VisIt-tutorial-Python-scripting

From VisItusers.org

Contents

Command line interface (CLI) overview

VisIt includes a rich a command line interface that is based on Python 2.7.

There are several ways to use the CLI:

1. Launch VisIt in a batch mode and run scripts
   - e.g.: `./visit -nowin -cli -s <script.py>`
2. Launch VisIt so that a visualization window is visible and interactively issue CLI commands
3. Use both the standard graphical user interface (GUI) and CLI simultaneously

Launching the CLI

We will focus on the use case where we have the graphical user interface and CLI simultaneously.

To launch the CLI from the graphical user interface:

1. Go to Controls->Launch CLI
2. This will launch a separate terminal that has the Python interface.

Note that you can also use VisIt's Command window to submit Python code to the CLI. The Command window provides a text editor with Python syntax highlighting and an *Execute* button that tells VisIt to execute the script. Finally, the Command window lets you record your GUI actions into Python code that you can use in your scripts.

A first action in the CLI

1. Open example.silo in the standard GUI if it not already open.
    - You can do also do this in the CLI, using *OpenDatabase()* with the path to the example.silo dataset.

```
AddPlot("Pseudocolor", "temp")
# You will see the active plots list in the GUI update, since the CLI and GUI communicate.
DrawPlots()
#You should see your plot.
```

Tips about Python

1. Python is whitespace sensitive! This is a pain, especially when you are cut-n-pasting things.
2. Python has great constructs for control and iteration, here are some examples:

```
for i in range(100):
   # use i
```

```
# strided range
for i in range(0,100,10):
   # use i
```

```
if (cond):
 # stmt
```

```
import sys
...
sys.exit()
```

Example Scripts

We will be using Python scripts in each of the following sections: You can get them by:

1. Cut-n-paste-ing it into your Python interpreter
2. Or copying it to a separate file and issuing: *Source("/path/to/script/location/script.py")*

For all of these scripts, make sure example.silo is currently open.

Setting Attributes

Each of VisIt's Plots and Operators expose set of *attributes* that control their behavior. In VisIt's GUI, these

attributes are modified via options windows. VisIt's CLI provides a set simple Python objects that control to these attributes.

```
DeleteAllPlots()
AddPlot("Pseudocolor", "temp")
DrawPlots()
p = PseudocolorAttributes()
p.minFlag = 1
p.maxFlag = 1
p.min = 3.5
p.max = 7.5
SetPlotOptions(p)
```

Animating an isosurface

This example demonstrates sweeping an isosurface operator to animate the display of a range of isovalues from example.silo.

```
DeleteAllPlots()
AddPlot("Pseudocolor", "temp")
iso_atts = IsosurfaceAttributes()
iso_atts.contourMethod = iso_atts.Value
iso_atts.variable = "temp"
AddOperator("Isosurface")
DrawPlots()
for i in range(30):
   iso_atts.contourValue = (2 + 0.1*i)
   SetOperatorOptions(iso_atts)
   # For moviemaking, you'll need to save off the image
   # SaveWindow()
```

