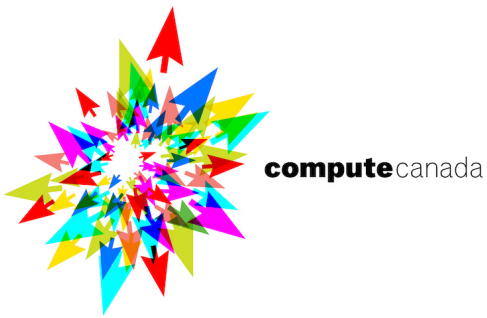
Scientific visualization with ParaView

Alex Razoumov
alex.razoumov@westgrid.ca



- ✔ slides, data, codes in `http://bit.ly/paraviewzip`
  - the link will download a file `paraview.zip`
  - unpack it to find `codes/`, `data/` and `slides.pdf`
- ✔ install ParaView 5.x on your laptop from `http://www.paraview.org/download`

Workshop outline

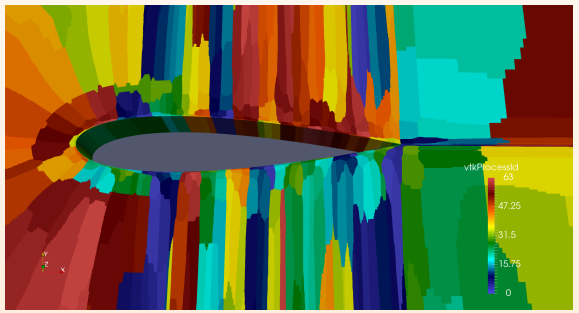
- Introduction to scientific visualization: general ideas, tools, plotting vs. multi-dimensional
- Overview of current general-purpose multi-dimensional vis. tools

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

- Scripting: ways to run scripts, few simple scripts, trace tool, programmable filter/source, camera animations
- Animation: three approaches, one big exercise on scripting/animation
- Remote visualization: overview of methods, demos of remote client-server rendering on GPUs and CPUs, batch rendering, large-dataset load balancing

Advanced topics

- Briefly on loading OpenFOAM data
- Advanced animation techniques via the GUI
  - camera animation
  - Stream Tracer With Custom Source through a slice
  - integration time contours along streamlines
  - animating many properties in a single timeline



- Camera animation via scripting
  - we'll write and run an off-screen ParaView Python script
- Remote visualization
  - running ParaView in **client-server** mode
  - opening 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: ParaView Cinema, in-situ visualization with Catalyst

Scientific visualization

- Visualization 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

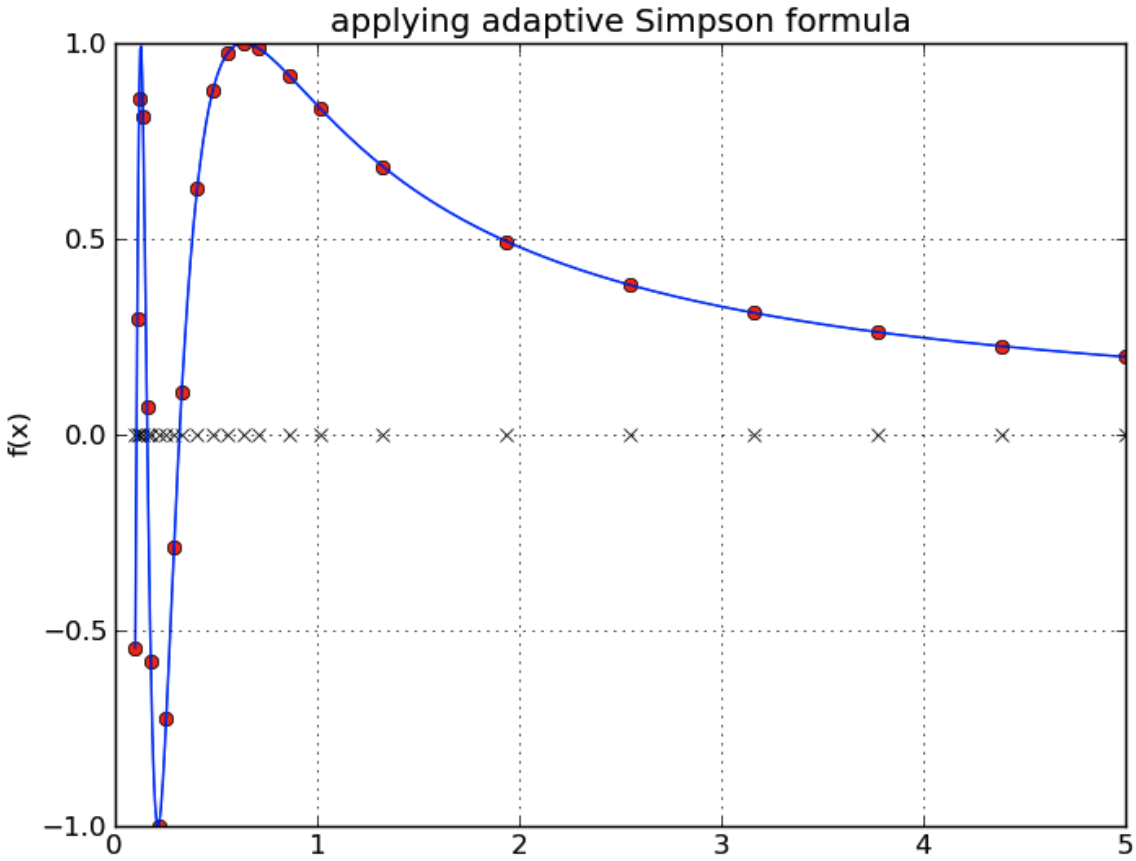
FIELD	VISUALIZATION TYPE
computational fluid dynamics	2D/3D flows, density, temperature, tracers
climate, meteorology, oceanography	fluid dynamics, clouds, chemistry, etc.
quantum chemistry	wave functions
molecular dynamics (phys, chem, bio)	particle/molecular data,
astrophysics	2D/3D fluids, particle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

1D plotting vs. 2D/3D visualization

- **1D plotting**: plotting functions of one variable, 1D tabulated data
  - something as simple as gnuplot or pgplot
  - highly recommend: Python's Matplotlib library, other Python libraries
  - another excellent option: R's ggplot2 library
- **2D/3D visualization**: displaying multi-dimensional datasets, typically data on 2D/3D structured grids or on unstructured meshes (that have some topology in 2D/3D)
- 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

Matplotlib example: 1D plotting

Adaptive Simpson integration



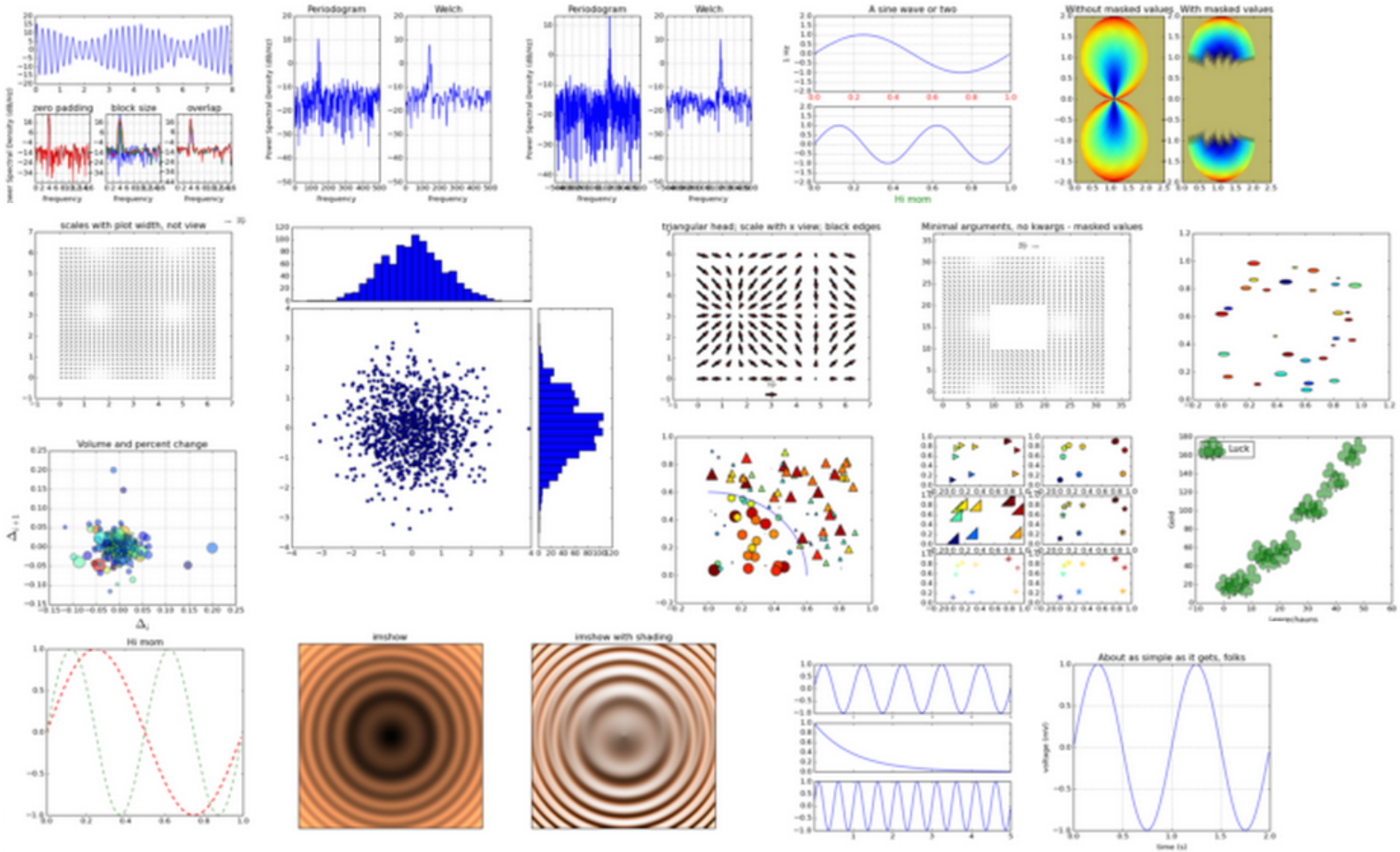
- Simple Python function `simpsonAdaptive(function,a,b,tolerance)` handles both calculation and plotting (∼40 lines of code)
- Code in `codes/adaptive.py`

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



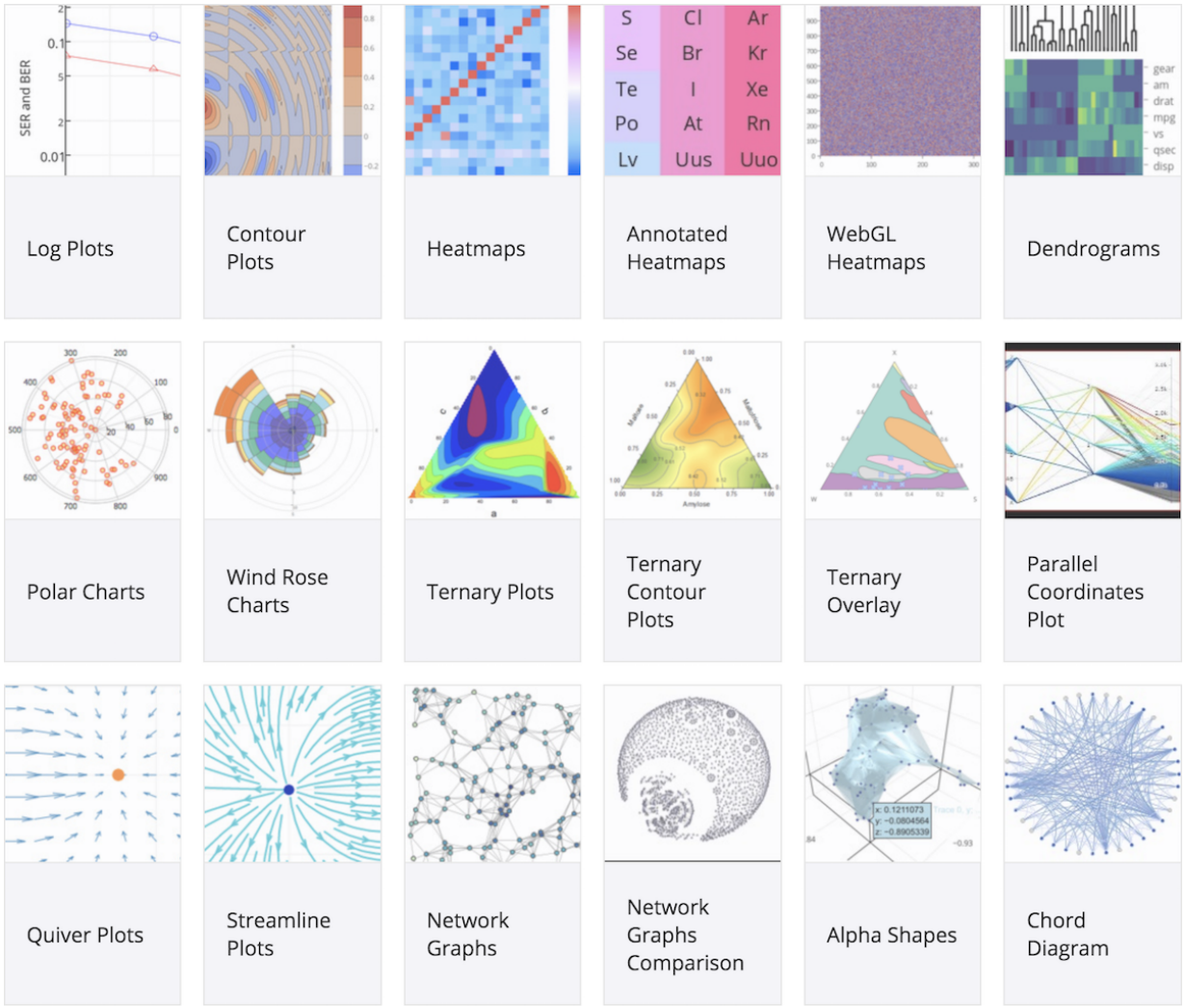
- `http://matplotlib.org/gallery.html` – click on any plot to get its source code

Bokeh gallery

- Open-source project from Continuum Analytics
  `http://bokeh.pydata.org/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 Python graphics and visualization libraries

PACKAGE	DESCRIPTION
MayaVi	serial 3D scientific data visualizer (Python + VTK)
yt project	visualization of volumetric multi-resolution data from astrophysical simulations (Enzo, FLASH, etc.)
Neuronvisio	GUI for NEURON simulator enviroment
VPython	3D graphics library
PyVisfile	storing data in a variety of scientific vis. file formats
PyVTK	tools for manipulating VTK files in Python
Chaco	interactive 2D plotting
NodeBox for OpenGL	2D animations (originally for game development)
ggplot from $\hat{y}$hat	plotting system for Python based on R's ggplot2
Seaborn	statistical data visualization based on Matplotlib

Other vis. tools you may find on large clusters

PACKAGE	DESCRIPTION
Avizo	general-purpose 3D visualization
Tablet	viewer for genome sequence assembly and alignment
Molden	visualization of molecular and electronic structure
WebMO	web portal for computing/visualization in chemistry
XCrySDen	crystalline and molecular structure visualisation
GNU Data Language (GDL)	data analysis and visualization in astronomy, geosciences and medical imaging; open-source implementation of IDL
GDIS	visualization of molecular and periodic structures
Molekel	molecular visualization
Ncview	visual browser for NetCDF files
ParaView	general-purpose scientific visualization
Rasmol	molecular visualization
VisIT	general-purpose scientific visualization
VTK	Visualization Toolkit library
VMD	visualization of large biomolecular systems

2D/3D visualization packages

- Open-source, multi-platform, and general-purpose:
  - 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 large ($10^3 - 10^5$ cores) computing facilities
  - interactive manipulation
  - support for scripting, common data formats, parallel I/O

