https://doi.org/10.2352/ISSN.2470-1173.2018.01.VDA-333

High-Quality Volume Rendering of Dark Matter Simulations

Ralf Kaehler; KIPAC/SLAC; Menlo Park, CA, USA

Abstract

Phase space tessellation techniques for N–body dark matter simulations yield density fields of very high quality. However, due to the vast amount of elements and self-intersections in the resulting tetrahedral meshes, interactive visualization methods for this approach so far either employed simplified versions of the volume rendering integral or suffered from rendering artifacts.

This paper presents a volume rendering approach for phase space tessellations, that combines state-of-the-art order–independent transparency methods to manage the extreme depth complexity of this mesh type. We propose several performance optimizations, including a view–dependent multiresolution representation of the data and a tile–based rendering strategy, to enable high image quality at interactive frame rates for this complex data structure and demonstrate the advantages of our approach for different types of dark matter simulations.

Introduction

N–body dark matter simulations are an important tool to study the formation of the large–scale structure in the Universe. In this numerical approach, the dark matter distribution is sampled by a set of particles of equal mass, whose positions are updated over time, according to the overall gravitational potential [26, 11, 27]. Many methods for analyzing dark matter simulations rely on the computation of accurate mass densities, for example, to identify features of the cosmic web or to predict dark matter annihilation signals. There exist various techniques to interpolate the mass at the discrete particle positions to generate a density field defined everywhere in the computational domain; e.g. resampling approaches using auxiliary grid structures [14, 5, 23] or *SPH* techniques that superimpose kernel functions centered at the particle positions [22]. However, these methods are often subject to artifacts due to sampling noise, which is problematic for many applications.

An improved density computation technique for *N*–body simulations, introduced in [24, 2], uses the particles to construct a tessellation of the 3–dimensional dark matter sheet, which is embedded in the 6-dimensional phase space. A well–defined density field is obtained by projecting the tessellation into configuration space and adding the mass density contributions of all elements that overlap the same spatial location. This method has been successfully applied in many applications, for example, to reduce numerical artifacts of N-body simulations [12, 13], or to create smooth maps of gravitational lensing potentials around dark matter halos [3].

A drawback of the phase space tessellation approach, in particular for visualizations, is the complexity of the resulting tetrahedral meshes, due to the large number of elements and the excessive amount of self-intersections – often in the order of 10^4 cells overlap in overdense regions. Previous methods to visualize dark matter simulations using this technique were either based on versions of the volume rendering integral without absorption [19], subject to high–frequency image noise [17] or did not achieve interactive frame rates [18]. This paper presents a volume rendering approach for phase space tessellations, that

- combines the advantages of a single–pass k–buffer approach with a view–adaptive voxelization grid,
- incorporates several optimizations to tackle the unique problems arising from the large number of self–intersections and extreme depth complexity,
- proposes a view–dependent level–of–detail representation of the data that requires only minimal preprocessing
- and thus achieves high image quality at interactive frame rates, enabling an artifact–free visualization of subtle, thin features like complex caustic structures in overdense regions.

Related Work

There exists extensive work on the visualization of point–based datasets resulting from N–body and *SPH* simulations. GPU–assisted hierarchical splatting, via spatial clustering, was presented in [15, 16]. An interactive rendering approach for very large N-body datasets, proposed in [10], employed a continuous level-of-detail particle representation and a hierarchical quantization scheme to compress the particle coordinates and data attributes. GPU-assisted voxelization for *SPH*, using an adaptively discretized viewing volume, has been proposed in [9] and, more recently, in [33]. However, in contrast to our work, all the above-mentioned approaches assume that the mass is concentrated at the particle positions, which can lead to image artifacts, as for example demonstrated in [19].

There is also a vast amount of literature on the visualization of data on tetrahedral grids. Cell–projection methods usually employ the *Projected Tetrahedra* (PT) algorithm, that decomposes each tetrahedron into a set of triangles and assigns scalar values for the entry and exit points of the viewing rays to each vertex [25]. A GPU–assisted method for decomposing the tetrahedra into triangles using the PT algorithm was presented in [31]. An artifact–free PT rendering approach using a logarithmically scaled pre–integration table was proposed in [20]. Other GPU–assisted raycasting methods for tetrahedral grids have been discussed in [29, 8]. Phase space meshes differ from common tetrahedral grids by the excessive amount of self–intersections, while the above–mentioned approaches assume that the grid cells can be processed separately, in a view–consistent order.

An alternative method to render tetrahedral grids is to remap the data to grid structures that are better supported by current

graphic hardware architectures, using cartesian grids [30] or oct–tree data structures [28]. These approaches are problematic in our case, as can miss thin tetrahedra that fall between consecutive sampling slices. These are particularly important in the phase space element approach, as they sample caustic structures of extremely high density in overdense regions of the domain. A voxelization approach for large phase space tessellations on GPU clusters has been presented in [18]. However, in contrast to the method proposed in this paper, it did not support view–adaptive depth layering at interactive frame rates.

