NeRF and Applications Overview

Technical Report

Sergio Calvo Ordonez
Computational Engineering Group
BAE Systems
Filton, Bristol UK
sergio.calvoordonez@baesystems.com

Abstract

NeRF is a computer vision technique that generates novel views of complex scenes by optimizing an underlying continuous volumetric function using a sparse set of input views. The continuous scene is described as a 5D vector function containing information about the spatial location and view angle. It optimises a deep fully connected multilayer perceptron (no convolution) through gradient descent by minimising the error between the ground truth image and the corresponding views from the generated render.

The intention of this document is to give a short but comprehensive overview on the NeRF method and its inner workings. In term of applications, we will focus on trying to explore the possibility of taking advantage of the NeRF scene synthesis capabilities to potential advancements in the field of systems and control. The focus is towards developing an understanding on how the NeRF technique might be used for online state updates in an autonomous system inside the same environment used for training NeRF. As well, we will study the probable relationship between the NeRF outputs and the current navigation planning algorithms available.

1 Background Concepts and Keywords

1. Rendering: This is the process of generating an image from a 2D/3D model. The model should be able to use features such as textures, shadows, or lightning together with a rendering engine in order to create a realistic image. Examples of rendering algorithms are ray tracing, rasterization, or ray casting.
2. Volume Rendering: Volume rendering is used to create a 2D projection of a 3D discretely sampled dataset. For a given camera position, a volume rendering algorithm obtains the RGBα for every voxels[1] in the space through which rays from the camera are cast.

   [1] A voxel is a value in a regular grid in 3D space.
3. View Synthesis: Ultimately, the objective of NeRF is to create a 3D view from 2D images. A common technique used in 3D reconstruction is to take multiple images from different camera angles and superimpose them. Here, the aim is to predict the depth (the missing axis to 2D $\mapsto$ 3D) through a view synthesis function given a series of images that cover different perspectives of an object. It will be discussed below the role of the NeRF algorithm for this inference task.

1.1 Forward Imaging Model

The entire process of transforming the world scene into an image is encapsulated in a mathematical model called the *forward imaging model*.

This mapping operation can be defined as a projection operation, ultimately, we are projecting 3D representations into a 2D plane. Consider:

$$x = PX, \tag{1}$$

where x is an arbitrary projected 2-dimensional point, X is the same point in 3D (real-world), and P is the projection matrix that, for this context, we will refer to as camera-matrix. Expanding on this equation:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{2}$$

We observe that the camera-matrix P is, in fact, an affine transform matrix that, with the concatenation of a 3x1 column vector (image height, image width, focal length), it is possible to create the *pose matrix*.

We can easily derive the camera-matrix P from the claim that we want to map between the camera coordinate frame (origin at the camera position) and the world coordinate frame (e.g. the frame of reference where wee see all shapes and objects that exist in the 3D real world). In fact, P is a consequence of this mapping.

Taking into accound that the camera (and therefore its coordinate frame) is inside the world coordinate frame, we can state:

$$X_c = R \times (X_w - C_w), \tag{3}$$

where X_c are the spatial coordinates in the camera's frame of reference, R is the matrix (3×3) representing the orientation angle of the camera frame with respect to the world coordinate frame. Each of R's rows are the directions in each of the coordinates (x_c, y_c, z_c) in the world coordinate system. C_w is the vector representing the position of the camera with respect to the world coordinate frame. Following with Eq. 3:

$$X_c = R \times X_w + t, \tag{4}$$

where $t = -(R \times C_w)$.

Now, we will compress this operation to a simple individual matrix multiplication with the use of *homogeneous coordinates*. Using this concept we can transform the 3-dimensional X_w to a 4D vector X'_w:

$$X_w \equiv \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \equiv \begin{bmatrix} wx_w \\ wy_w \\ wz_w \\ w \end{bmatrix} \equiv \begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} \equiv X'_w. \tag{5}$$

Equivalent to what we see in Eq. 2.

Therefore,

$$X'_c = C_{\text{cam}} \times X'_w, \tag{6}$$

where C_{cam} is the camera-matrix which represents the values of rotation and translation of the camera.

$$C_{\text{cam}} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

1.2 Inverse Camera Mapping (2D $\mapsto$ 3D)

Here is where techniques such as volume rendering with ray casting and tracing come into play.

Consider an image with N pixels. The process consists in shooting a ray through each pixel and sampling points across it while, at the same time, adding noise to each sample so that the samples correspond to a continuous distribution. We will discuss later on how the authors approach this problem and obtain a continuous space without the computational expense of having to excessively query the network for each sampled point. These rays are parameterised as:

$$r(t) = \mathbf{o} + t\mathbf{d}, \tag{8}$$

where t is the time-step parameter, o is the origin, and d is the unit directional vector. In the context of NeRF, rays are generated by taking the origin of the ray to be the pixel position of the image plane and the direction as the straight line joining the pixel and the camera aperture.

