Screen-Space Normal Distribution Function Caching for Consistent Multi-Resolution Rendering of Large Particle Data

Mohamed Ibrahim, Patrick Wickenhäuser, Peter Rautek, Guido Reina, Markus Hadwiger

Fig. 1. **Visualizing large molecular dynamics particle data at different zoom levels.** Top row: Screen-space normal distribution functions (S-NDFs) facilitate scale-consistent rendering at any zoom level by storing normal distributions in a screen-space mipmap (S-MIP), instead of storing output pixel colors. This enables zooming and re-lighting without having to re-sample the actual particle data. Bottom row: Standard ray casting suffers from aliasing and noise due to under-sampling of the particles in coarser levels (toward right).

Abstract—Molecular dynamics (MD) simulations are crucial to investigating important processes in physics and thermodynamics. The simulated atoms are usually visualized as hard spheres with Phong shading, where individual particles and their local density can be perceived well in close-up views. However, for large-scale simulations with 10 million particles or more, the visualization of large fields-of-view usually suffers from strong aliasing artifacts, because the mismatch between data size and output resolution leads to severe under-sampling of the geometry. Excessive super-sampling can alleviate this problem, but is prohibitively expensive. This paper presents a novel visualization method for large-scale particle data that addresses aliasing while enabling interactive high-quality rendering. We introduce the novel concept of screen-space normal distribution functions (S-NDFs) for particle data. S-NDFs represent the distribution of surface normals that map to a given pixel in screen space, which enables high-quality re-lighting without re-rendering particles. In order to facilitate interactive zooming, we cache S-NDFs in a screen-space mipmap (S-MIP). Together, these two concepts enable interactive, scale-consistent re-lighting and shading changes, as well as zooming, without having to re-sample the particle data. We show how our method facilitates the interactive exploration of real-world large-scale MD simulation data in different scenarios.

Index Terms—Multiresolution Techniques, Point-Based Data, Glyph-based Techniques, Scalability Issues, Molecular Visualization

1 Introduction

With the current advances in computational power and the wide-spread availability of large compute clusters, molecular dynamics (MD) simulations based on classical force field models are coming ever closer to accurately describing many physical effects at a macroscopic scale with atomistic accuracy. However, these simulations lead to very large data sets, which poses significant challenges to interactive high-quality visualization. A major motivation for large-scale MD simulations is that increasing the system size, i.e., the number of atoms, increases the odds for capturing events like nucleation during condensation or evaporation under technically relevant conditions. In order to be able to capture the relevant events, simulations usually also have to run for many time steps so that the required processes may occur. Modeling scenarios on an atomic scale while bridging to macroscopic laws requires even larger simulations. For example, a single barely-visible droplet of liquid contains on the order of a trillion molecules. Simulations on such a scale are costly, but have become feasible [7, 22]. This enables comparison with human-observable experiments. The ground truth visualization of atomistic simulations in the context of physics, materials science, or thermodynamics is commonly realized using spheres, whose arrangement and radii correspond to the typical distances between atom sites with respect to the force field employed in the simulation. Substantial overlap between two such spheres will usually lead to extreme repulsion and can be an indicator of a misbehaving simulation. Such direct insight from observing a simulation visualization has made this metaphor popular with domain scientists.

Using brute force ray casting enables rendering of data sets on the order of 10^8 particles on single-GPU systems [11]. More elaborate solutions using culling techniques push these limits even further [12]. However, at these scales the major factor that limits the effectiveness of visualizations is the insufficient sampling rate that is caused by the mismatch between data size and the resolution of today's output devices. In general, techniques based on ray casting [13, 18, 25, 33, 34] are prone to aliasing due to under-sampling, which leads to noisy renderings that are inconsistent across scales. The importance of this effect is shown in Fig. 1, where we compare brute force ray casting (bottom row) with our novel, scale-consistent rendering approach (top row). Approaches such as image-space normal smoothing [12] have been applied successfully to reduce noise artifacts and to make mesoscopic features easier to recognize. Another approach is to circumvent the problem by gradually removing the data features that would otherwise cause aliasing [20]. However, these techniques by design do not produce scale-consistent results, remove fine details, and result in undesired over-smoothing.

Scale-inconsistent rendering generally occurs when sphere glyphs become smaller than individual pixels. In this case, the ray-sphere in-

- *Mohamed Ibrahim, Peter Rautek, and Markus Hadwiger are with King Abdullah University of Science and Technology (KAUST), Thuwal, 23955-6900, Saudi Arabia. E-mail: {mohamed.ibrahim, peter.rautek, markus.hadwiger}@kaust.edu.sa.*
- *Patrick Wickenhäuser and Guido Reina are with Visualization Research Center (VISUS) at the University of Stuttgart, Germany. E-mail: patrick.wickenhaeuser@online.de, guido.reina@visus.uni-stuttgart.de.*

Manuscript received 31 Mar. 2017; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

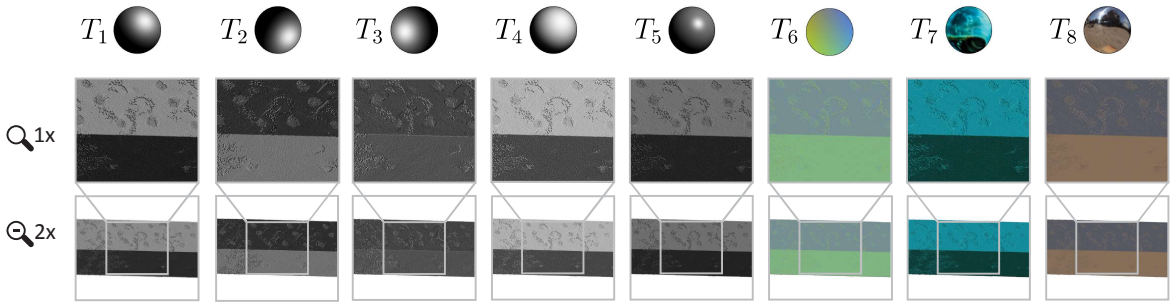

Fig. 2. **Rendering overview.** Screen-space normal distribution functions (S-NDFs) and screen-space mipmaps (S-MIPs) enable interacting with large-scale particle data without re-sampling the particles after every interaction. A high-quality visualization of a Copper-Silver Mixture is updated in real time after user operations, including changing the lighting environment, e.g., the light direction, as shown in T_1, T_2, and T_3, changing the specular exponent of the material, as shown in T_4 and T_5, changing the transfer function, as shown in T_6, T_7, and T_8, or zooming out (vertical direction).

tersections, and the subsequent shading computations, result in aliasing and noise (Fig. 1, bottom right). This masks the actual structure of the data and is also inconsistent between subsequent frames during interaction. This is especially bothersome during exploration when the user frequently switches between overview and detail. Therefore, a high-quality visualization method should be consistent across scales.

We present a novel approach for the visualization of large-scale particle data that significantly improves on previous techniques and enables scale-consistent top-down exploration. Our technique caches *screen-space normal distribution functions* (S-NDFs), which allows for progressive, consistent rendering of large-scale particle data at interactive rates. As illustrated in Fig. 2, for a given view direction our cached S-NDF representation enables interactive zooming and panning (Fig. 2, rows) as well as material and lighting changes (Fig. 2, columns). At the same time, we can progressively refine the current S-NDFs, by simply adding more samples to them. Without our representation, consistent rendering is impractical, since the sampling process must be restarted from scratch after changes to any visualization parameter.

Our implementation uses GPU ray casting [11], for computing S-NDFs as well as for comparing with brute force ray casting. However, our entire S-NDF pipeline could equivalently be implemented with state-of-the-art CPU ray tracers, e.g., [33, 34], with no conceptual differences, while obtaining the same benefits. The most important property of our method is the computation and use of S-NDFs, which makes it inherently consistent across scales and output resolutions.

2 Related Work

Ray casting has been the traditional approach to render large scale particle data sets, whether on the CPU or on the GPU. GPU-based ray casting of ellipsoids was introduced in 2006 [13, 18]. In light of the better performance and image quality compared to traditional polygon-based rendering as well as depth-correct billboards [3], the technique of ray casting quadratic surfaces has since been adopted as state-of-the-art for molecular dynamics visualization [11, 25, 28]. This technique constitutes a hybrid object/image space approach: billboard geometry for each particle is employed for starting per-fragment rays that trace the final result. As data set sizes increase, the billboards that are visible simultaneously grow increasingly smaller, making each glyph cheaper to render and allowing satisfactory scaling. No advanced culling techniques are thus needed to render data sets of over 100 million particles directly using current consumer hardware, although culling has been shown to be worthwhile [12]. Data sets with billions of atoms can also be rendered interactively—assuming memory requirements can be constrained via instancing, for example—by switching to sorting the particles/instances into a grid and casting rays for the whole scene [8, 21]. However, in principle all of these approaches suffer from aliasing due to severe under-sampling of very large scenes, which can still be explored interactively. The aliasing problem also applies to recent CPU-based methods [33, 34], which scale significantly better in terms of performance than in terms of visual accuracy. Complementary techniques have also been developed to reduce the resulting noise [12].

