Using visualization tools on SHARCNET

Alex Razoumov
razoumov@sharcnet.ca

Summer School 2010

a copy of these slides, data and sample C++, Fortran, Python codes can be found in /home/razoumov/visualization.tar.gz on SHARCNET

Visualization packages on SHARCNET

Visualization	
GRAPHVIZ	Represent structural information as diagrams of abstract graphs and networks
HYPERMESH	Altair Hyperworks Suite (commercial, hyperworks group, limited access)
ICEMCFD	ANSYS ICEM CFD Meshing Software (commercial, fluent group)
ITK	National Library of Medicine Insight Segmentation and Registration Toolkit
OpenDX	Visualization of Scientific, Engineering and Analytical Data
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

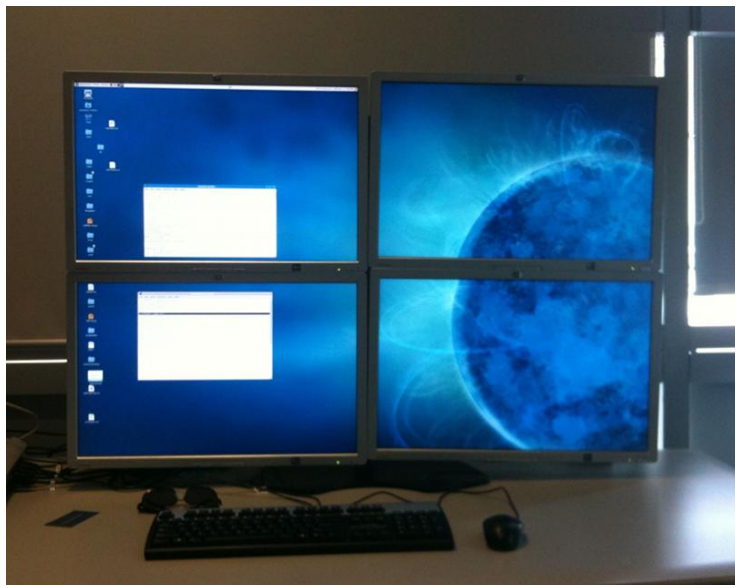
http://www.sharcnet.ca/my/software

- open-source, multi-platform, and general-purpose:
  - visualize scalar and vector fields
  - structured and unstructured meshes in 2D and 3D
  - ability to handle very large datasets (GB-TB)
  - iteractive manipulation

1. OpenDX 4.4.4 - installed on vizN-site
2. VisIT 1.7 - installed on vizN-site
3. ParaView 3.6.1 - installed on rainbow, vizN-site

Running on vizN-site vs. your desktop

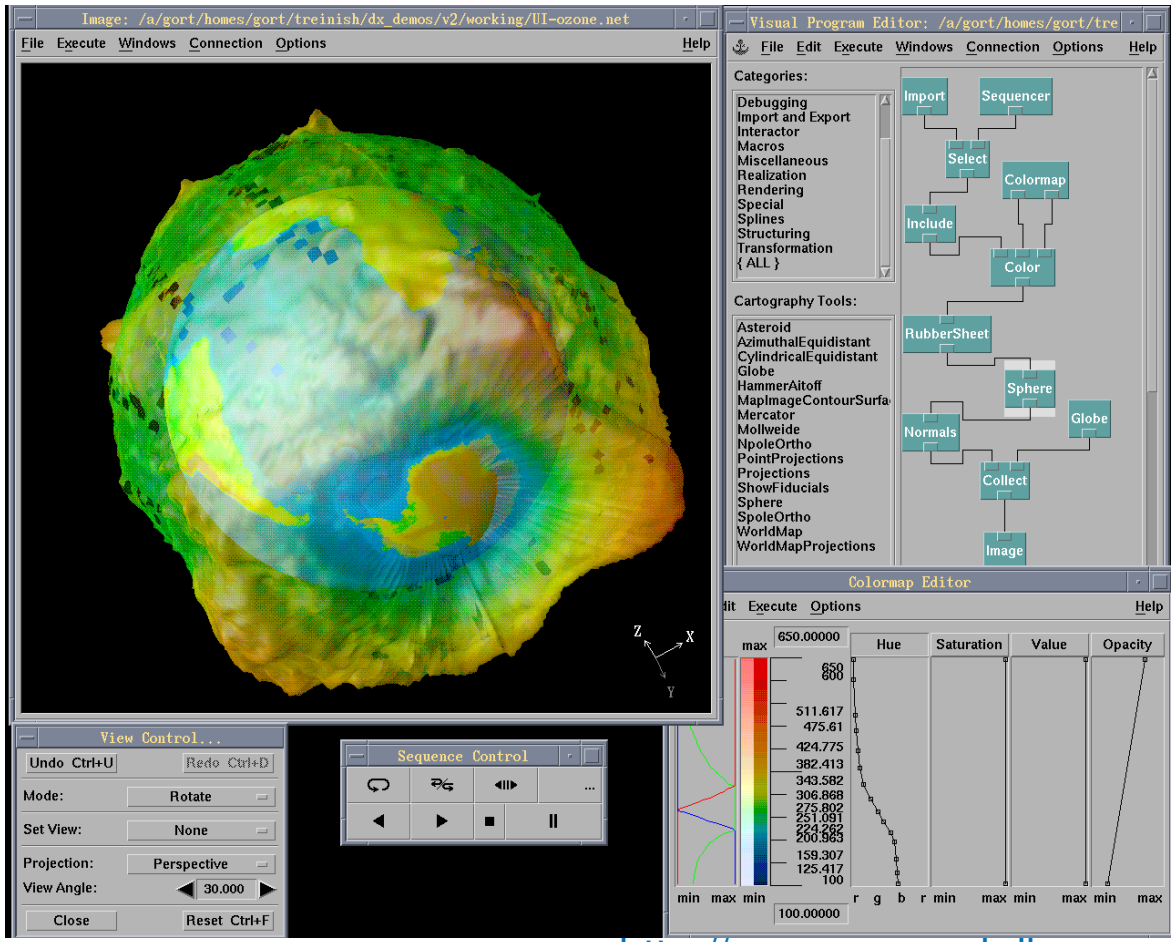
- open-source $\Rightarrow$ can install on your desktop (Linux / Mac / Windows)
- most of our partner universities have visualization workstations: fast machines with 8-16 GB memory, dual- or quad-core displays, all sitting on a dedicated SHARCNET network (1 Gbit/s)



- can connect remotely to workstations from your desktop via
  - X11: *ssh -X*, Cygwin, etc.
  - NX / openNX (X11 with compression and smart caching)

OpenDX = Open Visualization Data Explorer

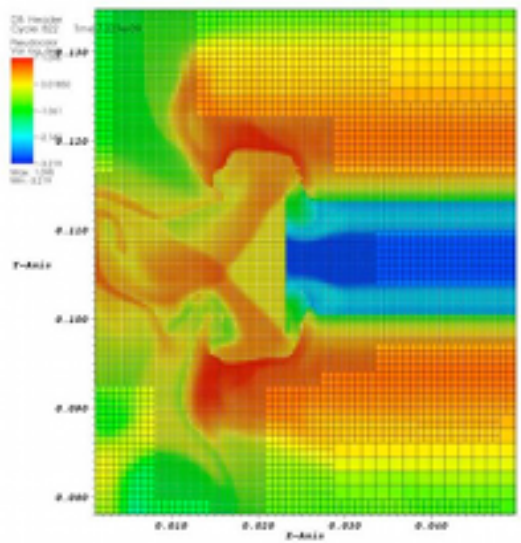
- http://www.opendx.org
- started in 1991 at IBM, released as open source in 1999, latest v4.4.4 (2006-Jan)
- Windows/Linux binaries free, need to compile on Mac (available in darwinports/fink)
- based on the Motif widget toolkit on top of X11
- interactive Visual Program Editor

