High Performance Graphics (2016)
Ulf Assarsson and Warren Hunt (Editors)

Adaptive Sampling for On-The-Fly Ray Casting of Particle-based Fluids

H. Hochstetter[1], J. Orthmann[2], and A. Kolb[1]

[1]Computer Graphics and Multimedia Systems Group, University of Siegen, Germany
[2]CST Computer Simulation Technology, Darmstadt, Germany

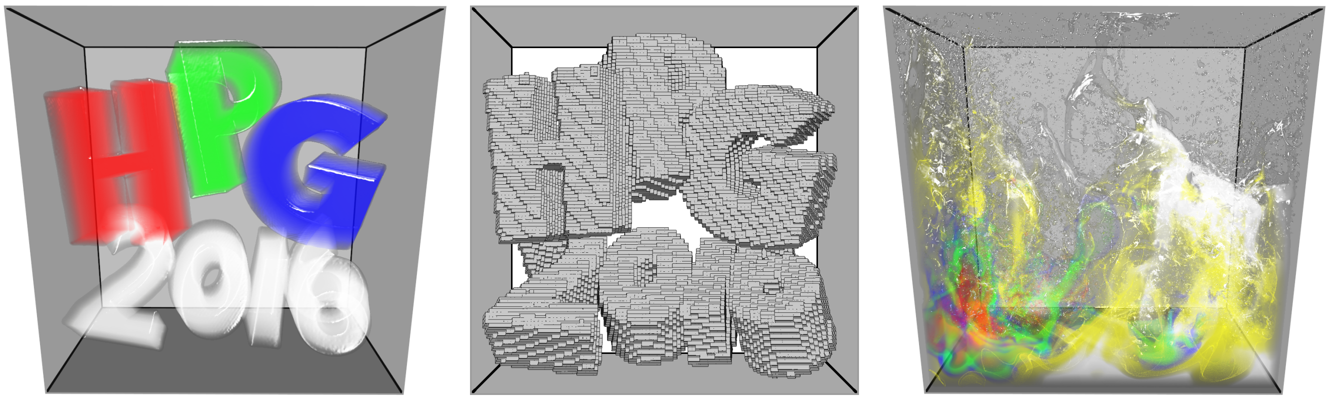


Figure 1: The fluid letters HPG 2016 (left) are dropped into a basin and cause a splashing effect (right). Our sparse, perspective particle access grid adapts to the scene (middle).

Abstract

We present a fast and accurate ray casting technique for unstructured and dynamic particle sets. Our technique focuses on efficient, high quality volume rendering of fluids for computer animation and scientific applications.
Our novel adaptive sampling scheme allows to locally adjust sampling rates both along rays and in lateral direction and is driven by a user-controlled screen space error tolerance. In order to determine appropriate local sampling rates, we propose a sampling error analysis framework based on hierarchical interval arithmetic. We show that our approach leads to significant rendering speed-ups with controllable screen space errors. Efficient particle access is achieved using a sparse view-aligned grid which is constructed on-the-fly without any pre-processing.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation— I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—

1. Introduction

Lagrangian simulations like *Smoothed Particle Hydrodynamics* (SPH) offer high spatial flexibility which is advantageous for convection-driven flow effects, free surfaces, and dynamic fluid-object interactions [IOS*14]. However, ad hoc rendering of large unstructured and dynamic particle sets is still a challenging task.

Several techniques exist to reconstruct smooth surfaces for SPH-like particle sets [SSP07, vdLGS09, OCD11, AIAT12, YT13, RCSW14, ZD15]. However, sole surface rendering can only convey the fluid's geometric shape, but does not provide any information about internal fluid structures, e.g., in convection-diffusion scenarios. Thus, volume rendering techniques [EHK*06, HLSR08] have to be applied in order to provide full insight into the fluid dynamics. In the context of unstructured particle sets, mainly hybrid splatting-slicing approaches have been proposed so far [FGE10, FAW10]. Here, particle contributions [Wes90] are scattered onto axis-aligned [SP09] or view-aligned [FGE10, FAW10, NMM*06] texture slices. These slices are then composited front-to-back yielding the final image. Such texture-slicing approaches rely on the GPU's rasterization pipeline which makes it very hard to incor-

DOI: 10.2312/hpg.20161199

porate adaptive sampling mechanisms for further rendering speed-up. The more generic approach to volume rendering is ray casting [EHK*06] which, however, requires very efficient particle access mechanisms that, in case of dynamic particle sets, cannot rely on any kind of costly pre-processing.

In this paper we propose an on-the-fly volume ray casting for unstructured particle data sets. Our ray casting makes use of an adaptive sampling scheme to allow for an efficient rendering of large dynamic particle sets as shown in Fig. 1. In detail, the proposed rendering approach incorporates the following contributions:

- An on-the-fly sampling error analysis framework based on hierarchical interval arithmetic for ray bundles which is able to derive strict bounds to the screen space error resulting from locally adapting sampling rates in lateral and viewing directions.
- A greedy algorithm that optimizes the degree of adaptivity both in viewing and lateral directions to yield significant speed-ups for a user-controlled screen space error.
- We enhance the perspective, view-aligned grid known from texture-slicing [FAW10] to an efficient sparse access structure for particle data that is constructed on-the-fly.

The remainder of this paper is structured as follows: Sec. 2 discusses foundations in SPH and volume ray casting. Sec. 3 gives an overview of our ray casting pipeline. Sec. 4 and 5 explain our sampling error analysis framework applied to determine proper sampling rates. Implementation details are given in Sec. 6. Sec. 7 discusses the results before conclusions are drawn in Sec. 8.

2. Foundations and Prior Work

2.1. Smoothed Particle Hydrodynamics (SPH)

In SPH, quantity fields are reconstructed by interpolating discrete particle quantities q_j in the local neighborhood of sampling positions x [Mon05]:

$$Q(x) = \sum_j q_j V_j \hat{W}_j(x). \tag{1}$$

$V_j \approx 1$ is a particle's dynamic volume. We use zero order consistent interpolation (CSPH) to avoid volume underestimation at fluid boundaries resulting from particle neighborhood deficiency [BK02]

$$\hat{W}_j(x) = \frac{W_j(x)}{\sum_k V_k W_k(x)}, \tag{2}$$

where $W_j(x) = W(||x - x_j||, h_j)$ is a radially symmetric smoothing function with compact support radius h_j [MCG03].

2.2. Volume Rendering

Volume ray casting evaluates a physically-based model of light transport by treating quantity fields as a participating medium. For each viewing ray, emission-absorption values are integrated from the camera through the fluid volume. Considering ray samples at integer coordinates $i = 0, \ldots, N-1$ with associated quantities $Q_0, \ldots, Q_{N-1}$, discretization of the volume rendering integral defines a ray's composited irradiance and transparency as [EHK*06]

$$I = \sum_{i=0}^{N-1} I_i \prod_{j=0}^{i-1} T_j^{\Delta s}, \text{ and } \quad T = \prod_{i=0}^{N-1} T_i^{\Delta s}, \tag{3}$$

respectively, where irradiance values $I_i = \tau_I(Q_i)$ and transparency values $T_i = \tau_T(Q_i)$ are defined via material-dependent transfer functions $\tau : [0,1] \to [0,1]$. Discrete samples $Q_i = Q(\boldsymbol{x}(s_i))$ are distributed at distances $s_i \in [s_{\text{near}}, s_{\text{far}}]$ along viewing rays, where s_{near} and s_{far} are the distances to the near and far clipping planes of the view frustum. Note, transparency values given for a unit reference length are corrected in Eq. 3 to match the sampling step size Δs. In the following, we will omit opacity correction terms to improve readability but in practice they have to be applied appropriately.

Adaptive Rendering of Grid Structures Adaptive sampling is mainly applied for data on regular grids [BHMF08, GS04, KHW*09]. Danskin and Hanrahan use importance sampling in order to locally adapt sampling rates [DH92]. Although it is possible to substantially speed up rendering using importance sampling, it is a stochastic approach and hence it is difficult to calculate explicit bounds for the screen space error. Ledergerber et al. [LGM*08] introduce a Moving-Least-Squares (MLS) approach to reconstruct higher order continuous field functions, which can also be applied to irregular grids. Similar to our approach, Guthe and Strasser [GS04] estimate screen space errors due to adaptive sampling when uncompressing wavelet representations. However, their approach requires a costly wavelet transform as precomputation, which is unfeasible for on-the-fly visualization of dynamic particle data sets.

