Scientific Visualization with ParaView

Geilo Winter School 2016

Andrea Brambilla (GEXCON AS, Bergen)

Outline

- Part 1 (Monday)
  - Fundamentals
  - Data Filtering
- Part 2 (Tuesday)
  - Time Dependent Data
  - Selection & Linked Views
- Part 3 (Thursday)
  - Python scripting

Batch Scripting

Python Batch Scripting

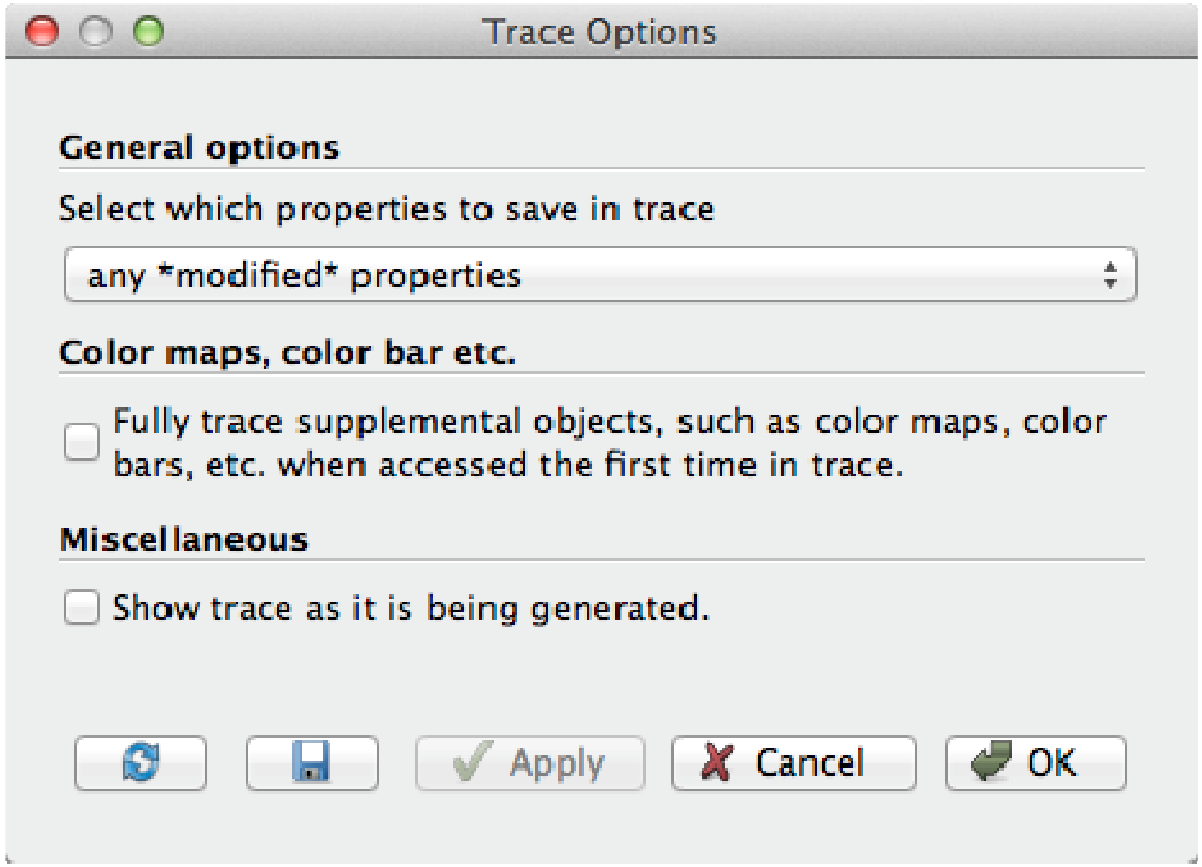
- Interpreter: Tools → Python Shell
  - External programs pvpython and pvbatch
  - Can add bindings to external interpreters
- Run Python scripts or bind to macros
- Can trace actions or capture state

www.paraview.org/Wiki/ParaView/Python_Scripting

- (Batch scripting is a different set of bindings than programmable sources/filters.)

Python Tracing Options

1. Select Tools → Start Trace



Tracing ParaView State

1. Select Tools → Start Trace
2. Accept defaults (click OK)
3. Build a simple pipeline
   - create a sphere source and clip it
4. Select Tools → Stop Trace
5. An editing window will open

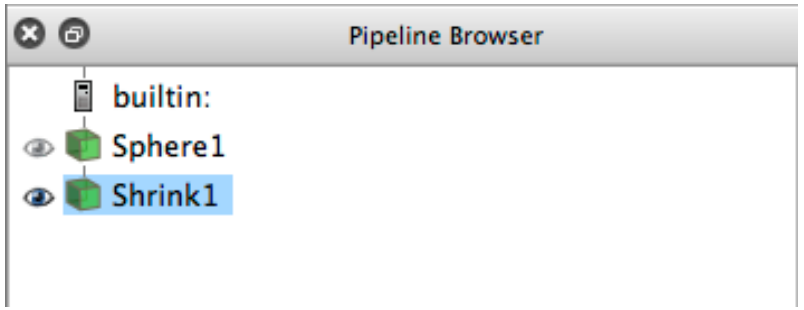
Adding a Macro

1. In Python edit window menu, select File → Save As Macro...
2. Chose a descriptive name. Save in directory provided
3. Close Python edit window
4. Select Edit → Reset Session
5. Activate your macro
   (on toolbar or Macros menu)

