UNIVERSITY OF ALBERTA

Final Project Presentation: Photorealistic 3D Scenes using SfM with Gaussian Splatting

---

Shijie Bu, John Tyler

https://github.com/john00003/sfm-gaussian-splatting

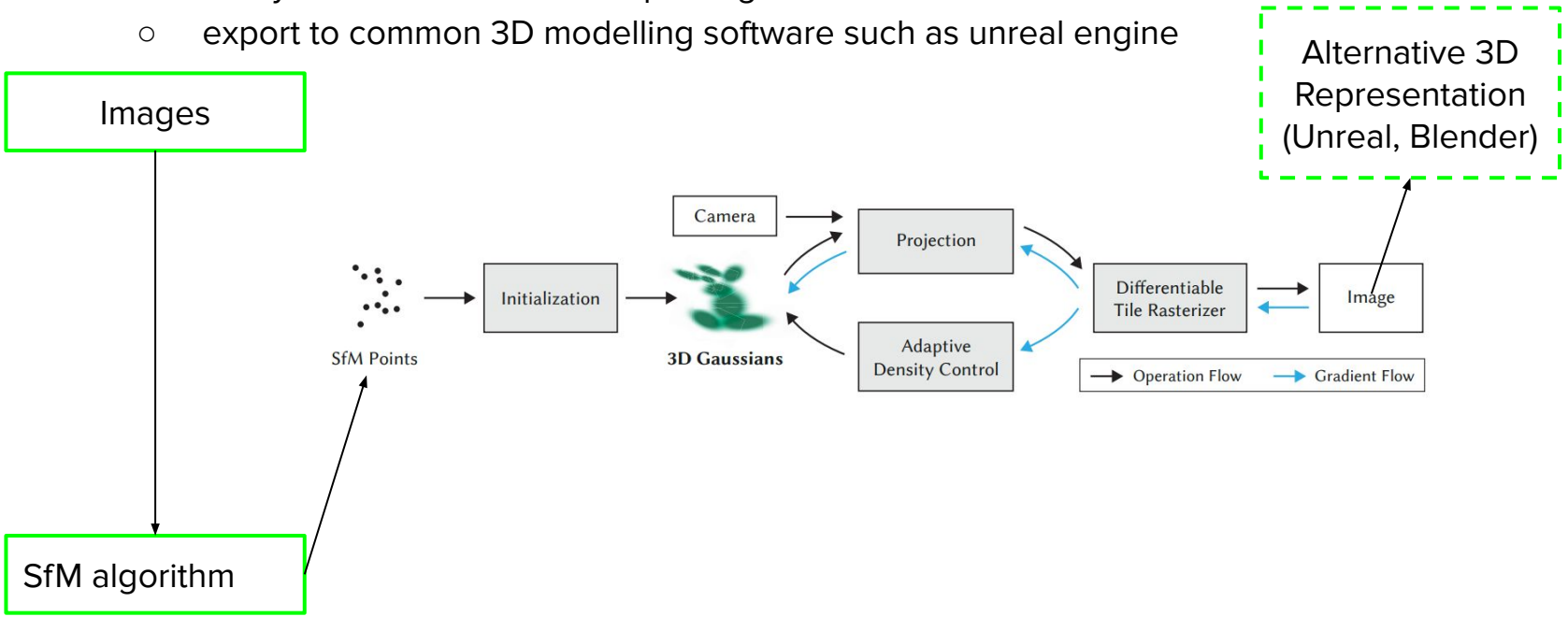
CMPUT 428 / 615 - Winter 2025

Overview - What is Gaussian Splatting?

Gaussian splatting is a method of reconstructing a 3D scene using a **dense** point cloud. This provides an alternative to meshing, texturing, and applying photorealistic rendering techniques to create a 3D scene.

Project Overview & Goals

- Apply fundamental techniques learned throughout the semester to create our own SfM algorithm
  - ready to use with Gaussian Splatting
  - export to common 3D modelling software such as unreal engine

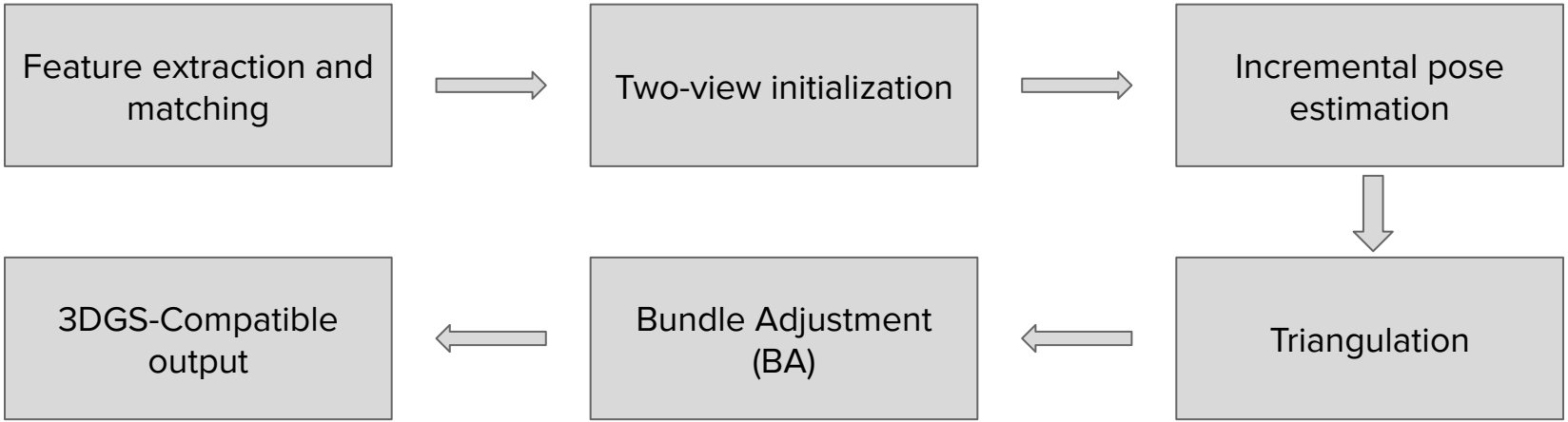


Our Work - SfM Algorithm Overview

Incremental Structure-from-Motion System

Goal: Reconstruct 3D scene structure and camera poses from a set of images.