An alternative strategy to deal with semi–transparent self–intersecting geometry, is adopted in *order–independent transparency* (OIT) techniques, which sort the rendering primitives on a pixel–pixel basis. The A–buffer approach [6] first captures lists of all fragments for each pixel, which are then sorted and blended in a second step. Recently added graphics hardware features, like fast atomic counters and writable GPU buffers, allow hardware implementations of the A–buffer, like per–pixel lists (PPL) [32, 21]. However, due to the extremely high depth complexity of phase space tessellations, a direct application of these approaches to extract and combine all fragments, would result in prohibitive memory requirements. A hybrid volume rendering method for phase space tessellations that reduces this memory overhead was proposed in [17]. In this multi–pass approach, pixels with low depth complexity are rendered exactly via an OIT method, whereas the depth range of pixels with high depth complexity is quantized using a binning approach. This enables interactive frame rates, but even for moderately sized dark matter simulations, only a small fraction of the pixels meets the criterion for the exact OIT method. The binning approach however, can lead to high–frequency image noise, since the density contributions of thin tetrahedra, whose front–, and back–faces fall into the same density bin, cancel each other out exactly. This is, for example, problematic for displaying thin, but nevertheless important features like caustic structures, whose superpositions generate overdense regions.

We propose a single–pass rendering solution that is not affected by image noise and extracts the depth range close to the viewpoint using a variant of the k–buffer approach [4], and voxelizes distant tessellation elements onto a viewport aligned 3D grid.

The Phase Space Element Approach

In this section, we briefly review the main ideas of the phase space tessellation method [24, 2], that are most relevant for this paper. *N*–body simulations follow the motion of dark matter in the Universe using point-like mass sources, also called *tracer particles* or just *tracers*. Their positions are updated according to the overall gravitational forces, assuming that the mass is concentrated at the tracers' positions. At the simulation's initial time step, the tracers are aligning with the vertices of a regular grid, referred to as *Lagrangian grid* in the remainder of this paper. Since all these cells have the same volume, in the phase space tessellation approach an equal amount of mass is assigned to each of them. The logical grid connectivity is kept fixed over time, whereas the spatial locations of the vertices vary according to the actual positions of the tracer particles, causing a deformation of the embedding of the logical rectangular Lagrangian grid in configuration space.

Given the constant mass per cell, its time-dependent volume provides an estimate of its local density. However, instead of directly using the hexahedral cells to compute the volumes, it is computational more efficient to first tessellate them by tetrahedra, as these are always convex, independent of the relative positions of their vertices. We will follow the tessellation choice using 6 tetrahedra, as for example used in [19]. So, with m denoting the constant mass per tetrahedron and $\sum_i V_i(\mathbf{x},t)$ the total volume of all tetrahedra that cover the spatial location $\mathbf{x}$ at time t, the total mass density at $\mathbf{x}$ and t is given by

$$\rho_{tot}(\mathbf{x},t) = m\sum_i \frac{1}{V_i(\mathbf{x},t)}. \quad (1)$$

The Volume Rendering Algorithm

Our approach consists of the following steps: First, the particle positions are uploaded to the GPU, where the vertex coordinates for each cell are sampled, the associated tetrahedra are constructed and their mass densities are computed. Next, for each pixel, we generate a depth-sorted list of the first k depth layers, given by the intersection points between the viewing ray and the tetrahedra closest to the camera. The density contributions of ray-segments beyond the first k fragments are sampled and accumulated using a view–adaptive 3D voxel grid. Finally, the resulting densities are mapped to colors and opacities, which are blended in a front–to–back order. We will discuss implementation details and optimization strategies for each part of this approach in the next subsections.

Geometry Construction

The particle positions are stored in an *RGB* floating–point 3D–texture, sorted by their logical indices on the initial Lagrangian grid. In an *instanced* rendering call – one instance per cell – the cell's 8-vertices are sampled and passed to a geometry shader, where the 6 tetrahedra are constructed and their volumes are computed. Only the front-facing triangles, as well as the normals of all tetrahedra's faces, are generated at this stage and passed to the fragment shader, where the viewing ray's entry point into the tetrahedra, as well as the length of the ray intersection, are computed.

K–Buffer Construction

Graphics hardware vendors recently added support for atomic operations on 64-bit integer data types [1], which enables the generation of correctly depth-sorted fragment lists in a single rendering pass, as for example demonstrated in [7].

For the single-pass k–buffer construction, we allocate a buffer of k unsigned 64-bit integers for each pixel and initialize it by setting all bits to 1. In the fragment shading stage, the density of each tetrahedron, as well as the depth of the ray's entry point, d_{in}, are encoded in the lower, respectively upper 32-bits of a 64-bit integer. The data item is inserted into the pixel's fragment list by looping over the k-entries, atomically exchanging it with stored entries, if its depth is smaller. In case the data was inserted, i.e. if it was at least closer to the camera than the last fragment in the list, a corresponding data item encoding the ray's exit point at depth d_{out}, along with the negative density $-\rho$, is generated and tested for insertion, starting at the next larger list position. The pseudocode for this is outlined in Algorithm 1.

The result is a depth-sorted list of the viewing rays entry and

Algorithm 1 Single-Pass k-Buffer Construction

```
1:  function INSERT_FRAGMENT( depth, ρ, search_start )
2:      new_data ← convert (ρ, depth) to 64-bit uint
3:      test_data = new_data
4:      insertion_pos = search_start
5:      for i ← search_start, k do
6:          old_data ← atomicMin(k_buffer[i], test_data)
7:          if hi32(old_data) == 0xFFFFFFFFu then
8:              break
9:          end if
10:         test_data = max ( old_data, test_data )
11:         if ( test_data == new_data ) then
12:             ++insertion_pos
13:         end if
14:     end for
15:     return insertion_pos
16: end function
17:
18: function FRAGMENT_SHADER_MAIN( )
19:     (d_in, d_out, ρ) ← compute intersection parameters
20:     search_start = INSERT_FRAGMENT(d_in, ρ, 0) + 1
21:     if search_start < k then
22:         INSERT_FRAGMENT(d_out, −ρ, search_start)
23:     end if
24: end function
```

exit points for the k closest tetrahedra faces, along with the tetrahedra's density ρ at entry points and $-\rho$ at the exit points, see Figure 1.

