GPU Accelerated Volume Rendering, Scalar Field Explorer

Data format and challenges
Stochastic Sampling
View Dependent Ratio
Data Importance
Phase Space Tessellation Technique
K-Buffer Concept
View Adaptive Voxelization
Optimizations
Progress So Far
Conclusion

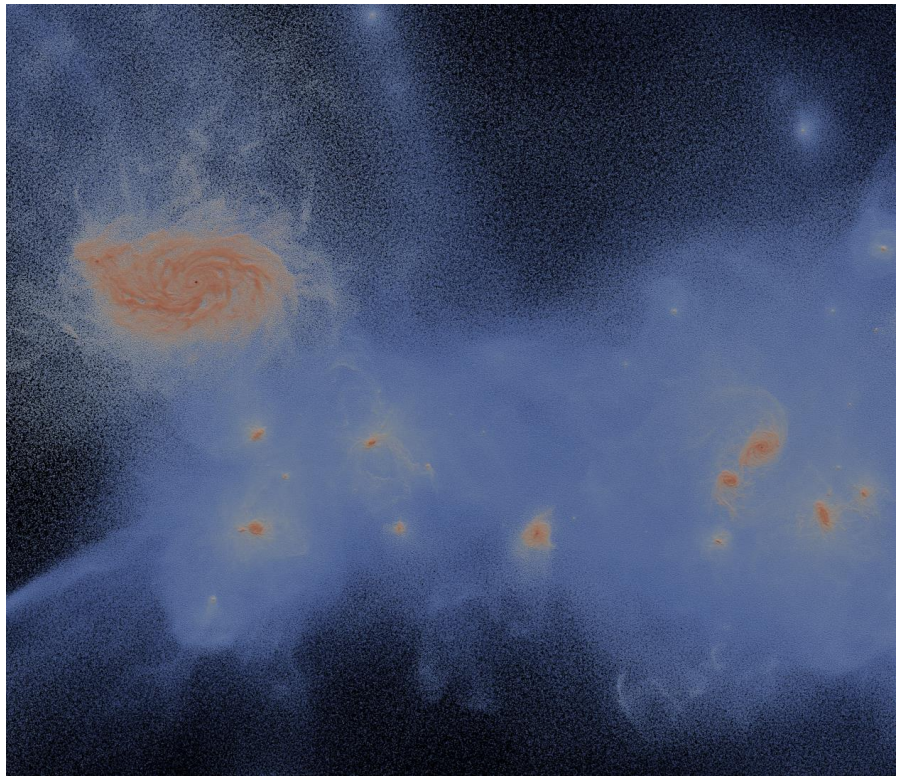
Presented by:
Utkarsh Sharma
Akash Maji

[1] Stochastic Volume Rendering of Multi-Phase SPH Data
[2] High-Quality Volume Rendering of Dark Matter Simulations

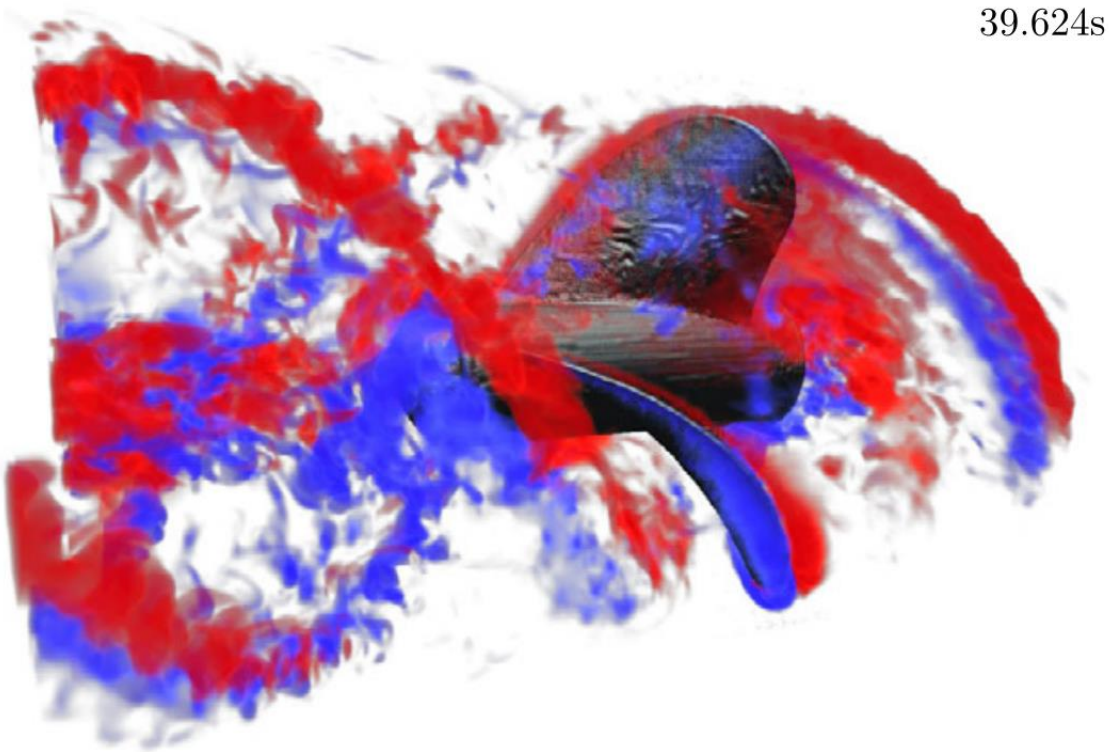
Motivation: Data Format and Challenges

Given an unstructured point cloud data of scalar field, how can we do Volume Rendering ?

Where does the unstructured data come from ?



Cosmological Simulations
IllustrisTNG (Voronoi cells)

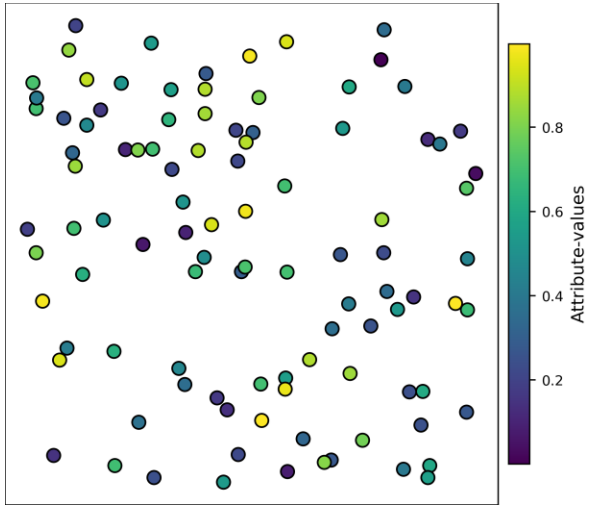


Fluid Simulations
Turbine Dataset (SPH particles)