A 6-step implementation:



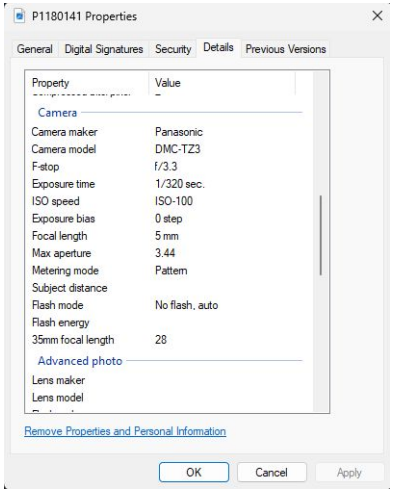
SfM - Feature Extraction

Image Loading & Intrinsics Extraction

- Load images and extract EXIF intrinsics or use hardcoded values
- Intrinsics matrix built per image

SIFT Keypoint Extraction (CPU)

- Detect and compute keypoints per view
- Attach RGB color to each keypoint for visualization



SfM - OpenCV Matching Interface and SIFT

Scale Invariant Feature Transform

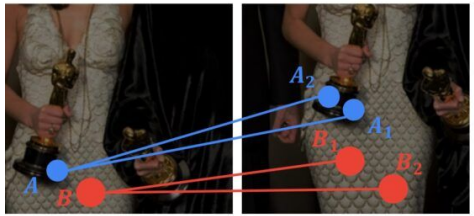
- select points to search for correspondences in other images

```
struct View {
    int id = -1;
    bool registered = false;
    std::string image_path;
    cv::Mat image;
    cv::Mat K;
    std::vector<cv::KeyPoint> keypoints;
    std::vector<cv::Vec3b> colors; // a parallel array with keypoints
    cv::Mat descriptors;
    std::map<int, std::vector<std::vector<cv::DMatch>>> matches_map;
    std::vector<std::pair<int, int>> points_3d;
    Eigen::Matrix4d pose = Eigen::Matrix4d::Identity();
};
```

SfM - GPU Feature Matching

CUDA Brute-Force Matcher

- GPU-accelerated descriptor matching
- KNN with Lowe's Ratio Test
- Full / Sequential / Windowed match strategies



$AA_1 = 20$ - The best match

$AA_2 = 80$ - The second best match

$BB_1 = 10$ - The best match

$BB_2 = 25$ - The second best match

True

$if\ 20 < 0.6 \cdot 80$

$20 < 48$

False

$if\ 10 < 0.6 \cdot 25$

$20 < 15$

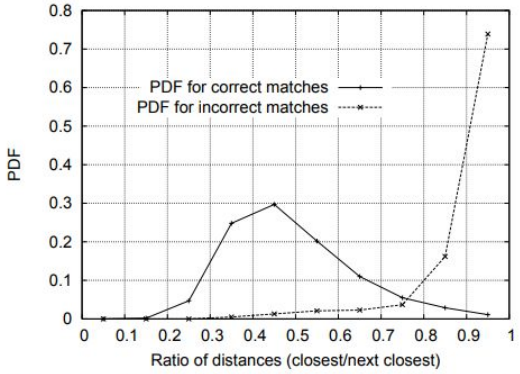


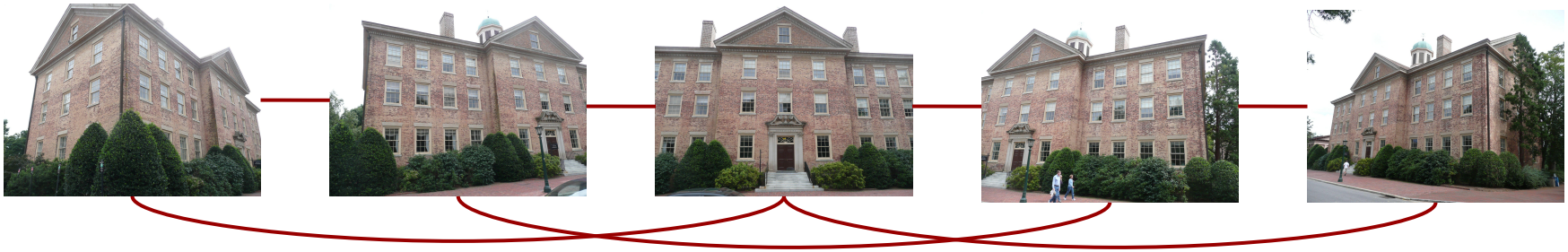
Figure 11: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

SfM - Matching Acceleration and Algorithm

Brute Force (repeat process on each image)

Sequential Search

SfM - Matching Acceleration and Algorithm



Windowed Sequential Search

Sequential Search with Anchor

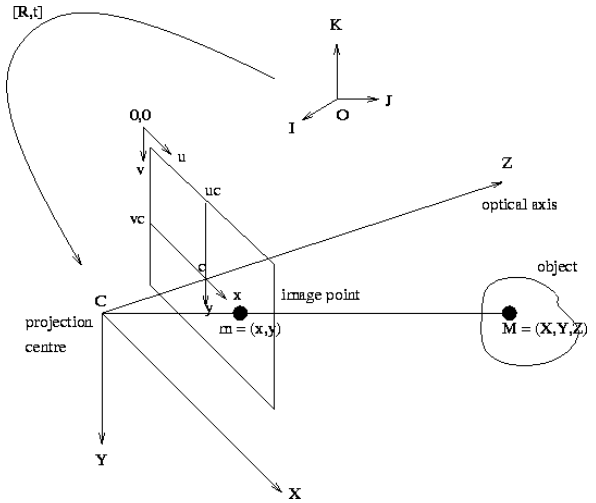
SfM - Two-View Initialization

Select the Best Initial Pair

- Two images with the most strong feature matches
- KNN + Lowe's ratio test to filter high-confidence matches

