Plotting in Gnuplot and VisIt

Stephen Millmore

Laboratory for Scientific Computing, University of Cambridge

May 13, 2019

Outline

What should you use to make a plot?

Whatever is easiest!

- Gnuplot - very good for 1D, good for simple 2D set ups, not recommended for 3D
- VisIt - not recommended for 1D, very good for 2D and 3D, especially multiple materials and AMR
- These two will be covered, but other options exist:
- Inbuilt plotting tools, e.g. those in Matlab or Mathematica (or Excel!)
- Paraview (coming soon?)
- Python, e.g. pyplot
- By hand?

Outline

Advantages of gnuplot

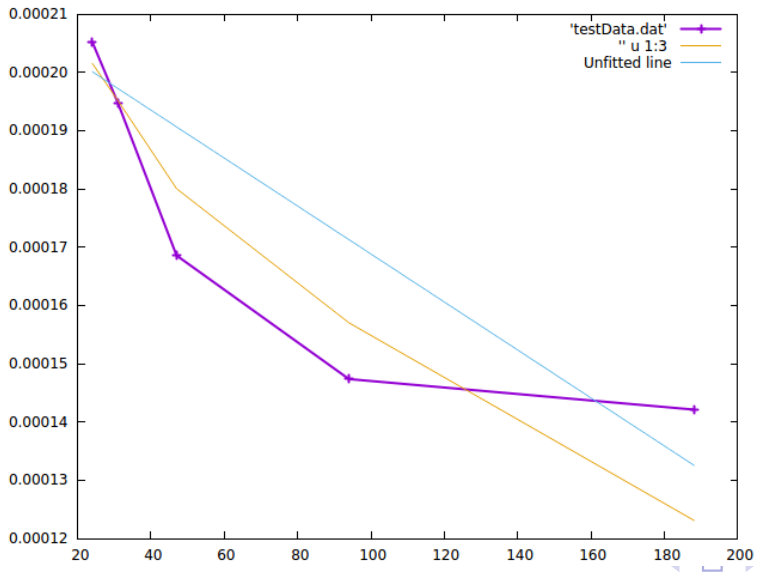
- It's free
- Plot and axis appearance are easily customisable
- 1D plots are very clear and easy to manipulate
- 2D plots work well, either as a surface rendering or a colour map image - harder to use if you don't want to plot the entire domain
- In-built data analysing tools (e.g. maxima and minima)
- Can either plot through command line, or from a file

Gnuplot via terminal

- `> gnuplot`
- Loads up gnuplot within a terminal, ready for commands
- Typing `help` will bring up an overall description of gnuplot, and a list of further topics for which help is available
- In general, gnuplot help is well written with useful examples

Plotting commands and shortcuts

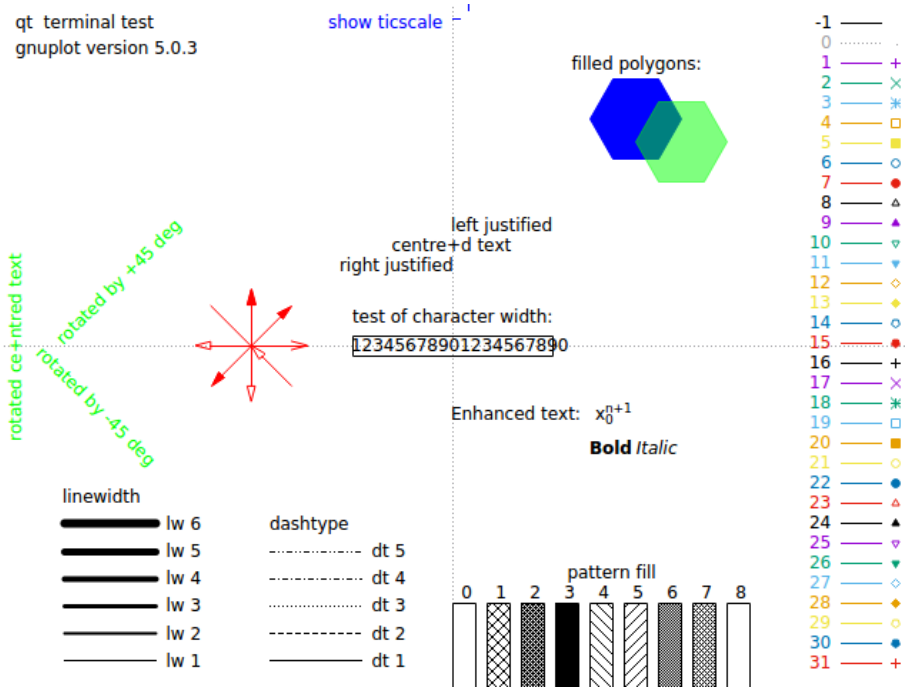
- There tend to be two forms of each gnuplot command, long (verbose) and short (shorthand)
- ```
  gnuplot> plot 'testData.dat' with linespoints linewidth 2,
   'testData.dat' using 1:3 with lines linetype 4, f(x) title 'line'
  ```
- ```
  gnuplot> p 'testData.dat' w lp lw 2, '' u 1:3 w l lt 4, f(x) t 'line'
  ```
- The two examples above do exactly the same thing



Basic commands

```
gnuplot> p 'testData.dat' w lp lw 2, '' u 1:3 w l lt 4, f(x) t 'line'
```

- `plot` / `p` - the actual plot command. Other commands come later
- `with` / `w` - keyword specifying how you want the plot rendered
- `linespoints` or `lines` or `points` etc./ `lp` or `l` or `p` etc. - types of style available
- `linetype` or `linewidth` etc. `lt` or `lw` etc. - control the chosen style
- Basic information about these (and other style capabilities) can be found by simply typing
  `gnuplot> test`



Ordering of commands

- In general, commands must be in the correct order or an error is given
- The general format is:

```
plot <range> <file or function> <what is plotted> <style commands>
```

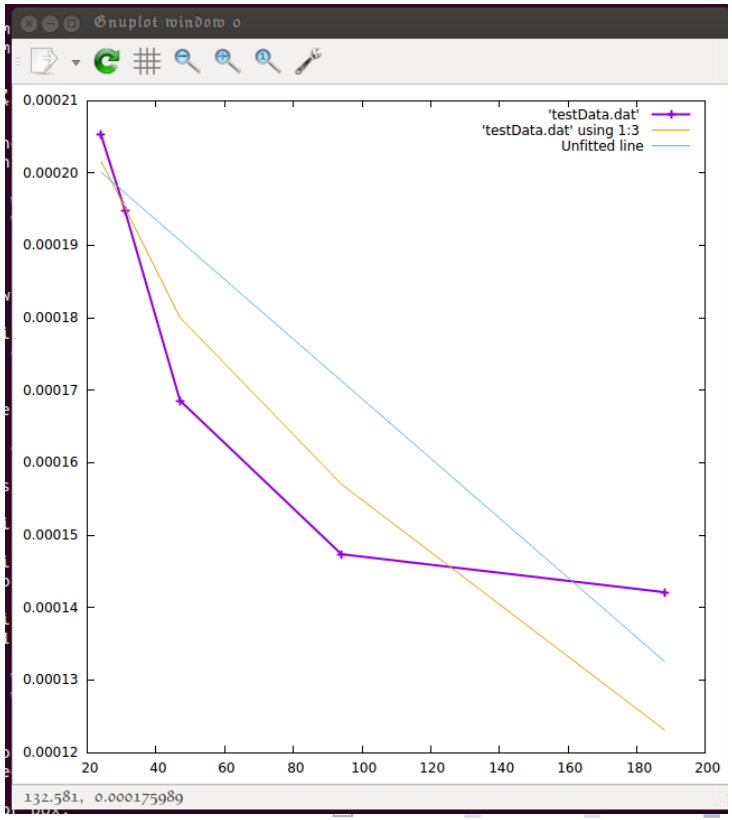
- For example:

```
gnuplot> p [1:50] 'testData.dat' every 2 u 1:3 lw 2 t 'penguin' w lp
```

Using the plot window

- This assumes you've loaded gnuplot with the default 'qt' terminal - other operating systems (e.g. Macs) might differ
- The terminal window is interactive, usually used just for zooming in and out, but may wish to do other things

- Zoom: +/- buttons at the top *or* +/- keys ('=' acts as '+' too) *or* right click and drag a region you want to zoom in on
- Reset view to default: magnifying glass button with '1', *or* 'a'
- Toggle grid: grid button *or* 'g'
- Other interactive options include replot (if plotfile is still being output) and window save options



Other plot window commands

- Not always useful, but sometimes hit by accident, and can be undone by pressing the same button!
- '1'/'2' - alter format of coordinate information in the bottom left of the window
- '6' - echoes button commands to terminal screen (as gnuplot commands)
- '7' - alter size ratio of the screen
- 'q' - quit window
- 'r' - toggle ruler (a cross-hair that appears at the cursor location)
- 'h' - help window for these commands
- 'l' - toggle logarithmic y-axis
- 'L' - toggle logarithmic <axis closest to the cursor>
- 'm' - toggle mouse interactivity
- arrow keys - move plot window in direction of arrow

Other commands than plot

- `gnuplot> help <topic>`
  `gnuplot> h <topic>`
  Brings up the help pages for a specific topic (cannot necessarily use shortcut for the topic name)
- `gnuplot> splot`
  `gnuplot> sp`
  surface plot (2D plot command)
- `gnuplot> replot`
  `gnuplot> rep`
  replot last plot or splot command
- `gnuplot> load <file>`
  `gnuplot> l <file>`
  load a file (should contain gnuplot commands)
- `gnuplot> print <arg>`
  `gnuplot> pr <arg>`
  print output of argument (e.g. function evaluation)
- `gnuplot> set <arg>`
  `gnuplot> se <arg>`
  set environment variable
- `gnuplot> unset <arg>`
  `gnuplot> unse <arg>`
  stop an environment variable from being shown
- `gnuplot> <var> = <value>`
  define a variable, or a function

Constants

- Defining constants is straightforward, and gnuplot has basic maths commands

```
gnuplot> pi = 4*atan(1)
gnuplot> pr pi
3.14159265358979
```

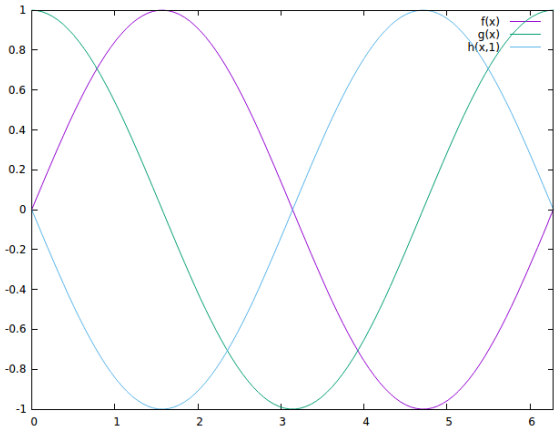
- Note - pi is defined already, but can be overwritten
- Take care when defining constants through integers

```
gnuplot> ratio = pi/3
gnuplot> pr ratio
1.0471975511966
gnuplot> ratio = 7/3
gnuplot> pr ratio
2
gnuplot> ratio = 7./3.
gnuplot> pr ratio
2.33333333333333
```

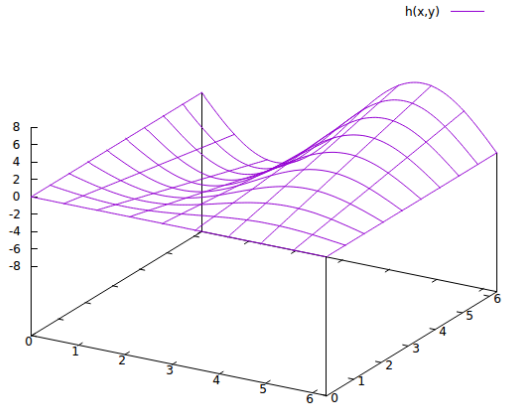
Functions

- Functions are defined intuitively, and can be functions of multiple variables
- When plotting, however, the argument must be ‘x’ in 1D, ’x,y’ in 2D

```
gnuplot> f(x) = sin(x)
gnuplot> g(penguin) = cos(penguin)
gnuplot> h(x,t) = -t*sin(x)
gnuplot> p [0:2*pi] f(x), g(x), h(x,1)
gnuplot> pr f(pi)
1.22464679914735e-16
```




```
gnuplot> h(x,t) = -t*sin(x)
gnuplot> sp [0:2*pi][0:2*pi] h(x,y)
```



Plotting from files

- Plotting from multiple files is straightforward

```
gnuplot> p 'testData.dat' w lp, '' u 1:3 w lp, 'testData2.dat' w lp
```

- The shortcut '' specifies the same file as the previous command is to be used
- Files that gnuplot can read are moderately flexible, this is 'testData.dat'

```
# Scan speed (mm/s), Simulation data, measured data
24, 0.0002052617        0.00020158
31, 0.0001947424        0.00019516
I shouldn't be typing here
47, 0.000168558         0.00018004
94, 0.0001473605        0.00015702
188, 0.0001421181       0.00012312
```

- The `u` or `using` command selects which columns of data are plotted - not specifying is equivalent to `u 1:2` the first column does not have to be the x value, e.g. `u 3:1`
- Columns can be separated by spaces, tabs or commas (at least), '#' comments a line in a manner that gnuplot may be able to recognise, garbage lines are ignored entirely

Indexed file format

- It is possible to have multiple data quantities within a single file, e.g. output from multiple time steps

```
# Scan speed (mm/s), Simulation data, measured data
24, 0.0002052617        0.00020158
31, 0.0001947424        0.00019516
47, 0.000168558         0.00018004
94, 0.0001473605        0.00015702
188, 0.0001421181       0.00012312


# Scan speed (mm/s), Simulation data, measured data
31      0.0001948399    0.0002405
41      0.0001868327    0.0002281
62      0.0001574733    0.0002080
124     0.0001521352    0.0001780
248     0.0001467972    0.0001400
```

- In order for gnuplot to distinguish these, each separate entry must be separated by *at least* two lines of blank space
- The `index` or `i` option specifies which will be used (starting with 0)

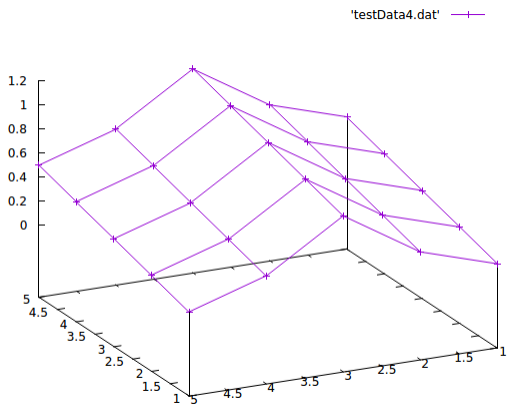
```
gnuplot> p 'testData3.dat' u 1:2 i 1 w lp, '' u 1:3 index 0 w lp
```

2D data format

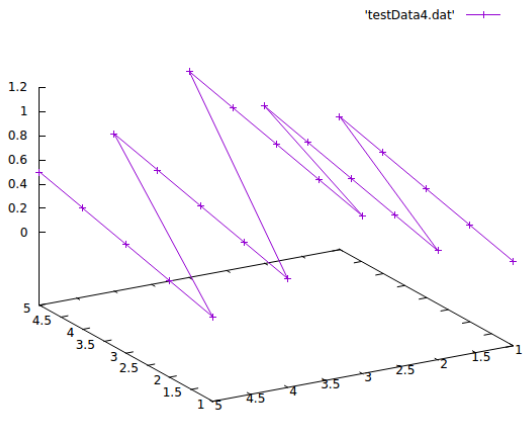
- 2D data should (ideally) be set out as
  `<x coords> <y coords> <data at x,y>`
- For rectangular grids, the format shown should be used (or inverse for x and y)
- After each y-sweep, a blank line must be left to allow for a grid to be plotted

```
1 1 0.1
1 2 0.2
1 3 0.3
1 4 0.4
1 5 0.5

2 1 0.3
2 2 0.4
2 3 0.5
2 4 0.6
2 5 0.7

3 1 0.7
3 2 0.8
```

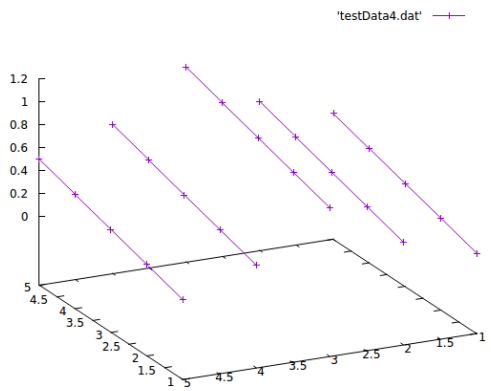
1 Line



No Lines



2 Lines



Plot ranges

```
plot <range> <file or function> <what is plotted> <style commands>
```

- One way to control the range of the plot is to specify it after the `plot` or `splot` command

```
gnuplot> p [0:2*pi] f(x), g(x), h(x,1)
gnuplot> p [0:2*pi][0:1] f(x), g(x), h(x,1)
gnuplot> p [][0:1] f(x), g(x), h(x,1)
gnuplot> sp [1:3][1:3][0:1] 'testData4.dat' w lp
```

- A single entry will always adjust the x-axis range, but all axes can be controlled (or ignored)
- This will control the ranges of *all* subsequent plots, attempting to specify different ranges later will result in a gnuplot warning

Controlling what is plotted

`plot <range> <file or function> <what is plotted> <style commands>`

- These are ways to use the data from a file, beyond plotting just the raw numbers
- Here we consider `index`, `using` and `every`
- `index` has already been described - there is little more to it
- `using` or `u` picks columns, but we can also operate on these columns

```
gnuplot> p 'testData.dat' u (1e-3*$1):2 w lp
gnuplot> p 'testData.dat' u (1e-3*$1):($3-$2) w lp
```

- `every` or `ev` selects points to plot, see help for full 2D file options

```
gnuplot> p 'testData.dat' u 1:2 ev 2 w lp
gnuplot> p 'testData.dat' u 1:2 ev 2::2 w lp
```

Ternary operator

- One additional note for the `using` command - the ternary '?:' operator is supported

```
p 'testData.dat' u 1:($1 > 50 ? sin($2) : 1/0) w lp
```

- Additionally, attempting to plot $1/0$ is rendered empty by gnuplot, e.g. this can be used to plot data for a specific sign of a level set function

Style commands

`plot <range> <file or function> <what is plotted> <style commands>`

- We consider three different style commands, `with` (and associated styles), `axis` and `title`
- `with` or `w` controls the look of the data plotted
- `axis` or `ax` controls the axis data is plotted on - for 1D plots, there are four possibilities, `x1`, `x2`, `y1`, `y2`, - useful for showing values of two very different variables

```
gnuplot> p 'testData.dat' u 1:2 axis x1y1 w lp, 'testData2.dat' u 1:2 axis x2y2 w lp
gnuplot> p 'testData.dat' u 1:2 axis x1y1 w lp, 'testData2.dat' u 1:2 ax x2y1 w lp
```

- `title` or `t` sets the title of the data within the legend

```
gnuplot> p 'testData.dat' u 1:2 w lp title 'Good data'
gnuplot> p 'testData.dat' u 1:2 w lp t 'Good data'
gnuplot> p 'testData.dat' u 1:2 w lp notitle
```

- `notitle` means the plot will not appear in the legend

Style commands - `with`

- `gnuplot> help with` gives a list of potential styles

```
Syntax:
      with <style> { {linestyle | ls <line_style>}
                   | {{linetype  | lt <line_type>}
                      {linewidth | lw <line_width>}
                      {linecolor | lc <colorspec>}
                      {pointtype | pt <point_type>}
                      {pointsize | ps <point_size>}
                      {fill | fs <fillstyle>}
                      {nohidden3d} {nocontours} {nosurface}
                      {palette}}
                  }


where <style> is one of

     lines        dots       steps     errorbars     xerrorbar    xyerrorlines
     points       impulses   fsteps    errorlines    xerrorlines  yerrorbars
     linespoints  labels     histeps   financebars   xyerrorbars  yerrorlines
     surface      vectors    parallelaxes
Press return for more:
or
     boxes            boxplot        ellipses       image
     boxerrorbars     candlesticks   filledcurves   rgbimage
     boxxyerrorbars   circles        histograms     rgbalpha   pm3d
or
     table
```

- Not all are applicable to all data (e.g. error bars needs error data as columns)
- Refer to `gnuplot> test` to see the styles and colours
- Note - `linetype` also alters `pointtype` by default, and in 1D, `linestyle` and `linetype` do the same thing
- Also `linewidth` controls a points thickness, `point size` its actual size

set

- `gnuplot> set <variable>` allows various aspects of the plot to be altered
- `gnuplot> help set` reveals the quantity of options available (we shall not detail them all!)

```
Subtopics available for set:
    angles              arrow               autoscale           bars
    bmargin             border              boxwidth            cbdata
    cbdtics             cblabel             cbmtics             cbrange
    cbtics              clabel              clip                cntrlabel
    cntrparam           color               colorbox            colorsequence
    contour             dashtype            data                datafile
    date_specifiers     decimalsign         dgrid3d             dummy
    encoding            fit                 fontpath            format
    function            grid                hidden3d            history
Press return for more:
    historysize         isosamples          key                 label
    linetype            link                lmargin             loadpath
    locale              log                 logscale            macros
    mapping             margin              margins             missing
    monochrome          mouse               multiplot           mx2tics
    mxtics              my2tics             mytics              mztics
    object              offsets             origin              output
    palette             parametric          paxis               pm3d
    pointintervalbox    pointsize           polar               print
    psdir               raxis               rmargin             rrange
    rtics               samples             size                style
    surface             table               term                terminal
    termoption          tics                ticscale            ticslevel
    time_specifiers     timefmt             timestamp           title
    tmargin             trange              urange              view
    vrange              x2data              x2dtics             x2label
    x2mtics             x2range             x2tics              x2zeroaxis
    xdata               xdtics              xlabel              xmtics
    xrange              xtics               xyplane             xzeroaxis
    y2data              y2dtics             y2label             y2mtics
    y2range             y2tics              y2zeroaxis          ydata
    ydtics              ylabel              ymtics              yrange
Press return for more:
    ytics               yzeroaxis           zdata               zdtics
    zero                zeroaxis            zlabel              zmtics
    zrange              ztics               zzeroaxis
```

Axis manipulation

- The range of data plotted can be adjusted through e.g. `set xrange` or `set xr`

```
gnuplot> set xr [0:100]
gnuplot> set yr [] reverse
```

- By default, axes are not labelled, labels are set through e.g. `set xlabel` or `set xl`

```
gnuplot> set xlabel 'Height ({/Symbol m}m)'
gnuplot> set ylabel 'Pressure (atm)'
gnuplot> set y2label 'Density (kg/m^3)'
```

- Control of the numbered tics on the axis is through e.g. `set xtics`

```
gnuplot> set y2tics
gnuplot> set ytics nomirror
gnuplot> set xtics (24, 31, 47, 94, 188)
```

Additional features on the plot

- It can be useful to give a plot a title, this is done through `set title`

```
gnuplot> set title 'Plot of a Penguin'
```

- It is also possible to draw arrows, or lines, on a plot, through `set arrow` - this can be useful for marking the location of a feature, e.g. a discontinuity

```
gnuplot> set arrow from 0.0,0.5 to 100,0.5
gnuplot> set arrow from 50,0.5 to 100,0 nohead
```

2D plotting

- There are three typical ways to plot 2D data through `splot`:

1) The default, height-mapped grid
2) Contours
3) Colour map

- For the second two options, it may be desirable to remove the grid entirely, and ensure a top-down view
- This is achieved through

```
gnuplot> set view map
gnuplot> unset surface
```

Contours

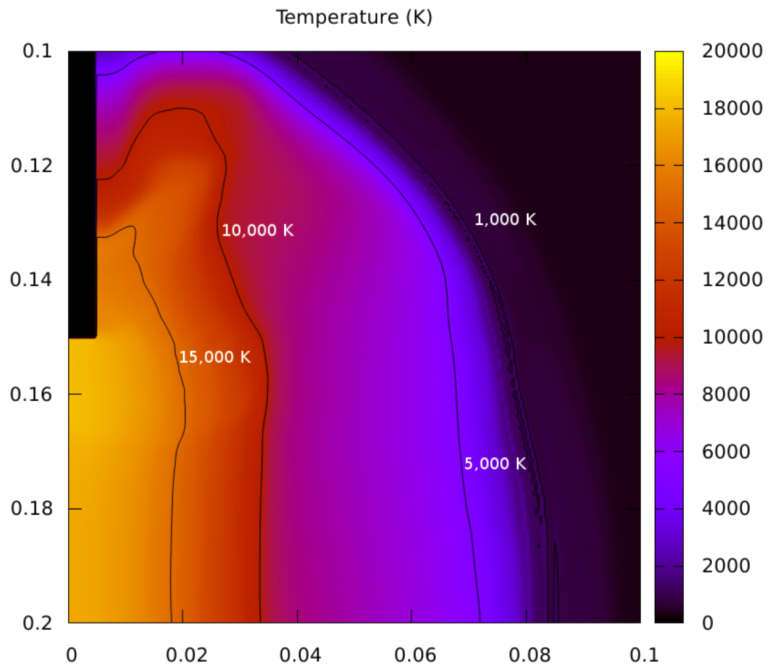
- Contours are turned on through `gnuplot> set contour` or `gnuplot> set cont`
- By default, contours are placed at the base of the three dimensional box containing the surface grid
- `gnuplot> set contour surface` or `gnuplot> set contour both` can change this
- Default contour levels are chosen by gnuplot, control over these levels is through `gnuplot> set cntrparam <options>`

```
gnuplot> set cntrparam levels discrete 0,1,2,3,4
gnuplot> set cntrparam levels incremental 0,1,4
```

- The contour style (line width etc.) is controlled through the plot style - it is not straightforward to independently control the colour of the contours

Palette-mapped 3d (pm3d)

- `gnuplot> set pm3d` allows a surface to be plotted as a 2D colour mapped image
- By default, the image will be plotted on the bottom 'surface' of the three dimensional box containing the surface grid
- This can be changed, through e.g. `set pm3d at top`, `set pm3d at s` - options are `bottom`, `top` and `surface` or `b`, `t`, `s`



Altering the pm3d style

- The colour range is set through `gnuplot> set cbrange [<low>:<high>]` - this is independent of the `zrange` setting
- The colour scheme is changed through `gnuplot> set palette <options>`
- Options here are numerous, recommended examples are at: `http://gnuplot.sourceforge.net/demo_5.2/pm3dcolors.html`
- The location of the colour box is altered through `gnuplot> set colorbox <options>`

Output to file

- Outputting a plot to a file requires two things; setting the output name (obviously) and setting the correct terminal for the output
- To set an output, simply use e.g.
  `gnuplot> set output 'Filename.png'`
- In order to generate the file in the desired format, the terminal type needs to be changed through `gnuplot> set terminal <options>` or
  `gnuplot> set term <options>`

```
Subtopics available for set term:
    cairolatex          canvas              cgm                 context
    corel               dumb                dxf                 eepic
    emf                 emtex               epscairo            epslatex
    fig                 gif                 hpgl                jpeg
    latex               lua                 mf                  mp
    pcl5                pdfcairo            png                 pngcairo
    pop                 postscript          pslatex             pstex
    pstricks            push                qms                 qt
    size                svg                 tek40xx             tek410x
    texdraw             tgif                tikz                tkcanvas
    tpic                vttek               wxt                 x11
Press return for more:
    xlib                xterm
```

Terminal options

- Some of these terminal options will open a new window when selected, e.g. ‘qt’ or ‘x11’, others will generate a file, e.g. ‘pngcairo’ or ‘postscript’
- Selecting the terminal also allows for size and font choices for the overall plot to be made

```
gnuplot> set term postscript enhanced color
gnuplot> set terminal pngcairo size 700,500 enhanced font 'Verdana,20'
gnuplot> set term postscript "Helvetica" 12 enhanced color portrait
```

- The `enhanced` mode is what allows for text formatting, e.g. superscripts
- Practical note - when outputting to files, the first plot will create the file, whilst for a second plot, gnuplot will attempt to append this to the file (which may not make sense)
- This can be avoided if the `set output` command is used again (even to the same filename)

Multiple plots in one window

- Gnuplot allows this through `gnuplot> set multiplot`, this is likely to be used with output to a file, rather than the default terminal
- Once multiplot mode is on, the size and origin of each plot must be chosen, e.g.

```
gnuplot> set size 0.4 0.2
```

```
gnuplot> set origin 0.1 0.1
```

- The bottom-left corner is (0,0) and the top-right corner is (1,1) always (regardless of the dimensions of the window)
- Some care may be needed if the width of the axis numbers do not match - the size defined may be the entire box containing the plot

Miscellaneous set options

- Log scaling of the axes can be done through `gnuplot> set logscale <axes>`
- The log scaling can be specified, if something other than base 10 (default) is required
- When plotting functions, it may be that they are not well resolved (e.g. a highly oscillatory function), this is due to gnuplot not sampling the function frequently enough
- If necessary, `gnuplot> set samples <number>` will force gnuplot to take a specified number of samples
- For 2D surfaces, `gnuplot> set isosamples <number>` achieves this (note - a lot of sampling will lead to long plot times)

Scripting

- All commands shown so far can be entered through the command line, though they can also be entered through a gnuplot script
- These are useful for saving plot commands, either to avoid forgetting, or to use upon restarting or on new files
- Scripts can either a complete process (loading, setting output file and closing), or partial process (e.g. setting constants and environment variables)
- A complete process can be run through:
  `> gnuplot PlotFile.gp`
- Whist a partial process is loaded within a gnuplot instance:
  ```
  gnuplot> load PlotCommands.gp
  ```

Script example 1 - how to split lines

```
set term postscript enhanced color
set output '100_z24_sqrtlaw.eps'

set format x "%3.1e"
set format y "%3.1e"

set xtics 6e-4,2e-4,2e-3

f(x) = 0.102 * x + 1.1e-05
g(x) = 0.0262 * x - 2.34e-5

set key outside right

################ Plot 1 ####################

set xlabel 'sqrt(time)'
set ylabel 'radius'

p 'crossingRadius_24e-6.dat'  u (sqrt($1)):3 w lp t '1600, 0Levels', \
  'crossingRadius_24e-6.dat'  u (sqrt($1)):3 w lp t '400, 1Levels', \
  'crossingRadius_24e-6.dat'  u (sqrt($1)):3 w lp t '800, 0Levels', \
  'crossingRadius_24e-6.dat'  u (sqrt($1)):3 w lp t '800, 1Levels', \
  'crossingRadius_24e-6.dat'  u (sqrt($1)):3 w lp t '400, 0Levels', \
  f(x) w l lt 0 lw 6 t '0.102x'

################ Plot 2 ####################

set xlabel 'time^{0.38}'
set xtics 2e-3,10e-4,10e-3

set output '100_z24_38law.eps'

p 'crossingRadius_24e-6.dat'  u (($1)**(0.38)):3 w lp t '1600, 0Levels', \
  'crossingRadius_24e-6.dat'  u (($1)**(0.38)):3 w lp t '400, 1Levels', \
  'crossingRadius_24e-6.dat'  u (($1)**(0.38)):3 w lp t '800, 0Levels', \
  'crossingRadius_24e-6.dat'  u (($1)**(0.38)):3 w lp t '800, 1Levels', \
  'crossingRadius_24e-6.dat'  u (($1)**(0.38)):3 w lp t '400, 0Levels', \
  g(x) w l lt 0 lw 6 t '0.0262x'
```

Script example 2 - do loops

```
set terminal pngcairo size 1400,260 enhanced font 'Verdana,14'

set pm3d
unset surface
set view map
set size ratio -1
set cbrange [0:5e6]
unset key

set origin -0.04, -0.01

set xrange [0:0.4]
set yrange [0:0.05]
set xtics out nomirror
set ytics out nomirror
unset x2tics
unset y2tics

set output 'DynamicVillaPressure_0.png'
sp 'DynamicVillaARP_2D_test_2D_0.dat' u 1:2:7 w lp

do for [t=1:97] {
   x = 20.*cos(2*pi*2e6*t*1e-7)
   set arrow from 0.0,0.5 to x,0.5
   set output 'DynamicVillaPressure_'.t.'.png'
   sp 'DynamicVillaARP_2D_test_2D_'.t.'.dat' u 1:2:7 w lp

}
```

Outline

Advantages and disadvantages of VisIt

Advantages

- It's free
- It is created by a group with a multiphysics AMR code, and as a result (LLNL), has many very useful features for our multiphysics AMR code
- 2D and 3D plots are handled well, and there is a lot of flexibility as to what is plotted

Disadvantages

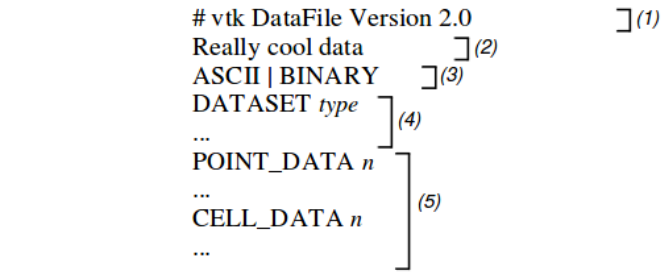
- It is not the most stable software
- Can be more work to compile and generate output for

Data files for VisIt

- Unlike gnuplot, VisIt cannot read tabulated files - it expects more information about what is in the file
- However, the list of other file formats that can be used is huge! `https://www.visitusers.org/index.php?title=Detailed_list_of_file_formats_VisIt_supports`
- General output is beyond the scope of this lecture
- Large codes (group code, AMReX) should already output VisIt readable files, test codes only need limited functionality (VTK)
- Standard output formats, such as HDF5, allow VisIt to deal with data stored on patches, in different materials, as well as allowing for parallel input/output

VTK output

- One of the simplest formats of VisIt-friendly output is VTK (visualization toolkit)
- In many ways, a tabulated data structure with additional header information
- Test codes will need to output this information correctly
  `https://vtk.org/wp-content/uploads/2015/04/file-formats.pdf`



Part 1: Header

Part 2: Title (256 characters maximum, terminated with newline \n character)

Part 3: Data type, either ASCII or BINARY

Part 4: Geometry/topology. *Type* is one of:

```
STRUCTURED_POINTS
STRUCTURED_GRID
UNSTRUCTURED_GRID
POLYDATA
RECTILINEAR_GRID
FIELD
```

Part 5: Dataset attributes. The number of data items *n* of each type must match the number of points or cells in the dataset. (If *type* is `FIELD`, point and cell data should be omitted.

