CMSC740
Advanced Computer Graphics

Matthias Zwicker
Fall 2025

Limitations of NeRF

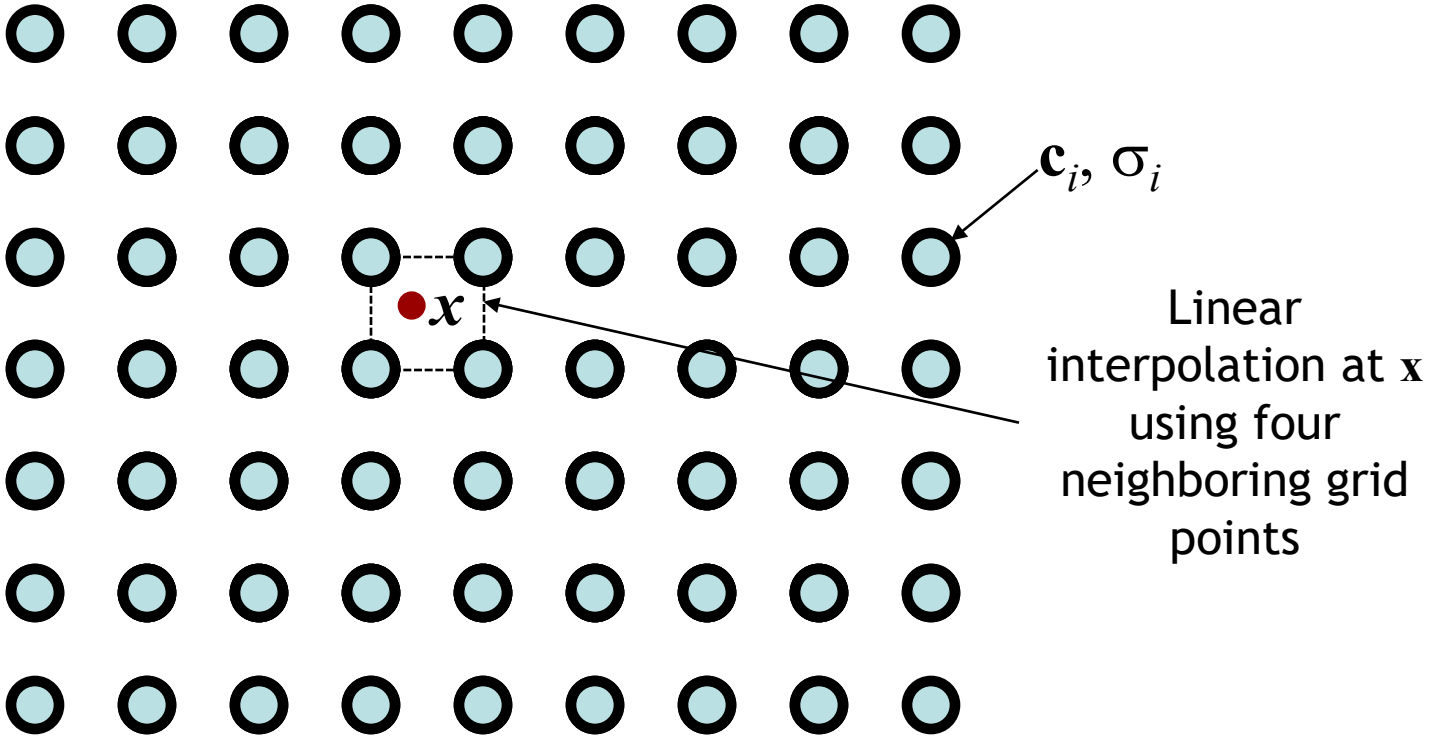
- Doesn't attempt to represent surface directly (only density σ)
  - SDF surface: NeuS, https://lingjie0206.github.io/papers/NeuS/
- Treats pixels as infinitesimal rays, doesn't take into account pixel areas
  - Extended spatial encoding to capture beam volume: MipNeRF, https://jonbarron.info/mipnerf/
- Only works for static scenes
  - Include temporal space deformation: Nerfies, https://nerfies.github.io/
- Doesn't take into account potential appearance variation in input images (different illumination, time, time of year, etc.)
  - Latent appearance vectors: NeRF in the wild, https://nerf-w.github.io/
- Doesn't recover BRDF parameters and illumination
  - Approximate model to better capture glossy reflection: RefNeRF, https://dorverbin.github.io/refnerf/
- Requires known camera parameters
- Training and rendering slow
- Requires many input images

Slow training and rendering

- Training/evaluating neural network for radiance and density is slow
  - Hundreds of network evaluations per ray, millions of rays per image
- Alternative representations for radiance, density that are faster to evaluate?
  1. Grid-based with interpolation function (e.g. linear)
  2. Radial (or elliptical) basis functions

Grid-based representation

- Parameters (to be optimized): radiance, density values at grid points $\mathbf{c}_i$, σ_i
- Evaluation at arbitrary point $\boldsymbol{x}$:
  $\mathbf{c}(\boldsymbol{x})$ = linear interpolation of $\{\mathbf{c}_i \mid i$ in neighborhood of $\boldsymbol{x}\}$

