

# Point Containment Queries on Ray-Tracing Cores for AMR Flow Visualization

Stefan Zellmann , University of Cologne and Bonn-Rhein-Sieg University of Applied Sciences, 53757, Sankt Augustin, Germany

Daniel Seifried , University of Cologne, 50937, Cologne, Germany

Nate Morrical , NVIDIA & SCI Institute, Salt Lake City, UT, 84112, USA

Ingo Wald , NVIDIA, Santa Clara, CA, 95951, USA

Will Usher , Intel, Santa Clara, CA, 95054, USA

Jamie A. P. Law-Smith , Center for Astrophysics, Cambridge, MA, 02138, USA

Stefanie Walch-Gassner , University of Cologne, 50937, Cologne, Germany

André Hinkenjann , Bonn-Rhein-Sieg University of Applied Sciences, 53757, Sankt Augustin, Germany

*Modern graphics processing units come with dedicated hardware to perform ray/triangle intersections and bounding volume hierarchy traversal. While the primary use case for this hardware is photorealistic 3-D 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 (RT) 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 of RT cores, to graphics processing units (GPUs) has led to a 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 3-D geometry in a unit of time.

RT cores have the purpose of geometrically intersecting rays that are defined by their 3-D origin  $o \in \mathbb{R}^3$  and an (often unit) direction vector  $\vec{d} \in \mathbb{R}^3$  with 3-D 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  $[t_{\min}, t_{\max}]$  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 RT application programming interfaces (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 RT hardware. For example, Zellmann *et al.*<sup>1</sup> 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.

---

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see <https://creativecommons.org/licenses/by/4.0/>  
Digital Object Identifier 10.1109/MCSE.2022.3153677  
Date of publication 23 February 2022; date of current version 27 May 2022.

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 an RT problem without introducing significant overhead. In the previous example of the radius point queries, Zellmann *et al.*<sup>1</sup> observed that a fixed radius point query over a set of particles could be reformulated as an RT 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.

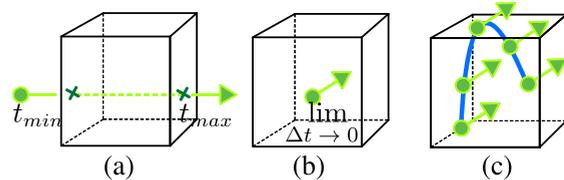
Observing that the abovementioned motivating example of point queries over a set of particles draws its inspiration from physics—in fact, force-directed graph drawing can also be implemented as an  $n$ -body simulation<sup>2</sup>—it becomes evident that this overall principle might apply to all sorts 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 (AMR) 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.*<sup>1</sup> Even more closely related to our approach is the work by Morrical *et al.*<sup>3</sup> who used point containment queries to evaluate the particle density at arbitrary sampling positions  $\mathbf{x} \in \mathbb{R}^3$  given an underlying unstructured grid representation with finite elements, such as tetrahedra or hexahedra.

By building a BVH over the set of finite elements using one of the SDKs that support hardware RT 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 Figure 1) through the hierarchy. For that, we just set  $t_{\min}, t_{\max} = 0$  and the direction to some arbitrary nonzero-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



**FIGURE 1.** Ray-traced point containment queries. For simplicity, the integration domain here is deliberately just a single box, but would typically be comprised of multiple overlapping finite elements or grid cells. (a) Ordinary RT only allows us to compute the intersections with the elements' *boundaries* in the interval of length  $\Delta t = t_{\max} - t_{\min}$ . (b) If the length of that interval is zero, the ray becomes a single point, so we can find the *overlapping* elements at the ray's origin position. (c) This principle can be used to perform general point containment queries to, for example, compute streamlines if the integration domain represents a vector field.

terminology. The description and terminology we use roughly follows the OptiX API.<sup>a</sup>

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

<sup>a</sup><https://raytracing-docs.nvidia.com/optix7/index.html>

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 is finished, as well as in the ray generation program where execution is returned to afterward.

