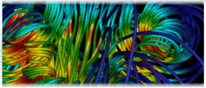
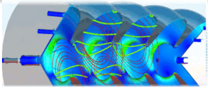
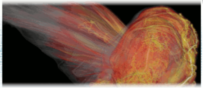
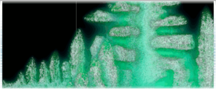





Scientific Visualization with VisIt

ALEX RAZOUMOV
alex.razoumov@westgrid.ca
slides in collaboration with Marcelo Ponce (SciNet)

computecanada

WESTGRID

✓ install VisIt from http://goo.gl/KcGWHa

✓ slides and data files at http://bit.ly/visitzip (∼26 MB)

✦ optional data for movies at http://bit.ly/2dTxkqx (∼361 MB)

Workshop outline

9AM-NOON ➩ MORNING SESSION, COFFEE BREAK @ ∼10:30AM

- Introduction to scientific visualization
  - general ideas, tools, plotting vs. multi-dimensional visualization
  - overview of current general-purpose multi-dimensional visualization tools
- VisIt basics: GUI, loading files, plots and operators
  - working with plots: overview, pseudocolour, contour, volume, ...
  - working with operators: slice, clip, threshold, isosurface, ...
- Quantitative analysis with VisIt
  - invited session by Artem Korobenko (Mechanical and Manufacturing Engineering)
  - data at `http://bit.ly/2pCYMis` (127MB)
- VisIt: professional quality plots (fine tuning) & animation

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

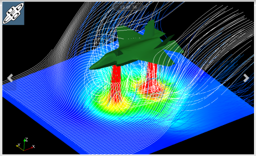
1PM-4PM ➩ AFTERNOON SESSION, COFFEE BREAK @ ∼2:30PM

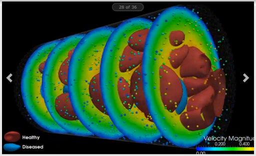
- Python scripting in VisIt
- Remote and distributed visualization with VisIt
- Summary

Ready to show your research or your visualization skills?

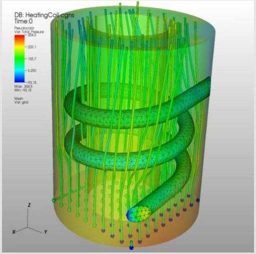
- **Spring:** SEEING BIG showcase (since 2015)
  - **researchers submit visualizations to showcase their own research**
  - March-01 to May-31 submission window
  - entries are displayed in a video loop on a large 3840 × 2160 flat screen in the conference lobby at HPCS in June
  - now accepting 2017 submissions `http://bit.ly/2l9FrR7`
  - don't be afraid to submit your work: we can help you with visualization!
- **Fall:** VISUALIZE THIS! challenge (since 2016)
  - **all participants work on the same dataset or problem**
  - competition with prizes; points awarded for interactive 3D visualization, innovative techniques to display multiple variables
  - one-month competition in 2016, likely two months in 2017
  - emphasis on creating something useful for the scientific community; techniques will be published online
  - always looking for interesting problems; suggestions welcome!

Introduction to scientific visualization

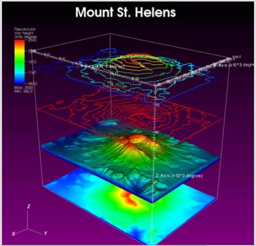




- Visualization is the process of mapping (scientific) data into *"visual form"*



- Much easier to understand images than a large set of numbers
- For interactive data exploration, debugging, communication with peers
- Many examples from different fields of science



compute | calcul
canada | canada

Portal Home
Account Management
Accessing Resources
Technical Support

National Services

- Compute
- Storage
- Compute Canada Cloud
- Data Movement (Globus)
- Visualization
- Research News

Grant Support
Sustainable Planning for Advanced Research in Canada (SPARC)
Feedback
Digital Humanities

RESEARCH PORTAL

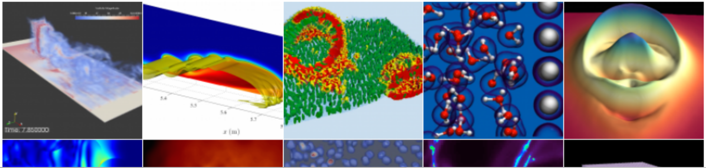
Visualization

French ➤

Using Compute Canada's resources and technical expert help, you can easily convert the results of your numerical simulations or your experimental data into engaging images or movies to share with colleagues, to put online, or into a publication. Our technical staff have extensive experience in scientific visualization and visual data analysis, primarily using open-source tools such as ParaView, VisIt, VTK, Blender, VMD, and various Python libraries to work with a wide variety of data types. Large multi-dimensional datasets can be visualized directly on Compute Canada clusters without having to move them to your desktop. We can help you with all stages of visualization, from preparing data in the right format to interactive analysis. For more information, please contact us at vis-support@computecanada.ca

Compute Canada Visualization Working Group

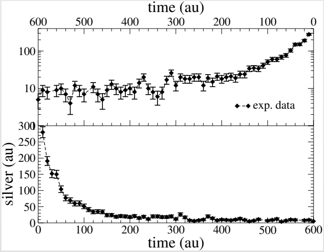
Alex Razoumov, Compute Canada (Lead)
Belaid Moa, UVic
Chris Want, Compute Canada
Dmitri Rozmanov, U of Calgary
Doug Phillips, U of Calgary
Frederick Lefebvre, Laval
Joey Bernard, UNB
Marcelo Ponce, U of Toronto
Maxime Boissonneault, Laval
Michael Hanlan, Queen's
Oliver Stueker, Memorial U
Phil Romkey, SMU
Tyson Whitehead, Western
Weiguang Guan, McMaster



1D/2D plotting vs. multidimensional visualization

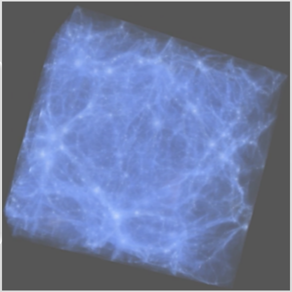
➡1D and 2D plotting

plotting 1D/2D functions and tabulated data, charts, using eg. gnuplot, xmgr, or Python's matplotlib, bokeh and other libraries, R's ggplot2 and its derivatives, or various derivatives of D3.js



➡ multidimensional visualization

usually displaying 3D datasets, typically spatially extended data on structured grids, or on unstructured meshes that have some topology in 2D/3D

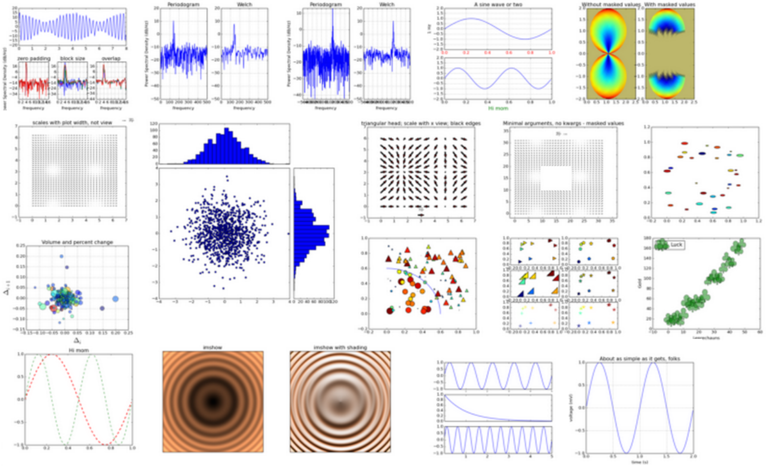


Open-source vs. proprietary

Whatever you do, may be a good idea to avoid proprietary tools, unless those tools provide a clear advantage (most likely not)

- large $$
- limitations on where you can run them, which machines/platforms, how many cores, etc.
- once you start accumulating scripts, you lock yourself into using these tools forever, and consequently paying $$ on a regular basis
- quite often the user base is smaller than for open-source tools, hence more difficult to get help from the community
- with a little bit of coding, there is nothing you cannot do with open-source tools, and we are happy to help!

1D/2D: Matplotlib gallery with hundreds of examples



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

1D/2D: 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

Multidimensional open-source visualization packages

- `gnuplot`: command-driven interactive 2d and 3d plotting program
- `Gephi, GraphViz`: graph visualization
- `HDFview`: visual tool for browsing and editing HDF4 and HDF5 files
- `ImageMagick`: manipulation of image
- `MayaVi`: serial 3D scientific data visualizer (Python + VTK)
- `Molden`: pre/post-processing for molecular and electronic structures
- `OpenDX`: very old, not mantained, but really nice interface and ideas
- **ParaView: parallel scientific visualization**
- `SciLab`: open-source platform for numerical computation
- **VisIt: large-scale scientific visualization**
- `XCrysDen`: crystaline and molecular structure visualization
- `yt`: Python library for visualizing AMR datasets
- `VMD`: visualization for molecular dynamics

2D/3D visualization packages

Desired features for large-scale scientific visualization

- 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
- Open-source, multi-platform, and general-purpose

Visualization Toolkit (VTK) library

`http://www.vtk.org`

- For 3D computer graphics, image processing and visualization
- Open source, multi-platform
- Supports distributed computation models
- Extensible modular architecture
- Collection of C++ libraries

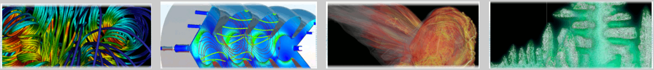
- Leveraged by many applications
- Divided into logical areas
  - filtering
  - information visualization
  - volume rendering
- Cross-platform, using OpenGL for GPU acceleration
- Wrapped in Python, Tcl, Java

VTK file formats can encode *spatial data* defined on Cartesian, rectilinear, curvilinear grids, on particles, on unstructured 2D (polygonal) and unstructured 3D meshes

ParaView and **VisIt** are end-user applications supporting:

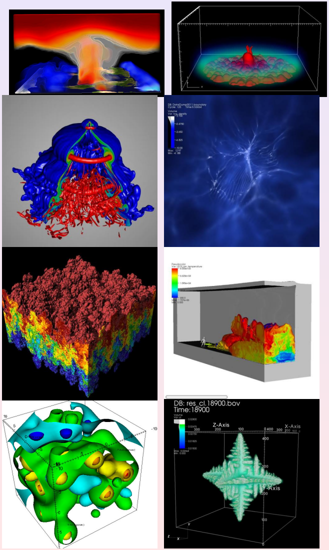
- *parallel* data reading/processing/rendering
- single-node, *client-server*, *MPI cluster* rendering
- 100+ input file formats, many different rendering options

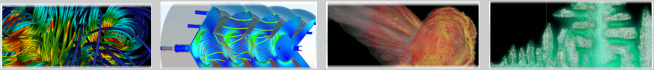
VisIt overview



VisIt

- `http://visit.llnl.gov`
- Developed by the DOE *Advanced Simulation and Computing Initiative* (ASCI) to visualize results of terascale simulations
- First release fall of 2002, mantained by LLNL
- Currently $\sim$ 20 developers from different organizations and universities
- v2.12 available as source and binary for Linux, Mac, Windows `http://bit.ly/2dMH091`
- Over 80 visualization features (contour, mesh, slice, volume, molecule, ...)
- Reads over 120 different file formats `http://bit.ly/2egAkzA`
- Interfaces with C++, Python, and Java
- Uses MPI for distributed-memory parallelism on HPC clusters

