# Exercise Sheet for Visualization Basics with ParaView

ParaView is a popular open-source application for the visualization of 2D and 3D scientific data. It is designed to run on a desk top (or parallel cluster for large data) and comes packaged with a wide variety of scientific software. Its graphical user interface (GUI) makes it quick to test visualization techniques on data. Once suitable techniques are selected you can build your own bespoke ParaView application, called a state, or use the knowledge gained to program your own application in another system/language.

This is a general hands-on course that introduces some of the basic visualization techniques using research data produced at the University of Leeds.

# 2D Data

This data has been provided by Ilkka Matero and Lauren Gregoire from the Faculty of Earth and Environment at the University of Leeds in 2016. The data is part of a climate study.
Climate:

- xlrjn.temp2m.monthly.nc

## Exercise 1 – Load Data

Load the vtk file called xlrjn.temp2m.monthly.vtm into ParaView" and then set the reader parameters:

- Point Arrays

Press the "Apply" button and then look at the in the RenderView.

- Change the Coloring from "Solid Color" to "temp"

Save the State.

### Questions:

Where are the hottest and coldest regions on the map?

Where are the regions that are below the freezing point of water?

## Exercise 2 – Alter Data Values

Find the Calculator module and put it next in the pipeline. Change the "Result Array Name" attribute to "tempInC", you will type this new name into a text box. And type "temp -273.15" into the text box below, this changes the temperature from Kelvin to Celsius. If your data loses its colouring change the Coloring attribute to the value you typed into the results text box. Save the State.

### Questions:

Where are the hottest and coldest regions on the map?
Where are the regions that are below the freezing point of water?
Is it easier to spot the hot and cold regions?

# Exercise 3 – Contour Data

Find the Contour module and put it next in the pipeline. Set contour values to -40, -30, -20, -10, 10, 20, 30. Put another Contour module so that it takes its data out of the Calculator module, make sure it does not take data from the other Contour module and set its value to 0.0. Now change the colour of this contour to black so that it looks different to all the others. Save the State.

### Questions:

Where are the hottest and coldest regions on the map?
Where are the regions that are below the freezing point of water?
Is it easier to spot the hot and cold regions?

# Exercise 4 – Load a State and Playing with a Pipeline and Recovering from Mistakes

Before you start make sure you have saved the state from the last exercise.
The aim of this exercise to try out some of the functions that are more tricky and to try breaking our applications so that we can feel comfortable with all the buttons and know to fix problems.
In your current ParaView session open one of the states that you previously saved. You will see that if you saved the states with the same name that ParaView creates a nested set of states and you can select any that you want. Select the one that you just saved. When it is loaded check it looks right then continue otherwise go back and save the latest version. Get help if this does not work for you.
Start a new ParaView session (while still keep the old one running) and load the same state that you have in the other session.

- Notice what the contents of the Pipeline Browser looks like in each session.
- Notice that in the session that now has 2 pipelines that the “eye” has been turned off in the older version.
- Notice how the view changes as you turn on and off the visibility of the various modules.
- Try changing the input to the contour modules so that they now contour on the original data values. Notice how the image changes in the RenderView.
- Try manipulating the objects in the RenderView. Use the short cuts in the crib sheet and the Camera Controls and Center Axis Controls too. The above the RenderView offer more complex functionality so ignore them for the time being.
- Try using the Undo and Redo buttons and notice how much they remember.
- Find and press the Reset button near the bottom of the Edit menu. Notice what it does.

When you are comfortable that you can understand these controls reload the state from the last exercise and continue.

# Exercise 5 – Edit the Data Legend

Edit the legend so that the temperature values are formatted as simple decimals i.e., without the exponent e. To open the legend editor, go to the View menu and open the colour map editor and press the top rightmost button in there. Now the color map editor is open the top right button opens the settings. The formatting of numbers as text uses the C programming language formatting convention so change the "%-#6.3e" format to a this "%-#4.3f" or a similar suitable value. Both the "Label Format" and "Range Label Format" need to be altered for the legend text to be fully formatted. Adjust the other attributes in this panel until you are happy with the way the legend looks.

### Question:

Are legends used in your area of research? If so, how are they formatted and can you create that format here?

# Exercise 6 – Edit the Colormap

In the Color Map Panel there are a selection of buttons to the right of the Mapping Data panel. One of these has a red heart on it, it is the favourite's button and selecting it makes a variety of pre-set colormaps become available to select. Open this and see how the data looks with the number of these colormaps. Remember to press the Apply button so that you can see how it looks.
Compare some of the rainbow colormaps to the blot color map to the colour maps Purples, Blues and Greens. See how the colormap works with the contours. Keep that colormap.