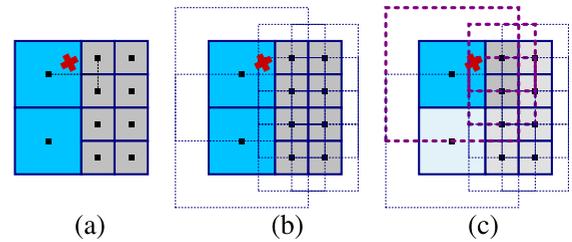
### ADAPTIVE MESH REFINEMENT

Simulation codes compute and output quantities, such as density, velocity, or other scalar and vector fields for a 3-D 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 3-D 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 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.<sup>5</sup> 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 Figure 2)—and consequently, first-order interpolation cannot easily be mapped to the customarily used tent reconstruction filter whose sample positions are aligned. For those reasons, GPU-based AMR rendering codes either concentrated on vertex-centric data only, or did not perform interpolation at all if the data was cell-centric.

### 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



**FIGURE 2.** Geometrical setup for reconstruction via basis functions, as proposed in related work by Wald *et al.*<sup>4</sup> (a) Cell-centric AMR dataset 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) Domains that overlap the sampling position determine the cells that contribute to the linear combination from (1).

refer the reader to the paper by Wald *et al.*<sup>6</sup> 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 [see Figure 2(b)].

The basis function reconstruction method by Wald *et al.*<sup>4</sup> reconstructs the value using a weighted linear combination

$$B(p) = \frac{\sum_{C_i} \hat{H}_{C_i}(p) C_{v_i}}{\sum_{C_i} \hat{H}_{C_i}(p)} \quad (1)$$

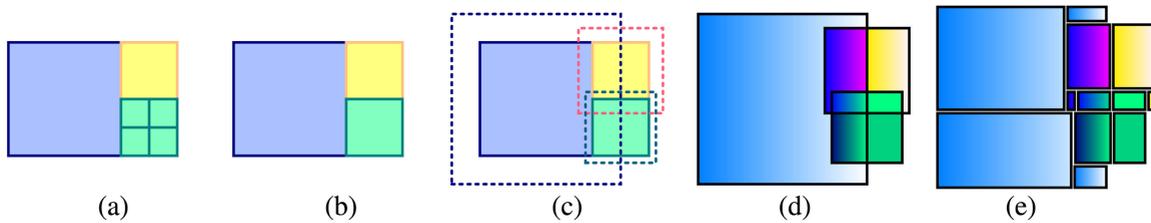
where  $\hat{H}_{C_i}$  is a tent-shaped basis function

$$\hat{H}_C(p) = \prod_{d \in \{x, y, z\}} \max\left(1 - \left(\frac{|C_{pd} - p_d|}{C_w}\right), 0\right) \quad (2)$$

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,<sup>6</sup> which is optimized for interactive rendering of AMR datasets on GPUs equipped with RT hardware. ExaBricks added to the state of the art by enabling reconstruction with a first-order interpolant and adaptive sampling for ray



**FIGURE 3.** ExaBricks data structure proposed in prior work<sup>6</sup> that our method builds upon. (a) Exemplary AMR dataset 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) 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 article.

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* (see Figure 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 nonzero basis function weight (denominator term in (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 previously 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 [see Figure 3(c)], 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 a given point, Wald *et al.*<sup>6</sup> 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 3-D equivalents of L-shapes, T-shapes, etc. [see, e.g., the yellow box in Figure 3(d)].

Given these spatial regions, Wald *et al.*<sup>6</sup> further subdivided these L- and T-like shapes into rectilinear 3-D 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 [see Figure 3(e)], and since these boxes do not overlap, any point in 3-D 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 isosurface 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 that 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.*,<sup>4</sup> 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 described earlier. More important than the individual techniques is how we integrate them in the presence of an RT framework for scientific visualization.

### Integration Methods

Approximation of integrals of ordinary differential equations (ODEs) can be achieved by the Euler's method or more sophisticated methods, like Runge–Kutta integration. The 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<sup>7</sup> 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  $\#seeds \times \#steps$  in GPU memory that can hold all the sample positions and that is available to the OptiX RT 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.

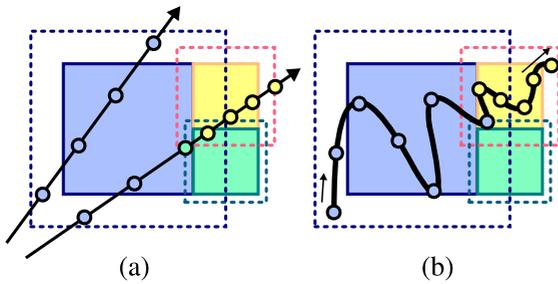
### Progressive Particle Tracing

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.



