Attribute-Aware RBFs: Interactive Visualization of Time Series Particle Volumes Using RT Core Range Queries

Nate Morrical* Stefan Zellmann† Alper Sahistan* Patrick Shriwise‡ Valerio Pascucci*

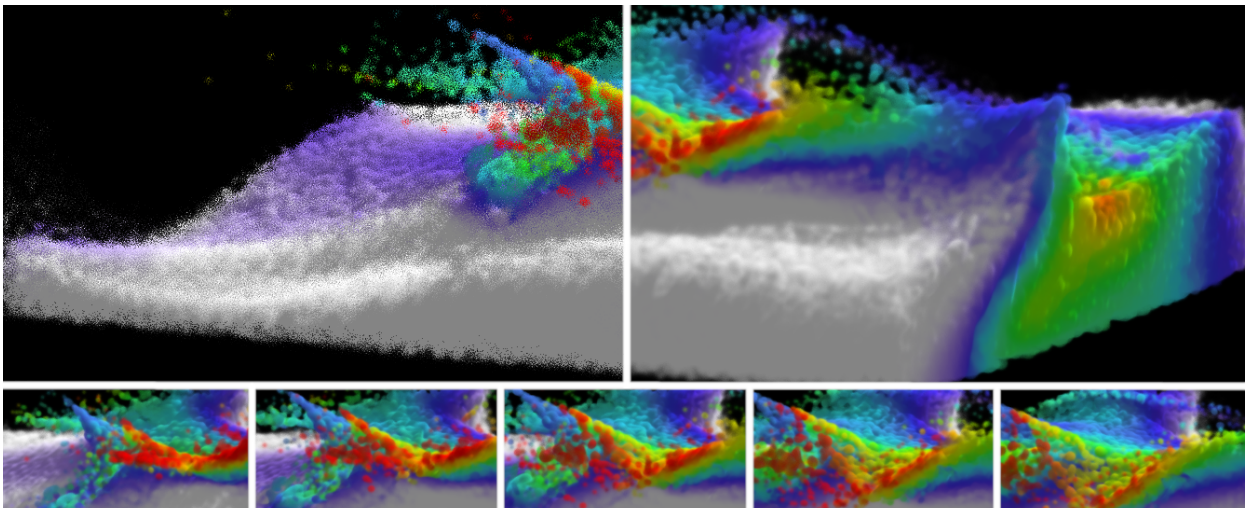


Fig. 1: The "Cabana Dam Break" data set, rendered interactively with our method at 46 FPS, 4 samples-per-pixel per-frame with volumetric shadows (left is 1 frame, right is 1024 averaged frames, bottom row are progressing time steps). GPU-accelerated tree construction and blue noise approach enable interactive animation and improved perception over time. (See supplemental for a video)

Abstract—Smoothed-particle hydrodynamics (SPH) is a mesh-free method used to simulate volumetric media in fluids, astrophysics, and solid mechanics. Visualizing these simulations is problematic because these datasets often contain millions, if not billions of particles carrying physical attributes and moving over time. Radial basis functions (RBFs) are used to model particles, and overlapping particles are interpolated to reconstruct a high-quality volumetric field; however, this interpolation process is expensive and makes interactive visualization difficult. Existing RBF interpolation schemes do not account for color-mapped attributes and are instead constrained to visualizing just the density field. To address these challenges, we exploit ray tracing cores in modern GPU architectures to accelerate scalar field reconstruction. We use a novel RBF interpolation scheme to integrate per-particle colors and densities, and leverage GPU-parallel tree construction and refitting to quickly update the tree as the simulation animates over time or when the user manipulates particle radii. We also propose a Hilbert reordering scheme to cluster particles together at the leaves of the tree to reduce tree memory consumption. Finally, we reduce the noise of volumetric shadows by adopting a spatially temporal blue noise sampling scheme. Our method can provide a more detailed and interactive view of these large, volumetric, time-series particle datasets than traditional methods, leading to new insights into these physics simulations.

Index Terms—Ray Tracing, Volume Rendering, Particle Volumes, Radial Basis Functions, Scientific Visualization,

1 Introduction

In high-performance simulation, the use of volumetric particle representations is widespread. Their memory representation is relatively compact, as a particle requires only a position and a radius. Particles can also carry corresponding scalar attributes, like velocity or temperature. Similar to finite element meshes, particles have the advantage that they can be placed anywhere in the computational domain and adapt to the underlying frequency of the data. However, unlike finite elements, these particles do not require memory-intensive connectivity information. Instead, they can be modeled using radial basis functions that naturally combine and blend together. This has the additional benefit of allowing particles to move freely in space without concern over re-meshing. Because of this ease of expression, particle representations often lend themselves to mesh-free simulation methods like Smoothed Particle Hydrodynamics (SPH) [6, 21].

However, this flexibility poses challenges to *interactive* visualization tools, as these often need to structure the data a priori or on the fly. Approximate methods splat particles onto the screen or into structured grids, causing overdraw issues or atomic contention. When particles have color attributes, post-interpolated grids (where cells average particle attributes before colormapping) produce incorrect results for categorical data, while pre-interpolated grids (where cells average pre-colormapped attributes) consume a lot of memory, and both fail to capture fine details; while splats require sorting particles from front to back to composite, which fails when particles overlap.

As various methods have been proposed to visualize these particle volumes, GPU architectures themselves have greatly evolved. Today, many GPU vendors include ray tracing cores, otherwise known as "RT cores", where ray traversal through an acceleration structure is implemented in the silicon of the GPU itself. GPU ray tracing frameworks

*Nate Morrical, Alper Sahistan, and Valerio Pascucci are with the Scientific Computing and Imaging Institute at the University of Utah

†Stefan Zellmann is with the University of Cologne

‡Patrick Shriwise is with Argonne National Laboratory

also include high performance tree construction routines, reducing preprocessing times and implementation complexity.

Leveraging these RT cores, recent work by Knoll et al. [15] visualize volumetric particles in a very different manner from prior methods. Their approach marches rays through the volume from front to back, and as rays intersect particles out-of-order, these particles are stored in a ray payload stack. Then, each ray segment sorts this stack of particles front to back before compositing these particles together.

This method is very enticing, as it does not require particles be sorted on the CPU during camera manipulation. Instead, only a small intersected subset of particles need to be sorted. Then, when pixels reach a saturation in opacity, the marching process can return without needing to process occluded particles further back. Regarding visualization quality, this approach can composite particles directly without the need for voxelization. And because this method is compatible with RT core frameworks, visualization tools can offload the technical complexity of GPU ray tracing to the driver and leverage the included, fast tree construction routines to avoid long preprocessing times.

Unfortunately, this approach also comes with several compromises. One issue is that particles are interpreted as view-aligned disks, which prevents them from overlapping and blending volumetrically. Instead, the disks discretely pop over or under each other as the camera moves. Another challenge is that the size of the ray payload stack is fixed, as per current GPU architectures. If the stack is too small, it will overflow when many particles overlap a ray segment, resulting in missing or dropped particles. If it is too large, registers spill to global memory, impeding interactivity. This problem are particularly apparent with large radii for smoother blending, as the particles become more likely to intersect rays. Then, despite ray intersection being hardware-accelerated, intersecting many particles can still cause overdraw-like issues where compositing becomes the bottleneck. Still, if these issues could be addressed, it might very well be possible to use RT cores to achieve truly interactive particle volume visualization with little to no preprocessing times or visual compromises.

Therefore, we explore a more robust solution to RT core accelerated particle volume rendering, taking inspiration from recent works on SPH particle rendering [14, 25, 40] that focus on visualization quality. We substitute stack-based particle intersection with a stackless radius range query, collecting and interpolating particles surrounding the query point to reconstruct the underlying scalar field. Then, to support per-particle attributes, we describe a novel radial basis function (RBF) integration scheme that computes a weighted color average in addition to local particle density. These blended colors can be combined with an RBF density map to gain insights into where particles overlap, what they represent, and how they contribute to the final result.

From there, we explore another key advantage of RT core frameworks—namely GPU-parallel tree construction—to enable interactive time series rendering. We leverage acceleration structure refitting to allow for interactive control over the degree of overlap of the particles. Then, to reduce memory consumption, we take inspiration from recent methods [7, 25] and cluster particles at the leaves.

Finally, we explore stochastic volumetric shadows to achieve higher fidelity visualizations with improved depth perception. Noise from volumetric path tracing can make it difficult to track particle movement over time. To address this, we implement a single-scattering model using a blue noise stochastic ray marching approach [41]. This improves the structure of the resulting noise in single-sample-per-pixel images, making it easier to track particle movement.

More specifically, we present the following contributions:

- an "Attribute-Aware RBF" interpolation scheme,
- an application of RT core range queries to accelerate particle field reconstruction,
- an exploration of GPU-parallel tree construction and refitting to enable user-driven per-particle radii and time series data,
- an application of Hilbert clusters to reduce the size of these trees to enable visualization of larger particle volumes,
- and lastly, a strategy to utilize blue noise for stochastic volumetric shadows for improved time series visualization analysis.