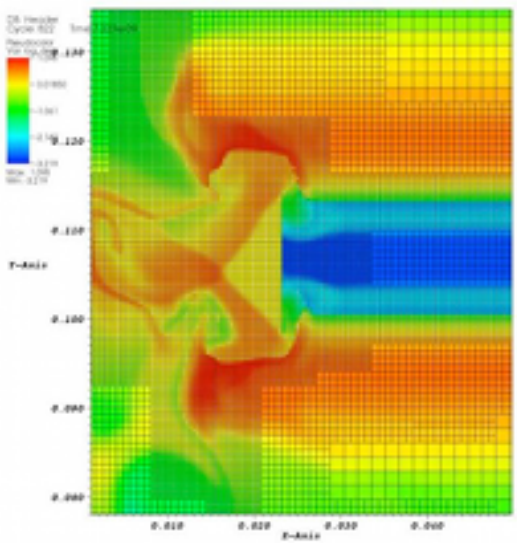
(1) VisIt (latest is 2.13.1)

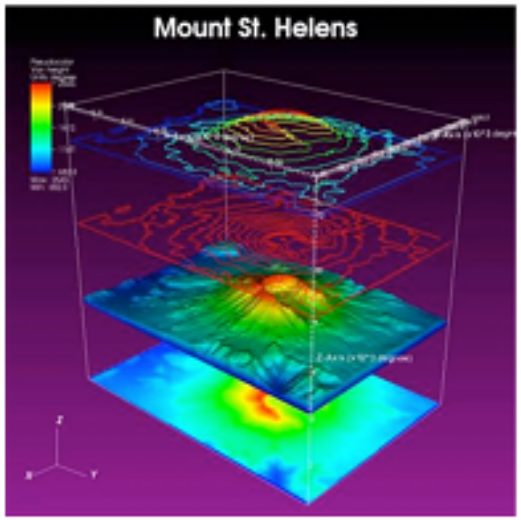
(2) ParaView (latest is 5.5)

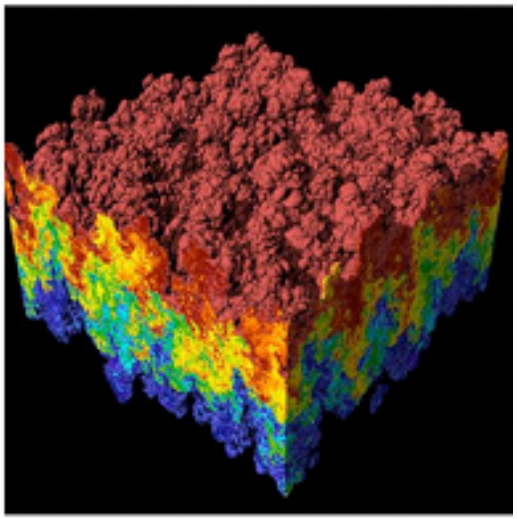
VisIt

https://wci.llnl.gov/simulation/computer-codes/visit

- Developed by the Department of Energy (DOE) Advanced Simulation and Computing Initiative (ASCI) to visualize results of terascale simulations, first release fall of 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





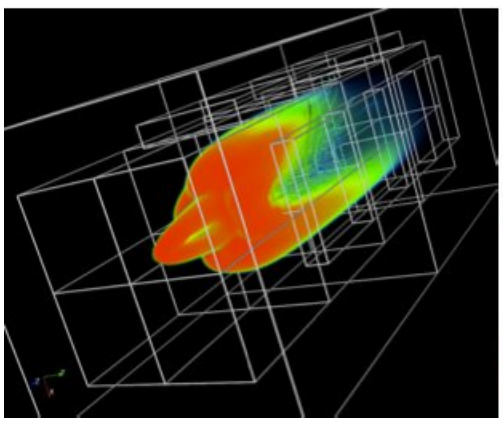
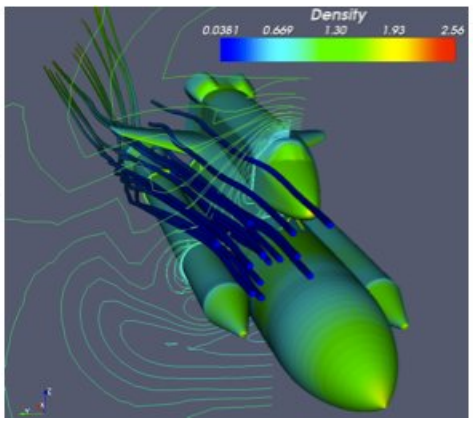


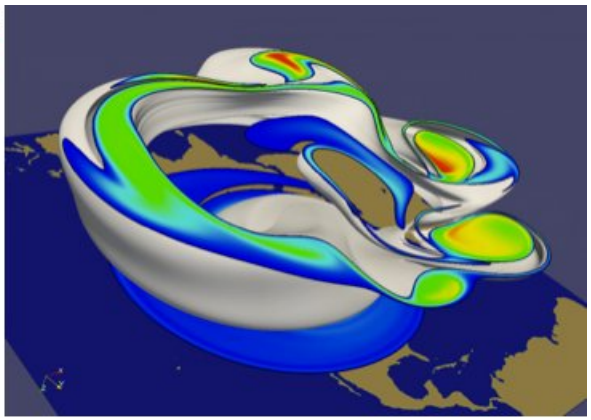
Lawrence Livermore National Laboratory

ParaView

http://www.paraview.org

- Started in 2000 as a collaboration between Los Alamos National Lab and Kitware Inc., later joined by Sandia National Labs
- Available as source and binary for Linux/Mac/Windows
- To visualize extremely large datasets on distributed memory machines
- Both interactive and Python scripting
- Uses MPI for distributed-memory parallelism on HPC clusters
- ParaView is based on VTK (Visualization Toolkit); not the only VTK-based open-source scientific visualizer, e.g. see MayaVi (written in Python + numpy + scipy + VTK) or VisIt; note that VTK can be used from C++, Tcl, Java, Python as a standalone renderer



Why ParaView for this workshop?

- When I was developing the first version of this course in 2010, I had to pick one
  - both's binaries are widely available, 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)
  - **KiwiViewer** is a mobile remote plugin to control ParaView from iOS/Android
  - **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)
- Stereoscopic viewing on 3D hardware; experimental build for Oculus Rift and HTC Vive
- We also teach and promote VisIt (`https://visit.llnl.gov`) and other open-source packages

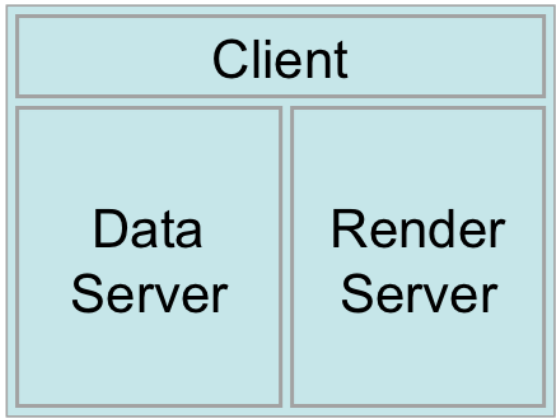
ParaView architecture and GUI

ParaView's distributed parallel architecture

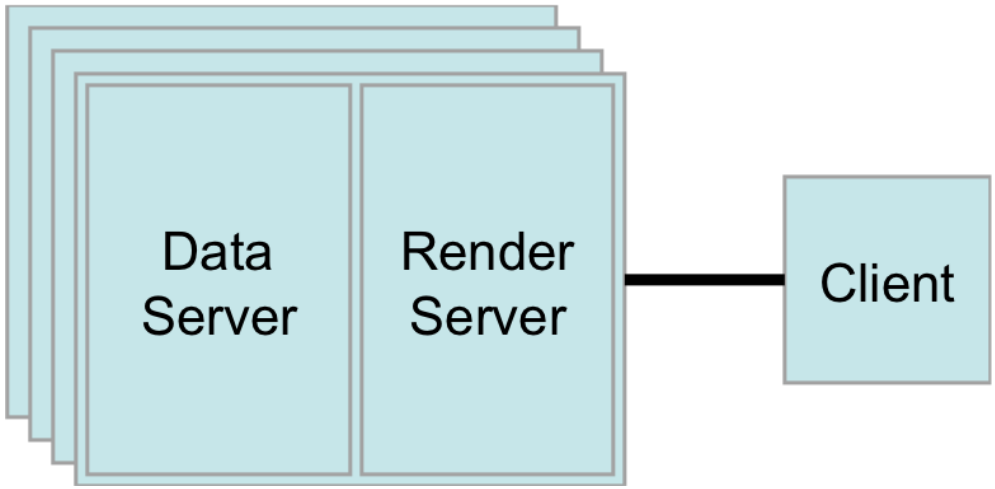
Three logical components inside ParaView – these units can be embedded in the same application on the same computer, but can also run on different machines:

- **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 on a distributed cluster

Advantages of remote client-server rendering

- Standalone ParaView has its limitations: limited memory, limited I/O bandwidth, limited CPU power, and limited 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 (and high-precision!) 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 8 nodes (= 64 cores) on colosse.calculquebec.ca with pvserver taking $\sim$ 120 GB memory
- We'll study remote ParaView in more detail towards the end of this workshop

Starting ParaView

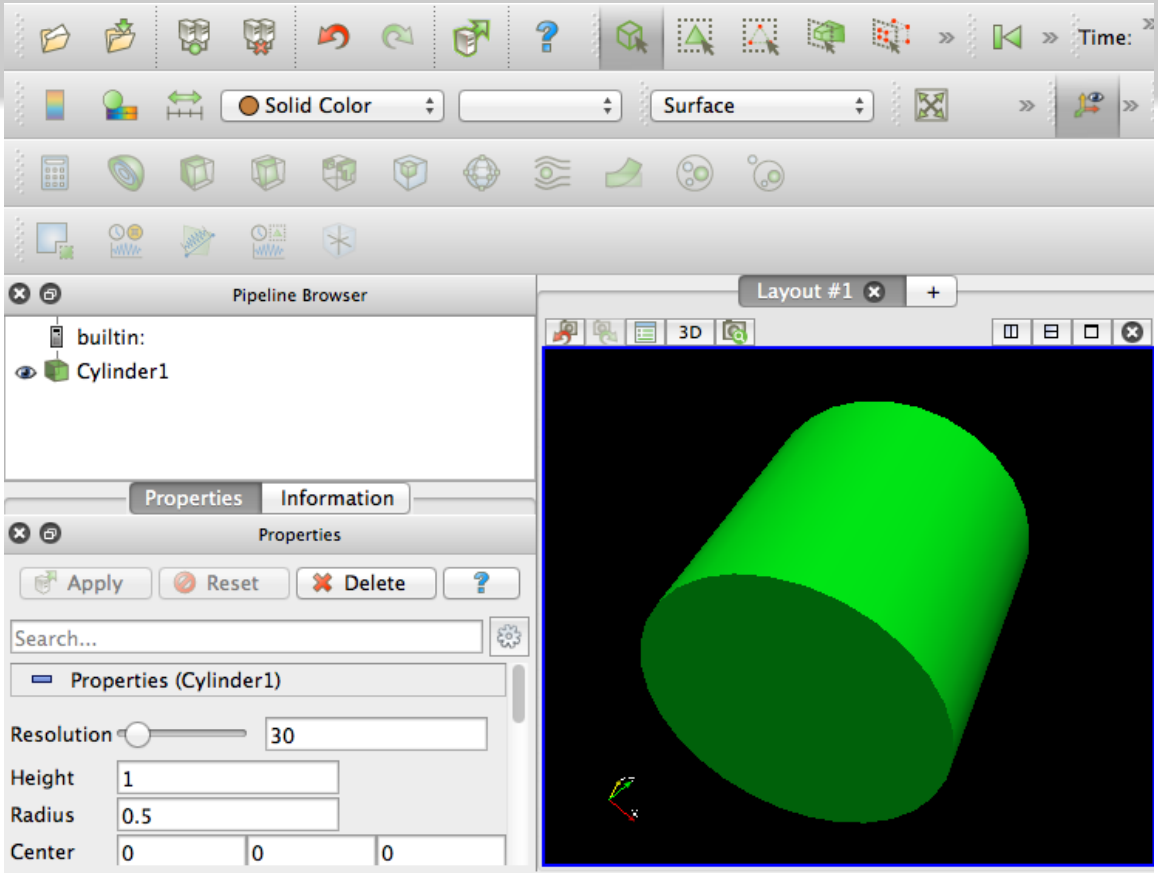
- Today we'll do everything in standalone ParaView on your desktop

- 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

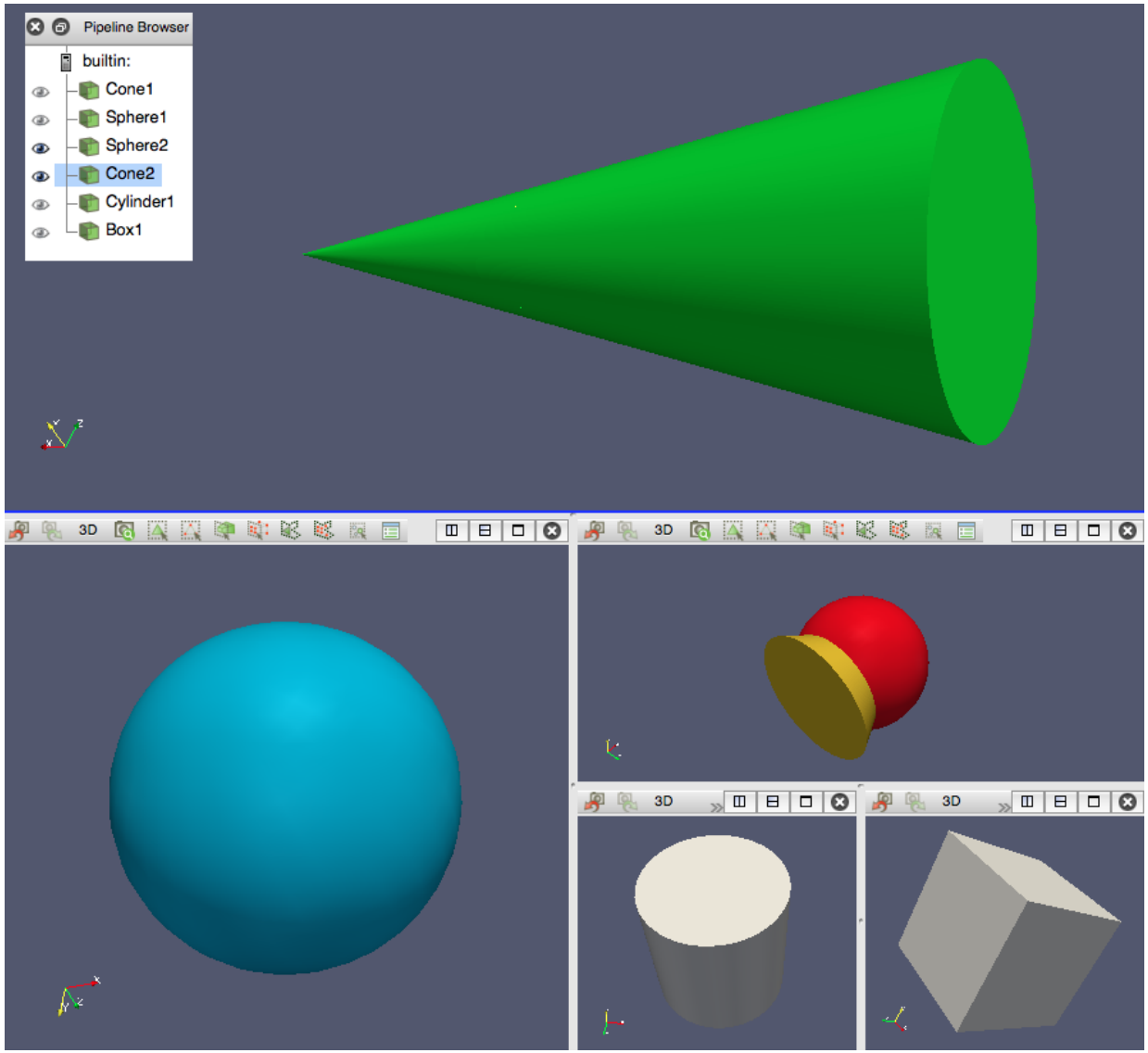
- **Pipeline Browser**: data readers, data filters, can turn visibility of each object on/off
- **Object Inspector**: view and change parameters of the current pipeline object (via tabs properties-display-info or properties-info)
- **View window**: displays the result



