CSC2457 3D & Geometric Deep Learning

Neural Sparse Voxel Fields

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, Christian Theobalt

Date: March 2, 2021

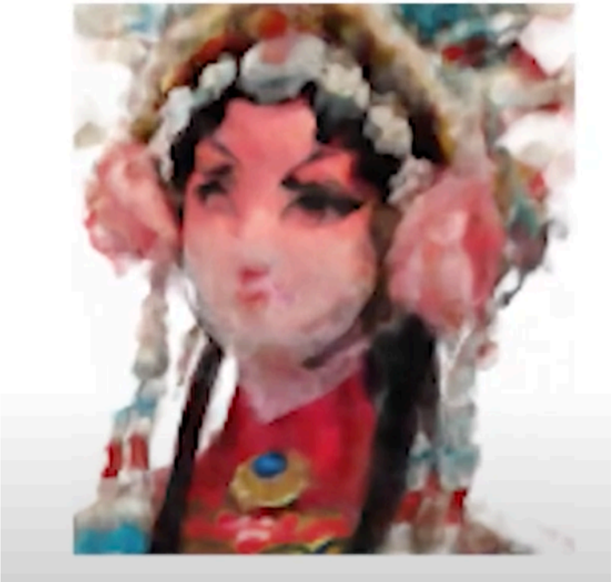
Presenter: Tianxing Li

Instructor: Animesh Garg

UNIVERSITY OF TORONTO

Motivation and Main Problem

- Reminder: we are trying to do realistic novel view synthesis
- We've recently achieved a breakthrough on this task with NeRF
- But … NeRF is not always great, and can be extremely slow



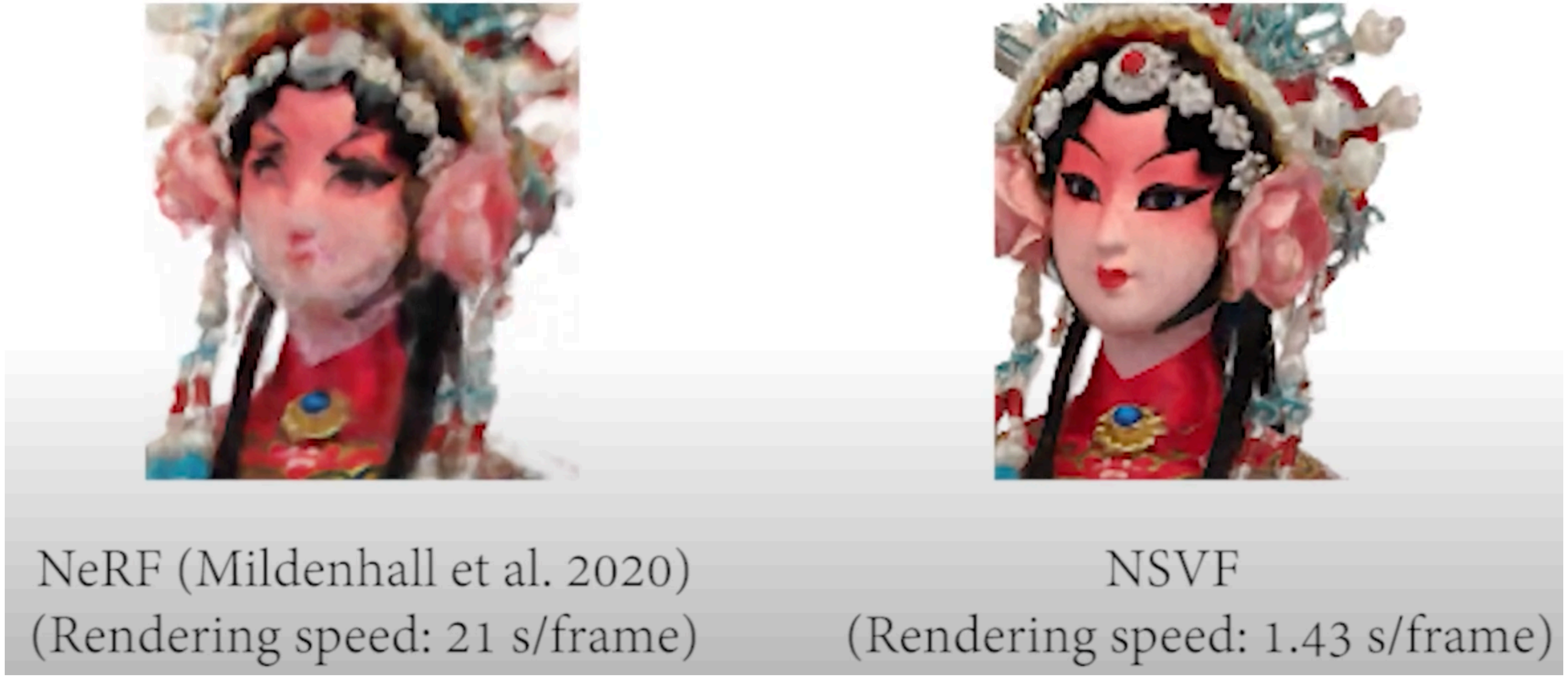
dataset requires 640k rays per image, and our real scenes require 762k rays per image, resulting in between 150 and 200 million network queries per rendered image. On an NVIDIA V100, this takes approximately **30 seconds per frame.**

Applications of Realistic Rendering

Free View Synthesis [Dai et al. 2017]

Motivation and Main Problem

- Good news: NSVF appears a few months after NeRF
- It appears much faster *and* better, what did they do?

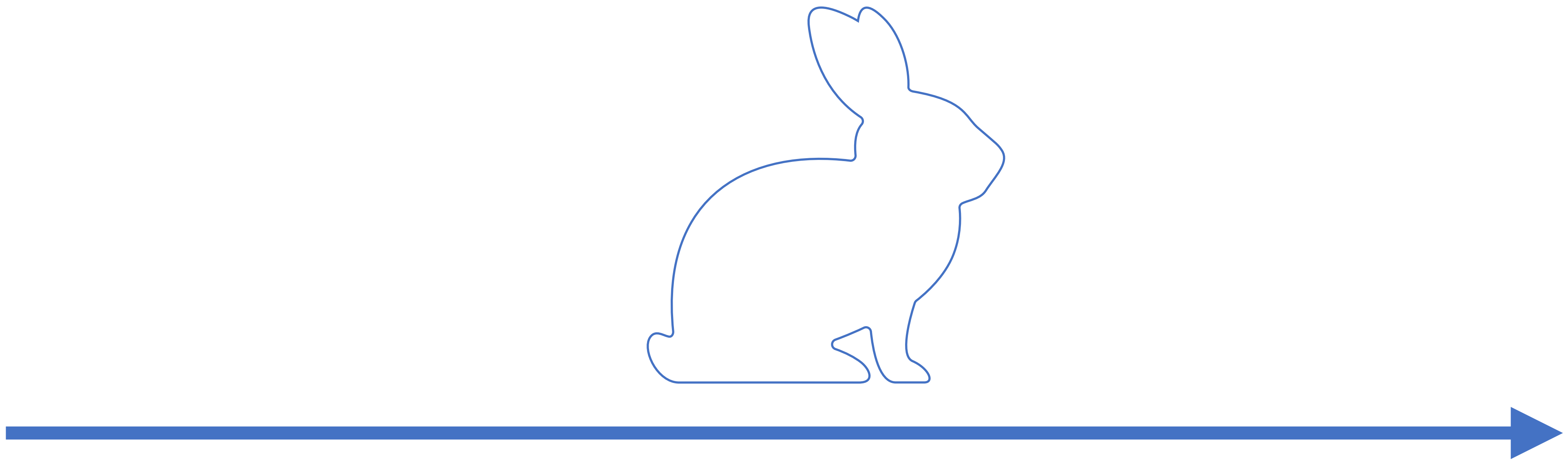


Neural Sparse Voxel Fields (NSVF)

- Fast and high quality novel view synthesis, extends NeRF.
- NeRF wastes a lot of computation by sampling in empty space.
- Key insight: use sparse voxel data structure to enable more precise sampling and detailed modelling of local properties.
- Result: render 10 - 20x faster than NeRF, at higher quality

