Novel View Synthesis

Oleg Desheulin, Kirill Struminsky

Problem: View Interpolation

Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” ECCV 2020.

Problem: View Interpolation

Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” ECCV 2020.

Synthetic Scenes

Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” ECCV 2020.

View-dependence

Important feature of NeRF
is also view dependence

Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” ECCV 2020.

Basic NeRF

NeRF Approach

Input: set of images and camera poses

For each pixel release a ray, sample points -> MLP -> integrate and minimize L2 loss

$$x = r(t) = o + td.$$

Positional encoding:

$$\gamma(x) = \left[sin(x), cos(x), \ldots sin(2^{L-1}x), cos(2^{L-1}x)\right]$$

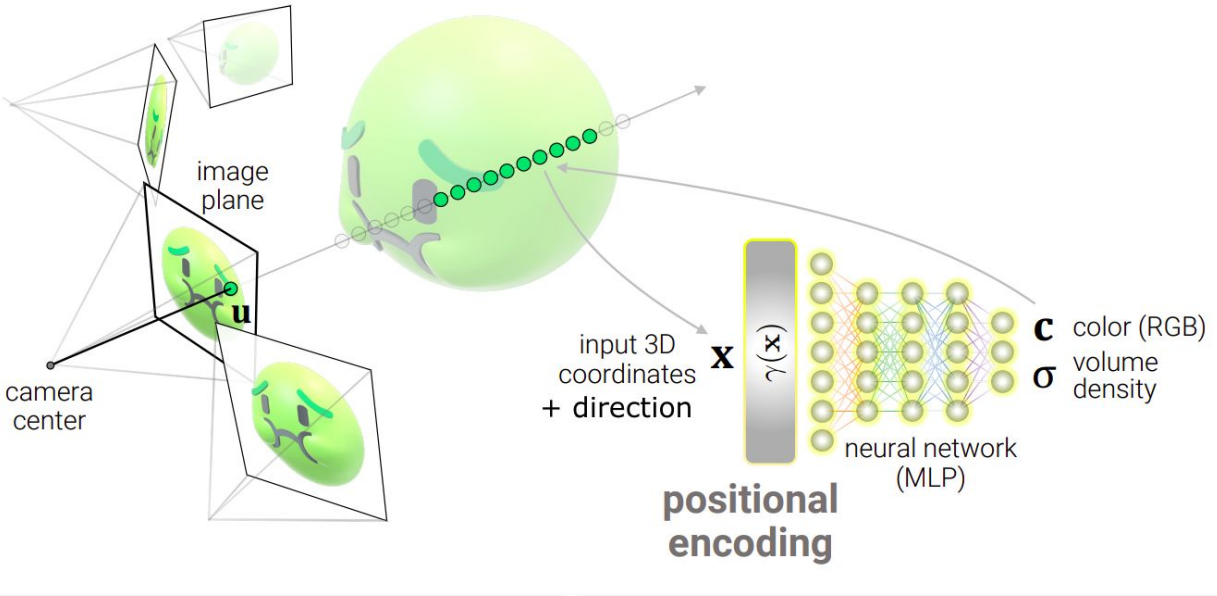


Image credit: https://chenhsuanlin.bitbucket.io/bundle-adjusting-NeRF/poster.pdf

Emission-Absorption Model

Model def: $g(t) = \sigma(t)c(t)$ - what we will see in t if nothing was absorbed

Differential eq. for absorption: $$\frac{dI}{dt} = g(t) - \sigma(t)I(t)$$

Solve and get:

$$I(t_T) = \int_{t_0}^{t_T} g(s)T(s)ds = \int_{t_0}^{t_T} \sigma(s)c(s)T(s)ds \qquad T(t) = \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right)$$

From integrals to real life

Our integral:

$$I(t_T) = \int_{t_0}^{t_T} g(s)T(s)ds = \int_{t_0}^{t_T} \sigma(s)c(s)T(s)ds \qquad T(t) = \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right)$$

Quadrature approximation:

$$\hat{C}(r) = \sum_k T_k(1 - \exp(-\sigma_k(t_{k+1} - t_k)))c_k; \text{ where } T_k = \exp\left(-\sum_{k'<k} \sigma_{k'}(t_{k'+1} - t_{k'})\right)$$

Loss function: $$\mathcal{L}_{\text{RGB}} = \sum_{\mathbf{r}\in\mathcal{R}} \left\| C(\mathbf{r}) - \hat{C}(\mathbf{r}) \right\|_2^2$$

Integral Approximation

Rendering model for ray r(t) = o + td:

$$C \approx \sum_{i=1}^{N} T_i \alpha_i c_i$$

weights

colors

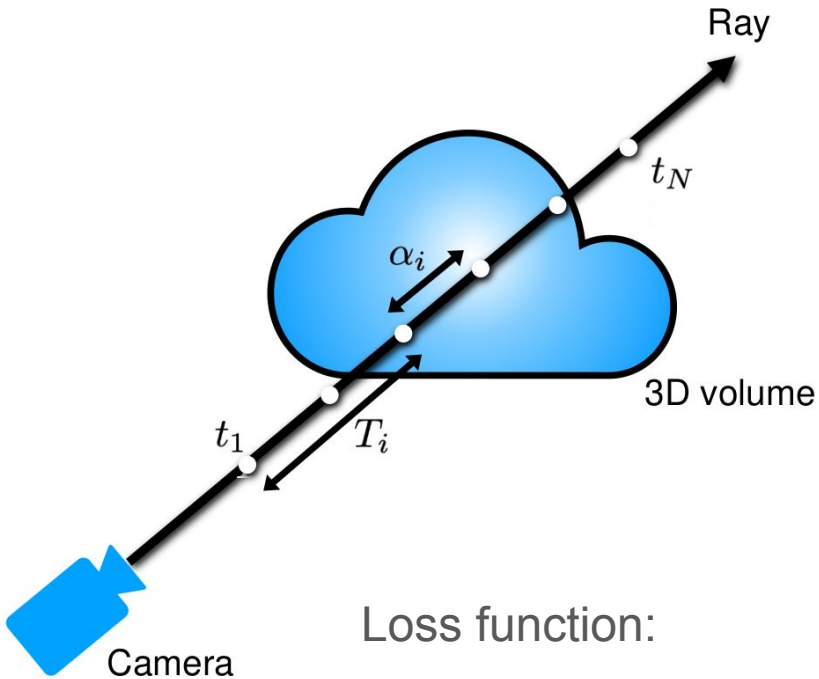
How much light is blocked earlier along ray:

