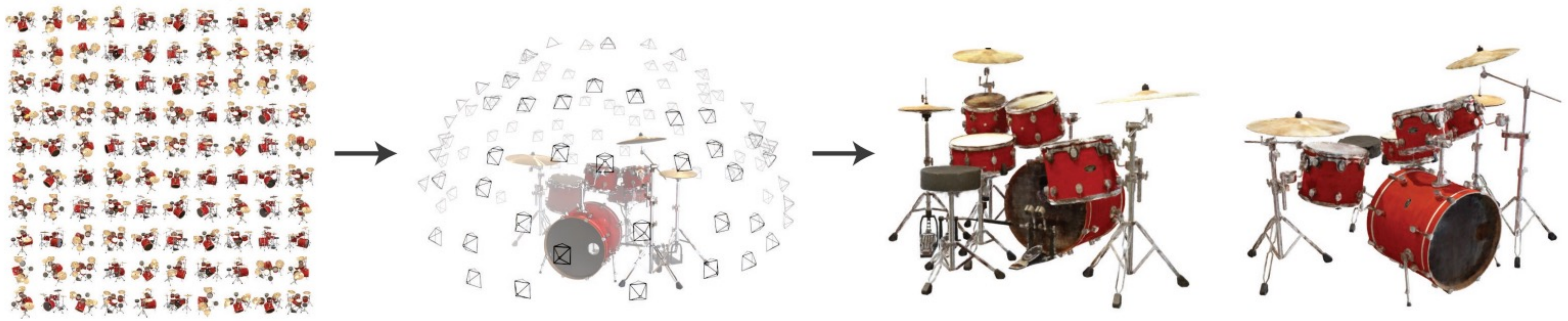
NeRFs

- Rahul Goel

Novel View Synthesis



Sparse Captured Images | Camera Poses of Images | Render from new unseen view

- NeRF or Neural Radiance Fields attempts to solve this problem.
- The entire scene representation is learnt using simply a multi-layer perceptron (MLP). No fancy CNNs or Transformers.

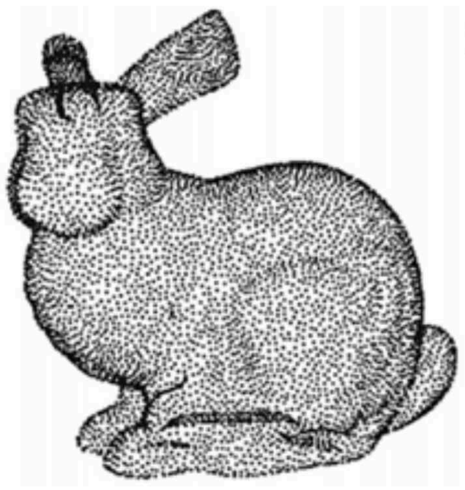
Radiance Fields

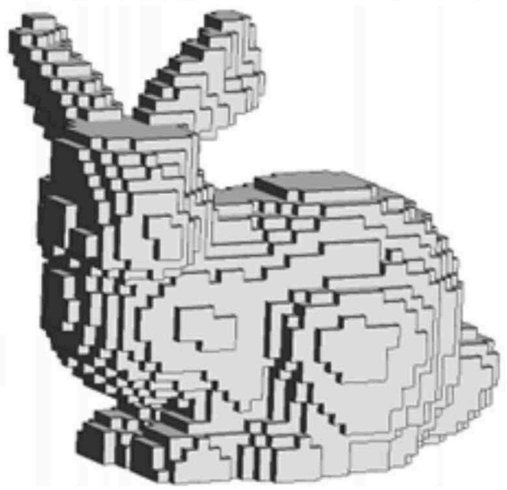
- Radiance fields can now be rendered on the web-browser for large scale scenes in REAL-TIME!
- Eg: https://gsplat.tech, https://niujinshuchong.github.io/mip-splatting-demo/

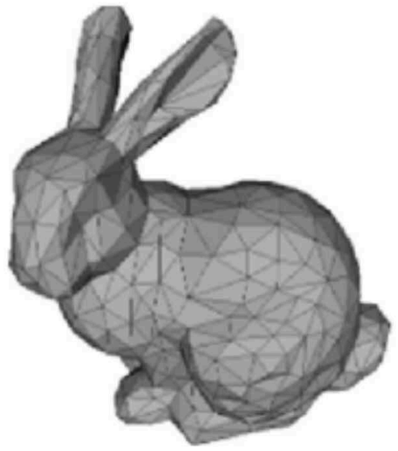
NeRF’s Impact on Computer Vision

- Conferences with Outstanding Paper Awards given to NeRF related works:
    - ECCV 2020
    - CVPR 2021
    - ICCV 2021
    - CVPR 2022
    - SIGGRAPH 2022
    - CVPR 2023
    - SIGGRAPH 2023
    - SIGGRAPH Asia 2023
    - ICCV 2023

Explicit 3D Representations


Point Cloud

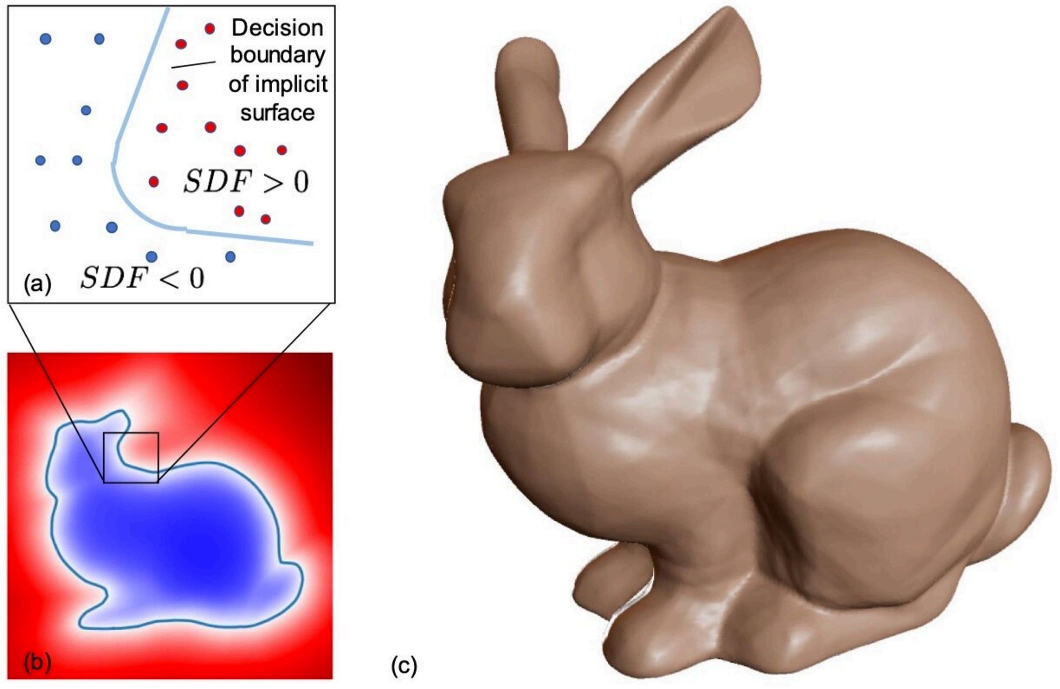

Voxels


Meshes

- Sampled from the underlying geometry.
- Discrete in nature.
- Storage requirements grows with the resolution.
- Often highly parallelizable in terms of computation.

