Efficient High-Quality Volume Rendering of SPH Data

Roland Fraedrich, Stefan Auer, and Rüdiger Westermann

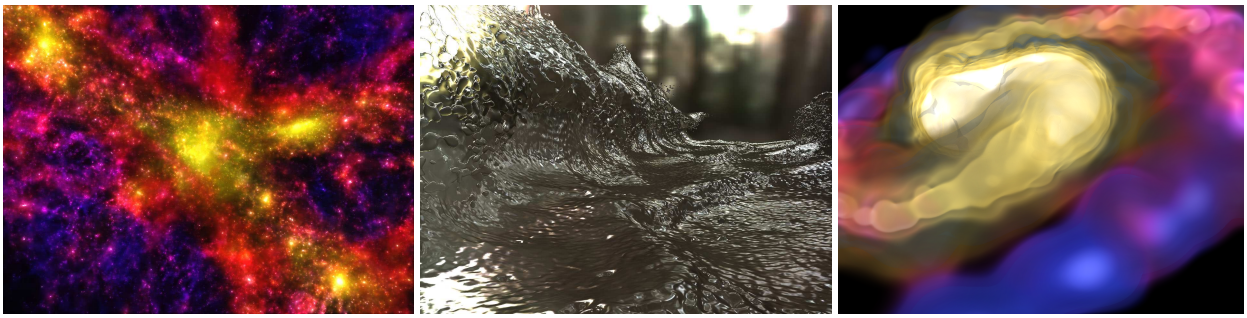

Fig. 1. A novel technique for order-dependent volume rendering of SPH data is presented. It provides rendering options like direct volume rendering (left), iso-surface rendering (middle), and mixed modes (right), and it renders data sets consisting of millions of particles at high quality and speed (42M @ 0.1 fps, 2.2M @ 4.5 fps, 2M @ 1.7 fps, from left to right) on a 1024×1024 viewport.

Abstract—High quality volume rendering of SPH data requires a complex order-dependent resampling of particle quantities along the view rays. In this paper we present an efficient approach to perform this task using a novel view-space discretization of the simulation domain. Our method draws upon recent work on GPU-based particle voxelization for the efficient resampling of particles into uniform grids. We propose a new technique that leverages a perspective grid to adaptively discretize the view-volume, giving rise to a continuous level-of-detail sampling structure and reducing memory requirements compared to a uniform grid. In combination with a level-of-detail representation of the particle set, the perspective grid allows effectively reducing the amount of primitives to be processed at run-time. We demonstrate the quality and performance of our method for the rendering of fluid and gas dynamics SPH simulations consisting of many millions of particles.

Index Terms—Particle visualization, volume rendering, ray-casting, GPU resampling.

1 Introduction

Particle-based simulation techniques like Smoothed Particle Hydrodynamics (SPH) have gained much attention due to their ability to avoid a fixed discretization of the simulation domain. By using an adaptive spatial data structure for particles, the memory requirements of particle-based methods depend only on the size of the fluid domain, but not on the size of the simulation domain. This makes the simulation less demanding on memory but substantially increases the computational load due to a more complex procedure to resolve particle adjacencies.

This limitation also becomes crucial in the rendering of a discrete SPH particle set, which, in general, means reconstructing a volume-covering and continuous density field from this set. To reconstruct a continuous least squares approximation, at every domain point a weighted average of the particle densities contributing to this point has to be computed. This can either be done by resampling and blending the densities onto a grid and using cell-wise interpolation, or instead by directly integrating along the rays of sight through the SPH kernels.

The first approach does not require particles to be processed in any specific order and results in a view-independent volume representation. This representation can be rendered efficiently using standard volume rendering techniques. For example, 3D texture-based volume rendering can be used if the reconstruction is onto a uniform grid. On the other hand, the approach requires a fixed discretization of the simulation domain, with a resolution that is high enough to capture all simulated details. Thus, it takes away much of the advantage of particle-based simulation techniques.

The second approach can leverage the same adaptive data structure as the simulation itself, but it is view-dependent and comes at high computational and memory access load to integrate particle quantities in the correct visibility order along the view rays. Due to these limitations, to the best of our knowledge the use of this approach is currently restricted to off-line visualizations [3, 28].

Our contribution. In this work we introduce a new volume rendering pipeline for SPH data on desktop PCs. This pipeline can efficiently render a continuous density field—or any other scalar quantity—that is given by a discrete SPH particle set. It employs 3D texture-based volume rendering on the GPU and therefore provides different rendering options like direct volume rendering and iso-surface rendering. Some examples are shown in Figure 1.

Similar to previous approaches for the rendering of SPH data the particle quantities are resampled onto a regular 3D grid. In contrast, however, the grid is not fixed to the simulation domain but to the view volume. Thus, it moves with the viewer and discretizes only the visible space, i.e., the view frustum, in front of it.

A further difference is that the grid is not uniform but adaptively discretizes the visible domain. The grid resembles a classical ray-tracing grid which fans out with increasing distance from the view plane [4], but it has a spacing between the grid vertices along the view rays that increases logarithmically. Thus, the grid results in an adaptive sampling of the view frustum with decreasing sampling rate along the viewing direction. We will subsequently call this grid the *perspective grid*.

- *Roland Fraedrich (E-mail: fraedrich@tum.de) is with the with the Computer Graphics and Visualization Group, Technische Universität at München.*
- *Stefan Auer (E-mail: auer@in.tum.de) is with the Computer Graphics and Visualization Group, Technische Universität at München.*
- *Rüdiger Westermann (E-mail: westermann@tum.de) is with the Computer Graphics and Visualization Group, Technische Universität at München.*

Manuscript received 31 March 2010; accepted 1 August 2010; posted online 24 October 2010; mailed on 16 October 2010.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org.