Motivation: Data Format and Challenges

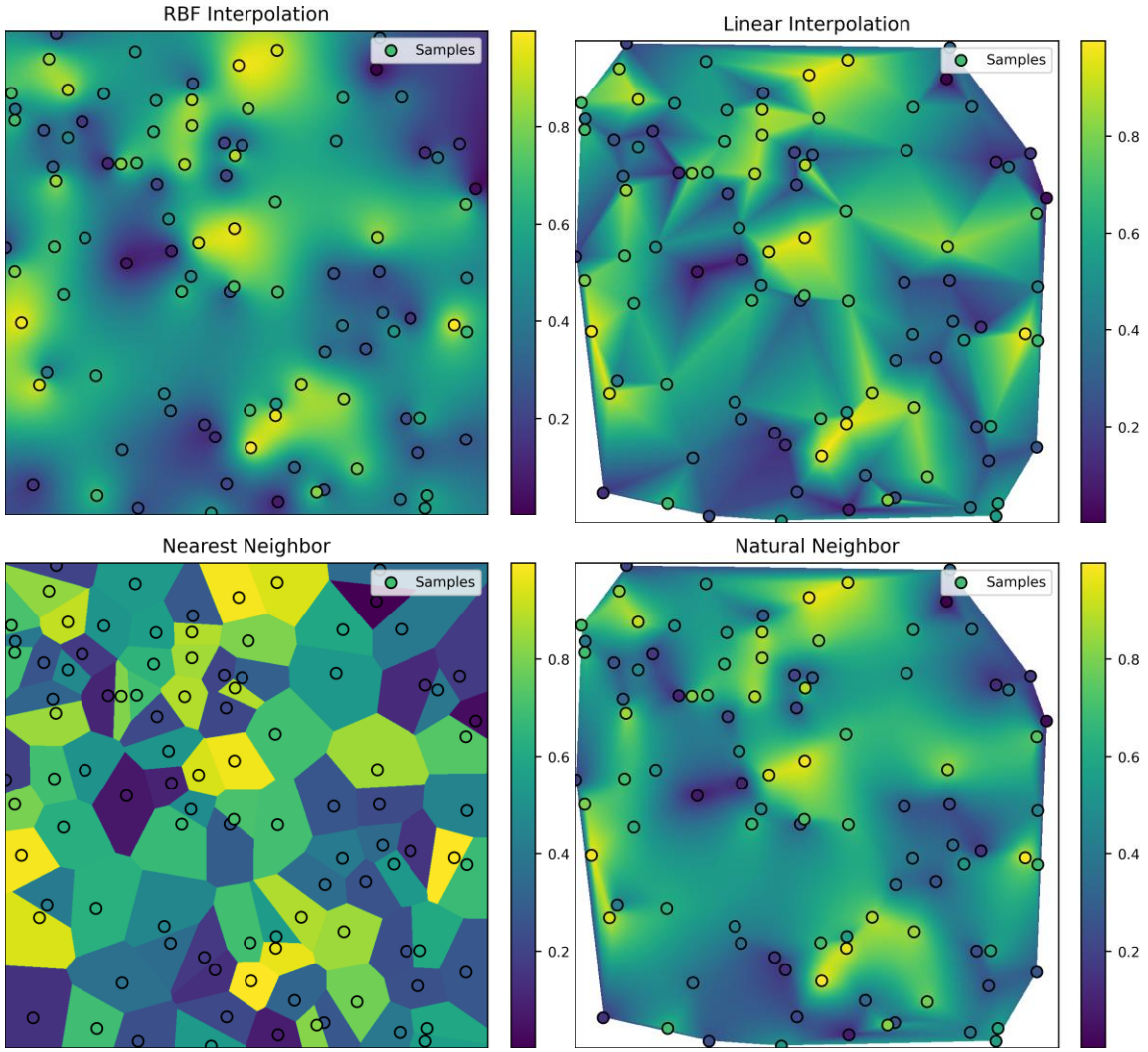
Given an unstructured data from simulations, how can we do Volume Rendering ?

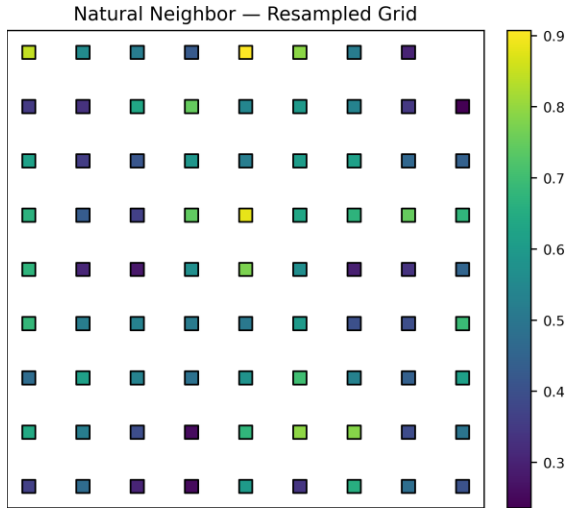
Can cause artifacts due to sampling noise!



Unstructured Data

$N \times D$ co-ordinates
$N \times K$ corresponding scalars (for data with K attributes)





Resampled Grid

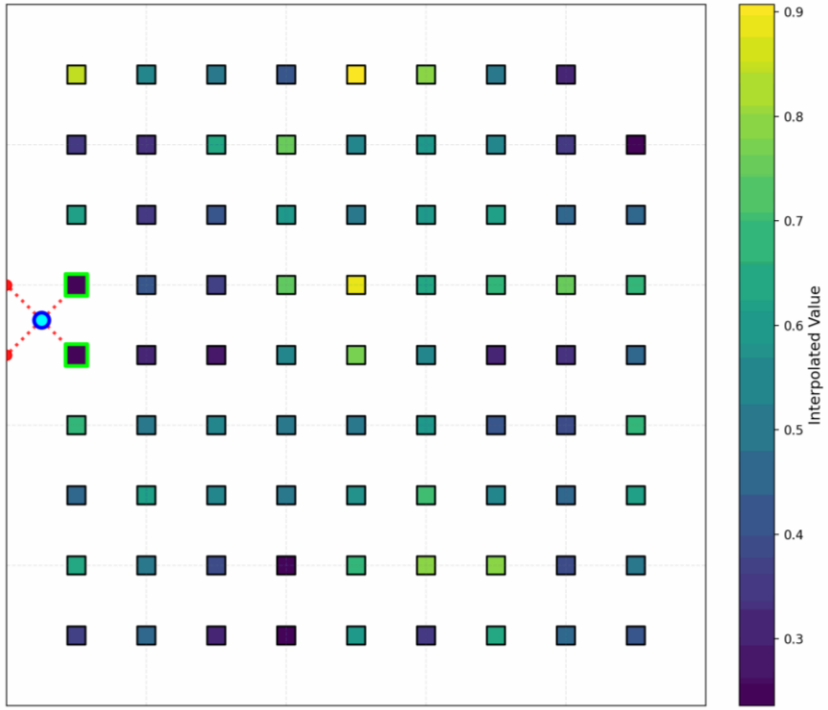
$H \times W \times D \times K$ scalar values and spacing is specified

Motivation: Data Format and Challenges

The Ray Marching using the Unstructured data can use empty space skipping

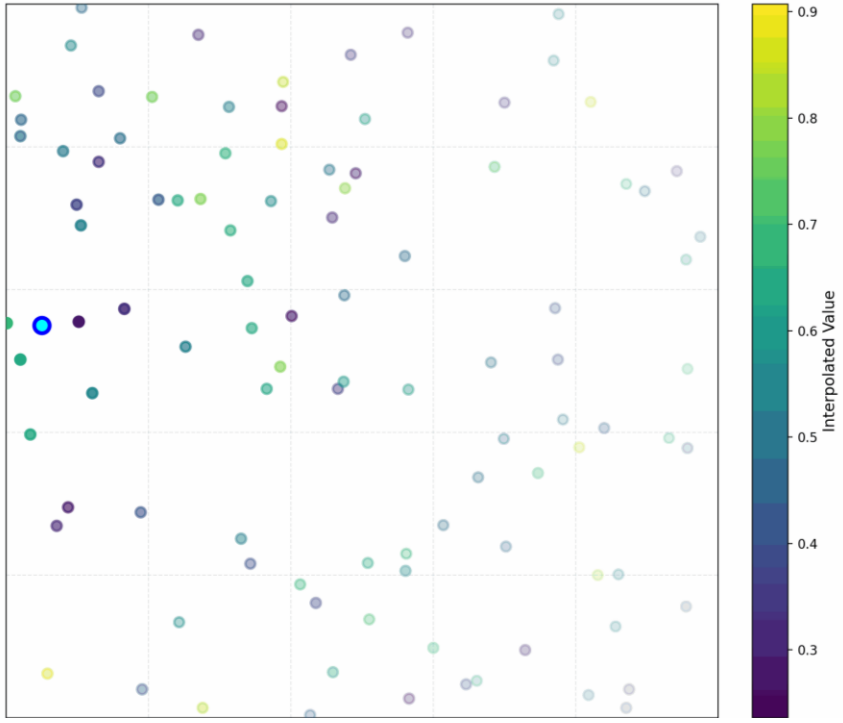
Resample to a grid:

- Well defined algorithm
- Trilinear interpolation is extremely fast
- Slow Pre-processing and high memory usage
- Faster methods have less accuracy



Directly use unstructured data

- Can skip Empty/Less Important Space
- More accurate interpolation
- Slow interpolation on the fly
- Might access many points for interpolation

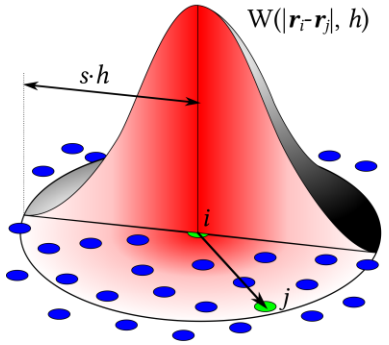


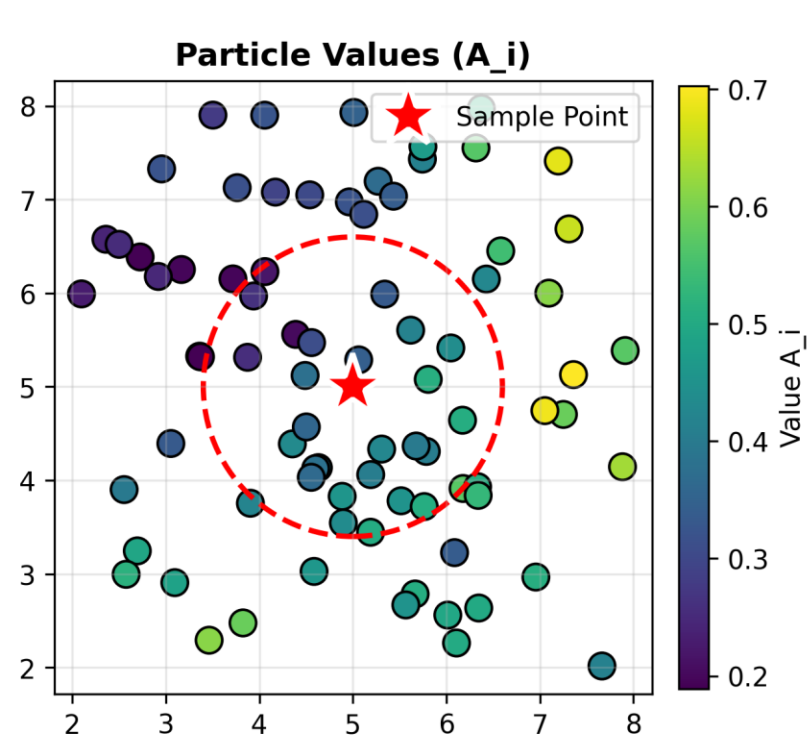
Assumption: SPH Data

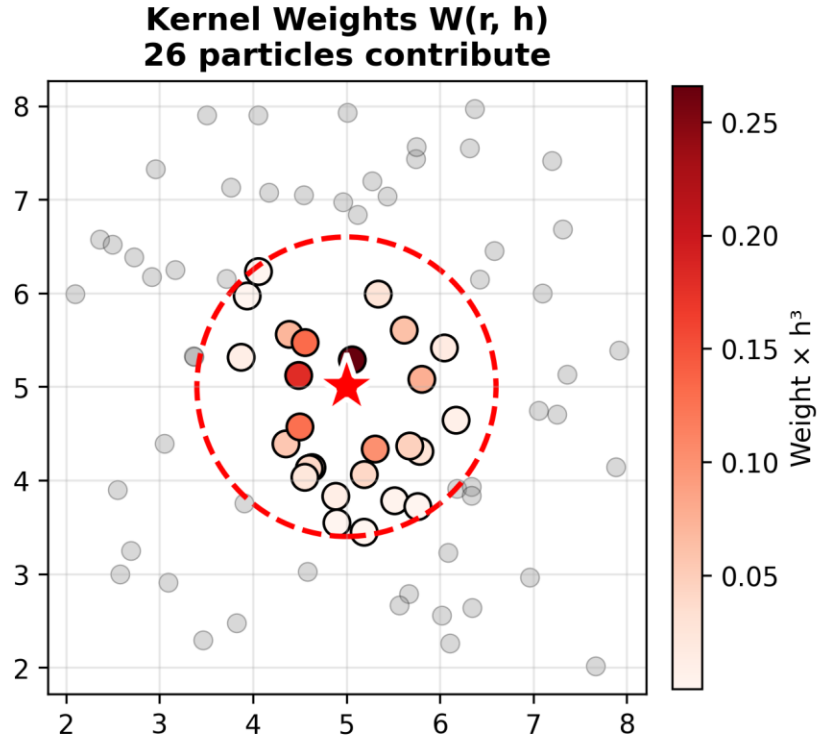
Smoothed-particle hydrodynamics (**SPH**) is a computational method used for simulating the mechanics of continuum media (solid mechanics/ fluid flows). SPH treats a fluid as particles with physical attributes that contributes to a continuous field via a **kernel function** (a smooth weighting function).

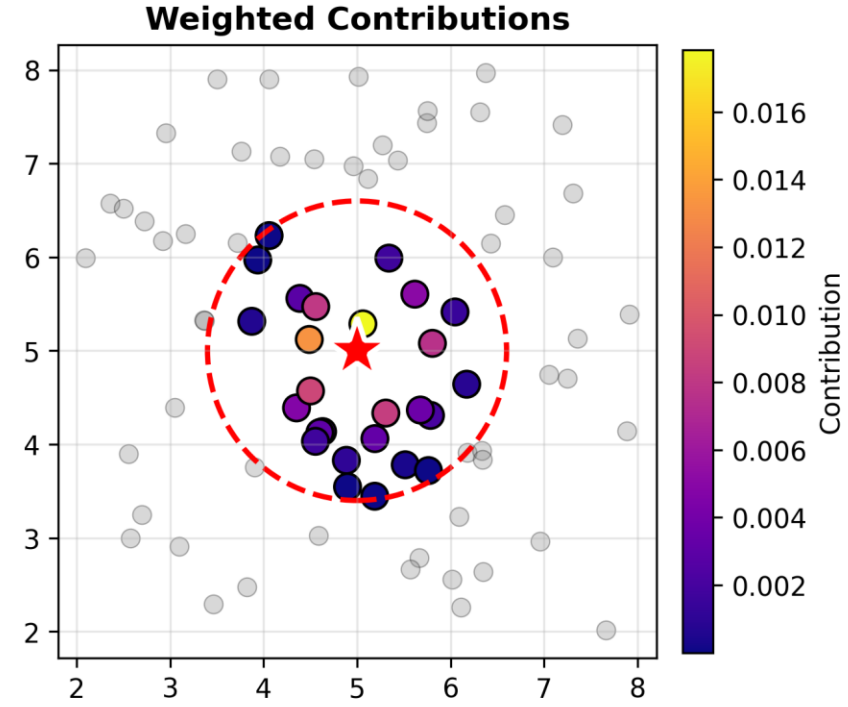
$$\rho(x) = \sum_i m_i \boldsymbol{W}(\|\boldsymbol{x_i} - x\|, h)$$

where m_i is particle mass, W is smoothing kernel and h is it's radius.