Recap: NeRF Sampling

For each pixel, we need to integrate colours along its viewing ray



Recap: NeRF Sampling

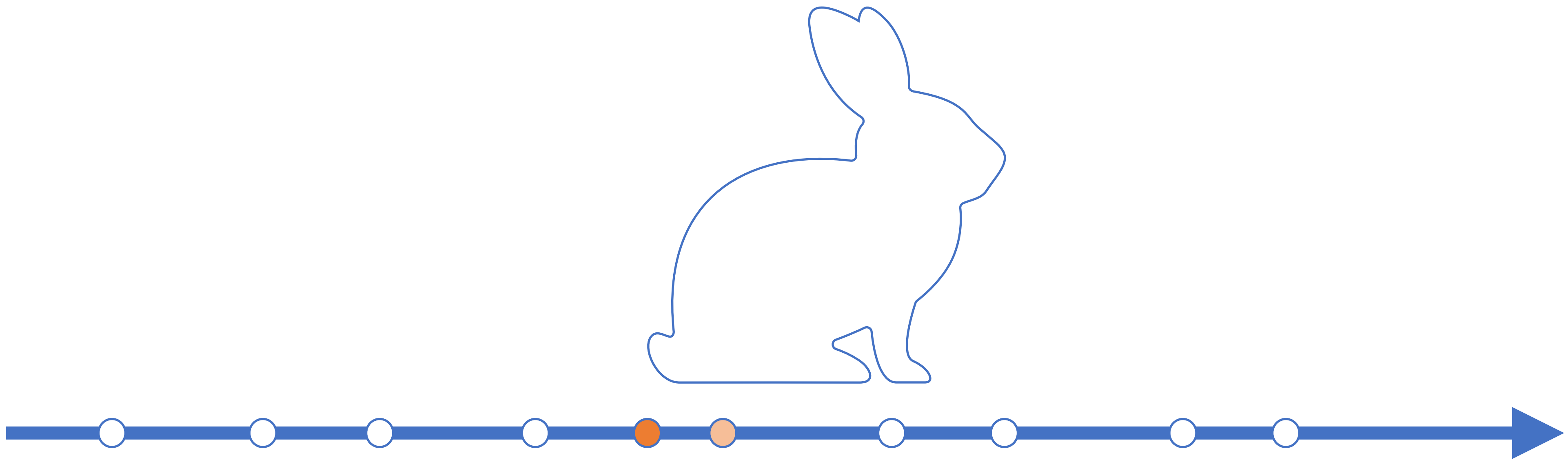
- To make this tractable, we need to sample at the relevant places

- How do we know where to sample?

Recap: NeRF Sampling

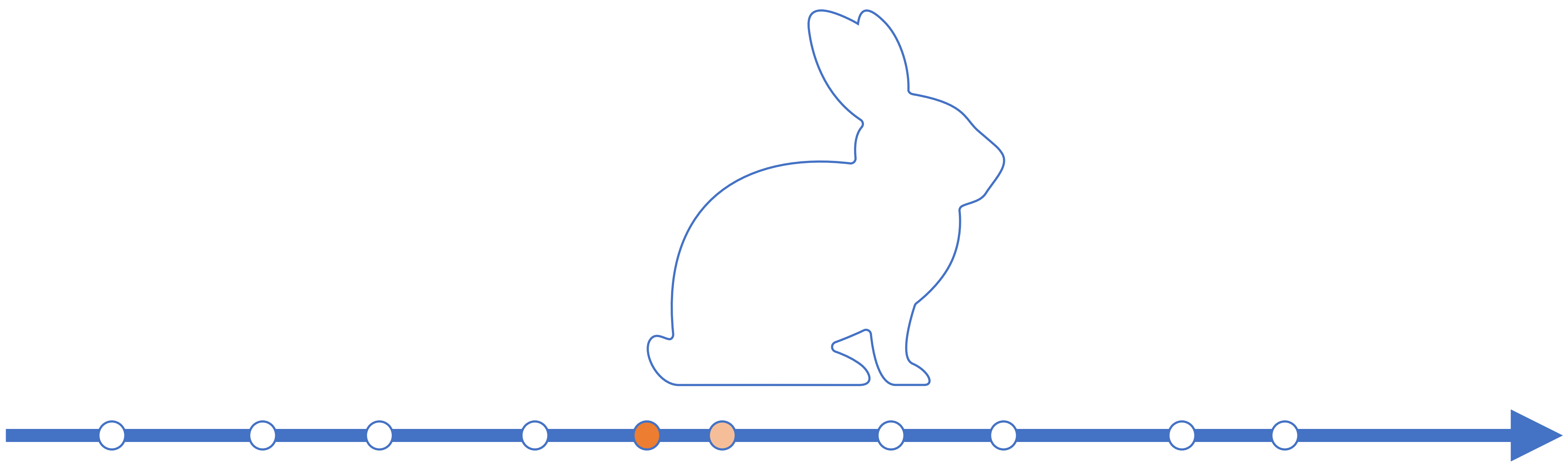
NeRF idea: use coarse sampling to guide next round of sampling

Issue: we use a lot of computation sampling on empty space!



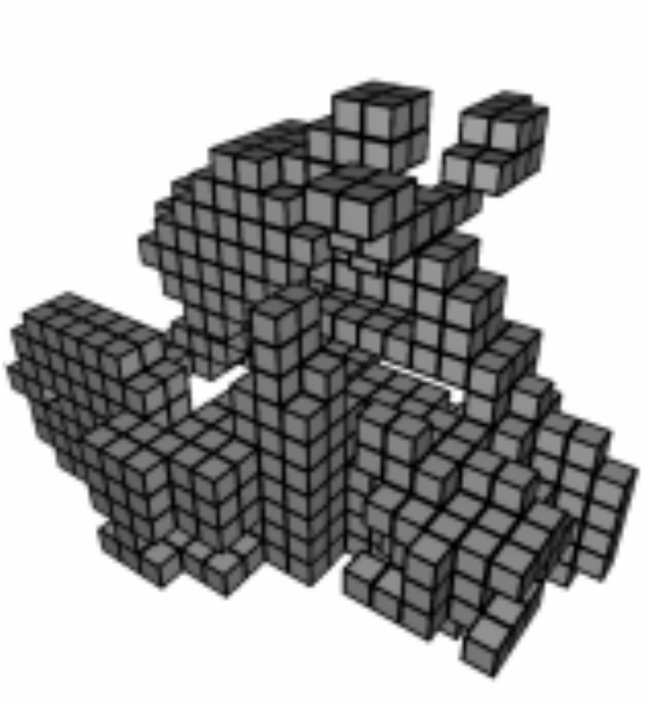
Motivating NSVF

- Can we remember which places are empty?
- And do it with a data structure that is multi-view consistent?



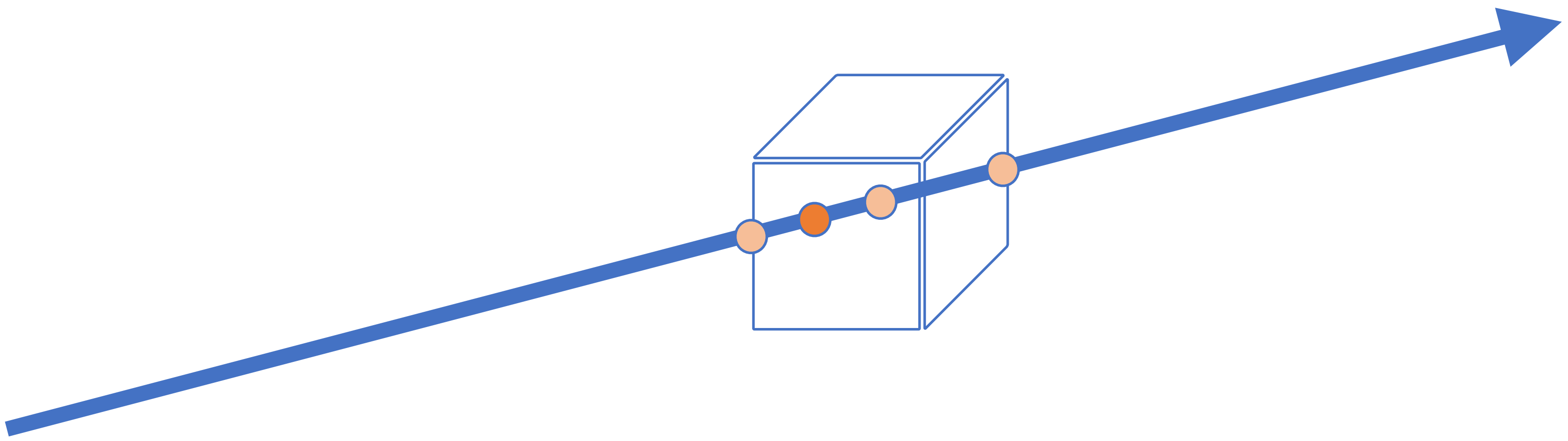
Sparse Voxels

- Use voxels to remember which places we need to sample from
- Note: our scene representation is still continuous



