Assignment 5 Computation Creativity

Batch size of 3

November 2023

1 NOTE: RESULTS ARE INCLUDED IN THE BOX FOLDER

2 NeRF

2.1 Data Acquisition

We gathered three videos (shot by a phone) of different items in order to test the quality and effectiveness of the NeRF model that we will use.

Figure 1: Dataset

As you can see, we want to test different objects with different refractive indices, complex objects like a guitar, and somewhat the environment around them. The videos are then transcribed into images which takes 5 frames per second and transforms those frames into 5 images. Note: we used these dataset mostly for NeRF and for testing, we noticed later on that there were some significant issues with the videos taking the images. We added some additional videos to the dataset along with the fox from the instant-ngp dataset.

2.1.1 Colmaps

In order to transcribe the data for it to train with NeRFs, we need to generate a Colmap as input data of spacial locations in order to process them for NeRFs. Colmaps, also known as "Collaborative Mapping", utilizes SfM (Structure from Motion) algorithms in order to infer 3D information from multiple 2D images.

2.2 Parameter Choice/Tuning step

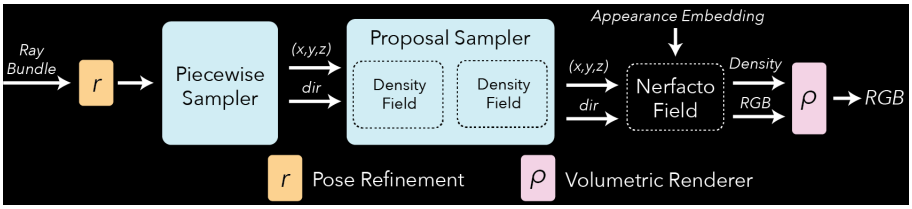
In our testing, we did not alter any of the tuning steps for training NeRF. Here is the truncated nerfacto configuration.

```
NerfactoModelConfig(
        collider_params={'near_plane': 2.0, 'far_plane': 6.0},
        loss_coefficients={'rgb_loss_coarse': 1.0, 'rgb_loss_fine': 1.0},
        eval_num_rays_per_chunk=32768,
        near_plane=0.05,
        far_plane=1000.0,
        hidden_dim=64,
        hidden_dim_color=64,
        hidden_dim_transient=64,
        num_levels=16,
        base_res=16,
        max_res=2048,
        log2_hashmap_size=19,
        features_per_level=2,
        num_proposal_samples_per_ray=(256, 96),
        num_nerf_samples_per_ray=48,
        num_proposal_iterations=2,
        proposal_initial_sampler='piecewise',
        interlevel_loss_mult=1.0,
        distortion_loss_mult=0.002,
        orientation_loss_mult=0.0001,
        pred_normal_loss_mult=0.001,
        implementation='tcnn',
        appearance_embed_dim=32)
```

2.3 Nerfstudio

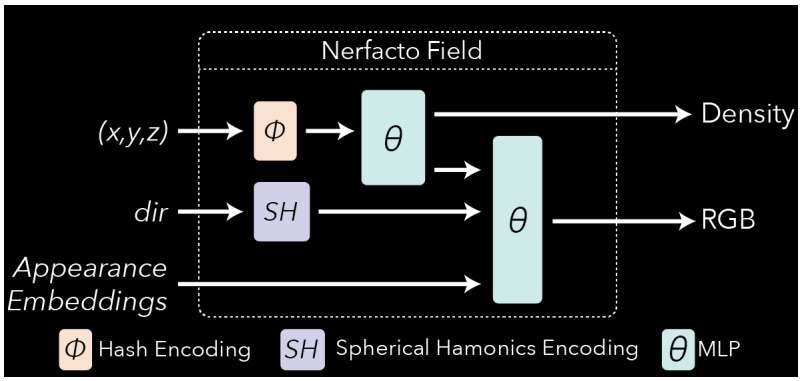
In order to train our data, we utilized nerfstudio, an API that provided the process of creating NeRFs, loading in NeRF models and support for other tools. The Nerfstudio developed a "defacto" method of NeRF that combines other methods of NeRF models that generally worked well. Unfortunately the only information of this model is defined on their documentation.

