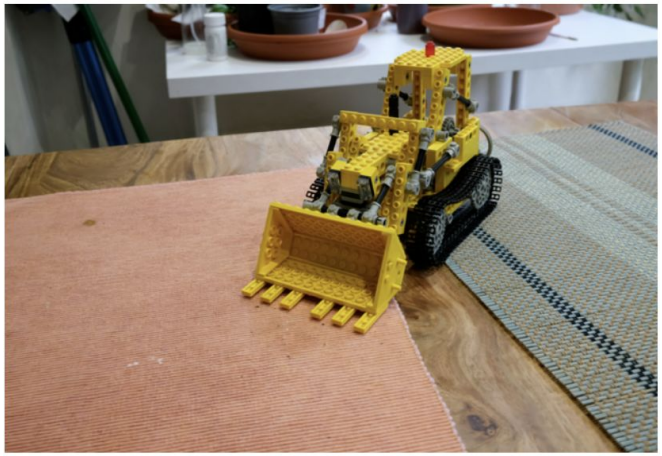
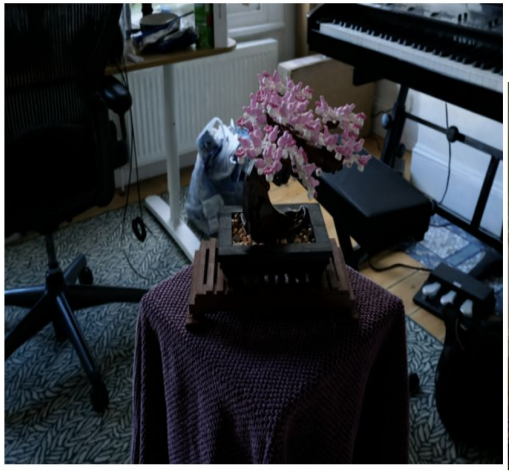
Neural Rendering & Neural Radiance Fields(NeRFs)

Arslan Artykov, PhD Candidate @ ENPC

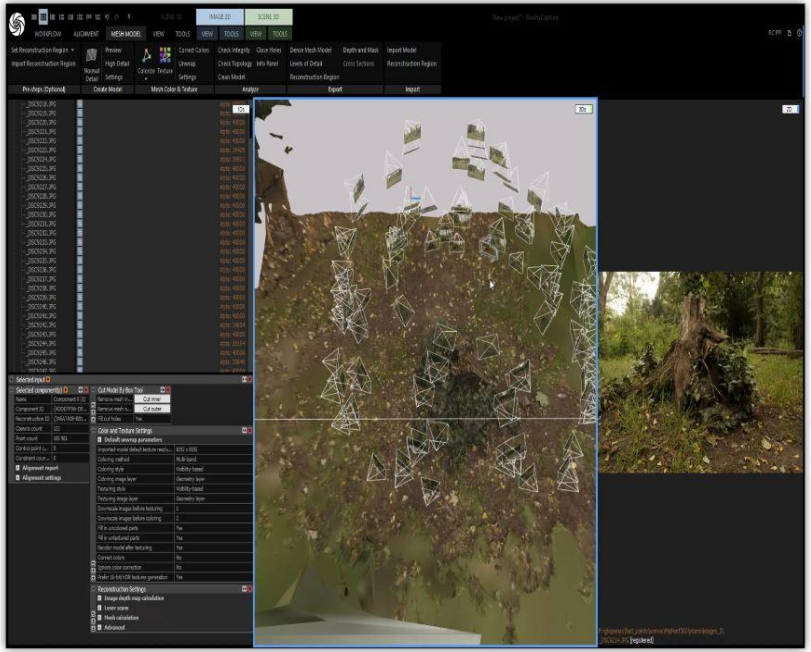




Novel View Synthesis

- Step 1: use the images to calibrate cameras (sparse scene geometry)

[…]



Screen capture from RealityCapture

Novel View Synthesis

- Step 2: use the images to build a dense 3D scene representation

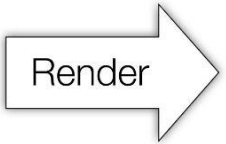
Kerbl, Kopanas et al. “3D Gaussian Splatting for Real-Time Radiance Field Rendering” SIGGRAPH 2023

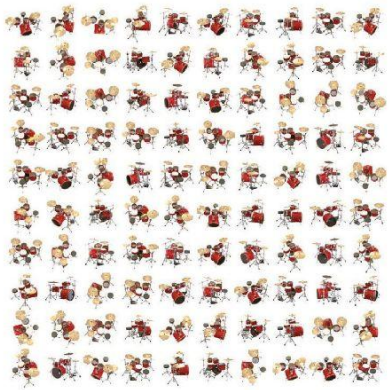
Forward Graphics (Rendering)

