Mitglied der Helmholtz-Gemeinschaft

Remote Visualization at JSC (with ParaView)

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

Herwig Zilken, 2024/11/14

Visualization at JSC

Algorithms, Tools and Methods Lab Visualization & interactive HPC

- **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) or presentations
- **Interactive HPC**
  - Jupyter@JSC
  - Workflows

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

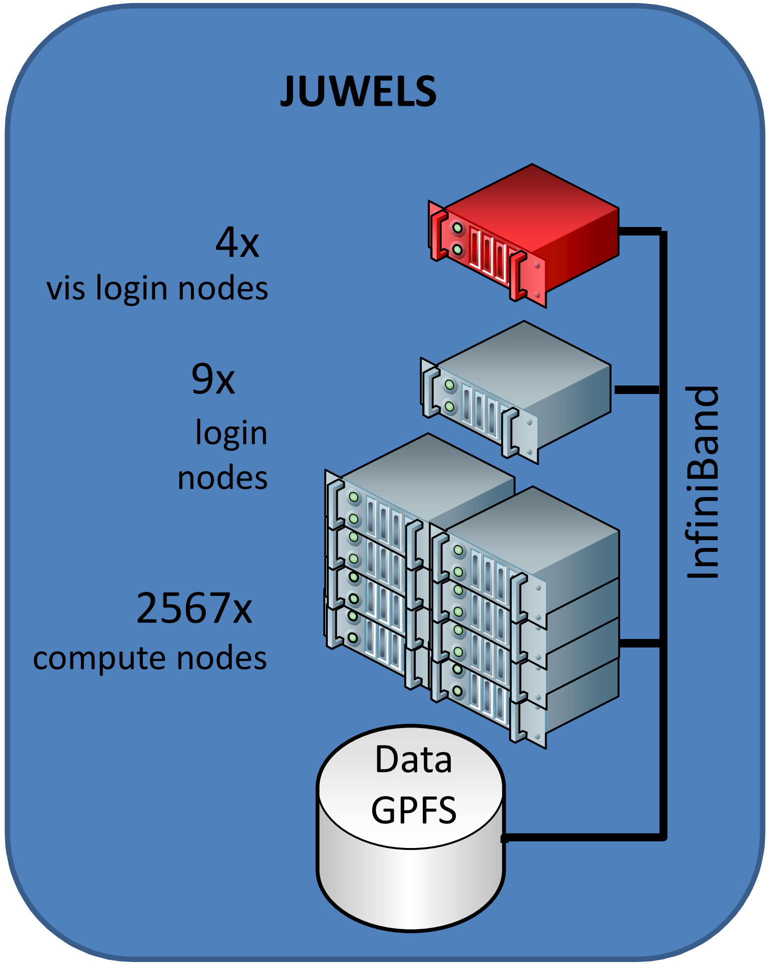
- juwels-cluster

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

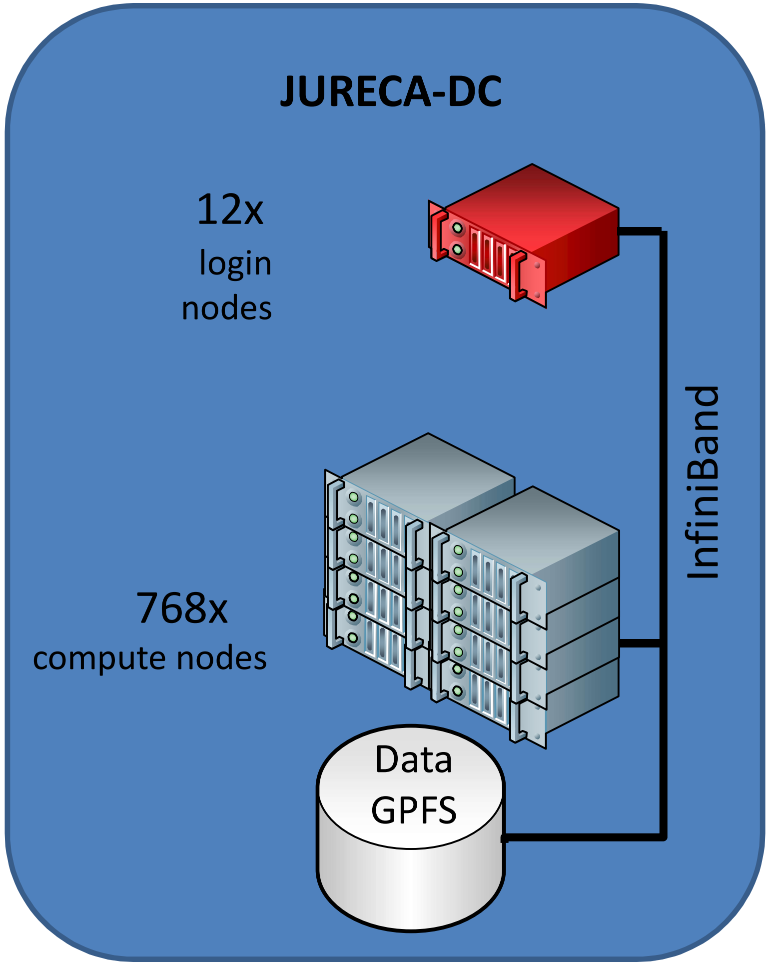
JURECA-DC: closer look at login nodes

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

Ascent, Conduit

VisIt

Other Visualization Packages (more packages on user demand):

Blender, GPicView, VTK, VMD

Remote 3D Visualization
at Jülich Supercomputing Centre

Bad

- X forwarding (“ssh –X”) + indirect Rendering
  slow, maybe incompatible → bad idea

Medium good

- “intrinsic remote capable” visualization apps
  application dependent, error-prone setup

Our recommendation

- Remote Windows or Desktop
  - Xpra - stream application content with H.264 + VirtualGL
    - Can be started with a single click in Jupyter@JSC
      For JURECA, Jupyterlab approach is buggy at the moment, don’t use until bug is fixed → For now start manually on JURECA
  - VNC + VirtualGL can be used also
    - Simple, but full featured remote desktop
    - Does not be started from Jupyter@JSC

Remote 3D Visualization

with Xpra or 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

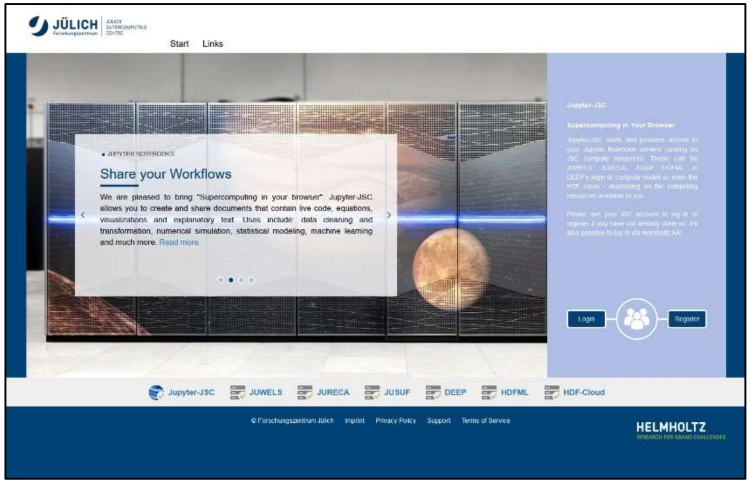
JÜLICH Forschungszentrum

Two ways to start an Xpra session:

