An Introduction to Scientific and Data Visualization with ParaView

https://github.com/ResearchComputing/Scientific-and-Data-Visualization-with-Paraview

Part 1 – Introduction & Overview
Part 2 – Overview part 2
Part 3 – Overview part 3 and Python Scripting
Part 4 – Python Scripting and Animation

Paraview Python (pvPython)

Way and by far the best way to really get heavy mileage out, spelled getting want you really need, out of Paraview requires you to jump into the wonderful world of pvPython (NO sarcasm implied here either!) In fact, a lot of things you may want/need to do in Paraview you can only accomplish with the assistance of Python.

When you installed Paraview it linked with the native Python on your system or created a small Python install for you. If you have your own install of Python, say Anaconda Python, you just need to make sure you add the path of your Python to your PYTHONPATH environment variable. If you do not have an install of Python and would like one, we HIGHLY recommend using Anaconda and its free (https://www.anaconda.com/download/)!

The best thing of getting started with pvPython is it does not require you to know python. Don't get me wrong to get the best bang for the buck you should have a basic understanding of Python but not much beyond the basics. How can that be? Macro Tracing! Basically that means that you run the gui as normal and have the python code generated for you. Then you can go back and edit that code to your desires.

Let's start with a simple setup and explore what we can do.

- Reset you're your session (Edit->Reset Session)
- Click Tools->Start Trace to open the Trace Options
- The 'Properties To Trace On Create' dropdown lets you choose just how much information gets 'Traced' For now we will leave it at the default 'any *modified* properties
- Likewise, we will leave everything else at the default
- Once you click OK you will start tracing your actions. Click OK
- Click Sources->Sphere to create a sphere
- Set 'Radius' to 1
- 'Theta Resolution' to 32
- 'Phi Resolution' to 32
- Change 'Coloring' to 'Normals'
- Change 'Coloring Attribute' to 'Y'
- Make sure representation is set to 'Surface'
- Now click Tools->Stop Trace to end our tracing
- As soon as you stop your trace the Python Shell will open and lo and behold there's your Python script already to use.

Macros

Let's first look at Macros. Macros are short simple scripts whose intent are to provide a rapid solution to something you will be doing frequently. So pretend that we want to use this sphere we create more times then you want to go through this process each time.

- From the Python Shell click on File->Save as Macro
- The 'Save Macro' panel opens, call the macro 'My_Sphere' and save it to where ever your course files are. NOTE the .py extension will be added for you.
- Close the Python Shell and reset your session
- The first thing we need to do is load our macro into the macro list. Click Macros->Add new macro….
- Find and select your 'My_Sphere.py' file
- Our macro is now ready to go. Click Macros and you should now see your new macro. Click on My_Sphere
- There you go you should now see your sphere.

- CONGRADULATIONS you just created and ran your first pvPython script.
- If you want/need to you can edit your macro a few different ways.
- The first way is to select Macros->Edit->My_Sphere this opens your script back into the Python Shell

Running Python Code

Another, and the more common way of running pvPython code it to just work from the Shell

- Go ahead and close anything that's open and reset your session
- Select Tools->Python Shell
- You now have a Shell with what's required to get you started with a new script. Namely you need to import the Paraview python core package;

```
from paraview.simple import *
```

- Let's ignore that for now and load and run our Sphere code. From the Shell click on 'Run Script' and select your 'My_Sphere.py' script.
- Notice that it ran but you can't see the code. Sadly, this Shell is basically for live running of existing code (most people tend to write their code in some other app, save it and then run it in Paraview). But there really should be the ability to load your script from here.
- What we can do is run Paraview with just Python. From the Shell type the following lines

```
sphere1 = Sphere()
renderView1 = GetActiveViewOrCreate('RenderView')
sphere1Display = Show(sphere1, renderView1)
RenderAllViews()
```

- Notice that when we enter the first line (`sphere1 = Sphere()`) a Sphere source layer is added to the Pipeline Browser window.
- The next three lines are what's needed (there are other methods as well) to start a renderer and display our sphere. Thus once you hit enter on the 4th and last line your sphere will show up in the render panel.

Modifying Python Code

This is fine and dandy if you really know what you're doing but who knows Paraview that well that's not working for Kitware? Let's get back to working on our 'My_Sphere.py' code. We kind of screwed up and forgot to set the color map to what we really want to use so let's fix that.