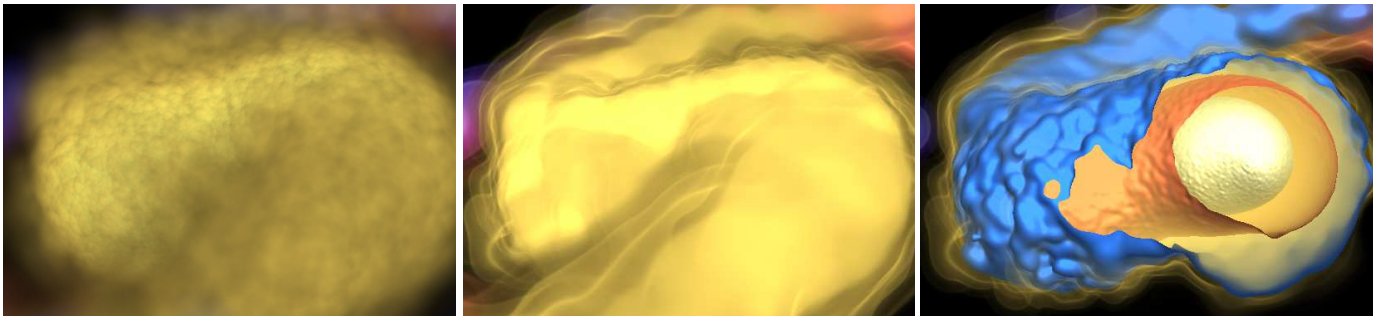

Fig. 2. Order-dependent particle splatting (left) results in vastly different visualizations than volume ray-casting (middle) or hybrid rendering with iso-surfaces (right). Splatting approaches cannot accurately represent high-frequency color and opacity transfer functions and have difficulties in revealing surface-like structures in the data.

The advantage of the perspective grid is twofold: Firstly, by adjusting the sampling distance along the viewing direction a nearly isotropic sampling rate in view space can be enforced. In contrast to a uniform sampling this significantly reduces the number of sample points. Secondly, the perspective grid can be combined effectively with a hierarchical particle representation, in which sets of particles at one level are represented by one enlarged particle at the next coarser level. By always selecting the level of detail with respect to the current sampling rate, the number of particles to be resampled can be reduced substantially. It is clear, on the other hand, that the view-dependent volume representation has to be recomputed in every frame.

To render the SPH data efficiently, we store the perspective grid in a 3D texture on the GPU and use texture-based volume ray-casting. To correctly resample the particle quantities into this texture, we derive the transformation that maps a Cartesian grid onto the corresponding perspective grid and consider this transformation in the resampling step. Resampling is entirely performed on the GPU to exploit memory bandwidth and computation parallelism. In the Cartesian grid, perspectively correct gradients are computed to simulate shading and lighting effects.

Since our approach reconstructs the continuous scalar field from the discrete particle samples and renders this field via ray-casting, high quality visualizations using arbitrary transfer functions can be generated. Especially compared to particle splatting, which projects each particle's reconstruction function separately to form a 2D pre-shaded footprint, vastly different image quality can be achieved. Figure 2 demonstrates this effect for a SPH data set in which the reconstruction functions from different particles overlap. The particle extents in this data set differ about a factor of 1400. While splatting (left) can only render a coarse approximation to this data, volume ray-casting (middle) generates a smooth continuous image of the discrete particle set and can reveal fine details of surface structures via additional iso-surfaces (right).

2 Related Work

Particle-based simulation techniques like SPH have been studied extensively over the last years. There is a vast body of literature related to this field and a comprehensive review is beyond the scope of this paper. However, Monaghan [23], Müller et al. [24], and Adams et al. [2] discuss the basic principles underlying such techniques and provide many useful algorithmic and implementation specific details.

Only very little work has been reported on the efficient rendering of SPH data. Most commonly this task is reduced to rendering iso-surfaces in the continuous 3D density field derived from the discrete set of particles. This can be accomplished by resampling particle quantities into a proxy grid [7, 26], and by using iso-surface extraction [22] or direct volume ray-casting [9]. Zhu and Bridson use a correction term for the density before the reconstruction, leading to smoother surfaces [38]. An efficient GPU technique for resampling particle quantities into 3D uniform grids has been presented recently in [37]. For iso-surface rendering, resampling can be restricted to the surface boundaries [36]. If a uniform proxy grid is used, high resolution is required to capture all details, but GPU-based volume rendering approaches can be employed to achieve high speed [10, 21].

In order to avoid the exhaustive memory consumption of a proxy grid, iso-surfaces can be rendered directly, for example, by evaluating ray-particle intersections on the GPU [20, 37]. Gribble et al. performed direct ray-sphere intersections on the CPU using a grid-based acceleration structure [14]. An iso-surface extraction technique that works directly on the particle set was introduced by [29]. Rosenthal et al. generate a surfel representation of the iso-surface from the discrete particle set [31]. A high-quality yet time-consuming surface-fitting approach using level-sets was presented in [30].

Order-independent splatting of transparent particle sprites was presented by [12, 18, 19] for volumetric astrophysical SPH data. Visualization systems for SPH data, including rendering options like splatting or slicing and providing application specific mechanisms to interact with and analyze particle data were discussed in [5, 6, 11, 27, 34].

Recently, screen-space approaches for rendering iso-surfaces in SPH data have gained much attention due to their efficiency. Adams et al. render particles as spheres and blend the contributions in the overlap regions [1]. Müller et al. reconstruct a triangle mesh in screen space from visible surface fragments that are generated via rasterization [25]. The method was improved by [16, 33] to generate smooth surfaces and to avoid an explicit triangulation.

3 Perspective Grid

The perspective grid is used to resample the particle data inside the view volume. It is a structured grid that partitions the view frustum into $k \times l \times m$ oblique subfrusta. Figure 3(right) illustrates this grid, which has the specific property that the sampling rate along z is the same as along x and y.

The perspective grid is stored on the GPU in a 3D texture map (see Figure 3(left)). In the resampling process, the 3D texture map is used as a render target and resampled quantities are scattered to this target using accumulative blend.

To perform the resampling efficiently on the GPU, a mapping of a point $\mathbf{r}' = (x', y', z')$ from 3D texture space to a point $\mathbf{r} = (x, y, z)$ in view space and vice versa is required. In the following, we will derive this mapping and describe how to exploit it in the resampling process.

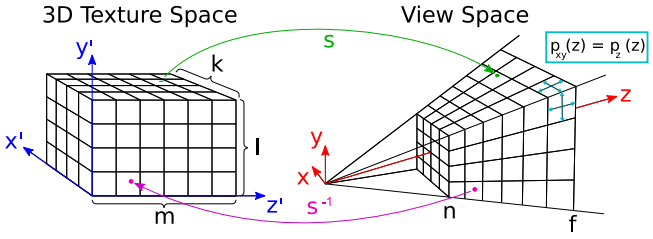

Fig. 3. The Cartesian grid in texture space (left) transforms onto the perspective grid in view space (right) under the mapping s. The grid has the same sampling rate along x and y ($p_{xy}(z)$) as along z ($p_z(z)$). We assume that the viewer is looking along the positive z-axis.

