Interactive Image-Space Volume Visualization for Dynamic Particle Simulations

M. Falk and S. Grottel and T. Ertl

VISUS – Visualization Research Center, University of Stuttgart, Germany

Abstract

Particle-based simulation plays an important role in many different fields of science and engineering. Two common visualization approaches for the resulting data are glyph-based rendering and density sampling employing volume rendering. Fine geometric features are inherently captured by glyph-based methods. However, they might suffer from aliasing and the global structure is often poorly conveyed. Volume rendering preserves the global structure but is limited due to the sampling resolution. To avoid aliasing artifacts and large memory footprints, we propose a direct volume rendering technique with on-demand density sampling of the particle data, as combination of splatting, texture slicing, and ray casting. We optimized our system with a novel ray cast termination employing early-z-test culling and hardware occlusion queries utilizing inter-frame coherency.

Our system contains a fully-featured volume renderer and captures all geometric features of the data set representable at the available display resolution. Since no pre-computation is required, the proposed method can be used easily to visualize time-dependent data sets. The effectiveness of our approach is shown with examples from different application fields.

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

1. Introduction

There are many particle-based simulation methods from different fields of science, such as molecular dynamics, agent-based simulations, discrete element method, and smoothed particle hydrodynamics. These simulations usually comprise a large number of entities which interact with each other, either in a pairwise interaction scheme or by contributing to a common continuous field which in return affects all entities, which can be atoms, molecules, or mesoscopic or macroscopic particles. From a visualization point of view all these entities can be handled rather equally as particles.

Particle-based simulations usually generate large data sets due to their time-dependent nature. Visualizing such data sets interactively is still a challenge. On commodity workstations, the mere data set sizes require highly optimized software to achieve interactivity. Even the mapping from the individual particles to visual primitives might not be clear. Depending on the focus of the simulation different visualization methods are commonly used, from points or sprite-based particles, over density or probability volumes, to highly specialized visualizations like protein secondary structures. While point- or sprite-based rendering directly shows the original data, these images often suffer from high visual complexity due to visual clutter from a high number of discrete graphical primitives.

To gain a more compact representation, which is usually easier to understand, the particle data is sampled into a density volume and an iso-surface is rendered for a significant iso-value. This approach is also related to metaball rendering (also called blobby surfaces). Surface representations are common for protein data sets, surfaces or layers separating specific areas of the data sets in molecular dynamics, and iso-surfaces of electron position probabilities in quantum mechanics simulations. However, this technique has the drawback of additional memory requirements for storing the sampled volume data in addition to the particle data sets.

We therefore propose a volume rendering system with on-demand reconstruction of the volume data to minimize this

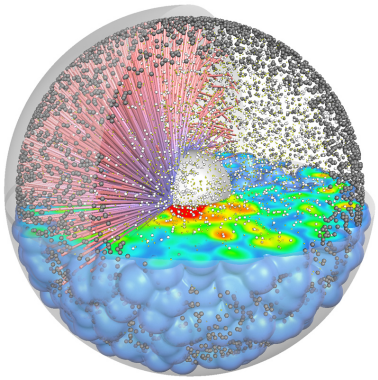


Figure 1: *Combining the volumetric visualization with the rendering of the simulated cell emphasizes the propagation of the signal. See Sec.4.1 for details.*

additional memory consumption. Our contribution is an optimized slice-based volume ray casting interleaved with an on-demand volume data reconstruction. Our method uses hardware occlusion queries to implement a novel termination method for the ray casting. We exploit the frame-to-frame coherency to compensate the additional overhead and latencies which these queries introduce. To show the effectiveness of our approach, we present examples from different fields of application, including propagation of the signaling front inside a cell in the context of systems biology, displaying relevant protein surfaces for biochemistry, as well as continuous material representations for data sets from molecular dynamics simulations of laser ablation processes. Compared to particle visualization and object-space volume ray casting our method results in superior image quality, due to the evaluation in image-space, has less memory requirements, since the volume data is only partially created on-demand, and still reaches comparative rendering performance.

2. Related work

Particle-based simulation Particle-based simulations have been used and studied for a very long time. The smoothed particle hydrodynamics introduced by Gingold and Monaghan [GM77] or molecular dynamics introduced by McCammon et al. [MGK77] are nowadays among the most popular simulation methods. In recent years the calculation power of graphics hardware was harvested for these techniques. Kipfer et al. [KSW04] presented a GPU-based system for particle simulations, which was, however, primarily intended for rendering and animation and not scientific simulation. A similar approach was presented by Kolb et al. [KLR04] in the same year and was extended to simulate coupled particles, i. e. particles reacting with each other and not only reacting to an underlying velocity field [KC05], allowing a Lagrangian particle simulation of fluids. Van Meel et al. [vMAF*07] used the modern general purpose GPU API CUDA for a fully functional molecular dynamics simulation on graphics hardware. A smoothed particle hydrodynamics simulation on the GPU was presented by Zhang et al. [ZSP08] to simulate and render liquids for computer graphics.