Creating a Pipeline

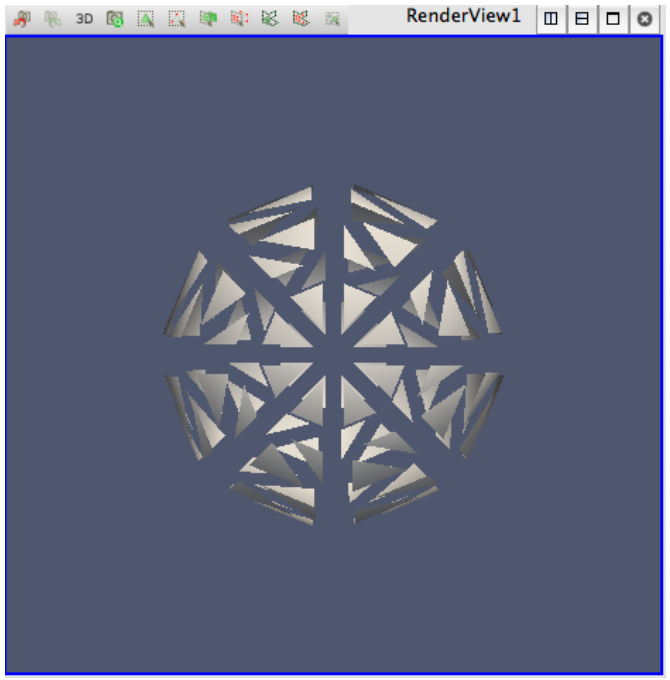
- In Python Shell create a source

```
sphere = Sphere()
Show()
Render()
```



- Then feed that into a filter

```
Hide()
shrink = Shrink()
Show()
Render()
```



Changing Properties

- Get listing of sphere's properties

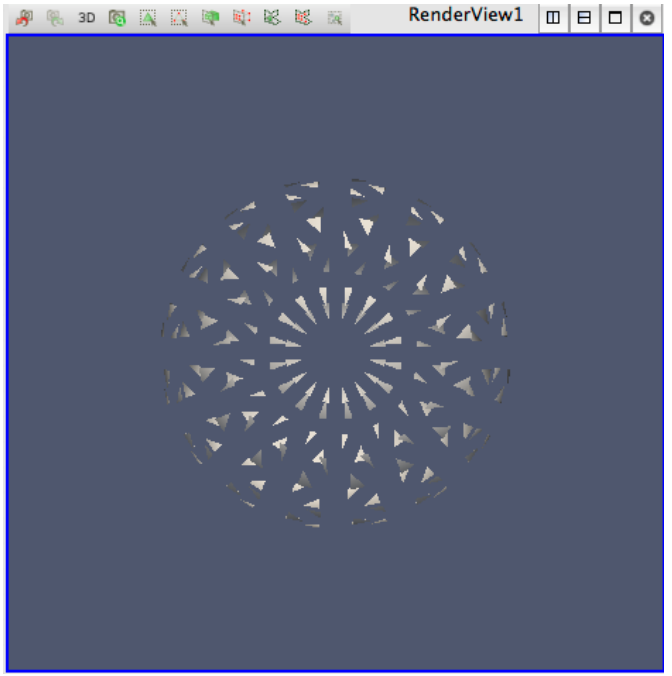
```
dir(sphere)
help(sphere) / help(Sphere)
```

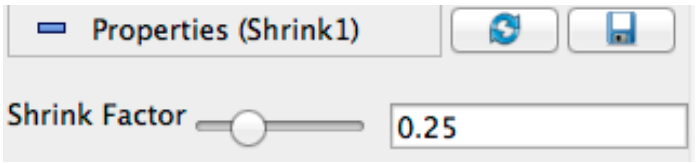
- Examine and then change one

```
print(sphere.ThetaResolution)
Sphere.ThetaResolution = 16
Render()
```

