NVIDIA IndeX

ParaView Plugin User's Guide

27 August 2024
Version 5.13.0

NVIDIA IndeX – ParaView Plugin User's Guide

Document build number rev380247

Contents

1 Introduction

The NVIDIA IndeX® for ParaView® plugin makes the large-scale and high-quality volume data visualization capabilities of the NVIDIA IndeX library available to Kitware's ParaView.

This document is intended for a ParaView user who is new to the NVIDIA IndeX plugin and wants to explore the features of NVIDIA IndeX supported by the plugin. The following sections will explain the installation procedure and the various features of the plugin, followed by a section of frequently asked questions and useful links for further reference.

For more details and download options please go to:

`https://www.nvidia.com/en-us/data-center/index-paraview-plugin`

1.1 Licensing

The NVIDIA IndeX for ParaView Plugin comes with a free evaluation license that enables all features for a limited time, including full scalability to run on multiple NVIDIA GPUs and on a cluster of GPU hosts. The software will continue to run after the evaluation period, but with multi-GPU features disabled.

For licensing requests, please contact `nvidia-index@nvidia.com`.

2 Installation

The NVIDIA IndeX for ParaView plugin is included in ParaView 5.13.0 and runs on Windows, Linux (x86-64 and ARM/aarch64) and macOS (remote rendering only). To install ParaView, you can either download a binary package[1] or build from source.

2.1 System requirements

The NVIDIA IndeX for ParaView plugin 5.13.0 requires:

- An NVIDIA GPU that supports at least CUDA compute capability 5.0, i.e. "Maxwell" GPU architecture or newer.

  To find out the compute capability of a specific GPU, go to `https://developer.nvidia.com/cuda-gpus`.

- NVIDIA display driver that supports CUDA 12. The minimum driver version is:
  - Linux: 525.60.13
  - Windows: 527.41
- Recommended driver version (CUDA 12.3):
  - Linux: 545.23.06 or newer
  - Windows: 545.84 or newer
- When connecting to a remote `pvserver` (see Simple client/server mode (page 6)), a suitable NVIDIA GPU and display driver is required on the remote host but not on the local host where the ParaView client is running.
- Supported operating systems:
  - Red Hat Enterprise Linux (RHEL) or CentOS version 7 or newer. The NVIDIA IndeX for ParaView plugin will typically also run on other Linux distributions.
  - Microsoft Windows 10.
  - macOS. The macOS version of the NVIDIA IndeX for ParaView plugin does not support local rendering. However, it can be used for remote rendering together with a Linux host that runs `pvserver` (see Simple client/server mode (page 6)).

2.2 Building ParaView and the NVIDIA IndeX plugin

If you have installed ParaView using a binary package provided by Kitware, then the plugin is already included and you can continue to "Loading the plugin" (page 3).

1. `https://www.paraview.org/download/`

The NVIDIA IndeX plugin can also be built from source, together with ParaView. See the ParaView build documentation[2] for a general overview of the build process. For compiling the plugin, two additional steps are necessary:

1. Since the NVIDIA IndeX plugin is not enabled in the default build, enable it by setting `PARAVIEW_PLUGIN_ENABLE_pvNVIDIAIndeX=ON` when running `cmake`, for example:

   ```
   cmake -DPARAVIEW_PLUGIN_ENABLE_pvNVIDIAIndeX=ON ../paraview-source
   ```

2. Download the latest NVIDIA IndeX libraries package for your platform from the ParaView dependency repository: `https://www.paraview.org/files/dependencies/`

   Windows
   `nvidia-index-libs-5.12.0.<YYYYMMDD>-windows-x64.tar.bz2`

   Linux x86-64
   `nvidia-index-libs-5.12.0.<YYYYMMDD>-linux.tar.bz2`

   Linux ARM/aarch64
   `nvidia-index-libs-5.12.0.<YYYYMMDD>-linux-aarch64.tar.bz2`

   macOS
   NVIDIA IndeX libraries are not available for macOS, but are also not required for remote rendering.

   Extract the contents of the downloaded package to the `lib` directory inside your ParaView installation, so that ParaView can find the libraries at runtime.

2.3 Loading the plugin

To load the plugin in ParaView, start the ParaView client and navigate to [Tools ▸ Manage Plugins] in the menu.

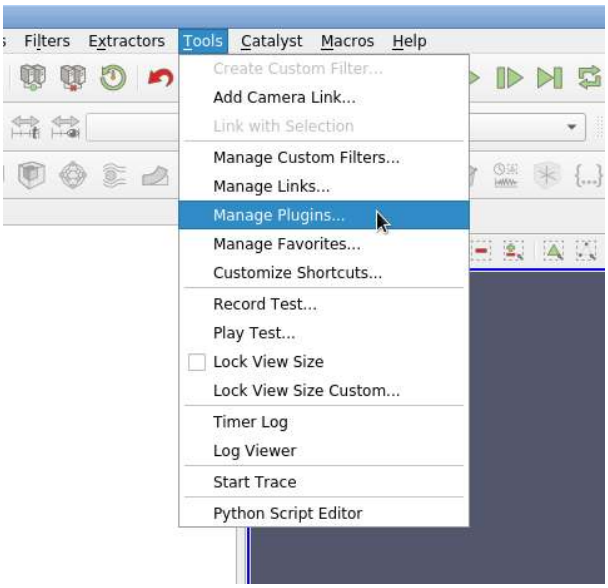

Fig. 2.1 - [Tools ▸ Manage Plugins] from ParaView menu

2. `https://gitlab.kitware.com/paraview/paraview/-/blob/master/Documentation/dev/build.md`

Select *pvNVIDIAIndeX* in the list of “Local Plugins” and choose *Load Selected*. To automatically load the plugin when ParaView starts, expand the list entry and check *Auto Load*.

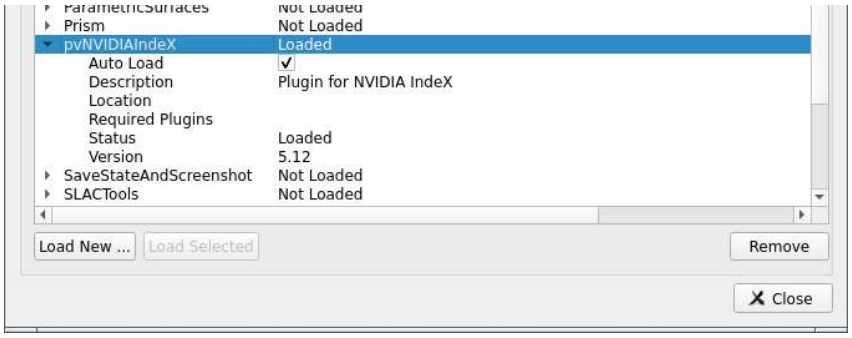

Fig. 2.2 - The plugin is shown as loaded

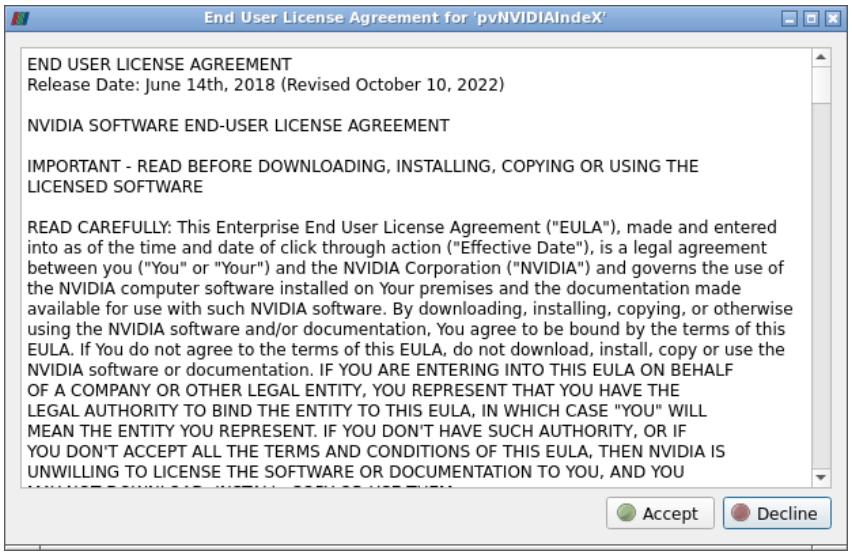

Fig. 2.3 - Read the EULA and click Accept

When loading the plugin for the first time, please read and accept the End User License Agreement (EULA).

Once the plugin is loaded, its name will show up in the list of “Local Plugins” with its status changed to *Loaded*. Make sure there are no errors messages shown in the terminal or on ParaView’s console.

When using the plugin in client/server mode, it must be loaded in both “Local Plugins” and “Remote Plugins”.

3 Getting started

This section will describe how to use the NVIDIA IndeX for ParaView plugin in both client-only and client/server mode.

3.1 Client-only mode

To run the plugin in client-only mode, simply launch the ParaView client (`paraview`) and load the plugin as described in Section 2.

With the default evaluation license or a separately provided license for the cluster version of the plugin, NVIDIA IndeX will automatically utilize all available NVIDIA GPUs for rendering. After the evaluation period, only a single GPU can be used with the default license.

To verify that the plugin is installed correctly and loaded successfully in ParaView, please create a *Wavelet* source (menu [Sources ▸ Alphabetical ▸ Wavelet]). Once you click *Apply*, an *Outline* representation will be shown in the viewport. Change the representation from *Outline* to *NVIDIA IndeX* to enable volume visualization.