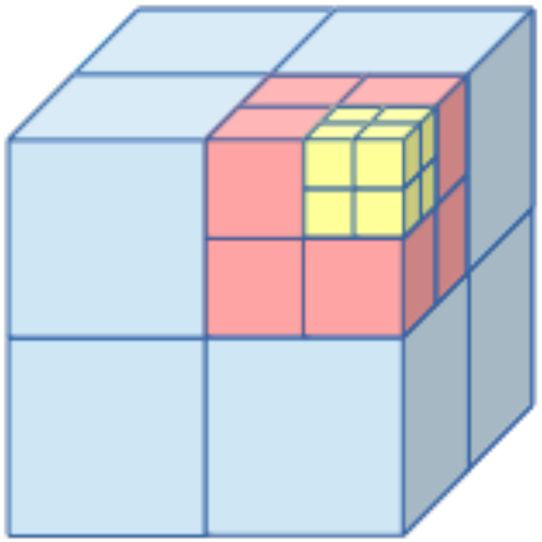
Sparse Voxels

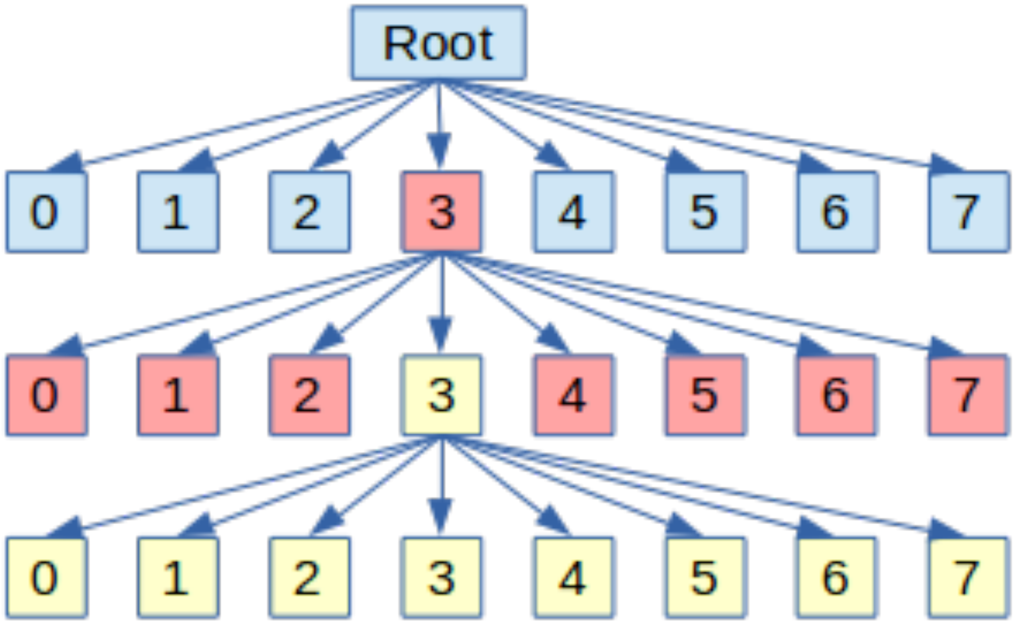
- Checking for ray-voxel intersections (AABB test) is very quick!
- But there may be many (over 100K) voxels, can we find the voxels we actually intersect quickly?



Voxel Scene Representation

- Use an octree to speed the search up
- If intersection occurs with voxel, it must occur with its bounding volume



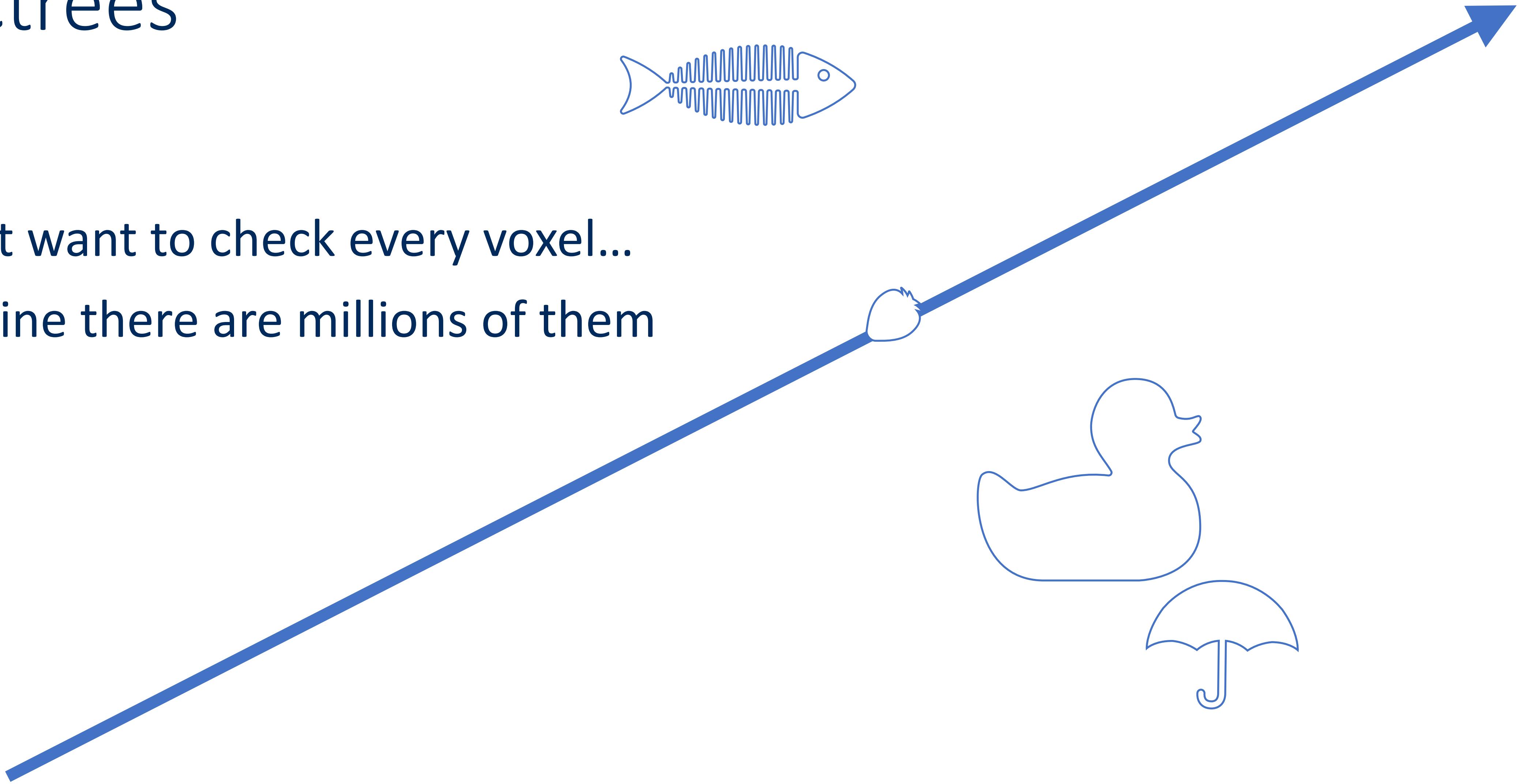


Source: https://geidav.wordpress.com/2014/07/18/advanced-octrees-1-preliminaries-insertion-strategies-and-max-tree-depth/

