emily & Matt): Assemble the whole reconstruction pipeline & ensure correctness.
4. 12/5-12/8 (Matt): Explore implementing live video processing to reconstruct points in real-time and investigate parallelization of the SIFT algorithm.
5. 12/5-12/8 (Emily): Optimize the RANSAC algorithm & overall pipeline to increase parallelism and performance.

5.2 Updated Deliverables

1. Finish parallelizing the RANSAC algorithm
2. Implement a parallel 3D reconstruction pipeline (excluding the SIFT in which it would be sequential)
3. Scale up the pipeline to work for multiple images
4. Provide method for visualizing / plotting the results (might be through Colab, or directly in GHC if storage allows)