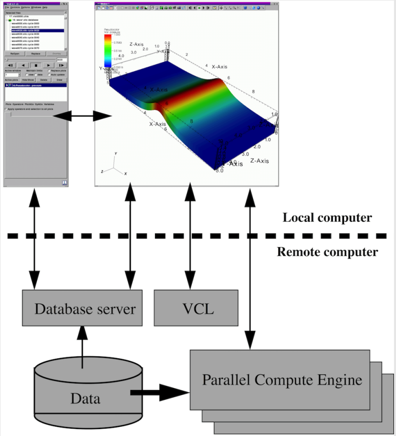




VisIt

- Can run locally, remotely, in the client/server mode
- GUI looks pretty much the same on each platform
- New database plugin readers can be developed
- Provides a library for in-situ visualization (*libsim*), to instrument your simulation code for VisIt to connect to, as though the simulation was a VisIt compute engine
- Supported mesh types:
  - 1D curves
  - 2D/3D: Cartesian, rectilinear, curvilinear, unstructured, points, AMR, molecular, CSG (constructive solid geometry)

➡ VisIt Architecture



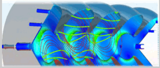
VCL = VisIt Component Launcher

VisIt: multiple interfaces

- GUI (graphical user interface)
- Python programming interface
- Java programming interface
- C++ programming interface

Use multiple interfaces simultaneously

➨ use VisIt as an application or a library
➨ C++, Python, Java interfaces allow other applications to control VisIt
➨ we'll see an example of this with Python where we can attach a Python shell to a VisIt session running on a specific port on your laptop



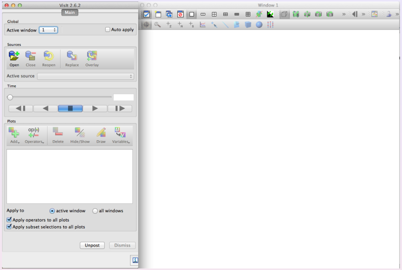
VisIt: GUI

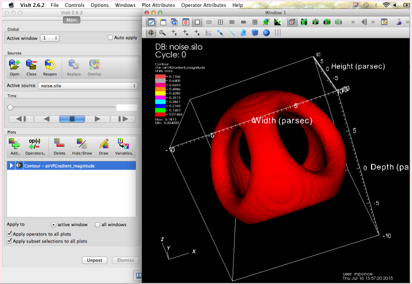
GUI window

- select files to visualize
- create and manage plots
- set plot attributes
- add operators
- set look and props. for visualization
- "Apply to..." really useful

Viewer window

- display all of the data being visualized
- mouse navigation
- up to 16 vis windows
- popup menu
- toolbars





VisIt's pipeline and core abstractions

1. Open a database
2. Create a plot
3. Set plot attributes
4. Apply operators to plot to modify data
5. Set operator attributes
6. Compute engine generates a plot displayed in the vis. window
7. Iterate/repeat ...
8. Save your visualization

- Databases
  - interchangeable with "dataset" or "file"
  - data can be defined on **nodes** (ParaView's *points*) or **zones** (ParaView's *cells*)
- Plots
  - visualize (render) the data
  - similar to ParaView's *representations* (Surface, Volume, Wireframe, etc)
- Operators
  - manipulate (process) the data
  - similar to ParaView's *filters*
- Expressions
  - derive quantities
  - similar to ParaView's *Calculator filter* and *Programmable Filter*
- Queries
  - obtain quantitative info

Importing data into VisIt

Importing your dataset into VisIt

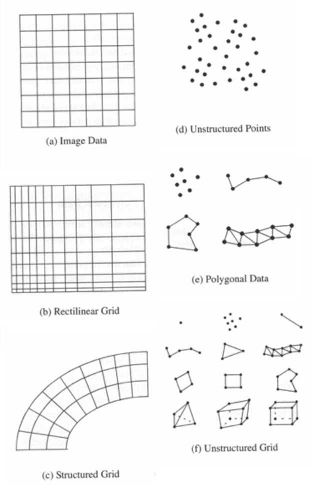
- ✓ If your code generates one of 120+ formats natively understood by VisIt ⇒ you are all set
- ✓ If you want to save a 2D/3D array of a scalar or vector variable (defined on a Cartesian grid) ⇒ store your data in NetCDF or VTK
  - NetCDF libraries are available for pretty much any programming language (C, C++, F90, Python, R, Java, ...)
- ✓ If your grid is rectilinear, curvilinear (e.g., cylindrical or spherical), unstructured, or your variable is on particles (atoms, N-body) ⇒ store your data in one of the VTK formats
- ✓ For small datasets can import CSV (forward four slides)
- ✓ Anything else (e.g., structured non-spatial data) ⇒ let me know

For large datasets **do not**:

- ✗ store your data as ASCII: ∼5X more space/bandwidth than binary
- ✗ use raw binary: not portable, no descriptive metadata

Importing your dataset into VisIt: VTK file formats

- Legacy serial format (*.vtk): ASCII header lines + ASCII/binary data
  - check two purely ASCII examples
    - (1) `datasets/volume.vtk` is $3 \times 4 \times 6$ Structured Points
    - (2) `datasets/density.vtk` is $2 \times 2 \times 2$ Structured Grid
  - can use these as templates for small files
- 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 endians byte order), etc.
  - could link your code against a *compiled* VTK library in C/C++, or install it in Python 2.x (not available in Python 3.x) with a package manager, e.g., conda
  - another option is PyEVTK library, although does not provide all the bells and whistles



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

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

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

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

Exercise: visualizing 3D data with legacy VTK

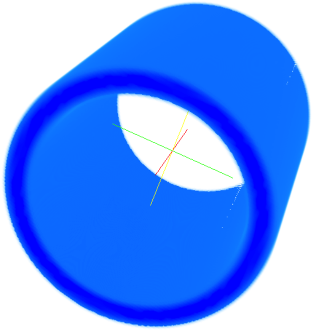
We will come back to this exercise after we cover the basics

- Visualize a 3D "cylinder" function inside a unit cube ($x, y, z \in [0, 1]$)

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

where $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$

➠ reproduce the view on the right



- You have two options:

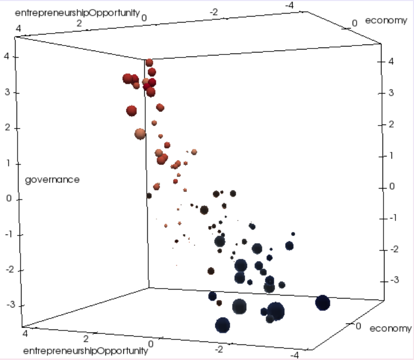
(1) `datasets/cylinder.dat` contains data in ASCII
  - add an appropriate header to create a legacy VTK file
  - use either `datasets/volume.vtk` as a template

(2) Use PyEVTK in Python to create an XML VTK file with binary data

Example of reading a CSV: country prosperity index

- Data from the Legatum 2015 Prosperity Index `http://www.prosperity.com/#!/ranking` (click on Scores, best to copy/paste from Firefox)
- Take a look at the data in `legatum2015.csv`: 8 rankings for each country
- 3D position by economy + entrepreneurshipOpportunity + governance, colour by education

  Add ▾ → Scatter, select the axes and then click Draw, in plot properties under Appearance tab set Point Type = Sphere, set radius
- In the plot on the right size by safetySecurity was done in ParaView, a little bit more difficult to do this in VisIt
- Can use Controls → Expressions... to name variables to appear on the axes



This is quick demo of reading a CSV

- ➠ do not attempt to run this now
- ➠ we'll dive into the GUI details shortly

Why CSV is not such a good idea for large datasets

Let's generate and store 10^6 particles with $x, y, z \in [0, 1]$:

the following produces a 43M file (without spaces!)

```
from random import random
for i in range(int(1e6)):
    print str(random())+','+str(random())+','+str(random())
```

the following produces a 14M file (still in double precision)

```
from writeNodesEdges import writeObjects
from random import random
coords = []
for i in range(int(1e6)):
    coords.append([random(),random(),random()])
writeObjects(coords, fileout='test', method = 'vtkUnstructuredGrid')
```

- In this VTK there is no data on top of the mesh, but still easy to visualize: Add▾ → Mesh → mesh
- With both gzip-compressed, still a factor of 2X discrepancy

VisIt basics

Reading data from files

Importing datasets

- File → Open file... and select the data file (e.g., `noise.silo`)
- It becomes available in Active source
- File → File information... will give you some info about the dataset

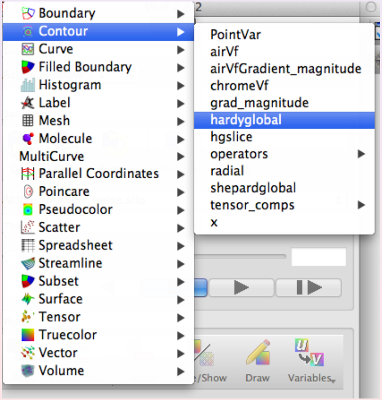
Restoring a previous session

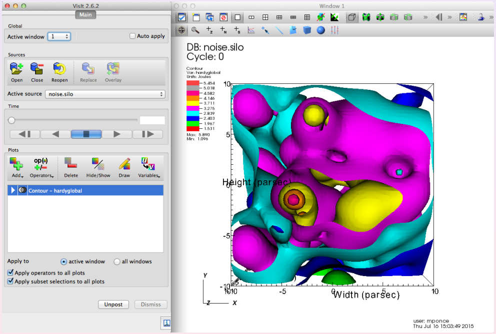
- File → **Restore session...** loads the previous state of the given session (**that needs to be specifically saved**)
- File → **Restore session w/sources...** is extremely useful for re-identifying datasets that could have been moved or renamed

Be aware that by default VisIt won't save your work (session) nor ask you when you try to exit the program!