Perspective Grid Mapping Let us denote by $s(\mathbf{r}') = \left(s_x(x',z'), s_y(y',z'), s_z(z')\right)$ the mapping we are looking for, and let us note that the sampling rate $p_{xy}(z)$ along x and y is given by $(2 \cdot \tan(fov_y/2)/res_y) \cdot z$, where fov_y is the vertical field of view and res_y the vertical resolution of the viewport. The mapping of x' and y' in texture coordinates to x and y in view space is simply the inverse view-space to screen-space transformation:

$$s_x(x',z') = \left(\frac{2 \cdot x'}{k} - 1\right) \cdot s_z(z') \cdot \tan(fov_y/2)$$

and for $s_y(y',z')$, respectively. The mapping s_z of $z' \in [0,m]$ in texture space to $z \in [n,f]$ in view space (n and f are the near and far plane) needs some further explanations.

We observe that the sampling rate along z, $p_z(z)$, is equal to $\frac{\mathrm{d}z}{\mathrm{d}z'} = \frac{\mathrm{d}s_z(z')}{\mathrm{d}z'}$, since z' is sampled uniformly with distance 1. Since $p_z(z)$ is supposed to be equal to $p_{xy}(z)$, we obtain:

$$p_z(z) = \sigma \cdot s_z(z') = \frac{\mathrm{d}s_z(z')}{\mathrm{d}z'}, \tag{1}$$

where $\sigma = \frac{2 \cdot \tan(fov_y/2)}{res_y}$ and z has been replaced by $s_z(z')$ in the second term. This differential equation is solved by any $s_z(z')$ of the form

$$s_z(z') = a \cdot b^{c \cdot z'/m}.$$

Since $z' \in [0,m]$ is mapped to $z \in [n,f]$, and by setting $c = 1$, $s_z(z')$ equates to

$$s_z(z') = n \cdot \left(\frac{f}{n}\right)^{z'/m}. \tag{2}$$

It's inverse is

$$s_z^{-1}(z) = m \cdot \frac{\ln(z/n)}{\ln(f/n)}.$$

Note that this corresponds to the transformation derived in [32, 35] to equally distribute the aliasing error in perspective shadow map parameterizations. This mapping is only correct, however, on the z axis. On every other view ray the sampling rate is larger, since all of them perform the same z sampling in world space. In the extreme case, which is along the edges of the frustum, the sampling rate scales by the factor

$$\lambda = \sqrt{\left(\frac{k \cdot \sigma}{2}\right)^2 + \left(\frac{l \cdot \sigma}{2}\right)^2 + 1}.$$

We must further determine the number of sample points m along the z-axis for a given range $[n...f]$. By inserting equation 2 into 1 and solving for m, we obtain

$$m = \frac{\ln \frac{f}{n}}{\sigma}.$$

Finally, to account for the increasing sampling distance towards the frustum boundaries, and thus to provide the appropriate sampling in the whole frustum, m must be scaled by λ.

4 Data Resampling

To resample the particle data to the perspective grid, for every particle the grid vertices within the support of the particle's smoothing kernel have to be determined and the data is interpolated according to the kernel function:

$$A(\mathbf{r}) = m_j \cdot \frac{A_j}{\rho_j} \cdot W(|\mathbf{r} - \mathbf{r}_j|, h_j).$$

Here, $A(\mathbf{r})$ is the resampled data at position $\mathbf{r}$, and m_j, ρ_j, $\mathbf{r}_j$, and A_j are the particle's mass, density, position, and data value, respectively. W is the kernel function with a support h_j that can vary from particle to particle. In Section 6 we describe the particular kernel functions underlying the SPH simulations used in this work. The interpolated value $A(\mathbf{r})$ is added to the corresponding texel in the 3D texture map.

Key to an efficient and high quality resampling of large particle sets is the use of a multi-resolution particle representation [12, 18, 19]. In such a representation the particle set is encoded at different levels of detail by merging subsets consisting of smaller particles into one larger particle. The hierarchical particle representation allows pruning particles that are too small to be reconstructed at the required sampling rate. Thus, aliasing artefacts can be avoided and the number of particles to be processed can be reduced.

4.1 Hierarchical Particle Representation

Our pre-computed multi-resolution particle representation is organized in an adaptive octree data structure similar to the one proposed in [12]. In particular, for large particle sets and high spatial resolution of the simulation we employ the same regular domain partition to allow for the construction of the particle hierarchy in parts. In addition, for the Millenium gas dynamics simulation we scale each particle component logarithmically and compress the data using vector quantization. Spherical pre-fetching regions are realized on the GPU and the CPU to exploit frame-to-frame coherence and thus reduce memory access latencies.

In contrast, however, we use different rules for merging particles. These rules build upon the resampling operators presented in [8, 17] for adaptive SPH simulations.

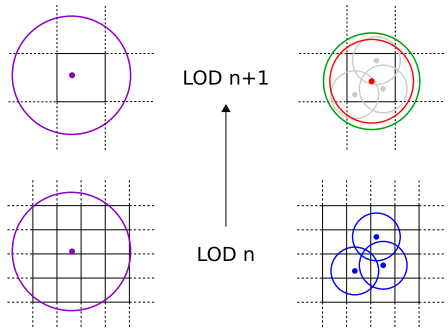

Fig. 4. Bottom-up construction of the particle LOD-hierarchy. Particles are copied to the next coarser level until their diameter falls below the grid sampling (left). Particles falling below the grid sampling are merged and eventually enlarged to the grid spacing (right).

Starting with a uniform grid at the resolution at which the simulation has been performed, particles in contiguous blocks of 2^3 cells are merged into a single particle. The volume of this particle is the sum of the volumes of the merged ones, and the scalar quantities of the merged particles are averaged into this particle according to their mass contribution. The merging process is illustrated in Figure 4. This process is recursively repeated until a user-given resolution level is reached.

In the merging process, at a particular level only particles with a radius less than the cell size on the next coarser level are merged. If the radius of a new particle falls below this size, it is increased to this size and its density is decreased proportionally to reflect the volume increase. All other particles are copied to the coarser level to guarantee a consistent representation of the particle field at all levels of detail.

4.2 Hierarchy Traversal

In every frame, of all particles those to be resampled at the current view have to be determined. This is done by traversing the particle hierarchy top-down on the CPU and pruning those nodes which do not overlap the view-frustum. For the remaining nodes, depending on the shortest distance of the node to the viewer the highest sampling rate of the particles in this node is computed. The traversal is stopped once the minimum size of the particles falls below this sampling rate, i.e., when sampling the particles into the grid would result in aliasing. The particles are then packed into vertex and associated attribute arrays, and they are sent to the GPU for resampling.