In practise, the points we sample from the ray have their own unique spatial coordinates (x, y, z), while, across the same ray, they will have the same viewing angle (θ, ϕ). If we manage to carry this sampling process across arbitrary rays for all pixels, we will manage to construct the 5D vector inputs that the network in NeRF requires.

It can be derived from the properties of similar triangles that:

$$u' = C_{in} \times x'_c, \tag{9}$$

where x'_c is the set of vectors containing the location of the point in the camera coordinate space, and u' is the set of values containing the location of the point on the image plane.

$$C_{in} = \begin{bmatrix} f & 0 & o_x & 0 \\ 0 & f & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{10}$$

C_{in} is referred to as *camera intrinsic* since it represents values like the focal length and the center of the image plane along the 2D axes, o_x and o_y (all internal properties of the camera).

2 Introduction

NeRFs take a collection of camera images and train a fully-connected neural network (multilayer perceptron, MLP) to give a function relating each 3D point in space with a density and a vector of RGB values ("radiance"). This representation can be then used to generate photo-realistic images through different classic volume rendering techniques.

In a slightly more detailed manner, the NeRF algorithm is capable to represent a scene thanks to a MLP network whose input is a single 5D vector in a continuous space that includes the spatial coordinates (x, y, z) and the viewing direction (θ, ϕ). This data points come from the sampling discussed in 1.2 after applying hierarchical sampling (see 5.2). The output of the network, on the other hand, is in a 4D space that includes the RGB color and a single volume density (Eq. 11).

$$(x, y, z, \theta, \phi) \rightarrow F_\theta \rightarrow (r, g, b, \sigma), \tag{11}$$

In contrast to previous work on volume representation, here a continuous function approximation is being fitted to the volume without ever instantiating a grid of individual samples (voxels). This allows to trivially add extra dimensions without this incurring any extra storage costs. With NeRF, this is possible by appending a few extra numbers to the input of the network and then it can decide how to allocate its capacity to represent the important parts of the function.

An important characteristic of the volume rendering technique used in this process is that it is differentiable, meaning that, for optimization, it will only be required a set of images with relative known position. The algorithm uses gradient descent to minimise the error between the observed image and the rendered images from the representation. The key to robust results in terms of the predicted volume density and colours is to conduct this minimisation process across many different camera angles.

Optimizing with gradient descent on rendering can be described in the following manner:

$$\min_\theta \sum_i ||\text{render}_i(F_\theta) - I_i||^2 \tag{12}$$

As discussed above, this function minimises the error between the values of the rendered pixels and the value of the same pixel from the ground truth. This is called *photometric loss*.

In order to develop NeRF, a concept from transformers was borrowed, more specifically, positional encoding. In transformers (in NLP), positional encoding is used as a mechanism that substitutes the ability that RNNs or LSTMs have of being able to capture the relative positions of words in a sentence. Unlike recursion or convolution, positional encoding is not a part of the model but it forms part of the preprocessing pipeline. In the original paper for transformers

Attention is all you need the method they used for injecting information about the position of the tokens was by adding sin and cos functions that have the same dimensions as the embeddings, d_{model}.

In the context of NeRF, the authors used the following encodings:

$$sin(2^{L-1}\mathbf{v}), cos(2^{L-1}\mathbf{v}), \tag{13}$$

where L is the position of the input in the d-dimensional row input vector, and $\mathbf{v}$ is the original input vector before applying positional encoding.

This simple preprocessing step allows the MLP to represent higher frequency functions. This is accompanied by a hierarchical sampling procedure to reduce the number of queries across the rays that are needed to sample from the high-frequency scene representation (high frequency sin and cos functions).

It could be argued that NeRF is really data hungy because it tries to memorise the whole world (overfit the scene). This exact reason also explains why this model is not able to generalise scenes, e.g. it needs to be retrained for each new scene that the user wants to synthesise. This generates considerable performance constraints that will be discussed in a latter section. As a positive note, there is a lot of work done on investigating alternatives on top of NeRFs that notably improve the performance.

3 Related Work

- **Neural Implicit Representations:** These use neural networks to create representations of the geometry of a 3D complex environment. With supervised learning, it learns a straight-forward function of the form:

  $$f_\theta(p) = \sigma, \tag{14}$$

  where f is the neural network in terms of its current weights θ, p is a low-dimensional query point (x, y, z coordinates), and σ is some quantity of interest.