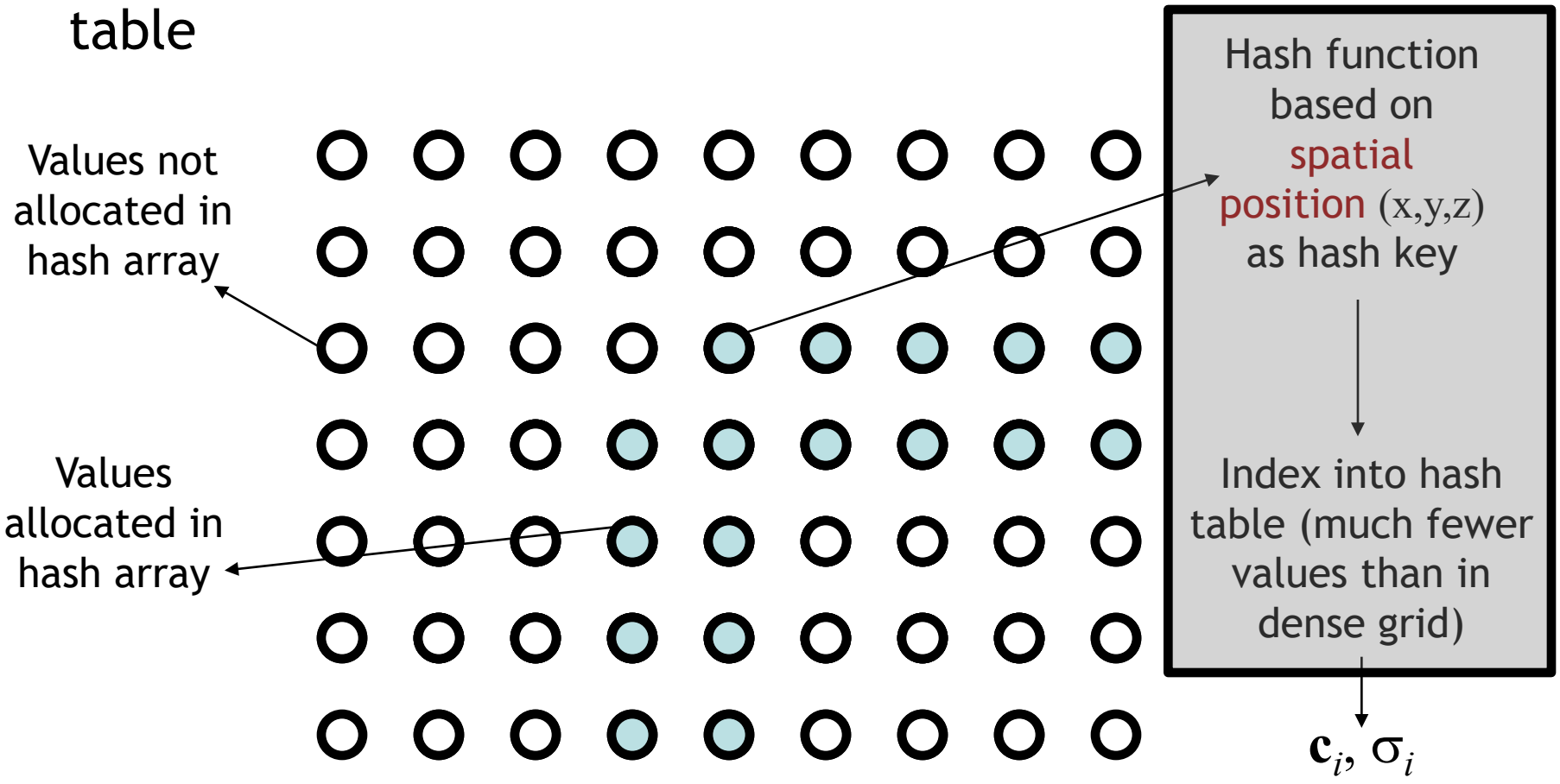


Challenges

- Dense grid requires a lot of memory
- How to pick appropriate grid resolution
- How to represent directional dependence of radiance $\boldsymbol{c}(\mathbf{x}, \mathbf{d})$, $\mathbf{d}$ ray direction

