Scientific visualization with ParaView Part 1

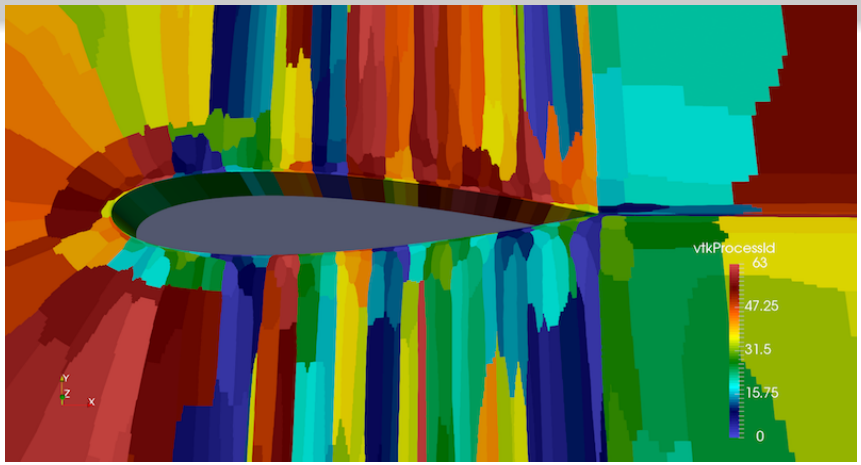
Alex Razoumov
alex.razoumov@westdri.ca

Digital Research Alliance of Canada

SFU SIMON FRASER UNIVERSITY

- ✔ slides, data, codes at `https://bit.ly/paraviewzipp`
  - the link will download a file `paraview.zip` (~36MB)
  - unpack it to find `codes/`, `data/` and `slides{1,2}.pdf`
  - command line: `wget https://bit.ly/paraviewzipp -O paraview.zip`
- ✔ install ParaView 5.10.x on your laptop from `http://www.paraview.org/download`

Workshop outline



- Introduction to sci-vis: general ideas, tools, plotting vs. multi-dimensional vis.
- Overview of current general-purpose multi-dimensional sci-vis tools

---

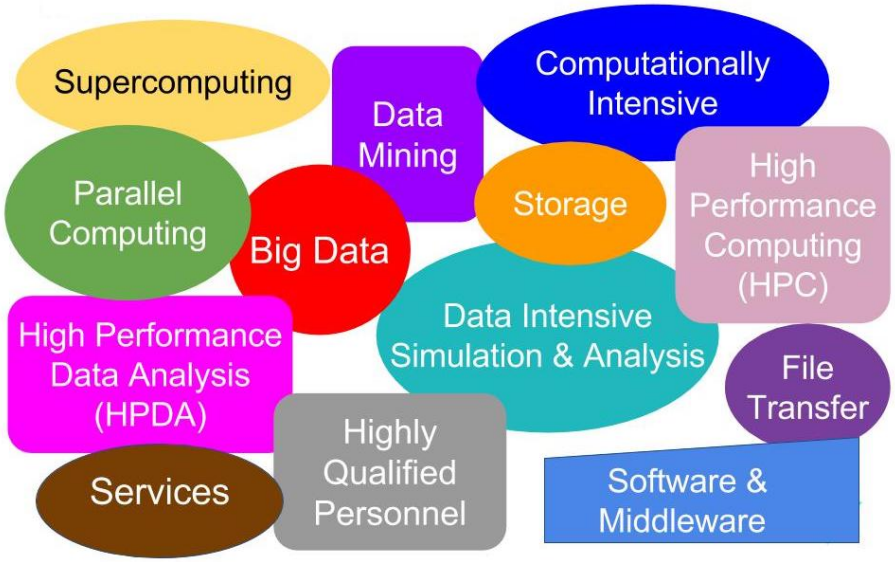
- ParaView's architecture
- Importing data into ParaView: raw binary, VTK data types, NetCDF/HDF5, OpenFOAM
- Basic workflows: filters, creating a pipeline, working with vector fields

---

- Scripting: ways to run scripts, few simple scripts, trace tool, programmable filter/source, camera animations
- Animation: three approaches, camera animation, one big exercise on scripting/animation, single timeline with many properties
- Remote visualization
  - running ParaView in **client-server** mode
  - opening and load balancing of very large, multi-GB datasets
  - recommendations on running on cluster GPUs vs. multiple CPUs
  - writing, debugging, running PV Python scripts as **batch rendering jobs**
- Not covered today: ParaView Cinema, in-situ visualization with Catalyst

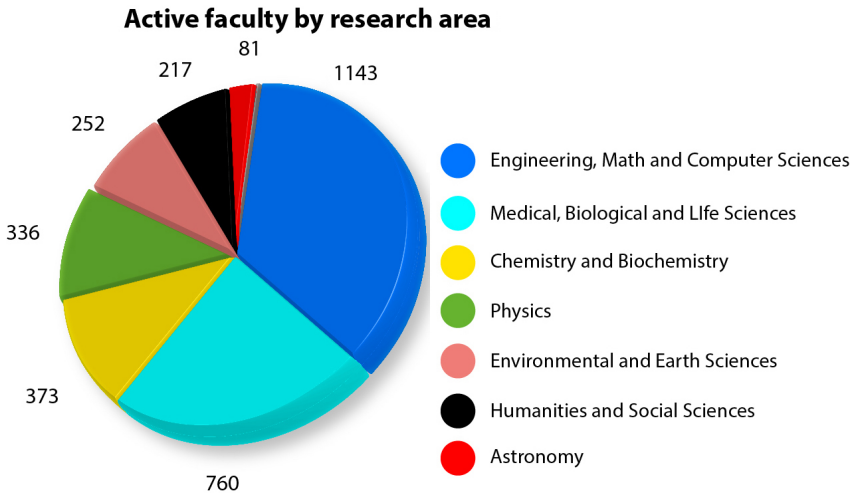
Who are we?

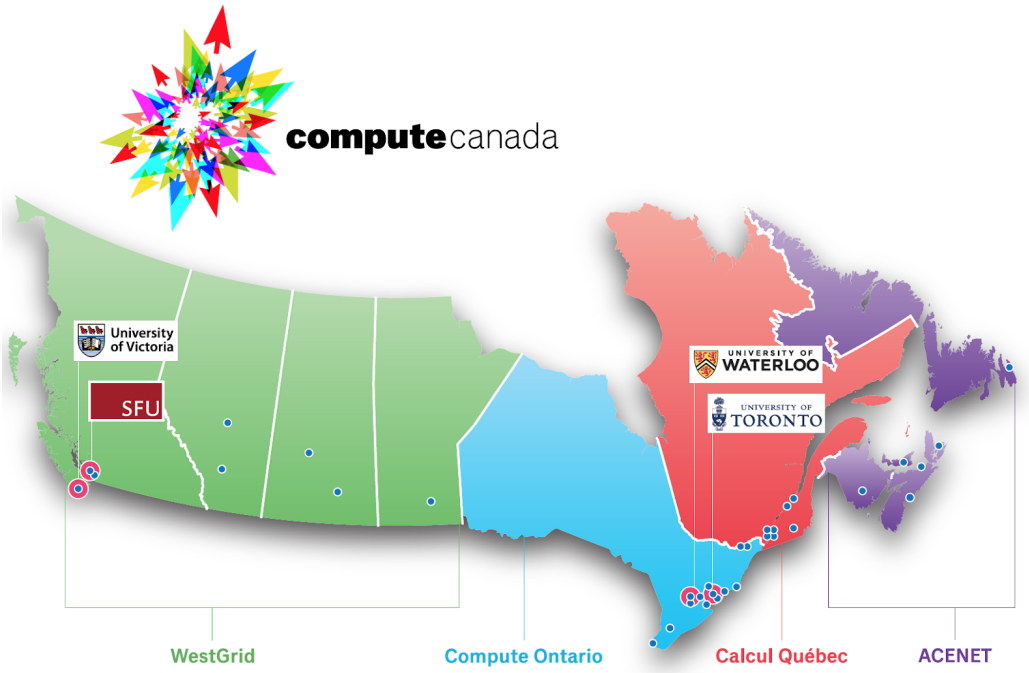
~~https://www.WestGrid.ca~~ https://docs.ALLIANCECAN.ca



We provide Advanced Research Computing (ARC) infrastructure and services, a.k.a. everything beyond a standard desktop, at *no cost* to researchers

- SUPERCOMPUTERS / HPC
- cloud computing
- data storage / management / transfer
- videoconf, training, support, visualization





Scientific visualization

Usually of spatially defined data

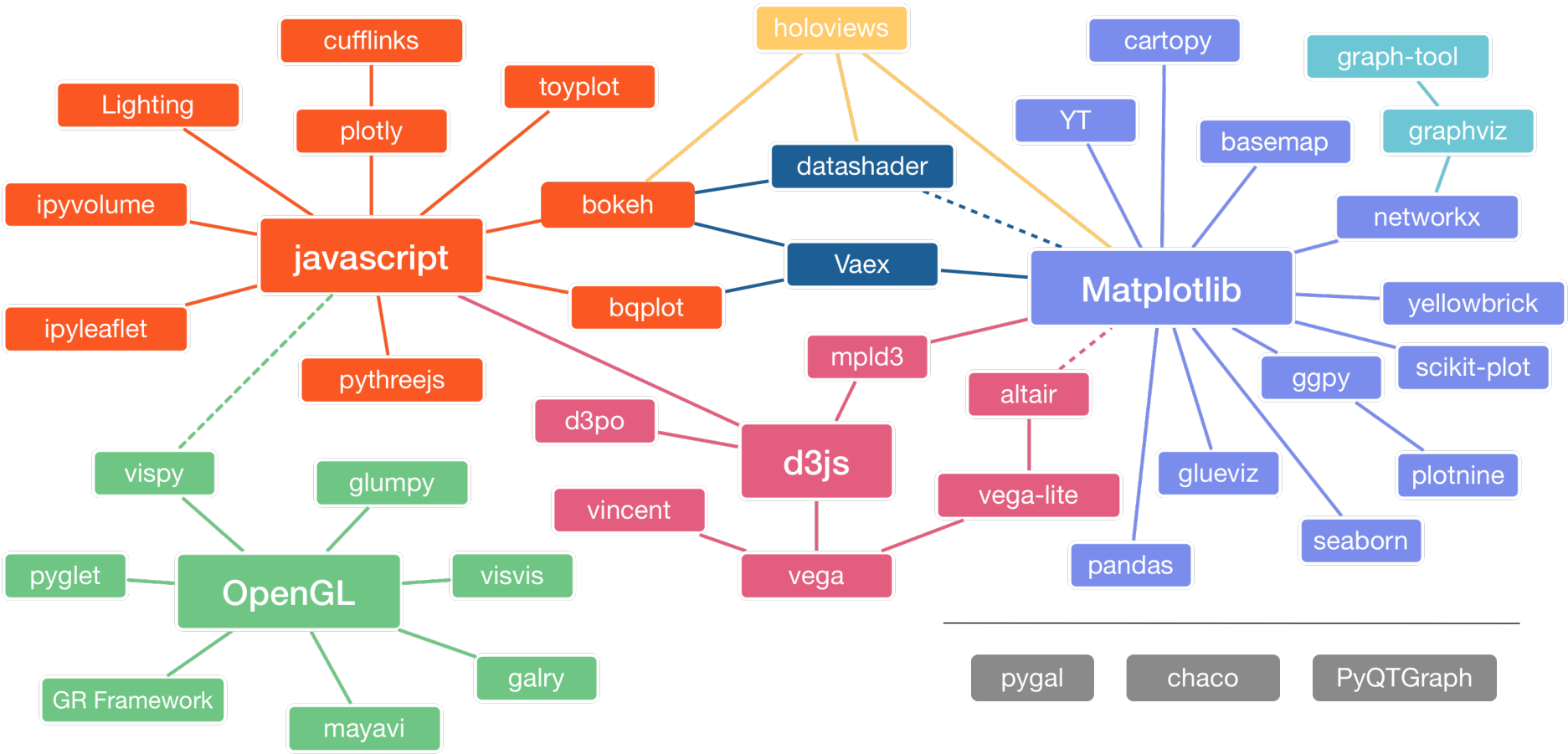
- Sci-vis is the process of mapping scientific data to VISUAL FORM
  - much easier to understand images than a large set of numbers
  - for interactive data exploration, debugging, communication with peers
- Sample visualizations in CC `https://ccvis.netlify.app`
- *Visualize This* / IEEE SciVis contests (since 2016)
  `https://ccvis.netlify.app/contests`

FIELD	VISUALIZATION TYPE
computational fluid dynamics	2D/3D flows, density, temperature, tracers
climate, meteorology, oceanography	fluid dynamics, clouds, chemistry, etc.
astrophysics	2D/3D fluids, particle data, $\leq$6D radiation field, magnetic fields, gravitational fields
quantum chemistry	wave functions
molecular dynamics (phys, chem, bio)	particle/molecular data
medical imaging	MRI, CT scans, ultrasound
geographic information systems	elevation, rivers, towns, roads, layers, etc.
bioinformatics	networks, trees, sequences
humanities, social sciences, info-vis	abstract data, or any of the above

1D/2D plotting vs. multi-dimensional visualization

- **1D/2D plotting**: plotting functions of one variable, 1D tabulated data
  - something as simple as gnuplot or pgplot
  - highly recommend: Python's Matplotlib, Plotly, Bokeh libraries
  - another excellent option: R's ggplot2 library
- **2D/3D visualization**
  - displaying multi-dimensional datasets, typically data on structured (uniform and multi-resolution) or unstructured grids (that have some topology in 2D/3D)
  - rendering often CPU- and/or GPU-intensive
- Whatever you do, may be a good idea to avoid proprietary tools, unless those tools provide a clear advantage (most likely not)
  - large $$
  - limitations on where you can run them, which machines/platforms, etc.
  - cannot get help from open-source community, user base usually smaller than for open-source tools
  - once you start accumulating scripts, you lock yourself into using these tools for a long time, and consequently paying $$ on a regular basis

Python visualization landscape broken by renderer

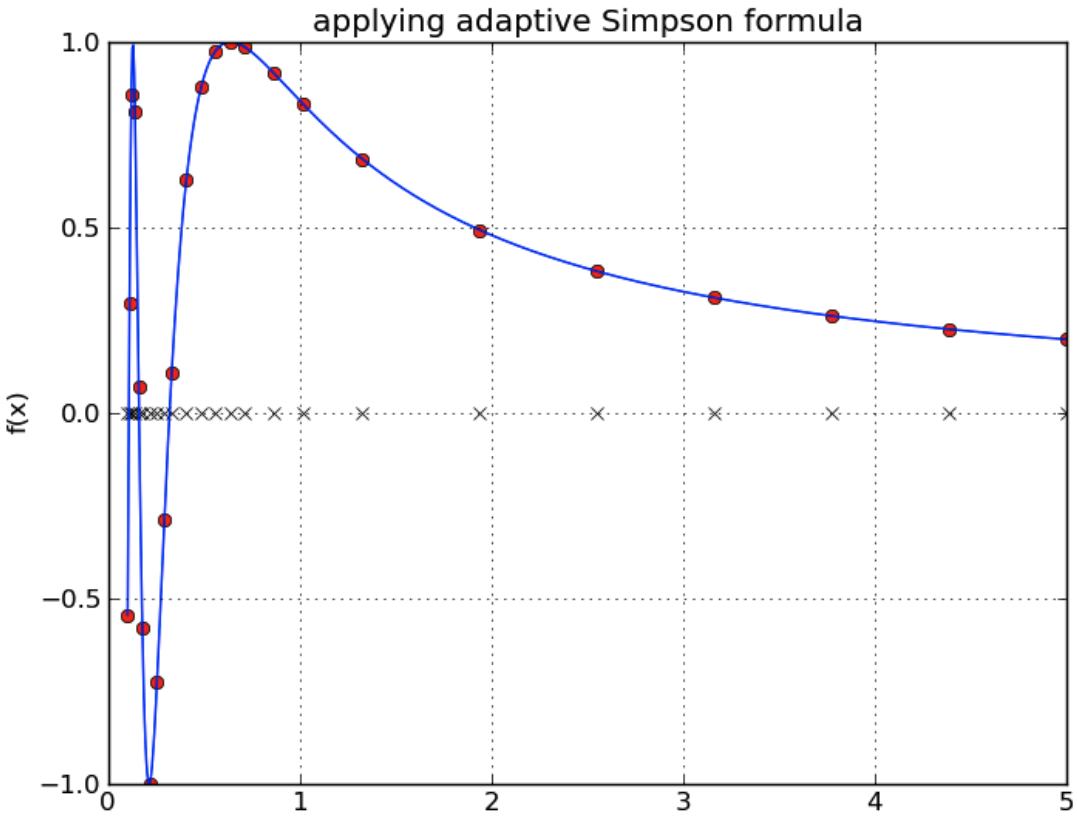


`https://github.com/rougier/python-visualization-landscape` by Nicolas Rougier

This figure is heavily influenced by 1D/2D plotting ... shows only a handful of 3D tools

Matplotlib example: 1D plotting

Adaptive Simpson integration



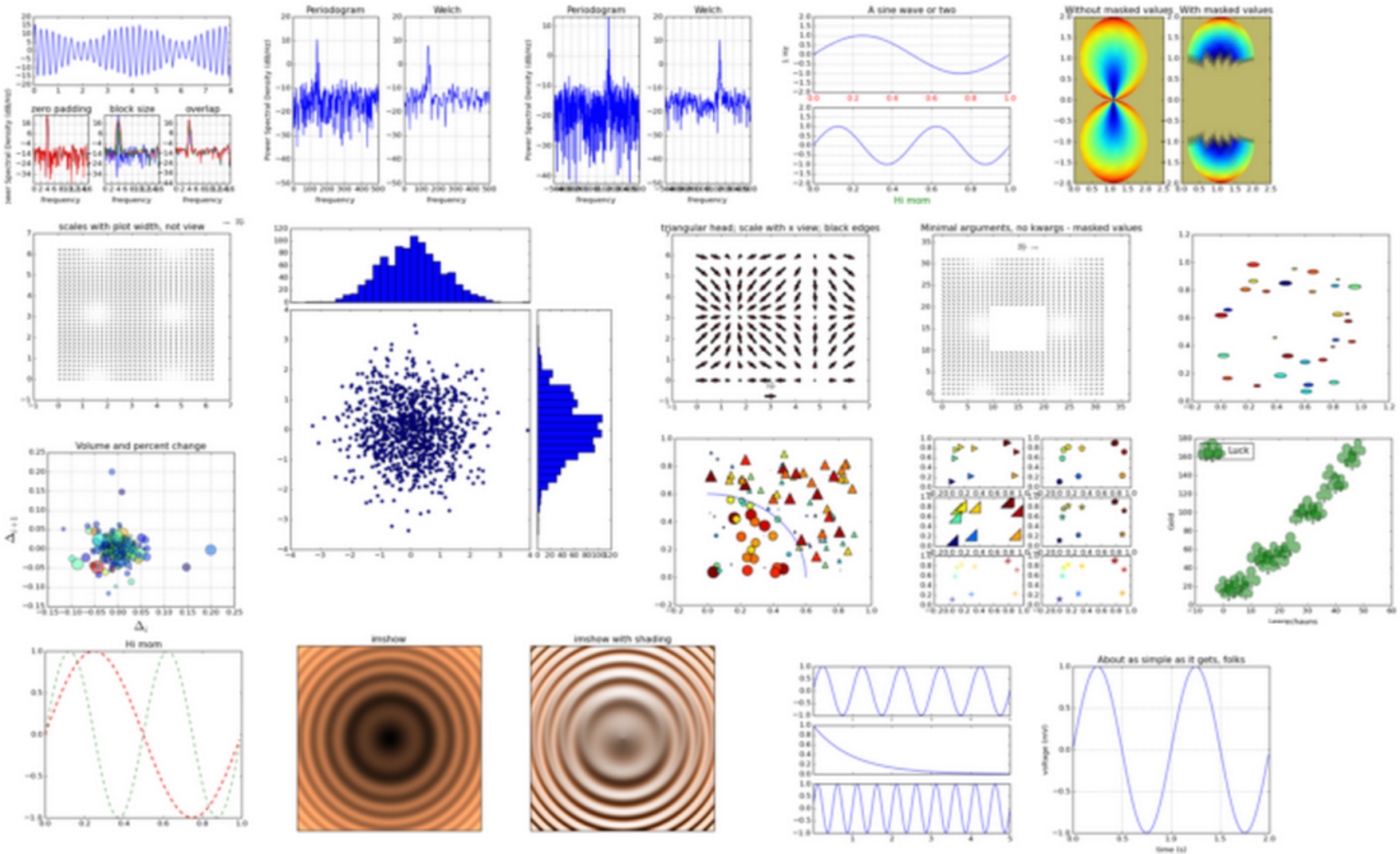
- Simple Python function `simpsonAdaptive(function,a,b,tolerance)` handles both calculation and plotting (∼40 lines of code)
- Code in `codes/adaptive.py`

Python Imaging Library (PIL) example: 2D plotting

Edge detection using numerical differentiation

- Simple Python script reading a colour PNG image, calculating gradient of the blue filter, plotting its norm in black/white (20 lines of code total)
- Code in `codes/fuji.py`

Matplotlib gallery contains hundreds of examples



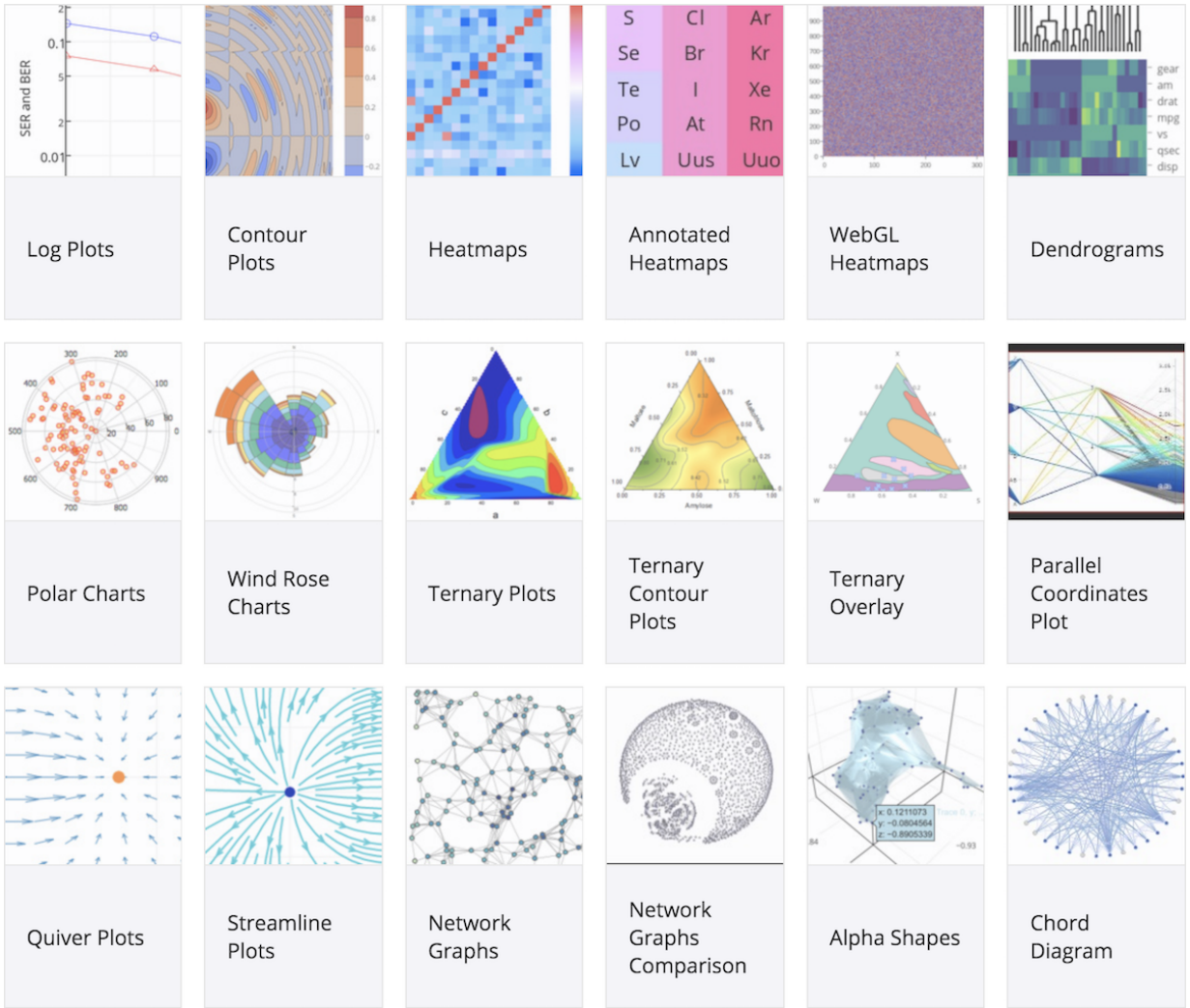
- `http://matplotlib.org/gallery.html` – click on any plot to get its source code
- `http://www.labri.fr/perso/nrougier/teaching/matplotlib` is a really good introduction

Bokeh gallery

- Open-source project from Continuum Analytics
  `https://docs.bokeh.org/en/latest/docs/gallery.html`
- Produces dynamic html5 visualizations for the web
- Basic server-less interactivity packed into a json object; more complex interactions via a Bokeh server

Plotly Python library

- Open-source project from Plot.ly `https://plot.ly/python`
- Produces dynamic html5 visualizations for the web
- APIs for Python (with/without Jupyter), R, JavaScript, MATLAB



- Can work offline (free) or by sending your data to your account on plot.ly (public plotting is free, paid unlimited private plotting and extra tools)

Other popular sci-vis software

- Other excellent plotting libraries in Python: Seaborn, HoloViews, Altair (each deserving its own page in this deck)
- Partial list of other sci-vis tools you may find on large HPC clusters:
  - Avizo for general-purpose 3D visualization; the only commercial package on this list
  - Tablet viewer for genome sequence assembly and alignment
  - Molden for visualization of molecular and electronic structure
  - WebMO web portal for computing/visualization in chemistry
  - XCrySDen for crystalline and molecular structure visualisation
  - GNU Data Language (GDL) for data analysis and visualization in astronomy, geosciences and medical imaging; open-source implementation of IDL
  - GDIS for visualization of molecular and periodic structures
  - Molekel for molecular visualization
  - Ncview visual browser for NetCDF files
  - ParaView for general-purpose scientific visualization
  - Rasmol for molecular visualization
  - VisIt for general-purpose scientific visualization
  - VTK = Visualization Toolkit library
  - VMD for visualization of large biomolecular systems

