13th Summer School on SCIENTIFIC VISUALIZATION

Paraview scripting

Ivan Spisso – i.spisso@cineca.it
SuperComputing Applications and Innovation Department

CINECA

OUTLINE

- Why scripting
- pvbatch and pvpython
- The Python GUI tools
- Accessing a Python Shell
- Create a pipeline
- Python in paraview: The paraview.simple module
- Trace recording
- Macros
- Save ParaView state as a Python script
- Hands-on Python Scripting in ParaView

Why Scripting

http://www.paraview.org/Wiki/ParaView/Users_Guide/Scripting_ParaView

- Automate repetitive tasks (several similar data-sets)
- GUI is not useful or unavailable (batch execution)
- In-situ visualization or co-processing (computing+post-processing at run-time)

The ParaView client provides an easy-to-use GUI in which to visualize data with the standard window, menu, and button controls. Driving ParaView with a mouse is intuitive, but is not easily reproducible and exact as is required for repetitive analysis and scientific result reproducibility. For this type of work, it is much better to use ParaView's scripted interface. This is an an alternative control path that works alongside of, or as a replacement for, the GUI.

pvbatch - pvpython

- Paraview contents 2 command lines able to run a python script:
  - **pvpython**;
  - **pvbatch**;

The differences is that:

- **pvpython** is equivalent to the paraview client where the python interpreter plays the role of the GUI (interactive, serial);
- **pvbatch**: is equivalent to the paraview server where the commands are executed from the python script in the server instead then from a socket plugged to a paraview client (parallel)

The Python GUI tools

http://www.paraview.org/Wiki/Python_GUI_Tools

If ParaView has been compiled with the Python wrapping, some advanced features become available to the user such as:

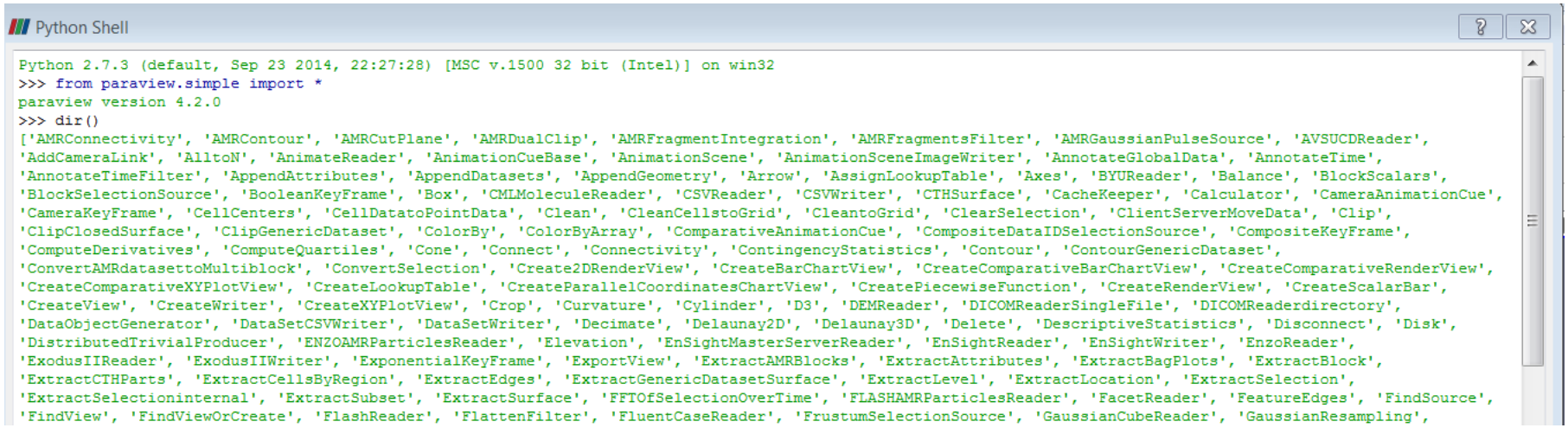
1. Accessing a Python Shell
2. Trace recording
3. Macros
4. Save ParaView state as a Python script

Those features can be reached from the Tools menu for the Shell and Trace access.