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

How much light is contributed by ray segment i:

$$\alpha_i = 1 - e^{-\sigma_i \delta t_i}$$

← Density * Distance Between Points



Loss function:

$$\mathcal{L}_{\text{RGB}} = \sum_{\mathbf{r} \in \mathcal{R}} \left\| C(\mathbf{r}) - \hat{C}(\mathbf{r}) \right\|_2^2$$

Slide source: https://courses.grainger.illinois.edu/CS598DWH/fa2021/lectures/Lecture%2006%20-%20Novel%20View%20Synthesis%20-%203DVision.pdf

Encoding modifications

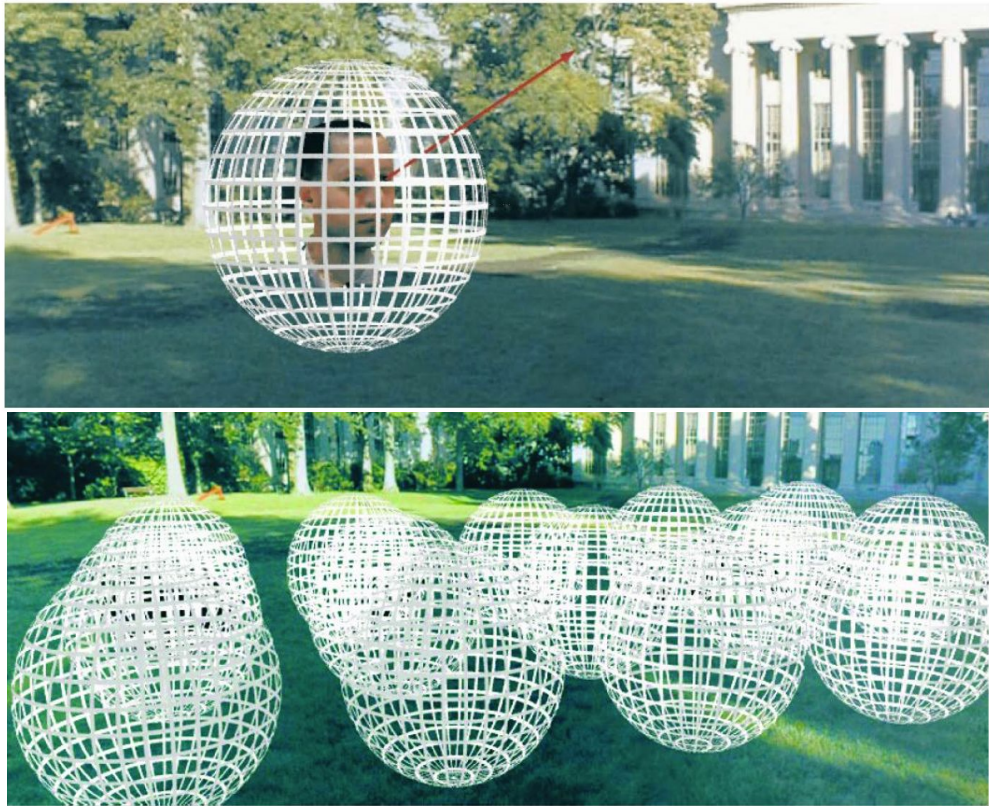
NeRF is a Plenoptic Function

Function $P = P(\theta, \phi, \lambda, t, V_x, V_y, V_z)$

describes everything viewer sees from position V at angle (ϕ,θ) at some time t and wavelength λ

If we drop time and wavelength we will get NeRF

Viewing direction & position can be factorized



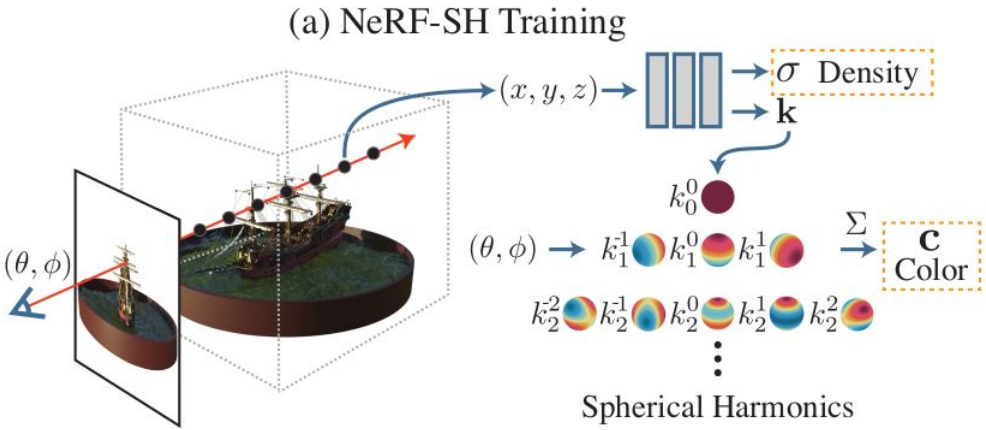
Plenoptic functions:
http://persci.mit.edu/pub_pdfs/elements91.pdf
Source:https://alexyu.net/plenoctrees/

