Abstract—Modern GPUs come with dedicated hardware to perform ray/triangle intersections and bounding volume hierarchy (BVH) traversal. While the primary use case for this hardware is photorealistic 3D computer graphics, with careful algorithm design scientists can also use this special-purpose hardware to accelerate general-purpose computations such as point containment queries. This article explains the principles behind these techniques and their application to vector field visualization of large simulation data using particle tracing.

RAY TRACING is a computer graphics algorithm that is based on geometric optics. While traditionally being focused on offline production rendering by the film industry, the recent addition
Department Head

of ray tracing cores, or RT cores, to graphics processing units (GPUs) has led to broader adoption of this technique for real-time applications. Ultimately, this newly gained popularity can be attributed to increased throughput in terms of the number of rays that can be traced through a given 3D geometry in a unit of time.

RT cores have the purpose of geometrically intersecting rays that are defined by their 3D origin \( o \in \mathbb{R}^3 \) and an (often unit) direction vector \( \vec{d} \in \mathbb{R}^3 \) with 3D objects that are usually given by a parametric equation, solving for the distance \( t \) of the point of intersection to the ray’s origin. Sometimes it is also necessary to restrict the objects tested to some interval \([t_{\text{min}}, t_{\text{max}}]\) along the ray, for example, when the ray is used to compute shadows where we are only interested in the objects between the origin and the light source. Therefore, most ray tracing APIs also store this interval along with the ray. Ray/object intersections are usually accelerated using search data structures, the most popular of which is arguably the bounding volume hierarchy (BVH). RT cores by NVIDIA, for example, have dedicated hardware for ray/triangle intersection and ray/BVH traversal.

In this regard, RT cores are essentially hardware units that accelerate tree traversal. Therefore, it is possible to use these units to accelerate general tree-traversal-based computations and not only graphics. The only requirement is that one can reformulate their algorithms to be mapped to the ray tracing hardware. For example, Zellmann et al. [1] used RT cores to accelerate force-directed graph drawing, where the efficacy of spring forces decreases with distance. The computations involved can be accelerated using a search tree over the graph’s vertex set. The vertices are interpreted as particles, and the forces are computed using radius point containment queries.

Reformulating one’s algorithm is, of course, not always possible or efficient. For example, the problem domain must be embeddable in \( \mathbb{R}^3 \), and in general, the problem must be mappable to a ray tracing problem without introducing significant overhead. In the previous example of the radius point queries [1], the authors observed that a fixed radius point query over a set of particles could be reformulated as a ray-tracing problem by using an inverse mapping. The authors promoted the set of particles to a set of (generally overlapping) spheres, and the sphere of interest inside which the neighboring particles are gathered becomes a ray with length zero.

Observing that the above motivating example of point queries over a set of particles draws its inspiration from physics—in fact, force-directed graph drawing can also be implemented as an n-body simulation [2]—it becomes evident that this overall principle might apply to all sorts of scientific fields such as fluid dynamics or flow visualization. The motivating example for this article is particle tracing in vector fields based on a particular grid representation—namely adaptive mesh refinement topologies that are common in astrophysical or meteorological simulation codes.

Point Containment Queries with NVIDIA RTX and OptiX

Point containment queries with RT cores have been proposed by several researchers, including the aforementioned force-directed graph drawing algorithm by Zellmann et al. [1]. Even more closely related to our approach is the work by Morrical et al. [3] who used point containment queries to evaluate the particle density at arbitrary sampling positions \( x \in \mathbb{R}^3 \) given an underlying unstructured grid representation with finite ele-
ments such as tetrahedra or hexahedra.

By building a BVH over the set of finite elements using one of the SDKs that support hardware ray tracing on NVIDIA GPUs (i.e., OptiX, DXR, or Vulkan), point containment queries can again be performed elegantly by just tracing a ray of length zero (see Fig. 1) through the hierarchy. For that, we just set \( t_{\text{min}}, t_{\text{max}} = 0 \) and the direction to some arbitrary non-zero-length value (for example \( \vec{d} = (1, 1, 1) \)). On NVIDIA GPUs that are labeled RTX, those point containment queries are hardware-accelerated.

