Institut für Informatik
Technische Universität München

Interactive Visualization Techniques for Large-Scale Particle Simulations

Roland Fraedrich

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. H. M. Gerndt
Prüfer der Dissertation:	Univ.-Prof. Dr. R. Westermann
	Univ.-Prof. Dr. A. Kolb, Universität Siegen

Die Dissertation wurde am 19.01.2012 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 20.03.2012 angenommen.

To Eva

Abstract

Particle-based simulation techniques such as smoothed particle hydrodynamics (SPH) have gained much attention due to their ability to avoid a fixed discretization of the simulation domain. Especially in the field of astrophysics, such simulations have emerged as an indispensable instrument to gain insight into the structure formation process of the Universe and to prove or contradict hypotheses on its matter distribution. For large data sets with up to several billions of particles, the amount of data exceeds the capabilities of conventional visualization techniques and thus an interactive visual analysis is not feasible.

In this thesis, I investigate scalability limitations in the visualization of such large-scale particle-based simulations and I present methods to allow the rendering of billion-particle data sets at interactive rates on current PC architectures. For this purpose, I develop a hierarchical multi-resolution particle representation that minimizes the amount of data required for rendering while preserving sub-pixel accuracy and visually continuous levels of detail. For a fast selection and access of the data, an efficient out-of-core and in-core data management is employed and I present techniques to exploit the capabilities of current graphics accelerators to enable interactive visual data exploration. Furthermore, I introduce a novel volume rendering pipeline for efficient high-quality visualization of particle data allowing for rendering options such as direct volume rendering and isosurface rendering. Finally, I present a new hierarchical space-time data structure for particle sets, which allows for a scale-space analysis of structural formation processes in the simulated fields. I demonstrate the quality and performance of my methods for the rendering of fluid and gas dynamics SPH simulations consisting of millions to billions of particles.

Zusammenfassung

Partikelbasierte Simulationverfahren wie Smoothed Particle Hydrodynamics (SPH) haben aufgrund ihrer Fähigkeit eine feste Diskretisierung zu vermeiden, erhöhte Aufmerksamkeit erlangt. Insbesondere in der Astrophysik haben sich derartige Simulationen als unentbehrliches Hilfsmittel erwiesen, um Einsicht in die Strukturformierungsprozesse des Universums zu gewinnen und Hypothesen über dessen Materieverteilung zu beweisen oder diese zu widerlegen. Für große Datensätze mit bis zu mehreren Milliarden Partikeln übersteigt die Datenmenge die Kapazitäten konventioneller Visualisierungstechniken und eine interaktive visuelle Analyse ist nicht durchführbar.

In dieser Arbeit untersuche ich die Grenzen der Skalierbarkeit in der Visualisierung von solchen großen partikelbasierten Simulationen und präsentiere Methoden, die das Rendern von Datensätzen mit Milliarden von Partikeln in interaktiven Frameraten auf heutigen PC-Architekturen erlauben. Zu diesem Zweck entwickle ich eine hierarchische Datenrepräsentation, die die zu rendernde Datenmenge minimiert und gleichzeitig eine Subpixel-Genauigkeit und ein visuell kontinuierliches Detaillevel sicherstellt. Um eine schnelle Auswahl und einen raschen Zugriff auf die Daten zu ermöglichen, verwende ich ein effizientes out-of-core und in-core Datenmanagement. Zudem präsentiere ich Methoden, um unter Ausnutzung der Fähigkeiten heutiger Grafikbeschleuniger eine interaktive visuelle Exploration der Daten zu ermöglichen. Ich stelle insbesondere eine neuartige Volume Rendering Pipeline zur effizienten und hochwertigen Visualisierung von Partikeldaten vor, die Rendering-Verfahren wie eine direkte Volumenvisualisierung und das Rendern von Isoflächen erlaubt. Abschließend präsentiere ich eine neue hierarchische Raum-Zeit-Datenstruktur für Partikeldaten, welche eine Scale-Space-Analyse von strukturellen Formierungsprozessen in der Simulation erlaubt. Ich zeige die Qualität und Leistungsfähigkeit meiner Methoden für das Rendern von SPH-Simulationen aus der Fluid- und Gasdynamik mit Millionen bis Milliarden Partikeln.

Acknowledgements

I gratefully acknowledge the support of all of the people who made this thesis possible. First and foremost, I would like to thank my advisor Prof. Dr. Rüdiger Westermann. Having read several papers from his group during my studies at the Friedrich-Alexander-Universität Erlangen-Nuremberg, his Computer Graphics and Visualization Group had become the prime address for me to pursue research in the field of scientific visualization. I am very grateful for the opportunity to work with him, for his supervision, his time commitment, and the numerous discussions. This thesis certainly would not have been possible without him.

I would like to thank the co-authors that have contributed to the journal articles that are part of this thesis, in particular Stefan Auer and Dr. Jens Schneider. I have enjoyed working with them. Furthermore, I am thankful to all of my current and former colleagues that have always been open for discussions: Dr. Kai Bürger, Shunting Cao, Matthäus Chajdas, Christian Dick, Raymund Fülöp, Dr. Joachim Georgii, Stefan Hertel, Dr. Polina Kondratieva, Prof. Dr. Martin Kraus, Prof. Dr. Jens Krüger, Hans-Georg Menz, Tobias Pfaffelmoser, Marc Rautenhaus, Florian Reichl, Dr. Thomas Schiwietz, Marc Treib, Mikael Vaaraniemi, and Jun Wu.

I had contact to several great scientists at the Max-Planck-Institute for Astrophysics: Volker Springel, Gerard Lemson, Markus Rampp, Klaus Reuter, Rüdiger Pakmor and Fritz Röpke. Thank you very much for giving me insight into the field of astrophysics and for providing the cosmological datasets. I also would like to thank Matthias Teschner for providing the fluid dynamic data sets that have been used in this thesis.

Furthermore, I would like to thank Prof. Dr. Günther Greiner, Prof. Dr. Marc Stamminger and all my former colleagues from the Computer Graphics Chair in Erlangen. You have drawn my interest towards computer graphics and visualization, and being part of your group was a wonderful experience.

Finally, I am enormously thankful to my wife, parents, family, and friends for giving me all the support I needed during the last years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Numerical simulations have become an important instrument in science and industry. Capable of providing detailed insights into natural phenomena and complex physical processes, they are of particular importance in cases where physical experiments are expensive, time-consuming, or when information is hard or impossible to measure. A prominent example is the field of astrophysics, where—compared to the age and size of the universe—all physical measurements constitute only a local snapshot of space and time, and where phenomena such as black holes and dark matter can only be observed in an indirect manner. Here, numerical methods contribute significantly to the investigation of physical processes and the validation of hypotheses, and they enable insight into the universe that can not be obtained by other means.

In the past decades, particle-based simulation techniques have received increased attention due to their ability to avoid a fixed discretization of the simulation domain. The concept of these methods is that the computational elements carrying physical quantities are free to move and interact with each other based on application-specific governing equations. Particle-based methods benefit from their spatial adaptivity, flexible deformability, and free boundaries. One of the oldest and best-established techniques is smoothed particle hydrodynamics (SPH). It was developed simultaneously by Lucy [Luc77] and Gingold and Monaghan [GM77] in 1977 in the astrophysical domain. Since its introduction, SPH has been subject to continuous improvement and has proven successful in the simulation of phenomena on many orders of magnitude, ranging from micro-scale to astronomical scale. Today, SPH is extensively used to investigate various cosmological processes such as stellar collisions, supernovae, and the evolution of the universe. Furthermore, it was successfully adapted to other fields of research including quasi-incompressible fluids, heat conduction, shock simulations, and fractures of brittle solids.

For the analysis of numerical simulations, an important role is taken by scientific visualization [HF11]. Due to the perceptual and cognitive abilities of humans, scientists can gain insights from a visual representation much faster and in a much wider extent than from results that are derived from mathematical models and statistical analyses. Moreover, statistical approaches are restricted to aspects of interest that are known in advance and that can be described in a mathematical manner to allow for computational evaluations or a data-driven feature search. Visualizing the data, on the other hand, moves scientists into the center of the exploration process and incorporates their skills and experience to recognize and visually analyze remarkable features, anomalies, and complex processes.

However, for many particle-based simulations an interactive visual analysis nowadays is not feasible. The primary reason for this is that visualizing particle-based simulations is a complex task that typically requires the interpolation of quantities from an unordered set of data. This is especially challenging as today's simulations are performed at ever increasing scales. The resulting amount of data demands sophisticated visualization algorithms that are able to process data that can not be held in-core. One pertinent example from the field of astrophysics is the *Millennium Run* [SWJ^{+}05], in which 10 billion particles were used to simulate the evolution of the universe. Since its computation in 2005, the Millennium Run has been subject to ongoing research that has resulted in more than four hundred scientific publications so far[1]. However, visualizations of the data set have only been available to offline rendering, requiring several minutes for a single image and multiple hours for rendered animations. Accordingly, a visual data analysis has been severely limited. Even for much smaller data sets containing hundreds of thousands of particles, interactive visualization techniques have only been feasible based on simple render modes using points or textured sprites. For high-quality volume rendering, on the other hand, the visualization of such data is also restricted to offline processing.

1.1 Objective

The objective of this thesis is to overcome the present restrictions in the visual exploration of particle-based simulations by developing interactive visualization techniques that are specifically designed with respect to the scalability to large-scale data sets. With the term "large-scale" we refer to data sets that are too large to be interactively rendered with existing visualization methods and which often exceed the capacity of

[1]A list can be found at `http://www.mpa-garching.mpg.de/millennium/`.

the graphics memory or even the main memory. In particular, we focus on smoothed particle hydrodynamics, which is one of the most important and most frequently used particle-based simulations. Handling orders of magnitude of many billions of particles, it has been used to perform several of the largest simulations to present time.

The most important, yet conflicting goals for an effective visual data analysis of large-scale data are visualization quality and rendering performance. Particular questions investigated in this thesis are whether it is feasible to perform interactive visualizations of the largest of today's SPH simulations and which quality penalties have to be accepted to achieve this. We will further investigate the maximum number of particles for which high-quality volume rendering can be provided at interactive frame rates as well as the rendering times that can be achieved for visualizing billion-particle data sets in such a visualization mode. In addition, we pursue the question how structural formation processes and particle motions can be effectively visualized in these data sets. To account for the technical facilities of scientists analyzing such simulations, we particularly focus on algorithms for the application on desktop computers that are equipped with commodity graphics cards.

1.2 Challenges

For an effective visual data analysis, interactivity is of crucial importance due to multiple reasons. Changing the viewing perspective is essential for understanding the three-dimensional shape of structures of interest. The camera movement enhances the depth perception due to motion parallax and allows the user to react if points of interest are occluded. Interactive modifications of data-dependent viewing parameters such as isovalues and transfer functions help the user to gain deeper insights by revealing or emphasizing structures or properties of interest. For time-dependent data sets, animations support the understanding of dynamic processes. Interaction in the time domain allows the user to focus on particular periods of time, to adapt the playback speed to his need, or to pause the animation in order to spatially investigate a specific time step. Finally, interactive visualizations are important for real-time simulations, which have gained much attention in the recent years. Here, rendering techniques are required that provide an instant visualization and that can be used to interact with the simulation in a computational steering environment.

The development of interactive visualization techniques in recent years was greatly influenced by the technical evolution of dedicated graphics hardware. Driven by the demand for more realistic 3D graphics in computer games, graphics processing units

(GPUs) have evolved from devices that were limited to rasterization based on a fixed-function pipeline architecture to powerful and freely programmable processors that permit the implementation of a vast variety of algorithms. Yet, the efficiency of GPU-based rendering techniques heavily depends on an algorithm's suitability for an implementation on the underlying parallel streaming architecture. Furthermore, the limited video memory and bandwidth restrictions regarding the geometry and rasterization throughput often turn out to be significant bottlenecks when visualization algorithms are applied to large-scale data sets.

Unfortunately, these constraints particularly affect interactive visualizations of SPH data due to multiple reasons. The flexibility that makes particle-based methods superior in terms of adaptability and memory demands during the simulation makes the rendering of the particle sets much more complicated. Visualization of SPH data requires the reconstruction of a continuous scalar field at the sampling points using a kernel-based interpolation scheme, for which all particles contributing to the particular sampling point need to be determined. Even if these search operations are supported by space partitioning data structures, this interpolation is extremely expensive and difficult to implement and accelerate on GPUs. Therefore, interactive visualizations so far have been restricted either to simplified rendering techniques avoiding kernel interpolations, or to high quality volume rendering of particle sets with an upper limit of a few ten thousand elements [OKK10].

Today, many simulations contain millions to billions of particles and, thus, exceed this limit by far. Furthermore, researchers aim at performing ever larger simulations to increase accuracy and size of the simulated domain. This process is supported by the growing computational power of super computers, which is still exponentially increasing in correspondence with Moore's law [Moo65, DeB04]. For such large-scale data sets, it is obvious that the challenge of an interactive visualization goes beyond the rendering itself as it requires the treatment of data that vastly exceeds the capacities of dedicated graphics memory and main memory. To significantly reduce the amount of data to be rendered, multi-resolution representations and data compression schemes can be employed. However, these methods must be carefully chosen to avoid any negative impact on the visualization quality. Furthermore, the largest data sets will still require an out-of-core processing and the streaming of data from storage devices, which should not affect the visual exploration of the data set. In comparison to stored particle data, ad-hoc visualizations for interactive particle simulations are even more challenging since the applied rendering algorithms can not rely on expensive pre-computations. Instead, spatial acceleration schemes must either utilize existing space partitioning data structures that are part of the simulation, or be highly efficient to be performed during rendering, for example by exploiting the coherence of the particle distribution between successive frames.

Another challenge is the visual analysis of dynamic processes within time-dependent SPH data sets. In cosmological simulations, processes such as the building of matter clusters and the formation of stars, galaxies, and halos are of particular interest. However, cosmological data sets contain a rather fuzzy matter distribution in which motion is very hard to perceive. This makes it very difficult to identify formation and splitting processes of particle clusters by only visualizing the matter distribution along the time domain via animations. One approach to reveal motion within SPH data is the rendering of particle traces, as known from pathlines within unsteady vector fields. However, for data sets containing millions of elements, the practicability of this approach is limited due to occlusion and visual cluttering, as well as by the geometry throughput of modern GPUs. Moreover, by solely following individual particles on the simulation level, it is very difficult to analyze cluster formations and splits in particle distributions. Therefore, sophisticated techniques are required that deal with these problems and allow for an efficient visual data analysis of large-scale particle flows.

1.3 Contribution

In this thesis, we address the described issues and present rendering techniques for the interactive visualization of large-scale particle distributions and their formation processes on desktop computers. To handle these data sets, we develop a hierarchical, visually continuous particle representation. We apply compression schemes that drastically reduce the amount of data that is transferred and rendered. These are designed to support a very efficient decoding during rendering and preserve a very high accuracy. We employ efficient data management algorithms to optimize the data transfer from secondary memory to the GPU's video memory such that bandwidth limitations and latencies are hidden from the user as far as possible.

We further introduce a new efficient volume rendering pipeline on commodity graphics cards that eliminates existing restrictions for the interactive visualization for SPH data. This technique utilizes a novel data structure, called the *perspective grid*. This structure allows an efficient reconstruction of a continuous data field along the viewing rays of classical ray casting based on a GPU-based voxelization of particles into a volumetric texture. By employing an adaptive sampling width that results in an almost isotropic sampling rate in view space, we can adjust the resampling effort according to the contribution on the rendered image. The resampling into the perspective grid is optimally suited for the combination with a hierarchical particle representation, which further increases rendering performance and helps to avoid aliasing artifacts. It is important to note, however, that our visualization approach is also highly efficient without a multi-resolution representation and can be directly coupled with GPU-based simulations. As with other texture-based volume rendering techniques, the perspective grid allows isosurface ray casting, high-quality volume renderings with arbitrary transfer functions, and mixed rendering options using both techniques. Combined with our efficient data management techniques, we demonstrate that our rendering approach enables an interactive exploration of million-particle data sets at high quality. Furthermore, we present simplified render algorithms that can be used to visualize the largest of today's data sets containing many billions of particles at interactive rates.

Finally, we introduce a new space-time hierarchy to visually analyze motions and clustering processes within time-dependent particle distributions. By clustering particles with coherent traces at various scales, our hierarchy can illustrate dynamic processes of large-scale data sets by minimizing the problems of occlusion and visual cluttering. Furthermore, it helps to reveal cluster mergers and splits within the data field. Regions of interest can be identified and emphasized by adapting the size and color of trajectories based on cluster properties, or by tracking interactively selected particle subsets. Besides this motion visualization, the space-time hierarchy can serve as hierarchical data representation for high-quality volume rendering of scalar fields. This allows to render the motion paths in the context of the matter distribution without the need for an additional particle representation.

1.4 Outline

The remainder of this thesis is structured as follows. Chapter 2 introduces technical constraints for rendering large-scale data sets on current PC architectures. In particular, we describe the memory hierarchy that needs to be considered and give a brief overview on the functionality and programmability of modern GPUs. Chapter 3 reviews elementary methods on which our work is based and describes existing techniques that we apply in subsequent chapters, including SPH, principle component analysis, vector quantization, and volume rendering. This chapter contains no scientific contribution but provides the reader with the fundamental background to understand and implement the algorithms presented in this work.

In Chapter 4, we investigate scalability limitations for large-scale data sets and we present techniques such as a multi-resolution particle hierarchy that enable a visual exploration of the Millennium Run on current PC architectures. To provide an interactive exploration, we apply and optimize an order-independent rendering scheme. Chapter 5 introduces the perspective grid as a new resampling data structure and demonstrates its use for an interactive high-quality visualization of SPH data sets containing millions of particles. For cosmological billion-particle data sets, we discuss a simplification of the visualization approach that allows renderings at a few frames per second with only minor quality penalties. In Chapter 6, we focus on the visualization of dynamic processes within SPH simulations and we introduce a novel space-time hierarchy that allows a scale-space analysis of particle motions within large-scale particle sets. We demonstrate the effectiveness of our method by applying it to particle sets with many millions of elements. The thesis is concluded with a summary of the proposed techniques and an outlook on future work.

1.5 List of Publications

The Chapters 4 to 6 include research results that were published during the creation of this thesis in the following peer-reviewed articles:

1. R. Fraedrich, J. Schneider, and R. Westermann, *Exploring the Millenium Run - Scalable Rendering of Large-scale Cosmological Datasets*, IEEE Transactions on Visualization and Computer Graphics **15** (2009), no. 6, 1251–1258 [FSW09].

2. R. Fraedrich, S. Auer, and R. Westermann, *Efficient High-quality Volume Rendering of SPH Data*, IEEE Transactions on Visualization and Computer Graphics **16** (2010), no. 6, 1533–1540 [FAW10].

3. R. Fraedrich and R. Westermann, *Motion visualization of particle simulations*, IS&T/SPIE Conference on Visualization and Data Analysis (VDA 2012), 2012, 82940Q-1–12 [FW12].

Chapter 2

Hardware Considerations

In this chapter, we provide the reader with the essential knowledge about today's PC and GPU architectures and their limitations regarding the visualization of large-scale particle data sets. Such particle sets can contain hundreds of GB of data, which by far exceeds the capacities of the main memory and the local video memory. Thus, it is necessary to stream the visible portions of the data from secondary storage to the video memory during the visual exploration process. This streaming is affected by hardware restrictions including bandwidth limitations and latency issues, which must be considered in the data management. Section 2.1 reviews these constraints within the involved memory hierarchy. Furthermore, interactive visualization of large-scale data sets poses a challenge for even the most recent generations of GPUs due to the enormous geometry and rasterization load that results from rendering millions of particles. To minimize this workload it is important to effectively exploit the capabilities of the GPU's streaming architecture and their programmability. Section 2.2 gives an introduction to the programmable rendering pipeline as exposed through the Direct3D 10 API. Both sections should help readers that are not entirely familiar with the according architectures in understanding the algorithm and design choices of the subsequent chapters.

2.1 Memory Hierarchy of Desktop Computers

Today, high-performance visualization on desktop computers is not possible without graphics hardware that has its own dedicated video memory. On such systems the memory hierarchy that needs to be considered for the visualization of large-scale data sets contains three levels (see Figure 2.1). At the bottom of the hierarchy are secondary storage devices, which are used to permanently store the data sets. For rendering, the visible part of the data set needs to be transferred to the GPU's video memory at the top

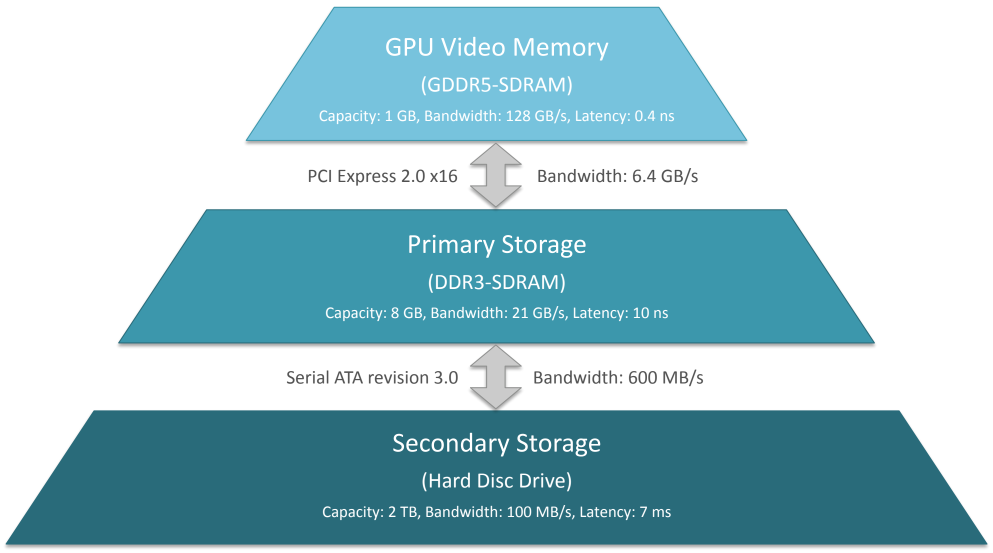


Figure 2.1: *Illustration of the memory hierarchy that is relevant for the interactive visualization of large-scale data sets on desktop computers. Memory capacities, latencies, and bandwidth limitations are given for an exemplary middle-class PC equipped with a single 2 TB hard disc, a Dual-Channel DDR3 SDRAM at 1333 MHz and an NVIDIA GeForce 560 Ti graphics card.*

of the memory hierarchy. Located between these layers is the main memory, to which the data needs to be loaded before sending it to the GPU. Each memory level is limited in capacity and its data access is restricted to the maximum transfer rate and by the latency, which is the time delay between initializing a data transfer and the actual start of the transmission. Subsequently, we review these constraints for the distinct memory levels as well as the throughput limitations of the connecting transmission channels.

Secondary Storage

Hard disk drives (HDD) are most commonly used for secondary storage on desktop computers. Typically, PCs can be equipped with multiple internal plus additional external hard drives, with each possessing a capacity of up to 3 TB (as of 2011). HDDs store data on radial tracks of magnetic disks, which rotate at high speed with up to 15,000 rpm. The data is read and written using a head that needs to be moved to access the individual tracks. Due to mechanical limitations of the disc rotation and head movement, HDDs with large capacities have usually a latency between 4 and 9 ms, which is a very long time period in modern computing. The data transfer rate depends on the rotational speed of the disks and additionally varies according to the position of the track to be read, but the sustained transfer rate of large HDDs is typically around 100 MB/s. This transfer rate can be increased using a *Redundant Array of Independent Disks* (RAID) in which multiple HDDs are combined into a logical unit. By reading data from several hard discs in parallel, for example via block-level striping (RAID 0) or mirroring (RAID 1), the data throughput can be increased up to the same factor as number of disks are combined.

With their increasing size, solid-state drives (SSD) have rapidly gained market share in the recent years due to some advantages over classical HDDs. Most importantly, SSDs posses a much lower latency (< 0.1 ms) and a considerably better transfer rate (up to 500 MB/s) since the data is stored on non-volatile memory chips exclusive of any mechanical parts. However, due to the restricted size of up to 600 GB and significantly higher costs (> 1 Euro per GB), they are currently no general alternative for the storage of large-scale data sets.

Besides internal and external hard drives, data can also be read from other computers or dedicated *network attached storage* (NAS) devices within local area networks, e.g. via Gigabit Ethernet that transfers data at a rate of one gigabit per second (= 125 MB/s). Thus, data can also be streamed via Ethernet at transfer rates that are comparable to local storage devices. This is especially beneficial for very large data sets exceeding the limit of local storage devices or if the data set needs to be accessed from multiple workstations.

Data transfer between secondary and primary storage

Data between secondary storage devices and main memory is transferred in units of *pages*, which contain 4096 Byte on most of today's systems. Accordingly, the maximum possible transfer rate of hard disks can only be achieved if the data of entire pages are required. Reading smaller amounts of data can significantly decrease the effective transfer rate and should therefore be avoided. For the data transfer to main memory, HDDs operate on high-speed serial interfaces such as Serial ATA (SATA), which has a transfer rate of 6 Gbit/s in the current standard (SATA revision 3.0). Since the clock signal is transmitted using the same pins based on a 8b/10b encoding, in which 8 bits of data are transferred using a 10-bit signal, the theoretical maximum data throughput is 600 MB/s. Accordingly, the transmission bandwidth lies clearly above the transfer rate of HDDs and represents no bottleneck for the memory hierarchy.

Primary Storage

Today, PCs are usually equipped with 4 to 16 GB of main memory, even though mainboards exist that support RAM with a size up to 128 GB. The most common memory interface is DDR3-SDRAM, which has a data access latency in the order of 10 ns. The transfer rate depends on the clock speed and on the number of channels of communication between main memory and the memory controller, but the theoretical limit is typically around 20 GB/s.

CPU-GPU memory transfer

From main memory, the data is transfered to the GPU's local video memory via the PCI Express (PCIe) bus. In the current standard (PCI Express 2.0 x16) the raw transfer rate is 8 GB/s in each direction. Like SATA, PCI Express employs a 8b/10b encoding scheme, which reduces the theoretical peak data throughput to 6.4 GB/s. In practice, however, the actual effective transfer does typically not exceed 2.5 GB/s.

GPU video memory

State of the art graphics cards usually have a dedicated video memory of size 1024 to 2560 MB for the consumer market and up to 6 GB for the high end market. GPU's possess a very high memory bandwidth of more than 100 GB/s. Memory latencies are below a nanosecond and are additionally hidden by sophisticated thread scheduling strategies.

2.2 Programmable Graphics Hardware

The primary purpose of GPUs is to support rasterization-based rendering [AMH02] that transfers three-dimensional geometric primitives such as point, lines, or triangles into a two-dimensional raster image. Since the rasterization process can be divided into a series of steps which need to be performed for up to millions of primitives, GPUs are designed as massively parallel processors based on the stream processing paradigm. In a streaming architecture, a large ordered set of data (referred to as *stream*) is processed by running the same series of operations (referred to as *kernel*) on each item. Usually, each kernel performs only a small set of operations on the stream whereas complex tasks are accomplished by serializing multiple kernels in a fixed order with the output of one kernel being the input of the subsequent one. Since the output of a kernel is defined to depend solely on the input, kernel computations can be efficiently parallelized by using processors that are based on the *Single Instruction Multiple Data* (SIMD) and *Single Instruction Multiple Threads* (SIMT)[1] paradigms comprising multiple processing elements that perform the same operations on distinct data simultaneously.

The first graphics cards that included the entire rasterization pipeline were presented in 1999. These GPUs implemented the pipeline in a fixed-function architecture with some kernels being configurable to a small extent. Driven by the demand of more realistic 3D graphics in computer games, both the computational power as well as the level of programmability was steadily increased since then. The kernels performing the vertex and fragment processing became freely programmable, new kernels such as the geometry stage were introduced, and with the introduction of *unified shaders* the programming of and the data access within the kernels were unified.

In addition, GPUs are no longer pure graphics devices but have evolved to parallel processors for general purpose computing (referred to as GPGPU[2]), that are freely programmable in C-like programming languages such as NVIDIA's CUDA, Microsoft's DirectCompute, or OpenCL, which is developed by the Khronos Group. For graphics programming, two cross-platform APIs can be used for writing graphics applications that are supported by state of the art GPUs: DirectX of Microsoft and OpenGL, which was initially developed by Silicon Graphics Inc. (SGI) and which is now managed by the Khronos Group. Both APIs follow a very similar rendering pipeline, in which shaders are programmable in a C-like shading language such as HLSL for DirectX and GLSL for OpenGL.

In the following sections, we will introduce the fundamental concepts of the rendering pipeline based on the Shader Model 4.0, which is the minimum programming model required for the implementation of the techniques presented in this thesis. For consistency reasons we will describe the pipeline according to the design and terminology of the DirectX 10 API, which was used for the validation of all algorithms within this thesis. It should be noted, however, that all implementations are directly transferable to OpenGL.

2.2.1 Data Structures

On GPUs, data can be stored in the dedicated video memory using several kinds of data structures, which differ in their representation, data access, and usage. The most important data structures are:

[1]The term SIMT was introduced by NVIDIA, but modern GPUs of other vendors build upon very similar architectures.

[2]General Purpose Computation on Graphics Processing Units

- **Vertex Buffers:** Vertex buffers are used to store attributes that are given per vertex for a geometric object or per point for singular primitives such as SPH particles. Each attribute is stored as a tupel with up to four components. Typical attributes for vertices are the position, colors, normals, and texture coordinates. In case of SPH rendering, the attributes store the quantities of each particle such as its position, velocity, smoothing length, and mass.

- **Index Buffer:** Instead of rendering the vertices in the order given by the vertex buffer, an index buffer can be used to define the order of processing. The index buffer stores references into a vertex buffer as a list of integers. This indirection is especially useful for meshes to avoid storing duplicates of vertices that are shared by multiple polygons. Furthermore, it can be used to define a subset of vertices to be drawn instead of the whole vertex list.

- **Texture Resources:** Texture resources (or short *textures*) are one-, two- or three-dimensional arrays of elements that can consist of up to four components. The texture elements are called *texels* (short for texture elements) in 1D and 2D, and *voxels* (derived from volume elements) in 3D. When reading from textures *samplers* can be used to filter the data, e.g. via linear interpolation. In addition, *mipmaps* are extensions of 1D or 2D textures that augment the original texture by a pyramid of filtered versions of reduced size.

2.2.2 The Rendering Pipeline

Figure 2.2 depicts the rendering pipeline according to Direct3D 10 and the Shader Model 4.0. The whole pipeline contains several stages that are assembled in a fixed order. Stages marked as blue are configurable based on specific state objects and stages marked as red are fully programmable. In the following, we will briefly describe the individual stages. For more detailed information, we refer the reader to [Bly06] and [Mic10].

- **Input Assembler Stages:** The first stage of the rendering pipeline is the input assembler, which creates the initial vertices of the primitives that are sent through the pipeline. Vertex attributes can be read from one or more vertex buffers. Via the *primitive topology* the programmer defines if the individual vertices should be interpreted as individual points, lines, or triangles, or as strips of the latter two primitive types. The vertex buffers can either be read sequentially, or an index buffer can be bound that specifies the order of the vertices to be created.

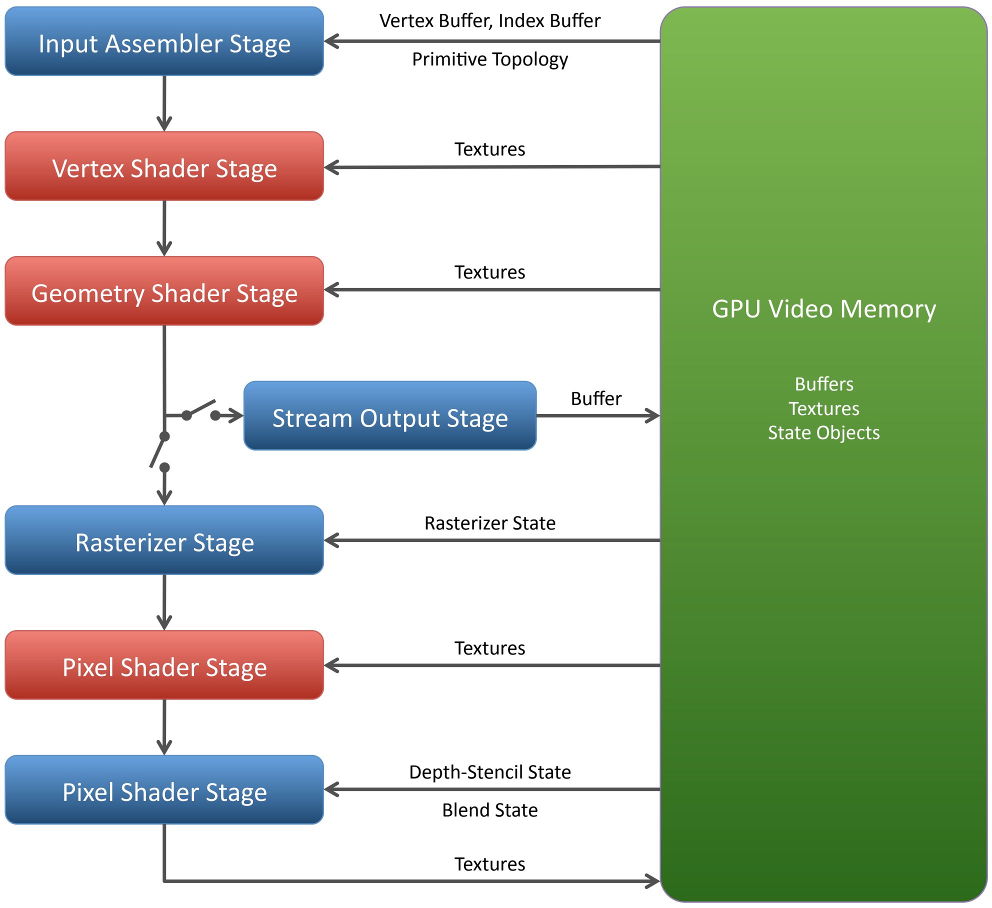


Figure 2.2: *Illustration of the graphics pipeline.*

Furthermore, *instanced rendering* can be used to replicate each primitive a fixed number of times. In addition to the user-given vertex attributes, the assembled primitive data can be augmented by system-generated properties including vertex, primitive, and instance IDs.