**FIGURE 4.** Adaptive sampling (a) with a volume ray marcher and (b) with a particle tracer. As the volume rays are *a priori* clipped against the active brick region boundary, it is possible to determine exact step sizes for every sampling point. With particle tracing, some sample points (like the blue sample point to the right) might use step sizes that do not match the cell size in the current region.

Adaptive sampling is guided by the finest-level cell sizes stored with the brick regions. The adaptive sampling process is illustrated in Figure 4. A Runge–Kutta integration step is presented in Figure 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 3-D sample position  $p$  into the (user-configurable)  $X$ ,  $Y$ , and  $Z$  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<sup>4</sup> 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

```
// Adaptive sampling cell width
float finestCellWidth;
vec3 v1, v2, v3, v4;
float steplen = globalParameters.steplen;

// Read last position (or seed if t=0) from
// pre-allocated buffer for timestep t-1
vec3 p = pos[threadIdx*numTimeSteps+(t-1)];
sampleDirection(p,v1,finestCellWidth);
v1 *= finestCellWidth * steplen;
vec3 p1 = p + v1 * .5f;

sampleDirection(p1,v2,finestCellWidth);
v2 *= finestCellWidth * steplen;
vec3 p2 = p + v2 * .5f;

sampleDirection(p2,v3,finestCellWidth);
v3 *= finestCellWidth * steplen;
vec3 p3 = p + v3;

sampleDirection(p3,v4,finestCellWidth);
v4 *= finestCellWidth * steplen;
vec3 p4 = p + v4;

p += 1/6.f*(v1 + 2.f*v2 + 2.f*v3 + v4);

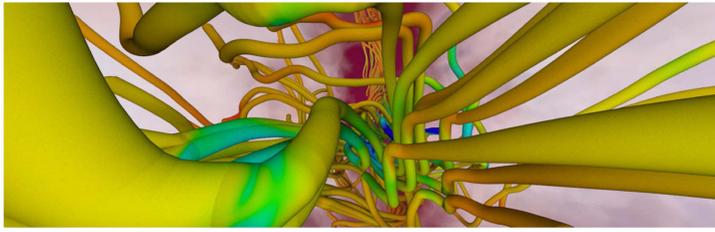
// Write new position into pre-allocated
// buffer for timestep t
pos[threadIdx*numTimeSteps+t] = p;
```

**FIGURE 5.** Performing a Runge–Kutta integration step.

$X$ ,  $Y$ , and  $Z$  channels of the vector field. The reconstructed 3-D 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).

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 Figure 4(b). 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 datasets 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.



**FIGURE 6.** Zoomed-in vector field visualization. Integration with an RT framework enables high-quality visualization compared to the omnipresent simple 2-D line strips that many rasterization-based visualization systems use to represent streamlines.

### High-Quality Rendering With Curved Geometries

At this point, when all the timesteps were computed, we are finished with the actual postprocessing 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.*<sup>8</sup> and just renders the control points as rounded cylinder strips (see Figure 6). The BVH for the geometry is rebuilt whenever the tracer, in a previous RT pipeline launch, generated new sample points.

Integration with an RT-based pipeline, of course, also enables high-quality rendering and shading techniques (see Figure 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 the two user case studies where our system was used for postprocessing followed by visual exploration to gain insights into the astrophysical datasets. The two datasets 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