With NVIDIA’s RTX extensions, the proper way to implement point containment queries is through user geometry representing the finite elements. We briefly review the steps involved in setting up this user geometry, and along the way also introduce the necessary terminology. The description and terminology we use roughly follows the OptiX API.\(^1\)

OptiX exposes entry points for the programmer to initiate and control ray generation and ray/object interactions. The ray generation program is comparable in nature to a compute kernel as it is executed by the threads of the programmable shading multiprocessors. In the case of a program that performs point containment queries, the ray generation program would be the place where the zero-length rays are generated at the appropriate positions. In the case of particle tracing, the ray generation program would hence also be used in the same way as one would use a compute kernel to implement particle tracing, by performing accesses to the computation domain through an OptiX BVH.

The point containment query rays of length zero are traced against this BVH, which is built using OptiX host API routines before the device program starts execution, and whose handle is then passed over to the ray generation program. OptiX supports both triangle BVHs, in which case the whole ensuing traversal routine is executed in hardware, or user geometry BVHs where the user specifies an intersection program that has access to the ray and that reports an intersection if the ray was found to intersect the user geometry. In the user geometry case, when the ray is traced, the hardware will perform many context switches between hardware BVH traversal and software intersection programs. Those context switches come at a performance penalty that gets worse the more often the context switches are necessary.

Point containment queries require the use of an intersection program and hence must be implemented using user geometry. The intersection program will report an intersection if the ray origin is contained inside the element. The list of elements that contain the point of interest can be stored and updated in a thread-local data structure (often referred to as the “per ray data”) and is available in both the closest hit program that is executed right after traversal finished, as well as in the ray generation program where execution is returned to afterwards.

Adaptive Mesh Refinement

Simulation codes compute and output quantities such as density, velocity or other scalar and vector fields for a three-dimensional domain that are assigned to data/sample points. The spatial arrangement of those data points defines the topology of the domain and is crucial to the computational and memory efficiency of the simulation code. The simplest topologies use structured grids with uniform cell sizes. Since memory bandwidth over the years has not grown at the same rate that transistor density has, more efficient topologies distribute the sample point budget to 3D regions where the entropy is relatively high. In particular, those topologies are usually either the already motivated, generally unstructured finite element meshes, or they fall into the category of adaptive mesh refinement (AMR) where the topology is locally structured and globally connected using a hierarchy. In practice, AMR topologies come in many forms such as Octree or block-structured AMR, where the major differences are in regards to branching factor and number of cells stored at different hierarchy levels.

A very common form is cell-centric structured AMR as for example produced by FLASH [5]. A challenge of cell-centric data is that the cell centers at level boundaries generally do not line up along the principle axes—this is called the T-junction problem (see Fig. 2)—and consequently, first order interpolation cannot easily be mapped to the customarily used tent reconstruction filter whose sample positions are aligned. For those

---

\(^1\)https://raytracing-docs.nvidia.com/optix7/index.html
Geometrical setup for reconstruction via basis functions as proposed in related work by Wald et al. [4]. (a) Cell-centric AMR data set with boundary cells of neighboring levels. This presents us with the T-junction problem. (b) Reconstruction of cell-centric data would locally be performed on the dual grids; neighboring dual grid cells form the domain (dotted squares) of the original grid cells that they overlap. (c) The domains that overlap the sampling position determine the cells that contribute to the linear combination from Eq. (1).

Figure 2.

Real-Time Reconstruction

Recent work on real-time AMR visualization has focused on high-quality reconstruction with cell-centric data and at level boundaries. For an extended discussion we refer the reader to the paper by Wald et al. [6]. Generally, if the topology used is a structured grid, one can just use the dual grid to perform reconstruction. In contrast to that, with AMR data the dual grid cells generally do overlap, in which case multiple cells affect the sample value at the sampling position (Fig. 2b).

The basis function reconstruction method by Wald et al. [4] reconstructs the value using a weighted linear combination:

\[
B(p) = \frac{\sum_{i} \hat{H}_{C_i}(p)C_{v_i}}{\sum_{i} \hat{H}_{C_i}(p)}.
\]  

(1)

\(\hat{H}_{C_i}\) is a tent-shaped basis function:

\[
\hat{H}_{C_i}(p) = \prod_{d \in x,y,z} \max \left(1 - \left( \frac{|C_{wd} - pd|}{C_w} \right), 0 \right)
\]  

where \(C_w \in \mathbb{R}\) and \(C_p \in \mathbb{R}^3\) denote the size and position of cell \(C_i\) and \(p \in \mathbb{R}^3\) refers to the sample point. Locally, using tent basis functions results in linear interpolation. But as the cell sizes at level boundaries differ, so do the extents of the tent shapes. To correct for coarser cells being over-represented, the linear combination is hence weighted by the cumulated tent basis functions.