Progressive rendering techniques have been employed to solve the aliasing problem. Early on, Painter et al. [24] proposed an adaptive progressive sampling approach to produce high-quality, anti-aliased images. Their sampling approach is dependent on three criteria: coverage of the image, location of features, and confidence in the values at a distinguished "pixel level" of resolution. Arikan et al. [2] presented a progressive ray casting technique for progressive generation of images.

Deferred shading techniques are a more recent development in particle rendering. Grottel et al. [12] use the 3x3 neighborhood of each fragment to construct a Bèzier patch that supplies the surface normal if glyphs approach single-pixel size. The two approaches developed by Falk et al. [8] and Lindow et al. [21] both use depth buffers storing the nearest ray-particle intersection. Based on the depth buffer, they create and light smooth surfaces, instead of using the actual ray-particle intersections. This removes aliasing, but introduces over-smoothing. Average surfaces can only represent structures accurately that are larger than individual pixels. Moreover, in sparsely populated areas non-existent surfaces are created, which occlude other features. We show that large-scale particle data exhibit complex sub-pixel shading behavior, and instead employ an accurate alternative to normal smoothing.

Other large-scale particle visualization and in-situ techniques. Rizzi et al. [26] show parallel splatting of cosmology data including simple LOD rendering by using subsets of particles. However, this does not offer consistency across scales, as intensity is just accumulated via blending. Rendering subsets loses intensity because of splats that contribute less, which also happens when zooming due to decreasing overlap. Subsetting and LOD generation can also be performed *in situ*. This significantly reduces the data size for later visualization [37]. Other previous work proposed attenuating minified splats and radiant-flux-preserving representatives for LOD rendering [16], or always choosing LOD nodes that fit the output resolution as closely as possible [10].

Explorable images [30] are a powerful paradigm for in-situ visualization, which has also been proposed for visualizing particle simulations [39]. In contrast to our solution, the latter work does not offer scale consistency, but allows for limited view changes using the stored depth buffer, with artifacts proportional to the angle change. Changing the camera zoom/distance is not allowed because the resulting holes would grow too large almost immediately. However, stored normals enable re-lighting and cutaways. Another related approach is the ParaView Cinema technology [1]. We leave exploring further in situ applications of our S-NDF-based approach for future work.

An approach employing a metaphor orthogonal to our rigid spheres for molecular dynamics rendering is proposed by Knoll et al. [19]. Here, particle data is volume-rendered as RBFs to allow for flexible surface reconstruction and element-dependent classification via multi-volumes. This technique is geared towards running distributed in-situ and taking advantage of hybrid CPU/Xeon Phi clusters.

Consistent multi-resolution rendering via PDFs. Representing image and volume data via probability density functions (PDFs) allows for consistent multi-resolution rendering and processing of both image [14] and volume data [27, 40]. These PDFs can be approximated sparsely and, e.g., be stored in sparse PDF maps [14] and sparse PDF volumes [27], respectively. These representations are very similar to standard mipmaps [36], but enable the accurate and efficient evaluation of color mapping and non-linear filtering. Accurate approximation of PDFs is possible using isotropic Gaussians [14, 27]. Our approach uses

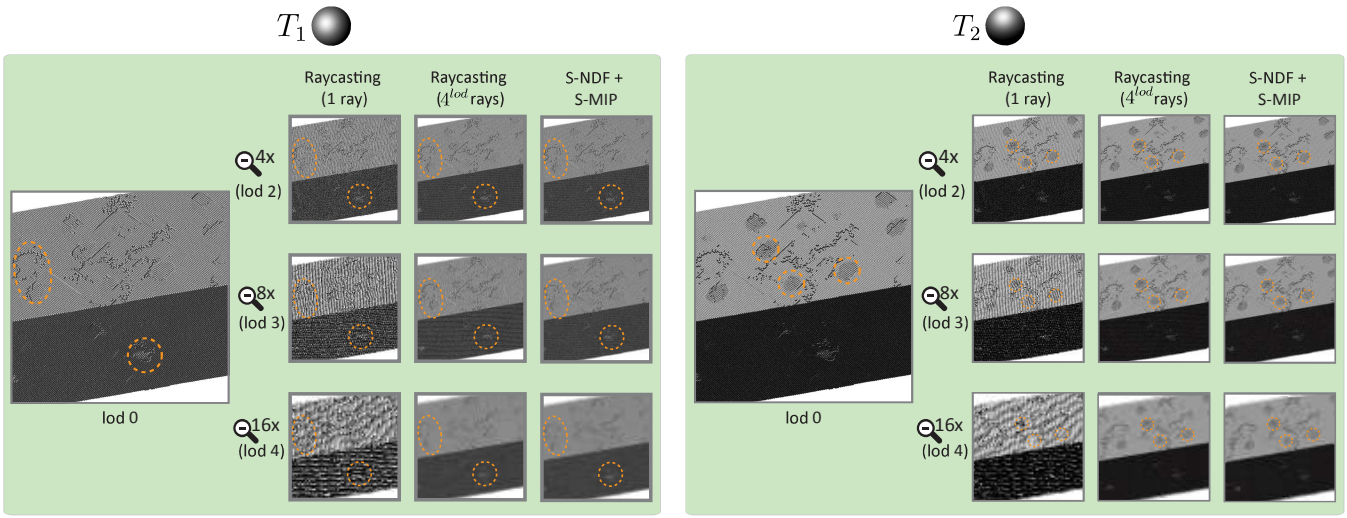

Fig. 3. **Scale-consistent visualization.** Standard ray casting (left-most columns) incurs strong aliasing artifacts and visible noise for zoomed-out views. This makes it impossible to discern small features (orange circles) in large particle data, such as the Copper/Silver mixture shown here, during interactive exploration. Super-sampling can remove this problem (middle columns), but is not possible at interactive rates. Moreover, being able to interactively change the lighting environment and/or lighting direction allows users to explore different features (T_1 vs. T_2). Our approach based on S-NDFs enables interactive exploration with a quality comparable to super-sampling, but at interactive rates (right-most columns).

similar ideas by preserving the distribution of surface normal vectors, which enables consistent, scale-independent shading.

Normal filtering for local illumination. Fournier [9] pointed out the importance of accurate surface normal vector filtering for local illumination. He introduced the normal distribution function (NDF) as an accurate representation of the surface normal vectors, which can be filtered linearly. The computation of mipmaps of normal maps leads to the same problem [15, 31]. Many approaches, including ours, build upon NDFs to capture local illumination effects more accurately and consistently than is possible with individual normal vectors. Bruneton et al. [5] give a survey of such techniques, including LEAN mapping [23], and LEADR mapping [6]. However, some of these techniques make assumptions about the material's BRDF, which restricts them to the cases that were specifically considered. Han et al. [15] employ sparse approximations via von Mises-Fisher distributions and spherical harmonics. Tan et al. [29] approximate the overall reflectance instead of the normal distribution. Another recent approach represents NDFs for surfaces exhibiting high-frequency glints using triangulation [38].

We also use full NDFs instead of individual normal vectors. However, we use binning instead of representing NDFs as von Mises-Fisher or spherical harmonics distributions, because we target straightforward progressive refinement of NDFs by adding additional samples.

3 Method Overview

The main idea of our approach is to use a representation of surface normals for visualization that is independent of the output resolution, in order to enable scale-consistent rendering. Fig. 3 illustrates this property, where our visualizations capture the same features in a Copper/Silver mixture data set in all output resolutions (right-most columns). This is otherwise only possible by using excessive super-sampling, with the number of samples increasing exponentially when zooming out (center columns). In contrast, standard interactive visualization cannot capture small features in zoomed-out views due to aliasing (left-most columns).

S-NDFs. We achieve this by defining a novel screen space variant of *normal distribution functions* [9, 35], which we call *screen-space normal distribution functions*, or S-NDFs. Similarly to standard NDFs, instead of capturing a single surface normal, our S-NDFs comprise a probability density function over the entire space of normal directions. However, our S-NDFs are defined entirely with respect to screen space (Sec. 4), instead of being defined with respect to a macrosurface [35].

S-NDF-based deferred shading. An important property of S-NDFs is that they enable users to interactively change the lighting and material properties, as well as zooming and panning. Since the distribution of surface normals that correspond to a pixel's footprint in screen space is known from the S-NDF, different lighting properties (direction, exponent, environment map, etc.) or material properties (BRDF, coefficients) can be applied in a simple deferred shading pass that computes each new pixel color without sampling the particle data again (Sec. 5).

Multi-resolution S-NDF filtering and zooming. NDFs, and likewise S-NDFs, can be filtered linearly [15]. This has important implications: Lighting is a non-linear operation, which prevents simple linear filtering of normals for down-sampling [23]. However, in contrast to individual normals, because S-NDFs can be down-sampled linearly, using them as the basis for (deferred) lighting results in consistent multi-resolution rendering. This property also enables us to perform interactive zooming without re-computing all S-NDFs: S-NDFs for lower output resolutions can be obtained directly by linearly filtering and down-sampling the S-NDFs of higher resolutions.

Caching S-NDFs. In order to perform zooming and panning efficiently in practice, we cache all already computed S-NDFs in screen space, using a virtual mipmap hierarchy [36] that, instead of storing pixel colors directly, stores an S-NDF for every output pixel. Like the S-NDFs, this mipmap is defined with respect to screen space. We therefore call it a *screen-space mipmap*, or S-MIP (Sec. 6).