Sparse grids using spatial hasing

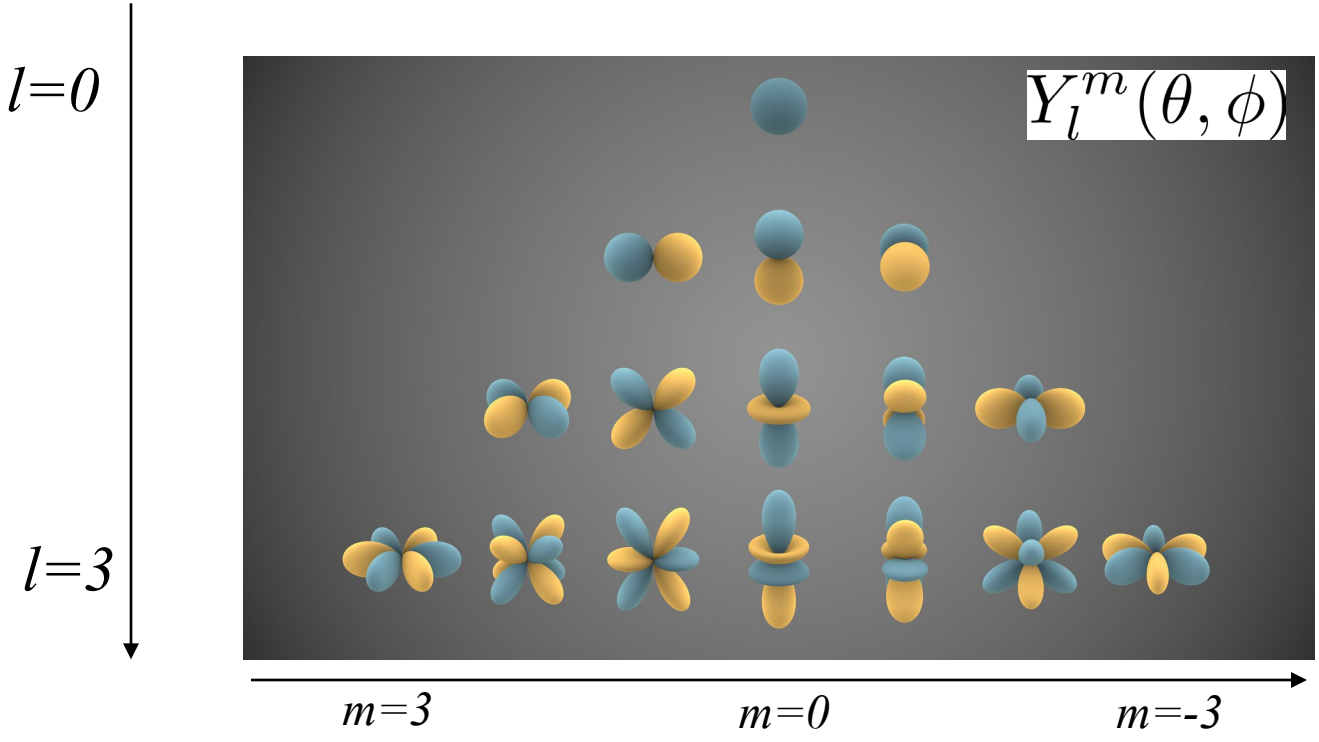
- Allocate storage for grid vertices only where needed (non-zero values)
- Spatial hashing: indirect lookup of grid value through hash table



Spatial hasing popular for many applications in computer graphics

Directional dependence: spherical harmonics

- Spherical harmonics: generalization of Fourier basis functions (sines, cosines of increasing frequencies) to sphere of directions $Y_l^m(\theta, \phi)$
  - "Frequency" l, "phase" m, spherical angles θ, ϕ



Blue: positive
Yellow: negative
Distance of surface from origin: absolute value

Directional dependence: spherical harmonics

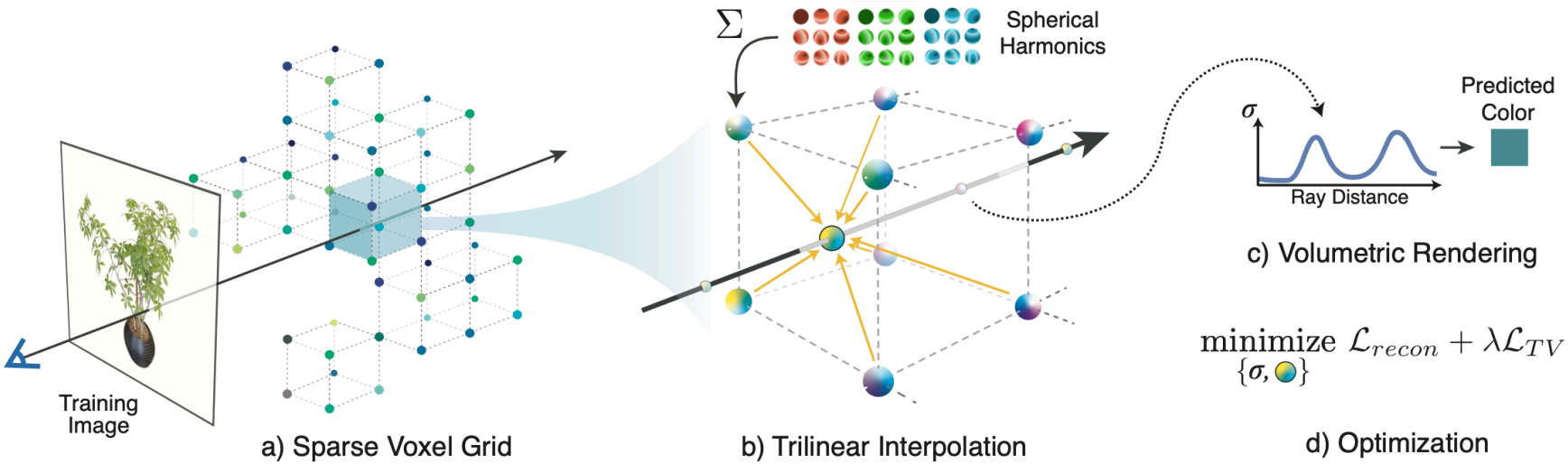
- Represent functions f (e.g. radiance) over the sphere using weighted sum of spherical harmonics
  - Weights (coefficients) f^m_l are parameters of representation
  - Using more coefficients (more spherical harmonics) enable higher frequencies