ExaBricks Data Structure

We base our particle tracing code off of the ExaBricks data structure and visualization software [6], which is optimized for interactive rendering of AMR data sets on GPUs equipped with ray tracing hardware. ExaBricks added to the state of the art by enabling reconstruction with a first order interpolant and adaptive sampling for ray marching-style algorithms, all without the need to perform per sample cell location via costly tree traversal. When building the data structure that allows for fast AMR cell location on GPUs, the ExaBricks software first drops the original AMR hierarchy that is generated by the simulation code, then builds a spatial subdivision of same-level cells to obtain bricks. At this point, where traditional AMR visualization systems would just render the bricks and composite the intermediate results, the ExaBricks software constructs what the authors call the active brick regions (Fig. 3).

The active brick regions are obtained by first computing all the bricks’ domains (i.e., the region of space where at least one cell in a brick has non-zero basis function weight (denominator term in. Eq. (1))). The brick domains are computed by extending the brick boundaries by an amount large enough to accommodate the interpolant that is later used for reconstruction. The ExaBricks data structure uses the basis function reconstruction method described above and hence requires domains whose bounds extend the brick bounds by half a cell in each direction. The size of these padding regions varies widely across different bricks as it depends on the AMR level of the cells contained inside. Consequently, domains of different bricks can overlap by almost arbitrary amounts (Fig. 3c), and any point in space could lie within an almost arbitrary number of domains of possibly different-level bricks.

Since any brick whose domain a point lies in will influence that point’s basis function evaluation, evaluating at a given point requires finding and iterating over all the bricks whose domain the point overlaps. To quickly find all such bricks for...
Figure 3. ExaBricks data structure proposed in prior work [6] that our method builds upon. (a) Exemplary AMR data set comprised of three levels and six subgrids. (b) We first combine same-level subgrids to single grids (bricks) using a spatial hierarchy builder. (c) For correct interpolation at level boundaries, we do not only have to integrate over the bricks themselves, but over the bricks’ domains which in general overlap. (d) The regions where the domains overlap form a space decomposition of generally concave shapes. Those shapes we call the active brick regions. Each such region stores a list of pointers to its respective bricks. (e) We decompose the active brick regions into convex blocks using yet another spatial decomposition, an example is shown in the illustration. We build an OptiX BVH over those blocks. The BVH can be used to perform ray marching with adaptive sampling and space skipping, or point containment queries as proposed in this paper.

A given point, Wald et al. [6] proposed to use the aforementioned active brick regions, which form a decomposition of space into regions where all points from a given region overlap the same domains. Given the near-arbitrary overlap of the different domains, the resulting same-domain regions are not generally rectangular any more; but with each domain being rectilinear they are also not arbitrarily shaped either, generally forming what are the 3D equivalents of L-shapes, T-shapes, etc. (see, e.g., the yellow box in Fig. 3d).

Given these spatial regions, Wald et al. [6] further subdivided these L- and T-like shapes into rectilinear 3D boxes (using a spatial kd-tree like subdivision), and for each such box stored a list containing the references to all the bricks whose domains are active for the space covered by this box; as well as the level of the finest cells contained by either of those bricks. The level will later be used to determine an appropriate step size for adaptive sampling.

The resulting rectilinear boxes of this ExaBricks data structure are then easily amenable to be rendered on a GPU with OptiX—the boxes induced by the active brick regions can directly be used as user-defined primitives in a user geometry BVH (Fig. 3e), and since these boxes do not overlap, any point in 3D space will only ever be contained in exactly one such primitive.

ExaBricks uses a ray generation program to implement both a volumetric and an implicit iso-surface ray caster. The ray casters step from brick region to brick region by using (regular) OptiX intersections with primary rays; when processing an active brick region, the interval \([t_{\text{near}}, t_{\text{far}}]\) resulting from those intersections is integrated over using adaptive sampling based on the finest cell size of the bricks the active brick region is derived from.

During ray marching, the basis function reconstruction method is used to reconstruct sample values (the choice of interpolant is however not restricted to the basis method, other interpolants are supported but potentially require larger domains and thus different configurations of active brick regions). In contrast to Wald et al. [4], ExaBricks does not need to perform cell location per sample position using kd-tree traversal. Instead, queries can just refer to the information that is stored by the current brick region. In particular, the brick IDs stored by each active brick region allow the software to perform sample location by just iterating over the regions’ bricks, and then locating the sample value using a simple offset calculation. The resulting memory access patterns are particularly well-suited for GPUs.