Interaction. Fig. 4 gives an overview of the interactions enabled by our approach. The S-NDFs enable interactive re-lighting (Fig. 4, horizontal direction), and caching S-NDFs in an S-MIP enables combining this with interactive zooming and panning (Fig. 4, vertical direction).

Progressive S-NDF refinement. The linearity of S-NDF filtering furthermore implies that it is straightforward to refine S-NDFs in a progressive manner, by simply iteratively adding new samples. Whenever the view point does not change, we iteratively add sub-pixel samples (normal vectors from sub-pixels) to the already cached S-NDFs. At the same time, we perform deferred shading to obtain the anti-aliased output. This progressively improves the quality of the S-NDFs and thus the output image, while amortizing sampling cost over multiple frames.

4 Screen-Space Normal Distribution Functions

In order to define a screen-space normal distribution function (S-NDF), we consider a region in screen space, e.g., the support of a pixel, and the particle geometry that projects to it. We call this region the *pixel footprint* $\mathbf{x}$, which is our area of interest in screen space. See Fig. 5.

Notation. Our definition of the S-NDF $D_{\mathbf{x}}(\mathbf{m})$ below is inspired by normal distribution functions used in microfacet models for rough surfaces, particularly the microfacet normal distribution $D(\mathbf{m})$ defined by Walter et al. [35]. However, we would like to emphasize that our S-NDFs are defined *in* (orthogonal to) *screen space*, whereas a microfacet normal distribution is defined with respect to a (macro)surface.

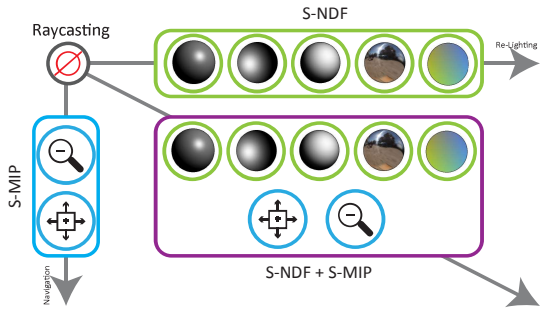

Fig. 4. **Interactions enabled by our approach.** Standard ray casting of particle data has to re-sample the data set after every user interaction. We enable two "orthogonal" ways of interacting with the data without re-sampling particles: (horizontal) S-NDFs enable changing the lighting; (vertical) S-MIPs enable zooming. Combining them jointly enables both.

For defining our domain of integration, we use the standard projected solid angle measure $\sigma^\perp(\omega)$, which is summarized in the supplementary material. For more details see, e.g., Jakob [17] or Veach [32].

4.1 Definition

Given a pixel footprint $\mathbf{x}$ in screen space, and a domain $\mathcal{D} \subseteq \Omega$ on the hemisphere Ω of normal directions, we consider the probability that a ray cast randomly with uniform probability anywhere inside the footprint $\mathbf{x}$ intersects a particle whose normal vector maps to $\mathcal{D}$.

"Projected area" probability. For a given pixel footprint $\mathbf{x}$, with corresponding screen space area $A(\mathbf{x})$, we thus define the probability

$$P_\mathbf{x}(\mathcal{D}) := \frac{A^\perp(\mathcal{D})}{A(\mathbf{x})}, \tag{1}$$

where $A^\perp(\mathcal{D})$ is the area measure for the total projected area that "sees" a normal vector that maps to $\mathcal{D} \subseteq \Omega$. This is illustrated in Fig. 5. Note that $P(\Omega) \leq 1$, because some rays might not intersect any geometry.

S-NDF definition. We define the screen-space normal distribution function $D_\mathbf{x}(\mathbf{m})$ of a pixel footprint $\mathbf{x}$ as the probability density

$$D_\mathbf{x}(\mathbf{m}) := \frac{dP_\mathbf{x}(\omega_m)}{d\sigma^\perp(\omega_m)}, \tag{2}$$

where $\mathbf{m}$ is a normal vector, and ω_m is a solid angle centered around $\mathbf{m}$. We specify $\mathbf{m}$ via 2D coordinates in the unit disk $\Omega^\perp$, i.e., via (x,y) with $x^2+y^2 \leq 1$. The corresponding 3D normal is $(x,y,\sqrt{1-x^2-y^2})$.

Integration. $D_\mathbf{x}(\mathbf{m})$ gives the normal distribution of the projected particle geometry that maps to the footprint $\mathbf{x}$ as a probability density function, where the domain of integration is the hemisphere with respect to the projected solid angle measure $\sigma^\perp(\omega)$. We can therefore obtain the probability defined above via integration over the domain $\mathcal{D} \subseteq \Omega$:

$$P_\mathbf{x}(\mathcal{D}) = \int_\mathcal{D} D_\mathbf{x}(\mathbf{m})\, d\sigma^\perp(\omega_m). \tag{3}$$

Table 1. S-NDF and lighting terminology.

Symbol	Description
$\mathbf{x}$	Pixel footprint in screen space
$A(\mathbf{x})$	Area measure of the footprint $\mathbf{x}$
$\Omega, \mathcal{D}$	Hemisphere of normal directions Ω; subset $\mathcal{D} \subseteq \Omega$
$\mathcal{D}_i$	Subset of Ω corresp. to "bin" i, where $\cup_i \mathcal{D}_i = \Omega$
$\Omega^\perp$	Projected unit disk corresponding to Ω
$P_\mathbf{x}(\mathcal{D})$	Probability of $\mathbf{x}$ "seeing" a normal mapping to $\mathcal{D}$
$D_\mathbf{x}(\mathbf{m})$	S-NDF for $\mathbf{x}$
$\mathbf{m}$	A normal vector with respect to screen space
ω_m	Solid angle on Ω centered around $\mathbf{m}$
$\sigma^\perp(\omega)$	Projected solid angle measure of solid angle ω
$f_r(\omega_i \to \omega_o)$	BRDF (bi-directional reflectance distribution func.)
ω_i, ω_o	Direction of incoming light / outgoing light
$L(\mathbf{p}, \omega)$	Radiance at $\mathbf{p}$ in direction ω

Pixel footprint coverage. Integration of $D_\mathbf{x}(\mathbf{m})$ for a given pixel footprint $\mathbf{x}$ over the entire hemisphere Ω (all normal directions) gives

$$\int_\Omega D_\mathbf{x}(\mathbf{m})\, d\sigma^\perp(\omega_m) = P_{geometry}. \tag{4}$$

$P_{geometry}$ is the probability that a random ray uniformly chosen over the footprint $\mathbf{x}$ intersects *some* particle. Conversely, the probability of not intersecting any particle at all is $(1 - P_{geometry})$. For example, if a given $\mathbf{x}$ only "sees" particles over 80% of its area $A(\mathbf{x})$, $P_{geometry} = 0.8$, and 20% of its area does not see any geometry at all. This also means that the average radiance received by $\mathbf{x}$ over its area $A(\mathbf{x})$ will be only 80% of what it would have been with full pixel coverage. See Sec. 5.

4.2 Discretization and Computation

In order to discretize an S-NDF for storage purposes, we subdivide the domain of the hemisphere Ω into disjoint subdomains $\mathcal{D}_i \subseteq \Omega$ that together cover the whole domain Ω: $\cup_i \mathcal{D}_i = \Omega$. Each of these subdomains $\mathcal{D}_i$ corresponds to a *bin* in the discretized S-NDF representation. Fig. 6 shows three different variants for binning patterns in practice.

Computing probabilities. The first step of computing the S-NDF $D_\mathbf{x}(\mathbf{m})$ for a pixel footprint $\mathbf{x}$ is estimating the corresponding probabilities (integrals) $P_\mathbf{x}(\mathcal{D}_i)$ for all bins $\mathcal{D}_i$ via Monte Carlo integration: We shoot rays over the pixel footprint $\mathbf{x}$ with uniform probability, and for each ray compute the intersection position with a particle and the corresponding normal vector there. Each such normal sample increases the corresponding bin count in the discretized representation of $P_\mathbf{x}(\mathcal{D}_i)$. After shooting as many rays for $\mathbf{x}$ as desired, the obtained bin counts simply have to be normalized by the total number of rays shot. This results in estimates of the probabilities $P_\mathbf{x}(\mathcal{D}_i)$.

Computing S-NDFs. After the integrals $P_\mathbf{x}(\mathcal{D}_i)$ have been estimated, the corresponding $D_\mathbf{x}(\mathbf{m})$ can be computed via numerical differentiation. We do this by simply dividing each $P_\mathbf{x}(\mathcal{D}_i)$ by the corresponding projected solid angle measure of the entire bin $\mathcal{D}_i$:

$$D_\mathbf{x}(\mathbf{m}_i) \approx \frac{P_\mathbf{x}(\mathcal{D}_i)}{\sigma^\perp(\mathcal{D}_i)}, \quad \sigma^\perp(\mathcal{D}_i) = \int_{\mathcal{D}_i} d\sigma^\perp(\omega). \tag{5}$$

Here, $\mathbf{m}_i$ denotes the representative normal direction $\mathbf{m}$ for bin $\mathcal{D}_i$, which we (approximately) choose to be in the center of the bin.