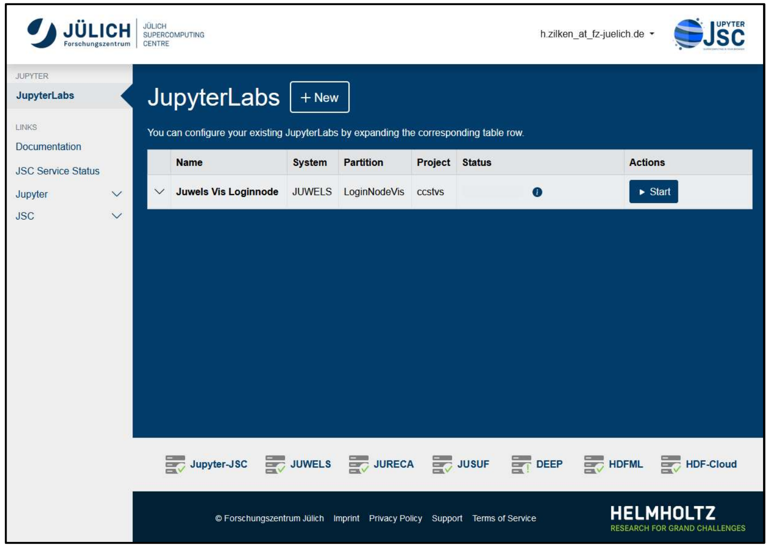
- **From JupyterLab@JSC**
  https://jupyter-jsc.fz-juelich.de
  - Very easy setup, no additional software needed (only browser, but use Chrome, not Firefox!)
  - Can be a little bit slow sometimes
- **Start Xpra session manually**
  - Can be a little bit faster than Xpra in browser
  - Xpra client needs 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

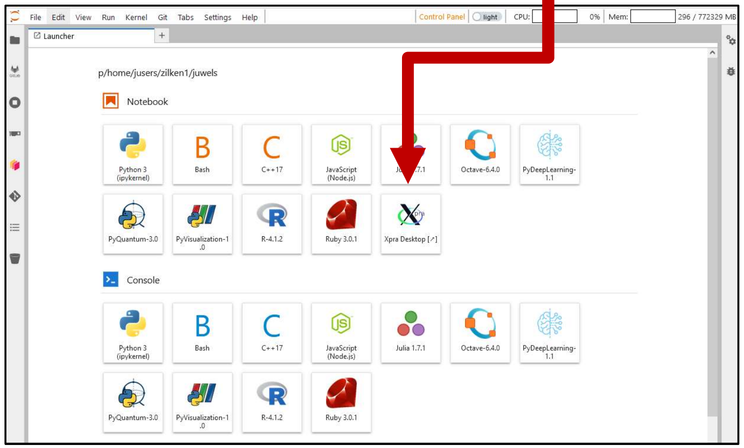


2. Add a new or start an existing JuperLab on JUWELS vis login node or JURECA.

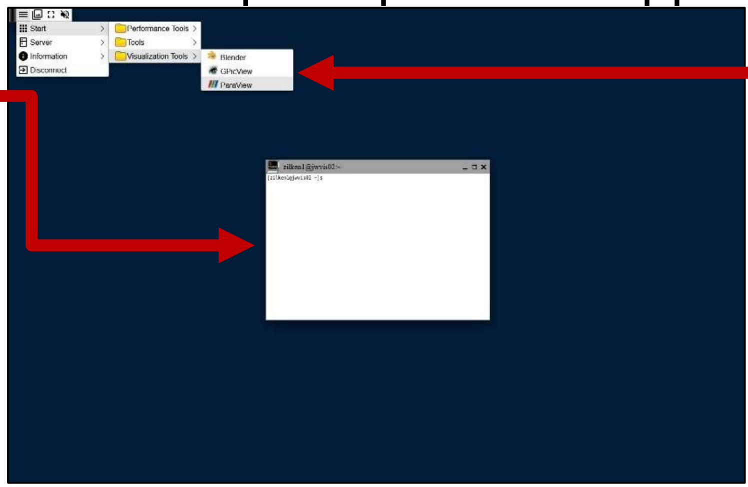


Xpra Integration in JupyterLab@JSC

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



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

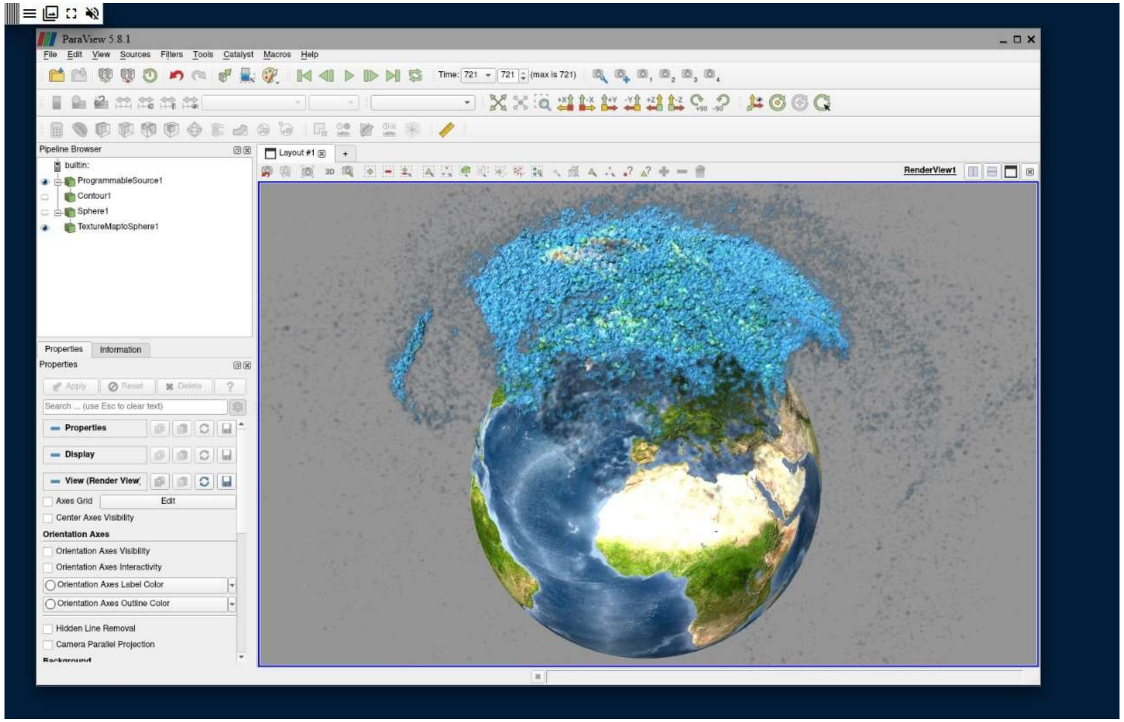


Xpra Integration in JupyterLab@JSC

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

Use **Xpra Menue** or load modules

```
ml Stages/2024 GCC/12.3.0 ParaStationMPI/5.9.2-1
ml ParaView/5.12.0-RC2
```



