

Remote Visualization at JSC (with ParaView)

Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Germany
Algorithms, Tools & Methods Lab Visualization

Herwig Zilken, 2023/06/01

Visualization at JSC

Algorithms, Tools & Methods Lab Visualization

- **Scientific Visualization**
 - R&D + support
for visualization of scientific data

- **Virtual/Augmented Reality**
 - VR visualization based on Unreal Engine, with head mounted displays and tablet computers
for data analysis and presentation

- **Multimedia**
 - multimedia productions
for websites (e.g. Youtube), presentations or on TV

Visualization at JSC

JUWELS: closer look at login nodes

Cluster

4 x Login Nodes with GPU

- juwelsvis00 to juwelsvis03
- (juwelsvis.fz-juelich.de)
- 768 GB RAM each
- 1 GPUs Nvidia Pascal P100 per node
- 12 GB RAM on GPU

9 x Login Nodes without GPU

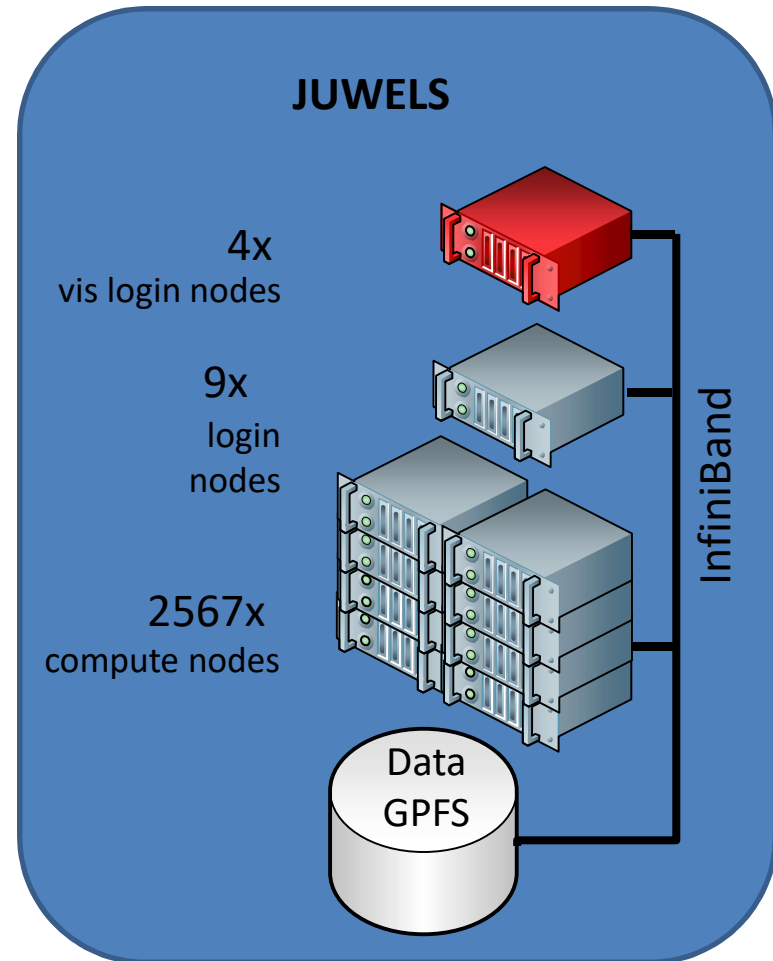
- juwels-cluster

Booster:

4 x Login Nodes without GPU

- juwels-booster
- no Xserver, no GPU → limited usage for visualization

Keep in mind: software rendering is possible on any node



Visualization at JSC

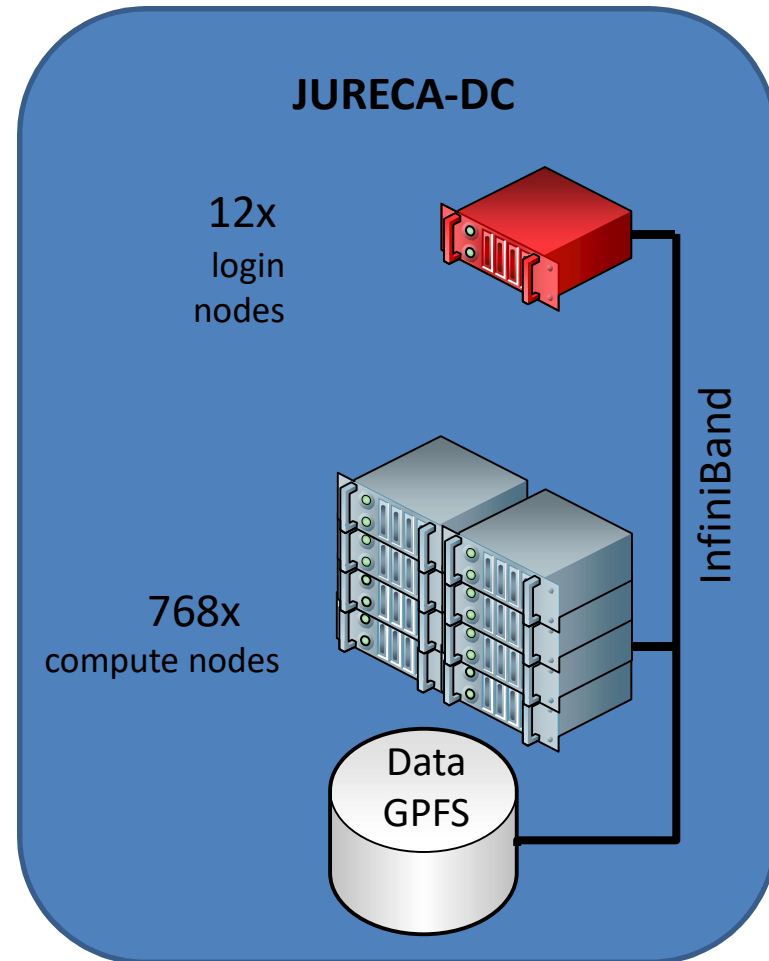
JURECA-DC: closer look at login nodes

12 x Login Nodes with GPU

- jureca01 to jureca12
- (jureca.fz-juelich.de)
- 1024 GB RAM each

- 2 x Nvidia Quadro RTX8000 per node
- 48 GB RAM on each GPU

Keep in mind: software rendering is possible on any node



Visualization at JSC

General Software Setup

Typical Software Stack for Visualization

Base Software:



X-Server, X-Client (Window-Manager)



OpenGL (libGL.so, libGLU.so, libglx.so), Nvidia or Mesa driver

Middleware:



Xpra



Virtual Network Computing: VNC-Server, VNC-Client



VirtualGL (for remote hardware rendering, if possible)

Parallel and Remote Rendering App, In-Situ Visualization:



ParaView



Visit

Other Visualization Packages (more packages on user demand):

Blender, GPicView, VTK, VMD

Remote 3D Visualization

at Jülich Supercomputing Centre

- X forwarding + Indirect Rendering
slow, maybe incompatible → bad idea
- “intrinsic remote capable” visualization apps
application dependent error-prone setup
- Xpra - stream application content with H.264 + VirtualGL
 - Can be started with a click in Jupyter@JSC
- VNC + VirtualGL can be used also
 - Simple, but full featured remote desktop
 - Does not run from Jupyter@JSC

Remote 3D Visualization

with X Forwarding + Indirect Rendering

Traditional Approach (X forwarding + Indirect Rendering)

```
ssh -X <USERID>@<SERVER>
```

- uses GLX extension to X Window System
- X display runs on user workstation
- OpenGL commands are encapsulated inside X11 protocol stream
- OpenGL commands are executed on user workstation
- **disadvantages**
 - User's workstation requires a running **X server**.
 - User's workstation requires a **graphic card** capable of the required OpenGL.
 - User's workstation defines the **quality and speed** of the visualization.
 - User's workstation requires **all data needed** to visualize the 3d scene.
 - This approach is known to be error prone (OpenGL version mismatch, ...)

Try to **AVOID** for 3D visualization.

Remote 3D Visualization

with Xpra/VNC + VirtualGL

- 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

<https://xpra.org/>

<https://sourceforge.net/projects/turbovnc/>

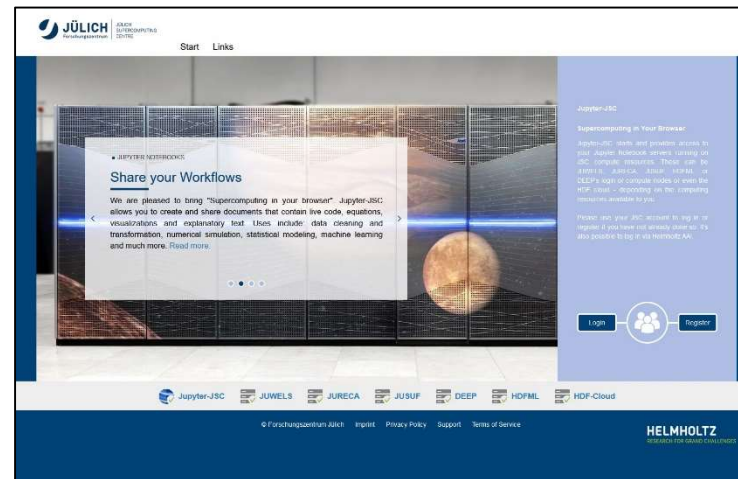
How to use Xpra @ JSC

How to start an Xpra session:

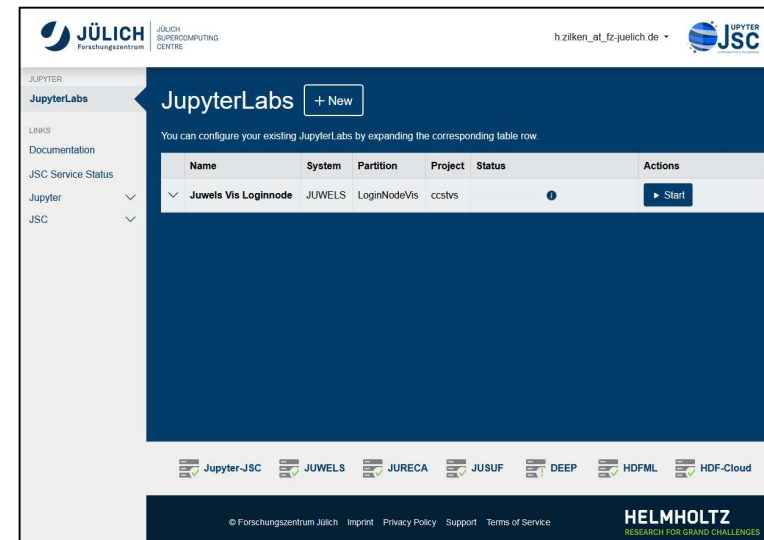
- **From JupyterLab@JSC**
<https://jupyter-jsc.fz-juelich.de>
 - See next slides
 - Very easy setup, no additional software needed (only browser)
 - Can be a little bit slow sometimes
- **Alternative: start Xpra session manually**
 - Can be a little bit faster than Xpra in browser
 - Xpra client needs to be installed on your local machine
 - Need to start Xpra on HPC system and locally by hand

Xpra Integration in JupyterLab@JSC

1. Go to <https://jupyter-jsc.fz-juelich.de> and login

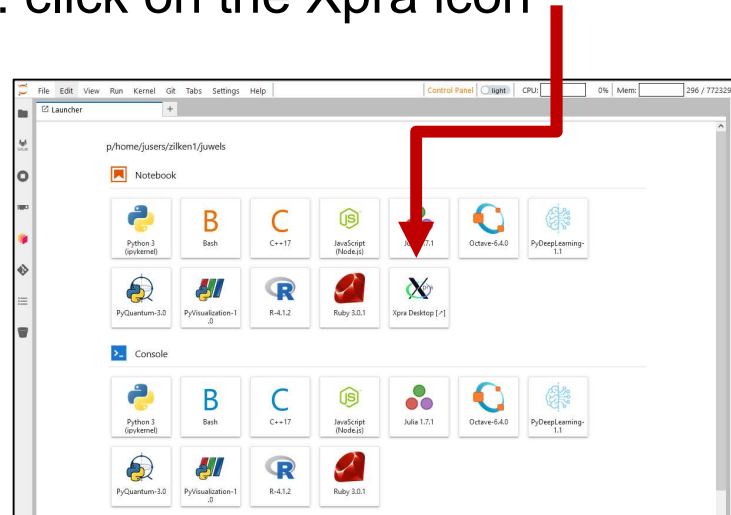


2. Add a new or start an existing JupyterLab on JURECA login node or JUWELS vis login node.

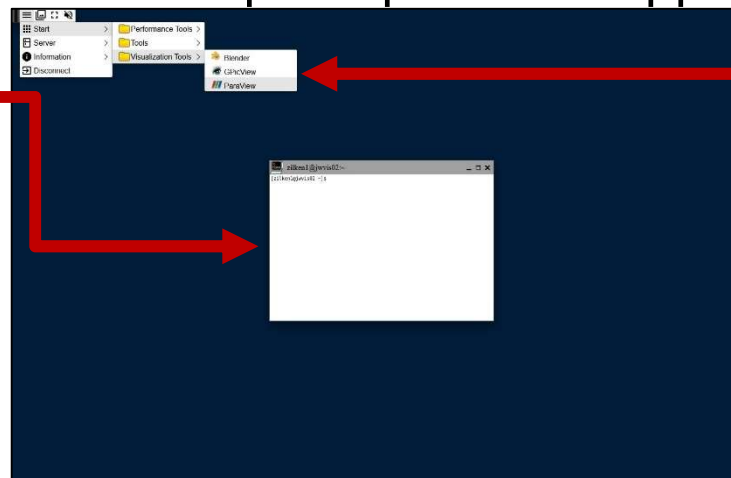


Xpra Integration in JupyterLab@JSC

3. If needed, start a new launcher by menu: File → New Launcher.
In the launcher: click on the Xpra icon



4. Wait for the HTML desktop of Xpra. Start apps from the menu or from the Xterm.

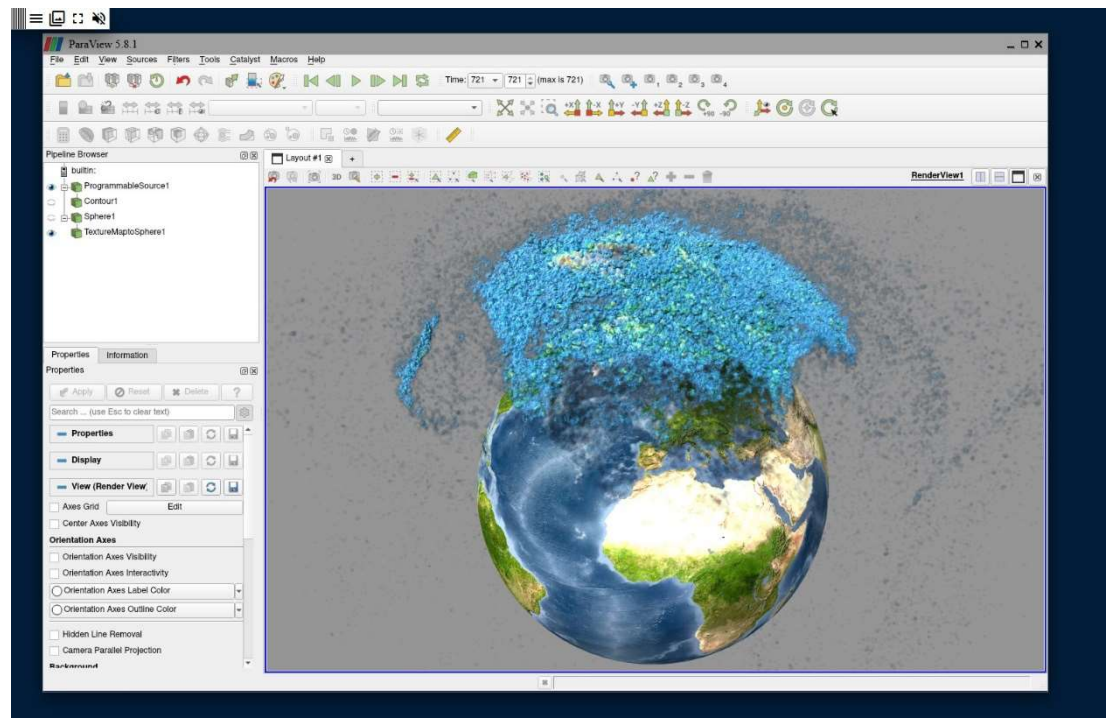


Xpra Integration in JupyterLab@JSC

5. Start ParaView in the Xpra environment in your browser, direct access to data stored on HPC filesystem

Use Xpra Menu or load modules

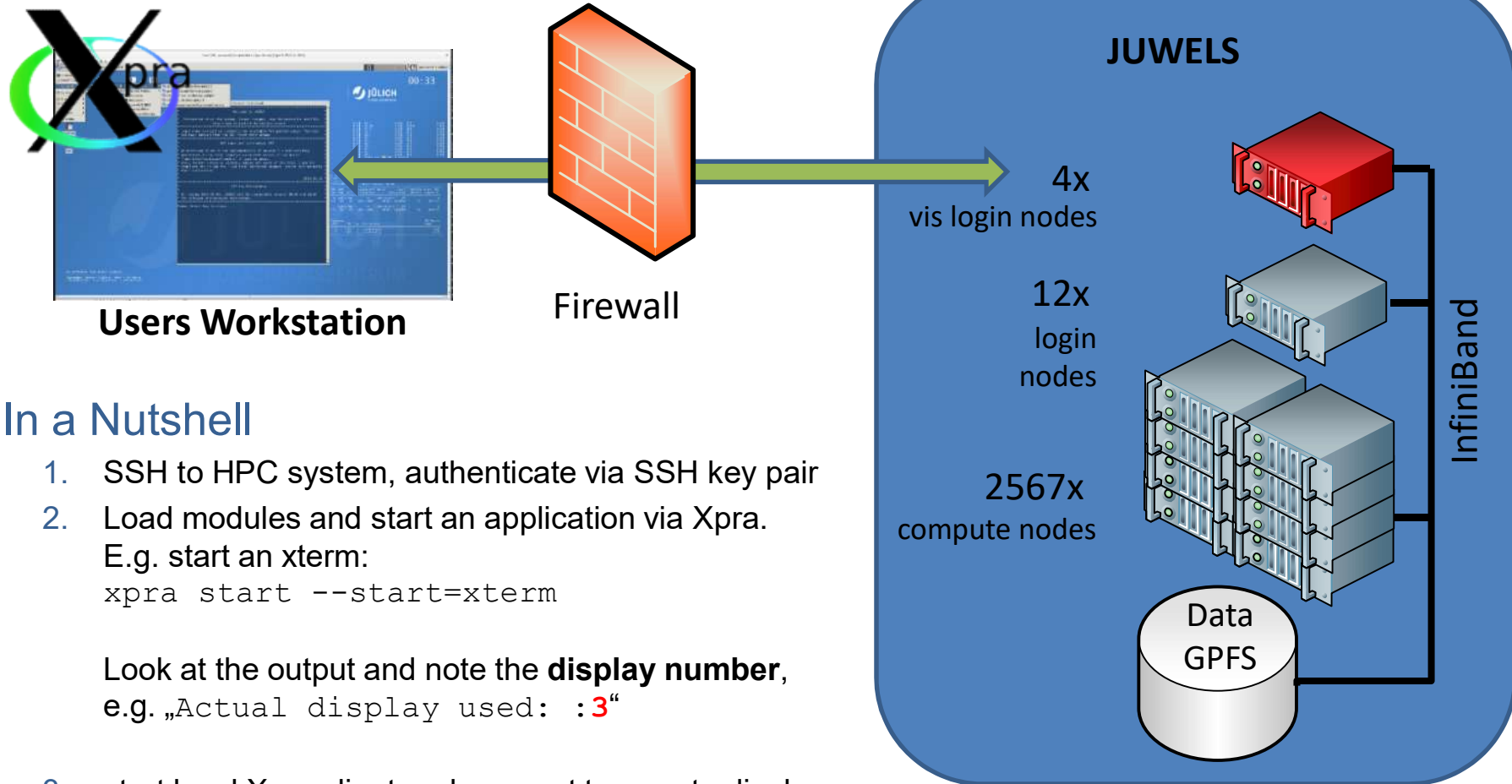
```
m1 Stages/2022 GCC/11.2.0 ParaStationMPI/5.5.0-1  
m1 ParaView/5.10.1
```