2.3.1 Nerfacto



Before the data is even sent to the neural radiance field, the pipeline performs some data preprocessing that assists the model. If we examine the dataset that is taken by humans, especially through mobile devices, notice that the movement of the camera is not always smooth and there will be artifacts in generalizing a field. This leads to problems when the NeRF generates artifacts around images and causing loss in detail, since it was unable to predict the correct camera pose. Therefore, in this step, when NeRF backpropagates for loss in order to find the input pose, we use that calculation to optimize the camera pose. This is a general and popular technique used for NeRFs known as "Camera pose refinement."

Afterwards, those inputs are sent to two different samplers. The first sampler, piece-wise sampler, generates two general samples of data: a uniformly distributed samples from the origin to the camera, and a larger step size of samples away. This allows for a more dense sample set for objects that are closer to the camera. Then a proposal sampler takes those samples and attempting to focus sample locations on more important areas of the scene. They is done by utilizing different density functions. Finally, the samples are sent to the neural radiance field to generate the density and color for the field.



Like NeRF, nerfacto takes spacial coordinates the the direction vector as an input and outputs the density and color. However, in this model, the spacial location is hashed instead of a positional encoding and we use a spherical harmonics encoding with an appearance embeddings to the MLP for color. The MLP is trained and rendered with the hash input encoding with the "tiny-cuda-nn" framework.

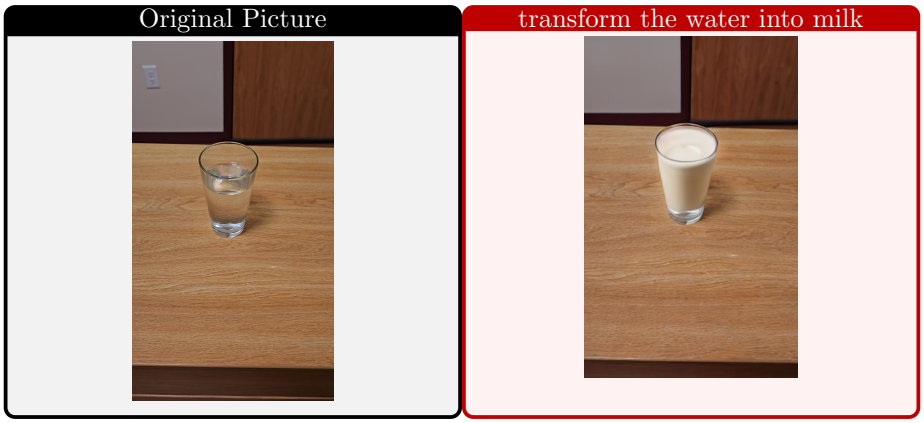
2.4 Reflection

There were many issues with dependencies for colmaps, and installing correctly. Through the Microsoft vkpkg repository that has colmaps, the build fails for windows and apparently only works through linux for conda-forge. Thus, if you use NeRF studio for instruct nerf2nerf or nerfacto, then consider not using Windows unless they fix the issue or bug (since it still is an open issue on that repository). If you look at the results for certain NeRFs, you might see that some parts are more noisy than others. This is due to the fact that we did not capture a full 360 image for that specific dataset. For instance, the chair had some issues where the classroom behind it was not captured. For the glass of water on the stool, most of the image is close up, so the surrounding image of it had not been captured nicely. Thus, when capturing data/images for NeRF, there are plenty of decisions behind trying to either capture the entire scene or capture the object of that image moreso. In our case, we tried mostly to capture the object, so the scene around that object may look a bit noisy.

For nerfstudio, using nerfacto took us 30,000 iterations in around 10 minutes to train the default model. Rendering the video with the camera paths took us around 5 minutes.

3 NeRF Transformations: InstructNeRF2NeRF

InstructNeRF2NeRF utilizes a diffusion model in order to edit NeRF scenes with given prompts. We were able to take a NeRF scene that was trained through nerfstudio and apply an extension to Nerfstudio that uses InstructNeRF2NeRF. This allowed us to experiment with changing the objects or items in the scene with a different item as well as try to change the environment around as well. For instance, if we have a cup of water, we want to use this framework to try and change that cup of water into a cup of milk.