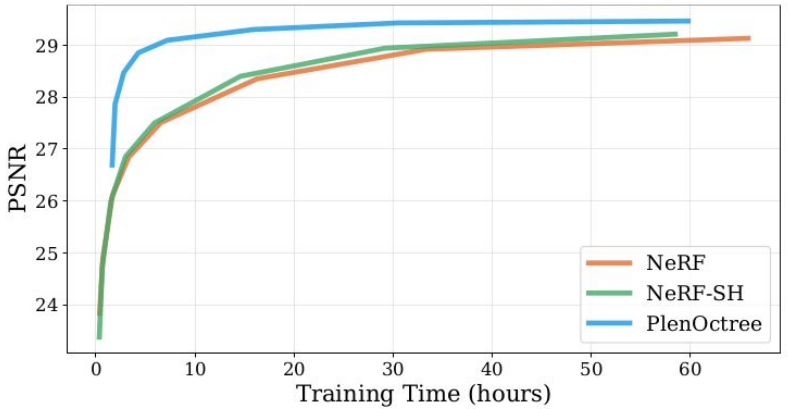
NeRF-SH: Spherical Harmonics

We can omit dependence of MLP on viewing direction and reparameterize the output

Coefficients Y are predefined spherical harmonics and color is obtained by following:

$$c(\mathbf{d};\mathbf{k}) = S\left(\sum_{\ell=0}^{\ell_{\max}} \sum_{m=-\ell}^{\ell} k_\ell^m Y_\ell^m(\mathbf{d})\right)$$

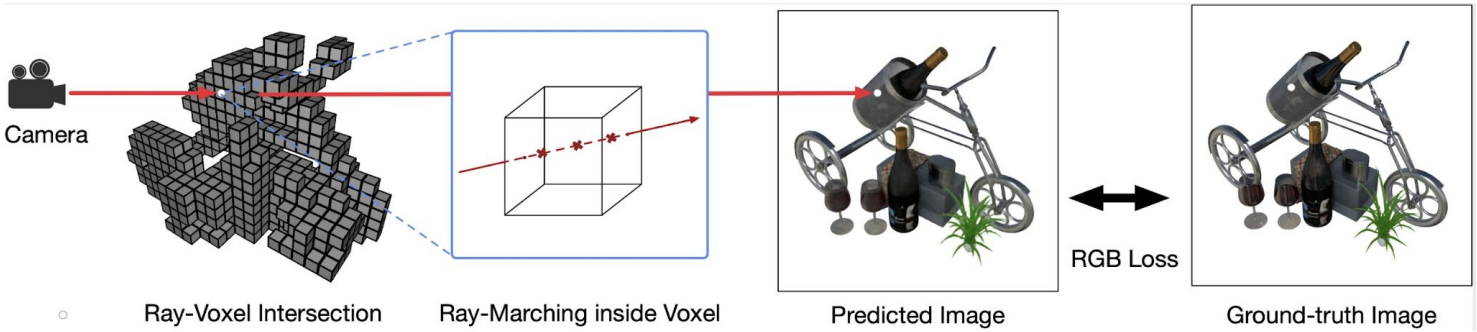




Source:https://alexyu.net/plenoctrees/

Neural Sparse Voxel Fields: Split Scene Into Voxels

Rendering procedure:



Self-pruning and updating octree:

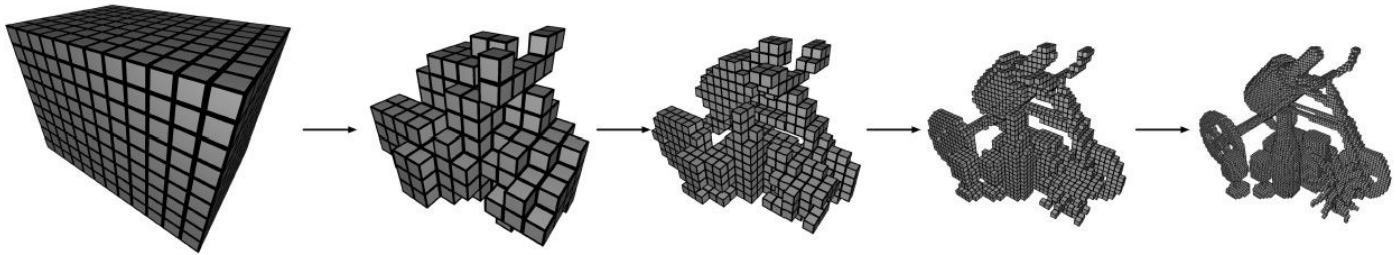


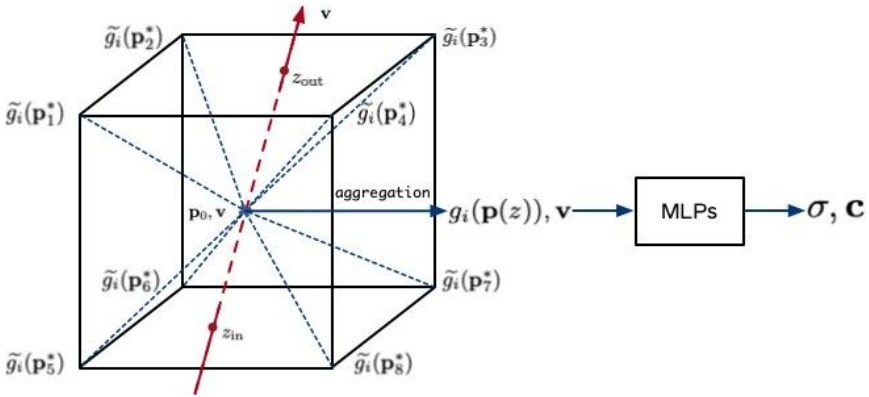
Image credit: https://lingjie0206.github.io/papers/NSVF/

Voxel-based Scene Representation

- For a given point ***p*** in voxel ***V***

$$F_\theta^i : (\boldsymbol{g}_i(\boldsymbol{p}), \boldsymbol{v}) \rightarrow (\boldsymbol{c}, \sigma), \forall \boldsymbol{p} \in V_i$$

voxel embedding *ray direction* *color* *density*



- Voxel embedding is defined as:

Trilinear interpolation *Voxel features (e.g. learnable voxel embeddings)*

$$g_i(\boldsymbol{p}) = \zeta\left(\chi\left(\tilde{g}_i(\boldsymbol{p}_1^*), \ldots, \tilde{g}_i(\boldsymbol{p}_8^*)\right)\right)$$

Positional encoding

Image credit: https://lingjie0206.github.io/papers/NSVF/

How to Prune Unused Voxels/Add a New One?