4.3 GPU Particle Slicing

GPU voxelization of particles into a 3D texture map work similar to the approach presented in [37]. For each particle a single vertex—positioned at the particle center and attributed by the particle quantity to be resampled—is sent to the GPU and passed to the geometry shader. The geometry shader computes the first (s_{min}) and the last (s_{max}) 2D texture slice that is covered by the particle as

$$s_{min} = \lceil s_z^{-1}(z_{center} - h) - 0.5 \rceil,$$
$$s_{max} = \lfloor s_z^{-1}(z_{center} + h) - 0.5 \rfloor.$$

Here, h is the particle's smoothing length, which can vary from particle to particle.

The shader spawns $s_{max} - s_{min} + 1$ equilateral triangles from this vertex, one for each slice $i, i \in \{s_{min}, ..., s_{max}\}$. The triangles are centered at the particle position and oriented orthogonal to the viewing direction. For the i-th triangle, it's depth $i + 0.5$ in texture space is transformed to view space as $z_i = s_z(i + 0.5)$.

Before rendering a triangle, the shader computes the radius of the circular cross-section between the sphere of radius h centered at the particle position and the slice z_i in view space, and it uses this value to resize the triangle so that it just covers the cross-section. Figure 5 illustrates this process. For every triangle vertex the distance vector to the particle center is determined and assigned as vertex attribute. Finally, the triangle is rendered into slice i of the 3D texture map using the standard perspective projection.

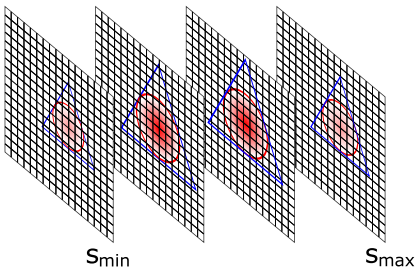

Fig. 5. Particle resampling on the GPU. As many proxy triangles as there are slices overlapped by a particle are rendered. Triangles are scaled to cover the particle-slice cross-sections.

During triangle rasterization the distance vectors are interpolated and its length is used in a pixel shader to test whether the corresponding texel is inside or outside the particle's kernel support. For fragments that are inside, the kernel function is evaluated using the interpolated distance vector and the smoothing length h, and the computed quantity is blended into the render target. Otherwise, the fragment is discarded.

5 Rendering

Once the particle quantities have been resampled to the perspective grid, the data can be rendered in turn on the GPU using texture-based volume ray-casting. This enables using different rendering options like iso-surface rendering or direct volume rendering simultaneously at very high speed. The major difference to classical texture-based ray-casting is in the kind of grid that is rendered. Usually, the data is given on a Cartesian grid in world space and has to be interpolated tri-linearly at the sample points along the rays. The data in the perspective grid, on the other hand, is already at the positions in view space where a sample is placed during ray-casting. Consequently, the data values in the 3D texture map that stores the perspective grid can be accumulated directly in front-to-back order along the z' coordinate axis in texture space. To account for the varying sampling distances, opacity correction of the samples has to be performed.

For simulating local illumination effects we compute the gradient of the resampled particle quantity. A gradient's x and y components can be approximated directly via central differences along x' and y', respectively. However, due to the perspective distortion of the resampling grid, an offset along the z'-coordinate axis in texture space does not correspond to an offset along the z-coordinate axis in view space. Thus, the texel center, (x'_t, y'_t, z'_t), is first transformed to view space, (x_t, y_t, z_t), and the positions of the two points $(x_t, y_t, z_t + \triangle)$ and $(x_t, y_t, z_t - \triangle)$ are transformed back to texture space. Here, $\triangle$ denotes the distance between the current and the previous slice in view space.

Hence the gradient of the sampled field f at a texel position (x'_t, y'_t, z'_t) in texture space is computed as

$$\nabla f = \begin{pmatrix} f(x'_t + 1, y'_t, z'_t) - f(x'_t - 1, y'_t, z'_t) \\ f(x'_t, y'_t + 1, z'_t) - f(x'_t, y'_t - 1, z'_t) \\ f\left(s_z\left(s_z^{-1}(x'_t, y'_t, z'_t) + \triangle\right)\right) - f\left(s_z\left(s_z^{-1}(x'_t, y'_t, z'_t) - \triangle\right)\right) \end{pmatrix}$$

As can be seen, computing a gradient's z component requires interpolating in texture space. 3D texture mapping hardware on the GPU supports tri-linear interpolation, which is not exact in our scenario due to the frustum-shaped cells underlying the texture grid. In order to improve the interpolation accuracy we have implemented distance-based interpolation in texture space. Figure 6, however, indicates the differences in the resulting illumination to be rather low. We thus use hardware-supported tri-linear interpolation throughout this work.

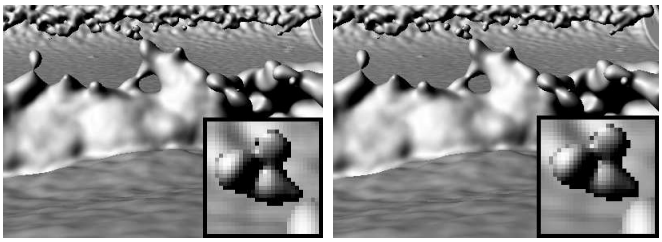

Fig. 6. Gradient computation in the perspective grid. Left: Trilinear interpolation; Right: Distance-based interpolation.

5.1 Grid Partitioning

Due to the limited amount of graphics memory on the GPU, it is not possible in general to store the entire perspective grid on the GPU. For this reason we partition the perspective grid into a number of view-aligned *slabs* of size $k \times l \times m_s$, each of which is small enough to fit into the graphics memory (see Figure 7). In front-to-back order these slabs are processed as described.

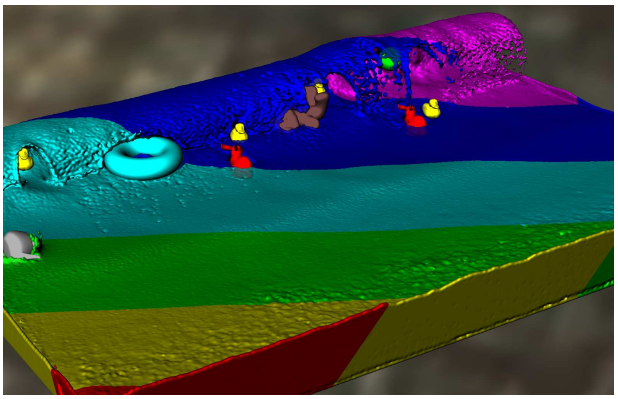

Fig. 7. If the perspective grid does not fit into GPU memory, it is partitioned into view-aligned slabs (illustrated by different colors) which are resampled and rendered in front-to-back order.

