Mitglied der Helmholtz-Gemeinschaft

JÜLICH Forschungszentrum

Remote Visualization at JSC (with ParaView)

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

Herwig Zilken, 2022/05/19

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, presentations or on TV

Visualization at JSC

JUWELS: closer look at login nodes

Cluster

4 x Login Nodes with GPU

- 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

9 x Login Nodes without GPU

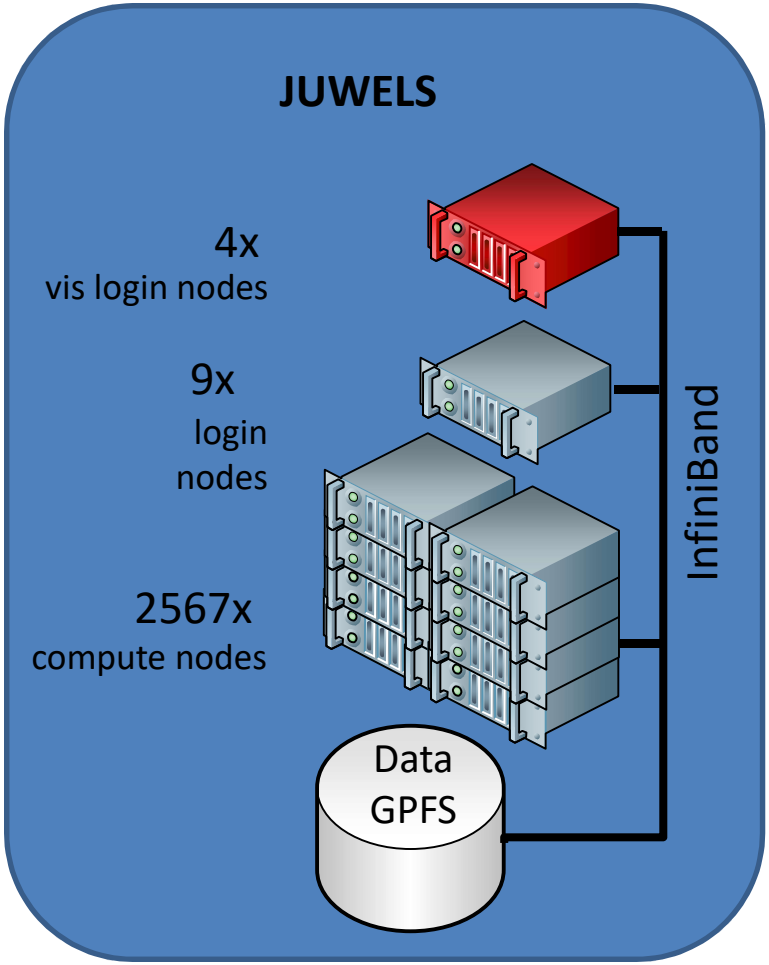
- juwels-cluster

Booster:

4 x Login Nodes without GPU

- juwels-booster

Keep in mind: software rendering is possible on any node



Visualization at JSC

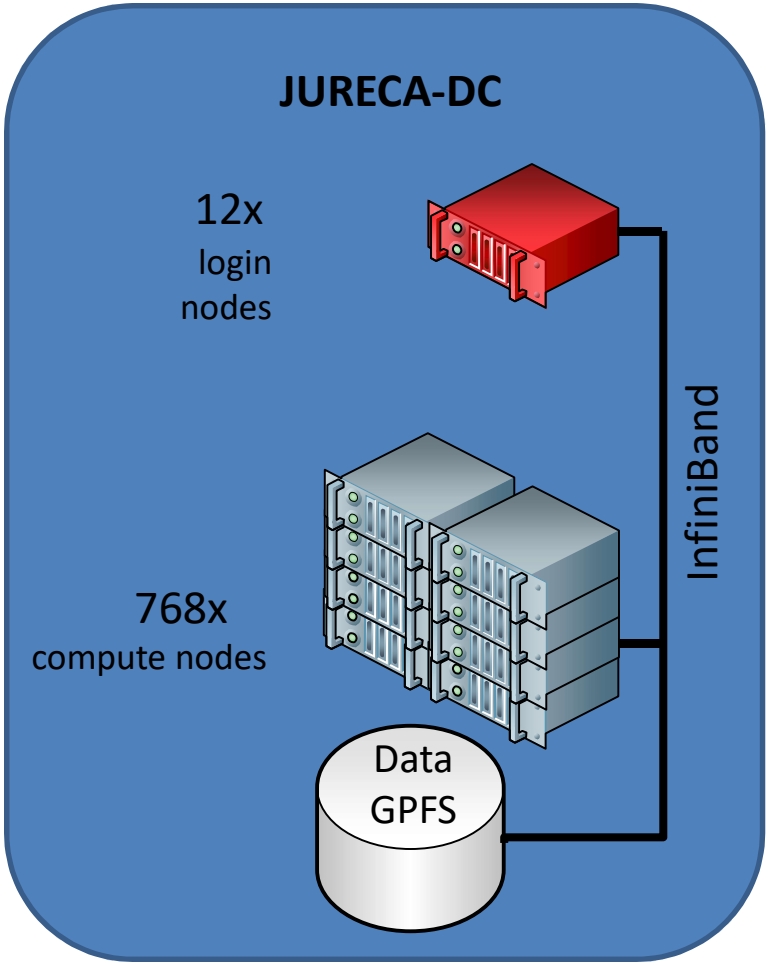
JURECA-DC: closer look at login nodes

12 x Login Nodes with GPU

- jureca.fz-juelich.de
- (jureca01 to jureca12 in round-robin fashion)
- 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

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
  easy setup, our recommendation → good idea

Remote 3D Visualization

with X Forwarding + Indirect Rendering

JÜLICH Forschungszentrum

Traditional Approach (X forwarding + Indirect Rendering)

ssh –X <USERID>@<SERVER>

- uses GLX extension to X Window System
- X display runs on user workstation
- OpenGL command 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 + 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/

How to use Xpra @ JSC

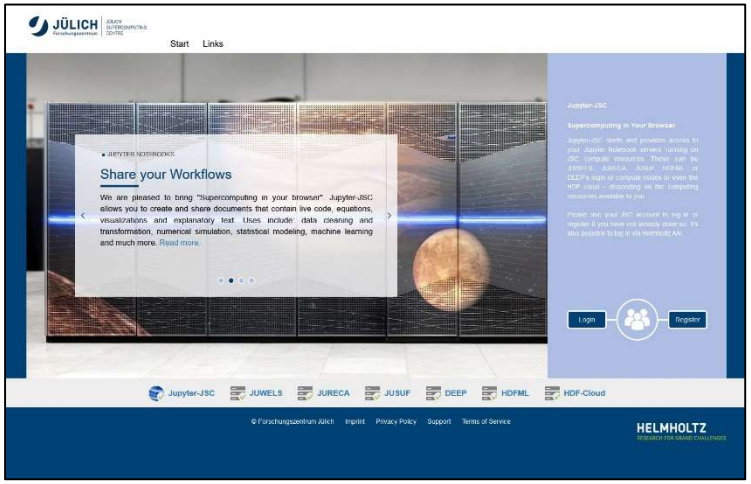
How to start an Xpra session:

- **Recommended way: from JupyterLab@JSC**
  https://jupyter-jsc.fz-juelich.de
  - See next slides