Voxel pruning rule:

$$V_i \text{ is pruned } \textbf{if } \min_{j=1\ldots G} \exp(-\sigma(g_i(\boldsymbol{p}_j))) > \gamma, \quad \boldsymbol{p}_j \in V_i, V_i \in \mathcal{V},$$

density

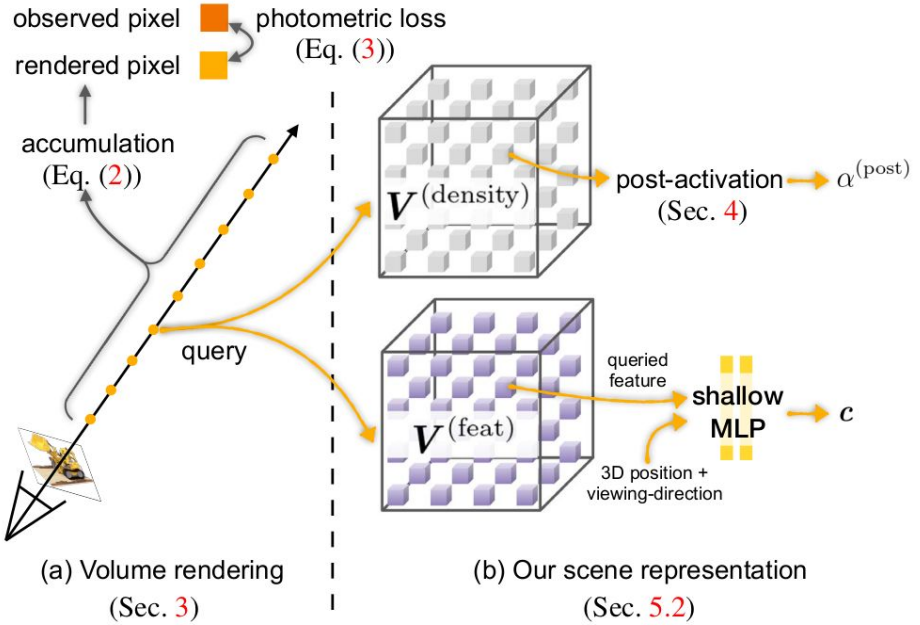
During training progressively scale number of voxels, adding children to octree:

- Halve the size of voxels → Split each voxel into 8 sub-voxels.
- The feature representations of the new vertices are initialized via trilinear interpolation of feature representations at the original eight voxel vertices.
- But training is still long!

Image credit: https://lingjie0206.github.io/papers/NSVF/

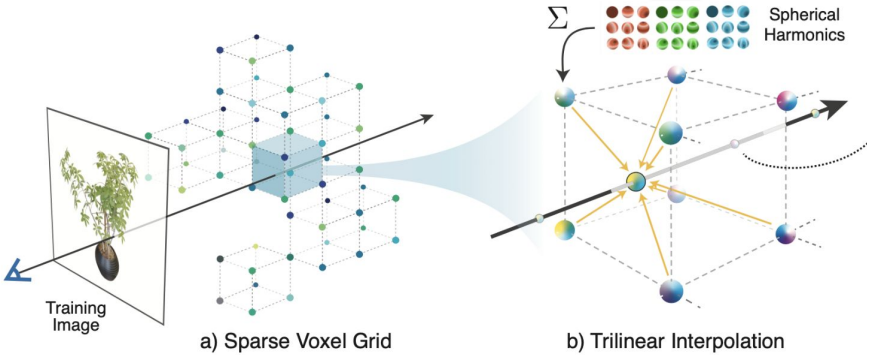
DirectVoxGO

- Trainable encoding rather then positional one
- Sparse array to store encoded parameters
- Pretraining without any Neural Network
- Progessive pruning and scaling of grid with interpolation for novel points
- Training length in minutes, rather than hours



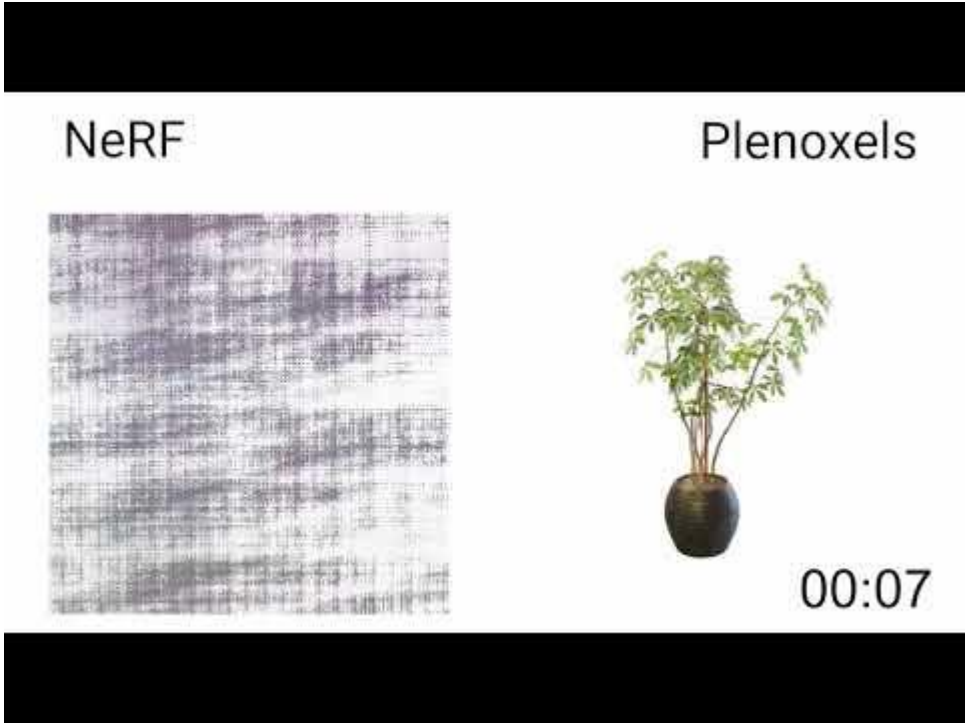
https://arxiv.org/abs/2111.11215

Plenoxels: Combine Spherical Harmonics and Sparse Grids



There is no need in any neural network to achieve the same quality!