How to start Xpra manually

Manual Setup of Xpra

with Xpra + VirtualGL



In a Nutshell

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`

Manual Setup of Xpra

Step 1: store your private ssh key in a key manager

- Windows: use pageant from the PuTTY (<https://www.putty.org/>)
- Linux: start ssg-agent (if not already running)
ssh-add your private key

Step 2: login to a (visualization) login node

- **Linux:**
`ssh <USERID>@juwelsvis02.fz-juelich.de`
- **Windows:**
connect via a ssh client, e.g. PuTTY

Setup Xpra

Step 3: start xpra on HPC node and notice the display-number in the output

For example, start an xterm in Xpra:

```
jwvis02> ml Stages/2023 GCCcore/.11.3.0 xpra/4.4.3
```

```
jwvis02> xpra start --start=xterm
```

...

Actual display used: :3

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

Setup Xpra manually

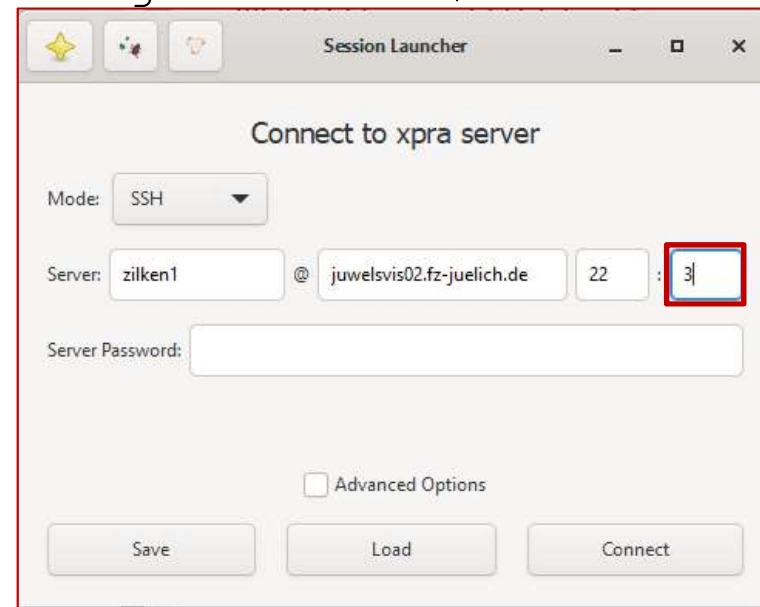
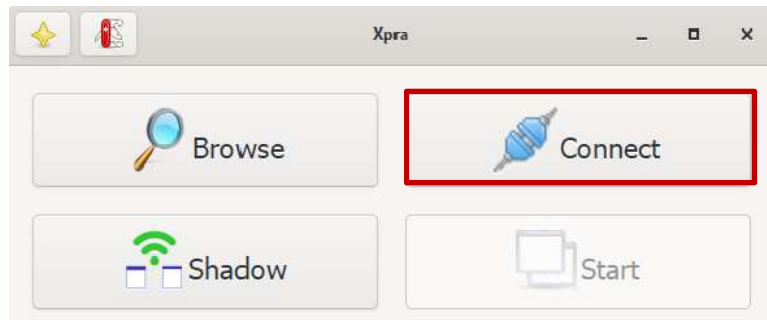
Step 4: connect to Xpra session

Install Xpra on your local machine. Download from <https://www.xpra.org/> (or just use your packagemanager in Linux)

Linux: use command (or GUI like Windows)

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

Windows: use Xpra GUI:

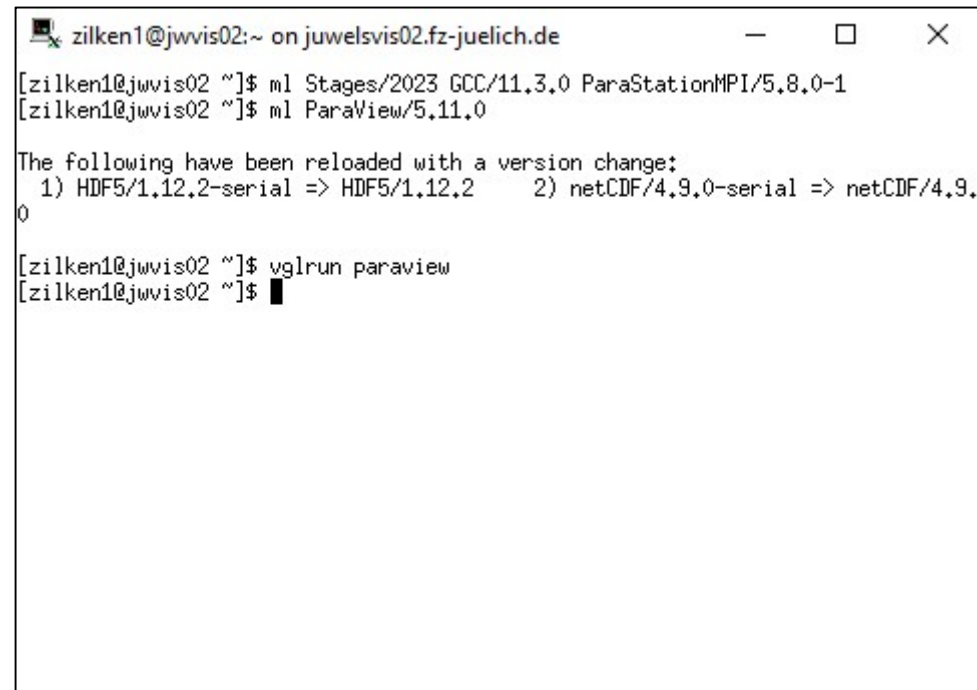


Setup Xpra

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 5.10.1:



```
zilken1@jwvis02:~ on juwelsvis02.fz-juelich.de
[zilken1@jwvis02 ~]$ m1 Stages/2023 GCC/11.3.0 ParaStationMPI/5.8.0-1
[zilken1@jwvis02 ~]$ m1 ParaView/5.11.0

The following have been reloaded with a version change:
 1) HDF5/1.12.2-serial => HDF5/1.12.2    2) netCDF/4.9.0-serial => netCDF/4.9.0
0

[zilken1@jwvis02 ~]$ vglrun paraview
[zilken1@jwvis02 ~]$
```

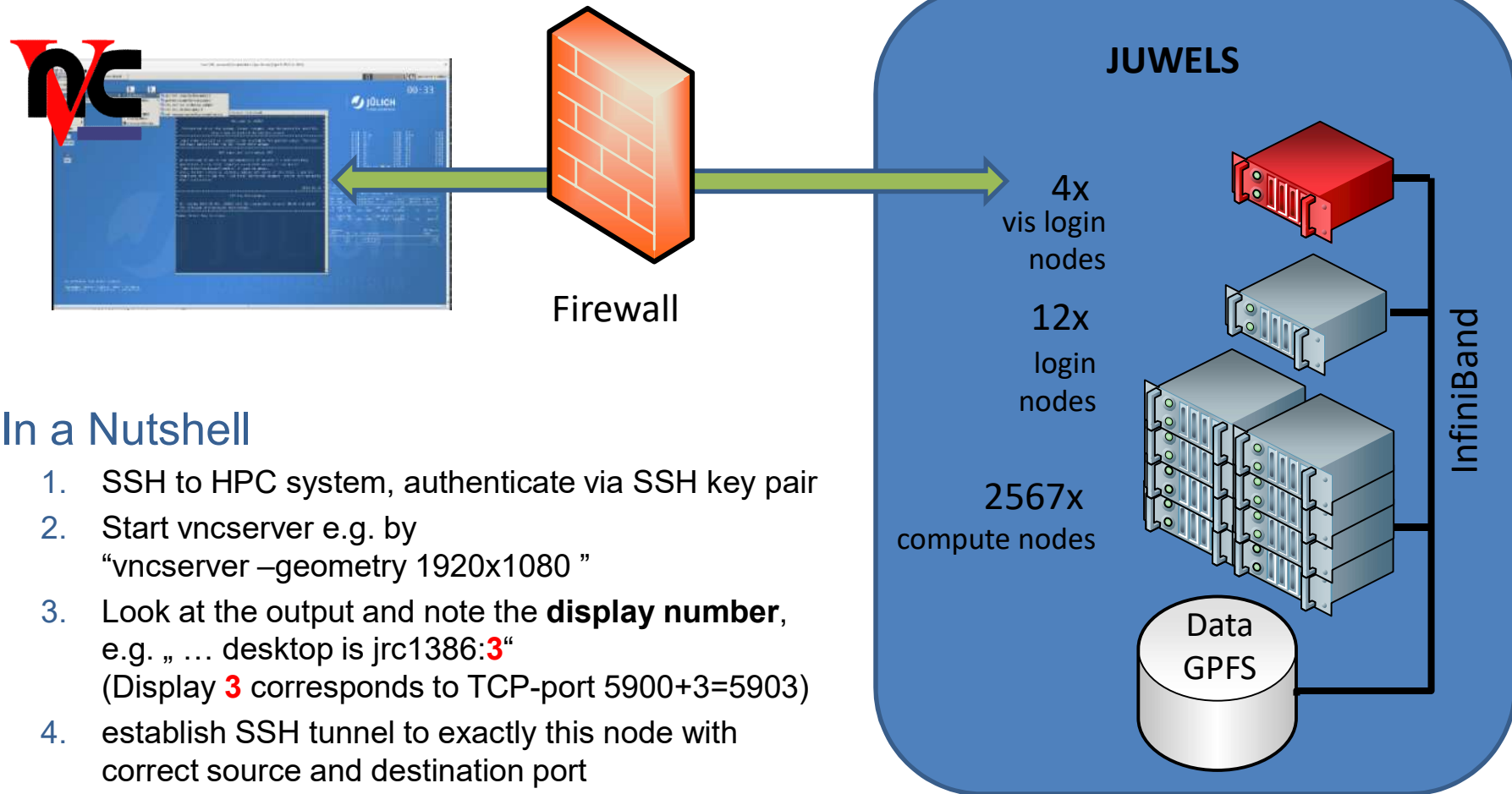
Step 5: When you are done, stop the session by