2 Related Works

2.1 Particle Volume Rendering

One class of methods to visualize particles is to rasterize them as elliptical Gaussian "splats" onto the screen [9], using additive compositing to determine a density of particles through a pixel [5, 34]. But when particles have color-mapped attributes associated with them—attributes like temperature or velocity—they must be sorted from front to back, then composited one by one [8]. This compositing is an approximate solution, as it cannot handle when colored particles overlap and mix together volumetrically. And as these volumes grow, this sort slows down the visualization to non-interactive framerates.

Another common strategy is to rasterize particles into structured grids [2,4,29], then visualize those grids using methods like ray marching [22] or null collision methods [18,42], whose algorithmic complexities are independent of data size. However, this particle-to-grid rasterization preprocess is costly, especially when many particles influence a common voxel and atomic contention serializes execution. In some methods, this grid splatting process must occur every frame [4]. Low-resolution grids can undersample data, making it impossible to differentiate particles during visualization, particularly when they have categorical attributes (e.g., electrons, protons, neutrons). Too high a resolution can lead to excessive memory usage due to empty voxels; and both problems often occur together as different parts of the simulation have different particle densities.

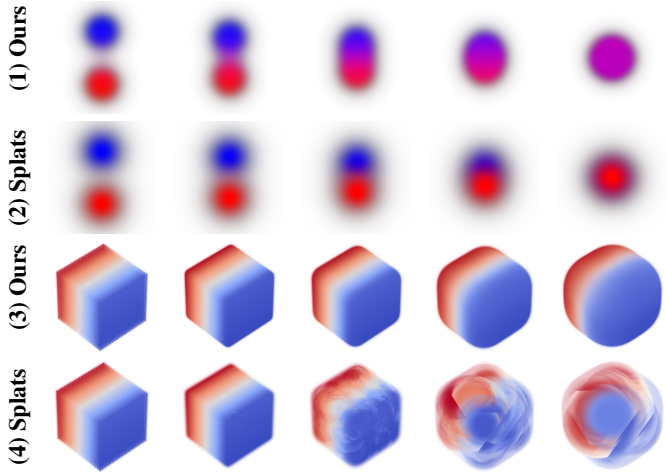
Other works visualize volumetric particles directly through radial basis function (RBF) field reconstruction, reducing algorithmic complexity using linearly indexed particles into BSP trees [12], octrees [32], bounding volume hierarchies [16], and structured grids [33, 35,40]. These methods traverse those data structures during ray marching or tracking to achieve a high quality image. However, building these data structures is an arduous task, and often prevents user interaction with particle radii or the time dimension of the simulation. These data structures can consume a significant amount of memory, and traversing these data structures on GPU architectures introduces load balancing issues or thread divergence that serializes execution, lowering GPU utilization and degrading visualization interactivity.

2.2 RT Core Methods

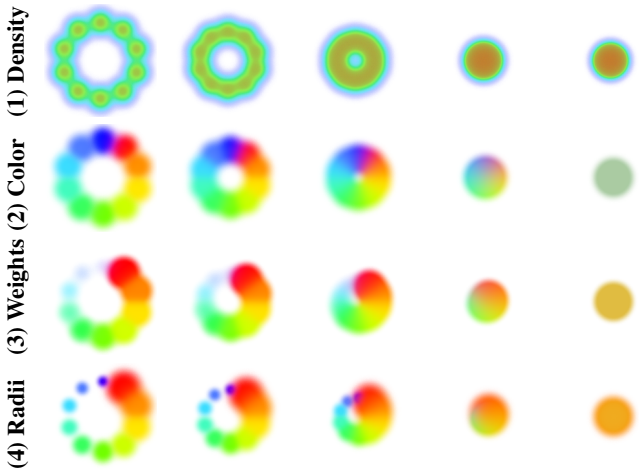
In scientific visualization, ray tracing cores have been used with great success to visualize otherwise challenging data modalities on GPU architectures. Early works by Wald et al. [28, 37, 38] and Morrical et al. [27, 28] use ray tracing cores to accelerate point location of finite element meshes, by substituting a point-based traversal for a ray traversal. Later work by Wald et al. [39] and Zellmann et al. [47] explored data transformations enabling these ray tracing cores to interpolate neighboring same-level-cell regions in the context of adaptive mesh refinement (AMR) visualization. Zellmann et al. build off this latter work and describe the system using ray tracing cores for rendering time series AMR data [45] and AMR flow visualization [44].

Closely related to our work, Zellmann et al. [46] demonstrate a connection between point location queries and range queries, and use this connection to implement a fast physics-driven graph layouter. Their work demonstrates significant performance improvements over software-based traversal strategies, and leverages GPU-parallel tree construction to allow the graph to change from step to step. Later work by Evangelou et al. [3] extend this idea to implement a truncated K-nearest-neighbors query. These concepts have successfully been used by Zhao et al. [48] to accelerate particle-based simulations.

Also closely related to our work is the method by Knoll et al. [15], which as described before uses RT cores to accelerate a particle volume splatter. Gralka et al. [7] observe that constructing hardware accelerated trees over particles consumes a significant amount of memory. Their work proposes a hybrid strategy which clusters particles into memory-efficient PKD treelets, which are then stored at the leaves of hardware compatible trees. This reduces memory consumption while maintaining benefits from hardware acceleration. Recent work by Morrical et al. [25] propose a simpler and faster Hilbert clustering strategy, in the context of finite elements.



(a) Attribute-Aware Radial Basis Functions (1) use a world-space color blending, overcoming compositing limitations of prior splatting methods (2). Since our approach uses a stackless traversal over overlapping points (3), we safely avoid stack overflow issues exhibited by prior work [15] when ray-particle overlap is large (4).



(b) Prior works visualize particle volumes using only the integrated RBF density field (1). Additionally, we can use our Attribute-Aware-RBFs to visualize per-particle properties, for example, per-particle colors (2). These attributes can additionally weight per-particle RBF contributions (3) or control per-particle radii (4).

Fig. 2: An illustration of RBF field reconstruction as ten particles move towards each other from left to right. (Also see the supplemental.)

3 Rendering Colored Time Series Particle Volumes

Taking inspiration from these prior works, here we describe our method for interactively rendering colormapped time series particle volumes on modern GPU architectures using RT cores.

In Section 3.1 we explain how to interpolate points in space to reconstruct the volumetric field. In addition to particle density, here we also describe how to interpolate color-attributed particles using a radial basis integration scheme. Then, we show how ray tracing cores can be used to accelerate this reconstruction process.

In Section 3.2 we describe how to leverage GPU-parallel tree construction routines to enable interactive exploration of particles over time. We also leverage a variant of tree construction, called "tree refitting", to allow users to manipulate particle radii by corresponding scalar attributes. Then, we adapt these construction routines to reduce memory consumption and improve build time.

In Section 3.3 we describe how we use this accelerated field reconstruction for direct volume rendering. There, we cover how to achieve stochastic volumetric shadows for improved depth perception, and also how to reduce noise in these shadowed regions to improve perception of data over time.

3.1 Field Reconstruction

By themselves, mathematical points have no volume, and only define a location in space. But for visualization purposes, we want these points to represent a sampling of the volumetric field around them. Therefore, we need to clarify the behavior we expect when transforming a cloud of particles into a continuous field of values. We refer to this transformation process as *field reconstruction*.

As this field is sampled closer and closer to a particle, that particle should influence the sampled location more and more; and as this field is sampled further away, that particle's influence should fall off with distance. Neighboring particles should combine together to achieve a smooth field; and to enable the exploration we want to remap this field to hide certain values and reveal others. This will also give us control over how particles blend together since we can choose only to show locations where multiple particles overlap.

We also want to use a colormap to visualize *the attributes* associated with each particle, in addition to the relative density of those particles in space. When particles are nearby, we aim to have a natural blending between color-mapped attributes. To our knowledge, no prior work on high-quality particle sampling accounts for this. So in Section 3.1.2, we rethink the idea of RBF-based particle sampling, and modify these RBFs to drive a weighted color averaging.