View–Adaptive Voxelization

In order to capture the density contributions of tetrahedra that extend beyond the depth interval of the elements in the k–buffer, we employ a voxelization via a view–adaptive grid with logarithmic depth spacing, see Figure 1. We, therefore, allocate a viewport–aligned, floating–point 3D–texture with $n_x * n_y * n_d$ texels, where $n_x * n_y$ is the viewport resolution, and n_d indicates the number of depth layers for each pixel. In principle the total depth interval, spanned by the voxel grid at each pixel, depends on the depth of the last item in the pixel's k–buffer and the maximal depth of all geometry at that pixel – information we could record during the construction of the k-buffer.

However, this would require two separate rendering passes, one for the k–buffer construction and another one for the voxelization of the remaing tetrahdra, impacting the overall performance due to the duplicated geometry and density computation. Instead, we generate the k–buffer and the voxelization grid in the same rendering pass, simultaneously inserting tetrahedra contributions to both. This results in some amount of redundant information for the first layers of the voxelization grid, but the overhead is low, since the depth interval spanned by the k–buffer is usually small.

In the fragment shading stage, after updating the k–buffer as described above, we atomically add contributions of

$$\rho_T * (dl_T \cap dl_i),$$

to the grid. Here ρ_T denotes the density of the tetrahedron T, and $dl_T \cap dl_i$ is the length of the intersection between the ray segment

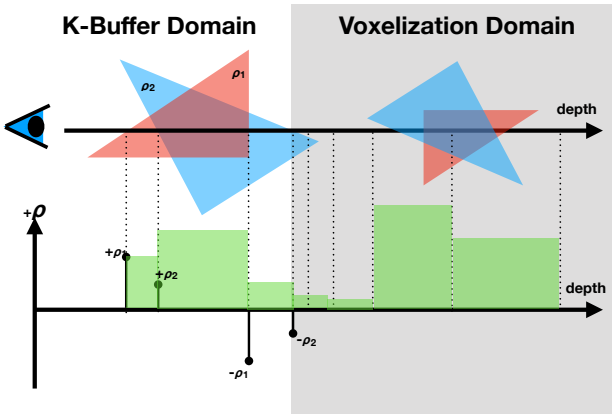

Figure 1. *Combined k–buffer and voxelization grid: In this example, two elements close to the camera result in four fragments, that are inserted into the depth–sorted k–buffer. Front–facing fragments contribute positive densities, whereas back–facing fragments contribute negative densities. Depth layers beyond the first k–fragments fall into the voxelization domain. The resulting densities at each location are shown in green in the lower part of the image.*

inside T and the depth layer i of the voxelization grid. Note that these contributions have the unit *mass per area*, and we need to divide the final values of each cell by the layer's depth interval dl_i, to transform them back to density units. Also note that our method is capable of capturing the contributions of thin tetrahedra, whose front-, and back-facing fragments fall into the same depth layer, which is crucial for correctly displaying caustic structures inside overdense regions, like dark matter filaments and halos, that consist of thousands of very thin phase space elements, as shown in Figure 2.

Color Mapping and Blending

In the last step, we combine the densities in a view–consistent order, which is straightforward in our case, since the k–buffer and the voxel grid are already explicitly, respectively implicitly sorted by depth.

Thus, we render a screen-filling quad and simply loop over the entries in the k–buffer, accumulate the density contributions at each buffer item, map the current density at each interval to colors and opacities, while taking into account the corrections due to the varying interval lengths, see Figure 1. The colors are combined using the standard front-to-back blending equation. Analogously, the densities stored in the view–adaptive voxelization grid are sampled, mapped to colors and opacities and blended, taking advantage of the fast interpolation and caching support for 3D textures on current graphics hardware. As discussed above, the first layers of the voxelization grid overlap the depth range covered by the k–buffer, and thus we start the accumulation at a depth corresponding to the last entry in the k–buffer, and rescale the grid's density value at that layer based on the overlap with the buffer's last depth interval. Finally, the resulting colors and opacities for each pixel are stored in the global framebuffer.

