Real-Time Rendering of Large AMR Data in Virtual Reality Environments

Department Informatik

Mathematisch-Naturwissenschaftliche Fakultät

Universität zu Köln

Masterarbeit

vorgelegt von Zhaoyang Wang

Matrikelnummer: 7307787

am 21. August 2023

Erstgutachter: PD Dr. rer. nat. Stefan Zellmann

Zweitgutachter: Prof. Dr.-Ing. Stefan Wesner

Abstract

Rendering large adaptive mesh refinement (AMR) data in real-time within virtual reality (VR) environments is a complex challenge that demands sophisticated techniques and tools. This master's thesis introduces a real-time rendering solution for sizable AMR datasets in both desktop and projection-based VR setups. The approach harnesses the ExaBrick framework, integrating it as a plugin in COVISE—a robust visualization system equipped with the VR-centric OpenCOVER render module. This setup enables direct navigation and interaction within the rendered volume through a head tracker or controller. The user interface incorporates rendering options and functions, ensuring a smooth and interactive experience. The proposed solution not only offers real-time rendering performance but also provides researchers with an immersive visualization tool for comprehensive AMR data analysis. The achieved results demonstrate the approach's viability and its potential applications in scientific visualization and data analysis.

Kurzfassung

Das Rendering großer adaptive mesh refinement (AMR) Daten in Echtzeit in Virtual-Reality-Umgebungen (VR) ist eine komplexe Herausforderung, die hochentwickelte Techniken und Tools erfordert. In dieser Masterarbeit wird eine Echtzeit-Rendering-Methode für große AMR-Datensätze sowohl in Desktop- als auch in projektionsbasierten VR-Konfigurationen vorgestellt. Der Ansatz nutzt das ExaBrick-Framework und integriert es als Plugin in COVISE - ein robustes Visualisierungssystem, das mit dem VR-zentrierten Rendermodul OpenCOVER ausgestattet ist. Dieses Setup ermöglicht die direkte Navigation und Interaktion innerhalb des gerenderten Volumens durch einen Headtracker oder Controller. Die Benutzeroberfläche enthält Renderoptionen und -funktionen, die ein reibungsloses und interaktives Erlebnis gewährleisten. Die vorgestellte Lösung bietet nicht nur Echtzeit-Rendering-Leistung, sondern stellt Forschern auch ein immersives Visualisierungstool für eine umfassende AMR-Datenanalyse zur Verfügung. Die erzielten Ergebnisse zeigen die Durchführbarkeit des Ansatzes und seine potenziellen Anwendungen in der wissenschaftlichen Visualisierung und Datenanalyse.

Acknowledgments

I would like to express my deepest gratitude to my supervisor PD Dr. rer. nat. Stefan Zellmann, for his invaluable guidance, insights, and support throughout the development of this master's thesis. His expertise and encouragement have been instrumental in shaping the course of this research.

I extend my sincere appreciation to the University of Cologne for providing access to the necessary resources and facilities for conducting this research.

My sincere thanks go to my family and friends for their constant encouragement and understanding. Your belief in me has been a driving force behind this work.

I am grateful to the developers of the ExaBrick framework [WZU*20] and COVISE [Ran95] for providing solid foundations for this work.

The completion of this thesis would not have been possible without the collective support of all these individuals and entities.

Overview

1. Introduction

1.1 Motivation

In recent years, scientific simulations have generated voluminous data, particularly in the domain of adaptive mesh refinement (AMR). AMR data [BO84, BC89] is characterized by its hierarchical structure, where the computational domain is divided into a series of nested grids with varying levels of refinement. It offers the advantage of enhanced simulations by enabling higher resolution in critical regions while maintaining a coarser representation in less significant areas. With the help of ExaBrick framework [WZU*20], large AMR data can be re-arranged into ExaBricks data structure and efficiently rendered into numerous frames within one second. However, real-time visualization of large AMR datasets presents formidable challenges due to their intricate nature and the imperative for interactive exploration and analysis.

Real-time rendering technology assumes a pivotal role in enabling interactive exploration and analysis, allowing users to seamlessly navigate and manipulate data in a responsive manner. Virtual reality (VR) has become a powerful technology that provides an immersive and interactive experience, making it an ideal platform for visualizing complex scientific data. By harnessing the potential of VR, researchers and scientists can attain a profound comprehension of the complex structures and phenomena represented by AMR data.

1.2 Research Objective

The main objective of this master thesis is to develop a real-time rendering solution for large AMR datasets in a VR environment. The proposed solution leverages the ExaBrick framework [WZU*20] and integrates it as a plugin into COVISE [Ran95], a powerful software framework that combines visualization and simulation capabilities. By enabling real-time rendering performance, we aim to provide researchers and scientists with an immersive and interactive visualization tool ExaBrick Plugin of COVISE to facilitate in-depth analysis and exploration of AMR data.

1.3 Significance

The research of this master thesis contributes to the field of scientific visualization by addressing the challenges associated with rendering large AMR datasets in both desktop and VR environment in real time. The developed solution provides a valuable tool, the ExaBrick Plugin of COVISE, for researchers and scientists working in different fields, such as astrophysics, computational fluid dynamics and medicine, enabling them to gain new insights and make more informed decisions based on simulations.

2 Related Work

In this section, we delve into an in-depth analysis of the current body of knowledge and cutting-edge research pertaining to the real-time rendering of volumetric 3D AMR data within VR environments. Developing a firm grasp of the historical context and the strides made in this domain is pivotal for contextualizing our proposed work and accentuating its distinctive contributions within the existing research landscape.

2.1 Scientific Visualization

As highlighted by Friendly [Fri08], scientific visualization "[...] is primarily concerned with the visualization of 3-D+ phenomena (architectural, meterological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time) component." Scientific visualization plays a pivotal role in enabling scientists to

comprehend and derive insights from diverse datassets across various scientific domains [Mas19]. The domain of scientific visualization offers a spectrum of software solutions, ranging from commercial tools to open-source frameworks. Prominent examples include Amira [Sta05], Avizo [Avi23], ParaView [Par23], and VisIt [Vis23].

The data maybe numerically simulated or captured using scientific or medical equipments, and the size of the data is important. Large-scale data could be a big challenge to visualize because it takes easily a large amount of computational sources. The visualization in this case becomes very expensive and slow.

2.2 Large-Scale Data Visualization

Techniques visualizing large-scale data is more and more needed, and there has been a lot of development in this area. J. Sarton et al. [SZD*23] reviewed the state-of-the-art in large-scale volume visualization, scrutinizing diverse data representations as shown in Image 2.1. These advancements contribute to the field's ability to handle complex volumetric data and improve the quality and accuracy of visualizations. For example, Sarton et al. [SCRL20] introduced an efficient solution for managing large 3D grid data on a GPU. This approach minimizes memory transfers between GPU and host and is designed to maintain

GPU performance while reducing CPU-GPU communication. Morrical et al. [MWUP22] harnessed ray tracing hardware in NVIDIA's RTX GPUs, achieving substantial performance enhancements in unstructured mesh point location.

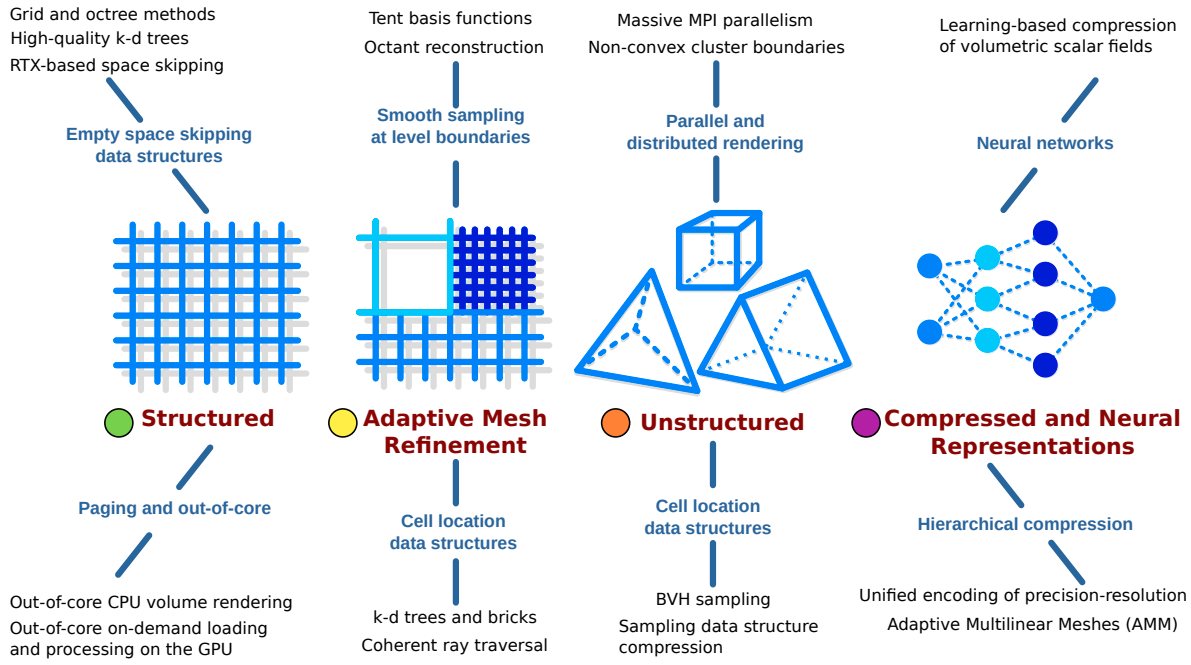


*Image 2.1: Different data representations in paper "State-of-the-art in Large-Scale Volume Visualization Beyond Structured Data" [SZD*23]*

2.2.1 AMR Visualization

Rendering AMR data has been a subject of significant interest and importance in the scientific visualization community. Berger et al. brought up with the AMR data structure [BO84,BC89], which enables

simulations to focus on more interesting parts in the data. Cells in data have different refinement levels according to their relative importance. The more important the area is, the smaller are the cells in this area. There are different subdivided partitioning of AMR data such as block-structured AMR [CGL*00] octrees [BWG11].
Kähler et al. proposed [KSH03,KWAH06] the data structure that discards the original AMR data hierarchy and build blocks containing only same-refinement-level cells. While their works excel in supporting nearest neighbor reconstruction and vertex-centered AMR, the challenge of smooth interpolation for cell-centered data remains unaddressed. This Challenge was addressed by Weber [WZM12] on CPU by parallelly generating stitch cells using extra layers of cells around boundaries. Wald et al. [WBUK17] presented tent-shaped basis function and computed the average weight using the reconstruction filter, enabling the sooth interpolation. In 2019 Wang et al. [WWW*19], presented reconstruction filters utilizing octants around a cell and operate directly on cell-centered AMR data.

2.2.2 ExaBrick Framework

In the year 2020, Ingo Wald and colleagues [WZU*20] introduced a significant advancement in the realm of computer graphics through their

research paper titled "Ray Tracing Structured AMR Data Using ExaBricks." This paper delves into an in-depth exploration of practical models and rendering techniques tailored for AMR datasets.

The authors' focal point is the proposal of a ray tracing-based solution known as ExaBricks, meticulously engineered for the efficient rendering of structured AMR data. By reconfiguring AMR data into brick-like structures, inspired by Kähler and others [KSH03,KWAH06], ExaBricks tackles the intricate challenge of achieving smooth interpolation. This is accomplished by integrating the overlapping support regions into a non-overlapping spatial partitioning scheme termed "Active Brick Region" (ABR). A notable strength of the ExaBricks technique lies in its strategic alignment with contemporary GPU capabilities, particularly harnessing NVIDIA's hardware RTX by leveraging the OptiX framework [PBD*10]. This alignment ensures that ExaBricks can optimally exploit the power of modern GPUs for efficient ray tracing operations.

Zellmann et al.'s [ZSM*22] subsequent extension of ExaBrick to support steady flow visualization using particle tracing. Later Zellmann et al. [ZWSH*22] harnessed modern workstations and APIs to visualize exa-scale time-varying AMR datasets and achieved smooth animation. Addressing the challenges on scattering events and global illumination, Zellmann et al. [ZWSM*22] implemented efficient volumetric path tracing for AMR data using Woodcock tracking [WMPT65] on ABRs.