In contrast to such physically motivated simulations, which usually simulate a high number of particles with rather simple particle interactions (e. g. pair potentials), there are simulations following an orthogonal approach. E. g. in systems biology the movement of molecules and proteins inside a cell are simulated, where the internal structure of the cell – the cell cytoskeleton – is used to direct the movement of the proteins toward the core of the cell [KLR09], resulting in a simulation of less particles but with rather complex behavior (see Fig. 1).

Particle-based visualization Particle data sets are often directly visualized using glyphs to represent the individual particles. Gumhold [Gum03] presented an efficient way of GPU ray casting for glyphs with polynomial surfaces, ellipsoids in his work, to visualize symmetric tensor fields. A similar approach was used by Klein et al. [KE04] to visualize magnetic fields. Based on the same method Reina et al. [RE05] showed how to ray cast complex glyphs, constructed from quadratic primitives, like spheres and cylinders. This approach was extended by Grottel et al. [GRE09] to arbitrarily composed glyphs with optimized data transfer between CPU and GPU. Tarini et al. [TCM06] used methods from computer graphics, like ambient occlusion and silhouettes to enhance the depth perception. A recent publication by Grottel et al. [GRDE10] highly optimizes the rendering of particle glyphs employing hardware occlusion queries and hierarchical depth buffers, allowing interactive visualization up to 100,000,000 particles on commodity workstations.

For some fields of application, like proteins in biochemistry, specialized visual metaphors exist. Krone et al. [KBE08] showed how to generate the protein secondary structure representation on the graphics hardware. In a recent publication they used GPU ray casting of quadratic and quartic surfaces to render protein surfaces [KBE09]. Falk et al. [FKRE09] used GPU ray casting of spheres and point splats to visualize the movement of proteins inside a cell structure with different visual metaphors.

Metaballs rendering Compact representations of particle data sets, like surfaces or layers between different parts of the data, are most of the time the structures of interest. Such surfaces can be defined as iso-surfaces of volumetric data. Blinn proposed [Bli82] a metaball surface, an iso-surface in a density volume, defined by radial symmetric density kernel functions based at each particle. These metaballs can be rendered by ray casting or by extracting a geometric representation. Marching Cubes [LC87] is the most prominent algorithm to extract the geometry of such a surface.

On modern graphics cards the metaballs can be interactively evaluated and rendered. Kooten et al. [vKvdBT07] distribute points with repulsive forces on the surface and perform point-based rendering after the correct iso-surface is found. Sampling with too few points results in holes and

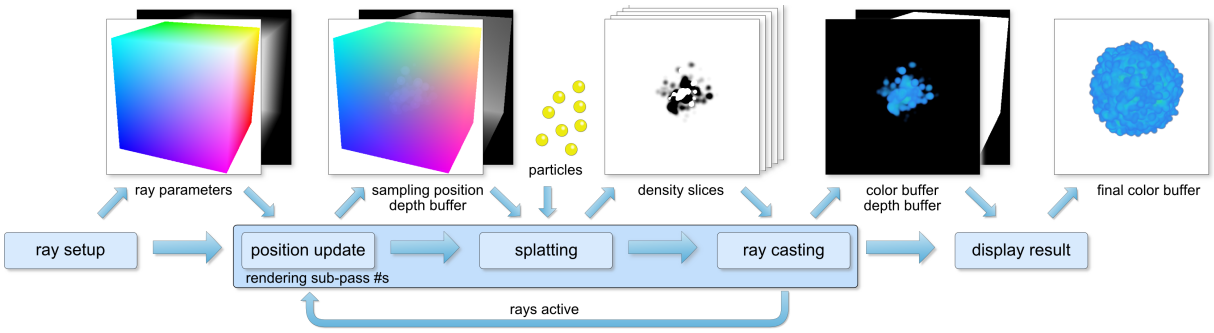


Figure 2: *Flow chart of our sliced ray casting method. In- and output buffers of the various stages are depicted above. After initial ray setup, multiple rendering sub-passes of sampling position update, density reconstruction through splatting and volume ray-casting are performed to generate the final image.*

visual artifacts. Müller et al. [MGE07] propose two possible ways of iterative ray casting metaballs. One approach – "walking depth plane" – uses multiple rendering passes and samples the density volume at specific depths in object space, which is related to our approach. The other approach pre-calculates a neighborhood list and evaluates the density from all neighboring particles in an iterative, single-pass raycasting. However, both methods only hardly achieve interactivity for medium-sized data sets. A method with a similar fundamental idea was presented by Kanamori et al. [KSN08], which however uses Bézier Clipping to find the ray-isosurface intersection. Linsen et al. [LvLRR08] extended this surface approach to multivariate data.