- Close panel that's open, reset your session.
- First run your 'My_Sphere.py' code
- Click Tools->Start Trace
- This time make sure you check both the 'Fully Traced Supplemental Proxies' to make sure we see the code for the changes we are about to make.
- Also check 'Show Incremental Trace' so you can see the code as you work
- Click OK
- Now open the Color Map editor
- Select the 'Jet' color map preset
- Change the green color to white (we are making a simple divergent color map)
- Stop your trace (Tools->Stop Trace)
- Now select and copy all the code from '*# get color transfer function/color map for 'Normals''* down to '*#### uncomment the following to render all views'*
- From the Shell menubar select File->Open and navigate to and open your 'My_Sphere.py' code
- Scroll to the bottom of the code and just above the '*#### uncomment the following to render all views'* line, paste your new code.

- First to prove it works save it as a new Macro named 'My_Sphere_2.py'. Close anything that's open and reset your session. Then run your new My_Sphere_2.py macro. NOTE: This time it should have automatically loaded it for you.
- Boom baby, your new code works!!!

Code Clean Up

- Go ahead and open your code
- Notice there are a lot of lines that say things that does not appear to do anything with what we told Paraview to setup. These are, for the most part, just default settings and we can get rid of those.
- AT the top you should find a section under a comment (in Python single line comments start with '#' signs) that says '*# trace defaults for the display properties.'* Delete this entire section.
- We don't really need to reset the camera nor update the renderview so delete those lines
- In fact, we see a lot of renderView1.Update() lines delete all of them and the comments above them
- Setting the resolutions to 32 still give us some ugly spheres so increase both resolutions lines to 64
- Delete the following lines

```
# set scalar coloring
ColorBy(sphere1Display, ('POINTS', 'Normals', 'Magnitude'))

# rescale color and/or opacity maps used to include current data range
sphere1Display.RescaleTransferFunctionToDataRange(True, False)

# show color bar/color legend
sphere1Display.SetScalarBarVisibility(renderView1, True)
```

- Let's decide not to include the colorbar legend. So delete the following lines

```
# rescale color and/or opacity maps used to exactly fit the current data
range
#sphere1Display.RescaleTransferFunctionToDataRange(False, False)

# Update a scalar bar component title.
#UpdateScalarBarsComponentTitle(normalsLUT, sphere1Display)
```

- Also these lines which update the legend with our coloring attribute change to the 'Y' normal direction;

```
# rescale color and/or opacity maps used to exactly fit the current data
range
#sphere1Display.RescaleTransferFunctionToDataRange(False, False)

# Update a scalar bar component title.
#UpdateScalarBarsComponentTitle(normalsLUT, sphere1Display)
```

- We have not discussed the camera yet (that comes next week) so delete the lines dealing with the camera settings.

```
#### saving camera placements for all active views

# current camera placement for renderView1
#renderView1.CameraPosition = [0.0, 0.0, 3.2903743041222895]
#renderView1.CameraParallelScale = 0.8516115354228021
```

- The first set of lines that were added when we created our new colormap are default settings which will be replaced by what we really wanted to add. So lets delete those default lines;

```
# get color transfer function/color map for 'Normals'
#normalsLUT = GetColorTransferFunction('Normals')
```

```
#normalsLUT.RGBPoints = [-0.9987165331840515, 0.231373, 0.298039, 0.752941,
1.50164813916831e-09, 0.865003, 0.865003, 0.865003, 0.9987165331840515,
0.705882, 0.0156863, 0.14902]
#normalsLUT.ScalarRangeInitialized = 1.0
#normalsLUT.VectorComponent = 1
#normalsLUT.VectorMode = 'Component'
```

We will leave the '*RenderAllViews()*' lines as we may need them later.
But scroll back up to the top and delete these two lines;

```
#### disable automatic camera reset on 'Show'
#paraview.simple._DisableFirstRenderCameraReset()
```

Your final code should look like the following:

```
#### import the simple module from the paraview
from paraview.simple import *

# create a new 'Sphere'
sphere1 = Sphere()

# get active view
renderView1 = GetActiveViewOrCreate('RenderView')

# show data in view
sphere1Display = Show(sphere1, renderView1)

# Properties modified on sphere1
sphere1.Radius = 1.0

# Properties modified on sphere1
sphere1.ThetaResolution = 64

# Properties modified on sphere1
sphere1.PhiResolution = 64

# get color transfer function/color map for 'Normals'
normalsLUT = GetColorTransferFunction('Normals')

# set scalar coloring
ColorBy(sphere1Display, ('POINTS', 'Normals', 'Y'))

# Apply a preset using its name. Note this may not work as expected when presets have
duplicate names.
normalsLUT.ApplyPreset('jet', True)

# Properties modified on normalsLUT
normalsLUT.RGBPoints = [-0.9987165331840515, 0.0, 0.0, 0.5625, -0.7767797477468252,
0.0, 0.0, 1.0, -0.2694946680309176, 0.0, 1.0, 1.0, -0.015852627531230512, 1.0, 1.0,
1.0, 0.2377894129684568, 1.0, 1.0, 0.0, 0.7450744926843644, 1.0, 0.0, 0.0,
0.9987165331840515, 0.5, 0.0, 0.0]

#### uncomment the following to render all views
# RenderAllViews()
# alternatively, if you want to write images, you can use SaveScreenshot(...).
```

