MP Project 2 Report

Members: Johnny Lin, Matt(Zitong) Wei, Boshen Pan

Introduction:

In this project, we aim to generate 3D objects and scenes using NeRF, leveraging images captured from different viewpoints. The project will be conducted using Nerfstudio, a framework that facilitates the use and modification of NeRF models for 3D reconstruction. The core objective is to verify the feasibility of NeRF models in generating high-quality 3D scenes and understanding the training process involved.

Training Process:

1. Verifying the Python Version

   During the setup process, we encountered a requirement for the correct Python version to install tinycudann. Specifically, the installation of tinycudann requires Python 3.10. To ensure compatibility, we used the following command: !pip install tinycudann-1.7-cp310-cp310-linux_x86_64.whl. This command is tailored for Python 3.10, which matches the version we are using in our environment.

```
[ ] !python --version

Python 3.10.16
```

2. Preparing the Datasets

   This project requires many images captured from various orientations of the object to serve as the dataset. Collecting such data manually can be time-consuming and inconsistent. Therefore, our primary approach is to utilize public datasets from the internet. However, we will also experiment with our own collected datasets for comparison and validation.

```
[ ] import os
    custom_dir = "/content/data/nerfstudio/custom_data/raw_images"

    if not(os.path.exists(custom_dir)):
      os.makedirs(custom_dir)

    custom_vid = "/content/data/nerfstudio/custom_data/raw_video"

    if not(os.path.exists(custom_vid)):
      os.makedirs(custom_vid)
```

3. Installing Nerfstudio and Dependencies

   Following the official Nerfstudio installation guide, we will install the required packages and dependencies using the provided instructions. Once the installation is complete, we will have access to essential tools and commands, such as COLMAP and tinycudann, to proceed with the project.

```
[ ] %cd /content/
    !pip install --upgrade pip
    !pip install torch==2.0.1+cu118 torchvision==0.15.2+cu118 torchaudio --extra-index-url https://download.pytorch.org/whl/cu118

    # Installing TinyCuda
    %cd /content/
    !gdown "https://drive.google.com/u/1/uc?id=1-7x7qQfB7bIw2zV4Lr6-yhvMpjXC84Q5&confirm=t"
    !pip install tinycudann-1.7-cp310-cp310-linux_x86_64.whl

    !pip install -q condacolab
    import condacolab
    condacolab.install()
    !conda install -c conda-forge colmap

    # Install nerfstudio
    %cd /content/
    !pip install git+https://github.com/nerfstudio-project/nerfstudio.git
```

4. Verifying COLMAP Installation

   In this step, we ensure that COLMAP is installed correctly, as it is crucial for the training process. COLMAP plays a key role in tasks like downsampling images and processing the input data efficiently.

5. Dataset Verification

   For custom datasets, we need to confirm that the data is successfully loaded into the target directory before starting preprocessing. Additionally, it is important to verify that all images are in the same format to maintain consistency throughout the process.

6. Process the images

   This is the custom images we used.

```
scene = 'custom'
!ns-process-data images --data /content/data/nerfstudio/custom_data/raw_images --output-dir /content/data/nerfstudio/custom_data/ --sfm-tool colmap


#!ns-process-data video --data /content/data/nerfstudio/custom_data/raw_video/{vid}.mp4 --output-dir /content/data/nerfstudio/custom_data/ --sfm-tool col

# scene = 'poster'
# %cd /content/
# !ns-download-data nerfstudio --capture-name=$scene
```

```
[20:31:28] 🎉 Done copying images with prefix 'frame_'.
```

   This is the public dataset we used.

```
[ ] #scene = 'custom'
    #!ns-process-data images --data /content/drive/MyDrive/nerfdata/image --output-dir /content/drive/MyDrive/nerfdata/ --sfm-tool colma


   #!ns-process-data video --data /content/drive/MyDrive/nerfdata/video/{vid}.mp4 --output-dir /content/drive/MyDrive/nerfdata/ --sfm-to

   scene = 'dozer'
   %cd /content/
   !ns-download-data nerfstudio --capture-name=$scene
```

7. Initializing the Interactive Viewer

   The interactive viewer in Nerfstudio plays a crucial role in this project. It provides a platform to adjust parameters and fine-tune the images interactively. Additionally, the viewer allows us to design camera trajectories and observe how the images evolve during the training process.