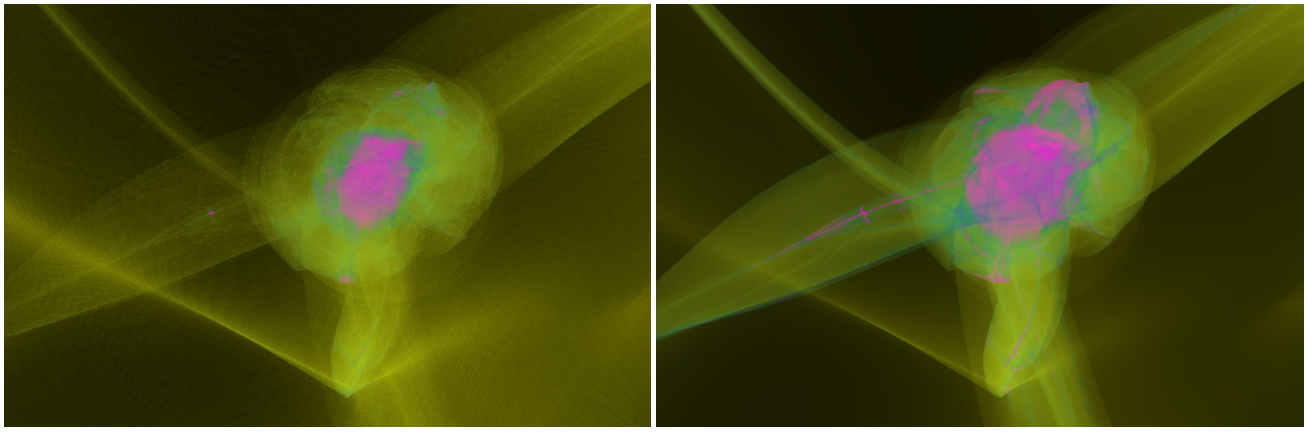

Figure 2. *Comparison between the hybrid rendering approach presented in [17] (on the left) and our method (on the right), using the same number of depth layers. The failure to capture many caustics consisting of thin tetrahedra in the central halo, results in high–frequency noise in the left image, whereas our method correctly displays these regions.*

Optimizations for Phase Space Tessellations

In this section, we will describe performance optimizations to the above-mentioned approach, that take advantage of special features of phase space tessellations.

Tiled Rendering

Subdividing a high-resolution image into smaller tiles is a well-known strategy to increase the number of depth layers that can be extracted with the limited amount of GPU memory, and also helps to avoid problems related to rendering large amounts of semi–transparent primitives, like GPU *overdraw*. A straightforward tiling approach is problematic in our case, since the generation of the tetrahedra in the geometry shader takes up a noticeable fraction of the whole rendering time and would add a constant overhead to each pass, since the culling of geometry outside the view–frustum is performed in later stages of the rendering pipeline.

In order to avoid the generation of geometry outside the tile's viewing frustum, we subdivide the position texture into smaller cubical *blocks* of cells that share a layer of texels at adjacent faces, and pre-compute the partially overlapping, axis–aligned minimum bounding boxes (*AABB*) spanned by the particles of each block. Only blocks whose *AABB* intersect the current frustum are rendered, as depicted in Figure 4.

For our examples, a choice of 8^3 cells per block resulted in a high fraction of culled geometry, but produced in the order of 10^5 blocks. Rendering that many blocks via separate high–latency CPU draw calls would have introduced a performance overhead, and we therefore decided to perform the entire intersection computation on the GPU, utilizing two compute shaders. The first one determines the blocks' bounding boxes and offsets on the Lagrangian grid, and stores them in a GPU buffer. This buffer is kept on the GPU and accessed during the rendering pass for each tile from a second compute shader, which detects the intersections between the blocks and the current view–frustum. For each intersection, an atomic counter is incremented and used as an offset into a separate GPU buffer that stores the IDs of the blocks that overlap with the frustum. Once the intersection computation is finished, the final value of the counter determines the total number of instances required for the tile's single *instanced* draw call. In the following vertex shader, the instance IDs determine the block number and cell offset within the block.

Multiresolution Rendering

The partition of the particle grid into smaller blocks also allows for an efficient view–dependent multiresolution representation of the phase space tessellation. For blocks that intersect the current view frustum, we compute the screen space extensions of their AABBs. The size of the projections determines the blocks' resolution levels, which are encoded in the highest bits of the block IDs and stored in the list of active blocks for the actual viewpoint, as described above. On the CPU, the IDs are inserted into different lists - one for each resolution level. The lists are bound to the GPU and for each of them, a separate *instanced* draw call is issued, with the number of instances given by the resolution and the block size in cells. The data for each resolution level is obtained by downsampling from the next higher resolution, averaging each vertex position with its nearest 8 neighbors. Note that this requires adapting the mass assigned to each tetrahedron sampled on level r by a factor of 2^{3r}, which is done in the geometry shader when the densities are computed.

K–Buffer Construction

By construction, the k–buffer is always sorted, and thus inserting new elements via a binary search is usually beneficial. Of course, this approach works best if the geometry is rendered in an almost depth-sorted order, since in this case, the buffer converges to its final form earlier, allowing to avoid vast amounts of atomic operations for later fragments. As discussed above, an exact sorting of the tetrahedra is infeasible, due to the tessellation's large amount of self-intersections. However, since dark matter particles usually move only fractions of the whole computational domain, relative to their initial position on the Lagrangian grid, there is a high correlation between their logical grid index (i, j, k) and their actual position at later times. We exploit this by sorting the block IDs in the lists based on their center's distance to the camera and by sampling the cells of each block in an order based on the major viewing direction. So if, for example, the viewing

