ParaView Introduction and Data Imports

Dr. Jean M. Favre, CSCS
October 12, 2020

Introduction - Objectives

- Introduce ParaView and VTK
- Describe the VTK pipeline and VTK Objects
- Exercise the Python Programmable Source
- Import data from numpy arrays

User Guide

You simply cannot check it out at least once: ☺ ☺ ☺

https://docs.paraview.org/en/latest/UsersGuide/index.html

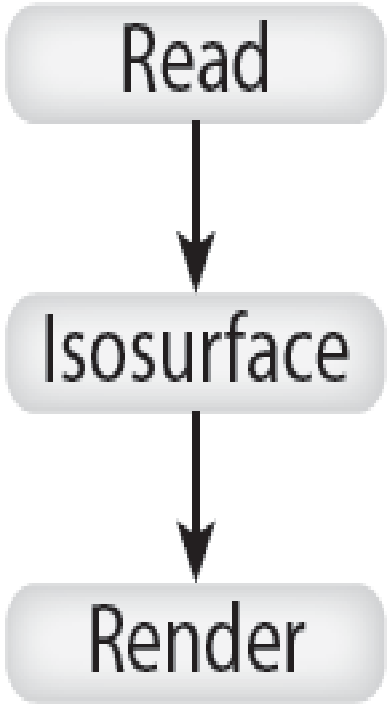
Discussion (Annoucements, PV support, in-situ support, web support, …)

https://discourse.paraview.org/

Visualization Pipeline: Introduction

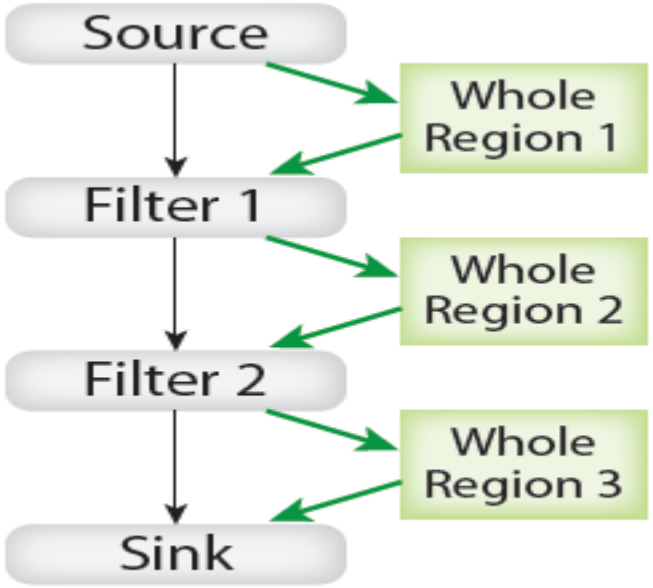
From a survey article by Ken Moreland, IEEE Transactions on Visualizations and Computer Graphics, vol 19. no 3, March 2013

«A visualization pipeline embodies a *dataflow network* in which computation is described as a collection of executable *modules* that are connected in a directed graph representing how data moves between modules. There are threee types of modules: *sources*, *filters* and *sinks*.»

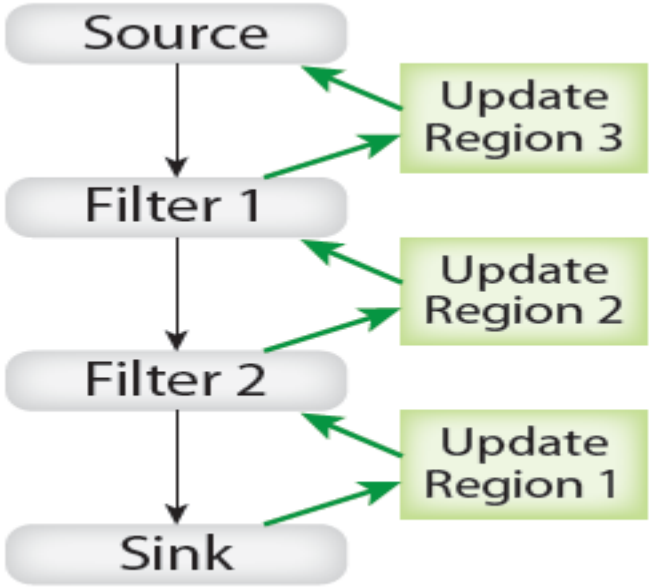


Visualization Pipeline: Metadata

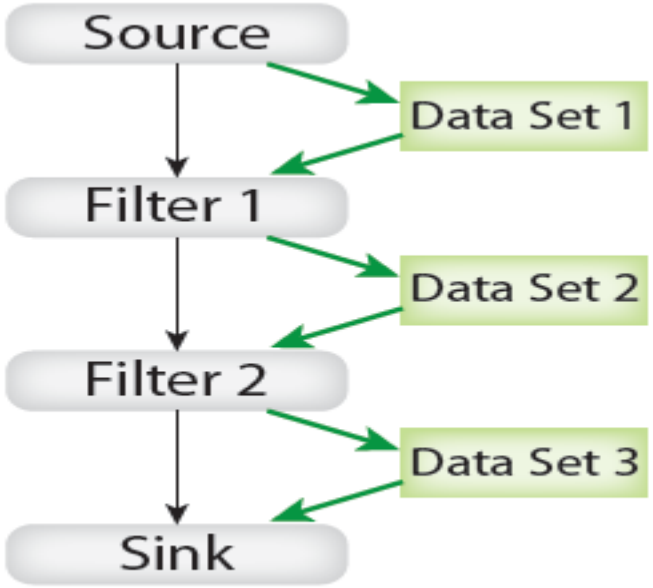
- 1st pass: Sources describe the region they can generate.
- 2nd pass: The application decides which region the sink should process.
- 3rd pass: The actual data flow through the pipeline



