A seminar report on DIVeR: Neural Radiance Fields with Deterministic Integration for Volume Rendering

Suraj Bhardwaj
Visual Computing Group
University of Siegen
Siegen, 57076
suraj.bhardwaj@student.uni-siegen.de

Margret Keuper
Visual Computing Group
University of Siegen
Siegen, 57076
margret.keuper@uni-siegen.de

Abstract

Many methods have been presented for the problem of developing a generative model capable of constructing a scene from multi-view 2D images with an adequate number of image samples and then generating an image from the created 3D models. To address this issue, generative models based on Monte Carlo Integrators fail to detect or represent translucent thin surfaces accurately. Generative Models such as NeRF and PlenOctrees can handle this issue by increasing the number of samples that strike the thin surfaces more than previously, but at the expense of linearly increasing computation, resulting in a new difficulty to tackle. DIVeR, with minimal adjustments, draws on the basis of NeRF [1], NSVF [2], and Continuous Volume rendering Integral to learning 3D scene representation and provides a realistic novel view image. DIVeR surpasses the NeRF model in terms of speed and accuracy by employing deterministic estimations of the volume rendering integral rather than stochastic estimates. From NeRF findings, the DIVeR64 model for offline rendering improves the PSNR score by 4.25%, the SSIM score [3] by 1.3%, and the LPIPS [4] measure by 60.49%. In terms of storage space and GPU resources, the DIVeR32 model for real-time rendering surpasses KiloNeRF [5], SNeRG [6], and PlenOctrees [7].

1 Introduction

Realistic novel view image generation using classical and modern computer vision techniques is an extensively researched topic in generative modeling, with applications in static and dynamic object rendering in 2D and 3D, cinematography [8], virtual reality [9], video stabilization [10], [11], appearance modeling, relighting, and computational photography. Recent breakthroughs in generative modeling, for example, NeRF [1] followed by PlenOctrees [7], FastNeRF [12] and KiloNeRF [5] have proposed unique solutions to the problem of developing a model capable of rendering a scene from multi-view 2D images with a reasonable number of image samples and then generating an image from the rendered 3D model using the ray casting technique with real-time rendering feature. This report is based on DIVeR [13], a NeRF [1] -based generative model that can generate an image from a learned 3-dimensional scene representation in real-time. Because 3-dimensional scene representation is learned using a fully connected neural network, this method can be classified as the 3-dimensional Neural Rendering [14] method.

In 1991, [15] described the complete holographic representation of the visual world and proposed the concept of The Plenoptic function. It generally refers to what a camera can see along a specific ray based on the parameters of the function. According to [15], if we could capture this function, we would be able to create any conceivable view, at any moment, from any point. Since then many researchers have worked on it and used this function to solve the problem under consideration in

Computer Vision Seminar on Recent Advances in Generative Models.

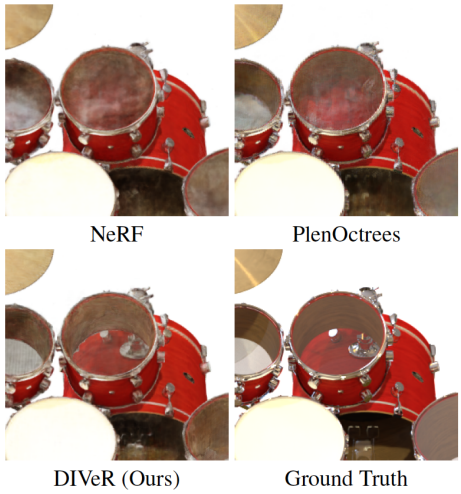

Figure 1: This figure displays the Qualitative rendering outcomes of models in relation to Ground Truth. It demonstrates that the DIVeR can render thin translucent surfaces such as drumheads better than other models such as NeRF and PlenOctrees [13].

this report. However, the disadvantage of Plenoptic function models is that they are difficult to smooth [13], which was addressed with the NeRF [1] approach, which is a good smoother. However, there are numerous drawbacks to the NeRF [1] model too. Using a model based on Monte Carlo integrator [16] to render images is one of the reasons that cause problems associated with NeRF [1]. Models based on Monte Carlo Integrators or stochastic integrators fail to render translucent thin surfaces with a practical number of image samples from a specific scene when photo-realistic image generation from a 3-dimensional scene representation using ray casting is taken into account. This problem can be addressed by increasing the number of samples in models based on stochastic integrators [16] but at the cost of computation that grows linearly, resulting in a new problem to solve [13]. That is why a detailed study of these limitations and a novel approach or improvement were much needed. DIVeR [13] highlighted the problem associated with stochastic integrator and proposed a novel integrator called Deterministic integrator for the integration of classical volume rendering equation. Along with minor tweaks in the sampling process, as described in the method section, than that of NeRF [1] model, DIVeR outperformed the NeRF model and other state-of-the-art methods in terms of speed and accuracy.