Particle Tracing Using RT Core Point Containment Queries

In the following we describe how we extend the ExaBricks framework to compute and later render streamlines from vector fields using a particle tracer. We accomplish this by combining the various methods and data structures de-
scribed above. More important than the individual techniques is how we the integrate them in the presence of a ray tracing framework for scientific visualization.

**Integration Methods**

Approximation of integrals of ordinary differential equations (ODEs) can be achieved by Euler’s method or more sophisticated methods, like Runge-Kutta integration. Euler’s method estimates next particle positions by adding a scaled tangent vector to the current position (starting at the seed position), resulting in a new position and repeating this procedure until a termination criterion is reached. While this is perfectly fine for linear functions, for more complex functions this leads to errors, depending on the vector length (step size). A better approach to approximately solving ODEs is the class of methods introduced by Runge and Kutta. The basic idea of these methods is to determine the step direction not only by the tangent at the current position but taking a weighted average of the value of tangents at midpoints. When averaging over four slopes, this results in the well known fourth order Runge-Kutta method. See the technical report by Ken Joy [7] for a derivation of those standard integration methods.

**Seed Points and Buffer Allocation**

It is the user’s responsibility to generate the seed points to initialize the particle tracer, and to make sure that they fall within the bounds of the vector fields. Before the tracer is initiated, we also require the user to set the maximum number of timesteps.

With that information, we initialize a buffer of size \#seeds × \#steps in GPU memory that can hold all the sample positions and that is available to the OptiX ray tracing pipeline. While this may seem excessive at first glance, we point out that the typical objective with particle tracers is to generate relatively few but informative streamlines, as this technique is generally prone to visual clutter.

**Progressive Particle Tracing**

Now that the seed points are available on the device, there are several different ways that we could initialize our particle tracer; a natural choice would be to initiate the tracer from within a separate ray generation program than the one that executes the other visualization algorithms, and asynchronously overlap the execution of the two ray generation programs. In practice however, we found it more convenient to initialize and subsequently update the particle tracer from within the same ray generation program that executes the other visualization algorithms. As one usually uses a relatively small number of seeds—using hundreds or even thousands of seeds usually leads to severe visual clutter—it is feasible to advance the particles by a few time steps in parallel without the visualization becoming unresponsive.

We thus decided to use a progressive particle tracing approach where, before we render a frame, we advance the tracer by a constant and small number of timesteps, devoting one GPU thread to each particle. After that, the ray generation program will perform volume and surface rendering—the latter step will also render the traces that were computed so far (but excluding the timesteps that were generated during this ray generation launch, as those are not present in the surface BVH yet).

**Particle Advection**

We use the ExaBricks data structure to realize parallel particle tracing for vector field visualization and implement that using RTX point containment queries. Using ExaBricks and its
spatial decomposition into active brick regions has the major advantage that we can make use of hardware-accelerated cell location and that the regions enable adaptive sampling.

Adaptive sampling is guided by the finest-level cell sizes stored with the brick regions. The adaptive sampling process is illustrated in Fig. 4. A Runge-Kutta integration step is presented in Fig. 5.

To initiate the integration step, each GPU thread is responsible for one particle and will first retrieve its previous sample position (or, if t=0, its seed point) from the device buffer (code handling particles whose traces terminated prematurely is omitted here for simplicity).

At the core of the algorithm is the function sampleDirection() that will construct a direction vector using three point containment queries at the 3D sample position p into the (user-configurable) X, Y, Z channels associated with the vector field (e.g., the three vector components of the velocity or vorticity field). The sampleDirection() function also returns the finest cell width in the sampled region using a modifiable C++ reference.

The sampleDirection() function performs basis method reconstruction [4] by tracing one ray with length zero into the region BVH. Since region boxes do not overlap, the first primitive that contains the ray origin is exactly the box that lists all bricks that influence this sample point. Thus, all the intersection program has to do is check if the ray origin is inside the active brick region’s bounds (it usually will be, but there is no guarantee that the BVH traversal might not call a false positive—so needs to be checked); then upon the first intersection store the pointer to the brick list in the per-ray-data for the sampleDirection() to iterate over, and terminate the ray. The sampleDirection() function then iterates over the bricks in the list and computes the basis function values and weights for the X, Y, Z channels of the vector field. The reconstructed 3D vector serves as one of the four direction vectors that are required by the fourth order Runge-Kutta integration method (v1 through v4 in the code listing above).