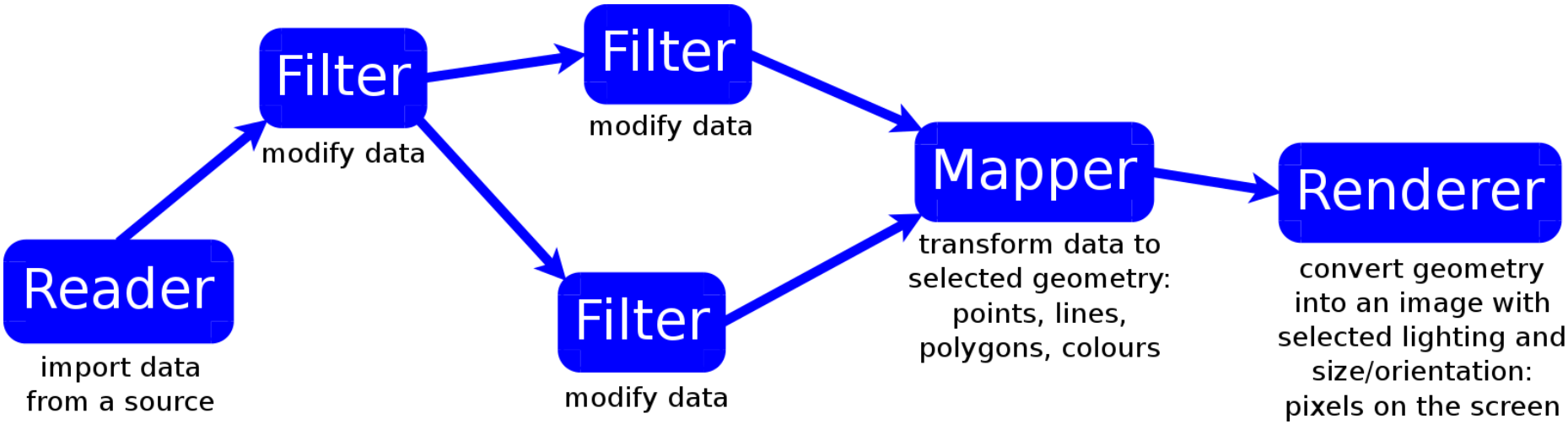
1. Find the following in the toolbar: "Connect", "Disconnect", "Toggle Colour Legend Visibility", "Edit Colour Map", "Rescale to Data Range"
2. Load a predefined dataset: in ParaView select Sources → Cylinder
3. Try dragging the cylinder using the left mouse button; also try the same with the right and 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



Visualization pipeline



- Mapper and Renderer always present but do not show in the Pipeline Browser
  - can still edit their properties via various menus and settings
- Pipeline components can be combined in many different ways to create a visualization
- Developers can add new components to extend package's functionality, e.g. ParaView allows python scripts as filters

Importing data into ParaView

Data sources

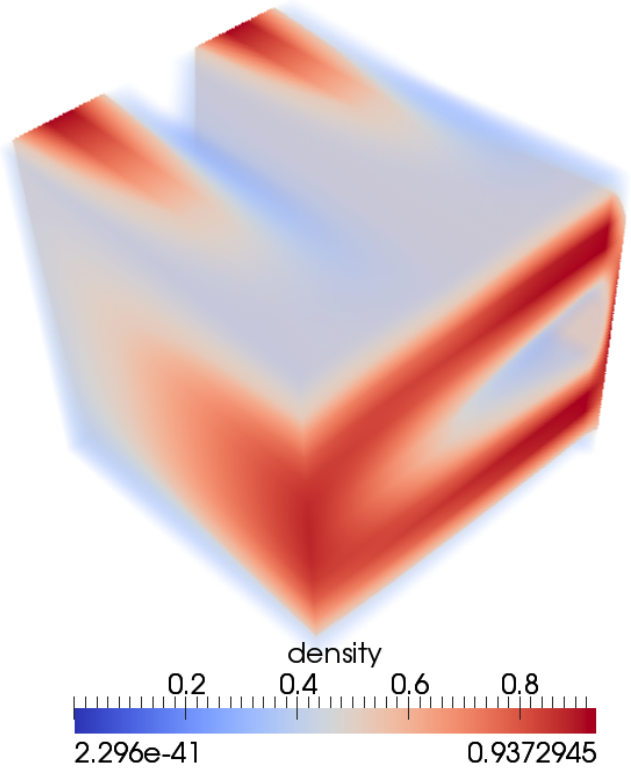
- Generate data with a *Source* object
- Read data from a file

ADAPT Files(*.nc *.cdf *.elev *.ncd)
AMR Enzo Files(*.boundary *.hierarchy)
AMR Flash Files(*.Flash *.flash)
ANSYS Files(*.inp)
AUXFile Files(*.aux)
AVS UCD Binary/ASCII Files(*.inp)
Adaptive cosmo files(*.cosmo)
BOV Files(*.bov)
BYU Files(*.g)
CAM NetCDF (Unstructured)(*.nc *.ncdf)
CCSM MTSD Files(*.nc *.cdf *.elev *.ncd)
CCSM STSD Files(*.nc *.cdf *.elev *.ncd)
CEAucd Files(*.ucd *.inp)
CGNS Files(*.cgns)
CMAT Files(*.cmat)
CML(*.cml)
CTRL Files(*.ctrl)
Case file for restarted CTH outputs(*.spcth-timeseries)
Chombo Files(*.hdf5 *.h5)
Claw Files(*.claw)
Cosmology Files(*.cosmo64 *.cosmo)
Curve2D Files(*.curve *.ultra *.ult *.u)
DDCMD Files(*.ddcmd)
DICOM Files (directory)(*.dcm)
DICOM Files (single)(*.dcm)
Delimited Text(*.csv *.txt *.CSV *.TXT)
Digital Elevation Map Files(*.dem)
Dyna3D Files(*.dyn)
ENZO AMR Particles Reader(*.boundary *.hierarchy)
EnSight Files(*.case *.CASE *.Case)
EnSight Master Server Files(*.sos *.SOS)
Enzo Files (VisIt)(*.hierarchy *.boundary)
ExodusII(*.g *.e *.ex2 *.ex2v2 *.exo *.gen *.exoII *.0 *.0(
ExtrudedVol Files(*.exvol)
FLASH AMR Particles Reader(*.Flash *.flash)
FLASH Files (VisIt)(*.flash *.f5)
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)
Facet Polygonal Data Files(*.facet)
Fluent Case Files(*.cas)
Fluent Files (VisIt)(*.cas)
GGCM Files(*.3df *.mer)
GMV Files(*.gmv)
GTC Files(*.h5)
GULP Files(*.trg)
Gadget Files(*.gadget)
Gaussian Cube Files(*.cube)
GenericIO files(*.gio)
H5Nimrod Files(*.h5nimrod)
Image Files(*.pnm *.ppm *.sdt *.spr *.imgvol)
JPEG Image Files(*.jpg *.jpeg)
LAMMPS Dump Files(*.dump)
LAMMPSStructure Files(*.eam *.meam *.rigid *.lammps
LODI Files(*.nc *.cdf *.elev *.ncd)
LODI Particle Files(*.nc *.cdf *.elev *.ncd)
LSDyna(*.k *.lsdyna *.d3plot d3plot)
Legacy VTK Files (partitioned)(*.pvtk)
Legacy VTK files(*.vtk)
Lines Files(*.lines)
M3DC1 Files(*.h5)
MFIX Unstructured Grid Files(*.RES)
MFiX Res Files (VisIt)(*.RES)
MFiX netcdf Files(*.nc)
MM5 Files(*.mm5)
MPAS NetCDF (Unstructured)(*.ncdf *.nc)
Meta Image Files(*.mhd *.mha)
Metafile for restarted exodus outputs(*.ex-timeseries)
Miranda Files(*.mir *.raw)
Multilevel 3D Plasma Files(*.m3d *.h5)
NASTRAN Files(*.nas *.f06)
Nek5000 Files(*.nek3d *.nek2d *.nek5d *.nek5000 *.nek
Nrrd Raw Image Files(*.nrrd *.nhdr)
OVERFLOW Files (VisIt)(*.dat *.save)
OpenFOAM Files (VisIt)(*.controlDict)
OpenFOAM(*.foam)
PATRAN Files(*.neu)
PFLOTRAN Files(*.h5)
PLOT2D Files(*.p2d)
PLOT3D Files(*.xyz)
PLOT3D Meta Files(*.p3d)
PLY Polygonal File Format(*.ply)
PNG Image Files(*.png)
POINT3D Files(*.3D)
POP Ocean NetCDF (Rectilinear)(*.pop.ncdf *.pop.nc)
POP Ocean NetCDF (Unstructured)(*.pop.ncdf *.pop.nc
ParaDIS Files(*.prds *.data *.dat)
ParaDIS Tecplot Files(*.fld *.field *.cyl *.cylinder *.dat)
ParaView Data Files(*.pvd)
Parallel POP Ocean NetCDF (Rectilinear)(*.pop.ncdf *.p
Phasta Files(*.pht)
Pixie Files(*.h5)
Protein Data Bank Files (VisIt)(*.ent *.pdb)
Protein Data Bank Files(*.pdb)
RAW Files(*.raw)
Raw (binary) Files(*.raw)
SAMRAI Files(*.samrai)
SAR Files(*.SAR *.sar)
SAS Files(*.sasgeom *.sas *.sasdata)
SLAC Mesh Files(*.ncdf *.nc)
SLAC Particle Files(*.ncdf *.netcdf)
Silo Files(*.silo *.pdb)
Spheral Files(*.spheral *.sv)
Spy Plot History Files(*.hscth *hsct*)
SpyPlot CTH dataset(*.spct* spct*)
Stereo Lithography(*.stl)
TFT Files(*.dat *.tft)
TIFF Image Files(*.tif *.tiff)
TSurf Files(*.ts_deg83)
Tecplot Binary Files (VisIt)(*.plt)
Tecplot Files (VisIt)(*.tec *.TEC *.Tec *.tp *.TP)
Tecplot Files(*.tec *.TEC *.Tec *.tp *.TP *.dat)
Tetrad Files(*.hdf5 *.h5)
UNIC Files(*.h5)
VASP CHGCA Files(*.CHG*)
VASP OUT Files(*.OUT*)
VASP POSCAR Files(*.POS*)
VPIC Files(*.vpc)
VRML 2 Files(*.wrl *.vrml)
VTK Hierarchical Box Data Files(*.vthb *.vth)
VTK ImageData Files (partitioned)(*.pvti)
VTK ImageData Files(*.vti)
VTK MultiBlock Data Files(*.vtm *.vtmb)
VTK Particle Files(*.particles)
VTK PolyData Ensembles(*.evtp)
VTK PolyData Files (partitioned)(*.pvtp)
VTK PolyData Files(*.vtp)
VTK RectilinearGrid Files (partitioned)(*.pvtr)
VTK RectilinearGrid Files(*.vtr)
VTK StructuredGrid Files (partitioned)(*.pvts)
VTK StructuredGrid Files(*.vts)
VTK UnstructuredGrid Files (partitioned)(*.pvtu)
VTK UnstructuredGrid Files(*.vtu)
Velodyne Files(*.vld *.rst)
VisIt MetaPLOT3D Files (VisIt)(*.vp3d)
VizSchema Files(*.h5 *.vsh5)
Voronoi Tesselation(*.out *.tess)
Wavefront OBJ Files(*.obj)
WindBlade Data(*.wind)
XMol Molecule Files(*.xyz)
XYZ Files(*.xyz)
Xdmf Reader(*.xmf *.xdmf)
Xmdv Files(*.okc)
netCDF files generic and CF conventions(*.ncdf *.nc)
proSTAR Files(*.cel *.vrt)

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
2. Describe the dataset in properties:
   - Data Scalar Type = float
   - Data Byte Order = Little Endian
   - File Dimensionality = 3
   - Data Extent: 1 to 16 in each dimension
   - Scalar Array Name = density
3. Try different views: Outline, Points, Wireframe, Volume
4. Depending on the view, can edit the colour map



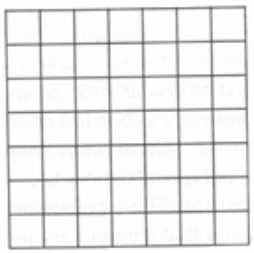
5. Try saving data as paraview data type (*.pvd), deleting the object, and reading back from *.pvd – file now contains full description of dataset

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.

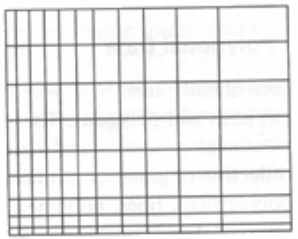
VTK 3D data: 6 major dataset (discretization) types

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



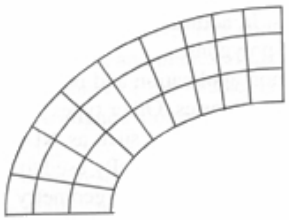
(a) Image Data

- **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

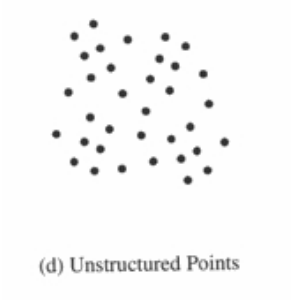
- **Structured Grid**: *.vts, regular topology and irregular geometry, need to indicate coordinates of each point



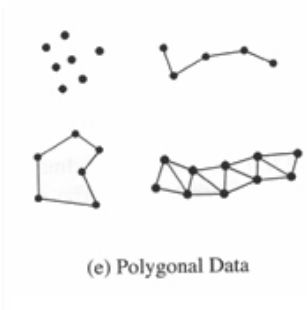
(c) Structured Grid

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

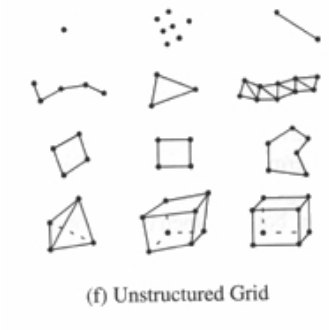
- **Particles/Unstructured Points**: *.particles


(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.