- Change a different filter

```
Shrink.ShrinkFactor = 0.25
Render()
```





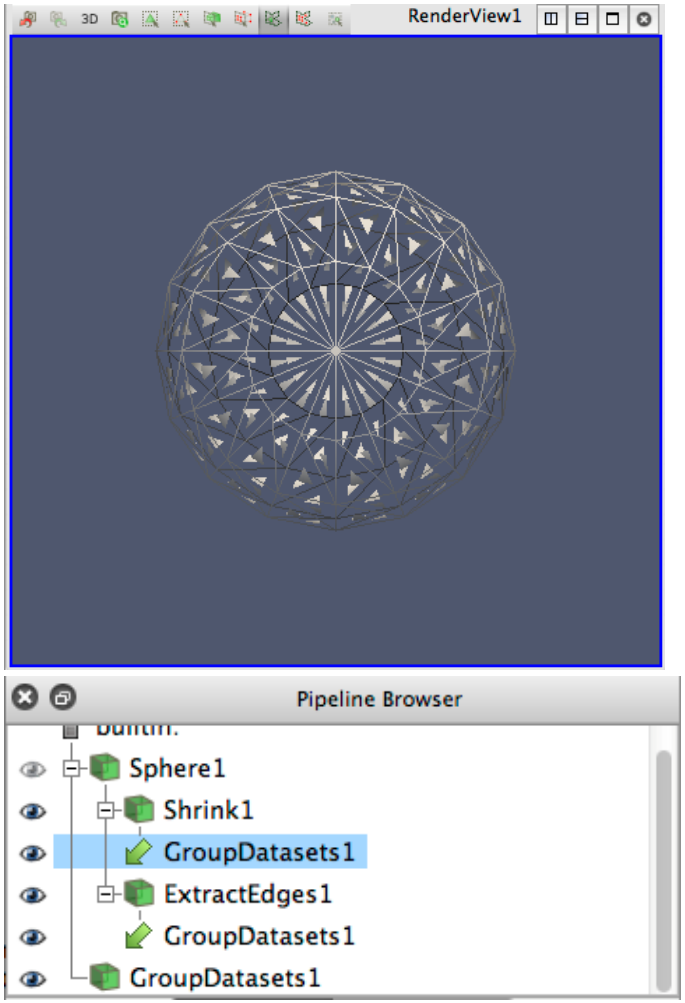
Branching and Merging Pipeline

- Feed sphere's output into a second filter

```
wireframe = ExtractEdges(
    Input=sphere)
Show()
Render()
```

- Feed two filters into one

```
group = GroupDatasets(
    Input=[shrink,wireframe])
Show()
```

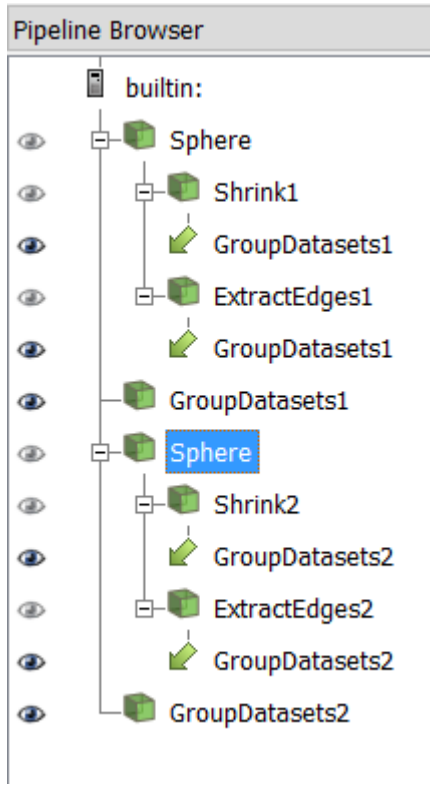


Active Object

- You may want to operate on a specific objects

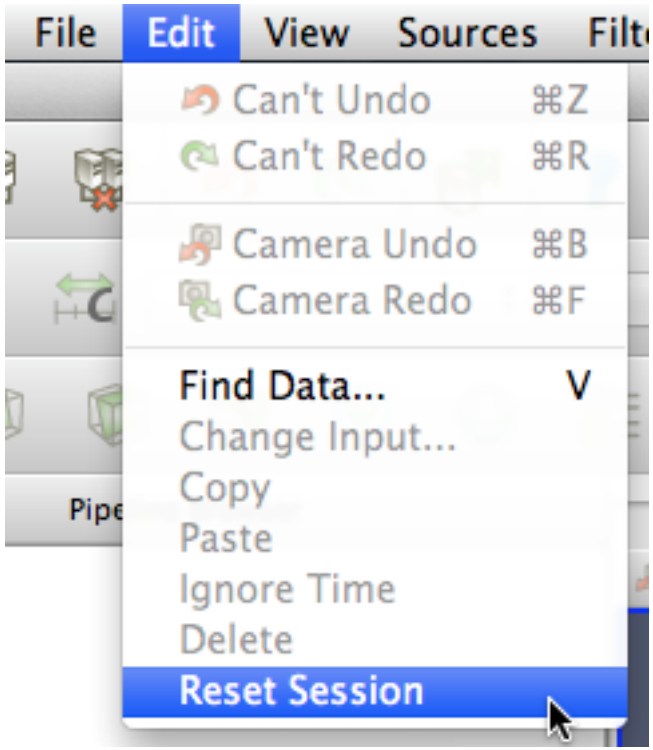
```
my_object = GetActiveSource()
SetActiveSource(my_object)

objs_dict = GetSources()
    # dict in the form
    # (‘MyObject’, ‘1234’) : my_object
my_object = FindSource(‘MyObject’)
```



Reset ParaView

Edit → Reset Session



Reading Files

- Open a dataset

```
reader = OpenDataFile(
    ‘<path>/disk_out_ref.ex2’)
Show()
Render()
ResetCamera()
```