- Given a 3D scene specification and Cameras, yield images

3D Scene

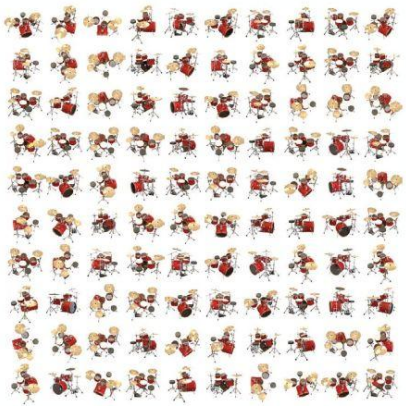
Camera Poses





Images

Inverse Graphics



Reconstruct

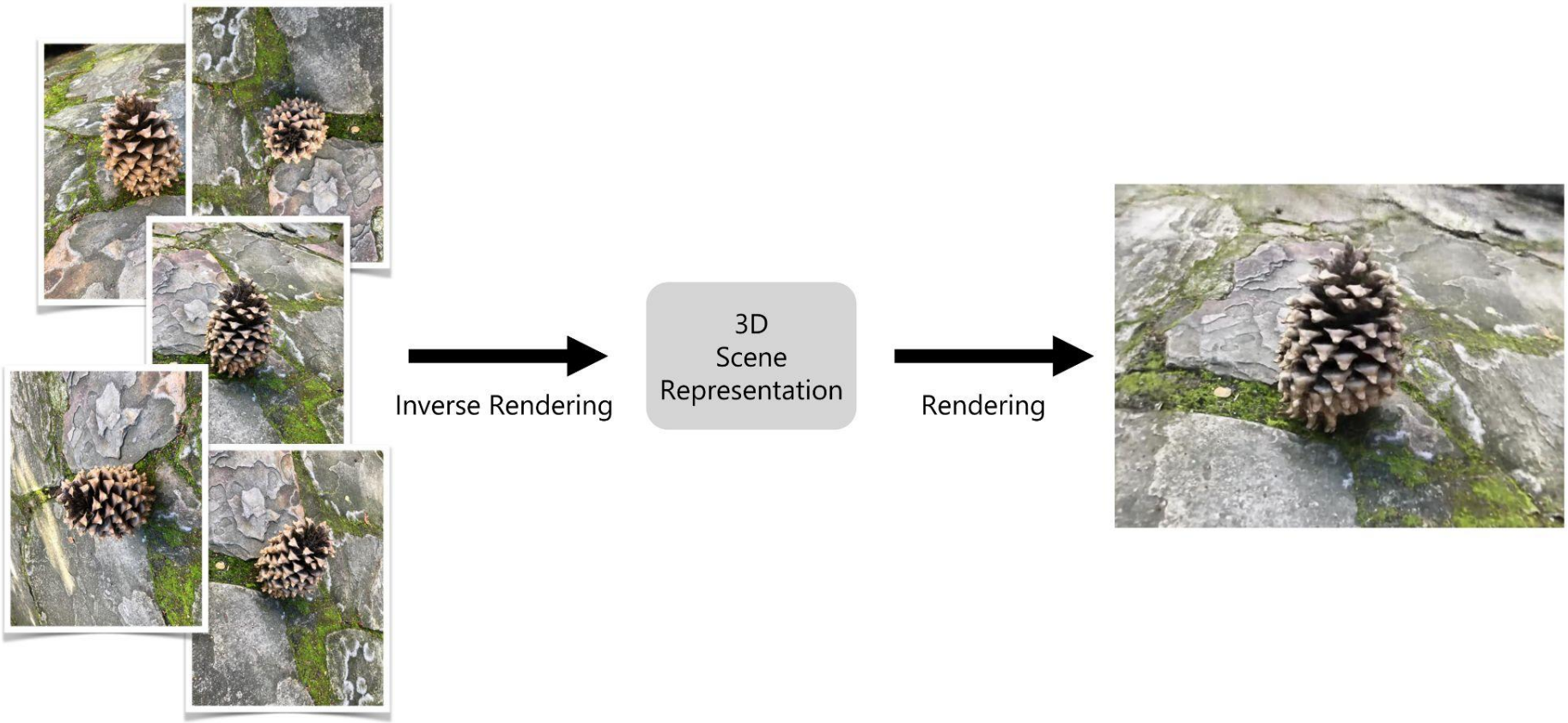
Images → Camera Poses

Assume Given

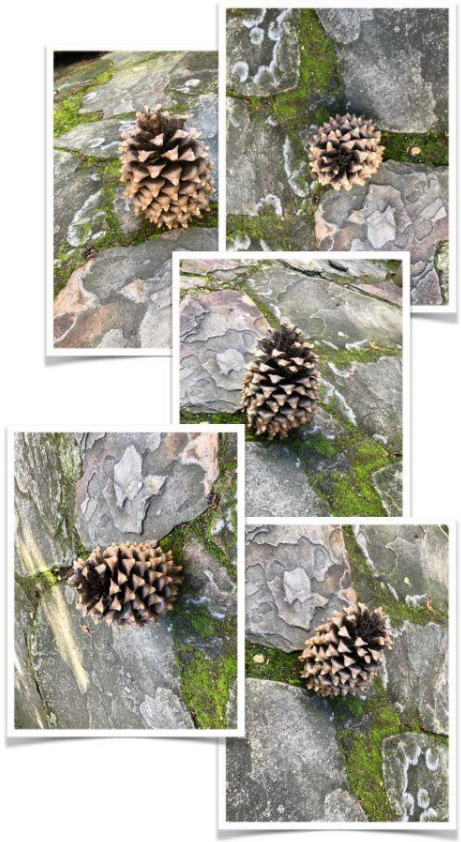
SLAM / BundleAdjustment
(OrbSLAM, COLMAP)

3D Scene

Put All Together…



Neural Radiance Fields (NeRF) as an approach to inverse rendering



Inverse Rendering

Neural Radiance Field

Rendering

Deep learning for 3D reconstruction

- Previously: we reconstruct geometry by running stereo or multi-view stereo on a set of images
  - "Classical" approach

- How can we leverage powerful tools of deep learning?
  - Deep neural networks
  - GPU-accelerated stochastic gradient descent

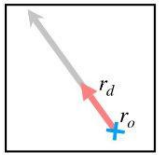
NeRF and related methods – Key ideas

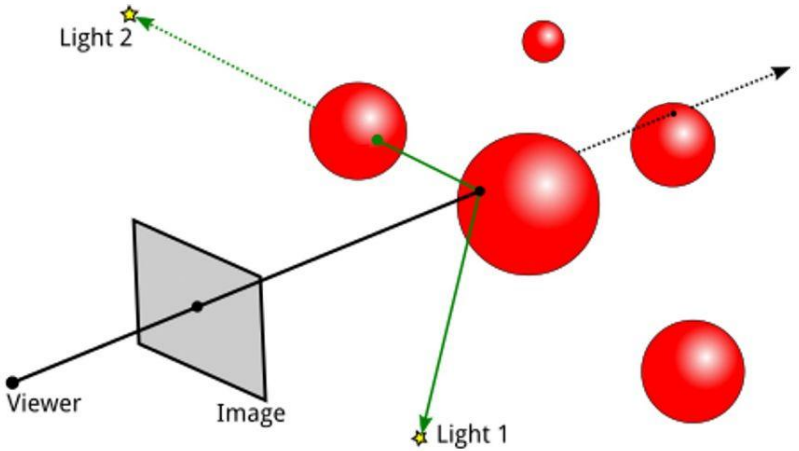
- We need to create a loss function and a scene representation that we can optimize using gradient descent to reconstruct the scene
- *Differentiable rendering*

Rendering (Signature)

- Given
  - Scene: {lights, materials, geometry, …}

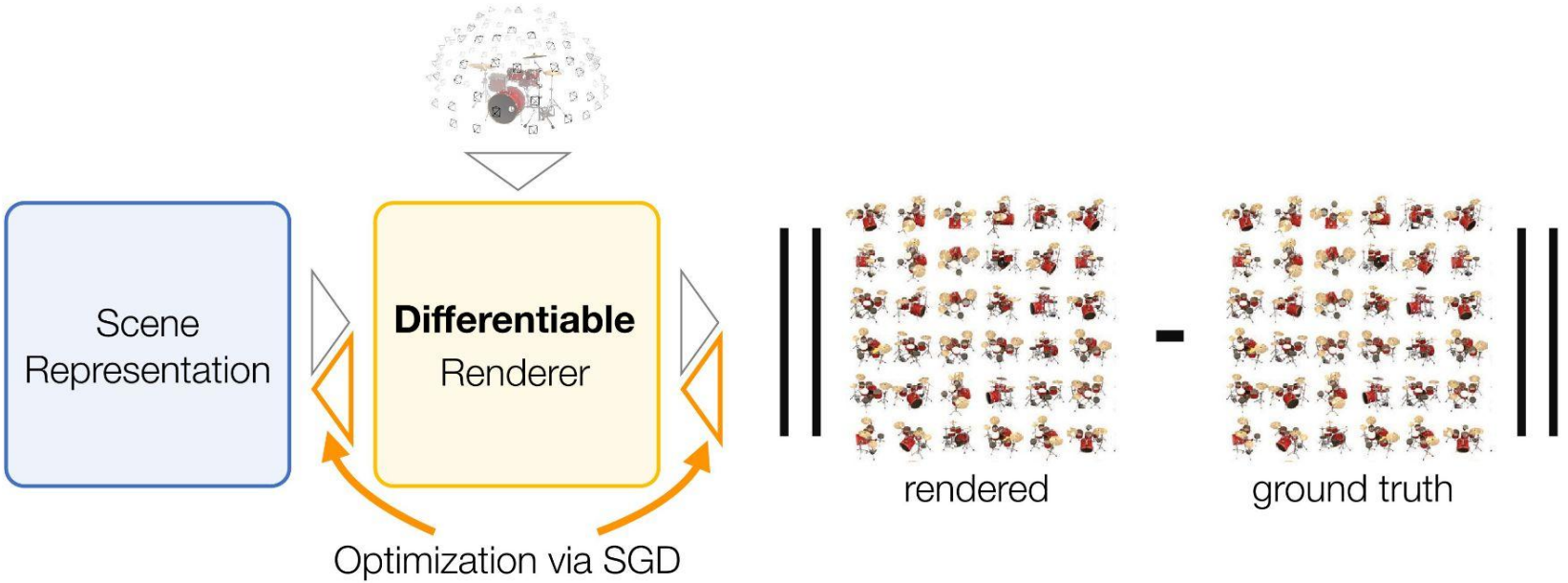
  - Camera Ray (origin + direction)





- Generate
  - ■ color of the ray/pixel

Differentiable Rendering



Given an observable variable (pixel colors), we will build a differentiable forward model that we then use to estimate unobserved (latent) variables (geometry, appearance)

NeRF == Differentiable Rendering with a **Neural Volumetric Representation**

Neural Volumetric Rendering

querying the radiance value
along rays through 3D space

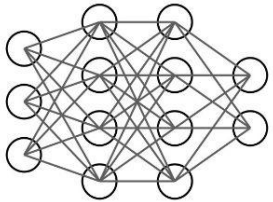
What color?

