

Programmable Interactive Visualization of a Core-Collapse Supernova Simulation

Roland Haas^a, Philipp Mösta^b, Mahendra Roopa^c, Alexander Kuhn^c, Marc Nienhaus^c

^aNational Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 1205 W. Clark St., Urbana, IL 61801, U.S.

^bDepartment of Astronomy, University of California at Berkeley, 501 Campbell Hall #3411, Berkeley, CA 94720

^cNVIDIA - Advanced Rendering Center, Germany

Abstract

Core-collapse supernovae (CCSN) are the birth places of neutron stars and black holes. They liberate the ashes of stellar evolution, seeding the interstellar gas with the elements from which planets form and life is made. Despite their importance for much of astrophysics, our understanding of the supernova explosion mechanism, and its dependence on progenitor star properties, is still incomplete. In addition, CCSN simulations generate terabytes of output posing additional challenges regarding the effective analysis of the simulated physical processes. To address those challenges, we present NVIDIA IndeX as a scalable framework to visualize the simulation output. The framework features a streaming-based architecture to interactively explore simulation results in distributed multi-GPU environments. We demonstrate how to customize specialized sampling programs for volume and surface rendering to cover specific analysis questions of CCSN simulations. This provides an extensive level of control over the visualization while efficiently using available resources to achieve high-levels of performance and visual accuracy.



Specific Entropy: Close-up view of the specific entropy simulated in model s15WH07 [1] rendered with a gradient-enhanced volume kernel: Low entropy values are shown in red, high entropy values in blue, colors and opacity are amplified by the gradient magnitude during the core collapse; the protoneutron star is highlighted at the center (cf. movie part VI).

1. Introduction

Core-collapse supernovae are the deaths of massive stars at the end of their lifetime. Once nuclear fusion stops in the center of the star its iron core cannot support itself and collapses under its own gravity until a neutron star is formed. The formation of the neutron star, which is also referred to as core bounce, sends an initial shockwave outwards into the remainder of the collapsing star. This shockwave however quickly stalls at a radius of $\approx 150 - 200$ km as it runs into the still infalling outer layers of the core and star. All of this happens within the final second of the star's life. A central and still incompletely understood problem in core-collapse supernova research is how

exactly and in which stars the shockwave gets re-energized such that an explosion occurs. Supercomputer simulations attempt to shed light on how this exactly happens, starting at core bounce.

In our research we simulate both, collapse and the time after bounce, however for analyzing we typically focus after core bounce. After core bounce, the newly formed neutron star sits at the center and the stalled shockwave surrounds it. The neutron star is very hot and cools down rapidly via the emission of vast quantities of neutrinos. Ninety-nine percent of the energy liberated in the gravitational collapse is emitted in neutrinos. Some of these neutrinos are absorbed behind the stalled shockfront and heat up the material there. This causes higher

and higher temperature bubbles to form. Once those are large enough, they start driving the shockfront outwards and launch a supernova explosion. This is what is depicted in the rendering of the simulation which picks up a short time after core bounce and continues until after the explosion takes off.

2. Distributed Scientific Data Management & Visualization

Interactive visualization facilitates the exploration of simulation properties and monitoring of the quality of the simulation output. It provides effective means to study the evolution of features (such as explosive plumes and shockfronts) over time, to compare different simulation runs, and to examine the impact of different models or parameterizations. However, astrophysics simulations, in particular the outlined core-collapse simulations, generate terabytes of n-dimensional volumetric data which often reduces the availability of advanced, high-performance visualization techniques. At the same time, visual exploration techniques that operate directly on the simulated output offer additional insights into the data and can be used to investigate the properties of features which might not be known in advance. Consequently, near-realtime, intuitive interaction with the data is key to more effective evaluation of the simulation results.