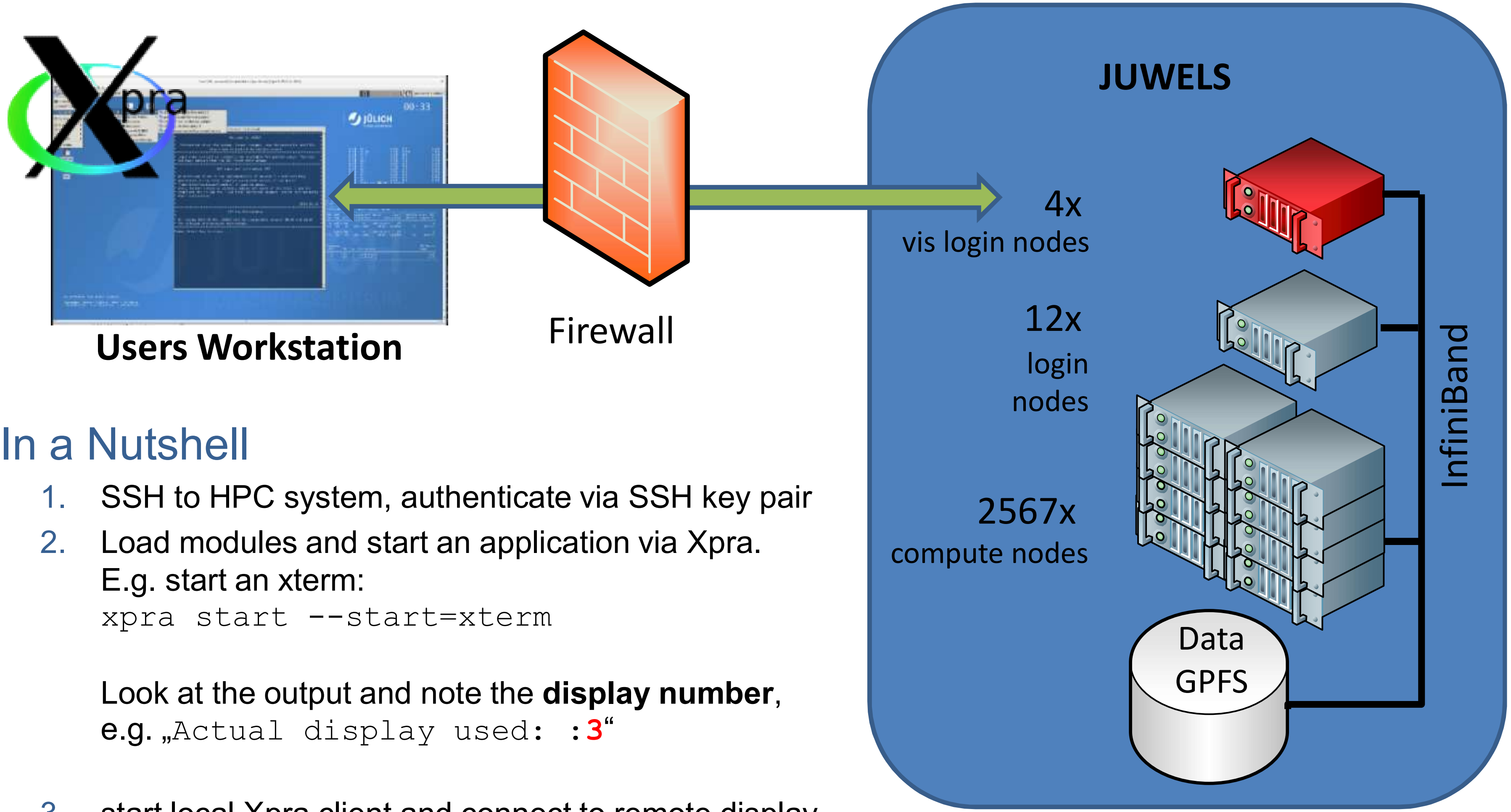
Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum

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,
e.g. juwelsvis02

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

Manual Setup of Xpra

Step 3: load Xpra modules, check for taken display numbers and choose a free one

```
jwvis02> ml Stages/2024 GCCcore/.12.3.0 xpra/5.0.8
jwvis02> xpra list # show taken display numbers
Found the following xpra sessions:
/tmp:
        INACCESSIBLE session at :0
        INACCESSIBLE session at :1
        INACCESSIBLE session at :2
```

Notice, that e.g. :3 is not taken. Try to start Xpra with a free display number:

```
jwvis02> xpra start :3 --start="xterm -xrm xterm*font:10x20"
...
Actual display used: :3
```

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

Manual Setup of Xpra

Step 4: check again if Xpra is running properly on this display number

```
jwvis02> xpra list
Found the following xpra sessions:
/tmp:
        INACCESSIBLE session at :0
        INACCESSIBLE session at :1
        INACCESSIBLE session at :2
        LIVE session at :3
```

- The display-number (here :3) is needed to connect to the Xpra session

Manual Setup of Xpra

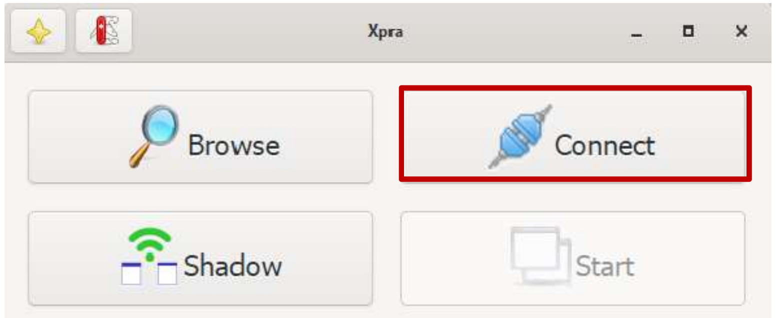
Step 5: connect to Xpra session

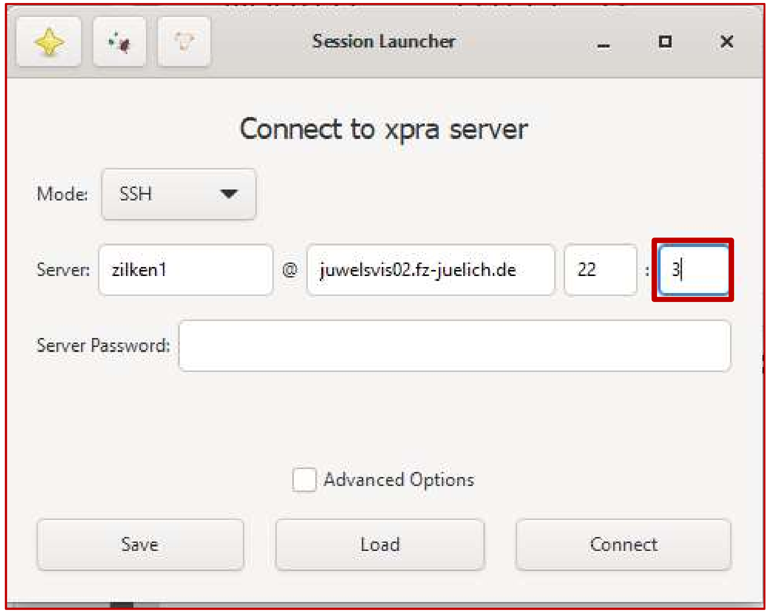
Install Xpra on your local machine. Download from https://www.xpra.org/ (or just use your packagemanager in Linux)

Linux: use command (or GUI like Windows)

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

Windows/Linux:
use Xpra GUI:





Manual Setup of Xpra

Step 6: start visualization application

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

Start your application there, e.g. ParaView 5.12.0-RC2:

```
jwvis02> ml Stages/2024 GCC/12.3.0
ParaStationMPI/5.9.2-1 ParaView/5.12.0-RC2
```

Step 7: When you are done, stop the session by

```
jwvis02> xpra stop :3 #‘xpra stop‘ works as well
```