- **Vertex Shader Stage:** Each assembled vertex is send to the programmable vertex shader stage, which operates on them by modifying, adding, or removing vertex attributes. Typical modifications include linear transformations of the vertex position from object space into the view or clip space. Furthermore, textures can be accessed to alter existing or to add further attributes.

- **Geometry Shader (optional):** The geometry shader operates on individual primitives (i.e. points, line segments, or triangles) and has access to all of its vertices. It can produce none, one, or multiple primitives as output. Hence, existing primitives can be removed, its vertices can be manipulated, or additional primitives can be generated. Furthermore, the primitive type of the output stream can differ from the one of the input stream. This can be used to expand point primitives to triangles that are used as textured sprites or to extrude line segments to tubes. In cases where the render target is a volumetric texture or a texture array, the programmer can additionally specify the target slice into which each primitive is rendered. Geometry shaders are optional and should be avoided unless their functionality is required or particularly beneficial, since their usage causes performance penalties.

- **Stream Output Stage:** Subsequent to the geometry shader, vertex information of the data stream can be written into a buffer of the GPU's video memory. This buffer can be used as vertex buffer or texture resource for a subsequent render pass, or its data can be read back into the main memory to extract information required by the application. The stream output can be used as an alternative to streaming the data to the rasterization stage or in parallel to it.

- **Rasterization Stage:** In the rasterization stage, all primitives are clipped against the view frustum and all vertex positions are transformed from homogeneous clip space into screen space, i.e. coordinates are now given according to the viewport on the screen. Afterwards, either for the whole polygon (*solid fill mode*) or its edges (*wireframe mode*) so called *fragments* are generated for all covered pixels. For each of these fragments, an interpolation unit interpolates all attributes that are demanded by the subsequent pixel shader stage according to a specified interpolation scheme, which in most cases is linear interpolation with perspective correction.

- **Pixel Shader Stage:** The pixel shader is evoked for each fragment generated by the rasterizer and is used to determine the fragment's color or other values that are written into the render target. This stage operates on the interpolated vertex attributes and has access to all texture resources in GPU memory. Typical per-pixel computations are texturing and lighting. In addition, the depth of each fragment can be modified or the fragment can be discarded from further processing.

- **Output Merger Stage:** The output merger stage merges the fragment's values with those stored at the same location in the render target. It can be configured via two state objects: the *depth-stencil state* and the *blend state*. For opaque geometry, the depth-stencil state is typically configured such that the depth value of the fragment must be smaller than the current depth of the regarding pixel. Otherwise the fragment is discarded. During rendering the writing of depth values can either be enabled (i.e. the values in the depth buffer are automatically updated to the closest fragment) or disabled (e.g. for the rendering of transparent fragments). If stencil operations are discarded and the depth value is not modified in the pixel shader stage, the depth test can be moved forward to the end of the rasterization stage, which is called *early-z test*. By this means, the discarded fragments need not to be processed in the pixel shader stage, which can lead to a significant performance gain due to the reduced fragment load. The blend state, on the other hand, specifies how fragments are blended with the existing pixels in the render target. Typical operations are to overwrite the value, to add the new value, or to composite both values by using alpha blending (see Section 3.5.2). The render target can either be the frame buffer that is displayed on the screen or an offscreen buffer, which can be used as input for a subsequent render pass or which is read back to main memory to pass information to the application.

Chapter 3

Background

This chapter provides an overview on methods that are used in the subsequent chapters. We introduce the visualization pipeline as a paradigm for the scientific visualization of data and explain the general workflow for the rendering of large-scale SPH data sets. With respect to the order within the pipeline, we introduce SPH (see Section 3.2), principal component analysis including its use for cluster analysis (see Section 3.3), describe data compression based on vector quantization (see Section 3.4), and review relevant issues regarding direct volume rendering (see Section 3.5).

3.1 The Visualization Pipeline

The visualization pipeline is a conceptual description of the processing of data from its acquisition to a visual representation. Most commonly, the pipeline is divided into four stages (depicted in red in Figure 3.1). The visualization process starts with the collection of data. In scientific visualizations, the two most important data sources are sensors and numerical simulations. In our application the data originates from SPH simulations, which is a Lagrangian method to simulate fluid and gas dynamic effects. Section 3.2 gives an introduction to SPH and discusses all necessary aspects regarding its visualization.

The kind of raw data resulting from data acquisition depends on the acquisition method. In case of SPH, the simulated fluid is defined by a set of discrete particles and their attributes. For dynamic data sets the particles are stored at discrete time steps. To allow interpolation between these time steps, the time-to-time coherence of SPH data is usually stored implicitly given by the order of the particles or explicitly based on particle IDs.

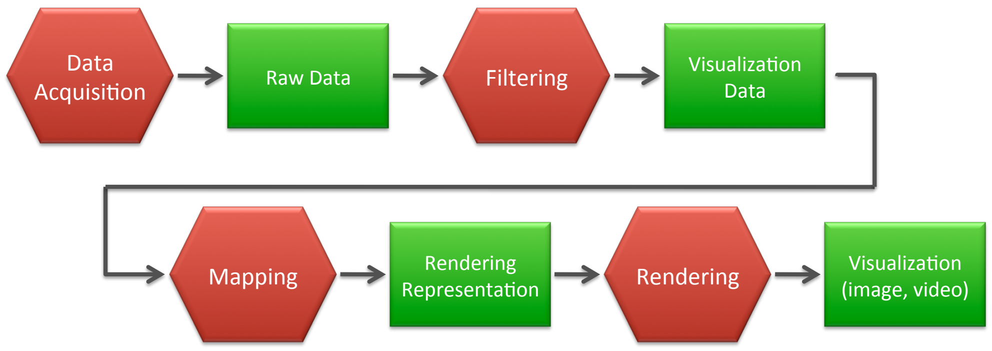


Figure 3.1: *The visualization pipeline: Beginning with the acquisition, the data (green) is processed in four consecutive processing steps (red) to obtain a final visual representation.*

The second step in the visualization pipeline is the filtering stage. It reduces the acquired information and outputs only that data that is required for visualization. This includes methods that are applied in a preprocess in order to save computational effort during runtime, for example interpolations and approximations of the data (e.g. for the generation of a multi-resolution representation) and data format conversion including data compression. In case of SPH, a multi-resolution hierarchy can be created based on cluster analysis algorithms. One such approach is built upon principal component analysis, which is described in Section 3.3. A brief review of vector quantization for the purpose of data compression is presented in Section 3.4. The filtering stage also comprises parts that need to be performed during rendering, as they depend on camera position and user settings. This includes clipping and culling techniques to discard data that is either not visible or should explicitly excluded from visualization.

For particle data, the output of the filtering stage is a set of particles to be rendered for a specific camera position and specific viewing properties. Next, the data has to be mapped to a graphical representation that can be visualized by modern graphic devices. This is performed during the mapping stage. SPH particles, for example, may be mapped to points or textured sprites, and their trajectories can be rendered as lines or tubes. With respect to the continuous scalar field that is defined by SPH (see Section 3.2.1), a more sophisticated approach is to reconstruct a volumetric data representation of this field that can be visualized using direct volume rendering (see Section 3.5). As part of the mapping stage, color and opacity of the primitives or the volumetric representation must be determined, which is usually done via a transfer function based on the physical quantities of the scientific data. Finally, the rendering stage projects and composes the graphical primitives to an image that is presented to the user. Depending on the visualization approach, the processing of subsequent stages can be tightly linked.

A prominent example is the close relation between the mapping and rendering stage in direct volume ray casting.

3.2 Smoothed Particle Hydrodynamics (SPH)

This section reviews the fundamentals of SPH and briefly describes aspects that are related to visualization purposes. For a more detailed overview, especially on simulation issues, we refer to the comprehensible summary papers by Monaghan [Mon92, Mon05] and the textbook by Liu and Liu [LL03].

3.2.1 Theoretical Foundations

At the core of SPH is the integral interpolant that defines the interpolation of any quantity A_I at a position $\mathbf{r}$ based on the values of A at all data points within a certain domain of interest:

$$A_I(\mathbf{r}) = \int A(\mathbf{r}')W(\mathbf{r} - \mathbf{r}', h)d\mathbf{r}', \tag{3.1}$$

where $d\mathbf{r}'$ is a differential volume element and W is the *smoothing kernel*, which has an area of influence that is defined by the smoothing length h. The kernel function is defined to be normalized to unity, i.e.

$$\int W(\mathbf{r} - \mathbf{r}', h)d\mathbf{r}' = 1,$$

and the limit of W should be the Dirac delta function δ as h approaches zero

$$\lim_{h\to 0} W(\mathbf{r} - \mathbf{r}', h)d\mathbf{r}' = \delta(\mathbf{r} - \mathbf{r}') = \begin{cases} \infty & \mathbf{r} = \mathbf{r}' \\ 0 & \mathbf{r} \neq \mathbf{r}' \end{cases}.$$

Please note that the kernel function is typically radially symmetric in practice. For this reason, it is frequently written in the form $W(|\mathbf{r} - \mathbf{r}'|, h)$ in literature.

If the fluid is discretized into a set of small mass elements, the integral in Equation 3.1 is approximated by a summation over these particles and the differential volume is replaced by the finite volume of the individual elements V_b

$$A_I(\mathbf{r}) \approx A_S(\mathbf{r}) = \sum_b A_b W(\mathbf{r} - \mathbf{r_b}, h_b)V_b,$$

where A_b is the particle's value of quantity A, r_b is the particle's position, and h_b is the particle's smoothing length. Since density is defined as mass per unit volume ($\rho_b = m_b/V_b$), we can transform the previous equation to obtain the summation interpolant

$$A_S(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} W(\mathbf{r} - \mathbf{r_b}, h_b). \tag{3.2}$$

As can be seen, the interpolation for any quantity A is defined as the weighted average over the particles, where the influence of each particle depends on its mass, density, and the weight according to the kernel function, which itself depends on the distance of the particle and its kernel size. For the case that the interpolant is used to estimate the density at any point $\mathbf{r}$, Equation 3.2 simplifies to

$$\rho(\mathbf{r}) = \sum_b m_b W(\mathbf{r} - \mathbf{r_b}, h_b),$$

which illustrates that the mass of each particle is smoothed over the particle's area of influence according to the kernel function. Besides the interpolation of quantities, the summation interpolant can also be used to analytically obtain their spatial derivative given a differentiable kernel function:

$$\nabla A_s(\mathbf{r}) = \sum_b m_b \frac{A_b}{\rho_b} \nabla W(\mathbf{r} - \mathbf{r_b}, h_b) \tag{3.3}$$

Generally the smoothing kernel can have infinite support. In practice, however, a compact support is used, i.e.

$$W(\mathbf{r} - \mathbf{r}', h) = 0 \quad \text{if} \quad ||\mathbf{r} - \mathbf{r}'|| > \kappa h, \tag{3.4}$$

where κ is a constant defining the limit of a particle's spatial influence as a factor of the smoothing length. For kernels with compact support the sum of Equation 3.2 can be restricted to those particles that cover the sampling point $\mathbf{r}$ with their smoothing kernel. This can lead to enormous computational savings, provided that the participating particles can be efficiently determined.

Especially for SPH simulations that contain large density inhomogeneities as in many cosmological simulations, the smoothing length of individual particles is often adapted in space and time. Typically h_b is chosen such that each kernel contains a constant mass for the estimated density or such that the number of particles inside this volume remains relatively constant. Within high density regions the adaption of the smoothing length significantly increases the local resolution of the simulation and additionally decreases the computational costs for interpolation since less particles contribute to a sampling point.

Based on the SPH formulation the governing equations for the motion of hydrodynamics, which are a given in a system of partial differential equations, can be transformed into a set of ordinary differential equations, which can be solved via time integration. In addition, gravitational algorithms that are known from N-body simulations can be efficiently integrated to simulate the self-gravity in cosmological applications [HK89]. Details on the simulation code that has been used to produce the astrophysical data sets presented in this thesis can be found in [Spr05]. The two-way coupling of fluids and rigid-bodies as shown in the fluid dynamics data sets that are used in our experimental results is described in [BTT09].

3.2.2 Smoothing Kernels

Depending on the area of application and the fluid to be simulated, different smoothing kernels are used. In the remainder of this thesis, we will present visualization results for two different kinds of data sets: astrophysical simulations and fluid dynamics. In all astrophysical simulations the *cubic spline*-kernel was used for particles with varying smoothing length h. Writing $q = |\mathbf{r}| / h$, this kernel is defined as

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

It has the gradient

$$\nabla W_{spline}(\mathbf{r}, h) = \frac{\mathbf{r}}{|\mathbf{r}|} \cdot \frac{48}{\pi h^4} \begin{cases} -2q + 3q^2, & 0 < q \le \frac{1}{2} \\ -\left(1-q\right)^2, & \frac{1}{2} \le q \le 1 \\ 0, & q > 1 \end{cases}$$

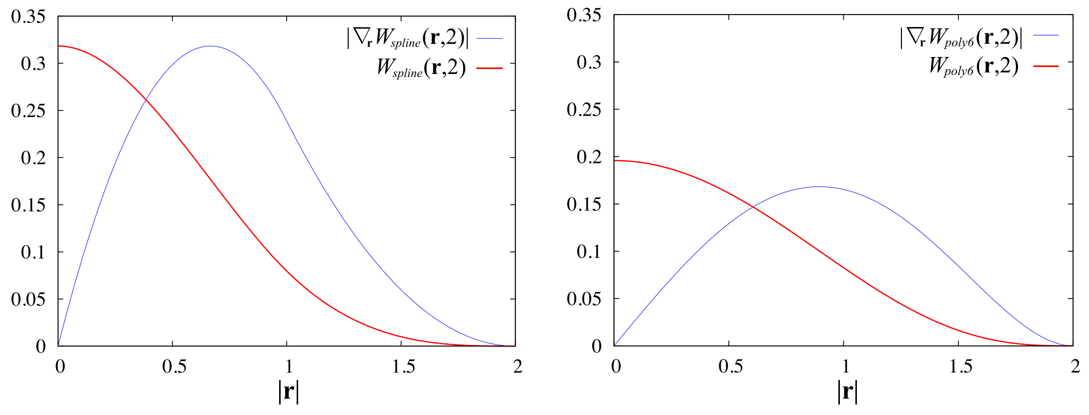
for $|\mathbf{r}| > 0$ and zero otherwise. For all fluid dynamics simulations the *poly6*-kernel was applied:

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

where $r = |\mathbf{r}|$. Its gradient is given by

$$\nabla W_{poly6}(\mathbf{r}, h) = -\mathbf{r} \cdot \frac{945}{32\pi h^9} \begin{cases} (h^2 - r^2)^2, & 0 \leq r \leq h \\ 0, & \text{otherwise} \end{cases}$$

Figure 3.2 shows both kernels and their gradients for a smoothing length of $h = 2$. In the given formulas, both kernels are defined such that $\kappa = 1$ (see Formula 3.4), i.e. that a particle has no influence for a distance greater than their smoothing length.



(a) Spline-Kernel (b) Poly6-Kernel

Figure 3.2: *Plots of the kernel functions (thick red lines) and the absolute values of their directional derivatives with respect to* **r** *(thin blue lines) for kernels with a smoothing length of* $h = 2$.

3.3 Principal Component Analysis

Principal component analysis (PCA) [Pea01], also known as Hotelling transform [Hot33] or Karhunen-Loève transformation [Kar47, Loè46], is a linear transformation of a set of possibly correlated variables to a set of uncorrelated variables, which are called *principal components*. These principal components are defined such that the first component represents the direction of maximum variance within the data, and all further components correspond to the remaining maximum variance that is orthogonal to all previous ones.

One application of PCA is hierarchical cluster analysis [Jol02]. Here, PCA can be used to steer an divisive algorithm that iteratively splits clusters into two subclusters according to the direction of maximum variance. The splitting plane is defined by the centroid of the cluster and the first principal component as normal. This binary division is repeated for the subcluster with the highest variance until a stop criterion is met. Usually this criterion is either a certain number of subclusters that has to be created or a maximum distortion threshold allowed in the resulting partitions.

More formally, let $x_i \in X \subset \mathbb{R}^n$ be a data set of n-dimensional variables and let PQ be a priority queue that stores mutually exclusive subsets $P_j \subseteq X$ with the priority given by the subsets distortion D_j. For our applications, we measure the distortion as the euclidean distance to the centroid of the subset:

$$D_j = \frac{1}{|P_j|} \sum_{x \in P_j} (x - \bar{x}_j)^2 , \tag{3.5}$$

where the subset's centroid $\bar{x}_j$ is defined as

$$\bar{x}_j = \frac{1}{|P_j|} \sum_{x \in P_j} x.$$

Given a minimal number of subclusters n or a maximum distortion threshold ρ as stop criterion, the partitioning is performed as follows:

1. Add X as single partition into PQ
2. Remove the partition P_j with the largest distortion from the priority queue.
3. Calculate the covariance matrix:
$$C = \sum_{x \in P_j} (x - \bar{x}_j)(x - \bar{x}_j)^T.$$
4. Calculate the maximum principal component, which is the eigenvector e_{max} of the absolute largest eigenvalue λ_{max} of C.
5. Assign all elements of $x \in P_j$ to one of the new groups P_k and P_l according to:
$$P_k = \{x \in P_j |\, \langle (x - \bar{x}_j) , e_{max} \rangle \leq 0\}$$
$$P_l = \{x \in P_j |\, \langle (x - \bar{x}_j) , e_{max} \rangle > 0\}$$
6. Calculate the residual distortions of P_k and P_l and add these partitions into the priority queue.
7. Proceed with step 2 if the size of PQ is smaller than n or if the distortion of the top of PQ is greater than ρ.

Since the covariance matrix C is real, symmetric, and positive definite we determine the greatest eigenvalue/eigenvector pair in step 4 by performing the Housholder reduction to reduce C to a tridiagonal form, followed by applying the QL algorithm with implicit shifts [PVTF02].

3.4 Vector Quantization

Vector quantization is a lossy data compression technique that maps a set of vectors $x = \{x_1, x_2, ..., x_n\}$ to a smaller set of N vectors called *code vectors* or *code words*, which as a whole build a codebook $C = \{y_i | i = 1, ..., N\}$. Typically, x and y_i are defined in $\mathbb{R}^n$. During encoding a quantizer maps a vector into an index of the codebook by determining the entry that leads to the smallest error

$$q(x) = i, \quad \delta(x, y_i) \leq \delta(x, y_j) \;\; \forall\, j \in [1, N],$$

where $\delta(x, y_j)$ is some kind of distortion measure, e.g. the squared Euclidean distance (see Equation 3.5 on page 25). The dequantizer, on the other hand, replaces the entry by the according codeword, i.e. $d(i) = y_i$. Vector quantization is computationally asymmetric, since the encoding, which requires the building of a codebook or at least finding the best representing code word, is much more complex than the decoding, which can be implemented via a single fetch into a lookup table.

The average distortion induced by the vector quantization for an input set X is given by

$$D = \frac{1}{|X|} \sum_{x \in X} \delta(d\left(q\left(x\right)\right), x).$$

If a codebook is not given a priori, finding an optimal codebook for a given input set is a non-linear data fitting problem. The most common approach to design a codebook is the Linde-Buzo-Gray (LBG) algorithm [LBG80], which is a generalization of Lloyd's algorithm [Llo82] that is also known as Voronoi relaxation in the context of centroidal Voronoi diagrams. The LBG-algorithm requires an initial codebook C and a training set X, for which the codebook is optimized. As stop criteria, a maximal number of iterations k_{max} and a minimal relative improvement ϵ of the average distortion with respect to the previous iteration can be defined. Then the algorithm proceeds as follows:

1. Initialize $k = 0$ and $D_{prev} = \infty$
2. Divide X into N disjunct groups such that each group S_i contains those vectors

of the training set that are closest to the code word y_i:

$$S_i = \{x \mid d(x, y_i) \leq d(x, y_j) \wedge i < j \quad \forall\, j \in C\}$$

3. Calculate the average distortion of the training set with regard to the current codebook:
$$D_{curr} = \frac{1}{|X|} \sum_{i=0}^{N} \sum_{x \in S_i} \delta(x, y_i)$$

4. If $\frac{D_{prev} - D_{curr}}{D_{curr}} < \epsilon$ or $k \geq k_{max}$, stop.

5. Set $D_{prev} = D_{curr}$ and replace each code word y_i by the centroid of the partition S_i:
$$y_i = \frac{1}{|S_i|} \sum_{x \in S_i} x$$

6. Increment k and go to step 2.

Due to the computational expense of each iteration step, which is dominated by the nearest neighbor search in step 2, and the fact that the LBG-algorithm converges towards a local minimum, the initial codebook has a strong influence on the runtime of the LBG algorithm and the quality of the resulting codebook. In our implementation we utilize the PCA-based binary split algorithm (see Section 3.3) to generate the initial codebook, which has proven to result in a high fidelity codebook by applying only a few LBG iterations within our experiments. Our approach builds upon the work of Schneider, who efficiently applied vector quantization in a wide range of visualization applications [SW03, KSW05, KSW06, JST$^+$10]. Detailed information on this topic can be found in the books of Sayood [Say00] and Gersho and Gray [GG91].

3.5 Volume Rendering

From the mathematical point of view, the SPH interpolation scheme defines a continuous three-dimensional field for each scalar quantity carried by the particles. The two most common options to visualize volumetric scalar fields are the rendering of isosurfaces and direct volume rendering. An isosurface is a three-dimensional representation of a surface that contains all points in space where a scalar value is equal to a certain *isovalue*. Such a surface is usually rendered either by extracting a polygonal representation, e.g. based on the Marching Cubes algorithm [LC87], or by casting rays through the volume for which the intersection with the isosurface is determined [Lev88, PMS+99]. Direct volume rendering, on the other hand, treats the volumetric scalar field as participating media interacting with rays of light. This requires the mapping of the scalar field to optical quantities such as color and opacity, which can be integrated along the line of sight. The subsequent sections give a brief introduction to volume rendering with focus on the aspects that are important for the remainder of this thesis. Thorough overviews on volume rendering can be found in the textbooks by Johnson and Hansen [JH04] and Engel et al. [EHK+06].

3.5.1 Volume Rendering Integral

Direct volume rendering is based on the physical theory describing the radiation field in a participating medium, which involves emission, absorption, and scattering of light. Due to its complexity scattering of light is typically neglected for volume rendering, leading to a simplified *emission-absorption model* [KvH84], which defines the light intensity along a line of sight of length L as

$$I = \int_0^L c(s)\ \tau(s)\ e^{-\int_0^s \tau(\hat{s})\ d\hat{s}} ds \tag{3.6}$$

where $\tau(s)$ is the opacity per unit length at position s, $c(s)$ is the light emission per unit length at s as non-associated color, i.e. $c(s)$ is not pre-multiplied by the opacity, and the exponential term represents the absorption between s and the eye. A numerical approximation of the equation of radiative transfer is commonly derived by calculating a Riemann sum for n segments of length $\Delta s = L/n$ and by approximating the exponential extinction coefficient by the first two elements of its Taylor series expansion:

$$\begin{aligned} I &\approx \sum_{i=0}^{n} c(i\Delta s)\ \tau(i\Delta s)\ \Delta s \prod_{j=0}^{i} e^{-\tau(j\Delta s)\ \Delta s} \\ &\approx \sum_{i=0}^{n} c(i\Delta s)\ \tau(i\Delta s)\ \Delta s \prod_{j=0}^{i} (1 - \tau(j\Delta s)\ \Delta s) \\ &= \sum_{i=0}^{n} c(i\Delta s)\ \alpha(i\Delta s) \prod_{j=0}^{i} (1 - \alpha(j\Delta s)). \end{aligned} \tag{3.7}$$

where $\alpha(i\Delta s) = \tau(i\Delta s)\ \Delta s$ is the opacity.

3.5.2 Compositing

For rendering purposes, the discretized volume rendering equation (Equation 3.7) is computed iteratively using a compositing scheme that is sequentially applied to the discretized segments. In this thesis, we apply front-to-back compositing using the under-operator [PD84]:

$$\begin{aligned} C_{i+1} &\leftarrow C_i + (1 - A_i)\, c_{i+1}\, \alpha_{i+1} \\ A_{i+1} &\leftarrow A_i + (1 - A_i)\, \alpha_{i+1}, \end{aligned}$$

where c and α are the color and opacity of the individual segments, and C and A hold the accumulated quantities along the viewing ray, initiated with $A_0 = 0$ and C_0 as black.

3.5.3 Opacity Correction

Instead of using equidistant segments it can be beneficial to locally adapt the sampling width for the discretization. In this case, the opacities for the segments need to be adjusted in order to result in the same degree of extinction per unit length. Since the overall extinction for a ray of length L through a homogeneous material has to be identical when using a segment width of Δt instead of Δs, we get the identity

$$(1 - \alpha_t)^{\frac{L}{\Delta t}} = (1 - \alpha_s)^{\frac{L}{\Delta s}}.$$

Solving for α_t yields the corrected opacity

$$\alpha_t = 1 - (1 - \alpha_s)^{\frac{\Delta t}{\Delta s}}.$$

for a segment of size Δt with an opacity α_s that is defined per unit length Δs.

3.5.4 Classification

For the visualization of physical quantities, a mapping to the optical properties emission and absorption is required. This process is termed classification and the mapping function itself is called transfer function. In direct volume rendering, classification is the most important instrument for visualizing and analyzing data since it defines which portions of a data set are visible and how the physical quantities are encoded in colors. Typically, the transfer function is defined via a user-interface and the changes are directly applied to the rendered image to allow an immediate feedback. Yet, the specification of an appropriate transfer function revealing the features of interest is often a challenging and time-consuming task [PLB$^+$01]. Flexibility and usability further depend on the number of quantities that are used to determine the optical properties. A one-dimensional transfer function determines this mapping as a function of a single scalar value. In two-dimensional transfer functions, the optical quantities are defined based on two different quantities. This allows more sophisticated mappings but is also more difficult to adjust due to the increase in degrees of freedom

During the work on this thesis, a simplified two-dimensional transfer function has proven to be effective and easy to adjust. In this function, the pixel's brightness is defined by the density and opacity and the hue is determined with regard to the temperature (or any other additional quantity), e.g. by mapping it to a rainbow color map. Due to their high dynamic range in the data sets, both physical quantities are scaled logarithmically prior to the mapping to enhance the differences of small values. A meaningful and user-friendly classification is already possible by a linear mapping of the two quantities within a user-defined value range (compare the images of the Millennium Run in Chapters 4 and 5). For more sophisticated visualizations, we have implemented a one-dimensional transfer function editor that allows the user to define the mapping of density to brightness and opacity as a piece-wise linear function, which can be configured based on control points. Such a transfer function has been used in the all volume renderings of the Merger and the Ejecta data sets of Chapters 5 and 6.

It should be noted that throughout this thesis, we have applied volume rendering based on post-classification, yielding high-quality visualizations with high-frequent transfer functions in our experiments as can be seen in Figure 5.15 on page 82. If required, however, pre-integrated volume rendering [Kra08] could be easily integrated into our rendering approach as well.

Chapter 4

Scalable Rendering of Large-Scale Cosmological Data Sets

In this chapter we investigate scalability limitations in the visualization of large-scale particle-based cosmological simulations, and we present methods to reduce these limitations on current PC architectures. To minimize the amount of data to be streamed from disk to the graphics subsystem, we propose a visually continuous level-of-detail (LOD) particle representation based on a hierarchical quantization scheme for particle coordinates and rules for generating coarse particle distributions. Given the maximum world space error per level, our LOD selection technique guarantees a specified sub-pixel screen space error during rendering. A brick-based page-tree allows to further reduce the number of disk seek operations to be performed. Additional particle quantities including density, velocity dispersion, and radius are compressed at no visible loss by using vector quantization of logarithmically encoded floating point values. Fine-grain view-frustum culling and presence acceleration in a geometry shader significantly reduce the required geometry throughput on the GPU. We validate the quality and scalability of our method by presenting visualizations of a particle-based cosmological dark matter simulation exceeding 10 billion elements.

4.1 Introduction

Particle-based cosmological simulations play an eminent role in reproducing the large-scale structure of the Universe. Today, the dark matter distribution has been numerically simulated with as much as billions of particles and the results of these simulations have to be analyzed for a better understanding of the structure formation process in the Universe.

One of the most popular simulations is the Millennium Run, where more than 10 billion particles have been used to trace the evolution of the matter distribution in a cubic region of the Universe with over 2 billion light-years on a side, resulting in terabytes of stored output. The simulation required 28 machine days at 0.2 Tflops using 1 Tbyte RAM. By comparing the simulation results to large observational surveys, the physical processes underlying the buildup of real galaxies and black holes can be clarified. Especially by analyzing the substructure in simulated halos and subhalos, the hope is to prove or contradict recent claims and hypotheses on the matter distribution in the Universe.

By today, for the largest available cosmological simulations an interactive visual analysis is not feasible. The exploration is mainly performed via database queries, data mining techniques, and visualizations of vastly reduced subsets of the full data set. However, it can be perceived that an even stronger awareness of the importance of interactive visualization techniques for data exploration is growing in the community. An important reason is that especially for cosmological data it is difficult to judge in advance which information in the vast amount of stored output is scientifically relevant. This makes it difficult to apply fully automatic analysis techniques, which typically need a precise specification of the structures to search for. In consequence, the community increasingly tries to put experts and their capabilities into the center of the exploration process, to support them by visualization tools that can effectively exploit the human's perceptual and cognitive abilities, and to let them interactively guide the search process.

To achieve interactive visualizations of cosmological particle simulations as large as the Millennium Run, we have to consider scalability with respect to the size of the data in the whole visualization process. Taking into account the massive amount of particles to be visualized, a major challenge in cosmological visualization today is to maintain interactive frame rates for data sets that vastly exceed the available main memory. Since the number of primitives to be sent to the GPU for rendering is significantly increasing as well, the visualization performance is strongly limited by bandwidth restrictions. In addition, even on recent high-end graphics accelerators, particle rendering induces a geometry throughput and rasterization load that considerably limits the performance.

4.2 Contribution

The primary focus of this chapter is to address scalability limitations in the visualization of very large cosmological particle simulations and to investigate how far one can reach on recent PC architectures by reducing these limitations. The methods we propose are based on encoding particle positions and attributes in a tree data structure. The data structure stores partitions of the underlying domain at increasing resolution, with the sub-domains in each partition being quantized uniformly using 8 bit per coordinate component. The partitions are recursively refined up to the spatial resolution of the simulation.

To reduce the number of particles at each resolution level, particles in a given quantization bin are merged if the rendering of the resulting particle at this level does not introduce a visible error. By this means, we build a level-of-detail representation that allows both reducing aliasing artifacts and the number of rendered primitives. In combination with a page-tree that stores a number of adjacent sub-domains in one contiguous memory block, memory bandwidth and the number of disk seek operations can be reduced considerably.

In addition to their coordinate, particles in cosmological simulations carry floating point attributes such as mass density, velocity dispersion, and radius, which significantly increase the data to be stored and transferred. To address this problem, we employ a vector quantization scheme after the attributes have been scaled logarithmically. Since the quantized data can be decoded efficiently on the GPU, we are able to trade CPU-GPU data transfer for GPU processing, resulting in three times faster streaming of large particle sets from disk into GPU memory.

Regardless of these efforts, geometry throughput and rasterization of particle splats on the GPU still remain critical bottlenecks for interactive visualizations. To further reduce this problem we propose techniques to discard particles from the rendering process as early as possible by fine-grain view-frustum culling and presence acceleration. The latter approach takes advantage of the fact that it is not necessary to render many particles in regions with small contributions to the final image. These problems as well as ideas for further optimizations are addressed in the discussion at the end of this chapter.

The remainder of this chapter is organized as follows: In the next section we review previous work that is related to ours. An overview of the Millennium Simulation and the visualization technique we employ are given in Sections 4.4 and 4.5. Subsequently, we present the components of the proposed scalable rendering pipeline and discuss our strategies for data representation (Section 4.7), memory management (Section 4.8), and rendering (Section 4.9). In Section 4.10 we analyze the different components of our method and the framework as a whole. We conclude the chapter with a discussion and summary of this chapter.

4.3 Related Work

Today, interactive data visualization is established as a key technology for the exploration of large and complex-structured data sets as they arise in almost all areas of science and engineering. There is a vast body of literature related to visual data analysis that we will not attempt to overview here, however [JH04, MJM$^+$06] discuss many related issues and provide useful references on these subjects.

Recent work on the visualization of astronomical data include the development of infrastructures for high-performance computing environments [BDG$^+$04, KCP$^+$02, ON97], volume rendering techniques for the visualization of star formation and planetary nebulae [LHM$^+$07, NJB07], GPU-assisted ray casting for hybrid data consisting of volumetric fields and point sets [KAH07], methods for visualizing uncertainty in large-scale astrophysical environments [LFLH07], and the comparison of different simulations by comparative visualization [AHH$^+$06].

Other approaches underline the importance of user-tailored visualization and interaction techniques in the field of astrophysics, including the development of scalable WIM (world-in-miniature) user interfaces that facilitate travel and wayfinding at large spatial scales [LFH06], and user-controlled interaction that supports multiple abstraction views of galaxy clusters and the selection of structures in spatial sub-domains [MQF06, RRE06].

For particle data from cosmological simulations, software rendering [DRGI08, Pri07, Hen08] is still most commonly used for visual data analysis. Efficient multi-resolution point splatting techniques based on spatial subdivision schemes have been proposed in [HE03, HLE04, SSL08]. However, the size of today's simulations demands scalable out-of-core rendering approaches, which support these visualization techniques by efficient data management and transfer [CE97]. Woodring et al. suggest to solve this problem via stratified random sampling of the particle distribution [WAF$^+$11].

