Scientific visualization with ParaView

UBC Okanagan

Alex Razoumov
alex.razoumov@westgrid.ca

WestGrid / Compute Canada

- copy of these slides in `http://bit.ly/pvslides` ➫ will download a file `paraview.pdf`
- data and C++, Fortran, Python codes in `http://bit.ly/pvarchive` ➫ will download a file `visualization.zip` with two directories inside (`code/` and `data/`)
- install ParaView on your laptop – available from `http://www.paraview.org/download`

Workshop outline

9AM-NOON ➩ MORNING SESSION WITH A BREAK ~10:30AM

- 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, multiview, vectors

NOON-1PM ➩ YOU ARE ON YOUR OWN FOR LUNCH

1PM-4:30PM ➩ AFTERNOON SESSION WITH A BREAK ~2:30PM

- Scripting: batch GUI-less mode, few simple scripts, trace tool, programmable filter/source
- Animation: three approaches, one big exercise on scripting/animation
- Remote visualization: VNC, client-server, batch, large datasets

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

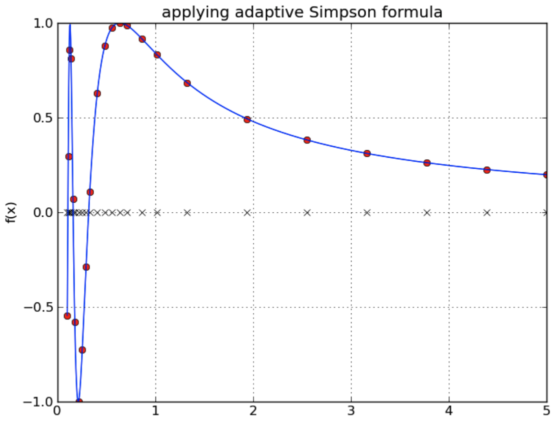
FIELD	VISUALIZATION TYPE
computational fluid dynamics	2D/3D flows, density, temperature, tracers
climate, meteorology, oceanography	fluid dynamics, clouds, chemistry, etc.
quantum chemistry	wave functions
molecular dynamics (phys, chem, bio)	particle/molecular data,
astrophysics	2D/3D fluids, particle 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
  - with some coding, there is nothing you cannot do with open-source tools

Matplotlib example: 1D plotting

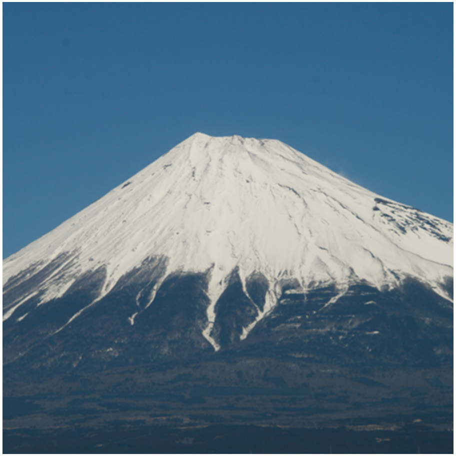
Adaptive Simpson integration

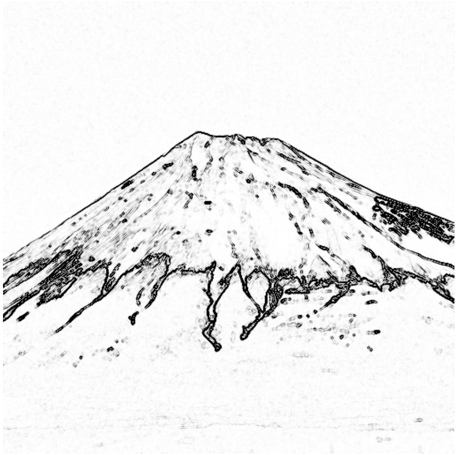


- Simple Python function `simpsonAdaptive(function,a,b,tolerance)` handles both calculation and plotting (∼40 lines of code)
- Code in `code/adaptive.py`

Python Imaging Library (PIL) example: 2D plotting

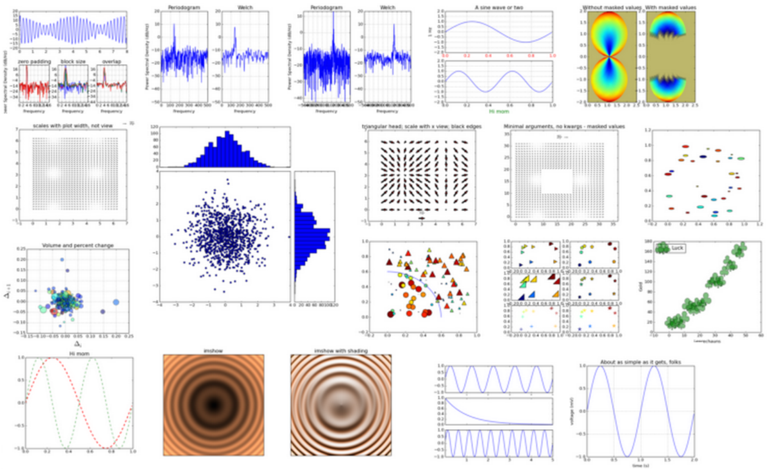
Edge detection using numerical differentiation





- 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 `code/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 data visualizations in the web browser via html5

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, Orion, FLASH, etc.)
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
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

Visualization tools on WestGrid systems

https://www.westgrid.ca/support/software

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

2D/3D visualization packages

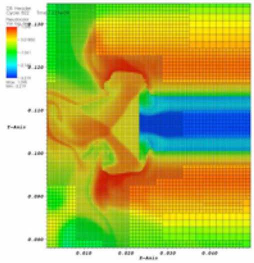
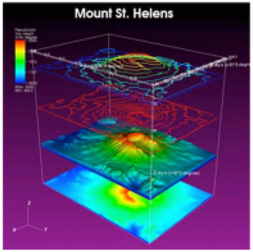
- Open-source, multi-platform, and general-purpose:
  - visualize scalar and vector fields
  - structured and unstructured meshes in 2D and 3D, particle data, polygonal data, irregular topologies
  - ability to handle very large datasets (GBs to TBs)
  - ability to scale to large ($10^3 - 10^5$ cores)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

