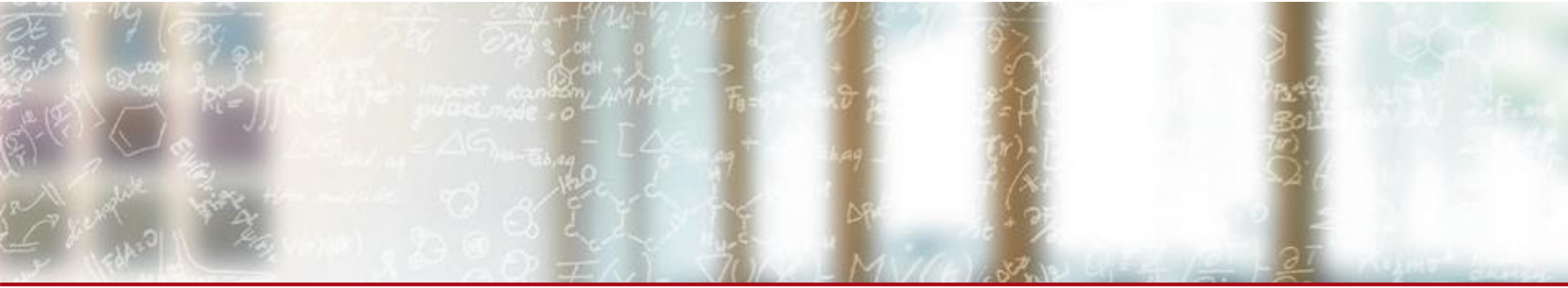


Scientific Visualization

Dr. Jean M. Favre, CSCS

May 17-18, 2021

Objectives, outline, motivations, executive summary

- Have fun
- Or at least, make the most efficient use of the compute resources, the visualization tools, …

to best represent what is in our data

https://www.cscs.ch/events/upcoming-events/event-detail/scientific-visualization-1/

resources

git clone git@github.com:jfavre/Visualization-training

https://www.cscs.ch/publications/video-portal/

https://discourse.paraview.org/

https://support.cscs.ch

Course account

- Each one of you should receive a loggin account valid only today and tomorrow.
- Associated with this account id, there is a reservation, so it will be very convenient to get quick access to the remote compute nodes.

- I recommend however, that you push every setup to your project allocation userid. My motivation is that all of you should be set up for the long term on Piz Daint.

Remote Visualization

Outline

- VNC desktop for OpenGL-enabled graphics
- Dedicated ssh connections for ParaView and VisIt

Pre-requisites if you use the Hybrid or EGL partition

Local installation of

vncviewer

Paraview (v5.9)

VisIt (v3.12, or 3.2)

A **must-read** ☺: https://user.cscs.ch/access/faq/#access-and-accounting

VNC-based server

Run the script *RemoteAccess/vnc_desktop.sbatch*

- You will need a VNC password
- ssh to daint and run the command: *vncpasswd*

- A directory called $HOME/.vnc is created and contains your password and xstartup* files. A copy of my own xstartup file is located in *RemoteAccess/.vnc* and these are guaranteed to work. If the original files are not adequate, feel free to use my copies.

VisIt Remote Connection

This is not a tutorial about VisIt

- If you wish to have one for your group (minimum 8 persons),please let me know

- Assuming you know how to use VisIt…

- We have a brand new version of VisIt, and I'd love to see people use it
- Version 3.2 is available on Piz Daint

- For automatic connection, see the script file : RemoteAccess/VisIt-v3.2.0ConnectToPizDaint.py

ParaView Remote Connection

The ParaView GUI will set it up...

Documentation page on Paraview interactive-mode-with-a-client-server-connection

Accepting connection(s): rancate:1100

```
#SBATCH --job-name=pvserver
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=8
#SBATCH --time=00:20:00
#SBATCH --partition=debug
#SBATCH –account=csstaff
#SBATCH --constraint=gpu

srun -n 8 -N 1 --cpu_bind=sockets pvserver -rc -ch=daint103.cscs.ch -sp=1100
Submitted batch job  123456789
```

Local ParaView (on your desktop) + parallel pv server on Piz Daint

Local desktop

*from paraview.simple import **

ReverseConnect("1100")

N.B. The client is put in wait mode with the call above, **before** issuing the srun command on compute node(s)

Local desktop

```
ssh -l jfavre -R 1100:localhost:1100 daint101.cscs.ch
```

On Daint:

```
module load daint-gpu ParaView
```

```
srun -C gpu -p debug -n 4 -N 1 -A csstaff –t 05:00 \
      pvserver -rc -ch=`hostname` -sp=1100

srun: job 31186667 queued and waiting for resources
srun: job 31186667 has been allocated resources
Connecting to client (reverse connection requested)...
Connection URL: csrc://daint101.cscs.ch:1100
Client connected.
```

Automatic connection?

For automatic connection, see the script file : *RemoteAccess/pvParaView-v5.9ConnectToPizDaint.py*

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

ParaView Introduction

Introduction - Objectives

- Introduce ParaView and VTK
- Describe the VTK pipeline and VTK Objects

User Guide

check it out at least once: ☺ ☺ ☺

https://docs.paraview.org/en/latest/UsersGuide/index.html

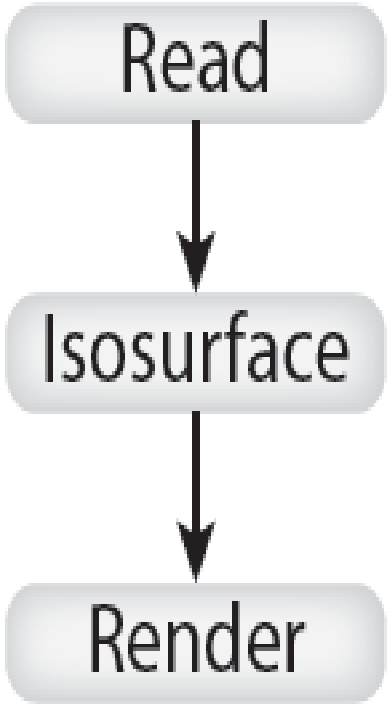
Discussion (Announcements, PV support, in-situ support, web support, …)

https://discourse.paraview.org/

Visualization Pipeline: Introduction

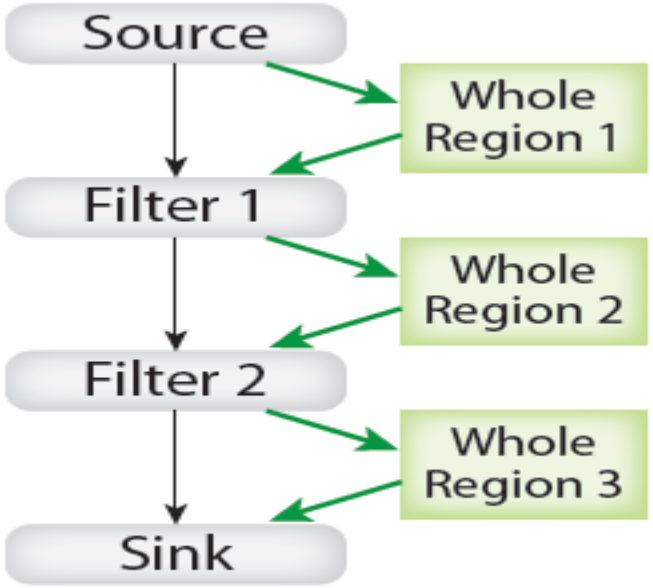
From a survey article by Ken Moreland, IEEE Transactions on Visualizations and Computer Graphics, vol 19. no 3, March 2013

«A visualization pipeline embodies a *dataflow network* in which computation is described as a collection of executable *modules* that are connected in a directed graph representing how data moves between modules. There are threee types of modules: *sources*, *filters* and *sinks*.»

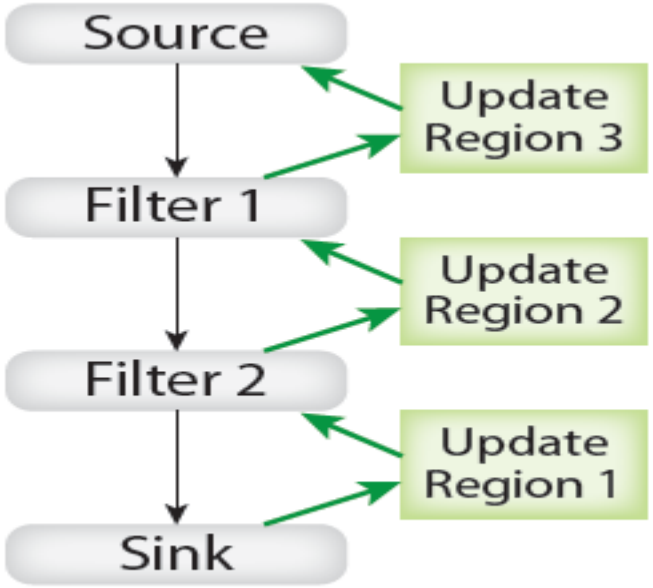


Visualization Pipeline: Metadata

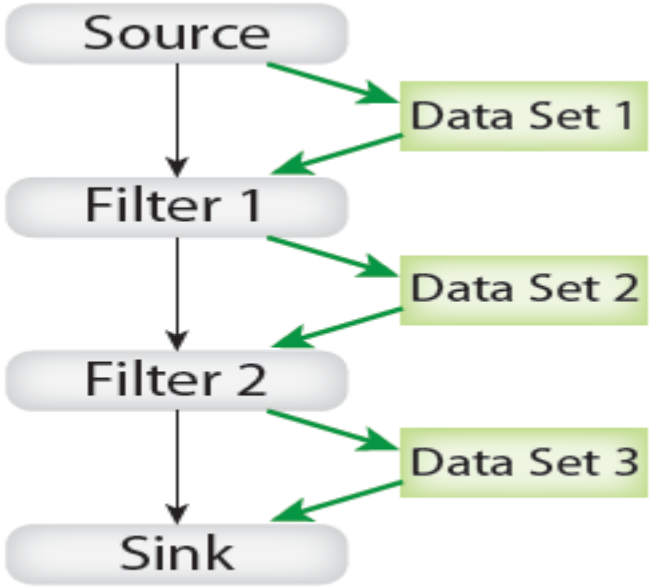
- 1st pass: Sources describe the region they can generate.
- 2nd pass: The application decides which region the sink should process.
- 3rd pass: The actual data flow through the pipeline



(a) Update Information



(b) Update Region



(c) Update Data

Why should you care?

- Using the ParaView GUI, you click Apply and everything works (luckily)
- When programming directly with Python, you may need to force the execution of the first pass, in order to properly set up the down-stream filters.

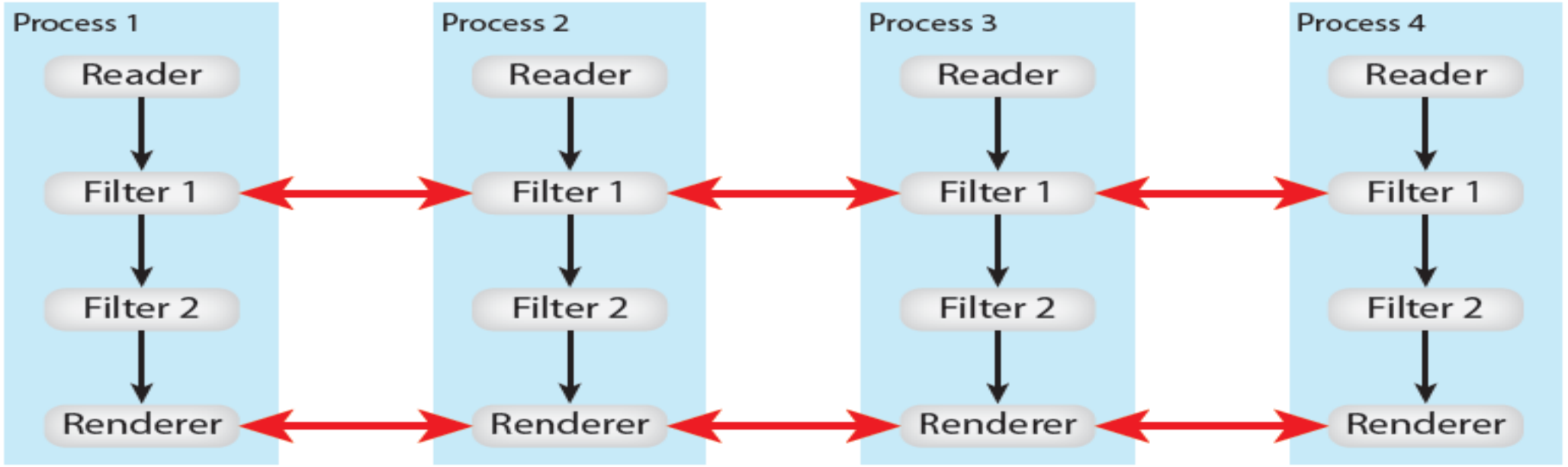
Source.UpdatePipelineInformation()

- Likewise, if you wish to force the execution of the pipeline,

Filter.UpdatePipeline()

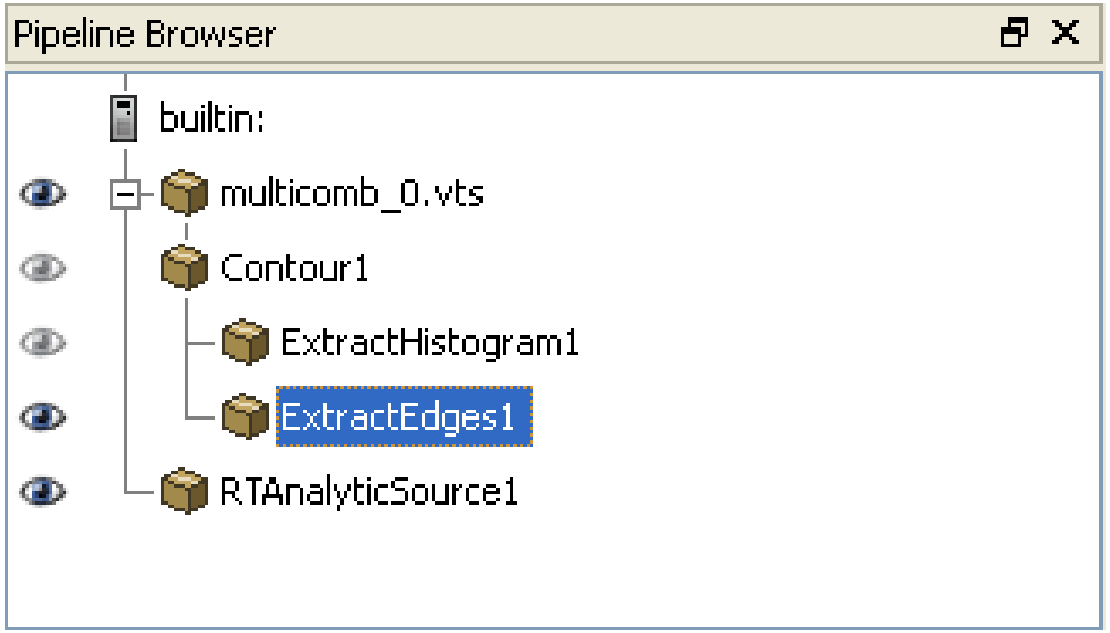
Visualization Pipeline: Data Parallelism

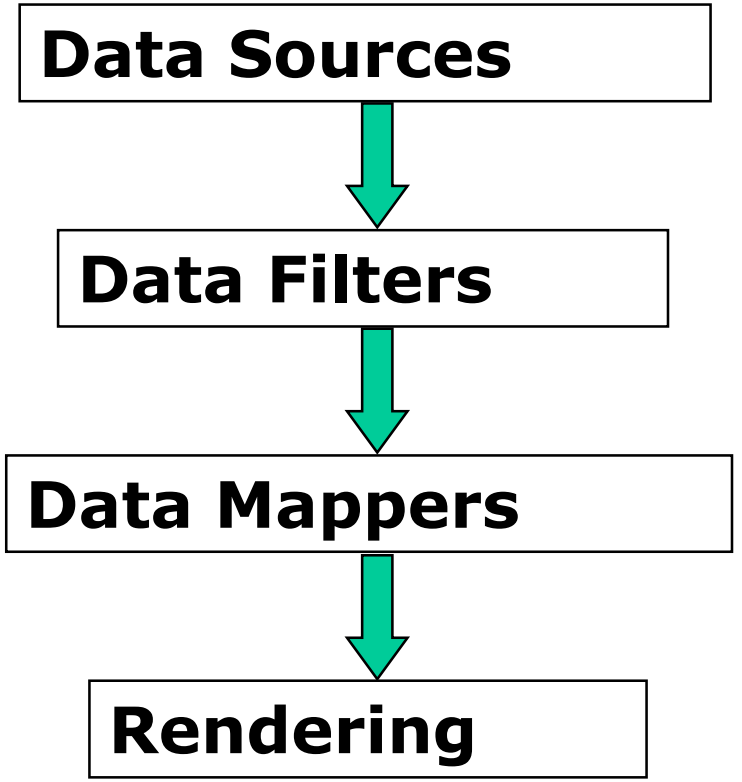
- Data parallelism partitions the input data into a set number of pieces, and replicates the pipeline for each piece.
- Some filters will have to exchange information (e.g. GhostCellGenerator)



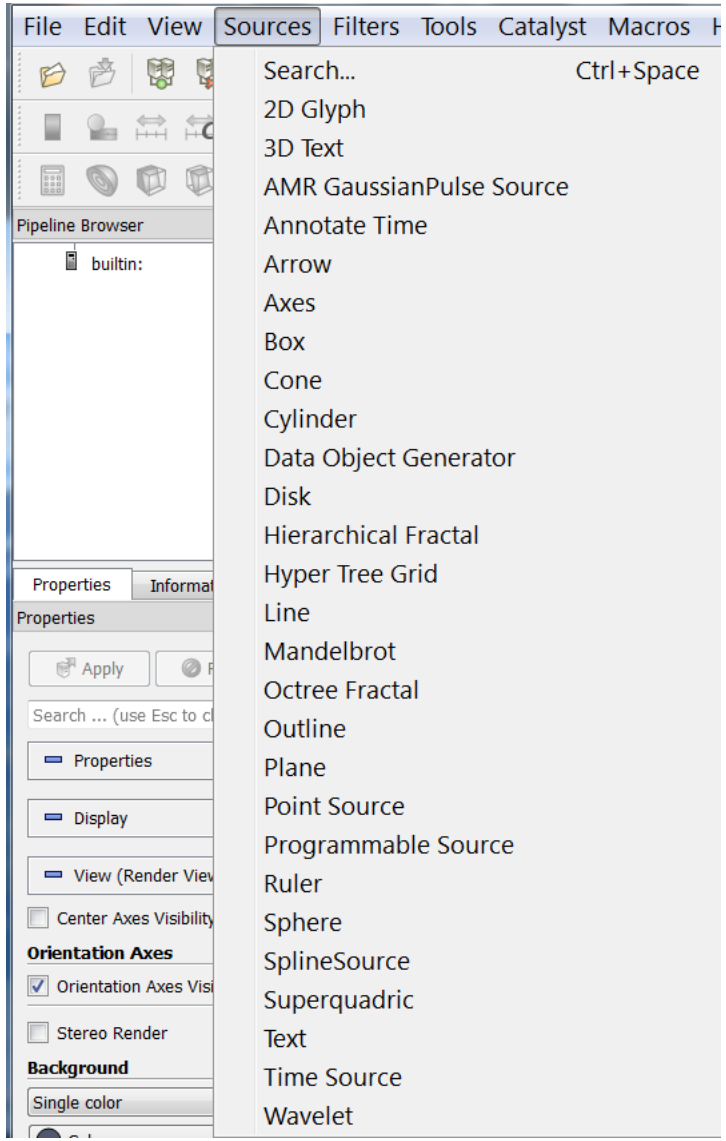
The VTK visualization pipeline

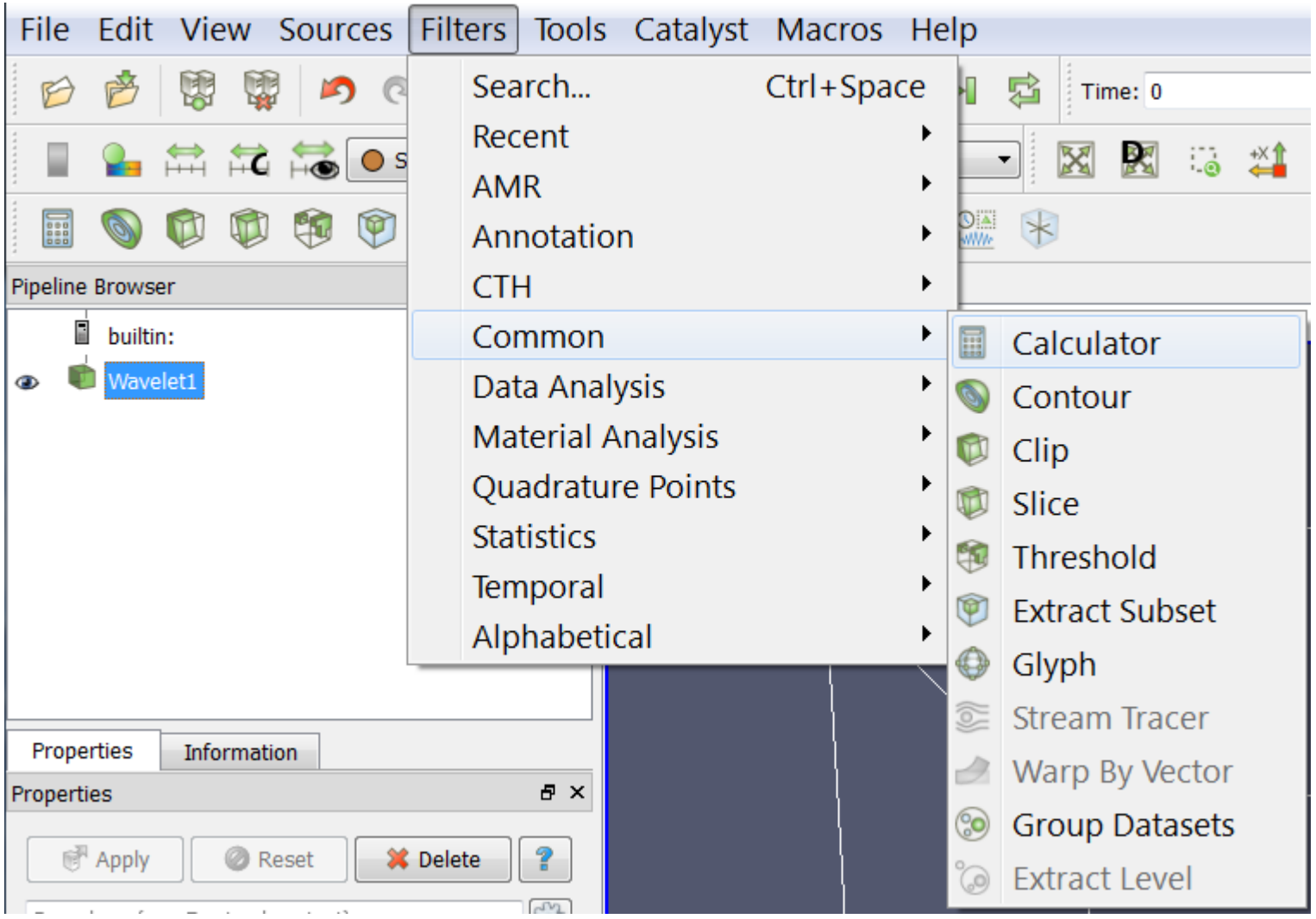
VTK's main execution paradigm is the *data-flow*, i.e. the concept of a downstream flow of data





