Explanation of NeRFs: Neural Radiance Fields.

Taskeen Jafri

Abstract. I will be explaining Neural Radiance Fields, also known as NeRF. [1] NeRFs use a method that achieves state-of-the-art results for synthesizing novel views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of input views. It's algorithm represents a scene using a fully-connected (nonconvolutional) deep network, whose input is a single continuous 5D coordinate (spatial location (x; y; z) and viewing direction (θ,ϕ)) and whose output is the volume density and view-dependent emitted radiance at that spatial location. The model synthesizes views by querying 5D coordinates along camera rays and use classic volume rendering techniques to project the output colors and densities into an image. Since volume rendering is naturally differentiable, the only input required to optimize the representation for the model is a set of images with known camera poses.

1. Introduction.

NeRFs are modeling softwares which take multiple 2D images and output a 3D model. Basically we take images of the object from all around it, and input this set of images into the NeRF model. It will process these images and give a stunning, high quality 3D model. Of course, images can't be taken from all 360 degrees around an object. However, the speciality of NeRF is, it gives the output such that it can generate new views by itself! By that, I mean that the model can generate an image from a new viewpoint for the same object, even those not included in the input dataset.
It is highly important and has multiple applications, such as video games, or computer graphics in general, and Virtual Reality.

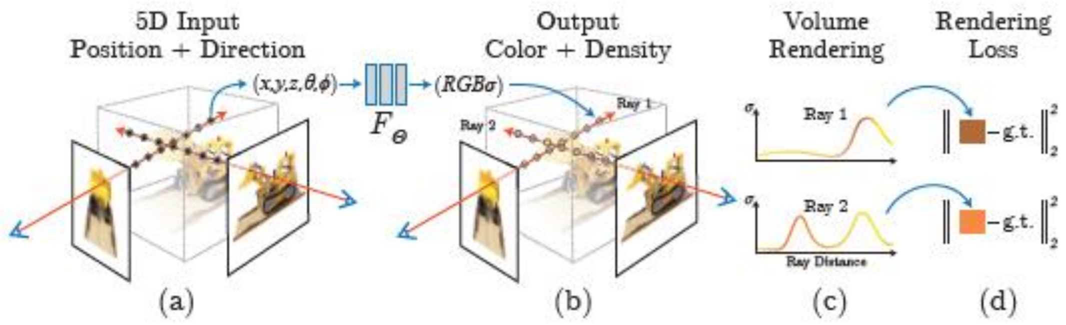
In this context, radiance means the intensity and colour of light at a particular point in the scene. So, radiance is just the RGB colour of the point in a scene.

NeRFs are multi-view consistent, meaning that when the same point is rendered from different views, it gives the same output.

2. NeRF Mechanism

One thing to remember in a NeRF is that the weights in a NeRF differ from that of a, say, CNN. In NeRFs, weights *represent* the scene itself, whereas in CNNs, the weights determine the contribution of each neuron to the overall output. Weights in a NeRF are learnt by optimizing the radiance values, i.e., by minimizing the difference between calculated and ground truth values.
We give the model the 2D images along with the camera poses (positions and orientations), and the model converts the 2D image pixels to 3D positions (x,y,z), along with the position and orientation (θ,ϕ), hence making it a 5D function. (Also, the poses are generated using SfM software.) This 5D function is passed through the NeRF:

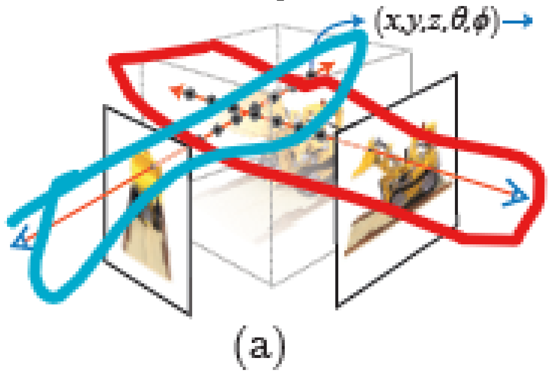
(image taken directly from the original paper)



The input is (x,y,z,θ,ϕ), where the position is called ‘x’, and orientation is called ‘d’. So, x is (x,y,z) and d is (θ,ϕ). The input, (x,y,z) and (θ,ϕ), is passed through the function, F_Θ, and the model converts this input to RGB, along with density σ, which shows whether the space at a particular point is opaque, translucent, transparent, etc.