Neural Volumetric Rendering

continuous, differentiable
rendering model without
concrete ray/surface intersections

Neural Volumetric Rendering

using a neural network as a scene representation, rather than a voxel grid of data

(x, y, z) →  → Scene properties

Multi-layer Perceptron (Neural Network)

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

ECCV 2020

Ben Mildenhall*

UC Berkeley

Pratul Srinivasan*

UC Berkeley

Google

Matt Tancik*

UC Berkeley

Jon Barron

Google Research

Google

Ravi Ramamoorthi

UC San Diego

UC San Diego

Ren Ng

UC Berkeley

NeRF Overview

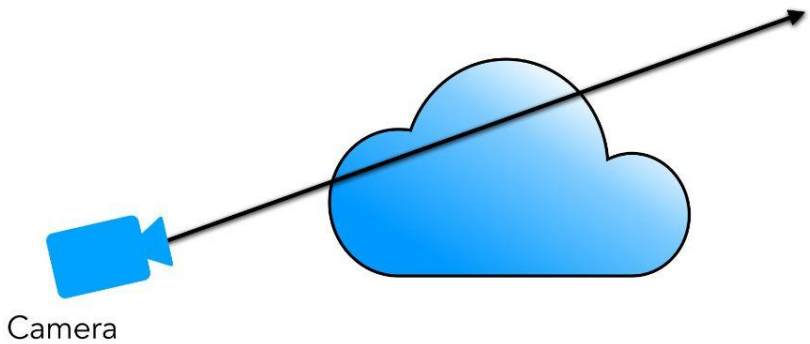
- Volumetric rendering
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)

NeRF Overview

- Volumetric rendering
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)

Volume Rendering

- Scene is a cloud of tiny colored particles



Volume Rendering Digest (for NeRF)

Andrea Tagliasacchi[1,2] Ben Mildenhall[1]
[1]Google Research [2]Simon Fraser University

Neural Radiance Fields [3] employ simple volume rendering as a way to overcome the challenges of differentiating through ray-triangle intersections by leveraging a probabilistic notion of visibility. This is achieved by assuming the scene is composed by a cloud of light-emitting particles whose density changes in space (in the terminology of physically-based rendering, this would be described as a volume with absorption and emission but no scattering [4, Sec 11.1]. In what follows, for the sake of exposition simplicity, and without loss of generality, we assume the emitted light *does not* change as a function of view-direction. This technical report is a condensed version of previous reports [1, 2], but rewritten in the context of NeRF, and adopting its commonly used notation[1].

Transmittance. Let the density field $\sigma(\mathbf{x})$, with $\mathbf{x}\in\mathbb{R}^3$ indicate the differential likelihood of a ray hitting a particle (i.e. the probability of hitting a particle while travelling an infinitesimal distance). We reparameterize the density along a given ray $\mathbf{r}=(\mathbf{o},\mathbf{d})$ as a scalar function $\sigma(t)$, since any point $\mathbf{x}$ along the ray can be written as $\mathbf{r}(t)=\mathbf{o}+t\mathbf{d}$. Density is closely tied to the transmittance function $\mathcal{T}(t)$, which indicates the probability of a ray traveling over the interval $[0,t)$ without hitting any particles. Then the probability $\mathcal{T}(t+dt)$ of *not* hitting a particle when taking a differential step dt is equal to $\mathcal{T}(t)$, the likelihood of the ray reaching t, times $(1-dt\cdot\sigma(t))$, the probability of not hitting anything during the step:

$$\mathcal{T}(t+dt) = \mathcal{T}(t)\cdot(1-dt\cdot\sigma(t)) \tag{1}$$

$$\frac{\mathcal{T}(t+dt)-\mathcal{T}(t)}{dt} \equiv \mathcal{T}'(t) = -\mathcal{T}(t)\cdot\sigma(t) \tag{2}$$

This is a classical differential equation that can be solved as follows:

$$\mathcal{T}'(t) = -\mathcal{T}(t)\cdot\sigma(t) \tag{3}$$

$$\frac{\mathcal{T}'(t)}{\mathcal{T}(t)} = -\sigma(t) \tag{4}$$

$$\int_a^b \frac{\mathcal{T}'(t)}{\mathcal{T}(t)}\,dt = -\int_a^b \sigma(t)\,dt \tag{5}$$

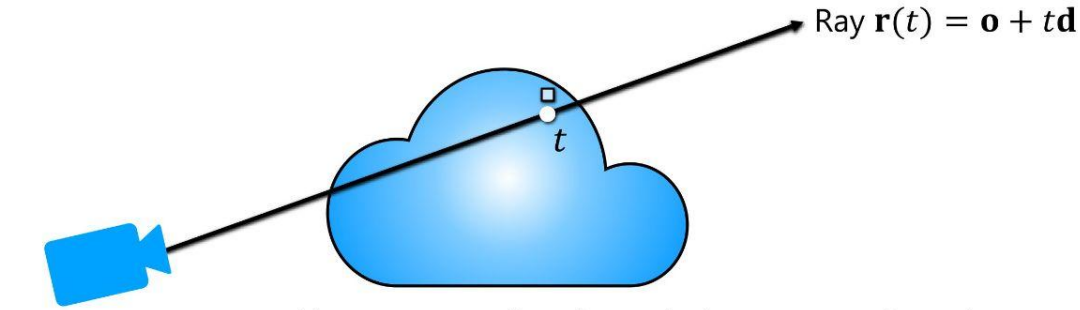
$$\log\mathcal{T}(t)\big|_a^b = -\int_a^b \sigma(t)\,dt \tag{6}$$

$$\mathcal{T}(a\rightarrow b) \equiv \frac{\mathcal{T}(b)}{\mathcal{T}(a)} = \exp\left(-\int_a^b \sigma(t)\,dt\right) \tag{7}$$

where we define $\mathcal{T}(a\rightarrow b)$ as the probability that the ray travels from distance a to b along the ray without hitting a particle, which is related to the previous notation by $\mathcal{T}(t)=\mathcal{T}(0\rightarrow t)$.

Tagliasacchi and Mildenhall “Volume Rendering Digest (for NeRF)” arXiv 2022

Volumetric Formulation for NeRF

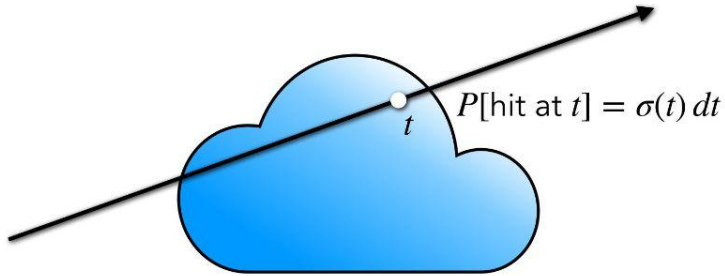


Consider a ray traveling through the scene, and a point at distance t along this ray. We look up its color $\mathbf{c}(t)$, and its opacity (alpha value) $\alpha(t)$

Volumetric Density

- Probability that ray stops in a small interval around t is $\sigma(t)\,dt$

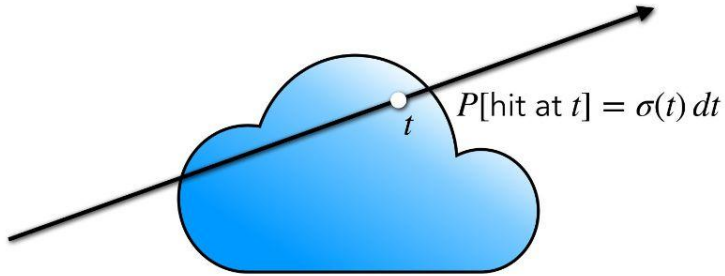
$\sigma(t)$ is called Volume[tric] Density



Scene Representation

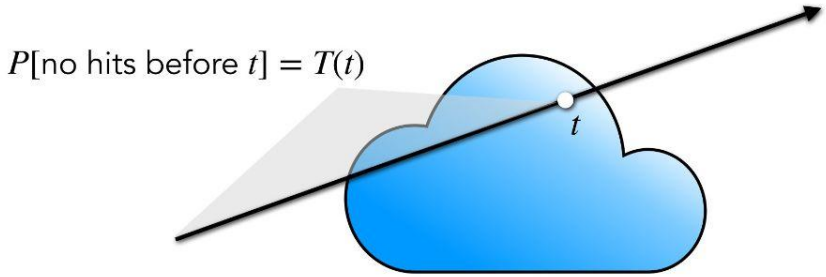
- Our 3D scene representation is therefore a field:

$$\Phi : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \times \mathbb{R}^+; \quad \Phi(\mathbf{x}) = (\sigma, \mathbf{c})$$