Computations in $\Omega^\perp$. We note that due to the use of the projected solid angle measure $\sigma^\perp(\omega)$, many of our computations can be performed in the (projected) unit disk $\Omega^\perp$, instead of requiring one to work on the hemisphere Ω. That is, each $\mathcal{D}_i$ corresponds to a region of $\Omega^\perp$. Carrying out computations in $\Omega^\perp$ greatly simplifies implementation.

5 Lighting Computations with S-NDFs

We first summarize the standard computation of lighting (radiance) at a single point $\mathbf{p}$ on a surface, with respect to a single corresponding normal vector $\mathbf{n}(\mathbf{p})$ and a given *bi-directional reflectance distribution function* (BRDF) [17]. Then, we introduce how we instead compute the

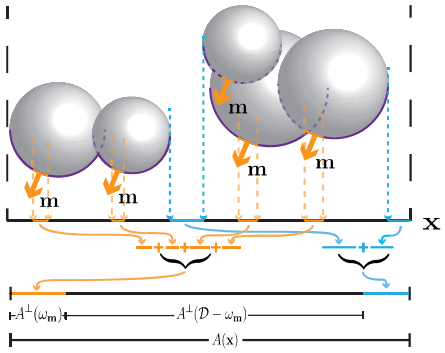

Fig. 5. **Pixel footprint and projected area probability.** S-NDFs are defined over the footprint of a pixel in screen space. They give the corresponding area in the footprint that "sees" normal vectors mapping to a given solid angle on the hemisphere of all possible normal directions.

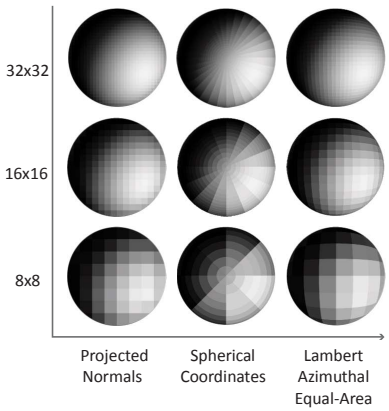

Fig. 6. **S-NDF discretization.** Three practical binning patterns for subdividing the projected hemisphere $\Omega^\perp$ into bins $\mathcal{D}_i$ such that $\cup_i \mathcal{D}_i = \Omega^\perp$, using (from bottom to top) 64, 256, and 1024 S-NDF bins, respectively.

lighting over a pixel footprint $\mathbf{x}$ in screen space, using *all* the normal vectors it "sees," as described by the corresponding S-NDF $D_\mathbf{x}(\mathbf{m})$.

5.1 Lighting at a given point on a surface

Radiance. The radiometric quantity *radiance* can be defined as the *density* of *radiant power* Φ, per unit projected solid angle ω, per unit area A orthogonal to ω, flowing through a point $\mathbf{p}$ [17]:

$$L(\mathbf{p},\omega) := \frac{d^2\Phi}{d\sigma^\perp(\omega)\,dA}. \tag{6}$$

BRDF. The BRDF $f_r(\cdot)$ at a point $\mathbf{p}$ is defined as the proportionality constant between the differential outgoing radiance dL_o, going out in direction ω_o, and the differential *irradiance* dE_i, coming in from direction ω_i, at the point $\mathbf{p}$, with respect to its normal $\mathbf{n}(\mathbf{p})$ [17]:

$$f_r(\omega_i \to \omega_o) := \frac{dL_o(\omega_o)}{dE_i(\omega_i)} = \frac{dL_o(\omega_o)}{L_i(\omega_i)\,d\sigma^\perp(\omega_i)}, \tag{7}$$

where the irradiance is obtained via the differential projected solid angle as $dE_i(\omega_i) := L_i(\omega_i)\,d\sigma^\perp(\omega_i)$, where $L_i(\omega_i)$ is the incoming radiance from direction ω_i. Integrating this equation to obtain $L_o(\mathbf{p},\omega_o)$ gives [17]

$$L_o(\mathbf{p},\omega_o) = \int_\Omega L_i(\mathbf{p},\omega_i)\, f_r(\omega_i \to \omega_o)\, d\sigma^\perp(\omega_i). \tag{8}$$

The parameterization of the BRDF $f_r(\omega_i \to \omega_o)$ and the expression for $L_o(\mathbf{p},\omega_o)$ depend on a given, single surface normal direction $\mathbf{n}(\mathbf{p})$ at $\mathbf{p}$.

5.2 Lighting with S-NDFs in screen space

Our goal is now to switch to computing everything with respect to *screen space*. We also want to remove the dependency on single normal vectors $\mathbf{n}$, and instead be able to refer to an entire solid angle of normal directions ω_m centered around a normal vector $\mathbf{m}$ in screen space.

Radiance with respect to m. We define the equivalent to Eq. 8 parameterized with an (arbitrary) normal vector $\mathbf{m}$, instead of position $\mathbf{p}$:

$$L_o(\mathbf{m},\omega_o) := \int_{\Omega_m} L_i(\omega_i)\, f_r(\omega_i \to \omega_o)\, d\sigma^\perp(\omega_i), \tag{9}$$

where the domain of integration is the hemisphere Ω_m centered around the normal vector $\mathbf{m}$, and ω_o and ω_i are given with respect to $\mathbf{m}$. Note that in contrast to Eq. 8, $L_i(\omega_i)$ now only depends on a direction, and not also on a surface point $\mathbf{p}$. This restriction means that we have to treat the lighting environment as constant for all surface points. This also means that we cannot handle shadowing-masking terms [35], for example. However, our approach targets *arbitrary* geometry, where computing such terms is very costly, instead of statistical surface descriptions.

Radiance of pixel footprint x. We can compute the average radiance at the footprint $\mathbf{x}$ by integrating over all rays $\mathbf{r}$ cast inside $\mathbf{x}$:

$$L(\mathbf{x}) := \frac{1}{A(\mathbf{x})} \int_{\mathbf{r}(\mathbf{x})} L_o\left(\mathbf{m}(\mathbf{r}), \omega_o(\mathbf{r})\right)\, dA^\perp(\mathbf{r}), \tag{10}$$

where $A^\perp(\mathbf{r})$ is the area measure for $\mathbf{r}$ orthogonal to screen space, $\mathbf{m}(\mathbf{r})$ is the normal vector at the intersection of ray $\mathbf{r}$ with a particle, and $\omega_o(\mathbf{r})$ is the ray direction transformed into the local frame of the BRDF.

Radiance from S-NDF. Now, changing the domain of integration from rays $\mathbf{r}(\mathbf{x})$ to the hemisphere Ω orthogonal to screen space, and plugging in the definition of $D_\mathbf{x}(\mathbf{m})$ (Eq. 2), allows us to write:

$$L(\mathbf{x}) = \int_\Omega D_\mathbf{x}(\mathbf{m})\, L_o\left(\mathbf{m}, \omega_o(\mathbf{r}_\mathbf{x})\right)\, d\sigma^\perp(\omega_m), \tag{11}$$

where $\mathbf{r}_\mathbf{x}$ is the ray direction through the center of the pixel footprint $\mathbf{x}$. This is equivalent to Eq. 10 (assuming constant ray directions within $\mathbf{x}$), because $D_\mathbf{x}(\mathbf{m})$ is the density of normal vectors pointing in direction $\mathbf{m}$, corresponding to the geometry that is seen over the pixel footprint $\mathbf{x}$ in screen space. Note that the $A(\mathbf{x})$ term has fallen out (cf. Eq. 1).

The radiance $L_o(\mathbf{m},\omega_o)$ can be computed from any given BRDF, e.g., from the simple Blinn-Phong BRDF [4] to more complicated ones.

5.2.1 Pre-integration of BRDF and lighting environment

To improve the accuracy of the S-NDF lighting computations, for any given discretization of $D_\mathbf{x}(\mathbf{m})$ into bins $\mathcal{D}_i \subseteq \Omega$ as described above, we

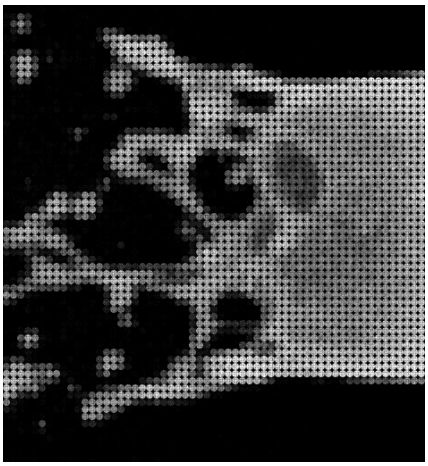

Fig. 7. **S-NDF visualization in screen space.** We illustrate the 64x64 S-NDFs corresponding to a coarse 64x64 pixel visualization of a laser ablation crown. A corresponding high-resolution visualization, with lighting computed in screen space using these S-NDFs, is shown in Fig. 8.