To cope with this ever increasing amount of data to be visualized, parallel visualization systems [BDG$^+$04, KCP$^+$02, FCDM$^+$11], and infrastructures for visual data mining in distributed data sources [CBB$^+$05] have been proposed. These system include mechanisms for data handling, querying, and interaction.

4.4 The Millennium Run

The goal of the Millennium Simulation Project[1] is to investigate the evolution of the Universe based on a Lambda-Cold Dark Matter (ΛCDM) model of the accelerating expansion of the universe. For this purpose, the growth of dark matter structure was traced from redshift $z = 127$ to the present by using an N-body/SPH-simulation in which the collisionless dark matter fluid was represented by a set of discrete point particles. With a total number of $2160^3 \approx 10^{10}$ particles, distributed in a periodic box of 2.23 billion light years and with the accuracy of an effective resolution of 100.000^3, this is one of the largest particle-based simulations ever performed. For a detailed description of the numerical code used to compute the Millennium N-body simulation as well as the scientific results of the Millennium Project let us refer to the work by Springel et al. [Spr05, SWJ$^+$05].

In order to investigate the time-dependent dark matter distribution, the intermediate results of the simulation have been stored in 64 snapshots. In addition to its own 3D single-precision floating point coordinates, each particle in a snapshot carries additional physical quantities such as an adaptive smoothing radius, density, and velocity dispersion, each of which is encoded in one single-precision floating point value. For a single snapshot this results in a total amount of memory of 225 GB.

4.5 Cosmological Particle Visualization

The technique we propose visualizes the dark matter distribution in 3D space. Direct volume rendering based on the emission-absorption model is well suited to reveal the fuzzy structures of evolving dark matter clusters and filaments in the data set. This is performed by casting rays through the volume, which are sampled at discrete positions. However, in case of SPH data, for each sample point an SPH kernel interpolation needs to be performed. This interpolation depends on all particles that cover the point with their kernel (see Section 3.2). Unfortunately, even when using acceleration structures such as kD-trees, the extensive search operations required to access the contributing particles can by no means be fast enough to support interactive visualizations of a data set containing billions of particles.

For this reason, we follow a visualization approach for cosmological particle data that is commonly used for offline visualizations of large-scale data sets such as the Millennium Simulation [SWV$^+$08]. In contrast to classical volume ray casting, this

[1] http://www.mpa-garching.mpg.de/galform/virgo/millennium/

technique neglects the absorption term, so that the emission-absorption model reduces to a simplified emission-only model. In this visualization approach, the brightness b of each pixel depends on the squared dark matter density that is integrated along the lines of sight

$$P = \int \rho^2(t)dt, \tag{4.1}$$

and this integral is mapped logarithmically into brightness to increase the contrast of the image

$$b = \frac{\log(P/P_{min})}{\log(P_{max}/P_{min})},$$

where P_{min} and P_{max} are user-defined extreme values that define the mapping from black to full brightness. The color hue, on the other hand, is derived from the dark matter velocity dispersion, which is the spread of velocities of stars or galaxies in a cluster and that is weighted with the squared dark matter density and averaged along the lines of sight:

$$S = \frac{1}{P}\int \sigma(t)\rho^2(t)dt. \tag{4.2}$$

The resulting value is encoded into the color hue using a special transfer function that emphasizes regions of high dispersion. In analogy to the behavior of light, the density is adapted to fall off as the square of its distance to the viewer and an additional fall-off is typically applied to fade-out the density field beyond some depth of interest. Figure 4.1 shows some images of the Millennium Simulation at different scales, which have been generated by this approach.

The important benefit of this rendering approach is that due to neglecting the absorption term the intensities along a viewing ray can be accumulated independent of the depth order. Thus, instead of performing expensive kernel interpolations for each sampling point, the integrals can be evaluated on a per-particle basis. This can be efficiently implemented on GPUs by splatting the density footprint of the individual particles on the image plane.

It is obvious that without absorption, depth perception gets lost and can only be revealed via motion parallax when translating the camera. However, the order-independent integration benefits from the sparse nature of cosmological data sets that typically contain only a small number of clusters and filaments surrounded by large void regions. Hence, the emission-only model has been shown to produce visualizations that are similar to those generated by an emission-absorption model and that also allow for an effective visual analysis of the data. A comparison of the order-independent approach to high-quality volume rendering can be found in Chapter 5.

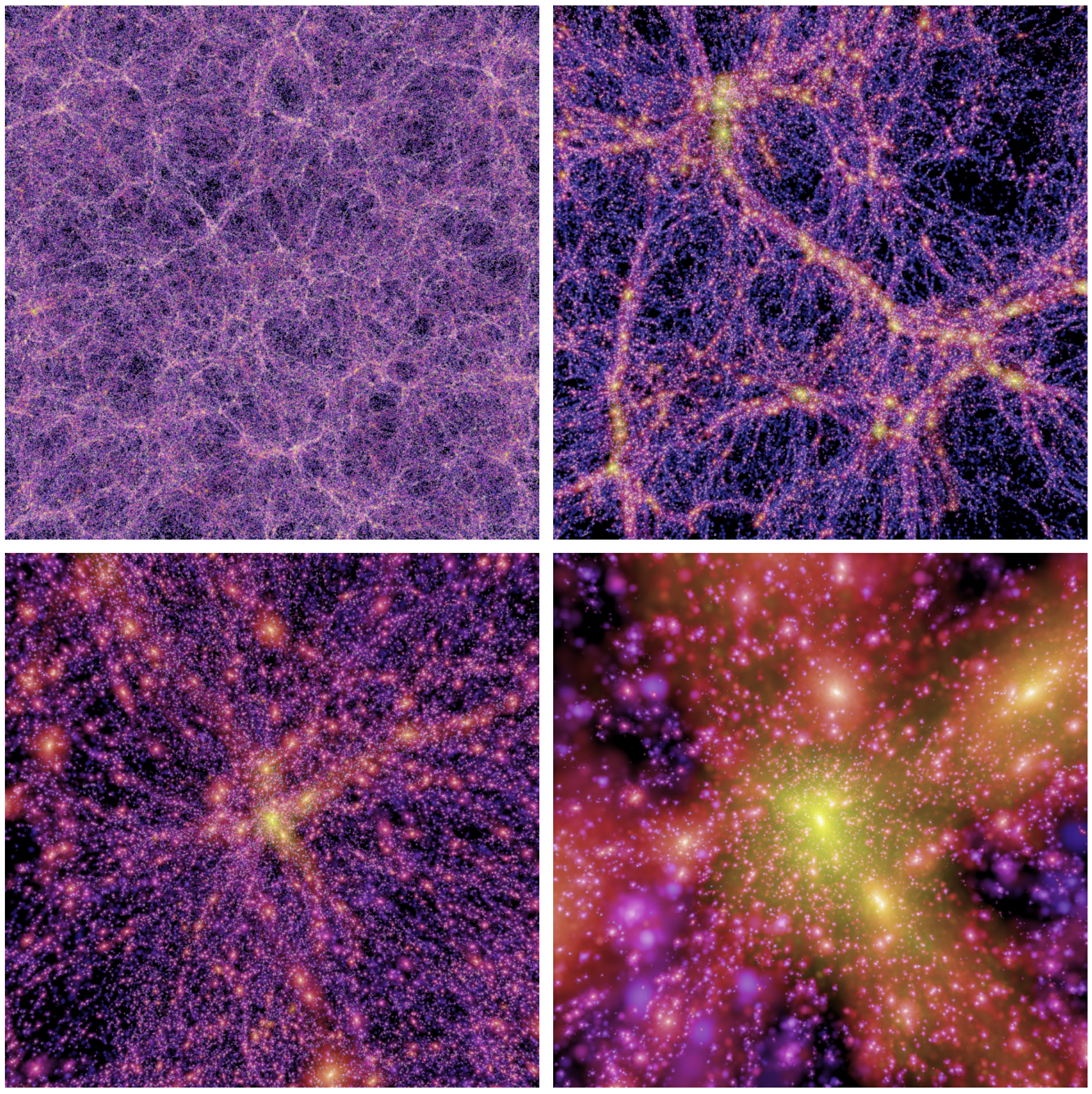


Figure 4.1: *Visualizations of the Millennium Simulation with more than 10 billion particles at different scales and a screen space error below one pixel. On a 1200x800 viewport the average frame rate is 11 fps.*

4.6 Challenges

Challenges of developing a cosmological visualization technique capable of dealing with data as large as the Millennium Simulation not only encompass the rendering. The key requirement of such a system is scalability throughout the entire visualization pipeline, including the streaming of the data from the hard disk to the graphics processor. The major constraints in this pipeline are memory and bandwidth limitations, disk access latency, and the geometry and rasterization throughput on current graphics hardware. According to these constraints, an efficient cosmological visualization technique has to be built on the following components:

- A data representation that minimizes the amount of data to be transferred and rendered while preserving visual quality.
- An efficient out-of-core and in-core data management to quickly select and access the relevant data.
- A rendering scheme that can effectively exploit the capabilities of current graphics accelerators to enable interactive visual data exploration.

4.7 Data Representation

Visualization techniques for large-scale data sets are usually built upon two major components: a multi-resolution representation of the data and an application-specific compression scheme. For the design of these components several aspects have to be considered. To preserve visual quality, it is of essential importance to know about the maximum error that is induced in each level of detail. Accordingly, there must be some kind of measurement or a conservative approximation of this error during the generation of the multi-resolution hierarchy. Furthermore, the compression scheme should not only yield a high compression ratio, but the decoding also needs to be efficiently implementable on GPUs to support a high rendering performance. Finally, our data representation should be designed such that it scales well for data sets that are even larger than the Millennium Run.

Due to these considerations, some common approaches are not applicable for our purpose. One example is hierarchical data encoding, which can yield very high compression ratios by storing the data of each resolution level relative to the next-coarser level. Accordingly, all lower resolution levels need to be accessed during rendering, for which a series of expensive dependent texture lookups needs to be performed. Hence, the performance heavily depends on the number of multi-resolution levels and scalability is not ensured.

Instead, our multi-resolution representation is stored in an octree-based data structure, that is the entire simulation domain is recursively divided into eight equally sized sub-domains. For each sub-space, the particles are discretized into regular grids of size 256^3. Each particle is placed at the center of its discretization bin. Within each of these cells, particles are merged if the difference is not perceivable during rendering (see Section 4.7.3). In the current application, we refine the octree up to the spatial resolution of the simulation, i.e. to an effective grid size of 128K^3, which requires 10 levels of detail. The finest level represents the full simulation with all individual particles. Due to the octree-based data structure, accessing the finest level of detail exhibits $\text{O}(\log_2(N))$ complexity, with N being the grid size corresponding to the maximum refinement level. The position of each particle within a node is quantized uniformly using 8 bit per coordinate component. Furthermore, additional particle attributes are compressed using vector quantization (see Section 4.7.4). In addition to a high compression ratio, we will show that no difference to uncompressed attributes can be perceived during rendering. Thus, the error of each level of detail can be solely derived from the quantization of the particle positions.

4.7.1 Multi-Resolution Hierarchy

The multi-resolution hierarchy is constructed bottom-up in a preprocess, i.e., the original particle data is stored in the nodes at the finest level of the tree. At coarser hierarchy levels, particles from the next finer level are either merged, copied, or omitted depending on their radius of influence and their density. The particular LOD-construction method we employ is described in Section 4.7.3. Figure 4.2 shows three nodes of the multi-resolution hierarchy, each of which represents a part of the region encoded in the previous node at increasing resolution.

As the entire data set with a size of 225 GB is much too large to be processed at once, we split the spatial domain into 8^3 blocks to fit the corresponding data into main memory. For each block we compute the respective subtree of the octree in parallel. Upon completion of this process all subtrees are merged into a single octree, which is then re-organized into a page tree to accommodate faster data access. The construction of this page tree is discussed in Section 4.8.1.

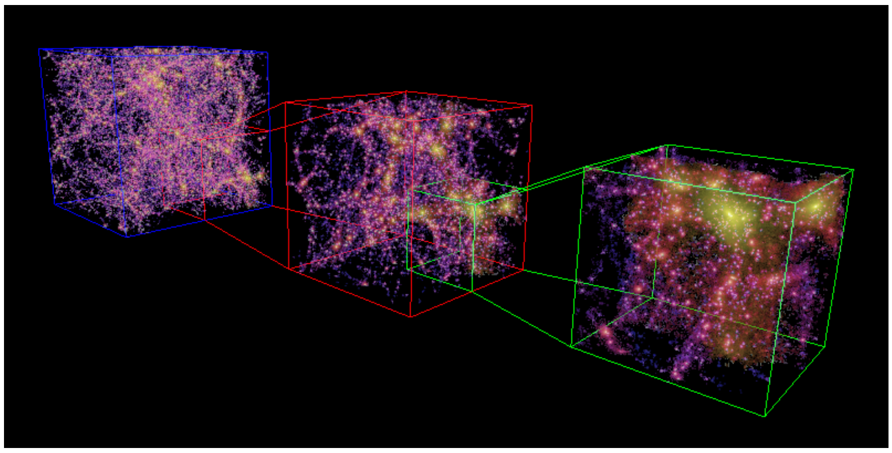


Figure 4.2: *Three nodes in the multi-resolution particle hierarchy with increasing resolution from from left to right.*

4.7.2 Level-of-Detail Selection

Given the size q of the quantization cells at a particular LOD, the maximum world space error that is introduced by quantizing particle coordinates to cell centers is $\delta = \sqrt{3} \cdot q/2$. During rendering a strategy is required to select those octree nodes that result in a sub-pixel error on the screen, and, thus, yield a visually continuous LOD rendering that is lossless wrt. the current image resolution.

Given the world space error δ, the vertical screen resolution res_y, and the vertical field of view $fovy$, we can determine the minimum distance $d_{\min}$ to the viewer as of which a node can be rendered before it has to be refined:

$$d_{\min} = \frac{\delta \cdot res_y}{2 \cdot \tan\left(fovy/2\right)} \cdot \tau.$$

Here, $\tau < 1$ is the maximum allowed screen-space error in pixels. To maintain this error during rendering, each node that is closer to the viewer than $d_{\min}$ has to be replaced by its 8 children.

4.7.3 Particle Thinning

The LOD-selection strategy tries to always render octree nodes for which the size of the projected cells is approximately equal or less than one pixel. Taking this into account, particles can be classified with respect to the number of pixels they would cover if they were rendered at a particular level: *points* cover at most one pixel and *sprites* cover more than one pixel.

Since all *points* in the same quantization bin will eventually fall into the same pixel on the screen, they can be merged to one single *point* (see Figure 4.3). To determine the dark matter density and velocity dispersion of this point wrt. to the individual contributions, the densities are accumulated according to Formula 4.1 and the velocity dispersion of the point is calculated as the weighted average with regard to Formula 4.2.

Sprites in one cell, on the other hand, can only be merged without visual loss if the differences of their radii h_i and h_j is not perceivable during rendering (see Figure 4.3). For this purpose, we use the fusion criterion $|h_i - h_j|/q < \epsilon$, where ϵ is a sufficiently small number. If this condition is not fulfilled, the particles are stored separately at the respective level. Note that due to the local coherence of smoothing lengths in a SPH simulation, almost identical radii within one cell are very likely to occur, particularly at finer resolution levels.

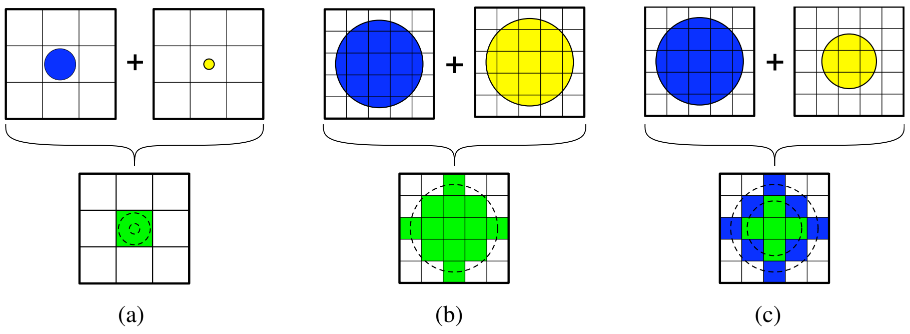


Figure 4.3: *Resolution-dependent particle merging: (a) Particles with a radius less than the cell size are merged into one single point primitive. (b) Particles covering more than one pixel can only be merged if their radii are equal. (c) Otherwise, they can not be combined in a meaningful way.*

In order to further decrease the number of particles per node, we introduce a rule for discarding particles depending on their density contribution. Subsequently, we refer to this neglection of particles as *presence acceleration*. Based on $d_{\min}$ and the quadratic density fall-off, the projected squared dark matter density of a particle on the screen can be computed as $P_i = \rho_i^2/d_{\min}^2$. Together with the minimum density for color coding $P_{\min}$, we discard a particle if $n \cdot P_i < P_{\min}$. Here, n can be interpreted as the maximum number of discarded particles whose sum of projected squared densities is smaller than $P_{\min}$. These particles do not have any influence on the final image.

The easiest way to find a reasonable value for n is to simulate the criterion on the GPU and to experiment with different values. As an objective criterion, the scene can be rendered with and without the threshold in order to compute the image difference.

This procedure can be repeated for a certain number of different views or while flying through the data set until a good threshold is found.

4.7.4 Attribute Compression

In addition to their coordinates, the particles stored in the multi-resolution representation carry floating point attributes such as dark matter density, velocity dispersion, and radius. Due to the nature of the simulation these properties are stochastically dependent. To reduce the memory required to store these attributes we employ a vector quantization scheme after the attributes have been scaled non-linearly.

As can be seen in the histogram in Figure 4.4, all three parameters have a high dynamic range. A standard vector quantizer, however, is designed to minimize the sum of squared errors. This is problematic because the same absolute distance between each data point and its respective quantization point is thus deemed acceptable by the vector quantizer.

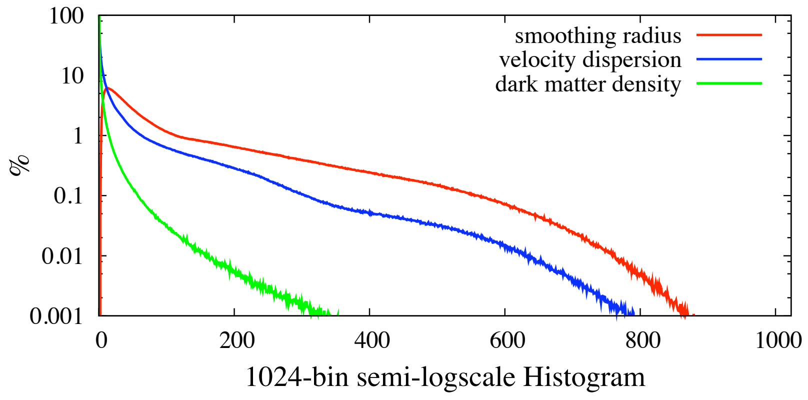


Figure 4.4: *Histogram of the three particle quantities. For each quantity, we split the range of values in 1024 equal intervals and count the number of occurences of a parameter value in each interval. Full ranges are* $[6.73 \cdot 10^{-47}, 2.11]$ *(smoothing radius),* $[8.80 \cdot 10^{-2}, 4.84 \cdot 10^{8}]$ *(dark matter density) and* $[2.26, 1.46 \cdot 10^{7}]$ *(velocity dispersion).*

Especially with regard to the logarithmic color coding of squared dark matter density and velocitiy dispersion during rendering, it is a much better choice for the quantizer design to minimize the relative error, because large absolute errors are not acceptable for small values. For exponentially distributed samples with a large number of small values and a few number of large values, vector quantization is known to lead to a very high sample distribution for large values, whereas the majority of small values is represented at a very poor resolution. It is apparent that quantizing logarithms of values reduces this imbalance and drastically decreases relative errors, albeit at the cost of increasing the sum of squared errors with respect to the original values. By logarithmizing (base 2) each component of the 3D data points prior to quantization we can reduce the maximum relative error from $3.63 \cdot 10^{39}\%$ to about 540% (average relative error from $3.63 \cdot 10^{39}\%$ to 13%) for an 8 Bit per 3D vector quantization. Note that this does not affect the coding efficiency, since it is sufficient to exponentiate the codebook entries prior to decoding. Also note that changing the base of the logarithm allows a further trade-off between minimization of squared errors and relative errors.

Figure 4.5 presents the resulting distortion induced by vector quantization using codebooks of varying size. As will be shown in the results, by using an 8 bit index with an average relative error of 13% we can already generate images which can hardly be distinguished from those that have been generated using the original data. By increasing the bit depth to 12 or 16 bit, the error can be reduced to 5% or 2% respectively.

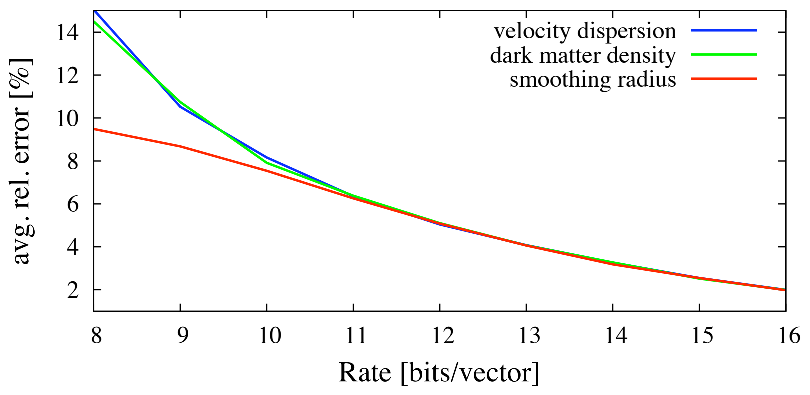


Figure 4.5: *Average relative error introduced by vector quantization of particle quantities with varying bit depth.*

Each level of detail has a different distribution of particle attributes. Accordingly, we generate a separate lookup table for each resolution level to improve fidelity. Since the quantized data can be decoded efficiently on the GPU by means of texture lookups, we can trade CPU-GPU data transfer for GPU processing, resulting in three times faster streaming. For details of the vector quantization we refer to Section 3.4.

4.8 Memory Management

In every frame, the octree is traversed in a top-down fashion on the CPU to determine the nodes that have to be rendered. Of these, those nodes that are not residing in main memory have to be read from disk. During tree traversal, the minimum distance criterion (see Section 4.7.2) is used to determine the LOD for the current view, and once a node has been determined the traversal in the current sub-tree can be stopped. In addition, view frustum culling is applied to discard those subtrees that are not visible.

Even though frame-to-frame coherence significantly reduces the amount of data to be read from disk, for very large data sets and high-resolution display systems there is still a vast amount of data that has to be streamed from disk to CPU memory in an interactive visual exploration session. In the following we describe a number of mechanisms to hide the data streaming from the user as far as possible.

4.8.1 Disk Transfer

To reduce the number of disk seek operations and thus to increase the data throughput from disk to main memory, in each octree level the nodes having the same parent are packed together and stored as a single data block on disk. We refer to these collections as *pages* and treat them as one entity in terms of storage and transfer. If a page size is below a given minimum size, additional nodes below those stored in the page are added in a breadth-first manner. Since a page typically contains more data than is required in a particular view, the minimum page size has to be chosen such as to yield a good balance between performance and main memory usage. In our experiments, a page size of 3 MB turned out to be a good trade-off. Figure 4.6 illustrates the relation between a tree data structure and pages in this data structure. As can be seen, the pages itself represent a tree data structure, which we refer to as *page tree*.

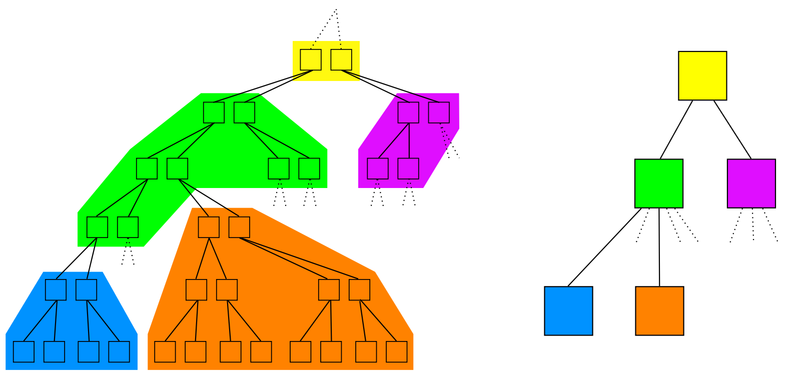


Figure 4.6: *Illustration of the relation between the tree data structure (left) and the page tree (right), which stores a number of nodes belonging to the same parent node.*

The page tree is stored on disk and dynamically loaded into main memory during runtime. To determine whether the data of a page is required for rendering, it is sufficient to check the distance criterion for the parent of the page since it has a smaller $d_{\min}$ than all of its descendants. According to the movements of the viewer, the page tree is dynamically built top-down and destroyed bottom-up within main memory, such that a maximum amount of data can be re-used in every frame. Simultaneously, the octree is built and destroyed by adding and removing the nodes contained in these pages. Note that nodes only store pointer to the respective data blocks in the page tree, so that the data is not stored twice in main memory.

4.8.2 Prefetching and Caching

For hiding disk access latency, we use asynchronous I/O combined with a prefetching scheme to load pages that might be required in upcoming frames. We load all nodes having a distance to the camera $d_{\mathrm{cam}} \leq d_{\mathrm{min}} + d_{\mathrm{prefetch}}$, resulting in a sphere-shaped prefetching region that supports arbitrary rotations. As has been shown by Ng et al. [NNT$^+$05], although the use of a sphere-shaped region increases the overall memory requirement, when moving along the line of sight the amount of data to be read from disk is only marginally increased. The prefetching scheme loads every page that might be required after a camera translation of length d_{prefetch}. In order to hide disk access latency, this distance is chosen such that the time for loading the largest page is less than the time for moving the camera to the position where the node is required for rendering.

When flying through the data set, there will eventually be pages, which are not required in main memory anymore because the corresponding pages have fallen outside the prefetching region. However, as long as enough main memory is available we do not remove the data but move it into a least-recently-used (LRU) cache. Hence, a page can be reused without any additional cost until the memory is required by another page.

4.9 Rendering

To render the particles stored in the set of selected octree nodes, they are copied into vertex array buffers on the GPU. These buffers are then rendered as described below. To optimize the resource management on the GPU, we avoid time-consuming creation and deletion of resources per node by using buddy memory allocation [Knu97]. This means that a set of vertex buffers of a size that it is sufficient for every node is allocated, and these buffers are split recursively into equally sized pieces if less memory is required. If the data in refined blocks is not used anymore, the blocks are merged again to be available for larger vertex arrays. Like the LRU caching scheme used on the CPU, the data corresponding to nodes that are not required anymore are kept in graphics memory until they have to be paged out due to resource shortage.

The rendering approach is realized in two passes. In the first pass, the particles are projected onto the screen in order to accumulate the squared dark matter density and the density-weighted velocity dispersion along the lines of sight. The result is stored in a render target that is bound as a texture resource in the second pass. In this pass, the pixels final colors are computed by using the accumulated values as texture coordinates into a precomputed 2D color transfer function.

Due to our multi-resolution representation, the particle coordinates of a node are encoded according to their discretization within the corresponding sub-domain. By using the size of the node as scale factor and its position as translation vector, we can decode them into spatial coordinates. The particle attributes are looked up from the quantization codebook.

4.9.1 Particle Projection

The particle projection onto the screen is performed by splatting. Particles that are classified as *points* are simply rendered as point primitives. For *sprites*, we use a geometry shader to expand each particle to a face that has to be large enough to cover the projected area of the particle's smoothing kernel. We use faces perpendicular to the vector from the camera's origin to the particle center since under this projection a sphere is mapped to a circle. In contrast, on screen-aligned faces a sphere would be projected to an ellipse, for which it is much harder to compute a minimum covering face. In case of a camera-oriented face, this distortion is automatically handled by the GPU in the projection transformation. The size of the projected circle r is computed as $r = hd/\sqrt{d^2 - h^2}$, where h is the smoothing radius and d the distance to the camera (see Figure 4.7).

Via texture coordinates the relative distance of each vertex of the face to the particle center is stored as vertex attribute, which is interpolated on a per-fragment basis during rasterization. In a pixel shader, we have to perspectively correct the interpolated distance r' in order to retrieve the minimum distance $s = \tilde{r}/\sqrt{1 + \tilde{r}^2/d^2}$ of the respective line of sight to the particle center (see Figure 4.7). Now, s can be either used for computing the line integral through the smoothing kernel on the fly or to look-up the precomputed integral value from a one-dimensional texture map.

4.9.2 Performance Issues on the GPU

Regardless of our efforts to minimize the number of rendered particles, the geometry throughput and rasterization of particle splats on the GPU still remain critical bottle-

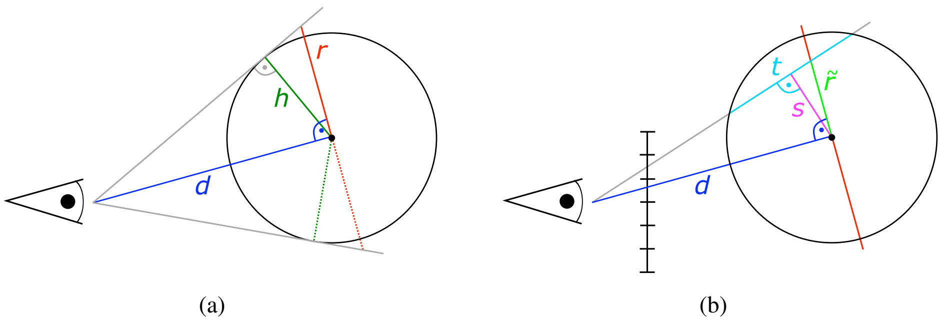


Figure 4.7: *Perspective correction for line of sight integration through a spherical particle. (a) Instead of using the smoothing length h as splat extend, the projective extent r at the distance d of the particle center is computed first. (b) During fragment shading the closest distance s from the line of sight to the particle center is determined based on $\tilde{r}$ and d (given as fragment attributes). s is then used to determine the line integral through the particle.*

necks for interactive visualizations. We further reduce these problem by discarding particles from the rendering process as early as possible in order to forgo the unnecessary expanding of splats and to reduce the number of primitives that are further processed and rasterized.

For this purpose we employ fine-grain view-frustum culling and in addition, we reapply presence acceleration that has been introduced in Section 4.7.3 on a per-particle basis. The impact of the latter technique is especially profitable, since we fade-out the densities at the end of our viewing distance. Thus, there is usually a significant number of particles that drop below the density threshold and can therefore be discarded.

In addition to the number of particles, the choice of the proxy geometry for rendering point sprites has a considerable impact on the rendering performance as well. Using equilateral triangles instead of quads cuts the number of primitives into halves and significantly increases the geometry throughput, even though the area for rasterization is enlarged by a factor of $3\sqrt{3}/4$.

Finally, rendering multiple fragments to the same screen location in a rapid succession leads to a considerable lack of performance. To minimize this effect, we create a random order for rendering the particles within an octree node by shuffling the particle data during the fusion of the partial octrees at the end of our preprocessing (see Section 4.7.1).

4.10 Results and Analysis

We have implemented our visualization framework in C++ using DirectX 10 and HLSL for rendering. Our experiments have been carried out on a standard PC equipped with an Intel Core 2 Quad Q9450 2.66 GHz processor, 4 GB of RAM, and a NVIDIA GeForce GTX 280 with 1024 MB video memory and PCI Express 2.0 x16. The data was stored on a striped RAID 0 consisting of two ATA Hitachi HDS72101 hard disks. In all of our experiments we have quantized the particle attributes using 8 bit per component, and we have selected a maximum allowed screen space error of 0.8 pixels.

4.10.1 Data Representation

The original data set has a total size of 225 GB, including particle density, radius of influence, and velocity dispersion. The generation of the multi-resolution hierarchy, the quantization of particle attributes, and the creation of the page tree took roughly 5:20 hours. In comparison to the simulation itself, which took 28 machine days on a super-computer, and the fact that rendering movies of the data set in software takes several hours as well, this duration is rather small. In fact, a major part of the processing time is spent on data I/O[2]. To avoid this overhead, the preprocessing could directly be performed on the super-computer before recording the snapshot of the running simulation. In addition, this would allow to significantly accelerate the preprocess by exploiting the parallelizability of the LOD generation and by utilizing the space partitioning data structures and the optimized neighborhood search of the underlying simulation.

The final octree has 10 LODs, with the coordinates of the particles in one node being quantized uniformly using 8 bits per component. Thus, at the finest LOD a uniform spatial resolution of 128K^3 is achieved, which is even slightly higher than the resolution ($100,000^3$) at which the simulation has been performed. To validate our visualization approach with respect to speed and quality we have created two data sets, the first one using 8 bits to quantize particle quantities and the other one using 16 bits. This results in an overall compression rate of $6 : 1$ and $4.8 : 1$, respectively, for the particle data. The respective page-trees are 159 GB and 198 GB in size.

Table 4.1 shows detailed statistics for the multi-resolution particle hierarchy. Level 0 represents the finest LOD and contains all particles of the simulation. With increasing level, the number of particles are successively reduced due to particle thinning as proposed in Section 4.7.3. At finer LODs the effect is rather minor, which is not surprising

[2]Due to the limited main memory, we need to read and write more than 735 GB of data. Assuming an average transfer rate of 100 MB/s, the data I/O would require more than 2 hours.

due to the conservative approach we use. With increasing LOD, however, the reduction between two consecutive levels increases up to 91%.