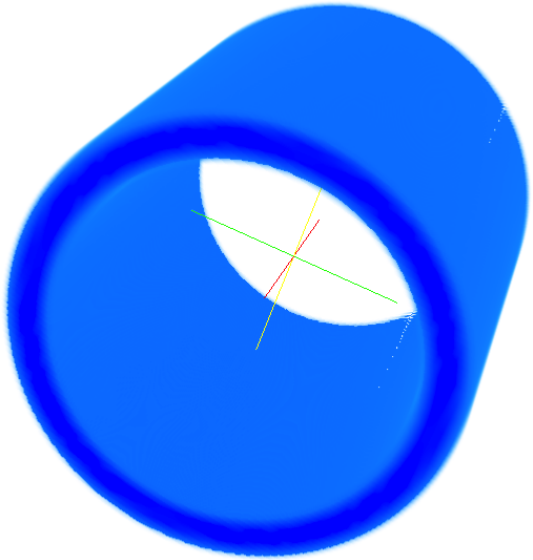
Exercise: visualizing 3D data with legacy VTK

- 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 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`

Another option for writing XML VTK from Python

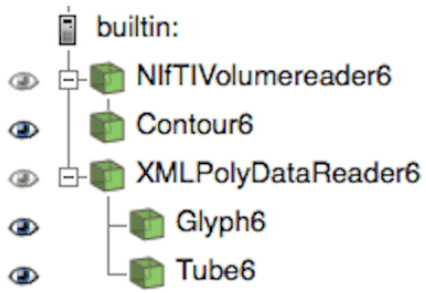
PyEVTK library `https://bitbucket.org/pauloh/pyevtk`

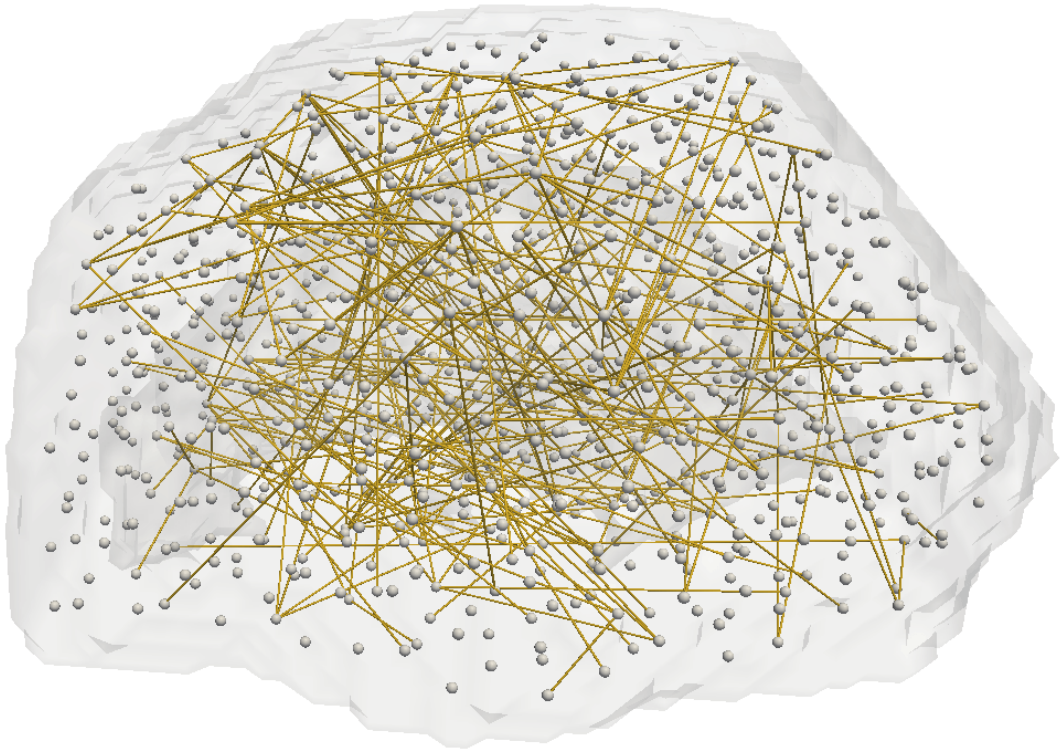
```
hg clone https://bitbucket.org/pauloh/pyevtk
cd pyevtk
python setup.py install --prefix=/Users/razoumov/miniconda
```

- Works in both Python 2 and Python 3
- Many examples in `src/examples/{image,points,rectilinear,structured,group,lowlevel}.py`

```
from evtk.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





```
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)
  - optionally 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, looked at:
  - Structured Points
  - Structured Grid
  - Polygonal Data
- VTK XML formats from C++ writing binary data with VTK libraries, looked at:
  - 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)

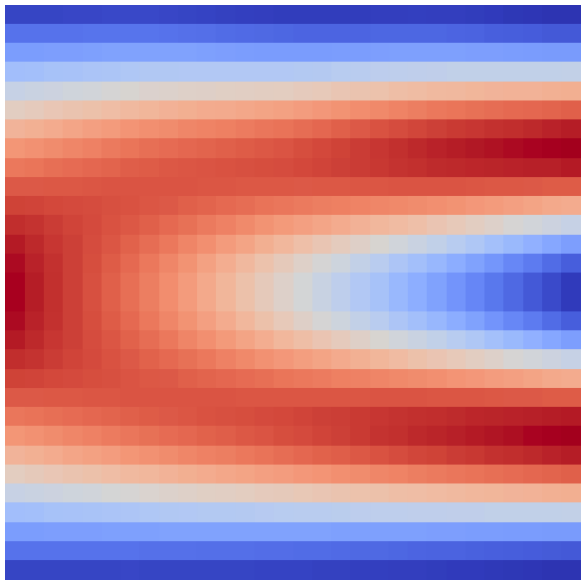
Filters are functional units that process the data to generate, extract, or derive additional features. The filter connections form a **visualization pipeline**

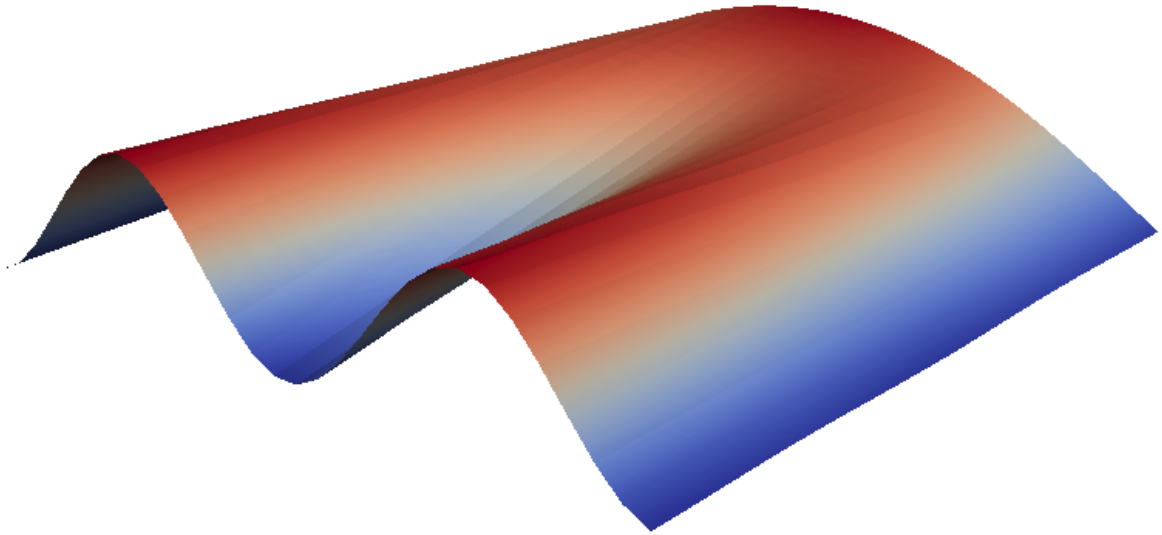
Last time I counted there were 146 filters. One can add new filters with python scripting

- 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, or 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.

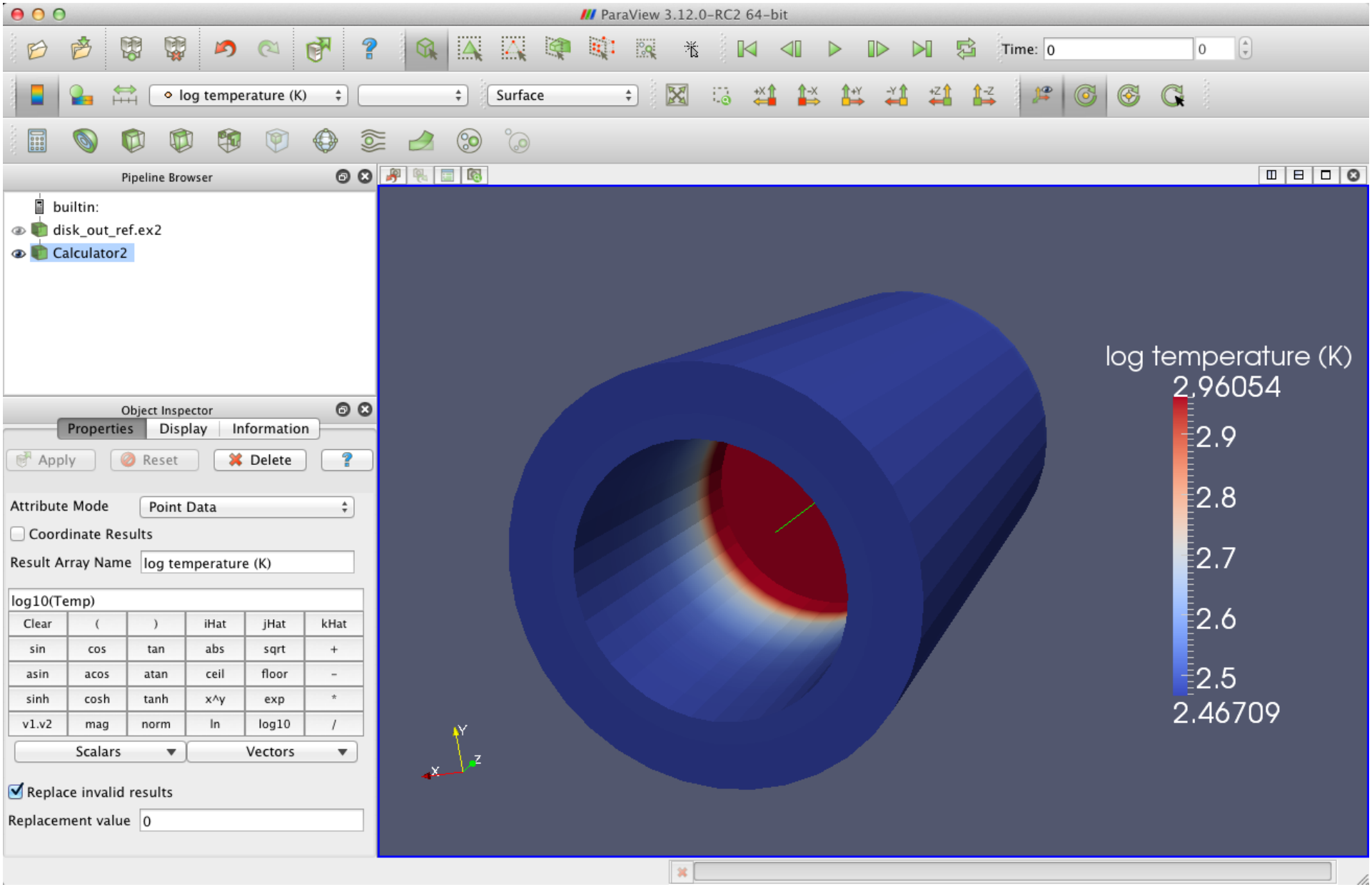
Calculator

- 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
Y
Z
X

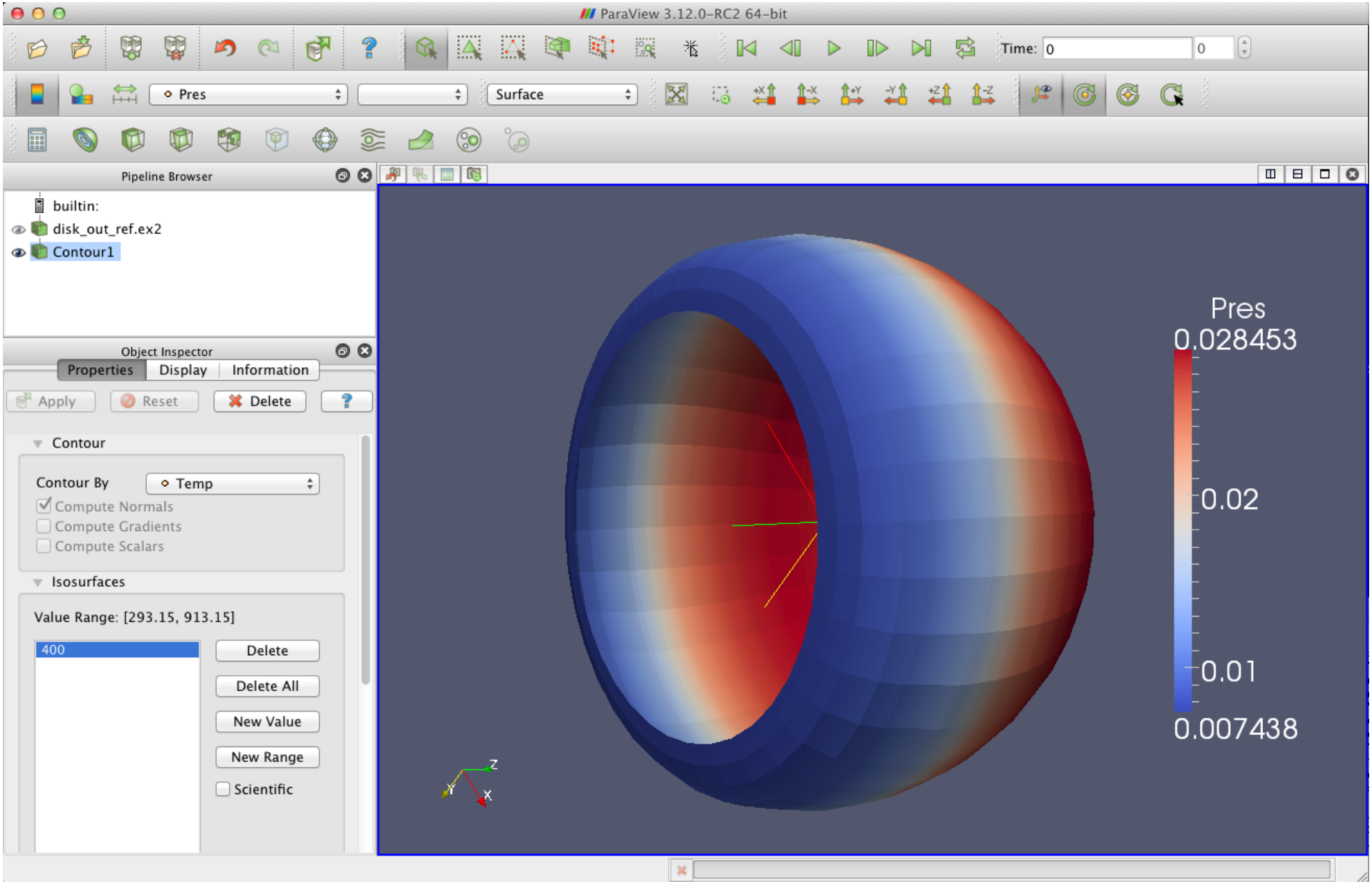
Contour

- 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

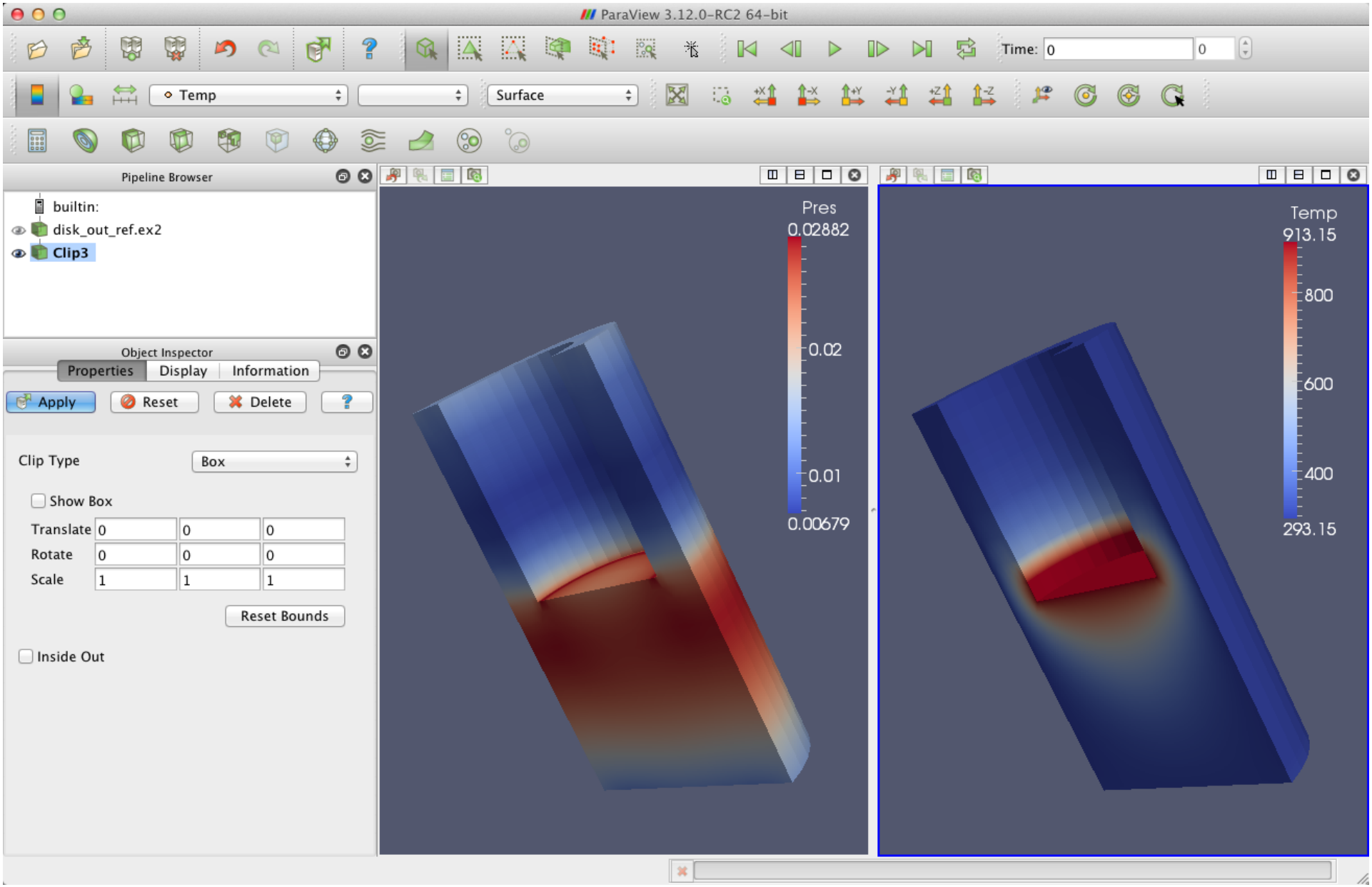
Multiview: several variables side by side

