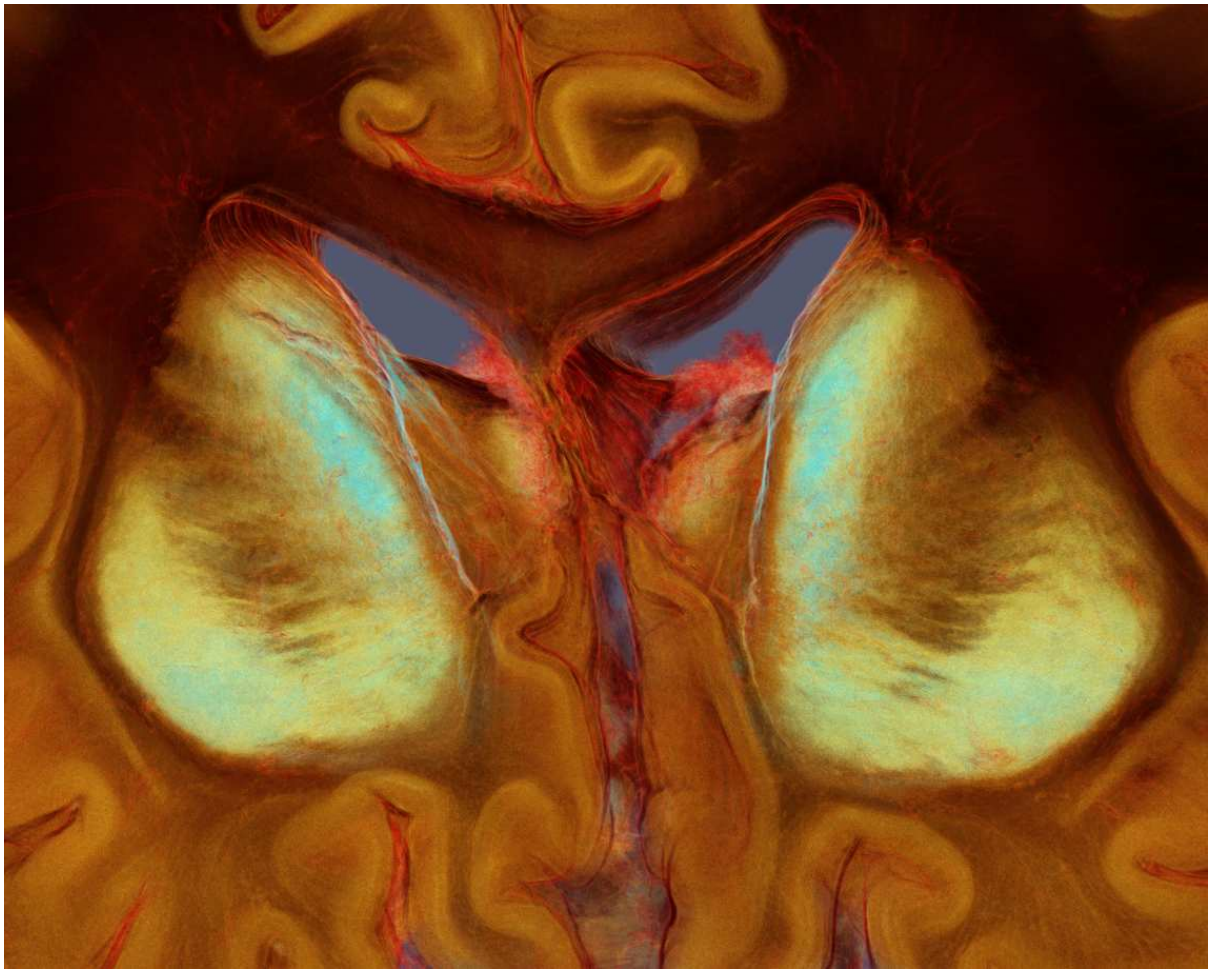
NVIDIA IndeX for ParaView Plugin

User's Guide

7 February 2020
Version 2.4



NVIDIA

NVIDIA IndeX for ParaView Plugin – User's Guide

Cover page rendering: BigBrain Project, dataset courtesy by Prof. Dr. med. Katrin Amunts and the Structural and Functional Organization of the Brain Lab at the Institute of Neuroscience and Medicine, Research Centre Juelich.

Document build number 328607

Contents

1 Introduction

The NVIDIA® IndeX™ for ParaView® Plugin enables large-scale and high-quality volume data visualization capabilities of the NVIDIA IndeX library inside Kitware's ParaView.

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

If you haven't downloaded the plugin yet, you can do so from this URL:

`http://www.nvidia.com/object/index-paraview-plugin.html`

1.1 Licensing

The NVIDIA IndeX for ParaView plugin comes with a free license that enables exploiting the capabilities of a single GPU. If you aim to use plugin on a cluster of multiple hosts and/or with multiple NVIDIA GPUs, then please contact us for appropriate licensing via email to `paraview-plugin-support@nvidia.com`. Users will be notified via email whenever a new version of the plugin is released.

2 Installation

NVIDIA IndeX for ParaView plugin is delivered with ParaView v5.8.0 and supports both Linux and Windows x86-64 platforms. Follow the installation instructions specific to your platform and install ParaView.

If you have installed ParaView v5.8.0 or later from binaries, the plugin is already included and you can continue to section 3.2, "Loading the plugin" (page 3).

2.1 Building the plugin

Please follow these steps to build the plugin matching your build environment.

1. You can download the ParaView source code from ParaView website[1]
2. Run `cmake` on your ParaView source tree and the plugin sources will be added to the ParaView plugins list to be compiled. Make sure you have the cmake option set, enabled by default in the plugin source code.

```
mkdir paraview_build
mkdir paraview_install
cd paraview_build
```

```
ccmake ../paraview-source-root
```

```
PARAVIEW_PLUGIN_ENABLE_pvNVIDIAIndeX=ON
```

3. Run `make` or `make install` from your ParaView source tree. The plugin will be compiled together with ParaView.

```
make paraview-binary-root
```

4. The plugin requires the NVIDIA IndeX libraries package. The package is not distributed with ParaView sources but can be downloaded from the ParaView dependency repository: `https://www.paraview.org/files/dependencies/`

   Linux
   : Download `nvidia-index-libs-2.4.<YYYYMMDD>-linux.tar.bz2` and uncompress it to a folder of your choice. Update your `LD_LIBRARY_PATH` environment with the library path of your newly created folder: *new-folder*`/lib`

   Windows
   : Download `nvidia-index-libs-2.4.<YYYYMMDD>-windows-x64.tar.bz2` and uncompress it to a folder of your choice. Update your `PATH` environment with the library path of your newly created folder: *new-folder*`\lib`

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

CUDA 10.1 backward compability

The default NVIDIA IndeX libraries distributed with ParaView 5.8.x binaries and also availables in the ParaView dependency repository require CUDA 10.2 driver (or newer) installed on your system. For systems where only CUDA 10.1 driver is available, we provide custom NVIDIA IndeX libraries packages that can be used instead of the default ones. In case ParaView 5.8.x was installed from binaries, please download, uncompress and replace the libraries contained in the <*ParaView-install*>`\lib` folder (Linux) or <*ParaView-install*>`/bin` folder (Windows). In case ParaView 5.8.x was built from sources, please follow the same procedure as for the previous 'Linux', 'Windows' sections. The NVIDIA IndeX libraries package for CUDA 10.1 drivers can be downloaded from the ParaView dependency repository:
`nvidia-index-libs-2.4.<YYYYMMDD>-<platform>_CUDA_101.tar.bz2`

For additional information about compiling ParaView please refer to the instructions on the ParaView website.[2]

2.2 Location of the plugin

In a ParaView installation, the plugin binary is located in the following directories:

Linux: *your-paraview-installation-directory*`/lib/paraview-`*version*`/plugins/pvNVIDIAIndeX/`
Windows: *your-paraview-installation-directory*`\bin\plugins\pvNVIDIAIndeX\`

The plugin binary will automatically show up in the [Tools ▸ Manage Plugins] menu.