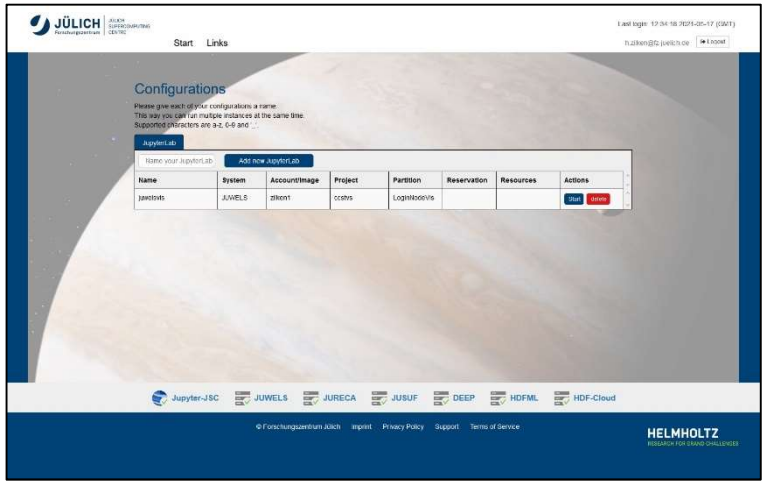
- Alternative: start Xpra session manually
  - Neither recommended nor necessary
  - Just in case you need it: documentation provided on later slides