The four Runge-Kutta direction vectors returned by sampleDirection() are individually scaled not only by the constant and user-configurable step length, but also by the finest cell width in the brick region. That way we adaptively sample the vector field. Finally, the newly computed particle position is written back into the sample positions buffer, at the position that is reserved for the current timestep.

In contrast to volume ray marching, with the particle tracer it is not always possible to choose the best step size for adaptive sampling, as can be seen in Fig. 4b. Volume rays are first clipped against the active brick region’s boundary, so that we know the exact length of the ray segment covered by the region. With particle tracing, when we choose an adaptive sampling step size, we cannot easily determine a priori if the curve will bend into a brick region with finer cell sizes and thus potentially undersample the volume at level boundaries. In practice, and with the data sets we used for our tests, we did not find this to cause any noticeable issues, but still want to point out this potential source of undersampling inherent to the approach.

High-Quality Rendering with Curved Geometries

At this point, when all the timesteps were computed, we are finished with the actual post-processing and particle advection procedure. With
the sample positions buffer filled, we can now proceed to render the traces, for example as streamlines.

How this is done is largely independent of the actual particle advection step. As we are using OptiX 7, one option would be to use the integrated curve primitives to visualize the streamlines. For technical reasons, our system however uses the tube primitives presented by Wald et al. [8] and just renders the control points as rounded cylinder strips (Fig. 6). The BVH for the geometry is rebuilt whenever the tracer, in a previous ray tracing pipeline launch, generated new sample points.

Integration with a ray tracing-based pipeline of course also enables high-quality rendering and shading techniques (also Fig. 6). ExaBricks implements a simple local shading model that is augmented with ray-traced ambient occlusion. It would however be straightforward to also support more advanced rendering algorithms. Optionally, we can also map an AMR field (e.g., velocity magnitude) as a color onto the streamlines.

**Example Applications from Astrophysics**

We present two user case studies where our system was used for post-processing followed by visual exploration to gain insights into the astrophysical data sets. The two data sets represent phenomena at different scale and are both time-varying. They illustrate the relevance of AMR vector field exploration using high-quality rendering for this particular scientific field.

**Example Application 1: Magnetic Fields in Star-Forming Molecular Clouds**

One key research question in modern astrophysics is how stars in our Galaxy (the Milky Way) are forming. Broadly speaking, stars condense out from the diffuse gas sitting between the already existing stars in our Galaxy. Due to the effect of gravity, large (several ten light-years in diameter) accumulations of this gas form so-called molecular clouds, which represent the nurseries of forming stars. As time evolves, parts of these molecular clouds become denser and denser due to the effect of gravity, ultimately leading to the formation of a new generation of stars.

Hence, understanding these properties is crucial for understanding the formation of individual stars. From observations it is known that molecular clouds have a highly complex and possibly hierarchical structure [9]. In addition, many physical processes like turbulence, gravity, radiation and magnetic fields influence the evolution of molecular clouds in a complex interplay. This in turn necessitates dedicated 3D simulations to model the formation and evolution of molecular clouds—and their embedded stars.

In the SILCC-Zoom project [10] the evolution of molecular clouds, which are embedded in a part of a spiral galaxy (in turn modelled in the larger-scale SILCC project, [11]), is modelled by means of state-of-the-art, 3D, magneto-hydrodynamical simulations. For this purpose, the versatile astrophysical AMR code FLASH is used [5], which was modified to include the various physical processes mentioned before to allow for one of the most realistic molecular cloud simulations to date.

In Fig. 7 we visualize one snapshot of the simulation, gradually zooming in from the largest onto the smallest scales. This particular snapshot consists of 72.8 M cells and 142 K AMR subgrids that are distributed across seven hierarchy levels. The logical grid—an imaginary structured grid that, if the simulation was resampled on it, would retain the detail of even the finest AMR level—has a size of $4096 \times 4096 \times 81920$ cells. This unveils an intrinsic complication of astrophysics that the characteristic length scales extend over a significant range often covering *four orders of magnitude and more*, which explicitly re-
Figure 7. Graphical representation of the SILCC-Zoom simulation zooming in from the galactic scale (a, b) onto the molecular cloud (c) and its embedded filamentary substructure (yellow gas in panels c and d). Panel (d) shows the complex magnetic field structure associated with the molecular cloud, and reveals the dependencies between the filaments and the magnetic flow.

quires the usage of AMR. In the SILCC-Zoom simulation, the represented scales extend from galactic scales ($\geq 1$ kpc, Fig. 7a and Fig. 7b), over molecular clouds (1 - 100 pc, Fig. 7c and Fig. 7d) down to their complex substructure (0.1 - 1 pc, yellow material in Fig. 7c and Fig. 7d). This substructure consists of so-called *filaments*, i.e. elongated dense structures embedded in gas of lower density, which arrange in a complex network. Filaments are very thin compared to the entire molecular cloud, i.e. have diameters of less than one lightyear, corresponding to 1% of the cloud’s diameter. Hence, to resolve this filamentary structure, high resolution facilitated by AMR is required.