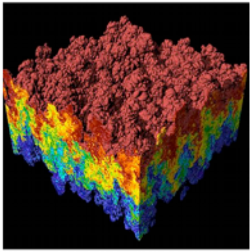
(1) VisIt (latest is 2.10)

(2) ParaView (latest is 5.0)

VisIt

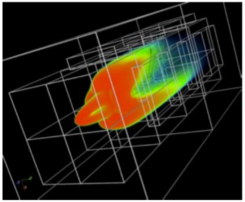
- `https://wci.llnl.gov/simulation/computer-codes/visit`
- Developed by the Department of Energy (DOE) Advanced Simulation and Computing Initiative (ASCI) to visualize results of terascale simulations, first release fall of 2002
- v2.10 available as source and binary for Linux/Mac/Windows
- Over 80 visualization 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

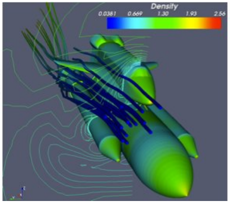



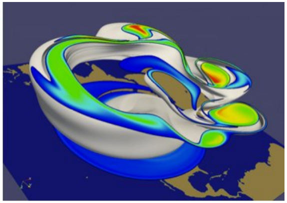

Lawrence Livermore National Laboratory

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
- Latest release 5.0, 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. 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 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), 130 input formats
- Remote client-server visualization, very good scalability
- A number of add-on projects
  - HTML5-based **ParaViewWeb** is a collection of components to connect to a remote ParaView server from a Web application
  - **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)
- Provides powerful parallel execution and stereoscopic viewing on 3D hardware
- Became the de-facto visualization teaching package across Compute Canada
- Visualization is not limited to ParaView $\Rightarrow$ I encourage you to try VisIt `https://www.westgrid.ca/support/software/visit` and other open-source packages. **And we are teaching VisIt at HPCS'2016.**

Online resources

ParaView in WestGrid is installed on parallel (CPU+GPU rendering), bugaboo (CPU only), lattice (CPU only)
`https://www.westgrid.ca/support/software/paraview`

Remote visualization in WestGrid `http://bit.ly/remoteviz`

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

ParaView wikis `http://www.paraview.org/Wiki/ParaView`
`http://www.itk.org/Wiki/ParaView/Users_Guide/Table_Of_Contents`

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

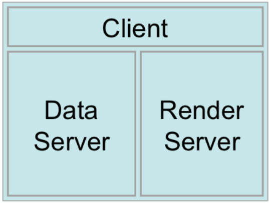
VTK tutorials `http://www.itk.org/Wiki/VTK/Tutorials`

ParaView architecture

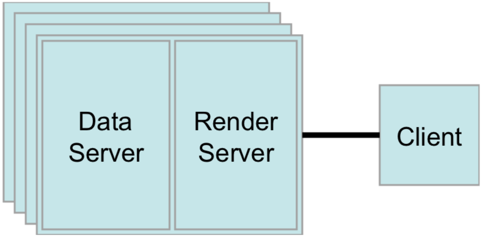
Three logical units of ParaView – these units can be embedded in the same application on the same computer, but can also run on different machines:

- **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

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 $\underline{2048^3}$ $\underline{\underline{\text{single-precision float variable}}}$ on $\underline{\underline{\underline{\text{structured grids}}}}$ $\underline{\underline{\underline{\underline{\text{stored locally}}}}}$
- Larger (and high-precision!) datasets, more complex grids, or datasets requiring complex filters won't fit
- Recent 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 this afternoon

Starting ParaView

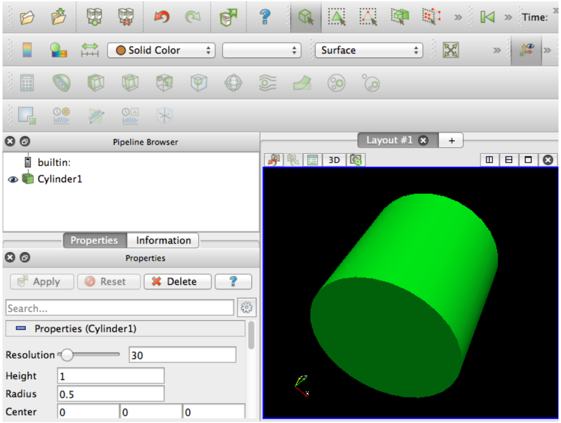
- Today we'll do everything in standalone ParaView on your desktop

- 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

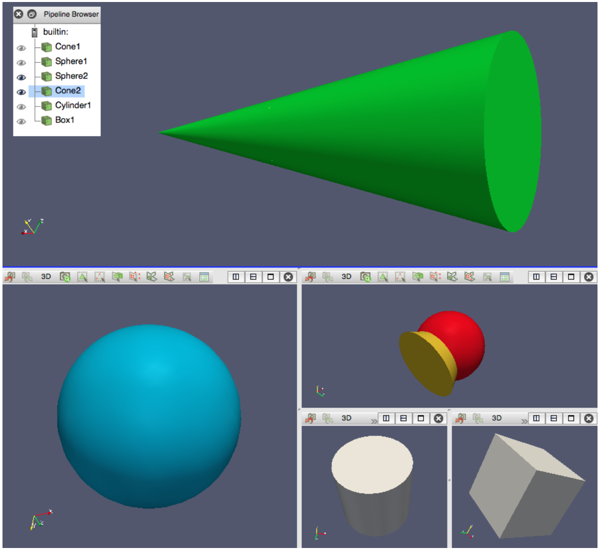


Resolution	30		
Height	1		
Radius	0.5		
Center	0	0	0

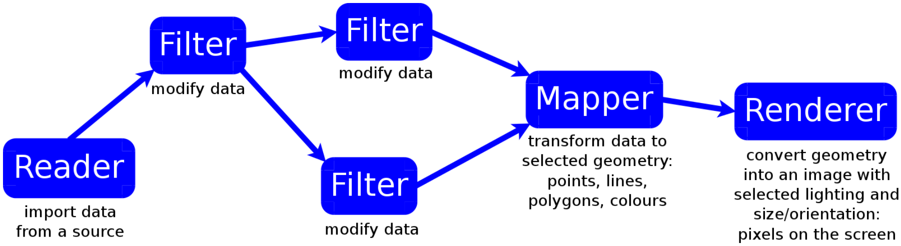
1. Find the following in the toolbar: "Connect", "Disconnect", "Toggle Colour Legend Visibility", "Edit Colour Map", "Rescale to Data Range"
2. Load a predefined dataset: in ParaView select Sources → Cylinder
3. Try dragging the cylinder using the left mouse button; also try the same with the right and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