### Question:

Which is your favourite colormap? What information elements of the data does it work well with and which elements does it not work with.

# Exercise 7 – Crop or Clip the Data 2 to 4 Times to Create a Region of Interest

Apply a clip module to the pipeline, to the output of the Calculator module. Press "Apply" to crop the data. This crops the data with a plane, in a straight line, vertical to the image is so removing the data at the left of the map.
Add another clip module to the last clip module but this time click the "Inside out" toggle so that the data it will crop the data to the right of the map.
Finally add another clip module to the last clip module. There is a grid of values under the title "Plane Parameters". The module at start-up has values in the first column of the grid but to crop the data at right angles we need to remove the values from the first column and put some values in the middle column, the middle Normal value needs to be 1.
Now you need to crop the contours, this will triple the number of clip modules you need to use.

### Questions:

Is there another way to create the pipeline so it produces the same visualization but uses less clip modules?
Is there a better pipeline that produces the same visualization that is more efficient for large data?
If you were rendering a large data version of this visualization on a remote supercomputer but rendering it on your desktop how would you design your pipeline differently to the case where you render the visualization on the supercomputer but pass an image stream to your desktop?

# Exercise 8 – Multiple Related Views

Create a second RenderVew. Put the uncropped data into the second RenderView. Use the Environment Annotation module to put useful information in the RenderView. And use the Text module in the Sources menu to create a title.
Create a title for each RenderView. Make sure your name and date goes into one of the titles.
Change the font and formatting for each title.

# Exercise 9 – Save Images

Save an image from each RenderView. Do this using the File -> Save Screenshot menu option.
Try changing the background color of the RenderView and changing the text color of the annotation and the text. The properties panel of the Text and Environment Annotate modules have the parameters you need to alter to make the colors different.
We may have access to a black and white printer, if we do try printing them both.

### Question:

What background color is best on a screen?
What background color is best in a grey scale print out?
What text color is best with a grey background a,) on the screen b,) on a grey scale print out.
What colormap is best on the screen and in a grey scale print out?

# 3D Data

ParaView comes with a small number of data sets that are used in their manuals and guides. For the animation exercises go to the File menu and select open. A file browser will open and one of the directories available there will be called data. Go into that directory and open the file.

# Exercise 1 – Make an Animation of Changing Parameter Values

Engineering 3D data:

- can.ex2

Press the reset button.
Open the can.ex2 file in ParaView. This data is a simulation of half a can being crushed by a brick.
Press Apply.
Use the Reset button in the Camera Control toolbar to put the data into the center of your RenderView if you cannot see it. (Do not select the Edit -> Reset Session as this deletes your application.).
In the Active Variable View toolbar select "Solid Color" for the variable to color the data with. This gives a 3D model with no color. We will use this to place the data in the center of the RenderView window.
Press the Rewind button in the VCR controls, this takes us to the first time step. In this time step the data is at its largest. Now use the other buttons in the Camera Control toolbar ie use the Set View Direction to +X, -X, +Y or -Y to find the best view of the data.  so that you get a good view of the data (my favourite is the one that views the data down the Y axis and looks inside the can).
Now the Active Variable View toolbar select the variable named "EQPS" rather than Solid Color. You should be looking at the first time step if you are not type 0.0 into the time text box which is in the Current Time Control toolbar.
Colouring data over a whole time set is different to colouring just one data set:

1. To do this we now want to select the Rescale to Data Range button in the Active Variable View toolbar when t is 0.0. Now press the play button to see the animation.
2. Select the Rescale to Data Range button in the Active Variable View toolbar when t is 43 (it is the end of the simulation). Now press the play button to see the animation.

### Questions:

Why are the two animations different? Which animation is more meaningful?

# Exercise 2 – Using the Animation Control Panel

Select the Animation View option from the View menu to make the panel visible. The animation can run to different times: it can run though each time step as fast as possible or it can run to real time. Try the different modes that you can set in the top left corner.

### Question:

What is the difference between the play modes?
Which play mode looks best and why?

# Exercise 3 – Saving an Animation File

Make an animation file so that you can play the animation outside of Paraiew. Animations should be annotated so use the Sources-> Annotation -> Text module to create a title at the top centre of the RenderView. Now select the Annotate Time Filter from the filter menu and the Environment Annotate module from the Filter menu.
Grab a screen shot by selecting the File-> Save Screenshot menu option. I do not save important images with any compression. Try saving an image with and without compression.
Now make an animation by selecting the File->Save Animation menu options.

### Questions:

What software do you need to play an animation file?
When is it better to have a real time animation run in ParaView and when is it best to have an animation file that you play outside ParaView?

# Exercise 4 – Contour the Data

The next few exercises use medical 3D scalar data.