Rendering a Dataset to a Movie

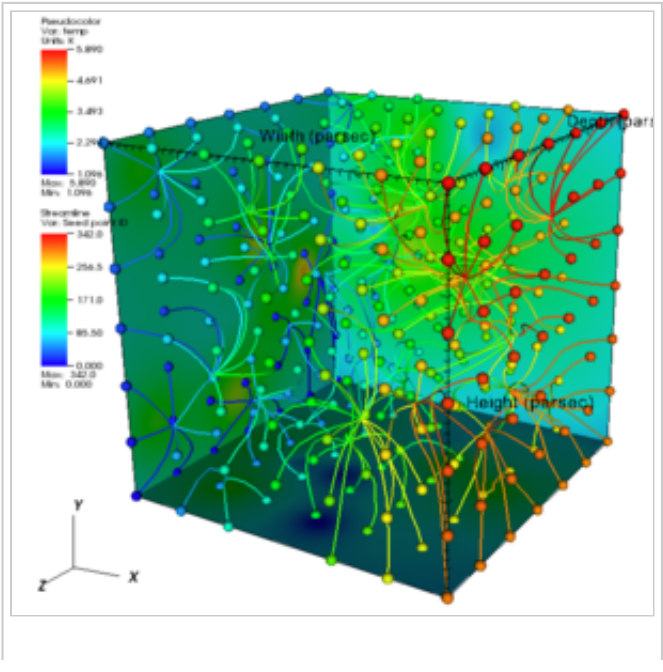
- Open a database, create a plot, render all timesteps and encode a movie.

```
# import visit_utils, we will use it to help encode our movie
from visit_utils import *

OpenDatabase("aneurysm.visit")
AddPlot("Pseudocolor","pressure")
DrawPlots()

# Set a better view
ResetView()
v = GetView3D()
v.RotateAxis(1,90)
SetView3D(v)

# get the number of timesteps
nts = TimeSliderGetNStates()

# set basic save options
swatts = SaveWindowAttributes()
#
# The 'family' option controls if visit automatically adds a frame number to
# the rendered files. For this example we will explicitly manage the output name.
#
swatts.family = 0
#
# select PNG as the output file format
```

```
#
swatts.format = swatts.PNG
#
# set the width of the output image
#
swatts.width = 1024
#
# set the height of the output image
#
swatts.height = 1024

for ts in range(0,nts,10): # look at every 10th frame
    # Change to the next timestep
    TimeSliderSetState(ts)
    #before we render the result, explicitly set the filename for this render
    swatts.fileName = "blood_flow_example_%04d.png" % ts
    SetSaveWindowAttributes(swatts)
    # render the image to a PNG file
    SaveWindow()

#################
# use visit_utils.encoding to encode these images into a "wmv" movie
#
# The encoder looks for a printf style pattern in the input path to identify the frames of the movie.
# The frame numbers need to start at 0.
#
# The encoder selects a set of decent encoding settings based on the extension of the
# the output movie file (second argument). In this case we will create a "wmv" file.
#
# Other supported options include ".mpg", ".mov".
#   "wmv" is usually the best choice and plays on all most all platforms (Linux ,OSX, Windows).
#   "mpg" is lower quality, but should play on any platform.
#
# 'fdup' controls the number of times each frame is duplicated.
#  Duplicating the frames allows you to slow the pace of the movie to something reasonable.
#
#################

input_pattern = "blood_flow_example_%04d.png"
output_movie = "blood_flow_example.wmv"
encoding.encode(input_pattern,output_movie,fdup=4)
```

Using all of VisIt's Building Blocks

```
# Clear any previous plots
DeleteAllPlots()
# Create a plot of the scalar field 'temp'
AddPlot("Pseudocolor","temp")
# Slice the volume to show only three
# external faces.
AddOperator("ThreeSlice")
tatts = ThreeSliceAttributes()
tatts.x = -10
tatts.y = -10
tatts.z = -10
SetOperatorOptions(tatts)
DrawPlots()
# Find the maximum value of the field 'temp'
Query("Max")
val = GetQueryOutputValue()
print "Max value of 'temp' = ", val
# Create a streamline plot that follows
# the gradient of 'temp'
DefineVectorExpression("g","gradient(temp)")
AddPlot("Streamline","g")
satts = StreamlineAttributes()
satts.sourceType = satts.SpecifiedBox
```



```
satts.sampleDensity0 = 7
satts.sampleDensity1 = 7
satts.sampleDensity2 = 7
satts.coloringMethod = satts.ColorBySeedPointID
SetPlotOptions(satts)
DrawPlots()
```

Streamlines

Creating a Movie of Animated Streamline Paths

This example extends the "Using all of VisIt's Building Blocks" example by

- animating the paths of the streamlines
- saving images of the animation
- finally, encoding those images into a movie

