Exploring Neural 3D Reconstruction: CS 7643

Ahmed Maher Ibrahim, Rohit Singh, Buhuang Liu, Juan Pablo Rivera
Georgia Institute of Technology
{aibrahim61, rohitsingh, bliu350, jrivera64}@gatech.edu

Abstract

Understanding the shape and appearance of a 3D scene given a few images is a long-standing problem in computer vision. Deep neural networks are showing great results in 3D reconstruction and novel view synthesis from a sparse set of input images, and are finding applications in 3D mapping, visualization, reality capture, and more. In this project, we explore deep learning techniques for 3D reconstruction and novel view synthesis, starting with neural voxel-based approaches, followed by Neural Radiance Fields (NeRFs). Our project investigates vanilla NeRF and and the more recent Generalizable NeRF Transformer(GNT). In this report, we provide a detailed description of our methodology, share the results of our experiments, and compare them with each other, as well as with those of the original papers.

1. Introduction/Background/Motivation

Inferring 3D scene geometry and appearance from sparse input images is a longstanding challenge in computer vision with applications in 3D mapping, photogrammetry, visualization, image editing, and robotics. Deep learning has significantly advanced 3D computer vision. In this project, we explore deep learning techniques for 3D reconstruction and novel view synthesis by re-implementing and comparing approaches such as single-view 3D reconstruction, Neural Radiance Fields (NeRFs) [4] for multi-view synthesis, and recent variants like the Generalizable NeRF Transformer [5].

Choy et. al.[2] and Xie[6] et. al. demonstrated the use of deep learning to take one or multiple input images and directly predict voxel occupancy grid with fixed orientation and scale. We followed Xie[6] et. al. to predict 3D binary voxel grid from single and multiple images. This approach works reasonably but requires supervision using image-3D pairs that are hard to obtain in real life. Our experiments are done using the ShapeNet [1] data set.

Voxel grids are an explicit 3D representation that directly represent the shape and appearance of scenes. The problem is that they grow exponentially in size with the scene size or resolution, and hence we explored the use of implicit 3D representations, such as a signed distance functions, that describe the surface of an object as an implicit function, without explicitly representing it's geometry. Training deep neural networks to map xyz coordinates to implicit representations like signed distance functions is thus possible [8, 9] but this requires access to accurate 3D geometry, usually obtained from synthetic 3D shape datasets such as ShapeNet [1].

Differentiable volumetric rendering enables learning implicit 3D representations without 3D ground truth supervision [10]. We implemented this technique, learning signed distance functions for simple shapes. This introduced us to Neural Radiance Fields (NeRFs), a compact neural network representation that synthesizes novel RGB views from a limited number of input images and their camera poses.

We also seek to eliminate differential volume rendering by using Generalizable NeRF Transformer (GNT) which leverages an encoder-decoder transformer scheme to implicitly learn scene representation which can generalize better to unseen scenes. We chose to implement the GNT model because of its use of transformers as a "general-purpose differentiable computer" [7] that also replaces volumetric rendering as a key foundational element of NeRFs.

We utilized ShapeNet [1] and the 3D-R2N2 dataset [2] from ShapeNet for single-view 3D shape reconstruction experiments. For NeRF experiments, we used the Realistic Synthetic 360°dataset, comprising eight scenes with eight objects rendered at 800x800 pixels, with 100 training and 200 testing views [4], and the LLFF dataset [3], containing 24 real-life scenes captured by handheld cellphone cameras with 20-30 images each, including the commonly used "lego" and "fern" scenes. To facilitate rapid prototyping on commodity GPUs, we employed down-sampled versions of these scenes provided by Facebook.

2. Approach

Our exploration of neural 3D reconstruction involved the implementation of several foundational deep learning milestones in the evolution of 3D deep learning for novel view synthesis. This approach allowed us to gain experience with implementing deep learning techniques for 3D computer vision.

2.1. Single/Multiple View to 3D

For single/multi-view to 3D reconstruction, we obtained starter code from Assignment 2 of CMU's Learning for 3D vision [4]course and modified it for this project's use.

2.1.1 Single view to 3D Methology

Single view to 3D takes image as input and outputs a voxel grid.

Voxel grid is view independent and rescaled to fit the entire object in 32*32*32 grid (scale not perserved but aspect ratio preseved).

Our model takes image as input and produce voxel grid directly.