2.3 VR Visualization Techniques

Various visualization techniques have been explored in the context of VR environments, aiming to enhance user experiences and enable immersive interactions with complex data and simulations [SC03].
The paper "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE" [CSD93] describes the CAVE (CAVE Automatic Virtual Environment) virtual reality and scientific visualization system. This paper demonstrates how projection technology applied to virtual reality goals achieves a system that matches the quality of workstation screens in terms of resolution, color, and flicker-free stereo. The CAVE system uses off-axis perspective projection techniques, which are shown to be simple and straightforward. It addresses the reduction of common tracking and system latency errors, and discusses the advantages and disadvantages of the projection paradigm.
In the CAVE, the position of the projected area corresponds to the position of the actual wall. Thus, when the viewer moves through the environment, the off-axis stereographic projection is calculated based on the position of the viewer relative to the walls (Image 2.2).

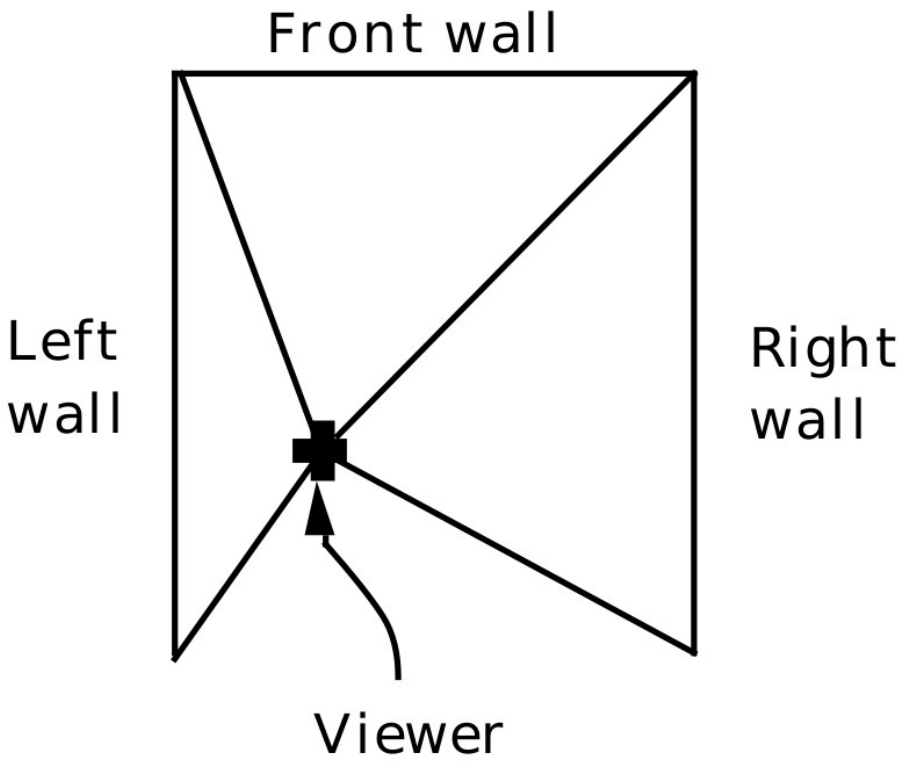


Image 2.2: Off-axis Projection in CAVE [CSD93]

There is also CAVE-like VR systems like this reconfigurable VR system Hor et al. [Hor18] presented. It integrates 6-degree-of-freedom haptic interaction and 3D surround sound audio. The system consists of four large projection screens using high-end projectors capable of rendering perspective orthographic and single-view stereoscopic projections. The proposed system also includes a 6DOF INCA haptic system and a 3D surround sound audio system. The paper details the design specifications of the system, including different configurations and use cases.

The study by Kalarat and Koomhin (2019) [KK19] developed a Virtual Reality application using the Oculus Rift headset and Oculus Touch controllers for real-time volume rendering interaction with 1D transfer functions, enabling users to visualize stereoscopic images at 60 frames per second and achieving high usability with an average score of 87.54 in the usability evaluation.

Utilizing the advantages of VR techniques, scientists are able to visualize data in different domains, for example Koger et al. in 2022 [KHYD22] investigated the potential of VR as an immersive platform for interactive medical analysis, enabling the manipulation and exploration of CT images of the cardiopulmonary system in a 3-D environment, facilitating the generation of new data analysis perspectives and enhancing data interpretation for medical practices including training, education, and investigation.

2.4 COVISE and COVER

The COVISE visualization framework [Ran95], along with its COVER module [RFL*98] (later OpenCOVER), serves as a pivotal avenue for rendering scientific data within VR environments. This integration of virtual reality with scientific visualization leverages OpenSceneGraph [Ope23] and OpenCOVER, enabling real-time 3D rendering in COVISE.

COVISE's versatility is highlighted by the incorporation of various rendering approaches, exemplified by Woessner et al.'s volume plugin [SWWL14] and by Zellmann et al’s Visionary library [ZWL20], which, notably, inspired ExaBrick's plugin development.

2.5 Limitations and Gaps in Existing Work

Despite the significant advancements in real-time rendering of AMR data in VR environments, there are still limitations and gaps that need to be addressed. Many existing approaches focus on specific aspects such as rendering techniques or interaction methods, while comprehensive solutions that address the challenges of large-scale AMR data in VR environments remain limited. Additionally, performance optimization techniques may need further exploration to achieve a balance between rendering speed and visual quality.

3 Background

In this section we explain the important theoretical foundations that we need for the ExaBrick Plugin in COVISE. First of all, we need the ExaBrick Framework developed by Wald et al. [WZU*20] as the basis to take care of the major AMR data processing and rendering part. Our plugin should similarly to the exaViewer, be able to read and render data in ExaBricks data structure. Thanks to the platform COVISE and the module OpenCOVER combining the VR techniques, we can integrate ExaBrick as a plugin into COVISE, and render the AMR data in VR environments.

3.1 ExaBrick Framework

ExaBrick is a specialized framework or tool that facilitates the visualization and analysis of large-scale AMR data. Since this work is an extension of ExaBrick framework in VR environments, the most

important functions and the principle of the ExaBrick tools and viewers will be introduced in this section. Particularly, they reorganize the AMR data structure into ExaBricks data structure with tools like exaBuilder, render the data with a ray-tracing based renderer, and display the frames on a GUI, the exaViewer. Further details that is not explained here can be found in the paper from Wald et al. [WZU*20] and from Zellmann et al. [ZSM*22,ZWSH*22,ZWSM*22].

3.1.1 AMR Data as Input

Adaptive Mesh Refinement (AMR) is a computational technique widely used in scientific simulations to achieve high resolution in regions of interest while maintaining a coarser representation in less significant areas. AMR data is characterized by its hierarchical structure, where the computational domain is divided into a series of nested grids with varying levels of refinement. This approach allows for efficient and accurate simulations of complex phenomena with dynamic spatial characteristics. The first step to do in ExaBrick is to reorganize the AMR cells into a set of non-overlapping bricks in an efficient way for memory storage.

3.1.2 Generating Bricks of Same-Level Cells

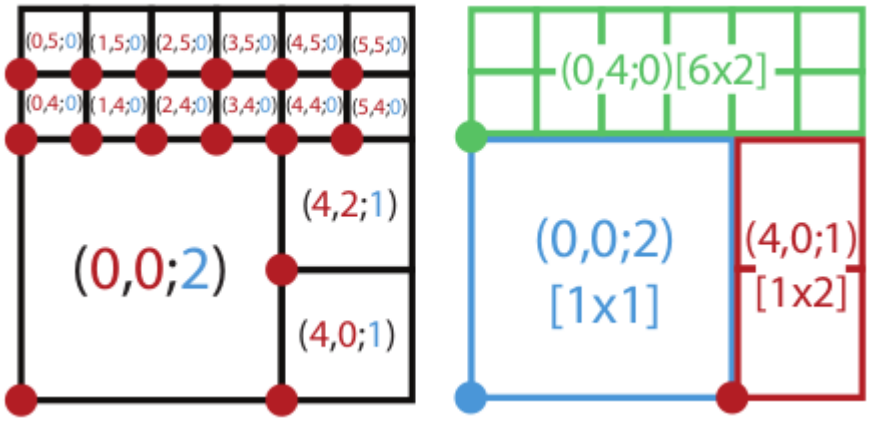


*Image 3.1: a 2D illustration of Cells and Bricks, (a) cells labeled with (i,j;l) i and j for cell location according to the left lower point and the l for refinement level, (b) cells organized into bricks of size [mxn] with same refinement level l [WZU*20]*

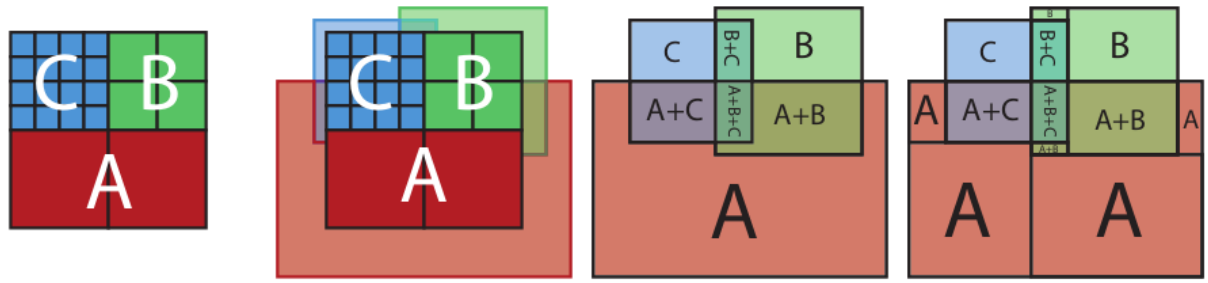
As outlined by Aftosmis et al. [ABA00] and Wald et al. [WZU*20], cells represent the smallest volumetric unit within the computational grid or mesh of AMR data (Image 3.1 (a)). Refinement level 0 corresponds to the finest level with the smallest cell size in the AMR data. Refinement levels are inversely proportional to cell size; higher refinement levels denote larger cells and coarser representations, as articulated in the literature. The bigger number (less fine) the refinement level of a cell is, the bigger is the cell size. Within ExaBrick, in alignment with Kähler et al. [KSH03], the inherent hierarchical details of the AMR data are

discarded. Instead, cells of the same level are structured into rectangular bricks (Image 3.1 (b)). This process employs a top-down k-d tree, wherein the leaf nodes exclusively encompass cells of the identical level.

The advantage here is that with the brick-structured AMR data, original hierarchy can be discarded, and only the resulting leaves need to stored. Limitation is that, if the neighbor bricks have level gap, the rendered picture will have gap at transition region between bricks, so that the interpolation is not smooth. Also the usable AMR data type for this approach is limited.

3.1.3 Constructing Active Brick Regions

To solve the challenge in the previous subsection (Section 3.1.2) Wald et al. extended each visible brick with the width of half a cell in each direction to build support regions (Image 3.2 (b)). And using a k-d tree, those overlapping support regions are subdivided into rendering-friendly rectangulars, that are called Active Bricks Regions (ABRs) (Image 3.2 (c) and (d)). An ABR stores a list of bricks that are "active" in this region and the position and level information in a low overhead storage. Building hardware acceleration over ABRs supports smooth interpolation for neighbor bricks with different level.



(a) (b) (c) (d)

*Image 3.2: A 2D Illustration of our Active Brick Regions: (a) A data set with three bricks, each of a different refinement level. (b) The brick support regions corresponding to each brick. (c) The overlap of these supports forms a spatial partitioning where each region knows which bricks are "active" within it. (d) We subdivide these regions into non-overlapping rectangular regions which we can traverse as before [WZU*20].*

3.1.4 Ray Traversing and Rendering with the Bricks

- Real-time rendering refers to the process of generating and displaying visual content in a responsive manner, typically at interactive rates of at least 15 frames per second. As input, the computer is given a sequence of information like the object vertices coordinates/transparency/texture, lighting settings, camera type and position, etc. And the computer calculates transparency, lighting effect (like reflecting, shading etc) for every voxel in the 3D space and decide the color for every pixel, that should be displayed on the screen. A display of for example 30 rendered images within 1 second, means a real-time rendering of 30 fps.

