ParaView/VTK Visualization Pipeline

December 2017

Jean M. Favre, CSCS
Introduction - Objectives

- Describe the VTK pipeline and VTK Objects
- Tie together numpy arrays and VTK Objects
- Write full pipelines in VTK using vtkpython
- Write ParaView pipelines
- Exercise the Python Programmable Source and Filter
Preamble: VTK numpy integration
What’s the big deal?

ParaView and VisIt use the VTK C++ library; a strong legacy with code that is +20 years old.

Contemporaneously, scipy and all the python-flavors of packages have flourished.

Enabling the transparent exchange of numpy arrays in VTK would bring both worlds together…

Support for parallelism (MPI-based) should also be supported
VTK Data Objects

The most fundamental data structure in VTK is a data object. Data objects can either be scientific datasets such rectilinear grids or finite elements meshes or more abstract data structures such as graphs or trees. These datasets are formed of smaller building blocks: mesh (topology and geometry) and attributes.

In general, a mesh consists of vertices (points) and cells (elements, zones). Cells are used to discretize a region and can have various types such a tetrahedra, hexahedra etc.
numpy provides an N-dimensional array type, the *ndarray*, which describes a collection of “items” of the same type. The items can be indexed using for example 3 integers.

http://scipy-lectures.github.io/intro/numpy/index.html

```python
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

```python
generated by https://github.com/guillaumepeyre/latex2sympy
```
numpy data arrays

>>> momentum = np.array(((0,1,2), (3,4,5), (6,7,8), (9,10,11)))

>>> momentum.shape
(4L, 3L)

>>> momentum[:, 0]
array([0, 3, 6, 9])

>>> momentum[:, 1]
array([ 1,  4,  7, 10])

>>> momentum[:, 2]
array([ 2,  5,  8, 11])
Numpy data arrays

momentum = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])

What does it take to visualize this array in ParaView assuming it sits on a 2x2 mesh?

Can we normalize the array using np.linalg.norm()?
Interface VTK and numpy

- Make the data arrays inter-changeable between numpy and VTK
- With the added advantage that the concept of the mesh will be known.

Motivation?

- How would you compute the gradient of a scalar on an unstructured grid in numpy?
The Visualization Pipeline
Visualization Pipeline: Introduction


«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 three types of modules: sources, filters and sinks.»
Visualization Pipeline: Definitions

- Modules are functional units, with 0 or more inputs ports and 0 or more output ports.

- Connections are directional attachments between input and output ports.

- Execution management is inherent in the pipeline
  - Event-driven
  - Demand-driven
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 thru the pipeline
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

**ParaView syntax:**

```
timeS = TimeSource()

# show data from timeSource1
representation1 = Show(timeS)
```

**VTK syntax:**

```
iD = vtk.vtkImageData()

geom = vtk.vtkGeometryFilter()
geom.SetInputData(iD)

m = vtk.vtkPolyDataMapper()
m.SetInputConnection(geom.GetOutputPort())
```
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

- VTK’s Rendering drives the execution:

  ```python
  ren = vtk.vtkRenderer()
  renWin = vtk.vtkRendererWindow()
  renWin.AddRenderer(ren)
  renWin.Render()
  ```
Basic ingredients of a pipeline (see ImageDataPipeline0.py)

```python
iD = vtk.vtkImageData()
geom = vtk.vtkGeometryFilter()
geom.SetInputData(iD)

m = vtk.vtkPolyDataMapper()
m.SetInputConnection(geom.GetOutputPort())

a = vtk.vtkActor(); a.SetMapper(m)
```

```python
ren = vtk.vtkRenderer()
ren.AddActor(a)

renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren)
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
renWin.Render()  # triggers execution
iren.Start()
```
SetInputData vs. SetInputConnection

iD = vtk.vtkImageData()

geom = vtk.vtkGeometryFilter()
geom.SetInputData(iD)

=================================

iD’s type:
vtkDataObject => Use SetInputData()

Geom’s type:
vtkAlgorithm => Use SetInputConnection() and GetOutputPort()

geom = vtk.vtkGeometryFilter()

m = vtk.vtkPolyDataMapper()
m.SetInputConnection(geom.GetOutputPort())
vtkDataObject vs. vtkAlgorithm
Basic ingredients of a pipeline (2)

```python
iD = vtk.vtkImageData()
dims = [31,31,31]
iD.SetSpacing(1., 1., 1.)
iD.SetOrigin(0, 0, 0)
iD.SetDimensions(dims)
xaxis = np.linspace(-.5, 1., dims[0])
yaxis = np.linspace(-1., 1., dims[1])
zaxis = np.linspace(-1., .5, dims[2])
[xc,yc,zc] = np.meshgrid(zaxis,yaxis,xaxis)
data = np.sqrt(xc**2 + yc**2 + zc**2)
```