Xpra Integration in JupyterLab@JSC

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

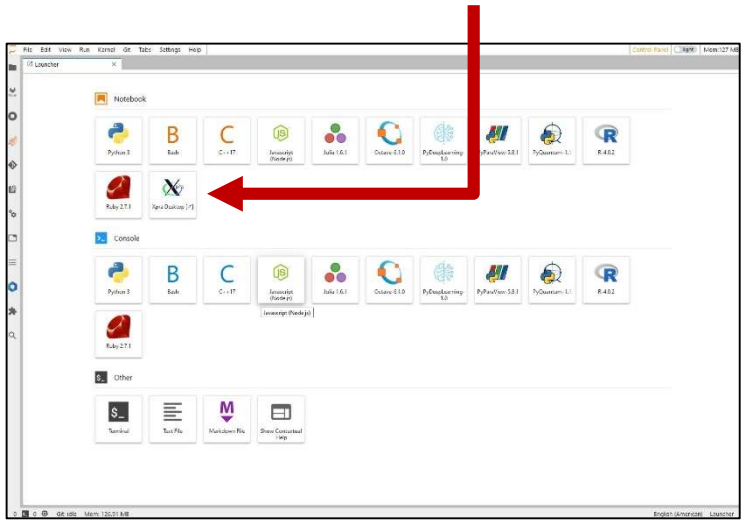


2. Add a new or start an existing JuperLab on JURECA login node or JUWELS vis login node.

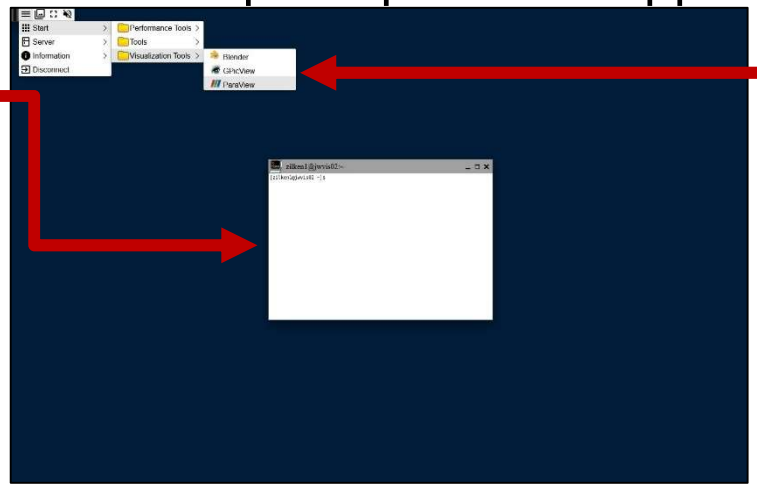


Xpra Integration in JupyterLab@JSC

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

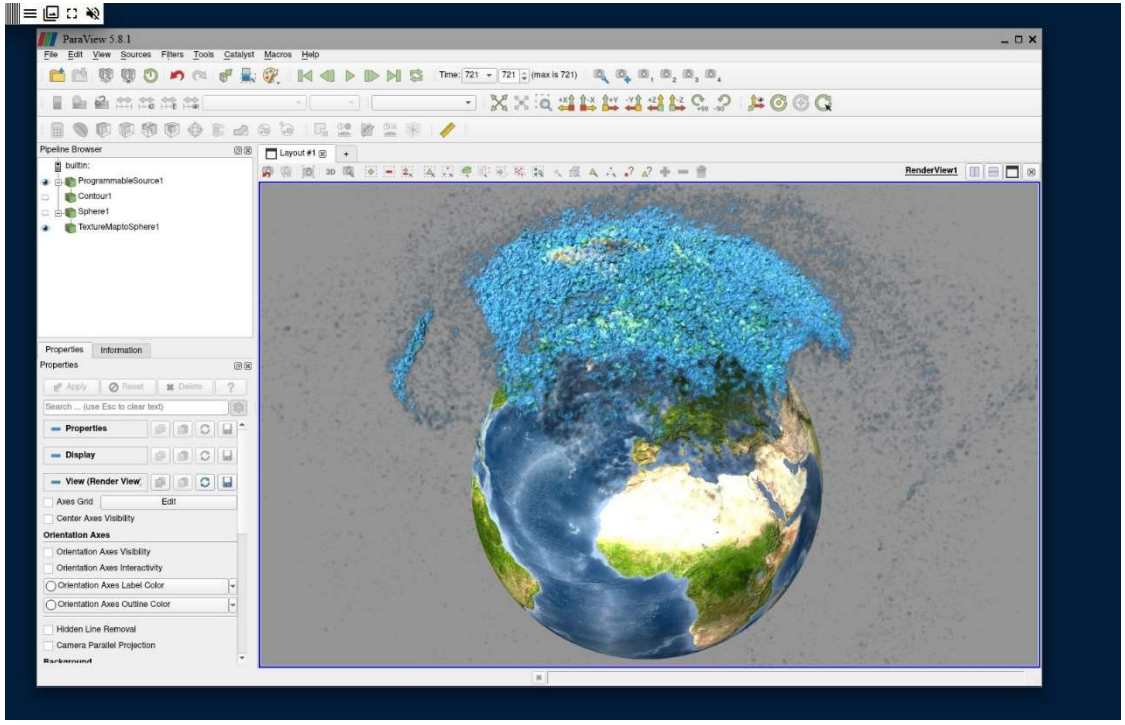


4. Wait for the HTML desktop of Xpra. Start apps from the menue or from the Xterm.



Xpra Integration in JupyterLab@JSC

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



ParaView

for data visualization

Exercise 1

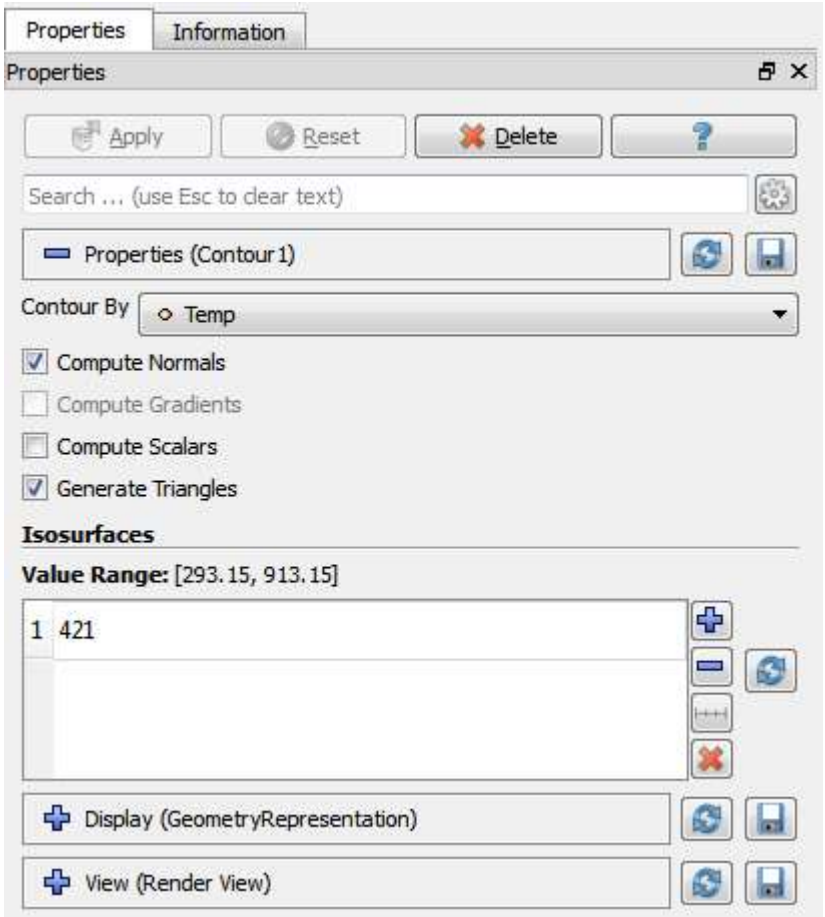
- Login to jupyter-jsc.fz-juelich.de
- Start Xpra and ParaView

- Load some data, e.g.

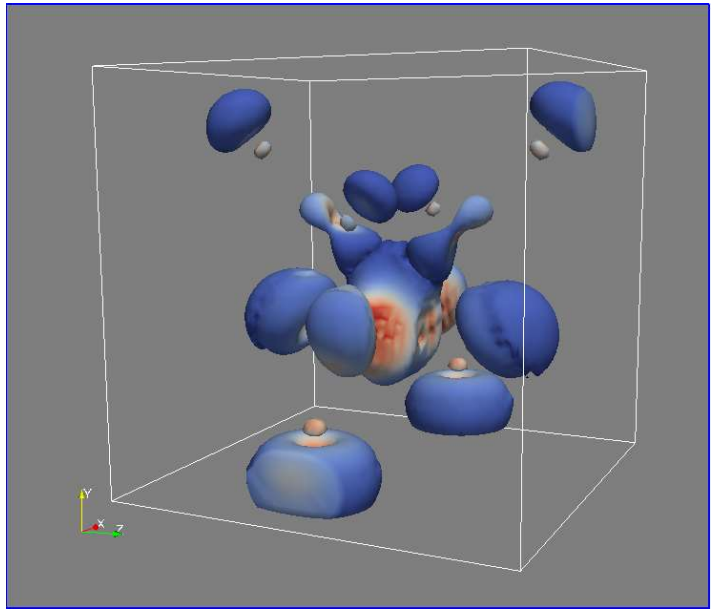
  /p/scratch/share/zilken1/ParaView_HandsOn/headsq.vti

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

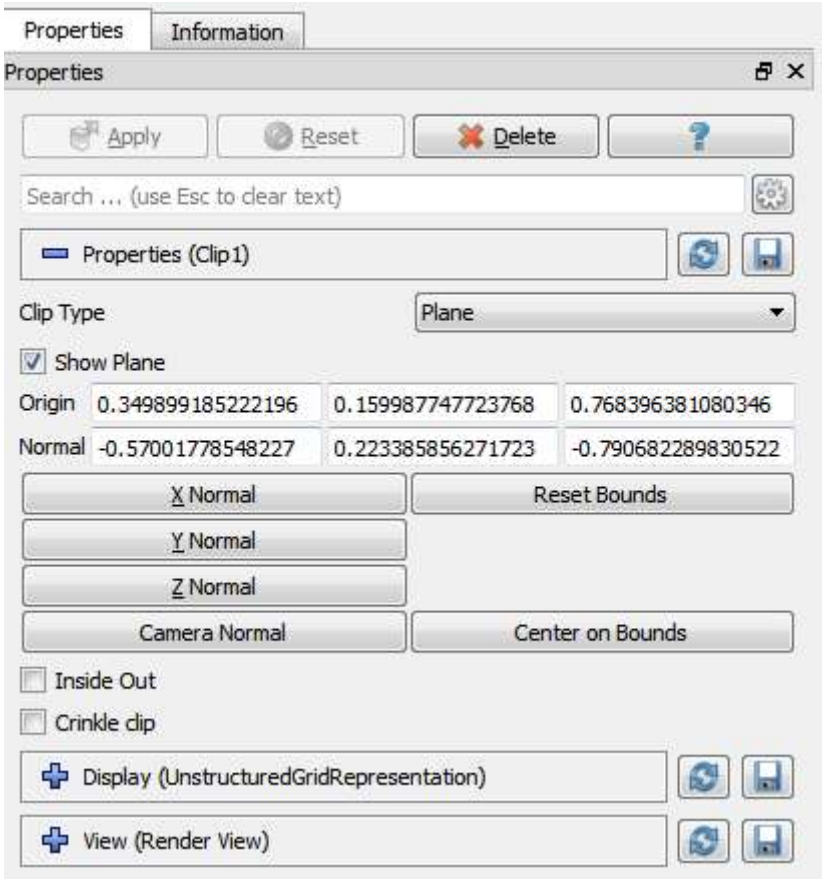
Common Filters: Contour



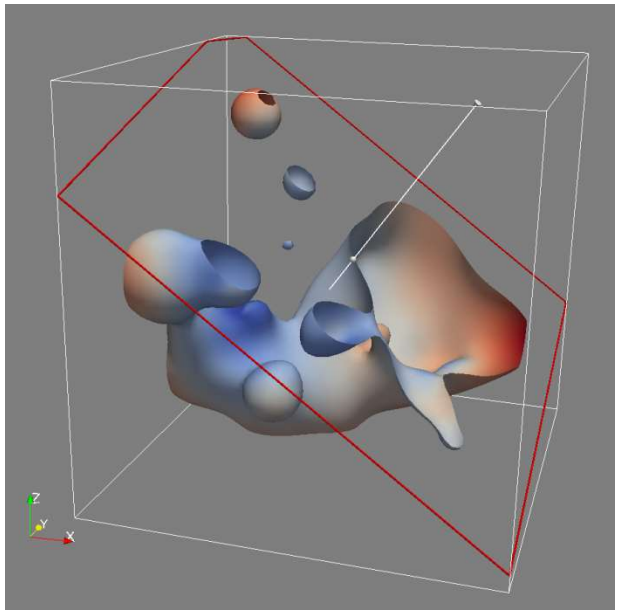
- Extracts the points, curves, or surfaces where a scalar field is equal to a user-defined value.
- This surface is often also called an isosurface