Save your final macro as ‘My_Sphere_Cleaned.py’, close the Shell, reset your session and run your new code. Boom baby nice clean code!

So the massive question is how the living heck did I know to delete all those lines and keep the others? Experience, experimenting and well, honestly, a lot of screwing up over time. So I HIGHLY suggest if you go this route in the future, that before you delete questionable lines just comment them out first. Save the code and run it to make sure everything is working the way you want it to. If all is well then go back and delete. If not just work through your code uncommenting bits till everything works right.

Exercise:

Create a Python script that

- Opens the CAT_Torso.vti dataset
- Adds a contour filter on Scalars_ with 10 isovalues
- Set Representation to Surface
- Set Coloring to Normals along the Y normal
- Save your python file as Torso_01.py
- Clean your code to just what is needed.
- Save your python file as Torso_02.py

Animation in Paraview GUI

While Paraview has ‘some’ built in animation functionality the real power of animation inside of Paraview is better harnessed using Python. Basically anything that you have control over in the gui you have control over in Python which means you can animate any of these controls, parameters, filters, ect. Today we will look just at a couple quick examples, one from the gui and one from python just to give you a feel of what you can do.

NOTE: While, at least within the gui, you can save animations directly out to an .avi movie file I highly recommend NEVER do this with the exception as just doing a quick demo to show that you can. You should ALWAYS render each individual frame out as an image file, usually a .png, and then comp them later.

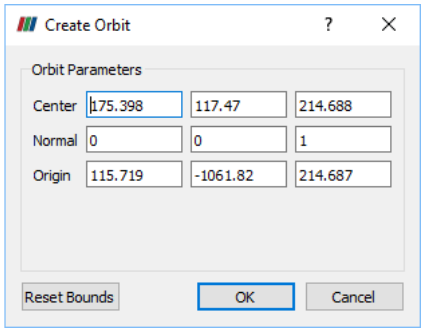
Description of Proper View Planes

Pretty much ALL scientific visualization applications do not handle view planes correctly, or even at all! Like Paraview most just consider them all as one plane which is a VERY BAD thing to do!!!

Animating Inside the Paraview GUI

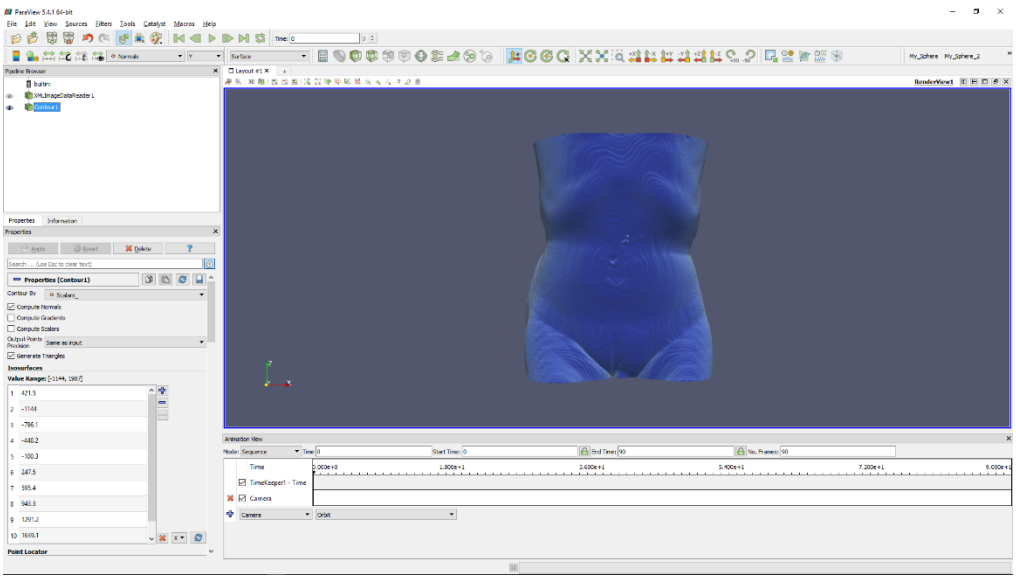
We will create a quick and dirty orbiting animation of our contoured Torso model and save it out as an .avi (just to show you, you can but refer to my note above).
Animation in Paraview is controlled from the ‘Animation View’ panel which can be found under the ‘View’ menu.