- 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

- Start with the original data from `data/disk_out_ref.ex2`, load velocity, Temp
- Add the **Stream Tracer** filter, set Radius = 10 (of sphere with tracer points), play with Number Of Points, Maximum Streamline Length
- Add shading and depth cues to streamlines: Filters → Alphabetical → **Tube** (could be also called Generate Tubes)
- 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
- Now try displaying "V" glyphs directly from data, can colour them using different variables ("Temp", "V")

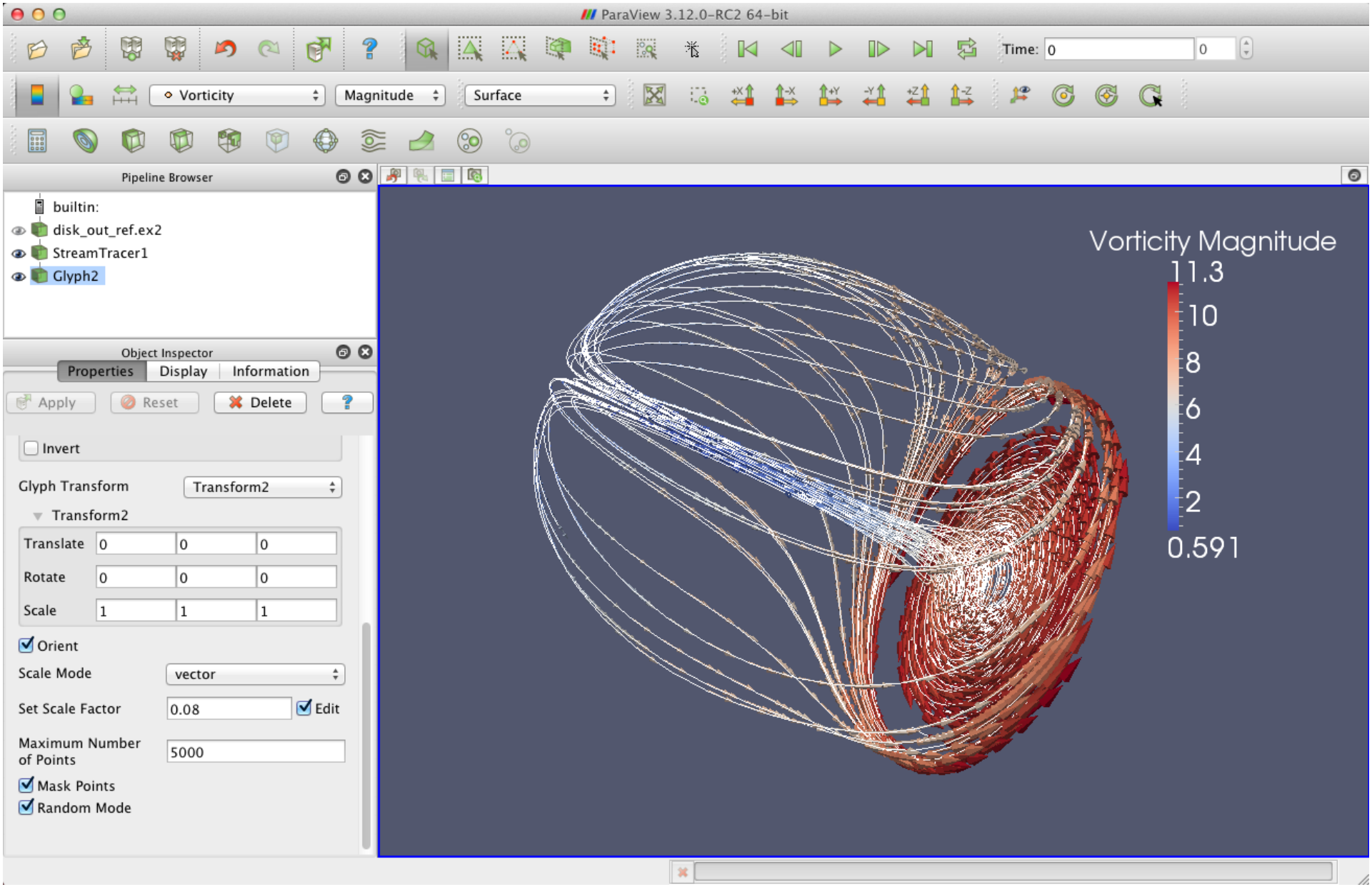
Vector visualization: streamlines and glyphs

- Start with the original data from `data/disk_out_ref.ex2`, load velocity, Temp
- Add the **Stream Tracer** filter, set Radius = 10 (of sphere with tracer points), play with Number Of Points, Maximum Streamline Length
- Add shading and depth cues to streamlines: Filters → Alphabetical → **Tube** (could be also called Generate Tubes)
- 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
- Now try displaying "V" glyphs directly from data, can colour them using different variables ("Temp", "V")

Vector visualization: streamlines and glyphs

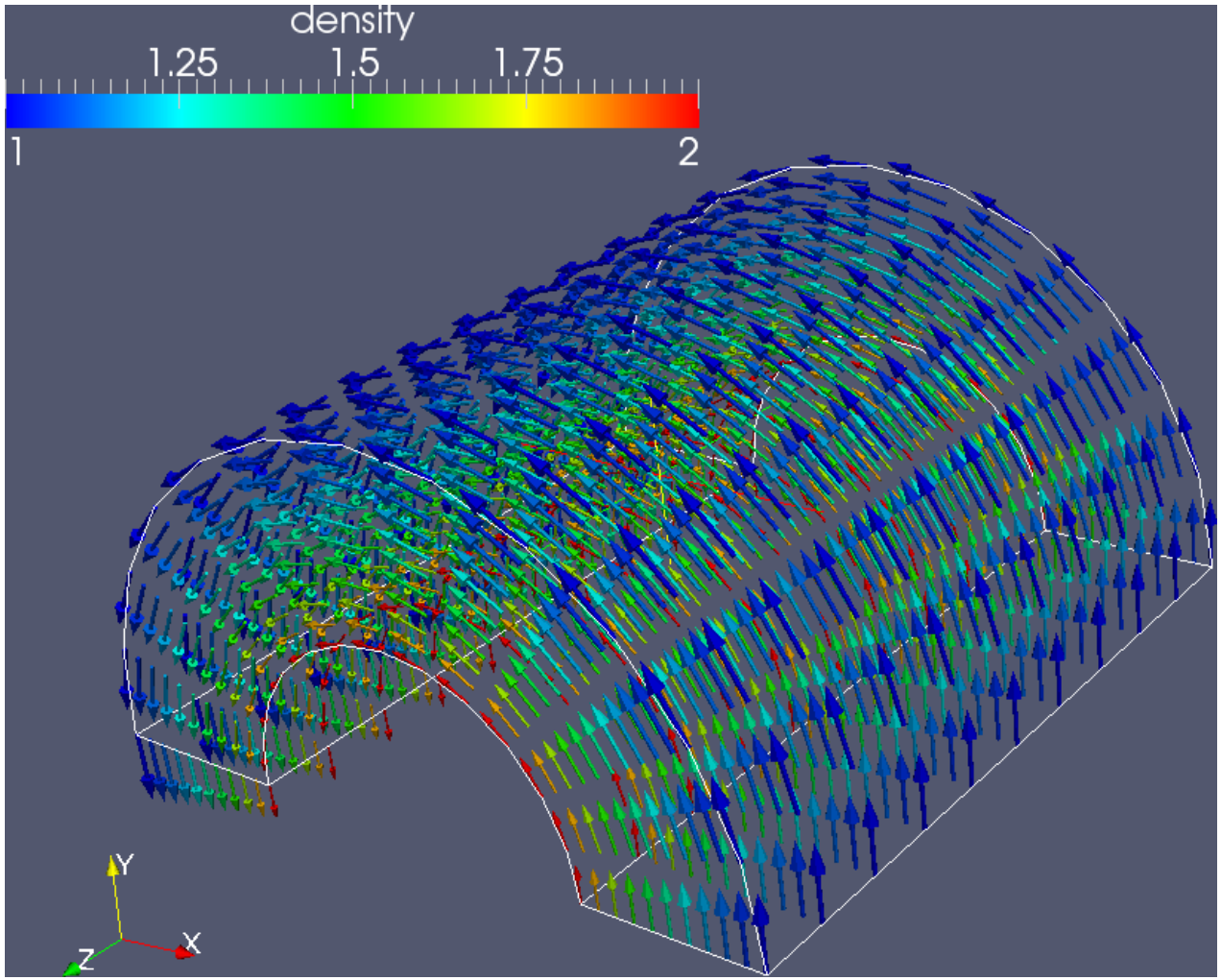
- Start with the original data from `data/disk_out_ref.ex2`, load velocity, Temp
- Add the **Stream Tracer** filter, set Radius = 10 (of sphere with tracer points), play with Number Of Points, Maximum Streamline Length
- Add shading and depth cues to streamlines: Filters → Alphabetical → **Tube** (could be also called Generate Tubes)
- 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
- Now try displaying "V" glyphs directly from data, can colour them using different variables ("Temp", "V")

Vector visualization: streamlines and glyphs

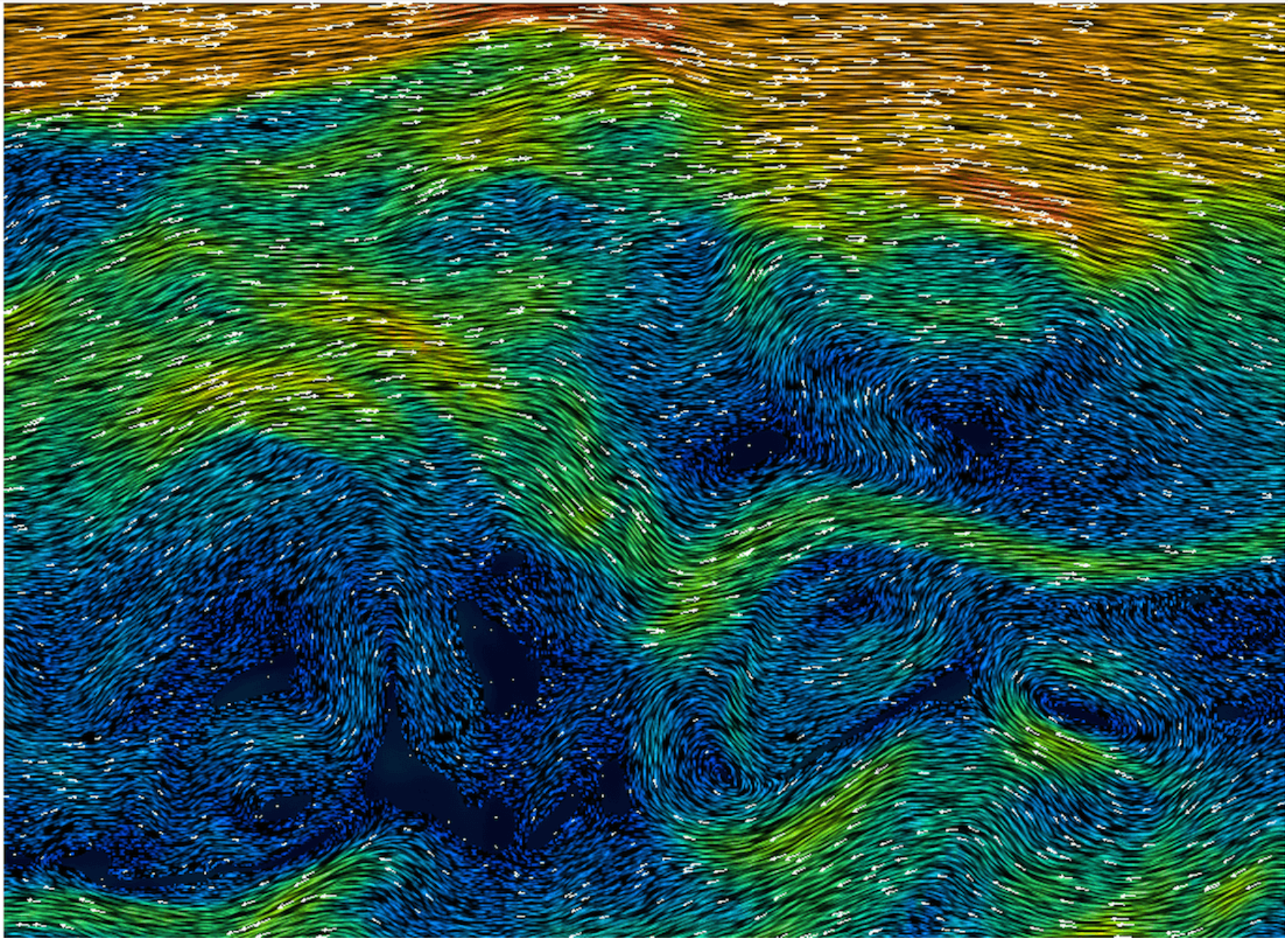


Exercise: vectors

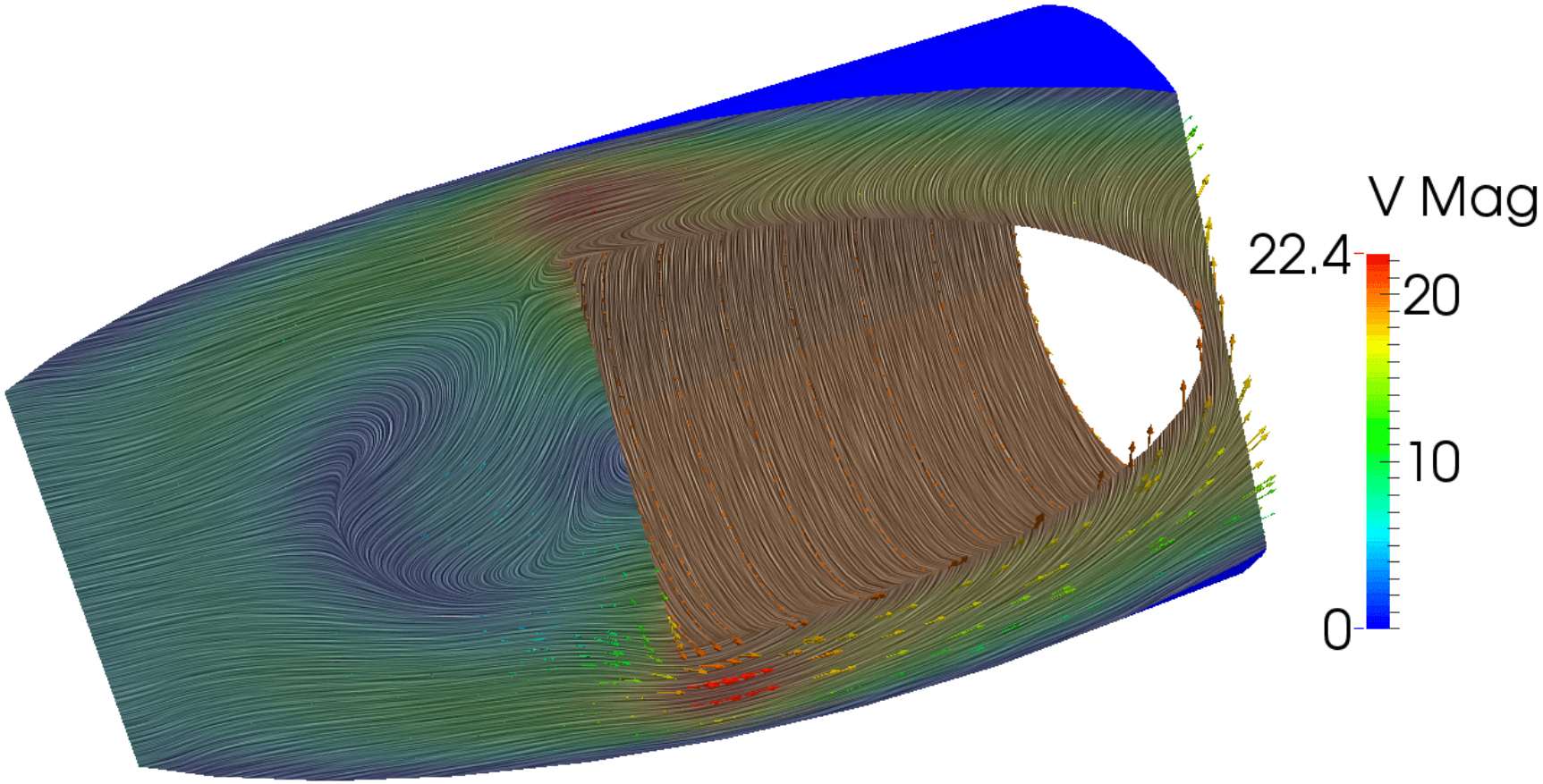
Load `data/halfCylinder.vts` and display the velocity field as arrows, colouring them by density – try to reproduce the view below



Visualizing vectors with 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
- Play with the number of steps and individual step sizes, adjust colour
- `http://www.paraview.org/Wiki/ParaView/Line_Integral_Convolution`

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 an implied topology for this filter to work!