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

How to start a VNC session

Remote 3D Visualization

with VNC + VirtualGL



In a Nutshell

1. SSH to HPC system, authenticate via SSH key pair
2. Start vncserver e.g. by
"vncserver -geometry 1920x1080 "
3. Look at the output and note the **display number**, e.g. „ ... desktop is jrc1386:**3**“ (Display **3** corresponds to TCP-port 5900+3=5903)
4. establish SSH tunnel to exactly this node with correct source and destination port (5903 in the example above)
5. start local VNC client and connect to remote display

Setup VNC Connection

Preliminary step: **setup a VNC Password**
(needs only be done once)

- Login to a JUWELS or JURECA visualization node, create the directory `~/ .vnc` and define VNC password

- E.g.:

```
ssh <USERID>@juwelsvis.fz-juelich.de
```

```
mkdir ~/ .vnc
```

```
vncpasswd
```

Setup VNC Connection

Step 1: login to a specific visualization login node

- Hint: to establish a ssh tunnel, you need to connect to the same login node twice! Therefore:
**Don't use the „generic“ names (juwelsvis, jurecavis).
Instead select a specific node randomly**
(juwelsvis00 .. juwelsvis03, jureca01 .. Jureca12)
- **Linux:**
`ssh <USERID>@juwelsvis00.fz-juelich.de`
- **Windows:**
connect via a ssh client, e.g. PuTTY. The PuTTY ssh keyagent pageant may be usefull, too.

Setup VNC Connection

Step 2: start VNC-server on HPC node and locate the display-number in the output

Example:

```
vncserver -geometry 1920x1080
```

```
...
```

```
desktop is <node-name>:3
```

```
...
```

- The display-number is needed to establish the ssh tunnel (see step 3).
The VNC-server listens to TCP-port 5900+display-number (**5903** in the example)

Setup VNC Connection

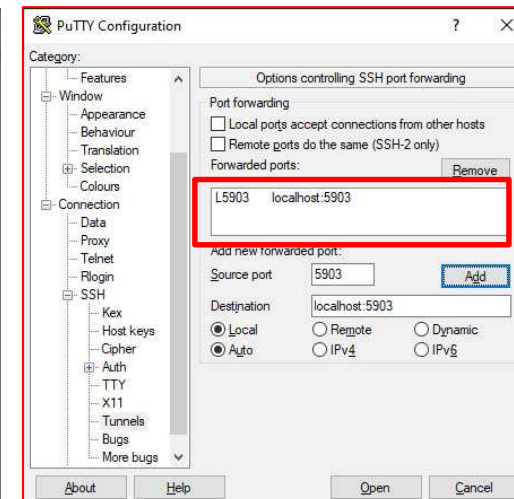
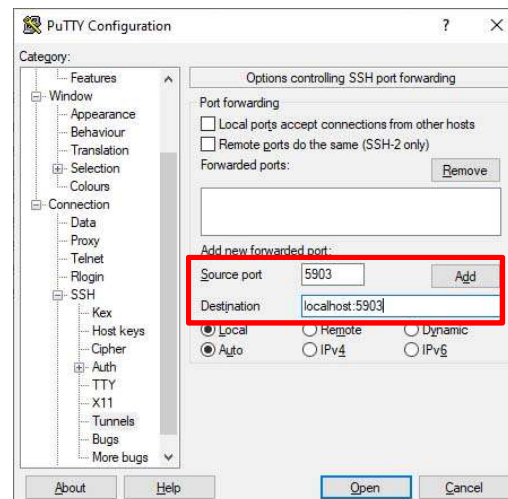
Step 3: establish the ssh tunnel

- Use the correct TCP port! Port must correspond to the display number (3 in this example)

- Linux:**

```
ssh -N -L 5903:localhost:5903
<USERID>@juwelsvis00.fz-juelich.de
```

- Windows:**
Use e.g. PuTTY to setup the tunnel



Setup VNC Connection

Step 4: start your local VNC viewer

Linux:

VNC viewer typically is already part of the Linux distribution or can be installed from a repository. Just start vncviewer with the correct display-number:

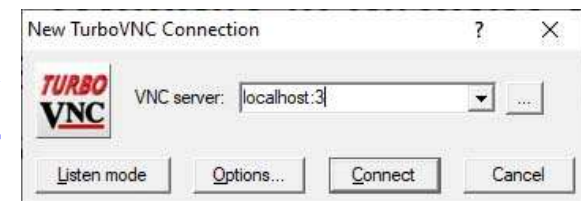
```
vncviewer localhost:3
```

Windows:

Download and install turboVNC:

<https://sourceforge.net/projects/turbovnc/>

```
Connect to localhost:3
```



ParaView

for data visualization

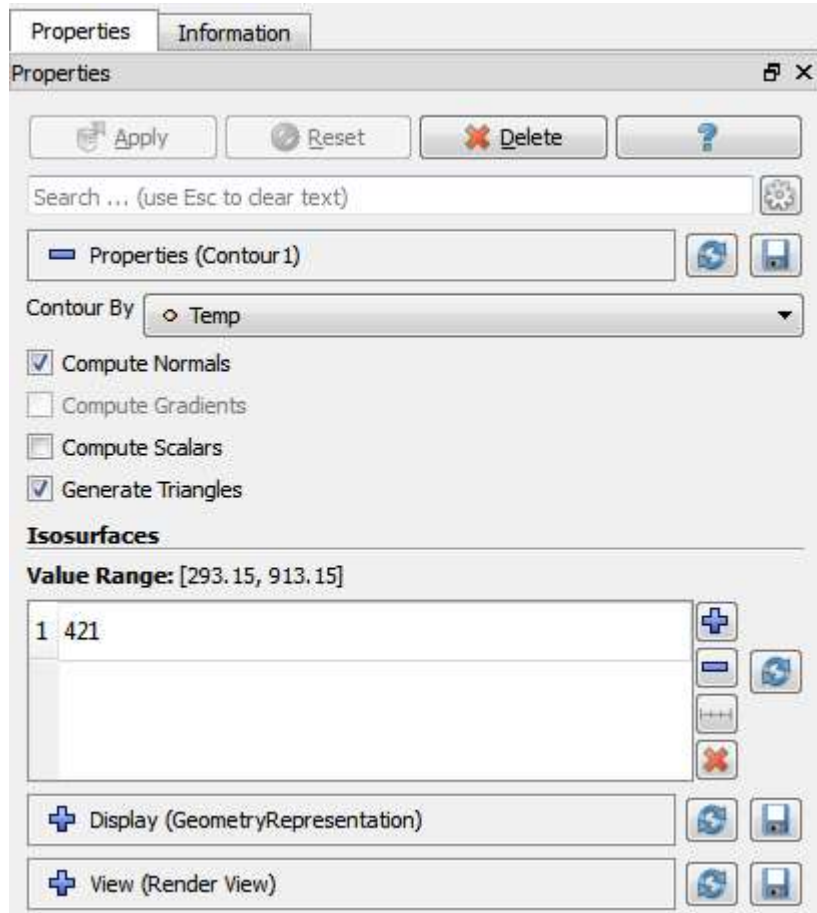
Exercise 1

- Login to `jupyter-jsc.fz-juelich.de`
- Start `xpra` or a `vncserver` and `paraview`
- Load some data, e.g.