For interactive visual exploration, we make use of the NVIDIA IndeX rendering software¹ that leverages GPU clusters for scalable, real-time visualization. NVIDIA IndeX overcomes the limitations of a single computer system, e.g., limited GPU memory, by utilizing the distributed resources of multi-GPU environments (such as multi-GPU single nodes or GPU clusters). By allocating the appropriate GPU cluster resources, larger dataset sizes can be efficiently processed on GPUs while ensuring interactive framerates. Effective parallelization enables more costly visualization operations (e.g. single-scattering 4) with higher visual quality. In addition, it provides a simple-to-use mechanism to support compute-intensive visualization strategies for exploration (see Sec. 2.1) in order to gain deeper and more comprehensive insights into the produced data. The efficient use of a multi-GPU cluster system also enables the analysis of data at its original resolution; no additional post-processing steps for downsizing, data abstraction, or reduction techniques (e.g., level-of-detail) are required. This ensures high visual fidelity of the rendering and reduces the risk of losing visual features or of introducing artifacts due to additional processing.

2.1. Programmable Scientific Data Visualizations

To increase the level of control over the visualization, NVIDIA IndeX provides an extensible programming interface to customize appearance of the source data visualization: The NVIDIA IndeX Accelerated Compute (XAC) interface provides access to two types of programs: *surface* and *volume sampling programs* (Figure 2.1, video part II). For a volume

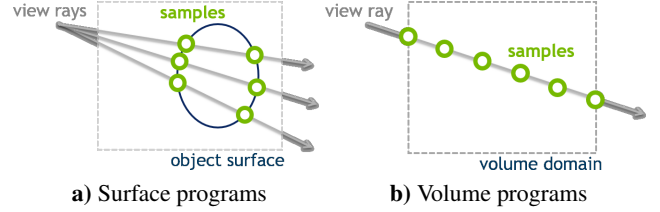


Figure 1: **Sampling Program Types:** Volume and surface sampling programs can be modified and executed at runtime. Both types are executed during rendering at different events (green points): The surface sampling programs are executed when view rays intersect the surface, while volume sampling programs are executed at equidistant sample locations along the view rays.

sampling program, the XAC interface provides access to the volume’s data at or around sampling positions, allows the use of transfer functions (e.g., for mapping volume data to a colors), and writing the colored results back into the rendering procedure for display. The XAC programs are written in CUDA and are directly compiled and injected into the inner loops of the volume or surface raycasters at runtime. This enables researchers to take advantage of scalable rendering and gives a highly flexible way to specify the final data visualization, based on the elements in the scene. Several example programs are presented in Figure 4 and in the movie starting from part II. Typically, each program contain less than 200 lines of code. The sampling programs shown in the video are attached to the submission.

2.2. Distributed, Interactive Rendering

To achieve interactive performance in a multi-GPU environment, NVIDIA IndeX efficiently parallelizes each step during the rendering workflow. First, the volumes are spatially split into several subvolumes that are assigned to individual GPUs and which fit into their memory. Second, each GPU performs a raycasting in a front-to-back composing manner for each sub-volume and stores its result in a separate framebuffer segment. Finally, the results are transferred and are assembled on a single GPU, which composites all rendered buffer segments into the final frame. The final rendered frame is then sent to a browser-based web client² shown in Figure 4. The client contains basic user interface elements to interactively set primary rendering parameters (such as camera, transfer function, volume filter quality, and time step parameters). It contains an editor for the CUDA sampling program code with some preset-programs, an generic user parameter slider (its value can be used in the program without recompilation), and sends the edited source code back to the server-side’s cluster. Each of the GPUs then compiles and injects the sent program into its raycasting procedure for instant data visualization. The use of the editor is highlighted in the video starting from part II. We provide default transfer functions and colormap presets derived from the Colorbrewer project [3].

¹NVIDIA IndeX is available as SDK or Paraview Plugin. It is free for the scientific community and for non-commercial use.

²NVIDIA IndeX ships with a client-server application with integrated video-streaming functionality.

3. Dataset and Computational Resources

The presented 3D simulations are generated using open-source 3D general-relativistic multi-group radiation-hydrodynamics CCSN code [1, 2]. The results shown in the figures and the movie represent the model s15WH07 with a final simulation time of 597 ms after core bounce and a final maximum shock radius of 867 km [1]. The resulting data represents a subregion of the simulated volume that includes the shockfront at the last time step. It is stored on a regular equidistant grid with a spatial resolution of 3.96 km, using 633^3 grid points to cover the domain. For one simulation variable the resulting volume data in single precision has a size of about 1 GB per time step with a total number of 1077 time steps. The total memory requirement for the entire sequence in single precision, is about 1.1 TB.