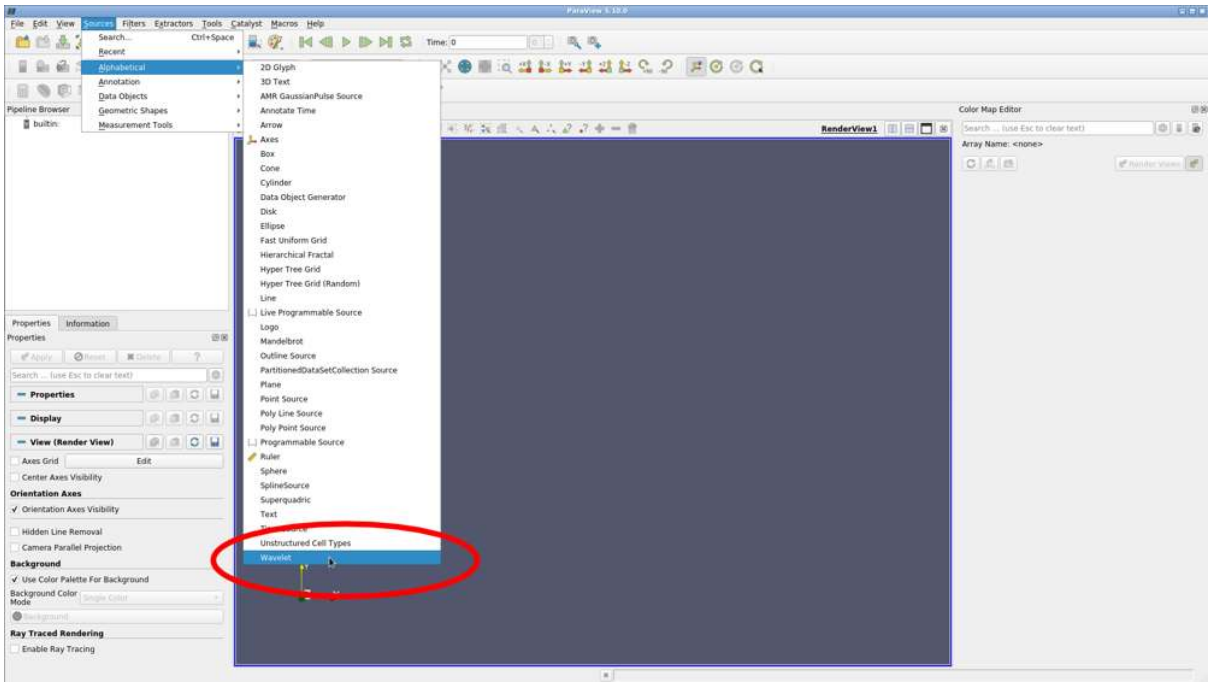

Fig. 3.1 - Create a Wavelet source

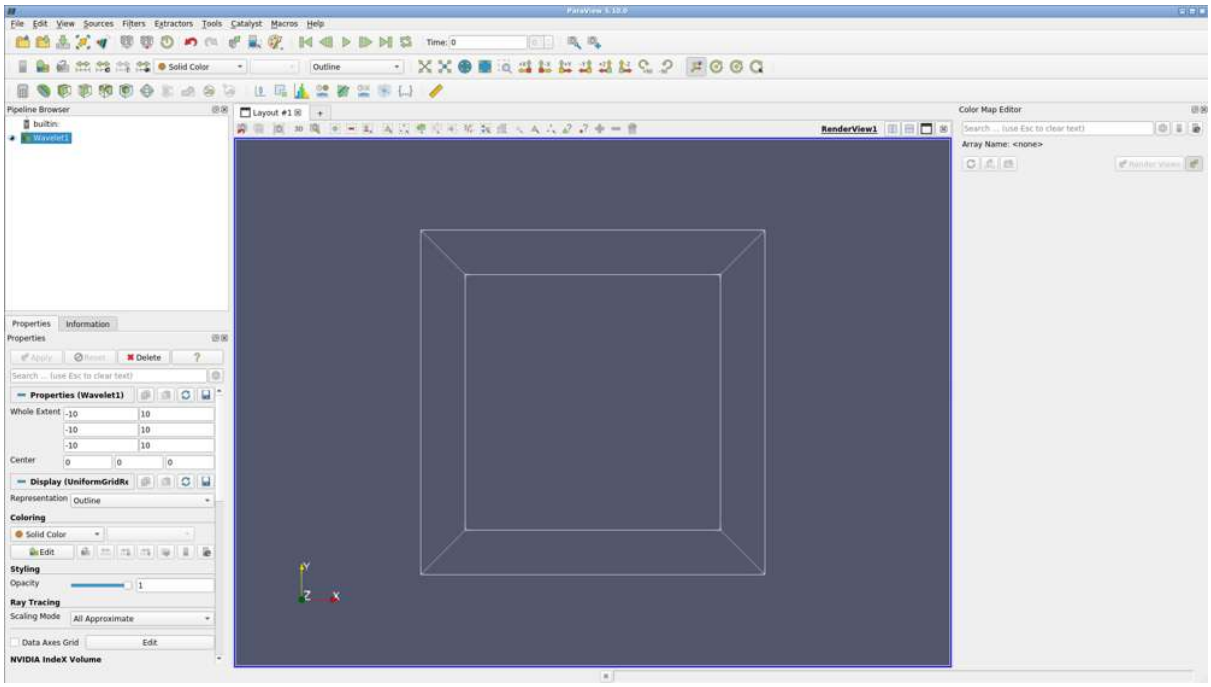

Fig. 3.2 - Outline is the default representation

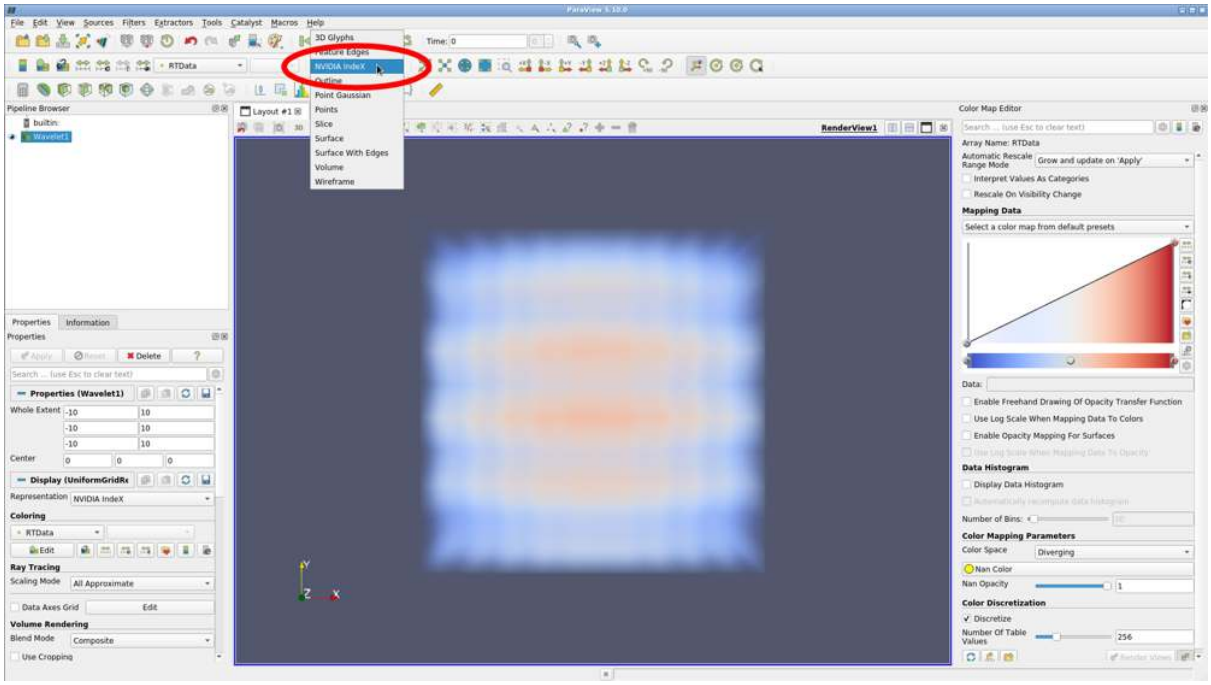

Fig. 3.3 - Wavelet source rendered by NVIDIA IndeX

3.2 Simple client/server mode

Please refer to the ParaView Wiki[1] for a general overview of `pvserver` and ParaView's client/server mode.

1. `https://www.paraview.org/Wiki/Setting_up_a_ParaView_Server`

To run ParaView with the NVIDIA IndeX plugin in client/server mode, first start `pvserver` on a host equipped with a suitable NVIDIA GPU:

```
pvserver --displays=:0
```

When using EGL, you need to specify the EGL device index (`0`, `1`, etc.) instead of the X11 display (`:0`). Please note that, in either case, this selection will not influence what GPUs will be used by NVIDIA IndeX, but only what other ParaView renderers will use.

Once the `pvserver` process is launched, run the ParaView client and connect to the host where `pvserver` is running by selecting [File ▸ Connect ▸ Add Servers] in the menu. Typically the server address is printed out in the console where `pvserver` was executed and, once the client has connected, "Client connected" will be printed there. Make sure to load the NVIDIA IndeX plugin on both the client and the server side as described in "Loading the plugin" (page 3).

3.3 Client/server mode with MPI

Running multiple `pvserver` processes enables parallel data loading and data processing in ParaView. The following command will launch `pvserver` on 8 MPI ranks:

```
mpirun -np 8 pvserver --displays=:0
```

Alternatively, the `mpiexec` program that is part of the ParaView package can be used:

```
mpiexec -n 8 pvserver --displays=:0
```

While NVIDIA IndeX will automatically use all available NVIDIA GPUs, other renderers in ParaView will utilize only those GPUs that are explicitly specified with `--displays`. Multiple GPUs can be specified, for example, with `--displays=:0.0,:0.1,:0.2,:0.3` (for X11) or `--displays=0,1,2,3` (for EGL). These GPUs will then be assigned to the MPI ranks in a round-robin fashion. See the ParaView documentation for details. Note that the `--displays` parameter will not affect the GPU assignment used by NVIDIA IndeX.

A valid license for the cluster version of the plugin is required for NVIDIA IndeX to be able to utilize more than a single GPU. The plugin comes with a default evaluation license that enables all multi-GPU and cluster features for a limited time. With a valid license in place (see "License setup" (page 10)), NVIDIA IndeX can also distribute the rendering workload to multiple GPUs on multiple hosts in a cluster environment. For example, this command will launch 4 `pvserver` processes each on `host1` and `host2`, which are equipped with 2 GPUs:

```
mpirun -H host1,host2 --bynode -np 4 pvserver --displays=:0.0,:0.1
```

Or, with `mpiexec`:

```
mpiexec -ppn 4 -hosts host1,host2 pvserver --displays=:0.0,:0.1
```

When exclusively using NVIDIA IndeX for rendering, it would be sufficient to just specify `--displays=:0`, independent of the actual number of GPUs.

Before connecting the ParaView client to `pvserver`, please ensure that *IceT compositing* is disabled. This can be controlled from in the *Settings* dialog at [Edit ▸ Settings ▸ Render View]. Restart ParaView after changing the setting. An error message will be printed if IceT compositing is still enabled when running the NVIDIA IndeX plugin on multiple MPI ranks.

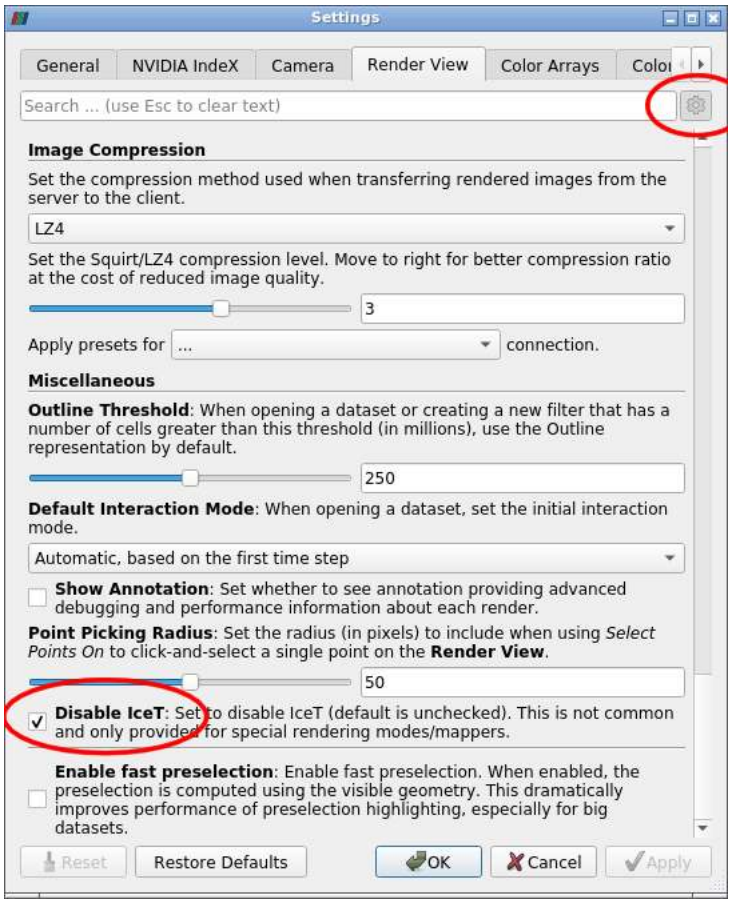

Fig. 3.4 - Disable IceT in the Settings *dialog (*Toggle advanced properties *in the top right corner must be enabled)*

3.4 NVIDIA IndeX settings

When the NVIDIA IndeX plugin is loaded, a new *NVIDIA IndeX* tab will appear in the *Settings* dialog. Enable the *Toggle advanced properties* button in the top right corner to show all available settings.

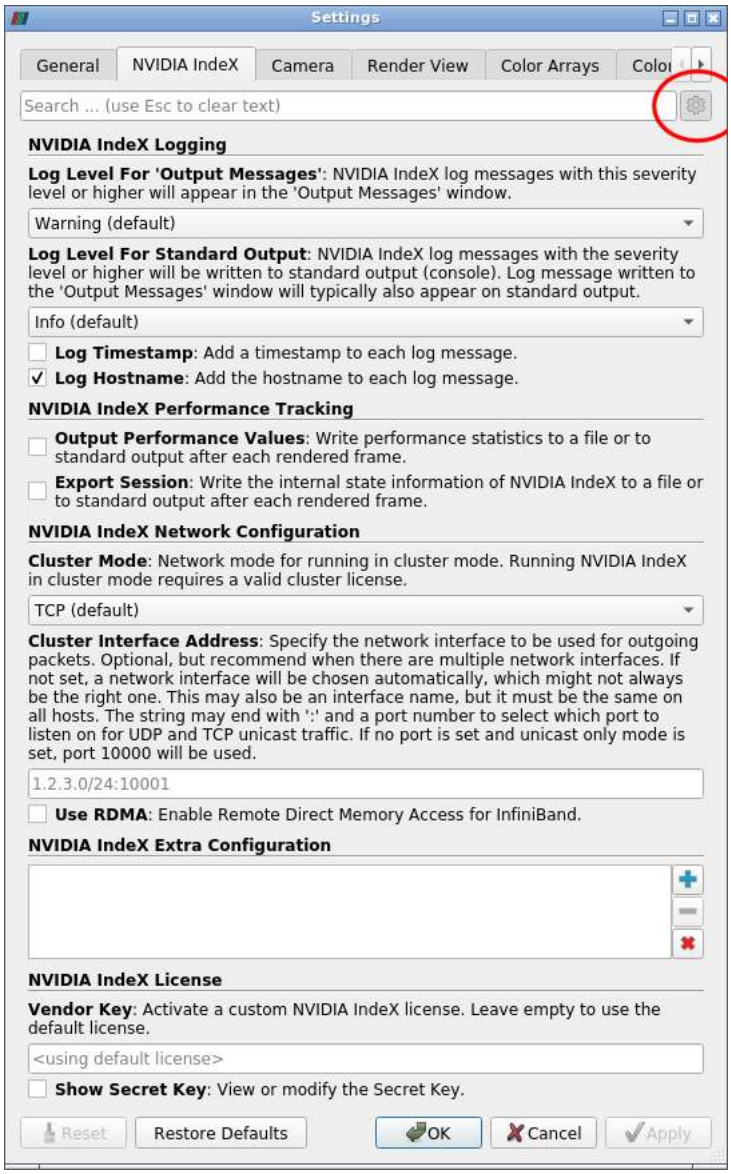

Fig. 3.5 - NVIDIA IndeX tab in the Settings *dialog*

The following setting are available:

- Logging: The minimum required severity level for messages to appear in ParaView's *Output Messages* window as well as on standard output can be set. The current timestamp and the hostname can optionally be added to each log message.
- Performance values: Write statistics about rendering performance to a file or to standard output.
- Session export: Write the state of the NVIDIA IndeX scene to a file or to standard output.
- Network configuration: Various settings for running on multiple hosts in cluster mode.
- Extra configuration: Allows setting low-level configuration options, e.g. for performance tuning.
- License keys: An NVIDIA IndeX license key that was obtained from NVIDIA can now be entered here.

3.5 Advanced NVIDIA IndeX configuration

Advanced features of the NVIDIA IndeX plugin can be configured in the optional configuration file `nvindex_config.xml`. Any settings specified in the configuration file have precedence over those from the *Settings* dialog.

To change the configuration, copy the template of the configuration file[2] to the ParaView settings directory[3] and modify it. On Linux the ParaView settings directory will be `$HOME/.config/ParaView` or `$XDG_CONFIG_HOME/ParaView`, while on Windows, settings are stored in `%APPDATA%\ParaView`.

If the ParaView settings directory is not accessible, you can set the environment variable `NVINDEX_PVPLUGIN_HOME` to point to the directory that contains `nvindex_config.xml`:

```
export NVINDEX_PVPLUGIN_HOME=path-to-directory-with-config-file
```

3.5.1 License setup

When you obtain a license for the plugin from NVIDIA, you will receive a file named `license.lic`. It contains the keys `NVINDEX_VENDOR_KEY` and `NVINDEX_SECRET_KEY` that need to be set as environment variables on all the hosts where NVIDIA IndeX is run.

```
export NVINDEX_VENDOR_KEY="vendor-key-here"
export NVINDEX_SECRET_KEY="secret-key-here"
```

Alternatively, enter those keys into the *Settings* dialog or copy them into the `<license>` section of the configuration file as shown below.

```
<index_config>
  <license>
    <vendor_key>vendor-key-here</vendor_key>
    <secret_key>secret-key-here</secret_key>
  </license>
</index_config>
```

3.5.2 Networking options

All the networking configuration options for the plugin are specified in the `<network>` section of the configuration file.

```
<index_config>
    <network>
        <cluster_mode>TCP</cluster_mode>
        <cluster_interface_address>
          172.161.123.0/24:10001
        </cluster_interface_address>
```

2. `https://gitlab.kitware.com/paraview/paraview/-/raw/master/Plugins/pvNVIDIAIndeX/nvindex_config.xml`
3. `https://www.paraview.org/Wiki/ParaView_Settings_Files`

```
        <multicast_address>224.1.3.2</multicast_address>
        <discovery_address>224.1.3.3:5555</discovery_address>
        <use_rdma>no</use_rdma>
    </network>
</index_config>
```

`<cluster_mode>` defines the networking mode of NVIDIA IndeX. The modes `OFF`, `UDP`, `TCP` and `TCP_WITH_DISCOVERY` are supported, and `TCP` is the default mode.

`<cluster_interface_address>` defines the network interface that is used for communication between the hosts. This may also be an interface name, but it must be the same on all hosts. If not set, a network interface will be chosen automatically, which might not always be the right one. The string may end with a colon character (:) and a port number to select which port to listen on for UDP and TCP unicast traffic. If no port is set then port 10000 will be used.

`<multicast_address>` defines the multicast address for communication between the hosts. This will only be used when `cluster_mode` is set to `UDP`.

`<discovery_address>` defines the discovery address used for `TCP_WITH_DISCOVERY`.

`<use_rdma>` can be set to `yes` or `no` to enable or disable InfiniBand RDMA mode for networking.

4 NVIDIA IndeX features

This section will provide a walk-through of the individual features of the NVIDIA IndeX for ParaView plugin.