In order to reduce the overall workload on the GPU, for every slab only the particles contributing to this slab are resampled into the respective sub-grid. This is accomplished during the traversal of the

spatial hierarchy by computing for every visible node the slabs it overlaps and rendering for every slab only the overlapping nodes. In the computation we consider an overlap between slabs as well as at the slab boundaries to allow for a consistent gradient computation.

5.2 Occlusion culling

To minimize the number of rendered particles on the GPU, we employ a hierarchical opacity-buffer similar to the hierarchical depth-buffer proposed in [13]. After each n-th node we store the accumulated opacity in a separate texture. This texture is converted into a boolean mipmap to encode ever larger areas that are completely opaque. At ever coarser mipmap levels a texel is set to 1 (opaque) only if all covered texels in the previous finer level are 1. By employing bilinear texture interpolation, 4 child texels can be tested with one texture lookup. The algorithm takes care of Non-Power-of-Two textures as described in [15].

The mipmap is used to discard all nodes, and further on all particles, which are covered completely by opaque structures in front of them. This is done by constructing a screen space bounding circle for each node and particle. We choose to sample the first mipmap level in which the circle radius is below half the texel width. By sampling at the circle center using bilinear interpolation, all 4 texels that are possibly covered by the circle can be analyzed at once. If the bounding circle covers only mipmap texels marked by 1, the object can be safely discarded. Occlusion culling is applied first to the visible octree nodes, and it is then tested on a per-particle basis to avoid resampling of occluded particles. During rendering it is used to avoid ray traversal in occluded areas.

6 Results and Discussion

To demonstrate the efficiency and quality of our approach we render SPH simulation data from fluid dynamics and astrophysics (see Figure 11). Table 1 gives specific information on these data sets. Particle positions and quantities are encoded in 32 bit and 16 bit floating point values, respectively.

Table 1. The data sets used in our experiments.

Data set	Time steps	# Particles	Quantities
Flume	2209	$110-83,275$	Density
LWMO	604	$2,575,500$	Density
LWSB	1232	$3,232,000$	Density
WDMerger	84	$2,000,000$	Density + Temp.
SNIaEjecta	99	$8,745,571$	Density + Temp.
Millennium Run	1	$42,081,574$	Density + Vel. Disp.

The first three data sets simulate fluid dynamics. WDMerger simulates merging of two white dwarf stars. SNIaEjecta simulates the impact of a supernova ejecta on a companion star. The Millenium data set contains one time step of a simulation of the evolution of the universe. The *poly6*-kernel was used in all fluid dynamics simulations with constant h for each dataset:

$$W_{poly6}(r,h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3, & 0 \le r \le h \\ 0, & \text{otherwise} \end{cases}$$

In the astrophysical simulations the *cubic spline*-kernel was used for particles with varying smoothing length:

$$W(r,h) = \frac{8}{\pi h^3} \begin{cases} 1 - 6\left(\frac{r}{h}\right)^2 + 6\left(\frac{r}{h}\right)^2, & 0 \le \frac{r}{h} \le \frac{1}{2} \\ 2\left(1 - \frac{r}{h}\right)^3, & \frac{1}{2} \le \frac{r}{h} \le 1 \\ 0, & \frac{r}{h} > 1 \end{cases}$$

Table 2 shows the times required for resampling and rendering the data sets. Timings were performed on a 2.4 GHz Core 2 Duo processor and an NVIDIA GTX 280 graphics card with 1024 MB local video memory. The viewing parameters were selected so that the entire set of particles is within the view volume. Successive time steps of time-varying particle sets are streamed consecutively to the GPU. Upload times are not included in the given timings. For the Millennium Run with a total number of 10 billion particles, the given timings and the number of rendered particles in Tables 1 and 2 are averages over a continuous flight through the data.

As can be seen, resampling induces a high workload on the GPU and thus vastly dominates the overall performance. Especially for rendering the astrophysics data, which is represented by many particles with very large smoothing kernels, resampling causes a significant performance bottleneck. 3D texture-based volume ray-casting, on the other hand, only contributes marginally—below 10%—to the overall time.

Table 2. Performance statistics for resampling and rendering in milliseconds. The viewport resolution is 512^2, with a corresponding resolution of the perspective grid. Timings for a 1024^2 viewport and corresponding grid resolution are given in brackets.

Data set	Grid Resolution	Resampling	Rendering	Total
Flume	$512^2 \times 364$	6 (38)	43 (76)	49 (114)
LWMO	$512^2 \times 544$	87 (549)	65 (127)	152 (676)
LWSB	$512^2 \times 664$	135 (804)	79 (178)	214 (982)
Merger	$512^2 \times 292$	642 (1492)	40 (81)	682 (1573)
Ejecta	$512^2 \times 396$	1560 (4574)	49 (100)	1609 (4674)
Millen.	$512^2 \times 480$	2267 (7771)	83 (162)	2350 (7933)

The quality of our multi-resolution hierarchy is illustrated in Figure 8. It shows the LWSB and the Ejecta data set resampled to grids of different resolutions with a corresponding node selection from the LOD hierarchy. The particle hierarchy preserves the basic structures in the data, at the same time providing an effective anti-aliasing structure. Compared to the first image the number of rendered particles decreases from 3.2M over 2.1M and 0.7M down to 0.3M particles for the LWSB data set, and from 8.7M over 4.6M and 3.3M down to 1.4M particles for the Ejecta data set. On a 1024^2 viewport the total rendering time decreases by a factor of 4.2/10/27 and 2.9/8.2/19, respectively. This also demonstrates that interpolation between grid samples can be used to decouple the resolutions for resampling and raycasting, which allows to find a good trade-off between image quality and rendering speed. Apart from the achieved performance gain, a LOD representation is mandatory for the rendering of very large data sets. For example, the image of the Millennium Run in Figure 11 would require 607 million particles instead of 48 million particles to be rendered without such a hierarchy.

The performance gain due to occlusion culling depends on the depth complexity and the opacity of the rendered data. The effect becomes apparent if many particles are completely occluded. For example, in the scene shown in Figure 6 occlusion culling leads to performance gain of about 68%. In Figure 8 (upper row) an increase of only 5% could be observed due to low depth complexity. In Figure 10 (first row, second image) the gain was only 17% due of high transparency. In extreme close-ups where most particles are occluded, however, we measured an overall performance gain of up to factor 8.

Finally, we have carried out a performance comparison between our approach and order-dependent splatting. Particle sorting was performed on the GPU using an optimized sort routine, which required less than 5% of the overall rendering time. Rendering the Merger data set (Figure 2(left)) was performed at 7.85 fps. Using our techniques, the frame rate dropped to 1.7 fps (Figure 2(right)).