Contours

(1) Add▾ → Contours ⇝ hardyglobal and then click Draw



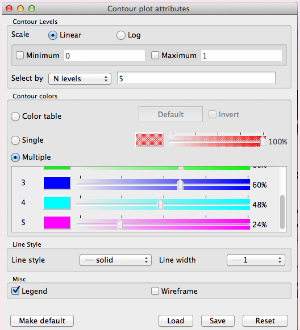


Contours

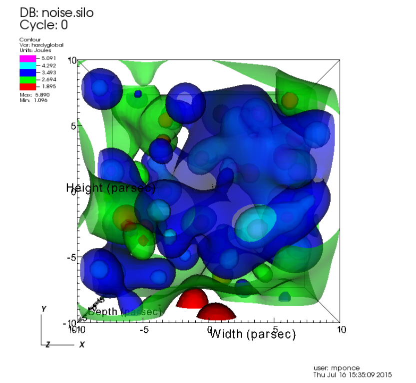
(1) double-click on Contour-hardyglobal

(2) under Select by, choose N levels = 5 Enter

(3) change opacity levels, e.g. 50%, 60%, 60%, 48%, 24%



(4) Apply & Dismiss

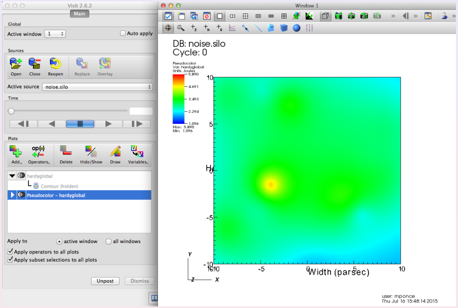


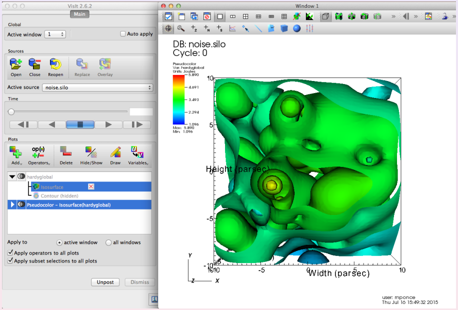
(5) Hide/Show or Delete

PseudoColor & IsoSurfaces

(1) [Add▾] → [**Pseudocolor**]
⇝ hardyglobal → [Draw]

(2) [Operators▾] → [**Slicing**]
→ [**Isosurface**] → [Draw]



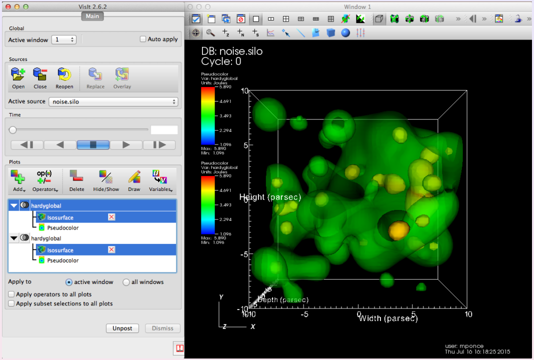


(3) click ► to expand, double click on [Isosurface]

(4) under Select by, choose [Percent (s)] = 50 [Apply] & [Dismiss]

(5) change the opacity of [Pseudocolor]

PseudoColor & IsoSurfaces



(1) unselect the [Apply ...] check-boxes

(2) add one more Pseudocolor+**Isosurface** w/Percent(s)=80 & adjust its opacity

(3) select/check the [Apply...] boxes

(4) Operators▾ → **Selection** → **Clip** and select a combination of unequal planes to modify the Clip

General remarks

- Operators/plots can be removed **Delete** or hidden **Hide/Show**
- Save your work **frequently**: File → **Save session...**

Slices

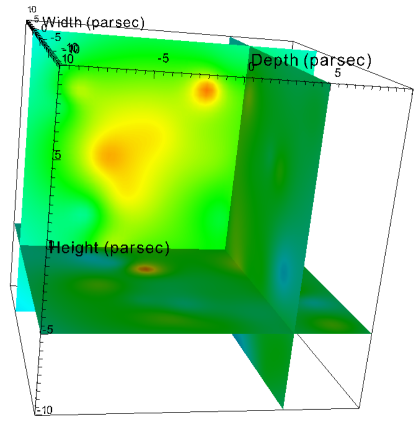
(1) Start from scratch

(2) Add▾ → **Pseudocolor**
⇝ hardyglobal → Draw

(3) Operators▾ → **Slicing**
→ **ThreeSlice** → Draw

(4) Optionally reposition the planes

(5) Optionally add a second light at ∼ 45° elevation on a side
Controls → Lighting...
(don't forget to enable it)



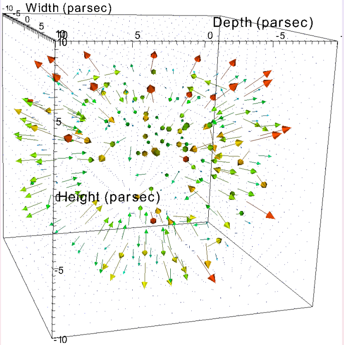
Vector field representation with glyphs

(1) With `noise.silo` loaded and no prior plots
Add▾ → **Vector** ⇝ grad and click
Draw

(2) Double-click on Vector and set Vector placement = Uniformly located throughout mesh, Vector amount = 3000

(3) Optionally can play with the glyph properties under Glyph tab

(4) Under **Variables▾** can switch to a different vector field (see on the right)



Vector field representation with streamlines

(1) With `noise.silo` loaded and no prior plots
Add▾ → Pseudocolor → operators → IntegralCurve ⇝ grad, click Draw

(2) Double-click on IntegralCurve, in Integration tab set Source type = Plane, Origin = (0,0,0), Normal = (0,1,0), Up axis = (0,0,1), Sampling type = Uniform, 15 samples in X/Y, Distance in X/Y = 20, Integration direction = Both, Max number of steps = 1000

- the "up axis" serves as the "Y" axis embedded in the plane
- distance in X/Y is the size of the source rectangle in the plane

(3) Operators▾ → **Geometry** → Tube, click Draw

(4) Double-click on Tube, set Radius = 0.003

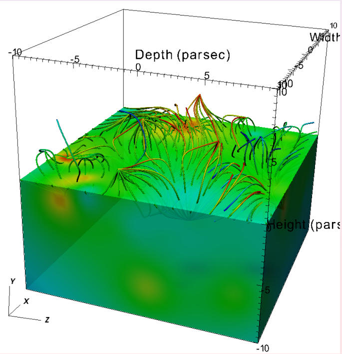
(5) Uncheck "Apply operators to all plots"

(6) Add▾ → Pseudocolor ⇝ hardyglobal

(7) Operators▾ → **Selection** → Clip

(8) Double-click on Clip, modify its properties to reproduce the picture on the right

(9) Next, experiment with with different Source types in IntegralCurve



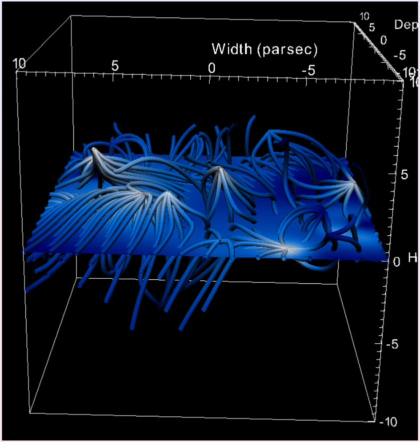
Streamlines: final touches

- Now the volume is coloured by hardyglobal, and the streamlines by grad
- Let's colour streamlines by hardyglobal
  - Double-click on IntegralCurve, under Appearance tab set Data value = Variable, and from the menu on the right select [Scalars] ⇝ hardyglobal, click [Apply]
  - Inspect your visualization to verify that both are coloured by the same variable
- Very easy to turn on/off legends, user info, axes via [Controls] → [Annotation...]
- [File] → [Save Window] to save the image
  - by default, on Mac/Linux will go into either the home directory, or the directory from which VisIt was launched

Exercise: slice

Try to reproduce the picture on the right

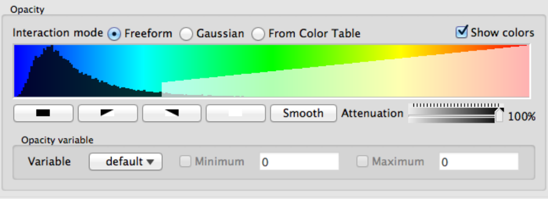
- Delete the Clip
- Slice the previous plot [Add▾] → [Pseudocolor] ⇝ hardyglobal with [Operators▾] → [Slicing] → [Slice]
- Invert the background
- Change both colour maps
- Make the tubes thicker

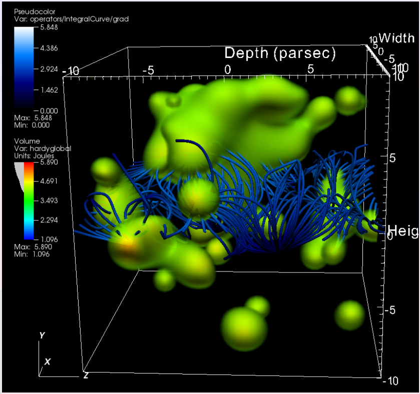


Exercise: volume rendering

Try to reproduce the picture on the right

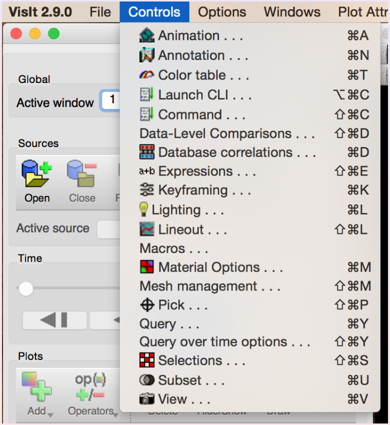
- Delete the slice and the hardyglobal plot, keep the streamlines
- Add [Add▾] → [Volume] ⇝ hardyglobal
- Double-click on Volume, under 1D transfer function tab set the Opacity, under Renderer Options tab set "Ray casting: compositing", click [Apply] and [Draw]
  - on my laptop takes ∼10s to render





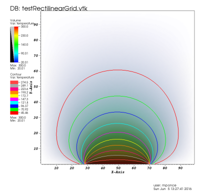
Controls

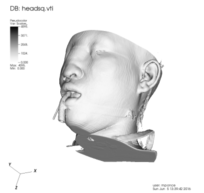
- We already saw some Controls menu items
  - Lighting... to set new lights, disable old ones, set their position, colour, brightness
  - Annotation... to set legends, axes, dataset name, user info
- Some other useful ones
  - Database Correlations... to create *correlations* between ≠ datasets
  - Expressions... to create *expressions* (new quantities) from variables/datasets
  - View... to list all camera setup variables

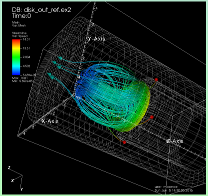


Hands-on

- Load some of the other datasets (`testRectilinearGrid.vtk`, `headsq.vti`) or *your own data*!!!
- Try to explore the data and visualize it, using some of the tools we have discussed
- If you have used other visualization packages (ParaView?), compare whether it is possible (and how easy) to obtain similar results with those tools
- Which tool/package/library is more intuitive, elegant, useful for you and your research?