Querying Attributes

- Examine all point data ranges

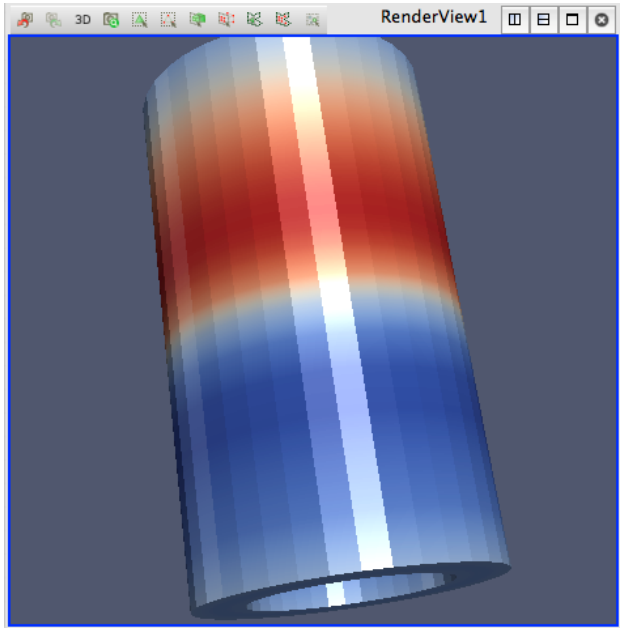
```
pd = reader.PointData   #or reader.CellData
for ai in pd.values():
    print ai.GetName(), ai.GetNumberOfComponents(),
    for i in range(ai.GetNumberOfComponents()):
        print ai.GetRange(i),
    print
```

```
AsH3 1 (0.08047682046890259, 0.18483947217464447)
CH4 1 (0.0, 0.00117023917846381 66)
GaMe3 1 (0.00022284436272457242, 0.007213925942778587)
GlobalNodeId 1 (1.0, 8499.0)
H2 1 (0.8076131343841553, 0.9176878929138184)
PedigreeNodeId 1 (1.0, 8499.0)
Pres 1 (0.006785521749407053, 0.028818512335419655)
Temp 1 (293.1499938964844, 913.1500244140625)
V 3 (-19.949113845825195, 19.949113845825195) (-19.949113845825195, 19.949113845825195)
(-21.1259765625, 6.7119011878967285)
```

Coloring Data

- Get a hold of and manipulate display properties

```
readerRep = GetRepresentation()
readerRep.DiffuseColor = [0, 0, 1]
readerRep.SpecularColor = [1, 1, 1]
readerRep.SpecularPower = 128
readerRep.Specular = 1
Render()
```



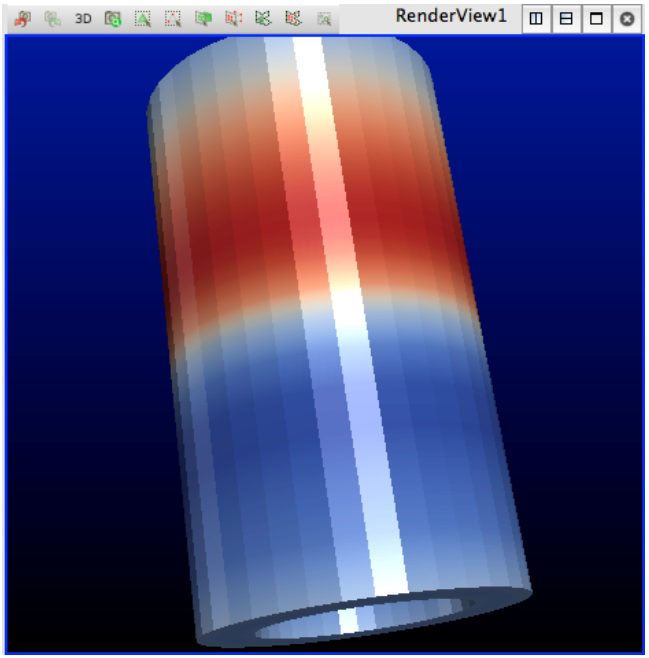
- Now assign a color transfer function

```
readerRep.ColorArrayName = ‘Pres’
readerRep.LookupTable = \
    AssignLookupTable(reader.PointData['Pres'],
    'Cool to Warm')
Render()
```

Controlling the View

Get a hold of, then manipulate view properties

```
view = GetActiveView()
view.Background = [0, 0, 0]
view.Background2 = [0, 0, 0.6]
view.UseGradientBackground =
    True
Render()
```



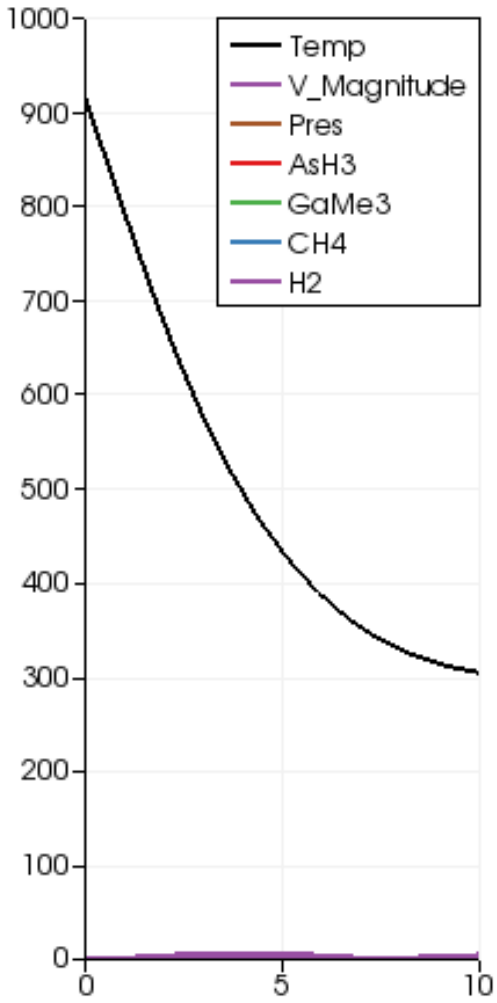
Save Results

Save Data

```
plot = PlotOverLine()
plot.Source.Point1 = [0,0,0]
plot.Source.Point2 = [0,0,10]
writer = CreateWriter(‘⟨path⟩/plot.csv’)
writer.UpdatePipeline()
```

