Volume Graphics (2007)
H. - C. Hege, R. Machiraju (Editors)

Simultaneous GPU-Assisted Raycasting of Unstructured Point Sets and Volumetric Grid Data

Ralf Kaehler[1,2], Tom Abel[2] and Hans-Christian Hege[1]

[1]Zuse Institute Berlin (ZIB), Germany
[2]SLAC/KIPAC, Stanford University, USA

Abstract

In the recent years the advent of powerful graphics hardware with programmable pixel shaders enabled interactive raycasting implementations on low-cost commodity desktop computers. Unlike slice-based volume rendering approaches GPU-assisted raycasting does not suffer from rendering artifacts caused by varying sample distances along different ray-directions or limited frame-buffer precision. It further supports direct implementations of many sophisticated acceleration techniques and lighting models.

In this paper we present a GPU-assisted raycasting approach for data that consists of volumetric fields defined on computational grids as well as unstructured point sets. We avoid resampling the point data onto proxy grids by directly encoding the point data in a GPU-octree data structure. This allows to efficiently access the (semi-transparent) point data during ray traversal and correctly blend it with the grid data, yielding interactive, high-quality rendering results. We discuss approaches to accelerate the rendering performance for larger point sets and give real world application examples to demonstrate the usefulness of our approach.

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

1. Introduction

In numerical analysis discretized data defined on computational grids as well as unstructured point sets are very common. Often both data types are employed at the same time in order to model different quantities or phenomena. This is for example the case for hydrodynamic simulations where mesh-based and mesh-less representations of the field variables are popular. As an example consider astrophysical simulations that approximate interstellar gas densities on structured grids and compute dark matter components using SPH-solvers (*Smoothed Particle Hydrodynamics* solvers) or large scale galaxy distributions via n-body simulations, resulting in potentially large grid-, and point-based data sets. Another example is the visualization of hydrodynamic simulations, which often imply the display of grid based data sets as well as point primitives for streamline tracing. Scenarios like this require appropriate interactive techniques to visualize both kinds of data simultaneously.

In the last years sophisticated hardware-accelerated visualization approaches have been developed for both types of data. Texture-based direct volume rendering and GPU-assisted raycasting for grid data and splatting approaches for point sets are among the most popular ones. Combining both types of rendering approaches with interactive frame rates is unproblematic if at least one of them is mapped to completely opaque colors. In this case the opaque quantity is rendered in a first pass whereas the transparent one is rendered in a second pass, utilizing hardware-supported depth buffer and blending operations to combine the two rendering results in real-time.

The situation is more problematic if both types of data are mapped to (semi-)transparent colors, since this case requires interactive, view-dependent depth sorting. Nevertheless this case is very important since non-opaque splats are a very popular and natural representation for many types of point-based data sets and the use of suitable semi-transparent colormaps is often crucial for good depth perception in direct volume rendering.

Resampling the point data onto an auxiliary grid structure and displaying both grids simultaneously using hardware-supported direct volume rendering is a possible approach but has its limitations. This either implies low-pass filtering or using highly refined grids to capture the high frequencies that are usually contained in the data. In the first case image quality is sacrificed; in the second one texture-memory usage and computational complexity are increased.

In this paper we propose an approach for hardware-accelerated, high-quality volume rendering of grid data and unstructured point sets. Our algorithm is based on GPU-assisted raycasting. The point data is organized in an octree that is encoded using 3D textures to efficiently access the data on the GPU and enables pixel-accurate depth sorting of the data fields. We further propose several acceleration techniques to improve rendering performance by reducing the number of sampling operations. The resulting approach is applied to visualize real-world simulation results.

2. Related Work

The most popular hardware-accelerated techniques for direct volume rendering of data on structured grids are texture-based slicing and GPU-assisted ray-casting approaches.

