Manipulation of NeRF

Hayeon Kim 2018-14265
Wonseok Choi 2018-11549
Yunseong Hwang 2018-11339
Electrical and Computer Engineering

1. Introduction

NeRF is an algorithm for representing a realistic volumetric scene using a fully-connected deep neural network called MLP. NeRF is known as one of the most influential work in 3D vision field due to its advantages on high-resolution output and storage space.

However, editing NeRF is an extremely challenging task. Since NeRF has an implicit function optimized per scene, we cannot directly edit its shape. Moreover, the final output is too strongly connected to the input images, it is hard to change the output's light direction or color. This multi-view dependency of NeRF makes the modification of its output more complicate.

For these reasons, we aimed at manipulation of NeRF's 3D output. We introduced the new way of editing the object color and the location of light source, through the RGB map information of naive NeRF. As a result, we've shown that our method could effectively generate the modification version of NeRF's output. Furthermore, we achieved the optimization of overall calculation and reduced our rendering time successfully.

Our code implementation is available at [*https://github.com/janeyeon/computerVision2022.git*].

2. Relevant Works

2.1. NeRF

NeRF is a representative one among representation progress of 3D models with neural networks.

Neural radiance field (NeRF) represents a static scene as a continuous 5D function$(x, y, z, \theta, \varphi)$ that outputs the radiance emitted in each direction (θ, φ) at each point (x, y, z) in space, and a density at each point.

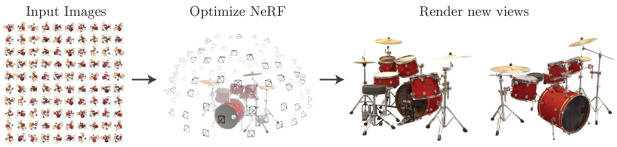


Figure 1. Overall pipeline of NeRF

It uses a method that optimizes a continuous 5D neural radiance field representation of a scene from a set of input images. By the techniques from volume rendering to accumulate samples of this scene representation along rays, NeRF renders the scene from any viewpoint.

NeRF encodes a continuous volume representation of shape and view-dependent appearance in the weights of an MLP network.

Also, it has strong capability in capturing high-resolution geometry and rendering photo-realistically novel views.

2.2. Homography matrix

Translation of a vector is essentially affine transformation. However, N-dimensional affine transformation could be replaced with the N+1 linear transformation by using homogeneous coordinate. For example, a specific vector $[\mathrm{x}, \mathrm{y}]^T$ would be $[\mathrm{x}, \mathrm{y}, 1]^T$ in homogeneous. In reverse, a vector $[\mathrm{x}, \mathrm{y}, \mathrm{z}]^T$ in the form of homogeneous coordinate

would be $\left[\frac{x}{z}, \frac{y}{z}, 1\right]^T$. In addition to the convenience with converting to linear transformation, division, which cannot be represented with the matrix multiplication, also can be applied by using matrix operation. If we use homogeneous coordinate, we could easily exploit the principles of linear algebra. Then, translation, rotation, scaling transformation matrix of N- dimensional vector would be (N+1) by (N+1) matrices. Then with the principle of linear algebra, this sequential transformation could be represented by just one matrix, which is called homography matrix. There are examples of transformation matrix of 2D vector in the form of homogeneous coordinate.

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

They are translation, rotation, scaling matrix in sequence. It can be easily expanded to the higher dimension. And we can check the overall transformation can be represented as a single matrix multiplication as above.

2.3. Lambertian reflectance model

When we see an object, our eyes detect the light with the color and brightness information. The Light we observe consists of 3 different components, which are diffuse, specular, ambient. Photons in light are reflected at the surface of an object. If the surface is not even, each photon will be scattered with every other direction, which is called diffuse reflectance. It gives us a uniform color of the object independent of the viewing direction. If the surface is somewhat even, some amounts of the light can tend to be reflected with a single direction. This is called specular reflection. Ambient color is the combined color of the lights which are reflected many times before it reaches the surface of an object. These kinds of light are independent of the direction or intensity of the light we provided. Combined light is uniformly colored. The overall color representation will be the sum of these three light components. To calculate the specular color, we can use the law of reflection. If we set the scenario which reflection happens on the surface of an object as below. We can let the specular light $R_e = \begin{cases} R_i \; if \; v = r \\ 0 \; otherwise \end{cases}$. And we can calculate the vector $\mathbf{s} = 2(\mathbf{n} \cdot \mathbf{r})\mathbf{n} - \mathbf{r}$. So, if we know the light origin information and the normal vector of the surface of image. We can get the specular color map.