```python
from vtk.numpy_interface import dataset_adapter as dsa
image = dsa.WrapDataObject(iD)
image.PointData.append(data.ravel(), "scalar")
```

```python
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> print np.ravel(x)
[1 2 3 4 5 6]
```
The teaser (http://www.kitware.com/blog/home/post/709)

```python
import vtk
from vtk.numpy_interface import dataset_adapter as dsa

s = vtk.vtkSphereSource()
e = vtk.vtkElevationFilter()
e.SetInputConnection( s.GetOutputPort() )
e.Update()
sphere = dsa.WrapDataObject( e.GetOutput() )
print(sphere.PointData.keys())                                ## ['Normals', 'Elevation']
print(sphere.PointData['Elevation'])                         ## is a numpy array
```
Dataset Attributes with numpy (http://kitware.com/blog/home/post/713)

```python
>> elevation = sphere.PointData['Elevation']

elevation.size                # returns 50
elevation.shape               # returns (50,)

>> elevation[:5]
VTKArray([0.5, 0., 0.45048442, 0.3117449, 0.11126047], dtype=float32)
```
w = vtk.vtkRTAnalyticSource()
w.Update()
image = dsa.WrapDataObject(w.GetOutput())
rtdata = image.PointData['RTData']

tets = vtk.vtkDataSetTriangleFilter()
tets.SetInputConnection(w.GetOutputPort())
tets.Update()
ugrid = dsa.WrapDataObject(tets.GetOutput())
rtdata2 = ugrid.PointData['RTData']

>>> rtdata[0:10:3]
VTKArray([  60.76346588,   95.53707886, 94.97672272, 108.49817657], dtype=float32)

>>> rtdata < 70
VTKArray([ True, False, False, ..., True], dtype=bool)

# generate a vector field.
>>> avector = algs.gradient(rtdata)

>>> algs.shape(rtdata)
(9261,)

>>> algs.shape(avector)
(9261, 3)

# access 0-th component
>>> avector[:, 0]
25
Exercise part 1

- `vtkpython`, (or `python(3)`), or `ipython(3)`
- `execfile("ImageDataPipeline0.py")`
- `exec(open("ImageDataPipeline0.py").read())`
- ‘q’ to exit the render window and return to python prompt
Exercise part 2

- Add colors for the scalar variable
- `ImageDataPipeline1.py`
Exercise part 3

- ImageDataPipeline2.py
- Display one isocontour
- What is the current threshold of the isosurface?
- Add other iso-contour surfaces at 0.5, 0.75, 1.25
ParaView/vtkpython in batch mode

- ParaView runs on any compute node with MPI and GPU-enabled graphics
- vtkpython runs on any compute node with GPU-enabled graphics

vtkpython Projects/VTK/ElevationBandsWithGlyphs.{sh,py}

Taken from:
http://www.vtk.org/Wiki/VTK/Examples/Python/Visualization/ElevationBandsWithGlyphs
Distributed data and Streaming

- Large data (when dividable) can be treated by pieces. The Source will distribute data pieces to multiple execution engines.

- Parallel pipelines will be instantiated to treat all pieces and create the graphics output. This is transparent to the user.

Wiki article: VTK-Parallel_Pipeline
Structured grids are split by IJK Extents

Use ExtractSubset

Parallel processing will enable requests for any subsets, including ghost-cells
XML format example with ghost cells

<VTKFile type="PStructuredGrid" version="0.1">
  <PStructuredGrid WholeExtent="0 65 0 65 0 65" GhostLevel="1">
    <Piece Extent="0 17 0 17 0 65" Source="d0372_00.vts"/>
    <Piece Extent="16 33 0 17 0 65" Source="d0372_01.vts"/>
    <Piece Extent="32 49 0 17 0 65" Source="d0372_02.vts"/>
    <Piece Extent="48 65 0 17 0 65" Source="d0372_03.vts"/>
    <Piece Extent="0 17 16 33 0 65" Source="d0372_04.vts"/>
    <Piece Extent="16 33 16 33 0 65" Source="d0372_05.vts"/>
    <Piece Extent="32 49 16 33 0 65" Source="d0372_06.vts"/>
  </PStructuredGrid>
</VTKFile>
How to write partitioned files? Structured Grids

// Use vtkXMLP*Writer with a serial program

N = 4

piw = vtk.vtkXMLPImageDataWriter()

piw.SetInputConnection(iD.GetOutputPort())
piw.SetFileName("/path/to/filename/file.pvti")
piw.SetNumberOfPieces(N)
piw.SetStartPiece(0)
piw.SetEndPiece(N-1)
piw.WriteSummaryFileOn()
piw.Write()  # this is best used with vtkImageData3.py
ParaView can read the data on any number of processors

ParaView can read any subsets (hyperslabs)
Running on 8 pv servers
Optimizing the reading order (X, Y or Z)

Reading 15 Gb of data with 12 cpus, with HDF5 hyperslabs

X hyperslabs: average read: 430 secs

Y hyperslabs: average read: 142 secs

Z hyperslabs: average read: 36 secs

Parallel Visualization is ALL about file I/O 😊
Zooming in to the interesting zone

How much data was read, isosurfaced, and never displayed in this picture?
Adjusting the Data Extents...

Reading much less data
display only 1/40-th of the data volume
25 millions instead of one billion cells
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)
vtkPythonAlgorithmBase
Advanced topic

It all starts here:

- [http://www.kitware.com/blog/home/post/737](http://www.kitware.com/blog/home/post/737)

- See files /users/jfavre/Projects/VTK/vtk*py
vtkProgrammableSource (Chapter 13 of Guide)
Three steps to define a 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

3. We’ll distinguish between three code sections:
   2. Code for ’RequestExtents Script’.
   3. Code for ’RequestData Script’.
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)
import numpy as np
executive = self.GetExecutive()
info = executive.GetOutputInformation(0)
whole = [executive.WHOLE_EXTENT().Get(info, i) for i in xrange(6)]
exts = [executive.UPDATE_EXTENT().Get(info, i) for i in xrange(6)]
output.SetExtent(exts)
output.PointData.append(data, “var_name”)
`vtkRectilinearGrid, ScriptRequestInformation`