We explored several losses such as binary cross entropy. The target is the ground truth voxel map(0 or 1). Model output is predicted voxel map probability.

For our work, we only trained and evaluated on "Chairs" in the R2N2 dataset which was adopted from ShapeNet dataset.

2.1.2 Network Architecture

Many architectures possible but we explored several in experiments. Our architecture is very similar to Pix2Vox with a encoder that encodes 2D image to embeddings and a Conv3D transpose decoder that generates voxels from this embedding.

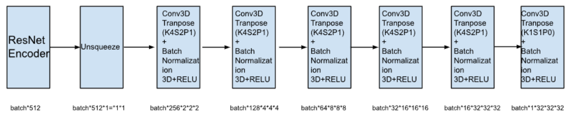
ResNet Encoder
Unsqueeze
Conv3D Tranpose (K4S2P1) + Batch Normalization 3D+RELU
Conv3D Tranpose (K4S2P1) + Batch Normalization 3D+RELU
Conv3D Tranpose (K4S2P1) + Batch Normalization 3D+RELU
Conv3D Tranpose (K4S2P1) + Batch Normalization 3D+RELU
Conv3D Tranpose (K4S2P1) + Batch Normalization 3D+RELU
Conv3D Tranpose (K1S1P0) + Batch Normalization 3D+RELU
batch*512
batch*512*1=*1*1
batch*256*2*2*2
batch*128*4*4*4
batch*64*8*8*8
batch*32*16*16*16
batch*16*32*32*32
batch*1*32*32*32

Figure 1: Network Architecture of Encoder-Decoder single image to 3D voxel

2.1.3 From Single image to Multi images

Since our output voxel is view independent (same direction with chair facing front), we can simply take multiple images of the same chair, predict voxel grids, then finally perform averaging to get final map. This ensembling boosts performance of prediction.

2.1.4 Novel View Synthesis

We synthesize new views from the voxel grid by running marching cube to generate meshes, then rendering the meshes in PyTorch3D with new camera position and lighting.

2.2. Volume Rendering and Neural Radiance Fields

To perform our exploration of NeRFs, we obtained starter code of Assignment 3: Volume Rendering and Neural Radiance Fields of the Learning for 3D vision [11] course and used it to implement ray marching, differentiable volume rendering, optimization of signed distance functions and develop the core components of the NeRF architecture.

Our implementation of NeRF modeled an implicit volume as a Multi-Layer Perceptron (MLP) that was used to map 3D position to volume density and color. We then used this implicit volume to optimize a scene from a set of RGB images. We also added view dependence to the NeRF, so that emission can vary with viewing direction, and tweaked its network architecture and hyperparameters in order to train it on higher resolution imagery.

2.2.1 Differentiable volume rendering

In the emission-absorption model, volumes are described by their emission (i.e. appearance) and absorption (i.e. density) at every point in 3D space. Differentiable renderers are a critical component of 3D deep learning as they allow us to optimize scene parameters using image supervision. We performed ray sampling by generating rays from a camera in 3D space and used a stratified sampling approach to generate sample points along each ray. We used equations 1 and 2 to render color along a ray.

$$L(x,\omega) = \sum_{i=1}^{N} T(x, x_{t_i})(1 - e^{-\sigma_{t_{i-1}}\Delta t_i}) L_e(x_{t_i,\omega}) \tag{1}$$

where σ is density, Δt is the length of the current ray segment and L_e is color. The transmittance represents the fraction of light that can pass through a volume and it can be computed using the following equation:

$$T(x, x_{t_i}) = T(x, x_{t_{i-1}}) e^{-(\sigma_{t_{i-1}}\Delta t_{i-1})} \tag{2}$$

The volume renderer evaluates the rendering function at each sample point along a ray and aggregates the results to render the scene. Since the volume renderer is differentiable, it can be used to optimize the parameters of a volume represented as a signed distance function or as neural network.

2.2.2 Optimizing a basic implicit volume

We used Mean Squared Error (MSE) between the predicted colors and the ground truth color value for each pixel in the image as the loss to optimize the parameters of signed distance functions, given a few ground truth images with known camera poses. Peak signal to noise ratio (PSNR) was used as the evaluation metric in our experiments.

2.2.3 Neural Radiance Fields (NeRF)

Neural Radiance Fields (NeRF) are an implicit, neural representation of a scene in the form of a multi-layer perceptron, that outputs a view-dependent color and density at each point (x, y, z) in space. This continuous function approximation of a scene can be rendered using the differentiable volume renderer that we implemented.