$$f(\theta,\phi) = \sum_{l=0}^{k} \sum_{m=-l}^{l} f_l^m Y_l^m(\theta,\phi)$$

- To represent functions with directional dependence, e.g. radiance $\mathbf{c}(x,y,z,\theta,\phi)$, store and optimize vector of coefficients f^m_l per grid point
  - Typically $k<=3$ (<=16 coefficients)

Plenoxels: Radiance Fields without Neural Networks

- CVPR 2022, https://alexyu.net/plenoxels/



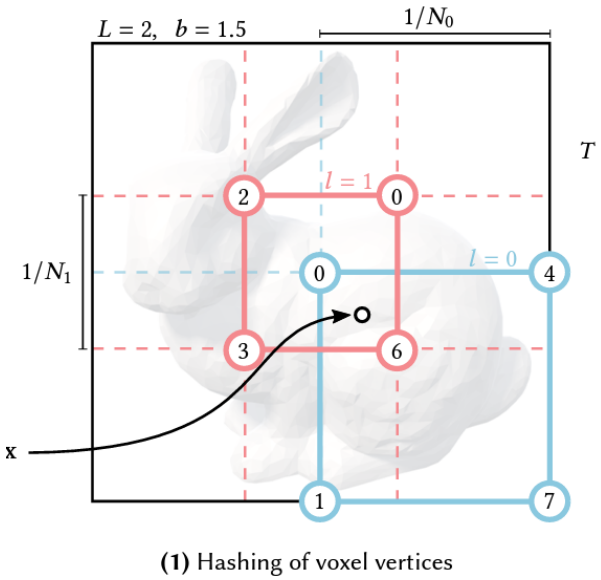
	PSNR ↑	SSIM ↑	LPIPS ↓	Train Time
Ours	31.71	**0.958**	**0.049**	**11 mins**
NV [20]	26.05	0.893	0.160	>1 day
JAXNeRF [7, 26]	**31.85**	0.954	0.072	1.45 days
Ours	26.29	**0.839**	**0.210**	**24 mins**
LLFF [25]	24.13	0.798	0.212	—*
JAXNeRF [7, 26]	**26.71**	0.820	0.235	1.62 days
Ours	20.40	**0.696**	**0.420**	**27 mins**
NeRF++ [57]	**20.49**	0.648	0.478	~4 days

Hybrid: grid/neural network

- Store learnable feature vectors on grid (instead of radiance or density)
- Use small neural network that takes feature vectors as input to produce desired output quantity (radiance, density, etc.)
- Potential advantages
  - Fast training due to smaller network
  - More accuracy due to feature vectors stored on spatial grid

Hybrid: grid/neural network

- Instant Neural Graphics Primitives (NGP)
  https://nvlabs.github.io/instant-ngp/

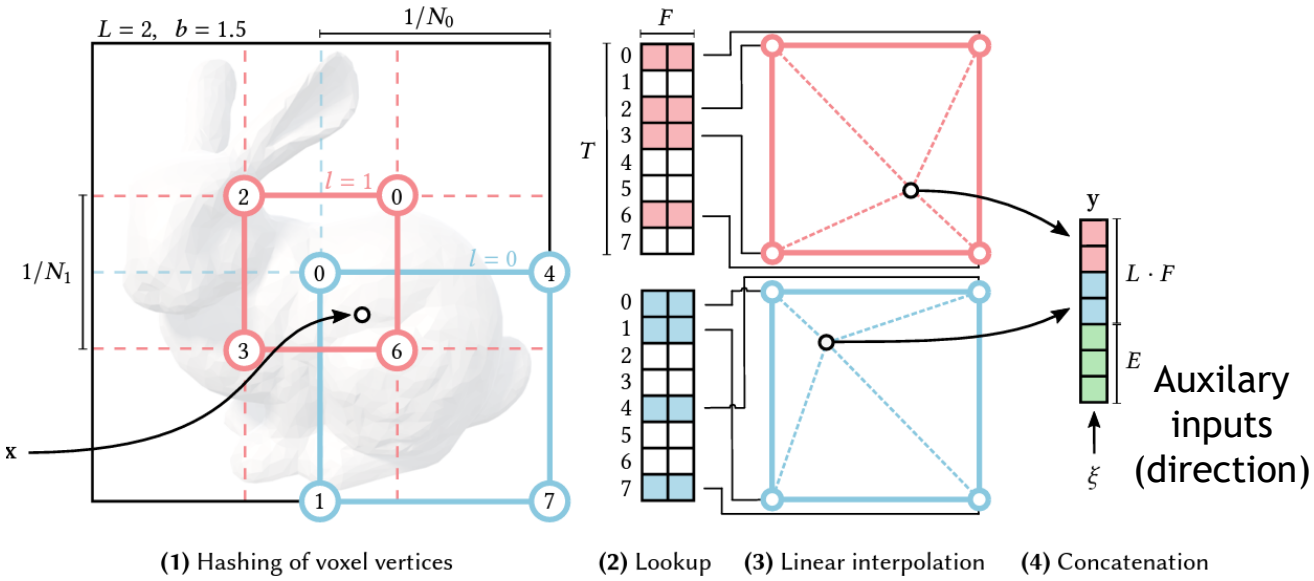


(1) Hashing of voxel vertices

Multi-resolution grid storing learnable feature vectors using hashing (blue: coarse grid; pink: finer grid)C