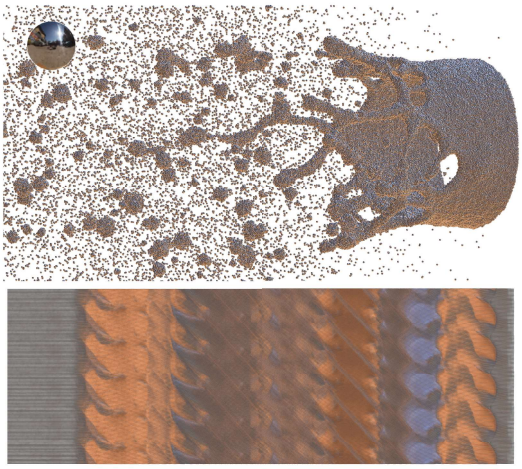

Fig. 8. **Lighting using S-NDFs.** Data sets such as (top) a 518,000-particle laser ablation crown, and (bottom) a 100 million particle instanced laser data set, can be lit interactively using S-NDFs, independent of the number of particles. (Top left) Environment map used for lighting.

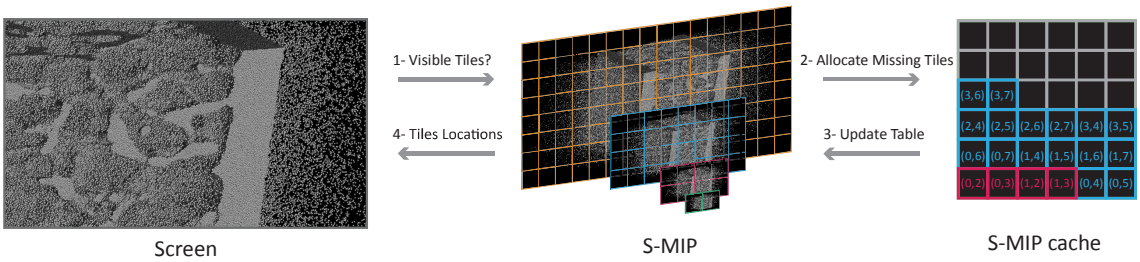

Fig. 9. **Screen-space mipmap (S-MIP).** We render in the texture space of a virtual mipmap hierarchy that is defined parallel to screen space. Virtualizing the S-MIP enables supporting large output resolutions as well as interactive panning. Texture space is split into 2D tiles, and the current view determines which of these tiles are visible (1). Tile storage is managed in an S-MIP cache that stores only current/previously visible tiles (2). Rendering on screen is done by locating the necessary tiles in the S-MIP cache (3), and rendering them in the correct location in the output image (4).

can *pre-integrate* the outgoing radiance corresponding to all normal vectors mapping to each bin $\mathcal{D}_i$. For any given $\mathbf{x}$, we assume that $D_\mathbf{x}(\mathbf{m})$ is constant for all $\mathbf{m} \in \mathcal{D}_i$, i.e., $D_\mathbf{x}(\mathbf{m}) = D_\mathbf{x}(\mathbf{m}_i)$. We can then move the constant term out of the integral in Eq. 11, and pre-integrate

$$\tilde{L}(\mathcal{D}_i) := \int_{\mathcal{D}_i} \left(\int_{\Omega_m} L_i(\omega_i)\, f_r(\omega_i \to \omega_o)\, d\sigma^\perp(\omega_i) \right) d\sigma^\perp(\omega_m), \quad (12)$$

which is independent of the pixel footprint $\mathbf{x}$ and the corresponding S-NDF $D_\mathbf{x}(\mathbf{m})$, and solely depends on the BRDF $f_r(\omega_i \to \omega_o)$ and the incoming radiance $L_i(\omega_i)$. We can therefore pre-compute $\tilde{L}(\mathcal{D}_i)$ for a given BRDF and lighting environment. This simply requires storing one radiance value per bin $\mathcal{D}_i$. Using the pre-integrated $\tilde{L}(\mathcal{D}_i)$, the lighting computation for a given footprint $\mathbf{x}$, taking into account the actual corresponding $D_\mathbf{x}(\mathbf{m}_i)$, becomes a simple sum over all bins $\mathcal{D}_i$:

$$L(\mathbf{x}) \approx \sum_i D_\mathbf{x}(\mathbf{m}_i)\, \tilde{L}(\mathcal{D}_i). \quad (13)$$

Looking at Eq. 5, we observe that we can also write this as

$$L(\mathbf{x}) \approx \sum_i P_\mathbf{x}(\mathcal{D}_i)\, \frac{\tilde{L}(\mathcal{D}_i)}{\sigma^\perp(\mathcal{D}_i)}, \quad (14)$$

which allows us to fold the differentiation with respect to $\sigma^\perp(\mathcal{D}_i)$ into a pre-computed table that stores $\tilde{L}(\mathcal{D}_i)/\sigma^\perp(\mathcal{D}_i)$ for each bin $\mathcal{D}_i$. Please see the supplementary material for more details on the implementation.

6 Caching S-NDFs in a Screen-Space Mipmap

Mipmapping is a standard method for texture anti-aliasing [36]. An original texture image is repeatedly down-sampled to smaller resolutions, creating a pyramid of texture images. This pyramid is then used during rendering for fast anti-aliasing. Using a pre-computed image pyramid is computationally much cheaper than down-sampling the original texture to each desired output resolution on-the-fly.

S-MIPs. We adapt this concept to screen space, and store a discretized S-NDF per "texel" instead of simply a texel color. This allows us to reduce the computational cost of rendering large-scale particle data, and facilitates interactive navigation. Specifically, in contrast to standard mipmaps our *screen-space mipmap*, or S-MIP, is defined parallel to screen space, and not parameterized onto a surface. This is in contrast to the standard case where texture space is parameterized onto general polygons in 3D space. Moreover, our S-MIP is completely *virtualized* in order to be able to address very high output resolutions.

6.1 S-MIP Representation

The S-MIP is composed of a pyramid of *resolution levels*. We choose level 0 to correspond to the desired maximum render resolution. Subsequent levels of the pyramid are then of a width and height that is a power of two smaller than the preceding level. For a given data set, we determine the resolution of level 0 such that it guarantees that each sphere will be sampled at least N times, where N is configurable.

Level 0 is typically too large to fit into GPU memory, even for small N and moderately-sized data sets. We therefore divide each level of the S-MIP into equally-sized 2D tiles of a user-specified size of $T_L \times T_L$ pixels, and allocate an S-MIP cache that only stores the tiles currently needed for rendering. Fig. 9 depicts an overview.

For rendering, we compute the required (visible) part of the S-MIP depending on the current view frustum and output resolution. The final output image is computed by interpolating between two adjacent S-MIP levels (LODs), as in standard tri-linear mipmap filtering.

6.1.1 Virtualizing the S-MIP

The tiled representation enables computing only those tiles that are needed to render the current output image. Initially, a fixed-size memory pool of tiles is allocated (the S-MIP cache). The first step of our rendering pipeline checks which tiles are required by the current output image. The S-MIP is augmented with a page table that is kept on the GPU, which keeps track of mapped tiles. If a required tile is not in the memory pool, a cache miss is reported. If there is memory available, the required tile is allocated and entered in the page table. When the cache is full, we use a *least recently used* strategy to free up memory.

Overall, this results in an *output-sensitive* pipeline. That is, only the S-NDFs in tiles of the S-MIP that contribute to output pixels on the screen will be computed and be used for deferred shading.

6.1.2 Down-sampling S-NDFs in the S-MIP

As in a standard mipmap, coarser levels of detail (LODs) in the S-MIP can be computed by down-sampling a finer LOD. This feature of the S-MIP is very useful for our purposes of large-scale particle rendering. It is specifically useful for the case when the user performs a 'zoom out' operation as shown in Fig. 10 (a) and (b). This requires a coarser version of the image currently being rendered, and therefore LOD_i can be computed by down-sampling the adjacent finer LOD_{i-1} in the S-MIP. Using a box filter, this corresponds to simply averaging four (2×2) adjacent S-NDFs in the S-MIP. This is illustrated in Fig. 10.

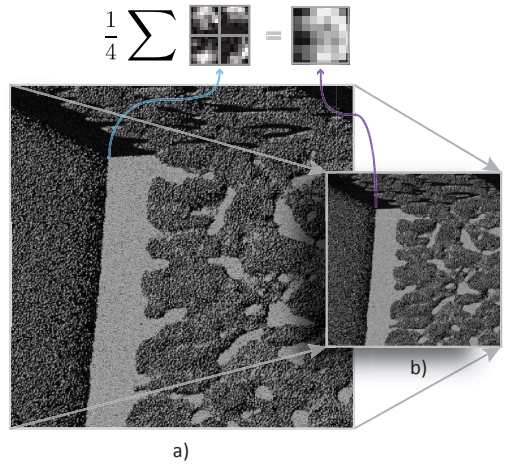

Fig. 10. **Down-sampling S-NDFs in the S-MIP.** Each S-NDF in a given LOD_i of the S-MIP (b), can be computed by simply averaging the four corresponding S-NDFs from the higher-resolution LOD_{i-1} (a).

6.1.3 Navigation using the S-MIP

In addition to the zooming described above, the second operation important for navigation is panning (translating) the view. Panning the view simply amounts to either fetching the S-MIP tiles corresponding to the new view from the cache, or allocating tiles that have just become visible in the cache and computing the contained S-NDFs from scratch.