Examples of Filters/Sources

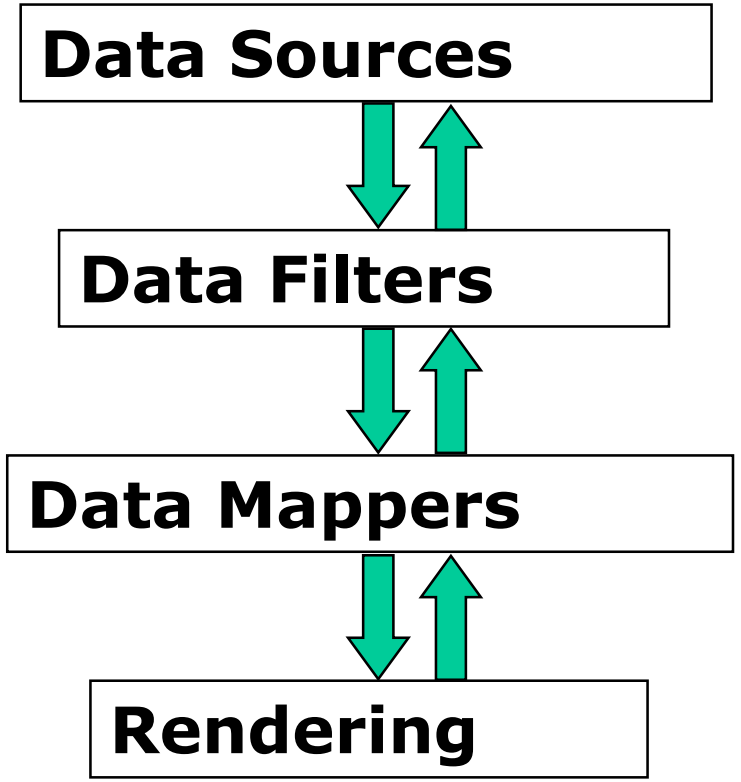




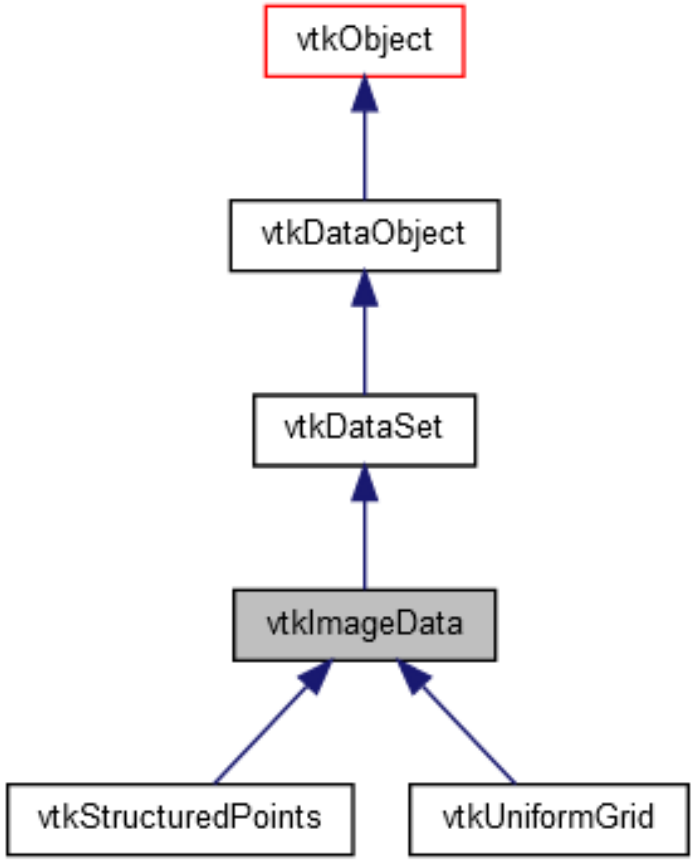
The VTK visualization pipeline

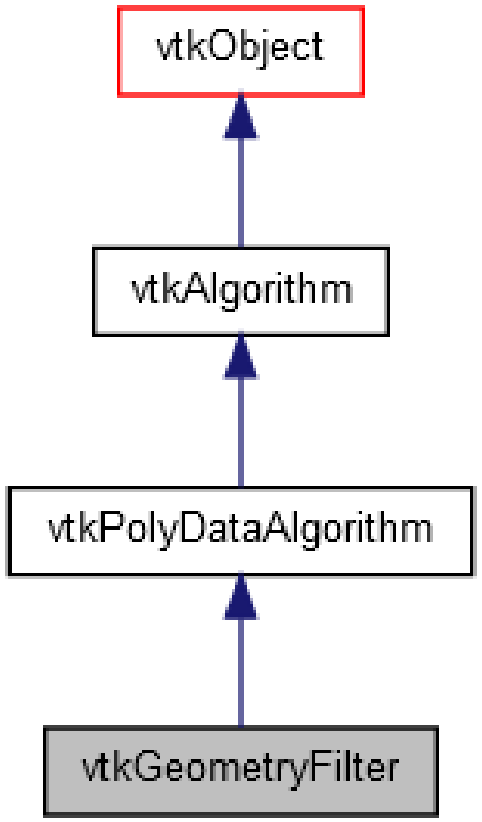
- VTK extends the *data-flow* paradigm
- VTK acts as an *event-flow* environment, where data flow downstream and events (or information) flow upstream
- ParaView's Rendering **triggers** the execution:

```
view = GetRenderView()
view.ViewTime = 5
Render()
```



vtkDataObject vs. vtkAlgorithm





CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

ParaView Python Scripting

Agenda from 13:30-14:30

- pvpython and pvbatch (15 min)
- ParaView in jupyter.cscs.ch (10 min)
- Python Programmable Sources/filters, Python Calculator (15 min)
- numpy interface (10 min)
- What is missing in Paraview’s trace (10 min)

Foreword

- ParaView is a very mature 3D parallel visualization ecosystem, in use at CSCS for many years.
- Users usually create a client-server connection from their remote desktop to a set of compute nodes on Piz Daint.
- ParaView uses an efficient and productive interface via Python scripts:
  - The client will read Python commands and the execution takes place [in parallel], on the server side
- A jupyter notebook can execute, stand-alone, or connected to a ParaView parallel server.

pvbatch, pvpython

- **pvpython** is the Python interpreter that runs ParaView's Python scripts. You can think of this as the equivalent of the paraview for scripting.

- Similar to **pvpython**, **pvbatch** is also a Python interpreter that runs Python scripts for ParaView. The one difference is that, while **pvpython** is meant to run interactive scripts, **pvbatch** is designed for batch processing. Additionally, when running on computing resources with MPI capabilities, **pvbatch** can be run in parallel.

Demonstration. Save a Python Trace

Running pvbatch on Piz Daint

- https://user.cscs.ch/access/running/jobscript_generator/
- module load ParaView
- srun *pvbatch* script.py

N.B

do not attempt to connect. *pvbatch* will automatically have a parallel server

- #Connect("…")

Moving to Jupyter.cscs.ch

- /users/jfavre/Projects/Visualization-training/ParaView

Pre-requisites valid for both the Hybrid or EGL partition

Edit your $HOME/.jupyterhub.env

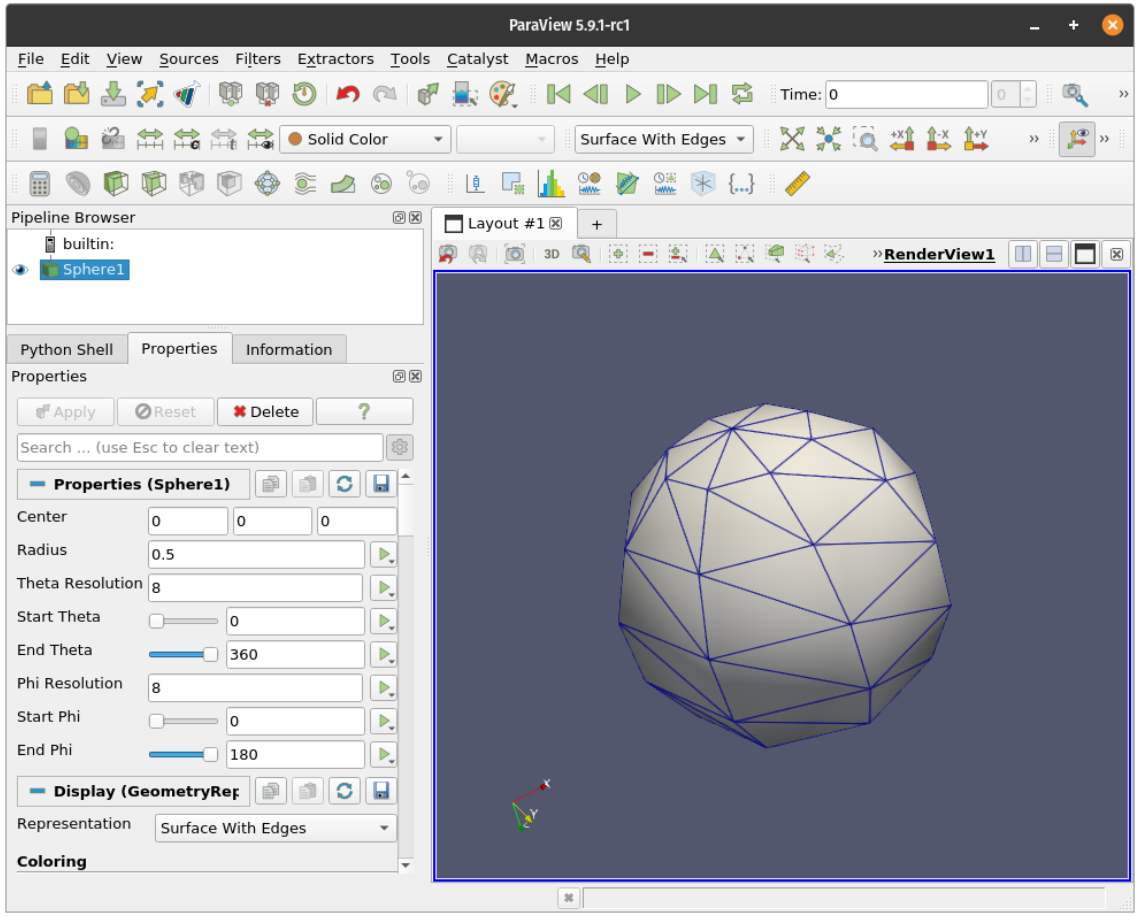
module load PyExtensions h5py/2.10.0-CrayGNU-20.11-python3-serial

module load ParaView

Presentation by Tim Robinson@cscs for all generic details (webinar of April 26,).

From your desktop to a Jupyter notebook

- The desktop ParaView app can give you a Python script with a new View, and all objects and their properties…
- You can bootstrap the process of writing a notebook with most of the Python code saved by the ParaView desktop app.
- The View will need a special handling.
- Live demonstration…



From your desktop……. to a Jupyter notebook

```
# Create a new 'Render View'
renderView1 = CreateView('RenderView')
renderView1.ViewSize = [512, 512]
renderView1.CameraPosition = [0.0, 0.0, 3.3]
renderView1.CameraFocalDisk = 1.0
renderView1.CameraParallelScale = 0.86


SetActiveView(None)


# create new layout object 'Layout #1'
layout1 = CreateLayout(name='Layout #1')
layout1.AssignView(0, renderView1)
SetActiveView(renderView1)
```

```
view = GetRenderView()
View.ViewSize = [512, 512]


# this is new


from ipyparaview.widgets import PVDisplay
pvdisp = PVDisplay(view)
w = display(pvdisp)
```

Hello_Sphere-ParaView.0.ipynb

- Standard ParaView Python initialization
- Standard pipeline
  - ParaView Source
  - ParaView Representation
  - Render

+

- PVDisplay widget (contributed by NVIDIA)

ParaView Hello Sphere

This notebook creates a synthetic data source (a sphere), and creates a polygonal display of it. T

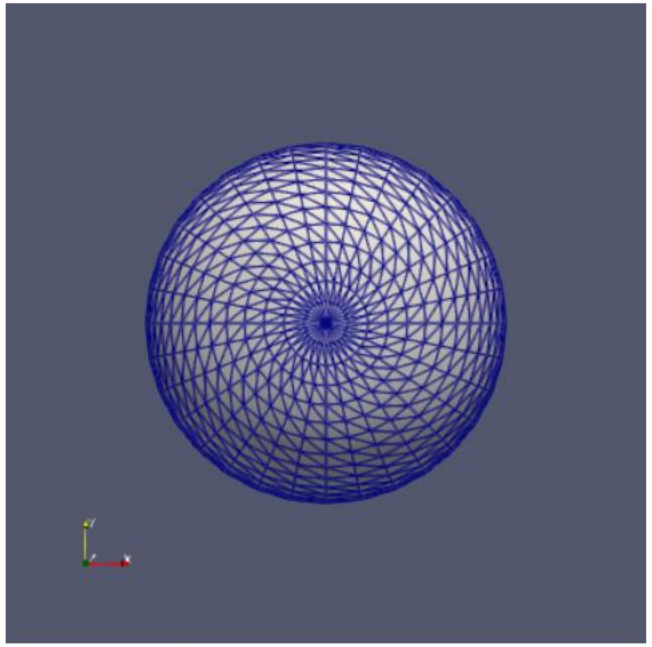
[1]:
```
from paraview.simple import *
```

[2]:
```
sphere = Sphere(ThetaResolution=32, PhiResolution=32)

rep = Show()
rep.Representation = "Surface With Edges"
```

[3]:
```
# create a default View
view = GetRenderView()
```

[4]:
```
# The ipyparaview is provided by NVIDIA. Thanks to Nick Leaf for his support
```

[5]:
```
from ipyparaview.widgets import PVDisplay
pvdisp = PVDisplay(view)
w = display(pvdisp)
```



Hello World (Sphere) augmented with ipywidgets

sphere.ListProperties()

Attach PhiResolution and ThetaResolution to an IntSlider

['Center',
'EndPhi',
'EndTheta',
'PhiResolution',
'PointData',
'Radius',
'StartPhi',
'StartTheta',
'ThetaResolution‘]

Hello World (Sphere) augmented with ipywidgets

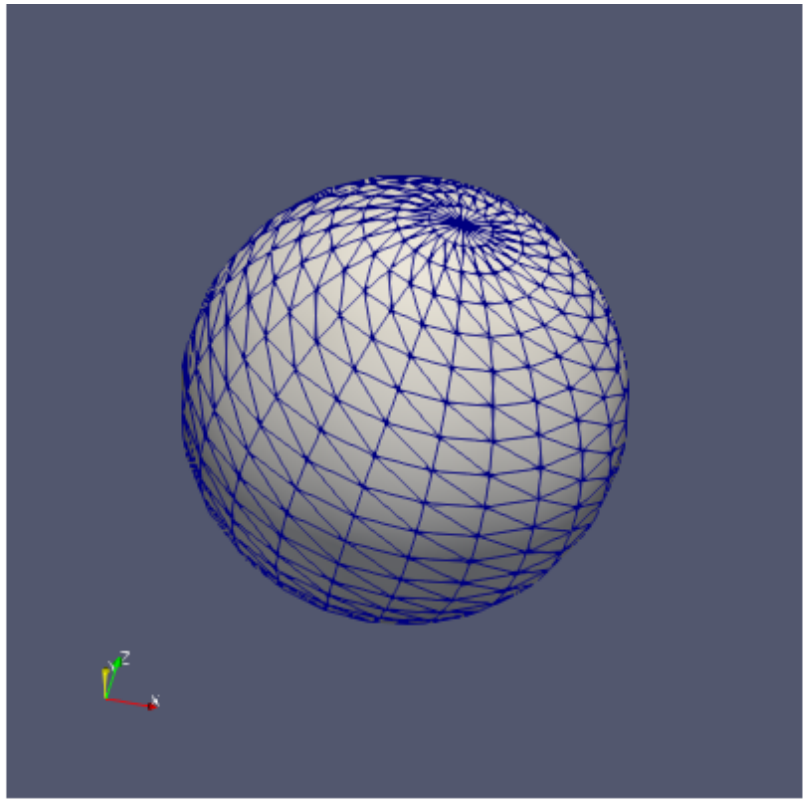
sphere.ListProperties()

Attach PhiResolution and ThetaResolution to an IntSlider

```
from ipywidgets import interact, IntSlider

# automatically triggers a pipeline update, and a render event
def Sphere_resolution(res):
    sphere.ThetaResolution = sphere.PhiResolution = res
    sphere.UpdatePipeline()

i = interact(Sphere_resolution,
                res=IntSlider(min=3, max=48, step=1, value=12)
               )
```

```
['Center',
'EndPhi',
'EndTheta',
'PhiResolution',
'PointData',
'Radius',
'StartPhi',
'StartTheta',
'ThetaResolution‘]
```

Hello_Sphere-ParaView.1.ipynb



[6]:
```
# Interact from ipywidgets gives us a simple way to interactively control values
# with a callback function
from ipywidgets import interact, IntSlider

# set the Theta and Phi resolution and trigger a pipeline update
def Sphere_resolution(res):
    sphere.ThetaResolution = sphere.PhiResolution = res
    sphere.UpdatePipeline()

i = interact(Sphere_resolution, res=IntSlider(min=3, max=48, step=1, value=12))
```


res 24

Caveat

The standard SaveScreenshot() no longer works

Nick Leaf of NVIDIA has been very helpful in debugging the issue and for the moment, we have agreed to use the following code instead.

```
def SaveImage(filename):
  from vtk import vtkPNGWriter
  img_writer = vtkPNGWriter()
  img_writer.SetInputConnection(disp.w2i.GetOutputPort())
  img_writer.SetFileName(filename)
  img_writer.Write()

SaveImage("/users/jfavre/screenshot.png")
```

From the notebook to a standard Python script

jupyter nbconvert --to python file.ipynb

⇒ creates file.py

```
# comment out 3 lines
“““
from ipyparaview.widgets import PVDisplay
pvdisp = PVDisplay(view)
w = display(pvdisp)
”””
```

Molecular Data Animation. Hello_Molecule.ipynb

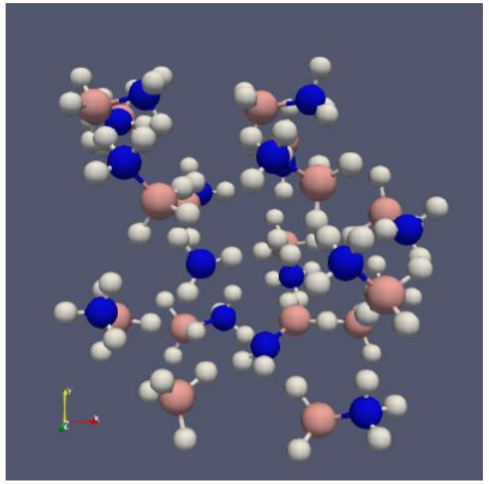
[1]:
```
from paraview.simple import *
```

[2]:
```
molecule1 = XYZReader(FileName='/users/jfavre/Projects/Rizzi/release_H2_ex.xyz')
nb_of_timesteps = len(molecule1.TimestepValues)
print("Molecule trajectories with ", nb_of_timesteps, " steps")
```

```
Molecule trajectories with  1500  steps
```

[3]:
```
computeMoleculeBonds1 = ComputeMoleculeBonds(Input=molecule1)
computeMoleculeBonds1.UpdatePipeline()
computeMoleculeBonds1Display = Show(computeMoleculeBonds1, GetActiveView())
```

[4]:
```
from ipyparaview.widgets import PVDisplay
disp = PVDisplay(GetActiveView())
w = display(disp)
```



[5]:
```
# Interact from ipywidgets gives us a simple way to interactively control values
# with a callback function
from ipywidgets import interact, IntSlider

# set a new time-step
def time_slider(t):
    GetActiveView().ViewTime = t

i = interact(time_slider, t=IntSlider(min=0, max=nb_of_timesteps-1, step=1, value=0))
```

t 568

- Given a time aware data source called “reader”
- reader.TimestepValues is defined for all readers
- The ActiveView’s ViewTime variable is what triggers a timestep update

Data interface to your favorite (numpy) data arrays

From arrays to VTK grid objects

See Visualization-training/ParaView/vtkGridConstructors.py …

is a prototype to use different types of [numpy] arrays and create the proper grid structures from the VTK world.

