Scientific visualization

HPC Summer School, June 2014

Alex Razoumov
razoumov@sharcnet.ca

SHARCNET/UOIT

- copy of these slides in `http://razoumov.sharcnet.ca/paraview.pdf`
- data and sample C++, Fortran, Python codes in `http://razoumov.sharcnet.ca/visualization.tar.gz` (two directories inside: `code/` and `data/`)
- ensure that you have ParaView installed on your laptop `http://www.paraview.org` (click on Download and select your version)

Outline for the day

Morning 9:30am-12:30pm with a short break at 11am

- Introduction to scientific visualization: general ideas, tools, plotting vs. multi-dimensional visualization
- Overview of current general-purpose multi-dimensional vis. tools
- ParaView: architecture, visualization pipelines
- Importing data into ParaView
- Basic workflows: filters, creating a pipeline, multiview, vectors

Afternoon 1:30pm-4:30pm with a short break at 3pm

- Scripting: batch GUI-less mode, few simple scripts, trace tool
- Animation: three approaches, one big exercise on scripting/animation
- Dealing with large datasets
- Briefly on composite datasets
- Putting it all together: final exercise

Scientific visualization

- Visualization is the process of mapping scientific data into VISUAL FORM
- Much easier to understand images than a large set of numbers
- For interactive data exploration, debugging, communication with peers

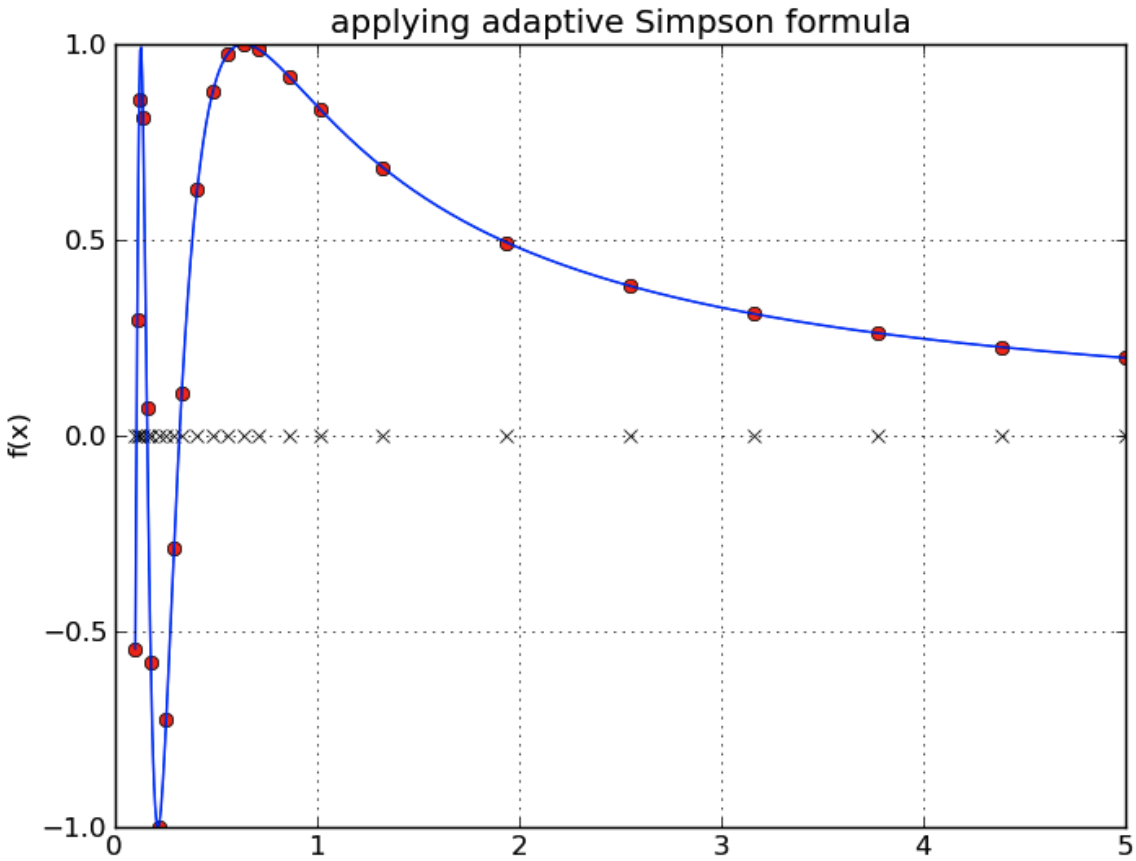
FIELD	VISUALIZATION TYPE
computational fluid dynamics	2D/3D flows, density, temperature, tracers
climate, meteorology, oceanography	fluid dynamics, clouds, chemistry, etc.
quantum chemistry	wave functions
molecular dynamics (phys, chem, bio)	particle/molecular data
bio-informatics	networks, trees, sequences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

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
- **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 $$
  - limitations on where you can run them, which machines/platforms, etc.
  - cannot get help from open-source community, user base usually smaller than for open-source tools
  - once you start accumulating scripts, you lock yourself into using these tools forever, and consequently paying $$ on a regular basis
  - there is nothing you cannot do with open-source tools

Matplotlib example: 1D plotting

Adaptive Simpson integration



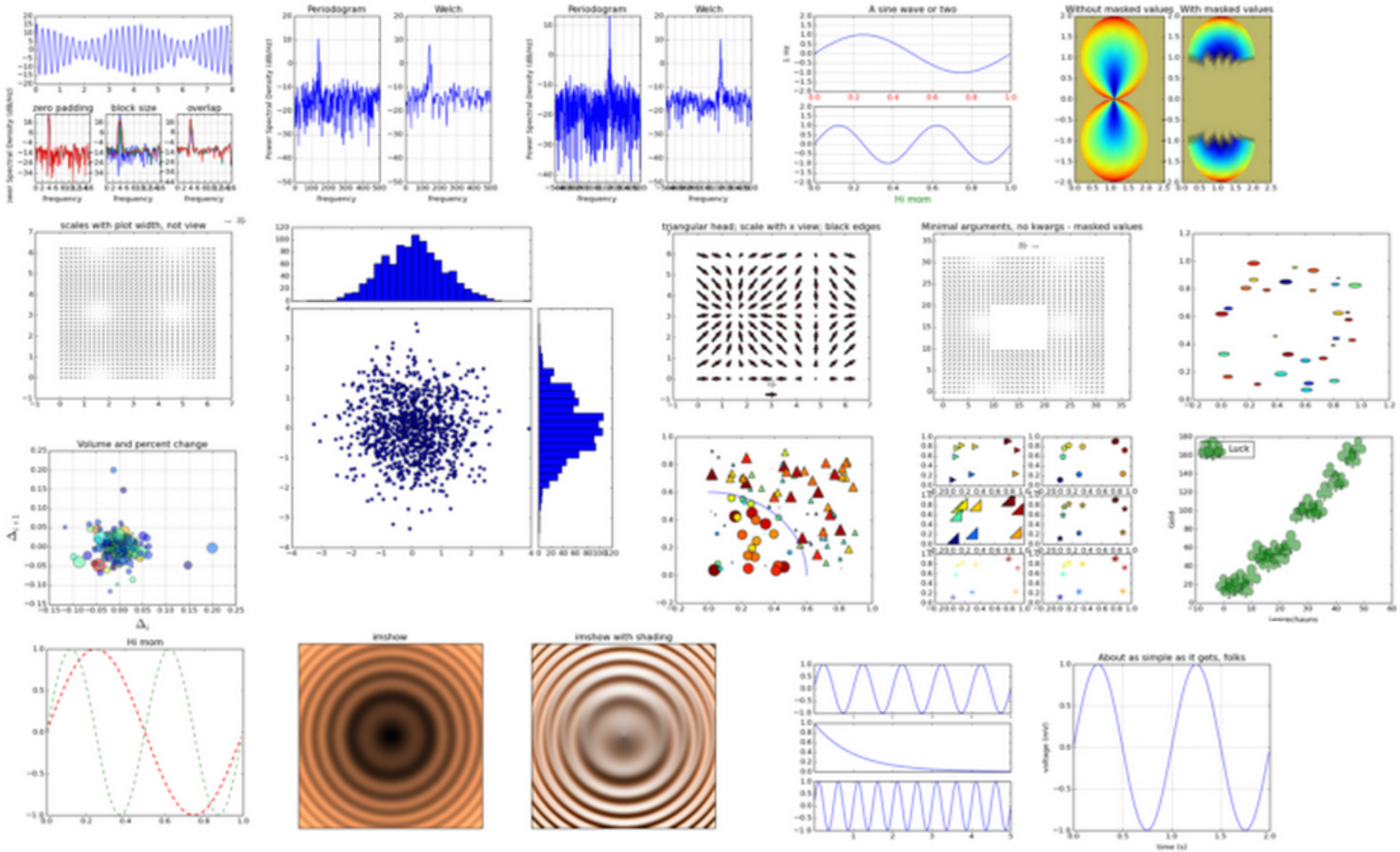
- Simple Python function `simpsonAdaptive(function,a,b,tolerance)` handles both calculation and plotting (∼40 lines of code)
- Code avalable at request

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 avalable at request

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 data visualizations in the web browser via html5

Other Python graphics and visualization libraries

PACKAGE	DESCRIPTION
Mayavi2	3D scientific data visualizer (Python + VTK)
yt project	analysis/visualization of volumetric, multi-resolution data from astrophysical simulations (Enzo, Orion, FLASH, etc.), recently used in other fields
Neuronvisio	GUI for NEURON simulator enviroment
VPython	3D graphics library
PyVisfile	storing data in a variety of scientific visualization file formats
PyVTK	tools for manipulating VTK files in Python
ScientificPython	various Python modules for scientific computing and visualization
Chaco	interactive 2D plotting
NodeBox for OpenGL	2D animations (originally for game development)

2D/3D visualization packages

Visualization	
GNUPLOT	A command-driven interactive 2d and 3d function plotting program
GRAPHVIZ	Represent structural information as diagrams of abstract graphs and networks
HDFVIEW	Visual Tool for Browsing and Editing HDF4 and HDF5 files.
IMAGEMAGICK	Software for the creation, modification and display of bitmap images
MOLDEN	A pre- and post processing program of molecular and electronic structure
OPENCV	A Library of programming Functions for Real Time Computer Vision.
PARAVIEW	Parallel Visualization Application
SCILAB	Open Source Platform for Numerical Computation
VISIT	VisIT Visualization Tool
VMD	Visual Molecular Dynamics
XCRYSDEN	Crystalline and Molecular Structure Visualisation
YT	Python-based package for visualization of AMR datasets

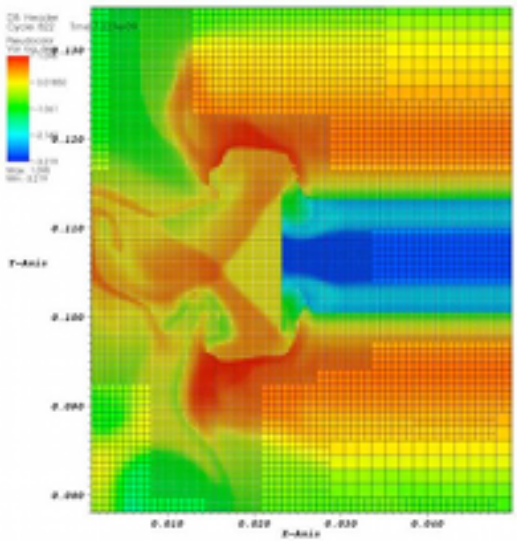
- Open-source, multi-platform, and general-purpose:
  - visualize scalar and vector fields
  - structured and unstructured meshes in 2D and 3D, particle data, polygonal data, irregular topologies
  - ability to handle very large datasets (GBs to TBs)
  - ability to scale to large ($10^3 - 10^5$ cores) computing facilities
  - iteractive manipulation
  - support for scripting, common data formats, parallel I/O