Common Filters: Clip

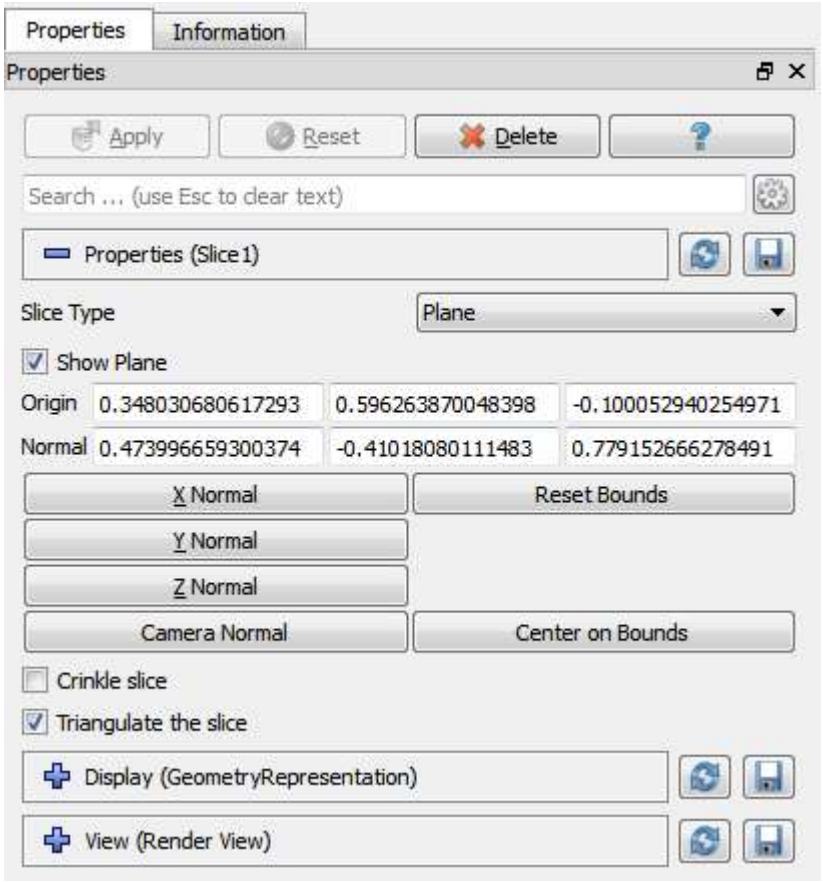


- Intersects the geometry with a user-defined plane, box or sphere
- Removes all the geometry on one side of this plane (box, sphere)

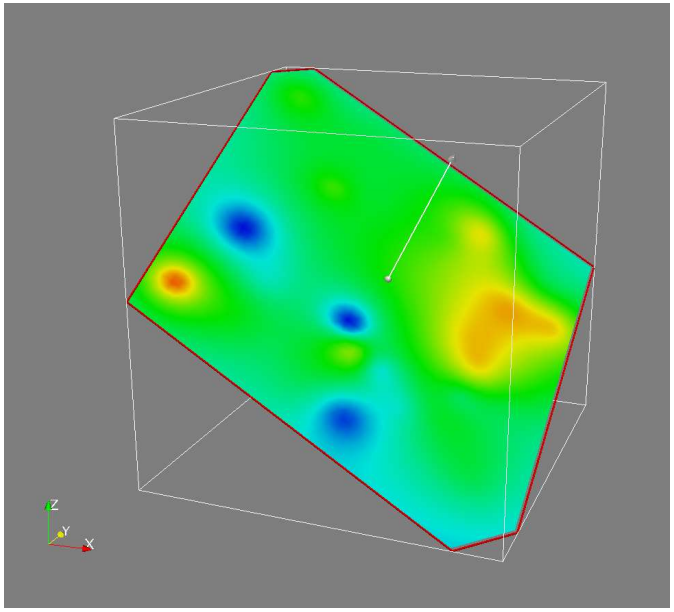


Beware of data explosion:
Structured data is converted to unstructured!

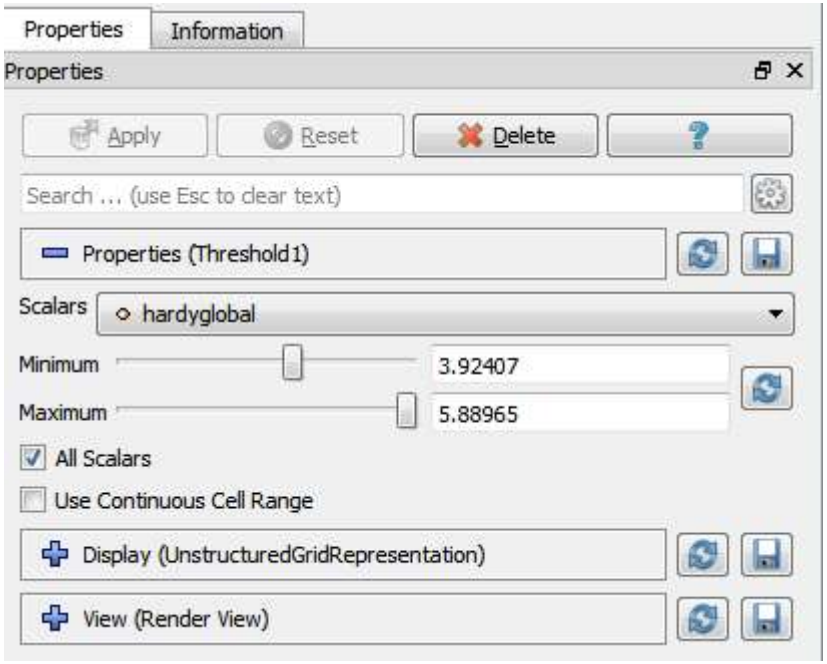
Common Filters: Slice



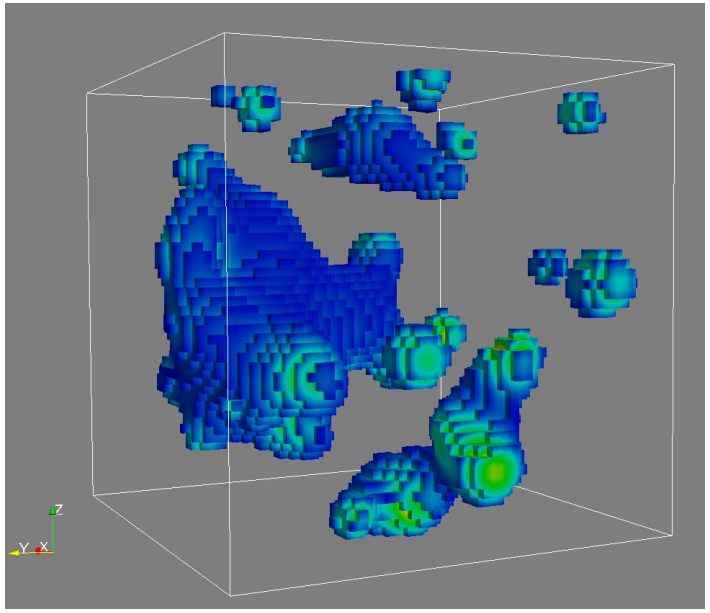
- Intersects the geometry with a plane, box, sphere or cylinder
- Similar to clipping, except that all that remains is the geometry where the plane is located.