Accessing a Python shell

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

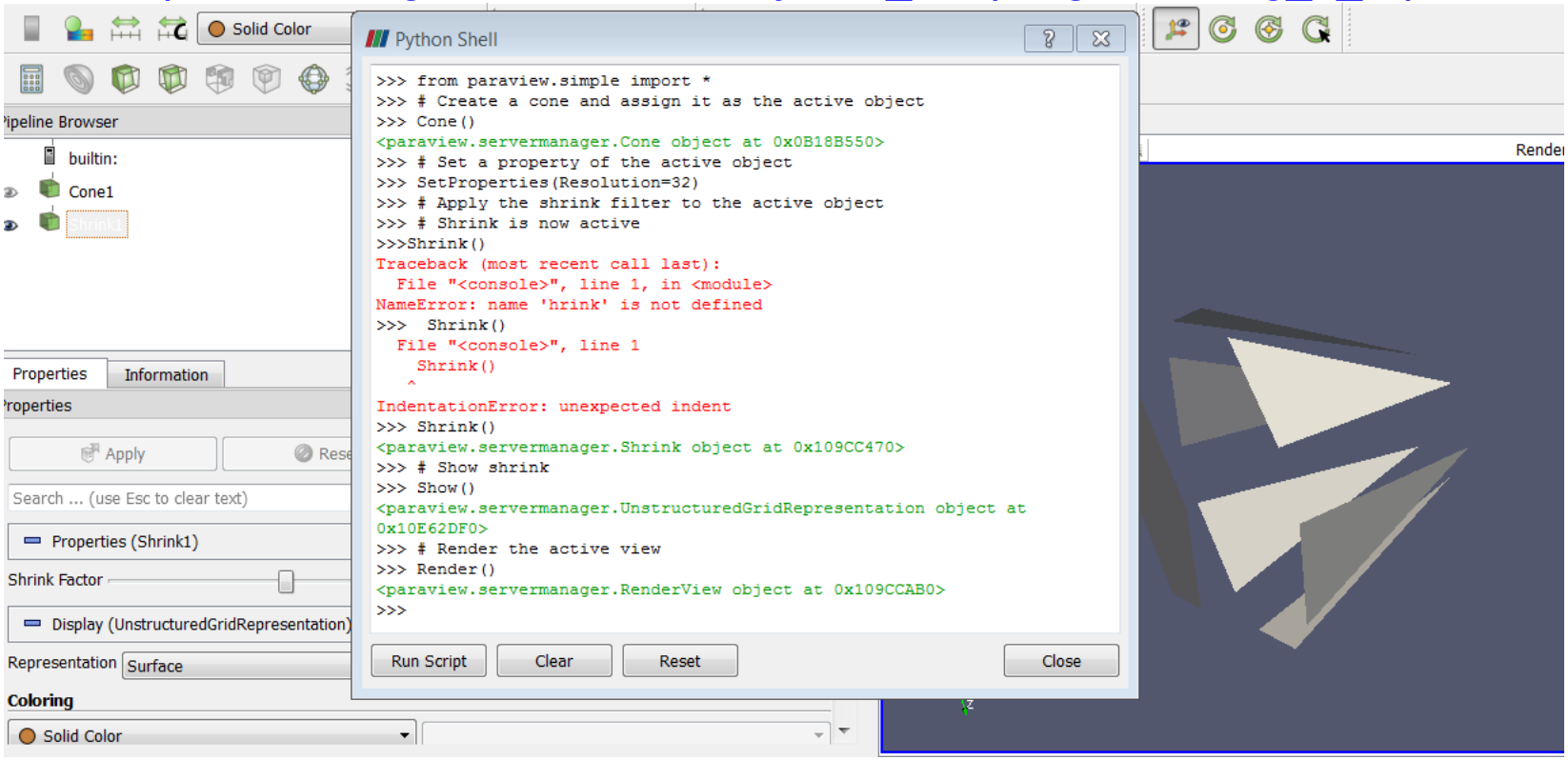
The python interpreter is already runnable from paraview (under *Tools* > *Python Shell*) to perfom interactive applications



Create a Pipeline:

create an object, shrink and render it

http://www.paraview.org/Wiki/ParaView/Python_Scripting#Creating_a_Pipeline



Or run the script cone.py

The paraview.simple module

http://www.paraview.org/Wiki/ParaView/Python_Scripting#paraview.simple_Module

- The first thing any ParaView Python script must do is load the paraview.simple module.

  *from paraview.simple import **

  Or in a save way using the try/except construct

  try: paraview.simple

  *except: from paraview.simple import **
- This command is automatically invoked in the ParaView GUI tracing facility, but it must add when you are writing a script to be used in batch. The simple module can be loaded from Python interpreters running in several applications:
  - pvpython
  - Pvbatch
  - Paraview (Tools > python shell)
  - External Python interpreter
- The *paraview.simple* module contains a function for every source, reader,filter, and writer with off-course the same name as shown in the GUI menus but with spaces and special characters removed.
- Each function creates a pipeline object, which will show up in the pipeline browser (with the exception of writers), and returns an object that is a proxy that can be used to query and manipulate the properties of that pipeline object.

