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

Volume 33 (2014), Number 3

Fast RBF Volume Ray Casting on CPU and MIC

online paper id 268

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 CPU's and accelerators. In this paper, we present a fast 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 80 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 preprocessing other than the BVH. Lastly, our technique enables multi-field, multi-material classification of atoms or particles for improved representation 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 flexibly reconstructs, classifies and shades any continuous scalar field, enabling better insight than surface-based visualization. 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 texture 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 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. More recent GPU implementations, e.g. using a grid acceleration structure [RKN*13] have proven interactive on current GPU's, but slow compared to 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 frame-

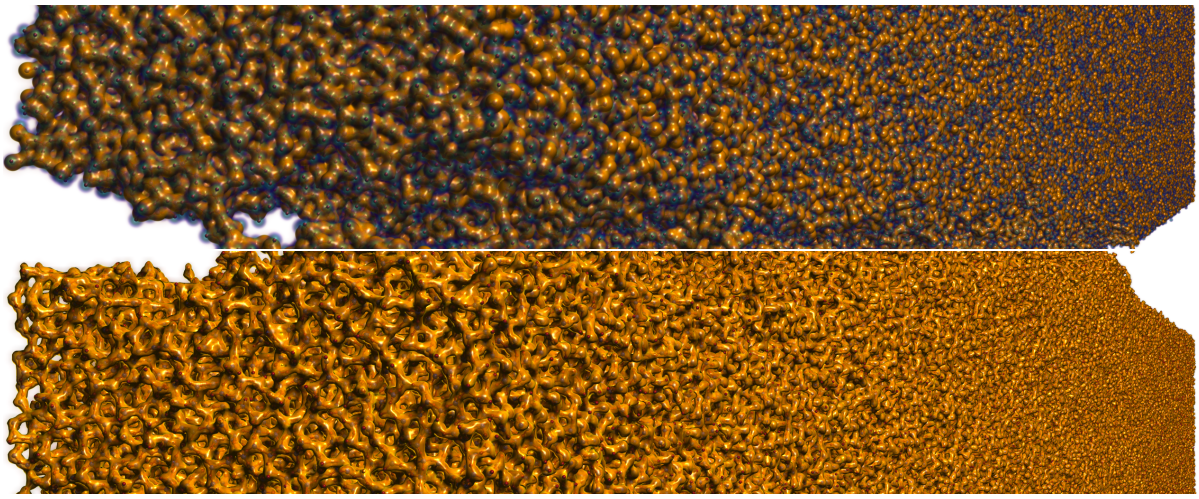

Figure 1: *5 million atom molecular dynamics glass (SiO_2) fracture data, rendered on a SE10P Xeon Phi in Stampede at* 2560×1024. ***Top:*** *RBF volume rendering and ball-and-stick glyphs using 2 transfer functions to separately classify silicon and oxygen atoms as multiple fields (1.35 fps), rendered directly from the BVH of the ball-and-stick data using our technique.* ***Bottom:*** *a single-field precomputed structured dataset (2 voxels per Angstrom, 1.5 GB) required 2 minutes to precompute, exhibits noticeable loss in reconstruction quality and renders at almost exactly the same speed (1.38 fps).*

work 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 integration 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.

The paper is organized as follows: Section 2 discusses state-of-the-art in particle visualization; Section 3 covers theoretical background; Section 4 describes our method; Section 5 describes implementation on the CPU and MIC; Section 6 details results and use cases; and we conclude in Section 7.

2. Related Work

Direct volume rendering, or DVR, [DCH88] is a process of directly rendering a 3D scalar field by reconstructing 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 reconstruction. The same general method can in fact be used for other particle data, for example atomistic data from molecular dynamics. We refer to this more general technique as *RBF volume rendering*.

2.1. Volume rendering of point data

RBF volume rendering is costly, and interactive applications have generally been limited to the GPU. 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 [JFSP10] a kd-tree was used instead of an octree to improve bound tightness, however performance remained sub-interactive. 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 up to 42M particles (0.1 fps). This approach likely remains state-of-the-art, and would be faster still on current GPU's. 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. 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. Volume rendering of point data has also been proven on the GPU using even more expensive kernels than RBFs [LGM*08,NLKH12].

2.2. 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 surface 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 for 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.

2.3. 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. Yt [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 in VTK 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:

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

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

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

where k_i is the value of the kernel (e.g. density), and r_i is the radius of the Gaussian (e.g. covalent radius for molecular data). We truncate at a support radius of σr_i, defaulting to $\sigma = 2$; this can be changed by the user dynamically.