The NeRF model is trained using supervised learning to minimize the discrepancy between the predicted color and density along points on light rays, and the ground truth color obtained by rendering the training images using the volume rendering pipeline. The trained model can be used to render novel views of the scene by predicting the view-dependent color and density along the rays used for rendering the scene. This involves sampling points along each ray and computing the radiance field at each point using the NeRF. The final color of each pixel is obtained by integrating the radiance along the entire ray using volume rendering.

Although the NeRF model is computationally intensive, it can generate photo realistic images of a scene given a sparse set of input images.

2.3. Generalizable NeRF Transformer

To compare NeRF with the Generalizable NeRF Transformer (GNT), we re-implemented its deep learning components using only the papers as references. Our implementations might differ from the official ones based on the level of detail described. GNT improves the neural network's ability to generalize to unseen data with a two-stage transformer similar to an encoder-decoder structure. It leverages a view transformer, replacing the MLP, and a ray transformer instead of the hard-coded rendering equation in NeRF.

We re-implemented the forward pass and attention mechanisms for the view and ray transformers, as well as a simpler MLP feature extractor Unet. We believe this will help us understand both approaches' mechanisms and inspire ideas for module improvements. We also performed experiments by changing hyperparameters and network architectures.

GNT is a novel view synthesis approach based on NeRF that challenges the need for some analytical equations in the typical NeRF pipeline. It replaces the MLP with a View Transformer and the volume rendering equation with a Ray Transformer. Our group recreated the View transformer with its UNet feature extractor and the Ray transformer, maintaining the same supplemental code. We didn't recreate the rest of the code as it mainly involves boilerplate code for parsing datasets and isn't related to deep learning.

2.3.1 View Transformer

The view transformer is responsible for describing each sampled point on the sampled ray by providing a single feature vector per point. The first component of the View transformer is the feature extractor. This is a UNet feature extractor that provides a feature map for each image in the current scene. The second component is a transformer that takes as input a single feature vector from each input image. This feature vector is a single cell in the feature map provided by the UNet model. The coordinates of this cell are obtained by projecting the 3D coordinates of the sampled point on the ray to each input image. This can easily be done as we have the extrinsics and intrinsics of each camera. Given a feature vector from each image, the view transformer produces a query by pooling all the feature vectors and uses this query in its attention mechanism to produce a single feature vector that is passed to the ray transformer. Typically, attention mechanisms produce a feature vector for each input, but here we have a single query instead of treating all feature vectors as queries. Surprisingly,

2.3.2 Ray Transformer

The Ray Transformer focuses specifically on improving the classical rendering technique. Instead of relying on a hard-coded equation, the Ray Transformer learns what needs to be represented during the rendering portion of the equation. To render the color of a ray, the Ray Transformer computes a feature representation for each point sampled on the ray, incorporating position encoding of spatial location and view direction. The color is obtained by feeding these features into the Ray Transformer, performing mean pooling over all predicted tokens, and mapping the pooled feature vector to RGB using an MLP—notice that an MLP is still part of the solution.

Additionally, the Ray Transformer can automatically adjust attention distribution to control the sharpness of the reconstructed surface and capture desirable lighting effects from illumination and material features. By leveraging the expressiveness of the image encoder, it can simulate complex light transport (e.g., refraction, reflection) beyond the limitations of ray casting and epipolar geometry. Interestingly, the Ray Transformer can also infer explicit physical properties, such as depth, despite operating in latent space. These emergent properties of the Ray Transformer are incredibly interesting to see how well it can generalize to unseen data.

3. Experiments and Results

3.1. Single/Multi-views to 3D

3.1.1 Loss function comparison

We experimented with three loss functions: binary cross entropy loss, weighted binary cross entropy loss and focal loss. As far as we know, we are the first to apply focal loss to this problem.

To compare performance, we use the IOU metric where it is defined as:

$$IoU = \frac{\sum_{i,j,k} I(p(i,j,k)>t)I(gt(i,j,k))}{\sum_{i,j,k} I[I(p(i,j,k)>t)+I(gt(i,j,k))]}$$

where p(i,j,k) and gt(i,j,k) represent the predicted occupancy probability and the ground truth at (i, j, k), respectively. I(·) is an indicator function and t denotes a voxelization threshold. Higher IoU values indicate better reconstruction results. For t, we use a probability of 0.5 such that every grid with 0.5 and above positive is marked positive to compare between our models. It is possible to get better performance by tuning this t(for example using a higher threshold), which will not be discussed here.