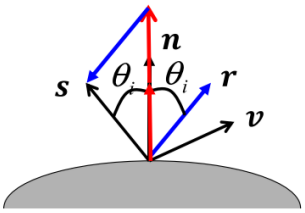


Figure 2. Law of reflection

Ambient color is the multiplication of diffuse color with the intensity of ambient light. It can be represented as below (note that each term is 3-dimensional RGB vector). And multiplication is applied element-wisely.

$$\text{Ambient\_color} = \text{diffuse\_color} * \text{ambient\_light}$$

3. Data Processing

3.1. Blender

We used blender program to accurately generate the 3D object we wanted. The object we used is a yellow colored sphere, which center is in (0, 0, 0.5) from origin. We used the point light as our light source located in (0, 0, 5), on the top of the sphere.

Using the composition feature in Blender, we rendered the normal and depth map together. The procedure of generating the whole images by hand is a bit burdensome, we used python script instead. The Python script is included in the given blender files[1].

With Blender, we rendered the wanted output

[1] https://drive.google.com/file/d/1RjwxZCUoPlUgEWIUiuCmMmG0AhuV8A2Q/view

image, whose size is 800×800 pixels. Output images include the normal, depth and naïve object images from different views of the camera sample on the upper hemisphere. Training the NeRF network requires approximately 800 images that will later be divided into training, testing, and validation sets.

In NeRF, all the camera coordinates are represented as translation matrix, called c2w, we also convert our camera position to matrix and export it to the json file. The json file also includes images' file directory and rotation information etc.

4. Approaches

Before starting the concrete explanation about our implementation, we would introduce the overall pipeline and NeRF's basic dataset analysis. Since our project is focused on the modification of the NeRF network's output, it is also important to understand NeRF correctly.

4.1. Overall Pipeline

In original NeRF, the pre-trained network takes camera pose as an input and generates the RGB and depth map as output. Using that information, NeRF renders the 2D image which represents the target 3D object.

In our project, we first trained the original NeRF with the 3D objects we intended. And we appended the new RGB function right before the rendering process, to handle the NeRF's RGB map. For this, we additionally gave the light direction and object's color information as the arguments of our RGB function. After the whole re-calculation of RGB map, we could finally get the manipulated output we wanted.