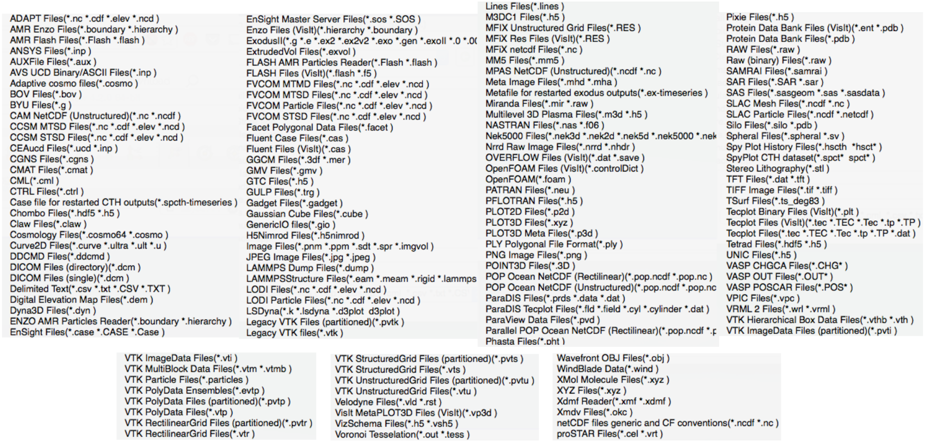
Visualization pipeline


Reader
import data from a source
Filter
modify data
Filter
modify data
Filter
modify data
Mapper
transform data to selected geometry: points, lines, polygons, colours
Renderer
convert geometry into an image with selected lighting and size/orientation: pixels on the screen

- 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

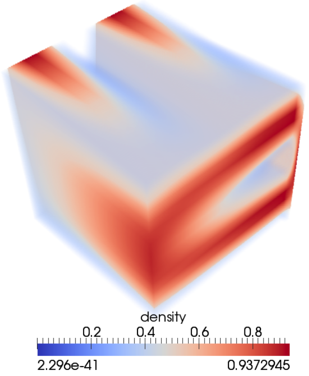


ADAPT Files(*.nc *.cdf *.elev *.ncd)
AMR Enzo Files(*.boundary *.hierarchy)
AMR Flash Files(*.Flash *.flash)
ANSYS Files(*.inp)
AUXFile Files(*.aux)
AVS UCD Binary/ASCII Files(*.inp)
Adaptive cosmo files(*.cosmo)
BOV Files(*.bov)
BYU Files(*.g)
CAM NetCDF (Unstructured)(*.nc *.ncdf)
CCSM MTSD Files(*.nc *.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/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

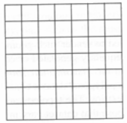


VTK = Visualization Toolkit

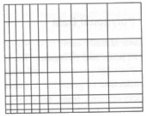
- Open-source software system for 3D computer graphics, image processing and visualization
- Bindings to C++, Tcl, Java, Python
- ParaView is based on VTK ⇒ supports all standard VTK file formats
- VTK file formats
  `http://www.vtk.org/VTK/img/file-formats.pdf`
  - legacy serial format (*.vtk): ASCII 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

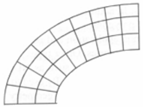
- **Image Data/Structured Points**: *.vti, points on a regular rectangular lattice, scalars or vectors at each point


(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

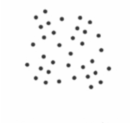

(b) Rectilinear Grid

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

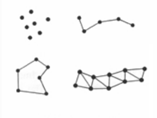

(c) Structured Grid

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

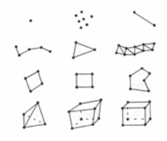
- **Particles/Unstructured Points**: *.particles


(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

- 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

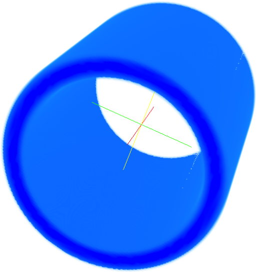
1. File: `data/vtk/legacy/volume.vtk`
   - simple example (Structured Points): $3 \times 4 \times 6$ dataset, one scalar field, one vector field
2. File: `data/vtk/legacy/density.vtk`
   - another simple example (Structured Grid): $2 \times 2 \times 2$ dataset, one scalar field
3. File: `data/vtk/legacy/cube.vtk`
   - 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

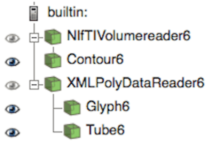
- A good option is to use XML VTK format with binary data and XML metadata, calling VTK library functions from C++ / Java / Python to write data
- Here is an example: `code/SGrid.cpp` and `code/Makefile`, generates the file `data/vtk/xml/halfCylinder.vts`

  This example shows how to 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

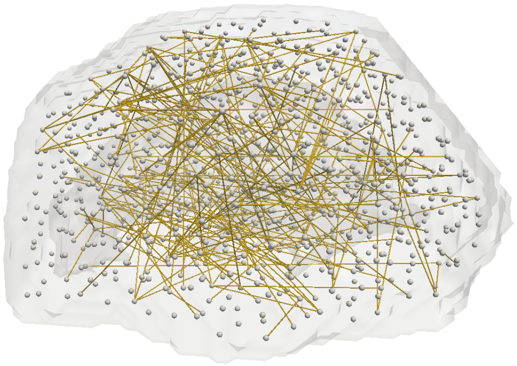
```
cd code
make SGrid
(on Linux: export LD_LIBRARY_PATH=/path/to/vtk/lib:$LD_LIBRARY_PATH)
(on a Mac: export DYLD_LIBRARY_PATH=$HOME/Documents/local/vtk/lib)
./SGrid
```

