Calculating depth using a Neural Radiance Field

Evan Krainess
krain022@umn.edu

Evan Pochtar
pocht004@umn.edu

Hady Kotifani
kotif004@umn.edu

William Anderson
and09931@umn.edu

I. Abstract

This project investigates the potential of Neural Radiance Fields (NeRF) for depth estimation, focusing on their ability to reconstruct 3D scenes and generate high-quality depth maps from 2D image inputs. Depth estimation is essential for various applications, including robotics, augmented and virtual reality, autonomous vehicles, and computer vision. NeRF offers a promising alternative to traditional depth estimation techniques, excelling in generating accurate reconstructions of complex scenes. The focus of our work was on improving the implementation and performance of NeRF-based depth estimation rather than tackling challenges like sparse imaging or transparent object detection, as explored by advanced techniques such as SinNeRF and Dex-NeRF. Instead, we concentrated on reimplementing and optimizing the TinyNeRF algorithm in PyTorch, which allowed us to enhance the computational efficiency and quality of depth outputs.

Key achievements of this project include accelerating the NeRF algorithm, creating custom functions to generate diverse depth outputs, and improving the accuracy of these outputs. Notably, we applied Gaussian blurring to the input data to mitigate noise and refine depth calculation, resulting in more precise and visually coherent depth maps. By iteratively refining our approach, we observed significant improvements in both computational speed and output quality. Our work demonstrates how fundamental adjustments, such as enhanced preprocessing and optimized implementation, can elevate the performance of NeRF in depth estimation tasks. This study underscores the practicality of NeRF for generating detailed depth maps and sets the foundation for future applications in 3D scene reconstruction and beyond.

All code for the project can be found here: NeRF or Nothin GitHub Repository.

II. Introduction

In this project, we aim to explore how effectively Neural Radiance Fields (NeRF) can estimate depth by leveraging their ability to reconstruct 3D scenes from 2D images. Depth data is particularly important in fields like robotics, where tasks such as obstacle detection, navigation, and object manipulation require an accurate understanding of the 3D structure of the environment. NeRF's ability to provide detailed depth maps can help robots interact with and navigate through complex scenes, even in situations where traditional sensors struggle, such as with transparent or reflective surfaces. Beyond robotics, accurate depth estimation has applications in augmented reality and virtual reality, where understanding the spatial data of real-world objects is essential for creating immersive experiences. NeRF can also be useful in autonomous vehicles, where precise depth maps can improve obstacle detection and path planning. Additionally, in the field of computer vision, NeRF's depth data can support tasks like 3D reconstruction, scene understanding, and object tracking. By testing NeRF's performance in these areas, we hope to demonstrate its potential to enhance depth estimation in a variety of real-world applications, although we will not be making these real-world applications themself. This is why our project focuses on building and evaluating a NeRF model to generate accurate depth maps from 2D images. We plan to capture images of simple scenes or objects from multiple angles to cover a full range of perspectives. For these scenes, we will use public datasets that already include depth information or generate synthetic data using tools like BlenderNeRF. Once we have the images, we will train our NeRF model by casting rays from the camera through each pixel to reconstruct the 3D scene. NeRF's ability to simulate how light behaves within a scene allows it to capture the underlying geometry, from which we can extract depth information. After training, we will generate depth maps by querying the model for the depth at each pixel and compare these maps to the ground-truth data. By measuring the accuracy with metrics like mean absolute error, we will evaluate how well NeRF performs in various scenarios. We aim to experiment with different factors, such as the number of input images and the complexity of the scenes, ultimately comparing NeRF's depth estimation capabilities with other depth prediction methods.

III. Related Work

A. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.

