# Probabilistic Occlusion Culling using Confidence Maps for High-Quality Rendering of Large Particle Data

Mohamed Ibrahim, Peter Rautek, Guido Reina, Marco Agus, and Markus Hadwiger

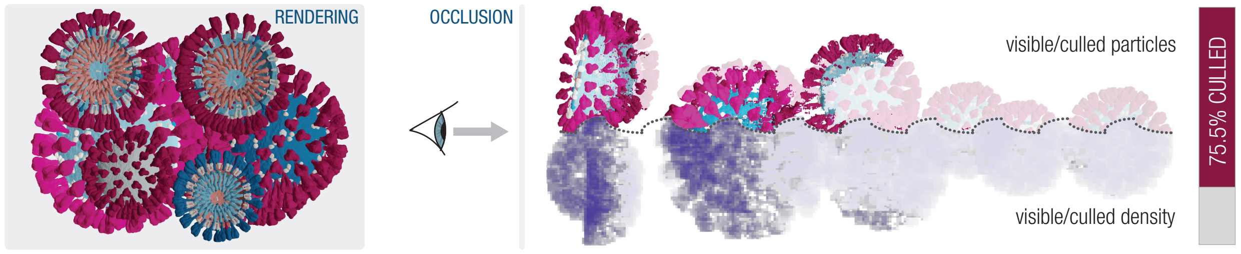

Fig. 1. Rendering a synthesized SARS-CoV-2 virus with atomistic resolution (∼40M particles). (Left) Main view rendered with 75.5% of particles culled. (Right) We estimate occlusion probabilities using particle density functions sampled from a coarse particle density volume. From front to back (left to right), both the number of particles and the volume density decrease due to detected occlusion.

**Abstract**—Achieving high rendering quality in the visualization of large particle data, for example from large-scale molecular dynamics simulations, requires a significant amount of sub-pixel super-sampling, due to very high numbers of particles per pixel. Although it is impossible to super-sample all particles of large-scale data at interactive rates, efficient occlusion culling can decouple the overall data size from a high effective sampling rate of visible particles. However, while the latter is essential for domain scientists to be able to see important data features, performing occlusion culling by sampling or sorting the data is usually slow or error-prone due to visibility estimates of insufficient quality. We present a novel probabilistic culling architecture for super-sampled high-quality rendering of large particle data. Occlusion is dynamically determined at the sub-pixel level, without explicit visibility sorting or data simplification. We introduce confidence maps to probabilistically estimate confidence in the visibility data gathered so far. This enables progressive, confidence-based culling, helping to avoid wrong visibility decisions. In this way, we determine particle visibility with high accuracy, although only a small part of the data set is sampled. This enables extensive super-sampling of (partially) visible particles for high rendering quality, at a fraction of the cost of sampling all particles. For real-time performance with millions of particles, we exploit novel features of recent GPU architectures to group particles into two hierarchy levels, combining fine-grained culling with high frame rates.

**Index Terms**—Large-scale particle data, sub-pixel occlusion culling, super-sampling, anti-aliasing, coverage, probabilistic methods

## 1 Introduction

The steadily growing size of simulation data leads to significant novel challenges. One area where this is particularly apparent is in molecular dynamics (MD) simulations. Such data sets comprise millions to billions of particles, and practitioners increasingly require high rendering quality and accuracy to capture important data features, in addition to high performance. However, the particles comprising large-scale particle simulations are very small, and therefore the number of particles per pixel in visualizations is typically much larger than for many other types of data. In this context, the standard approach of sampling pixels with only a few samples (rays) thus often leads to significant aliasing and missing features, because the data are severely undersampled.

Many approaches therefore employ pre-computed levels of detail, or abstractions. This typically means skipping groups of particles that are small, or aggregating particles into simplified proxy geometries. This not only removes high-frequency detail from the visualization, but can also bias the analyses of domain scientists toward data that is simplified for the current scale of exploration. This poses a problem for accurate exploratory analysis, as it limits the detail level at which data can be understood, and the assessment where to continue exploring. Moreover, for data with similar features on different scales, simplification makes it unnecessarily hard to understand which scale is currently shown.

Particles in large-scale data sets are often distributed heterogeneously in space, e.g., depending on the type of matter being simulated. In molecular dynamics, particles in a solid/crystalline state tend to be tightly packed, as do liquid particles. In contrast, particles in a gas phase often lead to sparsely populated regions, e.g., in the left-hand side of Fig. 2. In such regions, the particles require accurate super-sampling without overestimating their potential for occluding, which would lead to visible particles not being sampled. The right-hand side, however, is densely packed, and most particles there in fact do not contribute to the visualization, providing significant opportunities for *occlusion culling*. (See Table 1 for an overview of the terminology that we use in this paper.) Efficient and accurate occlusion culling must take these properties into account, but particles that are smaller than a pixel pose a major challenge for dynamic (on-the-fly) occlusion culling. If state-of-the-art occlusion culling is employed for such data, small particles lead to overestimation of potential occlusion: the depth of a sample is not representative for an entire pixel. This holds unless occlusion is also super-sampled on top of super-sampling for rendering.

The major kinds of state-of-the-art interactive molecular visualization approaches and their drawbacks are: (1) Reducing individual cells of an acceleration structure, or even entire molecule bounding boxes, to a single pixel when their projection is small enough [7]. This skips details of large parts of the scene. (2) Clustering and simplifying the

• *Mohamed Ibrahim, Peter Rautek, and Markus Hadwiger are with King Abdullah University of Science and Technology (KAUST), Visual Computing Center, Thuwal, 23955-6900, Saudi Arabia. E-mail: { mohamed.ibrahim, peter.rautek, markus.hadwiger } @kaust.edu.sa.*
• *Guido Reina is with Visualization Research Center (VISUS) at the University of Stuttgart, Germany. E-mail: guido.reina@visus.uni-stuttgart.de.*
• *Marco Agus is with College of Science and Engineering, Hamad Bin Khalifa University, Qatar Foundation, Doha, Qatar. E-mail: magus@hbku.edu.qa.*

*Manuscript received 31 Mar. 2021; 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*

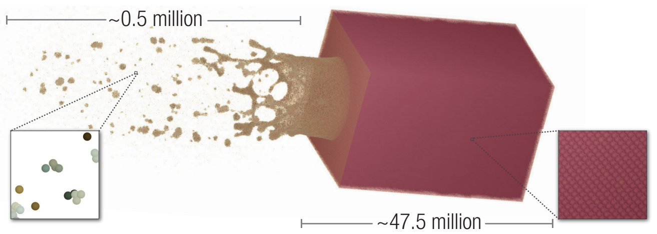

Fig. 2. **Heterogeneous particle distributions.** Often, only a small part of a data set, e.g., the $\sim 0.5$ million particles in the crown (left), and the faces of the aluminum block (right), contribute to the actual visualization.

data to fit the current zoom level [27, 32], filtering out finer details. In terms of performance and perception, all of these approaches scale well, but at the cost of lost accuracy. (3) Progressive super-sampling achieves high accuracy after convergence, but at a significant cost [37]. For fast re-lighting, the result of costly super-sampling can be cached in normal distribution functions [22]. This captures details well, but super-sampling needs to be re-computed for every new rotated view.

**Main goals.** We build on the basic premise that accurate, high-quality visualization of the intricate details in large-scale particle data (Fig. 1 and Fig. 2) is impossible without extensive super-sampling per pixel. However, doing this for all particles in large data sets was so far prohibitive at interactive rates. We therefore target improving performance by concentrating most of the sampling effort on the *visible* parts of the data only. We propose a novel architecture for *dynamic occlusion culling* that is probabilistic, uses progressive refinement, and also takes current GPU architectures into account for high performance. Our approach performs both occlusion culling and high-quality rendering via progressive sampling, adding more and more samples from frame to frame, without requiring any data simplification or approximations.

**Contributions.** The major contributions of this paper are:

(1) A novel probabilistic occlusion culling architecture (Fig. 4) that goes hand in hand with progressive super-sampling of particle data containing intricate detail. While we start progressively super-sampling the scene, we likewise progressively build up visibility information that enables culling more and more parts of the scene that become known to be occluded with high confidence. We combine fine culling granularity with fast rendering via a two-level object space subdivision, grouping particles into *nodes* and *meshlets*, and using recent GPU features.

(2) Instead of computing sub-pixel coverage via explicit discretization, as in Fig. 3 (left), we track a *representative pixel depth* ($d$), and probabilistically estimate sub-pixel coverage, as in Fig. 3 (right), to determine *confidence* ($C$) in the current value of $d$. We track depth and confidence information $(d, C)$ in a multi-resolution image pyramid, the *depth confidence map*, enabling probabilistic culling.

(3) To enable the accurate estimation of depth and confidence updates, in addition to the particle data we maintain and progressively update a coarse 3D *particle density volume*, from which we extract *particle density functions* along view rays on-the-fly via ray casting. This enables a novel probabilistic computation of visibility estimates.

We show that our probabilistic culling converges quickly, enabling extensive super-sampling of *visible* particles at real time rates, while significantly reducing the overall number of particles that are sampled.

## 2 Related Work

Super-sampling for high-quality rendering is a standard technique in graphics and visualization. Many approaches subdivide pixels into a sub-pixel grid, and perform sampling on this finer grid (see Fig. 3 (left)). An early seminal work is the A-buffer [3]. The accumulation buffer [19] sums color samples per pixel in multiple rendering passes, reusing the standard depth buffer for correct visibility without sub-pixel depth storage overhead. However, this accumulation buffer cannot be used for sub-pixel-accurate culling. We also do not store or use an explicit sub-pixel grid (see Fig. 3 (right)), but overall our approach is very different and focuses on probabilistic estimates that enable sub-pixel-accurate culling. We also use an "accumulation buffer," but this is not related to the concept just described [19]. Our accumulation buffer performs a probabilistic accumulation of *confidence* in depth values.