	PSNR ↑	SSIM ↑	LPIPS ↓	Train Time
Ours	31.71	**0.958**	**0.049**	**11 mins**
NV [20]	26.05	0.893	0.160	>1 day
JAXNeRF [7, 26]	**31.85**	0.954	0.072	1.45 days



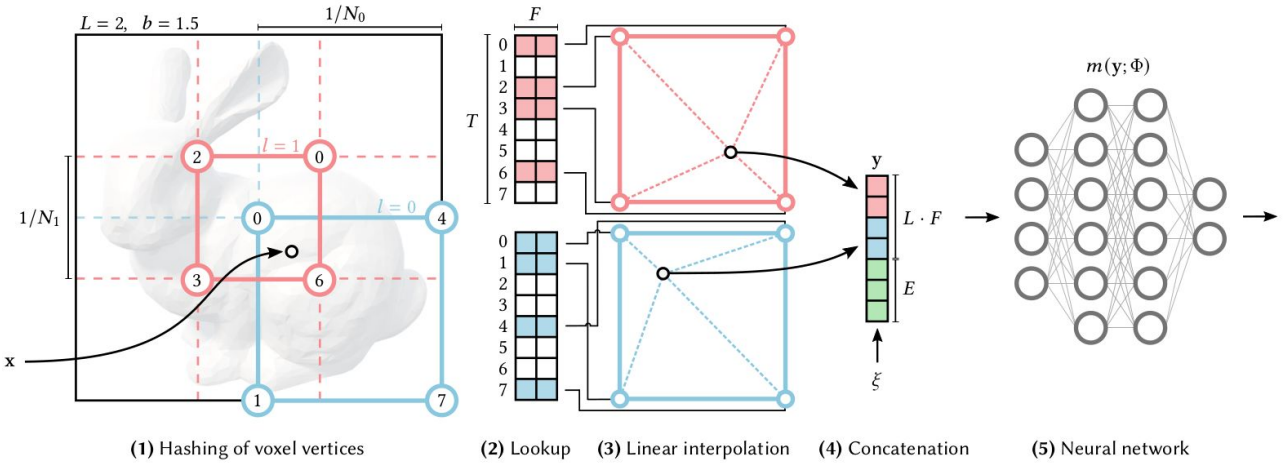
Source: https://alexyu.net/plenoxels/

Instant Neural Graphic Primitives

- Multi-level hash, L - number of voxels

  $\lfloor x * L \rfloor$ or $\lceil x * L \rceil$

- Trilinear interpolation at each level
- Tiny NN that outputs color and density

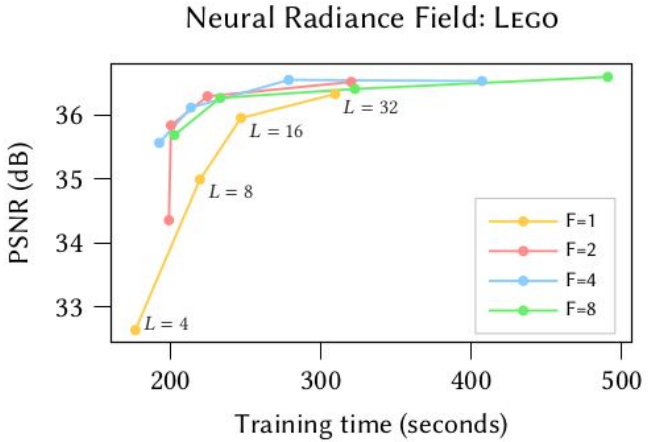


https://nvlabs.github.io/instant-ngp/

Instant Neural Graphic Primitives

F - feature vector size, L - encoding levels

Very fast training - in several minutes!



No encoding

Positional encoding

Hash-based encoding

https://nvlabs.github.io/instant-ngp/

Our approach to integration

Integral as Expectation Along Ray

We can view previously mentioned integral as expectation along ray:

$$I(t_T) = \int_{t_0}^{t_T} T(s)\sigma(s)c(s)ds = \mathbf{E}_{p(s)=\sigma(s)T(s)}c(s) \quad T(t) = \exp\left(-\int_{t_0}^{t}\sigma(s)ds\right)$$

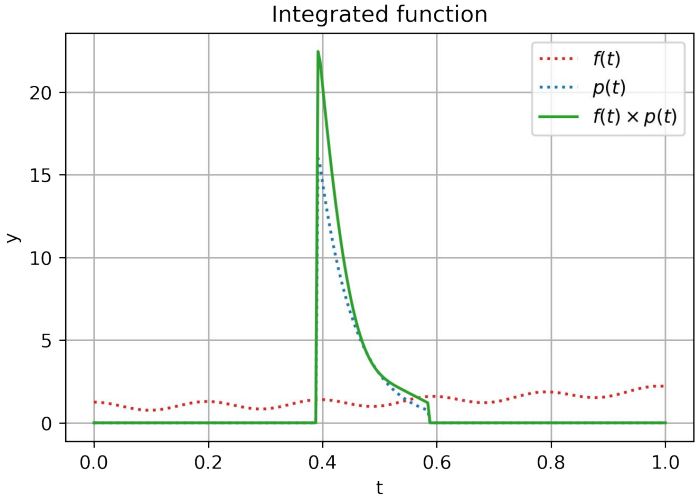
Than opacity $O(t) = 1 - T(t)$ became a cdf of corresponding distribution:

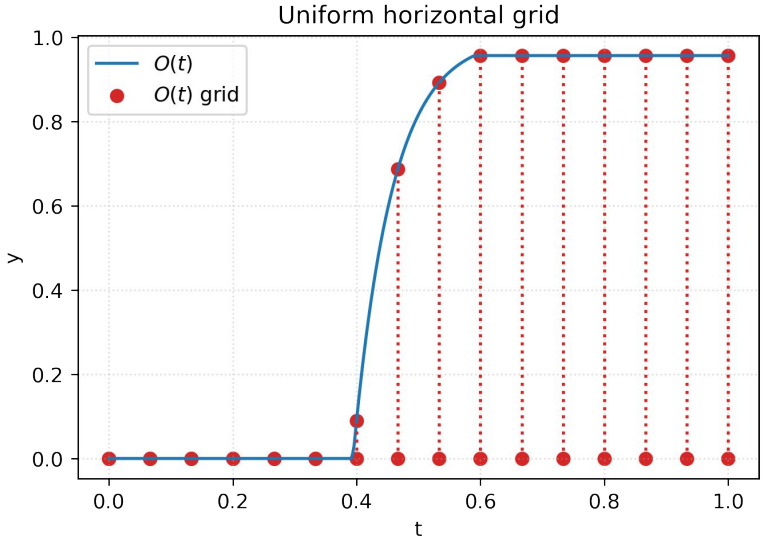
$$\frac{dO(t)}{dt} = -\exp\left(-\int_{t_0}^{t}\sigma(s)ds\right)(-\sigma(t)) = p(t)$$

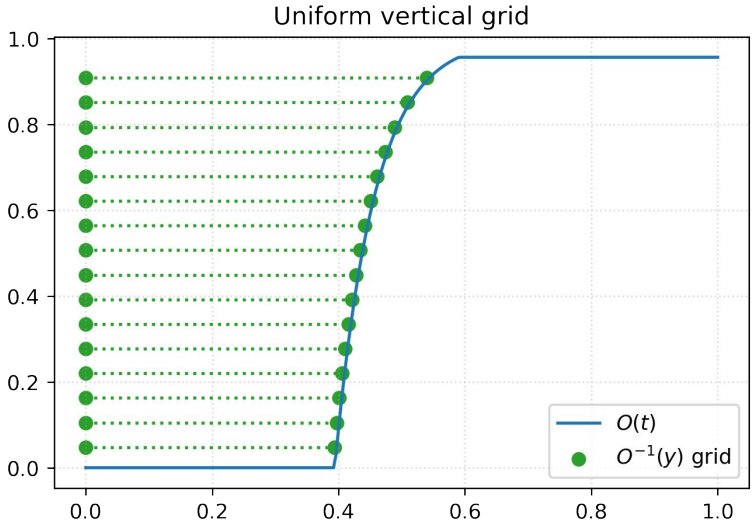
Our Intuition

$$\int_0^T f(t)p(t)\mathrm{d}t$$

$O(t)$ is CDF







Change of Variables

It would be great if we could sample from distribution induced by $\sigma(t)$. This can be done through the change of variables:

$$I(t_T) = \int_{t_0}^{t_T} \sigma(s)c(s)T(s)ds = \int_{y_{min}}^{y_{max}} c(O^{-1}(y))dy; \text{where } y(t) = O(t)$$

If we do so we will avoid empty areas by integration procedure.

What Problems Do We Have in Computing the Integral?

Problem	Solution
How to invert opacity?	This is a growing function, than we can use binary search
How to calculate opacity, there is an integral: $$O(t) = 1 - \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right)$$	In some cases integral can be calculated explicitly: 1) Sigma is a linear layer with some encoding and square activation 2) Sigma is piecewise linear
What to do with gradients, if we use binary search?	Next slide ->

Gradient Computation: Implicit Function theorem

For function: $y = O(t, \Phi)$, we want obtain gradient of ***t*** with respect to ***Φ***

Full gradient:

$$\frac{dy}{d\Phi} = 0 = \frac{\partial O(t,\Phi)}{\partial \Phi} + \frac{\partial O(t,\Phi)}{\partial t}\frac{\partial t}{\partial \Phi}$$

Simplification:

$$\frac{\partial t}{\partial \Phi} = -\frac{\frac{\partial O}{\partial \Phi}}{\frac{\partial O}{\partial t}} = -\frac{\frac{\partial}{\partial \Phi}\int_0^t \sigma(s)\mathrm{d}s}{\sigma(t)}$$

Implicit reparametrization for gradients https://arxiv.org/abs/1805.08498

Approximation Procedure: Step-by-step

NeRF	Our (Trapezoidal Rule & Stratified Sampling)
Estimate intersections ray and scene: t_0 and t_T	Find borders as: $y_{min} = O(t_0), y_{max} = O(t_T)$
Sample grid between borders: $t_j = t_0 + j\frac{t_T - t_0}{n}$ (or use importance sampling based on previously sampled grid)	Sample grid between borders $y_j = y_{min} + j\frac{y_{max} - y_{min}}{n}$ or unbiased noise: $Y_i \sim U[y_i, y_{i+1}]$
Calculate quadrature approximation over t: $\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i\big(1 - \exp(-\sigma_i\delta_i)\big)\mathbf{c}_i$	Calculate approximation over y: Trapezoidal: $I(t_T) = \sum_{i=1}^{N-1}(y_{i+1} - y_i)\frac{c(O^{-1}(y_{i+1})) + c(O^{-1}(y_i))}{2}$ Unbiased: $I(t_T) = \sum_{i=1}^{N-1}(y_{i+1} - y_i)c(O^{-1}(Y_i))$