The original paper that introduced the concept of a neural radiance field is named "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." A Neural Radiance Field (NeRF) models a 3D scene by taking 5D inputs comprising a 3D position (x,y,z) and two angles (θ, ϕ) [1]. As output, the Neural Radiance Field returns a color value (rgb) along with the volume density σ (opacity). The general process of a NeRF includes three steps. The first step is to trace camera rays through a 3D scene to generate points. The second step is to use those points as input to a multilayer perceptron (MLP). The third step is to use those densities and colors to create a 2D image. To ensure multiview consistency in the MLP, the network can only predict the volume density σ as a function of the location (x,y,z). When predicting the color value, the MLP has access to all inputs. However, optimizing the neural radiance field's representation for complex scenes is often inefficient and yields suboptimal results. To improve upon the method, the authors use positional encoding to allow the MLP to learn higher-frequency functions [1]. It has been shown that deep networks are biased towards low-frequency functions; therefore, mapping the input data to a higher-dimensional space enables the MLP to learn the function better. Additionally, the authors introduce a technique called hierarchical sampling. This method addresses the inefficiencies of sampling each point along a camera ray, as many of these points will not contribute to the final image. The solution is to use two networks: a 'fine' network and a 'coarse' network [1]. The coarse network uses stratified sampling to generate a set of points, which are then evaluated. Using these evaluations, the fine network samples a more focused subset of points. The final result combines the outputs from both the coarse and fine networks. The authors reveal that their model outperforms current methods, demonstrating substantial improvements in performance over other methods, as measured by the Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS). The methods NeRF is compared to are Scene Representation Networks (SRN), Neural Volumes (NV), and Local Light Field Fusion (LLFF). The paper also provides a time and space comparison between these methods. LLFF can compute the scene in 10 minutes, whereas other methods, such as NeRF, take up to 12 hours. However, the storage space for LLFF is massive, and NeRF is 3000 times less storage-intensive [1]. This paper is seminal to our project, and methods from it will be used extensively to calculate and create depth maps.

B. "SinNeRF": Training Neural Radiance Fields on Complex Scenes from a Single Image

A new interesting development in NeRFs is using a single image to train the model. In the paper "SinNeRF: Training Neural Radiance Fields on Complex Scenes from a Single Image," the authors discuss an approach to accomplish this task using a new model named SinNeRF. Training a normal NeRF with sparse imagery will lead to incorrect geometry and blurry photos [3]. Since traditional NeRFs are prone to overfitting, the goal of this method is to train on only a single image, this poses a significant problem. To address this, the authors build a "semi-supervised framework." In this setup, the output from the NeRF model for the single available image is treated as the labeled set, while the unseen views are treated as the unlabeled set [3]. To help the NeRF model learn the scene's geometry from a single image, the authors utilize a depth prior. Additionally, they propagate information from pixels in the seen view to corresponding pixels in multiple unseen views through image warping. This propagation helps the model handle the inherent challenge of learning 3D geometry from a single view, which would otherwise lead to overfitting. Through image warping, the authors are then able to create the depth map of an unseen view. To account for uncertain regions caused by the warping, the authors use a self-supervised inverse depth smoothness loss to smooth the predicted depths. While image warping helps obtain depth information for unseen views, it doesn't address color and texture inconsistencies. To resolve these issues, the authors propose using a generative adversarial network (GAN) for local texture guidance, ensuring finer texture details are preserved. Additionally, they employ a pre-trained Vision Transformer (ViT) to ensure consistency between the unseen views and the original image [3]. Using these methods, the authors show that, compared to other sparse imaging NeRFs, SinNeRF "achieves the best results both in pixel-wise error and perceptual quality" [3]. The authors also demonstrate that SinNeRF outperforms the other models using the three metrics from the original NeRF paper (PSNR, SSIM, LPIPS). This paper presents a novel way to create a NeRF using sparse imagery and represents a significant development that we will use in our project.