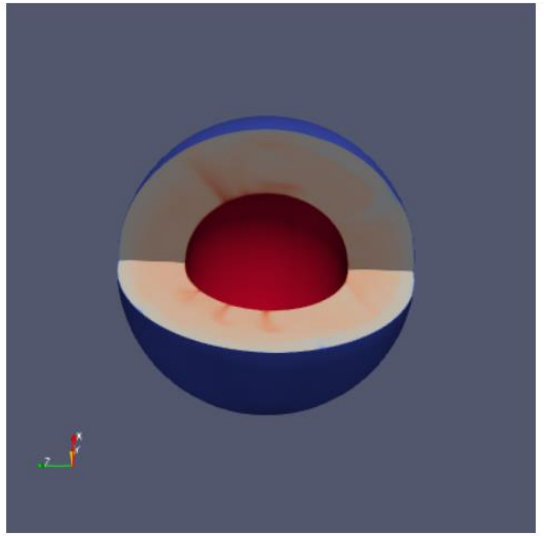
Please try it, report issues, contribute…

Example:

pvTestGridConstructors-ParaView.ipynb

Example with xarray

- Xarrays-ParaView-Import.ipynb



```
[3]: import xarray as xr
data = xr.open_dataset('/scratch/snx3000/jfavre/SciVis2021/spherical001.nc')
print(data)

P = (90. - data.lat.values) * np.pi / 180.
R = data.r.values
T = data.lon.values * 2 * np.pi / 360.

p, r, t = np.meshgrid(P, R, T, indexing="ij")
X = r * np.cos(t) * np.sin(p)
Y = r * np.sin(t) * np.sin(p)
Z = r * np.cos(p)

SphericalGrid = vtkStructuredGridFromArrays()

SphericalGrid.SetCoordinates(X, Y, Z)

#add scalar variables
scalar_vars=["temperature",
             "temperature anomaly",
             "thermal conductivity",
             "thermal expansivity",
             "spin transition-induced density anomaly"]

for var in scalar_vars:
  SphericalGrid.AddArray(data.data_vars[var].values.ravel(), var)

# Add velocity (rescaled) as a vector
SphericalGrid.AddArray(1.0e9*algs.make_vector(data.vx.values.ravel(),
                                              data.vy.values.ravel(),
                                              data.vz.values.ravel()), "velocity")
```

Programmable filters

- https://docs.paraview.org/en/latest/ReferenceManual/pythonProgrammableFilter.html#

- custom code for filters and sources.
- Knowledge of the basics of a VTK (and ParaView) data processing pipeline, including the various stages of the pipeline execution as well as the data model is required.

Example. Read the compressed binary data use with matplotlib earlier

- Python Calculator

- See Visualization-training/ParaView/ParaView-ProgrammableSource.Isabel.ipynb

Epilog

What is missing in ParaView's trace

Using an animation?

Given a time-aware source called "reader"

```
scene = GetAnimationScene()
scene.UpdateAnimationUsingDataTimeSteps()
scene.NumberOfFrames = len(reader.TimestepValues)
scene.PlayMode = 'Snap To TimeSteps'



SaveAnimation('/scratch/snx3000/jfavre/paraview2.png')

SaveData('fname.vti', Writetimestepsasfileseries=1,
                      Filenamesuffix='_%02d')
```

What is missing in ParaView's trace

Using a query selection?

extractSelection1 = ExtractSelection(registrationName='ExtractSelection1', *input=inputdata*)

The Query is missing!

q1 = QuerySelect(QueryString="(mag(Wind) >= 50.) &
(TCf <= 0.)")

extractSelection1 = ExtractSelection(registrationName='ExtractSelection1', input=inputdata, selection=q1)

What is missing in ParaView's trace

Doing a camera keyframe animation, using PythonAnimationCue(s)?

All keyframes will be lost!

*Use a help file written by the CSCS local Paraview expert. It's "**Work in progress**" and does not cover 100% of all cases.*

See Visualization-training/ParaView/*pvSaveKeyFrames.py*

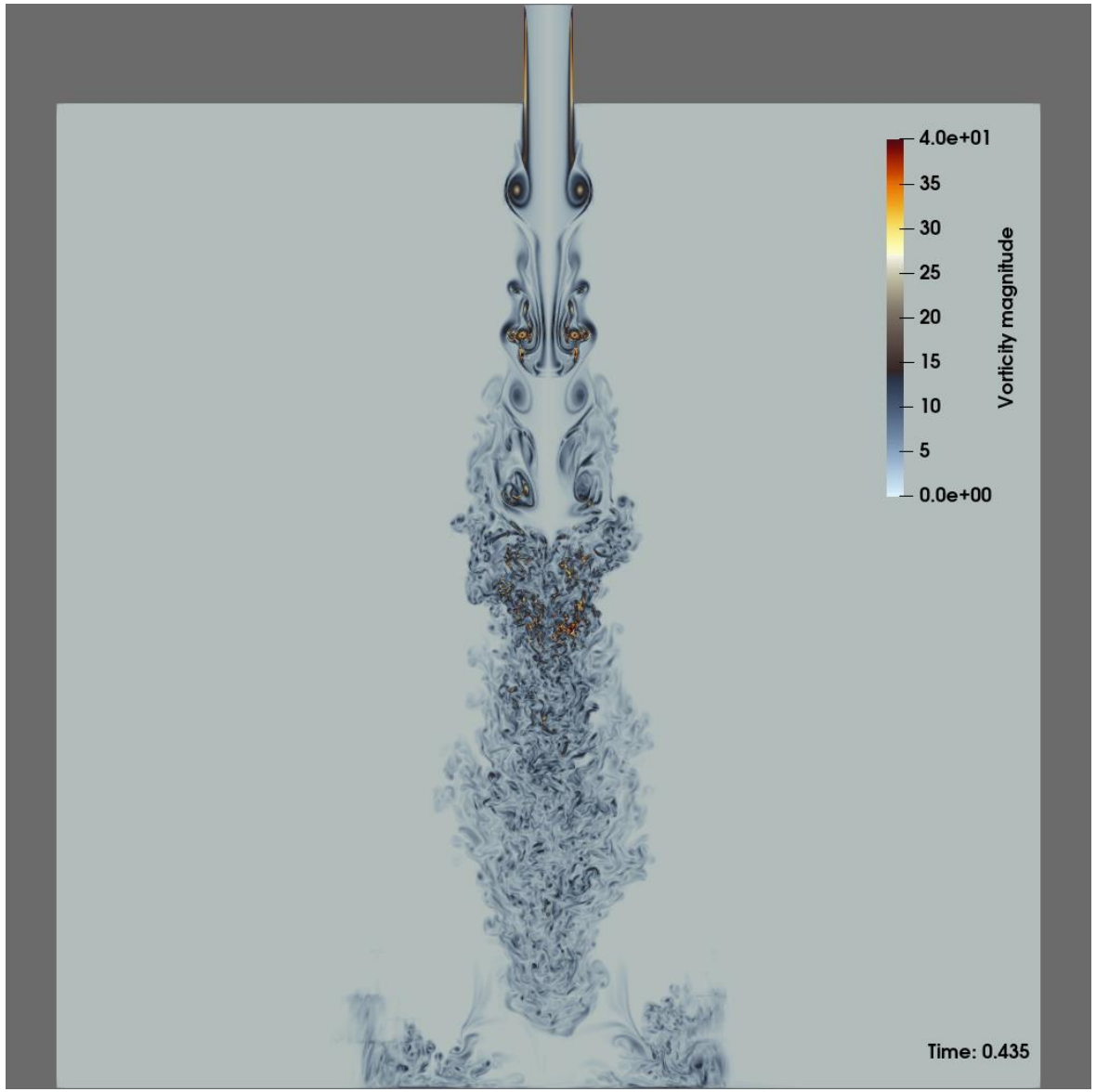
ParaView/VTK Volume Rendering

Introduction – Objectives 3:00pm-3:55pm

- Give users practical tips about Volume Rendering
- Discuss Mesh types and Data types
- Write python pipelines for VTK and ParaView
- Present NVIDIA IndeX plugin
- Present Intel OSPRay plugin

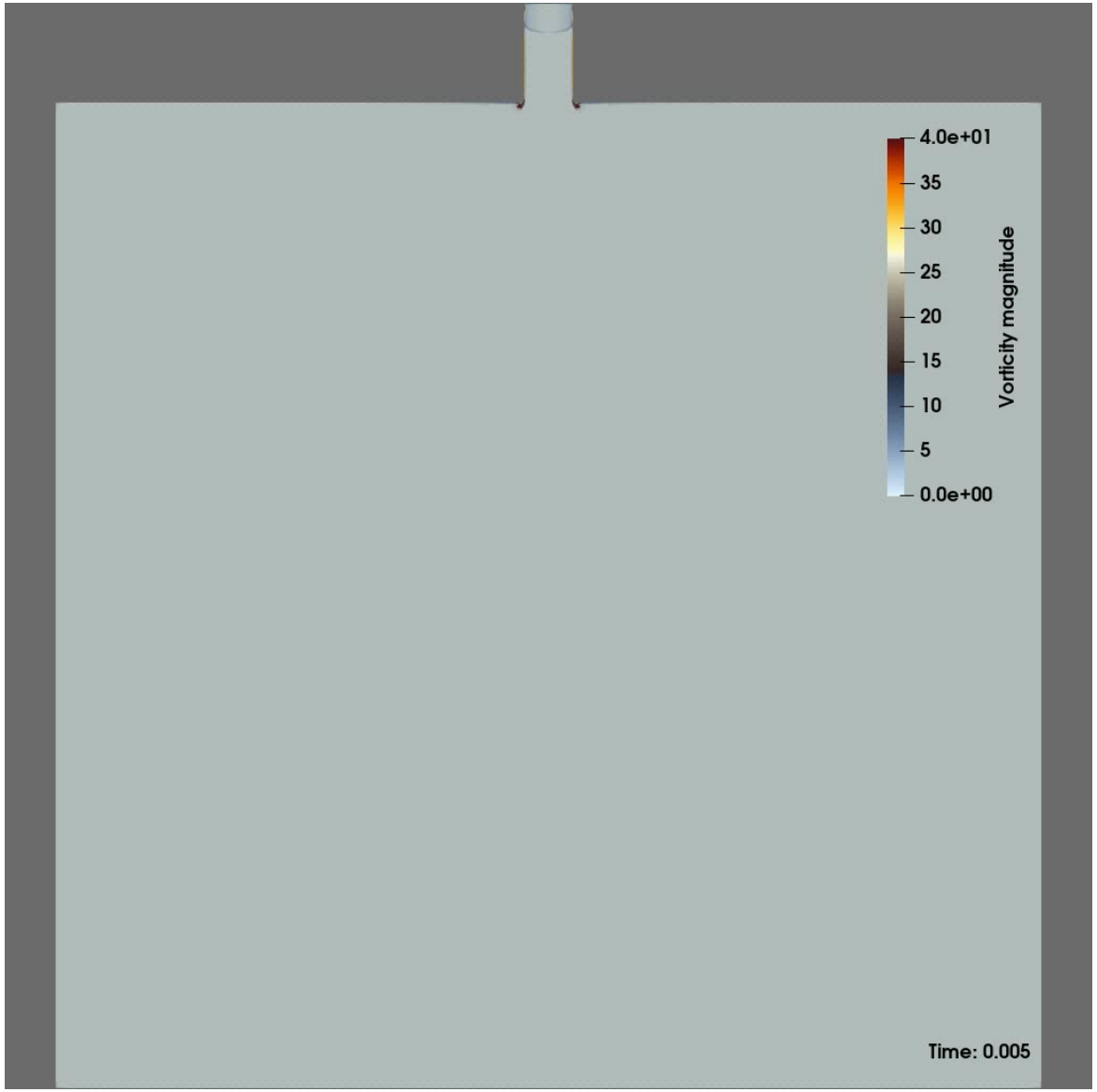
Volume Rendering showcase: Turbulent Jet Ignition

- Turbulent Jet Ignition (TJI) is a technique to enhance ignition via a jet of hot reactive gases generated in a small pre-chamber. The jet initiates combustion in the main chamber, improves combustion efficiency and reduces pollutant emissions.



Volume Rendering showcase: Turbulent Jet Ignition

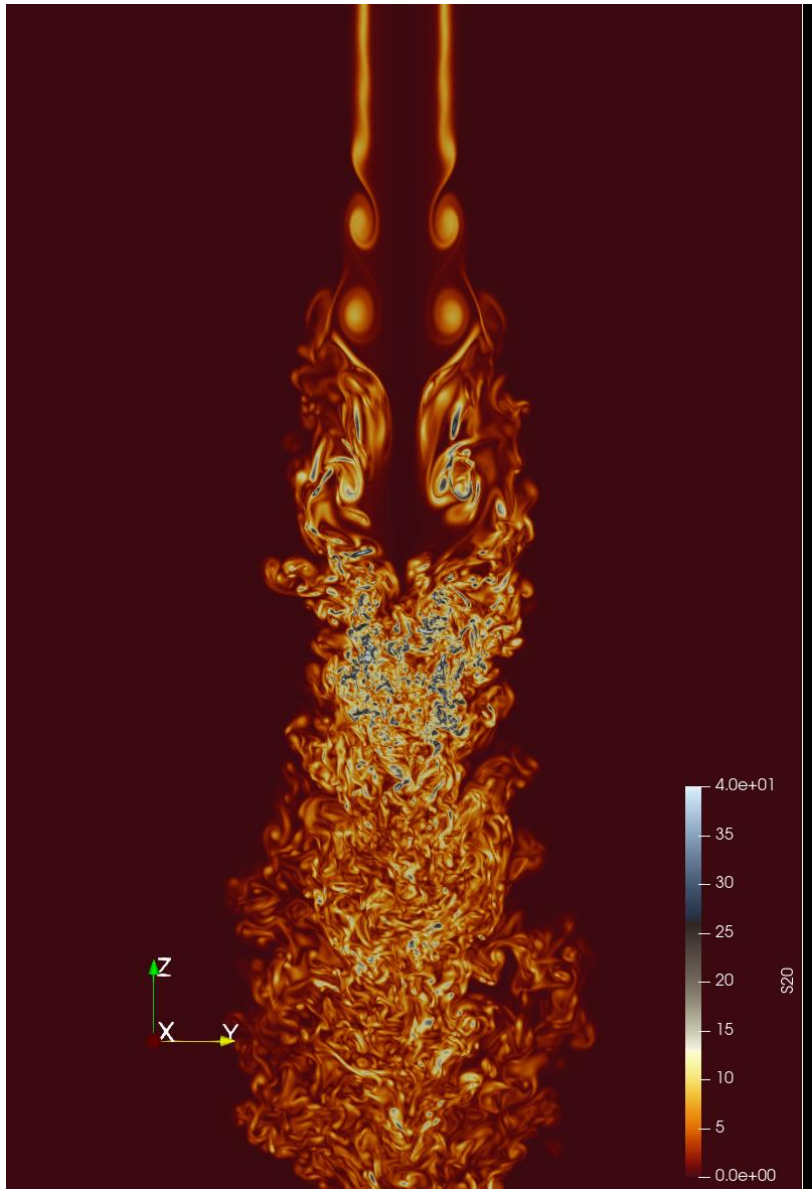
- Numerical computations were conducted by a team of ETH Zurich led by Dr. Christos Frouzakis, from the Aerothermochemistry and Combustion Systems Laboratory (LAV).
- Simulations used the spectral element solver Nek5000, enhanced by a combustion plugin developed at ETH Zurich. Computations used up to 822k spectral elements and 7-th order polynomials. We concentrated the visualization on the flow, looking at vorticity magnitude.



The need for 3D interactive 3D volume rendering

- Looking at 2D slices through a volume is not enough to capture the complex interactions taking place. A 3D interactive visualization is a must, and a best fit for the high resolution 3D data

- NVIDIA IndeX is the only solution we know of that can deliver interactive 3D Volume Rendering of unstructured grids distributed over a cluster of GPUs

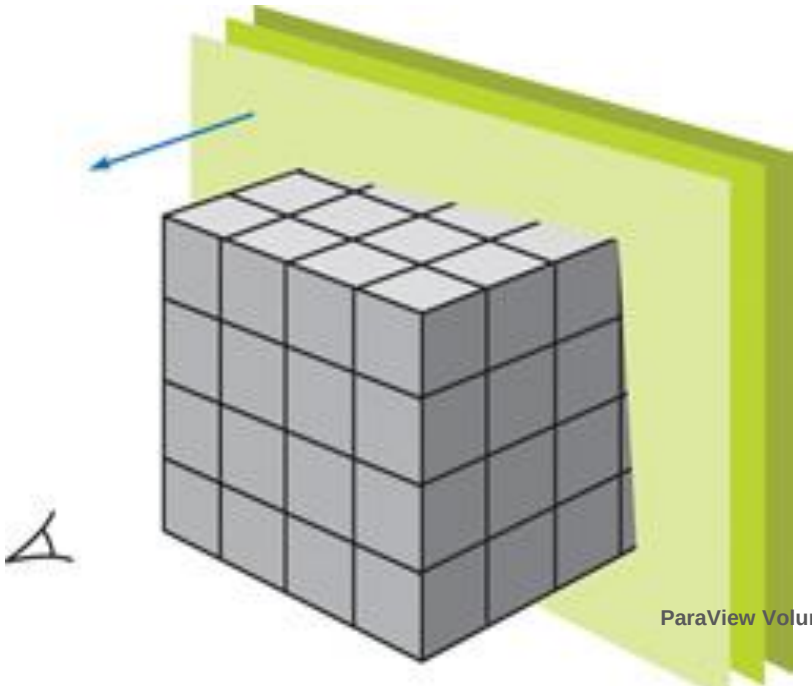


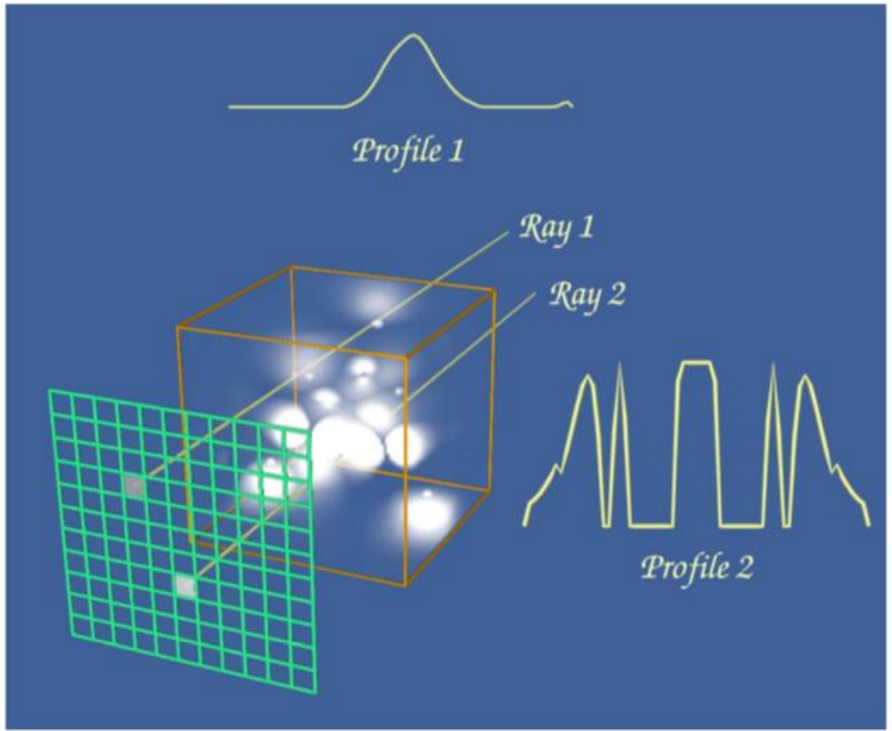
cscs

Volume Rendering (VR)

A technique for visualizing 3D volumetric data sets generated by medical imaging techniques such as computed tomography (CT), magnetic resonance imaging (MRI), and 3D ultrasound. VR is also used in many scientific fields … to visualize volumetric data.