Table 4.1: *Statistics for each level of the particle multi-resolution hierarchy. Merged shows the number of merged particles from level $n-1$ to level n. Skipped gives the number of particles which have been skipped at level n due to presence acceleration, but which may contribute to a merged particle at level $n+1$. Sprites and Points show the number of sprites and points, respectively, with their sum being listed in Particles. Average gives the average number of particles per node.*

Level	Merged	Skipped	Sprites	Points	Particles	Average
9	1 149 203 319	196 713 663	0	5 443 448	5 443 448	5 443 448
8	1 389 651 438	1 335 618 568	0	15 741 862	15 741 862	1 967 732
7	1 270 473 711	2 700 351 786	6 324 410	34 335 672	40 660 082	635 313
6	1 231 730 796	3 551 487 161	403 898 837	56 099 581	459 998 418	898 434
5	1 278 797 785	3 197 259 628	1 992 911 560	53 045 187	2 045 956 747	499 501
4	1 284 656 859	2 135 299 112	4 343 694 323	43 020 725	4 386 715 048	133 871
3	1 111 595 593	1 082 261 354	6 692 845 763	31 563 902	6 724 409 665	25 651
2	736 830 482	319 788 817	8 574 537 356	23 940 439	8 598 477 795	4 100
1	422 598 906	0	9 655 097 094	0	9 655 097 094	575
0	0	0	10 077 696 000	0	10 077 696 000	75

In comparison to the total number of particles, the fraction that is represented as *points* is marginal up to level 7. In fact, this number is considerably larger, but since the *points* are typically part of very dense regions their number is reduced significantly by the proposed particle thinning strategy. While the total number of particles decreases for ever coarser LODs, the average number of particles per node is steadily increasing due to the decreasing number of nodes. In addition, on larger scales the data is distributed more uniformly across the nodes, resulting in a higher fill rate of the quantization bins. According to the increasing average number of particles per node, on levels 4 and higher the amount of memory required to store the particles exceeds the selected page size. Therefore, the collection of several nodes into one page does only occur on the four finest LODs.

4.10.2 Attribute Compression

In Figure 4.8 we compare the quality that can be achieved by vector quantization of particle attributes using 8 bits and 16 bits per component. In both cases the differences to the original data can hardly be noticed, and they are only recognizable by investigating a magnified area in the data. To give an objective measure of the difference, we computed the Euclidean distance of all pixels in the RGB color space for multiple views. The histogram of the difference between the 8 bits compressed data and the original data backs up the visual impression by an average error of 2.0 in color space, where 255 corresponds to full intensity. With regard to the compression rate of 12 : 1, a high fidelity can be achieved. Comparing the 16 bit data set with the uncompressed version, the difference is not even visible in the zoomed area so that the images can be regarded as visually equivalent. The analysis of the histogram proves that the images are not identical, but with an average error of 0.2 these differences are negligible.

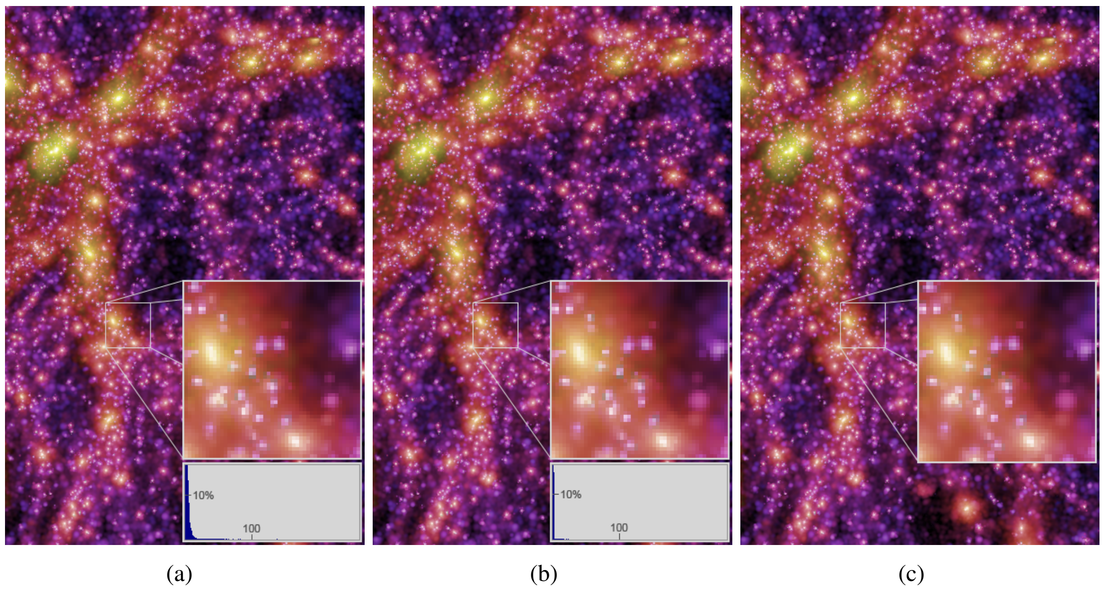


Figure 4.8: *The quality of vector quantization using 8 bit (a) and 16 bit (b) for the compression of particle attributes is compared. On the right, the data was visualized using the original floating point attribute values. Close-up views are used to emphasize the quality of vector quantization. For the compressed data, the histogram of the color differences to the original image are shown below the close-up views.*

4.10.3 Performance Measurements

To analyze the impact of the various parts of our visualization approach on its performance, we have measured memory consumption, data transfer rates, geometry throughput, and rendering performance during a straight flight at constant velocity through the Millennium Simulation. The velocity was 12.76 million light years per second at a side length of the cubic simulation domain of 2.23 billion light years. The results are shown in Figure 4.9. By using a straight path, the amount of data in the spherical prefetching region that can be re-used in every frame is minimized. The particle data was rendered onto a 1200$\times$800 viewport.

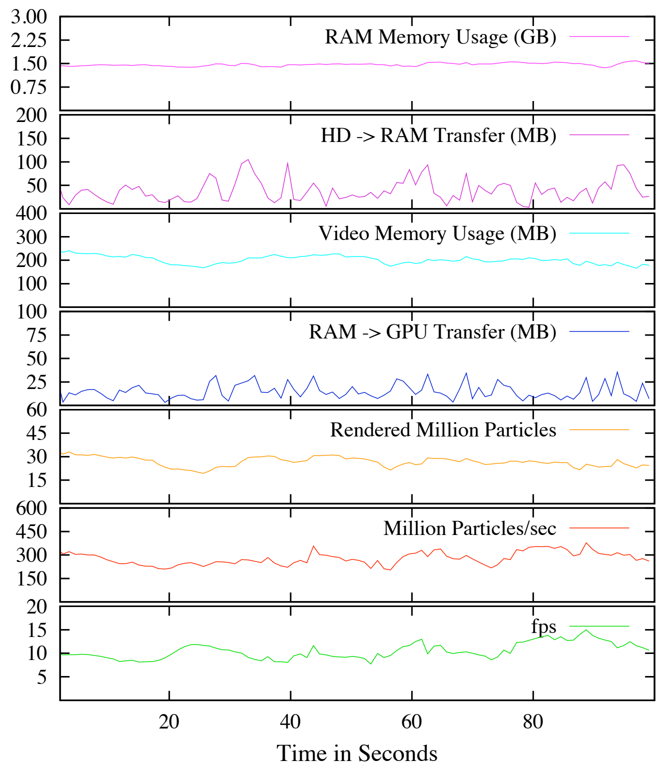


Figure 4.9: *Performance measurements and memory statistics for a flight through the Millennium data set. Memory and bandwidth requirements are not at the system limits. Geometry throughput on the GPU effectively limits the performance when rendering onto a 1200×800 viewport.*

As can be seen in the uppermost plot, the main memory consumption is more or less constant around 1.5 GB. This is due to the prefetching scheme, which effectively hides variations in bandwidth requirements that are imposed by spatial variations of the particle densities. In the hard disk transfer rate in the second plot these variations are much more apparent, which is in particular caused by a large number of parallel I/O requests. However, with a maximum of about 105 MB/s we are still far away from the maximum data throughput of the striped RAID 0, which is above 200 MB/s. Most of the time, the transfer rate is below 60 MB/s. Hence, the sustained transfer rate of today's standard hard disks is absolutely sufficient for data streaming. Note that even if the used disk system would not be able to achieve the required peak rate, this would probably not result in a loss of detail since the measured transfer is only caused by data prefetching.

The third plot in Figure 4.9 shows the amount of graphics memory that is required for rendering. With an average size of 202 MB and a maximum size of 246 MB, the memory requirements are remarkably low. In addition, with a CPU-GPU transfer rate of at most 35 MB/s we are far below the bandwidth that is available on current graphics hardware. In combination with the proposed prefetching scheme, the data to be rendered is resident in main memory when it is needed and can directly be streamed to the GPU.

The lower three plots of Figure 4.9 show the performance measurements for rendering. Overall, an average frame rate of 10.5 fps is achieved, with a minimum and maximum of 7.7 and 15.0 fps, respectively. To analyze the effectiveness of the proposed data reduction strategies, we have tried to render the same flight without applying particle thinning and presence acceleration, yielding an average frame rate of 1.63 fps.

Regarding geometry throughput, we achieve a peak performance of 394 million particles per second, with 280 million particles per second in average. This is interesting, because the theoretical maximum of our target architecture is at roughly 300 million triangles per second. Due to fine-grain view frustum culling and presence acceleration, however, the number of effectively rendered sprites is significantly below the number given in the diagram. Based on benchmarks we can state that the maximal throughput of our rendering approach is in fact about 220 million sprites per second, where the loss in performance is introduced by the geometry shader. This throughput is further reduced when rendering very dense regions. Despite our multi-resolution strategy, an overdraw of more than 2000 is not unusual in the current application, clearly indicating that rasterization becomes the rendering bottleneck.

Figure 4.10 illustrates this problem and the effect of the density based culling strategy proposed in Section 4.9.2. The bottom pictures give information about the fragment overdraw for rendering with both culling techniques enabled (left) and without these techniques (right), leading to the same visual output (top image). The color coding ranges from an overdraw of 0 (black), 500 (magenta) to 3000 (red) with a linear interpolation of the color's hue for the values between 500 and 3000. Obviously, the number of pixels with a high overdraw as well as the maximum overdraw of fragments is much lower in the left image. Combined with GPU view frustum culling this results in a performance gain of 236%, yielding 10.4 fps instead of 4.4 fps. The culling techniques for themselves achieve frame rates of 9.2 (view frustum culling) and 4.7 fps (density-based culling).

Figure 4.11 illustrates the scalability of our system with regard to the viewport resolution. We attribute this scalability to the presence acceleration scheme, as it effectively prevents the geometry load from scaling linearly with screen resolution. Consequently, the drop in performance is mainly due to the increased fragment load, which—due to the use of a geometry shader—scales more gracefully.

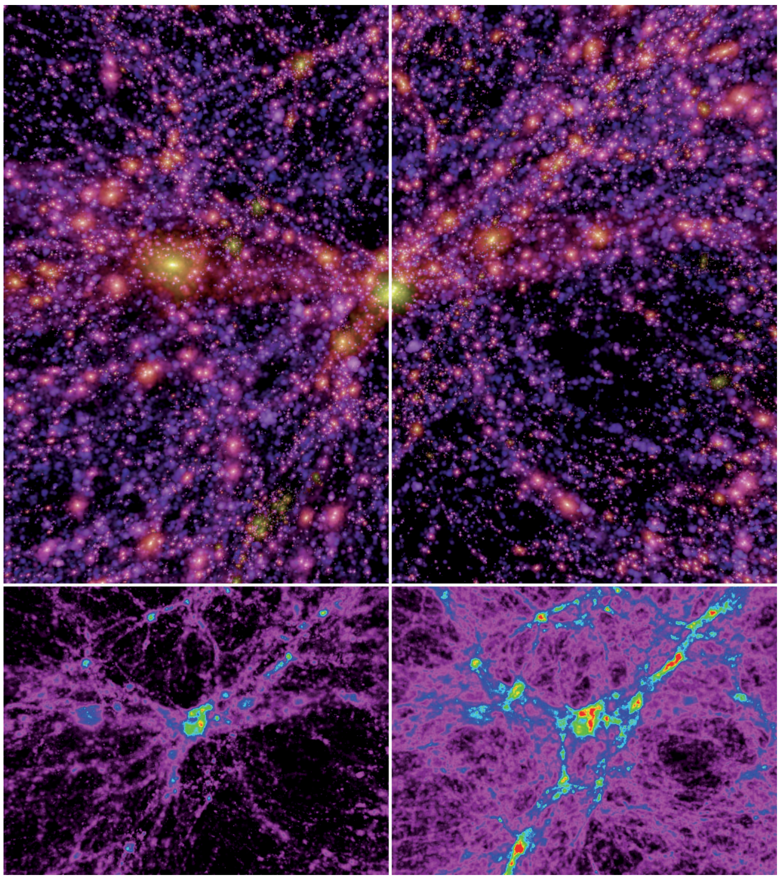


Figure 4.10: *Top: Comparison of image quality with (left) and without (right) presence acceleration. The rendering performance for rendering the full image is 10.4 fps with the acceleration enabled and 4.4 fps if not. Bottom: Pixel overdraw in the full image for the compared rendering options. Hue encodes the overdraw from black (0 fragments) over magenta (500 fragments) to red (3000 fragments).*

4.11 Discussion

The results presented in the previous section have proven the capability of our proposed techniques to render data sets with billions of particles at interactive frame rates on common PC hardware. By quantizing particle coordinates into a regular spatial grid and by using vector quantization to compress particle attributes, compression rates of up to $6 : 1$ are achieved at very high fidelity. By employing an octree-based multi-resolution hierarchy, the amount of data to be transferred and rendered is considerably reduced. Based on an efficient memory management the streaming of data from hard disk to the video memory is completely hidden from the user for continuous camera movements. The measurements in Figure 4.9 clearly demonstrate the scalability of

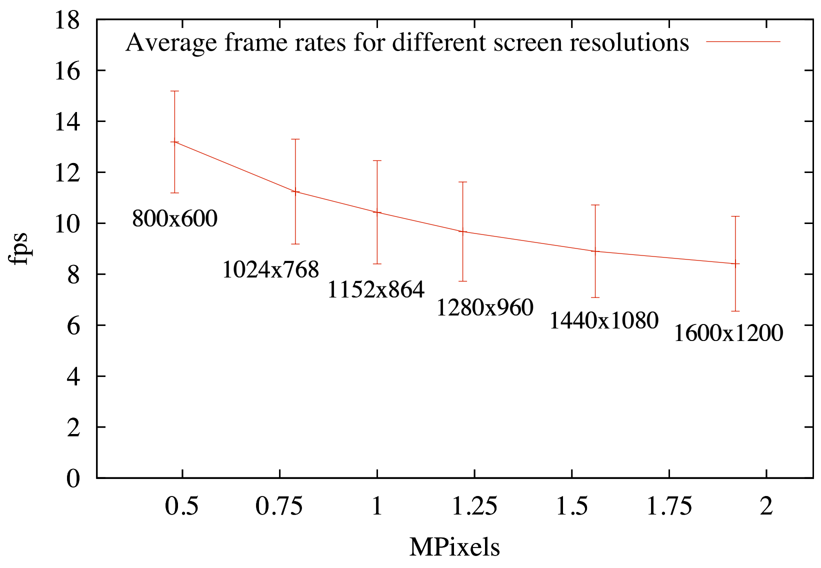


Figure 4.11: *Average frame rate and standard deviation for rendering the Millennium Simulation data at different viewport resolutions.*

the framework, allowing for the interactive rendering of much larger data sets than the Millennium Run.

The bottleneck in the rendering of large particle data turns out to be the geometry throughput on recent GPUs as well as the overdraw at high density regions. Despite their highly parallel streaming architecture, the massive geometry load induced by the rendered particles lets the GPU become the limiting performance factor. To overcome this limitation, one possible solution is to reduce the number of rendered particles by using alternative rendering primitives. For instance, recent findings in the context of GPU-based terrain rendering [DKW09] show significant advantages of ray casting over rasterization for the rendering of high-resolution polygonal terrains, where the polygons are approximately one pixel in size. The reason is that ray casting can effectively exploit the regular grid structure underlying such fields and avoids pixel overdraw, while the performance of rasterization is limited by the polygon throughput on the GPU. Similar to this approach, we have experimented to resample regions exhibiting a high particle density into volumetric textures, and to render the data set by means of a hybrid scheme using volume ray casting and particle rasterization. Indeed, experiments with high density regions of the Millennium Run revealed that ray casting can outperform particle splatting up to a factor of 6. However, efficient ray casting is only possible for uncompressed volume data. As a consequence, replacing a significant amount of particle data by a volumetric representation, would clearly exceed the memory capabilities of today’s graphics hardware and is therefore no alternative for such large-scale data sets today.

4.12 Conclusion

In this chapter, we have presented a scalable approach for the interactive visualization of large cosmological particle data on recent PC architectures. By quantizing particles into a multi-resolution octree structure and introducing rules for particle merging and deletion when no visual error is implied, we can significantly reduce the amount of data while maintaining a user-defined screen space error. Combined with an out-of-core and in-core data management to efficiently access the data and optimized particle splatting we are able to visualize one time-step of the Millennium Run exceeding 10 billion particles at interactive frame rates. To our best knowledge, this is the first approach that is able to interactively visualize particle data of this size.

One drawback of our approach is that depth perception gets lost according to the order-independent rendering scheme. The subsequent chapter shows how high-quality visualization approaches such as direct volume rendering and isosurface ray casting can be efficiently performed on GPUs. Using the data management strategies of this chapter and an adapted particle hierarchy, we apply this technique to large-scale SPH data sets including the Millennium Run.

Chapter 5

Efficient High-Quality Volume Rendering of SPH Data

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

5.1 Introduction

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

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

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

The second approach can leverage the same adaptive data structure as the simulation itself, but it is view-dependent and comes at high computational and memory access load to integrate particle quantities in the correct visibility order along the view rays. Due to these limitations, to the best of our knowledge the use of this approach is currently restricted to off-line visualizations [ACP08, DRGI08] or to data sets of restricted size [OKK10].

5.2 Contribution

In this work we introduce a new volume rendering pipeline for SPH data on desktop PCs. This pipeline can efficiently render a continuous density field—or any other scalar quantity—that is given by a discrete SPH particle set. It employs 3D texture-based volume rendering on the GPU and therefore provides different rendering options including direct volume rendering and isosurface rendering. Some examples are shown in Figure 5.1.

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

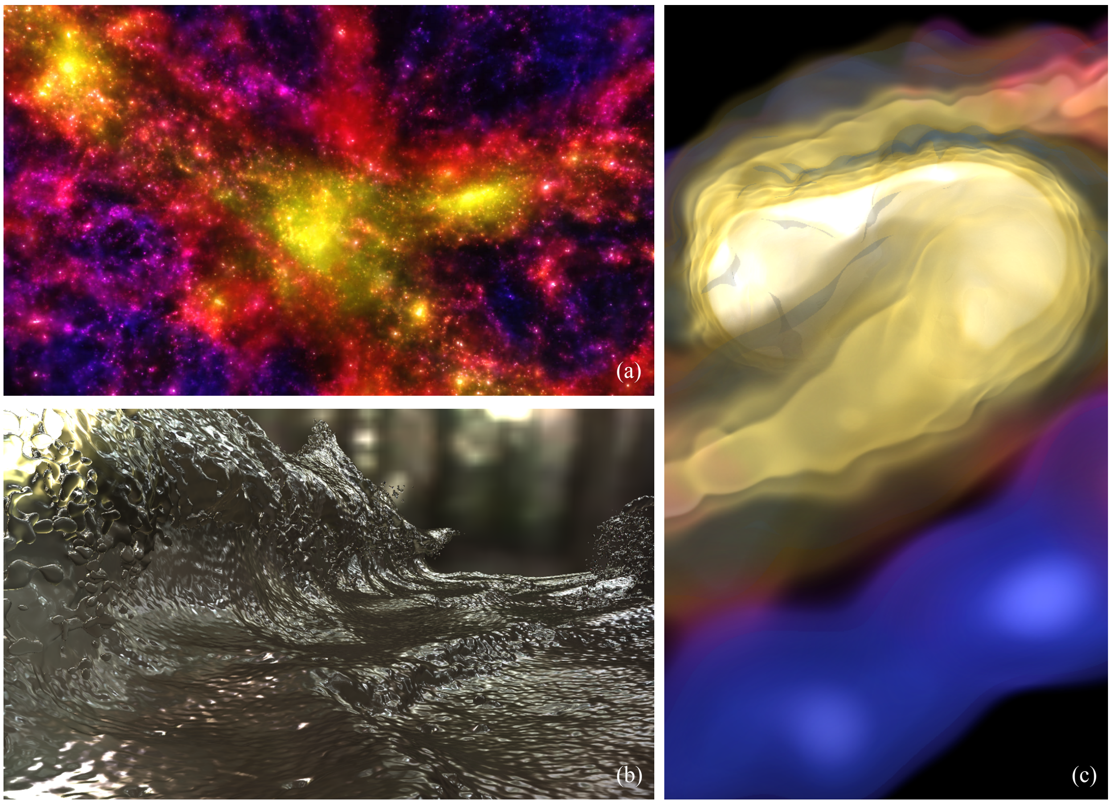


Figure 5.1: *A novel technique for order-dependent volume rendering of SPH data is presented. It provides rendering options such as (a) direct volume rendering, (b) isosurface rendering, and (c) mixed modes, and it renders data sets consisting of millions of particles at high quality and speed: (a) 42M @ 0.1 fps, (b) 2.2M @ 4.5 fps, and (c) 2M @ 1.7 fps, on a 1024×1024 viewport.*

ference is that the grid is not uniform but adaptively discretizes the visible domain. The grid resembles a classical ray-tracing grid which fans out with increasing distance from the view plane [APB87], but it has a spacing between the grid vertices along the view rays that increases exponentially. Thus, the grid results in an adaptive sampling of the view frustum with decreasing sampling rate along the viewing direction. We will subsequently call this grid the *perspective grid*.

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

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

Since our approach reconstructs the continuous scalar field from the discrete particle samples and renders this field via ray casting, high-quality visualizations using arbitrary transfer functions can be generated. Especially compared to particle splatting, which projects each particle's reconstruction function separately to form a 2D pre-shaded footprint, vastly different image quality can be achieved. Figure 5.2 demonstrates this effect for a SPH data set in which the reconstruction functions from different particles overlap. The particle extents in this data set differ about a factor of 1400. While splatting (a) can only render a coarse approximation to this data, volume ray casting (b) generates a smooth continuous image of the discrete particle set and can reveal fine details of surface structures via additional isosurfaces (c).

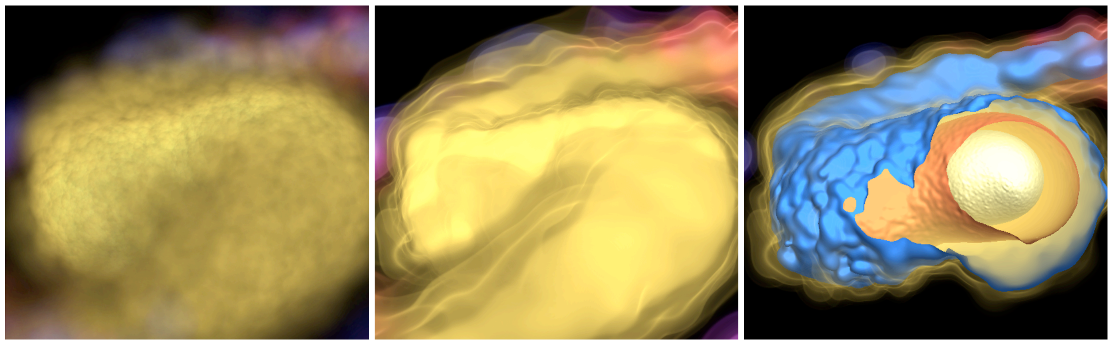

(a) (b) (c)

Figure 5.2: *Order-dependent particle splatting (a) results in vastly different visualizations than volume ray casting (b) or hybrid rendering with isosurfaces (c). Splatting approaches cannot accurately represent high-frequency color and opacity transfer functions and have difficulties in revealing surface-like structures in the data.*

5.3 Related Work

Particle-based simulation techniques such as SPH have been studied extensively over the last years. There is a vast body of literature related to this field and a comprehensive review is beyond the scope of this thesis. However, Monaghan [Mon05], Müller et al. [MCG03], and Adams et al. [APKG07] discuss the basic principles underlying such techniques and provide many useful algorithmic and implementation specific details.

Only very little work has been reported on the efficient rendering of SPH data. Most commonly this task is reduced to rendering isosurfaces in the continuous 3D density field derived from the discrete set of particles. This can be accomplished by resampling particle quantities into a proxy grid [CSI09, NJB07], and by using direct volume ray casting [DCH88] or isosurface extraction [LC87]. Schindler et al. have extended the Marching Cubes algorithm on a virtual grid to extract the isosurface of SPH data [SFWP11]. Zhu and Bridson use a correction term for the density before the reconstruction, leading to smoother surfaces [ZB05]. GPU-based particle slicing has been introduced by Kolb and Cuntz using multiple render targets [KC05] and was extended to an indexed scattering approach by Harada et al. [HKK07a]. An efficient GPU technique for resampling particle quantities directly into 3D uniform grids has been presented in [ZSP08]. For isosurface rendering, resampling can be restricted to the surface boundaries [YHK09]. If a uniform proxy grid is used, high resolution is required to capture all details, but GPU-based volume rendering approaches can be employed to achieve high speed [DZTS08, KW03]. Jang et al. accelerate the neighboring search for direct SPH interpolations on the GPU by a BSP tree data structure [JFSP10]. Orthmann et al. present optimized GPU-based volume ray casting using efficient caching strategies for multi-resolution SPH particles [OKK10]. Parallel to our work, Falk et al. have developed a similar volume rendering technique that is based on an iterative splatting of metaballs into a moving stack of equidistant slices [FGE10].

In order to avoid the exhaustive memory consumption of a proxy grid, isosurfaces can be rendered directly, for example, by evaluating ray-particle intersections on the GPU [KSN08, ZSP08]. Gribble et al. performed direct ray-sphere intersections on the CPU using a grid-based acceleration structure [GIK$^+$07]. An isosurface extraction technique that works directly on the particle set was introduced by [RB08]. Rosenthal et al. generate a surfel representation of the isosurface from the discrete particle set [RRL07]. A high-quality yet time-consuming surface-fitting approach using level-sets was presented in [RL08].

Splatting of transparent particle sprites for volumetric astrophysical SPH data is presented in Chapter 4 along with a review of related work. Visualization systems for SPH data, including rendering options such as splatting or slicing and providing application specific mechanisms to interact with and analyze particle data were discussed in [BGM+07, BGM08, EGM04, Pri07, WKM05].

Recently, screen-space approaches for rendering isosurfaces in SPH data have gained much attention due to their efficiency. Adams et al. render particles as spheres and blend the contributions in the overlap regions [ALD06]. Müller et al. reconstruct a triangle mesh in screen space from visible surface fragments that are generated via rasterization [MSD07]. The method was improved by [CS09, vdLGS09] to generate smooth surfaces and to avoid an explicit triangulation.

5.4 Perspective Grid

The perspective grid is used to resample the particle data inside the view volume. It is a structured grid that partitions the view frustum into $k \times l \times m$ oblique subfrusta. Figure 5.3(right) illustrates this grid, which has the specific property that the sampling rate along z is the same as along x and y. Optimally, k and l correspond to the viewport resolution but it is also possible to choose a smaller grid size in order to reduce the number of samples.

The perspective grid is stored on the GPU in a 3D texture map (see Figure 5.3(left)). In the resampling process, the 3D texture map is used as a render target and resampled quantities are scattered to this target using accumulative blend. To perform the resampling efficiently on the GPU, a mapping of a point $\mathbf{r}' = (x', y', z')$ from 3D texture space to a point $\mathbf{r} = (x, y, z)$ in view space and vice versa is required. In the following, we will derive this mapping and describe how to exploit it in the resampling process.

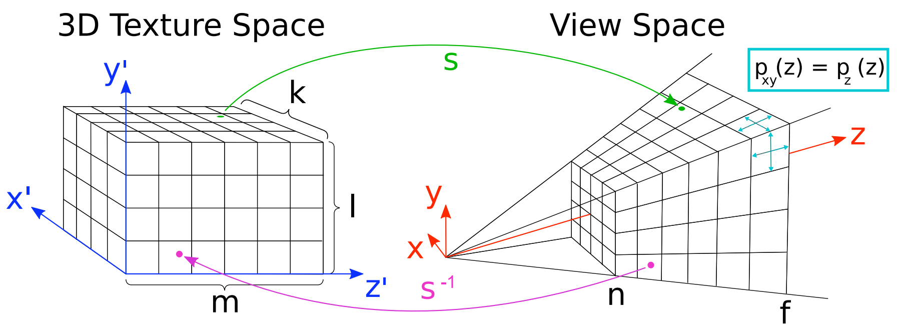


Figure 5.3: *The Cartesian grid in texture space (left) transforms onto the perspective grid in view space (right) under the mapping s. The grid has the same sampling rate along x and y ($p_{xy}(z)$) as along z ($p_z(z)$). We assume that the viewer is looking along the positive z-axis.*

Perspective Grid Mapping

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

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

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

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

$$p_z(z) = \sigma \cdot s_z(z') = \frac{\mathrm{d}s_z(z')}{\mathrm{d}z'}, \tag{5.1}$$

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

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

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

$$s_z(z') = n \cdot \left(\frac{f}{n}\right)^{z'/m}. \tag{5.2}$$

It's inverse is

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

Note that this corresponds to the transformation derived in [SD02, WSP04] to equally distribute the aliasing error in perspective shadow map parameterizations. This mapping is only correct, however, on the z axis. On every other view ray the sampling rate is larger, since all of them perform the same z sampling in world space. In the extreme case, which is along the edges of the frustum, the sampling rate scales by the factor

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

We must further determine the number of sample points m along the z-axis for a given range $[n...f]$. By inserting equation 5.2 into 5.1 and solving for m, we obtain

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

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

5.5 Data Resampling

To resample the particle data to the perspective grid, for every particle the grid vertices within the support of the particle's smoothing kernel have to be determined and the data is interpolated according to the kernel function (see Section 3.2):

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

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

One key to an efficient and high-quality resampling of large particle sets is the use of a multi-resolution particle representation [FSW09, HE03, HLE04]. In such a representation the particle set is encoded at different levels of detail by merging subsets consisting of smaller particles into one larger particle. The hierarchical particle representation allows pruning particles that are too small to be reconstructed at the required sampling rate. Thus, aliasing artifacts can be avoided and the number of particles to be processed can be reduced.

It is important to note, however, that resampling into the perspective grid is highly beneficial even without a multi-resolution particle representation, e.g. for an ad-hoc visualization of a GPU-based SPH simulation [HKK07b, GSSP10]. Compared to the resampling into a uniform grid, the perspective grid effectively reduces the number of samples and thus the memory requirements as well as the SPH interpolations to be performed. A performance comparison between rendering particle data with and without a multi-resolution representation can be found in Section 5.8.

5.5.1 Hierarchical Particle Representation

Our pre-computed multi-resolution particle representation is organized in an adaptive octree data structure similar to the one proposed in Chapter 4. In particular, for large particle sets and high spatial resolution of the simulation we employ the same regular domain partition to allow for the construction of the particle hierarchy in parts. In addition, for the Millennium gas dynamics simulation we scale each particle component logarithmically and compress the data using vector quantization. Spherical pre-fetching regions are realized on the GPU and the CPU to exploit frame-to-frame coherence and thus reduce memory access latencies.

In contrast, however, we use different rules for merging particles to account for the three dimensional particle reconstruction. These rules build upon the resampling operators presented in [DC99, HHK08] for adaptive SPH simulations.

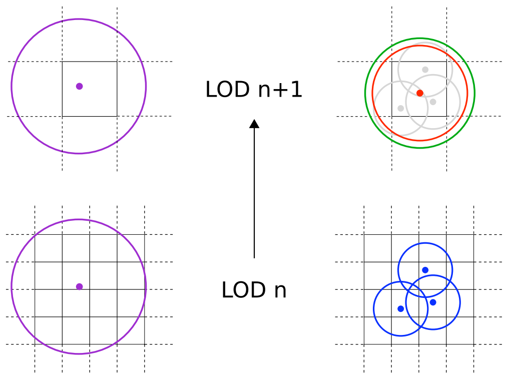


Figure 5.4: *Bottom-up construction of the particle LOD-hierarchy. Particles are copied to the next coarser level until their diameter falls below the grid sampling (left). Particles falling below the grid sampling are merged and eventually enlarged to the grid spacing (right).*

Starting with a uniform grid at the resolution at which the simulation has been performed, particles in contiguous blocks of 2^3 cells are merged into a single particle. The volume of this particle is the sum of the volumes of the merged ones, and the scalar quantities of the merged particles are averaged into this particle according to their mass contribution. The merging process is illustrated in Figure 5.4. This process is recursively repeated until a user-given resolution level is reached. In the merging process, at a particular level only particles with a radius less than the cell size on the next coarser level are merged. If the radius of a new particle falls below this size, it is increased to this size and its density is decreased proportionally to reflect the volume increase. All other particles are copied to the coarser level to guarantee a consistent representation of the particle field at all levels of detail.

5.5.2 Hierarchy Traversal

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

5.5.3 GPU Particle Slicing

GPU voxelization of particles into a 3D texture map work similar to the approach presented in [ZSP08]. For each particle a single vertex—positioned at the particle center and attributed by the particle quantity to be resampled—is sent to the GPU and passed to the geometry shader. The geometry shader computes the first (s_{min}) and the last (s_{max}) 2D texture slice that is covered by the particle as

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

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

The shader spawns $s_{max} - s_{min} + 1$ equilateral triangles from this vertex, one for each slice $i \in \{s_{min}, ..., s_{max}\}$. The triangles are centered at the particle position and oriented orthogonal to the viewing direction. For the i-th triangle, it's depth $i + 0.5$ in texture space is transformed to view space as $z_i = s_z(i + 0.5)$. The size of each triangle is chosen such that its inner circle is equal to the circular cross section between the sphere of radius h centered at the particle position and the slice z_i. Figure 5.5 illustrates this process. For every triangle vertex the distance vector to the particle center is determined and assigned as vertex attribute. Finally, the triangle is rendered into slice i of the 3D texture map using the standard perspective projection.