Volume rendering As the metaball surface is an isosurface of a density volume, another approach is the complete reconstruction of this density volume and to perform volume rendering. The system described by Cha et al. [CSI09] follows this approach. In biochemistry conformational changes of proteins can be visualized as flexibility volume in a similar way, as presented by Schmidt-Ehrenberg et al. [SEBH02].

Early implementations of volume rendering including shading, like presented by Westermann et al. [WE98] blend object-space-aligned texture stacks. View-aligned texture stacks provide higher visual quality but require 3D textures. Nowadays, volume ray casting is used, as presented e. g. by Drebin et al. [DCH88]. Krüger et al. [KW03] presented a view-aligned ray casting approach employing programmable graphics hardware. These multi-pass rendering approaches have been superseded by single-pass volume ray casting, e. g. presented by Stegmaier et al. [SSKE05].

Another approach for volume rendering is splatting, as proposed by Westover [Wes90]. This approach has also been optimized to harvest the computational power of modern GPUs, e. g. by Neophytou et al. [NM05], to perform correct image-space aligned splatting of rectangular voxels. Fraedrich et al. [FSW09] presented a system using this technique in a hierarchical fashion to interactively visualize a large astronomy data set.

3. Our Approach

Our method of sliced ray casting with on-demand volume reconstruction combines the ideas of texture slicing, ray casting, and splatting. The reconstruction of the density field from the particles is performed in screen space at viewport resolution and at the sampling depths used by the volume ray casting. The issue of perspective correction is thus moved from volume rendering to data reconstruction. A tight interleaving between the volume rendering and the data reconstruction allows to minimize the required memory footprint. A novel method for termination of the ray casting based on inter-frame occlusion queries is employed to optimize the overall performance of our system.

3.1. Overview

We propose to subdivide the viewing volume in viewing direction in concentric spherical shells centered at the eye **e**. The sampling positions on this shells coincide with the sampling positions of classical ray casting. These shells are equally separated by the volume sampling distance $\Delta\lambda$ and are used to reconstruct the density volume. This is done by perspectively correct splatting of the particles density footprints. The first slice is placed at λ_{min}, the minimal distance between **e** and bounding box of the particles to skip the empty space in front of the camera. This way λ is used as volume ray casting parameter and controls the density volume reconstruction, thus avoiding sampling artifacts. The method, as depicted in Fig. 2, consists of several calculation steps solely performed on the GPU. Three inner steps are performed in a loop for #*s* times and form the iterative ray casting. The whole process is controlled by the CPU.

Ray setup Ray direction **v**, maximum ray length λ_{max} based on the back-side of the particles bounding box, and the start position at the front face of the bounding box are stored

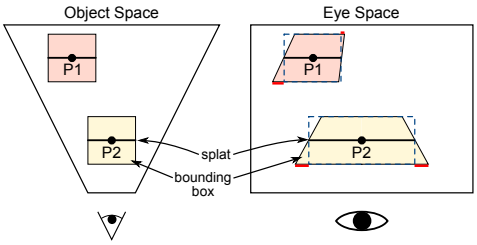


Figure 3: *Perspective correction is necessary when projecting splats from object space into image space. Otherwise the splats are not fully represented in image space.*

in buffers for each ray, which corresponds to a single image fragment. Additionally, rays might be flagged as initially inactive, if their starting positions lie outside the bounding box. Compositing our image with an opaque scene is possible by adjusting λ_{max} based on scenes depth buffer.

Position update In sub-pass number s we first update the sampling positions $\mathbf{p} = \mathbf{e} + \lambda\mathbf{v}$ with $\lambda = \lambda_{min} + s \times \Delta\lambda$. Initially inactive rays might now become active, if their new sampling position lies inside the bounding box. Rays which stepped behind the bounding box are terminated using the depth buffer. The depth test then will kill any fragments for inactive rays. After the update, a hardware occlusion query might be issued to feed-back the number of active rays to the CPU. See Sec. 3.4 for details.

Splatting To evaluate the density volume at the sampling positions $\mathbf{p}$ we splat perspectively correct footprints of the particle kernel functions into the buffer used for the current sub-pass s. Employing multiple render targets, the density is evaluated for several subsequent depths at once, reducing the required overall draw calls for the particles. Details are discussed in Sec. 3.2.

Ray casting We perform direct volume rendering using ray casting for the reconstructed density slices and composite the outcome with the result of previous sub-passes to gain the final image. Additionally, rays may be terminated by setting the depth buffer accordingly, if the accumulated opacity exceeds a threshold. See Sec. 3.3 for details.

Display result After these steps of our sub-passes are performed $s_{max}(f)$ times, a value determined by the occlusion queries of the previous frame $f-1$, the image of frame f can be displayed. The results of the occlusion queries can now be collected to determine $s_{max}(f+1)$ for the following frame $f+1$ (see Sec. 3.4).