The basic idea of texture-based volume rendering, introduced in 1993 [CN93], is to approximate the volume rendering integral by mapping the data volume to a 3D-texture, respectively a stack of 2D-textures and to exploit graphics hardware to extract sets of slices from the volume. The resulting slices are rendered in a view-consistent order and their color intensities are combined by hardware-accelerated framebuffer blending. The algorithm has been extended in many ways to incorporate sophisticated acceleration techniques and optical models, see e. g. [KPHE02, EKE01a, GWGS02, WWH*00].

With the advent of programmable graphics hardware that supports flexible fragment programs, it was feasible to perform the ray-integration on a per-pixel basis at interactive frame rates on standard desktop computers, as proposed in [RGW*03, KW03, SSKE05]. In this approach a fragment program is executed for each pixel that is covered by the projected bounding box of the data volume. The ray is parameterized in texture coordinates and the volume rendering integral is computed within the shader. GPU-assisted raycasting is very attractive, since it enables a direct realization of advanced shading models and does not suffer from typical rendering artifacts of slice-based methods, which are caused by limited precision for the blending operations or varying sample distances for different ray directions. The approach has been extended to multi-resolution data in [VSE06, Lju06, KWAH06].

Point primitives are employed in many different areas of visualization and computer graphics. Consider for example footprint-based splatting approaches of structured and unstructured grid data, e.g. hierarchical splatting [LH91, MMC99] and more recently [CDM06]. Point primitives have been successfully applied in surface rendering algorithms [ABCO*03, RL00]. A rendering approach for large unstructured point data using point splats has been recently proposed in [HE03]. The authors employ a sophisticated data structure to efficiently sort the points on the CPU and stream the resulting subsets to the GPU. The approach has been extended for time-dependent data and molecular dynamics visualization [HLE04, RE05].

Octree-textures have been proposed in [BD02] and an implementation tailored for GPU-processing was presented in [LHN05, KLS*05]. The underlying idea is to represent each node of the octree as a set of 8 RGBA-texels. The RGB-channels are used as indices to texels that represent child nodes whereas the alpha channel is employed to distinguish internal nodes and leaf nodes. This way a recursive top-down traversal of the octree-texture can be realized in fragment shaders via dependent-texture lookups.

We base our approach on GPU-assisted raycasting since it achieves image quality comparable to pure software solutions and is flexible enough to support the rather complex sampling operations. We choose to store the points in an octree, since this data structure can be efficiently accessed on the GPU and does not require the storage of bounding box coordinates for the internal nodes, which can be recomputed on-the-fly based on the current sample position and node depth.

The remainder of this paper is organized as follows: We start with a general description of the overall rendering algorithm in Section 3. Next, the point data structure and its representation on the GPU are discussed in Subsection 3.1. We then propose acceleration techniques in Subsection 3.2 and give results of the application of the approach to simulation data in Section 4. We end with conclusions and a discussion of future work in Section 5.

3. The Algorithm

We will start this section with a presentation of the overall structure of the rendering algorithm. In order to simplify the discussion we will refer to the scalar data discretized on the grid structure as *grid data*, denoted by g and to the unstructured point set as *point data*, denoted by p. Although the approach in principle generalizes to an arbitrary number of grid and point data sets, we will restrict ourself to the case of one grid data set and one point data set for the sake of simplicity. We implemented it for structured grids, but it could also be incorporated into GPU ray-casting schemes for unstructured grids like [WKME03].