It is a subset of the Seedme sharing example.

```
# import visit_utils, we will use it to help encode our movie
from visit_utils import *

# Set a better view
ResetView()
v = GetView3D()
v.RotateAxis(0,44)
v.RotateAxis(1,-23)
SetView3D(v)

# Disable annotations
aatts = AnnotationAttributes()
aatts.axes3D.visible = 0
aatts.axes3D.triadFlag = 0
aatts.axes3D.bboxFlag = 0
aatts.userInfoFlag = 0
aatts.databaseInfoFlag = 0
aatts.legendInfoFlag = 0
SetAnnotationAttributes(aatts)

# Set basic save options
swatts = SaveWindowAttributes()
#
# The 'family' option controls if visit automatically adds a frame number to
# the rendered files. For this example we will explicitly manage the output name.
#
swatts.family = 0
#
# select PNG as the output file format
#
swatts.format = swatts.PNG
#
# set the width of the output image
#
swatts.width = 1024
#
# set the height of the output image
#
swatts.height = 1024


####
# Crop streamlines to render them at increasing time values over 50 steps
####
for ts in range(0,50):
    # set the streamline attributes to change the where we crop the streamlines
    satts.displayEnd = ts * .5
```

```
    satts.displayEndFlag = 1
    # update streamline attributes and draw the plot
    SetPlotOptions(satts)
    DrawPlots()
    #before we render the result, explicitly set the filename for this render
    swatts.fileName = "streamline_crop_example_%04d.png" % ts
    SetSaveWindowAttributes(swatts)
    # render the image to a PNG file
    SaveWindow()

################
# use visit_utils.encoding to encode these images into a "wmv" movie
#
# The encoder looks for a printf style pattern in the input path to identify the frames of the movie.
# The frame numbers need to start at 0.
#
# The encoder selects a set of decent encoding settings based on the extension of the
# the output movie file (second argument). In this case we will create a "wmv" file.
#
# Other supported options include ".mpg", ".mov".
#   "wmv" is usually the best choice and plays on all most all platforms (Linux ,OSX, Windows).
#   "mpg" is lower quality, but should play on any platform.
#
# 'fdup' controls the number of times each frame is duplicated.
#  Duplicating the frames allows you to slow the pace of the movie to something reasonable.
#
################

input_pattern = "streamline_crop_example_%04d.png"
output_movie = "streamline_crop_example.wmv"
encoding.encode(input_pattern,output_movie,fdup=4)
```

Animating the camera

See here.

Automating data analysis

See here.

Extracting a per-material aggregate value at each timestep.

See example here.

Recording GUI Actions to Python Scripts

VisIt's Commands window provides a mechanism to translate GUI actions into their equivalent Python commands.

1. Open the Commands Window by selecting *Controls Menu->Command*
2. Press the *Record* button
3. Perform GUI actions
4. Return to the Commands Window
    - Select a tab to hold the python script of your recorded actions
    - Click the *Stop* button.

- The equivalent Python script will be placed in the tab in the Commands window.

- Note that the scripts are very verbose and contain some unnecessary commands, which can be edited out.

Commands window

Learning the CLI

Here are some tips to help you quickly learn how to use VisIt's CLI:

1. From within VisIt's python CLI, you can type "dir()" to see the list of all commands.
   - Sometimes, the output from "dir()" within VisIt's python CLI is a little hard to look through. So, a useful thing on Linux to get nicer list of methods is the following shell command (typed from *outside* VisIt's python CLI)...

```
echo "dir()" | visit -cli -nowin -foro
```

   - Or, if you are looking for CLI functions having to do with a specific thing...

```
echo "dir()" | visit -cli -nowin -foro
```

2. You can learn the syntax of a given method by typing "help(MethodName)"
   - (Type "help(AddPlot)")
3. Use the GUI to Python recording featured outlined above.
4. Use *WriteScript()* function, which will create a python script that describes all of your current plots
   - For more details, see WriteScript
5. When you have a Python object, you can see all of its attributes by printing it.

```
s = SliceAttributes()
print s
# Output:
originType = Intercept  # Point, Intercept, Percent, Zone, Node
originPoint = (0, 0, 0)
originIntercept = 0
originPercent = 0
originZone = 0
originNode = 0
normal = (0, -1, 0)
axisType = YAxis  # XAxis, YAxis, ZAxis, Arbitrary, ThetaPhi
upAxis = (0, 0, 1)
project2d = 1
interactive = 1
flip = 0
originZoneDomain = 0
originNodeDomain = 0
meshName = "default"
theta = 0
```

```
phi = 0
```

Tips for searching for help

VisIt's CLI provides a large set of functions. To can limit the scope of your search using a helpers functions. One such helper is the *lsearch()* function in the visit_utils module:

```
from visit_utils.common import lsearch
lsearch(dir(),"Material")
```

lsearch() returns a python list of strings with the names that match the given pattern. Here is another example that each of the result strings on a separate line.

```
from visit_utils.common import lsearch
for value in lsearch(dir(),"Material"):
    print value
```

Advanced features

1. You can set up your own buttons in the VisIt gui using the CLI. See here.
2. You can set up callbacks in the CLI that get called whenever events happen in VisIt. See here.
3. You can create your own custom Qt GUI that uses VisIt for plotting. See here.

Retrieved from "http://visitusers.org/index.php?title=VisIt-tutorial-Python-scripting"

Category: VisIt Tutorial

- This page was last modified 16:49, 23 June 2015.