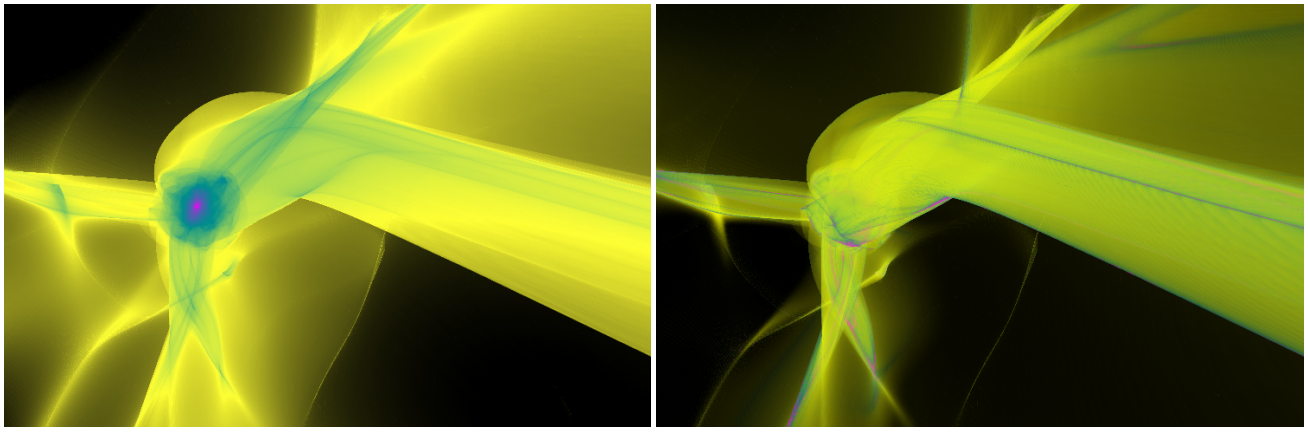

Figure 3. *Comparison between the density projection rendering approach [19] (left) and the volume rendering method proposed in this paper, that takes absorption into account (right). The depth ordering between features, like the two twisted filaments on the bottom, or the top filament and the central halo, become much clearer in the image on the right.*

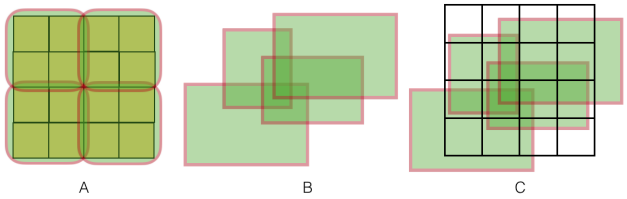

Figure 4. *2D illustration of the optimization for tile-based rendering: (A) shows a Lagrangian grid texture with 5×5 particles, which is subdivided into 4 blocks. (B) The axis–aligned minimum bounding boxes, spanned by the particles of each block, are computed and only blocks that overlap the current tile, part of the black array of cells in (C), are rendered at each pass.*

direction is mainly along the positive x–axis, cells with smaller i–coordinate are processed first, this way increasing the probability that geometry closer to the camera is inserted early into the k–buffer.

Results and Discussion

We conducted our performance measurements on a *Red Hat Enterprise Linux 6* operating system, equipped with a *Nvidia GeForce GTX 1080 Ti* (graphics driver version 384.90), with 12 GB of graphics memory and a 2.27 GHz 8-core *Intel Xeon E5520* CPU. The algorithms were implemented in *C++*, *OpenGL* and the *OpenGL Shading Language*. We used two datasets with different characteristics: a *warm dark matter* (WDM) simulation of a single, highly resolved central halo with detailed caustic structures, shown in Figures 2 and 3, as well as a cold dark matter (CDM) simulation, capturing the halos and filaments of the cosmic web in a region about 350 million light year across, see Figures 9 and10. The datasets consisted of 258^3, respectively 256^3 dark matter particles, resulting in phase space tessellations with about 10^8 tetrahedra. We used a viewport resolution of $750x500$ pixels for all of our measurements.

Figure 5 states the rendering time for the different optimization strategies, i.e. the block–based view–frustum culling, the view–dependent cell–sorting, as well as the screen-projection-based multiresolution representation (with a threshold of one tetrahedron per pixel). Each different column adds another optimization level, starting with the basic rendering approach on the left. We used 5×4 image tiles, a block size of 8^3 cells and 100, respectively 1000 depth layers for the k–buffer and the voxelization grid. The performance numbers show that the view–frustum culling using the block–based approach reduced the rendering times by more than 50%, by drastically reducing the overhead of the geometry construction of tetrahedra that do not contribute to the current image tile. The view–aligned processing order for the Lagrangian cells improved the rendering performance for the CDM data by almost 40%. The effect was less pronounced for the WDM simulation, where a large fraction of the particles accumulated in the central halo region, resulting in a lower correlation between the indices of the vertices on the logical rectangular Lagrangian grid and their positions in actual world coordinates.

Adding the level-of-detail rendering approach, enabled interactive frame rates for both datasets. A detailed comparison in terms of rendering performance versus image quality for different multiresolutions settings is given in Figures 6 and 7, respectively in Figures 10 and 11. A threshold of about one (projected) tetrahedron per pixel results in almost no loss of image quality compared to rendering the full resolution data, while it increased the performance by about 300%. Lower thresholds of two and three tetrahedra per pixel resulted in further performance gains, but led to a higher image error in high–frequency domains of the data, for examples at the boundaries of the domain, or at the caustics, as shown in Figures 10 and 11.