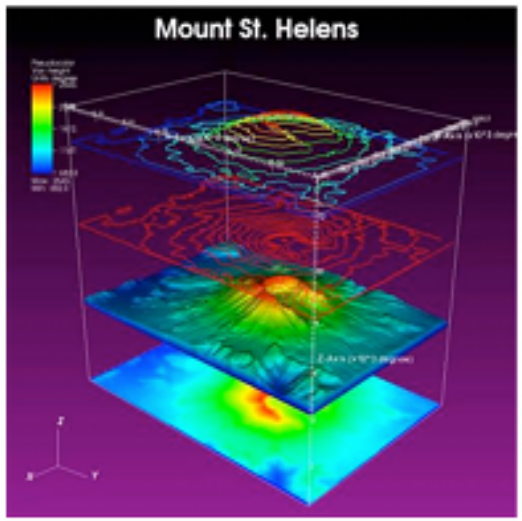
(1) VisIT (latest is 2.7.2) – installed on vizN-site

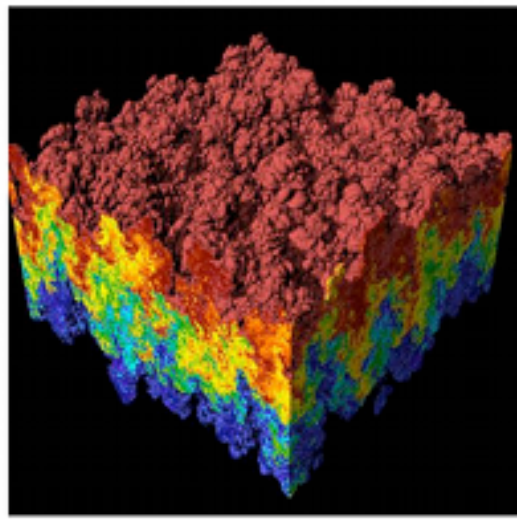
(2) ParaView (latest is 4.1) – installed on vizN-site

VisIT

- `http://wci.llnl.gov/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
- v2.7.2 available as source and binary for Linux/Mac/Windows
- Over 80 visualization features (contour, mesh, slice, volume, molecule, ...)
- Reads over 110 different file formats
- Interfaces with C++, Python, and Java
- Uses MPI for distributed-memory parallelism on HPC clusters



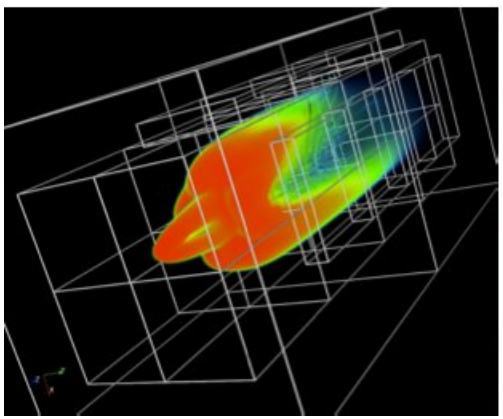
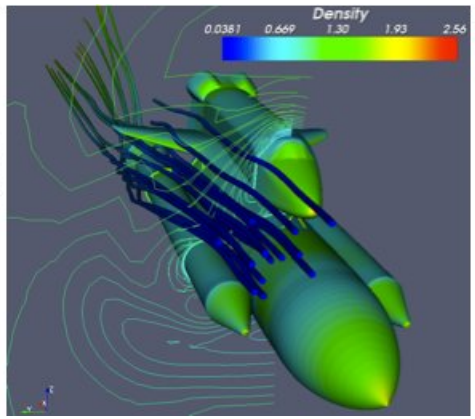


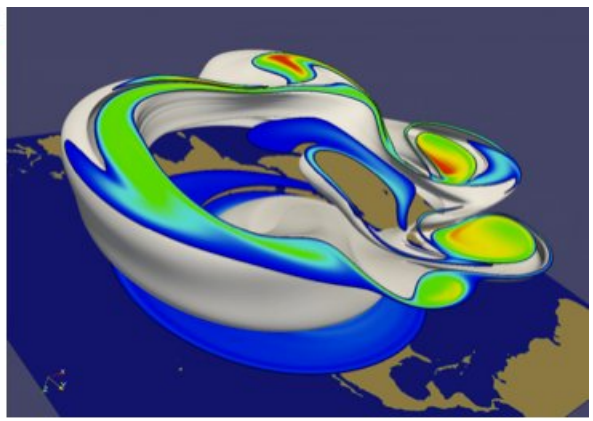


Lawrence Livermore National Laboratory

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
- Latest stable release 4.1, available 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. also see Mayavi (written in Python + numpy + scipy + VTK); note that VTK can be used from C++, Tcl, Java, Python as a standalone renderer



Why ParaView for this course?

- A lot of interest in ParaView among HPC users
- When I was developing the first version of this course in 2010, I preferred ParaView's interface to VisIT (purely subjective, but had to choose one)
- Wide binary availability, active development
- Tight integration with VTK (developed by the same folks)
- Support for over 130 input file formats
- Comes with many filters and plugins, including a Mobile Remote plugin to control ParaView from an iOS device (KiwiViewer)
- Provides powerful parallel execution and advanced stereoscopic viewing on 3D hardware
- In the last few years became the de-facto visualization teaching package across several HPC consortia in Canada

- Visualization is not limited to ParaView $\Rightarrow$ I encourage you to also try VisIT and other open-source packages

Online resources

ParaView on SHARCNET
`http://www.sharcnet.ca/my/software/show/67`
`http://www.sharcnet.ca/help/index.php/ParaView`

ParaView official documentation
`http://www.paraview.org/OnlineHelpCurrent`

ParaView wiki `http://www.paraview.org/Wiki/ParaView`

ParaView/Python batch scripting
`http://www.paraview.org/Wiki/ParaView/Python_Scripting`

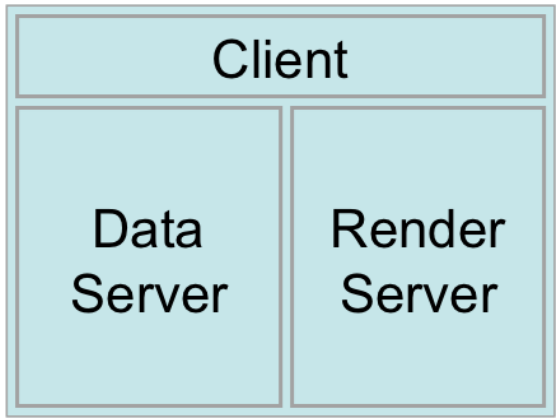
VTK tutorials `http://www.itk.org/Wiki/VTK/Tutorials`

ParaView architecture

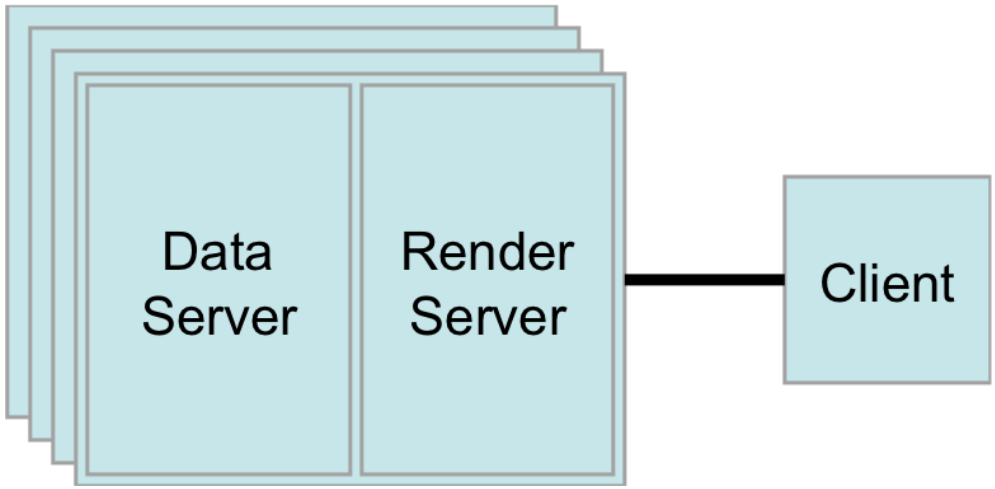
Three logical units of ParaView – these units can be embedded in the same application on the same computer, but can also run on different machines:

- **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

PV client-server mode in Compute Canada consortia

- **WestGrid**: GPU rendering on parallel.westgrid.ca `http://bit.ly/1p5ZWKi`
- **SciNet**: CPU rendering on SciNet's GPC nodes `http://bit.ly/1p60dx0`
- **Calcul Québec**: on colosse.calculquebec.ca now testing CPU rendering, plan to offer GPU rendering; offer GPU rendering on guillimin.clumeq.ca
- **SHARCNET**: used to offer ParaView server on rainbow.sharcnet.ca, due to lack of demand switched to offering standalone ParaView on ∼20 visualization workstations (accessible via VNC) with up to 48GB memory, mounting global /home and /work via sshfs
  - this setup has its limitations: limited memory and limited I/O bandwidth
  - works well up to 2048^3 single-precision variable on structured grids stored locally
  - larger datasets, more complex grids, or datasets requiring complex filters won't fit
  - recent problem that won't fit on our workstations 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

Starting ParaView

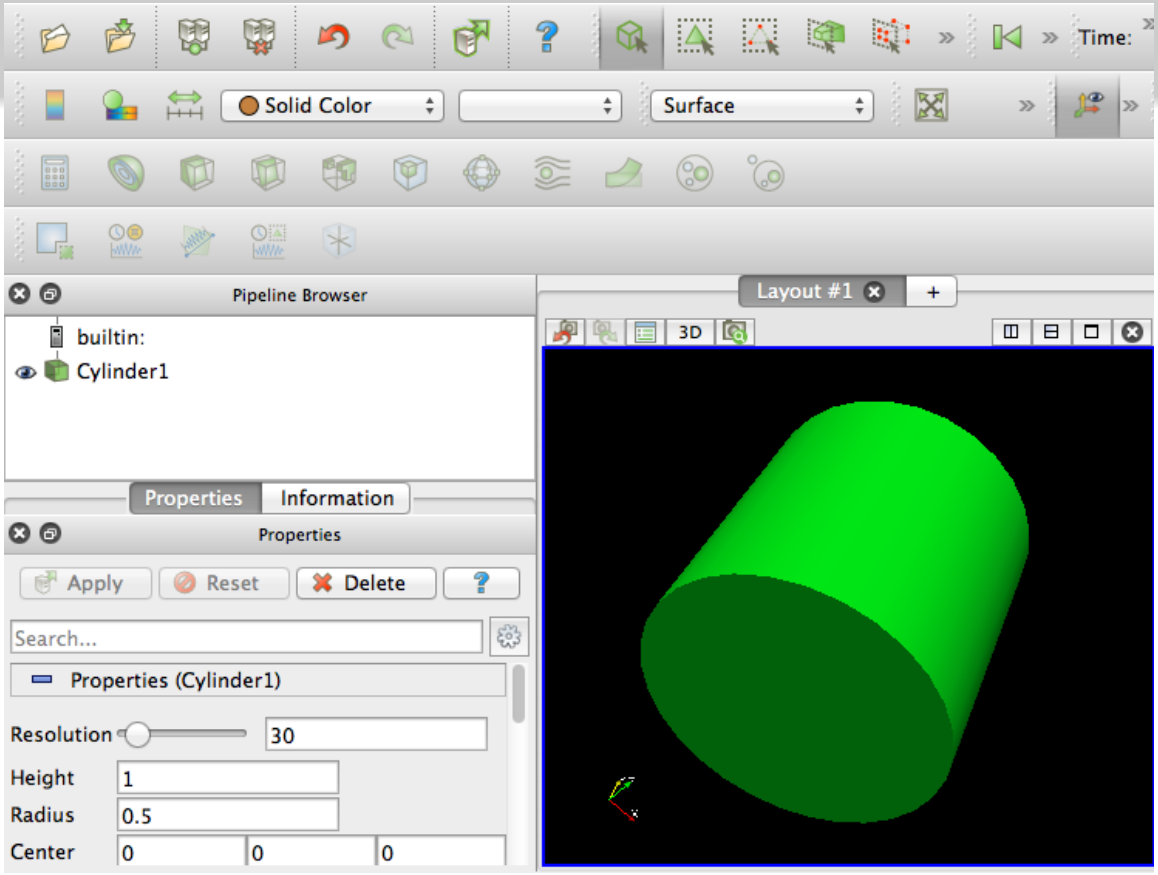
- Today we'll do everything in standalone ParaView on your desktop