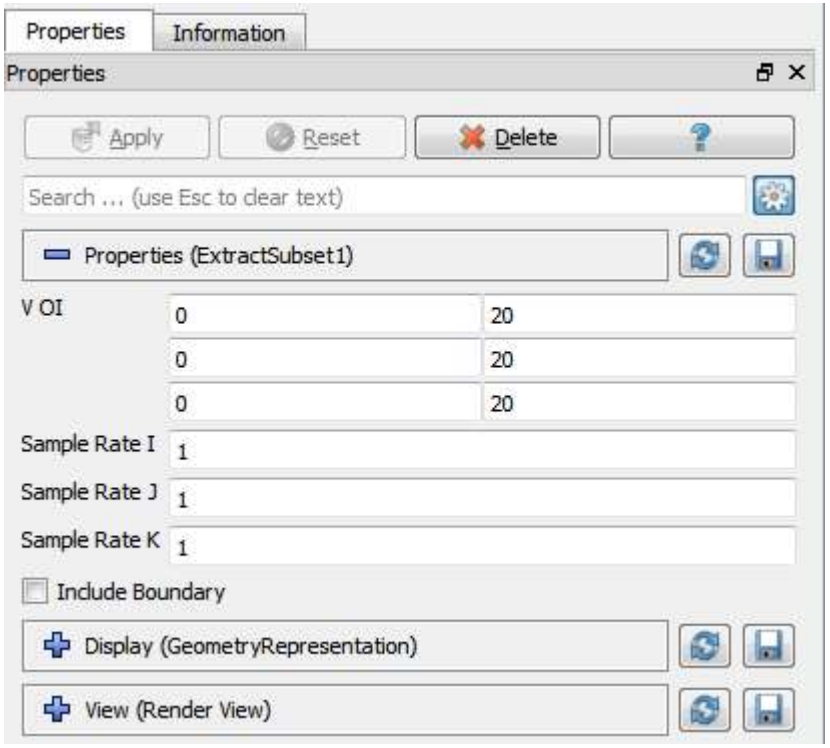
Common Filters: Threshold



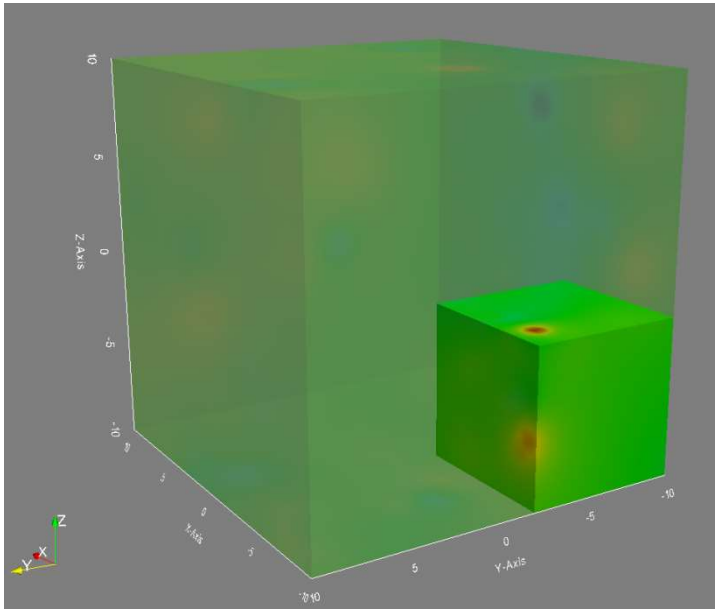
- Extracts cells that lie within a specified range of a scalar field



Common Filters: Extract Subset



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



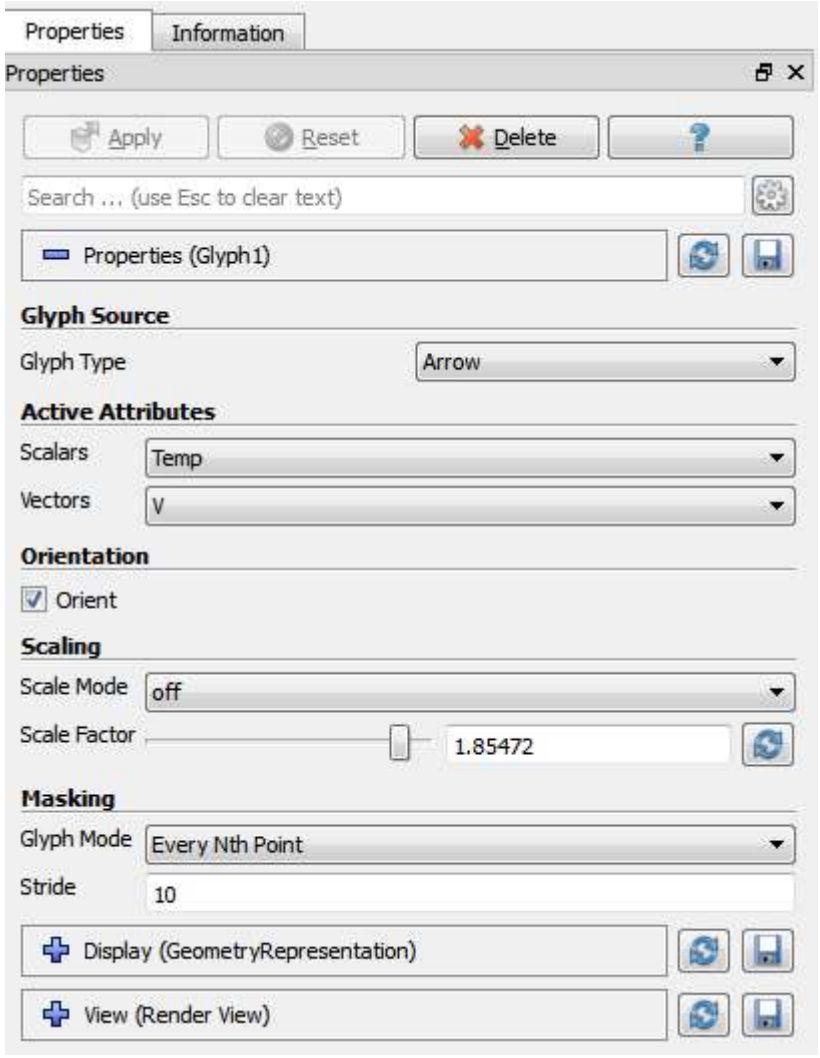
Exercise 2

- Load

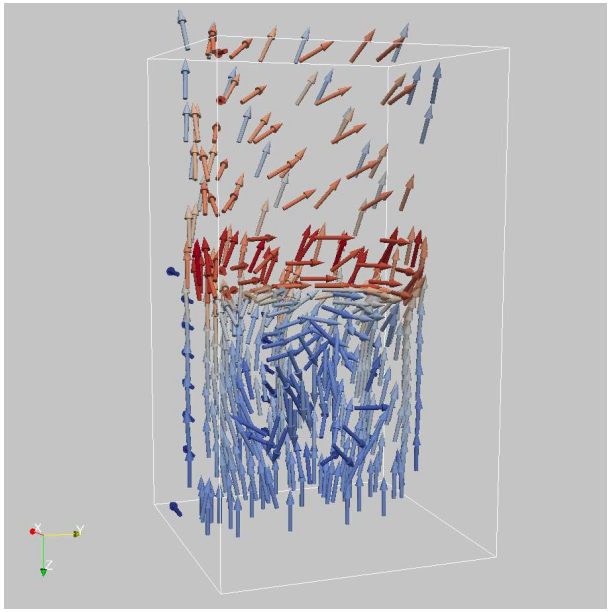
  /p/scratch/share/zilken1/ParaView_HandsOn/disk_out_ref2.ex2

