Relightable NeRFs : Rendering Different Lighting Effects with NeRFs

Feiran Wang (feiranw2), Shourya Ghoush (shourya5), Athena Zheng (athenaz2)

Motivation and Impact

A Neural Radiance Field (NeRF) is a method that synthesizes novel viewpoints of a scene. To generate a NeRF scene, a neural network is trained on a collection of 2D images that capture the scene with varying camera angles, outputting a 3D representation of the scene.

As a standard NeRF only takes into account cameras' position and orientation, our project focuses on extending the original architecture in a multitude of approaches so that the lighting is considered, and thus, enabling the scene to be viewed under different lighting effects. Furthermore, we compare different approaches and their effectiveness in relighting the synthesized scenes with two newly published datasets. Motivated by our desire to learn more about NeRFs, we believe that the potential impact of rendering scenes under different lighting conditions is one that helps enhance the versatility of synthesized scenes.

Implementation Details

Our project is based on Python and PyTorch, as the open-source libraries and packages we used and referenced are also generally in Python. The work was done on two machines: one in ECEB 5072 with an Nvidia RTX A2000 with 16 GB, and a Tencent Cloud VM instance with a Nvidia T4 with 16 GB. Lastly, the project requires two components for implementation: a dataset that includes lighting data and is compatible for generating NeRF, and a platform that can train said dataset and synthesize the scene.

Datasets

Relighting NeRF (ReNe): https://eyecan-ai.github.io/rene/ – the ReNe dataset, which contains 20 unique scenes, is created with the use of two robotic arms, one holding the camera and one holding the light source. Each scene is taken with 50 different camera perspectives under 40 distinct lighting conditions, resulting in a total of 2000 images per scene.

OpenIllumination: https://oppo-us-research.github.io/OpenIllumination/ – the OpenIllumination dataset is created with the use of various cameras and controllable lights set up around a central platform. It includes 64 unique scenes under 13 different lighting patterns, each taken with 72 different camera perspectives.

NeRF Implementation Methods

Original NeRF: https://www.matthewtancik.com/nerf – read this paper to understand the original NeRF architecture.

Instant NGP: https://nvlabs.github.io/instant-ngp/ – based on the original NeRF paper, Instant NGP is a method that optimizes the training time. Written mostly in CUDA.

Torch NGP: https://github.com/ashawkey/torch-ngp/tree/main – a PyTorch implementation of Instant NGP, we use Torch NGP to synthesize ReNe scenes. Written mostly in Python.

Instant NSR PL: https://github.com/bennyguo/instant-nsr-pl – another PyTorch implementation of Instant NGP, we use Instant NSR PL to synthesize OpenIllumination scenes. Written in Python.

Approach

We implement our idea in two different ways by synthesizing scenes from two different datasets, Relighting NeRF (ReNe) and OpenIllumination, and inputting it into Instant Neural Graphics Primitives (Instant NGP)'s implementation of NeRF. Both of these datasets are formatted differently, but both provide a variety of lighting conditions. Once we succeed in rendering the scene with the canonical original NeRF architecture, which we call V0, we then build upon it in several different approaches. These approaches are ideas we referenced from the ReNe dataset paper; ReNe does not provide the code for these ideas and thus we implemented them in code entirely by ourselves.

ReNe + Torch NGP

We synthesize ReNe scenes using the Torch NGP PyTorch library in a three-step process. First, we parse the scene dataset for the camera intrinsics parameters and camera transform matrices into an input JSON file using a Python script we wrote, formatted in a manner that can be properly processed by Torch NGP. Next, we then look to train the model and succeed in rendering the scene using V0. Once that is accomplished, we extend V0 in two approaches: ReNe V2 and V3.

ReNe V2 is a NeRF architecture that first constructs a 3D unit vector $l = \frac{t-x}{||t-x||}$ and use $\zeta(l)$ as an additional input into the RGB MLP, where $\zeta(l)$ is the spherical harmonic encoding of l. In the spherical harmonic encoding, l represents the relative position between the light and the camera. As an additional input into the RGB MLP, l should help modulate the color output $(R,\ G,\ B)$ and improve the rendering of lighting-related image artifacts, such as shadows and spectral highlights. ReNe V3 adds extra inputs into the RGB MLP, the camera position $h(x,\ y,\ z)$, wherein h represents the multiresolution hash encoding.

OpenIllumination + Instant NSR PL

We synthesize OpenIllumination scenes in Instant NSR PL. Once the scene is rendered with V0, we extend the basic architecture in three approaches: OpenIllumination V1, V2, and V3.

OpenIllumination V1 (OI_V1) adds the lighting ID as an extra input into the RGB MLP. We also use positional encoding to map the lighting ID to a higher dimension, enabling the MLP to better understand the lighting.