3.1.1 Density Field Φ

To visualize particle density, we implement an integration scheme over radial basis functions (RBF). An RBF is a real-valued function φ which takes as input two points in space, a sample point x and a center point c_i, and returns a value based on the distance between these two. We use a truncated Gaussian variant of this RBF definition, where this value drops to zero if the distance between these two points exceeds a given radius r_i. In practice, we chose this RBF to be a Gaussian distribution truncated three standard deviations away from the mean, as this value tightly bounds the particle while still producing an artifact-free volumetric field. Lastly, we supply a weight w, used to enable or disable a particle's influence on the field, which we define as a user-driven mapping from the per-particle attribute s_i.

$$\hat{\varphi}(d,r,w) = w * e^{-\frac{1}{2}(\frac{3d}{r})^2} \tag{1}$$

$$\varphi_i(x,c_i,r_i,s_i) = \begin{cases} \hat{\varphi}(||x-c_i||, r_i, w(s_i)), & \text{if } ||x-c_i|| \leq r_i \\ 0, & \text{if } ||x-c_i|| > r_i \end{cases}, \tag{2}$$

Using these density RBFs, we can define a scalar density field Φ as the sum of all RBFs contributing to a point in space:

$$\Phi(x) = \sum_i^n \varphi_i(x,c_i,r_i,s_i) \tag{3}$$

We can then map this RBF density field to an optical density field to direct what final density values constitute surfaces or volumes in our visualization, and can also directly colormap this RBF density field to gain visual insights on where particular density values occur.

3.1.2 Attribute Field Θ

Alternatively, we may want to visualize per-particle attributes using a colormap. In quantitative cases—for example, a temperature or mass associated with each particle—we want to show a direct spatial blending of these attributes; and for particles that are qualitative—e.g. an enumeration of particles as either "electrons", "neutrons", or "protons"—we want to classify the presence of these in space using distinct colors. This is often referred to as pre-vs-post-interpolative classification and ultimately we want to support both of these cases.

Therefore, rather than visualizing the RBF density field directly, we use these basis functions to interpolate per-particle attributes. For post-classifications, continuous attributes blend gradually in space; and for pre-classifications, overlapping particles of different classes can be

Fig. 3: An illustration of RT core range queries being used for radial basis function (RBF) field reconstruction. Given a collection of particles (a) with varying positions, colors and radii, we want to efficiently reconstruct the density field Φ and attribute field Θ at the query point (the open circle in white). Particles are bound by their support radius r (b), which in turn are bounded by axis aligned boxes compatible with RT cores (c). We trace a zero-length ray whose origin is the query point, and RT cores cull away all particles whose support do not overlap the query point (d). The RBF contributions of all particles within range are integrated during traversal, producing the desired Φ and Θ values in (e).

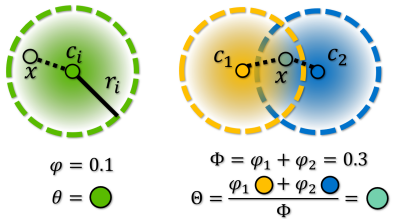


Fig. 4: (left) The influence of a particle φ falls off with distance and is truncated to radius r, while a particle's color θ remains uniform throughout this radius. (right) These radial basis functions are summed to produce a volumetric density Φ, and produce a weighted color average Θ.

identified by the presence of mixed colors, such as purple in a field of blue electrons and red protons—albeit with careful curation of a colormap to avoid ambiguities.

To achieve this intended behavior, we can extend our previous RBF density field. We start by defining a single particle's attribute as a uniform $\hat{\theta}$ throughout the particle's radius r_i, and zero otherwise. This $\hat{\theta}$ is defined as either the per-particle attribute s_i, or a user-driven colormapping of s_i.

$$\theta_i(x, c_i, s_i, r_i) = \begin{cases} \hat{\theta}(s_i), & \text{if } ||x - c_i|| \leq r_i \\ 0, & \text{if } ||x - c_i|| > r_i \end{cases}, \tag{4}$$

Using these attributed RBFs, we can define an attribute field Θ as a weighted average of all particle attributes influencing a given location in space, where the weight of each attribute is derived from the previously defined density RBF φ_i of that particle at the same location.

$$\Theta(x) = \frac{\sum_i^n \theta_i(x, c_i, r_i) * \varphi_i(x, c_i, r_i)}{\Phi(x)} \tag{5}$$

This weighted average results in a natural blending of colormapped attributes while still retaining a uniform color for when particles do not overlap, and whose influence falls off with distance.

3.1.3 Optimizing Field Reconstruction

With the definitions of Φ and Θ above, we can now visualize our particle data volumetrically. However, to visualize these datasets *interactively*, we must be capable of sampling this scalar field hundreds of times per pixel per frame in only a couple milliseconds. A naive approach would be to iterate over all particles in our data, summing up each particle's density and averaging all contributing colors together; but as the number of particles increase, this will quickly become a bottleneck. However, because we truncate the influence of a particle to a finite radius around it, only a small set of particles will actually influence our field at any given point in space. Therefore, we can substitute this exhaustive traversal for a search that returns only the particles within range of our query point.

In computational geometry, this problem is otherwise known as a "radius range query". Traditionally, these queries are implemented on the GPU either using linearly indexed particles in a regular grid, or by using bounding volume hierarchies. Unfortunately, both of these schemes come with certain disadvantages. With grids, load balancing is an issue in cases where particle density is highly non-uniform, where the majority of particles will fall into a select few cells and result in near-exhaustive traversals. On the other hand, tree traversal on Single-Instruction-Multiple-Data (SIMD) units introduces thread divergence on GPU architectures, which serializes execution.

Fortunately, modern GPU architectures support hardware-accelerated tree traversal as part of ray tracing coprocessors. These RT cores act more like Multiple-Instruction-Multiple-Data (MIMD) units [31], which are more resilient to divergent tree traversal. As demonstrated by Zellmann et al. [46], we can (ab)use these units to implement a hardware-accelerated range query. Their work uses range queries in a two-dimensional graph context to simulate the repulsive forces between nearby nodes, but we can easily extend this idea to support our three-dimensional particle RBF integration (cf. to Equations 3 and 5).

We begin by building a bounding box over every particle in our data with a radius r_i. Then, we pass these bounding boxes to our ray tracing API, using the built-in tree constructor. By constructing our tree this way, we "bake" the ranges of our particles into the tree. Then, when we want to traverse through all particles within range of a given location, we trace a ray whose origin is set to that query location. We set the *tmin* and *tmax* values of the ray to 0, turning the ray query into a point query. Due to the inner workings of these RT cores, we must set the ray direction to something with non-zero length, so we set the direction to a constant $(1, 1, 1)$. Finally, as this ray intersects boxes at the leaves of our tree, if the ray origin lies within the radius range of the corresponding particle, we can conclude that particle is within range. An illustration of this method can be found in Figure 3.

To use these range queries for field reconstruction, we reserve several ray payload registers, one to hold our density field value Φ and a small number of registers to hold our attribute field values Θ. We initialize these values to 0, and then call the appropriate intrinsic to dispatch ray traversal to our RT cores. As rays intersect our range boxes, RT cores return for intersection testing. We test to see if our ray origin is contained within the intersected particle's radius, and if so, we report a potential intersection, passing the evaluated particle's distance d, scalar attribute s and radius r as "hit attributes".

From there, execution in the fixed function ray tracing pipeline moves to "any hit" evaluation. We use the given particle's scalar attribute s to determine that particle's weight w. For post-interpolation we use the attribute s as our $\hat{\theta}$ directly, but for pre-interpolation we use this s to instead lookup a color for the given particle, and assign that to our $\hat{\theta}$. Then we use the given distance d, radius r and weight w to evaluate our particle's RBF density $\hat{\varphi}$ at the queried location. Once this is evaluated, we add our particle's contributing density $\hat{\varphi}$ to our total density field value $\Phi(x)$ in our ray payload. We also weight our $\hat{\theta}$ by this RBF density $\hat{\varphi}$ and add the result to our total attribute field

value Θ in our ray payload. Finally, we ignore the hit, tricking our RT cores to continue traversal to the next particle within range.

When ray traversal completes, the returned value stored in Θ technically represents just the numerator of our weighted average. We divide this accumulated value by the returned Φ density value to solve for the true Θ value. If pre-interpolating the attribute field, this attribute field Θ can be colormapped; and with post-interpolation, this attribute field can be visualized directly. From here, we can also map the integrated density field value Φ to an optical density using a user-driven density map. If users wish to visualize the density field directly, we can substitute the previously returned color value with a colormapping of the returned density field value Φ.

3.2 Tree Construction and Refitting

By using hardware accelerated ray tracing, we also benefit from high performance tree construction. Prior particle volume rendering methods construct a hierarchy offline on the CPU [14]; but if particles move or radii are edited, this hierarchy must be rebuilt. Because we are no longer limited to offline tree construction, we now have the opportunity to interactively move and resize particles to enable additional exploration. (See also Table 1).

3.2.1 Interacting with Time

To handle particle movement between time steps, we allocate a buffer of axis-aligned bounding boxes in advance, with one box per particle. Then, whenever the simulation progresses or the user interacts with the time dimension of the data, we compute these bounding boxes in parallel on the GPU using a compute shader. This compute shader adds and subtracts the given particle's radius from its current origin, storing these two extremes into the bounding box buffer.