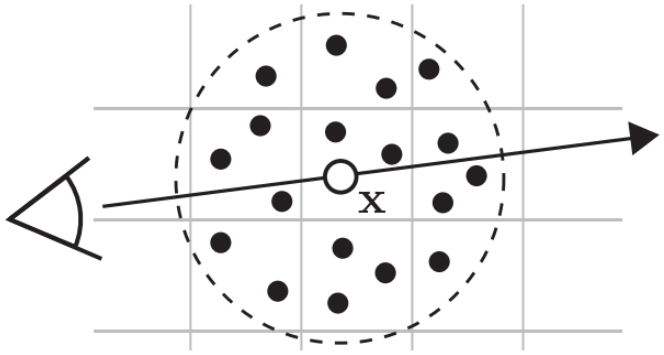




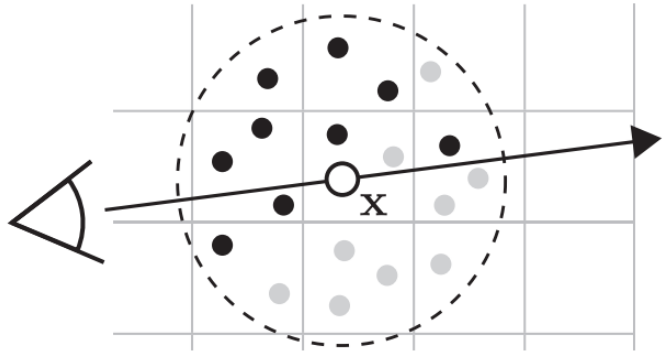




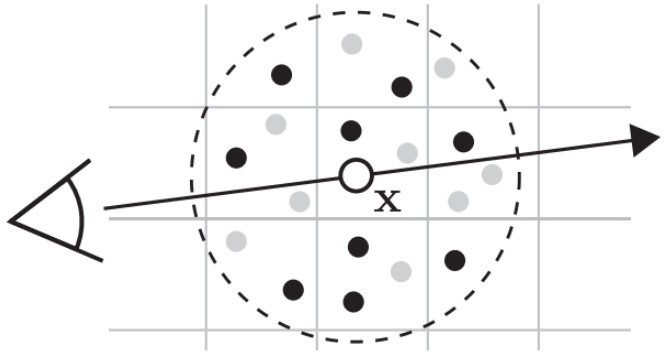
Key Method: Stochastic Sampling



Without subsampling



Stochastic Sampling



Stratified Sampling

Use defined **sampling ratio interval**: $[\boldsymbol{p_{min}}, \boldsymbol{p_{max}}]$ (Different for different use cases and datasets)

The following criteria affect the probability:

- **View dependent** ratio $\boldsymbol{p_t(x)}$ in real time during rendering
- A **data importance** $\boldsymbol{p_i(x)}$ that is precomputed for SPH dataset

If an attribute approximation is performed during ray marching, the two criteria $p_t, p_i \in [0, 1]$ are combined multiplicatively to obtain the particle sampling probability as:

$$\boldsymbol{p(x) = p_{min} + (p_{max} - p_{min}) p_t(x) p_i(x)}$$

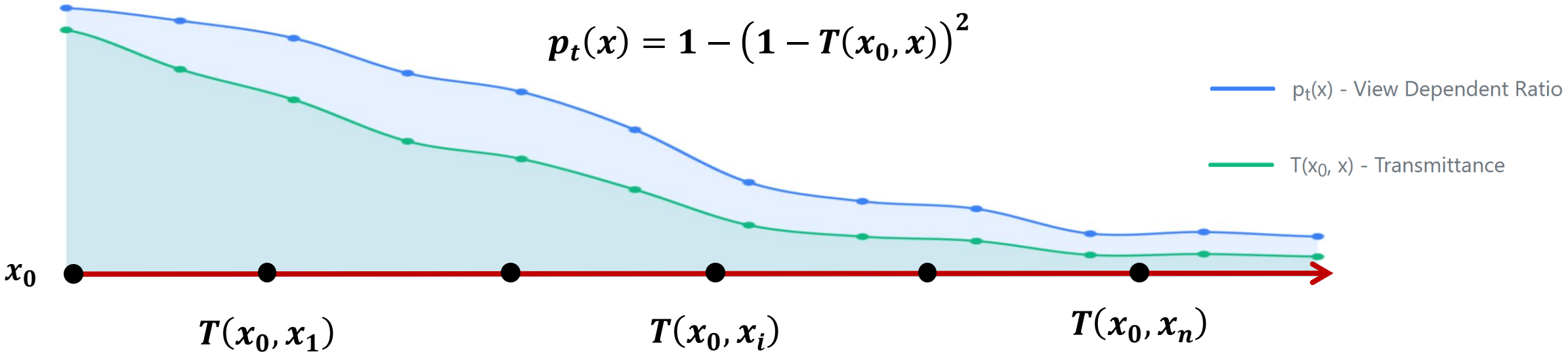
$n = p(x)|\mathcal{P}(x)|$ particles will be sampled from each grid cell for stratified

Key Method: Stochastic Sampling

View Dependent Ratio ($p_t(x)$):
Idea: Sampling points that are less visible have less contribution to the final pixel color therefore set $p_t(x)$ according to $T(x_0, x)$ for all SPH approximations at a sampling point.

To maintain a near-correct sampling at points that are almost completely visible and increase the subsampling at mostly occluded points we use a quadratic falloff as:

$$p_t(x) = 1 - \left(1 - T(x_0, x)\right)^2$$



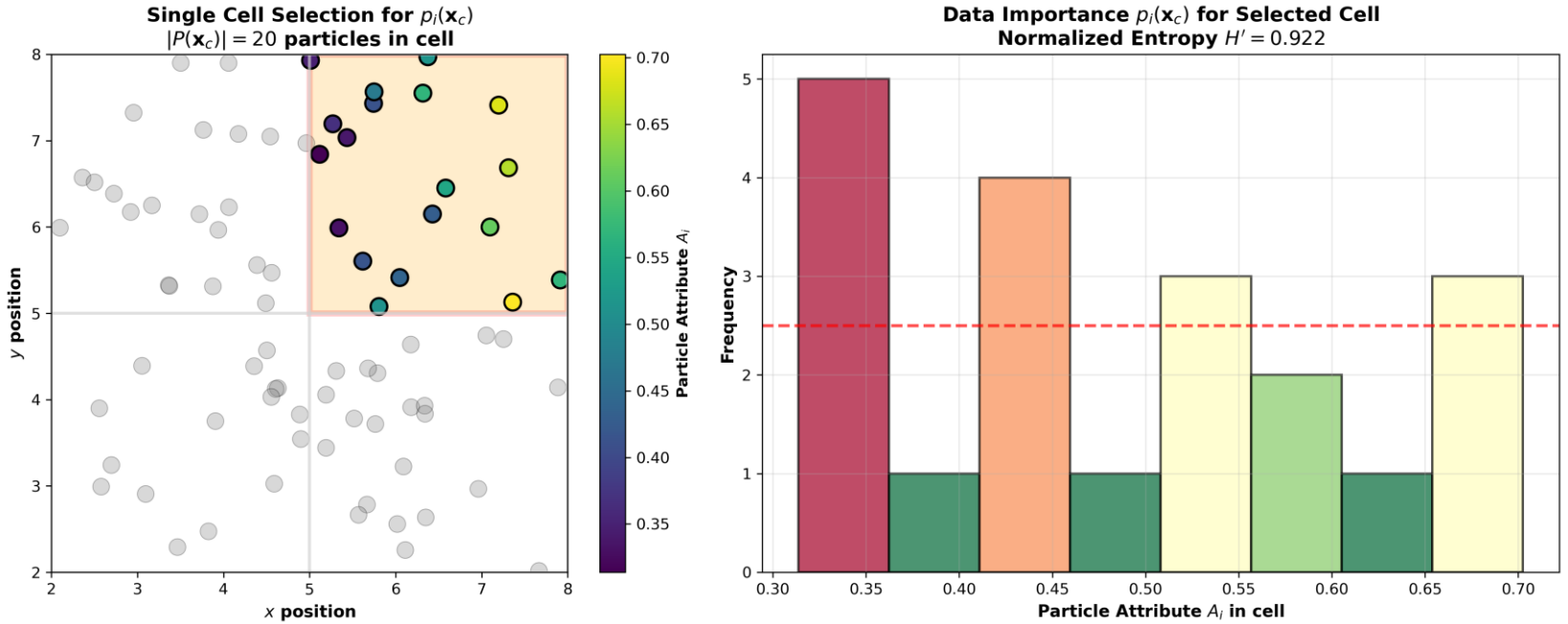
Transmittance between points x_0 and x_1 is given by $T(x_0, x_i)$
Occlusion can be obtained by $\left(1 - T(x_0, x_i)\right)$
Sample less from the occluded region and more from visible region