Today's focus: multi-dimensional sci-vis packages

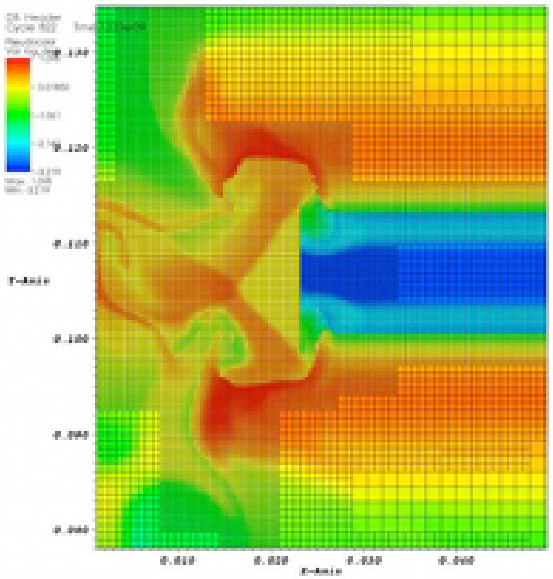
- Open-source + multi-platform + general-purpose + must support the following features:
  - visualize scalar and vector fields
  - structured and unstructured meshes in 2D and 3D, particle data, polygonal data, irregular topologies
  - ability to handle very large datasets (GBs to TBs)
  - ability to scale to $10^{\sim 12}$ resolution elements
  - ability to scale to large ($10^3 - 10^5$ cores) computing facilities
  - interactive manipulation
  - support for scripting
  - support for most common data formats, parallel I/O

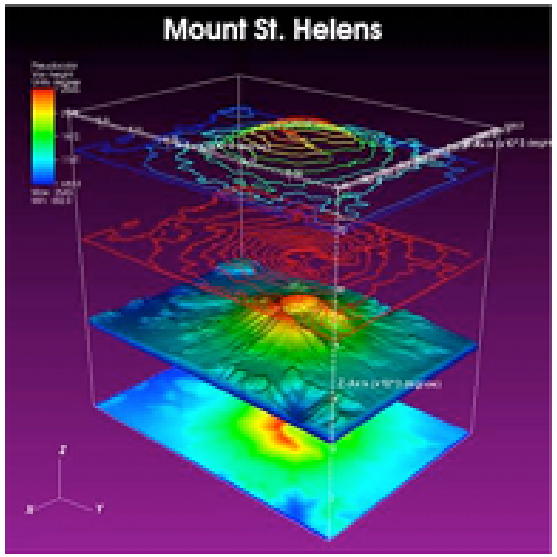
1. VisIt (latest is 3.3)
2. ParaView (latest is 5.10)

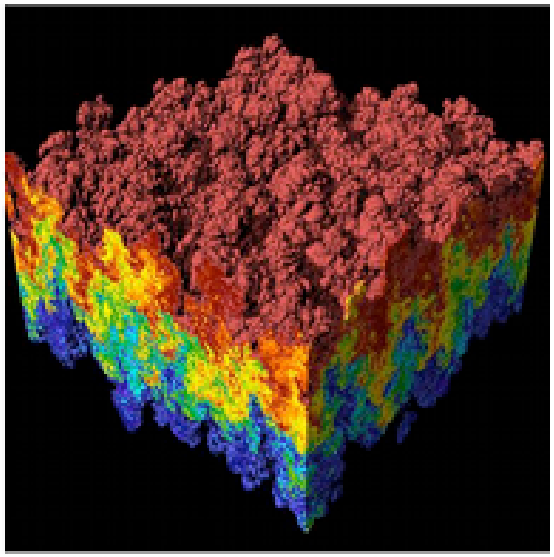
VisIt

https://visit-dav.github.io/visit-website

- Developed by the Department of Energy (DOE) Advanced Simulation and Computing Initiative (ASCI) to visualize results of terascale simulations, first public release in fall 2002
- Available as source and binary for Linux/Mac/Windows
- Over 80 visualization features (contour, mesh, slice, volume, molecule, ...)
- Reads over 110 different file formats; APIs for C++, Python, and Java
- Interactive and Python scripting; full integration with VTK library
- Uses MPI for distributed-memory parallelism on HPC clusters





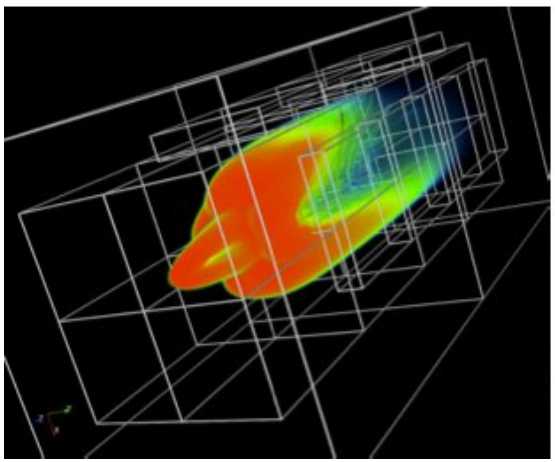
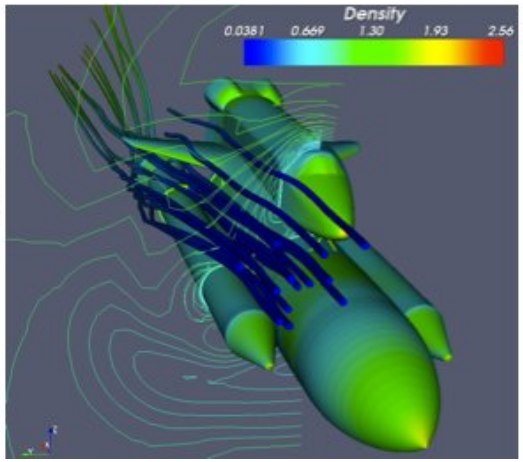


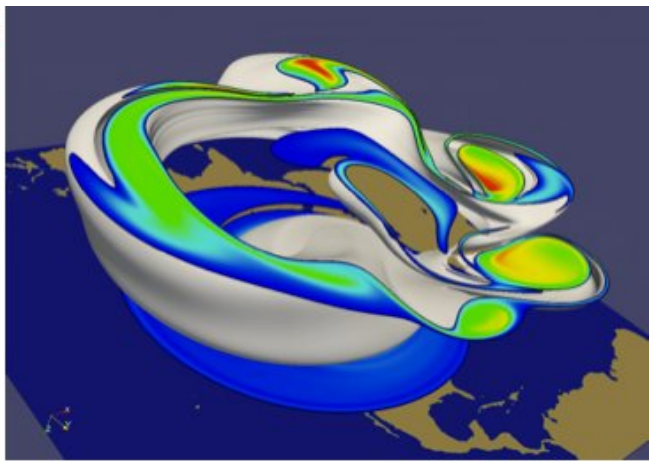
Lawrence Livermore National Laboratory

ParaView

`http://www.paraview.org` and `https://github.com/Kitware/ParaView`

- Started in 2000 as a collaboration between Los Alamos NL and Kitware Inc., later joined by Sandia NL and other partners; first public release in 2002
- Available as source and pre-compiled binary for Linux/Mac/Windows
- To visualize extremely large datasets on distributed memory machines
- Both interactive and Python scripting
- Client-server for remote interactive visualization
- Uses MPI for distributed-memory parallelism on HPC clusters
- ParaView is based on VTK (Visualization Toolkit)
  - not the only VTK-based open-source scientific renderer, e.g. VisIt, MayaVi (Python + numpy + scipy + VTK), an of course a number of Kitware's own tools besides ParaView are based on VTK
  - VTK can be used from C++, Python, and now JavaScript as a standalone renderer



Why ParaView for this workshop?

- Back in ∼2010, I had to pick one
  - both ParaView's and VisIt's binaries are widely available, in active development
  - both can do remote client-server visualization, very good parallel scalability
  - ParaView and VisIt interfaces are very different
- Tight integration with VTK (developed by the same folks), 130 input formats
- A number of add-on projects
  - **ParaViewWeb** is a JavaScript library to write web applications that talk to a remote ParaView server; can reproduce full standalone ParaView in a web browser (WebGL + remote processing)
  - **vtk.js** is a scientific rendering library for the web (standalone WebGL)
  - **ParaView Glance** is a standalone open-source web app for in-browser 3D sci-vis
  - **Catalyst** is an open-source *in-situ visualization* library that can be embedded directly into parallel simulation codes; interaction through ParaView scripts
  - **ParaView Cinema** for interactive visualization from pre-rendered images (rotation, panning, zooming, variables on/off)
- Stereo viewing on 3D hardware; experimental builds for various head-mounted displays (HMDs), Looking Glass displays
- We also use and promote `https://visit-dav.github.io/visit-website`, other open-source packages

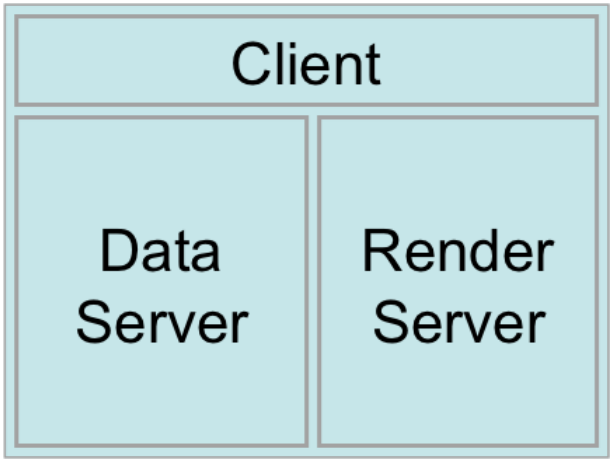
ParaView architecture and GUI

ParaView's distributed parallel architecture

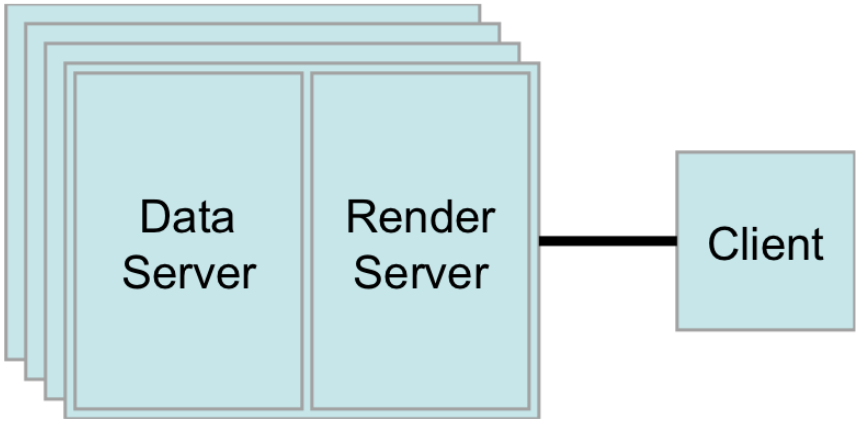
Three logical components inside ParaView – these units can be embedded in the same application on the same computer, but can also run on different machines:

- **Data Server** – The unit responsible for data reading, filtering, and writing. All of the pipeline objects seen in the pipeline browser are contained in the data server. The data server can be parallel.
- **Render Server** – The unit responsible for rendering. The render server can also be parallel, in which case built-in parallel rendering is also enabled.
- **Client** – The unit responsible for establishing visualization. The client controls the object creation, execution, and destruction in the servers, but does not contain any of the data, allowing the servers to scale without bottlenecking on the client. If there is a GUI, that is also in the client. The client is always a serial application.

Two major workflow models



Standalone mode: computations and user interface run on the same machine



Client-server mode: `pvserver` on a multi-core server or distributed cluster

Advantages of remote client-server rendering

- Standalone ParaView has its limitations: limited memory, limited I/O bandwidth, and limited CPU / GPU power
- For example, on a workstation with 48GB memory works well up to 2048^3 single-precision float variable on structured grids stored locally
- Larger / higher-res datasets, more complex grids, or datasets requiring complex filters won't fit
- Typical problem that won't fit on a 48GB workstation and is too slow to read via sshfs: simulation of the airflow around a wing on an *unstructured grid* (*.vtu) with 246×10^6 cells (equiv. to 627^3), one variable takes 25GB — however, can do this interactively without problem on 64 cores on a cluster with `pvserver` taking $\sim$ 120 GB memory
- We'll study remote ParaView in more detail towards the end of this workshop

Starting ParaView

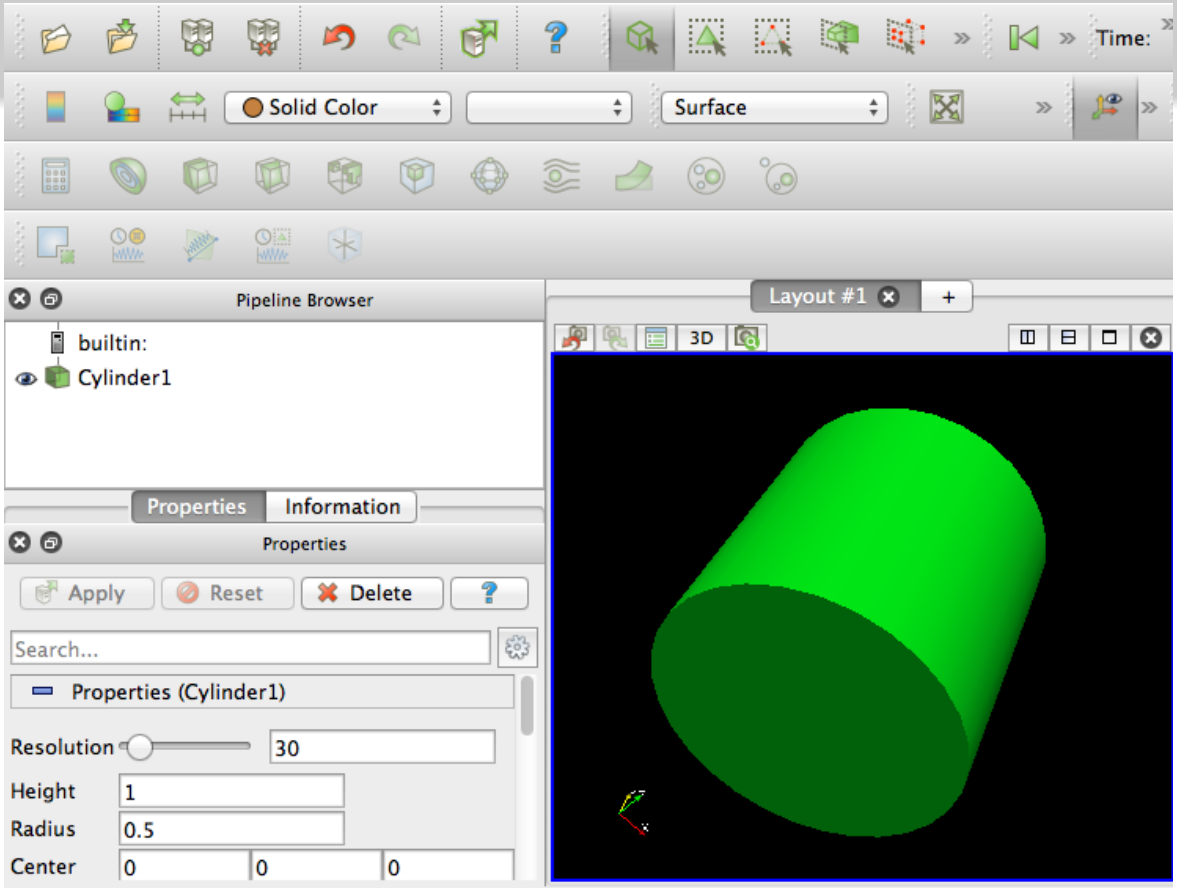
- Today we'll do everything in standalone ParaView on your desktop

- Linux/Unix: type paraview at the command line
- Mac: click paraview in Applications folder
- Windows: select paraview from start menu

- ParaView GUI should start up
- The server `pvserver` is run for you in the background

User interface

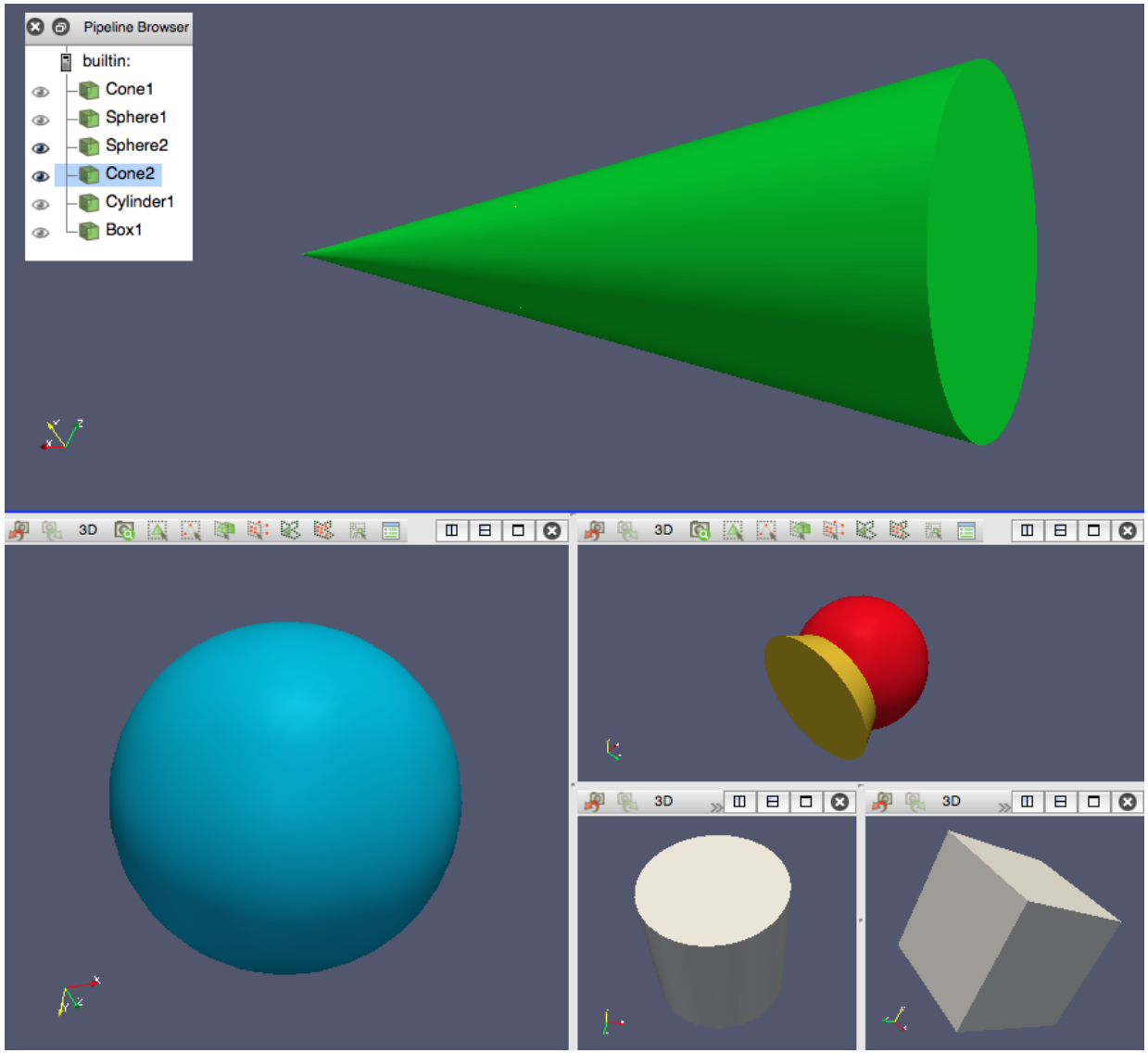
- **Pipeline Browser**: data readers, data filters, can turn visibility of each object on/off
- **Object Inspector**: view and change parameters of the current pipeline object (via tabs properties-display-info or properties-info)
- **View window**: displays the result



1. Find the following in the toolbar: “Connect”, “Disconnect”, “Toggle Colour Legend Visibility”, “Edit Colour Map”, “Rescale to Data Range”
2. Load a predefined dataset: in ParaView select Sources → Cylinder
3. Try dragging the cylinder using the left mouse button; also try the same with the right and middle buttons
4. Identify drop-down menus; try changing to a different view (e.g. from Surface to Wireframe) or changing colour via “Edit Colour Map”

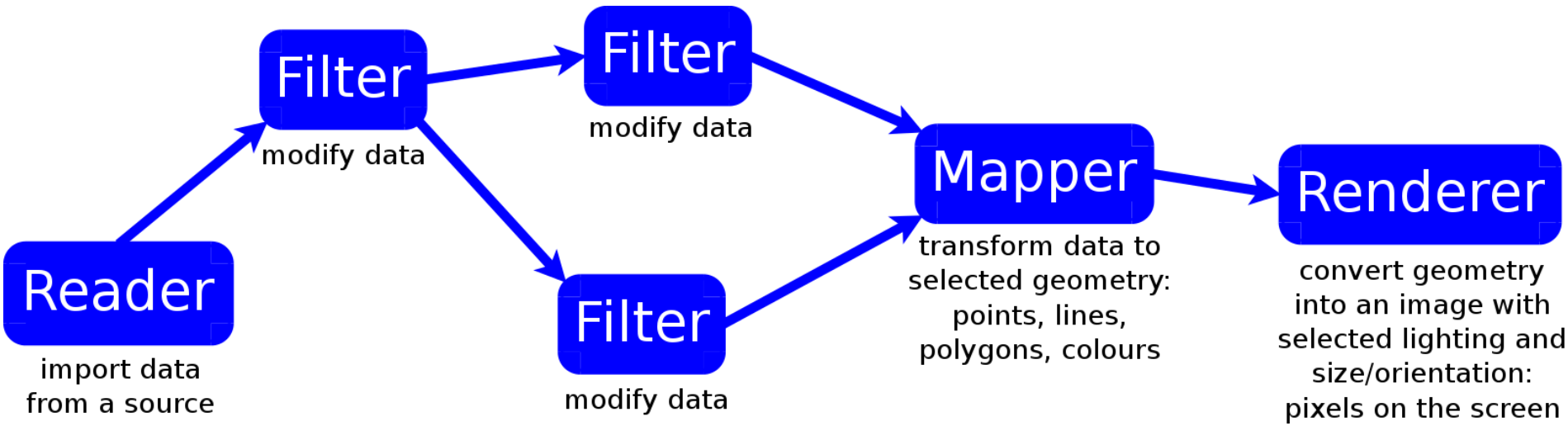
ParaView windows

- Reproduce this image
- Use objects from the "Sources" menu (cone, sphere, cylinder, box), can edit their properties
- Use the icons in the upper right of each window to split the view
- Optionally can link any two views by right-clicking on an image, selecting "Link Camera", and clicking on a second image



Data flow in VTK

https://vtk.org/Wiki/VTK/Tutorials/VTK_Terminology



- Data goes through Mapper which knows how to draw it, places that data into the rendered scene via a VTK Actor
  - `mapper.setInputConnection(object.getOutputPort())`
- Actor is an OpenGL object = the part that is rendered
  - takes data from Mapper: `actor.setMapper(mapper)`
  - passed to Renderer: `renderer.addActor(actor)`
- Renderer can hold multiple actors
- RendererWindow (on the screen) can hold multiple renderers

Importing data into ParaView

Data sources

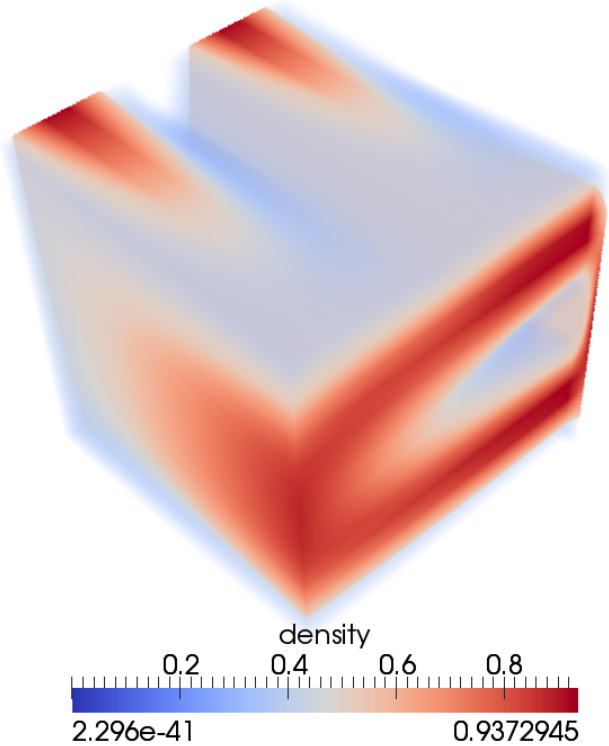
- Generate data with a *Source* object
- Read data from a file