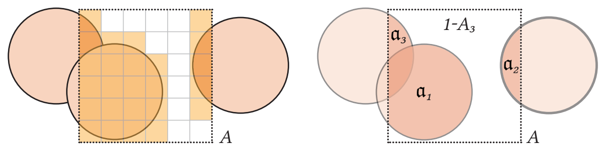

Fig. 3. **Sub-pixel coverage.** We do not discretize or store sub-pixel samples on a grid (left), but instead probabilistically estimate the accumulated sub-pixel coverage $A_{n(k)}$ (right) after $k$ samples, out of which $n(k)$ are from unique particles (here, $n = 3$), where $n$ is also probabilistically estimated. The *confidence* $C(k)$ for the pixel's *representative depth* $d$ then corresponds to the covered area, for which depth $\leq d$ is guaranteed.

**Occlusion culling.** We perform hierarchical occlusion testing, inspired by hierarchical occlusion maps (HOMs) [41] and the hierarchical z buffer [13]. However, we emphasize that instead of aggregating depth information of pixels or sub-pixels, our "finest resolution" occlusion information is fully probabilistic and not an aggregation of discretized sub-pixel data. Bittner et al. [2] and Cohen-Or et al. [5] provide comprehensive overviews of occlusion culling methods. Many approaches exploit GPU hardware occlusion queries, e.g., in OpenGL [4], which are challenging to use because they are asynchronous. Various methods reduce stalls in the occlusion pipeline, such as coherent hierarchical culling [1], and later improvements [18, 29, 30]. On top of these methods, fast hierarchical culling can be built [13, 41]. To achieve scalability to large scenes, hybrid image/object-space methods use spatial scene partitionings and hierarchical data structures, amortizing occlusion query and culling cost over many primitives [10, 31]. Proximity information of occluding triangles can also be used to infer good occluders from neighboring triangles [40]. A more recent CPU-based approach for early occlusion culling was presented by Hasselgren et al. [20].

**Particle rendering.** A common approach to render particle data is to use GPU-computed glyphs [17], where the particle geometry is replaced by implicit descriptions that are rendered as billboards and then filled in, for example by computing ray-sphere intersections for each pixel in the billboard's screen space projection. We render particles based on the same principle. Large-scale particle data can also be rendered via volume rendering, e.g., via RBF volume ray casting [23], or ray tracing for molecular visualization of ball-and-stick models [24], also in ultra-resolution immersive environments [33]. Other approaches for molecular visualization that exploit a grid and instancing for large particle data have been proposed. Lindow et al. [28] traverse a grid in layers to just require a single reference per atom even when it spans multiple cells, additionally optimizing normal computation to reduce aliasing. Falk et al. [7] use a grid containing local grids for entire molecules to further optimize intersection testing depending on screen-space size. Grottel et al. [16] perform occlusion culling of particle scenes on two levels, in both cases against a (hierarchical) depth buffer of the previous frame. On the coarse level, they check whether the bounding boxes of large chunks of data are occluded. On the fine level, they do per-glyph occlusion culling against the appropriate depth hierarchy level in the vertex shader, that is, after the data is streamed (chunk-wise) to the GPU. This produces good results in terms of occlusion and frame rates. However, this technique always streams the data

Table 1. **Terminology and main concepts** used in this paper.

| term | explanation |
|---|---|
| occlusion culling | determining occluded subsets of the data. here: to exclude groups of particles from super-sampling |
| sub-pixel super-sampling | more than one ray per pixel |
| sub-pixel coverage | percentage of occluded pixel area |
| depth confidence; $C$ | sub-pixel area with depth $\leq d$ |
| representative depth; $d$ | depth valid for sub-pixel area $C$ |
| depth confidence map | $(d, C)$ tracked in image pyramid |
| particle density volume; $\mathscr{D}$ | 3D volume of local particle density |
| particle density function; $D(t)$ | density along ray; extracted from $\mathscr{D}$ |
| meshlet | a small group of particles (used in Vulkan *mesh shaders*) |
| node | a group of *meshlets* |

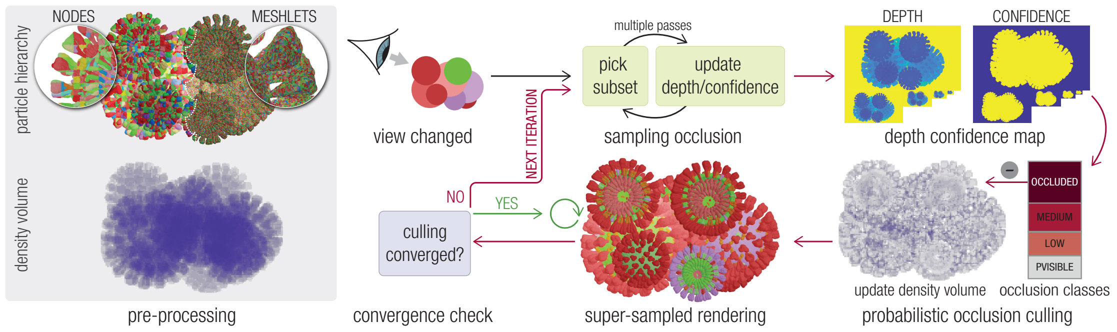

Fig. 4. **Probabilistic occlusion culling architecture**. To facilitate culling, particles are grouped together in a two-level *object space hierarchy* comprising *nodes* and *meshlets* (top left). For probability estimates, we also compute a coarse *particle density volume* (bottom left). For each new view, we sample subsets of particles not (yet) known to be occluded, and update a novel *image space hierarchy* for culling: *depth confidence maps*. Confidence values are updated probabilistically, requiring particle density estimates along rays through the density volume, determining an *occlusion class* for each meshlet, from *visible* to *occluded*. Rendering is performed by super-sampling only the meshlets that are not known to be occluded, with culling as well as super-sampling performed progressively. After culling has converged, we continue super-sampling for higher rendering quality.

to the GPU and is bandwidth-limited for large data: To use occlusion queries, the respective data has to be uploaded to the GPU. The authors have implemented a caching mechanism such that relevant batches of particles are kept on the GPU for rendering, but for each batch with unclear visibility uploading is required. Wang et al. [39] introduce a particle-specific metric to determine occlusion on the CPU.

**Ray tracing frameworks.** A recent CPU ray tracing framework is OSPRay [36], built on Embree [38], using a standard bounding volume hierarchy (BVH). The Particle Kd-Tree (P-k-d) is an extension for particle data [37]. Similar approaches are available for GPUs, for example the NVIDIA RTX hardware ray tracing capabilities. This approach also requires an acceleration structure to be efficient, which can only have two levels, of which the bottom one has bounding information [35]. A more recent approach investigates nesting of P-k-ds inside a standard BVH to balance memory requirements and resulting performance [12] and benchmarks implementations using OptiX, OSPRay and OpenGL.

**Point-based rendering.** Particle rendering and point-based rendering share several of the basic approaches, and they manage occlusion in similar ways. For example, layered point clouds [9, 11] are organized top-down in chunks through a hierarchical data structure, exploiting the chunks together with frame-to frame coherence for culling.

## 3 Occlusion Culling Architecture Overview

Fig. 4 depicts an overview of the major components of the pipeline of our probabilistic occlusion culling architecture. Table 1 gives an overview of our terminology. In the following, we first summarize the major ideas that our probabilistic approach is built on, and then describe each of the pipeline stages depicted in Fig. 4 in more detail.

All input particles are spatially grouped into a simple two-level hierarchy (Sect. 3.1), comprising small groups of particles stored as *meshlets*, and meshlets further grouped into *nodes* (Fig. 4, top left). Moreover, we compute an auxiliary 3D *particle density volume* to facilitate accurate probability estimates (Fig. 4, bottom left).

The main idea is to sample the scene progressively, over several iterations of the right-hand side of our pipeline, with fully *dynamic occlusion culling*, consisting of (1) *dynamic sampling of occlusion* (Sect. 3.2), (2) probabilistic updates of *depth confidence maps* (Sect. 3.3), and (3) probabilistic occlusion culling using the depth confidence to classify groups of particles (meshlets) into one of four *occlusion classes*; culling the particles in the class $\mathbb{O}$ (*occluded*). These particles are known to be *occluded with high confidence*. After each iteration of determining occlusion classes, we dynamically update the particle density volume by removing the contribution of particles that were put into $\mathbb{O}$. These updates significantly improve our probabilistic confidence estimates.

Our approach converges quickly, separating *occluded* ($\mathbb{O}$) particles from *potentially visible* ($\mathbb{P}$) particles , which enables extensive super-sampling of only those remaining particles at real time rates (Sect. 3.5).

### 3.1 Pre-Processing and Object Space Data Structures

**Object space particle hierarchy.** For efficient hierarchical culling, and to exploit recent GPU architectures, we do not perform occlusion tests at the granularity of single particles. We group particles into two hierarchy levels in a simple *view-independent* pre-processing step: Small groups (e.g., 16) of nearby particles are grouped into *meshlets* (corresponding to GPU *mesh shaders*), and nearby meshlets are grouped further into *nodes*. While this significantly increases efficiency, the small size of meshlets preserves fine culling granularity (Fig. 5).

**Particle density volume.** We generate a low-resolution 3D particle density volume that is used to determine a *per-pixel particle density function* for each view ray via ray casting. It is needed to allow accurate estimation of occlusion probabilities for each sample.