3.2. Volume rendering integral

Volume rendering is a special case of the light transport equation [Kaj86], in which (emissive) irradiance integrated over t along a ray, for the scalar field Φ and transfer function with color $\mathbf{C}$ and opacity α, is given as:

$$I(t) = \int \mathbf{C}(\Phi(t))\alpha(\Phi(t))\, \mathrm{e}^{-\int_a^t \alpha(\Phi(s))ds} dt \tag{3}$$

This is integrated numerically by blending, as sketched in Listing 1, where `dt` is the sampling step size.

Listing 1: *DVR integration*

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

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 4:

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 samples that overlap that particle

In the direct approach the scalar field is evaluated per-sample, which easily lends itself trivially to volume rendering. Reconstructing $\Phi(\mathbf{x})$ requires determining all kernels whose support overlaps $\mathbf{x}$. This problem is commonly referred to as region-finding [Sam90], and costs $O(Pk\ N)$ 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. Pseudocode is given in Listing 2.

Listing 2: *Direct (per-sample) method*

```
1  foreach ray
2    foreach sample t in {tenter..texit}
3      vec3f p = ray.org + ray.dir * t
4      {i} = find_neighbors(p)
5      foreach particle i in {i}
6        if (i contains p)
7          Phi += phi(i,p)
8      dvr_integrate(Phi,ray,tf)
```

Listing 3: *Proxy (per-particle) method*

```
1  foreach particle i in tree
2    foreach sample s in Phi_grid overlapped by i
3      vec3f p = Phi_grid.get_vertex(s)
4      Phi_grid[s] += phi(i,p)
5  //render grid with DVR
6  foreach ray
7    foreach sample t in {tenter..texit}
8      dvr_integrate(Phi_grid[t],ray,tf)
```

In the proxy, or per-particle method, each particle is evaluated only once for all samples that it overlaps. This approach is taken by all methods precomputing a structured volume, e.g. the dynamic grid method of Fraedrich et al. [FAW10]. An acceleration structure can also be traversed to cull particles outside a view frustum or perform level-of-detail simplification. The obvious 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 the per-particle approach is the memory required to store the proxy (in many cases larger than the original RBF data) and worst-case time required to compute it (linear time for the selected particle data or grid, whichever is greater). An example for a precomputed uniform grid, and subsequent volume rendering is sketched in Listing 3.

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. It enables efficient use of SIMD vector instructions, with each

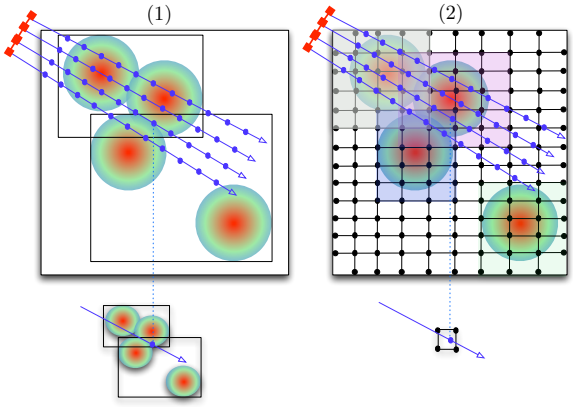

Figure 2: *Options for RBF field reconstruction. (1) Direct method: the field is evaluated directly at each sample along the ray, which can prove expensive. (2) Proxy method: the field is resampled into a grid, and then inexpensively reconstructed from the proxy, at the cost of memory and/or quality.*

ray mapped to a separate SIMD lane and ray data laid out in structure-of-arrays (SOA) fashion. In particular for traversal and intersection, the cost of memory access is amortized over the number of rays in the packet. Coherent ray tracing methods has 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 likely 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 with coherent BVH traversal, 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. With SPMD languages such as IVL [LHW12] and ISPC [PM12], it is possible to write coherent BVH ray tracers (e.g. Embree [WBW13]) for multiple SIMD hardware backends, including the Xeon Phi.

Our application is built on top of bnsView [KWN*13], a molecular visualization tool written in the IVL language that achieves GPU levels of performance on CPU and MIC architectures for ray tracing ball-and-stick 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, similar to the algorithm in Listing 2. 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, it was apparent that a new algorithm would be 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 coherent BVH traversal to traverse and compute samples in a RBF field *per packet* instead of per-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 (Listing 3), but maintaining only a small buffer of samples with little overhead and the exact same quality as the direct ray casting approach (Listing 2). Pseudocode for this algorithm is sketched in Listing 4, and the concept is illustrated in Figure 3.