C. "Dex-NeRF: Using a neural radiance field to grasp transparent objects."

Another application of NeRFs is the detection of transparent objects. In the paper "Dex-NeRF: Using a Neural Radiance Field to Grasp Transparent Objects," the authors propose the use of NeRFs in detecting transparent objects so robots can grasp and manipulate the objects. Traditional depth cameras assume that objects reflect light uniformly, which does not hold for transparent objects [16]. The solution proposed in the paper is a model called "Dex-Nerf," which is used to sense the geometry of transparent objects. The generated depth map from Dex-Nerf is then pipelined for robot-object manipulation–the pipelining process is not relevant to our study. The inputs are the same as traditional NeRFs, being the set of images taken from the camera and the known intrinsic camera properties and extrinsic position and orientation parameters [16]. Prior work on detecting transparent objects is "data-driven" and prior methods used Convolutional Neural Networks, deep-learning models, or random forests, while "Dex-NeRF does not require any prior dataset" [16]. The other benefit of NeRFs is that they output a volume density and view-dependent emitted radiance at a coordinate that can represent "non-Lambertian effects" (specularities and reflections) and thus capture the geometry of transparent objects [16]. The method proposed starts with a traditional NeRF, training the multilayer perceptron with "multi-view" RGB images of the scene and the intrinsic and extrinsic parameters, while error is minimized between rendered and ground truth rays colors using gradient descent [16]. NeRFs do not directly support transparent object effects, but the incorporation of "the viewing direction in its regression and supervising with view-dependent emitted radiance allows recovery of non-Labertian effects" like reflections from a specular surface [16]. During training, "a NeRF model learns a density σ of each spatial location". The density represents "the transparency of the point". Additionally, the volume density σ is not view-dependent, so a non-zero value represents that

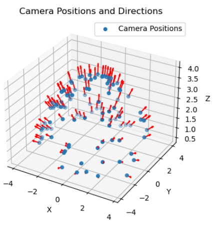


Fig. 1: Tiny Nerf Data Poses

some color (or physical property) exists at that spatial location. The raw value σ is then converted to an occupancy probability bounded by 1 representing distance along a ray; however, just using the raw value can "determine if a point in space is occupied" [16] Knowing this, the model searches for a sample along a ray for which σ, the raw value, is greater than some value, m, and the depth is then set to the distance of that sample [16]. Low values of m "result in noise depth" while higher values result in "holes" or "no detection" of the transparent object in the depth map. The paper determined an optimal value of m=15 which resulted in the best detection of the object [16]. The paper also suggests that adding multiple light sources to the scene increases the accuracy of the model because it would result in more specular reflections from more angles, increasing the value of σ at a certain point [16]. The results of the paper showed that NeRF could "recover the geometry of transparent objects" [16].

IV. Approach

A. Datasets

Two datasets were used for our analyses. First, we used a down-scaled version of the lego excavator from the LLFF dataset. The lego excavator dataset was provided to us by the Google Berkeley NeRF [1]. This lego excavator was mostly uniform in color and was set on a transparent background. The scene in this dataset can be described as a sphere of radius 3. Camera poses were sampled on the positive-Z portion of the sphere. In other words, only poses showing the top of the excavator were used. This dataset consisted of 100 samples, and contained only images rendered in RGB. Throughout this paper, this dataset may be referred to as the TinyNeRF dataset since it was the dataset used in the original paper [1]. This dataset was mostly free from noise, and the original TinyNeRF converged very quickly with it, with the excavator being visible after only approximately 100 iterations.