- Ray tracing, a fundamental technique in 3D volume rendering, hinges on the concept of rays emulating lines of sight (see example in Image XX in section 3.5.2) that originate from the viewpoint of the virtual camera. This visualization technique simulates how light rays travel through a scene, striking surfaces and generating visual information. The culmination of this intricate process results in the continuous display of meticulously computed images on a viewport, most commonly a computer screen. The core of ray tracing involves projecting rays through each pixel on the viewport and calculating intersections with objects in the scene. These intersections yield crucial surface data like color and transparency, ultimately determining the pixel's final appearance.

To render frames (computer rendered images) with ExaBricks data structure, ExaBrick uses the ray tracing engine, OptiX 7.0 [Par10], to build BVH on ABRs (Section 3.1.3) and traverse them on GPU. The heart of this process lies in the ray generation program, intricately implemented within ExaBrick [WZU*20], where ray intersects with ABRs and integration within the bricks is performed using the rendering methods in the subsequent section.

3.1.5 Space Skipping and Adaptive Sampling

A lot of parameters for render settings, such as opacity, ray marching step size, iso-surfacing threshold, are initialized in program as selected standard value and also enabled to be manually changed on the GUI, the exaViewer, so that the user can interactively customize and manipulate the data. Upon changes of the settings, the volume BVH will be rebuilt over the ABRs. The interaction part will later also be integrated into the COVISE plugin we developed in this thesis (Section 4.5).

Empty spaces such as holes inside the mesh or areas outside the mesh will be skipped, as well as for regions, whose maximum opacity is or changed to 0 will be skipped.

Fetching samples adaptively enables us to render AMR data in different frequency, there will be more samples taken in ABRs of fine level cells than in ABRs of coarse level cells and this enables an adaptive domain resolution. The base sampling rate is defined as two samples per smallest cell (ray marching step size dt=0.5), and this variable can be scales manually by the user on GUI as well. Reasonably, increasing the ray marching step size will increase the distance of each samples and reduce the computational burden, increase the performance but sacrifice details in the rendered images (discussed in Section 5.5.1 and 6).

3.2 ExaViewer and More

Having the theories above, Wald et al. developed the exaViewer as a viewing tool and GUI to display the rendered AMR data in ExaBricks data structure and enabled interactions (Image 3.3).

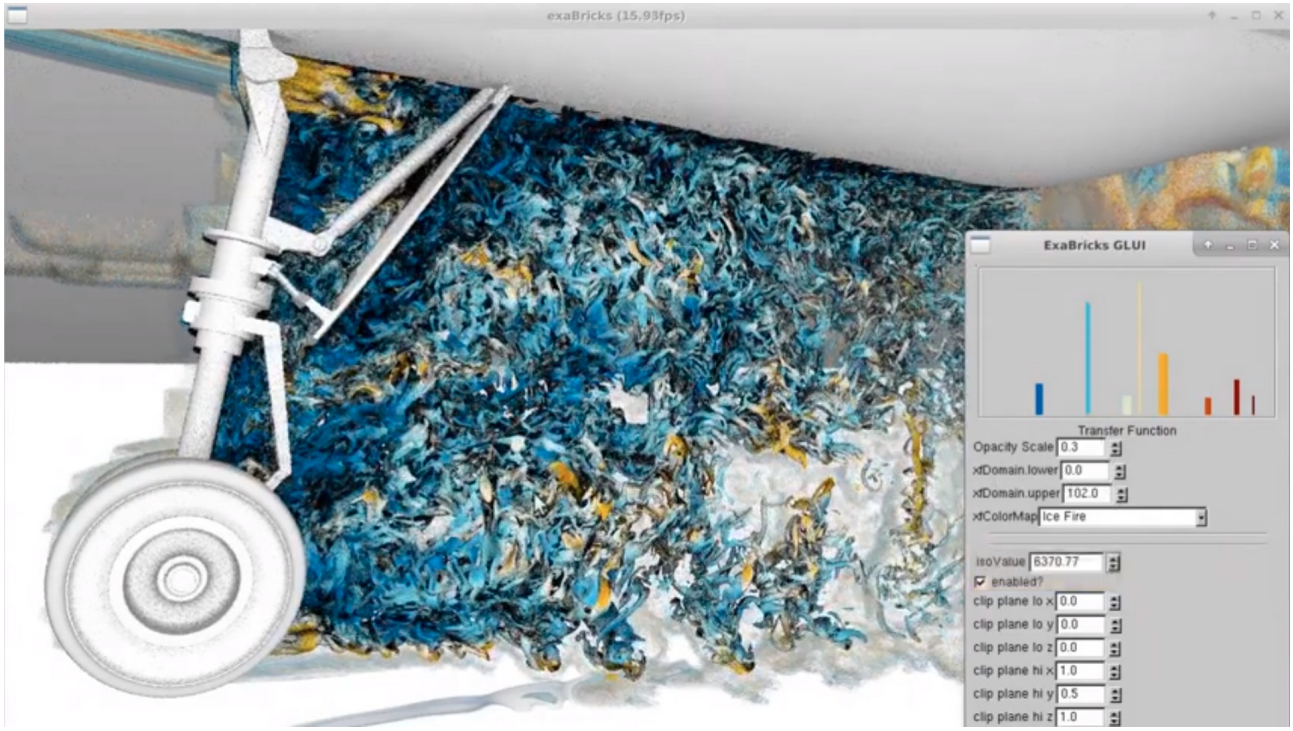


*Image 3.3: ExaViewer displaying the NASA Landing Gear, a simulation of air flow around an airplane's landing gear [WZU*20], interactions with GUI of ExaViewer*

The exaViewer, while valuable, does come with limitations. It relies on an on-axis camera, strategically placed in direct alignment with the scene's center. Consequently, it captures scenes from a viewpoint where the camera's optical axis is congruent with the line of sight to the center

of the scene. This yields a straightforward and centrally-focused perspective of the scene.

For a more immersive experience that caters to users' natural head movements and exploratory look around in the VR space, an off-axis camera is essential. Such a camera is positioned at an angle that doesn't consistently align with the scene or object's center (implemented in Section 4.6.2).

With the development of VR techniques and the COVISE platform, we have the opportunity to build extension of ExaBrick and perform 3d rendering in VR environments.

3.3 Rendering in Virtual Reality

Virtual reality is a transformational technology that immerses users in a computer-generated, simulated environment, replicating real-world experiences or creating entirely fictional worlds. VR has found applications across various industries, including gaming, education, healthcare, architecture, and training. Its potential extends beyond entertainment, empowering users to explore distant locations, visualize complex data, and engage in realistic training scenarios. The immersive nature of VR provides a compelling platform for visualizing and

analyzing complex scientific data, offering researchers new perspectives and insights.

Cave Automatic Virtual Environment (CAVE) is a room-sized VR system that utilizes multiple walls or screens to create a fully immersive virtual environment. The CAVE environment enhances the sense of presence and spatial awareness, allowing users to navigate and interact with virtual objects as if they were physically present. This makes CAVE systems particularly suitable for visualizing and analyzing scientific datasets, including large-scale AMR simulations.

For the Performance Test Section of this paper we use the CAVE built in Regional Computing Centre (RRZK) of University of Cologne [CAV23] (Image 3.4), important technical information can be found in Section 5.1.

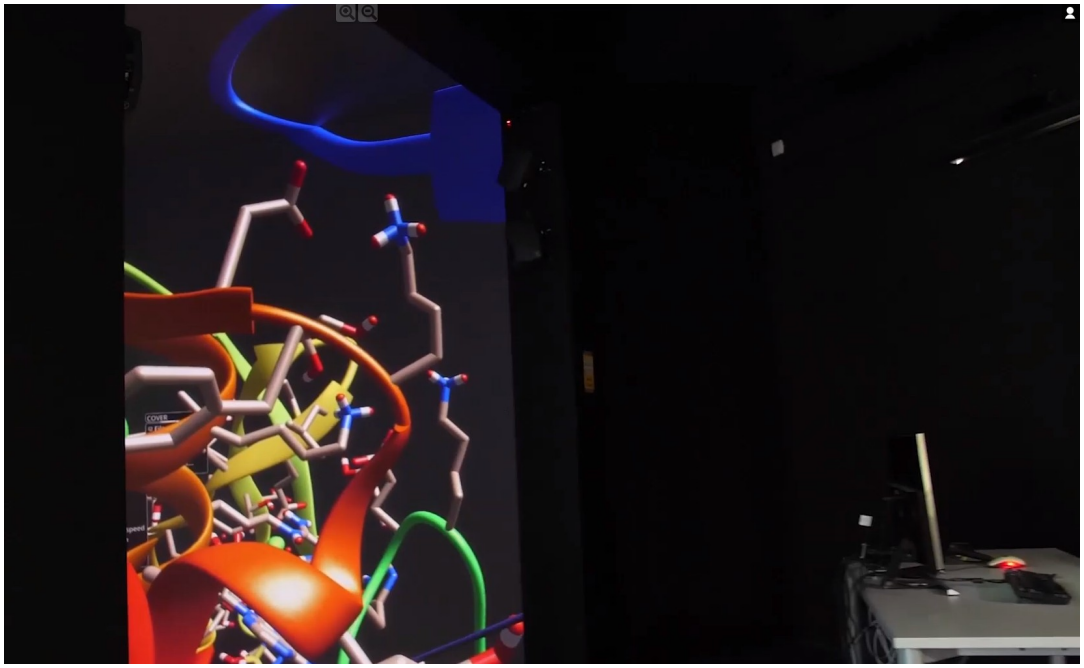


Image 3.4: CAVE in RRZK University of Cologne [CAV23]

3.4 COVISE and OpenCOVER

COVISE is a comprehensive software framework that integrates visualization, simulation, and data analysis capabilities (Image 3.5). It enables integration of various modules and plugins [Ran95,SWWL14], allowing users to enhance their workflows and leverage specialized functionalities, such as real-time rendering and VR capabilities.

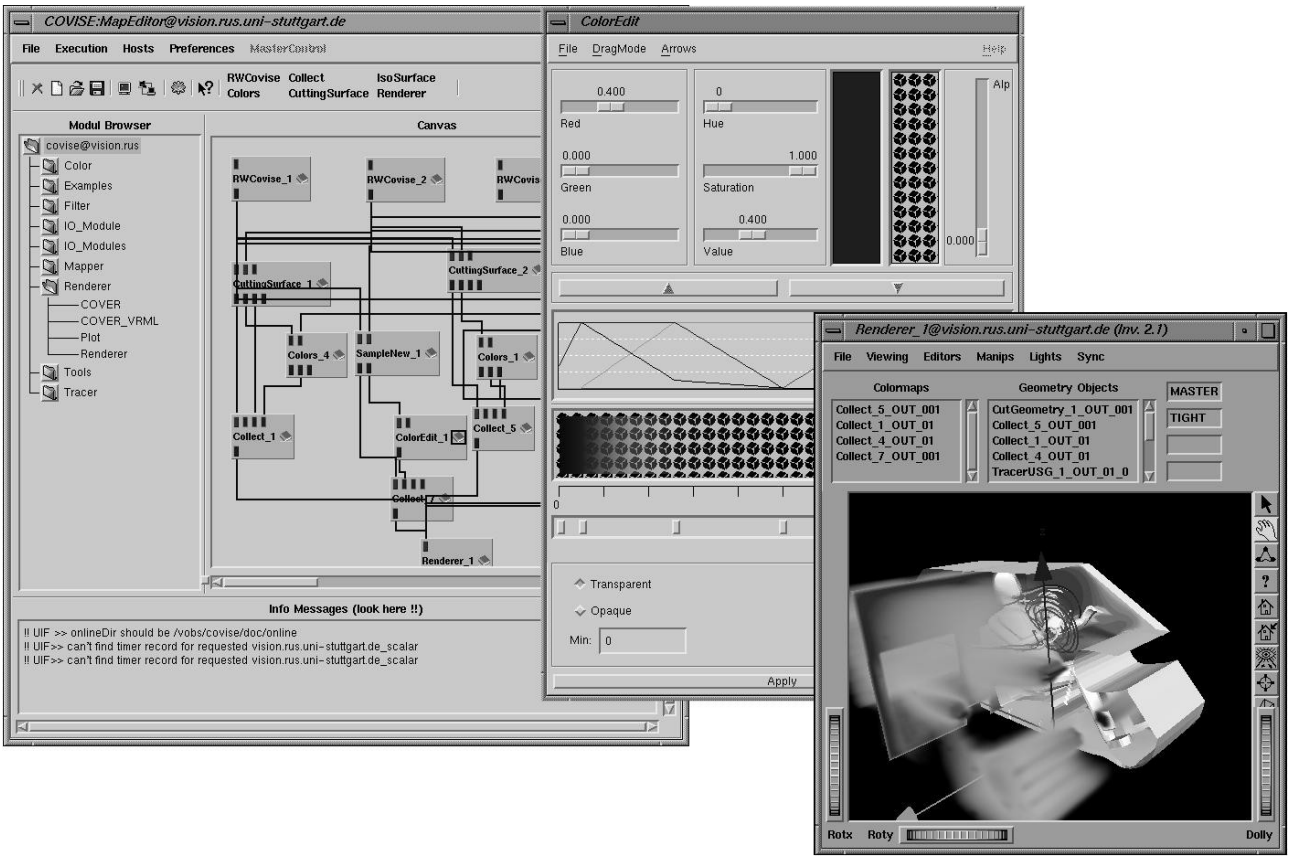


Image 3.5: COVISE desktop user interface and OpenInventor renderer [SWWL14]

OpenCOVER is a COVISE renderer module with VR support, and can be used as a VR viewer for 3D geometry. OpenCOVER uses the

OpenSceneGraph (OSG) toolkit, which is a OpenGL-based (for “Open Graphics Library”) portable, high-level graphics toolkit for high-performance graphics development. It uses the OpenGL graphics API for rendering. This means the rendered images will show on the OpenCOVER window and later projected on the CAVE walls.