2.3 Loading the plugin

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

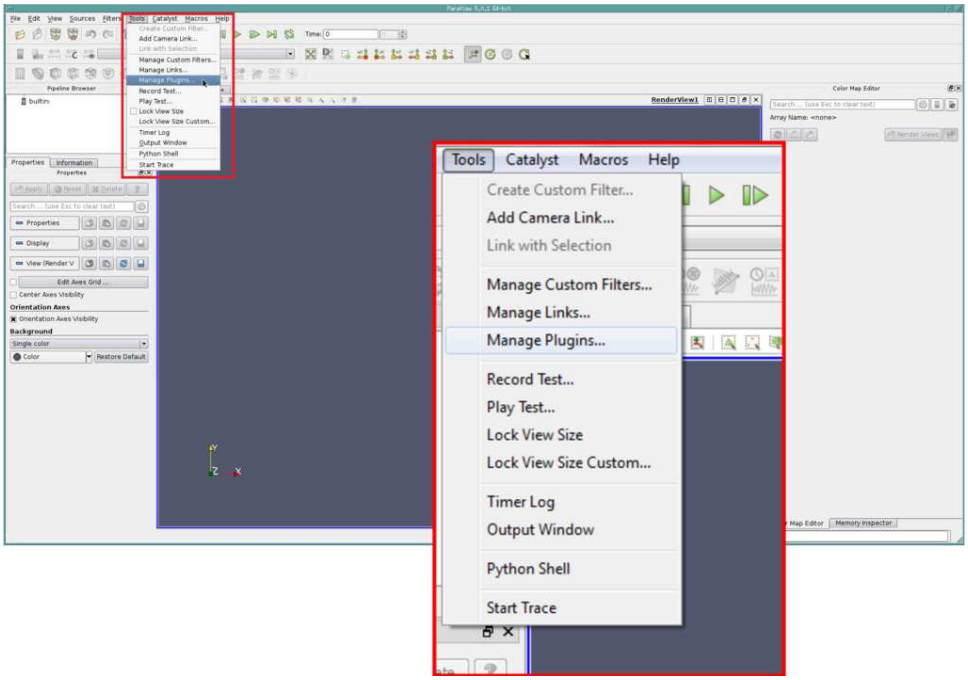


Fig. 2.1 – [Tools ▸ Manage Plugins] from ParaView menu

2. `https://www.paraview.org/Wiki/ParaView:Build_And_Install`

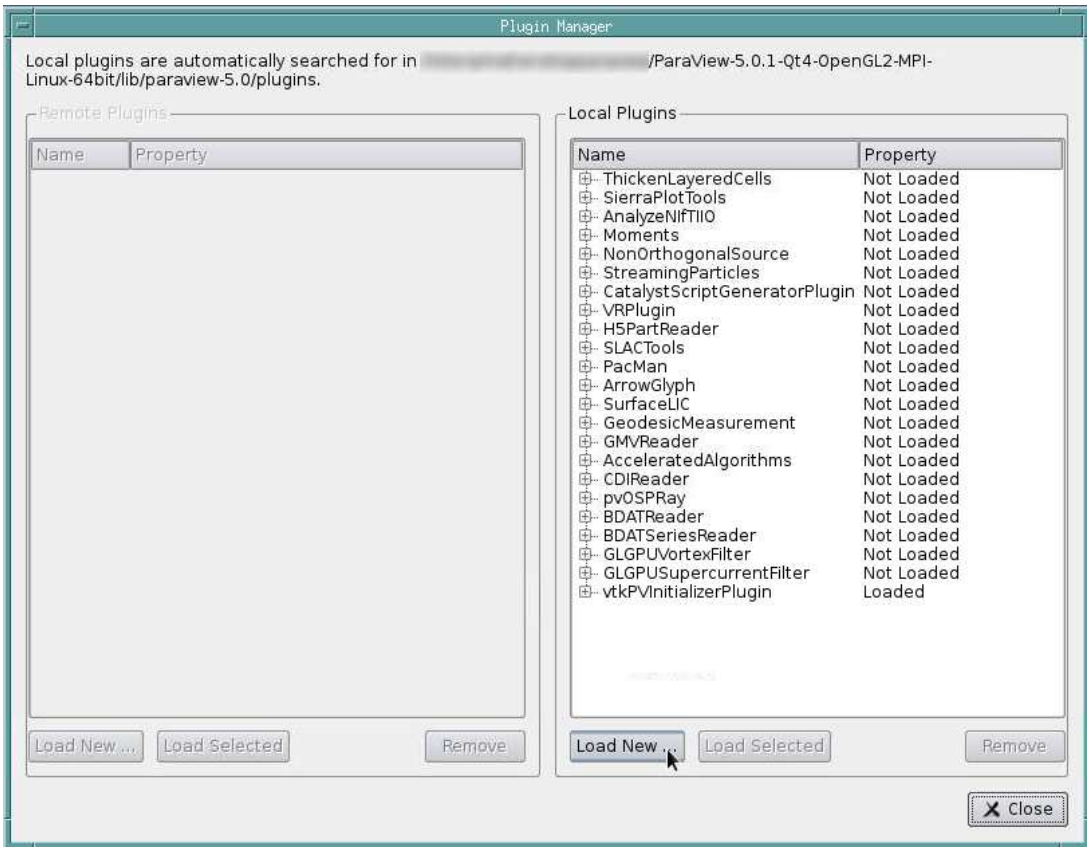


Fig. 2.2 – Click Load New option and locate the plugin

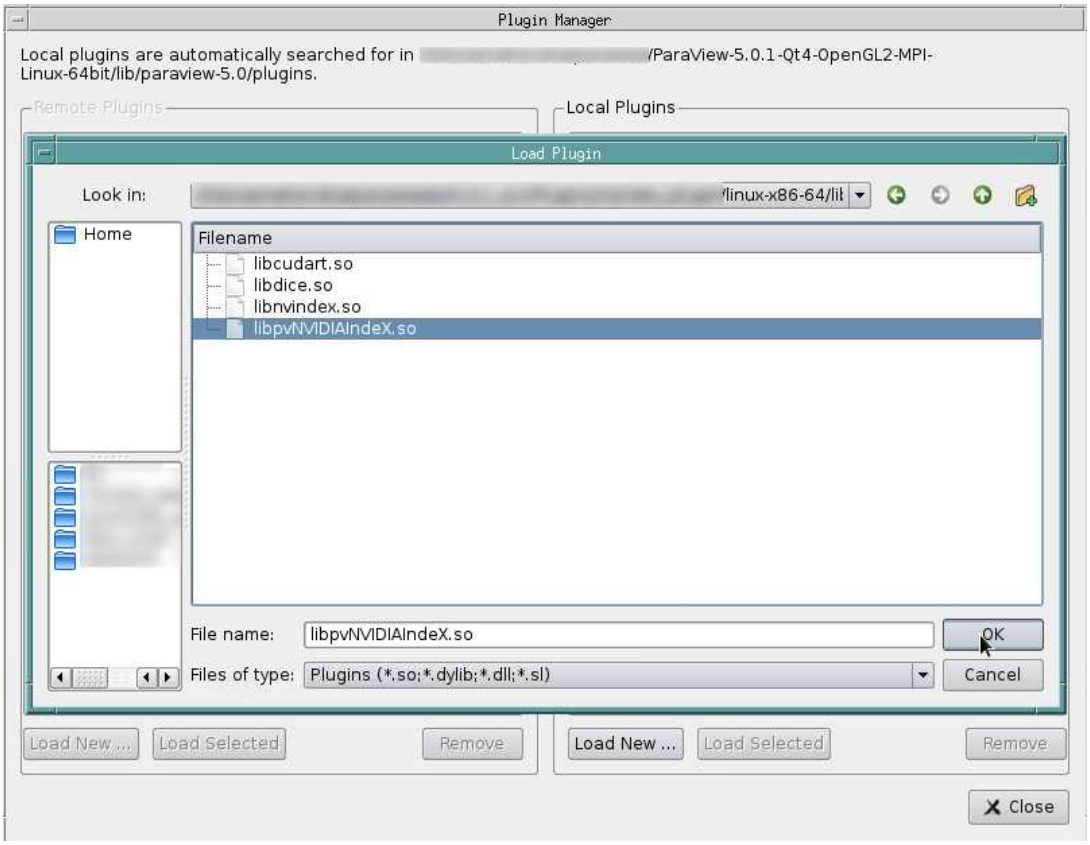


Fig. 2.3 – Load NVIDIA IndeX for ParaView plugin

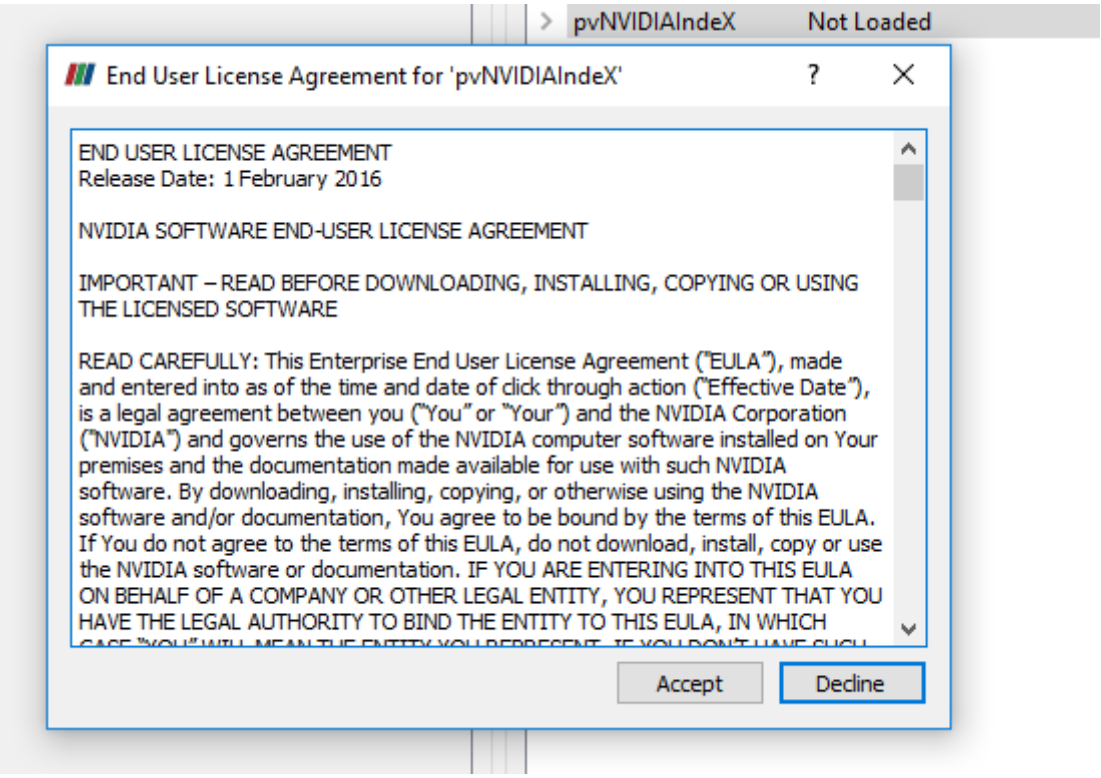


Fig. 2.4 – Read the EULA and click Accept

Once the plugin is loaded, the name of the plugin shows up in the [Tools ▸ Manage Plugins] dialog box with status changed as *loaded*. Make sure there no errors in the terminal or on ParaView's console. Also, when using the plugin in client-server mode, be sure to load the plugin in both client and server side of the [Tools ▸ Manage Plugins] window.

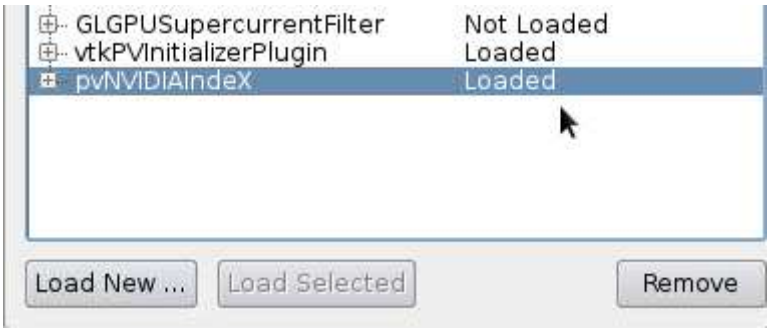


Fig. 2.5 – Status is shown as Loaded when no errors

3 Getting started

This section will describe instructions on 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 and load the plugin as described in section 2.

3.2 Client-server mode on a single GPU

To run the plugin in client-server mode, start `pvserver` with `mpirun` as shown below.

```
mpirun -bynode -np 1 pvserver -display :0.0 --force-offscreen-rendering
```

Once `pvserver` process is launched, run the ParaView client and connect to the server where pvservers are running by using [File ▸ Connect ▸ Add Servers] option in ParaView's menubar. Typically the server address is printed out in the console where mpirun was executed, once the client-server is connected, console will update the status with "Client connected" message. Make sure to load the plugin on both client and server side as described in section 2.

To verify that the plugin is installed correctly and loaded successfully in ParaView, please create a *Wavelet* source by clicking the menu option [Sources ▸ Wavelet] in ParaView client.

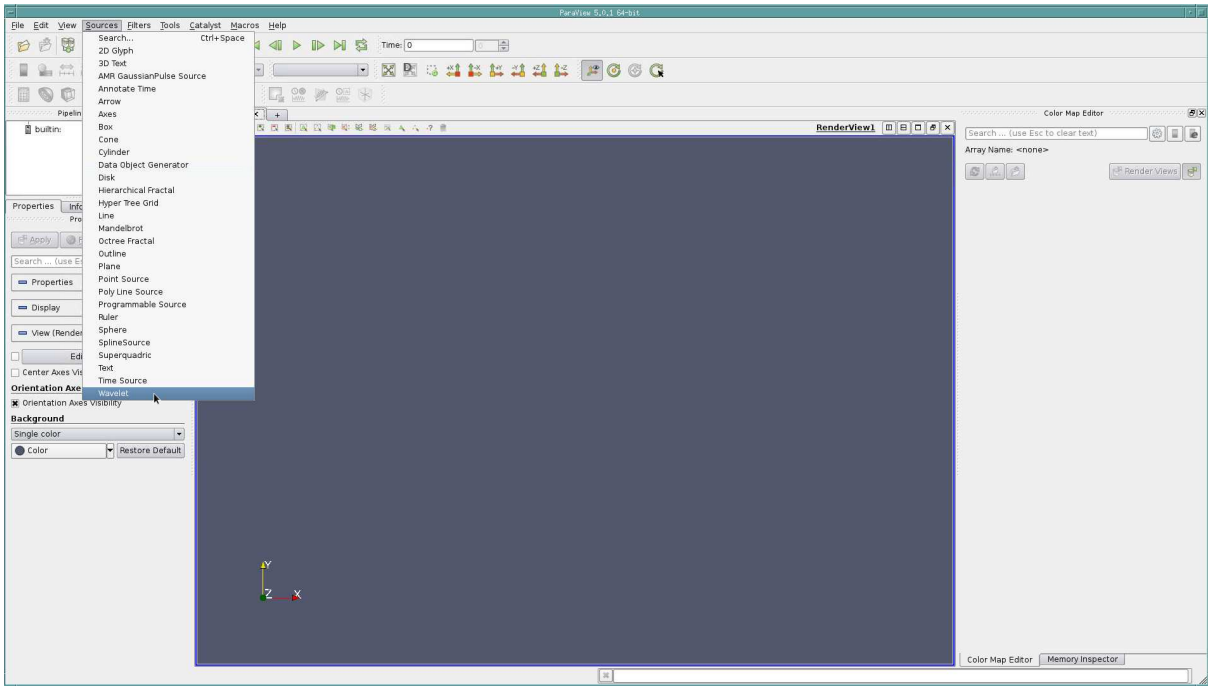


Fig. 3.1 – Create a Wavelet source

Once you click apply an *Outline* representation will be shown in the viewport. Select *RTData* as the scalar array from the dropdown box instead of *Solid Color* and *NVIDIA IndeX* instead of *Outline* as the representation as shown below.

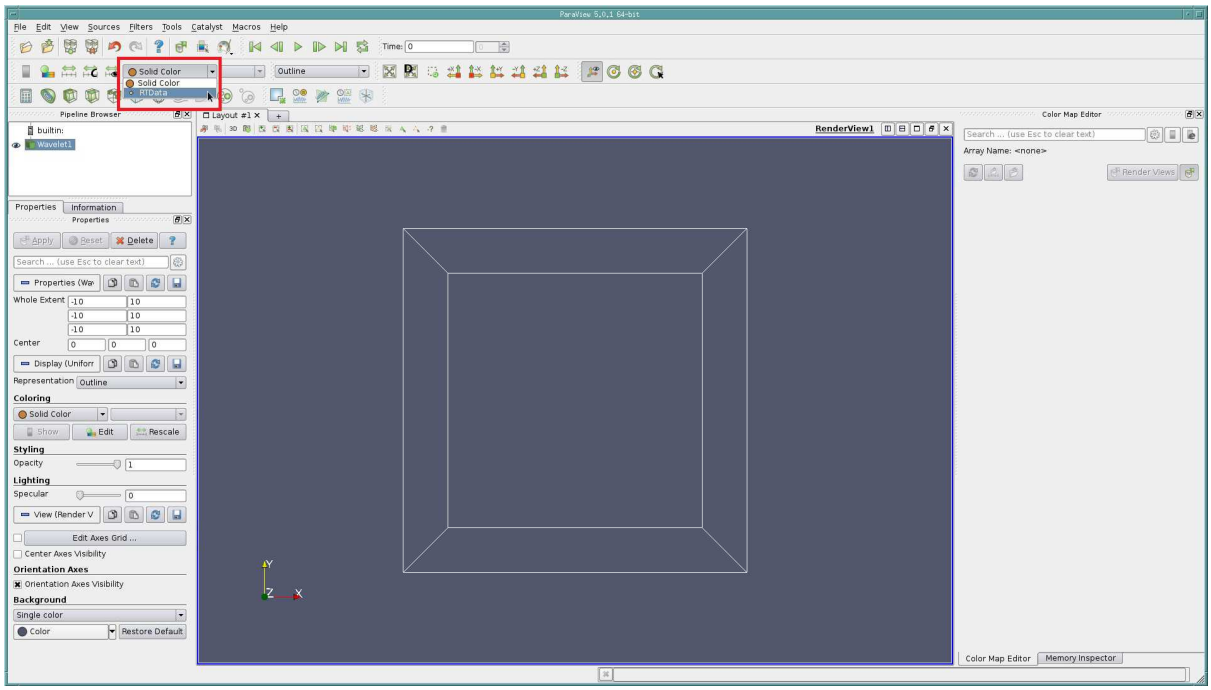


Fig. 3.2 – Outline is default representation

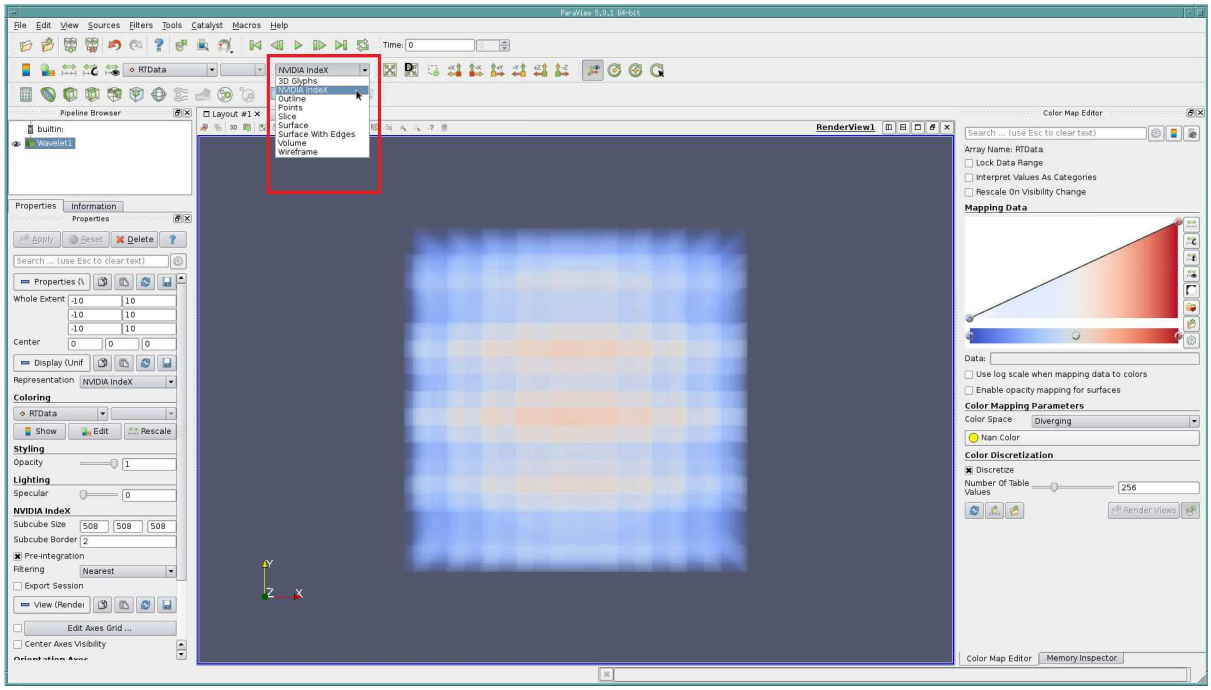


Fig. 3.3 – Wavelet source rendered by NVIDIA IndeX

3.2.1 Client-server mode on multiple GPUs

If you have acquired a valid license for the cluster version of the plugin, you can run the plugin in client-server mode on multiple GPUs. When using multiple GPUs on the same machine it is required to start one MPI process per GPU. For example, on a machine with 4 GPU's:

```
mpirun -bynode -np 1 pvserver -display :0.0 --force-offscreen-rendering \
 : -np 1 pvserver -display :0.1 --force-offscreen-rendering \
 : -np 1 pvserver -display :0.2 --force-offscreen-rendering \
 : -np 1 pvserver -display :0.3 --force-offscreen-rendering \
```

When using multiple GPUs on multiple machines in a cluster environment MPI's host-file functionality can be used. For example, to utilize two machines with two GPUs each:

```
mpirun --hostfile myhosts --bynode -np 2 pvserver -display :0.0 --force-↦
   offscreen-rendering \
 : -np 2 pvserver -display :0.1 --force-offscreen-rendering
```

Where *myhosts* is a text file with list of host names where MPI will spawn pvserver instances.

Before connecting pvservers with ParaView client, please disable IceT compositing. This can be done from ParaView client's settings menu [Edit ▸ Settings ▸ Render view] and restart ParaView client.

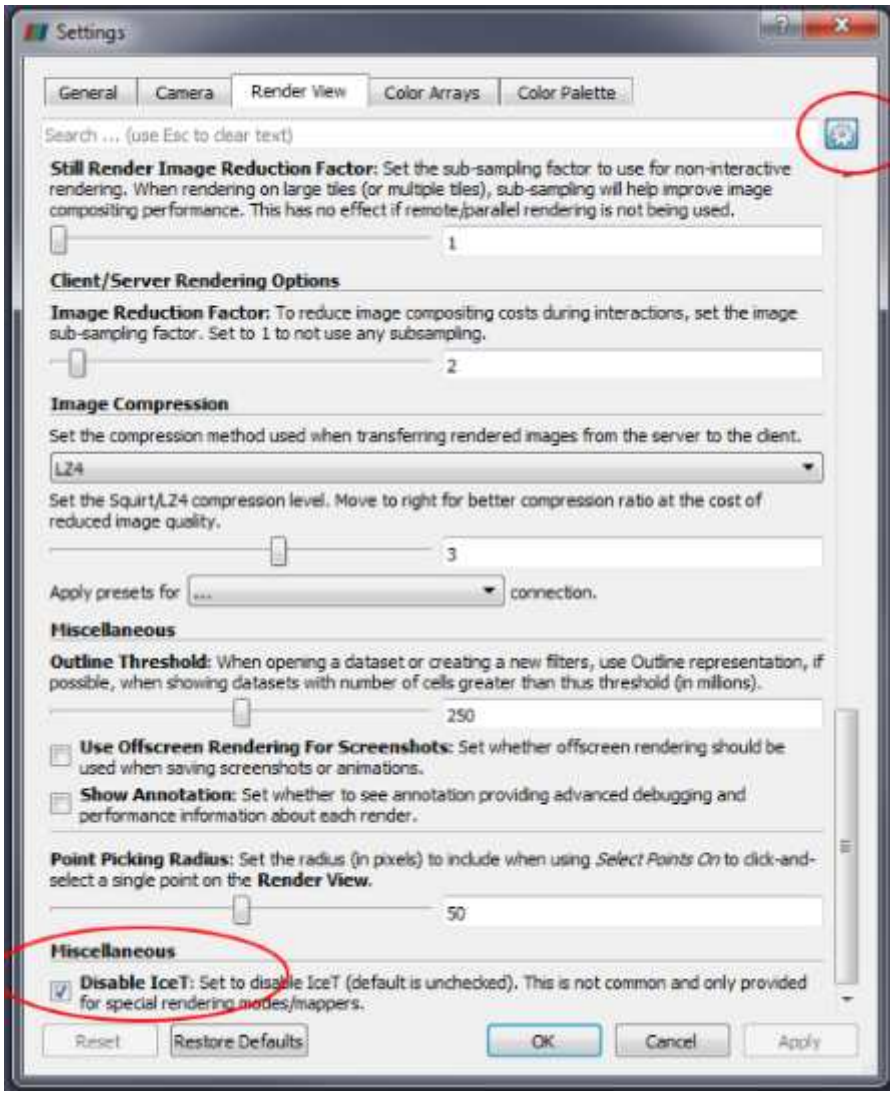


Fig. 3.4 – Disable IceT from Settings menu

Refer to this page[3] for more details about pvserver and running ParaView in client-server mode.