Some simulation codes also introduce new particles or remove particles as the simulation progresses. For our application, we assume a maximum theoretical number of particles to avoid expensive reallocation of the bounding box buffer. Then, if a box contains no points, possibly due to one timestep having fewer particles than another, we set the minimum and maximum bounding box coordinates to floating point *NaN*. This disables the box from being considered by our ray tracing graphics API during tree construction. Finally, we pass this buffer directly to our ray tracing framework's real-time tree construction routine, recycling existing scratch memory required by this tree construction to avoid stalls from buffer reallocations.

3.2.2 Interacting with Particle Radii

To handle particle radii manipulation, we can *refit* our trees rather than rebuilding them. This refitting process is significantly faster than full tree construction, and allows for smoother particle radii manipulation, especially for very large particle volumes. Here, refitting is allowed, since radii manipulation does not change the underlying tree topology.

In practice, we introduce another user-driven map, which we call a "radius map", which allows users to control individual particle radii, and can be used as a more efficient means of hiding particles. With the RBF weight in Equation 1, near-zero weighted particles are still intersected by range queries, and result in redundant computation. By instead hiding particles by radii, users reduce bounding box overlap in the tree and avoid this unnecessary computation, effectively enabling empty-space skipping.

To enable per-particle RBF manipulation, we modify our bounding box compute shader to read the current particle's scalar attribute s from the radius map to set a unique particle radius. For convenience, we specify a global RBF particle radius, then have this radius map return a percentage, which we multiply by the global RBF particle radius to compute our final particle radius. Then, we use the appropriate refitting instructions supplied by our ray tracing framework to account for these updated particle bounds.

3.2.3 Reducing Memory Consumption by Clustering

Finally, with some large datasets, memory consumption can be an issue—especially for consumer GPUs with limited memory resources.

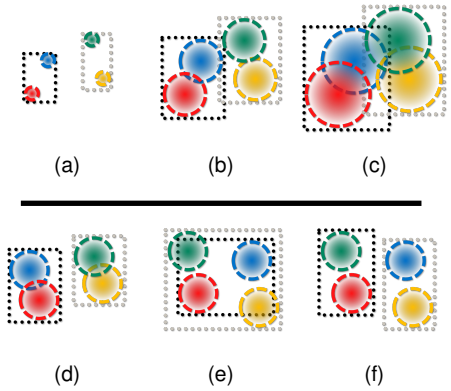


Fig. 5: Illustrations of bounding boxes over a dynamic scene: On the top figure we see particles(a) with expanding radii(b) that can be handled by tree refitting(c); on the bottom, we see particles with changing positions(d) that may create suboptimally overlapping bounding boxes(denoted with dotted lines)(e) hence requiring rebuilds(f).

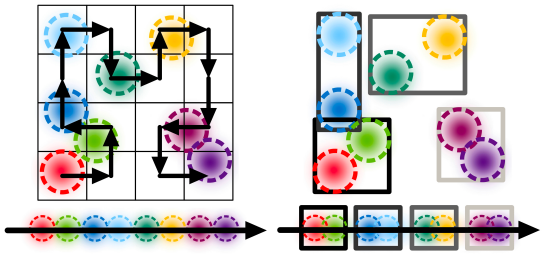


Fig. 6: Sorting the particle along a Hilbert curve to quickly and efficiently create spatially coherent clusters(denoted in solid lines).

At the moment, we need to account not only for the particles themselves, but also their bounding boxes. However, a buffer of bounding boxes over each particle will alone be twice as large as the particle data itself, since each axis-aligned box must be defined using two additional points, a minimum corner and a maximum corner. To reduce memory consumption, we can reduce the number of bounding boxes we create by clustering nearby particles into the same box.

To do this clustering, we follow in the footsteps of recent works [7, 28], and reorder particles along a Hilbert space filling curve. This reordering operation can be done either on the CPU, or on the GPU using a parallel radix sort. By reordering particles this way, nearby particles in space will also be nearby in memory. Then, rather than compute an individual bounding box over every particle, we can build "cluster" bounding boxes over sets of N neighboring particles in memory. The first cluster contains particles $0 \rightarrow N-1$, the second cluster contains particles $N \rightarrow 2N-1$ and so on (See Figure 6).

This will come at a slight cost to field reconstruction performance, as now with every intersected cluster, all particles in the cluster must be tested for intersection against our query point. However, when neighboring groups of particles are likely to overlap the query point already, processing all of these nearby particles together in a memory coherent linear process can be more efficient than a more intensive depth first search through the same set of particles.

3.3 Direct Volume Rendering

Using radial basis functions, we can reconstruct our scalar field at any point in space, including both the integrated RBF density field Φ as well as our colormapped attribute field Θ. We also have the means to quickly reconstruct this scalar field thanks to hardware acceleration. Then, we can use GPU-parallel tree construction to animate these fields interactively, and can use tree refitting to manipulate particle radii. With this, we have a solid framework in place to enable direct volume rendering of our time series particle volumes.

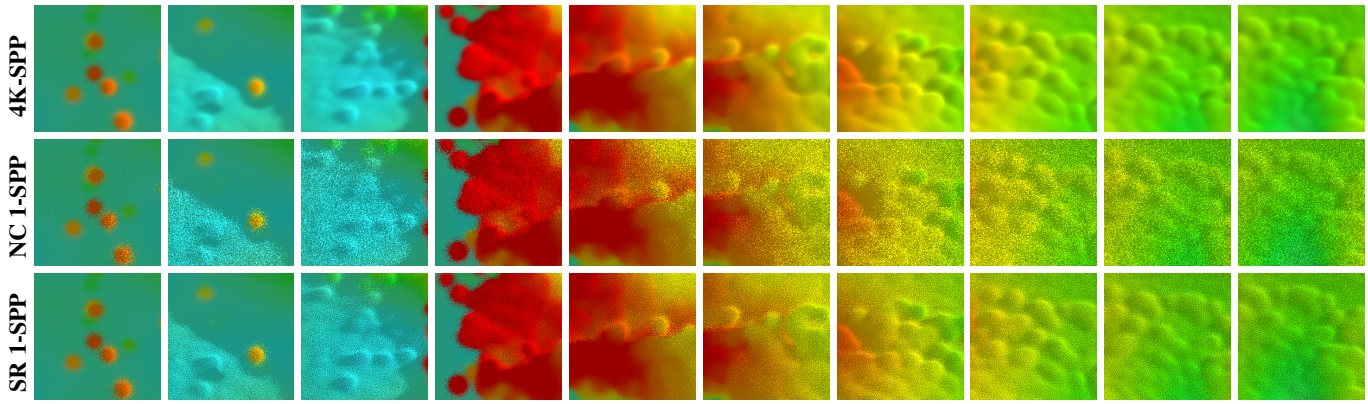


Fig. 7: An illustration of different volume rendering strategies for the Cabana Dam Break dataset over time from left to right (rendered interactively at 45 FPS). Single-Scattering (4K-SPP) highlights depth through shadows, but requires time to converge. Null Collision methods (NC 1-SPP) decorrelate input random numbers from output luminance, causing to significant noise in single sample per pixel configurations. Stochastic Ray Marching (SR 1-SPP) preserves this correlation, allowing for blue noise patterns in the final image structure of single-sample-per-pixel images, greatly improving image clarity. (We recommend zooming in to evaluate noise levels, or observing these results in the supplemental material.)

3.3.1 Emission and Absorption

One common approach to direct volume rendering is to use the emission and absorption lighting model. With this model, samples within participating media emit light that can be observed directly by the viewer. This emission is absorbed exponentially by the volume depending on the optical thickness of the media. We see the effects of this absorption in the form of occlusion, where opaque media in front occludes other media further back.

Starting at the camera origin, we generate a set of view-aligned rays to test for intersection against a global bounding box containing our particle data. We then march these rays from their entry point in the box to their exit, sampling our scalar field at every step. We use the RBF density field Φ as our optical density and the Beer-Lambert law to compute the transmittance along the ray, which we use to composite the corresponding Θ color values from front to back until the ray exits the bounding box or the pixel's opacity reaches saturation. To avoid "wood grain" artifacts from the regular step size, we jitter each ray with a random number, and average these results together over time.

3.3.2 Stochastic Volumetric Shadows

A draw back of the emission and absorption lighting model is that viewers can find it difficult to comprehend depth within the data. This is because we often depend on external light sources and shading to give shape to the implicit geometry within our volume [20, 43]. Therefore, to enable better recovery of depth information during visualization, we can extend this emission and absorption model to include volumetric shadows. These shadows provide more insight into the volume by allowing viewers to discern cracks and crevices through the interaction of light with surfaces inside the volume. Additionally, shadows cast by these surfaces can offer additional depth cues, especially if the user is able to manipulate the origin of the light.

To implement volumetric shadows, we can move over from an emission and absorption model to a single scattering model. Rather than interpreting sampled colors as emission sources, we instead interpret these colors as the *albedo*—or reflectiveness—of our media. With this change, the appearance of our media now depends on how much light is received at a given location in space. This can be controlled by manipulating the location of the light, or by rendering certain locations of the media optically transparent.

