**Novel views of 3D scenes from 2D images using Neural Radiance Fields**

**1 Introduction**

Visualising images in real 3D structures is an intriguing way to convey an idea rather than simply displaying a series of images. Previously, this technique used 3D modelling software, which was time-consuming and involved more effort for each object. The process has been simplified due to the interference of AI (Neural Rendering). Novel views of 3D scenes using multiple 2D images are done by many neural rendering techniques like novel view synthesis, semantic photo manipulation etc. Despite the process being made simple by the usage of AI, there are still challenges involved because of complex geometry, lighting effects and perspective of images from different angles. But the concept of volume rendering through MLP makes it a lot easier [1].

Many applications, such as Video games, Virtual reality, Architecture design, E-commerce, Movies, Real estate, Sports, etc., strongly demand photorealistic 3D visualisation. One potential use case is the creation of virtual environments that look and feel like real-world locations. For example, creating a virtual city like a real city with detailed features including buildings, streets and so on can provide immersive experiences for simulated tours. These experiences feel more realistic and engaging.

Another use-case in architecture design is where neural rendering can generate photorealistic visualisations of buildings and other structures. This can be useful for a variety of purposes, including creating marketing materials, visualizing design concepts, and communicating ideas to clients or stakeholders.

In the film industry, neural rendering helps in the creation of realistic special effects. This could be used to create detailed, photorealistic visual effects for movies, such as explosions, fire, or other kinds of natural disasters.

**2 Brief reviews of methods**

**2.1 Neural 3D shape representations**

The optimization of deep networks that map XYZ coordinates to signed distance functions [2,3] or occupancy fields [4,5] results in the implicit representation of continuous 3D shapes as level sets. However, some limitations associated with these models, such as their requirement for access to ground truth 3D geometry

**2.2 View synthesis and image-based rendering**

Using simple light field sample interpolation techniques [6,7], photorealistic novel views can be reconstructed from a dense sampling of views.

Having predicted traditional geometry and appearance representations based on observed images, the computer vision and graphics communities have made significant progress for novel view synthesis with sparser view sampling.

A popular approach to representing diffuse scenes is mesh-based [8]

**2.3 Volume rendering with a non-convolutional deep network**

This idea uses the concept of volume rendering with a non-convolutional deep network known as a Multilayer perceptron (MLP). It takes the input as 5D coordinates (spatial location & viewing direction) and gives the output as colour & density. Using volume rendering along the camera rays in those coordinates’ projects colour and density at that position.[1]

In addition to the use of volume rendering to visualise images, positional encoding and hierarchical volume rendering are also used to optimise complex scenes

This report focuses mainly on the mathematics involved in **volume rendering.**

**3 Analysis of the method**

Volume rendering, also known as volume ray casting, is a technique that creates a 3D representation of data. It involves casting rays through the volume data and determining the colour and opacity of each voxel (3D pixel) along the way by mapping the data to a 3D grid and rendering the resulting volume as a 2D image. The colours and opacities of the voxels are combined to generate the final image.

Consider an experiment for volume rendering where we are passing laser beams with low divergence on a non-empty cylinder (assuming filled with some particles) on one end and a person/camera is viewing on another end as displayed in Figure 1. We are going to find the light beam intensity (the amount of light that reaches the viewer’s eye) passing through the cylindrical volume.

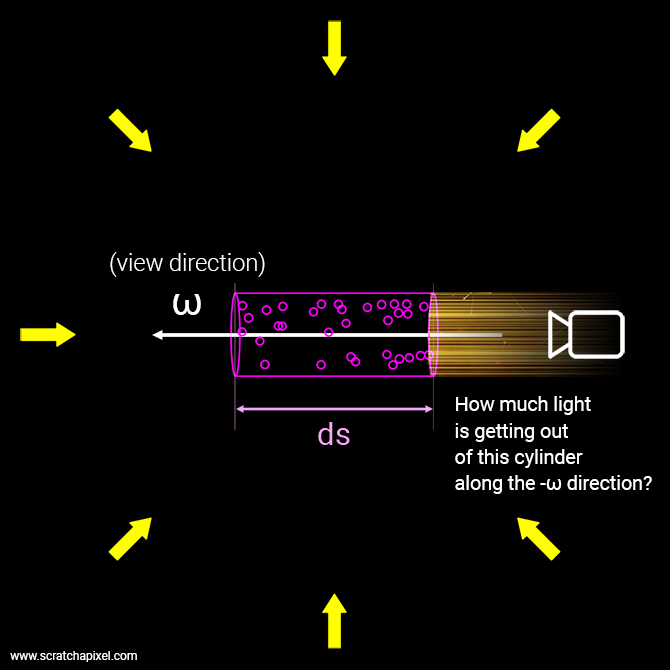


Fig. 1: experiment with a cylindrical tube with laser beams passing on one end and viewing on another end [12]

The light quantity is called radiance denoted as .