In a second experiment we compared our approach to screen-space methods for the visualization of iso-surfaces in SPH data. Such methods first render particles as spheres in an arbitrary order to obtain the visible surface parts and then generate a smooth surface from the resulting depth buffer imprint. Figure 9(left) shows the resulting image after the first pass for the LWMO data set. This image was generated at 12.5 fps. Figure 9(right) demonstrates an iso-surface reconstructed by our approach. This image was rendered at 5.75 fps, and, thus, only requires about twice the time of screen-space methods. Given that our method provides high-quality iso-surface and volume rendering, this seems to be a reasonable compromise.

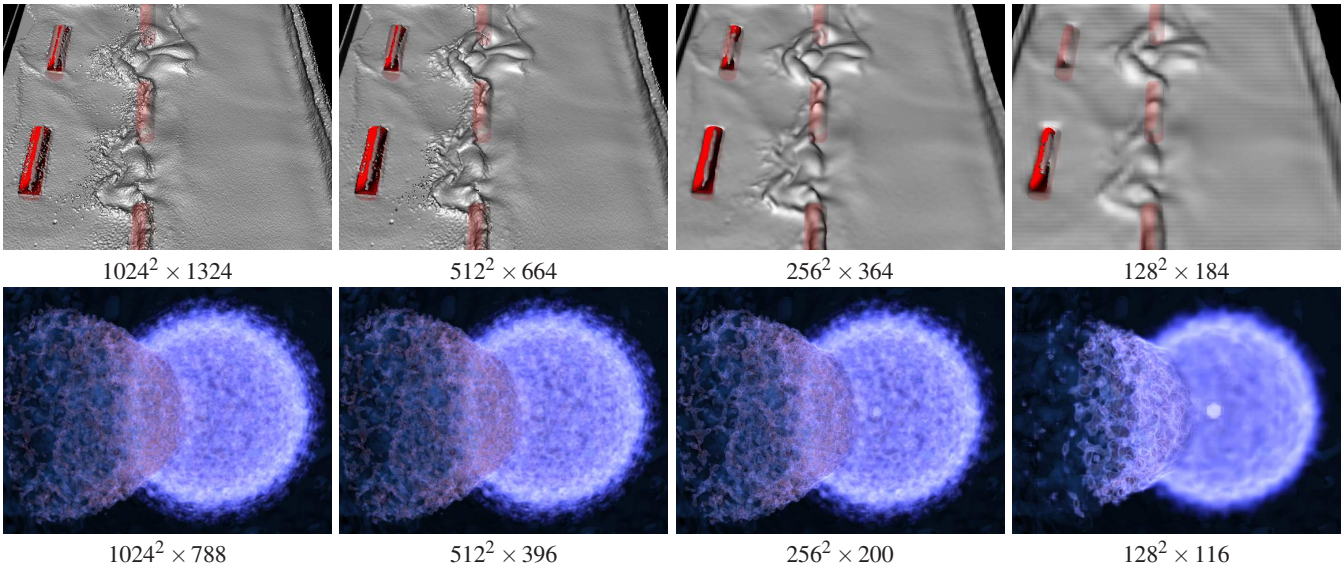

Fig. 8. Resampling to perspective grids of different resolutions with corresponding level of detail. Iso-Surface rendering of the LWSB data set (top row) and volume rendering of the ejecta data set (bottom row) at different grid resolutions.

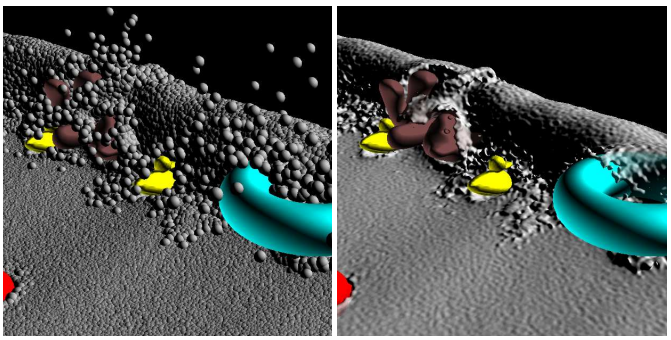

Fig. 9. Sphere rendering of the particle's support radius versus perspective grid-based iso-surface extraction. Screen-space techniques smooth the surface given by the depth buffer imprint of the left image, while our approach on the right directly reconstructs any given iso-surface.

7 Conclusion and Future Work

By eliminating the need for a uniform discretization of the fluid domain, we have overcome an essential limitation in the rendering of SPH data. We have introduced the perspective grid as an adaptive high-resolution discretization of the view volume, and we have shown how to efficiently resample particle quantities to this grid by using multi-resolution hierarchies for particle sets and by exploiting the GPU. Since the perspective grid places sample points along the view rays, resampled quantities can be rendered directly using 3D texture mapping. This enables integrating volumetric effects into SPH rendering, like volumetric emission and absorption, and using direct volume rendering and iso-surface rendering in combination.

To the best of our knowledge, for the first time we have shown that order-dependent resampling of high resolution SPH data can be performed at almost interactive rates. Compared to screen-space approaches for the rendering of iso-surfaces in SPH data, our technique is only slightly slower but achieves higher quality.

The proposed view-dependent discretization also has its limitation. Since resampling the fluid domain is restricted to the view volume, only parts of the fluid within this volume can be considered in the simulation of secondary effects like reflections or refractions. Especially in animations this can result in shimmering due to frame-to-frame incoherence.

In the future we will pursue research on filtering techniques for data given on a perspective grid. Since even with some improvements the surfaces in SPH data tend to look bumpy and can not adequately resolve flat structures, curvature-based smoothing as proposed in [33] will be considered on the grid.

Acknowledgments

The authors wish to thank Rüdiger Pakmor, Fritz Röpke, Volker Springel, and Gerard Lemson from the Max-Planck-Institute for Astrophysics for providing the gas dynamics data sets. We also thank Matthias Teschner for providing the fluid dynamic data sets and Markus Rampp (RZG) for his support. This work was funded in part by the Munich Centre of Advanced Computing at the Technische Universität München.

References

[1] B. Adams, T. Lenaerts, and P. Dutre. Particle splatting: Interactive rendering of particle-based simulation data. Technical report cw 453, Katholieke Universiteit Leuven, 2006.

[2] B. Adams, M. Pauly, R. Keiser, and L. J. Guibas. Adaptively sampled particle fluids. *ACM Trans. Graph.*, 26(3):48, 2007.

[3] G. Altay, R. A. C. Croft, and I. Pelupessy. SPHRAY: a smoothed particle hydrodynamics ray tracer for radiative transfer. *Monthly Notices of the Royal Astronomical Society*, 386:1931–1946, 2008.