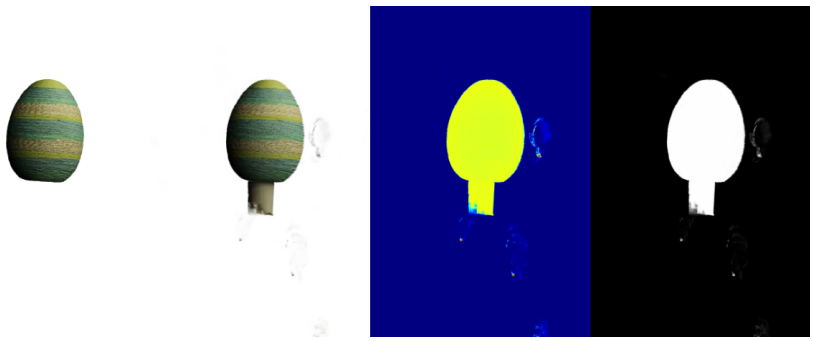
OpenIllumination V2 (OI_V2) builds upon OI_V1 and also asks the RGB MLP to provide an extra output of a predicted lighting ID. Therefore, the overall output of the network is the predicted $(R,\ G,\ B,\ \sigma)$ and a new predicted lighting ID. The predicted lighting ID is used alongside the ground truth lighting ID to calculate the lighting condition loss, which is then used to update the MLP network and optimize the scene.

OpenIllumination V3 (OI_V3) builds upon OI_V1 so that there are two extra inputs into the RGB MLP, the lighting ID and $(x,\ y,\ z)$. Aka, OI_V3 approach is very similar to ReNe V3.

Results

OpenIllumination + Instant NSR PL

We implement our approaches on four different objects with different materials of the OpenIllumination dataset. We firstly train on lighting conditions 1 and 4 and then test on novel viewpoints to calculate SSIM (Structural Similarity Index). The test result is shown in Fig 1.



(Fig 1: rendering result, the leftmost is the ground truth with mask, the second is the prediction without mask, the third is the prediction of depth, the fourth is the prediction of the mask.)

As shown in Table 1, OI_V1 achieves the best result. However, OI_V2 leads to the worst result. One potential reasoning for this result is that the RGB MLP is used to both build the corresponding RGB and lighting condition alongside the prediction, which may affect the prediction performance. In OI_V3, the performance is based on the complexity of the scene. When the scene is complicated like the toy dog, the SSIM is low, while for a simple scene like a bucket, it's high.

Method	Egg(paper)/SSIM	Dog(fabric)/SSIM	Pumpkin(foam)/SSIM	Bucket(metal)/SSIM
V0	20.777938842773438	22.131134033203125	20.52488899230957	15.922685623168945
OI_V1	**20.819135665893555**	**22.180362701416016**	**20.538007736206055**	**16.50079345703125**
OI_V2	20.58094596862793	21.27266502380371	19.3559513092041	16.431364059448242
OI_V3	20.765817642211914	18.63877296447754	20.518596649169922	16.37470245361328

(Table 1: implementation of approaches V0 to V3 on 4 scenes under lighting conditions 1 and 4.)

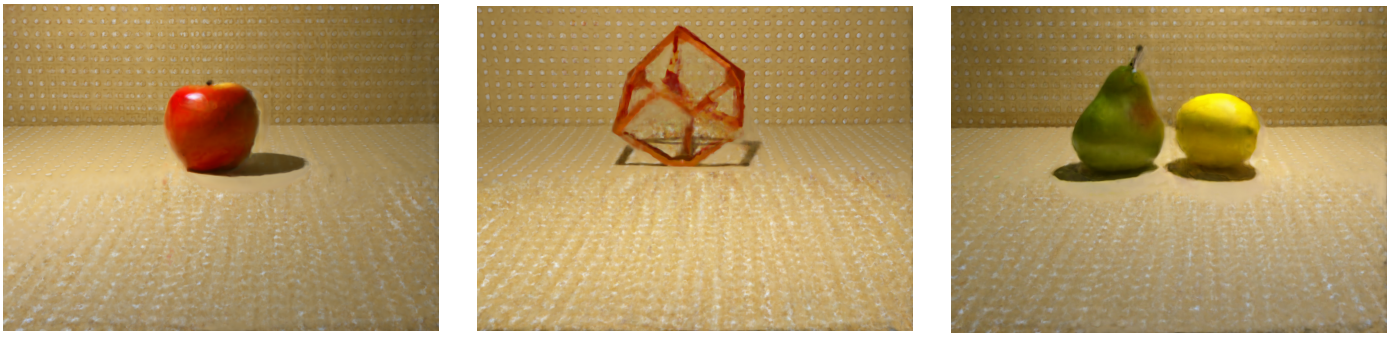
We made an ablation experiment on the position encoding of the input lighting condition. When the data is mapped to the same dimension as the input direction, the result is the best. Lower or higher dimensions decrease the performance.