For scalable scientific data visualization we ran NVIDIA IndeX on several multi-GPU systems: For the images, we use a NVIDIA DGX-1 system with eight P100 GPU with 16GB memory each, resulting in a total GPU memory capacity of 128 GB (movie part I to IV). In addition we used NVIDIA IndeX on a GPU cluster system with 8 DGX-1 instances (64 GPUs), mounting a total capacity of 1024 GB GPU memory. The visualization of the entire time-varying CCSN ran at approx. 20 frames per second (movie part V). Finally, we explored the NVIDIA DGX-2 with 16 V100 GPUs, each with 32 GB memory. The 512 GB of memory (in total) were used to visualize half of the core-collapse dataset with at least 15 frames per second.

4. Results and Scientific Value

Currently, the key question in core-collapse supernova theory is to determine the details of the explosion mechanism using 3D simulations. Multiple groups worldwide have performed detailed full 3D simulations of explosions with selected progenitor star initial conditions. One of the most interesting aspects in core-collapse supernova research is which conditions and physical effects create an explosion and how these conditions are at play in different stars. Two of the main effects are the turbulence caused by neutrinos depositing heat behind the shockfront and large-scale deformations of the shockfront itself, referred to as the standing accretion shock instability (SASI). The former causes the formation of hot gas bubbles that start driving the supernova explosion, like bubbles pushing to the surface in boiling water. The latter manifests itself in a large-scale sloshing motion of the entire shockfront. To which extent either of the effects is important in individual stars, or whether an interplay of the two is at work depends on the conditions and the structure of the collapsing star and the details of the neutrino transport.

In this case, 3D visualization is used to visually distinguish between these two different mechanisms. For this to be efficient, it is crucial that we can visualize the 3D datasets from the simulations quickly and intuitively. In Figure 4 we show two different kernels that emphasize different aspects of these phenomena: Figures a) and b) use the local entropy gradient to

alter color and opacity values to make the shockfront structure more transparent and keep the core and its location visible. In the movie part V, the kernel also shows where and when locally strong changes (bright blue regions) occur during the progression of the shockfront. Part II in the movie demonstrates how to combine volume and surface sample programs to investigate the spatial distribution of selected entropy regions, their temporal onset, and their interplay through the progression of the simulation.

In comparison, a typical rendering software on a single-node takes about 10 minutes per frame making the process of testing different color settings, dataset limits, viewing angles and camera positions, and exploring the dataset across the entire simulation time a tedious and time-intensive task. The ability to rapidly explore different timesteps as well as visualization configurations speeds up the visual exploration of such datasets significantly. The movie exemplifies several visual configurations (including isosurface raycasting and edge-enhancement in part II, non-local lighting in part IV, and gradient-based shading in part V and VI). Note that parts I to IV are taken from a single interactive session on a DGX-1 system.

Acknowledgments

Simulations were carried out on NSF / NCSA Blue Waters (PRAC ACI-1440083, OCI-0725070 and ACI-1238993), on Edison at the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, and on the Texas Advanced Computing Center Stampede and Stampede2 clusters under NSF XSEDE allocation TG-PHY100033. RH is supported by Blue Waters, the State of Illinois and NSF-1550514. PM acknowledges support by NASA through Einstein Fellowship grant PF5-160140. Simulation data was kindly provided by Christian Ott.

