Avkalan Labs Assignment

submitted by- Bishal Saikia (NIT Silchar)

1 . Environment setup and Dependencies Installation

```
conda create -n nerfstudio python=3.10 -y
conda activate nerfstudio
```

For **dependency isolation** , we create an conda environment named as nerfstudio for our project. Following nerfstudio installation guide, we do the following :

a . Install PyTorch 2.1.2 with CUDA 11.8:

```
pip install torch==2.1.2+cu118 torchvision==0.16.2+cu118 \
  --extra-index-url https://download.pytorch.org/whl/cu118
```

Other runtime libraries are already bundles with Pytorch , so we move on to nerfstudio installation.
b. Install nerfstudio and verify

```
pip install nerfstudio
ns-install-cli --help
```

Verify all installation using apt. commands as shown below:

```
python -c "import torch; print(torch.__version__)"
python -c "import torch; print(torch.cuda.is_available())"
python -c "import torch; print(torch.version.cuda)"
```

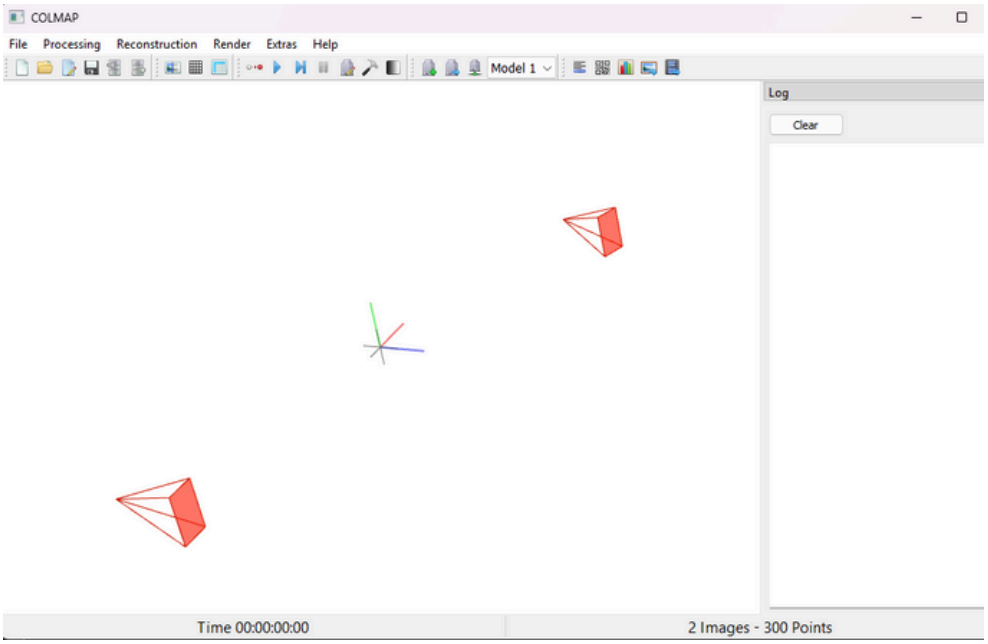
Created folders as shown below for further model training:

```
C:\nerf_project\
├── data\
│   └── video.mp4
├── processed\
└── outputs\
```