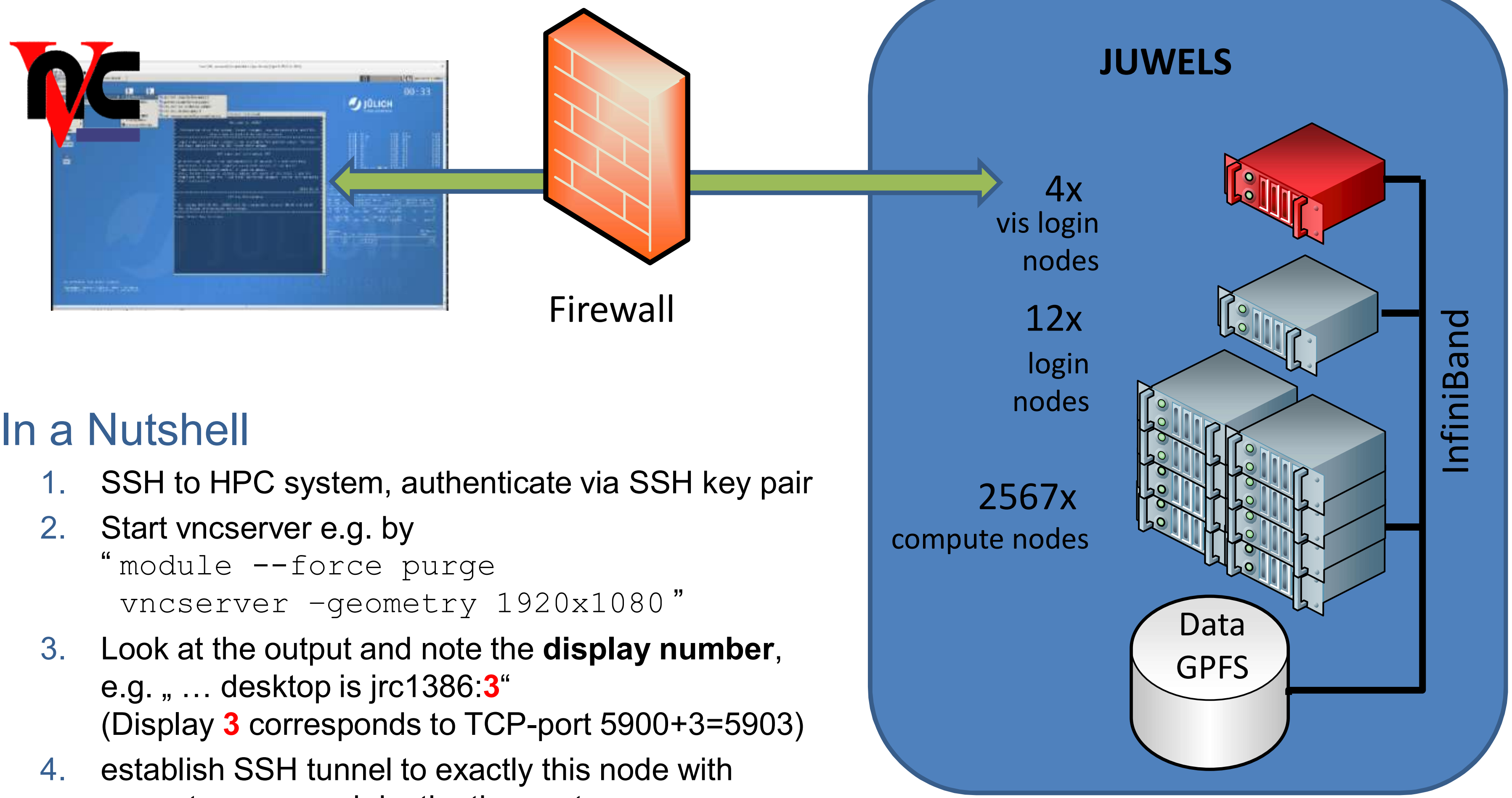
Mitglied der Helmholtz-Gemeinschaft

JÜLICH Forschungszentrum

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
   “`module --force purge`
   `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:

```
module --force purge
/opt/TurboVNC/bin/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

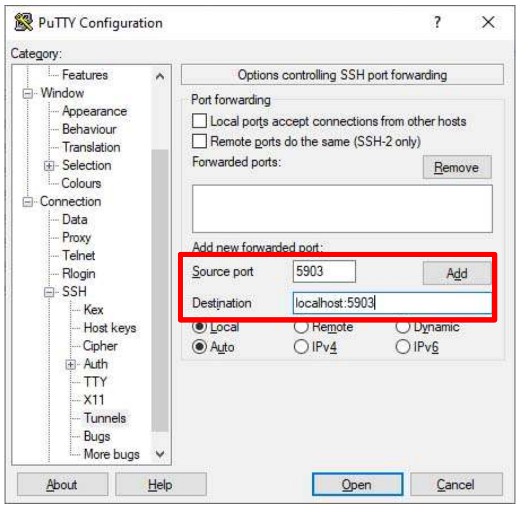
Step 3: establish the ssh tunnel

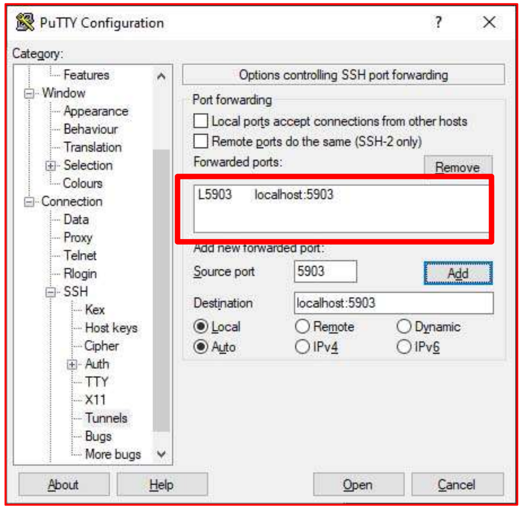
- Use the correct TCP port! Port must correspond to the display number (3 in this example)

- **Linux:**

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

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





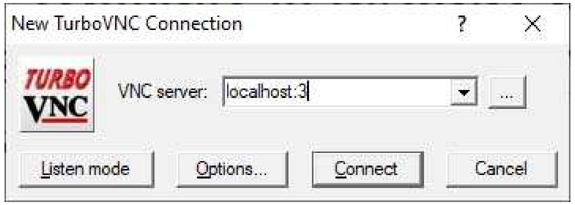
Setup VNC Connection

Step 4: start your local VNC viewer

Linux:
VNC viewer typically is already part of many Linux distributions 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



Start ParaView

Open a Terminal and execute:

```
ml Stages/2024 GCC/12.2.0 ParaStationMPI/5.9.2-1
ml ParaView/5.12.0-RC2
vglrun paraview
```

To stop the session:

```
/opt/TurboVNC/bin/vncserver -kill :3
```

Mitglied der Helmholtz-Gemeinschaft

JÜLICH Forschungszentrum

ParaView

for data visualization

Exercise 1

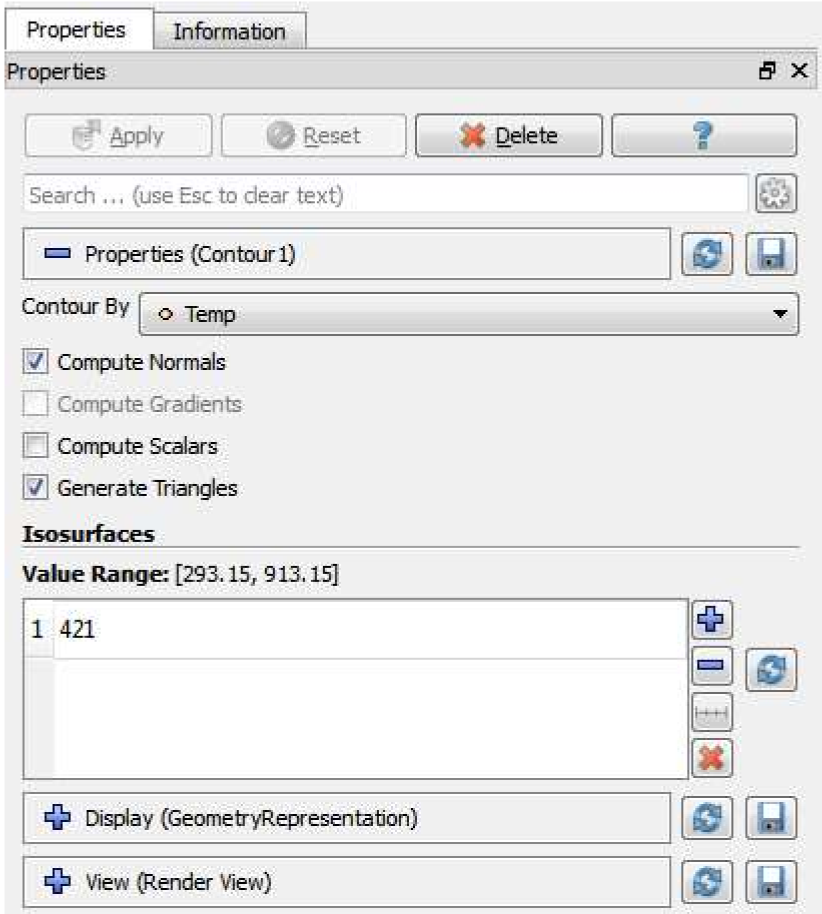
- Login to jupyter-jsc.fz-juelich.de
- Start xpra or a vncserver and paraview

- Load some data, e.g.

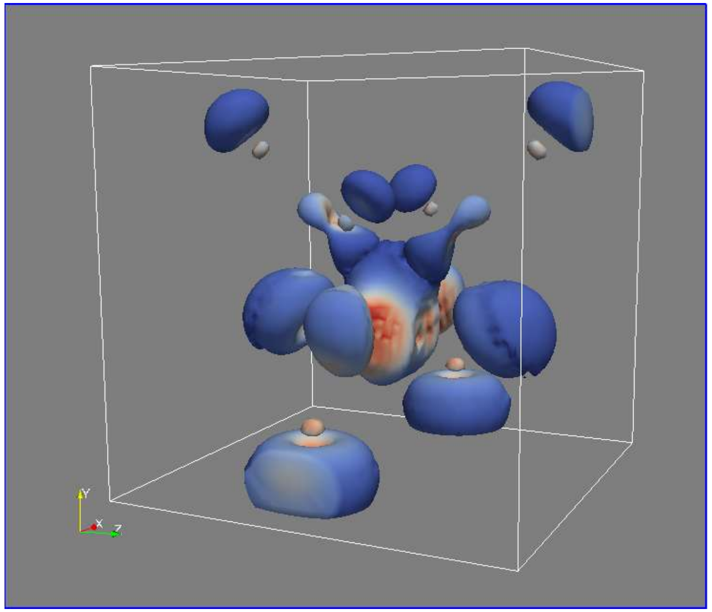
  /p/scratch/share/zilken1/HandsOn_Remote_Vis/headsq.vti

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

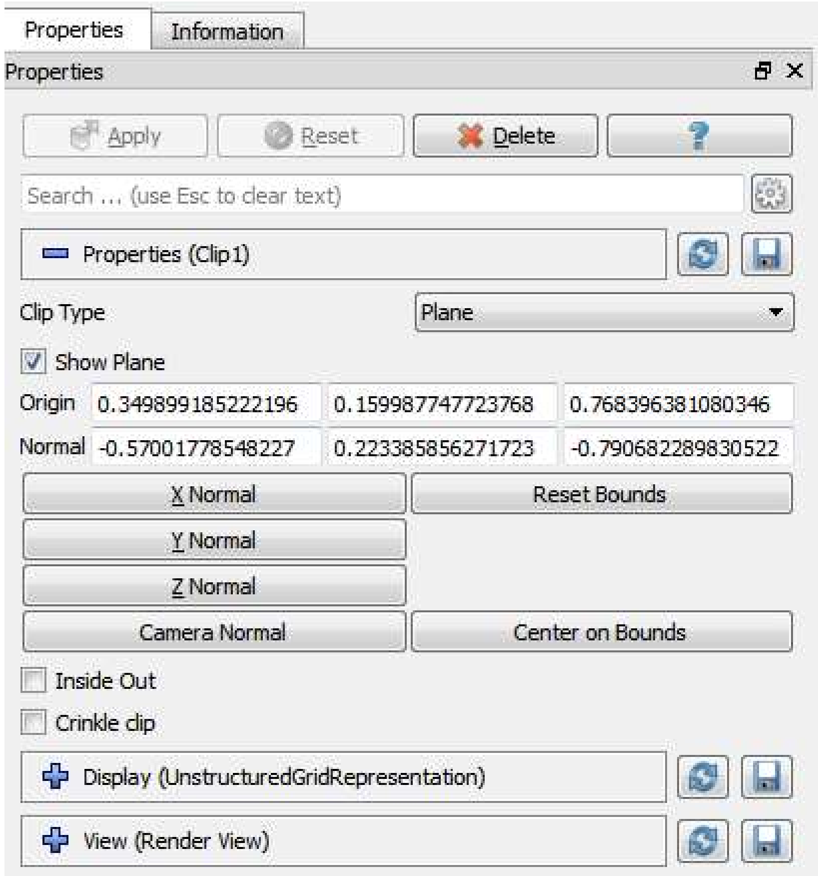
Common Filters: Contour



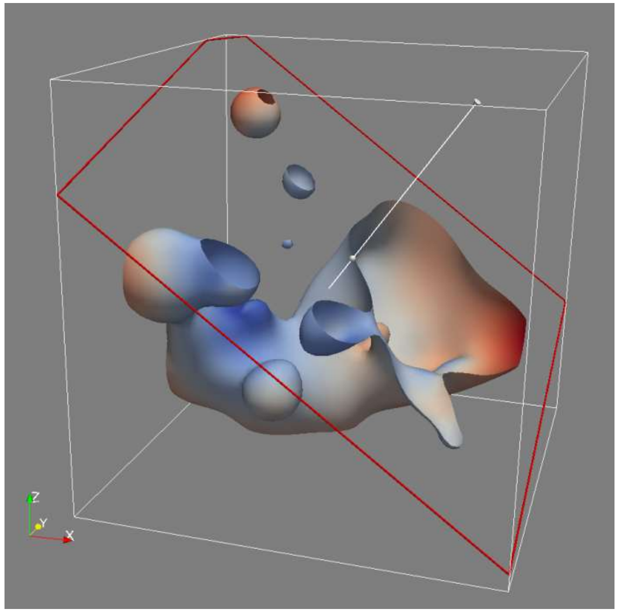
- Extracts the points, curves, or surfaces where a scalar field is equal to a user-defined value.
- This surface is often also called an isosurface



Common Filters: Clip

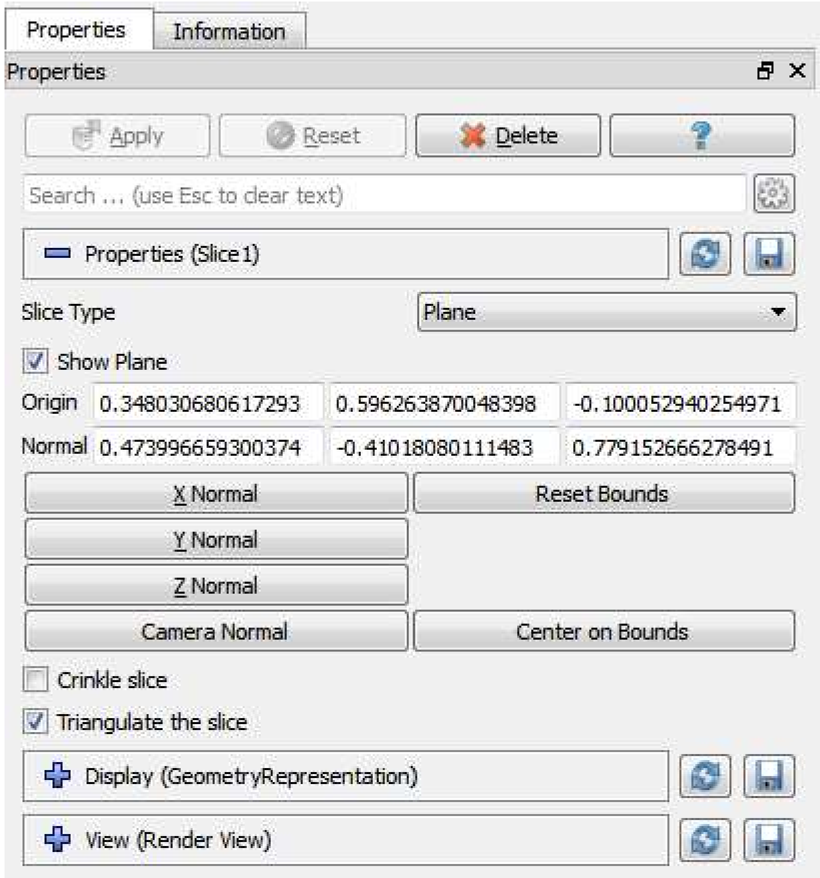