6.2 Progressive S-NDF Computation

Aliasing can be reduced by increasing the number of samples per pixel via super-sampling, i.e., gathering many *sub-pixel* contributions for each output pixel. However, while super-sampling increases the complexity of surface details that can be captured without aliasing, it at the same time increases the computational complexity to a level that exceeds the capability of current GPUs for rendering at interactive frame rates. To alleviate the latency when computing S-NDFs in S-MIP tiles, we compute them via progressive sampling. That is, for a desired output resolution, we perform multiple iterations where each one computes one normal vector per pixel, and enters it into the pixel's S-NDF. Our implementation uses GPU ray casting [12] to generate normal vector samples, but any other method could be used as well, e.g., [33, 34]. Fig. 11 illustrates progressive sampling over time, together with the impact that different kinds of user interaction have.

6.2.1 S-MIP and S-NDF Invalidation

When the user pans or zooms, the cached S-MIP tiles are simply reused and only newly visible parts of the S-MIP are newly computed. This preserves the responsiveness of the user interface while continuously improving rendering quality. When the particle size or the viewing direction changes, the S-MIP cache must be completely invalidated, and the progressive refinement process is restarted. However, a single iteration of the refinement process in our pipeline is only slightly slower than standard ray casting without S-NDFs. Therefore, the user can still edit the particle size or rotate the scene at interactive frame rates at the same image quality as standard ray casting techniques. This implies that our approach always performs as well as standard ray casting in the worst case, but quickly converges to higher-quality rendering.

6.3 Implementation

In our S-MIP implementation, we set the tile size T_L to 64 pixels. Each texel stores the corresponding discretized S-NDF, which is essentially a 2D histogram with each bin storing the observed probability that the normal vector of a random sample inside the S-NDF's footprint points in that direction. Fig. 6 shows three different discretization patterns that we have implemented in our framework. We refer the reader to the supplemental material for a comparison of these techniques. The more bins are stored per pixel, the higher the accuracy of the rendering will be. For our main implementation we use the 'Projected Normals' binning technique with 8×8 bins per S-NDF. The Projected Normals binning pattern is very cheap to compute and provides a good trade-off between quality and performance in practice.

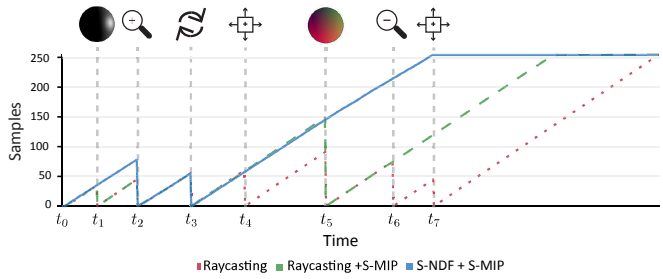

Fig. 11. **Progressive sampling and interaction.** The Laser Ablation data set is rendered using standard ray casting, an S-MIP without S-NDFs, and S-MIP plus S-NDFs. We plot the time it took to reach a maximum number of samples per pixel (here: 256). Different interactions have various impact on the time it takes to reach the maximum quality. From left to right, we interactively change the light direction, zoom in, rotate, pan, change the transfer function, zoom out, and pan.

7 Results and Discussion

We evaluate the image quality, performance, and possible user interactions of our technique using molecular dynamics data of different scales from two application domains. Table 3 gives an overview of all data sets we have used to evaluate our approach. Since all simulations use periodic boundary conditions (in case of the laser ablations only along the long faces) and the *instanced laser* data has an extremely elongated bounding box, we have replicated (stacked) it 5 times.

As we will show, our scale-consistent rendering not only significantly reduces noise, but also makes larger-scale structures more easily perceptible. Features contain coherent normals that, when collected into S-NDFs, result in a bias toward a specific direction.

The S-NDF refinement process can take a few seconds to deliver an accurate solution for large-scale particle data (10^7 particles upwards). To improve responsiveness during interaction, our S-MIP based S-NDF approach still allows multiple manipulations that do not invalidate the current refinement progress. These include changing the number and position of light sources, switching the BRDF (Lambert, Blinn-Phong), changing BRDF parameters like the specular exponent, changing the transfer function, zooming out, or panning. We provide several examples of these interactions for illustrative purposes.

7.1 Application Scenarios

We present multiple case studies that utilize our framework to explore data sets resulting from Molecular Dynamics Simulations. All data sets except one depict laser ablation simulations. These contain a block of solid matter, usually a metal. Where not noted otherwise, the metal is aluminum. These blocks are irradiated with short laser pulses to ablate material in the form of plasma. This process is used either for machining purposes or as a precise means for propulsion, which is especially useful in space, that is, in the absence of oxygen. In both cases, the trajectory of the ejected material is important to make sure the plasma does not collide with parts of the machine or the satellite, thus damaging it. The effect of the laser pulse on the metal block is also interesting, since the ensuing shock wave can cause compression of the otherwise ordered crystal lattice, which will later relax again.

Especially the second effect can be observed nicely with our technique, which on purposeful minification of the particles results in images that cross the boundary between glyph-based rendering and volume rendering. The last data set is a typical example from thermodynamics and consists of a fluid layer that is ripped in half, producing gas and droplets. The interesting features here are flat surfaces and small droplets, which have different scales that need to be reproduced consistently during exploration. We will now demonstrate exemplary exploration interactions and their effects on the features a user can detect in some of the data sets we have used for performance evaluation.

7.1.1 Exploration by changing the light direction

Fig. 12 shows how shock waves propagating within a solid block of aluminum can be highlighted using simple light source manipulation. The shock wave resulting from the laser impact alters the otherwise

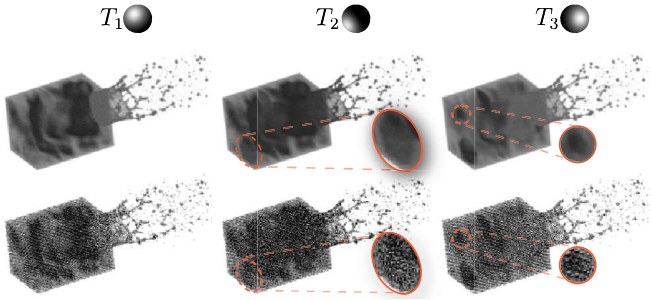

Fig. 12. **Changing the light direction** interactively at full quality allows exploring different features in the Laser Ablation data set. (Top row) Using S-NDFs makes the different features clearly visible, whereas (bottom row) standard ray casting obscures features due to under-sampling.

Table 2. **Interaction timings** for multiple interaction operations performed on multiple data sets, for an output resolution of $1,280 \times 720$ pixels. For each operation, we give timings for rendering using an S-MIP storing shaded pixels, vs. using an S-MIP storing S-NDFs. The first operation is re-lighting, and compares the time needed to re-light an image that is sampled with a specified number of samples per pixel (here: 256 samples). The second operation is zooming out by one level of detail, and compares the time needed to compute the final image from the finer-resolution LOD in the S-MIP (including down-sampling). The last operation gives the pure down-sampling time for the corresponding active tiles in the S-MIP cache.

data set	re-light [ms]		zoom out [ms]		down-sampling only [ms]	
	ray casting + S-MIP	S-NDF + S-MIP	color pixel S-MIP	S-NDF + S-MIP	color pixel S-MIP	S-NDF + S-MIP
laser ablation crown	1129.74	26.53	6.23	10.28	0.51	4.71
copper/silver mixture	1309.35	19.37	4.26	14.54	1.2	11.54
expanding fluid layer	1523.61	22.19	5.49	10.53	0.57	4.75
laser ablation	1253.84	22.17	6.34	14.07	0.53	4.72
instanced laser (5x)	1524.25	18.95	7.21	18.87	1.19	11.5
large laser ablation	2203.66	16.87	9.05	23.55	0.97	9.34

Table 3. **Data sets** with particle counts and sampling times for 1 sample per pixel, for an output resolution of $1,280 \times 720$ pixels. The first column is for inserting 1 sample per pixel into the corresponding S-NDF. The second column is for simply storing 1 sample in standard ray casting.

data set	# particles	S-NDF time [ms]	standard time [ms]
laser ablation crown	518,000	58.09	38.57
copper/silver mixture	14,500,000	53.34	45.77
expanding fluid layer	30,000,000	81.81	60.08
laser ablation	48,000,000	72.32	50.99
instanced laser ($5\times$)	$5\times$ 12,500,000	65.61	57.04
large laser ablation	199,940,704	119.76	97.42

regular crystal lattice of the aluminum block, resulting in particle density fluctuations. These fluctuations show up as flat structures that can be highlighted depending on the light source configuration: T_2 and T_3 reveal additional structures not noticeable using the original transfer function T_1 (highlighted in orange). Also note that the single-sample ray casting (bottom row) drowns most of the structures in heavy noise. No a-priori knowledge of features is required and no pre-processing, filtering or aggregation need to be performed to extract these features. This also means that the visualization is not biased toward one specific finding, which would in turn mask other discoverable properties.

7.1.2 Exploration across zoom levels and light directions

The Copper/Silver mixture data set depicted in Fig. 3 consists of a copper bar with several embedded silver spheres. The resulting material exhibits ridges and small dislocations on the faces, which cannot be detected in the overview with standard rendering. Note the scale consistency of the small features for both super-sampling and our S-NDF technique. This data set also shows how S-NDFs help in the analysis workflow: Even data sets of moderate size usually are explored starting from an overview, where the data is visible as a whole, and a mismatch between available screen pixels and number of atoms in the simulation leads to distracting noise and aliasing artifacts. The scale-consistency of our approach allows domain scientists to decide where potentially interesting features are located, thus directing the exploration.