3.2. Splatting

We follow the approach of metaballs as definition for the density field, with the density $\bar{\rho}(\mathbf{p})$ at position $\mathbf{p}$ computed from neighboring particles j by

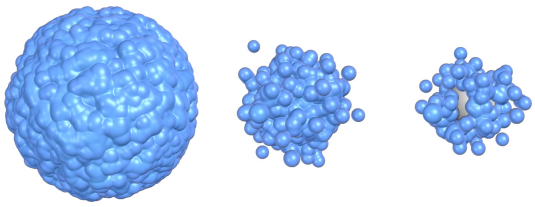
$$\bar{\rho}(\mathbf{p}) = \sum_j m_j W\left(|\mathbf{p} - \mathbf{p}_j|, h\right) , \qquad (1)$$

where $\mathbf{p}_j$ is the particle position and $W \in C^2[0,h]$ is a radial symmetric kernel function with support radius h. m_j is the particle's mass and is used to normalized the overall result to easier evaluate the iso-value (i. e. $\bar{\rho}(\mathbf{p}) = 1$). For each particle j we use a symmetric polynomial density kernel function (the smooth step function) with finite support radius h, but any other function with finite support could be used. Since we evaluate the density volume in image-space, the splatting of these kernel functions must be perspectively correct, as can be seen in Fig. 3. We employ the method of perspectively correct glyph raycasting, presented by Klein et al. [KE04]. The image space size of the point-sprite used to rasterize the splat is determined by h. The overall density $\bar{\rho}(\mathbf{p})$ is only a sum of the contributions of the individual particles $m_j W\left(|\mathbf{p} - \mathbf{p}_j|, h\right)$. Additive blending of these contribution yields the correct value. To compute several density slices at once, multiple render targets are used, with the first slice placed at λ and the remaining slices being $\Delta\lambda$ apart from each other.

Using slices with uniform depths the particles not contributing can be efficiently culled, by moving them outside the viewing frustum. This method would get far more complicated and computational expensive, if the density slices do not have uniform depth values, like in the approach of Müller et al. [MGE07], and the speed-up from culling is completely nullified by this computational load. The culling can be further optimized by sorting the particles along the viewing direction. Then the particles contributing to the current density slice are within an interval. We use a Radix Sort in CUDA to sort the indices of the particles according to their image space depth. Afterward two loops on the CPU can be used to get all indices of the first and last particles for all slices in $\mathcal{O}(n)$ with n particles. Alternatively, a parallel reduction of the indices can be performed on the GPU, requiring $\mathcal{O}(\log(n))$ for each slice, resulting in $\mathcal{O}(m\log(n))$ for m slices. Like Sec. 4.5 shows, sorting is only beneficial for large data sets.

3.3. Volume ray casting

Within a stack of density slices, generated using multiple render targets, we perform ray casting similar to single-pass volume rendering. For a continuous iso-surface, interpolation between the density slices is necessary, which is straightforward within a stack of density slices. To interpolate between stacks we reuse one slice of the previous sub-pass. For illuminating the iso-surface, gradients have to be computed. For a single particle the gradient is given by the vector from the position of the particle $\mathbf{p}_j$ to the sampling position $\mathbf{p}$, due to the radial symmetry of the kernel functions. The overall gradient is computed as summation similar to Eq. 1 by additive blending of the weighted gradients. The results of the sub-passes are composited yielding the final image. The accumulated opacity along an active ray is



$t = 5$ $t = 100$ $t = 140$

Figure 4: *Cellular signal transduction. Three time steps of the simulation are shown with the image-space approach. Signaling proteins are started at the cell membrane and then moved toward the nucleus.*

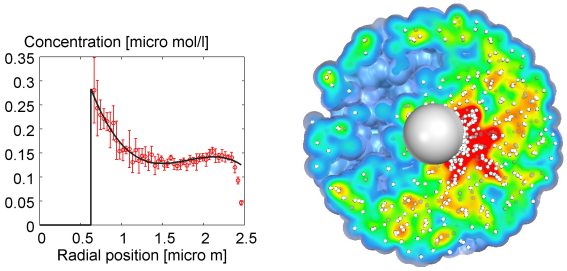


Figure 5: *Spatial effects are not covered by the radial concentration profile (left), but are visible with the volumetric visualization (right).*

also tested against a threshold to terminate the ray using the depth buffer.

3.4. Ray cast termination

Since the sub-passes of our approach are CPU controlled, but the states of the individual rays exist on the GPU, a feed-back is required. Using the depth buffer to mask inactive rays, a single hardware occlusion query can count the number of active rays using the depth test and deliver this number to the CPU. We use this information to control the number of sub-passes $s_{max}(f)$ for frame f. For the initial frame $f = 0$ the number is set to the maximum value required $s_{max}(0) = \max_{\forall \text{rays}} \frac{\lambda_{max} - \lambda_{min}}{\Delta\lambda}$ to sample the whole bounding box. Additionally, fragments for inactive rays are automatically rejected by the built-in early-z-test during position update and splatting.