Despite their small size, filaments are key to understanding the star formation process: They present the densest structures of the molecular cloud and thus stars will form preferentially inside them.

One way to assess the importance of magnetic fields is their orientation relative to the dense gas, i.e. the filaments. Observational and theoretical studies predict that magnetic fields pierce perpendicularly through the center of filaments and can be dragged along with the filament. As an analogy one could imagine a pearl on a rubber band: pulling the pearl to the side will bend the rubber band, but it will still pierce through the pearl’s center.

Gaseous phenomena that can be expressed as scalars, like the density field, can be effectively rendered using the typical volume and isosurface rendering modalities. In Fig. 7c the filaments for example are the structures that are assigned the color yellow and can be easily differentiated from the surrounding gas via an RGBA transfer function. For 3D vector fields such assignments are, however, often ineffective: the vector components can easily be mapped to RGB components and rendered as a volume, but such a visualization will not convey directionality. Using a vector field visualization like the one proposed, with high-quality ray tracing and ambient occlusion, the spatial relationship is however revealed. As can be nicely seen in Fig. 7d, the fieldlines associated with the magnetic field pierce through and bend around the dense, gaseous filaments.

A challenge unrelated to the point containment query technique, but inherent to vector field visualization, that we encountered, was the occurrence of visual clutter when not carefully choosing seed points. We therefore, in a pre-process, isolated the filaments by identifying the AMR cells with the highest density, and placed random seeds by choosing from the set of cells that belong to the filaments. It is noteworthy here that we found the integration of the technique into a ray tracing framework beneficial as we can easily make use of techniques such as ambient occlusion, which can help to reduce visual clutter and provide additional depth cues for the visualization.
Example Application 2: Binary Neutron Star Formation via Common-envelope Ejection

In a landmark discovery, astronomers have observed the merger of two neutron stars in both gravitational waves and electromagnetic radiation (from radio to gamma-rays). This discovery has opened up new lines of research in several areas of physics and astrophysics, from understanding gravity in the strong-field regime to understanding nucleosynthesis in the universe.

However, we do not understand how these systems are formed. In order to merge via emission of gravitational radiation within the age of the universe, two neutron stars must have a relatively small orbital separation, much smaller than the initial separations or even radii of their progenitor stars (the red giants that exploded in supernovae and left neutron stars behind). If the neutron stars started out at the orbital separations of their progenitor stars, they would not merge within the age of the universe.

One proposed solution to this problem is that the neutron stars tighten their orbital separation during the poorly understood “common-envelope” phase. During this phase, one star (say, that has already evolved to a neutron star) may interact with the envelope of another star (a red giant) and lose orbital energy, causing it to inspiral toward the center of the other star. This process will not stop, and the neutron star will merge with the core of the red giant, unless the neutron star is able to eject the envelope of the red giant and “park” its orbit in order to reach a final orbital separation around the core of the red giant. The remaining core of the red giant will then explode and form a second neutron star. The two neutron stars will then lose energy to gravitational waves, causing their orbit to shrink over time. If they start close enough, they will merge within the age of the universe and be observable in both gravitational and electromagnetic radiation, a so-called “multi-messenger” event.

Performance Measurements

We present results of a comparative performance evaluation in Fig. 9. There we compare against the kd-tree method by Wald et al. [4]. We however built the kd-tree over the brick set induced by the ExaBricks data structure [6]. With particle advection, and as opposed to, e.g., direct volume rendering, we cannot make use of the optimization where the tree is only traversed once per active brick region. The performance study is hence a direct comparison of software kd-tree traversal versus hardware-accelerated OptiX BVH traversal, as the samples taken will be exactly the same. For our comparison, we advect a varying number of particles and perform 100 K
Figure 8. Binary neutron star formation via common-envelope ejection. 3D rendering of the ratio of velocity magnitude to local escape velocity with streamlines as a neutron star inspirals through the envelope of a gas giant. Time evolves from left to right. Bottom panels are same time steps as top panels, but zoomed in. Gas is ejected from the initially gravitationally-bound envelope of the gas giant, allowing the neutron star to reach a final orbital separation before the gas giant evolves to a neutron star. The two neutron stars will then merge via emission of gravitational radiation (gravitational waves).