- headsq.vt

When you load the data it looks like a wire frame box and it gives you little idea of what is in the data. Contouring the data gives you an isosurface at the middle data value for your data. Rotate the head so that it is looking at you.
An isosurface is a 3D analogue of an isoline which are often used on weather charts. It is a surface that represents points of a constant value (e.g. pressure, temperature, velocity, density) within a volume or space.

### Questions:

What features are in the data?
Are the isovalues below or above the middle one most interesting?

# Exercise 5 – Make an Animation of a Changing Shape, Animate the Isovalue

From the View menu tick the “Animation View” option. A new window is visible at the bottom of the scene. At the bottom of the list is a row that starts with a blue plus sign. In that row is a drop down menu for “Contour1” and another for Isosurfaces. Click on the blue cross and the click on the play video button in the VCR controls menu in the Tool Bars.

Set the number of frames to 100 to “slow” the animation down. Press return to make sure that ParaView picks up the new number of frames.

### Questions:

How many ranges of isovalues give you interesting features?

# Exercise 6 – Add Rotation to the Animation

The animation now allows you to automatically see what features there are in the scaler data but some features may occur at the back where they are not visible. Now see if you can add a rotation to the animation. The Camera option from the pull-down menu in the Animation View window will let you do this.

### Questions:

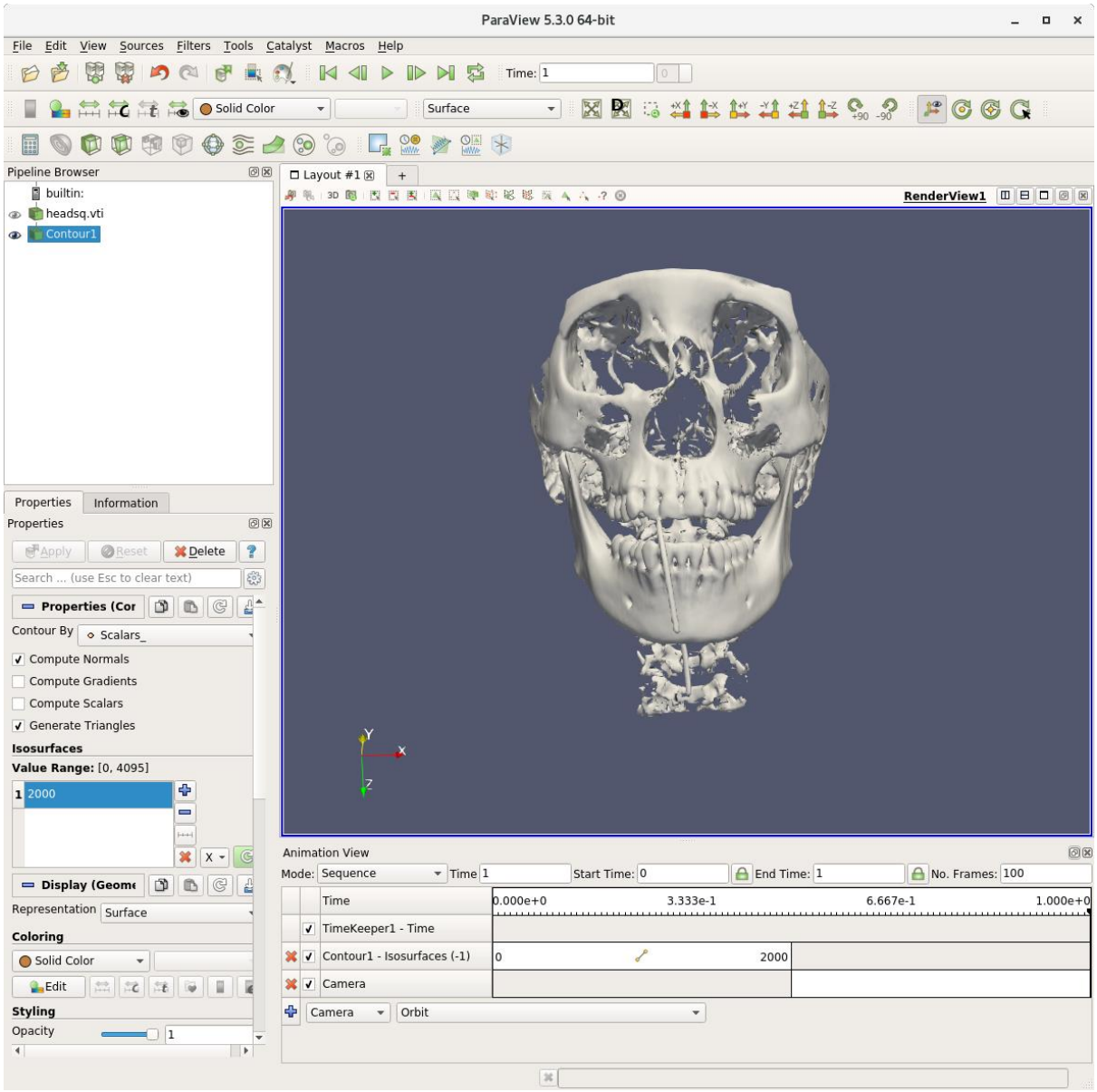
Are there any features only visible at the back of the data?