Key Method: Stochastic Sampling

Precomputed data importance ($p_i(x)$):
Idea: If neighboring particles have a higher attribute variation, the variance in the sampling process is increased and more particles should be considered in the SPH approximation

First, all attribute values of the particles in $\mathcal{P}(x)$ are binned in a normalized histogram $h(b)$ with N_{bins} bins. Then, the entropy is obtained as:

$$\mathbb{H}_{\mathcal{P}}(A) = -\sum_{b=0}^{N_{bins}-1} h(b)\, log_2\big(h(b)\big)$$

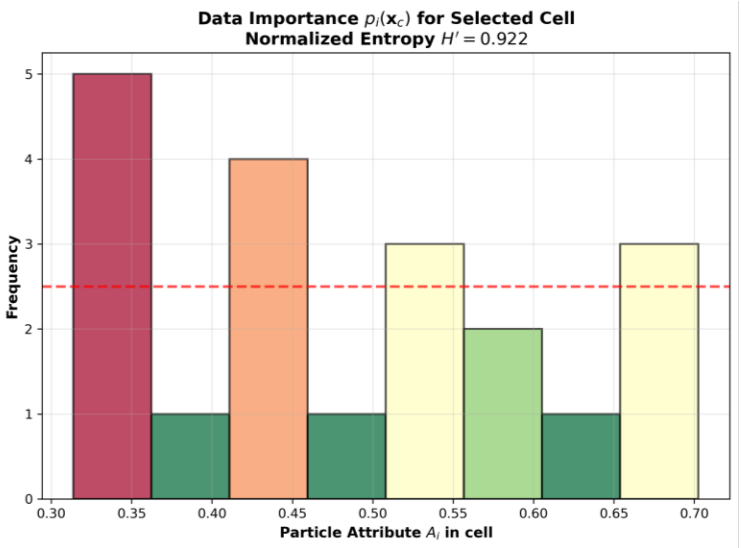


Key Method: Stochastic Sampling

Precomputed data importance ($p_i(x)$):
Idea: If neighboring particles have a higher attribute variation, the variance in the sampling process is increased and more particles should be considered in the SPH approximation

First, all attribute values of the particles in $\mathcal{P}(x)$ are binned in a normalized histogram $h(b)$ with N_{bins} bins. Then, the entropy is obtained as:

$$\mathbb{H}_{\mathcal{P}(x)}(A) = -\sum_{b=0}^{N_{bins}-1} h(b)\, log_2\big(h(b)\big)$$



Normalize the values to be independent from the histogram size:

$$\bar{\mathbb{H}}_{\mathcal{P}(x)}(A) = \frac{2^{\mathbb{H}_{\mathcal{P}}(A)}}{N_{bins}}$$

Define the importance value for position x, with set of particles in the local neighborhood $\mathcal{P}(x)$, as:

$$p_i(\mathbf{x}) = \begin{cases} 0, & \text{if} \quad |\mathcal{P}(\mathbf{x})| = 0 \\ 1, & \text{if} \quad \exists i, j \in \mathcal{P}(\mathbf{x}) : \pi_i \neq \pi_j \,, \\ \bar{\mathbb{H}}_{\mathcal{P}(\mathbf{x})}(A), & \text{otherwise} \end{cases}$$

Results

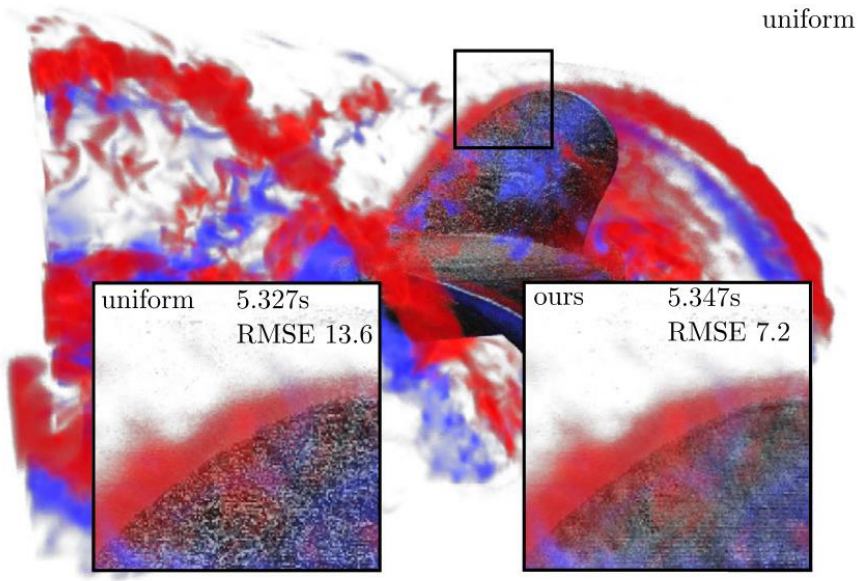


Figure 8: *Equal-time rendering using uniform particle sampling compared to our non-uniform sampling strategy in Figure 3(b). While both images took about 5.3 s to render, our strategy has significantly lower error.*

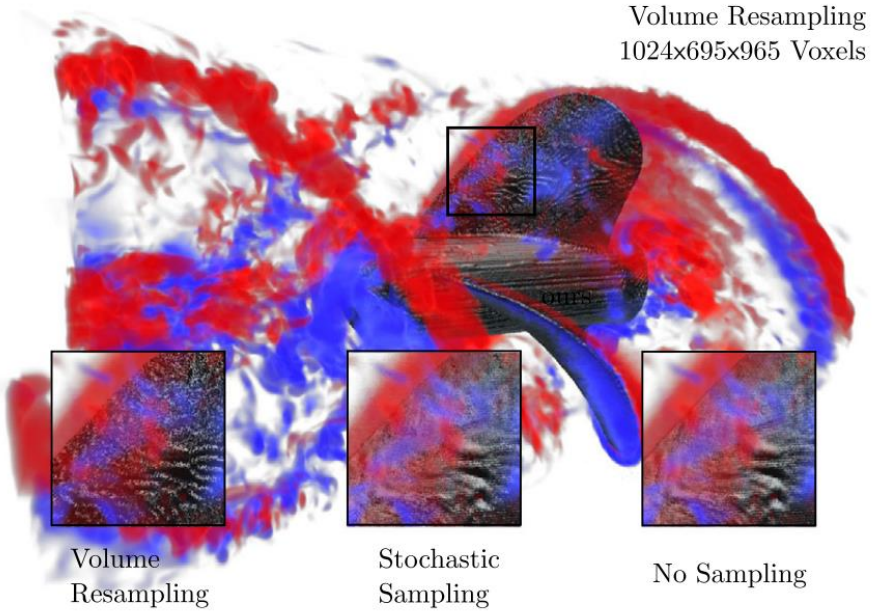


Figure 9: *Rendering using volume resampling (269.896 s preprocessing, 0.293 s rendering) with inlet comparisons to our high quality stochastic sampling configuration (0.029 s, 6.286 s) and a rendering without any sampling (0 s, 46.612 s).*

			Fast Sampling		Quality Sampling		No Sampling	Volume Resampling	
Dataset	Particles	Preproc.	Frame Time	RMSE	Frame Time	RMSE	Frame Time	Preproc.	Frame Time
Bubble	4,347,225	10 ms	0.993 s	8.277	5.050 s	4.360	10.274 s	143.234 s	124 ms
Spray Nozzle	43,188,662	28 ms	0.982 s	23.108	8.350 s	5.391	44.895 s	447.261 s	82 ms
Turbine	86,473,832	29 ms	0.962 s	23.828	5.347 s	7.187	39.624 s	269.896 s	293 ms

Root-mean-square errors are measured compared to the rendering without stochastic sampling.