### 3.2 Dynamic Sampling of Occlusion

We sample and improve occlusion estimates over multiple iterations, until sufficient convergence is reached (Fig. 4, red cycle). In each iteration, we perform multiple passes of choosing a subset of potential occluders (meshlets) to sample occlusion (Fig. 4, top center). In each pass, we generate one sub-pixel depth sample per pixel via rasterization.

A major motivation for progressively sampling occlusion is that, a priori, the best selection of particles for the purposes of occlusion is not known. In the simple case without sub-pixel accuracy, farther depths will be occluded by closer depths, which can be resolved by simple depth testing. However, with sub-pixel accuracy any sampled depth will in general not be representative for a whole pixel. For this reason, *samples with closer depth should not always lead to more occlusion*: Overall occlusion depends on the sub-pixel coverage corresponding to that depth. Therefore, if the closest depth is retained, *within* pixels a suboptimal choice of depth sampling will often occur. To solve this problem, our probabilistic approach (Sect. 4) dynamically estimates the *probability of a new sample improving occlusion*, and chooses the depth samples to retain according to this probability (Sect. 4.5).

### 3.3 Depth Confidence Maps

To avoid the overhead and discretization of explicit sub-pixels (Fig. 3 (left)), we estimate confidence values probabilistically, conceptually corresponding to continuous sub-pixel coverage (Fig. 3 (right)). For the depth value $d$ of each pixel in screen space, we estimate an associated *confidence* value $C$, as a probability $0 \leq C \leq 1$ that a randomly chosen, new sample in this pixel would hit a particle with a depth $\leq d$. (Smaller $d$ meaning closer to the camera.) We call this $d$ the *representative depth* of this pixel, with associated *depth confidence* $C$.

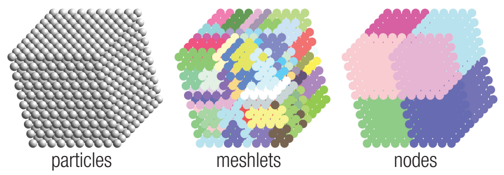

Fig. 5. **Meshlets and nodes**. For efficiency, we use two hierarchy levels to group particles. *Meshlets* (center) comprise a few particles (e.g., 16) each, and facilitate fine-grained culling. Meshlets are further grouped into *nodes* (right), to test a group of meshlets for occlusion in one step.

For each pixel, we maintain a pair $(d,C)$, with $d$ its representative depth, and $C$ its associated depth confidence. The details of this approach are described in Sect. 4.1 and Sect. 4.2. In fact, to progressively improve both confidence (higher is better) *as well as* representative depth (closer is better), we maintain two pairs $(d,C)$ per pixel: One in a *cull buffer*, which at any time is used for probabilistic culling, and another one in an *accumulation buffer* (Sect. 4.3). The latter is used to progressively build up confidence for a closer depth value with better potential occlusion (see Sect. 4.4 and Sect. 4.5). For hierarchical occlusion testing, we further maintain the cull buffer as a multi-resolution pyramid, similar to mipmaps or HOMs [41], as described in Sect. 4.6.

### 3.4 Probabilistic Occlusion Culling

**Occlusion classes.** Our architecture maintains and progressively updates a classification of meshlets into one of four *occlusion classes*. All meshlets start in the occlusion class $\mathbb{P}$ (*potentially visible*). As the confidence in the occlusion of a meshlet increases, it "moves up" in confidence, toward occlusion class $\mathbb{O}$ (*occluded*). See Sect. 5.

**Confidence-based culling.** In every iteration, the depth and confidence values $(d_{cull}, C_{cull})$ stored in the *cull buffer* are used to test meshlets that are not yet in occlusion class $\mathbb{O}$. This gives a new overall confidence value for each meshlet, which is its probability of being occluded. This in turn triggers an update of its occlusion class.

### 3.5 Super-Sampled Rendering

Occlusion culling, in each iteration of Fig. 4, results in a *potentially visible set* (PVS) of particles, which we define as all particles whose meshlets are not in occlusion class $\mathbb{O}$, with the PVS becoming smaller from iteration to iteration. Actual rendering is performed by progressively super-sampling all particles in all the meshlets in the PVS. The most important property of our pipeline is that this super-sampling effort is only spent on particles that are not occluded with sufficient confidence, as determined by culling against the depth confidence map.

## 4 Depth Confidence Maps

Considering sub-pixel occlusion (Fig. 3), the depth value of a pixel only corresponds to the sub-pixel samples computed so far. Future samples may have larger (farther) depth values. For each pixel we therefore estimate the *confidence* in the depth value currently associated with it, which we call its *representative depth*, with that associated confidence.

We estimate, improve, and use depth confidence probabilistically over multiple iterations (Fig. 6). We iterate over a sequence of samples of particles mapping to a given pixel, and for each sample we: (1) Estimate the sub-pixel area corresponding to all samples so far, for which the depth value is representative (Sect. 4.1 and 4.2). (2) Estimate whether the new sample's depth would increase particle occlusion—which also requires sufficient accumulation of confidence—using a novel probabilistic strategy for accepting the incoming depth (Sect. 4.5).

### 4.1 Probabilistic Sub-Pixel Coverage

We want to derive an estimate of how much of the sub-pixel area of a given pixel is covered so far, in an iteration over a sequence of particle samples. For this, we consider a given pixel of screen space size $A$, and perform the estimation of sub-pixel coverage iteratively, iterating from particle contribution to particle contribution (see Fig. 3 (right)).

**Representative depth.** A crucial concept in our method is the meaning of the depth value associated with a given pixel. Although we perform extensive super-sampling, we only store one depth value $d$ for every pixel, not one depth value per sub-pixel sample. We call this depth $d$ the *representative depth*, which is computed over the non-empty regions of the pixel (Fig. 3 (right)). The *farthest* visible depth anywhere marked in orange is $d$. In contrast, all regions shown in white have unknown depth, and (future) samples taken there can potentially hit particles with any depth. Thus, $d$ for a given pixel is a *representative conservative depth* for sub-pixel regions already known to be covered.

**Particle sampling without replacement.** We first consider iterating over all particles mapping to the pixel, with $0 < \mathfrak{a}_n \leq A$ the size of particle $n$ within the pixel footprint of area $A$. We denote the total pixel area that is covered jointly by $n$ particles, after $n$ iterations, by $A_n$. The fraction covered inside the pixel's area after $n$ iterations therefore is $A_n/A$. However, because particles often overlap in screen space, $A_n$ cannot be determined as $\sum_n \mathfrak{a}_n$. In order to estimate how much the next particle's area will contribute to the area $A_n$ of the pixel covered in iteration $n$, we estimate that the fraction of its area $\mathfrak{a}_n$, that will map to a sub-pixel area that was not already covered before, is given by the ratio of the sub-pixel area *not yet covered*, to the total area of the pixel, i.e., $(A - A_{n-1})/A$. We can view this estimate as computing the *expected value* of the newly covered area over a long series of trials of placing the same particle at different sub-pixel locations. This expected value will be $((A - A_{n-1})/A)\,\mathfrak{a}_n$. Thus, an iterative estimation of sub-pixel coverage $A_n$, starting with $n = 0$, is given by the recurrence relation

$$A_0 = 0, \quad A_n = A_{n-1} + \frac{A - A_{n-1}}{A}\,\mathfrak{a}_n. \tag{1}$$

We can simplify this by normalizing the pixel area, setting $A = 1$ (and thus $0 < \mathfrak{a}_n \leq 1$). If all particle sizes $\mathfrak{a}_n$ are equal ($\mathfrak{a}_n = \mathfrak{a}$, for all $n$), the closed-form solution of this recurrence for $n \geq 0$ is the function

$$A_n = 1 - (1 - \mathfrak{a})^n. \tag{2}$$

Since $0 < \mathfrak{a} \leq 1$, $\lim_{n\to\infty} A_n = 1$. If the $\mathfrak{a}_n$ are not equal, the equation above simply has to multiply $n$ different $(1 - \mathfrak{a}_n)$ terms accordingly.

**Particle super-sampling with replacement.** However, because our architecture employs extensive super-sampling of particles, this means that we are in fact sampling many particles multiple times, i.e., we sample *with replacement*. Thus, $n$ above must be the number of *unique* particles hit by $k$ samples, where $n \leq k$. Keeping track of individual sampled particles is too costly, so we cannot compute the exact number $n(k)$ of unique particles hit over $k$ iterations. However, for any $k \geq 0$, we can compute the *expected value* $\mathrm{E}\big(n(k)\big)$, via the recurrence relation

$$\mathrm{E}\big(n(0)\big) = 0, \tag{3}$$

$$\mathrm{E}\big(n(k)\big) = \mathrm{E}\big(n(k-1)\big) + \frac{N - \mathrm{E}\big(n(k-1)\big)}{N}. \tag{4}$$

Here, $N$ is the number of particles out of which we are choosing the $k$ samples, which in our case must be the total number of particles with a depth $\leq d$ (i.e., closer than the representative depth $d$ for which we are accumulating coverage). For any given pixel, we obtain the corresponding view-dependent number $N$ by performing volume rendering into a per-pixel particle density histogram, as described in Sect. 4.4. The closed-form solution of this recurrence relation, for $k \geq 0$, is

$$\mathrm{E}\big(n(k)\big) = \frac{1 - M^k}{1 - M}, \quad \text{with} \quad M := 1 - \frac{1}{N}. \tag{5}$$

We note that $\mathrm{E}\big(n(k)\big) \leq k$, for any $k \geq 0$, and $\lim_{N\to\infty} \mathrm{E}\big(n(k)\big) = k$, as expected. Fig. 7 illustrates $\mathrm{E}\big(n(k)\big)$ for different values of $k$ and $N$.