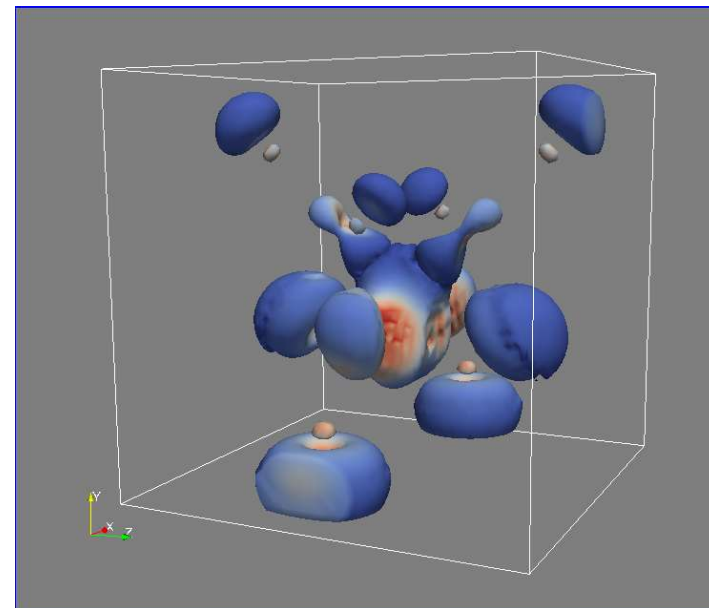
`/p/scratch/share/zilken1/ParaView_HandsOn/
headsq.vti`

- Lets have some fun with **filters**, see next slides

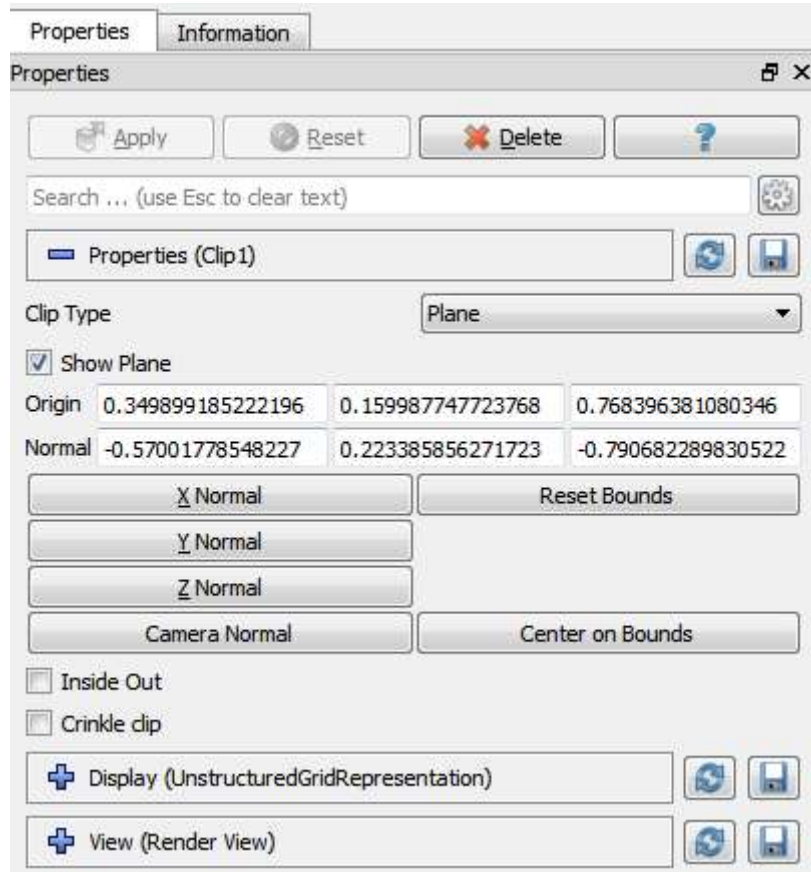
Common Filters: Contour



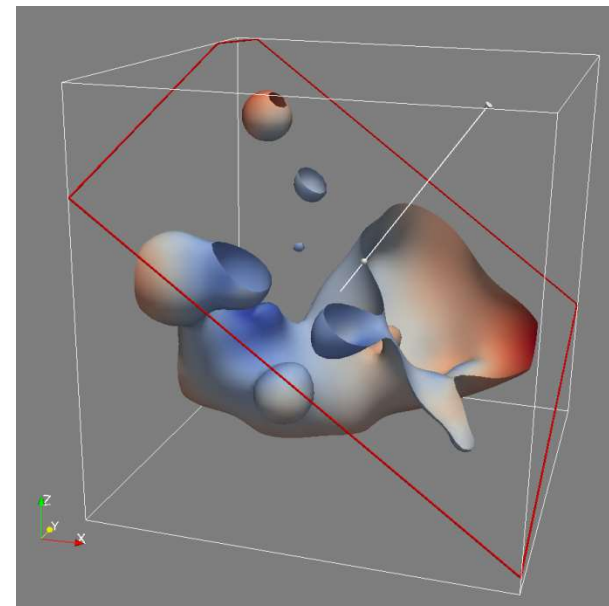
- Extracts the points, curves, or surfaces where a scalar field is equal to a user-defined value.
- This surface is often also called an isosurface



Common Filters: Clip

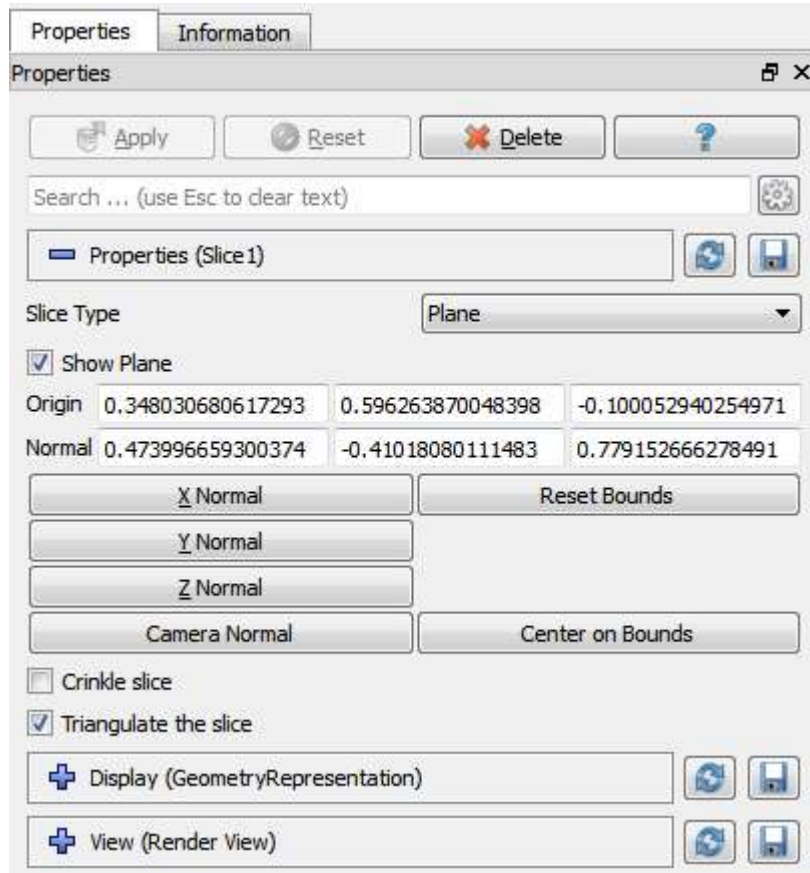


- Intersects the geometry with a user-defined plane, box or sphere
- Removes all the geometry on one side of this plane (box, sphere)

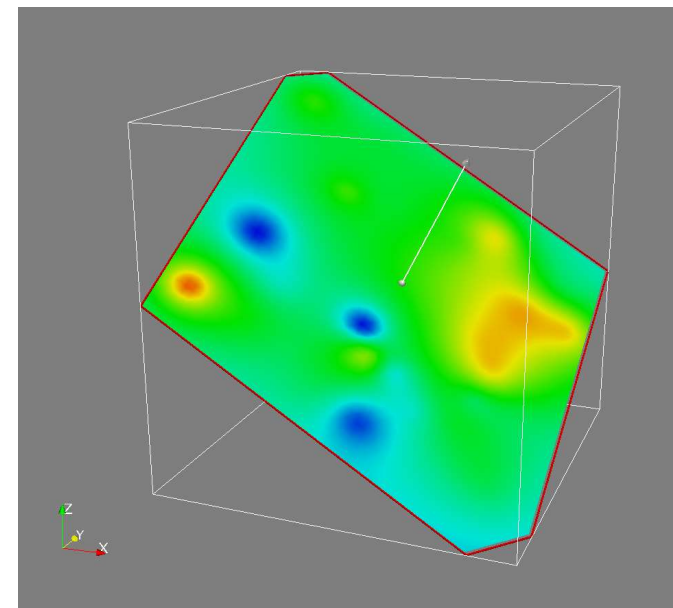


Beware of data explosion:
Structured data is converted to unstructured!