https://www.antoinetlc.com/blog-summary/3d-data-representations

Implicit 3D Representations



Signed Distance Function

- A mapping of the 3D coordinate to some function of the underlying signal.
- Continuous in nature.
- Infinite resolution since it is a mathematical formula.
- Computationally expensive to evaluate:
  - Querying a 3D coordinate every time.
  - Lots of samples to query.

https://www.antoinetlc.com/blog-summary/3d-data-representations

Learning of 3D Shapes

- What kind of representations are suitable for deep learning?
  - It depends on the task.
- Is the shape fixed for your task?
  - Meshes are good. Learn something on top of meshes.
- I know very coarse shape but fine details are missing?
  - Learn some deformation of the base shape.
- What if we don’t know anything about the shape beforehand?
  - Continuous representations are better.
  - NeRFs lie in this category.

Radiance Fields

- 5D function.
- What color does a point in space emit for a view direction?

$$f : \mathcal{R}^3 \times \mathcal{S}^2 \rightarrow \mathcal{R}^3$$

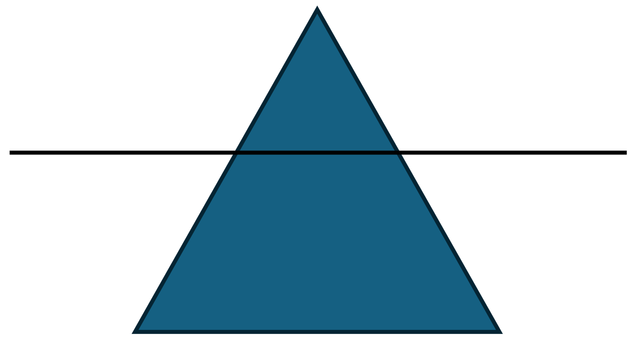
- Input: $(x, y, z, \theta, \varphi)$
- Output: RGB

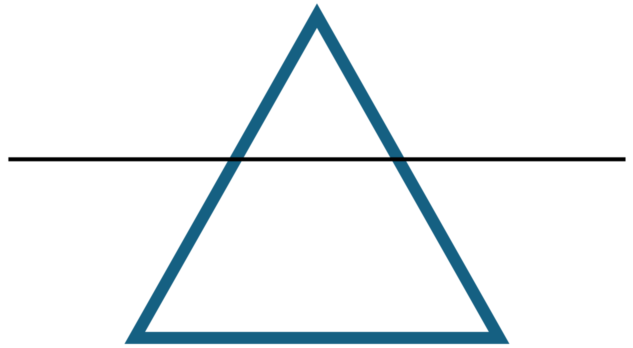
Volumetric Density (σ)

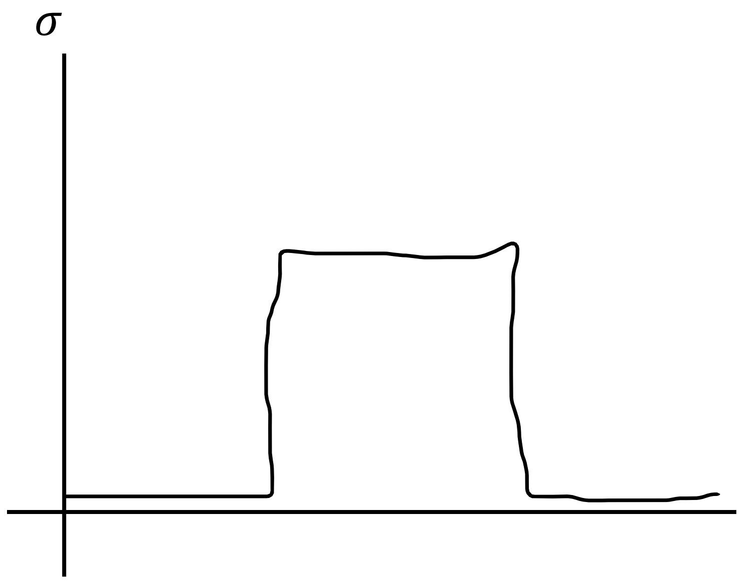
- 3D function
- Two analogous questions:
  - How much opaque content is present at a point in space?
  - What is the probability of a light ray terminating at a point in space?

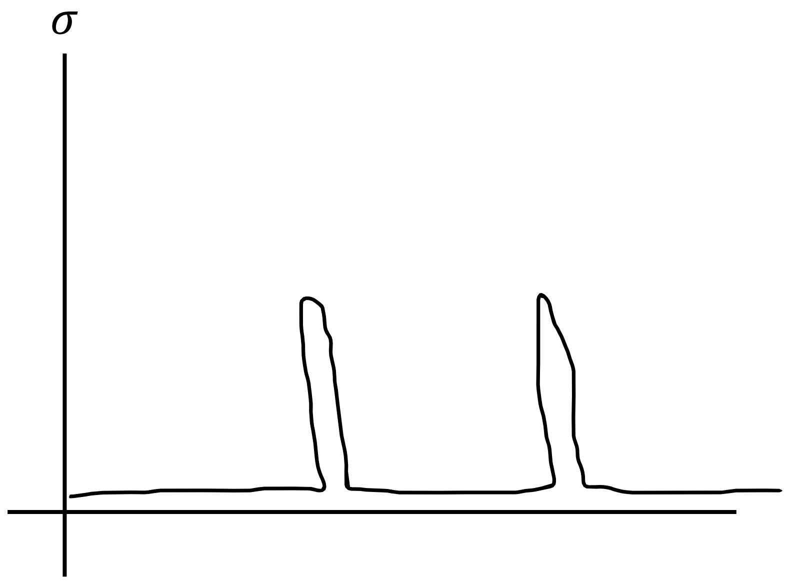
$$f : \mathcal{R}^3 \rightarrow \mathcal{R}$$