The overall structure of the GPU-accelerated raycasting algorithm is based on the single-pass approach described in [SSKE05]. The front faces of the data volume's bounding box are rendered with texture coordinates at the vertices that

Figure 1: *The image above shows a time step of a large scale cosmological simulation of the epoch of hydrogen reionization of the universe gas in a region of approximately one hundred million light years across at a time when the universe was 900 million years old. The point data represent the galaxies providing the ultra violet radiation whereas the blue and red regions are the interstellar gas densities given on a structured grid. The blue colors correspond to the hot ionized material and the red to the neutral primordial intergalactic medium.*

equal the vertex coordinates. For each covered pixel the interpolated texture coordinates are accessible within the fragment shader and directly give the object space coordinates of the ray's entry point into the data volume. After computing the ray direction in object space and transforming it to texture coordinates, the intensity integration is performed. Therefore the ray is discretized into a set of segments s_i that are processed from front-to-back to permit early ray termination, once the accumulated opacity exceeds a certain threshold. Along the ray both fields are sampled and mapped to color intensity and opacity. The grid data is stored as a 3D-texture. The data samples are obtained by texture lookups and mapped to color intensity $I_g(s_i)$ and opacity $\alpha_g(s_i)$ via an user-defined color table.

Next the intensity of the point data needs to be computed. As mentioned above we store this data in an octree. We will describe in detail how the structure is generated and accessed within the fragment shader in the next subsection. For the moment we assume that the resulting intensity and opacity for the point data at ray location s_i is given as $I_p(s_i)$, respectively $\alpha_p(s_i)$. Now the separate contributions of the two fields are combined to yield the resulting local intensity, which is given as the sum of the two components weighted by their opacities

$$I(s_i) := \alpha_g(s_i)\, I_g(s_i) + \alpha_p(s_i)\, I_p(s_i). \tag{1}$$

We assume the following relation between the separate opacities $\alpha_k(s_i)$ that are usually stored in the user-defined colormaps and the corresponding extinction coefficients $\xi_k(s_i)$

$$\alpha_k(s_i) := 1 - e^{-\xi_k(s_i)\,|s_i|} \quad \text{for} \quad k \in \{g, p\},$$

where $|s_i|$ denotes the length of the ray segment. By adding the separate extinction coefficients we obtain the following result for the total opacity of the segment

$$\begin{aligned} \alpha(s_i) &:= 1 - e^{-(\xi_g(s_i)+\xi_p(s_i))\,|s_i|} \\ &= 1 - e^{-\xi_g(s_i)\,|s_i|}\; e^{-\xi_p(s_i)\,|s_i|} \\ &= 1 - (1-\alpha_g(s_i))\,(1-\alpha_p(s_i)). \end{aligned} \tag{2}$$

Finally the total color intensity I_Σ and opacity α_Σ up to the ray-segment are updated according to the "front-to-back blending equation"

$$I_\Sigma = I_\Sigma + (1-\alpha_\Sigma)\, I(s_i), \tag{3}$$

$$\alpha_\Sigma = \alpha_\Sigma + (1-\alpha_\Sigma)\, \alpha(s_i). \tag{4}$$

The next subsections explains in more detail how the point data is stored and accessed within the fragment shader.

3.1. The Point Data Structure

We organize the points in an octree data structure, since it allows an efficient implementation with 3D-textures and thus can be evaluated directly in the GPU during ray traversal. In the following we assume that the only point features are center position c_p and radius r_p, but the approach can be extended to incorporate additional point attributes.

Points are inserted into the octree based on intersections between the spheres defined by c_p and r_p and the octree nodes. The root node of the octree is recursively refined until the maximal number of points that intersect a certain node is smaller than a predefined threshold or a maximal tree depth is reached. Notice that it is not sufficient to base the insertion of the point solely on the criterion if the center c_p is

contained in the node. The reason for this is that the octree is sampled locally along the viewing rays and parts of the spheres that extend across the boundaries of their nodes would be missed, as illustrated in Figure 2.

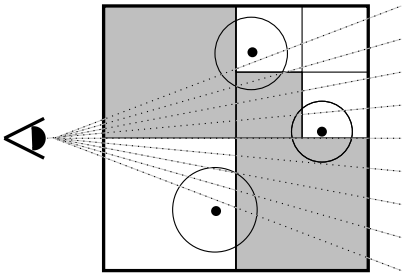

Figure 2: *Parts of the points within the grey-shaded nodes would be missed during ray-traversal, if points were inserted into the octree based only on the locations of their centers.*

Storing the point data multiple times increases the memory requirements, but permits a correct sorting of the points with pixel precision. This is usually not possible for footprint based point rendering, which can suffer from locally incorrect depth sorting for parts of the splats that exceed the node boundaries.