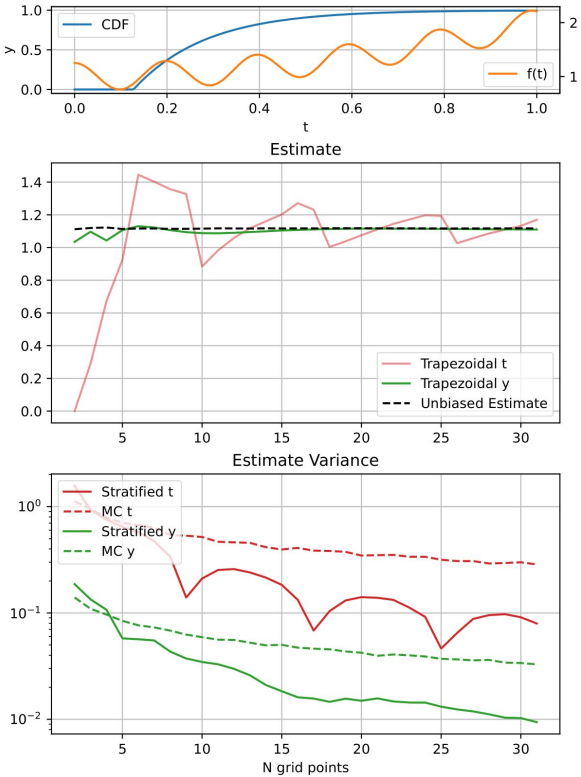
Example: Piecewise Linear Sigma

Our sigma has following form:

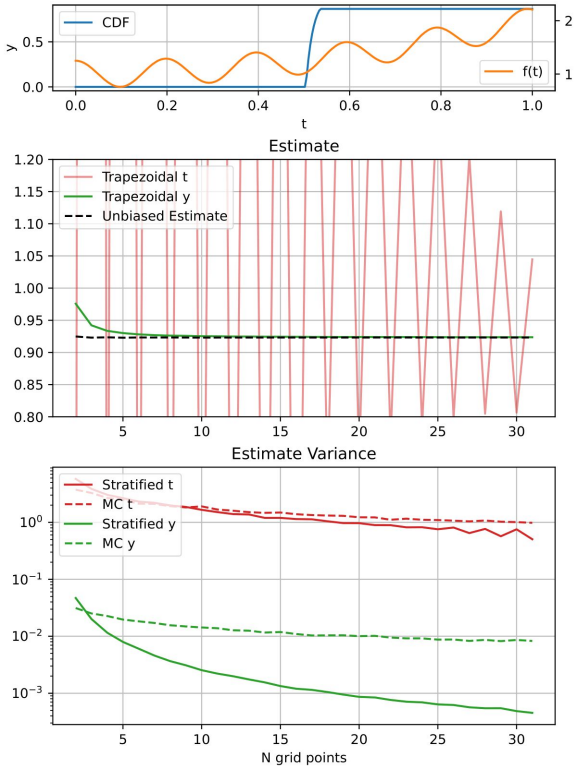
$$\sigma(x) = \chi(\text{Softplus}(w_1) \dots \text{Softplus}(w_8))$$

- This is exactly what was done in DirectVoxGo, Plenoxels and Instant-NGP
- We can force positive weights with softplus and avoid any activation function
- It’s easily integrable, so we can calculate opacity
- We can use either octree or sparse voxel grid or hash-table to store weights

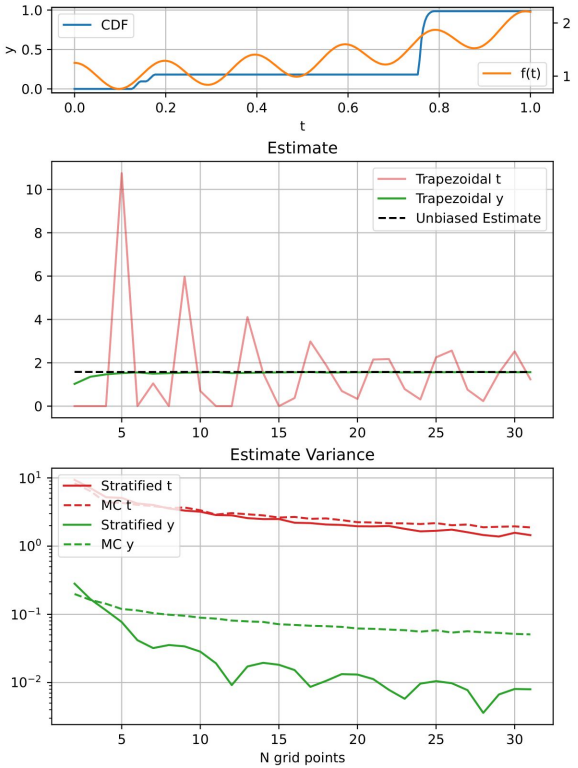
Comparison of Approximations with Piecewise Linear 1d



pdf “fog”



pdf “wall”



pdf “wall and glass”

Linear Layer and Square Activation

Our sigma has following form: $\sigma(x) = \left[\sum_{i=1}^{N} w_i \cos(u_i x + v_i)\right]^2$

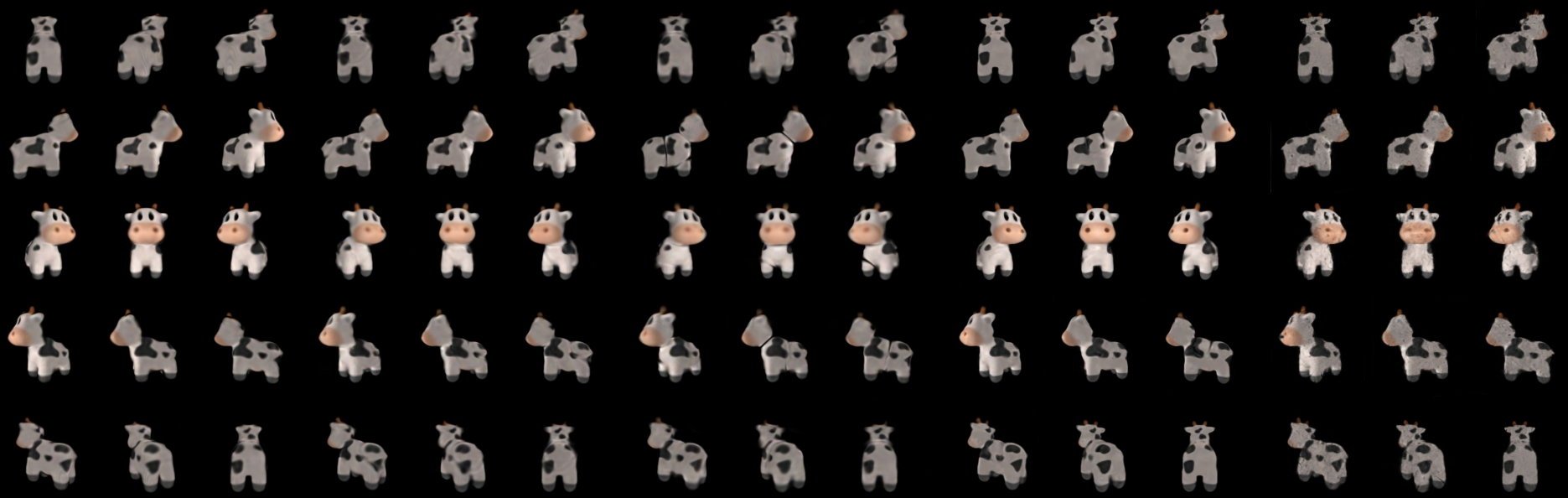
Then we can calculate opacity as following:

$$O(t) = 1 - \exp\left(-\int_{t_0}^{t} \sigma(s)ds\right) = 1 - \exp\left(-\sum_{i=1}^{N}\sum_{j=1}^{N} w_i w_j \int_{t_0}^{t} \cos(a_i s + b_i)ds\right)$$

Here is $O(N^2)$ summands, but we can use fast fourier transform if our input encoding has a special form (u_i from grid)

Lazy computations allow more efficient memory usage!

Results: Linear + Square



Orig nerf (simplified)
PSNR 27.46

Orig nerf + square activation
PSNR 27.82

Fourier features + square activation
PSNR 26.69

Non-spatial, 15 features
PSNR 29.04

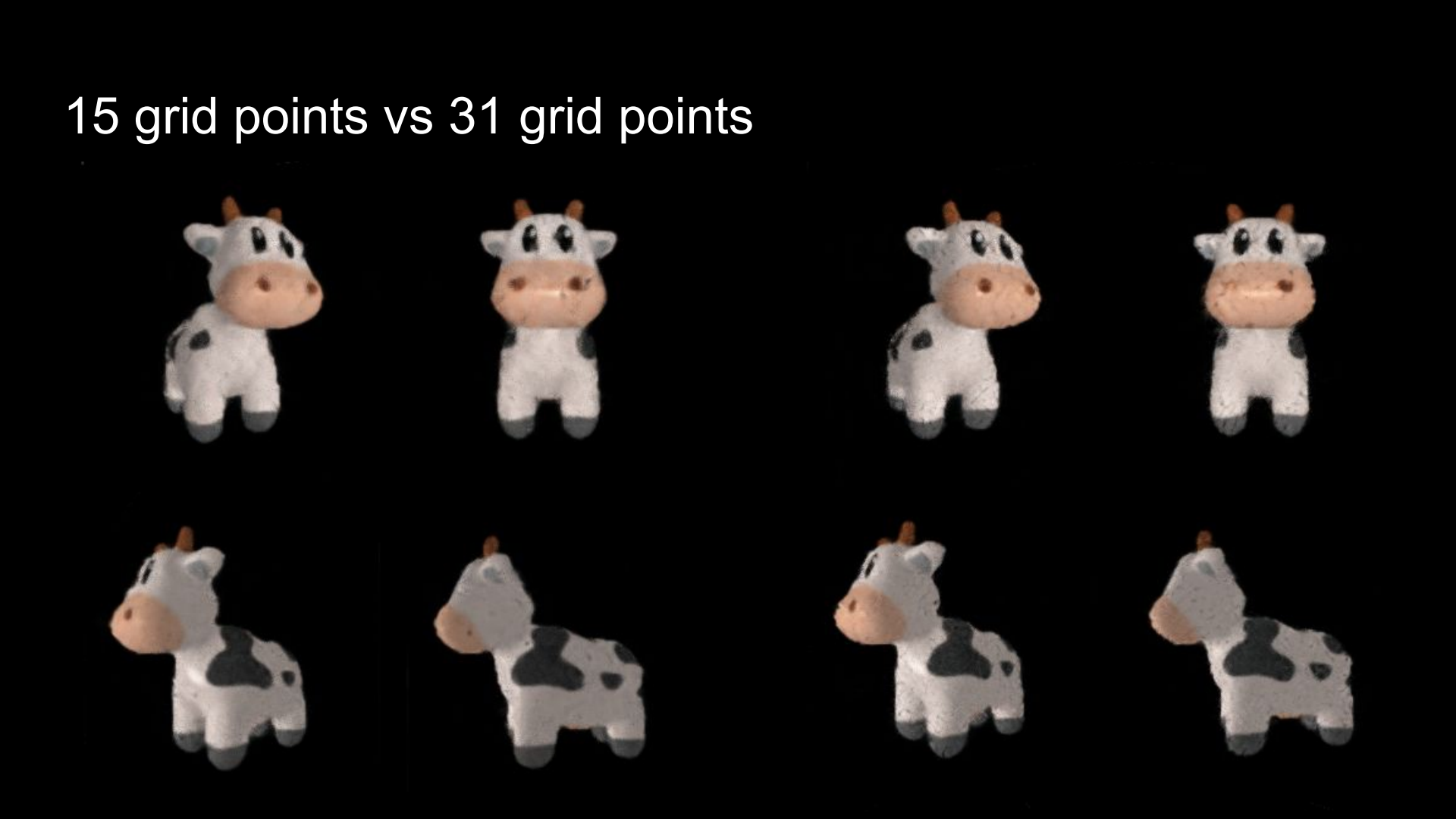
Spatial progressive upsampling
PSNR 29.34

Future Plans

- Use instant neural graphic primitives as MLP with encoding
- Test piecewise linear integration for real data, using DirectVoxGo/Plenoxels and Instant-NGP

More Images

15 grid points vs 31 grid points



Poorly Drawn Lego Truck