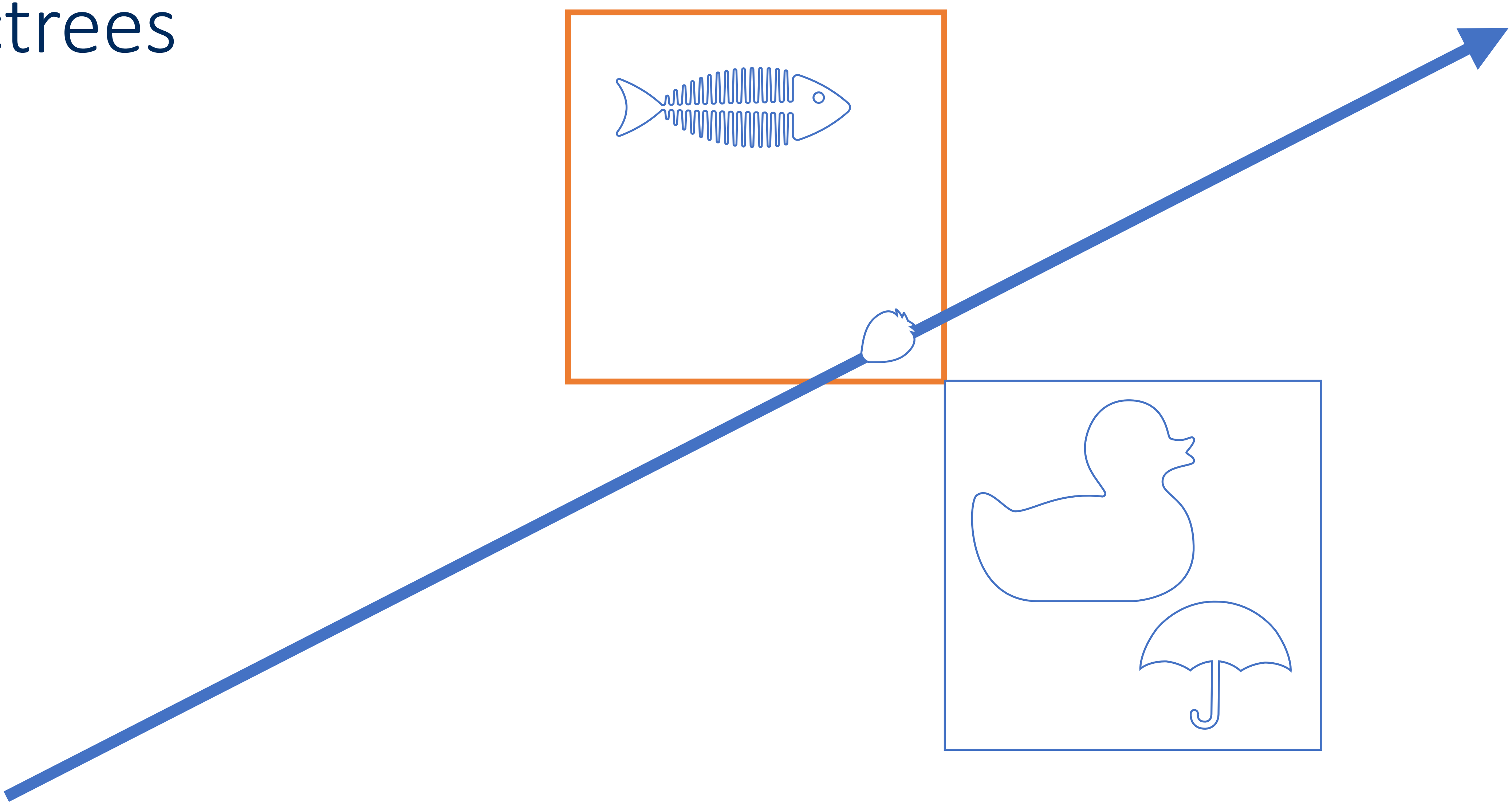
Octrees

Don’t want to check every voxel…

imagine there are millions of them

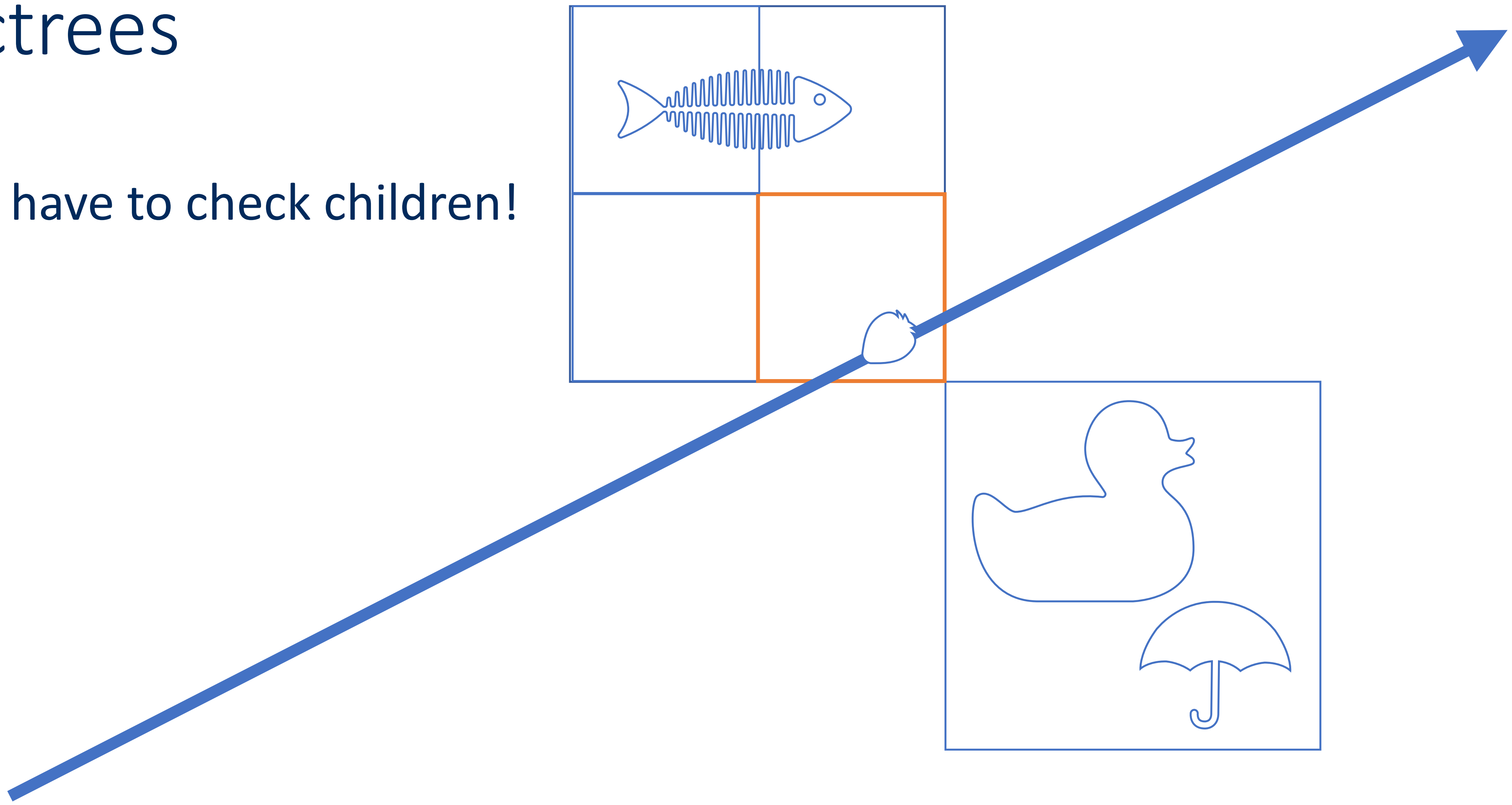


Octrees



Octrees

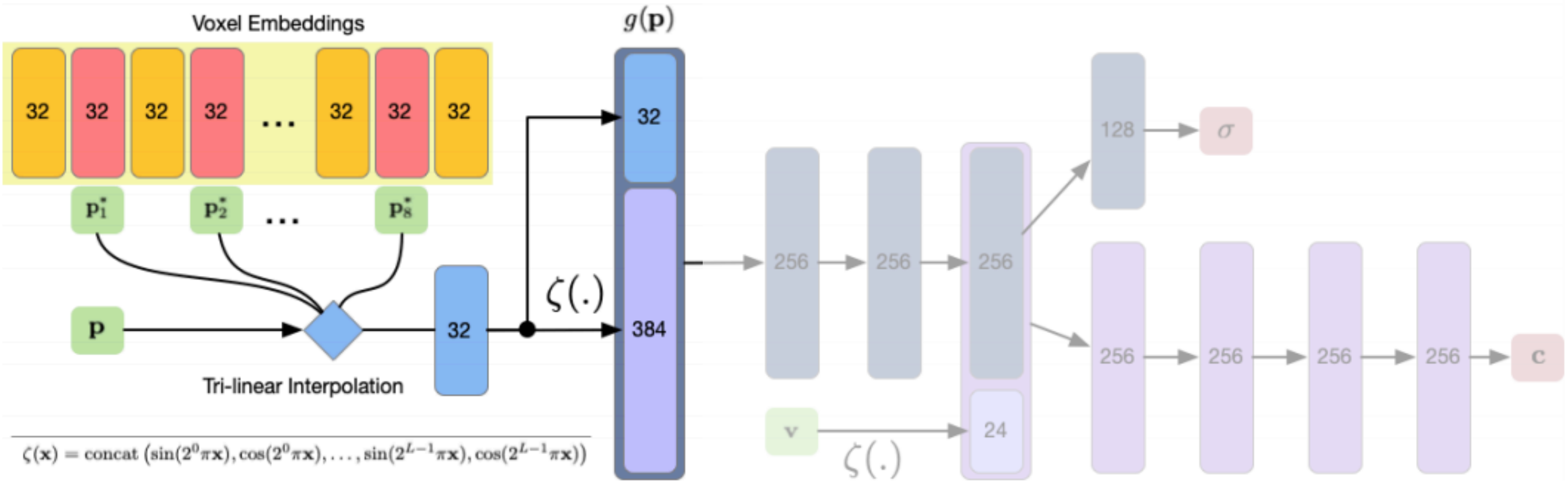
Only have to check children!