  These methods try to generate the representations by optimizing neural networks to map the spatial coordinates to a signed distance function (SDFs) or occupancy fields. The limitations, however, are originated from the fact that it is necessary to have access to the ground truth 3D geometry and therefore the datasets need to be synthetic (e.g. ShapeNet). Lately, this constraint has been relaxed by coming up with differentiable rendering functions in order to allow the neural implicit shape representations to be optimised with 2D images instead.

  In general, the downsides of the different methods used for neural implicit representations are that they are able to perform only with simple geometrical complexity shapes and therefore when more complex scenarios are used, we see oversmoothed renderings.

- **View Synthesis and image-based rendering:** View synthesis aims to create new views of a scene given a set of images with sparse-angle views. Similarly, image-based rendering techniques pretend to render novel views directly from input images. Work on these image-based rendering methods can be classfied into three main categories: rendering without geometry, rendering with geometry, and rendering with implicit geometry.

  The most advanced methods try to use volumetric representations to create the view synthesis from a set of RGB images obtaining realistic results and being able to represent complex shapes with less artifacts than previous work in the field. The idea is to train deep networks to predict a **sampled** volumetric representation of the set on input images and then use rendering techniques to generate the novel views. Interesting work has been done on using convolutional networks (CNNs) and sampled voxel grids. The CNN is presented to try to make up for the discretization that emerges from trying to predict sampled volumetric representations in low resolution voxel grids.

  These techniques are not able to scale to high resolution images due to its poor time and space complexity tied to the discrete sampling. NeRF solve tries to solve this issue by implementing an encoding step of a **continuous** volume within the parameters of the deep dense neural network. This improvement leads to better memory requirements and higher quality renders.

4 Neural Radiance Fields and Volume Rendering

As briefly mentioned in section 2, NeRF uses a MLP network to approximate the 5D input vector that encapsulates information of the scene representation.

$$F_\theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma). \tag{15}$$

The network's weights are optimised to map from each 5D input vector to its corresponding directional emitted colour and volume density. For consistency of the results across different views, it is essential to encourage the network to output the same volume density for the same point independently of the angle from where it is viewed from. This is not the case for the colour, as this can change with position due to different effects such as reflections or the properties of the material. To achieve this, the MLP processes the input data in two batches. First, the $\mathbf{x}$ vector goes through 8 fully-connected layers using ReLU activation and 128 channels, and outputs the volume density σ and a 256-dimensional feature vector. The feature vector will be used together with the camera ray's viewing direction to output the view-dependent RGB colour after being passed through an additional dense layer that uses a ReLU activation and 128 channels.

Volume rendering is used to render the colour $C(\mathbf{r})$ of any ray in the scene. The volume density $\sigma(\mathbf{x})$ is the probability of the ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ terminating at any given point $\mathbf{x}$ in space.

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt, \text{where } T(t) = exp(-\int_{t_n}^{t} \sigma(\mathbf{r}(s)))ds. \tag{16}$$

Here, $T(t)$ is the transmittance (probability of not interfering with another particle) along the ray from t_n to t.

To estimate this integral, the authors of the original NeRF paper use quadrature. To avoid discretization, they use a stratified sampling approach where they partition the range of t into N evenly spaced bins and sample uniformly at random within every bin. Although it might seem discrete, it is not because the MLP will end up evaluating the integral at continuous points over the course of the optimization.

$$t_i \sim \mathcal{U}[t_n + \frac{i-1}{N}(t_f - t_n), t_n + \frac{i}{N}(t_f - t_n)]. \tag{17}$$

These samples are then used to estimate $C(\mathbf{r})$ with a differentiable function:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i, \text{where } T_i = \exp(-\sum_{j=1}^{i-1} -\sigma_j\delta_j), \tag{18}$$

where $\delta_i = t_{i+1} - t_i$ is simply the distance between adjacent samples.

5 Key Elements in NeRF

The authors of the original NeRF paper introduce two important improvements that allow the technique to achieve state-of-the-art results when representing high-resolution complex scenes. These are *positional encoding*, and *hierarchical sampling*.

5.1 Positional encoding

Letting the network by itself handle the 5D input vector was shown to lead to poor results when representing high-frequency variation of color and geometry. Oversmoothing was a big problem. This is explained by the work by Rahaman *et al*, which shows that deep networks are biased towards learning lower frequency functions. This can be helped by mapping the inputs to a higher dimensional space made of high frequency functions.

In fact, this is exactly the aim of the positional encoding implementation. The idea is to break up the function approximator F_θ in two components. One is optimised through training and the other one is a fixed function that will handle the mapping to a higher dimensional space.

$$F_\theta = F'_\theta \circ \gamma, \tag{19}$$

where F'_θ is still an MLP network and γ maps from $\mathbb{R} \mapsto \mathbb{R}^{2L}$.

$$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p), ..., \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)). \tag{20}$$

This second function is applied to each coordinate of the inputs (that must be normalised into the range [-1,1]).