(a) Update Information



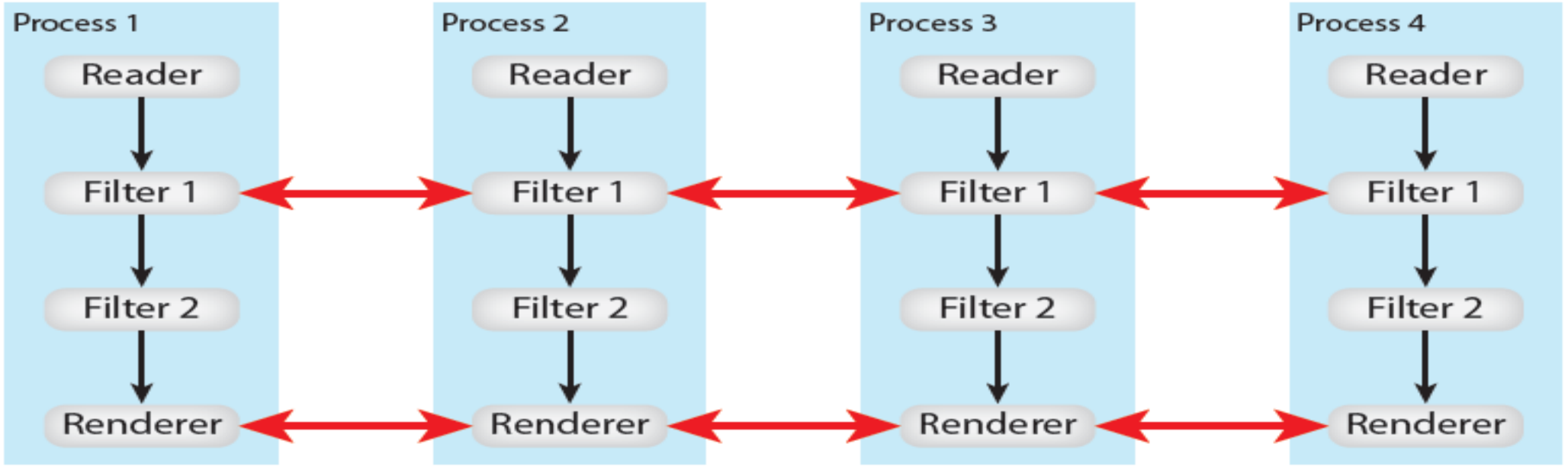
(b) Update Region



(c) Update Data

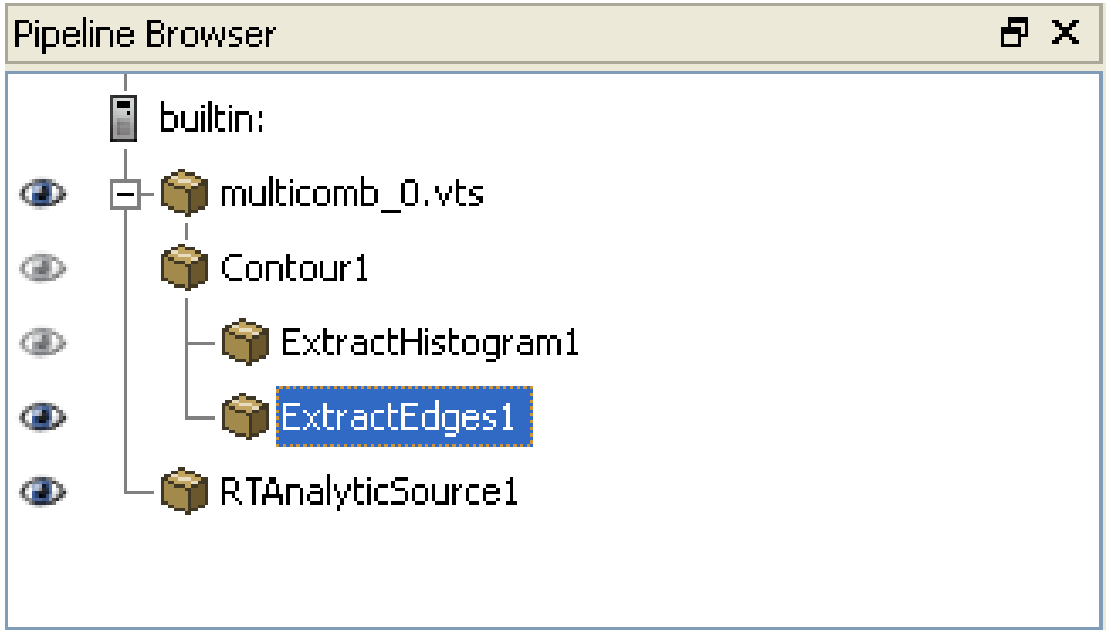
Visualization Pipeline: Data Parallelism

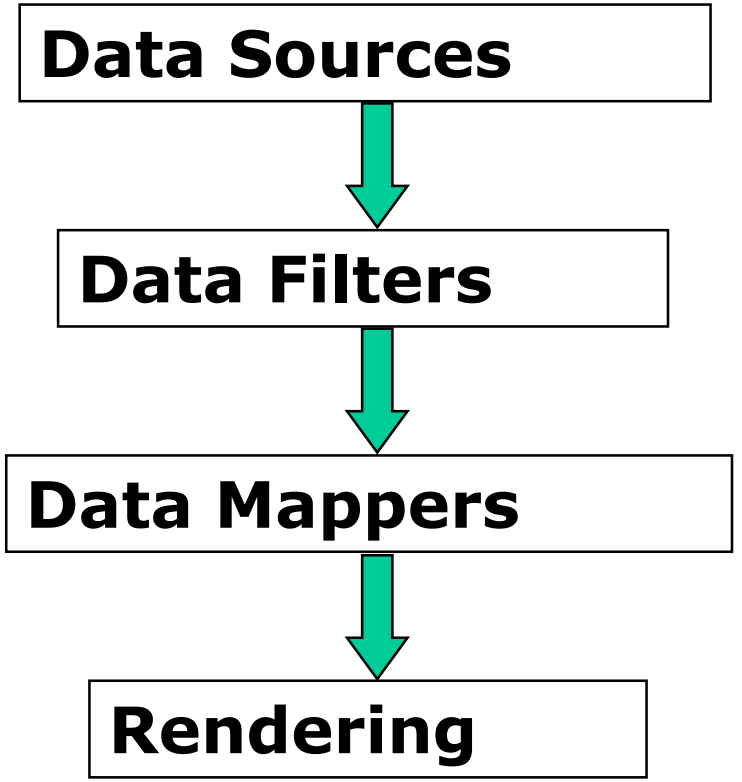
- Data parallelism partitions the input data into a set number of pieces, and replicates the pipeline for each piece.
- Some filters will have to exchange information (e.g. GhostCellGenerator)