- incoming radiance.

- outgoing radiance.

- is the view direction.

*s* - light beam travelled through the medium

The light interacts in the medium within the cylinder in 4 different ways.

*Absorption*: the particles absorb the light reducing the intensity of radiance.

*Out-scattering*: the particles scatter the light in a random direction other than the viewing direction

*In-scattering*: the particles scatter light in the light beam direction

*Emission*: Electrons with high energy pass as photons. Some of them pass in the direction of light.

Absorption and out-scattering cause loss in radiance whereas in-scattering and emission increase light intensity.

*Absorption coefficient*:

- probability density that the light is absorbed by the volume per unit distance travelling through the medium.

Scattering coefficient:

- probability that the light is scattered by the volume per unit distance through the medium

*Extinction coefficient*:

Total loss in radiance caused by absorption and scattering.

**Beer-Lamber Law:**[9]

Considering Figure 2, the derivate of radiance at point x in the direction is written as

Rewriting the function with respect to s (the distance the light beam travelled through the medium). The derivative of L(s) with respect to s.

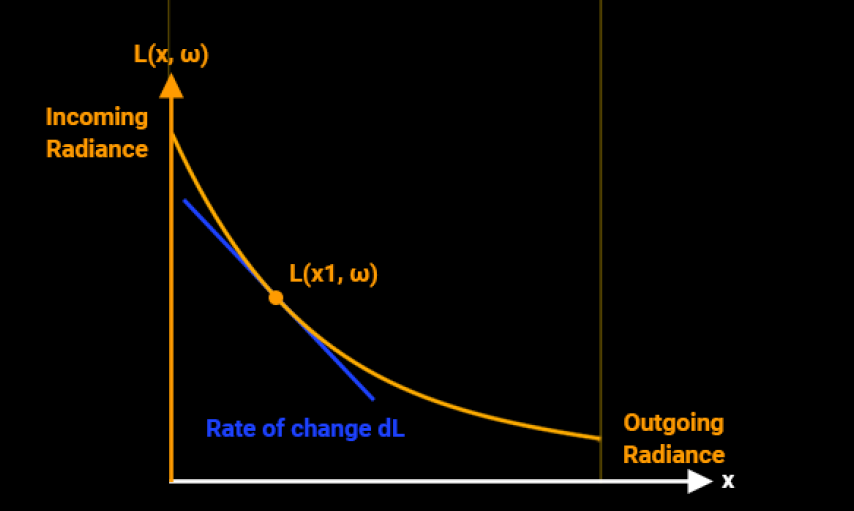


Fig. 2: The graph shows the plot of the function (for a given absorption coefficient) with respect to a particular point at x [12]

Applying integration [10] to find L(s),

Applying exponential to remove ln,

This equation is called Beer-Lambert equation. This works for a homogeneous medium (the composition is the same at all the points of the medium)

**Transmittance:**

It is the fraction of light that passes through the volume.

T calculated using Beer’s law,

- is the optical depth

For heterogenous medium, the extinction coefficient varies through a medium,

Integrating extinction coefficient along the rays,

d- distance travelled by ray through volume

Thus,

**Phase function:**

The in-scattering causes rays to travel in the light direction which is opposite of viewing direction. Phase function is the number of incoming rays travelling along the direction - also denoted as

Phase function is denoted as

**Radiative transfer equation:** [11]

Considering the phase equation, the change of radiance along the direction

where,

-- loss due to absorption and out-scattering

- in-scattering

Replacing with

using Radioactive Transfer equation standard form,

Multiplying by ,

Considering Derivative Product rule [10],

 , left-hand side of the equation multiplied by  similar to derivative computed by the Product Rule