2 Background

This section focuses on the mathematical background that serves as the foundation for today's 3D Neural Rendering field. This section includes the plenoptic function formulation, the volume rendering integral, and the loss for optimization used in NeRF.

Plenoptic function Edward H. Adelson and James R. Bergen [15] described the plenoptic function as a seven-dimensional function with parameters: 3-dimensional ray position, view direction, wavelength, and time. The Plenoptic function is shown in equation 1 below:

$$\mathrm{P}(\mathrm{x},\mathrm{y},\mathrm{z},\theta,\phi,\lambda,\mathrm{t}) \tag{1}$$

where (x,y,z) are the co-ordinates of the ray position in 3-dimensional Cartesian co-ordinate system, (θ,ϕ) are the angles which describes the view direction d, λ is the wavelength, and t is the time. Each parameter of the Plenoptic function represents a physical aspect of a vision system and, when combined, can solve a wide range of static and dynamic vision problems in computer graphics and computer vision. The time parameter is crucial in real-time applications.

NeRF The input to the MLP used in [1] is the location of sampling points along the ray represented by 3D co-ordinates (x,y,z) and known viewing direction described by (θ,ϕ) and its output is RGB

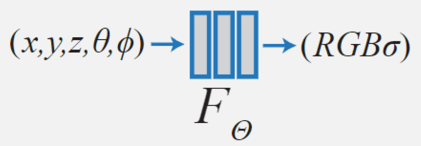

Figure 2: This figure shows the architecture of NeRF. The input to a fully connected Neural Network is a 5D Plenoptic function and the outputs are Radiance and volume density [1].

color c and volume density σ. We can compare this input with equation 1 and can formulate this as 5D Plenoptic input function as shown in equation 2.

$$\mathrm{P(x,y,z,}\theta,\phi) \tag{2}$$

The detailed working of the NeRF pipeline is described in detail in [1]. The initial stage in NeRF is to sample the 3D coordinates of a scene using the multiview 2D photographs of the scene. After sampling, the data is sent into a neural network, which generates radiance and volume density values. The amount of light traveling through the sample point in 3D space in the direction d under consideration is measured as radiance. The radiance field is constructed by collecting all such radiance values for all sampling locations in 3D space. If we know the radiance of every point in a scene in every direction, we have all of the visual information we need for the scene. The scene is represented by the weights of the Multi-Layer Perceptron (MLP). These weights are scene-specific and so cannot be utilized in another scene. As a result, for N-scene scenarios, we must overfit N Multi-Layer Perceptrons. The third phase is the reconstruction stage, which involves sampling field quantities for the novel view image from the 3D Scene represented by MLP weights. Volume density and radiance values are referred to as field quantities here. In the fourth stage, the reconstruction is then mapped back to the novel view 2D RGB picture. In the fifth stage, The MLP is then optimized depending on volume rendering loss.

Equation 3 represents the volume rendering integral [17] for the accumulation of radiance and generation of an image using NeRF.

$$\hat{\mathbf{c}}(\mathbf{r}) = \int_0^{\infty} e^{-\int_0^t \sigma(\mathbf{r}(\tau))d\tau} \sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t),\mathbf{d})dt \tag{3}$$

In equation 3, $\sigma(\mathbf{r}(t))$ represents the volume density of the ray **r**(t), which describes the presence of an object in 3D scene representation and $\mathbf{c}(\mathbf{r}(t),\mathbf{d})$ represents the color along the ray **r**(t) and direction **d**. However, the solution to this continuous function does not exist in the continuous domain and hence NeRF uses the Monte Carlo integration method to estimate equation 3. The key thing to notice in the case of a Monte Carlo-based integrator is that within the sampling intervals, Monte Carlo-based integrator assumes the value of radiance and density function as constant. Equation 4 approximates the continuous volume rendering equation and is shown below:

$$\hat{\mathbf{c}}(\mathbf{r}) = \sum_{i=1}^{n} \prod_{j=1}^{i-1} (1-\alpha_\mathrm{j})\alpha_\mathrm{i}\mathbf{c}_\mathrm{i} \tag{4}$$

$$\alpha_\mathrm{i} = 1 - e^{-\sigma_\mathrm{i}\delta_\mathrm{i}} \tag{5}$$

In equation 5, α_i indicates the collected alpha values along the period and its length.