Volume Rendering for Particle Sets In contrast to sampling in regular grids, efficiently accessing unstructured particle data that contribute to a sampling position poses quite a challenge. This is commonly addressed using spacial subdivision structures such as object-aligned octrees [OKK10]. As all object-aligned access structures require cell finding logic and may introduce thread divergence, parallelism can be drastically reduced [PGSS07].

Perspective, view-aligned grids can be employed to achieve memory coherence and to remove traversal efforts [HM08]. Starting from a pre-computed multiresolution particle representation Fraedrich et al. [FAW10] resample particle sets to a perspective grid for further texture-based ray casting. Their approach adjusts the sampling step size in viewing direction to be consistent with the perspectively increasing lateral resolution (see also Sec. 6.1). In a follow up work, Reichl et al. [RTW13] pre-process the SPH particle set by resampling it onto an object-aligned octree hierarchy before ray casting.

Furthermore, particle upsampling can be applied in order to reduce the overall particle count, and thus increase rendering performance. Upsampling operators [APKG07, ZSP08, HHK08] approximate particle subsets by fewer larger particles [HE03, FSW09, FAW10]. Upsampling, however, may introduce visual artifacts [BOT01, KAG*06] and should only be applied to avoid undersampling in case particle sizes fall below the pixel size.

Surface Rendering for Particle Sets Scalar field functions are usually employed to implictly describe the surface which require an extra smoothing pass either by redistancing of surface particles [APKG07], by distance [SSP07] or density [OCD11] decay functions or by using anisotropic smoothing kernels [YT13]. For instant rendering, either screen space techniques [ZPvBG01, ALD06, MSD07, vdLGS09] or direct rendering of the isosurface [GSSP10, FAW10] is employed. Recently, binary volume representations have been used to efficiently render surfaces of particle data [RCSW14, ZD15]. Reichl et al. [RCSW14] used attributed binary voxel hierarchies which, however, have to be pre-computed. Similar to our approach Zirr and Dachsbacher [ZD15] use a perspective data structure that is constructed on-the-fly. However, they only store entry and exit depths of rays passing through parts of the fluid and do not allow for volume rendering of scalar fields. Large data sets consisting of opaque particles can be efficiently rendered using P-k-d-trees [WJP14]. However, this has not been shown to be extensible to volume ray casting.

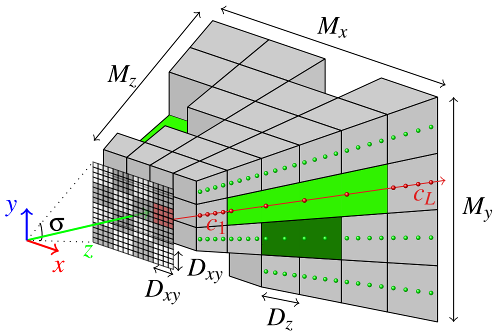


Figure 2: Sparse access structure. Cell merging ensures a constant number of $D_{xy} \times D_{xy} \times D_z$ samples per cell. The image plane is split into ray bundles of size $D_{xy} \times D_{xy}$ each storing cells C_1 and C_L in a look-up table. $M_x \times M_y \times M_z$ is the maximum number of cells present in a dense grid.

3. Proposed Adaptive Ray Casting Pipeline

Even though we focus on on-the-fly ray casting of dynamic SPH data sets, our scheme can be applied to any kind of unstructured particle sets based on local operators for recovering continuous quantity fields. However, we restrict ourselves to particle sizes which do not fall below pixel size in screen-space, thus, particle upsampling cannot be applied.

Our ad hoc on-the-fly ray casting of dynamic SPH particle sets uses the original particle set "as is" to prevent any additional error due to re-sampling or interpolating particle quantities onto intermediate data structures such as grids or coarser particles. Also, any other kind of prohibitive and costly pre-computation is omitted.

The proposed ray casting pipeline comprises five components:

Sparse View-Aligned Grid Structure: Our enhanced perspective grid subdivides the view frustum into cells that are aligned with view rays. In contrast to Fraedrich et al. [FAW10], who resample particle quantities into a dense grid, we use a sparse data structure to access the original particle data. Figure 2 shows the resulting sparse grid structure. The grid is built from scratch in every frame using only raw particle data as input. The particles are assigned to all cells that intersect their volume and only cells that contain particles are present in the final grid. During ray casting each cell is traversed by a *ray bundle* covering $D_{xy} \times D_{xy}$ pixels in screen space. As the cells are aligned with the view rays, all rays of a bundle traverse the same set of cells in viewing direction. Using an inverse perspective mapping, the perspectively distorted sampling positions in view space are described in uniform sample space (see Sec. 6.1).
Initially each cell contains $D_{xy} \times D_{xy} \times D_z$ samples, with $D_{xy} > 1$ and $D_z > 1$. Each ray samples D_z positions inside of each cell, however, the sampling error analysis allows to locally reduce the number of samples in powers of 2. Thus, the *sampling level* l_C corresponds to $D_z/2^{l_C}$ sampling positions per ray in cell C.

Sampling Error Analysis: To locally adapt the sampling rate for each cell, we introduce a formulation of the rendering equation based on hierarchical interval arithmetic. Inside of each cell, we determine upper and lower bounds to irradiance and transparency values due to adaptive sampling. This is efficiently realized by mapping particle data onto one representative ray that is cast through the cell instead of sampling all ${D_{xy}}^2$ rays. The cell bounds are then composited in front-to back to efficiently predict screen space errors due to adaptive sampling and to compute an optimal combination of per-cell sampling levels (see Sec. 4).
We further extend the interval arithmetic so that adaptive sampling in viewing and lateral directions can be combined. To that end, we calculate a representative irradiance and transparency for each cell. Lateral adaptivity is then realized by rendering the cell's irradiance and transparency as a *super-pixel* that spans all pixels the cell covers in screen space (see Sec. 5).

Greedy Optimization: In order to achieve an efficient adaptive ray casting, the degree of adaptivity has to be maximized for each cell for a user-defined screen space error tolerance (see Sec. 4.4). To yield higher speed-ups, the error prediction can be relaxed by removing the lateral error in the sampling error analysis. This leads to a performance optimized greedy algorithm which practically still satisfies the error bounds while allowing for higher sampling levels in viewing direction and for an alternative super-pixel rendering (see Sec. 5).

Cell Merging: Consecutive cells in viewing direction that support higher sampling levels are merged to reduce the number of particles to be sampled and keep a constant number of samples per cell (see Sec. 6.2).

Adaptive Ray Casting: The final volume ray casting algorithm simplifies to an entirely thread-coherent front-to-back traversal of cells. Cells are either rendered as super-pixels or else all rays only sample the necessary subset of particles that has been assigned to the cell (see Sec. 6.3).

4. Sampling Error Analysis Framework

The goal of our sampling error analysis is to determine sampling levels for each cell so that a user-defined error tolerance E_I is not exceeded by the screen space error. Therefore, we propose a hierarchical interval arithmetic scheme to determine upper and lower bounds to the irradiance and transparency on the level of ray bundles. These bounds determine the screen space error via the volume rendering equation (Eq. 3). In the following, upper and lower bounds will be denoted with superscripts $^\top$ and $^\perp$, respectively, and intervals with $^{\top\!\!\!\perp}$. We use standard interval arithmetic operations on

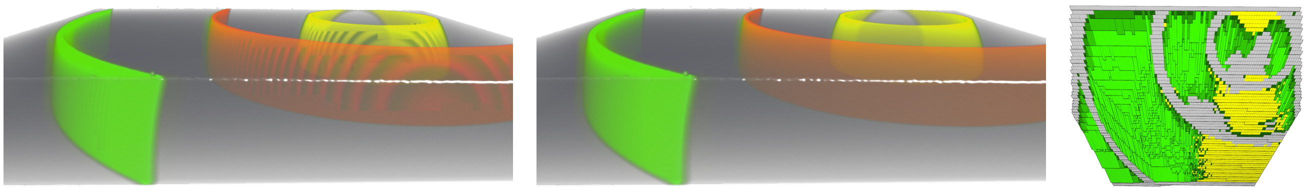


Figure 3: A radially increasing concentration profile rendered with a complex transfer function. The center of the concentration profile is in the upper right of all images. High-frequency transfer functions introduce visible artifacts if sampling rates are naïvely reduced (left). Our screen space error analysis locally adjusts sampling rates to retain visible features (middle). Only the volume covered by gray cells has to be sampled at the highest sampling rate to give correct rendering results, yellow cells are rendered as super-pixels and green cells by reducing the sampling rate in viewing direction (right).