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

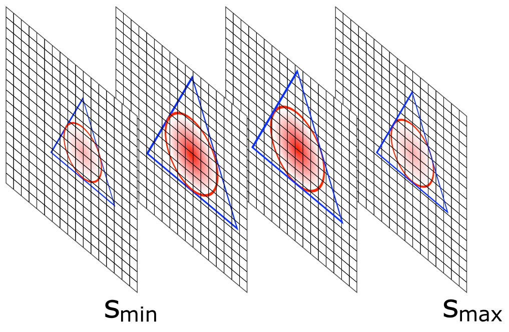


Figure 5.5: *Particle resampling on the GPU. As many proxy triangles as there are slices overlapped by a particle are rendered. Triangles are scaled to cover the particle-slice cross-sections.*

Even though the triangles for all covered slices can be spawned by a single geometry shader pass, this proceeding causes significant performance drawbacks due to the large size of the shader program and the high varying number of triangles to be created. Instead it is much more efficient to employ instanced rendering with each instance j creating the jth triangle of the corresponding particle. Accordingly, the number of instances must match the maximum number of covered slices of all particles within a single draw call. For particles with less slices, no geometry is created for the additional instances. The instanced rendering approach benefits from the less complex shader to be executed and from the increased synchronization of execution paths among different particles.

5.6 Rendering

Once the particle quantities have been resampled to the perspective grid, the data can be rendered in turn on the GPU using texture-based volume ray casting. This enables using different rendering options such as isosurface rendering or direct volume rendering simultaneously at very high speed. The major difference to classical texture-based ray casting is in the kind of grid that is rendered. Usually, the data is given on a Cartesian grid in world space and has to be interpolated tri-linearly at the sample points along the rays. The data in the perspective grid, on the other hand, is already at the positions in view space where a sample is placed during ray casting. Consequently, the data values in the 3D texture map that stores the perspective grid can be directly accumulated in front-to-back order along the z' coordinate axis in texture space after mapping them to color and opacity based on a user-defined transfer functions. In our implementation we use a two-dimensional transfer function with post-classification as described in Section

3.5.4. To account for the varying sampling distances, opacity correction of the samples has to be performed (see Section 3.5.3).

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

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

As can be seen, computing a gradient's z component requires interpolating in texture space. 3D texture mapping hardware on the GPU supports tri-linear interpolation, which is not exact in our scenario due to the frustum-shaped cells underlying the texture grid. In order to investigate the error that is made by a central differences approximation, we compared the gradients with the ground truth, which requires to interpolate the gradients of the participating particles at the found intersection points (see Formula 3.3 on page 22). Figure 5.6, however, indicates the differences in the resulting illumination are marginal. We thus apply hardware-supported tri-linear interpolation throughout this work.

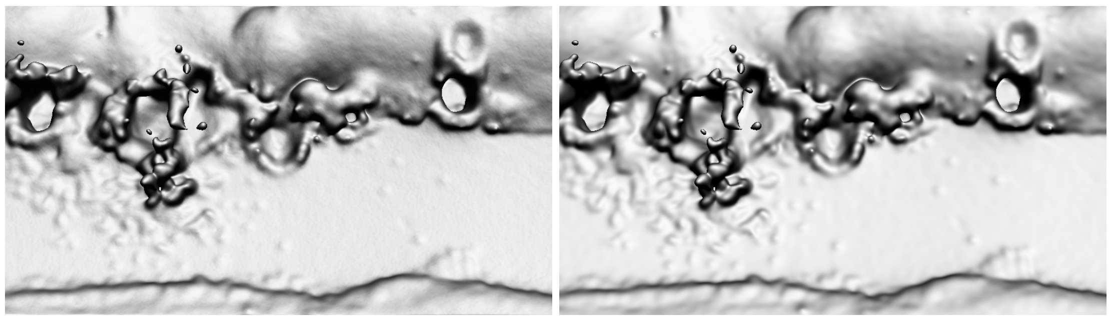

(a) (b)

Figure 5.6: *Gradient computation in the perspective grid: (a) Central differences with trilinear interpolation in texture space, (b) spatial derivative obtained via SPH kernel interpolation.*

5.7 Implementation Issues

5.7.1 Grid Partitioning

Due to the limited amount of graphics memory on the GPU, it is not possible in general to store the entire perspective grid on the GPU. For this reason we partition the perspective grid into a number of view-aligned *slabs* of size $k \times l \times m_s$, each of which is small enough to fit into the graphics memory (see Figure 5.7). In this division we consider an overlap between slabs as well as at the slab boundaries to allow for a consistent gradient computation. In front-to-back order the slabs are processed as described.

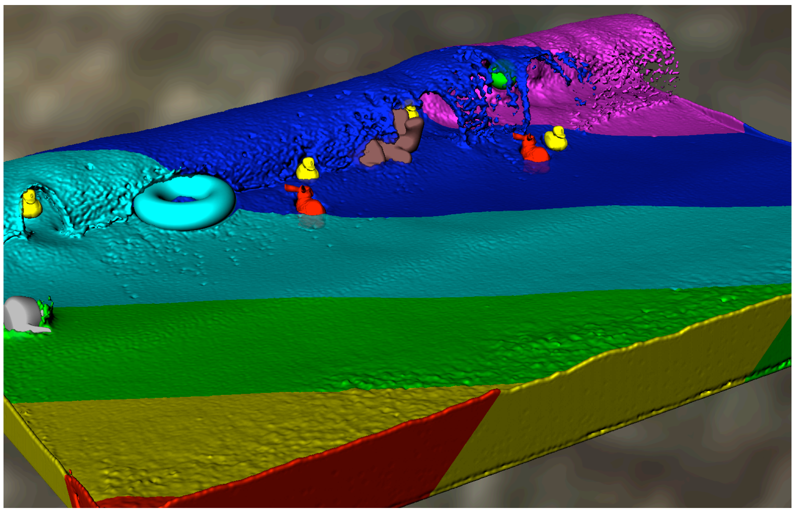


Figure 5.7: *If the perspective grid does not fit into GPU memory, it is partitioned into view-aligned slabs (illustrated by different colors) which are resampled and rendered in front-to-back order.*

In order to reduce the overall workload on the GPU, for every slab only the particles contributing to this slab are resampled into the respective sub-grid. For a multi-resolution hierarchy, this can be accomplished by rendering for every slab only the overlapping nodes. Fine grain view-frustum culling can be additionally applied within the geometry shader. For an ad-hoc visualization of SPH simulations where no LOD hierarchy is available, this approach can also be applied based on any other kind of spatial subdivision schemes that are employed to accelerate the neighborhood search. If no such structure is available, the overlapping particles for each slab can alternatively be determined by a binning pass on the particle set. This can be done by creating key-value pairs for all particles, where the values are the particle ids and the keys identify the slabs covered by each particle. Sorting for the keys gives an index list, which is then used for indexed rendering.

5.7.2 Occlusion Culling

To minimize the number of rendered particles on the GPU, we employ a hierarchical opacity-buffer similar to the hierarchical depth-buffer proposed in [GKM93]. After each or every n-th slab we store the accumulated opacity in a separate texture. This texture is converted into a boolean mipmap to encode ever larger areas that are completely opaque. At ever coarser mipmap levels, a texel is set to 1 (opaque) only if all covered texels in the previous finer level are 1 (see Figure 5.8). By employing bilinear texture interpolation, 4 child texels can be tested with one texture lookup. The algorithm takes care of non-power-of-two textures as described in [GH03].

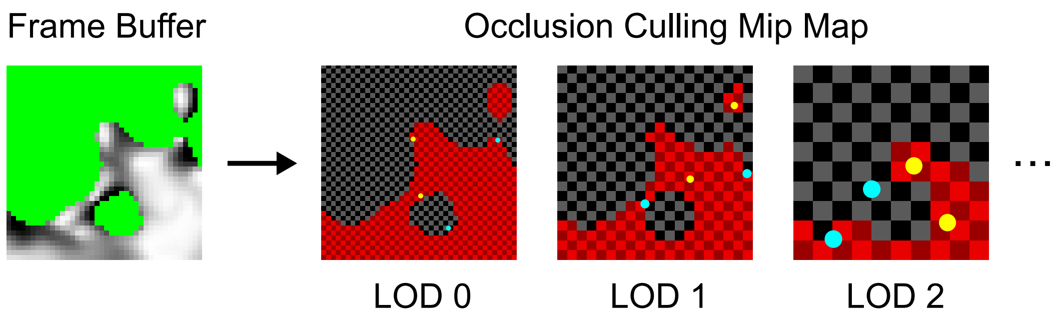


Figure 5.8: *Illustration of our occlusion culling approach. Based on the frame buffer, we create a boolean mipmap that encodes completely opaque areas (marked as red). By choosing the finest mipmap level, in which the diameter of the projected bounding sphere is below texel width, a single texture lookup with trilinear interpolation can be used to verify whether entire octree nodes or single particles are occluded (yellow points) or not (cyan points).*

The mipmap is used to discard all nodes, and further on all particles, which are covered completely by opaque structures in front of them. This is done by constructing a screen space bounding circle for each node and each particle. We choose to sample the first mipmap level in which the circle diameter is below texel width. By sampling at the circle center using bilinear interpolation, all four texels that are possibly covered by the circle can be analyzed at once. If the interpolated value is 1, we can be sure that the bounding circle covers only occluded mipmap texels so that the object can be safely discarded.

In the render pass for creating the boolean mipmap for the finest level, we additionally enable depth writes to set the depth values of all occluded particles to the near plane. This allows us to employ the early z-test to additionally discard occluded fragments in the resampling pass and to avoid ray traversal in occluded areas during the rendering pass.

5.8 Results and Discussion

To demonstrate the efficiency and quality of our approach we render SPH simulation data from fluid dynamics and astrophysics (see Figure 5.9). Table 5.1 gives specific information on these data sets. Particle positions and quantities are encoded in 32 bit and 16 bit floating point values, respectively.

Table 5.1: *Data sets used in our experiments.*

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

The first three data sets simulate fluid dynamics. WDMerger simulates the merger of two white dwarf stars. SNIaEjecta simulates the impact of a supernova ejecta on a companion star. The Millenium data set contains one time step of a simulation of the evolution of the universe. The *poly6*-kernel was used in all fluid dynamics simulations with constant h for each data set and the *cubic spline*-kernel with adaptive smoothing length for the astrophysical simulations. The kernels and their gradients are described in Section 3.2.2.

Performance

Table 5.2 shows the times required for resampling and rendering the data sets. Timings were performed on a 2.4 GHz Core 2 Duo processor and an NVIDIA GTX 280 graphics card with 1024 MB local video memory. The viewing parameters were selected such that the entire set of particles is within the view volume. Successive time steps of time-varying particle sets are streamed consecutively to the GPU. Upload times are not included in the given timings. For the Millennium Run, the given timings and the number of rendered particles in Tables 5.1 and 5.2 are averages over a continuous flight through the data.

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

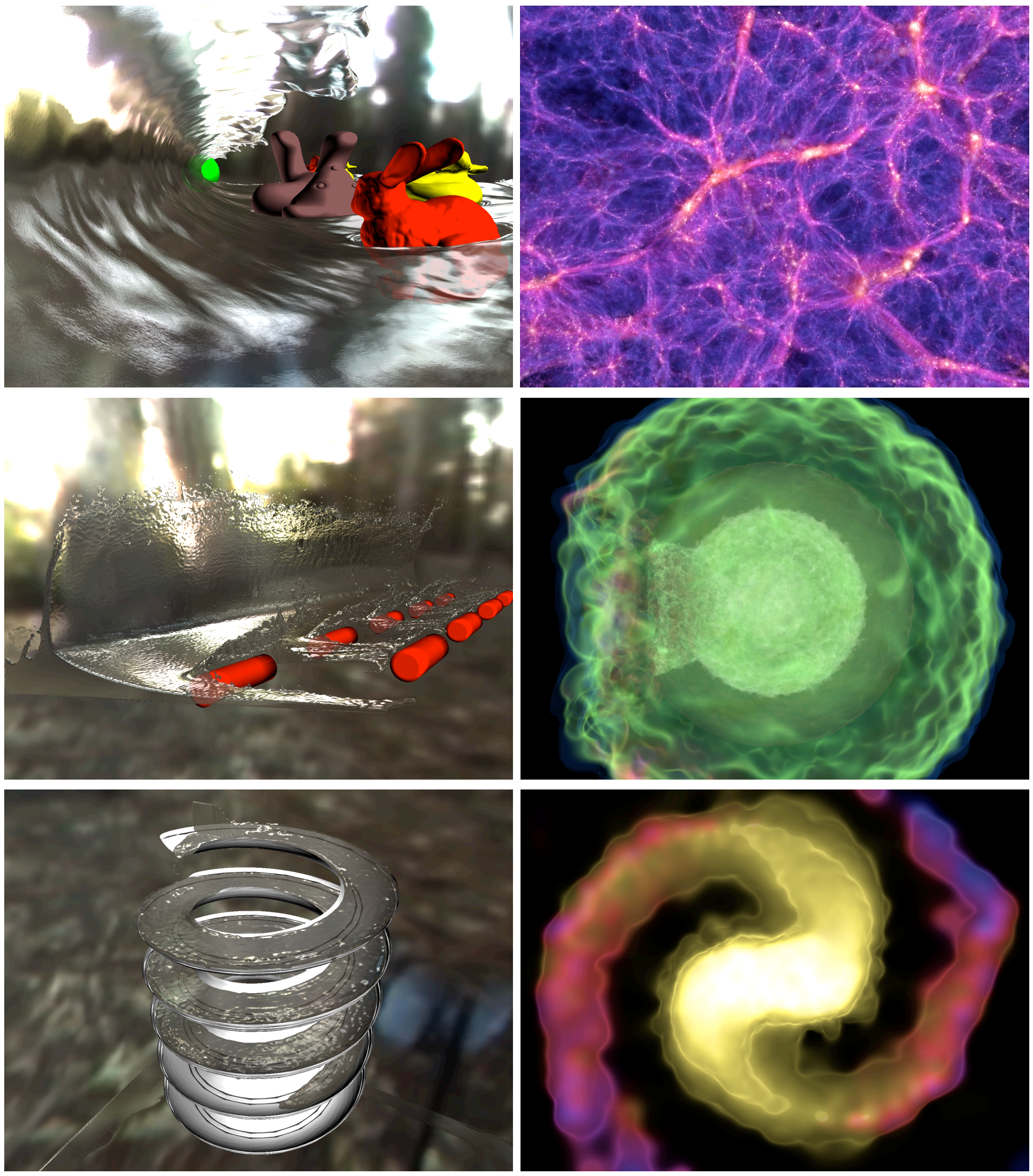


Figure 5.9: *The data sets we have used in our experiments (see Table 5.1 for more information). SPH fluid simulations (1st column, from top to bottom): Large Wave Moving Obstacles (LWMO), Large Wave Static Boundaries (LWSB), and Flume. Gas dynamics simulations (2nd column, from top to bottom): Millennium Run, SNIaEjecta, and WDMerger.*

The high quality of our approach can be seen in Figure 5.15, showing the SNIaEjecta data set and the Millennium Run from different perspectives and with varying transfer functions. Please note that no sampling artifacts are visible which is particularly worth mentioning as we applied high frequent transfer functions to reveal the thin surfaces in all images of the left column and the bottom image of the right column.

Table 5.2: *Performance statistics for resampling and rendering in milliseconds. The viewport resolution is* 512^2*, with a corresponding resolution of the perspective grid. Timings for a* 1024^2 *viewport and corresponding grid resolution are given in brackets.*

Data set	Grid Resolution	Resampling	Rendering	Total
Flume	$512^2 \times 364$ ($1024^2 \times 728$)	6 (38)	43 (76)	49 (114)
LWMO	$512^2 \times 544$ ($1024^2 \times 1088$)	87 (549)	65 (127)	152 (676)
LWSB	$512^2 \times 664$ ($1024^2 \times 1328$)	135 (804)	79 (178)	214 (982)
WDMerger	$512^2 \times 292$ ($1024^2 \times 584$)	642 (1492)	40 (81)	682 (1573)
SNIaEjecta	$512^2 \times 396$ ($1024^2 \times 792$)	1560 (4574)	49 (100)	1609 (4674)
Millennium	$512^2 \times 480$ ($1024^2 \times 960$)	2267 (7771)	83 (162)	2350 (7933)

Multi-Resolution Hierarchy

The quality of our multi-resolution hierarchy is illustrated in Figure 5.10. It shows the LWSB and the SNIaEjecta data set resampled to grids of different resolutions with a corresponding node selection from the LOD hierarchy. The particle hierarchy preserves the basic structures in the data, at the same time providing an effective anti-aliasing structure. Compared to the first image the number of rendered particles decreases from 3.2M over 2.1M and 0.7M down to 0.3M particles for the LWSB data set, and from 8.7M over 4.6M and 3.3M down to 1.4M particles for the SNIaEjecta data set. On a 1024^2 viewport the total rendering time decreases by a factor of 4.2/10/27 and 2.9/8.2/19, respectively. This also demonstrates that interpolation between grid samples can be used to decouple the resolutions for resampling and ray casting, which allows to find a good trade-off between image quality and rendering speed.

Direct Particle Visualization

To test the rendering performance for ad-hoc visualizations we rendered the LWSB and SNIaEjecta data sets without a pre-computed multi-resolution hierarchy or the help of other space partitioning data structures. To minimize the geometry load, we have applied an efficient parallel radix-sort pass [SHG09] using CUDA to determine the

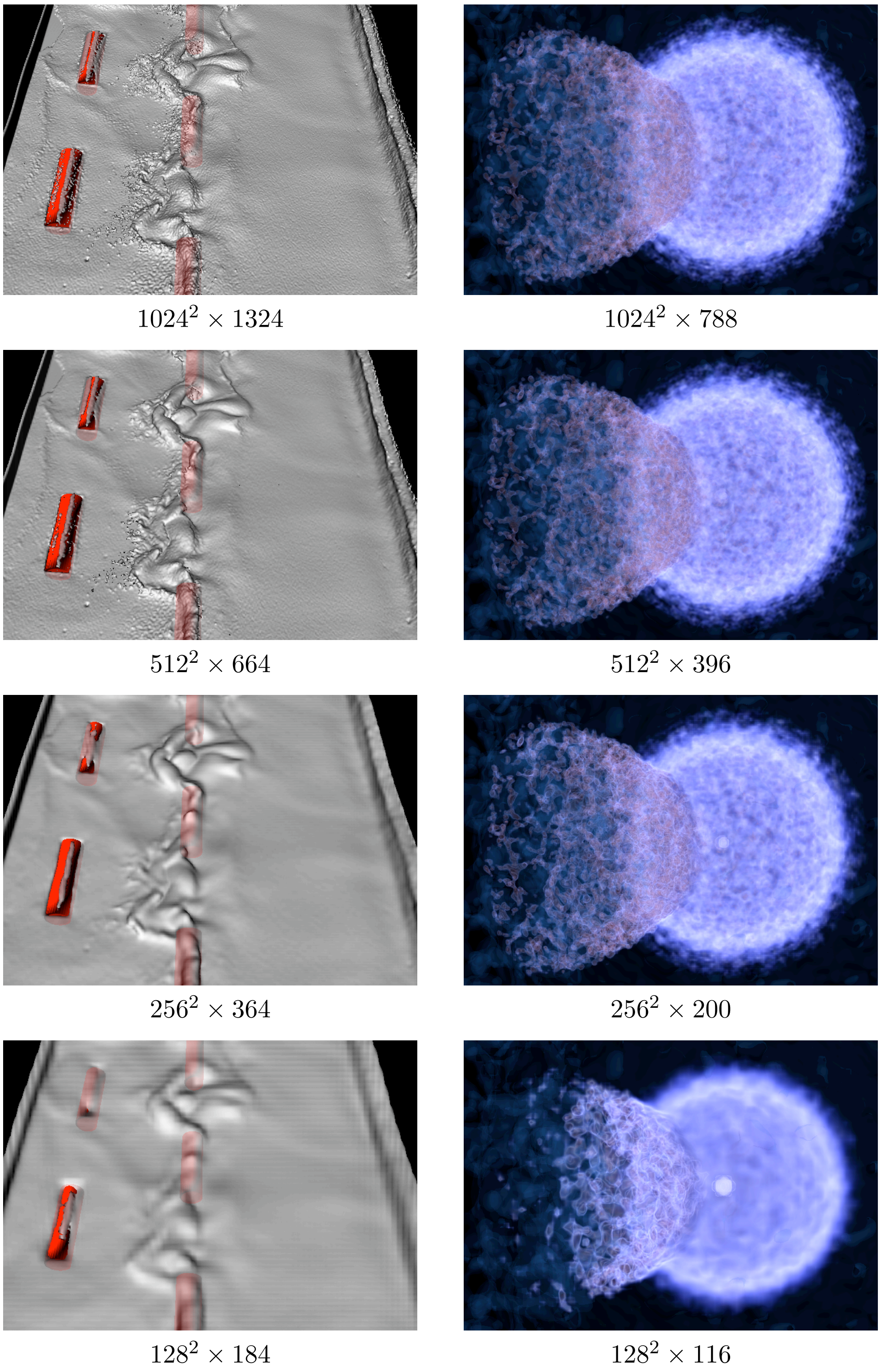


Figure 5.10: *Resampling to perspective grids of different resolutions with corresponding level of detail at a fixed viewport of* 1024^2. *Isosurface rendering of the LWSB data set (top row) and volume rendering of the ejecta data set (bottom row) at different grid resolutions.*

overlapping particles for each slab as described in Section 5.7.1. For settings equivalent to those in Table 5.2, the average rendering speed on a 512^2 viewport was 285 ms for the LWSB data set and 2116 ms for SNIaEjecta. The sorting pass took 12 ms and 18 ms, repecitvely, which is marginal compared to the whole rendering time. Compared to rendering with our multi-resolution hierarchy, the performance drawback is around 33% and it is obvious that this performance lack will further increase for larger data sets until a rendering without a LOD representation is not feasible. For example, the top right image in Figure 5.9 showing the Millennium Run would require 607 million particles instead of 48 million particles to be rendered without such a hierarchy, which would exceed by far the video memory and processing capabilities of modern GPUs.

Occlusion Culling

The performance gain due to occlusion culling depends on the depth complexity and the opacity of the rendered data. The effect becomes apparent if many particles are completely occluded. For example, in the scene shown in Figure 5.6 occlusion culling leads to performance gain of about 68%. In Figure 5.10 (left column) an increase of only 5% could be observed due to low depth complexity. In Figure 5.15 (left column row, middle image) the gain was only 17% due of high transparency. In extreme close-ups where most particles are occluded, however, we measured an overall performance gain of up to factor 8.

5.9 Comparison to Other Techniques

Finally, we have compared our rendering approach to existing techniques that are commonly used for the visualization of SPH data.

Order-independent Splatting

The images in Figure 5.11 have been rendered based on the order-independent emission-only model presented in Chapter 4 (a) and using perspective grid volume rendering with the emission-absorption model (b). As expected, the volume rendering gives a good depth impression of the data set, which is entirely missing in the left image. This difference becomes especially apparent for long viewing distances as used in Figure 5.11. Regarding performance it is no surprise that the depth-independent 2D splatting yields a much higher frame rate (8.1 fps) than the 3D reconstruction using the perspective grid (0.1 fps). In order to exploit the benefits of both approaches, Section 5.10 discusses a hybrid technique that is suited for the efficient rendering of very large data sets as the Millennium Run.

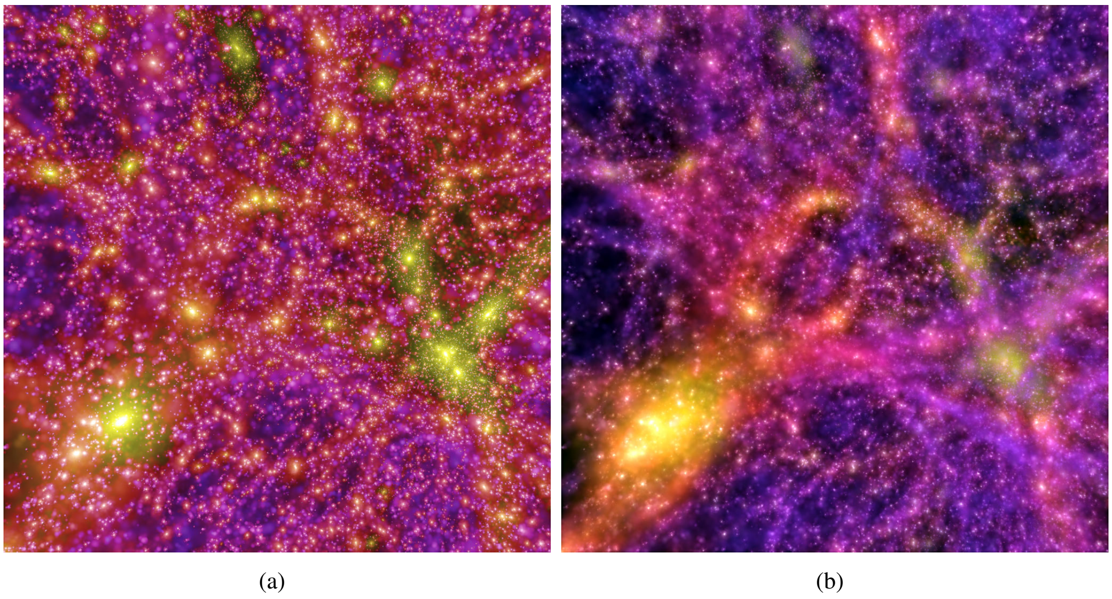


Figure 5.11: *Comparison of (a) order-independent splatting as described in Chapter 4 (8.1 fps) and (b) our perspective-grid ray casting technique (0.1 fps) for rendering 35 million particles of the Millennium Run at a* 1024^2 *viewport.*

Order-dependent Splatting

Another common technique for rendering SPH data is order-independent splatting. To compare quality and performance, we performed the particle sorting on the GPU using an optimized sort routine, which required less than 5% of the overall rendering time. Rendering the WDMerger data set (Figure 5.2(a)) was performed at 7.85 fps. Our technique, on the other hand, required 1.7 fps (Figure 5.2(b)), but achieves a much better quality.

Screen-space Techniques

In a third experiment we compared our approach to screen-space methods for the visualization of isosurfaces in SPH data. Such methods first render particles as spheres in an arbitrary order to obtain the visible surface parts and then generate a smooth surface from the resulting depth buffer imprint. Figure 5.12(a) shows the resulting image after the first pass for the LWMO data set. This image was generated at 12.5 fps. Figure 5.12(b) demonstrates an isosurface reconstructed by our approach. This image was rendered at 5.75 fps, and, thus, only requires about twice the time of screen-space methods. Given that our method provides high-quality isosurface and volume rendering, this seems to be a reasonable compromise.

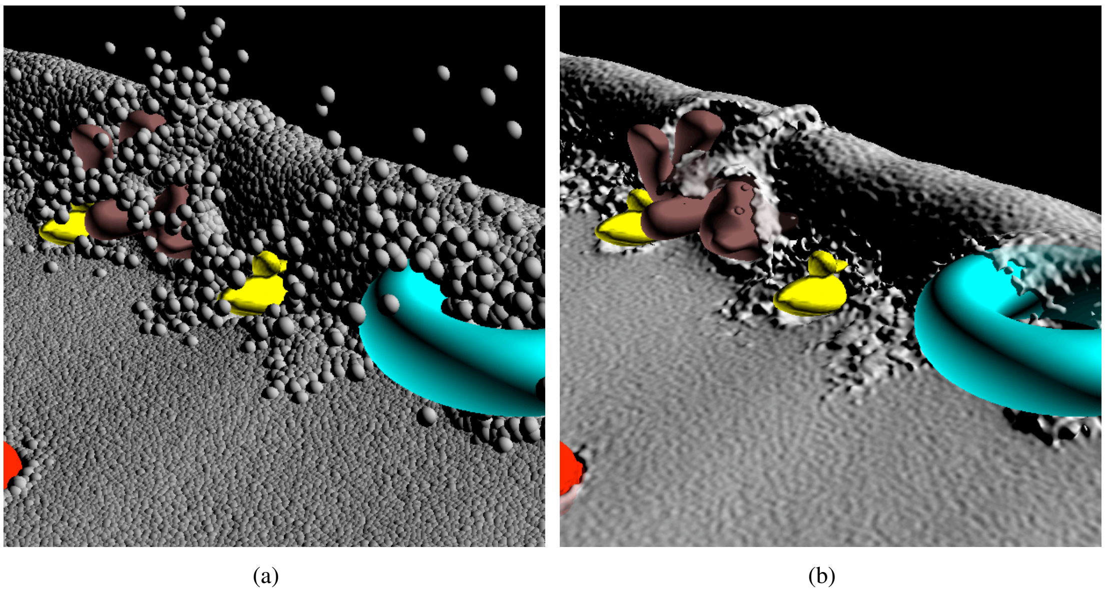


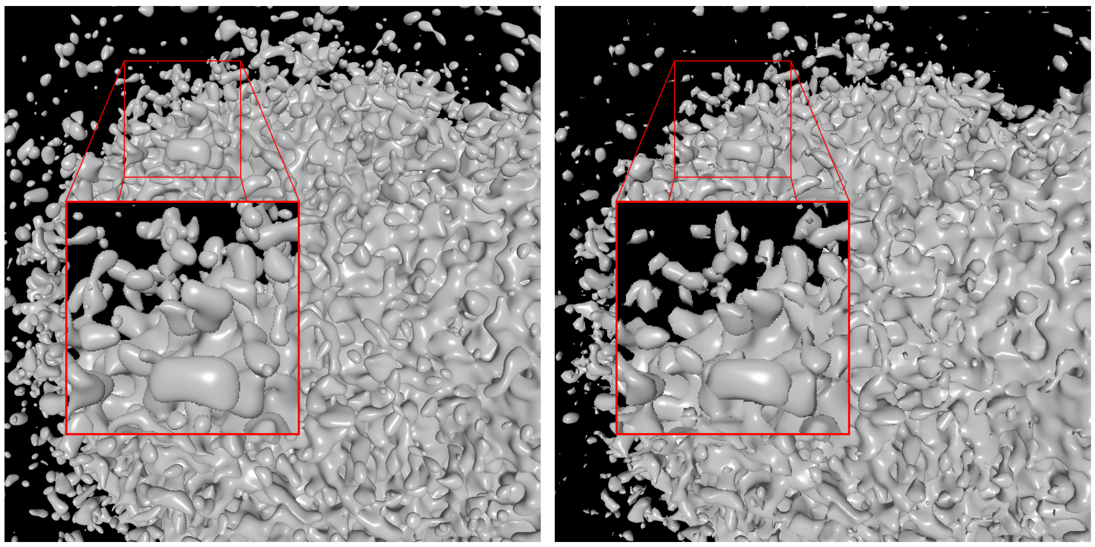
Figure 5.12: *Sphere rendering of the particle's support radius versus perspective grid-based isosurface extraction. Screen-space techniques smooth the surface given by the depth buffer imprint of the left image, while our approach on the right directly reconstructs any given isosurface.*

Volume Rendering of a Cartesian Grid

Another alternative to visualize SPH data is to resample the continuous scalar field given by the SPH particles onto a cartesian grid, which can then be rendered efficiently on the GPU using texture-based volume rendering [KW03, DZTS08]. The rendering quality of this approach depends on the resolution of the fixed discretization, which should be large enough to capture all important details. However, high-resolution volumes require a huge amount of memory as the data increases by a factor of 8 when doubling the sampling rate. This poses a problem since conventional volume rendering techniques provide efficient rendering only for data sets that fit into video memory, which indirectly limits the maximal resolution of the data set.

For a comparison of this approach to our perspective grid technique, we have created a volumetric data representation for a single time step of the SNIaEjecta data set. To account for the restricted video memory on GPUs, we focused on the density field only and quantized the resampled floating point values into 16 bit. We dismissed a further compression to 8-bit as it leads to significant quantization artifacts. For the resampling of the data, we chose a resolution of $880 \times 784 \times 736$. This results in a data volume of size 969 MB and is small enough to fit into the video memory of our target architecture, which has a capacity of 1024 MB. To reduce sampling artifacts, we calculated the value for each voxel by averaging 4^3 regularly distributed samples within the represented subdomain.

For rendering, we used texture-based volume ray casting and applied empty space skipping and early ray termination to optimize performance. Figure 5.13 shows an isosurface rendered with the perspective grid approach (a) in comparison to the texture-based rendering of the volumetric data set (b) for a zoom into the data set. As can be seen, Figure 5.13(b) contains significant artifacts and topological differences, which clearly illustrates that the volumetric resolution of the precomputed volume is not large enough for an adequate sampling of the density field. Regarding rendering performance, on the other hand, texture-based volume rendering is significantly faster (26.6 fps) than the perspective grid approach (0.2 fps) on a 1024^2 viewport. The large performance drawback of the perspective grid approach is due to the resampling that needs to be performed in every frame. However, the higher performance is of little value considering the clear quality penalties in the given example.



(a) (b)

Figure 5.13: *Comparison of (a) our perspective grid approach (8.7 million particles @ 0.2 fps) with (b) texture-based volume rendering of a resampled cartesian grid of resolution* $880 \times 784 \times 736$ *(26.6 fps) for rendering an isosurface on a* 1024^2 *viewport.*

Yet, there are sophisticated visualization systems that are able to render massive volumetric data sets in an out-of-core fashion based on a volumetric level-of-detail representation [LHJ99, GMI08]. Accordingly, more accurate volumetric data sets might be created that can be rendered with such techniques. However, even when using a multi-resolution hierarchy, the portions to be rendered will hardly fit into video memory of common GPUs for considerable viewports. Furthermore, already a volume of resolution 4096^3 results in 128 GB of raw data and the size can significantly increase by the additional acceleration structures. Thus, bandwidth limitations will heavily impact the visual exploration process of such a data set.