3.2.2 Networking parameters for NVIDIA IndeX

NVIDIA IndeX is a distributed renderer capable of utilizing multiple GPU's on a cluster, to configure networking features of NVIDIA IndeX an optional configuration file `nvindex_config.xml` is provided with the plugin. Copy this file under the default configuration directory of ParaView:

```
cp plugin-directory/nvindex_config.xml ~/.config/ParaView/
```

If the directory `~/.config/ParaView/` is not accessible, you can set the following environment variable pointing to the `nvindex_config.xml` file:

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

3. `http://www.paraview.org/Wiki/Setting_up_a_ParaView_Server`

Each networking option is enclosed within a pair of XML tags and the file is enclosed within `<index_config>` ... `</index_config>` tags.

3.2.2.1 Licensing

When you obtain a license for the cluster version of the plugin, you will receive a file named `license.lic`. It contains the keys `NVINDEX_VENDOR_KEY` and `NVINDEX_SECRET_KEY` that can 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, copy paste those keys in the `<license>` section of the config file as show below.

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

3.2.2.2 Networking options

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

Tag `<cluster_mode>` defines the networking mode of NVIDIA IndeX with modes `UDP` and `TCP` supported, with `UDP` being the preferred mode.

```
<cluster_mode>
   your-networking-mode
</cluster_mode>
```

Tag `<cluster_interface_address>` defines the Network Interface Card(NIC) that is used for communication between the nodes. On Linux, the `ifconfig` command gives the NIC address as `inet addr`. If not set, any address is valid. The string may end with a colon character (:) and a port number to select which port to listen to for UDP and TCP. If no port is set and unicast only mode is set, port 10000 will be used.

```
<cluster_interface_address>
   172.161.123.0/24:10001
</cluster_interface_address>
```

Tag `multicast_address` defines the multicast address for the nodes to communicate. This is valid only when `cluster_mode` is set to UDP.

```
<multicast_address>
   224.1.3.2
</multicast_address>
```

Tag `discovery_address` defines the discovery address used for TCP `cluster_mode`.


```
<discovery_address>
   224.1.3.3:5555
</discovery_address>
```

Tag <use_rdma> can be used to switch yes or no to use RDMA mode for networking.

```
<use_rdma>
   no
</use_rdma>
```

4 Features

This section will provide a walk-through on individual plugin features.

4.1 Structured and unstructred grids

NVIDIA IndeX for ParaView plugin enables volume rendering of both structured and unstructured grid types.

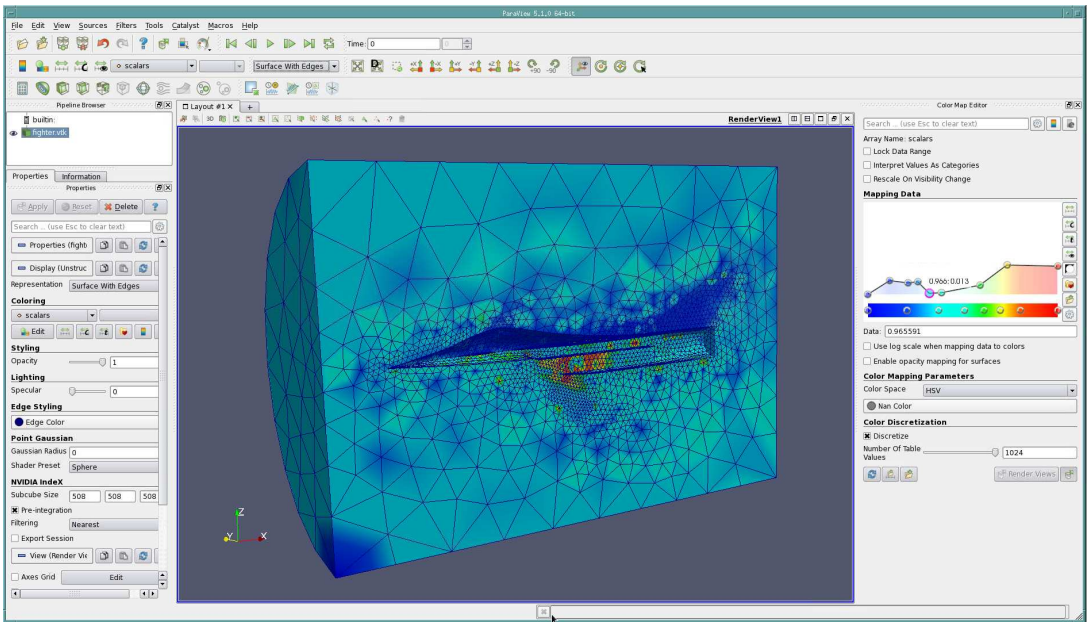


Fig. 4.1 – Grid rendered as a surface

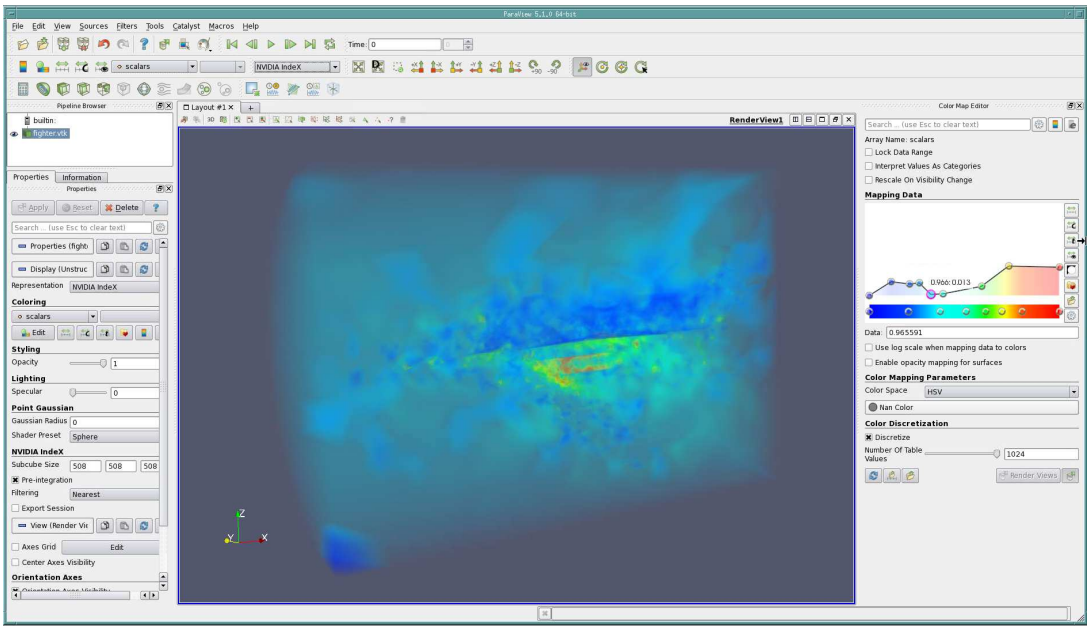


Fig. 4.2 – Grid rendered as a volume using NVIDIA IndeX

4.2 Datatypes

NVIDIA IndeX supports different datatype formats such as *unsigned char, unsigned short and floating point*. Make sure appropriate byte endianness is chosen when loading unsigned short and floating point datatypes otherwise your visualization might look like artifacts or even look completely random.

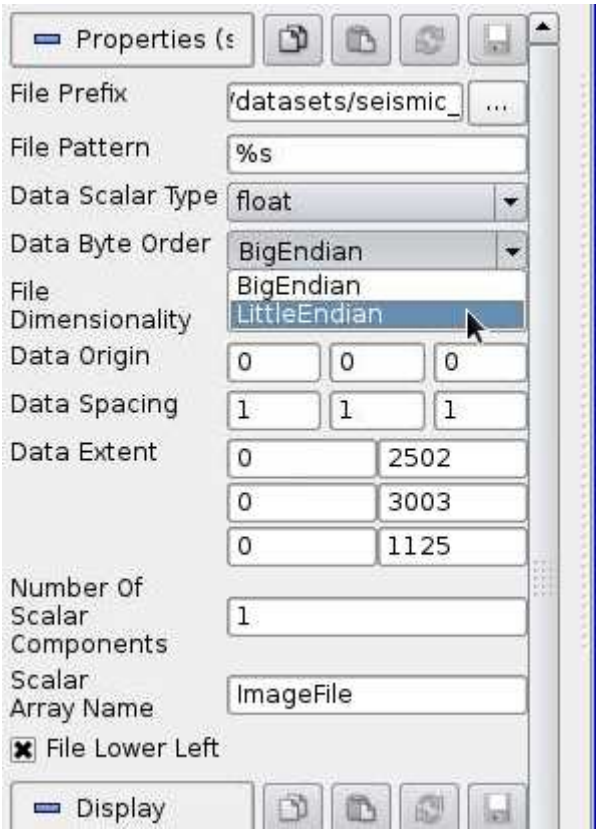


Fig. 4.3 – Choose endianness of the dataset

4.3 Transfer function and colormap changes

Transfer function changes can be done using ParaView's colormap editor. If the colormap editor is not open you can do so by using the menu option.

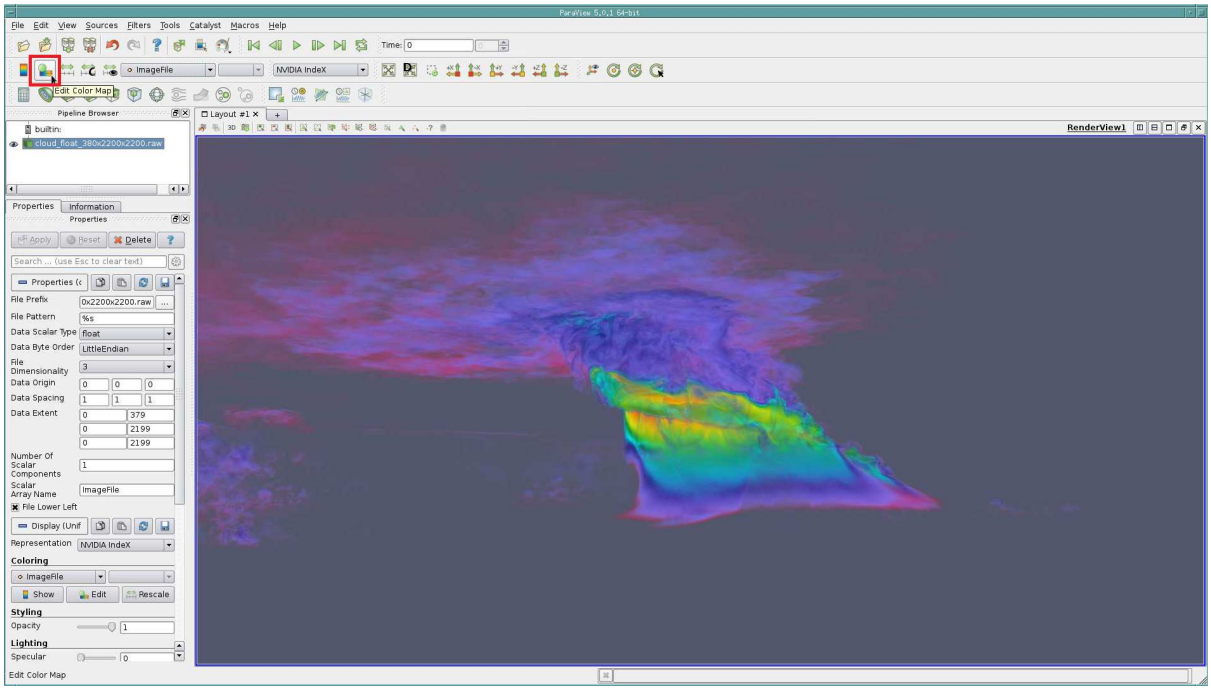


Fig. 4.4 – Click on the icon to open up Colormap Editor in ParaView

Using the colormap editor user interface you can visualize parts of the dataset that is interesting for you. This can be achieved by changing the colortable, by adjusting the transparency, or by setting custom domain range values to isolate parts of the dataset that is uninteresting.