Method	IOU
BCE	0.2408
BCE with positive example weighted 5x	0.4795
Focal Loss alpha=0.6, beta=1	0.2938

3.1.2 Network architecture comparison

We compare three architectures: (1) Encoder-Decoder architecture with conv2D blocks on image and conv3D transpose block generating voxel grid. See 1

(2) Only fully connected layers in the decoder. See 2

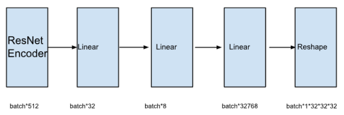

Figure 2: Network Architecture of Encoder-Decoder single image to 3D voxel

Here are the results compared:

Method	IOU
Linear	0.3317
BCE with positive example weighted 5x	0.4795

It is surprising that linear layers can achieve reasonable performance. One reason might be there are only a few templates for chairs that can give somewhat reasonable performance.

3.1.3 Analysis on success and failure cases

We present several success and failure cases to analyze network performance. While reconstruction looks very satisfactory, we do note complex details and uncommon chair shapes cannot be accurately estimated. Toilet was estimated to be more like a seat with round base and the last example showing uncommon chair cannot be easily estimated.

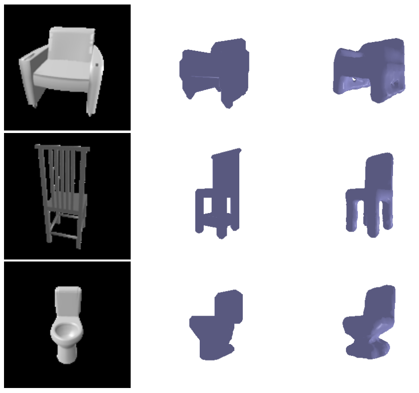

Figure 3: Left to Right: Input, GT, Prediction

3.1.4 Comparison with State of the Art

Our model is trained on a subset of R2N2(shapeNet). Specifically, we are only trained on chairs in R2N2. This greatly limits our models performance. Also, due to computation resource limitation, we only train for 10000 epoch with a smaller ResNet-18 backbone. We get decent results per 3d visualization but IOU cannot match state of the art. Though our IOU makes sense considering state of art is not significantly better.

Method	IOU
BCE with positive example weighted 5x	0.4795
3D-R2N2[2]	0.466
Pix2Vox[6]	0.567

3.1.5 Multi View 3D Reconstruction Results

Method	IOU
BCE with + example weighted 5x(Ours)	0.4938
3D-R2N2[2]	0.466
Pix2Vox[6]	0.567

3.2. Neural Radiance Fields

We followed the architecture of the original NeRF as described in the paper with a few modifications, shown in Figure 4.

While the original NeRF model used a combination of a coarse and fine model for improved rendering, we only implemented one model for simplicity. Also, we created a flexible version of the model where one can specify the number of layers to be used in the MLP, and where the skip connections are added. We made a slight variation to the

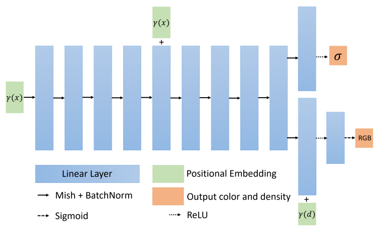

Figure 4: Network Architecture of our NeRF implementation.

network and added two heads to the network: one for predicting the volumetric density at any given point, and another to predict the color. The original NeRF paper used an additional output in one of the intermediate layers of the MLP to predict the density. We added view-dependent radiance to the model by feeding in the viewing direction as an input when predicting the color at any given point in the 3D volume. A high frequency positional embedding was added to the input ray positions and viewing directions, as this helps produce a sharper output.

We additionally used Batch Normalization layers that aren't used in the original NeRF implementation. This stabilized model training and also had a regularizing effect. In our experiments, the Mish activation gave better results than ReLU. To speed up model training further, we used the One Cycle learning rate schedule that quickly warms up the learning rate and then tapers it off gradually leading to better optimization. Upon hyperparameter tuning, the best results were obtained with a learning rate of 0.009, 192 points per ray, 9 layers of 256 hidden neurons and skip connection on the 5th layer.

3.2.1 Comparison with original implementation

The qualitative results of the trained NeRF on the lego scene are shown in Figure 5.