Unfortunately, with alpha-composited ray marching we run into a scalability issue, since as is, we have many samples along our viewing rays that need shading. To shade a given sample point, we must trace a secondary shadow ray to our light source. These shadow rays must march through the media—just like our primary rays—to determine how much light our volume transmits. Though our field reconstruction is fast, if every primary ray traces secondary shadow rays at each sampling point, we will quickly reach performance limits and lose our visualization interactivity.

Instead, we can reduce shadow sampling expense by substituting ray marching for a null collision method. This uses a Monte Carlo sampling process to sample equally likely distances into the volume (called "free flight distances"). Since the Beer Lambert law produces absorbance given a distance, we can invert this function to generate free flight distances given random absorbance values. In heterogeneous media, inverting absorbance is difficult, since optical thickness varies spatially. Null collision methods solve this problem by introducing "null particles" that homogenize the volume and enable inverting absorbance without altering the volume's appearance. Then, a rejection sampling process is used to determine if the current sample is a null collision event or a scattering event. For a more fully detailed explaination of null collision methods, we recommend the distance sampling section of the SIGGRAPH course by Novak et al. [30].

By sampling free flight distances, we can reduce the number of shade points along our viewing ray from N to 1. Therefore, we only need to trace one shadow ray for that one shade point, which significantly reduces sampling expense per frame. The compromise with this approach is the introduction of significant noise, similar to Monte Carlo path tracing, which must be converged over time.

3.3.3 Reducing Image Variance with Blue Noise

Null collision methods work well for visualizing static datasets where the image has time to converge. However, the high degree of noise present in single-sample-per-pixel images make it difficult to comprehend data that animates over time. Traditional methods use some variation of temporal antialiasing to converge these stochastic effects in the presence of motion, but these techniques require motion vectors, which we cannot easily compute due to the volumetric nature of our data. Alternatively, we could increase the number of samples taken per-pixel per-frame, but this would slow down our visualization to non-interactive framerates.

We observe that the difficulty in comprehending single-sample-per-pixel images from null collision tracking stems from a lack of structure in the noise, especially in the shadowed regions we depend on to perceive depth and form. At the same time, this noise is unavoidable, since null collision trackers use rejection sampling to handle null scattering events along the ray. Even with a high-quality random number generator, this rejection sampling process destroys any correlation between the input random noise and the output image structure.

Fortunately, there are other methods to compute free-flight distances. First, we can describe absorbance F as a function of t, where t

represents a distance along a ray:

$$F(t) = \xi = 1 - e^{-\int_0^t \mu_t(s)ds} \tag{6}$$

In the equation above, $\mu_t(s)$ represents the volumetric extinction at a point along our ray. The exponential term represents the Beer-Lambert Law, which models the transmittance of light through the volume as an exponential decay dependent on the distance traveled and the overall extinction along that distance.

In effect, the equation above produces an absorbance ξ between 0 and 1 given a free-flight distance t. Our objective then is to invert the above function, such that we can supply a single random absorbance value ξ between 0 and 1 to obtain a free flight distance t. Note, it is important that we require only one such random value ξ, so that we can maintain a strong correlation between the input random numbers and the output sampled distance.

If we attempt to invert the equation above, we will discover that this function can only partially be inverted:

$$\int_0^t \mu_t(s)ds = -ln(1-\xi) \tag{7}$$

This is because the extinction values μ are a function of t, which we simultaneously want to solve for. Fortunately, we can solve for an approximation of the above equation by converting this integral into a Riemann sum:

$$\sum_{i=1}^{n} \mu_{t,i}\Delta = -ln(1-\xi) \tag{8}$$

If we can solve for the right hand side analytically, then we perform a linear search over the left hand side until we find a solution to this equation. Therefore, we begin by solving for the right hand side of this equation by generating a random value ξ. Then, we use a ray marching process to search numerically for a solution to this equation, computing a running sum of the sampled extinction values μ along our ray multiplied by our step size until this sum exceeds the solution to the right hand side. The distance at which the left hand side exceeds the right hand side is an approximate solution to the free-flight distance.

The advantage of the above stochastic ray marching technique to computing our free-flight distance is that we no longer need to use rejection sampling, since we no longer need null particles to homogenize the volume in order to invert absorbance. With this alternative ray marching method, the sampled shading positions into our volume require one random number, and will have a high correlation with the input random number generator. Therefore, we can improve the structure of the volumetric noise by using Spatio-Temporal Blue Noise (STBN) textures, as described by Wolfe et al. [41]. The results of this transformation can be seen in Figure 7.

4 Experimental Results

To evaluate our method, our rendering backend [26] uses Vulkan 1.3 along with the `VK_KHR_raytracing_pipeline` extension to access hardware-accelerated ray tracing functionality on NVIDIA, AMD and Intel GPUs in a Linux based environment. Unless otherwise stated, measurements were taken using an NVIDIA RTX 4090 and an Intel i9 12900K processor. For Figure 10, we additionally include evaluations on an NVIDIA RTX 3060 Ti, an Intel ARC A770, and an AMD RX 6750 XT. All images were rendered at a resolution of 1024×1024 up to 64 samples per pixel, with one sample per frame.

4.1 Datasets

With this hardware, we performed a series of tests on a collection of time series particle volumes of varying sizes (cf. Figure 8):

1) Nozzle represents a simulation used to model jet fuel injection [11]. This dataset consists of an opaque cylindrical structure made of static particles, which serves to concentrate the jet. Over time, this simulation injects new matter into the simulation, resulting in increasing memory size and density variations that present a challenge for RBF particle visualization.

2) Coal Boiler represents a real-world simulation of coal particles being injected into a boiler [1]. Our copy of this data consists of 4.6 million particles at the beginning of the simulation to 41.5 million particles upon simulation completion. This simulation was produced using the Uintah computational framework [23].

3) Cabana Dam Break represents a free surface water column collapse simulation over 138 timesteps, with each timestep containing 768K particles. Particles repel each other, creating a relatively uniform distribution. This simulation was produced with the Cabana particle toolkit from the Exascale Computing Project [36].

4) Viscus Fingers models transient fluid flow obtained through a finite pointset method [17]. A cylinder is filled with pure water, then an unlimited salt concentration is placed on top. Over the course of 120 timesteps, (with approximately 550K particles per step), the highly concentrated salt solution sinks down the cylinder, creating "viscous fingers". This dataset comes from the 2016 Scientific Visualization Contest.

5) HACC Cosmology is an evolution of dark matter and baryon particles over the course of 5 million years, obtained using a CRK-HACC cosmological simulation [10]. This simulation studies the impact that Active Galactic Nuclei (AGN) have on the surrounding matter distribution. This AGN comes from matter building near black holes at the center of galaxies. This dataset comes from the 2019 Scientific Visualization Contest.

4.2 Evaluation

With these datasets, we measure preprocessing time, compression effectiveness, and rendering performance on various datasets and attributes. For "Nozzle", we configured a constant RBF map with 100% density, while all others use a linearly increasing density map. "Boiler" includes a larger particle radius to show overlap, while "Dam Break" and "Viscus Fingers" use a spiky transfer function to demonstrate a mix of volumetric and surface-like behavior. Finally, "Cosmology" demonstrates small features in a large domain.

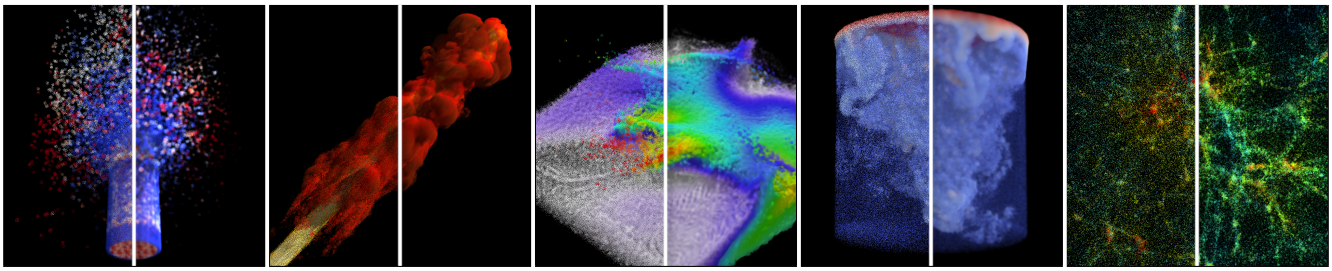
We render the representative view points in Figure 8, showing single sample-per-pixel images on the left and converged 64 sample-per-pixel images on the right. Here we also present timing estimates in both frames/second (FPS) and in milliseconds (ms) per frame for 1024×1024 viewports. Because our sample space is so large, we generally observe a large variation in framerate. Data sets with more particles per step, larger particle overlap, presence of more translucent particles, and small yet dense particles within a larger computational domain all contribute to slower framerates, and vice versa.

