An Introduction to Scientific and Data Visualization with ParaView

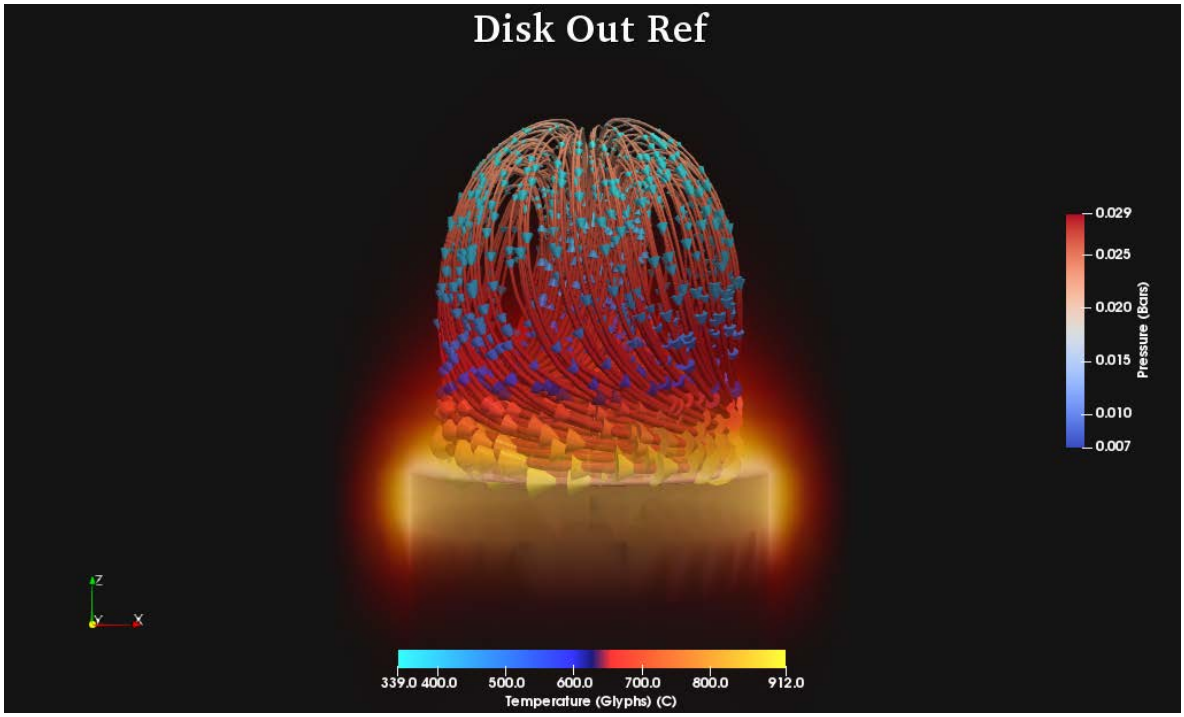
https://github.com/ResearchComputing/Scientific-and-Data-Visualization-with-Paraview

Part 1 – Introduction & Overview

Part 2 – Overview part 2

Part 3 – Overview part 3 and Python Scripting

Part 4 - Animation



Last week we ended with this being our final output. The saved state for this can be found in the Github repository for this class under the name 'Week_2_Disk_Out.pvsm'. We will start at this point and explore some more features available in Paraview.