- 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

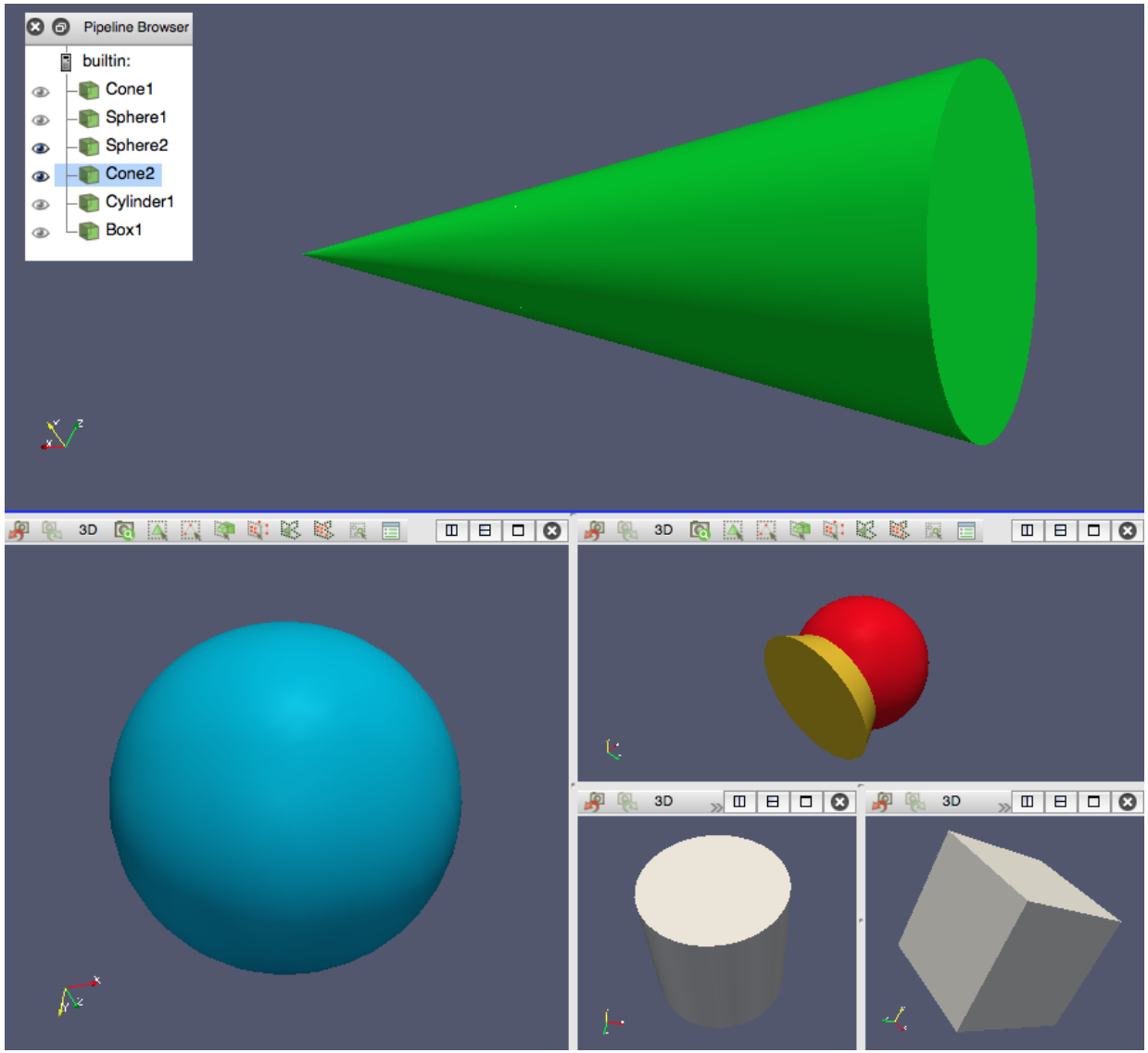
- **Pipeline Browser**: data readers, data filters, can turn visibility of each object on/off
- **Object Inspector**: view and change parameters of the current pipeline object (via tabs properties-display-info or properties-info)
- **View window**: displays the result



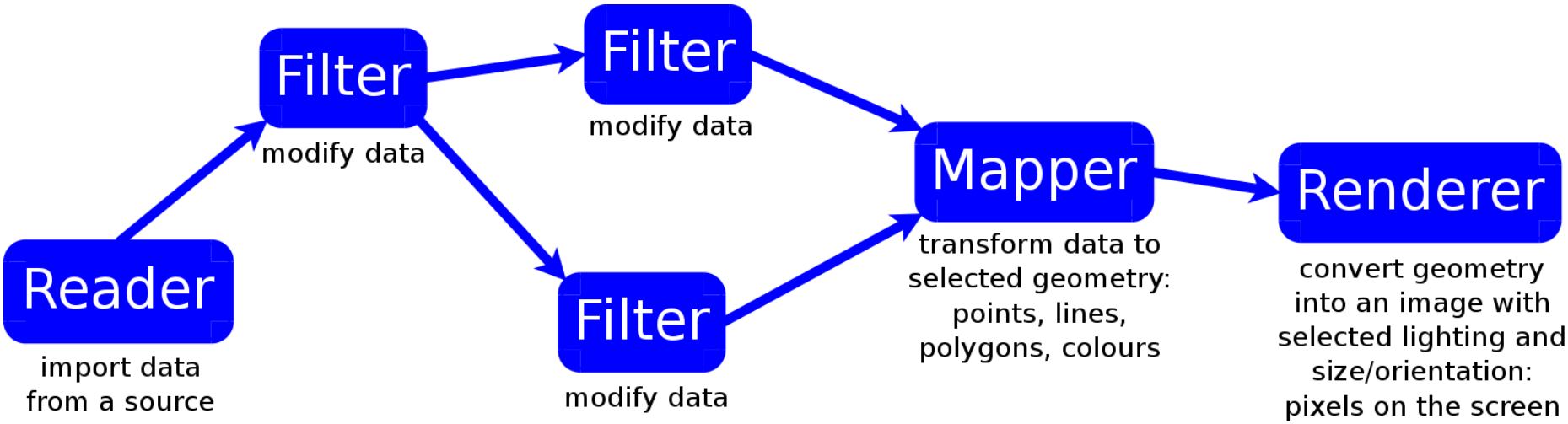
1. Find the following in the toolbar: "Connect", "Disconnect", "Toggle Colour Legend Visibility", "Edit Colour Map", "Rescale to Data Range"
2. Load a predefined dataset: in ParaView select Sources → Cylinder
3. Try dragging the cylinder using the left mouse button; also try the same with the right and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



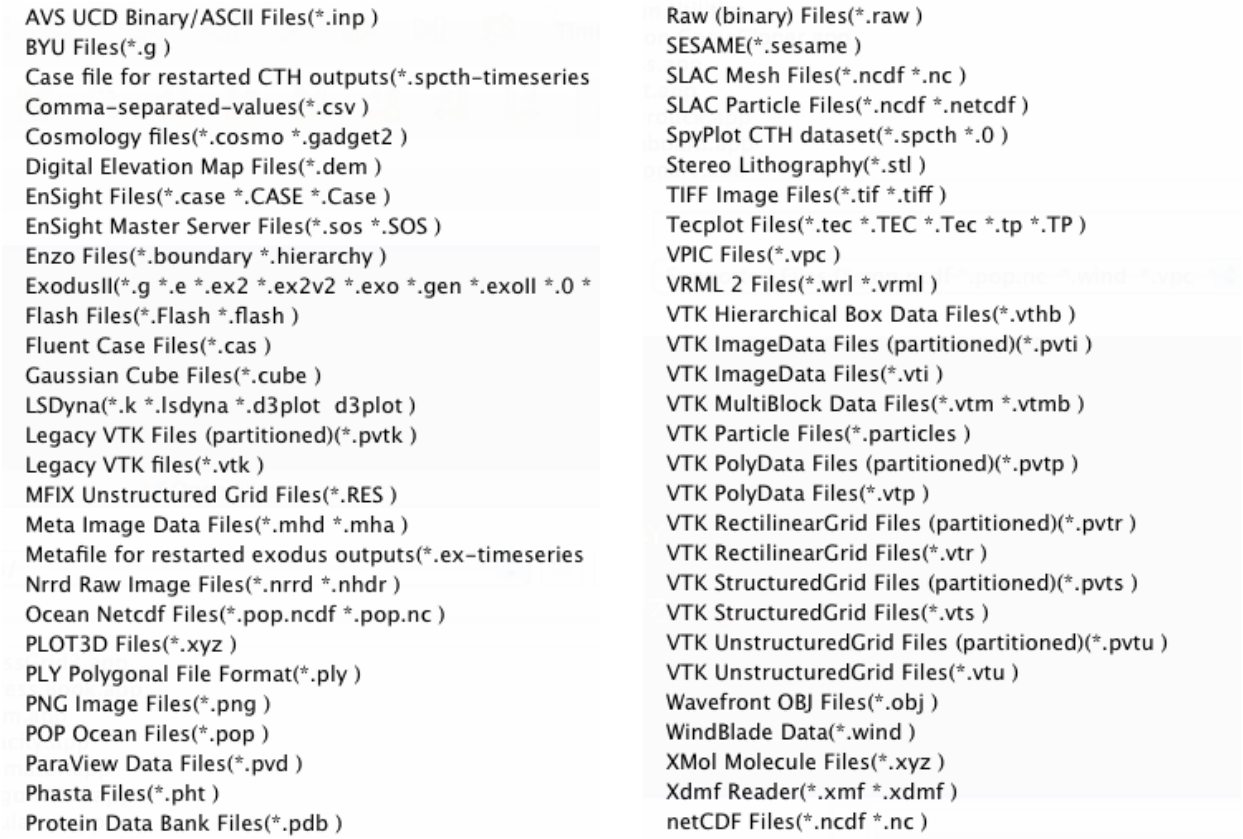
Visualization pipeline



- 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

Data sources

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



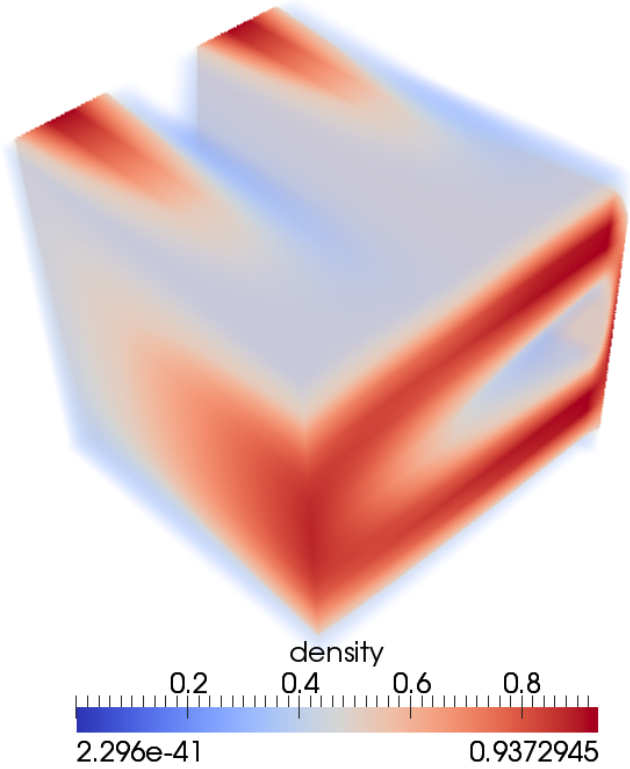
Somewhat incomplete list of file readers:

`http://paraview.org/OnlineHelpCurrent/ParaViewReaders.html`

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/raw/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

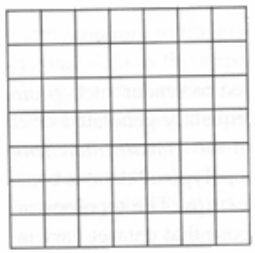


VTK = Visualization Toolkit

- Open-source software system for 3D computer graphics, image processing and visualization
- Bindings to C++, Tcl, Java, Python
- ParaView is based on VTK $\Rightarrow$ 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 (newer, much preferred, supports parallel file I/O, extension depends on data type): XML tags + ASCII/binary/compressed data

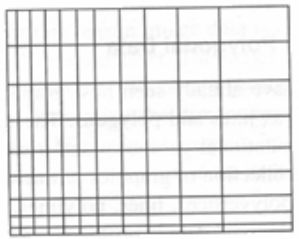
VTK 3D data: 6 major dataset (discretization) types

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



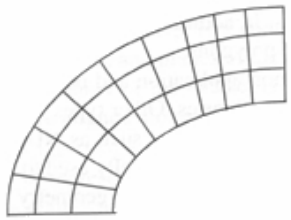
(a) Image Data

- **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

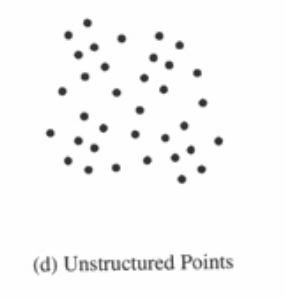
- **Structured Grid**: *.vts, regular topology and irregular geometry, need to indicate coordinates of each point

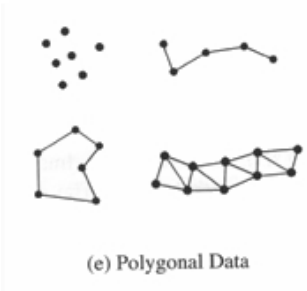


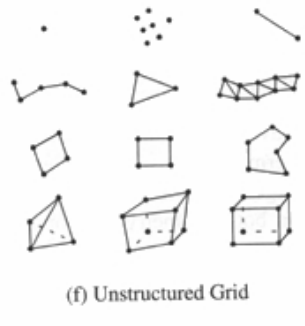
(c) Structured Grid

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

- **Particles/Unstructured Points**: *.particles
- **Polygonal Data**: *.vtp, unstructured topology and geometry, point coordinates, 2D cells only (i.e. no polyhedra), suited for maps
- **Unstructured Grid**: *.vtu, irregular in both topology and geometry, point coordinates, 2D/3D cells, suited for finite element analysis, structural design


(d) Unstructured Points


(e) Polygonal Data


(f) Unstructured Grid

VTK 3D data: dataset attributes

- Each VTK file can store a number of datasets, each with one of the following attributes
  - 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/vtk/legacy/volume.vtk`
   - simple example (Structured Points): $3 \times 4 \times 6$ dataset, one scalar field, one vector field
2. File: `data/vtk/legacy/density.vtk`
   - another simple example (Structured Grid): $2 \times 2 \times 2$ dataset, one scalar field
3. File: `data/vtk/legacy/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.

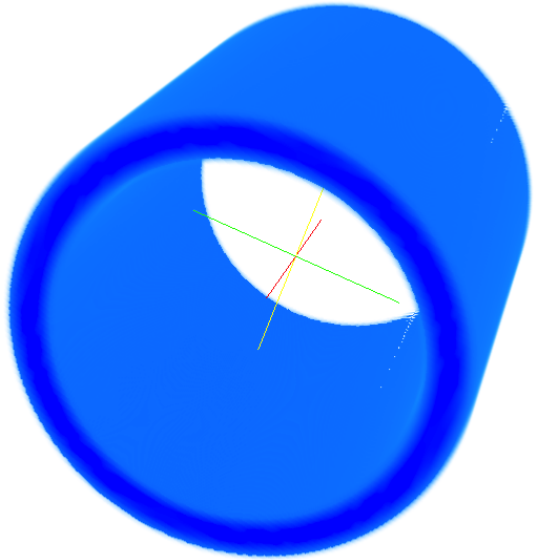
Exercise: visualizing 3D data with legacy VTK

- 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



- Option 1: for your convenience I already wrote `code/writeVolume.cpp | code/writeVolume.f90`, sampling the function at 30^3 and using `data/vtk/legacy/volume.vtk` as a template
  - the code creates a complete VTK file
  - study the code, compile and run it, import data into ParaView
- Option 2: `data/vtk/legacy/cylinder.dat` contains data in ASCII
  - add an appropriate header to create a vtk file
  - use either the code or `data/vtk/legacy/volume.vtk` as a template

Writing XML VTK from C++

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

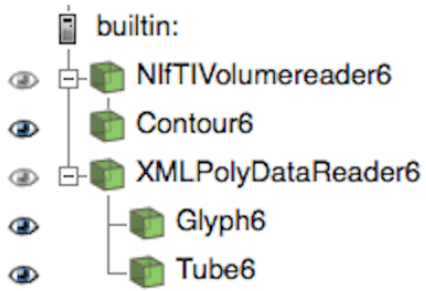
- A good option is to use XML VTK format with binary data and XML metadata, calling VTK library functions from C++ / Java / Python to write data
- Here is an example: `code/SGrid.cpp` and `code/Makefile`, generates the file `data/vtk/xml/halfCylinder.vts`

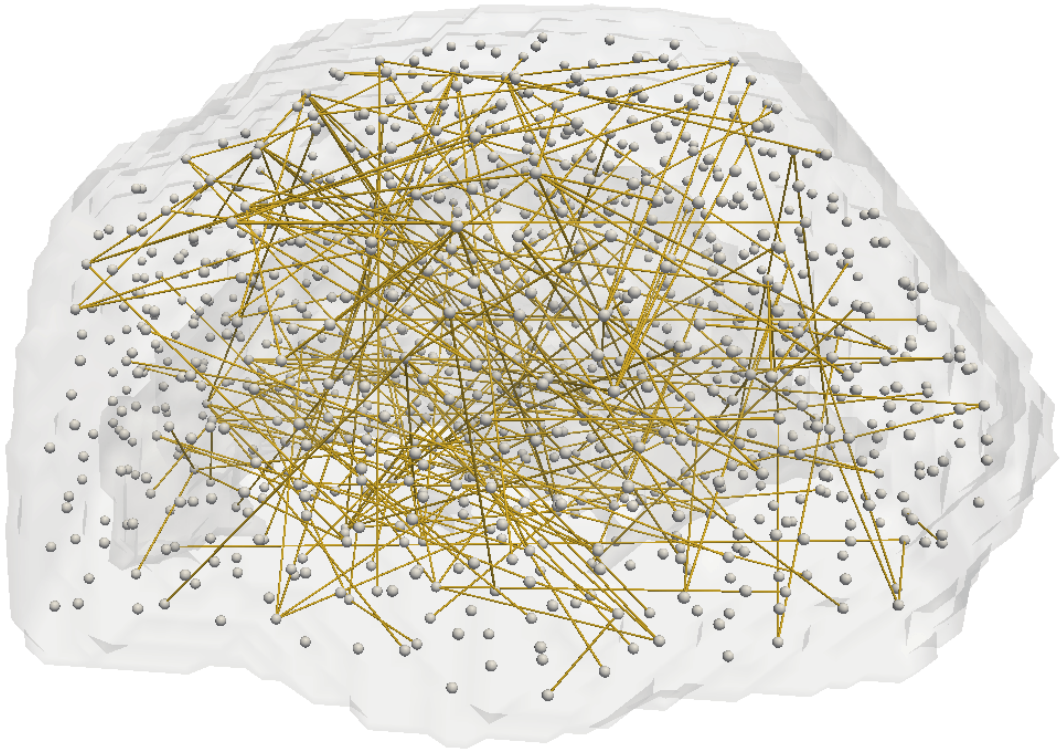
  This example shows how to manually create a structured grid, set grid coordinates, fill the grid with a scalar and a vector, and write it in XML VTK to a *.vts file.
- To run it, you need the VTK library installed (either standalone or pulled from ParaView); check `code/Makefile` to see the required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

```
export LD_LIBRARY_PATH=/path/to/vtk/lib:$LD_LIBRARY_PATH
cd code
make SGrid
./SGrid
```

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

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
  - optionally support compression

NetCDF and HDF5 support in ParaView

- NetCDF supported natively in ParaView (more about it in a minute)
- No native support for HDF5, however, ParaView supports a container format XDMF which uses HDF5 for actual data
- Also support for a number of file formats generated by third-party software that in turn use HDF5 underneath

XDMF = eXtensible Data Model and Format

- 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
- Also available 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`

NetCDF

- `code/writeNetCDF.c` (Fortran version `code/writeNetCDF.f90`) writes a 30^3 volume with a doughnut shape at the centre in NetCDF

C example

```
module load netcdf/intel/4.2
icc writeNetCDF.c -o writeNetCDF $CPPFLAGS $LDFLAGS -lnetcdf
./writeNetCDF
```

f90 example

```
module load netcdf/intel/4.2
ifort writeNetCDF.f90 -o writeNetCDF $CPPFLAGS $LDFLAGS -lnetcdff -lnetcdf
./writeNetCDF
```

Recap of input file formats

Now you know how to import data from your code:

- 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**

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/vtk/legacy/2d000.vtk`), but we might want to see it in 3D by displaying the elevation $z = f(x, y)$
- volumetric view – available only for Structured Points (regularly spaced grid) among all VTK datasets

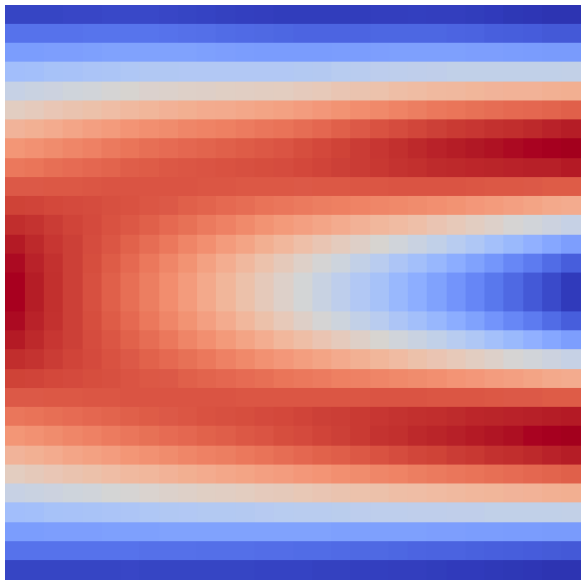
Filters are functional units that process the data to generate, extract, or derive additional features. The filter connections form a **visualization pipeline**