To relinquish concerns about this dataset being optimized for NeRFs, we decided that it would be best to use additional datasets. We explored using datasets like NYUv2, KITTI, and Make3D, but these datasets were too complex for the compute power that we had access to. This problem made it very difficult to find a dataset that fit our circumstances. In our early research, we noticed that many existing NeRFs use "synthetic data", or data created by people or a computer. This differs from real data that would have to be collected in the real world. Generally, Blender is used to generate synthetic datasets, and it offers a lot of customization. Additionally, a Blender, BlenderNeRF plugin exists for this purpose [20]. This plugin contains 3 primary functionalities: subset of frames, train and test cameras, and camera on sphere (COS). Subset of frames allows you to follow a camera through an animation and sample the scene along that camera's path. Test and train camera utilizes subset of frames, but runs on two different cameras to create a test and a train dataset. Lastly, COS allows you to define a sphere around the scene and take samples along that sphere. We were most interested in COS, because that seemed to be the way poses were gathered for most synthetic NeRF datasets, like the lego excavator. Experimenting with BlenderNeRF revealed some issues in this approach. First, Blender is a huge and expansive tool, and none of us had experience with it. The COS method was mostly automated, however the process of executing this method was still fairly involved on the Blender frontend. Additionally, we couldn't find documentation about where and how camera poses were stored within a BlenderNeRF dataset. To compensate for this and our lack of Blender knowledge, we decided to create our own custom Python scripts along with bpy, the Blender Python library, to create our datasets. This was advantageous because it allowed us to control, very granularly, our sample count, sample resolution, camera parameters, camera locations, and output, which included camera poses. Additionally, we were able to use the Blender compositor, and its Python functions, to generate Z-pass images. Z-pass is the Blender view layer that encodes depth from any perspective, so this data could act as our ground truth. We also created Python scripts to generate .npz files from our datasets, since this format is how most NeRFs, and our PyTorch NeRF, would read datasets. For the scene itself, we used a model from free3d.com with a personal use license. We tried to select a scene with high contrast and occlusion of objects from different perspectives. We settled on a dataset with 4 lego bricks of different colors in different orientations. A white background was then applied to ensure that the scene was equally lit. Upon further testing, we noticed that the NeRF took significantly longer to converge on our dataset than with the TinyNeRF dataset, so preprocessing was done. First, we normalized the image. Next, we applied a Gaussian blur using CV2 in an attempt to reduce noise.

Fig. 2: Tiny Nerf Data Example

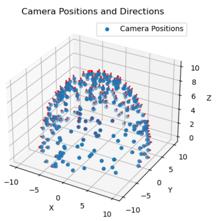


Fig. 5: Custom Data Poses

After this preprocessing, our final dataset was 400 images of our lego scene, evenly distributed across the positive-z portion of a sphere centered on the scene. The data was lightly blurred. Each RGB image had a corresponding, unblurred, depth image.

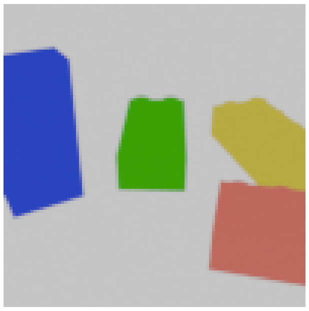


Fig. 3: Custom Data RGB Example

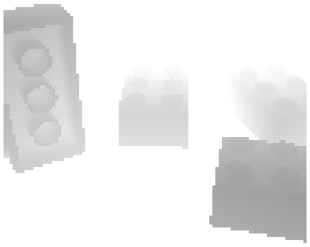


Fig. 4: Custom Data Depth Example

B. Exploration Process

Our exploration consisted of analyzing, running, testing, and modifying various NeRF algorithms to understand their underlying structure, optimize them to run faster on our machines, and produce accurate depth data on complex scenes compared to other depth-calculating algorithms. We initially explored the original NeRF algorithm described in "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis" [1]. This NeRF uses high-resolution images and scene views–specifically, this NeRF is trained using 100 images with a resolution size of 800 pixels. The approximate training time for this NeRF is rather long–about 15 hours. An unexpected difficulty with this specific algorithm, in addition to training time, was memory use. We used this algorithm to look at places for improvement and understand the more complex makeup of the NeRF system. Our next approach was to explore TinyNeRF, a modified, simplified version of the original NeRF algorithm that trains with a lower resolution count of 100 pixels with 100 images. This algorithm runs faster at about five hours for completion and provides a good baseline for further optimization. The other benefit of working with this algorithm is that it ran within the Jupyter Notebook which allowed ease in re-runability and testing modifications. Our first modifications for speedup looked at reducing and eliminating the RGB components (later reverted) and removing unnecessary functions. For modifications related to depth estimation, we made modifications to generate depth maps, depth map interactivity, and test depth estimations. Developing upon TinyNeRF, we explored DexNerf [16] and began to integrate its optimized components for speed up since DexNeRF was shown to have significant speedup compared to the original NeRF algorithm. Additionally, we recognized that TinyNeRF was composed of TensorFlow functions and written in Notebook. As an avenue for speedup, we translated TinyNeRF from tensorflow to pytorch, using other Pytorch NeRFs for reference, and ran the algorithm outside of the notebook. Later, we removed DexNerf integrations which seemed to cause inaccuracies and slowdowns. For our last avenue of speedup, we explored NeRFs that could run with a sparse dataset (fewer images) but found worse results and abandoned that approach