ADAPT Files (*.nc *.cdf *.elev *.ncd)
Adaptive cosmo files (*.cosmo)
ADIOS2 BP3 File (Corelmage) (*.bp)
ADIOS2 BP4 Directory (Corelmage) (*.bp)
ADIOS2 BP4 Metadata File (Corelmage) (md.idx)
AMR Enzo Files (*.boundary *.hierarchy)
AMR Flash Files (*.Flash *.flash)
AMR Velodyne Files (*.xarnr *.Xamr *.XAMR)
AMReX/BoxLib plotfiles (grids) (plt*)
AMReX/BoxLib plotfiles (particles) (plt*)
ANSYS Files (*.inp)
AU File Files (*.aux)
AVS UCD Binary/ASCII Files (*.inp)
BOV Files (*.bov)
BYU Files (*.g)
CAM NetCDF (Unstructured) (*.nc *.ncdf)
Case file for restarted CTH outputs
CCSM MTSD Files (*.nc *.cdf *.elev *.ncd)
CCSM STSD Files (*.nc *.cdf *.elev *.ncd)
CEAucd Files (*.ucd *.inp)
CGNS Files (*.cgns)
Chombo Files (*.hdf5 *.h5)
CityGML files (*.gml *.xml)
Claw Files (*.claw)
CMAT Files (*.cmat)
CML (*.cml)
CONVERGE CFD (*.h5)
Cosmology Files (*.cosmo64 *.cosmo)
CTRL Files (*.ctrl)
Curve2D Files (*.curve *.ultra *.ult *.u)
DDCMD Files (*.ddcmd)
Delimited Text (*.csv *.tsv *.txt *.CSV *.TSV *.TXT)
DICOM Files (directory) (*.dcm)
DICOM Files (single) (*.dcm)
Digital Elevation Map Files (*.dem)
Dyna3D Files (*.dyn)
EnSight Files (*.case *.CASE *.Case)
EnSight Master Server Files (*.sos *.SOS)
ENZO AMR Particles (*.boundary *.hierarchy)
Exodus11 (*.g *.gc *.ex2 *.ex2v2 *.exo *.gen *.par
ExtrudedVol Files (*.exvol)
Facet Polygonal Data Files (*.facet)
Fides Data Model File (JSON) (*.json)
Fides Files (ADIOS2 BP) (*.bp)
FLASH AMR Particles Reader (*.Flash *.flash)
FLASH Files (Visit) (*.flash *.f5)
Fluent Case Files (*.cas)
Fluent Files (Visit) (*.cas)
FVCOM MTMD Files (*.nc *.cdf *.elev *.ncd)
FVCOM MTSD Files (*.nc *.cdf *.elev *.ncd)
FVCOM Particle Files (*.nc *.cdf *.elev *.ncd)
FVCOM STSD Files (*.nc *.cdf *.elev *.ncd)
Gadget Files (*.gadget)
Gaussian Cube Files (*.cube)
GDAL Raster (*.tif *.gen *.thf *.adf *.arg *.blx *.xlb)
GDAL Vector (*.shp *.faa *.bnn *.dxf *.csv *.geojson)
Generic10 files to MultiBlockDataSet (*.gio)
Generic10 files to UnstructuredGrid (*.gio)
GGCM Files (*.3df *.mer)
gITF 2.0 Files (*.gltf *.glb)
GTC Files (*.h5)
GULP Files (*.trg)
H5Nimrod Files (*.h5nimrod)
H5Part particle files (*.h5part)
HyperTreeGrid (*.htg)
HyperTreeGrid (partitioned) (*.phtg)
Image Files (*.pnm *.ppm *.sdt *.spr *.imgvol)
JPEG Ima
LAMMPS Dump Files (*.dump)
LAMMPS Struct. (*.eam *.meam *.rigid *.lammps)
Legacy VTK files (*.vtk *.vtk.series)
Legacy VTK Files (partitioned) (*.pvtk)
Lines Files (*.lines)
LODI Files (*.no *.cdf *.elev *.ncd)
LODI Particle Files (*.nc *.cdf *.elev *.ncd)
LSDyna (*.k *.lsdyna *.d3plot d3plot)
M3DC1 Files (*.h5)
Meta Image Files (*.mhd *.mha)
Meta-Generic10 files (*.gios)
Metafile for restarted exodus outputs
MFiX netcdf Files (*.nc)
MFiX Res Files (Visit) (*.RES)
MFIX Unstructured Grid Files (*.RES)
Mill Files (*.m)
Miranda Files (*.mir *.raw)
MM5 Files (*.mm5)
MotionFX CFG Files (*.cfg)
MPAS NetCDF (Unstructured) (*.ncdf *sic)
MRC Image Files (*.mrc *.ali *.st *.rec)
Multilevel 3D Plasma Files (*.m3d *.h5)
NASTRAN Files (*.nas *.f06)
Nek5000 (*.nek3d *.nek2d *.nek5d *.nek5000 *.nek)
netCDF generic and CF conventions (*.ncdf *.nc)
Nrrd Raw Image Files (*.nrrd *.nhdr)
OME TIFF Files (*.ome.tif *.ome.tiff)
OpenFOAM (*.foam)
OpenFOAM Files (Visit) (*.controlDict)
openPMD files (*.pmd)
OVERFLOW Files (Visit) (*.dat *.save)
ParaDIS Files (*.prds *.data *.dat)
ParaDIS Tecplot (Mid *.field *.cyl *.cylinder *.dat)
Parallel POP Ocean NetCDF (*.pop.ncdf *.pop.nc)
ParaView Data Files (*.pvd)
ParaView Ensemble Data (*.pve)
PATRAN Files (*.neu)
PFLOTRAN Files (*.h5)
Phasta Files (*.pht)
PIO Dump Files (*.pio)
Pixie Files (*.h5)
PLOT2D Files (*.p2d)
PLOT3D Files (*.xyz)
PLOT3D Meta Files (*.p3d)
PLY Polygonal File Format (*.ply *.ply.series)
PNG Image Files (*.png)
POINT3D Files (*.3D)
POP Ocean NetCDF Rectilinear (*.pop.nc*)
POP Ocean NetCDF Unstructured (same as prev.)
proSTAR Files (*.cel *.vrt)
Protein Data Bank Files (*.pdb)
Protein Data Bank Files (Visit) (*.ent *.pdb)
PTS (Point Cloud) Files (*.pts)
Radiance HDR file (*.hdr)
Raw (binary) Files (*.raw)
RAW Files (*.raw)
SAMRAI series files (*.samrai)
SAR Files (*.SAR *.sar)
SAS Files (*.sasgeom *.sas *.sasdata)
SEG-Y Files (*.sgy *.segy)
SEP file (Plugin) (*.H)
Silo Files (*.silo *.pdb *.silo.series *.pdb.series)
SLAC Mesh Files (*.ncdf *.nc)
SLAC Particle Files (*.ncdf *.netcdf)
Spheral Files (*.spheral *.sv)
Spy Plot History Files (*.hscth *hsct*)
SpyPlot CTH dataset (*.spct* spot*)
Stereo Lithography (*.stl *.stl.series)
Tecplot Binary Files (Visit) (*.plt)
Tecplot Files (*.tee *.TEC *.Tec *.tp *.TP *.dat)
Tecplot Files (Visit) (*.tec *.TEC *.Tec *.tp *.TP)
Tecplot Table (*.dat *.DAT)
Tetrad Files (*.hdf5 *.h5)
TFT Files (*.dat *.tft)
TIFF Image Files (*.tif *.tiff)
TRUCHAS dataset (*.hdf5 *.h5)
TSurf Files (*.ts_deg83)
UNIC Files (*.h5)
VASP Animation Files (*.out)
VASP CHGCA Files (*.CHG*)
VASP OUT Files (*.OUT*)
VASP POSCAR Files (*.POS*)
VASP Tessellation Files (*.out)
Velodyne Files (*.vld *.rst)
Visit MetaPLOT3D Files (Visit) (*.vp3d)
VizSchema Files (*.h5 *.vsh5)
VPIC Files (*.vpc)
VRML 2 Files (*.wrl *.vrml)
VTK Hierarchical Box Data Files (*.vthb)
VTK ImageData Files (*.vti *.vti.series)
VTK ImageData Files (partitioned) (*.pvti)
VTK MultiBlock Data Files (*.vtm *.vtmb)
VTK Particle Files (*.particles)
VTK Partitioned Dataset Collection Files (*.vtpc)
VTK Partitioned Dataset Files (*.vtpd *.vtpd.series)
VTK PolyData Files (*.vtp *.vtp.series)
VTK PolyData Files (partitioned) (*.pvtp)
VTK RectilinearGrid Files (*.vtr *.vtr.series)
VTK RectiiinearGrid Files (partitioned) (*.pvtr)
VTK StructuredGrid Files (*.vts *.vts.series)
VTK StructuredGrid Files (partitioned) (*.pvts)
VTK Table (partitioned) (*.pvtt *.pvtt.series)
VTK Table Files (*.vtt *.vtt.series)
VTK UnstructuredGrid Files (*.vtu *.vtu.series)
VTK UnstructuredGrid Files (partitioned) (*.pvtu)
VTX reader: ADIOS2 BP3 File (*.bp)
VTX reader: ADIOS2 BP4 Directory (*.bp *.bp4)
Wavefront OBJ Files (*.obj)
WindBlade Data (*.wind)
Xdmf Reader (*.xmf *.xdmf *.xmf2 *.xdmf2)
Xdmf3 Reader (*.xmf *.xdmf *.xmf3 *.xdmf3)
Xdmf3 Reader (Top Level Partition) (*.xmf* *.xdmf)
Xmdv Files (*.okc)
XMol Molecule Files (*.xyz)
XYZ Files (*.xyz)

Example: reading raw (binary) data

Show $f(x,y,z) = (1-z)\left[(1-y)\sin(\pi x) + y\sin^2(2\pi x)\right]$
$+z\left[(1-x)\sin(\pi y) + x\sin^2(2\pi y)\right]$ in $x, y, z \in [0,1]$ sampled at 16^3

1. File: `data/simpleData.raw` – load it as RAW BINARY
   Update: starting with ParaView 5.9, this no longer works!
2. Describe the dataset in properties:
   - Data Scalar Type = float
   - Data Byte Order = Little Endian
   - File Dimensionality = 3
   - Data Extent: 0 to 15 in each dimension (1 to 16 will crash recent ParaView versions)
   - Scalar Array Name = density
3. Try different views: Outline, Points, Wireframe, Volume
4. Depending on the view, can edit the colour map



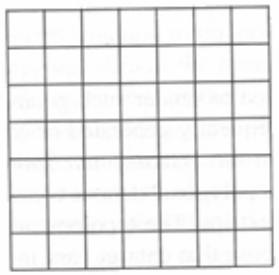
5. Try saving data as paraview data type (*.pvd), deleting the object, and reading back from *.pvd – file now contains full description of dataset

VTK = Visualization Toolkit

- Open-source software system for 3D computer graphics, image processing and visualization
- Bindings to C++, Tcl, Java, Python
- ParaView is based on VTK ⇒ supports all standard VTK file formats
- VTK file formats
  `http://www.vtk.org/VTK/img/file-formats.pdf`
  - legacy serial format (*.vtk): ASCII header lines + ASCII/binary data
  - XML formats (extension depends on VTK data type): XML tags + ASCII/binary/compressed data
    - newer, much preferred to legacy VTK
    - supports parallel file I/O, compression, portable binary encoding (big/little endian byte order), random access, etc.

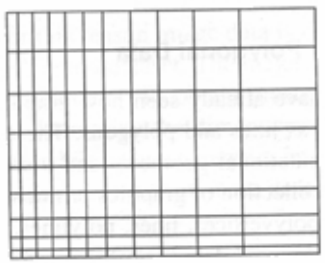
VTK 3D data: 6 major dataset (discretization) types

- **Image Data/Structured Points**: *.vti, points on a regular rectangular lattice, scalars or vectors at each point



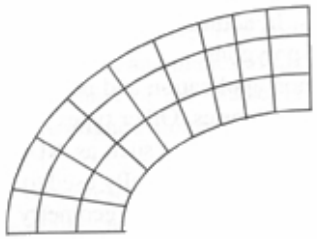
(a) Image Data

- **Rectilinear Grid**: *.vtr, same as Image Data, but spacing between points may vary, need to provide steps along the coordinate axes, not coordinates of each point



(b) Rectilinear Grid

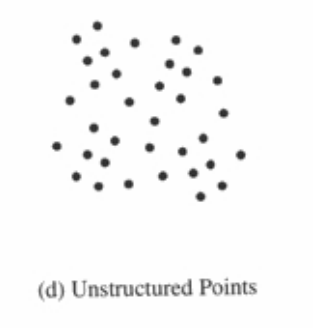
- **Structured Grid**: *.vts, regular topology and irregular geometry, need to indicate coordinates of each point



(c) Structured Grid

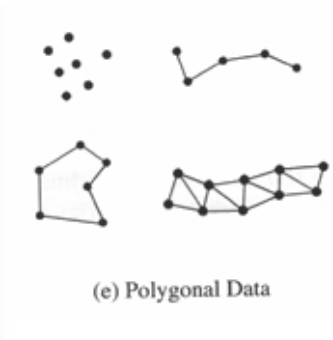
VTK 3D data: 6 major dataset (discretization) types

- **Particles/Unstructured Points**: *.particles



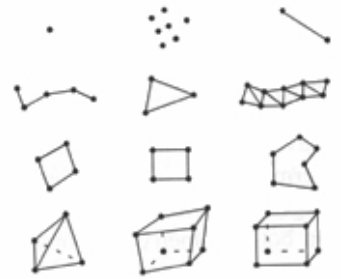
(d) Unstructured Points

- **Polygonal Data**: *.vtp, unstructured topology and geometry, point coordinates, 2D cells only (i.e. no polyhedra), suited for maps



(e) Polygonal Data

- **Unstructured Grid**: *.vtu, irregular in both topology and geometry, point coordinates, 2D/3D cells, suited for finite element analysis, structural design



(f) Unstructured Grid

VTK 3D data: dataset attributes

A VTK file can store a number of datasets, each could be of one of the following types:

- Scalars: single valued, e.g. density, temperature, pressure
- Vectors: magnitude and direction, e.g. velocity
- Normals: direction vectors ($|\mathbf{n}| = 1$) used for shading
- LookupTable: each entry in the lookup table is a red-green-blue-alpha array (alpha is opacity: alpha=0 is transparent); if the file format is ASCII, the lookup table values must be float values in the range [0,1]
- TextureCoordinates: used for texture mapping
- Tensors: 3×3 real-valued symmetric tensors, e.g. stress tensor
- FieldData: array of data arrays

Example: reading legacy VTK

Caution: storing large datasets in ASCII is not a very good idea – here we look at text-based VTK files for instructional purposes

1. File: `data/volume.vtk`
   - simple example (Structured Points): $3 \times 4 \times 6$ dataset, one scalar field, one vector field
2. File: `data/density.vtk`
   - another simple example (Structured Grid): $2 \times 2 \times 2$ dataset, one scalar field
3. File: `data/cube.vtk`
   - more complex example (Polygonal Data): cube represented by six polygonal faces. A single-component scalar, normals, and field data are defined on all six faces (CELL_DATA). There are scalar data associated with the eight vertices (POINT_DATA). A lookup table of eight colours, associated with the point scalars, is also defined.

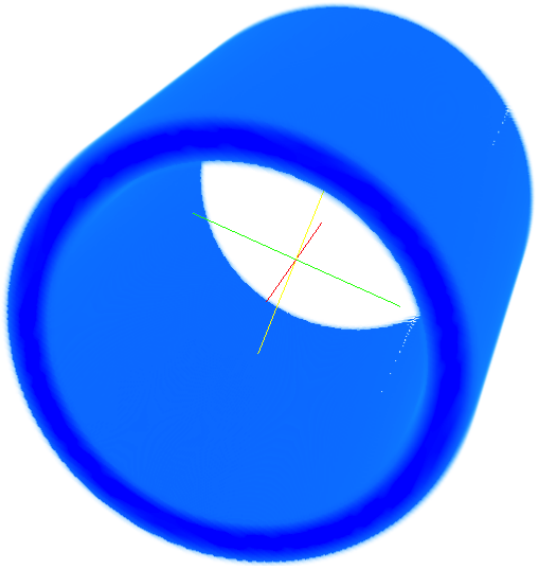
Exercise: visualizing 3D data with legacy VTK

- Visualize a 3D "cylinder" function

$$f(x, y, z) = e^{-|r-0.4|}$$

where $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$,
inside a unit cube ($x, y, z \in [0, 1]$)

➠ reproduce the view on the right



- ASCII data in `data/cylinder.dat` (discretized on a 30^3 Cartesian mesh)
- Add an appropriate header to create a VTK file using `data/volume.vtk` as template

Writing XML VTK from C++

Let's turn to larger datasets (MB, GB) – we should store them as binary

- A good option is to use XML VTK format with binary data and XML metadata, calling VTK library functions from C++ / Java / Python to write data
- Here is an example: `codes/SGrid.cpp` and `codes/Makefile`, generates the file `data/halfCylinder.vts`

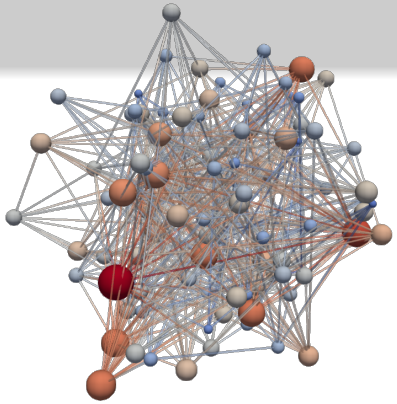
  This example shows how to create a Structured Grid, set grid coordinates, fill the grid with a scalar and a vector, and write it in XML VTK to a *.vts file.
- To run it, you need the VTK C++ library installed (either standalone or pulled from ParaView); check `codes/Makefile` to see the required library files

```
cd codes
make SGrid
(on Linux: export LD_LIBRARY_PATH=/path/to/vtk/lib:$LD_LIBRARY_PATH)
(on a Mac: export DYLD_LIBRARY_PATH=$HOME/Documents/local/vtk/lib)
./SGrid
```

- Many more examples included with the VTK source code or at `http://www.vtk.org/Wiki/VTK/Examples/Cxx`

Writing XML VTK from Python

```
$ pip install vtk networkx
```



Check out `codes/{writeNodesEdges,randomGraph}.py`

```
def writeObjects(nodeCoords,
                 edges = [],
                 scalar = [], name = '', power = 1,
                 scalar2 = [], name2 = '', power2 = 1,
                 nodeLabel = [],
                 method = 'vtkPolyData',
                 fileout = 'test'):
    """
    Store points and/or graphs as vtkPolyData or vtkUnstructuredGrid.
    Required argument:
    - nodeCoords is a list of node coordinates in the format [x,y,z]
    Optional arguments:
    - edges is a list of edges in the format [nodeID1,nodeID2]
    - scalar/scalar2 is the list of scalars for each node
    - name/name2 is the scalar's name
    - power/power2 = 1 for r~scalars, 0.333 for V~scalars
    - nodeLabel is a list of node labels
    - method = 'vtkPolyData' or 'vtkUnstructuredGrid'
    - fileout is the output file name (will be given .vtp or .vtu extension)
    """
```

```
import networkx as nx
from writeNodesEdges import writeObjects

numberNodes, numberEdges = 100, 500
H = nx.gnm_random_graph(numberNodes,numberEdges)
print('nodes:', H.nodes())
print('edges:', H.edges())

# return a dictionary of positions keyed by node
pos = nx.random_layout(H,dim=3)
# convert to list of positions (each is a list)
xyz = [list(pos[i]) for i in pos]

degree = [d for i,d in H.degree()]
writeObjects(xyz, edges=H.edges(), scalar=degree,
             name='degree', fileout='network')
```

Another option for writing XML VTK from Python

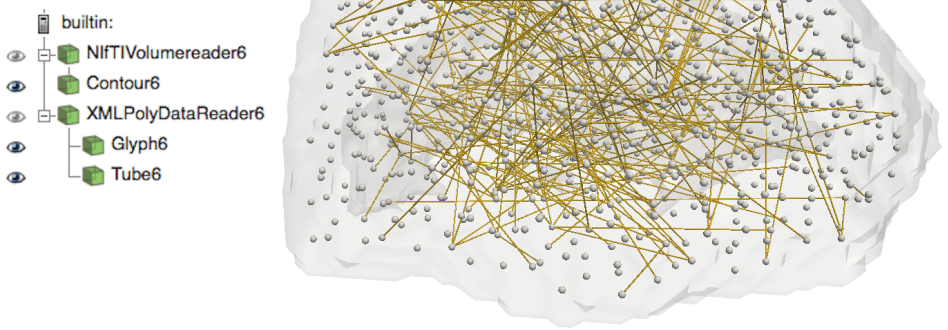
PyEVTK library https://bitbucket.org/pauloh/pyevtk

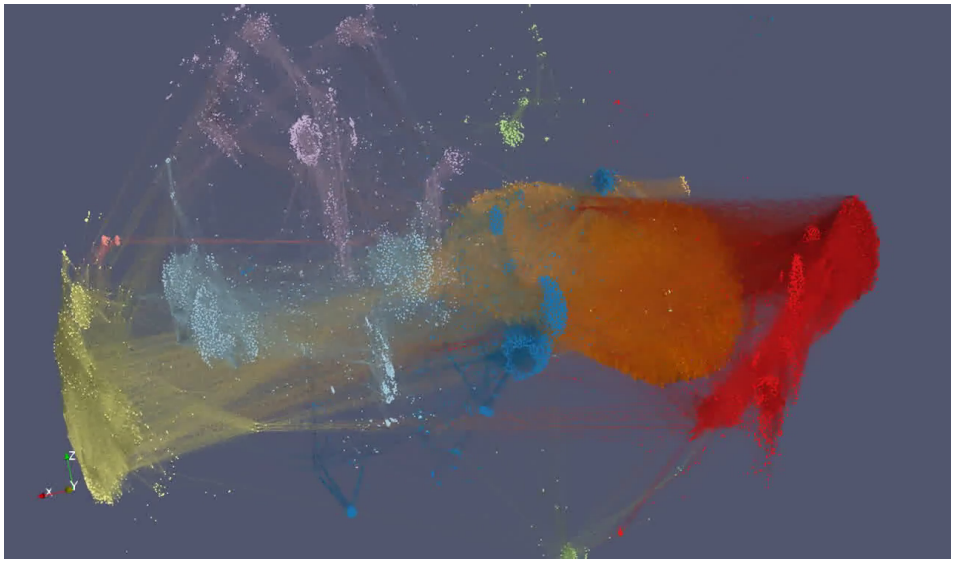
```
$ pip install pyevtk
```

- Works in both Python 2 and Python 3
- Many examples in https://github.com/paulo-herrera/PyEVTK/tree/master/evtk/examples

```
from pyevtk.hl import imageToVTK
from numpy import zeros
n = 30
data = zeros((n,n,n), dtype=float)
for i in range(n):
    x = ((i+0.5)/float(n)*2.-1.)*1.2
    for j in range(n):
        y = ((j+0.5)/float(n)*2.-1.)*1.2
        for k in range(n):
            z = ((k+0.5)/float(n)*2.-1.)*1.2
            data[i][j][k] = ((x*x+y*y-0.64)**2 + (z*z-1.)**2) * \
                            ((y*y+z*z-0.64)**2 + (x*x-1.)**2) * \
                            ((z*z+x*x-0.64)**2 + (y*y-1.)**2)
imageToVTK("decoCube", pointData={"scalar" : data})
```

Think of ParaView as a GUI front end to VTK classes





hidden/mutOnCtOrbits.mp4 on presenter's laptop

```
vtkPoints *points = vtkPoints::New();
for (i=0; i<1028; i++) points->InsertNextPoint(x[i], y[i], z[i]);
vtkCellArray *lines = vtkCellArray::New();
for (j=0; j<degree; j++) { // line from node to adjacent[j]
  lines->InsertNextCell(2);
  lines->InsertCellPoint(node);
  lines->InsertCellPoint(adjacent[j]); }
vtkPolyData* polyData = vtkPolyData::New();
polyData->SetPoints(points);  polyData->SetLines(lines);
vtkSmartPointer<vtkXMLPolyDataWriter> writer = vtkSmartPointer<vtkXMLPolyDataWriter>::New();
writer->SetFileName("output.vtp");  writer->SetInputData(polyData);
writer->Write();
```

NetCDF and HDF5

- VTK is incredibly versatile format, can describe many different data types
- Very often in science one needs to simply store and visualize multi-dimensional arrays
- Problem: how do you store a 2000^3 array of real numbers (30GB of data)?
  - ASCII – forget about it
  - raw binary – possible, but many problems
  - VTK – probably an overkill for simple arrays
- Scientific data formats come to rescue, two popular scientific data formats are NetCDF and HDF5
  - binary (of course!)
  - self-descriptive (include metadata)
  - portable (cross-platform): libraries for many OS's, universal datatypes, byte order in a word (little vs. big endian), etc.
  - support parallel I/O (through MPI-IO)
  - support compression

NetCDF support in ParaView

- NetCDF is supported natively in ParaView
  - `codes/writeNetCDF.cpp` (Fortran version `codes/writeNetCDF.f90`) writes a 100^3 volume with a doughnut shape at the centre in NetCDF

C++ example

```
$ icc writeNetCDF.cpp -o writeNetCDF -I/path/to/netcdf/include \
      -L/path/to/netcdf/lib -lnetcdf_c -lnetcdf
$ ./writeNetCDF
```

F90 example

```
$ ifort writeNetCDF.f90 -o writeNetCDF -I/path/to/netcdf/include \
      -L/path/to/netcdf/lib -lnetcdff -lnetcdf
$ ./writeNetCDF
```

- ParaView understands common NetCDF conventions, e.g., conventions for CF (Climate and Forecast) metadata (`http://cfconventions.org`): 2D or 3D datasets on a sphere, coordinate axes, fill-in values, etc.
  - example 1 on presenter's laptop: 2D dataset `hidden/ice.nc`
  - example 2: snapshot of a 3D dataset `hidden/temp1.png`
  - example 3: more polished 3D visualization `hidden/tempsalt.mp4`

On the subject of spheres ...

How about mapping topography on top of our visualization?

