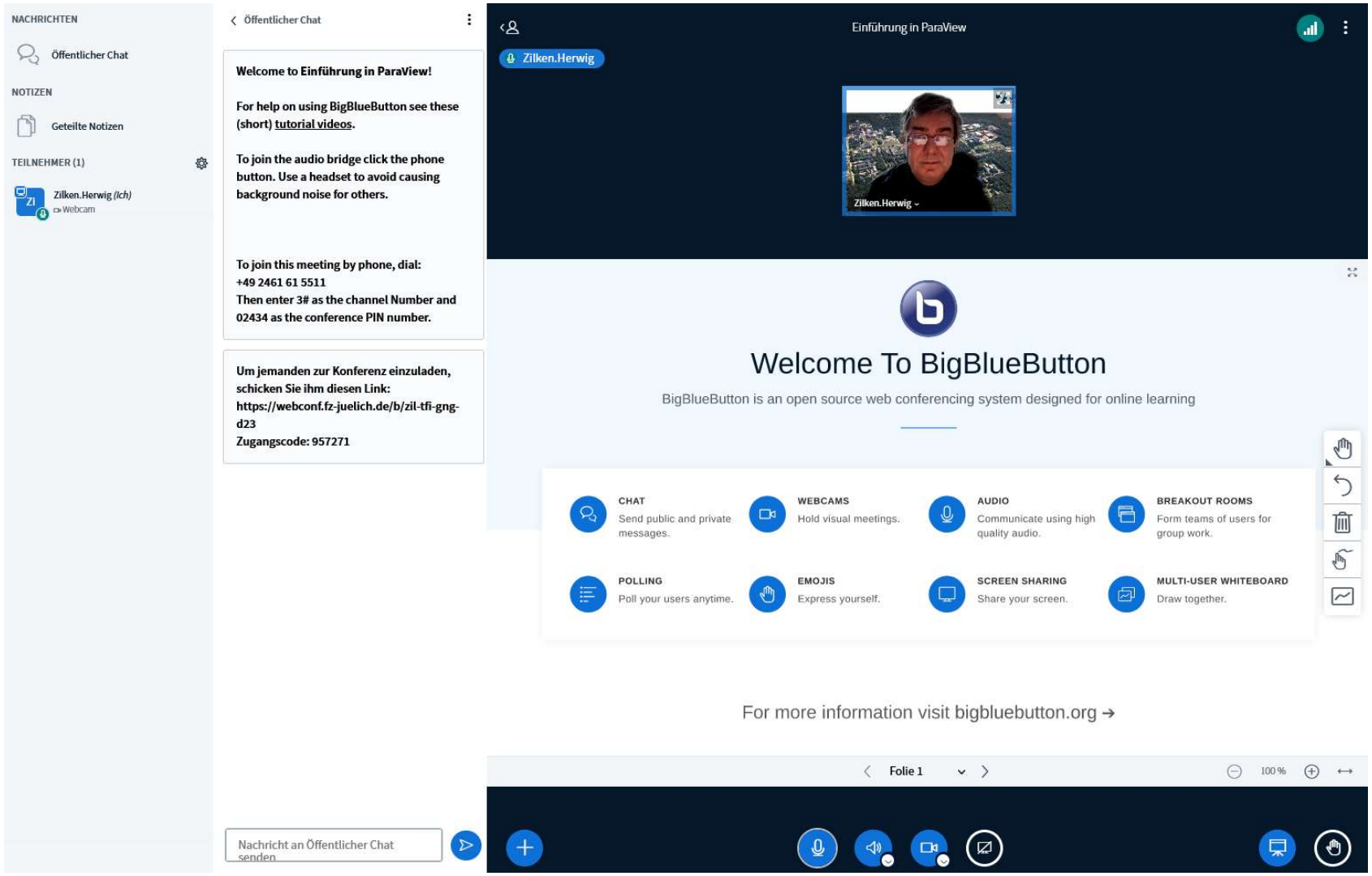
JÜLICH
Forschungszentrum

Introduction to ParaView

March 16/17, 2022 | Herwig Zilken

BigBlueButton

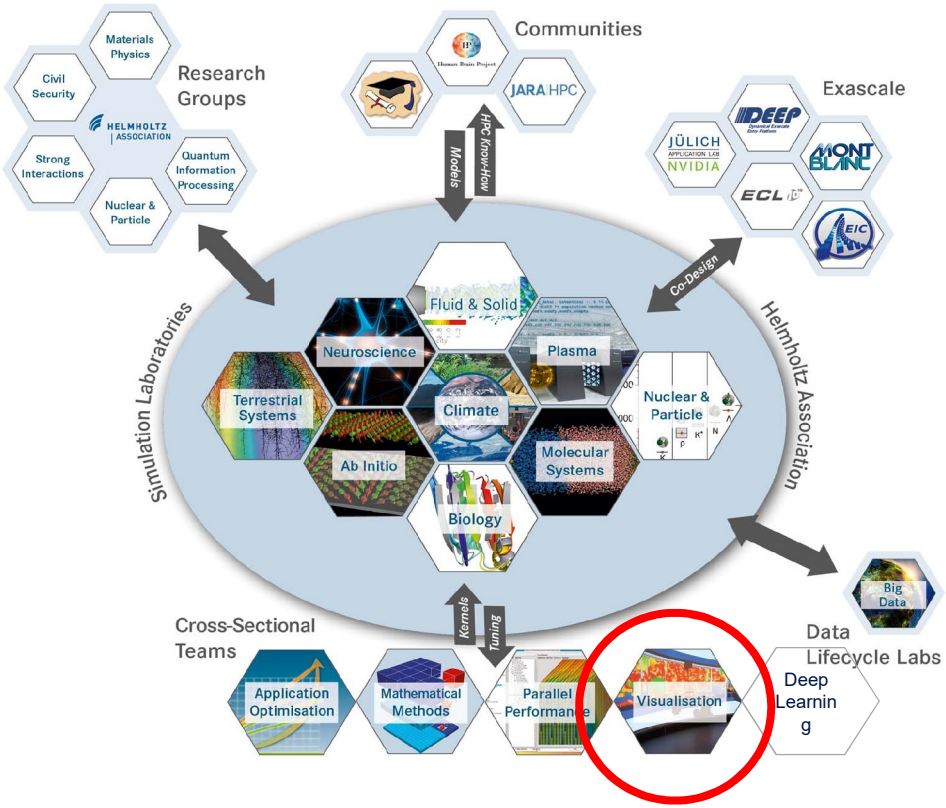


Visualization at JSC

Algorithm, Tools and Methods Lab Visualization

- **Scientific Visualization**
  - R&D + support
    for visualization of scientific data
- **Virtual Reality**
  - VR systems
    for the analysis and presentation
- **Multimedia**
  - multimedia productions
    for websites, presentations
    or on TV

Domain-specific User Support and Research at JSC



http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Visualization/_node.html

Outline

- Introduction
  - General Information
  - Basics of Visualization
- Getting Started
  - Getting Data into ParaView
  - User Interface (Pipeline Browser, Properties and Information Tab, ...)
  - Controlling the Camera
  - Exercise I (Creating a Source)
  - Exercise II (Loading and viewing Data)
- Applying Filters
  - Common Filters
  - Exercise III (applying filters, creating visualization pipelines)
  - Exercise IV (Streamlines, Volume Rendering)
- Color Maps
- Multiple Views
  - Comparative View
  - 2D Plots
  - Exercise V (Plotting)
- Text Annotation
- Time in ParaView
  - Exercise VI + VII (Time Dependent Data)
- Animations
  - Exercise VIII
- Visualizing Large Models
  - ParaView on HPC Systems
- Python-Scripting
  - Exercise VII
- Plugins

- Documentation, Useful Links

This course is based on Kitware's Paraview Tutorial.

What is ParaView?

- Open-source data analysis and visualization application (two- and three-dimensional data sets)
- Built on top of VTK => provides a comprehensive suite of visualization algorithms
- Supports many different file formats for both loading and exporting data sets.
- Supported platforms: Linux, Windows, Mac
- Processing Modes:
  - Stand-alone mode
  - Client server configuration
  - Batch

- Available at JSC on HPC Systems JUWELS and JURECA-DC

The History of ParaView

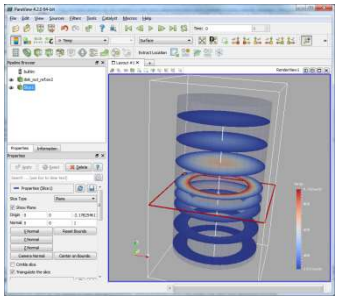
- **2000**: collaboration between Kitware Inc. and Los Alamos National Laboratory, funding provided by the US Department of Energy ASCI program
- **2002**: first release of ParaView
- **September 2005 - May 2007**: development of Paraview 3 by Kitware, Sandia National Lab. and other partners
  - ✓ user interface more user friendly
  - ✓ quantitative analysis framework
- **June 2013**: ParaView 4
  - ✓ more cohesive GUI controls
  - ✓ better multiblock interaction
  - ✓ In situ integration into simulation and other applications (Catalyst)
- **Recent Releases**: (currently ParaView 5.10.0)
  - ✓ Since 5.6: brand-new rendering backend with OpenGL 3.2 features (big change to the rendering infrastructure in ParaView's lifetime) since ParaView 5.0, e.g. new VR backend (multi screen 3D projection, VR devices)
  - ✓ Since 5.9: Extractors

Getting Started

- ParaView can be downloaded at www.paraview.org
  - precompiled versions available for Mac OS, Windows and Linux
  - sources for individual installations (e.g. using the mpi-version tailored to your system)

- ParaView client launches like most applications:
  - Windows: launcher located at the Start-Menu
  - Linux: execute paraview from a command prompt
  - Mac OS: open the application bundle that you installed

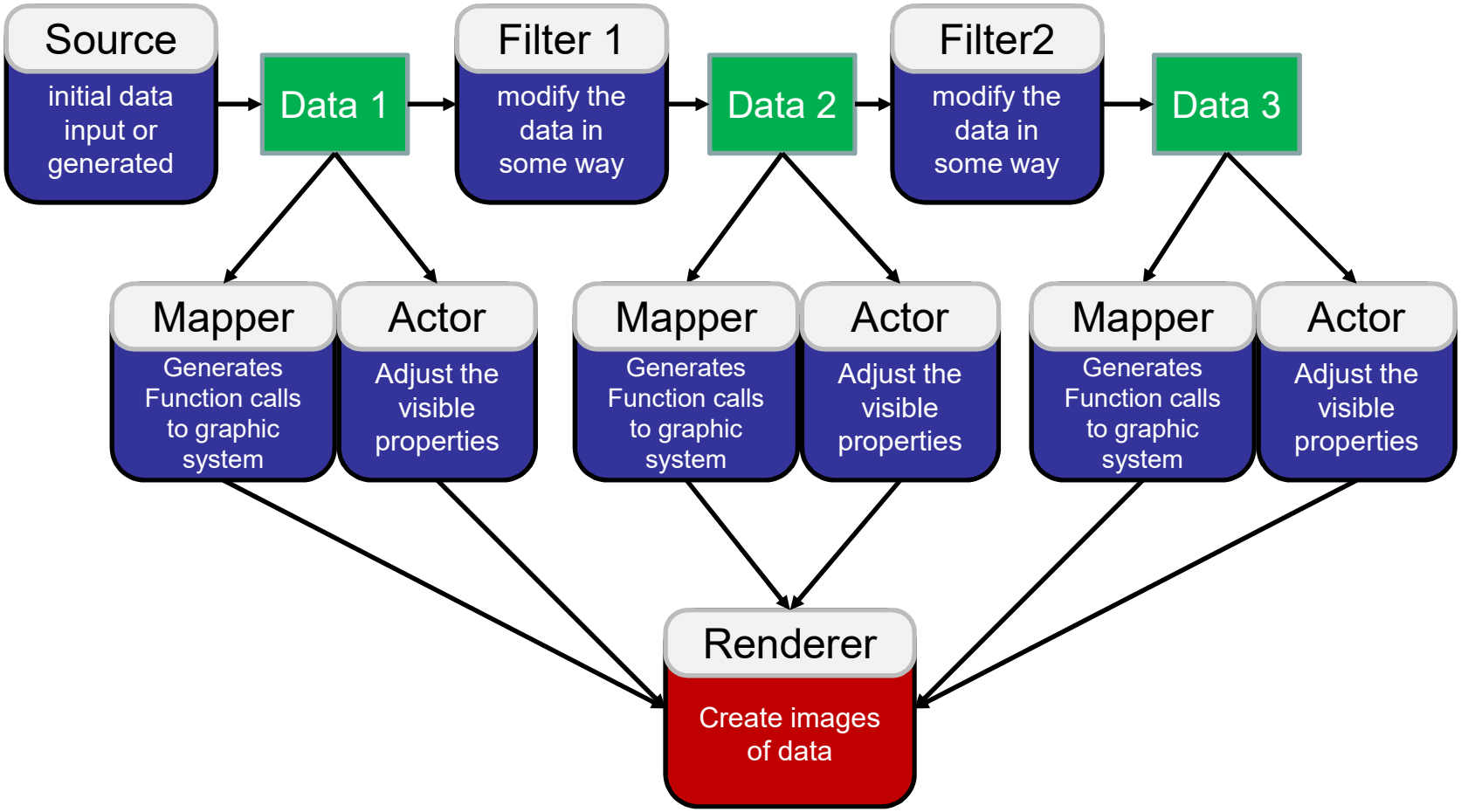
ParaView‘s Architecture



ParaView Client	pvpython	ParaView Web	Catalyst	Custom App
UI (Qt Widgets, Python Wrappings)				
ParaView Server				
VTK				
OpenGL	MPI	IceT	many more ...	

Visualization Pipeline

- concept of a visualization pipeline as implemented in VTK



Why use Paraview?

- Free and actively supported by major DOE laboratories
- Modular architecture => Separate processes possible for Data processing, Rendering, User interface
- Easy to use: all modes look the same (Stand alone, Client/Server)
- Can handle large files
- Built on VTK, which has a lot of more functionality

Drawbacks

- Not 100% stable
- Not well documented, in particular the internals

Scientific Data

The data describes WHAT values are located WHERE.

Example: what temperature is at location (x, y, z) = (1, 3, 5)

WHERE: the **structure** of the data

1. **Geometry**: defines the (3D) location
2. **Topology**: describes the connectivity (neighborhood) of points, defines which points form a cell

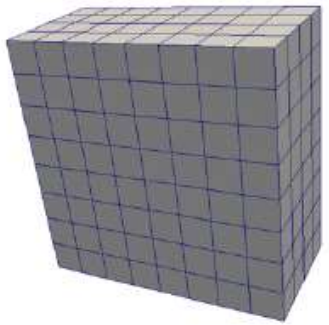
   Example: three points at (x_1,y_1,z_1), (x_2,y_2,z_2), (x_3,y_3,z_3) forming a triangle

WHAT: the **attributes** (values) of the data, e.g. temperature, pressure, …

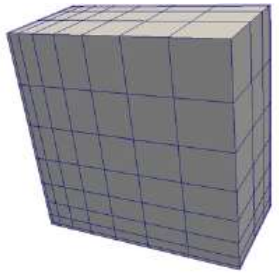
Typically the data is discrete (not continuous), given at a set of points in 3D space.

Data Types: Structured Data

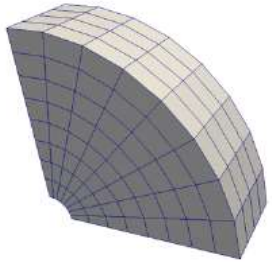
Uniform Rectilinear Grid (Image Data)
(vtkImageData)



Non-uniform Rectilinear Grid (Rectilinear Grid)
(vtkRectilinearData)



Structured (Curvilinear) Grid
(vtkStructuredData)



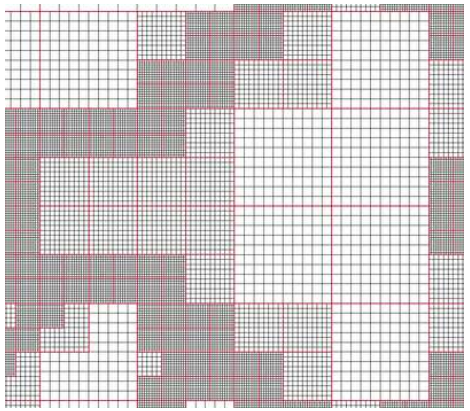
Data Types: Unstructured Data

Polygonal Mesh (Poly Data)
(vtkPolyData)

Unstructured Grid
(vtkUnstructuredGrid)

Multi-Block (several data sets grouped together)

Hierarchical Adaptive Mesh Refinement (AMR) ⟶



Data Attributes

- Data attributes at grid points (grid data, node data) or on cells (cell data, element data):
- General data attributes:
  - Scalar
  - Vector
  - Tensor (nxn matrix)
- Attributes with special meaning for the visualization process:
  - Normals (3D vector)
  - 2D or 3D texture coordinates
- Any number of attributes can be defined at points and cells

Supported Data File Formats

Some examples:
- ParaView files (.pvd)
- VTK xml based files (.vtp, .vti, .vtr, .vts, .vtu)
- Parallel xml based VTK files (.pvtp, .pvti, .pvtr, .pvts, .pvtu)
- VTK MultiBlock (.vtm, .vtmb, .vtmg, .vthd, .vthb)
- Legacy VTK files (.vtk for all types of data)
- Parallel legacy VTK files (.pvtk)
- EnSight files (.case)
- EnSight Master Server files (.sos)
- Exodus II files (.g, .e, .ex2, .ex2v2, .exo, .gen, .exoII)
- BYU (.g)
- XDMF files (.xmf, .xdmf)
- PLOT3D files (.xyz, .q)
- SpyPlot CTH files (.spcth)
- DEM files (.dem)
- VRML files (.wrl)
- PLY Polygonal files (.ply)
- Protein Data Bank files (.pdb)
- Xmol Molecule files (.xyz)
- Stereo Lithography files (.stl)
- Gaussian Cube files (.cube)
- Raw (binary) files (.raw)
- AVS files (.inp)
- Meta Image files (.mhd, .mha)
- Facet files (.facet)
- PNG files (.png)
- LSDyna files (.d3plot, .k, .lsdyna)
- Phasta files (.pht)
- SESAME files (.sesame)
- Comma-separated values (CSV) files (.csv)

Full list at: https://www.paraview.org/Wiki/ParaView/Users_Guide/List_of_readers

Input File Formats – VTK as an Example

VTK Family
(https://www.vtk.org/wp-content/uploads/2015/04/file-formats.pdf)

- Legacy serial (.vtk)
  - polygonal, uniform, rectilinear, curvilinear, unstructured
- Legacy parallel
  - same as legacy serial, but data is partitioned into several files
- XML-based
  - uniform (images, .vti)
  - rectiliniear (.vtr)
  - curvilinear (.vts)
  - polygonal (.vtp)
  - unstructured (.vtu)
- XML-based parallel
  - same as serial XML, but data is partitioned into several files

Converting Data to VTK Format

- Example on next slide: time dependent vector data on structured grid
- Converter to convert ascii data has to VTK XML was implemented
- VTK provides a number of source and writer classes to read and write popular data file formats
- In this example an XML file format of type vtkStructuredGrid (. vts) was chosen
- Result of converted data:
  - xml file including the name of the dataset for each timestep
  - n datasets files, each including data of each timestep

XML based StructuredGrid Fileformat: .vts

This is the description dataset, which lists the names of all existing time datasets in an XML-based format

```
<?xml version="1.0"?>
<VTKFile type="Collection" version="0.1" byte_order="LittleEndian">
<Collection>
<DataSet timestep="50"group="" part="0" file =
   "/private/sutmann/vts_Exercise_50.vts"/>
<DataSet timestep="100"group="" part="0" file =
   "/private/sutmann/vts_Exercise_100.vts"/>
<DataSet timestep="150"group="" part="0" file =
   "/private/sutmann/vts_Exercise_150.vts"/>

.....

</Collection>
</VTKFile>
```

XML Data (first timestep) : vts_Exercise_50.vts

```
<?xml version="1.0"?>
<VTKFile type="StructuredGrid" version="0.1" byte_order="LittleEndian">
  <StructuredGrid WholeExtent="0 47 0 47 0 151">
  <Piece Extent="0 47 0 47 0 151">
    <PointData>
      <DataArray type="Float32" Name="Velocity" NumberOfComponents="3" format="ascii">
        -0.21008299291 -0.051097899675 -0.04968290031 0.16106699407 -0.065898902714 0.010098299943
        0.10389400274 0.19959899783 -0.16598799825 0.15902400017 0.20383499563 -0.017544399947
        ……
      </DataArray>
    </PointData>
    <CellData>
    </CellData>
    <Points>
      <DataArray type="Float32" Name="Points" NumberOfComponents="3" format="ascii">
        -48.916801453 -49.018299103 -149.00500488 -46.871700287 -48.936100006 -149.16099548
        -44.773601532 -48.920101166 -149.07600403 -42.637401581 -48.958599091 -149.1734432
        . . .
      </DataArray>
    </Points>
  </Piece>
  </StructuredGrid>
</VTKFile>
```

HDF5

- Hierarchical data format: HDF5
- Efficient, portable IO library for storing scientific data
- High level of abstraction
- HDF5 is becoming a standard (even NetCDF is based on HDF5)
- HDF5 is a container for your specific data, not a standard data format

BUT: HDF5 files do not contain meta-information needed for visualization (semantics of datasets)

Solution: eXtensible Data Model and Format (XDMF)

XDMF

- Description of the meaning of HDF5 datasets, e.g. what data defines geometry and topology, what data defines attributes
- Small XML files (light data) in addition to (large) data files (heavy data, typically HDF5)
- Light data: XML file containing XDFM language statements and references to datasets in heavy data
- Heavy data: HDF5 files (or binary)
- http://www.xdmf.org

XDMF

How to generate XDMF files

1. Write XDMF files by hand (its just a text file)

2. Add commands to write an additional XDMF file in your existing I/O procedures

3. Use XDMF API to write both light and heavy data

XDMF Elements

Important XDMF elements are:

- **Grid**: defines a single grid, an array or tree of grids, or a sub set of another grid
- **Topology**: defines the topology of structured and unstructured grids (definition of “cells”)
- **Geometry**: defines coordinates of node points
- **Attributes**: defines values associated with the grid, node and cell centered
- **Time**: defines time values for a set of grids
- **DataItem**: the actual data, typically references to HDF5 or binary data sets, but may also be directly defined in the XDFM File. Also functions are possible in a DataItem, e.g. to calculate the magnitude of a vector on the fly.

XDMF: Example for Structured Grid

```
<?xml version ="1.0" ?>
<!DOCTYPE xdmf SYSTEM "Xdmf.dtd" []>
<Xdmf Version="2.0">
<Domain>
<Grid Name="sphere" GridType="Uniform">
<Topology TopologyType="3DCoRectMesh" NumberOfElements="2048 2048 2048"/>
<Geometry GeometryType="ORIGIN_DXDYDZ">
    <DataItem Dimensions="3" NumberType="Float" Precision="4" Format="XML">
          -1024.000000 -1024.000000 -1024.000000
    </DataItem>
    <DataItem Dimensions="3" NumberType="Float" Precision="4" Format="XML">
          1.000000 1.000000 1.000000
    </DataItem>
</Geometry>
<Attribute Name="distance" AttributeType="Scalar" Center="Node">
    <DataItem Dimensions="2048 2048 2048" NumberType="Float" Precision="4"
    Format="HDF">
          /viswork/hdf5Test/test_32768.h5:/sphere
    </DataItem>
</Attribute>
</Grid>
</Domain>
</Xdmf>
```

XDMF: Example for Unstructured Grid (1/2)

```
<?xml version ="1.0" ?>
<!DOCTYPE xdmf SYSTEM "Xdmf.dtd" []>
<Xdmf Version="2.0">
<Domain>
<Grid Name="flow" GridType="Uniform">
<Topology TopologyType="Tetrahedron" NumberOfElements="13340987">
  <DataItem Format="HDF" Dimensions="13340987 4">
    out.h5:/flow_Cells
  </DataItem>
</Topology>
<Geometry GeometryType="XYZ">
  <DataItem Format="HDF" NumberType="Float" Precision="8" Dimensions="2286605 3">
    out.h5:/flow_Points
  </DataItem>
</Geometry>
```

XDMF: Example for Unstructured Grid (2/2)

```
<Attribute Name="u" AttributeType="Scalar" Center="Node" >
  <DataItem Format="HDF" NumberType="Float" Precision="8" Dimensions="2286605">
    out.h5:/u
  </DataItem>
</Attribute>
<Attribute Name="v" AttributeType="Scalar" Center="Node" >
  <DataItem Format="HDF" NumberType="Float" Precision="8" Dimensions="2286605">
    out.h5:/v
  </DataItem>
</Attribute>
<Attribute Name="w" AttributeType="Scalar" Center="Node" >
  <DataItem Format="HDF" NumberType="Float" Precision="8" Dimensions="2286605">
    out.h5:/w
  </DataItem>
</Attribute>
<Attribute Name="p" AttributeType="Scalar" Center="Node" >
  <DataItem Format="HDF" NumberType="Float" Precision="8" Dimensions="2286605">
    out.h5:/p
  </DataItem>
</Attribute>
</Grid>
</Domain>
</Xdmf>
```