- Intersects the geometry with a user-defined plane, box or sphere
- Removes all the geometry on one side of this plane (box, sphere)

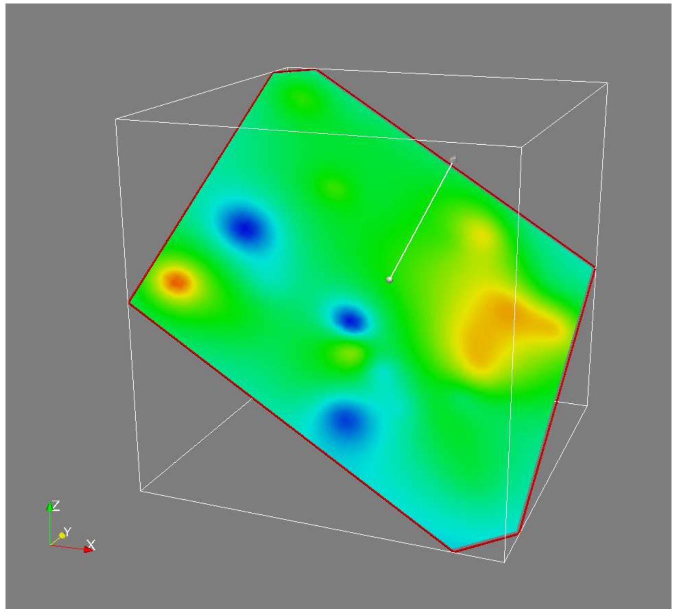


Beware of data explosion:
Structured data is converted to unstructured!

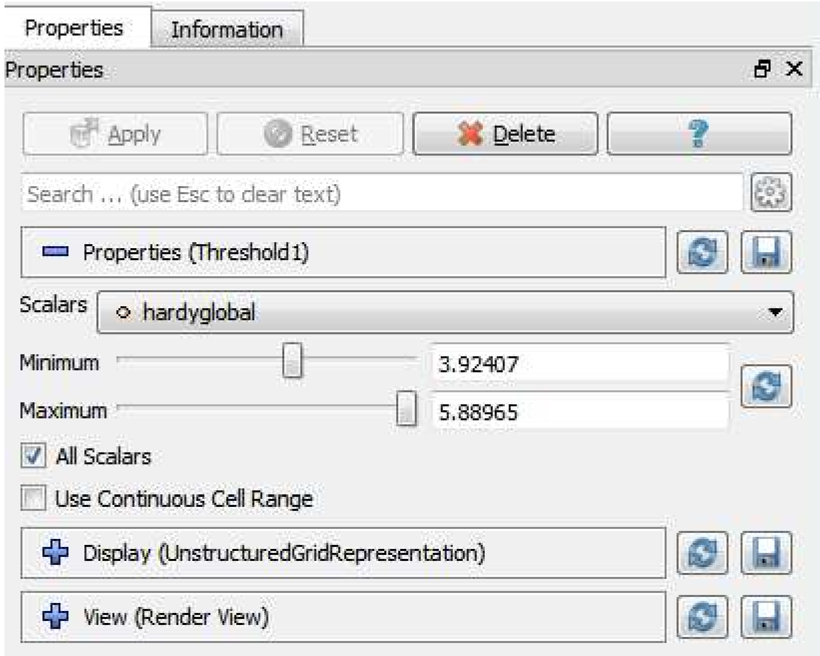
Common Filters: Slice



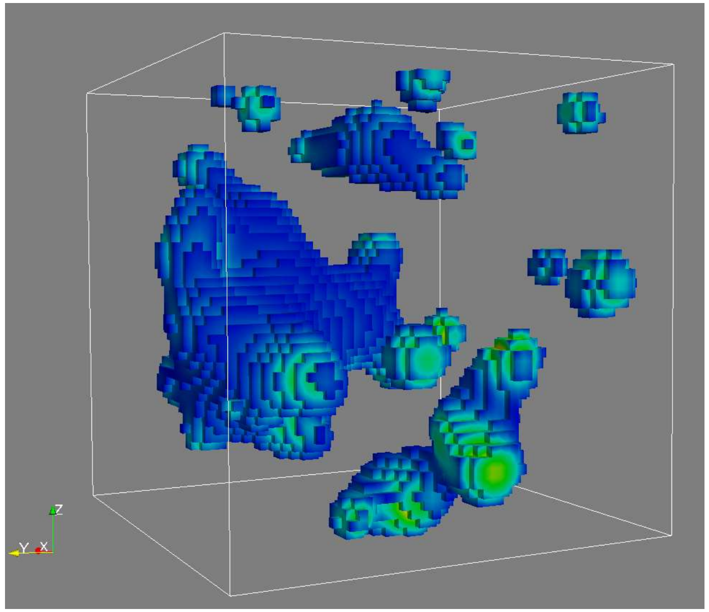
- Intersects the geometry with a plane, box, sphere or cylinder
- Similar to clipping, except that all that remains is the geometry where the plane is located.



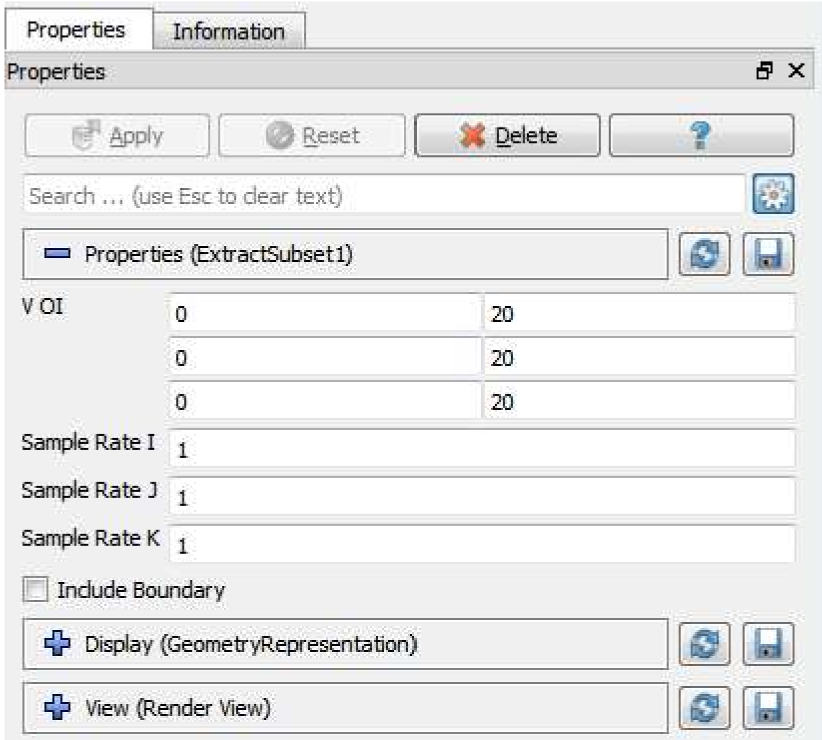
Common Filters: Threshold



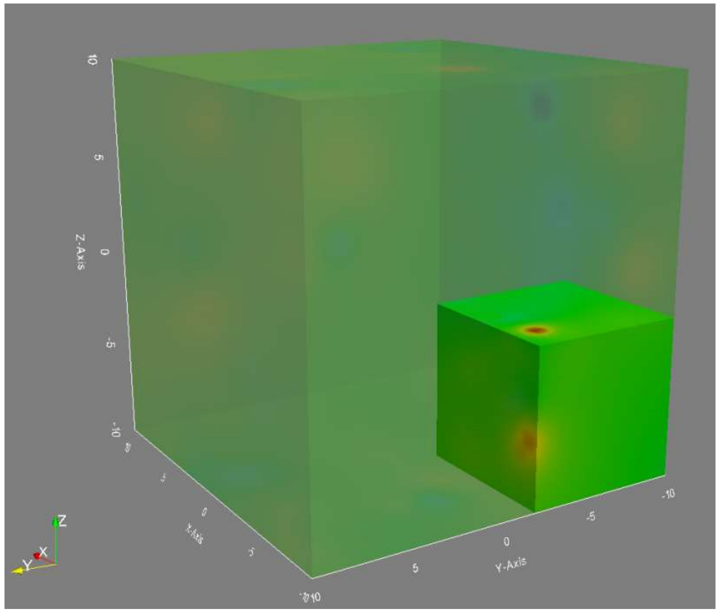
- Extracts cells that lie within a specified range of a scalar field



Common Filters: Extract Subset