# Exercise 7 – Animate the Isovalue Before the Rotation

If you double click on any of the animation tracks a Key Frames Window will open for that track. In media production, the term key-frame is a point on a timeline which marks the beginning or end of a transition. It holds the information that defines where a transition should start or stop. The intermediate frames are interpolated over time between those definitions to create the illusion of motion. Use the Animation Keyframe window to alter the animation so for example the change in subsurface value happen from 0.0 to 0.5 seconds and the rotation occurs from 0.5 to 1.0 seconds. You will also need to set the maximum value for the isosurface so that there is something visible in the Viewer.
You are now able to automatically scan through the scalar values and through 3D space to see all the data easily. You can use the bar on the timeline to select any point in the scan.

### Question:

What other parameters might be interesting to animate in the other datasets you have used today?


ParaView 5.3.0 64-bit
File Edit View Sources Filters Tools Catalyst Macros Help
Time: 1
Solid Color
Surface
Pipeline Browser
builtin:
headsq.vti
Contour1
Layout #1
RenderView1
Properties
Information
Properties
Apply
Reset
Delete
Search ... (use Esc to clear text)
Properties (Cor
Contour By
Scalars
Compute Normals
Compute Gradients
Compute Scalars
Generate Triangles
Isosurfaces
Value Range: [0, 4095]
1 2000
Display (Geom
Representation
Surface
Coloring
Solid Color
Edit
Styling
Opacity
1
Animation View
Mode: Sequence
Time 1
Start Time: 0
End Time: 1
No. Frames: 100
Time
0.000e+0
3.333e-1
6.667e-1
1.000e+0
TimeKeeper1 - Time
Contour1 - Isosurfaces (-1)
0
2000
Camera
Camera
Orbit

# CFD – 3D Data with Scalar and Vector Components

ParaView comes with a small number of data sets that are used in their manuals and guides. For this exercise go to the File menu and select open. A file browser will open and one of the directories available there will be called data. Go into that directory and open the file:

- disk_out_ref.ex2

# Exercise 1 – Load Data and Render Scalar Data

Load the data with the Temp and V variables selected. Temp is a scalar value and V is a vector value. When you first read in the data render it with the default parameter on show. This is the vtkBlockColours. Change the render option between Outline, Points, Surface, Surface with Edges, Volume and Wireframe.

Volume refers to volume rendering which is an algorithm that creates an image similar to a medical X-Ray image. It passes radiation through a volume where some materials in the volume absorb the radiation and some let it pass through and so are invisible.

Simulations of 3D objects normally use a mesh that represents the shape of the objects or area that are being simulated.

### Questions:

What do each of these options show?
Now select the scalar value Temp and pick the same rendering options. When rendering using the Volume option try altering the transparency in the Colormap, this technique is the one where transparency has the largest effect.
Do these show the same or different information?

# Exercise 2 – Render Vector Values as a Scalar

Now select the V (velocity) parameter and render it as a Surface. Notice that ParaView sets this parameter to render as Magnitude which is a scalar value. Now try rendering it with the volume option. Now change the Magnitude to the X, Y and Z values and see how they look when surface and volume rendered.

### Questions:

What is a vector and what is the magnitude of a vector?

# Exercise 3 – Using the Calculator on Vector Data

In some simulation data the vectors are stored as separate components. The calculator module can be used to extract components or combine them. Add the calculator module to the pipeline. Using the Scalars menu create an output to that is just 1 component of the V (velocity) data. Look at this using surface and volume rendering.

Now use the scalars and the mathematical functions to create this output “v_X * iHat + v_Y * jHat + v_Z * kHat” and name the output “Results” vel. Render as a surface and as a volume and in the Active Variable Controls see how the variable values change where the Magnitude was previously. Now alter that to “mag(v_X * iHat + v_Y * jHat + v_Z * kHat)” and look again at the variable values. Now alter that to “((v_X + coordsX) * iHat) + ((v_Y + coordsY) * jHat) + ((v_Z + coordsZ) * kHat)” and tick the selection box “Coordinate Results” and render the result using the temperature value.

### Questions:

When may you want to displace the coordinates of a simulation data set?