3.5 OpenGL and Coordinate Transformations

It is mentioned (Section 3.2) that an off-axis camera need to be built instead of the original camera in ExaBrick. As explained (Section 3.1.2), in ray tracing we use rays to simulate the sight of line from the eyes to the object. The new camera should be able to calculate where the ray starts and its direction like the old one, and the ray generation program in ExaBrick (Section 3.1.4) will do the rest of the ray tracing work.

In the this subsection, I will introduce how the rays should be calculated for the ray-traversing through the ABRs (Section 3.1.3).

To calculate the rays, we need to have an overview of OpenGL pipeline [Sil90,SA97] , to understand the relative coordinate of the ray origin and ray direction.

Our goal is to give OpenGL vertices, that we want to show on the screen, in normalized device coordinates (NDC), where the x, y and z

coordinates of the vertices are between -1.0 and 1.0. So that the coordinates outside this range can be clipped of before displaying. After that these NDC are transformed by the rasterizer to the 2D coordinates, that are displayed on the screen. To let our object show on the screen, we need to perform some transformations to the object, so that their vertices are transformed into NDC.

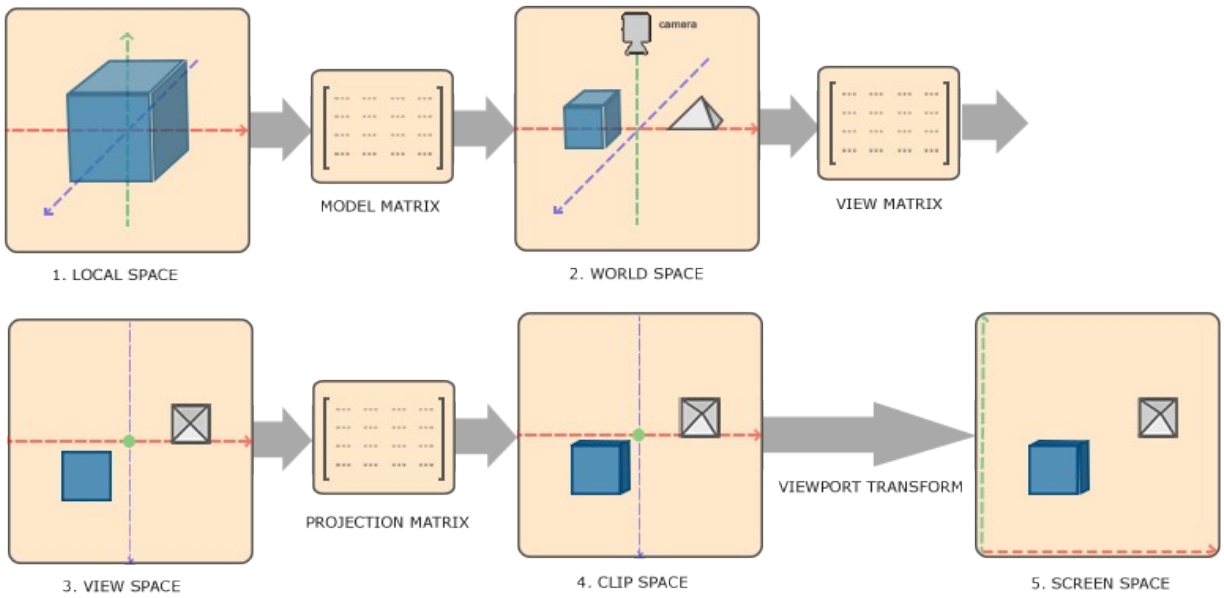


Image 3.6: Coordinate Systems

3.5.1 Forward Coordinate Transformations

In the OpenGL pipeline, the vertices of 3D objects are initially specified in their local coordinate systems (also known as model space) (Image 3.6, 1). One might have several objects to display, and they may all have

(0,0,0) as their initial position. All vertices undergo transformations as they move through different stages of the pipeline.

The model transformation places the vertices of one or more objects in the world coordinate system by multiplying the model matrix (Image 3.6, 2). The model matrix is a transformation matrix in charge of translating, scaling and rotating the object vertices into world coordinates. So we can position the objects into the 3D world the way we want during model transformation.

According to the camera position, the object vertices from world coordinates are mapped to coordinates in view space (also known as camera space or eye space) with the view matrix (Image 3.6, 3). In this space, objects are presented as the viewer is seeing from the camera.

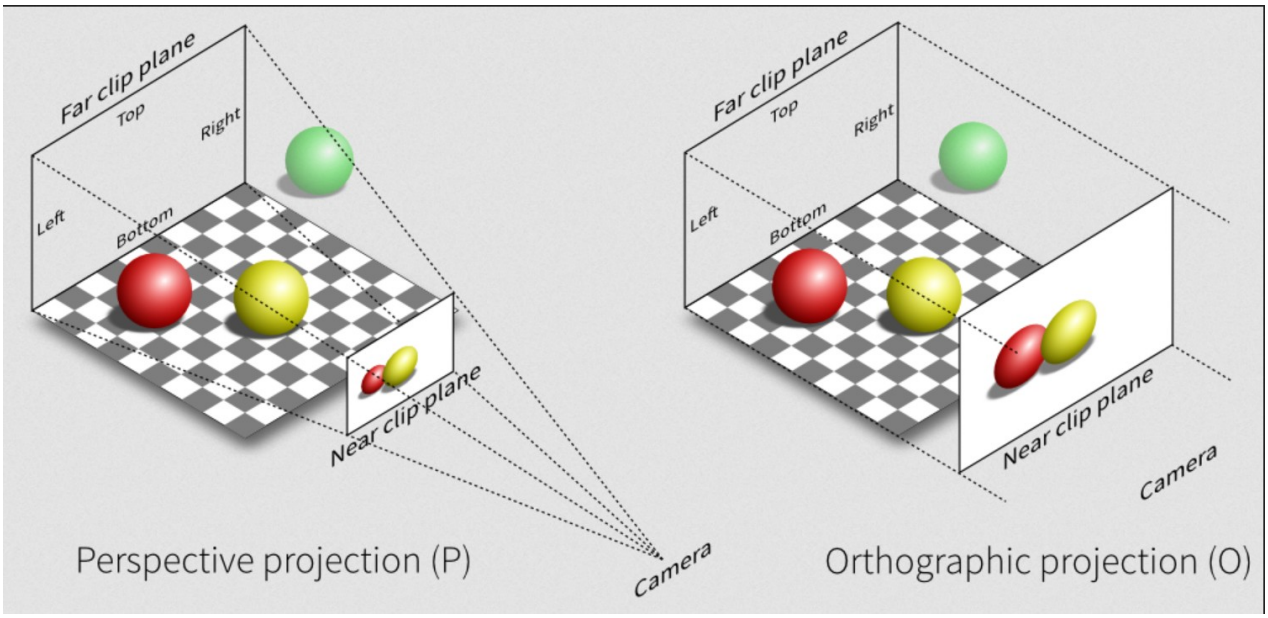


Image 3.7: Perspective projection and Orthographic projection

To project 3D volume onto a 2D screen, there are different ways (Image 3.7). For a realistic VR experience, we need the perspective projection to make object near the camera look bigger, and those are far from camera look smaller. For that we can define the parameters to build a near plane and a far plane. Between the near plane and far plane, there is a frustum with all the relevant objects for the display. Everything outside the frustum are not to consider for the display and will be discarded. With the projection matrix the 3D world in the frustum will be “compressed” onto a 2D screen, which means the projection projects the camera space coordinates to normalized device coordinates (NDC) (Image 3.6, 4 and 5). Detailed description can be seen in this Graphics Library Programming Guide from Silicon Graphics, inc. [Sil90].

3.5.2 Calculating Rays Backwards with Inverse Matrices

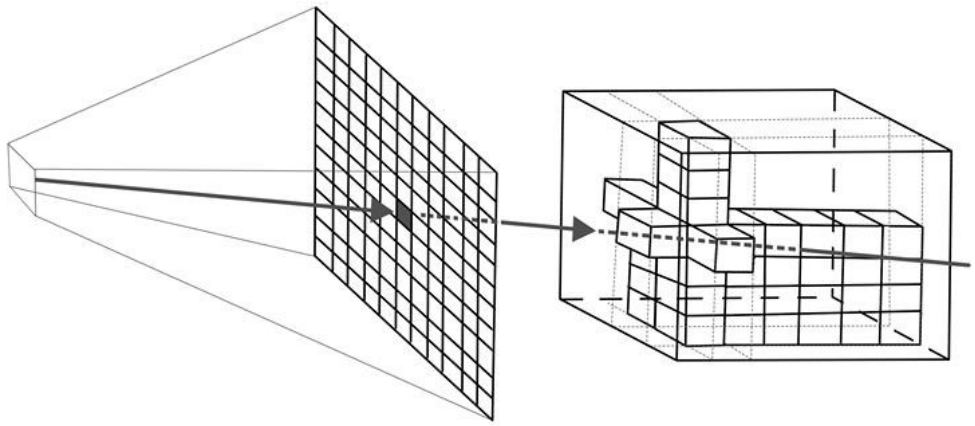


Image 3.8: An example of Ray marching using backward methods, a ray goes through one pixel on the screen and finds intersection point with the object [KK19]

In virtual reality environments, off-axis cameras are utilized to match the viewer's perspective and create a more immersive experience. These off-axis cameras introduce complexities in ray generation because conventional projection matrices are insufficient. To address this challenge, the ExaBrick plugin employs a technique that involves obtaining the modelview and projection matrices from the OpenGL viewport. By calculating the inverse of these matrices, the pixel coordinates will be transformed from the screen space to the local space.