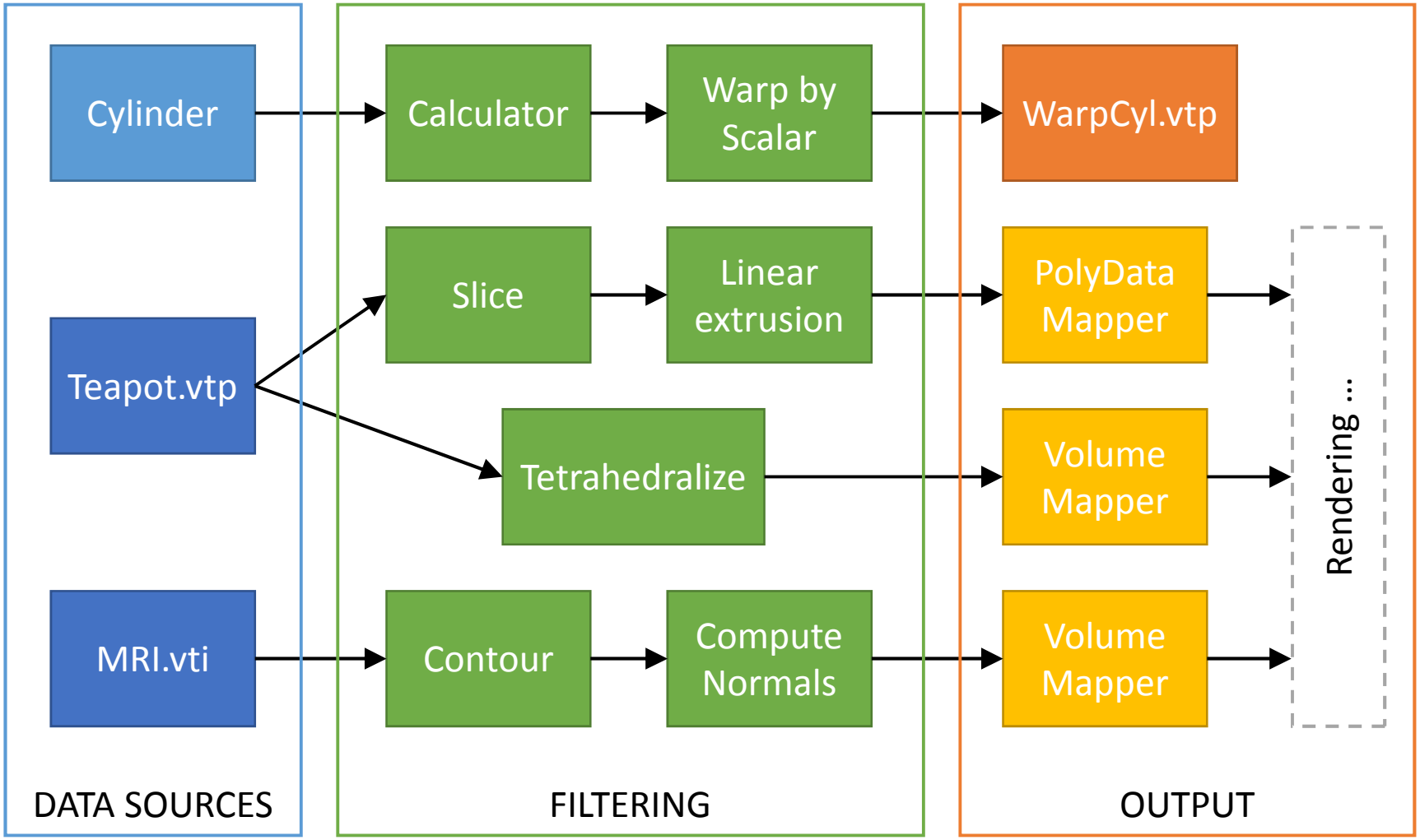
Then save screen capture of plot

```
plotView = CreateView(’XYChartView’)
Show(plot)
Render()
SaveScreenshot(‘<path>/plot.png’)
```