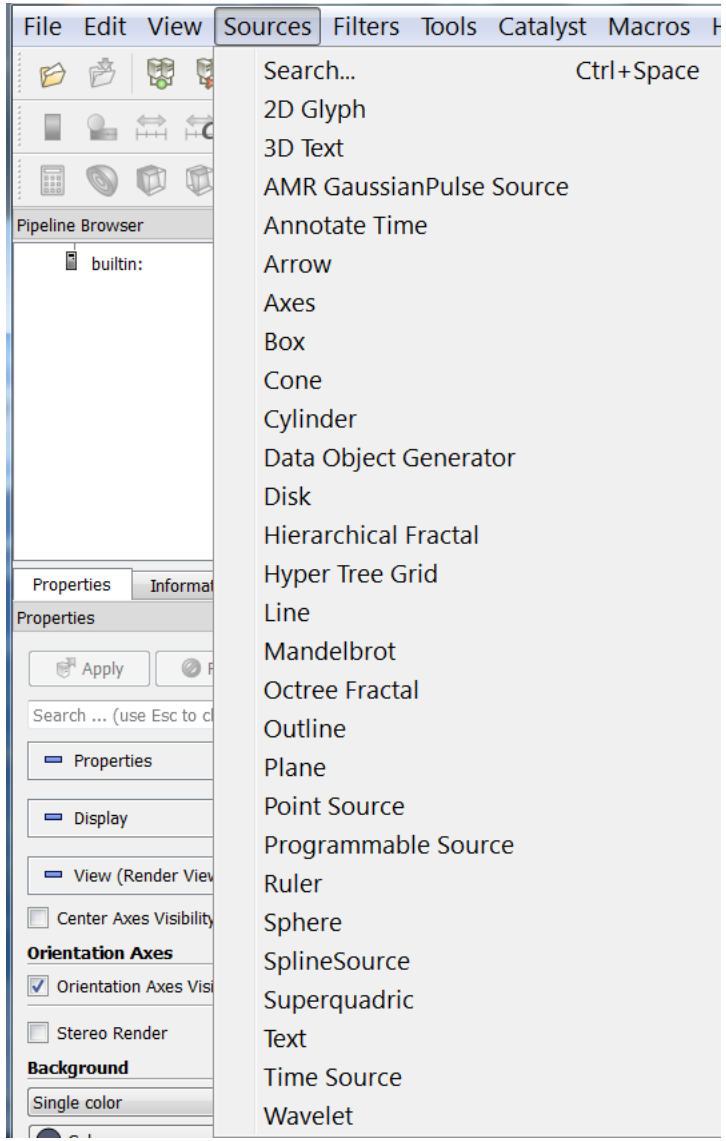
The VTK visualization pipeline

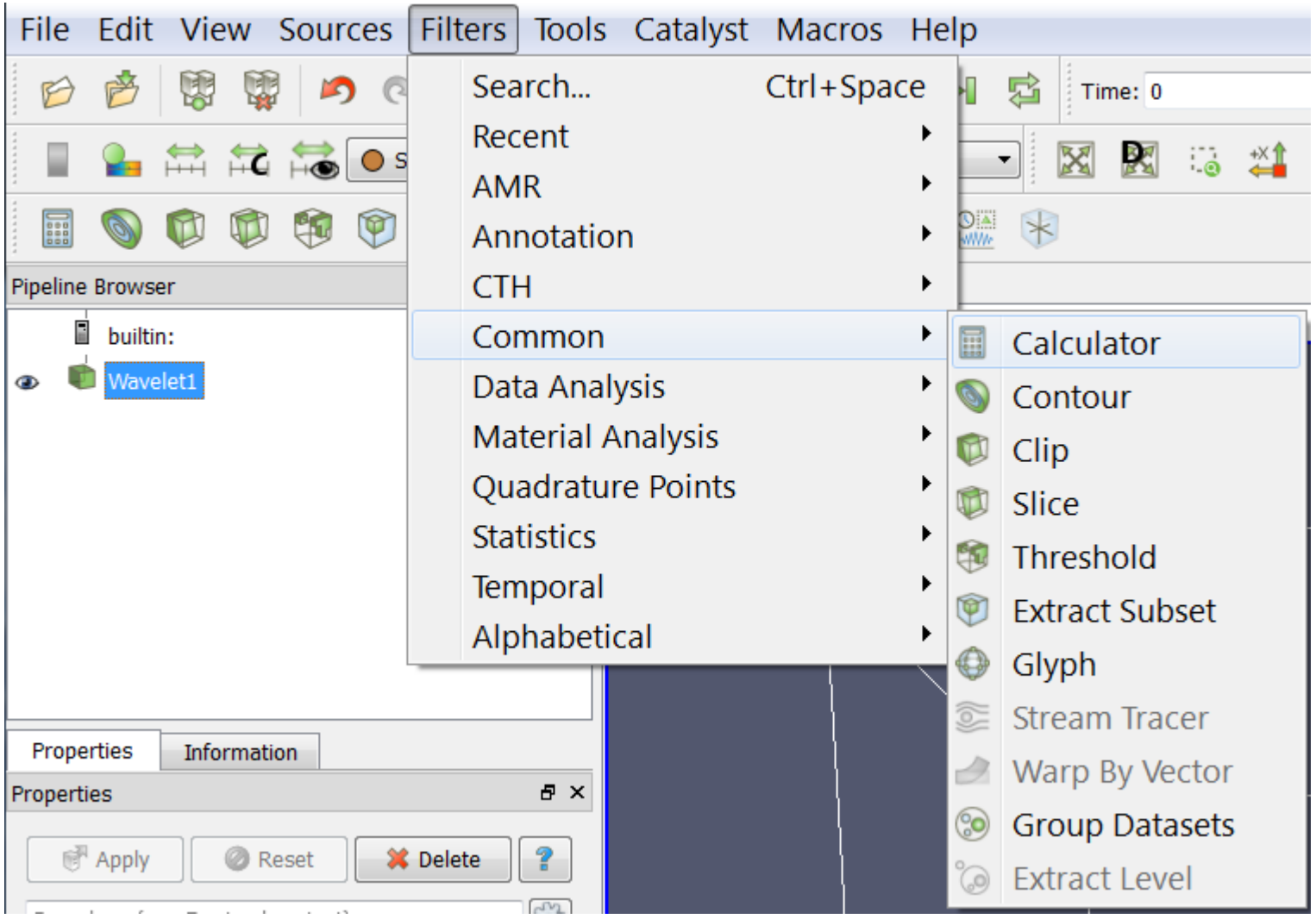
VTK's main execution paradigm is the *data-flow*, i.e. the concept of a downstream flow of data