In order to efficiently access the point data in the fragment shader, we map it to a GPU-octree data structure [LHN05, KLS*05]. The octree structure is encoded by a three-dimensional 4-channel (RGBA) texture. Each internal node of the tree is represented by 2^3 RGBA-texels. The distinction of internal and leaf nodes is based in the texel's alpha channel:

- $a = 1$ indicates that the node is an internal node. In this case the RGB-triple is interpreted as an index to the node's "child"-texels.
- $a = 0.5$ indicates that the texel represents a leaf node and the RGB-triple stores an index to another data texture that encodes the point data, see discussion below.
- $a = 0$ indicates that the texel represents an empty node that is not further refined.

Since we usually must store more than one point per octree leaf node and each point requires at least 4 floats for its center and radius, we can not encode the point data directly within the index texture. We use a separate 3D floating-point texture for this purpose. The RGB-values in the index texture that correspond to leaf nodes are used as texture coordinates to the floating point data texture. They index to the texel that stores the first point entry for the corresponding leaf node. The end of each node's point list is indicated by inserting a texel with a 4th-component that equals zero into the data texture. The whole point encoding strategy is illustrated in Figure 3.

In the current implementation we use 8-bit for each channel of the index texture. This allows us to index into 256^3 texels of the RGBA-floating point texture, which corresponds to 16.7 million points entries. The octree index texture lookup is done as described in [LHN05]. Once the index into the data texture is obtained, the data texture is sampled until a texel with an alpha-component of 0.0 is processed, which indicates the end of the point list for this octree leaf node.

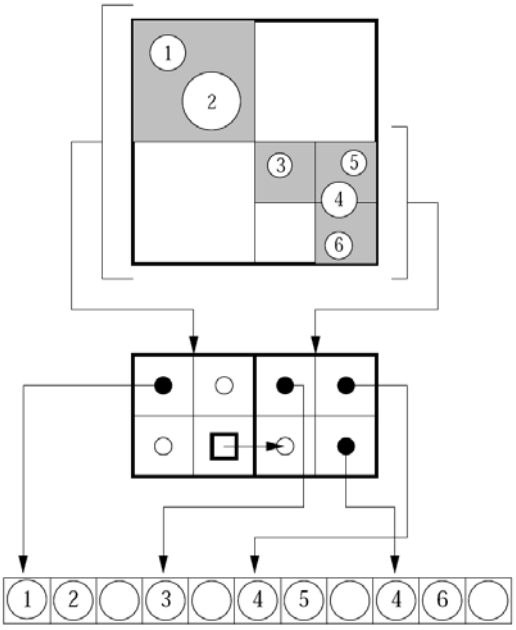

Figure 3: *In the upper part of the figure a simple 2D example of a point data set with 6 points and the corresponding quadtree data structure are shown. The middle part of the figure depicts the resulting index texture. Black circles represent leaf nodes that point into the data texture on the lower part of the figure. White circles represent empty leaf nodes and the white square is an internal node that stores an index to the 4 child nodes. The white circles in the lower part of the figure represent RGBA-texels with vanishing alpha component (radius) that indicate the end of the particle list for each node. Notice that point number 4 intersects the bounding boxes of two leaf nodes, and thus it is inserted two times into the data texture.*

Sampling the octree index and data textures is a computationally intensive operation and turns out to be the bottleneck in our approach. It is therefore crucial to minimize the sampling operations for the octree. Strategies for this will be discussed in the following subsection.