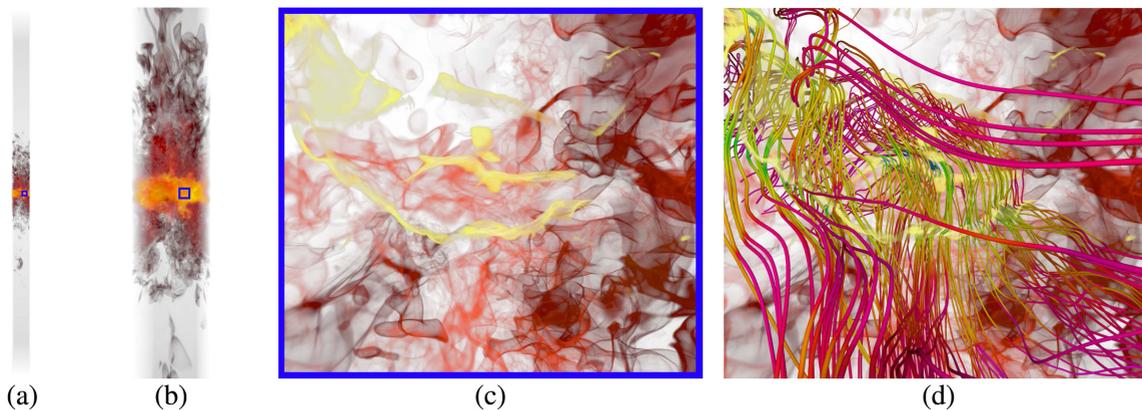
One key research question in modern astrophysics is how stars in our Galaxy (the Milky Way) are forming.

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 the 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.<sup>9</sup> 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 3-D simulations to model the formation and evolution of molecular clouds—and their embedded stars.

In the SILCC-Zoom project,<sup>10</sup> the evolution of molecular clouds, which are embedded in a part of a spiral galaxy (in turn modeled in the larger-scale SILCC project,<sup>11</sup>), is modeled by means of state-of-the-art, 3-D, magneto-hydrodynamical simulations. For this purpose, the versatile astrophysical AMR code FLASH is used,<sup>5</sup> 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 Figure 7, we visualize one snapshot of the simulation, gradually zooming in from the largest onto the smallest scales. This particular snapshot consists of 72.8M cells and 142K 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  $4096 \times 4096 \times 81,920$  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 requires the usage of AMR. In the SILCC-Zoom simulation, the represented scales extend from galactic scales [ $\geq 1$  kpc, Figure 7(a) and (b)], over molecular clouds [1–100 pc, Figure 7(c) and (d)] down to their complex substructure [0.1–1 pc, yellow material in Figure 7(c) and (d)]. This substructure consists of the 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



**FIGURE 7.** (a) and (b) Graphical representation of the SILCC-Zoom simulation zooming in from the galactic scale onto (c) the molecular cloud 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.

light-year, 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 Figure 7(c), 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 3-D 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 RT and ambient occlusion, the spatial relationship is, however, revealed. As can be nicely seen in Figure 7(d), 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 preprocess, 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 an RT 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 be a neutron star) may interact with the envelope of the other 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, the so-called “multimessenger” event.

---

*THE REMAINING CORE OF THE RED GIANT WILL THEN EXPLODE AND FORM A SECOND NEUTRON STAR.*

---

This process of common envelope ejection has only recently been studied in 3-D hydrodynamics.<sup>12</sup> This study also used the AMR code FLASH, but the physical scale of the star-forming molecular clouds described in the previous section is of order 1 kpc, whereas the physical scale of the binary neutron star system is of order 1000 solar radii, with the neutron star having a radius of 10 km. The timescales involved are also quite different. For the star-forming molecular clouds described in the previous section, the relevant physical processes occur on timescales of order  $10^6$  years, whereas the common envelope ejection occurs on timescales of order one day. Thus, both the spatial and temporal resolution of the two datasets is quite different.

In this study, a neutron star orbits the core of a red supergiant and successfully ejects its envelope after a few orbital passages. This allows the neutron star to reach a final orbital separation before the red giant evolves to a neutron star and, thus, for the eventual neutron star merger to be observable. Figure 8 shows the passage of the neutron star through the envelope of the red giant.

In the visualization, the streamlines track the velocity flow of the gas, which is imparted to it by the orbital motion of the neutron star. There is a velocity flow both locally, surrounding the orbiting neutron star (this shows the flow of material around the neutron star and the possibility of Bondi–Hoyle–Lyttleton accretion of material onto the neutron star—this is potentially

observable, as a secondary electromagnetic source, and is also important for determining the growth of the neutron star during its orbit; if it accretes too much material, the neutron star will collapse to a black hole), and globally, as a result of multiple passages of the neutron star, showing the ejection of gas material in the common envelope. The flow visualization conveys how the gas is shocked approximately radially outward after the neutron star sweeps it out.