# Exercise 4 – Glyphing Vector Data

Delete the calculator module and add a Glyph module to the pipeline. Scroll down the Properties panel to the Orientation Array and make sure that the vector property is set to V. Then scroll further to the Scale Array and alter the Scale Mode parameter to V. Render the image and adjust the Scaling value just below to a very low value, I used 0.08. Just view the glyphs.

### Question:

What is the best value for the scaling? And Why?

# Exercise 5 – Adding Context

Render the glyphs with a surface, volume, wireframe and an outline added from the input data to see how that alters the result.

### Question:

Can you change the colour of the wireframe and outline?
What is the best colour for these and why?

# Exercise 6 – Reducing Clutter

Go to the Masking panel in the Glyph module and set the Glyph mode to “Every Nth Point” and alter values for the stride parameter from 5 to 20.

### Question:

Which stride value do you think is the best and why?

# Exercise 7 – Related Variables in the Data

With the stride value set to 5 change the colouring in the Glyph module to V add the Temp data to the image using volume rendering of the input data. Alter the colormaps so that Temp and V have different colours.

### Question:

Is there a relationship between temperature and the fluid flow in the data?
How best can you show this?

# Exercise 8 – Streamlines of Vector Data

Add the StreamTracer module to the pipeline. Remove the other elements of the image so that you can see the streamlines. Change the resolution value to values between 10 and 100.

### Question:

Which other pieces of data provide useful information to help you understand the streamlines?

# Exercise 9 – Related Variables in the Data

Alter the seed type to point cloud, change the resolution to 10 and make sure “apply changes to parameters automatically” is set. Move the sphere by dragging and dropping it in the data and see how the streamlines change. Also try decreasing the number of points that seed the streamlines.

### Question:

Is there a relationship between temperature and the fluid flow in the data?
How best can you show this?
Is this better than your glyph image?

# Exercise 10 – Combining Streamlines and Glyphs

Streamlines show a path for the flow but have no direction while the glyph shows the direction of flow for points in the data so it is difficult to both. Select the StreamLine Tracer module and add the Glyph to the pipeline so that it get the output of the StreamLine as an input. In the glyph module switch the geometry to cone, reduce the number so that you can now see the direction of the streamlines and change the orientation array to V.

# Customisation with Python

This is an advanced topic and you will need to have knowledge of python or a good working knowledge of a high-level programming language such as C++ or Fortran. This does not cover plugins because 1,) a plugin is different as it is compiled with the ParaView source code (this means a plug in should be more computationally efficient than using python scripts within a ParaView session) and 2,) can have a user interface that is integrated into the ParaView interface. A plugin is more sophisticated than a python script but a python script can be the basis of a plugin.

# Exercise 1 – Make a Python Trace

We can create Python code by using the trace command while we are using ParaView. Here we will read the can.ex data set it to display a particular variable and then create a screen shot and a movie.

1. Run ParaView.
2. Tools -> Start Trace
3. Read can.ex. Turn all of the variables on. Apply.
4. +Y
5. Go forward one timestep
6. Color by EQPS
7. Save Screenshot
8. Save Animation
9. Stop trace
10. Save python code

Now look at the output images that ParaView created while you created the trace. Then open the python code in a python editor such as Geany or Spyder (or in a text editor such as gedit or notepad++ if there is not a python editor available). Look at the code.

### Questions:

Are the images the right size for a publication?

How many lines in the Python trace are there and out of that how many do you understand?

If you had to type each of these lines in how long do you think it would take?

# Exercise 2 – Edit the Trace and Run it Interactively

We can run this trace code in several ways. Firstly, we will run it through the internal python shell but first we will edit the script so that the size and name of the output files are different.

Find the 2 ‘Save’ functions in the Python code generated from the trace. Edit these functions so that the filename and resolution are different.

11. Run ParaView.
12. View -> Python Shell
13. At the bottom of Python Shell select Run Script
14. When file browser opens run the script you have just edited

Look at the output images that were created when you ran this script.

**Questions:**

The script runs much faster than when you were tracing it. When is it good to run something fast in ParaView and when is it good to run something slow?
When do you want high resolution images and when do you want low resolution images?
What other things might you want to change in your script to adjust?

# Exercise 3 – Edit the Trace and Run it From the External Command Line

- Change the names of the output files so it is easy to know the new script has been executed.
- Open a shell on Linux or a Command Line Prompt on Windows and navigate to the directory where the paraview binaries are.
- Run the command `./pvpython –usage` on Linux or `pvpython.exe –-usage` to see what command line options this binary has. The `?` flag will give you the same information.
- Run the script you created through the trace from the command line.
  - `pvpython 'c:/Users/myName/MyScript.py'`

**Questions:**

Notice that you can alter the input data using the –data=opt flg but there is no similar option for how you change the output filename and directory. How could you automate this so you do not overwrite your results 1,) but editing the file and 2,) with python code? Can you use arguments to do this?
What do you think the effect of the `–force-offscreen-rendering` flag is and why.