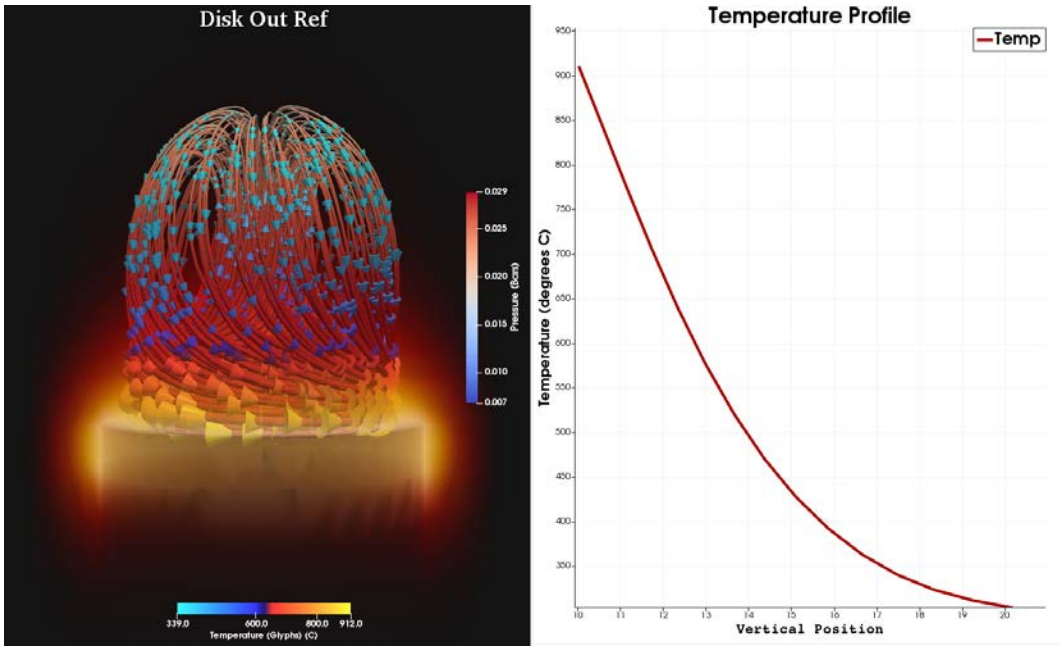
Let's Do a Little Data Analysis

Paraview allows you to not only visualize 3D and higher dimensional datasets but offers a small suite of charting and statistical functions as well. We'll take a quick peek at this by plotting a temperature profile of our data.

- Click on the 'disk_out_ref.ex2' layer in the Pipeline Browser to make it active
- Click Filters->Data Analysis->Plot Over Line
- Rename the layer to 'Temperature Profile Plot'
- In the Layout #1 window move the top point and bottom points try to set the line vertically down the center of the object.

- Remember that you are working in 3D and you're only looking at a 2D representation so click the '+Y' view direction toolbar button and once again try to align your points centered and vertically.
- Isn't that a pain to set that up perfectly? Well it doesn't have to be as you'll see next.
- In the 'Line Parameters' section set 'Point 1' to [0, 0, -4]
- Set 'Point 2' to [0, 0, 8]. There, a nice and easy centered vertical line!
- Even easier in the 5.4.x versions you can just click on the 'Z-axis' button and it will align it automatically for us. Not this works because we have a centered and symmetrical object we are working with and I choose the center as the area we want to probe.
- Scroll down to 'Series Parameters' and uncheck everything except for 'Temp'
- Double click the color swab next to the 'Temp' variable in the 'Series Parameters' section and change the color to a dark red.
- Change 'Line Thickness' to 4
- Change 'Chart Title' to 'Temperature Profile'
- Change the font size to 30 and make it Bold
- Change the 'Legend Properties' font size to 24 and make it Bold