$\top\!\!\!\!\bot$-quantities and denote the width of an interval $x^{\top\!\!\!\!\bot}$ for quantity x as $w(x^{\top\!\!\!\!\bot}) := x^{\top} - x^{\perp}$ [MKC09].

Our error analysis works hierarchically on the level of samples, of cells and finally of whole ray bundles. Inside each cell, we first compute bounds to the quantity field at each sampling depth along the ray bundle. The quantity bounds $Q^{\top\!\!\!\!\bot}$ are mapped to irradiance and transparency sample bounds $i^{\top\!\!\!\!\bot}, t^{\top\!\!\!\!\bot}$ that bound the irradiance and transparency of all rays of the bundle (see Sec 4.1). Sample bounds are composited inside cells to cell bounds $I^{\top\!\!\!\!\bot}, T^{\top\!\!\!\!\bot}$ (see Sec. 4.2). The screen space error analysis (see Sec. 4.3) composites bounds of traversed cells in viewing direction to ray bounds $\mathbb{I}^{\top\!\!\!\!\bot}, \mathbb{T}^{\top\!\!\!\!\bot}$ using different combinations of cell sampling levels. Note that only D_z lower and upper bounds have to be composited inside each cell to yield cell bounds and only one upper and lower cell bound has to be composited per cell along the cell sequence to efficiently calculate the bounds for all rays of a ray bundle.

Based on the error analysis, we determine cell sampling levels l_C^{opt} for all cells C for the final adaptive ray casting. As a direct or analytic identification of the optimal sampling levels is not possible, we greedily optimize sampling levels keeping the width of the error bound below a user defined error tolerance E_I (see Sec. 4.4). Alg. 1 gives pseudocode of the sampling error analysis. Tab. 1 gives an overview of all symbols that will be used throughout the paper.

```
foreach cell $c \in$ perspective grid do
    foreach sample $0 \leq k < D_z$ do
        $Q_k^{\top\!\!\!\!\bot}$ = lateral_quantity_bounds(particle quantities $q_{j_c}$)
        $[i_k^{\top\!\!\!\!\bot}, t_k^{\top\!\!\!\!\bot}]$ = sample_bounds $(Q_k^{\top\!\!\!\!\bot})$
    foreach sampling level $l$ do
        $[I_c^{\top\!\!\!\!\bot,l}, T_c^{\top\!\!\!\!\bot,l}]$ = cell_bounds $(\{i_k^{\top\!\!\!\!\bot}\}, \{t_k^{\top\!\!\!\!\bot}\}, l)$
foreach ray bundle $b$ do
    $\vec{l}$ = greedy_optimization$(E_I, \{I_c^{\top\!\!\!\!\bot,l}\}, \{T_c^{\top\!\!\!\!\bot,l}\})$
```

Algorithm 1: Our sampling error analysis determines bounds for particle quantities and samples in cells, composites sample bounds to cell bounds and finally composites cell bounds along ray bundles to determine ray bundle bounds and thus to the screen space error. Our greedy algorithm finally returns appropriate cell sampling levels $\vec{l}$.

4.1. Lateral Quantity and Sample Bounds

First, we determine the maximum and minimum of all ${D_{xy}}^2$ samples at each sampling depth s_k inside of cells. Since CSPH quantities are affine combinations of particle quantities (cf. Sec. 2.1), sample quantities are bounded by the particle quantities contributing to the lateral neighborhood at depth s_k, i.e.

$$Q_k^{\top} = \max_{|z_j - s_k| < h_j} q_j, \qquad Q_k^{\perp} = \min_{|z_j - s_k| < h_j} q_j,$$

where z_j is the z-coordinate of particle j in view space and h_j its radius. As transfer functions can introduce high frequencies (see Fig. 3), the bounds $Q_k^{\top\!\!\!\!\bot}$ of the quantity field are mapped to irradiance bounds using exhaustive search for extremes in $\tau_I(Q)$ as shown in Fig. 4:

$$i_k^{\top} = \max_{Q \in [Q_k^{\perp}, Q_k^{\top}]} \tau_I(Q), \qquad i_k^{\perp} = \min_{Q \in [Q_k^{\perp}, Q_k^{\top}]} \tau_I(Q).$$

In practice, $i^{\top}, i^{\perp}$ are accessed from a 2D-texture using $Q^{\perp}$ and $Q^{\top}$ as lookup coordinates. Analogously, transparency bounds $t_k^{\top\!\!\!\!\bot}$ are computed and stored by analyzing $\tau_T(q)$. Note that only D_z sample bounds are calculated each of which strictly bounds the ${D_{xy}}^2$ samples of the ray bundle passing through the cell.

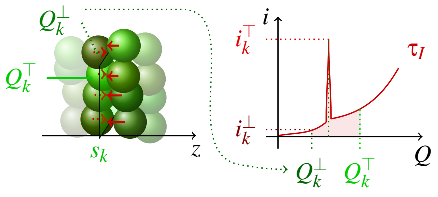


Figure 4: For each sampling depth s_k, quantity bounds $Q_k^{\top}, Q_k^{\perp}$ are determined by finding the maximum and minimum quantities of the particles that contribute to s_k depicted by the red arrows (left). We then search the transfer function's maximum and minimum $i_k^{\top}, i_k^{\perp}$ in the parameter range $[Q_k^{\perp}, Q_k^{\top}]$ (right).

Table 1: Symbols and their interpretation as used throughout the text.

Symbol	Interpretation
$w(x^{\top\!\!\!\!\bot})$	Width of interval $x^{\top\!\!\!\!\bot}$
E_I	Error tolerance $\in [0,1]$
q_i	Quantity of particle i
$Q_k^{\top\!\!\!\!\bot}$	Quantity bounds at sampling depth k
$i_k^{\top\!\!\!\!\bot,l}$ / $t_k^{\top\!\!\!\!\bot,l}$	Irradiance / Transparency sample bounds at sampling depth k and level l
$I_C^{\top\!\!\!\!\bot,l}$ / $T_C^{\top\!\!\!\!\bot,l}$	Irradiance / Transparency cell bounds of cell C and level l
$\mathbb{I}_L^{\top\!\!\!\!\bot,\vec{l}}$ / $\mathbb{T}_L^{\top\!\!\!\!\bot,\vec{l}}$	Irradiance / Transparency ray bundle bounds for cell sequence $l_0, \ldots, l_L$
$l_C^{\max}$	Maximum sampling level of cell C
$\vec{l}$	Vector of sampling levels of cell sequence $l_0, \ldots, l_{\text{last}}$
D_{xy} / D_z	Resolution of a cell in xy- and z-directions
$\Delta i_l = 2^l$	Discrete sampling step size at level l

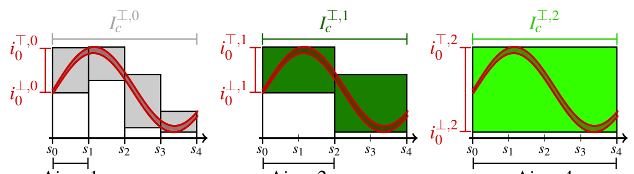
4.2. Cell Bounds

The second stage of our hierarchical interval arithmetic calculates bounds for cell C at sampling level l. For $l = 0$, cell irradiance and transparency bounds $I_C^{\top\!\!\!\!\bot,0}$, $T_C^{\top\!\!\!\!\bot,0}$ are found by compositing sample bounds $i_k^{\top\!\!\!\!\bot}, t_k^{\top\!\!\!\!\bot}$ using the volume rendering Eq. 3. Note that the sample bounds already include the lateral variation of the ray bundle.

If a cell is sampled at a higher level l, less samples are used. Each of these coarser samples, however, represents a larger span of $\Delta_l = 2^l$ samples of level 0 along the ray. Thus, samples of higher levels are bounded by the minimum and maximum of the original samples at level 0 they span. For samples $k = 0,\dots,D_z - 1$ we get

$$i_k^{\top,l} = \max_{j\in\left\{\left\lfloor\frac{k}{\Delta_l}\right\rfloor\cdot\Delta_l,\dots,\left\lceil\frac{k}{\Delta_l}\right\rceil\cdot\Delta_l\right\}} \left\{i_j^{\top}\right\}.$$

For $i_k^{\perp,l}$, min is used instead of max. Analogously, $t_k^{\top\!\!\!\!\bot,l}$ is calculated using transparency samples. After determining the proper set of sample bounds for level l in cell C, the samples are composited using the volume rendering Eq. 3, to yield cell bounds $I_C^{\top\!\!\!\!\bot,l}$ and $T_C^{\top\!\!\!\!\bot,l}$ as shown in Fig. 5.