Volumetric Density Example









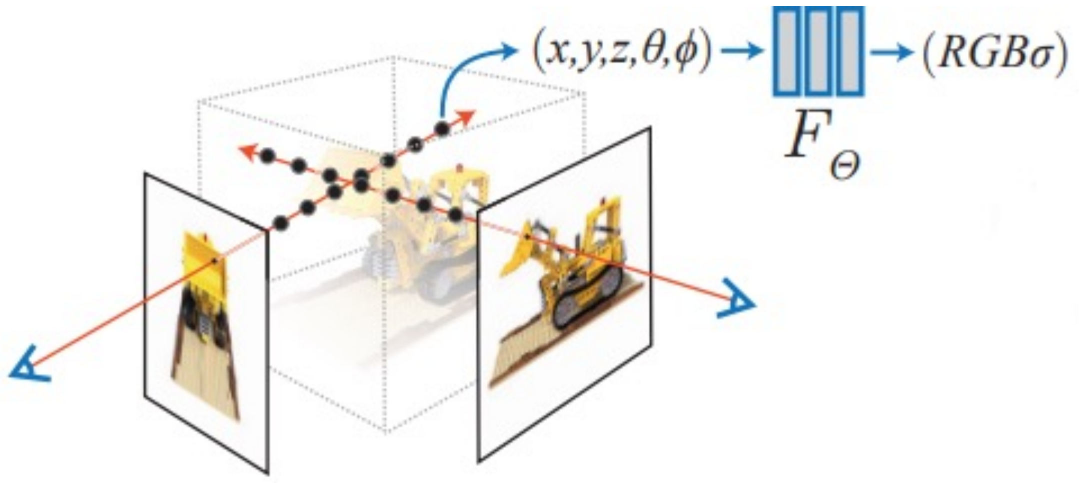
How to render a radiance field?

- We have a radiance field function and a volumetric density function that collectively tell us light emitted from points and probability of light ray terminating at points.
- How do we obtain a 2D image from this 3D representation?

How to render a radiance field?

- We have the camera pose for which we want to render an image.
- We shoot a ray from the camera center into the pixel on the image plane.
- Take lots and lots of samples on the ray.
- For each sample, evaluate the radiance field and volumetric density.

- Somehow blend these values together into a single RGB color for the pixel.


$(x,y,z,\theta,\phi) \rightarrow \boxed{F_\Theta} \rightarrow (RGB\sigma)$

Volumetric Rendering Equation

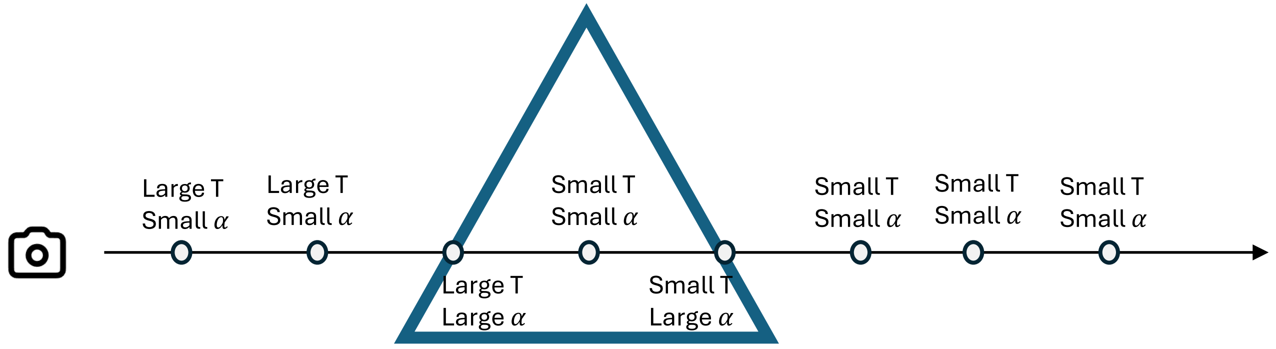
- The sampled values are blended together using the volumetric rendering equation.

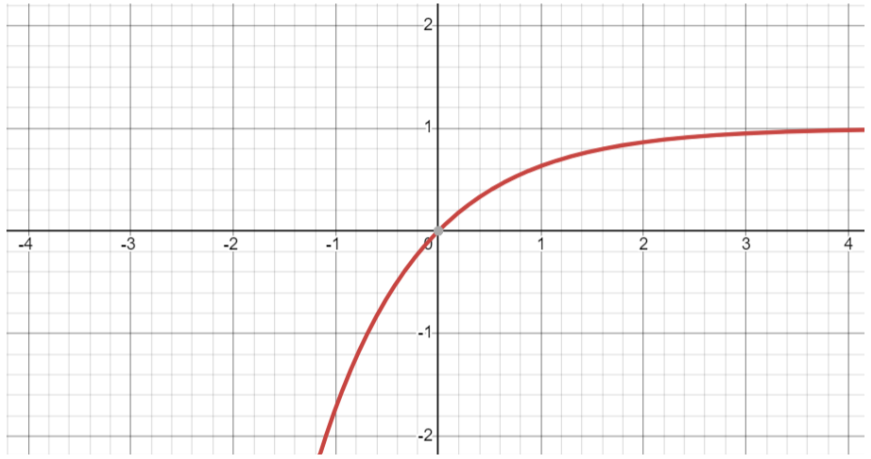
$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt\,, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))ds\right)$$

Discrete Volume Rendering Equation

$$\hat{C}(r) = \sum_{i=1}^{N} T_i \alpha_i c_i, \text{where } T_i = \prod_{j=1}^{i-1}(1-\alpha_i)$$

$$\alpha_i = 1 - \exp(-\sigma_i \delta_i)$$





Quick Summary

- We saw the radiance field and volumetric density functions for a 3D space.
- Given a camera pose, we can shoot a ray into a pixel in the image plane and take lots of samples on the ray.
- If we know the radiance and density values on those points, we can use the discrete volume rendering equation to get the pixel value.
- We can repeat this for all pixels to get the entire image.

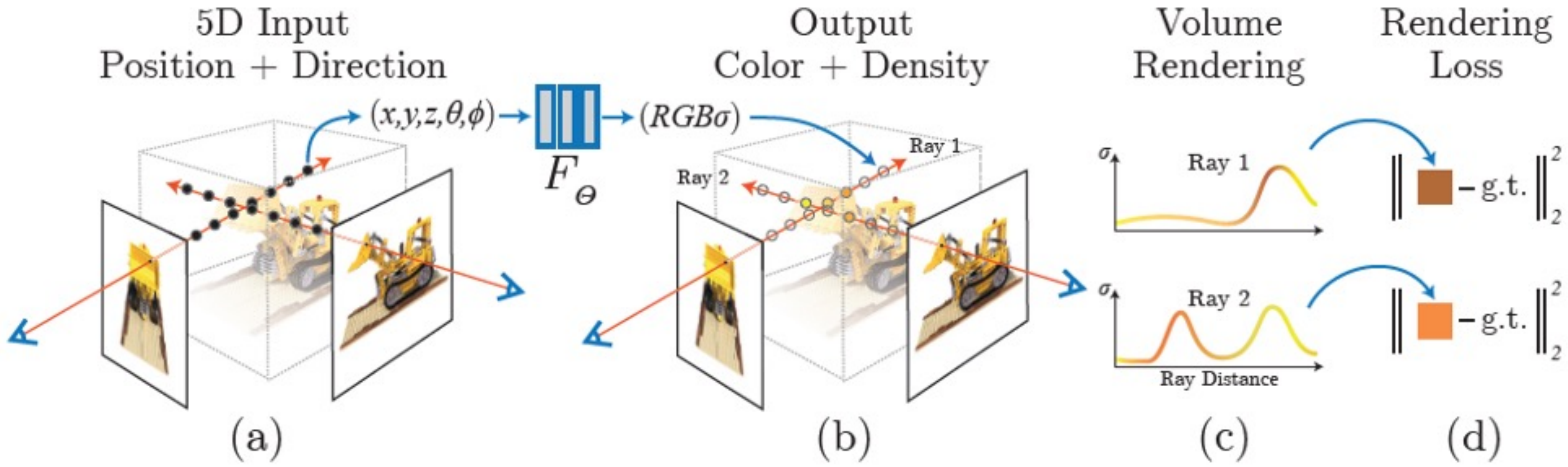
Differentiable Rendering

- The entire process described above is differentiable!
- This means that gradients can flow through it. We can use backpropagation through the volume rendering equation.
- We assumed that we have the radiance field and density function and using it we can render the images.