From Eq. 4, we also obtain the incremental contribution of the $k$'th sample to the expected number of unique particles $n(k)$, as

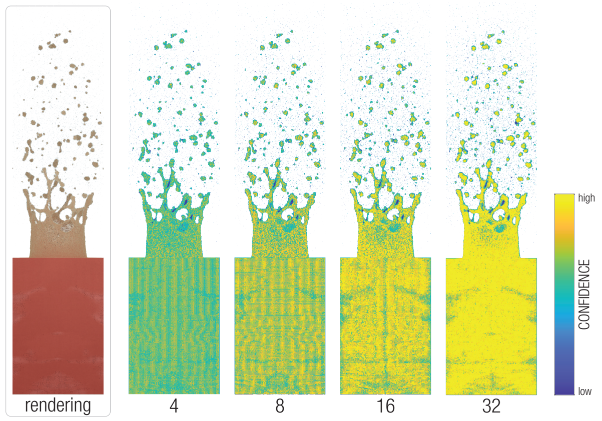

Fig. 6. **Confidence accumulation** over sampling iterations for the Laser Ablation data set of about 48 million particles. From left to right, confidence accumulates from iteration to iteration ($k = 4$ to $32$ samples).

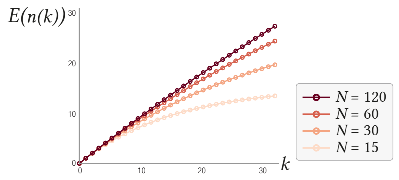

Fig. 7. **Expected value of unique particles**, $\mathrm{E}(n(k))$, for various sample counts $k$ (non-unique), and particle counts $N$ with smaller (closer) depth.

$$\Delta\mathrm{E}\big(n(k)\big) := \mathrm{E}\big(n(k)\big) - \mathrm{E}\big(n(k-1)\big) = 1 - \frac{\mathrm{E}\big(n(k-1)\big)}{N}. \quad (6)$$

**Particle sizes.** In the derivations above, we have considered particles of size $\mathfrak{a}_n$. However, for particles that we cannot consider to be extremely small compared to the pixel size $A$, we would first have to clip the particle against the pixel footprint in order to determine the actual size within the footprint. See Fig. 3 (right). In practice, we only do this probabilistically, by using effective particle sizes $\mathfrak{a}_n$ that are pre-computed as the *expected values* of the area of the particle *inside* the pixel, instead of the full particle size. We have also implemented exact clipping for comparison, and the differences in practice are negligible.

## 4.2 Depth Confidence

To obtain an interpretable probabilistic notion of *depth confidence* associated with the current *representative depth* of a given pixel, we interpret $(1 - A_n)$ as the probability that a randomly cast ray in the pixel's footprint, with its location determined by a uniform (spatial) distribution, hits a new particle at a sub-pixel location where no particle has been hit before. The confidence for *not* hitting such a particle that could have a farther depth than the representative depth is therefore $A_n$.

(1) Assuming *constant* particle size $\mathfrak{a}$, we can define the *depth confidence* $C(k)$ after the $k$'th sample using Eq. 2 for the estimated sub-pixel coverage $A_n$, with $n$ given by the expected value $\mathrm{E}\big(n(k)\big)$, as

$$C(k) := A_{n(k)} \approx 1 - (1-\mathfrak{a})^n, \quad \text{with} \quad n := \mathrm{E}\big(n(k)\big), \qquad (7)$$

with $\mathrm{E}\big(n(k)\big)$ given by Eq. 5. (2) For *non-constant* particle sizes $\mathfrak{a}_n$, using Eq. 1 (with $A = 1$) we iteratively estimate the *depth confidence*

$$C(k) := A_{n(k)} \approx 1 - \prod_{i=1}^{k} (1-\mathfrak{a}_i)^{n_i}, \quad \text{with} \quad n_i := \Delta\mathrm{E}\big(n(i)\big), \qquad (8)$$

with $\Delta\mathrm{E}\big(n(i)\big)$ given by Eq. 6. We note that in addition to Eq. 8 being only an expected value, it is also an approximation because the expected increments $\mathrm{E}\big(\Delta n(i)\big)$ for a particle of size $\mathfrak{a}_n$ get split over different particle sizes indiscriminately. However, Eq. 8 works well in practice.

Fig. 8 illustrates depth confidence curves for different particle sizes $\mathfrak{a}$, sample counts $k$, and different $N$. Fig. 6 depicts a real example.

**Sample count.** To be able to update the per-pixel $C(k)$ incrementally from sample to sample, in addition to the accumulated value $C$ we also need to store and update a *per-pixel* sample count $k$. See algorithm 1.

## 4.3 Double-Buffered Depth Confidence Updates

The representative depth and the associated depth confidence value for each pixel try to achieve a trade-off between two opposing goals:

In order to obtain the maximum possible amount of culling, we want an *as small as possible* representative depth for each pixel. However, this will only lead to sufficient culling if the associated confidence is also high enough: A small representative depth is not useful if its associated confidence is too low. Likewise, a high confidence is not useful if the depth is too large, because large depth values will usually occlude only a smaller number of particles. We quantify these considerations jointly, by estimating the number of occluded particles $OP(d,C)$ (see Eq. 12) for a given depth and confidence pair $(d,C)$.

Depth confidence is accumulated over occlusion sampling passes, which increase sub-pixel coverage and thus depth confidence. However, for a new representative depth value, the associated confidence needs a sufficient number of samples to build up. While this is happening, for some time the confidence value is low, and therefore $OP(d,C)$ is small.

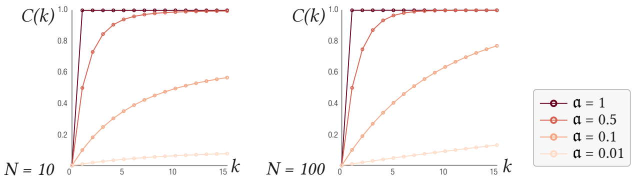

Fig. 8. **Depth confidence**, for the representative depth of a pixel, for various sample counts $k$, particle sizes $\mathfrak{a}$, and closer particle counts $N$.

We therefore maintain *double-buffered* depth and confidence maps, i.e., we maintain two buffers with one depth and confidence map each:

**Cull buffer.** The depth/confidence pair in the cull buffer, denoted by $(d_{cull}, C_{cull})$, is the one that so far has the largest occlusion value $OP(d_{cull}, C_{cull})$. Therefore, this is the buffer that we use for culling.

**Accumulation buffer.** The depth/confidence pair in the accumulation buffer, denoted by $(d,C)$, is used to try and accumulate confidence for a smaller, i.e., better, representative depth $d$ than the current $d_{cull}$. However, while confidence $C$ is accumulating due to new samples coming in progressively, the pair $(d_{cull}, C_{cull})$ will still be used for culling. Only when the accumulated number of occluded particles $OP(d,C)$ surpasses the number of occluded particles $OP(d_{cull}, C_{cull})$, we perform the assignment $(d_{cull}, C_{cull}) = (d,C)$, overwriting the previous values in the cull buffer. The accumulation buffer is not changed after having been copied. Accumulation and double-buffering simply continue in the same way, to try to improve $OP(d,C)$ even further (see Sect. 4.5), in order to potentially improve $OP(d_{cull}, C_{cull})$ again. See algorithm 1.

## 4.4 Per-Pixel Particle Density Functions

Several of our probability computations require an estimate for the number of particles that are in front of or behind a certain depth. To be able to estimate these numbers, we maintain a coarse 3D density volume $\mathscr{D}$ on a regular grid that stores a 3D scalar density function of particles, i.e., $(x,y,z) \mapsto \mathscr{D}(x,y,z)$ gives a *number of particles per unit volume*. For each pixel, we then extract a 1D *particle density function* $D(t)$ along a ray $t$ through that pixel via volume ray casting, gathering particle densities integrated over the 2D footprint $A$ of the pixel, i.e.,

$$D(t) := \int_A \mathscr{D}\big(x(t), y(t), z(t)\big)\, dA, \qquad A \perp \text{to the ray } t. \qquad (9)$$

Given $D(t)$, for any depth interval $[d_a, d_b]$ along the ray $t$ we then get the corresponding particle count between depth $d_a$ and depth $d_b$ as

$$D\big(d_a, d_b\big) := \int_{d_a}^{d_b} D(t)\, dt. \qquad (10)$$

**Per-pixel particle density histograms.** While conceptually $D(t)$ is a continuous function, for each pixel our ray caster traverses the particle

**Algorithm 1:** Double-Buffered Depth Confidence Updates

```
1  function updateAccumulationBuffer ($d_s, \mathfrak{a}_s$);
2  $P(d_s)$ = acceptanceProbability($d_s$);  $u$ = rand(0,1);
3  if (($d_s < d$) and ($u \le P(d_s)$)) then
4      $(d,C) = (d_s, \mathfrak{a}_s)$;  $k = 1$;
5  else
6      $C$ = accumulateConfidence($C, \mathfrak{a}_s, k$);  $k$++;
7  end
8  function UpdateDepthAndConfidenceForSample ($d_s, \mathfrak{a}_s$);
9  if ($d_s \le d_{cull}$) then
10     $C_{cull}$ = accumulateConfidence($C_{cull}, \mathfrak{a}_s, k_{cull}$);  $k_{cull}$++;
11 end
12 if ($d_s \le d$) then
13     updateAccumulationBuffer($d_s, \mathfrak{a}_s$);
14 end
15 if $OP(d_{cull}, C_{cull}) < OP(d,C)$ then
16     $(d_{cull}, C_{cull}) = (d,C)$;  $k_{cull} = k$;
17 end
```

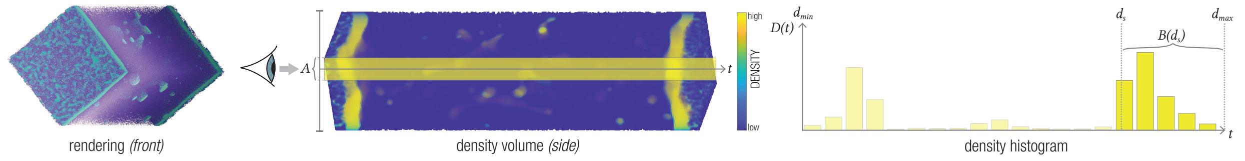

Fig. 9. **Per-pixel particle density functions.** We maintain a 3D *number of particles/unit volume* density function in a low-resolution volume $\mathscr{D}(x,y,z)$, with dynamic updates for fully occluded meshlets. From this, we compute 1D per-pixel particle density histograms $D(t)$ via volume ray casting.

density volume $\mathscr{D}$ and computes a discretized $D(t)$ in the form of a particle density *histogram*. That is, we store particle counts $D\big(d_i, d_{i+1}\big)$ in discrete bins $[d_i, d_{i+1}]$, from depth $d_{min}$ to $d_{max}$. See Fig. 9 (right).