Occlusion Modelling

- Should we retrieve the color $\mathbf{c}(t)$? …only if position $\mathbf{r}(t)$ is not occluded
  - Transmittance $T(t)$ is the probability of no particles hit in $[0, t)$ range

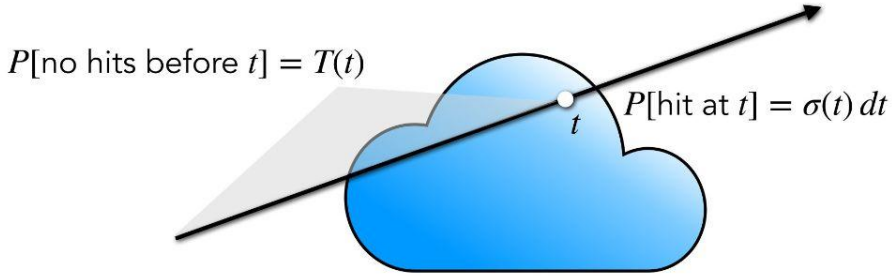


Relating $\sigma(t)$ to $T(t)$

- Hit probabilities are statistically independent along ray

$$P[\text{no hit before } t + dt] = P[\text{no hit before } t] \cdot P[\text{no hit at } t]$$

$$T(t + dt) = T(t) \cdot (1 - \sigma(t)\, dt)$$



Transmittance

$$T(0 \rightarrow t) := T(t) = \exp\left(-\int_0^t \sigma(u)\, du\right)$$

No hits before t is equal to integral over density up until t.

$(1\text{-}T(t))$: Opacity

$$(1\text{-}T(t))' = T(t) \cdot \sigma(t)$$

Volume Rendering

$$C = \int_0^\infty T(t) \cdot \sigma(t) \cdot \mathbf{c}(t)\, dt \quad \text{where} \quad T(t) = \exp\left(- \int_0^t \sigma(u)\, du \right)$$

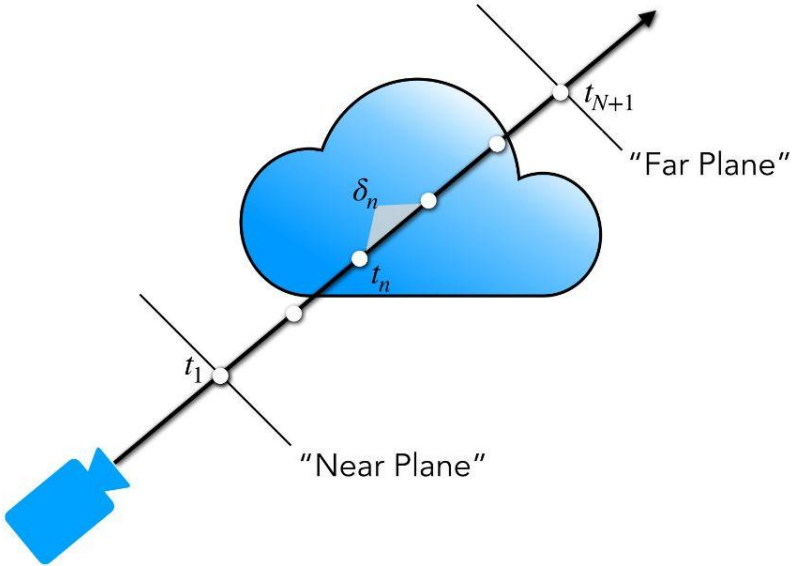
- How do we solve this?

- Discretize the nested integral!

Approximating the Integral

- Split the ray up into N segments with endpoints $\{t_1, t_2, \ldots, t_{N+1}\}$
- Segment length is $\delta_i = t_{i+1} - t_i$
  - warning: non-necessarily uniform!

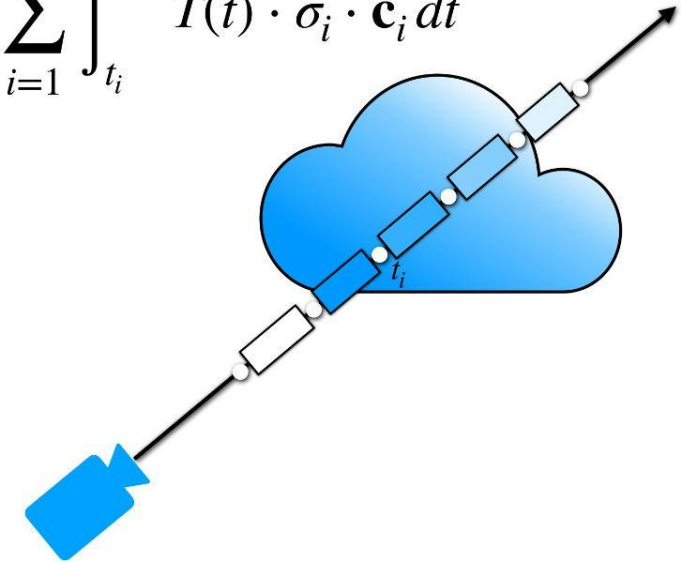
piecewise constant density
≠
piecewise constant transmittance

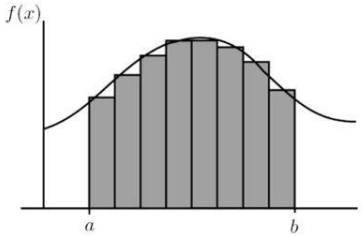


Approximating the Nested Integral

- Assume volume density/color are constant within interval (i.e. Reimann)
- Warning: constant density ≠ constant transmittance!

$$C = \int_0^{\infty} T(t) \cdot \sigma(t) \cdot \mathbf{c}(t)\, dt \approx \sum_{i=1}^{n} \int_{t_i}^{t_{i+1}} T(t) \cdot \sigma_i \cdot \mathbf{c}_i\, dt$$





Forward Model - NeRF

$$C = \int_0^{\infty} T(t) \cdot \sigma(t) \cdot \mathbf{c}(t)\, dt \approx \sum_{i=1}^{n} \int_{t_i}^{t_{i+1}} T(t) \cdot \sigma_i \cdot \mathbf{c}_i\, dt$$

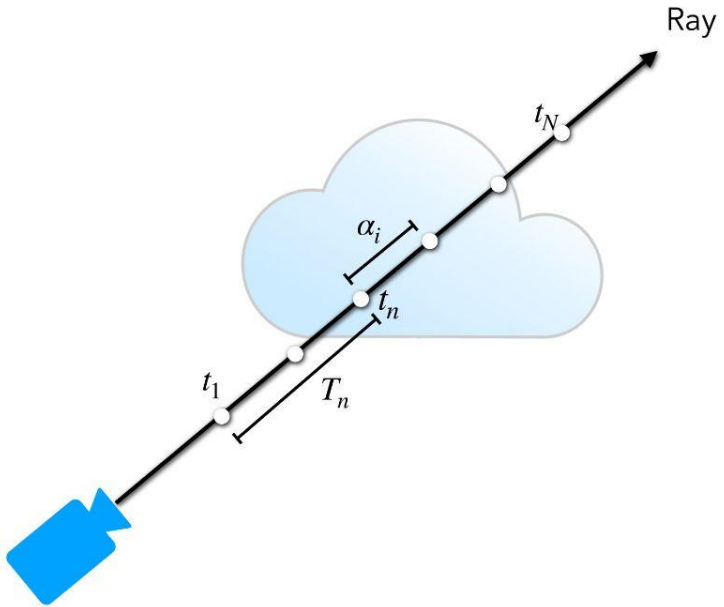
get rid of the inner integral…

NeRF as Alpha Blending

$$C = \sum_{n=1}^{N} T_n \cdot \underbrace{(1 - \exp(-\sigma_n \delta_n))}_{\alpha_n \equiv \text{opacity}} \cdot \mathbf{c}_n \quad \text{where} \quad T_n = \exp\left(\sum_{k=1}^{n-1} - \sigma_k \delta_k \right)$$