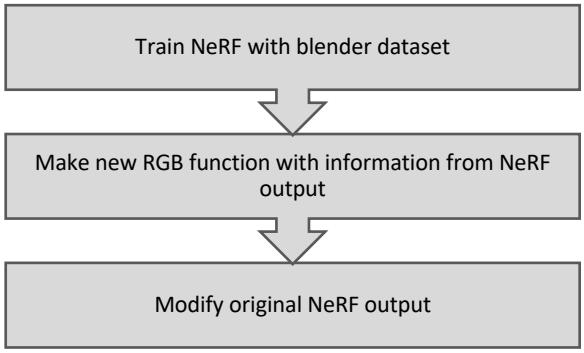


Figure 3. Overall pipeline of our project

4.2. NeRF dataset analysis

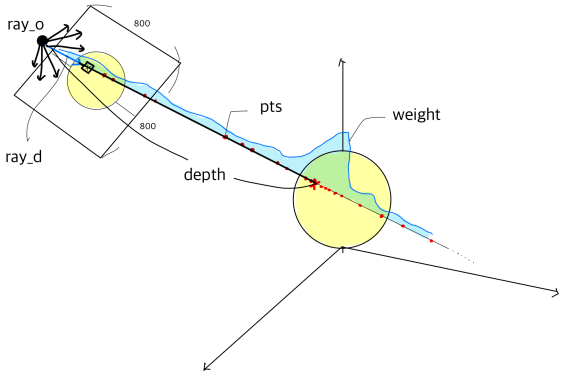


Figure 4. Visualization of basic NeRF

The above picture represents the basic NeRF network. The camera point, equals to the viewpoint of rendered image, is named ray_o. And the directions of rays, which named ray_d, spread out from the ray origin point to the output image pixel respectively. The total number of rays is equal to the size of output image, 640000. If we set the configuration of half-resolution as true, the total output size would be reduced by a quarter. Each ray is divided into several points that have irregular interval distance, followed by the coarse and fine hierarchy of NeRF network.

The blue area along the ray, called weight, represents the probability of a ray being terminated at a position x. If there is a point where the weight has higher value than other places, it means that there is a high probability of an object existing.

The accumulative sum of weights multiplied by the

positions of each point equals to a depth value. In this picture, the depth value is the distance between the ray_o and the object's surface point.

RGB map of the NeRF network means the color information represented by the pixel of output image. Using the RGB map, we could render the final 2D image immediately.

5. Implementation Details

In this part, we would explain about the detailed points of our implementation. The whole calculation process is implemented in the code, more specifically.

5.1. Ground Plane Shadow

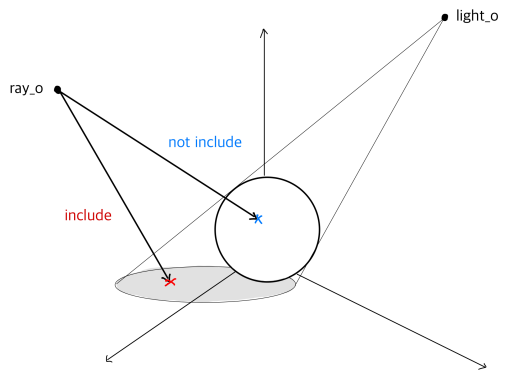


Figure 5. Geometrical relationship of the light and shadow of the sphere

The method of measuring ground plane shadow is the same method as determining whether the termination point of ray_d belongs to the shadow. This relation is depicted on the above image. However, calculating the shadow composed of an elliptical equation is quite difficult. Thus, we transformed the ellipse to a center aligned circle to use an affine transformation method.

Shift the ellipse's center point to the origin. The ellipse center can be calculated as the end point of the extension line, including both light source position and sphere center.

Next, by using the rotation matrix, make sure that the tilted ellipse becomes parallel to x-axis. For that, we projected the light source and sphere center to the x-y plane and found two distinctive points. Normally, light has the property of going straight, the direction of the shadow will be parallel to the line connecting those two points. The shadow would be parallelly generated to the line connecting those two points. The angle between x-axis and that the line create is the one we wanted to rotate.

Finally, scale the ellipse and make the circle whose diameter is equal to the major axis of the ellipse. The scale factor is major axis length over minor axis length of the ellipse.

Verify that the end point of ray_d is included in the circle after the transformation. Find the point at the z-value of the line is zeroed and check its inclusion relationship. If the point's norm value is less or equal than the circle's radius, it means that the point belongs to a circled area, in other words, the shadow area.

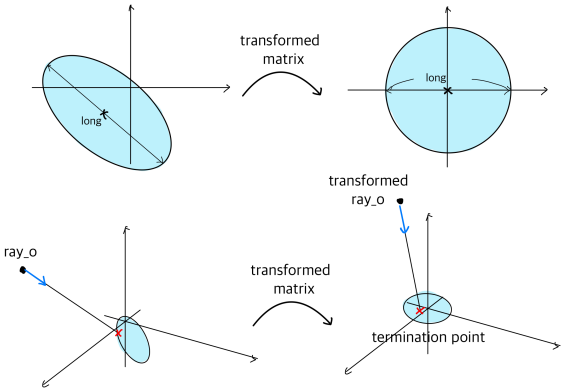


Figure 6. Affine transformation of shadow ellipse

5.2. Sphere Color

Simply, we could check inclusion relationship of the sphere by using the distance between center point and ray. But it takes high computational cost, and hard to generalize due to the irregularity of points interval. Therefore, to determine whether a particular ray_d belongs to the sphere or not, we've used cosθ.