Programmable Source

Read your data with a programmable source filter

See:
https://www.paraview.org/Wiki/Python_Programmable_Filter

Programmable filters are programmed in Python

Example later …

User Interface

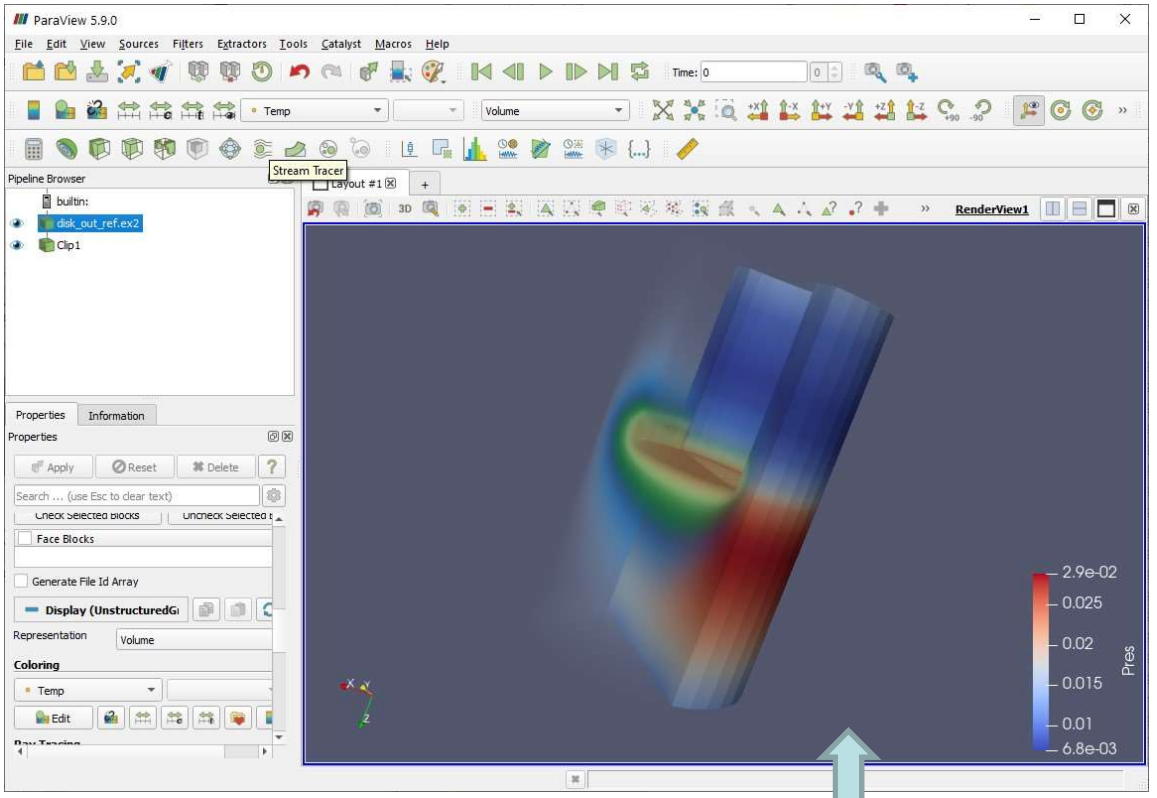
Menu Bar

Toolbars

Pipeline Browser

Properties & Information Panel

Toggle advanced properties:
Shows/hides additional control parameter

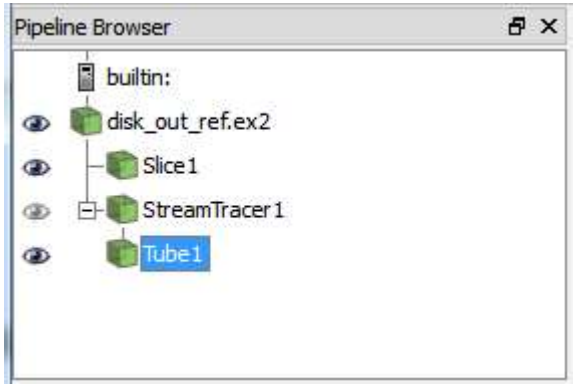


Display Area

Menu Bar: access the majority of features
Toolbars: quick access to the most commonly used tools

The Pipeline Browser

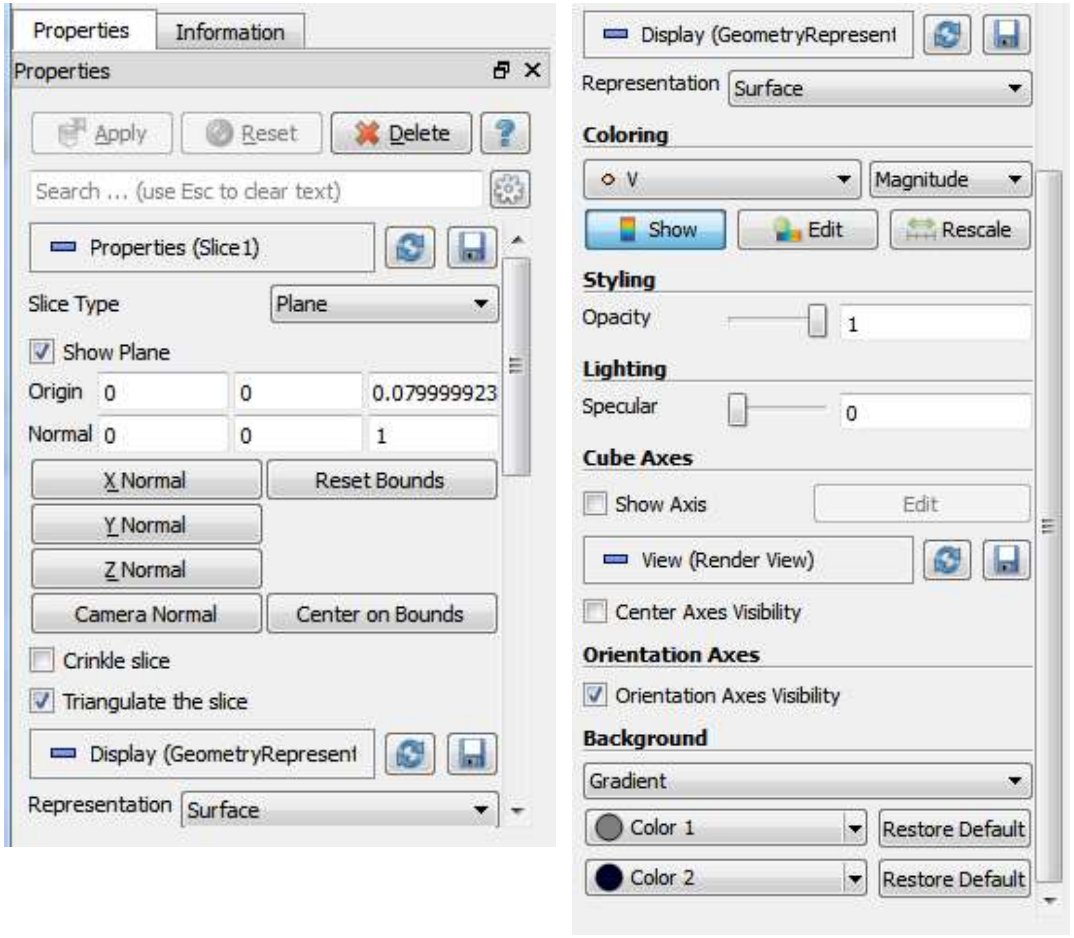
- shows data readers and filters
- shows current visualization pipeline structure
- allows you to
  - select objects within the current visualization pipeline
  - edit pipeline objects via the Properties Panel



- concept of **active objects**:
  - The selected object in the pipeline browser is the **active object**
  - All changes within the GUI will refer on what is actually active
  - The **active object** is used as the default object to use for operations like adding a filter

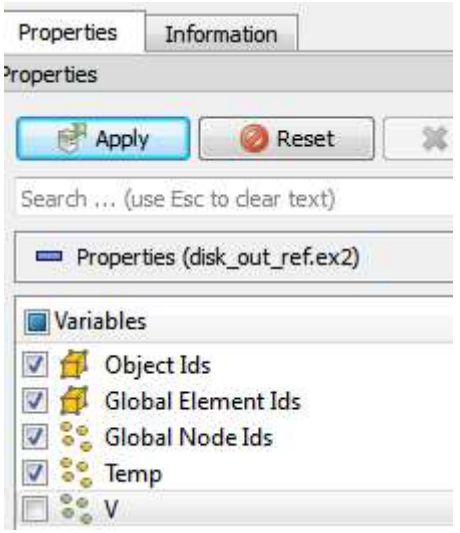
Properties Panel

- Depends on the active object in the pipeline
- Properties section: various parameters of the active object
- Display section: appearance of the active object (Representation, Color, Annotation, ...)
- View section: properties of the render view (Background, Orientation Axes, ...)



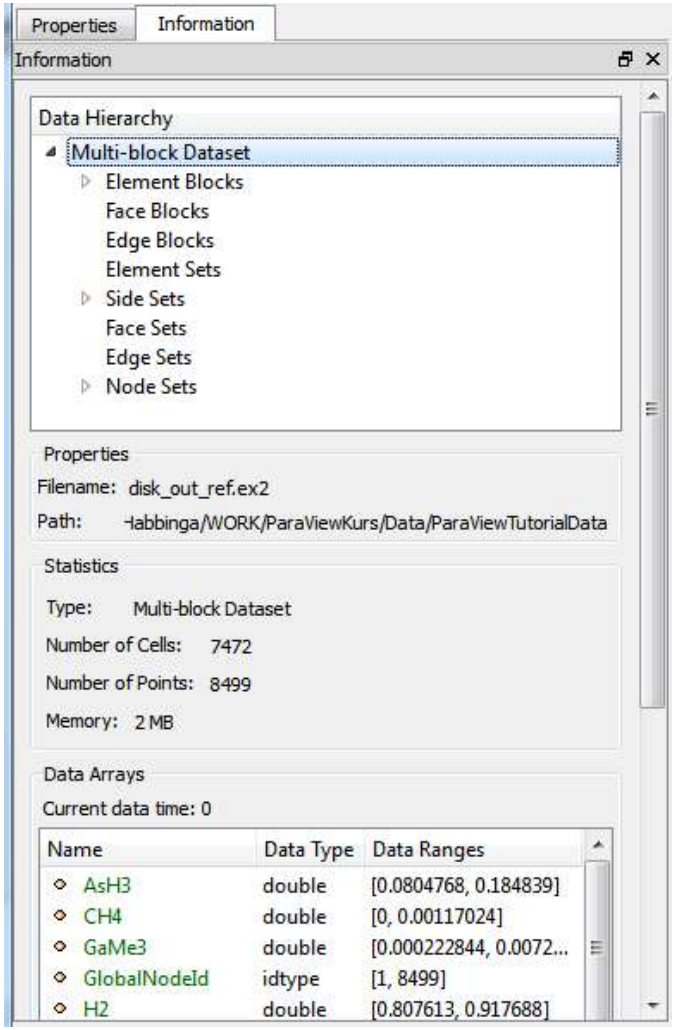
Apply/Reset

- changes to the object properties are not immediately applied
  - ➢ press Apply button to synchronize your view with the parameters
- Same applies to opening a file:
  - File -> open just reads the header information of the file
  - Hit the Apply button to load the data
- Hitting the Reset button will restore the options back to the last time they were applied.

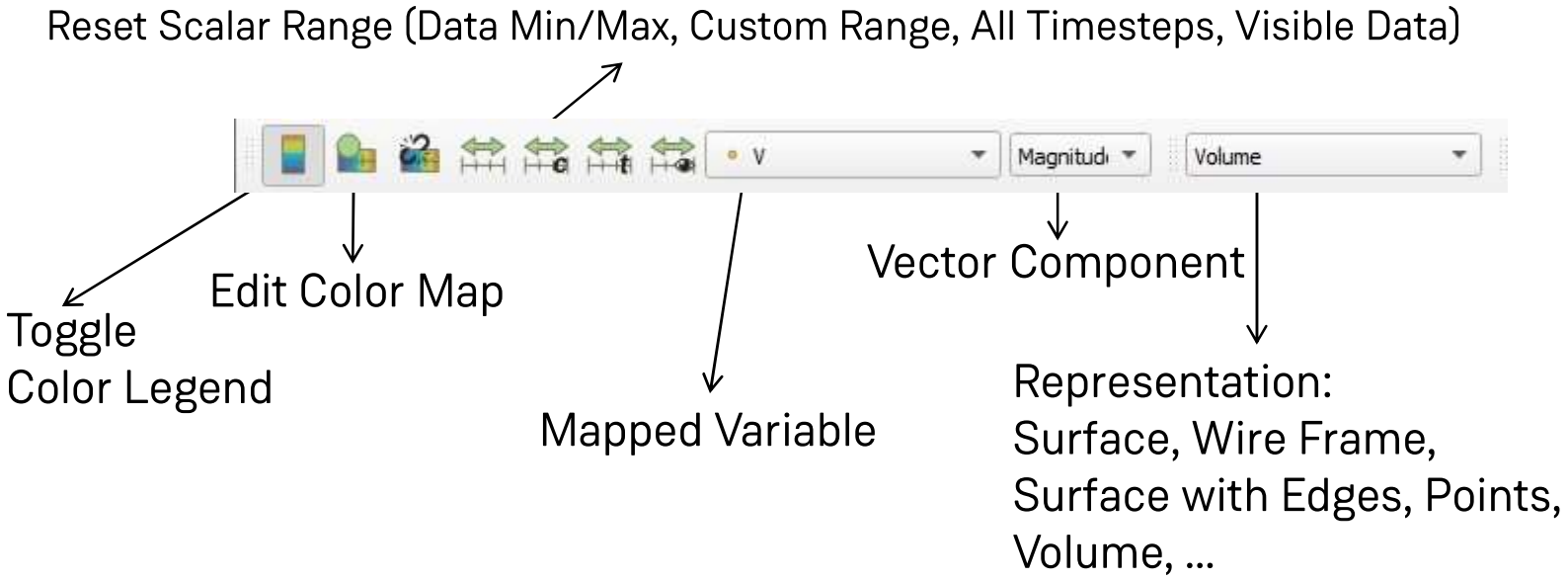


Information Tab:

- shows characteristics of the active object
  - Dataset Type
  - Number of Cells, Points
  - Data Arrays
  - ...



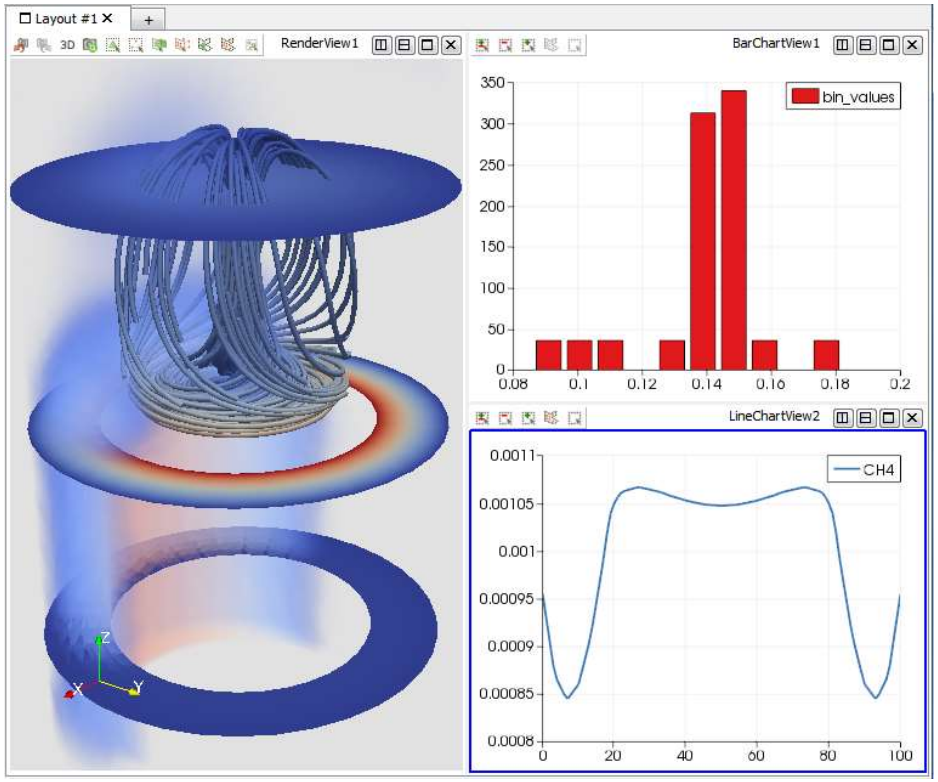
Active Variable Controls, Representation Toolbar



Can also be found in the display group of the properties panel!

Display Area

- probably most important part
- shows results of the current visualization pipeline
- various possible formats (2D and 3D views)
- Each display can be split vertically/horizontally



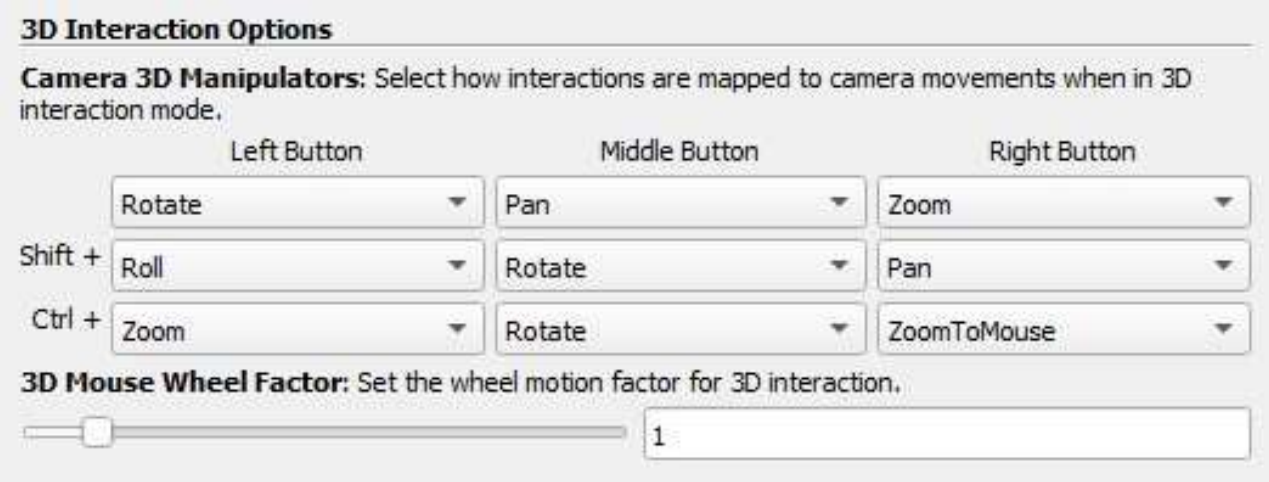
Getting Data into ParaView

There are several methods loading data in ParaView:

- Reading data
  - from a file (File – Open)
  - command line argument --data=datafile
- Generate data from SOURCES Menu
- load a previously saved state file (File menu, Load State)
  - ➢ this will return ParaView to its state at the time the file was saved (loading data, applying filters, setting preferences etc.)
- If you want to start a fresh visualization:
  - ➢ reset ParaView with the Reset Session button: 
  - ➢ Edit -> Reset Session

Controlling the camera

- Mouse interaction is mapped to camera movements as follows:



- Can be customized in the Edit -> Settings menu
- Camera Controls toolbar:

Reset camera (all visible objects) maintains view but repositions camera, that entire objects are visible

Zoom to data camera is placed to look at data (selected in pipeline browser)

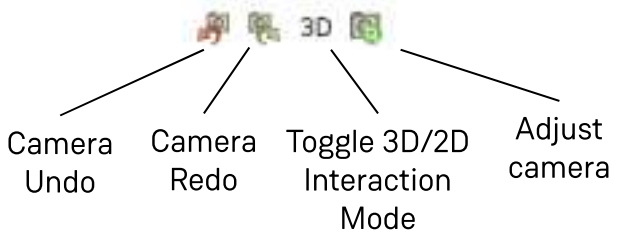
Zoom to box

Align view to coordinate axis

Rotate camera by 90 degree

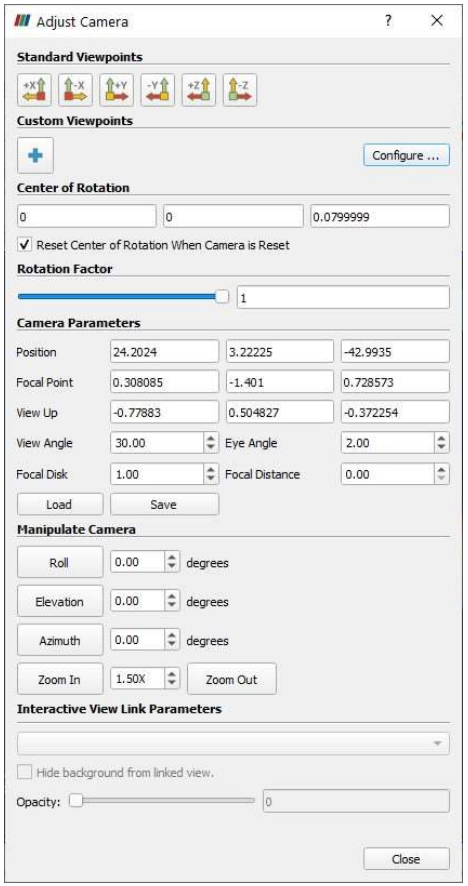
Controlling the camera (continued)

- Within the toolbar of each view:



- Adjust camera:
  - 4 custom (and more) views can be stored and exported
  - Camera parameters (position, view direction, view angle) can be set manually
- Hint: if you want to rotate the camera around its position, open Python shell and execute:

```
v = GetActiveView()
c = GetActiveCamera()
v.CenterOfRotation=c.GetPosition()
```



Controlling the camera (continued)

Show/hide orientation axes at bottom-left corner of the active view

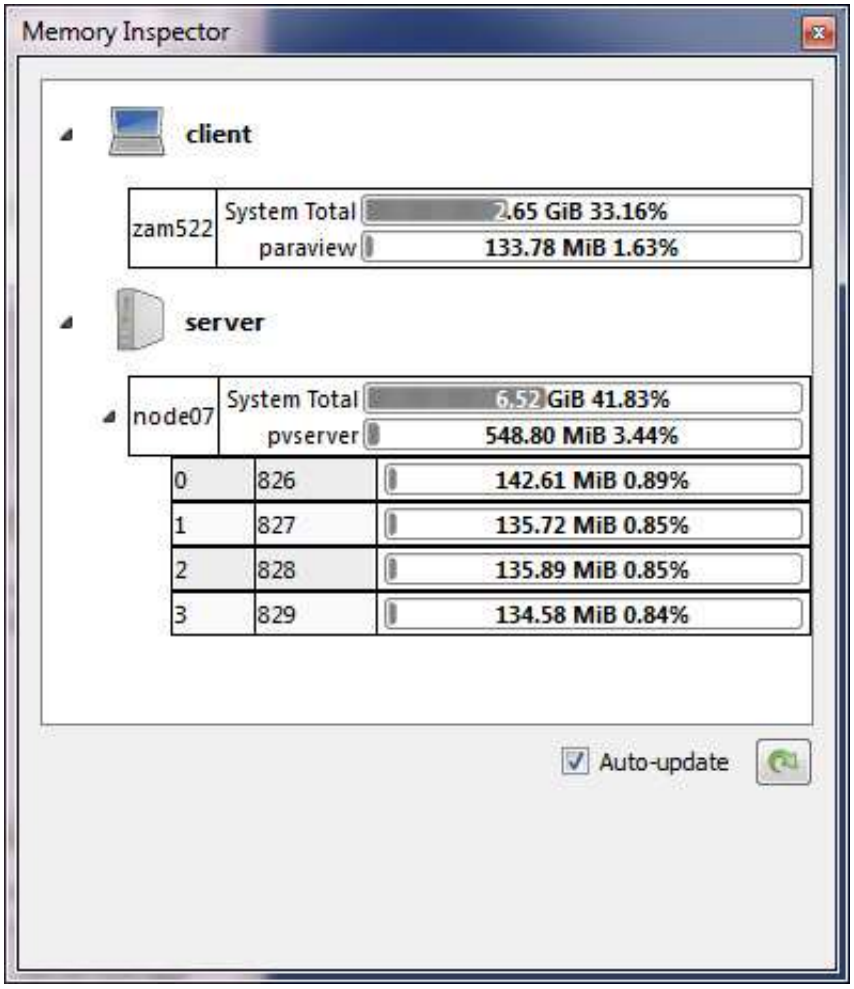
Show/hide axis drawn at the center of rotation

Set center of rotation to the center of the object

Allows to pick the center of rotation

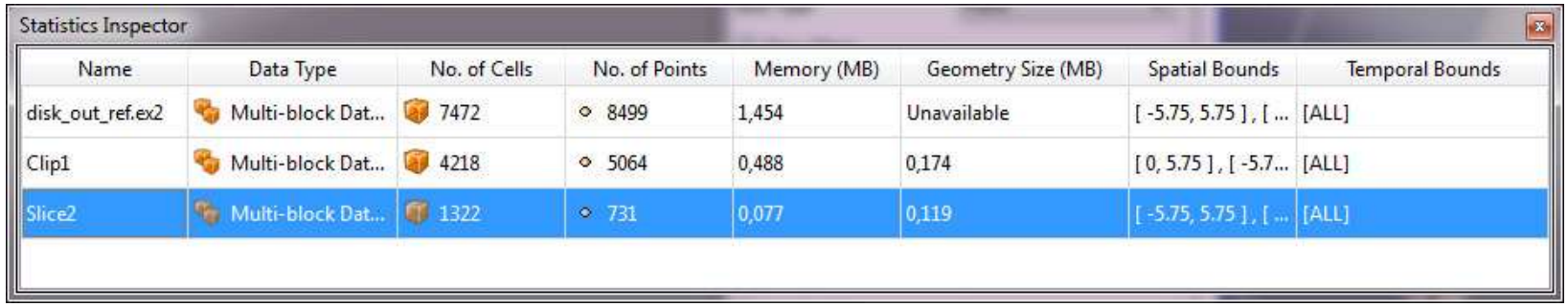
Memory Inspector

- Gives overview about the memory footprint
- In parallel mode, information about client and server is given



Statistics Inspector

- Shows statistical information about all loaded data and filters
  - Name
  - Data type
  - Number of cells and points
  - Memory consumption
  - Spatial and temporal bounds



Exercise I: Creating a Source

1. Select a Sphere from the Source Menu
   - Show it in a 3D view and increase it's resolution
   - Experiment with dragging different mouse buttons within the view (try also shift, ctrl)

2. (Delete the pipeline) Select a text from the Source Menu
   - Additionally take a 3D text from the Source Menu

Exercise II: Loading and viewing Data

1. Open Open **disk_out_ref.ex2**
   - Select all variables, press “Apply”
   - Select variable V for the coloring
   - Try the different representation styles, finally choose surface with edges
   - Turn the opacity value to 0.25
   - Save the actual state in a ParaView state file.
2. Experiment with the Center of Rotation:
   - Show/Hide it
   - Pick another Center (try some camera movements)
   - Reset the Center Of Rotation

Applying Filters

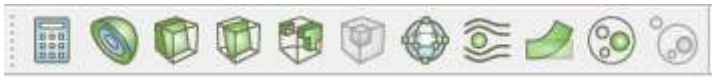
Achievements so far:

- Load the data, glean some information about it
- See the structure of the mesh, map some attribute data on its surface
  But! Many interesting features cannot be determined by just looking at the original data

➢ Discover much more about the data applying filters!

  - Extract or generate new features from the data
  - filters are attached to readers, sources, or other filters to modify its data in some way.