More datasets available for playing, at: `http://www.visitusers.org/index.php?title=Tutorial_Data`

Quantitative analysis

Quantitative analysis

- Expressions (similar to ParaView's *Calculator* filter)
- Pick modes: zone/node, spreadsheet, time curves
- Lineout mode (similar to ParaView's *Plot Over Line* filter)
- Queries
- Creating new database correlation time sliders

Expressions

- Create new derived variables from existing ones
- Mathematical expressions can operate on scalars, vectors, tensors
- **Option 1**: use Standard Editor to select existing functions and expressions
  - similar to ParaView's *Calculator* filter
- **Option 2**: use Python Expression Editor - this is an advanced option for working with VTK objects
  - similar to ParaView's *Programmable Filter*

Expressions

- Load *noise.silo*, visualize *hardyglobal* with Pseudocolor
  - Controls → Expressions... → New, set name = *squared*, type = Scalar Mesh Variable, definition = *hardyglobal^2*, click Apply
  - Now in the list of variables switch to *squared*

- Controls → Expressions... → New, set Name = *gradient*, Type = Vector Mesh Variable, Definition = *gradient(hardyglobal)*
  - You can use the dropdown menus to accelerate typing (gradient will be found in Insert Function → Miscellaneous)
  - Add a vector plot to show *gradient*, picking "Uniformly located troughout the mesh", displaying 5000 vectors, making them bigger, overlaying onto a semi-transparent Pseudocolor plot of *hardyglobal*

- Controls → Expressions... → New, set Name = *truncated*, Type = Scalar Mesh Variable, Definition = *max(2,hardyglobal)*
  - Now make a Pseudocolor plot of *truncated*

Expressions

- Load *noise.silo*, visualize *hardyglobal* with Pseudocolor
  - Controls → Expressions... → New, set name = *squared*, type = Scalar Mesh Variable, definition = *hardyglobal^2*, click Apply
  - Now in the list of variables switch to *squared*

- Controls → Expressions... → New, set Name = *gradient*, Type = Vector Mesh Variable, Definition = *gradient(hardyglobal)*
  - You can use the dropdown menus to accelerate typing (gradient will be found in Insert Function → Miscellaneous)
  - Add a vector plot to show *gradient*, picking "Uniformly located troughout the mesh", displaying 5000 vectors, making them bigger, overlaying onto a semi-transparent Pseudocolor plot of *hardyglobal*

- Controls → Expressions... → New, set Name = *truncated*, Type = Scalar Mesh Variable, Definition = *max(2,hardyglobal)*
  - Now make a Pseudocolor plot of *truncated*

Expressions

- ☛ Load *noise.silo*, visualize *hardyglobal* with Pseudocolor
  - Controls → Expressions... → New, set name = *squared*, type = Scalar Mesh Variable, definition = *hardyglobal^2*, click Apply
  - Now in the list of variables switch to *squared*

- ☛ Controls → Expressions... → New, set Name = *gradient*, Type = Vector Mesh Variable, Definition = *gradient(hardyglobal)*
  - You can use the dropdown menus to accelerate typing (gradient will be found in Insert Function → Miscellaneous)
  - Add a vector plot to show *gradient*, picking "Uniformly located troughout the mesh", displaying 5000 vectors, making them bigger, overlaying onto a semi-transparent Pseudocolor plot of *hardyglobal*

- ☛ Controls → Expressions... → New, set Name = *truncated*, Type = Scalar Mesh Variable, Definition = *max(2,hardyglobal)*
  - Now make a Pseudocolor plot of *truncated*

Zone/node pick mode

- Interactively pick values inside the visualization window
- Load *noise.silo*, visualize *hardyglobal* with Pseudocolor, apply [Operators▾] → [Selection] → [Clip] and click [Draw]
- Data here is defined on nodes, not zones
- Right click on the visualization, select [Mode] → [Zone Pick] (or use a mode button in the vis window toolbar), and click anywhere on the vis – it'll display 8 nodes forming the zone, and their variable values
  - each pick point leaves a marker that you can look up in the Pick window
  - the Pick window displays information in tabs arranged by a point
- Similarly, [Mode] → [Node Pick] will display a single node, its variable, and its 8 "incident" zones

Zone/node pick mode

- Interactively pick values inside the visualization window
- Load *noise.silo*, visualize *hardyglobal* with Pseudocolor, apply Operators▾ → Selection → Clip and click Draw
- Data here is defined on nodes, not zones
- Right click on the visualization, select Mode → Zone Pick (or use a mode button in the vis window toolbar), and click anywhere on the vis – it'll display 8 nodes forming the zone, and their variable values
  - each pick point leaves a marker that you can look up in the Pick window
  - the Pick window displays information in tabs arranged by a point
- Similarly, Mode → Node Pick will display a single node, its variable, and its 8 "incident" zones

Zone/node pick mode

- Interactively pick values inside the visualization window
- Load *noise.silo*, visualize *hardyglobal* with Pseudocolor, apply Operators▾ → Selection → Clip and click Draw
- Data here is defined on nodes, not zones
- Right click on the visualization, select Mode → Zone Pick (or use a mode button in the vis window toolbar), and click anywhere on the vis – it'll display 8 nodes forming the zone, and their variable values
  - each pick point leaves a marker that you can look up in the Pick window
  - the Pick window displays information in tabs arranged by a point
- Similarly, Mode → Node Pick will display a single node, its variable, and its 8 "incident" zones

Spreadsheet pick mode and time curves in the picks

- Selecting Mode → Spreadsheet Pick shows a spreadsheet view of one of the dataset variables highlighting the picked node (i,j,k) and its value of the variable
  - the spreadsheet window is controlled from the pipeline!
- Mode → Navigate will take you back to default interaction
- Try loading a time-dependent dataset, e.g., *2d0**.vtk*, and display it in Pseudocolor
  - a time slider will become active
  - depending on the data use either Zone Pick or Node Pick
  - inside the Pick window in the Time Pick tab select "Do time curve with next pick"

Spreadsheet pick mode and time curves in the picks

- Selecting Mode → Spreadsheet Pick shows a spreadsheet view of one of the dataset variables highlighting the picked node (i,j,k) and its value of the variable
  - the spreadsheet window is controlled from the pipeline!
- Mode → Navigate will take you back to default interaction
- Try loading a time-dependent dataset, e.g., *2d0**.vtk*, and display it in Pseudocolor
  - a time slider will become active
  - depending on the data use either Zone Pick or Node Pick
  - inside the Pick window in the Time Pick tab select "Do time curve with next pick"

Spreadsheet pick mode and time curves in the picks

- Selecting Mode → Spreadsheet Pick shows a spreadsheet view of one of the dataset variables highlighting the picked node (i,j,k) and its value of the variable
  - the spreadsheet window is controlled from the pipeline!
- Mode → Navigate will take you back to default interaction
- Try loading a time-dependent dataset, e.g., *2d0**.vtk*, and display it in Pseudocolor
  - a time slider will become active
  - depending on the data use either Zone Pick or Node Pick
  - inside the Pick window in the Time Pick tab select “Do time curve with next pick”

Lineout mode

- Extracts 1D curves from 2D data (unlike ParaView's *Plot Over Line* filter, does not seem to work on 3D datasets)
- Load a 2D dataset or apply Operators▾ → Slicing → Slice to a 3D dataset
- Select Mode → Lineout and draw a line ⇒ the profile will be plotted in a new window

Queries

- Controls → Query...
- **Option 1**: use Standard Queries
  - very useful: Memory Usage
  - quick ways to probe data: MinMax, NumNodes, NumZones, Average Value, Volume
  - Lineout and Pick are also queries (this time enter selection manually)
  - certain queries provide a "Do Time Query" button that calculates the query on each time step and creates a curve
- **Option 2**: use Python Query Editor for custom queries
  - this is an advanced topic for working with VTK objects
  - instead we'll use Query() in Python scripting (later today)

Database correlations

- In VisIt each time-varying database (if more than one loaded) gets its own independent slider
- Sometimes it's useful to compare two time-varying databases, but one would need to set them both to the same moment(s) in time
- Controls → Database Correlations... lets you do this with a single time slider for both databases, using one of four correlation methods
- Can try creating a single time slider from *2d0**.vtk* and *modified0**.vtk*
  1. load both databases, for each draw Pseudocolor and apply Operators → Transforms → Elevate, make them both visible
  2. verify you can animate either switching the active time slider
  3. now select Controls → Database Correlations..., click New, use Correlation Method = Padded Index, select both Sources and move them to Correlated Sources, click Create Database Correlation
  4. a new active time slider appears that lets you animate both

Database correlations

- In VisIt each time-varying database (if more than one loaded) gets its own independent slider
- Sometimes it's useful to compare two time-varying databases, but one would need to set them both to the same moment(s) in time
- Controls → Database Correlations... lets you do this with a single time slider for both databases, using one of four correlation methods
- Can try creating a single time slider from *2d0**.vtk* and *modified0**.vtk*
  (1) load both databases, for each draw Pseudocolor and apply Operators▾ → Transforms → Elevate, make them both visible
  (2) verify you can animate either switching the active time slider
  (3) now select Controls → Database Correlations..., click New, use Correlation Method = Padded Index, select both Sources and move them to Correlated Sources, click Create Database Correlation
  (4) a new active time slider appears that lets you animate both

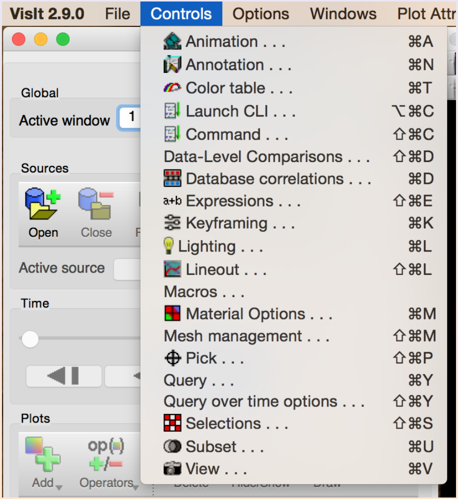
Advanced topics in quantitative analysis

- Cross-mesh field evaluation (CMFE) and database comparison
  `http://bit.ly/2faoAKs`
  - CMFE expressions evaluate a field from a donor mesh onto a target mesh to form a new field
  - Different ways to access it:
    (1) Controls → Data-Level Comparisons...
    (2) Controls → Expressions...
    (3) Python scripting

More controls: professional quality plots and animation

Professional quality plots

- Annotations
- Colors
- Lighting
- Views



Annotations

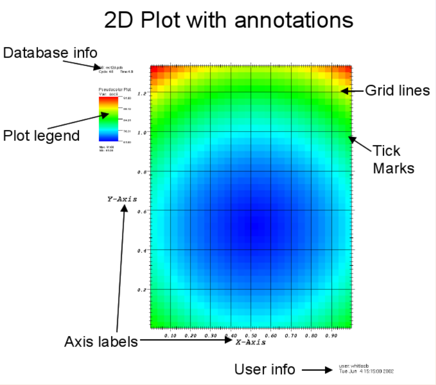
Annotations

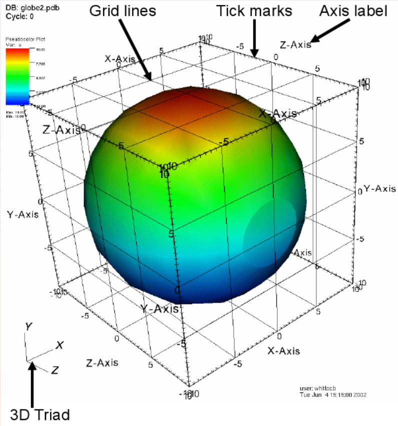
- objects in the vis. window that convey information about the plots
- make clear what is being visualized and make the visualization appear more polished

Types of annotations

- database name
- user name
- plot legends
- plot axes and labels (2D & 3D)
- 3D triad
- 2D, 3D text
- time slider
- images
- lines and arrows

2D & 3D annotations



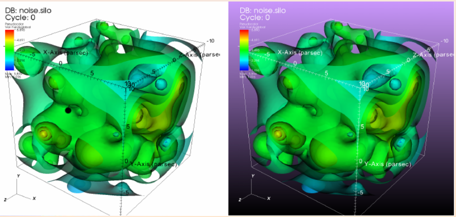


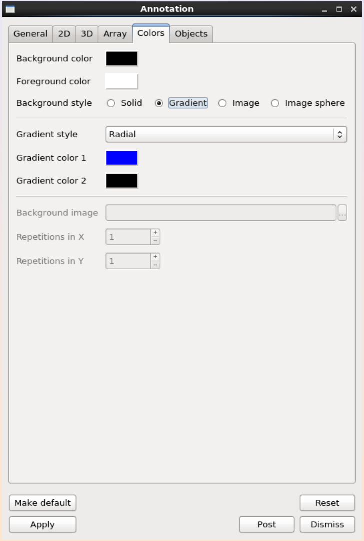
➠ Controls → **Annotation...** ➡ ["2D" tab]