Examples of Filters/Sources

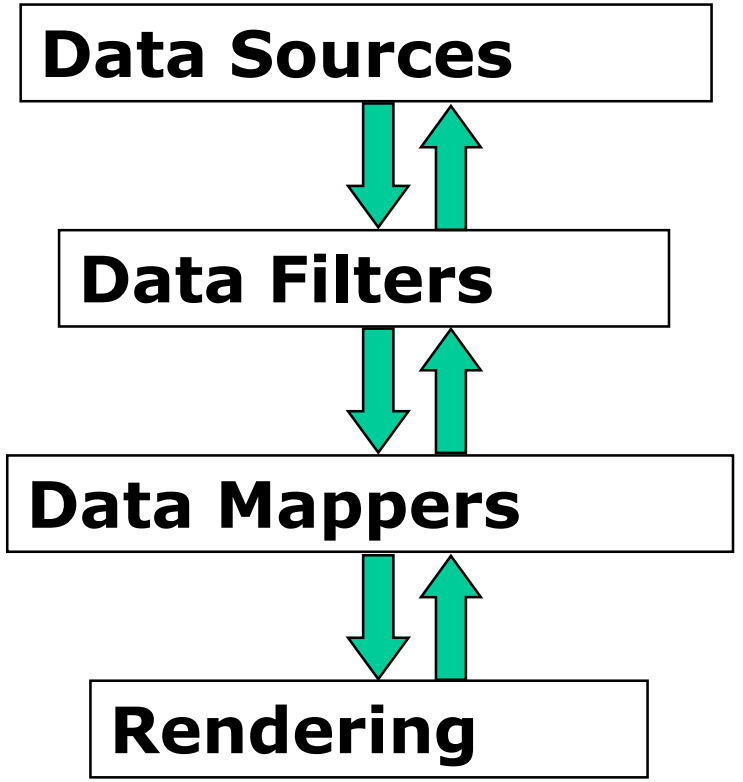




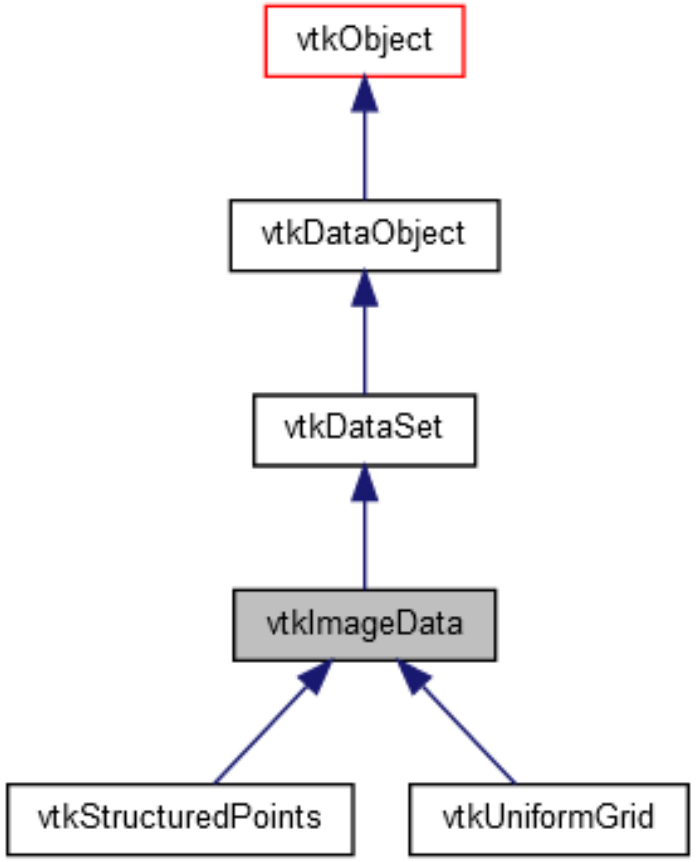
The VTK visualization pipeline

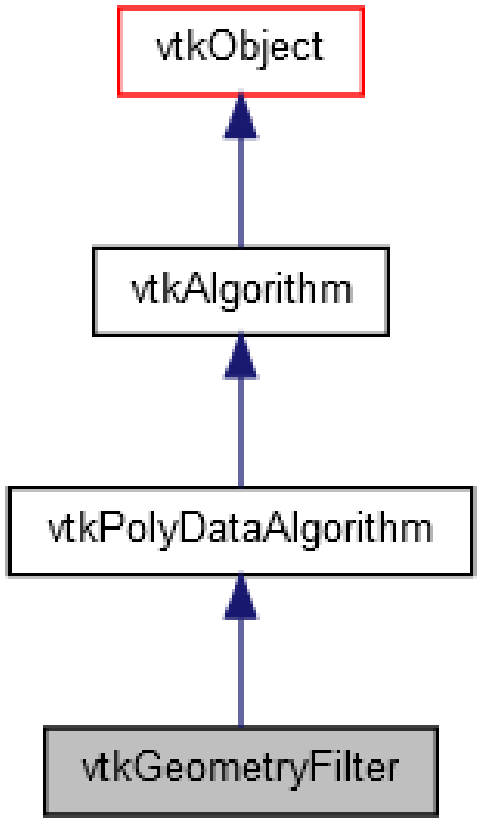
- VTK extends the *data-flow* paradigm
- VTK acts as an *event-flow* environment, where data flow downstream and events (or information) flow upstream
- ParaView's Rendering **triggers** the execution:

```
view = GetRenderView()
view.ViewTime = 5
Render()
```



vtkDataObject vs. vtkAlgorithm

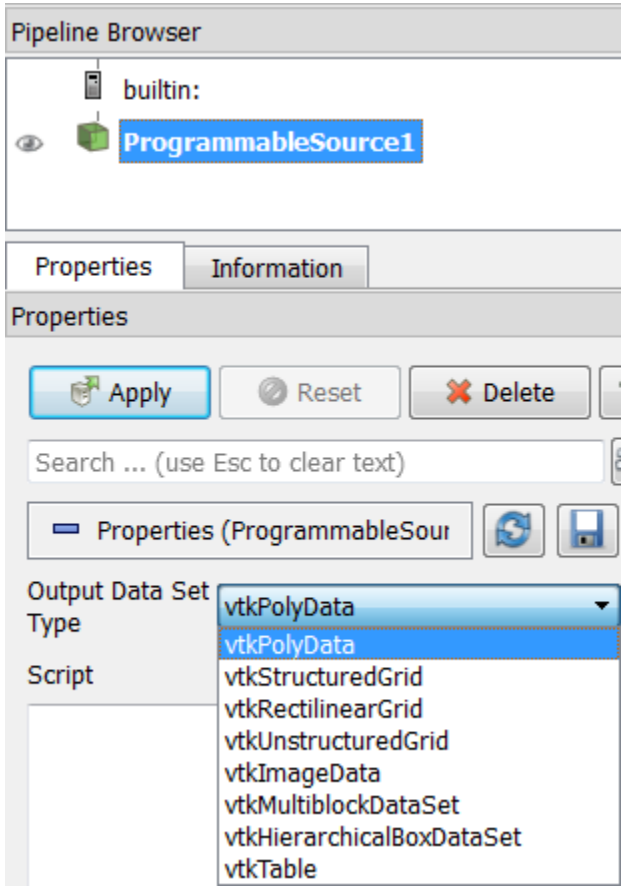




ParaView Python Programmable Source

ParaView: Python Programmable Source

1. Define an output dataset type,
2. Define the meta-data,
3. Execute the Script.



The python script

1. N.B. in client-server mode, the script is going to be executed on the server side
2. The python code is numpy-centric and will also use the VTK python API to create and access data arrays

We'll distinguish between two code sections:

Python code for 'RequestInformation Script'.