V. Experiments and Results

A. Speed Up

The original NeRF algorithm takes approximately 15 hours to run [1], with TinyNeRF taking about three times faster on CPU. Our first interation upon TinyNERF, without DexNeRF produced speed up in which our algorithm was 1.4x Faster than TinyNeRF on GPU and 1.7x Faster than TinyNeRF on CPU. Our final algorithm, pytorchNeRF without DexNeRF integration, was 5.4x Faster than our updated TinyNeRF and 1.7x Faster than other Pytorch NeRFs on CPU, and 2.1x Faster than our Updated TinyNeRF and 1.3x Faster than other Pytorch NeRFs on GPU.

B. Comparison Algorithms

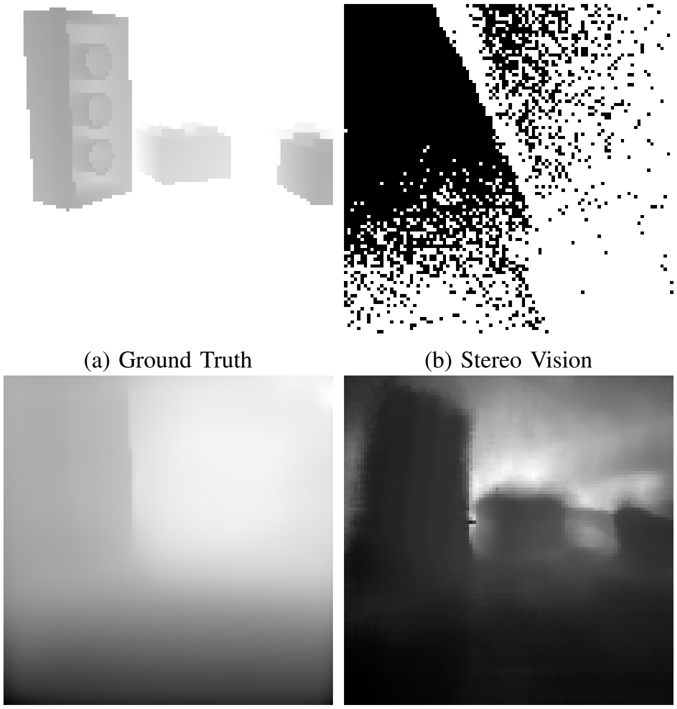
1) MiDaS: For our experiments, we leveraged MiDaS v2.1 [21]. MiDaS is a deep learning network that specializes in the prediction of depth maps based on the input of a single RGB image. MiDaS comes in two different sizes. The large model prioritizes accuracy over speed. The small model is for real time applications and prioritizes speed foremost. For our experiments we used the large model. MiDaS is trained on multiple datasets and is able to generalize to other images easily.

2) Stereo: The second comparison algorithm we used leveraged techniques from stereo vision. Stereo vision is a method that utilizes 2 different views of a similar scene. By identifying corresponding points in both images, stereo vision can compute the corresponding 3D coordinates of those points. Using stereo we were able to construct a 3D point cloud of the scene. This method allowed for precise depth estimation for scenes that have a lot of identifiable correspondences.