$I_C^{\top\!\!\!\!\bot,0}$
$i_0^{\top,0}$
$i_0^{\perp,0}$
s_0 s_1 s_2 s_3 s_4
$\Delta i_c = 1$
$I_C^{\top\!\!\!\!\bot,1}$
$i_0^{\top,1}$
$i_0^{\perp,1}$
s_0 s_1 s_2 s_3 s_4
$\Delta i_c = 2$
$I_C^{\top\!\!\!\!\bot,2}$
$i_0^{\top,2}$
$i_0^{\perp,2}$
s_0 s_1 s_2 s_3 s_4
$\Delta i_c = 4$

(a) Level $l = 0$ (b) Level $l = 1$ (c) Level $l = 2$

Figure 5: Computation of irradiance bounds of cell C for three sampling levels. With larger sampling step sizes $\Delta i_C = 4$, bounds (right, green area) differ from accurate bounds at $\Delta i_C = 1$ (left, gray area) to the signal (red area). This may lead to sampling errors. Compositing sample bounds $i_k^{\top\!\!\!\!\bot,l}$ and $t_k^{\top\!\!\!\!\bot,l}$ yields cell irradiance and transparency bounds $I_C^{\top\!\!\!\!\bot,l}$, $T_C^{\top\!\!\!\!\bot,l}$.

High irradiance variations, i.e. $w(I_C^{\top\!\!\!\!\bot,l}) \gg 0$, strongly influence the error bounds. Thus, we limit the sampling level of each cell C to $l_C^{\max} = \arg\max_l \left\{w(I_C^{\top\!\!\!\!\bot,k}) < E_I\right\}$, according to the cell's maximum potential error. Thus, we prevent that coarser sampling of a single cell exhausts the error tolerance, yielding a better distribution of the error tolerance between cells. The more cells are sampled coarsely the higher the speed-up.

An important aspect is the handling of cells C which contain surface particles. As particles only model the fluid but not the air phase, irradiance and transparency bounds cannot be computed correctly. We detect surface particles using the approach of Orthmann et al. [OHB*13]. If, e.g., only one particle is sampled inside a cell and it is only hit by some rays while all other rays pass through empty space, the cell bounds would still yield $w(I_C^{\top\!\!\!\!\bot,0}) = 0$ although visible errors can be introduced into the image. In order to prevent erroneous adaptive sampling in surface cells, we set $l_C^{\max} = 0$. Apart from this, surface particles are treated just like bulk particles.

4.3. Screen Space Error Analysis

The final stage of our hierarchical interval arithmetic computes bounds for ray bundles by compositing cell bounds. As ray bundle bounds automatically bound each single ray of the bundle, they also naturally bound the screen space error. To calculate ray bundle bounds over a cell sequence $C_1,\dots,C_L$ with cell sampling levels $\vec{l} = (l_1,\dots,l_L)$, we again apply the volume rendering equation as

$$\mathbb{I}_L^{\top\!\!\!\!\bot,\vec{l}} = \sum_{C=1}^{L} I_C^{\top\!\!\!\!\bot,l_C} \prod_{D=1}^{C-1} T_D^{\top\!\!\!\!\bot,l_D}, \qquad \mathbb{T}_L^{\top\!\!\!\!\bot,\vec{l}} = \prod_{C=1}^{L} T_C^{\top\!\!\!\!\bot,l_C}. \tag{4}$$

The width of the ray bundle bounds $w(\mathbb{I}_L^{\top\!\!\!\!\bot,\vec{l}})$ is an upper bound to the screen space error that sampling at cell sampling levels $\vec{l}$ may introduce. However, we still have to find an optimal choice of $\vec{l}$ that satisfies the user's error tolerance $w(\mathbb{I}_L^{\top\!\!\!\!\bot,\vec{l}}) < E_I$. Therefore, we propose the following greedy algorithm.

4.4. Greedy Optimization of Sampling Levels

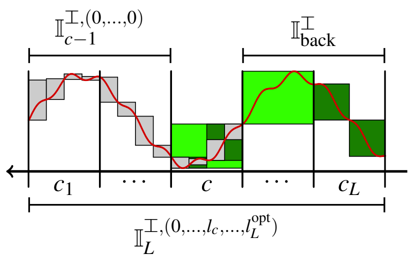
We start by compositing bounds from Eq. 4 using $l_C = 0$ for all cells C, i.e., $\vec{l} = \vec{0} = (0,\dots,0)$. Walking backwards from cell $C_L,\dots,C_1$, we then greedily increase sampling levels, while the error tolerance E_I for the ray bundle is not exceeded. Given bounds $\mathbb{I}_L^{\top\!\!\!\!\bot,\vec{0}}$, $\mathbb{T}_L^{\top\!\!\!\!\bot,\vec{0}}$ over the full cell sequence, we decompose Eq. 4, split off the last cell and replace it by coarser sampling level l_L to get the new bounds $\mathbb{I}_L^{\top\!\!\!\!\bot,(0,\dots,0,l_L)}$. The sampling level is increased while $w(\mathbb{I}_L^{\top\!\!\!\!\bot,(0,\dots,0,l_L)}) < E_I$ until level $l_L^{\max}$ is reached. Hence, it sets

$$l_L^{\text{opt}} = \arg\max_{l_L \le l_c^{\max}} \left\{w(\mathbb{I}_L^{\top\!\!\!\!\bot,(0,\dots,0,l_L)}) < E_I\right\}. \tag{5}$$

Having decided on the sampling level for cell C_L, we collect the final results for cell C_L in background irradiance $\mathbb{I}_{\text{back}}^{\top\!\!\!\!\bot} = I_L^{\top\!\!\!\!\bot,l_L^{\text{opt}}}$. As shown in Fig. 6, the algorithm proceeds backwards sequentially testing coarser sampling levels l_C in cell C in back-to-front order:

$$\mathbb{I}_L^{\top\!\!\!\!\bot,(0,\dots,0,l_C,l_{C+1}^{\text{opt}},\dots,l_L^{\text{opt}})} = \mathbb{I}_{C-1}^{\top\!\!\!\!\bot,\vec{0}} + \mathbb{T}_{C-1}^{\top\!\!\!\!\bot,\vec{0}} \cdot (I_C^{\top\!\!\!\!\bot,l_C} + T_C^{\top\!\!\!\!\bot,l_C} \cdot \mathbb{I}_{\text{back}}^{\top\!\!\!\!\bot}).$$

Analogously to Eq. 5 we compute the optimal level l_C^{opt} and, finally,



$\mathbb{I}_{c-1}^{\top\!\!\!\!\bot,(0,\dots,0)}$
$\mathbb{I}_{\text{back}}^{\top\!\!\!\!\bot}$
c_1 $\cdots$ c $\cdots$ c_L
$\mathbb{I}_L^{\top\!\!\!\!\bot,(0,\dots,l_c,\dots,l_L^{\text{opt}})}$

Figure 6: Back to front exchange of sampling levels for an unknown signal (red). In the depicted step, the greedy algorithm estimates the screen space error (green and gray areas) for different sampling levels l_C of cell C, i.e. by compositing cell bounds $I_C^{\top\!\!\!\!\bot,l_C}$ with foreground irradiances $\mathbb{I}_{C-1}^{\top\!\!\!\!\bot,\vec{0}}$ and the already adaptively sampled background $\mathbb{I}_{\text{back}}^{\top\!\!\!\!\bot}$.

update the background irradiances:

$$\mathbb{I}_{\text{back}}^{\top\!\!\!\!\bot} \leftarrow I_C^{\top\!\!\!\!\bot,l_C^{\text{opt}}} + T_C^{\top\!\!\!\!\bot,l_C^{\text{opt}}} \cdot \mathbb{I}_{\text{back}}^{\top\!\!\!\!\bot}$$

Our derivation of error bounds using interval arithmetic guarantees, if $w(\mathbb{I}^{\top\!\!\!\!\bot,\vec{l}}) < E_I$, the screen space error stays below E_I for levels $\vec{l}$.

5. Performance Optimizations

Although the sampling error analysis framework reveals proper bounds to the screen space error, in practice the bounds are too strict and, thus, the resulting speed-up is rather limited. Furthermore, so far only adaptivity in viewing direction is considered. In this section we propose two extensions to the error analysis framework that allow to combine adaptive sampling in viewing direction and lateral adaptivity and result in significant performance speed-ups while causing only negligible violations of the screen space error bounds.