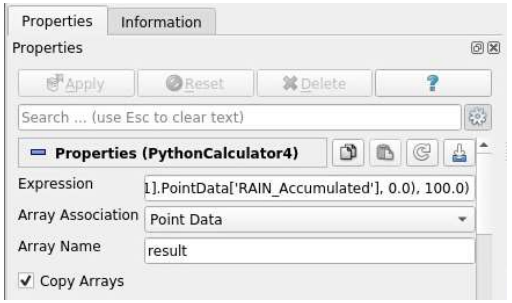
➢ Most common filters are available in the filters toolbar:



Calculations within ParaView

Calculator: calculates new attributes based on simple expression

- example: „LANDMASK*(abs(HGT) + 20.0)“
- Can generate vectors from scalars via „iHat*velocity_x + jHat*velocity_y + kHat*velocity_z“
- Can generate new coordinates
- Unflexibel, no „if“ statement



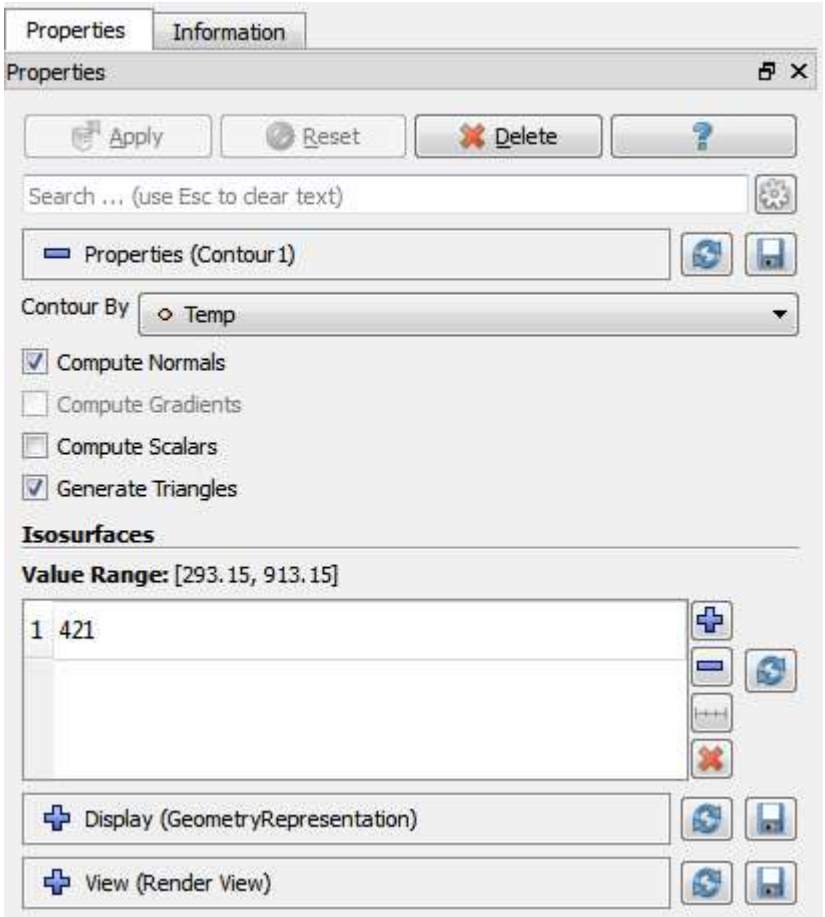
PythonCalculator: calculates new attributes based on simple Python expression

- NumPy and SciPy functions can be used
- Can generate vectors from scalars via „make_vector (velocity_x , velocity_y , velocity_z)”
- No „if“ statement, but numpy.where works, e.g. „numpy.where(Rain > 20, -1 * Rain, LANDMASK*(numpy.abs(HGT)+20))“

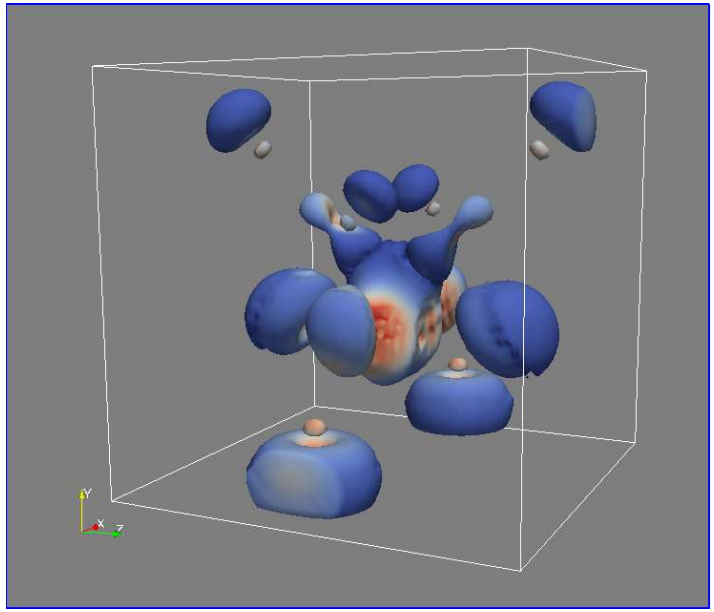
Programmable Source/Filter

- Most flexible
- Needs some deeper knowledge of ParaView conventions and data flow

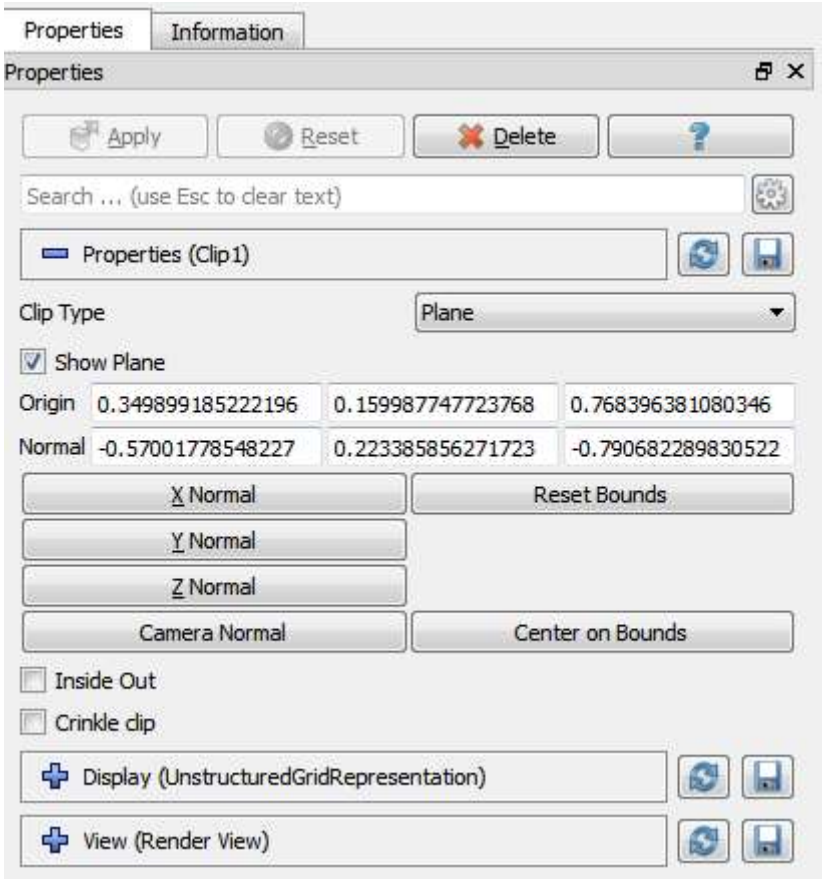
Common Filters: Contour



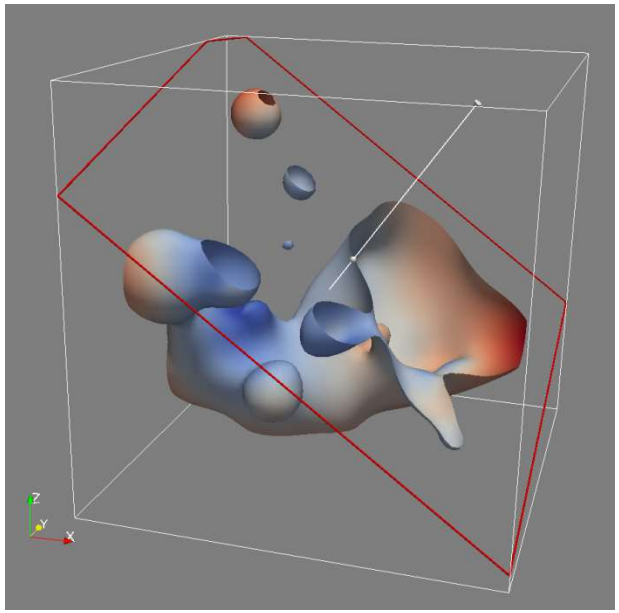
- Extracts the points, curves, or surfaces where a scalar field is equal to a user-defined value.
- This surface is often also called an isosurface



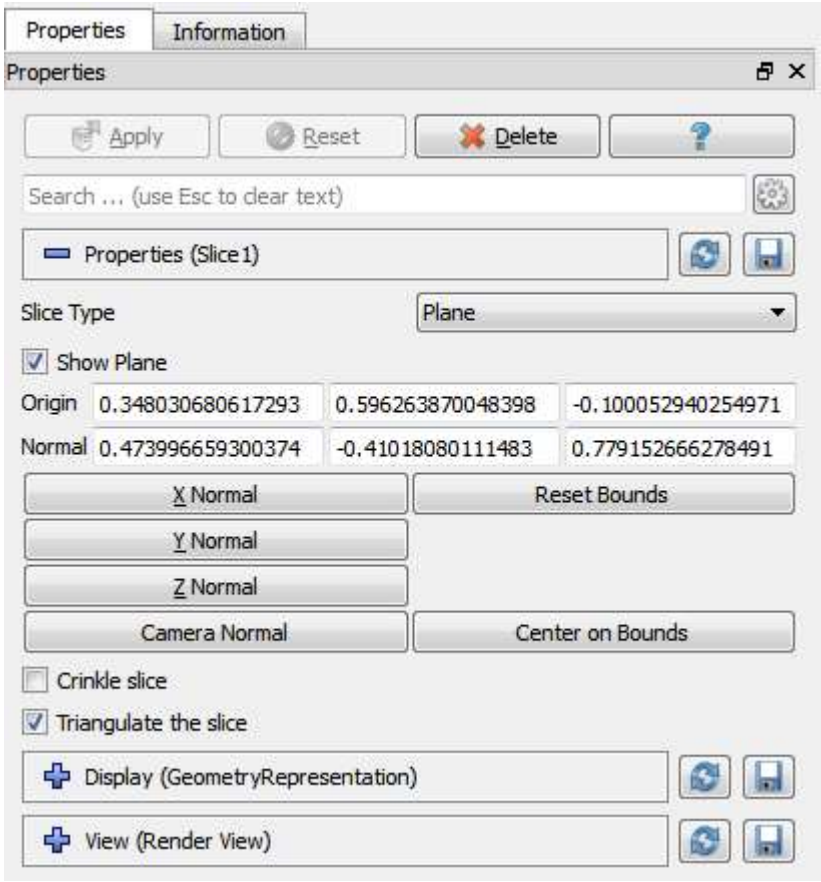
Common Filters: Clip



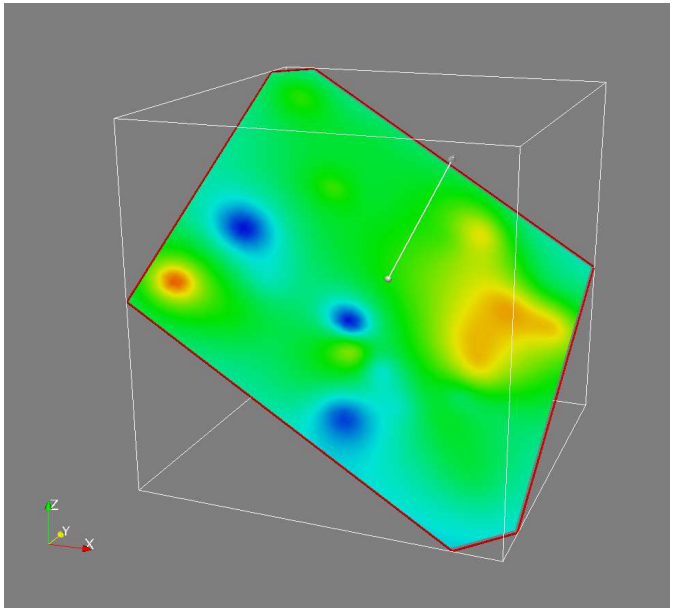
- Intersects the geometry with a user-defined plane, box or sphere
- Removes all the geometry on one side of this plane (box, sphere)



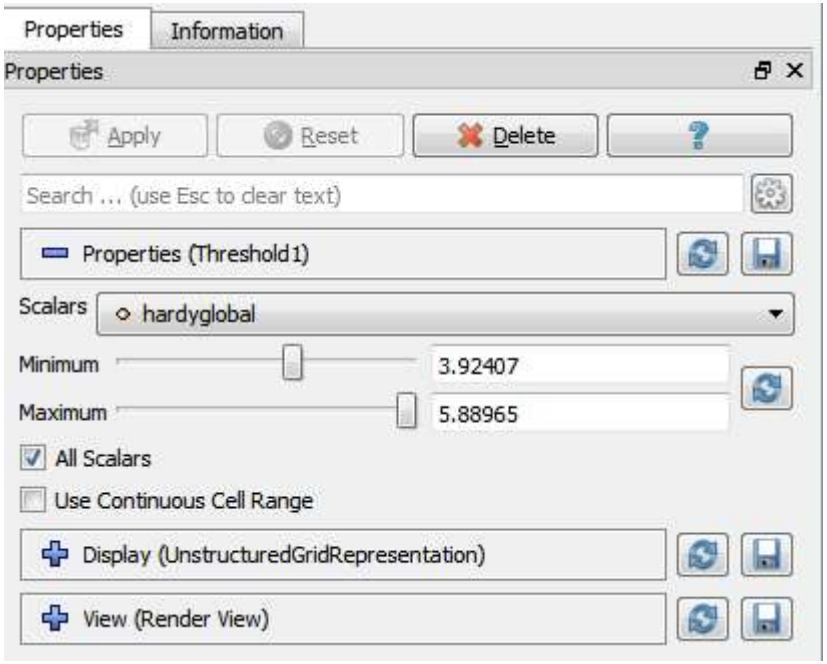
Common Filters: Slice



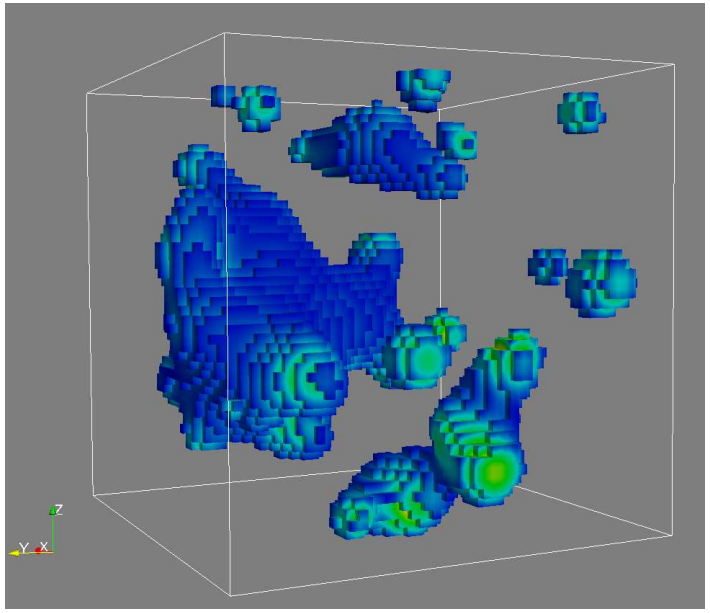
- Intersects the geometry with a plane, box, sphere or cylinder
- Similar to clipping, except that all that remains is the geometry where the plane is located.



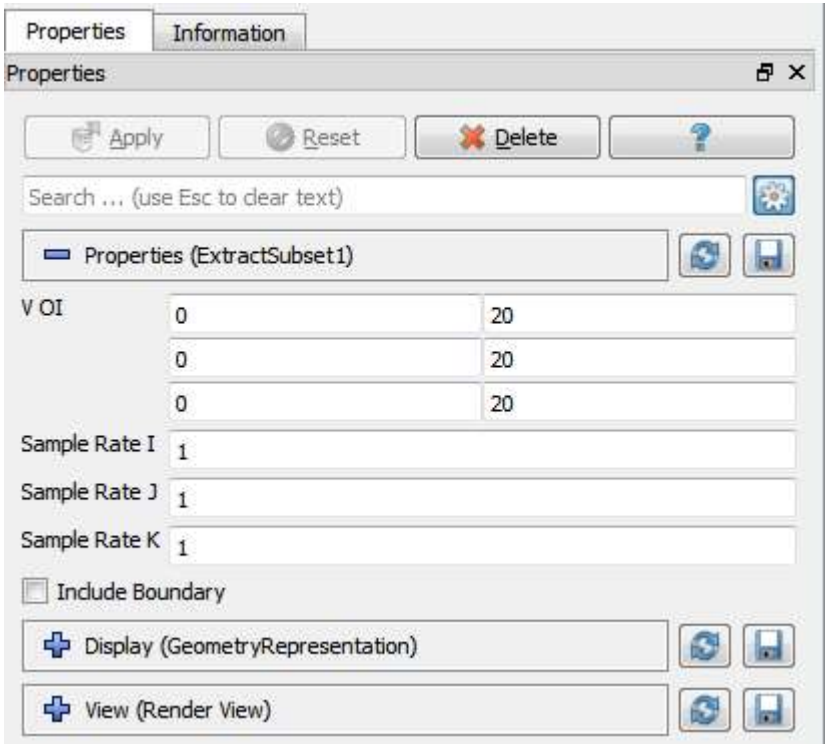
Common Filters: Threshold



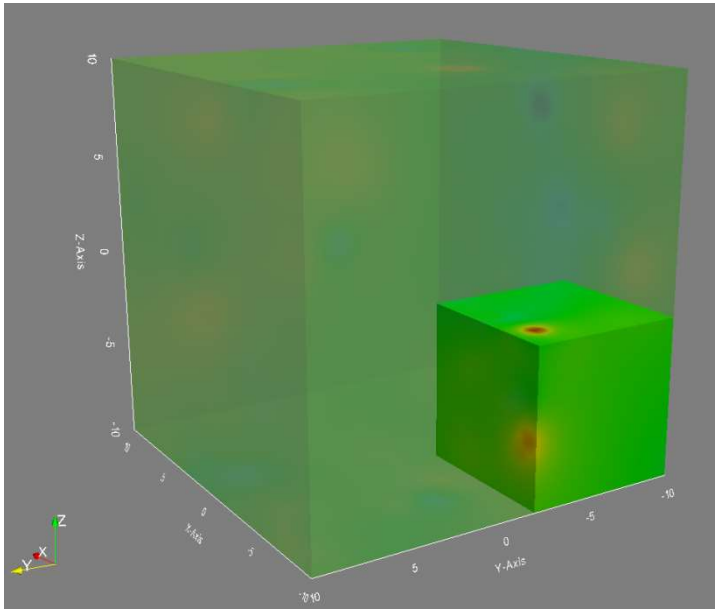
- Extracts cells that lie within a specified range of a scalar field



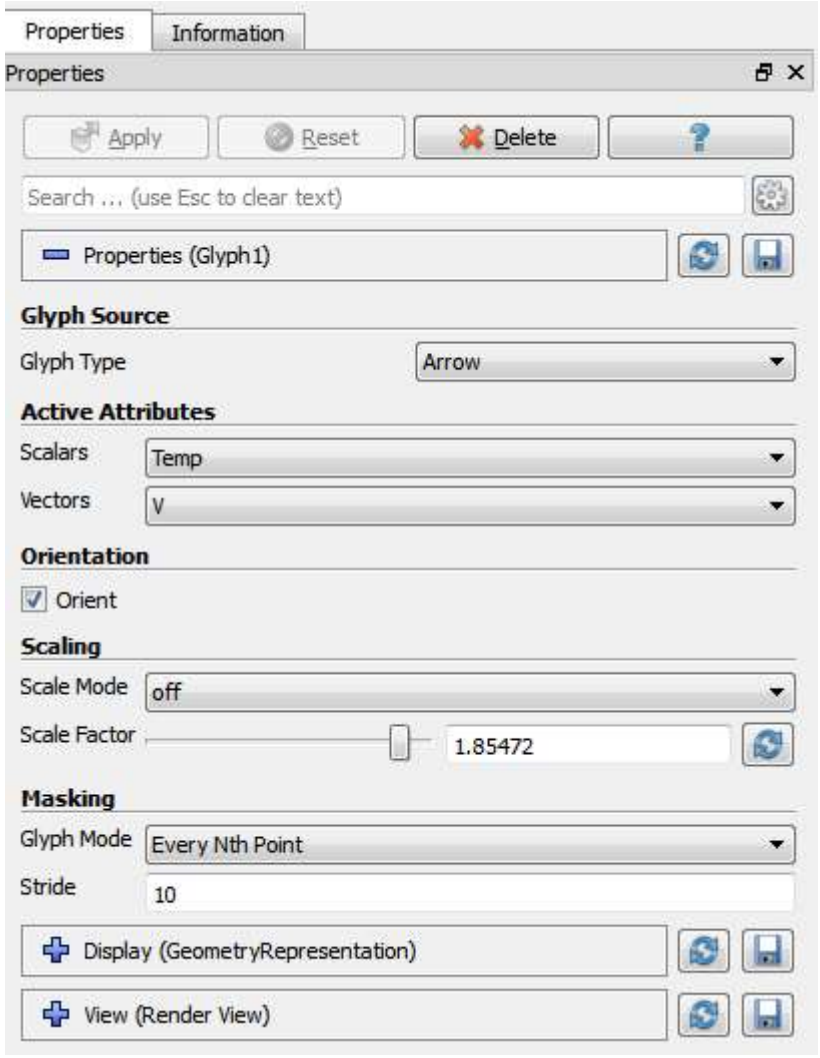
Common Filters: Extract Subset



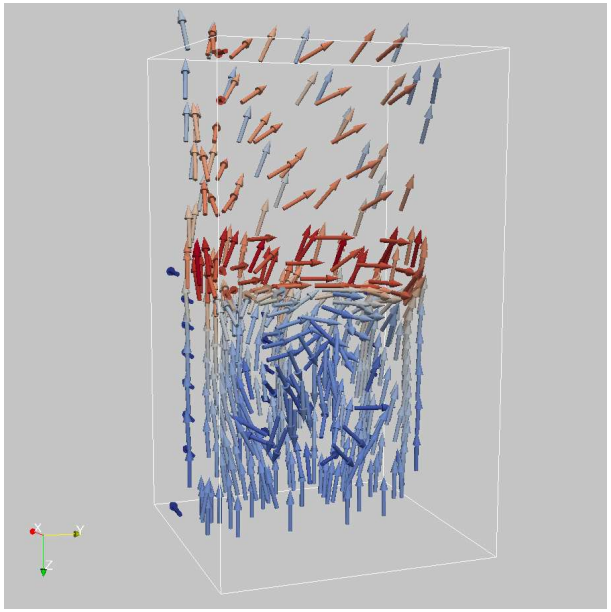
- Extracts a subset of a grid by defining a volume of interest and a sampling rate



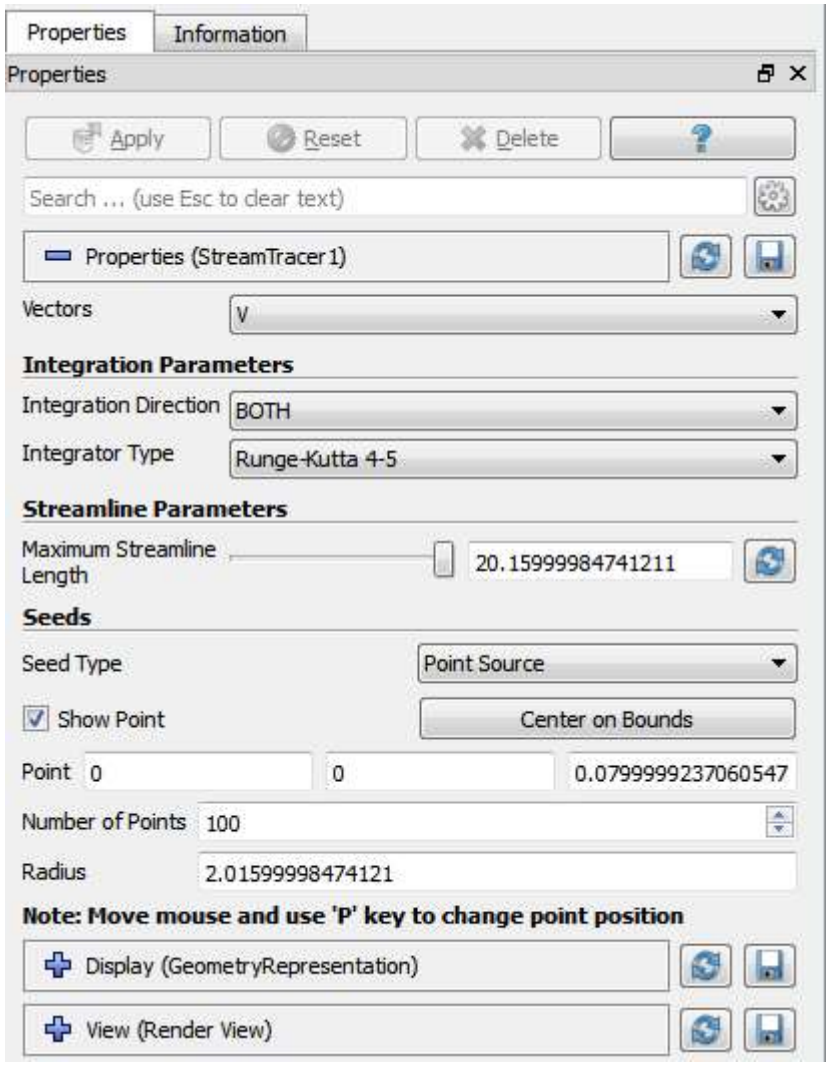
Common Filters: Glyph



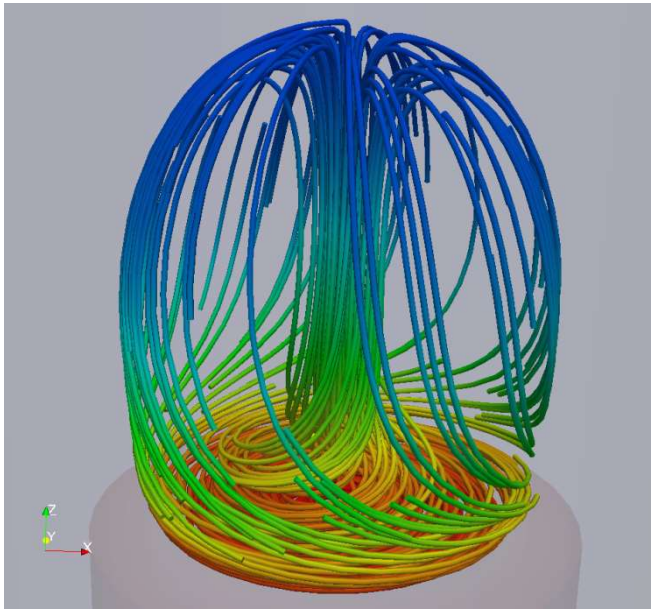
- Places a glyph, a simple shape, on each point (or subset) in a mesh
- glyphs may be oriented by a vector and scaled by a vector or scalar.