Texture-based or Ray-casting-based





Pseudo Volume Rendering

Volume Rendering Essentials

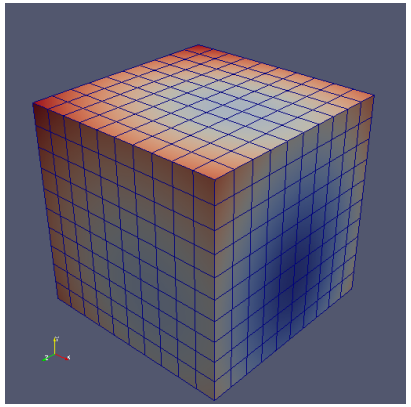
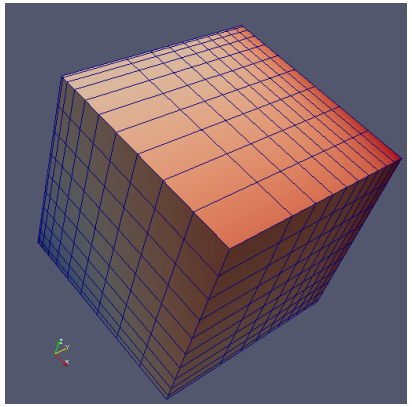
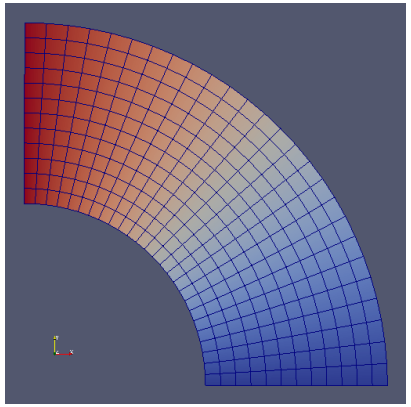
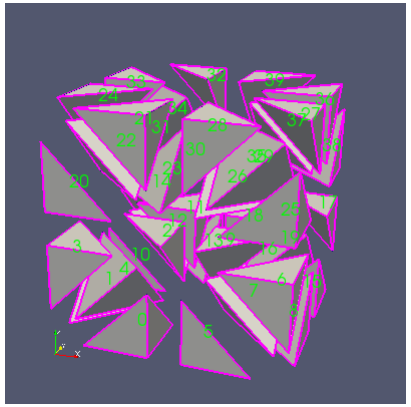
- https://www.kitware.com/products/books/VTKTextbook.pdf (Page 218)

- See “inside” the volume.

VTK Mesh Types

Not all grid types are supported:

- vtkImageData: Yes
- vtkRectilinearGrid: Yes
- vtkStructuredGrid: No
- vtkUnstructuredGrid: Yes

VTK DataArray Types

- Data Array types are not all supported, or will be transformed under the cover (e.g. *float32* vs. *Double)*

- Be aware of what you are sending down the pipeline.

- Example from NVIDIA IndeX

Datasets in double format are not natively supported by IndeX.

Converting scalar values to float format...

VTK DataArray Types

- If a data type is not supported, you can convert it with a simple ParaView Python Programmable Filter

```
from paraview.vtk.vtkImagingCore import vtkImageShiftScale
from numpy import iinfo
irange = inputs[0].PointData["MetaImage"].GetRange()
lmin = irange[0]
lmax = irange[1]
readerSS = vtkImageShiftScale()
 readerSS.SetInputData(inputs[0].VTKObject)
 readerSS.SetShift(lmin * -1)
 readerSS.SetScale(iinfo("ushort").max / (lmax - lmin))
 readerSS.SetOutputScalarTypeToUnsignedShort()
 readerSS.Update()
 output.ShallowCopy(readerSS.GetOutput())
```

Pseudo Volume Rendering 1

- ./ParaView/pvPseudoVolumeRendering.py

- # Perform pseudo volume rendering in a structured grid by compositing
- # translucent cut planes. This same trick can be used for unstructured
- # grids. Note that for better results, more planes can be created. Also,
- # if your data is vtkImageData, there are much faster methods for volume
- # rendering.

Pseudo Volume Rendering 2

- Select „PLOT3DReader1“ and create a slice perpendicular to Y axis
- Create many slices (e.g. 40)
- See the Color Map Editor „*Enable opacity mapping for surface*“

Volume Rendering the StructuredGrid ?

- Select „PLOT3DReader1“ and switch Representation to “Volume”

Error: Can not execute simple algorithm without output ports

- Tetrahedralize and switch to ‘Volume”

- The default mapper is the “OpenGL Projected Tetrahedra Mapper”

- The “Z sweep”, “Bunyk ray cast” are historical artifacts (very slow, or slow)
- The last option is “Resample to Image”

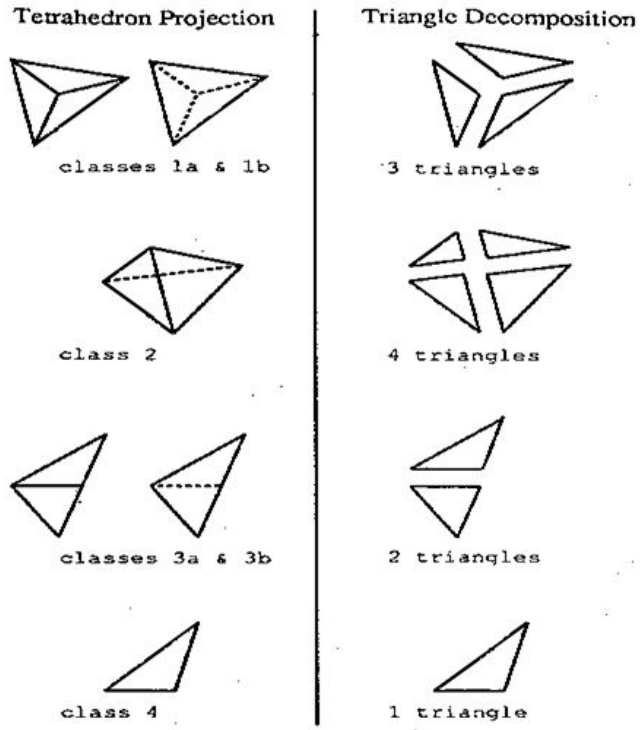
Volume rendering of Unstructured grid

- Default mapper “Projected Tetra”
- Almost 30 years old.

- Although today’s implementation are very sophisticated, using OpenGL shaders and other GPU optimizations techniques, it remains a slow rendering technique because of the sheer size of the triangles in the scene

PT - Decompose

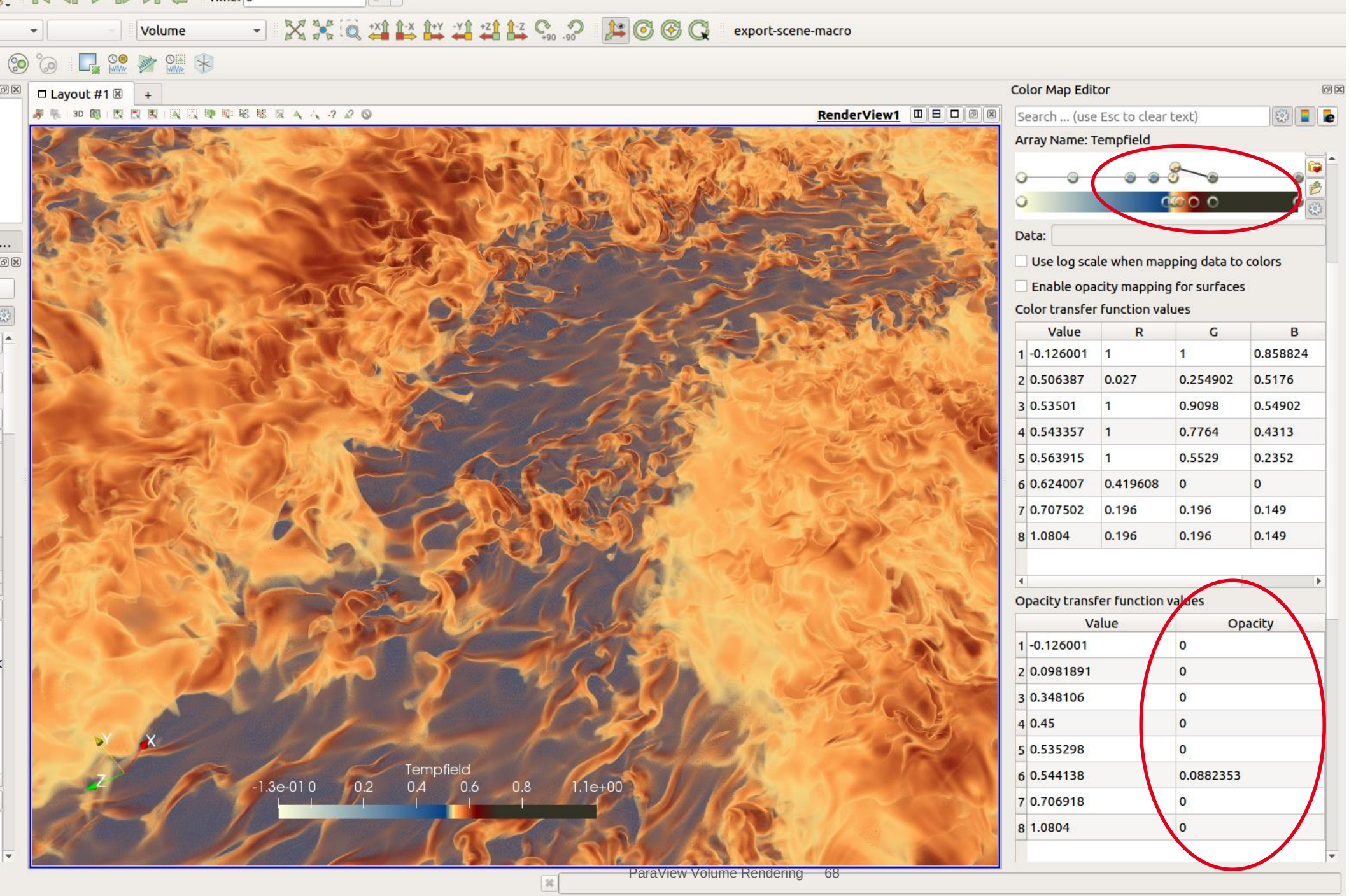
- Decompose projections of the tetrahedra into triangles
- Find the triangle vertex positions (tetrahedron vertex or edge intersection) after perspective projection



Resample to Image

- Resample to Image is multi-threaded. Will use all available cores

- It creates a vtkImageData (very well supported for Volume Rendering)



Volume rendering is a technique which requires a bit of fine-tuning

ImageData Volume rendering has three parallel libraries

- Kitware's native GPU-based renderer
- NVIDIA IndeX GPU-based renderer
- Intel OSPRay CPU-based renderer (using TBB multi-threading)

- Published in August 2019

Volume rendering tested with three parallel libraries

- Running on 1 node, with a 1024^3 volume:
- Memory consumption by GPU (see nvidia-smi)
  - Kitware's native renderer: 4252 MiB
  - NVIDIA Index Volume Renderer: 4810 MiB

- GPU-based rendering will scale across nodes, at the condition that it fits in memory

- Intel OSPRay CPU-based renderer can use a lot more memory (thus read much bigger data). It's on the CPU.

Volume rendering of Unstructured grid. A new mapper

- vtkUnstructuredGridVolumeRayCastMapper is a new VTK mapper
- Software ray-caster

NVIDIA IndeX for ParaView

- https://developer.nvidia.com/index

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

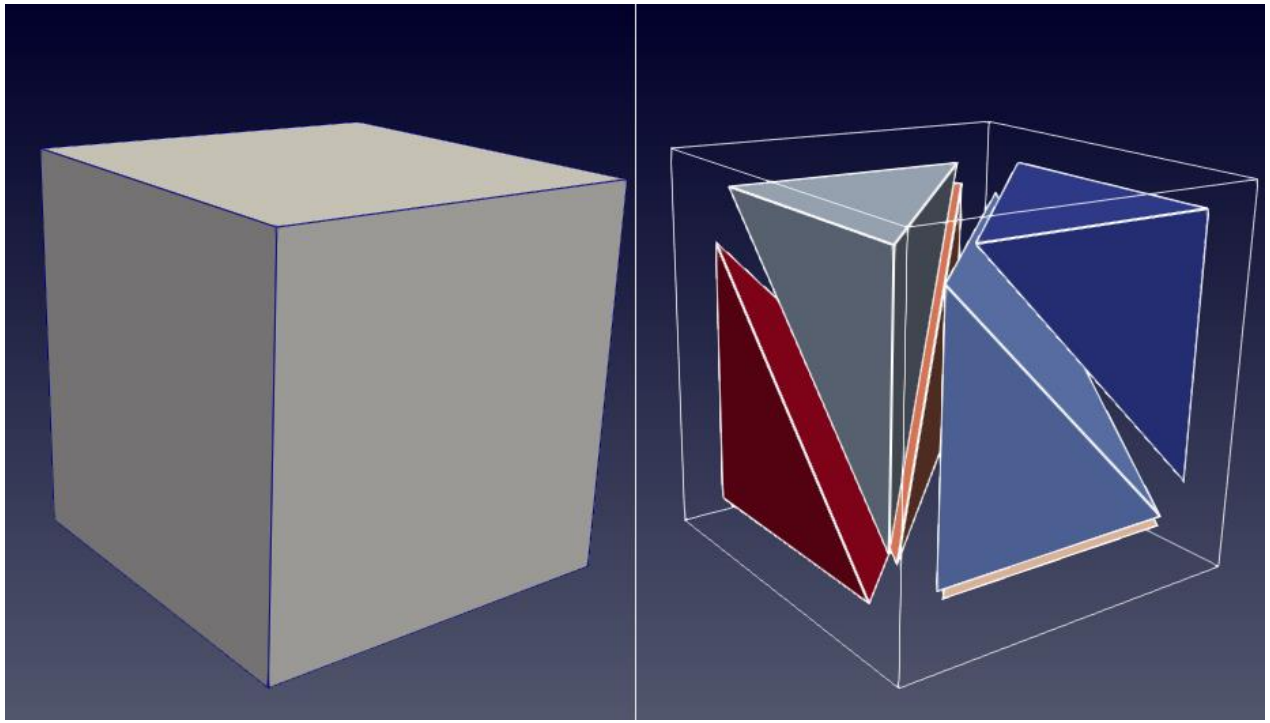
- NVIDIA IndeX supports different datatype formats such as unsigned char, unsigned short and floating point.

NVIDIA IndeX unstructured grid volume renderer

converts hexahedra to tetrahedra

#-of-tetrahedra = 6 x #-of-hexahedra

200 million hexahedra => 1.2 billion tetrahedra...



NVIDIA IndeX now capable of volume rendering billions of cells

Running on Piz Daint with Cray compute nodes and NVIDIA P100 GPUs

We have looked at two Turbulent Jet Ignition cases:

	Medium resolution	High resolution
# spectral elements	444,320 (kernel 8^3)	584,304 (kernel 12^3)
# of hexahedra	152,401,760	777,708,624
# of tetrahedra	914,410,560	4,666,251,744
# of GPUs	16-32	64

The full movie

Movie is computed in batch mode, with a scripted animation to move the camera around

The movie can be found on our CSCS YouTube channel

https://www.youtube.com/user/cscsch/videos

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 four visual presets available, each preset has one ore more parameters for finer control over that visual element

- None
- Isosurface (really, isovolume)
- Depth enhancement
  - Isolating features with high opacity values in the transfer function mapping
- Edge enhancement
  - enhance the edges or the "silhouettes" based on the transfer function setting
- Gradient

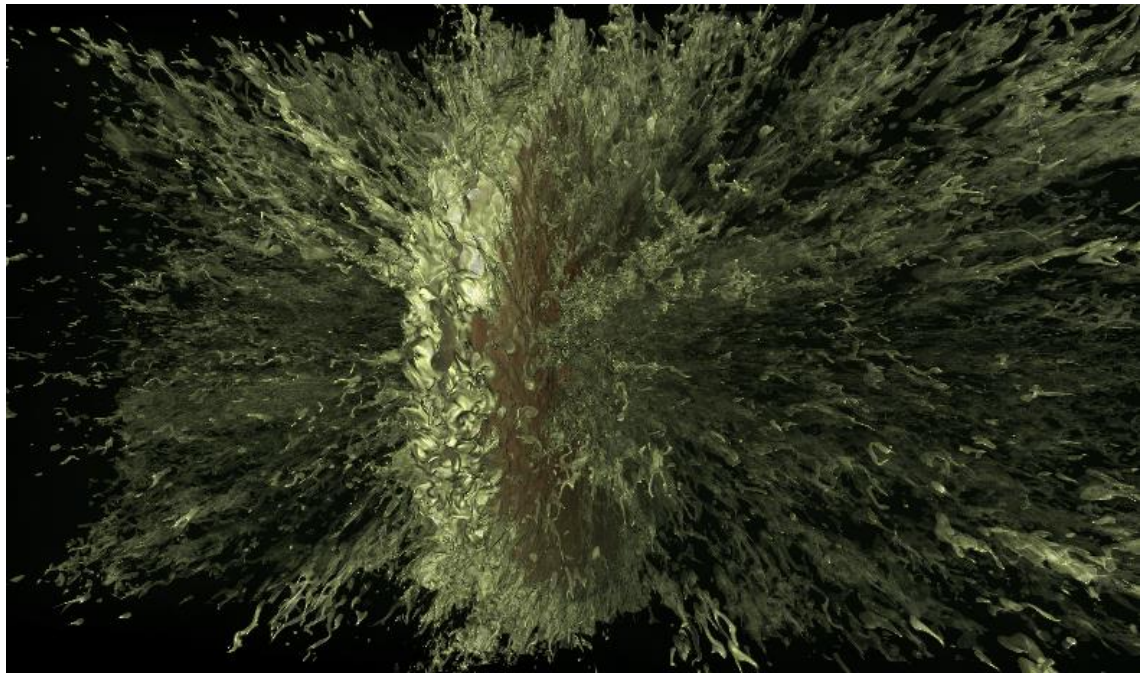
- The next 3 slides with images were kindly provided by the NVIDIA team (Eduardo Olivares and Alexander Kuhn)

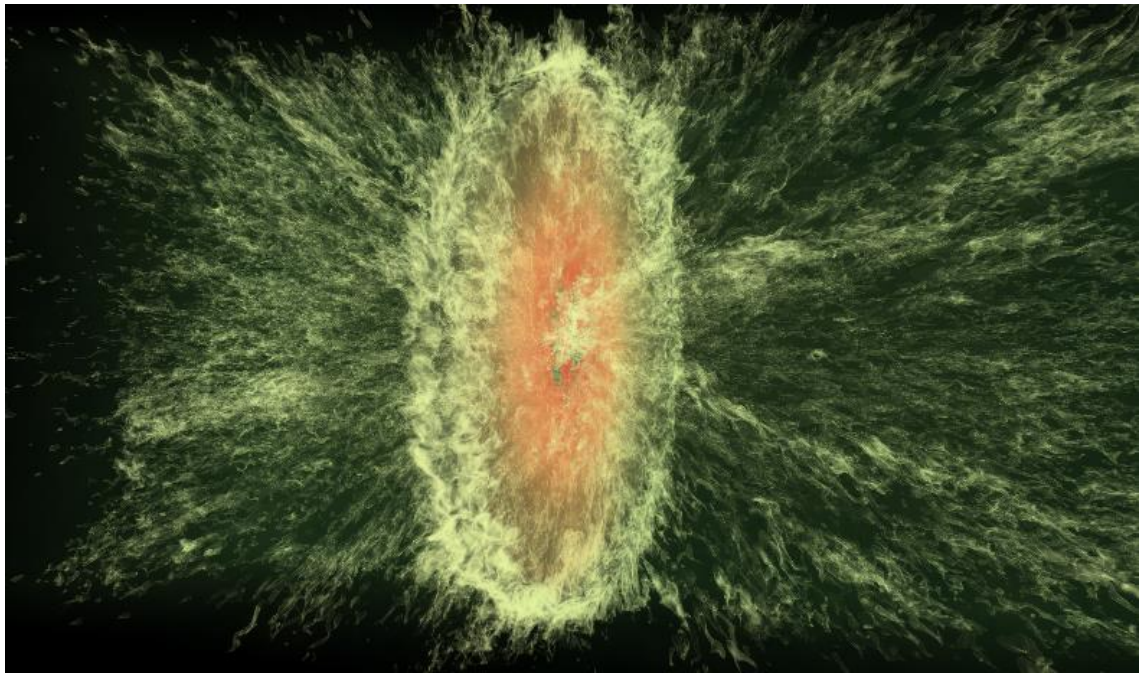
- The dataset is one time step of the CHOLLA galactic outflow simulation that was kindly provided by Evan E. Schneider (Princeton University) and Brant Robertson (UC Santa Cruz):

- [1] Schneider EE, Robertson BE. Introducing CGOLS: The Cholla Galactic Outflow Simulation Suite. The Astrophysical Journal. 2018 Jun 20;860(2):135.
- [2] Schneider EE, Robertson BE. CHOLLA: a new massively parallel hydrodynamics code for astrophysical simulation. The Astrophysical Journal Supplement Series. 2015 Apr 10;217(2):24.

