Eurographics Conference on Visualization (EuroVis) 2014
H. Carr, P. Rheingans, and H. Schumann
(Guest Editors)

Volume 33 (2014), Number 3

RBF Volume Ray Casting on Multicore and Manycore CPUs

Aaron Knoll[1] Ingo Wald[2] Paul Navratil[1] Anne Bowen[1] Khairi Reda[3] Michael E. Papka[4] Kelly Gaither[1]

[1]Texas Advanced Computing Center, the University of Texas at Austin
[2]Intel Corporation
[3]Electronic Visualization Laboratory, University of Illinois at Chicago
[4]Argonne National Laboratory

Abstract

Modern supercomputers enable increasingly large N-body simulations using unstructured point data. The structures implied by these points can be reconstructed implicitly. Direct volume rendering of radial basis function (RBF) kernels in domain-space offers flexible classification and robust feature reconstruction, but achieving performant RBF volume rendering remains a challenge for existing methods on both CPUs and accelerators. In this paper, we present a fast CPU method for direct volume rendering of particle data with RBF kernels. We propose a novel two-pass algorithm: first sampling the RBF field using coherent bounding hierarchy traversal, then subsequently integrating samples along ray segments. Our approach performs interactively for a range of data sets from molecular dynamics and astrophysics up to 82 million particles. It does not rely on level of detail or subsampling, and offers better reconstruction quality than structured volume rendering of the same data, exhibiting comparable performance and requiring no additional preprocessing or memory footprint other than the BVH. Lastly, our technique enables multi-field, multi-material classification of particle data, providing better insight and analysis.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics

1. Introduction

Direct volume rendering (DVR) is an increasingly popular modality for visualizing 3D scalar fields in scientific data. It reconstructs, classifies and shades any continuous scalar field, enabling better insight than surface-based visualization in many applications. Volume rendering of structured data is now commonplace, and optimized methods have been developed for unstructured mesh and finite-element data. Generally, these methods have been implemented on GPUs, due to their high computational throughput and built-in hardware texture sampling features. However, volume rendering directly from unstructured point data remains a challenge.

N-body codes in particular produce large quantities of data. For example, large molecular dynamics simulations can generate megabytes-to-gigabytes per time step and tens-to-hundreds of thousands of time steps; large astrophysics simulations can generate terabytes to petabytes per timestep. At scale, post-processing and moving such data is prohibitive. Resampling particle data into a structured volume costs memory and computational time, as well as sacrificing information and visual quality (e.g., Figure 1). Computing isosurfaces is similarly costly, and prevents interactive classification and analysis of the original scalar fields. These factors motivate in transit and in situ visualization on high performance computing (HPC) resources, minimal post-processing, and efficient algorithms for direct volume rendering of point data.

Existing methods for particle volume rendering vary. Though efficient, splatting techniques that filter in screen-space do not provide the same level of quality as volume rendering with full domain-space reconstruction. The current state-of-the-art GPU technique [FAW10] resamples data into an image-space structured grid, which is sensitive to choice of resolution and limits multi-field classification. Other GPU approaches [RKN*13, JFSP10, OKK10] suggest RBF performance significantly slower than that of structured DVR. Moreover, for in situ and in transit visualization not all HPC resources have GPUs. We desire the flexibility to efficiently render on a wide variety of architectures with SIMD-capable CPUs, and new CPU-like "many integrated core" (MIC) hardware such as the Intel® Xeon Phi™ coprocessor. MICs are increasingly used in supercomputers, such as Tianhe-2 (currently #1 on the Top500 [TOP13]) and Stampede (currently #7). Efficient visualization on these architectures requires a framework that takes advantage of SIMD vector instructions with varying width and arithmetic capabilities.

Our contribution is a novel method for efficient RBF volume rendering on CPU and MIC hardware. Our algorithm uses coherent bounding volume hierarchy (BVH) traversal to efficiently evaluate the RBF field, and performs DVR inte-

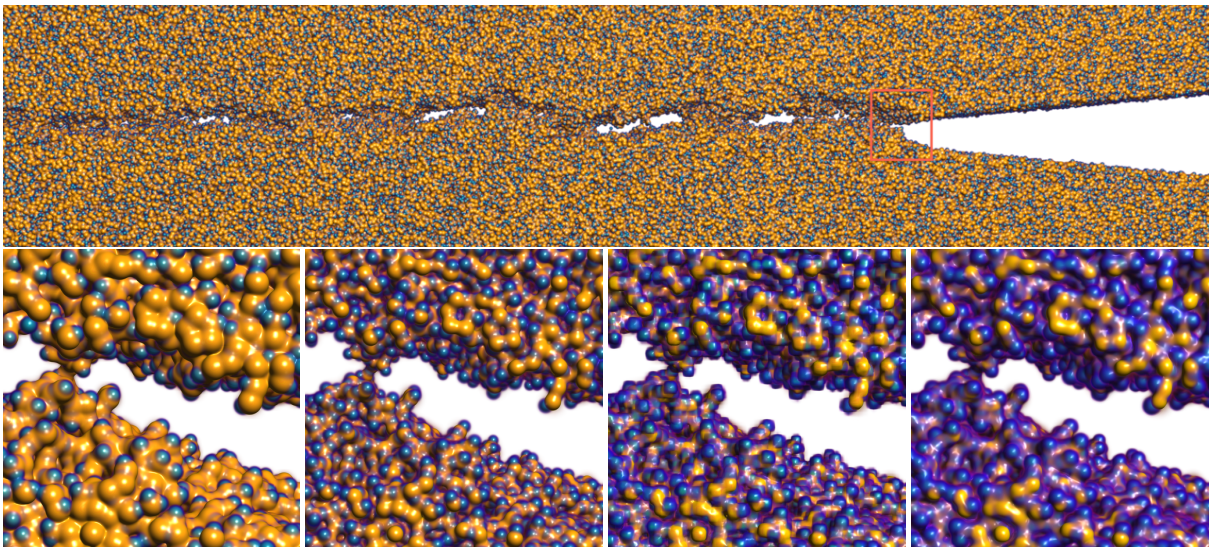


Figure 1: *5 million atom molecular dynamics glass (SiO_2) fracture data, rendered on a Intel® Xeon Phi™ SE10P in Stampede. **Top:** RBF volume rendering using two transfer functions to classify silicon and oxygen atoms as separate fields (11.7 fps at 2560×512, $\kappa = 4, dt = .5, \sigma = 2$). **Bottom:** From left to right: close-up of the the top view (4.7 fps at 1024×1024); the same view reconstructed with narrower RBFs (3.1 fps, $\kappa = 1, dt = .125, \sigma = 2.835$); trilinear interpolation of two-field structured data (2.1 fps, $dt = .125$), and tricubic B-spline interpolation of structured data (0.221 fps, $dt = .125$). The two-field structured data (1 voxel per Ångström, 32-bit float) occupies 800 MB on disk and took 80 seconds to precompute on a 16-core CPU. RBF volume rendering incurs less memory overhead and exhibits better performance and quality for rendering atomistic data.*

gration along rays in a subsequent step. Crucially, this eliminates the need for costly per-ray neighbor search, and repeat queries of the same basis functions at different samples. We implement this method in bnsView [KWN*13] in the IVL [LHW12] SPMD language, which generates optimized vector instructions in C++ for multicore CPU or MIC backends, enabling fast rendering. Our technique does not rely on simplification or LOD, does not use a proxy to downsample ray samples, and achieves interactivity on both MIC and CPU platforms. We show that our system performs competitively with the best-available GPU approaches, and enables a variety of different use cases in HPC-driven visualization.