```python
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

```python
xaxis = np.linspace(0., 1., dims[0])
output.SetXCoordinates(
    dsa.numpyTovtkDataArray(xaxis, "X")
)
```
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)
# 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.])
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
#make an array of element types, and cell offsets

CELL_TYPES = np.full((nelts), VTK_TETRA, np.ubyte)

CELL_OFFSETS = np.arange(nelts)
    array([0,1,2,3,4, ...,39])

CELL_OFFSETS = 5 * CELL_OFFSETS
    array([0,5,10,15,20, ...,195])
vtkUnstructuredGrid, VTK python script

```python
output.SetCells(CELL_TYPES, CELL_OFFSETS, CONNECTIVITY)
output.Points = XYZ.reshape((nnodes,3))
```
Exercise 1

See files:

- ParaView/{ImageData.py, RectilinearGrid.py, StructuredGrid.py}

- These files are python scripts including Programmable Sources
Complete examples with Source and Filter for time-aware processing

- The Filter will “pull” data from the Source

- Data Sources must:
  - Advertise how many timesteps they can provide, and give the values of the timesteps.
  - Respond to a request for a specific timestep value => create the data

```python
  ts =
  outInfo.Get(vtk.vtkStreamingDemandDrivenPipeline.UPDATE_TIME_STEP())
```
Complete examples with Source and Filter

Data Filters must:

- Ask how many timesteps are available, and get the values of the timesteps.

  ```python
  self.tlen = outInfo.Length(executive.TIME_STEPS() )
  self.ts = [ executive.TIME_STEPS().Get(outInfo, i) for i in range(self.tlen) ]
  ```

- Get a specific timestep value

  ```python
  ```

- If pulling all timesteps one at a time to do temporal statistics…

  ```python
  request.Set( vtk.vtkStreamingDemandDrivenPipeline.CONTINUE_EXECUTING(), 1)
  request.Remove( vtk.vtkStreamingDemandDrivenPipeline.CONTINUE_EXECUTING() )
  ```
Demonstration TransientPoints.py

- Provides a source of time-varying particle points

- Use the TemporalParticlesToPathlines filter
Demonstration TemporalStatistics.py

- Provides a source of 100 timesteps (data on a 4x2x1 plane)
- Define a filter to do temporal averages of the CellData

------------------ INIT Phase-----------------------
('INIT: CurrentTimeIndex = ', 0, ', tlen = ', 100)
('set data =', VTKArray([ 0.,  0., -0.]))

------------------ ACCUMULATE Phase-----------------------

FINISH: Finish
('avg_data =', VTKArray([ 0.14803234,  0.11384104, -0.18647398]))
Thank you for your attention.