Common Filters: Stream Tracer

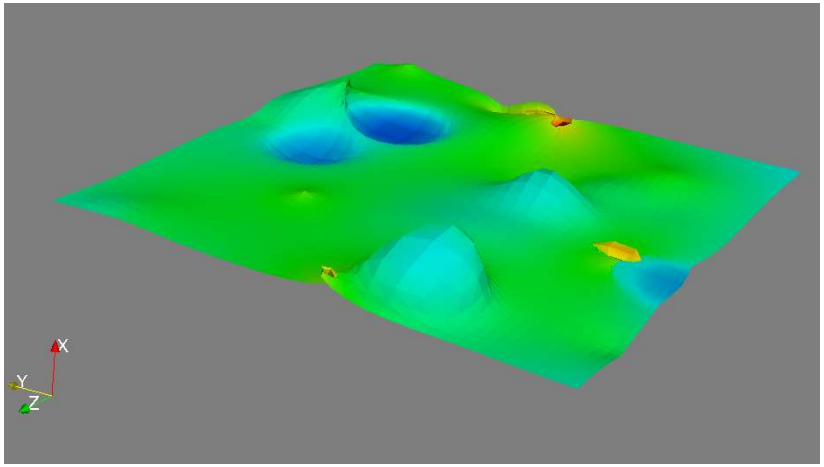


- Seeds a vector field with points and then traces those seed points through the (steady state) vector field.

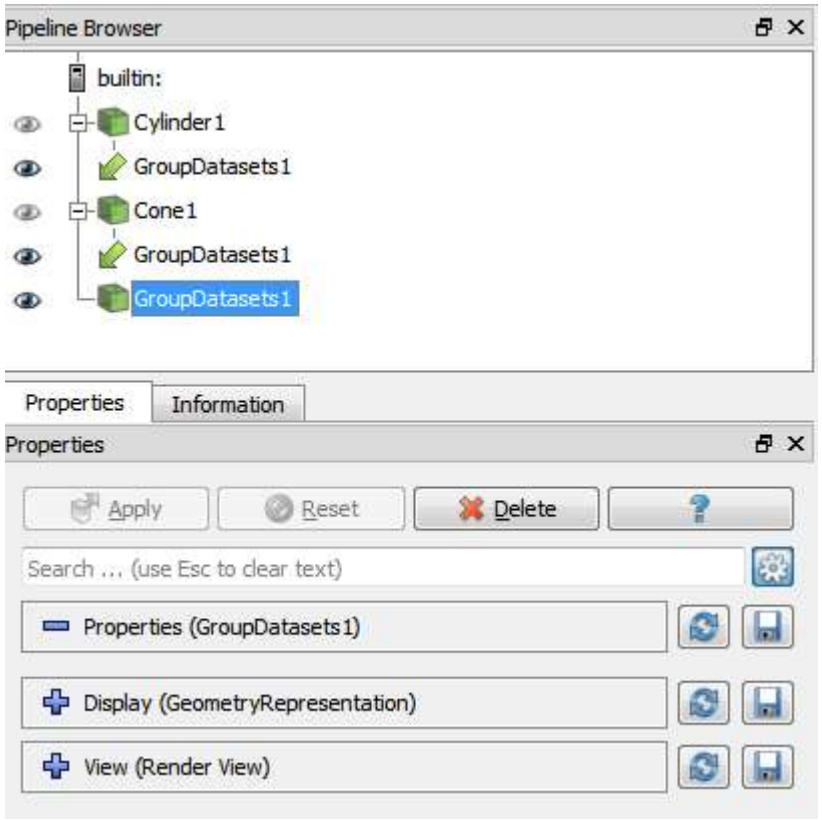


Common Filters: Warp (vector)

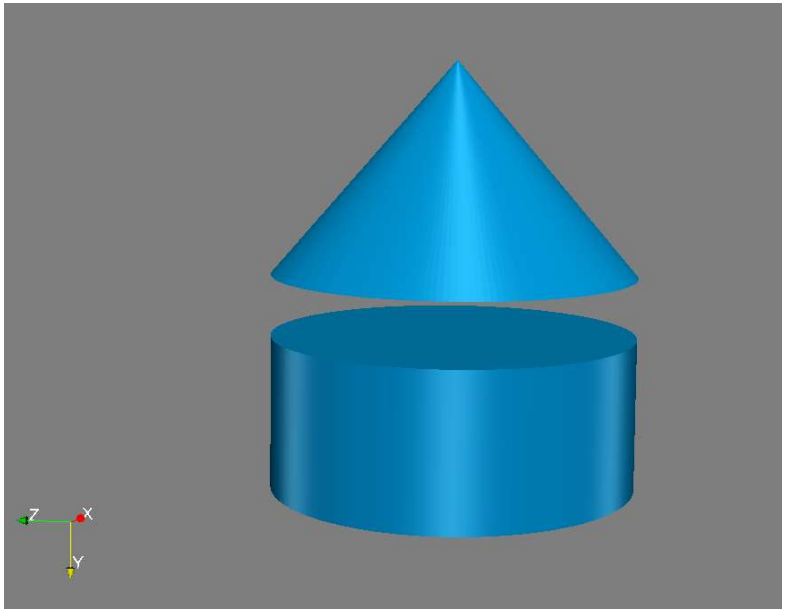
- Displaces each point in a mesh by a given vector field.



Common Filters: Group Datasets

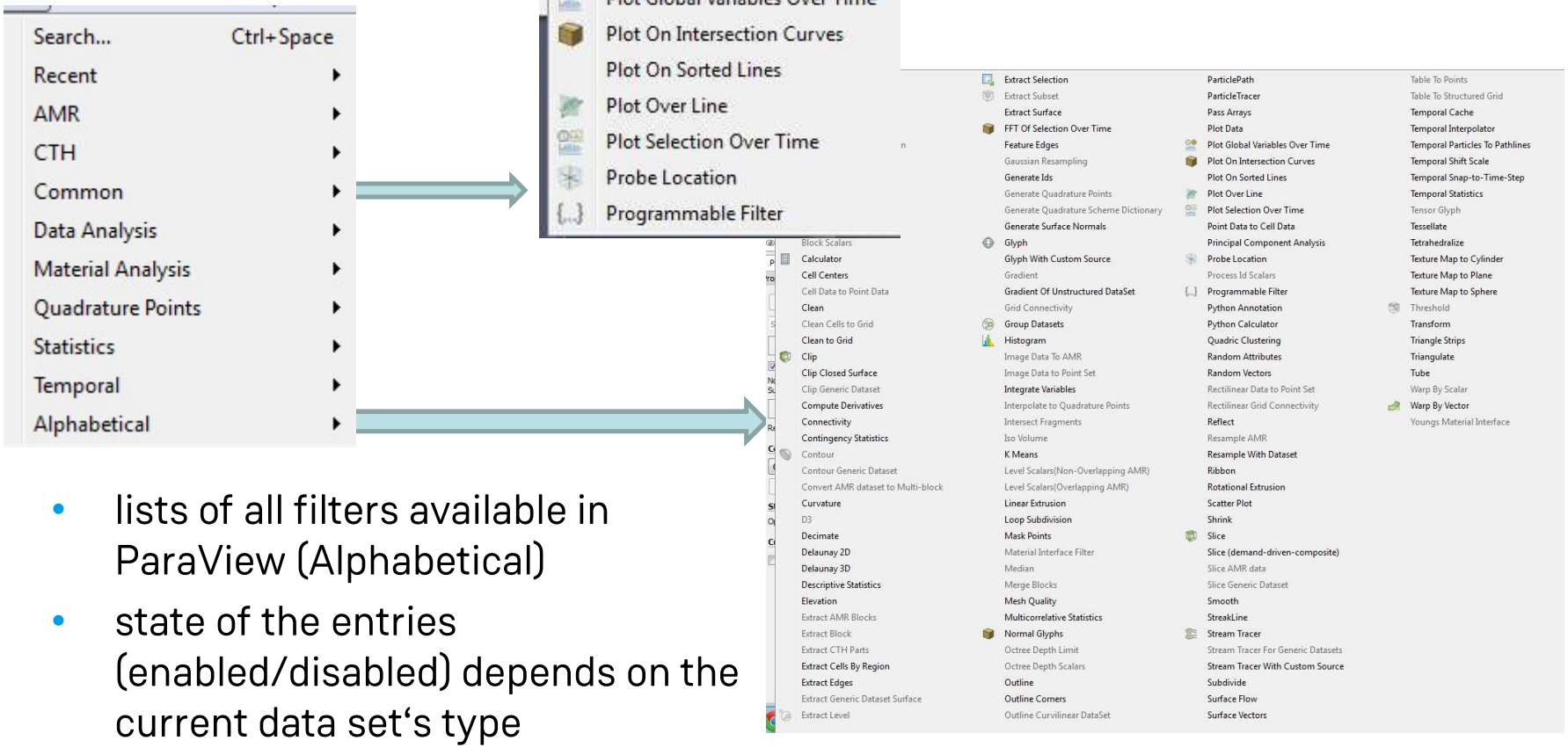


- Combines the output of several pipeline objects into a single multi block data set



Filter Menu:

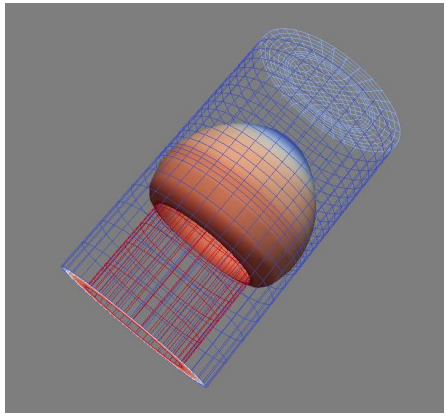
Many more filters in the Filters Menu



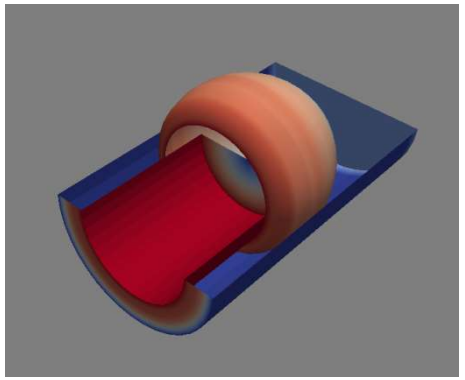
- lists of all filters available in ParaView (Alphabetical)
- state of the entries (enabled/disabled) depends on the current data set‘s type
- See https://www.paraview.org/Wiki/ParaView/Users_Guide/List_of_filters

Exercise III:
Applying filters, creating Visualization Pipelines

1. Applying Filters
   - Open the file: disk_out_ref.ex2
   - Add a Contour Filter, select variable Temp and set the isosurface value to 400
   - Use variable V to color data
   - View the data as a wireframe and the contour as an isosurface

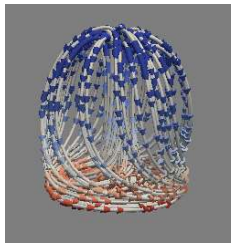


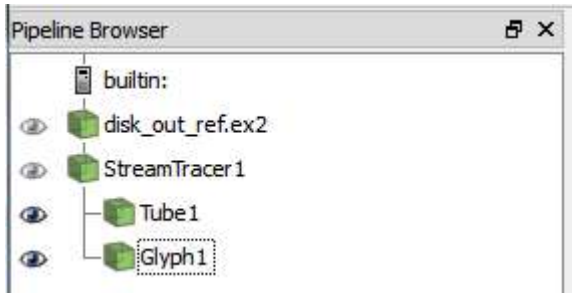
2. Creating Visualization Pipelines
   - Add (to the original data) an additional **Extract surface filter** (see: Filter - alphabetical)
   - Create a **Clip filter** to **Extract surface filter**
   - Make Contour and Clip visible



Exercise IV: Streamlines, Volume Rendering

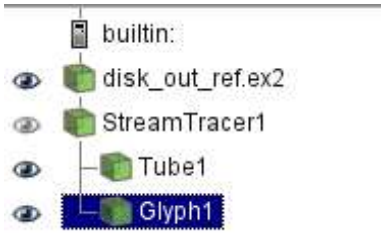
1. Vector Visualization: Streamlines
   - Open the file: disk_out_ref.ex2
   - Select a **Stream Tracer Filter,** Add a **tube filter**
   - Add a **glyph filter** to the **Stream Tracer Filter.**
     Set Vectors to V, show the glyphs as cones and color them with variable Temp
   - Play with the glyphs parameters until the result pleases you

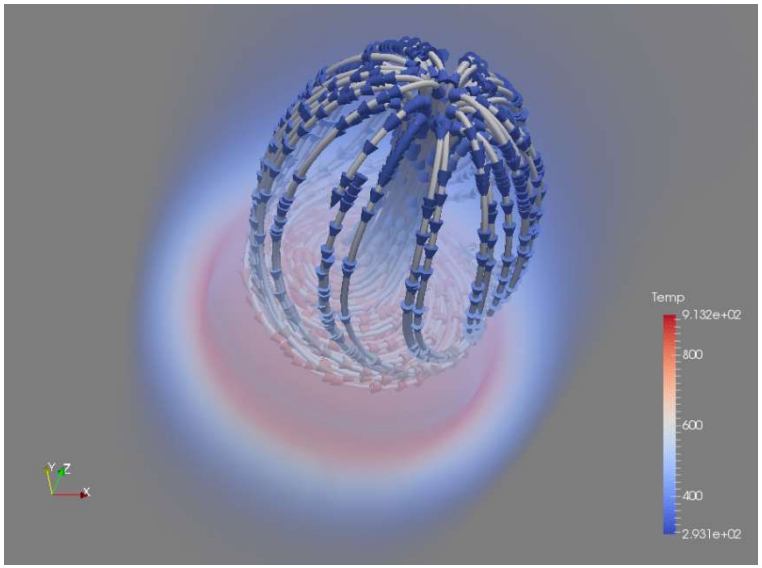




Exercise IV: Streamlines, Volume Rendering

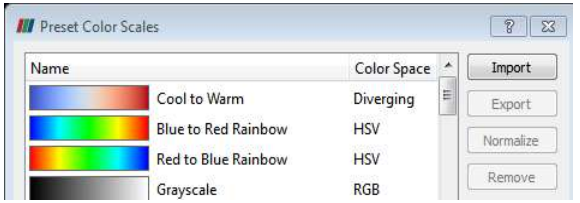
2. Volume Rendering
   - In previous example:
     Change the representation of the input data to **Volume**
   - make the input data visible,
   - Choose Temp for the coloring and make color legend visible





Changing the color map

- Via Color Map Editor (View->Color Map Editor, )
- ParaView provides a number of preset color scales 



- To add another color/value pair to the mapping function (double-)click into the gradient bar 
  (or add it manually into the table underneath)
- (Double-)click into the graph to modify the opacity transfer function.

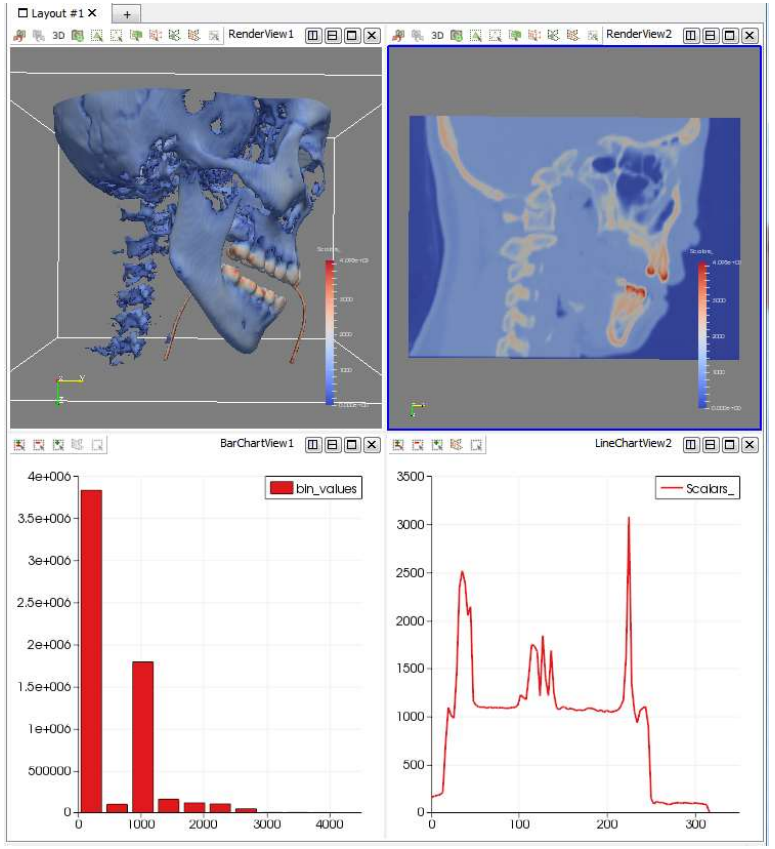
Multiple Views

Achievements so far:

- Coloring with one variable, extract iso surfaces with another one, streamlines …

But: still difficult to make correlations between the variables!

➢ Use **Multiple Views**!



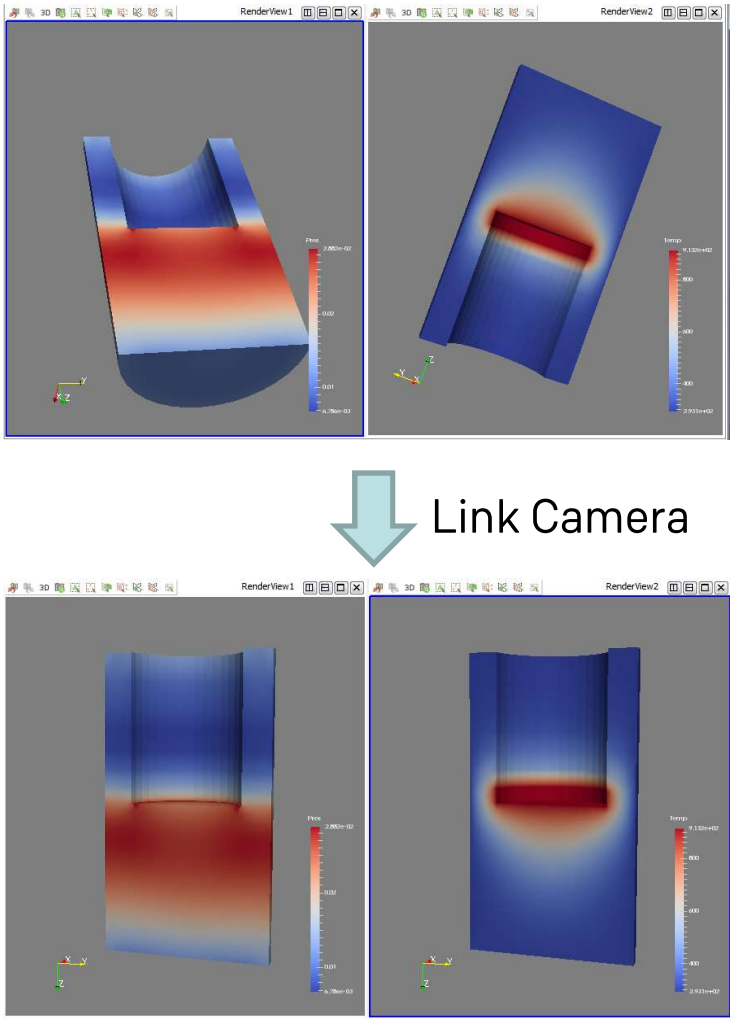
- Each view can show an independent aspect of the data
- 2D (e.g. histograms, line charts, spreadsheets …) and 3D views are possible

Multiple Views

- Multiview is not limited to only two views
- Each view has its own set of multiview buttons  to split horizontally/vertically or erase the view:
- The view with the blue border is the active view
  - ➢ The display and view parameters in the Properties Panel depend on the active view (and on the selected item in the Pipeline Browser).
- The views can be changed in size by clicking in the space between views
- Swap two views by clicking and holding on one of the view toolbars and dragging onto one of the other view toolbars

Multiple Views: Linking the Camera of RenderViews

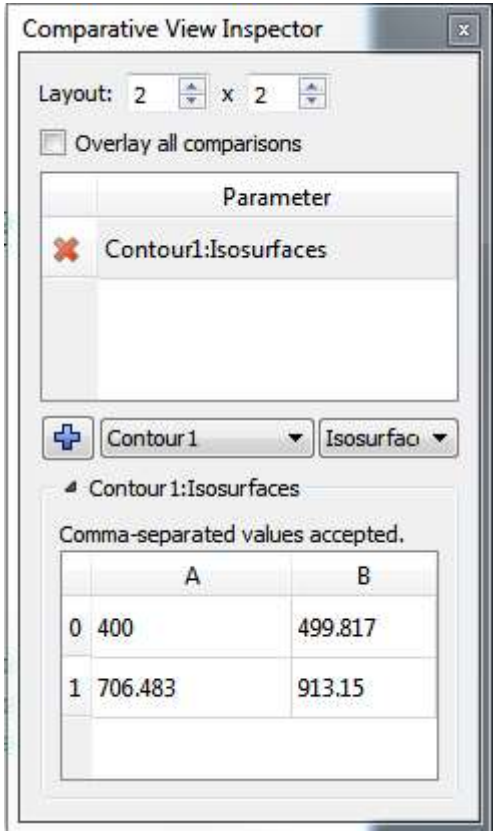
- Make Views comparative by adding Camera Links:
  - Right click on one of the views over the background -> Link Camera, then click in the view to link with
  - The two cameras are linked, both draw the view from the same viewpoint



Comparative View

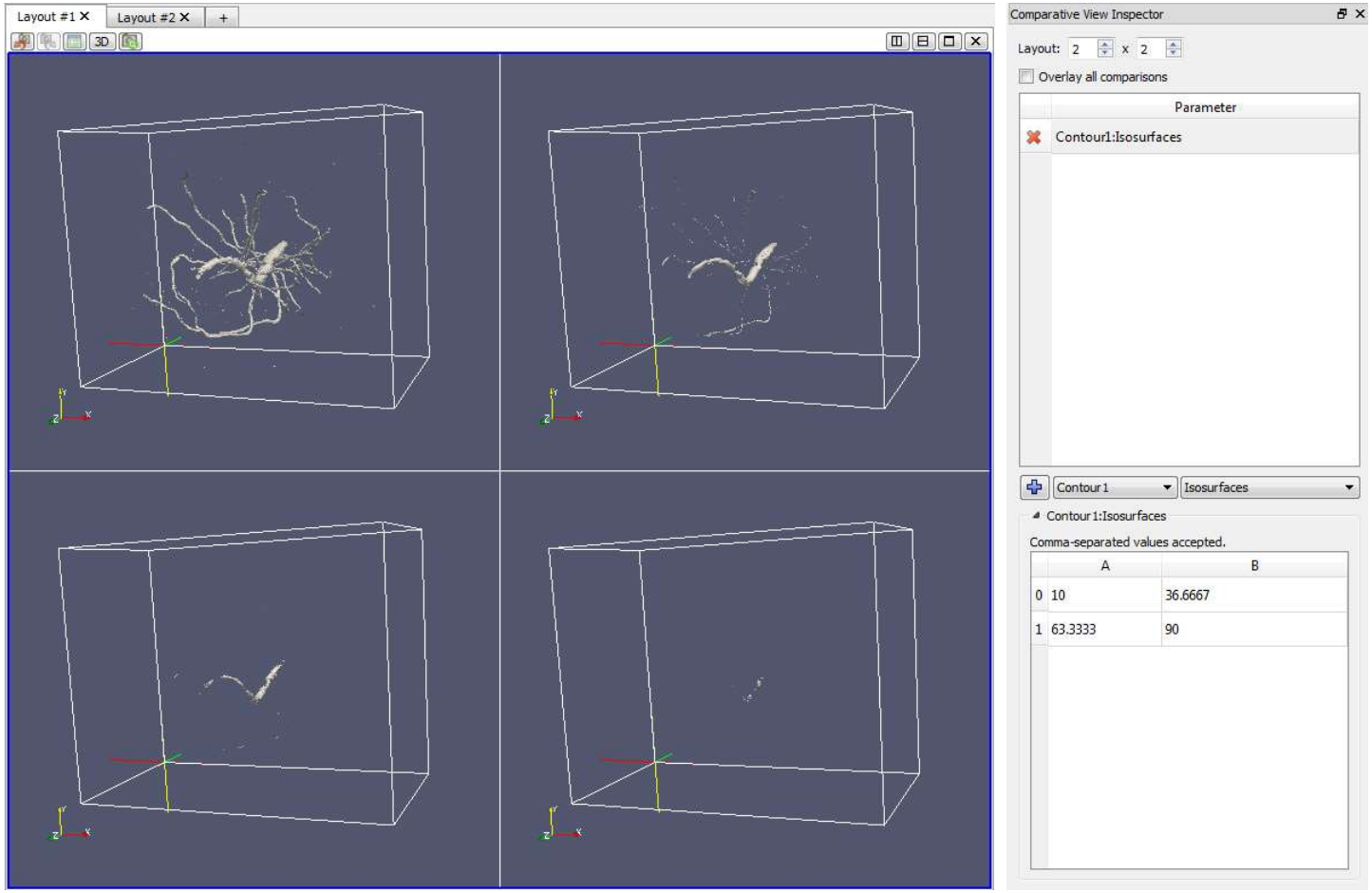
Visualize several representations of the same scene, but with different parameters to make parameter studies

- Open a new view and select Render View (Comparative)
- Enable the comparative view inspector in the View Menu
- Choose a layout (e.g. 2x2)
- Select the parameter(s) you want to vary (upper panel)
- Adjust the parameter for the views (lower panel)
  - Either click on one cell in the parameter table, OR
  - Select a range of cells and enter a parameter range



Comparative View

Example: Visualization of a contour with 4 different isovalues



Plotting

- mechanism to drill down into your data to allow quantitative analysis.
- Plots are usually created with filters (Filters->DataAnalysis)
- also: data analysis toolbar containing the most common data analysis filters, some of which are used to generate plots.