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

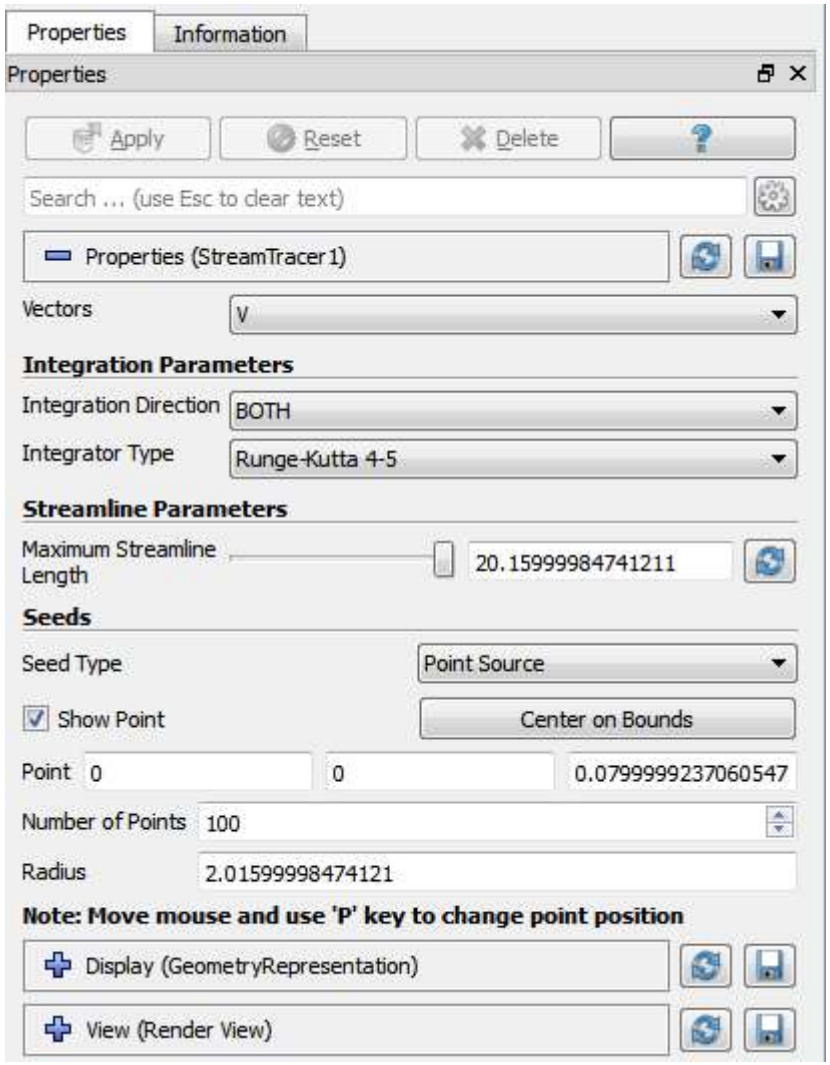
Common Filters: Glyph



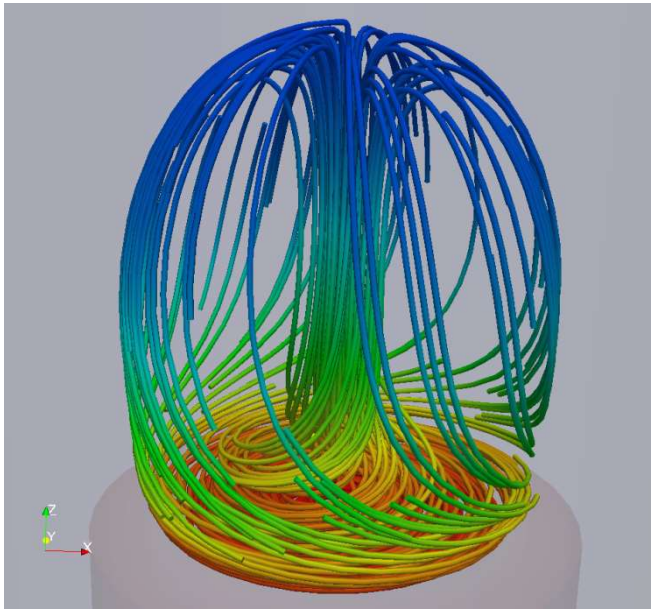
- Places a glyph, a simple shape, on each point (or subset) in a mesh
- glyphs may be oriented by a vector and scaled by a vector or scalar.



Common Filters: Stream Tracer

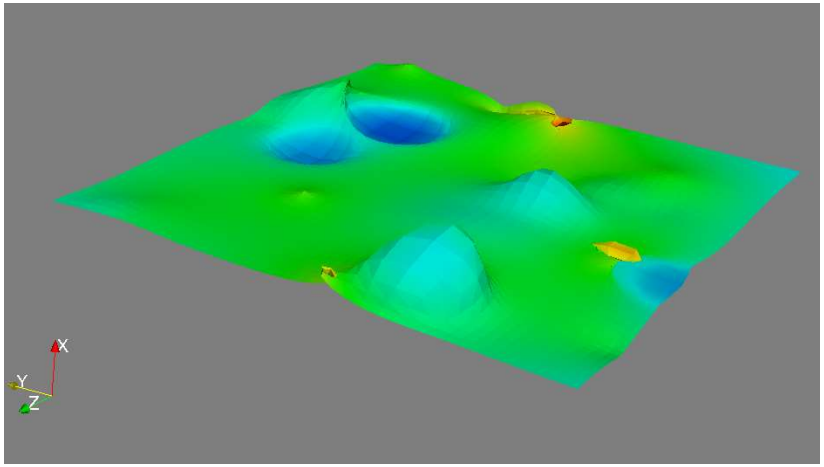


- Seeds a vector field with points and then traces those seed points through the (steady state) vector field.



Common Filters: Warp (vector)

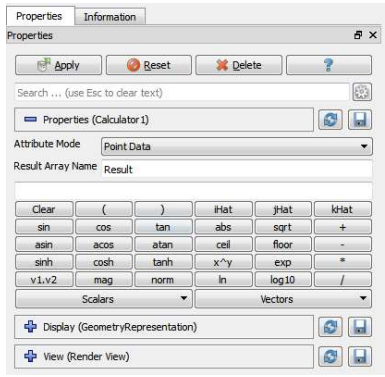
- Displaces each point in a mesh by a given vector field.



Calculations within ParaView

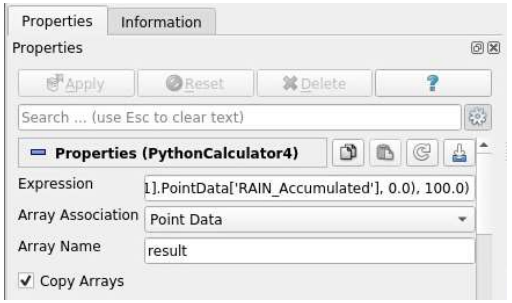
Calculator: calculates new attributes based on simple expression