**video.mov*

2 . Video processing

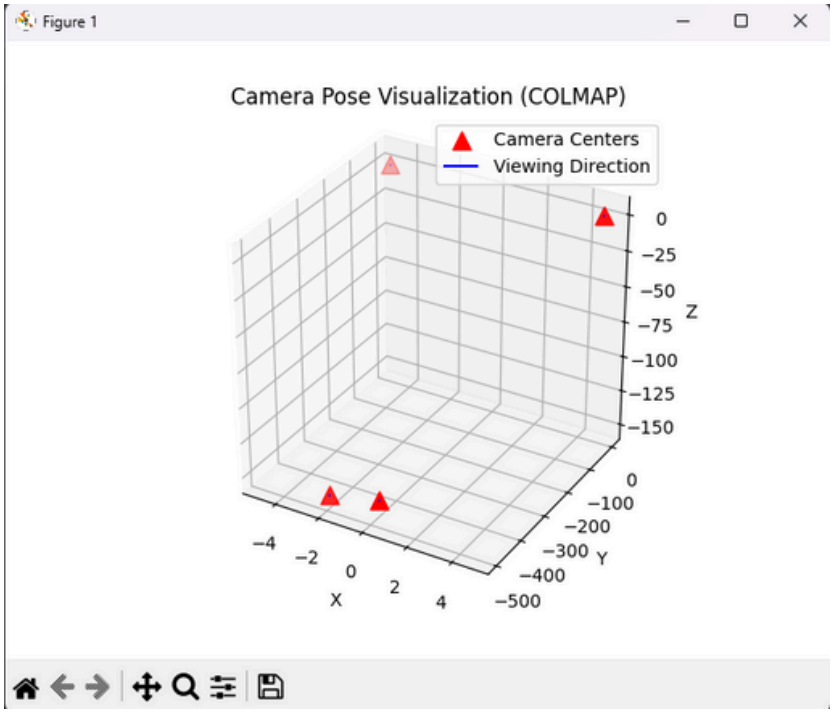
Camera pose estimation was carried out using COLMAP's Structure-from-Motion pipeline. An initial attempt using 70 target frames resulted in poor pose recovery. Increasing the number of target frames improved feature overlap and pose stability; however, the motion characteristics of the input video still constrained successful pose estimation to a small subset of frames (≈1.33%). The recovered camera poses and sparse 3D points were visualized and analyzed as the final result.



Above figure shows the sparse reconstruction and estimated camera poses visualized using the COLMAP GUI. Due to limited parallax in the input video, camera poses were recovered for only two frames. The red pyramids represent the estimated camera extrinsics, while the sparse point cloud represents triangulated feature points.

3 . Pose Estimation

Plotting camera positions and orientations using matplotlib and open3d using the python script cam_pose_display.py . Since ".bin" files cannot be parsed by python, we converted them into interpretable ".txt" file and then used the python script to get following result.