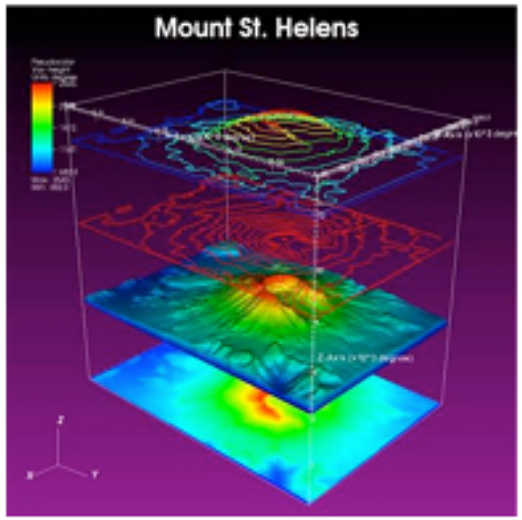


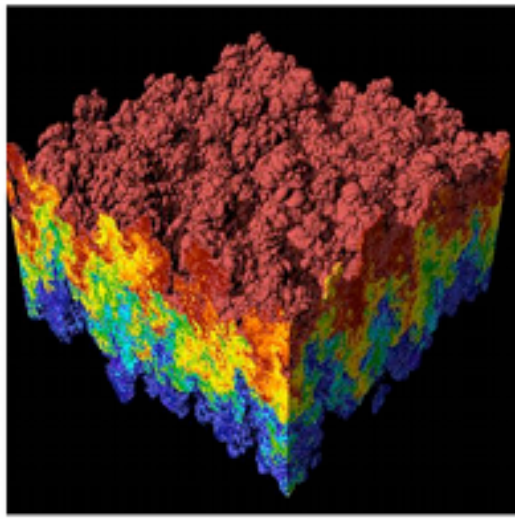
http://www.research.ibm.com

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
- v2.0.0 beta (and earlier) available for Linux/Mac/Windows
- over 60 visualization features (contour, mesh, slice, volume, molecule, ...)
- interfaces with C++, Python, and Java



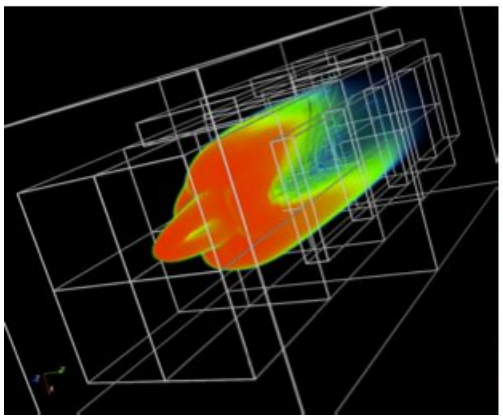


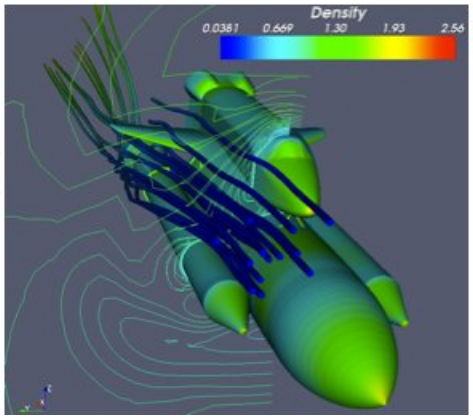


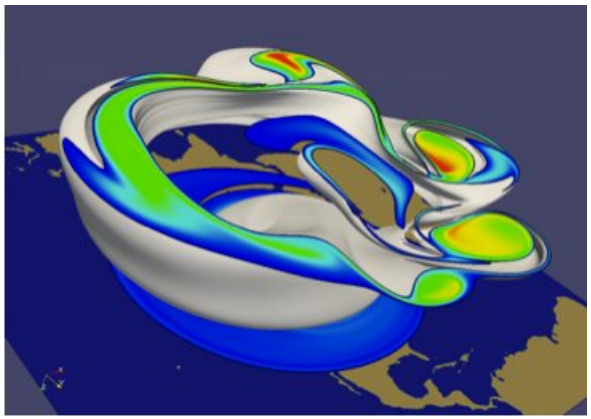
Lawrence Livermore National Laboratory

ParaView

- http://www.paraview.org
- developed jointly by Sandia National Labs + Los Alamos National Lab + Kitware Inc.
- latest binary release 3.8.0 (2010-Apr), available for Linux/Mac/Windows
- developed for extremely large datasets on distributed memory machines
- both interactive and Python scripting
- 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?

- seems to be a lot more interest in ParaView from SHARCNET users
- in my experience, ParaView on a desktop is a little bit more enjoyable than VisIT
- ParaView server is installed on rainbow (our visualization cluster)

- not that I discourage you from using VisIT

Online resources

ParaView on SHARCNET http://www.sharcnet.ca/my/software/show/67

help page http://www.paraview.org/OnlineHelpCurrent

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

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

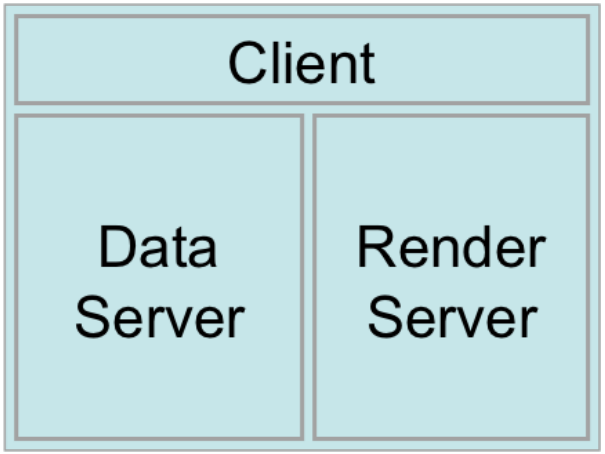
VTK tutorials http://www.itk.org/Wiki/VTK/Tutorials

ParaView architecture

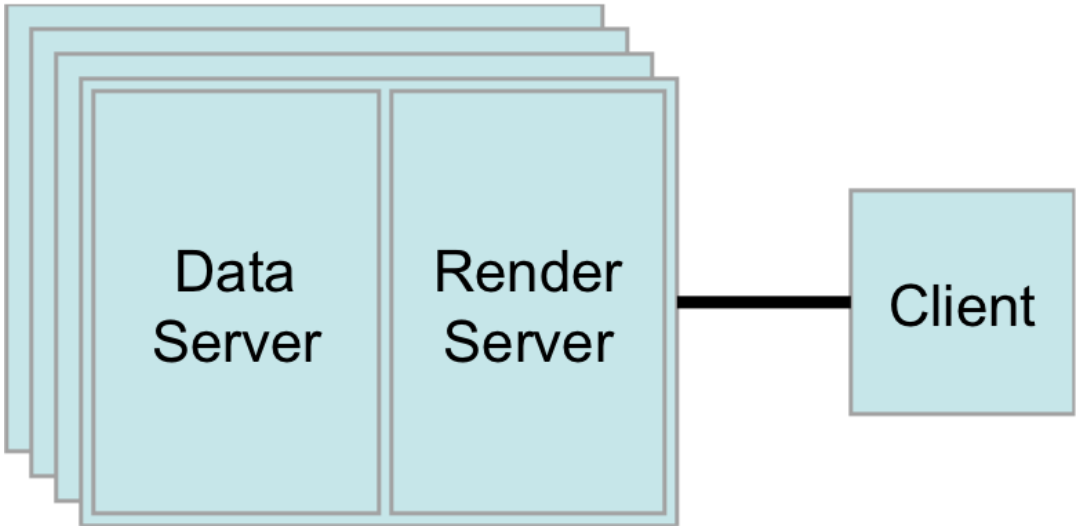
Three logical units of ParaView – these units can be embedded in the same application on the same computer, but can also run on different machines:

- **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

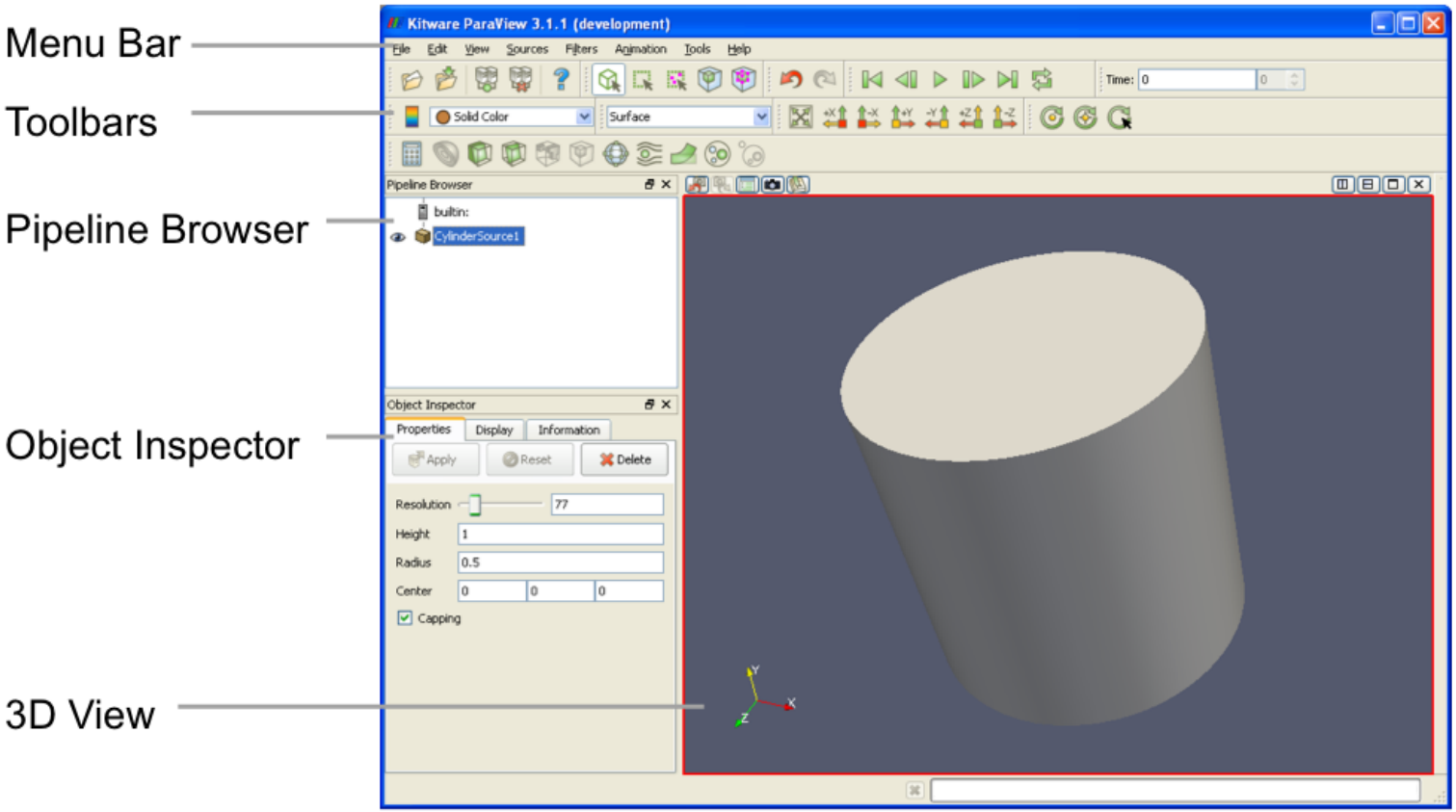


standalone mode



client-server mode: pvserver on a parallel machine

User interface



- Pipeline Browser: reading and filtering of data
- Object Inspector: view and change parameters of the current pipeline object (properties - display - information)
- View window

find the following in the toolbar: “Connect”, “Disconnect”, “Toggle Color Legend Visibility”, “Edit Colour Map”, “Rescale to Data Range”

Data sources

- generate data with a *Source* object
- read data from a file

AVS UCD Binary/ASCII Files(*.inp)
BYU Files(*.g)
Case file for restarted CTH outputs(*.spcth-timeseries
Comma-separated-values(*.csv)
Cosmology files(*.cosmo *.gadget2)
Digital Elevation Map Files(*.dem)
EnSight Files(*.case *.CASE *.Case)
EnSight Master Server Files(*.sos *.SOS)
Enzo Files(*.boundary *.hierarchy)
ExodusII(*.g *.e *.ex2 *.ex2v2 *.exo *.gen *.exoII *.0 *
Flash Files(*.Flash *.flash)
Fluent Case Files(*.cas)
Gaussian Cube Files(*.cube)
LSDyna(*.k *.lsdyna *.d3plot d3plot)
Legacy VTK Files (partitioned)(*.pvtk)
Legacy VTK files(*.vtk)
MFIX Unstructured Grid Files(*.RES)
Meta Image Data Files(*.mhd *.mha)
Metafile for restarted exodus outputs(*.ex-timeseries
Nrrd Raw Image Files(*.nrrd *.nhdr)
Ocean Netcdf Files(*.pop.ncdf *.pop.nc)
PLOT3D Files(*.xyz)
PLY Polygonal File Format(*.ply)
PNG Image Files(*.png)
POP Ocean Files(*.pop)
ParaView Data Files(*.pvd)
Phasta Files(*.pht)
Protein Data Bank Files(*.pdb)

Raw (binary) Files(*.raw)
SESAME(*.sesame)
SLAC Mesh Files(*.ncdf *.nc)
SLAC Particle Files(*.ncdf *.netcdf)
SpyPlot CTH dataset(*.spcth *.0)
Stereo Lithography(*.stl)
TIFF Image Files(*.tif *.tiff)
Tecplot Files(*.tec *.TEC *.Tec *.tp *.TP)
VPIC Files(*.vpc)
VRML 2 Files(*.wrl *.vrml)
VTK Hierarchical Box Data Files(*.vthb)
VTK ImageData Files (partitioned)(*.pvti)
VTK ImageData Files(*.vti)
VTK MultiBlock Data Files(*.vtm *.vtmb)
VTK Particle Files(*.particles)
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)
Wavefront OBJ Files(*.obj)
WindBlade Data(*.wind)
XMol Molecule Files(*.xyz)
Xdmf Reader(*.xmf *.xdmf)
netCDF Files(*.ncdf *.nc)

somewhat incomplete list of file readers:
http://paraview.org/OnlineHelpCurrent/ParaViewReaders.html