Estimate Relative Camera Pose

- Find essential matrix using matched keypoints
- Decompose E into rotation (R) and translation (t)
- OpenCV's recoverPose to extract relative camera motion



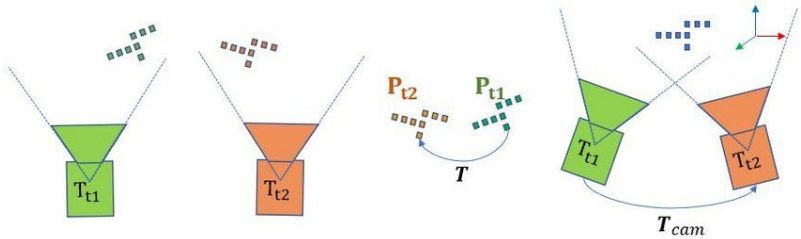
SfM - Two-View Initialization

Two-View Initialization

- Triangulate Initial 3D Points
- Back-project 2D matches into 3D space
- Filter points using reprojection error and depth checks
- Store these as the first 3D landmarks (tracks)

Then we got:

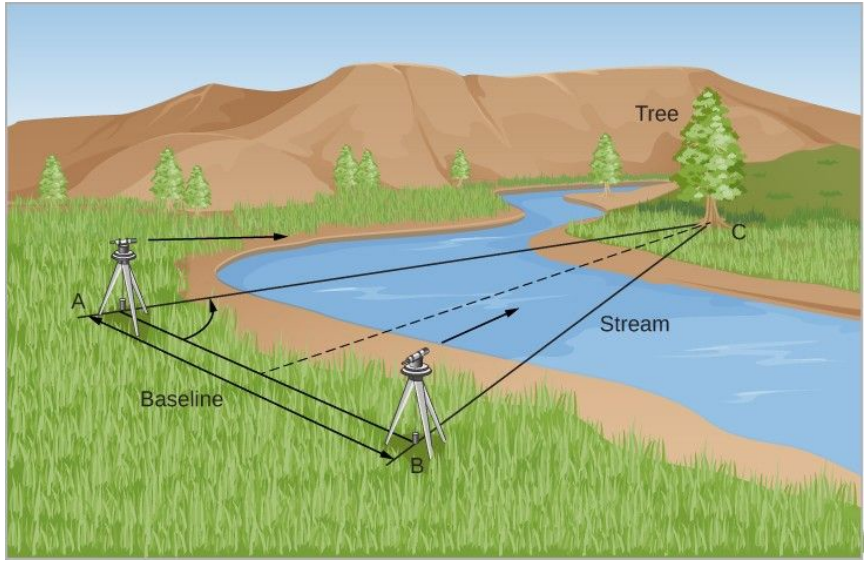
- Two images now have known poses
- A small set of 3D points are initialized



SfM - Two-View Initialization

Triangulation in surveying - same logic!

https://courses.lumenlearning.com/suny-astronomy/chapter/surveying-the-stars/

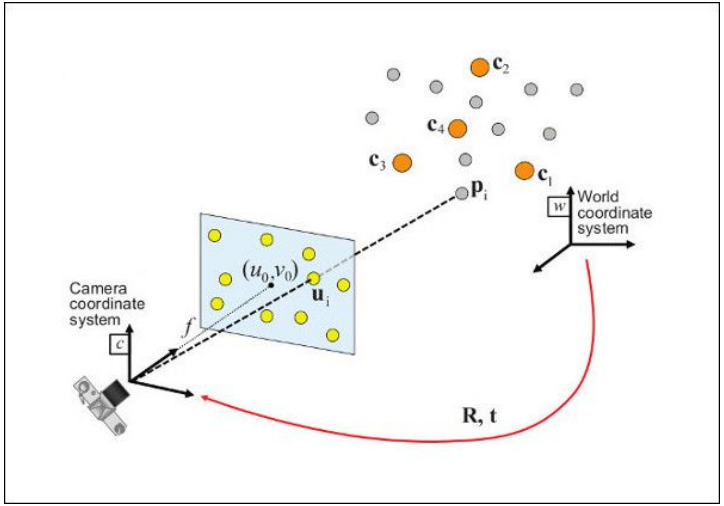


SfM - Registering New Views

We need to estimate the pose of a new image using known 3D points.

Find 2D-3D correspondences

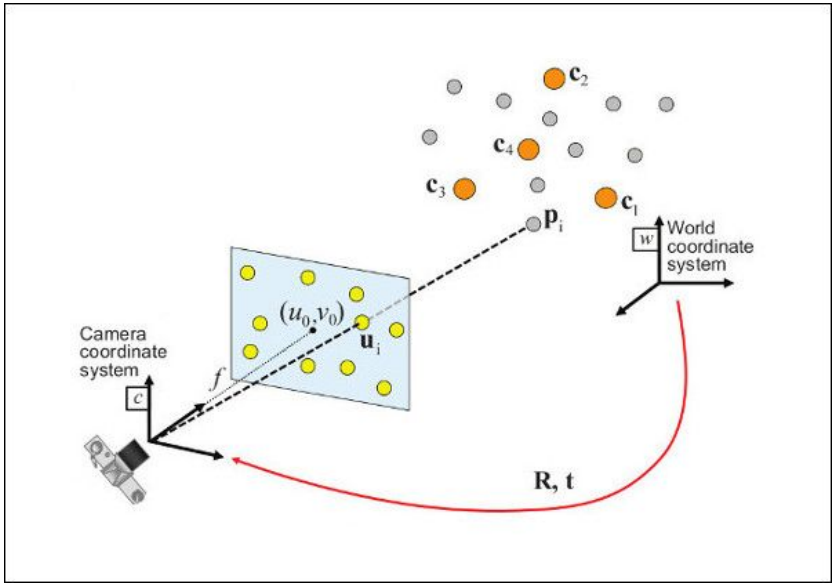
- Match the new image's keypoints to already-triangulated 3D tracks
- For each matched keypoint, find the corresponding 3D point