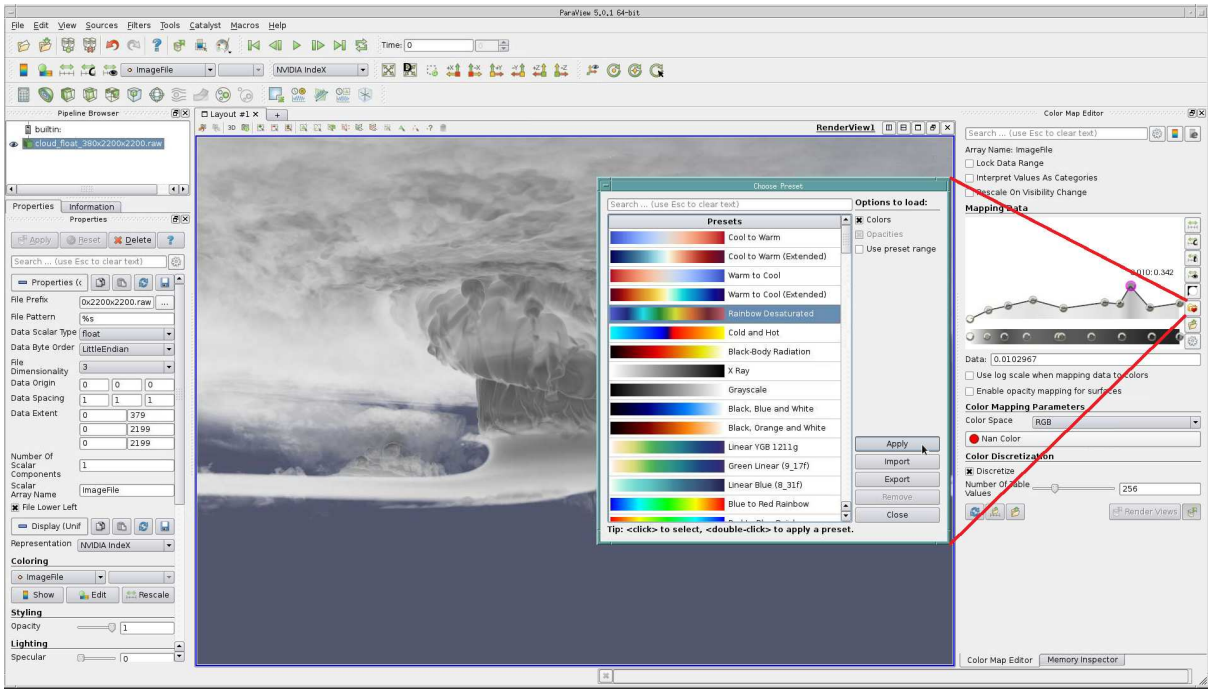


Fig. 4.5 – Choose a suitable colortable and click Apply

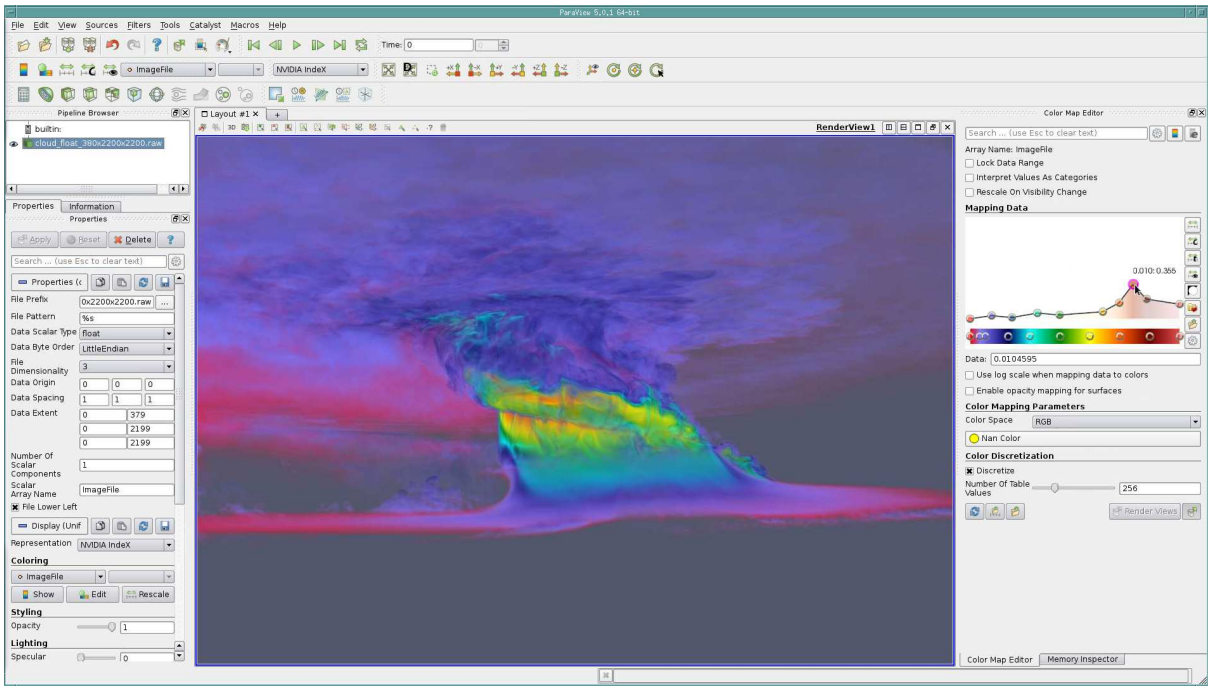


Fig. 4.6 – Colortable matching your dataset domain

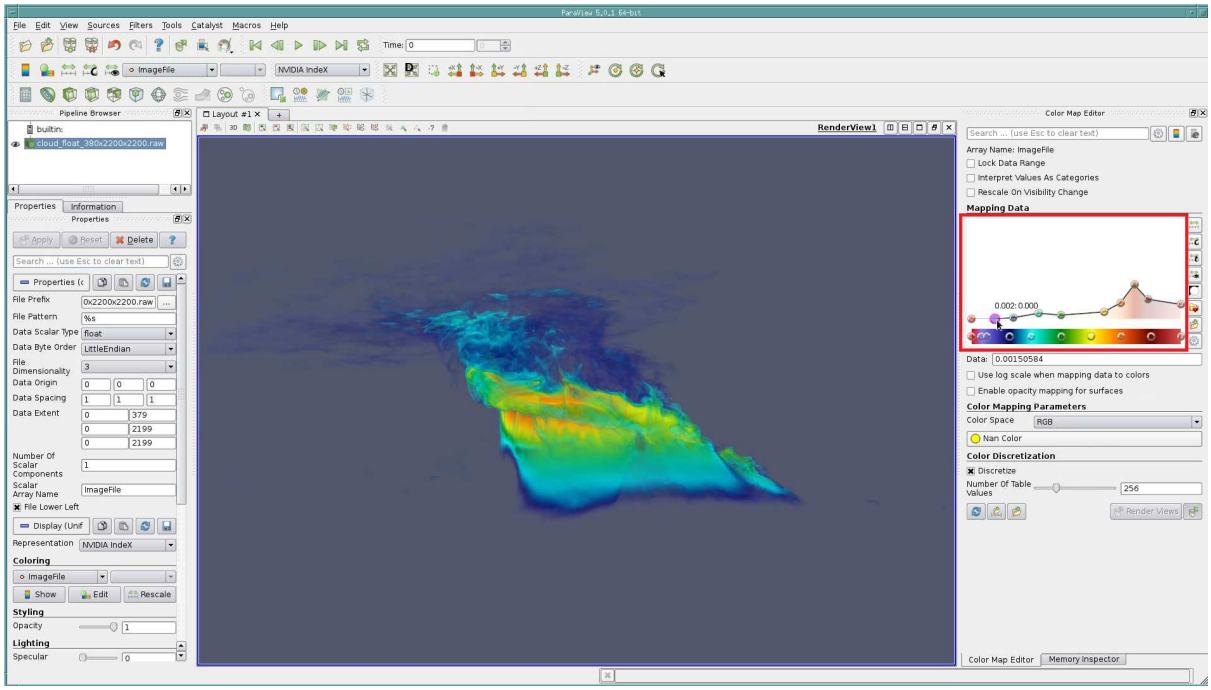


Fig. 4.7 – Changing the opacity values

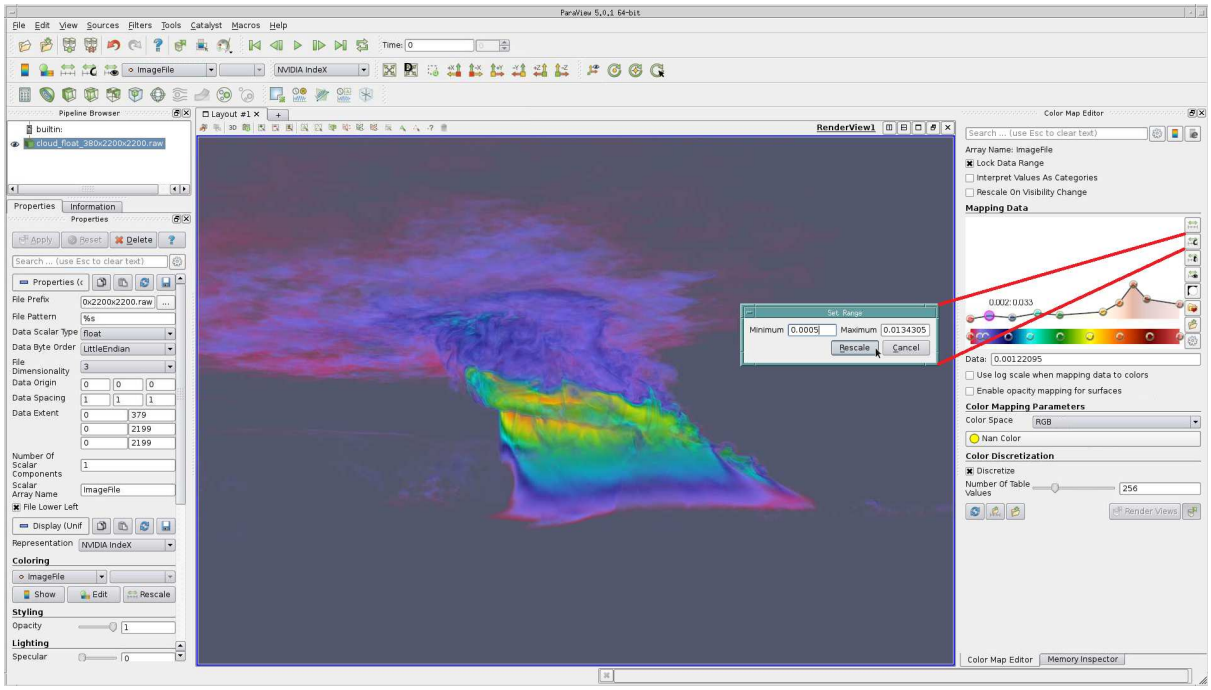


Fig. 4.8 – Custom data domain range

4.4 Region of interest changes

Users can select a custom region of interest to visualize specific sections of the dataset using the sliders in the properties panel. This is tagged as an experimental feature since changing region of interest is restricted to axis aligned directions.

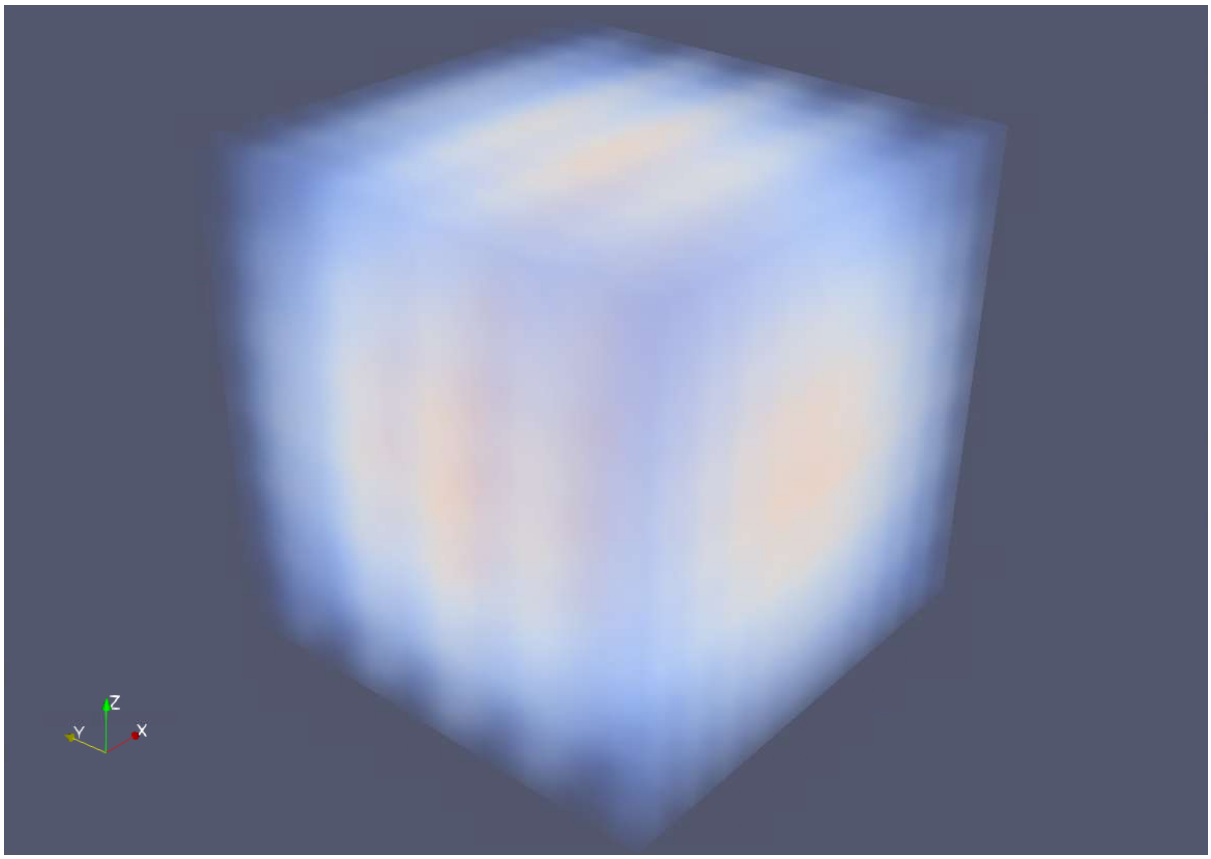


Fig. 4.9 – Wavelet volume

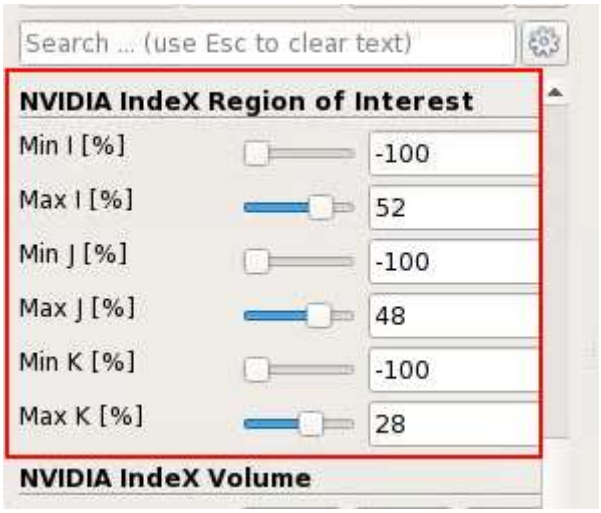


Fig. 4.10 – Changing region of interest

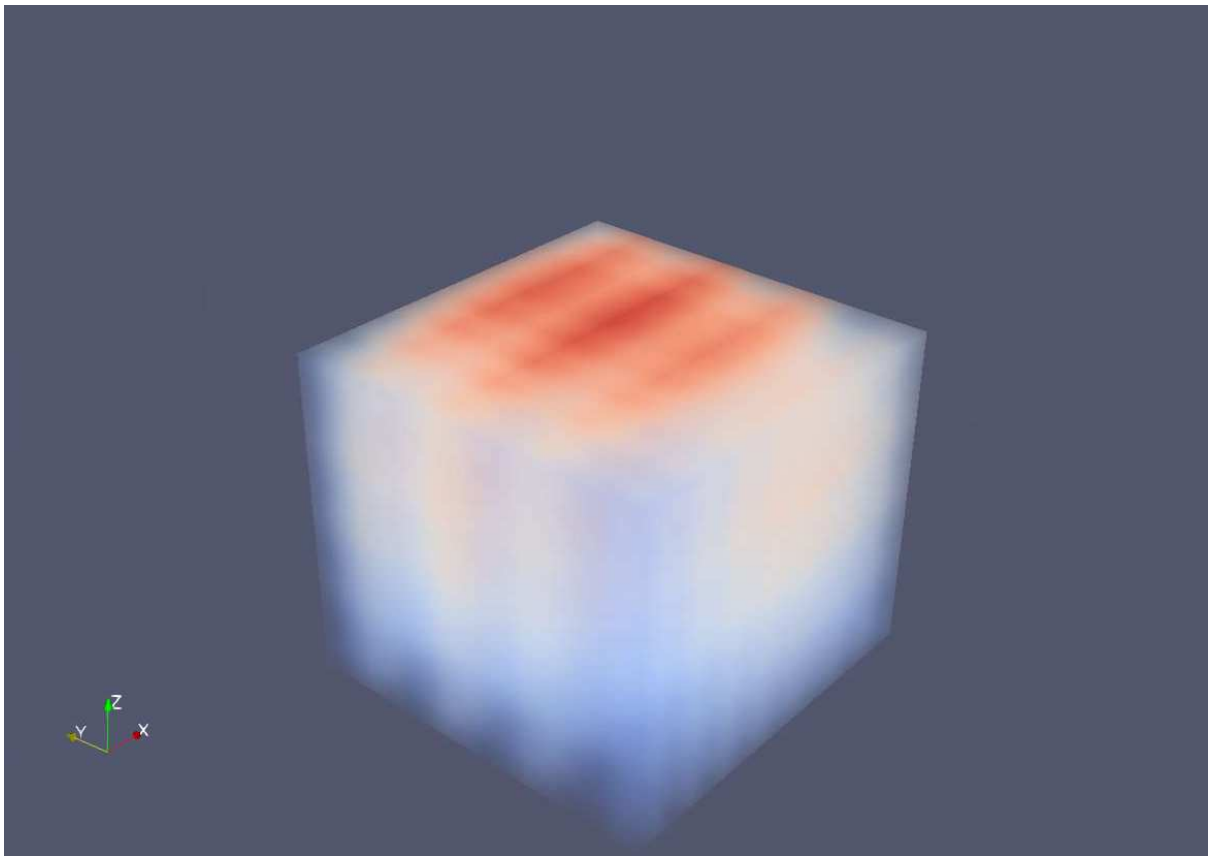


Fig. 4.11 – Wavelet volume with a custom region of interest

4.5 High-quality rendering

High-quality rendering can be achieved by using pre-integration and filtering techniques of NVIDIA IndeX exposed in the plugin, there is a known performance-quality trade off when using some of these filtering techniques.