In volume rendering, rays are cast from the viewer's eye (or camera) into the 3D volume data to calculate the color and opacity of each voxel along the ray path. Since now it is already defined in OpenCOVER about how the OpenGL is going to transform the object from local space to screen

space, I can obtain the model matrix, view matrix and projection matrix from OpenCOVER, as well as the pixel coordinate my ray is going through (see example in Image 3.8). With these information from OpenCOVER, I can calculate backwards to generate the ray origin and ray direction in local space, tell my renderer the ray information and let the OptiX-renderer start the rendering.

```
1  (* two opposite points in normalized device coordinates *)
2  u ← (thread.x/(imagewidth − 1)) ∗ 2 − 1
3  v ← (thread.y/(imageheight − 1)) ∗ 2 − 1
4  ori ← make_float4(u, v, −1, 1)
5  dir ← make_float4(u, v, 1, 1)
6  (* convert back to eye coordinates *)
7  ori ← InvProjectionMatrix ∗ ori
8  dir ← InvProjectionMatrix ∗ dir
9  (* convert back to object coordinates *)
10 ori ← InvModelviewMatrix ∗ ori
11 dir ← InvModelviewMatrix ∗ dir
12 (* divide out homogeneous coordinate *)
13 ori.xyz ← ori.xyz/ori.w
14 dir.xyz ← dir.xyz/dir.w
15 (* make dir a direction vector by subtracting the two points and
   normalizing *)
16 dir.xyz ← dir.xyz − ori.xyz
17 normalize(dir.xyz)
18 make_ray(ori.xyz, dir.xyz)
```

Image 3.9: Algorithm calculating ray backwards using inverse projection and modelview matrices [Zel14]

As described for example in the dissertation of Zellmann (Image 3.9), every pixel sample is firstly converted into NDC (Image 3.9, Line 2-3). Then the NDC are multiplied by the inverse matrix of projection and modelview matrix (model matrix times view matrix) , so we can generate the ray which goes through each pixel (Image 3.9, Line 18). Note the order of multiplication with inverse matrices is also reversal, because the projection matrix is the last matrix multiply in the Section 3.5.1.

In this way, the new off-axis camera can be built. And that is all the theoretical background we need for the plugin, we can now start the implementation.

4 Technical Implementation

4.1 Functions to Implement

The implementation of the real-time rendering plugin involves several key functions essential for enabling the visualization and interaction with large AMR datasets in VR environments. The main programming language used in this section is C++. Declarations and initializations of variables or functions will be found in the header data ExaBrickPlugin.h and are not listed here due to the length cause. Important code snippets will be presented here, mostly from the data ExaBrickPlugin.cpp, and the complete code is available on the GitHub page [Wan23]. Information one might need to rebuild the plugin and the performance shown in section 5 is written in the file README-ExaBrick.txt.

4.2 Structure of the Plugin

The ExaBrick plugin follows a structured workflow that includes initialization, data loading, setting up initial values and menus, creating the renderer, executing the render loop, and the ending phase. The figure below (Image 4.1) illustrates an overview of the plugin's structure, which will be explained in subsequent subsections.

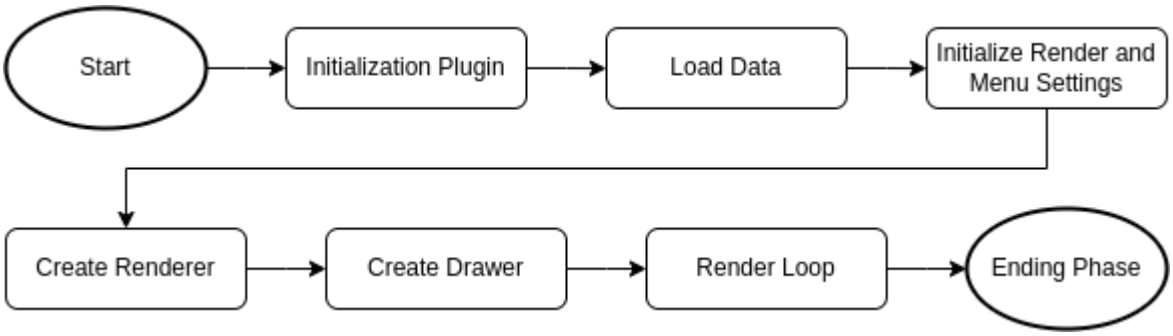


Image 4.1: Flowchart for a brief Structure of the Plugin

4.3 ExaBrick as a Third-Party Library

The ExaBrick plugin utilizes various functions and tools from the original ExaBrick library, available at [Exa23]. For successful implementation, ExaBrick Tools and Viewer need to be built locally on the user's device, linked into the 3rdparty folder in COVISE. The UseExaBrick.cmake file [Wan23] contains configuration directives that ensure proper linkage and access to the necessary ExaBrick libraries.

4.4 Load Data and Create Renderer

The corresponding code for loading data is encapsulated in the “loadFile()” function using the File Handler in COVISE. The File Handler is defined to read file type of “.exa”, along with the “.scalar”, and “.bricks” data in the current folder. The scalar and bricks data can be converted from AMR cells or other formats using tools from owlExaBrick, such as exaBuilder [Exa23]. The ExaBrick plugin registers a File Handler of COVISE to load files with the ".exa" extension, which is text-based. The function "parseConfigFile()" from the original ExaBrick Tool is then used, followed by the creation of the Optix-Renderer of ExaBrick, utilizing NVIDIA OptiX 7.0.

4.5 Initialize Render and Menu Settings

During this phase, a multitude of parameters for the renderer are initialized. This encompasses setting up transfer function values, configuring rendering preferences, and determining the ideal ray marching step size. Furthermore, the incorporation of VR menus facilitates dynamic data manipulation. These menus facilitate interactive

alterations encompassing adjustments in opacity, the fine-tuning of iso-surfacing and contour plane configurations, and the activation of diverse rendering alternatives. These settings are established directly after the creation of the OptixRenderer.

4.6 Render Loop

The Multi-Channel-Drawer class serves as a versatile and flexible tool for managing graphics rendering with support for multiple views and stereoscopic rendering. The class can handle various rendering modes, update geometry, and handle the data for each view independently or collectively, depending on the selected view settings. The Multi-Channel-Drawer is used in the plugin to manage rendering and displays.

A Multi-Channel-Drawer will be created (Image 4.2, Line 3) before the render loop is initiated with the function “renderFrame()”. Within the render loop, essential steps will be explained in subsequent subsections.

```
1  if (!multiChannelDrawer) {
2   std::cout<< "creating multiChannelDrawer..."<<"\n";
3   multiChannelDrawer = new MultiChannelDrawer(false, false);
4   multiChannelDrawer->setMode(MultiChannelDrawer::AsIs);
5   cover->getScene()->addChild(multiChannelDrawer);
6   channelInfos.resize(numChannels);
7  }
8  for (unsigned chan=0; chan<numChannels; ++chan) {
9   renderFrame(info, chan);
10 }
```

Image 4.2: Code snippet for the render loop from function "renderFrame()" in ExaBrickPlugin.cpp.

4.6.1. Auto FPS Mode

In the preframe Phase, the fps will be measured and compared with the user chosen fps for the auto fps mode (discussed in Section 5.5.4). If the selected fps is not reached, the plugin dynamically adjusts the value of Ray Marching Step (Image 4.3, Line 18 and 26) to change the fps until the desired value is achieved.

```
1  // Measure fps:
2 // float end = cover->frameTime();
3 float end = cover->frameDuration();
4 if (firstFPSmeasurement)
5 {
6  firstFPSmeasurement = false;
7  fps = INITIAL_FPS;
8 }
9 else
10 {
11  // add a wait to prevent the loop from running too fast
12  std::this_thread::sleep_for(std::chrono::milliseconds(frequence));
13  fps = 1.0f / (end - start);
14 }
15
16 // increase ray marching step to let fps increase
17 if (fps < plugin->cmdline.chosenFPS - 0.5*amplitude){
18  plugin->cmdline.dt += speed; // change speed of dt
19  plugin->rayMarchingStepSizeSlider->setValue(plugin->cmdline.dt);
20  plugin->resetAccumulation();
21  // add a wait to prevent the loop from running too fast
22  std::this_thread::sleep_for(std::chrono::milliseconds(frequence));
23 }
24 // drop ray marching step to let fps drop
25 if (fps > plugin->cmdline.chosenFPS + 0.5*amplitude){
26  plugin->cmdline.dt -= speed; // change speed of dt
27  plugin->rayMarchingStepSizeSlider->setValue(plugin->cmdline.dt);
28  plugin->resetAccumulation();
29  // add a wait to prevent the loop from running too fast
30  std::this_thread::sleep_for(std::chrono::milliseconds(frequence));
31 }
```

Image 4.3: Code snippet from function “preFrame()” in ExaBrickPlugin.cpp.

4.6.2 Transfer Matrices for the Off-axis Camera

To facilitate the inverse transformation of pixels from world to local space, modelview and projection matrices need to be obtained from the

OpenCOVER viewport (Section 3.5.2). These matrices enable the renderer to calculate backward to derive the local coordinate of each ray.

The following code snippet demonstrates how the camera matrices are obtained from the Multi-Channel-Drawer (Image 4.4, Line 2-3) and sent to the renderer (Image 4.4, Line 11) with the function “setCameraMat()”:

```
1  // get the matrices and send to Renderer
2  osg::Matrix mv = multiChannelDrawer->modelMatrix(chan) * multiChannelDrawer-
>viewMatrix(chan);
3  osg::Matrix pr = multiChannelDrawer->projectionMatrix(chan);
4  math::mat4f view = osg_cast(mv);
5  math::mat4f proj = osg_cast(pr);
6  // to update camera when the viewport changes
7  if (notsame(channelInfos[chan].mv,view) || notsame(channelInfos[chan].pr,proj)) {
8   cout<<"updating matrices......."<<"\n";
9   channelInfos[chan].mv = view;
10  channelInfos[chan].pr = proj;
11  plugin->renderer->setCameraMat(view,proj);
12 }
```

Image 4.4: Transfer camera matrices from function “renderFrame()” in ExaBrickPlugin.cpp.

4.6.3 Off-Axis Camera and Calculation of Rays

The original ExaBrick uses an on-axis camera with the function “generateRay()”. However, since the plugin requires an off-axis camera,

we have extended the functionality of this function to calculate rays suitable for the plugin's needs. The ray generation involves converting pixels to NDC, multiplying the inverse matrices (Image 4.5, Line 19-20) and generating the rays with the corresponding calculations (Section 3.5.2).

The following code snippet demonstrates the calculation of ray for the off-axis camera:

```
1  // off-axis camera
2  static __device__ owl::Ray generateRay(const FrameState &fs,
3  const vec2f &pixelSample,
4  Random &rnd,  int width,  int height,
5  math::mat4f view,  math::mat4f proj)
6  {
7   // convert pixel to NDC [0,1] -> [-1,1]
8   // +0,5 to sample pixel centers
9   auto x_NDC = float(2.0) * (pixelSample.x + float(0.5)) / width - float(1.0);
10  auto y_NDC = float(2.0) * (pixelSample.y + float(0.5)) / height - float(1.0);
11  auto inv_view = math::inverse(view); //TODO not here
12  auto inv_proj = math::inverse(proj);
13  // z = -1/+1 point on the near/far plane
14  math::vec4f vec_ori = {x_NDC, y_NDC, -1.0, 1.0};
15  math::vec4f vec_dir = {x_NDC, y_NDC, 1.0, 1.0};
16  auto o = inv_view * inv_proj * vec_ori;
17  auto d = inv_view * inv_proj * vec_dir;
18  const vec3f origin = { o.x / o.w, o.y / o.w, o.z / o.w};
19  vec3f d_point;
20  if(d.w==0){
21   auto a = 100000000;
22   d_point = { d.x * a, d.y * a, d.z * a};
```

```
23  }
24  else{
25   d_point = { d.x / d.w, d.y / d.w, d.z / d.w};
26  }
27  const vec3f direction = (d_point - origin);
28  return owl::Ray(/* origin : */ origin,
29  /* direction: */ normalize(direction),
30  /* tmin : */ 1e-6f,
31  /* tmax : */ 1e8f);
32 }
```

Image 4.5: Ray generation function "generateRay()" in owlExaBrick (owlExaBrick/programs/Camera.h).

4.6.4 Render

```
1 plugin->renderer->render();
2 multiChannelDrawer->update();
3 multiChannelDrawer->swapFrame();
```

Image 4.6: Code snippets from function "renderFrame()" in ExaBrickPlugin.cpp.

After having all the nessiceray parameters set, the render function calls the plugin's renderer to render the scene (Image 4.6, Line 1). The Multi-Channel-Drawer updates its view and channels, and the generated frames are swapped for display. This loop continues until the view is no longer changing.

4.7. Ending Phase

In this phase, the file handler is unregistered, the Multi-Channel-Drawer is removed, and data related to the drawer is cleaned up. This ensures that no display or cached data remains in OpenCOVER.