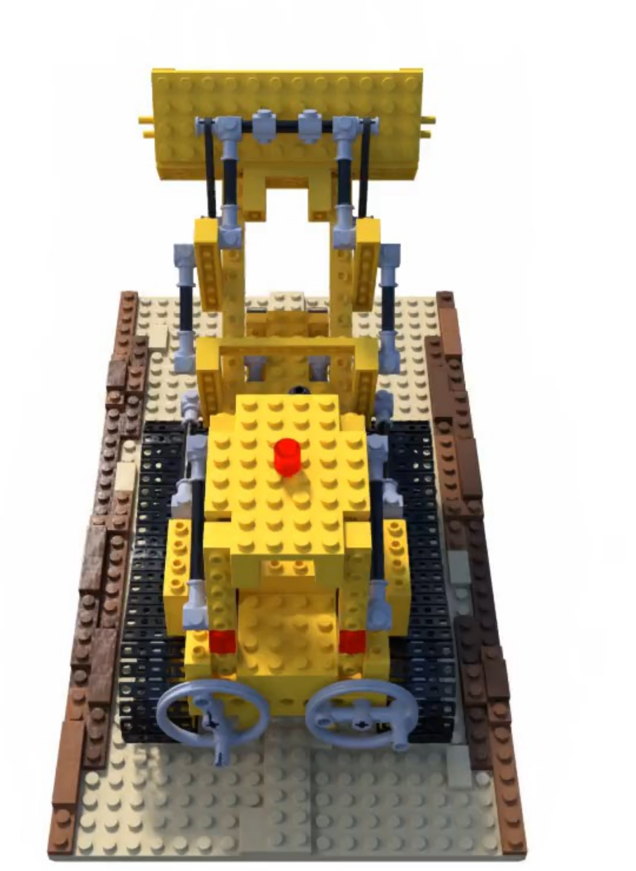
- But due to differentiability of the entire process, if we have images and their camera poses, we can take loss on the images and learn the radiance field and density function.

NeRF: Neural Radiance Field

- We need a differentiable "container" to learn and store the radiance field.
- Simply use a neural network to represent it.
- After learning, take any new camera pose and repeat the procedure to generate a "novel view".

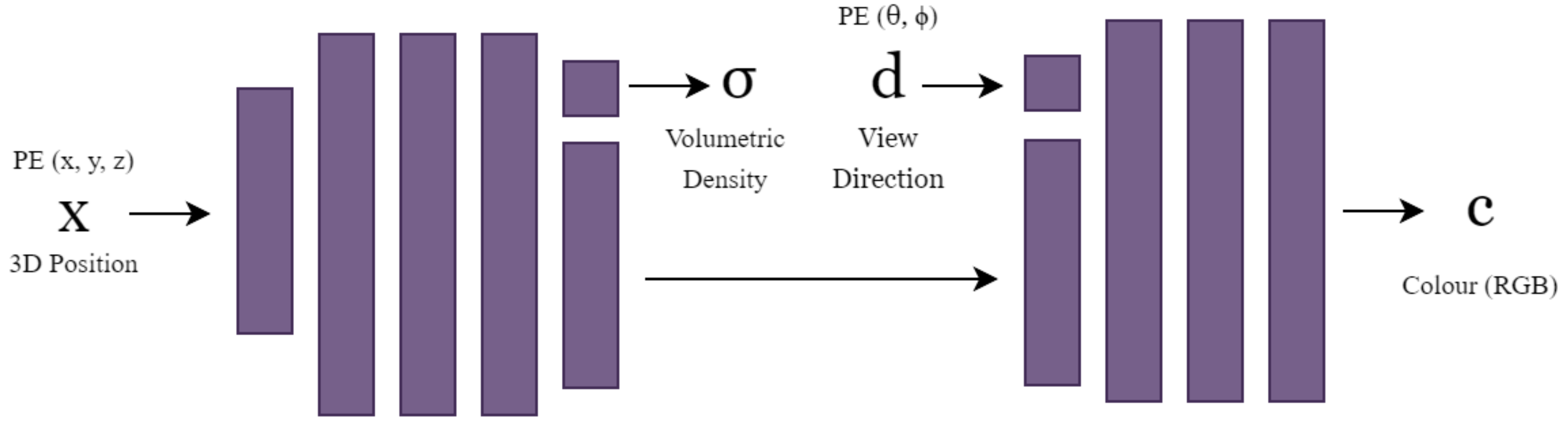


NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, ECCV 2020



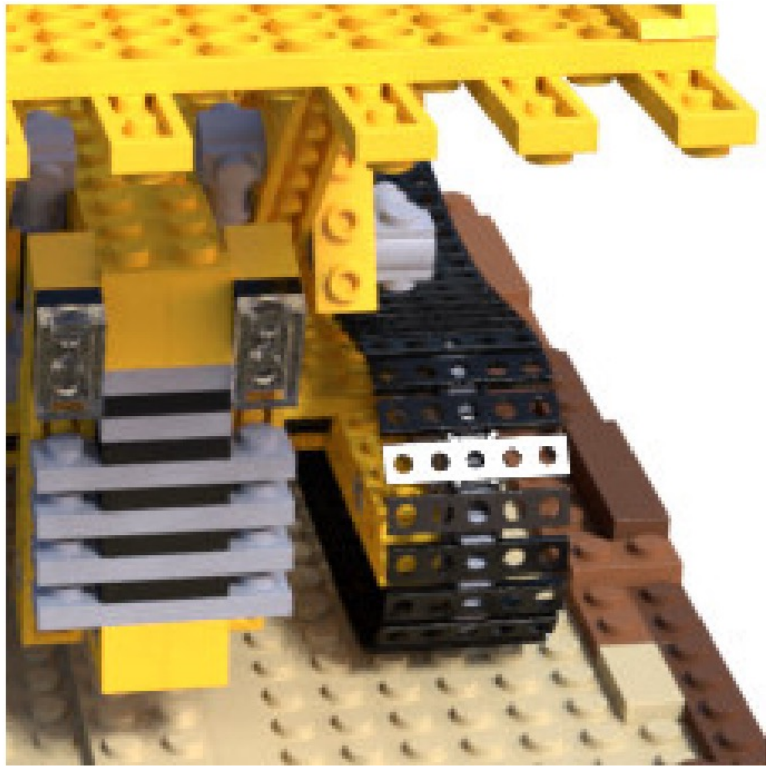
NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, ECCV 2020