Dataset	basic approach	blocks	cell-sorting	LOD
WDM	4.2	2.0	1.8	0.5
CDM	10.1	4.5	2.7	0.9

Figure 5. *The rendering times (in seconds) for the two datasets and the different optimization approaches.*

The effects on the image quality of different sizes for the k–buffer and the voxelization grid are demonstrated in Figure 9. We increased the number of layers for the k–buffer from 0 to 300,

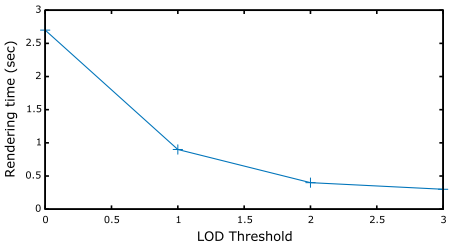

Figure 6. *Performance improvement using the LOD approach for the CDM dataset.*

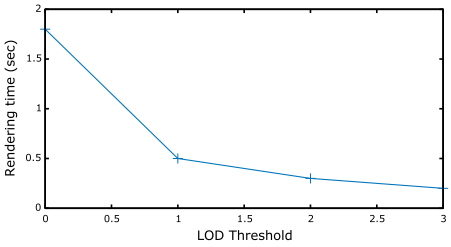

Figure 7. *Performance improvement using the LOD approach for the WDM dataset.*

while the slabs in the voxelization grid were reduced by the same amount, starting with 1000, thus keeping the overall memory budget constant. As shown in Figure 9, using the k–buffer with 100 layers substantially reduced visible banding artifacts, while the effect of adding additional layers to the k–buffer resulted only in minor improvements. However, the rendering performance dropped by about 50% for each additional set of 100 layers in the k–buffer, indicating that the rendering time is dominated by the k–buffer construction and a choice of 100 resulted in a good tradeoff between image quality and rendering performance for our examples.

Conclusions and Future Work

We presented a high quality, order–independent single–pass volume rendering approach for N–body simulation data, based on a phase space tessellation technique. By combining a single–pass k–buffer approach to extract the first depth layers and a view–adaptive voxelization for distant depth intervals, it achieves interactive frame rates, while still capturing important features, like thin caustic structures, with significantly improved image quality compares to previously published methods. We proposed several optimization techniques, including an efficient multiresolution representation of the data, that take advantage of the logical structure of the tessellations and demonstrated the usefulness of our approach with datasets, whose phase space tessellations resulted in complex, self–intersecting meshes with 10^8 tetrahedra.

As future work, we plan to extend our algorithm to support higher–order interpolation inside the tetrahedral elements and also apply it to support dark matter velocity data. We would also like to investigate the feasibility of extending our approach to distributed GPU–cluster environments.

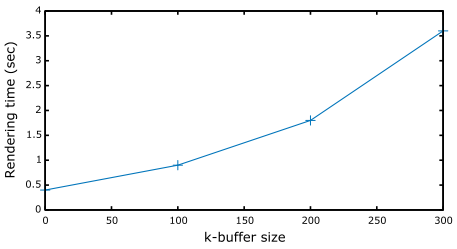

Figure 8. *Rendering performance for increasing k-buffer sizes.*

Acknowledgments

The dark matter simulation datasets were provided by Tom Abel and Oliver Hahn. We would also like to thank Ziba Madhavi and David Stricker for their help with hardware purchases used for this publication.

References

[1] OpenGL Extension 455. `https://www.khronos.org/registry/OpenGL/extensions/NV/NV_shader_atomic_int64.txt`.

[2] Tom Abel, Oliver Hahn, and Ralf Kaehler. Tracing the Dark Matter Sheet in Phase Space. *Monthly Notices of the Royal Astronomical Society*, 427(1):61–76, 2012.

[3] Raul E. Angulo, Ruizhu Chen, Stefan Hilbert, and Tom Abel. Towards noiseless Gravitational Lensing Simulations. *Mon. Not. Roy. Astron. Soc.*, 444(3):2925–2937, 2014.

[4] Louis Bavoil, Steven P. Callahan, Aaron Lefohn, João L. D. Comba, and Cláudio T. Silva. Multi-fragment Effects on the GPU using the k-Buffer. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 97–104, New York, NY, USA, 2007. ACM.

[5] Charles K. Birdsall and Dieter Fuss. Clouds-in-Clouds, Clouds-in-Cells Physics for Many-body Plasma Simulation. *Journal of Computational Physics*, 3:494–511, 1969.

[6] Loren Carpenter. The A-buffer, an Antialiased Hidden Surface Method. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 103–108, New York, NY, USA, 1984. ACM.

[7] Christoph Kubisch. Order Independent Transparency in OpenGL 4.x, GPU Technology Conference 2014. `http://on-demand.gputechconf.com/gtc/2014/presentations/S4385-order-independent-transparency-opengl.pdf`, 2014.

[8] Rodrigo Espinha and Waldemar Celes. High-Quality Hardware-Based Ray-Casting Volume Rendering Using Partial Pre-Integration. In *Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing*, pages 273–, Washington, DC, USA, 2005. IEEE Computer Society.

[9] Roland Fraedrich, Stefan Auer, and Rudiger Westermann. Efficient High-Quality Volume Rendering of SPH Data. *IEEE Transactions on Visualization and Computer Graphics*, 16:1533–1540, November 2010.

[10] Roland Fraedrich, Jens Schneider, and Rüdiger Westermann. Exploring the Millennium Run - Scalable Rendering of Large-Scale