**Dynamic volume updates.** For correct estimates, the volume $\mathscr{D}$ must correspond to particles still being considered for sampling. Thus, we dynamically update $\mathscr{D}$ when a meshlet reaches the occlusion class $\mathbb{O}$. This is efficient, because $\mathscr{D}$ is low-resolution, and we pre-compute a (static) list of voxels affected by any meshlet becoming occluded.

## 4.5 Estimating and Improving Particle Occlusion

From its per-pixel particle density function $D(t)$, we can estimate the number of particles behind a given pixel with representative depth $d$ as

$$B(d) := D\big(d, d_{max}\big). \tag{11}$$

To get the number of *occluded particles*, we need to take into account the sub-pixel coverage of the pixel, which is given by the confidence $C$. We therefore get an estimate for the number of occluded particles as

$$OP(d,C) := B(d)\,C. \tag{12}$$

We now want to compare two depths $d_s$ and $d$ in terms of numbers of occluded particles, with $d_s < d$ to consider depth improvements. For this, we compute the unknown confidence $C_s$ to be associated with $d_s$, such that $OP(d_s, C_s) > OP(d,C)$, i.e., that more particles are occluded. The minimum threshold for the unknown confidence $C_s$ is therefore

$$C_s > \frac{B(d)}{B(d_s)}\,C. \tag{13}$$

Here, we always have $C_s < C$, because we only consider $d_s < d$, and thus $B(d_s) > B(d)$. Now, we want the minimum number $n$ of *unique* particles, such that the confidence $C_s$ above is reached as $C_s = C_s(n)$. This $n$ can be determined by inverting the pixel coverage relationship

$$C_s(n) = 1 - (1-a)^n. \tag{14}$$

The number $n$ depends on $d_s$, and we therefore now write $n(d_s)$. Combining the above two equations, its minimum threshold is given by

$$n(d_s) > \log_{(1-a)}\left(1 - \frac{B(d)}{B(d_s)}\,C\right). \tag{15}$$

The base $(1-a)$ of the logarithm here is less than one. Thus, $n(d_s) > 0$. Now, we additionally have to take into account that we are actually sampling *with replacement*. We can solve this problem by computing the required minimum $k$ such that the *expected value* of the corresponding number of *unique* samples is at least $n(d_s)$. Using Eq. 5, we get

$$k(d_s) = \log_M\Big(1 - (1-M)\,\mathrm{E}\big(n(k(d_s))\big)\Big). \tag{16}$$

In this context, we already know that we want $\mathrm{E}\big(n(k(d_s))\big) \geq n(d_s)$, with $n(d_s)$ as given by Eq. 15. We can therefore obtain the minimum number of necessary samples $k(d_s)$, which, in expectation, will reach sufficient coverage $C_s(n)$ as derived in Eq. 13, Eq. 14, and Eq. 15, as

$$k(d_s) > \log_M\Big(1 - (1-M)\,n(d_s)\Big). \tag{17}$$

The expected value computation, which is now only implicit in Eq. 17, corresponds to a maximum number of particles $N$, from which we are sampling ($N$ in Eq. 5). Here, this must be $N := D\big(d_{min}, d_s\big)$, because our coverage only counted samples closer than $d_s$. Thus, $M$ must be

$$M = 1 - \frac{1}{D\big(d_{min}, d_s\big)}. \tag{18}$$

As above, the base of the logarithm is less than one, because $0 < M < 1$. In our framework, we use the minimum threshold $k(d_s)$ computed as

$$k(d_s) = \left\lceil \log_M\left(1 - \frac{n_{max}}{D\big(d_{min}, d_s\big)}\right)\right\rceil, \tag{19}$$

where we have defined the clamped value $n_{max}$ as

$$n_{max} := \min\Big(n(d_s), \eta \cdot D\big(d_{min}, d_s\big)\Big), \quad \text{with} \quad 0 < \eta < 1. \tag{20}$$

We compute $n(d_s)$ as the right-hand side of Eq. 15. The reason for clamping is to prevent $k(d_s)$ to increase without bound, due to the logarithm. In our implementation, we use a fixed percentage of $\eta = 0.98$. Moreover, in practice we in fact check explicitly whether $n(d_s)$ would be clamped by Eq. 20. If that is the case, we treat this sample with a symbolic $k(d_s) = \infty$, from which we know without further computation that the sample acceptance probability below (Eq. 21) will be zero.

### 4.5.1 Probability of fast enough culling improvement

We now address the following crucial question: Whenever a new particle is sampled by our pipeline, should its depth $d_s$ be used as the new representative depth $d$ for the corresponding pixel, i.e., should we assign $d = d_s$ and continue? There are two specific considerations:

- If $d_s \geq d$, the sample will not be considered further, because $d$ cannot be improved, i.e., decreased, from this sample's depth $d_s$.
- If $d_s < d$, this sample would improve the representative depth $d$. However, if we assign $d = d_s$ as the new representative depth, the current depth confidence $C$ associated with $d$ becomes invalid, because then the sub-pixel region known to have depth $\leq d_s$ is only the area $\mathfrak{a}_s$ of this particle. That is, if we update $d$ we essentially have to restart accumulating confidence from scratch.

Because we are sampling particles progressively, the crucial question of the latter point is whether the confidence $C_s$ of depth $d_s$ would accumulate *fast enough*. We define fast enough in terms of a number of samples $N_{budget}$. That is, if we are willing to wait for $N_{budget}$ future samples, will we then have reached a confidence $C_s$ such that $OP(d_s, C_s) > OP(d,C)$? In order to answer this, we compute the probability of this happening.

Eq. 19 tells us how many samples $k(d_s)$ with a depth $\leq d_s$ we need, in expectation, to reach $OP(d_s, C_s) > OP(d,C)$. However, our pipeline samples particles with uniform probability over the entire depth range $[d_{min}, d_{max}]$. We thus need the probability of seeing a large enough subset of particles with depth $\leq d_s$, within a budget of $N_{budget}$ samples. A sample's depth being $\leq d_s$ or not constitutes a *Bernoulli trial*. Thus, we can compute the probability of getting *at least* $k := k(d_s)$ samples with depth $\leq d_s$, in $N = N_{budget}$ samples, from the *binomial distribution*

$$P\big(d_s\big) := P\big(k \geq k(d_s)\big) = \sum_{k=k(d_s)}^{N} \binom{N}{k} p^k (1-p)^{N-k}. \tag{21}$$

The probability $p$ is given by the particle density function $D(t)$ as

$$p = p(d_s) := \frac{D\big(d_{min}, d_s\big)}{D\big(d_{min}, d_{max}\big)}. \tag{22}$$

Note that if the minimum threshold $k(d_s)$ exceeds the maximum allowed $N_{budget}$, the probability $P\big(d_s\big)$ will be zero, as should be expected.

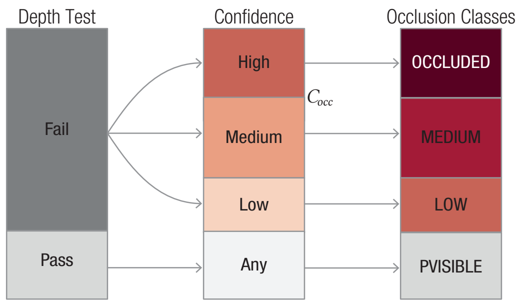

Fig. 10. **Occlusion classes** of meshlets are determined during occlusion testing from computed confidence values and from passing or failing the depth test. Only class $\mathbb{O}$ (*occluded*) will be excluded from rendering.

#### 4.5.2 Sample acceptance probability and strategy

Bringing everything together, we use the probability $P(d_s)$ determined by Eq. 21 as the *sample acceptance probability* in algorithm 1: Every time a new sample with depth $d_s$ comes in, we compute the minimum sample number $k(d_s)$ using Eq. 19, which requires computing Eq. 15, Eq. 18, and Eq. 20 first. We then compute Eq. 21 for the new sample. In practice, however, evaluating Eq. 21 directly is inefficient. We therefore use a much faster standard approximation, which is very accurate. The details are described in App. A. Finally, in order to determine whether the new sample with depth $d_s$ will be *accepted* or not, we compute a uniform random number $u \in [0,1]$. If the random number $u \leq P(d_s)$, we accept the sample with depth $d_s$. Otherwise, we do not.