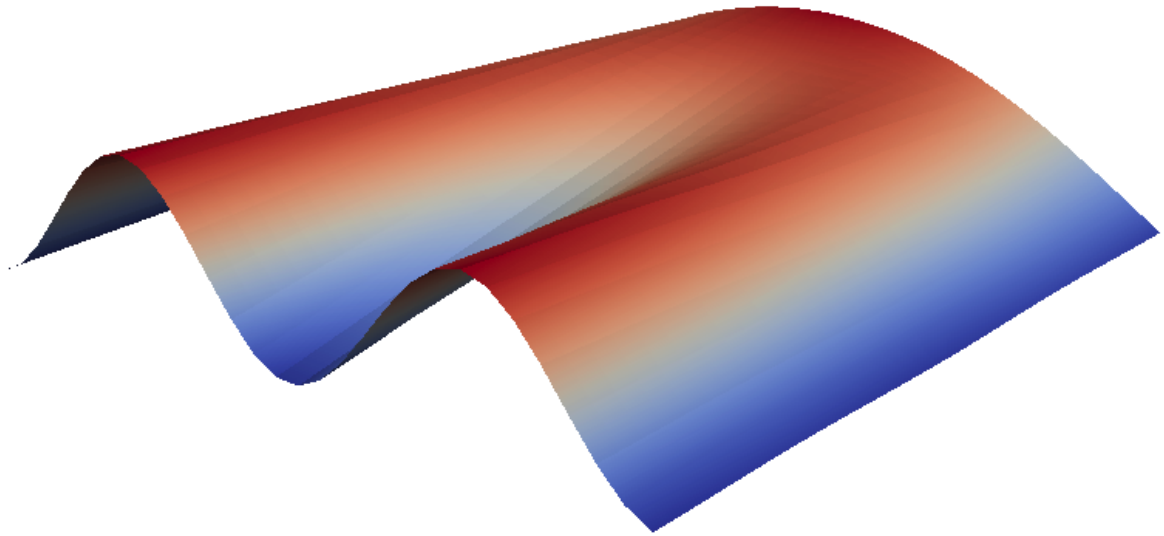
Last time I counted in my ParaView client there were 146 filters. One can add new filters with python scripting

➠ Check out "Filters" in the menu; some are found in the toolbar

Simple filter to visualize a 2D dataset in 3D

- Load the file `data/vtk/legacy/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.

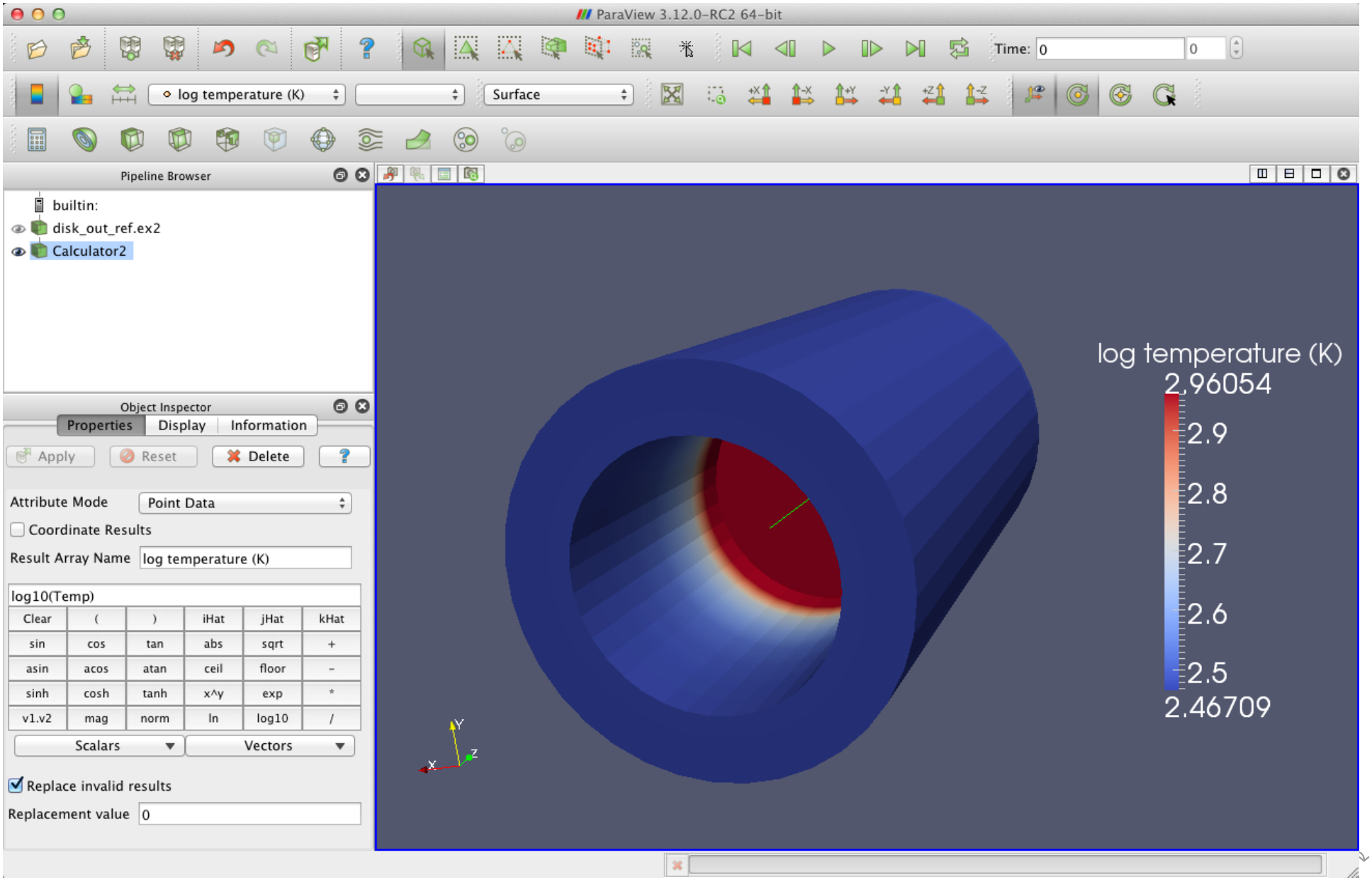
Calculator

- Load one of the datasets, e.g. `data/other/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/other/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

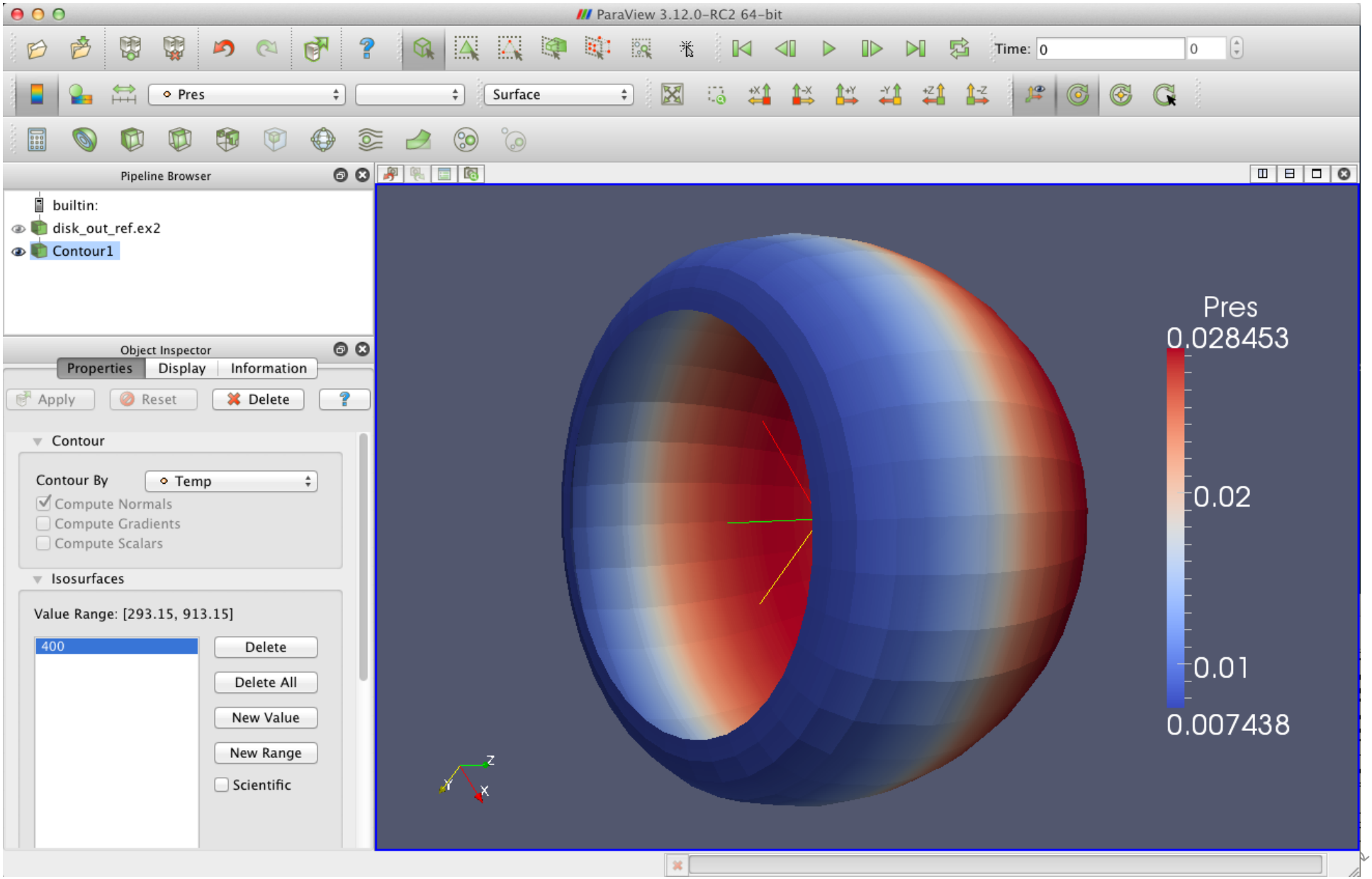
Contour

- 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/other/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/other/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/other/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