These filtering options can be found in ParaView's **Properties** panel typically on the left hand side of the ParaView client user interface when the plugin is loaded.

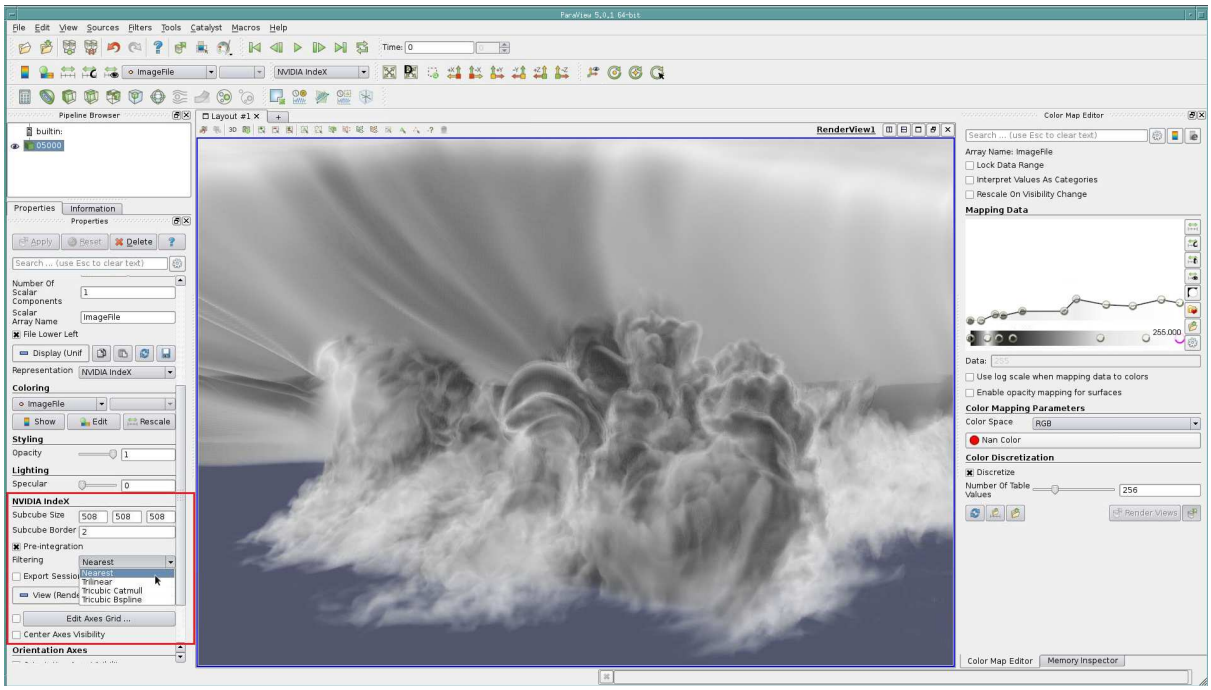


Fig. 4.12 – Properties panel in ParaView

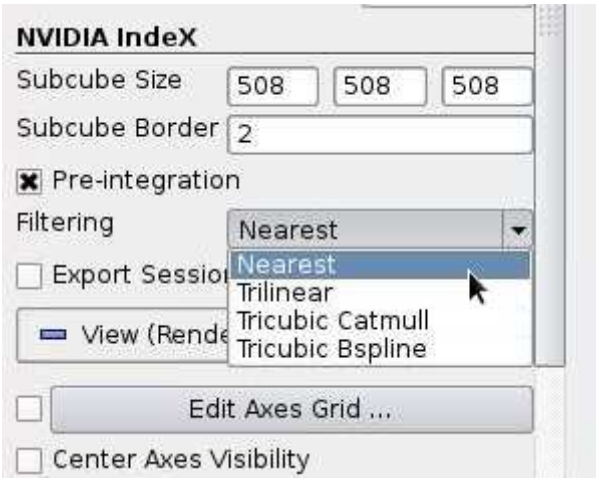


Fig. 4.13 – Filtering options in NVIDIA IndeX properties panel

There is no one filtering option optimal for all the datasets, each filtering option 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 options are shown below.

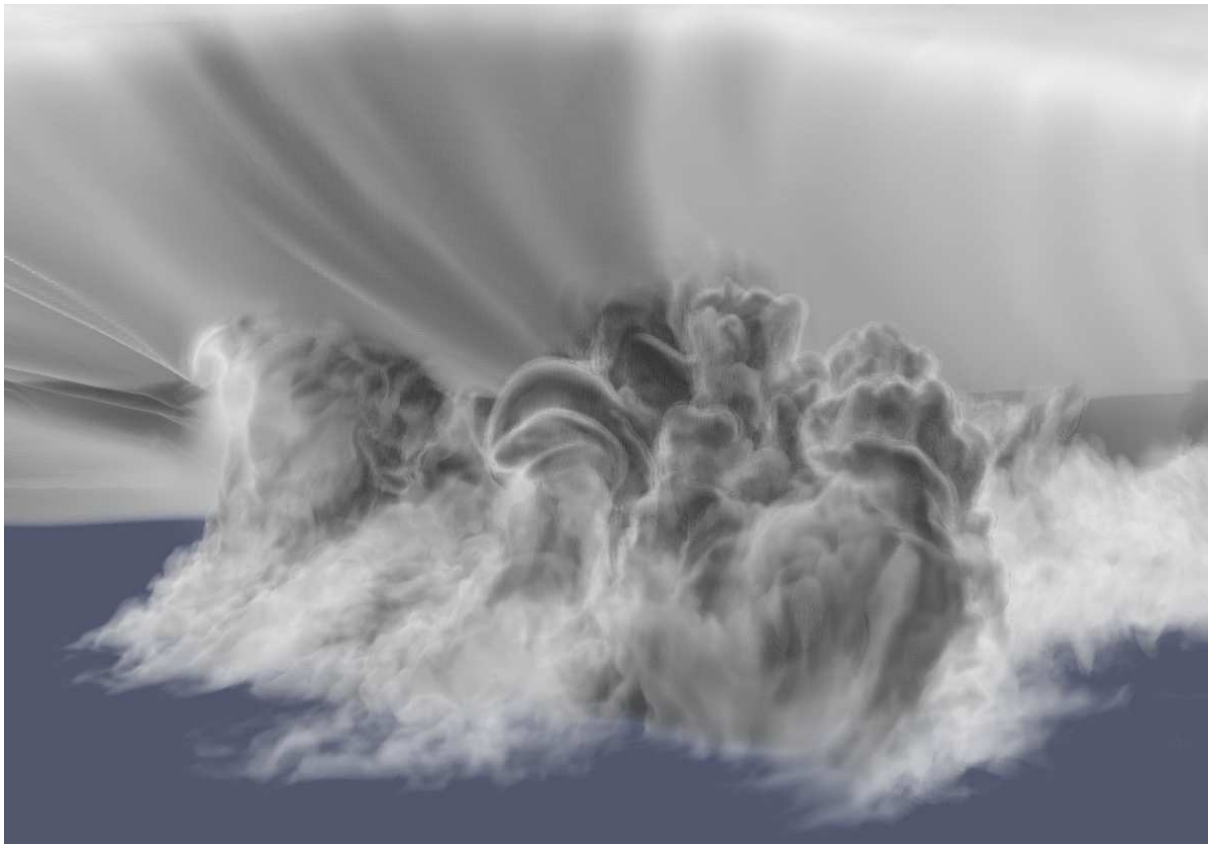


Fig. 4.14 – Base of an EF5 tornado cloud

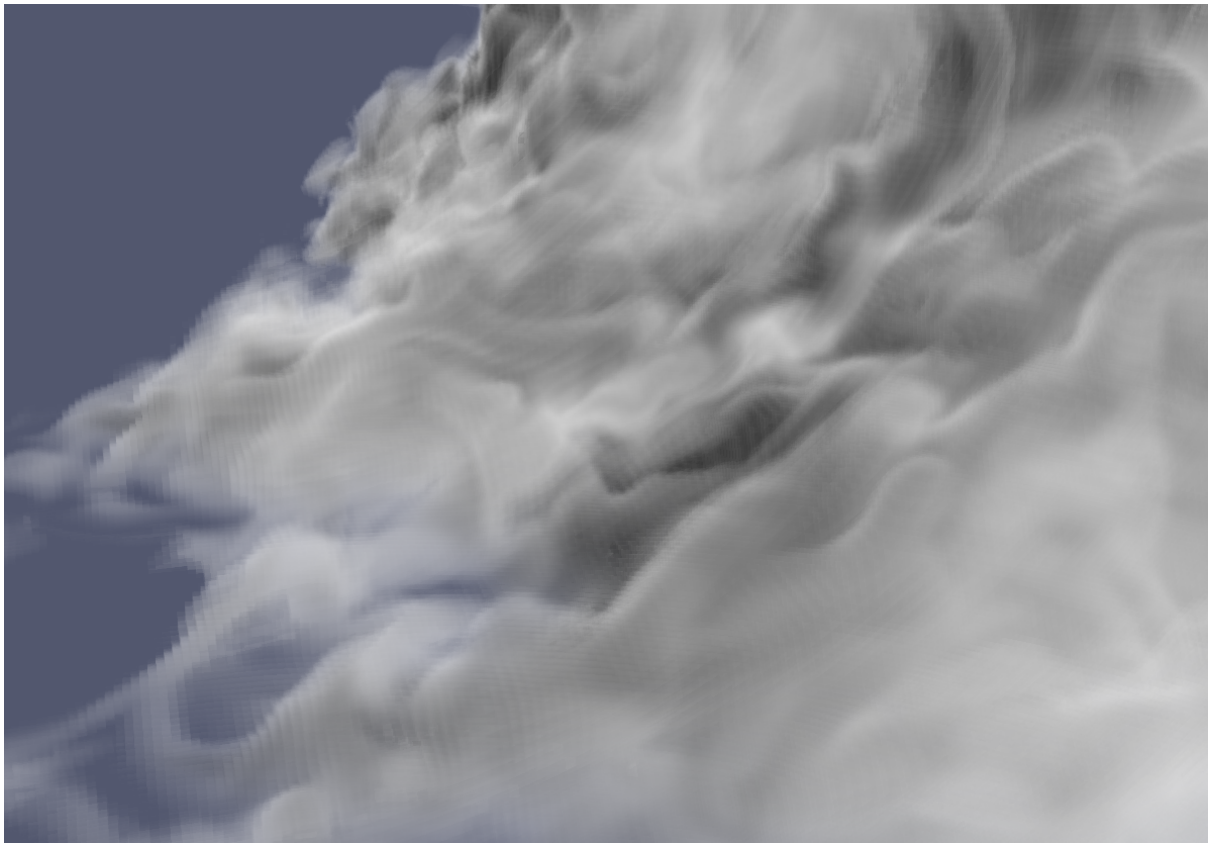


Fig. 4.15 – Nearest neighbor interpolation

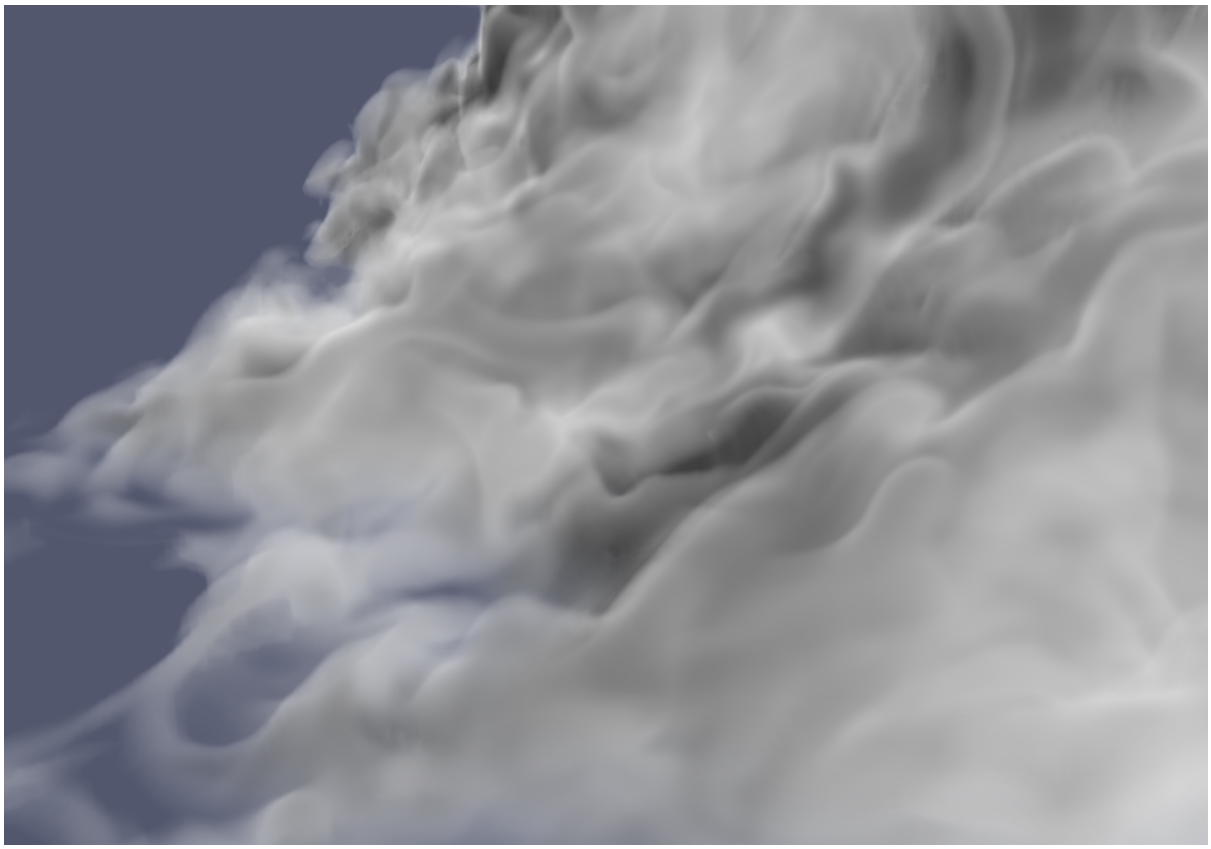


Fig. 4.16 – Trilinear interpolation

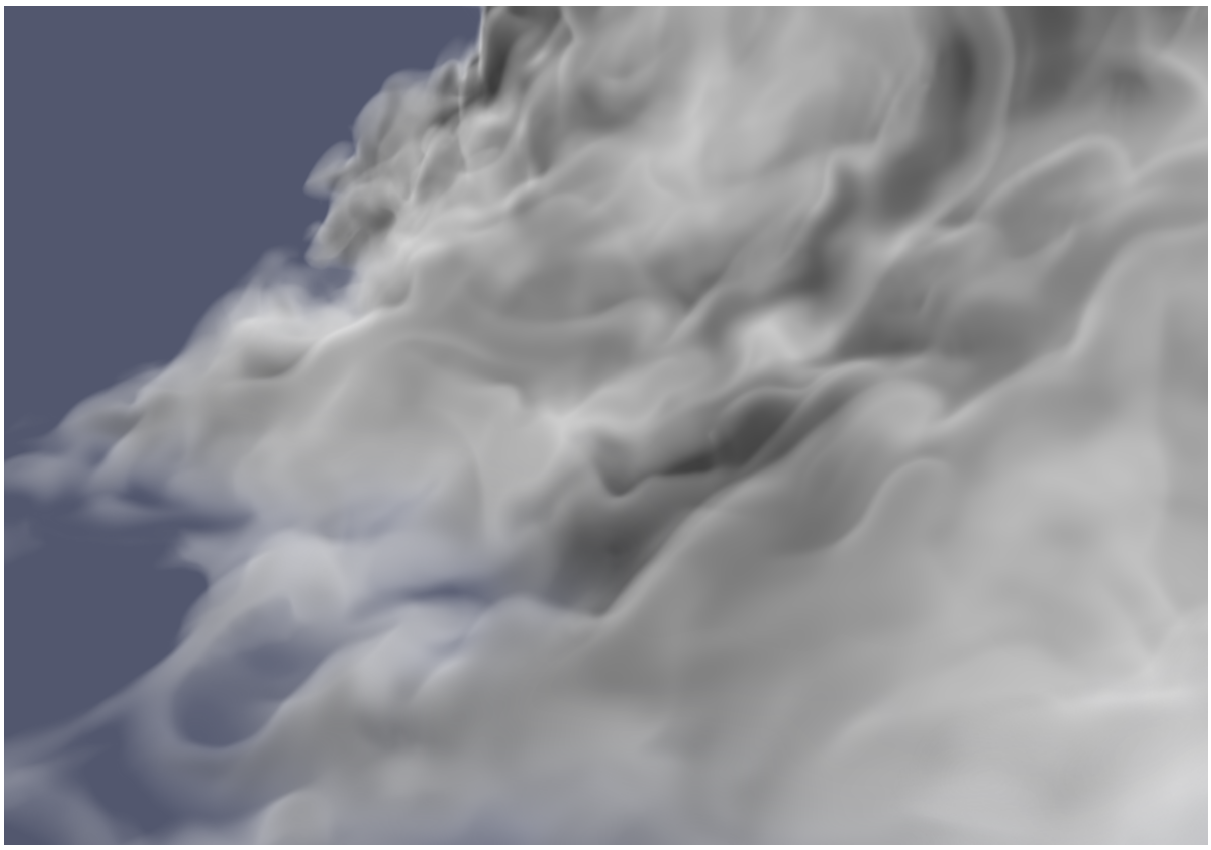


Fig. 4.17 – Tricubic Bspline interpolation

4.6 Slice rendering

Volume slices can be enabled from the GUI. Currently it is limited to three axis aligned slices with ability to move individual slice positions, native support with ParaView slices is work in progress. This is tagged as an experimental feature since the slice rendering is not part of the ParaView slice rendering mechanism in the user interface.