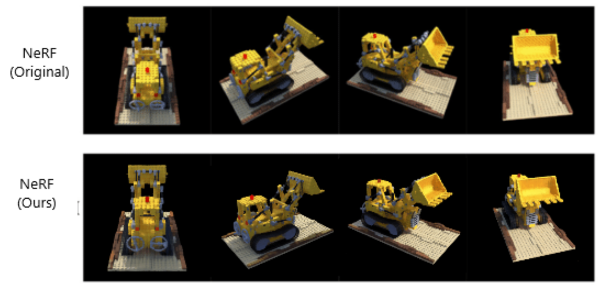

Figure 5: Qualitative results. NeRF (Original) above. NeRF (Ours) below.

Due to limitation of computation resources, we only trained our implementation of NeRF for 10000 iterations with a downsampled version of the Lego dataset. This greatly limits our performance. Additionally, our model architecture is a simplified version of NeRF as it does not use two (coarse and fine) networks but relies on one. Despite these limitations, we get comparable results qualitatively and the PSNR metric, while comparable, is still not as high as that of the original NeRF as shown in Table 1.

Model	Type	PSNR
NeRF (Original)	Full	32.51
NeRF (Original)	No view dependence	30.48
NeRF (Ours)	Full	28.05
NeRF (Ours)	No view dependence	25.07

Table 1: Comparison of NeRF model performance on Lego dataset.

3.3. Generalizable NeRF Transformer

We evaluate Vanilla NeRF and GNT quantitatively and qualitatively, training both on the Fern scene from LLFF dataset [3] and Lego scene from Blender Synthetic dataset [11]. Limited by computational resources, we restrict training to 10,000 back-propagation steps and two scenes, striving for a fair comparison.

For GNT experiments, we successfully re-implemented the view and ray transformers, testing on Fern and Lego datasets. Our implementation's performance is similar or slightly better than the original research team's. However, note that our results differ due to sampling 2048 rays for 10K iterations, compared to the original 4096 rays and 250K iterations. Training was conducted using personal computers and Google Colab. Further details are provided below.

3.4. GNT Experiments

We evaluate our Generalizable NeRF Transformer implementation against the official version [5] using PSNR, LPIPS, and SSIM (Table 1). Additionally, we optimize hyperparameters and experiment with an architectural change by replacing the UNet feature extractor with a shallow one, consisting of a single 1x1 convolutional layer and ReLU activation. Results are in Table 1. On NVIDIA RTX 2070, the official GNT runs at 0.476 seconds/step, our implementation at 1.635 seconds/step, and with the shallow feature extractor, it runs at 0.174 seconds/step.

The PSNR (pixel-signal to noise ratio) represents the original image's signal power compared to the noise generated during construction. LPIPS (learned perceptual image patch similarity) is designed for human visual perception, focusing on high-level features rather than pixel-to-pixel differences. SSIM (structural similarity index) compares images using structural information, luminance, and contrast. In this context, higher PSNR and SSIM values are desirable, along with a lower LPIPS score.

The table below compares different runs of code, evaluating performance on two datasets, Fern and Lego. Note that the run was limited to 10K iterations of calculating the loss function, due to computational constraints, rather than the full 250K iterations in the original GNT paper [5].

3.5. GNT model comparisons

In this comparison of different Generalizable NeRF Transformer (GNT) models, we evaluate their performance using three metrics: Average PSNR, Average LPIPS, and Average SSIM. Our proposed GNT model achieves an average PSNR of 22.8657 and 26.062979 on the Fern and Lego datasets, respectively, indicating a moderate level of reconstruction quality. The model also has an average LPIPS score of 0.2533 for Fern and 0.0766 for Lego, suggesting a noticeable perceptual difference from the ground truth images, particularly for the Fern dataset. However, the average SSIM scores of 0.739 and 0.92317 for Fern and Lego, respectively, demonstrate that our model maintains structural similarity and overall visual quality.

Comparing our GNT model with the original, we observe similar PSNR and SSIM performance, with slight improvements in the Fern dataset. The GNT model with a shallow feature extractor (SFE) performs comparably in PSNR and SSIM, achieving slightly better LPIPS scores on both datasets. This is surprising since the SFE is faster and requires less memory. We hypothesize that ResNet is useful only for large numbers of training iterations, which we couldn't verify. The SFE is beneficial for fast prototyping. Overall, our GNT model maintains competitive performance and produces high-quality 3D reconstructions.