# Exercise 4 – Create a Graphical Primitive (a Sphere) in the Internal ParaView Python Shell

During this exercise you can get help on any function by using the help function like this `help(paraview.simple.Sphere)`.

- Start ParaView
- `View → Python Shell`
- Type `sphere = Sphere()`
- use auto-completion to see the properties `sphere. <TAB>`
- Make it visible with `Show(sphere)`
- change the radius `sphere.Radius = 1.0`
- see the value `sphere.Radius`
- see help for sphere `help(Sphere)`
- create a new sphere `sphere1 = Sphere(Radius = 10.0)`
- `Show(sphere1)`
- see this value now `sphere1.Radius`
- clip the larger sphere with `clip = Clip(sphere1)`
- `Hide(sphere1)`

- `Show(clip)`
- `Render()`
- see how the sphere can be clipped `print(clip.GetProperty("ClipType").GetAvailable())`
- set the Clip Type with `clip.ClipType = 'Plane'` or `SetProperties(ClipType='Plane')`
- Create an object to hold the display characteristics of the sphere with `sphereDisplay = GetDisplayProperties(sphere)`
- Now set the colour with (you can update the colour by rotating the objects in the RenderView with your mouse) `sphereDisplay.AmbientColor = [1.0, 1.0, 0.0]` and `sphereDisplay.DiffuseColor = [1.0, 1.0, 0.0]`
- update the pipeline with in python with `sphere.UpdatePipeline()` and `clip.UpdatePipeline()` and then `Render()`
- Get information about the clipped sphere with `clip.GetDataInformation().GetNumberOfCells()` and `clip.GetDataInformation().GetNumberOfPoints()`
- Get information about the whole sphere with `sphere.GetDataInformation().GetNumberOfCells()` and `sphere.GetDataInformation().GetNumberOfPoints()`
- Remove the clip plane from the scene `Hide3DWidgets(proxy=clip.ClipType)`

You will use this application in the next exercise so please do not delete it.

**Questions:**

How many pipelines are there in this ParaView state?

Are the names of the objects in the pipeline different when viewed through the ParaView Pipeline Browser the same as those you used when you created them?

What would you do now to change the clipped sphere to the colour blue?

# Exercise 5 - Selecting Objects in the Pipeline in the Internal ParaView Shell/Interpreter

Pipelines are made of one or more objects connected and the data flows through them from the first to the last or we could say from the data reader to the viewer. There may be multiple pipelines in any ParaView state and each pipeline and all objects for all pipelines can be seen in the Pipeline Browser. Some objects in a pipeline are visible in the viewer (the eye is highlighted beside it in the Pipeline Browser) while some objects are hidden and so not visible in the viewer (the eye is not highlighted in the Pipeline Browser).
The active object is the one in the ParaView state that can be altered i.e. have properties set or filters executed on them. Objects need to be made active so that you can alter them. When a pipeline object is created, it is set as the active object. You can also set an object as the active one. This is equivalent to clicking-on an object in the pipeline browser.
ParaView allows objects to have multiple names. The part of ParaView that keeps track of the objects and their unique names in the VTK Server Manager. Those names can be long and will not be

meaningful to your application. It is good practice to use meaningful names, the VTK Server Manager creates systematic names such as Sphere1 and Sphere2 but Oxygen1 and Nitrogen1 may be more meaningful names. You can give objects a second name as a Python variable name which is also termed a proxy name. Any object can have many Python variable names and while it possible to find the VTK Server Manager's name through the Python variable name it is not possible to go the other way. This system affects how you can identify objects and so delete them.