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



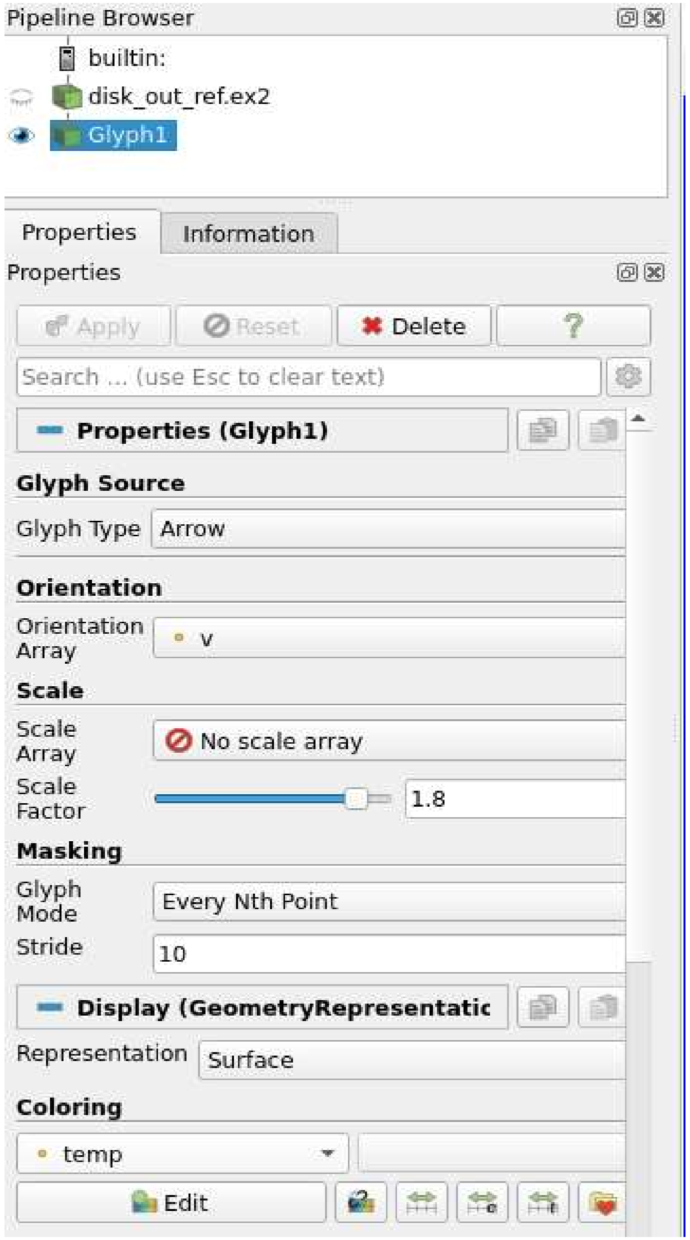
Exercise 2

- Load

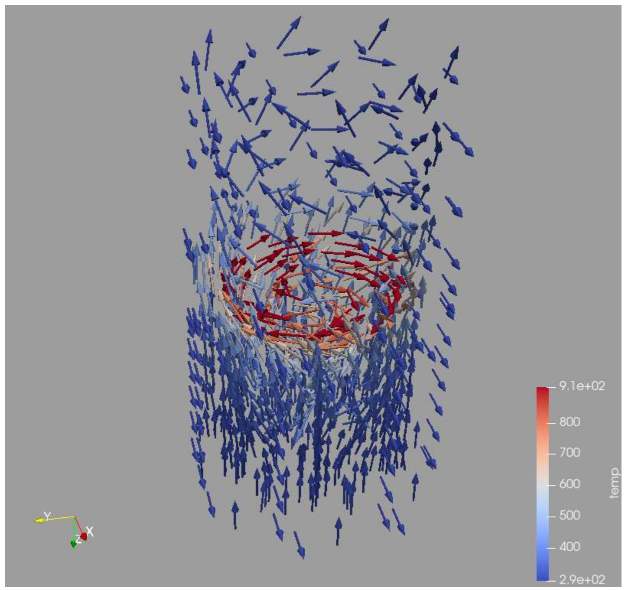
  /p/scratch/share/zilken1/HandsOn_Remote_Vis/disk_out_ref2.ex2

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

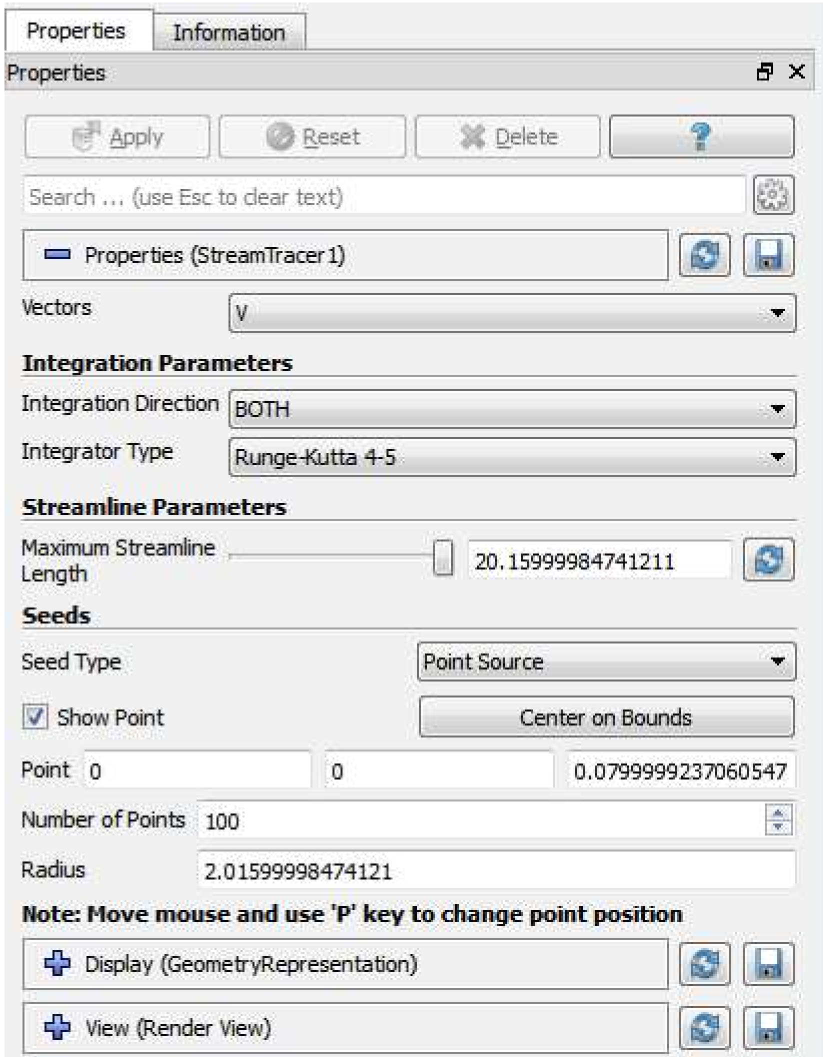
Common Filters: Glyph



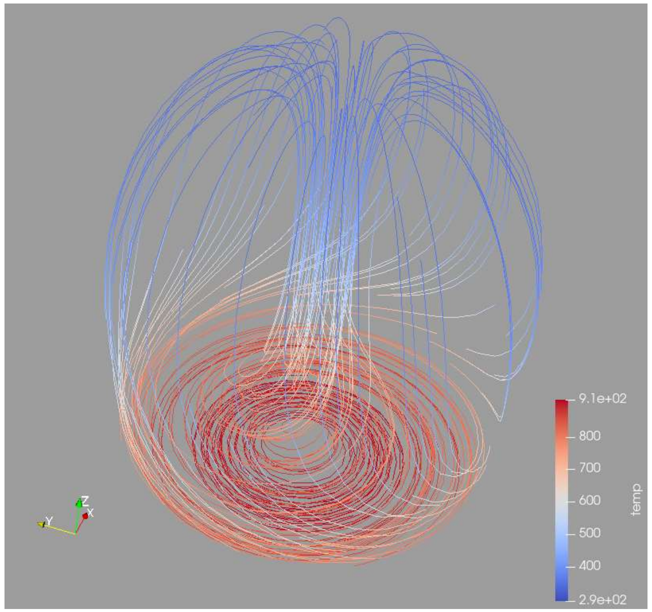
- Places a glyph, a simple shape, on each point (or subset) in a mesh
- glyphs may be oriented by a vector and scaled by a vector or scalar.



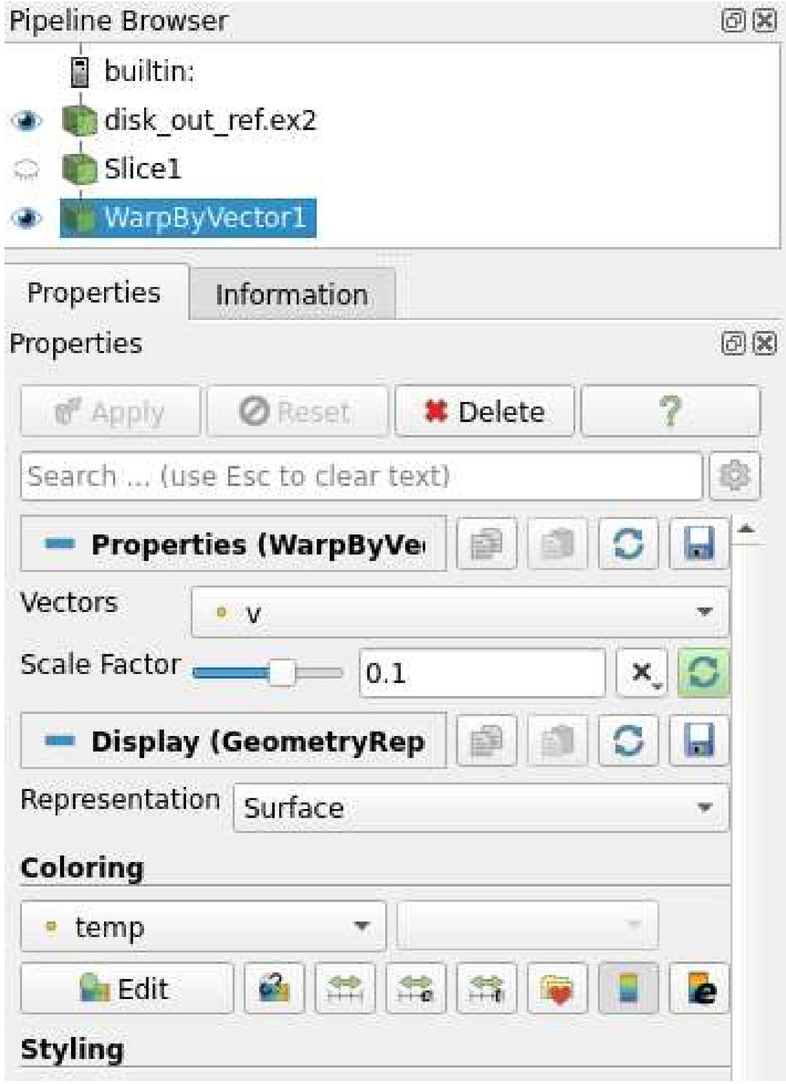
Common Filters: Stream Tracer



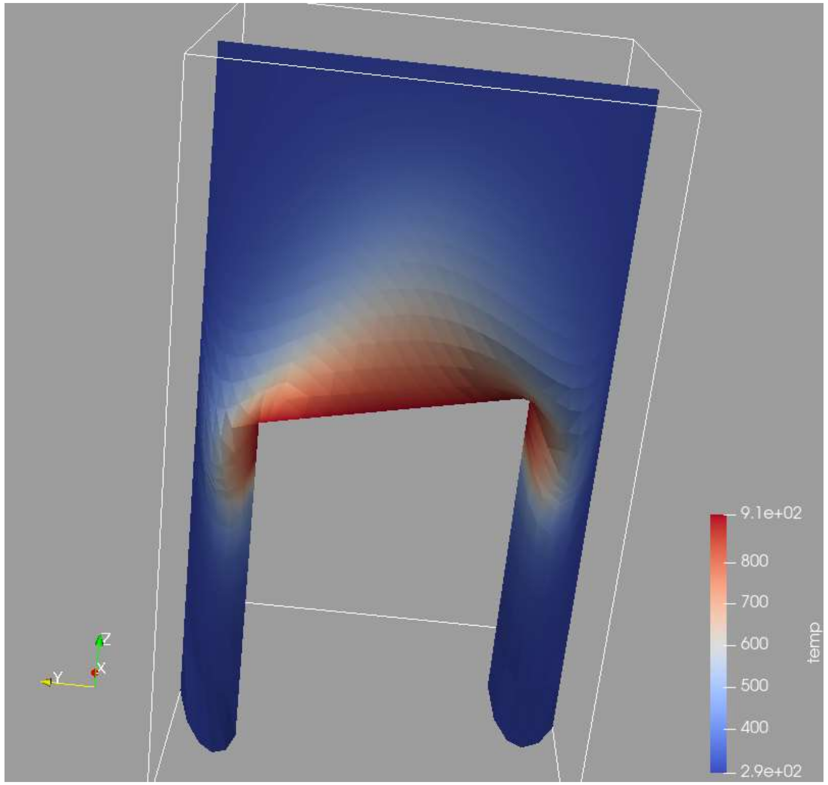
- Seeds a vector field with points and then traces those seed points through the (steady state) vector field.



Common Filters: Warp (vector)