3.2. Performance Optimizations

An efficient method to increase the rendering performance is to adapt the sampling rate for the point data to the underlying

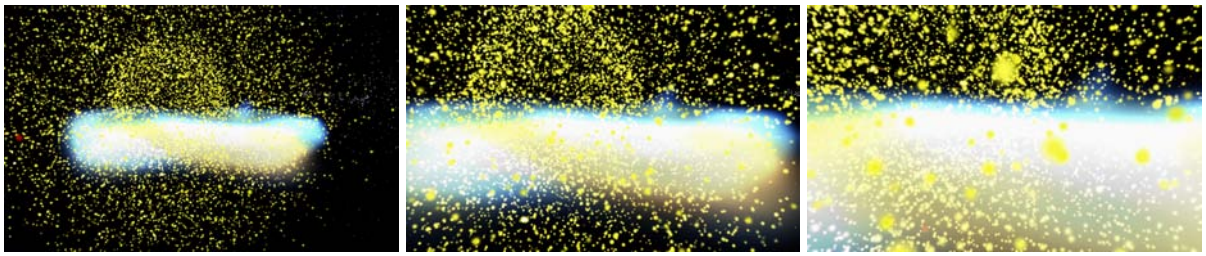

Figure 4: *The images show a time step of a galaxy formation simulation for three different view points. The interstellar gas density was modeled on a structured grid and the point data presents stars that formed within the galaxy.*

data characteristics, for example by skipping ray-segments that do not intersect any points. Therefore the rays are intersected with the leaf nodes during ray traversal. Since we do not store bounding box coordinates for the nodes within the index or data textures, we must compute this information on-the-fly. First the depth l of the octree at an initial sample location x is obtained via the octree texture lookup and from this information the bounding box coordinates of the leaf node are computed as

$$x_{min} = \Omega_{x,min} + \lfloor \frac{x - \Omega_{x,min}}{2^l(\Omega_{x,max} - \Omega_{x,min})} \rfloor (\Omega_{x,max} - \Omega_{x,min})$$
$$x_{max} = x_{min} + 2^{-l}(\Omega_{x,max} - \Omega_{x,min}),$$

and similar for the y and z coordinates. Here $\lfloor\rfloor$ denotes the floor function and $\Omega_{max}, \Omega_{min}$ are the root level extensions of the octree. The resulting ray-segment is tested for intersection with the points stored in the leaf node and it is skipped completely in case no intersection occurs.

The ray-segments should be sampled with a sample distance that corresponds to the smallest point that influences that region. This could still result in an computational overhead if the box is large compared to size of this point. The overhead could be reduced using a pre-integrated volume rendering approach [EKE01b] for the point data.

We chose a different approach that works for rotationally symmetric points and allows us to reduce the number of point sampling operations to one per segment. Therefore each of the point centers that is stored at the node is projected onto the ray as shown in Figure 5. If this projection is contained in the intersection of the ray and the leaf node, its intensity and opacity contribution is computed based in the distance r of the center to the ray. In principle every function that depends solely on r and has support in $[0, r_p]$, were r_p is the point radius, is admissible. It could for example be given by a user-defined one-dimensional texture. We currently use a Gaussian decay in the opacity of the form

$$\alpha(r) \sim \begin{cases} exp(-\kappa(\frac{r}{r_p})^2) & : \quad r \leq r_p \\ 0 & : \quad r > r_p \end{cases}$$

Once the intensity for each point within the node is obtained,

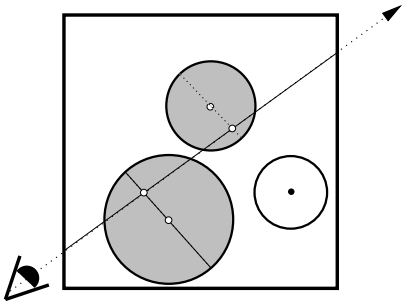

Figure 5: *In order to reduce the number of samples required to reconstruct the point data, the point centers are projected onto the viewing ray. If the projection is within the intersection of the ray and the leaf node, the intensity and opacity of the point is computed based on the distance between the center and its projection onto the ray. This reduces the number of necessary octree sampling operations to one per ray-segment.*

the resulting contribution can be computed using Equation 1 and 2. Notice that this step requires a view-dependent sorting of the points within the leaf. Currently we omit this step in order to increase the rendering performance. In our experience this usually does not result in noticeable visual artifacts, since only small spatial regions are affected.