References

- [1] C. D. Ott, L. F. Roberts, A. Schneider, J. M. Fedrow, R. Haas, E. Schnetter, The progenitor dependence of three-dimensional core-collapse supernovae, arXiv preprint arXiv:1712.01304. 1, 3
- [2] L. F. Roberts, C. D. Ott, R. Haas, E. P. O'Connor, P. Diener, E. Schnetter, General-relativistic three-dimensional multi-group neutrino radiation-hydrodynamics simulations of core-collapse supernovae, The Astrophysical Journal 831 (1) (2016) 98. 3
- [3] M. Harrower, C. A. Brewer, ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps, The Cartographic Journal 40 (1) (2003) 27–37. doi:10.1179/000870403235002042. URL <http://dx.doi.org/10.1179/000870403235002042> 2

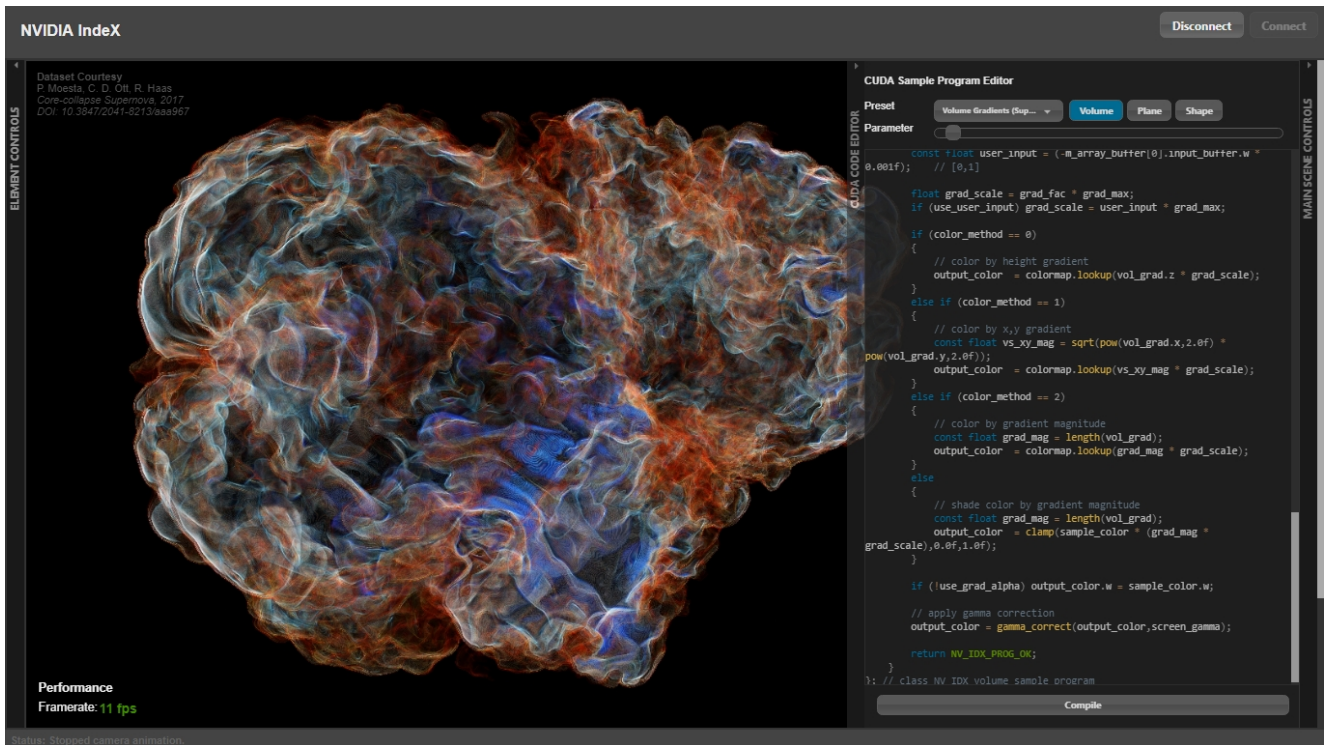
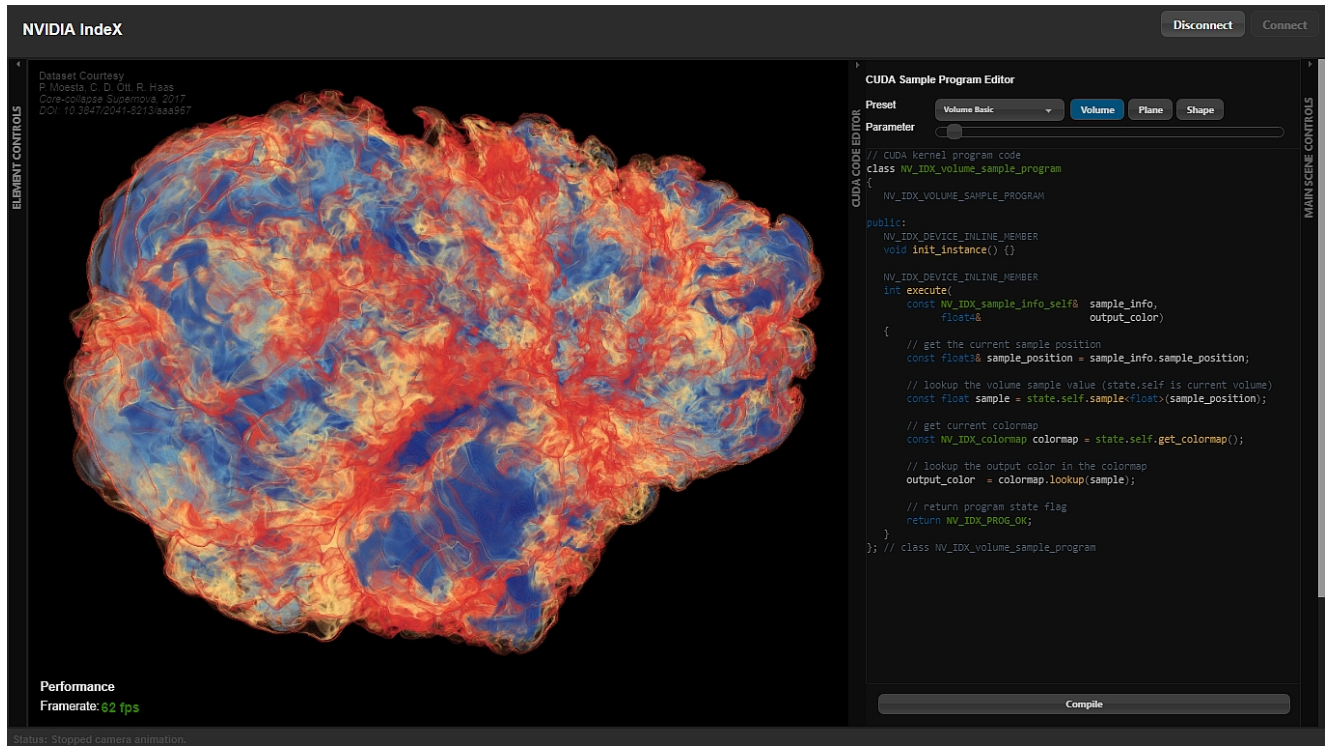
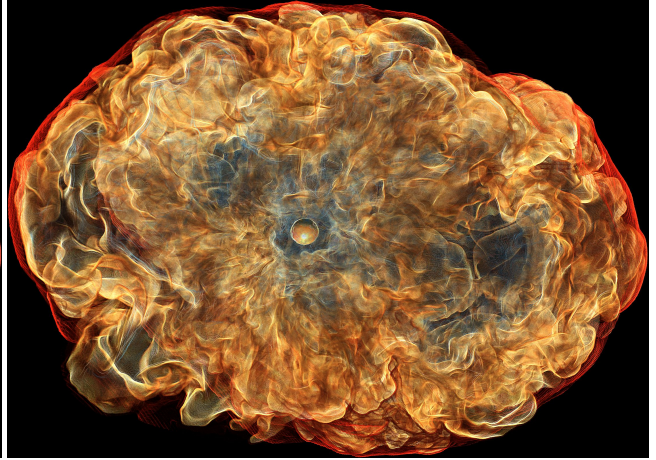
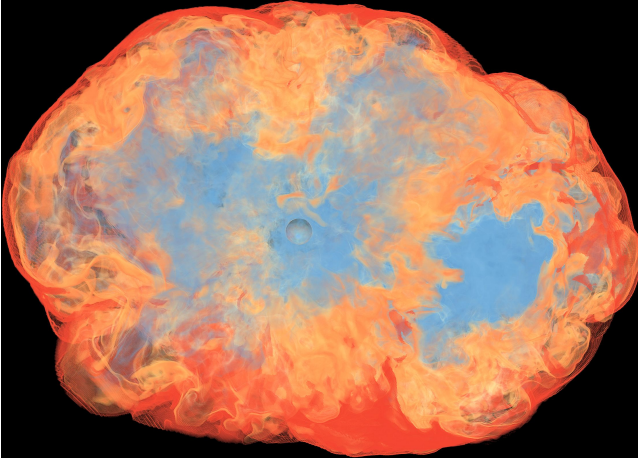