[4] B. Arnaldi, T. Priol, and K. Bouatouch. A new space subdivision method for ray tracing CSG modelled scenes. *The Visual Computer*, 3(2):98–108, 1987.

[5] J. Biddiscombe, B. Geveci, K. Martin, K. Moreland, and D. Thompson. Time dependent processing in a parallel pipeline architecture. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1376–1383, 2007.

[6] J. Biddiscombe, D. Graham, and P. Maruzewski. Visualization and analysis of SPH data. *ERCOFTAC Bulletin*, 76:9–12, 2008.

[7] D. Cha, S. Son, and I. Ihm. GPU-assisted high quality particle rendering. *Computer Graphics Forum*, 28(4):1247 – 1255, 2009.

[8] M. Desbrun and M.-P. Cani. Space-time adaptive simulation of highly deformable substances. Technical Report 3829, INRIA, BP 105 - 78153 Le Chesnay Cedex - France, December 1999.

[9] R. A. Drebin, L. Carpenter, and P. Hanrahan. Volume rendering. In *SIGGRAPH '88: Proceedings of the 15th annual Conference on Computer Graphics and Interactive Techniques*, pages 65–74, 1988.

[10] C. Dyken, G. Ziegler, C. Theobalt, and H.-P. Seidel. High-speed marching cubes using histopyramids. *Computer Graphics Forum*, 27(8):2028–2039, 2008.

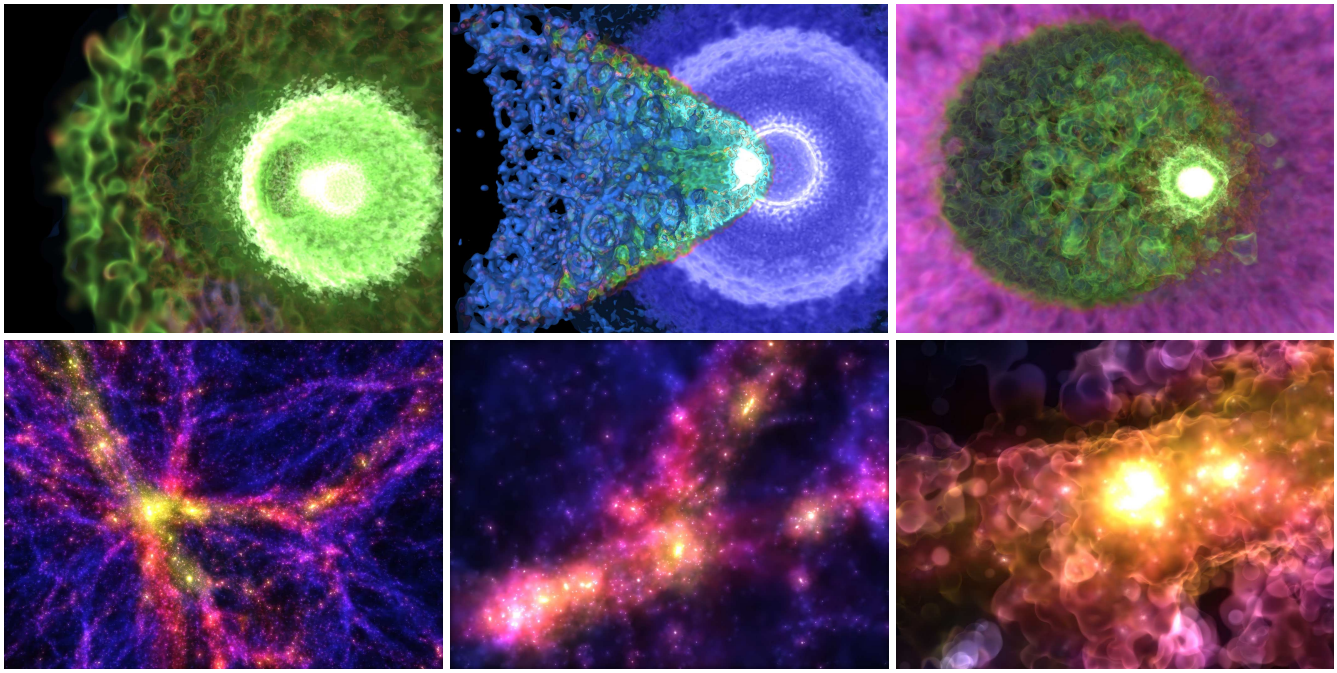

Fig. 10. Different transfer functions and camera settings reveal various details of the Ejecta (top row) and the Millennium data set (bottom row).

[11] D. Ellsworth, B. Green, and P. Moran. Interactive terascale particle visualization. In *VIS '04: Proceedings of the Conference on Visualization '04*, pages 353–360, Washington, DC, USA, 2004. IEEE Computer Society.

[12] R. Fraedrich, J. Schneider, and R. Westermann. Exploring the Millennium Run - scalable rendering of large-scale cosmological datasets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1251–1258, 2009.

[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*, pages 231–238, 1993.

[14] C. P. Gribble, T. Ize, A. Kensler, I. Wald, and S. G. Parker. A coherent grid traversal approach to visualizing particle-based simulation data. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):758–768, 2007.

[15] S. Guthe and P. Heckbert. Non-power-of-two mipmap creation. Technical Report TR-01838-001, NVIDIA Corporation, 2003.

[16] C. H. and S. O. Interactive screen-space surface rendering of dynamic particle clouds. *Journal of Graphics, GPU, and Game Tools*, 14(3):1–19, 2009.

[17] W. Hong, D. H. House, and J. Keyser. Adaptive particles for incompressible fluid simulation. *Vis. Comput.*, 24(7):535–543, 2008.

[18] M. Hopf and T. Ertl. Hierarchical splatting of scattered data. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 443–440, 2003.

[19] M. Hopf, M. Luttenberger, and T. Ertl. Hierarchical splatting of scattered 4D data. *IEEE Computer Graphics and Applications*, 24(4):64–72, 2004.

[20] Y. Kanamori, Z. Szego, and T. Nishita. GPU-based fast ray casting for a large number of metaballs. *Computer Graphics Forum*, 27(2):351–360, 2008.

[21] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 38–43, 2003.

[22] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987.

[23] J. J. Monaghan. Smoothed particle hydrodynamics. *Rep. Prog. Phys.*, 68:1703–1758, 2005.

[24] M. Müller, D. Charypar, and M. Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 154–159, 2003.

[25] M. Müller, S. Schirm, and S. Duthaler. Screen space meshes. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 9–15, 2007.