- Reset your session
- Run either your Torso_02.py script or the one provided on Github
- Open the Animation View panel, click on View->Animation View
- We want to create an animation based off camera movement, in this case a camera orbit around the object.
- In the Animation View change ‘Contour1’ to ‘Camera’
- Make sure ‘Orbit’ is selected in the methods dropdown next to ‘Camera’
- Click on the blue ‘+’ button to the left of ‘Camera’ to add this to the Timeline
- You will be presented with a new panel called ‘Create Orbit’ which asks you to specify parameters important to the camera setup and movement. Just select OK to use the camera and object positioning as is.



The Create Orbit camera/object options panels

- We will leave the Time parameters just as they are to quickly another issue with Paraview-GUI animation



Your Paraview setup should look like this

- You can play your animation now by simply clicking the play button in the top toolbar
- To create your movie file, click on File->Save Animation to open the Save Animation Options panel
- Set the desired Image resolution. Remember its usually best to click the lock button to lock the aspect ratio. Click OK when ready
- The file browser now pops up asking for the path and file name for your movie. Navigate to where you want to save your animation then give it a file name (eg 'Torso_orbit')
- For 'Files of Type' we will go ahead and just leave it as a .avi file. Normally this is where you would tell it to save each frame out as a .png file!
- Click OK
- It will now render all the frames (in theory 10) and save it as an .avi file which you can now play in most any common media players
- Play the file, what issues do you see if your animation?

Animation with pvPython

Well that was not pretty and most other animations you may try to set up begin to be more complex then you want so again, 'just because you can doesn't mean you should'. So let's move to how to do things properly via pvPython. True if you are new to Python this may seem like a much longer and harder route but as you become comfortable with it and get to know pvPython you will find this a very fast and effective method. In fact, for real, spelled massive, sciviz datasets which require a super computer to render it is your only solution.

For this exercise we will go back and modify our Torso_02.py code to incorporate making a clipping animation along the y-axis and saving it all out to individual frames.

I will leave it as an exercise for you to modify your code to add the above features but do to the high probability of lack of time I'll just post my final code, Torso_03.py which can be found in the Github repository and attached below. In lieu of creating the code I'll just explain it. Contrary to the length it only took me about 10 minutes from start to finish to create this code.