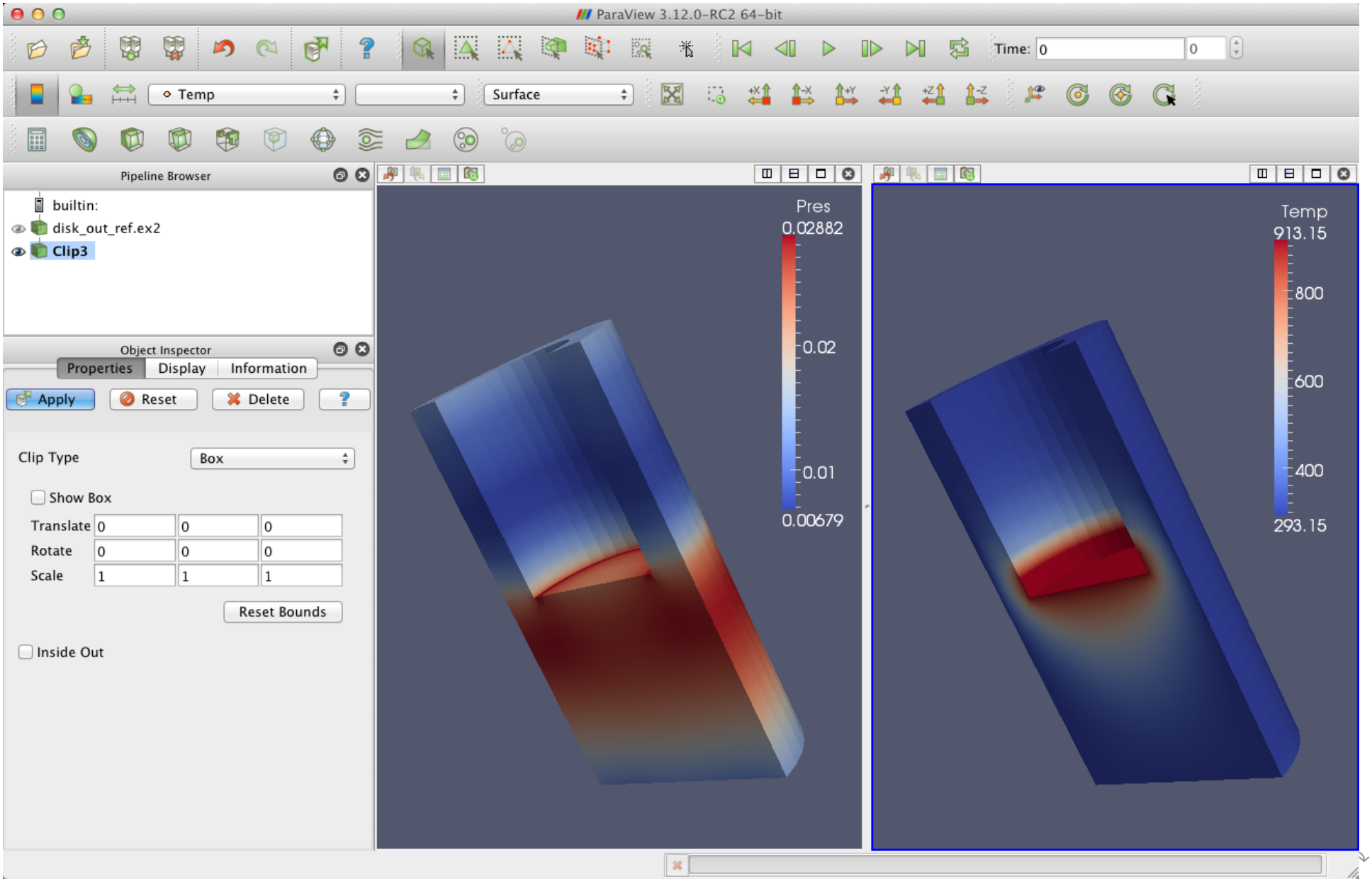
Multiview: several variables side by side

- Start with original data (`data/other/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

Multiview: several variables side by side

- Start with original data (`data/other/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

Multiview: several variables side by side



Vector visualization: streamlines and glyphs

- Start with the original data from `data/other/disk_out_ref.ex2`, load velocity, Temp
- Add the **Stream Tracer** filter, play with Seed Type ("Point Source", "Line Source"), other parameters
- 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/other/disk_out_ref.ex2`, load velocity, Temp
- Add the **Stream Tracer** filter, play with Seed Type ("Point Source", "Line Source"), other parameters
- 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")

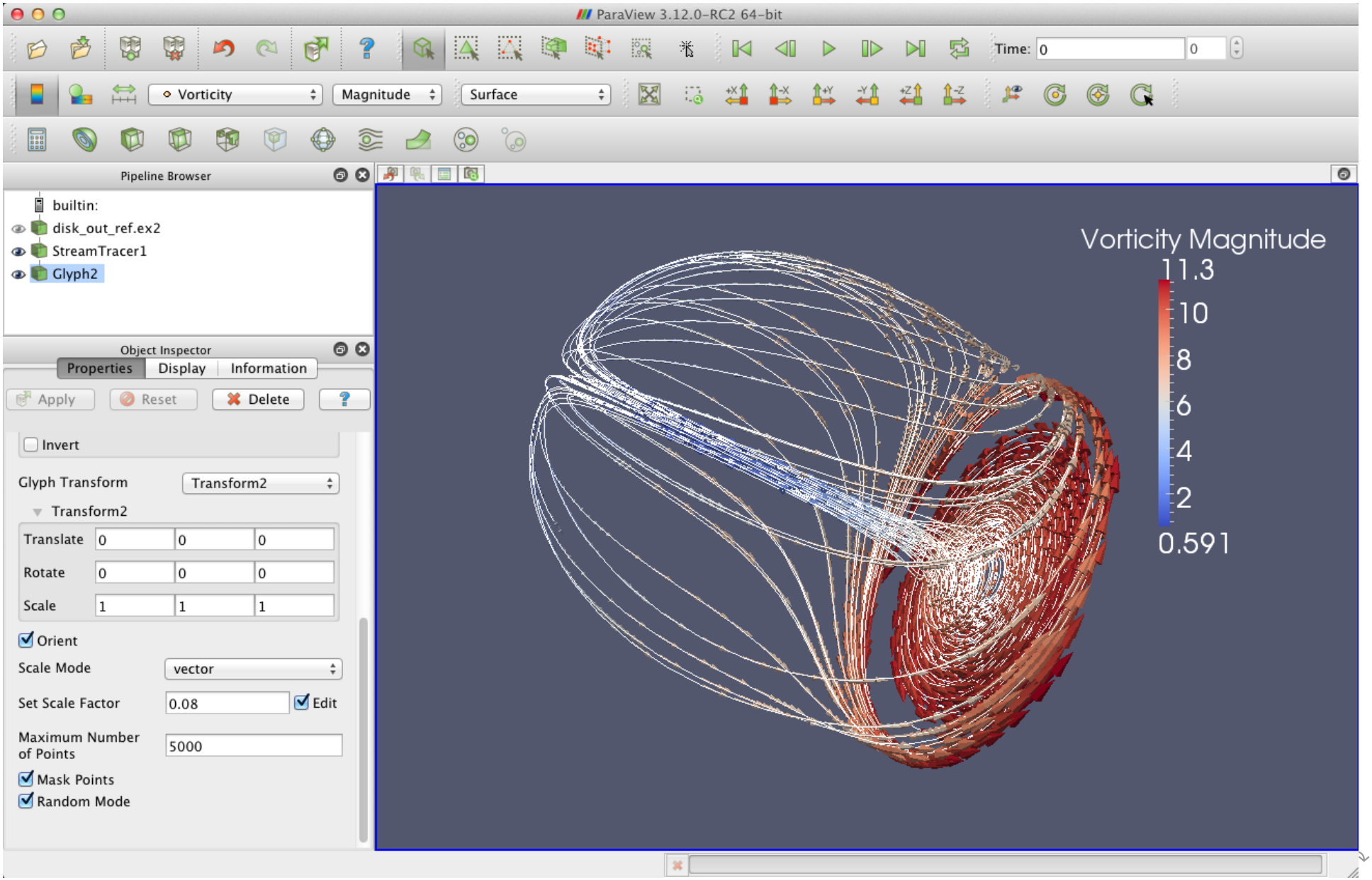
Vector visualization: streamlines and glyphs

- Start with the original data from `data/other/disk_out_ref.ex2`, load velocity, Temp
- Add the **Stream Tracer** filter, play with Seed Type ("Point Source", "Line Source"), other parameters
- 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

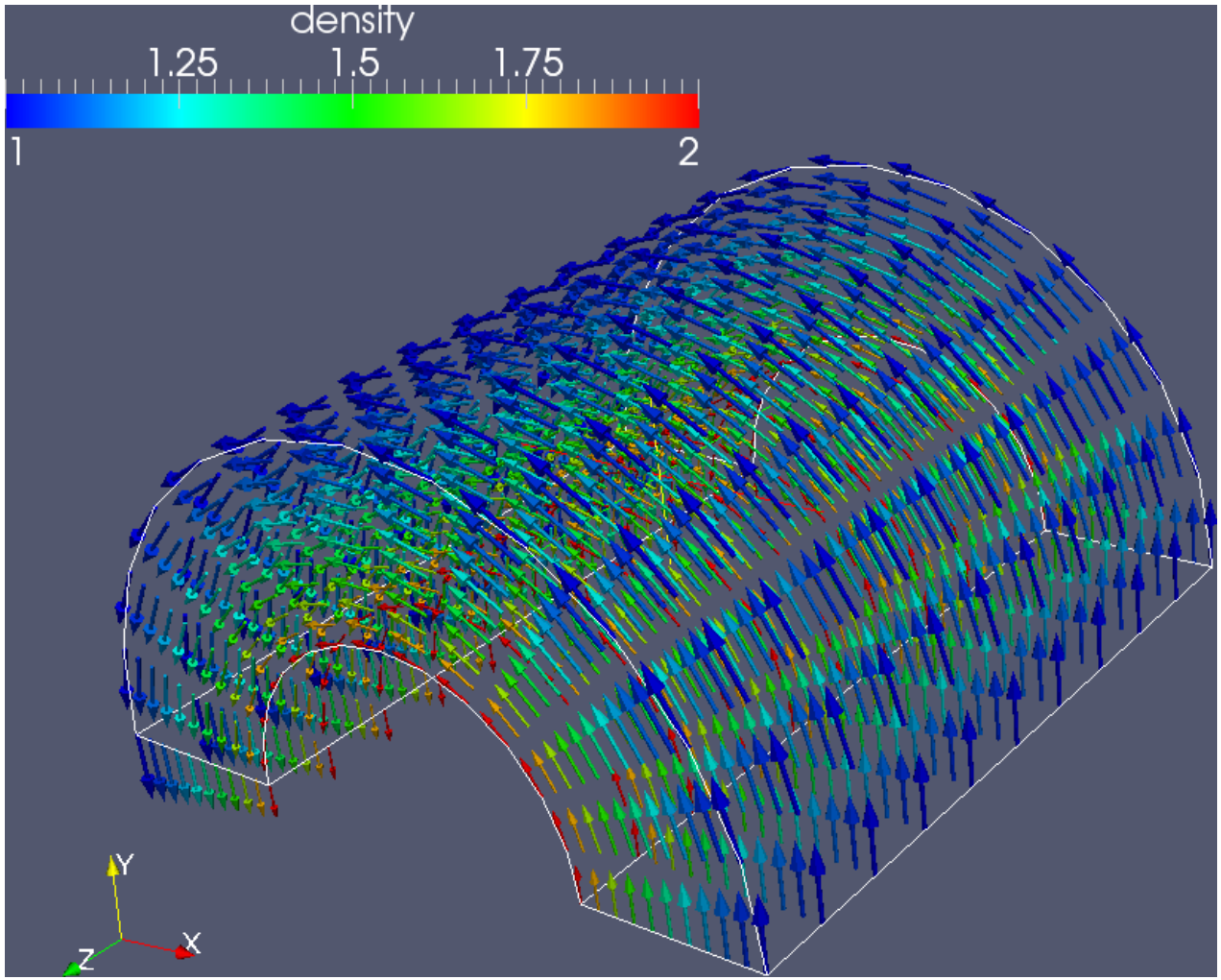
- Start with the original data from `data/other/disk_out_ref.ex2`, load velocity, Temp
- Add the **Stream Tracer** filter, play with Seed Type ("Point Source", "Line Source"), other parameters
- Add shading and depth cues to streamlines: Filters → Alphabetical → **Tube** (could be also called Generate 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/vtk/xml/halfCylinder.vts` and display the velocity field as arrows, colouring them by density – try to reproduce the view below



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

3D optimization exercise

`data/other/stvol.nc` contains a discretized scaled variant of the 3D Styblinski-Tang function inside a unit cube ($x_i \in [0, 1]$), built with `code/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 location of the *global minimum* of $f(x_1, x_2, x_3)$

Batch scripting for automating visualization

- One can automate mundane or repetitive tasks or use ParaView without GUI, complete documentation at `http://www.paraview.org/Wiki/ParaView/Python_Scripting`
- Tools → Python Shell
- *[/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
  - great for making movies!