2. Related Work

Direct volume rendering, or DVR, [DCH88] is a process of directly rendering a 3D scalar field by evaluating the field at sample points, classifying samples into colors via a transfer function, and integrating these classified color values to produce a final image. Smoothed particle hydrodynamics [Mon92] is a mesh-free (Lagrangian) method for simulating motion of fluids, employed in cosmology, astrophysics, materials science, and more generally applications of computational fluid dynamics. *SPH volume rendering* [JFSP10,FAW10] refers to the process of volume rendering SPH data directly, using basis functions from the SPH data for image reconstruction. The same method can in fact be used for other particle data, for example atomistic data from molecular dynamics. We refer to this class of techniques as *RBF volume rendering*.

Volume rendering of point data. RBF volume rendering is costly, and past interactive applications have generally been pursued on GPUs. Jang et al. [JWH*04] were the first to use a small number of RBF's to approximate larger volume data, reconstructed in a GPU fragment shader and rendered with slice-based volume rendering. This approach was extended to ellipsoidal basis functions [JBL*06] and density functionals from quantum chemistry [JV09]. For rendering of larger SPH data, Jang et al. [JFSP10] employed a kd-tree to improve bound tightness. Fraedrich et al. [FAW10] dynamically resample from an octree hierarchy into perspective-space uniform grids of predetermined size, and achieve nearly interactive performance on an NVIDIA 280 GTX for up to 42M particles (0.1 fps). This approach likely remains state-of-the-art, and would be faster still on current GPUs. However, it is difficult to fairly evaluate, as it uses LOD to prune the particle octree, and filters pixels through a (trilinearly interpolated) proxy grid volume. Orthomann et al. [OKK10] describe a similar system traversing an octree, using ray packets similar to the concept described in our paper, but implemented differently on the GPU, and employed as a form of proxy. Reda et al. [RKN*13] demonstrate interactive performance for megascale molecular data using a uniform grid as an acceleration structure, and volume ray casting from RBF's in a GPU shader.

Splatting, particle and glyph approaches. We differentiate between volume rendering of point data and point-based volume rendering (splatting). Splatting performs reconstruction in image space using different kernels entirely. While less computationally costly, volume splatting techniques [ZPvBG01a, CRZP04] are insufficient for reproducing continuous surfaces, and techniques optimized for sur-

face reconstruction [ZPVBG01b] are ill-suited for volume rendering. In one of the first applications of volume rendering SPH data, Kähler et al. [KAH07] used an octree to simultaneously splat particle data (simplified using LOD) and volume-render approximated data on a structured grid.

Absent image-space reconstruction, many techniques exist for fast rasterization or ray casting of large number of points, glyphs or particle impostors. Gribble et al. [GIK*07] employed coherent ray tracing algorithms on the CPU to efficiently render millions of opaque sphere glyphs. Megamol [GBM*12] uses a combination of GPU rasterization, ray casting of sphere impostors, and image-space filtering to efficiently render millions of atoms. Fraedrich et al. [FSW09] demonstrated an extremely fast out-of-core LOD particle renderer for real-time rendering of astrophysics data. In contrast to their SPH volume rendering work [FAW10], the particle approach is faster by 1-2 orders of magnitude and excels at its specific application. However, to reconstruct smooth isosurfaces and classify material boundaries, full volume rendering with postclassification, thus RBF volume rendering, is necessary.

Offline and surface approaches. The astrophysics and cosmology communities frequently employ offline parallel batch tools [Pri07, DRGI08, TSO*11] for rendering, plotting and specialized analysis such as radiative transfer [ACP08]. Generally, these do not take advantage of SIMD, have limited if any GPU acceleration, and are not suitable for interactive rendering. Splotch [DRGI08] assumes that particles do not overlap and blends in potentially incorrect order, resulting in artifacts similar to those of rasterization-based particle renderers. Turk et al. [TSO*11] converts data to structured volumes and renderers in software.

A large body of existing work exists on extraction of implicit surfaces from radial basis functions, as pioneered by Wyvill et al. [WMW86]. Relevant to our examples below, Navrátil et al. [NJB07] use marching cubes to extract single isosurfaces from multifield cosmological data. Stone et al. [SHUS10] implement CUDA-accelerated isosurface extraction from Gaussian RBF fields for fast computation of molecular surfaces. Though efficient, these approaches would sacrifice reconstruction quality and limit opportunities for dynamic classification.

3. Background

This section covers our method's theoretical underpinnings. Readers familiar with RBFs (Sections 3.1–3.3) and coherent ray tracing (Section 3.4) may proceed directly to the presentation of our algorithm in Section 4.

3.1. Radial Basis Functions

A radial basis function (RBF) is a continuous, real-valued function ϕ whose value decays with respect to distance from a particle. A RBF scalar field Φ is defined by summing the kernels for all kernels i contributing to a point $\mathbf{x}$ in space:

$$d_i(\mathbf{x}) = ||\mathbf{x} - \mathbf{x_i}|| \qquad \Phi(\mathbf{x}) = \sum_i \phi_i(d_i(\mathbf{x})) \qquad (1)$$

Common choices of ϕ are Gaussians or piecewise-smooth polynomials with compact support. Compact support has the advantage of having zero contribution outside of the radius, whereas Gaussians have infinite support and decay smoothly. It is equally possible to use RBFs that model the physical properties of the particles, based on empirical or semi-empirical data (e.g., radially averaged plots for DFT-plotted molecular data [KCL*13]). In practice, truncating Gaussians outside of a sufficiently wide radius of influence (support) is effective, and allows the user control over desired kernel width. Thus we use the Gaussian kernel

$$\phi_i(x) = \kappa\, k_i\, e^{-d_i(\mathbf{x})^2/2r_i^2} \qquad (2)$$

where k_i is the value of the kernel (e.g. density), r_i is the radius of the Gaussian (e.g. covalent radius for molecular data), and κ is a global density scale (defaulting to $\kappa = 1$). We truncate at a support radius of σr_i (defaulting to $\sigma = 2$). Both σ and κ can be changed by the user dynamically, without modification of data structures.