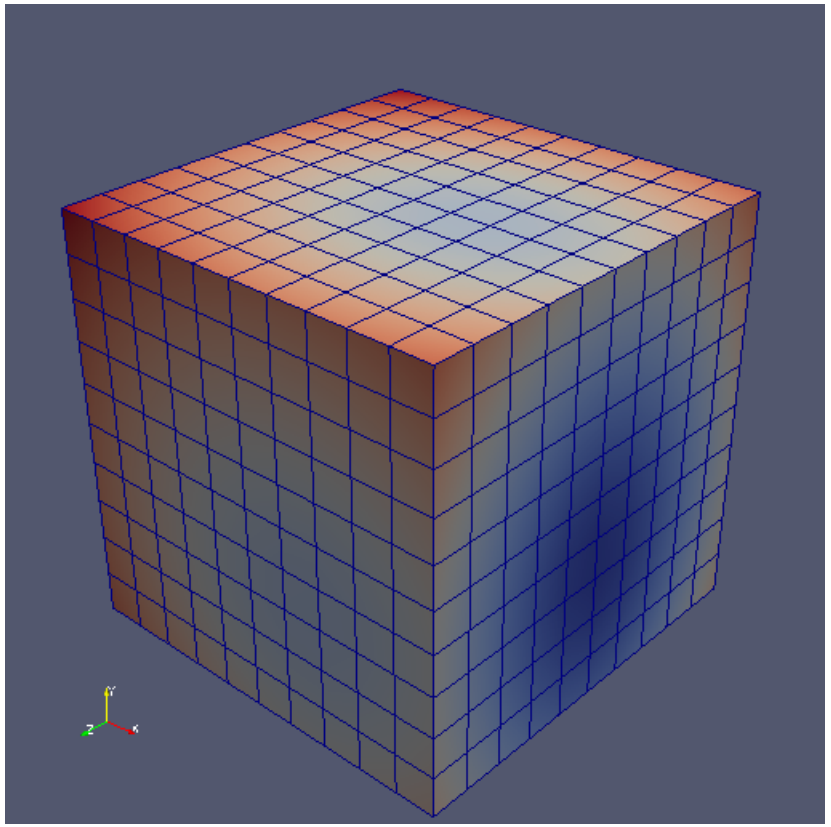
Python code for 'RequestData Script'.

The Pipeline meta-information (Example) (syntax has been simplified)

```
def RequestData():
 # VTK's pipeline is designed such that algorithms can ask a data producer for a subset of its whole extent.
# using the UPDATE_EXTENT key

exts = info.Get(UPDATE_EXTENT())

whole = info.Get(WHOLE_EXTENT())
```

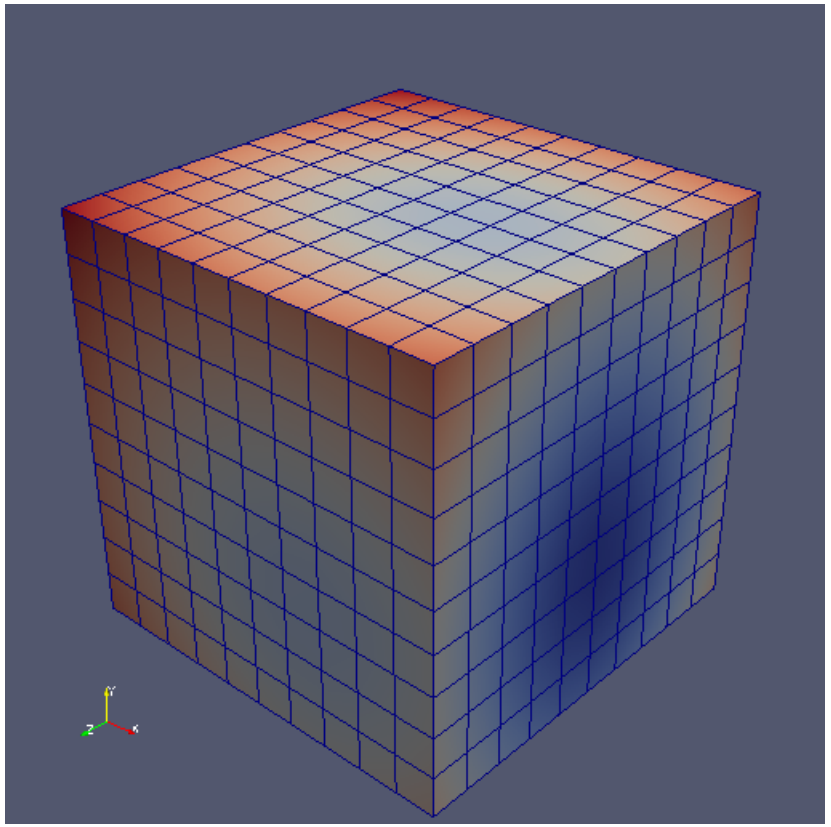
```
def RequestInformation():
 dims = [31,31,31]
 info = outInfo.GetInformationObject(0)

Set(WHOLE_EXTENT(),
 (0, dims[0]-1, 0, dims[1]-1, 0, dims[2]-1), 6)

Set(CAN_PRODUCE_SUB_EXTENT(), 1)
```

vtkImageData, ScriptRequestInformation



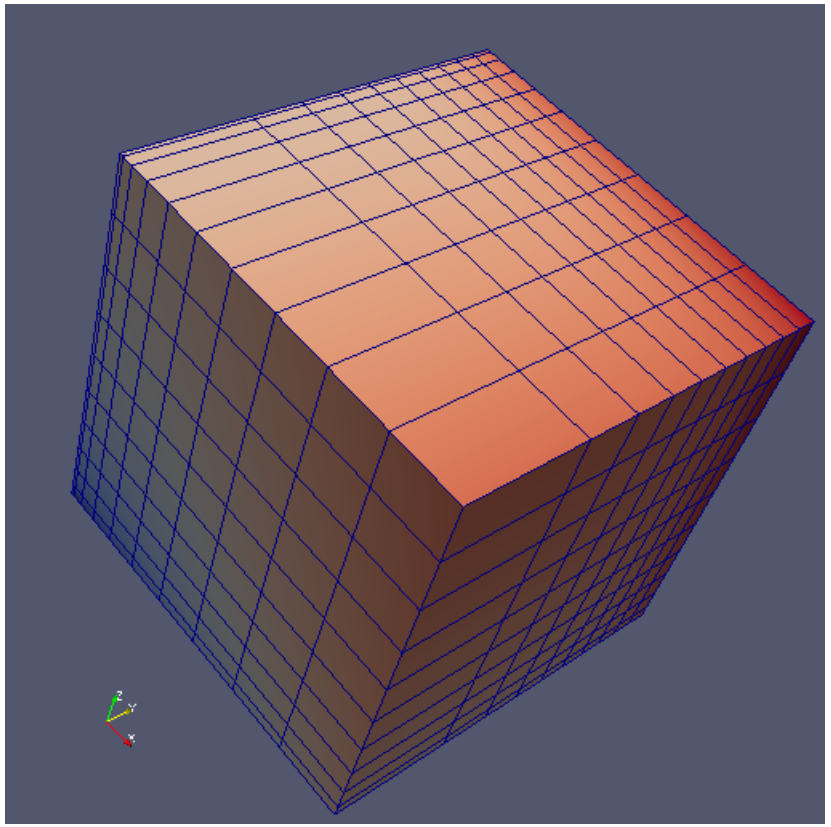
```
executive = self.GetExecutive()
info = executive.GetOutputInformation(0)
dims = [11,11,11]
info.Set(executive.WHOLE_EXTENT(),
    0, dims[0]-1, 0, dims[1]-1, 0, dims[2]-1)
info.Set(vtk.vtkDataObject.SPACING(), 1, 1, 1)
info.Set(vtk.vtkDataObject.ORIGIN(), 0, 0, 0)
info.Set(
vtk.vtkAlgorithm.CAN_PRODUCE_SUB_EXTENT(), 1)
```