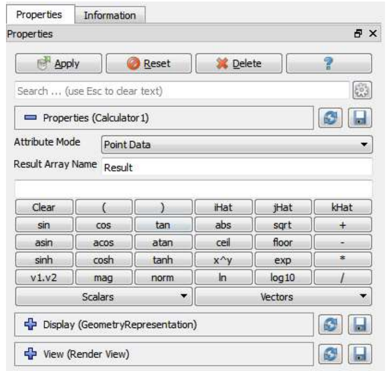


- Displaces each point in a mesh by a given vector field.



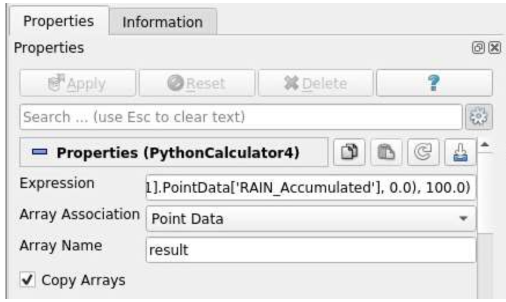
Calculations within ParaView

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



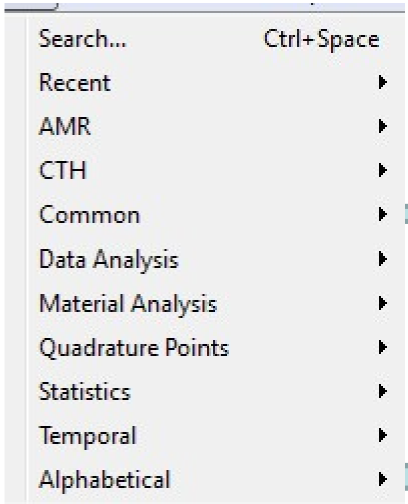
- NumPy and SciPy functions can be used
- Can generate vectors from scalars via „make_vector (velocity_x , velocity_y , velocity_z)”
- No „if“ statement, but numpy.where works, e.g. „numpy.where(Rain > 20, -1 * Rain, LANDMASK*(numpy.abs(HGT)+20))“

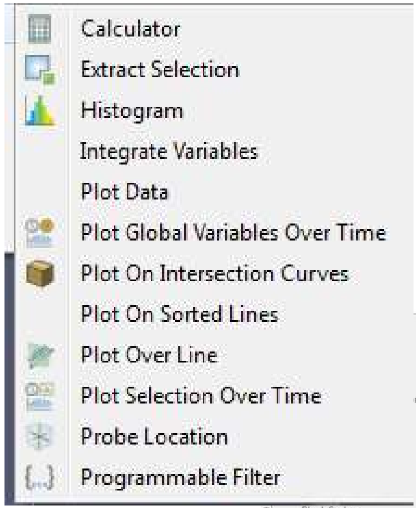
Programmable Source/Filter

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

Filter Menu:

Many more filters in the Filters Menu





- 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