In contrast, the particle data comprises only 200 MB per time step[1], which is far away from those restrictions. Accordingly, our approach scales significantly better with increasing size of the data as has been shown at the example of the Millennium Simulation. This benefit is not only important for the rendering of large-scale particle sets, but becomes crucial when time-dependent data needs to be explored. Here, quick access to a large number of time steps is required, which can only be provided by a memory-friendly particle representation.

5.10 Efficient Approximation for Very Large Data Sets

As shown in the previous section, our approach allows interactive rendering for data sets containing millions of particles. In addition, the rendering performance can be further increased by choosing a grid resolution that is half the size of the viewport, which has shown to have only marginally impacts on the rendering quality for million-particle data sets (see Figure 5.10). For even larger data such as the Millennium Run, however, an interactive exploration is unfeasible due to the enormous geometry and rasterization load. To make matters worse, grids of reduced resolution have a much more severe impact than for the other data sets used in our experiments, since the high density regions that are of primary interest are often composed of particles around pixel size. Accordingly, these structures would be significantly blurred by using low-resolution grids.

Depth-independent 2D splatting as described in Chapter 4, on the other hand, yields interactive frame rates but lacks any depth information (see Figure 5.11). The main reason for the performance difference is the reduced geometry and rasterization load when particles are resampled in 2D instead of in 3D. To exploit this performance gain, we suggest to substitute the 3D voxelization for a 2D projection as described in Section

[1]containing 6 floating point values to store the particle's position, smoothing length, mass and temperature.

4.9, which is written into the z-slice that is nearest to the particle center. Accordingly, the result for each slice corresponds to the order-independent integration presented in the previous chapter. Subsequently, we apply the volume rendering pass, which finally blends the slices in a front-to-back order to approximate the depth information. Furthermore, the performance can be improved by using a significantly smaller number of z-slices within the perspective grid.

Figure 5.14 compares the result of this layered splatting scheme using 124 z-slices (a) with the high-quality rendering using the perspective grid technique, which requires 775 slices for an appropriate sampling (b). Even though the hybrid algorithm neglects the order within each slice, the differences of the images are only marginal. The primary reason is that due to the nature of the simulation and the applied color coding scheme, the local ordering is less significant since there are only few areas with high opacities and different colors in which changes in the particle order become apparent. The global order of larger low-density regions and smaller high density regions, on the other hand, is largely preserved. Even though the quality is very similar to the volume rendering approach, the performance is only about three times slower than the order-independent rendering scheme described in Chapter 4. This difference in performance is due to the additional ray casting pass that is required to blend the different layers, the overhead of rendering into a volumetric render target, as well as the high effectiveness of the presence acceleration for order-independent splatting.

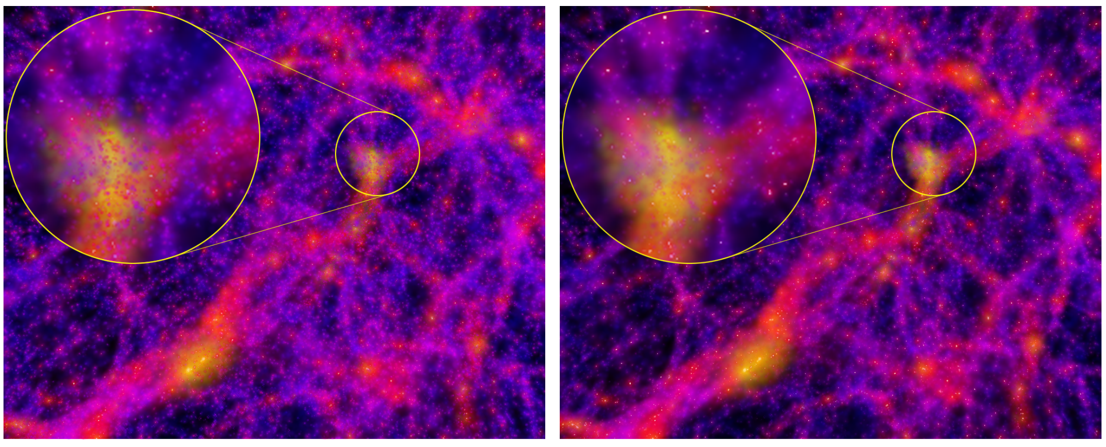


Figure 5.14: *Comparison of (a) the layered 2D splatting approximation (with 124 z-slices @ 2.3 fps) with (b) perspective volume ray casting (with 775 z-slices @ 0.1 fps) for the Millennium Run with 39 million visible particles at a* 1200×900 *viewport and with an corresponding slice resolution.*

One drawback of the approach is that particles are mapped to different depth slices during camera motions, which can result in a slight flickering on the screen. However, in our experiments these artifacts were hardly noticeable when we used at least one sixth of the number of depth slices that would be required for an isotropic sampling.

5.11 Conclusion and Future Work

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

To the best of our knowledge, for the first time we have shown that order-dependent resampling of million-particle SPH data can be performed at almost interactive rates. Compared to screen-space approaches for the rendering of isosurfaces in SPH data, our technique is only slightly slower but achieves higher quality. Furthermore, we have presented a simplified rendering technique that allows to efficiently visualize billion-particle data sets such as the Millennium Run in a high quality.

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

For the future, it would be interesting to pursue research on filtering techniques for data given on a perspective grid. Since even with some improvements isosurfaces in SPH data tend to look bumpy and can not adequately resolve flat structures, curvature-based smoothing as proposed in [vdLGS09] could be considered on the grid.

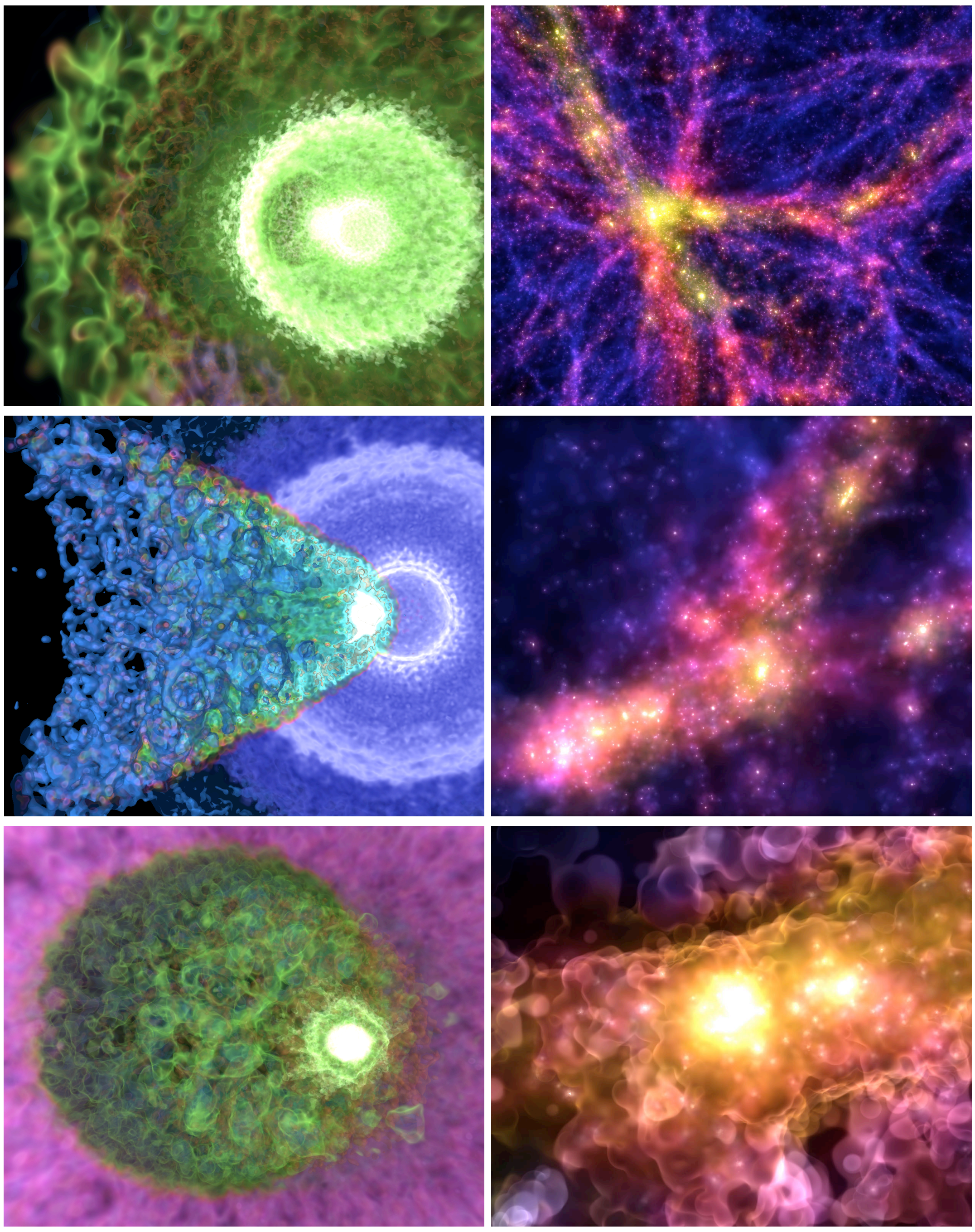


Figure 5.15: *Different transfer functions and camera settings reveal various details of the SNI-aEjecta (left column) and the Millennium data set (right column).*

Chapter 6

Motion Visualization in Large Particle Simulations

Interactive visualization of large particle sets is required to analyze the complicated structures and formation processes in astrophysical particle simulations. While some research has been done on the development of visualization techniques for steady particle fields, only very few approaches have been proposed to interactively visualize large time-varying fields and their dynamics. Particle trajectories are known to visualize dynamic processes over time, but due to occlusion and visual cluttering such techniques have only been reported for very small particle sets so far. In this chapter we present a novel technique to solve these problems, and we demonstrate the potential of our approach for the visual exploration of large astrophysical particle sequences. We present a new hierarchical space-time data structure for particle sets which allows for a scale-space analysis of trajectories in the simulated fields. In combination with visualization techniques that adapt to the respective scales, clusters of particles with homogeneous motion as well as separation and merging regions can be identified effectively. The additional use of mapping functions to modulate the color and size of trajectories allows emphasizing various particle properties including direction, speed, or particle-specific attributes such as temperature. Furthermore, tracking of interactively selected particle subsets permits the user to focus on structures of interest. We validate the effectiveness of our technique using multiple data sets that contain many millions of particles.

6.1 Introduction

For the visual analysis of unsteady flow fields, particle-based approaches, such as interactively seeding and tracking massless particles and constructing pathlines showing their motion, are a well-established class of techniques. In contrast to pure particle visualization, where particles are rendered as individual graphical primitives, these characteristic lines allow for an improved understanding of complicated motion patterns. This is because pathlines can depict the time history of the position of a particle in one image, at the same time providing additional exploration possibilities. For instance, geometric properties of the graphical objects used to render these lines can be modified according to the intrinsic flow properties. In this way local structures in the domain can be revealed quite effectively.

Characteristic flow lines are usually computed by seeding massless particles into the flow domain and using numerical techniques to integrate these particles through space based on the underlying flow field. For the visualization of grid-based Eulerian simulations, where the fluid velocities are observed at locations that are fixed in space, numerical integration requires the interpolation of velocities from these locations to progress a particle in space and time. In Lagrangian particle simulations, for instance Smoothed Particle Hydrodynamics (SPH), where individual fluid material volumes are carried with the flow, the quantities given at the discrete set of particles are smoothed using some kernel function to obtain a continuum.

Accordingly, in Lagrangian particle simulations of fluids, the features of the entire physical system can also be obtained by tracing the motion of the simulated material volumes itself. This motion is defined by the underlying physical model, and it describes the dynamics of particles that carry a mass, which is smoothed over a certain volume around the particle. Accordingly, the mass transport within the simulated domain is completely described by the motion of the entire particle set, which is not guaranteed for the tracing of massless particles. It is worth noting, however, that the simulated particles behave slightly different than massless particles, since they interact with neighboring particles through a repelling force that represents the material pressure and gravitational effects. Mathematically this means that the smooth function that results from the kernel interpolation is usually not equal to the particle quantities at the particle positions, yet this function will generally not be too far from these values. Consequently, the trajectories of SPH particle will be close to the trajectories of massless particles seeded into the continuous vector field given by the discrete particle set, but they will not exactly coincide with these trajectories.

In this work we address the problem of interactively exploring the motion paths of particles in astrophysical simulations via visualization. The time-dependent data sequences we explore are comprised of sets of particles that were simulated using cosmological N-body/SPH simulations. In these simulation, the motion of a large number of particles under their own self-gravity is followed using Newton's equations of motion. Each sequence shows the evolution of a set of particles over time, with the particles being represented by their position as well as the physical quantities they carry, such as mass, temperature, and volume.

At first glance, the given problem seems to be easy. By simply connecting corresponding particles in adjacent time steps via line segments or higher order curves, and by rendering the so constructed pathlines via computer graphics techniques, the motion paths of each particle over time can be visualized. However, in practice this approach is of limited use due to the following reasons: Firstly, the visualization of pathlines does not allow for a (relative) analysis of the particles' velocities. Secondly, by solely following individual particles on the simulation level, it is very difficult to analyze cluster formations and splits in the particle distributions. Thirdly, since the simulations we address are comprised of many millions of particles, occlusion effects and visual clutter thereof prohibit an effective visualization in terms of revealing the relevant motion structures. Furthermore, the massive amount of geometry that has to be rendered for visualizing the pathlines of all particles can easily exceed the capacities of even the most powerful graphics accelerators.

To overcome these limitations we propose a novel space-time hierarchy for representing time-dependent particle distributions at different scales. Based on this hierarchy, we present interactive visualization techniques for depicting the motion paths of single particles as well as coherent particle clusters. In addition to the direction, we can show the motion velocity along these paths via animated color maps or geometry structures that move according to the velocity. In particular, this chapter makes the following specific contributions:

- A space-time hierarchy for particle sets.
- A level-of-detail approach for the visualization of particle motions via pathlines.
- Enhancement of regions of interest via user-defined mapping functions to modulate the appearance of trajectories according to cluster properties or via interactive selection and tracking of particle subsets.

Once the space-time hierarchy has been built on the CPU in a pre-process, the visualization runs entirely on the GPU and provides interactive rates even for high resolution particle sequences. In combination with in-core and out-of-core strategies to stream particle sets that are too large to fit into GPU memory, rates of more than 40 frames per second can be achieved for time-varying data sets consisting of millions of particles.

The space-time hierarchy can be used for representing the mass flow on different levels of detail, giving rise to an effective analysis of physical processes such as colliding and merging mass structures (see Figure 6.1). Furthermore, it enables level-of-detail visualizations of particle trajectories to reduce the amount of graphical primitives to be rendered. Notably the performed clustering can more generally be used to create a time-dependent multi-resolution hierarchy of the particle field, by merging particles according to the resampling operators presented for adaptive SPH simulations[DC99, APKG07]. This hierarchy can then be used to combine visualizations of particle trajectories with multi-resolution visualizations of the particle density field, or any other quantity carried by the particles as presented in Chapter 5 (see Figure 6.2)

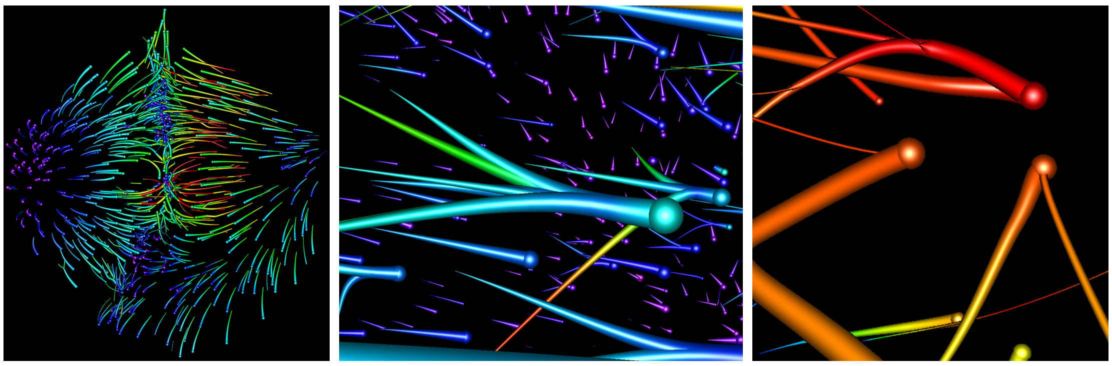

(a) (b) (c)

Figure 6.1: *Our approach allows to classify the motion dynamics of particle-based simulations based on directional information of particle flows at multiple scales and can be used to analyze formation processes in astrophysical particle simulations, such as the merging of particles into clusters. The left image shows a coarse level of detail for the Aquarius data set, which contains of 18.5 million particles, and the other two images are close-ups on particle mergers at finer resolutions. All images were taken on a* 1440×900 *viewport at more than 600 fps.*

The remainder of this chapter is organized as follows. In the next section we review previous work that is related to ours. The proposed space-time hierarchy for particle sets is introduced in Section 6.3. Section 6.4 presents the scale-dependent mapping

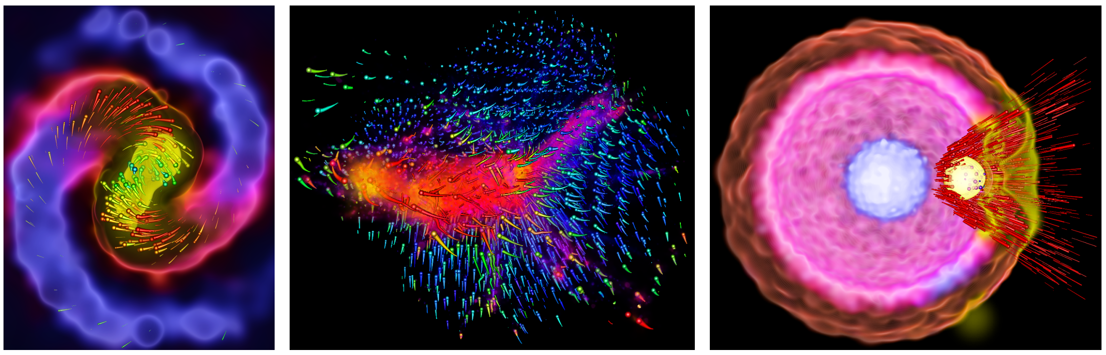


Figure 6.2: *The proposed space-time hierarchy can not only be used for the motion visualization of particle clusters via pathlines but can also serve as a time-dependent level-of-detail representation of particle distributions. Accordingly, the motion visualization can be directly coupled with volume visualizations of the scalar fields as proposed in Chapter 5. The images show three different astrophysical data sets: (a) WDMerger, (b) Aquarius, and (c) SNIaEjecta.*

techniques we propose to effectively visualize motion clusters in these sets. In Section 6.5 we evaluate the effectiveness of our approach for the visual analysis of particle simulations, and we analyze the performance of our approach. Finally, we give an outlook on further applications for our technique, followed by a brief summary of the work presented in this chapter.

6.2 Related Work

Particle-based visualization techniques have been studied extensively over the last years, mainly in the context of flow visualization. There is a vast body of literature related to this field and a comprehensive review is beyond the scope of this thesis. However, Post et al. [PVH+02], Laramee and Hauser [LH05], and McLoughlin et al. [MLP+10] discuss the basic principles underlying such techniques and provide many useful algorithmic and implementation specific details.

Only very little work has been reported on the application of flow visualization techniques for visualizing time-varying particle data. Most commonly, such sequences are rendered by visualizing the continuous scalar fields represented by the particle quantities in succession and revealing the motion dynamics via the evolution of the so constructed fields. Exemplary visualization techniques are the order-independent splatting scheme and rendering based on the perspective grid that are presented in Chapters 4 and 5, where we have also discussed related work in this field.

To the best of our knowledge, only very few approaches have explicitly addressed the efficient visualization of particle pathways in particle based simulations. In the work of Schindler et al. [SFBP09] the focus was on the visualization of vortex core lines in flow fields given by SPH simulations. One major contribution was that all computations were performed directly on the SPH representation to achieve high quality and consistency with this representation.

Our work is inspired by multi-scale techniques for time-varying scalar fields, as proposed, for instance, in [Wes95, LPD$^+$02, WS09b, WS09a]. Underlying these approaches is some form of multi-resolution representation to classify distribution characteristics in space as well as temporal motion characteristics. Even though we do not explicitly encode the particles' motion trajectories at different time scales, by performing motion-based clustering in combination with trajectory visualization a very similar analysis of temporal activities in particle data is achieved.

6.3 Space-Time Hierarchy

Our construction of a space-time hierarchy for time-dependent particle data is related to clustering methods for vector fields [GPR$^+$04, TvW99, GPR$^+$00, HWHJ99, MLD05] in that it considers similar merging criteria for clustering particles. In contrast to these techniques, however, our objective is not to approximate a given flow field by a sparse set of vector samples, but to hierarchically describe the distribution and flow of mass over time that is represented by the motion of individual particles. For this purpose, we introduce an estimate for the error in this transport when approximating a particle by a cluster. This error estimate is used as a metric to guide the hierarchical clustering of the particles in the initial time step, and to track splits and mergers of clusters over time (see Figure 6.3). The constructed space-time structure encodes the particle sets of all time steps hierarchically. This hierarchy is stored in a spatially-adaptive octree data structure, attributed by additional information to keep track of the clusters' evolutions over time.

6.3.1 Error Estimation

Since at the core of our method is the merging of particles into clusters and the splitting of clusters once their particles start to diverge, a criterion is required to steer these operations. The criterion we propose is based on the error between the trajectory of a particle and the trajectory of the cluster into which this particle should be merged, or which contains this particle and should be split.

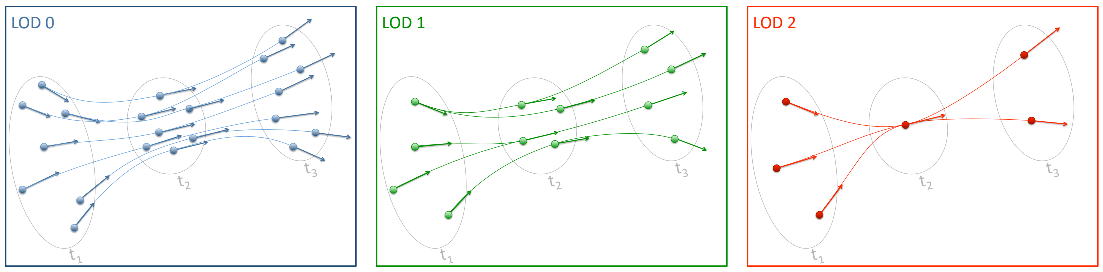


Figure 6.3: *Illustration of our space-time hierarchy for particle trajectories with 3 time steps and for 3 levels of detail.*

To generate a smooth trajectory from one particle's position at time step i to its position at time step $i+1$ we interpolate smoothly between these two positions using a cubic Hermite spline:

$$\begin{aligned}\mathbf{r}(t) &= (2t^3 - 3t^2 + 1) \cdot \mathbf{r}_p^{(i)} + (t^3 - 2t^2 + t) \cdot \mathbf{v}_p^{(i)} + \\ &\quad + (-2t^3 + 3t^2) \cdot \mathbf{r}_p^{(i+1)} + (t^3 - t^2) \cdot \mathbf{v}_p^{(i+1)}. \qquad (6.1)\end{aligned}$$

Here, $\mathbf{r}_p^{(i)}$ and $\mathbf{r}_p^{(i+1)}$ are the particle's positions at the current and the next time step, respectively, $\mathbf{v}_p^{(i)}$ and $\mathbf{v}_p^{(i+1)}$ are the corresponding tangents, $t \in [0,1]$ is the interpolation factor, and $\mathbf{r}$ is the interpolated position. The tangents are given by the particle velocities.

When we merge a particle into a cluster, we essentially replace the particle trajectory by the trajectory of that cluster. The trajectory of a cluster is also computed as a Hermite spline from adjacent cluster positions. Let us assume that a cluster has control points $\mathbf{r}_C^{(i)}$ and $\mathbf{r}_C^{(i+1)}$ and tangents $\mathbf{v}_C^{(i)}$ and $\mathbf{v}_C^{(i+1)}$. Then, the error function expressing the spatial deviation between the particle trajectory and the cluster trajectory is given by

$$\begin{aligned}\epsilon(t) &= \big|\big|\, (2t^3 - 3t^2 + 1)(\mathbf{r}_C^{(i)} - \mathbf{r}_p^{(i)}) + (t^3 - 2t^2 + t)(\mathbf{v}_C^{(i)} - \mathbf{v}_p^{(i)}) + \\ &\quad + (-2t^3 + 3t^2)(\mathbf{r}_C^{(i+1)} - \mathbf{r}_p^{(i+1)}) + (t^3 - t^2)(\mathbf{v}_C^{(i+1)} - \mathbf{v}_p^{(i+1)})\, \big|\big|.\end{aligned}$$

The maximum of this function is used to decide whether merging a particle yields an acceptable approximation of its trajectory or not. To avoid calculating the maximum, we derive an upper bound $\tilde{\epsilon}$ for the maximum and use this bound as a reasonable estimate. Based on the observations that the sum of the weights of the control point positions always evaluates to 1 and the maximum weight of each tangent is $4/27$, the following bound is obtained:

$$\tilde{\epsilon} = \max\big(||\mathbf{r}_C^{(i)} - \mathbf{r}_p^{(i)}||, ||\mathbf{r}_C^{(i+1)} - \mathbf{r}_p^{(i+1)}||\big) + \frac{4}{27}\big(||\mathbf{v}_C^{(i)} - \mathbf{v}_p^{(i)}|| + ||\mathbf{v}_C^{(i+1)} - \mathbf{v}_p^{(i+1)}||\big).$$

6.3.2 Particle Merging

Based on the error estimate that has been derived for quantifying the maximum distance between trajectories, we now describe how this estimate is used to guide our clustering process, and how the properties of a cluster based on the particles contained in this cluster are obtained. For the latter, we apply the resampling operators for adaptive SPH simulations [APKG07, DC99], which require the volume of the cluster to be the sum of the volumes of the merged particles, while any other vector or scalar quantity is averaged according to the particles' mass contributions.

Due to this definition, a cluster's position $\mathbf{r}_C^{(i)}$ and velocity $\mathbf{v}_C^{(i)}$ are not constant over time, but they have to be updated whenever a new particle is integrated. Accordingly, in a merging operation it is not sufficient to test whether a particle can be merged into a cluster, but it also has to be verified that after merging the particle and updating the cluster all particles contained in it are still well represented with respect to the error estimate. Instead of computing the error estimate separately for every particle of a cluster, we use a conservative estimation of the maximum error of the cluster members. The basic idea is to memorize a former position and velocity of the cluster along with the maximum error of the particles with respect to these values. As long as the difference between the former representation and the new cluster properties, plus the memorized maximum error does not exceed the permitted error threshold, the merging of the particle is valid.

More formally, let $\hat{\mathbf{r}}_C^{(i)}$ and $\hat{\mathbf{v}}_C^{(i)}$ be some former position and velocity of a cluster C. In correspondence to the error estimate $\tilde{\epsilon}$, where the estimated error for each represented particle depends only on the error regarding the position within one of the involved time steps, we separately memorize the maximal errors of all particles in the cluster that result from considering these two options, the positional error of the previous time step $(i-1)$

$$\theta_C^{(i)} = \max\left(||\mathbf{r}_{C'}^{(i-1)} - \mathbf{r}_p^{(i-1)}|| + \frac{4}{27}\left(||\mathbf{v}_{C'}^{(i-1)} - \mathbf{v}_p^{(i-1)}|| + ||\hat{\mathbf{v}}_C^{(i)} - \mathbf{v}_p^{(i)}||\right) \,\middle|\, p \in C\right)$$

and that of the current time step (i)

$$\delta_C^{(i)} = \max\left(||\hat{\mathbf{r}}_C^{(i)} - \mathbf{r}_p^{(i)}|| + \frac{4}{27}\left(||\mathbf{v}_{C'}^{(i-1)} - \mathbf{v}_p^{(i-1)}|| + ||\hat{\mathbf{v}}_C^{(i)} - \mathbf{v}_p^{(i)}||\right) \middle| \, p \in C\right).$$

Here, C' denotes the cluster that contains particle p at time $(i-1)$. Given a new cluster center $\mathbf{r}_C^{(i)}$ and velocity $\mathbf{v}_C^{(i)}$ that would result from merging an additional particle, the new maximum error within the cluster is estimated by

$$\epsilon' = \max\left(\theta_C^{(i)}, \delta_C^{(i)} + ||\mathbf{r}_C^{(i)} - \hat{\mathbf{r}}_C^{(i)}||\right) + \frac{4}{27}||\mathbf{v}_C^{(i)} - \hat{\mathbf{v}}_C^{(i)}||$$

where the additional error regarding the position influences $\delta_C^{(i)}$ but not $\theta_C^{(i)}$, since the latter considers only the positional error within the previous time step, and the error with respect to the velocity is added to both error terms. If this estimate is below the maximum permitted error, the particle can be merged with the cluster. Otherwise, we reset $\hat{\mathbf{r}}_C^{(i)}$ and $\hat{\mathbf{v}}_C^{(i)}$ to $\mathbf{r}_C^{(i)}$ and $\mathbf{v}_C^{(i)}$, and determine the new maximum error terms $\theta_C^{(i)}$ and $\delta_C^{(i)}$ to recalculate ϵ'. If the error still exceeds the threshold, no merging is performed. Otherwise, the particle is integrated into the cluster. For performance reasons, the update of $\theta_C^{(i)}$ and $\delta_C^{(i)}$ are skipped when the values are close to the maximum permitted error. Nevertheless, particles may still be inserted if the update of $\mathbf{r}_C^{(i)}$ and $\mathbf{v}_C^{(i)}$ is small enough so that ϵ' does not exceed the threshold.

If clusters are built by merging clusters, which have already been formed at a finer level of detail and which already introduce errors with respect to position and direction, we adapt the calculation of $\theta_C^{(i)}$ and $\delta_C^{(i)}$ such that we add the maximal errors within such a cluster $\theta_p^{(i)}$ and $\delta_p^{(i)}$ to the respective error estimates.

6.3.3 Hierarchy Generation

Based on the proposed particle merging strategy, the particle space-time hierarchy can now be constructed. The hierarchy is comprised of one adaptive octree per time step, each of which is created in a bottom-up fashion. In the hierarchy of each time step, the particles are clustered consecutively up to a maximum error for the given level. The maximum permitted error for the root level is defined by the user, and according to the octree data structure it is halved for each additional level of detail. Additionally, we store the particle trajectories that map the clusters of one time step to the clusters of the next time step.

Initial Time Step

At the first time step, we start by storing the original particles at the leaves of the octree. Afterwards, we successively pass all particles to the next coarser level. For each particle that is inserted we look for potential merging candidates and add each pair of particles that can be merged into a priority queue that is ordered according to our error estimate. To accelerate the neighbor search, we organize the particles within each octree node in a uniform grid with a spacing that corresponds to the error threshold of the specific level. Thus, at most 27 of such bins have to be searched to find the candidate pairs. When all particles are inserted into the current level, we successively merge the particle pair with the smallest distance according to our error estimate. For each merger, the octree as well as the priority queue are updated accordingly. This process is repeated using the next candidate pair until the priority queue is empty. Note that in the special case of the initial time step there are no trajectory to be represented, but just the starting points of trajectory in space and time. Accordingly, the error terms regarding the position and velocity of the previous time step are set to zero.

Subsequent Time Steps

For all successive time steps, we again begin by storing the particles into the leaf nodes of the octree. Then, however, we do not proceed by passing particles from one level to the next coarser one, but we try to merge particles into the clusters of the previous time step for all levels of detail. If due to a merging operation the error estimate is violated for a cluster, it is iteratively split into sub-clusters until the merging criterion is met. These sub-clusters are then inserted into the respective level of the octree. As before, we search for merging candidates within the neighborhood and add each mergeable pair into the priority queue. After insertion of all clusters, we successively merge the clusters as before until the priority queue is empty. Note that by propagating the cluster information to the next time step, we preserve continuity over time and accelerate the merging process by reducing the number of neighboring particles that have to be tested.

The splitting of clusters needs some further explanations. As mentioned before, we start by calculating the error estimate for all particles a cluster is comprised of, in order to verify whether the cluster will remain unchanged. If a cluster has to be split, it is separated into sub-clusters by iteratively applying a PCA-based split algorithm [PGK02, LGK^{+}01]. That is, we determine the optimal splitting plane by computing the centroid of the particles' positions and the largest eigenvector of the auto-covariance matrix as surface normal (see Figure 6.4). Since we aim at minimizing the error estimate at the current time step, the PCA split is performed in the 6-dimensional error space that is given by the particle positions and velocities, where the latter vector is scaled by $4/27$ according to Equation 6.2. If the merging criterion is still violated within a newly formed sub-cluster, it is split as well.

In addition to the clusters that are build for the distinct time steps, we store the trajectories that result from the propagation of the clusters over time including cluster splits and mergers. Each trajectory consists of the cluster indices that define its start and end points within the successive time steps as well as the mass and radius of the represented cluster.