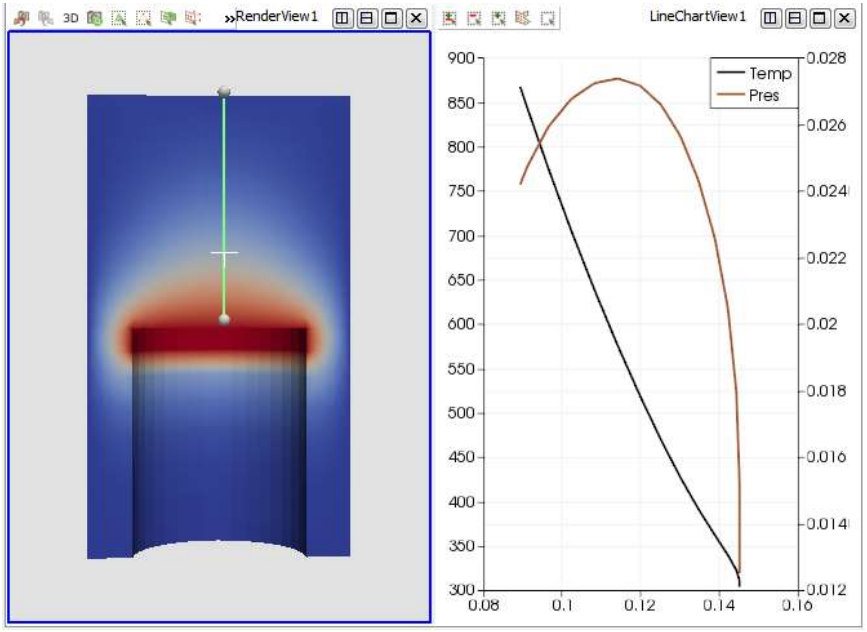
 *Plot Global Variables Over Time*

 *Plot Over Line*

 *Plot Selection Over Time*

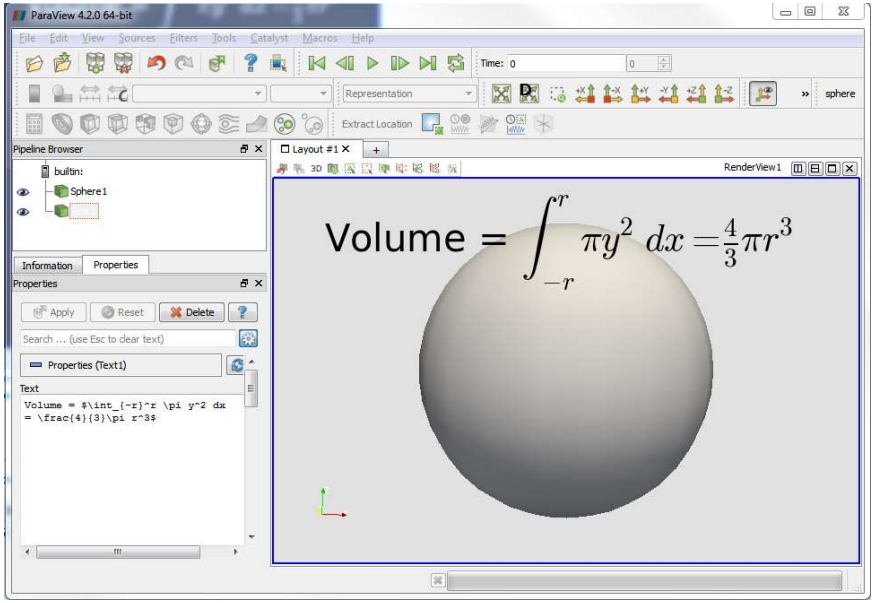
Plotting Example: Plot over a line in space

- Hit Plot Over Line from the Data Analysis Toolbar
- Place a Line Widget in the 3D space, hit Apply
- Choose the variables you would like to see in the Line Chart View (in the Properties Panel, while Line Chart View and the Plot Over Line filter are active!)
- Line plot allows for a different scale on the left and right axis -> scale each variable individually on each axis.
- save the plot as a screenshot (File-> Save Screenshot) or export it to a vector graphics that scales properly (File->Export Scene).



Text Annotation

- Use Text source to place some text in the 3D view
- Drag it with the mouse to the position you like
- Use the Display section of the property panel to change the appearance of the Text (font, size, position, color)
- ParaView supports MathText markup for mathematical expression rendering



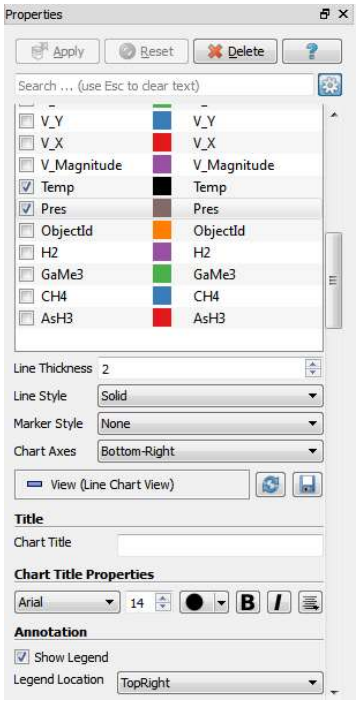
(See http://matplotlib.org/users/mathtext.html for details)

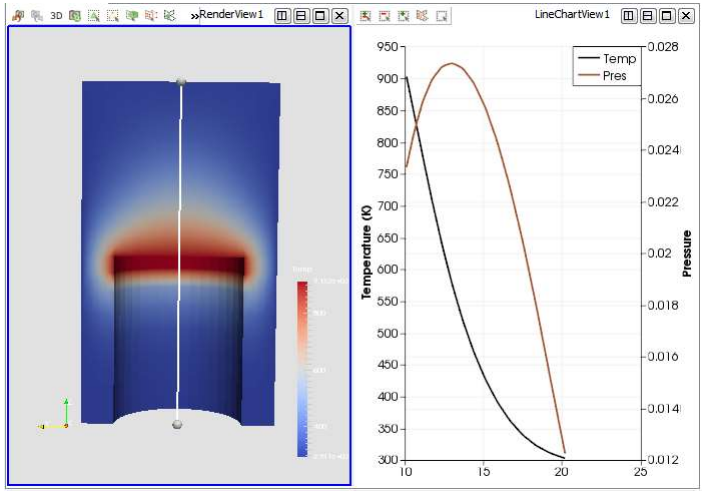
Exercise V: Plotting

1. Creating a Plot Over Line:
   - Open disk_out_ref.ex2, load all variables
   - Add a Clip filter, uncheck the Show Plane checkbox in the Properties Panel, click apply
   - Click on disc_out_ref.ex2 to make it the active object and select the plot over line filter.
   - Try to position the line segment in 3D space
     - try grabbing and moving the handles, the line itself, different mouse buttons, press “p” for picking a point under the curser)
   - Finally, hit the button “Z Axis” in the Properties panel, hit apply

Exercise V: Plotting (continued)

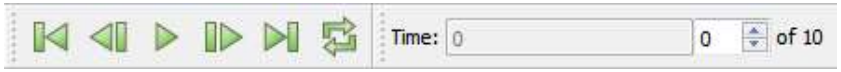
2. Changing the display options
   - Make the plot view active and deselect all variables except for Temp and Pres
   - Select Pres Variable in the Display options of the Properties panel
   - Change the Chart Axis to Bottom-Right
   - Add annotations to the left and right Axis (Temperature, Pressure)





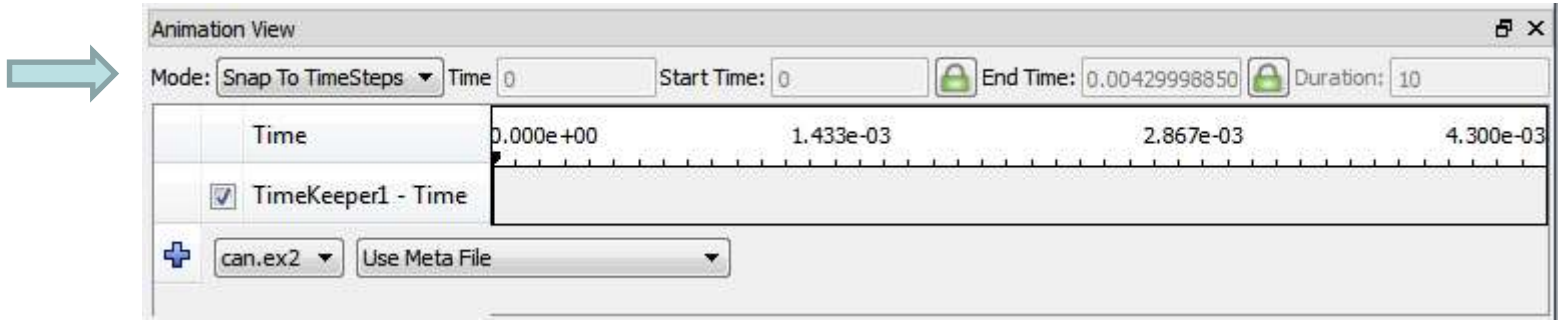
Time in ParaView

- ParaView supports visualization of temporal data providing
  - Readers for file series and datasets containing several time steps
  - Temporal filters, e.g.
    - Annotate Time
    - Temporal Interpolator (good for mesh deformations, topology has to remain consistent!)
    - Particle Tracer
- Toolbars: VCR Controls, Current Time Controls:



- View -> Animation View: see next Slide

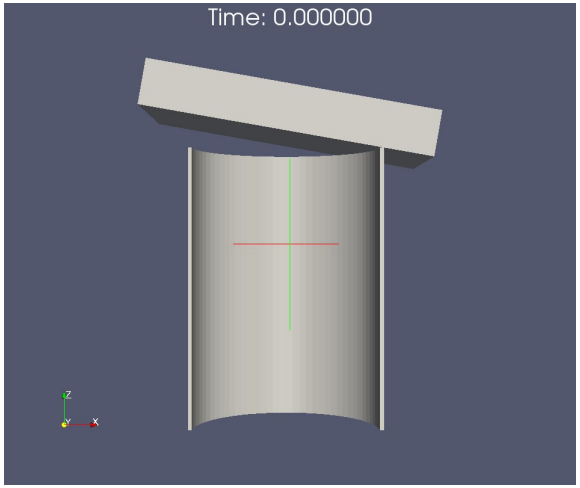
Controlling Time: Animation View

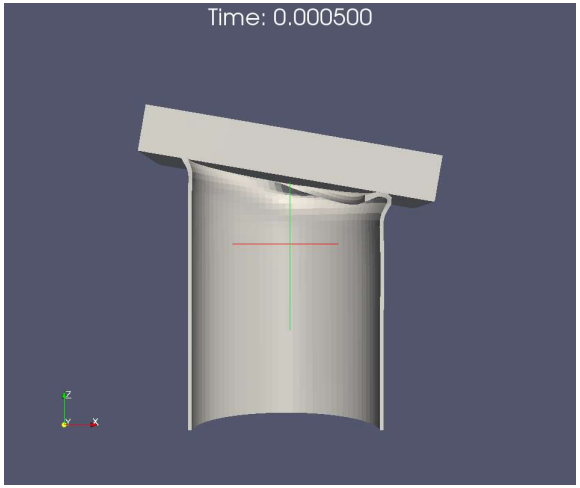


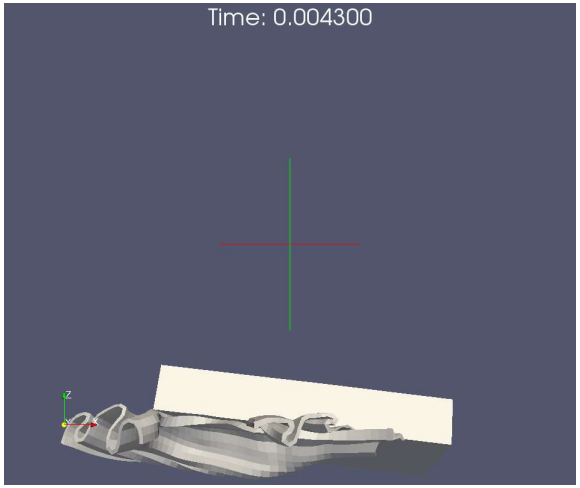
- **Animation Mode**: determines how ParaView steps through time during playback
  - **Snap To TimeSteps**: plays exactly those steps defined by your data
  - **Sequence**: Specified number of frames equally spaced between start and end time
  - **Real Time**: animation lasts approx. the specified number of seconds
  - Default: Snap to TimeSteps

Exercise VI: Time Dependent Data

- Open the file: can.ex2
- Set the view direction to +y
- Add an Annotate Time filter to observe time changes
- Animate the mesh

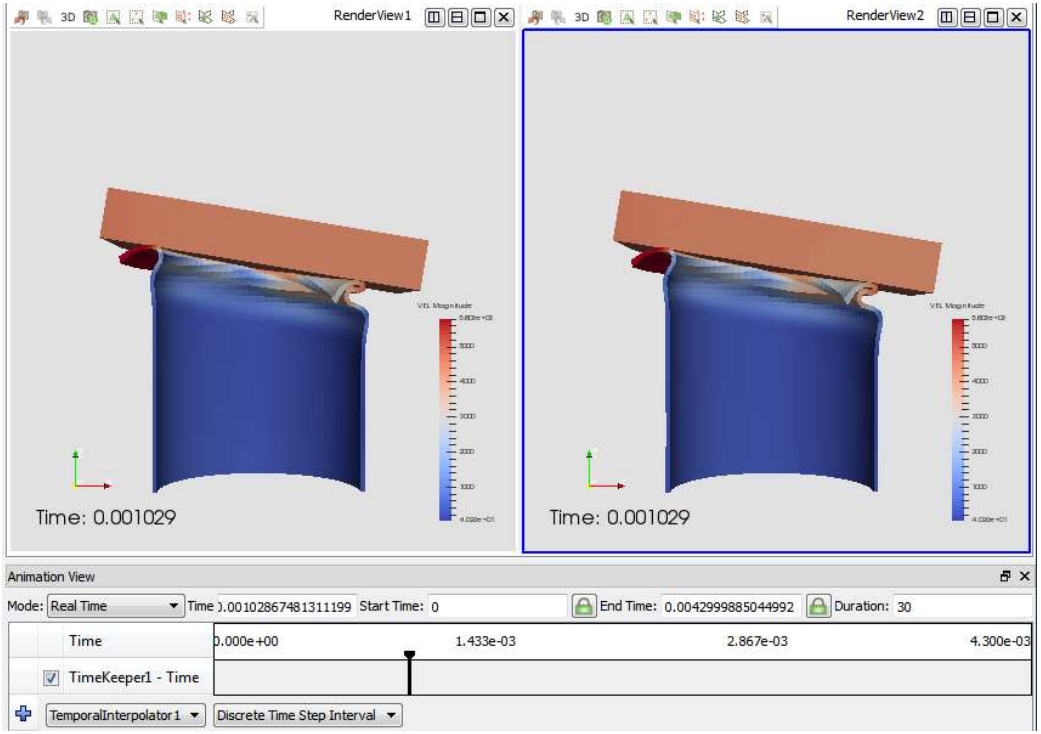






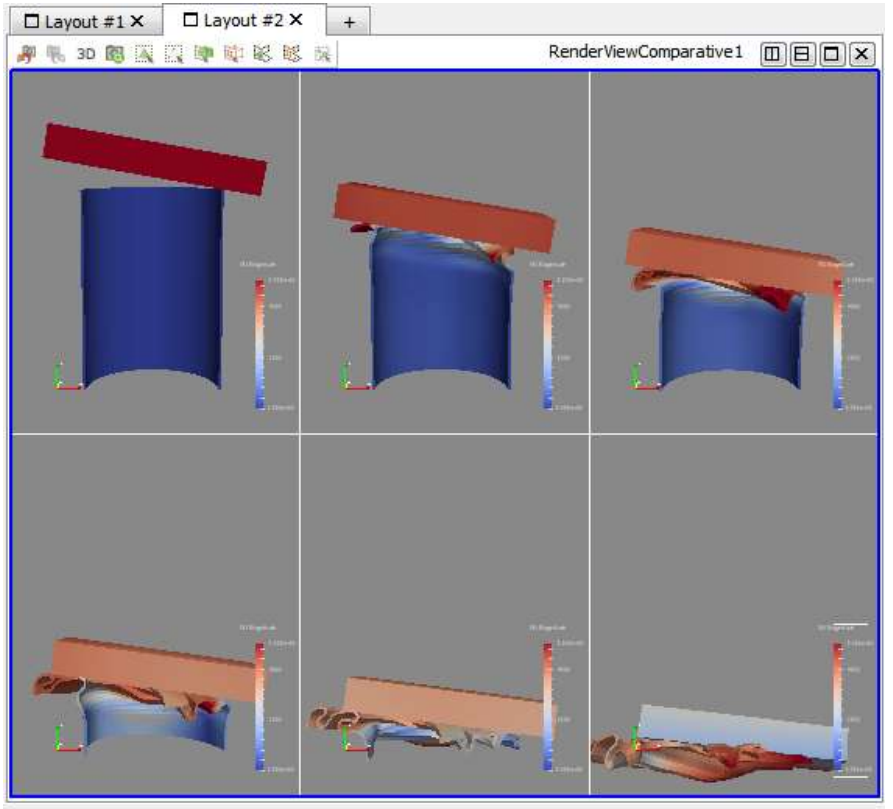
Exercise VI: Time Dependent Data (cont.)

- Split the view horizontally and apply the **Temporal Interpolator Filter** to the data.
- Select **Real Time** as animation mode, set duration to 30 seconds
- Link the cameras
- Hit **Play**



Exercise VII: Time Dependent Data, Comparative View

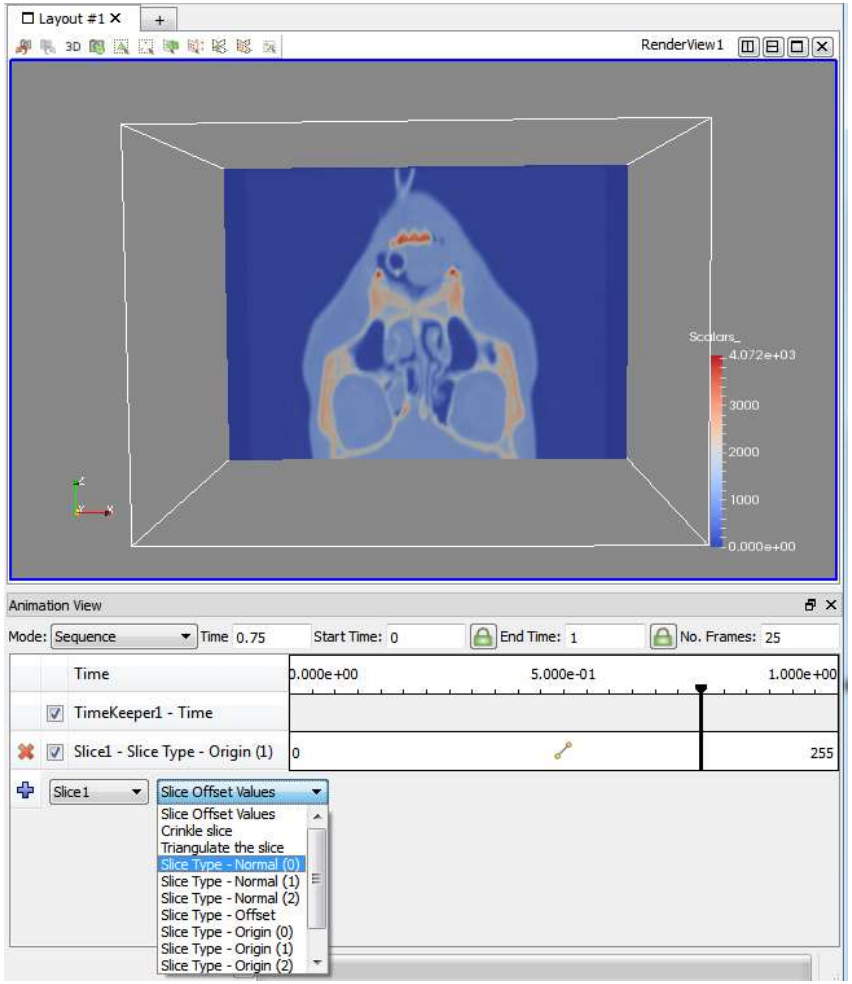
- Reset the session and open the file can.ex2, enable all variables.
- Color the surface by VEL
- Generate a comparative view with a 3x2 layout showing 6 different time steps.



Animations

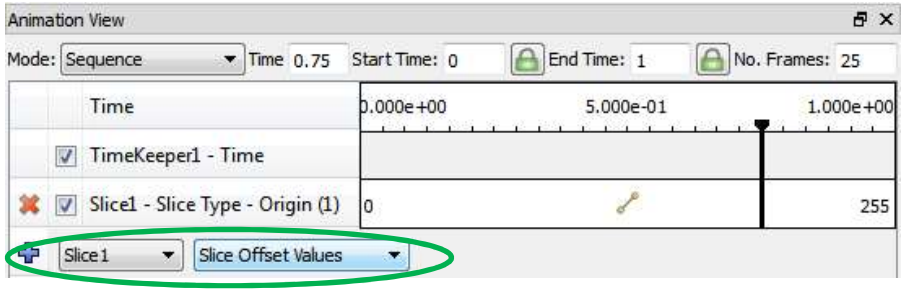
With ParaView, you can animate

- Data time steps
- nearly any property of any pipeline object
- The camera, to perform camera flights along a specified path or orbit.
- Use Python scripts to manipulate the scene every time step

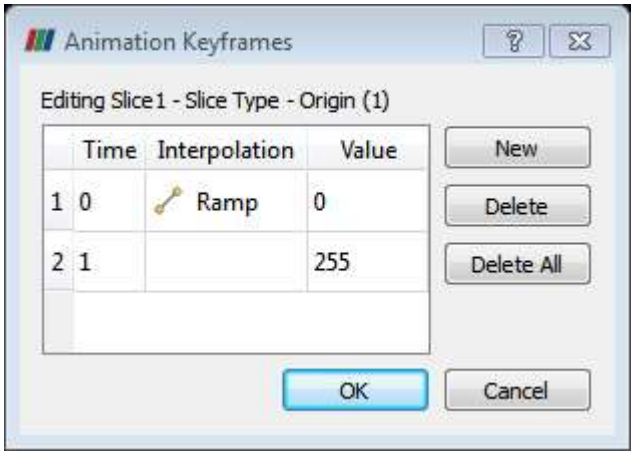


Animating Properties

- Select object and property from the combo boxes within the Animation View, press 
  - ➢ A new track is created, holding key frames that specify values for the property at a specific time instance

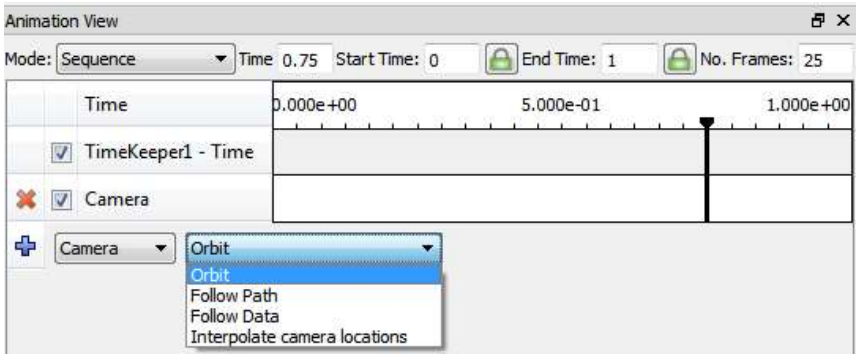


- Double-click at the new track to edit the keyframes:



Animating the camera

- Select “Camera” in the object combo box within the Animation View.

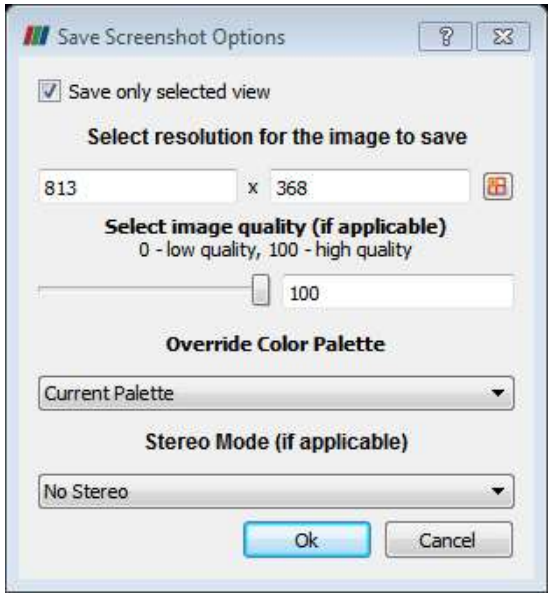


- Choose one of the options:
  - “Orbit” and “Follow Path” to create a (closed) spline widget around your data along which the camera is animated
  - “Follow Data” to look at the data in every step of the animation.
  - “Interpolate camera locations” to create a path between two or more locations.

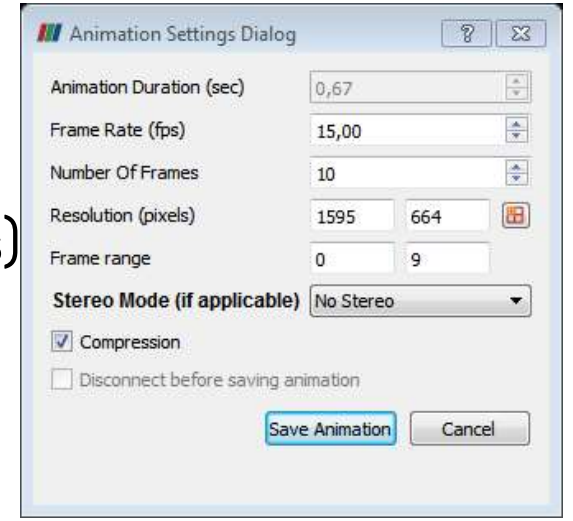
- Double-click at the new element to edit the keyframes

Saving Screenshots and Animations

- File -> Save Screenshot
  - Save the current view(s) as PNG, BMP, TIFF, PPM, JPEG, PDF
- File -> Export Scene
  - save your Image as a vector graphic

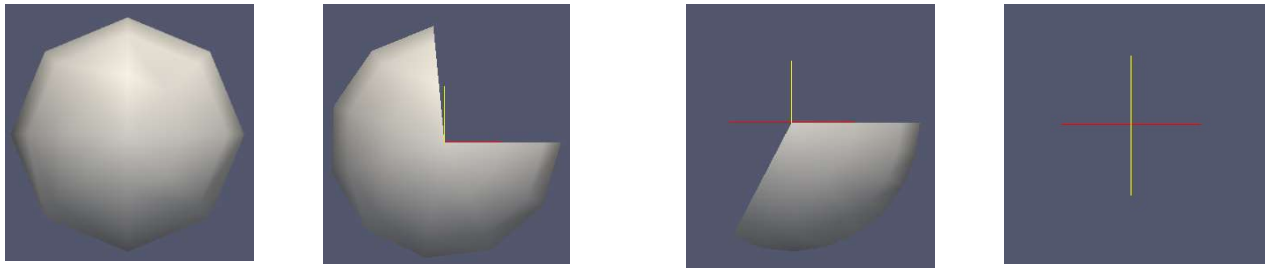


- File -> Save Animations saves the current animation as
  - Ogg/Theora (many open source viewers) AVI (windows, some open source viewers)
  - Sequence of JPEG, TIFF, PNG images



Exercise VIII: Animations

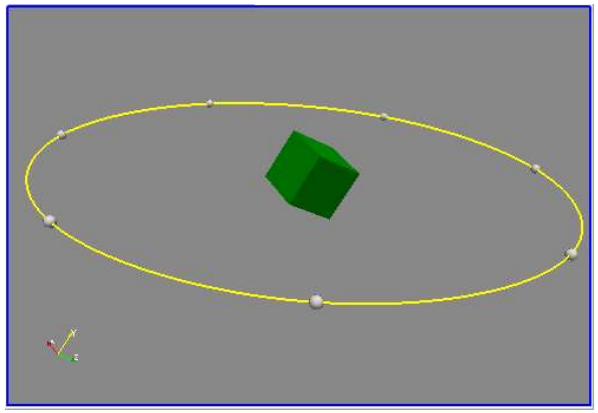
1. Animating Properties
   - Create a sphere from the Sources-Menu
   - Animate the Start Theta property of the sphere from 0 to 360.
   - Set Number of Frames to 50.
   - Save the animation as an AVI file.