Batch scripting in python

Python 2.7.3 (default, Feb 4 2013, 20:52:48) [MSC v.1500 64 bit (AMD64)] on win32

>>> paraview version 3.98.1-RC2

from paraview.simple import *

>>> help(Sphere)

Help on function CreateObject in module paraview.simple:

CreateObject(*input, **params)

The Sphere source can be used to add a polygonal sphere to the 3D scene. The output of the Sphere source is polygonal data with point normals defined.

This function creates a new proxy. For pipeline objects that accept inputs, all non-keyword arguments are assumed to be inputs. All keyword arguments are assumed to be property,value pairs and are passed to the new proxy.

>>> help(Show)

Help on function Show in module paraview.simple:

Show(proxy=None, view=None, **params)

Turns the visibility of a given pipeline object on in the given view.

If pipeline object and/or view are not specified, active objects are used.

>>> help(Render)

Help on function Render in module paraview.simple:

Render(view=None)

Renders the given view (default value is active view)

>>> help(Hide)

Help on function Hide in module paraview.simple:

Hide(proxy=None, view=None)

Turns the visibility of a given pipeline object off in the given view.

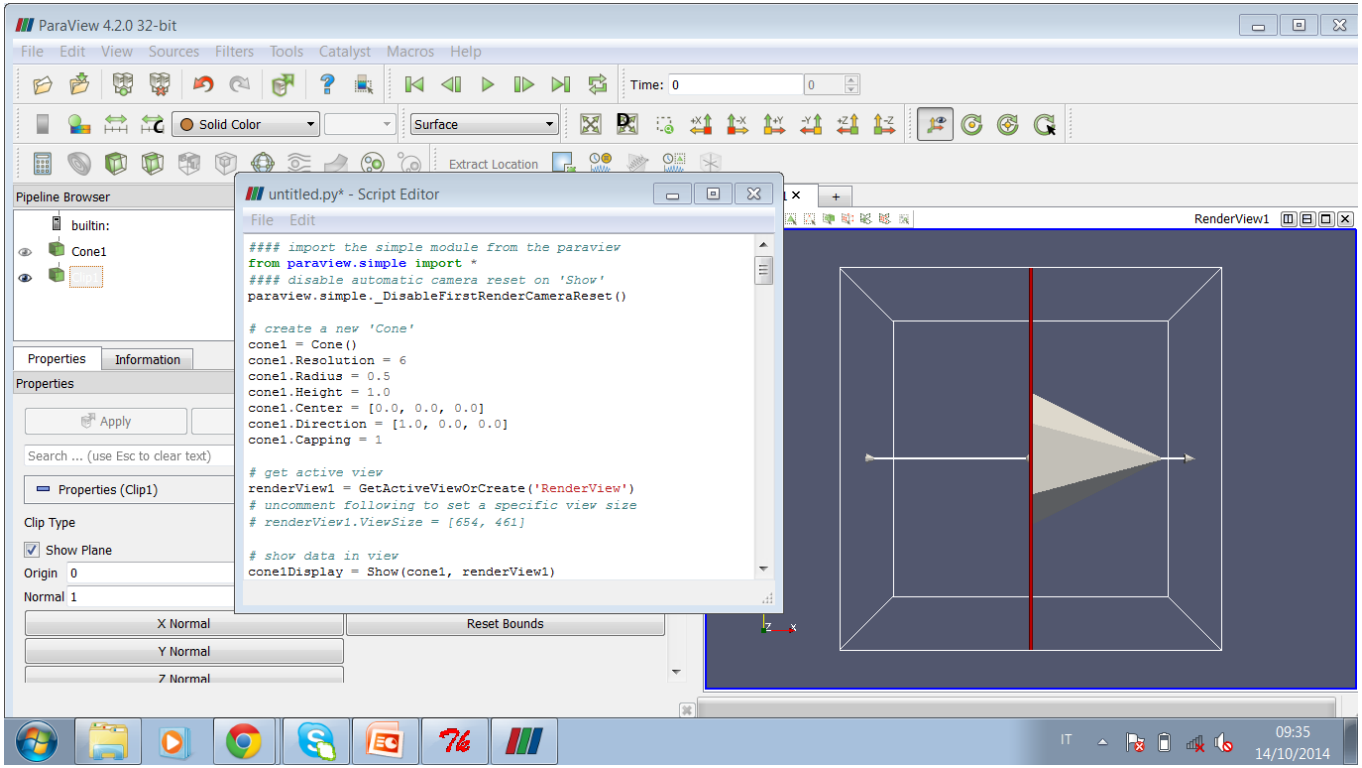
If pipeline object and/or view are not specified, active objects are used.