4.1 Features overview

- Real-time and interactive high-quality volume data visualization of both structured and unstructured volume grids.
- Interactive visualization of time-varying structured volume grids.
- Support for 8-bit and 16-bit signed/unsigned integer and 32-bit floating point volume data types. 64-bit floating point data and 32-bit integer data is supported via an automatic conversion.
- XAC (NVIDIA IndeX Accelerated Compute) visual elements for volume rendering with four different configurable presets: Isosurfaces, depth enhancement, edge enhancement, gradient.
- User-defined programmable XAC visual element for volumes.
- Multiple, axis-aligned volume slice rendering combined with volumetric data.
- Catalyst support for regular grids to perform in-situ based visualization.
- User-defined cropping and region of interest selection.
- Advanced filtering and pre-integration techniques enabling high-fidelity visualizations.
- Depth-correct integration of ParaView geometry rendering into NVIDIA IndeX volume rendering.
- Multi-GPU and GPU cluster support for scalable real-time visualization of datasets of arbitrary size. Requires an appropriate license after the evaluation period (please contact NVIDIA).

4.2 Structured and unstructured grids

The NVIDIA IndeX for ParaView plugin can perform volume rendering of both structured and unstructured grid types.

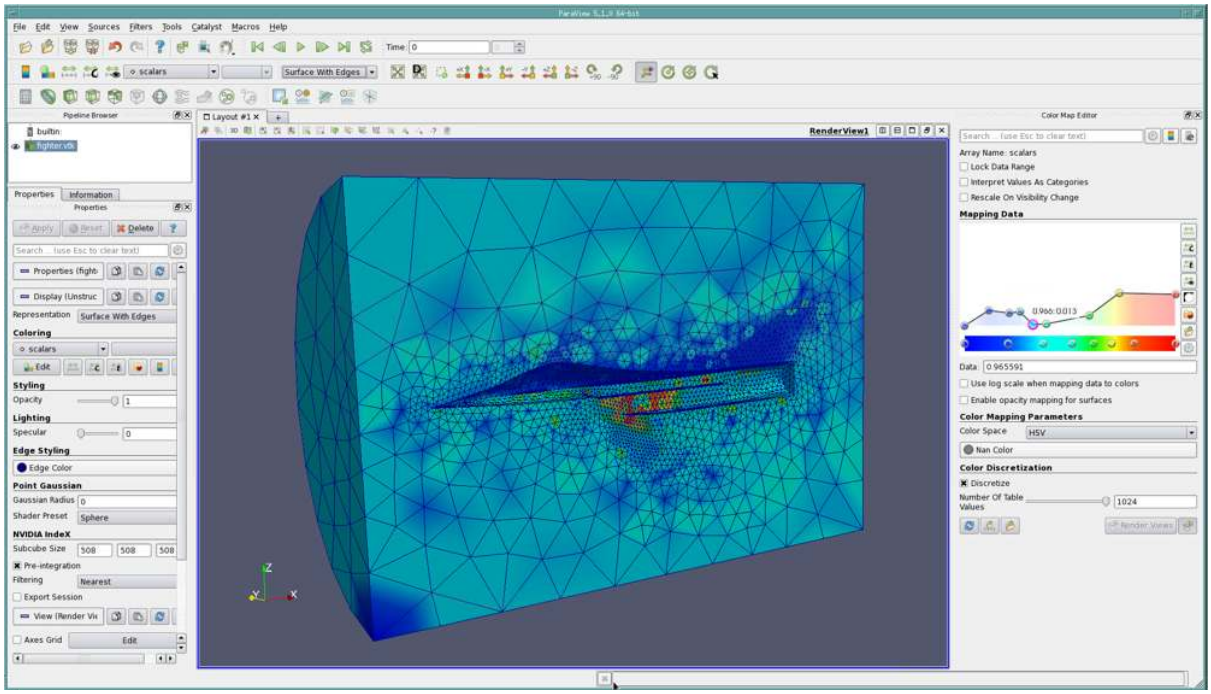

Fig. 4.1 - Grid rendered as a surface

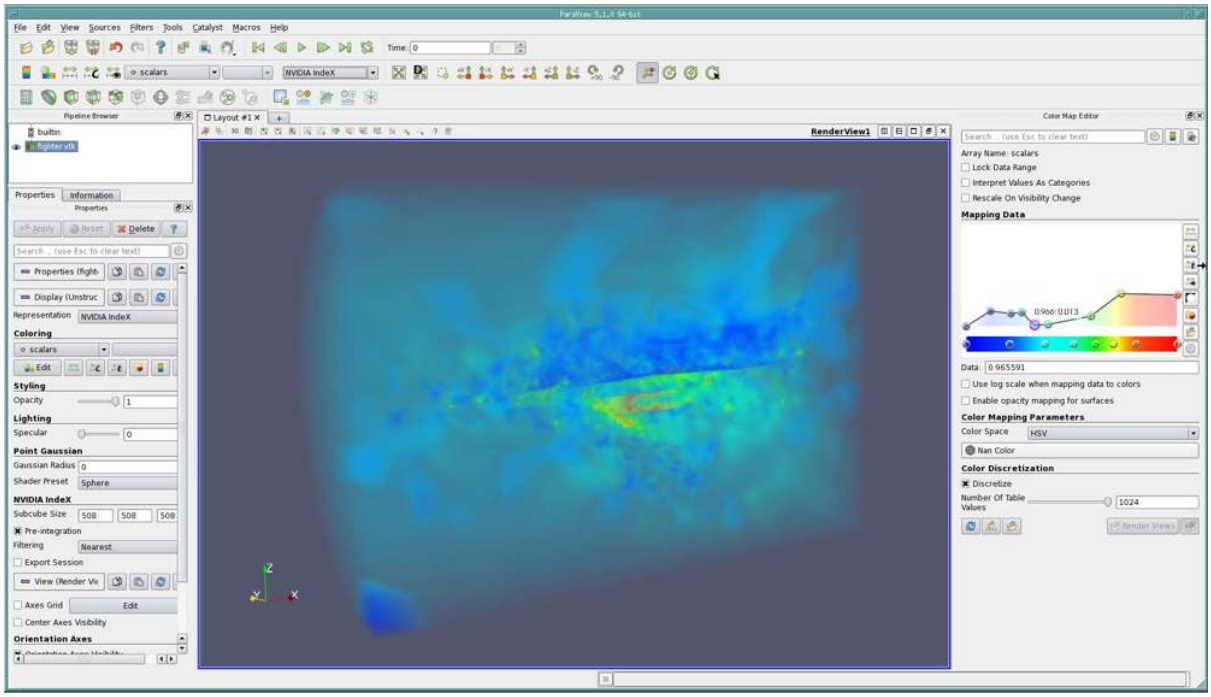

Fig. 4.2 - Grid rendered as a volume using NVIDIA IndeX

4.3 Data types

NVIDIA IndeX supports different data scalar types such as *unsigned char*, *unsigned short* and *float*. Data in *double* format will be automatically converted to *float*.

When loading raw data, always make sure that the correct byte order is chosen, as otherwise your visualization might have artifacts or even look completely random.

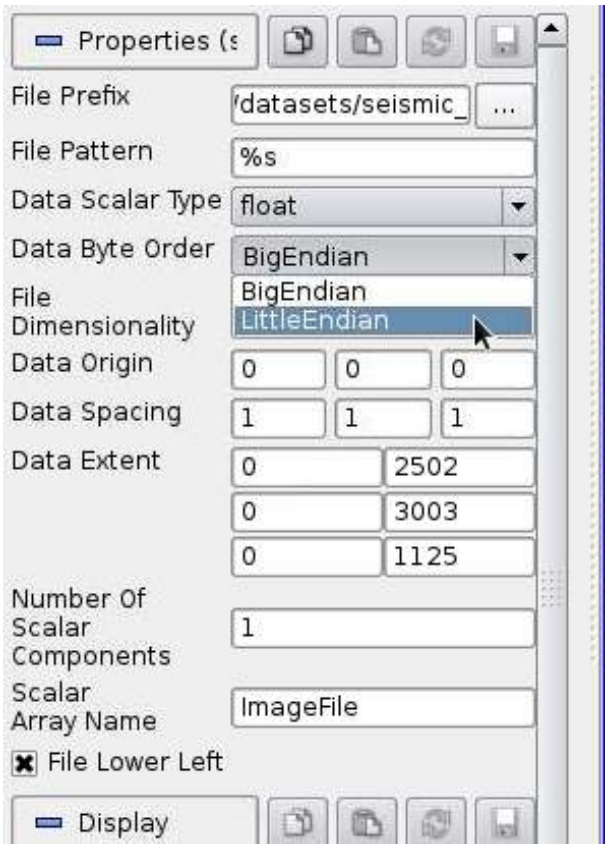

Fig. 4.3 - Choose data byte order

4.4 Transfer function and colormap

The transfer function used for volume rendering can be modified using ParaView's colormap editor.

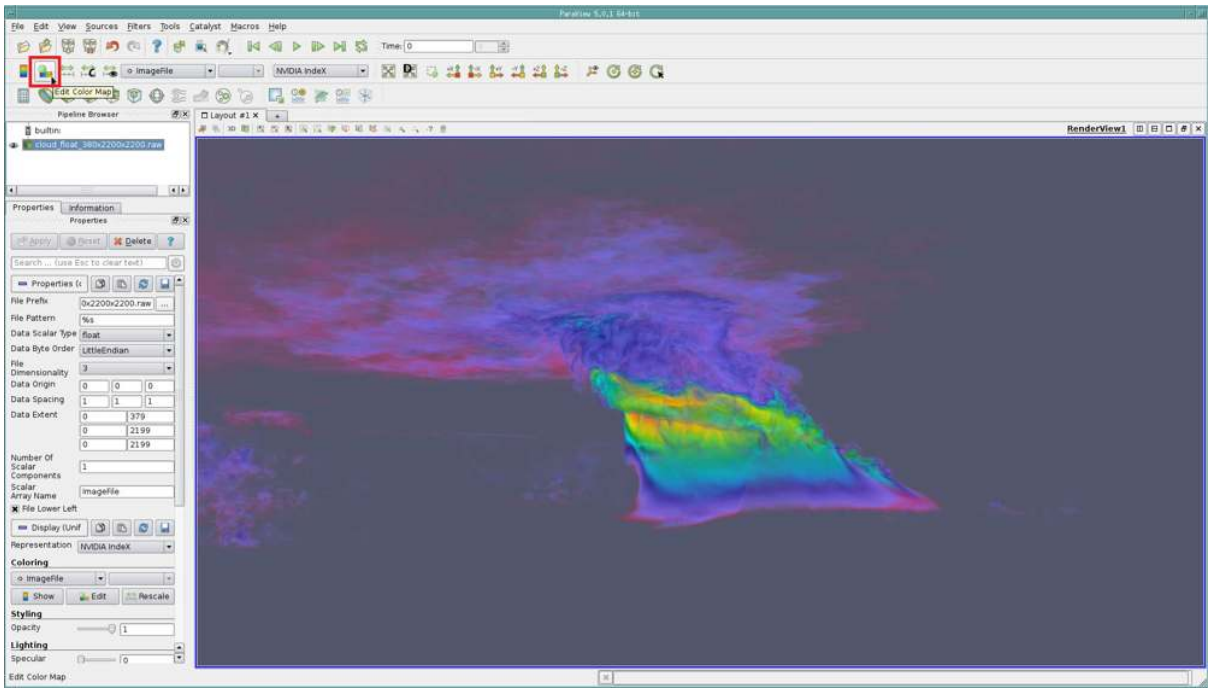

Fig. 4.4 - Click on the icon to open up the colormap editor

By editing the colormap you can restrict the visualization to regions of the dataset that are interesting to you. This can be achieved by changing the colortable, by adjusting the transparency, or by setting custom domain range values to hide specific parts of the dataset.