- 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))“

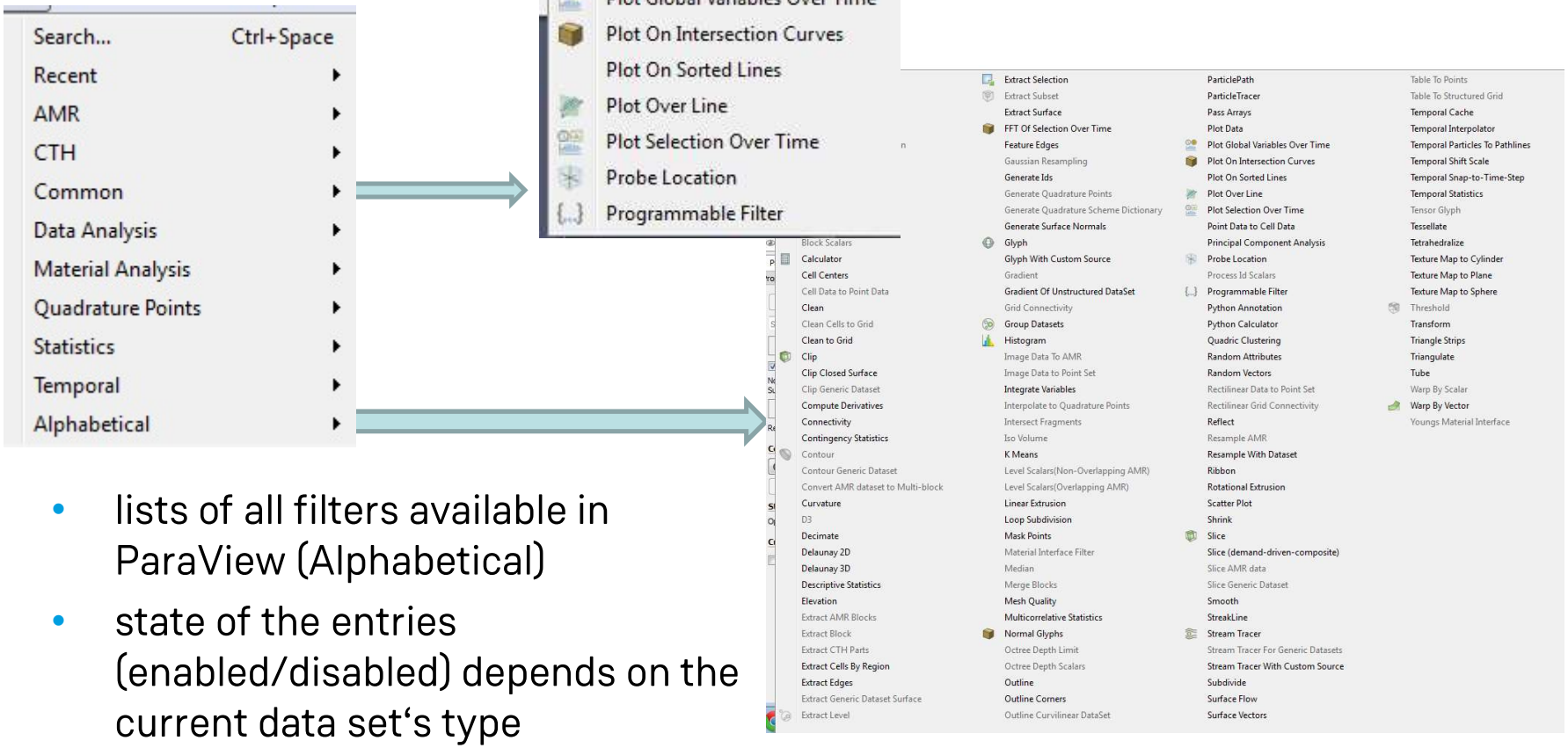


Programmable Source/Filter

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

Filter Menu:

Many more filters in the Filters Menu

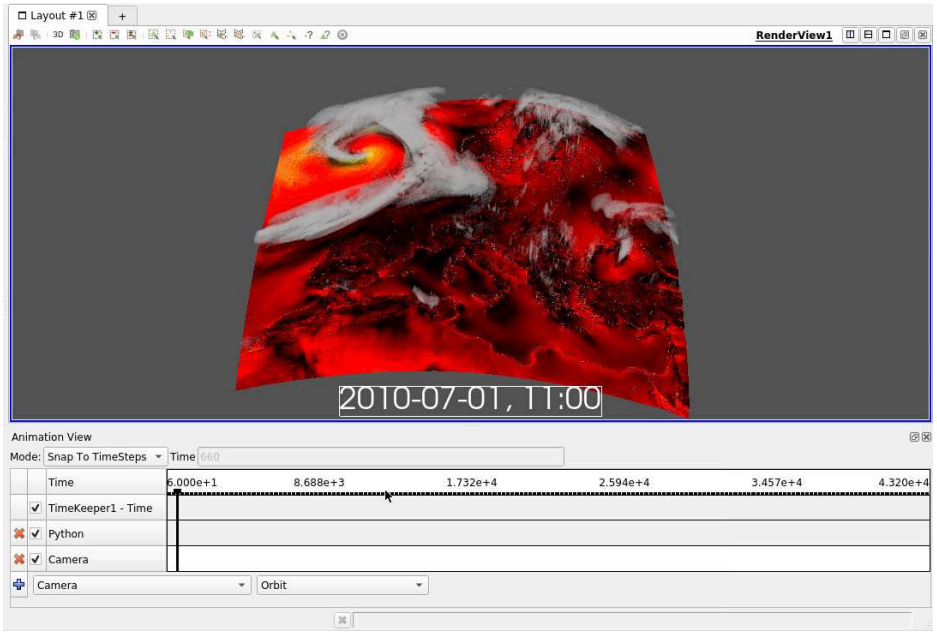


- lists of all filters available in ParaView (Alphabetical)
- state of the entries (enabled/disabled) depends on the current data set‘s type
- See https://www.paraview.org/Wiki/ParaView/Users_Guide/List_of_filters

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

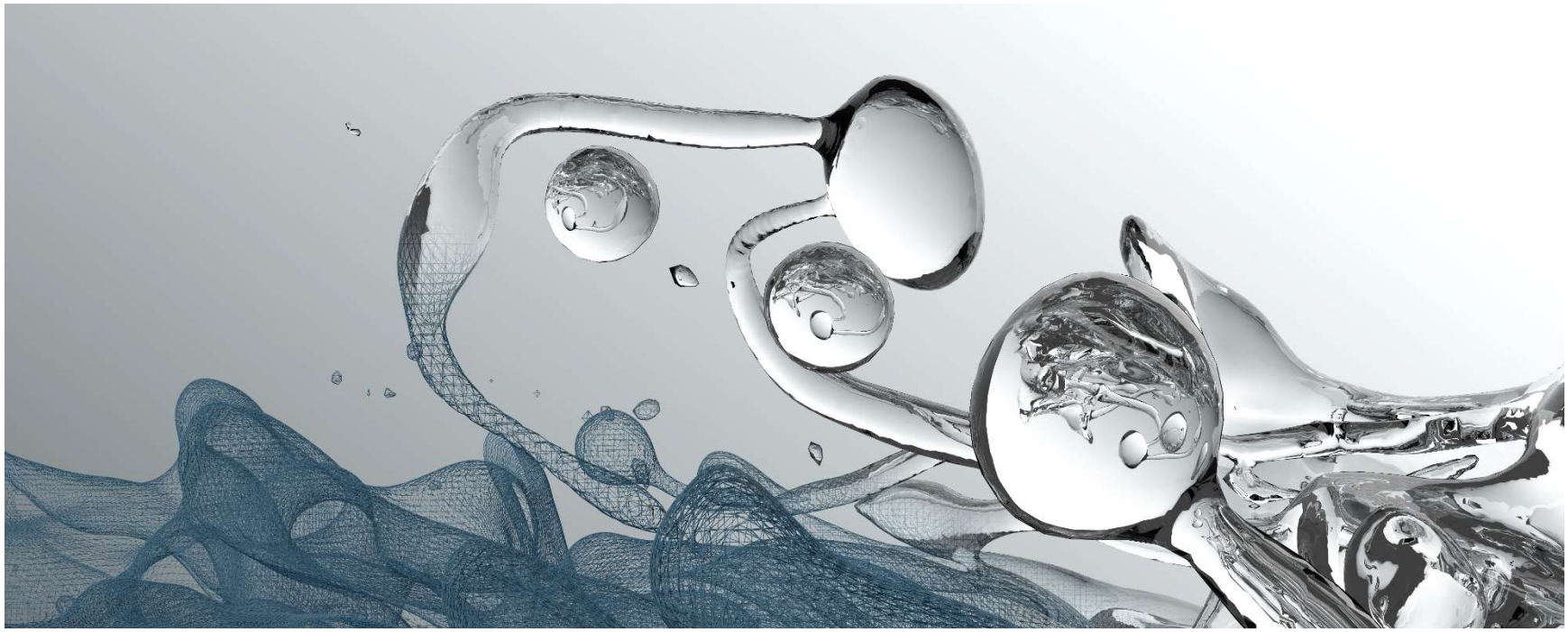
- Load

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

- Lets have some fun with **animations**

Thank you for your attention

Questions … ?