High Quality Volume Rendering of Dark Matter Simulations

N-body dark matter simulations are essential for studying the large-scale structure of the Universe

We rely on accurate **mass densities** for identifying features of the **cosmic web**

Traditional Approaches

Problem: suffer from **artifacts due to sampling noise**

Alternate Approach

The **Phase Space Element Approach (PSEA)** constructs a tessellation of the dark matter sheet (3D) embedded in **6-dimensional phase space.**

This approach yields **density fields of very high quality** by summing the density contributions of all overlapping elements in **configuration space**

Paper Goal: To introduce a robust volume rendering approach that achieves **high image quality** at **interactive** frame rate for PSEA

Ref: High-Quality Volume Rendering of Dark Matter Simulations

So, we want PESA, but...

The PSEA generates depth-complex **tetrahedral meshes**

They suffer from an **excessive amount of self-intersections** due to dark matter sheet folding over itself

In **overdense regions**, it is common for up to **10^4 cells to overlap** a single spatial location

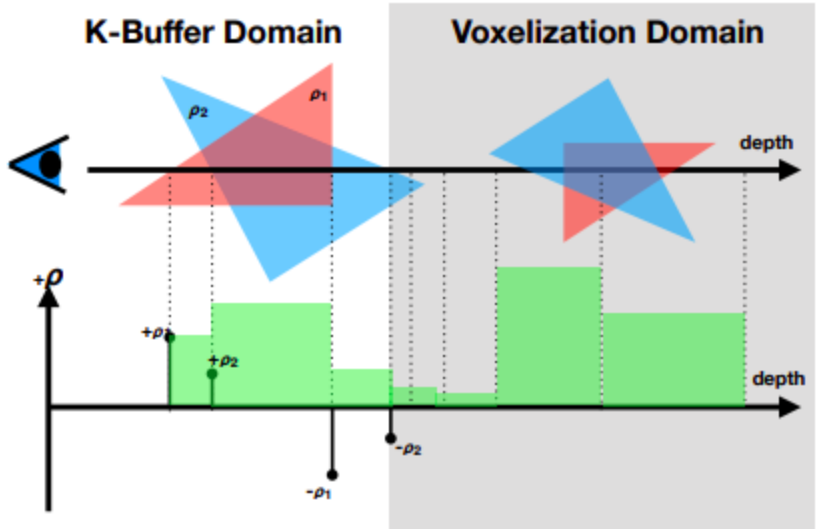
Previous Attempts:

They suffered from **image-noise;** used a **binning approach** for high depth complexity areas

The Novel Hybrid Volume Rendering Algorithm [4 Steps] for PESA

It combines a **single–pass k–buffer approach** with a **view–adaptive voxelization grid**

It achieves Order–Independent Transparency (OIT) and guarantee correct depth sorting for the *k* closest, highly overlapping fragments



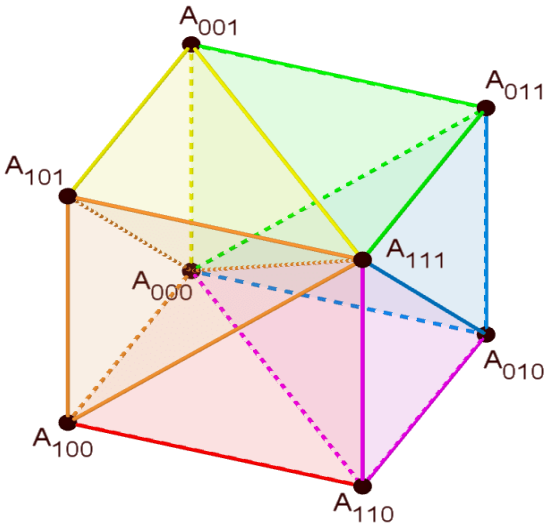
Needs:

Leverages atomic operations on 64-bit integers on the GPU

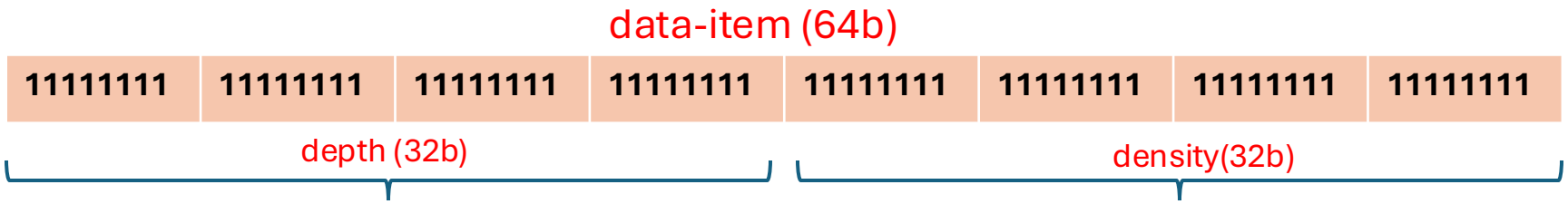
Parameter choices (k, voxel resolution) affect quality and memory

Part-1: Geometry Construction

- Particle positions (vertices) are uploaded to GPU(3D–texture) and accessed via Lagrangian grid indices.
- **GS** constructs the **six tetrahedra** per cell and calculates their mass densities.
- **FS** computes **entry point depth (d{in})** and the **intersection length(dt)** inside tetrahedron for the viewing ray.



Part-2: K-Buffer Construction

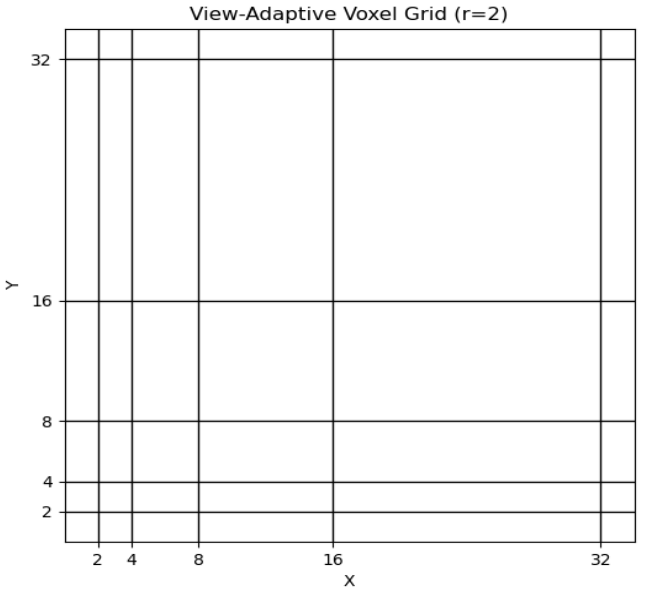


- Handle extreme depth complexity near camera by creating a perfectly depth-sorted list of the first *k* fragments.
- The data item is inserted into the pixel's fragment list by looping over the k-entries, atomically exchanging it
- **Result**: depth-sorted list of the viewing rays' entry and exit points for the k-closest tetrahedra faces

  (along with the tetrahedra's density ρ at entry points and −ρ at the exit points)

Part-3: View Adaptive Voxelization

- Capture the density contributions of tetrahedra beyond *k*-buffer coverage
- Viewport-aligned 3D–texture; **logarithmic depth spacing**.
- Density contributions (ρT) are atomically added and weighted by (dT)
- Prevents high-frequency image noise because it correctly accumulates the contributions of **thin** tetrahedra (not possible using binning)



Part-4: Colour-mapping and Blending

- Combine foreground (*k*-buffer) and background (voxel grid) density into the final image color.
- Densities are combined in a **front-to-back, view–consistent order**.
- Render by looping over the entries in the k–buffer, accumulate the density contributions, map the current density at each interval to colors and opacities.
- Analogously, the densities stored in the view–adaptive voxelization grid are sampled, mapped, blended.