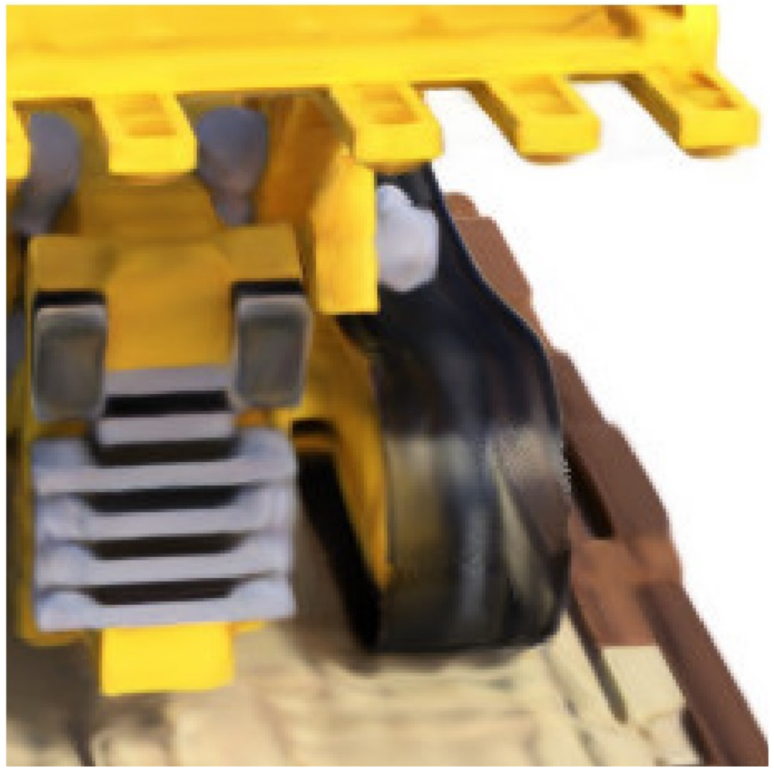
NeRF MLP*



*not exactly but it's ok.

Positional Encoding


Ground Truth


NeRF w/o PE

Positional Encoding

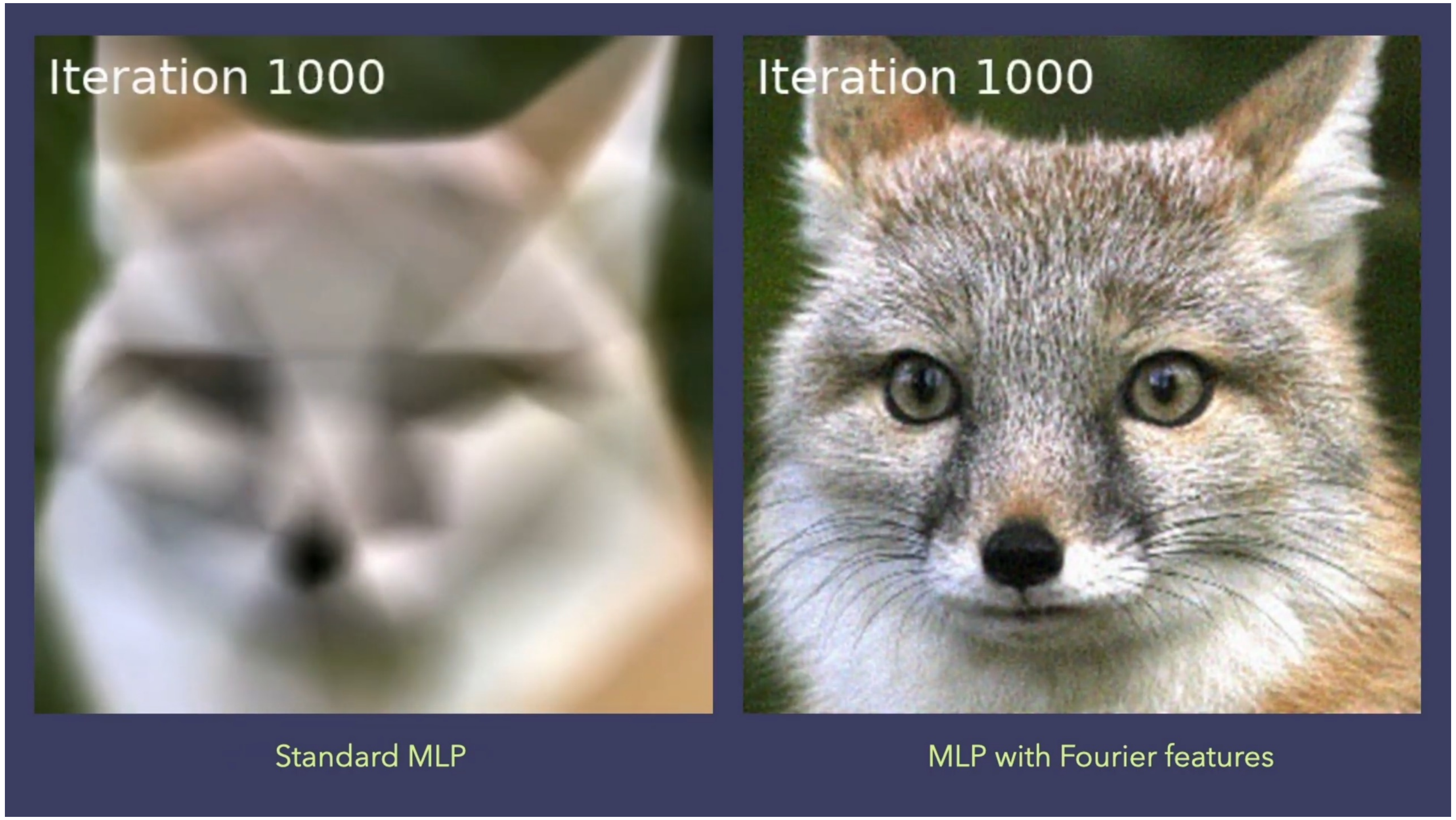
- Coordinate Neural Networks by default converge on low frequency signals.
- But if we perform positional encoding, they are able to learn high frequency details.

- What is positional encoding?

$$PE(x) = (\sin x, \cos x, \sin 2x, \cos 2x, \ldots \sin nx, \cos nx)$$

- PE helps a neural network to differentiate well between near-by coordinates. This helps in learning high frequency information in the spatial domain.

Positional Encoding



Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains, NeurIPS 2020

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis, ECCV 2020

Question Break

NeRFs are bad.

- Every point is treated to be a light-emitter. That’s not what happens in the real-world.
- You cannot re-light the scene on-the-fly.
- Cannot decompose the scene into a surface + material/texture.
- You cannot animate them properly (yet).
- Solid objects are learnt to be hollow or perhaps something else altogether.