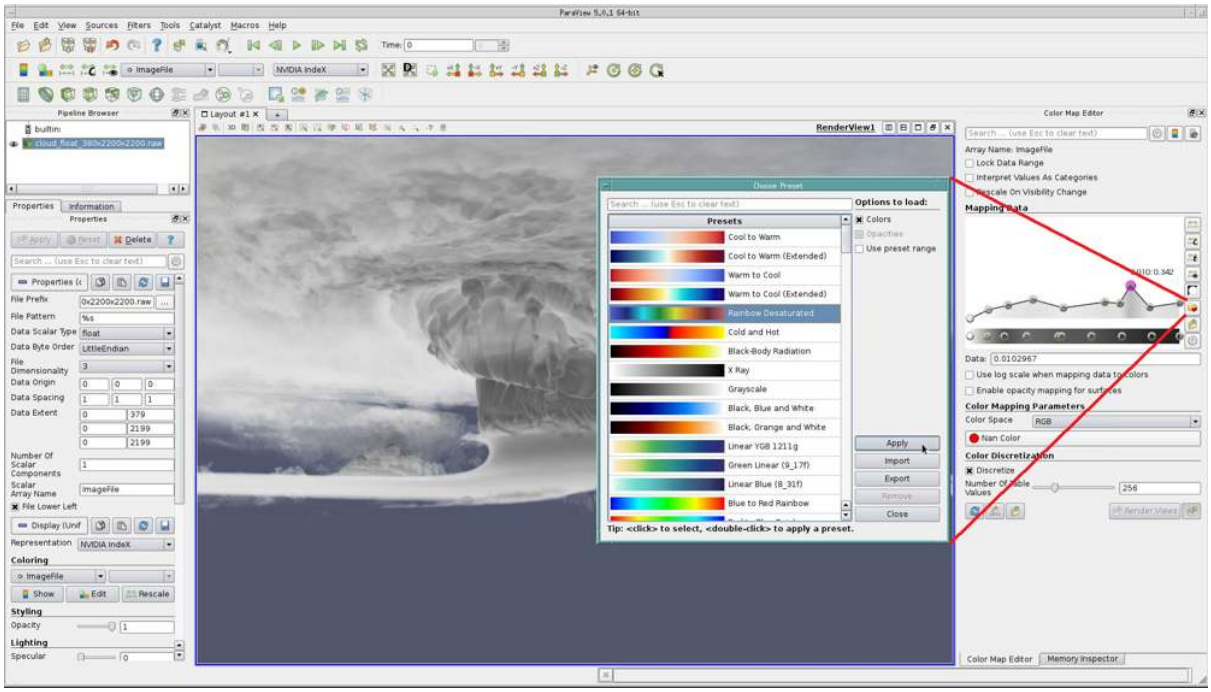

Fig. 4.5 - Choose a suitable colortable and click Apply

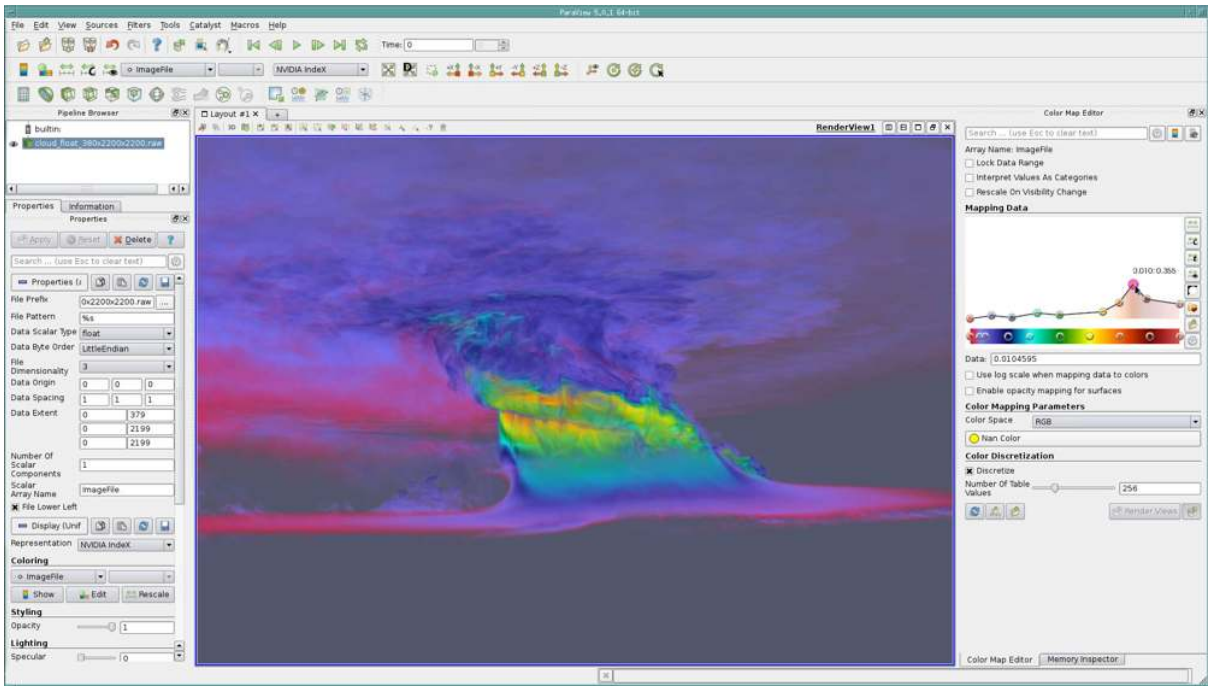

Fig. 4.6 - Colortable matching your dataset domain

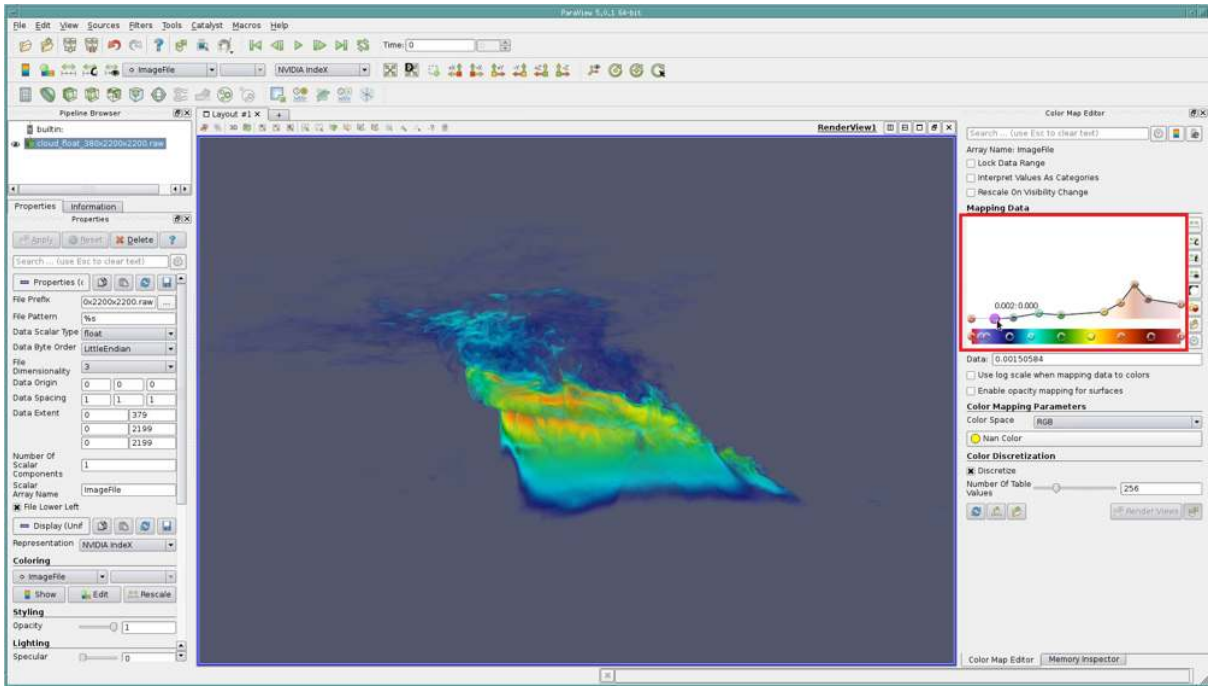

Fig. 4.7 - Changing the opacity values

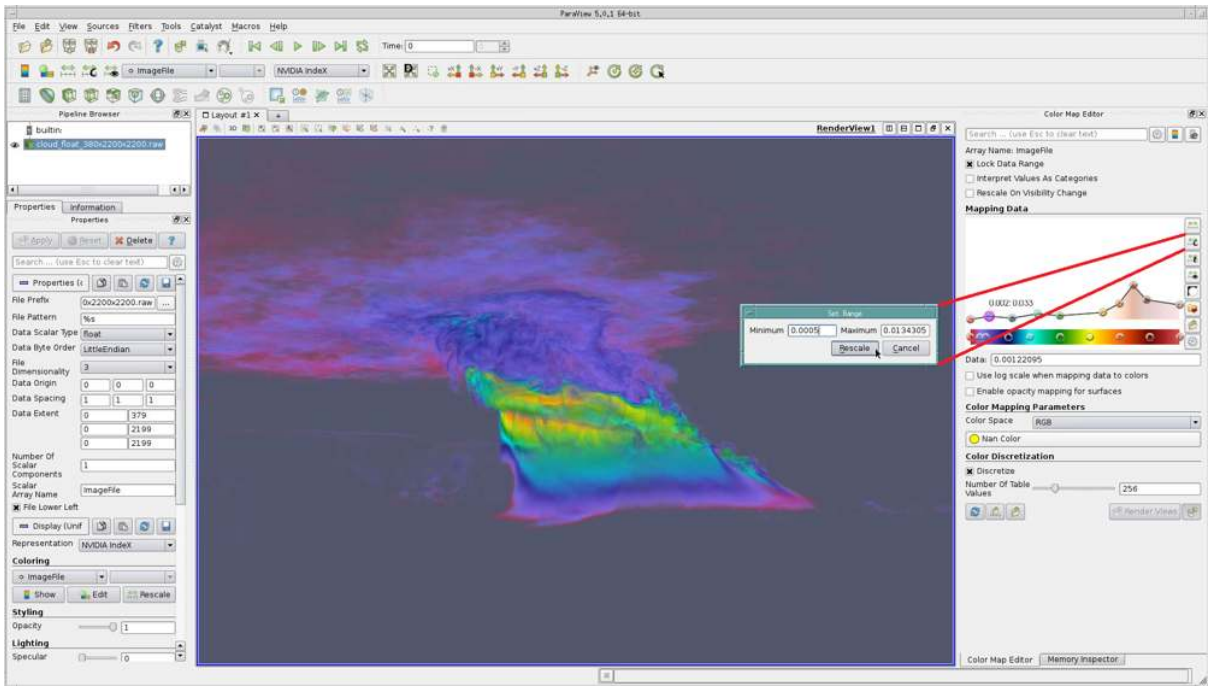

Fig. 4.8 - Custom data domain range

4.5 Cropping / region of interest

Structured grids can be cropped using the standard ParaView cropping controls. After enabling *Use cropping* in the *Properties* panel, you can apply cropping either by dragging the control points in the viewport with the mouse or by manually changing the *Cropping Origin* and *Cropping Scale* settings.

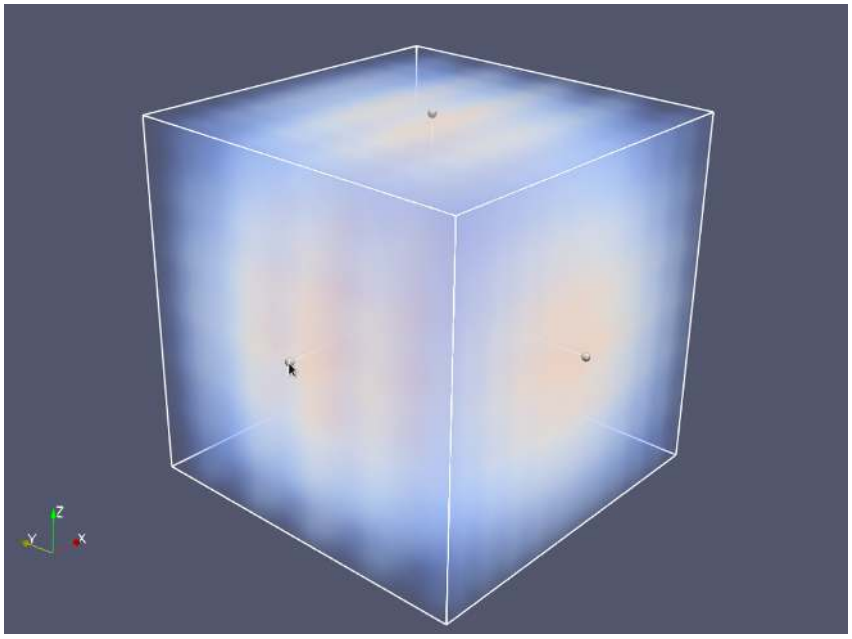

Fig. 4.9 - Wavelet volume with cropping controls visible

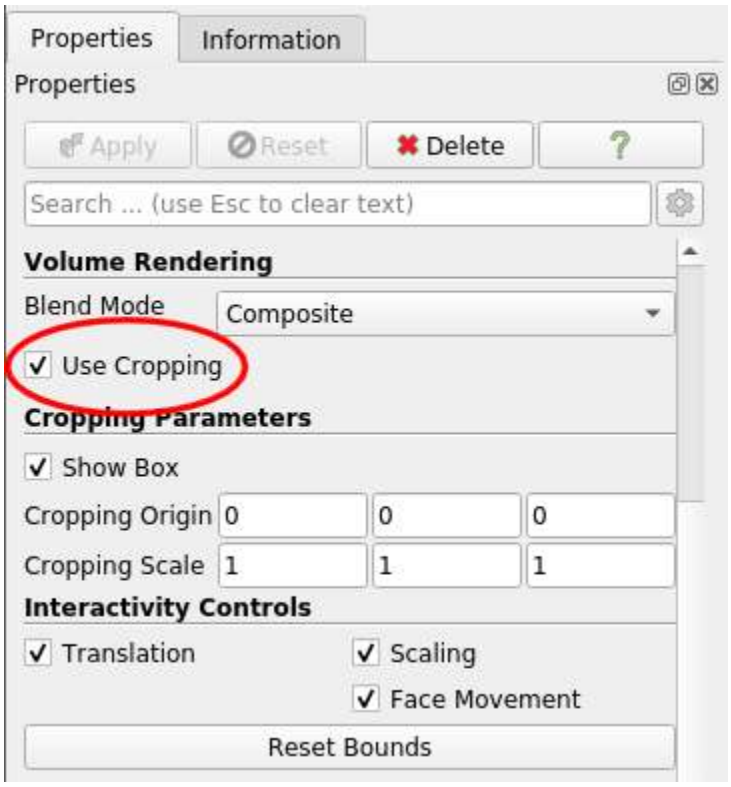

Fig. 4.10 - Cropping settings for structured grids

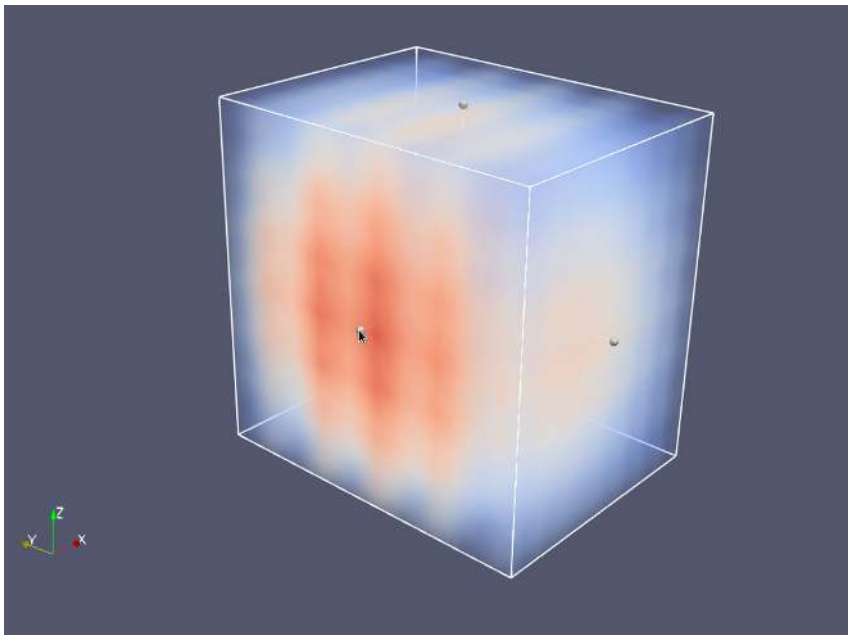

Fig. 4.11 - Wavelet volume with a custom cropping applied by dragging a control point

For unstructured grids, cropping is controlled using the sliders in the *NVIDIA IndeX Region of Interest* section of the *Properties* panel.

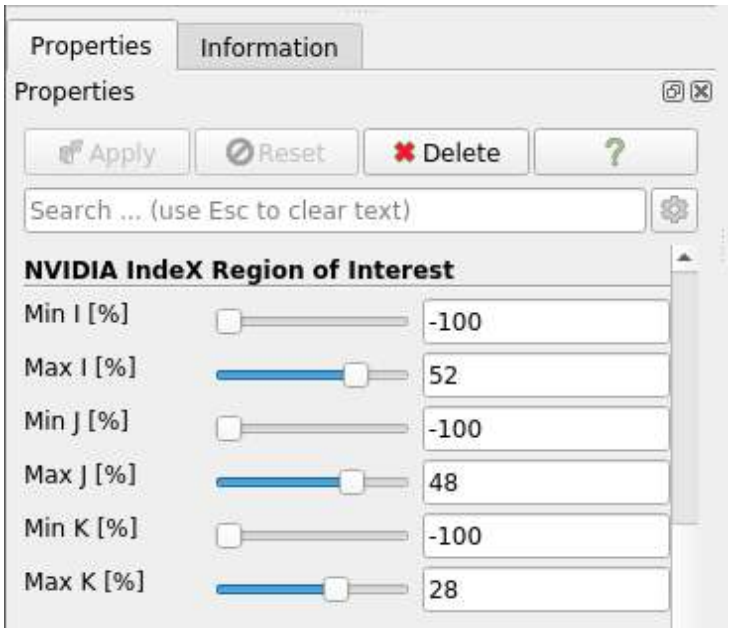

Fig. 4.12 - Changing the region of interest for unstructured grids

4.6 High-quality rendering

High-quality rendering can be achieved by enabling the pre-integration and filtering techniques of NVIDIA IndeX. Some of these filtering techniques are computationally expensive, so there is a tradeoff between rendering quality and performance.