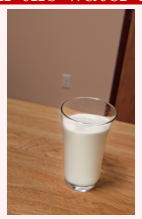


4 Discussion

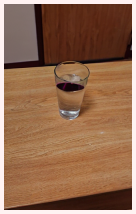
If we examine our results, we have varying levels of success with mostly failures for InstructNeRF2NeRF. For all of the NeRF examples, especially the glass water in the cup, we can see that the refraction of the water in the cup actually matches very well with the scene in the result. Moreover, in our other scenes, we can see that it generalizes the scene very well based on the data that is captured for it. Therefore, we can claim that our NeRF models were successful with nerfacto.

Through our research, InstructNeRF2NeRF utilizes InstructPix2Pix in order to generate images, and it is recommended that if our prompt works for InstructPix2Pix then it would work well for InstructNeRF2NeRF: (in this case we did not use the "glass of wine" prompt since many results did not turn out well for pix2pix).

transform the water into milk



turn the cup into a glass of wine



To easily differentiate between InstructPix2Pix and InstructNeRF2NeRF, a red box will be used for Pix2Pix results while a blue box will be used for NeRF2NeRF results.

There were certain prompts that did not have good results. For instance, animals that are closer to each other than others have higher similarities and in respect to that, looks much closer to each other. In the example below, note that the wolf looks very good, meanwhile the dog could need some work with that makeup.

turn the fox into a wolf

turn the fox into a dog

However, in general, most results for InstructNeRF2NeRF did not work well as intended. Although we are uncertain about the exact reasoning why, we might have several indications of why this might be the case. Previously, we mentioned that InstructNeRF2NeRF takes InstructPix2Pix over the NeRF scene. Thus, certain cases worked well and some didn't for InstructPix2Pix. However, since InstructPix2Pix is applied to different images over time, there can be large inconsistencies between the images generated between the datapoints. Thus, the result for the NeRF will attempt to go inbetween and result in some errors. For example, for one image it can generate orange juice perfectly, but for the other image it can generate orange juice, but add random oranges in the background (throwing everything off).

turn the water to orange juice

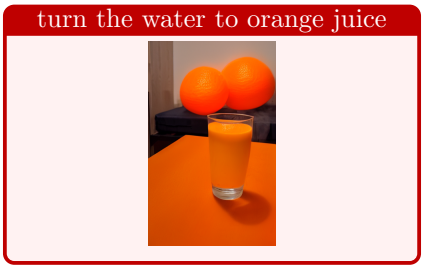
Moreover, there are certain cases where we have a working example for InstructPix2Pix, but the result for InstructNeRF2NeRF did not look as precise as the result from Pix2Pix.

turn the water to a red solo cup

turn the water to a red solo cup

At least for this example, the glass is a red cup. However, most of the room transformed to red. This can be a cause of the diffusive steps being too high or possibly too long in training.

In some instances we can see that there are certain errors with training, but early stopping can help with making a result look decent rather than terrible. In this case, we have a InstructPix2Pix example that failed. However, when we tried with InstructNeRF2NeRF it had similar results as the "red solo cup" example. That is, where the surroundings transformed into that color.

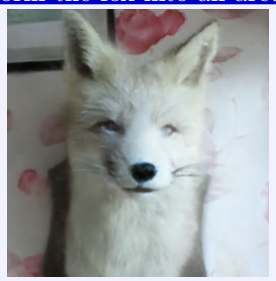


In addition to the orange juice example, we can see that this model does not work too well with an environment that has too many different and obtuse backgrounds (with different items). The color of the background eventually bleeds in with the color of the item.

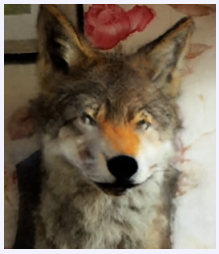
Here is an example of InstructNeRF2NeRF working very well:

Original NeRF

transform the fox into an arctic fox



transform the fox into a wolf



For some reason, the nose turned more orange than the fox's nose. Perhaps, all of the fox's color got drained into the nose when transforming into a wolf.

5 Resources Used

InstructNerf2Nerf: link
Nerfstudio: link
Instant-Ngp (for instant-ngp testing v. nerfacto): link
InstructPix2Pix Testing: link