Using the relation depicted on the picture below, we could calculate the cosθ, as $\|p\|/\sqrt{p^2 - radius^2}$.

The cosθ is inverse proportional to the maximum angle that ray_d can have, and proportional to the size of sphere radius. If the dot product value between ray_o_center and ray_d is greater than pre-calculated $\cos\theta$, it means that ray_d belongs to the sphere. Due to this geometric relationship, we could easily find the

whole ray directions entering the sphere. Hereafter, all the subsequent calculations are only based on the ray_d inside the sphere to reduce the amount of computation.

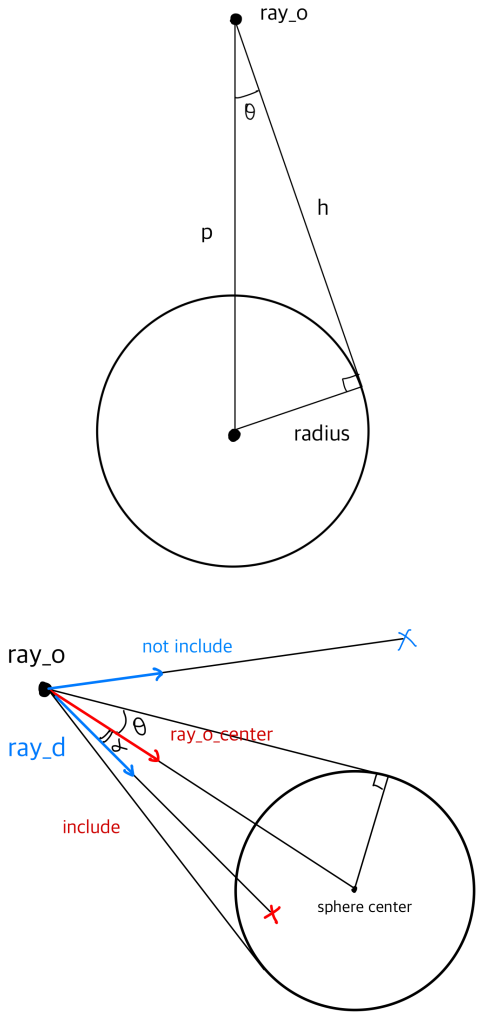


Figure 7. Inclusion relationship between ray and the sphere

5.3. Sphere Shadow

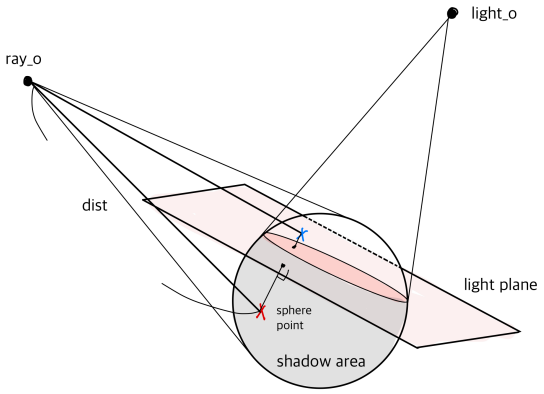


Figure 8. Calculating distance between the light plane and ray

Next discussion is about the method of finding whether the particular ray_d is included in the sphere shadow. The shadow means the area that light cannot reach. Based on the points at which the light touches the sphere, we can draw a plane as follows. The lower portion of the plane represents an area to which light does not reach in the sphere, and it becomes a shadow area. By calculating the distance between the plane and the sphere point, we can figure out whether each point belongs to the shadow or not. If the distance value is positive, it means that the point located the upper area of the plane, and negative, the point located the under area of the plane.

However, finding the sphere point only with the ray_d information is a bit difficult. Thus, we used NeRF's depth map value to extract the sphere point immediately. As mentioned before, the depth map represents the length between the ray origin point and the object surface.