The hardware occlusion queries however introduce additional computational overhead and latencies, as the results are not immediately available. To reduce the overhead, the number of occlusion queries is minimized. Six queries are issued during the last few sub-passes of the current frames, since we expect high frame-to-frame coherency. We issue them in the sub-passes $s_{max}(f) - \{50, 20, 5, 2, 1, 0\}$, what we found makes our system quite adaptive.

To hide the latencies, we exploit the frame-to-frame coherence and postpone the evaluation of the query result and the adjustment of the maximum number of sub-passes for the next frame $s_{max}(f+1)$ until the current frame f is finished and displayed. We collect the results of the queries in the order they have been issued and test the number of active rays against a threshold. The sub-pass of the first query with too few active rays will be set as $s_{max}(f+1)$. If all queries show too few active rays, the number of sub-passes is strongly reduced to $s_{max}(f+1) = s_{max}(f)/2$. If no query has few enough active rays, the number of sub-passes is increased by a fix value $s_{max}(f+1) = s_{max}(f) + 25$, limited by the overall maximum $s_{max}(f+1) \leq s_{max}(0)$.

4. Results

Our algorithm is implemented in C++ with OpenGL and GLSL shaders. For comparison with our image-space approach, we implemented the object-space method from Kolb et al. [KC05], where the density field is reconstructed as uniform grid and stored as 3D texture. The density field is subsequently visualized with standard ray casting. We conducted tests on a Windows PC with a Intel Core2 with 2.4 GHz, 2 GB RAM, and a NVIDIA GeForce 280 GTX with 1 GB. The viewport size was 512×512 and the resolution of the uniform grid was set to 256^3. The sampling distance $\Delta\lambda$ was set to $1/256$.

The ray parameters, sampling position, direction and length, are stored in floating point textures with 32 bit precision. Density values are stored in 16 bit floats. The ray parameter textures and the density slices have viewport resolution. For a 512×512 window we therefore need 8 MB for ray parameters and 12 MB for our image-based slicing with 8 slices compared to 256 MB for a full 512^3 volume. The particle positions are stored on the GPU in a vertex buffer and are updated once per frame. In the following, we show the effectiveness of our approach and discuss quality and performance results.

4.1. Signal transduction

The signal transduction process inside a cell is modeled with an agent-based Monte Carlo simulation [FKRE09]. The cell model contains signaling proteins, their reaction partners, the cytoskeleton, and the nucleus. The signal is initiated at the cell membrane and transported to the nucleus by signaling proteins.

The initial 25,000 signaling proteins move by diffuse and motorized transport toward the nucleus. Due to reactions along the signaling pathway, the number of active signal proteins decreases. Fig. 4 depicts three time steps visualized with our approach. Compared to the radial protein concentration profiles biologists use (Fig. 5 left), the visualization of the protein densities includes more details. Spatial effects, like a higher concentration on one side of the nucleus, are not visible in the concentration profile, but these effects are clearly visible when visualizing the density field (Fig. 5 right).

In Fig. 1, the volumetric visualization of signaling pro-

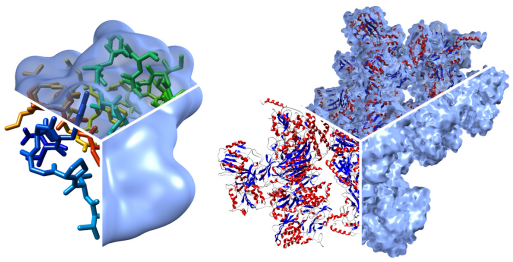


Figure 6: *Visualizations of two proteins. Left: crambin with ∼330 atoms (∼45 amino acids). Right: oxidoreductase with ∼75,000 atoms (∼10,000 amino acids). The lower images show proteins in stick representation and the proteins secondary structure, as well as the SES volume rendering. The upper images show combinations of both.*

teins is combined with the rendering of the proteins, the nucleus, and the cytoskeleton from the simulation. The protein concentration was mapped to colors from blue to red for low and high densities, respectively. The combination of both visualizations exposes the signal distribution clearly.

4.2. Protein surfaces

In biochemistry, the visualization of smooth molecular surfaces is of great importance for many applications such as docking or protein-solvent interaction. Solvent Excluded Surface (SES) and Molecular Skin Surface (MSS) are the most commonly used analytic models. However, they are computationally expensive, which makes them unfeasible for large dynamic data sets. Interactive rendering of large proteins must therefore often rely on approximations like metaballs. Our method can be used as such a fast approximation. The representation can easily be adjusted to resemble SES or MSS. The result is a smooth surface visually equivalent to ray casted MSS presented in [CLM08]. Fig. 6 shows a semitransparent SES approximation in combinations with different molecular models.

4.3. Laser ablation

The data sets shown in Fig. 7 and 8 result from molecular dynamic simulation which was coupled with finite element analysis. A pulsed laser beam heats up a solid metal block and a bulge is build.