The filtering options can be found in the *Properties* panel, typically located on the left hand side of the ParaView client user interface.

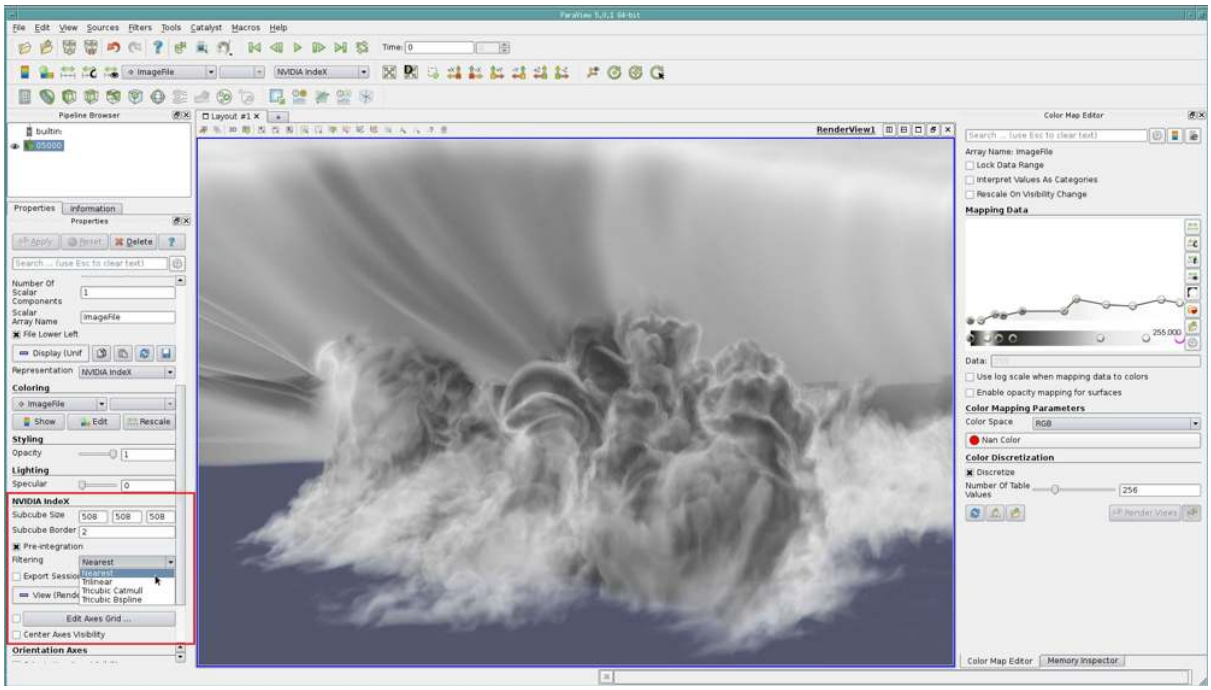

Fig. 4.13 - Properties *panel in ParaView*

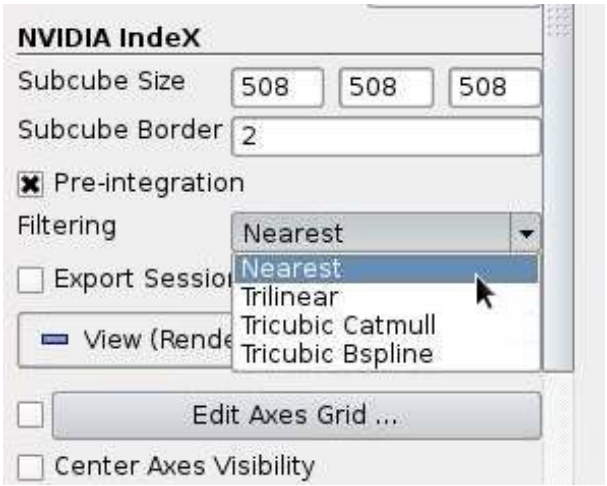

Fig. 4.14 - Filtering options in the NVIDIA IndeX properties panel

There is no single filtering mode that will be optimal for all datasets, but each filtering mode achieves different levels of quality with different datasets and transfer function combinations, with nearest neighbor interpolation being the most basic one. Some example images comparing different filtering modes are shown below.

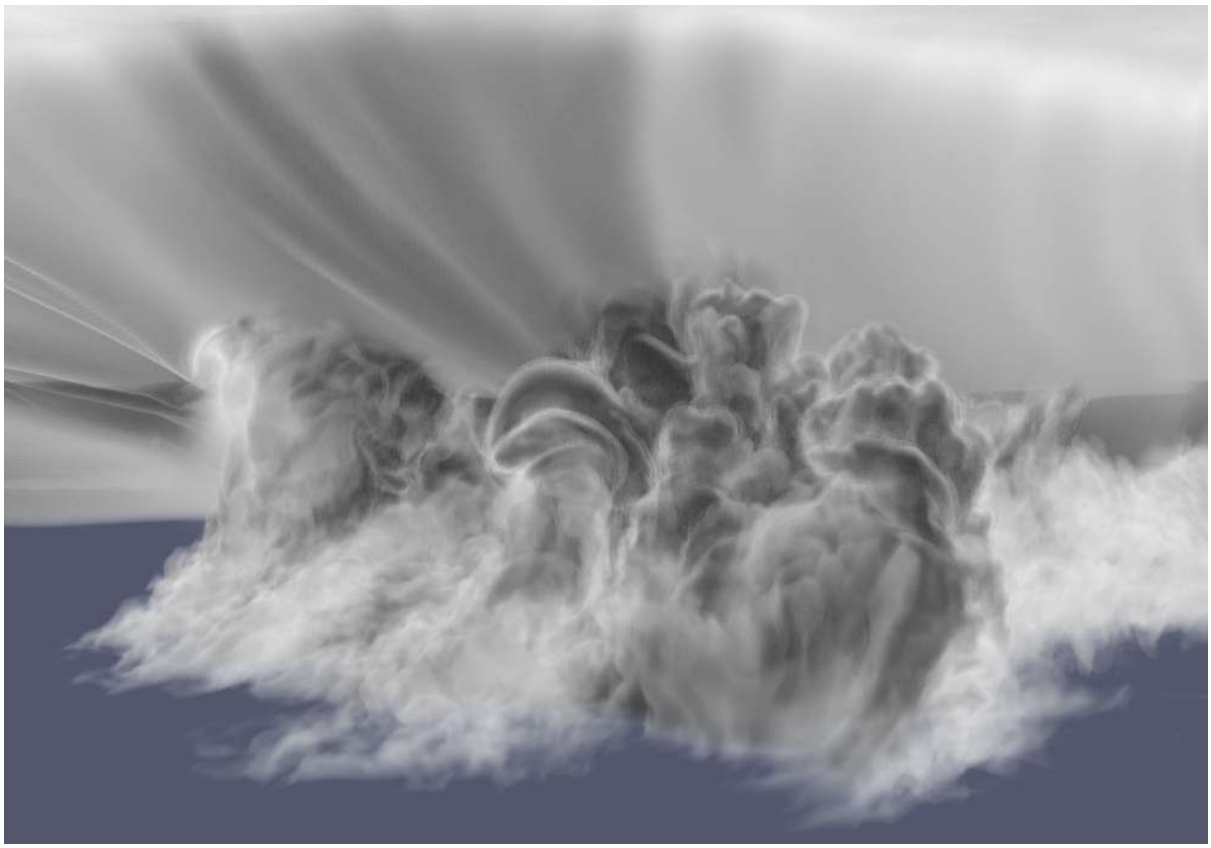

Fig. 4.15 - Base of an EF5 tornado cloud

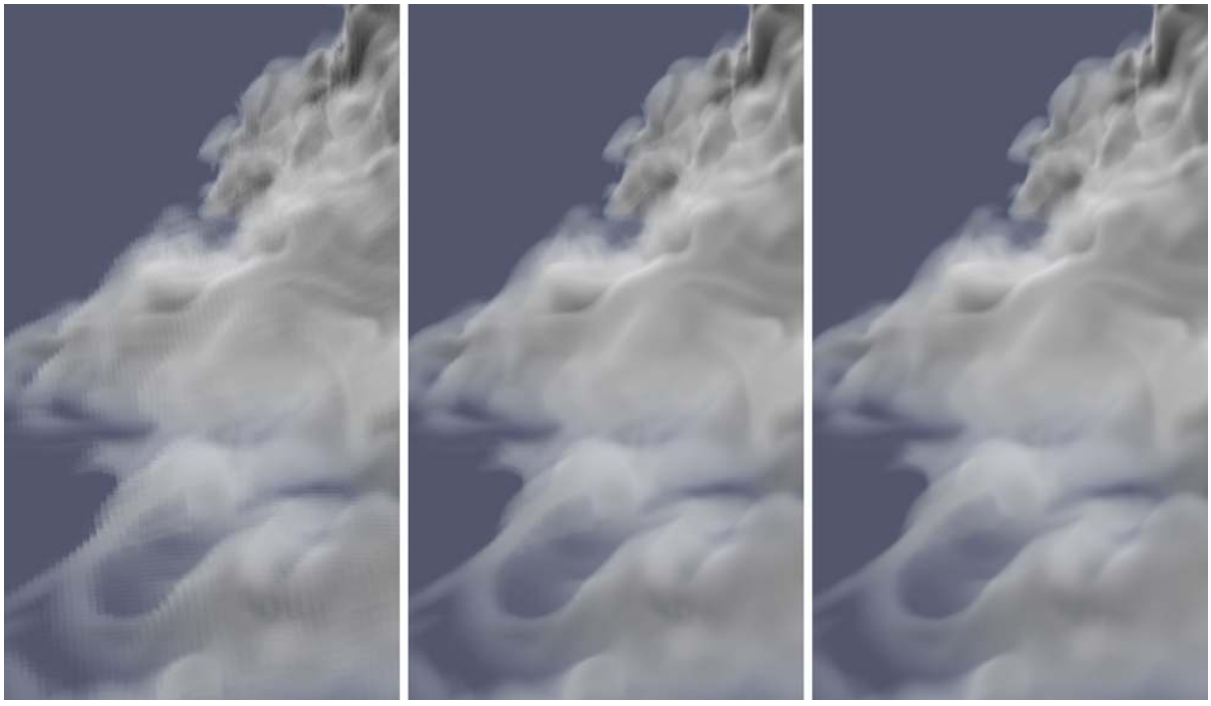

Fig. 4.16 - Nearest neighbor, trilinear and tricubic B-spline interpolation

4.7 Slice rendering

Volume slices can be enabled from the user interface. Currently they are limited to three axis-aligned slices with the ability to move individual slice positions.

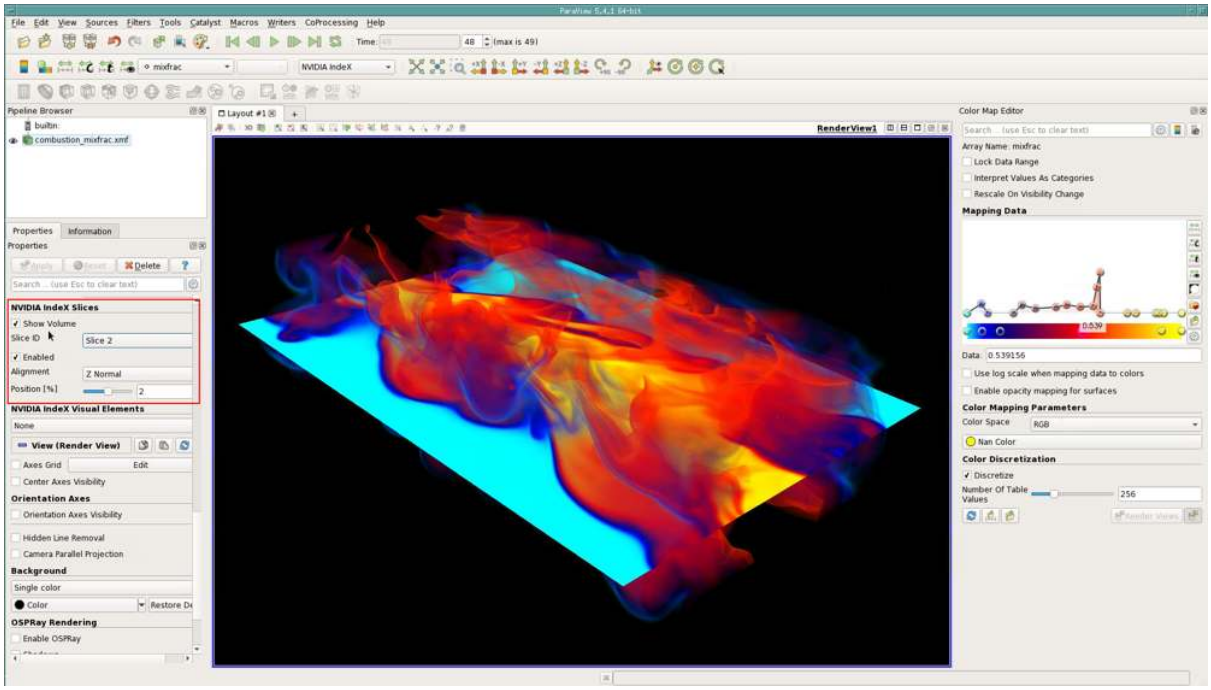

Fig. 4.17 - Slice rendering options in the Properties *panel. Dataset was made available by Dr. Jackqueline Chen at Sandia Laboratories through US Department of Energy's SciDAC Institute for Ultrascale Visualization.*

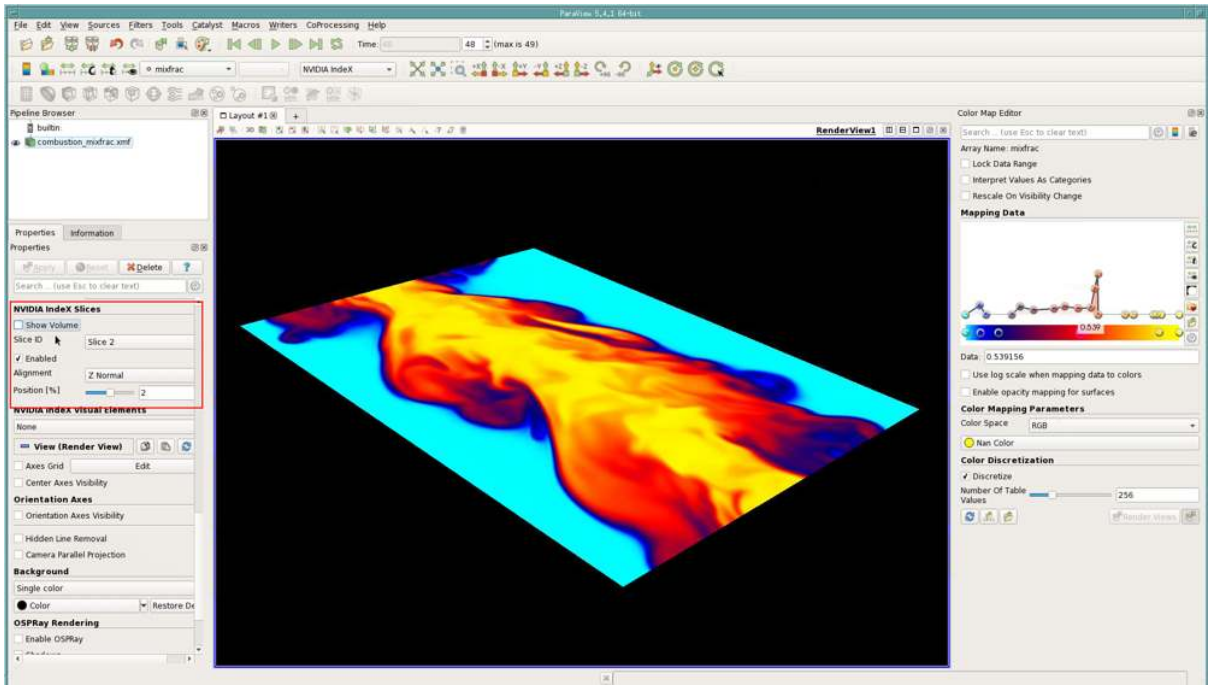

Fig. 4.18 - Slice rendering with volume disabled

4.8 NVIDIA IndeX visual elements

The *visual elements* feature of NVIDIA IndeX enables you to enhance the visual cues in the dataset. The plugin provides five visual presets, each preset having one or more parameters for finer control over the rendering.