- You can see all the objects in the ParaView state by typing `GetSources()` which returns a dictionary of (name, id) object pairs. This is necessary because multiple objects can have the same name and the (name, id) pair identifies objects uniquely. NB the results of the `GetSources()` function are not the Python variable names we used in the last section when we created the objects.
- You can check whether the name of an object is the one created by the VTK Server Manager or not by typing `print(FindSource("Sphere1"))` or `print(FindSource("sphere"))`
- You can also create a new name to reference an object managed by the VTK Server Manager by typing `myClip = FindSource("Clip1")`. You cannot go the other way and get the Python variable name from the names stored in the VTK Server Manager.
- You can see which is the active object by typing `GetActiveSource()`
- Now change this by typing `SetActiveSource(clip)` look at the pipeline in the Pipeline Browser to see which object is highlighted. Now type `SetActiveSource(sphere)` and look again at the pipeline in the Pipeline Browser, a different object is highlighted. Finally, type `SetActiveSource(myClip)` and look again at the Pipeline Browser.
- You can create object that only have the name given to them by the VTK Server Manager, type `Line()`.
- You can see this object has been created and given a name by typing `GetSources()`
- But you cannot use that name to make that object active. Try typing `SetActiveSource(Line1)` instead you have give it a name `myLine = FindSource('Line1')` and then you can use it `SetActiveSource(myLine)`.
- You can delete an object with the Python variable name using the command `Delete(myLine)`.
- If the object does not have a Python variable you will need to create one to be able to delete it (remember you can use `GetSources()` to find all the objects). Type `Cone()` to create an object with no Python variable name; then `GetSources()` finds its name from the VTK Server Manager; `myCone = FindSource("Cone1")` gives the object a Python variable name; and finally `Delete(myCone)` removes it. (Alternatively you could use the garbage collector but that is not covered here.)

**Questions:**

How would you could you keep track of the of VTK Server Manager created names and your Python variable names in a Python code if you had a very large number of objects?

# Exercise 6 – Properties of Pipeline Objects Internal ParaView Shell

The VTK Server Manager controls the pipelines and the objects in them. This is implemented in C++ but we are using Python. If you have a Python variable name for an object some properties are

accessible through that name. Where we do not have a Python variable name or we want to change a property that is not linked to that name we need to use Python to pass information to the VTK Server Manager so that it can update the objects in the pipeline.

- Create a line object with the Python variable name `myLine`, `myLine = Line()`.
- Create a sphere object with the Python variable name `mySp`, `mySp = Sphere()`.
- Look at all the properties and functions that are available for `myLine` and `mySp` using the auto-completion option in the Python interpreter.
- Look at the object properties relate to that type of object (geometrical primitive) using the `ListProperties` function eg `myLine.ListProperties()`. See how these properties have names that relate to have the geometric primitive is defined.
- You can give these object property values when you create them. Type `myLine2 = Line(Point1=[1, 1, 1], Point2=[1.5, 1.5, 1.5], Resolution=4)` and now make both lines visible in the viewer. Now Type `mySp2 = Sphere(Center=[11,11,11], Radius=5)` and make both spheres visible.
- You can change the properties once objects have been created in different ways. Type `SetProperties(mySp, PhiResolution=100, ThetaResolution=100)` and then type `GetProperty(mySp, "Radius")` to confirm what the values are.
- Alternatively you can create a proxy for a property that you want to change. Type `radiusMySp = mySp.GetProperty("Radius")` to create the proxy. Check the value in the proxy by typing `print(radiusMySp)`. Alter the value of the proxy with `radiusMySp.setElement(2)`. Confirm the proxy value has changed with `print(radiusMySp)` and then with `GetPropery(mySp, "Radius")`.
- You can change other properties that do not relate to the geometry with other functions. For example the display properties relate the the OpenGL and scene properties of a view.
  - To get access to the display properties type `display = GetDisplayProperties(myLine)` (or `display = GetDisplayProperties()` if the object you are interested in is the active object) and the type `display.ListProperties()` to see all the properties. Type `SetDisplayProperties(myLine2, LineWidth=50)` and then make the lines visible and hide the spheres (you may also need to interact with the scene to make the changes appear). For the alternative method type `display=getDisplayProperties(myLine2)` and `display.ListProperties()` and change the `LineWidth` property with `display.SetPropertyWithName("LineWidth", 20)` and `display.LineWidth` to confirm it.
  - To get access to the view properties type `v1 = GetViewProperties()`, to see a list of all the properties type `v1.ListProperties()` and to alter one type `v1.SetPropertyWithName("UseGradientBackground", True)`, you may need to interact with the scene to see the effects. Now you can create a light `light1 = AddLight(view=v1)` and then alter its properties to change the colour of the light used in the view with `light1.DiffuseColor = [0.0, 1.0, 1.0]`or `light1.DiffuseColor = [1.0, 1.0, 0.0]`. This looks better on the sphere rather than the line.

**Questions:**

What are the different types of properties that an object like a cone has compared to its display properties and the view's properties?

# Exercise 7 – Editing demo1()

The paraview.simple python package that has some demo functions. Make sure you get the right ones as there are also demos in paraview.servermanager which is not useful for this course. The demo1() is integrated into the ParaView executable which we would do if we wanted to create a ParaView plugin but that is too advanced for this course. The ParaView packages and any plugin have a Proxy interface to the VTK class but we cannot do that in a free standing ParaView script. Instead if we create a script from the demos in the paraview.simple package we need to create proxies explicitly.