	4-dimensional	16-dimensional	24-dimensional
SSIM of Bucket	15.894981384277344	16.50079345703125	15.326083183288574

(Table 2: SSIM of bucket scenes with different dimensions of lighting condition after position encoding.)

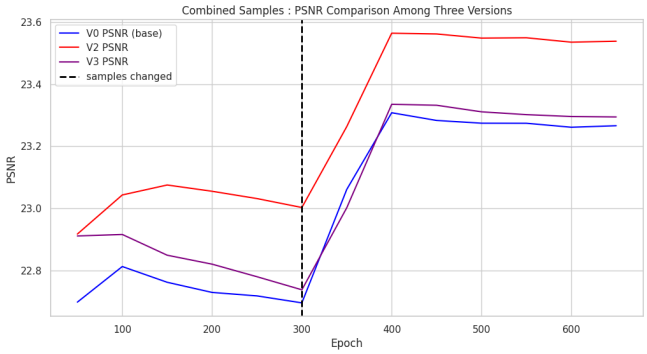
ReNe + Torch NGP

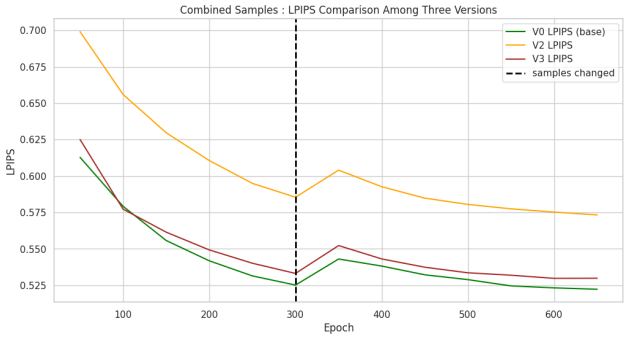
We plotted the evaluation metrics PSNR (peak signal to noise ratio) and LPIPS (Learned Perceptual Image Patch Similarity) for a generated image in Fig 4 every 50 epochs while training the network, as shown in Fig 2 and Fig 3.



(Fig 4: test result from left to right: ReNe V2 under one lighting; ReNe V3 under one lighting; ReNe V2 under 2 lighting.)

The PSNR and LPIPS plot clearly demonstrate the effectiveness and advantages of all three versions of the architecture, while also informing us of the limitations of each. The V2 architecture clearly shows a marked improvement in the PSNR while showing slightly worse LPIPS all around. The V3 architecture has a slightly better LPIPS metric than V2, indicating that positional encoding in the RGB network can have some advantages, however at the cost of PSNR, most probably due to overfitting. Both V2 and V3 architecture showed better PSNR at especially lower epochs during training validation, which validates the claim made by the ReNe paper authors that the knowledge about the location of the query point may vanish as the depth of the network increases. This may also be why LPIPS is better on V3 where the query point is fed into the Color MLP.





(Fig 5: left image: Combined samples PSNR ; right image: Combined samples LPIPS)

We also tried a combined approach where we tried to combine images of one object shot under different lighting conditions, while also accounting for the change in lighting direction. Here are the plots that demonstrate the results.

The models were basically trained partially on one set of images and then partially on another. This approach provides a better generalized PSNR, at the cost of the LPIPS metric. In this case, PSNR-wise V2 clearly performed the best, while the original model V0 still ended up with the least LPIPS as before.

Challenge/Innovation

It was clear from the start that our project was not straightforward to execute – we were to experiment and relight different datasets with various approaches. But this plan was not what we had in mind in our project proposal. Our initial idea was to train N models under N lighting conditions, but we realized that this was not reasonable. After some investigation, we found two newly published datasets, ReNe and OpenIllumination, so we adjusted our plan in implementing the relighting.

In addition to finding datasets with included lighting data, we also needed to find a NeRF model that complemented our project goals. Specifically, we tried using NerfStudio (https://docs.nerf.studio/) and NGP PL (https://github.com/kwea123/ngp_pl), but found both difficult to work with. Eventually we found Torch NGP and Instant NSR PL more suitable to implement for the project. Furthermore, it took us quite a bit to realize that both ReNe and OpenIllumination's datasets were in OpenCV format, but the NeRF model typically uses a Blender format. Thus, we transformed the datasets' format from OpenCV to Blender. This was done by writing a custom Python script that was able to convert between formats without using computationally expensive programs like COLMAP.