Batch scripting in python

```
>>> mysphere=Sphere()
>>> dir(mysphere)
['CellData', 'Center', 'EndPhi', 'EndTheta', 'FieldData', 'FileNameChanged', 'GetCellDataInformation', 'GetDataInformation',
'GetFieldDataInformation', 'GetPointDataInformation', 'GetProperty', 'GetPropertyValue', 'Initialize', 'InitializeFromProxy', 'ListProperties',
'Observed', 'ObserverTag', 'PhiResolution', 'PointData', 'Port', 'Radius', 'SMProxy', 'SetPropertyWithName', 'StartPhi', 'StartTheta',
'ThetaResolution', 'UpdatePipeline', 'UpdatePipelineInformation', '_Proxy__ConvertArgumentsAndCall', '_Proxy__GetActiveCamera',
'_Proxy__LastAttrName', '_Proxy__Properties', '__class__', '__del__', '__delattr__', '__dict__', '__doc__', '__eq__', '__format__', '__getattr__',
'__getattribute__', '__getitem__', '__hash__', '__init__', '__iter__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'add_attribute']
>>> print mysphere.Radius
0.5
>>> print mysphere.PointData
<paraview.servermanager.FieldDataInformation object at 0x0000000017BBAC50>
>>> Render()
<paraview.servermanager.RenderView object at 0x0000000018044FD0>
>>> Show()
<paraview.servermanager.GeometryRepresentation object at 0x0000000017A50748>
>>>
```

Trace record

1. Start tracing
2. Build your pipeline as usual
3. Stop tracing and save the trace file with a name



Use the trace file

You can use the traced file in two ways:

- As a macro

- As a python script using the GUI
- As a python script in batch

Macro

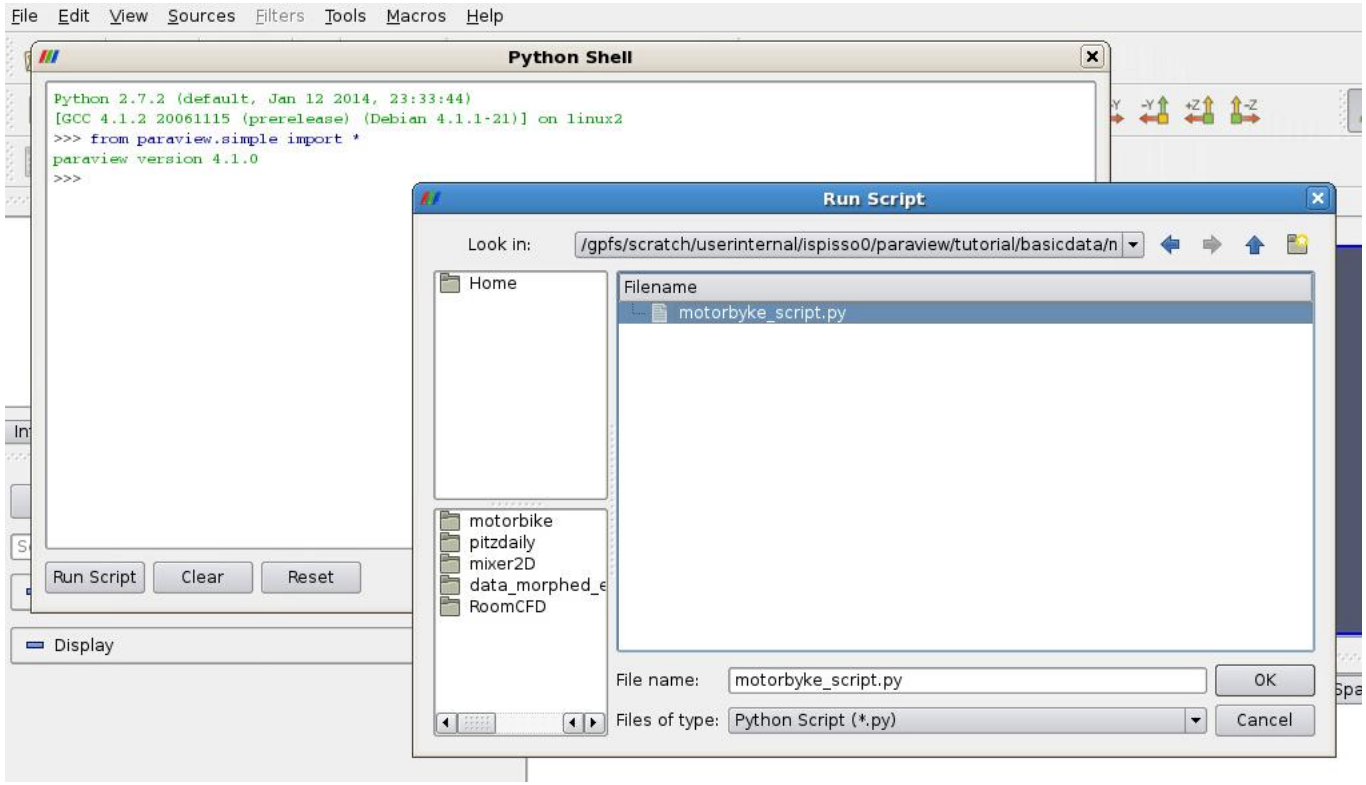
• Macros allow the user to define a Python script as built-in actions that become accessible from the Macros menu or directly inside the Toolbar.