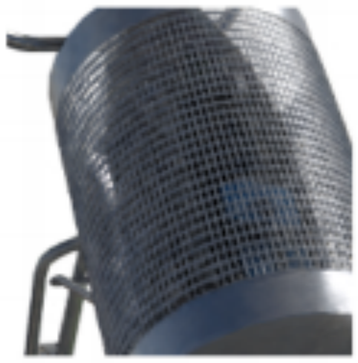


Getting the Details Right

- Use of learnable 32 dimensional embedding at each vertex (8 corners of voxel) improves detail modelling

- Supposed to model local “geometry, materials, colour”



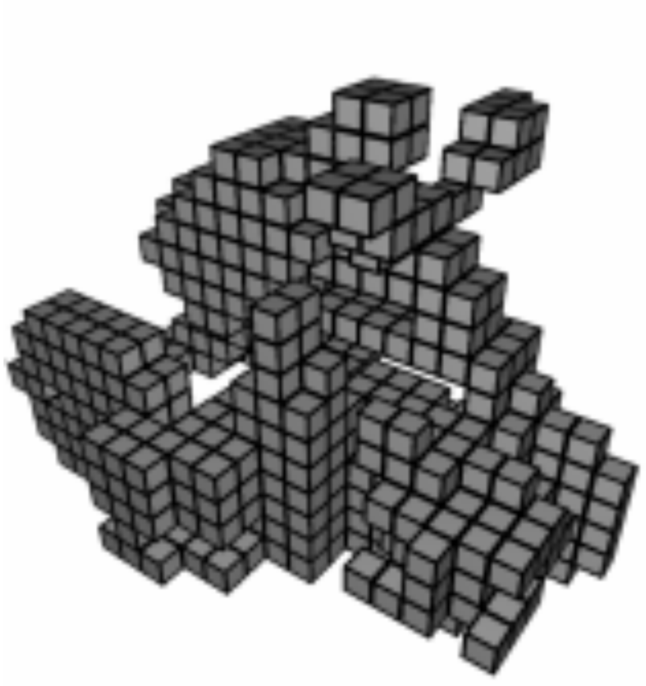


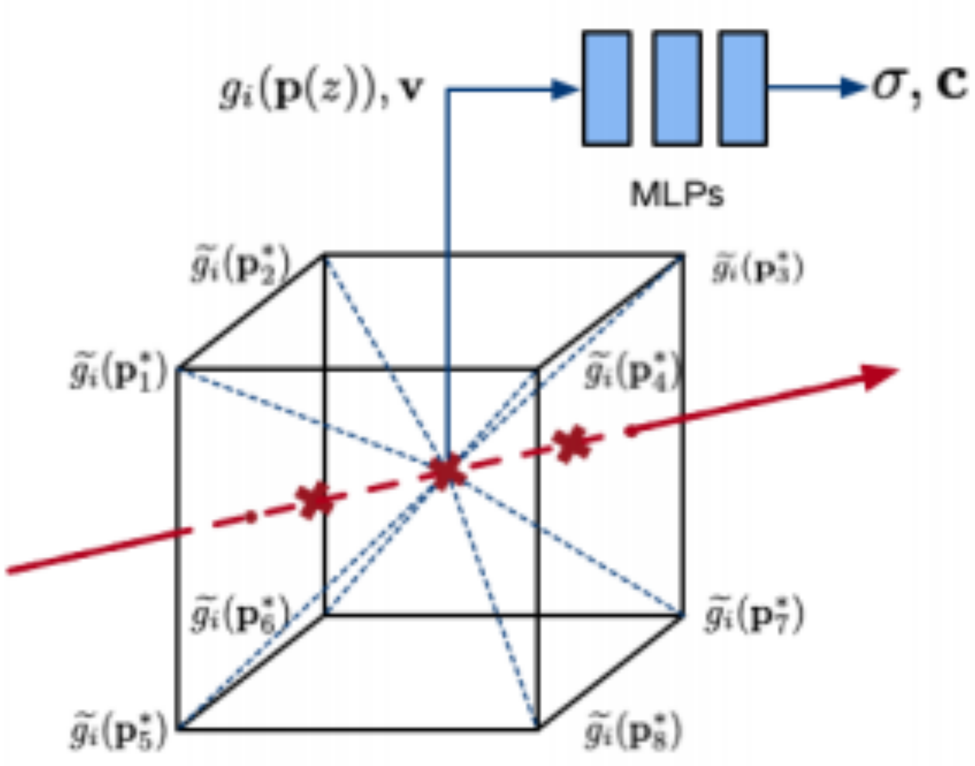
NSVF

NSVF w/o EMB

NSVF Inference

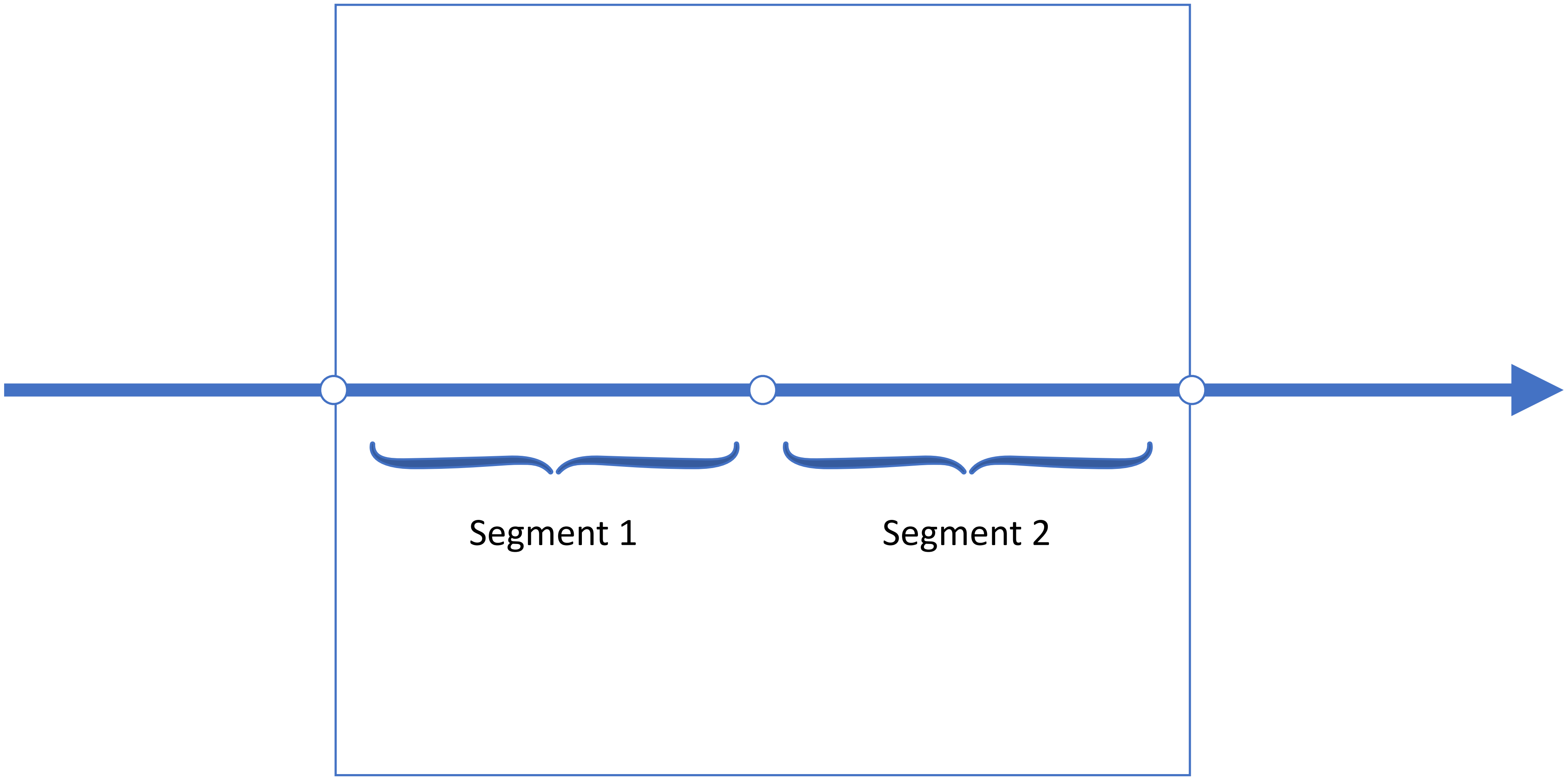
- For each ray, find all voxels that intersected it
- Sample at a fixed step size within each voxel (1/8 of its length)
- Accumulate colour along ray