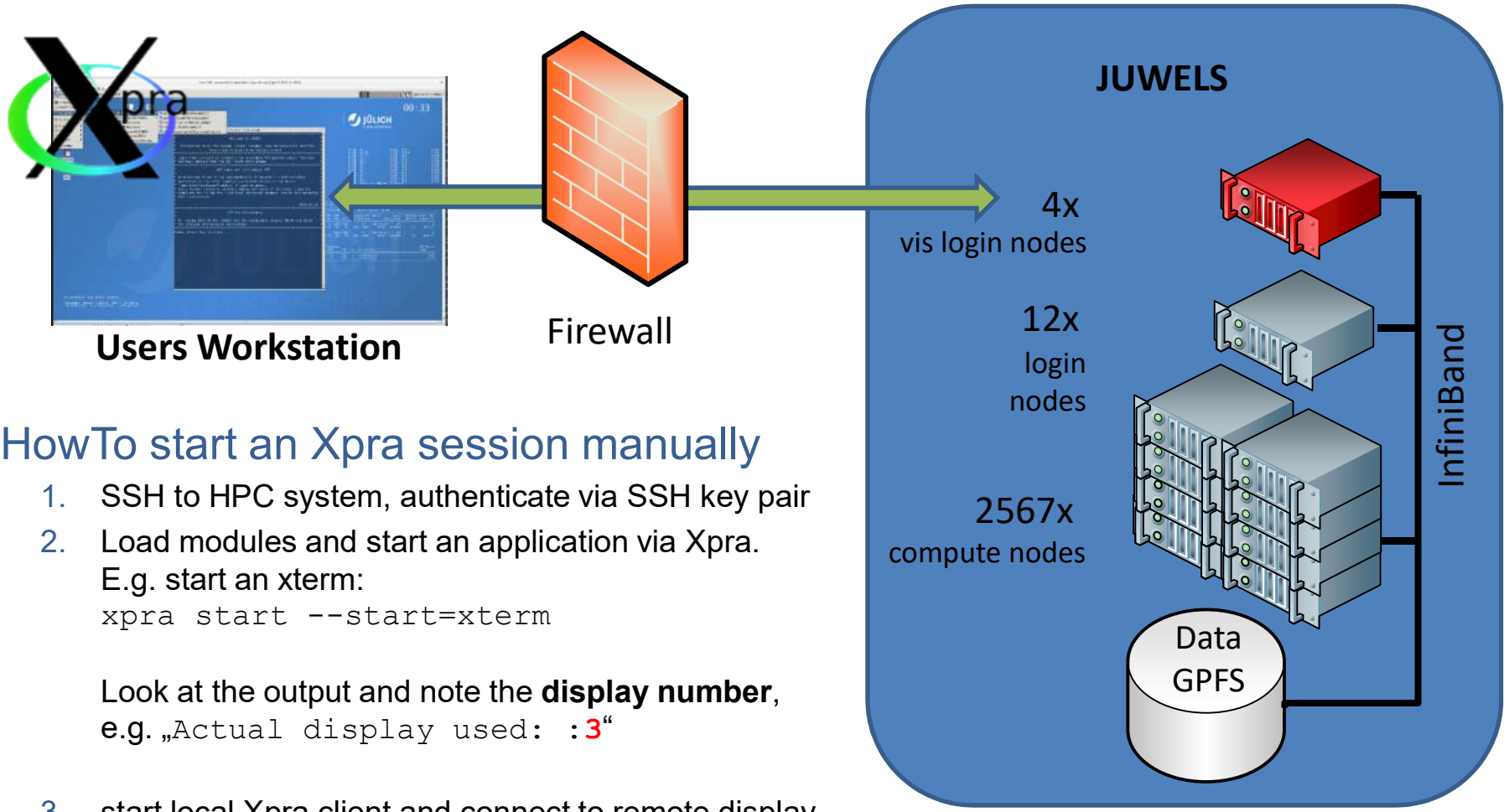


rendered with Blender from a DNS of a diesel injection spray of ITV, RWTH Aachen University

Appendix: How to start Xpra manually

Manual Setup of Xpra

with Xpra + VirtualGL



HowTo start an Xpra session manually

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: 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 in Xpra:

```
jwvis02> ml Stages/2022  GCCcore/.11.2.0 xpra/4.3.3

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

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

Setup Xpra

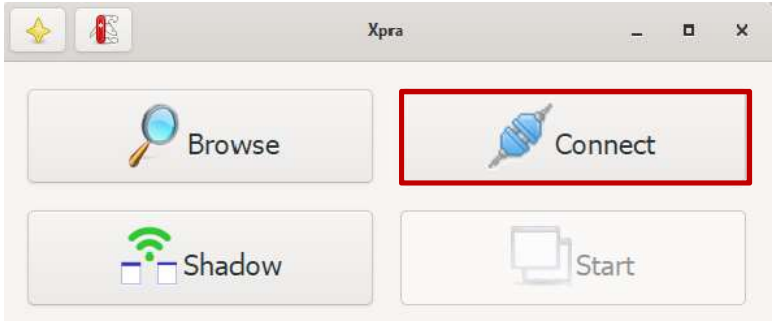
Step 3: connect to Xpra session

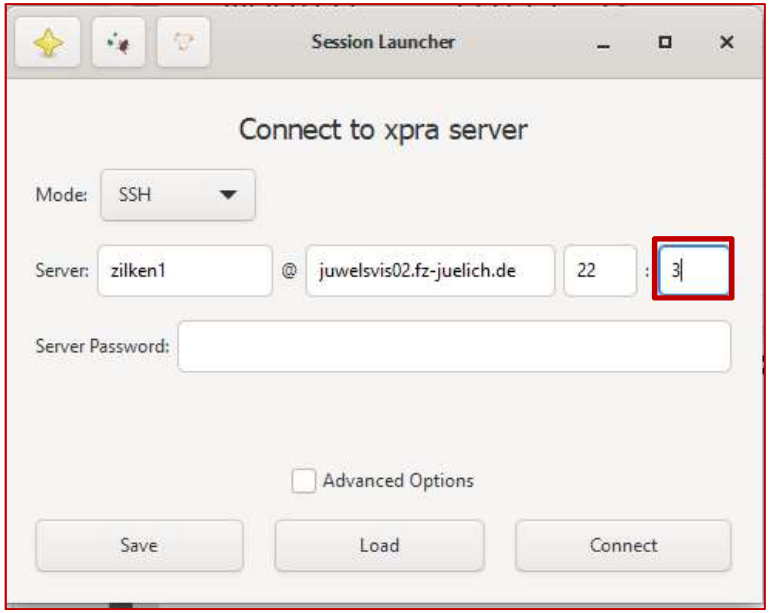
Install Xpra on your local machine. Download from www.xpra.org

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 5.10.1:

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

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