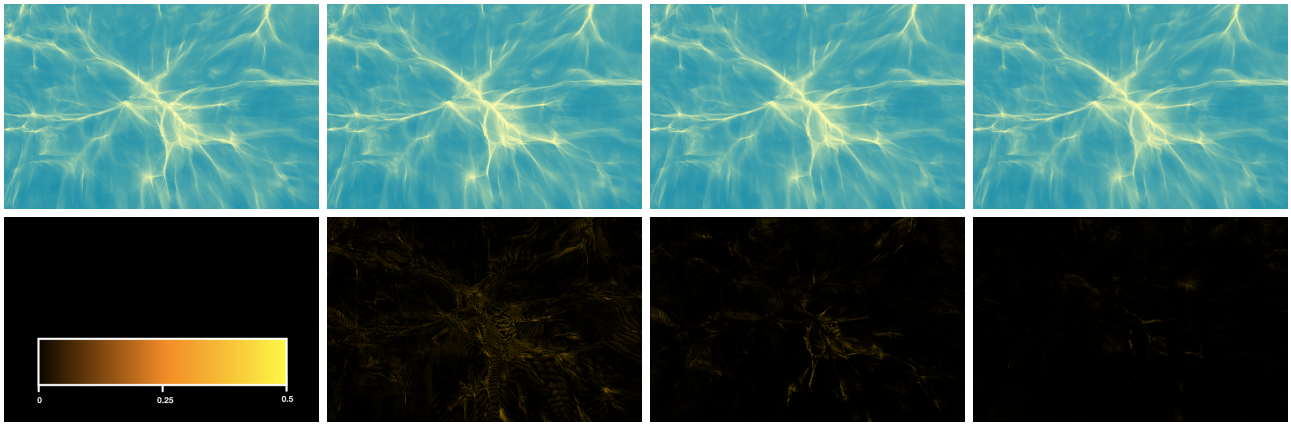

Figure 9. *Image quality for different sizes of the k–buffer and the voxelization grid: the number of depth layers in the k–buffer increases in steps of* 100*, starting with* 0 *on the left, while the number of layers in the voxelization grid is reduced by the same amount, from an initial value of* 1000*. The upper row shows the visualization results, while the lower row displays the relative pixel differences between the image above and the previous one to the left.*

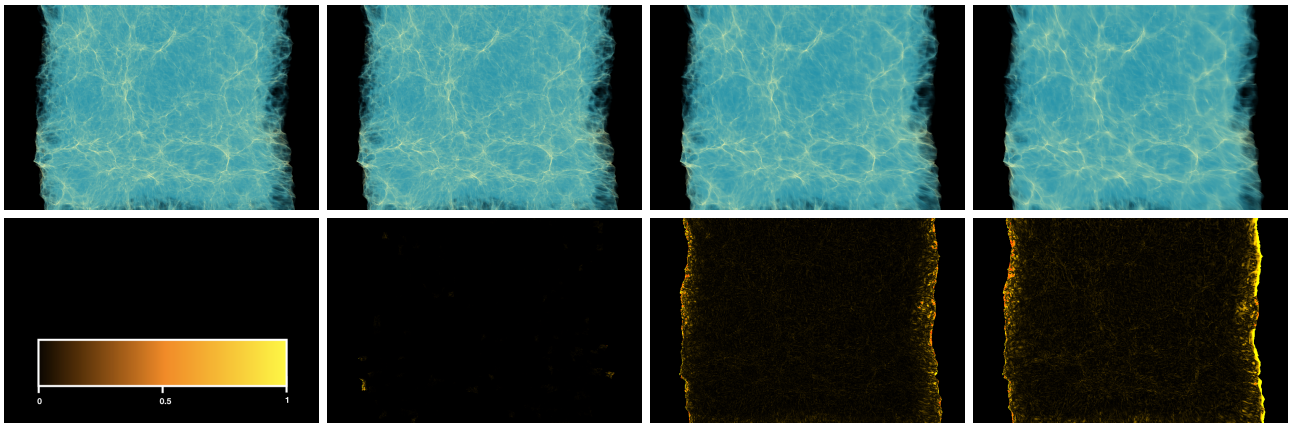

Figure 10. *Renderings of the CDM dataset using different level-of-detail settings. From left to right: the full resolution data and results for refinement thresholds of one, two and three pixels per tetrahedron. The lower row shows the relative errors between each image and the full resolution rendering.*

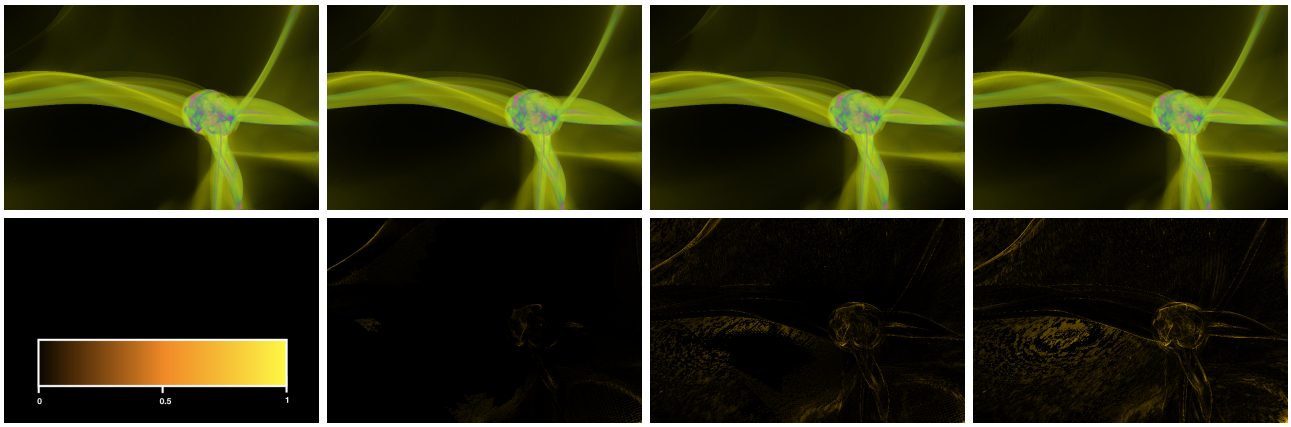

Figure 11. *Renderings of the WDM dataset using the same level-of-detail settings as in Figure 10.*

Cosmological Datasets. *IEEE Transactions on Visualization and Computer Graphics*, 15:1251–1258, November 2009.