5.1. Relaxed Error Estimation

While cells with a large width $w(I^{\top\!\!\!\!\bot,0})$ strongly increase the error estimation along rays, they do not cause any screen space error if sampled at the highest sampling rate. We interpret $w(I_C^{\top\!\!\!\!\bot,0})$ as a measure of lateral variation and by removing it from the interval widths of higher levels, we can separate the effects of adaptive sampling in viewing direction from potential errors due to lateral variation. We thus propose a relaxation approach that partially substracts $w(I^{\top\!\!\!\!\bot,0})$ from the lower bounds of $l = 0$ as

$$I_{C,\text{relax}}^{\perp,0} := I_C^{\top,0} - (1 - E_{\text{relax}})\, w(I_C^{\top\!\!\!\!\bot,0}),$$

and for higher levels $l > 0$ as

$$I_{C,\text{relax}}^{\perp,l} := I_{C,\text{relax}}^{\perp,0} + (I_C^{\perp,l} - I_C^{\perp,0}).$$

The user-defined relaxation parameter $E_{\text{relax}} \in [0,1]$ controls the influence of cell bounds on level $l = 0$ on the error estimation of ray bundles. $E_{\text{relax}} = 1$ completely removes the influence of the width of level $l = 0$ and $E_{\text{relax}} = 0$ gives a strict error estimation without relaxation. Transparency is adjusted analogously.

As relaxing, i.e. decreasing, the upper transparency bound could lead to severe underestimates for the errors behind the current cell, we leave the upper error bounds unchanged. Thus, we ensure that errors in the background remain visible and are adequately accounted for in the error analysis.

Note, that our relaxation approach does not affect the maximum sampling levels $l_C^{\max}$ per cell, as they also account for the lateral error in the ray bundle (cf. Sec. 4.2).

5.2. Additional Lateral Adaptive Sampling Using Super-Pixels

In cells that contain large differences in viewing direction but sample bounds $i^{\top\!\!\!\!\bot}$ of small width, reducing the sampling rate in viewing direction causes large errors. In these cases, adapting the lateral sampling rate instead, i.e., the number of cast rays, is a promising alternative. To this end, we can directly utilize the non-relaxed cell bounds and render the mean irradiance $\frac{I_C^{\top,0}+I_C^{\perp,0}}{2} =: I_{\text{SP}}$ and transparency $\frac{T_C^{\top,0}+T_C^{\perp,0}}{2} =: T_{\text{SP}}$ as super-pixels in the final image. Instead of sampling any particles we just composite $(I_{\text{SP}}, T_{\text{SP}})$ to the accumulated image for all pixels covered by the cell. This approach optimally reuses the previously calculated cell bounds.

Apparently, super-pixel rendering is by far faster than sampling particles but causes larger screen space errors. To estimate this error, the relaxation scheme cannot be used. By relaxing bounds, the lateral error is removed from the error estimation, however, rendering super-pixels introduces exactly these lateral errors. To properly bound errors due to the reduced lateral resolution of super-pixels, we have to use the original cell bounds of level 0.

5.3. Combined Greedy Optimization

The lateral sampling optimization using super-pixels can be combined efficiently with the adaptivity in viewing direction using the proposed error estimation. In order to decide between super-pixel rendering and adaptive sampling in viewing direction, we sort the relaxed cell bounds $I_{C,\text{relax}}^{\top\!\!\!\!\bot,l}, l = 0,\dots,l^{\max}$ and the non-relaxed cell bound $I_C^{\top\!\!\!\!\bot,0}$ in ascending order. The greedy algorithm just works as described in Sec. 4.4, only if $I_C^{\top\!\!\!\!\bot,0}$ is found to be the largest acceptable error, the super-pixel will be used during ray casting.

In Fig. 3, we rendered a radially increasing concentration profile using both optimizations. Areas where concentration gradient and viewing rays run perpendicularly have small sample bound widths, hence, they are rendered using super-pixels (yellow cells). In other areas only adaptive sampling in viewing direction is applicable (green cells).

6. Implementation Details

We implemented our algorithm using Nvidia CUDA 7.5 and OpenGL. We will first give the details of the particle-to-cell mapping which in fact constructs the perspective grid as particle access structure (Sec. 6.1). Then, we will present our cell merging approach which allows to reduce the sampling rate at a constant number of samples per cell and prevents unnecessary duplicate particle accesses (Sec. 6.2). Last we will give some details of our ray casting (Sec. 6.3).

6.1. Particle Access via Perspective Grids

In the perspective grid structure, each cell C stores references to particles that overlap the cell's volume $\Omega_C := \{\boldsymbol{x} \mid C(\boldsymbol{x}) = C\}$ which is defined via the indexing function $C : \mathbb{R}^3 \to \mathbb{N}_0$

$$C(\boldsymbol{x}) = (C_x(\boldsymbol{x})M_y + C_y(\boldsymbol{x}))M_z + C_z(\boldsymbol{x}), \tag{6}$$

which subdivides the view space into $M_x \times M_y \times M_z$ view-aligned cells. The cell coordinates $(C_x(\boldsymbol{x}), C_y(\boldsymbol{x}), C_z(\boldsymbol{x}))$ at position $\boldsymbol{x} = (x,y,z)^T$ in view space are given as

$$\left(\left\lfloor \frac{x}{\gamma\alpha(z)} + \frac{M_x}{2} \right\rfloor, \left\lfloor \frac{y}{\alpha(z)} + \frac{M_y}{2} \right\rfloor, \left\lfloor M_z \left(\frac{\ln\left(\frac{z}{s_{\text{near}}}\right)}{\ln\left(\frac{s_{\text{far}}}{s_{\text{near}}}\right)} \right) \right\rfloor \right). \tag{7}$$

Here, C_z is derived using the inverse of the following perspective transformation that maps samples from uniform sampling space to non-uniform view space [FAW10]:

$$s_i = s_{\text{near}} \left(\frac{s_{\text{far}}}{s_{\text{near}}} \right)^{\frac{i}{N}}. \tag{8}$$

C_x and C_y are defined by the window's aspect ratio γ. $\alpha(z) = \frac{2z}{M_y}\tan\left(\frac{\sigma}{2}\right)$ is the cell height at distance z, which depends also on the current field of view σ of the view frustum.

Assigning particles j to cells C yields a list of particle-cell-pairs (j, C). Sorting the particle-cell pairs by the cell index C yields cell-particle-pairs that describe the perspective access structure. Only cells are referenced to which at least one particle contributes and only particles are present that contribute to at least one cell. Thus, empty cells are skipped implicity and frustum culling is performed (cf. Fig. 2).

As ray bundles always sample a whole cell sequence in viewing direction, we also store the indices of the first and last cell of the sequence. To allow for a fast access to particles inside cells, we store the first index j_C and the number N_C of the relevant cell-particle-pairs for each cell. During error estimation and ray casting, the "traversal logic" reduces to a simple loop over the relevant cells in the sequence of cell-particle pairs.

6.2. Cell Merging

Although our adaptive sampling can increase the sampling level along rays inside cells, this results in a different number of samples per cell and complicates the ray casting. Furthermore, particles in subsequent cells often are redundantly sampled. We thus propose to pairwise merge neighboring cells that allow for sampling at a higher level. Thus, overlapping particle references in the newly merged cell can be removed and, instead of adjusting the number of samples per cell, only the sampling distance is adjusted. This both simplifies the ray casting and reduces the number of particles that have to be sampled redundantly.

Merging of cells is realized level by level, starting from level 0. Figure 7 shows two subsequent merging steps. Neighboring cells are only merged if they both allow for sampling at the next higher level. During the process of setting up the grid, we bit-shift the linear cell index C one bit to the left and set the least significant bit if a particle contributes to samples of the subsequent cell $C+1$. The cell index then also directly indicates if a particle is redundantly referenced in the subsequent cell. After sorting the particle-cell-pairs, all redundantly referenced particle indices of a cell contiguously lie in memory and can be removed easily.

We only merge cells that are sampled adaptively in viewing direction. Cells that are rendered as a super-pixel would not benefit from merging because no particles are accessed during ray casting.