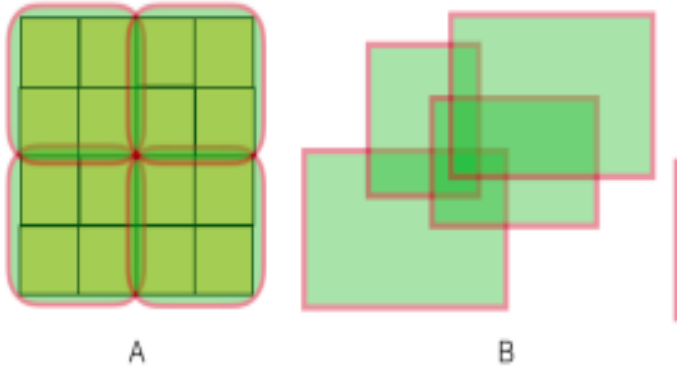
Optimization1: Tiled Rendering & View-Frustum Culling

Generating geometry in the **GS** is a major overhead, especially for outside the view

Solution:

Subdivide the data into cubical blocks (e.g., 8^3 cells per block)

Only blocks whose **AABB** intersect the current tile's view frustum are rendered



Optimization2: View-Dependent Multiresolution (LOD)

Rendering full resolution data is slow

Solution:

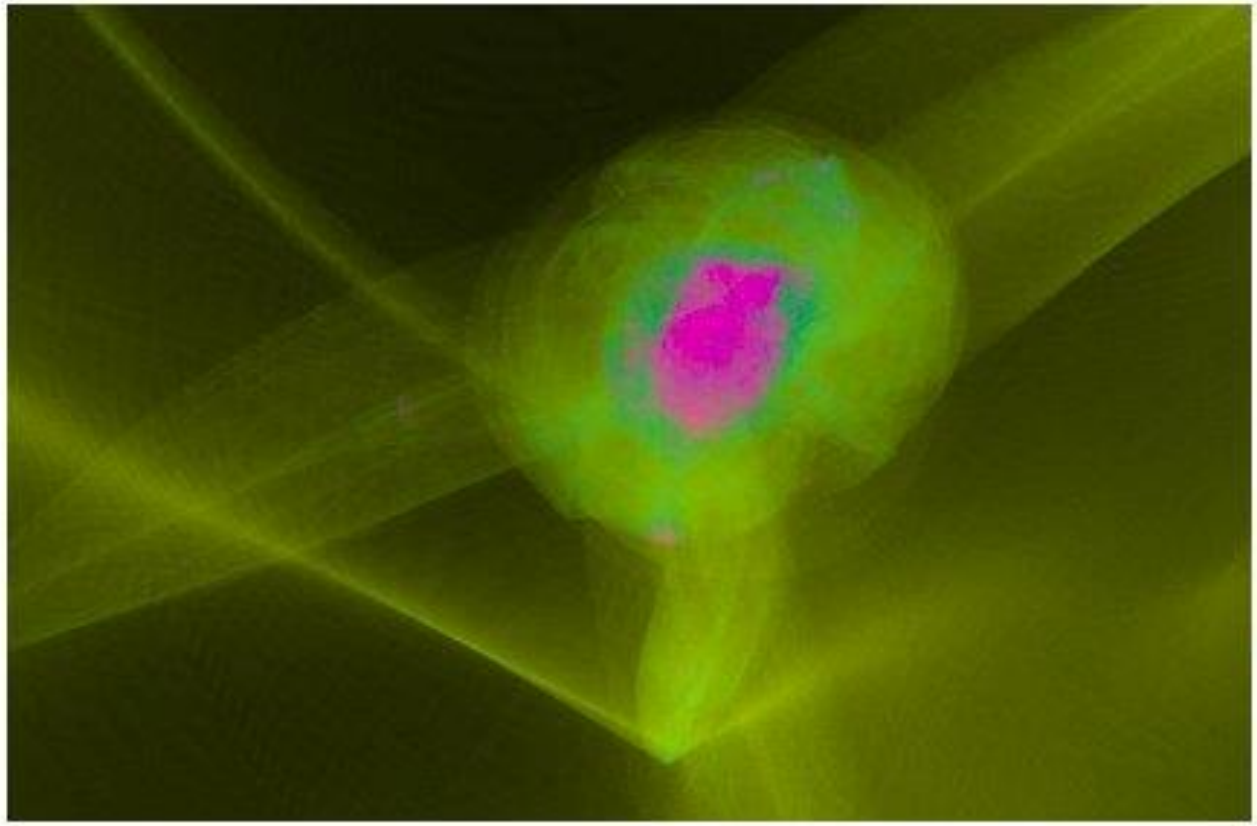
Resolution is determined by the **screen-space extension** of the intersecting blocks' **AABB**.

Data for lower resolutions are generated by **downsampling**.

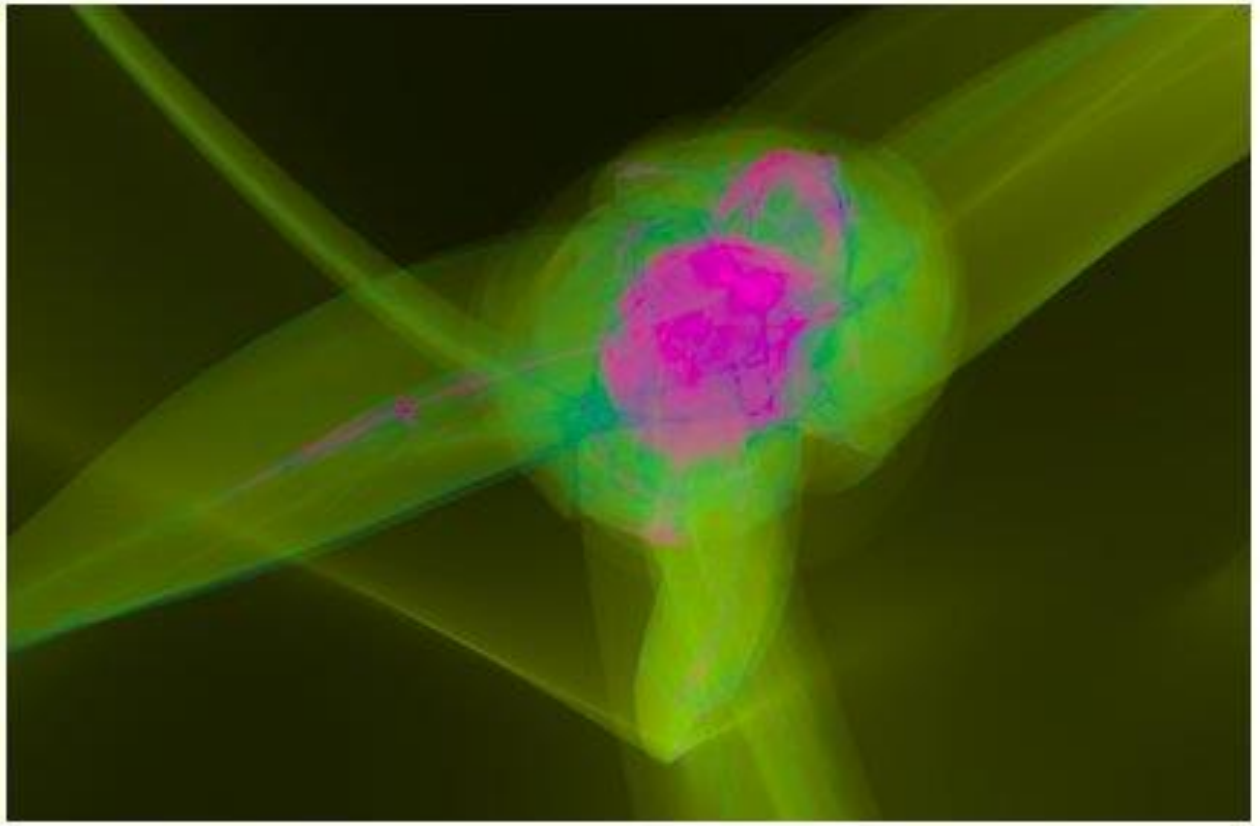
Results

- Achieves high quality; optimizations give interactive frame rates.
- Captures subtle features: caustic structures
- Single-pass k–buffer + View–adaptive voxelization: resolves depth complexity and image noise.