First script

- Bring up Tools → Python Shell
- "Run Script" `code/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)
help(sphere)
help(Sphere)
```

Using filters

- "Run Script" `code/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()
```

- Now try replacing Show() with Show(sphere)

Reading from files

- "Run Script" `code/readDiskOutRef.py` (change the path!)

readDiskOutRef.py

```
from paraview.simple import *
path = '/Users/razoumov/Dropbox/visualization/data/other/'
reader = ExodusIIReader(FileName=path+'disk_out_ref.ex2')
Show()
Render()
```

- With VTK file formats can use something like:

```
reader = XMLStructuredGridReader(FileName='/Users/.../vtk/xml/halfCylinder.vts')
```

- Starting with ParaView 3.8, can load correct reader automatically using file extension:

```
reader = OpenDataFile('/Users/.../vtk/xml/halfCylinder.vts')
```

Querying field attributes: `readStructuredGrid.py`

```
from paraview.simple import *

path = '/Users/razoumov/Documents/05-summerSchool/visualization/data/vtk/xml/' # edi
reader = OpenDataFile(path+'halfCylinder.vts')

Show()
Render()

print 'print all variables'
print reader.PointData[:]

print 'get a handle to PointData and print all point fields'
pd = reader.PointData
print pd.keys()

print 'get some info about individual fields'
print pd['density'].GetNumberOfComponents()
print pd['density'].GetRange()
print pd['velocity'].GetNumberOfComponents()

print 'run through all arrays and print the ranges of all components'
for ai in pd.values():
    print ai.GetName(), ai.GetNumberOfComponents(),
    for i in xrange(ai.GetNumberOfComponents()):
        print ai.GetRange(i),
    print
```

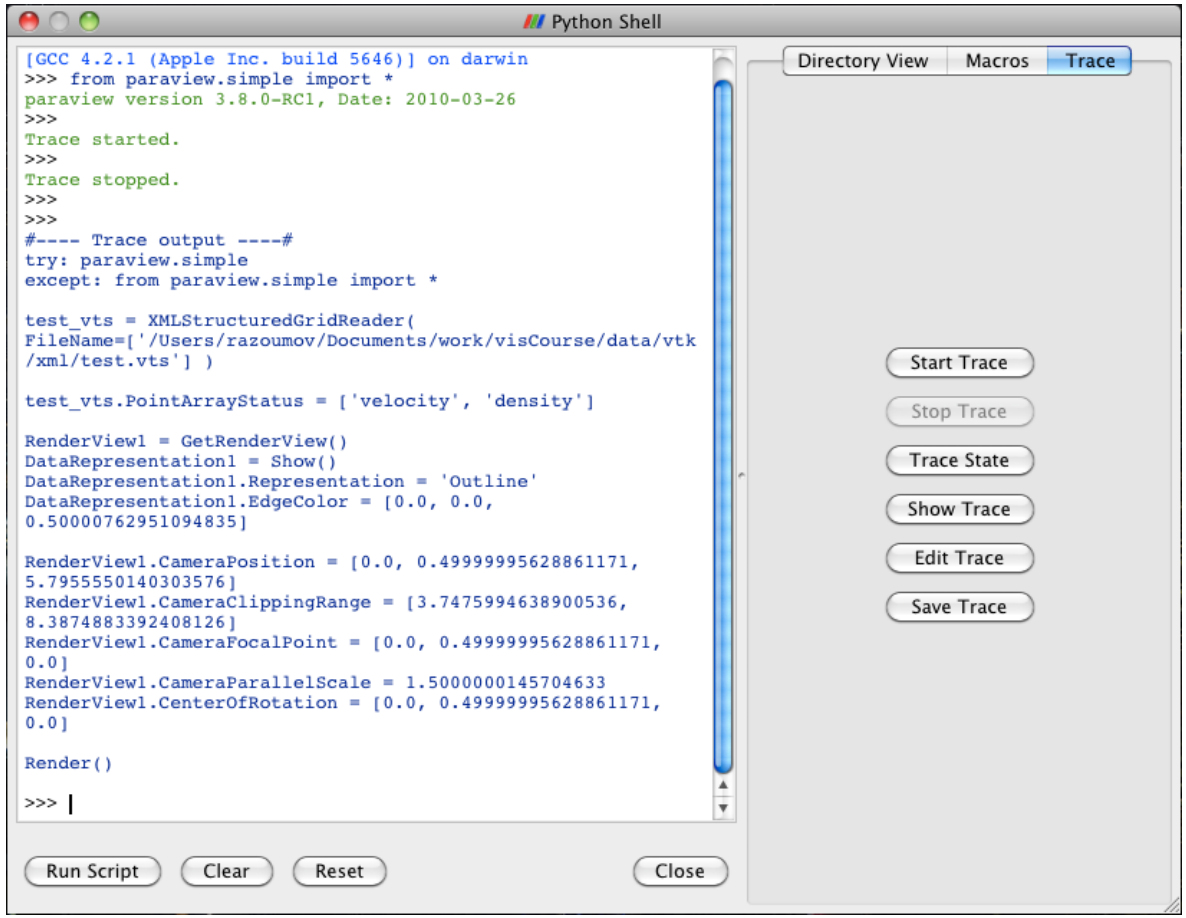
Trace tool

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 Trace | Stop Trace]



More complex example generated via trace

"Run Script" `code/writeImage.py` – draws vector field in half-cylinder

```
from paraview.simple import *

path = '/Users/razoumov/Dropbox/visualization/data/vtk/xml/' # edit the path accordingly
test_vts = XMLStructuredGridReader(FileName=[path+'halfCylinder.vts'])

DataRepresentation1 = Show() # turn on outline
DataRepresentation1.Representation = 'Outline'
DataRepresentation1.EdgeColor = [0.0, 0.0, 0.5]

# set camera position
RenderView = GetRenderView()
RenderView.CameraViewUp = [-0.25, 0.82, -0.51]
RenderView.CameraFocalPoint = [0., 0.5, 0.]
RenderView.CameraClippingRange = [2.91, 9.55]
RenderView.CameraPosition = [1.85, 3.79, 4.40]

Glyph2 = Glyph(GlyphType="Arrow" )
Glyph2.Scalars = ['POINTS', 'density']
Glyph2.SetScaleFactor = 0.2
Glyph2.Vectors = ['POINTS', 'velocity']
Glyph2.SetScaleFactor = 0.2

DataRepresentation2 = Show() # turn on vectors
DataRepresentation2.EdgeColor = [0.0, 0.0, 0.5]
DataRepresentation2.ColorAttributeType = 'POINT_DATA'
DataRepresentation2.ColorArrayName = 'density'

# set colour table
a1_density_PiecewiseFunction = CreatePiecewiseFunction( Points=[-15.70, 0.0, -5.7, 0.0, -4.23, 0.0, -4.07, 0.1, -3.21,
a1_density_PVLookupTable = GetLookupTableForArray( "density", 1, RGBPoints=[1.0, 0.0, 0.0, 0.0, 1.64, 0.90, 0.0, 0.0,
DataRepresentation2.LookupTable = a1_density_PVLookupTable

WriteImage('/Users/razoumov/Desktop/output.png')
Render()
```

Working with pipeline objects

`GetSources()` - get a list of objects
`GetActiveSource()` - get the active object
`SetActiveSource` - change the active object
`GetRepresentation()` - return the *view representation* for the active pipeline object and the active view

the following two scripts produce identical results
(see `getRepresentation.py`):

from paraview.simple import *	from paraview.simple import *
test_vts = XMLStructuredGridReader(FileName=['halfCylinder.vts'])	test_vts = XMLStructuredGridReader(FileName=['halfCylinder.vts'])
DataRepresentation = Show()	Show()
	handle = GetRepresentation()
DataRepresentation.Representation = 'Surface'	handle.Representation = 'Surface'
DataRepresentation.DiffuseColor = [0, 0, 1]	handle.DiffuseColor = [0, 0, 1]
DataRepresentation.SpecularColor = [1, 1, 1]	handle.SpecularColor = [1, 1, 1]
DataRepresentation.SpecularPower = 200	handle.SpecularPower = 200
DataRepresentation.Specular = 1	handle.Specular = 1
Render()	Render()

Animation

1. Use ParaView's built-in animation of any property of any pipeline object
   - lets you 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 "play"
2. Use ParaView's ability to recognize a sequence of similar files
   - reasonably powerful, very convenient
   - try loading `data/vtk/legacy/2d*.vtk` sequence and animating it (visualize one frame and then press "play")
3. Script your animation in Python
   - steep learning curve, very powerful
   - typical usage scenario: generate one frame per input file
   - we'll try a simpler exercise without input files – see next slide

Animation

1. Use ParaView's built-in animation of any property of any pipeline object
   - lets you 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 "play"
2. Use ParaView's ability to recognize a sequence of similar files
   - reasonably powerful, very convenient
   - try loading `data/vtk/legacy/2d*.vtk` sequence and animating it (visualize one frame and then press "play")
3. Script your animation in Python
   - steep learning curve, very powerful
   - typical usage scenario: generate one frame per input file
   - we'll try a simpler exercise without input files – see next slide

Animation

1. Use ParaView's built-in animation of any property of any pipeline object
   - lets you 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 "play"
2. Use ParaView's ability to recognize a sequence of similar files
   - reasonably powerful, very convenient
   - try loading `data/vtk/legacy/2d*.vtk` sequence and animating it (visualize one frame and then press "play")
3. Script your animation in Python
   - steep learning curve, very powerful
   - typical usage scenario: generate one frame per input file
   - we'll try a simpler exercise without input files – see next slide

Exercise: scripting cone animation

- Write a script that
  - imports Sources → Cone
  - sets the cone's resolution to $6 + i$, where $i = 0, \ldots, 20$
  - saves each frame to a file called "cone"+str(i)+".png"
  - optionally make a movie from these frames

1. Use trace tool to create a script that loads a single cone, sets its resolution, and then stores visualization in a PNG file
2. Edit the script to enclose its content into a loop

```
for i in range(21):
    <visualize and save the cone of resolution 6+i>
```

3. Run the script to produce 21 image files
4. If you can in your OS, make a movie from these frames

Visualizing remote data

- So far we covered working with standalone ParaView on your desktop
- Let's say, your dataset is on cluster.consortium.ca
  $\Rightarrow$ fundamentally there are three options:
  1. download data to your desktop and visualize it locally (limited by dataset size and your desktop's CPU+GPU/memory)
  2. in SHARCNET can work through a visualization workstation
     yourDesktop $\xrightarrow{\text{ssh/VNC}}$ vizN-site running ParaView mounting /work and /home via sshfs
     performance limited by memory (up to 48GB) and I/O speed
  3. work in the **client-server mode** connecting to cluster.consortium.ca directly
     ParaView client on yourDesktop $\rightleftharpoons$ ParaView server on remote cluster
     - currently not used in SHARCNET but possible to set up if necessary
     - available in other consortia, setup details vary