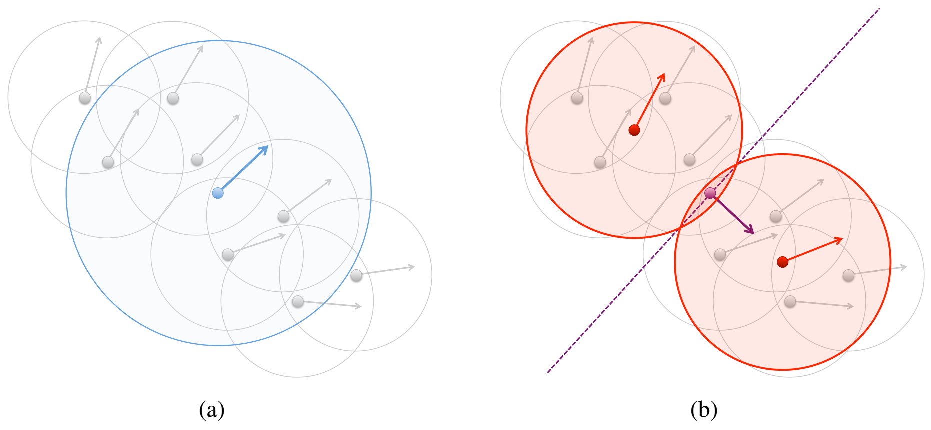


Figure 6.4: *For the splitting of a cluster that violates the merging rule in the successive time step (a), we apply a PCA split to build subclusters (b).*

6.3.4 Data Structures

The described preprocess results in one adaptive octree per time step, including particle and cluster data as well as the cluster trajectories representing the particle motion between successive time steps. The octrees serve as data structures to hierarchically represent the spatial distribution of particles at the discrete set of time steps. We now augment the octrees with additional information to efficiently determine for a node the spatial region that is traversed by the particles stored at this node between the current and the next time step, yielding a continuous space-time partitioning data structure. Accordingly, successive nodes must contain the data that is required for the interpolation of particle trajectories within the enclosed space and time. This is usually true for most clusters but not for those that have left the domain of a node within the represented time. To allow for the interpolation between successive nodes, we insert a *cluster copy* of the end point of such particles into the successor of the starting point's node and adapt the bounding box of the node accordingly. Such an update is also required if a particle or cluster leaves the domain temporarily.

For each node of the space-time hierarchy, we finally store three data structures. The *cluster list* contains the clusters built at the distinct time steps and stores any quantity that is interpolated between two time steps including the control points for the spline interpolation. The *trajectory list*, on the other hand, represents the information of the actual flow of mass between successive time steps, including the amount of mass itself, the radius of the cluster, and, most importantly, references to the control points of the trajectory. These are given by the indices in the cluster lists of the octree node and its successor (see Figure 6.5). The trajectory list is sorted by the cluster index of the first control point to allow for an efficient tracking of particles (see Section 6.4.1). To further support the tracking of particles, for each cluster copy we store the node ID and cluster list index of the original cluster in a *copy list*.

On the finest level of detail, which contains the individual particles of the simulation, merging and splitting of particles does not occur and the trajectory list has the same entries as the cluster list of the same time step. Thus, the index within the trajectory list into this cluster list is redundant and does not need to be stored.

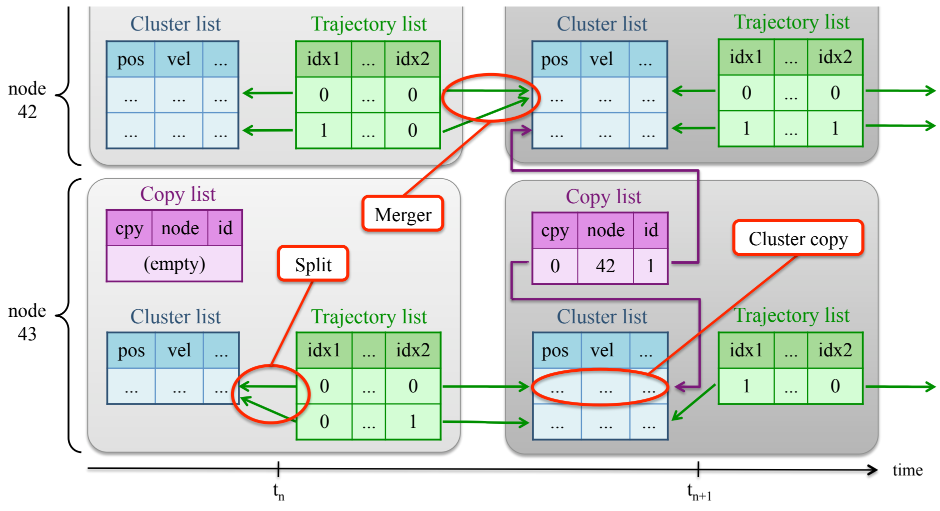


Figure 6.5: *Illustration of the data structures of our space-time hierarchy. The clusters of each time step are stored in a cluster list. The trajectory list contains the propagation of clusters over time with idx1 and idx2 referencing the end points for interpolation. The copy list helps to track clusters that left the domain of the octree node at this time step. In this illustration the copy list entry denotes that one particle moves from node 43 to node 42 between the time steps t_n and t_{n+1}.*

Since the mass is given per trajectory, mergers and splits can lead to discontinuities during rendering when the size of the trajectories is chosen dependent on their mass (see Figure 6.6(a)). To avoid this, we additionally store for each cluster in the cluster list the maximum mass of all outgoing trajectories and adapt the size of the ingoing trajectories close to the merger or split (see Figure 6.6(b)).

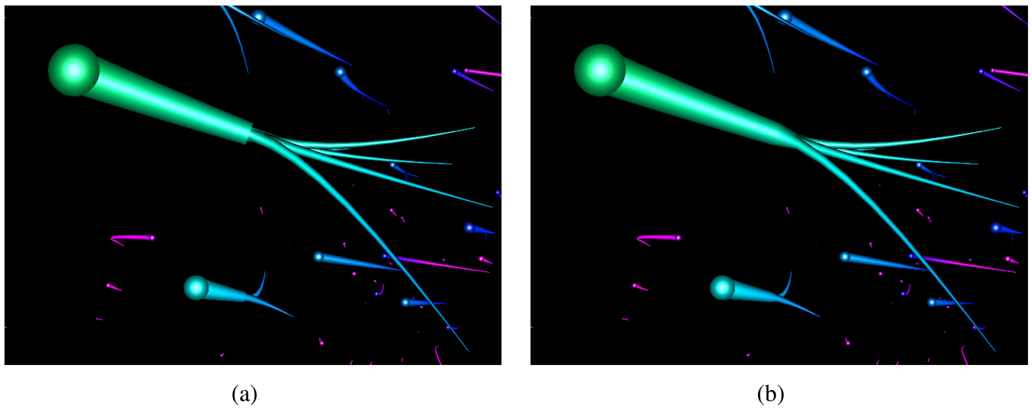


Figure 6.6: *(a) Discontinuities of trajectories at mergers and splits can arise when the size of each trajectory depends on the cluster's mass. (b) By storing the maximum mass of the subsequent trajectories the size can be adapted at the connection.*

6.4 Rendering

The interpolation of clusters between successive time steps and the rendering of cluster trajectories is entirely performed on the GPU. Therefore, the data of all visible octree nodes are transferred to the GPU's local memory. The trajectory list of each octree node is bound as a vertex buffer and the cluster lists of this and the successive time step are bound as buffer resources. Based on the indices given per vertex, the position and tangents of the trajectories' end points can be accessed, and they are then used to interpolate the cluster's position and velocity by cubic Hermite splines. The mass and (if required) the radius of the cluster are given per vertex. Particle trajectories are approximated by piecewise linear segments that are adaptively refined in the geometry shader by interpolating intermediate spline positions. To give a better impression of the shape and direction of the trajectory, the line can additionally be extruded to a tube or arrow.

6.4.1 User Interaction

Especially in large SPH simulations with many millions of particles, a scientist needs to be able to efficiently guide the exploration process and focus on the structures of interest. For this purpose, the visualization of the time-space hierarchy can be enriched with additional features to graphically emphasize certain regions or properties.

LOD Selection

Rendering is performed for all visible octree nodes and for all successive time steps that are covered by the current trajectory, i.e., all time steps that are required for rendering the trajectory. Since this data contains clusters and trajectories at different resolution levels, a mechanism is provided that allows the user to manually select a specific level of detail for the entire data set. By interactively switching between different levels of detail while navigating through space and time, a quite effective exploration of particle trajectories is already possible.

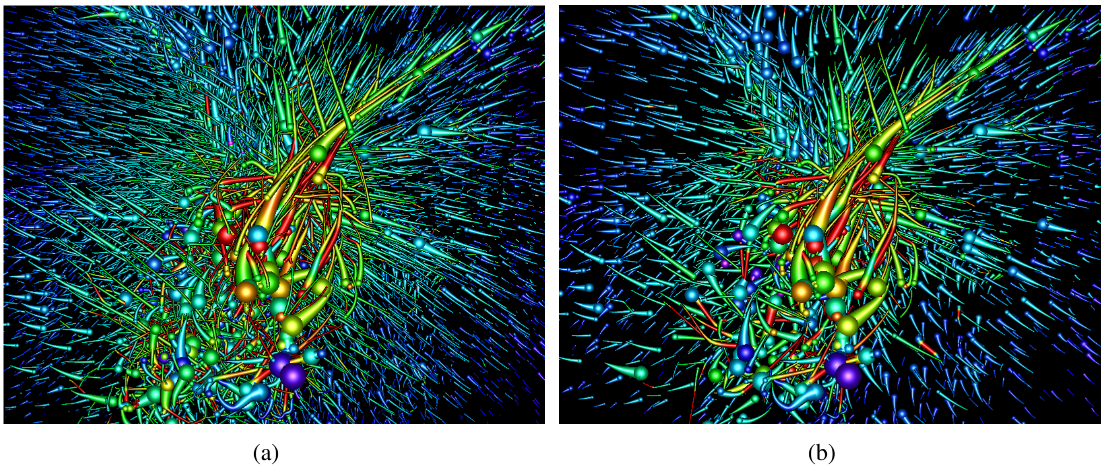


Figure 6.7: *(a) Fixed LOD (46 fps) in comparison to a (b) view-dependent LOD (65 fps) on a 1440×900 viewport.*

A drawback of this approach is that choosing a fine level of detail for a close-up view into a certain region can lead to very unpleasant visual clutter further away from the viewer (see Figure 6.7(a)). As an alternative, the octree nodes can be rendered automatically at the coarsest level of detail that maintains a given screen space error. This error is given by the error that is caused by replacing particles by the cluster in the current node (see Section 6.3.1), and the distance of this node to the viewer. Thus, the particle trajectories near the camera are automatically visualized in more detail, whereas the particle motions far away from the camera are represented via few, but larger clusters. This leads to good perception of the particle flow in the entire view frustum (see Figure 6.7(b)).

One drawback of an adaptive LOD selection is that discontinuities might be visible for particle trajectories that run through neighboring octree nodes with different levels of detail, and which correspondingly do not share control points between these levels. In practice, however, we measured that less than 0.2 % visible trajectories showed such discontinuities, and since they appear at the borders between different levels of detail, usually they show up only in the background.

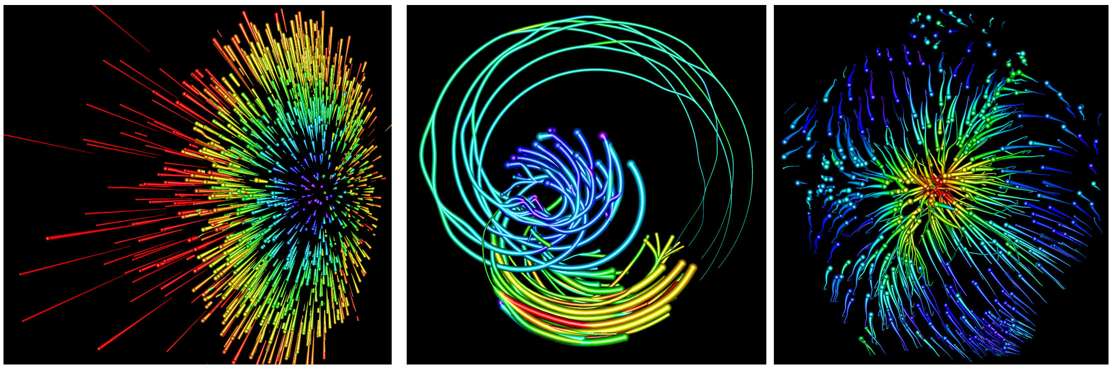
Besides a view-dependent LOD, the appropriate level of detail might also be chosen as part of *focus & context* techniques, which allow a high detailed insight in a user-defined focus region while preserving context information of the surrounding space via a low-resolution visualization.

Tracking of Particles

An important feature requested by astrophysicists is the possibility to define a specific set of particles and to track the position of only these particles over time. The selection itself can be performed via picking of one or more trajectories on the screen, based on a user-defined region of interest (e.g. a bounding box), or the cluster of interest can be retrieved from a pre-computed cluster catalogue. For the picking of clusters, we write the time step, the ID of the octree node as well as the index of the trajectory to an off-screen render target in order to identify the corresponding cluster of each trajectory on the screen. In case of an interactive spatial selection, we render the particles of all octree nodes that intersect the region of interest. For all particles within this region, the octree's node ID as well as the trajectory list index is written to a stream output buffer and read back to main memory.

The indices of the selected trajectories are stored in *tracking lists* of the corresponding octree nodes. To propagate the cluster selection over time, we determine the cluster list indices of the trajectory end points in the successive time step on the CPU. Based on these cluster indices, we further determine the corresponding trajectory list indices by a binary search and add the selected indices to the node's *tracking list*. Duplicates that result from the merging of particles can be discarded by using a set data structure for the tracking list. The tracking of clusters which leave the domain of an octree node is propagated to the corresponding octree node within the same time step by using the copy list (see Section 6.3.4). For typical selections with up to a few thousand of particles, the propagation on the CPU has no perceivable impact on the rendering performance. If needed, however, the Propagation could be performed in an additional thread or parallelized on the GPU.

The trajectories of the selected particles can be exclusively visualized by using the tracking list as index list for rendering (see Figure 6.8). As an alternative, the particles and their trajectories can be rendered together with the unselected data, but highlighted, e.g. by color. We realize this feature by tagging the selected entries of the trajectory lists using the sign bit of the mass property.



(a) (b) (c)

Figure 6.8: *Examples for user defined tracking of particles showing a cross section of the SNIaEjecta data set (a), user-selected clusters over 30 time steps in the WDMerger data set (b) and a subset of the aquarius data set (c). The colors of the trajectories represent the cluster's velocities (a) and temperature (b/c).*

Trajectory Appearance

Another possibility to highlight a certain set of sub-clusters is to modulate the appearance of the trajectories according to certain cluster properties. In our standard setting, the size of the trajectory is set such that the area of the cross-section represents the mass of the according cluster. To avoid extreme differences in size for coarser LODs, the user can set a maximum mass threshold to clamp the radius of the trajectories. On the other hand, the trajectories of small clusters can disappear when their diameter falls below the pixel size. To preserve these trajectories the user can define a minimum mass threshold for which cluster trajectories are automatically enlarged to pixel size. On the contrary, if the user wants to focus on the primary motions within the data field, trajectories of clusters below a certain number of particles may also be discarded.

The color of the trajectories can be determined by using arbitrary cluster properties, such as direction, velocity, mass, or data-specific attributes such as temperature as input for a color map. Thus, the visualization can be enhanced by emphasizing the differences of cluster trajectories (see Figure 6.8). To enhance the time and velocity along the trajectories, we have integrated animations via highlights or additional geometric primitives with the speed being configurable by the user (see Figure 6.9).

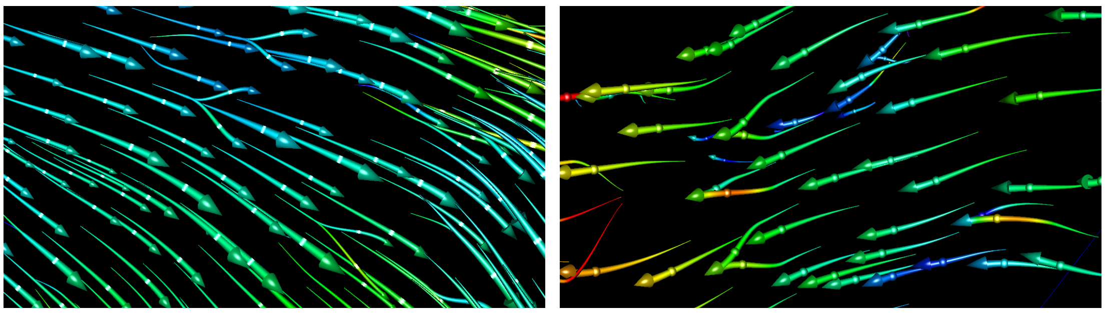

(a) (b)

Figure 6.9: *Animated highlights (a) or additional geometric primitives (b) repeatedly running along the trajectories can be used to improve the impression of motion along a fixed time period.*

6.4.2 Implementation Details

We have implemented the presented technique using C++ and DirectX 10. To optimize the data management on the CPU, we have used asynchronous I/O in combination with prefetching of octree nodes in space and time. When nodes are streamed into main memory, they are doubly linked along the time domain. By this means, the octrees of the required time steps can be traversed collectively in order to gather the octree nodes required for rendering. View frustum culling is used to omit invisible parts of the data set. On the GPU, we have implemented buddy memory allocation [Knu97] to avoid time-consuming creation and deletion of resources per node. For the rendering of trajectories as spheres with tails, we have adapted the approach of Merhof et al. [MSE$^+$06] to approximate tubes by a hybrid rendering of triangle stripes and point sprites. Furthermore, we added black halos with a size of 1 pixel around the trajectories to enable a better differentiation of the clusters. For more complex trajectory heads such as a cone, we pre-compute a *sprite texture atlas* with a set of views of the geometry under different rotations [GGS02, KKKW05] . By rendering the appropriate point sprite instead of the original geometry, the vertex load can be significantly reduced. To avoid artifacts for intersecting trajectories, we additionally adapt the depth value of each pixel according to the depth print within the sprite atlas and by approximating the depth offset for the trajectories. Volume rendering of the SPH data for the hybrid visualization was implemented using perspective grid ray casting as proposed in Chapter 5.

6.5 Results

Our experiments have been carried out on a standard PC equipped with an Intel Core 2 Quad Q9450 2.66 GHz processor, 8 GB of RAM, and a NVIDIA GeForce GTX 480 with 1.5 GB video memory and PCI Express 2.0 x16. We demonstrate the efficiency of our approach for three different astrophysical particle simulations. *WDMerger* simulates the merger of a white dwarf binary system. *SNIaEjecta* simulates the impact of a supernova ejecta on a companion star. The *Aquarius* data set simulates the evolution of a MilkyWay-sized dark matter halo from shortly after the big bang to the present. In all of these simulations, each particle has a size of 40 Byte containing a unique ID, position and velocity, as well as additional floating point attributes, such as smoothing length, mass, and temperature. Table 6.1 gives detailed information about the data sets.

Table 6.1: *Data sets used in our experiments.*

Data set	Time steps	# Particles	Data size
WDMerger	84	2,000,000	6.3 GB
SNIaEjecta	49	8,745,571	16.0 GB
Aquarius	25	18,535,972	17.3 GB

Space-Time Hierarchy

For each of the three data sets, we created the proposed space-time hierarchy with 6 levels of detail. This preprocess took 34 minutes for WDMerger, 101 minutes for SNIa-Ejecta and 115 minutes for the Aquarius data set. Table 6.2 gives information about the number of particles and cluster copies per level, averaged over all time steps, and shows the amount of data required to store each level of detail and the entire hierarchical data representation. Compared to the original data, the data size increases by about 39% in average and the memory overhead of all low-resolution levels did not exceed 3 GB. Accordingly, without the finest level it is possible to load the entire hierarchy including all time steps into main memory. Thus, no expensive disk I/O is required when navigating through space and time, or between different levels of detail.

Figure 6.10 shows the trajectories of the WDMerger data set at three levels of detail. As can be seen, the motion within the data is well preserved while the number of trajectories is significantly reduced. Moreover, the original particle trajectories result in a high degree of occlusion that obscures the flow of mass further inside the data set. Our scale-space approach permits the user to analyze the dynamics within the data by

Table 6.2: *Number of clusters and copy clusters per level as averages over all time steps as well as the total amount of data for the distinct levels and for the entire space-time hierarchy.*

	WDMerger (8,7 GB)			SNIaEjecta (22.3 GB)		
Level	# Clusters	# Copy Cl.	Total size	# Clusters	# Copy Cl.	Total size
5	2,029	0	7 MB	1,495	0	4 MB
4	4,627	378	18 MB	11,443	782	28 MB
3	14,698	801	58 MB	64,606	3,532	155 MB
2	56,527	2,570	225 MB	282,963	24,529	685 MB
1	201,310	10,876	796 MB	844,435	195,037	2,185 MB
0	2,000,000	356,804	7,850 MB	8,745,571	1,375,428	19,782 MB

	Aquarius (24.1 GB)		
Level	# Clusters	# Copy Cl.	Total size
5	533	0	2 MB
4	2,549	95	3 MB
3	18,233	1087	20 MB
2	135,009	17,649	162 MB
1	846,592	215,892	1,120 MB
0	18,535,972	5,936,997	23,408 MB

selecting an appropriate abstraction levels. Thus, flows that are marginal within the data set are attenuated and the primary motion structure is revealed.

The movement of longer time intervals is usually visualized for smaller groups of clusters since the problem of occlusion and visual cluttering naturally increases with the length of the rendered trajectories (compare Figure 6.8(b)). Note, however, that this reduction is only motivated by a better visual impression. Regarding rendering performance and video memory requirements, the worst case of our experiments[1] was 46 fps and 9 MB (Figure 6.7(a)), respectively, which is far away from the theoretical throughput of modern GPUs. It is self-evident, that for large data sets occlusions and visual cluttering will impose more severe limitations than rendering throughput. Accordingly, our visualization approach should scale very well with the size of the data set given that a reasonable number and length of trajectories is chosen.

Combination with Volume Rendering

Especially when focusing on the major particle directions at coarse resolution levels of the proposed space-time hierarchy, detailed information about the underlying mass

[1]except the counter-example for occlusion and visual cluttering in Figure 6.10(a)

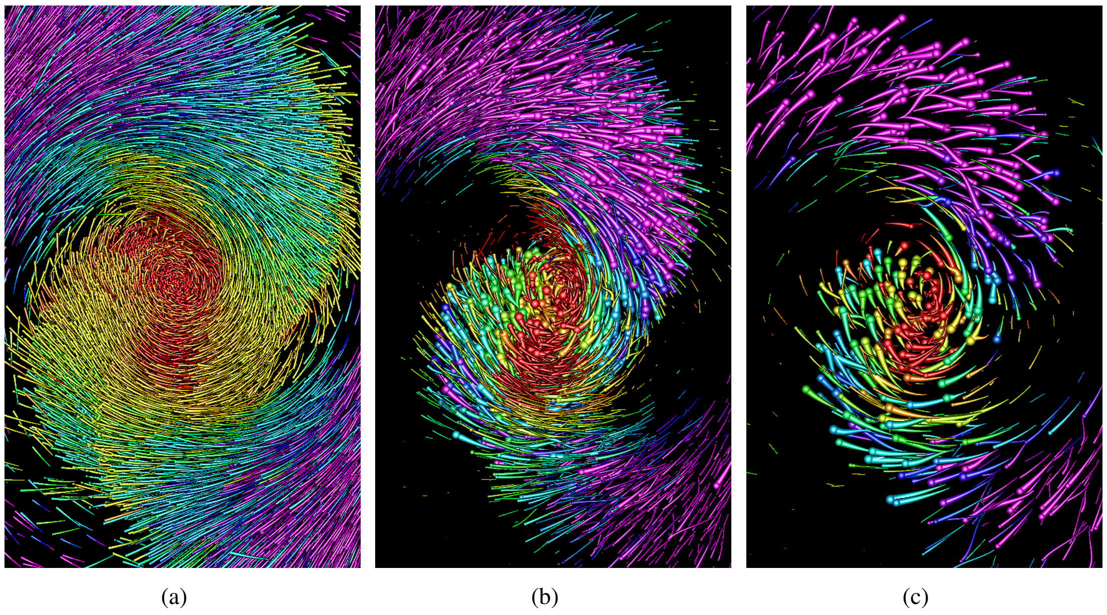


Figure 6.10: *Three levels of detail from our space-time hierarchy, color coded according to the cluster's temperature: (a) 2M particles rendered on a 1440×900 viewport at 3.1 fps, (b) 5.6K @ 563 fps, and (c) 1.1K @ 704 fps.*

distribution gets lost (see Figure 6.11(b)). Ray casting the density field, on the other hand, can effectively reveal this distribution in one single time step, but has problems in showing the dynamics with respect to the directional transport (see Figure 6.11(a)). By combining volume rendering of scalar particle quantities with the rendering of space-time trajectories, both drawbacks can be addressed (see Figure 6.11(c)).

6.6 Application to Non-Cosmological Data Sets

Even though the space-time hierarchy has been developed with the primary focus on astrophysical n-body simulations, it can be applied to any other kinds of SPH simulations or arbitrary time-dependent particle sets as well. Beyond that, such motion paths do not have to be part of the data to be analyzed as long as there is a possibility to determine them in an intermediate processing step. One well-known example is the field of steady and unsteady vector fields, where pathlines are created by advecting massless particles via time integration. To demonstrate that our approach can be used to effectively visualize the motion within such applications, we exemplary employ our space-time hierarchy on SPH data from the field of fluid dynamics and on unsteady vector fields.

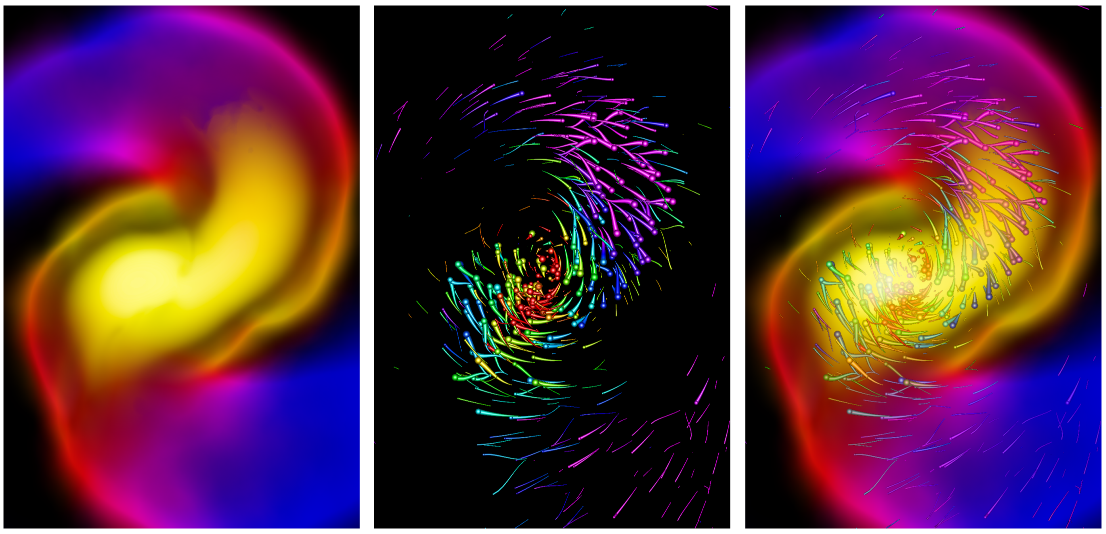

(a) (b) (c)

Figure 6.11: *Comparison of (a) volume rendering of the density field (1.5 fps), (b) motion visualization (900 fps), and (c) a hybrid rendering using both techniques (1.5 fps). All images were rendered on a 1440×900 viewport.*

6.6.1 SPH Fluid Dynamics

As already shown in Chapter 5, one application of SPH is the simulation of quasi-incompressible fluids. Accordingly, mergers and splits of particle clusters can not occur in contrast to the collisionless n-body simulations that have been used in the preceding experiments. Nevertheless, the space-time hierarchy allows to abstract the flow motions within the simulated fluid by joining neighboring particles with similar flow directions. By this means, different motions can be better distinguished and—as in the astrophysical data sets—the motions inside the fluid can be revealed by resolving the problems of occlusion and visual cluttering. Note that in contrast to simply rendering the trajectories of a random particle subset, our approach guarantees to arrange the trajectories at almost regular distances, which are automatically refined in turbulent areas. Furthermore, the space time-hierarchy ensures that all motions—even those of single particle sprinkles—are included in the motion visualization.

Figure 6.12 shows images of fluid dynamics data sets that were rendered with our approach. For the rendering of the fluid's surface, which serves as an important context information, we have implemented scale-invariant shading [Kra05]. To improve the depth perception of the underlying trajectories, the fluid can be additionally rendered as participating medium with a constant opacity and color (see Figure 6.12(a)).

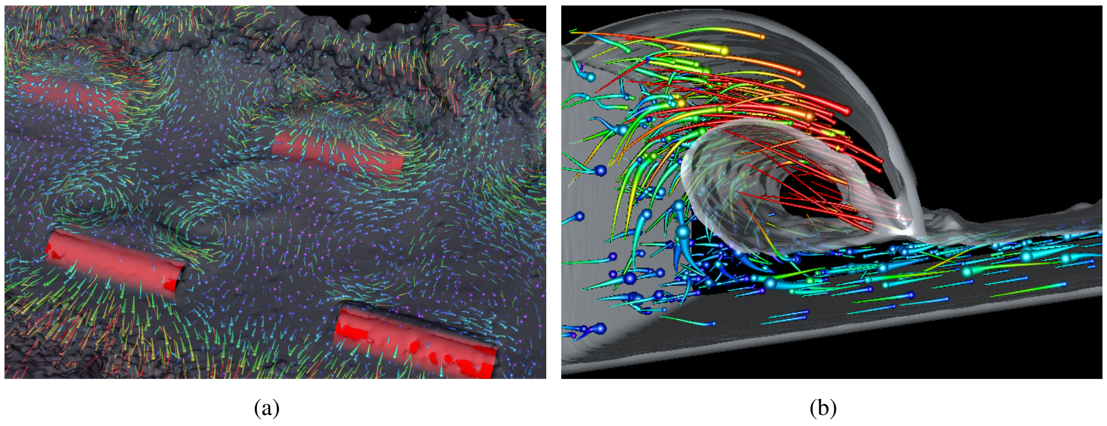


Figure 6.12: *(a) A top view on a fluid simulation showing the water surface in combination with the underlying flow directions and vortices. To improve the depth perception, the fluid is rendered with a constant opacity to attenuate deeper trajectories. (b) Cut through a fluid simulation revealing the flow within a breaking wave. The number of trajectories in both images is about 5% of the original number of particles.*

6.6.2 Unsteady Vector Fields

Particle tracing is a widely used technique to reveal flow characteristics within steady and unsteady vector fields. In analogy to large-scale particle simulations such as SPH, the differentiation of the distinct flow pattern does often require a large number of particles, which on the other hand limits the visual perception due to the problems of occlusion and visual cluttering. Accordingly, scientific effort has been made to minimize the number of particle trajectories while preserving distinct flow patterns within the data. In case of stationary vector fields, various techniques have been developed to solve this problem by applying cluster algorithms to discern distinct motion patterns, which can then be visualized by a reasonable number of characteristic lines [GPR$^+$04, TvW99, GPR$^+$00, HWHJ99, MLD05]. For unsteady flow fields, on the other hand, research has focused more on *focus-and-context* techniques, which reduce the density and appearance of particles or characteristic trajectories depending on a user-defined focus-region or specific flow properties [BKKW08, FG98, LG98, MTHG03]. Since our approach builds upon algorithms that are similar to the clustering techniques for stationary vector fields, our method can be seen as an extension of these techniques to time-dependent data sets.

The basic idea for applying the motion-hierarchy to vector fields is to seed and trace particles within a domain of interest and to store their positions and velocities that result from time integration. As for particle tracing in general, care must be taken that the number of seeded particles is large enough to capture all features of interest. Yet, in conventional particle tracing applications a trade-off must be found between accuracy and density, since a larger number of particles increases the problems of occlusion and visual cluttering. This problem becomes particular problematic for data that contains a high degree of divergence, leading to a non-uniform distribution of the seeded particles over time. Given that a sufficient number of particles has been traced, our space-time hierarchy solves this problem since converging particles are automatically thinned out and diverging clusters are split apart.

To illustrate the effectiveness of our technique, we traced one million particles in a sub-domain of the JHU turbulence data set [PBLM07, LPW^{+}08], which contains 1024 time steps for 1024^3 nodes, and created a space-time hierarchy for the resulting trajectories. Figure 6.13 presents the motion hierarchy at three levels of detail. As can be seen, our approach allows to abstract the distinct flow characteristics at different granularities by visualizing almost regularly spaced trajectories, even when particles move together or drift apart. However, these benefits of our space-time hierarchy comes at the costs of an additional preprocessing step. Accordingly, it is obvious that our approach is not competitive regarding the flexibility of interactive GPU-based particle tracing. Yet, our approach can be advantageous if the particle tracing itself has to be performed in an offline process, e.g. due to the size of the data set, or if an adaptive coarsening and refinement of trajectories is explicitly desired.

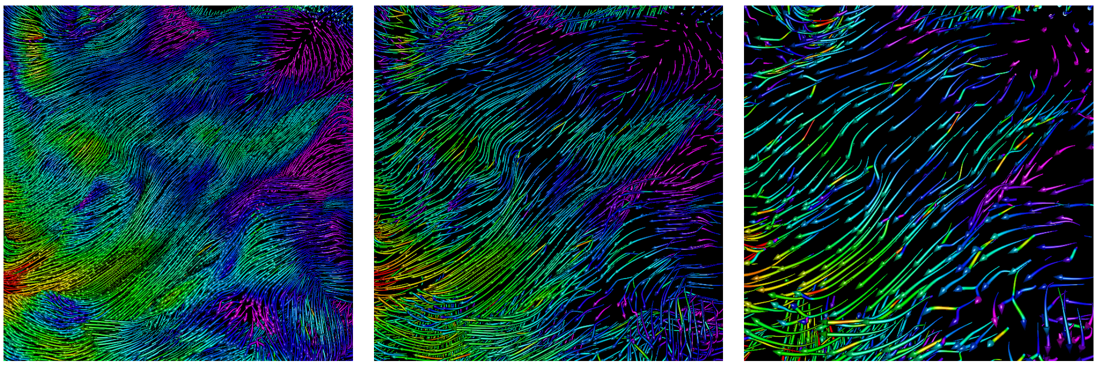

(a) (b) (c)

Figure 6.13: *Three different levels of detail of our space-time hierarchy that is based on the tracing of one million particles in the unsteady vector field of the JHU turbulence data set.*

6.7 Conclusion and Future Work

We have introduced a hierarchical space-time data structure for time-dependent particle sets that allows to analyze the flow of mass by rendering cluster trajectories at different scales. By choosing an appropriate level of detail, occlusion and visual clutter can be effectively avoided and the distinct motion patterns within the simulated data field can be revealed. To focus on regions of interest, arbitrary mapping function can be specified to modulate the appearance of the trajectories based on any particle quantity. Furthermore, sets of clusters can be interactively selected for exclusive rendering or visual highlighting. Combining the rendering of dynamic trajectories with other rendering techniques such as volume ray casting enables to analyze both, the shape of the density field and the underlying flow dynamics at a glance.