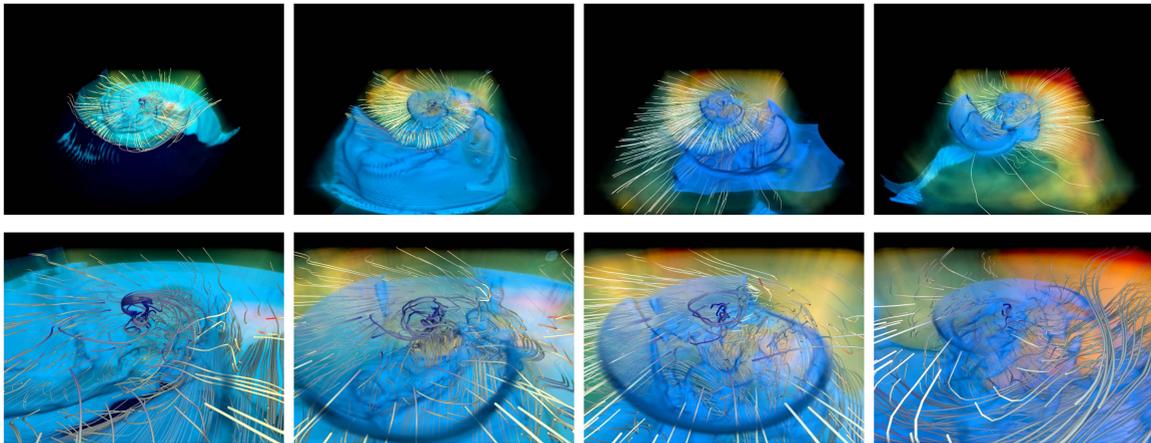
## PERFORMANCE MEASUREMENTS

We present results of a comparative performance evaluation in Figure 9. There we compare against the kd-tree method by Wald *et al.*<sup>4</sup> We, however, built the kd-tree over the brick set induced by the ExaBricks data structure.<sup>6</sup> 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 100K steps per particle through the molecular cloud dataset from our first example application. We observe how, with increasing input sizes, the GPU becomes fully saturated at around  $2^{15}$  particles advected and we gradually benefit from hardware-accelerated tree traversal. The advantage of using RT cores becomes more pronounced the more particles we advect.

## DISCUSSION AND CONCLUSION

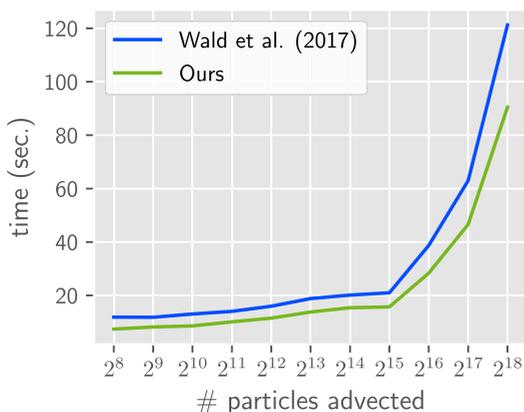
We presented an unconventional yet effective technique to perform point containment queries into AMR vector fields. The technique is quite general and has previously also been used by other authors, e.g., to accelerate point containment queries in finite-element datasets and other compute-intensive applications. We adapted the technique to AMR vector field visualization, but note that the overall idea of mapping 3-D tree traversal to an RT problem and by that enabling the use of RT cores is generally applicable to all sorts of problems. Using RT 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 3-D 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



**FIGURE 8.** Binary neutron star formation via common-envelope ejection. 3-D 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).

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 not possible. In the future, we, however, plan to investigate if an approach that is not based on RT but instead parallelizes the particle advection step over the set of active brick regions can result in a performance increase. Similar approaches based on macrocells from structured grids, however, often suffer from scalability issues,



**FIGURE 9.** Execution times for advecting particles 100K times through the magnetic field of the molecular cloud dataset example application.

and the adjacency information that is required to advect the particles from one region to another is nontrivial 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 addressing interactive data structure construction.

We argue that the approach of using RT 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 semistructured grids, but also to generally unstructured, finite elements. The method could also potentially be extended to support multiple vector fields. In this respect, our article is also meant to serve as an example and potentially a guide for practitioners to implement this technique in their own frameworks.

