

# Scripting and other advanced topics in VisIt visualization

Alex Razoumov  
alex.razoumov@westgrid.ca

WestGrid / Compute Canada

- slides and scripts at <http://bit.ly/visitscripting> (a small ZIP file)
- main data file already included into VisIt

```
$ find /Applications/VisIt.app -name noise.silo  
/Applications/VisIt.app/Contents/Resources/data/noise.silo
```

if you can't find it, check <http://bit.ly/visitfiles> (~24 MB)

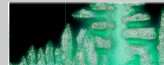
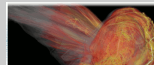
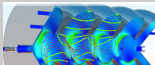
# 2D/3D visualization packages

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**

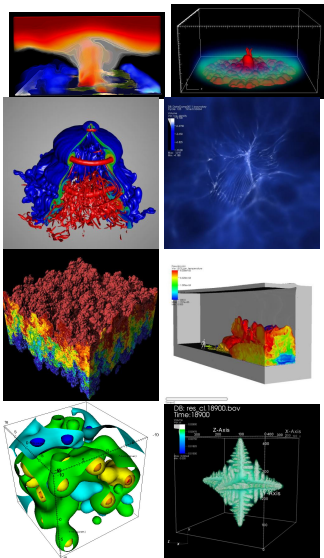


- Full-day VisIt workshop at UBC on Nov-03  
[https://www.westgrid.ca/events/data\\_visualization\\_workshop\\_university\\_british\\_columbia](https://www.westgrid.ca/events/data_visualization_workshop_university_british_columbia)
- Full-day VisIt workshop at UVic on Nov-08  
[https://www.westgrid.ca/events/data\\_visualization\\_workshop\\_university\\_victoria](https://www.westgrid.ca/events/data_visualization_workshop_university_victoria)
- Contact me if you want a VisIt or ParaView workshop at your institution (they are always free!)
  - ▶ rotating through major WestGrid partner universities



# 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, maintained by LLNL
- Currently ~ 20 developers from different organizations and universities
- v2.11 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>
- Uses MPI for [distributed-memory parallelism](#) on HPC clusters





# 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 our laptop

# 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 = 30, 17.3205
c0.nearPlane, c0.farPlane = -34.641, 34.641
c0.perspective = 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 = 30, 17.3205
c0.nearPlane, c0.farPlane = -34.641, 34.641
c0.perspective = 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
```

# 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/2dTxxqx> (~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

## 2D terrain

- 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_0100_deme.dem")  
AddPlot("Pseudocolor", "height")  
DrawPlots()
```

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



# 3D terrain

```
# this is terrain3d.py
DeleteAllPlots()
OpenDatabase("~/teaching/visitWorkshop/datasets/092g06_0100_deme.dem")
AddPlot("Pseudocolor", "height")

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

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

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_0100_deme.dem")
AddPlot("Pseudocolor", "height")

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

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

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/115q.pdb", 0)
AddPlot("Molecule", "element", 1, 1)
DrawPlots()
```

```
MoleculeAtts = MoleculeAttributes()
```

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

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

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

# Conclusions

- VisIt's Python interface is very concise and clean (and that's 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`

# Questions?