➠ Controls → **Annotation...** ➡ ["3D" tab]

Colors and backgrounds

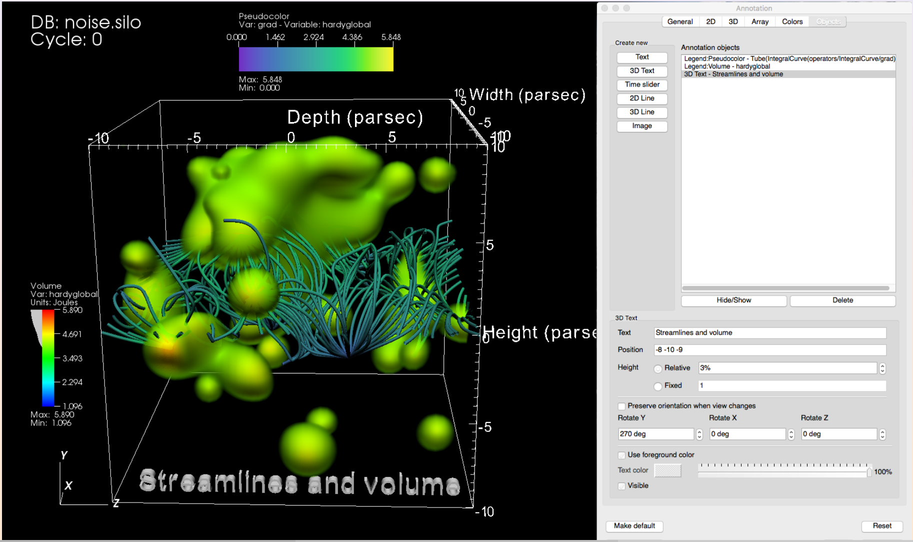
- Controls → **Annotation...**
- **["Colors" tab]**
  - Set background/foreground
  - Background styles:
    - solid
    - gradient
    - image (flat image)
    - image sphere (image that rotates with the view)
    - number of image repetitions





Annotation objects

Controls → Annotation... ➨["Objects" tab]

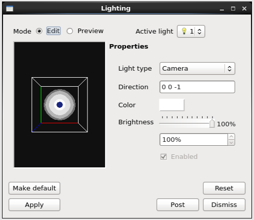


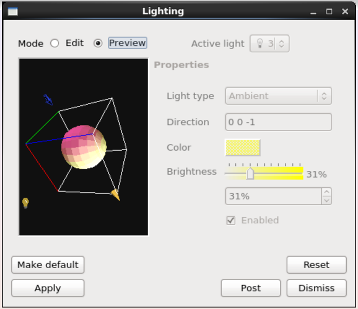
Lighting

- Affects the brightness of plots
- 3D visualizations may require multiple *light sources*
- VisIt allows up to 8 sources
- Each light source can be positioned and coloured

Controls → **Lighting...**

- [Edit]: configure light sources
- [Preview]: all sources visible



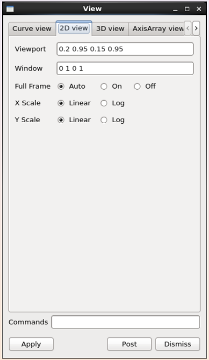


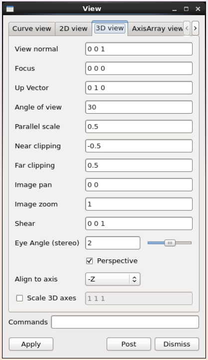
- Only the active light can be modified
- Types of lights: ambient, camera, object light
- Position, colour, brightness

View

(1) "View" can be set *interactively* in the vis. window (click and drag, ...)

(2) Or using a "View Window" Controls → **View...**



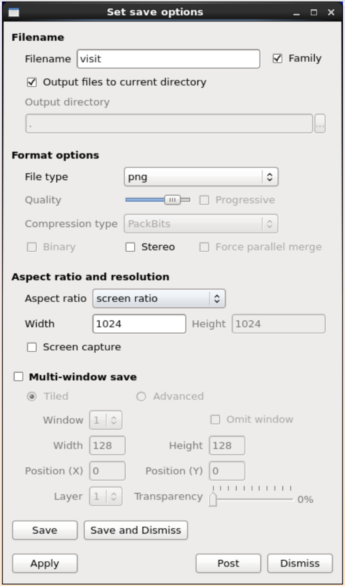


- "**Locked view**": when the view changes in any locked window, all other locked windows readjust to it
  - accessible from the [Advanced] tab or Lock menu item in the vis. window

Saving images to file

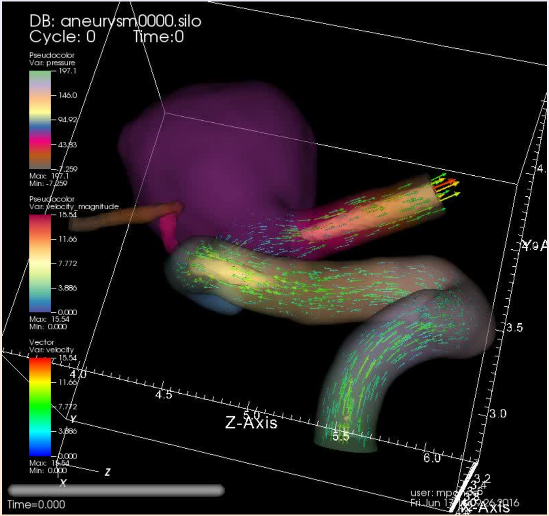
(1) File → **Set Save option...** allows you to control the properties of the image: file type, resolution, naming convention, etc.

(2) File → **Save window** generates the image/file of the currently displayed window



Movie generation

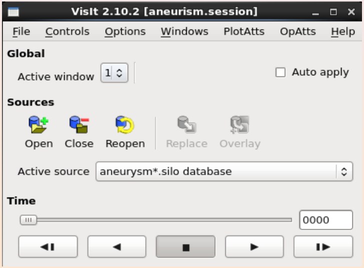
- Sequence in time (*evolution*)
- Motion through space or any property of any pipeline object (see the *Scripting* section)



Basic *timestep* animation

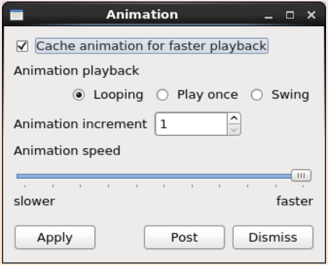
- Simplest case: sequence of similar files in time
- Allows database behaviour over time to be quickly inspected (without the complexity of scripting)
- Controlled through [VCR]-type buttons in the main window
- Load either `datasets/evolution/2d*.vtk` or `aneurysm_data/aneurysm0*.silo`

[Time Slider]



[Main Window]

[Animation Window]



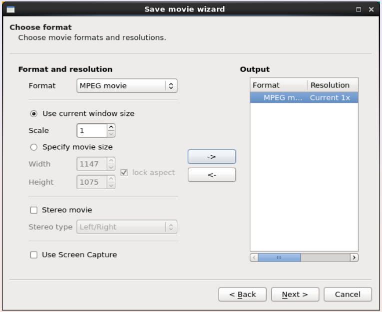
Controls → **Animation...**

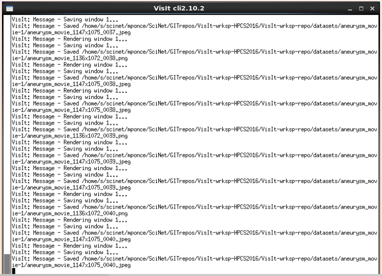
Movie wizard

➠ File → **Save Movie...**

Guided movie generation

- can produce several formats and resolutions, at the same time
- stereo movies
- can use currently allocated processors or spawn another VisIt session for movie generation
- can use movie templates to assemble complex sequence of frames

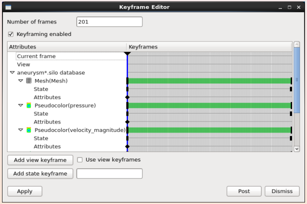




Keyframing

- Advanced form of animation to “play back attributes”
- Attributes that can be keyframed: plots attributes, database states, view
  - ✘ operator attributes are mentioned in docs but don’t seem to play back (a bug?)
- E.g., can make a plot slowly fade out, slowly spin, etc.

Controls → **Keyframing...**



(1) Enable “keyframing mode”

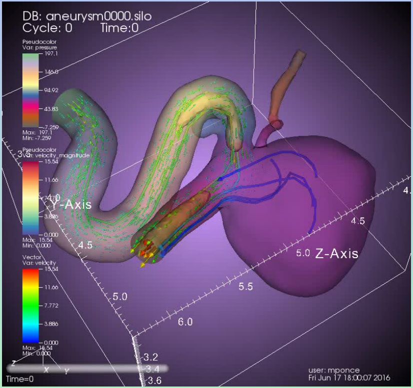
(2) Adjust number of frames

- Add a first keyframe: open a plot’s attribute window, change settings, click its Apply button
- Add a second keyframe: move the **keyframe time slider** to a later time, change the plot attribute(s) again, click its Apply button ✘ this part often does not seem to work (a bug?)
- Each time you add a keyframe to the animation, a small black diamond (**keyframe indicator**) will appear
- You can drag the **keyframe indicator** to set the time range for each attribute
- Right-click on a **keyframe indicator** to delete it

➠ If time slider present, it changed to “Keyframing Animation” – in this case simply select the desired **active** time slider from the pull-down menu

Hands-on

- Using the “`aneurysm`” dataset or *your own data*, generate a time-sequence movie
- Experiment with keyframming, lighting, ..., or any of the other techniques we have been discussing



More info about this dataset at

`http://www.visitusers.org/index.php?title=Blood_Flow_Aneurysm_Tutorial_Dataset_Exploration`

Python scripting in VisIt

Why scripting?