## ACKNOWLEDGMENTS

The work of Stefan Zellmann was supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—under Grant 456842964. The work of Daniel Seifried and Stefanie Walch-Gassner

was supported in part by the Deutsche Forschungsgemeinschaft (DFG) via the Collaborative Research Center SFB 956 “Conditions and Impact of Star Formation” (subprojects C5 and C6). The work of Stefanie Walch-Gassner was supported in part by European Research Council via ERC Starting under Grant 679852 (RADFEEDBACK). The work of Jamie A. P. Law-Smith was supported in part by the lux supercomputer at UCSC, funded by NSF MRI under Grant AST 1828315 and in part by the HPC facility, University of Copenhagen, funded by a grant from VILLUM FONDEN (Project 16599). The authors would like to thank the Ministry of Culture and Science of the State of North Rhine-Westphalia for supporting the work through the PROFILBILDUNG Grant PROFILNRW-2020-038 C. 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, 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 *Proc. IEEE Visual. Conf.*, 2020, pp. 96–100, doi: [10.1109/VIS47514.2020.00026](https://doi.org/10.1109/VIS47514.2020.00026).
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 *Proc. IEEE Visual. Conf.*, 2019, pp. 256–260, doi: [10.1109/VISUAL.2019.8933539](https://doi.org/10.1109/VISUAL.2019.8933539).
4. I. Wald, C. Brownlee, W. Usher, and A. Knoll, “CPU volume rendering of adaptive mesh refinement data,” in *Proc. SIGGRAPH Asia Symp. Visual.*, 2017, pp. 9:1–9:8, doi: [10.1145/3139295.3139305](https://doi.org/10.1145/3139295.3139305).
5. A. Dubey *et al.*, “Challenges of extreme computing using the FLASH code,” in *Numerical Modeling of Space Plasma Flows* (Astronomical Society of the Pacific Conference 385), N. V. Pogorelov, E. Audit, and G. P. Zank, Eds., Orem, UT, USA: ASP Books, 2008, pp. 145–150.
6. I. Wald, S. Zellmann, W. Usher, N. Morrical, U. Lang, and V. Pascucci, “Ray tracing structured AMR data using ExaBricks,” *IEEE Trans. Vis. Comput. Graphics*, vol. 27, no. 2, pp. 625–634, Feb. 2021, doi: [10.1109/TVCG.2020.3030470](https://doi.org/10.1109/TVCG.2020.3030470).
7. K. Joy, “Numerical methods for particle tracking in vector fields,” Visual. Res. Lab., Univ. California, Davis, CA, USA, Visualization Notes, Feb. 1999. [Online]. Available: [https://web.cs.ucdavis.edu/~ma/ECS177/papers/particle\\_tracing.pdf](https://web.cs.ucdavis.edu/~ma/ECS177/papers/particle_tracing.pdf)
8. I. Wald *et al.*, “Using hardware ray transforms to accelerate ray/primitive intersections for long, thin primitive types,” *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 3, no. 2, pp. 1–16, Aug. 2020, doi: [10.1145/3406179](https://doi.org/10.1145/3406179).
9. J. Roman-Duval, J. M. Jackson, M. Heyer, J. Rathborne, and R. Simon, “Physical properties and galactic distribution of molecular clouds identified in the galactic ring survey,” *Astrophys. J.*, vol. 723, no. 1, pp. 492–507, Nov. 2010, doi: [10.1088/0004-637X/723/1/492](https://doi.org/10.1088/0004-637X/723/1/492).
10. D. Seifried *et al.*, “SILCC-Zoom: The dynamic and chemical evolution of molecular clouds,” *Monthly Notices Roy. Astron. Soc.*, vol. 472, no. 4, pp. 4797–4818, Dec. 2017, doi: [10.1093/mnras/stx2343](https://doi.org/10.1093/mnras/stx2343).
11. S. Walch *et al.*, “The SILCC (Simulating the LifeCycle of molecular Clouds) project—I. Chemical evolution of the supernova-driven ISM,” *MNRAS*, vol. 454, no. 1, pp. 238–268, Nov. 2015, doi: [10.1093/mnras/stv1975](https://doi.org/10.1093/mnras/stv1975).
12. J. A. P. Law-Smith *et al.*, “Successful common envelope ejection and binary neutron star formation in 3-D hydrodynamics,” Nov. 2020, *arXiv:2011.06630*, doi: [10.48550/arXiv.2011.06630](https://doi.org/10.48550/arXiv.2011.06630).