5.2 Hierarchical volume sampling

Evaluating the neural network at N points along each ray is not optimal. It would not make sense that this happens for regions of space where there is nothing or they are occluded, in fact, these do not contribute anything to the final render and therefore it is a waste of computational resources.

Hierarchical volume sampling is presented to address this problem by providing the model with the ability to do a smarter sampling. It gives more weight to regions of space where objects are more likely to be found. This technique makes use of two different (but identical) networks, the "coarse" and the "fine" network. The first step is to sample a set of N_c locations with stratified sampling and evaluating the "coarse" network on them. With this output, it is possible to carry out an informed sampling along each ray in areas where there is only relevant information. This will roughly tell us about how much each location contributed towards the final colour. From this distribution we can now sample using inverse transform sampling and evaluate the second network, the "fine" one, at the union of the first and second set of samples to then compute the final rendered color of the ray. This is possible due to the fact that we are using Eq. 18 whose weights can be treated as a probability distribution from where we can sample more points instead of having to query the network at more points in the 3D space or in the object.

Ultimately, this allows the model to focus on regions where it is expected to find visible objects.

This works in the order:

1. Use stratified sampling to sample N_c points across the ray and then input them into the course network. Lastly, we use this N_c sample for the following steps.
2. We rewrite Eq. 16 as Eq. 18, or in clearer terms:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N}(w_i c_i), \tag{21}$$

   where $w_i = T_i(1 - \exp(-\sigma_i\delta_i))$

3. We normalise the weights, w_i:

$$\hat{w}_i = \frac{w_i}{\sum_{j=1}^{N_c} w_j} \tag{22}$$

4. Now, we can treat this as a probability density function and sample a second set of N_f samples using the inverse transform sampling method.
5. Once this is done, we can use the union of both sampled sets ($N_c + N_f$) to input to the fine network and produce the final rendered color of the ray.

5.3 NeRF Multilayer Perceptron

In order to reproduce the results from the original paper, we will use the same parameters listed there. They use a batch size of 4096 rays, each sampled at $N_c = 64$ coordinates in the coarse volume and $N_f = 128$ additional coordinates in the fine volume. They use an Adam optimizer with a learning rate that begins at 5×10^{-4} and decays exponentially to 5×10^{-5}, the rest of its hyperparameters are left at default.

With respect to the architecure of the MLP, it processes the input 3D coordinates with 8 dense layers using ReLU activations and 256 channels per layer, and outputs σ and a 256-dimensional feature vector. Then, this feature vector is concatenated with the camera ray's viewing direction and passed to one additional dense layer using ReLU activation and 128 channels. A final layer with a sigmoid activation outputs the view-dependent RGB colours.

A Coordinate Transformations

We can compute the pixel positions of the 2D image with respect to the camera coordinate frame using:

$$u = f\frac{x_c}{z_c} + o_x \rightarrow x_c = z_c\frac{u - o_x}{f}, \tag{23}$$

$$v = f\frac{y_c}{z_c} + o_y \rightarrow y_c = z_c\frac{v - o_y}{f}, \tag{24}$$

where u, v are the pixel values, f is the focal length, and o_x, o_y are the coordinates of the origin in the image plane.

From Eq. 6, we can rearrange,

$$X'_w = C^{-1}_{\text{cam}} \times X'_c, \tag{25}$$

where C^{-1}_{cam} is the inverse of C_{cam}.

To define a 3D direction vector we will not need the entire C^{-1}_{cam} but only the inner 3×3 orientation matrix R. With this matrix, we can get the unit direction vector.

$$\mathbf{d} = \frac{R'_{\text{cam}} \times X_c}{|R'_{\text{cam}} \times X_c|} \tag{26}$$

Therefore, the translation vector of the camera-to-world matrix will be:

$$t'_{\text{cam}} = \begin{bmatrix} t'_x \\ t'_y \\ t'_z \end{bmatrix}. \tag{27}$$

B The Mathematics of Sampling

A default sampling approach would be sampling points at regular intervals, the equation is as follows:

$$t_i = i\frac{t_f - t_n}{N}, \tag{28}$$

where t_f and t_n are the farthest and nearest points on the ray respectively. N is simply the number of intervals we want to divide the ray into.

However, the approach authors use for NeRF is beginning with stratified sampling. They regularly partition the interval $[t_n, t_f]$ into N evenly-spaced bins, but then draw one sample uniformly at random from within each bin:

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n), t_n + \frac{i}{N}(t_f - t_n)\right] \tag{29}$$