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

Think of ParaView as a GUI front end to VTK classes



builtin:
- NIfTIVolumereader6
  - Contour6
- XMLPolyDataReader6
  - Glyph6
  - Tube6



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

C example

```
icc writeNetCDF.c -o writeNetCDF -I/path/to/netcdf/include \
    -L/path/to/netcdf/lib -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 (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

(1) Can use OpenFOAM's built-in **foamToVTK** utility to convert data from OpenFOAM format to VTK – works with all versions of ParaView

(2) Can use OpenFOAM-supplied ParaView reader module libraries PV4FoamReader and vtkPV4Foam with precompiled ParaView 4.x.x through **paraFoam** launch script

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

(3) ParaView should recognize *.controlDict and *.foam files (**File** → **Open**) but these are not present in OpenFOAM's output; can create an empty *.foam file in the case directory and load it into ParaView – sometimes it works, sometimes it doesn't, the error can be traced to VTK/IO/Geometry/vtkOpenFOAMReader.cxx in ParaView's source code

(4) Can recompile ParaView with a third-party (not from OpenFOAM or ParaView) **vtkPOFFReader** plugin – crashes ParaView 4.1.0/4.3.1; officially the plugin was written for ParaView 3.10-3.14 – do not recommend this method

(5) Can 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 – do not recommend this method

Reading OpenFOAM: using foamToVTK utility

- Assuming you have OpenFOAM installed, e.g., on jasper

```
>>> log in to jasper
qsub -I -l nodes=1:ppn=1 -l walltime=0:30:00,pmem=2000mb
>>> wait for an interactive prompt

module load application/OpenFOAM/2.3.0
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 X

Step 1: install and configure paraFoam

```
brew install open-mpi
brew install scotch
brew install boost --without-single --with-mpi
brew install cgal
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.p
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
```

Problem: 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 (consider the case of a system-wide ParaView 4.3.1 installation on parallel.westgrid.ca; alternatively, you can use your own ParaView and/or use a different cluster)

```
>>> log in to parallel
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 on parallel (more on this later)

```
>>> 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!)

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 – 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**

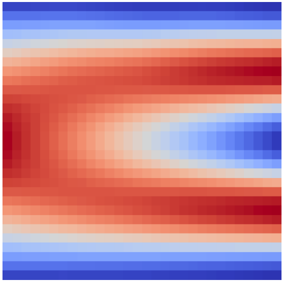
Last time I counted there were 146 filters. One can add new filters with python scripting

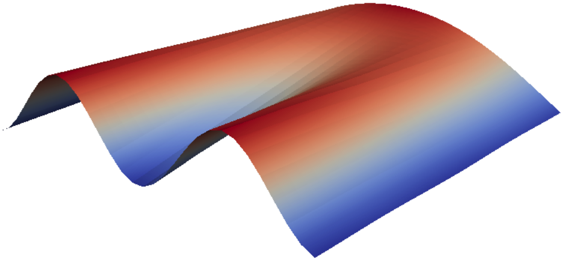
➠ Check out "Filters" in the menu; some are found in the toolbar

➠ List of filters `http://bit.ly/ZX5u2q` with documentation

Simple filter to visualize a 2D dataset in 3D

- Load the file `data/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.