Example: reading raw (binary) data

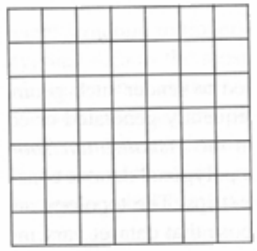
1. file: data/raw/[S29-2D.raw,S29-3D.raw]
2. describe the dataset in properties:
   - Data Scalar Type = float
   - Data Byte Order = Little Endian
   - File Dimensionality = 2 or 3
   - Data Extent: 1 to 128 in each dimension
   - Scalar Array Name = density
3. for 2D in display set:
   - Rescale to Data Range
   - Edit Color Map
4. for 3D try different views: outline, points, wireframe, ...
5. depending on the view, can set:
   - Rescale to Data Range
   - Edit Color Map
6. try saving data as paraview data type (*.pvd), deleting the object, and reading back from *.pvd – file 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 formats, extension depends on data type): XML tags + ASCII/binary/compressed data

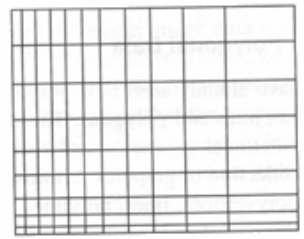
VTK 3D data: 6 major dataset types

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



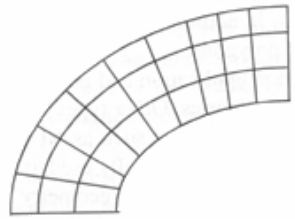
(a) Image Data

- **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

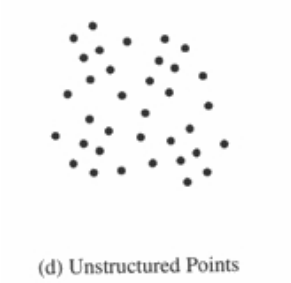
- **Structured Grid**: *.vts, regular topology and irregular geometry, need to indicate coordinates of each point



(c) Structured Grid

VTK 3D data: 6 major dataset types

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

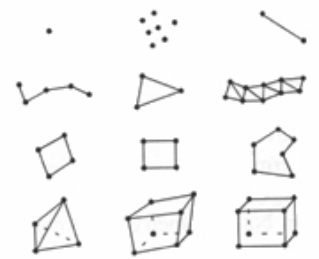


(d) Unstructured Points

- **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 ($|\vec{n}| = 1$) 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

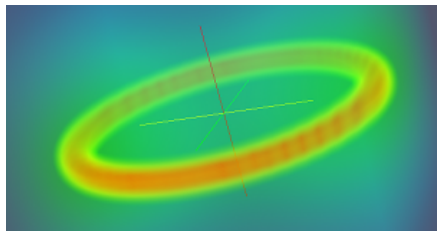
Example: reading legacy VTK

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.vtp
   - 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 colors, associated with the point scalars, is also defined.

Exercise: visualizing 3D data with legacy VTK

- if you have your own 3D scalar data $\Rightarrow$ try writing it from C/C++/Fortran/etc as a VTK ASCII legacy file (*.vtk) in structured points format using *volume.vtk* as a template

- if not $\Rightarrow$ try visualizing a 3D ring function $f(x,y,z) = e^{-[(z-0.5)^2+(r-0.4)^2]^{0.5}}$, where $r = \left[(x-0.5)^2+(y-0.5)^2\right]^{1/2}$, or some other function of your choice, inside the unit cube



  - in either case, try sampling it at some moderate resolution, e.g. 30^3, since we are dealing with ASCII

Writing XML VTK from C++

You will find VTK libraries for C++, Python, Tcl in saw.sharcnet.ca:/work/razoumov/vtk/lib/vtk-5.6

Please add the library path to your environment on saw:

- if default shell is bash, add to ~/.bashrc

  setenv LD_LIBRARY_PATH /work/razoumov/vtk/lib/vtk-5.6:$LD_LIBRARY_PATH
- if default shell is tcsh, add to ~/.tcshrc

  export LD_LIBRARY_PATH=/work/razoumov/vtk/lib/vtk-5.6:$LD_LIBRARY_PATH

Study SGrid.C, use it to generate data, and visualize it with ParaView

```
cd code
make SGrid
./SGrid
move halfCylinder.vts to lab PC
```

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.

netCDF and HDF5

two other common scientific data formats: netCDF and HDF5

netCDF supported natively

ParaView does not support HDF5 format natively, however it supports a container format XDMF which uses HDF5 for actual data

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

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
- take a look at the following examples: vase_1comp.vti, disk_out_ref.ex2 (ExodusII format)

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

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. These filter connections form a **visualization pipeline**.

Over 80 filters are currently available.

Check out “Filters” in the menu; some are found in the toolbar.

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. disk_out_ref.ex2 (load temperature, velocity, pressure), and try to visualize individual variables: Pres, Temp, V
- in “Object Inspector” > “Display” use “Rescale to Data Range” and “Edit Color Map ...” to see the data range
- now try to apply **Calculator** filter to display the following variables: Pres/Temp, log10(Temp), mag(V) - pay attention to the data range
- dropdown menus “Scalars” and “Vectors” will help you enter variables
- “?” button is surprisingly useful
- you change visibility of each object in the pipeline browser by clicking on the eyeball icon next to it

Contour

- delete **Calculator** from the pipeline browser, load **Contour**
- create an isosurface where the temperature is 400 K
- try different views (Surface, Wireframe, ...)

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 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 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)