- Automate repetitive GUI tasks
- Reproducibility
  - a script is a documented workflow
  - can easily pass a script to someone else
  - can run it yourself years later
- Batch processing on large systems (clusters)
  - perhaps, no GUI
  - submit a rendering job

Python scripting in VisIt

- Launching VisIt's Python scripts from the Unix command line without the GUI

```
$ /path/to/VisIt -nowin -cli -s script.py
```

  - flag `-nowin` for offscreen (typically OSMesa) rendering
  - similar to ParaView's `pvbatch`
  - very useful for running a batch rendering job on a cluster

- Launching VisIt's Python scripts from the GUI
  - VisIt has a built-in Python 2.7 shell through Controls → Launch CLI...; it'll start VisIt's Python interpreter in a terminal and attach it to the running VisIt session on a specific port on your laptop with a one-time security key
  - alternatively, Controls → Command... provides a text editor with Python syntax highlighting and an Execute button, lets save up to eight snippets of Python code

Python scripting in VisIt

- Recording scripts from the GUI
  - Controls → Command... window lets you record your GUI actions into Python code that you can use in your scripts (similar to ParaView's Trace Tool)

- Other places to use Python in VisIt's GUI

(1) in Controls → Expressions... → Python Expression Editor (similar to the Programmable Filter in ParaView)

- ★ expressions on VTK datasets
- ★ tutorial at `http://bit.ly/2ezF6qr`
- ★ can modify the geometry of the dataset, e.g., warp the grid in 3D or create a projection
- ★ result appears in the list of variables to plot
- ★ more advanced topic for another time

(2) in Controls → Query... → Python Query Editor

- ★ queries on VTK datasets
- ★ more advanced topic for another time

Adding plots (typing in interactive shell)

Starting from scratch, run Python shell Controls → Launch CLI... and type in the following commands (adjust the file path!):

```
OpenDatabase("~/teaching/visitWorkshop/datasets/noise.silo")
AddPlot("Pseudocolor", "hardyglobal")
DrawPlots()
```

- Each plot in VisIt has a number of attributes that control its appearance
- To access them, first create a **plot attributes object** by calling a function PlotNameAttributes(), e.g., PseudocolorAttributes(), or VolumeAttributes()
- If changing attributes, pass the object to the SetPlotOptions()
- If setting new defaults, pass the object to SetDefaultPlotOptions()

Adding plots (typing in interactive shell)

Starting from scratch, run Python shell Controls → Launch CLI... and type in the following commands (adjust the file path!):

```
OpenDatabase("~/teaching/visitWorkshop/datasets/noise.silo")
AddPlot("Pseudocolor", "hardyglobal")
DrawPlots()
```

- Each plot in VisIt has a number of attributes that control its appearance
- To access them, first create a **plot attributes object** by calling a function PlotNameAttributes(), e.g., PseudocolorAttributes(), or VolumeAttributes()
- If changing attributes, pass the object to the SetPlotOptions()
- If setting new defaults, pass the object to SetDefaultPlotOptions()

Probing and setting plot attributes (interactive shell)

Note the colour map range in the current plot.
Next, add the following commands:

```
p = PseudocolorAttributes()
p   # will print out all attributes
p.min, p.max = 1, 3   # colour map range
p.minFlag, p.maxFlag = 1, 1   # turn it on
SetPlotOptions(p) # set active plot attributes
help(SetPlotOptions)
```

Revert to the original colour map range:

```
p.minFlag, p.maxFlag = 0,0   # turn it off
SetPlotOptions(p)
```

Pick a different colour map:

```
p.colorTableName = "Greens" # new colour map
SetPlotOptions(p)
```

Probing and setting plot attributes (interactive shell)

Note the colour map range in the current plot.
Next, add the following commands:

```
p = PseudocolorAttributes()
p   # will print out all attributes
p.min, p.max = 1, 3   # colour map range
p.minFlag, p.maxFlag = 1, 1   # turn it on
SetPlotOptions(p) # set active plot attributes
help(SetPlotOptions)
```

Revert to the original colour map range:

```
p.minFlag, p.maxFlag = 0,0   # turn it off
SetPlotOptions(p)
```

Pick a different colour map:

```
p.colorTableName = "Greens" # new colour map
SetPlotOptions(p)
```

Probing and setting plot attributes (interactive shell)

Note the colour map range in the current plot.
Next, add the following commands:

```
p = PseudocolorAttributes()
p   # will print out all attributes
p.min, p.max = 1, 3    # colour map range
p.minFlag, p.maxFlag = 1, 1    # turn it on
SetPlotOptions(p) # set active plot attributes
help(SetPlotOptions)
```

Revert to the original colour map range:

```
p.minFlag, p.maxFlag = 0,0    # turn it off
SetPlotOptions(p)
```

Pick a different colour map:

```
p.colorTableName = "Greens" # new colour map
SetPlotOptions(p)
```

Running scriptName.py from inside GUI

- Option 1: paste the code into Controls → Command... window and click Execute
- Option 2: inside the Python shell change to the directory containing your scripts (can use relative or absolute paths) and source your script

```
os.getcwd()    # to check the current directory
os.chdir('/Users/razoumov/teaching/visitWorkshop/scripts')
# os.chdir('C:\Users\Josh\Desktop\20130216')    # Windows example
Source('scriptName.py')
```

Setting attributes before drawing

With *noise.silo* loaded, let's draw a plot:

```
# this is orange.py
DeleteAllPlots()
AddPlot("Pseudocolor", "hardyglobal")
p = PseudocolorAttributes()
p.colorTableName = "Oranges"
SetPlotOptions(p)
DrawPlots()
```

Scripting an operator

With *noise.silo* loaded, run the following:

```
# this is addOperator.py
DeleteAllPlots()
AddPlot("Pseudocolor", "hardyglobal")
AddOperator("Isosurface")
isoAtts = IsosurfaceAttributes() # create an operator attributes object
isoAtts.contourMethod = isoAtts.Level  # contour by level(s)
isoAtts.variable = "hardyglobal"
SetOperatorOptions(isoAtts) # set operator attributes to above values
DrawPlots()
print isoAtts   # default is 10 isosurface levels
```

Now let's produce 3 single-isosurface plots at *hardyglobal = 2., 3.5, 5.* respectively:

```
# this is threeSurfaces.py
isoAtts.contourMethod = isoAtts.Value  # contour by value(s)
for i in range(3):
    isoAtts.contourValue = 2. + i*1.5
    SetOperatorOptions(isoAtts)
```

These images play back, but aren't saved to disk ...

Scripting an operator

With *noise.silo* loaded, run the following:

```
# this is addOperator.py
DeleteAllPlots()
AddPlot("Pseudocolor", "hardyglobal")
AddOperator("Isosurface")
isoAtts = IsosurfaceAttributes() # create an operator attributes object
isoAtts.contourMethod = isoAtts.Level  # contour by level(s)
isoAtts.variable = "hardyglobal"
SetOperatorOptions(isoAtts) # set operator attributes to above values
DrawPlots()
print isoAtts   # default is 10 isosurface levels
```

Now let's produce 3 single-isosurface plots at *hardyglobal = 2., 3.5, 5.*, respectively:

```
# this is threeSurfaces.py
isoAtts.contourMethod = isoAtts.Value  # contour by value(s)
for i in range(3):
    isoAtts.contourValue = 2. + i*1.5
    SetOperatorOptions(isoAtts)
```

These images play back, but aren't saved to disk ...

Saving images to disk

```
s = SaveWindowAttributes()
s.format = s.PNG
s.fileName = 'someName'
s.outputToCurrentDirectory = 0    # for some reason this is 'yes'
s.outputDirectory = "/path/to/directory"
SetSaveWindowAttributes(s)
...
build and display a plot
...
name = SaveWindow()    # returns the name of the file it wrote
```

Now let's save the three surfaces to disk:

```
# this is saveSurfaces.py
s = SaveWindowAttributes()
s.format, s.fileName, s.outputToCurrentDirectory = s.PNG, 'iso', 0
s.outputDirectory = "/Users/razoumov/Documents/teaching/visitWorkshop"
SetSaveWindowAttributes(s)
for i in range(3):
    isoAtts.contourValue = 2. + i*1.5
    SetOperatorOptions(isoAtts)
    name = SaveWindow()
```

Saving images to disk

```
s = SaveWindowAttributes()
s.format = s.PNG
s.fileName = 'someName'
s.outputToCurrentDirectory = 0   # for some reason this is 'yes'
s.outputDirectory = "/path/to/directory"
SetSaveWindowAttributes(s)
...
build and display a plot
...
name = SaveWindow()   # returns the name of the file it wrote
```

Now let's save the three surfaces to disk:

```
# this is saveSurfaces.py
s = SaveWindowAttributes()
s.format, s.fileName, s.outputToCurrentDirectory = s.PNG, 'iso', 0
s.outputDirectory = "/Users/razoumov/Documents/teaching/visitWorkshop"
SetSaveWindowAttributes(s)
for i in range(3):
    isoAtts.contourValue = 2. + i*1.5
    SetOperatorOptions(isoAtts)
    name = SaveWindow()
```

Animating camera position: create a plot

With *noise.silo* loaded, draw a single isosurface at *hardyglobal = 3.8* in green:

```
# this is oneSurface.py
DeleteAllPlots()
AddPlot('Contour', 'hardyglobal')
contAtt = ContourAttributes()
contAtt.contourMethod = contAtt.Value
contAtt.contourValue = (3.8)
contAtt.colorType = contAtt.ColorBySingleColor
contAtt.singleColor = (0, 255, 0, 255)
SetPlotOptions(contAtt)
DrawPlots()
```

```
# this is printView.py
print GetView3D()   # print all its attributes of the current view
# GetView3D().viewNormal   # can also print a single attribute
```

Animating camera position: create a plot

With *noise.silo* loaded, draw a single isosurface at *hardyglobal = 3.8* in green:

```
# this is oneSurface.py
DeleteAllPlots()
AddPlot('Contour', 'hardyglobal')
contAtt = ContourAttributes()
contAtt.contourMethod = contAtt.Value
contAtt.contourValue = (3.8)
contAtt.colorType = contAtt.ColorBySingleColor
contAtt.singleColor = (0, 255, 0, 255)
SetPlotOptions(contAtt)
DrawPlots()
```

```
# this is printView.py
print GetView3D()    # print all its attributes of the current view
# GetView3D().viewNormal    # can also print a single attribute
```

Animating camera position: set a view by hand

Create a view by hand by explicitly setting the important attributes:

```
# this is setControlPoint.py
from math import *
c0 = View3DAttributes()
phi = 0    # 0 <= phi <= 2*pi
theta = 0    # -pi/2 <= theta <= pi/2
c0.viewNormal = (cos(theta)*cos(phi),cos(theta)*sin(phi),sin(theta))
c0.focus, c0.viewUp = (0, 0, 0), (0, 0, 1)
c0.viewAngle, c0.parallelScale, c0.imageZoom = 30, 17.3205, 1
c0.nearPlane, c0.farPlane, c0.perspective = -34.641, 34.641, 1
SetView3D(c0)
```

Note: with a trackpad, the zoom scroll is not very smooth, but we can always set the zoom level with

```
vatts = View3DAttributes()
vatts.imageZoom = 3
SetView3D(vatts)
```

or via Controls → View... and setting Image zoom in the GUI

Animating camera position: set a view by hand

Create a view by hand by explicitly setting the important attributes:

```
# this is setControlPoint.py
from math import *
c0 = View3DAttributes()
phi = 0    # 0 <= phi <= 2*pi
theta = 0    # -pi/2 <= theta <= pi/2
c0.viewNormal = (cos(theta)*cos(phi),cos(theta)*sin(phi),sin(theta))
c0.focus, c0.viewUp = (0, 0, 0), (0, 0, 1)
c0.viewAngle, c0.parallelScale, c0.imageZoom = 30, 17.3205, 1
c0.nearPlane, c0.farPlane, c0.perspective = -34.641, 34.641, 1
SetView3D(c0)
```

Note: with a trackpad, the zoom scroll is not very smooth, but we can always set the zoom level with

```
vatts = View3DAttributes()
vatts.imageZoom = 3
SetView3D(vatts)
```

or via Controls → View... and setting Image zoom in the GUI

Animating camera: rotate around the vertical axis

```
# this is rotateAroundVertical.py
nsteps = 300
for i in range(nsteps):
    phi = float(i)/float(nsteps-1)*2.*pi
    c0.viewNormal = (cos(theta)*cos(phi), cos(theta)*sin(phi),
                     sin(theta))
    SetView3D(c0)
```