As a further acceleration technique we implemented a two-pass hybrid rendering approach for large point sets, compare Figure 6. In the first pass points outside a user-defined region of interest (ROI) are rendered as textured splats perpendicular to the viewing direction and blended into an 16-bit offscreen render buffer that is equipped with an additional depth buffer. The ROI is currently defined by a radial camera-distance threshold. The depth buffer is written for every newly rendered splat, which leads to the correct result, since the points are rendered in back-to-front order using a CPU-octree data structure.

The offscreen frame- and depth buffer is accessed in the fragment shader in the second pass that performs the actual GPU-raycasting step. The rendering of the grid and point

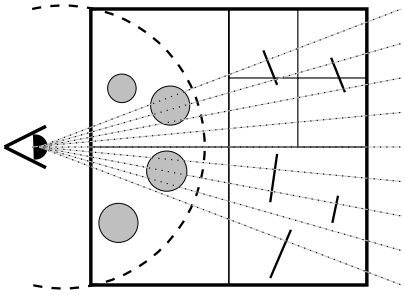

Figure 6: *Two pass rendering approach that allows to trade-off between image quality and rendering performance. Points outside a user-defined region of interest are rendered using hardware-accelerated texture splatting, whereas points inside the region are rendered via the computationally more intensive GPU-assisted raycasting approach.*

data is carried out as discussed above until the depth stored in the offscreen buffer is reached, corresponding to the depth of the closest splat outside the ROI. At that location the resulting color from the first pass is added according to Equation 3 and 4. From this point on only grid data is integrated. Of course the resulting pixel intensity differs from the correctly blended one in the case that more than one point is intersected outside the ROI, but for distant regions the visual artifacts are usually not noticeable. Increasing and decreasing the region of interest allows an adjustable trade-off between rendering performance and image quality.

4. Result

We tested the rendering performance on a *Windows XP* system with a *Nvidia Geforce 8800 GTX* graphics card equipped with 768 Mbytes of graphics memory. The shader was implemented with the *OpenGL Shading Language*. The splats in the hybrid rendering approach discussed at the end of Section 3.2 have been implemented using *OpenGL* point sprites.

We used a threshold of 8 points per node for the tests. The screen resolution was 830×520 pixels. The first data set is a time step of a cosmological reionization simulation, see Figure 1. The grid data was given on a 256^3 structured grid and the point data consists of about $1.5 * 10^5$ points. The dimensions of the octree's index and data textures were $256 \times 256 \times 50$, respectively $256 \times 256 \times 5$ texels. We tested the rendering performance for three different cases: for the case that all points were rendered as point sprites, the hybrid rendering approach that uses textured splats for points in the second half of the data volume, and the case that the whole data was rendered using the GPU-raycasting approach. The resulting rendering performance was 6.3, 3.3, respectively 2.1 frames per second.

The second data set is a time step of a galaxy formation simulation, see Figure 4. The grid data was given on a 256^3 structured grid and the point data consists of about $1.2 * 10^5$ points. The dimensions of the octree's index and data textures were $256 \times 256 \times 47$, respectively $256 \times 256 \times 4$ texels. The achieved rendering performance for this data was 6.6, 3.4, respectively 2.3 frames per second.

A direct visual comparison of the hybrid rendering approach for different regions of interest is given in Figure 7. In the left image, rendered using point sprites for all points, artifacts in the near field are clearly visible. The image on the right was rendered with the GPU-based approach that does not suffer from the artifacts due to incorrect depth sorting. In the middle image only points in the near field were rendered with the GPU-based approach, while in the far field point sprites were employed. Subtle differences between the images are visible only in small background regions.