The mechanism is, basically the model takes a 2D image, and constructs a number of **camera rays** from that image. Each camera ray has a specified number of randomly sampled ‘points’ in it. When generating a new viewpoint, the model has to evaluate what will be there in place of every pixel (for the scene). This is done by evaluating the radiance and density at every point in a camera ray generated from an image, for multiple images. However, this is computationally extremely expensive and could take a long time to generate this viewpoint. So we specify the number of **sample points** in every camera ray. More number of sample point = high resolution, but longer processing time. The original code specified the number of sample points as 1024 in every camera ray, using the ‘N_rand = 1024’ command.

Here is a bit of visual representation:



You can see the red **camera ray** originating from image ‘1’ while the blue camera ray originates from image ‘2’. The black points are the **sample points**, per ray. These sampled points are then passed through the function F_Θ, which generates radiance and density at those locations.

One great idea applied while generating new views, the developers made NeRFs multi-view consistent by making the density, σ, dependant only on ‘x’, whereas the colours, RGB, depend both on ‘x’ and ‘d’.

2.1 Rendering method

While integrating the sample points for the radiance at each point, the NeRF model uses quadrature, specifically, stratified quadrature. Between deterministic quadrature and stratified quadrature, the basic difference is that deterministic quadrature performs well for smooth images but cannot process complex details, such as lego in this example. However, stratified sampling divides and samples each block and integrates them individually, hence, it can process extremely complex and fine details in the image. Stratified sampling is a bit heavy and hence takes a longer time to process, but for fine and detailed images, it's worth it.

2.2 Optimization techniques

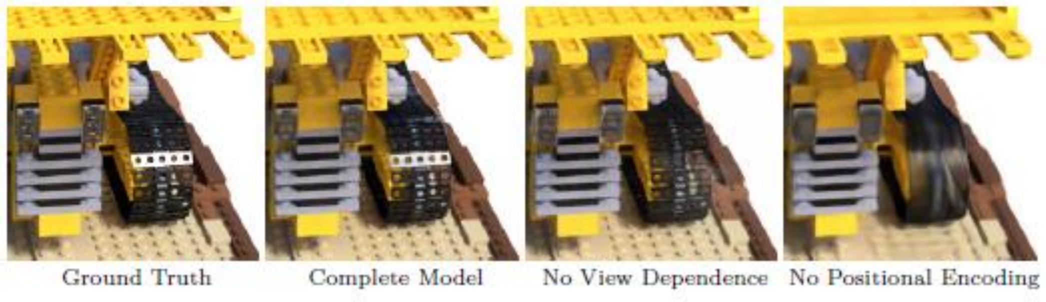
2.2.1 Positional Encoding:

The authors also made use of the 'Fourier Transform', which they called 'positional encoding'. Fourier Transform helps in calculation and approximation and, basically, the detailing of the final 3D scene. It uses sums of sine and cosine waves to compute the approximate output. Another important thing this does is, that it *increases the number of dimensions!* It has been observed that neural networks work significantly better with higher dimensional input. This is the function used:
$\gamma(p) = (\sin(2^0\pi p), \cos(2^0\pi p),\ldots \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p))$.
'p' is the input to the NeRF (Input meaning (x,d)), which is normalized to lie between [-1,1].
In the paper, the value of 'L' is 10 for $\gamma(x)$ {The spatial coordinates}, whereas the value of 'L' is 4 for $\gamma(d)$ {orientation}.
The values of 'L' aren't arbitrary, the authors tested higher, as well as lower values of 'L', and for these scenes, the above stated values give the best results.

2.2.2 Hierarchal Volume Sampling:

Another 'trick' used is 'Hierarchical Volume sampling'. What this does is, it divides the space into sub-volumes and processes the scenes. While processing, it renders different volumes at different resolutions depending on their importance to the scene! The model first makes a coarse network, and then a fine (detailed) network. This helps in reducing computational load, and still delivers a quality output.

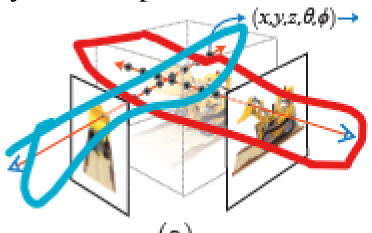
Why, and how much these two affect the final result, can be seen from the picture belong, provided by the authors in their paper:



Without Positional Encoding, the final product is a kind of an oversmoothed image. We can also see how view dependance can show reflections when viewed from a certain angle, making it so much more realistic.