The cutaway in Fig. 7, rendered at 1500×900, reveals the density distribution inside the material and contains roughly 250,000 particles. Glyph-based rendering of the particles (top left) hardly reveals any structures in static images.

As the bulge builds up, it breaks open at some point. Droplets of atoms are formed and expelled from the bulge leaving a crater in the metal. In Fig. 8, semi-transparent isosurfaces were visualized from the splash-like structures with

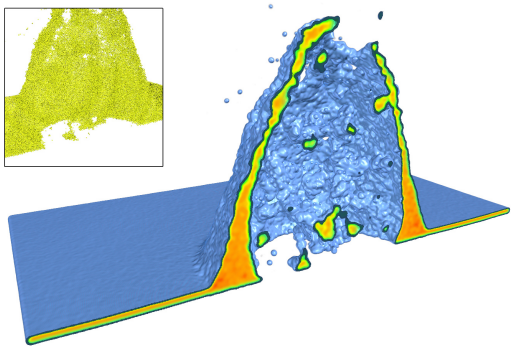


Figure 7: *Laser ablation leads to a bulge on a metal block. Particle rendering with glyphs is not sufficient to grasp the overall shape in still images (top left). Cutaway of the bulge, visualized with the image space approach at* 1500×900 *pixels, revealing the density distribution inside.*

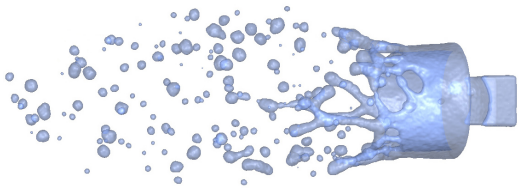


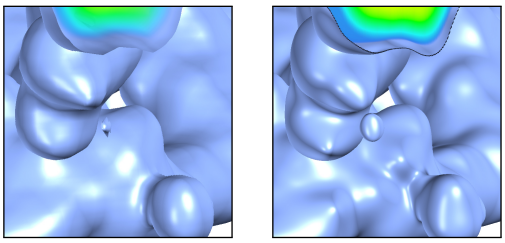
Figure 8: *When the bulge created by laser ablation breaks open, atoms are expelled and droplets form. The data set contains 742,141 particles and the image was rendered at* 1500×450 *at 0.7 fps with our image space method.*

our approach. The data set was rendered at a resolution of 1500×450 pixels and contains about 750,000 particles.

4.4. Qualitative results

In the object-space approach, problems can arise because the uniform grid has a finite resolution as illustrated in Fig. 9(a). Density splats smaller than a voxel cannot be stored completely leading to distorted or cube-like appearances of the original filter kernel. Sharp corners get smoothed out by trilinear interpolation of the uniform grid. With our approach all geometric features are conveyed in image space (Fig. 9(b)). Small features like the ellipsoid in the center of the close-up are fully captured with the image space method, whereas being deformed and barely visible in object space. The image-space technique also yields more distinct highlights. Hence, groves and ridges are depicted more clearly.

The hardware clip planes of OpenGL can be used to cut away parts of the volume. However, when particles are clipped at the intended cut planes, the final volume will show no sharp cuts because the kernel is evaluated for the complete particle footprints (Fig. 10, left). Additionally, the missing contribution of clipped particles leads to wrong results. Therefore, we perform the clipping in the fragment



(a) (b)

Figure 9: *Close-ups (800 × 800 pixels) of a small region of Fig. 7 are depicted: (a) object space, (b) image space. Ridges, grooves, highlights, and small features are more distinct in image space.*

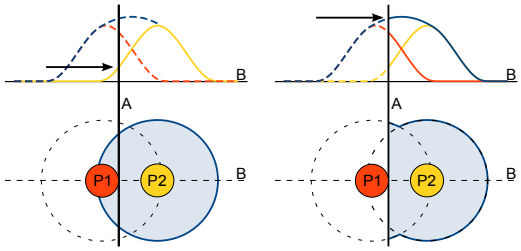


Figure 10: *If P1 is clipped, its density contribution is neglected leading to a wrong density (left). For correct volumetric clipping, P1 has to be considered (right). A denotes the clip plane, the density profiles are shown for the cross-sections B.*

program during density computation. The hardware clip planes are positioned in a way that only non-contributing particles are discarded. The density computed in Eq. 1 is set to zero if the sampling position **p** is outside the cut planes. The final result is a correct volumetric clipping as illustrated in Fig. 10, right.

4.5. Performance

A synthetic data set was used to demonstrate the scalability of our approach to time-dependent data sets with about 100,000 particles. Over 300 time steps, the number of particles increases linearly from zero. The performance was measured for this data set, examples of signal transduction (*Cell*), and two data sets from laser ablation simulation (*Bulge* and *Splat*). Table 1 shows the rendering performance for these data sets. The effect of sorting along the viewing direction becomes apparent for a large number of particles when the benefit outweighs the additional costs.