Programmable Source & Filter

ParaView Visualization Pipeline

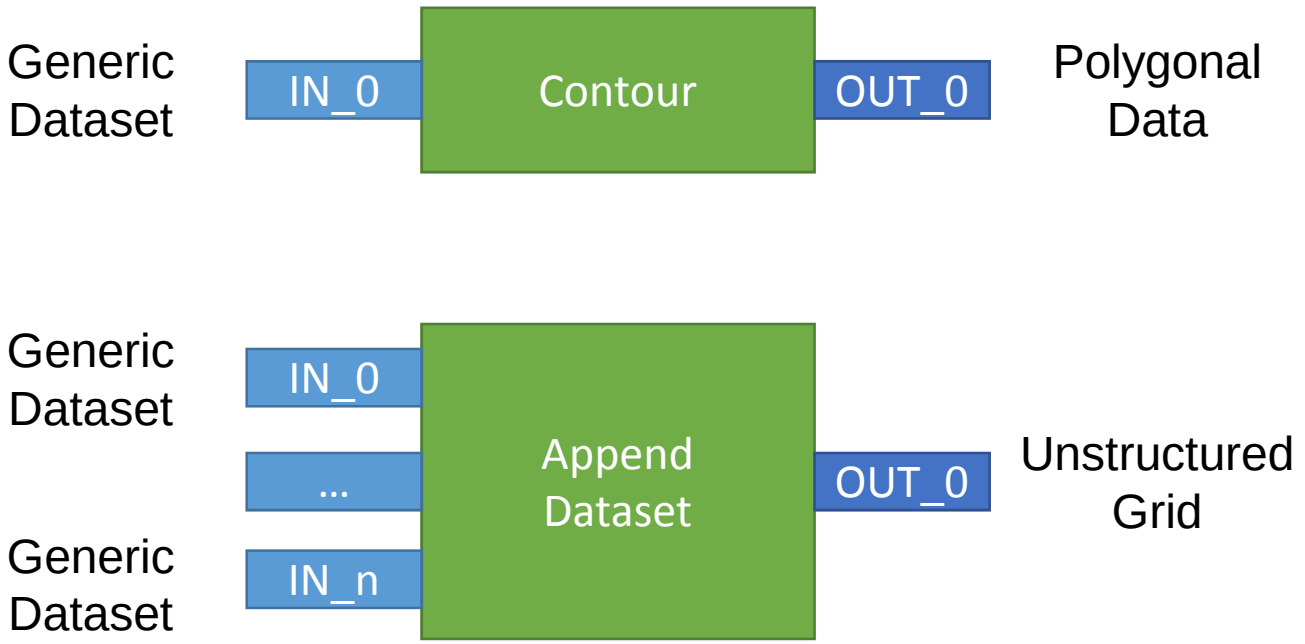


Programmable Source / Filter

- ParaView cannot do what I need
- ParaView cannot read my data
- I don’t have data, I have equations

- Define custom operation using Python
- Fully integrated with the ParaView pipeline

Input and Output ports



Examples / Documentation

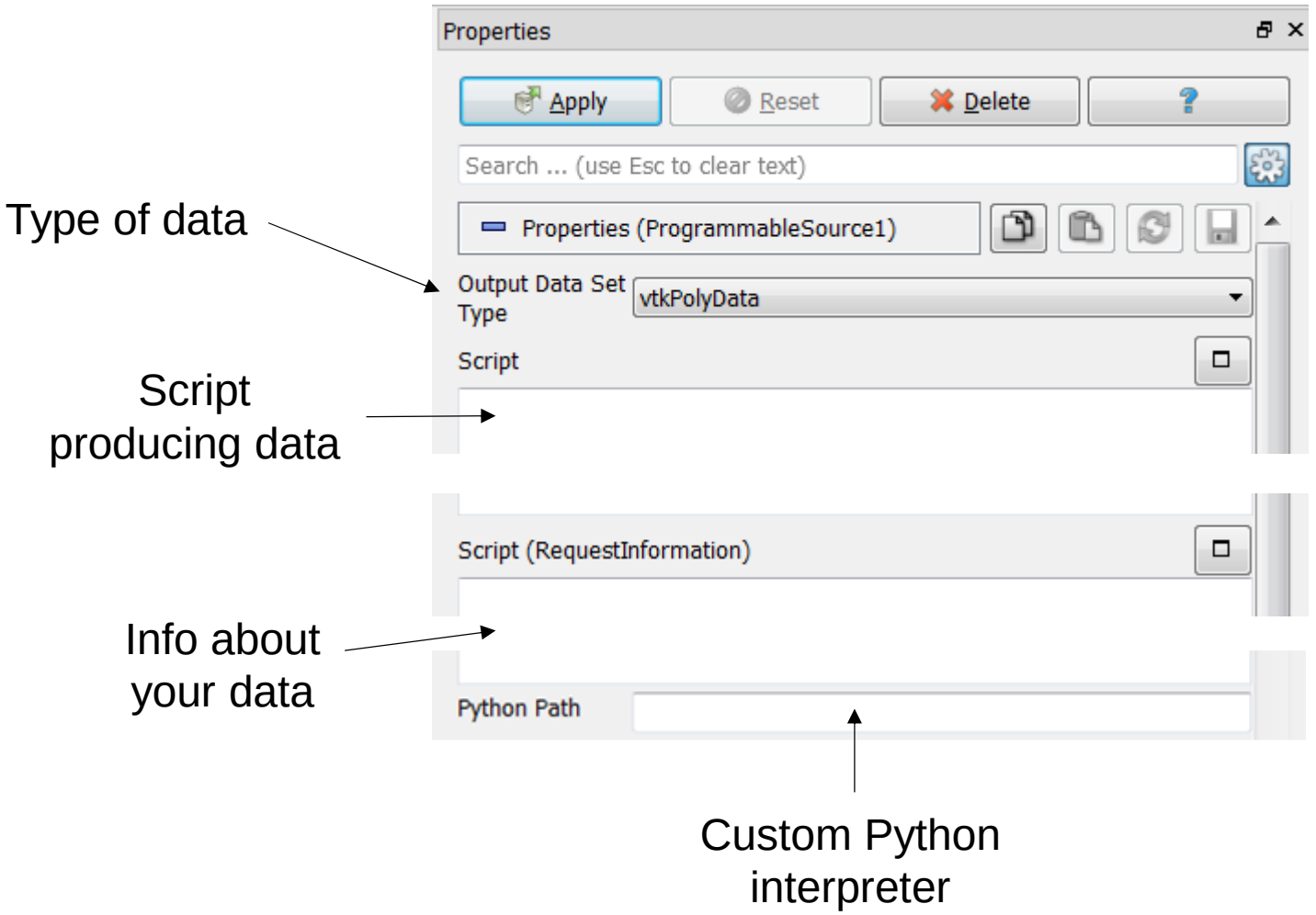
www.paraview.org/Wiki/Python_Programmable_Filter