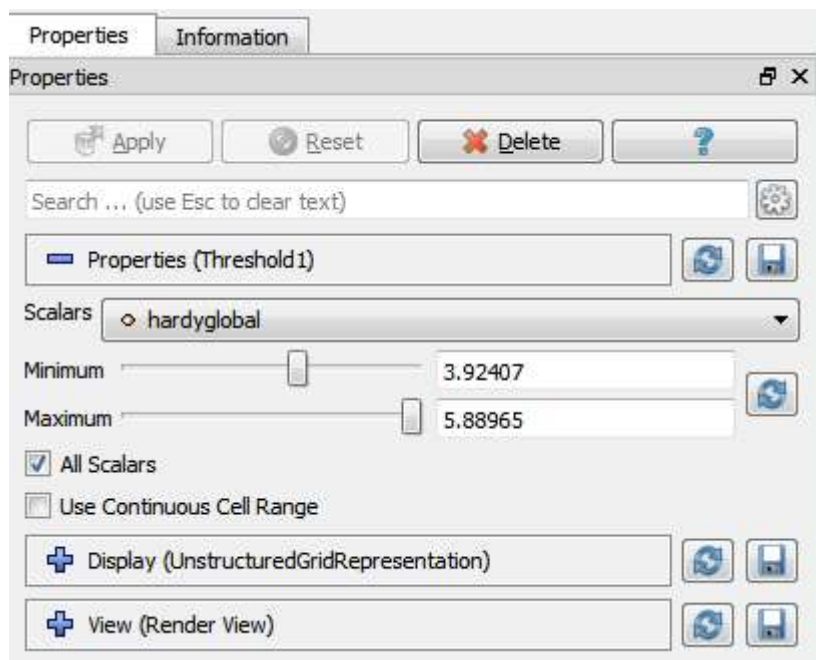
Common Filters: Slice



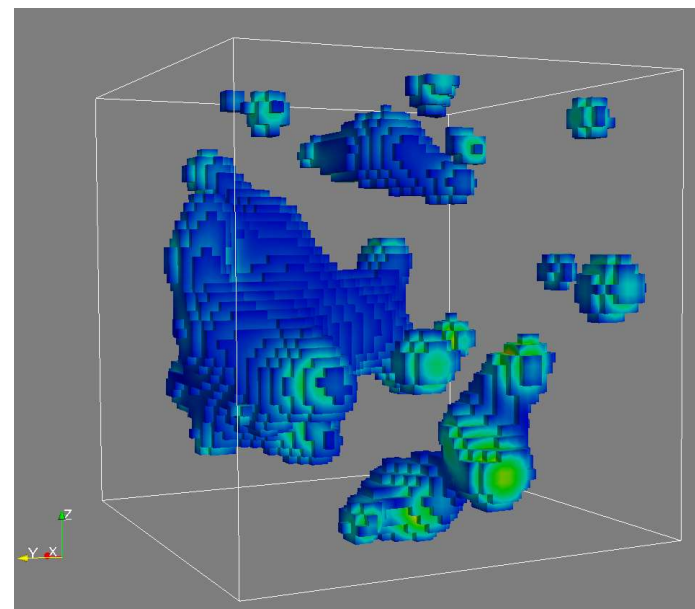
- Intersects the geometry with a plane, box, sphere or cylinder
- Similar to clipping, except that all that remains is the geometry where the plane is located.



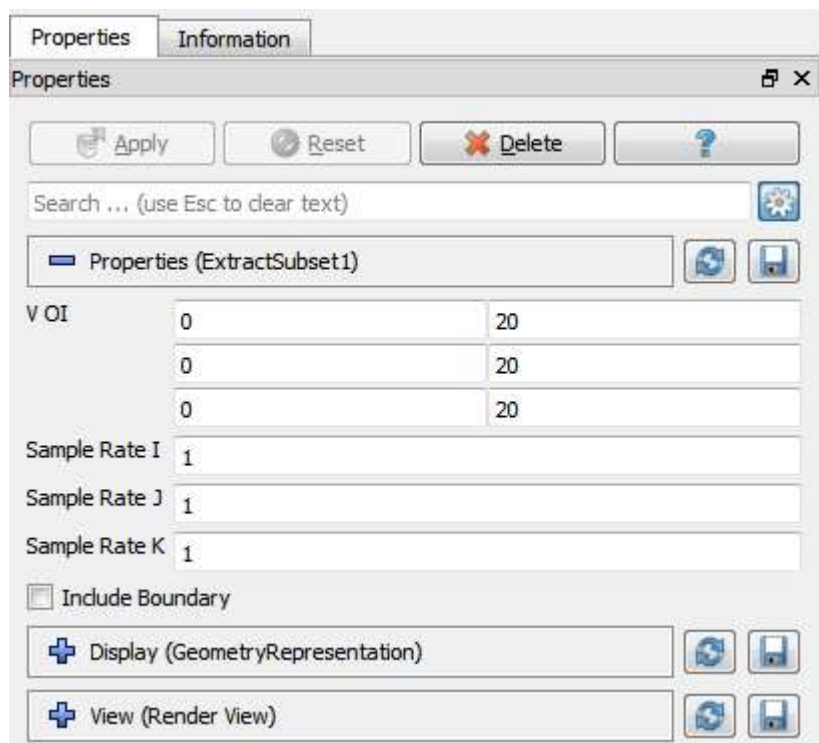
Common Filters: Threshold



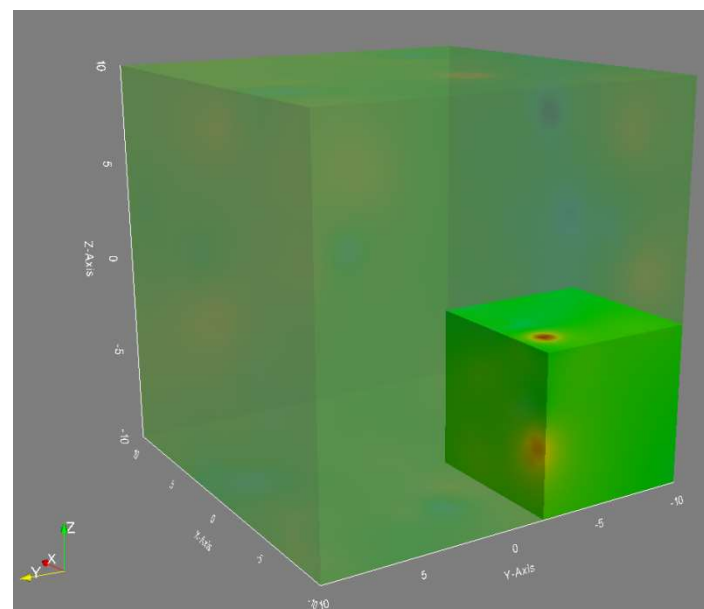
- Extracts cells that lie within a specified range of a scalar field



Common Filters: Extract Subset



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



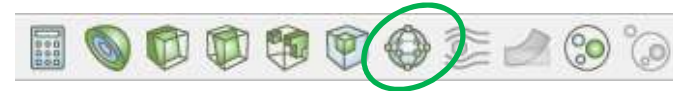
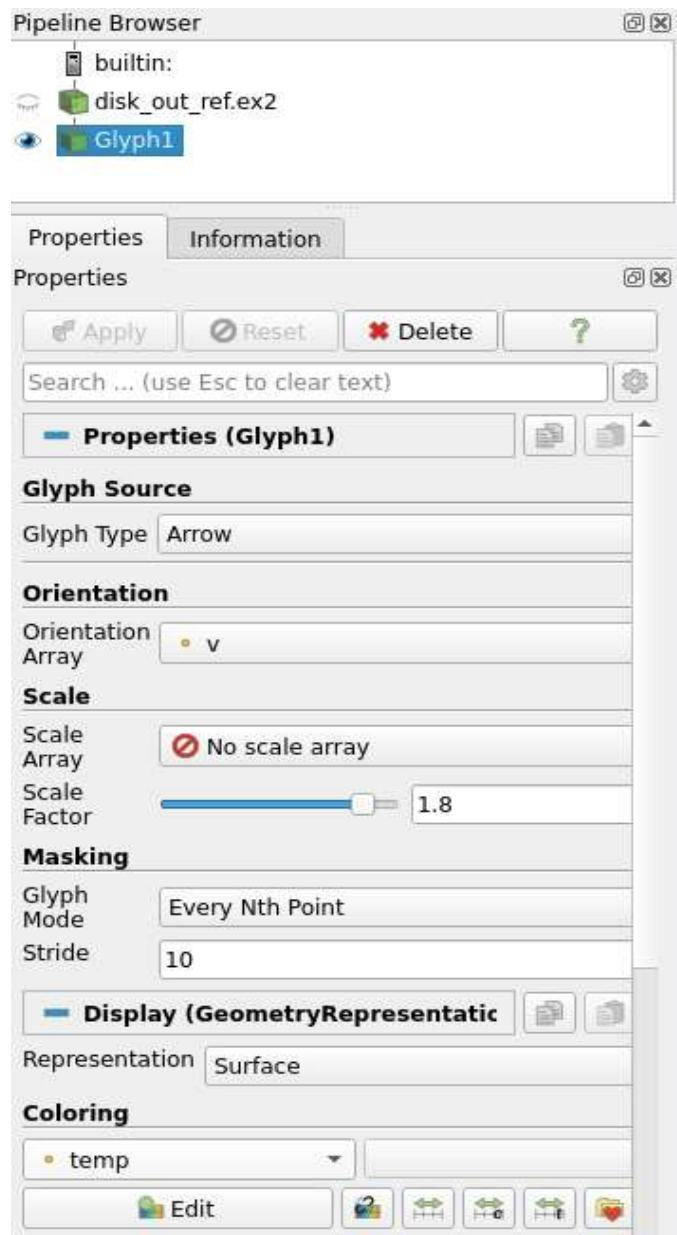
Exercise 2

- Load

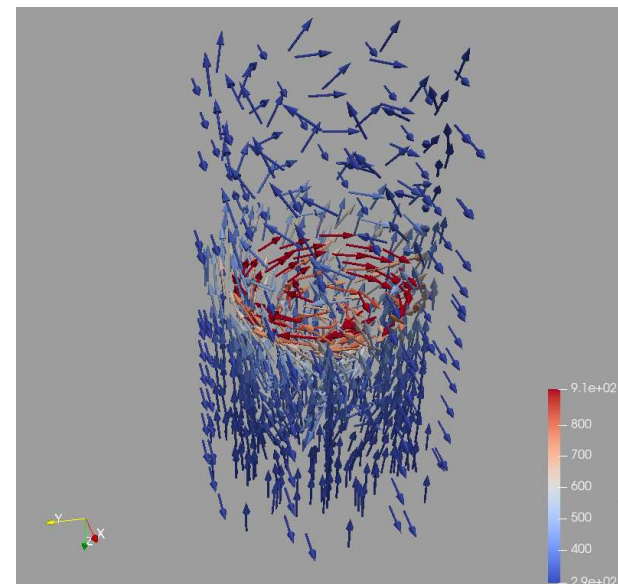
`/p/scratch/share/zilken1/ParaView_HandsOn/
disk_out_ref2.ex2`