Animating camera: fly into the volume and out

```
# this is flyInOut.py
nsteps = 100
xfirst = 0
xlast = -40
for i in range(nsteps):
    x = xfirst + float(i)/float(nsteps-1)*(xlast-xfirst)
    c0.focus = (x, 0, 0)
    SetView3D(c0)
for i in range(nsteps):
    x = xlast + float(i)/float(nsteps-1)*(xfirst-xlast)
    c0.focus = (x, 0, 0)
    SetView3D(c0)
```

Animating camera: play the perspective angle

```
# this is perspective.py
nsteps = 100
a1 = 30
a2 = 60
for i in range(nsteps):
    c0.viewAngle = a1 + float(i)/float(nsteps-1)*(a2-a1)
    SetView3D(c0)
for i in range(nsteps):
    c0.viewAngle = a2 + float(i)/float(nsteps-1)*(a1-a2)
    SetView3D(c0)
```

Camera animation: interpolate between control points

First, define a function to copy all attributes from one control point to another

```
# this is copyView.py
def copyView(a,b):
    b.viewNormal = a.viewNormal
    b.focus = a.focus
    b.viewUp = a.viewUp
    b.viewAngle = a.viewAngle
    b.parallelScale = a.parallelScale
    b.nearPlane = a.nearPlane
    b.farPlane = a.farPlane
    b.perspective = a.perspective
    b.imageZoom = a.imageZoom
```

Camera animation: interpolate between control points

Next, set three new control points, based on c0

```
# this is threeControlPoints.py
c1 = View3DAttributes()
copyView(c0,c1)
phi = pi/2
c1.viewNormal = (cos(theta)*cos(phi), cos(theta)*sin(phi),
                 sin(theta))

c2 = View3DAttributes()
copyView(c1,c2)
theta = pi/6
c2.viewNormal = (cos(theta)*cos(phi), cos(theta)*sin(phi),
                 sin(theta))

c3 = View3DAttributes()
copyView(c2,c3)
c3.focus = (0, -30, -20)
```

Camera animation: interpolate between control points

Finally, interpolate between the control points with a small step

```
# this is interpolate.py
# define a tuple of control points
cpts = (c0, c1, c2, c3)

# define a corresponding tuple of their positions in time
# from 0 to 1, in this case (0, 1/3, 2/3, 1)
x, n = [], len(cpts)
for i in range(n):
    x = x + [float(i) / float(n-1)]

# interpolate between control points to cover [0,1]
# time interval with a much smaller step
nsteps = 200
for i in range(nsteps):
    t = float(i) / float(nsteps - 1)
    c = EvalCubicSpline(t, x, cpts)
    SetView3D(c)
```

Animating an operator: no animation yet

```
# this is clipStatic.py
DeleteAllPlots()
AddPlot("Pseudocolor", "hardyglobal")
c0 = View3DAttributes()
c0.viewNormal = (0.9, 0., 0.4358898943540673)
c0.focus, c0.viewUp = (0, 0, 0), (0, 0, 1)
c0.viewAngle, c0.parallelScale = 30, 17.3205
c0.nearPlane, c0.farPlane, c0.perspective = -171.473, 171.473, 1
SetView3D(c0)

light0 = LightAttributes()
light0.enabledFlag, light0.type = 1, light0.Camera
light0.direction = (0., -0.6, -0.8)
light0.color, light0.brightness = (255, 255, 255, 255), 1
SetLight(0, light0)

AddOperator("Clip")
clipAtts = ClipAttributes()
clipAtts.funcType, clipAtts.plane1Status = clipAtts.Plane, 1
clipAtts.plane1Origin, clipAtts.plane1Normal = (0, 0, 0), (0, 0, 1)
SetOperatorOptions(clipAtts)
DrawPlots()
```

Animating an operator: exercise

Exercise: Try to do the following:

(1) Modify the previous slide's script to animate the clip plane through the volume from bottom to top

(2) Write each image to disk as PNG

(3) Use a third-party tool to merge these into a movie; e.g., in Linux/MacOSX can use ffmpeg to merge frames into an efficiently compressed Quicktime-compatible MP4

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

Scripting queries: minMax of Pseudocolor

Controls → Query... produces a query dialogue with dozens of options

```
# this is queryPseudocolor.py
DeleteAllPlots()
AddPlot("Pseudocolor","hardyglobal")
DrawPlots()
print Query("MinMax")
val = GetQueryOutputValue()
print val
```

```
hardyglobal -- Min = 1.09554 (node 105026 at coord <0.612245, -10, 7.14286>)
hardyglobal -- Max = 5.88965 (node 83943 at coord <7.55102, 1.42857, 3.46939>)
(1.0955432653427124, 5.889651775360107)
```

Now try commenting out DrawPlots() and running the script again

```
VisIt: Error - MinMax requires an active non-hidden Plot.
Please select a plot and try again.
```

... so, do we query the original data or the plot?

Scripting queries: minMax of Pseudocolor

Controls → Query... produces a query dialogue with dozens of options

```
# this is queryPseudocolor.py
DeleteAllPlots()
AddPlot("Pseudocolor","hardyglobal")
DrawPlots()
print Query("MinMax")
val = GetQueryOutputValue()
print val
```

```
hardyglobal -- Min = 1.09554 (node 105026 at coord <0.612245, -10, 7.14286>)
hardyglobal -- Max = 5.88965 (node 83943 at coord <7.55102, 1.42857, 3.46939>)
(1.0955432653427124, 5.889651775360107)
```

Now try commenting out DrawPlots() and running the script again

```
VisIt: Error - MinMax requires an active non-hidden Plot.
Please select a plot and try again.
```

... so, do we query the original data or the plot?

Scripting queries: minMax of Contour

- Let's query an isosurface plot:

```
# this is queryContour.py
DeleteAllPlots()
AddPlot("Contour","hardyglobal")
contAtts = ContourAttributes()
contAtts.contourMethod = contAtts.Value
contAtts.contourValue = (3.8)
SetPlotOptions(contAtts)
DrawPlots()
print Query("MinMax")
val = GetQueryOutputValue()
print val
```

- Produces exactly the same query output!
- ✔ We definitely query the original 3D data, not the plot.
- ☞ Why require a plot when we run a query not on the plot but on the original data?...

Scripting queries: minMax of Contour

- Let's query an isosurface plot:

```
# this is queryContour.py
DeleteAllPlots()
AddPlot("Contour","hardyglobal")
contAtts = ContourAttributes()
contAtts.contourMethod = contAtts.Value
contAtts.contourValue = (3.8)
SetPlotOptions(contAtts)
DrawPlots()
print Query("MinMax")
val = GetQueryOutputValue()
print val
```

- Produces exactly the same query output!

✔ We definitely query the original 3D data, not the plot.

☞ Why require a plot when we run a query not on the plot but on the original data?...

Scripting queries: weighted variable sum of Slice

Answer: query script authors can make it operate on *anything in the pipeline*, so best to check documentation and/or test your script

```
# this is querySlice.py
DeleteAllPlots()
AddPlot("Pseudocolor", "hardyglobal")
AddOperator("Slice")
DrawPlots()
for i in range(10):
    position = i*2 - 9
    print 'position =', position
    s = SliceAttributes()
    s.axisType = s.XAxis
    s.originType = s.Intercept
    s.originIntercept = position
    SetOperatorOptions(s)
    Query("MinMax")    # queries the 3D volume!
    print '  minMax =', GetQueryOutputValue()
    Query("Weighted Variable Sum")   # queries the 2D slice!
    print '  sum =', GetQueryOutputValue()
```

Recording GUI actions to Python scripts

- Controls → Command... window lets you convert your GUI workflow into a Python code (similar to ParaView's *Trace Tool*)
  (1) delete all plots and databases
  (2) select Controls → Command... window to open the Commands window
  (3) press Record
  (4) load the file *noise.silo*, add a plot, draw it
  (5) press Stop
  (6) you'll see an automatically generated script in the Commands window

- Often the output will be very verbose and contain many unnecessary commands, which can be edited out
  - **exercise:** try translating object rotation around an axis into Python; which variables are important and which ones are not?

- Some GUI operations will not be recorded the way you expect

Scripting a time-sequence animation

- One of them is File → Save movie... recording which will produce an identical script for each frame, resulting in a very ... long code
- In this case, you want to record a single frame and then wrap everything into a Python loop
  - for time-dependent datasets at each loop iteration make sure to
    (1) read a new file and
    (2) write a new image
- **Exercise:** Script a movie from the time-dependent aneurysm data from `http://bit.ly/2dTxkqx` (∼361 MB) or from a simpler 2D dataset `datasets/evolution/2d*.vtk` from `http://bit.ly/visitfiles` (∼24 MB)

Other places to use Python in VisIt

- Python Expression Editor (mentioned earlier)
- Python Query Editor (mentioned earlier)
- Setting up your own buttons in the VisIt GUI, creating other custom Qt GUIs based on VisIt
- Setting up callbacks that get called whenever events happen in VisIt
  - requires GUI and Python interface running at the same time
  - example 1: as you move the time slider by hand, have the position of a slice plane adjust automatically, or have your visualization window pan and/or zoom in automatically on different regions of interest
  - example 2: click on your visualization with Pick and have a script process coordinates of the picked points and produce something interesting based on that
  - more advanced topic for another time

Could be useful: plotting 2D terrain in 3D

- Natural Resources Canada provides free topographic maps for the entire country `http://bit.ly/2dDcywN`
- We'll use one of their 1:50,000 maps showing a part of coastal BC near Vancouver
- The file is a 2D digital elevation map (DEM) file – VisIt can understand DEM files natively

```
OpenDatabase("~/teaching/visitWorkshop/datasets/092g06.dem")
AddPlot("Pseudocolor", "height")
DrawPlots()
```

Wouldn't it be nice to plot it in 3D?

3D terrain

```python
# this is terrain3d.py
DeleteAllPlots()
OpenDatabase("~/teaching/visitWorkshop/datasets/092g06.dem")
AddPlot("Pseudocolor", "height")

AddOperator("Elevate")
e = ElevateAttributes()
e.useXYLimits = 1 # if X/Y are longitude/latitude, z-height would be off
SetOperatorOptions(e) #    => simply rescale all 3 axes to a cube

AddOperator("Transform")
t = TransformAttributes()
t.doScale = 1       # turn on scaling
t.scaleX, t.scaleY, t.scaleZ = 1, 1, 0.05    # and make z-heights smaller
SetOperatorOptions(t)

DrawPlots()
```

- DEM files are raster images
- ESRI shapefiles with vector data (roads, buildings, etc) can also be converted to 3D to be plotted on top of terrain, details at `http://bit.ly/2eoAzaj`

3D terrain

```
# this is terrain3d.py
DeleteAllPlots()
OpenDatabase("~/teaching/visitWorkshop/datasets/092g06.dem")
AddPlot("Pseudocolor", "height")

AddOperator("Elevate")
e = ElevateAttributes()
e.useXYLimits = 1 # if X/Y are longitude/latitude, z-height would be off
SetOperatorOptions(e) #   => simply rescale all 3 axes to a cube

AddOperator("Transform")
t = TransformAttributes()
t.doScale = 1       # turn on scaling
t.scaleX, t.scaleY, t.scaleZ = 1, 1, 0.05   # and make z-heights smaller
SetOperatorOptions(t)

DrawPlots()
```

- DEM files are raster images
- ESRI shapefiles with vector data (roads, buildings, etc) can also be converted to 3D to be plotted on top of terrain, details at `http://bit.ly/2eoAzaj`