Example vtk script

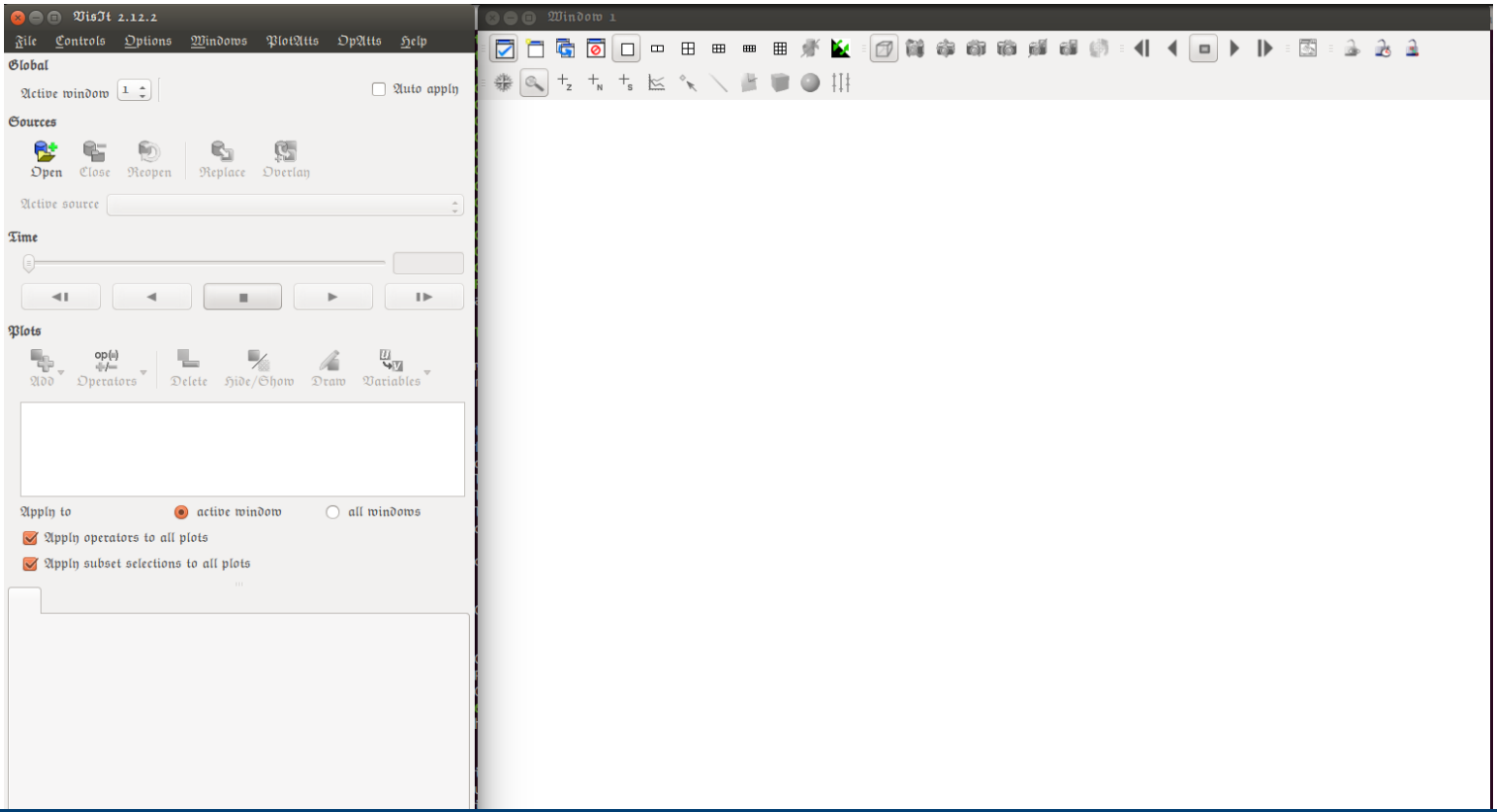
Much of the work we do uses the RECTILINEAR_GRID output type, hence we consider this for an example

```
write(1001, '(a26)') "# vtk DataFile Version 3.0"
write(1001, '(a26)') "Data produced by mf_evolve"
write(1001, '(a5)')  "ASCII"
write(1001, '(a24)') "DATASET RECTILINEAR_GRID"
write(1001, '(a10,3i6)') "DIMENSIONS", &
     npx, npy, 1
write(1001, '(a13,i6,a6)')    "X_COORDINATES", npx, " FLOAT"
do i = 1, npx
  write(1001, '(es21.12E3, a1)', advance = 'NO') q(nnt, i, 1, 1), ' '
end do
write(1001, *)
write(1001, '(a13,i6,a6)')    "Y_COORDINATES", npy, " FLOAT"
do i = 1, npy
  write(1001, '(es21.12E3, a1)', advance = 'NO') q(nnt, 1, i, 2), ' '
end do
write(1001, *)
write(1001, '(a13,i6,a6)')    "Z_COORDINATES", 1     , " FLOAT"
write(1001, '(i1)') 0
write(1001, '(a10,i12)') "POINT_DATA", npx * npy
do k = nv1, nv2
  write(1001, *)
  out_string = "SCALARS "//base_name
  write(n_string, '(i1,a6)') k, " FLOAT"
  write(tmp_string,'(i3)') len_trim(out_string) + len_trim(n_string)
  format_string = "(a"//adjustl(trim(tmp_string))//")"
  write(1001, format_string) adjustl(trim(out_string))//adjustl(trim(n_string))
  write(1001, '(a20)') "LOOKUP_TABLE default"
  do j = 1, npy
    do i = 1, npx
       if (abs(q(nnt, i, j, k+2)) .le. 1.d-16) then
          write(1001, '(es21.12E3)') 0.d0
       else
          write(1001, '(es21.12E3)') q(nnt, i, j, k+2)
       end if
    end do
  end do
end do

close(1001)
```

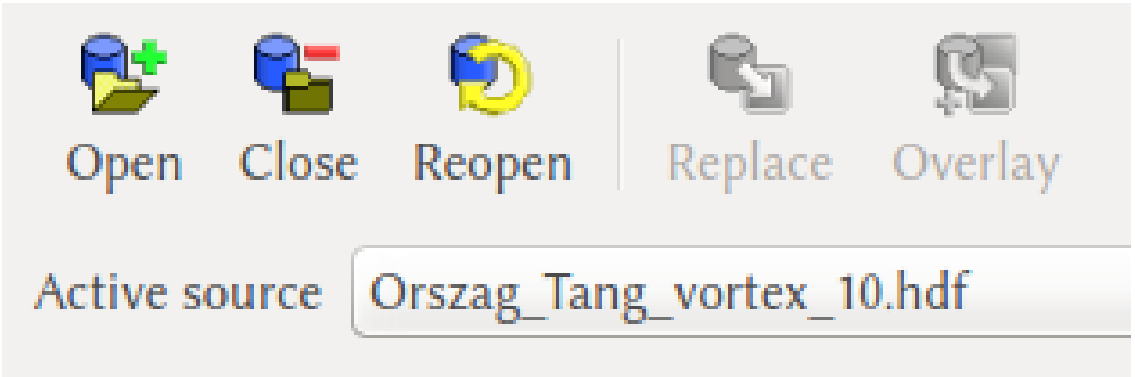
- First information about the data is inserted into the file
- Then the x-coordinates, y-coordinates and, if applicable, z-coordinates
- Then the data, if you have multiple variables, these follow one after another
- Each variable is given a name and a data type
- The data points are then output in the specific order (z-loop then y-loop the x-loop)

Opening VisIt

- When opening VisIt, two windows (hopefully) open
- The window on the left controls all aspects of the plots you make, we refer to this as the 'VisIt window', or the 'main window'
- The window on the right contains the plot, you can have multiple of these

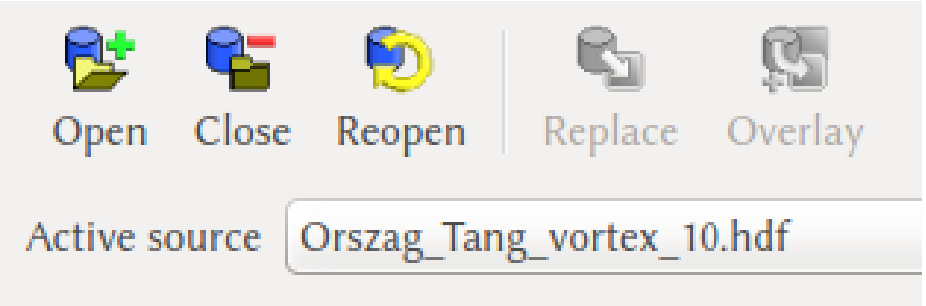


Basics of the interface



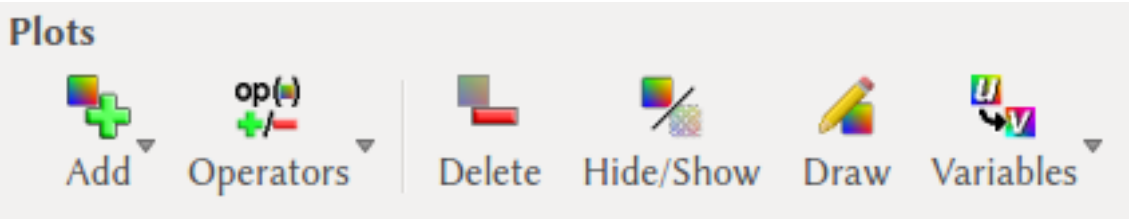
- **Open** and **Close** self explanatory, though closing a file requires *all* plots from it to be deleted
- VisIt can open a file series, hence you can **Reopen** once you have more output - will cause errors if file is being written (or VisIt is feeling grumpy)
- Multiple files can be opened as a database, by default VisIt groups numerical files of the same file type automatically
- **Active source** shows the files/databases you currently have open, and you select which one you want to plot

Basics of the interface



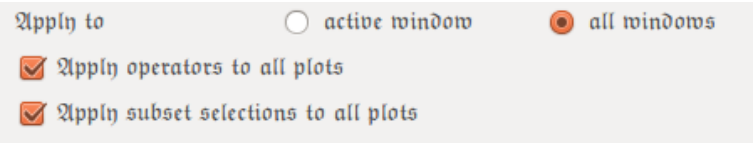
- Plots appear in the order they were created in the main window
- This order determines what appears 'on top' - the most recent plot will always on top
- *Sometimes*, VisIt is a bit cleverer - e.g. contours will always appear on top of colour plots in 2D
- **Replace** will replace a highlighted plot with the current **Active source**, maintaining all formatting
- **Overlay** will duplicate the current highlighted plot, but use the current **Active source** instead
- Note - multiple plots can be selected (hold shift or control)

Basics of the interface



- **Add** a new plot
- Add **Operators** to an existing plot, altering how it looks
- **Delete** a plot entirely, or **Hide/Show** to toggle its visibility
- **Draw** a plot, once it has been added, operated upon, or changed significantly
- **Variables** list, from all the variables available in your output file
- By default, VisIt often sets its main window size such that the **Variables** menu is hidden (it can be accessed through the '≪' button)

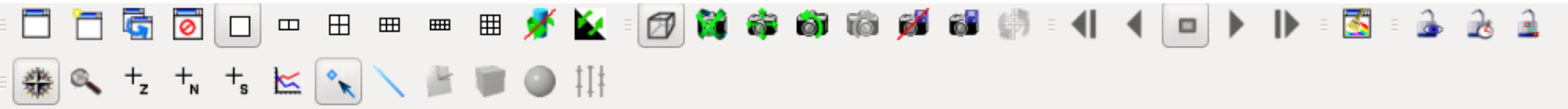
Basics of the interface



- These are three options which select how you modify plots
- Operators are added through the **Operators** menu, subset selections are available once you've made a plot (covered later)
- VisIt can have multiple plots open, and visible, at once, and you may wish to alter them all in the same way (e.g. revolve)
- These options are on by default (though not on an LSC machine), but are better turned off (I think)
- You can also alter operators across all windows open (off by default)

Basics of the plot window - row 1

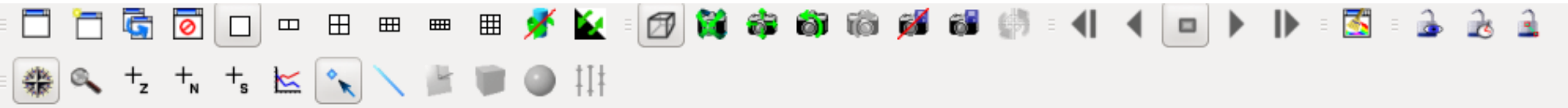
At the top of the plot window are a series of buttons



- Numbered by position from the left
- 1 - Activate window, for when you have multiple windows open
- 2 - New (blank) window
- 3 - Clone window (and all plots on it)
- 4 - Delete window (also achieved by the 'x' button)
- 14 - Reset view (when you've zoomed in, and want to return)
- 21-25 - Change frames - also in the main viewer

Basics of the plot window - row 2

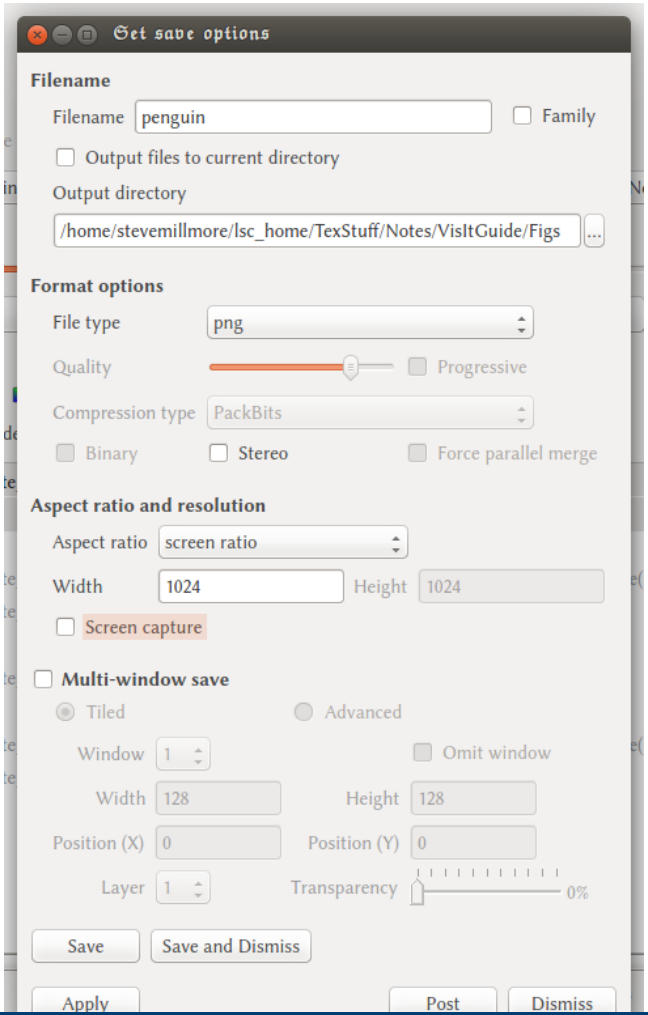
At the top of the plot window are a series of buttons



- Numbered by position from the left
- 1 - Navigate mode - click and drag the plot, slides the plot in 2D, revolves in 3D (holding ctrl will translate the plot in 3D)
- 2 - Zoom mode - use mouse to draw a zoom region
- 3/4/5 - Zone/Node/Spreadsheet pick - Click on a point, and cell centred, node centred or spreadsheeted data will be given about it
- 6 - Lineout - 2D only, click and drag a line and a 1D slice along that line will be plotted in a new window

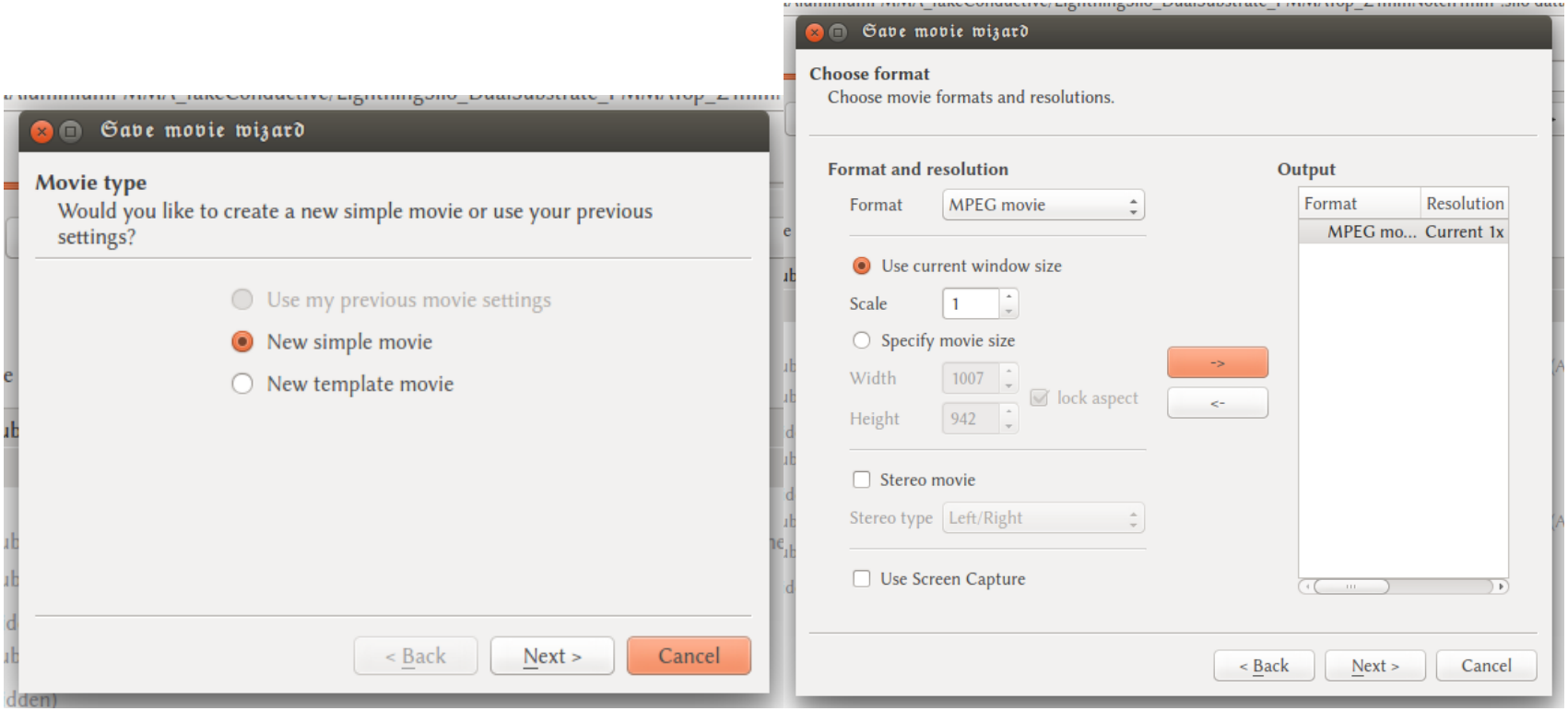
Saving the window

Default (ctrl-s) might not save where you want it to, nor with a useful name (default filename is 'visit')



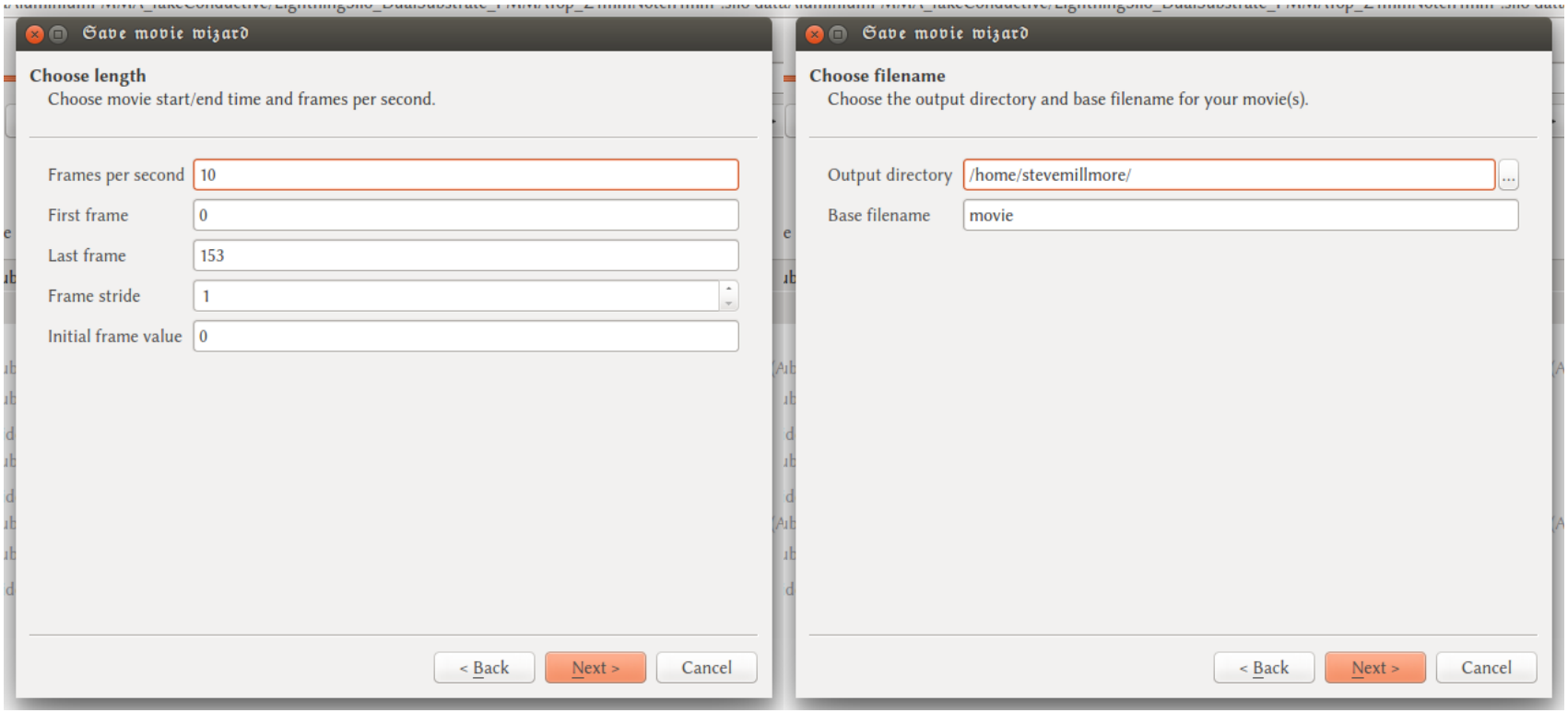
- File → Set save options
- **Family** will consecutively number each plot (default on)
- Default is to **Output files to current directory**, '...' button lets you choose directory
- Various formatting options, traditional graphics format, but also 'curve' - can be used to save 1D output, and be read by gnuplot
- **Screen capture** - save picture at current screen resolution (often suitable for presentations) - buggy in some older versions

Saving movies (in theory)



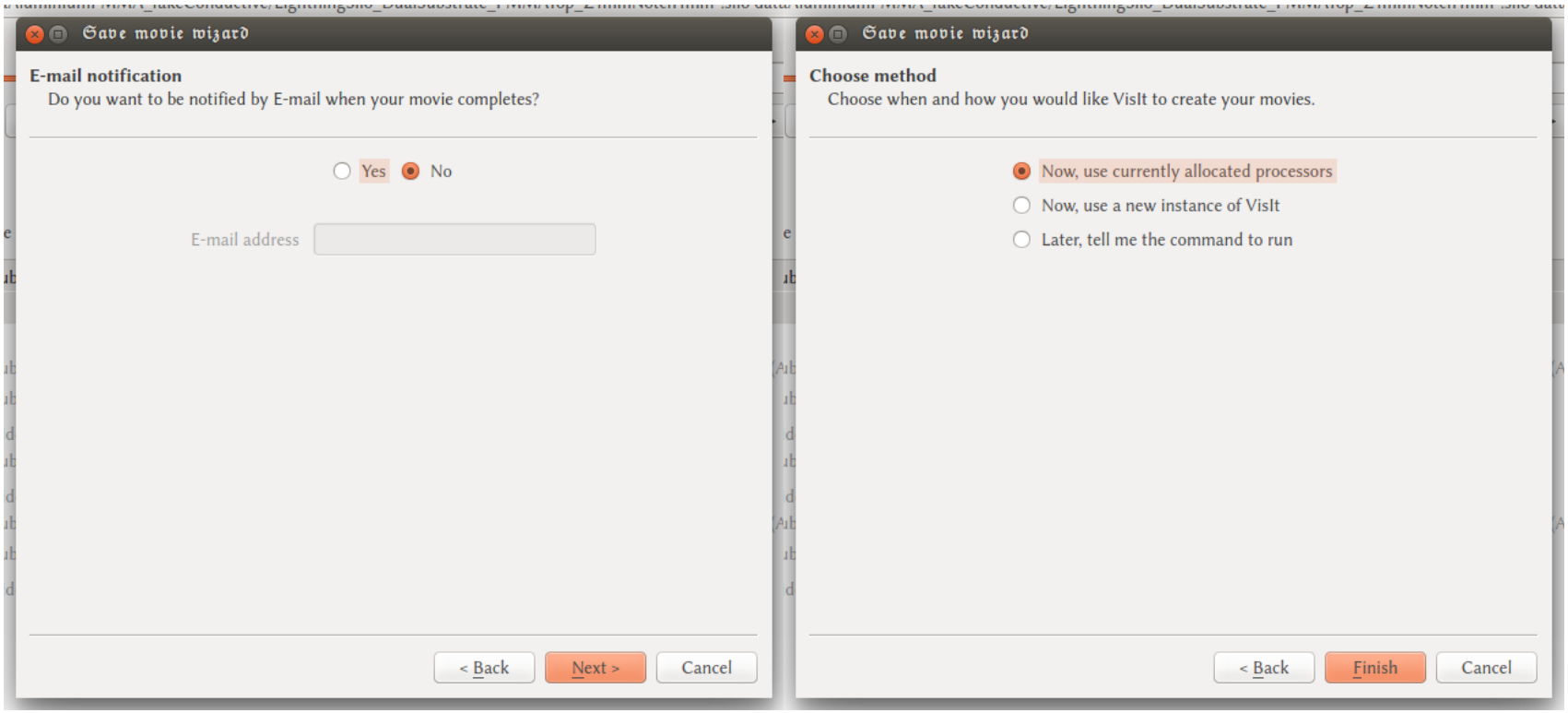
- This can also be used to save some or all of the images in the current database

Saving movies (in theory)



- Frames per second only does something if you successfully make a movie, not images

Saving movies (in theory)

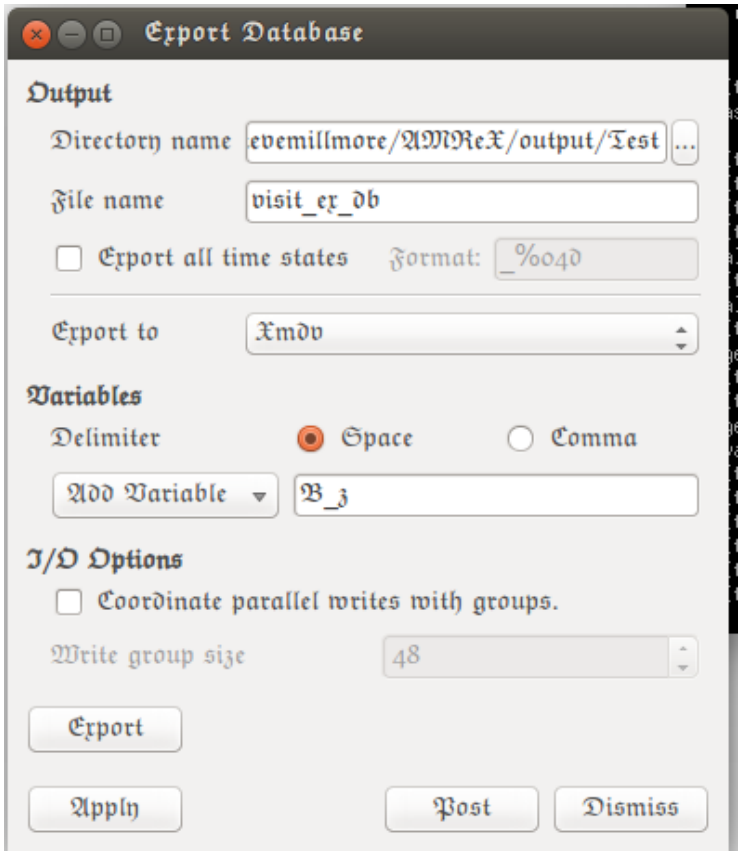


- Finish will bring up an additional window (visit terminal instance) running the movie script, and may result in a movie - if it fails, it will still give the files it saved

Other save options - save session

- Saving a session allows you to save the plots you've already made, including operators, so that you can work on them later
- Useful if making a complex plot, especially if VisIt keeps crashing on you
- Two restore options - 'restore session' and 'restore session with sources'
- The first will reload exactly as you had the session before
- The second allows you to choose the sources you load e.g. if you want to re-do a plot using different data files, but to show the same thing

Other save options - export database

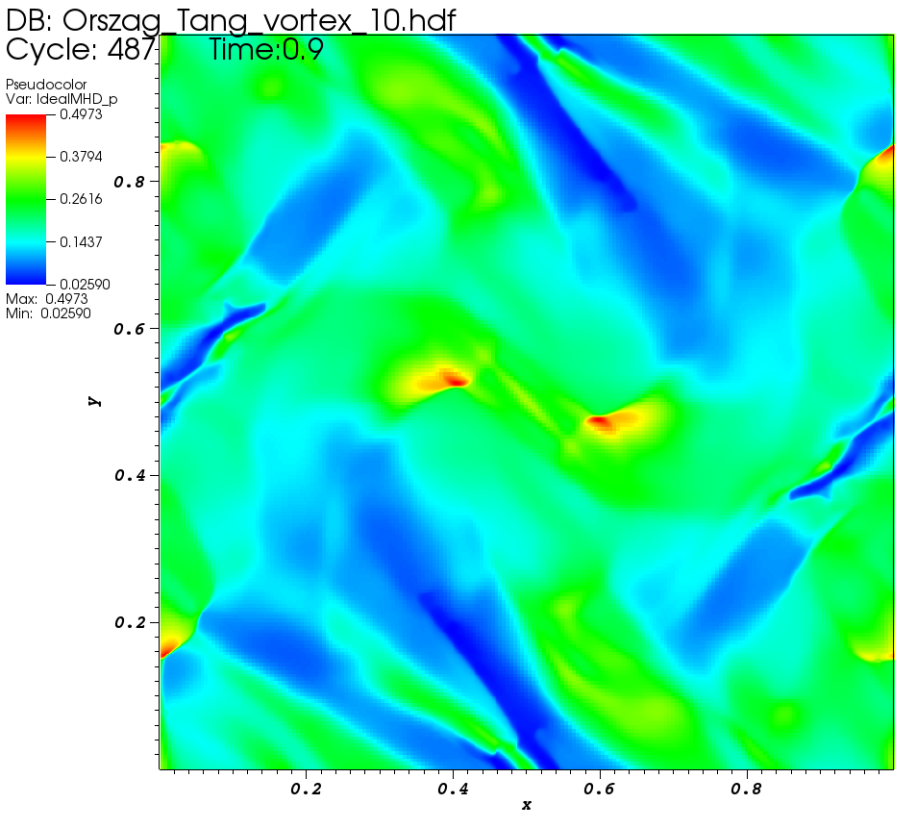


- Exporting a database allows you to save the data under a different format
- Some of the formats are also available from the save window, but in this case, you can append the entire, time-varying database to the file
- Others formats need to be exported this way - doing this, it is possible to get VisIt output into gnuplot readable files (though not necessarily optimally output)
- Note - not all output forms can deal with AMR, multiple materials very well

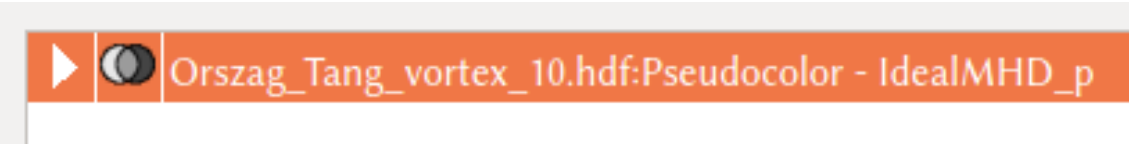
Outline

Plotting - **Pseudocolor**

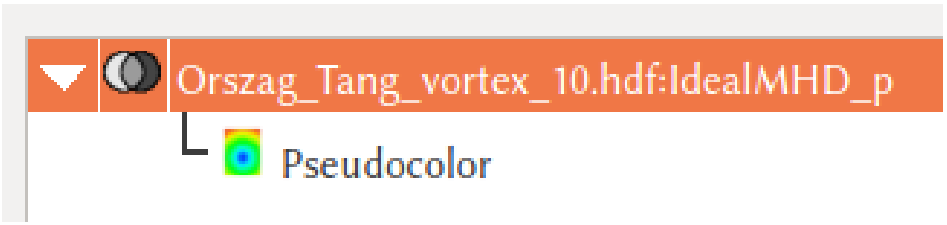
The default colour map style plot - **Add** $\rightarrow$ **Pseudocolor** $\rightarrow$ <**variable**>



Plotting - **Pseudocolor** - modifying the plot

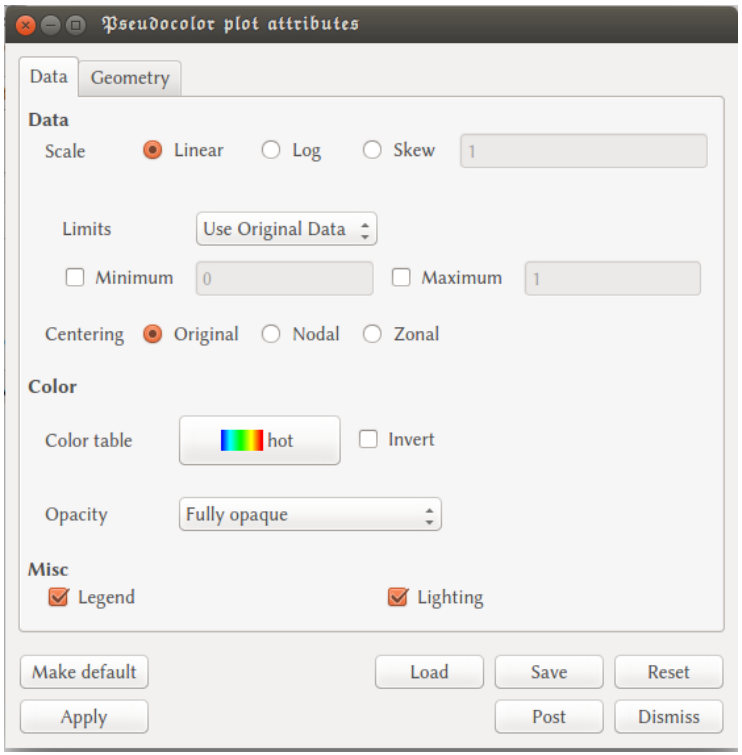


Clicking the triangle...



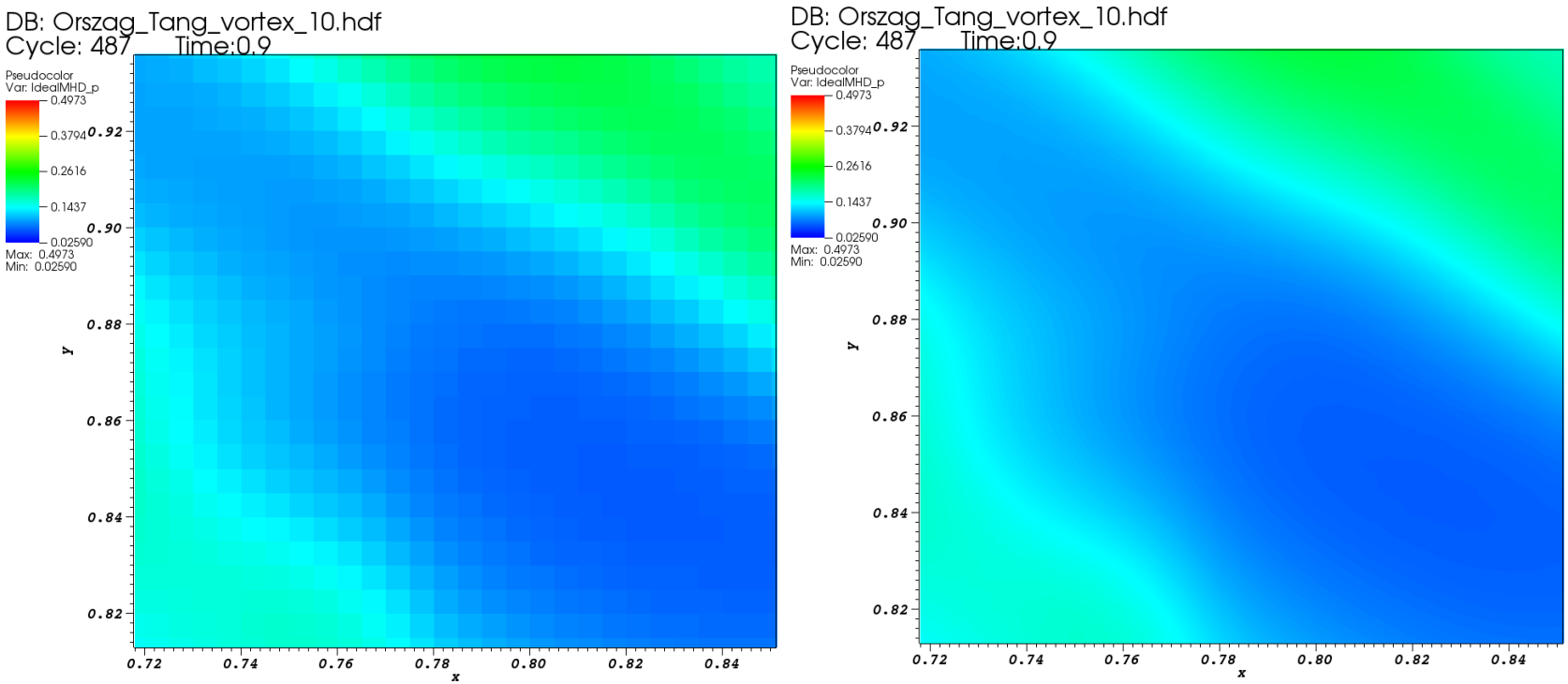
- Clicking the two circles will bring up the **Subset** menu
- Clicking **Pseudocolor** will bring up the **Pseudocolor plot attributes** menu
- Any other operators on the plot will be listed when clicking the triangle

Plotting - **Pseudocolor** - attributes



- **Scale** - If using a log scale, VisIt will complain if any values are 0 or negative
- **Limits** - Change when making a series of plots, for consistency, or when interested in data at a particular value
- **Centering** - see next slide
- **Color** - Various colour schemes are available, some hideous
- **Misc** - Turning off the legend can be useful, e.g. multiple plots at the same scale, or creating aesthetically pleasing, but data-light plots

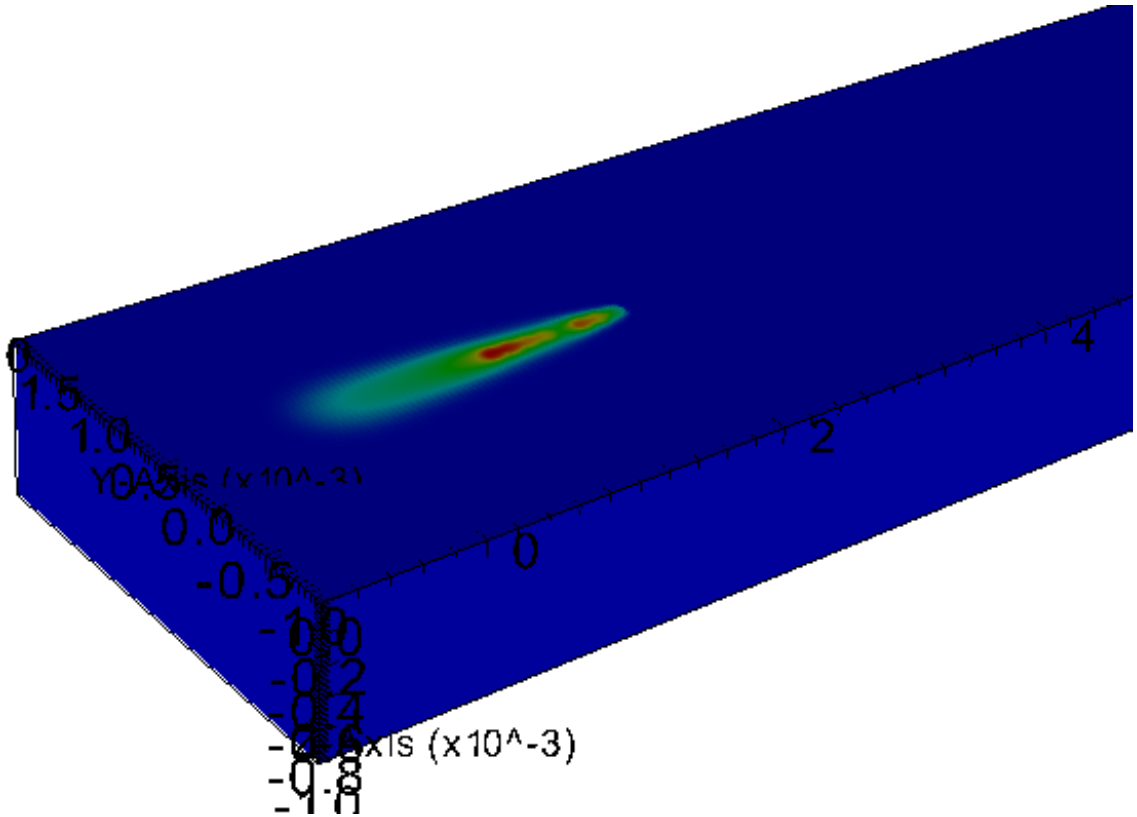
Plotting - **Pseudocolor** - attributes - centering



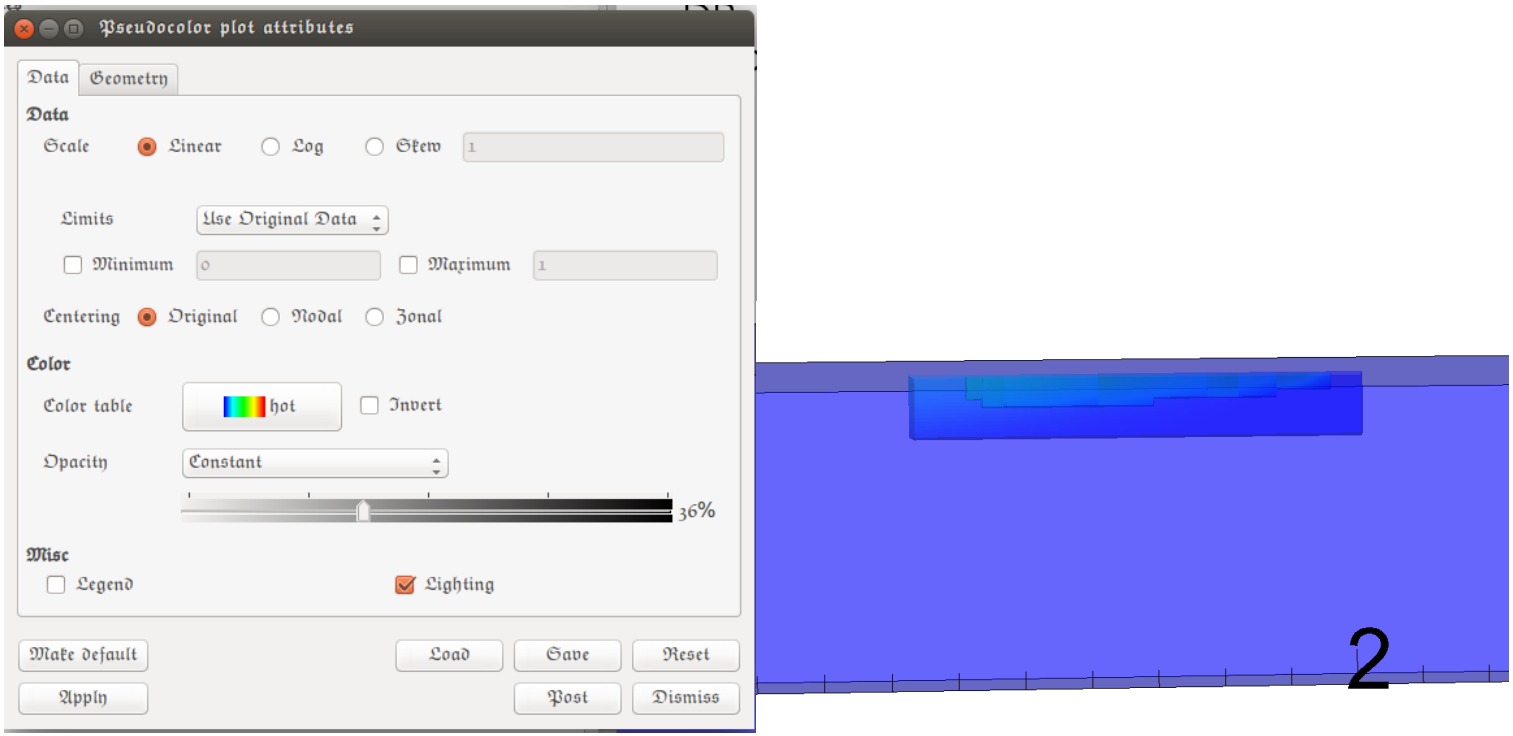
- **Zonal** on the left, **Nodal** on the right - the group code defaults to zonal

Plotting - **Pseudocolor** - 3D

By default, a 3D pseudocolour plot shows an opaque box and the external data plotted

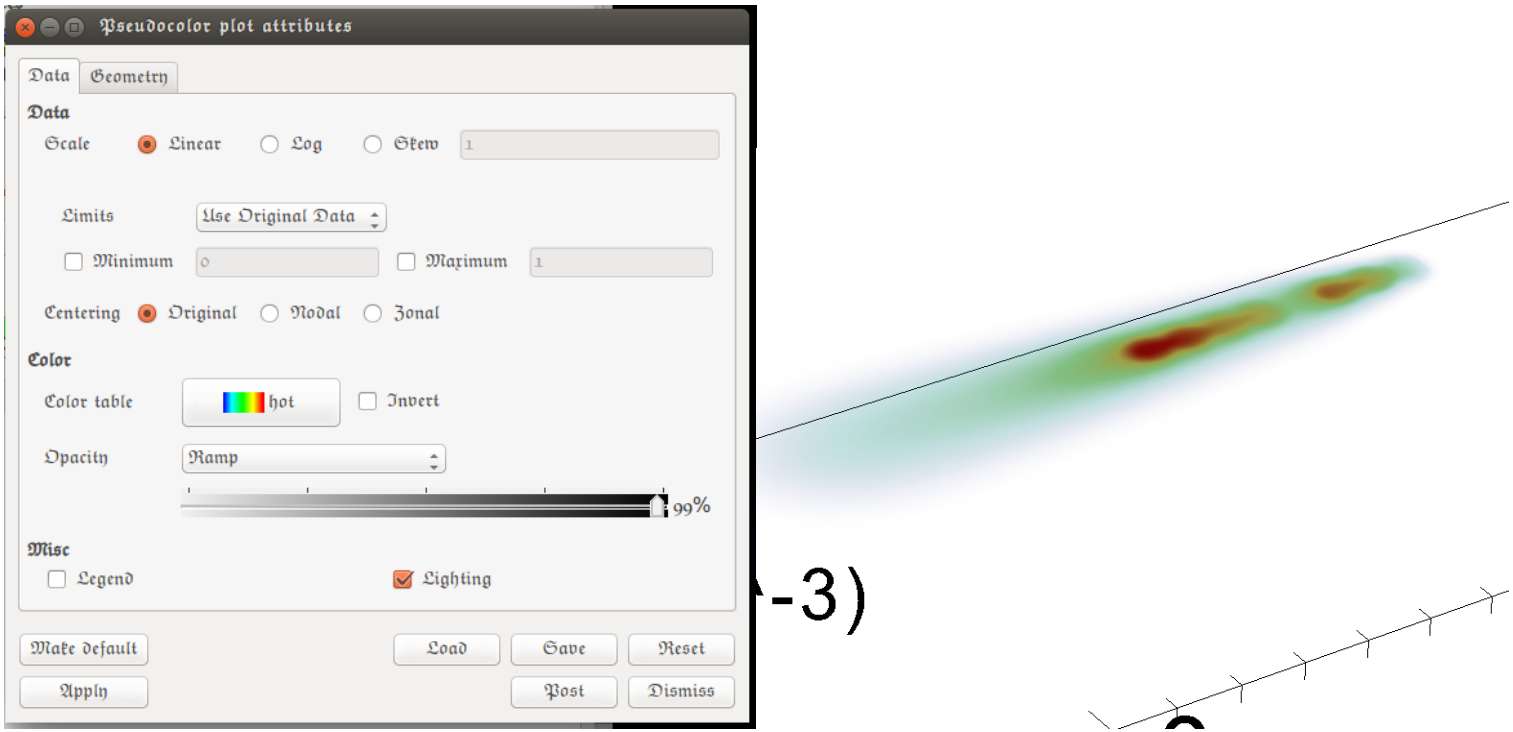


Plotting - **Pseudocolor** - attributes



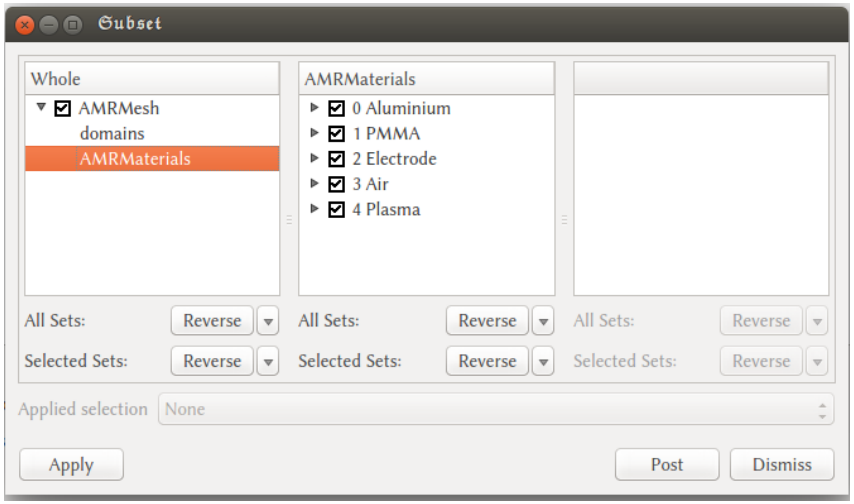
- Altering opacity can show behaviour within the domain
- If nodal centring is used, this will, however, show patch boundaries

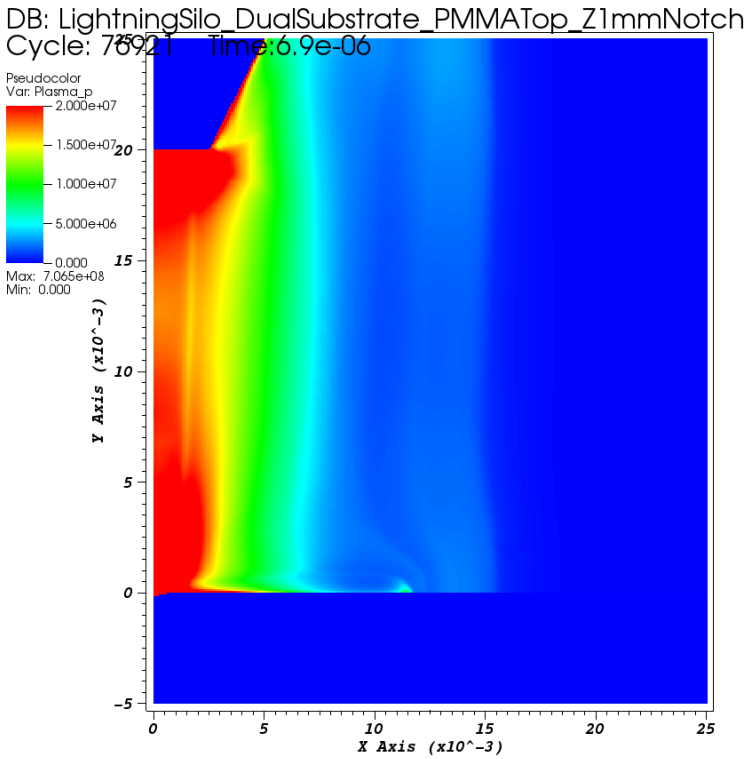
Plotting - **Pseudocolor** - attributes



- Zonal centring avoids this
- This is an example of a ramped opacity gradient

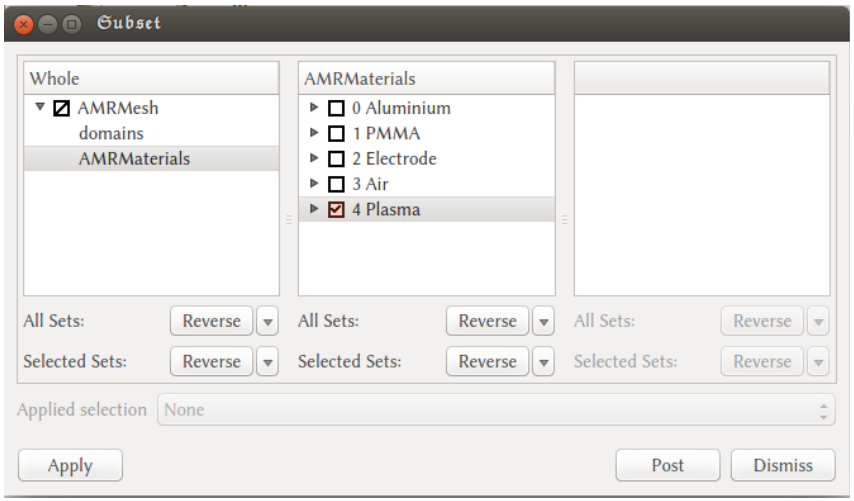
Plotting - **Pseudocolor** - **Subset** from plot description

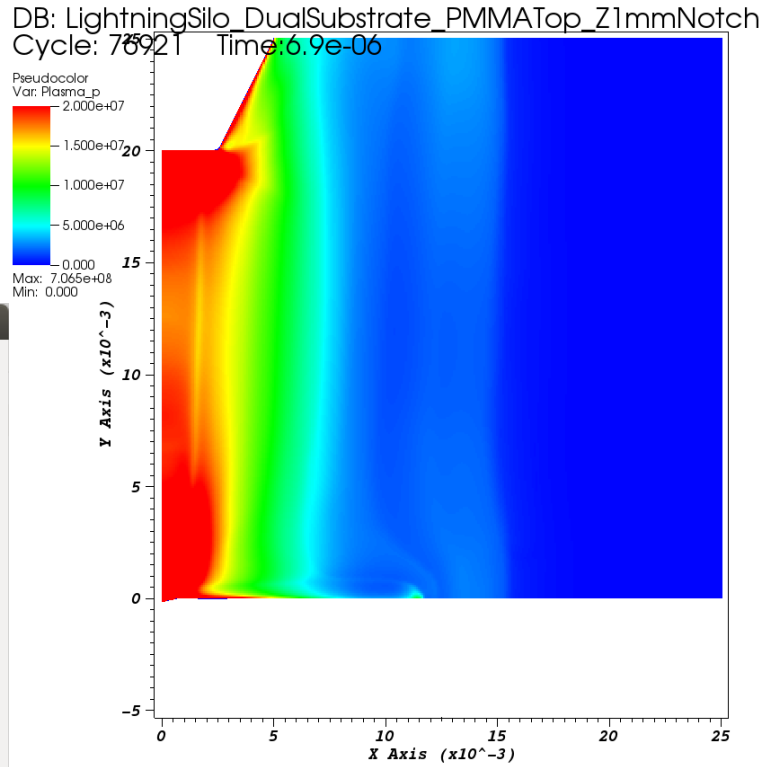




- VisIt output can allow for **Materials**, i.e. quantities which each have their own set of variables
- By default, these exist everywhere in the domain, but are 0 where they don't actually exist

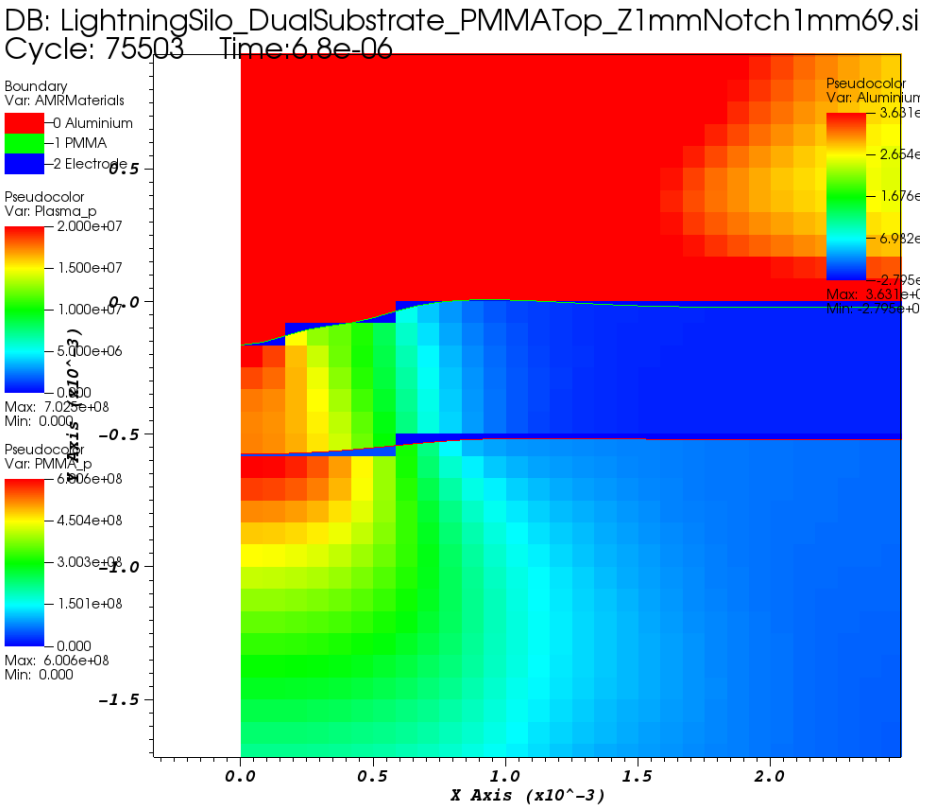
Plotting - **Pseudocolor** - **Subset** from plot description





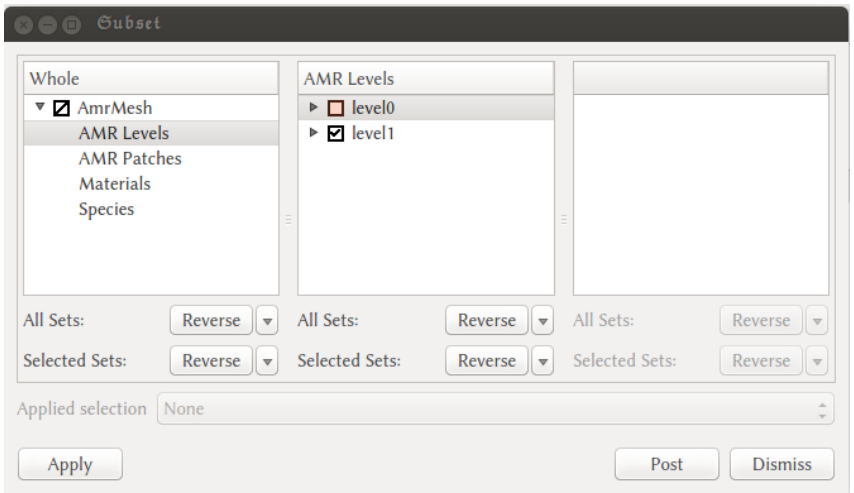
- Through the plot descriptions **Subset** menu (the two circles), materials can be turned on and off

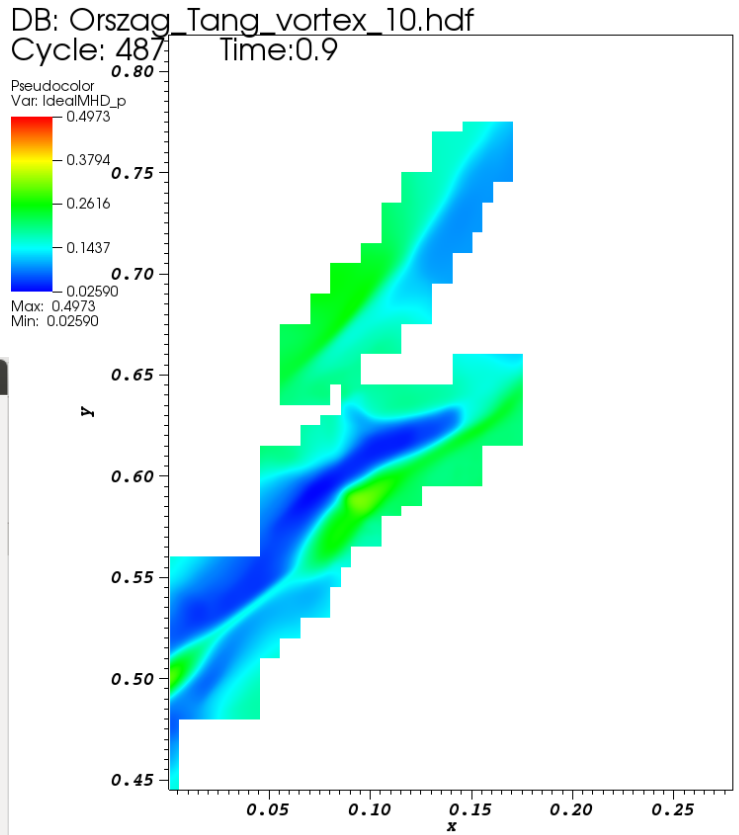
Plotting - **Pseudocolor** - **Subset** from plot description



- Multiple plots, limited by subset, can be plotted
- The material boundaries are interpolated automatically by VisIt, but code output may have only a single value per cell
- This will result in plot artefacts when zooming in

Plotting - **Pseudocolor** - **Subset** from plot description

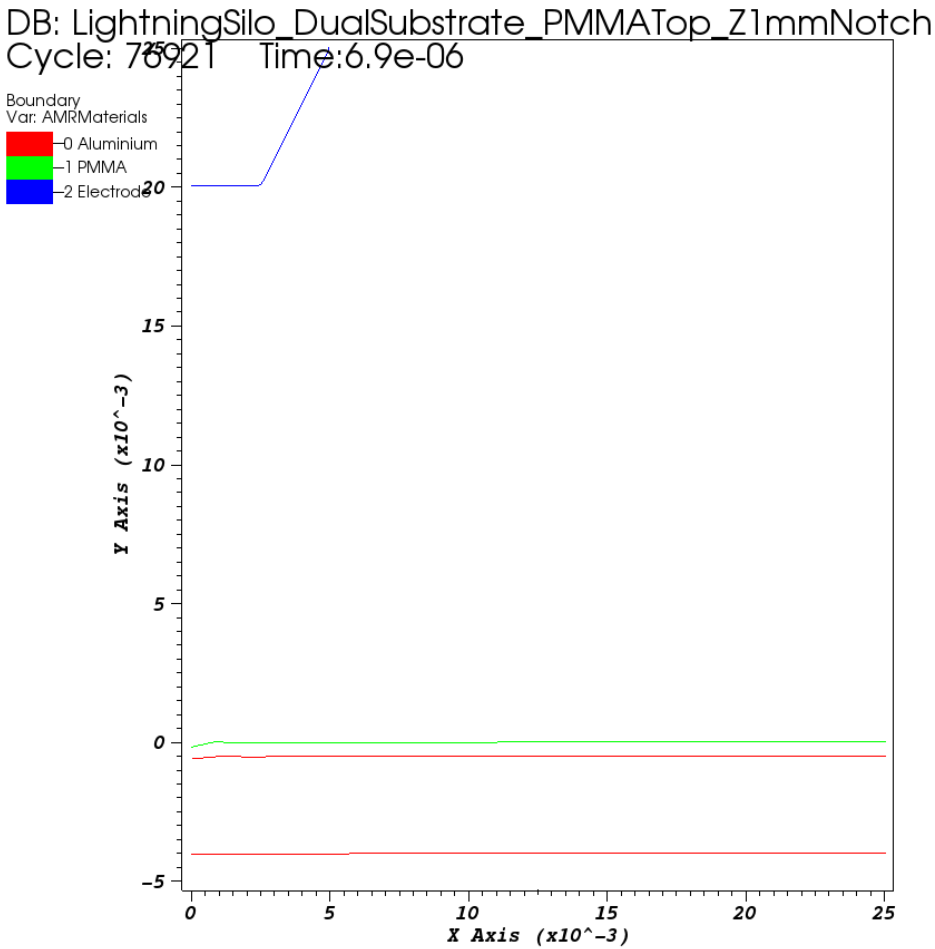




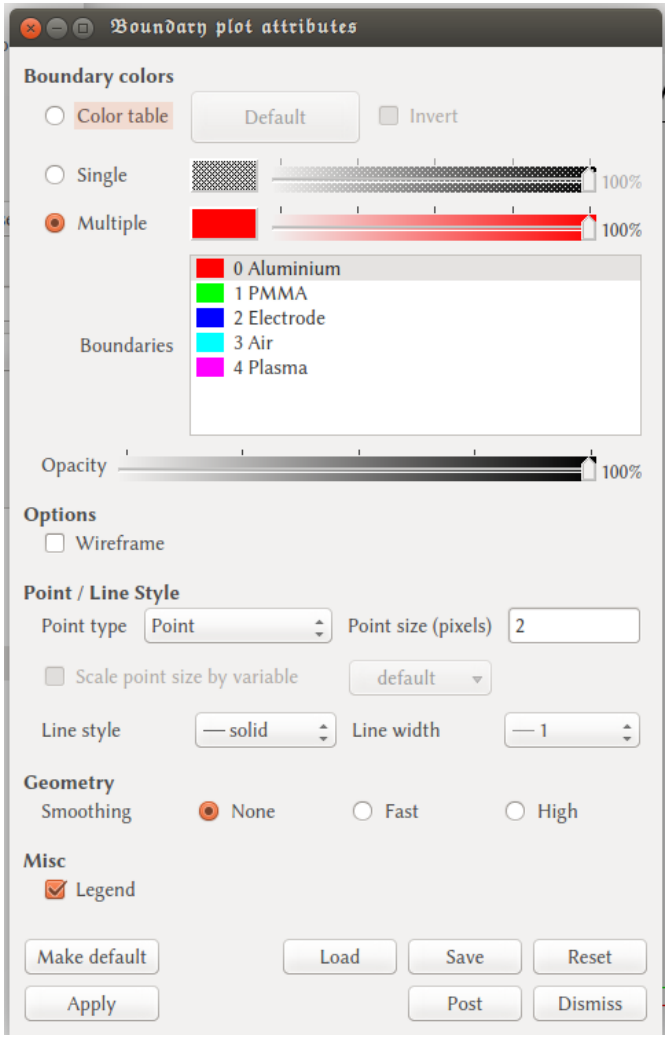
- **Subset** can also be used to show only a single refinement level
- It can even show just a single patch (use **Pick** to identify the patch in question if there are many)

Plotting - **Boundary**

Show only boundaries between materials - **Add** → **Boundary** → <**boundary_variable**>



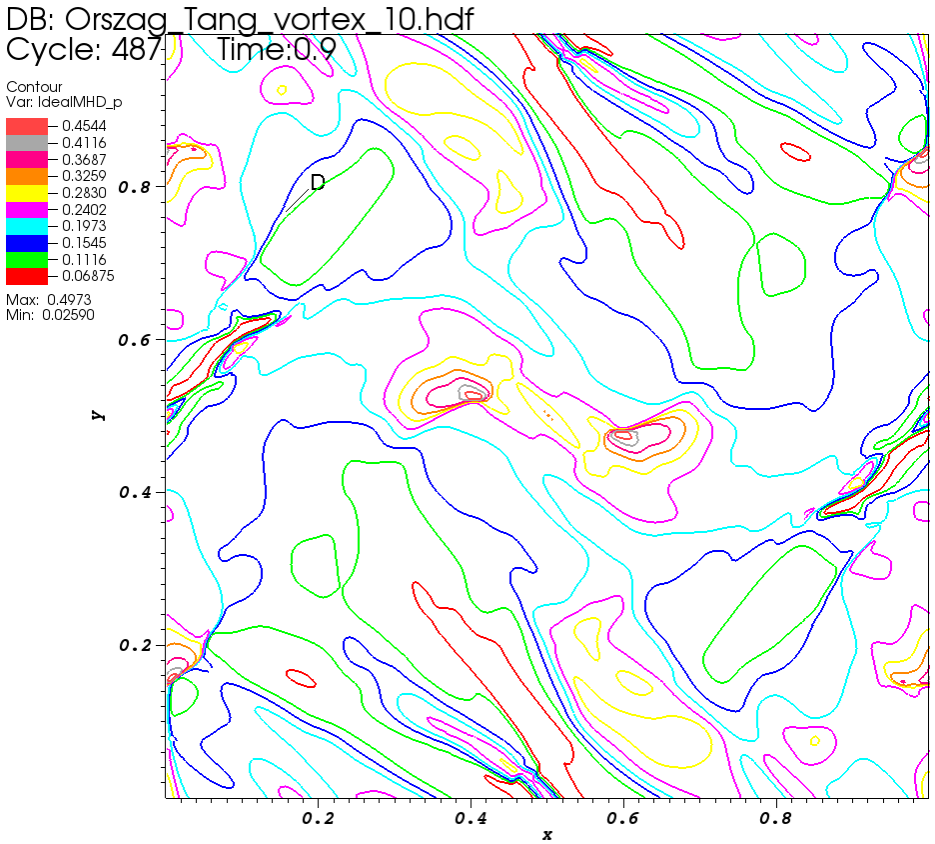
Plotting - **Boundary** - attributes



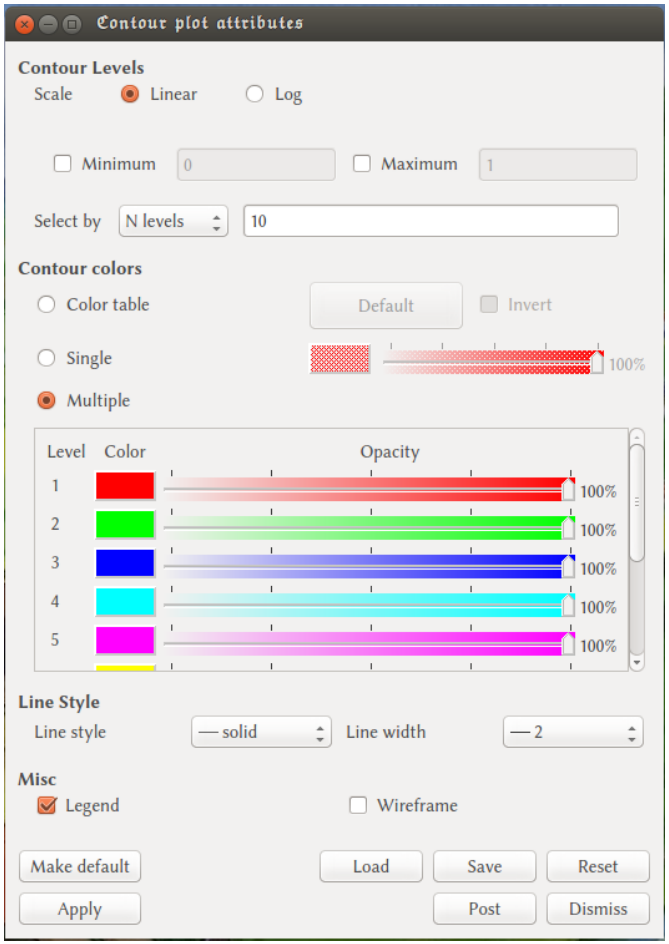
- **Boundary colors** - Each boundary can be individually chosen, or a single colour, whatever is best for visibility
- **Point/Line Style** - adjusting line width is often useful, to highlight boundaries (and hide material subset artefacts!) The style of the boundary line can also be altered.
- **Misc** - the legend is often useless, and can be removed

Plotting - **Contour**

Show contours of a variable - **Add** → **Contour** → <**variable**>



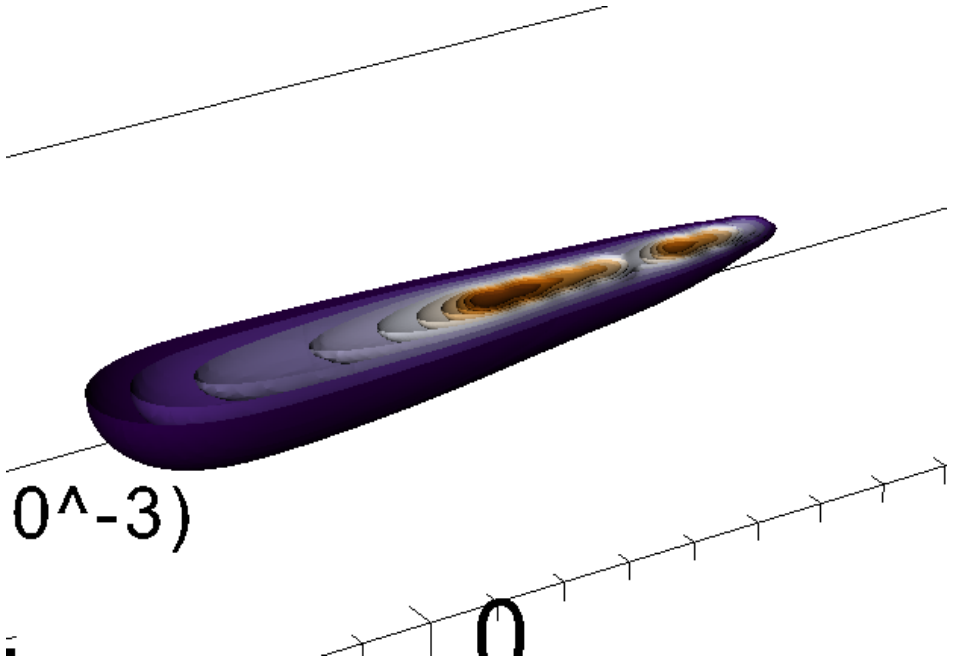
Plotting - **Contour** - attributes



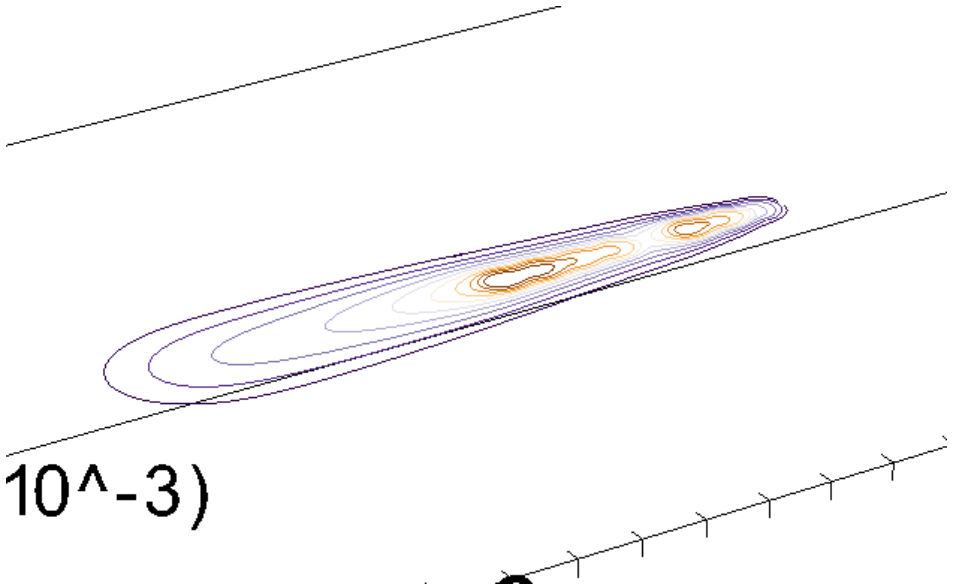
- **Contour Levels** - change the contour number, spacing and selection. 10 evenly spaced contours is default. Can **Select by**:
- **N levels** - alter number of evenly (or log) spaced contours
- **Value(s)** - fixed parameter values for the contours - must be a space-separated list (e.g. '0 1 1.4 4')
- **Percent(s)** - Fixed intervals of the variable range - again a space separated list without '%'
- **Contour colours**, **Line style** and **Misc** work as for boundary plots

Plotting - **Contour** - 3D

Contour plots in 3D produce surfaces of constant value, unfortunately opacity of a colour scheme cannot be changed, only of individual colours

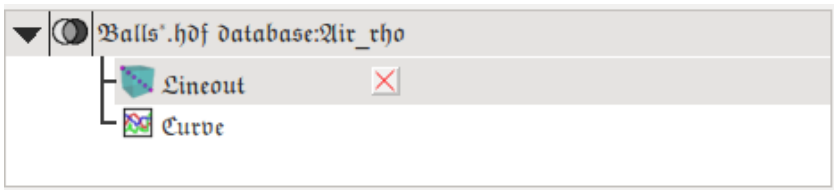


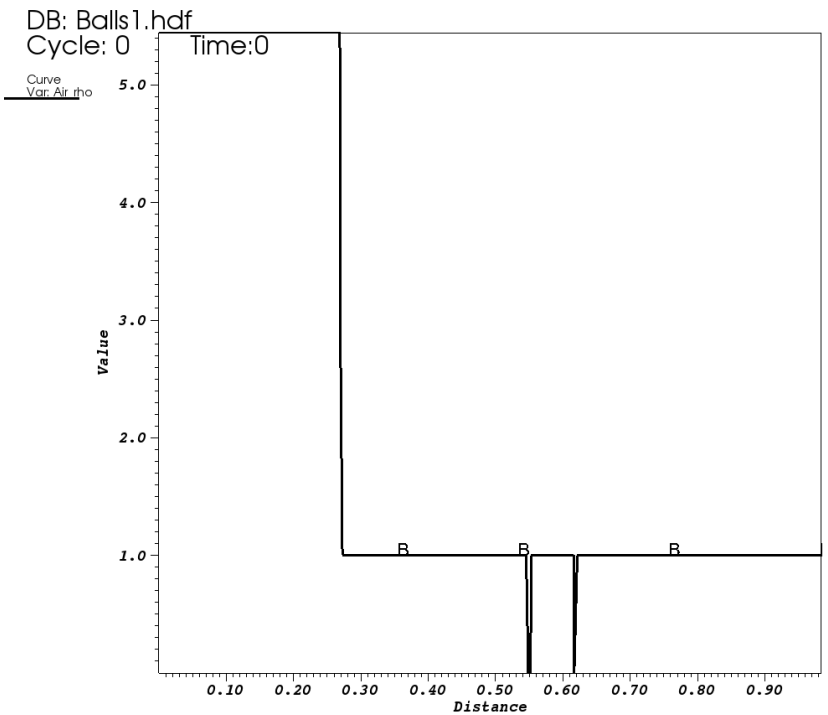
Plotting - **Contour** - attributes



- If contours meet a domain boundary, **Wireframe** can be used to show them only at the boundary
- Operators can generalise this, however

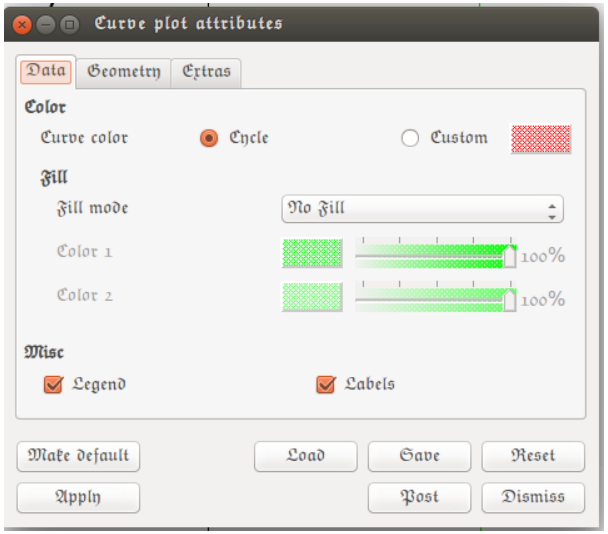
Plotting - **Curve**

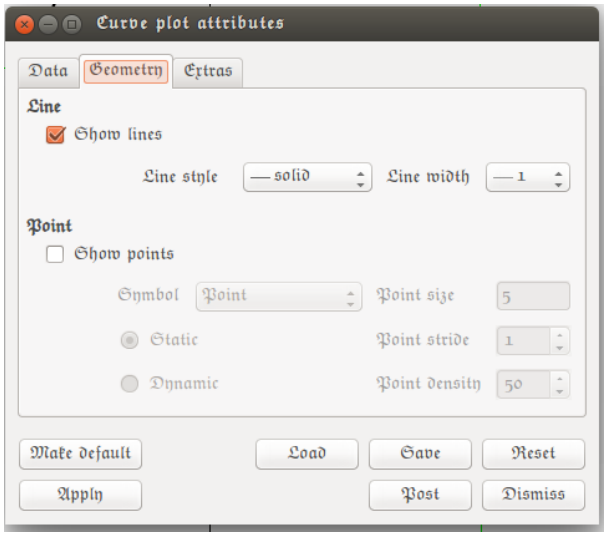




- Generates 1D data from both 2D and 3D plots (equivalent to the **lineout** window button in 2D)
- **Add** → **Curve** → **operators** → **Lineout** → <**variable**>
- A curve plot has two ways to modify the data, **Lineout** and **Curve**
- Distance is always measured from the start of the line, regardless of start coordinate

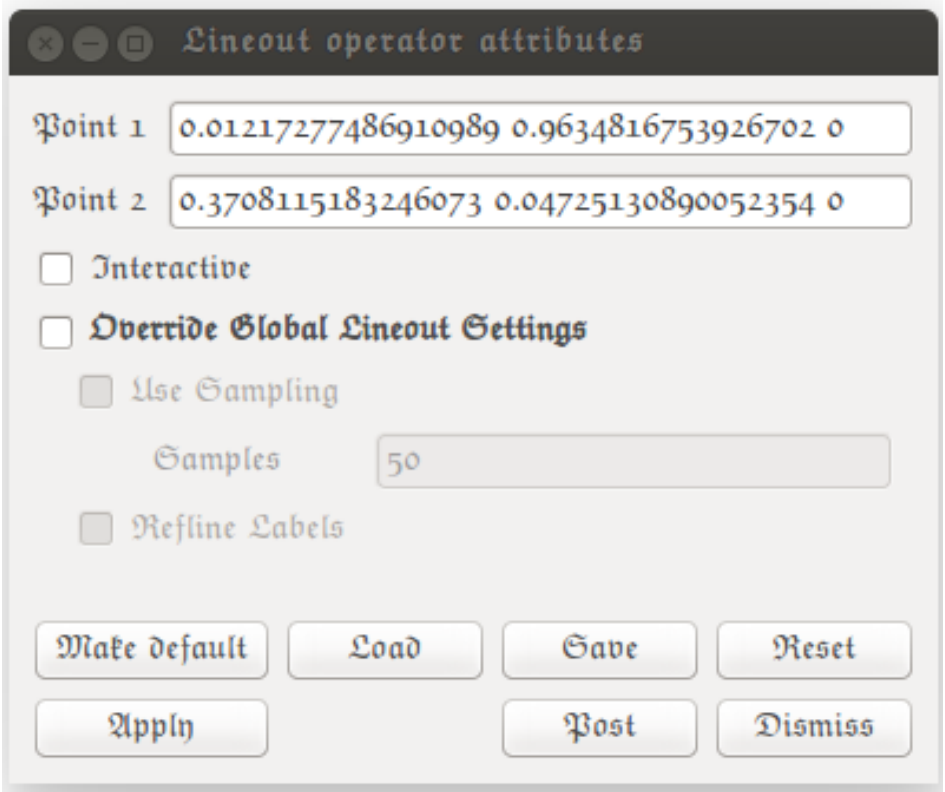
Plotting - **Curve** - curve attributes





- **Curve colour** - by default, VisIt cycles through curve colours, happily plotting the near-invisible yellow - this can be changed
- **Fill** - solid colour fill beneath the plotted line
- In addition to the **Legend**, the **Labels** (letters along the plot) can be turned off
- **Line** - line style (e.g. thickness) or turn off altogether
- **Point** - show the actual data points, stride takes one point for every n cells passed through (regardless of the amount of cell passed through for diagonal lines)

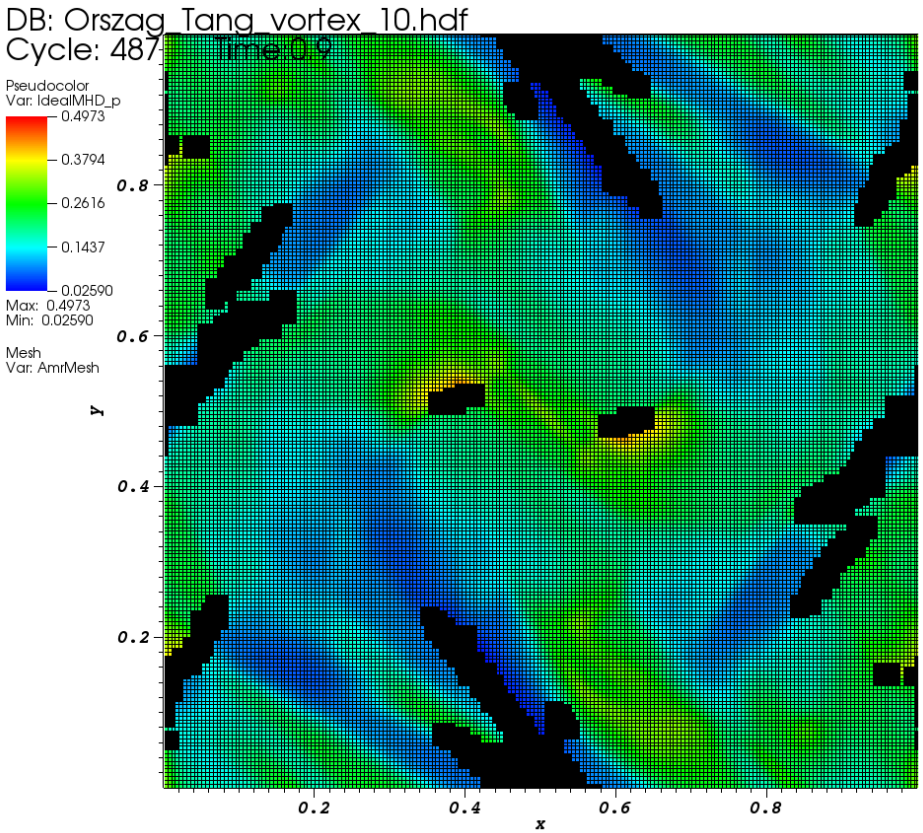
Plotting - **Curve** - Lineout attributes



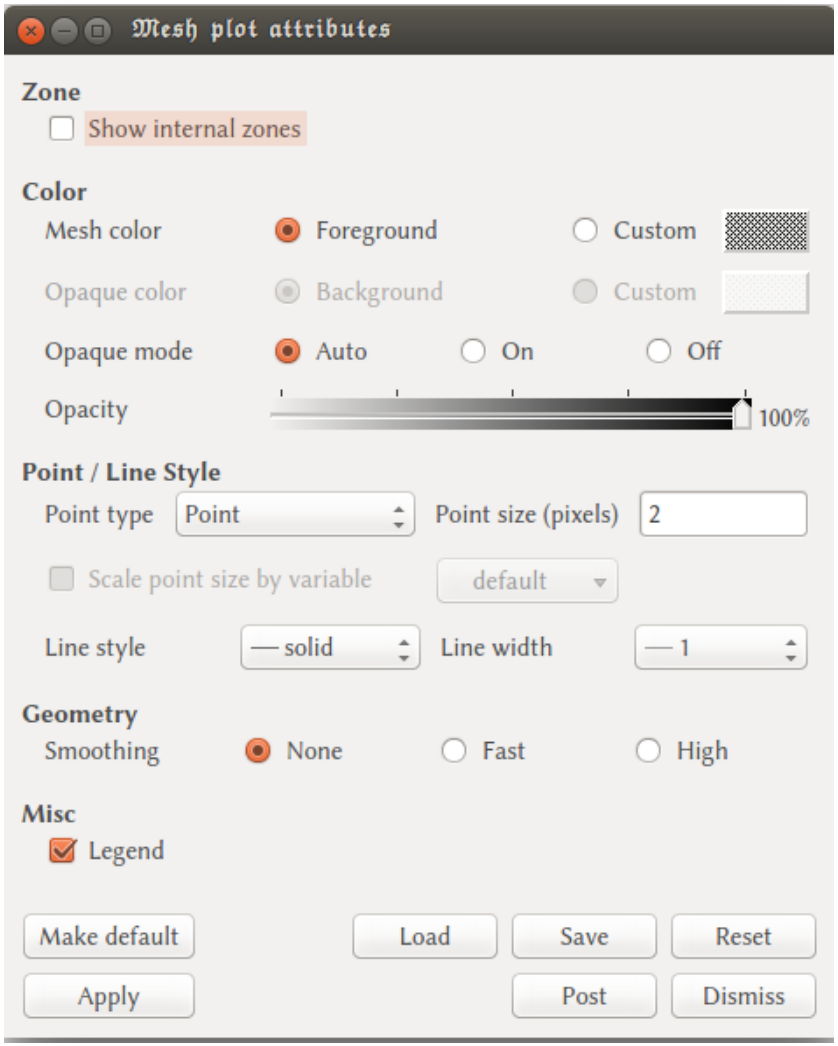
- The start and end points of the line can be controlled
- By default, VisIt uses 50 **Samples** - this may not be enough to capture sharp features

Plotting - **Mesh**

Show the mesh - **Add** → **Mesh** → <**mesh_variable**>



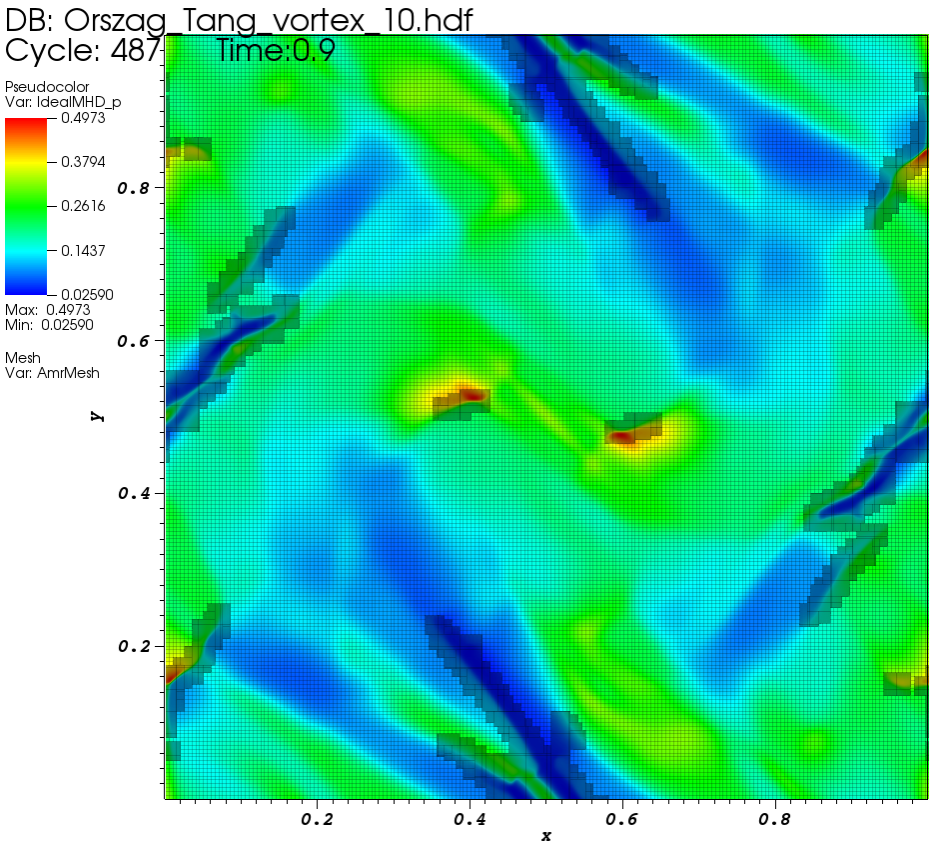
Plotting - **Mesh** - attributes



- **Color** - As seen above, the default mesh opacity of 100% obscures the plot beneath it
- Typically, values of 20% to 30% are good if you wish to see both plot features and the mesh

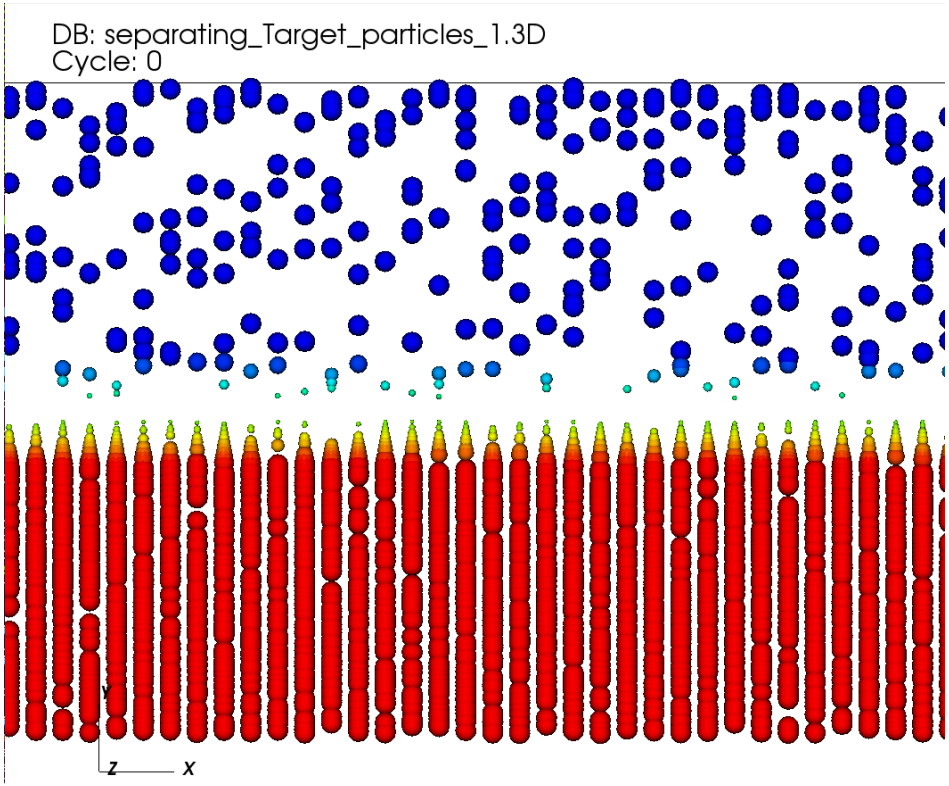
Plotting - **Mesh**

This plot has mesh opacity 20%



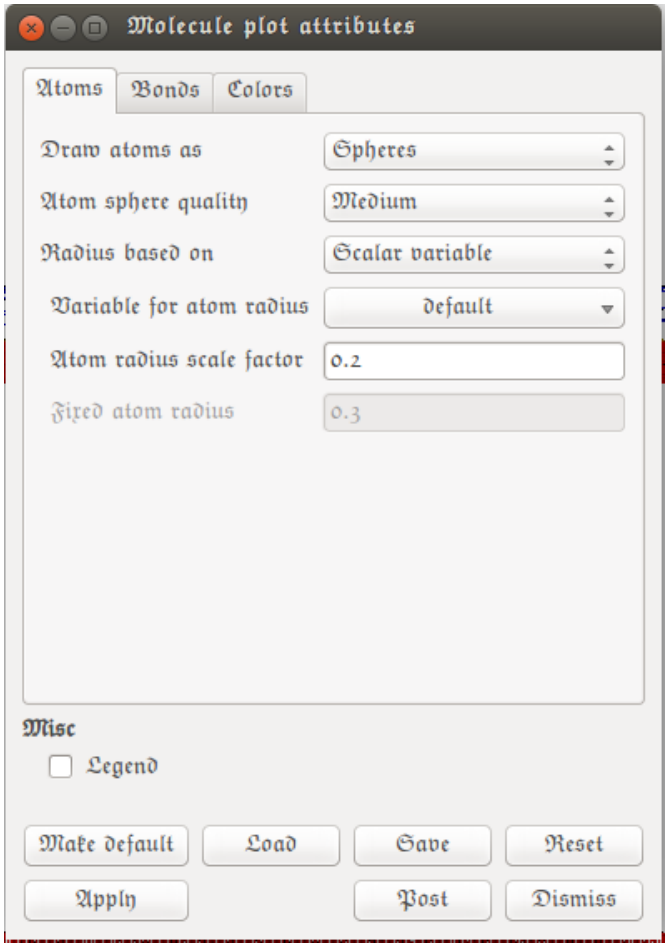
Plotting - **Molecule**

If code outputs particles, these can be plotted through
Add $\rightarrow$ **Molecule** $\rightarrow$ **radius**



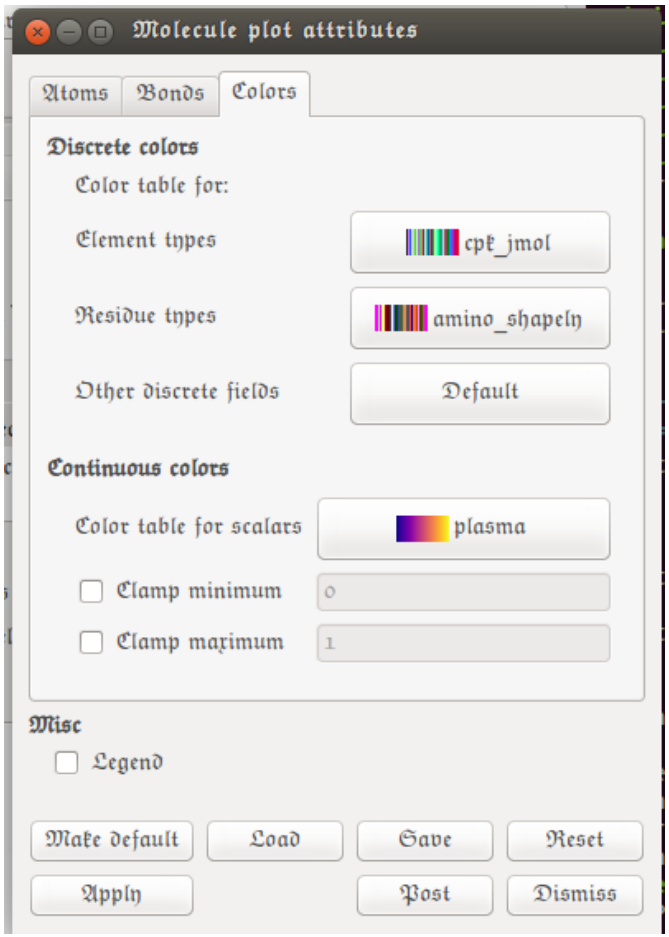
Simulation courtesy of Tomé

Plotting - **Molecule** - attributes



- The use of particles in a level set method is not the original intent of the molecule plot option
- Another use of VisIt - molecular visualisation (not covered here)
- **Radius based on** - when not dealing with atoms, options are **Fixed value** and **Scalar variable**
- Note - particles are 3D spheres, even when generated by a 2D level set function - the **Project** or **Slice** operators can be used if you wish to plot with 2D images (see later)

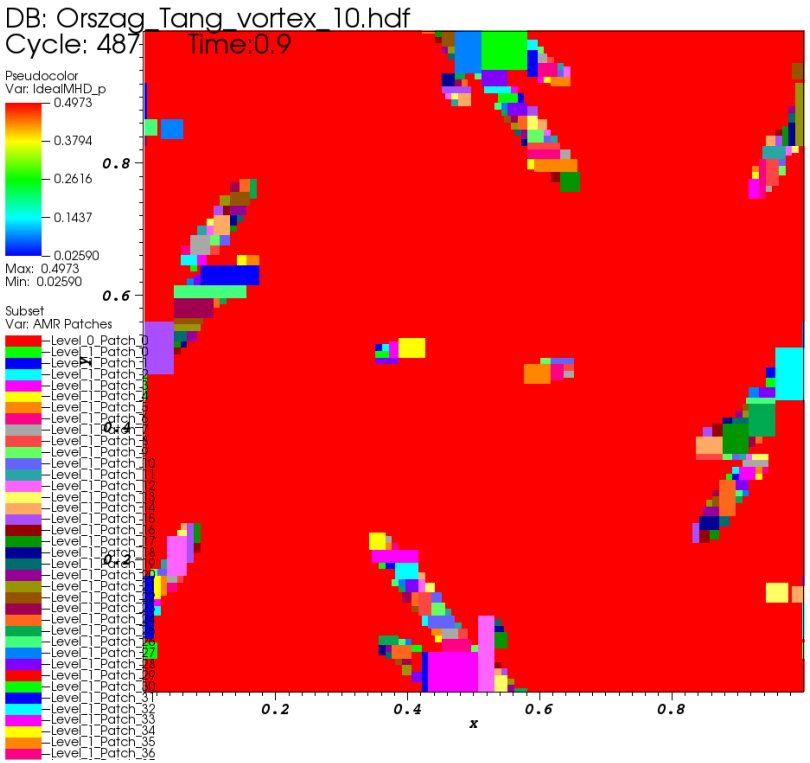
Plotting - **Molecule** - attributes

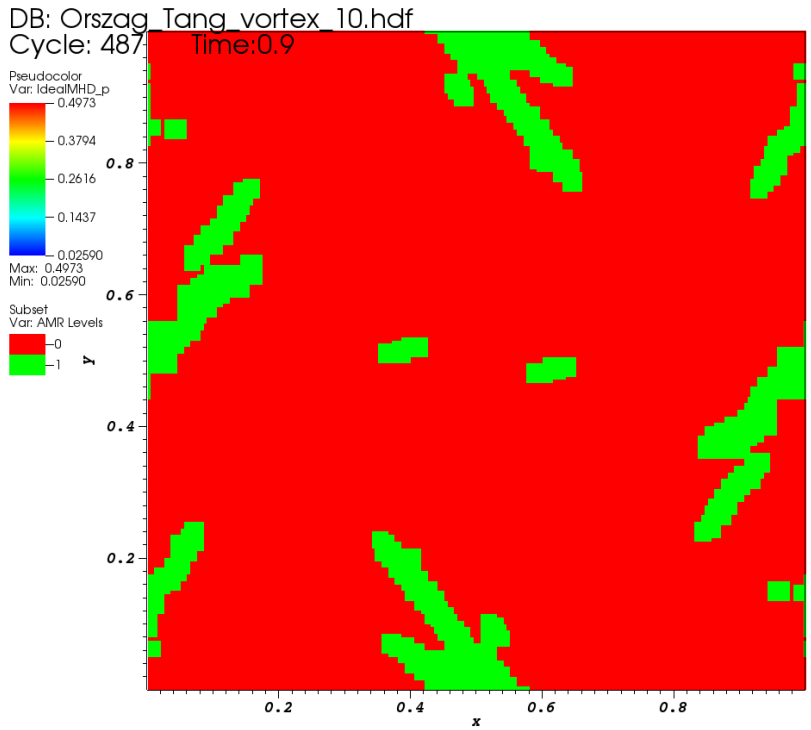


- Two types of particle output are possible, discrete or continuous (the group code outputs continuous)
- The **Colors** options depend on the particle output, for continuous output, options are similar to other colour selections
- When plotting level set based particles, **Bonds** does nothing

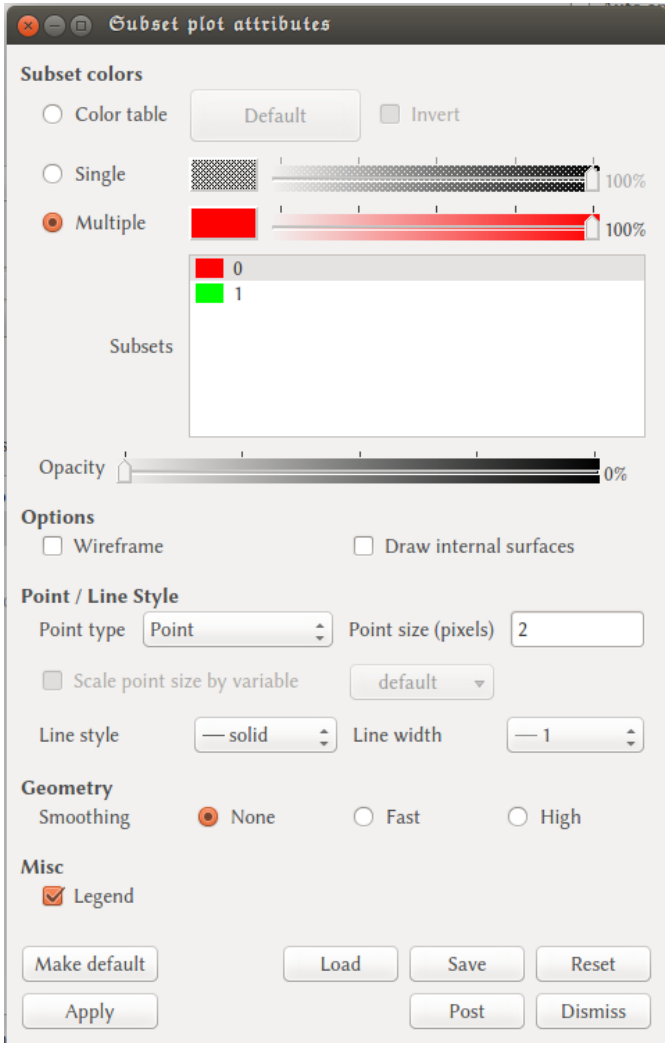
Plotting - **Subset**

Can plot either patches or levels, - **Add** $\rightarrow$ **Subset** $\rightarrow$ <**subset_variable**>





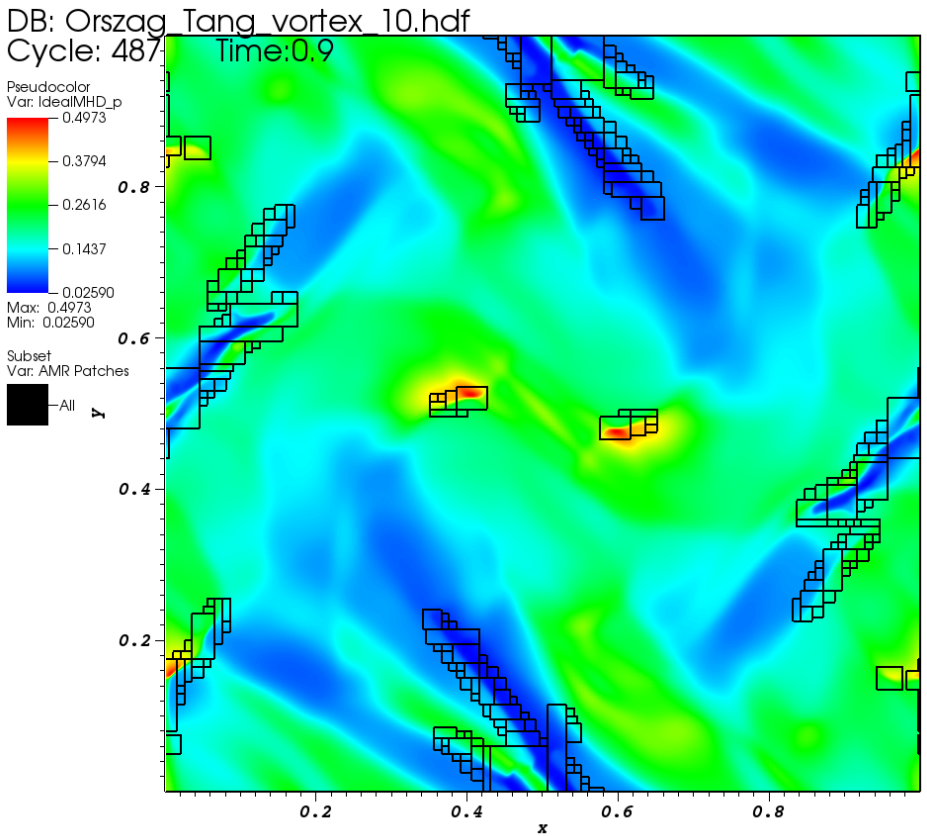
Plotting - **Subset** - attributes



- Most options here as seen previously
- **Options** - **wireframe** - this makes these plots more useful, instead of block of colour, the outlines of each patch/mesh are plotted

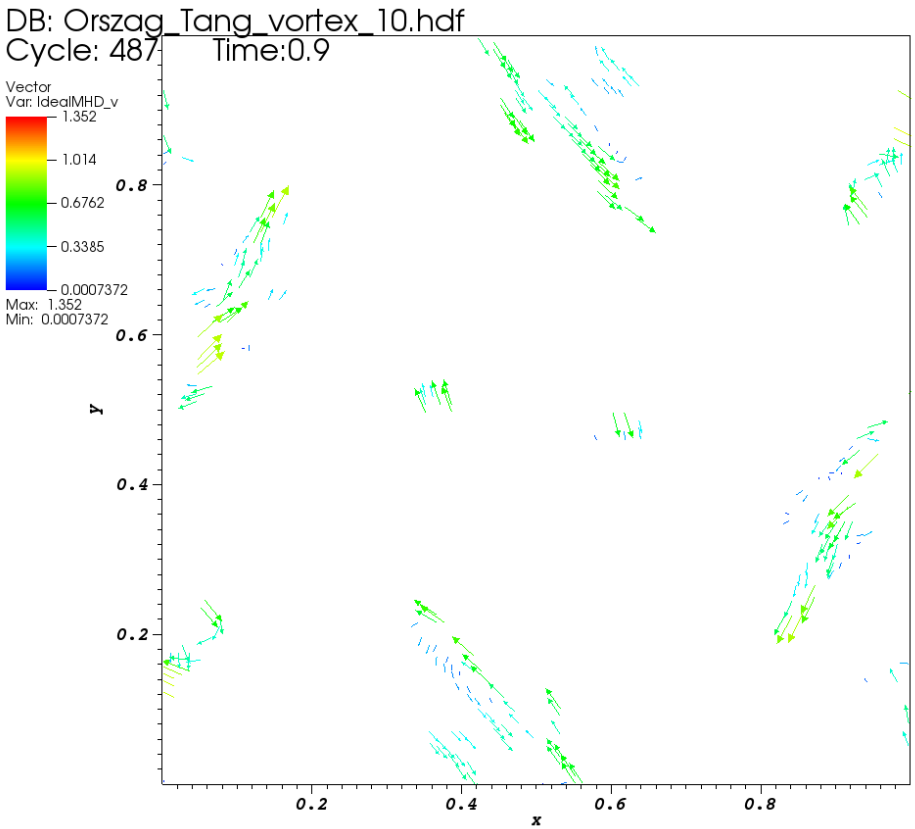
Plotting - **Subset**

This example shows a wireframe subset of the patches

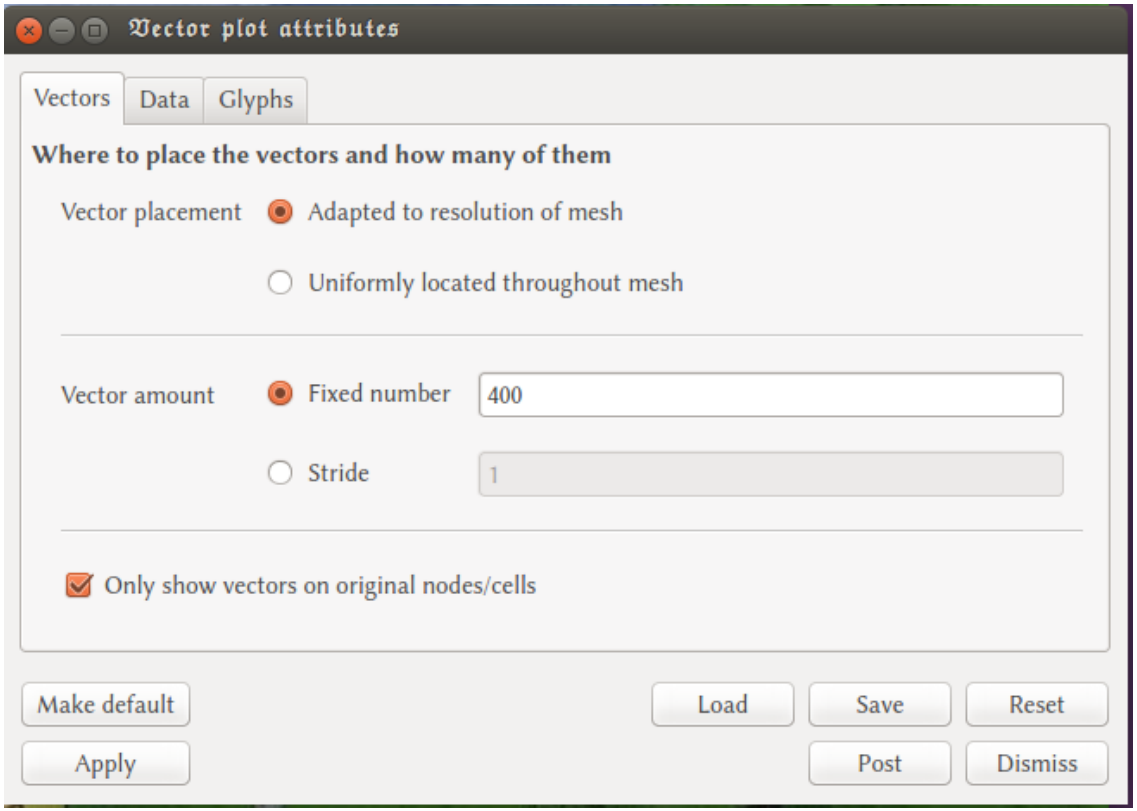


Plotting - **Vector**

VisIt can render vector variables, - **Add** → **Vector** → <**variable**>

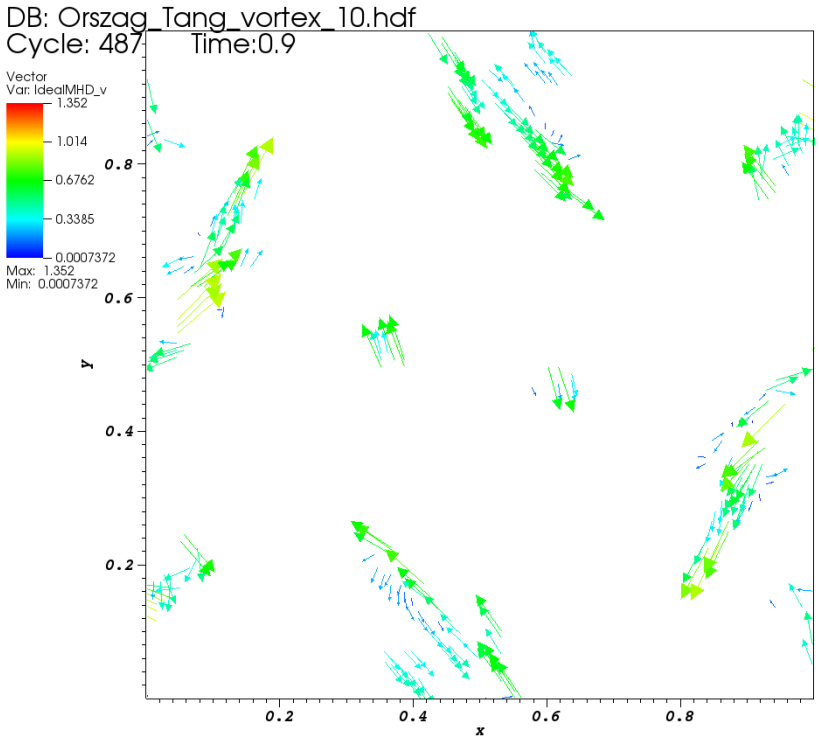


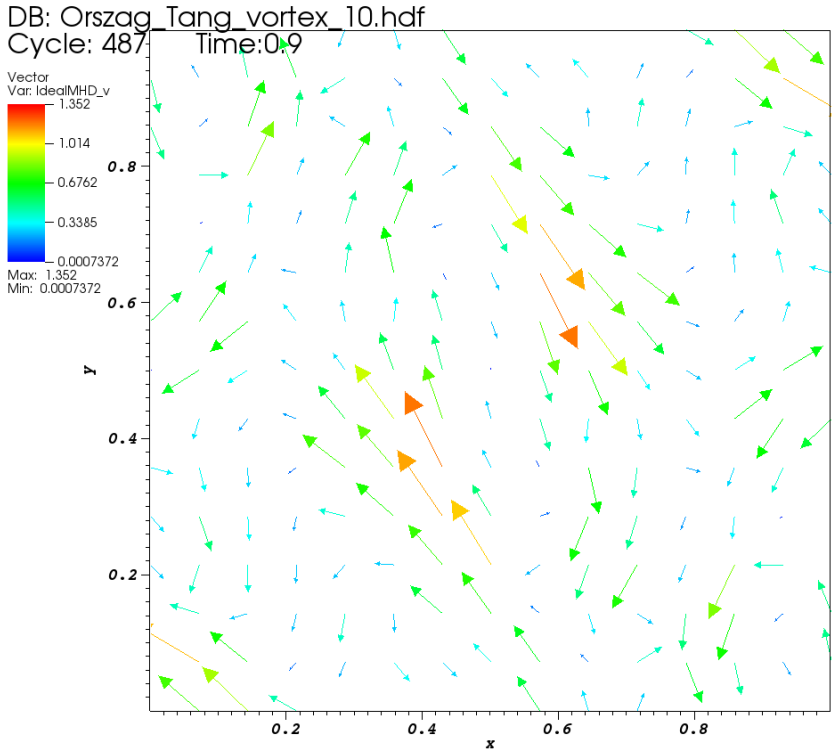
Plotting - **Vector** - attributes



- Vector plots typically need work actually be useful through control over their placement and quantity

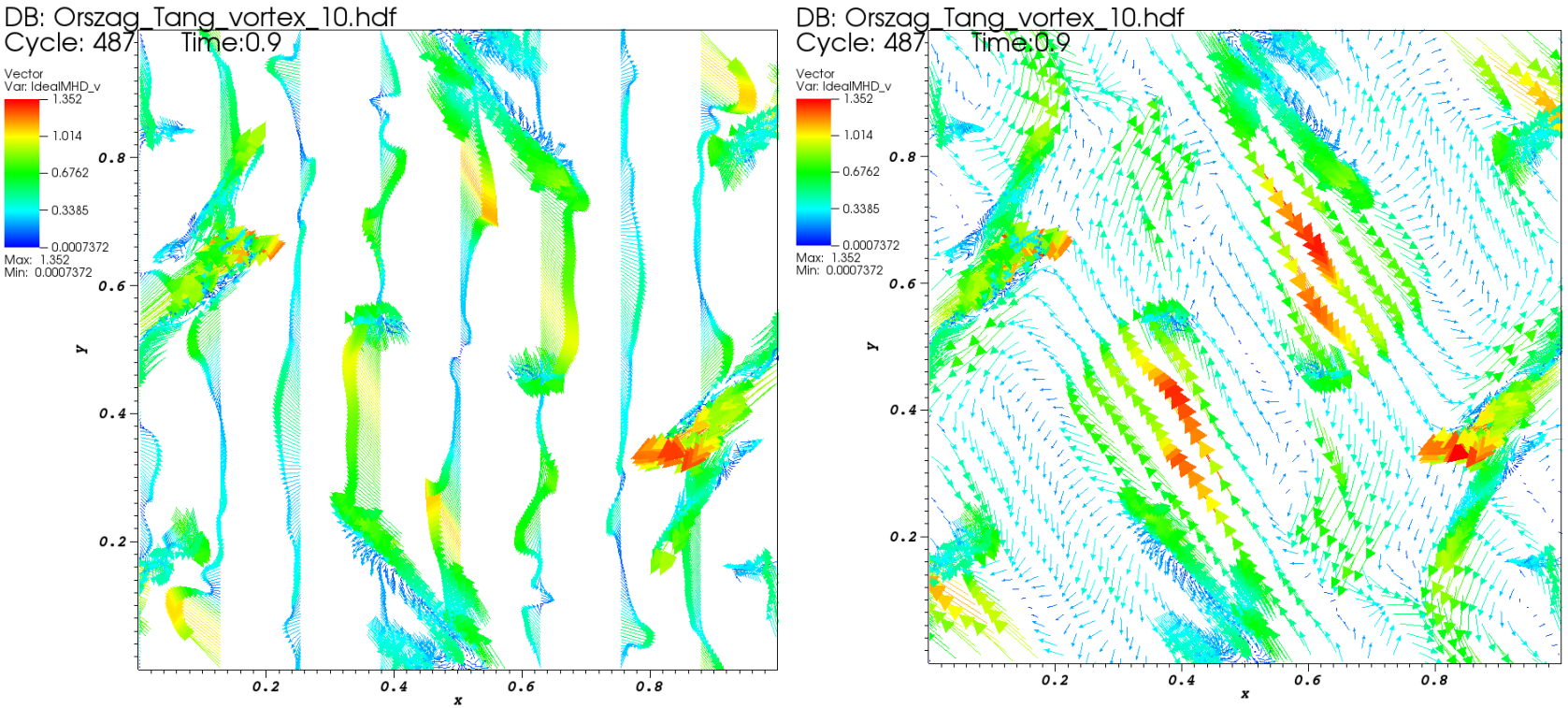
Plotting - **Vector** - attributes - placement





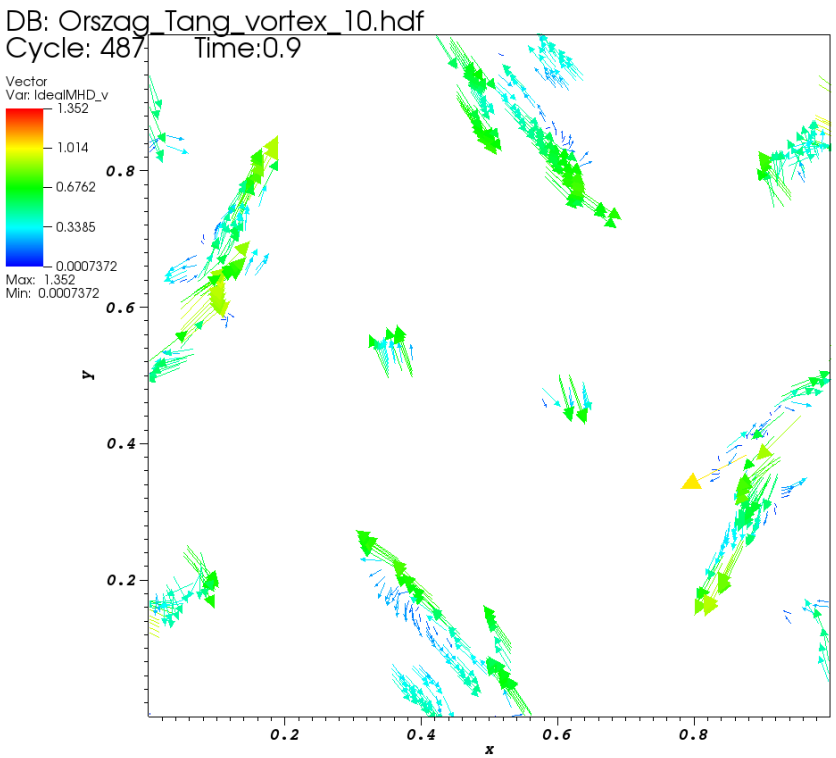
- Adaptive placement (default, left) prioritises placement on finer patches, uniform placement can look better as a result

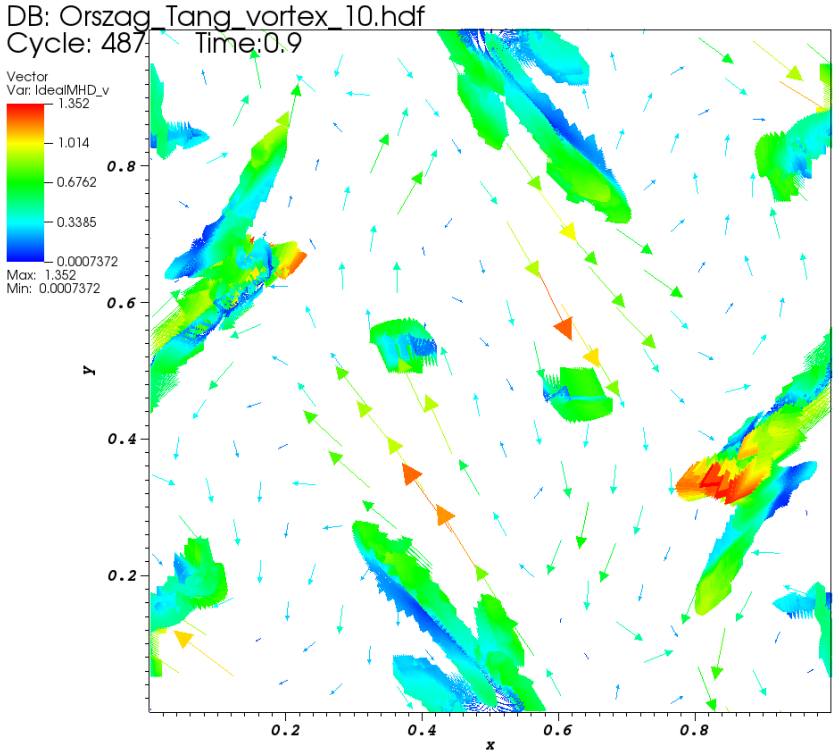
Plotting - **Vector** - attributes - stride



- Controls number of vectors in an adaptive setting by changing number of cells between each vector
- This can look odd if the resolution is a multiple of the stride (25 on left, 26 on right)

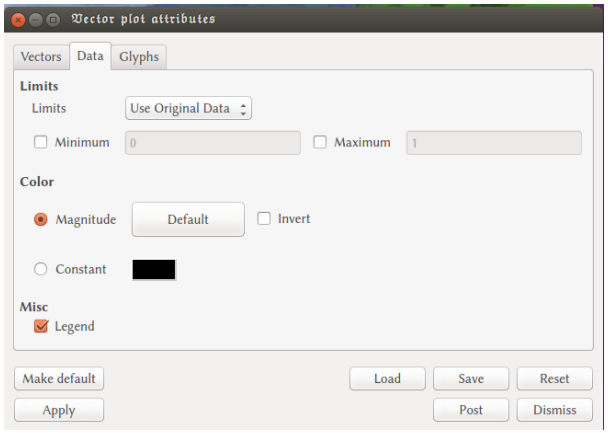
Plotting - **Vector** - attributes - fixed number

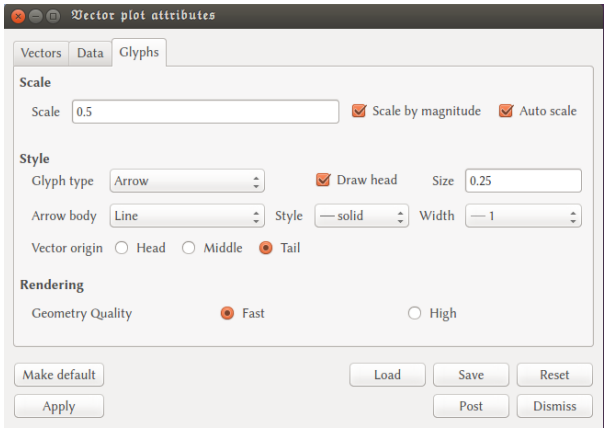




- Useful for uniformly placed vectors, not as effective as stride for adaptively placed vectors
- 600 vectors (left) and 60000 (right)

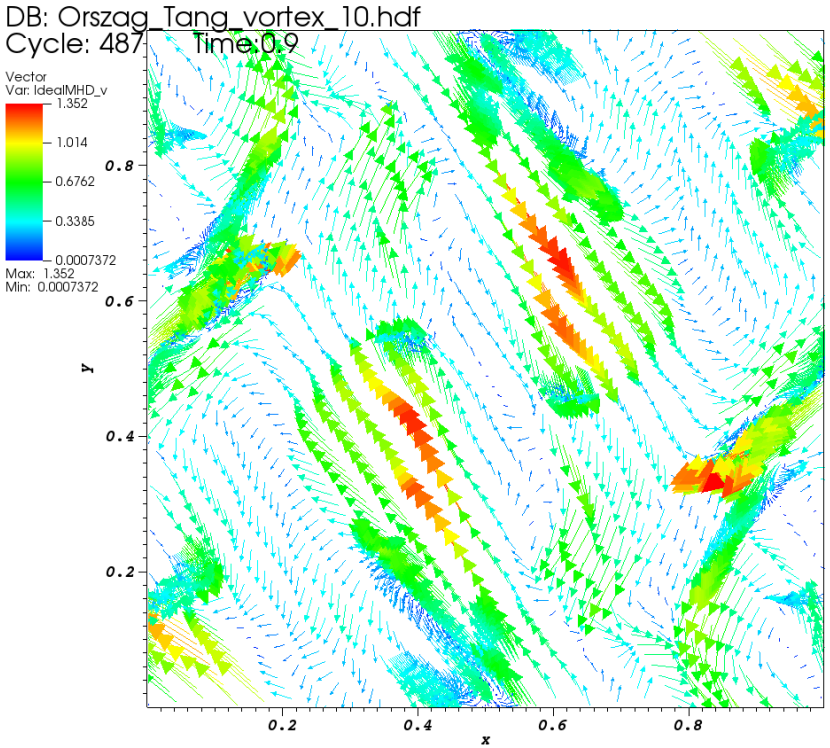
Plotting - **Vector** - attributes

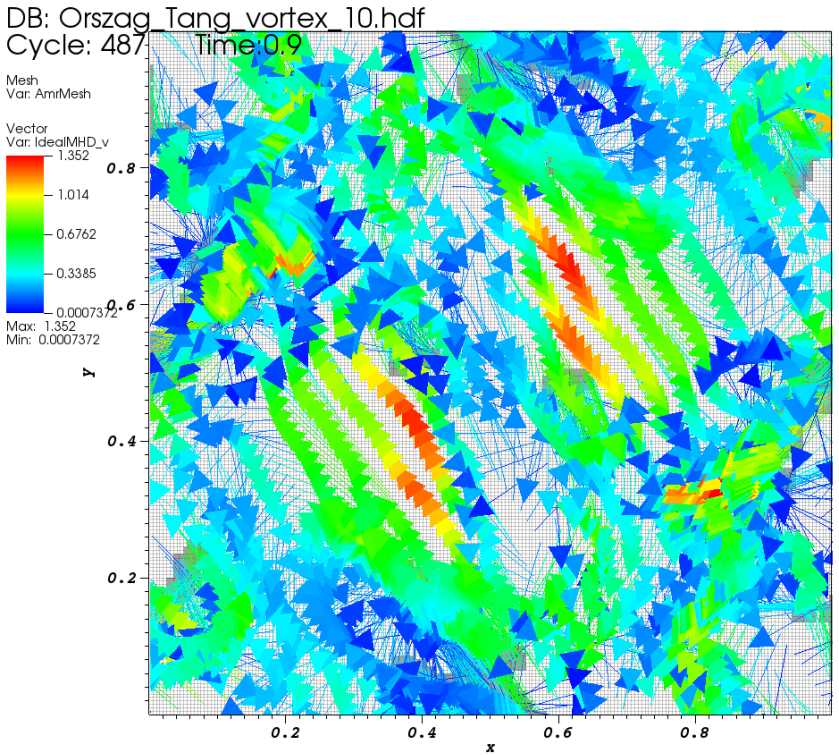




- The **Data** tab lets us control the range and colour of the vectors
- Colour is based on magnitude of the vector, though no error is given if e.g. a negative minimum is chosen
- The **Glyphs** tab controls the look of the plotted arrows
- **Style** is used to ensure vectors are thick enough to be legible on a plot
- **Scale** is a scaling factor for all the vector arrows - scaling by magnitude means large magnitude vectors appear longer

Plotting - **Vector** - attributes



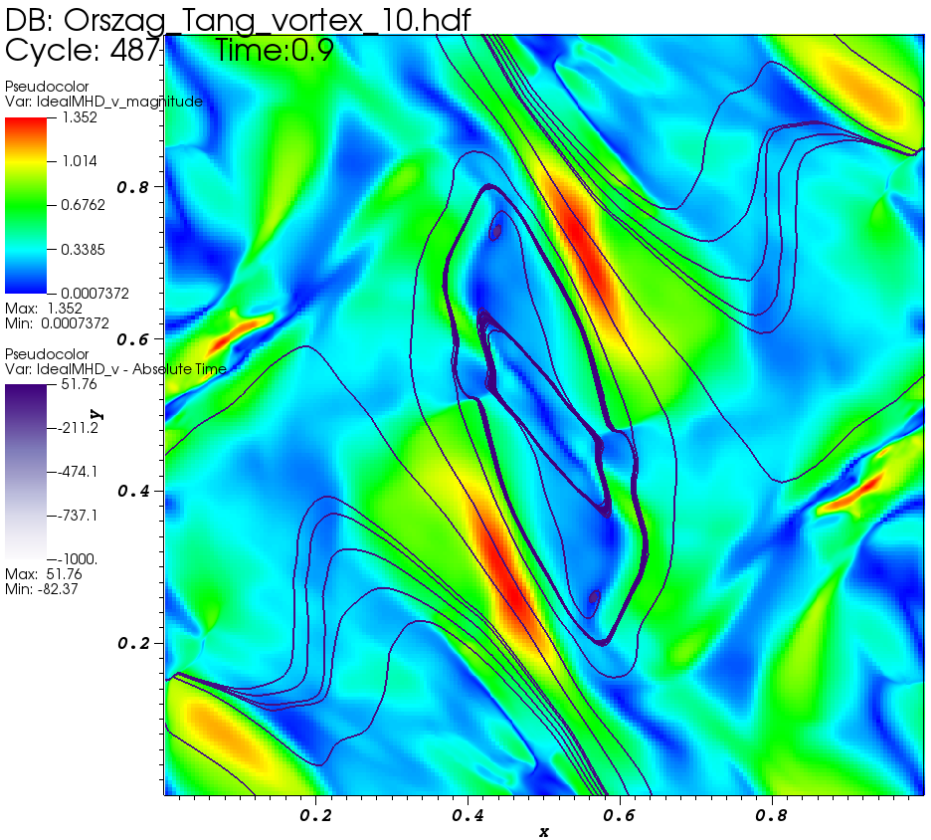


- Scaling by magnitude turned on (left) or off (right)
- However, if most vectors are small, scaling by magnitude might not be the right option (or needs combining with data limiting)

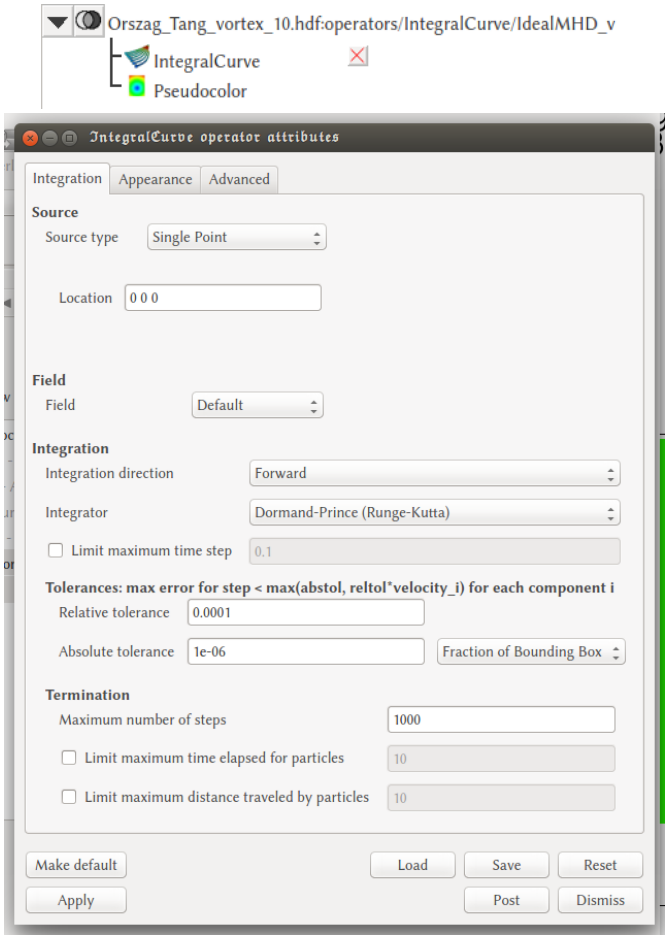
Plotting - **Streamlines**

Technically an operator, not a plot (in recent VisIt versions)

Add → **Pseudocolor** → **Operators** → **IntegralCurve** → <**variable**>



Plotting - **Streamlines** - attributes

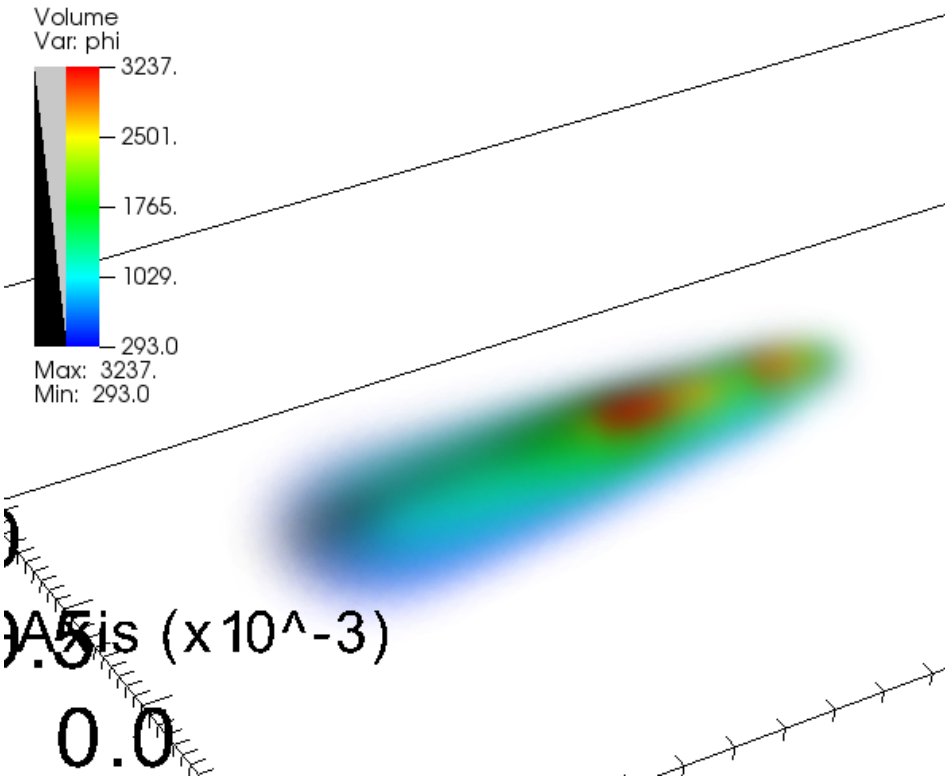


- Since it is an operator, the **IntegralCurve** attributes must be selected
- **Source** - how the initial points for the streamlines are seeded, e.g. along a line, circle or at a point
- **Sampling** - how many streamlines (default is 2)
- **Integration** - the direction and method can be chosen - direction **both** is often useful, unless you have an exact source - changing the method is not advised

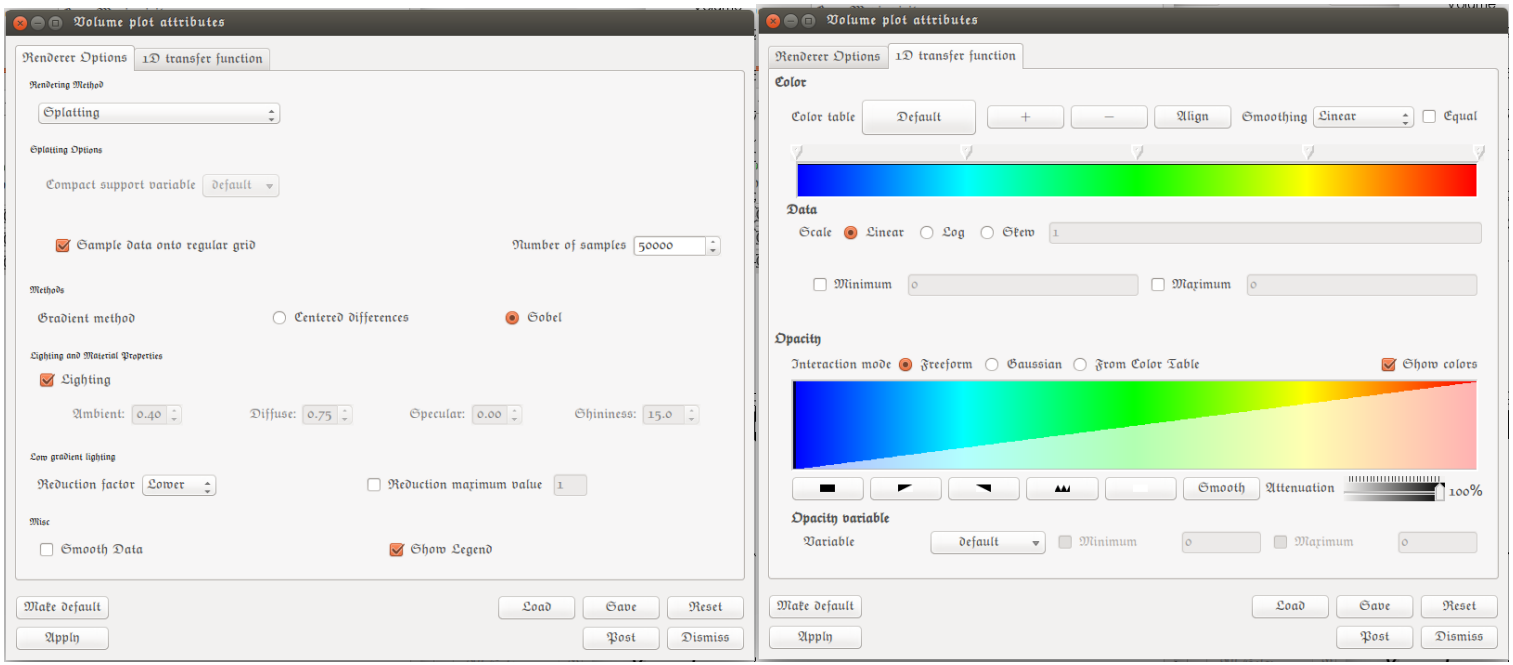
Plotting - **Volume**

Volume plots provide a smoothed rendering of 3D data, using opacity to show features

Add → **Volume** → <**variable**>



Plotting - **Volume** - attributes

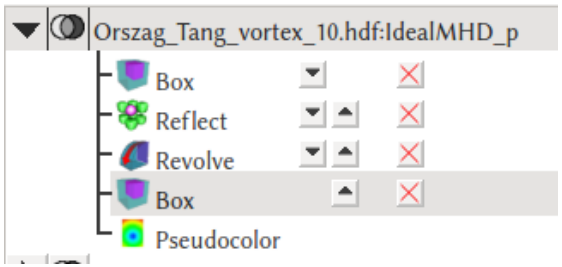


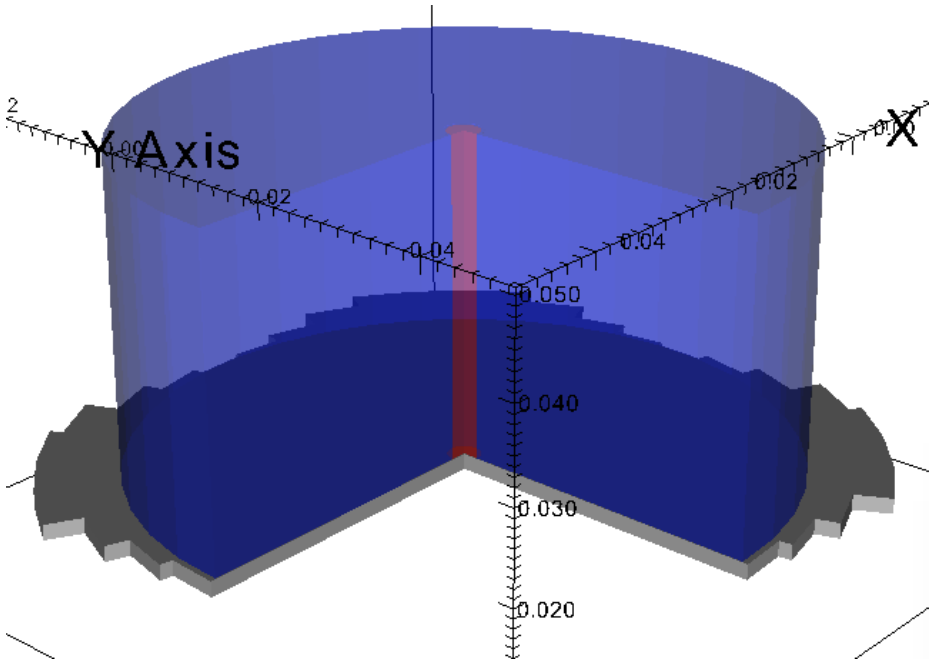
- **Rendering Method** controls how the plot is made, with computationally cheap options looking more smeared
- **1D transfer function** applies the opacity, with controls over the colour scheme (see later) and which colours are rendered opaque
- Counter-intuitive - the opaque colour on the **opacity** bar is the fully transparent colour in the plot

Outline

Operators

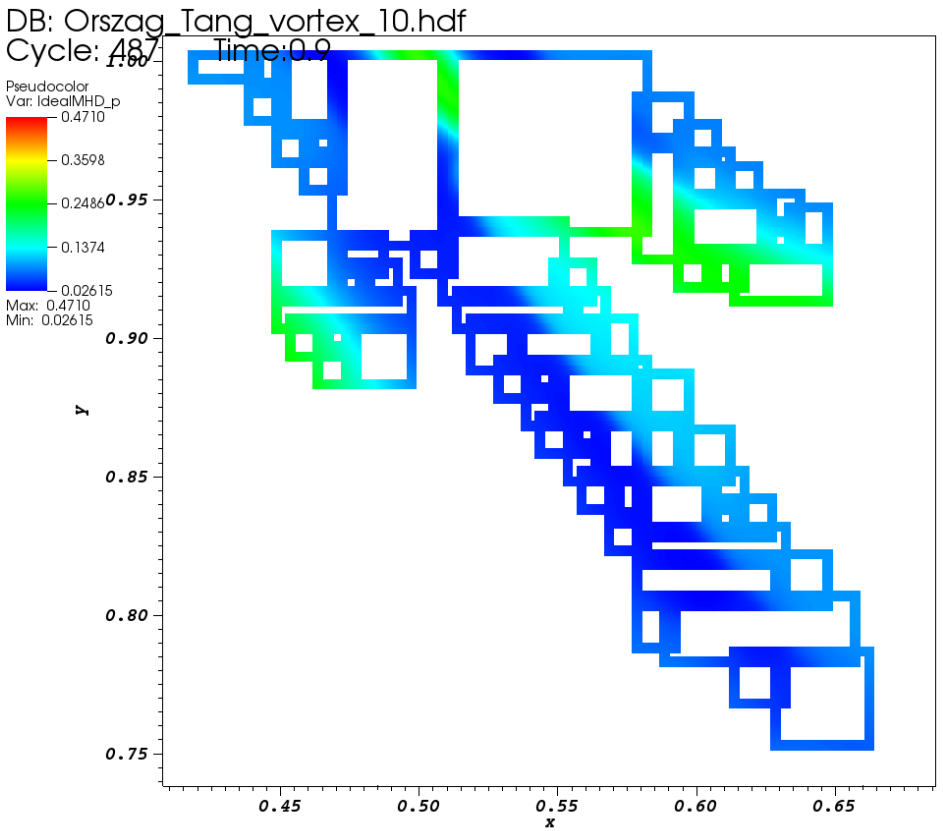
- Operators act on a plot to control how the data appears
- Multiple operators (even of the same type) can be applied
- Order of operation is important!





Operators - **Inverse Ghost Zone**

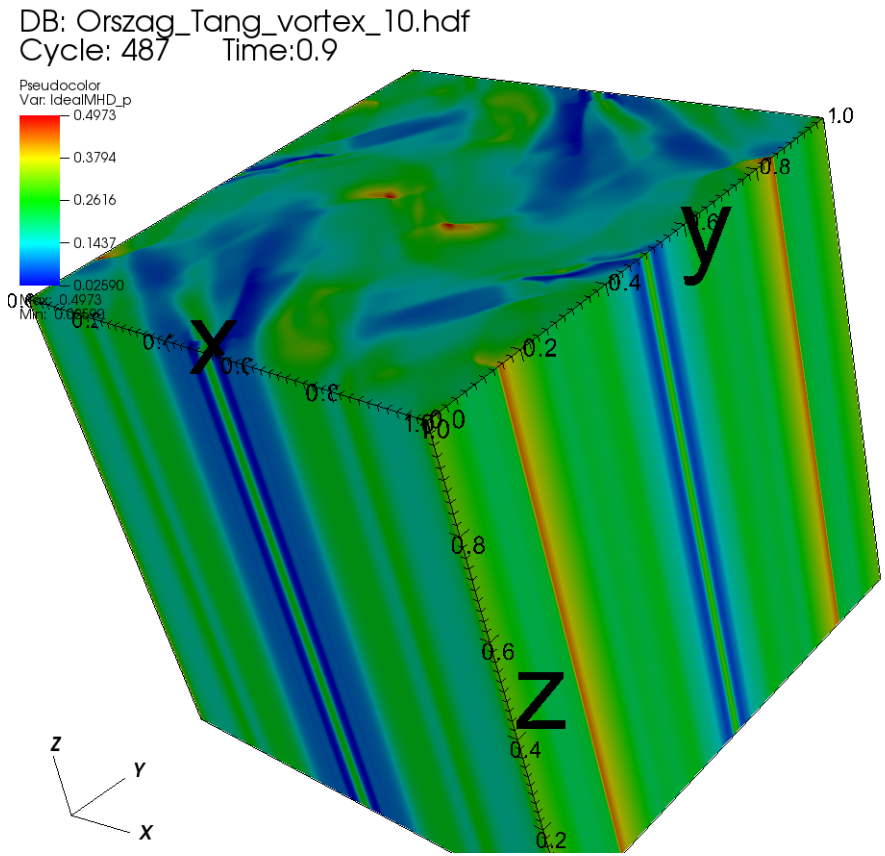
Shows the ghost zones of a patch - **Operators** → **Debugging** → **Inverse Ghost Zone**



Operators - **Extrude**

Extends a 2D plot along a given axis to act as a 3D plot

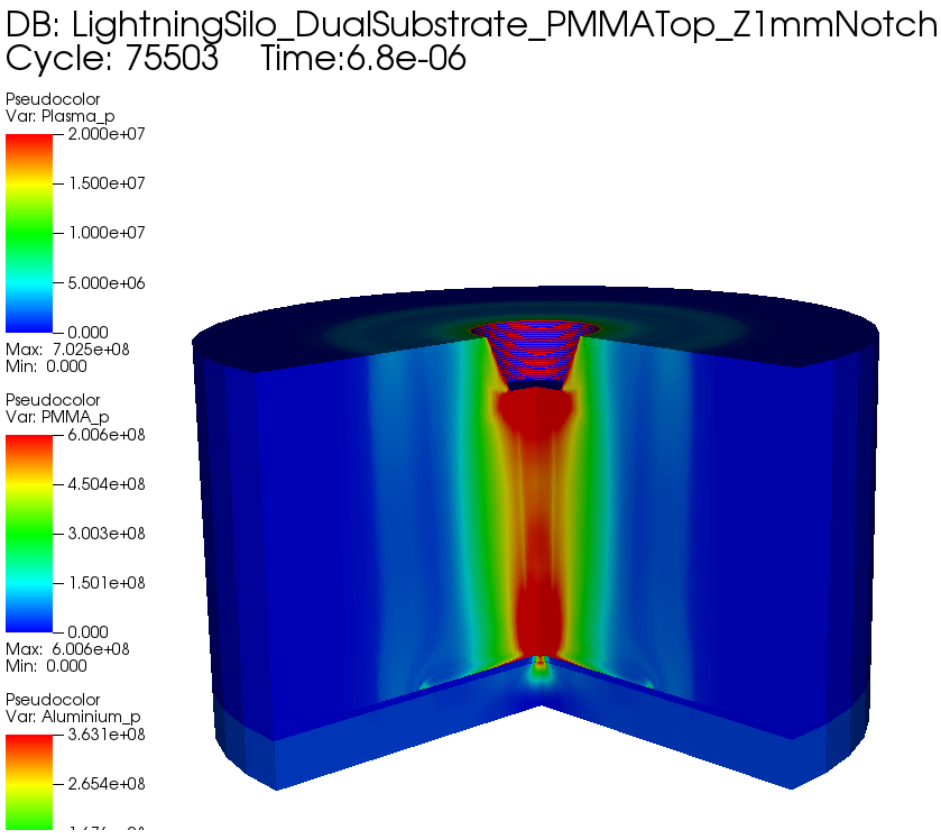
Operators → **Geometry** → **Extrude**



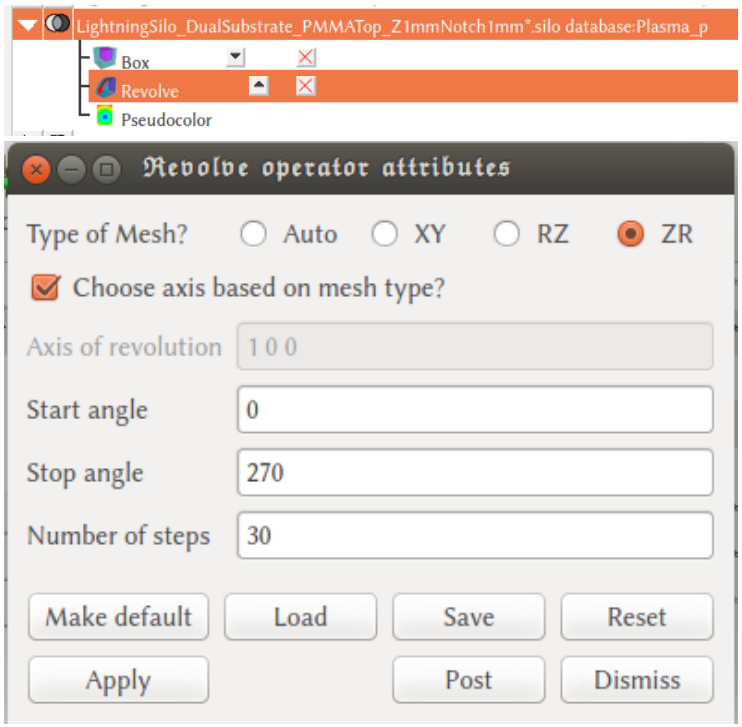
Operators - **Revolve**

Revolves a plot about a given axis

Operators → **Geometry** → **Revolve**



Operators - **Revolve** - attributes

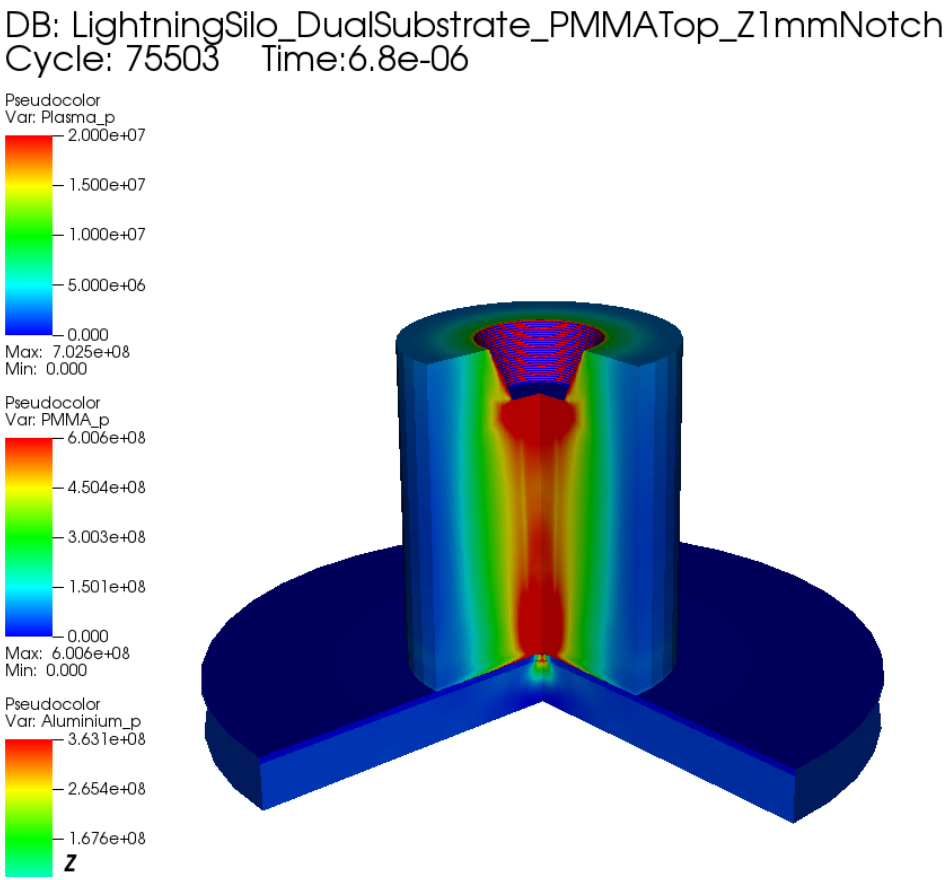


- **Type of Mesh?** will attempt to automatically revolve a plot based on the mesh type, e.g. an r-z cylindrical plot is revolved through the **ZR** option
- Alternatively, an arbitrary axis can be chosen
- **Start angle** and **Stop angle** allow for partial revolving
- **Number of steps** - each 'step' is effectively a triangular wedge. Low numbers have visible boundaries, but render quickly, 100-300 is suitable for a revolution that looks right

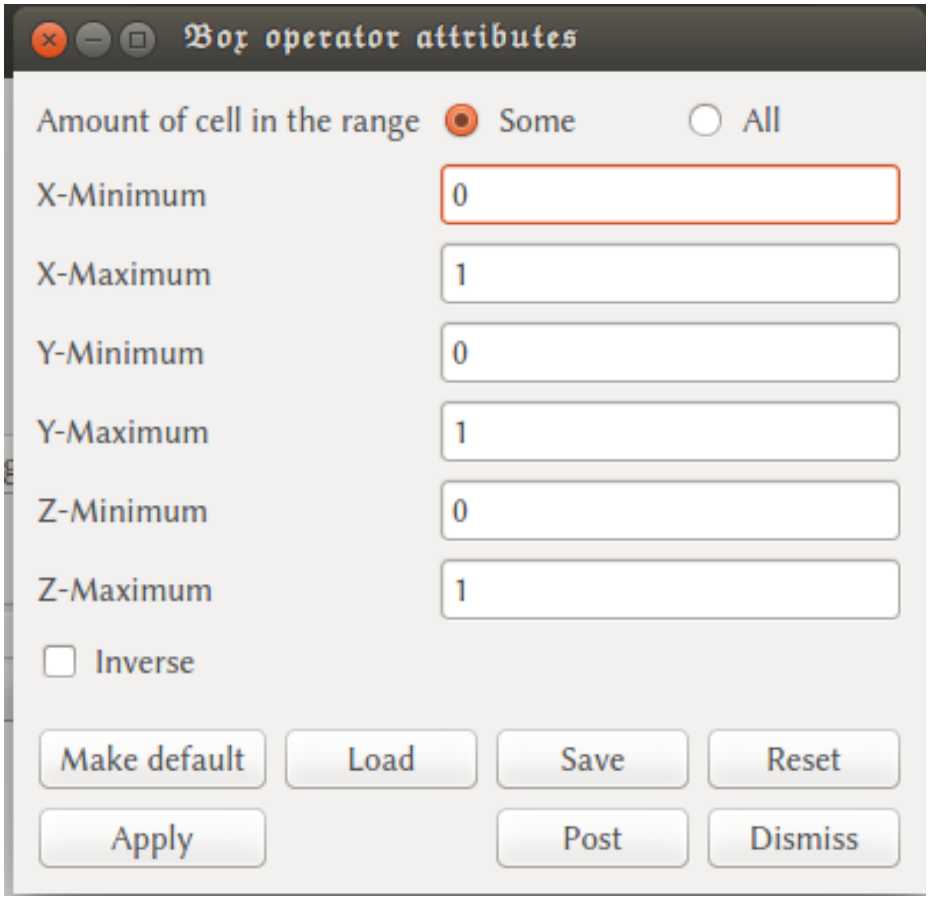
Operators - **Box**

Only plots material within the domain of a given box

Operators $\rightarrow$ **Selection** $\rightarrow$ **Box**



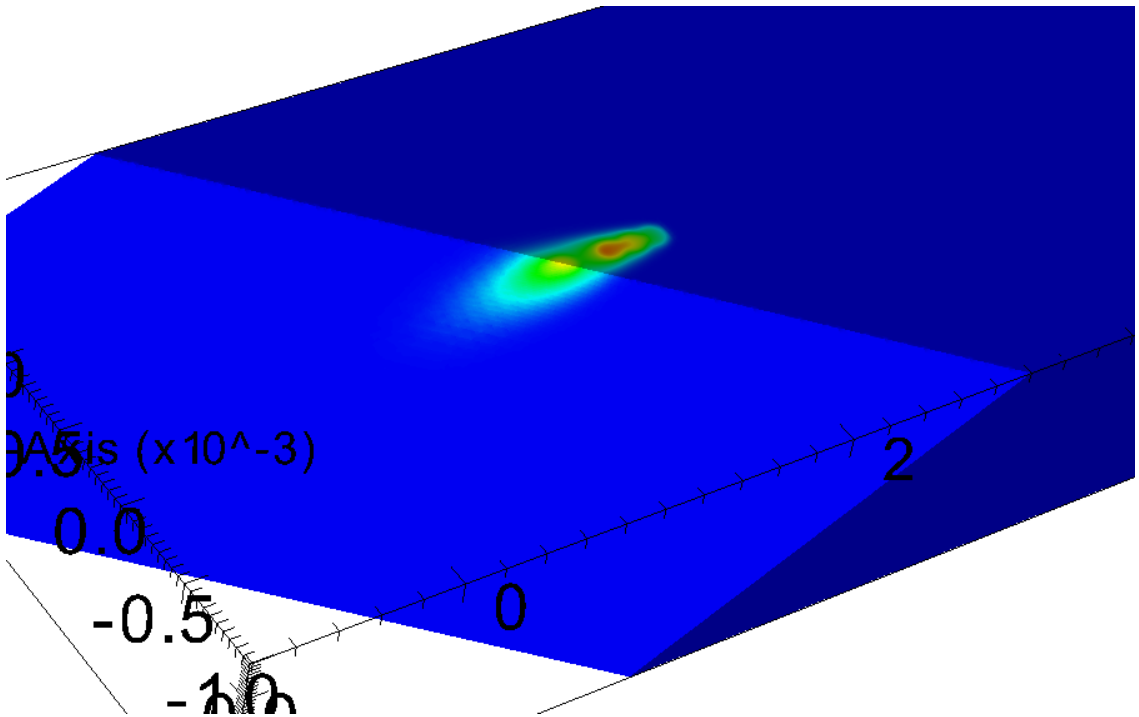
Operators - **Box** - attributes



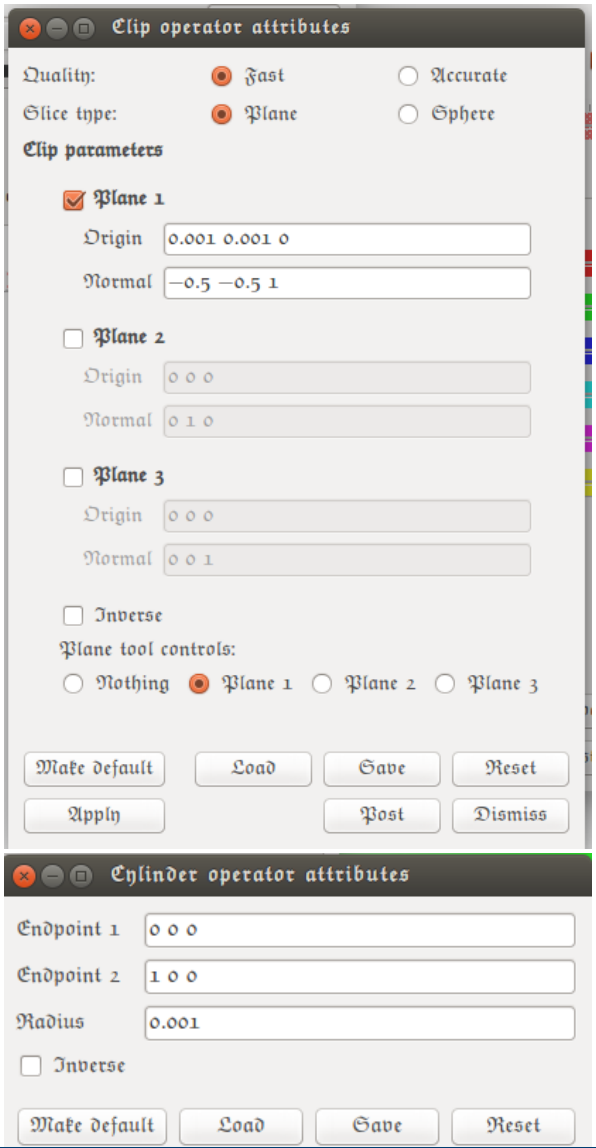
- **Amount of cell in the range** chooses how the edges of the box are defined, only really visible at low resolution
- **Extents** - Based on physical coordinates of the data, in 2D, z-extents can be ignored
- **Inverse** - only the area *outside* the given box is plotted

Operators - **Clip**, **Cylinder**

Two operators that allow for a region of the plot to be 'cut away'
Operators $\rightarrow$ **Selection** $\rightarrow$ **Clip**
Operators $\rightarrow$ **Selection** $\rightarrow$ **Cylinder**



Operators - **Clip**, **Cylinder** - attributes

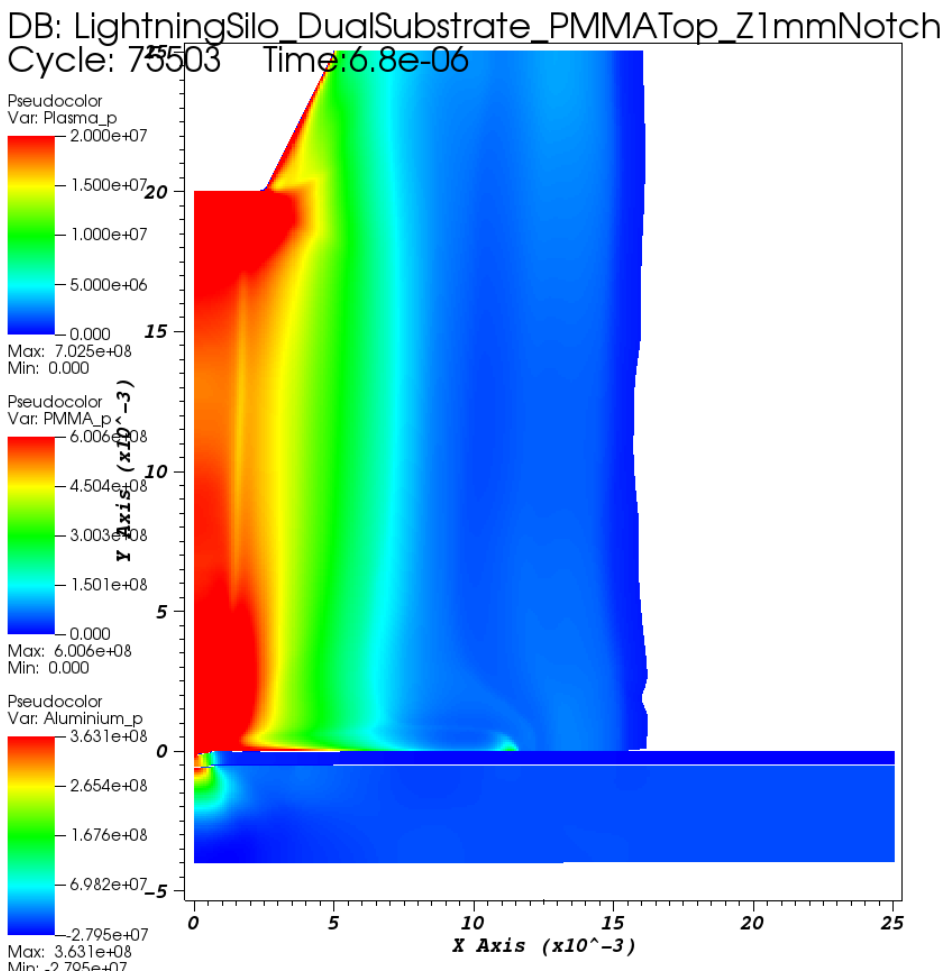


- **Slice type** - up to three different planes, or a single sphere
- **Clip parameters** - planes are defined through origin and normal, spheres through origin and radius
- **Inverse** - as with **Box**, inverts what is removed
- **Plane control tools** - The top of the plot window has a 'plane control' button, which can dynamically move the axes of a single selected plane
- Cylinders are defined through **Endpoint**s and a **Radius**

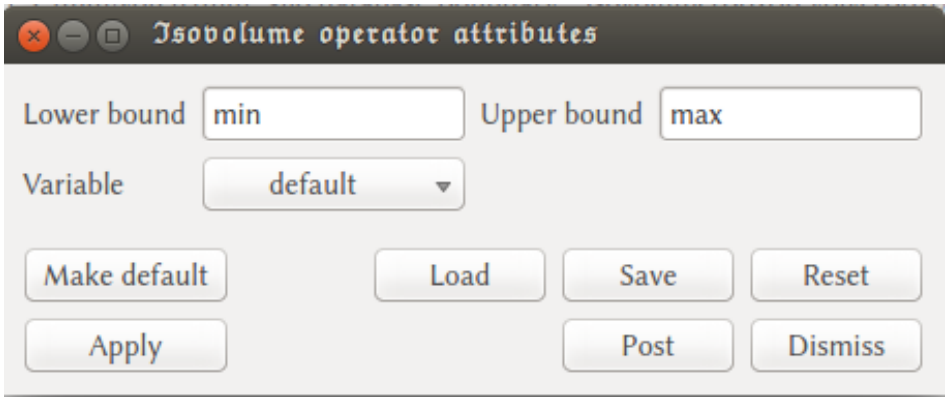
Operators - **Isovolume**

Allows a plot to be limited based on the extents of a variable

Operators → **Selection** → **Isovolume**



Operators - **Isovolume** - attributes

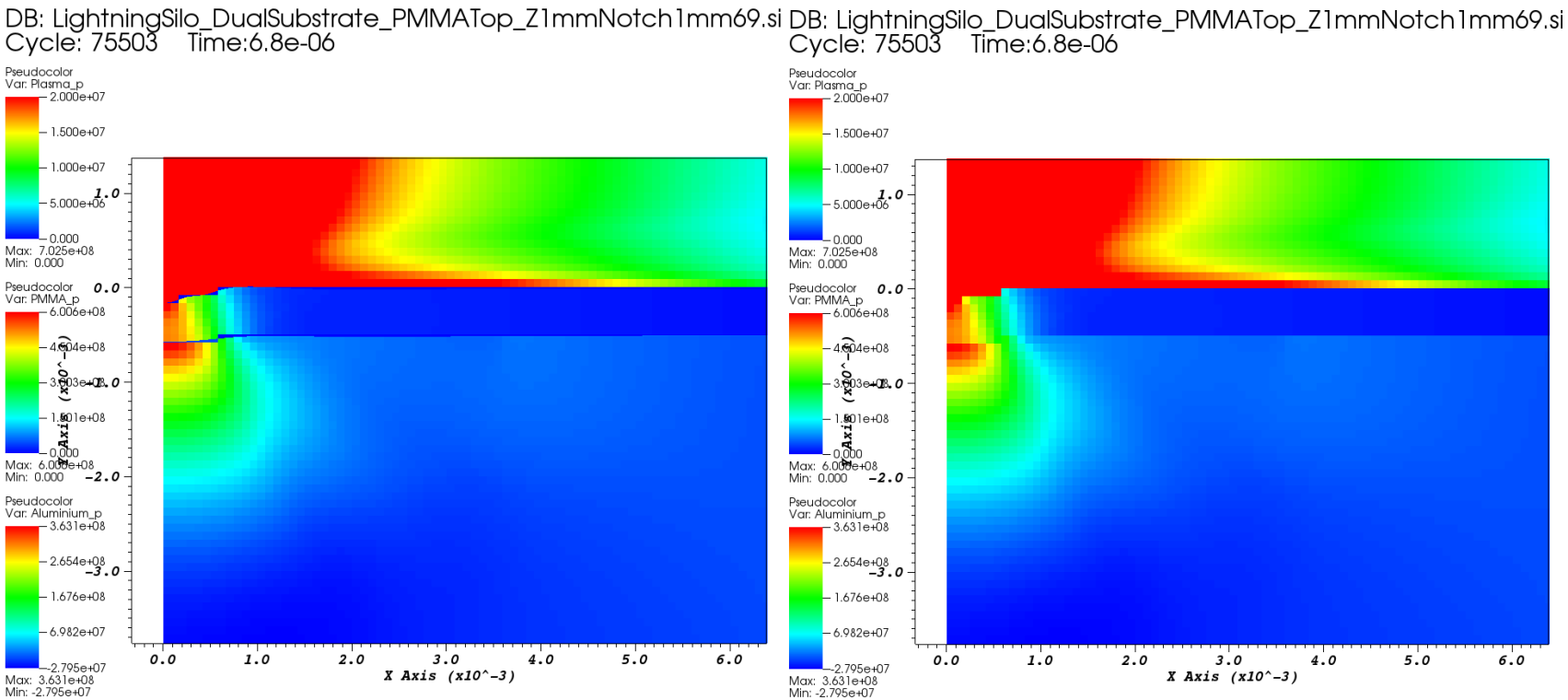


- A single **Variable** can be selected, **default** is the variable in the plot itself
- **min** and **max** are default values (±double precision limits)

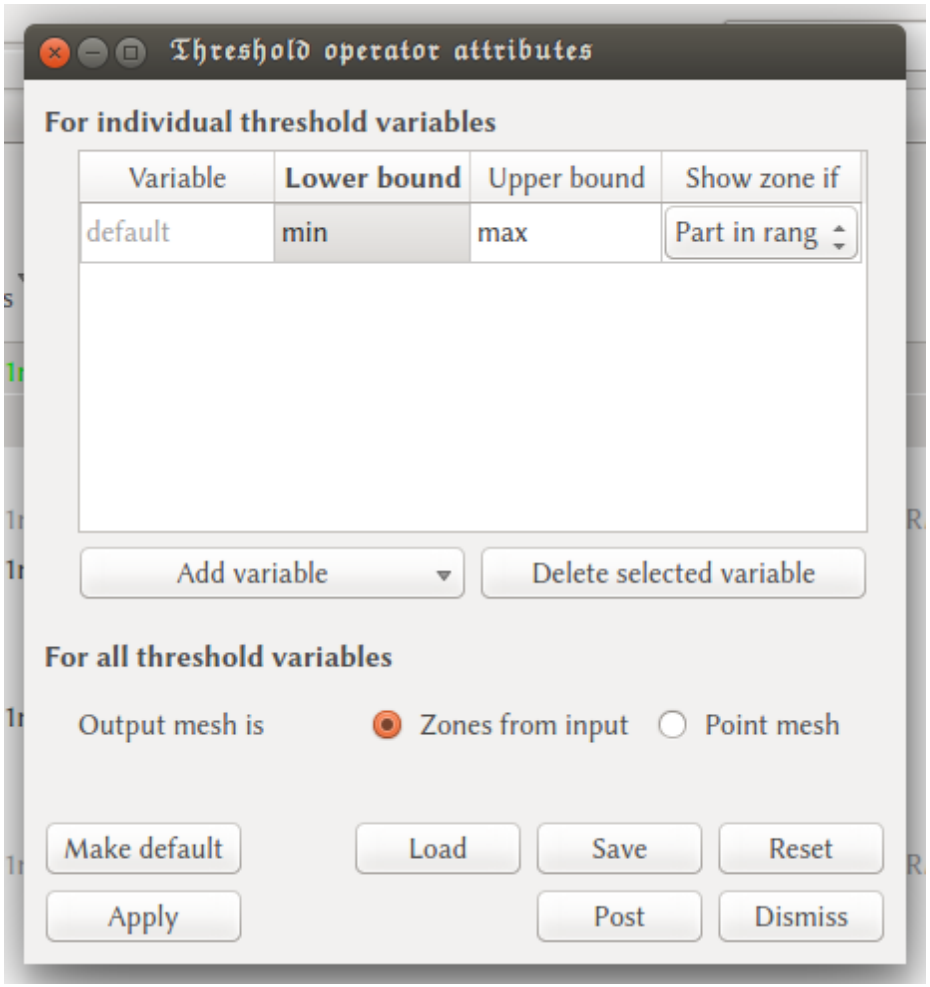
Operators - **Threshold**

Threshold (right) is similar to isovolume (left) with different interpolation

Operators → **Selection** → **Threshold**

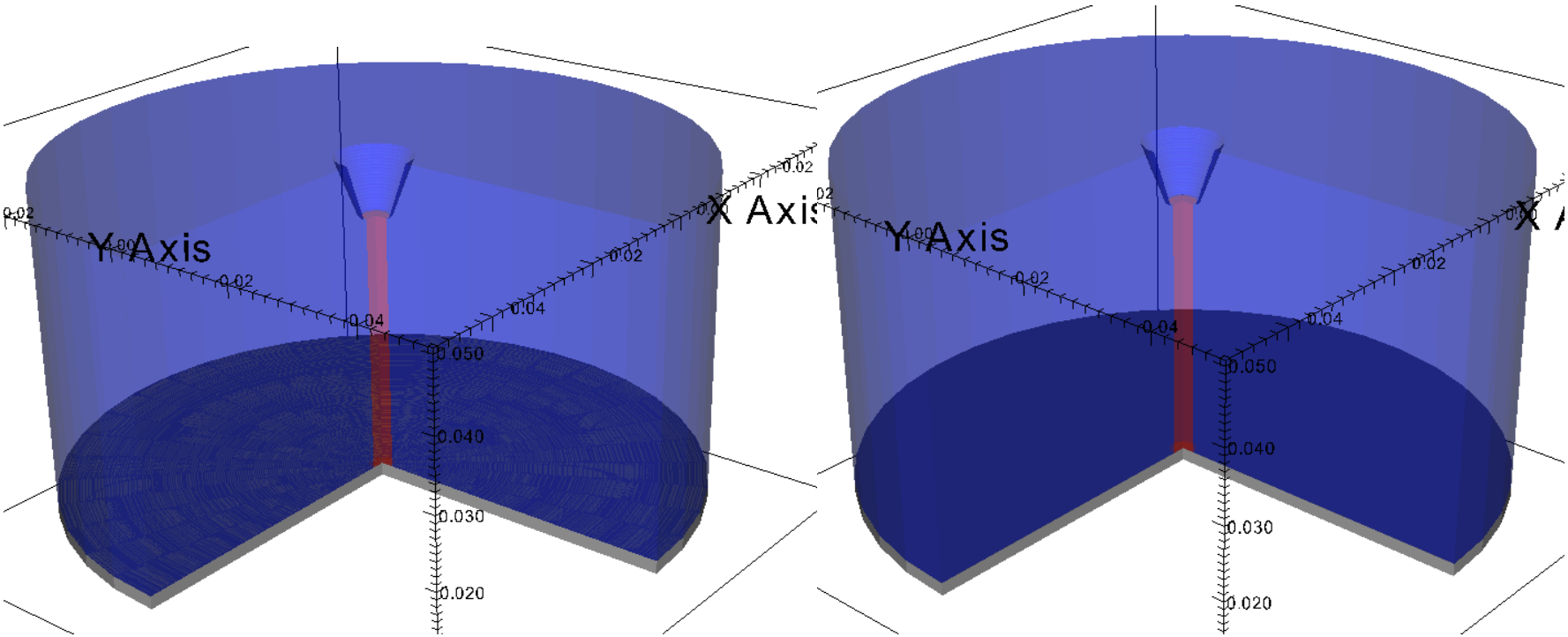


Operators - **Threshold** - attributes



- The threshold selection is similar to isovolume, in this case, clicking 'min' or 'max' allows these numbers to be edited
- Additionally, a single threshold operator can use multiple variables to apply the limits, through **Add variable**
- As with the **Box** operator, the selection criteria can consider either partial or fully filled grid cells

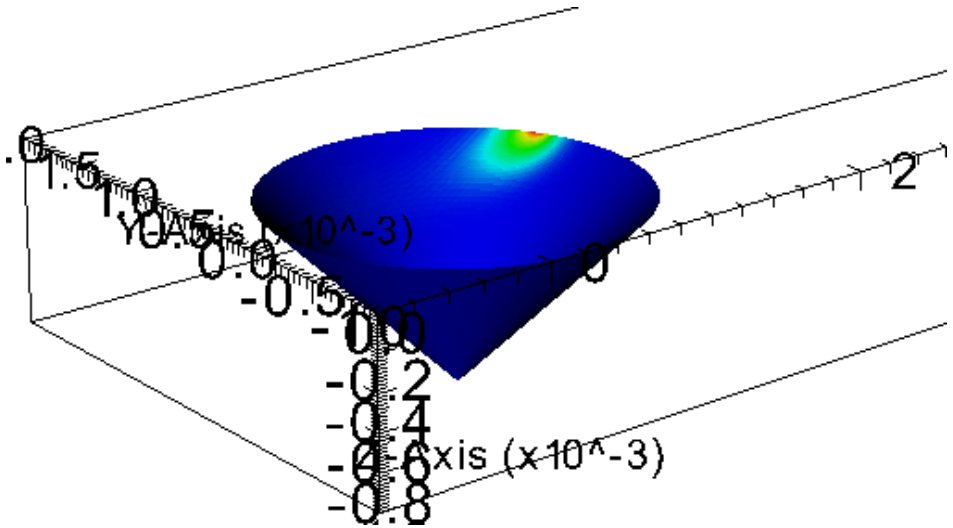
Operators - **Threshold**/**Isovolume** with **Revolve**



- Sometimes, especially when using transparency, visual artefacts appear
- The left plot shows material selected only through the **Subset** button, the right has an isovolume operator on the level set function at $\phi = 1 \times 10^{-4}$

Operators - **Cone**, **Ellipsoid**, **Spherical**

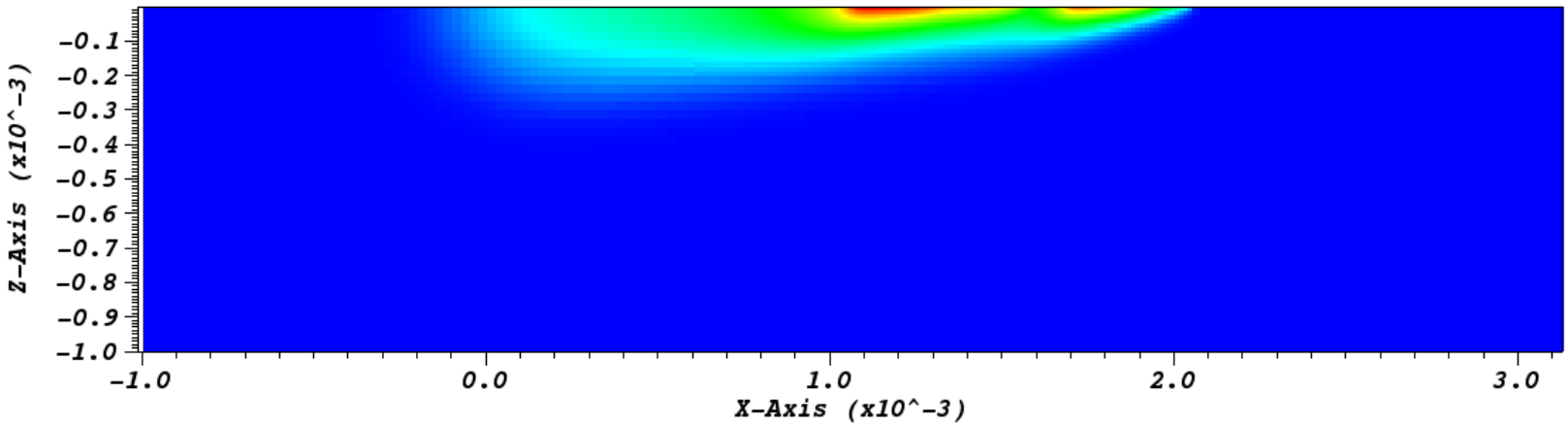
All three produce a 2D surface from 3D data, e.g.
Operators → **Slicing** → **Cone**



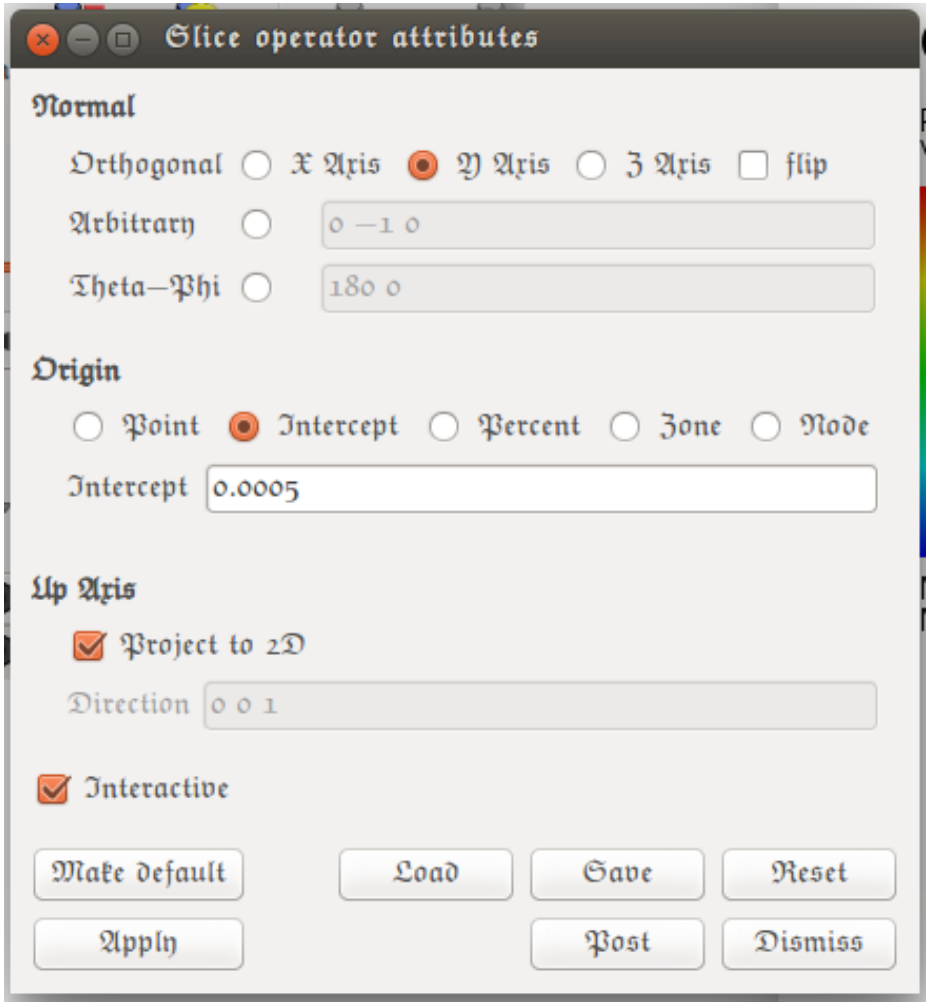
Cones can be projected to 2D, ellipsoids and spheres cannot