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

➢ tabulatedPoints.txt is 6231 bytes on disk, in binary 1600/3200 bytes in single/double precision

➢ tabulatedGrid.txt is 20,013 bytes on disk, in binary 4000/8000 bytes in single/double precision

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`

3D optimization exercise

`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:

- What is the size of the grid? Does it agree with the size of the file?
- 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)

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 "▶"
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

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

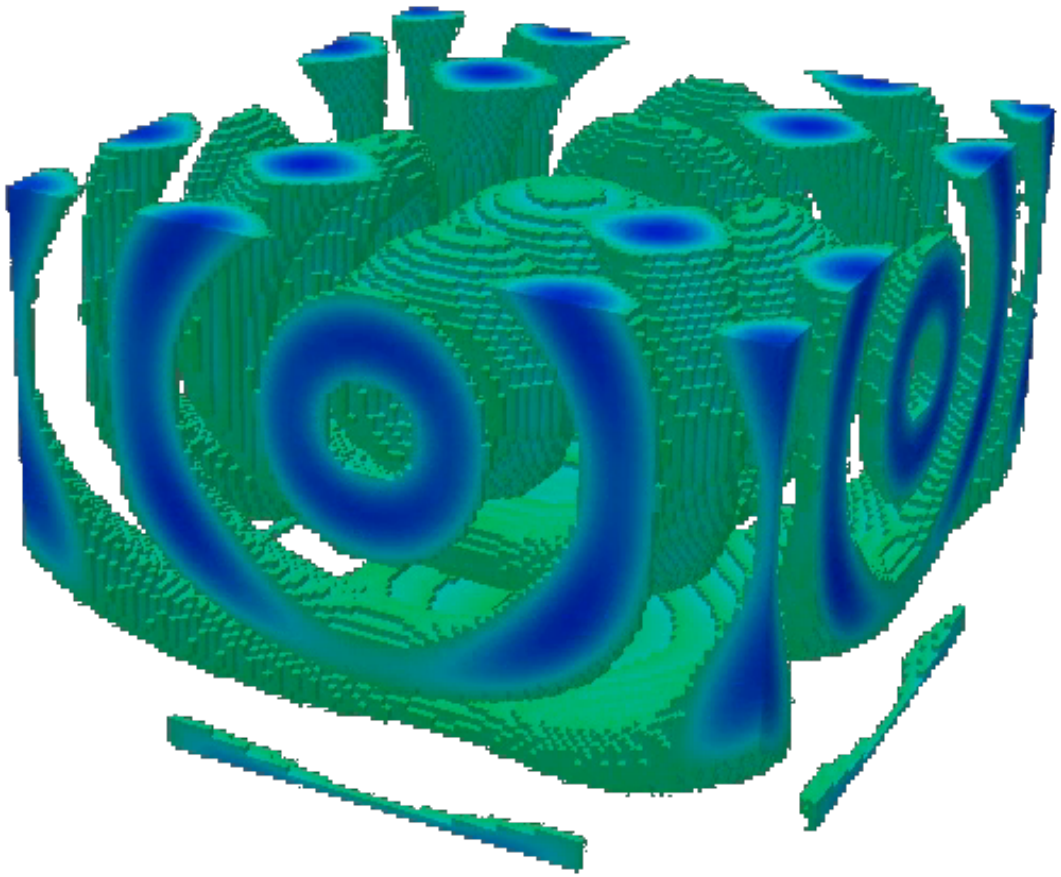
Animation methods

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 ($x_i \in [0, 1]$)

$$f(x_1, x_2, x_3) = \sum_{i=1}^{2} \left[\frac{\sin^2\left(\sqrt{\xi_{i+1}^2 + \xi_i^2}\right) - 0.5}{\left[0.001(\xi_{i+1}^2 + \xi_i^2) + 1\right]^2} + 0.5 \right], \text{ where } \xi_i \equiv 30(x_i - 0.5)$$



- 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

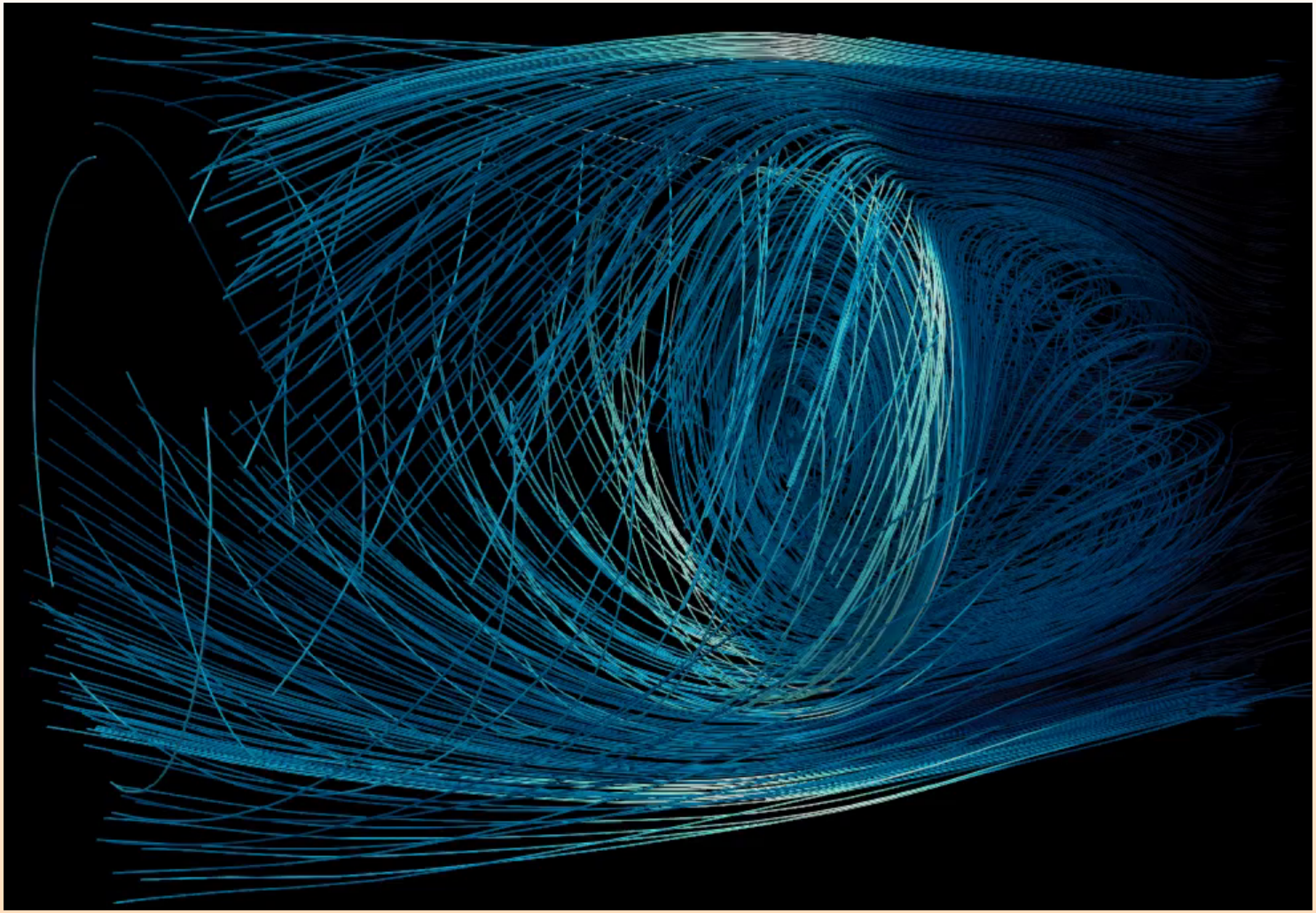
(3) Bring up Animation View (or erase all previous timelines)

(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 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

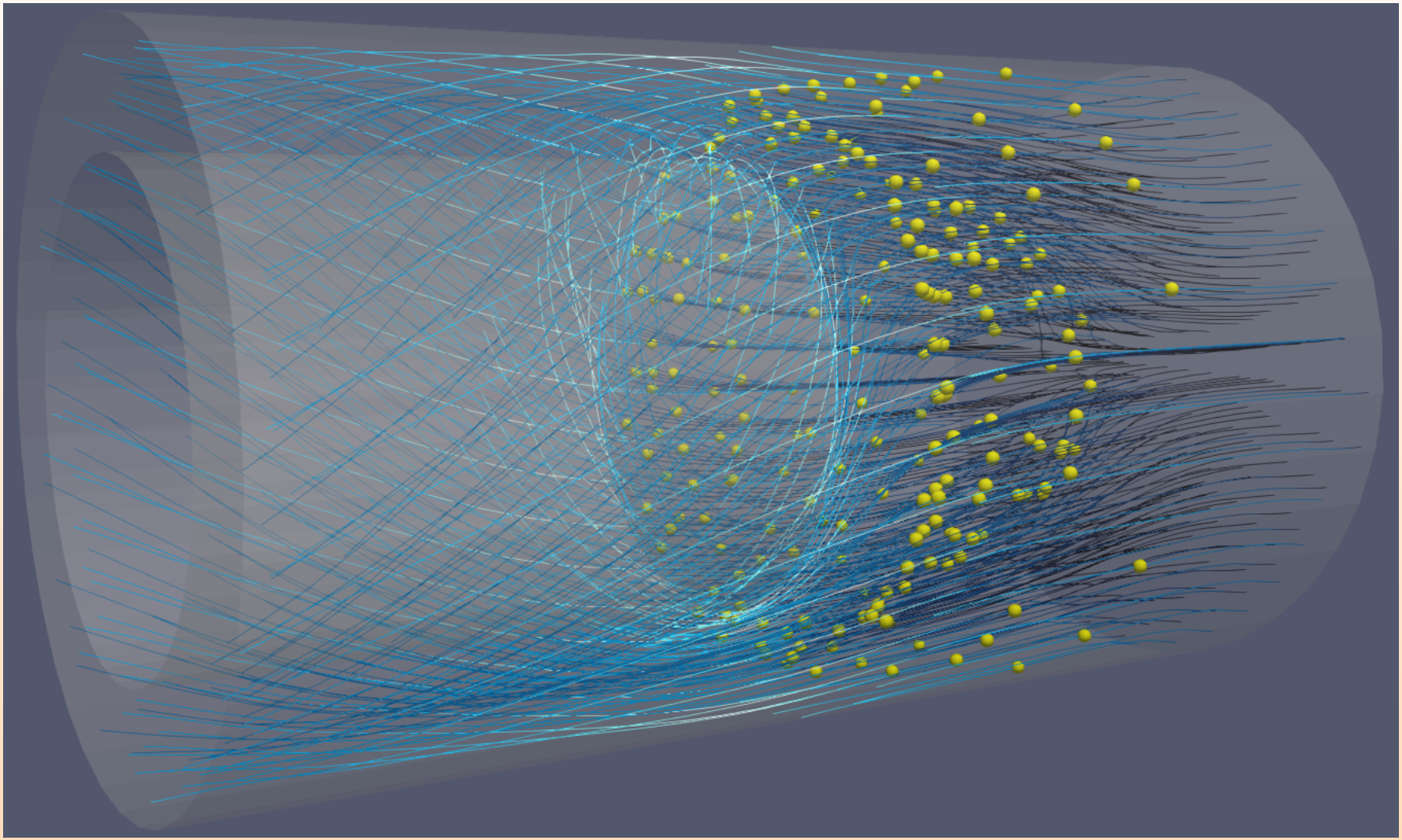


- `https://vimeo.com/248501893` or `hidden/radialSlice.mp4` on presenter's laptop
- `https://vimeo.com/248502086` or `hidden/xySlice.mp4` on presenter's laptop

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

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

Animating a stationary flow: time contours



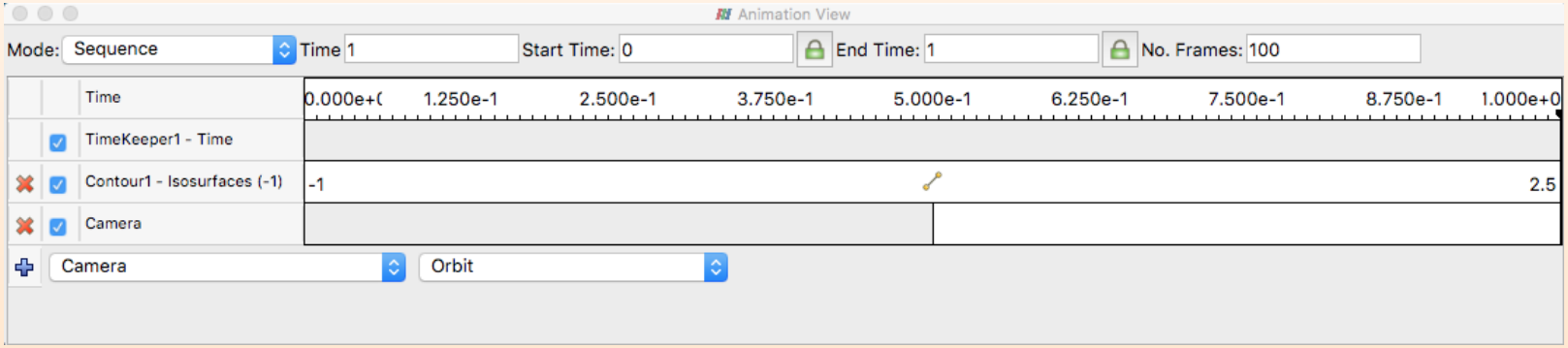
`https://vimeo.com/248509153` or `hidden/timeContours.mp4` on presenter's laptop

Animating a stationary flow: time contours (cont.)