C. Performance Evaluation

To evaluate performance quantitatively, we used three metrics: Structural Similarity Index Measure (SSIM), Mean Absolute Error (MAE), Mean Squared Error (MSE). SSIM assesses the overall similarity between images by considering the structural information and texture. Higher SSIM indicates that the images are more similar. MAE is used to measure the average differences between the predicted depth values and the actual values. MSE is used to measure the average squared differences between the predicted depth and the actual depth values. A lower MSE and MAE both indicate that the images are more similar. We will also evaluate performance on a qualitative level by inspecting the output of the depth maps to see which model most correctly represents the actual ground truth.

D. Experiments


(a) Ground Truth

(b) Stereo Vision

(c) MiDaS

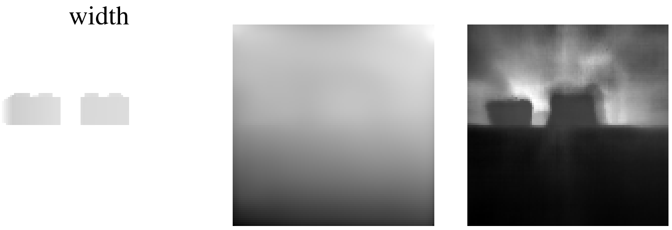
(d) NeRF

Fig. 6: Comparison of depth maps from Ground Truth, Stereo Vision, MiDaS, and NeRF.

TABLE I: Error metrics for NeRF, MiDaS, and Stereo Vision.

Method	**MAE**	**MSE**	**SSIM**
NeRF	0.6222 ± 0.0602	0.4339 ± 0.0821	0.1200 ± 0.0378
MiDaS	0.3959 ± 0.0734	0.2141 ± 0.0520	0.2556 ± 0.0846
Stereo Vision	0.7443 ± 0.1746	0.7116 ± 0.1759	0.0160 ± 0.0424

Our experiments involved creating depth maps using the three outlined methods: NeRF, MiDaS, and Stereo. Using the dataset outlined earlier, 300 depth maps were created from each method. Using the three metrics outlined above, we averaged the metrics across all 300 depth maps to produce Table I. The table shows that our NeRF method for estimating depth across images underperformed compared to MiDaS. MiDaS outperforms in all three metrics, showing an improvement of 36% in MAE, 50% in MSE, and 112% in SSIM. However, NeRF is able to outperform the Stereo method, showing a 20% improvement in MAE, a 64% improvement in MSE, and an 86% improvement in SSIM. However, qualitatively, the NeRF model was able to output better images that more accurately represented the depth of the scene. Looking at Fig. 6, it is apparent that NeRF bears the closest resemblance to the ground truth. However, this result did not show in our quantitative results. A possible reason for this is that the background of the ground truth is white, which could throw off the error for the metrics and also not align with the original goal of using metrics to evaluate the quality of the depth maps. To address this issue, a possible solution would be to segment the images to ensure that the background is not interfering with the error. Another issue was that the background of the dataset is one solid color, which created noise when it came to the horizon line. There were also issues in fine-tuning the hyperparameters, which proved challenging on the custom data. An explanation for the higher error seen in the Stereo Vision model is most likely due to issues in finding correspondences between the images. The dataset used lacks meaningful texture and has a lot of similar pixels, which did not allow SIFT to find many correspondences. To further reinforce the idea that, although the MiDaS model outperformed NeRF quantitatively, it is not indicative of a better depth representation, observe Fig. 7. The figure shows that MiDaS is not able to recover the depth of any of the Lego bricks from the dataset; however, NeRF is able to grasp those objects and assign a depth to them. The majority of the MiDaS depth maps were like this, and the quantitative error was lower since the background was a lighter color, which is similar to the ground truth background.



(a) Ground Truth

(b) MiDaS

(c) NeRF

Fig. 7: Quantitative comparison of MiDaS and NeRF.

VI. Conclusions

Our work applies the prior knowledge already developed for representing scenes as 5D neural radiance fields to produce depth renders that compete with other traditional depth approximation approaches. Our approach produced speed up compared to the traditional NeRF and demonstrated the usability of NeRF systems to produce depth data. Specifically, the depth data produced a better understanding of the scene compared to other depth approximation but had mixed results for error improvements compared to other depth-aproximation algorithms.