5. Conclusions and Future Work

In this paper we presented an approach for GPU-assisted raycasting of combined grid and unstructured point data sets. The point data is stored in a GPU-octree data structure to efficiently access it during ray traversal. The approach achieves high image quality, since it is capable of pixel accurate depth sorting of the different data components. All computations, including the blending operations are performed in full 32-bit floating point precision. The approach further supports analytical expressions for point shading that do not necessarily have to be rotationally symmetric.

We proposed several acceleration techniques to increase the rendering performance that is affected by the number of octree's index and data texture lookup operations. We applied adaptive sampling and skipping of ray segments that contain no point data. We further presented a technique that reduces the number of sampling steps of the octree down to one sample per ray-segment for rotational symmetric point shading by projecting the centers of the points onto the ray. In that case the color and intensity distribution is computed by a radial function. We further proposed a two-pass rendering mode, in which points outside a user-defined region of interest are rendered as hardware-supported *OpenGL* point sprites. In the second pass they are combined with the high-quality GPU-based raycasting that accesses the point data in the octree only within the region of interest. This allows significant performance gains while introducing only minor artifacts due to incorrect depth sorting for small background regions.

As future work we will incorporate a multi-resolution approach for the particle data by storing coarse points at the internal nodes, representing the points of the subtree nodes. We also plan to support more than 4 attributes per point by using multiple texels per point in the data texture. We will further extend the algorithm to support multi-resolution grid data.

Figure 7: *The images show a comparison of the image quality obtained for the hybrid rendering approach discussed in Section 3.2 for different extensions of the region of interest. In the image on the left is the result for a vanishing region of interest, so all points were rendered as textured splat. The artifacts in front are clearly visible. The image in the middle is the result for a region of interest corresponding to half the extension of the data volume and for the picture on the right all point data was rendered solely with the GPU-assisted ray-casting approach.*

6. Acknowledgments

This work was partially supported by the *Max-Planck-Institute for Gravitational Physics (Albert-Einstein Institute)*, Potsdam/Germany and the *NSF CAREER* award AST-0239709 from the *National Science Foundation*.

We thank Marcelo Alvarez (*KIPAC/SLAC*) and Ji-hoon Kim (*KIPAC/SLAC*) for their kind permission to use the reionization and galaxy formation data sets. We further thank Steffen Prohaska (*Zuse Institute Berlin*) for fruitful discussions.

References

[ABCO*03] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9*, 1 (January 2003), 3–15.

[BD02] BENSON D., DAVIS J.: Octree textures. In *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), ACM Press, pp. 785–790.

[CDM06] CHILDS H., DUCHAINEAU M. A., MA K.-L.: A scalable, hybrid scheme for volume rendering massive data sets. In *Eurographics Symposium on Parallel Graphics and Visualization* (2006), pp. 153–162.

[CN93] CULLIP T., NEUMANN U.: *Accelerating volume reconstruction with 3D texture mapping hardware*. Tech. Rep. TR93-027, Department of Computer Science at the University of North Carolina, Chapel Hill, 1993.

[EKE01a] ENGEL K., KRAUS M., ERTL T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (2001), pp. 9–16.

[EKE01b] ENGEL K., KRAUS M., ERTL T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *HWWS '01: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware* (New York, NY, USA, 2001), ACM Press, pp. 9–16.

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive rendering of large volume data sets. In *VIS '02: Proceedings of the Conference on Visualization '02* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 53–60.

[HE03] HOPF M., ERTL T.: Hierarchical Splatting of Scattered Data. In *Proceedings of IEEE Visualization '03* (2003), IEEE.

[HLE04] HOPF M., LUTTENBERGER M., ERTL T.: Hierarchical Splatting of Scattered 4D Data. *Computer Graphics and Applications 24*, 4 (2004), 64–72.