Multiview: several variables side by side

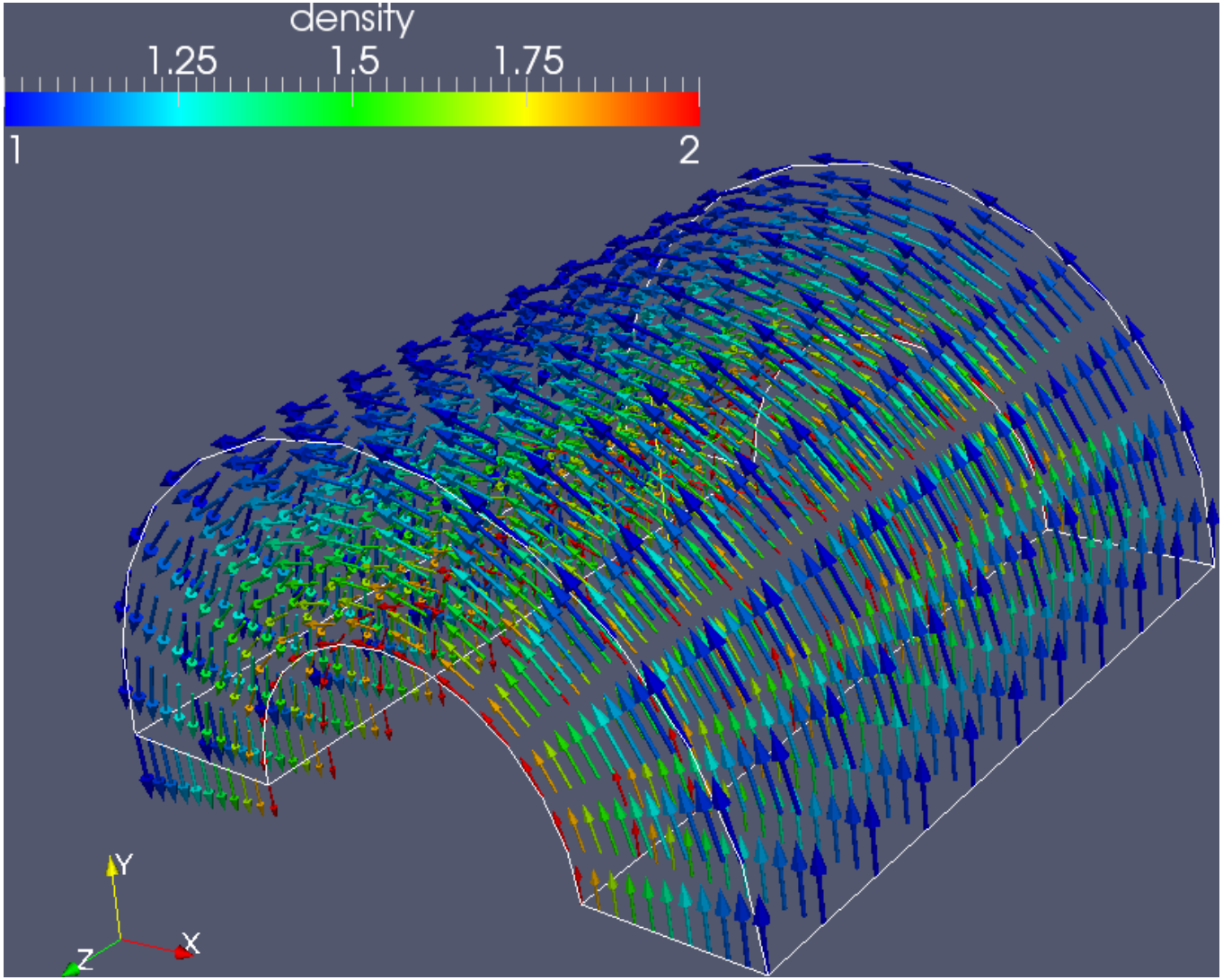
- start with the original data from disk_out_ref.ex2, load all variables
- add the **Clip** filter, uncheck “Show Plane” in the object inspector, click “Apply”
- color the surface by pressure by changing the variable chooser in the toolbar from “Solid Color” to “Pres”
- press “Split horizontal”, make 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
- color the surface by temperature by changing the toolbar variable chooser from “Solid Color” to “Temp”
- can reset (fit/reposition) the view in either column by clicking “Reset”
- 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 Color Legend Visibility”, try moving colourbars around

Vector visualization: streamlines and glyphs

- start with the original data from disk_out_ref.ex2, load velocity
- 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”
  - color the glyphs with the “Temp” variable
- now try displaying “V” glyphs directly from data, can colour them using different variables (“Temp”, “V”)

Exercise: vectors

load “halfCylinder.vts” and display the velocity field as arrows, colouring them by density