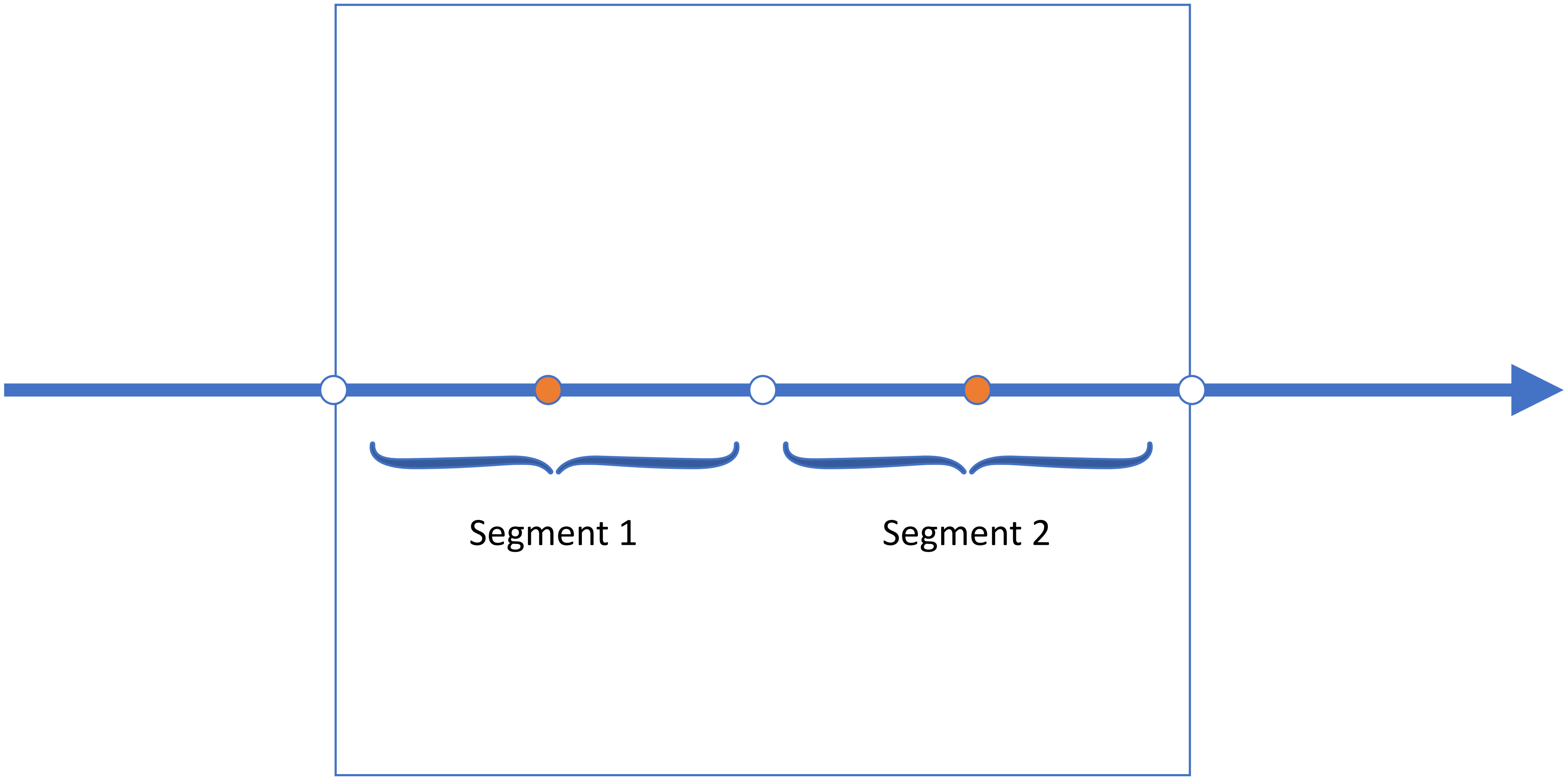


NSVF Inference Example

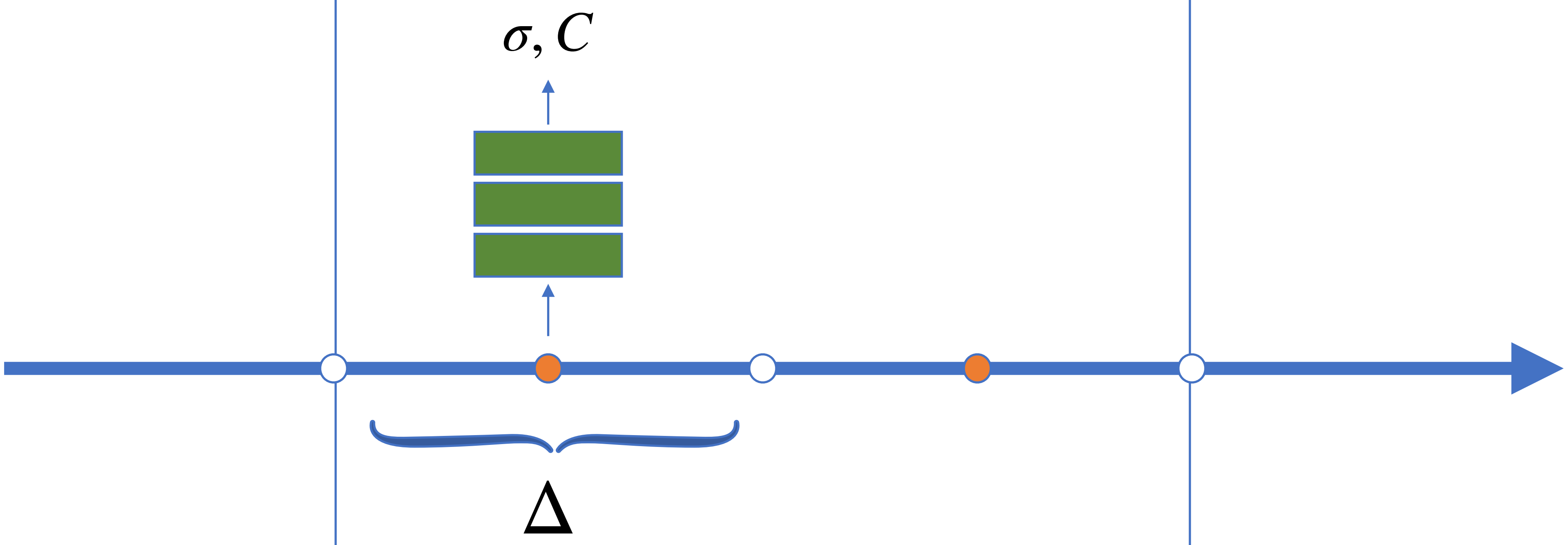
Step Size = 1/2 voxel length



“Midpoint rule” to do piecewise constant approx.



Trilinear interpolate embedding from corners, and predict density and colour



$$\alpha = 1 - e^{-\sigma \cdot \Delta}$$

Transmittance T is ratio of light that has managed to reach this segment

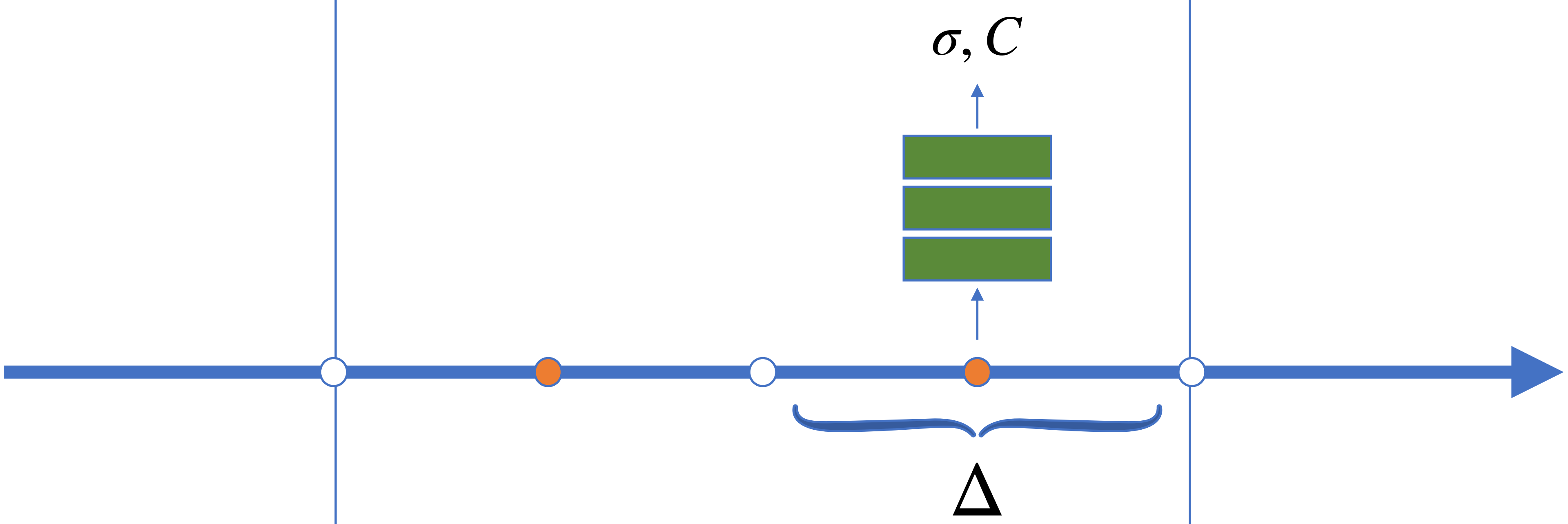
1) Compute segment weight: $R_i = \alpha_i \cdot T$

2) Accumulate colour: $C \leftarrow C + C_i \cdot R_i$

3) Decrease transmittance: $T \leftarrow T - R_i$

If T is below some threshold, terminate early

If we still have some transmittance left, repeat …



$$\alpha = 1 - e^{-\sigma \cdot \Delta}$$

If we run out of samples / voxels and *still* have some transmittance

$$C \leftarrow C + C_{background} \cdot T$$

Assumes a constant background colour!