Molecular visualization

- VisIt can read LAMMPS, PDB (Protein Data Bank), XYZ files, and a few other molecular structure file formats
- Molecular options are very basic compared to VMD
- Rendering 10^6 atoms at medium quality takes 4.5 s on my laptop, so with scripting it is feasible to render $\sim 64 \times 10^6$ atoms of a virus shell with few minutes per frame
- More details on molecular data features of VisIt at `http://bit.ly/2epWHn3`

Molecular visualization

Let's try a molecule with 12,837 atoms:

```
# this is drawMolecule.py
OpenDatabase("~/teaching/visitWorkshop/datasets/molecules/1l5q.pdb", 0)
AddPlot("Molecule", "element", 1, 1)
DrawPlots()

m = MoleculeAttributes()

m.drawAtomsAs = m.SphereAtoms # NoAtoms, SphereAtoms, ImposterAtoms
m.scaleRadiusBy = m.Fixed   # Fixed, Covalent, Atomic, Variable
m.atomSphereQuality = m.Medium   # Low, Medium, High, Super
m.radiusFixed = 0.5

m.drawBondsAs = m.CylinderBonds # NoBonds, LineBonds, CylinderBonds
m.colorBonds = m.ColorByAtom   # ColorByAtom, SingleColor
m.bondCylinderQuality = m.Medium   # Low, Medium, High, Super
m.bondRadius = 0.08

m.elementColorTable = "cpk_jmol"
m.legendFlag = 1
SetPlotOptions(m)
```

Conclusions for scripting part

- VisIt's Python interface is very concise and clean (good!)
- Access attributes by creating an attributes object through PlotNameAttributes(), OperatorNameAttributes(), View3DAttributes(), SaveWindowAttributes(), LightAttributes()
- Every time you change attributes, don't forget to set them with SetPlotOptions(), SetOperatorOptions(), SetView3D(), SetSaveWindowAttributes(), SetLight()
- Several ways to use Python
  - command line: `/path/to/VisIt -nowin -cli -s script.py`
  - Python shell: Controls → Launch CLI...
  - Python editor: Controls → Command...
- Use the built-in recorder to produce Python scripts from scratch Controls → Command... → Record

Remote and distributed visualization with VisIt

Visualizing remote data (interactively or not)

So far we covered working with standalone VisIt on your desktop. If your dataset is on cluster.consortium.ca, you have many options:

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

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

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

- any node with X11 server (for VNC only); scheduled or a login/head/development node with/without a GPU

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

(5) run VisIt via a GUI-less batch script (interactively or scheduled) – ideal for large routine visualizations

X11 forwarding

- Need a client-side X11 server (comes by default on Linux and Mac laptops) to which a remote application sends its window
- *ssh -X* lets you forward X11 graphics and mouse/keyboard interactions through ssh (encrypted!)
- Can forward through several consecutive connections, e.g.,
  laptop $\xrightarrow{\text{ssh } -\text{X}}$ login node $\xrightarrow{\text{qsub } -\text{X}}$ interactive compute node
- X11 forwarding is very chatty (lots of roundtrips!) and can be very slow on a high-latency network ... in general we don't recommend it

X11 forwarding

- Need a client-side X11 server (comes by default on Linux and Mac laptops) to which a remote application sends its window
- *ssh -X* lets you forward X11 graphics and mouse/keyboard interactions through ssh (encrypted!)
- Can forward through several consecutive connections, e.g.,
  laptop $\xrightarrow{\text{ssh -X}}$ login node $\xrightarrow{\text{qsub -X}}$ interactive compute node
- X11 forwarding is very chatty (lots of roundtrips!) and can be very slow on a high-latency network ... in general we don't recommend it

VNC (Virtual Network Computing)

- Remote graphical desktop system
- Has gone to tremendous effort to optimize communication via data compression and caching
- X11 server is on the remote side
- Does not handle user authentication (by itself not secure)
  - best to run VNC server on either (1) an externally inaccessible compute node or (2) a login node with a really good firewall not allowing any incoming connection on the VNC port
  - in both cases need to set up an SSH tunnel between the VNC ports to connect (virtually without a performance drop)
  - in addition, always employ a non-empty VNC password for higher security
- Your remote collaborators can connect to the same VNC session with full keyboard/mouse control as long as they have the VNC password (different from your cluster password which should never be shared!)
- Setting it up is a little bit more involved but well worth it

x2go: an alternative to VNC

- Also remote desktop like VNC, but there are some differences:
  - windows are managed by the client-side (laptop's) X11 server
  - x2go server can be run system-wide for all users, supports user authentication $\Rightarrow$ can easily be run on the login node for all users
  - persistent sessions (can reconnect to a suspended desktop)
- Open-source implementation of *NX protocol*
- X2go server must be on Linux
- Client could be on Linux, Mac, Windows

New national systems

New clusters **Cedar** (SFU) and **Graham** (Waterloo) online in ∼May

- `https://docs.computecanada.ca/wiki/Cedar`
  27,696 CPU cores and 584 GPUs
- `https://docs.computecanada.ca/wiki/Graham`
  33,576 CPU cores and 320 GPUs

We are aiming to implement an interactive visualization setup on several nodes on these cluster, details yet to be determined

- how many nodes exactly
- whether accessible directly from outside (likely!)
- whether with GPUs
- if yes, how to share individual GPUs among multiple users

In addition, users will be able to run batch-mode (non-interactive) visualizations on regular compute (CPU and/or GPU) nodes via the job scheduler

Remote VisIt via VNC on WestGrid (page 1 of 2)

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

(1) Install TigerVNC (`http://tigervnc.org`) or TurboVNC (`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 (don't leave it empty) – you'll use it in step 6

(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
It'll produce something like *"New 'X' desktop is cn0553:1"*, where the syntax is *nodeName:displayNumber*

Remote VisIt via VNC on WestGrid (page 2 of 2)

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

(5) On your desktop **set up ssh forwarding** to the VNC port on the compute node:
ssh username@parallel.westgrid.ca -L xxxx:cn0553:yyyy
Here xxxx = 5901 is the local VNC port, and yyyy = 5900 (VNC's default) + *displayNumber* and is usually 5901 as well

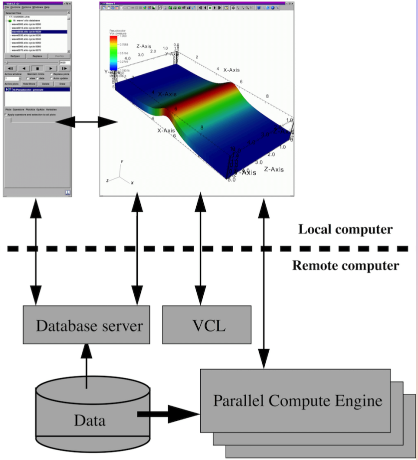
(6) **Start TurboVNC viewer** on your desktop, enter *localhost:1* (that's xxxx-5900) and then enter the password from step 2 above

(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 VisIt with a VirtualGL wrapper**
vglrun /global/software/visit/visit271/bin/visit

Client-server VisIt in a Cloud West VM (page 1 of 2)

more details at `http://bit.ly/2kUTCNL`



- Your local VisIt will start remote VCL (VisIt Component Launcher) responsible for launching other remote VisIt components
- On your laptop (VisIt client) set up Host and Launch profiles (could run on a server/login node or even launch a serial/parallel VisIt job via sqsub + mpirun)
  - for a cloud VM only need a host profile
  - don't need a GPU for rendering (most cloud VMs don't have one!)
- Ports 5600 - 5609 should be open throughout
- Once set up, to connect simply open a data file on the remote system

Client-server VisIt in a Cloud West VM (page 2 of 2)

more details at `http://bit.ly/2kUTCNL`

Prerequisites:

- your own cloud VM `https://docs.computecanada.ca/wiki/CC-Cloud`
- a bunch of system dependencies for compiling VisIt
- a copy of VisIt compiled with Python, Mesa (open-source OpenGL implementation supporting software rendering), support for your input file format – usually need to compile your own

(1) [Options] → [Host profiles...] to set nickname (`cloud west`), host name (VM's public IP address), path to remote VisIt installation (`/home/centos/visit`), username (`centos`), tunnel through ssh

(2) [Options] → [Save Settings]

(3) [File] → [Open file...] → [Host=] cloud west

Batch scripting on HPC (page 1 of 2)

example on parallel.westgrid.ca's GPU node

Example 1: serial rendering via a scheduled interactive job

```
qsub -q interactive -I -l nodes=1:ppn=1:gpus=1,walltime=1:00:00
... wait for an interactive shell ...
firstgpu=$( head -n 1 "$PBS_GPUFILE" )
gpuindex=${firstgpu:(-1)}
export DISPLAY=:0.$gpuindex
/global/software/visit/visit271/bin/visit -nowin -cli -s script.py
--- script.py ---
your debugged VisIt Python script
```

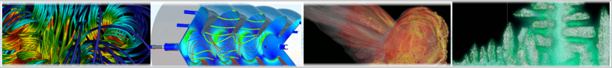
Batch scripting on HPC (page 2 of 2)

example on parallel.westgrid.ca's GPU node

Example 2: parallel rendering via a scheduled batch job

```
qsub -q interactive ./visualization.sh
--- visualization.sh ---
#!/bin/bash
#PBS -S /bin/bash
#PBS -q gpu
#PBS -l nodes=1:ppn=4:gpus=1
#PBS -l pmem=2000mb
#PBS -l walltime=01:00:00
cd $PBS_O_WORKDIR
firstgpu=$( head -n 1 "$PBS_GPUFILE" )
gpuindex=${firstgpu:(-1)}
export DISPLAY=:0.$gpuindex
/global/software/visit/visit271/bin/visit -np 4 -nowin -cli -s script.py
--- script.py ---
your debugged VisIt Python script
```

VisIt resources



- Website
  https://wci.llnl.gov/simulation/computer-codes/visit
  https://wci.llnl.gov/codes/visit
- Documentation
  https://wci.llnl.gov/simulation/computer-codes/visit/manuals
- Gallery
  https://wci.llnl.gov/simulation/computer-codes/visit/gallery
- Visit users' wiki
  http://www.visitusers.org
- Tutorials
  http://www.visitusers.org/index.php?title=VisIt_Tutorial
- Examples datasets
  http://www.visitusers.org/index.php?title=Tutorial_Data

Online WestGrid visualization webinars

- Bimonthly during the academic year (January, March, May, September, November), advertised at `https://www.westgrid.ca`
- One-hour long, usually very specific topics
- Past webinars are available with slides and video at `https://www.westgrid.ca/events/archive`
  - "Introduction to batch visualization"
  - "Graph visualization with Gephi"
  - "3D graphs with NetworkX, VTK, and ParaView"
  - "CPU-based rendering with OSPRay"
  - "Scripting and other advanced topics in VisIt visualization"
  - "Visualization support in WestGrid / Compute Canada"
  - "Using ParaViewWeb for 3D visualization and data analysis in a web browser"
  - coming up: "3D visualization on new CC systems"
- We are looking for topic suggestions!

Non-VisIt resources

- `http://www.paraview.org/Wiki/The_ParaView_Tutorial`

- `https://www.westgrid.ca/support/visualization`
- email support@westgrid.ca

- `http://bit.ly/cctopviz`
- `https://docs.computecanada.ca/wiki/Visualization`
- support@computecanada.ca
- email vis-support@computecanada.ca

Questions?