Hybrid: grid/neural network

- Instant Neural Graphics Primitives (NGP)
  https://nvlabs.github.io/instant-ngp/

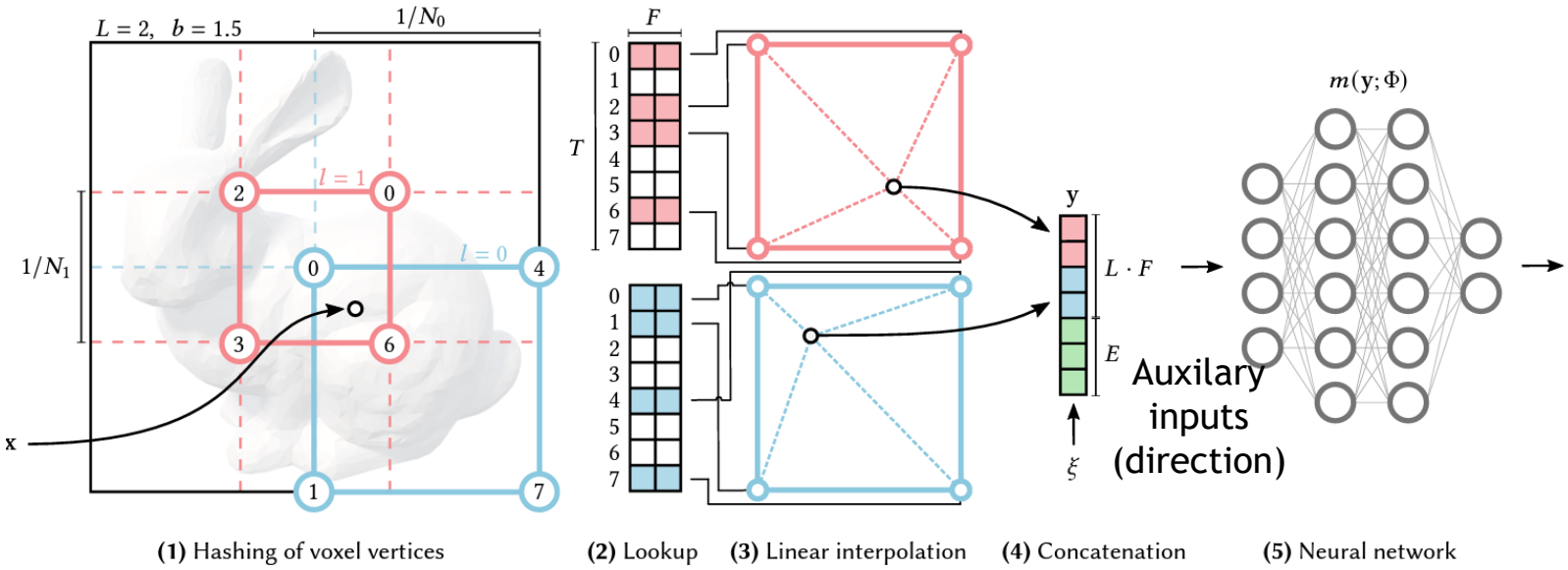


Multi-resolution grid storing learnable feature vectors using hashing (blue: coarse grid; pink: finer grid)C

Sparse grid storage using spatial hashing, linear interpolation; concatenation of feature vectors over multiple resolutions

Hybrid: grid/neural network

- Instant Neural Graphics Primitives (NGP)
  https://nvlabs.github.io/instant-ngp/



Multi-resolution grid storing learnable feature vectors using hashing (blue: coarse grid; pink: finer grid)C

Sparse grid storage using spatial hashing, linear interpolation; concatenation of feature vectors over multiple resolutions

Small neural network to predict desired quantity (radiance, density)

Instant NGP

- Well-optimized implementation https://nvlabs.github.io/instant-ngp/
- Much faster than original NeRF
- No collision handling for spatial hashing, still works, see paper
- Hybrid sparse multiresolution hash grid/neural network better than spatial encoding (higher quality, faster)