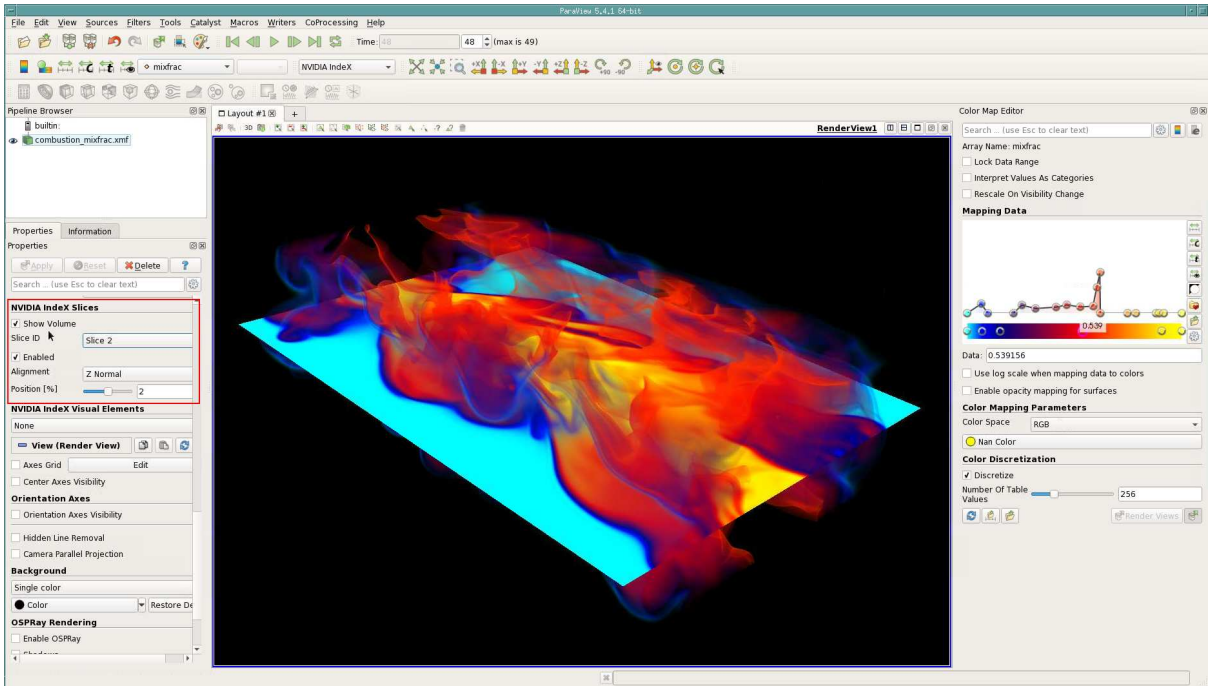


Fig. 4.18 – Slice rendering options through properties panel. Dataset is made available by Dr. Jackqueline Chen at Sandia Laboratories through US Department of Energy's SciDAC Institute for Ultrascale Visuaization.

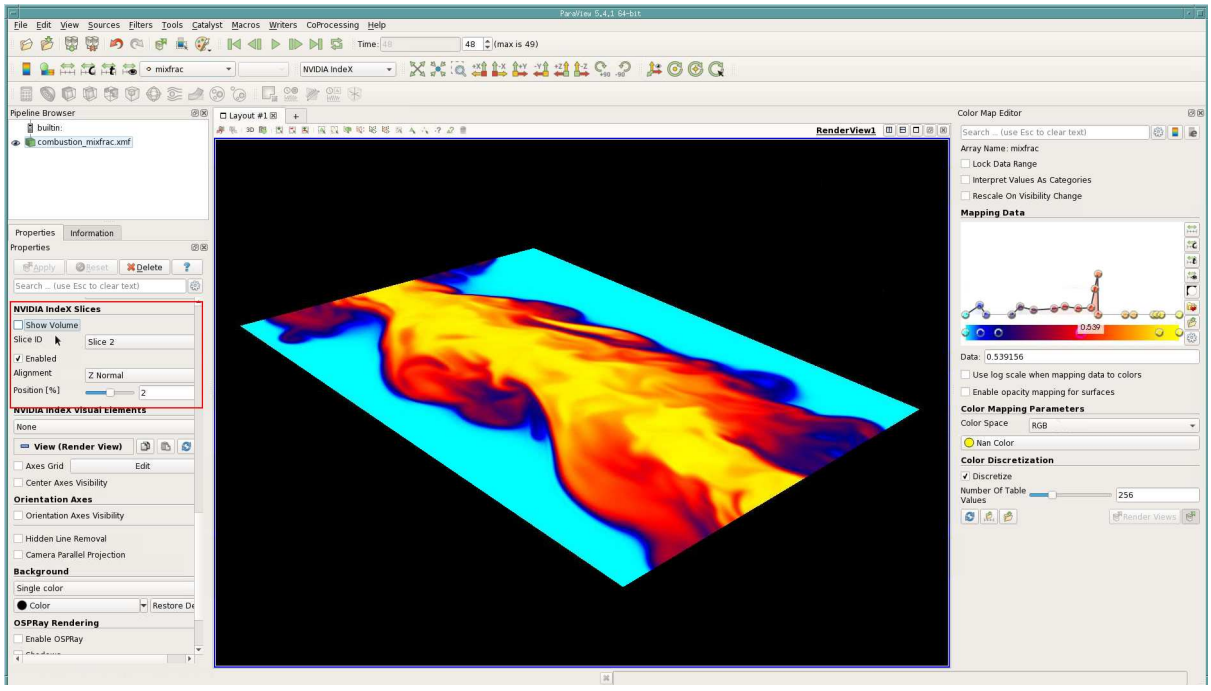


Fig. 4.19 – Slice rendering with volume disabled

4.7 NVIDIA IndeX visual elements

Visual elements feature of NVIDIA IndeX library enables you to enhance the visual cues in the dataset. In this version of the plugin there are five visual presets available, each preset has one or more parameters for finer control over that visual element.

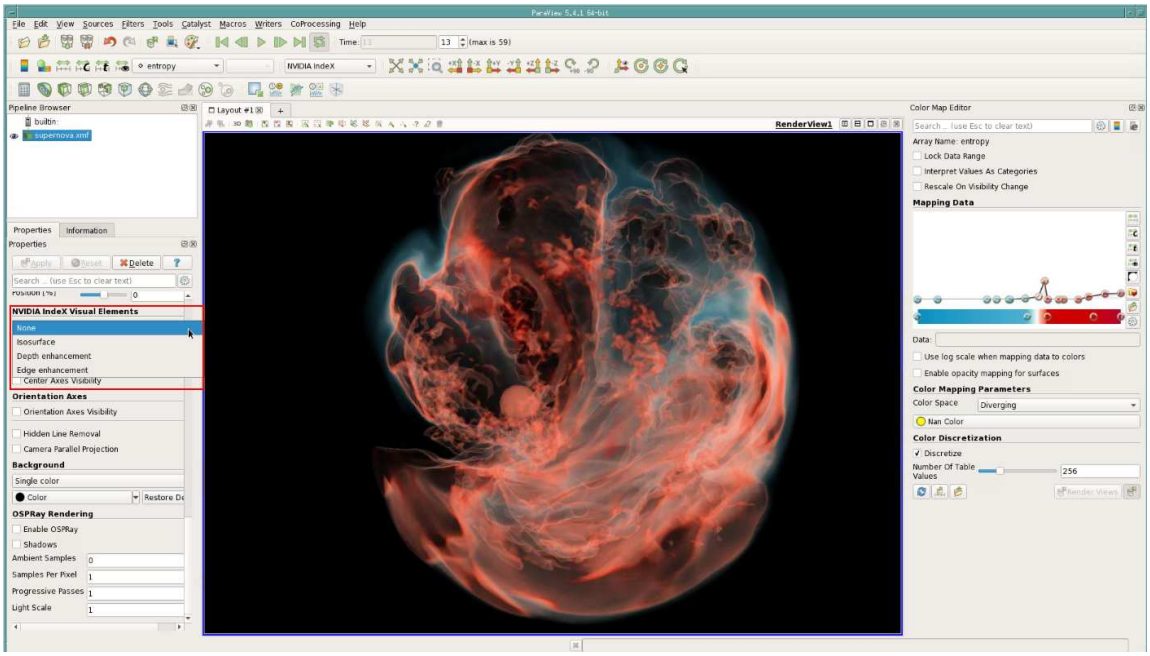


Fig. 4.20 – Supernova SASI visualized as a volume. Dataset courtesy by Dr. John Blondin at the North Carolina State University through US Department of Energy's SciDAC Institute for Ultrascale Visuaization.

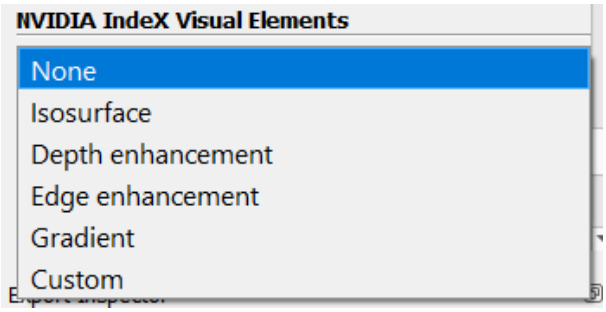


Fig. 4.21 – Visual element presets in the properties panel

4.7.1 Isosurface preset

The Isosurface preset allows you to extract the isosurface and volume contained inside the range `[iso-min, iso-max]` and shade both with different ways to map color sample values.

Iso min/Iso max
: Define the iso-surface range, expressed in percentage of the volume scalar range.

Fill mode
: Defines the shading mode for the inside volume. "No Fill" : Volume samples are set to transparent; "Single Color": Volume samples are set to iso-min value; "Colormap": Volume samples are taken from colormap.

Use shading
: Enables/Disables the phong-blinn lighing model for the iso-min/iso-max surfaces.

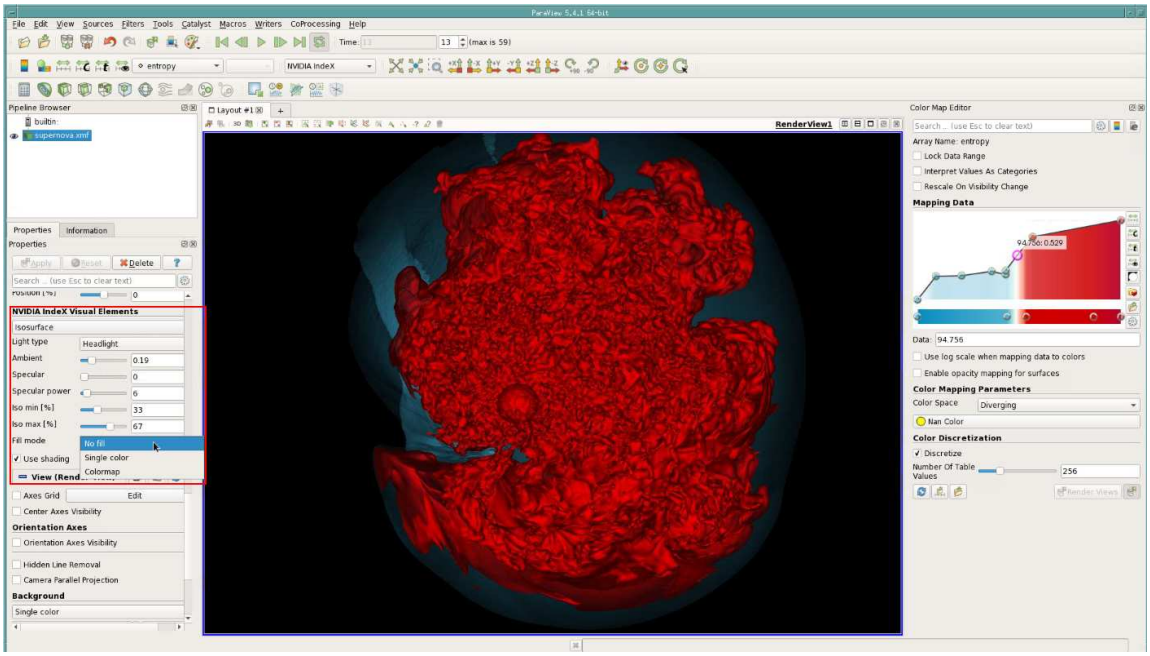


Fig. 4.22 – Supernova SASI visualized as an iso-surface

4.7.2 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 the current volume position it accumulates colormap alpha values along a predefined line segment and darken samples in regions with low alpha distribution.

Samples
: Number of samples taken along the line segment.

Gamma
: Used to increase contrast.

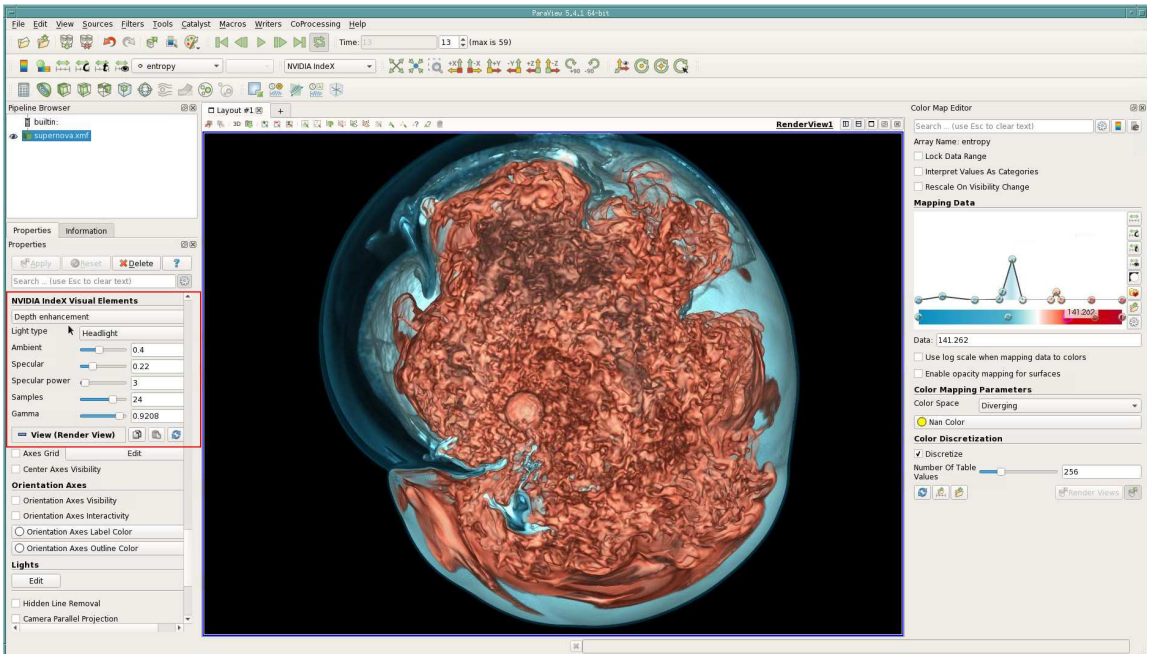


Fig. 4.23 – Supernova SASI visualized using depth enhancement preset

4.7.3 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 the current volume position it calculates the gradient of the colormap alpha-channel along a predefined line segment and darken samples that contain higher gradient magnitude.