There is a good resource on this
`http://www.earthmodels.org/data-and-tools/topography`

- Option 1: load precomputed topography stored as Polygonal Data
  - e.g., `http://bit.ly/1QIH0lh` (downloads `ETOPO_10min_Ice.vtp`) provides full globe (both land and ocean) at 10 arcmin resolution
  - or `http://bit.ly/1nBKoTN` (downloads `ETOPO_10min_Ice_only-land.vtp`) provides only land at 10'
- Option 2: map a bitmap image to the globe; e.g., `http://bit.ly/1nrghh4` downloads a 8192×4096 image `texture_land_ocean_ice_8192.png`
  1. create a high-resolution Sphere (from Sources)
  2. apply Texture-Map-to-Sphere filter, make sure to click Apply before you can see Miscellaneous:Texture
     - creates "texture coordinates"
     - we haven't studied filters yet
  3. in Properties of the filter: under Miscellaneous:Texture use the drop-down menu to load a PNG image, click Apply
  4. in Properties of the filter: uncheck Prevent Seam at the top, again click Apply
  5. still colouring by Solid Color and viewing as Surface

HDF5 support in ParaView

- No native support for HDF5, however, ParaView supports a container format XDMF (eXtensible Data Model and Format) which uses HDF5 for actual data – only briefly mention it, details at `http://www.xdmf.org`
- XDMF = XML for **light** data + HDF5 for **heavy** data
  - data type (float, integer, etc.), precision, rank, and dimensions completely described in the XML layer (as well as in HDF5)
  - the actual values in HDF5, potentially can be enormous
- Single XML wrapper can reference multiple HDF5 files (e.g. written by each node on a cluster)
- Don't need HDF5 libraries to perform simple operations
- C++ API is provided to read/write XDMF data
- Can be used from Python, Tcl, Java, Fortran through C++ calls
- In Fortran can generate XDMF files with HDF5 calls + plain text for the XML wrapper `http://www.xdmf.org/index.php/Write_from_Fortran`
- Also support for a number of file formats generated by third-party software that in turn use HDF5 underneath

Reading OpenFOAM 2.3.x datasets

✔ ParaView can read *.controlDict and *.foam files (**File** → **Open**) but these are not present in OpenFOAM's output; can create an empty `case.foam` file in the case directory and load it into ParaView

- most of the time it works – when it does not, the error can be traced to `VTK/IO/Geometry/vtkOpenFOAMReader.cxx` in ParaView's source code

✘ Use OpenFOAM's built-in **foamToVTK** utility to convert data from OpenFOAM format to VTK – works with all versions of ParaView

✘ Use OpenFOAM-supplied ParaView reader module libraries PV4FoamReader and vtkPV4Foam with precompiled ParaView 4.x.x through **paraFoam** launch script

- tested with precompiled binary ParaView 4.x.x
- no need to compile anything (contrary to OpenFOAM documentation!)
- not 100% compatible with ParaView Python scripting

✘ Deprecated: recompile ParaView with a third-party (not from OpenFOAM or ParaView) **vtkPOFFReader** plugin – crashes newer ParaView; officially the plugin was written for ParaView 3.10 - 3.14

✘ Deprecated: use the same OpenFOAM-supplied reader libraries PV4FoamReader and vtkPV4Foam with bundled third-party software pack **ThirdParty-2.3.1** that includes an older ParaView-4.1.0; requires compilation of ParaView with OpenFOAM's unconventional build scripts

Reading OpenFOAM: using foamToVTK utility

- Assuming you have OpenFOAM installed:

```
<submit an interactive job on the cluster and wait for the prompt>
module load application/OpenFOAM/2.3.0    # or similar
source $OPENFOAM_SETUP
<change to the case directory containing system/, constant/,
  processorXXX/, time outputs>
foamToVTK                       # to process everything
foamToVTK -latestTime           # to process the last frame in the model
foamToVTK -time 2.98:2.99       # to process a range of timesteps
foamToVTK -time 9.39:           # to process a range of timesteps
```

- This will create a VTK subdirectory with one main VTK file per timestep containing the 3D volume, and auxiliary VTK files describing the boundaries
- Next simply load the main VTK files into ParaView and script a movie with ParaView's Python (more on scripting/animation later)

Reading OpenFOAM: paraFoam script on MacOS

Step 1: install and configure paraFoam

```
brew install open-mpi scotch cgal
brew install boost --without-single --with-mpi
cd ~/Downloads && mkdir OpenFOAM && cd OpenFOAM
wget http://downloads.sourceforge.net/foam/OpenFOAM-2.3.1.tgz
tar xvfz OpenFOAM-2.3.1.tgz && cd OpenFOAM-2.3.1
wget https://raw.githubusercontent.com/mrklein/openfoam-os-x/master/\
OpenFOAM-2.3.1.patch
patch -p1 < OpenFOAM-2.3.1.patch
```

Step 2: launch paraFoam

```
<change to the case directory containing system/, constant/,
 processorXXX/, time outputs>
export FOAM_INST_DIR=~/Downloads/OpenFOAM
source $FOAM_INST_DIR/OpenFOAM-2.3.1/etc/bashrc
paraFoam      # launches ParaView, points it to a sequence
              # of time step files, loads the first time step
```

Warning: paraFoam script is not entirely compatible with ParaView's Python
(can't use the trace tool to reproduce paraFoam customization)

Reading OpenFOAM: paraFoam script on a cluster

Step 1: install and configure paraFoam (for system-wide or your own ParaView)

```
cd /scratch2/razoumov
wget http://downloads.sourceforge.net/foam/OpenFOAM-2.3.1.tgz
tar xvfz OpenFOAM-2.3.1.tgz && cd OpenFOAM-2.3.1
export FOAM_INST_DIR=/scratch2/razoumov
sed -i -e 's|4.1.0|4.3.1|' $FOAM_INST_DIR/OpenFOAM-2.3.1/etc/config/paraview.sh
source $FOAM_INST_DIR/OpenFOAM-2.3.1/etc/bashrc
mkdir -p $ParaView_DIR && cd $ParaView_DIR
cp -r /global/software/ParaView/ParaView-4.3.1-Linux-64bit/* .
```

Step 2: launch paraFoam inside a VNC session

```
<change to the case directory containing system/, constant/,
 processorXXX/, time outputs>
export FOAM_INST_DIR=/scratch2/razoumov
source $FOAM_INST_DIR/OpenFOAM-2.3.1/etc/bashrc
vglrun paraFoam -builtin    # launches ParaView, points it to
                            # a sequence of time step files,
                            # loads the first time step
```

Recap of input file formats

- Raw binary data
- VTK legacy format (*.vtk) with ASCII data for small datasets
  - Structured Points
  - Structured Grid
  - Polygonal Data
- VTK XML formats for large datasets: most versatile, use from C++ and Python
  - Structured Grid (*.vts)
  - other formats can be written using the respective class, e.g. vtkPolyData, vtkRectilinearGrid, vtkStructuredGrid, vtkUnstructuredGrid
- HDF5 files via XDMF, **native NetCDF**
- Many 3rd-party file formats understood natively by ParaView
- OpenFOAM is doable but need to use the right technique (don't trust the available documentation: a lot of it is wrong!)

Working with ParaView: filters

Filters

Many interesting features about a dataset cannot be determined by simply looking at its surface: a lot of useful information is on the inside, or can be extracted from a combination of variables

Sometimes a desired view is not available for a given data type, e.g.

- a 2D dataset $f(x, y)$ will be displayed as a 2D dataset even in 3D (try loading `data/2d000.vtk`), but we might want to see it in 3D by displaying the elevation $z = f(x, y)$
- volumetric view – not available for all VTK datasets (available, among others, for Structured Points and for UnstructuredGrid with connectivity provided)

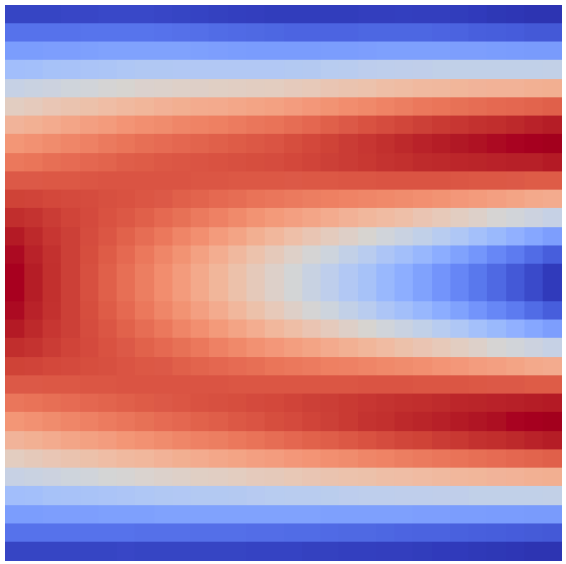
Filters are functional units that process the data to generate, extract, or derive additional features. The filter connections form a **visualization pipeline**

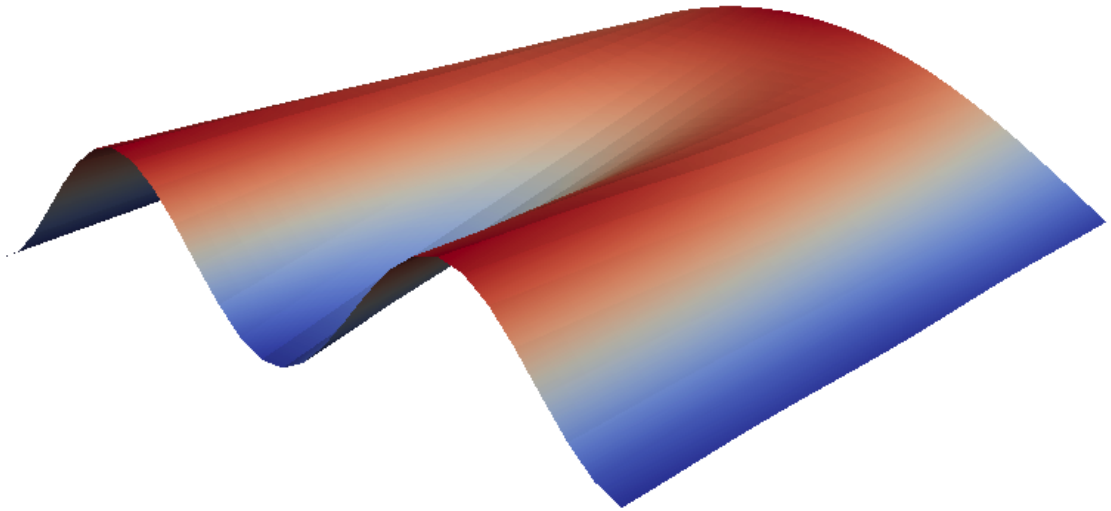
Last time I counted there were 146 filters. One can add new filters with python scripting

➠ Check out "Filters" in the menu; some are found in the toolbar
➠ List of filters `http://bit.ly/ZX5u2q` with documentation

Simple filter to visualize a 2D dataset in 3D

- Load the file `data/2d000.vtk` that samples the 2D function $f(x, y) = (1 - y)\sin(\pi x) + y\sin^2(2\pi x)$, where $x, y \in [0, 1]$, on a 30^2 grid
- Highlight the dataset in the pipeline browser and apply the WarpByScalar filter
- Change to 3D view, edit the offset factor to reproduce the 3D view below





Toolbar filters

- **Calculator** evaluates a user-defined expression on a per-point or per-cell basis.
- **Contour** extracts user-defined points, isocontours/isosurfaces from a scalar field.
- **Clip** removes all geometry on one side of a user-defined plane.
- **Slice** intersects the geometry with a plane. The effect is similar to clipping except that all that remains is the geometry where the plane is located.
- **Threshold** extracts cells that lie within a specified range of a scalar field.
- **Extract Subset** extracts a subset of a grid by defining either a volume of interest or a sampling rate.
- **Glyph** places a glyph on each point in a mesh. The glyphs may be oriented by a vector and scaled by a vector or scalar.
- **Stream Tracer** seeds a vector field with points and then traces those seed points through the steady state vector field.
- **Warp By Vector** displaces each point in a mesh by a given vector field.
- **Group Datasets** combines the output of several pipeline objects into a single multi-block dataset.
- **Extract Level** extracts one or more items from a multi-block dataset.

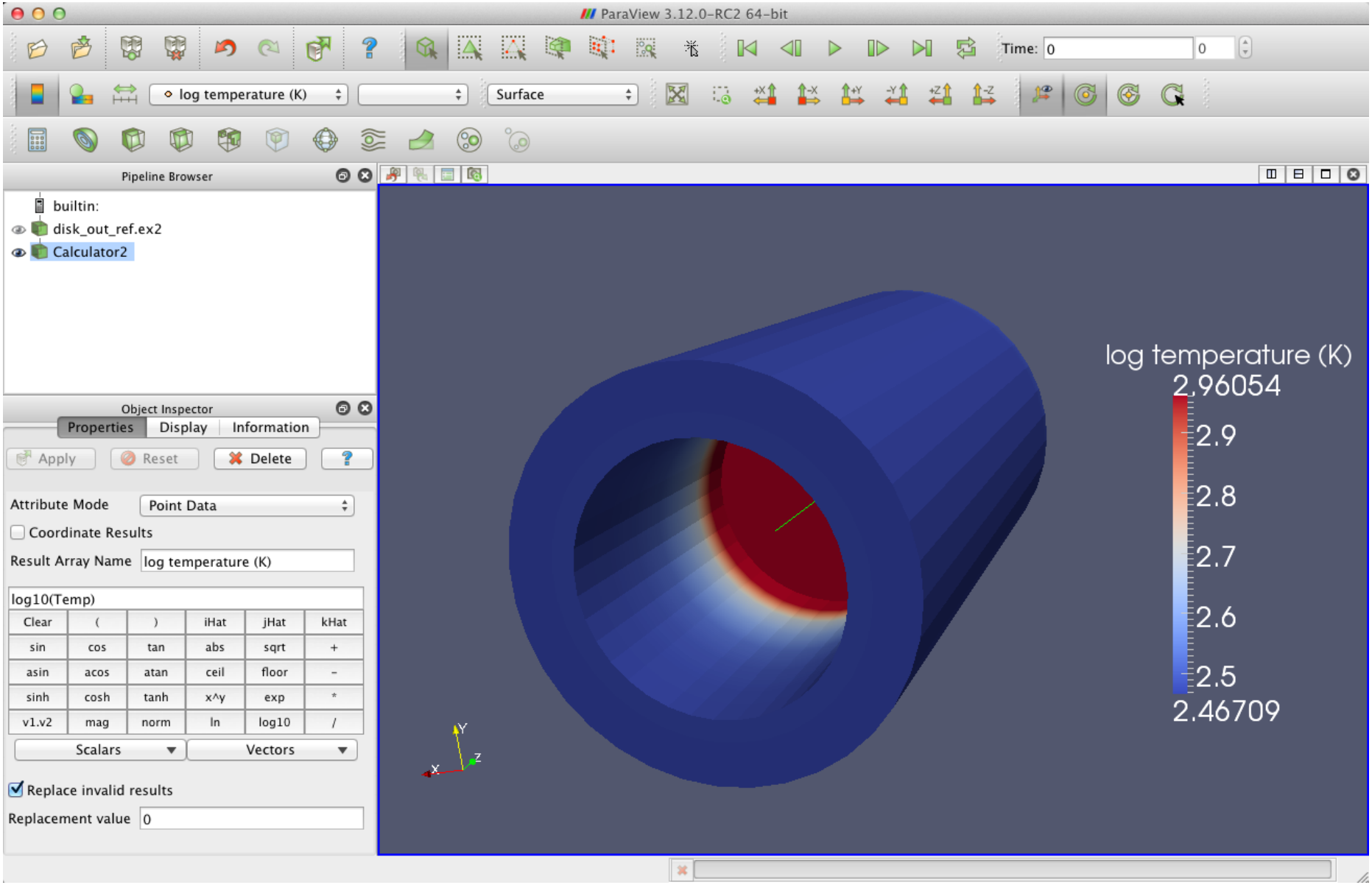
Calculator

- Load one of the datasets, e.g. `data/disk_out_ref.ex2` (load *temperature, velocity, pressure*), and try to visualize individual variables: Pres, Temp, V
- Click on "Toggle Colour Legend Visibility" to see the temperature range
- Now apply **Calculator** filter to display log10(Temp) – see the next slide
  - can also try to visualize Pres/Temp, mag(V)
  - dropdown menus "Scalars" and "Vectors" will help you enter variables
  - the "?" button is surprisingly useful
- You can change visibility of each object in the pipeline browser by clicking on the eyeball icon next to it

Calculator

- Load one of the datasets, e.g. `data/disk_out_ref.ex2` (load *temperature, velocity, pressure*), and try to visualize individual variables: Pres, Temp, V
- Click on "Toggle Colour Legend Visibility" to see the temperature range
- Now apply **Calculator** filter to display log10(Temp) – see the next slide
  - can also try to visualize Pres/Temp, mag(V)
  - dropdown menus "Scalars" and "Vectors" will help you enter variables
  - the "?" button is surprisingly useful
- You can change visibility of each object in the pipeline browser by clicking on the eyeball icon next to it

Calculator


ParaView 3.12.0-RC2 64-bit
Time: 0
log temperature (K)
Surface
Pipeline Browser
builtin:
disk_out_ref.ex2
Calculator2
Object Inspector
Properties
Display
Information
Apply
Reset
Delete
Attribute Mode
Point Data
Coordinate Results
Result Array Name
log temperature (K)
log10(Temp)
Clear ( ) iHat jHat kHat
sin cos tan abs sqrt +
asin acos atan ceil floor -
sinh cosh tanh x^y exp *
v1.v2 mag norm ln log10 /
Scalars
Vectors
Replace invalid results
Replacement value
0
log temperature (K)
2.96054
2.9
2.8
2.7
2.6
2.5
2.46709

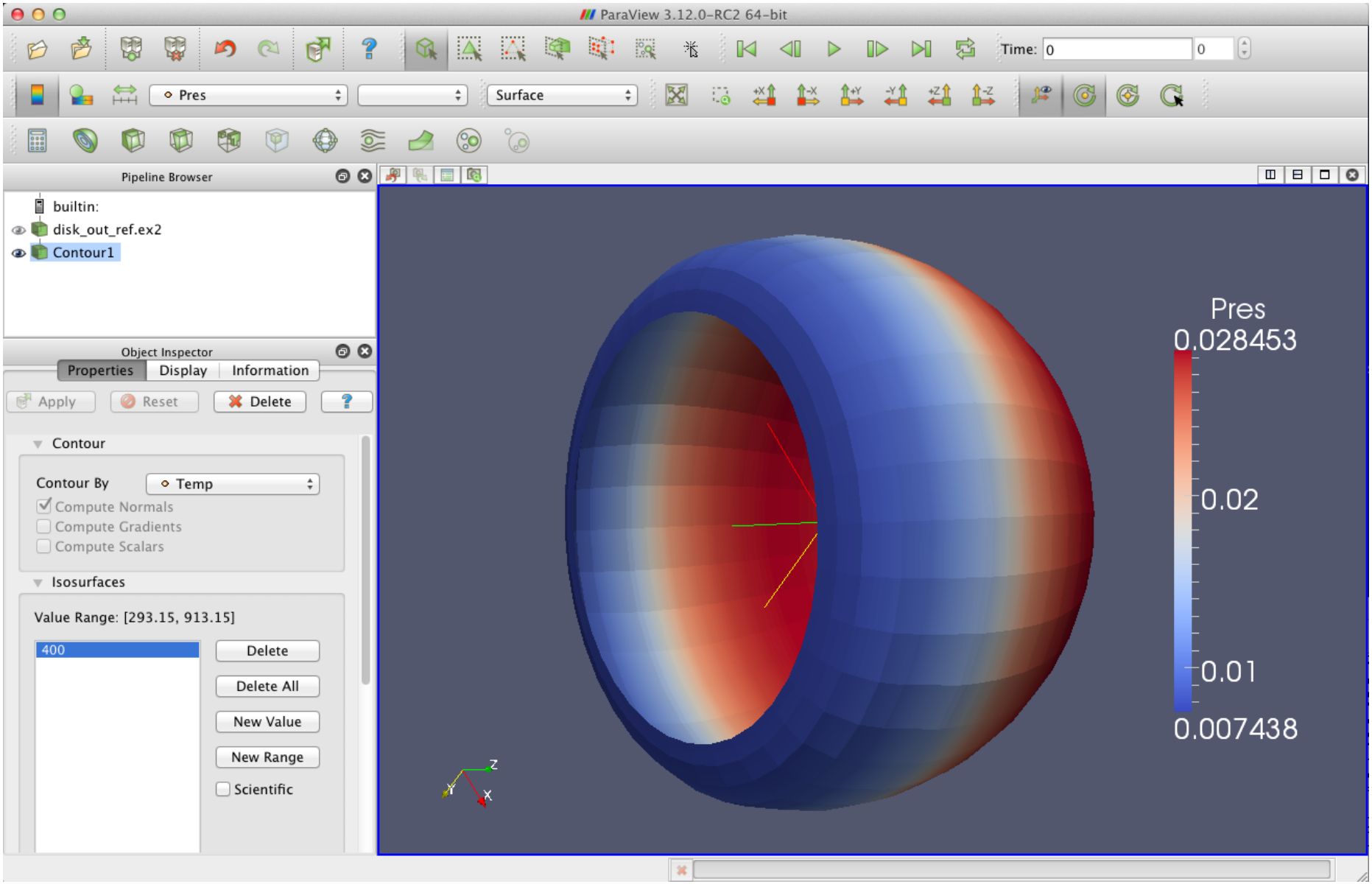
Contour

- Delete **Calculator** from the pipeline browser, load **Contour**
- Create an isosurface where the temperature is 400 K and colour it with pressure – see the next slide
- Now delete the isosurface at 400K and draw two isosurfaces (300K and 800K) and colour them with temperature (add the colour legend to distinguish between the two temperatures)
- Switch to the Wireframe view to see both surfaces clearly

Contour

- Delete **Calculator** from the pipeline browser, load **Contour**
- Create an isosurface where the temperature is 400 K and colour it with pressure – see the next slide
- Now delete the isosurface at 400K and draw two isosurfaces (300K and 800K) and colour them with temperature (add the colour legend to distinguish between the two temperatures)
- Switch to the Wireframe view to see both surfaces clearly

Contour


ParaView 3.12.0-RC2 64-bit
Time: 0
Pres
Surface
Pipeline Browser
builtin:
disk_out_ref.ex2
Contour1
Object Inspector
Properties
Display
Information
Apply
Reset
Delete
Contour
Contour By
Temp
Compute Normals
Compute Gradients
Compute Scalars
Isosurfaces
Value Range: [293.15, 913.15]
400
Delete
Delete All
New Value
New Range
Scientific
Pres
0.028453
0.02
0.01
0.007438

Creating a visualization pipeline

You can apply one filter to the data generated by another filter

Delete all previous filters, start with the original data from `data/disk_out_ref.ex2`, or just press "Disconnect" and reload the data

1. Apply **Clip** filter to the data: rotate, move the clipping plane, select variables to display, make sure there are data points inside the object (easy to see with points/wireframe, uncheck "Show Plane")

2. Delete **Clip**, now apply Filters → Alphabetical → **Extract Surface**, and then add **Clip** to the result of **Extract Surface** ⇒ the dataset is now hollow (use wireframe/surface)

Creating a visualization pipeline

You can apply one filter to the data generated by another filter

Delete all previous filters, start with the original data from `data/disk_out_ref.ex2`, or just press "Disconnect" and reload the data

1. Apply **Clip** filter to the data: rotate, move the clipping plane, select variables to display, make sure there are data points inside the object (easy to see with points/wireframe, uncheck "Show Plane")

2. Delete **Clip**, now apply Filters → Alphabetical → **Extract Surface**, and then add **Clip** to the result of **Extract Surface** ⇒ the dataset is now hollow (use wireframe/surface)

Multiview: several variables side by side

- Start with original data (`data/disk_out_ref.ex2`), load all variables
- Add the **Clip** filter, uncheck "Show Plane" in the object inspector, click "Apply"
- Colour the surface by pressure by changing the variable chooser in the toolbar from "Solid Colour" to "Pres"
- Press "Split horizontal", make sure the view in the right is active (has a blue border around it)
- Turn on the visibility of the clipped data by clicking the eyeball next to Clip in the pipeline browser
- Colour the surface by temperature by changing the toolbar variable chooser from "Solid Colour" to "Temp" – see the next slide
- To link the two views, right click on one of the views and select "Link Camera...", click in a second view, and try moving the object in each view
- Can add colourbars to either view by clicking "Toggle Colour Legend Visibility", try moving colourbars around
- To unlink, go to Tools -> Manage Links, delete the camera link in question

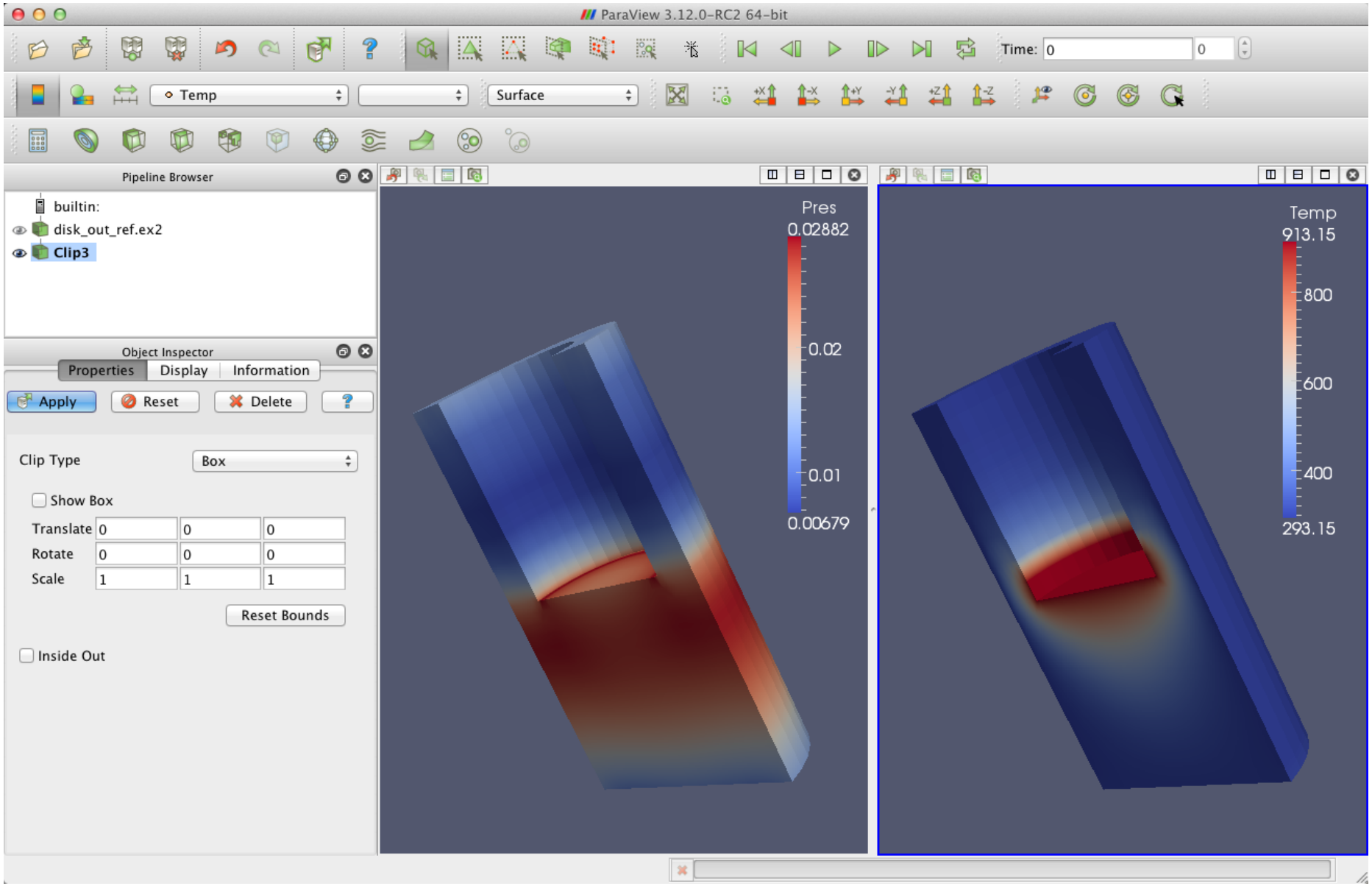
Multiview: several variables side by side