### 4.6 Depth Confidence Pyramid

We maintain the depth confidence maps in the *cull buffer* in a multi-resolution pyramid, like a mipmap, to facilitate hierarchical occlusion tests similar to hierarchical occlusion maps [41], see Fig. 4 (top right). The strategies described above are used to update the highest-resolution pyramid level $LOD_0$. All other, coarser-resolution levels $LOD_i$, with $i > 0$, are updated with the iterative update rule

$$(d,C)_{LOD_i} = \left( \max_{p \in \mathscr{N}} d_p, \frac{1}{n} \sum_{p \in \mathscr{N}} C_p \right)_{LOD_{i-1}} . \tag{23}$$

This rule combines all pixels $p$ in a pixel neighborhood $\mathscr{N}$, comprising $n$ pixels, in resolution level $LOD_{i-1}$, into a single pixel in resolution level $LOD_i$. As in standard mipmaps, we use neighborhoods of $2 \times 2$ pixels ($n = 4$). The maximum operator for depth values guarantees that the combined depth is representative for the whole coarser-resolution pixel, and averaging the confidence gives the correct total confidence, corresponding to the actual combined normalized sub-pixel coverage.

## 5 Occlusion Culling

In every iteration, the current *cull buffer* is used for confidence-based occlusion culling. We perform hierarchical culling, first testing nodes, and then testing the meshlets contained inside each node if necessary.

**Confidence-based culling.** We test the axis-aligned bounding box of each meshlet/node for occlusion in two ways: (1) The bounding box is rasterized, and the standard GPU depth test determines whether the depth test *failed* (occluded with respect to the depth buffer) or *passed* (not occluded). The depth buffer used for this test is the *representative depth $d_{cull}$* stored in the cull buffer. (2) Considering sub-pixel occlusion, it is important to take into account that particles in a node/meshlet that *failed* the depth test can still be visible. This depends on the associated confidence values $C_{cull}$. We aggregate all confidence values $C_{cull}$ in the set of pixels in the screen space projection of the bounding box that is tested, resulting in an overall confidence for the tested geometry. Only if this confidence value is high enough (and the depth test *failed*), is a meshlet/node considered to be occluded (in occlusion class $\mathbb{O}$, below).

### 5.1 Occlusion Classes

At any time, meshlets belong to one of four *occlusion classes* (Fig. 10), according to the criteria below. After each iteration, the occlusion class must be updated according to new cull buffer contents. The classes are

- $\mathbb{O}$ (Occluded): *Meshlets considered definitely occluded with high confidence*. They have failed the depth test with a high confidence of occlusion. We define this as confidence greater than a *confidence threshold $C_{occ}$*, e.g., $C_{occ} = 0.95$ (see Fig. 11). Meshlets in this class will not be sampled again (unless we restart for a new view).
- $\mathbb{M}$ (Medium): *Meshlets with medium confidence of occlusion.* They have failed the depth test, but with medium confidence, so they cannot be culled yet. However, the probability of occlusion is too high to consider them as good potential occluders themselves.
- $\mathbb{L}$ (Low): *Meshlets with low confidence of occlusion.* They have failed the depth test. They could be occluded, but confidence in occlusion is low. They will be sampled as potential occluders, but only when there are not enough meshlets in $\mathbb{P}$ to increase occlusion.
- $\mathbb{P}$ (Potentially visible): *Potentially visible meshlets.* They have *passed* the depth test, so they cannot be culled, *independent of confidence*. They are the main candidates for good potential occluders.

Initially, and every time the view has changed, all meshlets are put into class $\mathbb{P}$, because nothing is known yet regarding their visibility. Then, in each iteration, a set of potential occluders is selected (see below) and sampled into the cull and accumulation buffers. Afterward, all meshlets in any class except $\mathbb{O}$ are culled against the cull buffer to update their occlusion class. The overall goal of iterating is to move as many meshlets as possible into class $\mathbb{O}$. Meshlets are moved between classes according to their current overall confidence values and may only move in the “upward” direction, i.e., increase in confidence from iteration to iteration, because confidence values can never decrease.

### 5.2 Occluder Selection

To select potential occluders in each iteration, we pull meshlets from $\mathbb{P}$ until we reach a pre-specified budget per iteration. If the budget is not yet reached after all meshlets in $\mathbb{P}$ have been consumed, we continue by sampling meshlets selected from $\mathbb{L}$. We do not prevent the same meshlets from being selected as potential occluders in multiple iterations. This is crucial for our approach: It allows confidence values to increase due to *super-sampling occlusion*, iteratively attaining more known sub-pixel coverage and thereby occlusion.

**Avoiding starvation.** In the common case when there are more meshlets in $\mathbb{P}$ and $\mathbb{L}$ together than the specified budget for potential occluders per iteration, it is important to not select the same subset of potential occluders in every subsequent iteration, to avoid “starving” other good potential occluder candidates. We therefore *randomize* the selection of potential occluders by *shuffling* meshlets in $\mathbb{P}$ and $\mathbb{L}$. This can be implemented very efficiently on GPUs using *task shaders*, Vulkan *subgroups* [21], and a random permutation buffer.

### 5.3 Occlusion Convergence

After sampling nodes/meshlets in order to determine the confidence of occlusion, we need to decide whether or not occlusion computations have converged. We discuss several criteria for convergence below.

**Rendering budget.** We can check whether a small enough number of meshlets remains after all other meshlets have reached class $\mathbb{O}$. This criterion is often too crude, unless the number of remaining meshlets is very small, because there could still be room for improving occlusion.

**Overall confidence.** A potential test could be to check the root node of the confidence map pyramid to determine that the total confidence in all depth values is high enough. However, this just means that the known depth values—which could be unnecessarily large—are accurate enough. Because high coverage does not directly correspond to small depth values, it does not necessarily correspond to high occlusion. For this reason, we do not use this criterion for determining convergence.

**Active changes of occlusion class membership.** We use the following criterion to determine convergence: If over multiple iterations no meshlet changes its occlusion class, we stop iterating and assume that we have converged to a sufficient amount of occlusion. At the same time, this implies that we have already reached high confidence for the occlusion decisions that we have performed, i.e., we are confident (with threshold $C_{occ}$) about occlusion of all meshlets in occlusion class $\mathbb{O}$.

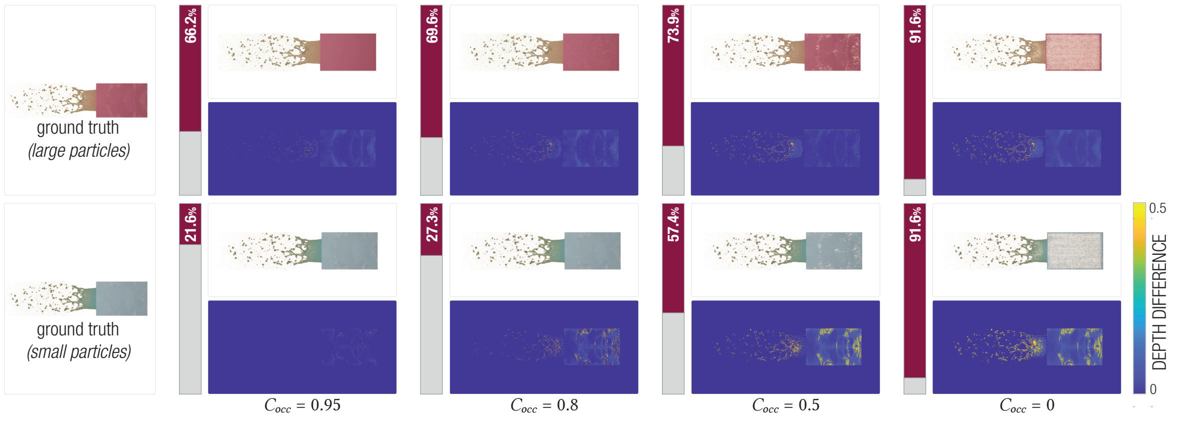

Fig. 11. **Varying confidence threshold $C_{occ}$ for class $\mathbb{O}$.** Ground truth renderings (left) use 1,024 samples per pixel, without any culling. We compare two different particle sizes, large at the top, and small at the bottom. From left to the right, we decrease the confidence threshold $C_{occ}$ that determines when meshlets are moved into occlusion class $\mathbb{O}$. (Top rows) Renderings. (Bottom rows) Depth difference images highlighting pixels for which the representative depth is closer than the ground truth with confidence $\geq C_{occ}$. Too low $C_{occ}$ increases culling, but result in artifacts.

## 6 Implementation

We exploit recent advancements in GPU architecture introduced with the NVIDIA Turing RTX GPU. We employ a new graphics pipeline that comprises *task* and *mesh shaders*, instead of the traditional pipeline.

**The task/mesh shader pipeline** operates on small groups of triangles called *meshlets*, which target vertex re-use to reduce the data to fetch in parallel rendering. However, particle data sets are not typical meshes with high vertex re-use. Rather, we are dealing with a soup of disjoint billboards with two triangles per particle. However, our occlusion culling architecture exploits the parallel processing of small groups of particles packed into meshlets for fast occlusion testing.

**Meshlet generation** uses the Point Cloud Library (PCL) [34] in a pre-processing step to subdivide the input data into small neighborhoods of particles, which then become meshlets. We use a neighborhood size of 16 particles, because this requires 64 vertices to be drawn: Two triangles and four vertices per billboard, and 64 is currently the maximum number of emitted vertices recommended per meshlet [26].

**Nodes.** For the purpose of hierarchical culling of particle data sets, we group neighborhoods of meshlets together into larger *nodes*. Each node fits a fixed maximum number of particles (we are using $10,000$).

**Mesh shaders.** We use mesh shaders to generate the actual particle billboard geometry for potentially visible meshlets. Each invocation of a mesh shader operates on a meshlet, and generates the corresponding geometric primitives. These primitives are then rasterized to produce the fragments that are then shaded by a traditional *fragment shader*.

**Task shaders.** These are the most important feature of the Turing geometry pipeline for our framework. *Task shaders* are compute shaders that dynamically enable/disable emittance of *mesh shader workgroups*. We use them to efficiently determine whether whole meshlets are occluded, and issue workgroups only for potentially visible meshlets.