As time-dependent, real-world data set, we use the signal transduction data depicted in Fig. 4. In Fig. 11, the performance over the number of particles is shown as well as the number of particles over time. Our sliced ray casting in image space benefits from early-ray termination and outperforms the object space techniques when dealing with more than 6,000 particles. The time spent to rasterize the splats

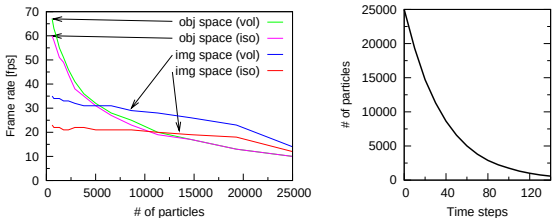


Figure 11: *Rendering performance of object-space and image-space methods for the signal transduction data set (left). Timings for direct volume rendering (vol) and isosurface rendering (iso) were measured. The number of particles is decreasing during the simulation (right).*

Data set	Particle number	Object space vol	Object space iso	Image space vol	Image space iso
Synthetic	12,276	26.51	34.00	12.58	8.57
(sorted)		23.40	–	12.74	–
Synthetic	101,664	7.28	7.71	3.23	2.62
(sorted)		9.57	–	3.98	–
Cell	25,000	10.90	10.75	14.62	12.28
	584	67.36	60.08	35.13	23.05
Bulge	509,423	–	1.16	–	0.83
(cutaway)	∼250,000	–	1.59	–	0.30
Splat	742,141	–	0.97	–	0.7

Table 1: *Rendering performance. Timings for direct volume rendering (vol) and isosurface rendering (iso) are measured in frames per second.*

corresponds with the number of proteins while the time needed to perform the multiple render passes keeps almost constant, which may explain the almost constant frame rate for low particle numbers. The rendering of the isosurface is affected by higher memory bandwidth. Instead of only one value for the density, now four values, have to be written and read for each density slice in our sliced ray casting. In object space, the bad performance might be due to the gradient computation which requires six additional texture lookups in the density texture.

5. Conclusion and future work

We present an interactive image-space ray casting technique with on-demand reconstruction of the volume data from point data, following the metaball approach. A novel method of terminating the ray casting based on hardware occlusion queries and frame-to-frame coherence is used to optimize the overall performance. The possibilities for visual cues and effects of volume rendering are superior to classical particle-based glyph rendering, and our approach does not significantly increase the memory requirements, compared to classical volume rendering. Since our technique does not require

any per-computation it is perfectly suited for time-dependent data. We achieve high image quality and good interactivity for large particle data sets. The effectiveness of our approach has been shown on several examples from simulations in systems biology, biochemistry, and physics.

We plan to further extend our algorithm and to integrate it into a visualization framework for particle data sets. Employing an hybrid image-space object-space subdivision we will implement empty-space-skipping as well as optimization of the ray setup. The reconstruction of the density volume could also be completely implemented with CUDA, however, preliminary experiments showed that the processing power of the OpenGL rasterization engine cannot be achieved by a CUDA-only implementation, due to the requirement of random-access to the memory of the density volume. However, we want to further investigate this approach, especially because of the cache optimizations of the upcoming Fermi graphics cards.

Acknowledgments

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) and the Collaborative Research Centre SFB 716 at the University of Stuttgart. The laser ablation data sets were kindly provided by Steffen Sonntag. The authors also want to thank Michael Krone for his rendering of protein data sets.

References

[Bli82] BLINN J. F.: A generalization of algebraic surface drawing. *ACM Transactions on Graphics 1*, 3 (1982), 235–256. 2

[CLM08] CHAVENT M., LEVY B., MAIGRET B.: MetaMol: High-quality visualization of molecular skin surface. *Journal of Molecular Graphics and Modelling 27*, 2 (2008), 209–216. 6

[CSI09] CHA D., SON S., IHM I.: GPU-assisted high quality particle rendering. In *Eurographics Symposium on Rendering* (2009), pp. 1247–1255. 3

[DCH88] DREBIN R. A., CARPENTER L., HANRAHAN P.: Volume rendering. In *ACM SIGGRAPH 1988* (1988), pp. 65–74. 3

[FKRE09] FALK M., KLANN M., REUSS M., ERTL T.: Visualization of signal transduction processes in the crowded environment of the cell. In *IEEE Pacific Visualization Symposium (PacificVis '09)* (2009), pp. 169–176. 2, 5

[FSW09] FRAEDRICH R., SCHNEIDER J., WESTERMANN R.: Exploring the millennium run - scalable rendering of large-scale cosmological datasets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1251–1258. 3

[GM77] GINGOLD R., MONAGHAN J.: Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly Notices Royal Astronomical Society 181* (1977), 375–389. 2