Results



Cell Projection Approach



Our Hybrid Approach

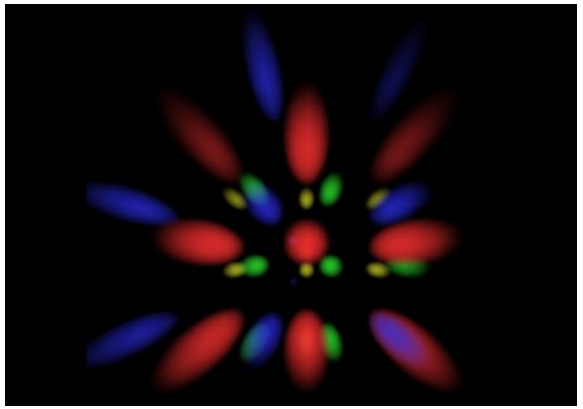
Progress

What we Have

- CUDA + Python based Volume renderer and Transfer Function Editor
- Support for IllustrisTNG (cosmological data) and regular grid
- Simple GUI to load and visualize the data
- A simple Neural Representation and renderer in python (incomplete)

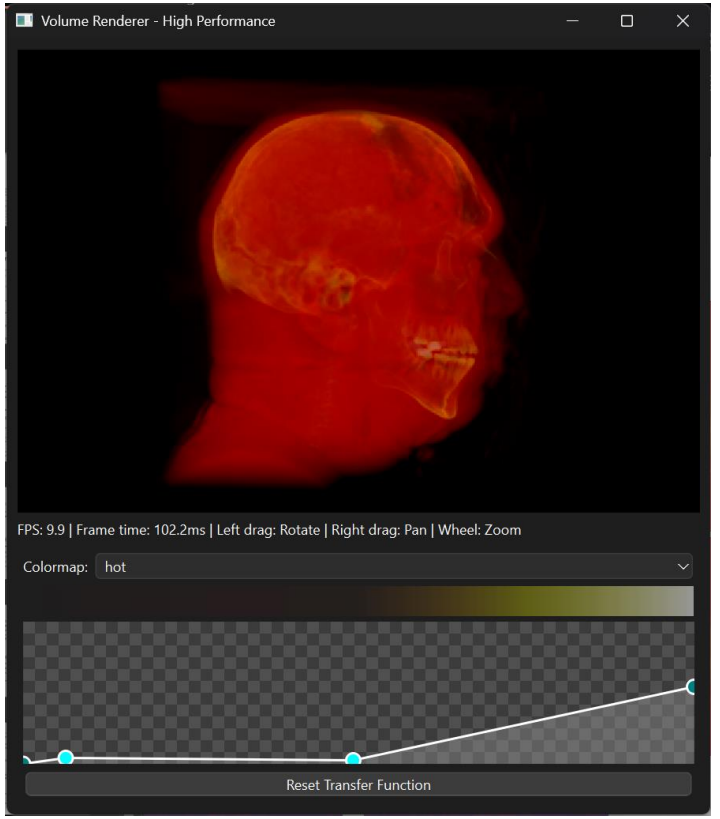
What we are working on

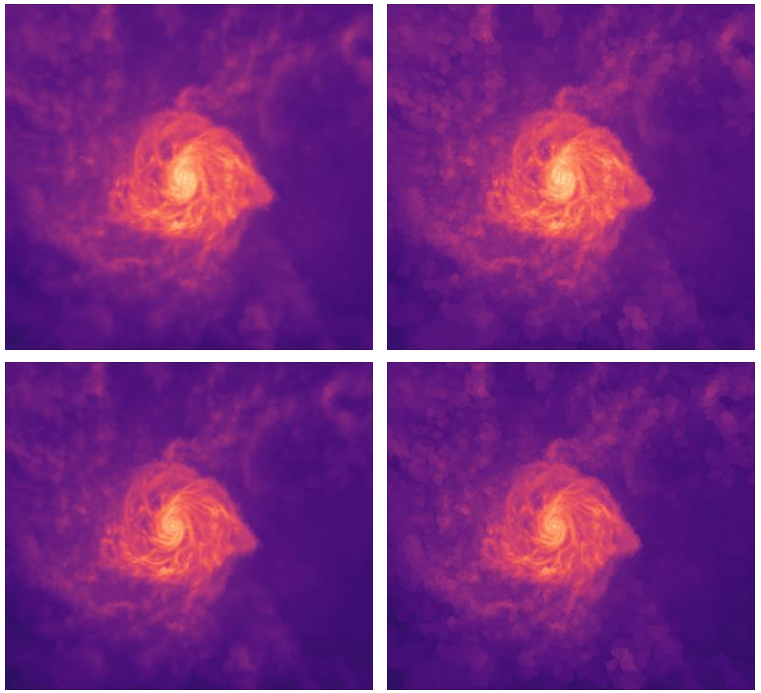
- Support for data importance sampling and empty space skipping
- Support for different/compressed representation of volume data
- Quality comparison and metrics for comparison



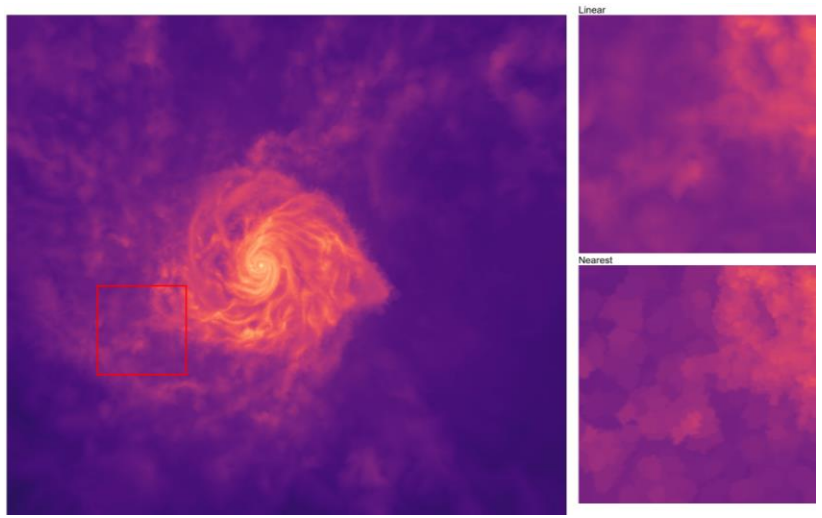
Neural Representation (Incomplete)

A visualization of the Head dataset (open sci vis) to verify the implementation





Linear Nearest Neighbour



Comparison of Different Resampling Techniques

Illustris Subhalo Cutout 568590

Thank You

References

- M. Piochowiak, T. Rapp, and C. Dachsbacher, “**Stochastic Volume Rendering of Multi-Phase SPH Data**,” *Computer Graphics Forum*, vol. 00, no. 00, pp. 1–13, 2020. doi: 10.1111/cgf.14121.
- Ralf Kaehler, "**High-Quality Volume Rendering of Dark Matter Simulations**" in *Proc. IS&T Int’l. Symp. on Electronic Imaging: Visualization and Data Analysis*, 2018, pp 333-1 - 333-8, https://doi.org/10.2352/ISSN.2470-1173.2018.01.VDA-333

Presented by:
Utkarsh Sharma
Akash Maji