4.8. Usage ExaBrick Plugin

In this section there is a brief introduction about how to use this Plugin.

4.8.1. Load ExaBrick Plugin

To load the ExaBrick plugin in COVISE, follow these steps:
1) Place a text-based configuration data of type “.xml” into the folder covise/config.
2) Set the environment variable COCONFIG to the name of configuration data using the command “export COCONFIG=file_name.xml”
3) Start OpenCOVER together with the ExaBrick configuration data. For example, use the command “opencover

/home/mueller/data/data_name.exa”, where the scalar and bricks data (transfer function data if needed) are stored in this directory.

4.8.2 Interactive Navigation and Manipulation

Usage of OpenCOVER can be found in the User’s Guide of COVISE Online Documentation [Use23].
To change transfer function settings or rendering options, utilize the buttons or sliders in the VR-Menu. Use the "load TF" button in the "Transfer Function" menu to load the transfer function data, which can be generated with the exaViewer of ExaBrick [Exa23].

5 Performance Test

The performance test section aims to evaluate the effectiveness and efficiency of the implemented solution for real-time rendering of large AMR data in VR environments. The objective is to access the rendering speed and the quality of the rendered output.

5.1 Hardware and Software Information

The results including test data and images etc, are all generated under the following hardware and software setups:

- 3m x 3m x 3m CAVE projecting on five 3m x 3m screens, each screen has 2 GPUs for either eye of the viewer, with VR glasses with head tracker and hand controller
- NVIDIA Corporation TU102GL [Quadro RTX 6000/8000] GPU with 24 GB GDDR memory
- Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz

- 131.5 GB of RAM
- Linux Ubuntu 20.04
- NVIDIA driver version 525.125.06
- CUDA 12.0
- OptiX 7.0

Detailed Information including COVISE configuration data and transferfunction we used can be found in the GitHub link [Wan23].

5.2 Datasets

We have tested the performance of the plugin on two AMR datasets:

a) SILCC Data: data release “DR1 SILCC_hdf5_plt_cnt_0150” of type hdf5 from [SIL23][SWG*17] based on publications [WGN*15] [GWN*16], converted into cells and scalar files using the flash converter of ExaBrick tools [WZU*20].

b) LANL Meteor Impact: Meteor impact simulation datasets (at time t=46.112s) [Dee23] produced by Dr. Galen Gisler et al. at Los Alamos National Laboratory (LANL) [PST*16] using simulation code xRage [GWC*08], aim to study the tsunami generation and propagation when near-Earth Objects (NEOs) hit water.

5.3 Evaluation Criteria

To comprehensively evaluate the performance of this real-time rendering application, we will consider the most important factor: frame rates.

The frames per second (fps) is a crucial metric in real-time rendering, as it directly impacts the smoothness and visual quality of the VR experience. For a usable and comfortable VR experience, we aim to achieve a minimum fps of 12. If the fps drops below this threshold, the display becomes less smooth and may cause discomfort. A good or even fluent performance will be considered achieved if the fps reaches at least 30 or higher.

5.4 Graphic Demonstration

To visually demonstrate the capabilities of the implemented solution, a series of graphical demonstrations will be presented. These demonstrations will showcase the real-time rendering of large AMR datasets in the VR environment, highlighting interactive navigation and visual representation of the AMR data. Through these demonstrations, users will gain insights into the system's ability to handle complex data structures, maintain real-time performance, and deliver an immersive experience. It's noteworthy that the best immersion is achieved not just

through images but through actual exploration with VR glasses and controller in a VR environment.

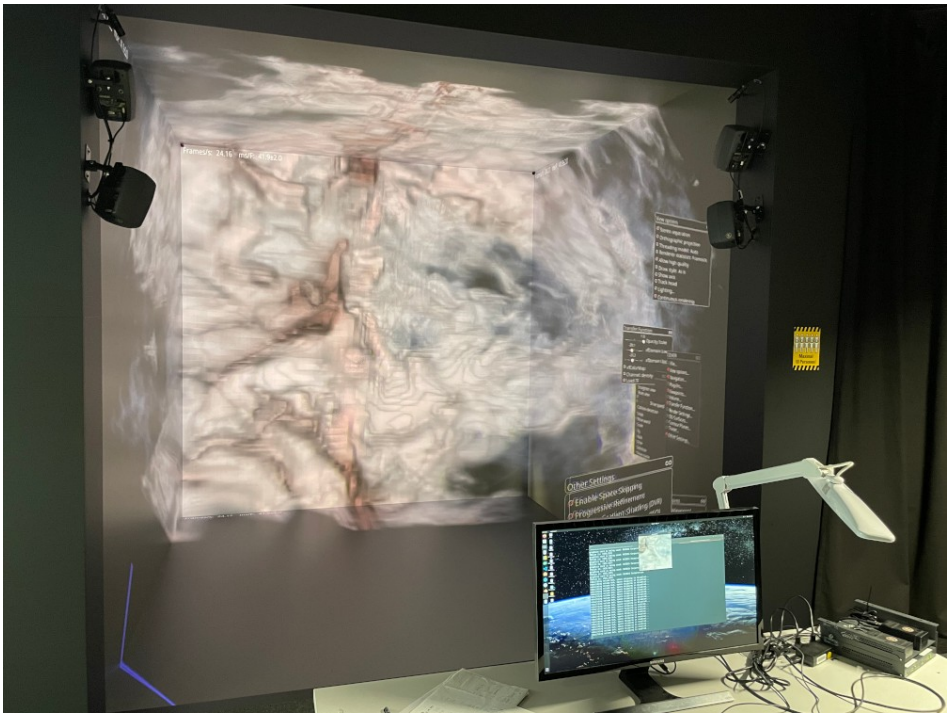


Image 5.1: ExaBrick Plugin rendering dataset molecular cloud using OpenCOVER in CAVE, with the VR menu projected on the right wall.

Image 5.2: ExaBrick Plugin rendering dataset LANL meteor impact using OpenCOVER in CAVE.

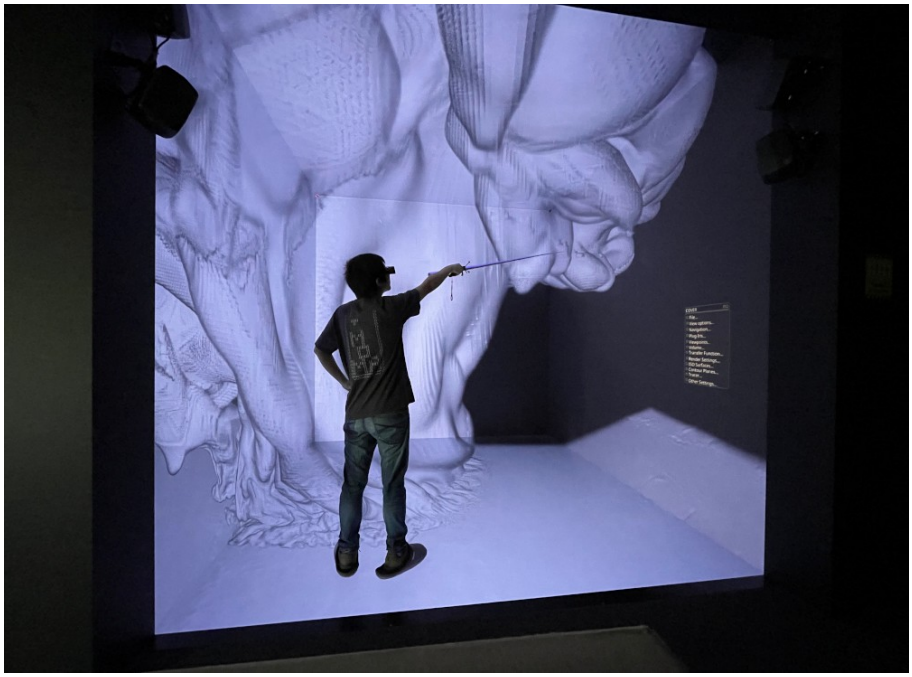


Image 5.3: ExaBrick Plugin rendering the center of dataset LANL meteor impact using OpenCOVER in CAVE, observing and interactively moving into the volume with navigation options.

5.5 Performance

Based on the experience in the CAVE, the rendered images align closely with the visualization results of the ExaBrick Plugin and exaViewer. The VR experience offers an immersive environment, further validating the successful implementation.

5.5.1 Adjusting Ray Marching Step Size

As delineated in Section 3.1.5. on adaptive sampling, the ray marching step size plays a pivotal role in shaping our rendering process. This step size dictates the density of samples taken along each ray's journey through a given region while intersecting with brick boundaries. Amplifying this step size results in a reduction of sampled points, heralding augmented rendering speed (higher fps), thanks to the minimized computational load per frame.

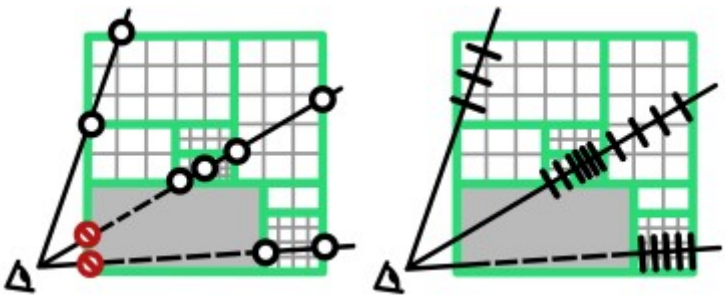


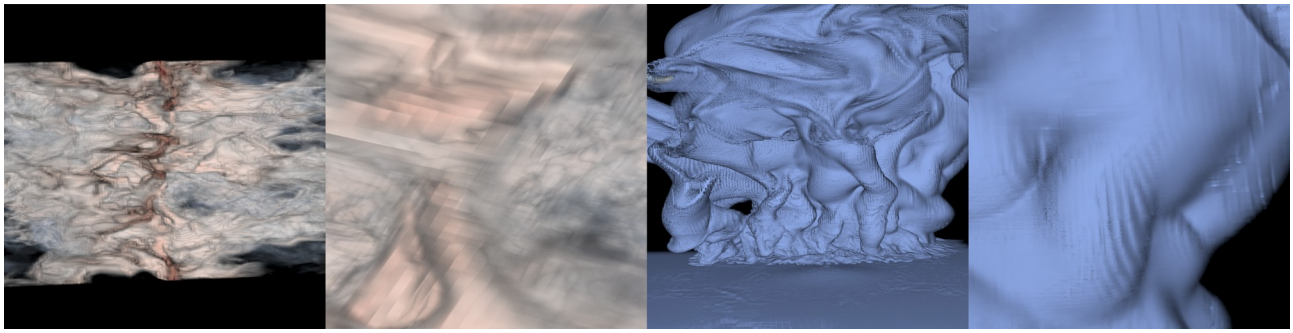
*Image 5.4: Ray Marching Step Size along the ray through the bricks [WZU*20].*

However, a concomitant increase in the ray marching step engenders potential complications. Larger steps heighten the likelihood of bypassing minute intricacies, thus engendering aliasing artifacts and compromised rendering precision. In consequence, the delicate equilibrium between step size and rendering fidelity assumes paramount importance in the realm of ray marching-based rendering techniques.

5.5.2 Measuring

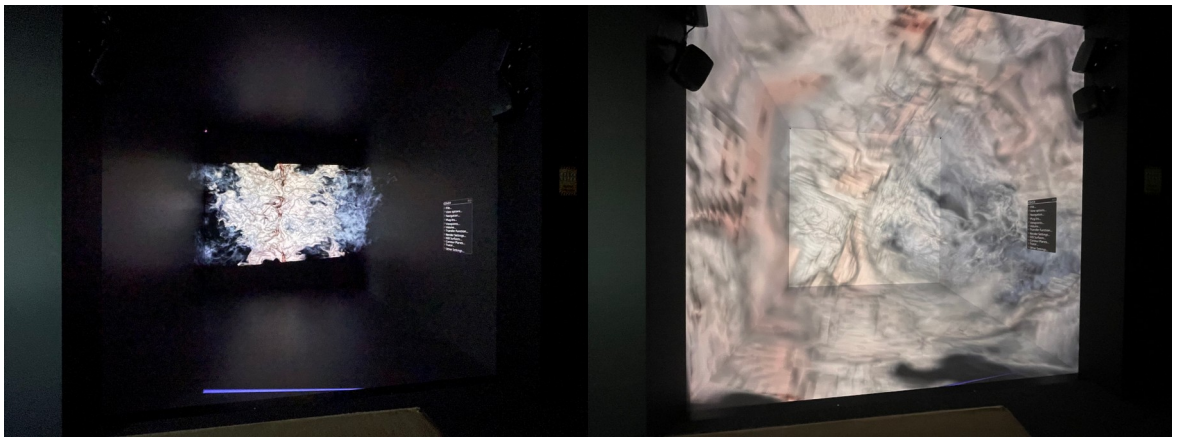
For both datasets, namely the molecular cloud and the LANL meteor impact, a deliberate selection of two viewpoints was made (Image 5.5 and Image 5.6). The location information (such as scale and position) of these 2 viewpoints are stored on GitHub [Wan23].