- Start with original data (`data/disk_out_ref.ex2`), load all variables
- Add the **Clip** filter, uncheck "Show Plane" in the object inspector, click "Apply"
- Colour the surface by pressure by changing the variable chooser in the toolbar from "Solid Colour" to "Pres"
- Press "Split horizontal", make sure the view in the right is active (has a blue border around it)
- Turn on the visibility of the clipped data by clicking the eyeball next to Clip in the pipeline browser
- Colour the surface by temperature by changing the toolbar variable chooser from "Solid Colour" to "Temp" – see the next slide
- To link the two views, right click on one of the views and select "Link Camera...", click in a second view, and try moving the object in each view
- Can add colourbars to either view by clicking "Toggle Colour Legend Visibility", try moving colourbars around
- To unlink, go to Tools -> Manage Links, delete the camera link in question

Multiview: several variables side by side

- Start with original data (`data/disk_out_ref.ex2`), load all variables
- Add the **Clip** filter, uncheck "Show Plane" in the object inspector, click "Apply"
- Colour the surface by pressure by changing the variable chooser in the toolbar from "Solid Colour" to "Pres"
- Press "Split horizontal", make sure the view in the right is active (has a blue border around it)
- Turn on the visibility of the clipped data by clicking the eyeball next to Clip in the pipeline browser
- Colour the surface by temperature by changing the toolbar variable chooser from "Solid Colour" to "Temp" – see the next slide
- To link the two views, right click on one of the views and select "Link Camera...", click in a second view, and try moving the object in each view
- Can add colourbars to either view by clicking "Toggle Colour Legend Visibility", try moving colourbars around
- To unlink, go to Tools -> Manage Links, delete the camera link in question

Multiview: several variables side by side



Vector visualization: streamlines and glyphs

1. Start with the original data from `data/disk_out_ref.ex2`, load velocity, Temp
2. Add the **Stream Tracer** filter, set Radius = 10 (of sphere with tracer points), play with Number Of Points, Maximum Streamline Length
3. Add shading and depth cues to streamlines: Filters → Alphabetical → **Tube** (could be also called Generate Tubes)
4. Add glyphs to streamlines to show the orientation and magnitude:
   - select StreamTracer in the pipeline browser
   - add the **Glyph** filter to StreamTracer
   - in the object inspector, change the Vectors option (second from the top) to "V"
   - in the object inspector, change the Glyph Type option (third from the top) to "Cone"
   - hit "Apply"
   - colour the glyphs with the "Temp" variable – see the next slide
5. Now try displaying "V" glyphs directly from data, can colour them using different variables ("Temp", "V")

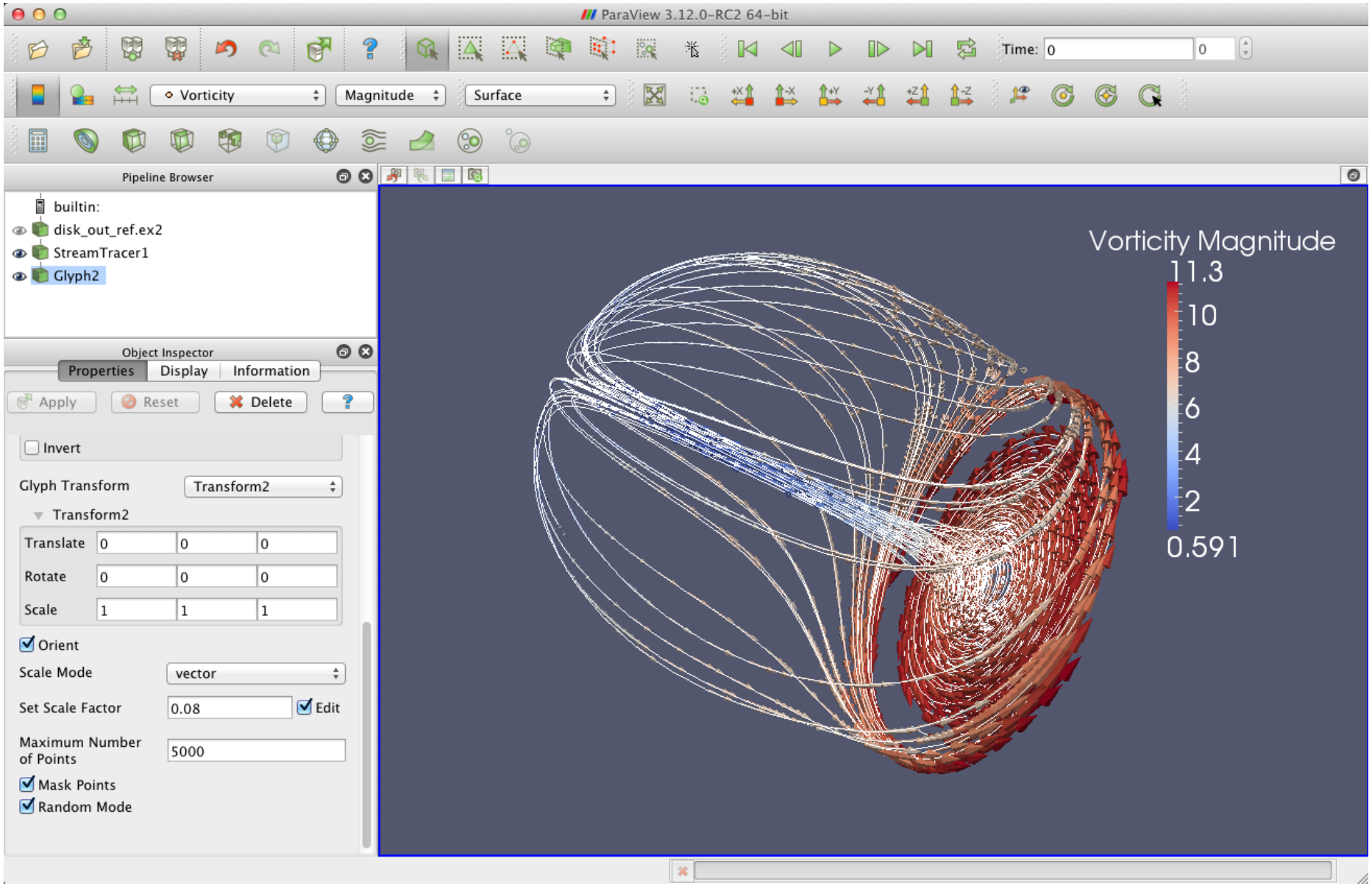
Vector visualization: streamlines and glyphs

1. Start with the original data from `data/disk_out_ref.ex2`, load velocity, Temp
2. Add the **Stream Tracer** filter, set Radius = 10 (of sphere with tracer points), play with Number Of Points, Maximum Streamline Length
3. Add shading and depth cues to streamlines: Filters → Alphabetical → **Tube** (could be also called Generate Tubes)
4. Add glyphs to streamlines to show the orientation and magnitude:
   - select StreamTracer in the pipeline browser
   - add the **Glyph** filter to StreamTracer
   - in the object inspector, change the Vectors option (second from the top) to "V"
   - in the object inspector, change the Glyph Type option (third from the top) to "Cone"
   - hit "Apply"
   - colour the glyphs with the "Temp" variable – see the next slide
5. Now try displaying "V" glyphs directly from data, can colour them using different variables ("Temp", "V")

Vector visualization: streamlines and glyphs

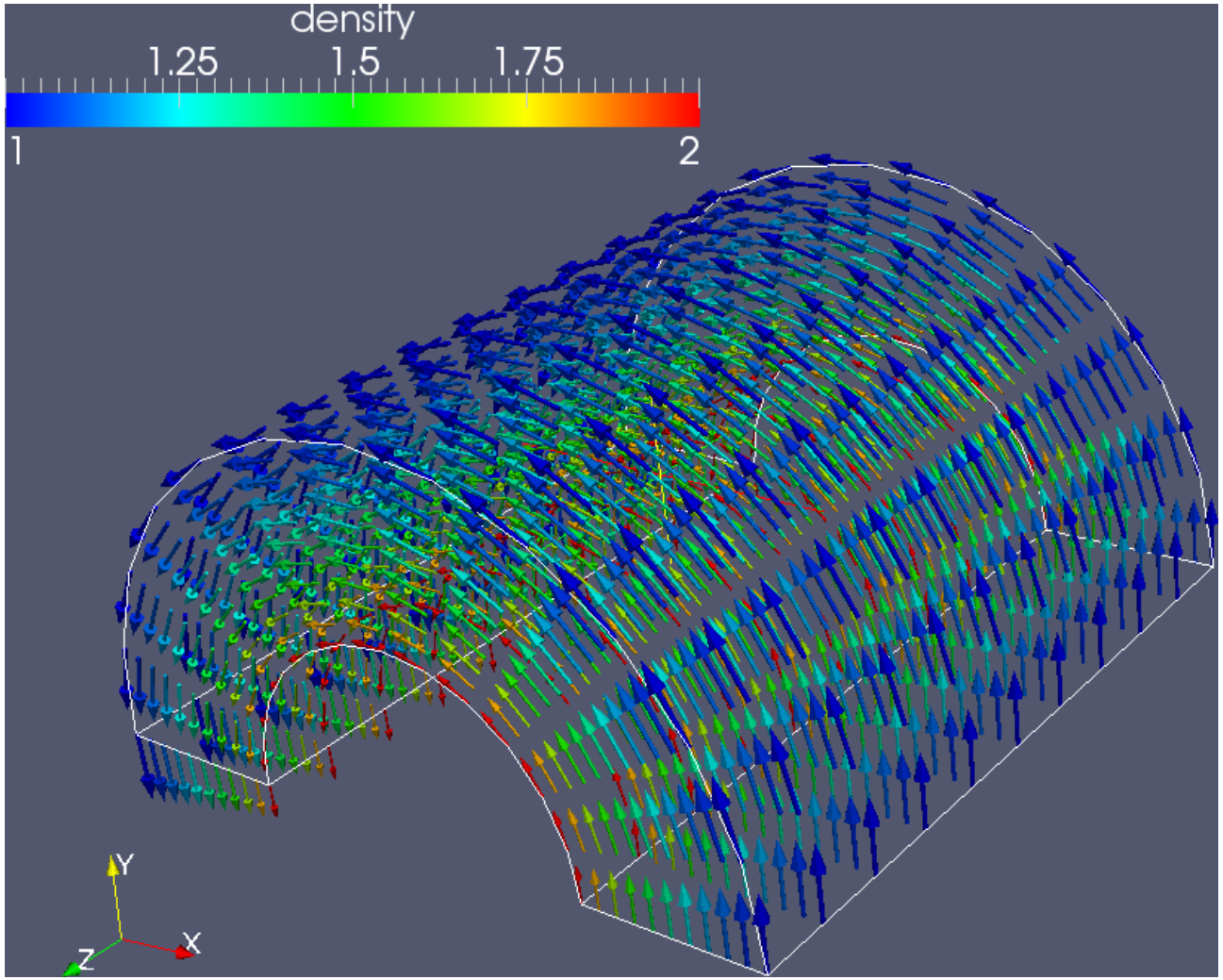
1. Start with the original data from `data/disk_out_ref.ex2`, load velocity, Temp
2. Add the **Stream Tracer** filter, set Radius = 10 (of sphere with tracer points), play with Number Of Points, Maximum Streamline Length
3. Add shading and depth cues to streamlines: Filters → Alphabetical → **Tube** (could be also called Generate Tubes)
4. Add glyphs to streamlines to show the orientation and magnitude:
   - select StreamTracer in the pipeline browser
   - add the **Glyph** filter to StreamTracer
   - in the object inspector, change the Vectors option (second from the top) to "V"
   - in the object inspector, change the Glyph Type option (third from the top) to "Cone"
   - hit "Apply"
   - colour the glyphs with the "Temp" variable – see the next slide
5. Now try displaying "V" glyphs directly from data, can colour them using different variables ("Temp", "V")

Vector visualization: streamlines and glyphs



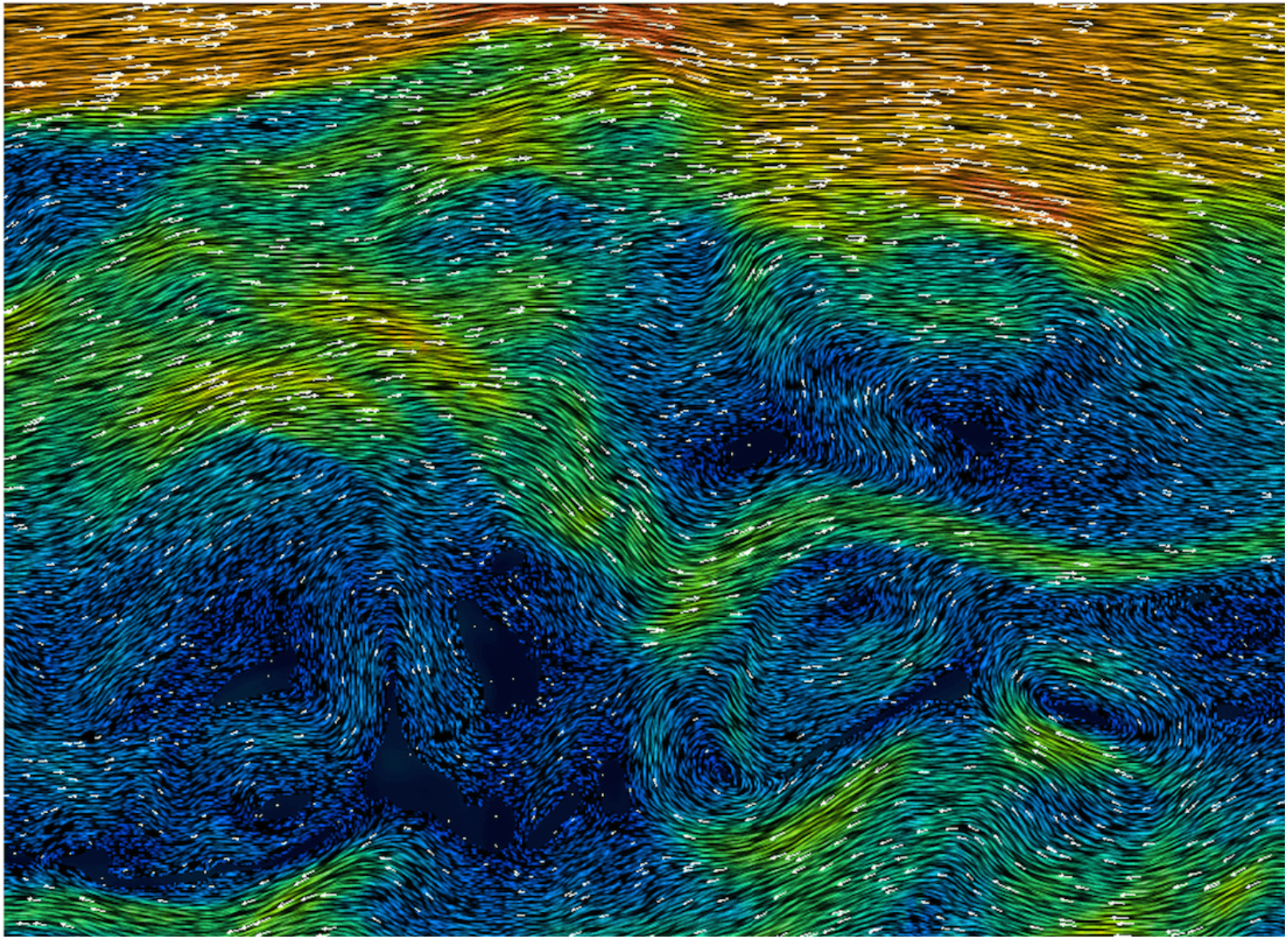
Exercise: vectors

Load `data/halfCylinder.vts` and display the velocity field as arrows, colouring them by density – try to reproduce the view below

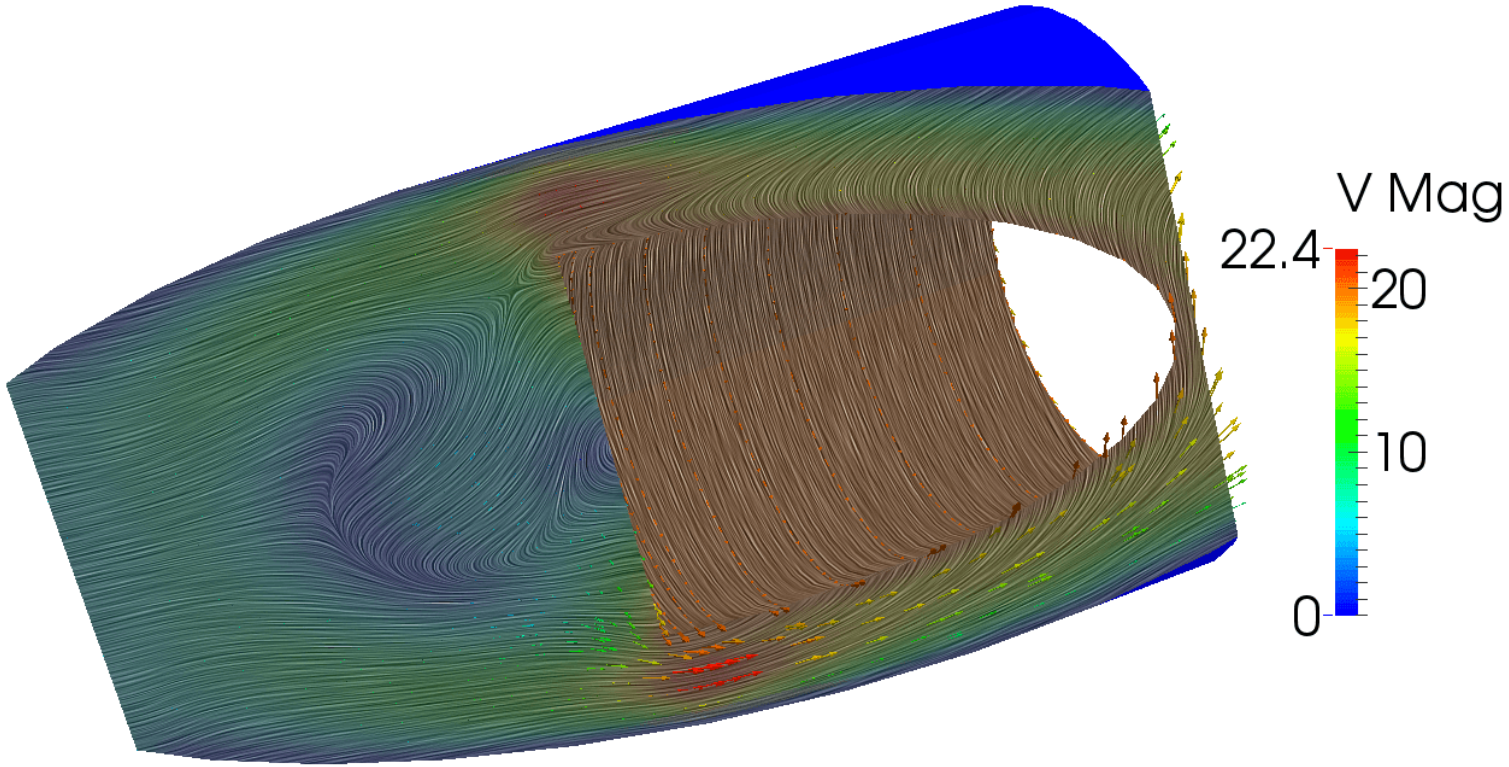


Line Integral Convolution representation

Details at `http://www.paraview.org/Wiki/ParaView/Line_Integral_Convolution`



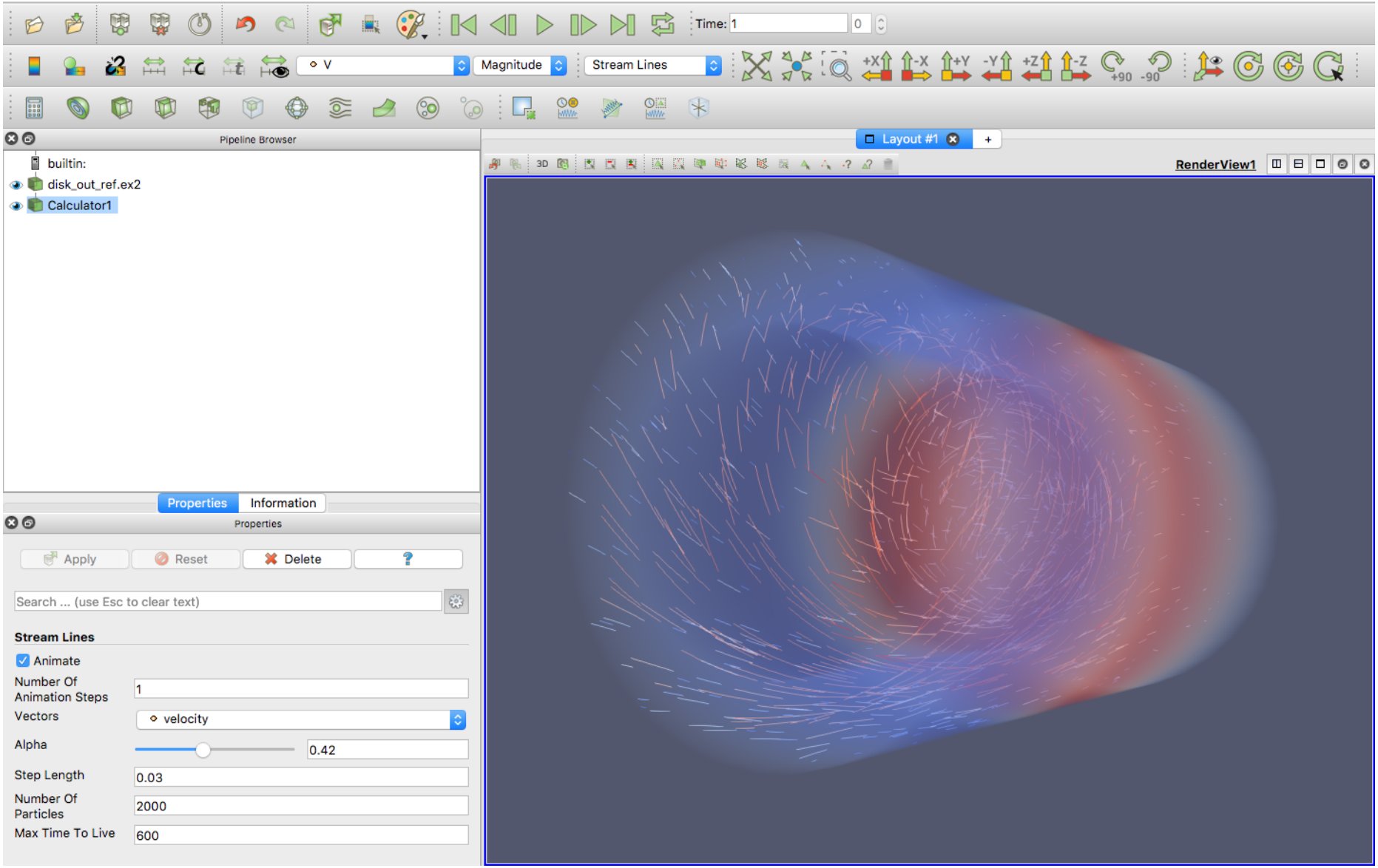
Line Integral Convolution in ParaView



- From Tools → Manage Plugins load *Surface LIC plugin*
- Load `data/disk_out_ref.ex2` or `data/halfCylinder.vts`
- Apply a filter to see its interior (required step for `data/halfCylinder.vts`)
- Switch to *Surface LIC* representation in the drop-down menu
- In properties make sure to select the velocity variable for Surface LIC
- Play with the number of steps and individual step sizes, adjust colour

Stream Lines representation (live drawing)

Details at `http://bit.ly/2NFNcvQ`



Quick and dirty input format: 3D data as columns

- `data/tabulatedPoints.txt` contains 100 random points, with each line storing *x*, *y*, *z*, *scalar* of a point
- Import it into ParaView, apply the **Table To Points** filter, making sure to edit the fields (X/Y/Z Columns)
- Apply the **Glyph** filter to view points as spheres, colour them by *scalar*
- No implied topology here!
- You can optionally pass the points through the **Delaunay 3D** filter, followed by **Extract Edges**, followed by **Tube**

- `data/tabulatedGrid.txt` contains 1000 points representing a 10^3 Cartesian mesh, with each line storing *x*, *y*, *z*, *scalar* of a point
- Import it into ParaView, apply the **Table To Structured Grid** filter, making sure to edit the fields (Whole Extent 0 to 9 in each dimension, X/Y/Z Columns)
- The data must have some implied topology for this filter to work!

Not recommended for large datasets: waste of disk storage and bandwidth!