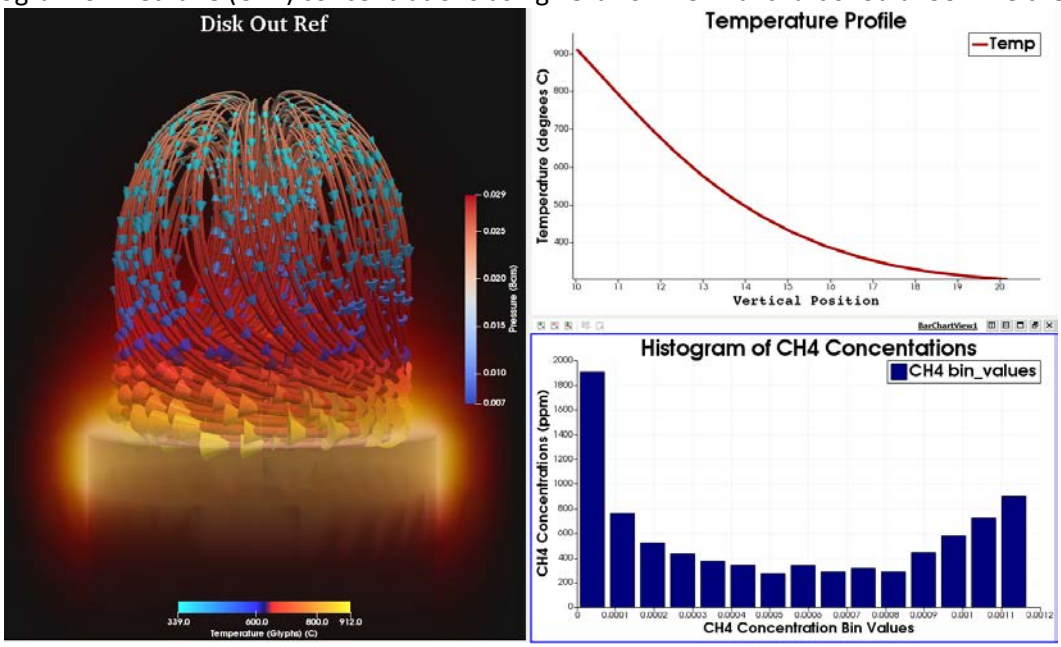
- Change the ‘Left Axis Title’ to ‘Temperature (degrees C)’ change the font size to 20 and make sure its Bold
- Change the ‘Bottom Axis Title’ to ‘Vertical Position’ change the font size to 20 and make sure its Bold
- Hide the visibility of this (‘PlotOverLine1’) layer (your chart will remain but the line will be hidden in the ‘Layout #1’ window.)



Your viewports should resemble the above

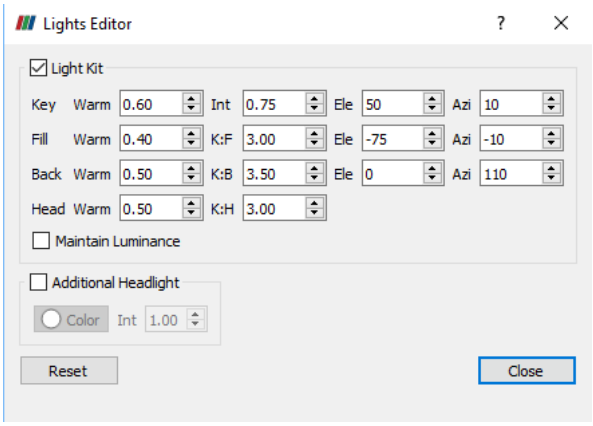
Exercise:

Create a histogram of methane (CH4) concentrations using 15 bins. The final chart should look like the following.



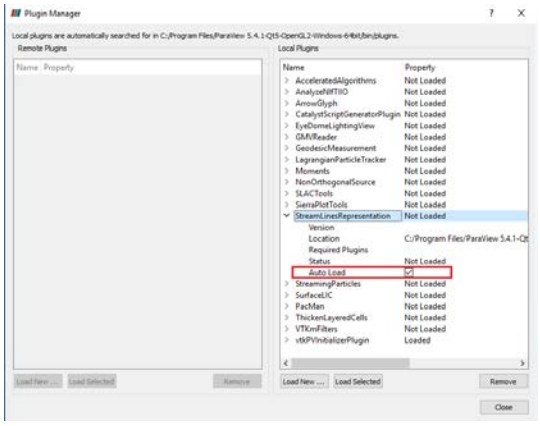
Lighting

Paraview provides you with a default four-point (Key, Fill, Back, and Head) lighting rig with the potential of a static 5 head light. Sadly it's one of the worst lighting rigs in any CG application. In fact the only way to make it worse is to simply not have any control. Having said this with some practice and patients you can provide a nice lighting setup, if needed, for your scene.
Close both plot windows