SfM - Registering New Views

Solve camera pose

- Use OpenCV's solvePnPRansac to estimate camera pose
- Accept the pose only if enough inliers are found

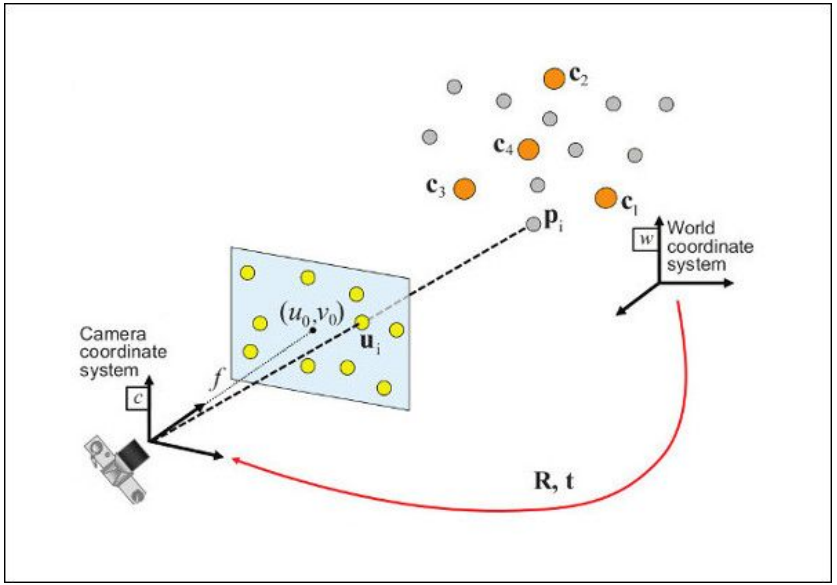


SfM - Registering New Views

Add new view to the system

If successful, mark the view as “registered”

It can now contribute to further triangulations and BA



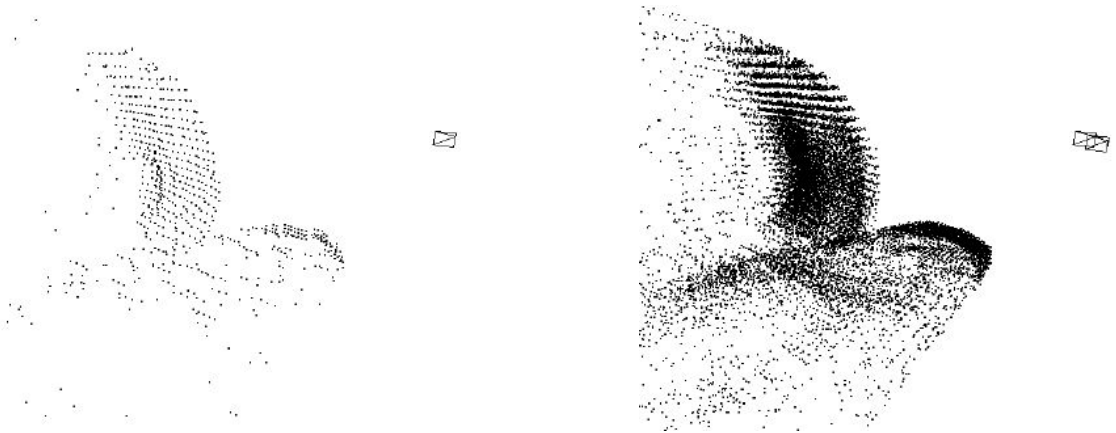
SfM - Triangulating New Points

When a new view is registered

- Match it to all previously registered views

For each match:

- Check if the matched keypoints are not already used in a 3D track
- Use known camera poses to triangulate a new 3D point

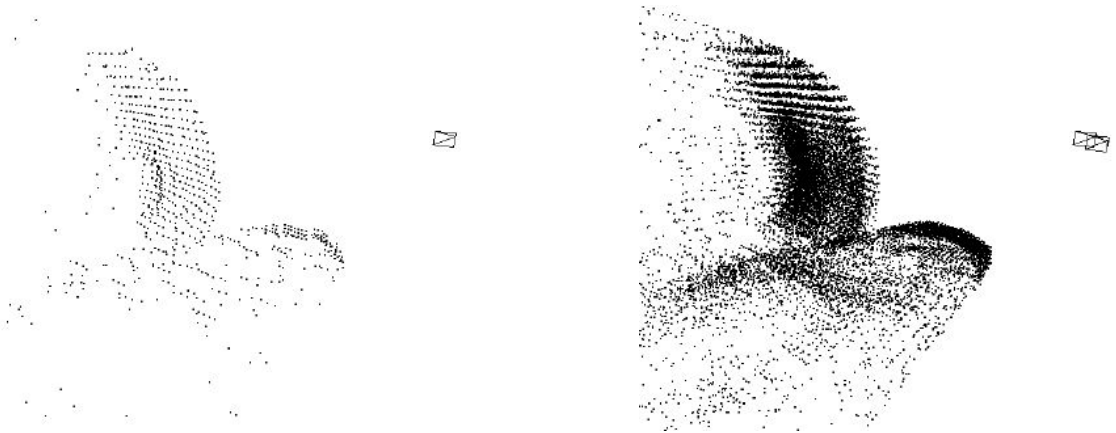


SfM - Triangulating New Points

Validation:

- Reprojection error must be low
- Depth must be positive and not too far
- Add to global track list only if valid

Result:



SfM - Bundle Adjustment (BA)

Purpose:

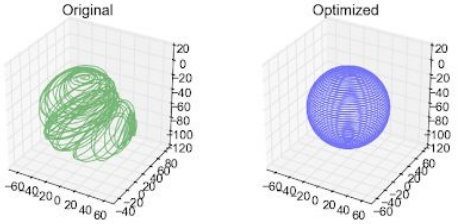
- Refine all camera poses and 3D points to reduce projection error

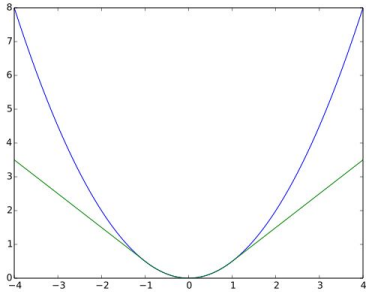
Global BA:

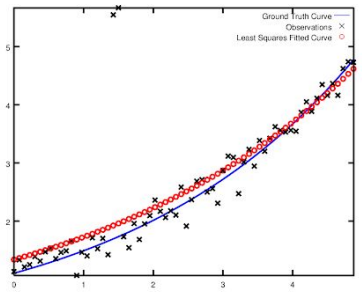
- Run after many views are registered
- Optimizes:
  - All camera poses
  - All 3D point positions

Ceres Solver with EigenSparse backend

- Robust to outliers using Huber loss







SfM - Bundle Adjustment (BA)

Local BA

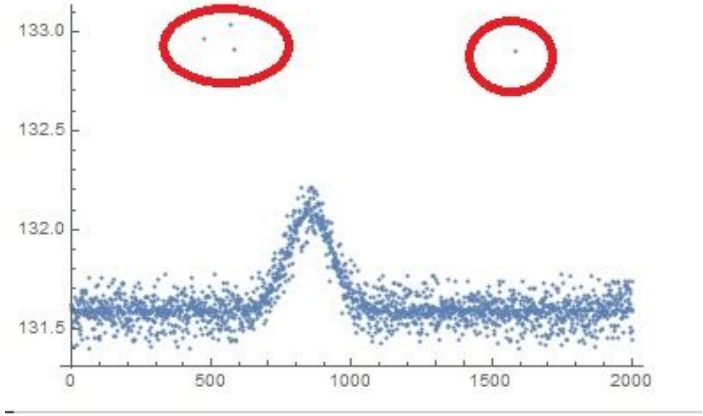
- Run after each new view is added

Optimizes:

- The latest 3 views (sliding window)
- Their shared 3D tracks

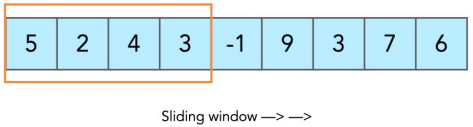
Much faster than global BA

Keeps structure accurate during expansion



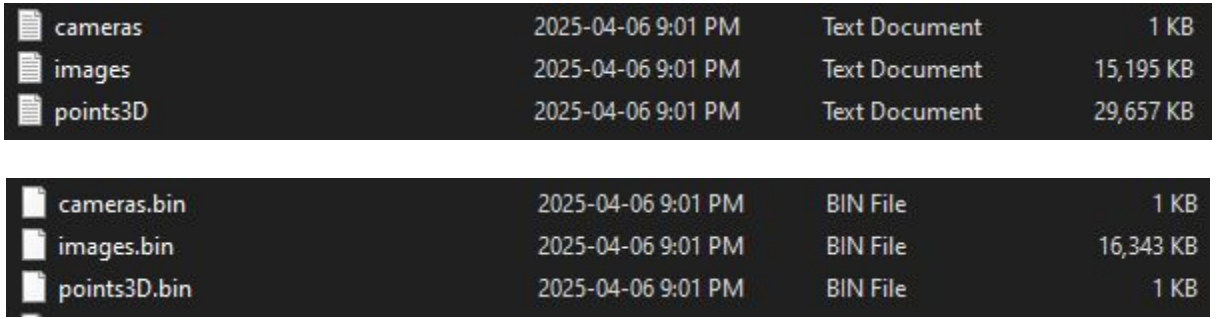
Final Step:

- Remove outlier points with large reprojection error

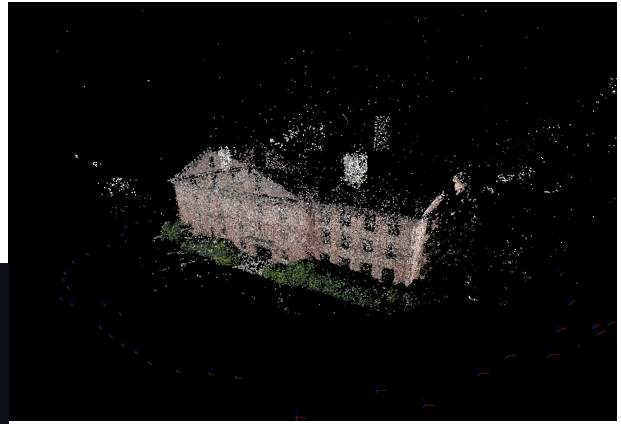


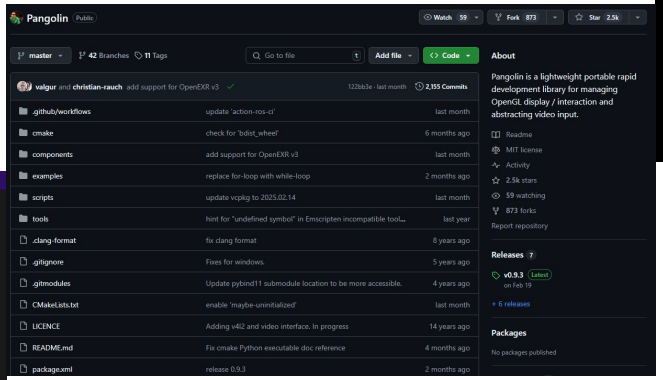
SfM - Output Formatting

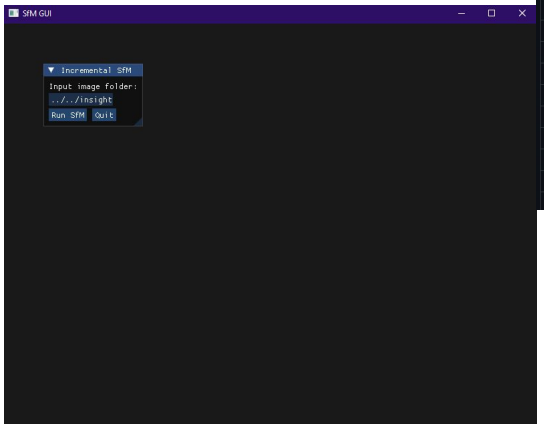
- cameras.txt
  - encode intrinsic matrix of all cameras used to photograph scene
- images.txt
  - encode each key point seen from each view
- points3D.txt
  - encode the 3D coordinates of each point, and the keypoint in any view that observes this point



SfM - 3D Point Cloud Visualizer and GUI







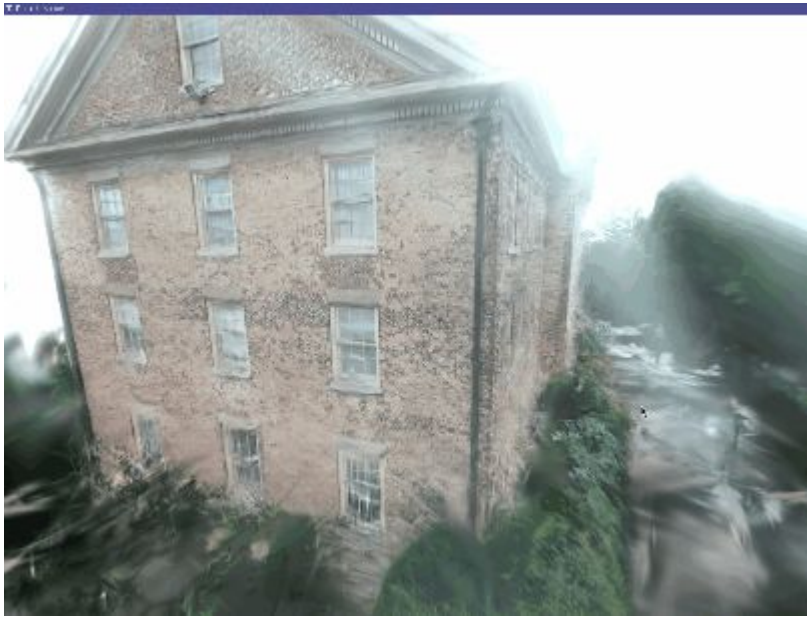
Incremental SfM

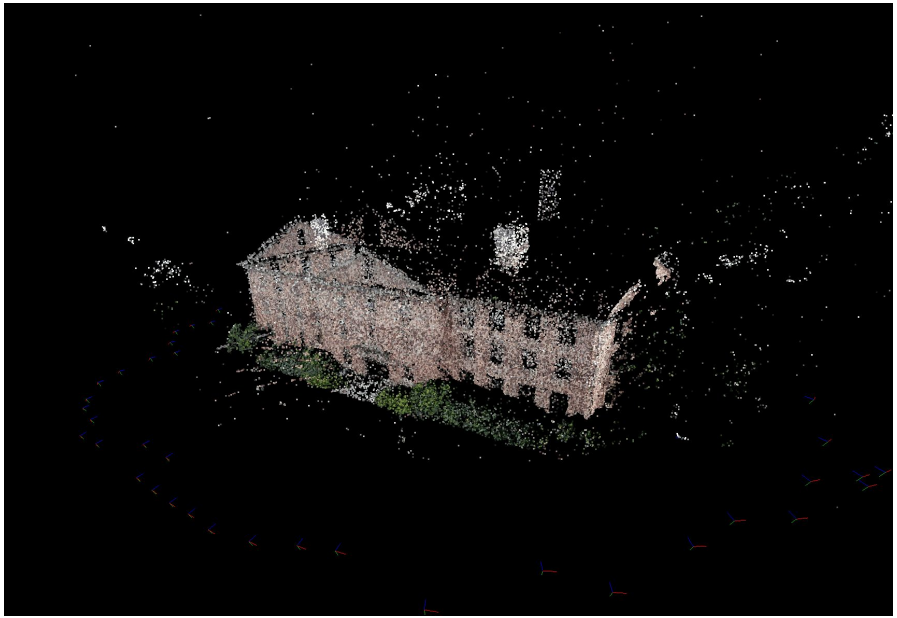
Keep registering new cameras + keep adding new point clouds

Builds the entire scene bit by bit!