NVIDIA IndeX visual elements: Depth enhancement

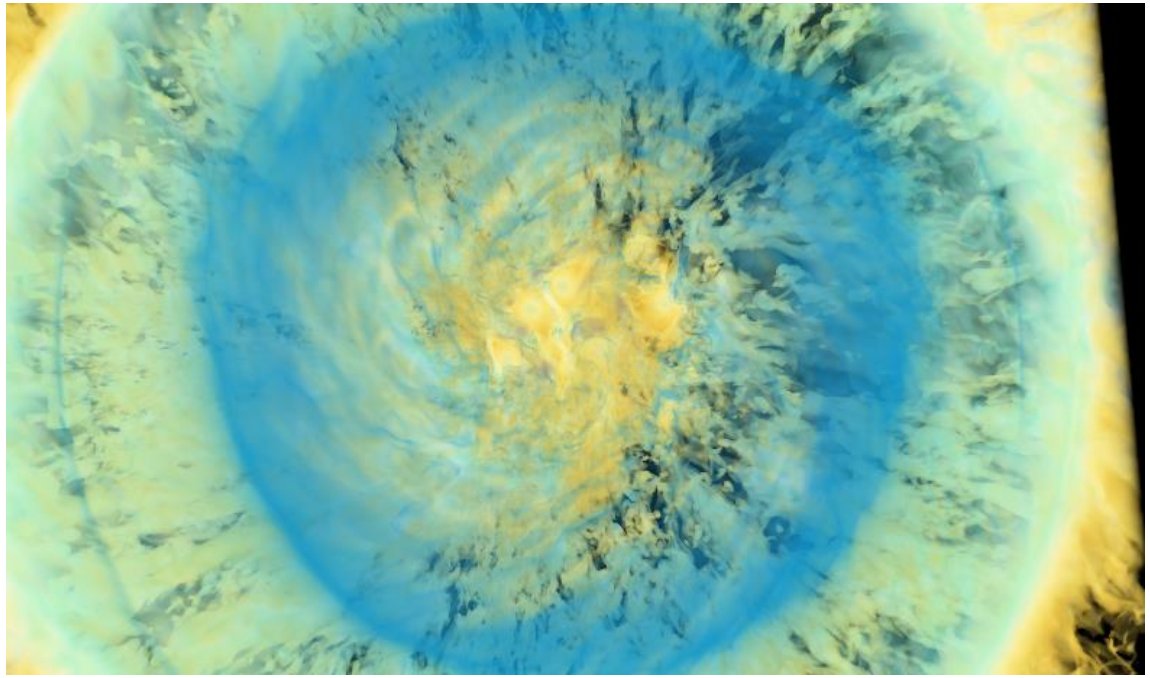
Enhancing depth cues

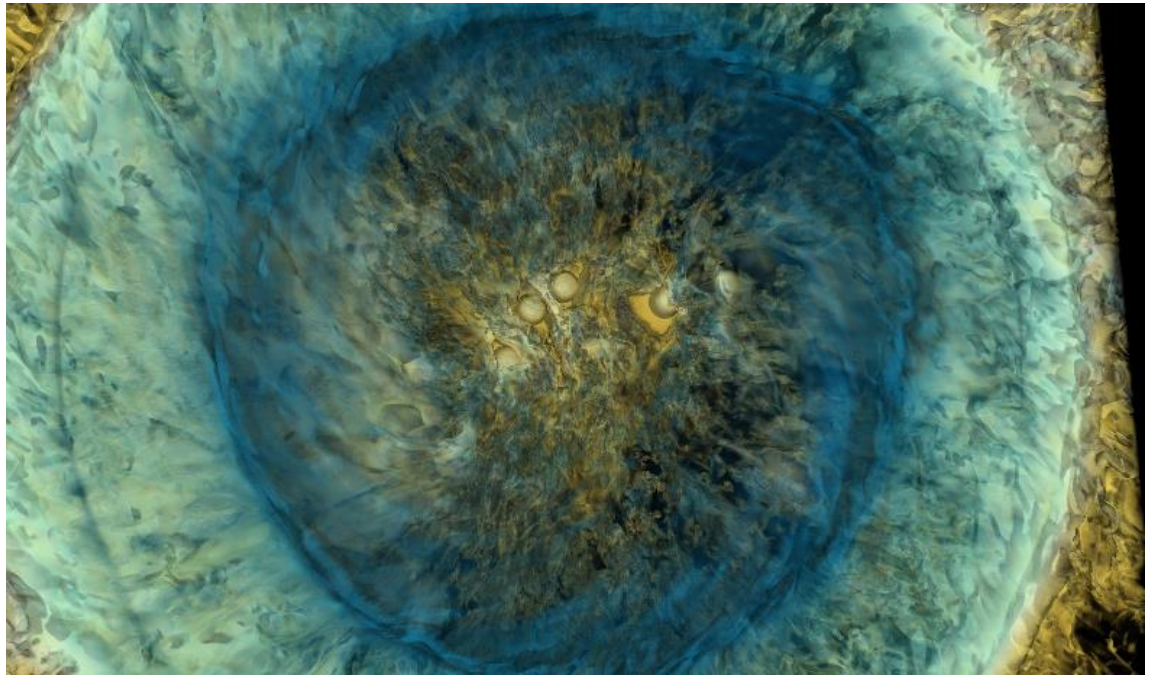




NVIDIA IndeX visual elements: **Edge enhancement**

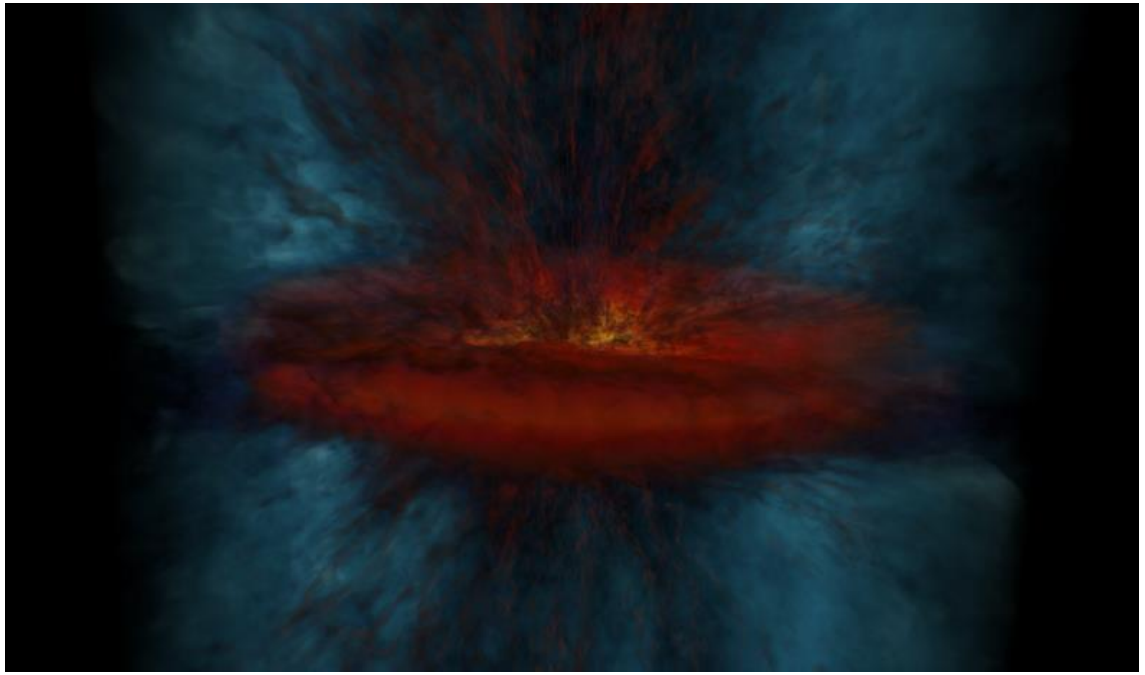
Highlighting the silhouettes

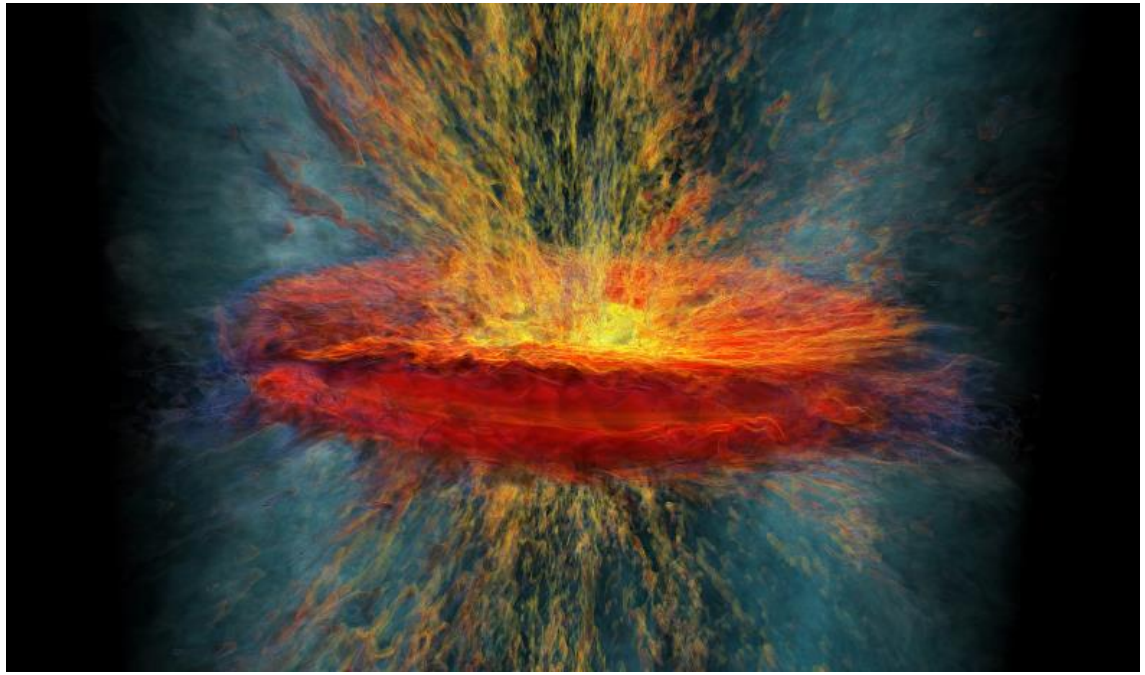




NVIDIA IndeX visual elements: Gradient

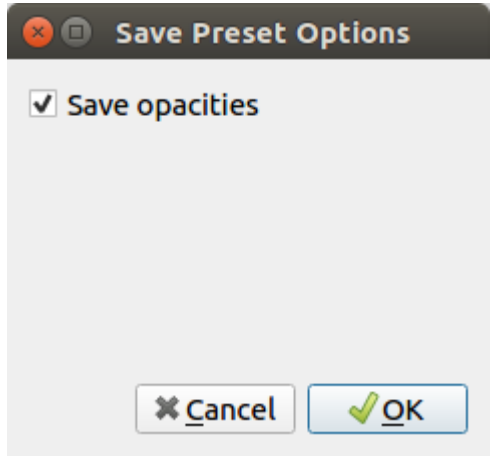
highlight features of great variation in the dataset

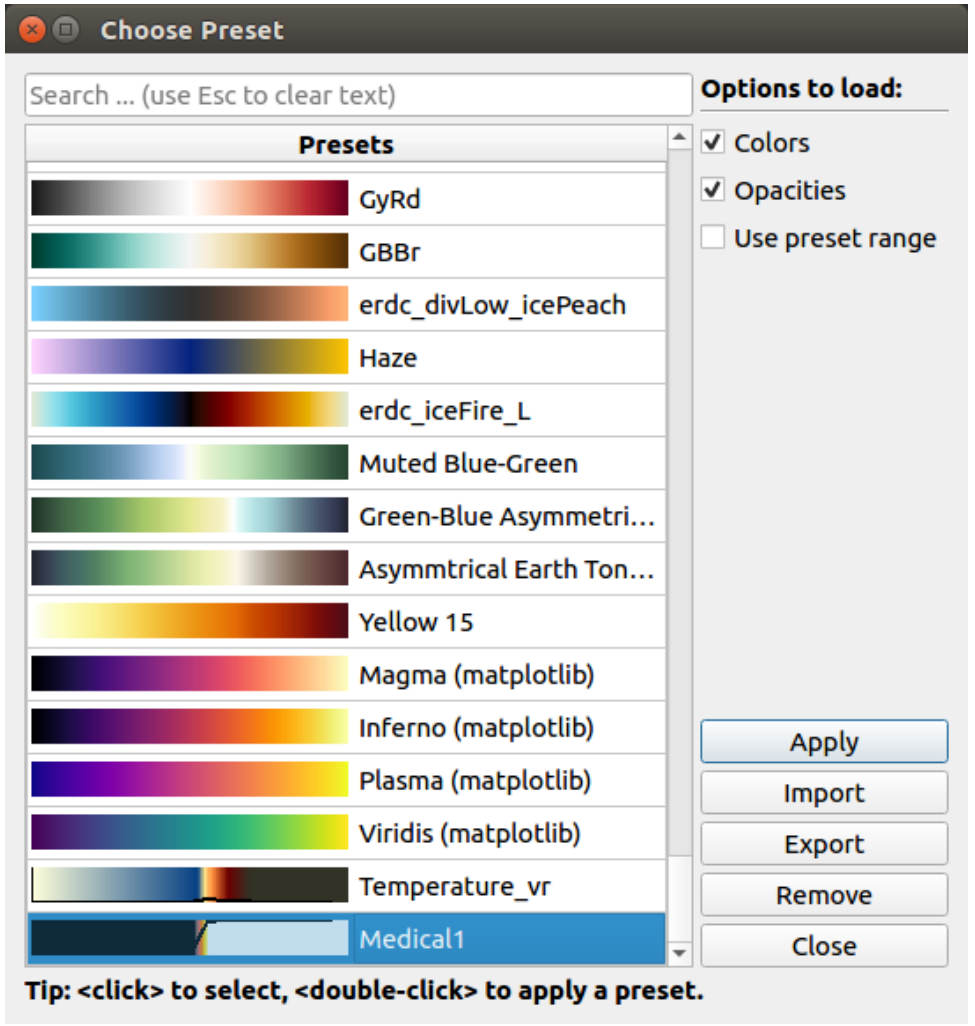




Sharing colormap and opacity transfer function

- So the xfer functions are were all the work resides.
- Sharing them between colleagues is very easy





Export/Import *.json files

GPU rendering of isosurfaces in ParaView

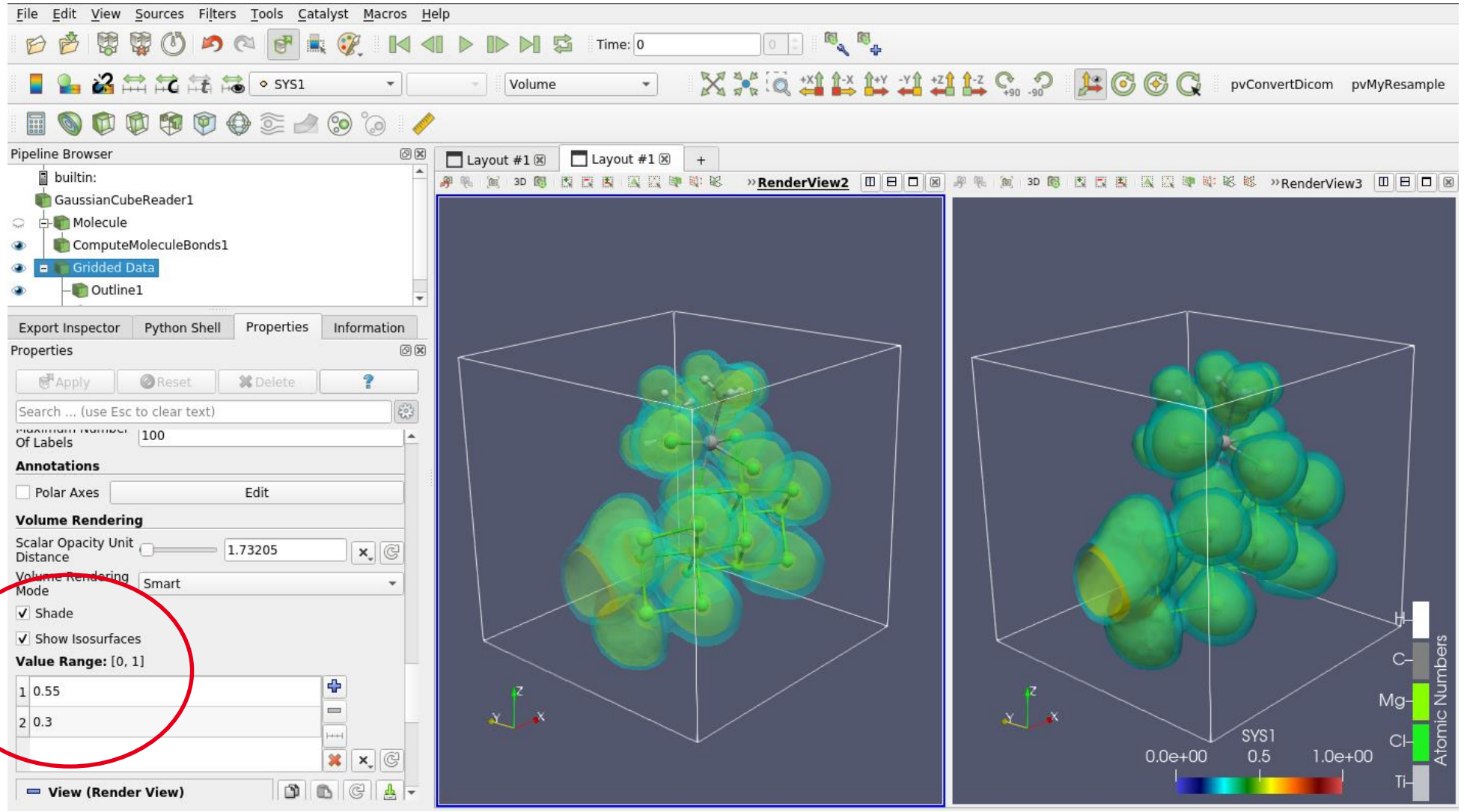
- https://blog.kitware.com/gpu-rendering-of-isosurfaces/

Traditional Visualization:

- extract and visualize some contours of the data (isosurface visualization)
  - a mesh is generated by the CPU for each isosurface and every time the contour value change, a new computation and data transfer to the GPU are necessary.
  - Multiple *semi-transparent* isosurfaces have to be rendered with *Depth Peeling* turned on.