Our work was limited by the computational intensity of running NeRFs preventing us from working with higher resolution images and is limited in field usability due to the number of images required to train the network. New work could further explore the integration of sparse imaging NeRFs with our work. Another avenue for future work may also look at the integration of NeRF-generated depth data into other systems. Additional work may look into optimizing the runtime of the system, specifically optimizing for images of higher fidelity.

VII. Individual Contributions

A. William Anderson

I focused on collecting and creating datasets, as well as testing the NeRF. I learned how to use Blender to render images, and I learned quite a bit about bpy, the Blender Python library. Blender was a big learning curve, and bpy was very poorly documented. I wrote the scripts in Data Utils, which took a very long time to create due to bugs and lack of documentation in bpy. I had difficulty finding the correct data for camera poses and intrinsics within bpy. I also learned a lot about data pre-processing. I implemented the Gaussian blur to speed up convergence in our dataset. I also did most of the scene-specific model tuning for our dataset, including finding an adequate number of iterations, model parameter size, and batch size per iteration. Additionally, I spent a lot of time testing other NeRFs like "SparseNeRF", "pixel-nerf", and "MultiNeRF". I was surprised by the amount of technical knowledge necessary to run, let alone develop, a NeRF. Complex knowledge of CUDA, virtual environments, and GPU architecture are necessary to develop a NeRF, so I felt as if I did a lot more learning and debugging than developing throughout this project.

B. Evan Pochtar

Evan Pochtar focused on the performance upgrades of the NeRF algorithm, in this vein, Evan P created the updated TinyNeRF code, as well as the final Pytorch TinyNeRF code. Evan P also worked on getting results, such as creating the code for the comparison MiDaS function, as well as the error generation code (SSIM, MAE, MSE). Evan P also put some effort into working on depth optimizations/outputs and hyperparameter tuning, as well as doing the writing for the check-in. For algorithms tested before finally settling down only Pytorch TinyNerf, Evan tested out "Sparf", "Neo-360", "Know-Your-Neighbors NeRF", and "dense_depth_priors_nerf".

Working on this project was definitely a big learning experience, as it was my first time using neural networks in computer vision. Rewriting TinyNeRF in both TensorFlow and PyTorch gave me a solid understanding of how these frameworks work and the different strengths they offer, as well as being my first time working in a large project with either of the languages. PyTorch's flexibility made it easier to experiment with changes, while TensorFlow's structured approach helped me better understand how neural networks process data. Also understanding and working through difficulties with my GPU not accepting tensorflow got me researching a lot about it, and helped me understand how hardware works with machine learning a lot more. Testing other algorithms like Sparf, Neo-360, and Know-Your-Neighbors NeRF showed me how complex and advanced modern computer vision techniques can be, and it gave me a lot of insight into their potential and challenges.

One of the biggest takeaways from this project was realizing just how compute-heavy these tasks can be, especially when working with high-resolution images and 3D data. Training neural networks or improving depth estimations required a lot of computational power, which made it clear how important it is to write efficient code and optimize workflows. I also learned a lot about recent innovations in machine learning while researching these topics, which gave me a better understanding of the cutting-edge developments in the field. Overall, this project was an eye opening experience that taught me practical skills in neural networks, computer vision, and problem-solving in computationally demanding code.

C. Evan Krainess

At the beginning of the project I focused on implementing DeXNeRF in order to create depth maps of transparent images. We later realized that our project scope was too wide and decided to narrow it down. I shifted to focusing on the comparison algorithms and collection of results. I wrote the Stereo vision algorithm. I created and ran the experiments to collect the results used in our paper.