[11] Salman Habib, Vitali Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, and Katrin Heitmann. Hacc: Extreme scaling and performance across diverse architectures. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 6:1–6:10, New York, NY, USA, 2013. ACM.

[12] Oliver Hahn, Tom Abel, and Ralf Kaehler. A new approach to simulating collisionless dark matter fluids. *Mon. Not. Roy. Astron. Soc.*, 434:1171, 2013.

[13] Oliver Hahn and Raul E. Angulo. An adaptively refined phase space element method for cosmological simulations and collisionless dynamics. *Mon. Not. Roy. Astron. Soc.*, 455(1):1115–1133, 2016.

[14] Roger W. Hockney and Joseph W. Eastwood. *Computer Simulation Using Particles*. Taylor & Francis, Inc., Bristol, PA, USA, 1988.

[15] Matthias Hopf and Thomas Ertl. Hierarchical Splatting of Scattered Data. In *Proceedings of the 14th IEEE Visualization 2003*, pages 57–65, Washington, DC, USA, 2003. IEEE Computer Society.

[16] Matthias Hopf, Michael Luttenberger, and Thomas Ertl. Hierarchical Splatting of Scattered 4D Data. *IEEE Comput. Graph. Appl.*, 24:64–72, July 2004.

[17] Oleg Igouchkine, Nick Leaf, and Kwan-Liu Ma. Volume rendering dark matter simulations using cell projection and order-independent transparency. In *SIGGRAPH ASIA 2016 Symposium on Visualization*, SA '16, pages 8:1–8:8, New York, NY, USA, 2016. ACM.

[18] Ralf Kaehler. Massively Parallel Computation of Accurate Densities for N-body Dark Matter Simulations using the Phase Space Element Method. *Astronomy and Computing*, 20:68 – 76, 2017.

[19] Ralf Kaehler, Oliver Hahn, and Tom Abel. A Novel Approach to Visualizing Dark Matter Simulations. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2078–2087, 2012.

[20] Martin Kraus, Wei Qiao, and David S. Ebert. Projecting tetrahedra without rendering artifacts. In *Proceedings of the conference on Visualization '04*, VIS '04, pages 27–34, Washington, DC, USA, 2004. IEEE Computer Society.

[21] Stefan Lindholm, Martin Falk, Erik Sunden, Alexander Bock, Anders Ynnerman, and Timo Ropinski. Hybrid Data Visualization Based on Depth Complexity Histogram Analysis. *Computer Graphics Forum*, 34(1):74–85, 2015.

[22] Joseph Monaghan. An Introduction to SPH. *Computer Physics Communications*, 48:89–96, January 1988.

[23] Mark C. Neyrinck. ZOBOV: A parameter-free Void-finding Algorithm. *mnras*, 386:2101–2109, June 2008.

[24] Sergei Shandarin, Salman Habib, and Katrin Heitmann. Cosmic web, multistream flows, and tessellations. *Phys. Rev. D*, 85:083005, Apr 2012.

[25] Peter Shirley and Allan Tuchman. A Polygonal Approximation to Direct Scalar Volume Rendering. In *Proceedings of the 1990 workshop on Volume visualization*, VVS '90, pages 63–70, New York, NY, USA, 1990. ACM.

[26] Volker Springel. The Cosmological Simulation Code Gadget-2. *Monthly Notices of the Royal Astronomical Society*, 364, 2005.

[27] Michael S. Warren. 2hot: An Improved Parallel Hashed Oct-tree N-body Algorithm for Cosmological Simulations. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 72:1–72:12, New York, NY, USA, 2013. ACM.

[28] Manfred Weiler and Thomas Ertl. Hardware-software-balanced resampling for the interactive visualization of unstructured grids. In *Proceedings of the conference on Visualization '01*, VIS '01, pages 199–206, Washington, DC, USA, 2001. IEEE Computer Society.

[29] Manfred Weiler, Martin Kraus, Markus Merz, and Thomas Ertl. Hardware-Based Ray Casting for Tetrahedral Meshes. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, VIS '03, pages 44–, Washington, DC, USA, 2003. IEEE Computer Society.

[30] Ruediger Westermann. The rendering of unstructured grids revisited. In *EG/IEEE TCVG Symposium on Visualization (VisSym '01)*, 2001.

[31] Brian Wylie, Kenneth Moreland, Lee Ann Fisk, and Patricia Crossno. Tetrahedral Projection using Vertex Shaders. In *Proceedings of the 2002 IEEE Symposium on Volume Visualization and Graphics*, VVS '02, pages 7–12, Piscataway, NJ, USA, 2002. IEEE Press.

[32] Jason C. Yang, Justin Hensley, Holger Grün, and Nicolas Thibieroz. Real-time concurrent linked list construction on the gpu. In *Proceedings of the 21st Eurographics Conference on Rendering*, EGSR10, pages 1297–1304, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[33] Tobias Zirr and Carsten Dachsbacher. Memory-efficient on-the-fly voxelization of particle data. In *Proceedings of the 15th Eurographics Symposium on Parallel Graphics and Visualization*, PGV '15, pages 11–18, Aire-la-Ville, Switzerland, Switzerland, 2015. Eurographics Association.