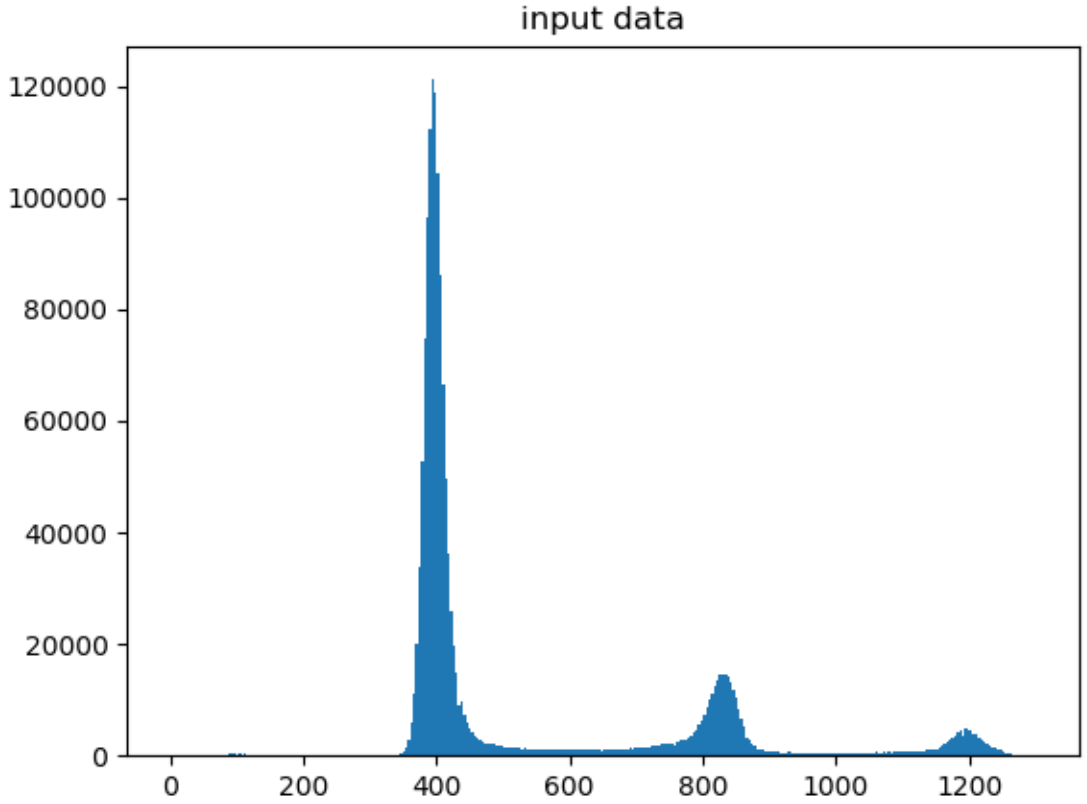
- New Visualization
  - Render several isosurfaces directly on the GPU from the Volume representation/mapper without mesh computation or specific data transfer.

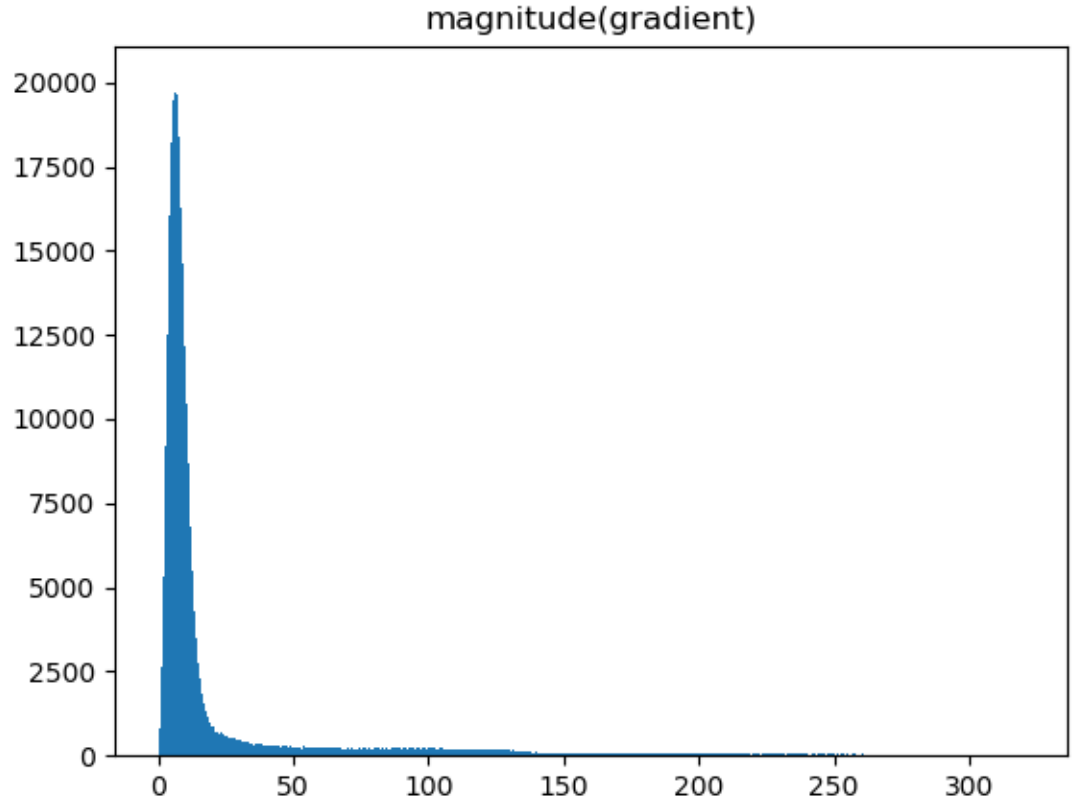
GPU rendering of isosurfaces in ParaView



There is more…

- Not all Volume rendering algorithms are exposed in ParaView
- VTK offers a gradient-based feature to highlight sub-volumes of higher gradient.
- Example: a tooth dataset





The tooth dataset

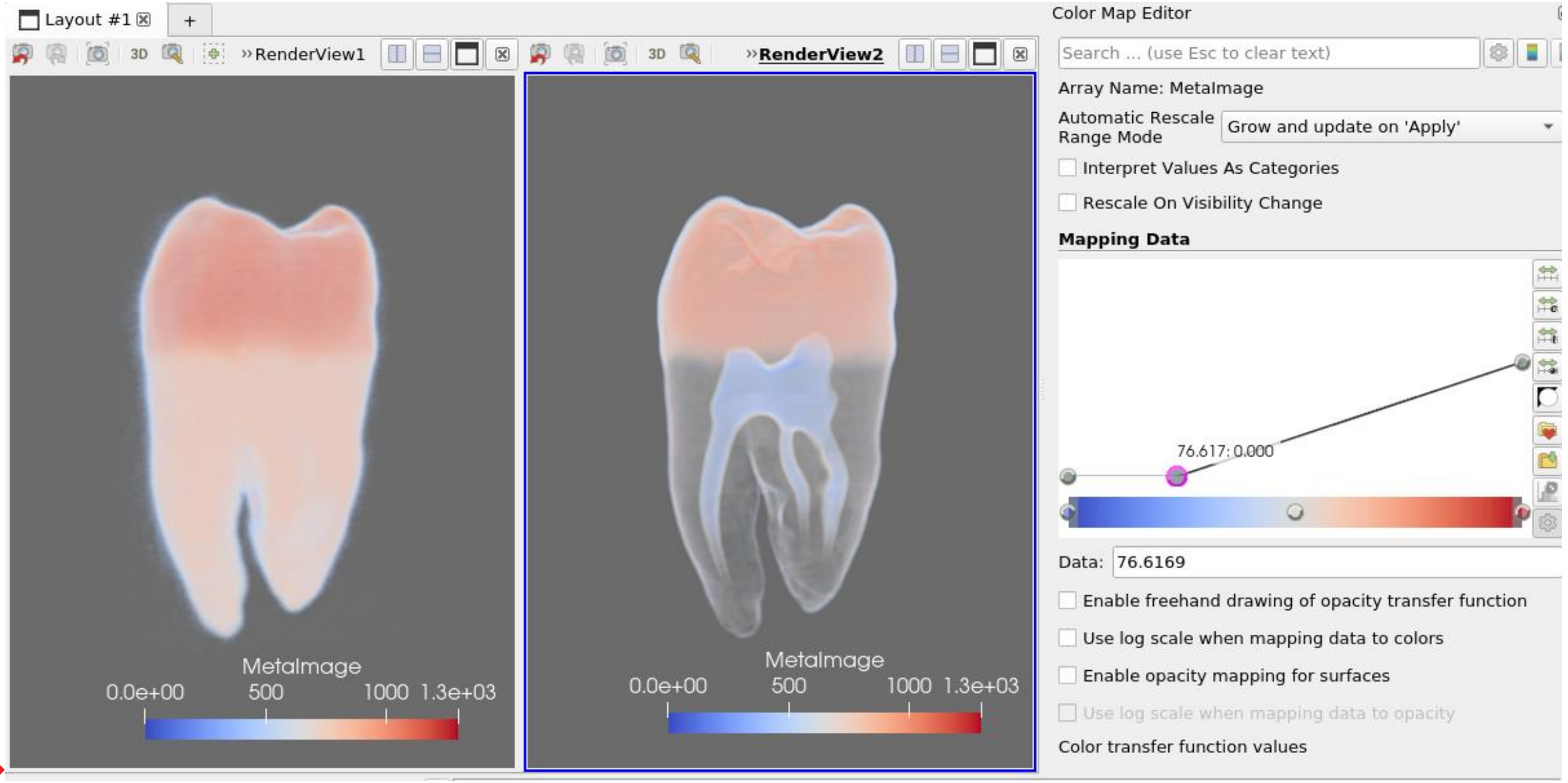
```
grad_fn = vtk.vtkPiecewiseFunction()
grad_fn.AddPoint(0, 0.0)
grad_fn.AddPoint(50, 0.0)
grad_fn.AddPoint(100, 1.0)
grad_fn.AddPoint(max(grad), 1.0)


volumeProperty.SetScalarOpacity(opacity_fn)
volumeProperty.SetGradientOpacity(grad_fn)
volumeProperty.DisableGradientOpacityOff()
volumeProperty.SetColor(scalar_fn)
```

New in v5.9: Volume rendering with a separate opacity array

- The Volume representation using the “Smart” or “GPU Based” modes has a new option for specifying that a separate array should be used for opacity mapping. Additionally, options for choosing the opacity array and component (or vector magnitude) to use for opacity mapping have also been provided.
- When the *UseSeparateOpacityArray* option is enabled, it is now possible to set a custom range for the scalar opacity function that is separate from the range used for the color transfer function.

- *Demo ./ParaView/pvVolumeRenderingSeperateOpacity.py*

Volume rendering with a separate opacity array



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Visualizing large Data Models - Parallel Visualization

Agenda from 9:00am until 10:30am

- SMP Parallelism. The easy way to performance
- Data partitioning
- Parallel Data interfaces for MPI-Parallelism
- Ghost-cells
- Volume rendering in parallel
- Questions

Remote and parallel visualization

- Reference manual

Did you know?

Remote and parallel processing are often used together, but they refer to different concepts, and it is possible to have one without the other.

In the case of ParaView, remote processing refers to the concept of having a client, typically `paraview` or `pvpython`, connecting to a `pvserver`, which could be running on a different, remote machine. All the data processing and, potentially, the rendering can happen on the `pvserver`. The client drives the visualization process by building the visualization pipeline and viewing the generated results.

Parallel processing refers to a concept where instead of single core — which we call a `rank` — processing the entire dataset, we split the dataset among multiple ranks. Typically, an instance of `pvserver` runs in parallel on more than one rank. If a client is connected to a server that runs in parallel, we are using both remote and parallel processing.

In the case of `pvbatch`, we have an application that operates in parallel but without a client connection. This is a case of parallel processing without remote processing.

Parallel visualization is provided in multiple ways

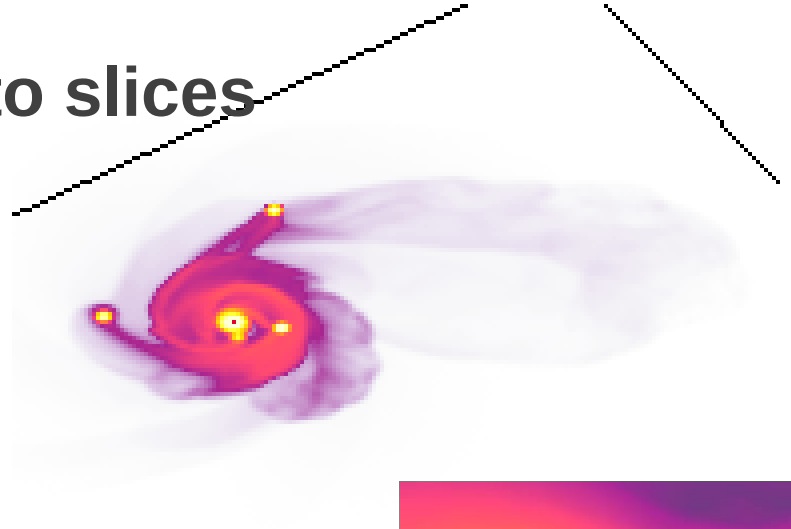
- Client-server (even with a single task on a Piz Daint computer node) is already one way to get more memory and some parallelism
- Client-server and batch mode execution of MPI-based data filtering and rendering engines are a must for big data
  - How big is “big”?
  - Structured grids are lighter
  - Unstructured grids store their topology explicity
  - Beware that often, filters in ParaView will change the topology in some way, and these filters will write out the data as an unstructured grid
  - Must read:
    - Avoid Data Explosion
    - Culling Data
- Still today, many users still use the desktop version of ParaView.
  - So let us start with on-the-node parallelism

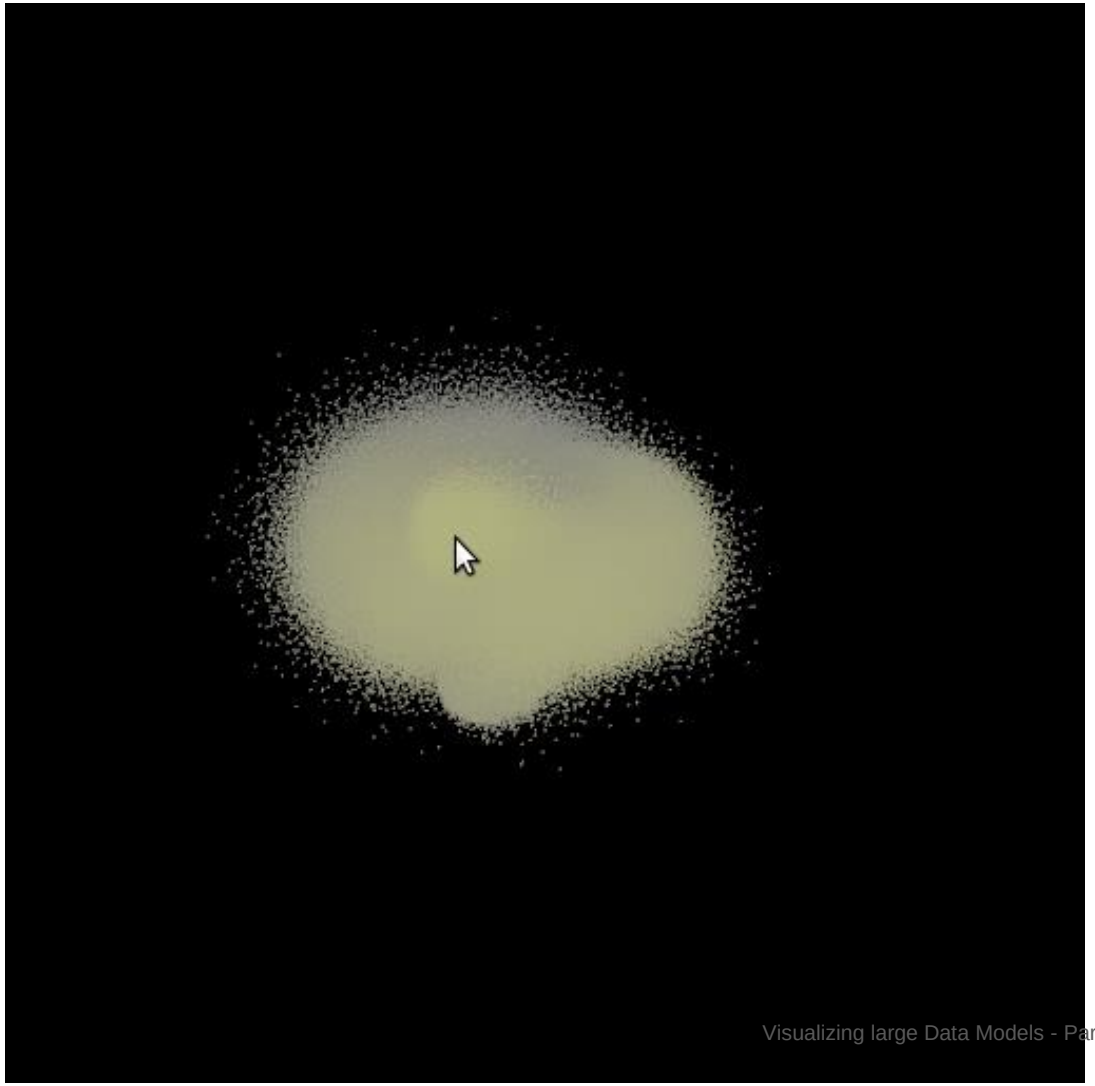
SMP parallelism: examples

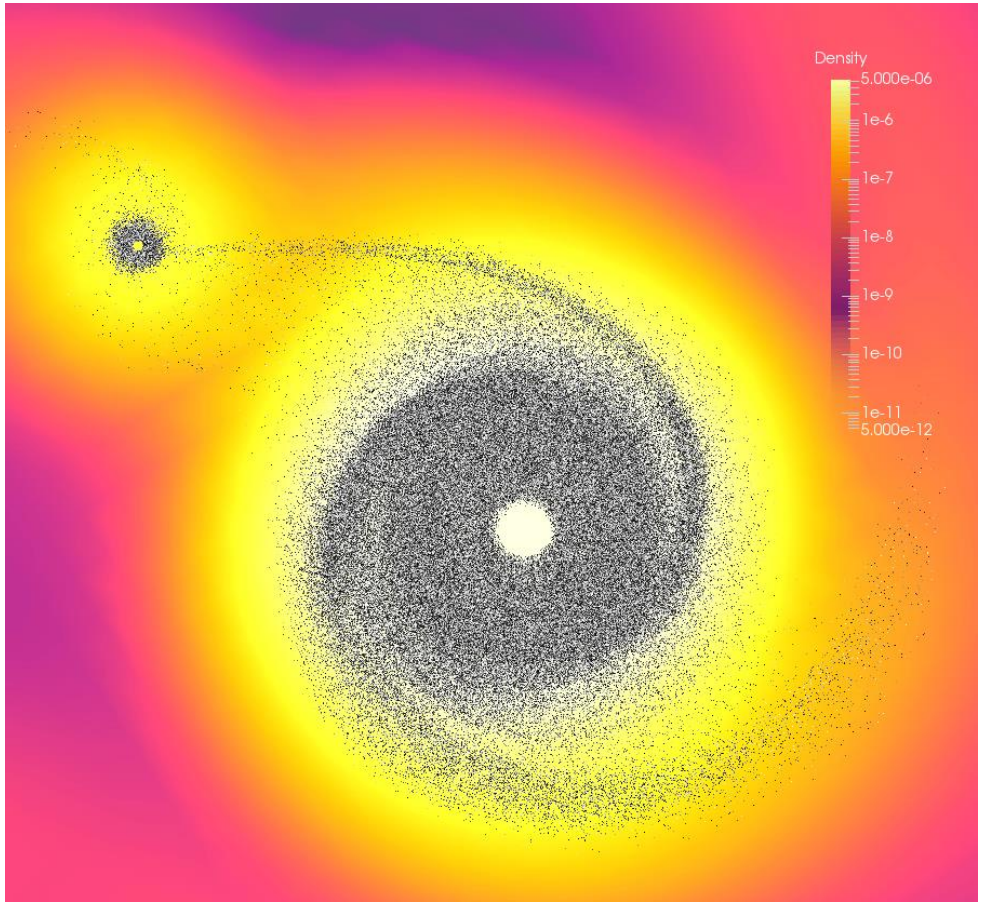
VTKSMPTools

- Shared memory parallel algorithms were kick-started in 2013
- vtkContour3DLinearGrid, vtkDepthImageToPointCloud, vtkShepardMethod, vtkGaussianSplatter, vtkCheckerboardSplatter, vtkImageHistogram, vtkImageDifference, vtkStatisticalOutlierRemoval, vtkPointOccupancyFilter, vtkVoxelGrid, vtkExtractHierarchicalBins, vtkPCACurvatureEstimation, vtkSPHInterpolator, vtkRadiusOutlierRemoval, vtkPointDensityFilter, vtkSignedDistance, vtkExtractPoints, vtkFitImplicitFunction, vtkSignedDistance, vtkPCANormalEstimation, vtkPointInterpolator, vtkPointCloudFilter, vtkHierarchicalBinningFilter, vtkMaskPointsFilter, vtkPointInterpolator2D, vtkExtractSurface, vtkDensifyPointCloudFilter, vtkFlyingEdgesPlaneCutter, vtkSimpleElevationFilter, vtkVectorDot, vtkFlyingEdges3D, vtkPlaneCutter, vtkVectorNorm, vtkFlyingEdges2D, vtkElevationFilter, vtkSampleImplicitFunctionFilter, vtkSortDataArray, vtkSortFieldData, vtkStaticPointLocator, vtkStaticCellLocator, ...
- I *grep*-ed for SMPTools and found it in 119 C++ class for ParaView 5.9
- As of VTK-8, VTK-m is opening to multiple back-ends and different implementations (CUDA,)

Example 1: SPH Particle clouds to slices





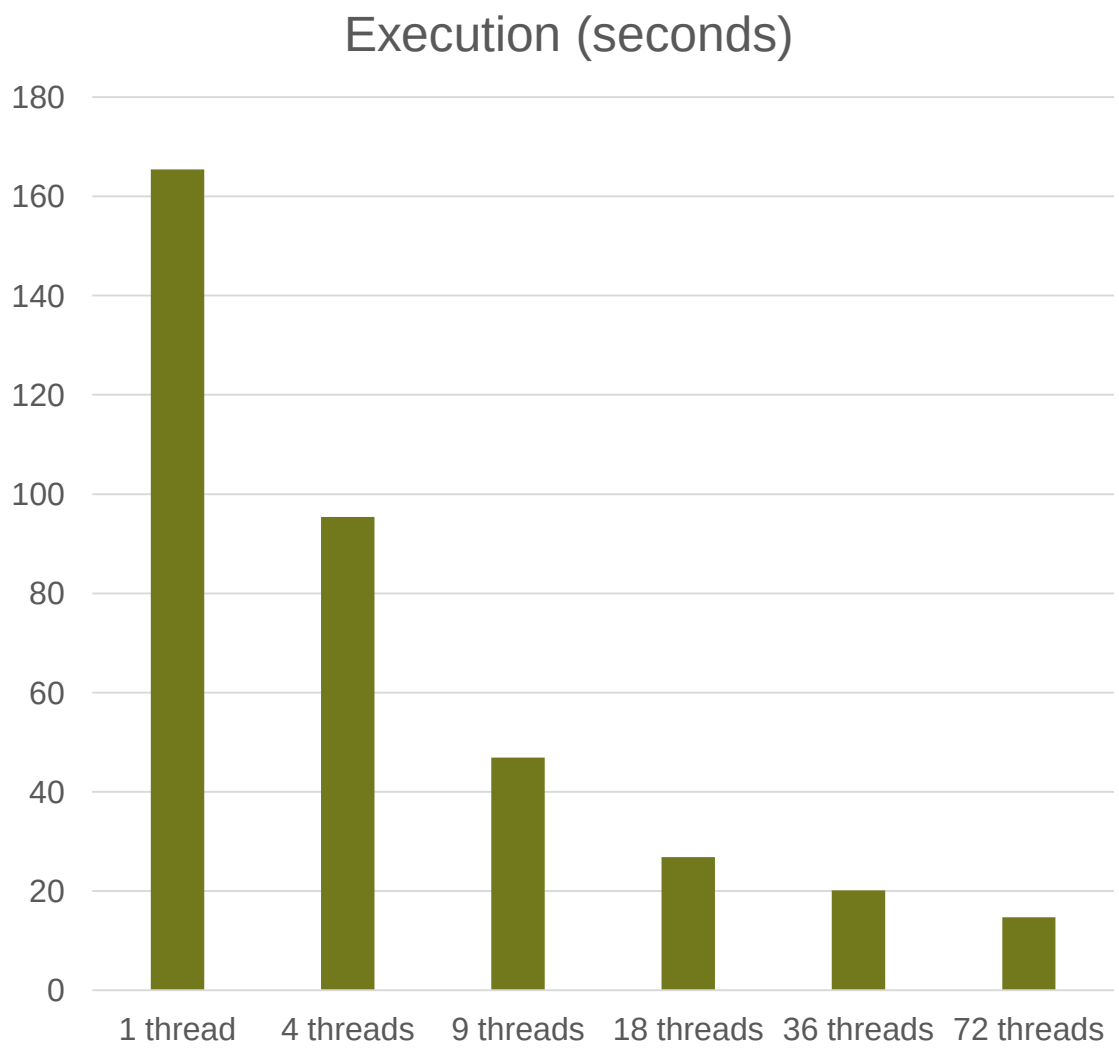


Interactive demo on https://jupyter.cscs.ch

/users/jfavre/Projects/Visualization-training/ParaView/ParaView-SMP-Parallel.ipynb

SPH data interpolation

- The vtkSPHInterpolator filter uses SPH (smooth particle hydrodynamics) kernels to interpolate a data source onto an input structure.
- A Point locator is a crucial part of the execution path, to accelerate queries about points and their neighbors.
- SMP-based acceleration provides a fast plane interpolation using parallelism-on-the-node with Intel TBB.
- Piz Daint Compute node: dual socket Intel® Xeon® E5-2695 v4 @ 2.10GHz (18 cores)

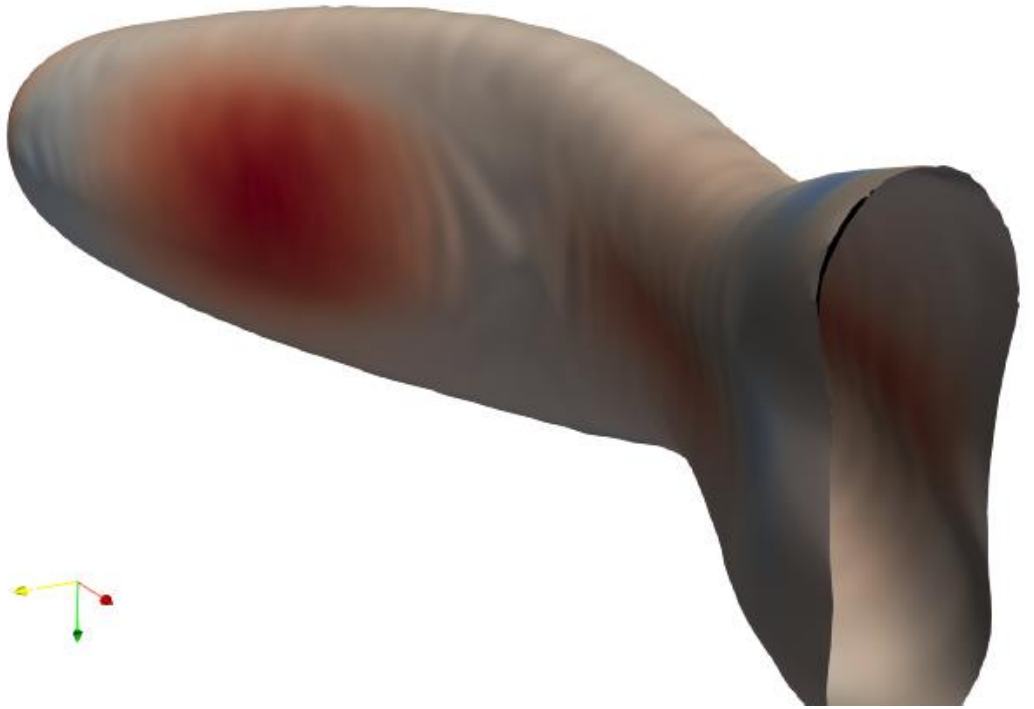


Example 2: Reconstruct a surface from a cloud of points

- Use vtkPCANormalEstimation to add Normals from a cloud of points

- Use vtkSignedDistance filter to generate a volumetric scalar field

- Use vtkExtractSurface to generate the zero-crossing isosurface from the truncated signed distance

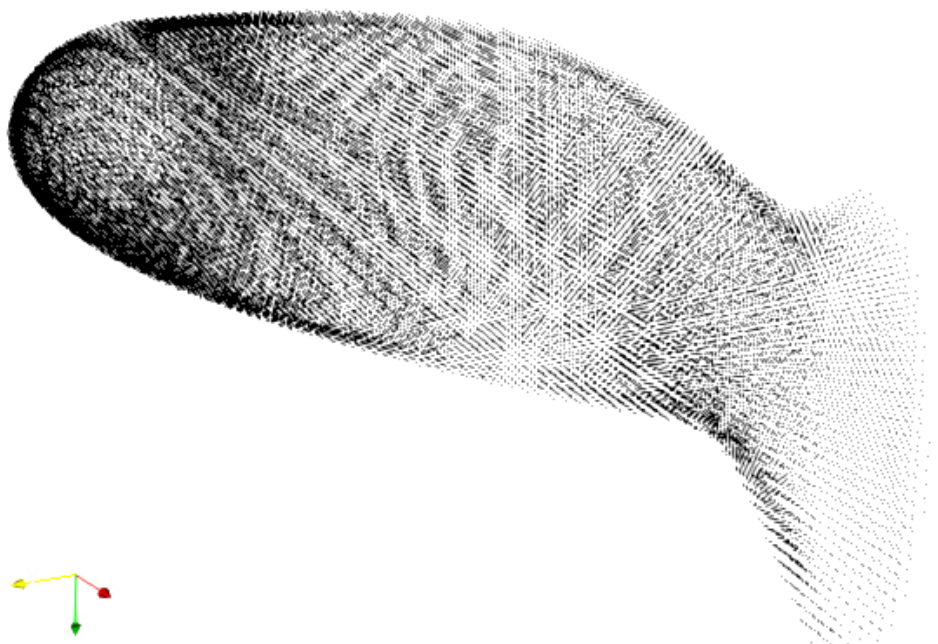


Reconstruct a surface from a cloud of points

- Use vtkPCANormalEstimation to add Normals from a cloud of points

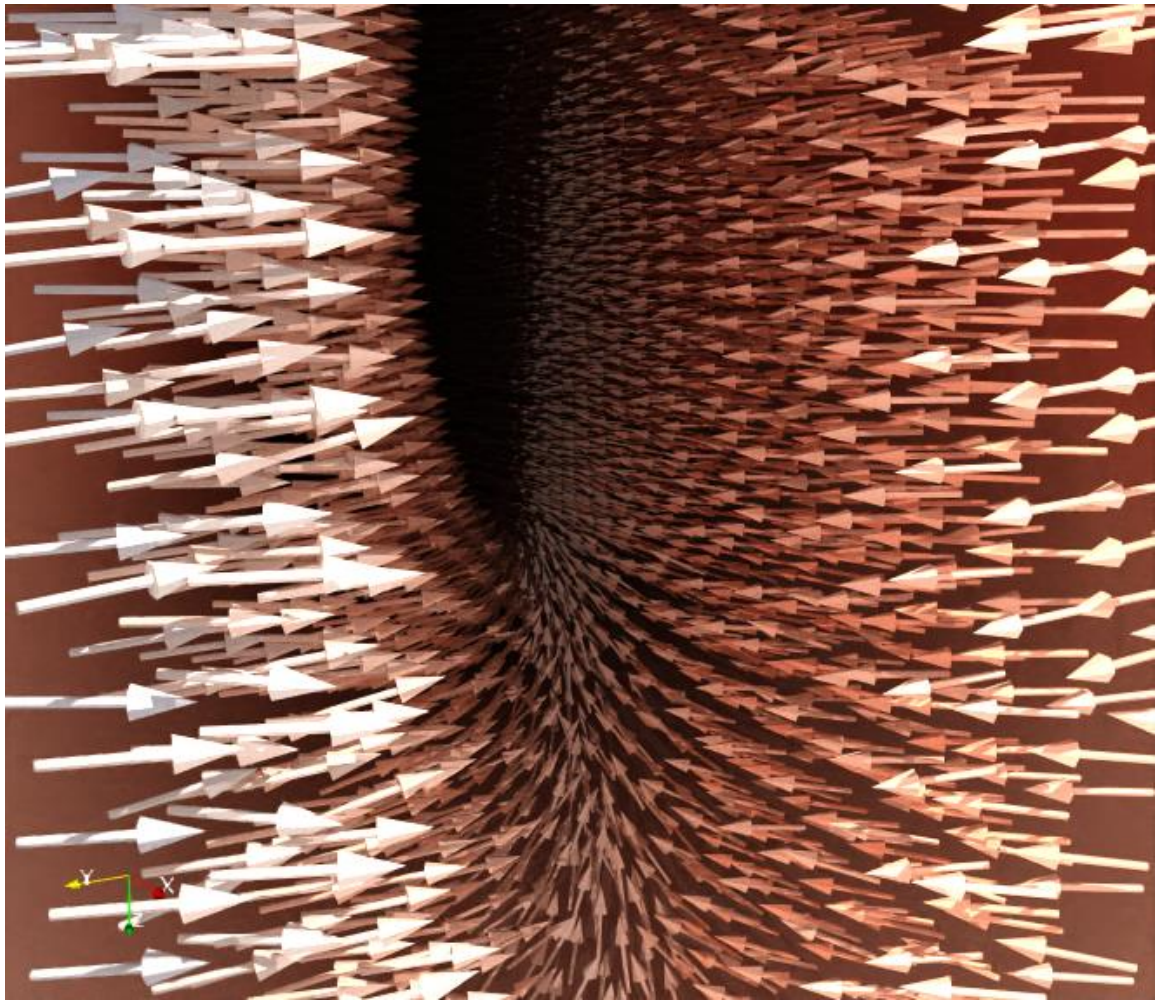
- Use vtkSignedDistance filter to generate a volumetric scalar field

- Use vtkExtractSurface to generate the zero-crossing isosurface from the truncated signed distance



Reconstruct a surface from a cloud of points

- Use vtkPCANormalEstimation to add Normals from a cloud of points
- Use vtkSignedDistance filter to generate a volumetric scalar field
- Use vtkExtractSurface to generate the zero-crossing isosurface from the truncated signed distance



Example 3: Slicing through a 1024^3 volume

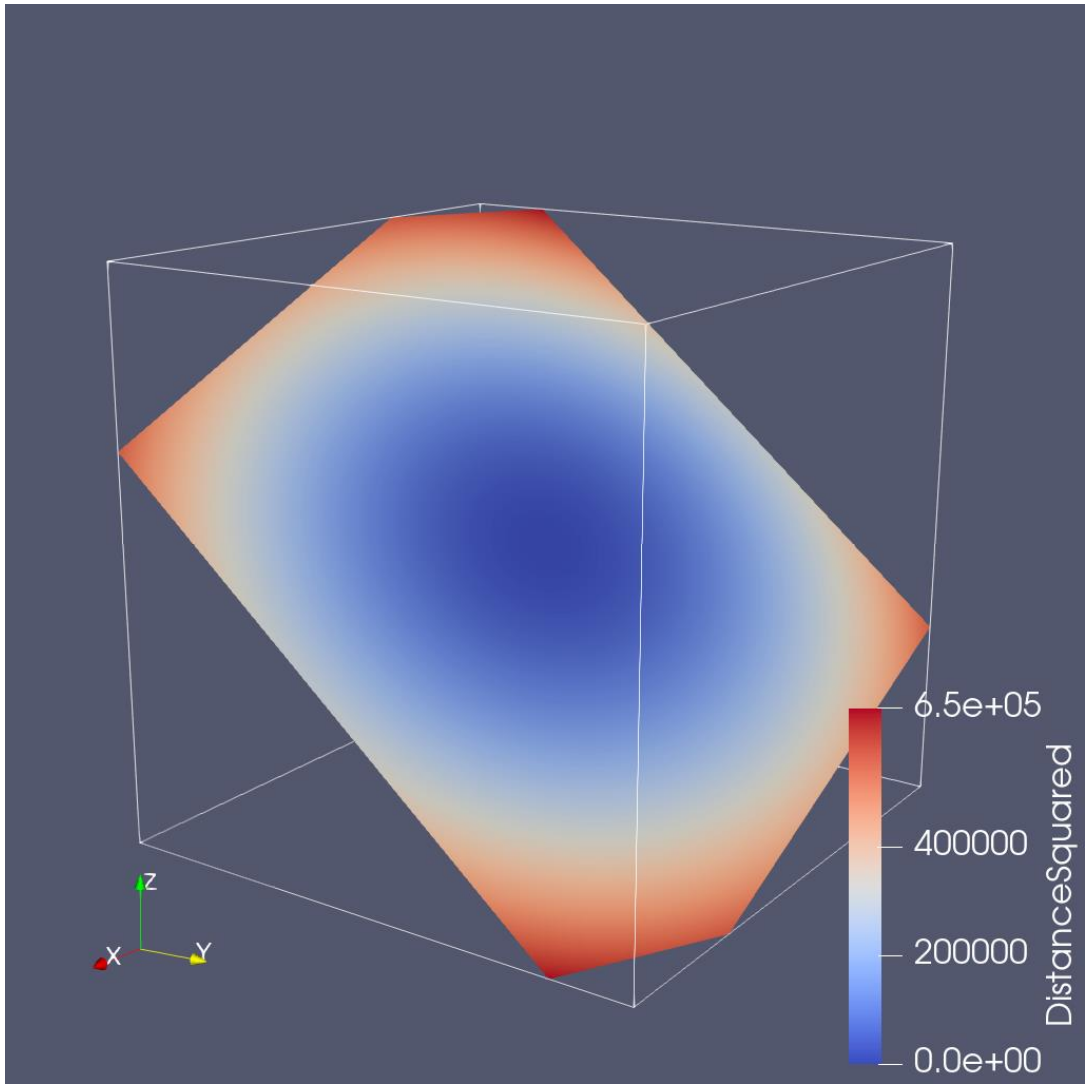
Use the default slicer tool

Execution time: ~ 25 seconds

Use the FlyingEdgesPlaneCutter

(from the *Accelerated Algorithms* plugin)