This project was a big learning experience for me. I learned a lot about how to add on to and implement other computer vision projects such as DeXNeRF. I learned how to read an academic paper and apply the ideas stated in it to code. I gained new analysis skills and learned how to consider both qualitative and quantitative results when judging the performance of a model. While creating the Stereo vision method I enjoyed trying out different ideas that were not discussed in class to see if they would lead to better results. Overall this project was a lot of fun to do and I learned a lot about NeRF's and other depth estimation methods.

D. Hady Kotifani

I focused on running the initial NeRF model and optimizing and running the original tinyNeRF to work with fewer parameters and outputting fewer parameters to focus exclusively on depth data. My approach ruled out other methods of speed-up that did not offer speed-up improvement as expected. I also worked on the initial outputs of depth maps and images.

Through this project, I learned in more detail how machine learning algorithms work. I learned how to use different TensorFlow functions and the purpose of several more not having had any prior experience with TensorFlow. I also learned the many pains of training machine learning algorithms, especially with large data sets, how computationally intensive they can be on different hardware systems, and how time-intensive they can be. I also learned different ways of working with and modifying the algorithms, especially since they initially took very long to train which made it difficult to quickly make edits and test the edits to see if they work, as well as methods to work around those. I also learned a lot about how NeRFs work, how they calculate and produce results, and ways to modify their usual application.

REFERENCES

[1] B. Mildenhall, et al., “Nerf: Representing scenes as neural radiance fields for view synthesis,” Communications of the ACM, vol. 65, no. 1, pp. 99–106, 2021.

[2] K. Zhang, et al., “Nerf++: Analyzing and improving neural radiance fields,” arXiv preprint arXiv:2010.07492, 2020.

[3] D. Xu, et al., “Sinnerf: Training neural radiance fields on complex scenes from a single image,” in European Conference on Computer Vision, Cham: Springer Nature Switzerland, 2022.

[4] A. Pumarola, et al., “D-nerf: Neural radiance fields for dynamic scenes,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021.

[5] Y.-C. Guo, et al., “Nerfren: Neural radiance fields with reflections,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022.

[6] A. Yu, et al., “Pixelnerf: Neural radiance fields from one or few images,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021.

[7] Y. Jeong, et al., “Self-calibrating neural radiance fields,” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021.

[8] Z. Wang, et al., “NeRF–: Neural radiance fields without known camera parameters,” arXiv preprint arXiv:2102.07064, 2021.

[9] T. Hu, et al., “Efficientnerf: Efficient neural radiance fields,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022.

[10] A. Yu, et al., “Plenoctrees for real-time rendering of neural radiance fields,” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021.

[11] R. Martin-Brualla, et al., “Nerf in the wild: Neural radiance fields for unconstrained photo collections,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021.

[12] C.-H. Lin, et al., “Barf: Bundle-adjusting neural radiance fields,” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021.

[13] M. Niemeyer, et al., “Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022.

[14] S. Peng, et al., “Animatable neural radiance fields for modeling dynamic human bodies,” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021.

[15] A. Bergman, et al., “Generative neural articulated radiance fields,” Advances in Neural Information Processing Systems, vol. 35, pp. 19900–19916, 2022.

[16] Ichnowski, Jeffrey, et al. ”Dex-NeRF: Using a neural radiance field to grasp transparent objects.” arXiv preprint arXiv:2110.14217 (2021).

[17] Raafat, M. (2024). BlenderNeRF (Version 6.0.0) [Computer software]. https://doi.org/10.5281/zenodo.13250725

[18] Depth Estimation using Monocular and Stereo Cues, Ashutosh Saxena, Jamie Schulte, Andrew Y. Ng. In IJCAI 2007.

[19] 3-D Depth Reconstruction from a Single Still Image, Ashutosh Saxena, Sung H. Chung, Andrew Y. Ng. In IJCV 2007.

[20] Raafat_BlenderNeRF_2024, Raafat, Maxime. https://github.com/maximeraafat/BlenderNeRF

[21] Ranftl, René, et al. ”Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer.” arXiv preprint arXiv:1907.01341 (2020). Available at: https://arxiv.org/abs/1907.01341.