3.2. Volume rendering integral

Volume rendering is a special case of the light transport equation [Max95], in which (emissive) irradiance integrated over t along a ray segment $[a,b]$, for the scalar field Φ and transfer function with color $\mathbf{C}$ and opacity τ, for a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ where $\Phi(t) = \Phi(\mathbf{r}(t))$, is given as:

$$I_{[a,b]}(u) = \int_a^b \mathbf{C}(\Phi(u))\; \mathrm{e}^{-\int_a^u \tau(\Phi(t))dt} du \qquad (3)$$

If we assume C and τ are constant on $[a,b]$, we can approximate opacity discretely as $\alpha = 1 - exp(-\tau\Delta t)$ where $\Delta t \approx \int_a^u \tau(\Phi(t))dt$. This is integrated numerically via blending, as sketched in Listing 1, where Δt (abbreviated `dt`) is the sampling step size in domain space.

Listing 1: *DVR blending*

```
void dvr_blend(float Phi, Ray ray, TransFunc tf){
  Color s = shade(tf.classify(Phi))
  float alpha = 1 - exp(-s.a * dt)
  ray.color.rgb += s.rgb * (1 - ray.color.a) * alpha
  ray.color.a += (1 - ray.color.a) * alpha
}
```

The main challenge in RBF volume rendering is efficiently evaluating Φ. As blending is non-commutative, the order in which samples are integrated in Equation 3 matters. This has ramifications on the choice of volume rendering algorithm.

3.3. Reconstruction

To reconstruct the scalar field Φ into discrete samples along viewing rays, we have two options, as shown in Figure 2:

1. *Direct method*: for each sample, evaluate kernels for all particles that overlap that sample;
2. *Proxy method*: for each particle, evaluate kernels for all proxy samples that this particle overlaps, followed by separate interpolation of the proxy samples.

These are also referred to as *gathering* and *scattering*, repsectively, in existing reconstruction literature (see, e.g., [OKK10]).

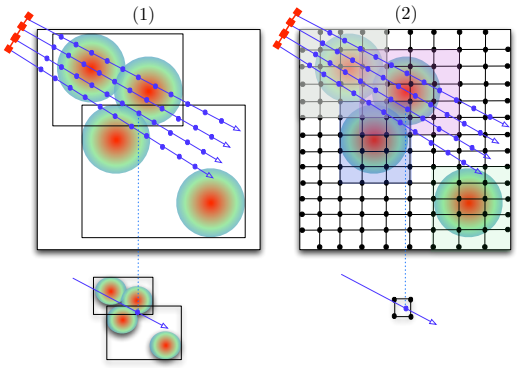


Figure 2: *Options for RBF field reconstruction. (1) Direct method: the field is evaluated directly at each sample along the ray, which can prove expensive. (2) Proxy method: the field is resampled into a grid, and then inexpensively reconstructed from the proxy, at the cost of memory and/or quality.*

In the direct approach the scalar field is evaluated per-sample, which lends itself trivially to volume rendering. Evaluating $\Phi(\mathbf{x})$ requires determining all kernels whose support overlaps $\mathbf{x}$. This problem is commonly referred to as region-finding [Sam90], and costs $O(P\ k\ N)$ to $O(P \log N)$ depending on the chosen acceleration structure, for k desired particles out of N total, and P pixels. While slicers (e.g. [JWH*04, JFSP10]) typically perform reconstruction at all samples, methods employing ray casting with acceleration structures (e.g. [RKN*13]) can use the structure to region-find, skip empty space, and exploit early ray termination. Though conceptually simple, the direct method requires repeat evaluation of the same kernels (potentially far apart in memory) inside a tight inner loop, making it costly.

In the proxy method, each particle is evaluated only once for all samples that it overlaps. This approach is taken by all methods precomputing a structured volume, and the dynamic grid method of Fraedrich et al. [FAW10]. The main advantage is that proxy geometry can be lower resolution or less expensive to render than the original RBF field using the direct method. Equally important, iterating once over particles that are close together in memory, for multiple samples in the proxy (also close together) fosters better access patterns, hence performance gains. The major disadvantage of this approach is the memory required to store the proxy (in many cases larger than the original RBF data) and time required to compute it, as well as detail lost by the proxy itself.

3.4. Coherent ray tracing and bnsView

Coherent ray tracing [WSBW01] is a technique for bundling rays together into packets that simultaneously perform traversal, intersection, and shading routines in lockstep, in a manner that encourages full use of vector instructions. Consequently, both memory access and computational costs are amortized over the number of rays in the packet. Coherent ray tracing methods have enabled interactive ray tracing of polygonal data on CPUs [BSP06], and efficient visualization of structured and unstructured data [WFKH07, KTW*11, BPL*12]. Coherent BVH traversal [WBS07] is the most popular acceleration method due to performance, simplicity, availability of fast builders, and graceful degeneration to single-ray performance for incoherent rays or large data. For structured volume rendering, Knoll et al. [KTW*11] show that coherent BVH traversal fosters similar performance for small (2 MB) and large (8 GB) data of varying resolution, given the same number of samples along the ray.

IVL SPMD Language. With SPMD languages such as IVL [LHW12] and its open-source relative ISPC [PM12], it is possible to write a coherent BVH ray tracers in a single code path for multiple SIMD CPU backends, including Intel®Xeon Phi™. IVL kernels are written in scalar form. From these, the compiler emits C++ and intrinsics code with data automatically laid out in structure-of-arrays (SOA) format for efficient use of SIMD instructions. As with ISPC, the programmer defines variables as `uniform` or `varying`, which determines whether they are scalar or vector quantities, respectively. Control flow within each thread is handled automatically by the compiler. IVL offers several advantages over ISPC (C++ classes and operator overloading), and both offer certain advantages over GPU languages (support for recursion, little penalty for large kernels or high in-kernel memory usage). It would be straightforward to reproduce this work in ISPC by leveraging Embree [WWB*14].

bnsView. Our application is built on top of bnsView [KWN*13], a molecular visualization tool written in IVL that achieves GPU levels of performance on CPU and MIC architectures, for ray tracing and volume rendering molecular models. Prior to this work, bnsView employed coherent BVH traversal for ball-and-stick rendering, and uniform grid traversal for structured volume rendering of precomputed volume data. We experimented with straightforward implementation of the direct approach in bnsView and IVL, using the BVH for region-finding but not ray traversal. While efficient for small data and opaque transfer functions encouraging early termination, it exhibited very slow performance for larger data (over 100K atoms). In extending bnsView to RBF volume rendering, a new approach was necessary.

4. RBF DVR with Coherent BVH Traversal

This section describes our algorithm, which applies fast CPU ray tracing techniques to RBF volume rendering. In particular, we use a novel technique combining coherent BVH traversal (working on packets rather than rays) with a buffered traveral method that amortizes cost over multiple RBF samples along each ray.