www.paraview.org/Wiki/Here_are_some_more_examples_of_simple_ParaView_3_python_filters

VTK Documentation:

- http://www.vtk.org/doc/nightly/html/annotated.html
- Or on the USB sticks

Programmable Source



2D Cosine Wave (points)

```
import math

step = 0.1 * pi
side_pn = 20
side_size = side_pn * 2 + 1
pts_num = side_size**2

pdo = self.GetPolyDataOutput()

new_pts = vtk.vtkPoints()
new_pts.SetNumberOfPoints(pts_num)
pt_id = 0
for y_id in range(-side_pn, side_pn + 1):
  for x_id in range(-side_pn, side_pn + 1):
    x = x_id * step
    y = y_id * step
    z = cos(sqrt(x**2 + y**2))
    new_pts.SetPoint(pt_id, x,y,z)
    pt_id += 1
pdo.SetPoints(new_pts)
```

2D Cosine Wave (cells)

…

```
newCells = vtk.vtkCellArray()
pt_id = 0
for y_id in range(-side_pn, side_pn + 0):
  for x_id in range(-side_pn, side_pn + 0):
    newCells.InsertNextCell(4,
      [pt_id,
       pt_id + 1,
       pt_id + side_size + 1,
       pt_id + side_size])
    pt_id += 1
  pt_id += 1

pdo.SetPolys(newCells)
```

Custom Reader (Cartesian grid)

```
import struct
with open("path\\lobster.dat", "rb") as f:
  my_data = f.read()

dim = struct.unpack("HHH", my_data[:6])
pts_num = dim[0] * dim[1] * dim[2]

data = vtk.vtkDoubleArray()
data.SetNumberOfComponents(1)
data.SetNumberOfTuples(pts_num)
data.SetName("Density")

for pt_id in range(pts_num):
  val = struct.unpack("H",
    my_data[6 + pt_id*2 : 6 + pt_id*2 + 2])
  data.SetValue(pt_id, float(val[0]))
```

…

Custom Reader (Cartesian grid)

…

```
output = self.GetOutput()

output.SetExtent(0,
  dim[0]-1, 0, dim[1]-1, 0, dim[2]-1)

output.GetPointData().SetScalars(data)

# or output.GetPointData().SetVectors(…)
# or output.GetPointData().SetTensors(…)
# or output.GetPointData().AddArray(…)
```