**Vulkan.** We have implemented our framework in C++ using the Vulkan API, which offers more low-level control over GPU operation than APIs such as OpenGL. We make use of Vulkan's subgroups [21] to determine which meshlets to sample, and perform culling for them.

## 7 Results

We evaluate rendering quality and performance, and compare to previous work. Table 2 lists the data sets we have used for the evaluation. A more extensive evaluation is given in the supplemental App. B.

### 7.1 Rendering Quality

In Fig. 11, we analyze the effects of different confidence thresholds $C_{occ}$ for occlusion class $\mathbb{O}$. Too low values for $C_{occ}$ lead to culling of meshlets even when not all of their particles are already sufficiently covered, leading to artifacts. However, Fig. 11 demonstrates that our probabilistic approach is quite robust. In practice, we use $C_{occ} = 0.95$.

Fig. 12 illustrates the importance of super-sampling, by comparing the quality of our method and the method of Grottel et al. [16].

We also employ screen-space ambient occlusion as a visual cue, since the understanding of spatial structure is essential for molecular dynamics simulations [25]. Since we already maintain a low-resolution particle density volume (see Sect. 4.4), we could easily also extend this to the object-space technique presented by Grottel et al. [15].

Table 2. **Data sets** used for evaluation. (See the appendixes for more.)

| data set | # particles | # meshlets |
|---|---|---|
| covid-19 | 40,048,645 | 2,733,504 |
| large laser ablation | 199,940,704 | 12,513,900 |
| 16 $\times$ copper/silver mixture | 232,000,000 | 14,420,300 |

### 7.2 Performance

We evaluate culling performance in terms of percentage of particles culled, time taken to cull, and the rendering speed (frames per second/fps) during ("dc") and after ("ac") culling. Performance was measured on a dual Intel Xeon X5550 2.67 GHz, Geforce RTX Titan (24 GB), at $1920 \times 1080$ resolution. Table 3 gives performance results for culling and rendering, for three different views. (See full tables and more data sets in the supplemental material.) Our method results in a significant speed up after culling is finished ("ac"). This is more apparent for large data sets, such as the 16 $\times$ copper/silver mixture data set, where the speed up is up to $10\times$. We note that the fps during culling ("dc") is usually faster than the fps without culling ("w/o c"). This is because our method determines meshlets in class $\mathbb{O}$ progressively, and therefore will place more and more meshlets there as confidence builds up to greater or equal the confidence threshold $C_{occ}$. This in turn will result in a progressive decrease in the number of meshlets to render until the occlusion computations converge, leading to higher fps.

Table 4 shows percentages of culled meshlets, and number of samples required until convergence (full results are in the supplemental material). To evaluate our probabilistic acceptance strategy (Sect. 4.5), we also report numbers where instead we simply accept incoming sam-

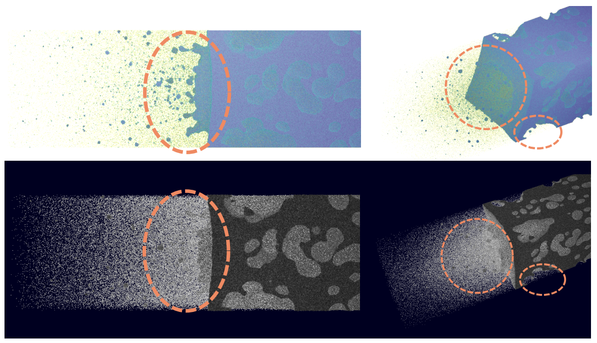

Fig. 12. Top: Our method successfully captures interesting features and removes noise due to super-sampling. Bottom: The same views using a single sample per pixel (with Grottel et al. [16]) exhibit undersampling.

Table 3. **Rendering performance.** Average rendering speed for different data set/view (v) combinations: averaged over the whole culling process (dc); after occlusion convergence (ac); without occlusion culling (w/o c). Confidence threshold $C_{occ} = 0.95$; rendering budget $b = 2$ meshlets/node.

| data set | v | **dc** [fps] | **ac** [fps] | w/o c [fps] | MegaMol [fps] | Grottel [fps] |
|---|---|---|---|---|---|---|
| covid-19 | 0 | 22 | 56 | 19 | 32 | 18 |
| | 1 | 25 | 63 | 28 | 38 | 17 |
| | 2 | 24 | 59 | 27 | 50 | 20 |
| large laser ablation | 0 | 13 | 35 | 16 | 14 | 59 |
| | 1 | 18 | 42 | 15 | 9 | 28 |
| | 2 | 14 | 36 | 16 | 13 | 40 |
| 16 × copper/ silver mixture | 0 | 8 | 34 | 3 | 7 | 35 |
| | 1 | 12 | 41 | 6 | 13 | 12 |
| | 2 | 9 | 37 | 5 | 8 | 16 |

Table 4. **Culling efficiency.** Percentage of culled meshlets (c) and number of samples (smp) required for culling convergence. (Column "random" uses 50:50 sample acceptance for comparison only.) Confidence threshold $C_{occ} = 0.95$; rendering budget $b = 2$ meshlets per node.

| data set | view | **probabilistic** | | random | | Grottel |
|---|---|---|---|---|---|---|
| | | c [%] | smp | c [%] | smp | c [%] |
| covid-19 | 0 | 75 | 112 | 42 | 200 | 56 |
| | 1 | 58 | 120 | 29 | 128 | 32 |
| | 2 | 60 | 120 | 25 | 152 | 24 |
| large laser ablation | 0 | 68 | 208 | 8 | 64 | 89 |
| | 1 | 79 | 136 | 36 | 96 | 79 |
| | 2 | 70 | 176 | 32 | 272 | 83 |
| 16 × copper/ silver mixture | 0 | 89 | 88 | 35 | 104 | 93 |
| | 1 | 92 | 72 | 59 | 104 | 82 |
| | 2 | 90 | 88 | 44 | 112 | 87 |

ples randomly with a probability of 50% (a coin flip). Our probabilistic approach achieves much higher culling percentages than random acceptance (about 2-8×). We note that the bigger the data set, the smaller the particles' projection is on screen, and therefore more samples are required in order to achieve confidence in the depth used for culling.

## 7.3 Comparison to Previous Work

Our approach is conceptually *orthogonal* to the actual sampling method for the final rendering: Both rasterization and ray tracing are valid approaches. In our implementation, we perform ray casting of the particle density volume, but sample particles via rasterization [17]. Given that ray tracers exploit spatial data structures to accelerate ray-geometry intersections, integration of our culling approach with these data structures and the corresponding (dynamic) updates go far beyond the scope of this work and will require more in-depth investigation. We therefore compare our results to the free and open-source particle renderers available in MegaMol, given that they perform better than some other available implementations [14]. We use both the baseline brute-force SSBO-based renderer, and the multilevel culling variant by Grottel et al. [16] (with all culling and caching enabled). Both of these renderers, however, do not support multisampling, so their output will always suffer from aliasing. The performance of these approaches is still useful for interpreting the performance of our approach: The brute-force renderer represents a low-overhead, straightforward implementation without any acceleration structure. The impact of large data sets on performance is mitigated by the low number of fragments generated for each sphere (just one in the limit case), but this renderer cannot scale to arbitrary data sizes, it is both limited by available GPU RAM and shader performance. The occlusion culling approach uses both occlusion queries and culling against a hierarchical z buffer in the vertex stage, and thus serves as a reference for a basic culling technique. In principle, it scales to arbitrary data sizes because only occluders are cached on the GPU. It progressively updates the GPU cache, but renders and tests the whole data set, thus streaming the uncached data to the GPU. Interactivity is limited by upload bandwidth and, to a lesser degree, rendering performance. Our approach is completely progressive to guarantee interactivity. Since on current GPUs 8 GB RAM or more are common, keeping all data in GPU memory is not a problem.

**Rendering quality** is significantly improved over brute-force and smoothing alike (see also App. D). Our method captures interesting features better (see dashed ellipses in Fig. 12). Sub-pixel details allow the discovery of features without bias from smoothing or LOD.

**Performance.** The MegaMol brute-force renderer is consistently faster than our approach with culling disabled ("w/o c"): As long as the view does not change, our approach always uses the averaging pass over new samples and the previous results to reduce aliasing, while the MegaMol renderers do not. Also, the brute-force renderer in MegaMol is probably better optimized than our sampling. Despite this, the converged culling achieves much better performance than the culling-assisted MegaMol rendering, especially for large data.

**Culling.** Our culling percentage is similar to the approach by Grottel et al., with a few notable exceptions: Since our meshlets have much finer granularity than the bricks used by Grottel et al., we can remove particle groups from medium-density gas phases present in these data sets. Their approach can only accomplish this for each sphere separately in the hierarchical z test and thus has higher geometry processing load.

**Ray tracing.** We compare our method against GPU ray tracing using P-k-d trees [12] in App. D. While P-k-d trees are faster per sample, increasing the number of samples per pixel to allow for sub-pixel detail degrades performance linearly, as expected. Interactively sampling tens to hundreds of times per pixel is only feasible with P-k-d trees if the ray tracer accumulates samples as long as no user interaction takes place.

## 7.4 Storage Requirements

We consider the memory usage of our approach in two categories:

**Data-dependent storage.** We maintain density volumes $\mathscr{D}$ of very small resolutions, from 700k to 1M voxels (in total, not per dimension), depending on the size of the data set. We store a (node/meshlet)/voxel correspondence vector that for each node/meshlet stores the indices of the voxels in $\mathscr{D}$ to which they correspond. For each node/meshlet we store: (1) An axis-aligned bounding box in four float (32 bit) values. (2) An unsigned int (8 bit) for its occlusion class. (3) An index to the voxel in the (node/meshlet)/voxel correspondence vector (32 bit).