[KLS*05] KNISS J., LEFOHN A., STRZODKA R., SENGUPTA S., OWENS J. D.: Octree textures on graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches* (New York, NY, USA, 2005), ACM Press, p. 16.

[KPHE02] KNISS J., PREMOZE S., HANSEN C., EBERT D.: Interactive translucent volume rendering and procedural modeling. In *Proceedings of IEEE Visualization 2002* (2002), IEEE Computer Society Press, pp. 109–116.

[KW03] KRUEGER J., WESTERMANN R.: Acceleration techniques for GPU-based volume rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 38.

[KWAH06] KAEHLER R., WISE J., ABEL T., HEGE H.-C.: GPU-Assisted Raycasting for Cosmological Adaptive Mesh Refinement Simulations. In *Proceedings of the International Workshop on Volume Graphics 2006* (Boston, 30 - 31 July 2006), Eurographics / IEEE VGTC 2006, pp. 103–110.

[LH91] LAUR D., HANRAHAN P.: Hierarchical splatting:

a progressive refinement algorithm for volume rendering. *SIGGRAPH Comput. Graph. 25*, 4 (1991), 285–288.

[LHN05] LEFEBVRE S., HORNUS S., NEYRET F.: *GPU Gems 2 - Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison Wesley, 2005, ch. Octree Textures on the GPU, pp. 595–613.

[Lju06] LJUNG P.: Adaptive Sampling in Single Pass, GPU-based Raycasting of Multiresolution Volumes. In *Proceedings of the International Workshop on Volume Graphics 2006* (Boston, 30 - 31 July 2006), Eurographics / IEEE VGTC 2006, pp. 39–46.

[MMC99] MUELLER K., MOELLER T., CRAWFIS R.: Splatting without the blur. In *VIS '99: Proceedings of the conference on Visualization '99* (Los Alamitos, CA, USA, 1999), IEEE Computer Society Press, pp. 363–370.

[RE05] REINA G., ERTL T.: Hardware-Accelerated Glyphs for Mono- and Dipoles in Molecular Dynamics Visualization. In *Proceedings of EUROGRAPHICS - IEEE VGTC Symposium on Visualization 2005* (2005), K. W. Brodlie and D. J. Duke and K. I. Joy, (Ed.), pp. 177–182.

[RGW*03] ROETTGER S., GUTHE S., WEISKOPF D., ERTL T., STRASSER W.: Smart hardware-accelerated volume rendering. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003* (Aire-la-Ville, Switzerland, Switzerland, 2003), Eurographics Association, pp. 231–238.

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: A multiresolution point rendering system for large meshes. In *Siggraph 2000, Computer Graphics Proceedings* (2000), Akeley K., (Ed.), ACM Press / ACM SIGGRAPH / Addison Wesley Longman, pp. 343–352.

[SSKE05] STEGMAIER S., STRENGERT M., KLEIN T., ERTL T.: A simple and flexible volume rendering framework for graphics-hardware-based raycasting. In *Fourth International Workshop on Volume Graphics* (Washington, DC, USA, 2005), pp. 187– 241.

[VSE06] VOLLRATH J. E., SCHAFHITZEL T., ERTL T.: Employing Complex GPU Data Structures for the Interactive Visualization of Adaptive Mesh Refinement Data. In *Proceedings of the International Workshop on Volume Graphics 2006* (Boston, 30 - 31 July 2006), Eurographics / IEEE VGTC 2006, pp. 55–58.

[WKME03] WEILER M., KRAUS M., MERZ M., ERTL T.: Hardware-Based Ray Casting for Tetrahedral Meshes. In *Proceedings of IEEE Visualization '03* (2003), IEEE, pp. 333–340.

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMAN K., ERTL. T.: Level-of-detail volume rendering via 3D textures. In *IEEE Volume Visualization and Graphics Symposium 2000* (2000), pp. 7–13.