Advanced Real-Time Volume Graphics

Stefan Zellmann
University of Cologne
Cologne, Germany
zellmann@uni-koeln.de

Alper Sahistan
University of Utah
Salt Lake City, USA
asahistan@sci.utah.edu

Ingo Wald
NVIDIA
Salt Lake City, USA
iwald@nvidia.com

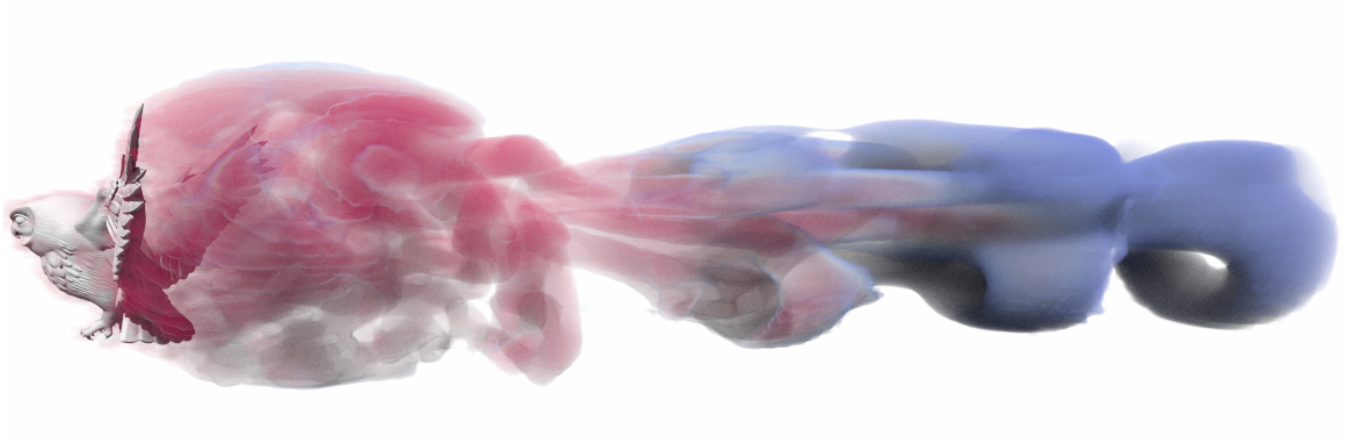


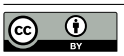
Figure 1: Interactive visualization of a CFD simulation with a modern sci-vis rendering pipeline. Techniques from volumetric path tracing, which are used here, are more common in visual effects, but also have several advantages for sci-vis that are easily overlooked. In this course, we teach the basics of such a pipeline and its pros and cons when used in sci-vis.

Abstract

In this course we teach how to write interactive volume renderers for scientific visualization (sci-vis) that run on modern GPUs. We establish a framework adopting techniques from volumetric path tracing as used for visual effects rendering that also have advantages for sci-vis. We teach those concepts through a set of small, self-contained sample applications provided as additional material. Starting from a simple "hello world" app, we gradually extend the samples with more advanced features used by modern, ray tracing-based volume renderers. Those features include rendering of non-trivial volumetric primitives, handling of large data, adaptively sampling and skipping over empty or homogeneous space, and the use of ray tracing hardware and multi-GPU systems. Finally, we also show how to integrate a renderer like this into sci-vis applications like ParaView.

CCS Concepts

• **Computing methodologies** → **Rendering**; **Ray tracing**; • **Human-centered computing** → **Scientific visualization**.



SIGGRAPH Courses '26, Los Angeles, CA, USA

ACM ISBN 979-8-4007-2543-2/2026/07
https://doi.org/10.1145/3799820.3812500

Keywords

Volume Rendering, Scientific Visualization, GPGPU, Ray Tracing, Volume Path Tracing, Adaptive Volume Sampling, RT Cores

ACM Reference Format:
Stefan Zellmann, Alper Sahistan, and Ingo Wald. 2026. Advanced Real-Time Volume Graphics. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Courses (SIGGRAPH Courses '26), July 19–23, 2026, Los Angeles, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3799820.3812500

1 Introduction

Real-time volume graphics is one of the most matured methods in scientific visualization. About 25 years ago, when this topic became so popular, the most urgent challenge was how to use the limited programming models of GPUs from these times to efficiently implement volume rendering algorithms. Fastforward more than 20 years from the last SIGGRAPH course on this topic [Engel et al. 2004], the vast majority of volume renderers these days uses ray tracing models to accumulate samples, either driven by parallel C++ programs or by GPGPU kernels.