Substituting the value of

We know,

Substituting left hand term with right hand term in equation (32),

Integrating both sides,

Converting to general equation,

After replacing p and q with and respectively, we get

Since,

Sub the T value,

The derived equation corresponds to **volume rendering.**

**4 Numerical example**

Suppose we have a volume of data with three voxels (3D pixels) that have the following values:

Voxel 1: Absorption coefficient = 0.2, Distance = 1.0

Voxel 2: Absorption coefficient = 0.4, Distance = 0.5

Voxel 3: Absorption coefficient = 0.6, Distance = 0.25

As per Equation (13) to calculate the transmittance at each voxel as follows:

Voxel 1:

T(x,y,z) = e^(-0.2 \* 1.0) = 0.8

Voxel 2:

T(x,y,z) = e^(-0.4 \* 0.5) = 0.6

Voxel 3:

T(x,y,z) = e^(-0.6 \* 0.25) = 0.5

The transmittance values can then be used to determine the colour and opacity of each voxel along a ray as it travels through the volume

For a detailed explanation of the remaining equations, please refer to the programming code in Appendix A [12].

**5 Discussion**

**5.1 Mathematical assumptions**

When we calculate transmittance for different materials, we assume that all materials have the same density. This means we assume that the amount of material present (density) in each voxel (3D pixel) will remain constant throughout the process.

In ray casting, the interaction of light with volume data is often simulated by modelling the scattering and absorption of light as it travels through the volume.

Ray casting often involves interpolating the values of the voxels (3D pixels) along the ray to determine the colour and opacity of each voxel. Linear interpolation is a common technique used for this purpose (uses straight-line relationships between the two points of data to make an estimation).

**5.2 Strength**

Unlike other models which uses MLP can take in a 3D spatial location and generate an implicit representation of the shape, such as the signed distance [13], this method involves computing the colour and opacity of each voxel (3D pixel) along a ray as it passes through the volume of a scene in order to create a 3D model. This method can generate these models with high detail and accuracy without requiring a large amount of storage space.

**5.3 Weakness**

It takes a long amount of time to complete the training of a model, usually at least 12 hours. Every model must be trained individually and there is no way of quickly generalising the training to multiple models without having to do the training for each one separately.

Instant neural graphics [14] addresses this drawback by processing the model quickly.

**5.4 Impact on the use case**

This method of volume rendering increases the accuracy of complex scenes. It optimises the parameters associated with a continuous 5D scene representation which minimizes the error associated with rendering a set of captured images.

Other methods that use signed distance functions [2] or occupancy fields [4] couldn’t achieve this level of accuracy.

**References**

1. Mildenhall B, Srinivasan PP, Tancik M, Barron JT, Ramamoorthi R, Ng R. Nerf: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM. 2021 Dec 17;65(1):99-106.
2. Jiang C, Sud A, Makadia A, Huang J, Nießner M, Funkhouser T. Local implicit grid representations for 3d scenes. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2020 (pp. 6001-6010).
3. Park JJ, Florence P, Straub J, Newcombe R, Lovegrove S. Deepsdf: Learning continuous signed distance functions for shape representation. InProceedings of the IEEE/CVF conference on computer vision and pattern recognition 2019 (pp. 165-174).
4. Genova K, Cole F, Sud A, Sarna A, Funkhouser T. Local deep implicit functions for 3d shape. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition 2020 (pp. 4857-4866).
5. Mescheder L, Oechsle M, Niemeyer M, Nowozin S, Geiger A. Occupancy networks: Learning 3d reconstruction in function space. InProceedings of the IEEE/CVF conference on computer vision and pattern recognition 2019 (pp. 4460-4470).
6. Levoy M, Hanrahan P. Light field rendering. InProceedings of the 23rd annual conference on Computer graphics and interactive techniques 1996 Aug 1 (pp. 31-42).
7. Gortler SJ, Grzeszczuk R, Szeliski R, Cohen MF. The lumigraph. SIGGRAPH’96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.
8. Waechter M, Moehrle N, Goesele M. Let there be color! Large-scale texturing of 3D reconstructions. InEuropean conference on computer vision 2014 Sep 6 (pp. 836-850). Springer, Cham.
9. Swinehart DF. The beer-lambert law. Journal of chemical education. 1962 Jul;39(7):333.
10. Mohammed Hadi, Martin Rudorfer. Cs4755: Mathematics for AI, Calculus
11. Chandrasekhar S. Radiative transfer. Courier Corporation; 2013 Apr 15.
12. Scratchapixel, 2009, “Volume rendering for developers”. Available at: https://www.scratchapixel.com/lessons/3d-basic-rendering/volume-rendering-for-developers/volume-rendering-3D-density-field (Accessed: 12 Dec 2022)
13. Curless B, Levoy M. A volumetric method for building complex models from range images. In Proceedings of the 23rd annual conference on Computer graphics and interactive techniques 1996 Aug 1 (pp. 303-312).
14. Müller T, Evans A, Schied C, Keller A. Instant neural graphics primitives with a multiresolution hash encoding. arXiv preprint arXiv:2201.05989. 2022 Jan 16.