The rendering loss used in the optimization step in [1] is shown by equation 6 and is defined as a squared error between the mean of radiance along a ray and the ground truth value of radiance at that pixel.

$$L = \sum_{k} \|\hat{\mathbf{c}}(\mathbf{r}_\mathrm{k}) - \hat{\mathbf{c}_\mathrm{gt}}(\mathbf{r}_\mathrm{k})\|_2^2 \tag{6}$$

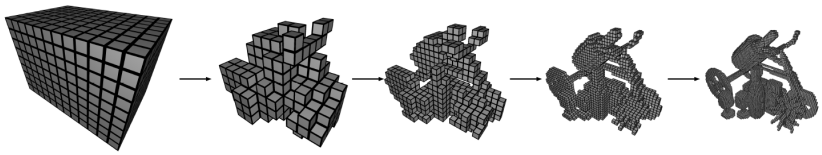

Figure 3: Illustration of self-pruning and progressive training [2].

NSVF It is an abbreviation for Neural Sparse Voxel Fields. It is a NeRF method extension. It tackles two major NeRF issues: addressing the problem of high computation cost and blurry rendering quality. To prevent sampling in empty space, NSVF employs a sparse voxel data structure, which also allows for realistic modeling of local scene features. NSVF is composed of a group of voxel-bounded implicit fields structured in a sparse voxel octree, rather than representing the entire scene as a single implicit field [2].

Implicit Fields in NSVF The initial assumption in NSVF approach is that non-empty elements of a scene are significant and are confined inside a set of sparse voxels. The scene is represented as a collection of voxel-bounded implicit functions. Unlike NeRF [1], the input to the MLP in this case is a representation of the spatial position and ray direction. And the output is color and density at that 3D point location. The spatial position of a single relevant voxel representing a portion of a scene is represented by four components. The first element is eight vertices of the voxel, the second element is feature vectors saved at each vertex, the third element is a trilinear interpolation, and the fourth element is the positional embedding post-processing function. Features in NSVF are aggregated from eight voxel embeddings, which contain details such as material, geometry, and color specific to each region. It considerably simplifies further MLP learning and promotes high-quality rendering [2].

Ray-voxel Intersection in NSVF It is performed using the Axis Aligned Bounding Box intersection test [2]. The number of sparse voxels depicting the scene, on the other hand, maybe rather huge. As a result, speeding up this intersection test becomes increasingly critical. The author employed a hierarchical octree structure to overcome this problem. The basic notion underlying the speed is that if a ray intersection occurs with a voxel, it must also occur with its bounding volume. Rather than completing the intersection check for every single voxel within a bounding volume, it is preferable to first determine whether or not the ray intersects with the specific bounding volume. Bounding volume refers to the large voxel that contains tiny voxels. So we can skip a lot of intersection tests by checking the root bounding volumes of the octree first.

NSVF Inference and Training The initial step for each ray is to determine the voxels that intersect it. The second step is to sample each voxel at a specified step size. The color is then added along the ray in the third stage. The voxel is partitioned into segments for this integration, and the Midpoint rule is utilized to perform a piecewise constant approximation for each segment sampled within a voxel. Using MLP, the density and color at these Midpoints are obtained.

The MLP receives the interpolated embeddings from the voxel vertices as input. The alpha value for the segment has been defined using the MLP's output volume density and the segment length. The greater the length and volume density of the segment, the greater the alpha value for the segment under consideration. The value of alpha indicates how much light is blocked. Transmittance T represents the ratio of light that has reached this section. As a result, the segment's light consumption ratio is the product of the segment's alpha and the current Transmittance. This ratio is also the weight of the color in the approximation integral. Finally, update transmittance based on how much light the section absorbed. If T falls below a certain level, then end the integration early. If it is not below the threshold, the operation will be repeated for the next segment. If we run out of voxels but still have some transmittance, the remainder will be spent on a constant background term. This will model the background of the whole scene. This is a poor modeling assumption and, as a result, a shortcoming of this strategy.

At the beginning of the training, the region of interest is unknown. Assuming that everything is occupied, we may begin with a dense voxel field. We occasionally prune voxels if all of their vertices fall below a certain alpha threshold. However, with high resolution, there may be an overflow of

voxels, to begin with. To address this issue in [2], the author began by pruning large voxels and then gradually subdividing them into smaller voxels, repeating these two stages for 3 to 4 rounds of voxel subdivision as shown in figure 3.