All the clever, yet arcane, methods to force the GPU hardware into sampling and compositing the volume rendering equation through its rasterization pipeline appear to be long obsolete by now; yet new challenges arose since the field has matured that can be largely attributed to the volumes' sizes. Different field types than voxel grids have become more important especially in times of Exascale. As the sample count is highly correlated with the volume size, the importance of space skipping techniques has increased dramatically. With the new compute power of modern GPUs at

hand the use of Monte Carlo sampling for the volume rendering integral has become an option even for real-time graphics; and finally, the sheer sizes of the volumes demand a new look at data parallel rendering (and sample compositing) beyond such volumes that are easily decomposed into simple blocks.

The course develops a ray tracing framework for scientific volumes. It starts by introducing simple voxel grid integrators sampling at equidistant steps. The course proceeds by showing how to support different volume types within this same framework. A paradigm of segment traversal and sample accumulation along segments is established to show how to implement different ways of space skipping; this is complemented by adding Monte Carlo integrators to the framework that have a special relationship to transfer functions and space skipping techniques. We finally shed light on how all that ties into data parallel rendering, and provide cues how to integrate a framework like this into production systems like ParaView [Ahrens et al. 2005].

2 Prerequisites

Our course will teach the basics of writing a modern sci-vis volume renderer from first principles. For that we provide a base framework written in portable C++. Though not strictly intended to be hands-on, the course format will allow the participants to follow along the code samples, and to build and run them on their own hardware. Later in the course, there are code samples that make use of NVIDIA RTX ray tracing hardware, exclusively.

Prerequisites to our course are:

- familiarity with the programming languages C++ and CUDA,
- to follow along and interactively build and run the code samples presented, a laptop with Windows 11, macOS, or Linux, a C++ compiler and CMake installation, and a code editor are required.

To follow along with the samples using ray tracing cores that are discussed later in the course, the following is required:

- a laptop with an NVIDIA GPU that is compatible with OptiX [NVIDIA Corporation 2026a],
- and an installation of the OptiX Wrapper Library (OWL) [NVIDIA Corporation 2026b].

The source code for our framework is available on GitHub.com: https://github.com/dvr-course/dvr-course-cpp. Attendees planning to follow along are expected to download, build, and run this project on their laptop. By providing the source code online, users can also interact with the samples in a self-paced way when working through the course material offline. The framework supports an interactive mode, where all the sample apps have an interactive viewport and a transfer function editor, and a non-interactive mode with command line execution where the samples produces an image as output. In non-interactive mode the framework has fewer external dependencies, making it simpler for users attending the course in person to download and build the project. In interactive mode, dependencies such as SDL3 or ImGui are downloaded on-the-fly and built by CMake when configuring the source code project.

3 Course Overview

The course material is presented as a combination of slides and sample code. While the slides will lay the theoretical foundation for the topics we convey during the course, the sample code is used for deepening the concepts learned. The sample code is organized into a set of apps, a selection of which are shown in Figure 2.

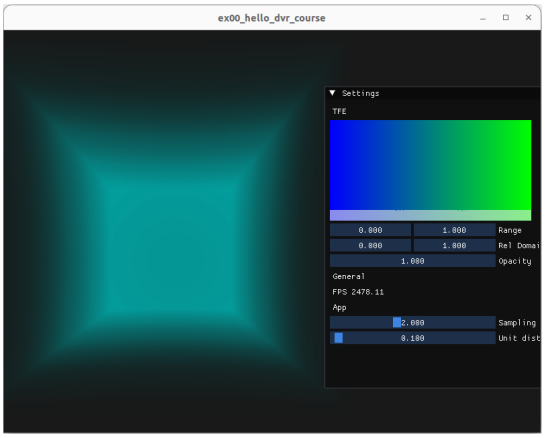
"Hello DVR Course" sample. the first course segment and sample app (see Figure 2a) will introduce the basics of writing a ray tracing-based volume renderer. The sample uses absorption plus emission rendering via ray marching as done by traditional sci-vis volume renderers. This is the baseline for modern GPGPU-based renderers implementing what is considered the status quo in sci-vis. The course segment will discuss the pros and cons of that approach.

Voxel rendering sample. While the previous code sample used a simple homogeneous volume, we now extend the sample app to use voxel rendering via NanoVDB [Museth 2021]. We also establish an abstraction to sample from volumes in a general way that can later be extended to support different cell types. The interactive sample app is shown in Figure 2b.