Another important aspect of our method is data structure construction and update performance. Ray tracing based methods are well known for their efficiency with respect to the number of input primitives. However, a typical characteristic for SPH simulations is time-varying particle data. While interaction over time series data is trivial with a raster-based splatter, a common criticism towards ray tracing methods is that they are traditionally reserved for visualizing static datasets, due to long acceleration structure construction times. Fortunately, we can build these structures interactively, and are not constrained to these prior assumptions. We report acceleration structure construction time (when time steps change) and update times (when radii change) in Tab. 1. Note that these times are in *milliseconds*.

Finally, we present benchmarks that explore the influence of specific parameters (particle radius, Hilbert cluster size, hardware acceleration, and the impacts of spatio-temporal blue noise) on rendering times, as we found that our method is particularly sensitive to these.

4.2.1 Increasing Particle Radii

Using a range query sampling-based approach means that our method is especially sensitive to particle overlap. Samples taken in dense regions where numerous RBFs overlap have a higher cost because more primitives need to be tested and contribute to the weighted average



(a) Nozzle
100 M Points
768K Points / Step
105 FPS (9.5 ms)

(b) Uintah Boiler
253 M Points
23,079 K Points / Step
6.79 FPS (147.2ms)

(c) Dam Break
105 M Points
768 K Points / Step
46 FPS (21.7 ms

(d) Viscus Fingers
67 M Points
550 K Points / Step
24 FPS (41.7 ms)

(e) Cosmology
330 M Points
500 K Points / Step
64.5 FPS (17.2 ms)

Fig. 8: Datasets used for testing. All datasets are colored using per-particle velocity magnitude. Animations of all these datasets can be found in the supplemental material. All timings above measure noisy single-sample-per-pixel images. Left sides of images show single-sample-per-pixel images, while right sides show converged images at 64 samples per pixel, with both sides using spatio-temporal blue noise (STBN).

from Equations 3 and 5. With large particle clusters, smaller radii also can incur potential costs due to potentially excessive empty space.

To evaluate this, we first explore the influence of varying particle radii. For this evaluation, we visualize datasets over multiple randomized viewpoints representative of a typical explorative visualization session, as culling effectiveness, empty space removal, etc. not only depend on spatial arrangement, but also on camera parameters. This averaging over a number of camera positions also allows us to report a performance *envelope* rather than a static performance estimate.

Based on the representative viewpoints from Fig. 8, we orbit around the centroid of the given datasets/time step and average rendering times over 50 different positions. Then, we pick ranges of radii that we find are sensible choices for our five datasets (the lower end of the range allowing to see through most of the data while the upper end of radii results in significant overlap), and probe that range at uniformly spaced positions. We present results in Fig. 9 (a-e) (the blue polylines) for cluster sizes of 1 (solid) and 16 (dotted).

We observe that render times generally go up with increasing radii, which we primarily attribute to an increase in the number of primitives intersected per sample taken. With decreasing particle radii, we observed that any introduced empty space does not significantly impact visualization performance. This is likely because queries in empty regions intersect very few internal nodes in the containing tree, making these queries inexpensive. Very similar observations were also drawn by prior work [27]. To our surprise, clustering actually improves visualization performance in three of the five datasets tested. We suspect this clustering helps more in cases where particles are themselves highly clustered and overlap is high, where a linear search is more efficient than exhaustive tree traversal.

4.2.2 Particle Clustering

Next, we evaluate Hilbert cluster sizes and their influence on render times *and* memory consumption. We generally find that sensible choices for cluster sizes are power of two's in the (discrete) range $[2^0, 2^4]$. We perform the same benchmark from before using the 50 different camera orbits and now vary the cluster sizes in that range, while keeping the RBF radius fixed as the median particle radius used in the prior test. These results are reported in the same Figure 9, marked using red lines. By sharing a common figure, we can compare the relative sensitivity of our method to cluster size versus particle radius. We contrast these results to the reduction in acceleration structure size reported in Table 2, for the same discrete range of cluster sizes.

Interestingly, we observe that, unlike the radius variable, rendering times are less dependent on cluster sizes. For example, with "Boiler", the effect when increasing cluster size is actually positive, and the red "cluster size" curve is in general lower than the solid blue "unclustered" line. At the same time, the improvement in memory consumption when increasing cluster size is an order of magnitude; cluster and acceleration structure sizes exhibit a near perfect linear correlation. Clustering also yield improvements to tree construction and update times, (cf. Tab. 1), though these times are already quite low, so we suspect the primary advantage of clustering lies in memory consumption.

Table 1: Rebuild and Refit Times (in ms, rebuild before "/" and refit after) for increasing particles per leaf from left to right. We observe faster tree updates with more particles per leaf.

	particles/leaf				
data set	1	2	4	8	16
Viscus	1.0 / 0.0	0.8 / 0.3	0.6 / 0.2	0.5 / 0.1	0.5 / 0.1
Boiler	65. / 10.	34. / 5.9	18. / 3.6	9.4 / 2.1	4.3 / 1.4
Cabana	1.2 / 0.4	0.9 / 0.3	0.7 / 0.2	0.6 / 0.2	0.5 / 0.2
Nozzle	0.9 / 0.3	0.7 / 0.2	0.6 / 0.2	0.5 / 0.2	0.4 / 0.2
Cosmology	1.5 / 0.3	0.9 / 0.6	0.7 / 0.3	0.7 / 0.2	0.6 / 0.2

Table 2: Accel Structure Size for increasing particles per leaf from left to right. We observe that more particles per leaf result in linear memory savings. (in MB. OOM indicates "Out Of Memory")

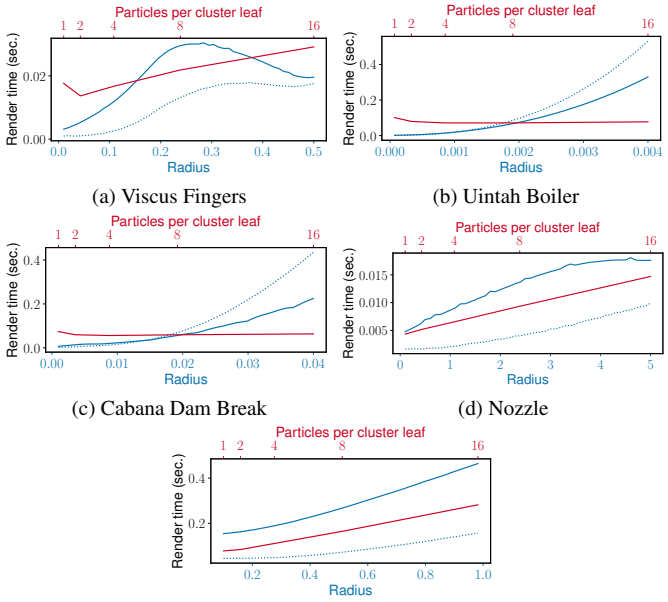
		Particles / Leaf				
Data set	Particles (in MB)	1	2	4	8	16
Viscus	9.0	81.1	40.1	20.3	10.1	5.1
Boiler	704	6200	3100	1600	797	398
Cabana	11.7	106	53.1	26.5	13.3	6.6
Nozzle	6.0	54.4	27.2	13.6	6.8	3.4
Cosmo 500K	8.0	72.5	36.2	18.1	9.1	4.5
Cosmo 50M	563	5000	2500	1200	637	318
Cosmo 500M	7400	OOM	OOM	OOM	8400	4200

4.2.3 Impacts of Hardware Acceleration

With our method, we make use of ray tracing hardware to improve query performance. These ray tracing cores cannot be disabled, making performance improvements difficult to evaluate directly. Instead, we compare our hardware accelerated traversal to a software-based, stack-facilitated, depth-first traversal on the same GPU, using an optimized linear bounding volume hierarchy (LBVH) [19], as these trees can be built in parallel on the GPU at interactive rates [13] to facilitate our time-series particle data.

We compare relative performance improvements in Figure 10 on three different RT core architectures: Intel ARC Alchemist, NVIDIA's Ampere, and AMD's RDNA 2. NVIDIA and Intel RT core implementations accelerate full tree traversal through a Multiple-Instructions, Multiple-Data (MIMD) coprocessor, while AMD's RDNA 2 supports intrinsics for direct ray-box intersection testing on their Single-Instruction, Multiple-Data (SIMD) compute units. (Note that larger speedups in Figure 10 do not necessarily equate to faster render times).