In contrast to NeRF, calculation in NSVF is dependent on the number of samples, which in turn is dependent on the number of intersected voxels. Although NSVF is substantially quicker than NeRF, it is still not real-time, which is another disadvantage of the NSVF technique.

3 Method

The novelty in the method of DIVeR [13] compared to NeRF [1] can be presented under two subheadings (i) Scene representation and (ii) Deterministic integrator as described below. In DIVeR the scene is represented by sparse voxel fields similar to NSVF [2]. Figure 4 depicts the ray rendering process in a detailed manner. Features at the eight vertices of the voxel represent a trilinear function. This trilinear function is integrated from the ray's intersection at the entry to the exit, and the result is passed to an MLP, which decodes the voxel color and alpha values. The final integral estimate for the ray is obtained by accumulating color and alpha values along that particular ray. The main distinction between this integration method and the Monte Carlo integrator is that the Monte Carlo method fills an interval with a constant, whereas, feature integration fills the interval with trilinear functions. And by using trilinear interpolation we can estimate the density and color values between the interval which is very helpful in the case of detailed modeling of geometry and materials.

3.1 Scene Representation

DIVeR represents the scene fields as a sparse voxel grid of feature vectors, similar to [2]. The scene field representation evolved from the early stages of development with a research question of how to represent a scene in a data structure that is multi-view consistent and how to sample at exact locations where there is a value of volume density. NeRF [1] proposed a hierarchical sampling approach to solve this problem. Hierarchical sampling is a two-step sampling process. In the first step, coarse sampling is done at uniformly distant sampling points along the shot ray which passes through an object in the 3D space. A fully connected neural network processes these uniformly distant coarse points and outputs volume density and color at those coarse locations. The volume density versus distance of the shooted ray plotted clearly shows a density peak for an object in 3D space between two coarse points along that particular ray resulting in a clear indication of the location where an object exists. This location serves as a guide for the second sampling to get more information about the 3D volume at that location. The drawback of NeRF's approach is that it wastes a lot of computation time by sampling in vacant space where the volume density is very close to zero, for example, the air in the 3D space. NSVF [2], a NeRF extension, solved this computation cost problem by proposing sparse voxel fields in which voxels can remember non-vacant spaces. This sparse representation is obtained by performing a ray-voxel intersection test and employing an octree to accelerate the search for non-vacant voxels.

3.2 Deterministic Integrator

The main difference between DIVeR and NeRF models is the use of NSVF [2] to represent the fields and a decoder MLP for the integrator. In DIVeR, the feature vectors are placed at the vertices of the voxel grid as shown in Figure 4. The piece-wise trilinear feature function is used for interpolation to find feature values within a specific voxel. The sparse voxels representing the scene will intersect the ray shot in 3D space from specific viewing angles to obtain volume density and color along that ray. Each ray-voxel intersection represents an interval for estimating partial integrals. This estimate is deterministic, which is why the authors of DIVeR [13] call it Deterministic integration. These voxel estimates are added together to render the entire scene. In DIVeR an implicit MLP with parameters w_{r} is trained to initialize voxel features which are optimized explicitly skipping the regularization MLP. Two types of decoder architectures are used in the DIVeR named DIVeR64 for offline rendering and DIVeR32 for real-time rendering applications. Both decoder architectures take integrated features and positional encoded viewing direction as inputs and output corresponding to integrated density and color. DIVeR64 has 8K parameters whereas DIVeR32 has 4k parameters.

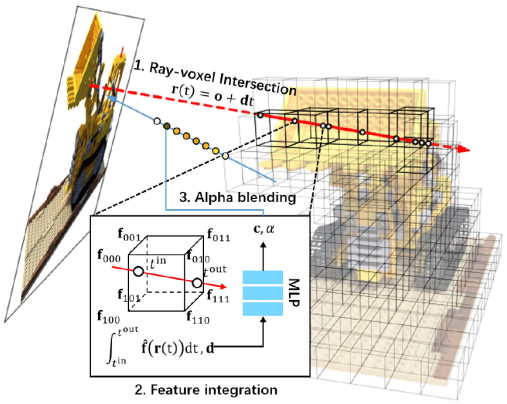

Figure 4: Rendering pipeline overview of DIVeR [13]. Rendering a ray begins by locating its intersection with voxels. Features at eight vertices of each intersected voxel represent a trilinear function. Integration of this trilinear function from the ray's intersection at the entrance to the exit is input to decoder MLP, which decodes the voxel's color and alpha values. The final integral estimate for the ray is obtained by accumulating color and alpha values along the ray.