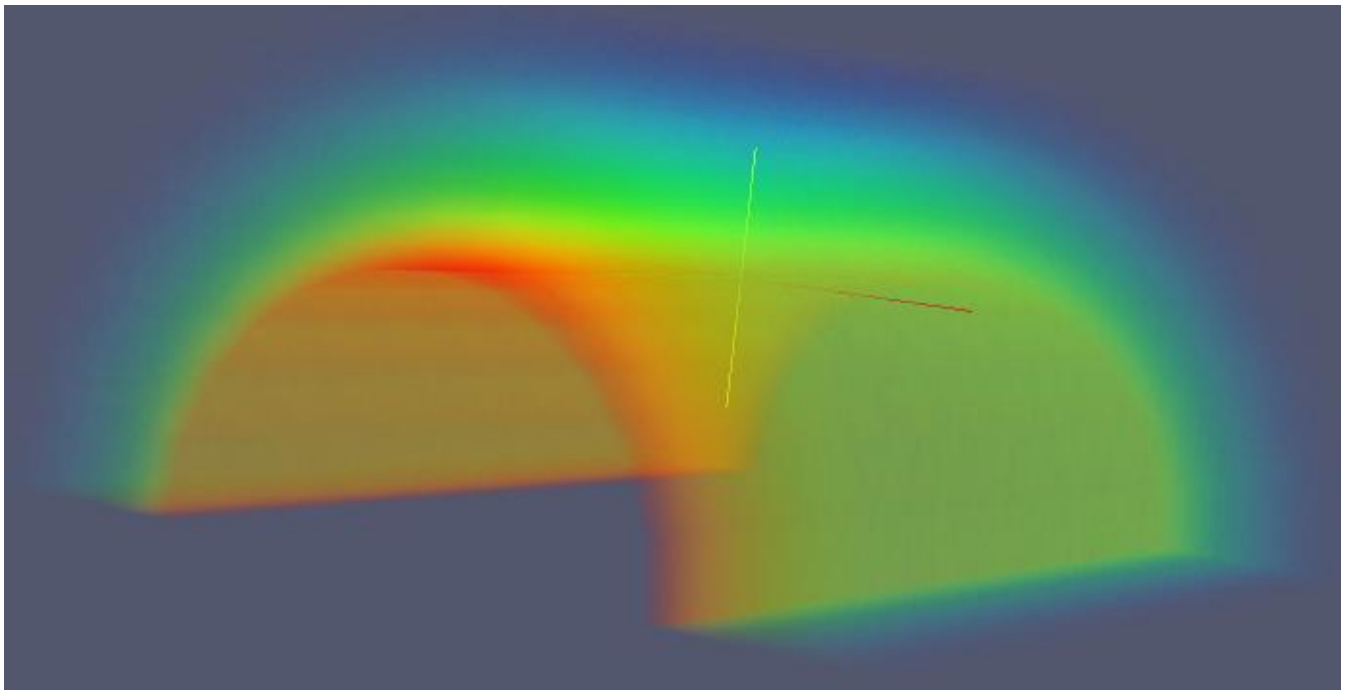
Word of caution

- 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

Volume Rendering

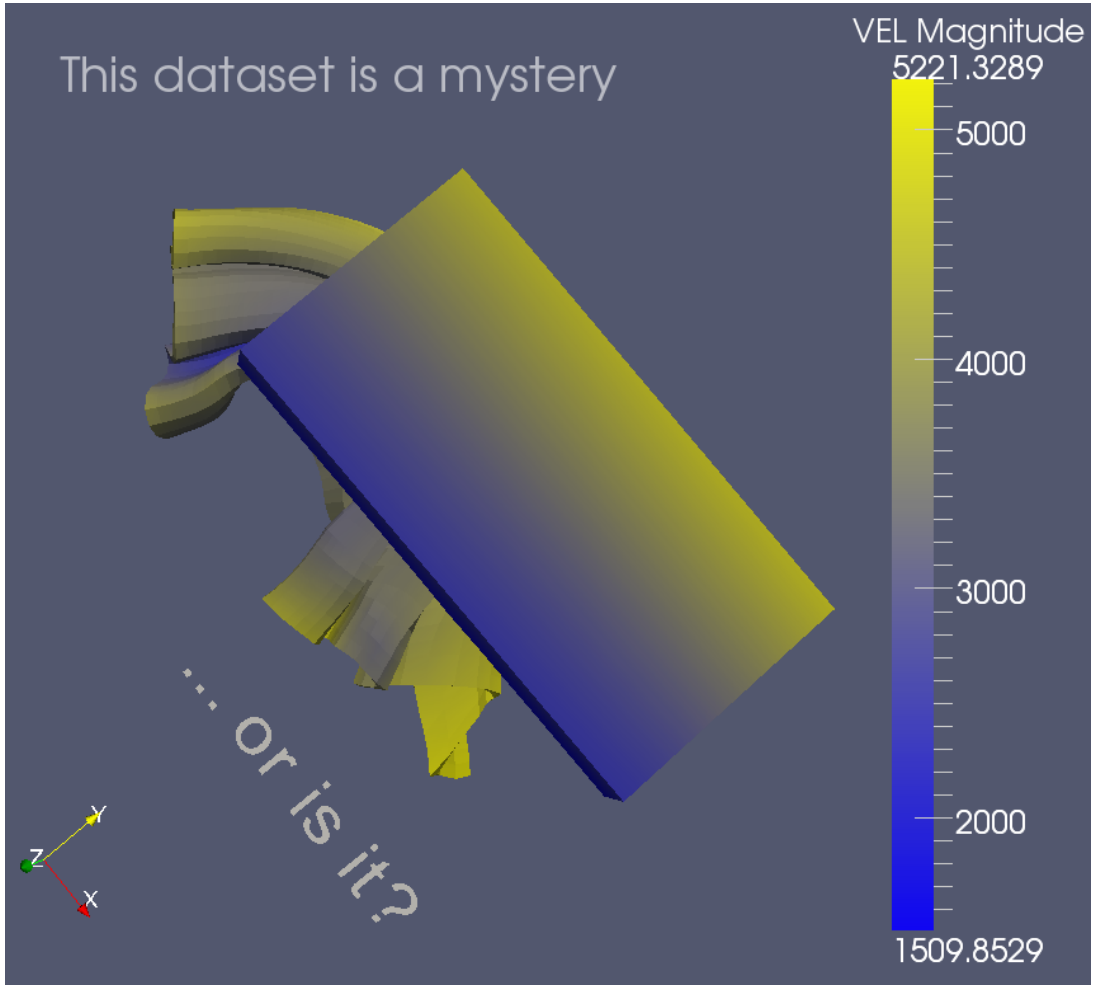
How can we do volumetric rendering for less structured datasets? “Volume” - available only for Structured Points (regularly spaced grid).

What about Structured Grid – try loading halfCylinder.vts and doing volumetric rendering of density.



Text Annotation

- select Sources → Text, type in the text edit box of the object inspector, hit “Apply”, edit Display properties
- Sources → 3D Text



Batch scripting for automating visualization

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

[vizN-site:/usr/bin/] pvpython will give you a python shell connected to a ParaView server (local or remote)

[vizN-site:/usr/bin/] pvbatch pythonScript is a serial (on some machines parallel) application using local server

First script

bring up Tools → Python Shell

"Run Script" displaySphere.py

```
from paraview.simple import * # load paraview.simple module
sphere = Sphere() # create a data object
print sphere.ThetaResolution # print one of the properties of the sphere
sphere.ThetaResolution = 16
Show() # turn on visibility of the object in the view
        (i.e. create a representation object which is a view of the data)
Render()
```

can always get help from the command line

```
help(paraview.simple)
help(sphere)
help(Sphere)
```

Using filters

"Run Script" displayWireframe.py

```
from paraview.simple import *

sphere = Sphere(ThetaResolution=36, PhiResolution=18) # create data object

wireframe = ExtractEdges(Input=sphere) # create another data object

Show() # create representation object (turn on the view of the data)

Render()
```

now try replacing Show() with Show(sphere)

Reading from files

"Run Script" readDiskOutRef.py

```
from paraview.simple import *
reader = ExodusIIReader(FileName='/Users/.../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:

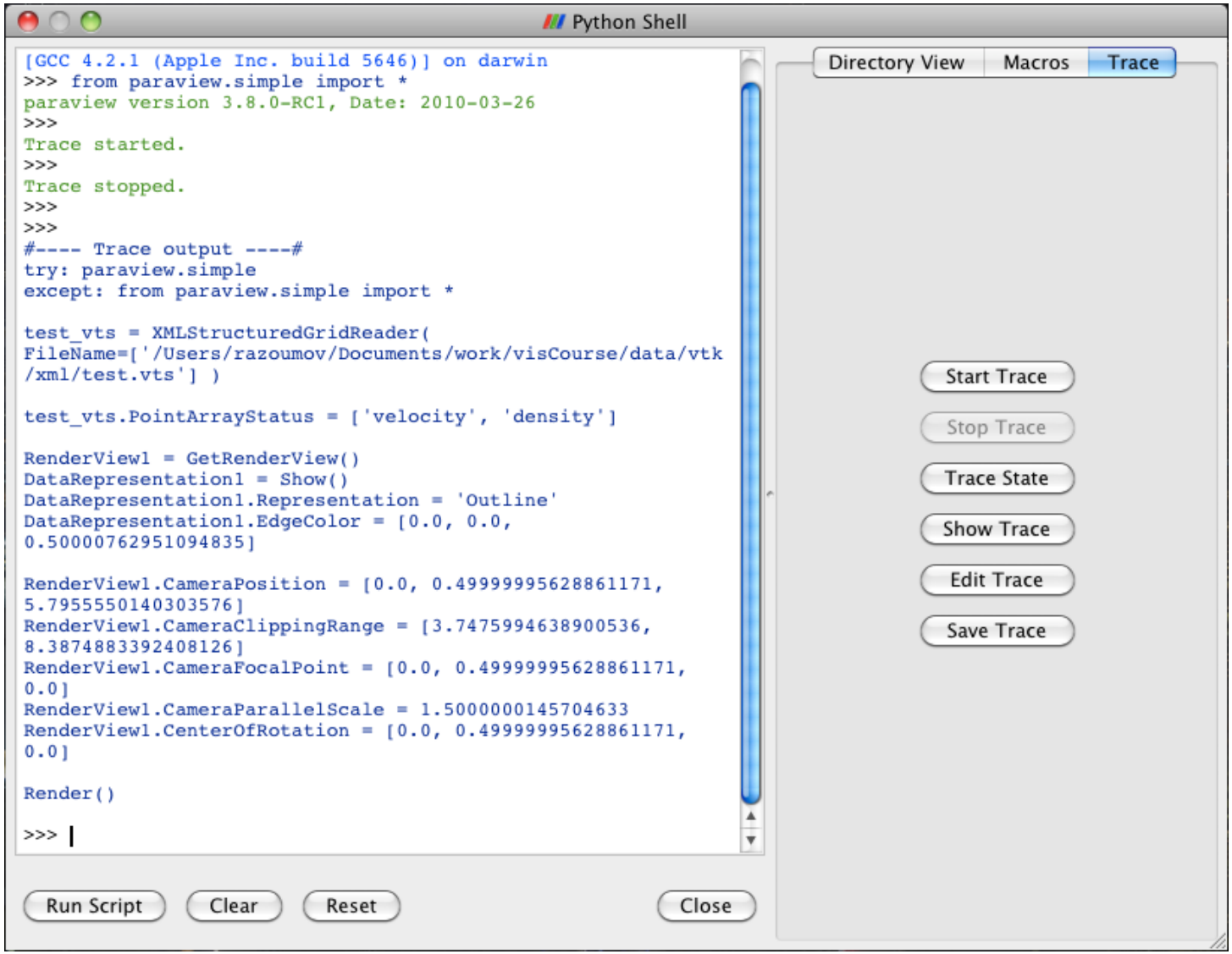
```
reader = OpenDataFile('/Users/.../vtk/xml/halfCylinder.vts')
```

Querying field attributes

“Run Script” readStructuredGrid.py