NeRFs are great.

- 3D from a few RGB images.
- Can be converted to meshes.
- Use for 3D semantics and segmentation.
- Generation of 3D assets when coupled with diffusion.
- Used for scene understanding and robotics tasks.

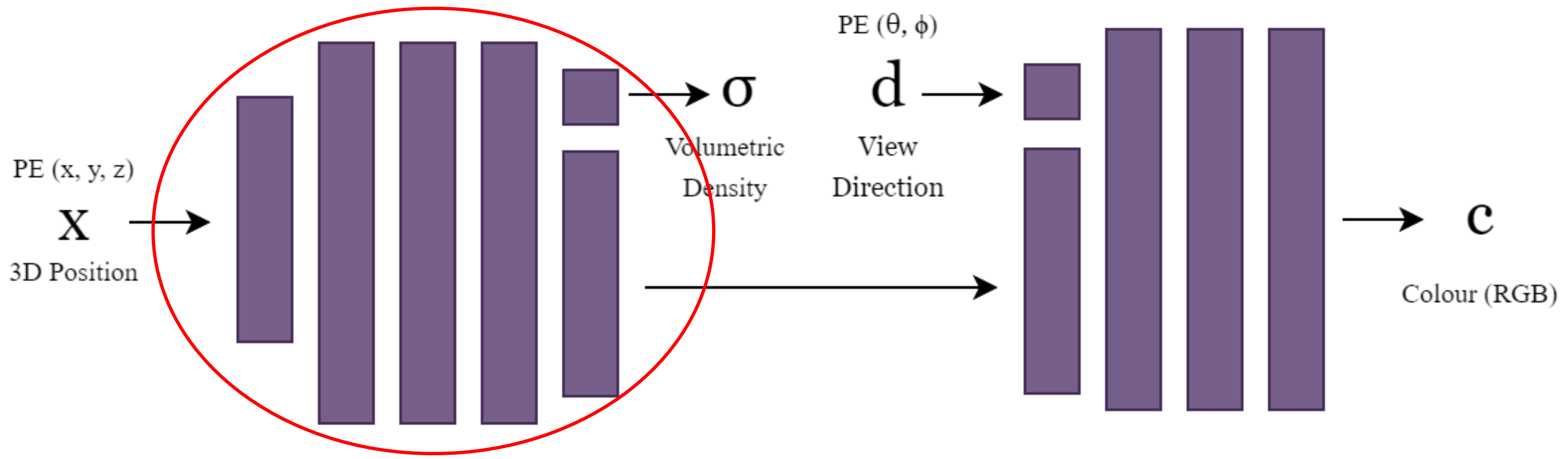
NeRFs are slow. Why?

- Training one NeRF takes 16-24h.
- Rendering an image takes ~30 seconds or so.

- For every sampled point, an 8-layer MLP is evaluated!
- Volumetric rendering is expensive, especially when sampled densely.

- We’ll tackle these one by one.

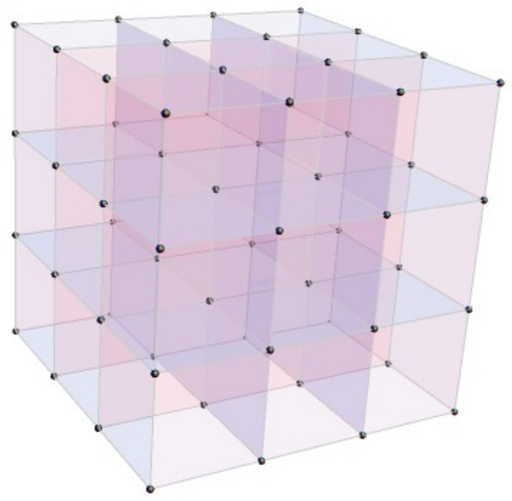
How can we make it faster?

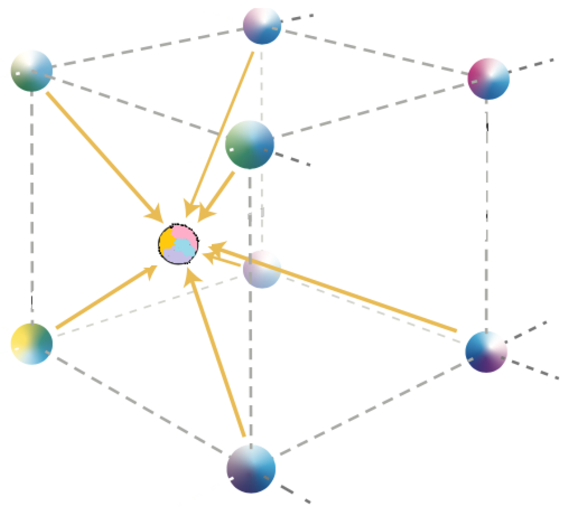


- This part encodes information that is **dependent only on position**.
- Yet, the entire MLP contributes to the output. There is hardly any spatial flow of information.

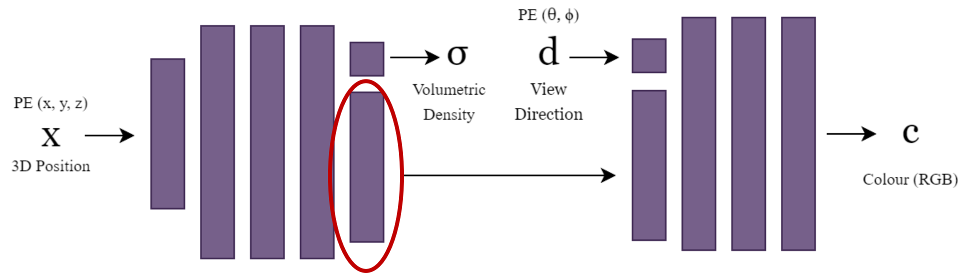
Grid Storage

- The density can be stored explicitly in a grid-like manner. (Simply a 3D tensor/matrix.)
- The bounding box of the 3D scene is mapped to the bounds of this grid. (Linear mapping.)
- Density storage points are uniformly distributed in this grid.
- Density at x is evaluated by doing tri-linear interpolation of the nearest 8 values. (Making the density field continuous.)