Feature Integration The proposed DIVeR integrator is unique in its approach to feature integration. When a ray contacts the sparse voxel entry and exit during the ray tracing procedure, it produces an interval for the partial integration process. All produced intervals following the intersection of the ray with voxels are treated individually in DIVeR. The feature vectors at the vertices of each intersected voxel contribute to the partial integration process.

The volume density and radiance values for a certain interval are derived by sending the normalized integral of feature value within the voxel at position x along the interval to MLP, as shown by equations 7 below:

$$(\sigma_{\mathrm{i}}, \mathbf{c}_{\mathrm{i}}) = \mathrm{MLP}_{\mathrm{w}} \left(\int_{t_i^{in}}^{t_i^{out}} \hat{\mathbf{f}}(\mathbf{r}(t)) dt, \mathbf{d} \right) \tag{7}$$

in which the expression for the normalized integral of the feature value is shown by equation 8.

$$\int_{t_i^{in}}^{t_i^{out}} \hat{\mathbf{f}}(\mathbf{r}(t)) dt = \int_{t_i^{in}}^{t_i^{out}} \sum_{k=1}^{8} \mathbf{f}_k^i \frac{\chi_k(\mathbf{r}(t))}{t_i^{out} - t_i^{in}} dt = \sum_{k=1}^{8} \mathbf{f}_k^i \int_{t_i^{in}}^{t_i^{out}} \frac{\chi_k(\mathbf{r}(t))}{t_i^{out} - t_i^{in}} dt \tag{8}$$

The subscript w in equation 7 indicates the learnable weights of the MLP and $\chi(r(t))$ in equation 8 represents the weights of trilinear interpolation. More details about feature integration technique can be found under the heading of additional implementation details of [13]. The view direction d in equation 7 models the view-dependent effects present in the scene.

$$\hat{\mathbf{c}}(\mathbf{r}) = \sum_{i=1}^{n} \left(\prod_{j=1}^{i-1} (1 - \alpha_{\mathrm{j}}) \alpha_{\mathrm{i}} \mathbf{c}_{\mathrm{i}} \right) \tag{9}$$

Equation 9 includes the variable α_i, for which the expression is provided in equation 10.

$$\alpha_{\mathrm{i}} = 1 - e^{-\sigma_{\mathrm{i}}} \tag{10}$$

The expression for the variable σ_i and $\mathbf{c}_i$, as used in equation 10 and 9 respectively, is provided in equation 11.

$$\sigma_i = \int_{t_i^{in}}^{t_i^{out}} \sigma(\mathbf{r}(t))dt, \quad \mathbf{c}_i = \int_{t_i^{in}}^{t_i^{out}} \mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \tag{11}$$

Equation 9, 10, 11 denotes the accumulation of the partial integrals of all intervals into a final radiance value.

Regularisation Loss For sparse voxel representation, regularization loss [6] shown in equation 12 below is used in which λ_s denotes the regularization weight and σ_i denotes i^{th} accumulated density. It prevents DIVeR from predicting background color in empty spaces or voxels.

$$L_{sparsity} = \lambda_s \left(\sum_i \log(1 + \frac{{\sigma_\mathrm{i}}^2}{0.5}) \right) \tag{12}$$

Training For each scene, feature vectors, implicit multi-layer perceptron weights, and multi-layer perceptron weights are optimized. In order to render the color during training, a batchwise feature integration is employed. Gradient descent using rendering loss has then been used to generate the model. To speed up the training culling approach mentioned in Inference time optimization is utilized, these techniques determine whether particular regions are empty or not in the early phases of training. Then, high-resolution image training is conducted, and effectively unoccupied regions based on the coarse occupancy map are bypassed. The MLP weights and features learned on the coarse images are discarded.

Inference time optimization The sparse voxel culling approach, as utilized in [7], was employed to create a meaningful representation of the 3D scene. The 98% voxels are deleted by using a threshold on the maximum value of blended alphas. This culling occurs for both coarse and fine training. The authors solved the problem of occluded voxels due to a particular camera pose by imposing a minimum transmittance threshold of 0.001 to the transmittance value. In addition, a volume density threshold employing alpha value is used to save computation time by disregarding density values below a specified threshold.

4 Important Results

4.1 Experimental setup

The network optimization is done on PyTorch and the speed of training is enhanced using customized CUDA kernels. NeRF-synthetic dataset [1], a subset of the Tanks and Temples dataset [18], and the BlendedMVS [19] are used for experimentation. The explicit grid initialized from the implicit model is trained and optimized on feature vectors for three days.