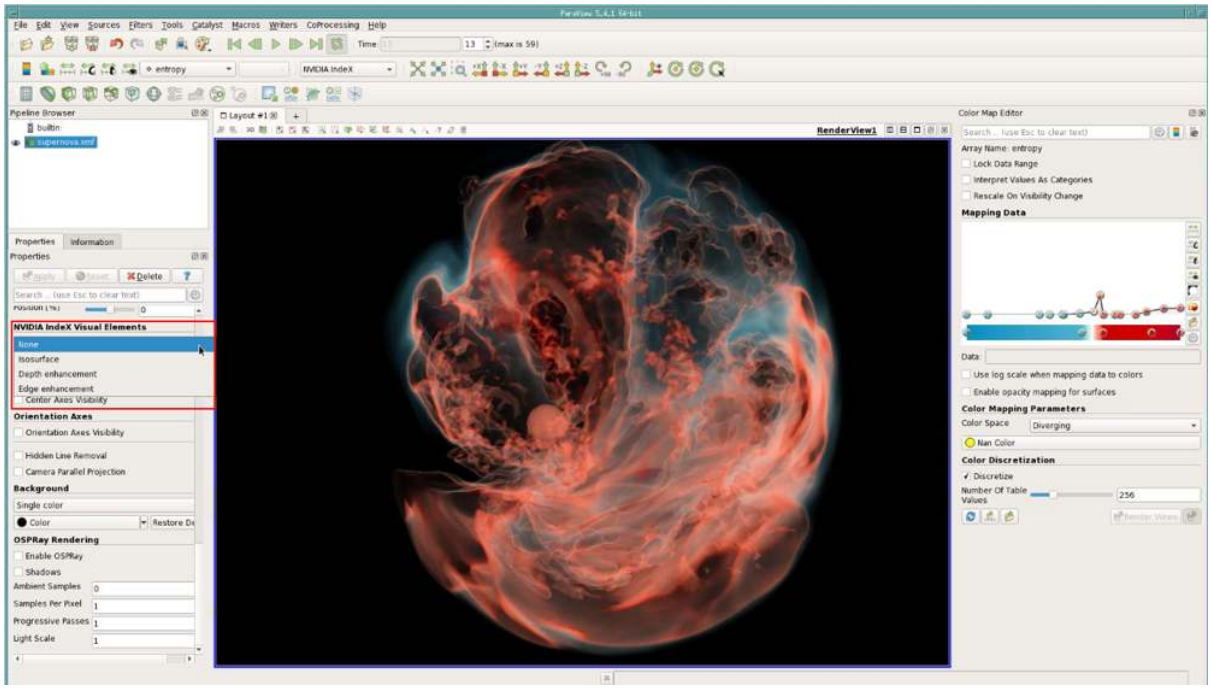

Fig. 4.19 - Supernova SASI visualized as a volume. Dataset courtesy of Dr. John Blondin at the North Carolina State University through US Department of Energy's SciDAC Institute for Ultrascale Visualization.

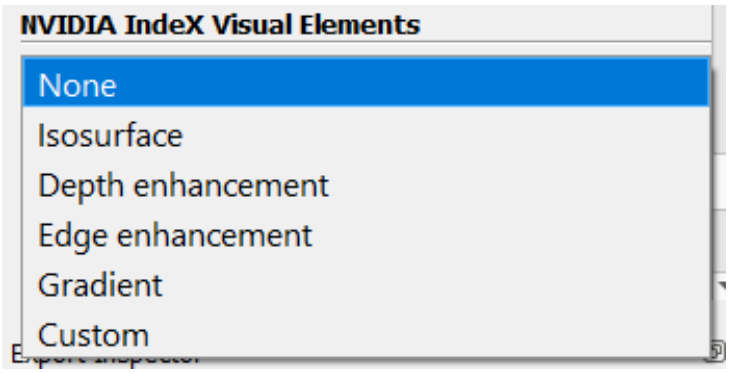

Fig. 4.20 - Visual element presets in the Properties *panel*

4.8.1 Isosurface preset

The *isosurface* preset allows you to extract the isosurface and volume contained inside the range `[iso-min, iso-max]` and to render both with different options for mapping sample values to colors.

Iso min/Iso max

Define the isosurface range, expressed in percentage of the scalar value range of the volume.

Fill mode

Defines the shading mode for the volume.

No Fill

The interior of the volume is not shown.

Single Color

The volume is shown with the iso-min value.

Colormap

The colormap is applied to the volume data.

Use shading

Enables/disables the Phong-Blinn lighing model for the iso-min/iso-max surfaces.

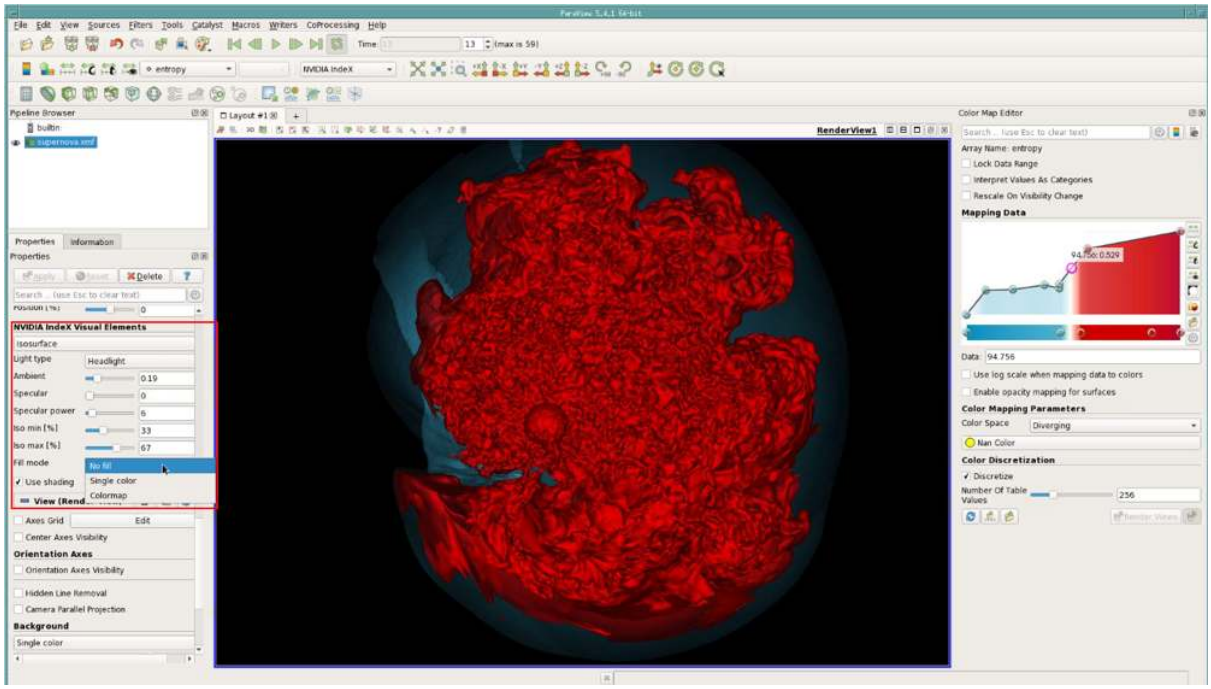

Fig. 4.21 - Supernova SASI visualized as an isosurface

4.8.2 Depth enhancement preset

The *depth enhancement* preset allows you to enhance the depth perception of a dataset by isolating features with high opacity values in the transfer function mapping. At each volume position, colormap alpha values are accumulated along a predefined line segment and samples in regions with a low opacity distribution are darkened.

Samples
: Number of samples taken along the line segment.

Gamma
: Controls contrast.

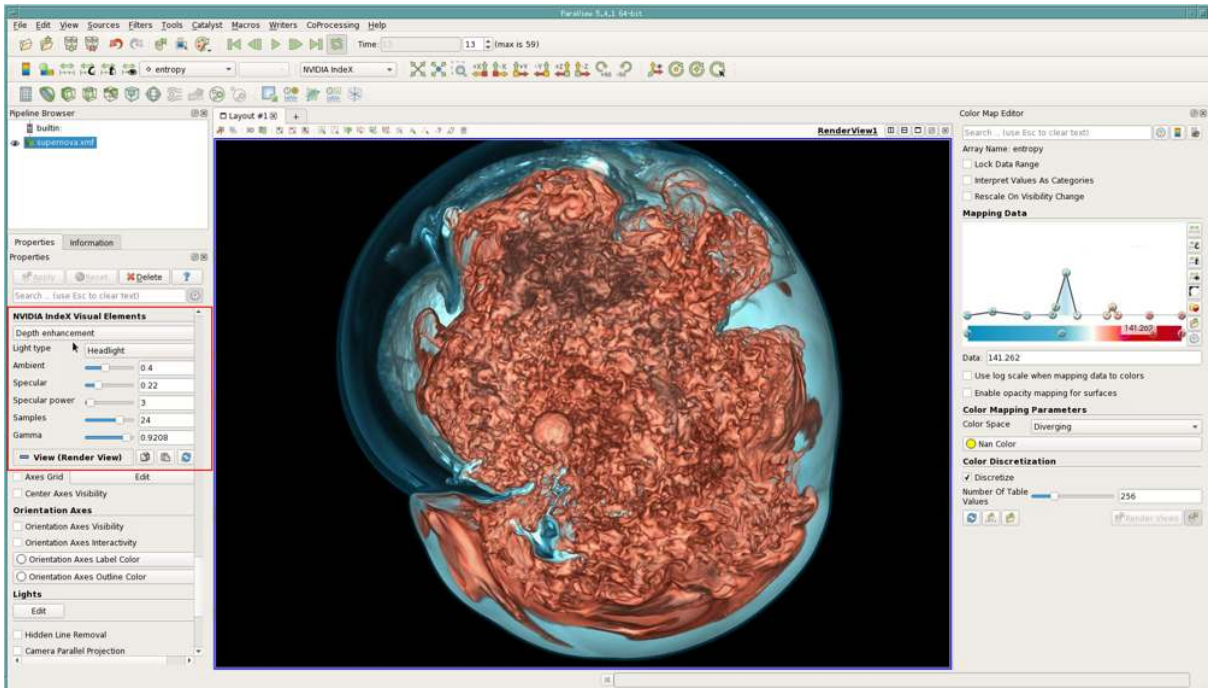

Fig. 4.22 - Supernova SASI visualized using the depth enhancement preset

4.8.3 Edge enhancement preset

The *edge enhancement* preset allows you to enhance the edges or "silhouettes" of a dataset based on variations of the transfer function. At each volume position, the gradients of the colormap alpha-channel are accumulated along a predefined line segment and samples that contain higher gradient magnitudes are darkened.

Edge Range
: The length of the line segment along which the gradient is calculated.

Samples
: The number of samples used to calculate the gradient along the line segment.

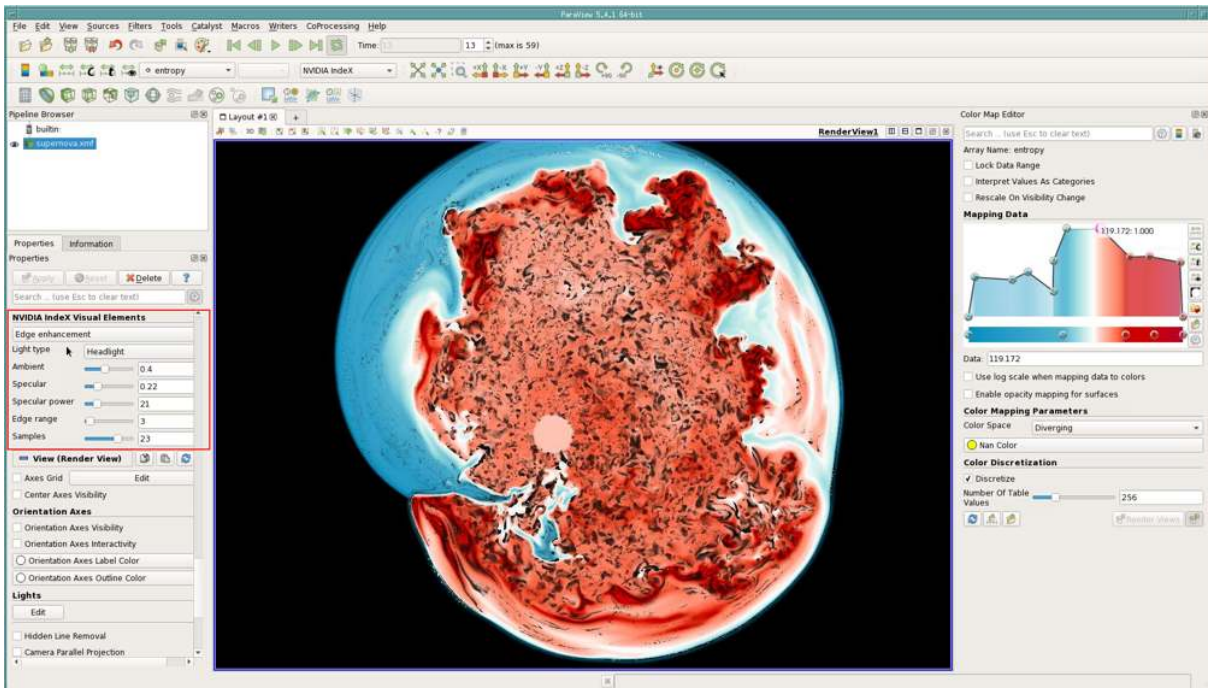

Fig. 4.23 - Supernova SASI visualized using the edge enhancement preset

4.8.4 Gradient preset

The *gradient* preset highlights features of greater variation in the dataset by calculating the gradient of the volume scalar field and using its magnitude to scale colormap samples.