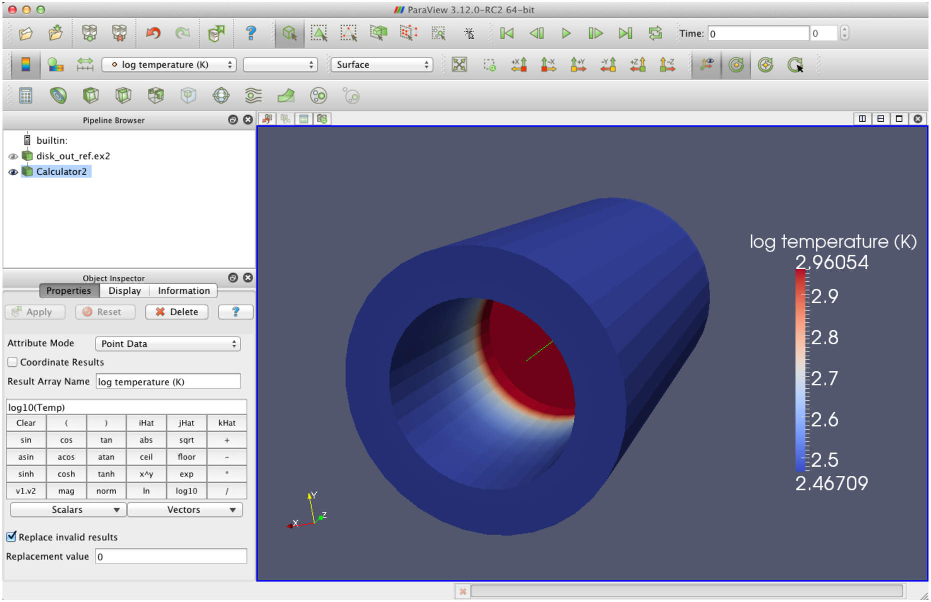
Calculator

- 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

Pipeline Browser
- builtin:
- disk_out_ref.ex2
- Calculator2

Object Inspector: Properties | Display | Information

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	/

☑ Replace invalid results

Replacement value: 0

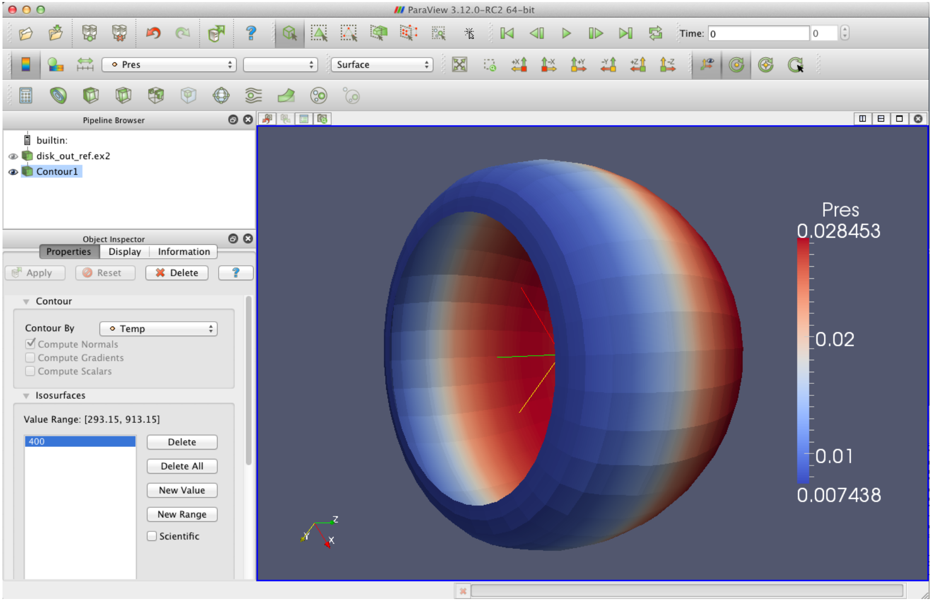
Contour

- 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

Pipeline Browser
- builtin:
- disk_out_ref.ex2
- Contour1

Object Inspector: Properties | Display | Information

Contour
- Contour By: Temp
- Compute Normals: ✓
- Compute Gradients
- Compute Scalars

Isosurfaces
- Value Range: [293.15, 913.15]
- 400

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 $\rightarrow$ Alphabetical $\rightarrow$ **Extract Surface**, and then add **Clip** to the result of **Extract Surface** $\Rightarrow$ 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
- To unlink, go to Tools -> Manage Links, delete the camera link in question

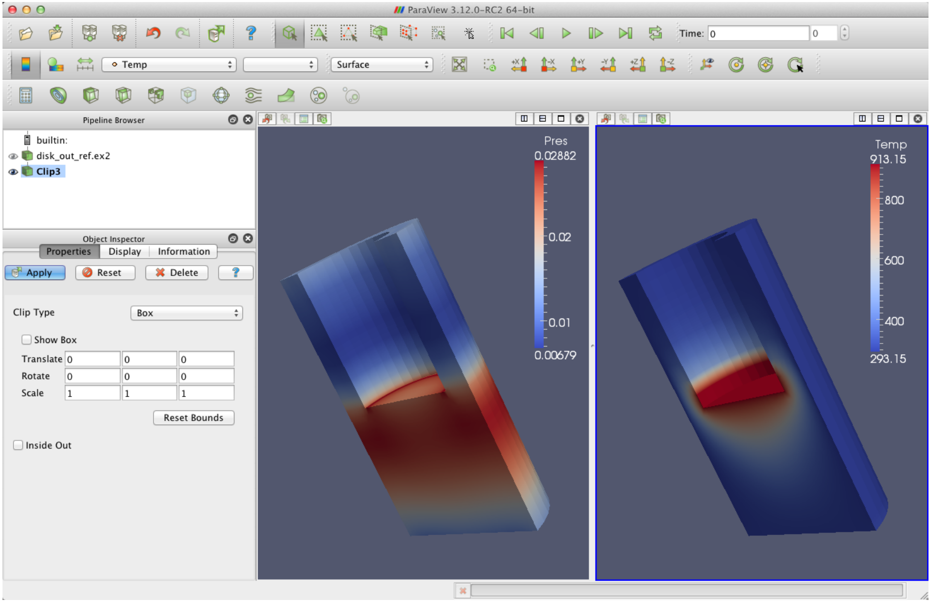
Multiview: several variables side by side

- 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
- To unlink, go to Tools -> Manage Links, delete the camera link in question

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
- To unlink, go to Tools -> Manage Links, delete the camera link in question

Multiview: several variables side by side



ParaView 3.12.0-RC2 64-bit

Time: 0 0

Temp Surface

Pipeline Browser

builtin:
disk_out_ref.ex2
Clip3

Object Inspector

Properties | Display | Information

Apply Reset Delete ?

Clip Type: Box

Show Box

Translate	0	0	0
Rotate	0	0	0
Scale	1	1	1

Reset Bounds

Inside Out

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, 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")

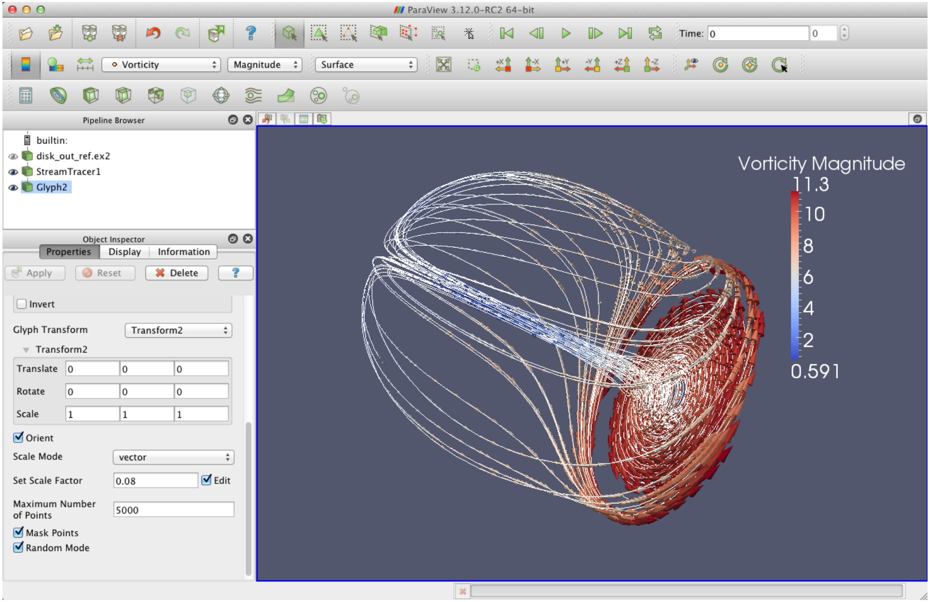
Vector visualization: streamlines and glyphs