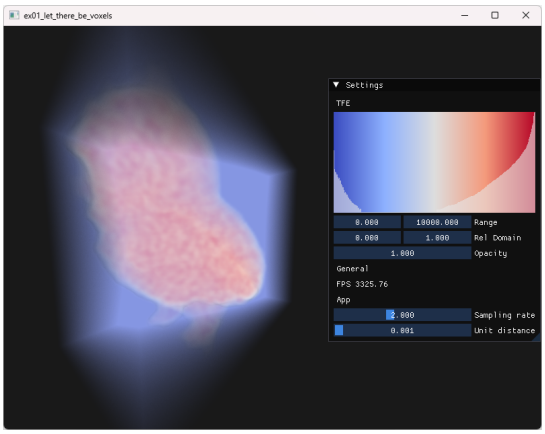
Woodcock tracking sample. We now gradually transition to a framework more commonly used in visual effects (VFX) that uses Monte Carlo algorithms for collision tracking. The resulting free-flight distances can be used even when computing absorption and emission only, as is common in sci-vis. Tracking gives us for each collision what is essentially a color and depth fragment with a probability density attached to it. That greatly simplifies integration of volume rendering with surface renderers. Rendering volumes has traditionally been done after all surfaces were rendered, and the ability to compute light transport interaction between volumes and surfaces were quite limited in sci-vis. The presence of RGBA transfer functions in sci-vis has some practical implications determining how color and depth are computed that limit the way that transmission and distance estimates are computed; we also discuss these during the theoretical part of this course segment.

Multi-volume rendering sample. Switching to a tracking framework also makes rendering multiple overlapping volumes, or single volumes with more than one channel, more intuitive. While a traditional ray marcher has to explicitly detect segments along the viewing ray that the volumes overlap, and then also explicitly integrate over exactly those volumes, using free-flight distances simplifies this process substantially. While naïvely blending overlapping volumes in a traditional ray marcher leads to inconsistent normalization and color artifacts, free-flight distances provide a physically correct alternative that naturally handles combined extinctions without heuristic biasing.

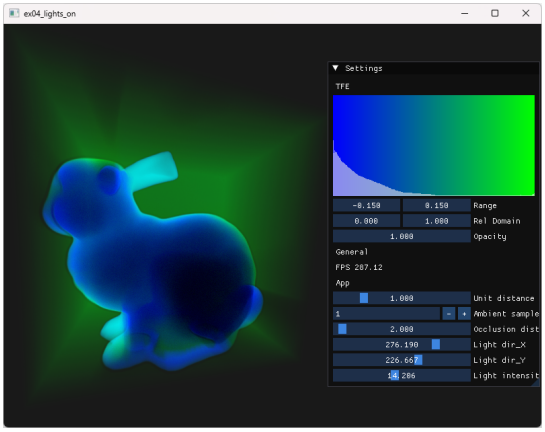
Advanced lighting sample. We discuss adding more advanced lighting effects from directional and omni-directional light sources, resulting in a single-scattering, direct lighting-based renderer shown in Figure 2c. More advanced global illumination and scattering effects are often counterproductive in sci-vis because they complicate interpreting colors in the final output that are assigned through color mapping, while shadows provide additional depth cues and help with interactive exploration. During this segment of the course we will also discuss different collision tracking algorithms, their pros and cons, and their applicability to sci-vis.



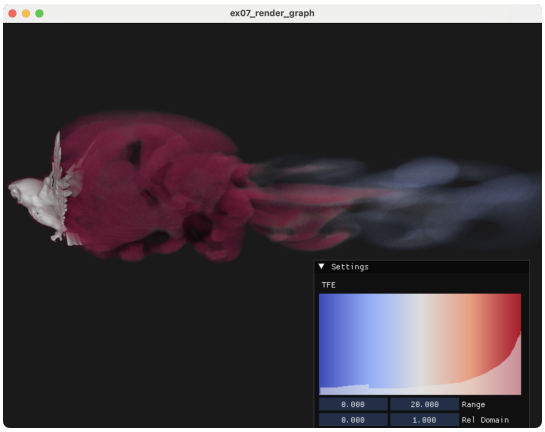
(a) "Hello DVR Course" sample, uses sci-vis ray marching to sample a homogeneous volume with an RGBA transfer function.



(b) Voxel rendering sample, adds NanoVDB volumes as a data source for voxel grids.



(c) Direct lighting sample, extends the pipeline to support single scattering, and is used for discussion of different tracking algorithms.



(d) Render graph sample, combines the techniques discussed in this course and adds them into a more versatile renderer supporting complex geometry.

Figure 2: Sample apps. The course centers around a framework consisting of ten sample apps coming with source code that we gradually extend with more advanced volume rendering features. A subset of those apps is shown in this figure.

Tetrahedra sampling sample. In this course segment we discuss more complex volume primitives than the voxel cells we used so far. By using the volume sampling abstraction the framework remains largely unchanged, only the sampling routine itself needs to be adapted. Here we also make use of hardware ray tracing for sample location. The accompanying sample app is based on OptiX [NVIDIA Corporation 2026a] and OWL [NVIDIA Corporation 2026b].

Space skipping sample. Here we discuss the use of space skipping data structures in a ray tracing-based volume renderer. Ray marching renderers often use empty space skipping only, but switching to free-flight distances naturally gives rise to space skipping structures with majorant densities that allow for fine-grained adaptive sampling. Other than in VFX, the presence of RGBA transfer functions requires that the space skipping data structure can be rebuilt or updated interactively. We discuss the technical choices and alternatives during this course segment.