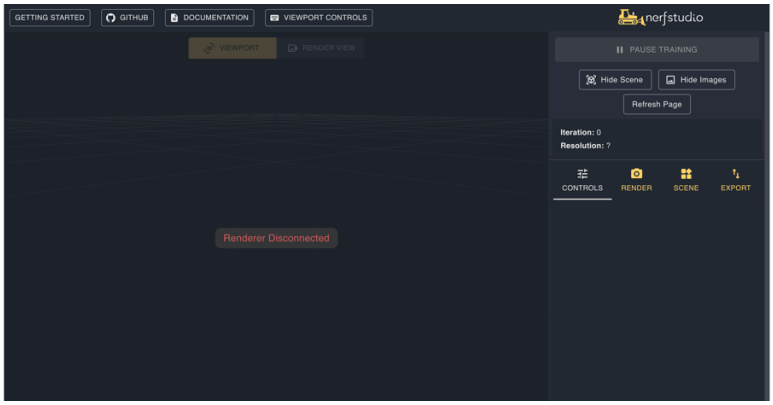
```
[ ] import os
    %cd /content

    # Install localtunnel
    # We are using localtunnel https://github.com/localtunnel/localtunnel but ngrok could also be used
    !npm install -g localtunnel

    # Tunnel port 7007, the default for
    !rm url.txt 2> /dev/null
    get_ipython().system_raw('lt --port 7007 >> url.txt 2>&1 &')

    import time
    time.sleep(3) # the previous command needs time to write to url.txt


    with open('url.txt') as f:
      lines = f.readlines()
    websocket_url = lines[0].split(": ")[1].strip().replace("https", "wss")
    # from nerfstudio.utils.io import load_from_json
    # from pathlib import Path
    # json_filename = "nerfstudio/nerfstudio/viewer/app/package.json"
    # version = load_from_json(Path(json_filename))["version"]
    url = f"https://viewer.nerf.studio/?websocket_url={websocket_url}"
    print(url)
    print("You may need to click Refresh Page after you start training!")
    from IPython import display
    display.IFrame(src=url, height=800, width="100%")
```



8. Start the training

   Then we can start the training process and see how the image is generated on the viewer.

```
[ ] %cd /content

    if os.path.exists(f"data/nerfstudio/custom_data/transforms.json") and scene == 'custom':
        # Custom data
        !ns-train nerfacto --viewer.websocket-port 7007 nerfstudio-data --data data/nerfstudio/custom_data --downscale-factor 4
    elif os.path.exists(f"data/nerfstudio/{scene}/transforms.json") and scene == 'dozer':
        # Pre-processed dataset
        !ns-train nerfacto --viewer.websocket-port 7007 nerfstudio-data --data data/nerfstudio/$scene --downscale-factor 4
    else:
        print('Preprocessing not complete')
```

9. Training Process Analysis

Step (% Done)	Train Iter (time)	ETA (time)	Train Rays / Sec
660 (2.20%)	119.006 ms	58 m, 11 s	40.30 K
670 (2.23%)	119.594 ms	58 m, 27 s	40.17 K
680 (2.27%)	120.291 ms	58 m, 46 s	40.07 K
690 (2.30%)	120.117 ms	58 m, 40 s	40.05 K
700 (2.33%)	121.150 ms	59 m, 9 s	39.40 K
710 (2.37%)	122.375 ms	59 m, 44 s	38.87 K
720 (2.40%)	122.456 ms	59 m, 45 s	39.02 K
730 (2.43%)	123.258 ms	1 h, 0 m, 7 s	38.73 K
740 (2.47%)	122.736 ms	59 m, 51 s	38.79 K
750 (2.50%)	121.541 ms	59 m, 15 s	39.32 K

Training Parameters:

The graph displays the training metrics, including iteration time, estimated completion time, and rays processed per second.

Performance Evaluation:

The public data showed faster progress and higher quality results compared to custom datasets due to better image diversity and coverage.

The training viewer allowed us to monitor parameter adjustments, model convergence, and output quality interactively.

10. Convert simple captures into 3D videos

Video_cup.mp4

Video_dozer.mp4

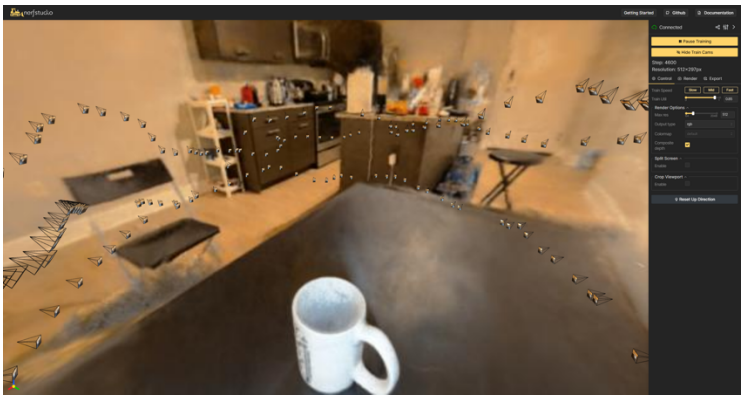
After we complete the training process, the next step is to launch the nerfstudio web-based viewer to visually examine and adjust our 3D model. First, we open the link provided after training, which is an address that is going to start with “http://localhost:7007.” If our training took place on a remote machine, we could still access the viewer by setting up an SSH tunnel. This ensures that we can view the scene directly in our web browser, regardless of where the model was trained.