- Start with the original data from `data/other/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/other/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



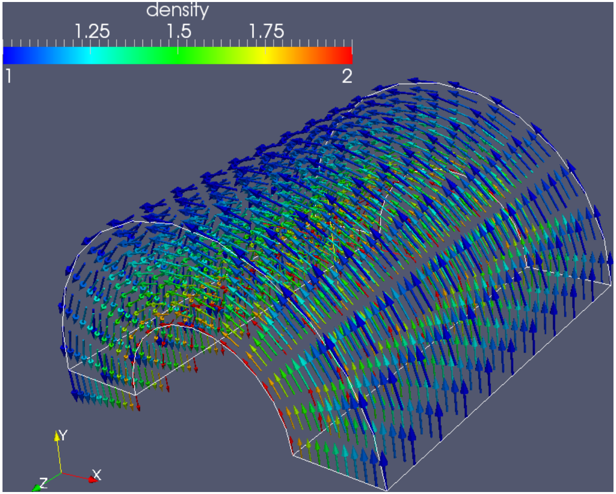
Invert: ☐
Glyph Transform: Transform2
Transform2

Translate	0	0	0
Rotate	0	0	0
Scale	1	1	1

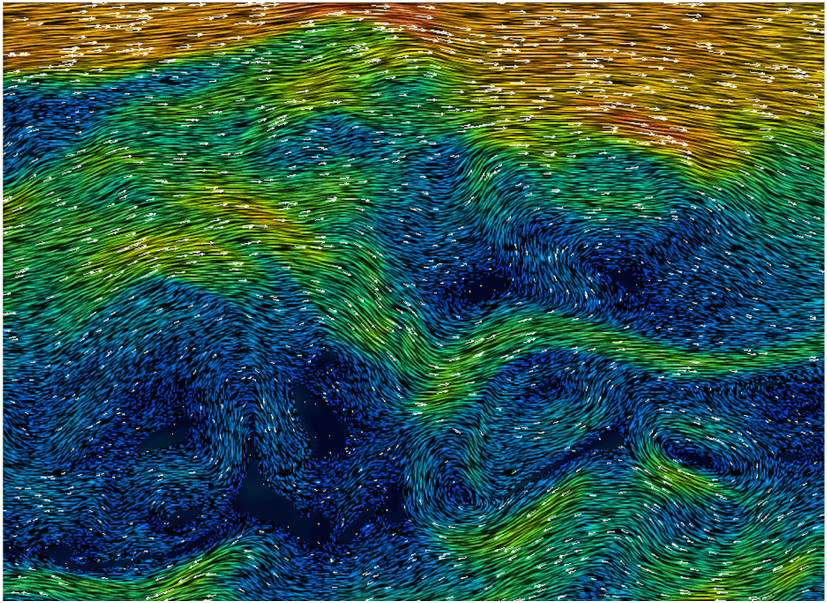
Orient: ✓
Scale Mode: vector
Set Scale Factor: 0.08 ✓ Edit
Maximum Number of Points: 5000
Mask Points: ✓
Random Mode: ✓

Exercise: vectors

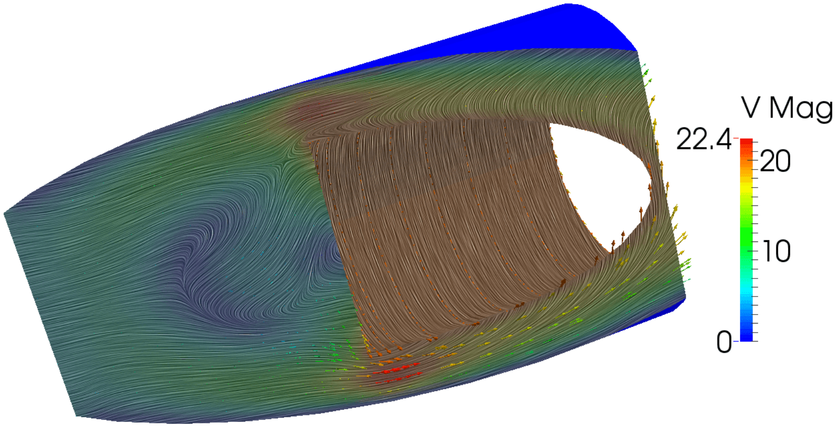
Load `data/vtk/xml/halfCylinder.vts` and display the velocity field as arrows, colouring them by density – try to reproduce the view below



Visualizing vectors with Line Integral Convolution



Line Integral Convolution in ParaView



- From Tools → Manage Plugins load *Surface LIC plugin*
- Load `data/other/disk_out_ref.ex2` or `data/vtk/xml/halfCylinder.vts`
- Apply a filter to see its interior (required step for `data/vtk/xml/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/other/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/other/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 B on disk, in binary 1600/3200 B in single/double precision

➢ tabulatedGrid.txt is 20,013 B on disk, in binary 4000/8000 B 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

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/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 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)

Batch scripting for automating visualization

- Why use scripting?
  - automate mundane or repetitive tasks, e.g., making mutiple frames for a movie
  - use ParaView on clusters without GUI
  - in-situ visualization
- Complete documentation at `http://www.paraview.org/Wiki/ParaView/Python_Scripting`
- 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

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)     # 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” `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)