Models	Dataset	Avg. PSNR	Avg. LPIPS	Avg. SSIM
GNT (Ours)	Fern	22.8657	0.2533	0.739
GNT (Ours)	Lego	26.062979	0.0766	0.92317
GNT (Original)	Fern	22.63	0.26	0.73
GNT (Original)	Lego	25.35	0.08	0.92
GNT (SFE)	Fern	22.56	0.23	0.74
GNT (SFE)	Lego	25.57	0.06	0.93

Table 2: Comparison of GNT model performance on Fern and Lego datasets. SFE stands for shallow feature extractor. Fern (128x128) , lego (128x128), n rays = 1024, n points = 128)

In the qualitative analysis of Fern images, we observe slight variation of quality across the ground truth, the vanilla GNT, and our own implementation of GNT. The ground truth image serves as the benchmark, showcasing the highest level of visual quality and accurate representation of the scene. On the other hand, the vanilla GNT produces images that are high quality, but exhibit a lower quality compared to the ground truth.

This may be due to limitations in the original GNT's ability to capture fine details or accurately represent complex lighting and occlusions. Lastly, our own GNT implementation demonstrates similar or slightly lower image quality among the three. This result could stem from sub-optimal hyper parameter choices or deficiencies in the model architecture, which could hinder the model's ability to reconstruct the 3D scene faithfully. However, it is important to note that, despite its limitations, our GNT implementation still manages to generate visually coherent images that bear resemblance to the original scene. Further improvements to our model may lead to better performance and higher quality reconstructions in the future.

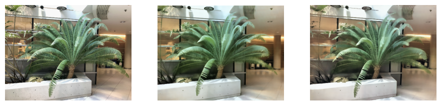

Figure 6: Left to Right: Ground Truth, GNT (original), GNT (ours)

Hyper parameter tuning for GNT: Hyperparameter tuning for GNT was conducted on the fern dataset for 2K iterations to maximize experiment quantity. The results reveal that a learning rate decay factor of 0.012 leads to lower loss and higher PSNR than the default setting, while 0.025 yields slightly worse performance, suggesting an optimal value between these two. A learning rate of 0.0005 for features results in lower loss and higher PSNR, indicating that a smaller rate might be more effective. Adjusting the GNT learning rate, both 0.00025 and 0.001 improve performance compared to the default, with the latter achieving the lowest loss and highest PSNR among tested configurations. Based on PSNR, the only improvement over the default implementation came from adjusting the learning rate for features, which might be due to the network benefiting from a faster feature extraction rate with a smaller structure.

Parameter	Value	Loss	PSNR
gnt llff default	Default	0.011814	19.275792
lrate decay factor	0.012	0.00747	21.266336
lrate decay factor	0.025	0.00793	21.006566
lrate decay steps	25000	0.008825	20.542382
lrate decay steps	80000	0.008416	20.74829
lrate features	0.0005	0.00745	21.277655
lrate features	0.005	0.042938	13.671466
lrate gnt	0.00025	0.009543	20.20261
lrate gnt	0.001	0.006976	21.563566

Table 3: Hyperparameter tuning results for the GNT model.

Student Name	Contributed Aspects	Details
Ahmed Maher Ibrahim	Research and Implementation	Researched the different methods for 3D reconstruction. Implemented the View transformer for GNT. Performed some experiments for GNT including the shallow feature extractor experiment.
Rohit Singh	Implementation and Analysis	Reimplement differentiable volume rendering, optimization of signed distance functions and the core components of the vanilla NeRF architecture. Hyperparameter tuning, and analysis of results.
Buhuang Liu	Implementation and Analysis	Investigated single view to 3D and researched vanilla NeRF; Trained single/multi view to 3D and performed various experiments, parameter tuning and analysis of results.
Juan-Pablo Rivera	Research and Implementation	Researched mechanisms of transfomers in 3D reconstruction. Re-implemented the attention and forward pass of ray transformer, performed hyper-parameter tuning, and analysis of results.

Table 4: Contributions of team members.

4. Work Division

The work division is explicitly stated with Table 3, but there were three areas we divided the project into. The first was a focus on the Single/Multi view to 3D reconstruction which was completed by Buhuang. Rohit worked on re-implementation of volumetric rendering and optimization of signed distance functions and vanilla NeRF, while Maher and Juan-Pablo focused on the transformer based architecture. Within the generalizable NeRF transformer, Juan-Pablo was in charge of re-implementing the forward pass, and attention of the ray transformer along with hyper parameter tuning. Maher was in charge of re-implementing the forward pass and attention mechanism for the view transformer, feed forward layer, and the UNet representation.The team equally contributed to writing of this report.