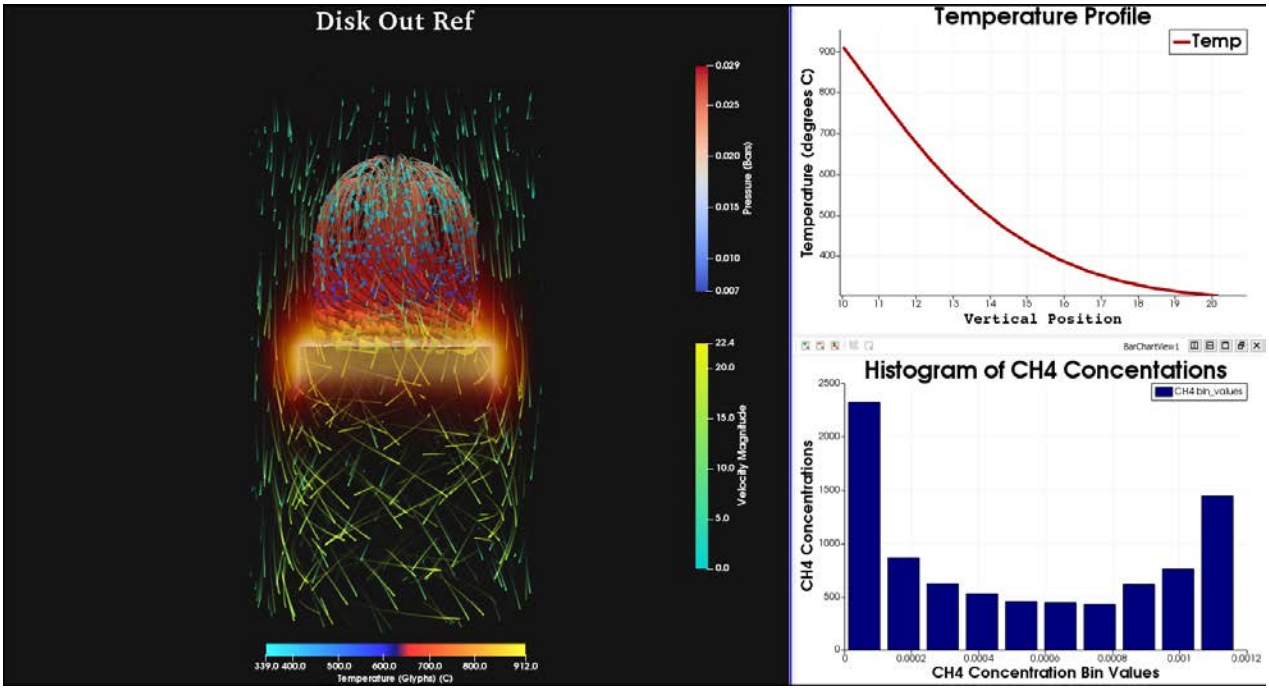
Loading and Using New Plugins

- Click on Tools->Manage Plugins
- Click on the arrow to the left of 'StreamLinesRepresentation' to expand it
- Click on 'Auto Load', this will automatically load the plugin whenever you start Paraview now
- Click the 'Load Selected' button and close



- If you were successful it should say 'Loaded' next to your plugin name
-
- Now let's open a new instance of our original object, File->Open 'disk_out_ref.ex2'
- Change the layer name to 'Anim Stream Lines'
- For variables this time just select the 'V' variable
- Set the 'Representation' to 'Stream Lines' (this really should be a Filter but….)
- Set 'Coloring' to 'V' and the coloring attribute to 'Magnitude'
- Scroll down to the 'Stream Lines' section

- Set ‘Alpha’ to 0.1. The higher the value (0-1) the longer the tail. If you just want particle and no trail then set ‘Alpha’ to 0
- Set ‘Step Length’ to 0.005
- Set ‘Number of Particles’ to 500
- Set ‘Max Time to Live’ to 150
- Change the transfer function color bar preset to ‘blue to yellow’
- Change the blue color handle to a more cyan color
- While it’s not obvious and it’s a little known secret (that needs shared better) you can actually change the thickness of the line by going up to the Styling section and changing ‘Line Width)
- If you desire you can fix up the legend/colorbar
- Click on the Loop video control button then click the Play button



Final Check point: Your viewports should resemble the above.

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)
```

- Lets 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_3.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 for next week:

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
- Clean your code to just what is needed.

If you are not familiar with Python coding, or more importantly coding at all. I suggest doing a little Googling on using python for loops and python functions.