➢ tabulatedPoints.txt is 6231 bytes vs. 1600 bytes in single-precision binary

➢ tabulatedGrid.txt is 20,013 bytes vs. 4000 bytes in single-precision binary

Word of caution

- Many visualization filters transform stuctured grid data into unstructured data (e.g. Clip, Slice)
- Memory footprint and CPU load can grow very quickly, e.g. clipping 400^3 to 150 million cells can take $\sim$ 1 hour on a single CPU $\Rightarrow$ might want to run in distributed mode

Python Calculator filter

https://www.paraview.org/Wiki/Python_Calculator

- To calculate vorticity, pick a fector field, enter `curl(V)`, call it vorticity
- Supported functions on arrays: abs(), cross(), curl(), det(), dot(), eigenvalue(), eigenvector(), global_mean(), global_max(), global_min(), gradient(), inverse(), laplacian(), ln(), log10(), max(), min(), mean(), mag(), norm(), strain(), trace(), vorticity()

More filter functionality

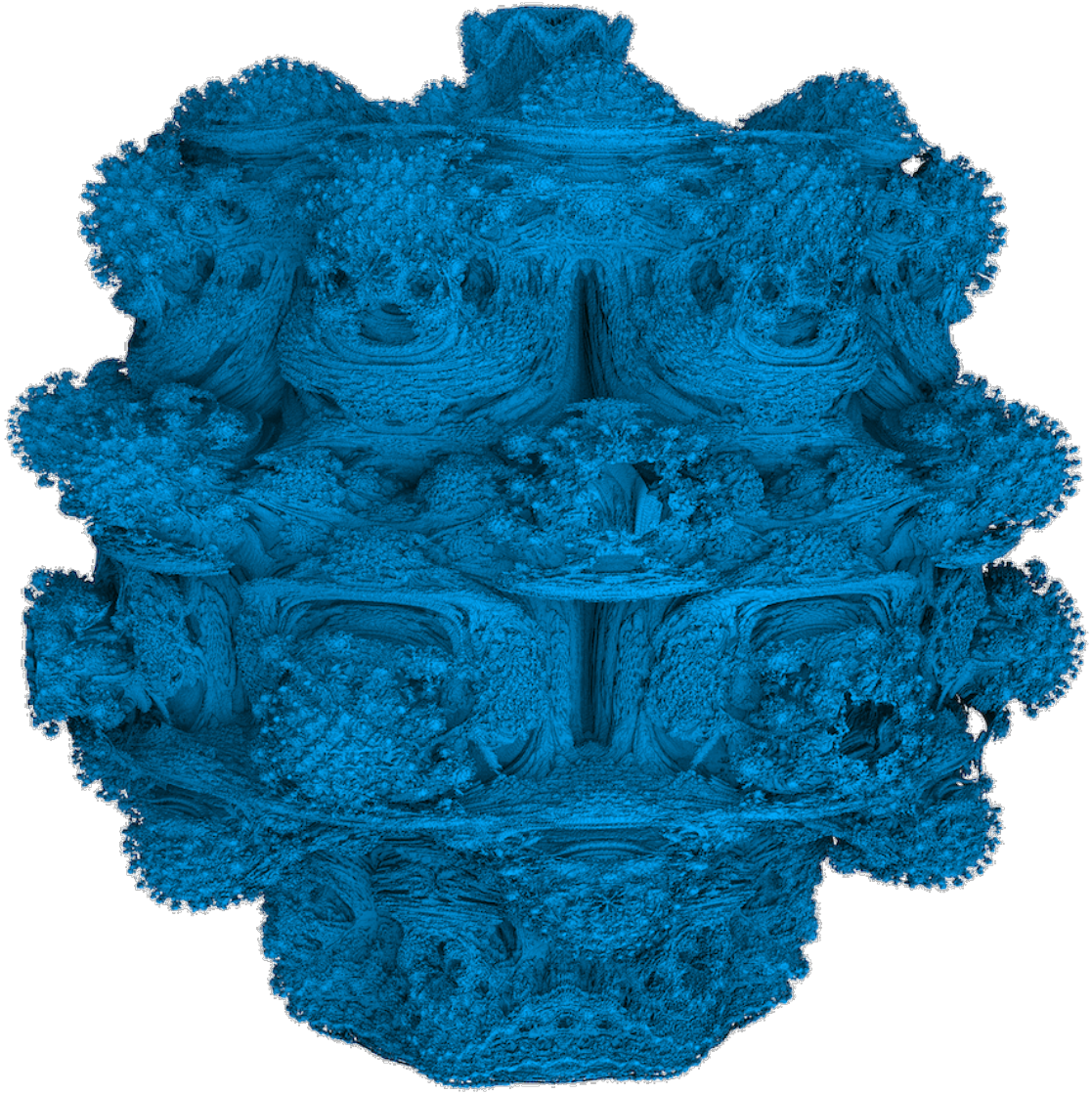
- Can merge several existing filters into a *custom filter*
  `http://www.paraview.org/Wiki/ParaView/Custom_Filters`
  Tools → **Create Custom Filter** and edit its input, output and properties

- Can script filters in Python
  `http://www.paraview.org/Wiki/Python_Programmable_Filter`
  Filters → **Alphabetical** → **Programmable Filter**
  (more on general scripting later today)

- Can write new filters as plugins, compile them as shared libraries with the same version of ParaView they are expected to be deployed with
  `http://www.paraview.org/Wiki/ParaView/Plugin_HowTo`

Exercise: try out something more complex

- Power-8 Mandelbulb is a 3D fractal
- Load the file `mandelbulb300.nc` – sampled here at 300^3
- Let's do this:
  1. check the grid size and the file size: do these agree?
  2. try slices, volumetric rendering, isosurfaces
  3. try to recreate the picture on the right (at a lower resolution): pay attention to the lights and shadows



Exercise: 3D optimization

`data/stvol.nc` contains a discretized scaled variant of the 3D Styblinski-Tang function inside a unit cube ($x_i \in [0, 1]$), built with `codes/optimization.c`

$$f(x_1, x_2, x_3) = \frac{1}{2} \sum_{i=1}^{3} (\xi_i^4 - 16\xi_i^2 + 5\xi_i), \text{ where } \xi_i \equiv 8(x_i - 0.5)$$

Let's answer the following questions:

1. What is the size of the grid? Does it agree with the size of the file?
2. Find the approximate location of the *global minimum* of $f(x_1, x_2, x_3)$ using visual techniques (slices, isosurfaces, thresholds, volume renderings, etc.)

- Note: you can find the exact coordinates of the global minimum by using Filters -> Statistics -> **Descriptive Statistics**, clicking Apply, and sorting points in order of increasing f(x,y,z)

Scientific visualization with ParaView
Part 2

Alex Razoumov
alex.razoumov@westdri.ca

Digital Research Alliance of Canada

SIMON FRASER UNIVERSITY

- ✔ slides, data, codes at `https://bit.ly/paraviewzipp`
  - the link will download a file `paraview.zip` (~36MB)
  - unpack it to find `codes/`, `data/` and `slides{1,2}.pdf`
  - command line: `wget https://bit.ly/paraviewzipp -O paraview.zip`
- ✔ install ParaView 5.x on your laptop from `http://www.paraview.org/download`

Exporting scenes (pre-computed polygons)

ParaView Glance

`https://kitware.github.io/paraview-glance`

PV Glance is an open-source standalone web app for in-browser 3D sci-vis

- very easy to use, ideal for sharing pre-built 3D scenes via the web
- no server ⇒ up to medium-size data (server support planned in future versions)
- interactive manipulation of pre-computed polygons
  - volumetric images, molecular structures, geometric objects, point clouds
- written in JavaScript and vtk.js + can be further customized with vtk.js and ParaViewWeb for custom web and desktop apps
- source and installation instructions
  `https://github.com/kitware/paraview-glance`

---

1. Create a visualization with several layers, make **all layers visible in the pipeline**
2. Many options in File → Export Scene... ⇒ save as VTKJS to your laptop
3. Open `https://kitware.github.io/paraview-glance/app`
4. Drag the newly saved file to the dropzone on the website
5. Interact with individual layers in 3D: rotate and zoom, change visibility, representation, variable, colourmap, opacity

Automatically load a visualisation into Glance

https://discourse.paraview.org/t/customise-pv-glance/2831

- Use the query syntax
  GLANCEAPPURL?name=FILENAME&url=FILEURL
  to pass `name` and `url` to the web server
- E.g. using ParaView Glance website
  https://kitware.github.io/paraview-glance/app?name=sineEnvelope.vtkjs&url=https://raw.githubusercontent.com/razoumov/publish/master/data/sineEnvelope.vtkjs
  - shortened to https://bit.ly/2KtPWNf
- You can parse long strings with JavaScript (next slide)

Embed your vis into a website with an iframe (`embed.html`)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sine envelope function</title>
  </head>
  <body>
    <h1>3D sine envelope function</h1>

    <script>
      var app = "https://kitware.github.io/paraview-glance/app";
      var dir = "https://raw.githubusercontent.com/razoumov/publish/master/data/";
      var file = "sineEnvelope.vtkjs";
      document.write("<iframe src='" + app + "?name=" + file + "&url=" +
                     dir + file +
                     "' id='iframe' width='1100' height='900'></iframe>");
    </script>

    <p>More stuff in here</p>
  </body>
</html>
```

- JavaScript here only to parse long strings

Animation in ParaView

Animation methods

1. Use ParaView's built-in animation of any property of any pipeline object
   - easily create snazzy animations, somewhat limited in what you can do
   - in Animation View: select object, select property, create a new track with "+", double-click the track to edit it, press "▶"

Animation methods

1. Use ParaView's built-in animation of any property of any pipeline object
   - easily create snazzy animations, somewhat limited in what you can do
   - in Animation View: select object, select property, create a new track with "+", double-click the track to edit it, press "▶"
2. Use ParaView's ability to recognize a sequence of similar files
   - time animation only, very convenient
   - try loading `data/2d*.vtk` sequence and animating it (visualize one frame and then press "▶")

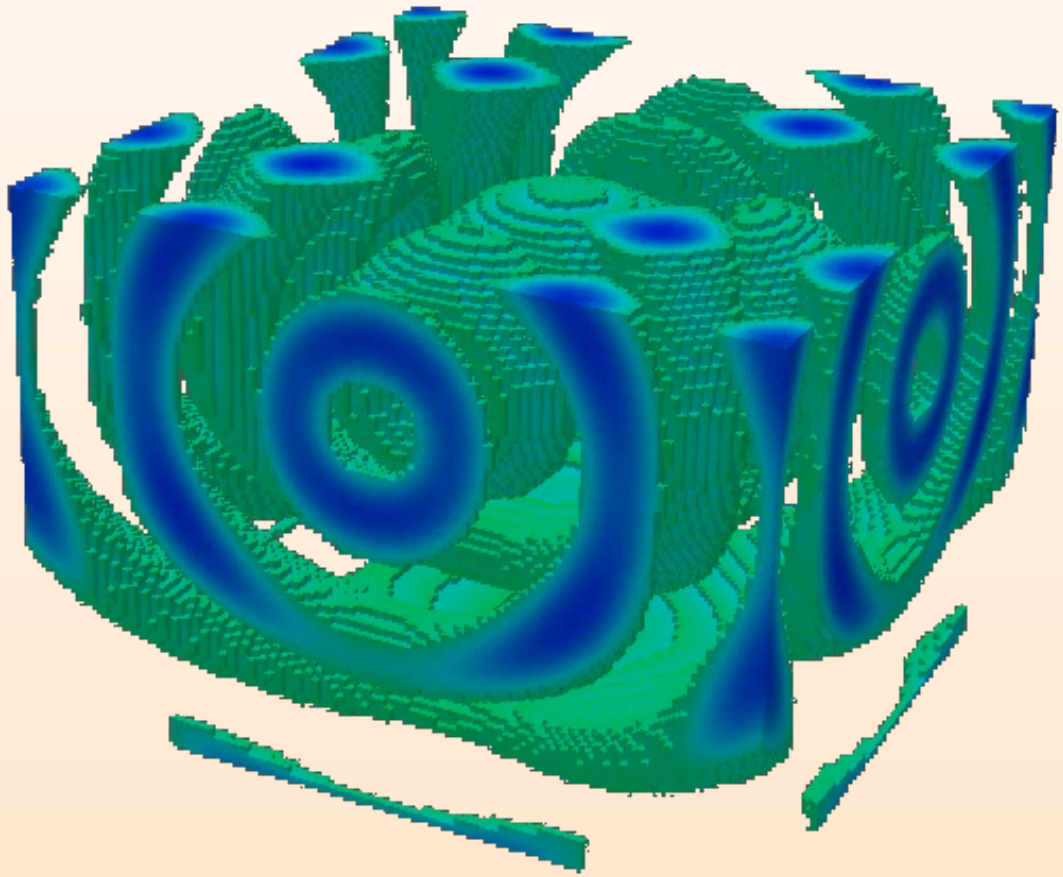
Animation methods

1. Use ParaView's built-in animation of any property of any pipeline object
   - easily create snazzy animations, somewhat limited in what you can do
   - in Animation View: select object, select property, create a new track with "+", double-click the track to edit it, press "▶"
2. Use ParaView's ability to recognize a sequence of similar files
   - time animation only, very convenient
   - try loading `data/2d*.vtk` sequence and animating it (visualize one frame and then press "▶")
3. Script your animation in Python (covered in next section)
   - steep learning curve, very powerful, can do anything you can do in the GUI
   - typical usage scenario: generate one frame per input file
   - a simpler exercise without input files: see next slide

Exercise: animating function growth

- 3D sine envelope wave function defined inside a unit cube ($x_i \in [0, 1]$)

$$f(x_1, x_2, x_3) = \sum_{i=1}^{2} \left[\frac{\sin^2\left(\sqrt{\xi_{i+1}^2 + \xi_i^2}\right) - 0.5}{\left[0.001(\xi_{i+1}^2 + \xi_i^2) + 1\right]^2} + 0.5 \right], \text{ where } \xi_i \equiv 15(x_i - 0.5)$$



- Reproduce the movie on the screen

  `https://vimeo.com/248501176`

  or `hidden/growth.mp4` on presenter's laptop

Exercise: animating function growth (cont.)

To visualize a single frame of the movie:

1. load `data/sineEnvelope.nc` (discretized on a 100^3 grid)
2. apply Threshold keeping only data from 1.2 to 2
3. apply Clip: origin $O = (49.5, 15, 49.5)$, normal $N = (0, -1, 0)$
4. colour by the right quantity

Two possible solutions:

1. bring up **Animation View** to animate Clip's O_2 from 0 to 99, for best results save animation as a sequence of PNG files
2. covered in the next section: Start/Stop Trace to record the workflow, save the corresponding **Python script**, enclose parts of it into a loop changing O_2 from 0 to 99 and writing a series of PNG screenshots, run it inside ParaView to produce 100 frames

   in either case, merge PNGs using a 3rd-party tool, e.g.

```
ffmpeg -r 30 -i frame%04d.png -c:v libx264 -pix_fmt yuv420p \
       -vf "scale=trunc(iw/2)*2:trunc(ih/2)*2" movie.mp4
```

Camera animation in the GUI

Good introductory resource `https://www.paraview.org/Wiki/Advanced_Animations`

1. Start with any static visualization
2. Click on 'Adjust Camera' icon (one of the left-side icons on top of the visualization window)
   - adjust / write down Camera Focal Point
3. Bring up Animation View (or erase all previous timelines)

(3a) In Animation View:

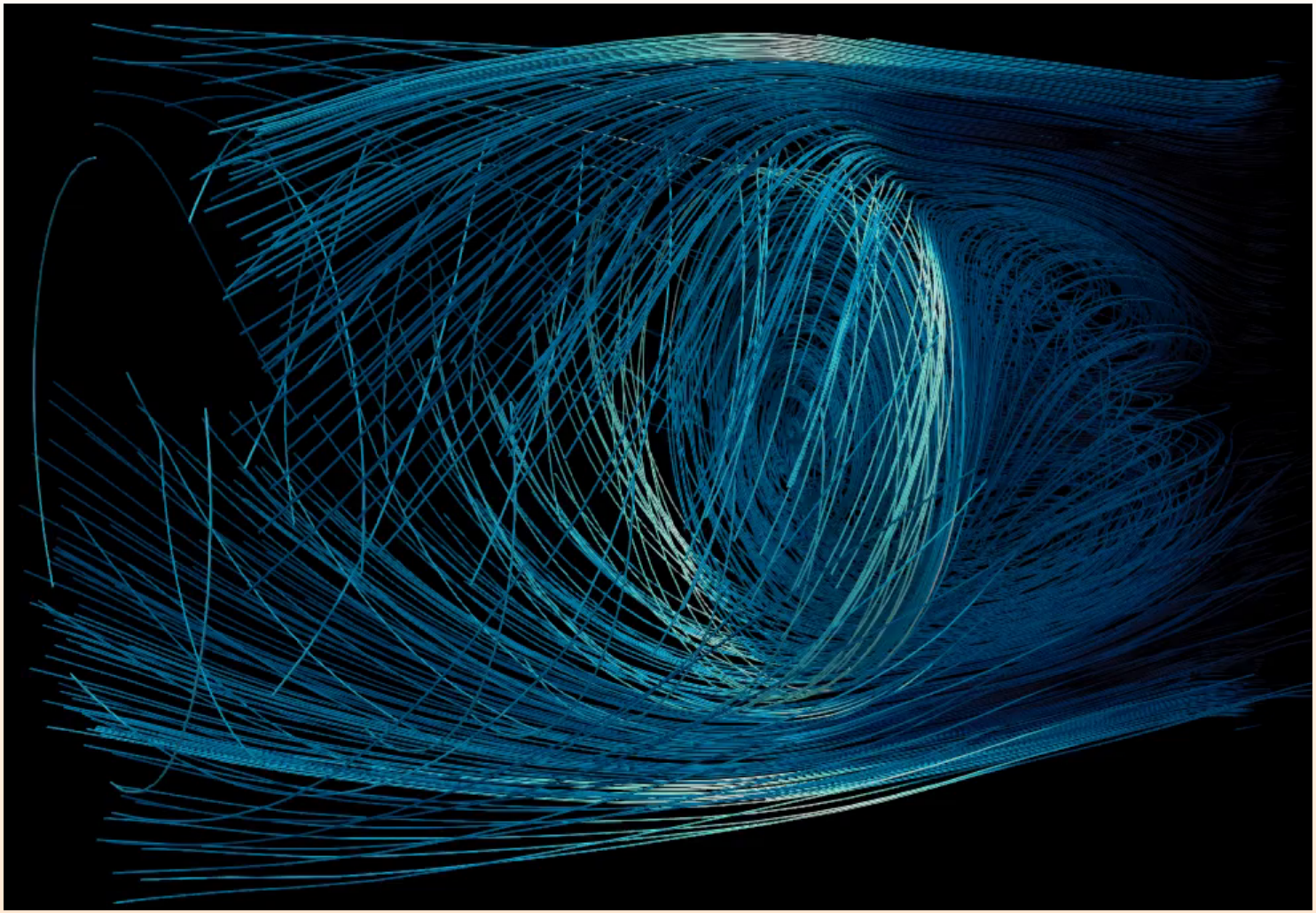
- select Camera - Orbit
- click "+" to create a new timeline
- set Center = Camera Focal Point, for the rest accept default settings
- adjust the number of frames

(3b) In Animation View:

- select Camera - Follow Path
- click "+" to create a new timeline
- double-click on the white (or black) timeline
- double-click on Path... in the right column
- click on Camera Position
  - a yellow path with spheres will appear
  - drag the spheres around
- also can change Camera Focus and Up Direction

4. Click "▶"

Animating stationary flow: streamlines through a slice

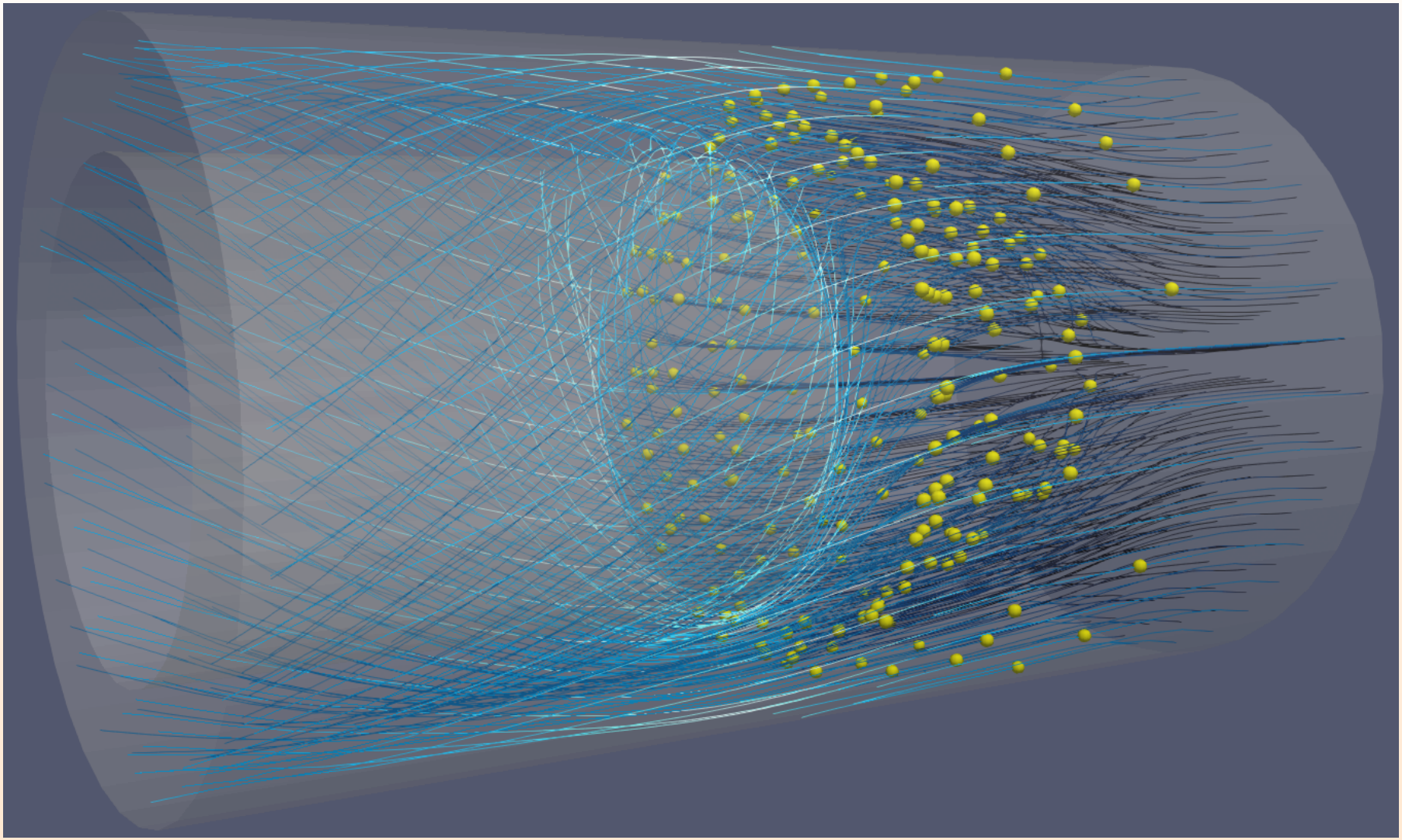


- `https://vimeo.com/248501893` or `hidden/radialSlice.mp4` on presenter's laptop
- `https://vimeo.com/248502086` or `hidden/xySlice.mp4` on presenter's laptop

Animating stationary flow: streamlines through a slice (cont.)

1. Load `disk_out_ref.ex2` making sure to load velocity
2. Draw a radius-z plane slice through the center, origin $O = (0, 0, 0)$ and normal $N = (1, 0, 0)$
3. Stream Tracer With Custom Source: *input*=`disk_out_ref.ex2`, *seedSource*=`Slice1`
4. Tube filter with $r = 0.015$
5. Animation View: animate Slice's O_0 from -1 to 1 (full range [-5.75,5.75])
6. Use 100 frames, black background, blue2cyan colourmap, colour with vorticity
7. Unselect "Show Plane"
8. Save animation as PNGs, encode at 10 fps

Animating a stationary flow: time contours



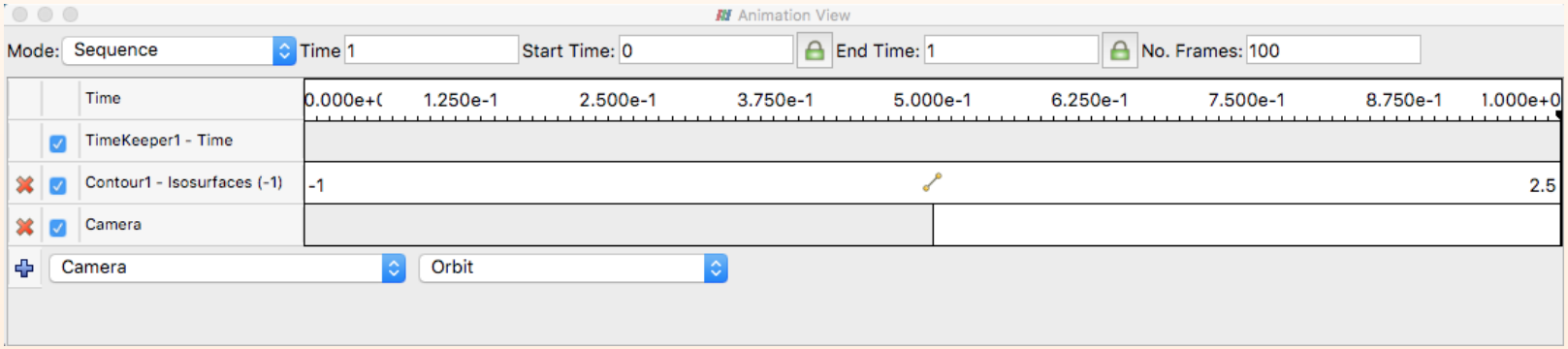
`https://vimeo.com/248509153` or `hidden/timeContours.mp4` on presenter's laptop

Animating a stationary flow: time contours (cont.)

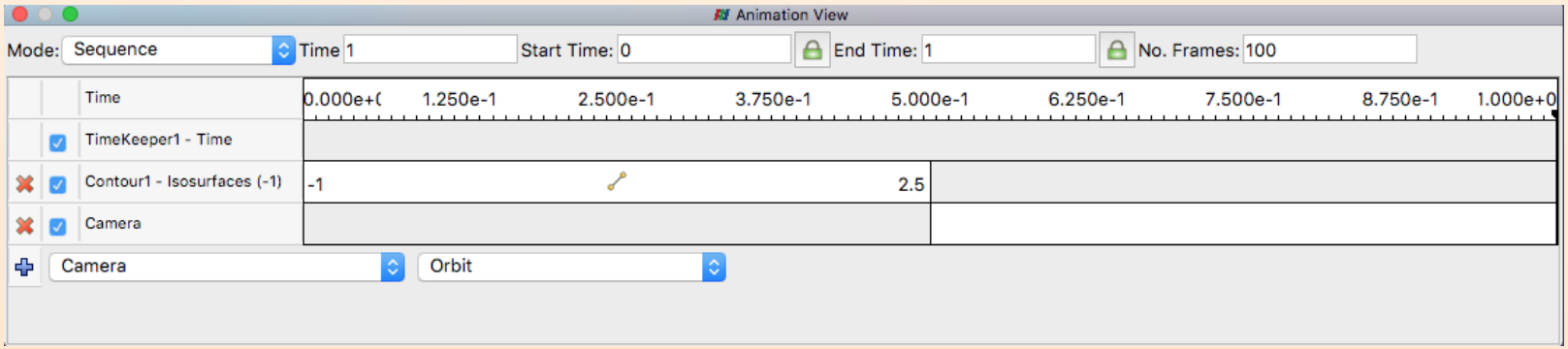
1. Start with the streamtracer lines, however drawn
2. Apply a Countour filter to the output of Streamtracer
   - contour by Integration Time
   - probe the range of values that works best
3. Apply Glyph filter to the output of Countour
4. Animation View: animate Contour | Isosurfaces
5. This video was recorded with 2000 frames at 60 fps
   - such high resolution only for the final production video
   - debugging animation with 100 frames is perfectly Ok

Exercise: several timelines in one animation

1. Start with the previous integration-time-contour animation
2. Add the second timeline to the animation: Camera - Orbit from $t = 0.5$ to $t = 1$ (while the first animation is still playing for its second half)

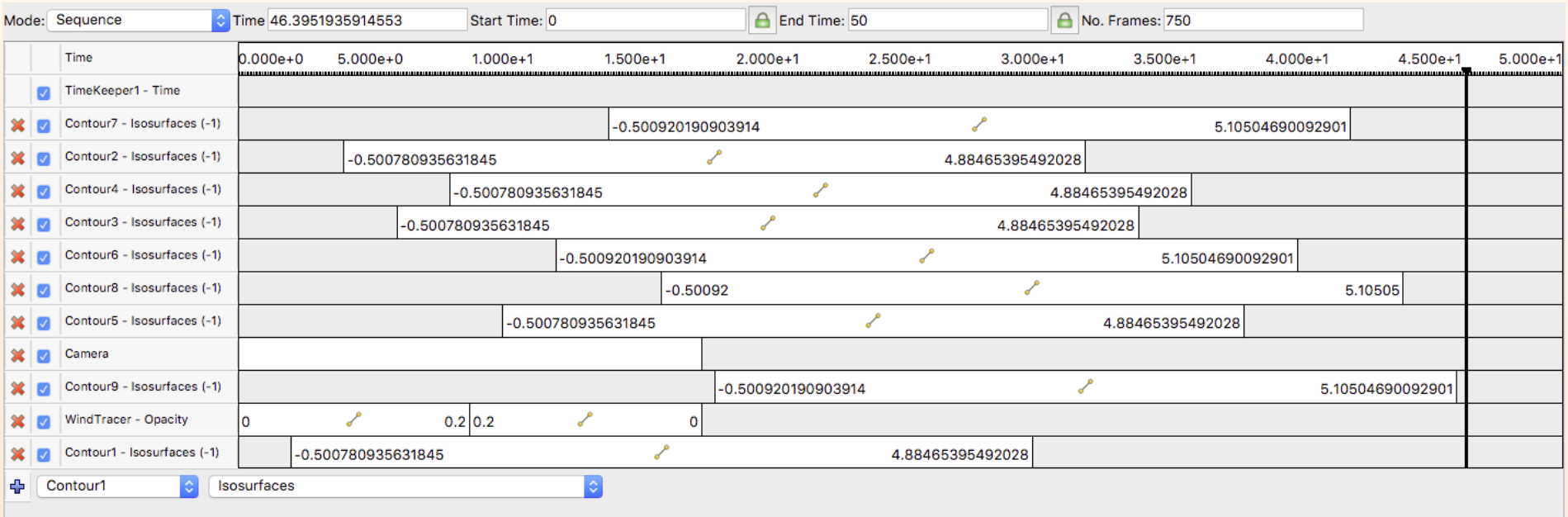


3. Now complete integration-time-contour animation before rotation



Combining many timelines in one animation (cont.)

- In principle, can add as many timelines (with their individual time intervals and variables!) to the animation as you want
- Here is an example from WestGrid's 2017 *Visualize This* competition submission by Nadya Moisseeva (UBC)



`hidden/complexAnimation.mp4` on presenter's laptop

Python scripting in ParaView

Batch scripting for automating visualization

Official documentation at `https://www.paraview.org/Wiki/ParaView/Python_Scripting`

- Why use scripting?
  - automate mundane or repetitive tasks, e.g., making frames for a movie
  - document and store your workflow
  - use ParaView on clusters from the command line and/or via batch jobs
- In the GUI: View | Python Shell opens a Python interpreter
  - write or paste your script there
  - use the button to run an external script from a file
- *[/usr/bin/ /usr/local/bin/ /Applications/Paraview*.app/Contents/bin/]* **pvpython** will give you a Python shell connected to a ParaView server (local or remote) without the GUI
- *[/usr/bin/ /usr/local/bin/ /Applications/Paraview*.app/Contents/bin/]* **pvbatch** *--force-offscreen-rendering script.py* is a serial (on some machines parallel) application using a local ParaView server ✎ make sure to save your visualization
- *[/usr/bin/ /usr/local/bin/ /Applications/Paraview*.app/Contents/MacOS/]* *paraview --script=codes/displayWireframe.py* to start ParaView GUI and auto-run the script

First script

- Bring up View | Python Shell
- "Run Script" `codes/displaySphere.py`

displaySphere.py