References

[1] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 1

[2] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. 1, 4

[3] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 2019. 1, 5

[4] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 2

[5] Mukund Varma T, Peihao Wang, Xuxi Chen, Tianlong Chen, Subhashini Venugopalan, and Zhangyang Wang. Is attention all that neRF needs? In *The Eleventh International Conference on Learning Representations*, 2023. 1, 5, 6

[6] Haozhe Xie, Hongxun Yao, Xiaoshuai Sun, Shangchen Zhou, and Shengping Zhang. Pix2vox: Context-aware 3d reconstruction from single and multi-view images. In *ICCV*, 2019. 1, 4

[7]Karpathy, A. [@karpathy]. (2022, October 20). The Transformer is a magnificient neural network architecture because it is a general-purpose differentiable computer...

[8] Jiang, C., Sud, A., Makadia, A., Huang, J., Nießner, M., Funkhouser, T.: Local implicit grid representations for 3d scenes. In: CVPR (2020)

[9] Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: DeepSDF: Learning continuous signed distance functions for shape representation. In: CVPR (2019)

[10] [Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision](https://arxiv.org/pdf/1912.07372)

[11] 16-825: Learning for 3D Vision https://learning3d.github.io/pages/assignments.html

[12] T, M. V., Wang, P., Chen, X., Chen, T., Venugopalan, S., Wang, Z. (2023). Is Attention All That NeRF Needs? The Eleventh International Conference on Learning Representations. Retrieved from https://openreview.net/forum?id=xE-LtsE-xx

[14] Lin, C., Ma, W., Torralba, A., Lucey, S. (2021). BARF: Bundle-Adjusting Neural Radiance Fields. 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 5721-5731.

[15] Yu, A., Ye, V., Tancik, M., Kanazawa, A. (2020). pixelNeRF: Neural Radiance Fields from One or Few Images. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 4576-4585.

[16] Jain, A., Tancik, M., Abbeel, P. (2021). Putting NeRF on a Diet: Semantically Consistent Few-Shot View Synthesis. 2021 IEEE/CVF International Conference on Computer Vision (ICCV), 5865-5874.

[17] Xiangli, Y., Xu, L., Pan, X., Zhao, N., Rao, A., Theobalt, C., Dai, B., Lin, D. (2021). BungeeNeRF: Progressive Neural Radiance Field for Extreme Multi-scale Scene Rendering. European Conference on Computer Vision.

[18] Lin, K., Lin, Y., Lai, W., Lin, T., Shih, Y., Ramamoorthi, R. (2022). Vision Transformer for NeRF-Based View Synthesis from a Single Input Image. 2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 806-815.

[19] Kulh'anek, J., Derner, E., Sattler, T., Babuvska, R. (2022). ViewFormer: NeRF-free Neural Rendering from Few Images Using Transformers. ArXiv, abs/2203.10157.

[20] Practical view synthesis with prescriptive sampling guidelines SIGGRAPH 2019. Local Light Field Fusion. (n.d.). Retrieved March 18, 2023, from https://bmild.github.io/llff/

[21] VITA-Group. (n.d.). Vita-Group/GNT: [ICLR 2023] "is attention all nerf needs?" by Mukund Varma T*, Peihao Wang* , Xuxi Chen, Tianlong Chen, Subhashini Venugopalan, Zhangyang Wang. GitHub. Retrieved April 30, 2023, from https://github.com/VITA-Group/GNT

[22] Nelson Max. Optical models for direct volume rendering. IEEE Transactions on Visualization and Computer Graphics (TVCG), 1995.

[23] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding," ACM Trans. Graph., vol. 41, no. 4, pp. 102:1-102:15, Jul. 2022. DOI: 10.1145/3528223.3530127.

[24] Learning a Predictable and Generative Vector Representation for Objects. Girdhar et. al., ECCV 2016

[25] 9. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3D shapenets: A deep representation for volumetric shapes. In: CVPR (2015)

[26] https://dl.fbaipublicfiles.com/pytorch3d$_nerf_data : PyTorch3dversionofNeRFdatasetsincludingsmallerversions of\lego"and\fern".$