We've roughly explained what all this is doing (with some difference in notation for clarity)

Volume Rendering. Volume-based methods (Lombardi et al., 2019; Mildenhall et al., 2020) estimate the integral $\boldsymbol{C}(\boldsymbol{p}_0, \boldsymbol{v})$ in Eq. 1 by densely sampling points on each camera ray and accumulating the colors and densities of the sampled points into a 2D image. For example, the state-of-the-art method NeRF (Mildenhall et al., 2020) estimates $\boldsymbol{C}(\boldsymbol{p}_0, \boldsymbol{v})$ as:

$$\boldsymbol{C}(\boldsymbol{p}_0, \boldsymbol{v}) \approx \sum_{i=1}^{N} \left(\prod_{j=1}^{i-1} \alpha(z_j, \Delta_j) \right) \cdot (1 - \alpha(z_i, \Delta_i)) \cdot \boldsymbol{c}(\boldsymbol{p}(z_i), \boldsymbol{v}) \tag{2}$$

where $\alpha(z_i, \Delta_i) = \exp(-\sigma(\boldsymbol{p}(z_i) \cdot \Delta_i))$, and $\Delta_i = z_{i+1} - z_i$. $\{\boldsymbol{c}(\boldsymbol{p}(z_i), \boldsymbol{v})\}_{i=1}^N$ and $\{\sigma(\boldsymbol{p}(z_i))\}_{i=1}^N$ are the colors and the volume densities of the sampled points.

Algorithm 1: Neural Rendering with NSVF

Input: camera $\boldsymbol{p}_0$, ray direction $\boldsymbol{v}$, step size τ, threshold ϵ, voxels $\mathcal{V} = \{V_1, \ldots, V_K\}$, background $\boldsymbol{c}_{\text{bg}}$, background maximum depth $z_{\max}$, parameters of the MLPs θ

Initialize: transparency $A = 1$, color $\boldsymbol{C} = \boldsymbol{0}$, expected depth $Z = 0$

Ray-voxel Intersection: Return all the intersections of the ray with k intersected voxels, sorted from near to far: $z_{t_1}^{\text{in}}, z_{t_1}^{\text{out}}, \ldots, z_{t_k}^{\text{in}}, z_{t_k}^{\text{out}}$, where $\{t_1, \ldots, t_k\} \subset \{1 \ldots K\}, k < K$;

if $k > 0$ **then**

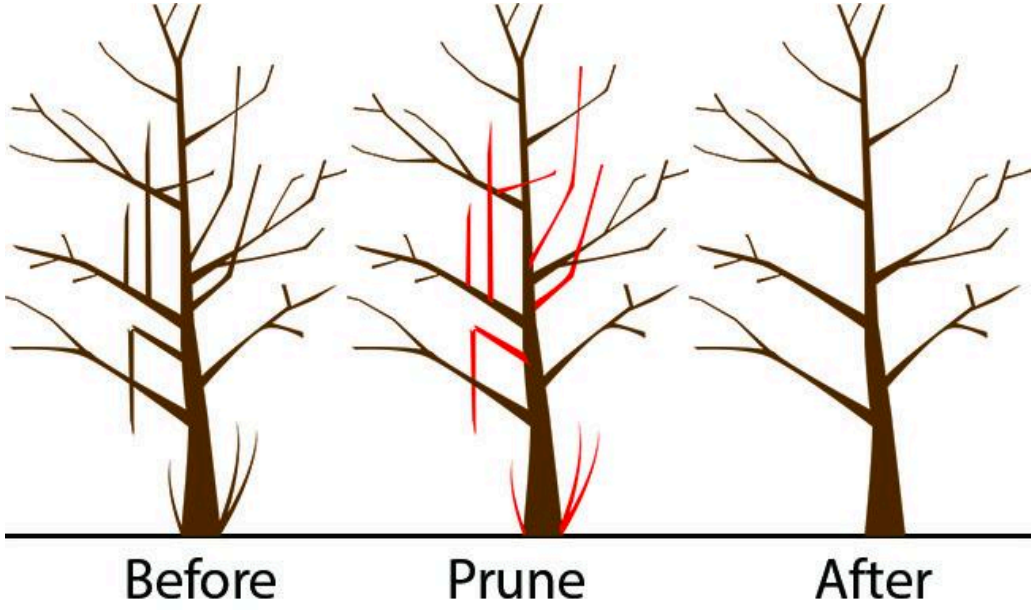
- **Stratified sampling:** $z_1, \ldots, z_m$ with step size τ, where $z_1 \geq z_{t_1}^{\text{in}}$ and $z_m \leq z_{t_k}^{\text{out}}$;
- **Include voxel boundaries:** $\tilde{z}_1, \ldots \tilde{z}_{2k+m} \leftarrow \text{sort}\left(z_1, \ldots, z_m; z_{t_1}^{\text{in}}, z_{t_1}^{\text{out}}, \ldots, z_{t_k}^{\text{in}}, z_{t_k}^{\text{out}}\right)$;
- **for** $j \leftarrow 1$ **to** $2k + m - 1$ **do**
  - **Obtain midpoints and intervals:** $\hat{z}_j \leftarrow \frac{\tilde{z}_j + \tilde{z}_{j+1}}{2}, \Delta_j \leftarrow \tilde{z}_{j+1} - \tilde{z}_j$;
  - **if** $A > \epsilon$ **and** $\Delta_j > 0$ **and** $\boldsymbol{p}(\hat{z}_j) \in V_i(\exists i \in \{t_1, \ldots, t_k\})$ **then**
    - $\alpha \leftarrow \exp\left(-\sigma_\theta\left(g_i(\boldsymbol{p}(\hat{z}_j))\right) \cdot \Delta_j\right), \quad \boldsymbol{c} \leftarrow \boldsymbol{c}_\theta\left(g_i(\boldsymbol{p}(\hat{z}_j)), \boldsymbol{v}\right)$;
    - $\boldsymbol{C} \leftarrow \boldsymbol{C} + A \cdot (1 - \alpha) \cdot \boldsymbol{c}, \quad Z \leftarrow Z + A \cdot (1 - \alpha) \cdot \hat{z}_j, \quad A \leftarrow A \cdot \alpha$;

$\boldsymbol{C} \leftarrow \boldsymbol{C} + A \cdot \boldsymbol{c}_{\text{bg}}, \quad Z \leftarrow Z + A \cdot z_{\max}$;

Return: $\boldsymbol{C}, Z, A$

How do we get the voxels?

- At the beginning of training, we don't know where anything is
- So we can start with a dense voxel field (everything occupied)
- Occasionally, we remove voxels if all its corners fall below a threshold alpha



https://hedgekingottawa.ca/signs-you-need-to-prune-your-trees/

How do we get the voxels?

- **Problem:** for high res, there may be too many voxels to begin with!
- Can we avoid unnecessary computation, like we did with octrees?
- **Fix:** start by pruning “big” voxels, then *progressively subdivide*, and repeat

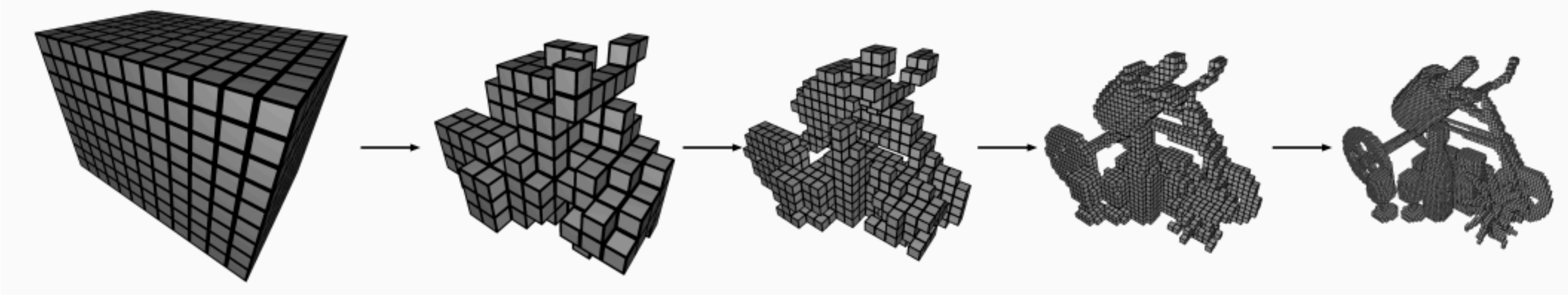


Figure 2: Illustration of self-pruning and progressive training

Loss function

$$\mathcal{L} = \sum_{(\boldsymbol{p}_0, \boldsymbol{v}) \in R} \|\boldsymbol{C}(\boldsymbol{p}_0, \boldsymbol{v}) - \boldsymbol{C}^*(\boldsymbol{p}_0, \boldsymbol{v})\|_2^2 + \underbrace{\lambda \cdot \Omega\left(A(\boldsymbol{p}_0, \boldsymbol{v})\right)}_{\text{Encourages transmittance to be 0 or 1}}$$