With this setup, we evaluate relative performance improvements on two synthetic datasets and one "real-world" dataset (i.e., the "Cabana Dam Break"). The "Uniform" dataset consists of 60 K particles generated using a Poisson sphere sampling process, while the "Random"

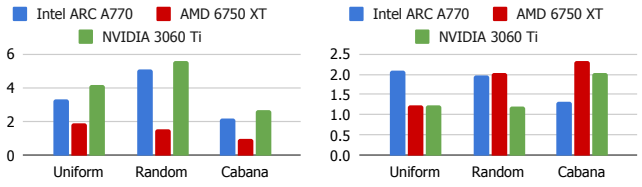


(a) Viscus Fingers (b) Uintah Boiler

(c) Cabana Dam Break (d) Nozzle

(e) HACC Cosmology

Fig. 9: Render time (including shadows) in seconds, as a function of particle radius (blue solid is 1 particle per cluster, blue dotted is 16 per cluster) and as a function of particles per cluster (red solid varies particles per cluster for a fixed median radius taken from the prior sweep).



(a) Speedups from RT Hardware (b) Speedups from Blue Noise

Fig. 10: Relative speedups when rendering three datasets on three different GPU vendors (all including secondary shadow rays and one particle per cluster). "Uniform" and "Random" consist of 60K synthetically generated particles distributed throughout a unit cube, and "Cabana" is the dam break seen in Figure 8. (a) demonstrates speedups from RT Hardware over a software-based tree traversal, while (b) shows speedups of blue noise over a white noise sampling distribution.

dataset perturbates these 60 K uniform samples by a displacement whose magnitude is equal to the radius of the Poisson sphere.

Our findings show that for all architectures, we see performance improvements by using the included RT cores and intrinsics made available through `VK_KHR_raytracing_pipeline`; however, performance improvements are much more significant for Intel and NVIDIA architectures than on AMD's RDNA 2. We speculate that this is due to differences between MIMD and SIMD acceleration structure traversal, as MIMD RT cores implementing full tree traversal are theoretically more resiliant to divergent traversal processes. This hypothesis is further backed by larger observed speedups on Intel ARC and NVIDIA when comparing the randomized particle distribution over the uniform particle distribution. In this case, AMD's SIMD tree traversal intrinsics yield smaller improvements over pure software-based traversal.

4.2.4 Impacts of Spatio-Temporal Blue Noise

With our approach, we use Spatio-Temporal Blue Noise to improve image quality when exploring particle volumes over the time dimension. To our surprise, this change resulted in a noticeable positive impact to render times. The impacts of using a blue noise distribution over a white noise distribution are shown on the right of Figure 10, where we see speedups of $1-2\times$. We suspect this positive impact is because blue noise sampling patterns are more amenable to GPU caching mechanisms than white noise between neighboring threads.

Table 3: A performance comparison against a pre-interpolated 512^3 voxel grid and to Knoll et al.'s RT core splatter [15] for Emission and Absorption versions of the images in Figure 8, using a pre-interpolated classification. To capture time series exploration performance, we rebuild trees every frame for our method and Knoll et al.'s, and for voxels we re-voxelize particles into the grid every frame. All numbers are reported in milliseconds per frame for single-sample-per-pixel images.

Data Set	Dataset				
	Viscus	Boiler	Cabana	Nozzle	Cosmo
Voxels	1564.5	4248.7	699.7	379.1	406.7
- Voxelization	1562.1	4244.6	692.9	375.4	401.9
- Rendering	2.4	4.1	6.3	3.7	4.8
Knoll et al.	204.1	370.4	23.8	5.3	30.2
Ours	41.0	27.9	13.3	6.7	12.0

4.2.5 Method Comparison

Finally, we evaluate the performance of our method against a preinterpolated voxel grid renderer and the method proposed by Knoll et al. [15]. For all methods, we account for any necessary datastructure construction to enable time series visualization. For the voxel grid, we atomically sum particles' RBF contributions into a 512^3 grid using a compute kernel, then divide this sum by the count of particles intersecting each cell. Due to technical constraints of the reference splatter, we limit our renderer to an emission and absorption lighting model. For the method by Knoll et al., we reduce the step size slightly to reduce any severe visual artifacting.

As shown in Table 3, we observe that in all cases, voxel based rendering is dominated by the voxelization process, especially in cases where particle overlap is high, as is the case with "Boiler". This issue is present for both time series exploration as well as colormap and particle radius manipulation. Rendering the resulting grid is fast, as particle RBF integration is done a priori, but results in visual artifacting due to quantization, especially for fine structures like those in "Cosmology". In contrast, both our method as well as that by Knoll et al. can interactively update the required acceleration structures, enabling smooth interaction over time.

Our method is competitive with respect to performance to the RT core splatter baseline. For more surface-like transfer functions like those used in "Nozzle", splatting benefits more from early-ray termination. However, for more optically thin media like the water in the "Viscus" dataset, splatting performance degrades when many transparent particles are intersected but do not contribute to early-ray termination. In these more volumetric cases, it appears that radial basis functions like ours perform significantly better, as point location can take a more conservative sampling along rays.

5 Conclusion

In this work, we presented a novel method for the visualization of time series volumetric particle datasets using a combination of attribute-aware radial basis functions, hardware ray tracing, and GPU-parallel tree construction. By integrating color-mapped attributes of the particles into the RBF field, we were able to increase the expressiveness of our data, and by incorporating blue noise, we were able to significantly improve comprehension of time-series exploration.

A potential limitation of our approach is that with many large and transparent particles, we face over-sampling issues due to excessive null collisions. Sampling empty regions with our method is cheap, but not free. We would likely benefit from tighter bounding majorants which would reduce the number of samples taken along a ray, especially in these empty regions. We also suspect that an alternative query formulation like a truncated K-Nearest-Neighbors query would help alleviate some issues where particle overlap is large.

Sample code and data demonstrating our method can be found online [24] at `https://github.com/gprt-org/attribute-aware-rbfs`, providing additional opportunities for insights as GPU architectures continue to evolve.

ACKNOWLEDGMENTS

The submitted manuscript was made in part by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. `https://energy.gov/downloads/doe-public-access-plan`.

This work was supported by the Department of Energy (DOE) under grant no. 001425594. This work was additionally supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant no. 456842964.

This work was also funded in part by NSF OAC award 2138811, the NSF CI CoE Award 2127548 DoE award DE-FE0031880, the Intel oneAPI Centers of Excellence at the University of Utah, the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the DoE and the NNSA, and the UT-Battelle LLC under contract DE-AC05-00OR22725.

REFERENCES

[1] M. Berzins, J. Beckvermit, T. Harman, A. Bezdjian, A. Humphrey, Q. Meng, J. Schmidt, and C. Wight. Extending the uintah framework through the petascale modeling of detonation in arrays of high explosive devices. *SIAM Journal on Scientific Computing*, 38(5):S101–S122, 2016. doi: 10.1137/15M1023270

[2] D. Cha, S. Son, and I. Ihm. GPU-assisted high quality particle rendering. In *Computer Graphics Forum*, vol. 28, pp. 1247–1255. Wiley Online Library, 2009. doi: 10.1111/j.1467-8659.2009.01502.x

[3] I. Evangelou, G. Papaioannou, K. Vardis, and A. Vasilakis. Fast radius search exploiting ray-tracing frameworks. *Journal of Computer Graphics Techniques Vol*, 10(1):25–48, 2021.

[4] R. Fraedrich, S. Auer, and R. Westermann. Efficient high-quality volume rendering of SPH data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1533–1540, 2010. doi: 10.1109/TVCG.2010.148

[5] R. Fraedrich, J. Schneider, and R. Westermann. Exploring the millennium run - scalable rendering of large-scale cosmological datasets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1251–1258, 2009. doi: 10.1109/TVCG.2009.142

[6] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181(3):375–389, 12 1977. doi: 10.1093/mnras/181.3.375

[7] P. Gralka, I. Wald, S. Geringer, G. Reina, and T. Ertl. Spatial partitioning strategies for memory-efficient ray tracing of particles. In *2020 IEEE 10th Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 42–52, 2020. doi: 10.1109/LDAV51489.2020.00012

[8] S. Green. Volumetric particle shadows, 2008.

[9] S. Gumhold. Splatting illuminated ellipsoids with depth correction. In *Proceedings of the Vision, Modeling, and Visualization Conference 2003 (VMV 2003)*, pp. 245–252, 2003.

[10] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Luki, S. Sehrish, and W. keng Liao. HACC: Simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy*, 42:49–65, 2016. doi: 10.1016/j.newast.2015.06.003

[11] M. Heinen and J. Vrabec. Evaporation sampled by stationary molecular dynamics simulation. *The Journal of Chemical Physics*, 151(4):044704, 2019. doi: 10.1063/1.5111759

[12] Y. Jang, R. Fuchs, B. Schindler, and R. Peikert. Volumetric evaluation of meshless data from smoothed particle hydrodynamics simulations. In R. Westermann and G. Kindlmann, eds., *IEEE/ EG Symposium on Volume Graphics*. The Eurographics Association, 2010. doi: 10.2312/VG/VG10/045-052

[13] T. Karras. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, EGGH-HPG'12, pp. 33–37. Eurographics Association, Goslar, Germany, 2012.