**Resolution-dependent storage.** Per pixel, we store the representative depth $d$, the depth confidence $C(k)$, and the sample count $k$. We store $d$ and $C(k)$ in 32 bit floats, and $k$ in an unsigned 32 bit int. Therefore, for the highest resolution pyramid level $LOD_0$, we store $3 \times 32 = 96$ bits/pixel. For the lower-resolution pyramid levels we store only $2 \times 32 = 64$ bits/pixel. Finally, for each pixel in $LOD_0$, we store the discretized function $D(t)$ in a maximum number of 64 bins.

## 8 Conclusion and Future Work

Our culling architecture is probabilistic in many respects, allowing us to target high-quality rendering with extensive super-sampling of finely detailed particle data with high performance. All our probabilistic estimates are based on expected value derivations, and therefore our pipeline operates with the idea of *expected* occlusion. Due to the large number of samples and particles in the data we are targeting, this turns out to be a very good strategy. We probabilistically estimate both sub-pixel coverage and probabilities of hitting samples in front or behind a certain depth, using a particle density estimate from a coarse density volume. Building on the latter capability, we can efficiently estimate an acceptance probability that determines whether an incoming depth sample is likely to improve occlusion and should therefore be accepted or not. Our results have shown that this strategy works very well in practice and significantly improves the percentage of culled particles.

For future work, we would like to investigate how progressive confidence can be exploited for even larger data sets or time-dependent data, where not all particles fit into GPU memory. This approach could also be combined with partial and differential updates to a GPU-resident cache that reflects the state of the current confidence class distribution.

## Acknowledgments

This work was partially supported by Intel® Corporation via the Intel® Graphics and Visualization Institutes of XeLLENCE program (CG #35512501). This work was supported by King Abdullah University of Science and Technology (KAUST).

## References

[1] J. Bittner, M. Wimmer, H. Piringer, and W. Purgathofer. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum*, 23(3):615–624, 2004. doi: 10.1111/j.1467-8659.2004.00793.x

[2] J. Bittner and P. Wonka. Visibility in computer graphics. *Environment and Planning B: Planning and Design*, 30(5):729–755, 9 2003. doi: 10.1068/b2957

[3] L. Carpenter. The a-buffer, an antialiased hidden surface method. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pp. 103–108. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1984.

[4] A. S. Christensen, B. Licea-Kane, C. Everitt, J. Bolz, and M. Ribble. ARB_occlusion_query2 OpenGL extension. `https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_occlusion_query2.txt`. Accessed: 2018-03-15.

[5] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, July 2003. doi: 10.1109/TVCG.2003.1207447

[6] J. L. Devore. *Probability and Statistics for Engineering and the Sciences*. Brooks/Cole, 8th ed., January 2011. ISBN-13: 978-0-538-73352-6.

[7] 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. doi: 10.1111/cgf.12197

[8] W. Feller. On the normal approximation to the binomial distribution. *Ann. Math. Statist.*, 16(4):319–329, 12 1945. doi: 10.1214/aoms/1177731058

[9] E. Gobbetti and F. Marton. Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics*, 28(6):815–826, 2004. doi: 10.1016/j.cag.2004.08.010

[10] E. Gobbetti and F. Marton. Far Voxels – a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. *ACM Transactions on Graphics*, 24(3):878–885, August 2005. Proc. SIGGRAPH 2005.

[11] P. Goswami, Y. Zhang, R. Pajarola, and E. Gobbetti. High Quality Interactive Rendering of Massive Point Models Using Multi-way kd-Trees. In *2010 18th Pacific Conference on Computer Graphics and Applications*, pp. 93–100. IEEE, sep 2010. doi: 10.1109/PacificGraphics.2010.20

[12] P. Gralka, I. Wald, S. Geringer, G. Reina, and T. Ertl. Spatial partitioning strategies for memory-efficient ray tracing of particles. In *10th IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 42–52, 2020. doi: 10.1109/LDAV51489.2020.00012

[13] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pp. 231–238. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993.

[14] 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, Feb 2015. doi: 10.1109/TVCG.2014.2350479

[15] S. Grottel, M. Krone, K. Scharnowski, and T. Ertl. Object-Space Ambient Occlusion for Molecular Dynamics. In *IEEE Pacific Visualization Symposium 2012*, pp. 209–216. IEEE, Feb. 2012. ISSN: 2165-8765. doi: 10.1109/PacificVis.2012.6183593

[16] S. Grottel, G. Reina, C. Dachsbacher, and T. Ertl. Coherent culling and shading for large molecular dynamics visualization. In *Proc. of Eurovis 2010*, pp. 953–962, 2010. doi: 10.1111/j.1467-8659.2009.01698.x

[17] S. Gumhold. Splatting Illuminated Ellipsoids with Depth Correction. In *Vision, Modeling, and Visualization*, pp. 245–252, 2003.

[18] M. Guthe, Á. Balázs, and R. Klein. Near optimal hierarchical culling: Performance driven use of hardware occlusion queries. In *Rendering Techniques*, pp. 207–214, 2006.

[19] P. Haeberli and K. Akeley. The accumulation buffer: Hardware support for high-quality rendering. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, pp. 309–319. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1990.

[20] J. Hasselgren, M. Andersson, and T. Akenine-Möller. Masked software occlusion culling. In *Proceedings of High Performance Graphics*, HPG '16, pp. 23–31. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2016.

[21] N. Henning. Vulkan subgroup tutorial.

[22] M. Ibrahim, P. Wickenhäuser, P. Rautek, G. Reina, and M. Hadwiger. Screen-Space Normal Distribution Function Caching for Consistent Multi-Resolution Rendering of Large Particle Data. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):944–953, jan 2018. doi: 10.1109/TVCG.2017.2743979

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

[24] A. Knoll, I. Wald, P. Navratil, M. Papka, and K. Gaither. Ray tracing and volume rendering large molecular data on multi-core and many-core architectures. In *UltraVis '13: Proceedings of the 8th International Workshop on Ultrascale Visualization*, pp. 1–8, 2013.

[25] B. Kozlíková, M. Krone, M. Falk, N. Lindow, M. Baaden, D. Baum, I. Viola, J. Parulek, and H.-C. Hege. Visualization of Biomolecular Structures: State of the Art Revisited. *Computer Graphics Forum*, 36(8):178–204, Dec. 2017. doi: 10.1111/cgf.13072

[26] C. Kubisch. Introduction to turing mesh shaders.

[27] M. Le Muzic, L. Autin, J. Parulek, and I. Viola. cellVIEW: a Tool for Illustrative and Multi-Scale Rendering of Large Biomolecular Datasets. In K. Bühler, L. Linsen, and N. W. John, eds., *Eurographics Workshop on Visual Computing for Biomedicine*, vol. 2015, pp. 61–70. The Eurographics Association, 2015. doi: 10.2312/vcbm.20151209

[28] 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. doi: 10.1111/j.1467-8659.2012.03128.x

[29] O. Mattausch, J. Bittner, A. Jaspe, E. Gobbetti, M. Wimmer, and R. Pajarola. Chc+ rt: Coherent hierarchical culling for ray tracing. In *Computer Graphics Forum*, vol. 34, pp. 537–548. Wiley Online Library, 2015.

[30] O. Mattausch, J. Bittner, and M. Wimmer. Chc++: Coherent hierarchical culling revisited. In *Computer Graphics Forum*, vol. 27, pp. 221–230. Wiley Online Library, 2008.

[31] J. Pantaleoni, L. Fascione, M. Hill, and T. Aila. Pantaray: Fast ray-traced occlusion caching of massive scenes. *ACM Trans. Graph.*, 29(4):37:1–37:10, July 2010. doi: 10.1145/1778765.1778774

[32] J. Parulek, D. Jönsson, T. Ropinski, S. Bruckner, A. Ynnerman, and I. Viola. Continuous Levels-of-Detail and Visual Abstraction for Seamless Molecular Visualization. *Computer Graphics Forum*, 33(6):276–287, 2014. doi: 10.1111/cgf.12349

[33] K. Reda, A. Knoll, K. Nomura, M. Papka, A. Johnson, and J. Leigh. Visualizing large-scale atomistic simulations in ultra-resolution immersive environments. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 59–65, 2013.

[34] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–4. IEEE.

[35] N. Subtil. Introduction to Real-Time Ray Tracing with Vulkan. `https://devblogs.nvidia.com/vulkan-raytracing/`. Accessed: 2018-03-31.

[36] I. Wald, G. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Gunther, and P. Navratil. Ospray - a cpu ray tracing framework for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):931–940, Jan. 2017. doi: 10.1109/TVCG.2016.2599041

[37] 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. *IEEE Transactions on Visualization and Computer Graphics*, pp. 57–64, 2015.

[38] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst. Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph.*, 33(4):143:1–143:8, July 2014. doi: 10.1145/2601097.2601199

[39] H. Wang, L. Xiao, Y. Cao, Z. Ai, and P. Xu. Visibility-culling-based geometric rendering of large-scale particle data. In *Proceedings - 2016 International Conference on Virtual Reality and Visualization, ICVRV 2016*, pp. 197–203, sep 2017. doi: 10.1109/ICVRV.2016.41

[40] P. Wonka, M. Wimmer, K. Zhou, S. Maierhofer, G. Hesina, and A. Reshetov. Guided visibility sampling. *ACM Trans. Graph.*, 25(3):494–502, July 2006. doi: 10.1145/1141911.1141914

[41] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff, III. Visibility culling using hierarchical occlusion maps. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pp. 77–88. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997. doi: 10.1145/258734.258781