	Mic	Ficus	Chair	Hotdog	Materials	Drums	Ship	Lego	avg.
Ours: Hash (1 s)	26.09	21.30	21.55	21.63	22.07	17.76	20.38	18.83	21.202
Ours: Hash (5 s)	32.60	30.35	30.77	33.42	26.60	23.84	26.38	30.13	29.261
Ours: Hash (15 s)	34.76	32.26	32.95	35.56	28.25	25.23	28.56	33.68	31.407
Ours: Hash (1 min)	35.92	33.05	34.34	36.78	29.33	25.82	30.20	35.63	32.635
Ours: Hash (5 min)	36.22	33.51	35.00	37.40	29.78	26.02	31.10	36.39	33.176
mip-NeRF (~hours)	36.51	33.29	35.14	37.48	30.71	25.48	30.41	35.70	33.090
NSVF (~hours)	34.27	31.23	33.19	37.14	32.68	25.18	27.93	32.29	31.739
NeRF (~hours)	32.91	30.13	33.00	36.18	29.62	25.01	28.65	32.54	31.005
Ours: Frequency (5 min)	31.89	28.74	31.02	34.86	28.93	24.18	28.06	32.77	30.056
Ours: Frequency (1 min)	26.62	24.72	28.51	32.61	26.36	21.33	24.32	28.88	26.669

Image reconstruction quality, PSNR

Bottom two rows: their approach, but spatial encoding instead of hash grid

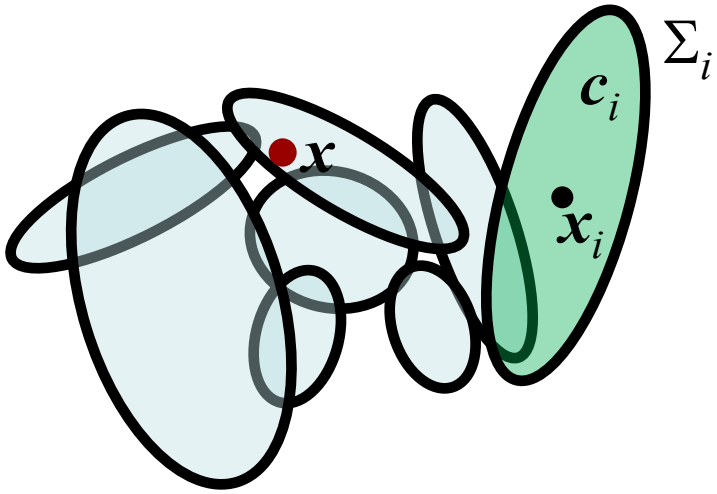
Slow training and rendering

- Training/evaluating neural network for radiance and density is slow
- Alternative representations for radiance, density that are faster to evaluate?
  1. Grid-based with interpolation function (e.g. linear)
  2. Radial (or elliptical) basis functions

Radial basis functions

- Function represented as weighted sum of simple (radially symmetric) basis functions centered at irregular locations $\mathbf{x}_i$ https://en.wikipedia.org/wiki/Radial_basis_function
- Here: N Gaussian basis functions G, covariance Σ_i, centered at locations $\mathbf{x}_i$, to represent radiance $\mathbf{c}(\mathbf{x})$
- Parameters (to be optimized): coefficients $\mathbf{c}_i$, covariances Σ_i, centers $\mathbf{x}_i$

$$\mathbf{c}(\mathbf{x}) = \sum_{i=1}^{N} \mathbf{c}_i G_{\Sigma_i}(\mathbf{x} - \mathbf{x}_i)$$

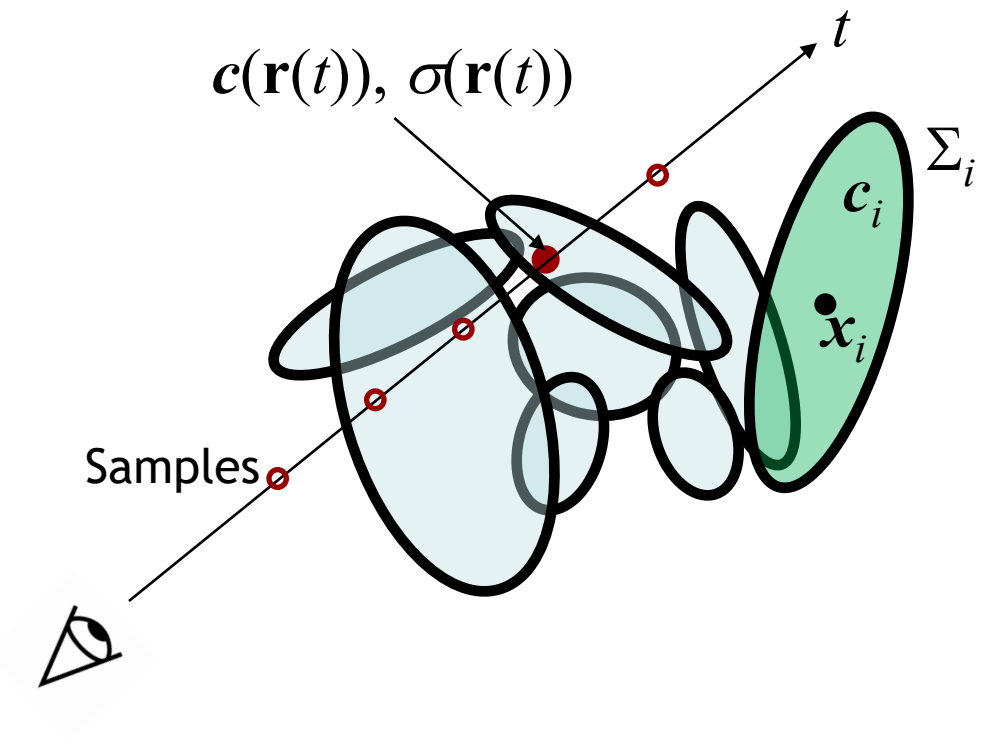


Volumetric rendering of Gaussians

- Same volume rendering model as before (NeRF)