[14] A. Knoll, G. P. Johnson, and J. Meng. *Ray Tracing Gems II, Chapter 44: Path Tracing RBF Particle Volumes*, pp. 713–723. Apress, Berkeley, CA, 2021. doi: 10.1007/978-1-4842-7185-8_44

[15] A. Knoll, R. K. Morley, I. Wald, N. Leaf, and P. Messmer. *Ray Tracing Gems, Chapter 29: Efficient Particle Volume Splatting in a Ray Tracer*, pp. 533–541. Apress, Berkeley, CA, 2019. doi: 10.1007/978-1-4842-4427-2_29

[16] A. Knoll, I. Wald, P. Navratil, A. Bowen, K. Reda, M. E. Papka, and K. Gaither. RBF volume ray casting on multicore and manycore CPUs. *Computer Graphics Forum*, 33(3):71–80, 2014. doi: 10.1111/cgf.12363

[17] J. Kuhnert and S. Sudarshan. *Advances in PDE modeling and computation: Meshfree numerical schemes for time dependent problems in fluid and continuum mechanics*, pp. 119–136. Anne books New Delhi, 2014.

[18] P. Kutz, R. Habel, Y. K. Li, and J. Novák. Spectral and decomposition tracking for rendering heterogeneous volumes. *ACM Transactions on Graphics (TOG)*, 36(4):1–16, 2017. doi: 10.1145/3072959.3073665

[19] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH construction on GPUs. *Computer Graphics Forum*, 28(2):375–384, 2009. doi: 10.1111/j.1467-8659.2009.01377.x

[20] F. Lindemann and T. Ropinski. About the influence of illumination models on image comprehension in direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1922–1931, Dec 2011. doi: 10.1109/TVCG.2011.161

[21] L. B. Lucy. A numerical approach to testing the fission hypothesis. *Astronomical Journal*, 82(12):1013–1024, December 1977. doi: 10.1086/112164

[22] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, June 1995. doi: 10.1109/2945.468400

[23] Q. Meng, A. Humphrey, and M. Berzins. The Uintah framework: A unified heterogeneous task scheduling and runtime system. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pp. 2441–2448. IEEE, 2012. doi: 10.1109/SCC.2012.6674233

[24] N. Morrical. Attribute Aware RBFs Supplemental Code, 2023. doi: 10.5281/zenodo.8173156

[25] N. Morrical, A. Sahistan, U. Güdükbay, I. Wald, and V. Pascucci. Quick clusters: A GPU-parallel partitioning for efficient path tracing of unstructured volumetric grids. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):537–547, 2022. doi: 10.1109/TVCG.2022.3209418

[26] N. Morrical and P. Shriwise. General Purpose Raytracing Tookit, 2023. doi: 10.5281/zenodo.8019116

[27] N. Morrical, W. Usher, I. Wald, and V. Pascucci. Efficient space skipping and adaptive sampling of unstructured volumes using hardware accelerated ray tracing. In *Proceedings of IEEE Visualization*, VIS '19, pp. 256–260. IEEE, 2019. doi: 10.1109/VISUAL.2019.8933539

[28] N. Morrical, I. Wald, W. Usher, and V. Pascucci. Accelerating unstructured mesh point location with RT cores. *IEEE Transactions on Visualization and Computer Graphics*, 28(8):2852–2866, 2022. doi: 10.1109/TVCG.2020.3042930

[29] P. Navratil, J. Johnson, and V. Bromm. Visualization of cosmological particle-based datasets. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1712–1718, 2007. doi: 10.1109/TVCG.2007.70616

[30] J. Novák, I. Georgiev, J. Hanika, J. Kivánek, and W. Jarosz. Monte Carlo methods for physically based volume rendering. In *ACM SIGGRAPH Courses*, Aug. 2018. doi: 10/c5fj

[31] NVIDIA Corp. NVIDIA Ampere GA102 GPU Architecture: Second-Generation RTX. Technical report, NVIDIA, 2021.

[32] J. Orthmann, M. Keller, and A. Kolb. Topology-Caching for Dynamic Particle Volume Raycasting. In R. Koch, A. Kolb, and C. Rezk-Salama, eds., *Vision, Modeling, and Visualization (2010)*, pp. 147–154. The Eurographics Association, 2010. doi: 10.2312/PE/VMV/VMV10

[33] M. Piochowiak, T. Rapp, and C. Dachsbacher. Stochastic volume rendering of multi-phase SPH data. In *Computer Graphics Forum*, vol. 40, pp. 97–109. Wiley Online Library, 2021. doi: 10.1111/cgf.14121

[34] A. Preston, R. Ghods, J. Xie, F. Sauer, N. Leaf, K.-L. Ma, E. Rangel, E. Kovacs, K. Heitmann, and S. Habib. An integrated visualization system for interactive analysis of large, heterogeneous cosmology data. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 48–55. IEEE, 2016. doi: 10.1109/PACIFICVIS.2016.7465250

[35] G. Simon. Particle simulation using CUDA, 2020.

[36] S. Slattery, S. T. Reeve, C. Junghans, D. Lebrun-Grandié, R. Bird, G. Chen, S. Fogerty, Y. Qiu, S. Schulz, A. Scheinberg, A. Isner, K. Chong, S. Moore, T. Germann, J. Belak, and S. Mniszewski. Cabana: A Performance Portable Library for Particle-Based Simulations, Apr. 2022. doi: 10.5281/zenodo.6423410

[37] I. Wald, N. Morrical, and S. Zellmann. A memory efficient encoding for ray tracing large unstructured data. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):583–592, 2021. doi: 10.1109/TVCG.2021.3114869

[38] I. Wald, W. Usher, N. Morrical, L. Lediaev, and V. Pascucci. RTX beyond ray tracing: Exploring the use of hardware ray tracing cores for tet-mesh point location. In *High Performance Graphics (Short Papers)*, pp. 7–13, 2019. doi: 10.2312/hpg.20191189

[39] I. Wald, S. Zellmann, W. Usher, N. Morrical, U. Lang, and V. Pascucci. Ray tracing structured AMR data using ExaBricks. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):625–634, 2021. doi: 10.1109/TVCG.2020.3030470

[40] R. Winchenbach and A. Kolb. Multi-level memory structures for simulating and rendering smoothed particle hydrodynamics. In *Computer Graphics Forum*, vol. 39, pp. 527–541. Wiley Online Library, 2020. doi: 10.1111/cgf.14090

[41] A. Wolfe, N. Morrical, T. Akenine-Möller, and R. Ramamoorthi. Spatiotemporal Blue Noise Masks. In A. Ghosh and L.-Y. Wei, eds., *Eurographics Symposium on Rendering*. The Eurographics Association, 2022. doi: 10.2312/sr.20221161

[42] E. R. Woodcock, T. Murphy, P. J. Hemmings, and T. C. Longworth. Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry. In *Proceedings of the Conference on Applications of Computing Methods to Reactor Problems*, pp. 557–579. Argonne National Laboratory, 1965.

[43] I. Yu, A. Cox, M. H. Kim, T. Ritschel, T. Grosch, C. Dachsbacher, and J. Kautz. Perceptual influence of approximate visibility in indirect illumination. 6(4), oct 2009. doi: 10.1145/1609967.1609971

[44] S. Zellmann, D. Seifried, N. Morrical, I. Wald, W. Usher, J. A. Law-Smith, S. Walch-Gassner, and A. Hinkenjann. Point containment queries on ray-tracing cores for AMR flow visualization. *Computing in Science & Engineering*, 24(2):40–51, 2022. doi: 10.1109/MCSE.2022.3153677

[45] S. Zellmann, I. Wald, A. Sahistan, M. Hellmann, and W. Usher. Design and evaluation of a GPU streaming framework for visualizing time-varying AMR data. In *Bujack, Tierny et al.(Eds.): EGPGV 2022, 22nd Eurographics Symposium on Parallel Graphics and Visualization, Rome, Italy, June 13, 2022*, pp. 61–71. The Eurographics Association, 2022. doi: 10.2312/pgv.20221066

[46] S. Zellmann, M. Weier, and I. Wald. Accelerating force-directed graph drawing with RT cores. In *2020 IEEE Visualization Conference (VIS)*, pp. 96–100. IEEE, 2020. doi: 10.1109/VIS47514.2020.00026

[47] S. Zellmann, Q. Wu, A. Sahistan, K.-L. Ma, and I. Wald. Beyond ExaBricks: GPU volume path tracing of AMR data, 2022. doi: 10.48550/arXiv.2211.09997

[48] S. Zhao, Z. Lai, and J. Zhao. Leveraging ray tracing cores for particle-based simulations on GPUs. *International Journal for Numerical Methods in Engineering*, 124(3):696–713, 2023. doi: 10.1002/nme.7139