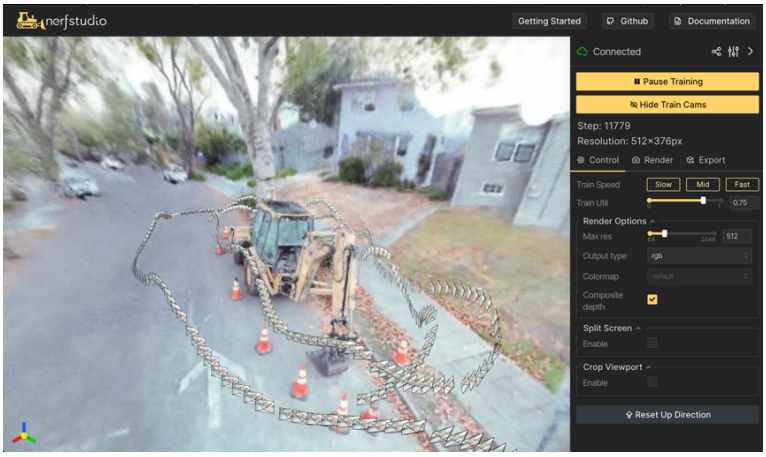
Next, once we have entered the viewer, we can observe our 3D environment as it forms. Each camera’s position will be displayed around the target. Also, we can move the scene around to better examine the training result, including both rotational and translational movements. Furthermore, we can adjust the visibility of objects to focus on specific areas that require closer inspection. Also, if we want to improve the training speed or try different output settings, the viewer provides tools to modify these parameters without difficulty.

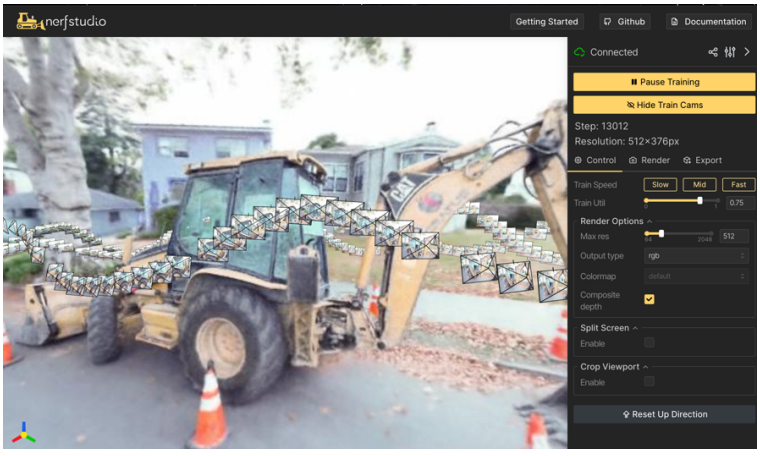
After that, we can create a custom camera path to produce new videos from the trained model. We do this by adding and arranging several virtual cameras that focus on the target, which is a mug in our case. Then, we adjust the movement speed and smoothness of the camera path, which can be done easily in the function bar. Once we have established a satisfactory camera path, we can preview the final trajectory to ensure it meets our requirements. Finally, after confirming that the path and settings are correct, we generate a command to render the video. By pasting this command into our terminal, we produce the finished video with the camera path moving around a mug.

In summary, the nerfstudio web-based viewer allows us to seamlessly review our trained models, refine their appearance, and create compelling videos. Each step, from initial viewing to final video creation, is designed to be straightforward, making it easy to monitor training results and render new videos.

Results:




nerfstudio
Getting Started
Github
Documentation
Connected
Pause Training
Hide Train Cams
Step: 11779
Resolution: 512×376px
Control
Render
Export
Train Speed
Slow
Mid
Fast
Train Util
0.75
Render Options
Max res
512
Output type
rgb
Colormap
default
Composite depth
Split Screen
Enable
Crop Viewport
Enable
Reset Up Direction


nerfstudio
Getting Started
Github
Documentation
Connected
Pause Training
Hide Train Cams
Step: 13012
Resolution: 512×376px
Control
Render
Export
Train Speed
Slow
Mid
Fast
Train Util
0.75
Render Options
Max res
512
Output type
rgb
Colormap
default
Composite depth
Split Screen
Enable
Crop Viewport
Enable
Reset Up Direction

Custom Images:

The second and third images demonstrate the model's output using our custom images. The reconstructed images were initially blurry but progressively became clearer as training continued. However, artifacts and inconsistencies were still noticeable, likely due to insufficient image coverage or fewer input viewpoints.

Public Images:

The fourth and fifth images showcase results using the public images. The outputs were significantly clearer and more detailed, even at earlier stages of training. The dataset's higher quality, greater coverage, and diverse orientations contributed to the improved reconstruction results.

Discussion:

This project highlights the strengths and challenges of training NeRF models for 3D scene reconstruction:

1. Dataset Size and Quality:

High-quality 3D reconstructions require datasets with hundreds of images captured from diverse angles. Custom datasets require careful planning to ensure proper image coverage, resolution consistency, and lighting uniformity.

2. Training Efficiency:

Public datasets are more efficient for training, as they are curated for NeRF-based tasks. Custom data increases training time and may result in lower-quality outputs if the dataset is insufficient.

3. Interactive Monitoring:

The Nerfstudio viewer proved valuable for real-time monitoring, parameter tweaking, and visualizing the training process. By observing metrics such as training speed, rays/sec, and output quality, we could optimize performance during runtime.