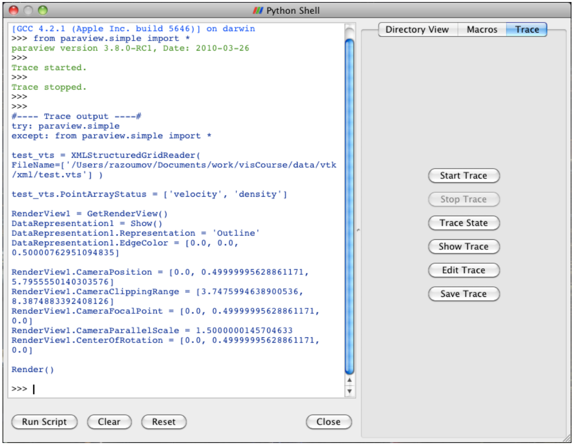
Trace tool

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]



```
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
>>> from paraview.simple import *
paraview version 3.8.0-RC1, Date: 2010-03-26
>>>
Trace started.
>>>
Trace stopped.
>>>
>>>
#---- Trace output ----#
try: paraview.simple
except: from paraview.simple import *

test_vts = XMLStructuredGridReader(
FileName=['/Users/razoumov/Documents/work/visCourse/data/vtk
/xml/test.vts'] )

test_vts.PointArrayStatus = ['velocity', 'density']

RenderView1 = GetRenderView()
DataRepresentation1 = Show()
DataRepresentation1.Representation = 'Outline'
DataRepresentation1.EdgeColor = [0.0, 0.0,
0.50000762951094835]

RenderView1.CameraPosition = [0.0, 0.49999995628861171,
5.7955550140303576]
RenderView1.CameraClippingRange = [3.7475994638900536,
8.3874883392408126]
RenderView1.CameraFocalPoint = [0.0, 0.49999995628861171,
0.0]
RenderView1.CameraParallelScale = 1.5000000145704633
RenderView1.CenterOfRotation = [0.0, 0.49999995628861171,
0.0]

Render()

>>> |
```

Passing information down the pipeline

`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()

Extracting data from VTK objects

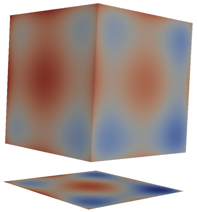
Do this from *Tools → Python Shell* or from *pvpython* (either shell will work)

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

dir = '/Users/razoumov/Documents/workshops/visualization/data/other/'
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 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 of the array
```

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/other/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 data (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

```
# code/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 = -30.
    newPoints.InsertPoint(i,x,y,z)   # insert a point
    pval = 0.
    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)
newCells = vtk.vtkCellArray()   # create 99x99 cells in the projection
for i in range(side-1):
    for j in range(side-1):
        newCells.InsertNextCell(4)   # insert a cell with four corners
        newCells.InsertCellPoint(i+j*side)
        newCells.InsertCellPoint(i+1+j*side)
        newCells.InsertCellPoint(i+side+j*side)
        newCells.InsertCellPoint(i+side+1+j*side)
output.SetCells(10,newCells)   # 10 refers to a VTK cell type
```

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 1: 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 (on presenter's laptop in `hidden/growth.mp4`)

To visualize a single frame of the movie:

1. load `data/other/sineEnvelope.nc` (discretized on a 100^3 grid)
2. apply Clip: origin=(49.5,15,49.5), normal=(0,-1,0)
3. apply Threshold keeping only data from 1.2 to 2

Use either of the two possible solutions:

1. Use animation view to animate Clip's origin_2 from 0 to 99
2. Use Start/Stop Trace to record the workflow, save the corresponding Python script, enclose it into a loop changing origin_2 from 0 to 99 and dumping a series of *.png screenshots, run it inside ParaView to produce 100 frames, merge them using a 3rd-party tool, e.g., `ffmpeg`

Exercise 2: scripting cone animation

- Write a script that for each $i = 0, \ldots, 50$
  - imports Sources $\rightarrow$ Cone
  - sets the cone's radius to $0.05 + 0.01 \times i$
  - saves a snapshot to a file called `coneXX.png`
- From cone00.png, cone01.png, ..., cone50.png make a movie
  (can use ParaView to stitch frames into a movie!)

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

```
for i in range(51):
    visualize and save the cone of radius 0.05+0.01*i
```

3. Run the script to produce 51 image files
4. In Linux/MacOSX can use ffmpeg to merge frames into a movie

```
ffmpeg -r 10 -i cone%02d.png -c:v libx264 -pix_fmt yuv420p \
     -vf "scale=trunc(iw/2)*2:trunc(ih/2)*2" cone.mp4
```

Visualizing remote data

So far we covered working with standalone ParaView on your desktop. If your dataset is on cluster.consortium.ca $\Rightarrow$ fundamentally there are five 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**
   your desktop $\xrightarrow{\text{VNC}}$ larger machine running ParaView
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)

For remote options (2) - (5) the setup details vary across the consortia

- render server can run with or without GPU rendering
- 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

Remote ParaView in Compute Canada consortia

- **WestGrid**: GPU rendering on parallel.westgrid.ca ← cluster at UofCalgary with 60 twelve-core GPU nodes with 24GB memory and 3 NVIDIA Tesla M2070s GPUs each, used for GP-GPU computing and visualization – some of these nodes are open to the interactive queue
  - VNC to a GPU node `http://bit.ly/remotevnc`
  - client-server connection to a GPU node `http://bit.ly/clientserver`
  - users need to request a separate account to access Parallel cluster

  Depending on demand, we can also configure CPU rendering on bugaboo and other major clusters
- **SciNet**: CPU rendering on SciNet's GPC nodes `http://bit.ly/1p60dx0`
- **Calcul Québec**: CPU rendering on colosse.calculquebec.ca, GPU rendering on guillimin.clumeq.ca
- **SHARCNET**: standalone ParaView on a visualization server (running multiple Linux containers, accessible via VNC, 64GB memory), all mounting global /home and /work via sshfs

Setup details vary across the consortia!

Remote ParaView via VNC on WestGrid

full details at `http://bit.ly/remotevnc`

1. Install TurboVNC from `http://www.turbovnc.org` on your desktop
2. Log in to parallel.westgrid.ca and run the command `vncpasswd`, at the prompt set a password for your VNC server (you'll use it below)
3. **Submit an interactive job** to the cluster
   ```
   qsub -q interactive -I -l nodes=1:ppn=1:gpus=1,walltime=1:00:00
   ```
   When the job starts, it'll return a prompt on the assigned compute node.
4. On the compute node **start the vncserver**: `vncserver -geometry 1024x768`
   It'll produce *"New 'X' desktop is cn0553:1"*, where the syntax is `nodeName:displayNumber`
5. On your desktop **set up ssh forwarding** to the VNC port:
   ```
   ssh username@parallel.westgrid.ca -L 5901:cn0553:5901
   ```
   Here the port number is 5901 = 5900 (VNC's default) + displayNumber
6. **Start TurboVNC vncviewer** on your desktop, enter `localhost:displayNumber`, e.g. `localhost:1`, and then enter the password from item 2 above
7. A remote Gnome desktop will appear inside a VNC window on your desktop
8. Inside this desktop start a terminal, use it to **start ParaView with a VirtualGL wrapper**
   ```
   vglrun /global/software/ParaView/ParaView-4.3.1-Linux-64bit/bin/paraview
   ```

Remote client-server ParaView on WestGrid

full details at `http://bit.ly/clientserver`

1. On parallel.westgrid.ca **submit an interactive job** to allocate compute resources on the cluster, e.g. a single core:
   `qsub -q interactive -I -l nodes=1:ppn=1:gpus=1,walltime=1:00:00`
   When the job starts, it'll return a prompt on the assigned compute node.
2. On the compute node **start the ParaView server**:
   `/global/scratch/razoumov/ParaView-4.3.1-Linux-64bit/bin/pvserver`
   If the default port is busy, optionally might need to set the port manually with `-server-port=11112`. The `pvserver` command will return something like: *"Connection URL: cs://cn0553:11111"* and *"Accepting connection(s): cn0553:11111"*
3. On your desktop **set up ssh forwarding** to the ParaView server port:
   `ssh username@parallel.westgrid.ca -L 11111:cn0553:11111`
4. On your desktop **start ParaView** and **edit its connection properties** under *File - Connect - Add Server* (name = parallel.westgrid.ca, server type = Client/Server, host = localhost, port = 11111), click *Configure*, choose *Manual* and click *Save*, then select the server from the list and click on *Connect*

Note 1: ParaView's client and server must have matching versions!

Note 2: the example above is serial, since it uses a precompiled serial ParaView server

Parallel remote client-server ParaView on WestGrid

based on demand, this setup will propagate to all GPU-less clusters

1. On bugaboo start either a serial job with a ParaView server