To the best of our knowledge, for the first time a scale-space analysis can be applied to the characteristic flows within arbitrary particle sets containing many millions of elements. We offer a new opportunity to visually explore the result of numerical particle simulations and to effectively visualize the dynamics of particle sets in static images. An interesting aspect for future work is to integrate focus-and-context strategies into our motion visualization. This would allow to locally adapt the level of detail according to regions or features of interest.

Chapter 7

Conclusion

In this thesis, we have presented scalable techniques to visualize large-scale particle simulations on commodity desktop PCs. The presented methods allow for interactive out-of-core rendering of SPH data sets whose size vastly exceeds the video memory capacities of modern graphics accelerators.

Our approach builds upon a novel particle representation that is based on a visually continuous multi-resolution hierarchy. The levels of detail of this data structure are constructed by applying merge and elimination operations to the particle set. This significantly reduces the number of particles to be rendered without impact on visual quality. Sophisticated data management and compression techniques allow for fast data access and transfer from secondary storage. We have introduced a resampling data structure, the perspective grid, that enables a high-quality, yet efficient volume ray casting of the addressed particle sets. In accordance with the divergence of viewing rays emanating from the camera, the perspective grid uses an adaptive sampling width that increases exponentially with distance to the view plane. Resampling is performed by using GPU-based voxelization. The samples are stored in a volumetric texture, which can be efficiently rendered by texture-based volume rendering techniques. We have furthermore presented a space-time hierarchy that addresses the need of scientists to analyze complex formation processes in cosmological simulations. The hierarchy enables the visualization of motion patterns at various scales by clustering adjacent particles of coherent motion. This provides the user with a means to select a level of detail appropriate for the size of a feature of interest.

Owing to adaptivity and the level-of-detail strategy, our methods scale well with increasing particle numbers. This significantly improves the exploration of large-scale particle sets. In contrast, existing visualization techniques are restricted either in interactivity (due to a high processing load) or rendering quality (due to required filtering of the data). With our approach particle sets containing millions of elements can be rendered in high quality at interactive frame rates. To the best of our knowledge, we are the first to show that this is feasible. To furthermore enable an effective visual data exploration for particle numbers on the order of billions of elements, we have presented alternative rendering techniques. By applying these methods to the Millennium Simulation, we have demonstrated that even the largest of today's data sets can be interactively explored on current PC architectures.

There are a number of topics for research building upon our work. Additional tools, for example probes to examine quantities in a region of interest, can be directly integrated into our visualization approach. Furthermore, our algorithms can be adapted to GPU clusters for the rendering on high resolution display walls or can be implemented on render clusters that provide remote visualization services. The latter issue is of particular interest since many scientists do not have local access to lage-scale data sets.

Even though the presented methods have been developed and evaluated with focus on SPH, many concepts and algorithms can be transferred to other particle-based simulations. The presented data compression and data management strategies are useful for a wide range of out-of-core applications. The multi-resolution particle hierarchy can be adapted to other particle data sets based on data-specific filtering and merging strategies. The proposed space-time hierarchy is a concept to analyze motion within any kind of particle sets. Finally, the perspective grid data structure can in general be used to effectively resample a continuous scalar field that is defined as the sum of per-particle functions, such as metaballs.

In conclusion, the work presented in this thesis has resulted in powerful instruments for scientists visually exploring and analyzing SPH data. By introducing interactivity to the visualization of large particle data sets, our techniques overcome severe limitations of available methods. Noting that, despite the focus on SPH, our methods can readily be applied to further types of particle data sets, we have thus provided a solid contribution to the further advancement of the visualization of particle-based simulations.

Bibliography

[ACP08] Gabriel Altay, Rupert A. C. Croft, and Inti Pelupessy, *SPHRAY: a smoothed particle hydrodynamics ray tracer for radiative transfer*, Monthly Notices of the Royal Astronomical Society **386** (2008), 1931–1946.

[AHH$^+$06] James Ahrens, Katrin Heitmann, Salman Habib, Lee Ankeny, Patrick McCormick, Jeff Inman, Ryan Armstrong, and Kwan liu Ma, *Quantitative and Comparative Visualization Applied to Cosmological Simulations*, Journal of Physics Conference Series **46** (2006), 526–534.

[ALD06] Bart Adams, Toon Lenaerts, and Philip Dutre, *Particle splatting: Interactive rendering of particle-based simulation data*, Technical report cw 453, Katholieke Universiteit Leuven, 2006.

[AMH02] Tomas Akenine-Möller and Eric Haines, *Real time rendering: Second edition*, A K Peters, Ltd., 2002.

[APB87] Bruno Arnaldi, Thierry Priol, and Kadi Bouatouch, *A new space subdivision method for ray tracing csg modelled scenes*, The Visual Computer **3** (1987), no. 2, 98–108.

[APKG07] Bart Adams, Mark Pauly, Richard Keiser, and Leonidas J. Guibas, *Adaptively sampled particle fluids*, ACM Transactions on Graphics **26** (2007), no. 3, 48–54.

[BDG$^+$04] Ken Brodlie, David Duce, Julian Gallop, Musbah Sagar, Jeremy Walton, and Jason Wood, *Visualization in grid computing environments*, Proceedings of the IEEE Visualization (Washington, DC, USA), IEEE Computer Society, 2004, pp. 155–162.

[BGM$^+$07] John Biddiscombe, Berk Geveci, Ken Martin, Kenneth Moreland, and David Thompson, *Time dependent processing in a parallel pipeline architecture*, IEEE Transactions on Visualization and Computer Graphics **13** (2007), no. 6, 1376–1383.

[BGM08] John Biddiscombe, David Graham, and Pierre Maruzewski, *Visualization and analysis of SPH data*, ERCOFTAC Bulletin **76** (2008), 9–12.

[BKKW08] Kai Bürger, Polina Kondratieva, Jens Krüger, and Rüdiger Westermann, *Importance-driven particle techniques for flow visualization*, Proceedings of the IEEE VGTC Pacific Visualization Symposium, 2008.

[Bly06] David Blythe, *The direct3d 10 system*, ACM Transactions on Graphics **25** (2006), 724–734.

[BTT09] Markus Becker, Hendrik Tessendorf, and Matthias Teschner, *Direct forcing for lagrangian rigid-fluid coupling*, IEEE Transactions on Visualization and Computer Graphics **15** (2009), no. 3, 493–503.

[CBB$^+$05] Henry R. Childs, Eric Brugger, Kathleen S. Bonnell, Jeremy S. Meredith, Mark Miller, Brad Whitlock, and Nelson Max, *A Contract Based System For Large Data Visualization.*, Proceedings of the IEEE Visualization, IEEE Computer Society, 2005, pp. 190–198.

[CE97] Michael Cox and David Ellsworth, *Application-controlled Demand Paging for Out-of-core Visualization*, Proceedings of the IEEE Visualization (Los Alamitos, CA, USA), IEEE Computer Society Press, 1997, pp. 235–244.

[CS09] Hilko Cords and Oliver G. Staadt, *Interactive screen-space surface rendering of dynamic particle clouds*, Journal of Graphics, GPU, and Game Tools **14** (2009), no. 3, 1–19.

[CSI09] Deukhyun Cha, Sungjin Son, and Insung Ihm, *Gpu-assisted high quality particle rendering*, Computer Graphics Forum **28** (2009), no. 4, 1247–1255.

[DC99] Mathieu Desbrun and Marie-Paule Cani, *Space-time adaptive simulation of highly deformable substances*, Tech. Report 3829, INRIA, BP 105 - 78153 Le Chesnay Cedex - France, December 1999.

[DCH88] Robert A. Drebin, Loren Carpenter, and Pat Hanrahan, *Volume rendering*, SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques, 1988, pp. 65–74.

[DeB04] Eric. P DeBenedictis, *Will moore's law be sufficient?*, Proceedings of the ACM/IEEE conference on supercomputing, 2004, pp. 45–56.

[DKW09] Christian Dick, Jens Krüger, and Rüdiger Westermann, *GPU ray-casting for scalable terrain rendering*, Proceedings of Eurographics 2009 - Areas Papers, Eurographics, 2009, pp. 43–50.

[DRGI08] Klaus Dolag, Martin Reinecke, Claudio Gheller, and Silvani Imboden, *Splotch: visualizing cosmological simulations*, New Journal of Physics **10** (2008), no. 12, 125006.

[DZTS08] Christopher Dyken, Gernot Ziegler, Christian Theobalt, and Hans-Peter Seidel, *High-speed marching cubes using histopyramids*, Computer Graphics Forum **27** (2008), no. 8, 2028–2039.

[EGM04] David Ellsworth, Bryan Green, and Patrick Moran, *Interactive terascale particle visualization*, Proceedings of the IEEE Visualization (Washington, DC, USA), IEEE Computer Society, 2004, pp. 353–360.

[EHK$^+$06] Klaus Engel, Markus Hadwiger, Joe M. Kniss, Christof Rezk-Salama, and Daniel Weiskopf, *Real-Time Volume Graphics*, A K Peters, Ltd., 2006.

[FAW10] Roland Fraedrich, Stefan Auer, and Rudiger Westermann, *Efficient high-quality volume rendering of sph data*, IEEE Transactions on Visualization and Computer Graphics **16** (2010), 1533–1540.

[FCDM+11] Yu Feng, Rupert A. C. Croft, Tiziana Di Matteo, Nishikanta Khandai, Randy Sargent, Illah Nourbakhsh, Paul Dille, Chris Bartley, Volker Springel, Anirban Jana, and Jeffrey Gardner, *Terapixel imaging of cosmological simulations*, The Astrophysical Journal Supplement Series **197** (2011), no. 2, 18.

[FG98] Anton Fuhrmann and Eduard Gröller, *Real-time techniques for 3d flow visualization*, Proceedings of the IEEE Visualization '98, 1998, pp. 305–312.

[FGE10] Martin Falk, Sebastian Grottel, and Thomas Ertl, *Interactive image-space volume visualization for dynamic particle simulations*, Proceedings of The Annual SIGRAD Conference, 2010, pp. 35–43.

[FSW09] Roland Fraedrich, Jens Schneider, and Rüdiger Westermann, *Exploring the Millenium Run - scalable rendering of large-scale cosmological datasets*, IEEE Transactions on Visualization and Computer Graphics **15** (2009), no. 6, 1251–1258.

[FW12] Roland Fraedrich and Rüdiger Westermann, *Motion visualization of particle simulations*, IS&T/SPIE Conference on Visualization and Data Analysis (VDA 2012), 2012.

[GG91] Allen Gersho and Robert M. Gray, *Vector quantization and signal compression*, Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[GGS02] Stefan Guthe, Stefan Gumhold, and Wolfgang Straßer, *Interactive visualization of volumetric vector fields using texture based particles*, Proceedings of WSCG, 2002.

[GH03] Stefan Guthe and Paul Heckbert, *Non-power-of-two mipmap creation*, Tech. Report TR-01838-001, NVIDIA Corporation, 2003.

[GIK+07] Christiaan P. Gribble, Thiago Ize, Andrew Kensler, Ingo Wald, and Steven G. Parker, *A coherent grid traversal approach to visualizing particle-based simulation data*, IEEE Transactions on Visualization and Computer Graphics **13** (2007), no. 4, 758–768.

[GKM93] Ned Greene, Michael Kass, and Gavin Miller, *Hierarchical z-buffer visibility*, Proceedings of the 20th annual Conference on Computer Graphics and Interactive Techniques, 1993, pp. 231–238.

[GM77] Robert A. Gingold and Joe J. Monaghan, *Smoothed particle hydrodynamics - theory and application to non-spherical stars*, Monthly Notices of the Royal Astronomical Society **181** (1977), 375–389.

[GMI08] Enrico Gobbetti, Fabio Marton, and José Antonio Iglesias Guitián, *A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets*, The Visual Computer **24** (2008), no. 7-9, 797–806.

[GPR+00] Harald Garcke, Tobias Preusser, Martin Rumpf, Alexandru Telea, Ulrich Weikard, and Jarke van Wijk, *A continuous clustering method for vector fields*, Proceedings of the IEEE Visualization 2000, 2000.

[GPR+04] Michael Griebel, Tobias Preusser, Martin Rumpf, Marc Alexander Schweitzer, and Alexandru. Telea, *Flow field clustering via algebraic multigrid*, Proceedings of the IEEE Visualization, 2004, pp. 35–42.

[GSSP10] Prashant Goswami, Philipp Schlegel, Barbara Solenthaler, and Renato Pajarola, *Interactive SPH simulation and rendering on the GPU*, Proceedings ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2010, pp. 55–64.

[HE03] Matthias Hopf and Thomas Ertl, *Hierarchical splatting of scattered data*, Proceedings of the IEEE Visualization, 2003, pp. 433–440.

[Hen08] Amy Henderson, *ParaView Guide, A Parallel Visualization Application. Kitware Inc.*, 2008.

[HF11] Amr Hassan and Christopher J. Fluke, *Scientific visualization in astronomy: Towards the petascale astronomy era*, Computing Research Repository **arXiv:1102.5123** (2011).

[HHK08] Woosuck Hong, Donald H. House, and John Keyser, *Adaptive particles for incompressible fluid simulation*, The Visual Computer **24** (2008), no. 7, 535–543.

[HK89] Lars Hernquist and Neal Katz, *TREESPH-A unification of SPH with the hierarchical tree method*, The Astrophysical Journal Supplement Series **70** (1989), 419–446.

[HKK07a] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi, *Sliced data structure for particle-based simulations on gpus*, Proc. of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia (New York, NY, USA), GRAPHITE '07, ACM, 2007, pp. 55–62.

[HKK07b] ______, *Smoothed particle hydrodynamics on GPUs*, Computer Graphics International, 2007, pp. 63–70.

[HLE04] Matthias Hopf, Michael Luttenberger, and Thomas Ertl, *Hierarchical Splatting of Scattered 4D Data*, IEEE Computer Graphics and Applications **24** (2004), no. 4, 64–72.

[Hot33] Harold Hotelling, *Analysis of a complex of statistical variables into principal components*, Journal of Educational Psychology **24** (1933), no. 6, 417–441.

[HWHJ99] Bjoern Heckel, Gunther Weber, Bernd Hamann, and Kenneth I. Joy, *Construction of vector field hierarchies*, Proceedings of the IEEE Visualization, 1999, pp. 19–25.

[JFSP10] Yun Jang, Raphael Fuchs, Benjamin Schindler, and Ronny Peikert, *Volumetric evaluation of meshless data from smoothed particle hydrodynamics simulations*, Proceedings of the Volume Graphics 2010, 2010, pp. 45–52.

[JH04] Christopher Johnson and Charles Hansen, *Visualization Handbook*, Academic Press, Inc., Orlando, FL, USA, 2004.

[Jol02] Ian T. Jolliffe, *Principal component analysis*, 2 ed., Springer, 2002.

[JST$^+$10] Won-Ki Jeong, Jens Schneider, Stephen G. Turney, Beverly E. Faulkner-Jones, Dominik Meyer, Ruediger Westermann, R. Clay Reid, Jeff Lichtman, and Hanspeter Pfister, *Interactive histology of large-scale biomedical image stacks*, IEEE Transactions on Visualization and Computer Graphics (2010).

[KAH07] Ralf Kähler, Tom Abel, and Hans-Christian Hege, *Simultaneous gpu-assisted raycasting of unstructured point sets and volumetric grid data*, Proceedings of the IEEE/EG International Symposium on Volume Graphics, 2007, pp. 49 – 56.

[Kar47] Kari Karhunen, *Über lineare methoden in der wahrscheinlickheitsrechnung.*, Annales Academiae Scientiarum Fennicae, Series A, **37** (1947), 1–79.

[KC05] Andreas Kolb and Nicolas Cuntz, *Dynamic particle coupling for GPU-based fluid simulation*, Proceedings of the Symposium on Simulation Technique, 2005, pp. 722–727.

[KCP$^+$02] Ralf Kähler, Donna Cox, Robert Patterson, Stuart Levy, Hans-Christian Hege, and Tom Abel, *Rendering The First Star in The Universe - A Case Study*, Proceedings of the IEEE Visualization (Robert J. Moorhead, Markus Gross, and Kenneth I. Joy, eds.), IEEE Computer Society, IEEE Computer Society Press, October/November 2002, pp. 537–540.

[KKKW05] Jens Krüger, Peter Kipfer, Polina Kondratieva, and Rüdiger Westermann, *A particle system for interactive visualization of 3D flows*, IEEE Transactions on Visualization and Computer Graphics **11** (2005), no. 6, 744–756.

[Knu97] Donald E. Knuth, *The art of computer programming. volume 1 (second ed.)*, ch. Fundamental Algorithms, pp. 435–455, Addison-Wesley, 1997.

[Kra05] Martin Kraus, *Scale-invariant volume rendering*, Proceedings of the IEEE Visualization, 2005, pp. 295–302.

[Kra08] ———, *Pre-integrated volume rendering for multi-dimensional transfer functions*, IEEE/EG Symposium on Volume and Point-based Graphics (2008), 2008, pp. 97–104.

[KSN08] Yoshihiro Kanamori, Zoltan Szego, and Tomoyuki Nishita, *GPU-based fast ray casting for a large number of metaballs*, Computer Graphics Forum **27** (2008), no. 2, 351–360.

[KSW05] Jens Krüger, Jens Schneider, and Rüdiger Westermann, *Duodecim - a structure for point scan compression and rendering*, Proceedings of the Symposium on Point-Based Graphics 2005, 2005.

[KSW06] ———, *Compression and rendering of iso-surfaces and point sampled geometry*, The Visual Computer **22** (2006), no. 8, 517–530.

[KvH84] James T. Kajiya and Brian P. von Herzen, *Ray Tracing Volume Densities*, SIGGRAPH Comput. Graph. **18** (1984), no. 3, 165–174.

[KW03] Jens Krüger and Rüdiger Westermann, *Acceleration techniques for GPU-based volume rendering*, Proceedings of the IEEE Visualization, 2003, pp. 38–43.

[LBG80] Yoseph Linde, Andres Buzo, and Robert M. Gray, *An algorithm for vector quantizer design*, IEEE Transactions on Communications **28** (1980), no. 1, 84–95.

[LC87] William E. Lorensen and Harvey E. Cline, *Marching cubes: A high resolution 3D surface construction algorithm*, SIGGRAPH Comput. Graph. **21** (1987), no. 4, 163–169.

[Lev88] Marc Levoy, *Display of surfaces from volume data*, IEEE Computer Graphics and Applications **8** (1988), 29–37.

[LFH06] Yinggang Li, Chi-Wing Fu, and Andrew Hanson, *Scalable WIM: Effective Exploration in Large-scale Astrophysical Environments*, IEEE Transactions on Visualization and Computer Graphics **12** (2006), no. 5, 1005–1012.

[LFLH07] Hongwei Li, Chi-Wing Fu, Yinggang Li, and Andrew Hanson, *Visualizing Large-Scale Uncertainty in Astrophysical Data*, IEEE Transactions on Visualization and Computer Graphics **13** (2007), no. 6, 1640–1647.

[LG98] Helwig Löffelmann and Eduard Gröller, *Enhancing the visualization of characteristic structures in dynamical systems*, Proceedings of Eurographics Workshop on Visualization in Scientific Computing, Springer-Verlag, Eurographics, 1998, pp. 59–68.

[LGK$^+$01] Hendrik P. A. Lensch, Michael Goesele, Jan Kautz, Wolfgang Heidrich, and Hans-Peter Seidel, *Image-based reconstruction of spatially varying materials*, Proceedings of the 12th EG Workshop on Rendering Techniques, 2001, pp. 103–114.

[LH05] Robert S. Laramee and Helwig Hauser, *Geometric flow visualization techniques for cfd simulation data*, Proceedings of the 21st spring conference on Computer graphics (New York, NY, USA), SCCG '05, ACM, 2005, pp. 221–224.

[LHJ99] Eric LaMar, Bernd Hamann, and Kenneth I. Joy, *Multiresolution techniques for interactive texture-based volume visualization*, Proceedings of the IEEE Visualization, 1999, pp. 355–361.

[LHM$^+$07] Andrei Lintu, Lars Hoffman, Marcus A. Magnor, Hendrik P. A. Lensch, and Hans-Peter Seidel, *3D Reconstruction of Reflection Nebulae from a Single Image*, Proceedings of the Vision, Modeling, and Visualization Conference, 2007, pp. 109–116.

[LL03] Gui-Rong Liu and Mou-Bing Liu, *Smoothed particle hydrodynamics: a meshfree particle method*, World Scientific, 2003.

[Llo82] Stuart P. Lloyd, *Least squares quantization in pcm.*, IEEE Transactions on Information Theory **28** (1982), no. 2, 129–136.

[Loè46] Michel Loève, *Fonctions aléatoires du second ordre*, Revue Scientifique **84** (1946), 195–206.

[LPD$^+$02] Lars Linsen, Valerio Pascucci, Mark A. Duchaineau, Bernd Hamann, and Kenneth I. Joy, *Hierarchical representation of time-varying volume data with "4th-root-of-2" subdivision and quadrilinear b-spline wavelets*, Proceedings of the Pacific Conference on Computer Graphics and Applications, 2002, pp. 346–355.

[LPW$^+$08] Yi Li, Eric Perlman, Minping Wan, Yunke Yang, Charles Meneveau, Randal Burns, Shiyi Chen, Alexander Szalay, and Gregory Eyink, *A public turbulence database cluster and applications to study lagrangian evolution of velocity increments in turbulence*, Journal of Turbulence **9** (2008), no. 31, 1–29.

[Luc77] Leon B. Lucy, *A numerical approach to the testing of the fission hypothesis*, The Astronomical Journal **82** (1977), 1013–1024.

[MCG03] Matthias Müller, David Charypar, and Markus Gross, *Particle-based fluid simulation for interactive applications*, Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2003, pp. 154–159.

[Mic10] Microsoft, *Windows directx graphics documentation (part of the directx software development kit)*, June 2010.

[MJM$^+$06] Tamara Munzner, Chris Johnson, Robert Moorhead, Hanspeter Pfister, Penny Rheingans, and Terry S. Yoo, *NIH-NSF Visualization Research Challenges Report Summary*, IEEE Computer Graphics and Applications **26** (2006), no. 2, 20–24.

[MLD05] Alexander McKenzie, Santiago V. Lombeyda, and Mathieu Desbrun, *Vector field analysis and visualization through variational clustering*, Proceedings of the EuroVis (Ken Brodlie, David J. Duke, and Kenneth I. Joy, eds.), Eurographics Association, 2005, pp. 29–35.

[MLP$^+$10] Tony McLoughlin, Robert S. Laramee, Ronald Peikert, Frits H. Post, and Min Chen, *Over Two Decades of Integration-Based, Geometric Flow Visualization*, Computer Graphics Forum **29** (2010), no. 6, 1807–1829.

[Mon92] Joe J. Monaghan, *Smoothed particle hydrodynamics*, Annual review of astronomy and astrophysics **30** (1992), 543–574.

[Mon05] ———, *Smoothed particle hydrodynamics*, Rep. Prog. Phys. **68** (2005), 1703–1758.

[Moo65] Gordon E. Moore, *Cramming more components onto integrated circuits*, Electronics **38** (1965), no. 8, 114–117.

[MQF06] Jameson Miller, Cory Quammen, and Matthew Fleenor, *Interactive Visualization of Intercluster Galaxy Structures in the Horologium-Reticulum Supercluster*, IEEE Transactions on Visualization and Computer Graphics **12** (2006), no. 5, 1149–1156.

[MSD07] Matthias Müller, Simon Schirm, and Stephan Duthaler, *Screen space meshes*, Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2007, pp. 9–15.

[MSE$^+$06] Dorit Merhof, Markus Sonntag, Frank Enders, Christopher Nimsky, Peter Hastreiter, and Guenther Greiner, *Hybrid visualization for white matter tracts using triangle strips and point sprites*, IEEE Transactions on Visualization and Computer Graphics **12** (2006), 1181–1188.

[MTHG03] Oliver Mattausch, Thomas Theußl, Helwig Hauser, and Eduard Gröller, *Strategies for interactive exploration of 3d flow using evenly-spaced illuminated streamlines*, Proceedings of the spring conference on Computer graphics, SCCG '03, 2003, pp. 213–222.

[NJB07] Paul Arthur Navrátil, Jarrett L. Johnson, and Volker Bromm, *Visualization of Cosmological Particle-Based Datasets*, IEEE Transactions on Visualization and Computer Graphics, Nov/Dec 2007.

[NNT$^+$05] Chu-Ming Ng, Cam-Thach Nguyen, Dinh-Nguyen Tran, Tiow-Seng Tan, and Shin-We Yeow, *Analyzing Pre-fetching in Large-scale Visual Simulation*, Computer Graphics International Conference **0** (2005), 100–107.

[OKK10] Jens Orthmann, Maik Keller, and Andreas Kolb, *Topology-caching for dynamic particle volume raycasting*, Proceedings of the Vision, Modeling and Visualization Conference, Eurographics Ass., 2010, pp. 147–154.

[ON97] Jeremiah P. Ostriker and Michael L. Norman, *Cosmology of the Early Universe Viewed Through the New Infrastructure*, Commun. ACM **40** (1997), no. 11, 84–94.

[PBLM07] Eric Perlman, Randal Burns, Yi Li, and Charles Meneveau, *Data exploration of turbulence simulations using a database cluster*, Proceedings of the ACM/IEEE conference on Supercomputing, 2007, pp. 23:1–23:11.

[PD84] Thomas Porter and Tom Duff, *Compositing digital images*, SIGGRAPH Comput. Graph. **18** (1984), 253–259.

[Pea01] Karl Pearson, *On lines and planes of closest fit to systems of points in space.*, Philosophical Magazine **2** (1901), no. 11, 559–572.

[PGK02] Mark Pauly, Markus Gross, and Leif P. Kobbelt, *Efficient simplification of point-sampled surfaces*, Proceedings of the IEEE Visualization (Washington, DC, USA), VIS '02, IEEE Computer Society, 2002, pp. 163–170.

[PLB$^+$01] Hanspeter Pfister, Bill Lorensen, Chandrajit Bajaj, Gordon Kindlmann, Will Schroeder, Lisa Sobierajski Avila, Ken Martin, Raghu Machiraju, and Jinho Lee, *The transfer function bake-off*, IEEE Computer Graphics and Applications **21** (2001), 16–22.

[PMS$^+$99] Steven Parker, Martin Martin, Peter-Pike Sloan, Peter Shirley, Brian Smits, and Charles Hansen, *Interactive ray tracing*, Symposium on Interactive 3D Computer Graphics, 1999, pp. 119–126.

[Pri07] Daniel J. Price, *Splash: An interactive visualisation tool for smoothed particle hydrodynamics simulations*, Publications of the Astronomical Society of Australia **24** (2007), 159–173.

[PVH$^+$02] Frits H. Post, Benjamin Vrolijk, Helwig Hauser, Robert S. Laramee, and Helmut Doleisch, *Feature extraction and visualisation of flow fields*, Eurographics 2002 STAR (Dieter Fellner and Roberto Scopigno, eds.), Eurographics Association, Saarbrücken Germany, 2002, pp. 69–100.

[PVTF02] William H. Press, William T. Vetterling, Saul A. Teukolsky, and Brian P. Flannery, *Numerical recipes in c++: The art of scientific computing*, ch. Eigensystems, pp. 474–486, Cambridge University Press, New York, USA, 2002.

[RB08] Ilya D. Rosenberg and Ken Birdwell, *Real-time particle isosurface extraction*, Proceedings of the Symposium on Interactive 3D Graphics and Games, 2008, pp. 35–43.

[RL08] Paul Rosenthal and Lars Linsen, *Smooth surface extraction from unstructured point-based volume data using PDEs*, IEEE Transactions on Visualization and Computer Graphics **14** (2008), no. 6, 1531–1546.

[RRE06] Alexander Rosiuta, Guido Reina, and Thomas Ertl, *Flexible Interaction with Large Point-Based Datasets*, Theory and Practice of Computer Graphics '06, 2006.

[RRL07] Paul Rosenthal, Stephan Rosswog, and Lars Linsen, *Direct surface extraction from smoothed particle hydrodynamics simulation data*, Proceedings of the High-End Visualization Workshop, 2007.

[Say00] Khalid Sayood, *Introduction to data compression*, 2 ed., Morgan Kaufmann Publishers, San Francisco, CA, USA, 2000.

[SD02] Marc Stamminger and George Drettakis, *Perspective shadow maps*, SIGGRAPH '02: Proceedings of the 29th annual Conference on Computer Graphics and Interactive Techniques, 2002, pp. 557–562.

[SFBP09] Benjamin Schindler, Raphael Fuchs, John Biddiscombe, and Ronald Peikert, *Predictor-corrector schemes for visualization ofsmoothed particle hydrodynamics data*, IEEE Transactions on Visualization and Computer Graphics **15** (2009), 1243–1250.

[SFWP11] Benjamin Schindler, Raphael Fuchs, Jürgen Waser, and Ronald Peikert, *Marching Correctors – Fast and Precise Polygonal Isosurfaces of SPH Data*, Proc. 6th SPHERIC workshop (T. Rung and C. Ulrich, eds.), 2011, pp. 125–132.

[SHG09] Nadathur Satish, Mark Harris, and Michael Garland, *Designing efficient sorting algorithms for manycore gpus*, Proceedings of the IEEE International Symposium on Parallel&Distributed Processing, 2009, pp. 1–10.

[Spr05] Volker Springel, *The Cosmological Simulation Code GADGET-2*, Monthly Notices of the Royal Astronomical Society **364** (2005), no. 4, 1105–1134.

[SSL08] Tamas Szalay, Volker Springel, and Gerard Lemson, *GPU-Based Interactive Visualization of Billion Point Cosmological Simulations*, Computing Research Repository **arXiv:0811.2055** (2008).

[SW03] Jens Schneider and Rüdiger Westermann, *Compression domain volume rendering*, Proceedings of the IEEE Visualization, 2003, pp. 293–300.

[SWJ$^+$05] Volker Springel, Simon D. M. White, Adrian Jenkins, Carlos S. Frenk, Naoki Yoshida, Liang Gao, Julio Navarro, Robert Thacker, Darren Croton, John Helly, John A. Peacock, Shaun Cole, Peter Thomas, Hugh Couchman, August Evrard, Joerg Colberg, and Frazer Pearce, *Simulating the Joint Evolution of Quasars, Galaxies and their Large-scale Distribution*, Nature **435** (2005), 629–636.

[SWV$^+$08] Volker Springel, Jie Wang, Mark Vogelsberger, Aaron Ludlow, Adrian Jenkins, Amina Helmi, Julio F. Navarro, Carlos S. Frenk, and Simon D. M. White, *The Aquarius Project: The Subhalos of Galactic Halos*, Monthly Notices of the Royal Astronomical Society **391** (2008), 1685–1711.

[TvW99] Alexandru Telea and Jarke J. van Wijk, *Simplified representation of vector fields*, Proceedings of the IEEE Visualization, 1999.

[vdLGS09] Wladimir J. van der Laan, Simon Green, and Miguel Sainz, *Screen space fluid rendering with curvature flow*, Proceedings of the Symposium on Interactive 3D Graphics and Games, 2009, pp. 91–98.

[WAF$^+$11] Jonathan Woodring, James P. Ahrens, J. Figg, Joanne Wendelberger, Salman Habib, and Katrin Heitmann, *In-situ sampling of a large-scale particle simulation for interactive visualization and analysis*, Computer Graphics Forum **30** (2011), no. 3, 1151–1160.

[Wes95] Ruediger Westermann, *Compression domain rendering of time-resolved volume data*, Proceedings of the IEEE Visualization Conference, 1995, pp. 168–175.

[WKM05] Rick Walker, Peter Kenny, and Jingqi Miao, *Visualization of smoothed particle hydrodynamics for astrophysics*, Theory and Practice of Computer Graphics 2005 (University of Kent, UK) (Louise Lever and Mary McDerby, eds.), Eurographics Association, June 2005, pp. 133–138.

[WS09a] Jonathan Woodring and Han-Wei Shen, *Multiscale time activity data exploration via temporal clustering visualization spreadsheet*, IEEE Transactions on Visualization and Computer Graphics **15** (2009), no. 1, 123–137.

[WS09b] Jonathan Woodring and Han-Wei Shen, *Semi-automatic time-series transfer functions via temporal clustering and sequencing*, Computer Graphics Forum **28** (2009), no. 3, 791–798.

[WSP04] Michael Wimmer, Daniel Scherzer, and Werner Purgathofer, *Light space perspective shadow maps*, Rendering Techniques 2004 (Proceedings Eurographics Symposium on Rendering), 2004, pp. 143–151.

[YHK09] Ren Yasuda, Takahiro Harada, and Yoichiro Kawaguchi, *Fast rendering of particle-based fluid by utilizing simulation data*, Proceedings of Eurographics 2009 - Short Papers (Munich, Germany) (Pierre Alliez and Marcus Magnor, eds.), Eurographics Association, 2009, pp. 61–64.

[ZB05] Yongning Zhu and Robert Bridson, *Animating sand as a fluid*, SIGGRAPH '05: ACM SIGGRAPH 2005 Papers, 2005, pp. 965–972.

[ZSP08] Yanci Zhang, Barbara Solenthaler, and Renato Pajarola, *Adaptive sampling and rendering of fluids on the GPU*, Symposium on Point-Based Graphics, 2008, pp. 137–146.