```
from paraview.simple import *

sphere = Sphere() # create a sphere pipeline object

print(sphere.ThetaResolution) # print one of the attributes of the sphere
sphere.ThetaResolution = 16

Show() # turn on visibility of the object in the view
Render()
```

- Can always get help from the command line

```
help(paraview.simple)      # will display a help page on paraview.simple module
help(Sphere)
help(Show)
help(sphere)   # to see this object's attributes
dir(paraview.simple)
```

Using filters

- "Run Script" `codes/displayWireframe.py`

displayWireframe.py

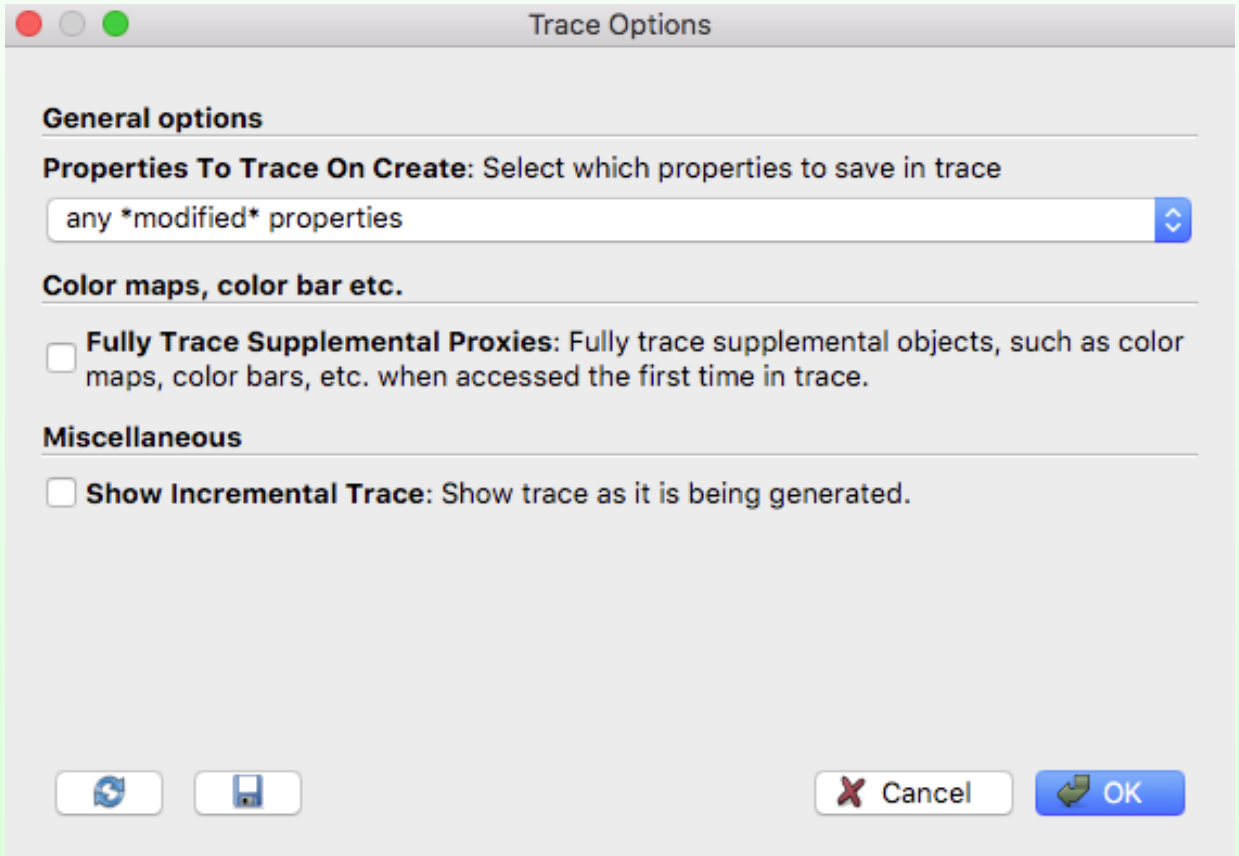
```
from paraview.simple import *

sphere = Sphere(ThetaResolution=36, PhiResolution=18)

wireframe = ExtractEdges(Input=sphere) # apply Extract Edges to sphere

Show() # turn on visibility of the last object in the view
Render()
```

- Try replacing Show() with Show(sphere)
- Also try replacing `Render()` with `SaveScreenshot('/path/to/wireframe.png')` and running via `pvbatch`

Trace tool

Generate Python code from GUI operations

- Newer ParaView: Tools | Start / Stop Trace
- Older ParaView: Tools | Python Shell | Trace | Start / Stop / Show Trace



Passing information down the pipeline

... and other useful high-level workflow functions

- `GetSources()` gets a list of pipeline objects
- `GetActiveSource()` gets the active object
- `SetActiveSource()` sets the active object
- `GetRepresentation()` returns the *view representation* for the active pipeline object and the active view
- `GetActiveCamera()` returns the active camera for the active view
- `GetActiveView()` returns the active view
- `CreateRenderView()` creates standard 3D render view
- `ResetCamera()` resets the camera to include the entire scene but preserve orientation (or does nothing ☺)

There is quite a bit of overlap between these two:

```
help(GetActiveCamera())
help(GetActiveView())
```

Camera animation with scripting

1. Let's load `data/sineEnvelope.nc` and draw an isosurface at $\rho = 0.15$
2. Compare the focal point to the center of rotation (must be the same for object to stay in view)

```
v1 = GetActiveView()
print(v1.CameraFocalPoint)
print(v1.CenterOfRotation)
```

   if not $\Rightarrow$ `ResetCamera()`

3. Look up azimuthal rotation

```
dir(GetActiveCamera())
help(GetActiveCamera().Azimuth)
```

4. Rotate by 10° around the view-up vector

```
camera = GetActiveCamera()
camera.Azimuth(10)
Render()
```

Camera animation: full rotation

✎ Can paste longer commands from `clipboard.txt`

5. Do full rotation and save to disk

```
nframes = 360
for i in range(nframes):
    print(v1.CameraPosition)
    camera.Azimuth(360./nframes)    # rotate by 1 degree
    SaveScreenshot('/path/to/frame%04d'%(i)+'.png')
```

6. Merge all frames into a movie at 30 fps

```
ffmpeg -r 30 -i frame%04d.png -c:v libx264 -pix_fmt yuv420p \
       -vf "scale=trunc(iw/2)*2:trunc(ih/2)*2" spin.mp4
```

Camera animation: flying towards the focal point

1. Optionally reset the view manually or with `ResetCamera()`
2. Now let's fly 2/3 of the way towards the focal point

```
initialCameraPosition = v1.CameraPosition[:]   # force a real copy
nframes = 100
for i in range(nframes):
    coef = float(i+0.5)/float(1.5*nframes)  # runs from 0 to 2/3
    print(coef, v1.CameraPosition)
    v1.CameraPosition = [((1.-coef)*a + coef*b) \
      for a, b in zip(initialCameraPosition,v1.CameraFocalPoint)]
    SaveScreenshot('/path/to/out%04d'%(i)+'.png')
```

3. Create a movie

```
ffmpeg -r 30 -i out%04d.png -c:v libx264 -pix_fmt yuv420p \
       -vf "scale=trunc(iw/2)*2:trunc(ih/2)*2" approach.mp4
```

Exercise: write and run a complete off-screen script

1. Mac/Linux/Windows: create a script with standalone ParaView GUI
   - use Start/Stop Trace
   - load `data/sineEnvelope.nc` and draw an isosurface at $\rho = 0.15$
   - save the image as PNG
2. Test-run your script with `pvbatch` on your laptop

```
$ pvbatch --force-offscreen-rendering script.py
```

   - Linux: `pvbatch` should be in one of your system's `bin` directories
   - Mac: `pvbatch` should be in `/Applications/ParaView*.app/Contents/bin`
   - Windows: `pvbatch` does not exist (or so I am told), but you can use `pvpython`
     - ➠ you will need to locate it yourself
   - those of you with a Compute Canada account can run this script on one of our HPC clusters with

```
$ module load gcc/9.3.0 paraview-offscreen/5.10.0
$ pvbatch --force-offscreen-rendering script.py
```

3. Modify the script to create some animation

Extracting data from VTK objects

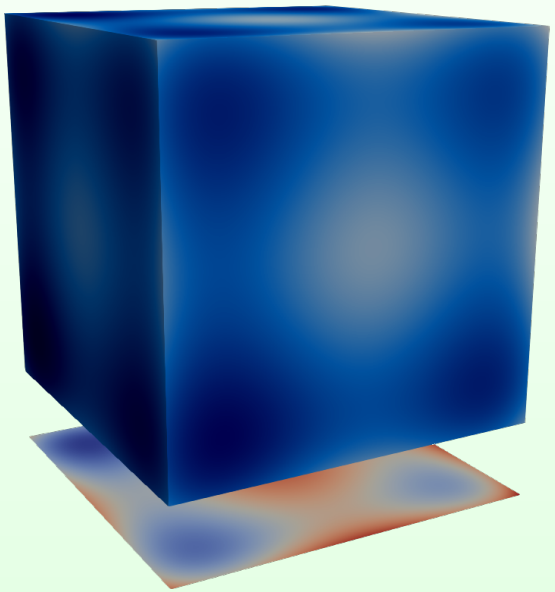
Do this from *View* | *Python Shell* or from *pvpython* (either shell will work)

```
# codes/extractValues.py
from paraview.simple import *

dir = '/Users/razoumov/training/paraviewWorkshop/data/'
data = NetCDFReader(FileName=[dir+'stvol.nc'])
local = servermanager.Fetch(data) # get the data from the server
print(local.GetNumberOfPoints())

for i in range(10):
    print(local.GetPoint(i))   # coordinates of first 10 points

pd = local.GetPointData()
print(pd.GetArrayName(0))   # the name of the first array

result = pd.GetArray('f(x,y,z)')
print(result.GetDataSize())
print(result.GetRange())

for i in range(10):
    print(result.GetValue(i))   # values at first 10 points
```

This is useful for post-processing, e.g., feeding these into **numpy arrays** and doing further calculations in a Python script

Creating/modifying VTK objects

Let's say we want to plot a projection of a cubic dataset along one of its principal axes, or do some other transformation for which there is no filter



- Calculator / Python Calculator filter cannot modify the geometry ...

Programmable filter

Watch our webinar `https://bit.ly/programmablefilter`

1. Apply Programmable Filter with OutputDataSetType = vtkUnstructuredGrid
2. Paste the following code `codes/projectionUnstructured.py` into the filter (this code was tested in ParaView 5.10.1)

```
numPoints = inputs[0].GetNumberOfPoints()
side = int(round(numPoints**(1./3.)))     # round() in this Python returns float type
layer = side*side
rho = inputs[0].PointData['density']      # 1D flat array
points = vtk.vtkPoints()       # create vtkPoints instance, to contain 100^2 points in the projection
proj = vtk.vtkDoubleArray(); proj.SetName('projection')   # create the projection array
for i in range(layer):         # loop through 100x100 points
    x, y = inputs[0].GetPoint(i)[0:2]
    z, column = -20., 0.
    for j in range(side):
        column += rho.GetValue(i+layer*j)
    points.InsertNextPoint(x,y,z)         # also points.InsertPoint(i,x,y,z)
    proj.InsertNextValue(column)          # add value to this point

output.SetPoints(points)                  # add points to vtkUnstructuredGrid
output.GetPointData().SetScalars(proj)    # add projection array to these points

quad = vtk.vtkQuad()                # create a cell
output.Allocate(side, side)         # allocate space for side^2 'cells'
for i in range(side-1):
    for j in range(side-1):
        quad.GetPointIds().SetId(0,i+j*side)
        quad.GetPointIds().SetId(1,(i+1)+j*side)
        quad.GetPointIds().SetId(2,(i+1)+(j+1)*side)
        quad.GetPointIds().SetId(3,i+(j+1)*side)
        output.InsertNextCell(vtk.VTK_QUAD, quad.GetPointIds())
```

Using 3rd-party libraries from ParaView's Python

- `pvpython` includes few common 3rd-party libraries such as `numpy`,`scipy`,`pandas`
- What if you want to use other libraries that were not bundled with ParaView?

Using 3rd-party libraries from ParaView's Python

- `pvpython` includes few common 3rd-party libraries such as `numpy`, `scipy`, `pandas`
- What if you want to use other libraries that were not bundled with ParaView?

---

1. Let's assume you work on a CC cluster; check your ParaView's Python version

```
module load gcc/9.3.0 paraview/5.10.0
pvpython   # let's assume it says Python 3.9.6
```

2. Load the closest Python module, create a virtual env. and install your library there

```
module avail python   # python/3.9.6 is one of them
module load python/3.9.6
virtualenv --no-download astro   # this will install a new virtual environment into ~/astro
source ~/astro/bin/activate
pip install --no-index --upgrade pip
pip install --no-index xarray   # install an external package into this new environment
```

3. Next time you log in to the cluster, start `pvpython`:

```
module load gcc/9.3.0 paraview/5.10.0
pvpython
```

4. Load your new virtual environment directly from Python:

```
filename = '/home/username/astro/bin/activate_this.py'
exec(open(filename).read(), {'__file__': filename})
from paraview.simple import *
import xarray     # this xarray comes from your new virtual environment
```

Remote and distributed visualization

Visualizing remote data

If your dataset is on a remote cluster, there are several options:

✘ **download data** to your desktop and visualize it locally
- limited by the dataset size and your desktop's CPU/GPU + memory

✘ run ParaView remotely on a larger machine via **X11 forwarding**
- your desktop $\xrightarrow{\text{ssh -Y}}$ larger machine running ParaView
- remote OpenGL apps with either (1) software rasterizer on the cluster (usually the default) or (2) on your laptop's GPU (need to re-enable INdirect GLX inside X11 server and set `LIBGL_ALWAYS_INDIRECT=1`)

✔ run ParaView remotely on a larger machine **via remote desktop**
- your desktop $\xrightarrow{\text{VNC}}$ larger machine running ParaView
- you can always start a VNC server on an interactive cluster compute node by hand as described in our documentation `https://bit.ly/startVNC`
- remote OpenGL apps will run either (1) using software rasterizer on the cluster (usually the default) or (2) on cluster's GPU(s) via VirtualGL wrapper (see our VNC docs)
- the VNC slide is coming up

✔ run ParaView in **client-server mode**
ParaView client on your desktop $\rightleftharpoons$ ParaView server on larger machine

✔ run ParaView via a GUI-less **batch script** (interactively or scheduled)
- render server can run with GPU rendering or purely in software
- data/render servers can run on single-core, or across several cores/nodes with MPI
- for interactive GUI work on clusters you should schedule interactive jobs, as opposed to running on the login nodes

Special remote vis cases

1. In-situ visualization = instrumenting a simulation code on the cluster to
   1.1 output graphics and/or
   1.2 act as on-the-fly server for a visualization frontend (ParaView/VisIt client on your laptop)
   - need to use a special library (ParaView's Catalyst or VisIt's libsim)
   - very advanced topic for another time

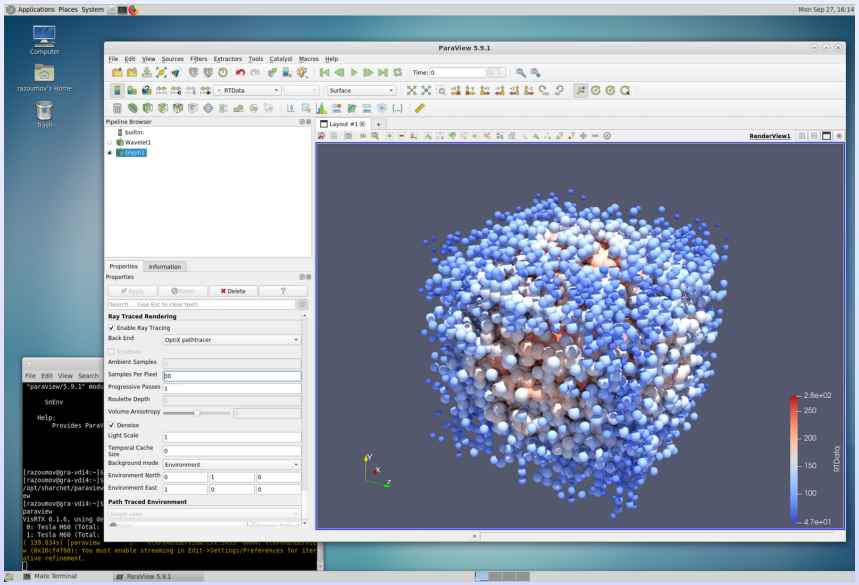
2. Web-based visualization with data served from another location

ParaView via remote desktop

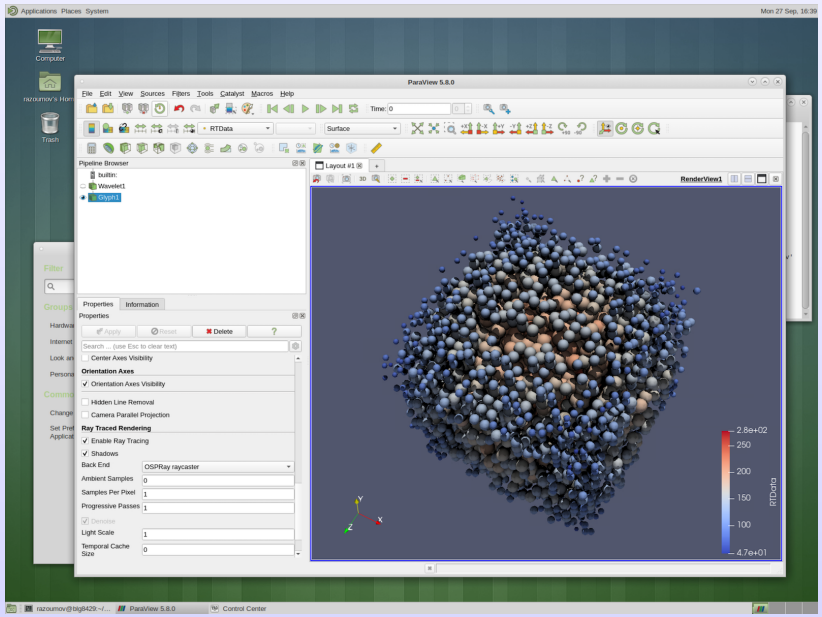
`https://docs.alliancecan.ca/wiki/VNC` or `https://docs.alliancecan.ca/wiki/JupyterHub`

You have several options:

(1) run a VNC server on compute nodes with SSH tunnelling, connect via a VNC client
`https://docs.alliancecan.ca/wiki/VNC#Compute_Nodes`



(2) VNC on gra-vdi.computecanada.ca, connect via a VNC client



(3) Remote Desktop via JupyterHub on Béluga, point your web browser at `https://jupyterhub.beluga.computecanada.ca`

Cedar, Graham, Béluga, Narval clusters

- General-purpose CC clusters for a variety of workloads
  - entered production in phases since June 2017
  - located at SFU, UofWaterloo, École de technologie supérieure (Montreal)
  - 101,568 / 44,444 / 39,120 / 80,720 CPUs
  - many hundred NVIDIA GPUs (with 12GB/16GB/32GB on-board memory)
  - multiple types of nodes, with 128GB/256GB/0.5TB/1.5TB/3TB memory
  - specs at `https://docs.alliancecan.ca/wiki/Cedar` (replace `Cedar` with `Graham` or `Beluga` or `Narval`)
- Batch-oriented environment for parallel and serial jobs $\Rightarrow$ use Slurm scheduler and workload manager
- Identical software setup `https://docs.alliancecan.ca/wiki/Available_software`

Interactive jobs on Cedar / Graham / Béluga

- **Client-server workflow** is by definition interactive
- On Cedar interactive jobs should automatically go to one of Slurm's interactive partitions (CPU or GPU)

```
$ sinfo -p cpubase_interac
    # will list nodes and their states (idle, mixed, allocated, ...)
```

- `salloc` without a script name will start an interactive shell inside a submitted job on a compute node