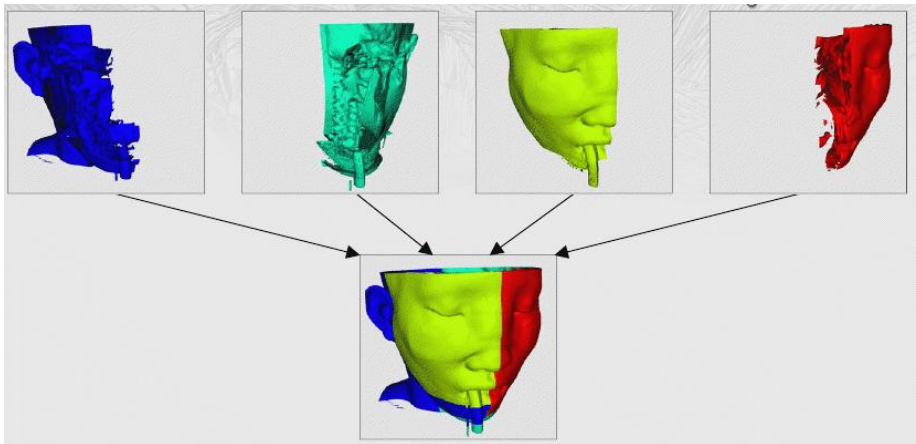
Data partitioning

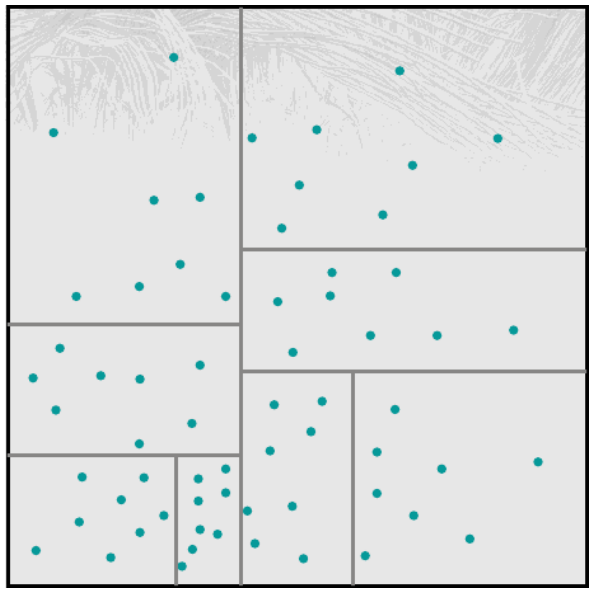
Scalable parallel distributed rendering – load balancing is handled automatically by ParaView for structured data:

- 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

- Working with structured data (Structured Points, Rectilinear Grid, Structured Grid): one processor core per 10-20 million cells ← *from ParaView documentation*
- Unstructured data (Unstructured Points, Polygonal Data, Unstructured Grid): one processor core per 0.5-1 million cells ← *from ParaView documentation*
- In practice on a desktop, memory and I/O speeds are the main issues (unless you do heavy processing), e.g. with structured data can do:
  - $\sim 1000^3$ on a notebook with 4 GB memory, single/dual core
  - $\sim 2000^3$ on a viz workstation with 50 GB memory, dual/quad core, if dataset is stored locally (/scratch)
- On large HPC systems 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

Working with large datasets

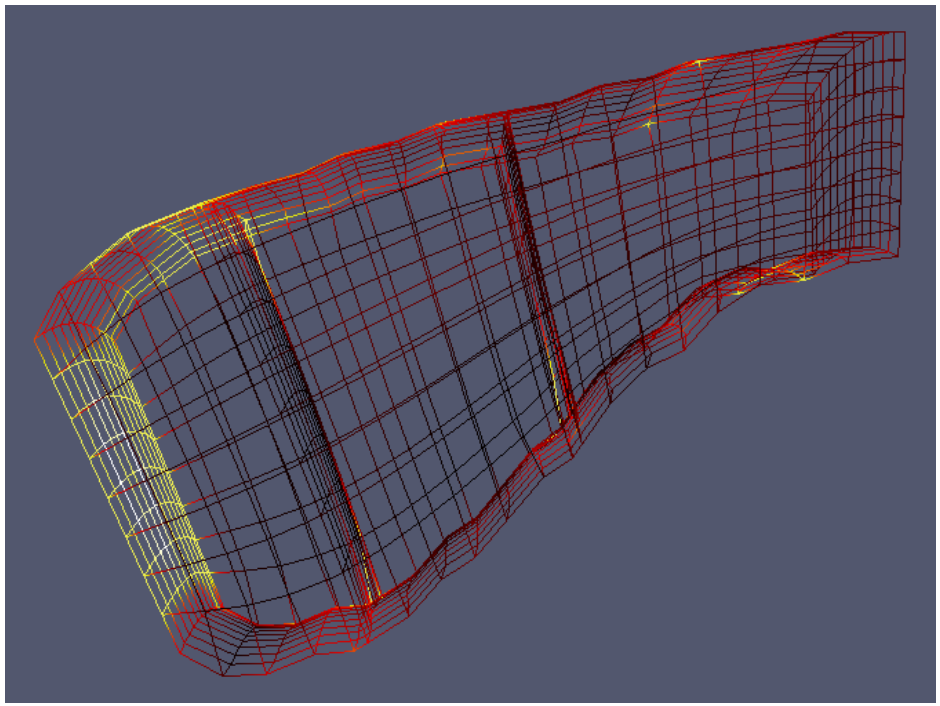
Some filters **should not be used with structured data**:
they write unstructured data, can be heavy on memory usage

- Append Datasets
- Append Geometry
- Clean
- Clean to Grid
- Connectivity
- D3
- Delaunay 2D/3D
- Extract Edges
- Linear Extrusion
- Loop Subdivision
- Reflect
- Rotational Extrusion
- Shrink
- Smooth
- Subdivide
- Tessellate
- Tetrahedralize
- Triangle Strips
- Triangulate

Use these with caution: Clip, Decimate, Extract Cells by Region, Extract Selection, Quadric Clustering, Threshold (also write unstructured, but not as heavy on memory)

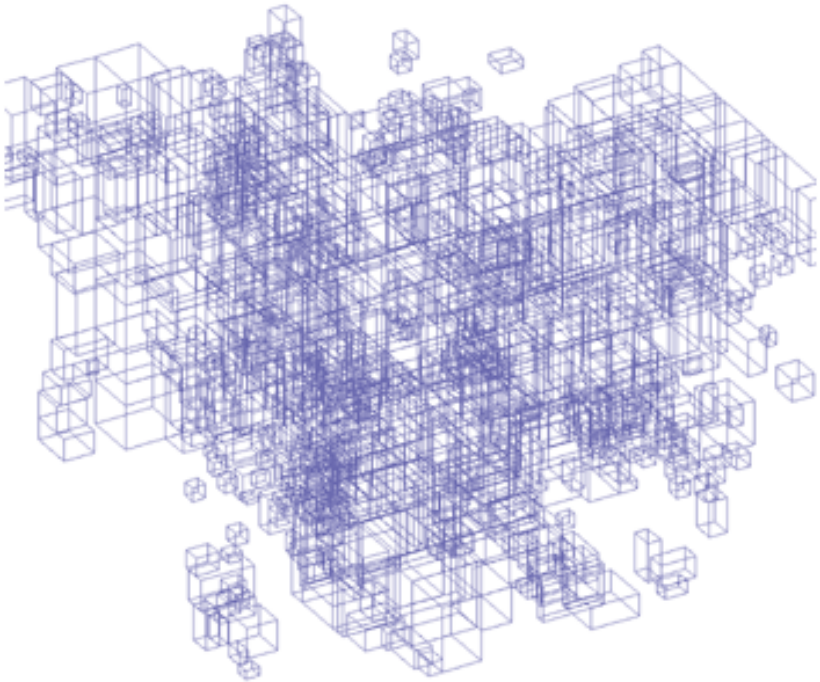
VTK composite datasets: vtkMultiBlockDataSet

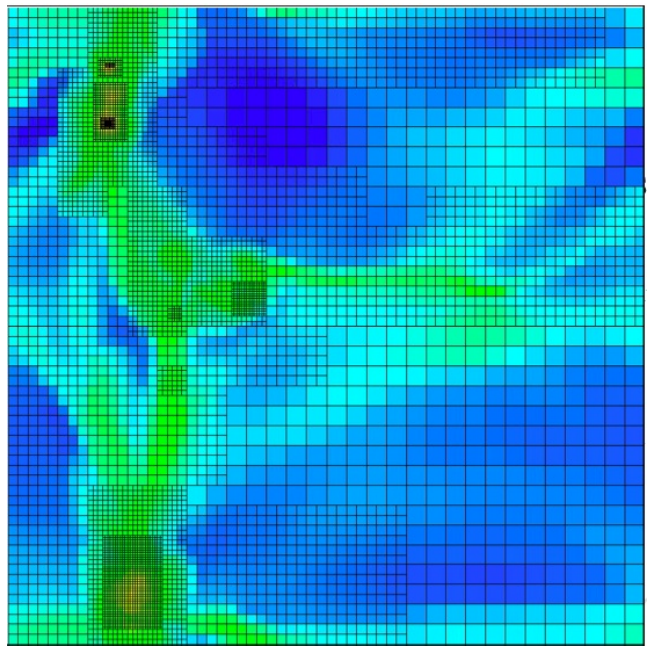
- **vtkMultiBlockDataSet** is a dataset comprising of blocks. Each block can be either a leaf (non-composite), or an instance of vtkMultiBlockDataSet itself – this makes is possible to build trees
- Study MultiBlock.cpp (adapted from from VTK/Examples/MultiBlock): loads three separate structured grid datasets, each from its own file, and writes them as a single multi-block *.vtm dataset (XML-based file format)



VTK composite datasets: vtkHierarchicalBoxDataSet

- **vtkHierarchicalBoxDataSet** is used for AMR datasets, comprises of refinement levels and uniform grid datasets at each refinement level
- Prototype code hierarchicalBoxDataWriter.cpp (does not assign scalars yet, need to sort out cell centers vs. cell edges) – writes multiple grids as a single hierarchical *.vtm dataset





- More on composite datasets
  `http://www.itk.org/Wiki/Composite_Datasets_in_VTK`
  `http://www.itk.org/Wiki/VTK/Composite_Data_Redesign`

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`

Where to get visualization help

- In SHARCNET
  - submit a problem ticket or email help@sharcnet.ca
  - email me razoumov@sharcnet.ca
  - `http://www.sharcnet.ca/help/index.php/visualization`
- Across the country
  - contact your local consortium
  - for visualization support viz-support@computecanada.ca
  - for general discussion viz-users@computecanada.ca
    (goes to a wider group of visualization users and enthusiasts)
  - `http://computecanada.ca/en/resources/visualization`

Final exercise: mystery dataset - (1) volumetric view

- Load the file `data/vtk/legacy/mystery.vtk` (41MB)
- Use ParaView to explore the dataset: how many / what type of variables, dimensionality, extent in each dimension
- Verify that the size of the file makes sense
- Build a volumetric view (*Volume* in *Representation* drop-down menu) -- you'll find this requires more than one step
  - volumetric view available only for scalars defined on Structured Points
- Edit the colour map to bring all the important details in a single image
- Compare this volumetric view to isosurfaces: which one provides better rendering?

Final exercise: mystery dataset - (2) orthogonal slices

- Slice the same dataset in three orthogonal planes, similar to this plot
- Move the planes to show the best view, play with the colour maps