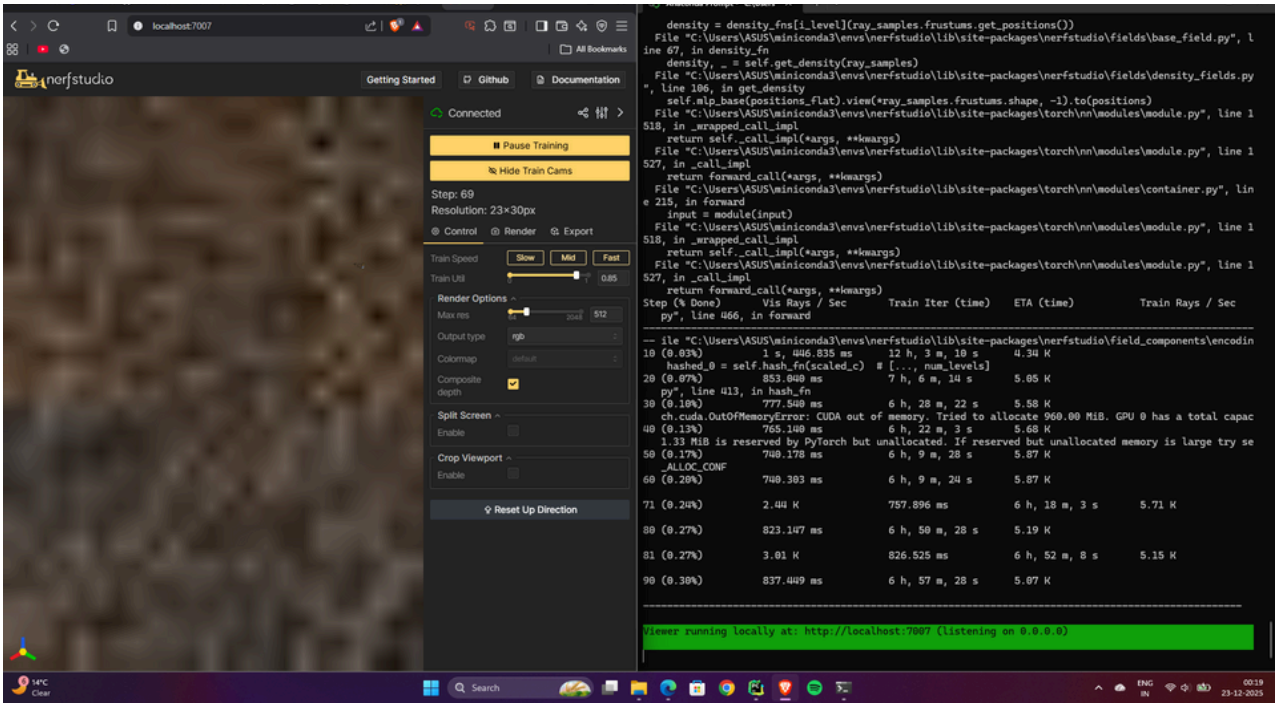


Red triangles: Camera centers
Blue arrows: Viewing direction

4 . NeRF Training and Experiment tracking using **Nerfacto** and **wandb**

NeRF training using the Nerfacto pipeline was attempted using the available camera poses. However, due to sparse pose estimation (≈1–2% of frames), the model could not learn a consistent radiance field. This highlights the dependence of NeRF on accurate and dense camera pose estimation.

```
ns-train nerfacto ^
  --viewer.quit-on-train-completion True ^
  --pipeline.model.predict-normals True ^
  --data C:\nerf_project\processed ^
  --output-dir C:\nerf_project\nerf_outputs
```



Nerfstudio viewer with training screenshots are shown above.

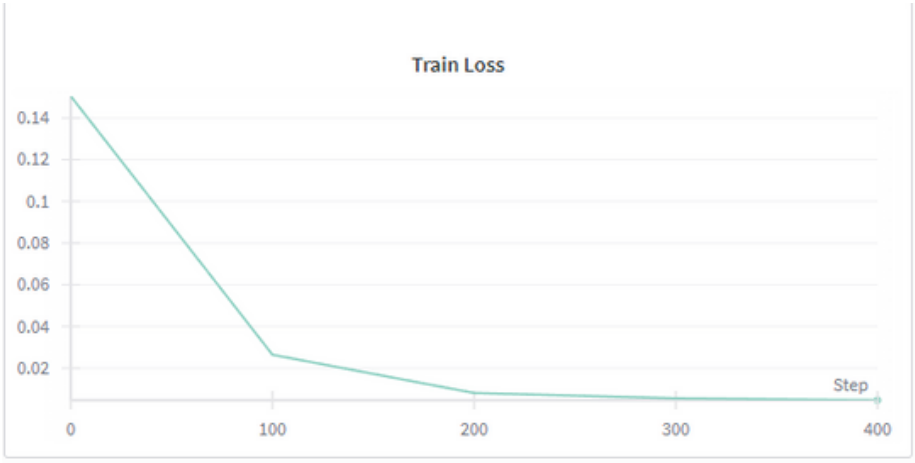
Because of the extremely sparse camera poses, further training would not yield meaningful reconstruction.

Weights & Biases was used to :

1. Track training loss over iteration
2. Monitor GPU memory usage
3. Observe training time per iteration
4. Estimate remaining training time (ETA)