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

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

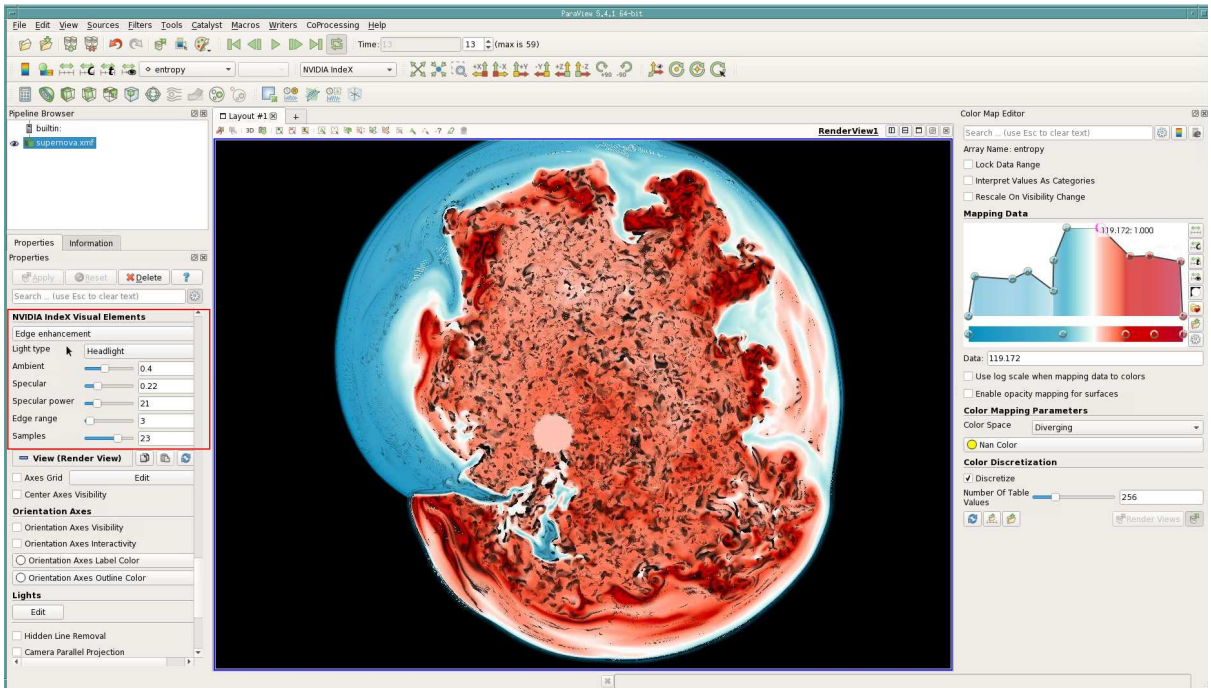


Fig. 4.24 – Supernova SASI visualized using edge enhancement preset

4.7.4 Gradient preset

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

Gradient Level
: The gradient level or intensity in percentage.

Gradient Scale
: The maximum gradient level.

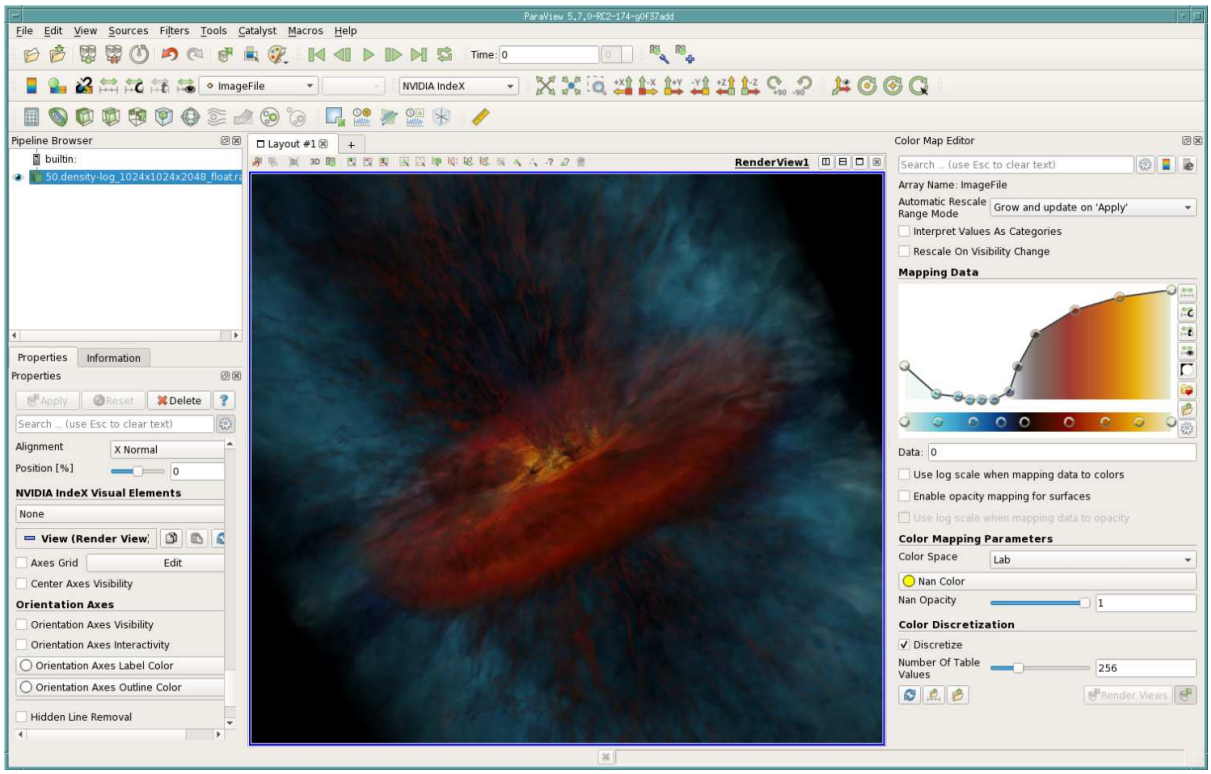


Fig. 4.25 – CHOLLA galactic outflow simulation visualized as a volume. Dataset courtesy by Evan E. Schneider (Princeton University) and Brant Robertson (University of California, Santa Cruz).

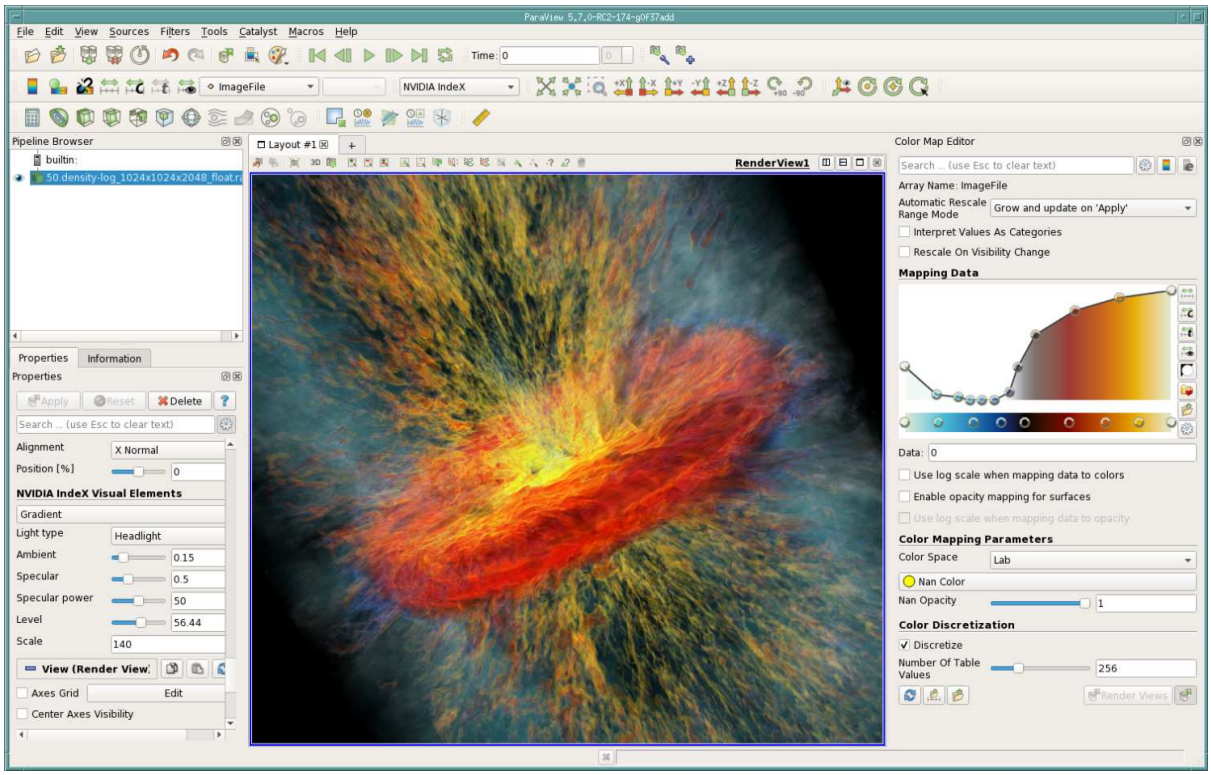


Fig. 4.26 – CHOLLA galactic outflow simulation visualized using gradient preset.

4.7.5 Custom preset

The Custom preset allow users to write their own volume kernel programs using XAC (NVIDIA IndeX Accelerated Computing Technology). The XAC kernels are small CUDA programs (editable with your preferred text editor) that can be used to replace the default volume shading kernel executed by IndeX. The XAC kernels are compiled by IndeX at runtime, which means that you can load a kernel and edit it on the fly and changes are applied immediately. The custom preset provides the means to load a XAC kernel from file, apply updates on the fly and provides a list of predefined user parameters that can be binded to your volume kernel program.

A tutorial with the basics on XAC programming can be found on Appendix A (page 36) of this User's Guide. Also a few volume kernel examples are included with ParaView binaries (`paraview-installation-directory\kernels_nvidia_index`) and Paraview sources (`paraview-source-root\Plugins\pvNVIDIAIndeX\kernel_programs`).

Kernel
: The XAC volume kernel program to be loaded from file. It triggers IndeX kernel compilation.

Update Kernel
: It re-triggers the IndeX kernel compilation to apply live changes done to the XAC volume kernel program, for example with an external text editor.

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

pint 1-4
: Four general purpose integer parameters that can be binded to the XAC volume kernel 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. 4.27 – Custom preset: Floating point and integer parameters binding with XAC kernel program.

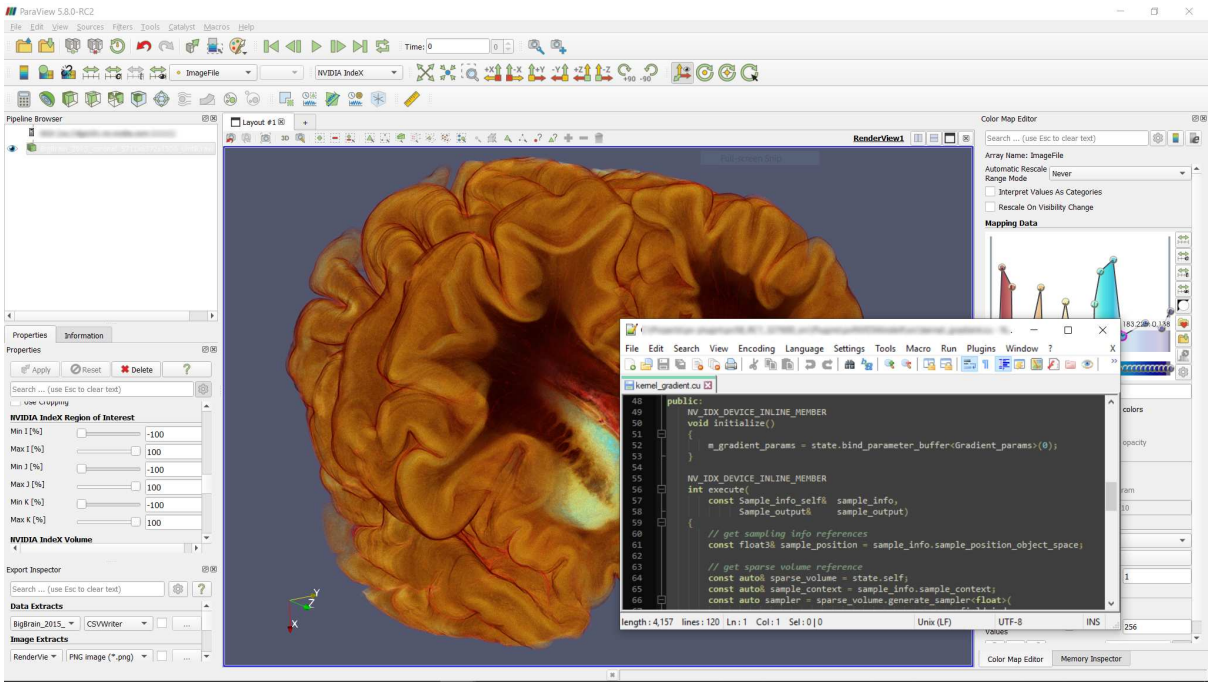


Fig. 4.28 – BigBrain Project brain visualized with a custom preset. Dataset courtesy by Prof. Dr. med. Katrin Amunts and the Structural and Functional Organization of the Brain lab at the Institute of Neuroscience and Medicine, Research Centre Juelich.

4.8 Time series animation

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 do with a static dataset. In order to have a smooth playback, please set the following setting from ParaView menu allowing you to cache geometry. Animation will be slower in the first cycle but once all the timesteps are loaded the playback should be smooth.