vtkImageData, VTK python script



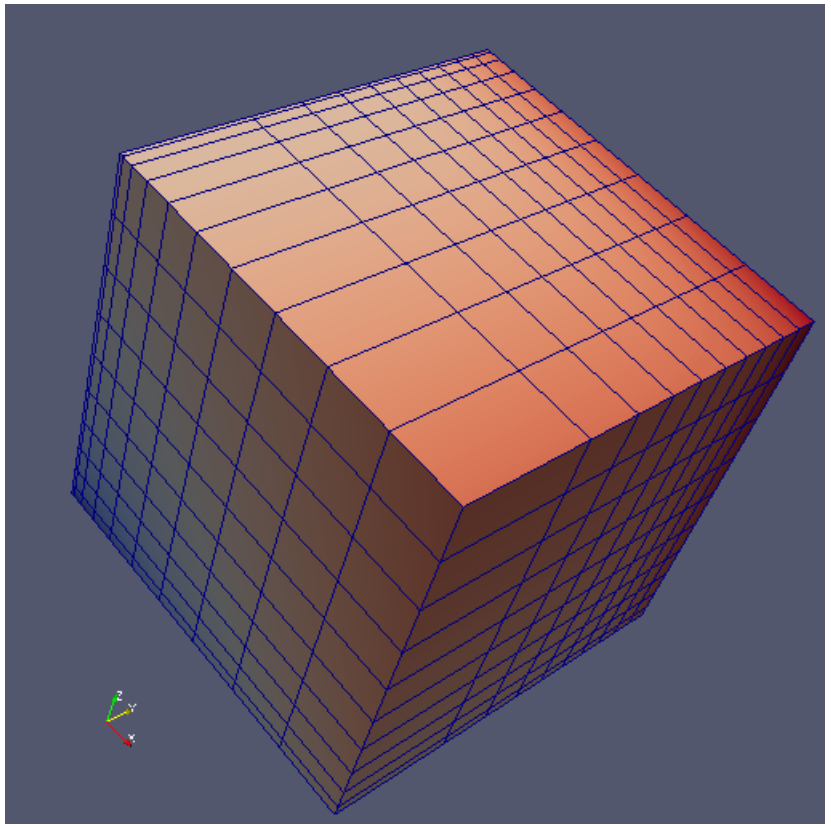
```
import numpy as np

executive = self.GetExecutive()

info = executive.GetOutputInformation(0)

whole = [executive.WHOLE_EXTENT().Get(info, i) for i in range(6) ]

exts = [executive.UPDATE_EXTENT().Get(info, i) for i in range(6) ]


output.SetExtent(exts)

output.PointData.append(data, “var_name”)
```

vtkRectilinearGrid, ScriptRequestInformation



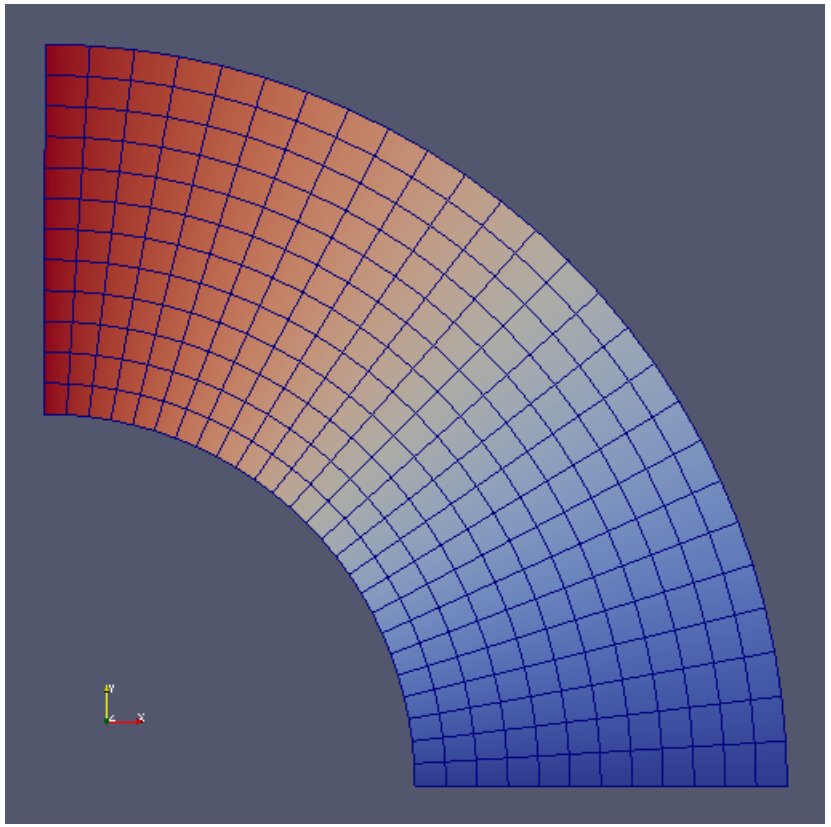
```
executive = self.GetExecutive()
info = executive.GetOutputInformation(0)
dims = [11,11,11]
info.Set(executive.WHOLE_EXTENT(),
    0, dims[0]-1, 0, dims[1]-1, 0, dims[2]-1)

info.Set(
vtk.vtkAlgorithm.CAN_PRODUCE_SUB_EXTENT(), 1)
```