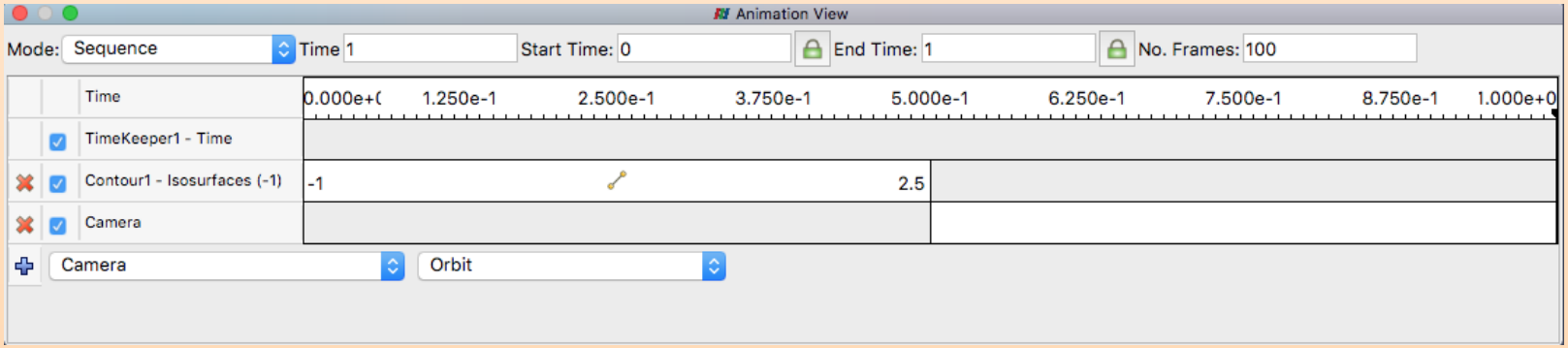
- Start with the streamtracer lines, however drawn
- Apply a Countour filter to the output of Streamtracer
  - contour by Integration Time
  - probe the range of values that works best
- Apply Glyph filter to the output of Countour
- Animation View: animate Contour - Isosurfaces
- This video was recorded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

Combining many timelines in one animation

- Start with the previous integration-time-contour animation
- 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)

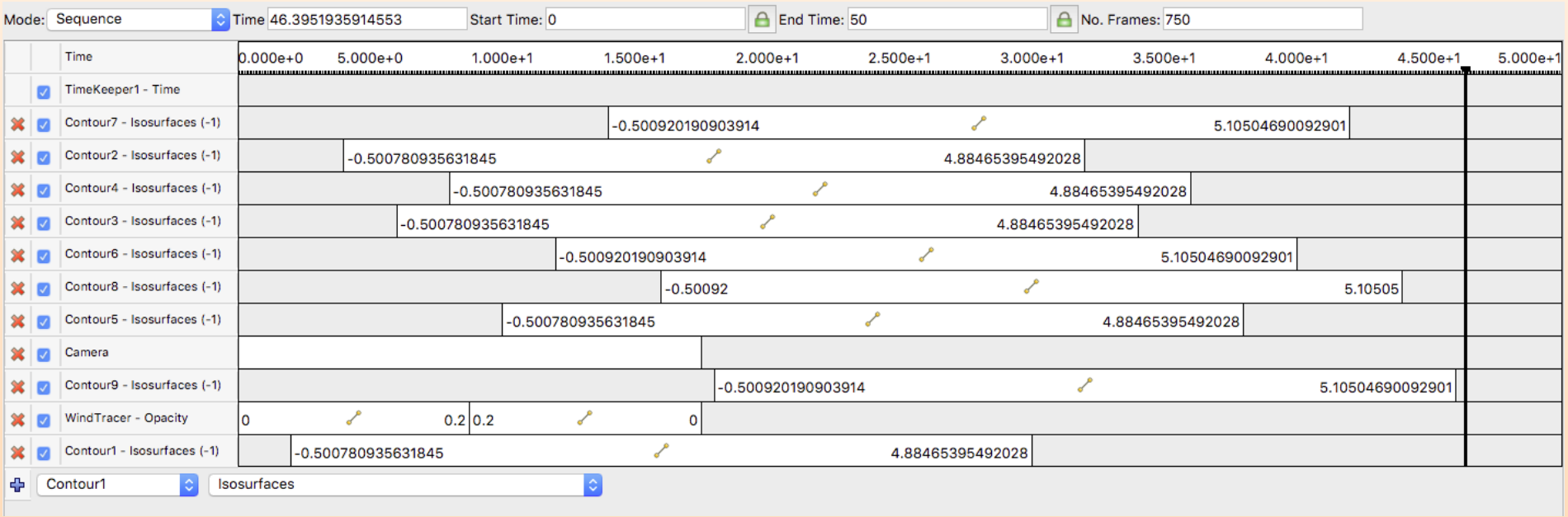


- Now complete integration-time-contour animation before rotation



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)



Python scripting in ParaView

Batch scripting for automating visualization

Official documentation at `http://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
- Tools → Python Shell opens a Python interpreter
- *[/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** *pythonScript.py* is a serial (on some machines parallel) application using local server ✐ make sure to save your visualization
- *[/usr/bin/ /usr/local/bin/ /Applications/Paraview.app/MacOS/]* *paraview --script=codes/displayWireframe.py* to start ParaView GUI and auto-run the script

First script

- Bring up Tools → 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`

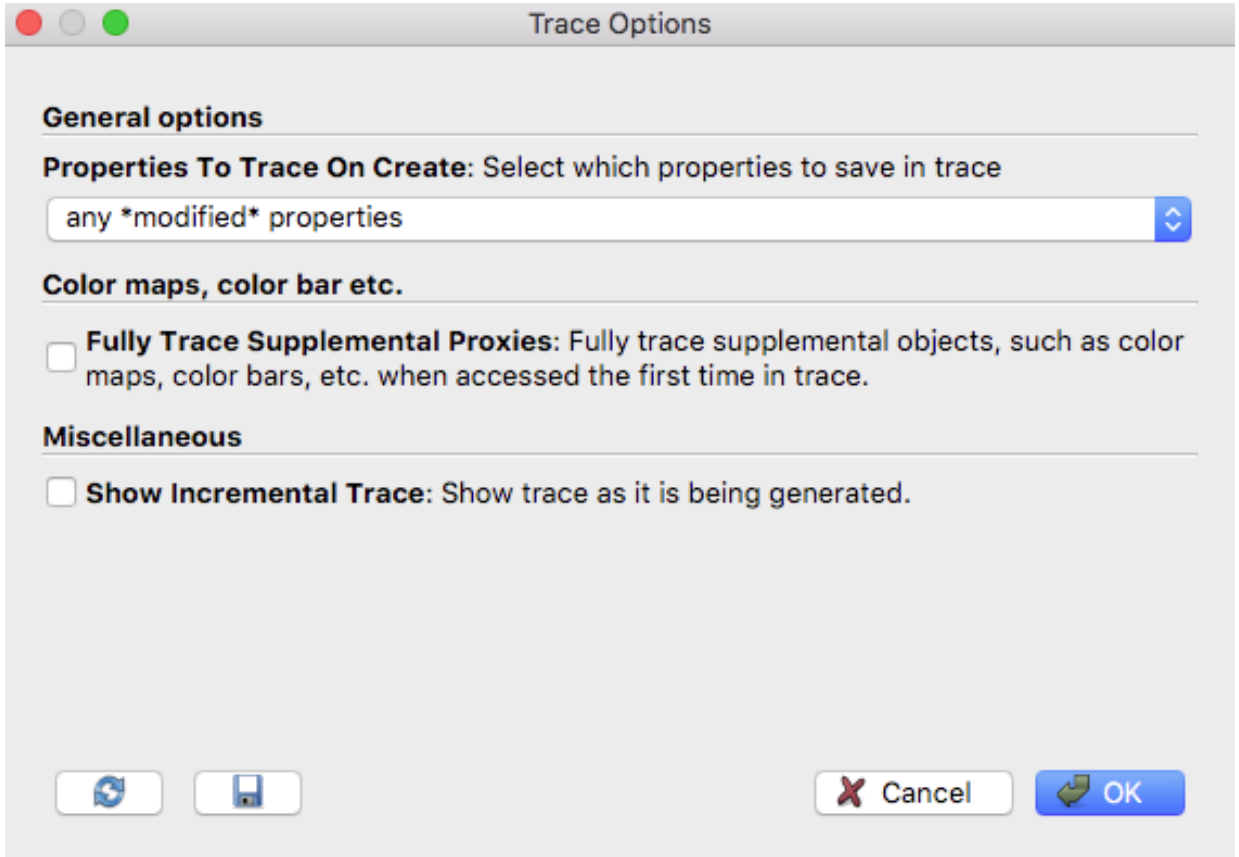
displayWireframe.py

```
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
- Start/Stop Trace at any time
- Older ParaView: Tools → Python Shell → Trace → Start | Stop | Show Trace
- Newer ParaView: Tools → Start | Stop 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

(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: write the script with standalone ParaView GUI

- load `data/sineEnvelope.nc` and draw an isosurface at $\rho = 0.15$
- use Start/Stop Trace
- save the image as PNG

(2) Mac/Linux: run it with `pvbatch` on your laptop

```
$ pvbatch pythonScript.py
```

Windows: if can't find `pvbatch` or don't know how to use it, run the script on Cedar

- `--use-offscreen-rendering` most likely won't work on your laptop, unless your ParaView was compiled with software Mesa rendering (should work on the cluster with the right module loaded)
- `ffmpeg` to merge frames into a movie (on your laptop or Cedar)

Extracting data from VTK objects

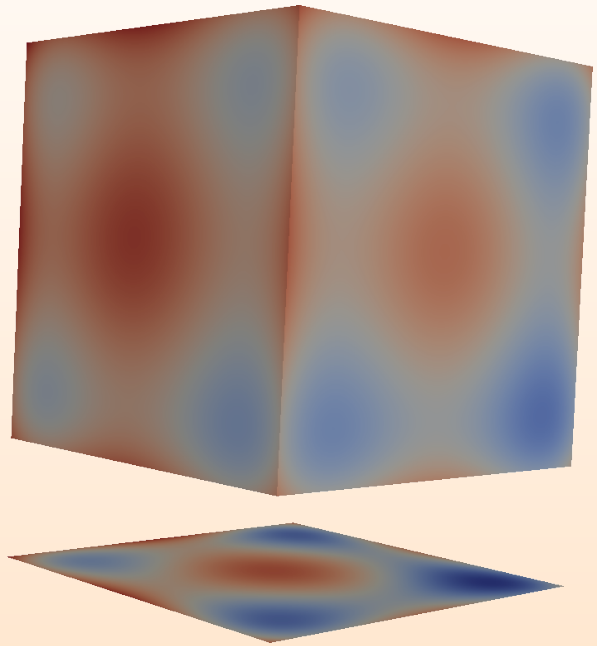
Do this from *Tools* → *Python Shell* or from *pvpython* (either shell will work)

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

dir = '/Users/razoumov/teaching/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) # print coordinates of first 10 points

pd = local.GetPointData()
print pd.GetArrayName(0) # print 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) # print 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

Modifying VTK objects

Let's say we want to plot a projection of the cubic dataset `data/stvol.nc` along one of its principal axes, or do some other transformation of the original dataset for which there is no filter.



- Calculator filter does not modify the geometry ...
- In ParaView's Python we can create a new VTK object from the existing array data (expanding on the previous slide), but there is no mechanism in ParaView to import it into the pipeline ...
  - could save the new VTK object to a file and then re-read it from ParaView, but that's slow – would like an in-memory solution ...

Programmable filter

1. Apply Programmable Filter with OutputDataSetType = vtkUnstructuredGrid
2. Copy and paste the following code into the filter

```
# codes/projectionFilter.py
input = self.GetInput(); output = self.GetOutput()
numPoints = input.GetNumberOfPoints(); numCells = input.GetNumberOfCells()
print numPoints, numCells
side = int(round(numPoints**(1./3.))); layer = side*side
pointData = input.GetPointData(); fref = pointData.GetArray('f(x,y,z)')
newPoints = vtk.vtkPoints()    # create 100x100 points forming the projection
proj = vtk.vtkDoubleArray()    # create the projection array
proj.SetName('projection')
for i in range(layer):
    x, y = input.GetPoint(i)[0:2]; z, pval = -30., 0.
    newPoints.InsertPoint(i,x,y,z)    # insert a point
    for j in range(side):
        pval += fref.GetValue(i+layer*j)
    proj.InsertNextValue(pval)    # add data to each point
output.SetPoints(newPoints); output.GetPointData().SetScalars(proj)
mesh = vtk.vtkCellArray()    # create 99x99 cells in the projection
for i in range(side-1):
    for j in range(side-1):
        mesh.InsertNextCell(4)    # insert a cell with four corners
        mesh.InsertCellPoint(i+j*side); mesh.InsertCellPoint(i+1+j*side)
        mesh.InsertCellPoint(i+side+j*side); mesh.InsertCellPoint(i+side+1+j*side)
output.SetCells(10, mesh)    # 10 refers to a VTK cell type
```

Remote and distributed visualization with ParaView

Visualizing remote data

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

(1) download data to your desktop and visualize it locally
limited by dataset size and your desktop's CPU+GPU+memory

(2) ✘ run ParaView remotely on a larger machine via X11 forwarding
your desktop $\xrightarrow{\text{ssh } -X}$ larger machine running ParaView

(3) run ParaView remotely on a larger machine **via VNC or x2go**
your desktop $\xrightarrow{\text{VNC}}$ larger machine running ParaView

- additional login nodes with secure VNC will be added to Cedar/Graham; not top priority with the system's teams

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

(5) ✔ 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 it's best to schedule interactive jobs, as opposed to running on the login nodes

Special case: in-situ visualization

In-situ visualization = instrumenting a simulation code on the cluster to

(1) output graphics and/or

(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

Cedar and Graham clusters

- General-purpose clusters for a variety of workloads
- Entered production in June 2017
- Respectively:
  - located at SFU and UofWaterloo
  - 27,696 (Broadwell) + 30,720 (Skylake) and 32,136 CPUs
  - 584 and 320 NVIDIA P100 Pascal GPUs (12GB/16GB on-board memory)
  - specs at `https://docs.computecanada.ca/wiki/Cedar` and `https://docs.computecanada.ca/wiki/Graham`
- Multiple types of nodes, with 128GB/256GB/0.5TB/1.5TB/3TB memory
- Batch-oriented environment for parallel and serial jobs, use Slurm scheduler and workload manager
- Identical software setup `https://docs.computecanada.ca/wiki/Available_software`

Interactive jobs on Cedar/Graham

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

```
$ sinfo | grep interac
    # will list nodes and their states (idle, mixed, allocated, ...)
```

unless you have a reservation (we do for this workshop)

- `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-razoumov-ws_cpu --reservation=arazoumov-may17
$ echo $SLURM_...   # access Slurm variables, or set your environment
$ ./serial
$ srun ./mpi    # run an MPI code
$ exit          # terminate the job (go back to the login node)
```

CPU vs. GPU 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
- Key question: **how many processors/GPUs do I need?**
  Answer: In addition to *rendering time*, your other bottlenecks will be *physical memory* and *disk read speed* $\Rightarrow$ to simplify things, for initial dataset exploration you may want to use memory as your point of reference
  - 128GB base node with 32 cores *minus the OS and utilities* $\Rightarrow$ **$\sim$3.5 GB/core is a good starting point** for estimating the number of cores for your visualization, based on the dataset size from a single time step
  - could ask for more memory/core, but don't want to starve other users of memory!

CPU vs. GPU rendering (cont.)