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

- Color and density values $\mathbf{c}(\mathbf{r}(t)) = \sum_{i=1}^{N} \mathbf{c}_i G_{\Sigma_i}(\mathbf{r}(t) - \mathbf{x}_i)$



Volumetric rendering of Gaussians: ray marching

- Discretization: can do ray marching as before

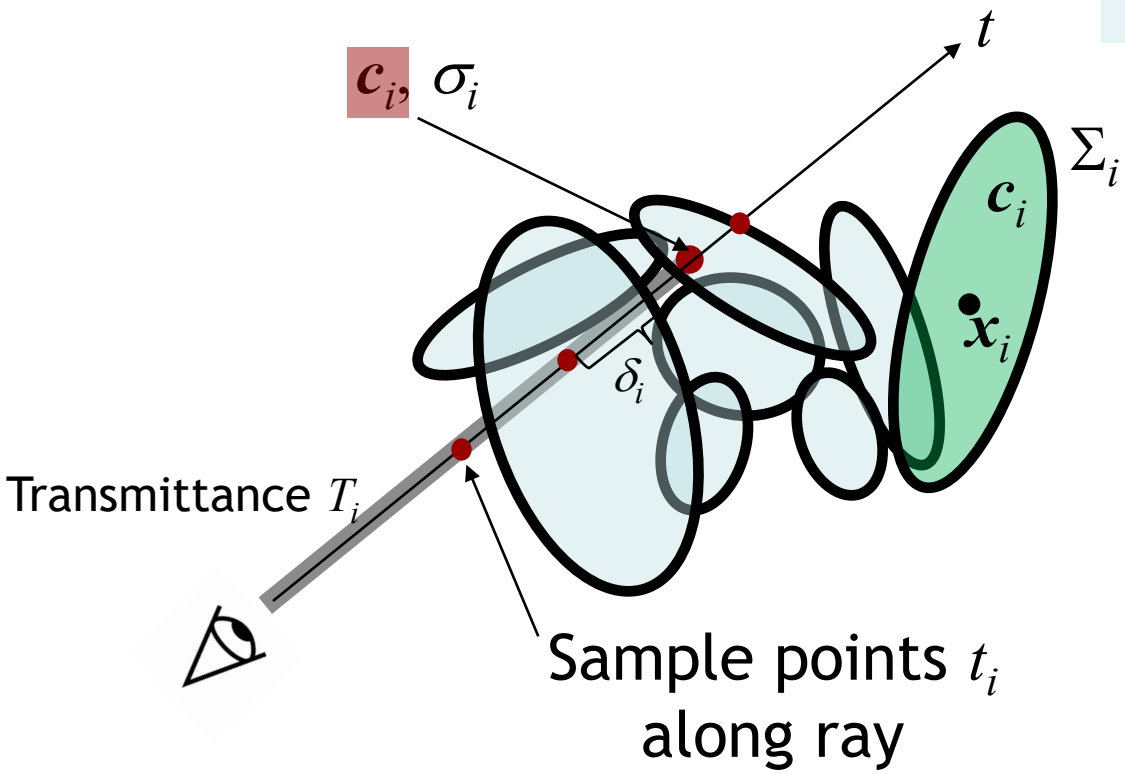
$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i (1 - \exp(-\sigma_i \delta_i)) \mathbf{c}_i \qquad T_i = \exp\left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right)$$

$$\delta_i = t_{i+1} - t_i$$

Same as original NeRF

$$\mathbf{c}_i = \sum_{i=1}^{N} \mathbf{c}_i G_{\Sigma_i}(\mathbf{r}(t_i) - \mathbf{x}_i)$$

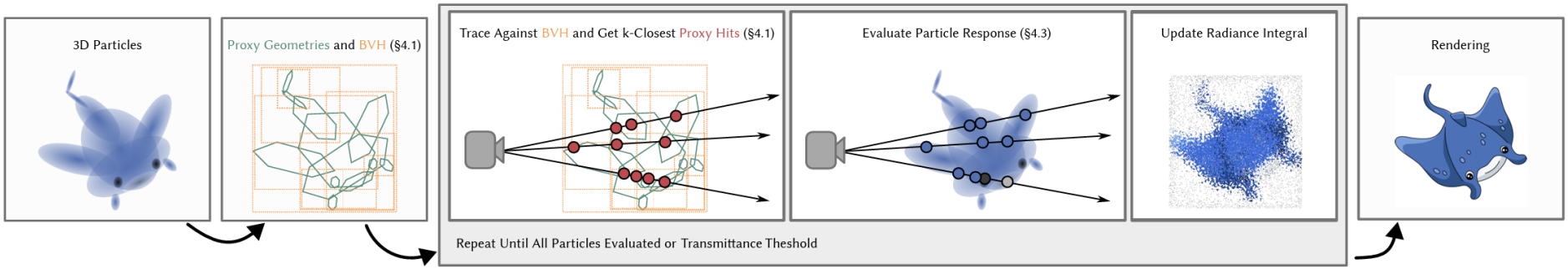
Gaussians instead of neural network, similar for density σ_i