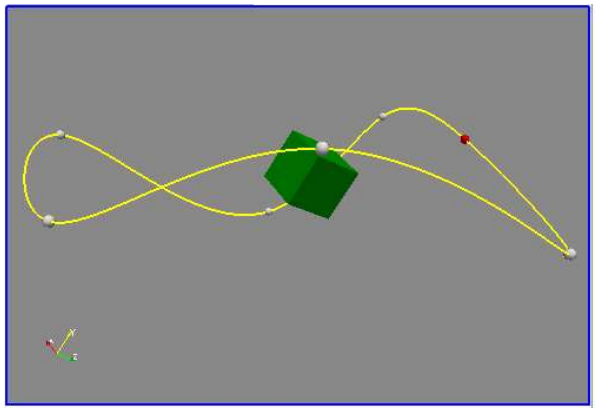


2. Modifying Animation Track Keyframes
   - Modify the animation track so that the sphere first becomes smaller, from 360 to 0 degress, and them becomes greater 0 to 360 degrees

Exercise VIII: Animations (cont.)

3. Animating the camera
   - Create a Box from the Sources-Menu
   - select “real time” as animation mode and 10 seconds duration
   - Animate the camera in an orbit around your box
4. Modify the key frames of the Orbit
   - Open the key frame editor, double-click on “Path …”
   - Select “Camera Position” in the Key Frame Interpolation Dialog
   - Modify the Spline Widget (now visible in the 3D view) in some way

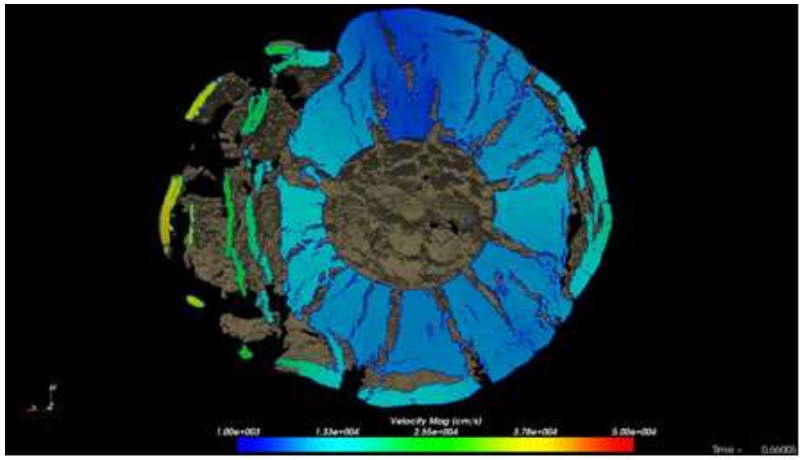


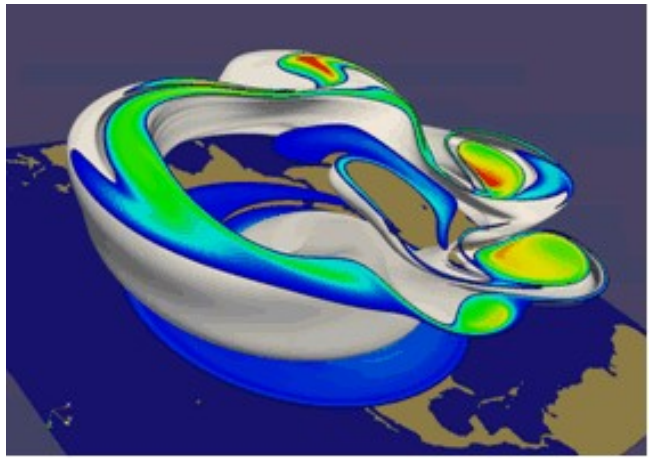


Visualizing Large Models

ParaView is meant for extremely large data processing:

Examples from Sandia National Lab in the web:





(about1 billion cells on 256 nodes)

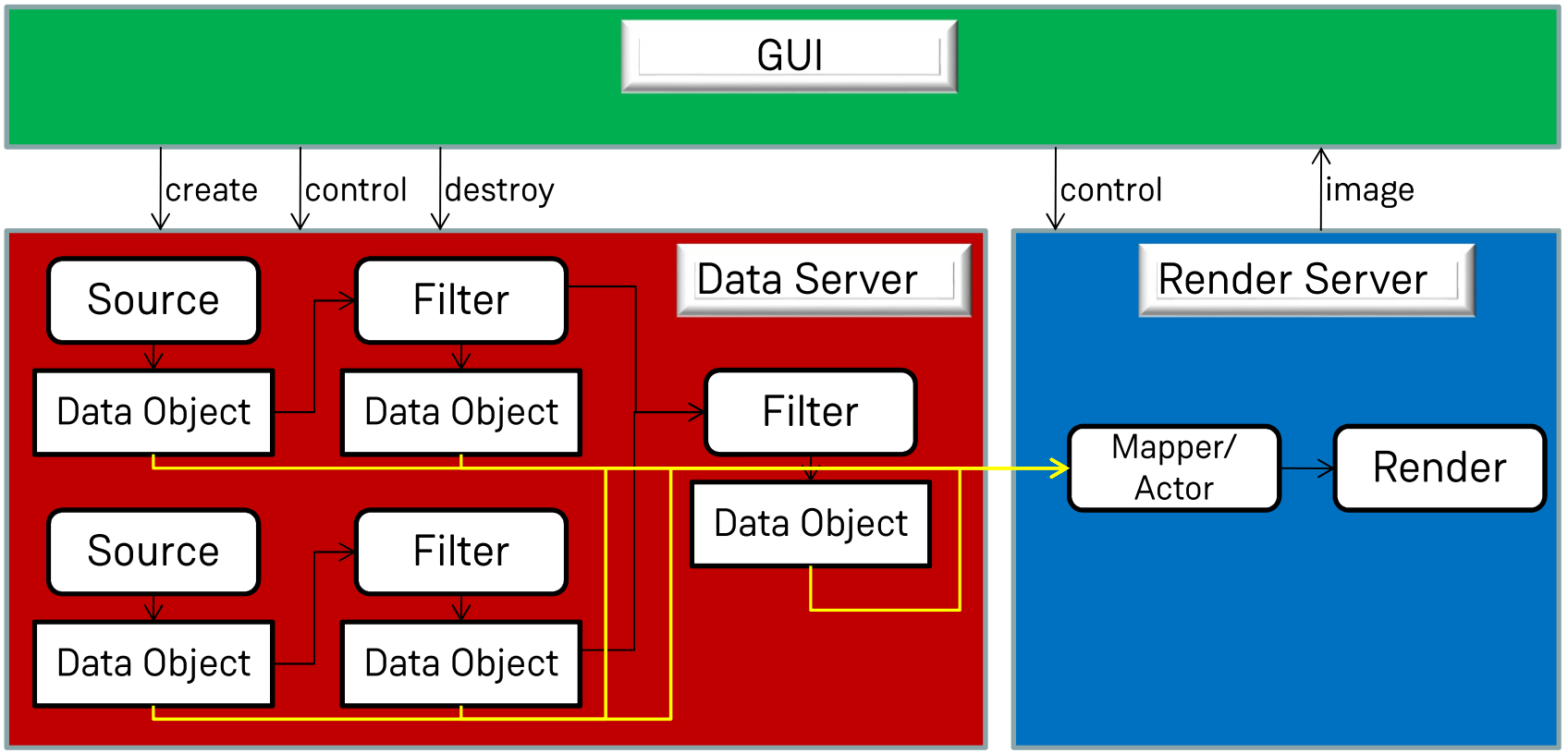
- Visualizing extremely large data requires a big amount of resources (especially memory)

➢ Use parallel visualization capabilities of ParaView

Parallel ParaView

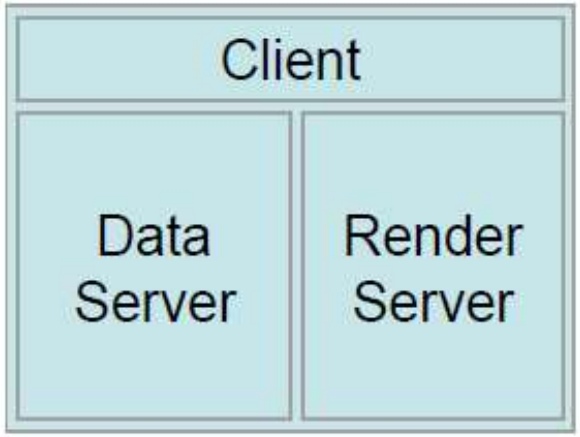
- ParaView has three main components:
  - GUI (paraview)
  - Data Server (pvdataserver)
  - Render Server (pvrenderserver)

Server (pvserver)



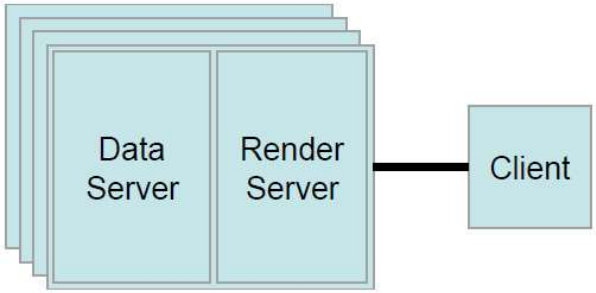
Parallel ParaView (continued)

- ParaView can be started in non parallel (standalone) mode:
  - All three components in one single process
    - *command: paraview*



Parallel ParaView (continued)

- ParaView can be started in two parallel modes:
  1. Local client and parallel server (data server and render server in one job)



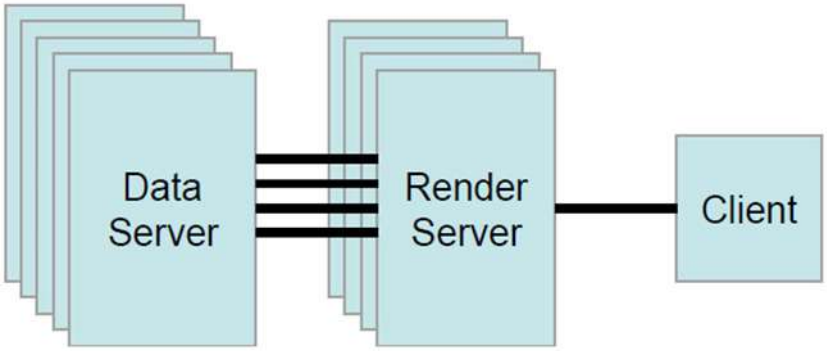
     - command on local client: *paraview*
     - command on remote cluster:

       mpiexec –n <num_processes> pvserver
     - connect client and server

Parallel ParaView (continued)

2. Local client and parallel data and render server (data server and render server may run on different machines)
   - command on local client: *paraview*
   - command on data processing server:
     mpiexec –n <num_dataserver> pvdataserver –m=machines.pvx
   - command on render server:
     mpiexec –n <num_renderserver> pvrenderserver –m=machines.pvx

   (num_dataserver >= num_renderserver)



Parallel ParaView (continued)

Guideline:

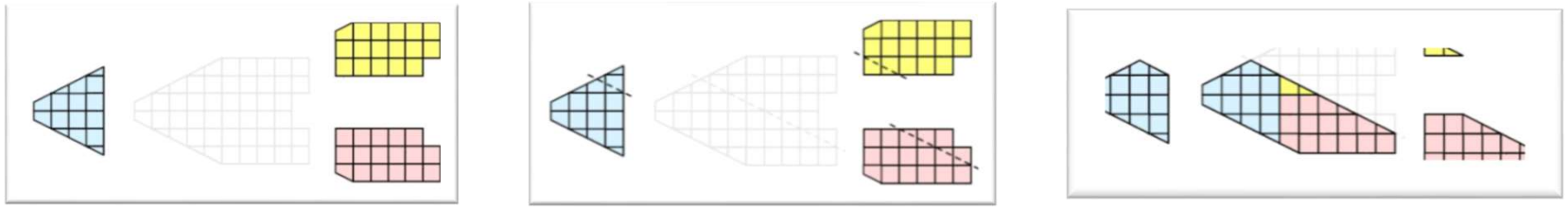
- Cost of connecting data server and render server over TCP sockets often overwhelms the benefit!
  - ➢ run data server and render server in one process!
- Avoid X11 forwarding! (inefficient, lots of low level communication underneath)
  - Run client on desktop, NOT on login-node
  - Or use Xpra+VirtualGL (mayby VNC) on login-node **(our recommendation)**

Advantages of using Parallel ParaView:

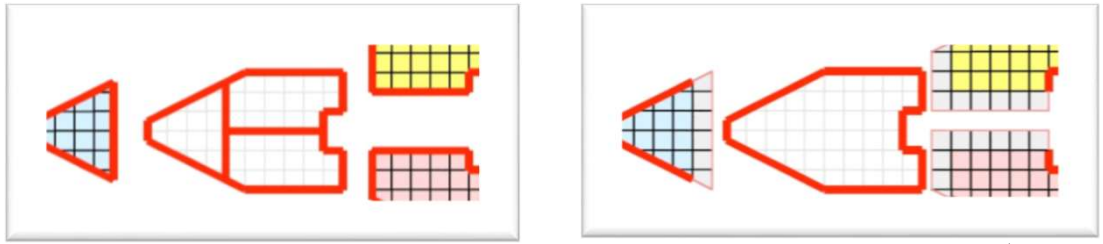
- Local view on remote data: “File -> Open” shows you the data server’s file system => big data may stay on the server!

Data Distribution

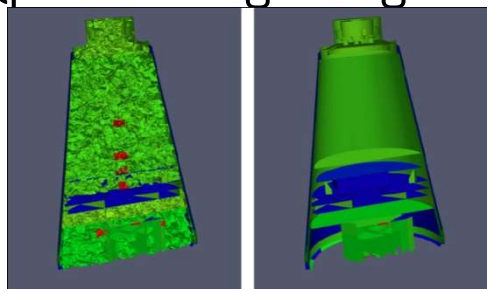
- Data needs to be distributed on a cluster (load balancing, memory usage)
- Some visualization algorithms work directly on distributed data, e.g. clipping



- Other algorithms need ghost cells, e.g. external faces



- Structured data is handled automatically (partitioning and ghost cells)
- For unstructured data use the D3 filter
- Example: extract surface filter with and without D3:



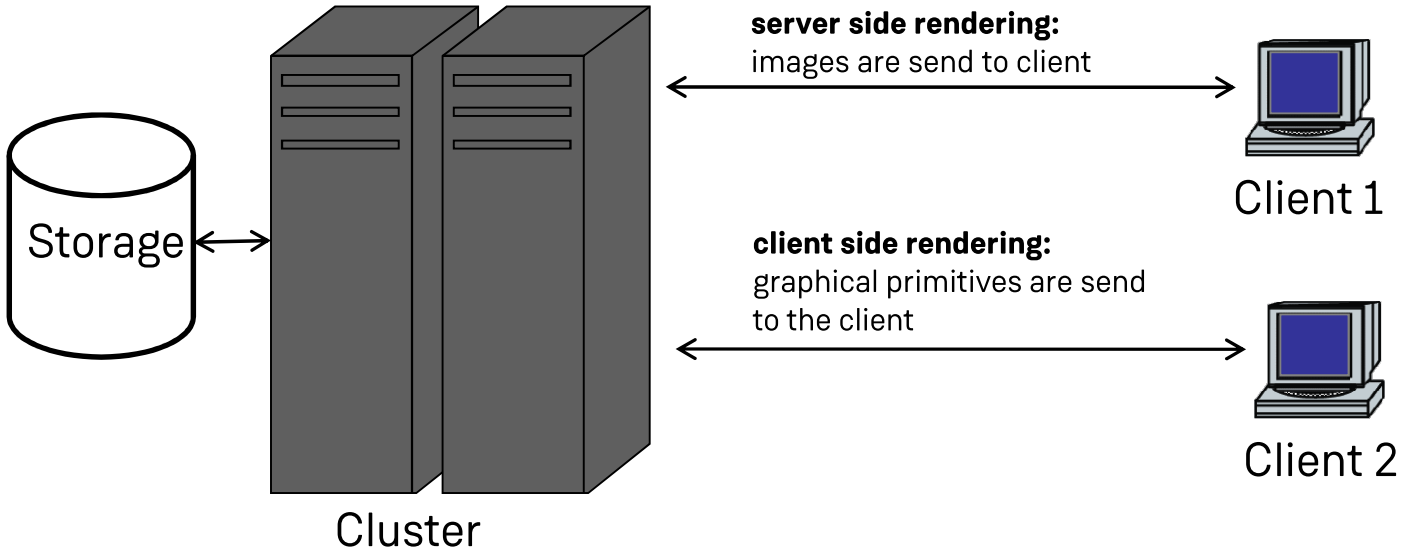
Memory Footprint

- Memory footprint of structured data is MUCH lower than unstructured data, because topology (and maybe geometry) is implicitly given
- Many filters need to convert structured data to unstructured data: BEWARE OF DATA EXPLOSION!
  - Examples: clip, threshold, ..., many, many others!
  - Some filters reduce the dimensionality → less dangerous
    Examples: contour, slice, stream tracer

- "Save" filters just don't change the data structure but only add new data values (attributes)
  Examples: calculator, gradient, curvature

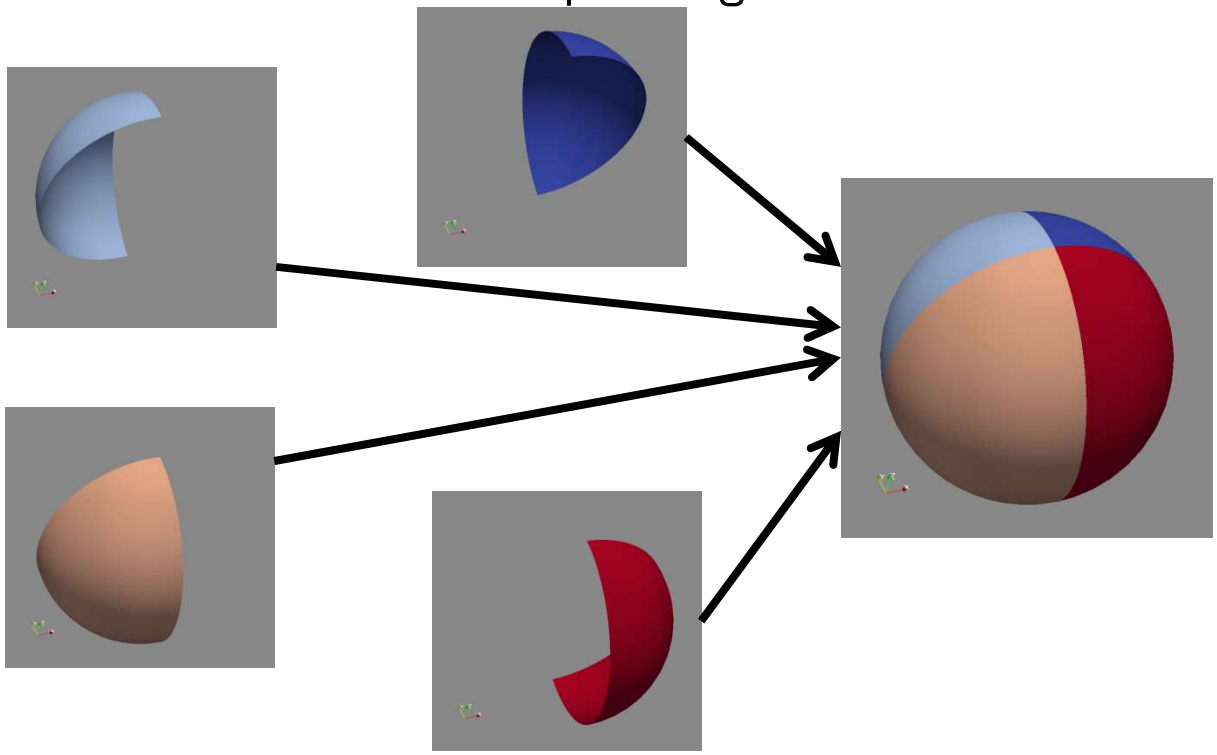
Remote Rendering

- **Server side rendering** and
- **client side rendering** possible:



Remote Rendering (Parallel Rendering)

- Depth Compositing: All nodes will render their own part of the data (local data)
  - Nodes cooperate to decide what the color of each pixel is based on the depth value
  - Implemented with the ICE-T compositing lib



Remote Rendering

- Recommended settings for Remote Rendering, see https://docs.paraview.org/en/latest/ReferenceManual/parallelDataVisualization.html#parallel-render-parameters
  - Edit -> Settings -> Render View -> Remote/Parallel Rendering Options:
    - Remote render threshold: set the data size beyond which to render the data remotely
    - Still Render Image Reduction Factor: image subsampling for still rendering
  - Edit -> Settings -> Render View -> Client/Server Rendering Options:
    - Image Reduction Factor: image subsampling for interactive rendering
    - Image Compression: LZ4, Squirt or Zlib for image compression

Visualization at JSC

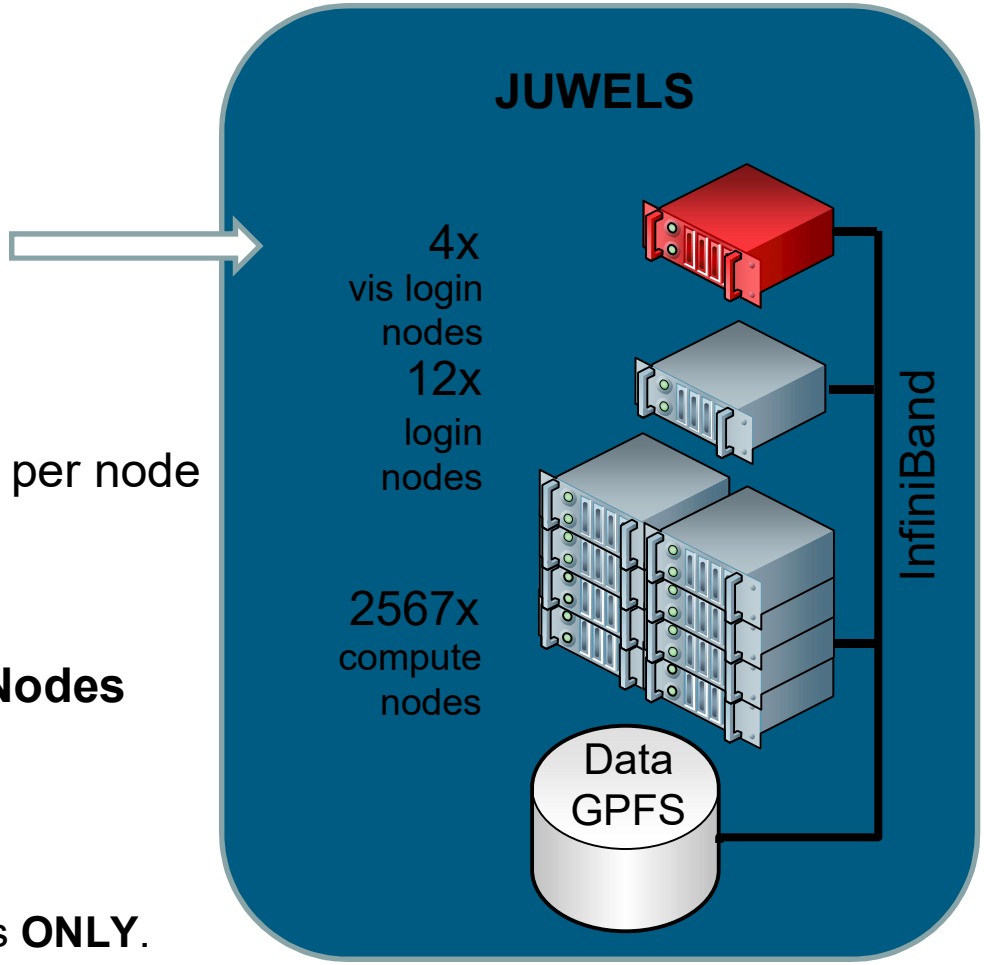
JUWELS: General Hardware Setup

4 x Visualization Login Nodes

- juwelsvis.fz-juelich.de
- (juwelsvis00 to juwelsvis03 in round-robin fashion)
- 768 GB RAM each

- 1 GPUs Nvidia Pascal P100 per node
- 12 GB RAM on GPU

No specific Visualization Batch Nodes

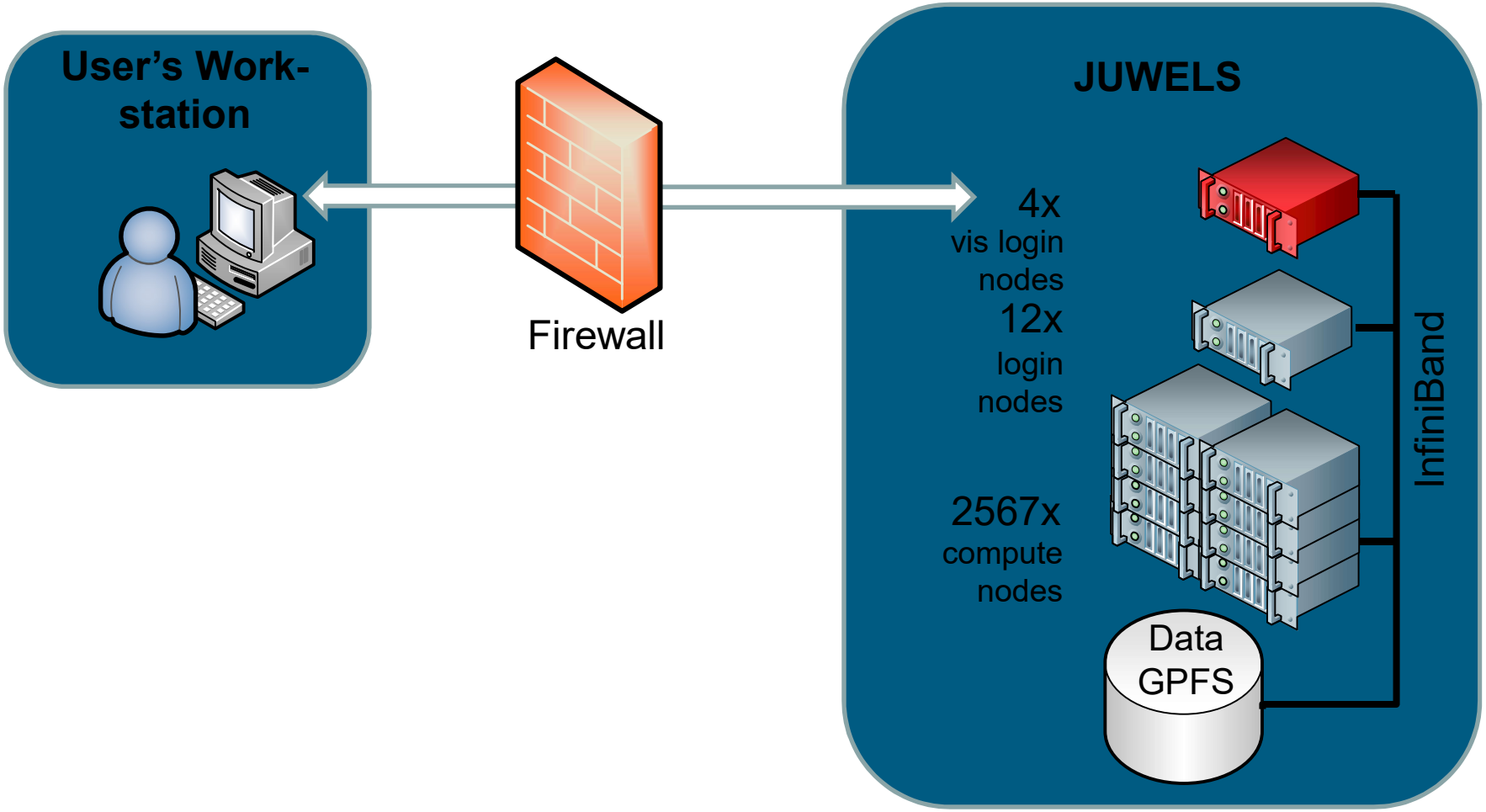


Keep in mind:

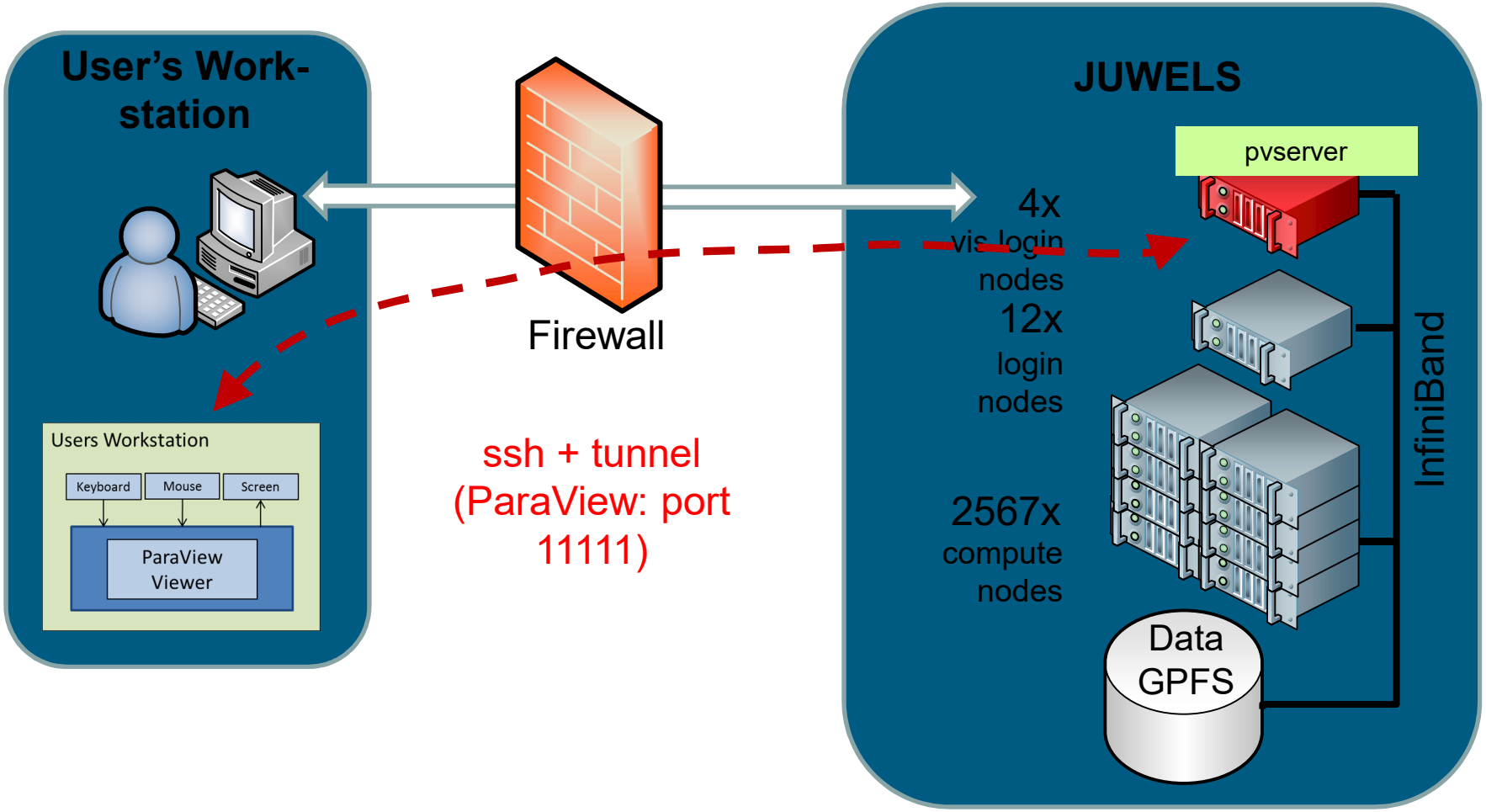
Visualization is **NOT** limited to vis. nodes **ONLY**.
(software rendering is possible on any node)

Remote 3D Visualization

General Setup



Remote 3D Visualization



Remote 3D Visualization
at Jülich Supercomputing Centre

JÜLICH Forschungszentrum

- X forwarding + Indirect Rendering
  slow, maybe incompatible → bad idea
- “remote aware” visualization apps (ParaView, VisIt)
  application dependent, error-prone setup
- Xpra - stream application content with H.264 + VirtualGL
  fast, our recommendation → good idea
- VNC (Virtual Network Computing) + VirtualGL
  full remote desktop, but slower than Xpra → medium good idea

Remote 3D Visualization

with Xpra (or VNC) + VirtualGL

JÜLICH Forschungszentrum

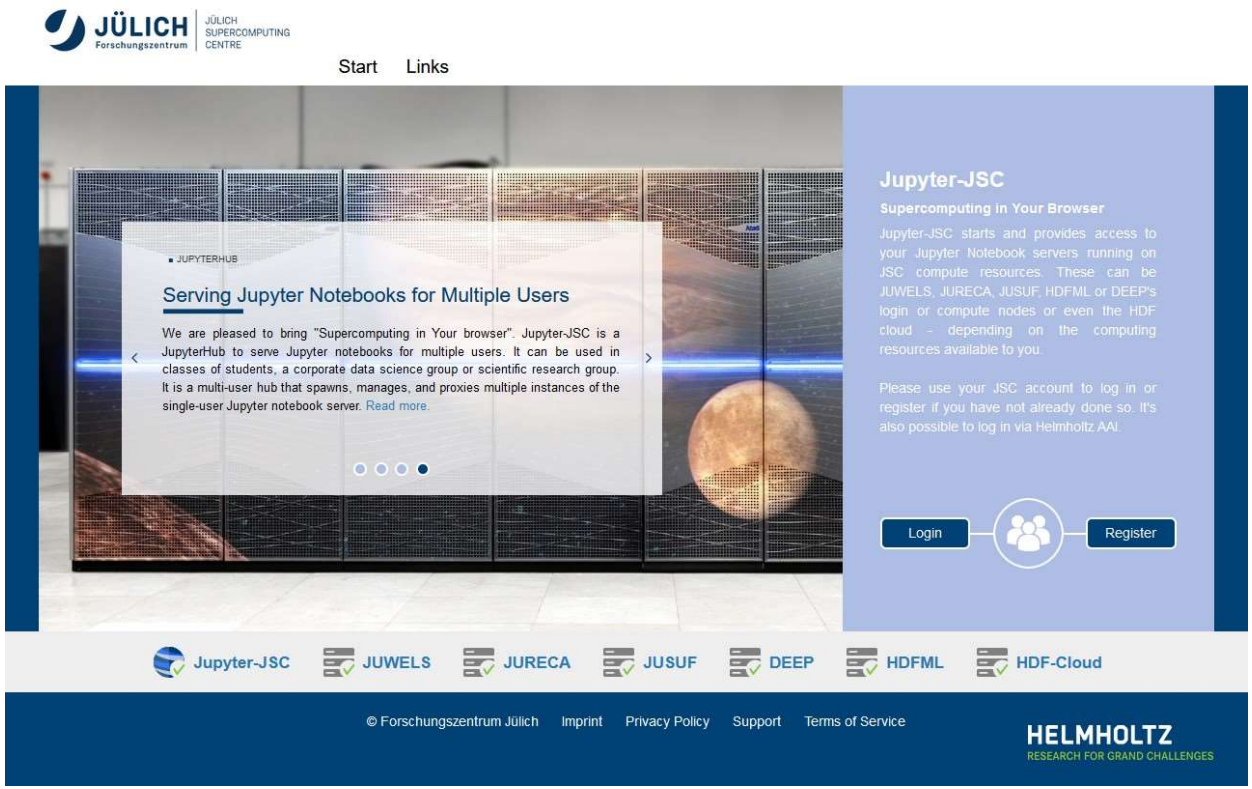
- X-applications forwarded by Xpra (or VNC) appear on the local desktop as normal windows
- allows disconnection and reconnection without disrupting the forwarded application

- **advantages**
  - **No X is required** on user´s workstation (X display on server).
  - **No OpenGL is required** on user´s workstation (only images are send).
  - Quality of visualization does **not depend** on user´s workstation.
  - Data size send is **independent** from data of 3d scene.
  - Disconnection and reconnection possible.
- **VirtualGL** for hardware accelerated rendering: use `vglrun <application>`
  - it **intercepts the GLX** function calls from the application and **rewrites them**.
  - The corresponding GLX commands are then sent to the X display of **the 3d X server**, which has a 3D hardware accelerator attached.

- Good solution for any OpenGL application e.g. ParaView, VisIt, IDL, VMD, PyMol, …

Xpra Integration in JupyterLab@JSC

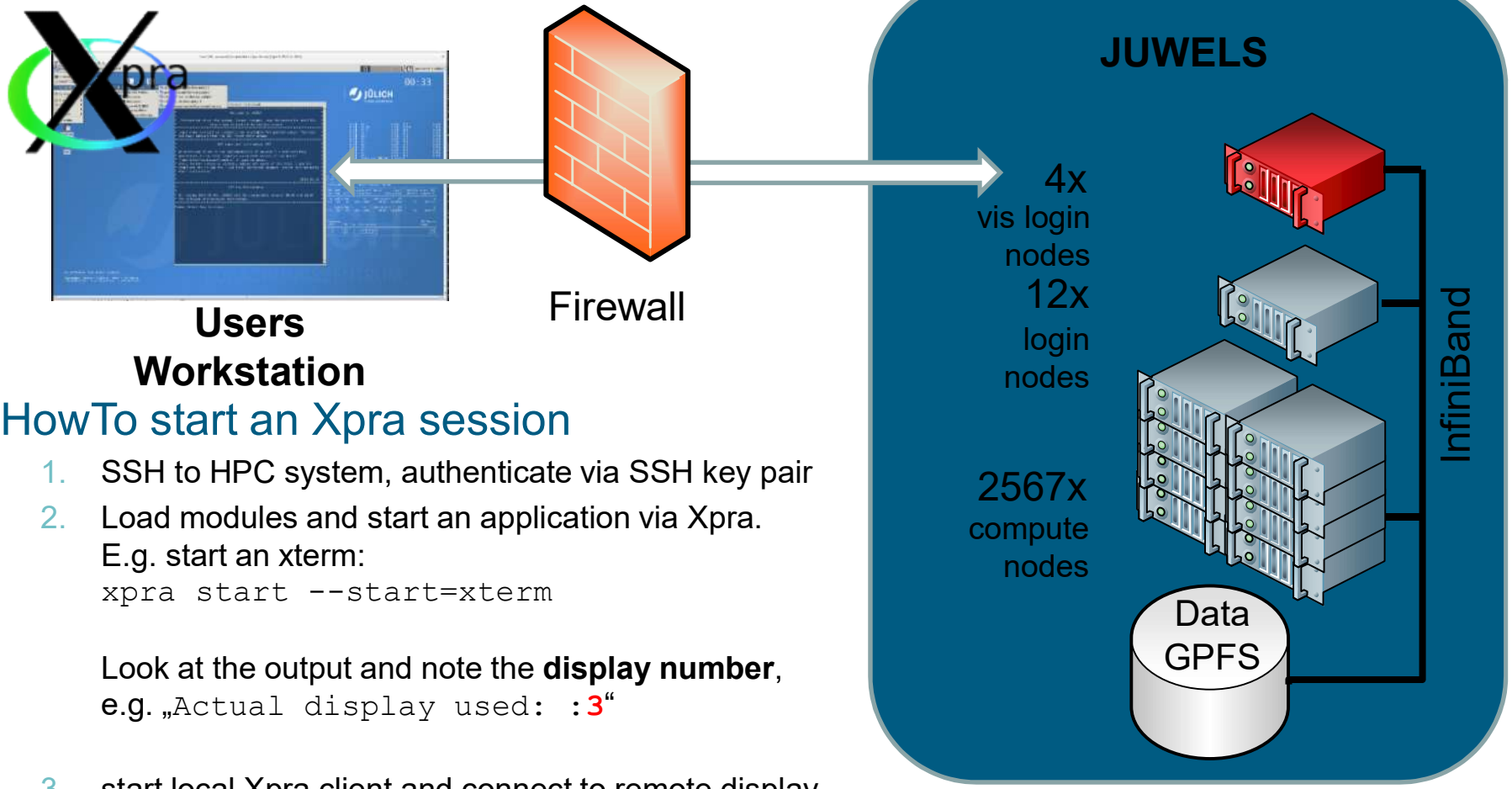
- How to start Xpra-Session:
  - **At JSC:** Within **JupyterLab@JSC** https://jupyter-jsc.fz-juelich.de



  - **Alternative**: start session manually, see next slides

Remote 3D Visualization

with Xpra + VirtualGL



HowTo start an Xpra session

1. SSH to HPC system, authenticate via SSH key pair
2. Load modules and start an application via Xpra.
   E.g. start an xterm:
   `xpra start --start=xterm`

   Look at the output and note the **display number**,
   e.g. „`Actual display used: :3`“
3. start local Xpra client and connect to remote display
4. Start visualization application in the xterm
5. Stop the Xpra session by `xpra stop :3`

Setup Xpra

Step 1: login to a (visualization) login node

- **Linux:**
  `ssh <USERID>@juwelsvis02.fz-juelich.de`
- **Windows:**
  connect via a ssh client, e.g. PuTTY. The PuTTY ssh keyagent pageant may be usefull, too.

Setup Xpra

Step 2: start xpra on HPC node and notice the display-number in the output

For example, start an xterm:

```
jwvis02> module --force purge
jwvis02> module use otherstages
jwvis02> ml Stages/Devel-2020  GCCcore/.9.3.0 xpra/4.0.4-Python-
3.8.5

jwvis02> xpra start --start=xterm
...
Actual display used: :3
```

- The display-number is needed to connect to the Xpra session

Setup Xpra

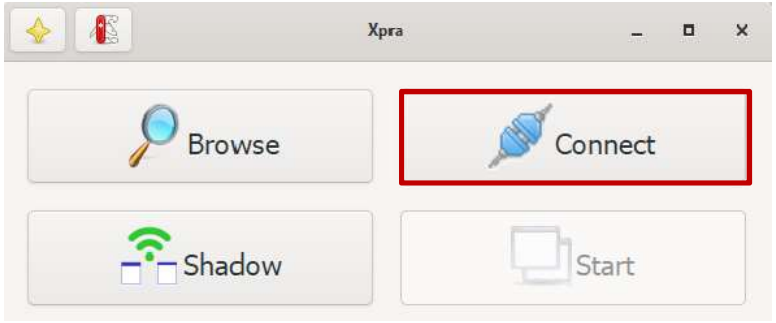
Step 3: connect to Xpra session

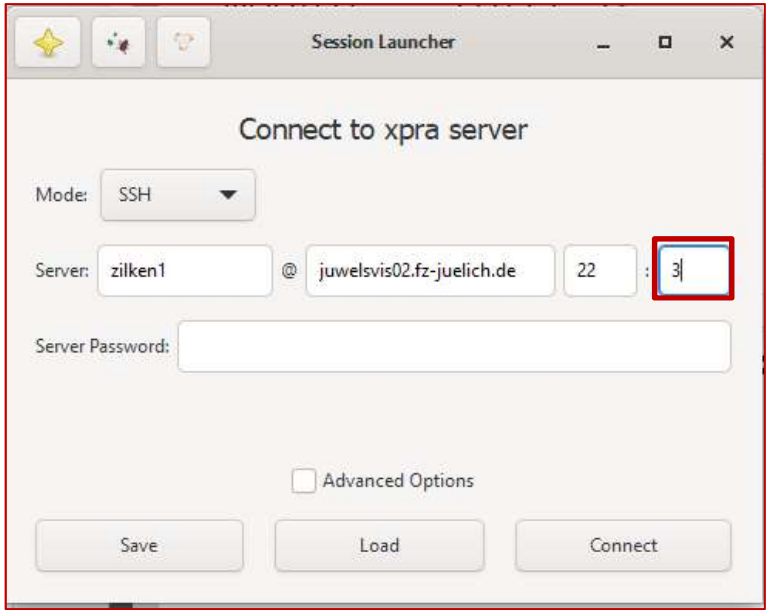
Install Xpra on your local machine. Download from www.xpra.org

Linux: use command

```
local_machine> xpra attach
ssh://USERNAME@juwelsvis02.fz-juelich.de/3
```

Windows: use Xpra GUI:



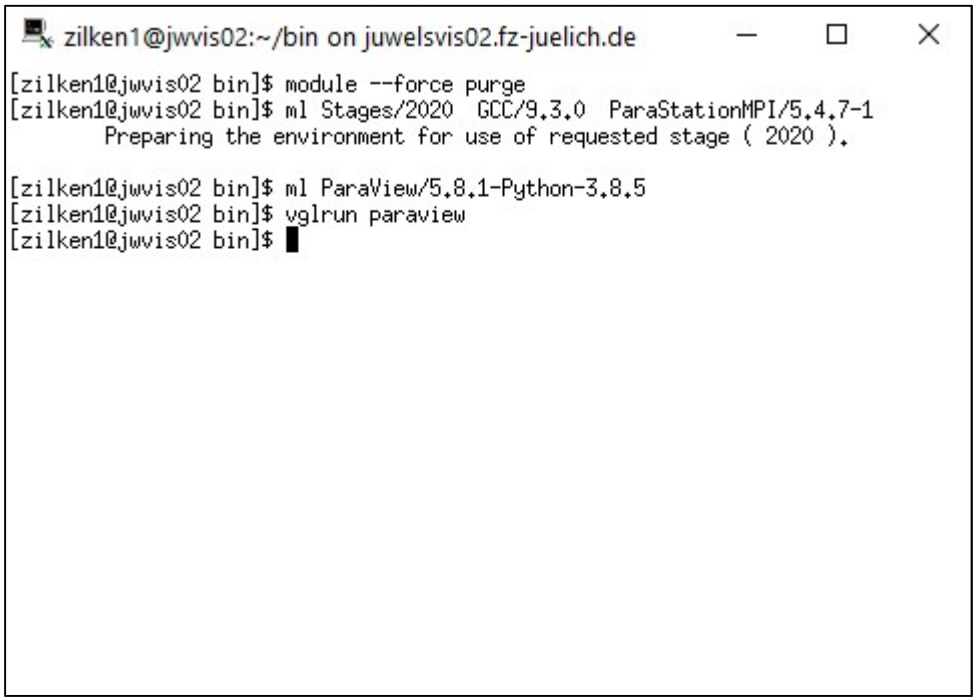


Setup Xpra

Step 4: start visualization application

After successful connection, an xterm window will show up on your local desktop.

Start your application there, e.g. ParaView:



Step 5: When you are done, stop the session by

```
jwvis02> xpra stop :3
```

ParaView Python Scripting

- ParaView can be fully controlled by a Python script

Reasons to do this:

- For batch processing of data (many files, many time steps, many different visualization methods)
- To store and reconstruct (and to document) the state of a (complex) ParaView pipeline
- As a workaround for some ParaView flaws
  - e.g. memory leak when loading time steps:
    close and restart ParaView every N frames (just to clear the memory) and let the script resume the animation exactly at that point

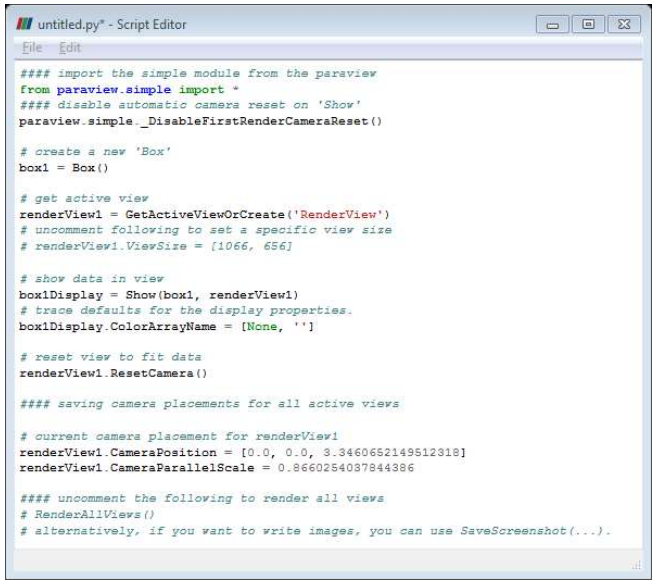
ParaView Python Scripting

Some useful links:

- https://www.paraview.org/Wiki/ParaView/Python_Scripting
- https://kitware.github.io/paraview-docs/latest/python/paraview.simple.html
- https://trac.version.fz-juelich.de/vis/wiki/Examples/ParaviewAnimating
- https://trac.version.fz-juelich.de/vis/wiki/Examples/Ear5Animating

ParaView Python Scripting: How To

- The GUI is a good tool to prototype the scene, especially pipeline properties, color tables and camera positions
- Automated facility for creating Python scripts: Tracing
- Tools -> Start Trace to begin trace recording,
  Tools -> Stop Trace to end it
- Produces a python script that reconstructs many (not all ☹) actions performed in the GUI
- Script can be started by
  `pvpython ./script.py`



- Images can be saved by
  `SaveScreenshot(„./foo.jpg“, renderView, ImageResolution=[1920,1080])`

From Gui To Script

- Color tables can be saved in the GUI and loaded in the script by

```
ImportPresets('colortable.json')
```

- Camera positions can also be saved in an xml file and loaded in the script by

```
import xml.etree.ElementTree as ET
from paraview.simple import *

def assignCameraParameters(root, camera, camIdx):
   camera.SetPosition( (root[camIdx-1][1][0][0][0][0].attrib['value']), (root[camIdx-
1][1][0][0][0][1].attrib['value']), (root[camIdx-1][1][0][0][0][2].attrib['value']))
   camera.SetFocalPoint((root[camIdx-1][1][0][0][1][0].attrib['value']), (root[camIdx-
1][1][0][0][1][1].attrib['value']), root[camIdx-1][1][0][0][1][2].attrib['value']))
   camera.SetViewUp((root[camIdx-1][1][0][0][2][0].attrib['value']), (root[camIdx-
1][1][0][0][2][1].attrib['value']), (root[camIdx-1][1][0][0][2][2].attrib['value']))
   camera.SetParallelScale((root[camIdx-1][1][0][0][6][0].attrib['value']))

tree = ET.parse('camera.pvcvbc')
camera = GetActiveCamera()
assignCameraParameters(tree.getroot(), camera, 1)
```

Macros

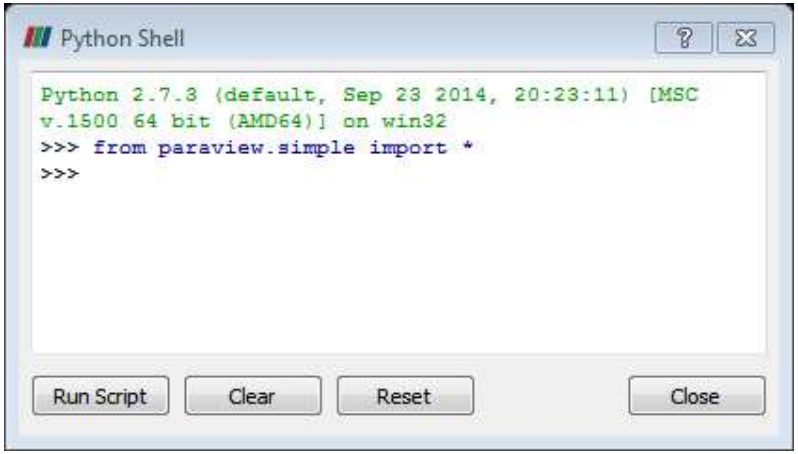
- Scripts can by used as a Macro (=automated script)
- Macros can be invoked from the Menu Button Macros or directly inside the toolbar by clicking on it.
- View and customize macros within the Macro Menu:
  - edit, delete existing macros,
  - add new macros (load python scripts from file system)
- Use Macros for:
  - Creating data from scratch
  - Applying filters to existing data (to automate the same visualization on different data)

pvpython and pvbatch

- Command line programs that execute Python scripts
- Feeding scripts from command line or file to the python interpreter
- pvpython:
  - serial program
  - equivalent to the GUI-Client, replacing the GUI with the python interpreter
- pvbatch:
  - parallel application, can be launched with mpirun
  - runs without user interaction, saves images to disk

ParaView Python Scripting – basic bindings

- First thing any ParaView Python script must do is load the **paraview.simple** module: `from paraview.simple import *`
- Automatically invoked in the scripting dialog in ParaView (Tools->Python Shell)



- Add it yourself when writing script from scratch
- **paraview.simple** defines convenient functions for
  - Sources
  - Readers
  - Filters
  - Writers

ParaView Python Scripting - functions

- Have the same name as shown in the GUI menus, but spaces and special characters removed:
  - Sphere = Sources->Sphere
  - PlotOverLine = Filters->Data Analysis->Plot Over Line
- Create a pipeline object that shows up in the pipeline browser (except for writers)
- Return an object that can be used to query and manipulate properties
- List of functions available in paraview.simple:
  - dir(paraview.simple)
  - help(paraview.simple) -> verbose listing
- Example: https://trac.version.fz-juelich.de/vis/wiki/Examples/Brain