$$C = \sum_{n=1}^{N} T_n \cdot \alpha_n \cdot \mathbf{c}_n, \text{ where}$$

$$T_n = \prod_{k=1}^{n-1} (1 - \alpha_k) \ \ldots \text{ segment occlusion}$$



Volume rendering estimation: integrating color along a ray

Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

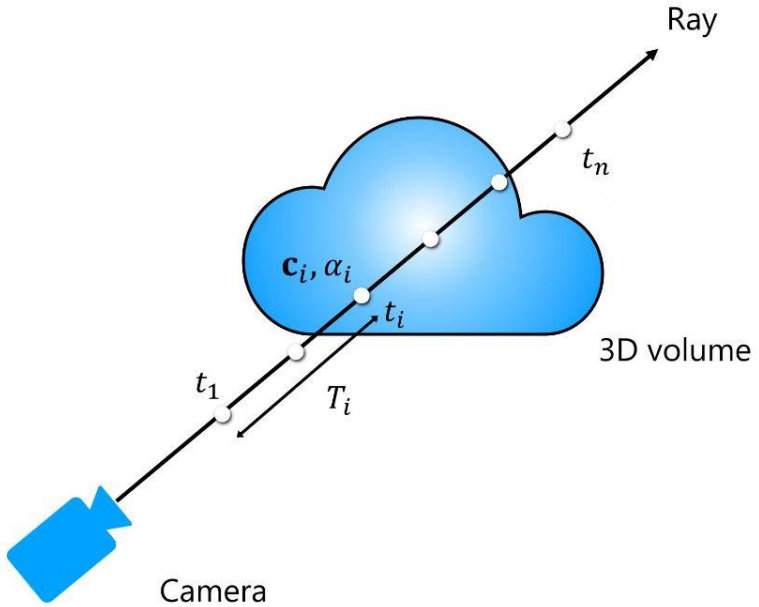
final rendered color along ray

weights

colors

How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$



Computing the color for a set of rays through the pixels of an image yields a rendered image

Volume rendering estimation: integrating color along a ray

Rendering model for ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$:

$$\mathbf{c} \approx \sum_{i=1}^{n} T_i \alpha_i \mathbf{c}_i$$

final rendered color along ray

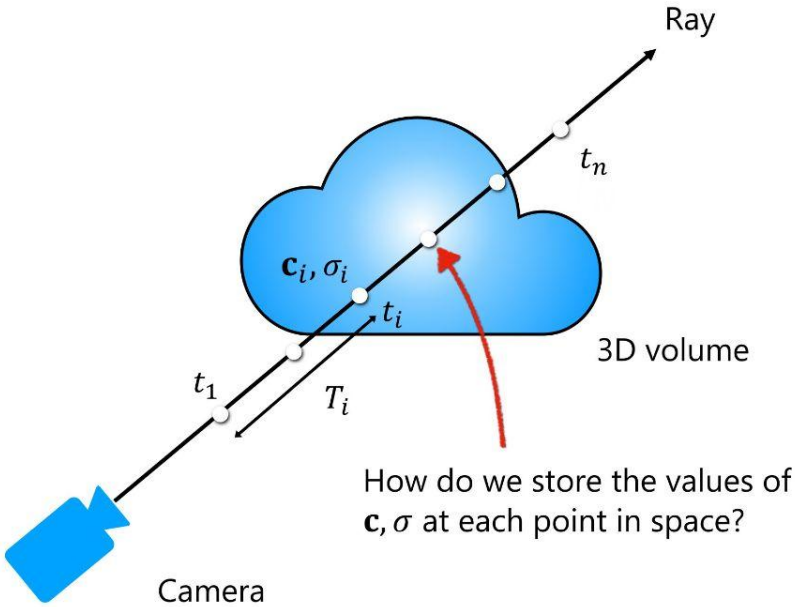
weights

colors

How much light is blocked earlier along ray:

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha_j)$$

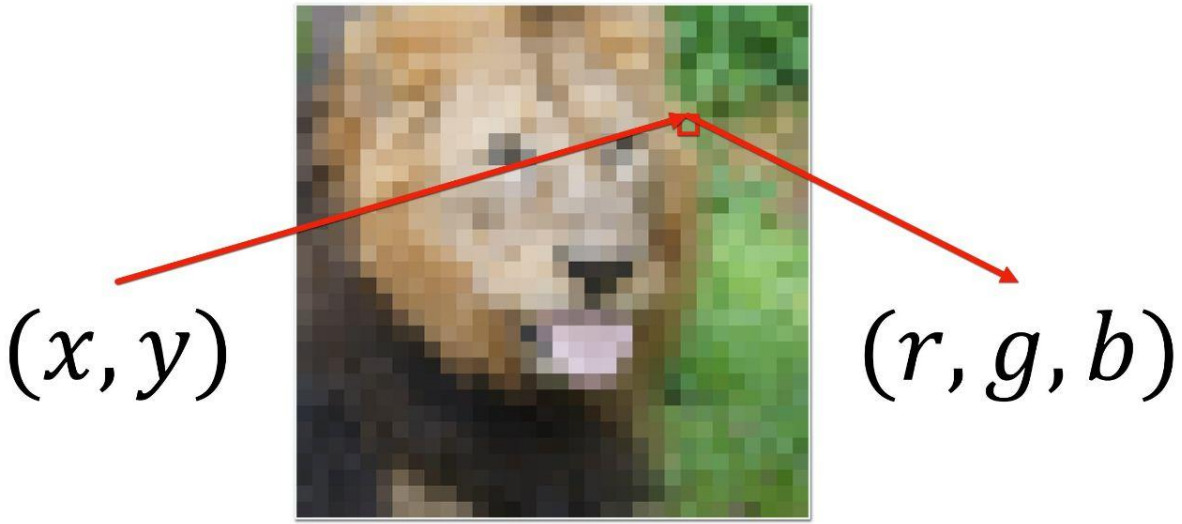
Computing the color for a set of rays through the pixels of an image yields a rendered image



NeRF Overview

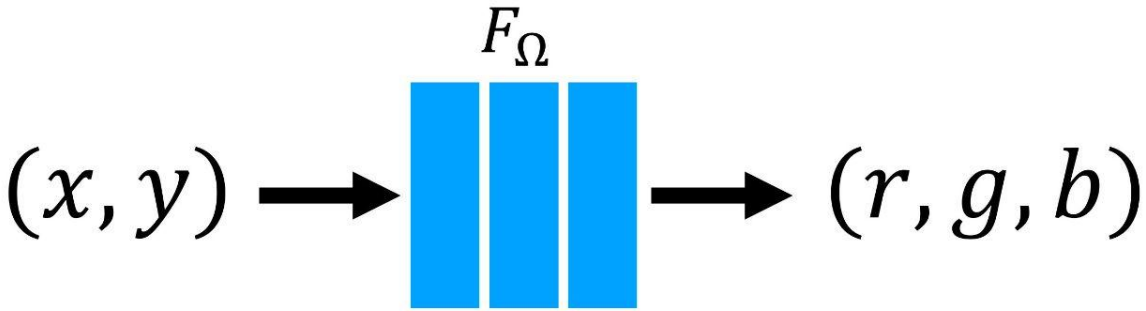
- Volumetric rendering
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)

Toy problem: storing 2D image data



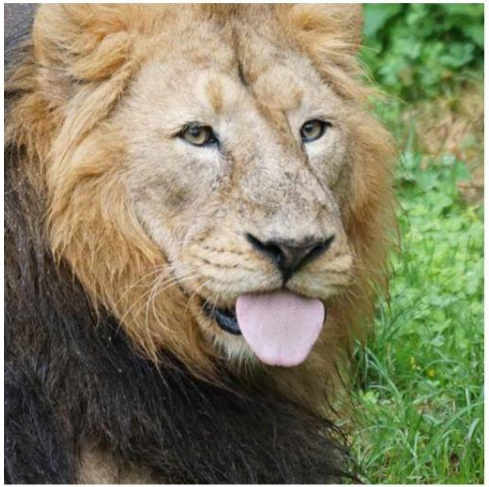
Usually we store an image as a
2D grid of RGB color values

Toy problem: storing 2D image data



What if we train a simple fully-connected network (MLP) to do this instead?