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

- by computing samples *per-packet* versus per-frame, we require less memory and can store and integrate all samples required for full ray casting, i.e. without subsampling
- with *coherent BVH traversal*, we ensure a particle is traversed once for each packet, not once per sample, improving memory access patterns. We are simultaneously able to skip space and scale to larger data with no LOD.
- using *object-decomposition* (the BVH) instead of spatial decomposition (a grid [RKN*13], kd-tree or octree), we can efficiently use the same structure for both RBF volume rendering and ball-and-stick ray tracing, without traversing duplicate objects/RBFs.
- 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*

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).

Listing 4: *RBF DVR with coherent BVH traversal*

```
1  void rbf_dvr(Packet packet, float tenter, float texit){
2   foreach bvh leaf i intersected by packet
3     foreach t in {tenter.. texit}
4       Phi_buffer[t] += phi(i,packet.org + packet.dir * t)
5   foreach t in {tenter..texit}
6     dvr_integrate(Phi_buffer[t],packet,tf)
7     if (packet.color.a > .99) break //early termination
8  }
```

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 reconstructs Φ for samples along each ray, and integrates these samples front to back (Equation 3).

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 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_buffer` with `K` samples for each ray
4. Traverse the BVH, summing ϕ for every leaf at every sample (Equation 1) and storing that in `Phi_buffer`.
5. Integrate `Phi_buffer` 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.

Listing 5: *Multi-pass algorithm*

```
1  void rbf_dvr_multipass(){
2    foreach packet
3      {first_t, last_t} = AABB_test(packet, bvh.bounds)
4      for (t = first_t; t < last_t; t += Phi_buffer.size){
5        rbf_dvr(packet, t, t + Phi_buffer.size)
6        if (packet.color.a > .99) break
7      }
8    }
```

4.2. Multi-pass algorithm with early termination

On the CPU and MIC, dynamically allocating a single large `Phi_buffer` 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 putting Listing 4 in a loop, as sketched in Listing 5. For larger data in particular, 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 implementation of this 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:

- Relatively large memory per core (for storing the buffer)
- SIMD vector units, and explicit mechanisms for control flow both inside and outside of SIMD lanes (to ensure coherent traversal and better memory access)

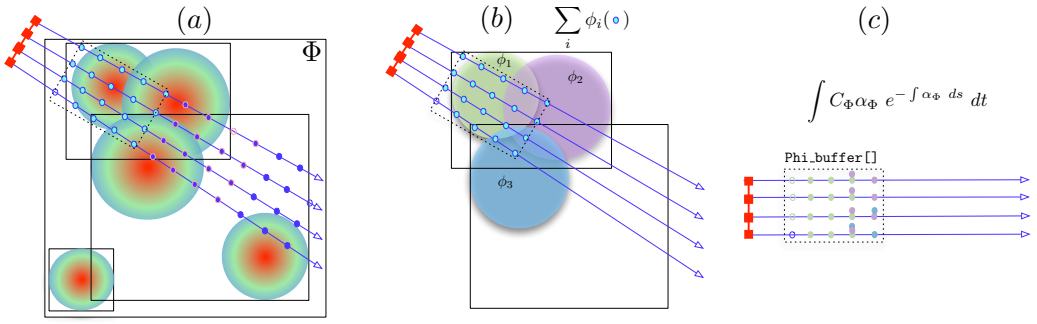

Figure 3: *Coherent RBF algorithm. (a) Iterate through samples along a packet, depositing into a fixed-size buffer. (b) Each BVH leaf node (particle) adds its value to all samples it overlaps in the buffer. (c) When this buffer has finished traversal, it is integrated front-to back using Equation 3, and we proceed to the next set of samples.*

- Encouragement of large, multi-function kernels designed to operate independently across separate hardware 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 (different versions of SSE, 8-float AVX, 16-float MIC, and potentially ARM NEON and BlueGene/Q intrinsics). For this, we employ the IVL SPMD compiler [LHW12] and the RIVL framework.