For efficient evaluation, need spatial data structure to avoid summing over all N Gaussians at each sample point

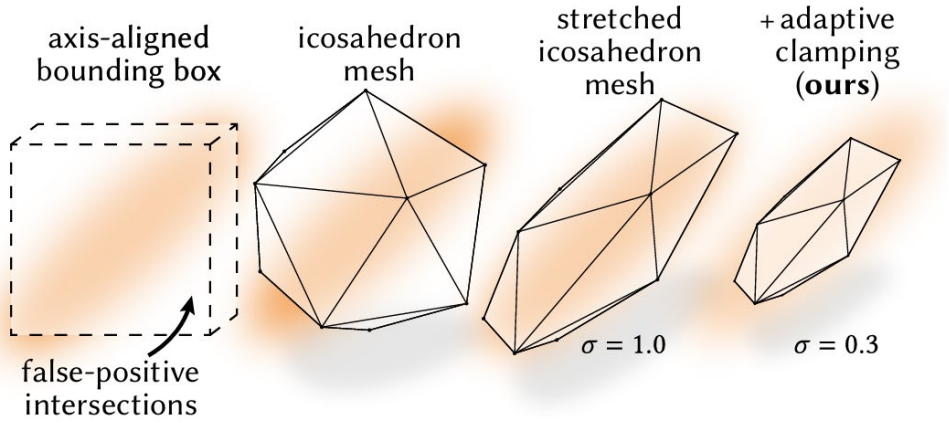
Volumetric rendering of Gaussians: ray marching

- Efficient ray marching requires spatial data structure (bounding volume hierarchy, BVH), proxy geometries for Gaussians



https://gaussiantracer.github.io/

- Potential proxy geometries for Gaussians



Volumetric rendering of Gaussians

- Implementation of ray tracing approach: 3D Gaussian Ray Tracing: Fast Tracing of Particle Scenes https://gaussiantracer.github.io/
- Advantage: light transport effects that require ray tracing (shadows, reflections, etc.)

Using ray tracing to include mirror reflection, reflection/refraction, depth-of-field blur, adding objects

Volumetric rendering of Gaussians: splatting

- 3D Gaussian splatting https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/
- Alternative rendering algorithm: instead of ray tracing/marching, use rasterization [Zwicker 2001]
- Loop over Gaussian in front to back order
  - Project 3D Gaussians one-by-one to 2D
  - Calculate integral along viewing rays per Gaussian, can be done analytically
  - Alpha-blend projected, integrated Gaussians based on densities
- Requires only single loop over all Gaussians per image (as opposed to per sample point loop over all Gaussians in naïve ray marching approach)
- Main benefit: fast rendering ("Ours": 3D Gaussian splatting)

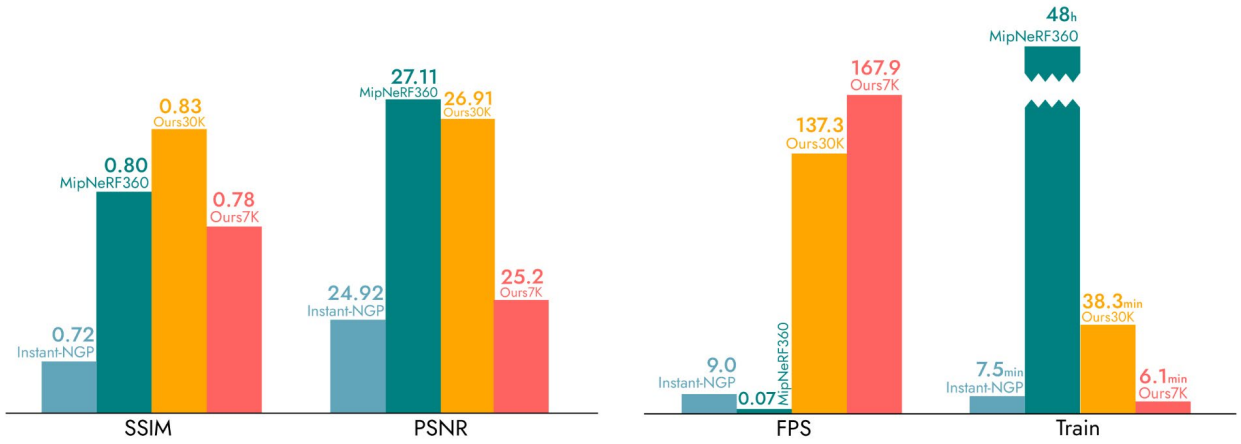


Image quality

Rendering speed

Training time