**STEFAN ZELLMANN** is at Institute of Visual Computing, Bonn-Rhein-Sieg University of Applied Sciences, 53757, Sankt Augustin, Germany, and also at the University of Cologne, 50937, Cologne, Germany, where he is a lecturer. His research interests include the interface between high-performance computing and real-time rendering. Zellmann received his Ph.D. degree in computer science from the University of Cologne. Contact him at [stefan.zellmann@h-brs.de](mailto:stefan.zellmann@h-brs.de).

**DANIEL SEIFRIED** is a postdoc in the field of astrophysics at I. Physical Institute, University of Cologne, 50937, Cologne, Germany. His research interests include 3-D, magneto-hydrodynamical simulations of star formation. Seifried received his diploma in physics from the University of Würzburg, Würzburg, Germany, and his Ph.D. degree in physics from the University of Heidelberg, Heidelberg, Germany. Contact him at [seifried@ph1.uni-koeln.de](mailto:seifried@ph1.uni-koeln.de).

**NATE MORRICAL** is currently working toward the Ph.D. degree with the University of Utah, Salt Lake City, UT, 84112, USA, and an intern alumni at NVIDIA, Santa Clara, CA, 95051, USA, and Pixar, Emeryville, CA, 94608. His research interests include high-performance GPU computing, real-time ray tracing, and human-computer interaction. Morriral received his B.S. degree in computer science from Idaho State University, Pocatello, ID, USA. Under Valerio Pascucci, he is a member of the CEDMAV Group with the Scientific Computing and Imaging Institute (SCI), Salt Lake City. Contact him at natemorriral@gmail.com.

**INGO WALD** is a director of ray tracing at NVIDIA, Santa Clara, CA, 95951, USA. Wald received his Ph.D. degree (Dr. ing.) in computer science from Saarland University, Saarbrücken, Germany. Contact him at iwald@nvidia.com.

**WILL USHER** is a ray-tracing software engineer at Intel, Santa Clara, CA, 95054, USA. His research interests include CPU and GPU ray tracing, distributed rendering, and scientific visualization. Usher received his Ph.D. degree in computer science on data management and visualization for massive scientific datasets from the Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT, USA. Contact him at will.usher@intel.com.

**JAMIE A. P. LAW-SMITH** is a fellow at the Institute for Theory and Computation, Center for Astrophysics—Harvard & Smithsonian, Cambridge, MA, 02138, USA. His research 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, with focus on theoretical astrophysics and physics. Law-Smith

received his Ph.D. degree in astronomy and astrophysics from the University of California Santa Cruz, Santa Cruz, CA, USA. Contact him at jamie.law-smith@cfa.harvard.edu.

**STEFANIE WALCH-GASSNER** is a full professor of theoretical astrophysics leading the theoretical astrophysics group and is the managing director at I. Physics Institute, University of Cologne, 50937, Cologne, Germany, and vice director at the Center for Data and Simulation Science, University of Cologne. Her research interests include the physics of the interstellar medium, star formation and stellar feedback as well as the development of new methods and algorithms for high-performance computing, in particular magneto-hydrodynamics, radiative transfer, and astrochemistry. Walch-Gassner received her diploma in physics and the Ph.D. degree (Dr.rer.nat.) in astrophysics from the Ludwig-Maximilians Universität München, Munich, Germany. Contact her at walch@ph1.uni-koeln.de.

**ANDRÉ HINKENJANN** is a full professor of computer graphics and interactive environments at the Department of Computer Science, Bonn-Rhein-Sieg University of Applied Sciences (BRSU), 53757, Sankt Augustin, Germany, and an adjunct professor at the Department of Computer Science, University of New Brunswick, Fredericton, NB, E2K 5E2, 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. Hinkenjann received his diploma and Ph.D. degree in computer science from TU Dortmund, Dortmund, Germany. He is a member of ACM SIGGRAPH, Eurographics, and German GI and Member of IEEE. Contact him at andre.hinkenjann@h-brs.de.