Discussion and Conclusion

We presented an unconventional yet effective technique to perform point containment queries into AMR vector fields. The technique is quite general and has previously also been used by other authors, e.g., to accelerate point containment queries in finite element data sets and other compute-intensive applications. We adapted the technique to AMR vector field visualization but note that the overall idea of mapping 3D tree traversal to a ray tracing problem and by that enabling the use of RT cores is generally applicable to all sorts of problems. Using ray tracing in this holistic way also potentially makes integration of high-quality rendering easier than with typical scientific visualization systems that use rasterization. Conversely, as such a system usually maintains its 3D geometry in a BVH anyway, using that for point containment queries is a convenient choice with very little overhead.

The major advantage of using RTX point containment queries, in our case, is that we can leverage adaptive sampling when extracting traces from the vector field, while other methods, such as direct volume rendering, will potentially benefit more if execution of multiple threads stays within the same active brick region for some time. Leveraging this effect with our approach is
Department Head

not possible. In the future, we however plan to investigate if an approach that is not based on ray tracing but instead parallelizes the particle advection step over the set of active brick regions can result in increased performance. Similar approaches based on macrocells from structured grids however often suffer from scalability issues, and the adjacency information that is required to advect the particles from one region to another is non-trivial to maintain, compared to the macrocell connectivity of a grid accelerator built on top of a uniform grid topology.

Future work will also include adding support for path lines and unsteady flow—the examples we presented were based on streamlines and fieldlines extracted from single animation frames. A major challenge is that AMR topologies often change over time and hence the ExaBricks data structure needs expensive rebuilds, so that addressing this problem would also require to address interactive data structure construction.

We argue that the approach of using RTX ray tracing cores for point containment queries is so general that it can be used for all sorts of different applications, ranging from physics like ours to engineering. Such applications are not limited to structured or semi-structured grids, but also to generally unstructured, finite elements. The method could also potentially be extended to support multiple vector fields. In this respect, our paper is also meant to serve as an example and potentially a guide for practitioners to implement this technique in their own frameworks.

Acknowledgments

SZ acknowledges funding by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—grant no. 456842964. SZ, DS, SW, and AH wish to thank the Ministry of Culture and Science of the State of North Rhine-Westphalia for supporting the work through the PROFILBILDUNG grant PROFILNRW-2020-038C. DS and SW acknowledge funding by the Deutsche Forschungsgemeinschaft (DFG) via the Collaborative Research Center SFB 956 “Conditions and Impact of Star Formation” (subprojects C5 and C6). SW further acknowledges funding from the European Research Council via ERC Starting Grant no. 679852 (RADFEEDBACK). J.L.-S. acknowledges use of the lux supercomputer at UCSC, funded by NSF MRI grant AST 1828315, and the HPC facility at the University of Copenhagen, funded by a grant from VILLUM FONDEN (project number 16599). The SILCC-Zoom simulation data used in this work is based on the FLASH code, which was in part developed by the DOE NNSA-ASC OASCR Flash Center at the University of Chicago. The simulations have been run on Supermuc at the Leibniz-Rechenzentrum Garching using supercomputing time acquired via the Gauss Centre for Supercomputing.