6.3. Adaptive Ray Casting

Alg. 2 depicts the pseudocode of our ray casting algorithm. Since consecutive cells in viewing direction are neighbors in memory, no specific cell finding logic is required. Either a super-pixel can be rendered or particles have to be sampled. As all rays then have to sample the same set of particles, particle data for all $D_{xy} \times D_{xy}$ adjacent rays can efficiently be cached using shared memory (cf. red lines in Alg. 2) and each particle has to be read only once per cell. Using a small thread-local cache, particles scatter their data to D_z samples of a ray at once to further reduce memory traffic. In our GPU-implementation of Alg. 2, we use $D_{xy} = 8$ and $D_z = 16$ so the maximum sampling level is $\log_2 D_z = 4$.

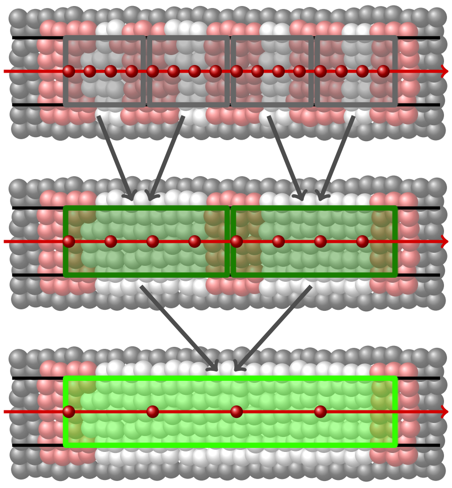


Figure 7: Two iterations of our proposed cell merging. The number of samples remains constant inside the merged cells, however, the distance between samples (red) is doubled. Redundant particle references (rose) between cells are removed and uniquely referenced in the merged cell (white).

7. Results and Discussion

In order to demonstrate the performance and to evaluate the image quality, our proposed adaptive ray casting has been tested in five scenarios: The *Checker Board scene* with cubes of varying concentrations that diffuse over time (see Fig. 8), the *Mixer scene* simulating the mixing of solvent with dye streaming from an inlet (see Fig. 9), the *HPG 2016 scene* with fluid letters splashing to the ground (see Fig. 1), the static *Radial Concentration scene* of a fluid with radially increasing concentrations rendered with an extreme transfer function (see Fig. 3), and the *Flubber scene* with two fluids mixing while orbiting a virtual center of gravity (see Fig. 10). The relaxation factor for the error analysis was set to $E_{\text{relax}} = 1$ in all examples and super-pixel rendering was enabled if not stated otherwise. Simulations and renderings were carried out on an NVIDIA GeForce GTX Titan with 6 GB of VRAM. All scenes were rendered on a 1024^2 viewport. Surfaces were also ray cast using our view-aligned access structure. However, to get smoothed surfaces, an increased particle support radius was employed. As our focus lies on the adaptive volume ray casting, timings are not included.

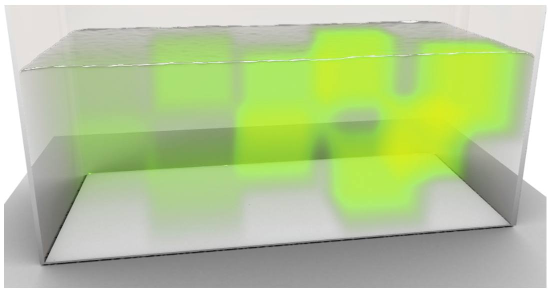


Figure 8: 3D checker board of increasing concentrations from left to right and front to back.

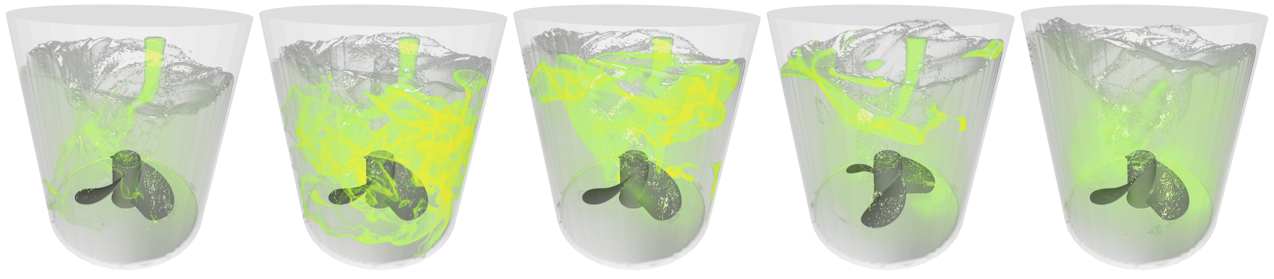

(a) Frame 130 (b) Frame 330 (c) Frame 530 (d) Frame 642 (e) Frame 930

Figure 9: A mixer is causing a stream of green dye to mix with solvent in the fluid tank. Above, five of the 1000 frames we rendered are shown.

```
$I = \mathbf{0}, T = 1$,   // initialize ray irradiance and transparency
$x[D_z] = \mathbf{0}$, $Q[D_z] = 0$, $V[D_z] = 0$   // sampling positions, sampled quantities
foreach cell $C$ in $C_1, \dots, C_L$ do
    $j_C$, $N_C$   // index to particle array and number of particles (cf. Sec. 6.1)
    $i_C$, $\Delta_{i_C}$   // linear cell index (cf. Eq. 6) and cell sampling distance
    $[j_C, N_C, i_C, \Delta_{i_C}]$ = read_celldata($C$);
                        Super-pixel Rendering
    if C can be rendered as Super-pixel then
        $[I, T] = \mathbf{composite}(I, T, I_{\mathrm{SP}}, T_{\mathrm{SP}}))$ //(cf. Sec. 5.2)
        continue
                        Sampling
    foreach sample $k$ in $0, \dots, D_z - 1$ do
        $Q[k] = V[k] = 0$   //initialize sampled quantity and volume
        $\boldsymbol{x}[k] = \boldsymbol{x}(s(i_C + \Delta i_C\, k))$   //get ray sampling position $\boldsymbol{x}$ in view space
    foreach particle $j$ in $j_C, \dots, j_C + N_C - 1$ do
        $[\boldsymbol{x}_j, h_j, q_j, V_j]$ = read_particledata($j$);
        foreach sample $k$ in $0, \dots, D_z - 1$ do
            $Q[k] = Q[k] + q_j\, V_j\, W_j(\boldsymbol{x}[k])$
            $V[k] = V[k] + V_j\, W_j(\boldsymbol{x}[k])$
    foreach sample $k$ in $0, \dots, D_z - 1$ do
        if $(V[k] > 0)$ $Q[k] = Q[k]/V[k]$   //CSPH normalization (cf. Eq. 2)
                        Compositing
    foreach sample $k$ in $0, \dots, D_z - 1$ do
        $\Delta s_k = s(i_C + \Delta i_C\,(k+1)) - s(i_C + \Delta i_C\, k)$   // view space distance
        $[I, T] = \mathbf{composite}(I, T, \tau_I(Q[k]), \tau_T(Q[k])^{\Delta s_k})$
```

Algorithm 2: Thread-coherent volume ray casting featuring adaptive sampling step sizes (green), super-pixel rendering and efficient shared memory access (red). Particles contribute to D_z ray samples at once using a thread-local cache.

Tab. 2 shows particle counts, timings and speed-ups as well as errors using different tolerances E_I averaged over all frames of the respective scene. Errors are given as maximum absolute difference over all pixels in any color or alpha channel $\in [0,1]$ compared to the non-adaptive rendering E_I='–'. Values of $\{0.001, 0.004, 0.01\}$ relate to error values $\{0.255, 1.02, 2.55\}$ in the respective 8-bit value range $[0,255]$, respectively.

The following discussion addresses the image quality and the performance characteristics of our adaptive volume ray casting approach and will also compare our method to previous work.

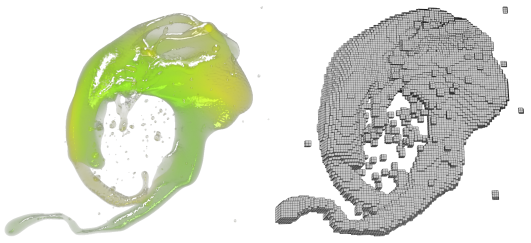

(a) Flubber (b) Sparse grid

Figure 10: A frame of our Flubber scene and the respective sparse grid showing the large surface to volume ratio of this scene.

Table 2: GPU timings and speed-ups of our adaptive ($E_I \geq 0$) and non-adaptive (E_I = '–') volume rendering. 'Grid' is the particle-cell assignment and setting up the view-aligned grid, 'Adapt' is the sampling error analysis and the merging of cells, 'RC' is the ray casting and 'Total' gives the total time to render. 'Error' is given as maximum absolute single pixel difference in any color or alpha channel compared to the non-adaptive rendering. 'Speedup' relates the timing to timings for E_I = '–'. Results are averaged over all frames of the given scenes.