2.2.3 Optimization:

Now that we have discussed the optimization techniques, let's see how they are implemented in the NeRF model. The model needs to improve the scene it is working on. It starts by randomly selecting a group of camera rays from the entire dataset of images and pixels, and then improves that group in each step. The camera rays can come from any pixel in any of the images, and they are rays that extend from each pixel.
[To clarify, the camera rays may come from any pixel in any of the images. Also, camera rays are the rays that are constructed and extend outwards from a pixel (or camera position). There may be any amount of camera rays from a pixel.]



(a)

Here, the red- and blue- highlighted rays are the camera rays I’m talking about.

So, we have the camera rays and hence, sample points to optimize. This is passed through Hierarchical Volume Sampling. For the coarse network made, it queries N_c, while for the fine (detailed) network, it queries $N_c + N_f$ samples. These samples are basically just 3D points along with position and orientation. They are passed through volume rendering to estimate their colour and density. In the paper, the authors used N_c equal to 64, which N_f = 128 points.

The optimization is carried out for the following function:

$$\text{Loss} = \sum [||\hat{C}_c(r) – C(r)||^2_2 + ||\hat{C}_f(r) – C(r)||^2_2]$$

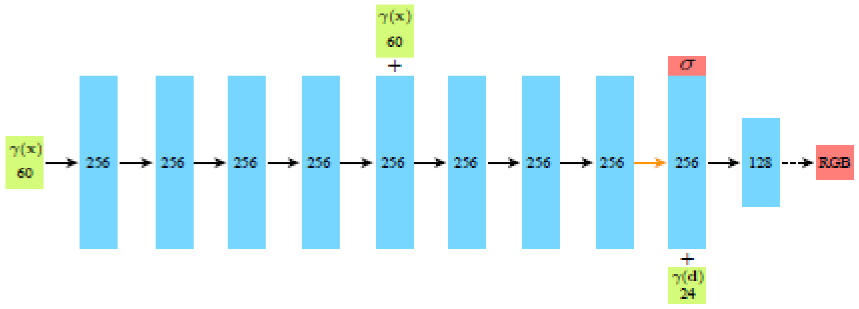
The subscript simply means the root of sum of the squares, i.e., it would be the same to write the above equation as:

$$\text{Loss} = \{\sum [||\hat{C}_c(r) – C(r)||^2 + ||\hat{C}_f(r) – C(r)||^2]\}^{1/2}$$

I added the green highlight to show changes.
Here, $C(r)$, $\hat{C}_c(r)$, $\hat{C}_f(r)$ are ground truth, coarse network colour, and fine network colour repectively. Even though $C(r)$ is ground truth, it is unknown to us, so we optimize all 3 variables, to minimize the loss. We optimize $\hat{C}_c(r)$, which belongs to coarse network, as well so that we can use the same weight distribution for our fine-detailed network.
In the paper, the authors used an Adam optimizer, with initial learning rate as $5\text{x}10^{-4}$, which decays to $5\text{x}10^{-5}$ over the course of optimization period. They used 100,000-300,000 iterations to construct each video, which took around 1-2 days on an NVIDIA V100 GPU.

3. The Layers

(image taken from the authors' paper)



The model contains 8 hidden layers. The green highlights show input vectors, hidden layers are shown in blue while red highlight shows output vectors. All layers are standard fully-connected layers, black arrows indicate layers with ReLU activations, orange arrows indicate layers with no activation, dashed black arrows indicate layers with sigmoid activation, and "+" denotes vector concatenation.
"256" denoted the number of channels. There are 64 samples for coarse network (N_c), and 64+128 = 192 samples for fine network (N_f), as mentioned previously, making it a total of 256 channels.
The positional encoding of the input passes through 8 fully connected layers, each with 256 channels. The architecture of the model includes a skip connection at the 5th layer, which helps improve the recognition of the model and also helps prevent the vanishing gradient problem. It helps the network retain more information and improves image recognition. An additional layer outputs the density, σ. All of this finally passes through a 128-channel layer, then a sigmoid activation function and we get an RGB radiance output, at position x, as viewed with direction d.

References:

Authors of the original NeRF paper:
Ben Mildenhall, Pratul P. Srinivasan, Matthew Tanick. Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng.