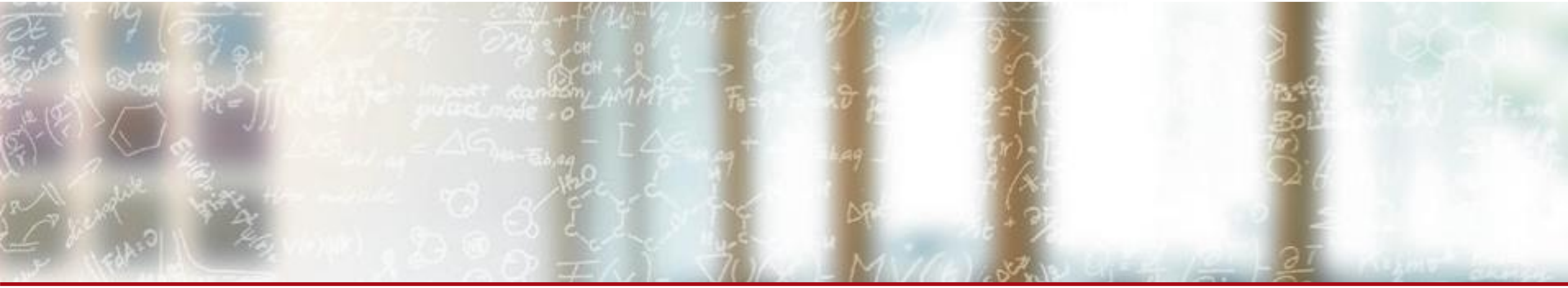
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETHzürich



Scientific visualization

PDC Summer School 2023

Jean M. Favre, CSCS

August 24, 2023

Preamble 1

- We will use two different methods of using/demonstrating visualization in an HPC context
  - On your desktop with small data
  - In Dardel, with small and big data
- Paraview can be driven both by a Python script, and by interactive GUI-based actions (understand the diffs between *paraview, pvpython, pvserver, pvbatch*)
- Thus, you should clone the repo on both sides to see the exercise/demo file, e.g

```
git clone https://github.com/jfavre/PDC-SummerSchool    Visualization
```

- In my lecture notes, all examples for exercises should be written in green, i.e. pvSphere.py. You should find them in the git repo.

Preamble 2

- No training dedicated to HPC I/O in this summer school
- Difficult to know what is our common understanding of file I/O and file formats

- Most demonstration/exercises will use auto-generated data

Outline of the day

- Scientific data analysis and visualization (1hr)
  - ParaView hands-on lab session
- Parallel data visualization (1hr)
  - ParaView hands-on lab session for parallel data visualisation

- Because of the limited time, I decided to focus on a single application (ParaView), to avoid context switching
- ParaView is open-source, available everywhere on earth, and a *de-facto* leader in HPC visualization apps.
- The comprehensive ParaView guide is here:

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

Definition, Data dimensions

Information Visualization is the study of visual representations of abstract data to reinforce human cognition. The abstract data include both numerical and non-numerical data, such as text and geographic information. However, information visualization differs from *Scientific Visualization*.

"It is *Information Visualization*, when the spatial representation is chosen,
and it is *Scientific Visualization*, when the spatial representation is given“

The X,Y,Z,Time space will be our natural basis, and we will deal with grid-less or gridded data.

Grid types

Grid Types. The VTK Data Model

https://docs.paraview.org/en/latest/UsersGuide/understandingData.html#

Take ParaView’s ProgrammableSource and list its possible output types:

pvpython

```
>>> from paraview.simple import *
>>> a = ProgrammableSource()
>>> a.OutputDataSetType.Available
['vtkPolyData', 'vtkStructuredGrid', 'vtkRectilinearGrid', 'vtkUnstructuredGrid',
'vtkImageData', 'vtkMultiblockDataSet', 'vtkHierarchicalBoxDataSet', 'vtkTable',
'vtkMolecule', 'vtkPartitionedDataSet', 'vtkPartitionedDataSetCollection']
```

Grid Types. The VTK Data Model

It is far more important to know the mesh type a reader will produce….

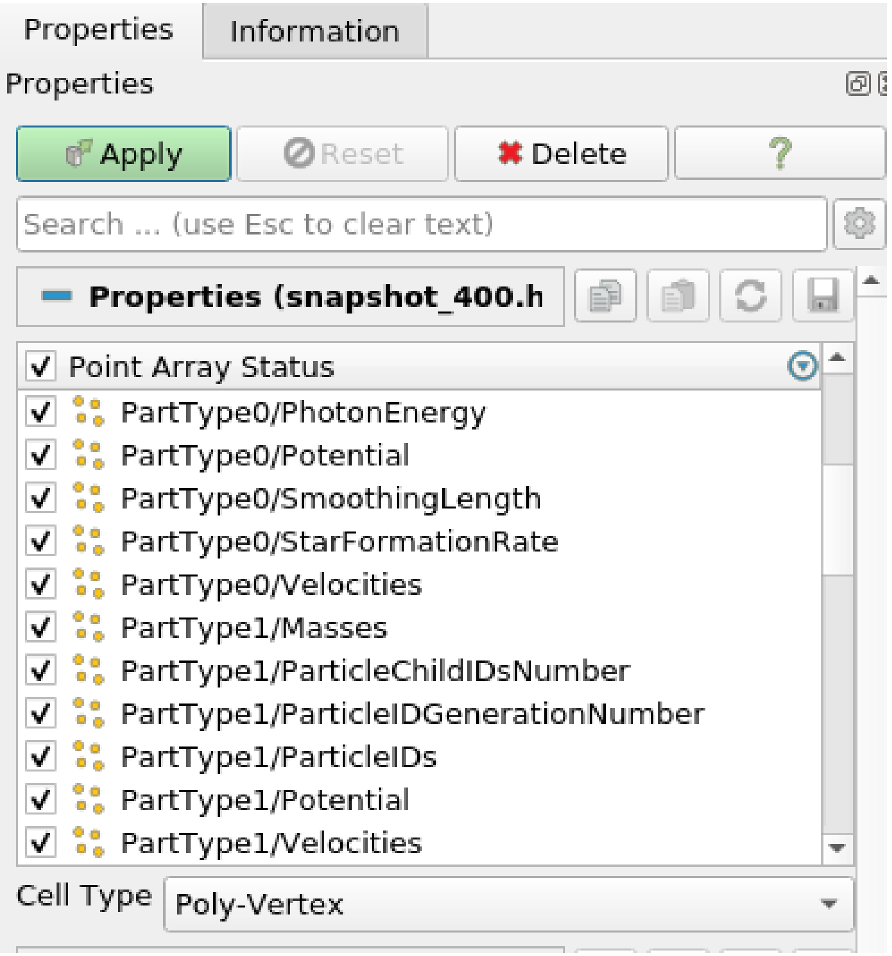
['vtkPolyData', 'vtkStructuredGrid', 'vtkRectilinearGrid', 'vtkUnstructuredGrid', 'vtkImageData', 'vtkMultiblockDataSet', 'vtkHierarchicalBoxDataSet', 'vtkTable', 'vtkMolecule']

…rather than its specific name

Said in a different manner “*you can encode the same data in multiple data formats*”

Implementation details will be important for time-series and parallel support

ParaView’s look at data



Properties

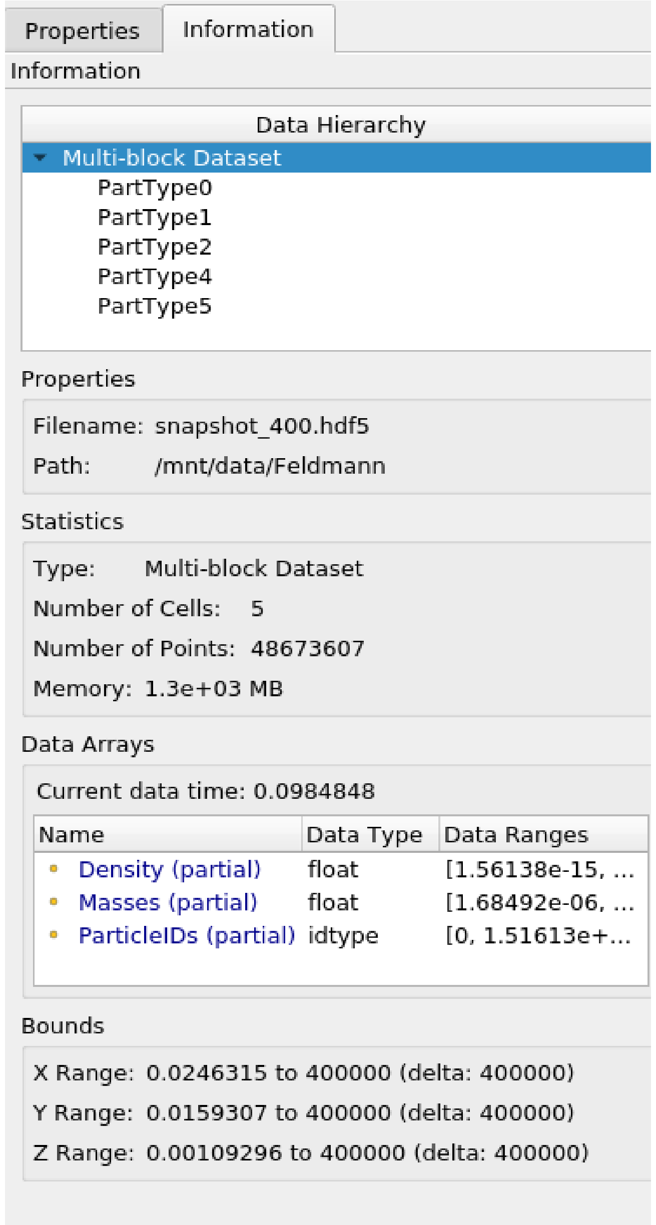
Search ... (use Esc to clear text)

Properties (snapshot_400.h

☑ Point Array Status

- ☑ PartType0/PhotonEnergy
- ☑ PartType0/Potential
- ☑ PartType0/SmoothingLength
- ☑ PartType0/StarFormationRate
- ☑ PartType0/Velocities
- ☑ PartType1/Masses
- ☑ PartType1/ParticleChildIDsNumber
- ☑ PartType1/ParticleIDGenerationNumber
- ☑ PartType1/ParticleIDs
- ☑ PartType1/Potential
- ☑ PartType1/Velocities

Cell Type: Poly-Vertex



Information

Data Hierarchy

- Multi-block Dataset
  - PartType0
  - PartType1
  - PartType2
  - PartType4
  - PartType5

Properties

Filename: snapshot_400.hdf5
Path: /mnt/data/Feldmann

Statistics

Type: Multi-block Dataset
Number of Cells: 5
Number of Points: 48673607
Memory: 1.3e+03 MB

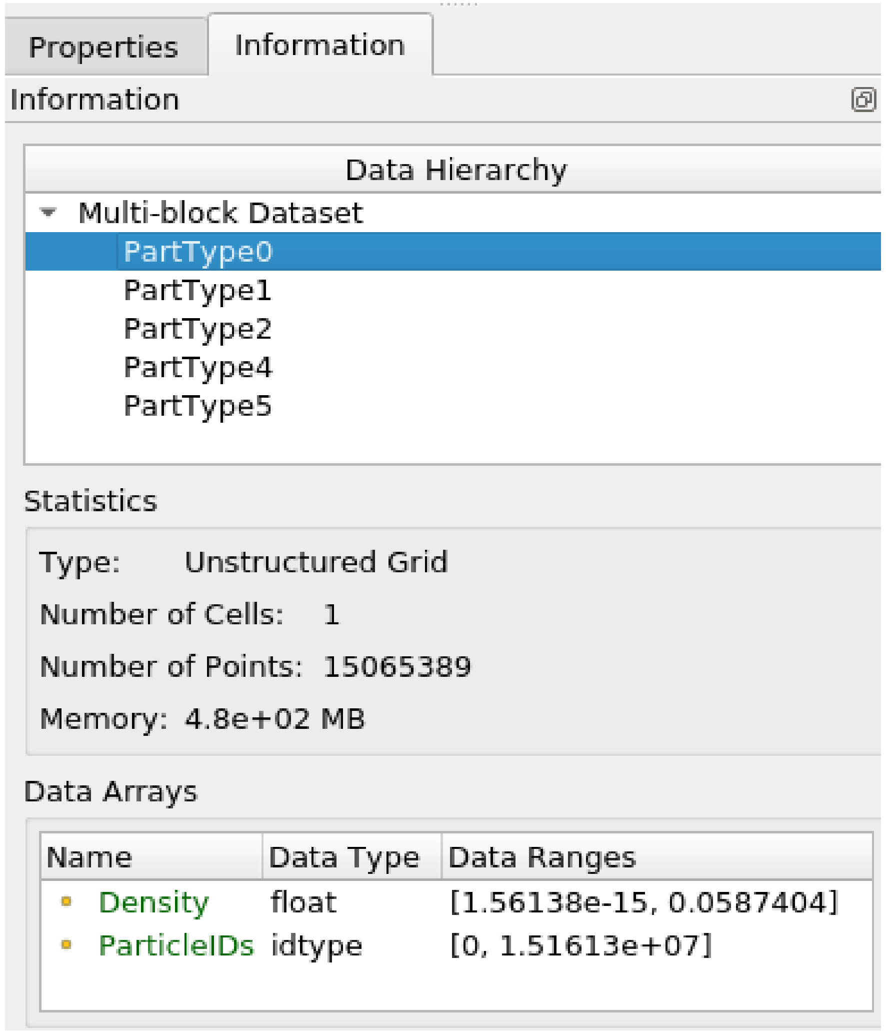
Data Arrays

Current data time: 0.0984848

Name	Data Type	Data Ranges
Density (partial)	float	[1.56138e-15, ...
Masses (partial)	float	[1.68492e-06, ...
ParticleIDs (partial)	idtype	[0, 1.51613e+...

Bounds

X Range: 0.0246315 to 400000 (delta: 400000)
Y Range: 0.0159307 to 400000 (delta: 400000)
Z Range: 0.00109296 to 400000 (delta: 400000)



Information

Data Hierarchy

- Multi-block Dataset
  - PartType0
  - PartType1
  - PartType2
  - PartType4
  - PartType5

Statistics

Type: Unstructured Grid
Number of Cells: 1
Number of Points: 15065389
Memory: 4.8e+02 MB

Data Arrays

Name	Data Type	Data Ranges
Density	float	[1.56138e-15, 0.0587404]
ParticleIDs	idtype	[0, 1.51613e+07]

Grid Types…and filtering

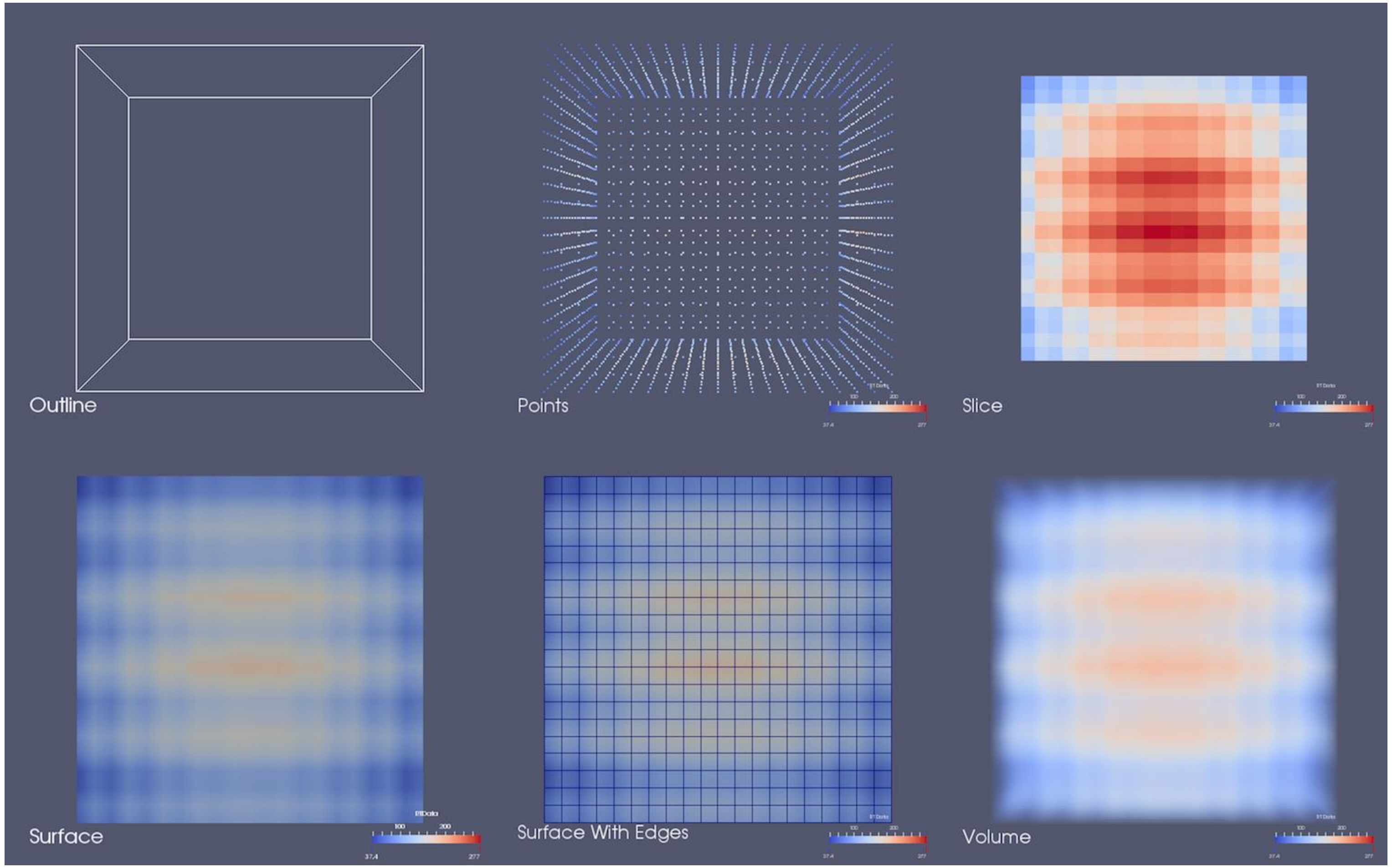
Each mesh type will have a corresponding subset of filters that can be applied to it.

Example:

Mesh Type	Can Use ExtractSubset ?
vtkPolyData	No
vtkStructuredGrid	Yes
vtkUnstructuredGrid	No
vtkImageData	Yes

Multiple view representations

- https://docs.paraview.org/en/latest/UsersGuide/displayingData.html#id3



Scalar field visualization

A quick look at the classic visualization methods

- Slicing
- Clipping
- Iso-contouring
- glyphing
- Volume rendering with a more in-depth presentation in the next slides

Exercise

- You can test each of these filters/representations using a “Fast Uniform Grid” source, or using the file pvSphericalGrid.01.py
- Benefit: see an example of how to create a spherical grid (which is not a native grid type)

Volume rendering

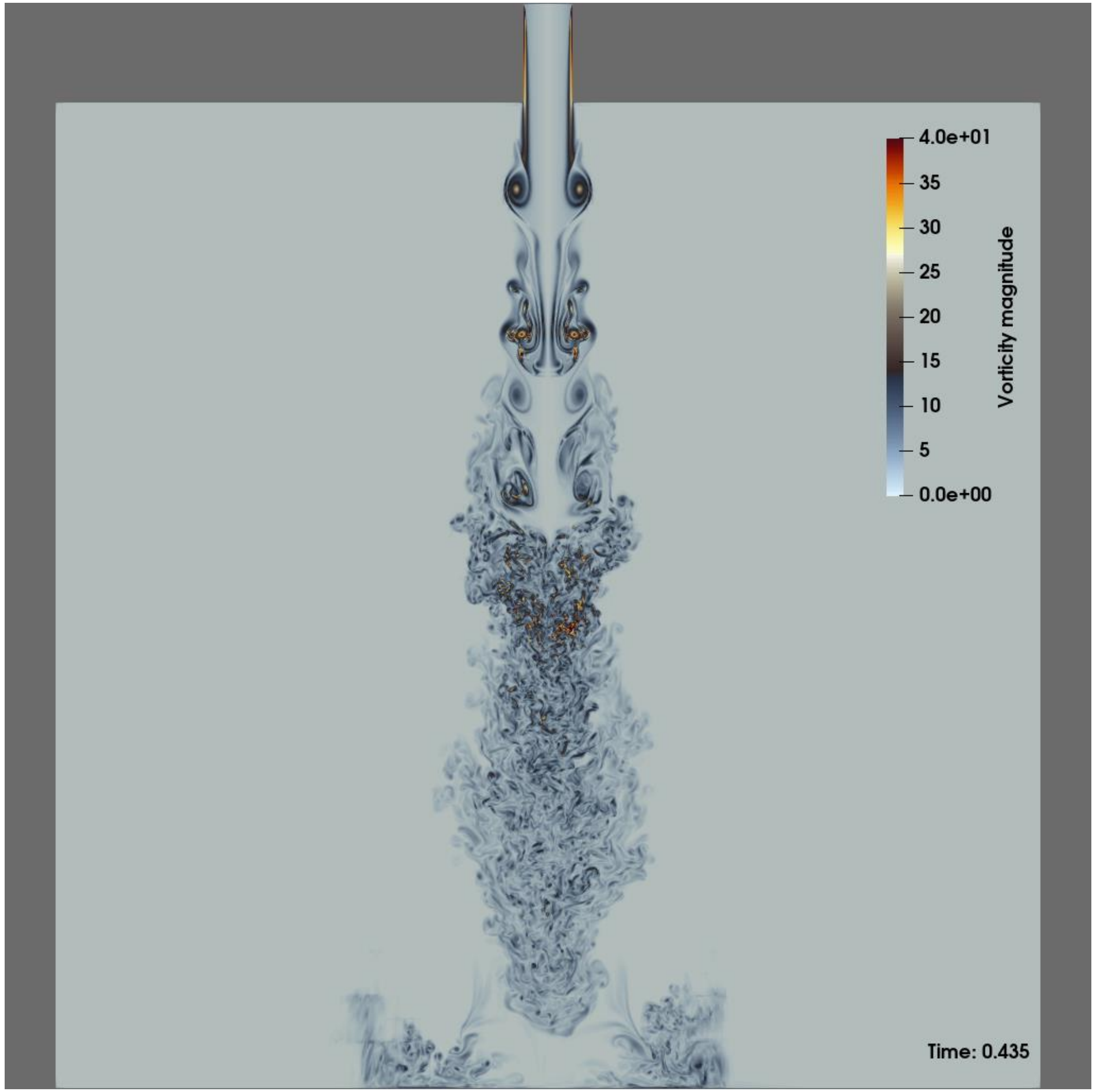
Examples:

- Turbulent Jet Ignition
- Sheared Thermal Convection

Motivations for Volume Rendering: Turbulent Jet Ignition

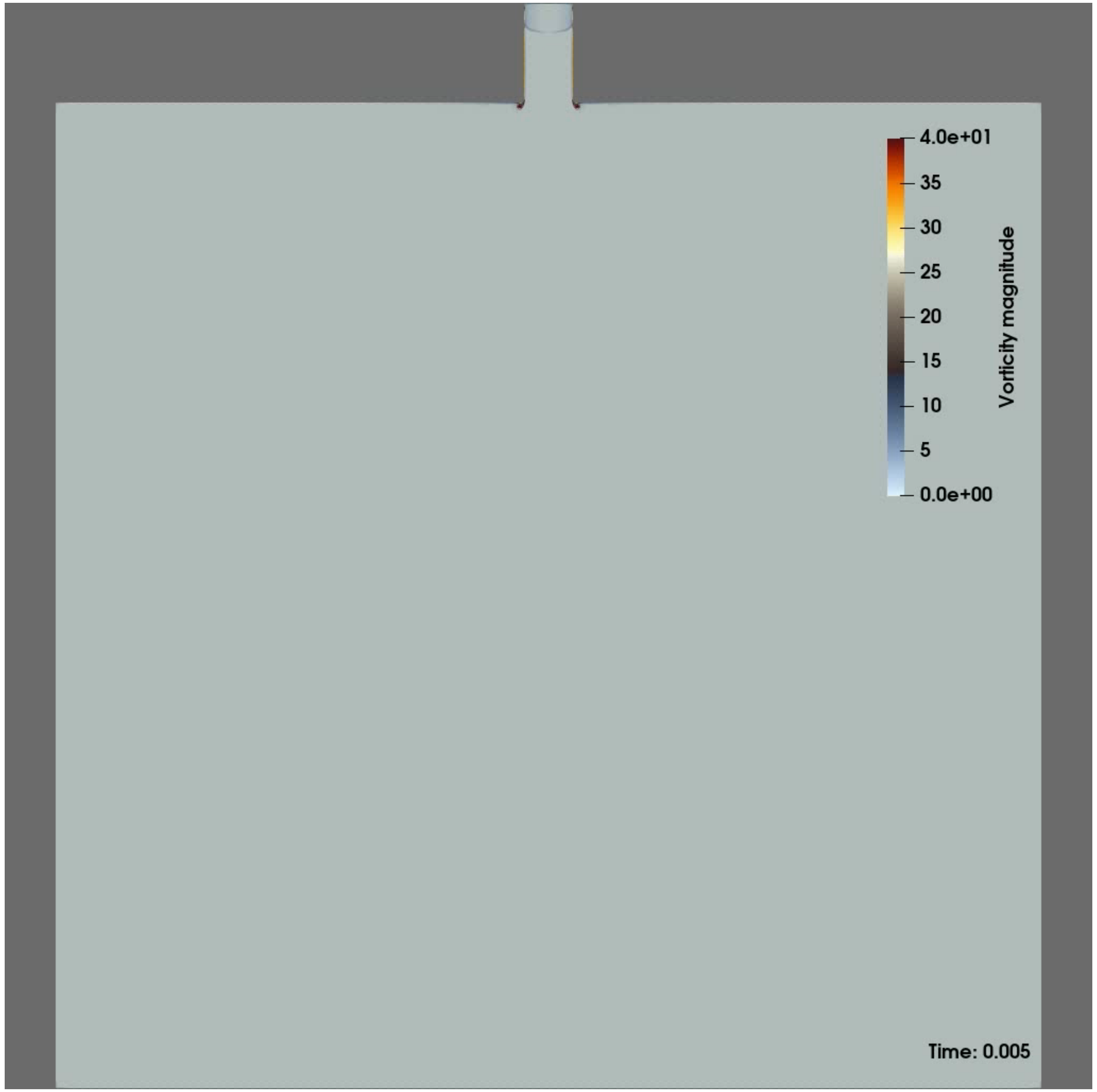
Turbulent Jet Ignition (TJI) is a technique to enhance ignition via a jet of hot reactive gases generated in a small pre-chamber. The jet initiates combustion in the main chamber, improves combustion efficiency and reduces pollutant emissions.

- 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.



Motivations for Volume Rendering: Turbulent Jet Ignition

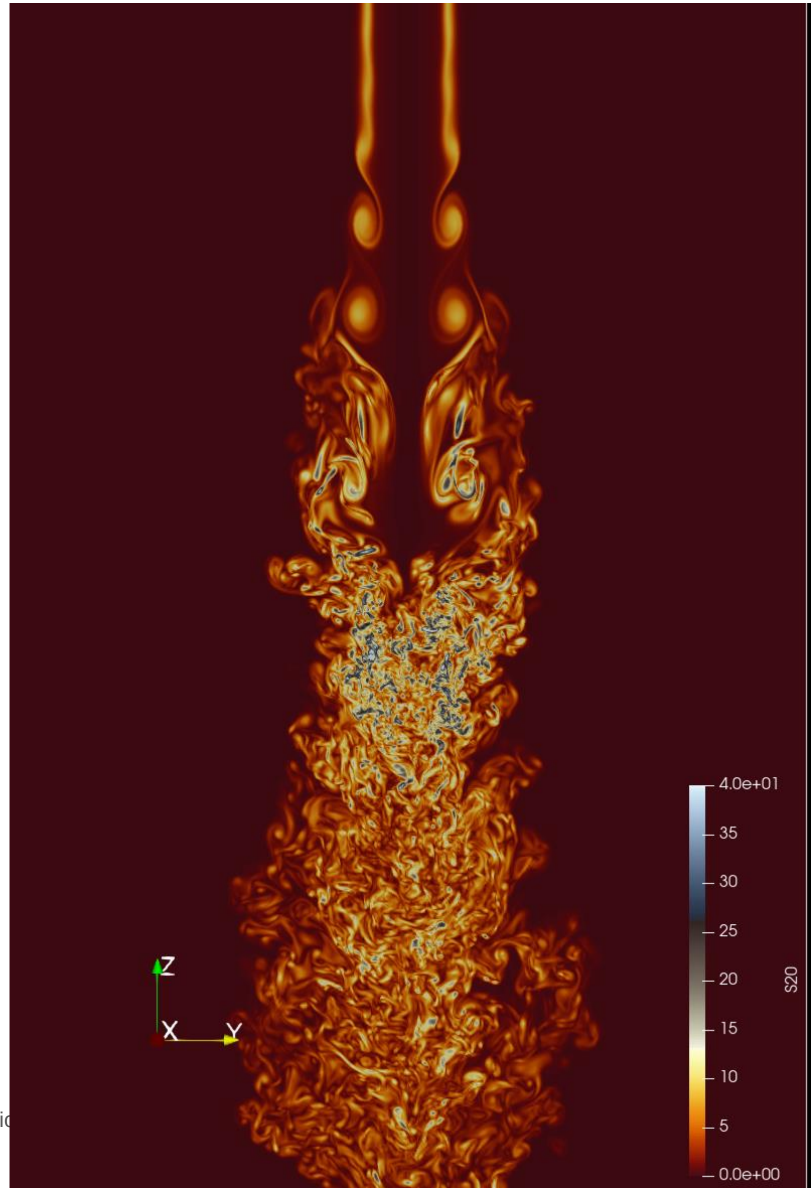
Slicing through a volume isn't enough to reveal the intricate features in this flow simulation, and the interaction between physical and chemical processes.

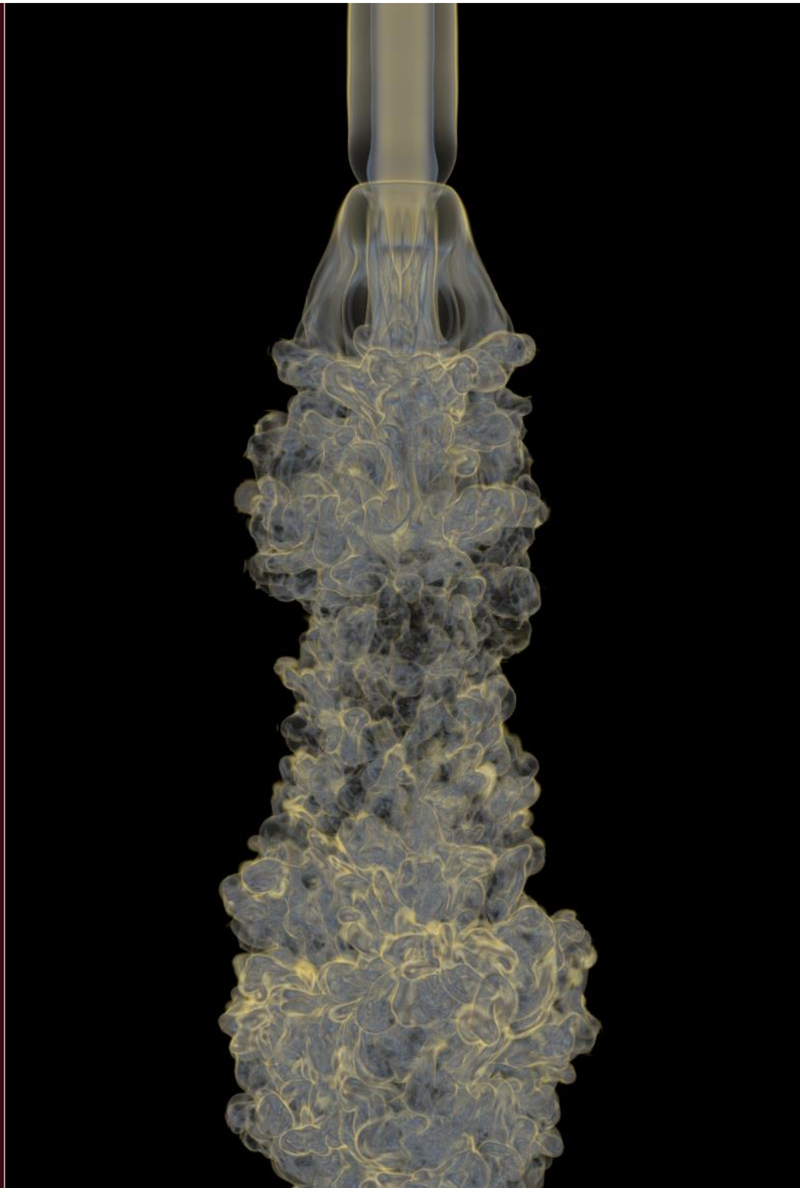


The need for 3D interactive 3D volume rendering

- 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

- Volume Rendering does a much better job and can be done at interactive speed.

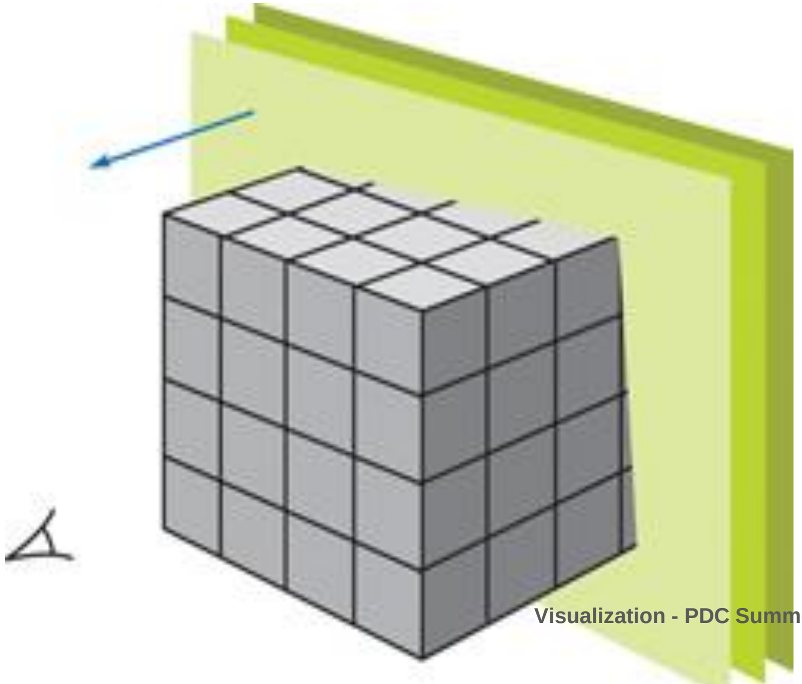


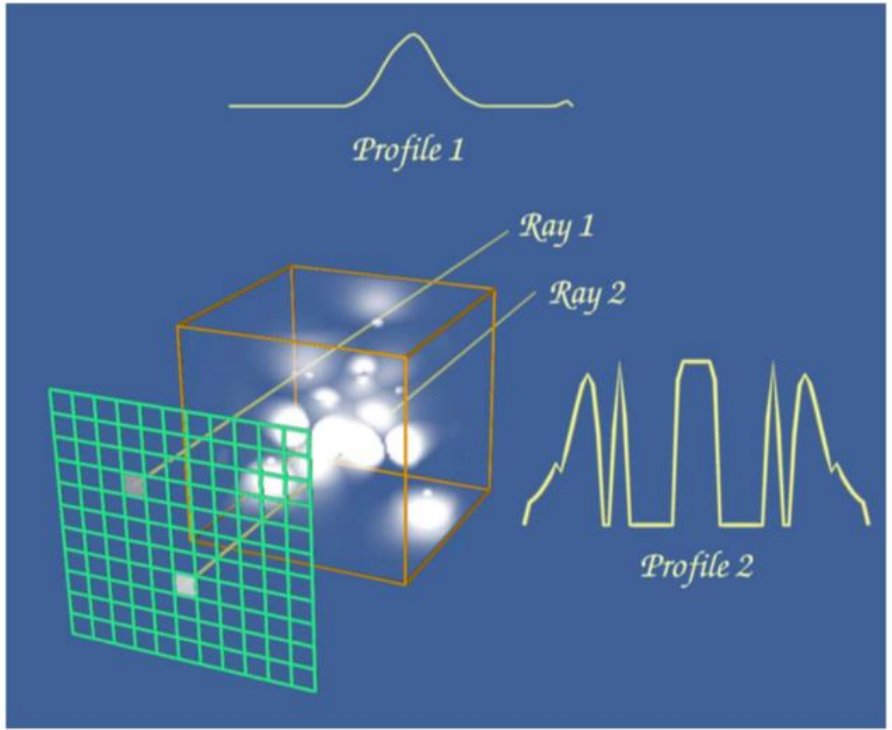


Volume Rendering (VR)

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





Volume Rendering Essentials

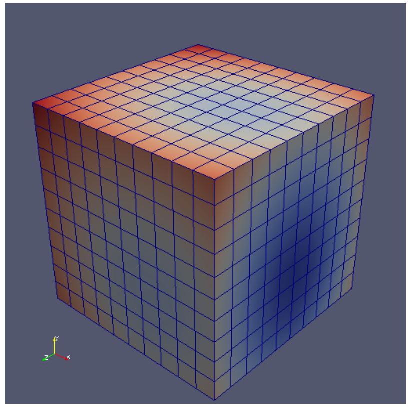
- https://www.kitware.com/products/books/VTKTextbook.pdf (Page 218)

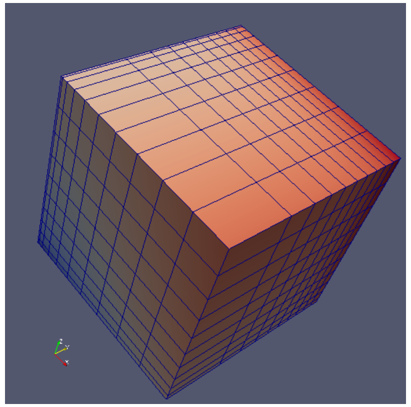
- See “inside” the volume.

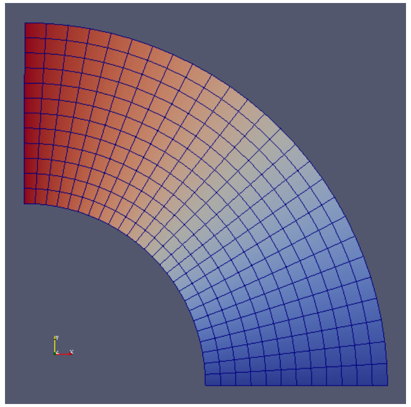
VTK Mesh Types

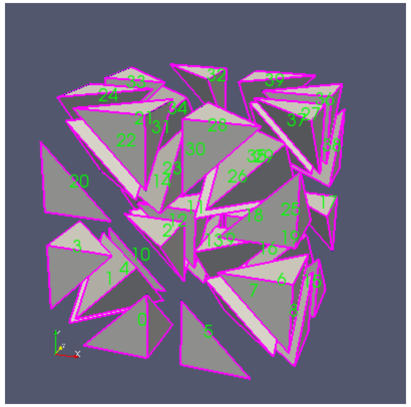
Most grid types are supported:

- vtkImageData: Yes
- vtkRectilinearGrid: Yes
- vtkStructuredGrid: Yes
- 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

Pseudo Volume Rendering 1

- See pvPseudoVolumeRendering.01.py

- # Perform pseudo volume rendering in a structured grid by compositing
- # translucent slice 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.

Volume Rendering an UnstructuredGrid ?

- Create a Fast Uniform Grid. Use “volume” representation

- Tetrahedralize and switch to ‘Volume”

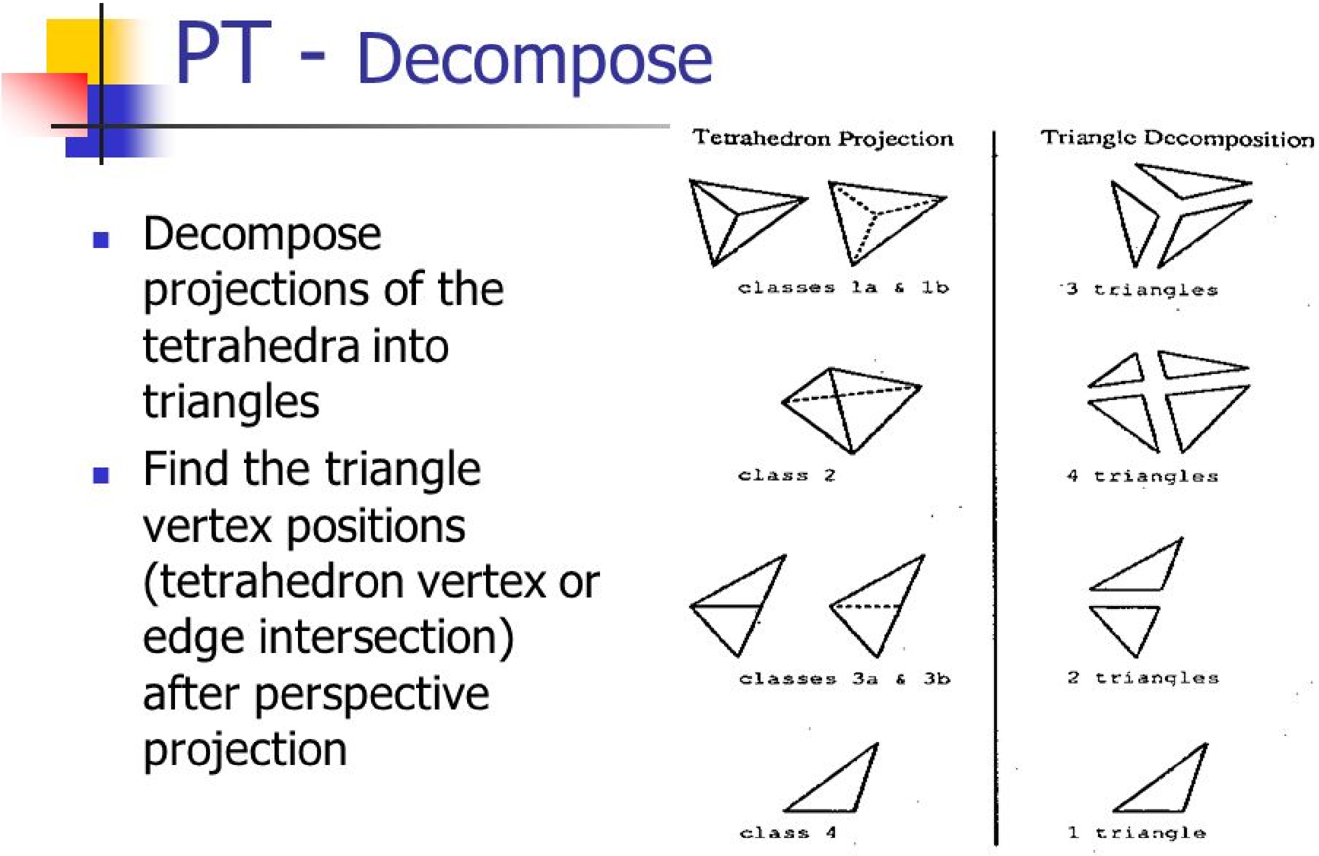
- The default mapper is the “OpenGL Projected Tetrahedra Mapper”

- 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



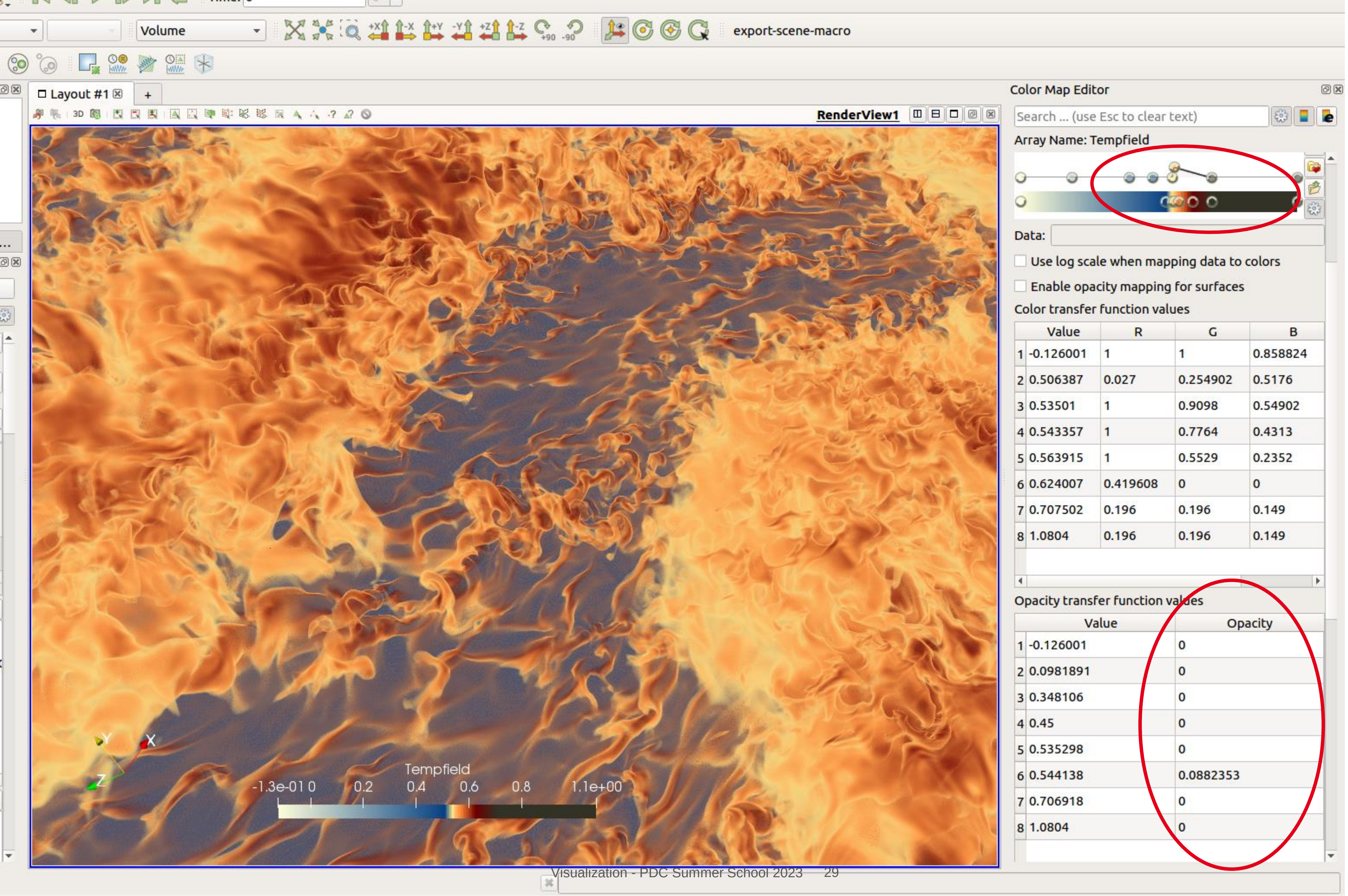
PT - Decompose

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

Resample to Image (for unstructured grids for example)

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

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



Array Name: Tempfield

□ Use log scale when mapping data to colors

□ Enable opacity mapping for surfaces

Color transfer function values

	Value	R	G	B
1	-0.126001	1	1	0.858824
2	0.506387	0.027	0.254902	0.5176
3	0.53501	1	0.9098	0.54902
4	0.543357	1	0.7764	0.4313
5	0.563915	1	0.5529	0.2352
6	0.624007	0.419608	0	0
7	0.707502	0.196	0.196	0.149
8	1.0804	0.196	0.196	0.149

Opacity transfer function values

	Value	Opacity
1	-0.126001	0
2	0.0981891	0
3	0.348106	0
4	0.45	0
5	0.535298	0
6	0.544138	0.0882353
7	0.706918	0
8	1.0804	0

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

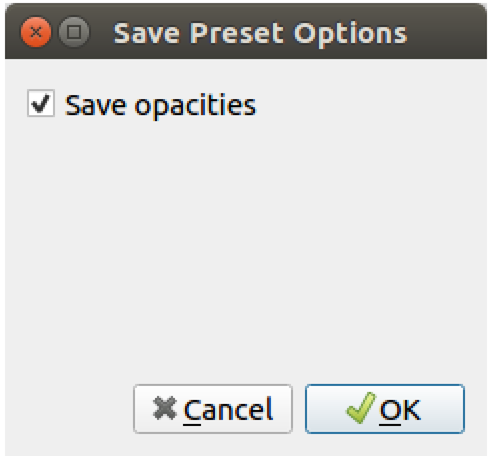
ImageData Volume rendering has three parallel libraries

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

- Published in August 2019

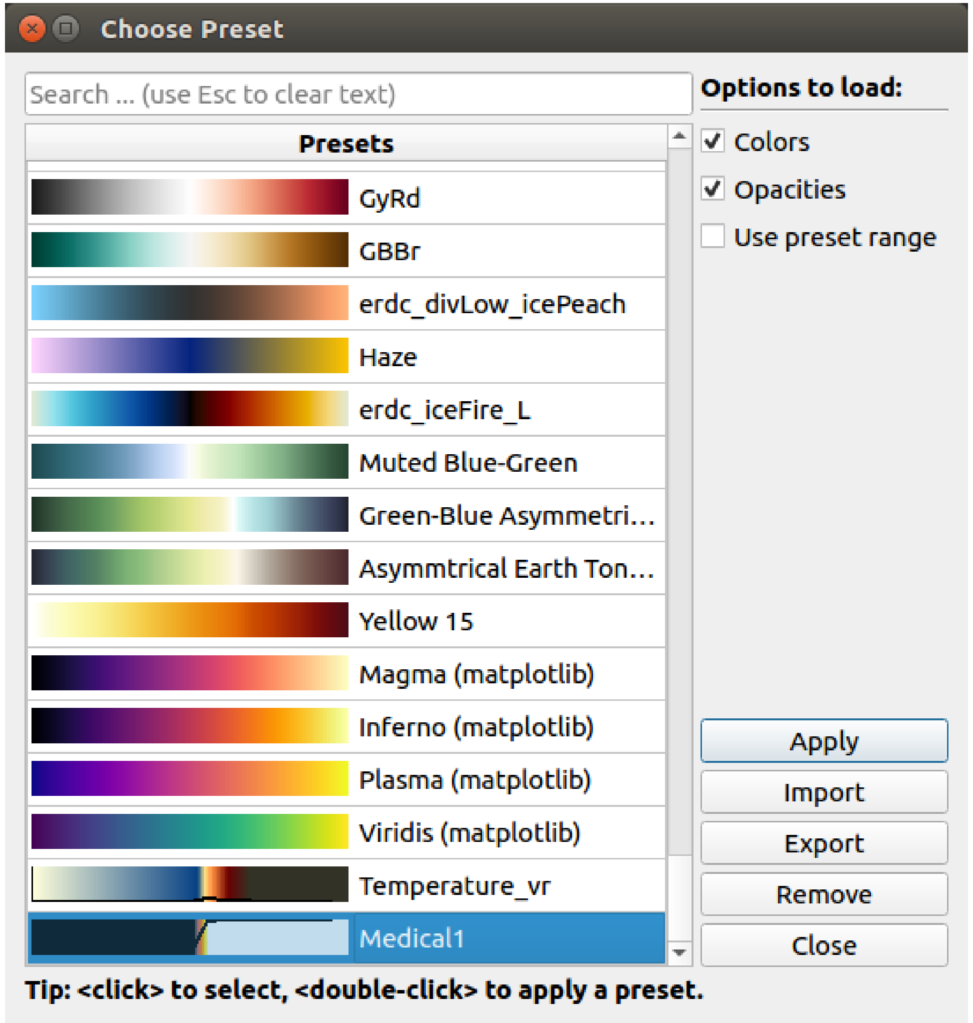
Sharing colormap and opacity transfer function

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



Save Preset Options

☑ Save opacities



Choose Preset

Search ... (use Esc to clear text)

Options to load:

☑ Colors
☑ Opacities
□ Use preset range

Presets

- GyRd
- GBBr
- erdc_divLow_icePeach
- Haze
- erdc_iceFire_L
- Muted Blue-Green
- Green-Blue Asymmetri…
- Asymmtrical Earth Ton…
- Yellow 15
- Magma (matplotlib)
- Inferno (matplotlib)
- Plasma (matplotlib)
- Viridis (matplotlib)
- Temperature_vr
- Medical1

Tip: <click> to select, <double-click> to apply a preset.

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.

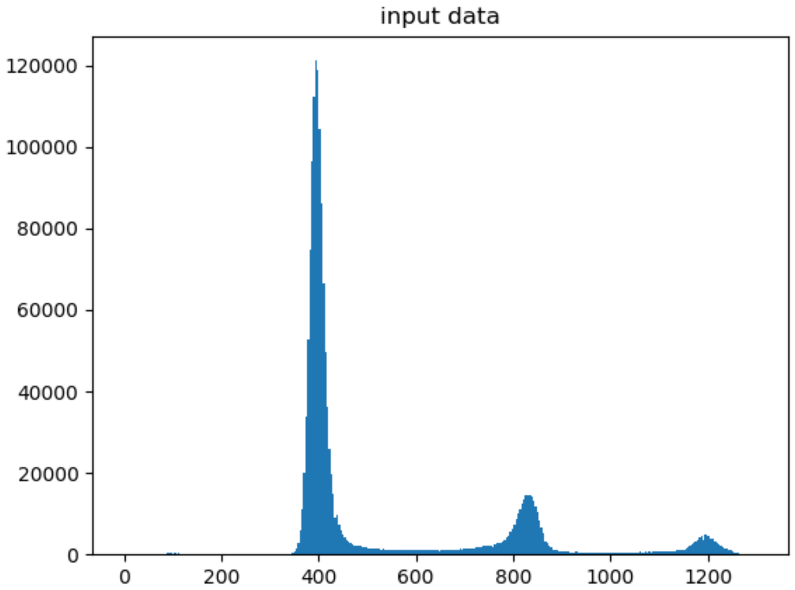
- New Visualization
  - Render several isosurfaces directly on the GPU from the Volume representation/mapper without mesh computation or specific data transfer.

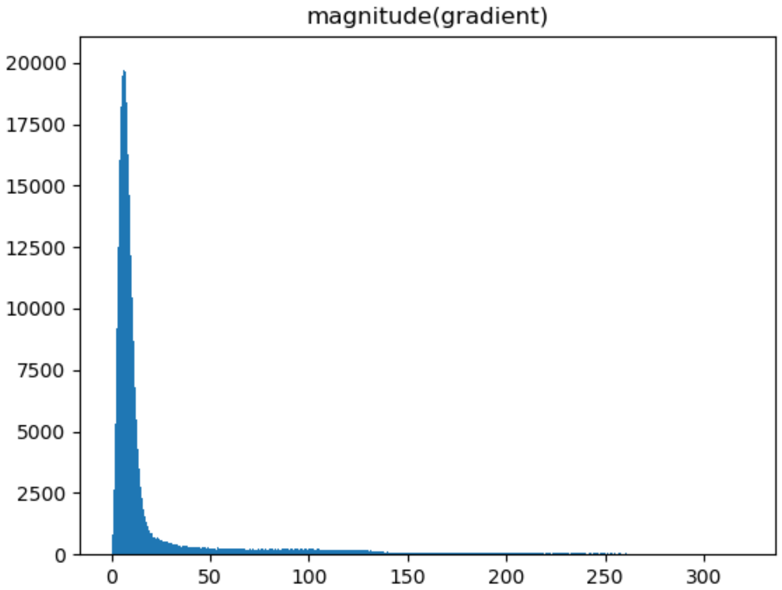
Exercise

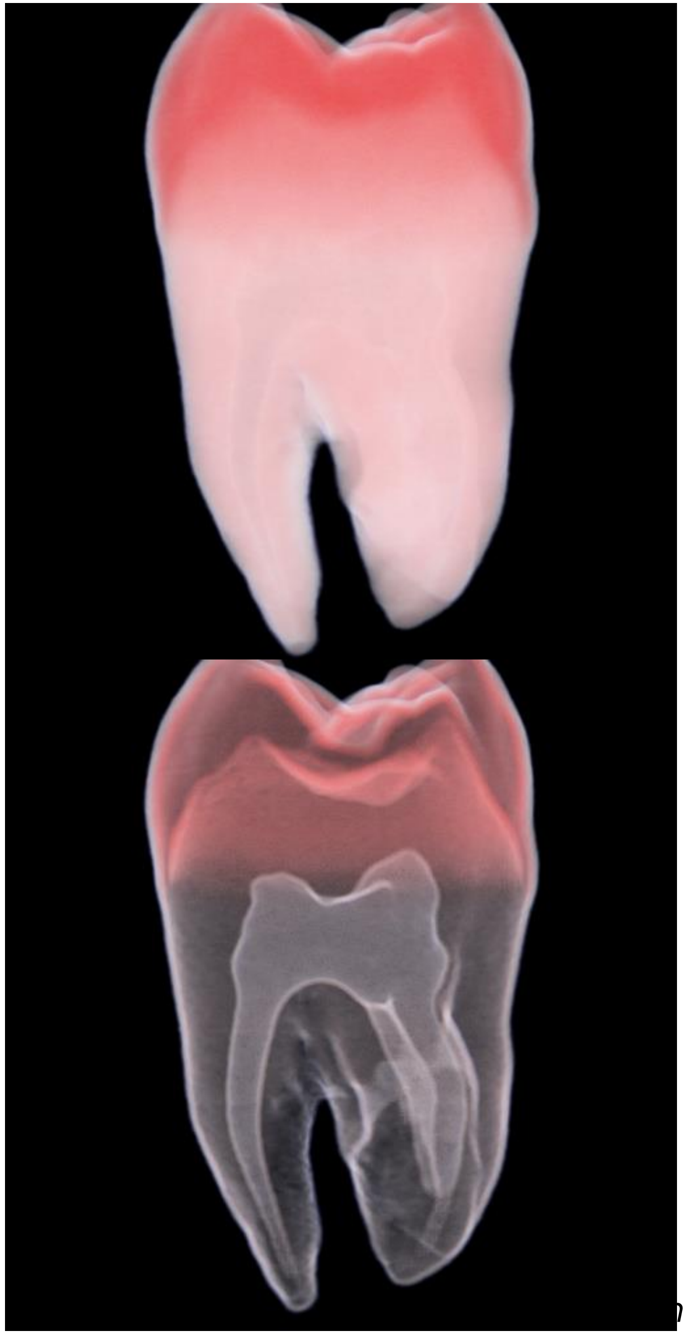
- Practice with pvGPU_Isosurfaces.01.py

There is more…

- VTK offers a gradient-based feature to highlight sub-volumes of higher gradient.
- Example: a tooth dataset


input data


magnitude(gradient)



The tooth dataset

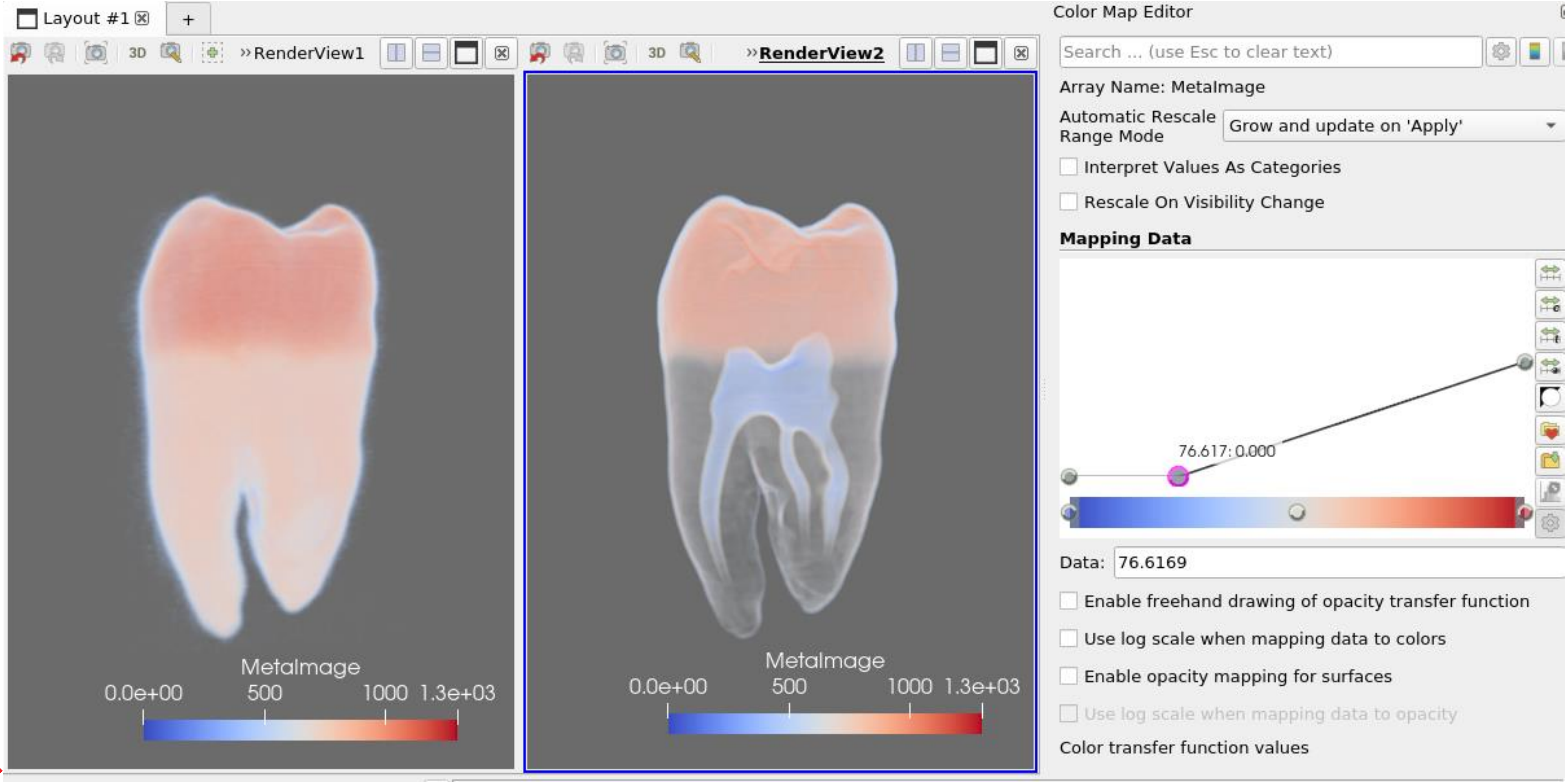
```
grad_fn = vtk.vtkPiecewiseFunction()
grad_fn.AddPoint(0, 0.0)
grad_fn.AddPoint(50, 0.0)
grad_fn.AddPoint(100, 1.0)
grad_fn.AddPoint(max(grad), 1.0)

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

- This is a VTK programming tip, specific to gradients.
- We can apply it by using a feature introduced in ParaView v5.9, enabling the use of separate transfer functions for color and opacity.

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 pvVolumeRenderingSeparateOpacity.py*

Volume rendering with a separate opacity array



Color Map Editor

Search ... (use Esc to clear text)

Array Name: MetaImage

Automatic Rescale Range Mode: Grow and update on 'Apply'

□ Interpret Values As Categories

□ Rescale On Visibility Change

Mapping Data

Data: 76.6169

□ Enable freehand drawing of opacity transfer function

□ Use log scale when mapping data to colors

□ Enable opacity mapping for surfaces

□ Use log scale when mapping data to opacity

Color transfer function values

Exercise

- I suggest to use two different (internal) data sources in the ParaView GUI
  - Fast Uniform Grid
  - Wavelet
- Try out all techniques (except Volume rendering) introduced above
- Add a “tetrahedralize” filter, and try again

Volume Rendering Exercise

- use the tooth dataset “tooth.mhd” and try to implement pseudo volume rendering with slices

Query-ing/filtering data

ParaView’s Find Data menu

- Demonstrations

- Limitations in version <= 5.11
- The Python commands to reproduce a data selection are not saved in ParaView’s session files. Following in the next 3 slides are examples of what you would have to do…

Query-based filtering examples

https://gitlab.kitware.com/paraview/paraview/blob/master/Wrapping/Python/paraview/selection.py

```
qs1="rho > 1e-05"
sel1 = QuerySelect(QueryString=qs1, Source=mergeBlocks1)
extractSelection1 = ExtractSelection(Input=mergeBlocks1)

rep1 = Show(extractSelection1)
rep1.Representation = 'Points'
ColorBy(rep1, ('POINTS','rho'))
```

Query-based filtering examples

```
# points is the array of coordinates. We select the 0-th coordinate < -12.15 AND rho > 1e-5

Qs2 = "np.logical_and(rho > 1e-05, points[:,0] < -12.15)"
sel2 = QuerySelect(QueryString=Qs2, Source=mergeBlocks1)
extractSelection2 = ExtractSelection(registrationName='ExtractSelection2', Input=mergeBlocks1)

rep2 = Show(extractSelection2)
rep2.Representation = 'Points'
ColorBy(rep2, ('POINTS','rho'))
```

Query-based filtering examples

```
# points is the array of coordinates. We select particles within a given Radius and Center


#Center = [-11.1431, 3.97869, -0.173805]; Radius = 10
Qs3 = "mag(points - [-11.1431, 3.97869, -0.173805]) < 10"
sel3 = QuerySelect(QueryString=Qs3, Source=mergeBlocks1)
extractSelection3 = ExtractSelection(registrationName='ExtractSelection3', Input=mergeBlocks1)



rep3 = Show(extractSelection3)
rep3.Representation = 'Points'
ColorBy(rep3, ('POINTS','rho'))
```

Query-based filtering

https://gitlab.kitware.com/paraview/paraview/blob/master/Wrapping/Python/paraview/selection.py

Other selections:

```
SelectThresholds(Thresholds=[-338000, 0], ArrayName= =‘Potential’, Source=GetActiveSource())
```

```
SelectIDs(IDs=[i for i in range(100000, 1100000)], , Source=GetActiveSource())
```

Vector field visualization

Vector field visualization

- Use a scalar field visualization technique
- Use glyphs representations (oriented arrows in the direction of the field)
- Use lines tangent to the vector field

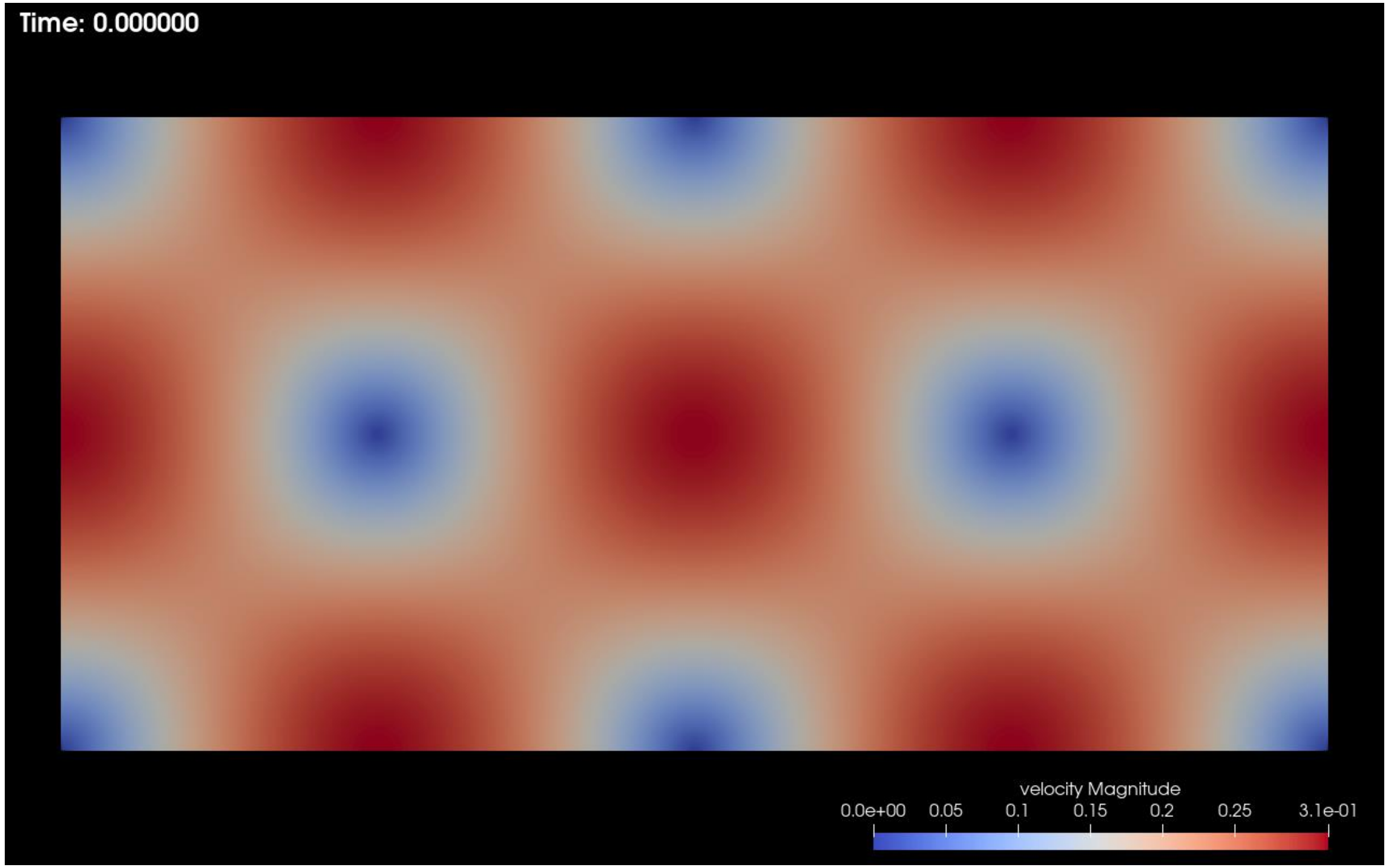
Time-dependent data:

  - Use particle traces
  - Use pathlines

Vector field visualization (1)

- Shade the vector field by its magnitude

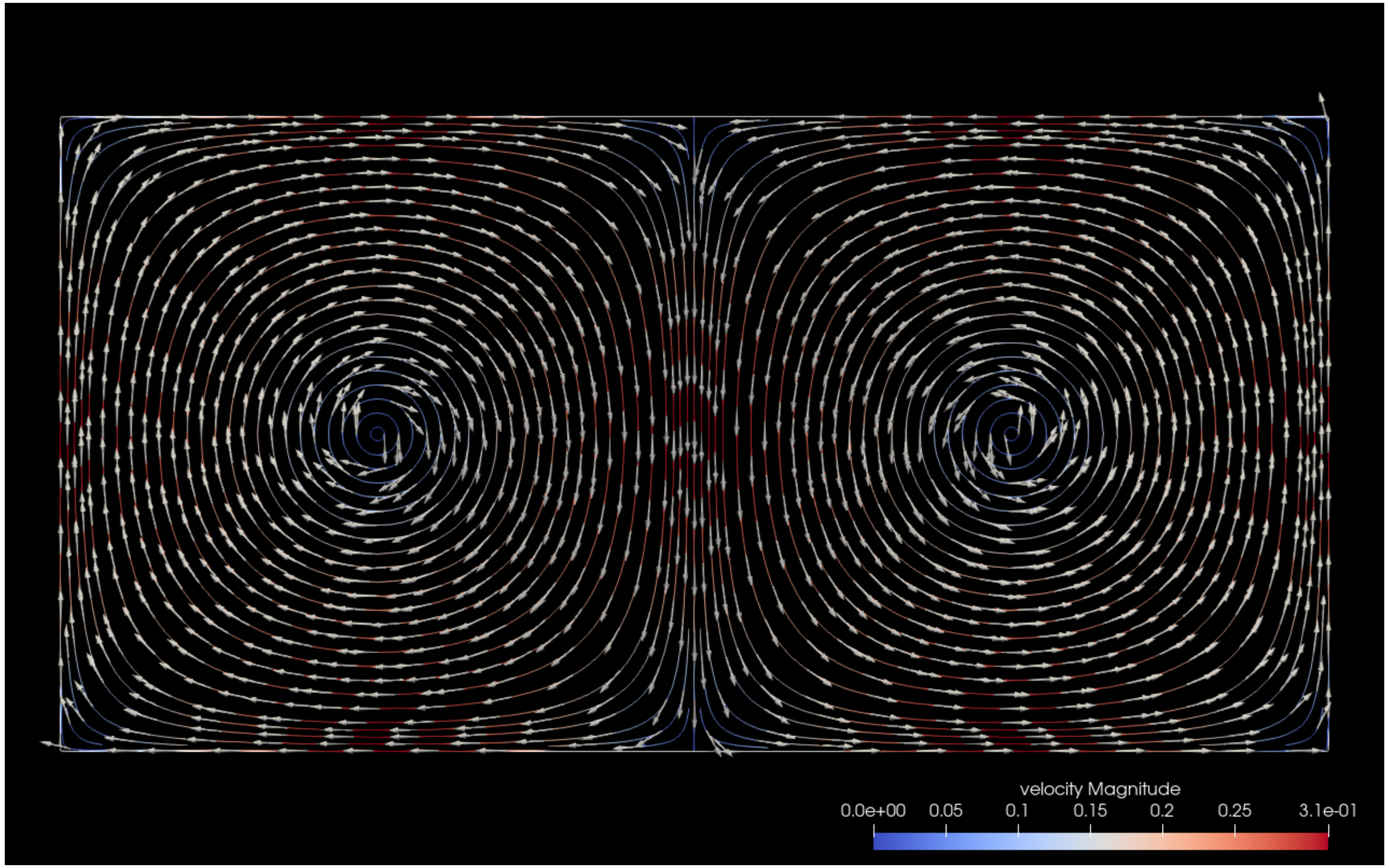
- Missing cues?
  - direction



Vector field visualization (2)

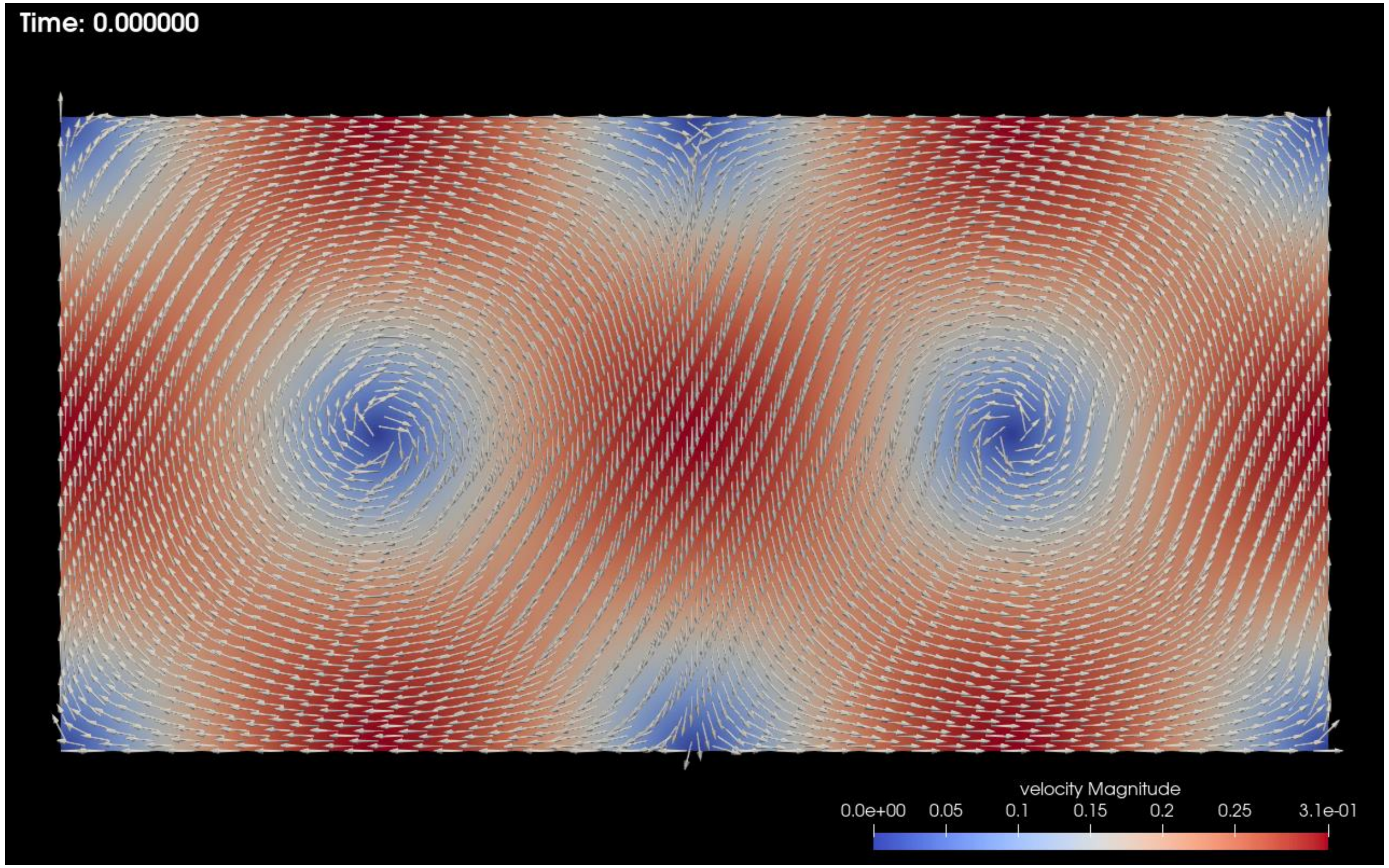
- Draw arrows oriented in the direction of the field

- Issues?
  - Make the density of arrows dependent on the zoom ration
- Difficulties?
  - Can be too complicated in 3D



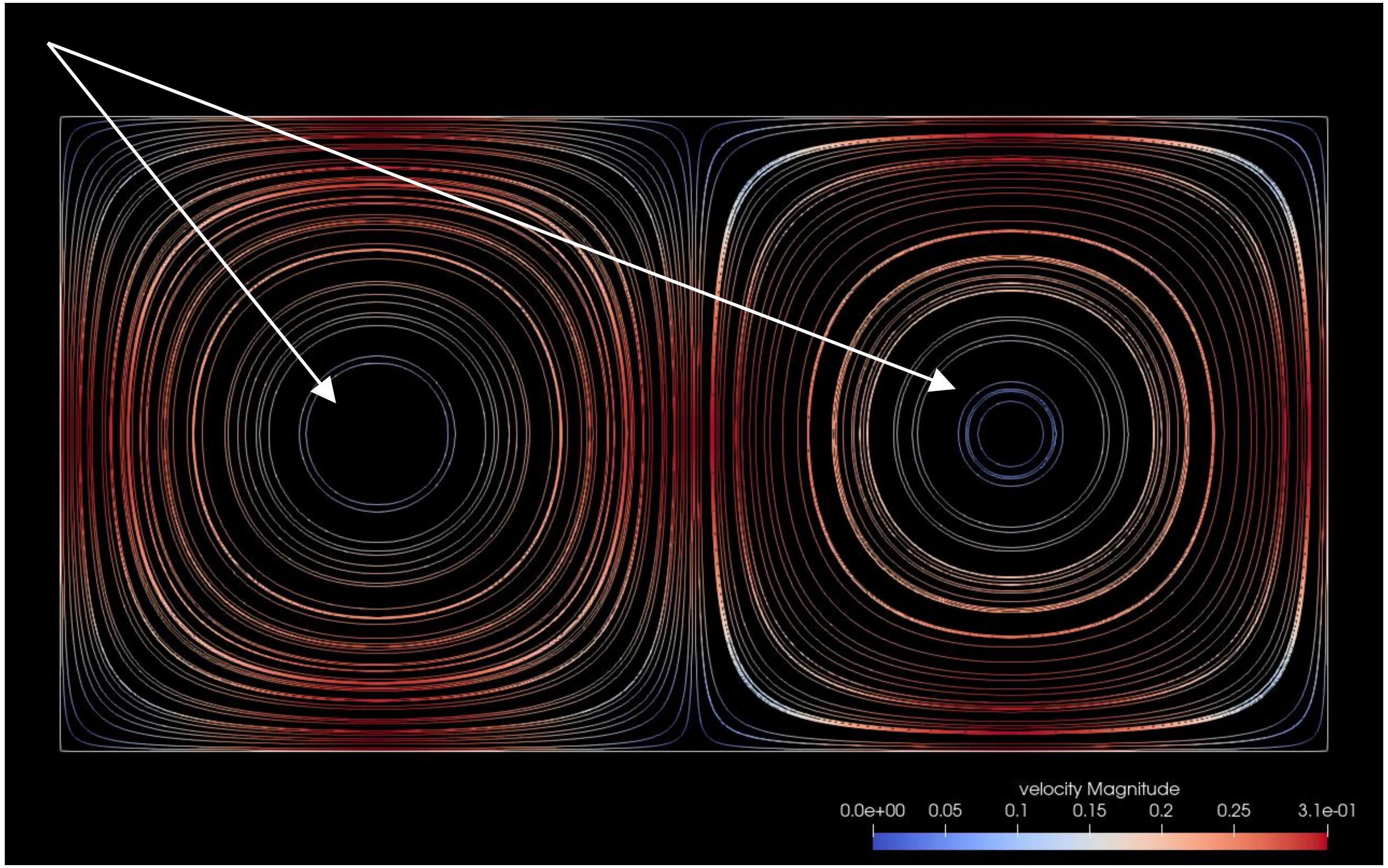
Vector field visualization (2)

- Apply both techniques seen earlier



Vector field visualization (2)

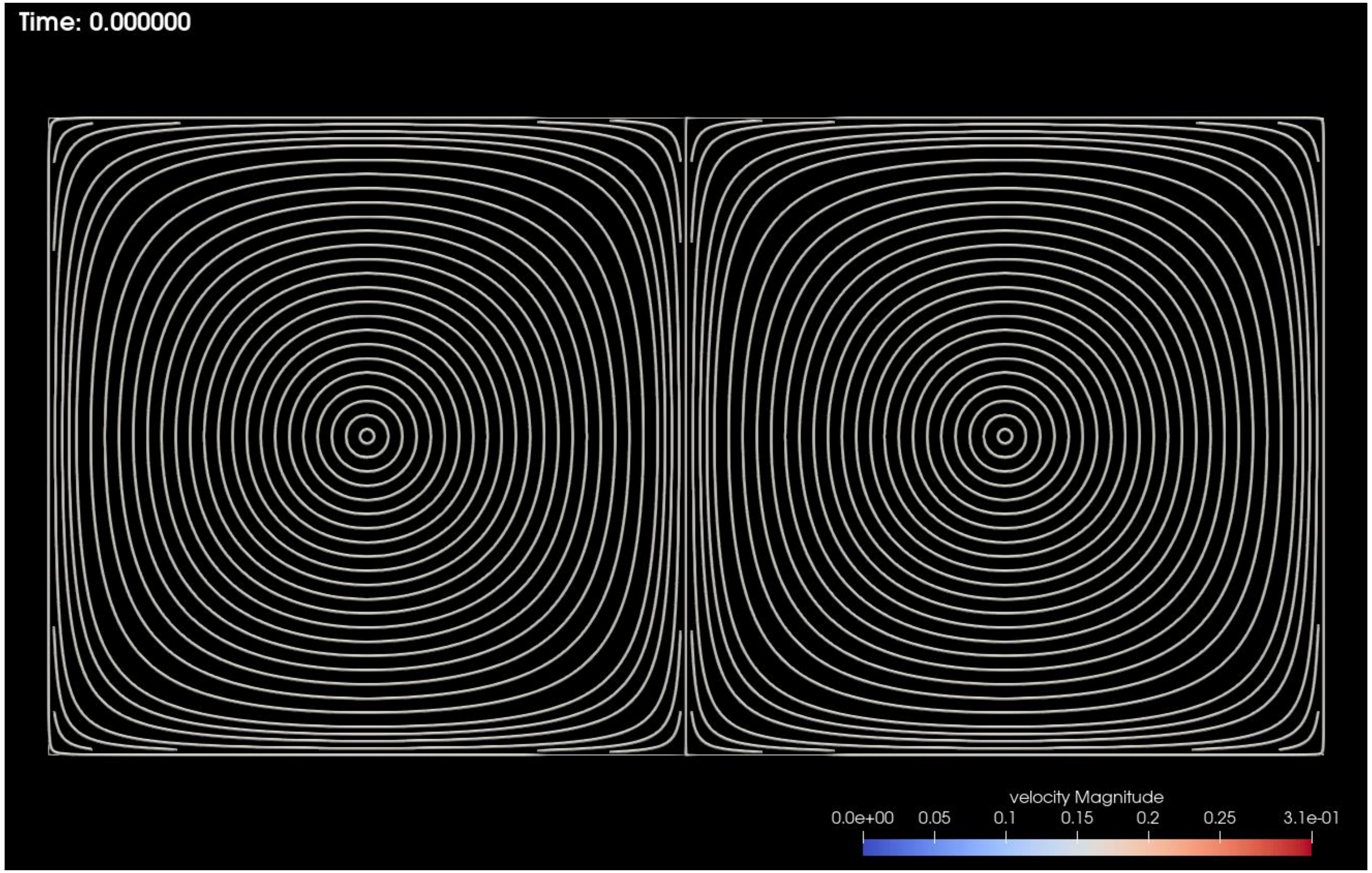
- Draw streamlines tangent to the vector field



- Difficulties?
  - too dense, or too sparse

Vector field visualization (2)

- Draw **evenly-spaced** streamlines tangent to the vector field

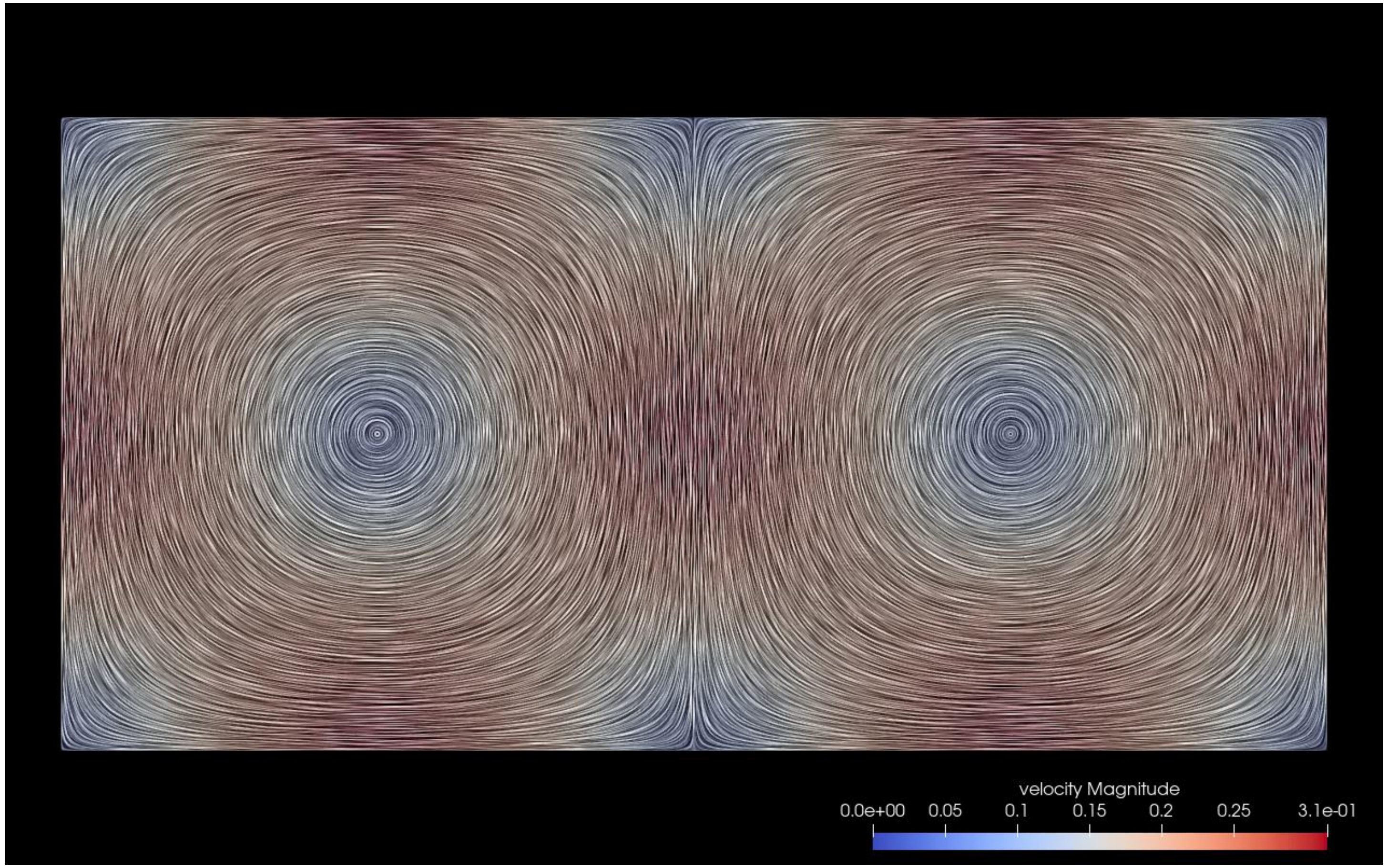


- Difficulties?
  - Missing in 3D

Vector field visualization (2)

- Use a screen-space representation to do a Linear Integral Convolution

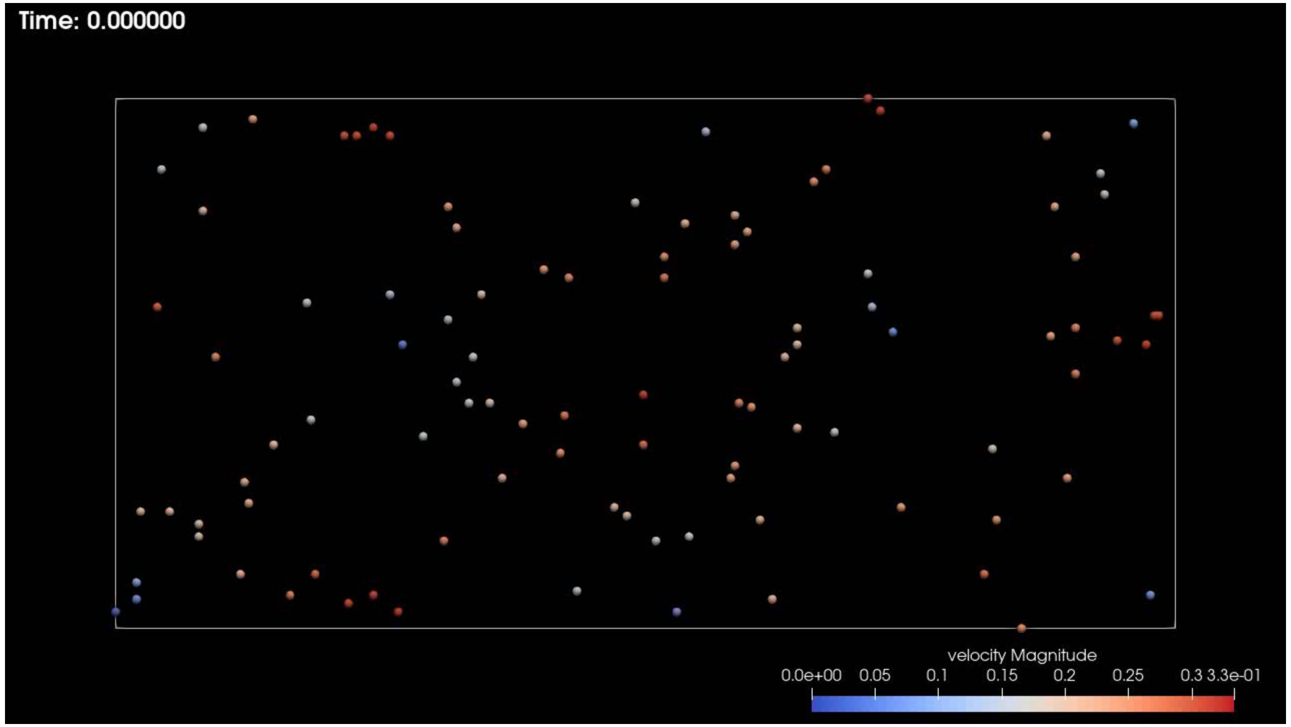
- Difficulties?
  - Missing in 3D
  - Not correct for transient data



Vector field visualization (2)

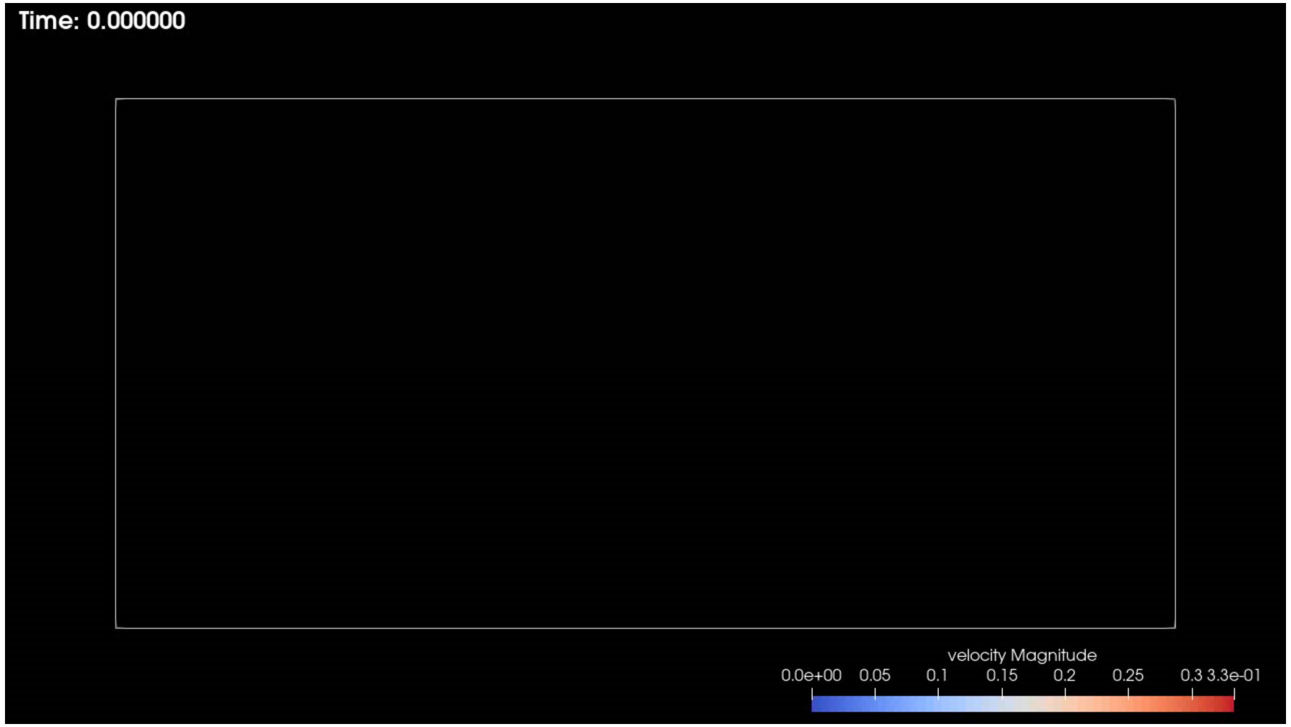
- Transient particles advected by the vector field

- Issues?
  - Particles disappearing
  - Can we re-inject particles at regular intervals?



Vector field visualization (2)

- Transient particles advected by the vector field



Exercise

- Login in to Dardel
- Execute pvTransientDoubleGyre.0*.py
- Make an animation (an mpeg file saved to disk) of temporal pathlines

Time series

- https://docs.paraview.org/en/latest/UsersGuide/dataIngestion.html#handling-temporal-file-series

- https://docs.paraview.org/en/latest/UsersGuide/dataIngestion.html#id1