7.1.3 Exploration via the S-NDF explorer widget

As shown already in Fig. 12, spatially oriented features like ridges, dislocations or other defects in regular structures (as in crystalline solids) as well as density fluctuations can be highlighted with an appropriate transfer function. Discussion with a domain scientist revealed much interest in this approach, since the computational detection especially of dislocations is an extremely expensive procedure.

Since different transfer functions highlight differently oriented features, the ability to change the lighting environment on-the-fly allows interactive exploration of lighting-dependent features. This lighting-based feature exploration interaction is novel in the context of thermodynamics and material science, as state-of-the-art visualization could not offer the rendering and lighting quality our approach introduces.

One way of altering the transfer function is directly altering the light direction by orbiting the light source around the data set (see, for example, $[T_1, T_3]$ in Fig. 2). To highlight multiple features with different orientation, multiple light sources are required. If each light source has a different color, the resulting features can also be distinguished.

To enable a more intuitive exploration of features with a consistent orientation, we have implemented two additional ways of manipulating the transfer function. Recall that the transfer function is visualized as a circular image, a projection of the frontal hemisphere. The direct manipulation now allows free-hand painting of arbitrary parts of the transfer function, thus coloring the accordingly oriented flat features. Both the transfer function visualization and particle rendering are updated interactively. An example of a hand-drawn transfer function is shown in the supplemental material. Analogously to scale-consistent transfer functions for volumes [27], accurate filtering is indispensable for unfiltered hand-drawn transfer function, as the sparsely sampled or linearly filtered normals would result in lookups that do not reflect the actual sub-pixel structures present in the data.

S-NDF explorer widget. To further improve this interaction, we propose an S-NDF explorer widget (see Fig. 13). This widget allows direct painting of the two most meaningful patterns we have observed so far directly onto the transfer function. One pattern is the central circle that represents all normals pointing towards the viewer, to an extent that can be changed by scaling the radius of this central circle. The other pattern is an arbitrary number of pie slices that can be rotated around the center, thus capturing a certain orientation, with a tolerance that can be set by enlarging or shrinking the angle the slices cover.

Varying particle radii. In addition to the intrinsic properties of our approach, we noticed how the manipulation of the physically motivated particle radius can generate additional insight. The shock waves in regular materials like metal or other crystal lattices are occluded if the material is too dense, but reducing the particle size makes the internal structures easily visible. Structure can be perceived using our rendering approach since any larger-scale patterns (regular and irregular) statistically coagulate to specific normal orientations, which will then be captured by the S-NDFs. Conversely, with conventional rendering, the following (wrong) alternatives exist: sub-pixel spheres occupy the whole pixel all the same (material is gained across scales) or disappear (material is lost across scales).

The inverse interaction, that is, enlarging particle radii beyond the, e.g., van-der-Waals radius or lattice constant, can be used to merge neighboring particles to form surfaces that can be used to detect low-frequency/large-scale features. As an example the ablation crown of the laser ablation data set (Fig. 8 (top)) is much more evident with increased particle sizes, whereas the shock waves caused by the laser within the aluminum lattice are accentuated using reduced radii (see Fig. 13 (d)).

7.2 Performance

We first discuss general performance considerations in terms of complexity, and then report and discuss actual measured performance.

7.2.1 Complexity

It is important to note that although our implementation uses GPU ray casting for sampling the particle data, the big benefit from using

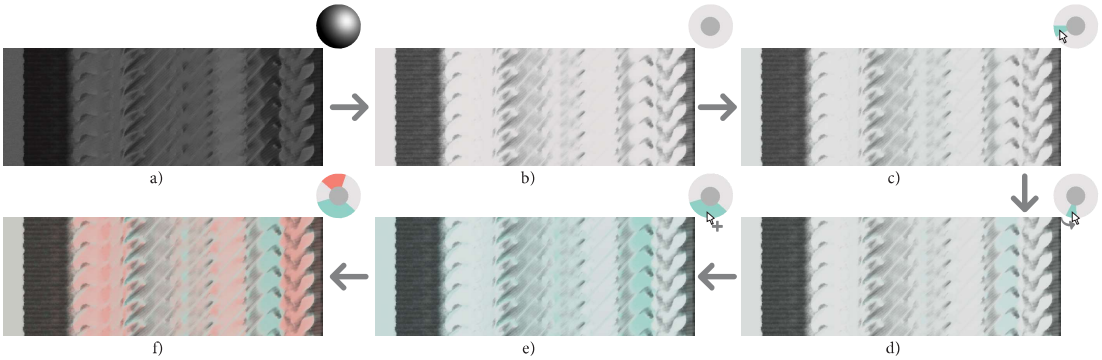

Fig. 13. **The S-NDF explorer widget** allows exploring the normal distributions of a data set such as the Instanced Laser shown here. a) rendering of the data set under the transfer function shown on the top right, b) the user activates the widget, which starts out as a faded gray transfer function topped with a dark gray region to highlight normal vectors pointing towards the viewer, c) the user creates a 'slice' to adjust the transfer function giving a color, cyan, to a region of interest, d) the user drags the slice to look for other regions of interest, e) the user enlarges the slice to highlight more features using the same color, f) the user adds another slice to highlight a region with normal orientation opposite to that of the first region of interest.

S-NDFs is independent of the actual ray casting technique that is used. This also means that the major performance benefit is independent of the exact performance of the ray caster implementation that is used.

For the following considerations, we denote the number of pixels that should be computed by N_P, and the cost of intersecting one viewing ray with the particle data set by T_R. We furthermore denote the number of bins in each S-NDF by P_L^2, and the number of samples that should be computed for a specific desired rendering quality by N_S.

Re-lighting complexity. The total cost of re-lighting using ray casting without S-NDFs is $\mathcal{O}(N_P \cdot T_R \cdot N_S)$. In contrast, the cost for re-lighting using S-NDFs is $N_P \cdot P_L^2 = \mathcal{O}(N_P)$. The number P_L^2 is a constant, and therefore does not differ between different levels of detail. In stark contrast, N_S is not a constant, and in order to achieve scale-consistent rendering (or, simply: high quality) it must be increased exponentially depending on the level of detail. For example, whenever the zoom factor halves (zooming out), N_S must be quadrupled in order to keep the rendering quality constant. This drastic difference in scalability does not depend on the actual ray casting implementation.

7.2.2 Performance measurements

We evaluate the performance of our approach using large scale particle data sets consisting of about 500,000 to almost 200 million particles. Our data sets and their sizes are listed in Table 3.

All our performance tests were run on a machine running Windows 7 with two Intel Xeon X5550 2.67 GHz processors with a Geforce GTX TITAN X (Pascal) with 12 GB graphics memory.

The basic performance for computing one sample per pixel is given in Table 3. The sampling performance is similar to other GPU ray casting approaches, e.g., using MegaMol [11]. While updating the S-NDFs decreases the performance slightly, our approach still runs at very interactive frame rates. After the iterative process of sampling is finished, the number of particles does not affect the rendering performance anymore, since the particle data is not required for rendering until the cache is invalidated.

Table 2 gives more detailed performance metrics. We report the time taken to perform an exploratory operation (re-lighting), and a navigational operation (zooming-out), on all of our data sets using either an S-MIP without S-NDFs (i.e., simply storing a color per pixel), as well as the full-featured S-MIP storing an S-NDF per pixel. We report the time taken to re-light an image after it was sampled an arbitrary number of times (256 in our experiments). As shown in Table 2, the re-lighting operation takes significantly less time using our proposed S-MIP + S-NDF approach, compared to ray casting without S-NDFs. This is clearly due to having to re-sample the data 256 times per pixel in the standard approach after the lighting properties are changed, in order to again reach the same image quality that is provided by the S-NDF approach. For the zoom-out operation, we report the time needed to compute a 'zoomed-out' image, given we already have computed the corresponding finer image in the S-MIP. We report the time taken for down-sampling without updating the S-MIP cache, and we report the time exclusively required to update the S-MIP cache with the down-sampled level. It is evident that zooming out using the S-MIP without S-NDFs takes less time compared to zooming out using the S-MIP with S-NDFs. This is because the former approach stores just four values (RGBA) per pixel, while each S-NDF approach stores 64 values (bins) per pixel in our implementation. However, this small performance loss is very minor compared to the capability of S-NDFs for interactive re-lighting operations.

7.3 Memory Consumption

The advantages of our approach come at a memory consumption cost. The memory cost of our approach, C_T (in bytes), can be computed as $C_T = C_{SC} + C_D$, where C_D is the memory footprint of the data, and C_{SC} is the memory cost of our S-MIP cache. The data cost C_D results from $C_D = n_D \cdot 4 \cdot \text{sizeof(float)}$, where n_D is the number of particles in the data set. We store four values for each particle, namely its (x,y,z) position and its radius. Each value is stored in a floating point data type of 4 bytes. The S-MIP cache cost, C_{SC} is computed as $C_{SC} = n_T \cdot T_L^2 \cdot P_L^2 \cdot \text{sizeof(} D_T \text{)}$, where n_T is the number of tiles that the cache should accommodate, T_L^2 is the number of pixels per tile, P_L^2 is the number of elements per pixel, and D_T the data type used in the S-MIP structure. For our experiments, the cost $C_{SC} = 1936 \cdot 64^2 \cdot 8^2 \cdot 4 \approx 2\ GB$.