- In this workshop we'll be using Joshua's OpenFOAM data from `cedar:/home/razoumov/data/joshua`
  - turbulent flow over an airfoil, two timesteps
  - decomposed into 512 processors
  - 83 GB total $\Rightarrow$ 24 cores, but also need to account for MPI buffers, filters, other data processing, possibly structured to unstructured conversion ($\sim$3X memory footprint) $\Rightarrow$ could barely work with 32 cores, should be comfortable running on 64 cores
  - on presenter's laptop show `hidden/spinZoom.mp4`
  - let's take a look at the data from the command line!
- Smaller dataset ✎ `cedar:/home/razoumov/data/challenge/air.vtu` (419 MB)
  Tiny dataset ✎ `cedar:/home/razoumov/data/sineEnvelope.nc` (3.9 MB)
- Normally need to process 100s of such timesteps (should not change our memory requirements!) - the importance of garbage collection
- One might have to resort to software rendering if no GPUs are available, e.g., all are taken by GP-GPU jobs
- We'll start with a **quick look at GPU rendering** on the cluster, but we'll do **all hands-on work with CPU rendering**

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/5.4.0` 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 **submit an interactive job** to the GPU partition, e.g., a serial job:

```
$ salloc --time=0:30:0 --ntasks=1 --gres=gpu:1 \
         --mem-per-cpu=4000 --account=your-ccdb-role
```

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 paraview-offscreen-gpu/5.4.0
$ unset DISPLAY   # so that PV does not attempt to use X11 rendering context
$ pvserver   # --egl-device-index=0 not needed: first 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 5.4.x** and **edit its connection properties** under *File - Connect - Add Server* (name = Cedar, server type = Client/Server, host = localhost, port = 11111), click *Configure* $\rightarrow$ *Manual* $\rightarrow$ *Save*, then select the server from the list and click on *Connect*

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

- ParaView's client and server must have matching major versions (5.4.x)
- Occasionally during client-server connection might get an error *"Only EGL 1.4 and greater allows OpenGL as client API"*
  - the GPU is stuck in a strange state $\Rightarrow$ need to reboot the node (let us know!)
- In ParaView's preferences can set Render View -> Remote/Parallel Rendering Options -> Remote Render Threshold (beyond which rendering will be remote)
  - default 20MB $\Rightarrow$ 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 $\Rightarrow$ all rendering (including rotation) will be remote, so you will be really using the cluster's GPU for everything
    - ★ good for large data processing
    - ★ not so good for interactivity
  - experiment with the threshold to find a suitable value

Parallel client-server software rendering

Details in `http://bit.ly/2x5zIB6`

(1) On Cedar or Graham **submit a parallel interactive job**:

```
$ salloc --time=1:00:0 --ntasks=64 --mem-per-cpu=3500 \
         --account=def-razoumov-ws_cpu --reservation=arazoumov-may17
```

- you could also try `--ntasks=32 --mem-per-cpu=7000`
- normally you would use `--account=def-your-ccdb-role`
- when the job starts, it'll return a prompt on the assigned compute node

(2) On the compute node inside the job **start a parallel ParaView server**

```
$ module load paraview-offscreen/5.3.0
$ srun pvserver --use-offscreen-rendering   # --mesa-swr-avx2 no effect in parallel
```

The `pvserver` command will return something like

```
Waiting for client...
Connection URL: cs://cdr544.int.cedar.computecanada.ca:11111
Accepting connection(s): cdr544.int.cedar.computecanada.ca:11111
```

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

```
$ ssh username@cedar.computecanada.ca -L 11111:cdr544:11111
```

(4) On your desktop **start ParaView 5.3.x** and **edit its connection properties** under *File - Connect - Add Server* (name = Cedar, server type = Client/Server, host = localhost, port = 11111), click *Configure* $\rightarrow$ *Manual* $\rightarrow$ *Save*, then select the server from the list and click on *Connect*

Parallel client-server software rendering (cont.)

(5) Once connected, load the file from
`cedar:/home/razoumov/data/joshua/case.foam`
with Case Type = Decomposed (takes ∼1 minute to load)
- make sure it's loaded properly: 174,080,000 cells, 186,876,522 points
- check memory usage View - Memory Inspector, the client should use around 270 MB, the cluster ∼ 2.3 GB/core for 64-core run

(6) Switch to surface view, pass your dataset through Filters - ProcessIdScalars, colour by ProcessId to check that rendering is taking place in parallel

(7) **Today's goal**: animate OpenFOAM data with one of the techniques below using both (1) Animation View (*client-server GUI*) and (2) off-screen batch jobs (see next few slides on running *remote batch scripts*)
- rotation around an axis
- zooming in to see turbulence near the airfoil
- moving integration time contours along streamlines
- streamlines through a moving slice
- time evolution (if we have data from many timesteps)

Off-screen batch rendering with scripts

Why?

- Very large renderings cannot be done interactively (they take too long!)
  - ⇒ submit a batch rendering job and come back to a nice visualization in a few hours or the next day
    - can be performed on any combination of GPUs or CPUs, but details vary
- Automate mundane or repetitive tasks, e.g., making mutiple frames of a movie

Make sure to:

- Debug the script either in client-server or standalone on your laptop
- Before multi-hour batch renderings, test the script on the cluster
  - file paths
  - cluster's ParaView version (its Python library is evolving quickly!)
  - support for third-party libraries, formats, macros, etc.

Software batch rendering: off-screen on CPU nodes

(1) On Cedar or Graham start with an interactive job

```
$ module load paraview-offscreen/5.3.0
$ cd /path/to/yourWorkingDirectory
$ salloc --time=1:00:0 --ntasks=64 --mem-per-cpu=3500 \
         --account=def-razoumov-ws_cpu --reservation=arazoumov-may17
$ pvbatch --use-offscreen-rendering serialRendering.py
$ srun pvbatch --use-offscreen-rendering parallelRendering.py
$ <possibly do another rendering>
$ exit   # terminate the interactive job
```

- It would read data from disk, do rendering, and write the resulting image to disk – all without opening any windows
- "Wait, I don't have the Python rendering script ..."
  - don't panic! the goal is to slowly write your own
  - first, try to run the short script for `data/sineEnvelope.nc` you ran earlier on your laptop; then switch to processing OpenFOAM data
  - as the last resort, you could skip forward three slides to see `codes/spheres.py`

Software batch rendering (cont.)

(2) Here is how you would run it as a serial batch (non-interactive) job

- the OpenFOAM dataset is too large to process in serial, but you can use this for smaller `sineEnvelope.nc`

```
#!/bin/bash
#SBATCH --time=00:15:00    # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --job-name="rendering"
#SBATCH --mem=2000     # in MB
#SBATCH --account=def-razoumov-ws_cpu          # normally --account=your-ccdb-role
#SBATCH --reservation=arazoumov-may17          # normally no reservation
pvbatch --use-offscreen-rendering serialRendering.py
```

```
$ module load paraview-offscreen/5.3.0
$ cd /path/to/yourWorkingDirectory
$ sbatch s1.sh
$ squeue -u yourUserName
$ squeue -u yourUserName --start      # estimate start time
```

Software batch rendering (cont.)

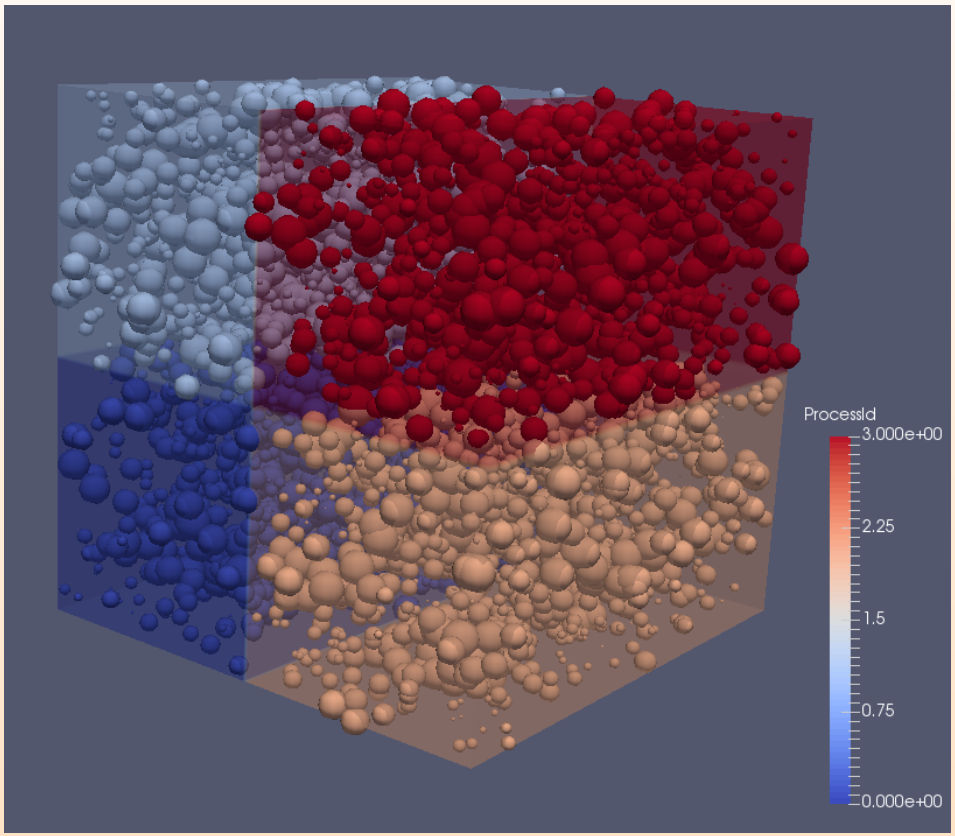
(3) And as a parallel batch job

- perfect for the OpenFOAM dataset!

```
#!/bin/bash
#SBATCH --ntasks=4    # number of MPI processes
#SBATCH --time=0-00:05    # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --job-name="rendering"
#SBATCH --mem-per-cpu=2000    # in MB
#SBATCH --account=def-razoumov-ws_cpu          # normally --account=your-ccdb-role
#SBATCH --reservation=arazoumov-may17          # normally no reservation
srun pvbatch --use-offscreen-rendering parallelRendering.py
```

```
$ module load paraview-offscreen/5.3.0
$ cd /path/to/yourWorkingDirectory
$ sbatch p1.sh
$ squeue -u yourUserName
$ squeue -u yourUserName --start      # estimate start time
```

Example: testing parallel rendering on the cluster



The script `codes/spheres.py`

(1) reads `data/sineEnvelope.nc`,

(2) uses ProcessIdScalars filter to assign a unique scalar (0, 1, 2, ...) to each subdomain of the dataset according to which processor it resides on,

(3) colours the corresponding subdomain with a unique colour and renders it as a semi-transparent volume,

(4) drops 5000 random spheres and scales them by the value of the function at that location,

(5) colours these spheres by the colour of its host subdomain

Copy `data/sineEnvelope.nc`, `codes/spheres.py`, `codes/p1.sh` to the cluster, modify the scripts to render a similar image on 4 cores. How about 8 cores?

GPU batch rendering on Cedar/Graham

Running a serial GPU job

```
#!/bin/bash
#SBATCH --gres=gpu:1          # GPUs per node
#SBATCH --mem=2000M           # memory per node
#SBATCH --time=0-05:00        # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --account=your-ccdb-role
unset DISPLAY
pvbatch batch.py
```

```
$ module load paraview-offscreen-gpu/5.4.0
$ cd /path/to/yourWorkingDirectory
$ sbatch gpu.sh
$ squeue -u yourUserName
$ squeue -u yourUserName --start      # estimate start time
```

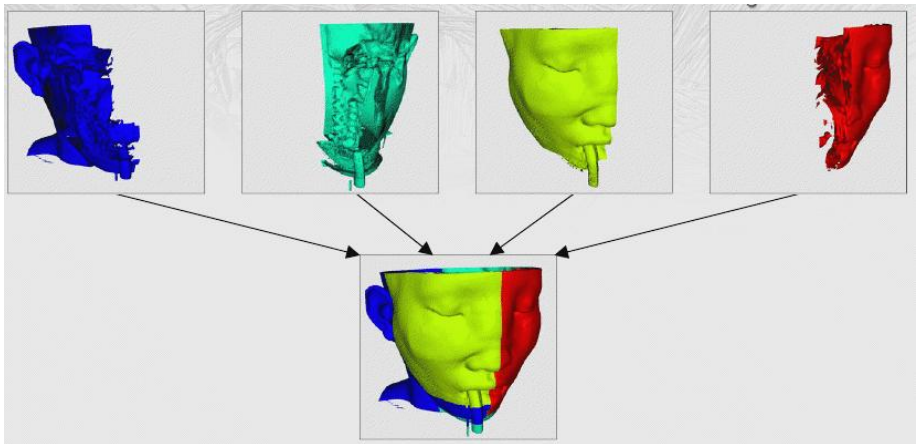
Data partitioning

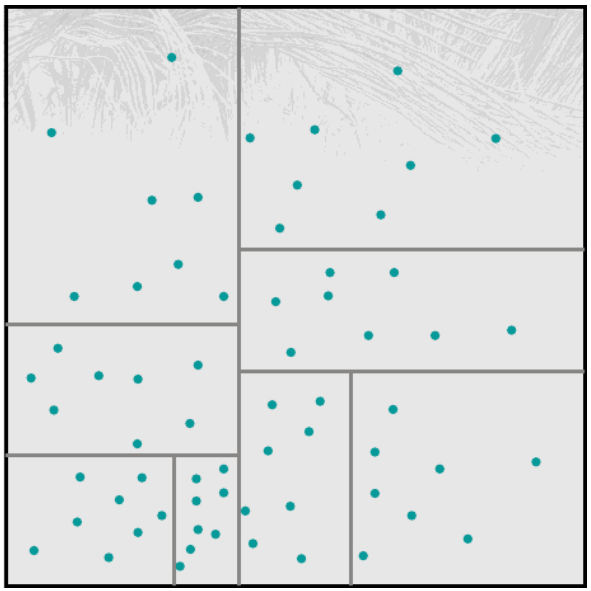
Scalable parallel distributed rendering – load balancing is handled automatically by ParaView for structured data:

- Structured Points
- Rectilinear Grid
- Structured Grid

Unstructured data must be passed through D3 (Distributed Data Decomposition) filter for better load balancing:

- Particles/Unstructured Points
- Polygonal Data
- Unstructured Grid





Best strategies for large datasets

- How many processors do we need? From *ParaView documentation*:
  - structured data (Structured Points, Rectilinear Grid, Structured Grid): one processor core per ∼20 million cells
  - unstructured data (Unstructured Points, Polygonal Data, Unstructured Grid): one processor core per ∼1 million cells
- In practice, memory and I/O speeds are the main bottlenecks (unless you do heavy processing)
  - consider 80 GB dataset
  - base nodes have 128 GB memory with 32 cores $\Rightarrow$ 4 GB/core (not to starve other cores of memory) $\Rightarrow$ 20 cores
  - also need to account for filters (and other processing), MPI buffers, the OS $\Rightarrow$ minimum 32 cores
  - I would even suggest 48 - 64 cores to be on the safe side
- On large HPC systems ParaView is known to scale to $\sim 10^{12}$ cells (Structured Points) on ∼10,000 cores
- 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 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 size of the dataset (GBs) to figure out the best number of processors, and adjust from there

Summary

Further resources

- Extended ParaView tutorial and sample data in many formats `http://www.cmake.org/Wiki/The_ParaView_Tutorial`
- 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/etc. code examples `http://www.itk.org/Wiki/VTK/Examples`
- VTK file formats (3rd-party intro) `http://www.earthmodels.org/software/vtk-and-paraview/vtk-file-formats`

Online WestGrid visualization webinars

- Bimonthly during the academic year (January, March, May, September, November), advertised at `https://www.westgrid.ca`
- Up to 60 mins, usually very specific topics
- Past webinars are available with slides and video at `https://www.westgrid.ca/events/archive`
  - "Introduction to batch visualization"
  - "Graph visualization with Gephi"
  - "3D graphs with NetworkX, VTK, and ParaView"
  - "CPU-based rendering with OSPRay"
  - "Scripting and other advanced topics in VisIt visualization"
  - "Visualization support in WestGrid / Compute Canada"
  - "Using ParaViewWeb for 3D visualization and data analysis in a web browser"
  - "3D visualization on new Compute Canada systems"
  - "Camera animation in ParaView and VisIt"
  - "Novel visualization techniques from the **Visualize This** competition"
  - "Scientific visualization with Plotly"
- We are always looking for topic suggestions!

Documentation and getting help

- Official documentation
  `https://docs.computecanada.ca/wiki/Visualization`
- WestGrid training materials
  `https://westgrid.github.io/trainingMaterials`
- Email **support@computecanada.ca** (goes to the ticketing system)
- Email **viz-support@computecanada.ca** (goes to CC::Visualization queue)
- Email me **alex.razoumov@westgrid.ca**
- Compute Canada visualization showcase `http://bit.ly/cctopviz`
- ParaView documentation
  - official documentation `http://www.paraview.org/documentation`
  - wikis `http://www.paraview.org/Wiki/ParaView`
  - Python batch scripting `http://bit.ly/2wF5v0B`
  - VTK tutorials `http://www.itk.org/Wiki/VTK/Tutorials`