Naive approach fails!



Ground truth image

Neural network output fit
with gradient descent

Problem:

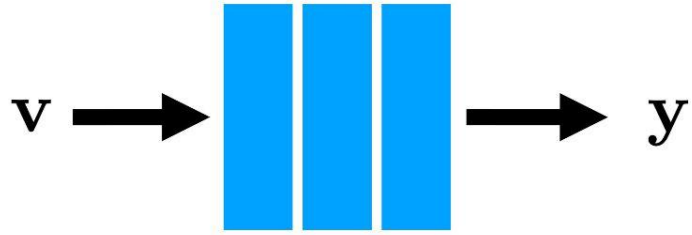
"Standard" coordinate-based MLPs cannot represent high frequency functions

Spectral Bias: Neural networks are biased to fit lower frequency signals.

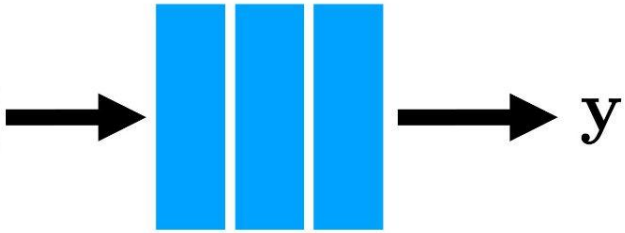
Solution:

Pass input coordinates through a high frequency mapping first

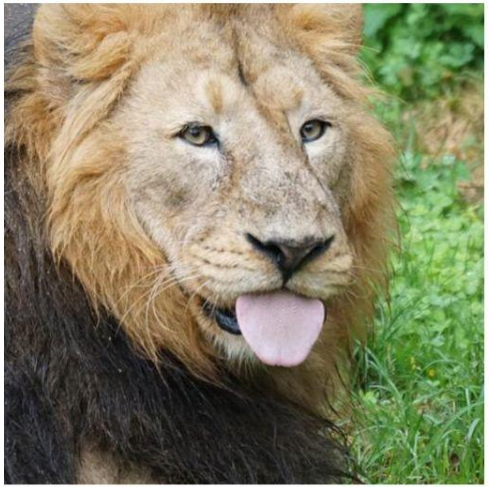
Example mapping: “positional encoding”



$$\begin{pmatrix} \sin(\mathbf{v}), \cos(\mathbf{v}) \\ \sin(2\mathbf{v}), \cos(2\mathbf{v}) \\ \sin(4\mathbf{v}), \cos(4\mathbf{v}) \\ \dots \\ \sin(2^{L-1}\mathbf{v}), \cos(2^{L-1}\mathbf{v}) \end{pmatrix}$$

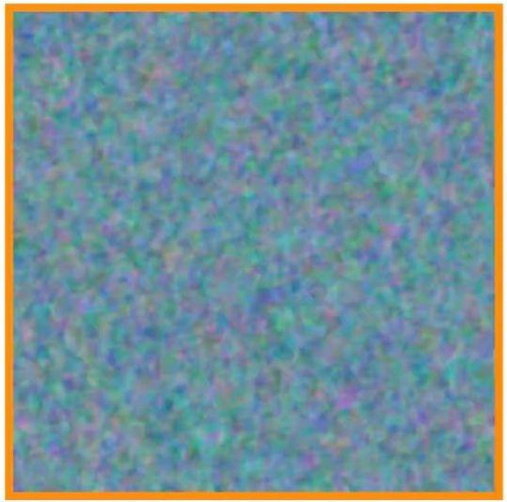


Problem solved!



Ground truth image

Neural network output without high frequency mapping



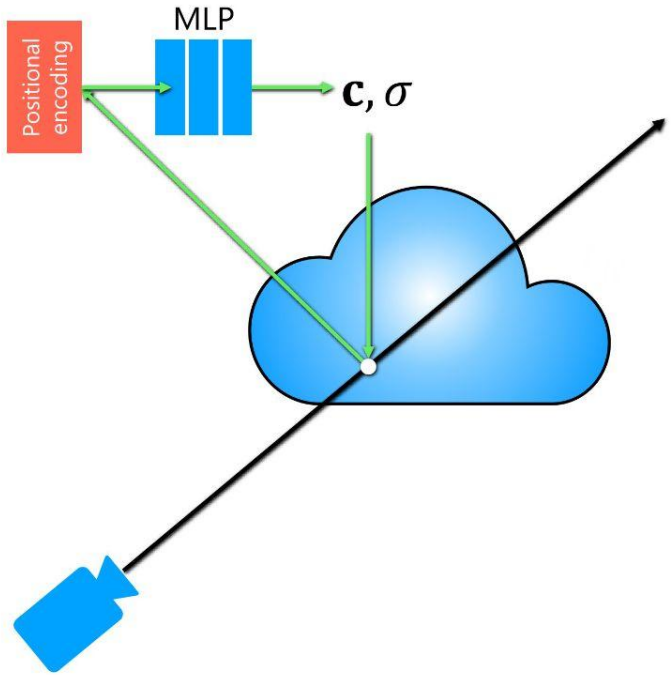
Neural network output with high frequency mapping

NeRF Overview

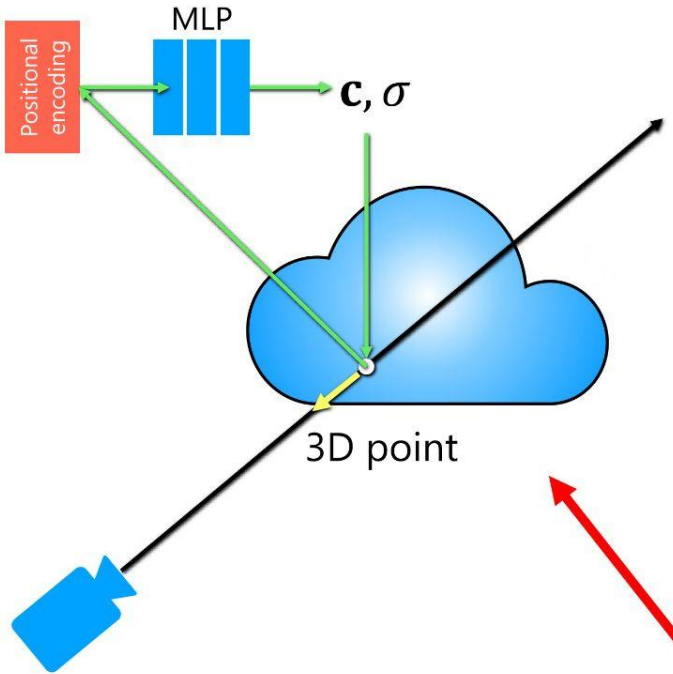
- Volumetric rendering
- Neural networks as representations for spatial data
- Neural Radiance Fields (NeRF)