More Grid Storage



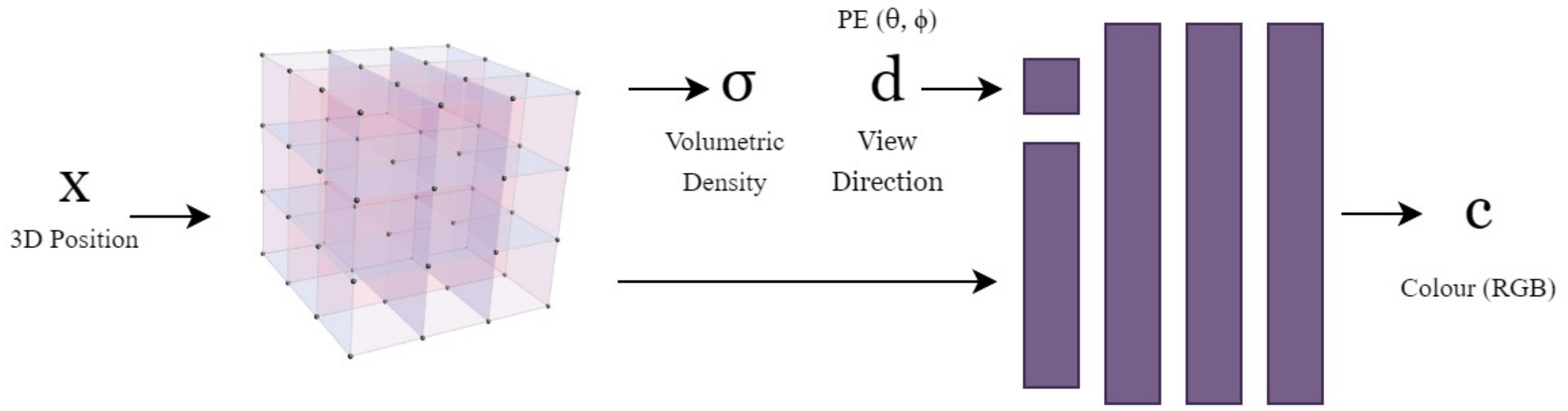
- Why limit ourselves to just density?
- We can store the other latent space vector there as well!

- We store a k-dimensional vector in a 3D Grid as well. (A 4D tensor.)
- Again, the discrete grid is seen as a continuous field due to tri-linear interpolation of these k-dimensional vectors.

Coarse Occupancy Grids

- A lot of empty space that doesn’t have any density.
- Maintain a coarse occupancy grid.
- Binary in nature:
  - If no dense point is there: 0.
  - If at least a single dense point is there: 1.
- Check quickly with this grid before querying for density, colour, etc.
- Easily avoid wasteful queries.

Updated Architecture



- Much smaller MLP which is faster to train.
- Now, every point **x** affects at most 8 learnable parameters. (Instead of an entire MLP.)
- Occupancy Grid for pruning out points.
- These “GridEncoded” NeRFs train in just 20 minutes.

Grids are Bulky

- MLP NeRFs took at least 16 hours to train.
- Grid NeRFs take at most 20 minutes.

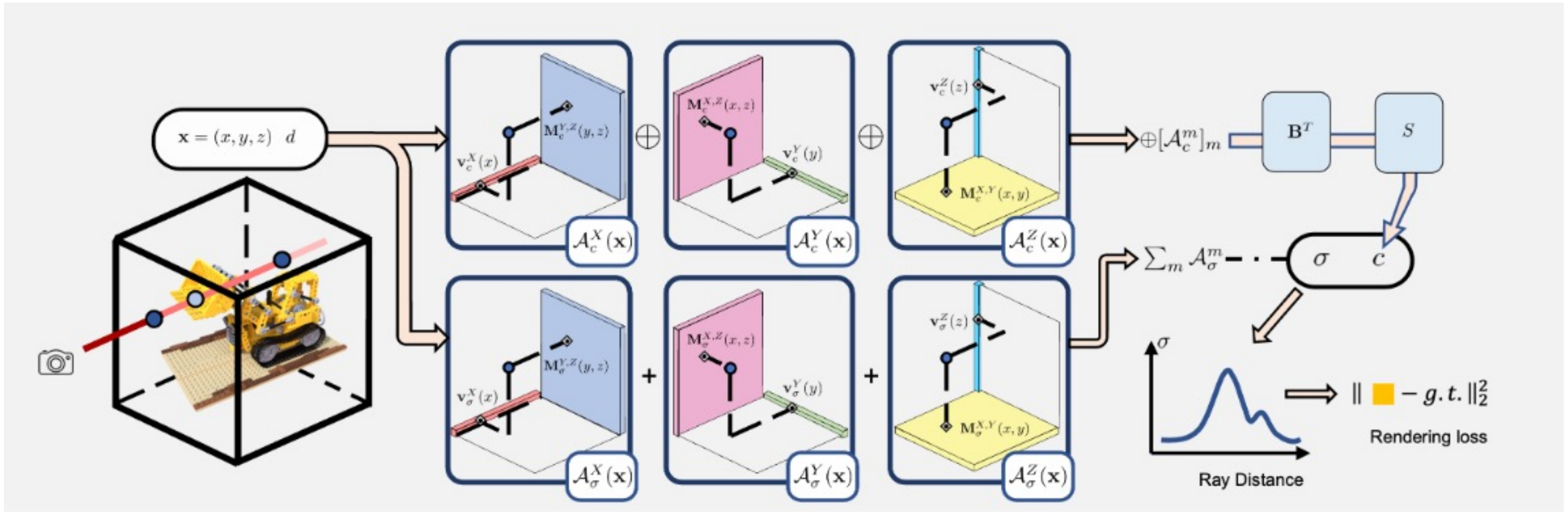
- MLP NeRFs were just 5-10 MB in size.
- These grids are bulky and consume 800MB-1GB of storage.

Can we do better?

- MLP based NeRFs have shown us that it is possible to store the entire learnt 3D information in 5-10MB.
- Therefore, it should be possible to reduce the 800MB-1GB memory footprint theoretically.

- These grids are redundant:
  - Most of the space in 3D is empty!
  - Some ways to reduce these redundant spaces: vector quantization, octrees, low-rank approximations.

Tensorial Decomposition of Grids


$\mathbf{x} = (x, y, z) \;\; d$
$\mathbf{v}_c^X(x)$
$\mathbf{M}_c^{Y,Z}(y,z)$
$\mathbf{M}_c^{X,Z}(x,z)$
$\mathbf{v}_c^Y(y)$
$\mathbf{v}_c^Z(z)$
$\mathbf{M}_c^{X,Y}(x,y)$
$\mathcal{A}_c^X(\mathbf{x})$
$\mathcal{A}_c^Y(\mathbf{x})$
$\mathcal{A}_c^Z(\mathbf{x})$
$\mathbf{v}_\sigma^X(x)$
$\mathbf{M}_\sigma^{Y,Z}(y,z)$
$\mathbf{M}_\sigma^{X,Z}(x,z)$
$\mathbf{v}_\sigma^Y(y)$
$\mathbf{v}_\sigma^Z(z)$
$\mathbf{M}_\sigma^{X,Y}(x,y)$
$\mathcal{A}_\sigma^X(\mathbf{x})$
$\mathcal{A}_\sigma^Y(\mathbf{x})$
$\mathcal{A}_\sigma^Z(\mathbf{x})$
$\oplus[\mathcal{A}_c^m]_m$
$\mathbf{B}^T$
S
$\sum_m \mathcal{A}_\sigma^m$
$\sigma \quad c$
σ
$\| \text{■} - g.t. \|_2^2$
Rendering loss
Ray Distance

- Tensors are decomposed using VM decomposition. (Vector Matrix decomposition.)
- Reducing the memory footprint from $O(N^3)$ to $O(N^2)$!
- Around 20MB of memory required.
- Since the memory consumed is low, they **blatantly increased resolution of the voxel grid** and beat the state-of-the-art quality.
- This is the representation I chose for my research as well.