5.1. IVL SPMD language

BnsView is a module for RIVL, a ray tracing engine written in the IVL SPMD language [LHW12], a research compiler which is a close relative of the open-source Intel SPMD Program Compiler (ISPC) [PM12] maintained by Intel. As with other SPMD languages on the GPU, IVL kernels are written in scalar form and execute in parallel across many threads. Unlike GPU languages, the IVL compiler generates C++ code, with data automatically laid out in structure-of-arrays (SOA) format for efficient use of SIMD. In both IVL and ISPC, the programmer defines variables as `uniform` or `varying`. Semantically, this determines whether they are scalar or vector quantities, respectively. SPMD control flow within each thread is handled automatically by the compiler, which emits standard (C++) conditionals for `uniform` variables and SIMD masks for `varying` quantities. Like ISPC, IVL has support for multiple SIMD backends (4-float SSE, 8-float AVX, 16-float Xeon Phi, and a "generic" backend for other architectures), enabling performance and functionality on a wide variety of CPU and MIC hardware. As a research prototype, IVL offers several advantages over ISPC (full C++ class inheritance, operator overloading, embedded C++ code). Both ISPC and IVL offer 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 using the publicly-available Embree [WBW13] framework.

5.2. 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.

5.3. 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 bounding volume herachy. This stores a single opaque hit position `t_hit` along the ray and primitive ID.

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.

5.4. Coherent RBF DVR in IVL

IVL implementation of our method is very similar to the pseudocode in Listing 4. In SPMD, a packet is simply a `varying Ray`, traversed with a uniform stack and pointers for all SIMD lanes. Otherwise, the differences between our algorithm and coherent BVH traversal [WBS07] (IVL implementation of which is described in [KWN*13]) are:

- multiple traversal passes, as described in Section 4.2
- dynamically expanding extents of every BVH node by σ, the support 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_buffer`
- when BVH traversal completes (and for each pass), DVR integration is performed on these buffered samples.

5.5. 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:

$$\nabla\Phi(\mathbf{x}) = -2\Phi(\mathbf{x})(\mathbf{x} - \mathbf{x_i}) \tag{4}$$

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 and Phong shading per sample incurs greater cost, but overall costs only about roughly 5–10% more than unlit DVR.

5.6. Multi-material selection and classification

An advantage of direct RBF volume rendering the ability to construct multi-field volume data from a single source of particles. One can classify separate field using separate transfer functions, and blend them as separate samples at the same position. This flexibility 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 (cosmology, astrophysics). Examples are shown in Figure 5. In the "tryptophan" example we have assigned separate transfer functions to a molecule (blue tryptophan), and atoms (white carbon lipids, and orange oxygen at their tips). This lets users "classify" molecules more effectively than using standard 1D transfer functions. In a structured volume framework, e.g. [RKN*13] this sort of classification would require construction, storage and rendering of separate precomputed volumes. Similarly, dynamic proxies (e.g. [FAW10]) require additional storage for each field rendered. Since our internal buffer is small (per-thread), this additional cost is not prohibitive, even for 4 or more fields. We find that a buffer with (64 / M) samples per ray, for M fields, works well on both CPU and MIC.

6. Results

To evaluate our implementation, we conducted the following benchmarks using a 1024^2 frame buffer 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.

6.1. 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. 5.6) we opted to benchmark scenes using high- quality multi-field transfer functions that require high sample rates but do not guarantee interactive performance. Generally, performance falls in the 1-20 fps range on the MIC and .5-5 fps range on the CPU. As in [KTW*11], performance depends more on the number of volume samples than on the number of particles in the screen. 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.

From a quality perspective, RBF volume rendering enables filter quality at least as good as cubic B-spline reconstruction of structured data [SH05], at a cost similar to (lit) structured volume rendering on CPU or MIC. As always with volume rendering, the sampling rate and correct parameters required to render without artifacts depend on the data and transfer function. Taking into account the multifield classification options and lower memory and preprocess requirements, we claim this method is worth the performance hit over structured volume rendering. Morever, even at a relatively fine resolution (1 voxel per Angstrom) there are clear differences between structured and particle DVR (Figure 1).

6.2. Comparisons with Nanovol on the GPU and structured DVR

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]. Nanovol is a GPU raycaster implemented in the OpenGL shading language (GLSL). It employs a uniform grid for acceleration and RBF reconstruction, which is different from the algorithm we have chosen. Nonetheless, it was the only available GPU implementation we had access to, and the best comparison we can make while noting its limitations. Although we could not match parameters in our RBF systems perfectly, using similar transfer functions and the same camera parameters our algorithm outperforms the grid-based RBF reconstruction in Nanovol on the GPU, on average by 10x. While we believe it would be possible to improve on the Nanovol GPU RBF implementation, fast RBF volume rendering has historically proven challenging on GPU's.