Figure 2: **Web-based Volume Rendering Client:** Preview of the web-based CUDA program editor to interactively change the executed ray sampling code.



```
NV_IDX_XAC_VERSION_1.0

using namespace nv::index::xac;

class Volume_sample_program
{
    NV_IDX_VOLUME_SAMPLE_PROGRAM

public:
    void initialize() {...}

    int execute(
        const Sample_info_self&    sample_info ,
        Sample_output&             sample_output)
    {
        // get the current sampling position
        float3& sp = sample_info.sample_position;

        // lookup the volume sampling value
        const Regular_volume volume = state.self;
        float sv = volume.sample<float>(sp);

        // get the current sample color
        const Colormap map = volume.get_colormap();
        float4 sample_color = map.lookup(sv);

        // store the computed output color
        sample_output.set_color(sample_color);

        // return program success
        return NV_IDX_PROG_OK;
    }
};
```

```
NV_IDX_XAC_VERSION_1.0

using namespace nv::index::xac;

class Volume_sample_program
{
    NV_IDX_VOLUME_SAMPLE_PROGRAM

public:
    void initialize() {...}

    int execute(
        const Sample_info_self&    sample_info ,
        Sample_output&             output)
    {
        const Regular_volume volume = state.self;

        // get the current volume sampling position
        float3& sp = sample_info.sample_position;

        // sample the volume scalar (here: entropy)
        float sv = volume.sample<float>(sp);

        // get transfer function color value
        const Colormap map = volume.get_colormap();
        float4 color = map.lookup(sv);

        // compute the volume gradient vector
        float3 gradient = volume.get_gradient(sp);

        // scale the color by gradient magnitude
        float scale = length(vol_grad);
        sample_color = clamp(color * scale, 0, 1);

        // set the compute color as output color
        output.set_color(color);

        return NV_IDX_PROG_OK;
    }
};
```

Figure 3: **Basic Volume Sampling Programs:** Two volume programs applied to visualize the core-collapse in the s15WH07 model: The program on the left shows a standard volume rendering, mapping the sampled values to a color using the transfer function. The program on the right shows a modified version, using the gradient of the field to scale the color and transparency output values.