```
$ salloc --time=1:0:0 --ntasks=4 ... --account=def-someuser
$ echo $SLURM_...    # access Slurm variables
$ module load ...    # set your environment
$ ./serialCode
$ srun ./mpiCode     # run an MPI code
$ exit               # terminate the job (go back to the login node)
```

- You might need to specify `pvserver --server-port=11112` (etc.) if someone else is already using the default port 11111 on the same node

Question 1: should I use CPUs or GPUs for rendering?

- Can render on GPUs (*hardware acceleration*) or CPUs (*software rendering*) with both interactive and batch visualizations
  - GPUs have traditionally been faster for rendering graphics
  - in recent years better open-source software rendering libraries such as OSPRay (Intel's ray tracing) and OpenSWR (Intel's rasterizer) have largely closed the performance gap for many types of visualizations

  ⇒ I recommend starting with CPU rendering since you already likely have many CPUs! (see next slide)

- One might have to resort to software rendering if no GPUs are available, e.g., all taken by GP-GPU jobs

- I suggest doing **all hands-on exercises with CPU rendering**; also included slides on **GPU rendering** on the cluster

Question 2: how many CPUs/GPUs do I need?

- How many processors do we need? From *ParaView documentation*:
  - structured data (Structured Points, Rectilinear Grid, Structured Grid): one CPU core per ∼20 million cells
  - unstructured data (Unstructured Points, Polygonal Data, Unstructured Grid): one CPU core per ∼1 million cells
- Your main bottlenecks will be physical memory and disk read speed, and to a lesser extent CPU/GPU rendering time $\Rightarrow$ to simplify things, to decide on the number of CPU cores for initial dataset exploration, use the dataset size
  - consider 80 GB dataset
  - base nodes have 128 GB memory with 32 cores $\Rightarrow$ 3.5 GB/core (accounting for the OS, system tools, etc.) $\Rightarrow$ 23 cores for this dataset
  - need to account for filters (and other processing), MPI buffers $\Rightarrow$ minimum 32 cores
  - for comfortable processing with complex filters use $48 - 64$ cores
- On large HPC systems ParaView is known to scale to $\sim 10^{12}$ cells (Structured Points) on ∼10,000 cores and beyond
- Always do a scaling study before attempting to visualize large datasets
- It is important to understand memory requirements of filters
  - a typical structured $\rightarrow$ unstructured filter increases memory footprint by $\sim$ 3X

Remote Render Threshold

In ParaView's preferences can set Render View → Remote/Parallel Rendering Options → Remote Render Threshold beyond which rendering will be remote

- default 20MB ⇒ small rendering will be done on your laptop's GPU, interactive rotation with a mouse will be fast, but anything modestly intensive (under 20MB) will be shipped to your laptop and might be slow
- 0MB ⇒ all rendering (including rotation) will be remote, so you will be really using the cluster's CPU(s)/GPU(s) for everything
  - good for large data processing
  - not so good for interactivity, especially on a slower connection
- experiment with the threshold to find a suitable value

Next few pages: remote rendering exercises

Short version:

1. create your visualization via interactive client-server using CPU rendering
2. save your visualization to PNG

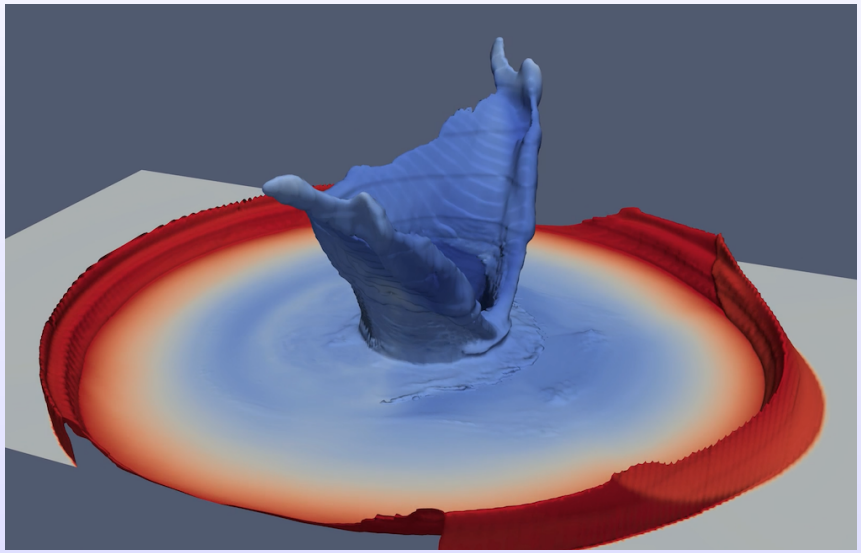
Long version:

1. create your visualization via interactive client-server using CPU rendering
2. save your visualization to PNG
3. convert this workflow into a Python script
4. upload this Python script to the cluster
5. try running the script inside an interactive (`salloc`) job; debug if needed
6. once happy with the result, write a Slurm job submission script and submit this rendering as a batch (`sbatch`) job

Exercise 1 (on Cedar): deep impact dataset

Dataset from IEEE 2018 SciVis Contest

- Dataset from *Deep Water Impact* simulation by John Patchett (LANL) and Galen Gisler (Univ. of Oslo)
  - dataset details at `https://bit.ly/2SXmjsq`
  - you can work with 269 low-resolution ($460 \times 280 \times 240$) snapshots in time
  - the original simulation is much higher resolution



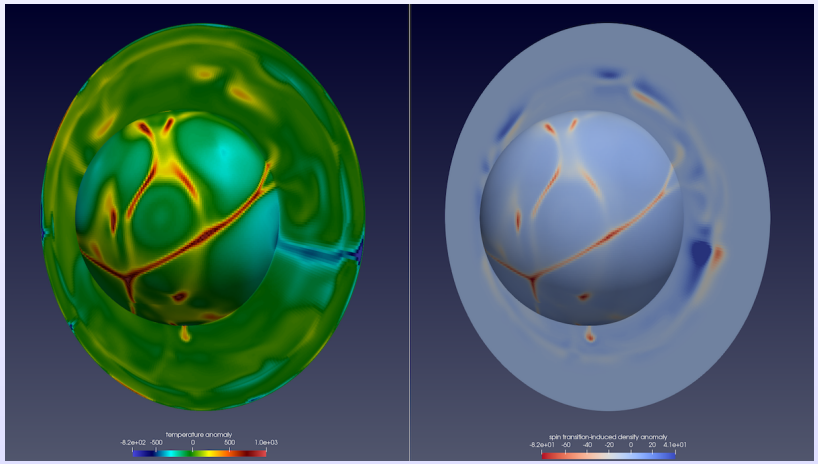
- You can render this dataset in serial
  - try to adapt the client-server instructions from "Parallel software rendering" slide (forward a few pages) to render on one CPU
- Data in `cedar:/project/6003910/razoumov/ieeevis2018/460x280x240` (115GB in total)
- To simplify navigating to the dataset in ParaView, I highly recommend creating a symbolic link:

```
[cedar]$ mkdir -p ~/data
[cedar]$ ln -s /project/6003910/razoumov/ieeevis2018/460x280x240/ ~/data/deepImpact
```

Exercise 2 (on Cedar): Earth's mantle convection

Dataset from IEEE 2021 SciVis Contest `https://scivis2021.netlify.app`

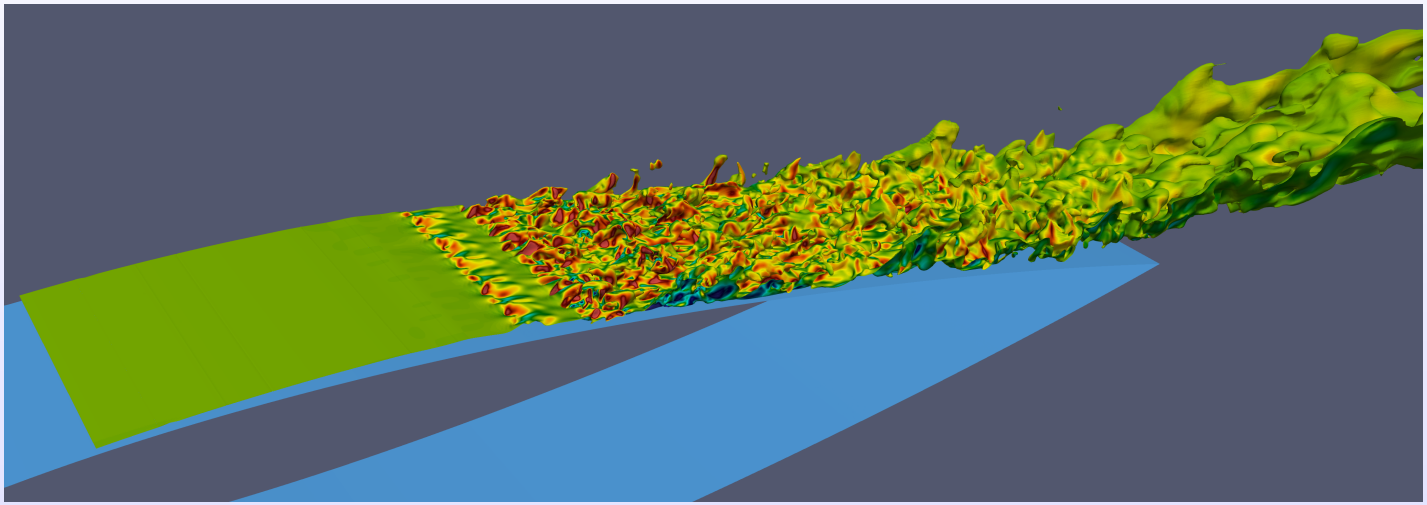
- Dataset from *Earth's Mantle Convection* simulation by Hosein Shahnas and Russell Pysklywec (U. of Toronto)
  - dataset details at `https://scivis2021.netlify.app/data`
  - 251 timesteps on a spherical $180 \times 201 \times 360$ grid
- You can render this dataset in serial
  - try to adapt the client-server instructions from "Parallel software rendering" slide (forward a few pages) to render on one CPU



- Data in `cedar:/project/6003910/razoumov/ieeevis2021/spherical` (89GB in total)
- Create a symbolic link to simplify navigating to the dataset in ParaView

Exercise 3 (on Cedar): airflow over a turbine blade

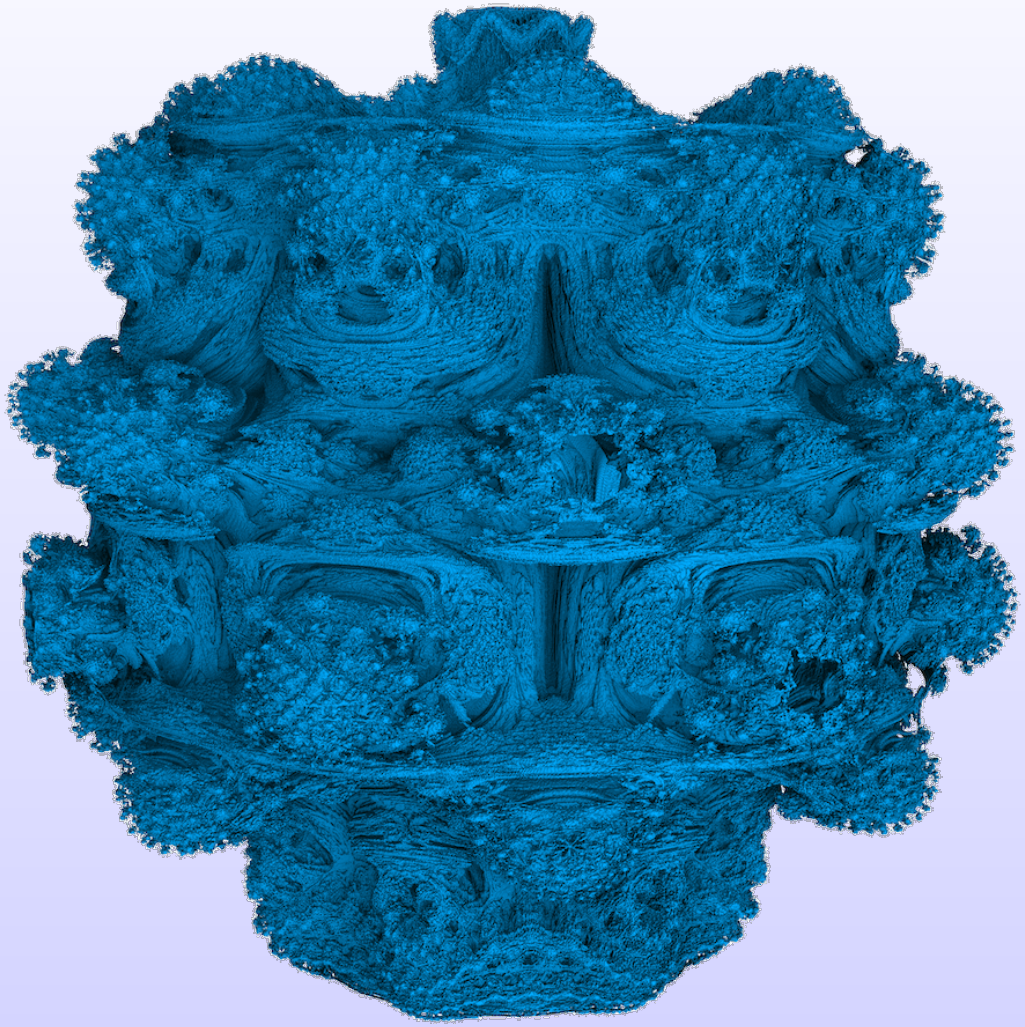
Dataset from WestGrid's 2019 `https://computecanada.github.io/visualizeThis`



- OpenFOAM *decomposed* dataset: 512 cores, 86 timesteps, 5 hydro variables, ~1TB in total
  - kindly provided for this competition by Joshua Brinkerhoff (UBC Okanagan)
  - unstructured mesh $\Rightarrow$ loading a single timestep from the 3D internal mesh requires 200GB+ physical RAM
  - the 2D airfoil mesh takes only 13.7 GB virtual memory for 1 timestep + 1 variable
  - data in `cedar:/project/6003910/razoumov/visThis2019`
- Image at the top shows the isosurface of constant air speed coloured by the Y-component of the vorticity, full animation rendering (86 timesteps) took 17 minutes on 128 Cedar CPU cores
- Create a symbolic link to simplify navigating to the dataset in ParaView

Exercise 4 (on the training cluster): Mandelbulb

- Visualize power-8 Mandelbulb
- Use the file `mandelbulb800.nc` – now sampled at 800^3
- Use 4–8 CPU cores on the training cluster via `salloc`
  1. consult the next three pages, use critical thinking – you will need to modify some of the commands!
  2. try to recreate the picture on the right: pay attention to the lights and shadows
  3. use View → Memory Inspector to keep an eye on memory usage
  4. optionally colour your dataset by `processID`



```
$ unzip /home/razoumov/shared/paraview.zip data/mandelbulb800.nc
$ ls -lh data/mandelbulb800.nc
```

Parallel software rendering

From interactive client-server debugging to remote batch rendering

1. On the cluster start remote parallel ParaView server:

```
$ cd scratch   # necessary on Cedar
$ module load StdEnv/2020 gcc/9.3.0 openmpi/4.0.3 paraview-offscreen/5.10.0
$ salloc --time=0:60:0 --ntasks=128 --mem-per-cpu=3600 --account=def-someuser
$ mpirun -np 128 pvserver
```

2. Wait for it to start waiting for incoming connection:

```
Waiting for client...
Connection URL: cs://cdr774.int.cedar.computecanada.ca:11111
Accepting connection(s): cdr774.int.cedar.computecanada.ca:11111
```

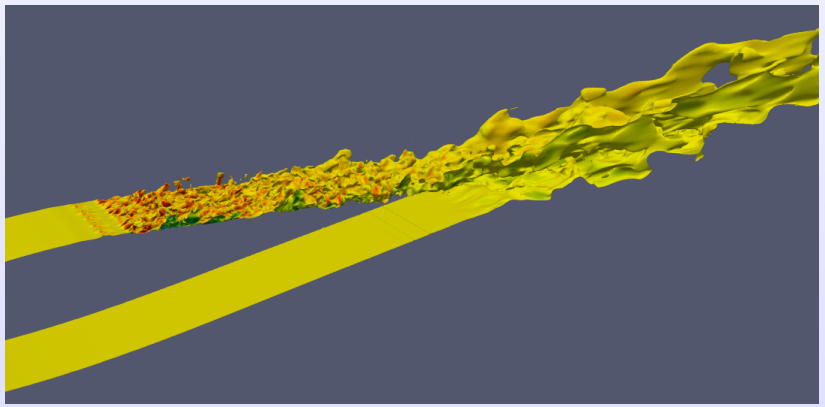
3. On your laptop start SSH port forwarding:

```
$ ssh cedar.computecanada.ca -L 11111:cdr774:11111  # use the actual compute node
```

4. On your laptop start ParaView 5.10.x, click Connect, then connect to `cs://localhost:11111`

Parallel software rendering (cont.)

5. Tools → Start Trace
6. Load OpenFOAM data, set Case Type = Decomposed
7. Apply Calculator: speed = mag(U)
8. Apply Contour at `speed=0.8`
9. Colour by $(\text{vorticity})_y$
10. Load *Rainbow Desaturated* colourmap
11. Save the image as a PNG file
12. Tools → Stop Trace
13. Save the generated script as `airflow.py` locally
    - edit it in a text editor, simplify (most generated lines will be setting defaults)
    - provide the correct output PNG path on the remote system



Parallel software rendering (cont.)

14. Upload the script to the cluster:

```
$ scp airflow.py cedar.computecanada.ca:scratch/
```

15. On the cluster try running it as a parallel interactive job:

```
$ cd ~/scratch
$ salloc --time=0:60:0 --ntasks=128 --mem-per-cpu=3600 --account=def-someuser
$ module load gcc/9.3.0 paraview-offscreen/5.10.0
$ mpirun -np 128 pvbatch --force-offscreen-rendering airflow.py
```

16. Once you are happy with the result, write a Slurm job submission script and submit it with `sbatch`

OpenGL context for off-screen rendering on a GPU

To render on a GPU from an OpenGL application such as ParaView, **traditionally you would require**:

1. OpenGL support in the GPU driver, and
2. an X server that handles windows and surfaces onto which client APIs can draw
   - run X11 server (typically started by root) on the GPU compute node, set DISPLAY=:0.$gpuindex (get GPU index from Slurm)

Latest NVIDIA GPU drivers include EGL (*Embedded-System Graphics Library*) support enabling creation of an OpenGL context for off-screen rendering without an X server.

- Your OpenGL application needs to be **recompiled with EGL support** $\Rightarrow$ use a special version of ParaView for GPU rendering without an X server; currently compiled into a module `paraview-offscreen-gpu/5.10.0` that provides both **`pvserver`** for client-server and **`pvbatch`** for batch rendering
- Unlike X11, EGL does not require any special setting to scale to very high resolutions, e.g., 4K (3840×2160) – simply ask it to render a 4K image

Interactive client-server rendering on a cluster's GPU

Details in `http://bit.ly/2wrSvKV`

1. On Cedar/Graham/Béluga **submit an interactive job** to the GPU partition, e.g., a serial job:

```
$ salloc --time=0:30:0 --ntasks=1 --gpus-per-node=[type:]count \
         --mem-per-cpu=3600 --account=def-someuser
```

   When the job starts, it'll return a prompt on the assigned compute node.

2. On the compute node inside the job **start the ParaView server** using a special version of ParaView with EGL support

```
$ module load gcc/9.3.0 paraview-offscreen-gpu/5.10.0
$ unset DISPLAY   # so that PV does not attempt to use X11 rendering context
$ pvserver        # --egl-device-index=0 not needed: first available GPU
                  #    is #0 inside the job
```

   For multiple GPUs can use

```
$ nvidia-smi -L    # will return 0, 1, ...
```

   The `pvserver` command will return something like

```
Waiting for client...
Connection URL: cs://cdr347.int.cedar.computecanada.ca:11111
Accepting connection(s): cdr347.int.cedar.computecanada.ca:11111
```

Interactive client-server rendering on a cluster's GPU (cont.)

3. On your desktop **set up ssh forwarding** to the ParaView server port:

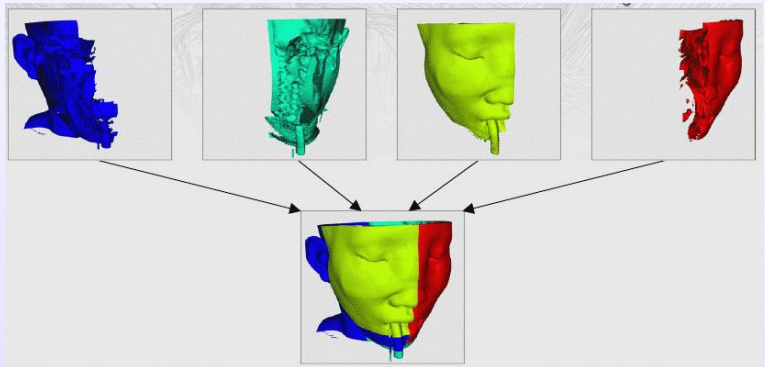
```
$ ssh username@cedar.computecanada.ca -L 11111:cdr347:11111
```

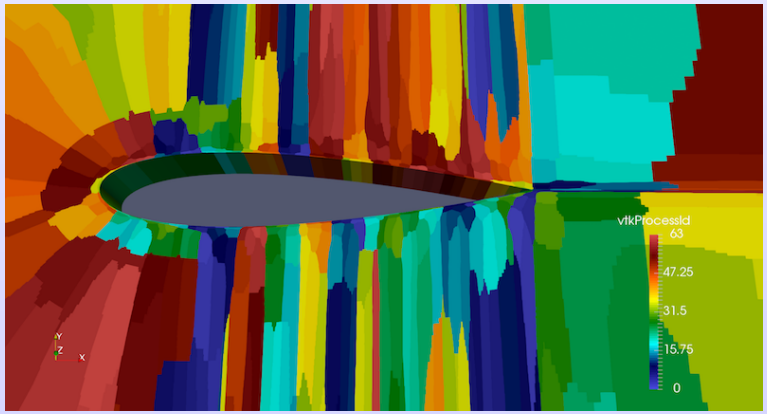
4. On your desktop **start ParaView 5.10.x** and **edit its connection properties** under *File - Connect - Add Server* (name = Cedar, server type = Client/Server, host = localhost, port = 11111), click *Configure* → *Manual* → *Save*, then select the server from the list and click on *Connect*

- ParaView's client and server must have matching major versions (5.10.x)

Data partitioning in parallel ParaView

- If loading unpartitioned data ⇒ dynamic load balancing is handled automatically for structured data:
  - structured points
  - rectilinear grid
  - structured grid
- Unpartitioned unstructured data will usually be read in serial, then must be passed through D3 (Distributed Data Decomposition) filter for dynamic load balancing:
  - particles/unstructured points
  - polygonal data
  - unstructured grid
- Some unstructured file formats can be read in parallel, e.g. the OpenFOAM reader will automatically read its unstructured data in parallel, distributing it among all available CPU cores
- After passing your unstructured data through D3, you can save it as parallel PVTU file ⇒ you'll get a statically distributed dataset that you can load next time with the same number of CPU cores
- Reading time ⇒ you can usually tell if your dataset is being read in serial or in parallel
- Look for `vtkProcessID` variable





Data partitioning in parallel ParaView (cont.)

If you have a large (many GBs) .vtu file:

1. Read your serial .vtu file into parallel ParaView on 16 cores - slow
   - and hope that it does not run out of memory on the reading core!
   - at this point the dataset is sitting in memory on one core
   - example: serial .vtu file at 9.1GB $\Rightarrow$ 1'49" reading time
2. Apply D3 filter to distribute the dataset - slowish (memory + MPI)
3. File $\rightarrow$ Save data as .pvtu with lz4 level-6 (fast) compression - fast
   $\Rightarrow$ 16 files + 1 header file
   - now you have a statically decomposed dataset
4. Restart parallel ParaView on 16 cores, read .pvtu from scratch into - fast!
   - at this point the dataset is distributed across all 16 cores
   - example: same (but now decomposed) .pvtu dataset at 5.1GB (fast compression) $\Rightarrow$ 11" reading time

- The same I/O speeds logic applies to .vti $\rightarrow$ .pvti (but there is no need for D3)

Exercise: parallel rendering of partitioned data

This is an extremely concise step-by-step guide for the turbine dataset:

1. Submit an interactive job
   `salloc --time=0:60:0 --ntasks=16 --mem-per-cpu=3600`
2. Start client-server ParaView session on 16 cores
3. Load all .vtm files (all 10 timesteps)
4. Apply Merge Blocks, output type = Unstructured Grid
5. Apply Cell Data to Point Data (so that you could use Contour)
6. Apply D3
7. Save data as `decomposed.pvtu`, write all timesteps as series, fast compression
8. Restart client-server ParaView session on 16 cores
9. Load all `decomposed.pvtu` files
10. Create visualization interactively
11. Save animation as 1000 × 800 PNG files ✍ this step should take ∼ 1min of processing time
12. Merge them into a movie with `ffmpeg`

Remote rendering summary: some orthogonal decisions

(1) interactive vs. batch

- interactive client-server for a quick look, exploration or debugging
  - another option is to download a scaled-down version of your dataset, debug a script locally on your laptop, and then run it as a batch job on the original full-resolution dataset on the cluster
- batch really preferred for production jobs and producing animations

(2) CPU vs. GPU

- in general, no single answer which one is better
  - you can throw many CPUs at your rendering job
  - modern software rendering libraries such as OSPRay (Intel's ray tracing) and OpenSWR (Intel's rasterizer) can be very fast, depending on your visualization
- might have to resort to software rendering if no GPUs are available (e.g., all are taken by GP-GPU jobs)
- for initial exploration, I would use the dataset size (GBs) to figure out the best number of CPU cores, and adjust from there

SUMMARY

Further resources

- ParaView Discourse
  `https://discourse.paraview.org`
- Self-directed ParaView tutorial
  `https://docs.paraview.org/en/latest/Tutorials/SelfDirectedTutorial/index.html`
- ParaView F.A.Q.
  `http://www.itk.org/Wiki/ParaView:FAQ`
- VTK wiki with webinars, tutorials, etc.
  `http://www.vtk.org/Wiki/VTK`
- VTK for C++/Python/Java/C#/JavaScript code examples
  `https://kitware.github.io/vtk-examples`
- VTK file formats (3rd-party intro)
  `http://www.earthmodels.org/software/vtk-and-paraview/vtk-file-formats`

Our visualization webinars

- ~3-4 visualization webinars per academic year
  - ✎ keep an eye on our emails, Twitter, `https://westgrid.github.io/trainingMaterials/blog`
  - ▶ ~50 mins + questions, usually on fairly specific or advanced topics
- Many past webinars are available with slides and screencasts at `https://bit.ly/vispages`
  - "Highlights from the 2021 SciVis Contest"
  - "Remote visualization on Compute Canada clusters"
  - "Scientific visualization on NVIDIA GPUs"
  - "Workflows with Programmable Filter / Source in ParaView"
  - "The Topology ToolKit (TTK)"
  - "Web-based 3D scientific visualization" (ParaViewWeb, vtk.js, ParaView Glance)
  - "Photorealistic rendering with ParaView and OSPRay"
  - "Batch visualization on Compute Canada clusters"
  - "Molecular visualization with VMD"
  - "Intermediate VMD topics: trajectories, movies, scripting"
  - "Using YT for analysis and visualization of volumetric data" (part 1)
  - "Working with data objects in YT" (part 2)
  - "Scientific visualization with Plotly"
  - "Novel visualization techniques from 2017 VISUALIZE THIS competition"
  - "Camera animation in ParaView and VisIt"
  - "3D visualization on new Compute Canada systems"
  - "Using ParaViewWeb for 3D visualization and data analysis in a web browser"
  - "Visualization support in WestGrid / Compute Canada"
  - "Scripting and other advanced topics in VisIt visualization"
  - "CPU-based rendering with OSPRay"
  - "3D graphs with NetworkX, VTK, and ParaView"
  - "Graph visualization with Gephi"
- We are always looking for topic suggestions!

Documentation and getting help

- Visualization in the Alliance `https://ccvis.netlify.app` (online gallery)
- Official documentation `https://docs.alliancecan.ca/wiki/Visualization`
- Western Canada research computing visualization resources `https://bit.ly/vispages`
- Email **support@tech.alliancecan.ca** and mention *"visualization"* in the subject line (goes to our ticketing system)
- Email me **alex.razoumov@westdri.ca**
- ParaView documentation
  - official documentation `http://www.paraview.org/documentation`
  - wiki `http://www.paraview.org/Wiki/ParaView`
  - Python batch scripting `http://bit.ly/2wF5v0B`
  - VTK tutorials `http://www.itk.org/Wiki/VTK/Tutorials`