Also from Table 2, comparing our method to structured DVR (with lighting, of 1-voxel-per-Angstrom data), RBF volume rendering is slower on MIC (1.5x) but not significantly so. The performance gap is less significant for larger data, with RBF volume rendering actually outperforming structured volumes rendering. On the CPU, likely due to lack of gather, RBF DVR is in fact consistently faster than structured DVR by 2x. Although the MIC is 2x – 5x faster than the 16-core SandyBridge CPU, bnsView is still highly usable on the CPU – we were able to run all test scenes except CubeP3M

Dataset	zeolite	trypt.	nanobowl	ns.90k	ns.740k	SiO_2	ANP3	CubeP3M
num. particles	3494	6830	21K	92K	742K	4.8M	14.7M	81.4M
data size per 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

Table 1: *Data sets in Figure 5, size and preprocess statistics. *Build times are on one thread of Stampede.*

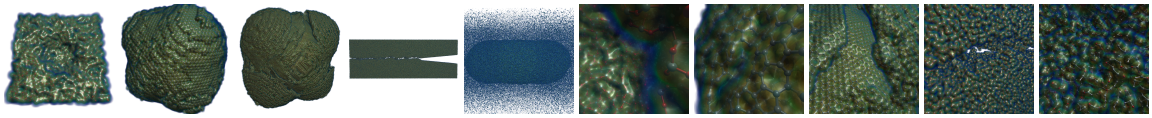

Figure 4: *Reference scenes rendered with a fixed dt of 0.5, using a standard 1D heatmap transfer function, with lighting enabled. The left and right five images are far and close reference scenes, respectively. Refer to frame rates in Table 2.*

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]).*

semi-interactively at 512^2 resolution on a 4-core IvyBridge laptop.

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. Moreover, we were able to achieve interactive or close performance for volume rendering our largest data sets (up to 82M particles) without relying on level of detail methods or subsampling. Multi-material selection (Section 5.6) is a major advantage of this technique, and in particular could draw chemistry users that have not previously considered volume rendering as part of their workflow.

In the near term, we see our system being deployed for in-transit visualization on supercomputers such as Stampede. MIC is a new architecture that not all codes can yet leverage; this presents opportunities for co-visualization and co-analysis software that can make effective use of the resource. We have already used this method for remote in-transit visualization of molecular dynamics computations (the tryptophan model in Figure 5) in process-parallel, running and visualizing 50 simulations at the same time as an ensemble. Live visualization paradigms such as this could help computational steering. Generally, we hope to continue our work in portable C++-based SPMD frameworks such as IVL and ISPC, and develop codes that benefit a wide variety of HPC users on predominately CPU architectures.

In future work, we wish to improve our currently non-parallel BVH build. The Intel Embree 2.0 framework [WBW13] offers fast on-the-fly builders [Áfr12] in a ray tracing framework similar to ours. In addition, we would like to extend our technique in data-distributed parallel, address larger data than would be effective on a single node. Lastly, we are interested in applications of the multi-atom classification techniques we have begun to explore in this paper.

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

[Áfr12] ÁFRA A. T.: Incoherent ray tracing without acceleration structures. In *Eurographics (Short Papers)* (2012), pp. 97–100. 8

[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, 6

[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. 3

[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, 7

[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. 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

[Kaj86] KAJIYA J. T.: The rendering equation. In *ACM SIGGRAPH Computer Graphics* (1986), vol. 20, pp. 143–150. 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, 7

[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), ACM, p. 5. 2, 4, 6, 7, 8

[LGM*08] LEDERGERBER C., GUENNEBAUD G., MEYER M., BACHER M., PFISTER H.: Volume MLS ray casting. *IEEE TVCG 14*, 6 (2008), 1372–1379. 2

[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

[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

[NLKH12] NELSON B., LIU E., KIRBY R. M., HAIMES R.: Elvis: A system for the accurate and interactive visualization of high-order finite element solutions. *Visualization and Computer Graphics, IEEE Transactions on 18*, 12 (2012), 2325–2334. 2

[PM12] PHARR M., MARK W.: ispc: A SPMD compiler for high-performance CPU programming. *Proceedings of Innovative Parallel Computing (InPar)* (2012). 4, 6

[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-high resolution immersive environments. In *IEEE LDAV (to appear)* (2013). 1, 2, 4, 5, 7