4.2 Implementation details

The voxel grid size employed for implicit and explicit models with both datasets NeRFsynthetic [1] and BlendedMVS [19] is (256 X 256 X 256) respectively. Whereas the same is (320 X 320 X 320) for Tanks and Temples [18] for high-resolution image training. For the coarse model training, this scale is proportional to 1/4 of the fine model scale. The NSVF [2] approach with a batch size of 1024 pixels for coarse training, 6144 for fine training of Tanks and Temples, and 8192 for fine training of other datasets are used to sample rays from the training set [13]. The coarse model is explicitly trained for 5 epochs to determine the empty regions. Then the implicit MLP model is trained until the convergence of validation loss. The maximum GPU memory utilization in this process existed at roughly 40GB. The Adam [20] optimizer with learning rates of 5e-4 for the fine model and 1e-3 for the coarse model is used. The hyper-parameter regularisation weight λ_s in sparsity regularization loss [6] is 1e-5.

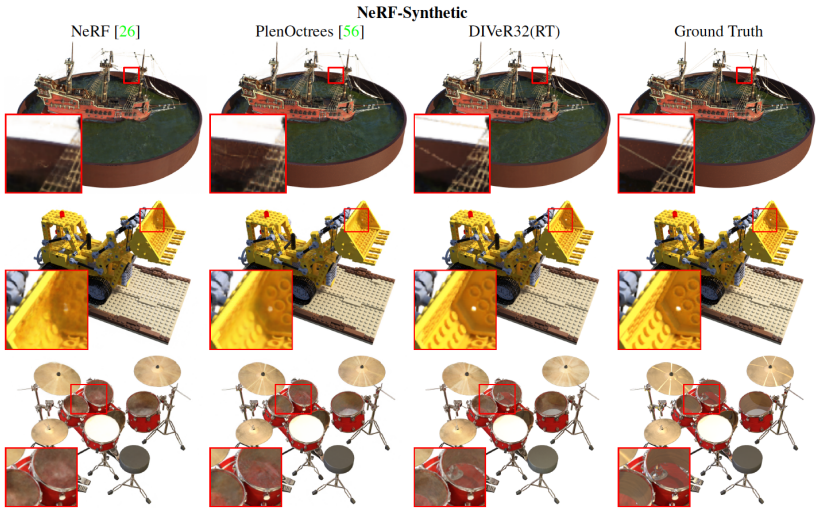

Figure 5: DIVeR qualitative results on a NeRF-synthetic dataset. The DIVeR can model all ground truth scenes better than NeRF and PlenOctrees, as demonstrated here [13].

The real-time application is written in Python and uses CUDA for parallel computing of all operations per picture pixel. The MLP output is blended into the image buffer. This method ends when the ray termination requirements mentioned in section 3 are met. During inference, feature vectors are stored in a 1D array, while the indices of the individual feature value are stored in a dense 3D array. As a result, GPU memory consumption is reduced while performance is maintained.

4.3 Results

This section includes the quantitative results of the DIVeR method for both offline and real-time rendering. Qualitative results of DIVeR on NeRF-synthetic [1] can be seen in figures 1 and 5 taken from [13]. The scope of this report is limited to the results presented in this report only. To get an insight into all qualitative and quantitative results one can refer [13]. Table 1 shows the quantitative findings on several benchmarks. Table 2 compares image quality to real-time pre-trained models using NeRF-synthetic. Table 3 shows the comparisons to various real-time variants based on NeRFsynthetic.

Evaluation Metrics To quantify the image perceptual quality of our generative model DIVeR, the Learned Perceptual Image Patch Similarity (LPIPS) metric has been used [4]. A lower number suggests a better similarity between Image patches. Peak Signal-to-Noise Ratio (PSNR) and Structural Resemblance (SSIM) [3] metrics are used to quantify image Fidelity, which quantifies the similarity between actual and created images. The maximum possible pixel value of the image and the mean squared error between images define the PSNR between the ground-truth image and the reconstructed one. SSIM assesses the structural similarity of pictures based on independent comparisons in terms of brightness, contrast, and structures. A higher number suggests better image fidelity.

5 Analysis and Discussions

Along with the rendering assessment criterion, Table 3 includes a criterion for real-time rendering (FPS), storage space (MB), and computation cost (GPU GB). The greater the FPS value, the better the result. In contrast, the lower the storage space and GPU resource value, the better for computation and storage. Table 3 demonstrates that the DIVeR32 architecture still has to be improved to outperform the SNeRG method's real-time rendering FPS score. However, by employing a basic architecture and explicitly addressing the storage and computation issues in the technique [13], the DIVeR32 model outperforms other methods in terms of storage space and GPU resources.

	Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF-Synthetic	NeRF	31.00	0.947	0.081
	JaxNeRF	31.65	0.952	0.051
	AutoInt	25.55	0.911	0.170
	NSVF	*31.74*	*0.953*	*0.047*
	DIVeR64	**32.32**	**0.960**	**0.032**
BlendedMVS	NeRF	24.15	0.828	0.192
	JaxNeRF	-	-	-
	AutoInt	-	-	-
	NSVF	*26.90*	*0.898*	*0.113*
	DIVeR64	**27.25**	**0.910**	**0.073**
Tanks and Temples	NeRF	25.78	0.864	0.198
	JaxNeRF	27.94	0.904	0.168
	AutoInt	-	-	-
	NSVF	**28.40**	*0.900*	*0.153*
	DIVeR64	*28.18*	**0.912**	**0.116**

Table 1: Quantitative results on different benchmarks [13]. The dash character '-' indicates that there are no publicly available results. The top results are provided in bold, while the second-best results are presented in italic.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF-SH	31.57	0.952	0.063
JaxNeRF+	**33.00**	**0.962**	*0.038*
NeRF	31.00	0.947	0.081
DIVeR32	*32.16*	*0.958*	**0.032**

Table 2: Image quality comparisons to real-time pre-trained models [13].

Method	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↑	MB ↓	GPU GB ↓
PlenOctrees	*31.71*	**0.958**	0.053	$\mathit{76 \pm 66}$	1930	1.65 ± 1.09
SNeRG	30.38	*0.950*	0.050	$\mathbf{98 \pm 37}$	*84*	1.73 ± 1.48
FastNeRF	29.97	0.941	0.053	-	-	-
KiloNeRF	31.00	*0.950*	**0.030**	28 ± 12	161	$\mathit{1.68 \pm 0.27}$
DIVeR32(RT)	**32.12**	**0.958**	*0.033*	47 ± 20	**68**	$\mathbf{1.07 \pm 0.06}$

Table 3: Comparisons to other real-time variants [13]. The dash character '-' indicates that there are no publicly available results. The top results are provided in bold, while the second-best results are presented in italic.

Table 4 compares the relative percentage errors between NeRF and DIVeR64 for offline rendering on various benchmarks. According to the relative percentage error calculation, the DIVeR64 improves the PSNR score by 4.25%, the SSIM score by 1.3%, and the LPIPS measure by 60.49%. Table 4 shows a trade-off between the values of PSNR, SSIM, and LPIPS scores based on the kind of dataset. A deep study on benchmarking the quality necessary for datasets might be one possible future scope in this topic so that the influence of different types of datasets on Image quality, diversity, and fidelity metrics can be investigated more thoroughly. DIVeR rendering quality is far superior to other baselines despite its simplistic design. The PSNR score of the NSVF technique on the Tanks & Temples dataset is 28.40, which is 0.78% higher than that of DIVeR64. To surpass this result, one viable answer is to make advancements in DIVeR64 architecture.

Table 5 shows the relative percentage errors for real-time rendering on the NeRF-Synthetic dataset between NeRF and DIVeR32 & JaxNeRF+ and DIVeR32. The DIVeR32 improves the PSNR score by 3.74%, the SSIM score by 1.16%, and the LPIPS score by 60.49%, according to the relative

Datasets	PSNR	SSIM	LPIPS
NeRF-Synthetic	4.25%	1.3%	60.49%
BlendedMVS	12.8%	9.903%	61.97%
Tanks and Tables	9.3%	5.55%	41.4%

Table 4: Relative Percentage error for offline rendering.

percentage error calculation. To exceed the JaxNeRF+, the DIVeR32 requires a 2.5% increase in the PSNR score, and 0.4% increase in the SSIM score. However, DIVeR32 outperforms JaxNeRF+ by a 15.78% increase in the LPIPS measure.

	PSNR	SSIM	LPIPS
DIVeR32 vs NeRF	3.74%	1.16%	60.49%
DIVeR32 vs JaxNeRF+	-2.5%	-0.41%	15.78%

Table 5: Relative Percentage error for real-time rendering.

5.1 Advantages

The advantages of the work done by the authors of DIVeR [13] are listed below.