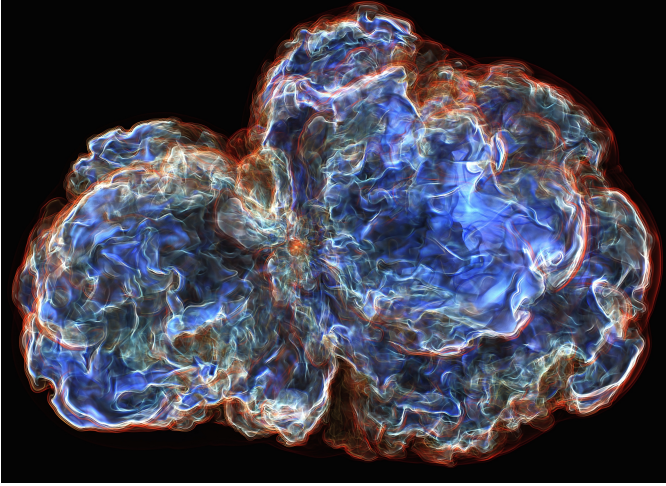


Fig. a) Gradient-enhanced rendering

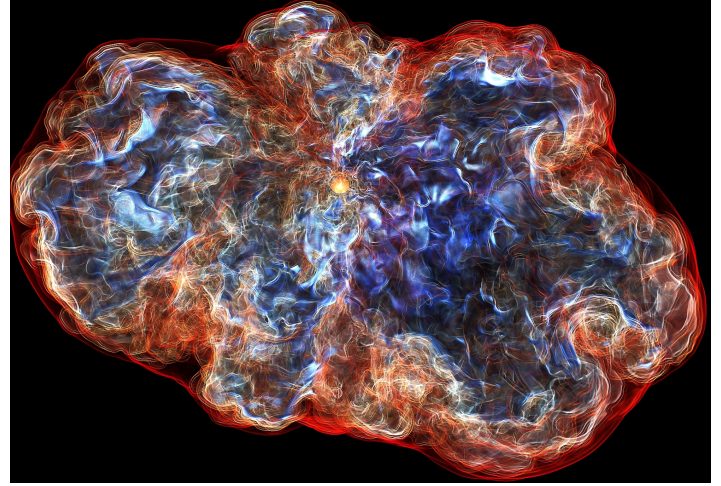


Fig. b) Gradient-enhanced rendering, clipped

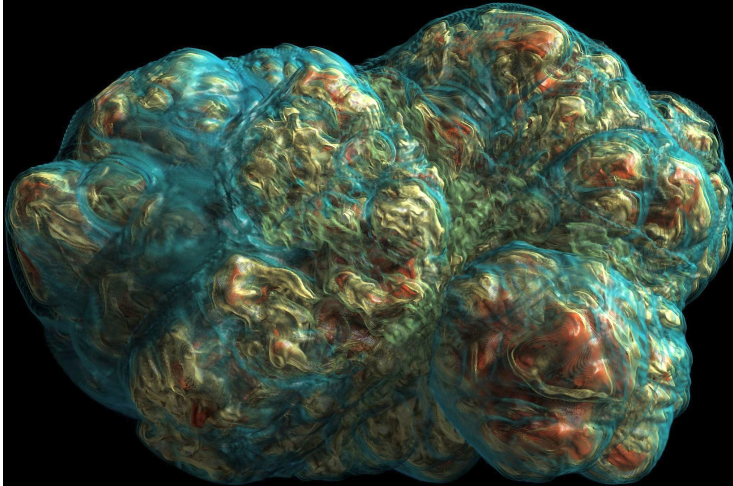


Fig. c) Single-scattering program

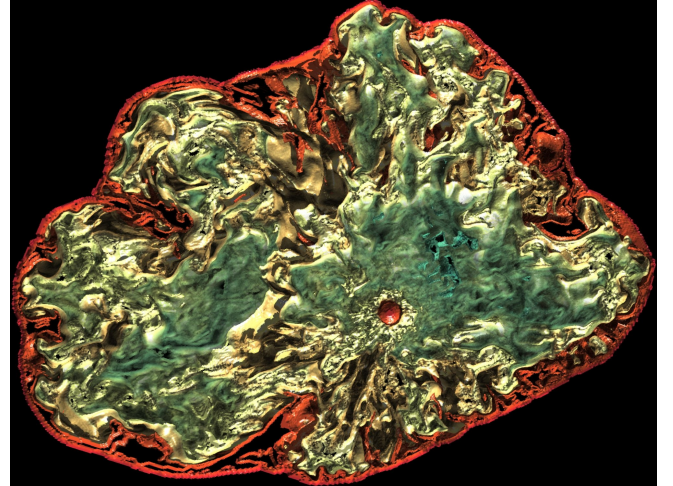


Fig. d) Single-scattering program, clipped

Figure 4: Example Volume Sampling Programs: The images show the adjustment of sampling programs and transfer functions to explore different options for the visual appearance of specific features: Fig. a) shows the gradient-enhanced kernel with an adapted transfer function to emphasize areas of larger entropy gradients (brighter colors denote higher gradient magnitudes) while keeping a high overall transparency; Fig. b) clips the same volume to reveal internal structure and highlights several layers of the shock front; Fig. c) uses a simple single-scattering model to emphasize bubble structures within the outer shock-front; Fig. d) clips the volume with the same sampling program, but different transfer function to visualize the entropy distribution within the inner shock region (red represents areas of low entropy, blue represents areas of high entropy, gradient directions are used for local shading).