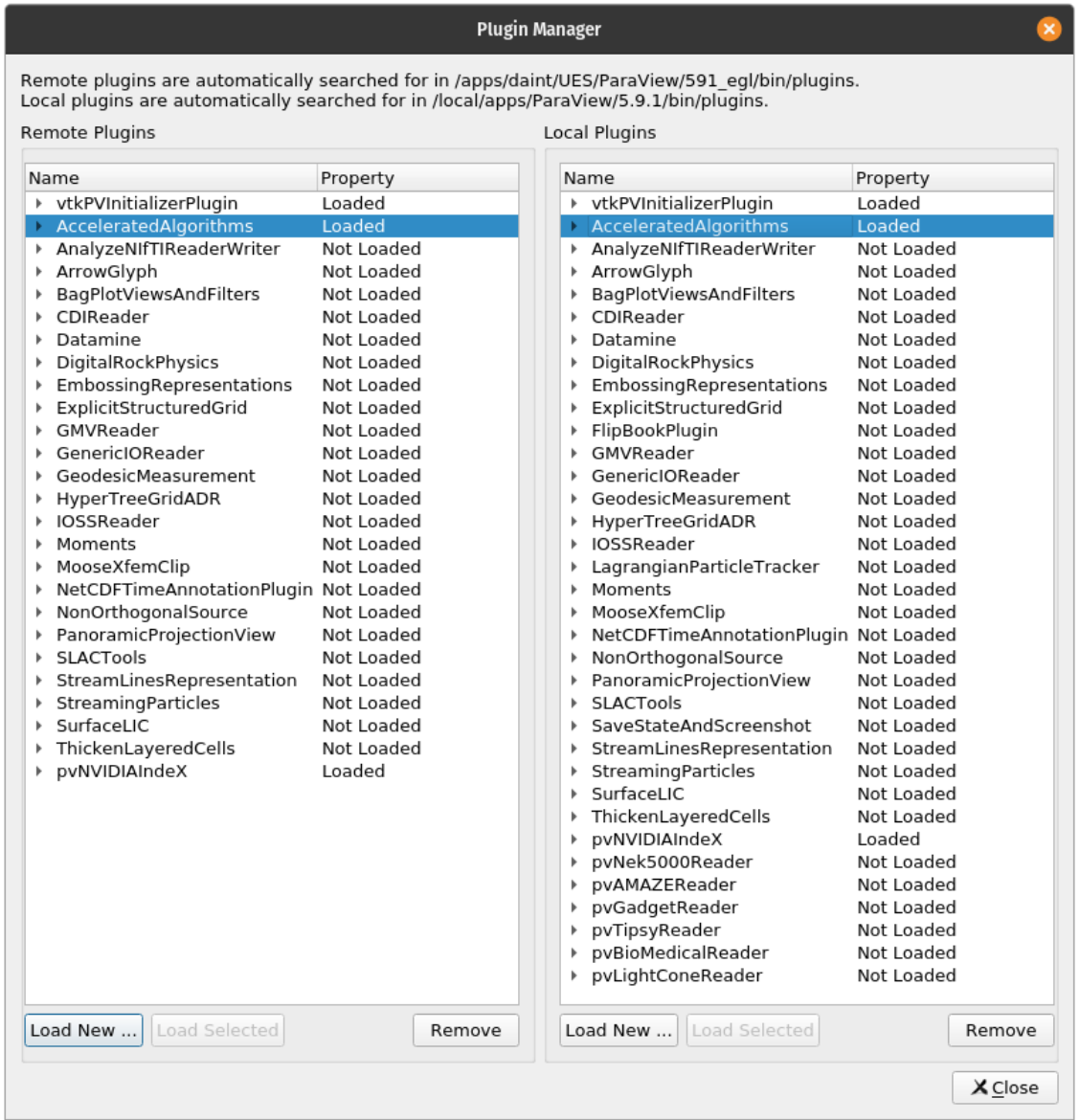
Execution time: ~ 0.23 seconds



SMP parallelism. Choose your filters carefully…

/users/jfavre/Projects/Visualization-training/ParaView/LargeData-Contours.ipynb

And offload the computation to a single node of Piz Daint



MPI-parallelism

Data partitioning

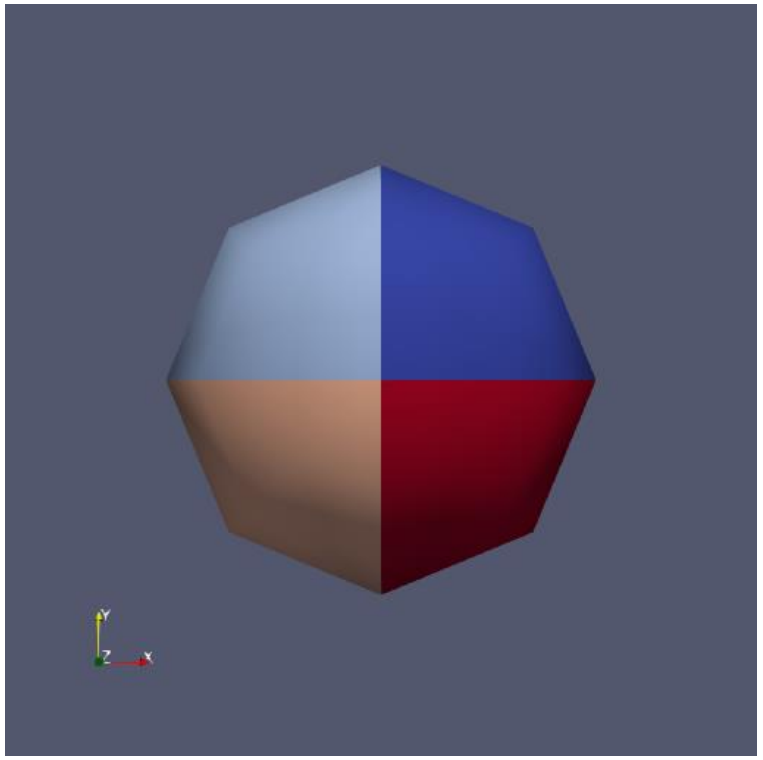
- So you wanna do “real” parallelism, and split your data among multiple nodes.

!!! Great idea !!!

- Now, to the details about splitting data...

“Hello Sphere” in parallel

```
from paraview.simple import *
Sphere() # a synthetic data generator
pid = ProcessIdScalars()
rep = Show()
# to scale for any number of processors.
rep.RescaleTransferFunctionToDataRange(False, True)
# to save an image
SaveScreenshot("sphere.png")
```



Can be executed *locally* with “mpiexec –n 4 pvbatch pvSphere.py”

Interactive demo on https://jupyter.cscs.ch

/users/jfavre/Projects/Visualization-training/ParaView/Hello_Sphere-ParaView.2.ipynb

The Sphere Source example is parallel-aware

- What about your data-reader?

- Please. Plan ahead of time. Test. Ask for help

parallel-aware data readers. Survey Time!

Not all data readers are capable of reading data in parallel, and splitting (load balancing) their data amongst multiple servers

How many of you think that ParaView can read, in parallel:

- An OpenFoam dataset?
- A regular cartesian grid for Volume Rendering? (*.mhd)
- A regular cartesian grid using VTK XML vtkImageData format? (*.vti)
- A grid stored using VTK XML vtkUnstructuredGrid format? (*.vtu)

Details about the MetaIO data reader

- https://www.paraview.org/Wiki/ITK/MetaIO

ObjectType = Image

NDims = 3

BinaryData = True

BinaryDataByteOrderMSB = False

CompressedData = False

ElementSpacing = 0.5625 0.5625 1

DimSize = 2048 1024 512

ElementType = MET_FLOAT

ElementDataFile = volume.raw

Live Demonstration

A quick fix

```
from paraview.simple import *
f = MetaFileSeriesReader(FileNames=["volume.mhd"])
f.UpdatePipeline()
SaveData("volume.vti")
```

Live Demonstration

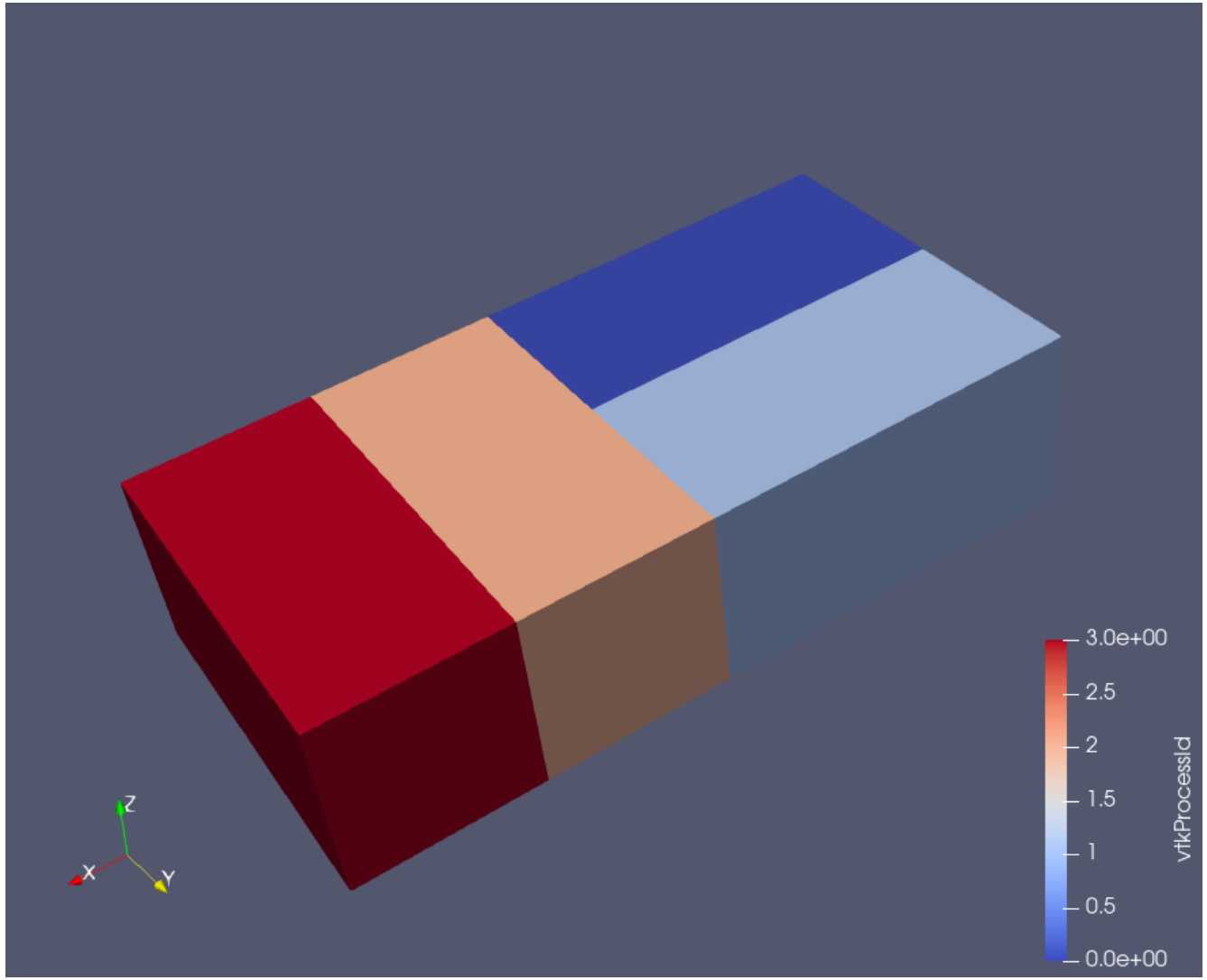
parallel-aware data readers

- An OpenFoam dataset in parallel? NO
- A regular cartesian grid for Vol Rendering? (*.mhd) NO
- A regular cartesian grid using VTK XML vtkImageData format? (*.vti) YES
- A grid stored using VTK XML vtkUnstructuredGrid format? (*.vtu) NO

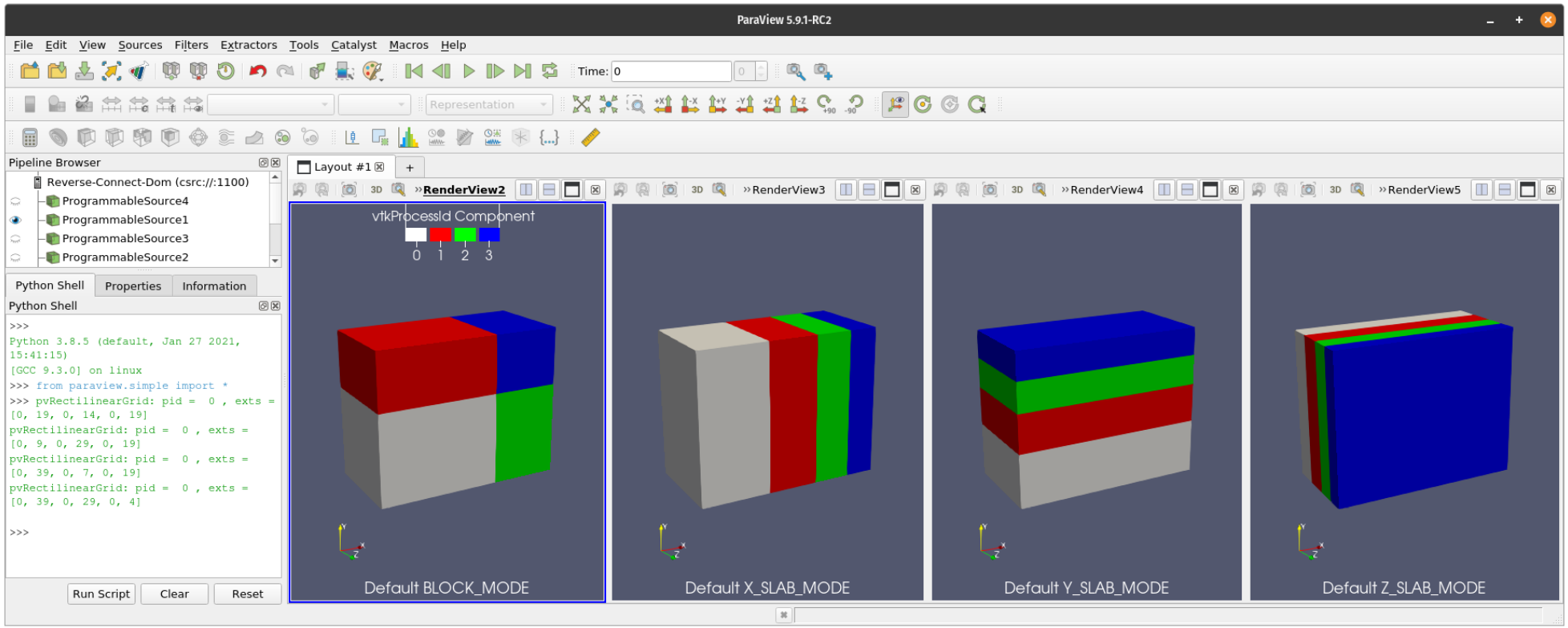
Data partitioning. Use a data format that supports it

By the way, how did ParaView do that?

What a strange way to split the data!



Data partitioning



There are 4 ways to split structured data (by block, or along any of the I,J, K direction)

But ParaView only gives you one, by default

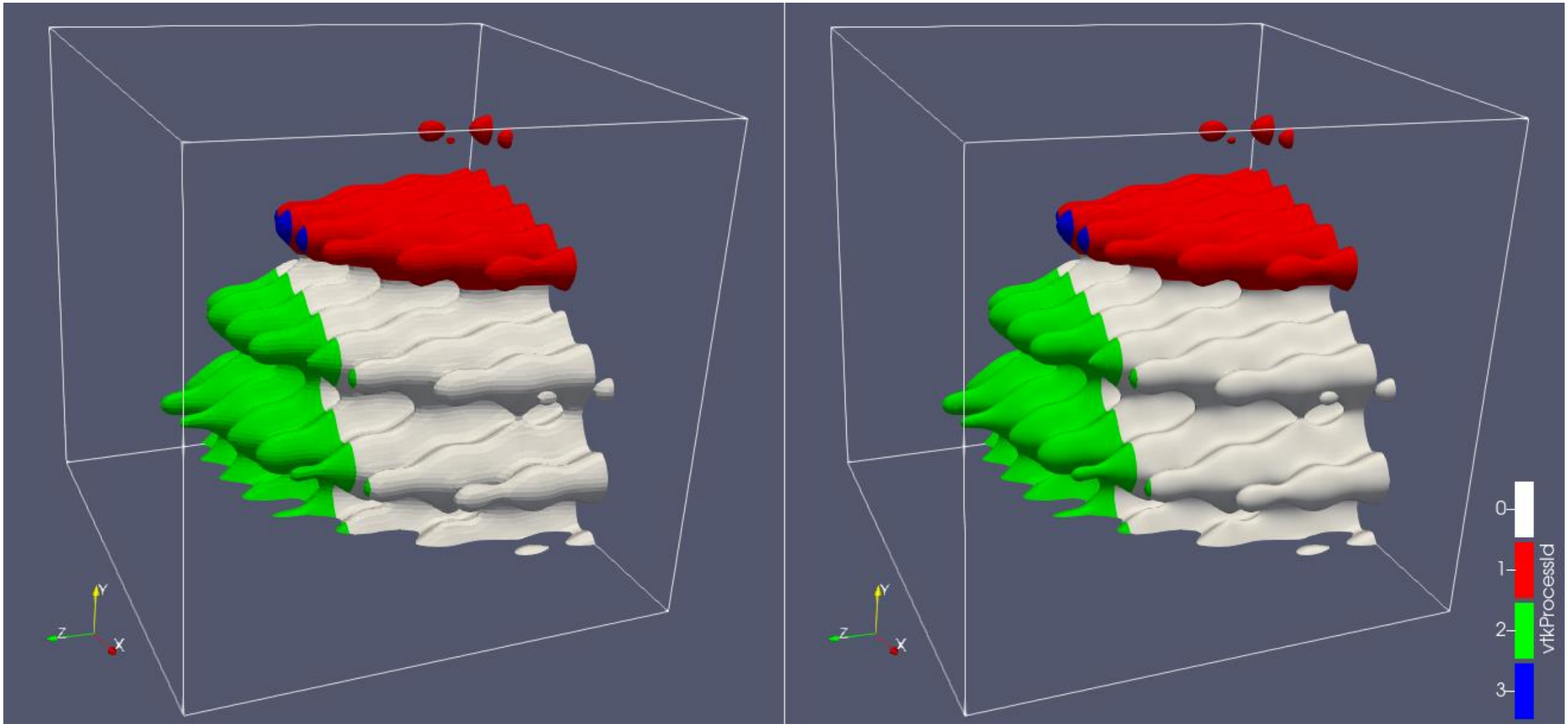
There are also other ways to split data…

Think about grid-less particles, AMR or multi-block data

MPI-parallelism

- Transparent to the user.
- Python scripts or interactive visualization should work without changes
- Beware of your I/O and refer to the discussion about Data Extents and ghost-cells

Ghost-cells?

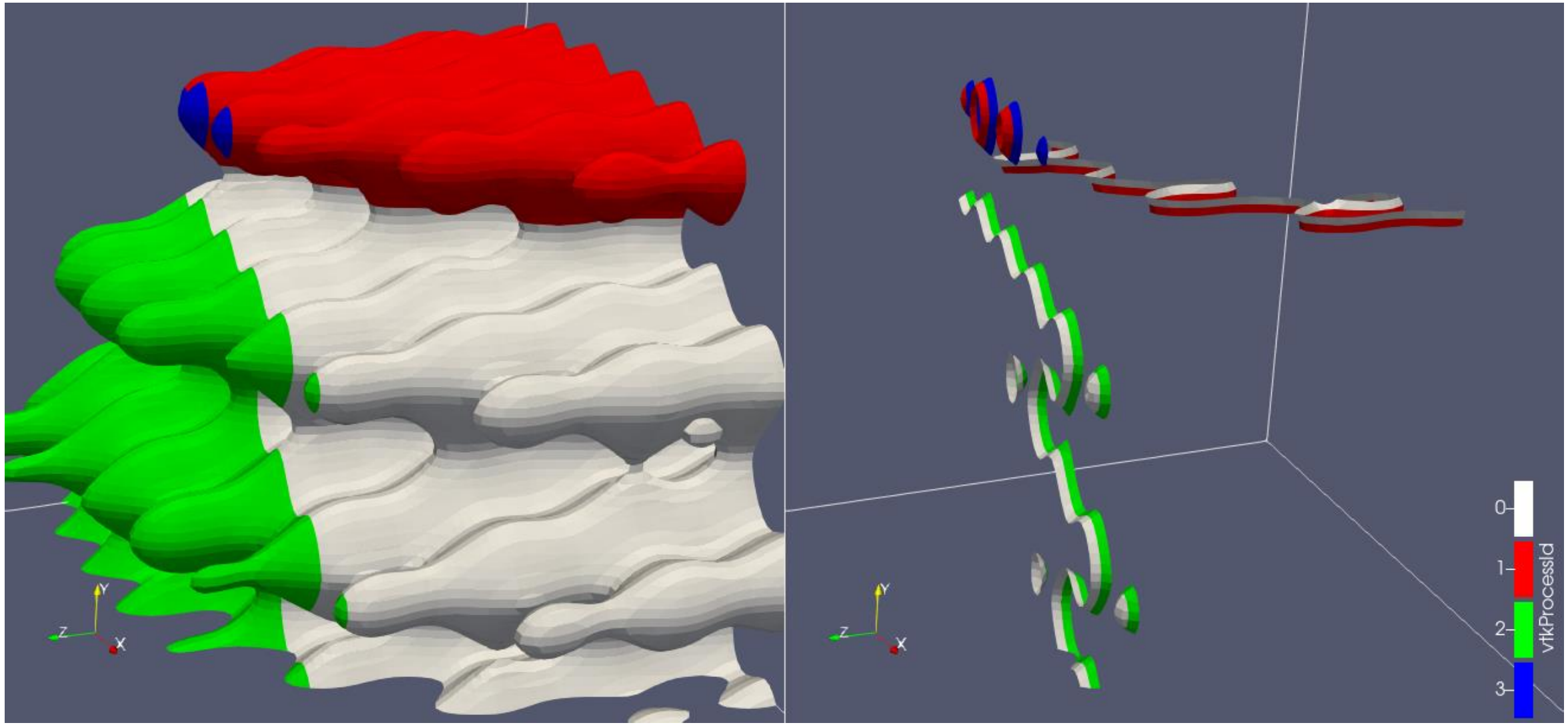


When the computation of ghost-cells is triggered by a filter

- To do a better surface shading, we must use Normals
- To compute Normals, we average normal vector of all vertices of a polygonal primitive
- What about when we are on the edge…?
- Typical of finite difference computational stencils. We ask neighbors for ghost cells.

- Implication => we most likely will re-read the data from disk with updated extents (the split indices)

Ghost-cells?



Ghost Cells Generator

If your unstructured grid data is already partitioned satisfactorily but does not have ghost cells, it is possible to generate them using the Ghost Cells Generator filter.

MPI-parallelism. Summary

- Transparent to the user.
- Python scripts or interactive visualization should work without changes (*well, almost no changes*)
- Beware of your I/O and refer to the discussion about Data Extents and ghost-cells