Render graph sample. In this course segment we teach how to integrate the sample we developed into a wider context provided by a renderer handling not just volumes, but also surfaces that are organized in a render graph. Here we also discuss the technical choices that come with using (NVIDIA) ray tracing hardware. Since hardware acceleration using bounding volume hierarchies is only available at one level, a choice is required at which stages of the pipeline acceleration is used. The stages that can benefit from hardware acceleration are: render graph traversal, acceleration structure traversal, and sample location with search data structures. We discuss the alternatives and potential implications on performance. The sample app for this segment is shown in Figure 2d.

Data-parallel rendering sample. In this course segment we discuss how to extend the framework to support data parallelism

targeting multi-GPU systems and PC clusters. These systems can be quite complex and the design decisions come with different trade-offs. Multi-GPU and multi-node rendering is an advanced topic and the course segment is intended to give an overview rather than an in-depth technical discussion with concrete recipes for implementation.

ANARI sample. We will finally discuss how to integrate a renderer such as the one developed during the course into sci-vis apps like ParaView. This can be done by using the Khronos standard ANARI [Stone et al. 2022]. As this is an advanced topic, we do not present a full-fledge rendering system, but only show an example of adding an ANARI layer on top of a minimal volume renderer (simple lighting, single volume only, no render graph) and how to load it into ParaView [Ahrens et al. 2005].

4 Learning Outcomes

Volume rendering has been approached differently by the sci-vis and the visual effects communities. While this was reasonable given the different constraints on performance, data size, and visual fidelity of the output, in the meantime GPUs have become more capable, GPGPU APIs have matured, and hardware ray tracing has become available and is accessible through specialized APIs. From the sci-vis perspective it is reasonable to adopt some of the features that are used in VFX, while at the same time certain factors lead to different choices how to build acceleration structures, how data is stored and manipulated, and how to interpret physical quantities given that some of the input, such as the RGBA transfer function, is not derived from physics.

With this course we want to bridge those two communities. Sci-vis practitioners will learn how to create a volume renderer from scratch based on Monte Carlo rendering techniques that can be integrated into a production-grade visualization system. Practitioners from VFX will learn about the different performance demands, the size of the data, the requirement to interactively adjust transfer functions, and how that leads to different design decisions than the ones common in their field. This includes the use of data-parallel rendering often used in sci-vis, but less so in VFX.

5 Conclusion

We presented the motivation, prerequisites for participation, organization, and intended learning outcomes of our course on advanced real-time volume graphics. The course is centered around a set of CUDA/C++ sample apps, each of which incrementally adds new features, culminating in a renderer that could be integrated into a production visualization system like ParaView. The primary goal of the course is to bridge communities by addressing practitioners from sci-vis and VFX.

Acknowledgments

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant no. 456842964. Part of this work was also funded by the German Federal Ministry of Research, Technology and Space under the funding code 01LK2204A. The responsibility for the content of this publication lies with the authors.

References

James P. Ahrens, Berk Geveci, and C. Charles Law. 2005. ParaView: An End-User Tool for Large Data Visualization. In *The Visualization Handbook*, Charles D. Hansen and Christopher R. Johnson (Eds.). Butterworth-Heinemann, Burlington, 717–731. doi:10.1016/B978-012387582-2/50038-1

Klaus Engel, Markus Hadwiger, Joe M. Kniss, Aaron E. Lefohn, Christof Rezk Salama, and Daniel Weiskopf. 2004. Real-time volume graphics. In *ACM SIGGRAPH 2004 Course Notes* (Los Angeles, CA) *(SIGGRAPH '04)*. Association for Computing Machinery, New York, NY, USA, 29–es. doi:10.1145/1103900.1103929

Ken Museth. 2021. NanoVDB: A GPU-Friendly and Portable VDB Data Structure For Real-Time Rendering And Simulation. In *ACM SIGGRAPH 2021 Talks* (Virtual Event, USA) *(SIGGRAPH '21)*. Association for Computing Machinery, New York, NY, USA, Article 1, 2 pages. doi:10.1145/3450623.3464653

NVIDIA Corporation. 2026a. NVIDIA OptiX Ray Tracing Engine. https://developer.nvidia.com/optix.

NVIDIA Corporation. 2026b. OWL – A Productivity Library for OptiX. https://github.com/NVIDIA/OWL.

John E Stone, Kevin Griffin, Jefferson Amstutz, David E DeMarle, William R Sherman, and Johannes Günther. 2022. ANARI: A 3-D Rendering API Standard. *Computing in Science & Engineering* 24, 02 (2022). doi:10.1109/MCSE.2022.3163151