vtkRectilinearGrid, VTK python script



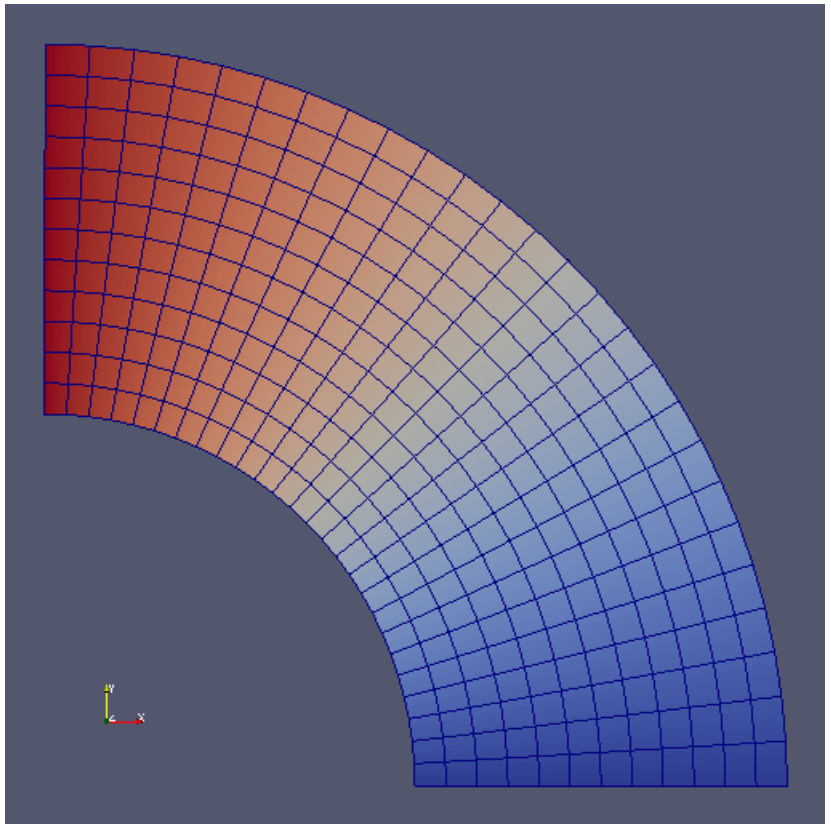
```
xaxis = np.linspace(0., 1., dims[0])
output.SetXCoordinates(
        dsa.numpyTovtkDataArray(xaxis, “X”)
        )
```

vtkStructuredGrid, ScriptRequestInformation



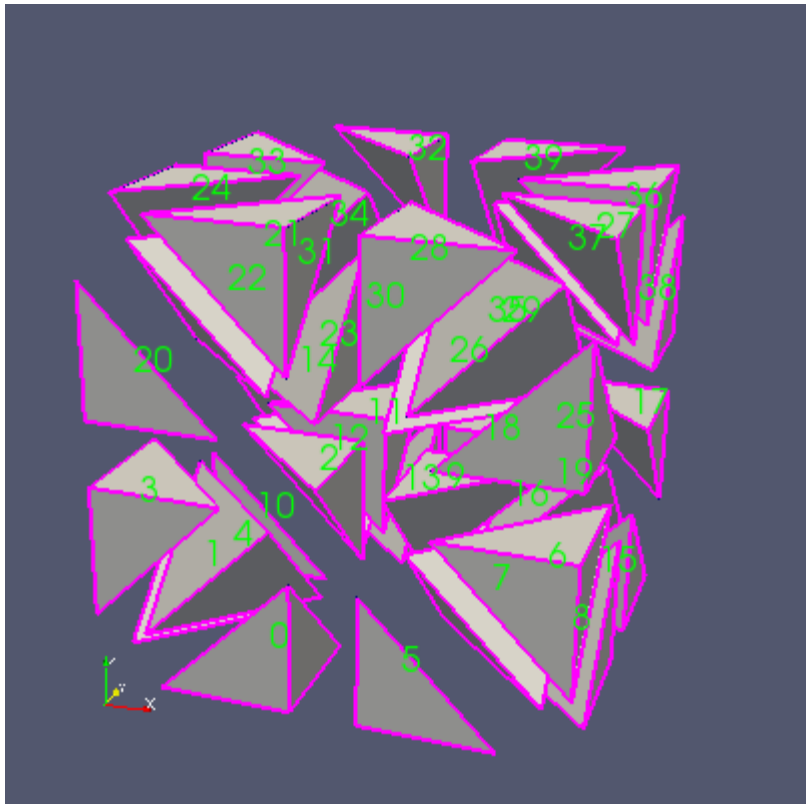
```
executive = self.GetExecutive()
info = executive.GetOutputInformation(0)
# make a 2D grid
dims = [13, 27, 1]
info.Set(executive.WHOLE_EXTENT(),
      0, dims[0]-1, 0, dims[1]-1, 0, dims[2]-1)


info.Set(
vtk.vtkAlgorithm.CAN_PRODUCE_SUB_EXTENT(), 1)
```

vtkStructuredGrid, VTK python script



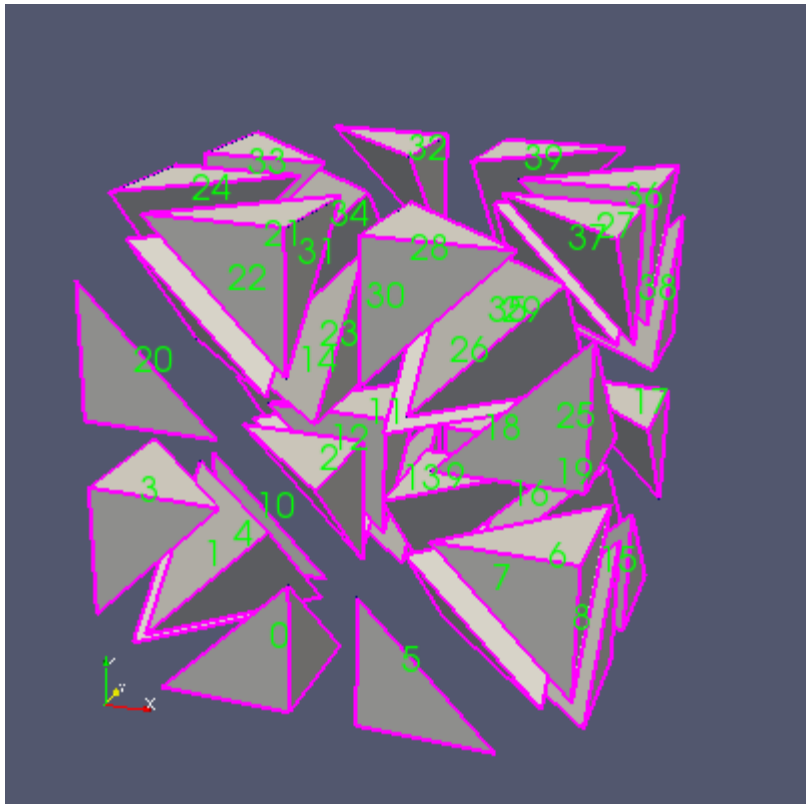
```
# make a 3D array of XYZ coordinates
pts = vtk.vtkPoints()
pts.SetData(
            dsa.numpyTovtkDataArray(coordinates, “coords”)
                )
 output.SetPoints(pts)
```