Python Scripting – Simple Example

Creating and showing a source

```
from paraview.simple import *
```

needs to be invoked at the beginning of any ParaView Python script

```
import os
os.chdir(“path_to_directory”)
```

generated Output will be written to the specified path

```
sphere = Sphere()
```

create a sphere pipeline object

```
Show()
```

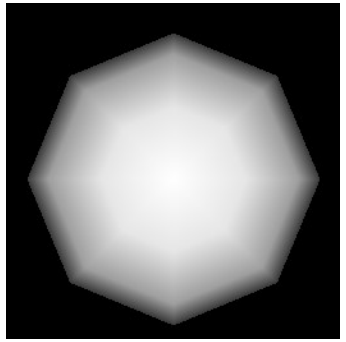
turn on visibility of this object

```
Render()
```

cause results to be seen in the view

```
WriteImage("test1.png")
```

create an output



Python Scripting – Simple Example

Creating and Showing a Filter

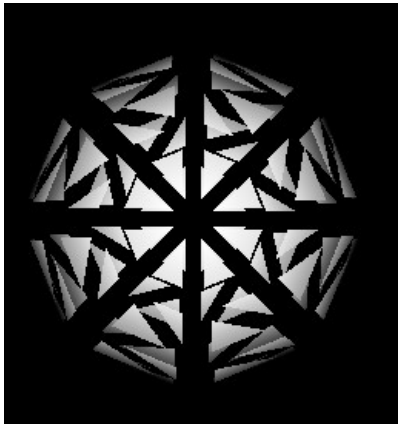
```
Hide()
shrink=Shrink()
Show()
Render()
WriteImage("test2.png")
```

hides the sphere

adds a Shrink Filter to the sphere

and shows that

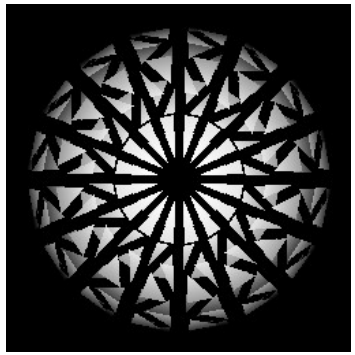
Shrink Filter makes all the polygons smaller and give the mesh an exploded type of appearance



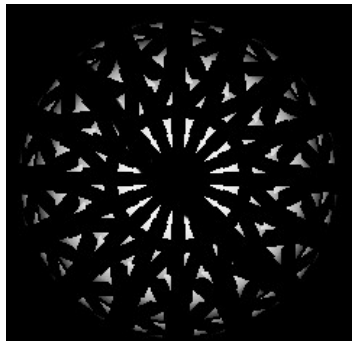
Python Scripting – Simple Example

Changing pipeline object properties

`help(sphere)`	get a full list of properties
`print sphere.ThetaResolution`	print current value of theta resolution (=8)
`sphere.ThetaResolution=16`	set it to a new value
`Render()`	turn on visibility
`WriteImage("test3.png")`	create an output



`print shrink.ShrinkFactor`	print current ShrinkFactor
`shrink.ShrinkFactor=0.25`	set it to a smaller value
`Render()`	turns on visibility
`WriteImage("test4.png")`	create an output

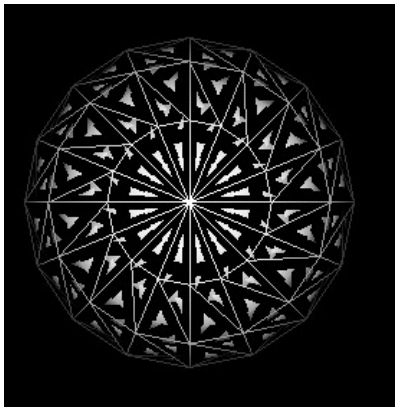


Python Scripting – Simple Example

Branching Pipelines

```
wireframe=ExtractEdges(Input=sphere)
Show()
Render()
WriteImage("test5.png")
```

add a second filter to the sphere source

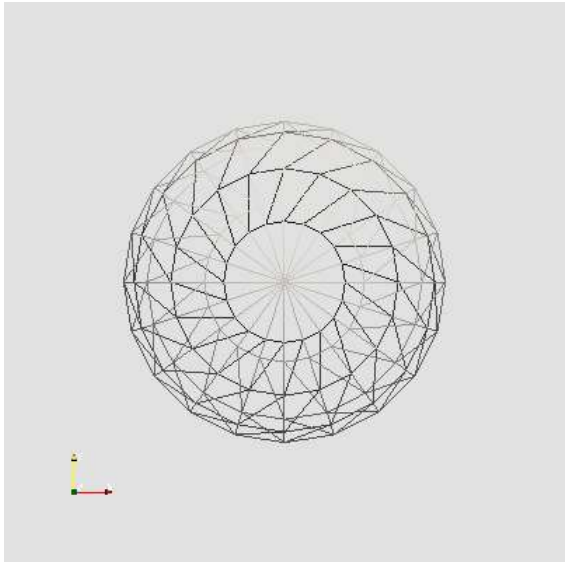


Note: Any action is performed to the active object

- get/set active object via GetActiveSource/SetActiveSource functions
- Set the Input property upon construction of an object to by-pass this behavior

Exercise IX: Create a Python Script and run it

- Create a Python script by **tracing** ParaView:
  - Create a Sphere Source with ThetaResolution=20, hit apply
  - Apply the Extract Edges filter to the sphere
  - Save a screenshot
- Test the script as a macro
- Test the script with pvpython
  (Note: you have to edit the script and add the “Connect( )” command at the beginning)



Loading Data with Programmable Source

Example: load csv data.

Input files: .csv (text) files with naming scheme
TOAR_YYYYMMDD_HHMM.csv

Content: latitude, longitude, value, of an unconnected “point cloud” e.g.:

```
 8.3082,  54.9250,  46
12.7253,  54.4367,  42
 6.0939,  50.7547,  29
 8.5484,  52.0232,  26
 6.8746,  51.8620,  32
 6.9769,  51.5260,  28
 7.4575,  51.5369,  25
 . . . . .
```

Loading Data with Programmable Source

Request Information Script:

```
def setOutputTimesteps(algorithm, timesteps):
    """helper routine to set timestep information"""
    executive= algorithm.GetExecutive()
    outInfo = executive.GetOutputInformation(0)

    outInfo.Remove(executive.TIME_STEPS())
    for timestep in timesteps:
        outInfo.Append(executive.TIME_STEPS(), timestep)

    outInfo.Remove(executive.TIME_RANGE())
    outInfo.Append(executive.TIME_RANGE(), timesteps[0])
    outInfo.Append(executive.TIME_RANGE(), timesteps[-1])
```

Loading Data with Programmable Source

JÜLICH Forschungszentrum

Request Information Script (continued)

```
def getFileList(fpath, fprefix, fpostfix):
    """get all files"""
    import os
    files=[]
    for i in os.listdir(fpath):
        if os.path.isfile(os.path.join(fpath,i)) \
           and fprefix in i \
           and fpostfix in i:
            files.append(i)
    files.sort()
    return files

def getTimeTuple(fname, fprefix, fpostfix):
    """get timetuple from file name"""

    import time
    import datetime

    ftime_str  = fname[len(fprefix):-len(fpostfix)]
    ftimetuple =
    time.mktime(datetime.datetime.strptime(ftime_str,"%Y%m%d_%H%M").timetuple())

    return ftimetuple
```

Loading Data with Programmable Source

Request Information Script (continued)

```
def getTimestepsFromFiles(files, fprefix, fpostfix):
    """generate timestap from filename"""
    ref_ftime = 0.0
    ftime_sec = []
    for i, fname in enumerate(files):
        cur_ftime = getTimeTuple(fname, fprefix, fpostfix)
        if i == 0:
            ref_ftime = cur_ftime
        ftime_sec.append(cur_ftime -ref_ftime)
    ftime_h = [ x / 3600 for x in ftime_sec ]
    ftime_h_int = [int(i) for i in ftime_h]
    return ftime_h_int
```

Loading Data with Programmable Source

Request Information Script (continued)

```
# get all files
fpath="/data/30-Projekte/2018-03-
   12_TagDerWissenschaft/m.schultz/"
fprefix="TOAR_"
fpostfix=".csv"
files = getFileList(fpath, fprefix, fpostfix)

timestamps = getTimestepsFromFiles(files, fprefix, fpostfix)

# get time-steps
setOutputTimesteps(self, timestamps)
```

Loading Data with Programmable Source

Main Script

```
def GetUpdateTimestep(algorithm):
    """Returns the requested time value, or None if not
   present."""
    executive = algorithm.GetExecutive()
    outInfo = executive.GetOutputInformation(0)
    return outInfo.Get(executive.UPDATE_TIME_STEP()) \
              if outInfo.Has(executive.UPDATE_TIME_STEP()) else
   None

*** getFileList, getTimeTupe, getTimestepsFromFiles as above
```

Loading Data with Programmable Source

Main Script (continued)

```
def getFileFromTimestep(tstep, fpath, fprefix, fpostfix):
    """generate filename for time-step"""
    import time
    import datetime
    files = getFileList(fpath, fprefix, fpostfix)
    timestamps = getTimestepsFromFiles(files, fprefix, fpostfix)
    # get a tstep, which we have data for
    ret_tstep = min(timestamps, key=lambda x:abs(x-tstep))
    # get date-string from tstep
    ref_ftime = getTimeTuple(files[0], fprefix, fpostfix)
    ftimetuple = time.localtime(ref_ftime + ret_tstep*3600)
    tstr = time.strftime("%Y%m%d_%H%M", ftimetuple)
    # create filename
    ret_fname = fpath + fprefix + tstr + fpostfix

    return ret_tstep, ret_fname
```

Loading Data with Programmable Source

Main Script (continued)

```
def readCSVFile(fname, output):
    import math
    import csv
    import vtk
    newPts = vtk.vtkPoints()
    valArray = vtk.vtkDoubleArray()
    valArray.SetName('Ozone')
    valArray.SetNumberOfComponents(1)
    # fill VTK objects
    with open(fname) as csvfile:
        csvReader = csv.reader(csvfile, delimiter=',')
        numPts = sum(1 for row in csvReader)
        i = 0
        csvfile.seek(0)
        for row in csvReader:
            # read row
            # longitude, latitude, (ozone) value [letzterer in nmol mol-
    1]
```

Loading Data with Programmable Source

Main Script (continued)

```
            lon = float(row[0])
            lat  = float(row[1])
            val  = float(row[2])
            # convert longitude,latitude,altitude to x,y,z
            # spherical coordinates to cartesian coordinates on a unit
    sphere (radius=1)
            radTmp = math.cos(lat);
            z = math.sin(lat);
            x = math.cos(lon +1.5*math.pi) *radTmp;
            y = math.sin(lon +1.5*math.pi) *radTmp;

            i = i+1
            newPts.InsertPoint(i-1, x,y,z)
            valArray.InsertValue(i-1, val)

        output.SetPoints(newPts)
        output.GetPointData().AddArray(valArray)
```

Loading Data with Programmable Source

Main Script (continued)

```
# req_time is the requested time-step.
# This may not be exactly equal to the time-step published in
   RequestInformation() (the code must handle that)
req_time = GetUpdateTimestep(self)

# Use req_time to determine which CSV file to read
fpath="/data/30-Projekte/2018-03-12_TagDerWissenschaft/m.schultz/"
fprefix="TOAR_"
fpostfix=".csv"
ret_time, ret_fname = getFileFromTimestep(req_time, fpath, fprefix,
   fpostfix)

# read CSV file
#pdo = self.GetPolyDataOutput()
readCSVFile(ret_fname, output)

# Mark the time-step produced
output.GetInformation().Set(output.DATA_TIME_STEP(), ret_time)
```

ParaView Plugins - Introduction

- https://www.paraview.org/Wiki/ParaView/Plugin_HowTo
- ParaView comes with a large range of readers, filters and views providing a large range of functionality
- ParaView provides an extensive plugin mechanism which enables the user to
  - add new **readers**, writers and **filters**
  - add custom GUI components such as toolbar buttons to perform common tasks
  - add new views for displaying data
- The following slides cover
  - how to use existing plugins
  - how to write new plugins for ParaView.

Using ParaView Plugins

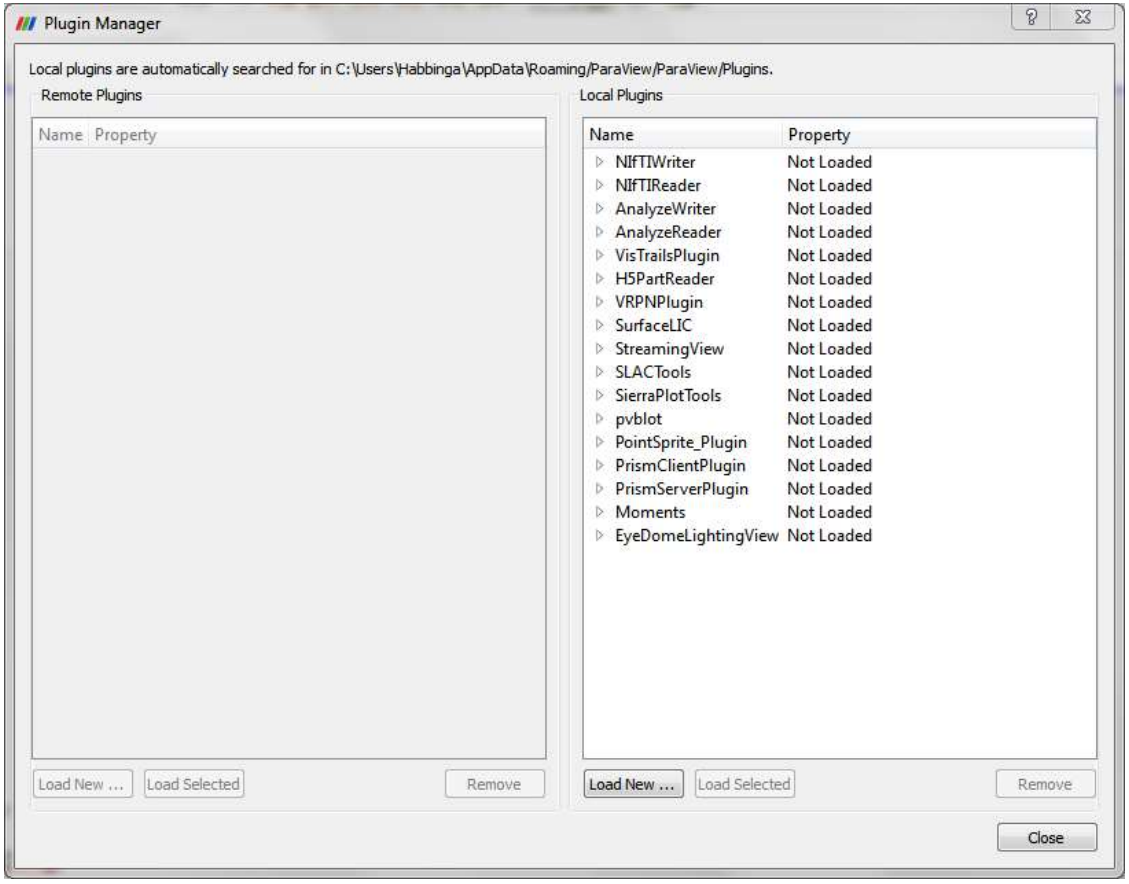
- Plugins are distributed as shared libraries (*.so on Unix, *.dylib on Mac, *.dll on Windows)
- Plugin must be built with the same version of ParaView
- ParaView has to be built with shared libs enabled
- three ways for loading plugins:
  - using environment variable PV_PLUGIN_PATH (Auto-loading plugins)
  - placing the plugins in a recognized location:
    - “plugins” subdirectory beneath the directory containing the paraview client or server executables
    - “plugins” subdirectory in the user's home area.

      $HOME/.config/ParaView/ParaView<version>/Plugins on Unix/Linux/Mac

      %APPDATA%\ParaView\ParaView<version>\Plugins on Windows
  - using the GUI

Loading ParaView Plugins Using the GUI

- Plugin manager accessible from Tools | Manage Plugins menu:



Writing ParaView Plugins – Server Manager Configuration

- ParaView uses the VTK Server Manager for building distributed client-server applications by means of the Proxy design pattern
- Proxies are defined in the **Server Manager Configuration XML**
- Proxies are organized in **ProxyGroups**
  - Examples: sources, filters, views, ...
- Each **Proxy** stores a reference to the corresponding VTK object located on the server
  - Examples: SourceProxy, ViewProxy, ...
- The Proxy contains **Properties** to describe the part of the interface which is exposed to the client (function name and arguments)
  - Examples: IntVectorProperty, StringVectorProperty, InputProperty, ...
- The range of values of a property can be restricted by **Domains**
  - Examples: IntRangeDomain, FileListDomain, DataTypeDomain, ProxyGroupDomain,...

Enabling existing VTK-Classes as Plugins

- To enable an existing VTK-class, write a Server Manager configuration xml-file and load it into the ParaView GUI
- example: enable the VTK-filter **vtkCellDerivatives**, write CellDerivatives.xml as follows

```
<?xml version="1.0"?>
<ServerManagerConfiguration>
  <ProxyGroup name=„filters“>
    <SourceProxy name=„MyCellDerivatives“ class=„vtkCellDerivatives“ label=„My Cell Derivatives“>
      <Documentation
            long_help=„Create point attribute array by projecting points onto an elevation vector.“
            short_help=„Create a point array representing elevation.“>
      </Documentation>
      <InputProperty
            name=„Input“
            command=„SetInputConnection“>
            <ProxyGroupDomain name=„groups“>
                <Group name=„sources“/>
                <Group name=„filters“/>
            </ProxyGroupDomain>
            <DataTypeDomain name=„input_type“>
                <DataType value=„vtkDataSet“/>
            </DataTypeDomain>
       </InputProperty>
    </SourceProxy>
  </ProxyGroup>
</ServerManagerConfiguration>
```

Writing New ParaView Plugins – Source Code

- Compiling plugins requires own build of ParaView since binaries downloaded from www.paraview.org do not include necessary header files
- Example: simple reader called **vtkCSVImageReader**
  - read a 2D image from a text file with simple delimited format
  - allow someone to export simple matrices of data from Excel and visualize them as heat maps or height fields in ParaView
  - assume the files consist of strictly numeric values separated by commas or some other delimiter
  - use a vtkDelimitedTextReader instance which will read delimited text into a vtkTable
  - methods to set and get the file name and field delimiter characters, which are delegated to the vtkDelimitedTextReader instance variable

Writing New ParaView Plugins

- Writing a new plugin requires generating the following files:
  - .cxx and .h files (the actual plugin source code)
  - Server manager configuration (xml-file)
  - configuration xml for the GUI (optional for filters)
  - CMakeLists.txt for compiling your source code into a shared library
- Hint: try to find similar plugin in the ParaView sourcecode (src/Plugins, src/Examples/Plugins)
- Use this code as a starting point

Writing New ParaView Plugins – Source Code

- Step 1: override the following methods of the chosen vtk superclass to make it operate properly in a standard vtk pipeline:
  - tell the VTK pipeline information about the data before creating the output, e.g. extent, spacing and origin in case of vtkImageData

```
int vtkCSVImageReader::RequestInformation( vtkInformation *request,
                                           vtkInformationVector **inputVector,
                                           vtkInformationVector *outputVector)
```

  - read data from the file and store it in the corresponding data object in the output port

```
int vtkCSVImageReader::RequestData(   vtkInformation *request,
                                      vtkInformationVector **inputVector,
                                      vtkInformationVector *outputVector );
```

- Step 2: override other methods, e.g. **SetFileName**(const char* fname), **SetFieldDelimiterCharacters**(const char* delim), …

Class vtkCSVImageReader

```
class vtkCSVImageReader : public vtkImageAlgorithm
      {
      public:
         static vtkCSVImageReader* New();
         vtkTypeRevisionMacro(vtkCSVImageReader,vtkImageAlgorithm);
         void PrintSelf(ostream& os, vtkIndent indent);
         virtual void SetFileName(const char* fname);
         virtual const char* GetFileName();
         virtual void SetFieldDelimiterCharacters(const char* delim);
         virtual const char* GetFieldDelimiterCharacters();
      protected:
        vtkCSVImageReader();
        ~vtkCSVImageReader();

        int RequestInformation(vtkInformation*, vtkInformationVector**,
      vtkInformationVector*);
        int RequestData(vtkInformation*, vtkInformationVector**, vtkInformationVector*);
        vtkDelimitedTextReader* Reader;

      private:
        vtkCSVImageReader(const vtkCSVImageReader&);
        void operator=(const vtkCSVImageReader&);  };
```

RequestInformation(...)

```
int vtkCSVImageReader::RequestInformation (vtkInformation*, vtkInformationVector**,
      vtkInformationVector* outputVector)
      {
        vtkInformation* outInfo =outputVector->GetInformationObject(0);
        this->Reader->Update();
        vtkTable* output = this->Reader->GetOutput();
        vtkIdType rows = output->GetNumberOfRows();
        vtkIdType columns = output->GetNumberOfColumns();

        int ext[6] = {0, rows, 0, columns, 0, 0};
        double spacing[3] = {1, 1, 1};
        double origin[3] = {0, 0, 0};

        outInfo->Set(vtkStreamingDemandDrivenPipeline::WHOLE_EXTENT(), ext, 6);
        outInfo->Set(vtkDataObject::SPACING(), spacing, 3);
        outInfo->Set(vtkDataObject::ORIGIN(), origin, 3);
        vtkDataObject::SetPointDataActiveScalarInfo(outInfo, VTK_FLOAT, 1);
        return 1;
      }
```

RequestData(...)

```
int vtkCSVImageReader::RequestData(vtkInformation*, vtkInformationVector**,
      vtkInformationVector* outputVector)
      {
        vtkImageData* image = vtkImageData::GetData(outputVector);
        vtkTable* output = this->Reader->GetOutput();
        vtkIdType rows = output->GetNumberOfRows();
        vtkIdType columns = output->GetNumberOfColumns();
        image->SetDimensions(rows, columns, 1);
        image->AllocateScalars();
        vtkDataArray* scalars = image->GetPointData()->GetScalars();
        scalars->SetName("Data");
        for (vtkIdType r = 0; r < rows; ++r)
          {
          for (vtkIdType c = 0; c < columns; ++c)
            {
            vtkVariant val = output->GetValue(r, c);
            float f = val.ToFloat();
            scalars->SetTuple1(c*rows + r, f);
            }
          }
        return 1;
      }
```

Writing New ParaView Plugins - Server Manager Configuration

- Server Manager Configuration for the example CSVImageReader:

```
<?xml version="1.0"?>
    <ServerManagerConfiguration>
        <!– Begin CSVImageReader -->
        <ProxyGroup name=„sources“>
                <SourceProxy name=„CSVImageReader“
                                class=„vtkCSVImageReader“>
                    <StringVectorProperty name=„FileName“
                                                number_of_elements=„1“
                                        command=„SetFileName“>
                                <FileListDomain name=„files“/>
                    </StringVectorProperty>
                    <StringVectorProperty
                                name=“FieldDelimiterCharacters”
                                command=“SetFieldDelimiterCharacters”
                                number_of_elements=”1”
                                default_values=”,”/>
                </SourceProxy>
         </ProxyGroup>
         <!– End CSVImageReader -->
 </ServerManagerConfiguration>
```

Writing New ParaView Plugins – Configuration XML for the GUI

- Provides the higher-level information about where new elements will be placed in the user interface.
- Example: CSVImageReaderGUI.xml

```
<?xml version="1.0"?>
<ParaViewReaders>
    <Reader
        name=„CSVImageReader“
        extensions=„csvimg“
        file_description=„csv Image data“>
    </Reader>
</ParaViewReaders>
```

  - tells ParaView to create GUI elements for “CSVImageReader”
  - creates a new entry in the File Open dialog for CSV image data files
  - associates files ending in “.csvimg” with the new reader

Writing ParaView plugins – CMakeLists.txt

- Use cmake to build the ParaView plugin
- CMakeLists.txt file ties sources and xml files together

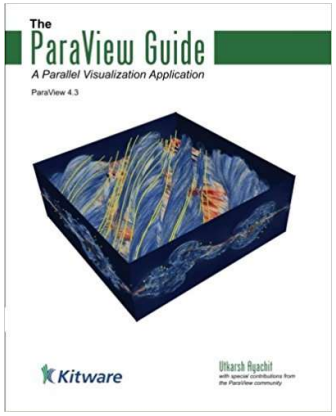
```
CMAKE_MINIMUM_REQUIRED(VERSION 2.6)
PROJECT(CSVImageReader)

FIND_PACKAGE(ParaView REQUIRED)
INCLUDE(${PARAVIEW_USE_FILE})

ADD_PARAVIEW_PLUGIN(CSVImageReader "1.0"
                    SERVER_MANAGER_XML CSVImageReader.xml
                    SERVER_MANAGER_SOURCES vtkCSVImageReader.cxx
                    GUI_RESOURCE_FILES CSVImageReaderGUI.xml)
```

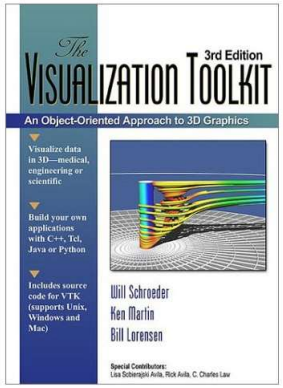
- locate ParaView with FIND_PACKAGE
- import the CMake configuration parameters from ParaView by including PARAVIEW_USE_FILE.
- ADD_PARAVIEW_PLUGIN specifies all the information needed to build the plugin.

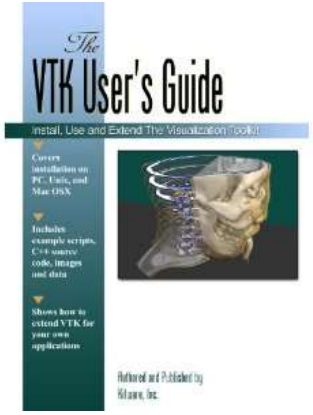
Documentation

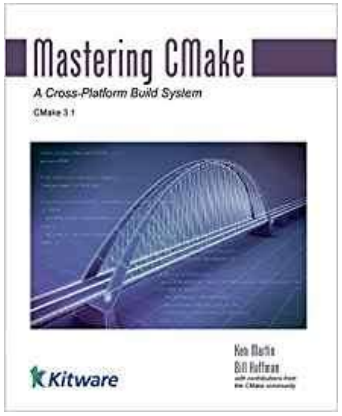
- The Paraview User's Guide
- The Visualization Toolkit
- The VTK Guide
- Mastering Cmake

All are published by Kitware, Inc.









ParaView Guide (Community Edition), Getting Started and Tutorial (pdf) are included in ParaView download

ParaView Links

ParaView Homepage: http://www.paraview.org

ParaView Wiki: http://www.paraview.org/Wiki/ParaView

- Importing Data
- Extending Paraview
- ParaView Server Setup
- Tutorials

ParaView Documentation:
https://www.paraview.org/documentation/

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

ParaView Forum (since Oct. 2018): https://discourse.paraview.org/

Link to (no longer supported) mailing list, can be useful to search old posts:
https://www.paraview.org/community-support/

Thank you for your attention!