- DIVeR [13] gives a comprehensive examination of the traditional integration technique employed in NeRF and proposes a unique integrator as an enhancement in the integration approach, resulting in the capturing thin and translucent surfaces.
- DiveR [13]introduced the MLP decoder in the ray rendering pipeline, an implicit and explicit training strategy to prevent overfitting and preserving high-frequency contents. These tiny changes can be integrated into various baselines and NeRF variations to evaluate how they affect picture quality and fidelity evaluation criteria.
- DIVeR [13] employed regularisation loss [6] to prevent the model from predicting background color in empty space, which is truly essential to build a sparse representation of the 3D scene and minimize computation cost.
- DIVeR [13] used an inference time optimization approach with a suitable threshold for culling operation to keep the features representing translucent surfaces.
- DIVeR [13] also evaluates voxel occlusion caused by a certain camera view and recommends a transmittance value threshold to handle this issue.

5.1.1 Disadvantages

List of the disadvantages of the work done by [13].

- The design of the MLP decoder is extremely simple and requires modification in order to obtain better results in real-time applications.
- This research is presently confined to a few scenes and physical qualities such as glossy surfaces, and it requires a fair comparison with methods such as AutoInt and FastNeRF, the findings of which are not publicly available.

5.2 Limitations

The DIVeR [13] does not naturally accelerate training and thus remains an expensive method for model training. Because of the deterministic integrator, DIVeR [13] can accumulate intersection errors, which can cause aliasing errors as shown in figure 6. DIVeR [13] fails to correctly model view-dependent effects as shown in figure 7. It currently does not apply to unbounded scenes, and its editing capabilities are quite limited.

Figure 6: This figure shows the intersection and aliasing errors in DIVeR [13].

Figure 7: This figure displays that neither NeRF nor DIVeR can accurately represent the reflection on the shovel [13].

6 Conclusion

This seminar report explains DIVeR, a novel integrator for the integration of the classical volume rendering equation used in the 3D Neural rendering of a scene. DIVeR surpasses the NeRF model in terms of speed and accuracy by employing deterministic estimations of the volume rendering integral rather than stochastic estimates. From NeRF findings, the DIVeR64 model for offline rendering improves the PSNR score by 4.25%, the SSIM score [3] by 1.3%, and the LPIPS [4]measure by 60.49%. In terms of storage space and GPU resources, the DIVeR32 model for real-time rendering surpasses KiloNeRF [5], SNeRG [6], and PlenOctrees [7]. The DIVeR also shows good results for the occurrence of natural edits in the generated image.

Future work The evaluation criterion used in this study is limited to rendered images only however the quality of the dataset is not benchmarked in this study. A deep study on benchmarking the quality necessary for datasets might be one possible future scope in this topic so that the influence of different types of datasets on Image quality, diversity, and fidelity metrics can be investigated more thoroughly.

Because of the deterministic integrator, DIVeR [13] can accumulate intersection errors, which can cause aliasing errors. To balance the trade-off between the pros and cons of stochastic as well as deterministic integrators, a novel hybrid integrator consisting of stochastic as well as deterministic integration can be employed.

References

[1] 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. *CoRR*, abs/2003.08934, 2020.

[2] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020.

[3] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[4] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[5] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021.

[6] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021.

[7] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021.

[8] Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. Automatic cinemagraph portraits. In *Computer Graphics Forum*, volume 32, pages 17–25. Wiley Online Library, 2013.

[9] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5712–5721, 2021.

[10] Yu-Lun Liu, Wei-Sheng Lai, Ming-Hsuan Yang, Yung-Yu Chuang, and Jia-Bin Huang. Hybrid neural fusion for full-frame video stabilization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2299–2308, 2021.

[11] Yufei Xu, Jing Zhang, and Dacheng Tao. Out-of-boundary view synthesis towards full-frame video stabilization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4842–4851, 2021.

[12] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14346–14355, 2021.

[13] Liwen Wu, Jae Yong Lee, Anand Bhattad, Yu-Xiong Wang, and David A. Forsyth. Diver: Real-time and accurate neural radiance fields with deterministic integration for volume rendering. *CoRR*, abs/2111.10427, 2021.

[14] Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, W Yifan, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, et al. Advances in neural rendering. In *Computer Graphics Forum*, volume 41, pages 703–735. Wiley Online Library, 2022.

[15] James R Bergen and Edward H Adelson. The plenoptic function and the elements of early vision. *Computational models of visual processing*, 1:8, 1991.

[16] Phelim P Boyle. Options: A monte carlo approach. *Journal of financial economics*, 4(3):323–338, 1977.

[17] James T Kajiya and Brian P Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH computer graphics*, 18(3):165–174, 1984.

[18] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017.

[19] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. Blendedmvs: A large-scale dataset for generalized multi-view stereo networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1790–1799, 2020.

[20] Diederik P Kingma. &ba j.(2014). adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2015.