```
from paraview.simple import *
reader = OpenDataFile('/Users/.../vtk/xml/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



can generate Python code from GUI operations,
can start/stop trace at any time

More complex example generated via trace

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

```
from paraview.simple import *

test_vts = XMLStructuredGridReader( FileName=['/Users/.../vtk/xml/halfCylinder.vts'] )

DataRepresentation1 = Show() # turn on visibility of the object
DataRepresentation1.Representation = 'Outline'
DataRepresentation1.EdgeColor = [0.0, 0.0, 0.5]

RenderView = GetRenderView() # set camera properties for current view
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" ) # set vector properties
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(...)
a1_density_PVLookupTable = GetLookupTableForArray(...)
DataRepresentation2.LookupTable = a1_density_PVLookupTable
WriteImage('/Users/.../output.png')

Render()
```

try generating similar script from GUI and run/modify it

Working with pipeline objects

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

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

```
from paraview.simple import *
test_vts = XMLStructuredGridReader(FileName=['halfCylinder.vts'])
DataRepresentation = Show()

DataRepresentation.Representation = 'Surface'
DataRepresentation.DiffuseColor = [0, 0, 1]
DataRepresentation.SpecularColor = [1, 1, 1]
DataRepresentation.SpecularPower = 200
DataRepresentation.Specular = 1
Render()
```

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

Remote serial shell execution

Theoretically, can *ssh -Y username@vizN-site.sharcnet.ca* and either

- start Python shell *pvpython* and type commands by hand
- *pvbatch remote.py*

```
from paraview.simple import *
sphere = Sphere()
print sphere.ThetaResolution
sphere.ThetaResolution = 16
Show()
WriteImage('image.png')
```

In practice, pvpython/pvbatch don't work well with all X-window servers ("Could not find a decent visual" or writing empty images), so for this model to work we could recompile ParaView with OSMesa and start pvpython/pvbatch with --use-offscreen.

Fortunately, there is no need to do so, as there is a better solution: working in a **client-server mode**.

Connecting to server on rainbow

- easiest, fastest solution:
  vizN-site $\rightleftharpoons$ rainbow
- very slow solution:
  yourDesktop $\xrightarrow{\text{X11 via ssh}}$ vizN-site $\rightleftharpoons$ rainbow
  ssh -Y username@vizN-site.sharcnet.ca
- slow solution:
  yourDesktop $\xrightarrow{\text{NX/OpenNX}}$ vizN-site $\rightleftharpoons$ rainbow
  NX = X11 with compression and smart caching
- fast, somewhat tricky to set up:
  yourDesktop $\rightleftharpoons$ rainbow

in all cases your data files are on rainbow and your state files (configuration, scripts) are on the local machine

Connecting to server from vizN-site

- put your data on rainbow
- start ParaView on vizN-site
- choose File-Connect
- click on rainbow and then connect
- select the desired number of processors, and press connect again
- might need to enter your SHARCNET password
- ParaView will submit a job to the vis queue on rainbow automatically

Connecting to server from your own desktop

steps described at https://http://www.sharcnet.ca/my/software/show/67

- install latest ParaView (3.8) on your machine
- on Linux/Mac edit ∼/.config/ParaView/servers.pvsc as described in our software page, on Windows similar file - see ParaView documentation
- allow port 11111 on your machine's firewall
- open paths through any additional firewalls (e.g., wireless router)
- *either* start ParaView from your machine, File-Connect, select rainbow and the number of processors
- *or* start from rainbow

  sqsub -q vis -n 4 -N 4 -r 8h -f mpi -o /tmp/pvsharcnet-mpi.%J.out /opt/sharcnet/paraview/3.6.1/bin/pvsharcnet-mpi –reverse-connection –client-host=$myhost

  - need to specify $myhost explicitly
  - limited to 8 hours in "vis" queue

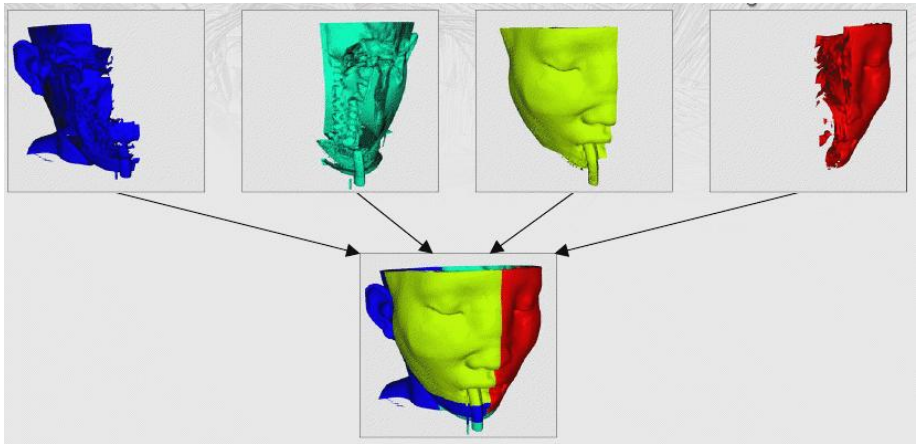
Data partitioning

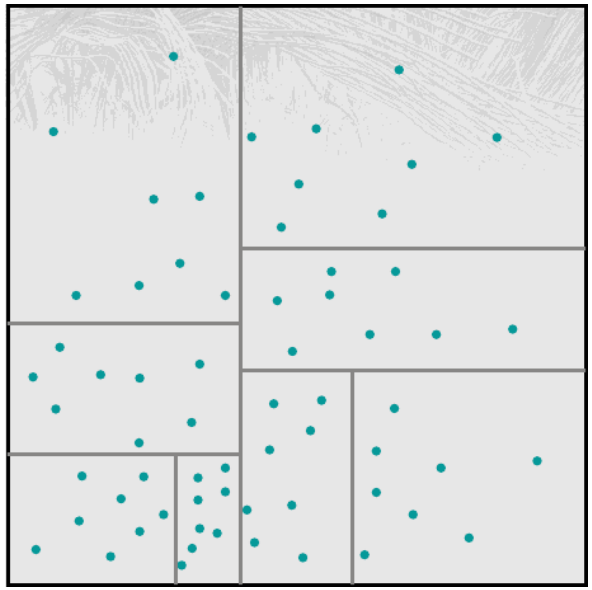
Scalable parallel distributed rendering – load balancing is handled automatically by ParaView for structured data:

- 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
- unstructured data (Unstructured Points, Polygonal Data, Unstructured Grid): one processor core per 0.5-1 million cells
- rainbow: 20 compute nodes, 4 cores / 8 GB memory per node
- always do a scaling study for large jobs!
- it is important to understand the 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