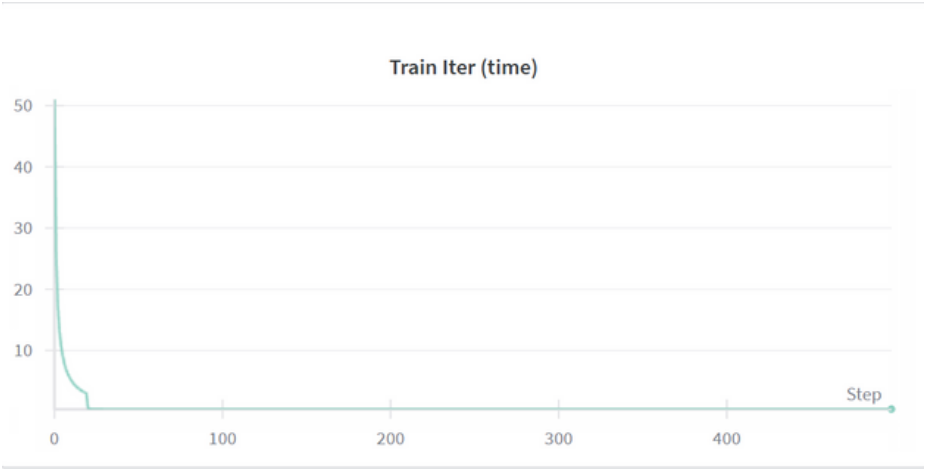
4.1 Training metrics observed

4.1.1 Training Loss



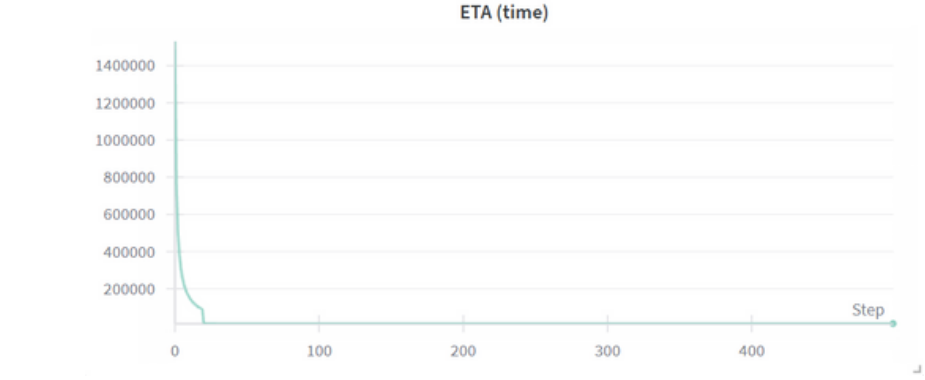
- The training loss showed a rapid decrease in early iterations, followed by a gradual decline.
- This behavior indicates that the model quickly learned coarse scene geometry and then refined appearance details.
- A smooth loss curve confirmed stable optimization and correct camera pose estimation.

4.1.2 Training Iteration Time



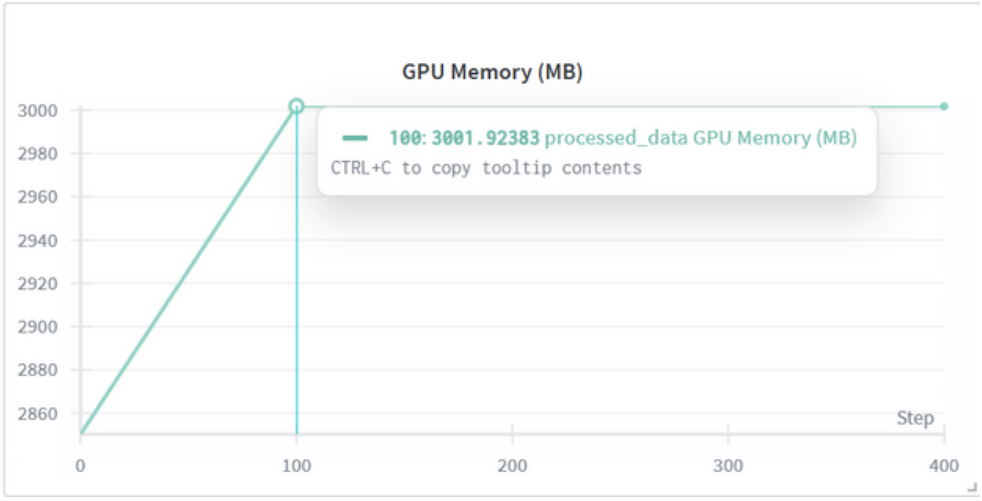
- The time per iteration was initially high due to model initialization and CUDA kernel setup.
- After the warm-up phase, iteration time stabilized, indicating efficient GPU utilization.

4.1.3 Estimated Time of Completion (ETA)



- Early ETA values were high and unstable.
- As training progressed, ETA estimates became more accurate, which is expected behavior in iterative training pipelines.

4.1.4 GPU Memory Usage



- GPU memory usage increased during the initial phase and then remained constant.
- This confirmed that the Nerfacto model fit well within GPU memory and that no memory leaks occurred.

5 . Exporting the point cloud

```
ns-export pointcloud ^
  --load-config C:\nerf_project\nerf_outputs\nerfacto\2025-01-02_22-10-45\config.yml ^
  --output-dir C:\nerf_project\nerf_outputs\exports ^
  --num-points 1000000 ^
  --remove-outliers True ^
  --normal-method open3d
```

Even after incomplete NeRF training, a dense point cloud was exported using the ns-export pointcloud utility. Outlier removal and normal estimation were applied to improve geometric quality. It can be viewed as below :