```
qsub -I -l nodes=1:ppn=1,pmem=2000mb,walltime=00:30:00 # wait for job to start
/global/scratch/razoumov/mesa43/bin/pvserver # wait for "Accepting
                                                connection(s): b402:11111"
```

or a parallel job with a ParaView server

```
qsub -I -l nodes=2:ppn=8,pmem=1500mb,walltime=00:30:00 # wait for job to start
mpiexec /global/scratch/razoumov/mesa43/bin/pvserver # wait for "Accepting
                                                        connection(s): b402:11111"
```

2. On your desktop

```
ssh username@bugaboo.westgrid.ca -L 11111:b402:11111  # set up ssh forwarding
start ParaView
File -> Connect -> Add Server: name = bugaboo.westgrid.ca,
     server type = Client/Server, host = localhost, port = 11111
click Configure, choose Manual and click Save
select the server from the list and click Connect
```

3. To check if rendering is parallel, use a special filter

```
Sources -> Sphere, click Apply
Filter -> Alphabetical -> Process ID Scalars, click Apply
```

or simply bring up View -> Memory Inspector.

Batch script GPU visualization on WestGrid

Method 1: via an interactive job

on parallel.westgrid.ca
```
qsub -q interactive -I -l nodes=1:ppn=1:gpus=1,walltime=0:30:00
          # wait for an interactive shell
firstgpu=$( head -n 1 "$PBS_GPUFILE" )
gpuindex=${firstgpu:(-1)}
export DISPLAY=:0.$gpuindex
/global/software/ParaView/ParaView-4.3.1-Linux-64bit/bin/pvbatch batch.py
```

- `batch.py` would be developed in an interactive debugging session
- It would read data from disk, do rendering, and write an image (or many) to disk
- In my experience, complex ParaView Python scripts are not exactly portable across architectures $\Rightarrow$ will likely need to debug in place

Batch script GPU visualization on WestGrid (cont.)

Method 2: via a batch job (using the same Python script)

on parallel.westgrid.ca

```
qsub -q interactive ./visualization.sh    # -q needed to send to an interactive
                                          # node where we have GPUs
```

visualization.sh script

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -q gpu
#PBS -l nodes=1:ppn=1:gpus=1
#PBS -l pmem=2000mb
#PBS -l walltime=00:30:00
#PBS -m b
#PBS -M yourEmailAddress
cd $PBS_O_WORKDIR
firstgpu=$( head -n 1 "$PBS_GPUFILE" )
gpuindex=${firstgpu:(-1)}
export DISPLAY=:0.$gpuindex
/global/software/ParaView/ParaView-4.3.1-Linux-64bit/bin/pvbatch batch.py
```

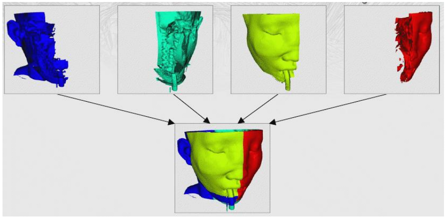
Data partitioning

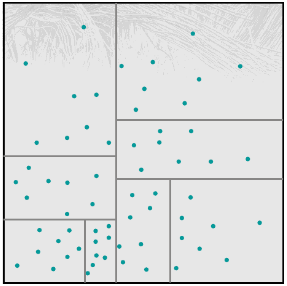
Scalable parallel distributed rendering – load balancing is handled automatically by ParaView for structured data:

- 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 $\leftarrow$ *from ParaView documentation*
- Unstructured data (Unstructured Points, Polygonal Data, Unstructured Grid): one processor core per 0.5-1 million cells $\leftarrow$ *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 $\sim$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)

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 WestGrid
  - email support@westgrid.ca
  - email me alex.razoumov@westgrid.ca
  - visualization quickstart guide `http://bit.ly/quickstartviz`
  - remote visualization `http://bit.ly/remoteviz`
  - batch rendering `http://bit.ly/batchviz`
- 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)
  - Compute Canada visualization showcase `http://bit.ly/cctopviz`