8 Conclusions and Future Work

Our normal distribution cache for large-scale particle rendering is capable of drastically improving the image quality of current GPU ray casting techniques using progressive refinement while maintaining interactive frame rates even for particle data consisting of millions of particles. We have shown that large-scale particle data exhibit complex sub-pixel features that have to be represented accurately across multiple scales to allow for consistent multi-resolution rendering even for small output resolutions. This feature-consistent representation is essential for the effective exploration of particle data. Our progressive sampling converges to an accurate solution within a short amount of time. Additionally we have shown new possibilities for exploratory analysis by using different transfer functions and light interactions in contrast to solely relying on the commonly used Blinn-Phong BRDF.

Future work. Two very interesting avenues for future work are (1) Combining S-NDFs with some form of global illumination. Specifically, ambient occlusion should be easy to integrate. (2) We are very interested in exploring the use of S-NDFs for in situ visualization. Since the S-MIP tiles store S-NDFs in image space, they could be generated on a server and transmitted to a client for deferred lighting, zooming, and panning, without requiring access to the actual particle data.

Acknowledgments

This work was funded by King Abdullah University of Science and Technology (KAUST), and Deutsche Forschungsgemeinschaft (DFG) as part of SFB 716.

REFERENCES

[1] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *Proc. of SC'14*, pages 424–434, 2014.

[2] O. Arikan and U. Güdükbay. An algorithm for progressive raytracing. In *Proc. of ADVIS 2000*, pages 248–256, 2000.

[3] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane. Texmol: interactive visual exploration of large flexible multi-component molecular complexes. In *Proc. of IEEE Visualization 2004*, pages 243–250, 2004.

[4] J. F. Blinn. Models of light reflection for computer synthesized pictures. In *Proc. of ACM SIGGRAPH*, pages 192–198, 1977.

[5] E. Bruneton and F. Neyret. A survey of nonlinear prefiltering methods for efficient and accurate surface shading. *IEEE Transactions on Visualization and Computer Graphics*, 18(2):242–260, 2012.

[6] J. Dupuy, E. Heitz, J.-C. Iehl, P. Poulin, F. Neyret, and V. Ostromoukhov. Linear efficient antialiased displacement and reflectance mapping. *ACM Transactions on Graphics*, 32(6):211:1–211:11, 2013.

[7] W. Eckhardt, A. Heinecke, R. Bader, M. Brehm, N. Hammer, H. Huber, H.-G. Kleinhenz, J. Vrabec, H. Hasse, M. Horsch, M. Bernreuther, C. Glass, C. Niethammer, A. Bode, and H.-J. Bungartz. 591 TFLOPS Multi-trillion Particles Simulation on SuperMUC. In *Proc. of Supercomputing (ISC) 2013*, pages 1–12. 2013.

[8] M. Falk, M. Krone, and T. Ertl. Atomistic visualization of mesoscopic whole-cell simulations using ray-casted instancing. *Computer Graphics Forum*, 32(8):195–206, 2013.

[9] A. Fournier. Normal distribution functions and multiple surfaces. In *Graphics Interface Workshop on Local Illumination*, pages 45–52, 1992.

[10] R. Fraedrich, J. Schneider, and R. Westermann. Exploring the Millennium Run–Scalable Rendering of Large-Scale Cosmological Datasets. *IEEE Trans. on Visualization and Computer Graphics*, 15(6):1251–1258, 2009.

[11] S. Grottel, M. Krone, C. Müller, G. Reina, and T. Ertl. MegaMol–A Prototyping Framework for Particle-based Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 21(2):201–214, 2015.

[12] S. Grottel, G. Reina, C. Dachsbacher, and T. Ertl. Coherent culling and shading for large molecular dynamics visualization. In *Proc. of Eurovis 2010*, pages 953–962, 2010.

[13] S. Gumhold. Splatting illuminated ellipsoids with depth correction. In *Proceedings of VMV 2003*, pages 245–252, 2003.

[14] M. Hadwiger, R. Sicat, J. Beyer, J. Krüger, and T. Möller. Sparse pdf maps for non-linear multi-resolution image operations. *ACM Transactions on Graphics*, 31(6):133:1–133:12, 2012.

[15] C. Han, B. Sun, R. Ramamoorthi, and E. Grinspun. Frequency domain normal map filtering. *ACM Trans. on Graphics*, 26(3):28:1–28:12, 2007.

[16] M. Hopf and T. Ertl. Hierarchical splatting of scattered data. In *Proc. of IEEE Visualization 2003*, pages 433–440, 2003.

[17] W. Jakob. *Light Transport on Path-Space Manifolds*. PhD thesis, Cornell University, 2013.

[18] T. Klein and T. Ertl. Illustrating Magnetic Field Lines using a Discrete Particle Model. In *Proc. of VMV 2004*, pages 387–394, 2004.

[19] A. Knoll, I. Wald, P. A. Navrátil, A. Bowen, K. Reda, M. E. Papka, and K. Gaither. RBF volume ray casting on multicore and manycore cpus. *Computer Graphics Forum*, 33(3):71–80, 2014.

[20] M. Le Muzic, L. Autin, J. Parulek, and I. Viola. cellVIEW: a Tool for Illustrative and Multi-Scale Rendering of Large Biomolecular Datasets. In *Proc. of VCBM 2015*, pages 61–70, 2015.

[21] N. Lindow, D. Baum, and H.-C. Hege. Interactive rendering of materials and biological structures on atomic and nanoscopic scale. *Computer Graphics Forum*, 31(3):1325–1334, 2012.

[22] C. Niethammer, S. Becker, M. Bernreuther, M. Buchholz, W. Eckhardt, A. Heinecke, S. Werth, H.-J. Bungartz, C. W. Glass, H. Hasse, J. Vrabec, and M. Horsch. ls1 mardyn: The massively parallel molecular dynamics code for large systems. *Journal of Chemical Theory and Computation*, 10(10):4455–4464, 2014.

[23] M. Olano and D. Baker. Lean mapping. In *Proc. of Symposium on Interactive 3D Graphics and Games*, pages 181–188, 2010.

[24] J. Painter and K. Sloan. Antialiased ray tracing by adaptive progressive refinement. In *Proc. of ACM SIGGRAPH*, pages 281–288, 1989.

[25] G. Reina and T. Ertl. Hardware-accelerated glyphs for mono- and dipoles in molecular dynamics visualization. In *Proc. of Eurovis 2005*, pages 177–182, 2005.

[26] S. Rizzi, M. Hereld, J. Insley, M. E. Papka, T. Uram, and V. Vishwanath. Large-Scale Parallel Visualization of Particle-Based Simulations using Point Sprites and Level-Of-Detail. In *Proc. of EGPGV*, pages 1–10, 2015.

[27] R. Sicat, J. Krüger, T. Möller, and M. Hadwiger. Sparse PDF volumes for consistent multi-resolution volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2417–2426, 2014.

[28] C. Sigg, T. Weyrich, M. Botsch, and M. Gross. Gpu-based ray-casting of quadratic surfaces. In *Proc. of Point-Based Graphics*, pages 59–65, 2006.

[29] P. Tan, S. Lin, L. Quan, B. Guo, and H. Shum. Filtering and rendering of resolution-dependent reflectance models. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):412–425, 2008.

[30] A. Tikhonova, C. Correa, and K.-L. Ma. Visualization by proxy: A novel framework for deferred interaction with volume data. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1551–1559, 2010.

[31] M. Toksvig. Mipmapping normal maps. *Journal of Graphics Tools*, 10(3):65–71, 2005.

[32] E. Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, 1997.

[33] I. Wald, G. P. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Günther, and P. Navrátil. OSPRay–A CPU Ray Tracing Framework for Scientific Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):931–940, 2017.

[34] I. Wald, A. Knoll, G. P. Johnson, W. Usher, V. Pascucci, and M. E. Papka. CPU Ray Tracing Large Particle Data with Balanced P-k-d Trees. In *Proc. of IEEE Scientific Visualization*, pages 57–64, 2015.

[35] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. Microfacet models for refraction through rough surfaces. In *Proc. of Eurographics Symposium on Rendering*, pages 195–206, 2007.

[36] L. Williams. Pyramidal parametrics. In *Proc. of ACM SIGGRAPH*, pages 1–11, 1983.

[37] J. Woodring, J. Ahrens, J. Figg, J. Wendelberger, S. Habib, and K. Heitmann. In-situ sampling of a large-scale particle simulation for interactive visualization and analysis. In *Proc. of Eurovis*, pages 1151–1160, 2011.

[38] L.-Q. Yan, M. Hašan, W. Jakob, J. Lawrence, S. Marschner, and R. Ramamoorthi. Rendering glints on high-resolution normal-mapped specular surfaces. *ACM Transactions on Graphics*, 33(4):116:1–116:9, 2014.

[39] Y. Ye, R. Miller, and K.-L. Ma. In situ pathtube visualization with explorable images. In *Proc. of EGPGV 2013*, pages 9–16, 2013.

[40] H. Younesy, T. Möller, and H. Carr. Improving the quality of multi-resolution volume rendering. In *Proc. of Eurovis*, pages 251–258, 2006.