In addition, we used the sigmoid function to blur the boundaries of the shadow and make them look more natural. We set the degree of shadow with the weight value determined as follows:

$$weight = (1 - ambient) * \frac{1}{1 + \exp(-20 * distance)} + ambient$$

The calculated weight value exists within the range [ambient color, 1]. If the weight becomes ambient, the object represents the darkest area of shadow, and if it becomes 1, it shows its diffuse color.

Using this sigmoid function, we can generate the shadow color that changes naturally. The number 20 in sigmoid function expresses the degree of how rapidly shadow changes.

5.4. Sphere Specular

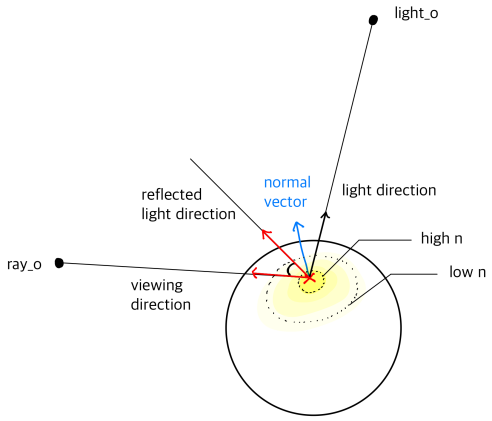


Figure 9. Visualization of specular reflection

The law of reflection states that a reflected ray of light emerges from the reflecting surface at the same angle to the surface normal as the incident ray, but on the opposing side of the surface normal in the plane formed by the incident and reflected rays.

The specular intensity is figured out through the formula written below:

$$\overrightarrow{light\_dir} = 2(\overrightarrow{normal} \cdot \overrightarrow{reflect})\overrightarrow{normal} - \overrightarrow{reflect}$$
$$specular = \left(\overrightarrow{viewing\ dir} \cdot \overrightarrow{reflected\ light\ dir}\right)^{n}$$

The normal vector represents the unit vector that orthogonal to the sphere surface. And the sphere point is extracted from depth map information, the same way as mentioned before.

The more the two directions – viewing direction and reflected light direction – coincide, the more the degree of specular intensity increases. And it differs to the viewing direction. As the value of N increases, the specular area becomes narrower, and as the value becomes smaller, the area becomes larger.

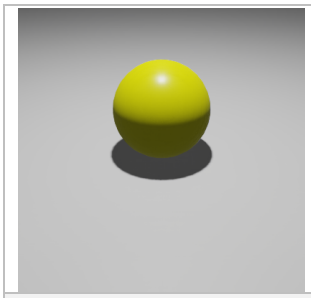
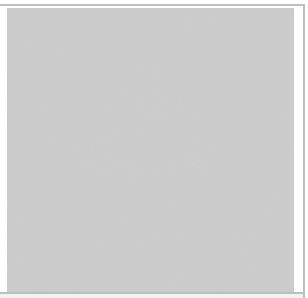
5.5. Optimization

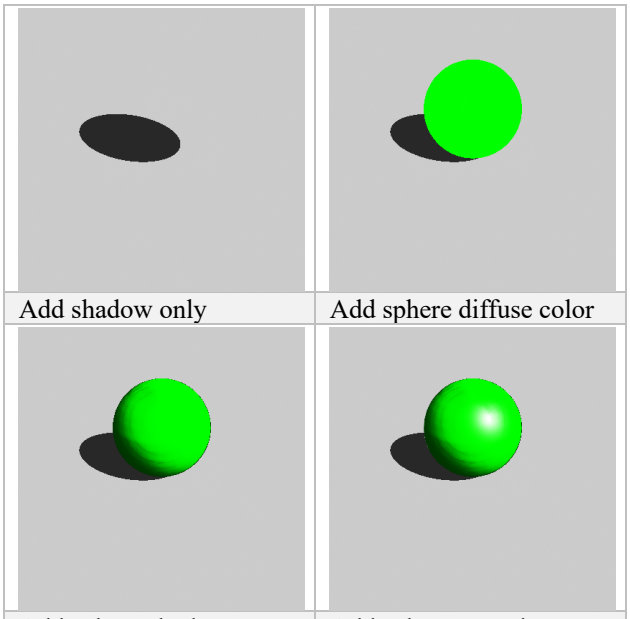
Our first approach was checking 3-dimensional inclusion relationship with all the discrete sample points of each ray. However, this approach should calculate all the points of the number of rays multiplied by the number of samples. Therefore, we changed our method by checking intersection of rays, which can be calculated with its direction and origin information. Thus, we could reduce its time complexity to the number of points times.