[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. 7

[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

[WBW13] WOOP S., BENTHIN C., WALD I.: Intel embree 2.0: Photorealistic ray tracing kernels, http://embree.github.io. 4, 6, 8

[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

[ZPvBG01a] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Ewa volume splatting. In *Visualization, 2001. VIS'01. Proceedings* (2001), IEEE, pp. 29–538. 3

[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

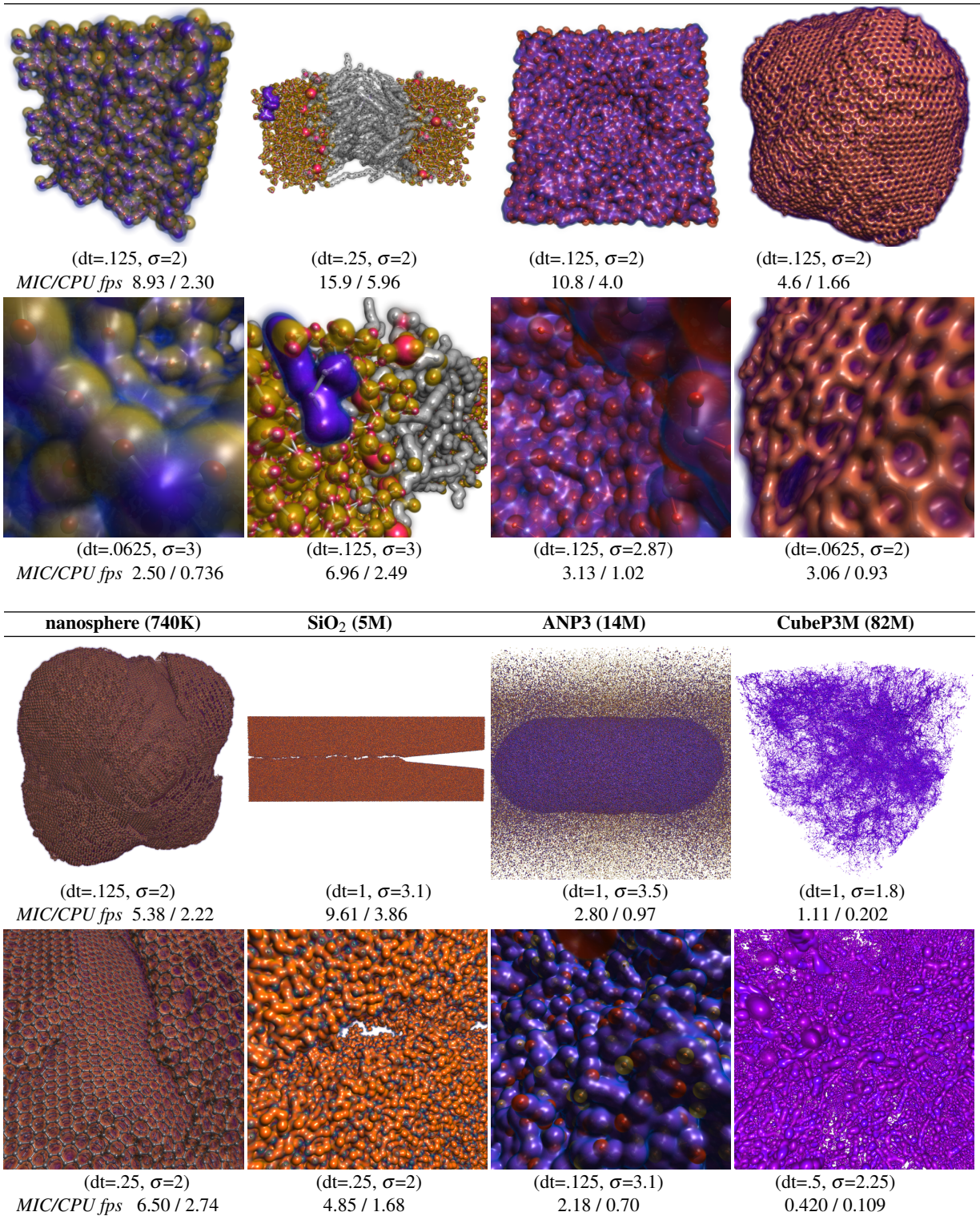

Figure 5: *Results on MIC (SE10P) and CPU (dual Xeon E5-2680) at 1 MP* (1024^2).