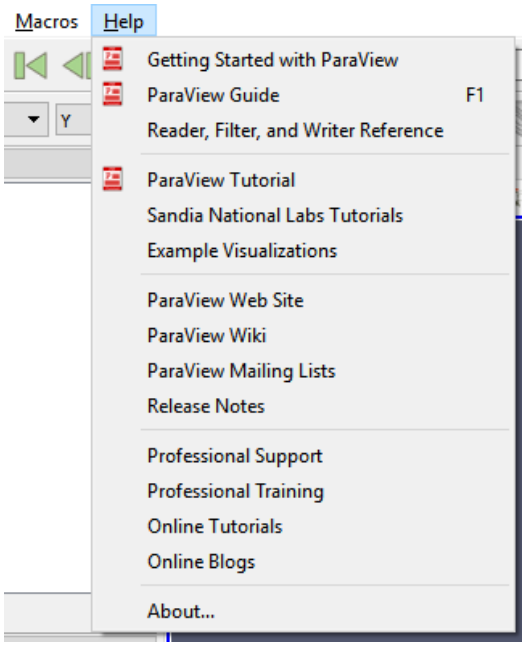
```
from paraview.simple import *

def getData(fName):
        """
        Load the desired data file

        Args: fName - Path and name of the data file to open

        Returns: data - A handle to the full, opened dataset
        """
        data = XMLImageDataReader(FileName=fName)
        return data

def setCamera():
        """
        Sets up the camera view for this scene

        Args: None

        Returns: renderView - A handle to the current active renderview
        """
        # get active view
        renderView = GetActiveViewOrCreate('RenderView')
        # reset view to fit data
        renderView.ResetCamera()

        # current camera placement for renderView
        renderView.CameraPosition = [178.12559509277344, -1053.147120033127, 214.6875]
        renderView.CameraFocalPoint = [178.12559509277344, 119.67813873291016, 214.6875]
        renderView.CameraViewUp = [0.0, 0.0, 1.0]
        renderView.CameraParallelScale = 303.54951354594255

        return renderView

def addContour(inData):
        """
        Adds a contour filter to the dataset using 10 hardcoded but even divisions

        Args: inData - The object layer (in this case the dataset itself) to apply the
                                                                contour filter to.

        Returns: contour - A handle to the new contoured filtered layers data
        """
        contour = Contour(Input=inData)
        contour.ContourBy = ['POINTS', 'Scalars_']
        contour.Isosurfaces = [421.5, -1144.0, -796.1, -448.2, -100.3, 247.5, 595.4, 943.3, 1291.2,
1639.1, 1987.0]

        return contour

def addClip(inData, y):
        """
        Adds a clip filter to the input data (in this case the contour layer) and clipping
        along the y-axis at a specified position

        Args: inData - the object layer (in this case the contour layer) to which the clipping
                       will be applied

              y - the y-axis position where the clipping will be made

        Returns: clip - A handle to thdatae new clipped layers
        """

        clip = Clip(Input=inData)
        clip.ClipType = 'Plane'

        # init the 'Plane' selected for 'ClipType'
        clip.ClipType.Origin = [175.39791604876518, y, 214.6875]

        # Properties modified on clip1.ClipType
        clip.ClipType.Normal = [0.0, 1.0, 0.0]

        clip1Display = Show(clip, renderView)
        return clip
```

```
##################
###### Main ######
##################
# Read in Data
fName = 'D:\\CRDDS\\Paraview_Tutorial\\CAT_Torso.vti'
data = getData(fName)

# Set up the camera view
renderView = setCamera()

# Add a contour filter to the data
contour = addContour(data)

# Add transfer function coloring the normals along the Y axis (depth in this case)
contourDisplay = Show(contour, renderView)
normalsLUT = GetColorTransferFunction('Normals')
ColorBy(contourDisplay, ('POINTS', 'Normals', 'Y'))

#Hide the contour layer so only the new clipped contour layer will be visible
Hide(contour, renderView)

#Create a loop for animating clipping along the y-axis and saving the image to disk
for y in range(0,240):
        #Create the clip along the the y axis position (0-240)
        clip = addClip(contour, y)

        # Hide the clipping plane widget
        Hide3DWidgets(proxy=clip.ClipType)

        #Update the renderView
        renderView.Update()

        #Create a padding for the frame numbers and build the image file name (iName)
        fnum = str(y)
        fnum = fnum.zfill(3)
        iName = 'D:/CRDDS/Paraview_Tutorial/Images/TorsoAnim/Torso_clip'+fnum+'.png'

        #Save the render to disk making sure they are all the same size
        SaveScreenshot(iName, renderView, ImageResolution=[800, 473])

        # Delete clip so we just create a replacement layer and not 240 new layers
        Delete(clip)
        del clip
```

- Reset your session
- Run the new Torso_03.py code
- Make sure change the path for both fName and iName to match your setup. Ideally I would create a new variable named ‘baseDir’ which holds the path to the python code and then join it to my file name. Note I have a special directory just for my animation frames. I highly recommend you always do this!
- Wait awhile for all the frame to render (note you can watch your directory
- When done you can composite all the frames in your favorite (hopefully you have one) movie app. If not either later this year or early next year, I’ll be hosting another short course on creating movies using applications I use when working on feature movies or TV shows (eg anything seen at the theaters or on TV and now they are FREE(ish)!)

Where Do You Go From Here?

Well that's it for this short course. You should now know enough to comfortably explore and work in Paraview to create most anything you may need, SciViz wise. Yet we have just barely scratched the surface on everything that can be done in Paraview. So what should you do if you want to learn more?



By clicking on the 'Help' menu you will see a bunch of links to just a few of the great many resources available to learning and helping you work with Paraview. If you do have questions you have a few options to really help you out.

- Google – Happily googling Paraview is easy and simple. Try Googling something for Visit! (Note if you do first try googling 'visit visualization' + whatever you want.
- The Paraview Mailing lists – there are a few of there and they are all VERY active. You should get responses back at worst within a day usually within just an hour or two or even faster.
- Consulting hours – I offer free SciViz/Data viz consulting hours on Tuesdays from 1-2pm here in Norlin E206. Please feel free to pop in!

Good Luck – now go make some pretty images and movies