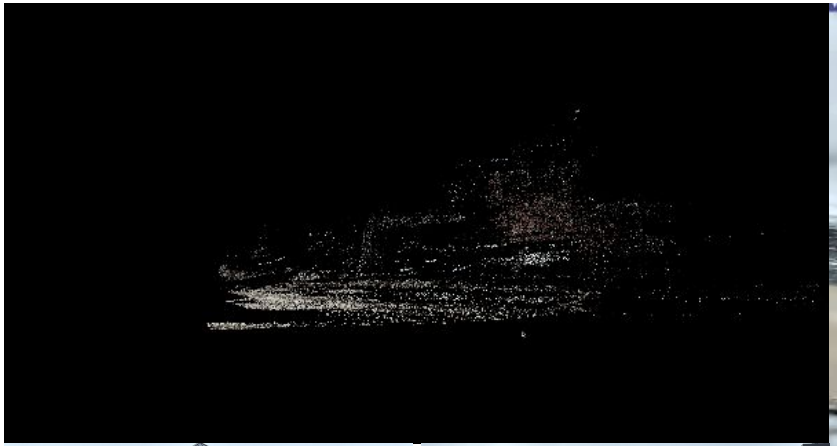
Image source:
https://capsulesbot.com/blog/2019/03/12/apolloscape-sfm.html