	Image quality		**Timing (in ms)**			**Speedup**
Scene (#Part.)	E_I	Error $\in [0,1]$	Grid	Adapt	**RC** (Total)	**RC** (Total)
Flubber	–	–	7	–	**66** (73)	**1** (1)
(500 K)	0.004	7.2e-04		5	**44** (56)	**1.5** (1.3)
No surface	0.004	$\gg E_I$	7	8	**19** (34)	**3.5** (2.15)
	–	–		–	**134** (157)	**1** (1)
Checker	0.001	3.3e-05		14	**109** (146)	**1.23** (1.08)
Board	0.004	2.1e-04	23	15	**87** (125)	**1.54** (1.26)
(1.2 M)	0.01	0.0011		16	**75** (114)	**1.79** (1.38)
	–	–		–	**297** (346)	**1** (1)
Mixer	0.001	7.4e-04	49	28	**181** (254)	**1.64** (1.36)
(2.5 M)	0.004	0.0027		29	**164** (242)	**1.81** (1.43)
	0.01	0.0056		29	**152** (230)	**1.95** (1.5)
No greedy alg.	0.004	0.016	49	23	**121** (193)	**2.45** (1.79)
Object space		0	75	–	**981** (1056)	**0.3** (0.33)
	–	–		–	**317** (388)	**1** (1)
HPG2016	0.001	5.4e-04		31	**217** (319)	**1.46** (1.21)
(5.3 M)	0.004	0.002	71	32	**191** (294)	**1.66** (1.32)
	0.01	0.0035		33	**173** (280)	**1.83** (1.38)
Radial	–	–		–	**598** (677)	**1** (1)
Concentrations	0.001	6.2e-04		49	**460** (588)	**1.3** (1.15)
(10 M)	0.004	0.0013	79	50	**340** (469)	**1.76** (1.44)
	0.01	0.002		50	**290** (419)	**2.06** (1.62)
Only z-Adapt	0.004	0.0014		59	**389** (527)	**1.54** (1.28)

Regarding *image quality*, the errors due to our adaptive volume ray casting of the Flubber, the Checker Board, and the Radial Concentration scenes always stayed below the error tolerance. For both the Mixer and HPG 2016 scenes, errors of all frames stayed below the error tolerance for $E_I = 0.01$ and $E_I = 0.004$. For $E_I = 0.001$, however, the error exceeded the tolerance for 1 of the 1200 frames of the Mixer scene and for 1 of the 1500 frames of the HPG 2016 scene. Table 3 summarizes the errors of our approach for all frames of all scenes. Figure 11 shows the error behaviour for the Mixer scene over the full 1200 simulation frames. The sampling error analysis allowed the adaptive sampling to very precisely exhaust the error tolerance E_I. We additionally rendered the Mixer scene using $l_C^{\max}$ for all cells C (cf. Sec. 4.2), i.e., we only locally limited the sampling level for each cell without applying the greedy algorithm to control the screen space error along cell sequences of rays. The error of 0.016 exceeded the user-defined tolerance $E_I = 0.004$ (cf. row 'No greedy alg.' in Tab. 2) which indicates that the error estimation has to consider the whole cell sequence along ray bundles.

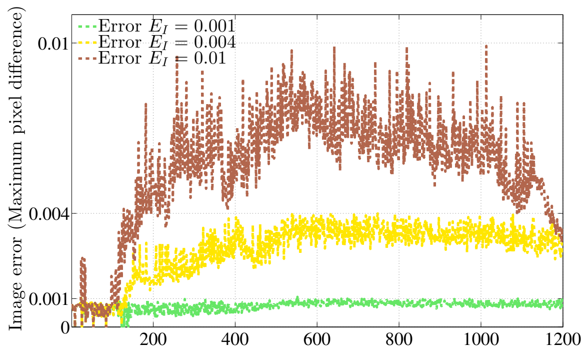


Figure 11: Errors of the 1200 simulation frames (x-axis) of our Mixer scene. Errors are displayed as dotted lines against the right y-axis.

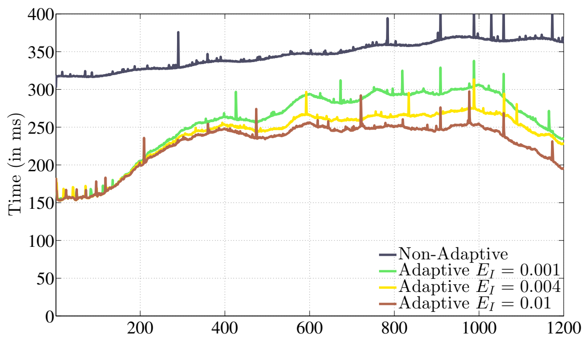


Figure 12: Timings of the 1200 simulation frames (x-axis) of our Mixer scene. Timings are displayed using solid lines against the left y-axis.

Considering *performance*, our adaptive sampling yields total speed-up factors between 1.08 and 1.62 in all scenes (cf. Tab. 2).

Figure 12 shows the detailed time analysis of the Mixer scene. We observed speed-up factors of about 2 for all values of E_I within the first 100 frames. With increasing scene complexity (appearance of iso-surfaces and spreading of dye) the speed-up factors for $E_I =$ 0.001 drop to 1.15 and for $E_I = 0.01$ to between 1.18 and 1.4. Towards the end of the sequence, the scene complexity decreases again (cf. Fig. 9, right) allowing for increasing speed-ups.

Table 3: Error statistics of our adaptive sampling for different scenes and tolerances E_I. 'Err$_{\max}$' gives the maximum single pixel error of all frames, '%Rays>E_I' gives the maximum percentage of erroneous rays over the total number of cast rays for the respective frame and '#Rays>E_I' gives the number of frames of the scene that exceeded the error tolerance.

Scene	E_I	E_{relax}	Err$_{\max}$	%Rays>E_I	#Frames>E_I
Flubber	*	1	$< E_I$	0	0
Checker Board	*	1	$< E_I$	0	0
Mixer	0.001	1	0.00106	0.0125 %	1/1200
	0.004, 0.01	1	$< E_I$	0	0
HPG 2016	0.001	1	0.00101	0.0004 %	1/1500
	0.004, 0.01	1	$< E_I$	0	0
Radial Concentrations	*	1	$< E_I$	0	0

Apparently, the variation in the speed-up factors depends on the homogeneity of the irradiance and transparency and the given error tolerance. Also, large surface to volume ratios impair speed-up factors as surface cells are excluded from adaptive sampling. To show this effect, we rendered the Flubber scene with surface detection disabled (cf. row 'No surface' in Tab. 2) and achieved a speed-up of 2.21 for $E_I = 0.004$. Although this causes visible artifacts where super-pixels are rendered for cells that are not fully covered with particles, it indicates that there is potential for further speed-ups.

In order to demonstrate the benefits of combined lateral adaptivity and adaptivity in viewing direction, we rendered the Radial Concentration scene using only adaptivity in viewing direction with $E_{\text{relax}} = 1$ and $E_I = 0.004$ and achieved a speed-up of 1.28 (cf. row 'Only z-Adapt' in Tab. 2). The speed-up with lateral adaptivity using super-pixels was 1.44 at an even lower screen space error.

A *comparison to previous work* was done for the Mixer scene which we rendered using an object-space particle access structure [OKK10] (cf. row 'Object space' in Tab. 2). We observed a speed-up of about 3 using only our view-aligned access structure.

8. Conclusions

In this paper an adaptive on-the-fly volume ray casting for unstructured particle data has been presented. The approach comprises a sparse perspective, view-aligned grid as access structure for particles and does not require any pre-computations. Inside each grid cell, sampling rates can locally be adapted both in viewing and lateral direction. The presented on-the-fly sampling error analysis for volume rendering of SPH-based quantity fields allows to precisely estimate screen space errors due to adaptive sampling. A greedy algorithm optimizes the adaptive sampling for each cell according to a user-defined error tolerance. The per-cell sampling information is used during ray casting to shift computational resources to salient regions of the fluid volume. Our proposed algorithm leads to significant rendering speed-ups without sacrificing image quality.