REFERENCES

1. S. Zellmann, M. Weier, and I. Wald, “Accelerating force-directed graph drawing with RT cores,” in 2020 IEEE Visualization Conference (VIS), 2020, pp. 96–100.
2. P. Eades, “A heuristic for graph drawing,” Congressus Numerantium, vol. 42, pp. 149–160, 1984.
3. N. Morrical, W. Usher, I. Wald, and V. Pascucci, “Efficient space skipping and adaptive sampling of unstructured volumes using hardware accelerated ray tracing,” in 2019 IEEE Visualization Conference (VIS), Oct 2019, pp. 256–260.
4. I. Wald, C. Brownlee, W. Usher, and A. Knoll, “CPU Volume Rendering of Adaptive Mesh Refinement Data,” in SIGGRAPH Asia 2017 Symposium on Visualization, 2017.
5. A. Dubey, R. Fisher, C. Graziani, I. Jordan, G. C., D. Q. Lamb, L. B. Reid, P. Rich, D. Sheeler, D. Townsley, and K. Weide, “Challenges of Extreme Computing using the FLASH code,” in Numerical Modeling of Space Plasma Flows, ser. Astronomical Society of the Pacific Conference Series, N. V. Pogorelov, E. Audit, and G. P. Zank, Eds., vol. 385, Apr. 2008, p. 145.
6. 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, 2020.
7. K. Joy, “Numerical methods for particle tracking in vector fields,” February 1999, visualization notes, Visualization Research Laboratory, University of California, Davis. [Online]. Available: https://web.cs.ucdavis.edu/~ma/ECS177/papers/particle_tracing.pdf
8. I. Wald, N. Morrical, S. Zellmann, L. Ma, W. Usher, T. Huang, and V. Pascucci, “Using hardware ray transforms to accelerate ray/primitive intersections for long, thin primitive types,” Proc. ACM Comput. Graph. Interact. Tech., vol. 3, no. 2, Aug. 2020. [Online]. Available: https://doi.org/10.1145/3406179
Stefan Zellmann received the graduate degree in information systems from the University of Cologne and the doctor’s degree in computer science in 2014, with a PhD thesis on high performance computing and direct volume rendering. His current affiliation is with the Institute of Visual Computing at the Bonn-Rhein-Sieg University of Applied Sciences. He was with the Chair of Computer Science, University of Cologne from 2009 to 2021, where he still serves as a lecturer. His research focuses on the interface between high performance computing and real-time rendering. There he is primarily concerned with algorithms and software abstractions to leverage real-time ray tracing and photorealistic rendering at scale and for large 3D models and scientific data sets. Contact him at stefan.zellmann@h-brs.de

Daniel Seifried is a PostDoc in the field of astrophysics at the I. Physical Institute at the University of Cologne, Germany. His research interests include 3D, magneto-hydrodynamical simulations of star formation. Contact him at seifried@ph1.uni-koeln.de

Nate Morrical is a PhD student at the University of Utah, an intern alumni from Nvidia and Pixar, and is currently working under Valerio Pascucci as a member of the CEDMAV group in the Scientific Computing and Imaging Institute (SCI). His research interests include high performance GPU computing, real time ray tracing, and human computer interaction. Prior to joining SCI, Nate received his B.S. in Computer Science from Idaho State University, where he researched interactive computer graphics and computational geometry under Dr. John Edwards. Email: natemorrical@gmail.com

Ingo Wald is a director of ray tracing at NVIDIA. He received his master’s degree from Kaiserslautern University and his PhD from Saarland University, then served as a postdoctorate at the Max-Planck Institute Saarbrücken, as a research professor at the University of Utah, and as technical lead for Intel’s software-defined rendering activities (in particular, Embree and OSPRay). Ingo has coauthored more than 90 papers, multiple patents, and several widely used software projects around ray tracing. Email: iwald@nvidia.com

Will Usher is a Ray Tracing Software Engineer at Intel, working on the oneAPI Rendering Toolkit. His work focuses on CPU and GPU ray tracing, distributed rendering, and scientific visualization. He completed his Ph.D. in Computer Science in 2021 on data management and visualization for massive scientific data sets at the Scientific Computing and Imaging Institute at the University of Utah, advised by Valerio Pascucci. Email: will.usher@intel.com

Jamie A.P. Law-Smith is a fellow at the Institute for Theory and Computation at the Center for Astrophysics | Harvard & Smithsonian. His research is in theoretical astrophysics and physics; some interests include tidal disruptions of stars by black holes, common envelope evolution in the context of gravitational-wave sources, and de Sitter space in theories of quantum gravity. He received his Ph.D. in Astronomy and Astrophysics from the University of California Santa Cruz. Email: jamie.law-smith@cfa.harvard.edu

Stefanie Walch-Gassner is a Full Professor of computational geometry under Dr. John Edwards. Email: walch@ph1.uni-koeln.de

André Hinkenjann is a Full Professor of com-
Department Head

puter graphics and interactive environments with the Department of Computer Science, Bonn-Rhein-Sieg University of Applied Sciences (BRSU), Sankt Augustin, Germany, and an Adjunct Professor with the Department of Computer Science, University of New Brunswick, Fredericton, NB, Canada. He is the Founding Director of the Institute of Visual Computing, BRSU. His research interests include high-quality rendering, interactive data exploration, and high-resolution display systems., Prof. Hinkenjann is a member of ACM SIGGRAPH, Eurographics, IEEE and German GI. He is a regular reviewer for international conferences and journals on the above mentioned topics. Email: andre.hinkenjann@h-brs.de