Other interesting details

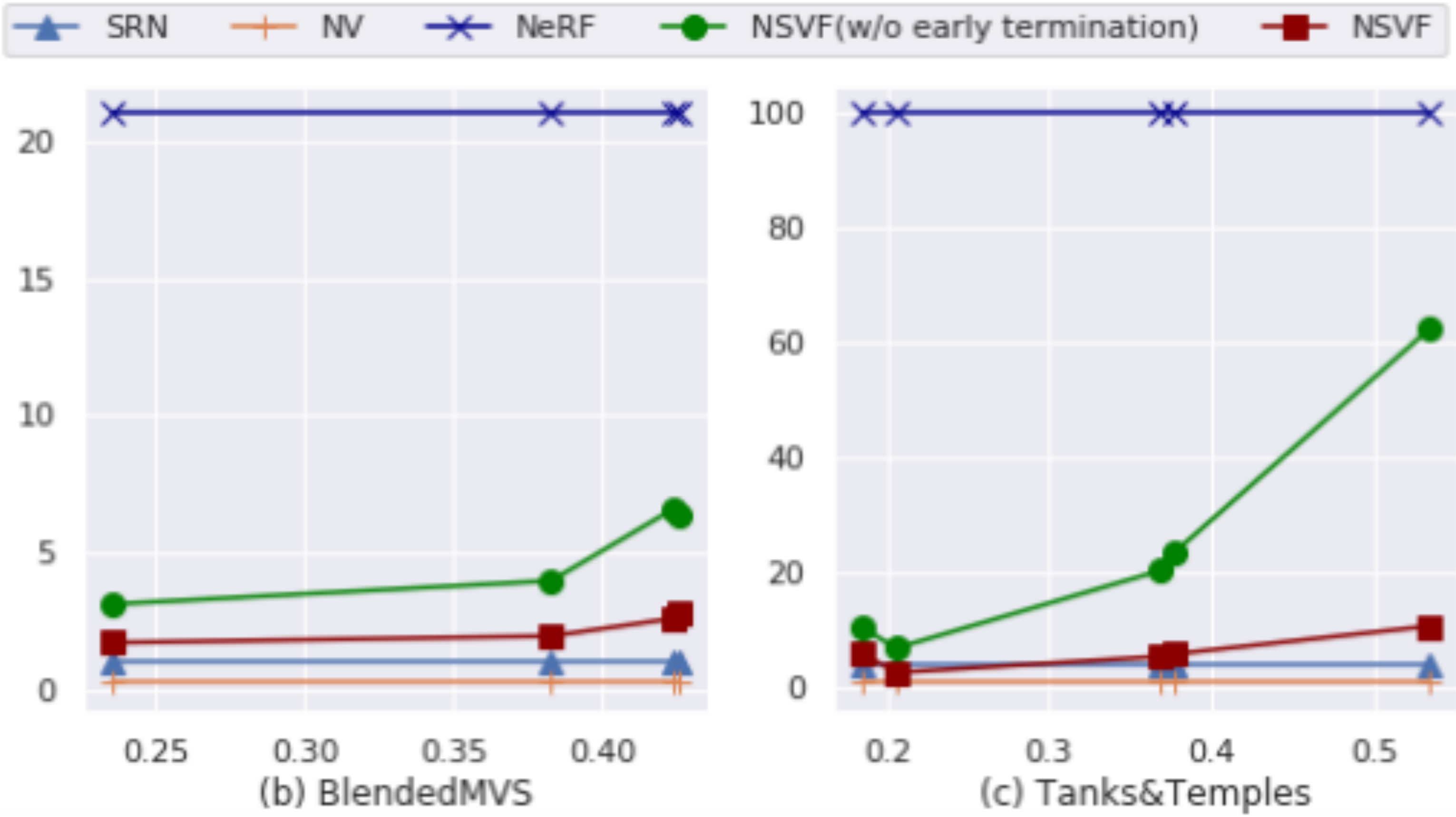
- Storage of MLP and voxels is **3.2 to 16 MB** depending on scene, compared to **5 MB** for NeRF

- Uses 3-4 rounds of voxel subdivision

- Unlike NeRF, computation is not constant per ray, # samples depends on # voxels intersected

Results and Discussion

“NSVF is typically over 10 times faster than the state-of-the-art (namely, NeRF) … ”



Results and Discussion

"… while achieving better quality."

	Synthetic-NeRF			Synthetic-NSVF			BlendedMVS			Tanks and Temples		
Models	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS	PSNR	SSIM	LPIPS
SRN	22.26	0.846	0.170	24.33	0.882	0.141	20.51	0.770	0.294	24.10	0.847	0.251
NV	26.05	0.893	0.160	25.83	0.892	0.124	23.03	0.793	0.243	23.70	0.834	0.260
NeRF	31.01	0.947	0.081	30.81	0.952	0.043	24.15	0.828	0.192	25.78	0.864	0.198
$NSVF^0$	**31.75**	**0.954**	0.048	**35.18**	**0.979**	**0.015**	26.89	**0.898**	0.114	**28.48**	**0.901**	0.155
NSVF	31.74	0.953	**0.047**	35.13	**0.979**	**0.015**	**26.90**	**0.898**	**0.113**	28.40	0.900	**0.153**

It looks great in videos too!

https://youtu.be/RFqPwH7QFEI?t=118

Results and Discussion

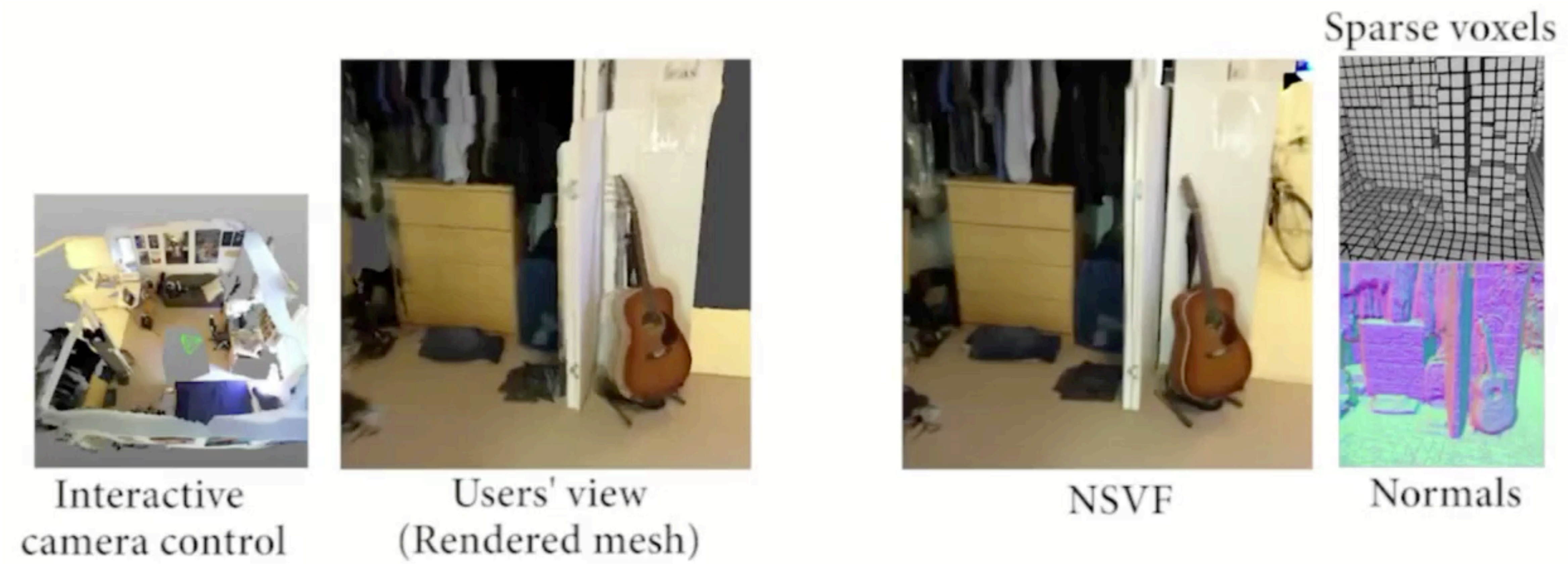
“NSVF can be easily applied to scene editing and composition.”

Train shared MLP on each individual scene,

then composite the voxel embeddings

Results and Discussion

"We also demonstrate a variety of challenging tasks, including multi-scene learning, free-viewpoint rendering of a moving human, and ***large-scale scene rendering.****"*



Critique / Limitations (from authors)

- Self-pruning threshold set to 0.5, may not be ideal for thin structures
- Complex backgrounds (no voxel intersection) can't be handled
- Requires known camera poses
- Still bad for complex geometry or lighting effects

Critique / Limitations (from me)

- It’s much faster than NeRF, but still **not real-time** (few seconds per frame)

- Though we expect a similar speedup to inference, **training time** is never mentioned

- Disentangling better sampling from detail modelling: what if we subdivided more, but avoided voxel embeddings? Would the model suffer from overfitting to embeddings at very fine levels?

Recap of Neural Sparse Voxel Fields

- Fast and high quality novel view synthesis, build upon NeRF.
- NeRF samples in empty space often, and bad sampling leads to slow and blurry renders.
- Key insight: use sparse voxel data structure to 1) avoid sampling empty space and 2) enable detailed modelling of local scene properties.
- Result: render 10 - 20x faster than NeRF, at higher quality