Lets create a python function that is stored in a file external to ParaView and not part of the ParaView executable.

- Start up a fresh ParaView session.
- Create a text file and in it create a simple function that prints hello

```
def my_print_demo1():
      """This simple demonstration prints in the ParaView Python
shell"""
      print("Hello")

def my_print_demo2():
      """This simple demonstration prints in the ParaView Python
shell and
      then and returns a string which is also printed. """
      print("Hello")
      return("hi")

def my_print_demo3(name="Joanna"):
      """This simple demonstration prints in the ParaView Python
shell and
      then and returns a string which is also printed. """
      print("Hello ", name)
      return("hi")
```

- Save this in a file called myFunctions.py
- In the python shell type

```
sys.path.append('/<filepath>/')
import myFunctions
myFunctions.my_print_demo1() # you can auto-complete on this
myFunctions.my_print_demo2()
myFunctions.my_print_demo3()
myFunctions.my_print_demo3("Jack")
```

Lets run the demo1() that is in paraview.simple

- Now turn on the python tracer.
- In the internal Python interpreter type `demo1()`.
- Look at the results in the viewer and then turn off their visibility.

Lets make a copy of the code to run as a script.

- Go to the documentation for the simple module, information is given on the crib sheet. Find the source code for the demo1() function.
  - Copy and paste this source code for this function except the line `def demo1()` into a text editor and save it to a file called `myScript.py`. Make sure that all the text you copied is not indented and then execute it through the Python shell by pressing the "Run Script" button.
- Now try editing this source code to find out more about how python works in ParaView, for example. Look at the last exercise or use the tracer to work out the syntax.
  - Change the cone into 3D text and alter the text in that to say "hello"
  - Add a green light to the view

Lets make demo1 into our own module

- Go to the documentation for the simple module, information is given on the crib sheet. Find the source code for the demo1() function.
  - Copy and paste this source code for this function but this time include the line `def demo1():` and edit this so it now reads as `def my_edited_demo1():` into a text editor and save it to a file called `myModule.py`. Make sure that all the text you copied is indented.
  - Add the line `from paraview.simple import *` to the top of the file.
  - Now execute `my_edited_demo1()` by typing.

```
sys.path.append('/<filepath>/')
import myModule
myModule.my_edited_demo1() # you can auto-complete on this
```

- Now try editing this source code to find out more about how python works in ParaView, for example. Look at the last exercise or use the tracer to work out the syntax.
  - Change the cone into 3D text and alter the text in that to say "hello"
  - Add a green light to the view

Lets make demo1 into our own module with a main function

**Questions:**

What vtk data type is the sphere and the cone?

What is the vtk data type of the can.ex2 data we used earlier and how is it different to that of the sphere?

# Exercise 8 – Editing demo2("/path/*filename*")

The paraview.simple python package that has 2 demo modules. In this demo the displayed model has a mesh (a geometry) as well as data attached to the mesh. ParaView automatically checks the data to see which reader is best. This demo is designed to demonstrate the way to use the ExodusIIReader which is the best for the disk_out_ref.ex2 data

- Start a fresh ParaView session.
- In the internal Python interpreter type `demo2(fname="/path/disk_out_ref.ex2")` where you alter "`path`" to the address of the directory where the data is stored.
- Type commands into the Python Shell to create and display scalar glyphs:

```
GetSources()
myData = FindSource('ExodusIIReader1')
glyph1 = Glyph(Input=myData, GlyphType='Sphere')
glyph1.OrientationArray = ['POINTS', 'No orientation array']
glyph1.ScaleArray = ['POINTS', 'Temp']
glyph1.ScaleFactor = 0.001
glyph1.GlyphTransform = 'Transform2'
Show()
```

- Now update the glyph so that it works for the vector 'V'

```
glyph1.GlyphType = 'Arrow'
glyph1.OrientationArray = ['POINTS', 'V']
glyph1.ScaleFactor = 0.005
UpdatePipeline()
Show()
```

- Go to the documentation for the simple module, information is given on the crib sheet. Find the source code for the demo2() function and read through it to see how each line has created the output to the shell and the view.
- Copy and paste this source code for this function into a text editor and alter it to display a glyph.

**Questions:**

What vtk data type is being glyphed here?

Can other vtk data types be glyphed?

# Exercise 9 - Using Programmable Filter to read in and write out data.

ParaView has a filter module that is designed so that you can add your own code to read or filter the data. These are called the Programmable Source and Programmable Filter and there are some examples of the code you can run through it in the ParaView documentation in particular the ParaView Guide, listed in the crib sheet.

- From the Filters menu search for the Programmable Source or the Programmable Filter filter.
- Copy the csv code from the documentation making sure to edit the paths and file names and work on to debug the code.