[GRDE10] GROTTEL S., REINA G., DACHSBACHER C., ERTL T.: Coherent culling and shading for large molecular dynamics visualization. In *Eurographics/IEEE Symposium on Visualization* (2010). to appear. 2

[GRE09] GROTTEL S., REINA G., ERTL T.: Optimized data transfer for time-dependent, gpu-based glyphs. In *IEEE Pacific Visualization Symposium (PacificVis '09)* (2009), pp. 65–72. 2

[Gum03] GUMHOLD S.: Splatting illuminated ellipsoids with depth correction. In *International Fall Workshop on Vision, Modelling and Visualization* (2003), pp. 245–252. 2

[KBE08] KRONE M., BIDMON K., ERTL T.: Gpu-based visualisation of protein secondary structure. In *EG UK Theory and Practice of Computer Graphics (TPCG)* (2008), pp. 115–122. 2

[KBE09] KRONE M., BIDMON K., ERTL T.: Interactive visualization of molecular surface dynamics. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1391–1398. 2

[KC05] KOLB A., CUNTZ N.: Dynamic particle coupling for GPU-based fluid simulation. In *Symposium on Simulation Technique (ASIM)* (2005), pp. 722–727. 2, 5

[KE04] KLEIN T., ERTL T.: Illustrating magnetic field lines using a discrete particle model. In *Vision, Modelling and Visualization (VMV '04)* (2004), pp. 387–394. 2, 4

[KLR04] KOLB A., LATTA L., REZK-SALAMA C.: Hardware-based simulation and collision detection for large particle systems. In *ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (2004), pp. 123–131. 2

[KLR09] KLANN M. T., LAPIN A., REUSS M.: Stochastic simulation of signal transduction: Impact of the cellular architecture on diffusion. *Biophysical Journal 96*, 12 (2009), 5122–5129. 2

[KSN08] KANAMORI Y., SZEGO Z., NISHITA T.: GPU-based fast ray casting for a large number of metaballs. *Computer Graphics Forum 27*, 2 (2008), 351–360. 3

[KSW04] KIPFER P., SEGAL M., WESTERMANN R.: Uberflow: a GPU-based particle engine. In *ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (2004), pp. 115–122. 2

[KW03] KRÜGER J., WESTERMANN R.: Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization 2003* (2003), pp. 287–292. 3

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. In *ACM SIGGRAPH Computer Graphics and Interactive Techniques* (1987), pp. 163–169. 2

[LvLRR08] LINSEN L., VAN LONG T., ROSENTHAL P., ROSSWOG S.: Surface extraction from multi-field particle volume data using multi-dimensional cluster visualization. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1483–1490. 3

[MGE07] MÜLLER C., GROTTEL S., ERTL T.: Image-space GPU metaballs for time-dependent particle data sets. In *Vision, Modelling and Visualization (VMV '07)* (2007), pp. 31–40. 3, 4

[MGK77] MCCAMMON J. A., GELIN B. R., KARPLUS M.: Dynamics of folded proteins. *Nature 267*, 5612 (1977), 585–590. 2

[NM05] NEOPHYTOU N., MUELLER K.: GPU accelerated image aligned splatting. In *International Workshop on Volume Graphics* (2005). 3

[RE05] REINA G., ERTL T.: Hardware-accelerated glyphs for mono-and dipoles in molecular dynamics visualization. In *Eurographics/IEEE VGTC Symposium on Visualization* (2005), pp. 177–182. 2

[SEBH02] SCHMIDT-EHRENBERG J., BAUM D., HEGE H. C.: Visualizing dynamic molecular conformations. In *IEEE Visualization 2002* (2002), pp. 235–242. 3

[SSKE05] STEGMAIER S., STRENGERT M., KLEIN T., ERTL T.: A simple and flexible volume rendering framework for graphics-hardware–based raycasting. In *International Workshop on Volume Graphics 2005* (2005), pp. 187–195. 3

[TCM06] TARINI M., CIGNONI P., MONTANI C.: Ambient occlusion and edge cueing to enhance real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1237–884. 2

[vKvdBT07] VAN KOOTEN K., VAN DEN BERGEN G., TELEA A.: Point-based visualization of metaballs on a GPU. In *GPU Gems 3*, Nguyen H., (Ed.). Addison-Wesley Professional, 2007, ch. 7, pp. 123–148. 2

[vMAF*07] VAN MEEL J. A., ARNOLD A., FRENKEL D., ZWART S. F. P., BELLEMAN R. G.: Harvesting graphics power for MD simulations. *Molecular Simulation 34*, 3 (2007), 259–266. 2

[WE98] WESTERMANN R., ERTL T.: Efficiently using graphics hardware in volume rendering applications. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998), pp. 169–177. 3

[Wes90] WESTOVER L.: Footprint evaluation for volume rendering. *Computer Graphics (Proceedings of SIGGRAPH 1990) 24*, 4 (1990), 367–376. 3

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