The first viewpoint presents the entirety of the observed object, rendered across at least three CAVE screens. The second one focuses into the volume and aim to see performance of the rendered image in details. In the second viewpoint, all five screens are blanketed with the rendered volume, affording a detailed assessment of performance under precise conditions. This dual approach facilitates a comprehensive evaluation of rendering performance across different scenarios.

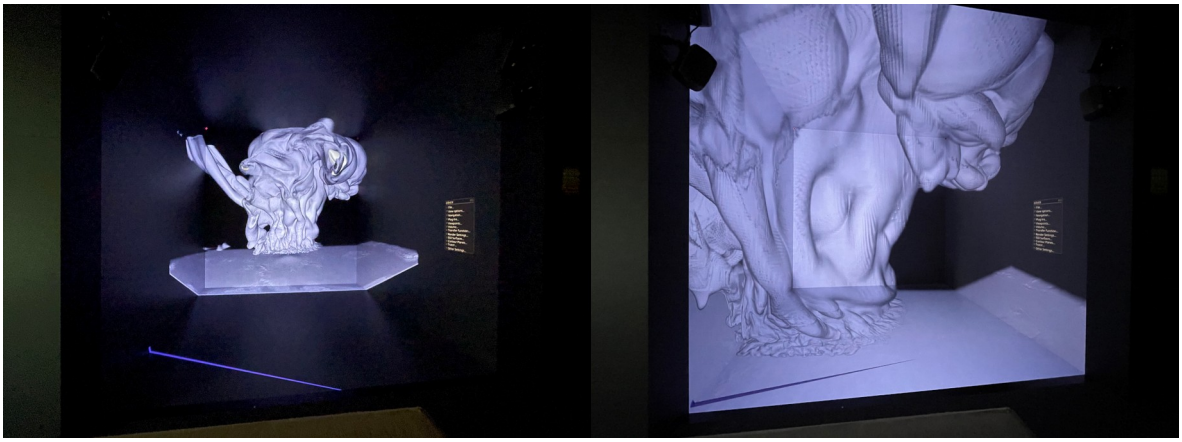


(a) *(b)* *(c)* *(d)*

Image 5.5: OpenCOVER desktop window screenshots (window size: 1180x1190) with stereo-rendering off for (a) molecular cloud viewpoint 1, (b) molecular cloud viewpoint 2, (c) LANL meteor impact viewpoint 1, (d) LANL meteor impact viewpoint 2.



(a) *(b)*



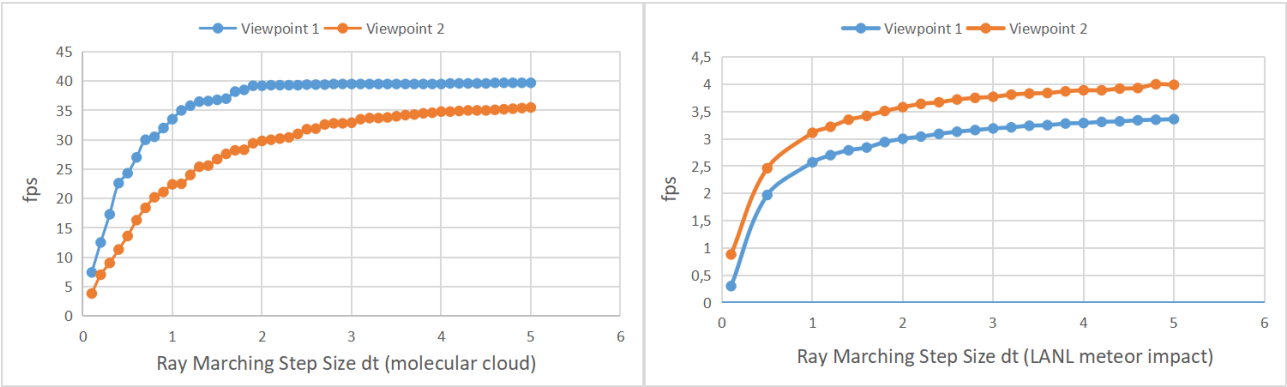
(c) *(d)*

Image 5.6: CAVE Photos with stereo-rendering off for (a) molecular cloud viewpoint 1, (b) molecular cloud viewpoint 2, (c) LANL meteor impact viewpoint 1, (d) LANL meteor impact viewpoint 2.

5.5.3 Results

To quantify the impact of adjusting ray marching step sizes, a meticulous analysis of frame per second (fps) variations was conducted. This investigation holds the viewpoints consistent, as established in Section 5.5.2. Notably, during the measurement process, it was observed that fps fluctuations remained well within a maximal deviation of 1.0, and no abrupt fps drops were detected.

For the molecular cloud dataset, a total of 50 fps values were meticulously measured across the range of dt=0.1 to dt=5.0. These values have been tabulated in Appendix Table 1. Similarly, for the LANL meteor impact dataset, 23 fps values were recorded spanning dt=0.1 to dt=5.0, and these figures have been tabulated in Appendix Table 2. The quantitative measurements from these tables have been translated into graphical representations (Image 5.7), thereby facilitating a more accessible analysis and discussion of the outcomes.



(a) (b)

Image 5.7: Graph translated from Ray Marching Step Size to FPS Table for both viewpoints (a) Molecular Cloud, (b) LANL Meteor Impact (t=46.112s).

In Image 5.7 (a), at viewpoint 1, we observe that the FPS ranges from 7.4 to 24.3 for a dt varying between 0.1 and 0.5. Within this range, the performance is only modestly acceptable. The FPS crosses the 25 threshold with ray marching step sizes over 0.5. For example at dt=1, the FPS achieves 35, denoting a comfortably interactive range for dt values of over 0.5. Shifting to viewpoint 2, the FPS exhibits a general decrement compared to the first viewpoint due to its deeper location within the volume. Here, a reasonable dt range of 0.5 to 2.0 maintains an acceptable performance bracket of 13.6 to 29.8.

In Image 5.7 (b), both viewpoints illustrate a flattening curve, with the FPS plateauing around 3 or 4. This suggests that even with significantly larger ray marching step sizes, performance enhancements remain marginal. Notably, the performance at viewpoint 2 surpasses that of

viewpoint 1, attributed to the rendering of fewer bricks in this configuration.

It's worth mentioning that the rendering performance for the molecular cloud dataset is notably superior compared to the meteor impact dataset. This discrepancy can be attributed to the latter dataset's size, which is roughly 3 to 4 times larger than the former. This discrepancy underscores a potential limitation of the ExaBrick Plugin when handling larger datasets, an aspect that could be addressed and refined in future endeavors.

5.5.4 Auto FPS Mode

The incorporation of an "auto fps" mode as the plugin's default setting aims to assure users a steady and delightful experience, an idea inspired by the volume plugin in COVISE [SWWL14]. This mode dynamically fine-tunes rendering parameters to uphold a predefined fps threshold. By employing the "Frame rate" slider within the "Other Settings" menu, users can establish their desired fps. This feature intuitively adapts rendering settings in line with scene intricacy, sidestepping unsettlingly low or abrupt fps drops that might compromise user engagement.

The effectiveness of the auto fps mode is evident in datasets like the molecular cloud, if the target fps is set reasonable. However, it encounters challenges when handling larger datasets like the meteor impact. To address this, the code could be enhanced to regulate the pace of ray marching step size changes and define available ranges for dt and target fps for future enhancements.

6 Discussion

In this section, results of the performance test (section 5.5.3) will be interpreted and critically assess the implemented solution for real-time rendering of large AMR data in VR environments. We identify the strengths, weaknesses, and limitations of the solution, discussing the implications of the findings.

6.1 Performance Assessment

According to the results in Image 5.7 and Table 1, the fps increases significantly while ray marching step increases in the dt range between 0.1 and 2.0 for both viewpoints and for both datasets. This means, the adjustments made to the ray marching step significantly improve rendering speed, allowing for smoother interactions when exploring the datasets. Rendering the dataset molecular cloud results a steady fps from 25 to 40 in the dt range of 0.5 to 2.0, and the fluctuation of fps remains

under 1.0, ensured the user a stable and comfortable fps exploring the AMR data.

However, while rendering the dataset meteor impact, the fps remains under 5.0 even for really big ray marching step sizes, which means users will have a hard time interacting with the data rendering this dataset. The interaction takes a relatively long time to respond.

Overall, with a reasonable adjusted ray marching step, our performance evaluation demonstrates that the ExaBrick Plugin excels in real-time rendering of some AMR data in VR environments, providing users with a seamless and immersive visualization experience.

6.2 Contributions

The research presented in this thesis makes several significant contributions to the field of real-time rendering of large AMR data.

Firstly, the developed ExaBrick Plugin successfully enables the real-time visualization and exploration of many complex AMR datasets, leveraging the VR visualization system of COVISE and interactive navigation techniques. The integration of ExaBrick as a COVISE plugin provides an efficient and scalable framework for rendering large-scale AMR data in VR environments. Additionally, the methods to adjust ray marching step

size guarantees a balanced trade-off between rendering speed and visual fidelity.

Overall, this work offers scientists and researchers a powerful tool for exploring, analyzing and understanding complex AMR datasets.

6.3 Limitations

While the implemented solution demonstrates promising results, it is important to acknowledge its limitations. One limitation is the performance impact when dealing with extremely large AMR datasets like meteor dataset, where further tests and optimization may be required to maintain real-time rendering capabilities. This plugin need to be used in VR environments like CAVE, there might be integrations done to enable the real-time visualization of large AMR data on smaller VR device like VR headsets for personal computers.

6.4 Future Directions

Also we need to point out the areas for future improvement.

First of all, since the ExaBrick Plugin still has limitations while rendering some larger datasets, this plugin can be further tested with a wider range

of large AMR datasets, to see if it works well with them. And one might discover what kind of datasets behave well using this plugin and what kind of datasets less well. There might be a solution to improve the performance with datasets like the dataset meteor impact.

To enhance the quality and user experience of visualizing AMR data with the ExaBrick Plugin, additional adjustments and improvements in coding can be explored, such as changing initial values for auto fps mode, using a speed function to let dt react quick, optimizing the transition path of the matrices, further exploring the interaction possibilities on VR menu, etc.

Future work may also involve investigating advanced rendering techniques to extend the plugin, considering scattering events and global illumination [ZWSM*22], for example. Additionally the performance may benefit from making use of the newest hardware, or incorporating machine learning algorithms to optimize the AMR data structure.

By addressing these areas, the solution will continue to offer researchers and scientists an efficient and immersive tool for exploring and analyzing complex scientific datasets in virtual reality environments.