vtkUnstructuredGrid, VTK python script



```
#make an array of coordinates for 27 vertices

XYZ = np.array([0., 0., 0., 1., 0., 0., 2., 0., 0., 0., 1., 0., 1.,
1., ….., 2., 2., 2.])       # x0,y0, z0, x1, y1, z1, etc…

nnodes = XYZ.shape[0] / 3


#make a connectivity array for 40 tetrahedra


CONNECTIVITY = np.array([4, 4, 1, 10, 0, 4, 0, 4, 3, 12,
4, 4, 10, 13, 12, ……, 4, 16, 26, 25, 22])

nelts = CONNECTIVITY.shape[0] / 5
```

vtkUnstructuredGrid, Connectivity list



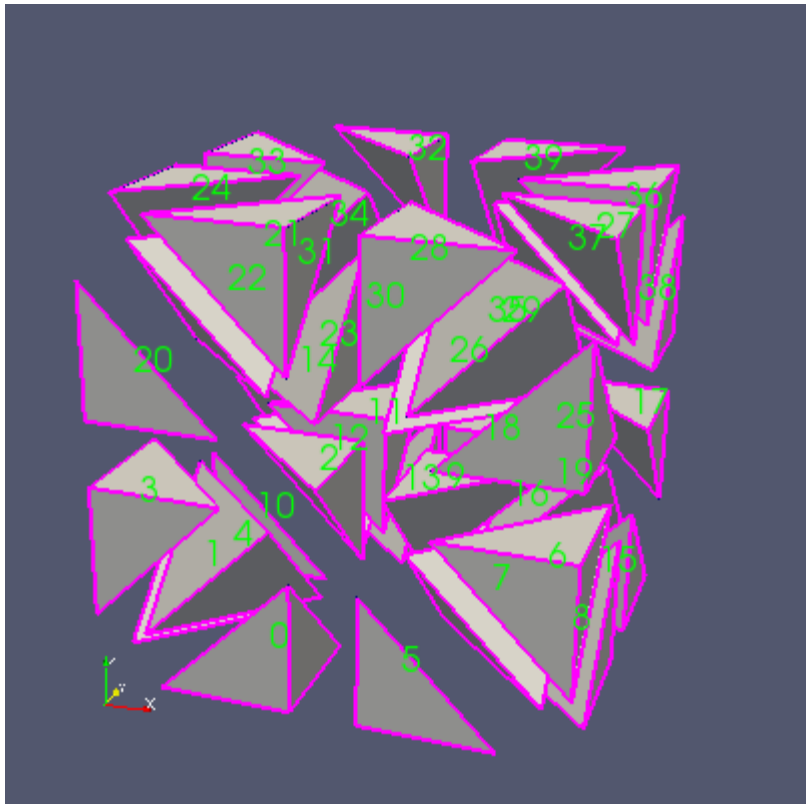
CONNECTIVITY = np.array([4, 4, 1, 10, 0, 4, 0, 4, 3, 12, 4, 4, 10, 13, 12, ……, 4, 16, 26, 25, 22])

A contiguous list of number of vertices (one integer N), followed by the N indices of the element's vertices.

Example: one tetrahedral (N = 4) using four vertices (4, 1, 10, 0)

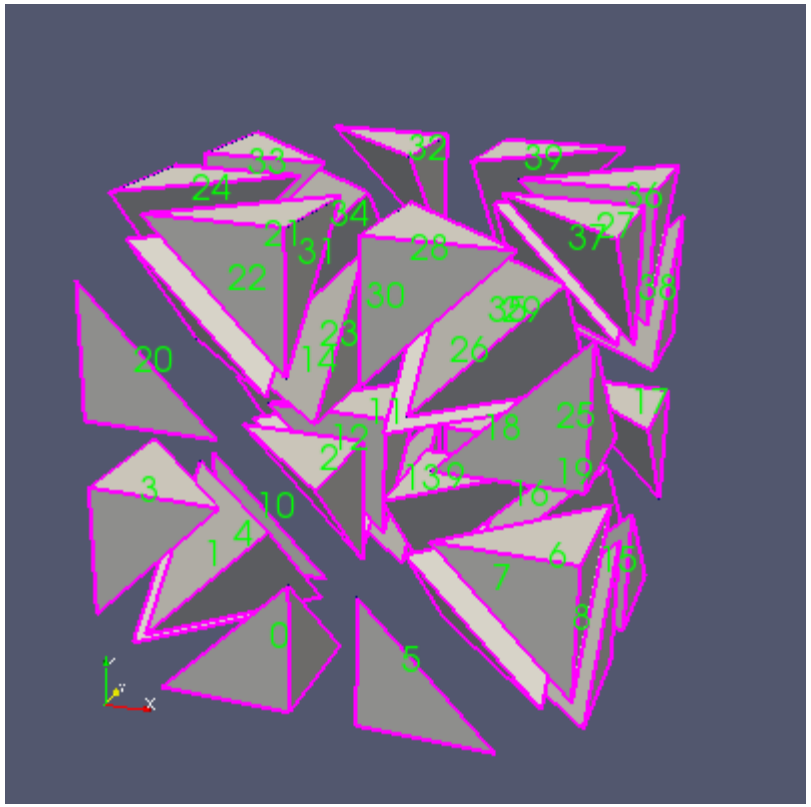
CONNECTIVITY = np.array([4, 4, 1, 10, 0, ….

vtkUnstructuredGrid, Cell types



```
#make an array of element types, and cell offsets


CELL_TYPES = np.full((nelts), VTK_TETRA, np.ubyte)
# np.array VTK_TETRA, VTK_TETRA, VTK_TETRA, …])


CELL_OFFSETS = np.arange(nelts)
            array([0,1,2,3,4, ….,39])
CELL_OFFSETS = 5 * CELL_OFFSETS
            array([0,5,10,15,20, ….,195])
1st element start at 0
2nd element starts at 5
```

vtkUnstructuredGrid, VTK python script



```
output.SetCells(CELL_TYPES, CELL_OFFSETS,
                CONNECTIVITY)

output.Points = XYZ.reshape((nnodes,3))
```

Time aware Python Programmable Sources

A time-aware source requirements:

- Advertize how many timesteps are available, and their values

```
outInfo = executive.GetOutputInformation(0)
outInfo.Remove( vtk.vtkStreamingDemandDrivenPipeline.TIME_STEPS( ) )
outInfo.Remove( vtk.vtkStreamingDemandDrivenPipeline.TIME_RANGE( ) )
outInfo.Set( vtk.vtkStreamingDemandDrivenPipeline.TIME_RANGE(), timeRange, 2)
outInfo.Set (vtk.vtkStreamingDemandDrivenPipeline.TIME_STEPS(), timesteps,
                                                    len(timesteps) )
```

- Get the value of the specific timestep requested

```
ts = executive.UPDATE_TIME_STEP().Get(outInfo)
output.GetInformation().Set(output.DATA_TIME_STEP(), ts)
```

Example: Generate time-dependent vtkImage datasets

- Try out pvGenerateImageData.py

- Try out GenerateTimeSeries.ipynb