• Once a local file is defined as a macro (by clicking-on Create New Macro) the given file is copied inside the user specific ParaView directory. No reference is kept to the original file. Therefore, if you keep editing the original file, the changes won't affect the macro itself.

• The macros list is built dynamically at the ParaView start-up by listing the content of the ParaView/Macros directory, as well as the user specific ParaView/Macros directory. Note: if you want to rename a macro, rename the file in one of the given directory.

Save ParaView state as a Python script

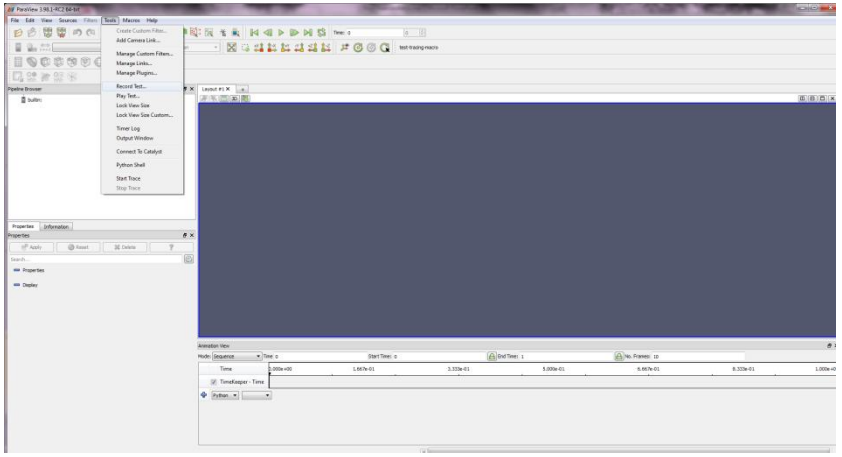
To save the state as a Python script, go to the File menu and choose Save State without forgetting to switch the file type to be Python *.py.



Recording a Paraview pipeline

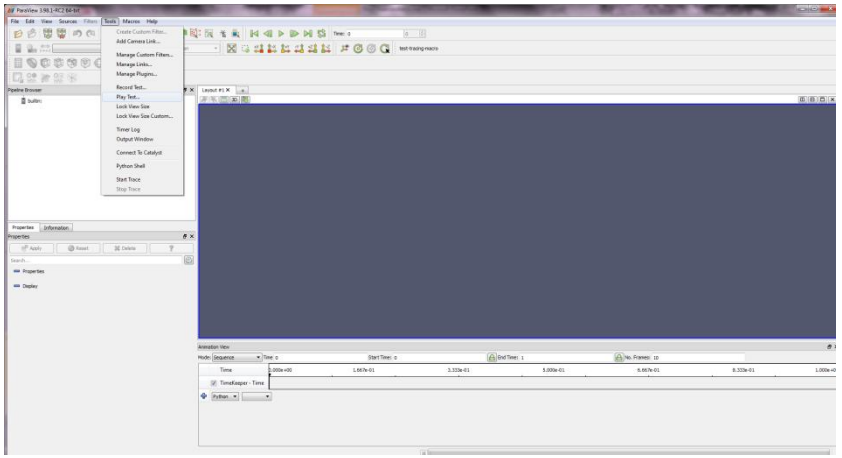
A similar (but different) solution to launch a pipeline in an automated way is to:

- record a test



and then to:

- Play a test



Workout

- Build a script to process several VTK dataset according to the pipeline obtained during the previous tutorial
- Launch the script over the dataset to archive different animations
- Record a test and play it for the above mentioned operations