Appendix

a. Tables

Table 1: Ray Marching Step Size dt vs fps for Viewpoint 1 and 2 of Rendered Dataset Molecular Cloud

molecular cloud	fps	
dt	Viewpoint 1	Viewpoint 2
0.1	7.4	3.8
0.2	12.5	7.0
0.3	17.3	9.0
0.4	22.6	11.3
0.5	24.3	13.6
0.6	27.0	16.3
0.7	30.0	18.4
0.8	30.5	20.2
0.9	32.0	21.1
1.0	33.5	22.4
1.1	35.0	22.5
1.2	35.8	24.0
1.3	36.5	25.4
1.4	36.6	25.6
1.5	36.8	26.7
1.6	37.0	27.6
1.7	38.2	28.2
1.8	38.5	28.3
1.9	39.2	29.4
2.0	39.2	29.8
2.1	39.3	30.0
2.2	39.3	30.2
2.3	39.3	30.4
2.4	39.3	31.0

2.5	39.4	31.8
2.6	39.4	31.9
2.7	39.4	32.6
2.8	39.5	32.8
2.9	39.5	32.8
3.0	39.5	32.9
3.1	39.5	33.5
3.2	39.5	33.7
3.3	39.5	33.7
3.4	39.5	33.8
3.5	39.5	34.0
3.6	39.5	34.2
3.7	39.5	34.3
3.8	39.5	34.5
3.9	39.5	34.6
4.0	39.5	34.8
4.1	39.6	34.8
4.2	39.6	34.9
4.3	39.6	35.0
4.4	39.6	35.0
4.5	39.6	35.0
4.6	39.7	35.1
4.7	39.7	35.2
4.8	39.7	35.3
4.9	39.7	35.4
5.0	39.7	35.5

Table 2: Ray Marching Step Size dt vs fps for Viewpoint 1 and 2 of Rendered Dataset LANL Meteor Impact (t=46.112s)

meteor impact dt	fps	
	Viewpoint 1	Viewpoint 2
0.1	0.30	0.88
0.5	1.97	2.46
1.0	2.57	3.11
1.2	2.70	3.22
1.4	2.79	3.35
1.6	2.84	3.42
1.8	2.94	3.51
2.0	3.00	3.58
2.2	3.04	3.64
2.4	3.09	3.67
2.6	3.13	3.72
2.8	3.16	3.75
3.0	3.19	3.77
3.2	3.21	3.81
3.4	3.24	3.83
3.6	3.25	3.84
3.8	3.28	3.87
4.0	3.29	3.89
4.2	3.31	3.89
4.4	3.32	3.92
4.6	3.34	3.93
4.8	3.35	4.00
5.0	3.36	3.99

b. List of Images

Image 5.2: ExaBrick Plugin rendering dataset LANL meteor impact using OpenCOVER in CAVE.

Image 5.3: ExaBrick Plugin rendering the center of dataset LANL meteor impact using OpenCOVER in CAVE, observing and interactively moving into the volume with navigation options.

Image 5.4: Ray Marching Step Size along the Ray through the Bricks, [WZU*20], Fig. 2.

Image 5.5: OpenCOVER desktop window screenshots (window size: 1180x1190) with stereo-rendering off for (a) molecular cloud viewpoint 1, (b) molecular cloud viewpoint 2, (c) LANL meteor impact viewpoint 1, (d) LANL meteor impact viewpoint 2.

Image 5.6: CAVE Photos with stereo-rendering off for (a) molecular cloud viewpoint 1, (b) molecular cloud viewpoint 2, (c) LANL meteor impact viewpoint 1, (d) LANL meteor impact viewpoint 2.

Image 5.7: Graph translated from Ray Marching Step Size to FPS Table for both viewpoints (a) Molecular Cloud, (b) LANL Meteor Impact (t=46.112s), generated with Microsoft Excel and LibreOffice Writer.

c. Reference

[ABA00] M. Aftosmis, M. Berger, and G. Adomavicius: A Parallel Multilevel Method for Adaptively Refined Cartesian Grids with Embedded Boundaries. Technical Report AIAA-00-0808, American Institute of Aeronautics and Astronautics, 2000. 38th Aerospace Sciences Meeting and Exhibit.

[ADS*23] W. Alexandre-Barff, H. Deleau, J. Sarton, F. Ledoux, L. Lucas: A GPU-based out-of-core architecture for interactive visualization of AMR time series data. 2023.

[Avi23] Avizo Basic User Guide. https://www.biotech.cornell.edu/sites/default/files/2020-06/Avizo%20Basic%20User%20Guide.pdf, August 2023.

[BC89] Marsha J. Berger, Philipp Colella: Local adaptive mesh refinement for shock hydrodynamics. 1989.

[BO84] M. J. Berger and J. Oliger: Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. Journal of Computational Physics, 1984.

[BWG11] Carsten Burstedde, Lucas C. Wilcox, Omar Ghattas: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. 2011.

[CAV23] CAVE RRZK University of Cologne. https://rrzk.uni-koeln.de/hpc-projekte/visualisierung/cave, August 2023

[CGL*00] P. Collela, D. Graves, T. Ligocki, D. Martin, D. Modinano, D. Serafini, B. Vans Traalen: Chombo Software Package for AMR Applications Design Document. 2000.

[CSD93] C. Cruz-Neira, D.J. Sandin, T.A. DeFanti: Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. 1993.

[Dee23] Deep Water Impact Asteroid Simulation Data, LANL. https://oceans11.lanl.gov/deepwaterimpact/, August 2023.

[Exa23] ExaBrick GitHub Page. https://github.com/owl-project/owlExaBrick#create-exa-file-and-run-viewer, August 2023.

[Fri08] Michael Friendly: Milestones in the history of thematic cartography, statistical graphics, and data visualizatio. October, 2008.

[GWC*08] M. Gittings, R. Weaver, M. Clover, T. Betlach, N. Byrne, R. Coker, E. Dendy, R.Hueckstaedt, K. New, W. R. Oakes, D. Ranta, and R. Stefan: The RAGE radiation-hydrodynamic code. Computational Science & Discovery, vol. 1, no. 1, p. 015005, 2008.

[GWN*16] P. Girichidis, S. Walch, T. Naab et al.: The SILCC (SImulating the LifeCycle of molecular Clouds) project - II. Dynamical evolution of the supernova-driven ISM and the launching of outflows. March 2016.

[Hor18] Ben Horan et al.: Feeling your way around a CAVE-like reconfigurable VR system. 2018.

[KHYD22] Casey R. Koger, Sohail S. Hassan, Jie Yuan, Yichen Ding: Virtual Reality for Interactive Medical Analysis. 2022.

[KK19] Kalarat and Koomhin: Real-Time Volume Rendering Interaction in Virtual Reality. 2019.

[KSH03] Ralf Kaehler, Mark Simon, Hans-Christian Hege: Interactive Volume Rendering of Large Sparse Data Sets Using Adaptive Mesh Refinement Hierarchies. September 2003.

[KWAH06] Ralf Kaehler, John Wise, Tom Abel, Hans-Christian Hege: GPU-Assisted Raycasting for Cosmological Adaptive Mesh Refinement Simulations. 2006.

[Mas19] Betsy Mason: Why scientists need to be better at data visualization. Knowable Magazine, doi:10.1146/knowable-110919-1. November 12, 2019.

[MWUP22] Nate Morrical, Ingo Wald, Will Usher, SCI Institute, University of Utah, Valerio Pascucci, NVIDIA: Accelerating Unstructured Mesh Point Location with RT Cores. 2022.

[Ope23] OpenSceneGraph Project Website. http://www.openscenegraph.com/, August 2023.

[Par10] Steven G. Parker et al.: OptiX: A General Purpose Ray Tracing Engine. August 2010.

[Par23] ParaView Documentation. https://docs.paraview.org/en/latest/references.html, August 2023.

[PBD*10] Steven G. Parker, James Bigler, Andreas Dietrich et al.: OptiX: A general purpose ray tracing engine. July 2010.

[PST*16] Patchett J. M., Samsel F. J., Tsai K. C., Gisler G. R., Rogers D. H., Abram G. D., Turton T. L.: Visualization and Analysis of Threats from Asteroid Ocean Impacts. Tech. rep., Los Alamos National Laboratory, 2016.

[Ran95] D. Rantzau et.al.: Collaborative and Interactive Visualization in a Distributed High Performance Software Environment. Proceedings of the International Workshop on High Performance Computing for Graphics and Visualization, Swansea, Wales, 1995.

[RFL*98] D. Rantzau, K. Frank, U. Lang, D. Rainer, U. Woessner: COVISE in the CUBE: An Environment for Analyzing Large and Complex Simulation Data. Proceedings of the IPTW 1998.

[SA97] Mark Segal, Kurt Akeley: The OpenGL Graphics System: A Specication (Version 1.1). 1997.

[SC03] W.R. Sherman, A.B. Craig: Understanding Virtual Reality: Interface, Application, and Design. Morgan Kaufmann. USA, pp. 4–16, 2003.

[SCRL20] Sarton J., Courilleau N., Remion Y., Lucas L.: Interactive visualization and on-demand processing of large volume data: A fully GPU-based out-of-core approach. 2020.

[SIL23] SILCC data website, http://silcc.mpa-garching.mpg.de/index.php, August 2023.

[Sil90] Silicon Graphics, Inc.: Graphics Library Programming Guide. Document Version 2.0, 1990.

[Sta05] Detlev Stalling et al.: Amira - a Highly Interactive System for Visual Data Analysis. 2005.

[SWG*17] D. Seifried, S. Walch, P. Girichidis, T. Naab, R. Wünsch, R. S. Klessen, S. C. O. Glover, T. Peters, P. Clark: SILCC-Zoom: the dynamic and chemical evolution of molecular clouds. December 2017.

[SWWL14] Jürgen Schulze-Döbold, Uwe Wössner, Steffen P. Walz, Ulrich Lang: Volume Rendering in a Virtual Environment. May 2014.

[SZD*23] J. Sarton, S. Zellmann, S. Demirci, U. Güdükbay, W. Alexandre-Barff,L. Lucas, J.M. Dischler, S. Wesner, I. Wald: State-of-the-art in Large-Scale Volume VisualizationBeyond Structured Data. June 2023.

[Vis23] VisIt. Main page - visitusers.org. http://www.visitusers.org, August 2023.

[Use23] User’s Guide of COVISE Online Documentation. https://fs.hlrs.de/projects/covise/doc/html/usersguide/index.html, August 2023.

[Wan23] Zhaoyang Wang, Github fork for this thesis with related code and file. https://github.com/zywang3/covise/tree/zhaoyang, August 2023.

[WGN*15] S. Walch, P. Girichidis, T. Naab, A. Gatto et al.: The SILCC (SImulating the LifeCycle of molecular Clouds) project - I. Chemical evolution of the supernova-driven ISM. November 2015.

[WBUK17] I. Wald, C. Brownlee, W. Usher, and A. Knoll: CPU Volume Rendering of Adaptive Mesh Refinement Data. In SIGGRAPH Asia 2017 Symposium on Visualization, 2017.

[WMPT65] E. R. Woodcock, T. Murphy, P. J. Hemming s, and T. C. Longworth: Techniques used in the GEM code for Monte Carlo neutronics calculation in reactors and other systems of complex geometry. Tech. rep., Argonne National Laboratory, 1965.

[WWW*19] F. Wang, I. Wald, Q. Wu, W. Usher, and C. R. Johnson: CPU Isosurface Ray Tracing of Adaptive Mesh Refinement Data. IEEE Transactions on Visualization and Computer Graphics, 2019.

[WZM12] Gunther H. Weber, Hank Childs, Jeremy S. Meredith: Efficient Parallel Extraction of Crack-free Isosurfaces from Adaptive Mesh Refinement (AMR) Data. October 2012.

[WZU*20] Ingo Wald, Stefan Zellmann, Will Usher, Nate Morrical, Ulrich Lang, and Valerio Pascucci: Ray Tracing Structured AMR Data Using ExaBricks. October 2020.

[Zel14] Stefan Zellmann: Interactive High Performance Volume Rendering. July 2014.

[ZSM*22] S. Zellmann, D. Seifried, N. Morrical, I. Wald et al.: Point containment queries on ray tracing cores for AMR flow visualization. Computing in Science Engineering, April 2022.

[ZWL20] Stefan Zellmann, Daniel Wickeroth, Ulrich Lang: Visionaray: A Cross-Platform Ray Tracing Template Library. September 2020.

[ZWSH*22] S. Zellmann, I. Wald, A. Sahistan, M. Hellmann, W. Usher: Design and Evaluation of a GPU Streaming Framework for Visualizing Time-Varying AMR Data. June 2022.

[ZWSM*22] Stefan Zellmann, Qi Wu, Alper Sahistan, Kwan-Liu Ma, Ingo Wald: Beyond ExaBricks: GPU Volume Path Tracing of AMR Data. November 2022.

Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht.

Ort, Datum Unterschrift

Köln, 21.08.2023 _____