[26] P. A. Navrátil, J. L. Johnson, and V. Bromm. Visualization of cosmological particle-based datasets. In *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Visualization 2007)*, Nov/Dec 2007.

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

[28] M. Reinecke, D. Dolag, C. Gheller, and Z. Jin. Splotch. http://www.mpa-garching.mpg.de/ kdolag/Splotch, 2009. Raycasting SPH data.

[29] I. D. Rosenberg and K. Birdwell. Real-time particle isosurface extraction. In *I3D '08: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, pages 35–43, 2008.

[30] P. Rosenthal and L. Linsen. Smooth surface extraction from unstructured point-based volume data using PDEs. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1531–1546, 2008.

[31] P. Rosenthal, S. Rosswog, and L. Linsen. Direct surface extraction from smoothed particle hydrodynamics simulation data. In *Proceedings of the 4th High-End Visualization Workshop*, 2007.

[32] M. Stamminger and G. Drettakis. Perspective shadow maps. In *SIGGRAPH '02: Proceedings of the 29th annual Conference on Computer Graphics and Interactive Techniques*, pages 557–562, 2002.

[33] W. J. van der Laan, S. Green, and M. Sainz. Screen space fluid rendering with curvature flow. In *I3D '09: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, pages 91–98, 2009.

[34] R. Walker, P. Kenny, and J. Miao. Visualization of smoothed particle hydrodynamics for astrophysics. In L. Lever and M. McDerby, editors, *Theory and Practice of Computer Graphics 2005*, pages 133–138, University of Kent, UK, June 2005. Eurographics Association.

[35] M. Wimmer, D. Scherzer, and W. Purgathofer. Light space perspective shadow maps. In *Rendering Techniques 2004 (Proceedings Eurographics Symposium on Rendering)*, pages 143–151, 2004.

[36] R. Yasuda, T. Harada, and Y. Kawaguchi. Fast rendering of particle-based fluid by utilizing simulation data. In P. Alliez and M. Magnor, editors, *Proceedings of Eurographics 2009 - Short Papers*, pages 61–64, Munich, Germany, 2009. Eurographics Association.

[37] Y. Zhang, B. Solenthaler, and R. Pajarola. Adaptive sampling and rendering of fluids on the GPU. In *Symposium on Point-Based Graphics*, pages 137–146, 2008.

[38] Y. Zhu and R. Bridson. Animating sand as a fluid. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 965–972, 2005.

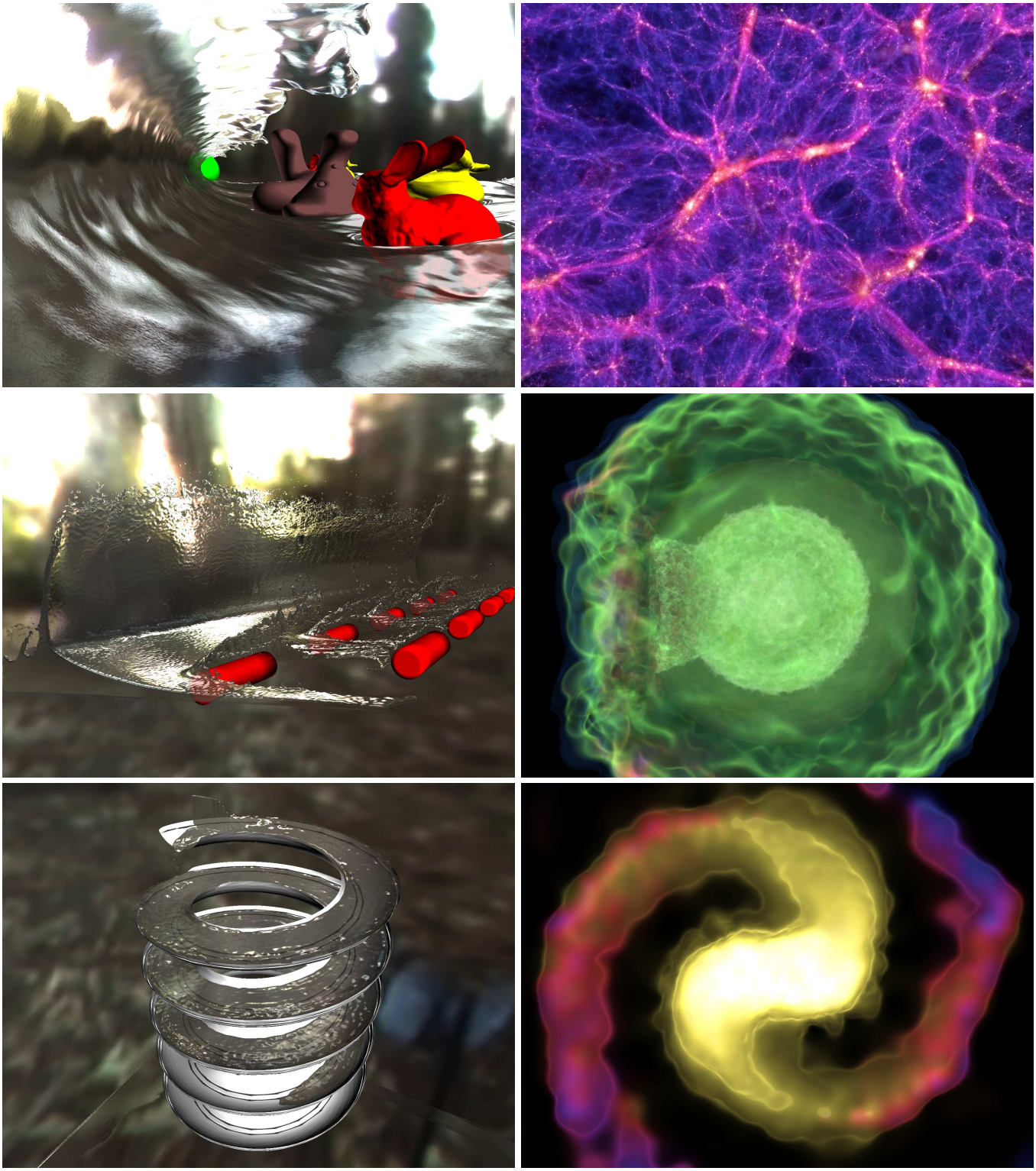

Fig. 11. The data sets we have used in our experiments (see Table 1 for more information). SPH fluid simulations (1st column, from top to bottom): Large Wave Moving Obstacles (LWMO), Large Wave Static Boundaries (LWSB), and Flume. Gas dynamics simulations (2nd column, from top to bottom): Millennium Run, SNIaEjecta, and WDMerger.