NeRF = volume rendering + coordinate-based network

How do we store the values of $\mathbf{c}, \sigma$ at each point in space

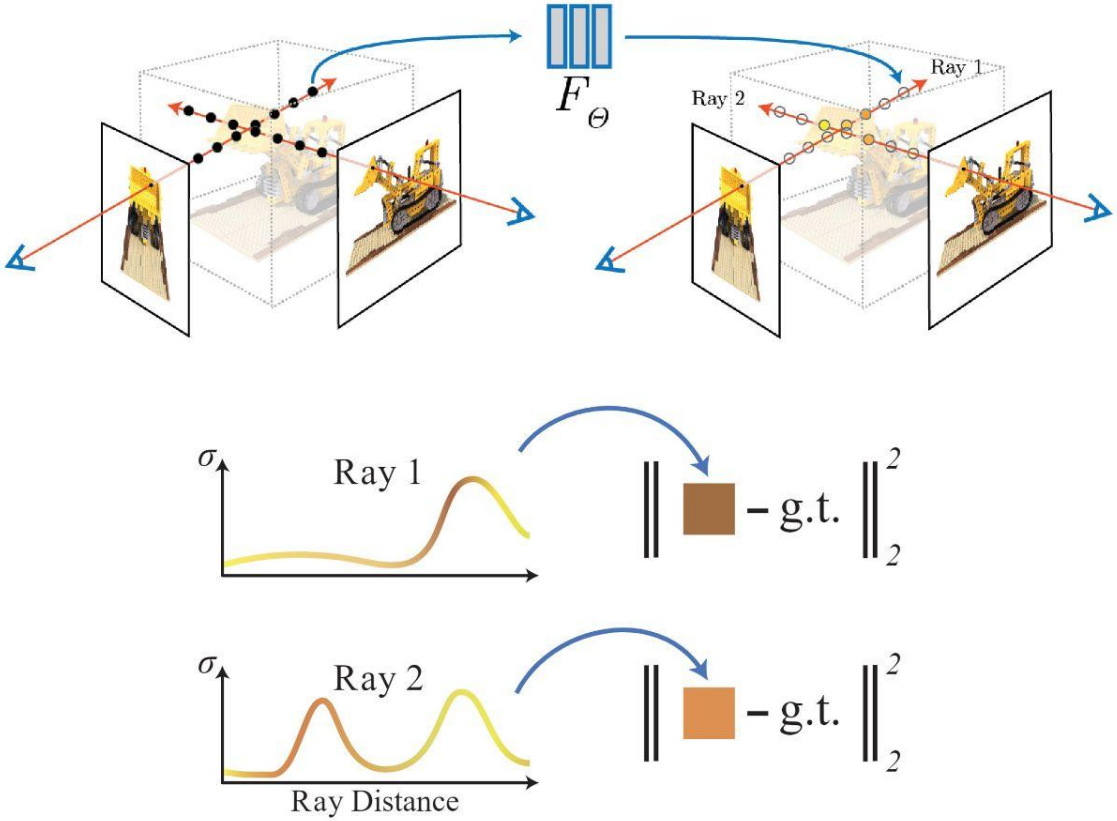


Extension: view-dependent field

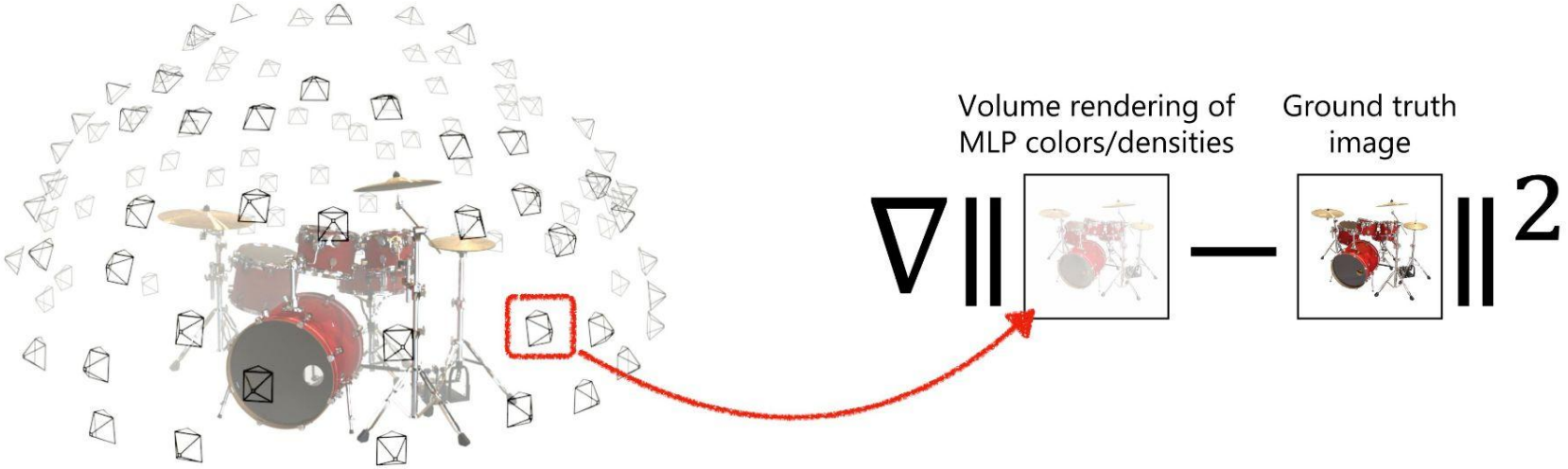


Include the ray direction in the input to the MLP → allows for capturing and rendering view-dependent effects (e.g., shiny surfaces)

Putting it all together



Train network using gradient descent to reproduce all input views of scene



NeRF Requires(incomplete list):

- Limited to either bounded or forward facing scenes
- Pixel-perfect camera calibration(e.g. COLMAP)
- Scene to be completely static
- Hundreds of images per scene(overfitting)
- Long training time(~1 day/scene)
- Real-time rendering is almost impossible(volume integration requires multiple samples at each ray)

Results

NeRF encodes convincing view-dependent effects using directional dependence

NeRF encodes detailed scene geometry with occlusion effects

Summary

- Represent the scene as volumetric colored “fog”
- Store the fog color and density at each point as an MLP mapping 3D position (x, y, z) to color c and density σ
- Render image by shooting a ray through the fog for each pixel
- Optimize MLP parameters by rendering to a set of known viewpoints and comparing to ground truth images

3D Gaussian Splatting for Real-Time Radiance Field Rendering

SIGGRAPH 2023

3D Gaussian Splatting for Real-Time Radiance Field Rendering

BERNHARD KERBL*, Inria, Université Côte d'Azur, France
GEORGIOS KOPANAS*, Inria, Université Côte d'Azur, France
THOMAS LEIMKÜHLER, Max-Planck-Institut für Informatik, Germany
GEORGE DRETTAKIS, Inria, Université Côte d'Azur, France

Fig. 1. Our method achieves real-time rendering of radiance fields with quality that equals the previous method with the best quality [Barron et al. 2022], while only requiring optimization times competitive with the fastest previous methods [Fridovich-Keil and Yu et al. 2022; Müller et al. 2022]. Key to this performance is a novel 3D Gaussian scene representation coupled with a real-time differentiable renderer, which offers significant speedup to both scene optimization and novel view synthesis. Note that for comparable training times to InstantNGP [Müller et al. 2022], we achieve similar quality to theirs; while this is the maximum quality they reach, by training for 51min we achieve state-of-the-art quality, even slightly better than Mip-NeRF360 [Barron et al. 2022].

Radiance Field methods have recently revolutionized novel-view synthesis of scenes captured with multiple photos or videos. However, achieving high visual quality still requires neural networks that are costly to train and render, while recent faster methods inevitably trade off speed for quality. For unbounded and complete scenes (rather than isolated objects) and 1080p resolution rendering, no current method can achieve real-time display rates. We introduce three key elements that allow us to achieve state-of-the-art visual quality while maintaining competitive training times and importantly allow high-quality real-time (≥ 30 fps) novel-view synthesis at 1080p resolution. First, starting from sparse points produced during camera calibration, we represent the scene with 3D Gaussians that preserve desirable properties of continuous volumetric radiance fields for scene optimization while avoiding unnecessary computation in empty space; Second, we perform interleaved optimization/density control of the 3D Gaussians, notably optimizing anisotropic covariance to achieve an accurate representation of the scene; Third, we develop a fast visibility-aware rendering algorithm that supports anisotropic splatting and both accelerates training and allows real-time rendering. We demonstrate state-of-the-art visual quality and real-time rendering on several established datasets.

CCS Concepts: • **Computing methodologies** → **Rendering**; **Point-based models**; **Rasterization**; **Machine learning approaches**.

*Both authors contributed equally to the paper.