Gradient Level

The gradient level or intensity as a percentage.

Gradient Scale

The maximum gradient level.

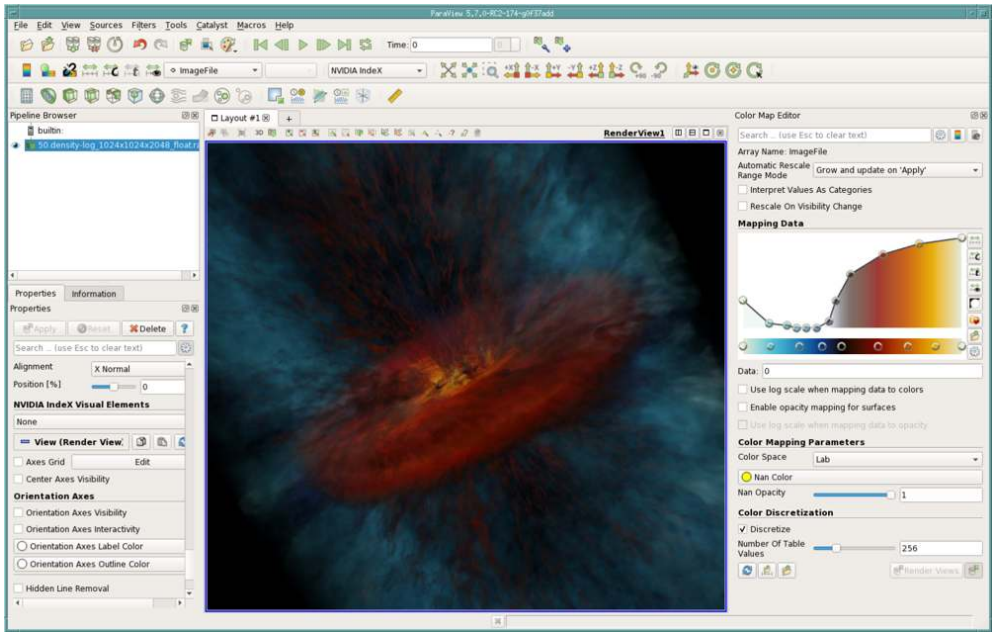

Fig. 4.24 - CHOLLA galactic outflow simulation visualized as a volume. Dataset courtesy of Evan E. Schneider (Princeton University) and Brant Robertson (University of California, Santa Cruz).

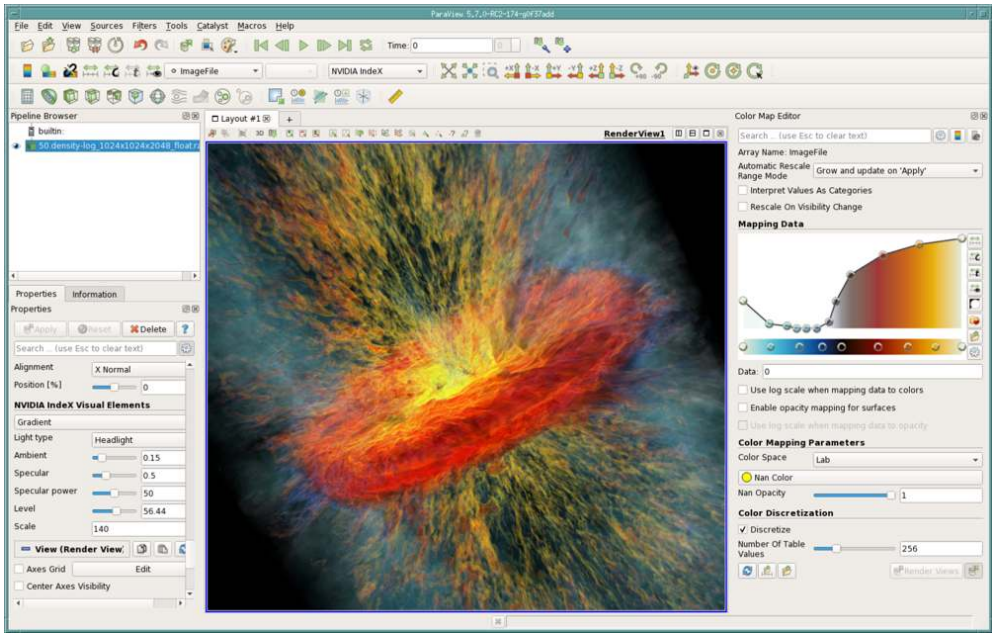

Fig. 4.25 - CHOLLA galactic outflow simulation visualized using the gradient preset.

4.8.5 Custom preset

The *custom* preset allows users to write their own volume rendering programs as XAC (NVIDIA IndeX Accelerated Compute) code. The XAC programs are small CUDA kernels, editable with your preferred text editor, that can be used to replace the default volume shading kernel executed inside NVIDIA IndeX. XAC programs are compiled at runtime, which means that you can load a program and edit it on the fly, with changes being applied immediately to the rendering. The *custom* preset provides the means to load an XAC program from a file, apply updates on the fly and comes with a list of predefined user parameters that can be accessed in your program.

For an overview of XAC, see NVIDIA IndeX XAC programming (page 34). A few volume program examples are distributed with the ParaView binary package (in `paraview-directory/kernels_nvidia_index`) and with the ParaView source distribution (in `paraview-sources/Plugins/pvNVIDIAIndeX/kernel_programs`).

Kernel
: The XAC volume program to be loaded from a text file.

Update Kernel
: Recompiles the XAC program, to apply live changes made to the XAC program file, for example with an external text editor.

pfloat 1-4
: Four general purpose floating point parameters that can be bound to the XAC volume program.

pint 1-4
: Four general purpose integer parameters that can be bound to the XAC volume program.

```
NV_IDX_XAC_VERSION_1_0

// DON'T CHANGE THIS STRUCT !!!
// It maps the GUI parameters from the Custom Visual Element GUI to this kernel.
// floats maps {pfloat1, pfloat2, pfloat3, pfloat4} GUI Parameters.
// ints maps {pint1, pint2, pint3, pint4} GUI Parameters.
struct Custom_params
{
    float4 floats;  // floats array
    int4 ints;      // ints array
};

class Volume_sample_program
{
    ...

    const Custom_params*  m_custom_params;

public:
    NV_IDX_DEVICE_INLINE_MEMBER

    void initialize()
    {
        // maps Custom Visual Element GUI parameters to this kernel.
      m_custom_params = state.bind_parameter_buffer<Custom_params>(0);
    }

    NV_IDX_DEVICE_INLINE_MEMBER
    int execute(
        const Sample_info_self& sample_info,
              Sample_output&    sample_output)
    {
        // map "pint 1" GUI parameter as use shading option (true/false)
        // enable/disable shading effect
        const bool use_shading = (m_custom_params->ints.x >= 1);

        // map "pfloat 1" GUI parameter as min shading alpha
        // min alpha threshold for shading.
        float min_shade_alpha = m_custom_params->floats.x/100.f;
        if(min_shade_alpha < 0.f)
          min_shade_alpha = 0.f;
```

Fig. 4.26 - Custom preset: Floating point and integer parameters binding in an XAC program.

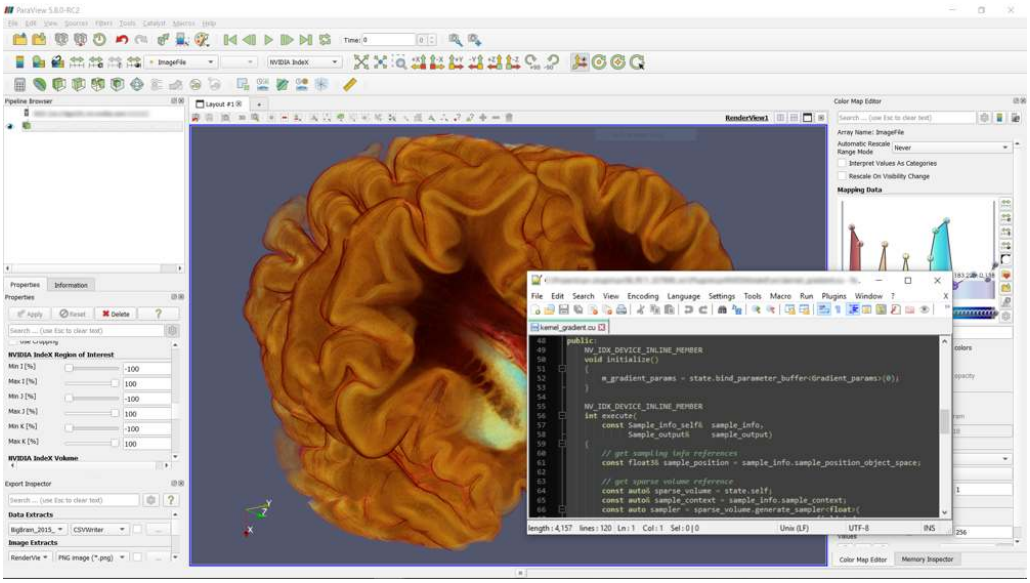

Fig. 4.27 - BigBrain Project dataset visualized with the custom *preset. Dataset courtesy of Prof. Dr. med. Katrin Amunts and the Structural and Functional Organization of the Brain lab at the Institute of Neuroscience and Medicine, Research Centre Jülich.*

4.9 Time series animation

The time series animation feature allows you to render timesteps of a dataset in real-time. You can navigate, change colormaps and perform data operations as you would with a static dataset. In order to have a smooth playback, please enable *Cache Geometry For Animation* in the ParaView *Settings* dialog. The animation will be slower in the first cycle, but once all the timesteps are loaded, the playback should be smooth, as long as there is enough system memory available.

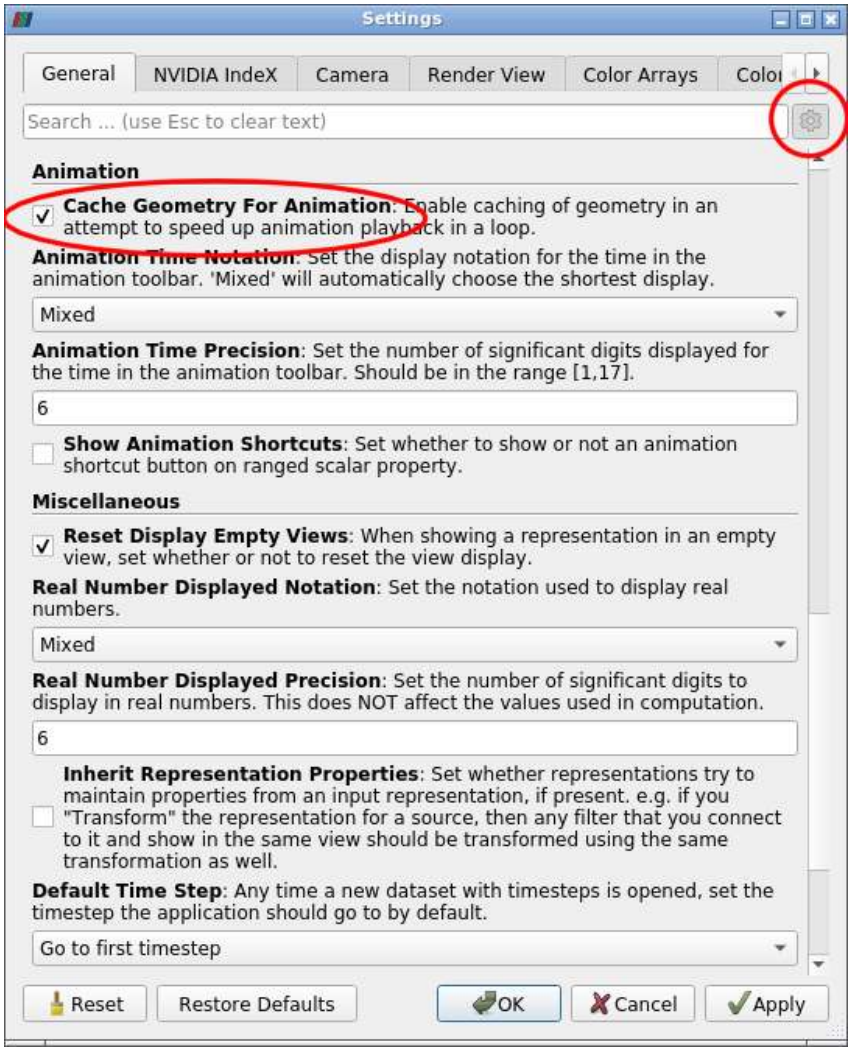

Fig. 4.28 - Enable Cache Geometry for Animation (*Toggle advanced properties* in the top right corner must be enabled)

4.10 Catalyst and in-situ visualization

NVIDIA IndeX supports in-situ visualization, for running a simulation and visualizing it in real-time without writing any data to disk. Catalyst[1] is the co-processing library that enables orchestration of simulation, analysis and visualization tasks together with VTK and ParaView. Catalyst can also be used to set up NVIDIA IndeX and ParaView to do live visualization of your simulation. Please visit the Catalyst website to learn how to write scripts to integrate your simulation and enable in-situ visualization.

1. `https://www.paraview.org/in-situ/`

As an example, a simple *Wavelet* source can be used to illustrate the Catalyst integration with NVIDIA IndeX rendering. Make sure you have compiled ParaView with Catalyst support before trying to do the live visualization.

You can start with 50 iterations of a *Wavelet* data source in a single process by running the following command. The scripts `CatalystWaveletDriver.py` and `CatalystWaveletCoprocessing.py` are located in the directory `Clients/ParaView/Testing/XML/` in the ParaView source repository.

```
mpirun -np 1 pvbatch -sym CatalystWaveletDriver.py ↦
   CatalystWaveletCoprocessing.py 50
```

Next, start the ParaView client and connect to the port where Catalyst is running with [Catalyst ▸ Connect] in the menu

Once ParaView connects to Catalyst, enable *Input* and click *Extract input* from the *Pipeline Browser*. Once the input is extracted, you can switch the rendering to the NVIDIA IndeX representation.

Fig. 4.29 - Enable input *and* extract input *to visualize*

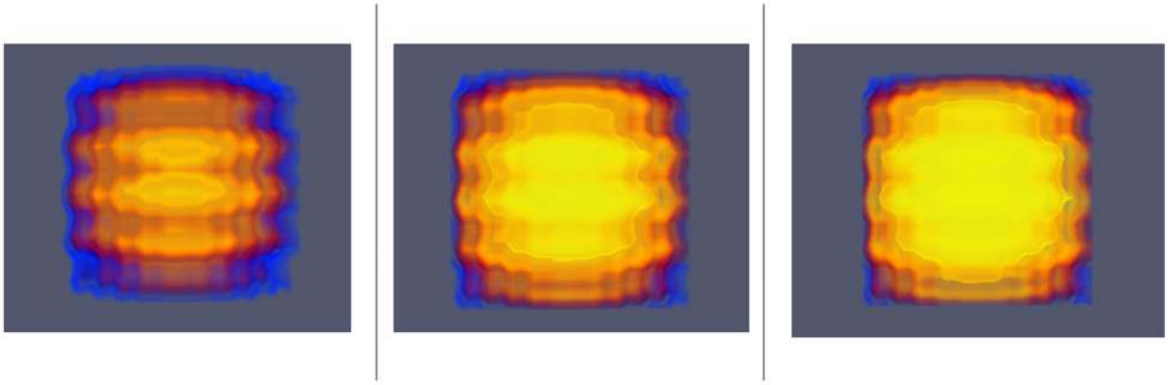

Fig. 4.30 - Wavelet example shown at different iterations with Catalyst

4.11 Mixing ParaView primitives

One of the unique features of the plugin is the ability to mix volume rendering from NVIDIA IndeX together with other ParaView primitives such as glyphs, streamlines and surfaces.