Moreover, while implementing the relighting architectures in Torch NGP,we suffered numerous CUDA memory issues, like memory access violations.This took a considerable amount of time to fix and could only be resolved by debugging the code one step at a time with incremental changes.

Lastly, it was difficult to find a machine on campus that had a GPU with sufficient compute power to not only successfully install all of the prerequisite libraries for NeRF, but to also train the NeRF and render a scene. When we were still trying to set up NerfStudio, we tried using EWS machines, AORUS GTX 1070 External GPU, Google Colab, Google Cloud, and other platforms. Essentially, we spent a lot of time figuring this out, but moved on by foregoing our usage of NerfStudio.

Group Member Contributions

Group members contributed similarly.

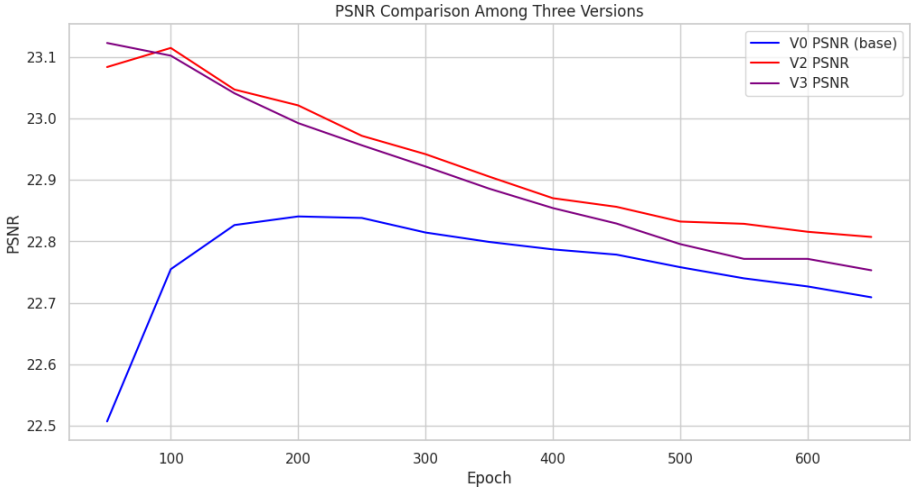
*Feiran Wang: [implemented OpenIllumination on Instant NGP, tested code on various machine and cloud servers to find suitable platform, and designed the OpenIllumination V1 method.]

*Shourya Ghoush: [implemented the ReNe dataset on Torch NGP and designed/implemented the ReNe V2, ReNe V3 method on pytorch.],plotted data for ReNe date evaluation metrics and results.

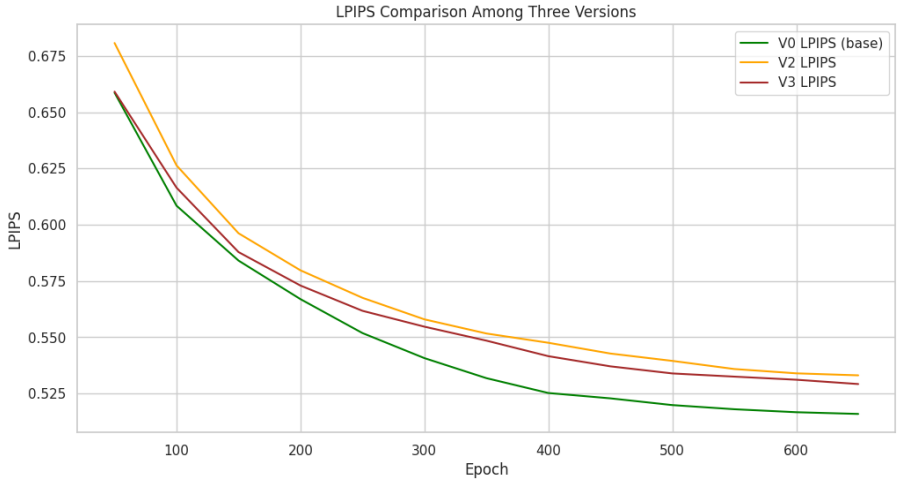
*Athena Zheng: [designed the OpenIllumination V2, V3 methods, wrote scripts to transform the format of dataset to implement on the NeRF model.]

Appendix

Images



(Fig 3, PSNR Plots of All three methods on ReNe Dataset, higher is better)



(Fig 4, LPIPS Plots of All three methods on ReNe Dataset, lower is better)

Video Links

V2 model rendering an apple: https://youtu.be/6COge0bGV4I

V3 model rendering a cube: https://youtu.be/-ShNo2hycnc

Combined V3 model rendering fruits: https://youtu.be/6sZTiprR6Ws