Authors' addresses: Bernhard Kerbl, bernhard.kerbl@inria.fr, Inria, Université Côte d'Azur, France; Georgios Kopanas, georgios.kopanas@inria.fr, Inria, Université Côte d'Azur, France; Thomas Leimkühler, thomas.leimkuehler@mpi-inf.mpg.de, Max-Planck-Institut für Informatik, Germany; George Drettakis, george.drettakis@inria.fr, Inria, Université Côte d'Azur, France.

0730-0301/2018/0-ART0 $15.00
https://doi.org/XXXXXXX.XXXXXXX

Additional Key Words and Phrases: novel view synthesis, radiance fields, 3D gaussians, real-time rendering

ACM Reference Format:
Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2018. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 0, 0, Article 0 (2018), 14 pages. https://doi.org/XXXXXXX.XXXXXXX

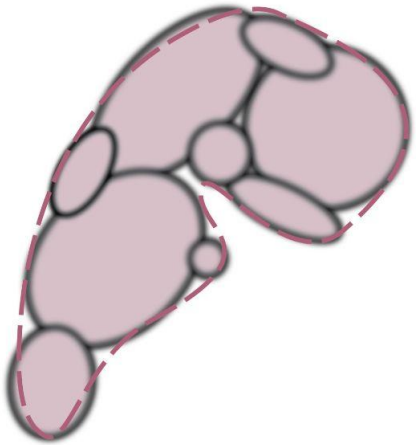
1 INTRODUCTION

Meshes and points are the most common 3D scene representations because they are explicit and are a good fit for fast GPU/CUDA-based rasterization. In contrast, recent Neural Radiance Field (NeRF) methods build on continuous scene representations, typically optimizing a Multi-Layer Perceptron (MLP) using volumetric ray-marching for novel-view synthesis of captured scenes. Similarly, the most efficient radiance field solutions to date build on continuous representations by interpolating values stored in, e.g., voxel [Fridovich-Keil and Yu et al. 2022] or hash [Müller et al. 2022] grids or points [Xu et al. 2022]. While the continuous nature of these methods helps optimization, the stochastic sampling required for rendering is costly and can result in noise. We introduce a new approach that combines the best of both worlds: our 3D Gaussian representation allows optimization with state-of-the-art (SOTA) visual quality and competitive training times, while our tile-based splatting solution ensures real-time rendering at SOTA quality for 1080p resolution on several previously published datasets [Barron et al. 2022; Hedman et al. 2018; Knapitsch et al. 2017] (see Fig. 1).

Our goal is to allow real-time rendering for scenes captured with multiple photos, and create the representations with optimization times as fast as the most efficient previous methods for typical real scenes. Recent methods achieve fast training [Fridovich-Keil

Volume Splatting

- 3D scene is represented as 3D Gaussians(blobs).



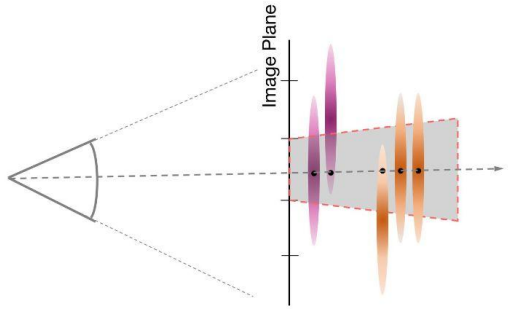
$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad \Sigma = \begin{bmatrix} \sigma_x & \sigma_{xy} & \sigma_{xz} \\ & \sigma_y & \sigma_{yz} \\ & & \sigma_z \end{bmatrix} \quad o \quad SH$$

position shape opacity color

How to Render? Sort/Splat/Blend

- Sort: based on depth
- Splat: compute the shape of the Gaussian after projection (3D→2D)
- Blend: alpha composite the gaussians (in 2D / screen space)

$$c = \sum_{i \in \mathcal{N}} c_i w_i \prod_{j=1}^{i-1} (1 - w_j)$$



$$p = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \quad \Sigma = \begin{bmatrix} \sigma_x & \sigma_{xy} & \sigma_{xz} \\ & \sigma_y & \sigma_{yz} \\ & & \sigma_z \end{bmatrix} \quad o \quad SH$$

position shape opacity color

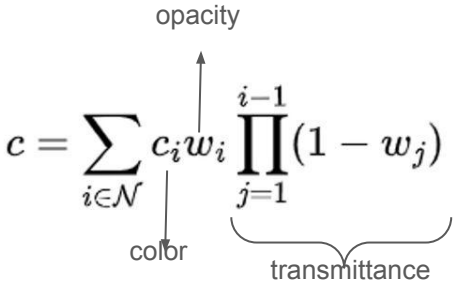
Splat

- Positions (3D→2D)
  - Just apply the classic matrices from graphics: $x' = P \bullet V \bullet x$
  - Perspective effect
    - divide by z-component of homogeneous coordinate
- Covariance (3D→2D)
  - World to view transformation is linear (Rotation+Translation)
  - Linear transformations apply a simple change to covariance matrix
  - Perspective effect
    - $\Sigma_{2d} = J_{proj} \, R \, \Sigma_{3d} \, R^T \, J_{proj}^T$

J: Jacobian of the affine approx. of projective matrix
R: 3D Gaussian orientation
∑: Covariance matrix

Blend

- How to implement this in a shader? (for serving in browser)
- Alpha Blending:



$$c = \sum_{i \in \mathcal{N}} c_i w_i \prod_{j=1}^{i-1} (1 - w_j)$$

- For 2 Gaussians:

$$c = c_0 w_0 + c_1 w_1 (1 - w_0)$$

- For 3 Gaussians:

$$c = c_0 w_0 + c_1 w_1 \left(1 - w_0\right) + c_2 w_2 \left(1 - w_1\right) \left(1 - w_0\right)$$

Optimization

- How to ensure that Gaussians remain Gaussians?

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{C}\sim\{\mathbf{C}_i\}}\, \mathbb{E}_{\mathbf{r}\sim\mathbf{C}}\, ||\mathbf{C}(\mathbf{r};\boldsymbol{\theta}) - \mathbf{C}_i(\mathbf{r})||_2^2$$
$$\boldsymbol{\theta} = \{(\mathbf{p}_k, o_k, \boldsymbol{\Sigma}_k, \mathbf{h}_k)\}$$

- Covariance matrices ought to be symmetric and positive semi-definite
  - all eigenvalues should be positive (or zero)

- Solution: re-parameterize the problem

$$\boldsymbol{\Sigma} = \mathbf{R}\mathbf{S}\mathbf{S}^\top\mathbf{R}^\top$$
$$\mathbf{R} = \mathrm{quat2mat}(\mathbf{q}), \quad \|\mathbf{q}\|_2 = 1$$
$$\mathbf{S} = \mathrm{diag}([s_x, s_y, s_z])$$

Can we just train a 3DGS now?

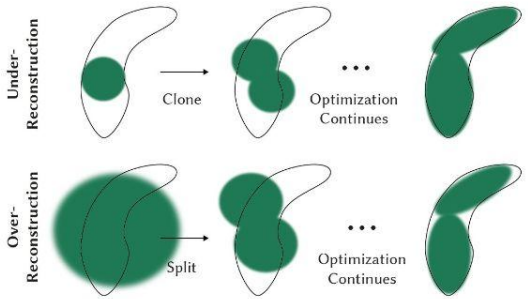
- No… you need to implement the “adaptive density control” heuristics

heuristics

Kerbl, Kopanas et al. “3D Gaussian Splatting for Real-Time Radiance Field Rendering” SIGGRAPH 2023

What are these heuristics?

- Densification
  - increase the number of points whenever under-reconstruction is detected, as measured by positional gradients
- Pruning
  - of very large Gaussians



- Opacity reset (every N iterations)
  - to remove floaters in the representation
- Require a good initialization (e.g. SFM)
  - as all the above are local heuristics

Kerbl, Kopanas et al. “3D Gaussian Splatting for Real-Time Radiance Field Rendering” SIGGRAPH 2023

Acknowledgements

- Slides modified from:

- Fundamentals of (neural) Inverse Rendering (Lecture talk at ICVSS ‘24)

  Assoc. Prof. Andrea Tagliasacchi
- Cornell Uni. CS5670: Computer Vision

Questions?