Also, we could effectively reduce its computational cost with the techniques using geometrical tricks such as projected plane, affine transformation etc. Additionally, by replacing the method of using loop with matrix operations, we derived effective calculations in code implementation process. As a result, our initial processing time which is about 3 hours is decreased to roughly 3 minutes per image. This is a result of including basic NeRF rendering process.

6. Result

For better understanding, we break our process into 5 steps and visualized them with following table. We started from the nothing but plane. First, we generated the plane shadow using affine transformation. Next, we identified the sphere region with the angle condition, and changed it to its diffuse color of the sphere. Lastly, we added the shadow to the sphere and specular color by using the depth map.

	
Basic NeRF output	Nothing

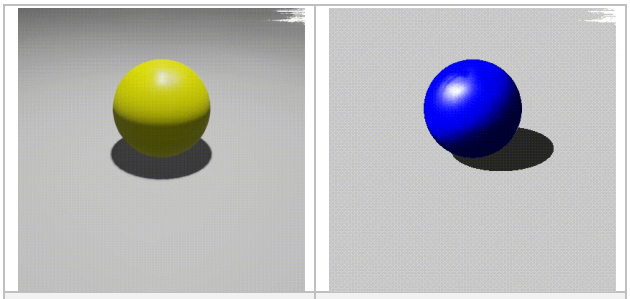


Add shadow only	Add sphere diffuse color
Add sphere shadow	Add sphere specular

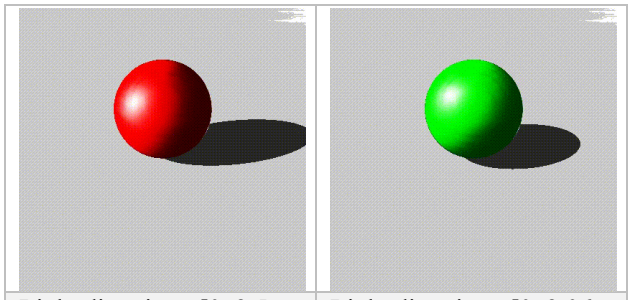
Figure 10. Progress of re-calculating RGB-map

The following table represents our integrated result. We produced three spheres with different diffuse color and light direction in same viewpoint. The result confirmed that the most of our works were acceptably accurate.

Notice that the size of specular is slightly larger and the color of plane shadow is darker than the original NeRF. It seems that there are some parametric differences between real NeRF and our result. But still we've shown the overall result of our project fits well with those of NeRF.



Light direction : [0, 0, 4], original NeRF result.	Light direction : [0, 1.5, 2.5]



Light direction : [0, 2.5, 1.5]	Light direction : [0, 2.06, 2.06]

Figure 11. Results with different light and diffuse color

7. Conclusion

From our manipulation of RGB function, we could produce the image of the object with the arbitrary viewing direction, light information, and diffuse color as we want. We also proposed various geometrical techniques that used to calculate the ray information effectively.

Still, it has some limitations. We evaluated our approach by modifying the RGB-map immediately. But the information that can be received through the RGB map is extremely limited and hard to calculate. In addition, our geometric tricks are highly relevant only to spheres, making them difficult to apply to other forms.

The follow-up study is required to generalize this whole process. If we check all the directions' inclusion relationship using the depth map, we will be able to modify our method to the arbitrary objects. We expect this kind of approaches could generalize our project for any object in the real world.

References

[1] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, 2020

[2] Can Wang, Menglei Chai, CLIP-NeRF: Text-and-Image Driven Manipulation of Neural Radiance Fields, 2021

[3] Richard Hartley, Multiple view geometry in computer vision, 2000