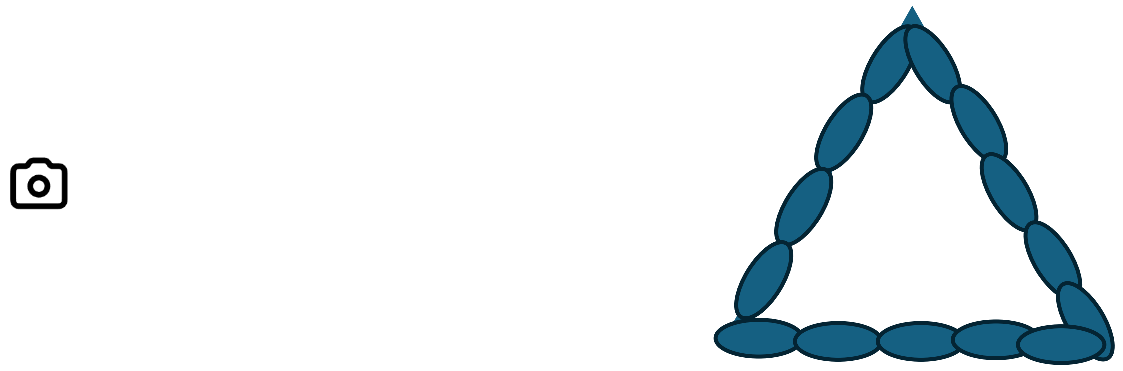
Question Break

3D Gaussian Splatting

- Recent breakthrough presented at SIGGRAPH 2023.
- Completely changed the paradigm of radiance fields:
  - Ray Casting → Rasterization
  - Neural Network → No Neural Network
  - Implicit Representation → Point Clouds

- Seconds per frame → Hundreds of frames per second
- All while maintaining incredibly high quality!

Scene represented using 3D Gaussians

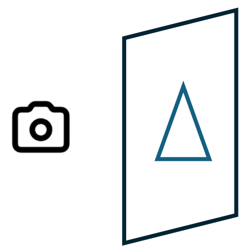


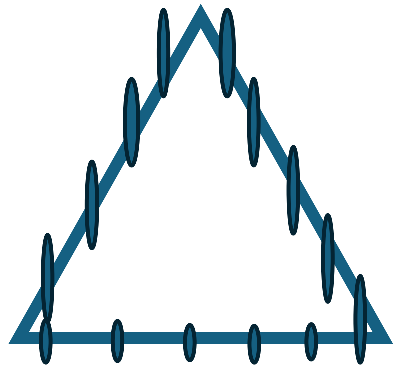
For every Gaussian there is:

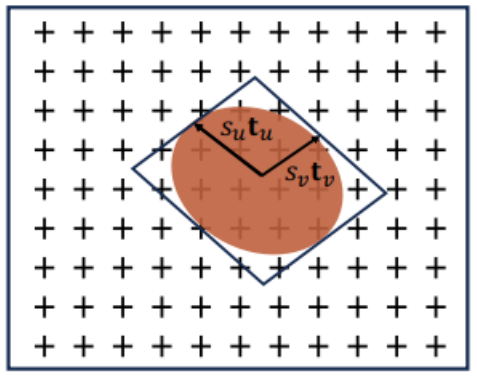
- 3D Position vector
- 3D Scaling vector
- 4D Rotation vector

- 1D Opacity Scalar
- 45 Spherical Harmonics Coefficients for color.

Project to 2D







Every 3D Gaussian is projected to the 2D plane perpendicular to the Image Plane.

In the Image Plane, a Gaussian covers multiple pixels.

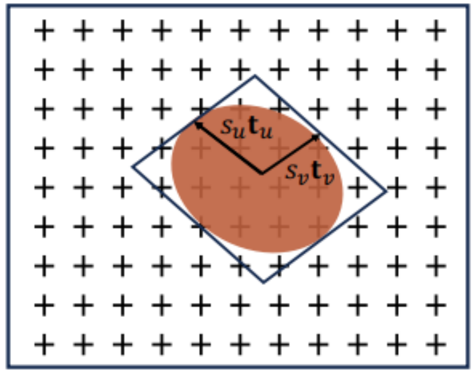
Blending

- Sort the Gaussians according to depth value.
- Blend them in back-to-front order.

Blending Equation

For one pixel:

- Evaluate the 2D Gaussian and multiply by the opacity of the Gaussian to get its contribution. This is α.
- Use the camera position and mean of gaussian to get view direction. Use this to evaluate color from SH.
- We know the back-to-front order of Gaussians from sorting. We can use the rendering equation.



$$\hat{C}(r) = \sum_{i=1}^{N} T_i \alpha_i c_i, \text{where } T_i = \prod_{j=1}^{i-1} (1 - \alpha_i)$$

Every step is differentiable

- Projection is differentiable.
- Blending follows the same old equation.

- We can take loss on the image, backprop and optimize the parameters of the gaussians.

- Geometrically, Gaussians move around, get bigger/smaller, rotate around and change their appearance so as to minimize the loss function.

Why is Gaussian Splatting so fast?

- Rasterization is faster than ray tracing (in general).
- We don’t evaluate every gaussian everywhere. We clip it till the third standard deviation. This means that a gaussian only has to shade a local region of pixels.
- Really efficient tile-based CUDA implementation by the authors.

- Project Page of Gaussian Splatting if anyone is interested in learning how it works: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

- 100+FPS for large scale scenes. https://gsplat.tech, https://niujinshuchong.github.io/mip-splatting-demo/

Question Break

Advancements made by IIIT Hyderabad.

- Interactive Segmentation of Radiance Fields (CVPR 2023)
  Rahul Goel, Dhawal Sirikonda, Saurabh Saini, P. J. Narayanan
- Canonical Fields: Self-Supervised Learning of Pose-Canonicalized Neural Fields (CVPR 2023)
  Rohith Agaram, Shaurya Dewan, Rahul Sajnani, Adrien Poulenard, Madhava Krishna, Srinath Sridhar
- HyP-NeRF: Learning Improved NeRF Priors using a HyperNetwork (NeurIPS 2023)
  Bipasha Sen, Gaurav Singh, Aditya Agarwal, Rohith Agaram, K Madhava Krishna, Srinath Sridhar
- MANUS-Markerless Hand-Object Grasp Capture using Articulated 3D Gaussians (CVPR 2024)
  Chandradeep Pokhariya, Ishaan N Shah, Angela Xing, Zekun Li, Kefan Chen, Avinash Sharma, Srinath Sridhar
- GSN: Generalisable Segmentation in Neural Radiance Fields (AAAI 2024)
  Vinayak Gupta, Rahul Goel, Sirikonda Dhawal, P. J. Narayanan