**Appendix A - Programming example explaining volume rendering**

**A.1 Rendering volume on uniform background:**

vec3 background\_color {xr, xg, xb};

float sigma\_a = 0.1;  //absorption coefficient

float distance = 10;

**// Calculated using Beer’s law - Equation (23),**

float T = exp(-distance \* sigma\_a);

vec3 background\_color\_through\_volume = T \* background\_color;

**A.2 Rendering volume for heterogenous background:**

float eval\_density(const vec3& p)

{

   float freq = 1;

   return (1 + noise(p.x \* freq, p.y \* freq, p.z \* freq)) \* 0.5;

}

vec3 integrate(

   const vec3& ray\_orig,

   const vec3& ray\_dir,

   const std::vector<std::unique\_ptr<Sphere>>& spheres)

{

   const float step\_size = 0.1;

   float sigma\_a = 0.5;   //absorption coefficient

   float sigma\_s = 0.5;   //scattering coefficient

   float sigma\_t = sigma\_a + sigma\_s;  //extinction coefficient

   float g = 0;

   uint8\_t d = 2;

   int ns = std::ceil((isect.t1 - isect.t0) / step\_size);

   float stride = (isect.t1 - isect.t0) / ns;

   vec3 light\_dir{ -0.315798, 0.719361, 0.618702 };

   vec3 light\_color{ 20, 20, 20 };

   float transparency = 1;  //initialize transmission to 1 (fully transparent)

   vec3 result{ 0 };  //initialize volumetric sphere color to 0

   // The main ray-marching loop (forward, march from t0 to t1)

   for (int n = 0; n < ns; ++n) {

       // Jittering the sample position

       float t = isect.t0 + stride \* (n + distribution(generator));

       vec3 sample\_pos = ray\_orig + t \* ray\_dir;

Evaluate the density at the sample location (space varying density)

float density = eval\_density(sample\_pos);

       float sample\_attenuation = exp(-step\_size \* density \* sigma\_t);

       transparency \*= sample\_attenuation;

       // In-scattering.

       IsectData isect\_light\_ray;

       if (density > 0 &&

           hit\_sphere->intersect(sample\_pos, light\_dir, isect\_light\_ray) &&

           isect\_light\_ray.inside) {

           size\_t num\_steps\_light = std::ceil(isect\_light\_ray.t1 / step\_size);

           float stide\_light = isect\_light\_ray.t1 / num\_steps\_light;

           float tau = 0;

Ray-march along the light ray. Store the density values in the tau variable.

for (size\_t nl = 0; nl < num\_steps\_light; ++nl) {

               float t\_light = stide\_light \* (nl + 0.5);

               vec3 light\_sample\_pos = sample\_pos + light\_dir \* t\_light;

               tau += eval\_density(light\_sample\_pos);

           }

           float light\_ray\_att = exp(-tau \* stide\_light \* sigma\_t);

**// Calculating using Volume rendering equation - Equation (40),**

           result += light\_color \*         //light colour

                     light\_ray\_att \*       //light ray transmission value

                     phaseHG(-ray\_orig, light\_dir, g) \*  //phase function

                     sigma\_s \*             //scattering coefficient

                     transparency \*       //ray current transmission value

                     stride \*             //dx in our Riemann sum

                     density;             //volume density at sample location

       }

       // Russian roulette

       if (transparency < 1e-3) {

           if (distribution(generator) > 1.f / d)

               break;

           else

               transparency \*= d;

       }

   }

   // combine background colour and volumetric sphere colour

   return background\_color \* transparency + result;

}