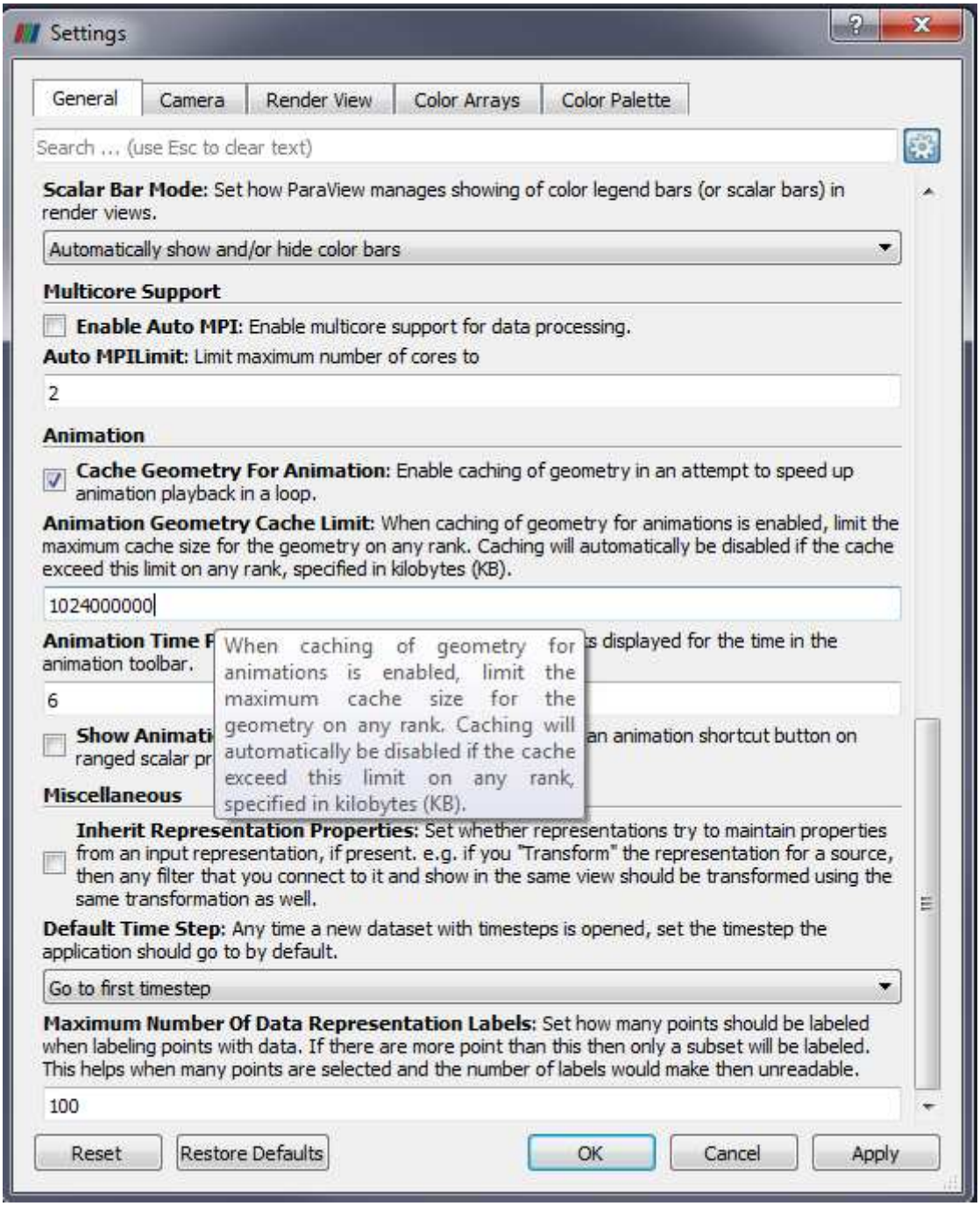


Fig. 4.29 – Enable Cache Geometry for Animation and set a high value for cache limit

4.9 Catalyst and in-situ visualization

NVIDIA IndeX supports in-situ visualization, a user can run a simulation and visualize it in real-time without writing any data to disk. Catalyst[4] 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 setup 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.

4. `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 50 iterations of a wavelet data source on a single process by running the following command. Both CatalystWaveletDriver.py CatalystWaveletCoprocessing.py scripts are under the directory `../Applications/ParaView/Testing/XML/` in ParaView source.

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

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

```
./paraview
```

Once ParaView connects to Catalyst, enable "input" and click "Extract input" from the pipeline browser. Once the input is extracted you can switch to NVIDIA IndeX representation from the menu.

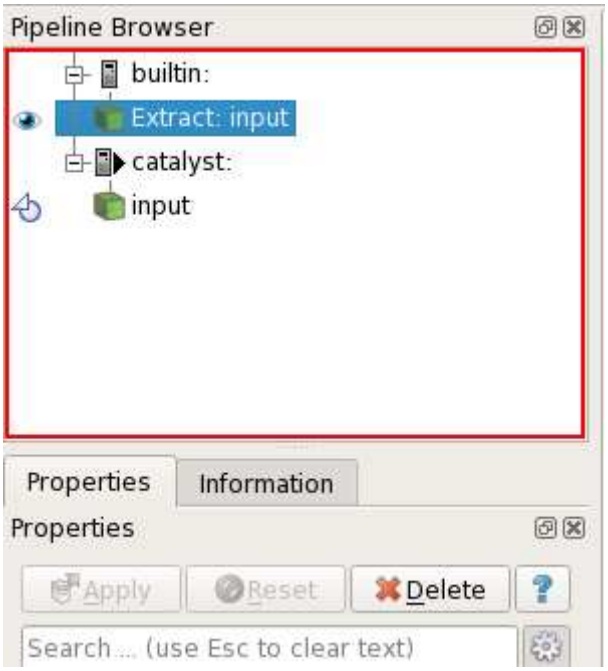


Fig. 4.30 – Enable input and extract input to visualize

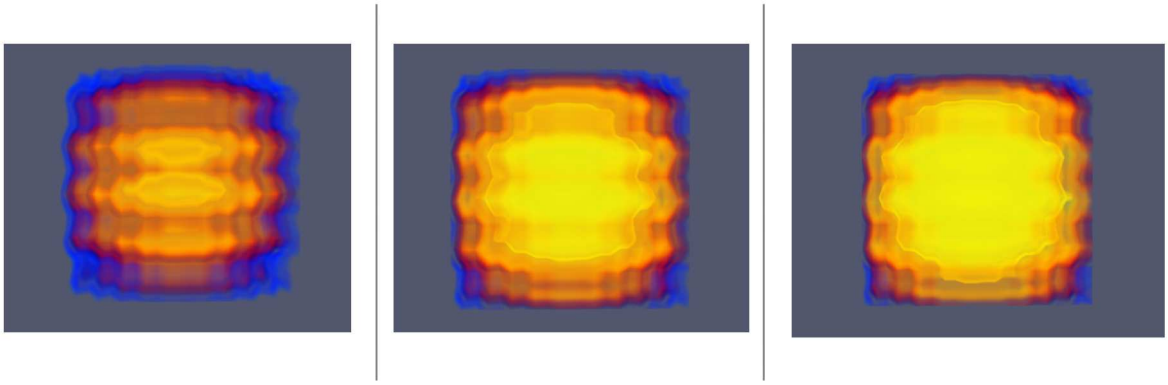


Fig. 4.31 – Wavelet example shown at different iterations

4.10 Mixing ParaView primitives

One of the unique features of the plugin is to mix volume rendering from NVIDIA IndeX along with other primitives such as Glyphs, Streamlines and Surfaces rendered by ParaView.

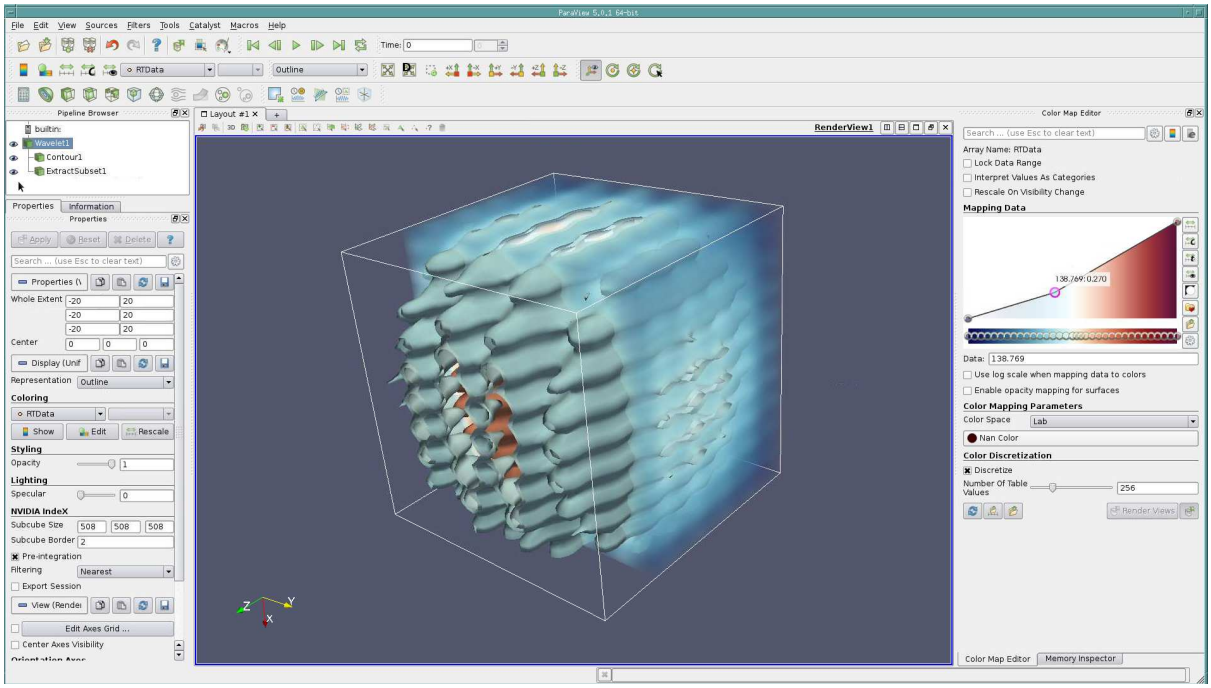


Fig. 4.32 – Wavelet data rendered as a Surface by ParaView and as a volume by NVIDIA IndeX

5 Frequently asked questions

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

A: There is no need to install CUDA separately as the plugin package is bundled with all the required libraries. However, you need to have an appropriate NVIDIA display driver for your graphics card.

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

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

Q: Plugin is loaded successfully without any errors but *NVIDIA IndeX* as a representation does not show up in ParaView's representation dropdown box.

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

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

A: This error message is usually printed when NVIDIA IndeX library (in `libnvindex`.) is not found. Make sure you have `libnvindex`. and `libdice`. in your `LD_LIBRARY_PATH` (or PATH on Windows). You can also copy all the libraries from the plugin directly into ParaView's library directories. Refer section-2 for more information.

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

A: Select 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.

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 optimizes for high latency networks and enables compression and level of detail, you can disable this from [Edit ▸ Settings] option and turn off LOD Resolution, Image Reduction Factor and Image Compression.

Q: Can I render multiple volumes at once in the same scene graph 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 NVIDIA IndeX library in my own application without ParaView?

A: Sure you can, contact us for more details.

Q: What if I want to have a feature that is part of NVIDIA IndeX but not integrated in the ParaView plugin?

A: Full set of NVIDIA IndeX features are described on this webpage.[5] If there is a feature that is important for you please contact us, we are happy to take workflow and feature requests.

5. `https://developer.nvidia.com/index`

6 Useful links

- NVIDIA IndeX for ParaView plugin website[6]
- NVIDIA IndeX for ParaView plugin forum[7]
- NVIDIA IndeX website[8]
- ParaView binaries and source code download[9]
- ParaView documentation[10]
- ParaView user guide[11]
- Contact email: `paraview-plugin-support@nvidia.com`

6. `http://www.nvidia.com/object/index-paraview-plugin.html`
7. `https://forum.nvidia-arc.com/forumdisplay.php?210-NVIDIA-IndeX-for-ParaView-Plug-in`
8. `https://developer.nvidia.com/index`
9. `http://www.paraview.org/download/`
10. `http://www.paraview.org/documentation/`
11. `http://www.paraview.org/paraview-guide/`

Appendix A Volume rendering tutorial

This tutorial 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 (or transfer functions). In addition it shows how to customize programs and make use of CUDA buffer parameters to interactively change properties of those programs.

A.1 XAC purpose and program structure

The NVIDIA IndeX Accelerated Compute (XAC) defines an infrastructure to work with interactive rendering and compute CUDA-based *sampling programs* (kernels or shaders) that are compiled at runtime into the IndeX rendering environment.

There are two fundamental types of XAC programs: *volume sample programs* and *surface sample programs*. Both programs have a predefined structure, receive specific input data, can set specific output, and can be specialized towards different types of scene elements. They have to be added to the scene and can be added using the [CUDA Code Editor].

A.2 XAC volume sample programs

To understand the purpose of those programs, lets dive into a few internal aspects of NVIDIA IndeX first:

During rendering, *rays* are generated for each pixel and intersected with elements in the scene. When such a view ray hits a volume scene element, *volume sample programs* are executed at each sample position along the ray and produce a four component vector (x: red, y: green, z: blue, w: alpha as RGBA color), and 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 (for example, the software version, libraries)

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

Execution function
: Computes the output color passed to the renderer

A.2.1 Example program outline

To successfully compile an XAC program your program should have the following components:

Listing 7.1

```
NV_IDX_XAC_VERSION_1_0                          Current XAC version string (optional)

using namespace nv::index;                      Include the default namespace (contains object and
using namespace nv::index::xac;                 helper classes)

class Volume_sample_program                     Declare the base class
{
    NV_IDX_VOLUME_SAMPLE_PROGRAM                Predefined type declaration (required)

   public:
    NV_IDX_DEVICE_INLINE_MEMBER                 Initialization function
    void initialize()
    {
       // initial setup...
    }
    NV_IDX_DEVICE_INLINE_MEMBER                 Main rendering function (required)
    int execute(
       const Sample_info_self& input, Sample_output& out)
    {
        float4 color = make_float4(1.0f);       Do some color computations here...

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

Add this code to the [CUDA Code Editor] panel and press the Compile button to update the volume sampling program in the scene. Note that the XAC program has to be *before* the target volume scene element to be executed.

A.3 Sampling a volume and map to a color

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

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

`sample_position` (float3)
: Stores a reference to the current sample position (volume grid)

`scene_position` (float3)
: Stores a reference to the scene position

`ray_origin` (float3) `ray_direction` (float3) `ray_t` (float)
: Stores a reference the current view ray properties

The full XAC API is documented in the *NVIDIA IndeX Programmer's Manual*.

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

Listing 7.2

```
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 reference to the sparse 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);   Sample the color value

    output.set_color(sample_color);                 Store the output color (RGBA) and return result

    return NV_IDX_PROG_OK;
}
```

A.4 Using XAC library functions

Within each XAC sampling program you have access to several helper functions. This includes the following core libraries:

- The CUDA math library[12]
- Basic linear algebra functions (for example, point, vector, and matrix operations)
- XAC library convenience functionality – `nv::index::xaclib`)
- 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.

12. `https://docs.nvidia.com/cuda/cuda-math-api/index.html`

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

Listing 7.3

```
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 a color

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

Store the output color and return result

A.5 Add basic volume shading

Now, 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 model based on a headlight) that helps to emphasize isosurface directions. To do this, we need to compute a isosurface normal, which can be derive from the *volume gradient*, for which we use a XAC library function.

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

Listing 7.4

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

Initialize a RGBA color (as four floats) reusing the color mapping

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

Approximate the the volume gradient based on finite-differences

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

Get the iso-surface normal (in outward direction) and view direction

```
const float4 specular_color =
   make_float4(1.0f, 1.0f, 1.0f, 1.0f);
```

Define specular (reflective highlight) color

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

Apply built-in headlight shading and set the sample color

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

Store the output color and return result

A.6 Using CUDA parameter buffers

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

To use those buffers, the following modifications to the scene file and to the CUDA kernel file are required.

A.6.1 Modifying the scene file

Add a parameter buffer element to a scene group:

Listing 7.5

```
#2 XAC parameter buffer element
app::scene::xac_parameters::type =
  rendering_kernel_program_parameters

#4 Setup parameter types per slot (optional)
app::scene::xac_parameters::nb_parameters = 3
app::scene::xac_parameters::0::type       = float32
app::scene::xac_parameters::1::type       = uint
app::scene::xac_parameters::2::type       = int

#1 Add the parameter element to a scene group
app::scene::main_group::children = xac_parameters ...
```

A.6.2 Modifying the CUDA kernel file

The buffer has to be bound and can then be used in the program:

Listing 7.6

```
float input_value = 0.0f;
```
Define the user parameter variable

```
NV_IDX_DEVICE_INLINE_MEMBER
void initialize()
{
    const float* buffer =
       state.bind_parameter_buffer<float>(1);
    input_value = buffer[0];
}

NV_IDX_DEVICE_INLINE_MEMBER
int execute(
    const Sample_info_self& sample_info,
         Sample_output&    sample_output)
{
```
Bind input parameter buffer from parameter slot 1 to the variable


```
    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, 0.0f, 1.0f);
```

Initialize a color and set color channels

```
    sample_output.set_color(modified_color);
    return NV_IDX_PROG_OK;
}
```

Compute the output color

Note that the buffer type and structure has to be aligned in the scene file and the CUDA kernel file. In the HTML5 viewer you can use the [CUDA Parameter Panel] to setup the values of the parameters interactively.


NVIDIA