References

[AIAT12] Akinci G., Ihmsen M., Akinci N., Teschner M.: Parallel surface reconstruction for particle-based fluids. *Computer Graphics Forum 31* (2012), 1797–1809. 1

[ALD06] ADAMS B., LENAERTS T., DUTRÉ P.: *Particle Splatting: Interactive Rendering of Particle-Based Simulation Data*. Tech. Rep. Tech. Rep. CW 453, Katholieke Universiteit Leuven, 2006. 3

[APKG07] ADAMS B., PAULY M., KEISER R., GUIBAS L. J.: Adaptively sampled particle fluids. *ACM Trans. Graph. 26* (2007). 2, 3

[BHMF08] BEYER J., HADWIGER M., MÖLLER T., FRITZ L.: Smooth mixed-resolution GPU volume rendering. In *Proc. EG / IEEE VGTC Conf. Point-Based Graphics* (2008), pp. 163–170. 2

[BK02] BONET J., KULASEGARAM S.: A simplified approach to enhance the performance of smooth particle hydrodynamics methods. *J. Applied Mathematics & Computation 126*, 2-3 (2002), 133–155. 2

[BOT01] BØRVE S., OMANG M., TRULSEN J.: Regularized smoothed particle hydrodynamics: A new approach to simulating magnetohydrodynamic shocks. *The Astrophysical Journal Supplement Series 561* (2001), 345–367. 2

[DH92] DANSKIN J., HANRAHAN P.: Fast algorithms for volume ray tracing. In *Proc. 1992 Work. Vol. Vis. - VVS '92* (New York, New York, USA, 1992), vol. I, ACM Press, pp. 91–98. 2

[EHK*06] ENGEL K., HADWIGER M., KNISS J. M., REZK-SALAMA C., WEISKOPF D.: *Real-time Volume Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2006. 1, 2

[FAW10] FRAEDRICH R., AUER S., WESTERMANN R.: Efficient high-quality volume rendering of SPH data. *IEEE Trans. Vis. & Comp. Graph. 16*, 6 (2010), 1533–1540. 1, 2, 3, 6

[FGE10] FALK M., GROTTEL S., ERTL T.: Interactive image-space volume visualization for dynamic particle simulations. In *Proc. SIGRAD Conference* (2010). 1

[FSW09] FRAEDRICH R., SCHNEIDER J., WESTERMANN R.: Exploring the millennium run – scalable rendering of large-scale cosmological datasets. *IEEE Trans. Vis. & Comp. Graph. 15*, 6 (Nov. 2009), 1251–1258. 2

[GS04] GUTHE S., STRASER W.: Advanced techniques for high-quality multi-resolution volume rendering. *Computers & Graphics 28*, 1 (Feb. 2004), 51–58. 2

[GSSP10] GOSWAMI P., SCHLEGEL P., SOLENTHALER B., PAJAROLA R.: Interactive SPH simulation and rendering on the GPU. In *Proc. Symp. Comp. Anim.* (2010), pp. 55–64. 3

[HE03] HOPF M., ERTL T.: Hierarchical splatting of scattered data. In *Proc. IEEE Vis.* (2003), pp. 433–440. 2

[HHK08] HONG W., HOUSE D. H., KEYSER J.: Adaptive particles for incompressible fluid simulation. *Vis. Comput. 24*, 7 (2008), 535–543. 2

[HLSR08] HADWIGER M., LJUNG P., SALAMA C. R., ROPINSKI T.: Advanced illumination techniques for GPU volume raycasting. In *SIGGRAPH Asia Courses* (2008). 1

[HM08] HUNT W., MARK W. R.: Ray-specialized acceleration structures for ray tracing. In *Proc. IEEE Symp. Interactive Ray Tracing* (2008), IEEE, pp. 3–10. 2

[IOS*14] IHMSEN M., ORTHMANN J., SOLENTHALER B., KOLB A., TESCHNER M.: SPH fluids in computer graphics. In *Proc. Eurographics (State-of-the-Art Report)* (2014), pp. 21–42. 1

[KAG*06] KEISER R., ADAMS B., GUIBAS L. J., DUTRÉ P. P., PAULY M.: *Multiresolution Particle-Based Fluids*. Tech. rep., ETH, 2006. 2

[KHW*09] KNOLL A., HIJAZI Y., WESTERTEIGER R., SCHOTT M., HANSEN C., HAGEN H.: Volume ray casting with peak finding and differential sampling. *IEEE Trans. Vis. & Comp. Graph. 15*, 6 (2009), 1571–1578. 2

[LGM*08] LEDERGERBER C., GUENNEBAUD G., MEYER M. D., BÄCHER M., PFISTER H.: Volume MLS ray casting. *IEEE Trans. Vis. & Comp. Graph. 14*, 6 (2008), 1372–1379. 2

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proc. Symp. Comp. Anim.* (2003), pp. 154–159. 2

[MKC09] MOORE R. E., KEARFOTT R. B., CLOUD M. J.: *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009. 4

[Mon05] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Reports on Progress in Physics 68* (2005), 1703–1759. 2

[MSD07] MUELLER M., SCHIRM S., DUTHALER S.: Screen space meshes. In *Proc. Symp. Computer Animation* (2007), pp. 9–15. 3

[NMM*06] NEOPHYTOU N., MUELLER K., MCDONNELL K. T., HONG W., GUAN X., QIN H., KAUFMAN A. E.: GPU-accelerated volume splatting with elliptical RBFs. In *Proc. EuroVis* (2006), pp. 13–20. 1

[OCD11] ONDERIK J., CHLADEK M., DURIKOVIC R.: SPH with small scale details and improved surface reconstruction. In *Proc. Spring Conf. Computer Graphics* (2011). 1, 3

[OHB*13] ORTHMANN J., HOCHSTETTER H., BADER J., BAYRAKTAR S., KOLB A.: Consistent surface model for SPH-based fluid transport. In *Proc. Symp. Comp. Anim.* (2013), pp. 95–103. 5

[OKK10] ORTHMANN J., KELLER M., KOLB A.: Topology-caching for dynamic particle volume raycasting. In *Proc. Vision, Modeling & Visualization (VMV)* (2010), pp. 147–154. 2, 9

[PGSS07] POPOV S., GÜNTHER J., SEIDEL H.-P., SLUSALLEK P.: Stackless KD-tree traversal for high performance GPU ray tracing. *Computer Graphics Forum 26*, 3 (2007), 415–424. 2

[RCSW14] REICHL F., CHAJDAS M. G., SCHNEIDER J., WESTERMANN R.: Interactive Rendering of Giga-Particle Fluid Simulations. *Proceedings of High Performance Graphics 2014* (2014). 1, 3

[RTW13] REICHL F., TREIB M., WESTERMANN R.: Visualization of big SPH simulations via compressed octree grids. In *IEEE International Conference on Big Data, Big Data 2013 (2013)* (2013), pp. 71–78. 2

[SP09] SCHLEGEL P., PAJAROLA R.: Layered volume splatting. In *Proc. Int. Symp. on Visual Computing (ISVC)* (2009), vol. 5876 of *Lecture Notes in Computer Science*, Springer, pp. 1–12. 1

[SSP07] SOLENTHALER B., SCHLÄFLI J., PAJAROLA R.: A unified particle model for fluid solid interactions: Research articles. *Comput. Animat. Virtual Worlds 18*, 1 (2007), 69–82. 1, 3

[vdLGS09] VAN DER LAAN W. J., GREEN S., SAINZ M.: Screen space fluid rendering with curvature flow. In *Proc. Symp. Interactive 3D Graphics & Games* (2009), pp. 91–98. 1, 3

[Wes90] WESTOVER L.: Footprint evaluation for volume rendering. *Computer Graphics 24*, 4 (Sept. 1990), 367–376. 1

[WJP14] WALD I., JOHNSON G. P., PAPKA M. E.: CPU Ray Tracing Large Particle Data with Balanced P-k-d Trees. *IEEE Visualization* (2014), 57–64. 3

[YT13] YU J., TURK G.: Reconstructing surfaces of particle-based fluids using anisotropic kernels. *ACM Trans. Graph. 32*, 1 (Feb. 2013), 5:1–5:12. 1, 3

[ZD15] ZIRR T., DACHSBACHER C.: Memory-Efficient On-The-Fly Voxelization of Particle Data. *Eurographics Symposium on Parallel Graphics and Visualization* (2015). 1, 3

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proc. Conf. Comp. Graph. and interactive techniques* (2001), pp. 371–378. 3

[ZSP08] ZHANG Y., SOLENTHALER B., PAJAROLA R.: Adaptive sampling and rendering of fluids on the GPU. In *Proc. Symp. Point-Based Graphics* (2008), pp. 137–146. 2