- Lets have some fun with filters for **vector-data**, see next slides

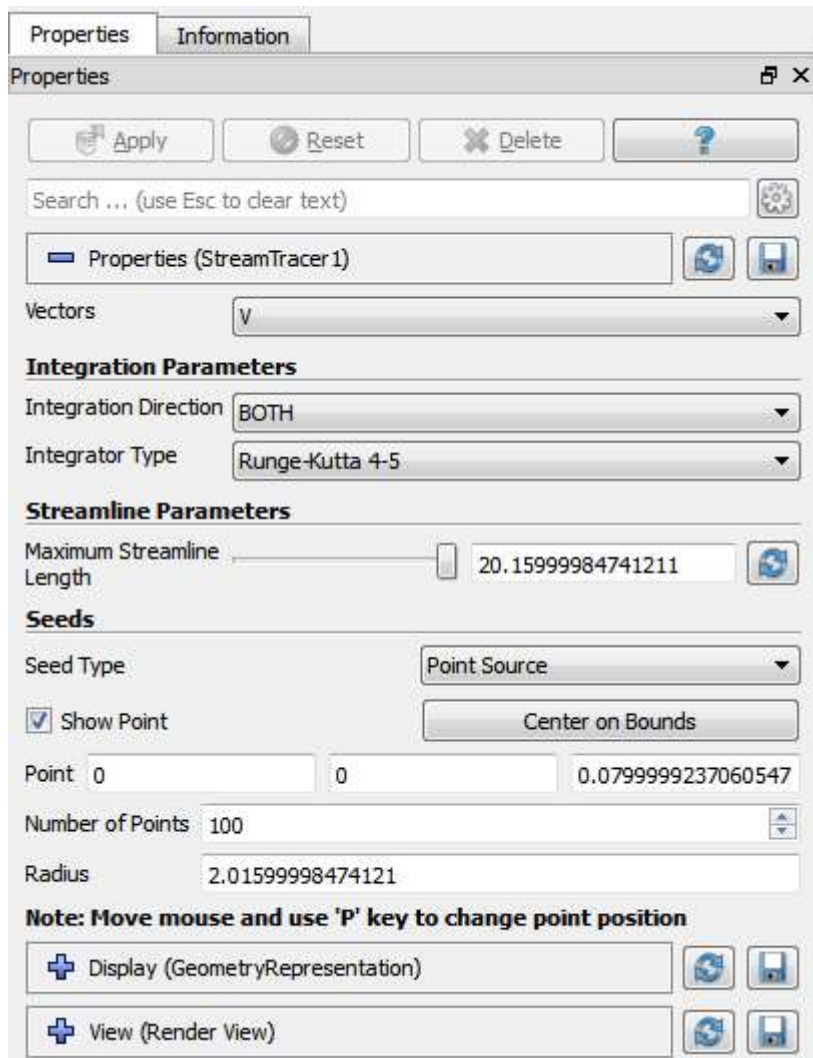
Common Filters: Glyph



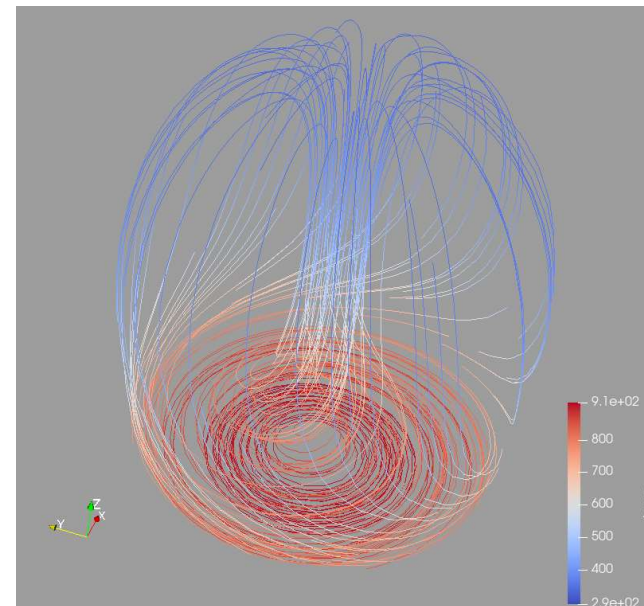
- Places a glyph, a simple shape, on each point (or subset) in a mesh
- glyphs may be oriented by a vector and scaled by a vector or scalar.



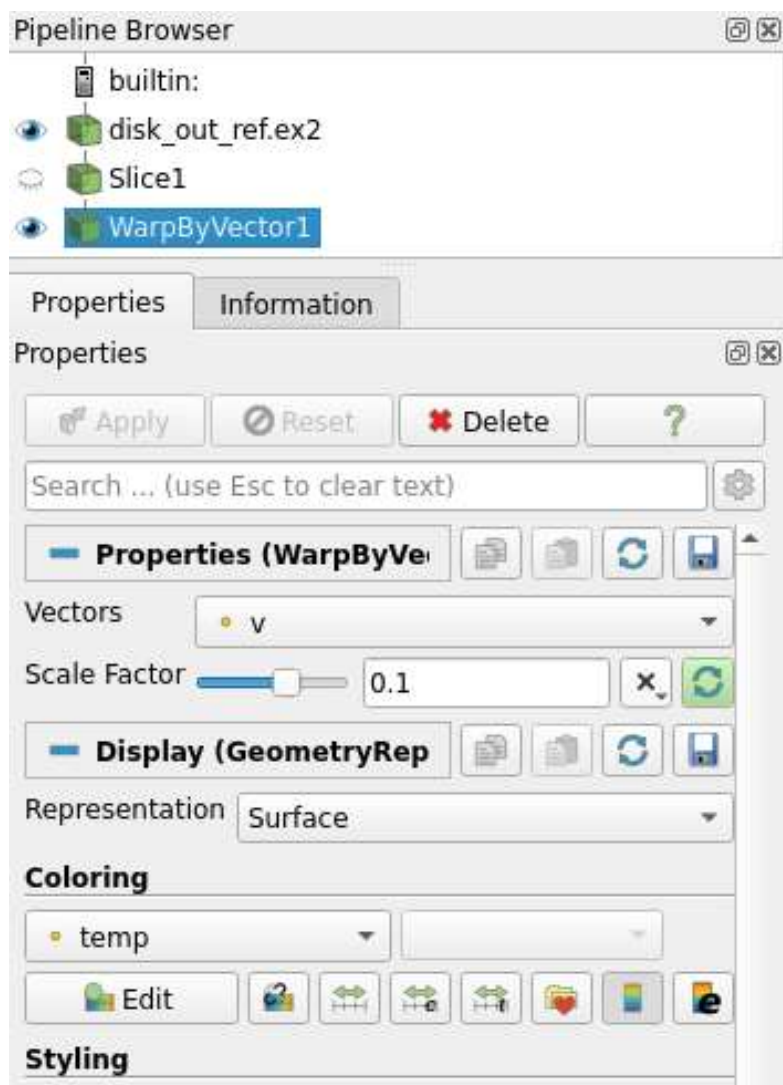
Common Filters: Stream Tracer



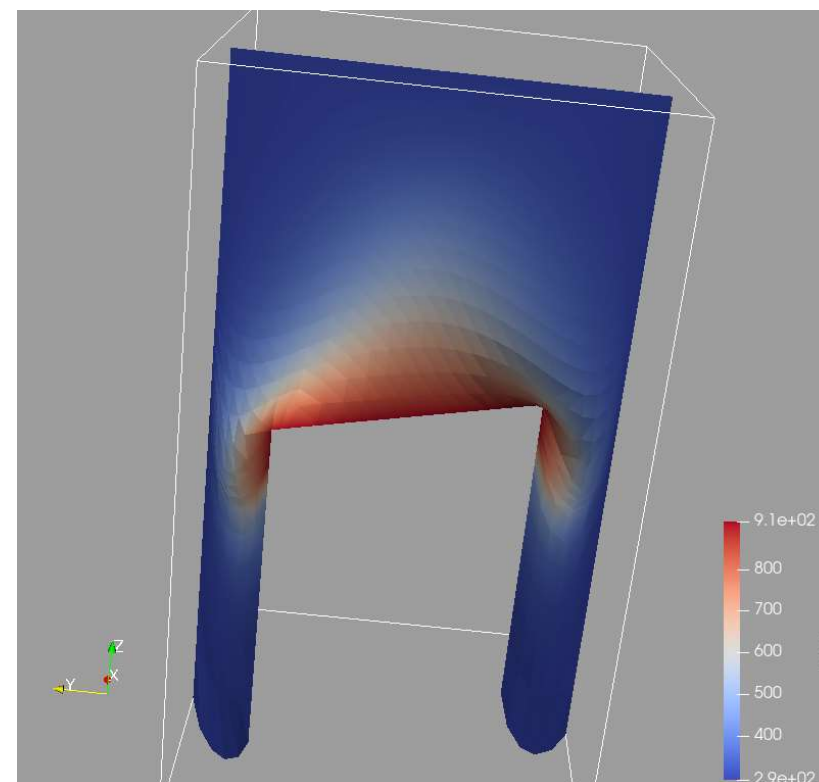
- Seeds a vector field with points and then traces those seed points through the (steady state) vector field.