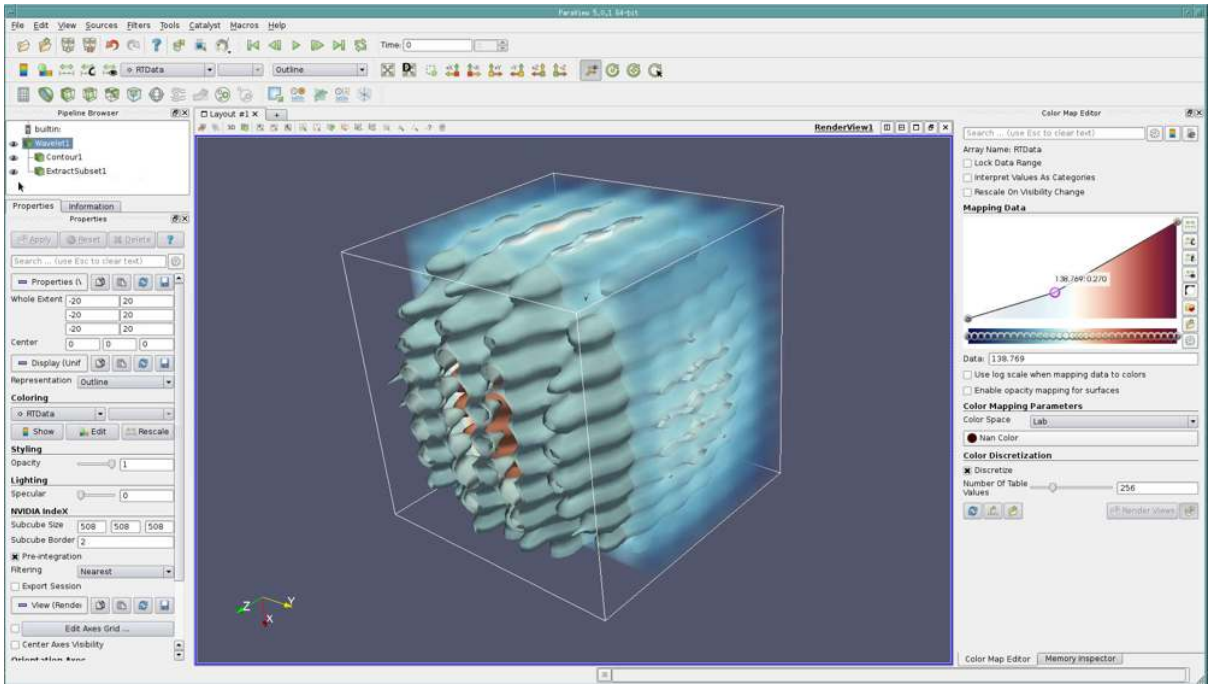

Fig. 4.31 - Wavelet data rendered as a surface by ParaView and as a volume by NVIDIA IndeX

4.12 Known limitation

Regular volume grids

When running with MPI on multiple `pvserver` ranks, datasets in `.vtk` format won't get distributed to multiple ranks by ParaView, and only a single GPU will be utilized. The NVIDIA IndeX for ParaView Plugin will print a warning if this case is detected. Please use a different data format such as `.pvti` that supports distributed data. Since this issue is specific to ParaView, please contact Kitware for additional details.

Unstructured grids

Datasets containing degenerate faces may result in incorrect renderings or cause ParaView to fail. The NVIDIA IndeX plugin will try to resolve all invalid faces automatically.

General

The Windows version of the NVIDIA IndeX plugin is restricted to run on a single host only, i.e., cluster rendering is not supported on Windows platforms.

When loading an older state file with both volumetric and geometry data without the NVIDIA IndeX representation saved in it, the first frame will show only volumetric data. After interacting with the scene, the subsequent frames will be correct again with both geometry and volume data visible.

5 NVIDIA IndeX XAC programming

This section covers aspects of volume rendering using the *NVIDIA IndeX Accelerated Compute (XAC)* technology. It dives into the structure of a basic CUDA-based XAC volume sample program, how to access volumes and how to use colormaps a transfer functions.

5.1 XAC purpose and program structure

NVIDIA IndeX Accelerated Compute (XAC) defines an infrastructure to work with interactive CUDA-based *sampling programs* (also called *kernels* or *shaders*) for rendering and compute that are compiled at runtime into the NVIDIA IndeX rendering environment. These programs are executed natively on the GPU and can therefore utilize the full performance of the GPU hardware.

There are two fundamental types of XAC programs: *volume* sample programs and *surface* sample programs. Both program types have a predefined structure, receive specific input data, can set specific output and can be specialized towards different types of scene elements.

5.2 XAC volume sample programs

To understand the purpose of XAC programs, it is helpful to look at a few internal aspects of how NVIDIA IndeX renders data.

During rendering, *rays* are generated for each pixel and intersected with elements in the scene. When such a view ray hits a volume, the *volume sample program* is executed at each sample position along the ray and produces a four component RGBA color vector (x: red, y: green, z: blue, w: alpha/opacity), which is then accumulated and blended into the final color of the pixel (compositing).

A standard XAC volume sample program consists of three parts:

Header and declaration
: Defines global variables and declarations

Initialization function
: Initialize parameters of the program run (for example, buffer bindings)

Execution function
: Computes the output color

5.2.1 Example program outline

A basic XAC program will look like this:

Listing 5.1

```
NV_IDX_XAC_VERSION_1_0   Current XAC version string (optional)

using namespace nv::index;
using namespace nv::index::xac;   Include the default namespace (contains library and helper functions)

class Volume_sample_program   Define class containing the XAC program implementation
{
    NV_IDX_VOLUME_SAMPLE_PROGRAM   Program type declaration (required)

public:
    NV_IDX_DEVICE_INLINE_MEMBER
    void initialize()   Initialization function
    {
       // initial setup...
    }
    NV_IDX_DEVICE_INLINE_MEMBER
    int execute(   Main rendering function (required)
       const Sample_info_self& input, Sample_output& out)
    {
        float4 color = make_float4(1.f);   Do some color computations here...

        out.set_color(color);   Store the output color
       return NV_IDX_PROG_OK;
    }
};
```

5.3 Sampling a volume and map to a color

By default, the XAC program receives information about the scene and the scene element for which the program is executed. The variable `state.self` contains a reference to the scene element which called the program (in this case the target volume). Note that this variable can have different types and contents depending on the type of the scene element.

The parameter `Sample_info_self& input` contains references to additional information, such as:

`sample_position` (*float3*)
  Current sample position (in the volume grid)

`scene_position` (*float3*)
  Current position in the scene

`ray_origin` (*float3*) `ray_direction` (*float3*) `ray_t` (*float*)
  Properties of the current view ray

The full XAC API is documented in the NVIDIA IndeX Programmer's Manual.[1]

To sample a data value from a volume and map it to a color using the transfer function (or colormap), the following lines have to be added to the `execute()` function:

Listing 5.2

```
NV_IDX_DEVICE_INLINE_MEMBER
int execute(const Sample_info_self& input, Sample_output& output)
{
    const float3& sample_position =
        input.sample_position_object_space;          // Get current sample position

    const Sparse_volume& volume = state.self;
    const Colormap colormap = volume.get_colormap(); // Get references to the volume and its colormap

    const auto& sample_context = input.sample_context;
    const auto sampler =
        volume.generate_sampler<float>(sample_context); // Get a sample context and generate a volume sampler

    float sample_value =
        sampler.fetch_sample(sample_position);       // Sample the volume at the current position

    float4 sample_color = colormap.lookup(sample_value); // Apply the colormap

    output.set_color(sample_color);                  // Store the output color (RGBA)

    return NV_IDX_PROG_OK;
}
```

5.4 Using XAC library functions

Within each XAC program you have access to several helper functions. This includes:

- The CUDA math library[2]
- Basic linear algebra functions (for example, point, vector and matrix operations)
- XAC library convenience functionality (`nv::index::xaclib`)
- NVIDIA IndeX reference types (`nv::index::xac`)

The XAC library holds a set of additional functions, which provide convenient access to typical operations, such as color transformations and volume gradient computation. You can access those functions by calling them from the `nv::index::xaclib` namespace.

For example, to use a simple color gamma operation, you can add the following lines to your program:

1. `https://raytracing-docs.nvidia.com/nvindex/index.html`
2. `https://docs.nvidia.com/cuda/cuda-math-api/index.html`

Listing 5.3

```
float4 sample_color =
    colormap.lookup(sample_value);
```

Initialize the sample color using the volume mapping from before

```
float screen_gamma = 0.7f;
sample_color =
    xaclib::gamma_correct(sample_color, screen_gamma);
```

Apply gamma function to the color

```
output.set_color(sample_color);
```

Store the output color

5.5 Adding basic volume shading

The basic volume sample program can be extended to integrate advanced visualization: In this case, we are adding a simple local lighting model (simple Phong lighting based on a headlight) that helps to emphasize isosurface directions. To do this, we need to compute a isosurface normal, which can be derived from the *volume gradient*, for which we use an XAC library function.

To integrate volume shading into the XAC program, the following lines have to be added:

Listing 5.4

```
float4 sample_color =
    colormap.lookup(sample_value);
```

Apply the colormap, creating an RGBA color

```
const float3 gradient =
    xaclib::volume_gradient(
        volume, sample_position);
```

Approximate the volume gradient based on finite differences

```
const float3 normal = -normalize(gradient);
const float3 view_dir = input.ray_direction;
```

Get the isosurface normal (in outward direction) and view direction

```
const float4 specular_color =
    make_float4(1.f, 1.f, 1.f, 1.f);
```

Define specular (reflective highlight) color

```
sample_color =
    xaclib::headlight_shading(
        state.scene, normal, view_dir,
        sample_color, specular_color);
```

Apply built-in headlight shading

```
output.set_color(sample_color);
```

Store the output color

5.6 Using CUDA parameter buffers

Changing fixed parameters in the XAC code requires recompiling the program. To change parameters interactively without recompilation, XAC allows to pass and bind custom parameter buffers to the program.

The NVIDIA IndeX plugin provides pre-defined parameter buffers that can be edited through the ParaVier user interface.

Listing 5.5

```
struct Custom_params
{
    float4 floats;  // floats array
    int4 ints;      // ints array
};
```

Custom parameters set in the ParaView user interface

```
const Custom_params*  m_custom_params;

NV_IDX_DEVICE_INLINE_MEMBER
void initialize()
{
     m_custom_params
        = state.bind_parameter_buffer<Custom_params>(0);
}
```

Bind input parameter buffer from parameter slot 0

```
NV_IDX_DEVICE_INLINE_MEMBER
int execute(
    const Sample_info_self& sample_info,
          Sample_output&    sample_output)
{
    const float input_value = m_custom_params.floats[0];
```

Retrieves input parameter

```
    const float red = input_value * 0.6f;
    const float green = 0.5f - input_value * 0.4f;
    const float blue = input_value * 0.2f;
    const float alpha = input_value + 0.1f;
```

Use input parameter to compute some values

```
    float4 modified_color =
       make_float4(red, green, blue, alpha);
    modified_color =
       xaclib::clamp(modified_color, 0.f, 1.f);
```

Color computation

```
    sample_output.set_color(modified_color);
```

Store the output color

```
    return NV_IDX_PROG_OK;
}
```

6 Frequently asked questions

Q: Do I need to install the CUDA SDK or any other libraries for using the plugin?

A: There is no need to install the CUDA SDK separately as the plugin package is bundled with all the required libraries. However, an appropriate NVIDIA display driver for your GPU needs to be installed.

Q: When I load the plugin from ParaView's [Tools ▸ Manage Plugins] window, *pvNVIDIAIndeX* does not show up as loaded.

A: Make sure you have no errors in ParaView's console or in the terminal where you started ParaView from. Any error messages will give you additional information about what the issue might be.

Q: The plugin is loaded successfully without any errors but *NVIDIA IndeX* does not show up in ParaView's *Representation* drop-down list.

A: Make sure you have loaded a structured or unstructured volume grid dataset and it is selected in ParaView's pipeline browser. ParaView chooses the available representations based on the input data format.

Q: There is an error saying "Failed loading NVIDIA IndeX library" and the viewport is empty.

A: This error message is usually printed when the NVIDIA IndeX libraries (`.so` on Linux and `.dll` on Windows) are not found. Make sure that `libnvindex.so` and `libdice.so` can be found by ParaView. Either copy all the libraries to ParaView's library directories or set the environment variable `LD_LIBRARY_PATH` (or `PATH` on Windows) accordingly. Refer to "Building ParaView and the NVIDIA IndeX plugin" (page 2) for more information.

Q: Viewport is blank when I choose *NVIDIA IndeX* as a representation.

A: Select an appropriate *Scalar Array* for the dataset instead of *Solid Color*.

Q: Viewport is blank when I choose *NVIDIA IndeX* as a representation with a *Scalar Array* and not with *Solid Color*

A: This is most likely because of an old NVIDIA display driver. Update your display drivers to the recommended versions. If this happens in client/server mode, make sure that IceT compositing is disabled, as described in Client/server mode with MPI (page 7).

Q: Why does my rendering look down-sampled when I interact?

A: NVIDIA IndeX does not down-sample the data and renders at full resolution. By default, ParaView will optimize for high latency networks and enables compression and level of detail. You can disable this under [Edit ▸ Settings] and turn off *LOD Resolution*, *Image Reduction Factor* and *Image Compression*.

Q: Can I render multiple volumes at once in the same scene in ParaView?

A: While the NVIDIA IndeX library itself supports multi-volume rendering, the ParaView plugin does not yet have this feature integrated so you can only render one volume at a given time.

Q: Can I use the NVIDIA IndeX library in my own application without ParaView?

A: Yes, an NVIDIA IndeX SDK is available. Please contact NVIDIA for more details: `nvidia-index@nvidia.com`

7 Resources

- NVIDIA IndeX for ParaView plugin website[1]
- NVIDIA IndeX website[2]
- ParaView binaries and source code download[3]
- ParaView documentation[4]
- ParaView support forum[5]
- Contact email: `paraview-plugin-support@nvidia.com`

1. `https://www.nvidia.com/en-us/data-center/index-paraview-plugin`
2. `https://developer.nvidia.com/index`
3. `https://www.paraview.org/download/`
4. `https://www.paraview.org/resources/`
5. `https://discourse.paraview.org`


NVIDIA