We compute Φ independently of DVR integration by maintaining a sample buffer, and computing ϕ once for each particle for all samples in that buffer. When traversal completes, the buffered samples are integrated per packet in the correct order. This approach fosters more coherent memory access by ensuring that each particle is traversed once in for the whole packet of rays. In this way, we achieve the advantages of both the direct and proxy methods, accessing memory in a more coherent per-particle fashion like the proxy approach

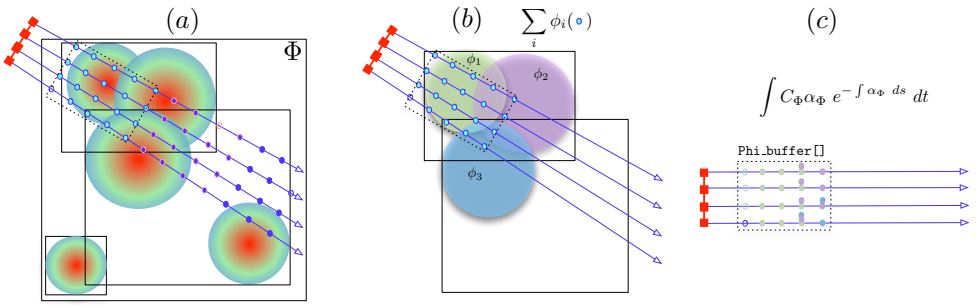


Figure 3: *Coherent RBF algorithm. (a) Iterate through samples along a packet, depositing into a fixed-size buffer. (b) Each BVH leaf node (particle with support σ) adds its value to all samples it overlaps in the buffer. (c) When this buffer has finished traversal, it is integrated front-to back using Equation 3, and we proceed to the next set of samples. In this manner, particles can be added to the buffer in any order, allowing for traversal of each particle once per packet and improving memory access.*

but maintaining only a small buffer of samples with little overhead, yielding the exact same quality as the direct ray casting method. This concept is illustrated in Figure 3.

While it shares some common features with the dynamic image-space grid method of Fraedrich et al. [FAW10] and [OKK10], our algorithm is fundamentally new in that:

- by computing samples *per-packet* as opposed to across the entire image, and more specifically not subsampling, we require less memory and can store and integrate all samples required for full ray casting with no loss in quality;
- with *coherent BVH traversal*, the nodes of the acceleration structure (as well as particles within) are traversed only once, improving efficiency;
- we use the *same structure* for both RBF volume rendering and ray tracing of ball-and-stick geometry;
- by performing this integration in multiple passes, we are able to further lower the memory requirements of our buffer, and take advantage of *early ray termination* on a per-packet basis.

Moreover, as discussed in Sections 5 and 6, this algorithm can run efficiently on non-GPU platforms with larger memory, and enables analyses that would be difficult with proxy methods (multi-field classification of different particles).

4.1. Coherent RBF volume rendering algorithm

Given a list of particles, a bounding volume hierarchy, a sampling step size `dt`, transfer function and shading method, our algorithm performs volume ray casting, e.g. it evaluates Φ for samples along each ray, and integrates Equation 3 using the front-to-back algorithm shown in Listing 1.

To accomplish this efficiently, we group rays into packets, (number of rays per packet N maps to chosen SIMD width; see Section 5). For each packet, we then do the following:

1. Intersect the packet with the root bounds of the BVH (extended by σ) to find `tenter, texit`;
2. Determine the total number of samples along any ray in the packet, `K = (texit - tenter) / dt`;
3. Create a buffer `Phi_buf` with `K` samples for each ray
4. Traverse the BVH, with nodes extended by σ, summing ϕ for every leaf at every sample (Equation 1) and storing that in `Phi_buf`;
5. Integrate `Phi_buf` front-to-back along the rays, classifying and shading as necessary (Equation 3).

This algorithm requires (and takes advantage of) methods to compute `phi()` and `dvr_integrate()` for entire packets as opposed to single rays. On GPUs this is handled internally; on CPU and MIC it entails SIMD vector instructions or a SPMD language. IVL/ISPC pseudocode is given in Listing 2 in Section 5.

4.2. Multi-pass algorithm with early termination

On the CPU and MIC, dynamically allocating a single large `Phi_buf` for each packet works fine and delivers acceptable performance. However, not surprisingly, we found that even better performance was possible by allocating a smaller buffer once and filling it in several separate BVH traversals. This simply requires replacing Step 2 with `K=32` samples once and placing the method described in Section 4.1 in a loop. IVL/ISPC pseudocode is sketched in Listing 3 in Section 5. In particular, for larger data this has the advantage of enabling early termination without traversing the full BVH: when every ray in the packet has reached maximum opacity, we do not need to proceed with further passes.

5. Implementation

In this section we describe the implementation of our method on CPU and MIC. We chose these architectures for the reasons outlined in the introduction (platform portability, larger memory) but also because they are well-suited to tackle this problem. Specifically, CPU and MIC offer:

- Large memory, both per core (for the buffer) and in total;
- SIMD vector units, and explicit mechanisms for control flow both inside and outside of SIMD lanes (to ensure coherent traversal and better memory access);
- Encouragement of large, multi-function kernels designed to operate independently across separate threads.

It would be possible to implement our method effectively in a framework such as Manta [BSP06], using explicitly defined ray packets and manually-coded SIMD intrinsics.

However, for portability we desire a system that enables operation on a wider variety of CPUs with different SIMD "backend" architectures (various versions of SSE, 8-float AVX, 16-float MIC). For this, we employ the IVL SPMD compiler [LHW12] and the RIVL framework.

5.1. Preprocess

The preprocessing phase is performed when data is read from disk, either statically or as part of in-transit visualization. The BVH is constructed on the host and, if necessary, data are then sent across the PCI bus to the MIC.

For BVH construction, we currently use the existing single-threaded SAH builder implemented in RIVL and described in [KWN*13]. We use 4 primitives (points) per leaf node; this can be modified for faster construction at some cost in rendering performance.

5.2. BnsView framework

BnsView and RIVL, including ball-and-stick and structured volume rendering are covered in greater detail in [KWN*13]. RIVL generates camera rays, distributing work to all CPU/MIC threads and calling the SPMD entry kernel. In, bnsView, the `trace()` kernel called by the RIVL renderer (either a ray tracer or ray caster) consists of two passes:

1. opaque geometry (e.g. ball-and-stick), using the BVH. This stores a single hit position `t_hit` along the ray.
2. volume rendering (either structured data, or RBF DVR using the new technique in this paper), which is then computed for samples from `t=0` to `t=t_hit`, and stores an integrated color and opacity.

Calling this function recursively, we can achieve secondary ray effects, such as shadows or ambient occlusion.

5.3. Coherent RBF DVR in IVL

The IVL implementation of our method is sketched in pseudocode in Listing 2. In SPMD, a packet is simply a `varying Ray`. The BVH, its stack and counters are all `uniform`. During traversal, if any ray intersects a node, the entire packet descends the tree and the sample buffer (for all rays) is filled. When `Phi` is evaluated , the associated `uniform` RBF is evaluated for all `varying` sample positions. Lastly, all samples for the packet are integrated, iterating using a `uniform t` variable spanning minimum and maximum sample positions. Otherwise, differences between our algorithm and coherent BVH traversal [WBS07] (the IVL implementation of which is described in [KWN*13]) are:

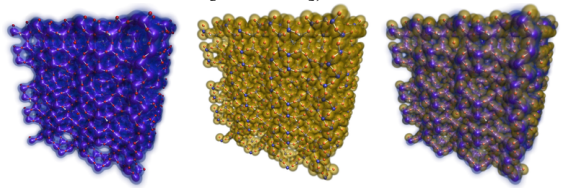


Figure 4: *Multi-material classification of RBF contributions of separate atoms as separate volumetric fields. From left to right: silicon atoms only, oxygen atoms only, and both silicon and oxygen in a zeolite structure.*

- multiple traversal passes, sketched in Listing 3;
- dynamically expanding extents of every BVH node by σ, the truncation radius specified by the user;
- in a leaf, for each particle, we determine the `uniform` minimum and maximum samples overlapping any ray, then iterate over them adding to `Phi_buf`;
- when BVH traversal completes (and for each pass), DVR integration is performed on these buffered samples.