Common Filters: Warp (vector)



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

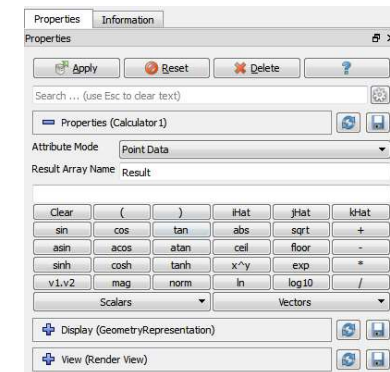


Calculations within ParaView

Calculator: calculates new attributes based on simple expression

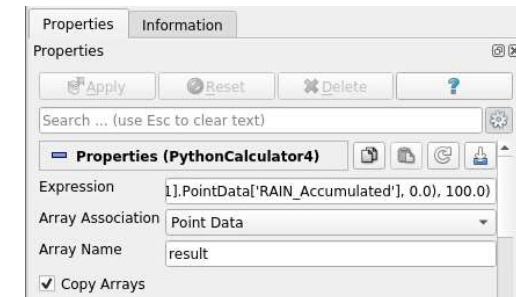


- example: „LANDMASK*(abs(HGT) + 20.0)“
- Can generate vectors from scalars via „iHat*velocity_x + jHat*velocity_y + kHat*velocity_z“
- Can generate new coordinates
- Unflexibel, no „if“ statement



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))“

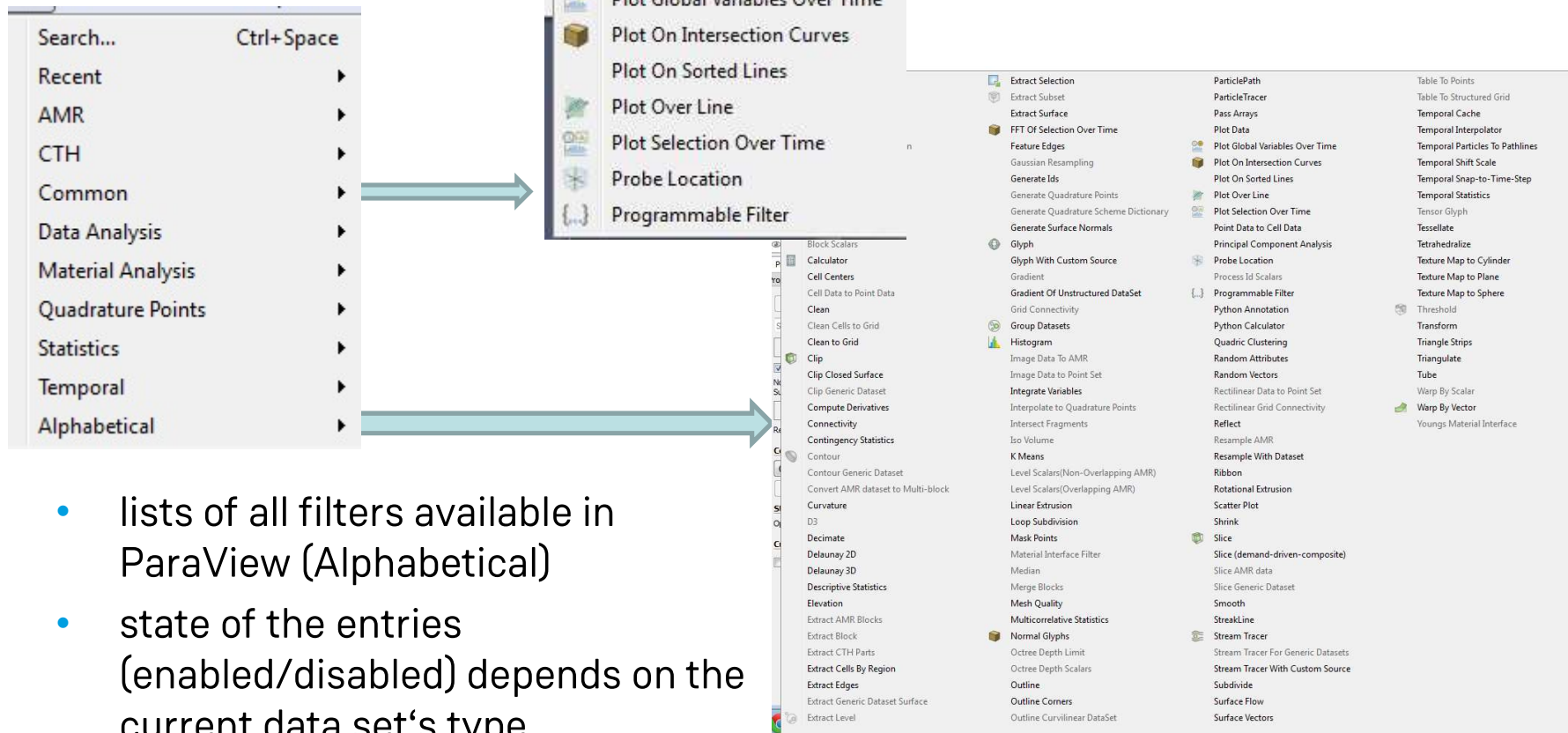


Programmable Source/Filter

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

Filter Menu:

Many more filters in the Filters Menu



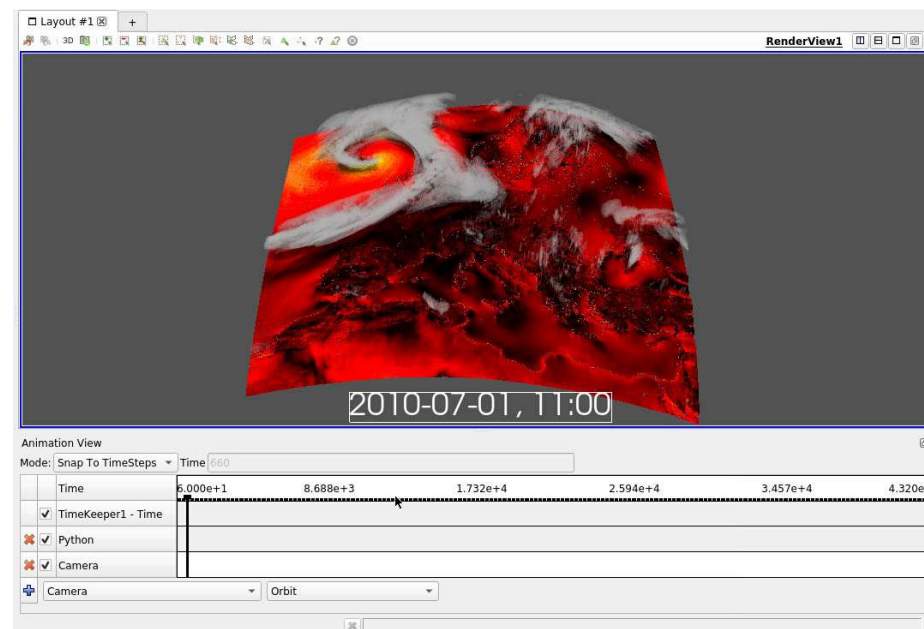
The image shows a screenshot of the ParaView software interface. On the left, the 'Filter' menu is open, displaying a search bar and a list of filter categories: Recent, AMR, CTH, Common, Data Analysis, Material Analysis, Quadrature Points, Statistics, Temporal, and Alphabetical. Two arrows point from the 'Common' and 'Alphabetical' categories to a larger, detailed view of the filter menu on the right. This detailed view lists numerous filters, including: Calculator, Extract Selection, Histogram, Integrate Variables, Plot Data, Plot Global Variables Over Time, Plot On Intersection Curves, Plot On Sorted Lines, Plot Over Line, Plot Selection Over Time, Probe Location, Programmable Filter, Block Scalars, Cell Centers, Clean, Clip, Clip Closed Surface, Compute Derivatives, Connectivity, Contour, Curvature, D3, Decimate, Delaunay 2D, Delaunay 3D, Descriptive Statistics, Elevation, Extract AMR Blocks, Extract Block, Extract CTH Parts, Extract Cells By Region, Extract Edges, Extract Generic Dataset Surface, Extract Level, FFT Of Selection Over Time, Feature Edges, Gaussian Resampling, Generate Ids, Generate Quadrature Points, Generate Quadrature Scheme Dictionary, Generate Surface Normals, Glyph, Glyph With Custom Source, Gradient, Gradient Of Unstructured DataSet, Grid Connectivity, Group Datasets, Image Data To AMR, Image Data To Point Set, Integrate Variables, Interpolate to Quadrature Points, Intersect Fragments, Iso Volume, K Means, Level Scalars(Non-Overlapping AMR), Level Scalars(Overlapping AMR), Linear Extrusion, Loop Subdivision, Mask Points, Material Interface Filter, Median, Merge Blocks, Mesh Quality, Multicorrelative Statistics, Normal Glyphs, Octree Depth Limit, Octree Depth Scalars, Outline, Outline Corners, Outline Curvilinear DataSet, ParticlePath, ParticleTracer, Pass Arrays, Plot Data, Plot Global Variables Over Time, Plot On Intersection Curves, Plot On Sorted Lines, Plot Over Line, Plot Selection Over Time, Point Data to Cell Data, Principal Component Analysis, Probe Location, Process Id Scalars, Programmable Filter, Python Annotation, Python Calculator, Quadric Clustering, Random Attributes, Random Vectors, Rectilinear Data to Point Set, Rectilinear Grid Connectivity, Reflect, Resample AMR, Resample With Dataset, Ribbon, Rotational Extrusion, Scatter Plot, Shrink, Slice, Slice (demand-driven-composite), Slice AMR data, Slice Generic Dataset, Smooth, StreamLine, Stream Tracer, Stream Tracer For Generic Datasets, Stream Tracer With Custom Source, Subdivide, Surface Flow, Surface Vectors, Table To Points, Table To Structured Grid, Temporal Cache, Temporal Interpolator, Temporal Particles To Pathlines, Temporal Shift Scale, Temporal Snap-to-Time-Step, Temporal Statistics, Tensor Glyph, Tessellate, Tetrahedralize, Texture Map to Cylinder, Texture Map to Plane, Texture Map to Sphere, Threshold, Transform, Triangle Strips, Triangulate, Tube, Warp By Scalar, Warp By Vector, and Youngs Material Interface.

- 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

Animating Data

Using the Animation View, ParaView can animate

- Data time steps (if you have time-dependent data)
- 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



Exercise 3

- Load

`/p/scratch/share/zilken1/ParaView_HandsOn/
can.ex2`

- Lets have some fun with **animations**