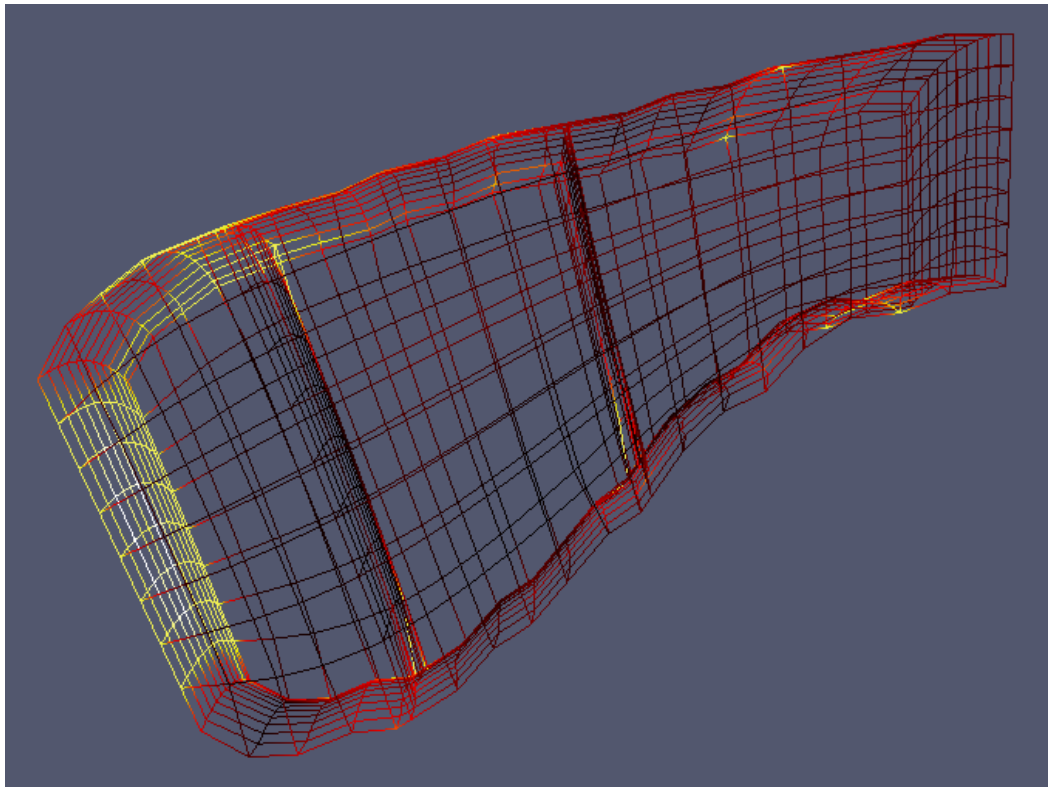
use these with caution: Clip, Decimate, Extract Cells by Region, Extract Selection, Quadric Clustering, Threshold

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.C (adapted from from VTK/Examples/MultiBlock):

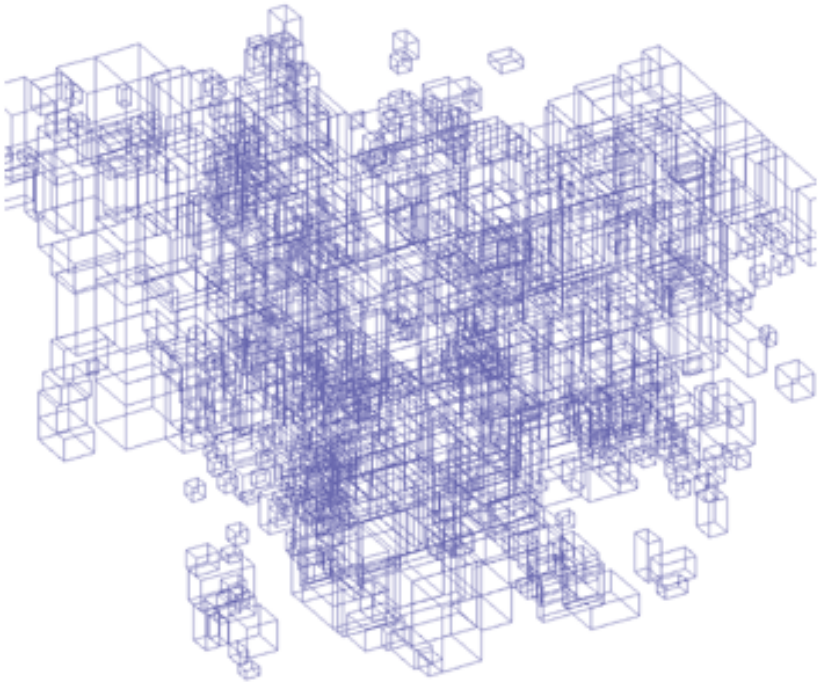
loads three separate structured grid datasets, each from its own file, and writes them as a single multi-block *.vtm dataset (XML-based file format)

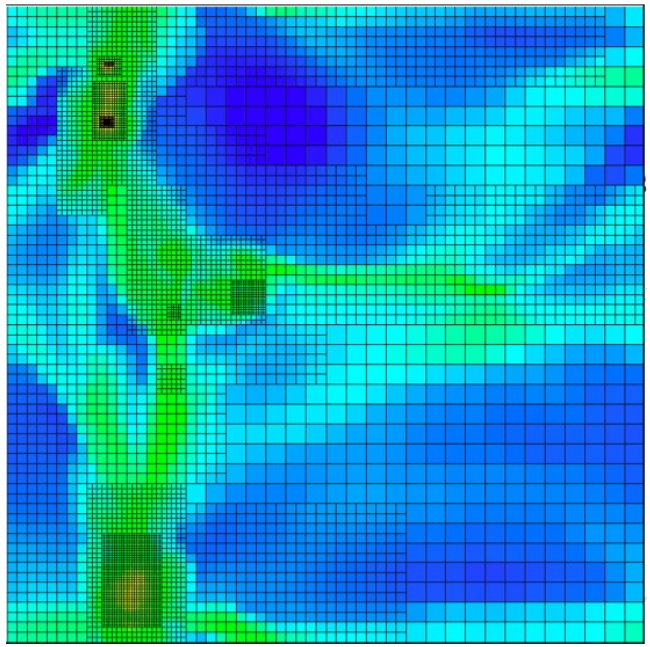


VTK composite datasets: vtkHierarchicalBoxDataSet

vtkHierarchicalBoxDataSet is used for AMR datasets, comprises of refinement levels and uniform grid datasets at each refinement level

prototype code hierarchicalBoxDataWriter.C (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 3.8 tutorial and sample data in many different formats http://www.cmake.org/Wiki/The_ParaView_Tutorial

ParaView F.A.Q. http://www.itk.org/Wiki/ParaView:FAQ

VTK from C++, Python, Tcl code examples
saw.sharcnet.ca:/work/razoumov/VTK/Examples/DataManipulation

VTK from C++ (e.g., StructuredGrid.cxx) code examples
http://www.itk.org/Wiki/VTK/Examples

many sample VTK datasets
saw.sharcnet.ca:/work/razoumov/VTKData/Data

VTK file formats http://www.vtk.org/VTK/img/file-formats.pdf

generate XDMF files in Fortran with
(1) either HDF5 calls + plain text for the XML,
(2) or Fortran $\rightarrow$ C $\rightarrow$ XDMF calls
http://www.xdmf.org/index.php/Write_from_Fortran

Exercises

- visualizing your dataset
- for netCDF users: writing/visualizing from your codes
- for HDF5 users: writing/visualizing xdmf from C++ or Fortran http://www.xdmf.org
- AMR data