Listing 2: *RBF DVR with coherent BVH traversal*

```
1  void rbf_dvr(varying Ray ray, varying float tenter,
2                                varying float texit)
3   while traversal stack not empty
4    pop node off traversal stack
5    if ray intersects extend(node,sigma) on [tenter, texit]
6     if node is a leaf
7      foreach varying t in {tenter.. texit}
8       Phi_buf[t] += phi(i,ray.org + ray.dir * t)
9     else
10     push both children
11  foreach uniform t in {rmin(tenter)..rmax(texit)}//reduce
12   dvr_blend(Phi_buf[t],ray,tf)
13   if (ray.color.a > .99) break
```

Listing 3: *Multi-pass algorithm*

```
1  void rbf_dvr_multipass()
2   foreach varying Ray ray
3    {first_t, last_t} = AABB_test(ray, bvh.bounds)
4    for (uniform t=first_t; t<last_t; t+=Phi_buf.size)
5     rbf_dvr(ray, t, t + Phi_buf.size)
6     if all(ray.color.a > .99) break
```

5.4. Shading

For most RBFs, it is straightforward to compute partial derivatives $\nabla\Phi$ analytically at the same time that ϕ is computed. In the case of our Gaussian, this is particularly trivial:

$$\nabla\Phi(\mathbf{x}) = -2\sum\phi_i(d_i(\mathbf{x}))(\mathbf{x}-\mathbf{x_i}) \qquad (4)$$

Computing analytical gradients for RBFs incurs little cost, in contrast to the high expense of central differences gradients for structured volume rendering in bnsView [KWN*13]. We do, however, need to store the gradient in the sample buffer, which requires quadrupling the size of our sample buffer (4 floats instead of one). On the CPU and MIC, this has relatively low impact (2%) on performance. Performing diffuse Phong illumination per sample incurs greater cost, but overall costs only about roughly 5–10% more than unlit DVR.

5.5. Multi-material selection and classification

An advantage of direct RBF volume rendering is the ability to construct multi-field volume data from a single source of particles on the fly. From that, one can classify separate fields with separate transfer functions. To accomplish this, we evaluate and classify two (or more) samples at each position, and blend them sequentially. This enables us to understand which basis functions are responsible for which regions of one original scalar field, allowing for classification of different atoms and molecules (in computational chemistry) or halos (astrophysics). A simple example is shown in Figure 4. In a more complex example in Figure 5, we assigned a separate transfer functions to an entire molecule (tryptophan in

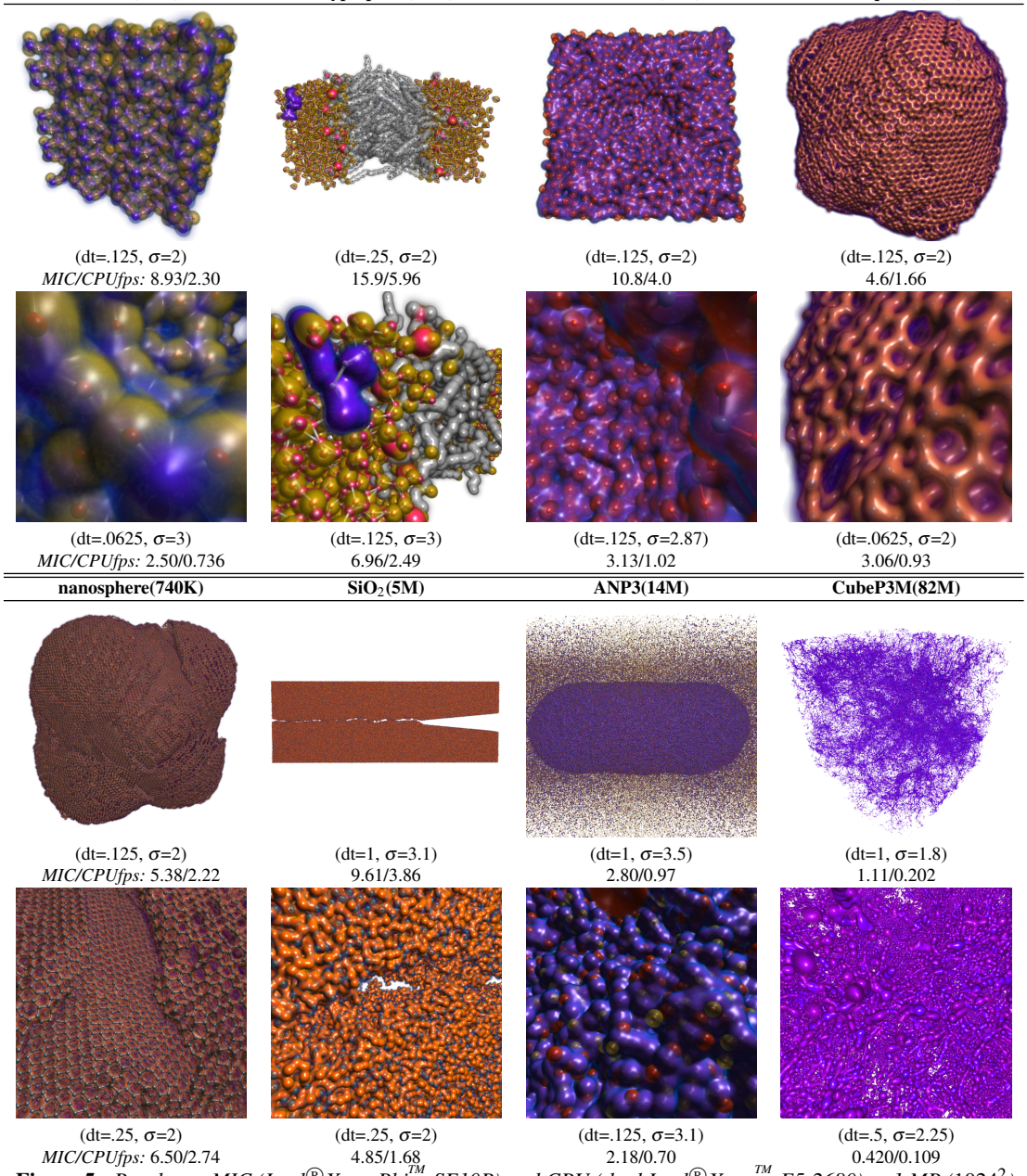
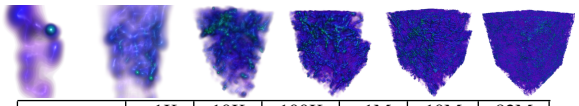


Figure 5: *Results on MIC (Intel® Xeon Phi™ SE10P) and CPU (dual Intel® Xeon™ E5-2680) at 1 MP (1024^2).*

Dataset	zeolite	trypt.	nanobowl	ns.90k	ns.740k	SiO_2	ANP3	CubeP3M
num. particles	3494	6830	21K	92K	742K	4.8M	14.7M	82M
size/timestep (MB)	.13	.33	0.8	3	40	160	950	2624
geometry size (MB)	.25	.49	0.7	6	52	130	504	3133
BVH size (MB)	.16	.339	0.5	4	34	160	430	2056
BVH build time* (s)	.0256	.057	0.081	0.91	7.5	50	128	541
#fields	2	4	2	1	1	2	3	1
str.vol. size** (MB)	.379	4.8	1.5	5.3	55	806	6036	-
str.vol. build time** (s)	.031	.026	.6	1.1	4.1	80	205	-

Table 1: *Data set statistics. (*)Single CPU thread (Xeon E5-2680). (**) 16 CPU threads. 32-bit float data, 1 voxel/Ångström.*



#particles	1K	10K	100K	1M	10M	82M
fps-fix σ	10.1	4.00	2.48	3.82	3.81	2.12
fps-var σ	2.62	2.10	2.40	2.25	2.22	2.12
σ	6.61	5.44	3.67	3.39	2.74	2.05

Figure 6: *Performance with respect to data size, using subsets of the 82 million particle CubeP3M astrophysics data rendered with $dt = 1$ and varying σ. The table shows performance with both fixed and varying σ, see Figure 7(left).*

blue, carbon lipids in white). With structured volumes, this requires more costly computation, storage and rendering of separate volumetric fields. As shown in Figure 1, the cost of computing and sampling multifield structured volume data is high, giving our method performance and memory advantages even for two fields. To implement this, we must store these additional samples in `Phi_buf`. Since our buffer is maintained per-thread, the cost per separate field is not prohibitive. A buffer with (64 / M) samples per ray for M fields works well on both CPU and MIC.

6. Results

We conducted results on a visualization node of Stampede with dual 8-core (16 cores total) 2.7 GHz Intel® Xeon™ E5-2680 with 32 GB RAM, an Intel® Xeon Phi™ SE10P with 61 cores at 1.1 GHz with 8 GB RAM, and an NVIDIA K20 (Kepler) GPU with 6 GB RAM. All computations were carried out in single-precision floating point. On the CPU, we used the 8-float AVX instruction set. Unless stated otherwise, performance numbers are measured at 1024^2 (1 MP) resolution using the Xeon Phi™. Units for all molecular data are Ångströms; CubeP3M uses an intrinsic unit cube corresponding roughly to one particle.

Overall performance and quality. In Figure 5, we examine eight datasets ranging in memory footprint from 250K to 2.6 GB. Statistics on these data and BVH are given in Table 1. To explore the potential uses of our algorithm, and in particular selection (Sec. 5.5) we benchmark scenes using multi-field transfer functions at sampling rates and σ delivering a converged image, but not guaranteeing interactive performance. Generally, performance falls in the 1–20 fps range on the MIC and 0.5–10 fps range on the CPU. From a quality perspective, RBF volume rendering enables reconstructions at least as smooth as cubic B-spline filtering of structured data [SH05] (Figure 1, lower right), but at far lower cost, and with better preservation of features.

Data size and number of samples. Like BVH-accelerated structured DVR [KTW*11], RBF DVR performance depends more on the number (and cost) of volume samples than on the total number of particles. In Figure 5, the smallest (zeolite) and largest (CubeP3M) data sets are only a factor of 5 apart in close-up frame rate, despite 4 orders of magnitude difference in number of particles. In Figure 6, we consider subsets of the CubeP3M data. When using a fixed $\sigma = 2.05$, performance fluctuates from 2 to 10 fps. However, when we allow σ to vary (increasing it until we achieve an

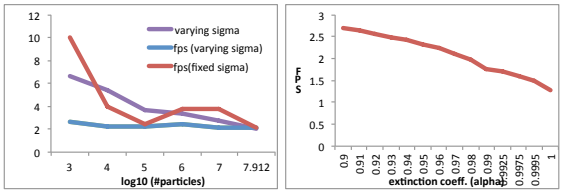


Figure 7: *Plots from Fig. 6. **Left:** Performance vs. number of particles. **Right:** Impact of early termination.*

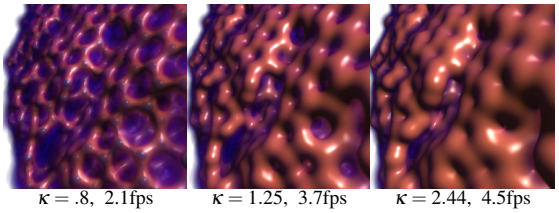
artifact-free image), we notice that performance is almost exactly constant: for a *converged* image, σ decreases linearly with the logarithm of data size (Figure 7, left).

To gauge the relationship between performance and number of samples, in Figure 7 (right) we examine how early termination impacts frame rate using the 82M CubeP3M scene as reference. As expected, performance drops significantly as α_{ext} approaches 1. Results at the default α_{ext}=0.97 are indistinguishable from larger values and yield nearly twice the performance of α_{ext}=1.

Effects of scale and truncation parameters. In Figure 8 we show the effects of modifying κ. With this type of preclassification, the resulting filter remains smooth, giving the user significant control over reconstruction behavior. Since κ has the effect of changing width as well as height of Gaussians, it is an effective mechanism for modeling as well as classification of structure.

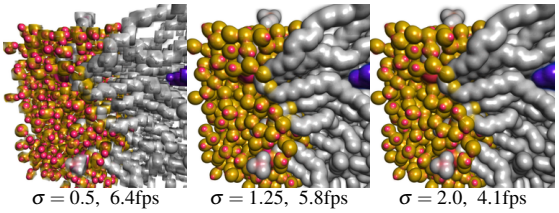
Figure 9 illustrates varying truncation radius σ. Larger σ increases the width of BVH bounds – this is factored in at traversal time, and modifying σ does not require rebuilding the data structure. Generally, higher σ results in worse performance. For most atomistic data, a truncation radius of $\sigma = 2$ Ångströms is sufficient to show features within the van der Waals radius. Increasing κ increases the width of Gaussians, which requires an increase in σ.

Comparison with Nanovol on the GPU. In Table 2, we benchmark our RBF method using the transfer functions from [KWN*13], and compare performance of RBF DVR to structured volume rendering in bnsView and both structured and RBF DVR in Nanovol [RKN*13].



$\kappa = .8$, 2.1fps $\kappa = 1.25$, 3.7fps $\kappa = 2.44$, 4.5fps

Figure 8: *Varying κ increases both RBF value* and *width.*



$\sigma = 0.5$, 6.4fps $\sigma = 1.25$, 5.8fps $\sigma = 2.0$, 4.1fps

Figure 9: *σ controls RBF truncation width, impacting performance and reconstruction quality.*

Dataset		nanobowl	ns.90k	ns.740k	SiO_2	ANP3
structured*	bnsView - MIC	36 / 22	12.4 / 14.8	9.98 / 14.1	20.3 / 10.7	1.18 / 14.1
	bnsView - CPU	6.15 / 4.02	2.42 / 2.97	1.57 / 2.46	4.51 / 2.02	.35 / 1.91
	nanovol - GPU	41 / 32.5	19.5 / 26	6 / 10.7	19.6 / 20.9	2.50 / 17.3
RBF	bnsView - MIC	9.7 / 6.9	8.8 / 10.2	6.25 / 7.66	6.44 / 5.30	2.85 / 3.91
	bnsView - CPU	3.65 / 2.53	3.8 / 4.0	2.93 / 3.58	2.27 / 2.03	1.06 / 1.07
	nanovol - GPU	2.42 / 3.10	0.71 / .65	0.48 / 0.31	1.03 / 0.303	0.83 / 1.0
	RBF / structured (MIC)	**.27x / .31x**	**.71x / .68x**	**.62x / .54x**	**.31x / .50x**	**2.4x / .28x**
	RBF / structured (CPU)	**.60x / .63x**	**1.57x / 1.34x**	**1.86x / 1.45x**	**.5x / 1x**	**3x / .56x**
	bnsView (MIC) / nanovol (GPU) – RBF	**4x / 2.2x**	**12.4x / 15.7x**	**13x / 24.7x**	**6.4x / 17.5x**	**3.43x / 3.91x**

Table 2: *Frame rates (far/close) of reference scenes benchmarked in bnsView and Nanovol, using the reference transfer functions from [KWN*13] and a fixed step size of dt=0.5. (*Structured numbers from [KWN*13]).*

Nanovol is a GLSL-based raycaster employing a uniform grid for acceleration and RBF evaluation. Though different from our BVH method, it is similar in functionaliy and a good candidate for comparison. Matching RBF parameters, camera and transfer function as closely as possible, our algorithm in bnsView outperforms Nanovol's RBF method on average by 10x. While Nanovol's implementation could potentially be improved, this suggests bnsView is competitive with current GPU approaches.

RBF and structured DVR, on CPU and MIC. Table 2 also compares RBF volume rendering with the structured volume rendering results of [KWN*13]. On MIC, RBF volume rendering is roughly 33% slower than structured DVR (with lighting, for 1-voxel-per-Ångström data). The performance gap is less significant for far views of larger data (ANP3). We note that on the CPU, RBF DVR is consistently faster than structured DVR by 2x. This is likely due to the lack of a hardware gather instruction on the CPU. Although the MIC overall 2x – 5x faster than the 16-core SandyBridge CPU (Table 2), bnsView is still highly usable on CPUs in general; we were able to run all test scenes semi-interactively at 512^2 resolution on a 4-core IvyBridge laptop.

Memory consumption and preprocess time. As seen in Table 1, lower memory consuption is a major advantage of this technique. Precomputed structured volumes are costly; even megascale particle data (SiO_2, ANP3) can generate gigascale volume data requiring minutes to generate. While the BVH incurs overhead (comparable to the original data size), it is possible to use a coarser BVH at some cost in performance. Moreover, construction could likely be achieved in real-time for gigascale datasets using the BVH builders in Embree [WWB*14].

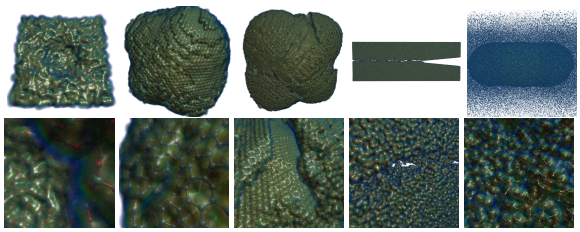


Figure 10: *Reference scenes from [KWN*13], rendered with a fixed dt of 0.5, using a standard 1D heatmap transfer function, lighting enabled. Refer to frame rates in Table 2.*

7. Conclusion

We have presented a new algorithm for efficient RBF volume rendering on CPU and MIC architectures. It performs competitively with the best-known GPU approaches, enables better image quality at lower memory and preprocessing costs, and is not significantly slower (and sometimes faster) than structured volume rendering. We achieve interactive or close performance for volume rendering our largest data sets without relying on LOD or subsampling. Multi-material classification, enabled by this technique, brings new capabilities to volume visualization.

In future work, we wish to accelerate our currently non-parallel BVH build by leveraging Embree [WWB*14]. To better evaluate IVL and ISPC as programming models, we wish to compare performance with OpenMP implementations of both SOA and non-SOA (non-packetized, single ray) methods. Lastly, we wish to extend this technique to large distributed data in parallel, and explore in situ and in transit visualization.

Acknowledgments

This work was funded in part by National Science Foundation grants OCI-1134872 and ACI-1339863, and under the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. We thank Lei Cheng, Maria Chan and Kah Chun Lau of Argonne National Laboratory for the zeolite, nanobowl and nanosphere data; Alfredo Cardenas and Ron Elber at University of Texas at Austin for the tryptophan; Ken-ichi Nomura and Priya Vashishta of Univeristy of Southern California for the SiO_2 and ANP3 data; and Anson d'Aloisio and Paul Shaprio of the University of Texas at Austin for the CubeP3M SPH. Additional thanks go to Mike Packard at TACC and Brian Dietrich at Intel.

References

[ACP08] ALTAY G., CROFT R. A., PELUPESSY I.: SPHRAY: a smoothed particle hydrodynamics ray tracer for radiative transfer. *Monthly Notices of the Royal Astronomical Society 386*, 4 (2008), 1931–1946. 3

[BPL*12] BROWNLEE C., PATCHETT J., LO L.-T., DEMARLE D., MITCHELL C., AHRENS J., HANSEN C. D.: A study of ray tracing large-scale scientific data in two widely used parallel visualization applications. In *Eurographics Symposium on Parallel Graphics and Visualization* (2012), The Eurographics Association, pp. 51–60. 4

[BSP06] BIGLER J., STEPHENS A., PARKER S. G.: Design for parallel interactive ray tracing systems. In *Interactive Ray Tracing 2006* (2006), pp. 187–196. 4, 5

[CRZP04] CHEN W., REN L., ZWICKER M., PFISTER H.: Hardware-accelerated adaptive EWA volume splatting. In *Proceedings of the conference on Visualization'04* (2004), IEEE Computer Society, pp. 67–74. 2

[DCH88] DREBIN R. A., CARPENTER L., HANRAHAN P.: Volume rendering. In *ACM Siggraph Computer Graphics* (1988), vol. 22, ACM, pp. 65–74. 2

[DRGI08] DOLAG K., REINECKE M., GHELLER C., IMBODEN S.: Splotch: visualizing cosmological simulations. *New Journal of Physics 10*, 12 (2008), 125006. 3

[FAW10] FRAEDRICH R., AUER S., WESTERMANN R.: Efficient high-quality volume rendering of SPH data. *Visualization and Computer Graphics, IEEE Transactions on 16*, 6 (2010), 1533–1540. 1, 2, 3, 4, 5

[FSW09] FRAEDRICH R., SCHNEIDER J., WESTERMANN R.: Exploring the millennium run: scalable rendering of large-scale cosmological datasets. *Visualization and Computer Graphics, IEEE Transactions on 15*, 6 (2009), 1251–1258. 3

[GBM*12] GROTTEL S., BECK P., MULLER C., REINA G., ROTH J., TREBIN H.-R., ERTL T.: Visualization of electrostatic dipoles in molecular dynamics of metal oxides. *IEEE TVCG 18*, 12 (2012), 2061–2068. 3

[GIK*07] GRIBBLE C. P., IZE T., KENSLER A., WALD I., PARKER S. G.: A coherent grid traversal approach to visualizing particle-based simulation data. *Visualization and Computer Graphics, IEEE Transactions on 13*, 4 (2007), 758–768. 3

[JBL*06] JANG Y., BOTCHEN R. P., LAUSER A., EBERT D. S., GAITHER K. P., ERTL T.: Enhancing the interactive visualization of procedurally encoded multifield data with ellipsoidal basis functions. In *Computer Graphics Forum* (2006), vol. 25, Wiley Online Library, pp. 587–596. 2

[JFSP10] JANG Y., FUCHS R., SCHINDLER B., PEIKERT R.: Volumetric evaluation of meshless data from smoothed particle hydrodynamics simulations. In *Proceedings of the 8th IEEE/EG international conference on Volume Graphics* (2010), Eurographics Association, pp. 45–52. 1, 2, 4

[JV09] JANG Y., VARETTO U.: Interactive volume rendering of functional representations in quantum chemistry. *Visualization and Computer Graphics, IEEE Transactions on 15*, 6 (2009), 1579–5186. 2

[JWH*04] JANG Y., WEILER M., HOPF M., HUANG J., EBERT D., GAITHER K., ERTL T.: Interactively visualizing procedurally encoded scalar fields. In *VisSym* (2004), pp. 35–44. 2, 4

[KAH07] KÄHLER R., ABEL T., HEGE H.-C.: Simultaneous GPU-assisted raycasting of unstructured point sets and volumetric grid data. In *Proceedings of the Sixth Eurographics/Ieee VGTC conference on Volume Graphics* (2007), Eurographics Association, pp. 49–56. 3

[KCL*13] KNOLL A., CHAN M., LAU K., LUI B., GREELEY J., CURTISS L., HERELD M., PAPKA M.: Uncertainty classification and visualization of molecular interfaces. *International Journal of Uncertainty Quantification 3*, 2 (2013), 157–169. 3

[KTW*11] KNOLL A., THELEN S., WALD I., HANSEN C. D., HAGEN H., PAPKA M. E.: Full-resolution interactive CPU volume rendering with coherent BVH traversal. In *Pacific Visualization Symposium (PacificVis)* (2011), pp. 3–10. 4, 8

[KWN*13] KNOLL A., WALD I., NAVRÁTIL P. A., PAPKA M. E., GAITHER K. P.: Ray tracing and volume rendering large molecular data on multi-core and many-core architectures. In *Proceedings of the 8th International Workshop on Ultrascale Visualization* (2013). 2, 4, 6, 8, 9

[LHW12] LEISSA R., HACK S., WALD I.: Extending a C-like language for portable SIMD programming. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming* (2012), ACM, pp. 65–74. 2, 4, 6

[Max95] MAX N.: Optical models for direct volume rendering. *Visualization and Computer Graphics, IEEE Transactions on 1*, 2 (1995), 99–108. 3

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Ann R Astronomy and Astrophysics 30* (1992), 543–574. 2

[NJB07] NAVRÁTIL P. A., JOHNSON J. L., BROMM V.: Visualization of cosmological particle-based datasets. *IEEE TVCG 13*, 6 (2007), 1712–1718. 3

[OKK10] ORTHMANN J., KELLER M., KOLB A.: Topology-caching for dynamic particle volume raycasting. In *Proceedings of Vision, Modeling and Visualization 2010, Siegen, Germany* (2010), Eurographics, pp. 147–154. 1, 2, 3, 5

[PM12] PHARR M., MARK W.: ISPC: a SPMD compiler for high-performance CPU programming. *Proceedings of Innovative Parallel Computing (InPar)* (2012). 4

[Pri07] PRICE D. J.: Splash: An interactive visualisation tool for smoothed particle hydrodynamics simulations. *Publications of the Astronomical Society of Australia 24*, 3 (2007), 159–173. 3

[RKN*13] REDA K., KNOLL A., NOMURA K., PAPKA M., JOHNSON A., LEIGH J.: Visualizing large-scale atomistic simulations in ultra-resolution immersive environments. In *IEEE Symposium on Large Scale Data Analysis and Visualization (LDAV)* (2013), pp. 59–65. 1, 2, 4, 8

[Sam90] SAMET H.: *The design and analysis of spatial data structures*, vol. 199. Addison-Wesley Reading, MA, 1990. 4

[SH05] SIGG C., HADWIGER M.: Fast third-order texture filtering. *GPU gems 2* (2005), 313–329. 8

[SHUS10] STONE J., HARDY D., UFIMTSEV I., SCHULTEN K.: GPU-accelerated molecular modeling coming of age. *Journal of Molecular Graphics and Modeling 29*, 2 (2010), 116–125. 3

[TOP13] TOP500.ORG: Architecture Share, November 2013. 1

[TSO*11] TURK M. J., SMITH B. D., OISHI J. S., SKORY S., SKILLMAN S. W., ABEL T., NORMAN M. L.: yt: A multi-code analysis toolkit for astrophysical simulation data. *The Astrophysical Journal Supplement Series 192*, 1 (2011), 9. 3

[WBS07] WALD I., BOULOS S., SHIRLEY P.: Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics (TOG) 26*, 1 (2007), 6. 4, 6

[WFKH07] WALD I., FRIEDRICH H., KNOLL A., HANSEN C. D.: Interactive isosurface ray tracing of time-varying tetrahedral volumes. *Visualization and Computer Graphics, IEEE Transactions on 13*, 6 (2007), 1727–1734. 4

[WMW86] WYVILL G., MCPHEETERS C., WYVILL B.: Data structure for soft objects. *The Visual Computer 2*, 4 (1986), 227–234. 3

[WSBW01] WALD I., SLUSALLEK P., BENTHIN C., WAGNER M.: Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum (Proceedings of EUROGRAPHICS) 20*, 3 (2001), 153–164. 4

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree–A Ray Tracing Kernel Framework for Efficient CPU Ray Tracing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)* (2014). (to appear). 4, 9

[ZPvBG01a] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: EWA volume splatting. In *Visualization, 2001. VIS'01. Proceedings* (2001), IEEE, pp. 29–538. 2

[ZPVBG01b] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM, pp. 371–378. 3