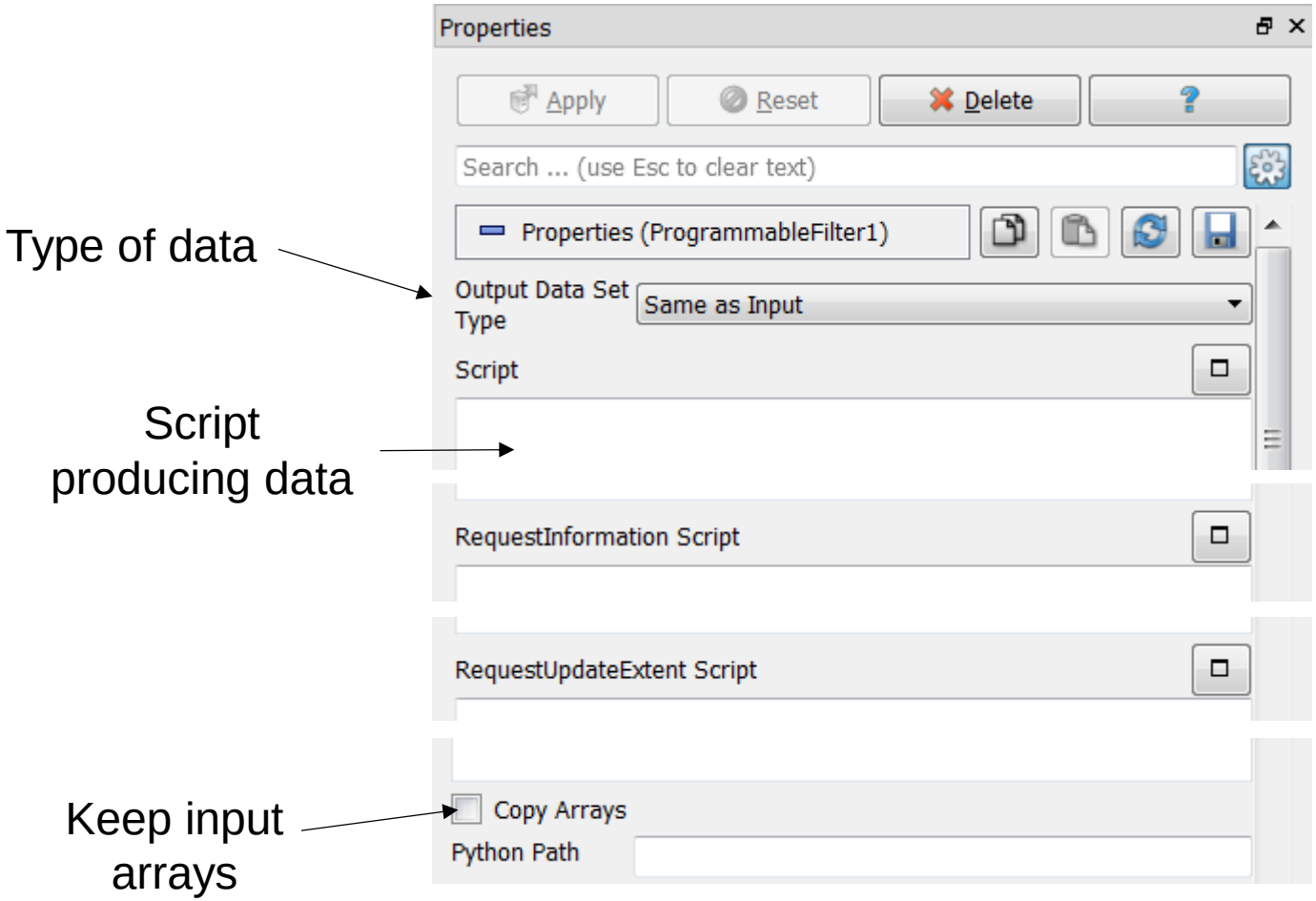
Custom Reader (Request Info)

```
import struct
with open("path\\lobster.dat", "rb") as f:
  my_data = f.read()

dim = struct.unpack("HHH", my_data[:6])
pts_num = dim[0] * dim[1] * dim[2]

# define spatial extent of image data
executive = self.GetExecutive()
outInfo = executive.GetOutputInformation(0)
outInfo.Set(executive.WHOLE_EXTENT(),
    0, dim[0]-1, 0, dim[1]-1, 0, dim[2]-1)
outInfo.Set(vtk.vtkDataObject.SPACING(),
    1, 1, 1)
outInfo.Set(vtk.vtkDataObject.ORIGIN(),
    0, 0, 0)
```

Programmable Filter



Distance Field (Script)

```
import math

def compute_xyz(pt_ids, origin, spacing):
    return [origin[i] + pt_ids[i] * spacing[i]
                for i in range(3)]

def min_distance(pt, polydata):
  min_dist = 9999999.
  for pt_id in range(
                    polydata.GetNumberOfPoints()):
    pd_pt = polydata.GetPoint(pt_id)
    dist = sum([(a-b)**2
                for (a,b) in zip(pt, pd_pt)])
    if dist < min_dist:
      min_dist = dist
  return sqrt(min_dist)

…
```

Distance Field (Script)

```
…

cos_input = self.GetInput()

extent = [-3, 3, -3, 3, -3, 3]
origin = [0, 0, 0]
spacing = [2, 2, 2]

pts_num = (extent[1]-extent[0]+1)*(extent[3]-
            extent[2]+1)*(extent[5]-extent[4]+1)

data = vtk.vtkDoubleArray()
data.SetNumberOfComponents(1)
data.SetNumberOfTuples(pts_num)
data.SetName("Distance")


…
pt_id = 0
for zId in range(extent[4], extent[5]+1):
  for yId in range(extent[2], extent[3]+1):
    for xId in range(extent[0], extent[1]+1):
```

Distance Field (Script)

```
…

pt_id = 0
for zId in range(extent[4], extent[5]+1):
  for yId in range(extent[2], extent[3]+1):
    for xId in range(extent[0], extent[1]+1):
      pt = compute_xyz(
             [xId, yId, zId], origin, spacing)
      distance = min_distance(pt, cos_input)
      data.SetValue(pt_id, distance)
      pt_id += 1


output = self.GetOutput()
output.SetOrigin(origin)
output.SetSpacing(spacing)
output.SetExtent(extent)
output.GetPointData().SetScalars(data)
```

Distance Field (Request Info)

```
# define spatial extent of image data
executive = self.GetExecutive()
outInfo = executive.GetOutputInformation(0)
outInfo.Set(executive.WHOLE_EXTENT(),
    -3, 3, -3, 3, -3, 3)
outInfo.Set(vtk.vtkDataObject.SPACING(),
    2, 2, 2)
outInfo.Set(vtk.vtkDataObject.ORIGIN(),
    0, 0, 0)
```

Exercises

- Exercise 5
  - Take any of the previous exercises and redo it by using python only
  - Try to identify re-usable procedures and create macros for them

Exercises

- Exercise 6
  - Custom reader for polydata.csv
  - Generate ABC flow
    - `u = √3 sin(z) +  1 cos(y)`
    - `v = √2 sin(x) + √3 cos(z)`
    - `w =  1 sin(y) + √2 cos(x)`
    - en.wikipedia.org/wiki/Arnold%E2%80%93Beltrami%E2%80%93Childress_flow