Results - South Building at UNC Chapel Hill

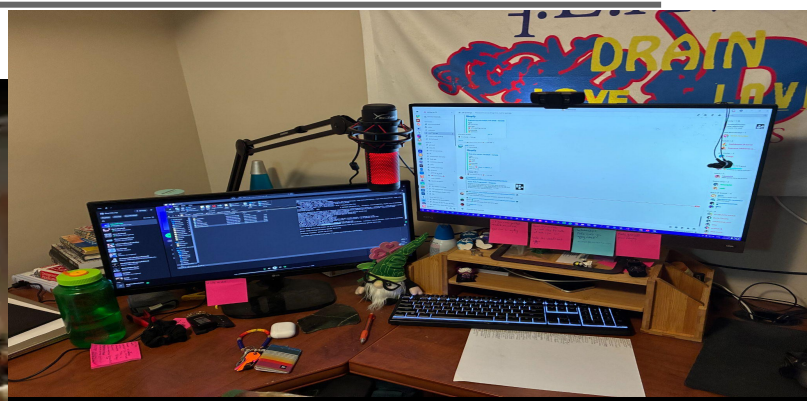




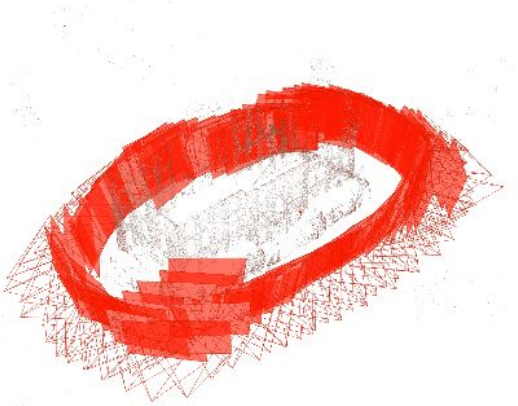
Results - Insight Medical Clinic

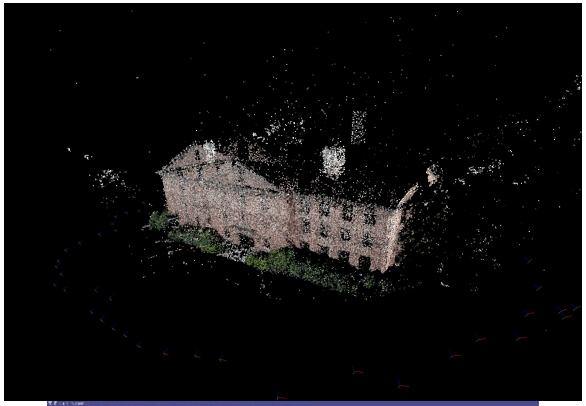


Results - John's Desk

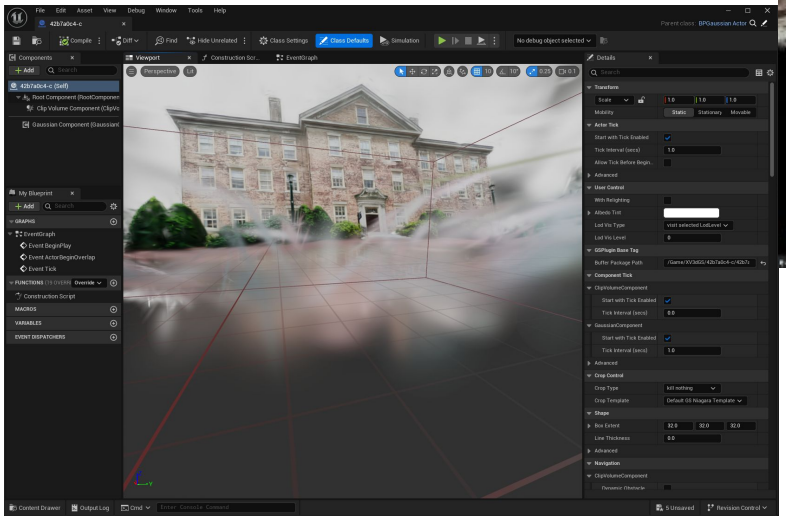


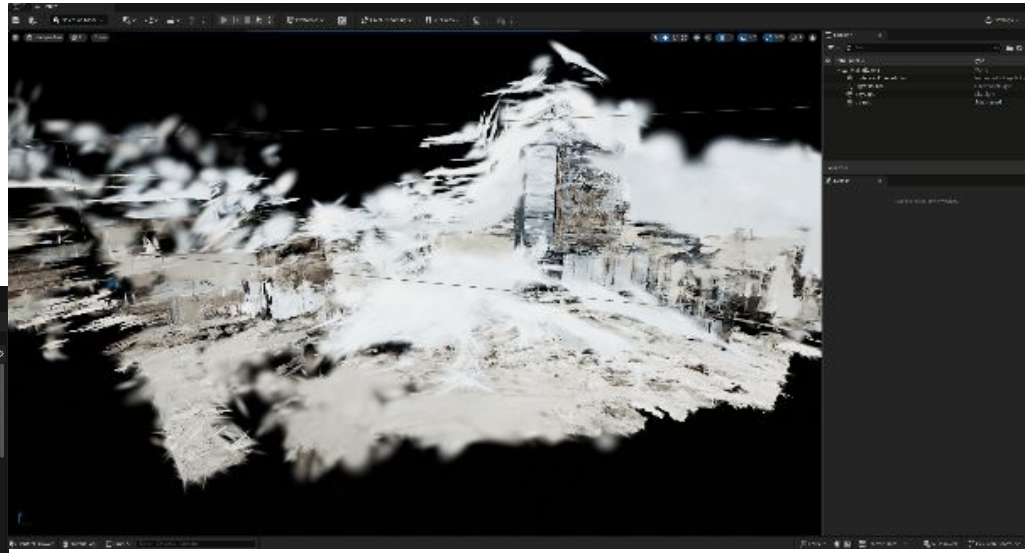
Results - COLMAP vs Ours





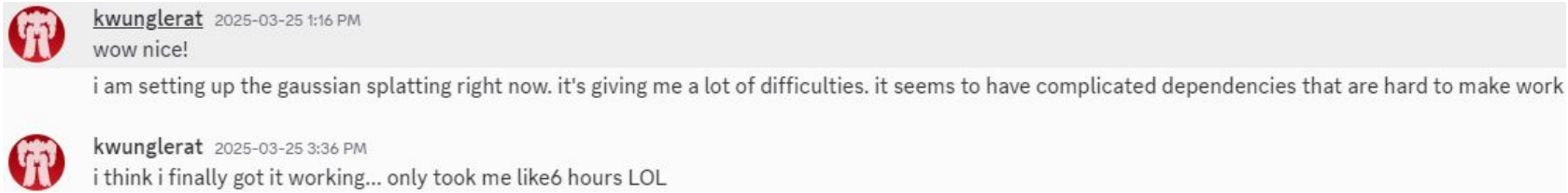
Results - Visualized in Unreal Engine





Challenges - Package Dependencies

- Some of our imported libraries (in order from hardest to easiest to install):
  - OpenCV with Cuda module
  - Google's Ceres Solver
  - Gaussian Splatting repository
  - Cuda
- A docker container, and conda environment (for gaussian splatting) is desirable.



Challenges - Matching Techniques

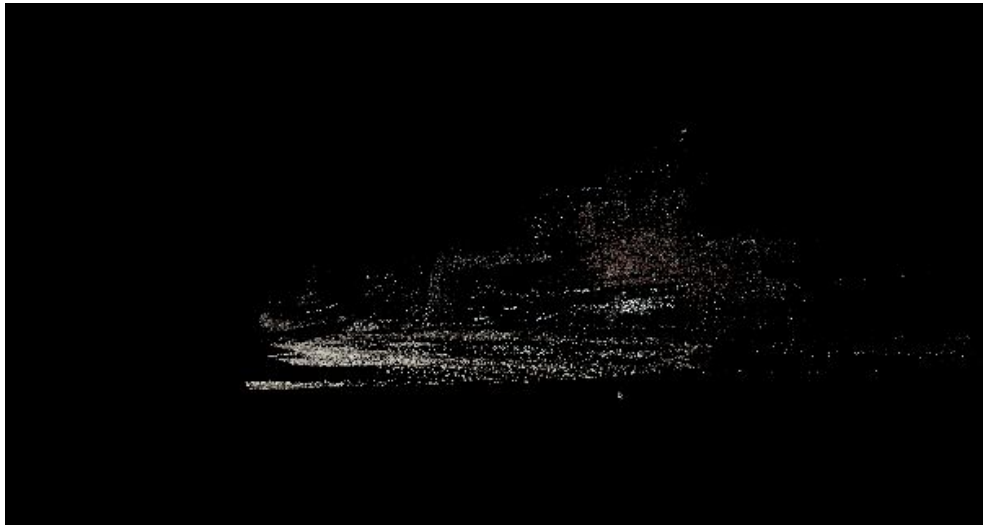
- Which matching technique should we use?
  - tradeoff: accuracy/completeness of feature correspondences vs. computational requirements
  - O(n^2) vs. O(n) many matches. days of computation vs hours.
- Is sequential matching sufficient?
  - did pretty good for one input, but then lost track
  - sequential matching only gives a few views for a given view to register with
    - smart next view/registration selection?
    - “Choosing the next best view is critical, as every decision impacts the remaining reconstruction. A single bad decision may lead to a cascade of camera mis-registrations and faulty triangulations” [1].

Challenges - Loop Detection

- Good 3D structure, but once view wraps around...
  - sequential matcher fails!
    - loop detection?
  - smart selection for next view to register?

Challenges - Outlier Filtering

- Good 3D structure, but what is that point over there?
  - some are detecting irrelevant features of the background
  - others are very inaccurate points from 3D structure in foreground
- Can we filter out these unnecessary points based on their (x,y,z) magnitude?
  - why aren't they being filtered by the reprojection error during triangulation?



Challenges - Input Considerations

- Too many images: SfM takes way too long
- SIFT feature detection likes edges, corners, and textured material
  - buildings do especially well
- Need to produce good SfM to get good Gaussian Splatting
  - not what we initially expected - authors reported good results with random initialization

Extensions and Future Work - Feature Matching

- Feature matching is slow
  - can we use Cuda to program better feature matching algorithms? simple heuristics like start searching other image from position of keypoint in initial
- Why do a brute force search when we could use trackers for feature correspondences?

Extensions and Future Work - 3 View Triangulation

- COLMAP uses a 3 view triangulation algorithm
  - likely more accurate

Extensions and Future Work - Next View Selection

- “Choosing the next best view is critical, as every decision impacts the remaining reconstruction. A single bad decision may lead to a cascade of camera mis-registrations and faulty triangulations” [1].
  - likely a source of error/outliers/loop closure problems
- Is there a better ordering for next view registration?
  - Can only triangulate with already registered views
- Can we create an iterative algorithm that re-registers a view?

Questions