

Chapter 1

Visualising oomph-lib's output files with Paraview

All of oomph-lib's existing elements implement the `GeneralisedElement::output(...)` functions, allowing the computed solution to be documented via a simple call to the `Mesh::output(...)` function, e.g.

```
// Open output file
ofstream output_file("output.dat");

// Call the mesh's output function which loops over the
// element and calls theirs...
Problem::mesh_pt() ->output(output_file);
```

By default, the output is written in a format that is suitable for displaying the data with `tecplot`, a powerful and easy-to-use commercial plotting package – possibly a somewhat odd choice for an open-source library.

We also provide the capability to output data in a format that is suitable for display with `paraview`, an open-source 3D plotting package. For elements for which the relevant output functions are implemented (they are defined as broken virtual functions in the `FiniteElement` base class) output files for all the elements in a certain mesh (here the one pointed to by `Bulk_mesh_pt`) can be written as

```
// Output solution to file in paraview format
some_file.open(filename);
Bulk_mesh_pt->output_paraview(some_file,npts);
some_file.close();
```

where the unsigned integer `npts` controls the number of plot points per element (just as in the tecplot-based output functions). If `npts` is set to 2, the solution is output at the elements' vertices. For larger values of `npts` the solution is sampled at greater number of (equally spaced) plot points within the element – this makes sense for higher-order elements, i.e. elements in which the finite-element solution is not interpolated linearly between the vertex nodes. It is important to note that when displaying such a solution in paraview's "Surface with Edges" mode, the "mesh" that is displayed does not represent the actual finite element mesh but is a finer auxiliary mesh that is created merely to establish the connectivity between the plot points.

Paraview makes it possible to animate sequences of plots from time-dependent simulations. To correctly animate results from temporally adaptive simulations (where the timestep varies) paraview can operate on `pvf` files which provide a list of filenames and the associated time. These can be written automatically from within oomph-lib, using the functions in the `ParaviewHelper` namespace:

```
/*****paraview_helper*****
/// Namespace for paraview-style output helper functions
*****namespace ParaviewHelper
{
    /// Write the pvf file header
    extern void write_pv_header(std::ofstream& pvf_file);

    /// Add name of output file and associated continuous time
    /// to pvf file.
    extern void write_pv_information(std::ofstream& pvf_file,
                                    const std::string& output_filename,
                                    const double& time);
```

```

/// Write the pvd file footer
extern void write_pvd_footer(std::ofstream& pvd_file);

} // namespace ParaviewHelper

```

Once the pvd file is opened, call `ParaviewHelper::write_pvd_header(...)` to write the header information required by paraview; then add the name of each output file and the value of the associated value of the continuous time, using `ParaviewHelper::write_pvd_information(...)`. When the simulation is complete write the footer information using `ParaviewHelper::write_pvd_footer(...)`, then close to the pvd file.

Currently, the paraview output functions are only implemented for a relatively small number of elements but it is straightforward to implement them for others.

The [FAQ](#) contain an entry that discusses how to display oomph-lib's output with `gnuplot` and [how to adjust oomph-lib's output functions to different formats](#).

Angelo Simone has written a python script that converts oomph-lib's output to the vtu format that can be read by `paraview`. This has since been improved and extended significantly with input from Alexandre Raczynski and Jeremy van Chu. The conversion script can currently deal with output from meshes that are composed of 2D triangles and quad and 3D brick and tet elements.

The oomph-lib distribution contains three scripts:

- `scripts/oomph-convert.py` : The python conversion script itself.
 - `scripts/oomph-convert` : A shell script wrapper that allows the processing of multiple files.
 - `scripts/makePvd`: A shell script the creates the * .pvf files required by paraview to produce animations.
-

1.1 The oomph-convert.py script for single files

1.1.1 An example session

1. Add oomph-lib's `scripts` directory to your path.
2. Here is what's in the current directory at the moment: `curved_pipe.dat` is the oomph-lib output produced from [a simulation of steady flow through a curved pipe](#).

```

biowulf: 11:05:10$ ll
total 824
-rw-r--r--    1 mheil    users        2292 May 21 09:19 curved_pipe.dat

```

3. Run `oomph-convert.py`

```

biowulf: 11:16:18$ oomph-convert curved_pipe.dat
-- Processing curved_pipe.dat
oomph-convert.py, ver. 20110531
Convert from oomph-lib Tecplot format to VTK XML format.
Dimension of the problem: 3
Plot cells defined

```

```

Field variables = 4
Conversion started
Coordinate defined
Connectivities defined
Offset defined
Element types defined
Field data defined
Conversion done
Output file name: curved_pipe.vtu

```

4. We now have the corresponding *.vtu file

```

biowulf: 11:32:08$ ll
total 1024
-rw-r--r--    1 mheil    users      329874 Jun 21 09:19 curved_pipe.dat
-rw-r--r--    1 mheil    users     705294 Jun 21 11:16 curved_pipe.vtu

```

5. ...which we can visualise with paraview:

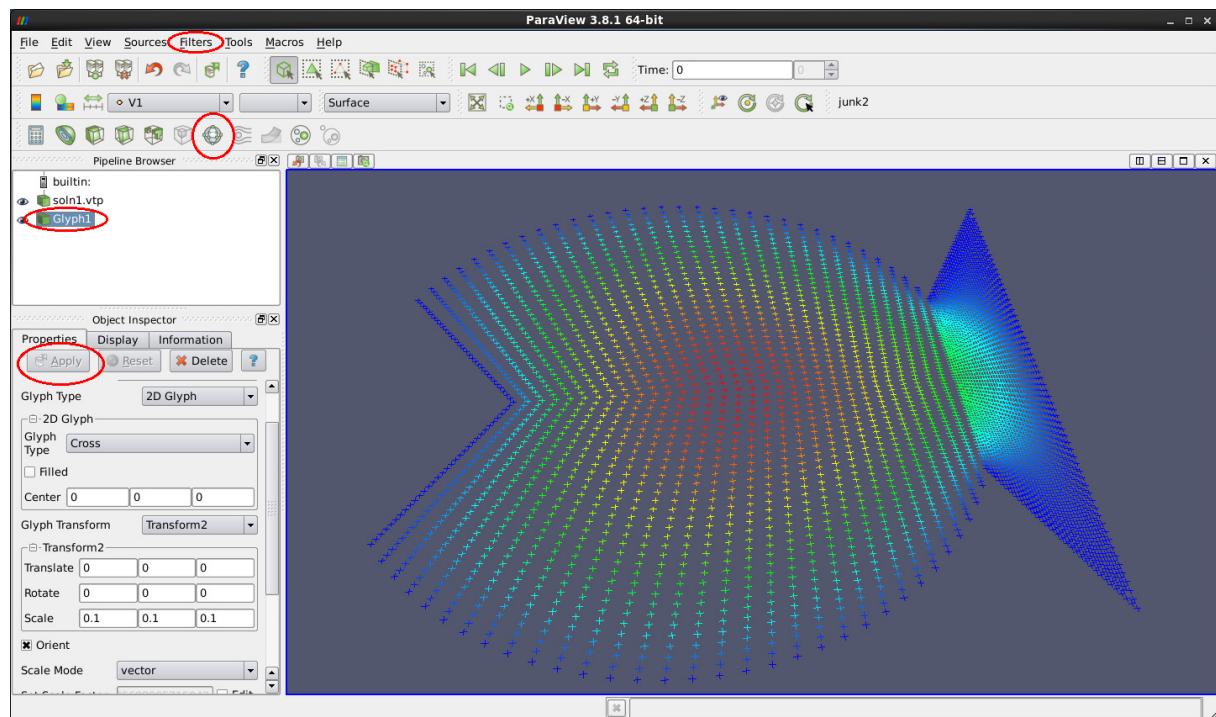
```
biowulf: 11:34:08$ paraview --data=curved_pipe.vtu
```

If your output file is invalid or contains elements that cannot currently be converted, you can use the `-p` option (followed by 2 or 3 to indicate the spatial dimension of the problem) to extract points only:
`biowulf: 11:16:13$ oomph-convert.py -p2 soln0.dat`

The output is now a .vtp data file (Visualization Toolkit Polygonal) which is also supported by Paraview. To display your .vtp data, use the



Glyph filter (displaying the points as crosses, say). Here is a representative plot in which `the adaptive solution of a 2D Poisson equation in a fish-shaped domain` is displayed with points.

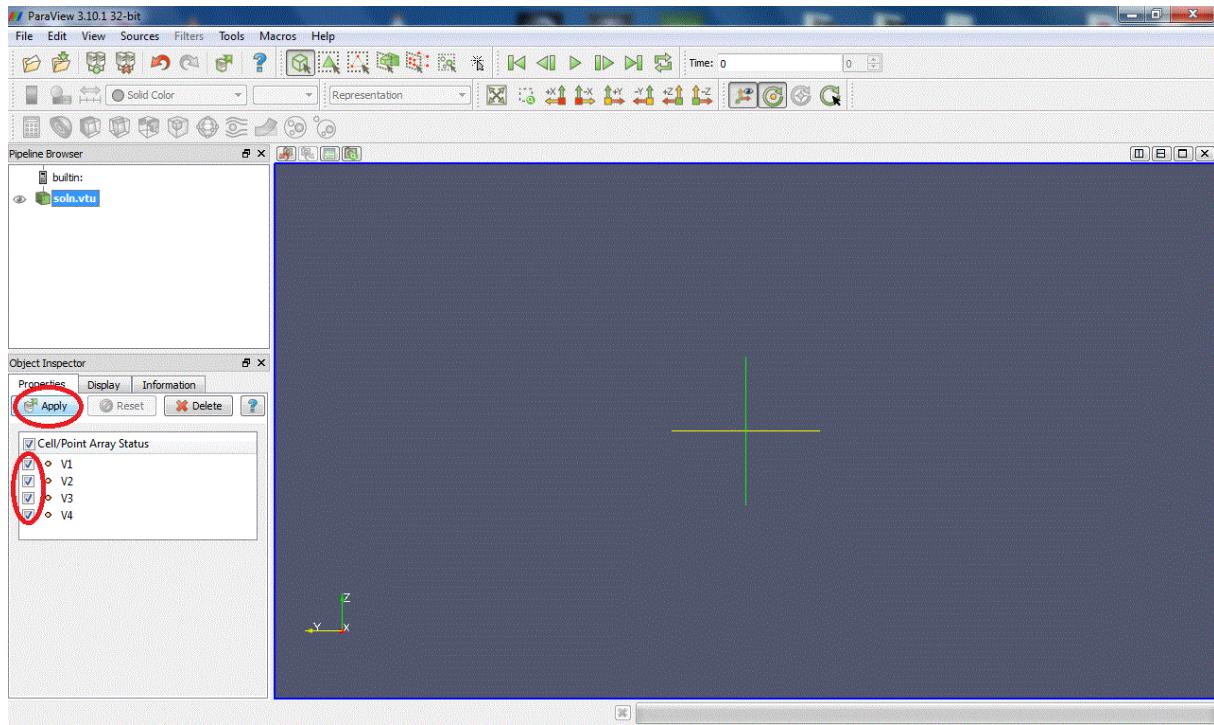


1.1.2 Display your data

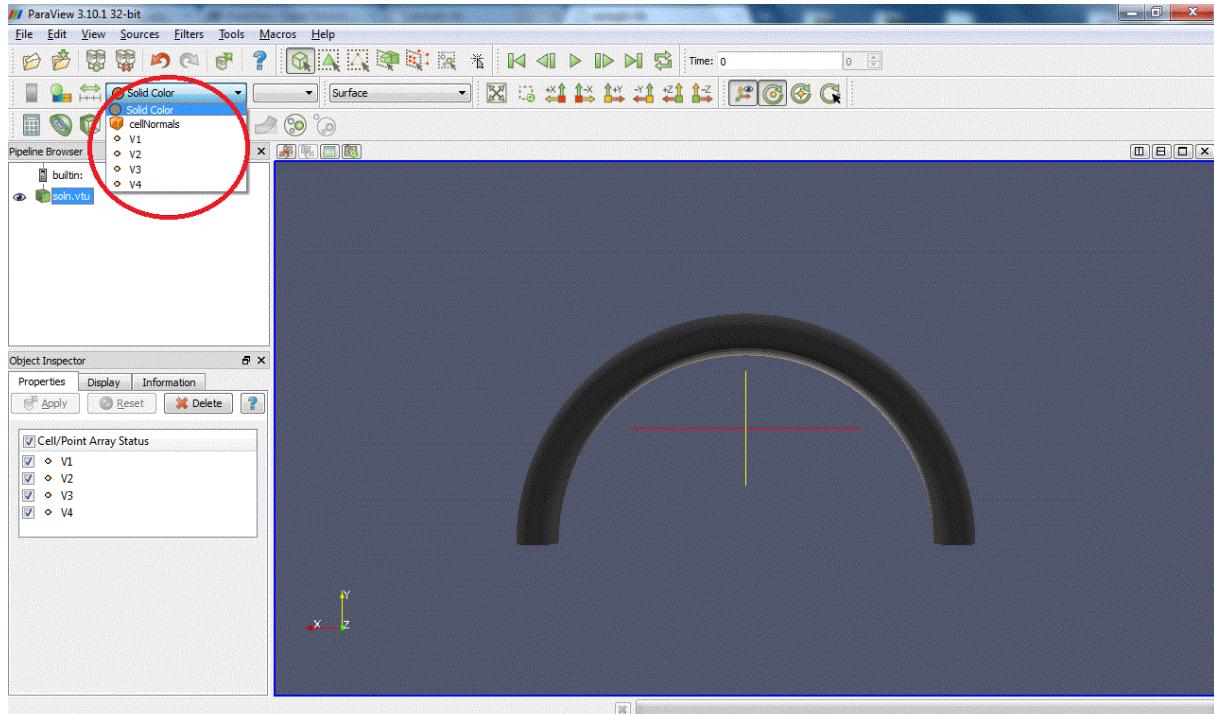
Here are a few screenshots from a paraview session to get you started. When paraview starts up, you have to select the arrays of values you want to load and click on



Apply:

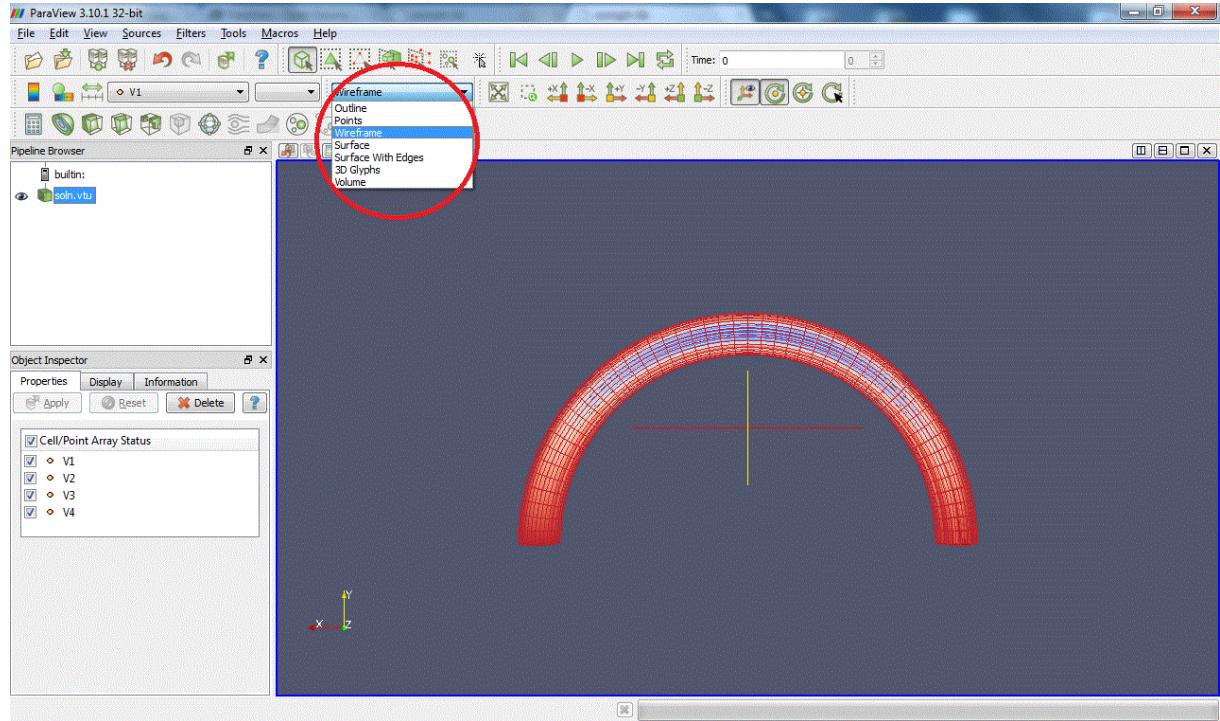


Select the array of values you want to display in the active window (V1, V2, V3...). You can only display one at a time. It is applied on the data set selected in the pipeline:



Now choose the plot style of your data. **Outline** display a box containing the data but not the data itself (it's not entirely clear to us why you would want to do this, but...). **Points** and **Wireframe** best suited for 3D

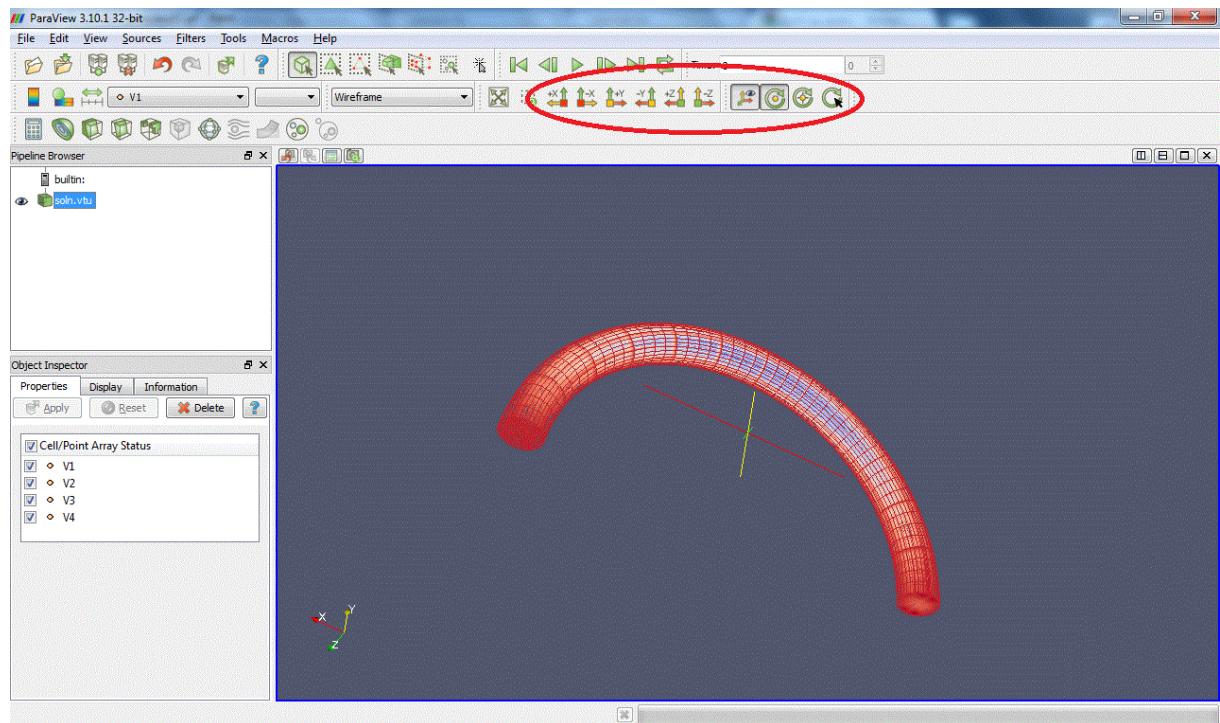
computations because they allow you to "see through" the data set. Surface and Surface With Edges is best suited for 2D computations because only the surface is displayed. Here is a view of the data in Wireframe mode:



You can move the figure with buttons



in the toolbar or simply with the mouse: "Left click + Move" to rotate, "Middle click + Move" to move, zoom in with scroll up or "Right click + Move up" and zoom out with scroll down or "Right click + Move down".



You can also display the colour legend by clicking on



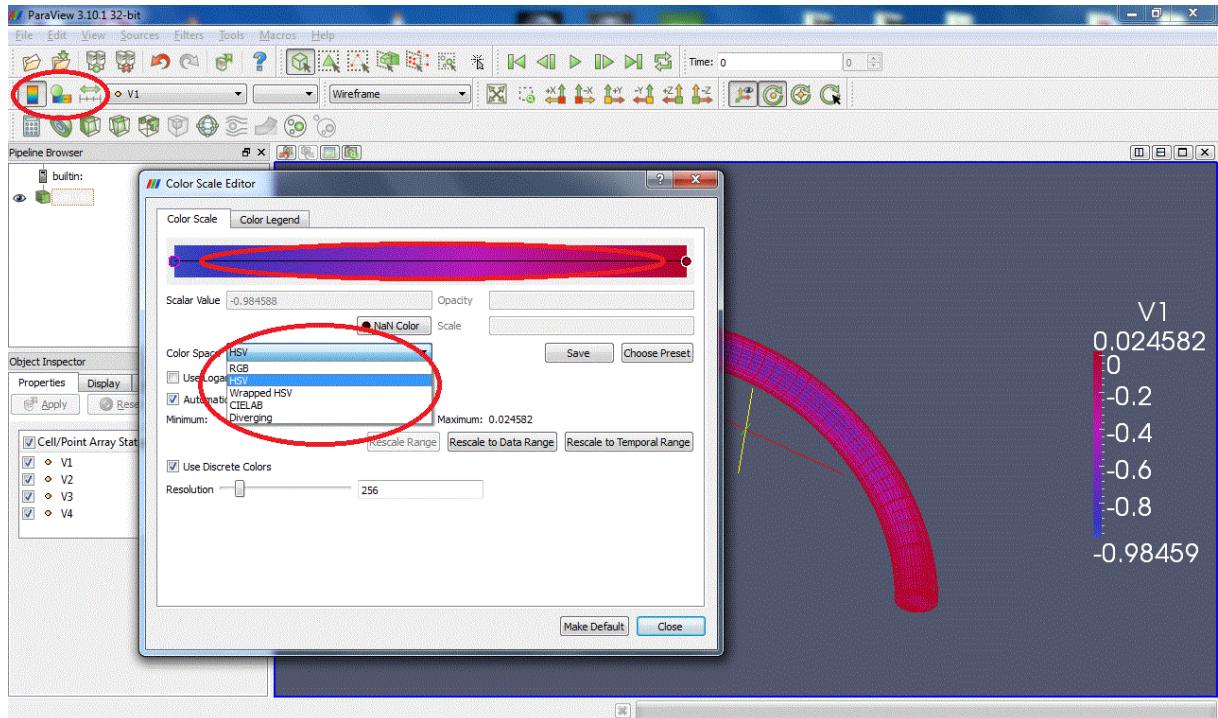
, change the display colours (HSV, RGB, Diverging...) by clicking on



, and rescale the colour scale to data by clicking on



. Modifying the colour scale is easy: add points by clicking on the colour bar to change the distribution of colours or use a logarithmic scale.



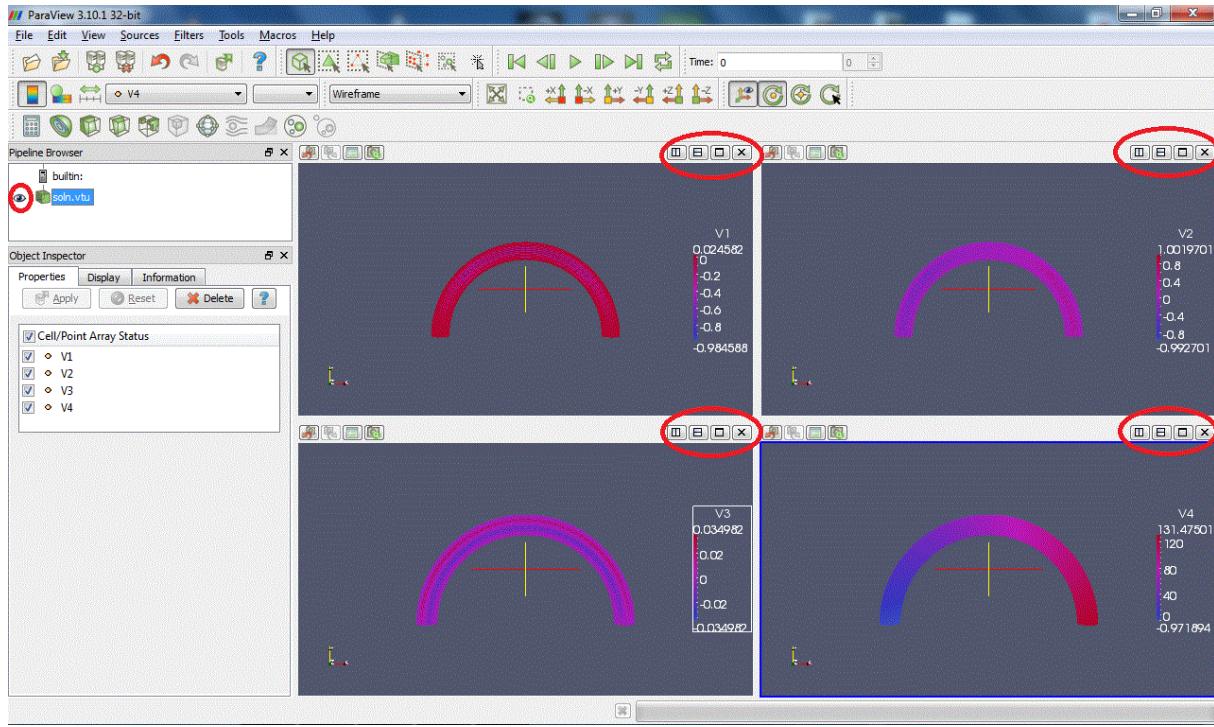
You can split a window by clicking on



. Clicking on the plot window will make it active. You can switch on/off the display of a pipeline member by clicking on the eye



on the left. You can display different values and states in different windows:



1.2 The oomph-convert and makePvd scripts for multiple files and animations

1.2.1 An example session for data from a serial computation

Here is a quick demonstration of `oomph-convert` and `makePvd` scripts in action

1. Add `oomph-lib`'s scripts directory to your path
2. Here is what's in the current directory at the moment: `soln?.dat` are the `oomph-lib` output files that illustrate the progress of the mesh adaptation during [the adaptive solution of a Poisson equation in a fish-shaped domain](#).

```
biowulf: 11:05:10$ ll
total 824
-rw-r--r--  1 mheil    users        2292 May 21 09:19 soln0.dat
-rw-r--r--  1 mheil    users     176776 May 21 09:19 soln1.dat
-rw-r--r--  1 mheil    users     278117 May 21 09:19 soln2.dat
-rw-r--r--  1 mheil    users    367408 May 21 09:19 soln3.dat
```

3. Run `oomph-convert` on all files (the `-z` option adds zeroes to the numbers – this is only required if the files are to be combined into an animation by paraview)

```
biowulf: 11:16:13$ oomph-convert -z soln*.dat

-- Processing soln0.dat
  oomph-convert.py, ver. 20110615
Parse input file for Tecplot zones.....done
  0 lines ignored
Write nodal coordinates.....done
Write cell connectivity.....done
Write cell offsets.....done
Write cell types.....done
Write field 01/01.....done
```

```

Conversion done in 0 seconds
Output file name: soln00000.vtu

-- Processing soln1.dat
oomph-convert.py, ver. 20110615
Parse input file for Tecplot zones.....done
 0 lines ignored
Write nodal coordinates.....done
Write cell connectivity.....done
Write cell offsets.....done
Write cell types.....done
Write field 01/01.....done
  Conversion done in 0 seconds
  Output file name: soln00001.vtu

[Further output suppressed]

```

4. We now have the corresponding * .vtu files

```

biowulf: 12:37:05$ ll
total 2568
-rw-r--r-- 1 mheil    users      5979 Jun 21 12:37 soln00000.vtu
-rw-r--r-- 1 mheil    users    377490 Jun 21 12:37 soln00001.vtu
-rw-r--r-- 1 mheil    users   592990 Jun 21 12:37 soln00002.vtu
-rw-r--r-- 1 mheil    users  789325 Jun 21 12:37 soln00003.vtu
-rw-r--r-- 1 mheil    users     2292 Jun 21 09:19 soln0.dat
-rw-r--r-- 1 mheil    users   176776 Jun 21 09:19 soln1.dat
-rw-r--r-- 1 mheil    users   278117 Jun 21 09:19 soln2.dat
-rw-r--r-- 1 mheil    users  367408 Jun 21 09:19 soln3.dat

```

These *.vtu files can be displayed individually as discussed above.

5. To produce an animation of the results with paraview, create a *.pvda file using makePvd

```

biowulf: 12:40:56$ makePvd soln mysoln.pvda
--> File mysoln.pvda created

```

6. ...and visualise it:

```

biowulf: 12:42:08$ paraview --data=mysoln.pvda

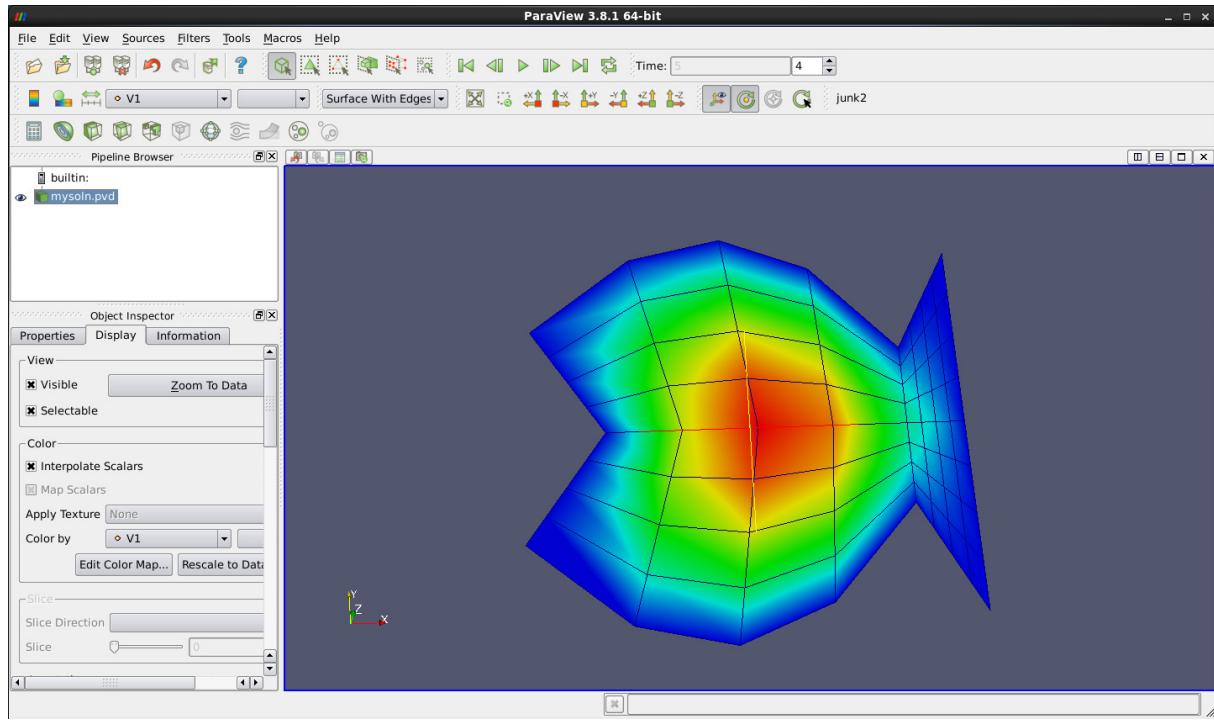
```

1.2.2 Screenshots from the paraview session

Here's a screenshot from the paraview session: once the *.pvda file is loaded you can customise the plot style as discussed in the previous example, and then use the



Play/Stop/... buttons to animate the progress of the mesh adaptation.



1.2.3 An example session for data from a parallel computation

oomph-lib typically outputs results from parallel (distributed) computations on a processor-by-processor basis, resulting in filenames of the form

```

soln_proc0_0.dat      \
soln_proc1_0.dat      |
:                     | Data for timestep 0
soln_proc[NPROC-1]_0.dat /
```



```

soln_proc0_1.dat      \
soln_proc1_1.dat      |
:                     | Data for timestep 1
soln_proc[NPROC-1]_1.dat /
```

where NPROC is the number of processors. An animation of such data obviously requires the output from different processors (but for the same timestep) to be combined. Provided, the filenames have the pattern [stem]proc[processor_number]_[timestep_number].dat

(note the "proc" and "_", both of which are required), the `.pvf` file can be generated by first processing the files with oomph-convert,

`oomph-convert -z [stem]proc*`

followed by

`makePvd [NPROC] [stem] myplot.pvd`

So, for the files listed above, to produce a `.pvf` file that contains data from a computation with four processors the commands

`biowulf: 12:40:56$ oomph-convert soln_proc*`

followed by

`biowulf: 12:40:59$ makePvd 4 soln_ soln.pvf`

would create the file `soln.pvf` from which paraview can create an animation of the solution.

1.3 Data analysis with filters

In order to analyse the data, we can apply filters. Some filters are accessible directly via the navigation bar; a full list is available in the `Filters` menu:

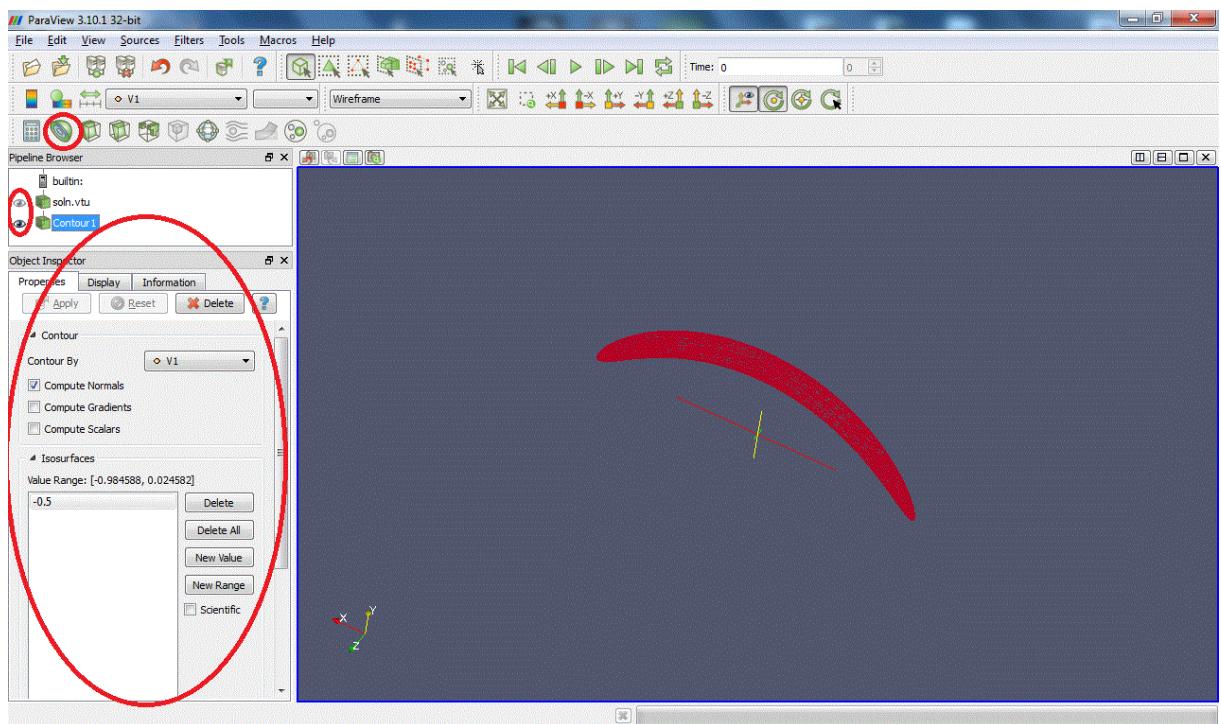


Here are few examples of filters available:

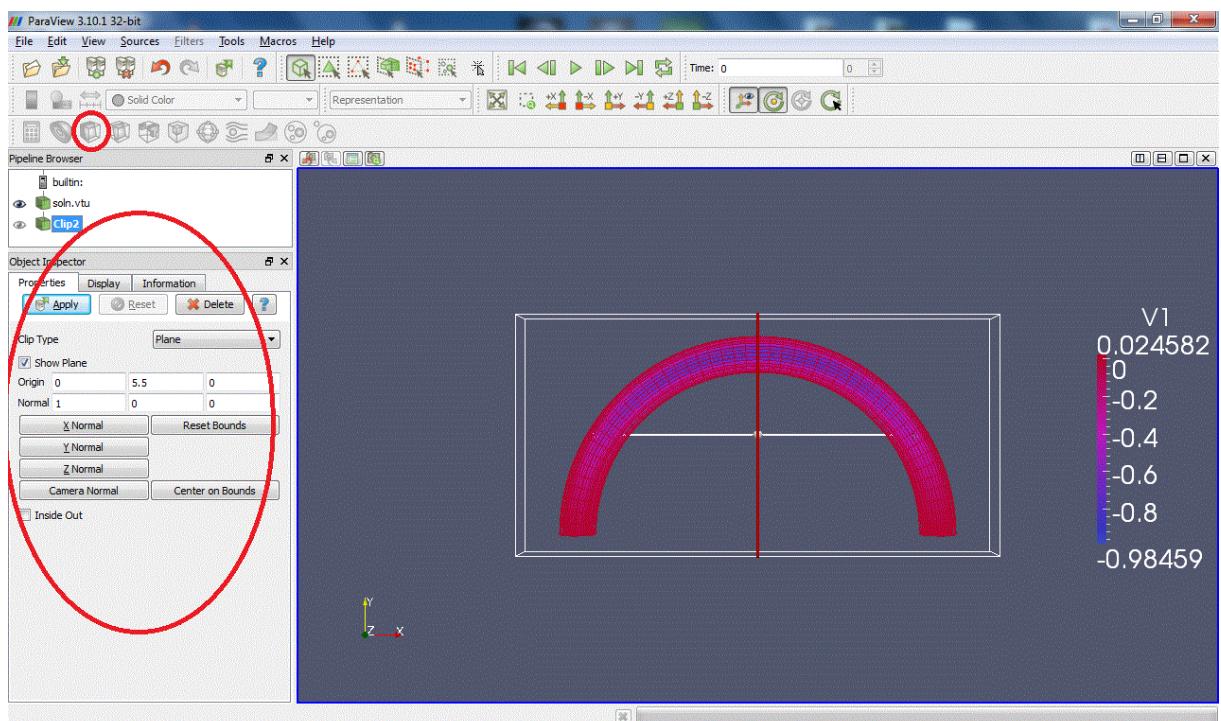
1. **Calculator:** Evaluates a user-defined expression e.g $\frac{1}{2}V1^2$ and creates a new data array, called here "Energy", containing the result of this expression:



2. **Contour:** Extracts the points, curves or surfaces where a scalar field is equal to a user-defined value e.g $V1 = -0.5$:



3. Clip: Intersects the geometry with a half space. (Warning: with some versions of Paraview, zooming on the clipped surface can cause the X server to crash.)





4. Slice: Intersects the geometry with a plane. (Warning: with some versions of Paraview, zooming on the clipped surface can cause the X server to crash.)





5. Threshold: Extracts cells that lie within a specified range of values



1.4 How to ...

1.4.1 Select and extract elements

Click on the button:



Select Cells On to select elements on the surface (2D selection)



Select Points On to select points on the surface (2D selection)



Select Cells Through to select elements through the region selected (3D selection)

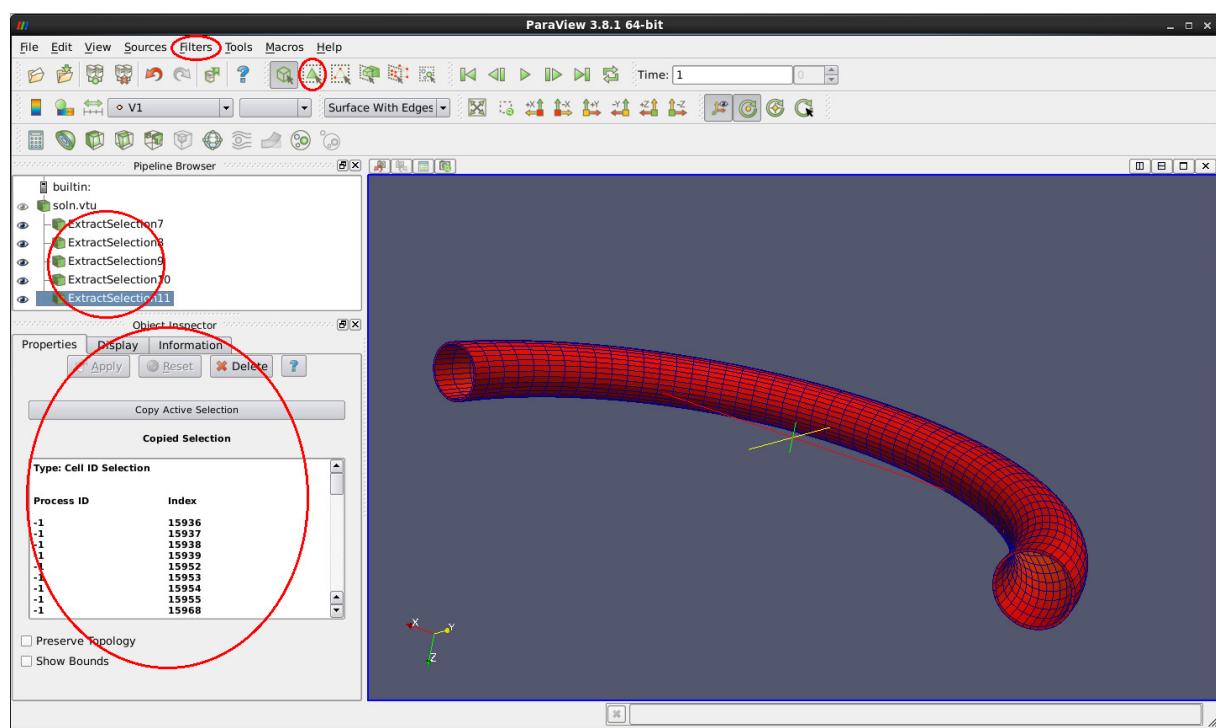


Select Points Through to select points through the region selected (3D selection)

When your selection is highlighted, go to Filters->Data Analysis->Extract Selection and



Apply the filter to extract the selected elements. You can now modify or apply filters on the extracted data only. Here is an example of extraction of the surface elements of the curved pipe data:



1.5 PDF file

A [pdf version](#) of this document is available. \