Operators - **Slice**

This takes a 2D slice from 3D data, and can project it to 2D
Operators $\rightarrow$ **Slicing** $\rightarrow$ **Slice**



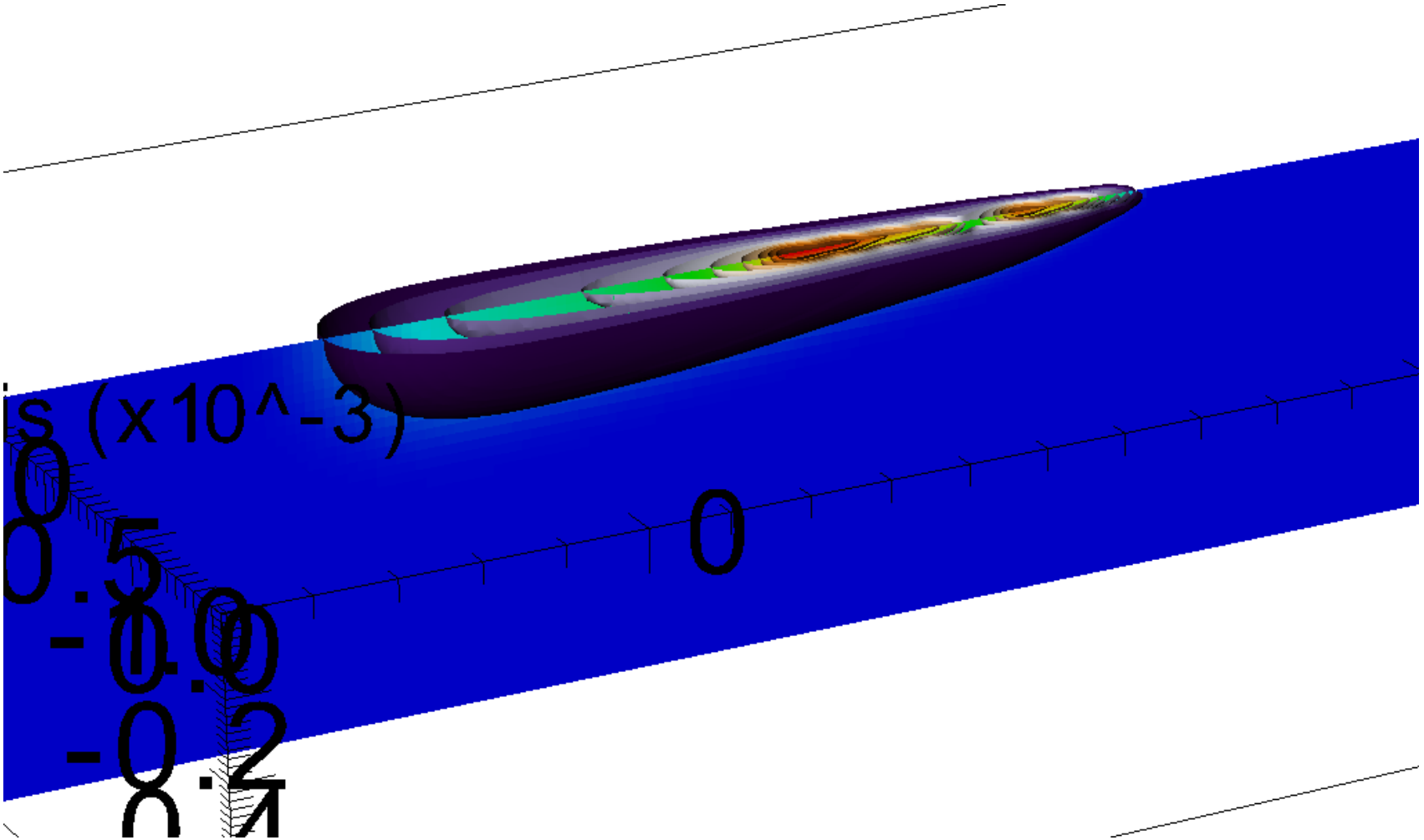
Operators - **Slice** - attributes



- **Normal** - the three Cartesian directions are available, otherwise either a vector or a rotation can be specified
- **Origin** - **Point** is a point which cuts the domain, useful for arbitrary normals, **Intercept** is a single point containing the slice, e.g. the origin in rotated case, or the position on the cut axis
- **Project to 2D** can be turned off, in which case the slice appears as part of a 3D plot
- **Direction** is used to project a non-orthogonal slice to 2D

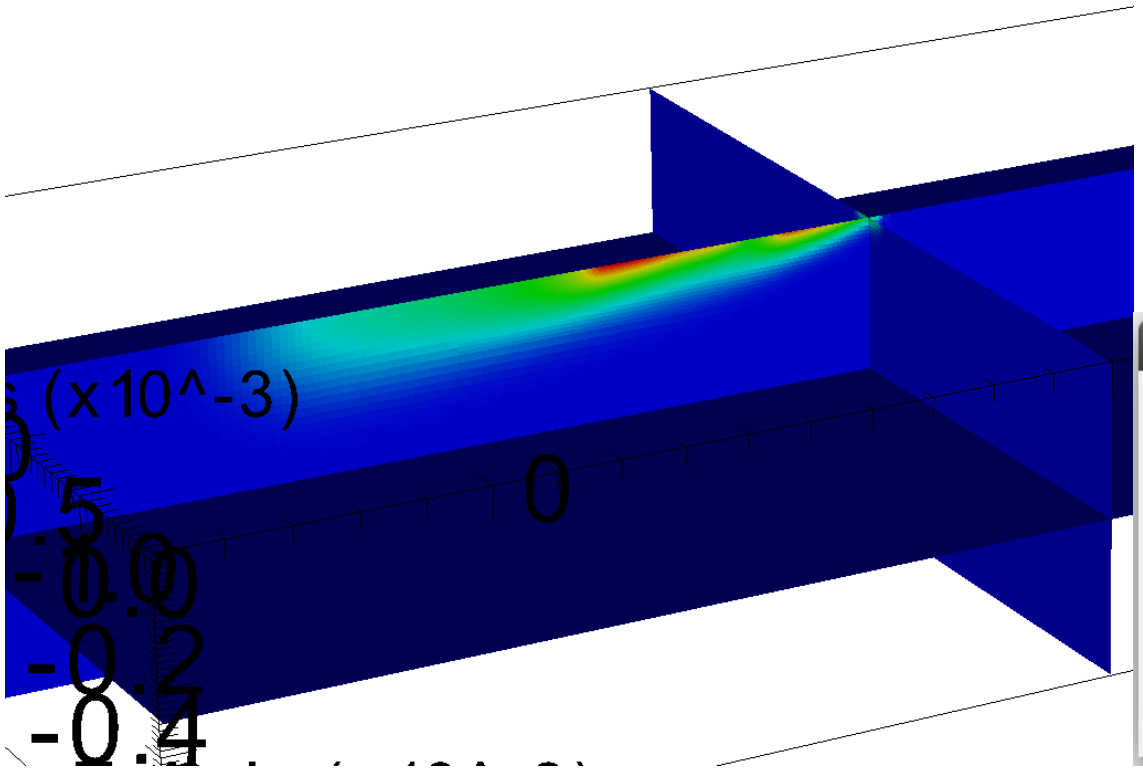
Operators - **Slice**

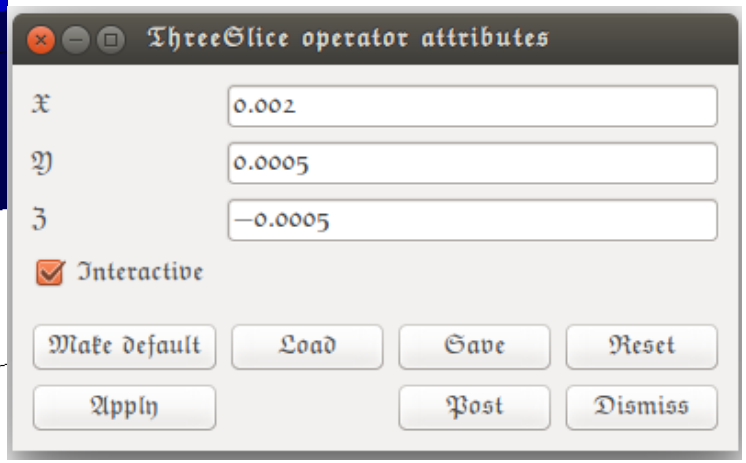
Example of a (non-projected) slice used in conjunction with contours



Operators - **ThreeSlice**

Three slices, in the x-, y- and z-planes
Operators $\rightarrow$ **Slicing** $\rightarrow$ **ThreeSlice**

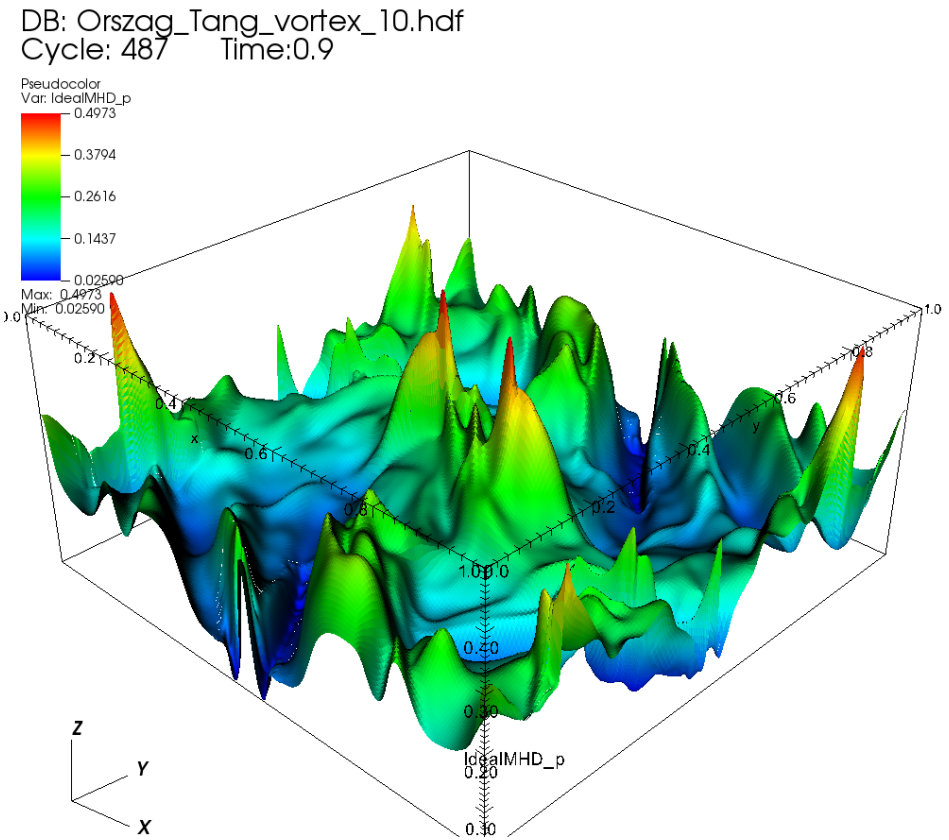




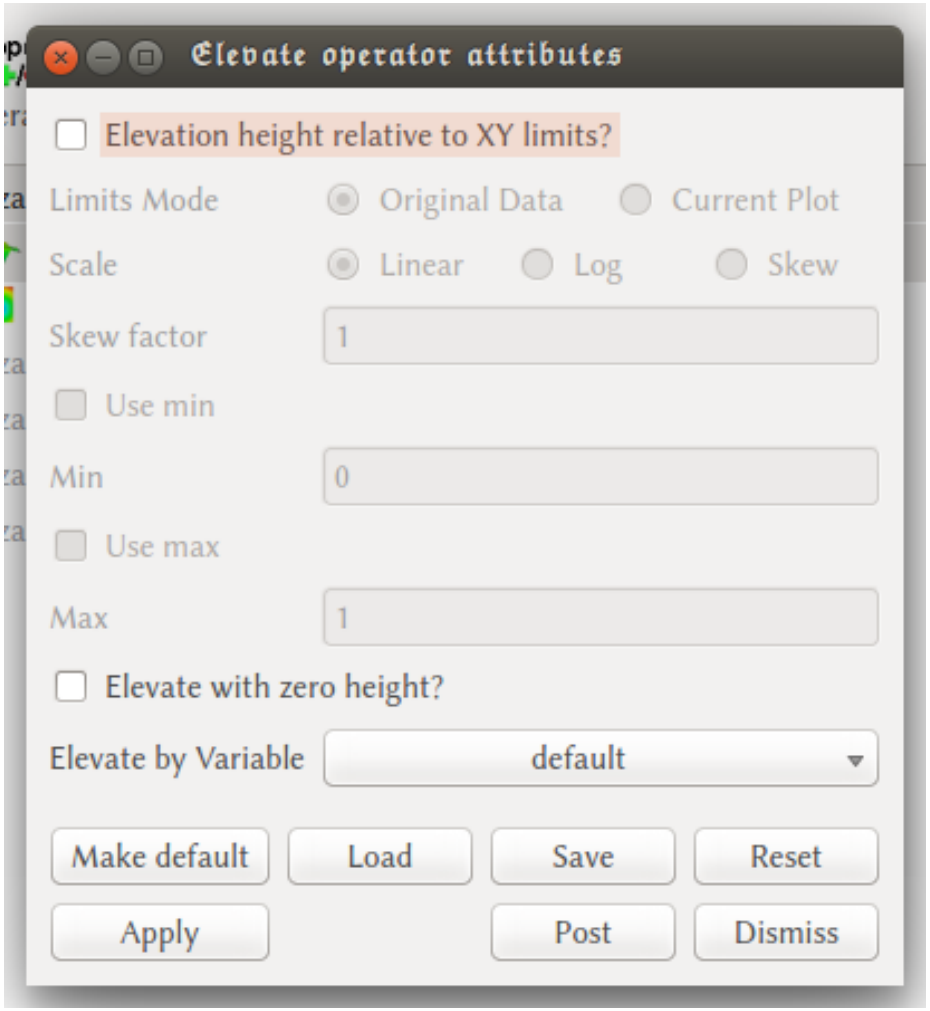
Operators - **Elevate**

Creates a 3D surface of 2D data

Operators → **Transforms** → **Elevate**



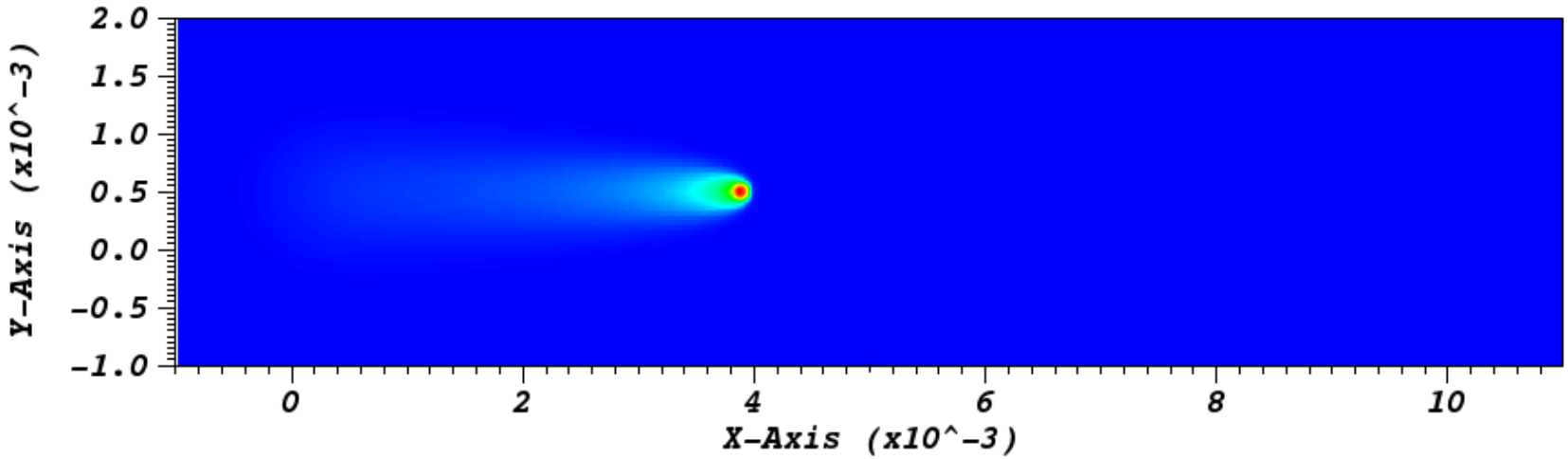
Operators - **Elevate** - attributes



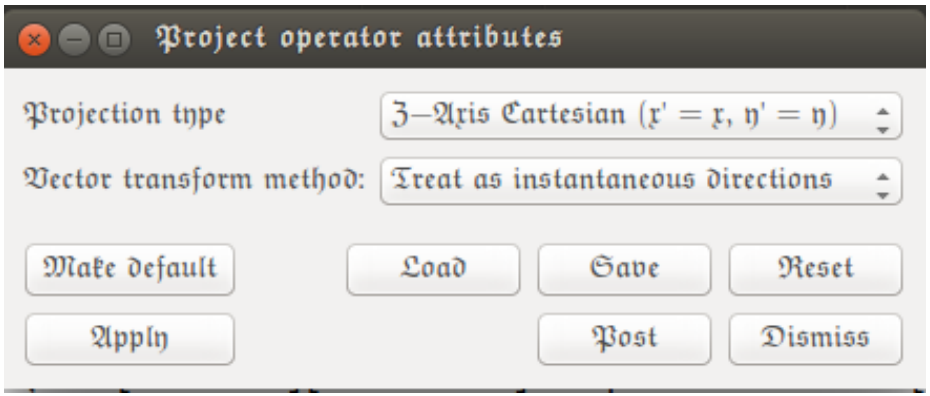
- Control over the way elevation is handled is used to scale the surface heights to make features clear
- **Elevate with zero height?** effectively converts a 2D plane into a 3D sheet (opposite of **Slice**), so 2D data can be plotted with 3D data
- **Elevate by Variable** means the plotted variable does not have to control the surface height

Operators - **Project**

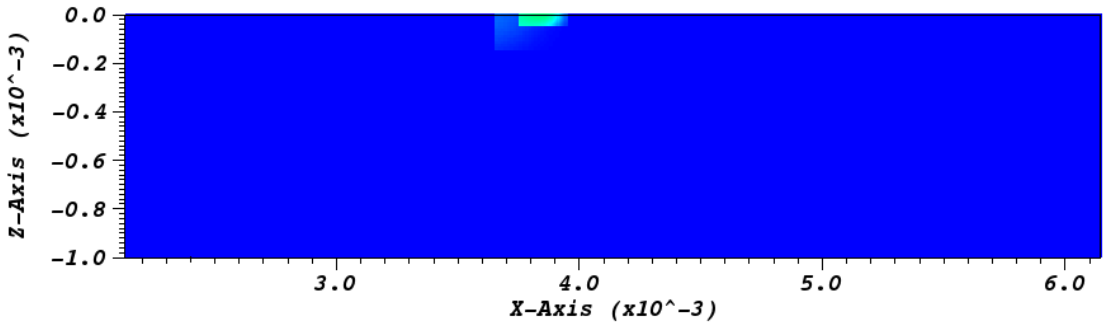
Projects a 3D plot to a 2D surface
Operators → **Transforms** → **Project**



Operators - **Project**



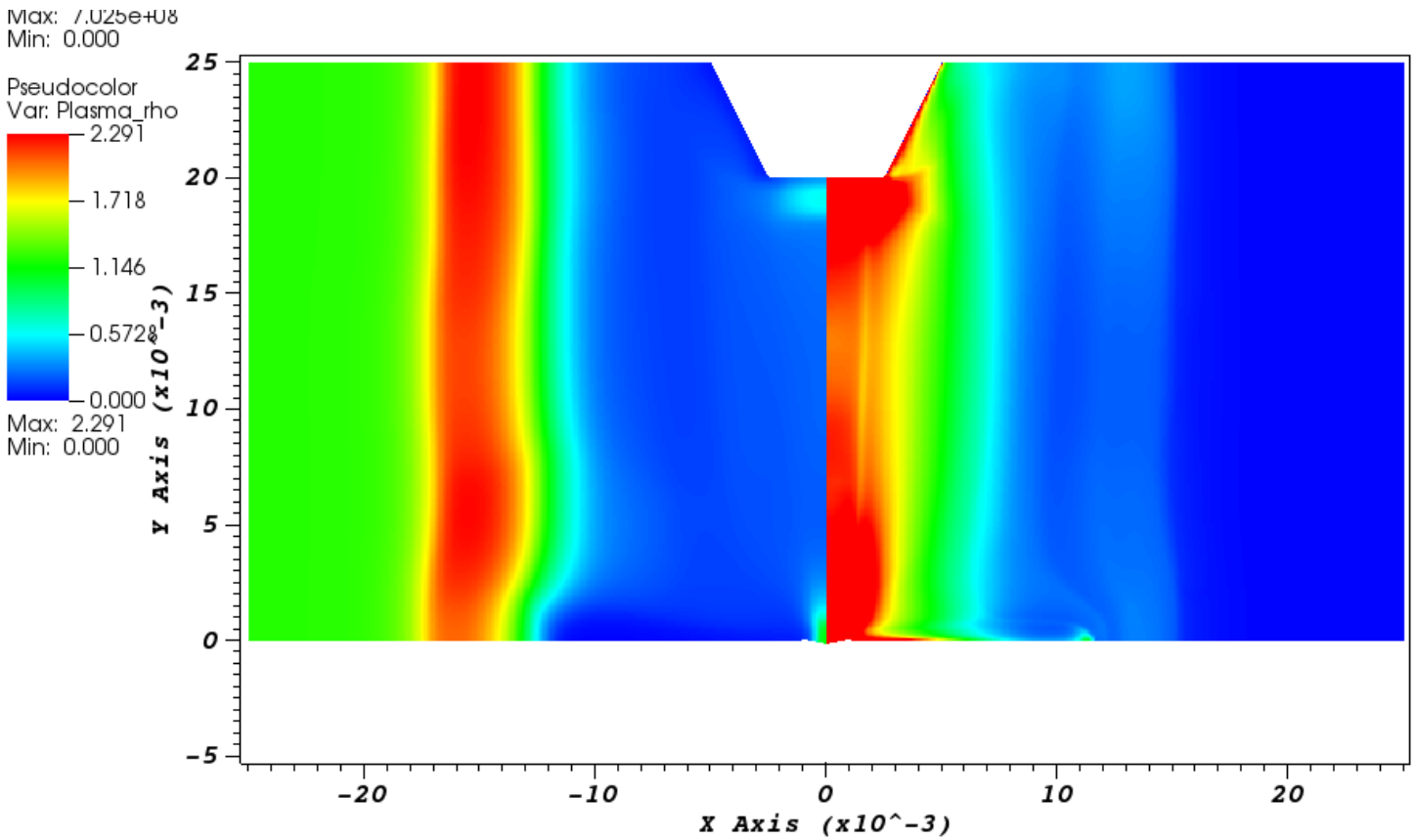
- Not all 3D plots are suitable for projection - the output may contain information combined through the plot, slice may be a better option
- **Projection type** below is **Y-Axis Cartesian** (x- and z-axes remain)
- Useful for **Molecule** plots, however



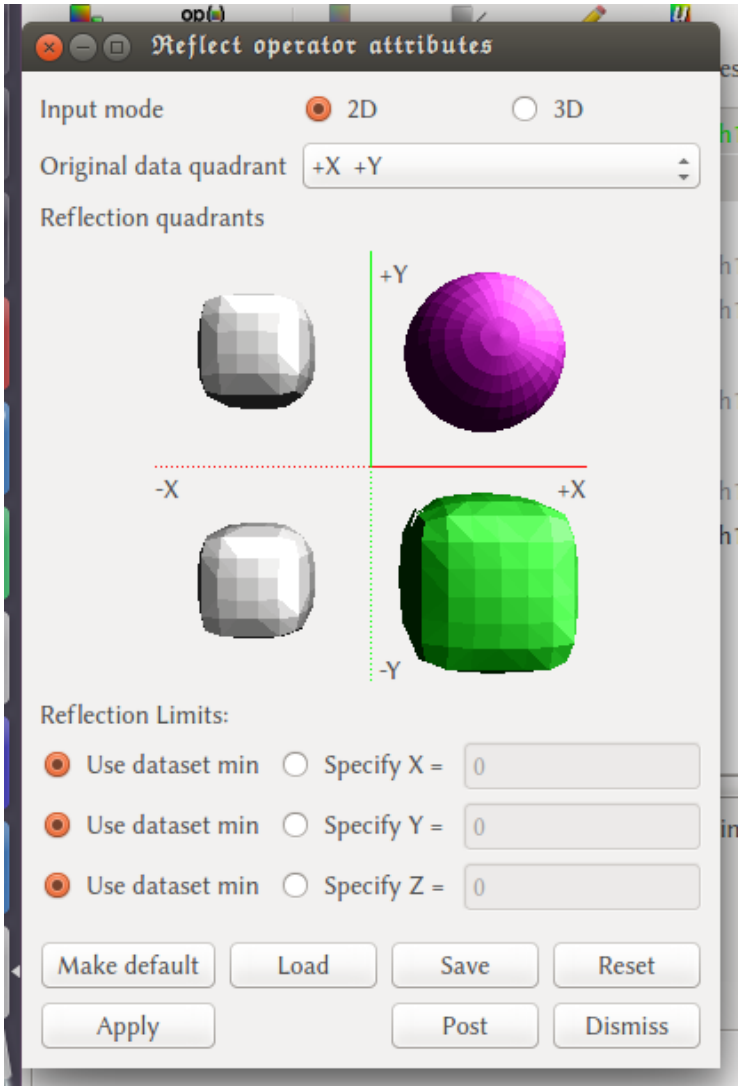
Operators - **Reflect**

Allows the plot to be reflected along a coordinate direction

Operators $\rightarrow$ **Transforms** $\rightarrow$ **Reflect**



Operators - **Reflect**

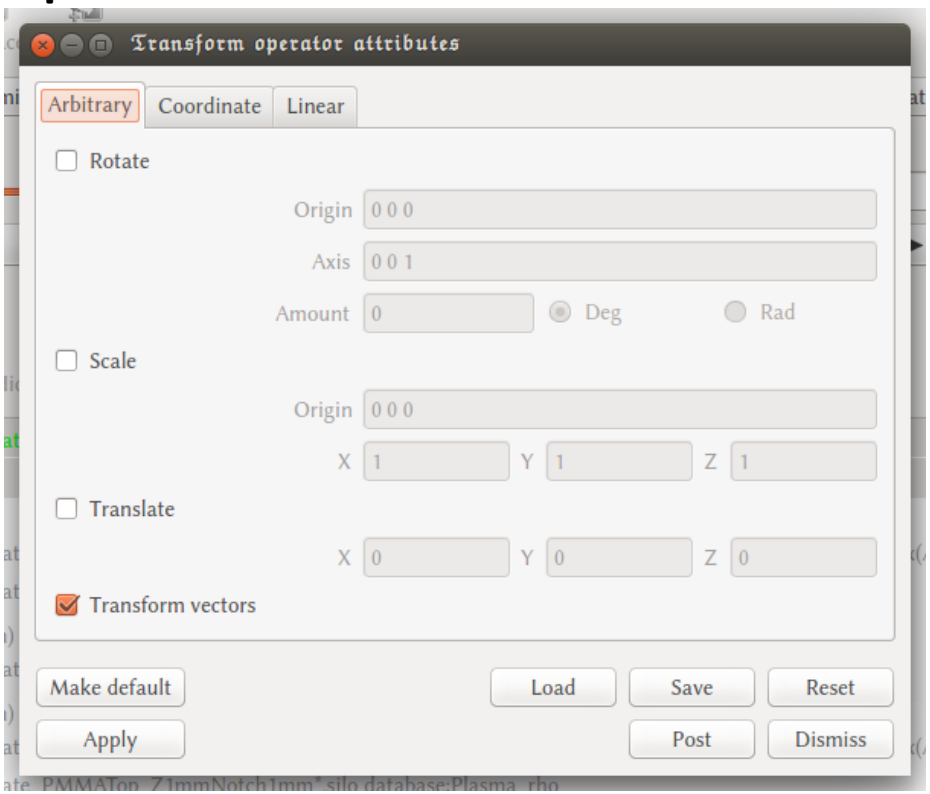


- **Input mode** - 2x2 grid in 2D, 2x2x2 grid in 3D, simply click the shapes corresponding to where you want to turn reflection on/off
- **Original data quadrant** - where is the purple ball placed, allows for easy reflection through any coordinate edge
- **Reflection limits** - Reflection does not have to correspond to the domain edge, and can be offset, either to remove trailing white-space, or introduce it for clarity

Operators - **Transform**

A general set of translation, rotation and scaling operations

Operators → **Transforms** → **Transform**



- **Rotate** - rotates a 2D plot about a point, or 3D about an axis
- **Scale** - multiply an axis by a given factor, e.g. can convert mm to m, or make a rectangular domain square
- **Translate** - move a plot in a specified direction, quantities always based on the physical coordinate values, useful for plotting more than one image in a single window

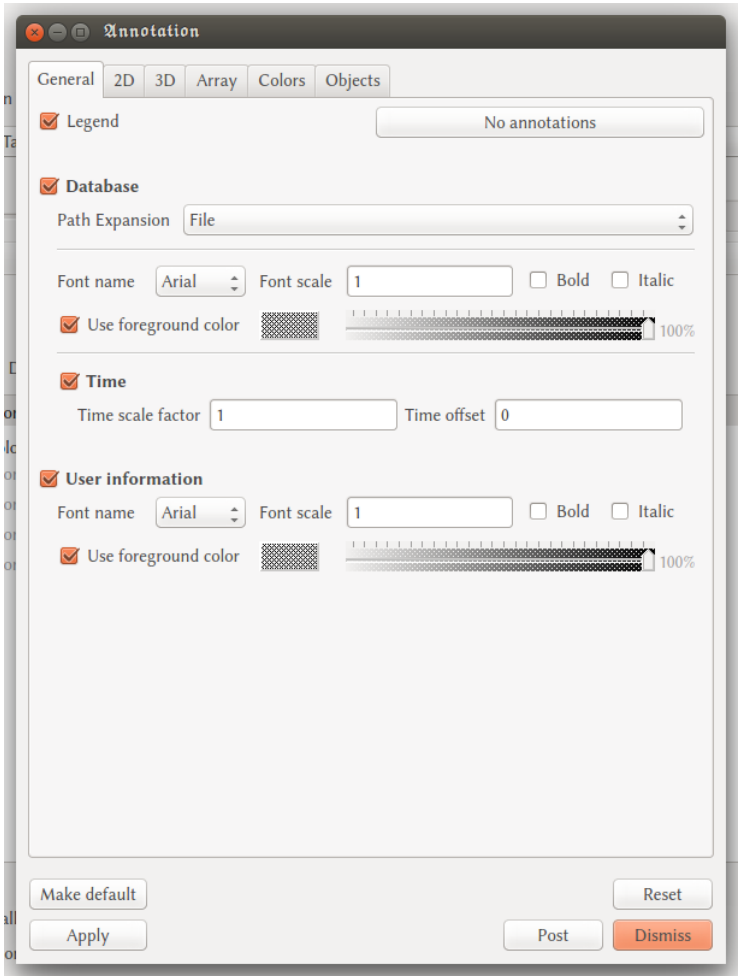
Outline

Controls

- Controls are accessed from the VisIt menu bar (or by shortcut keys)
- They affect the overall window display, rather than individual plots
- They also allow from manipulation of entire data files, e.g. comparisons or function creation
- We will only cover the most useful one (or, at least, the ones I find most useful...)

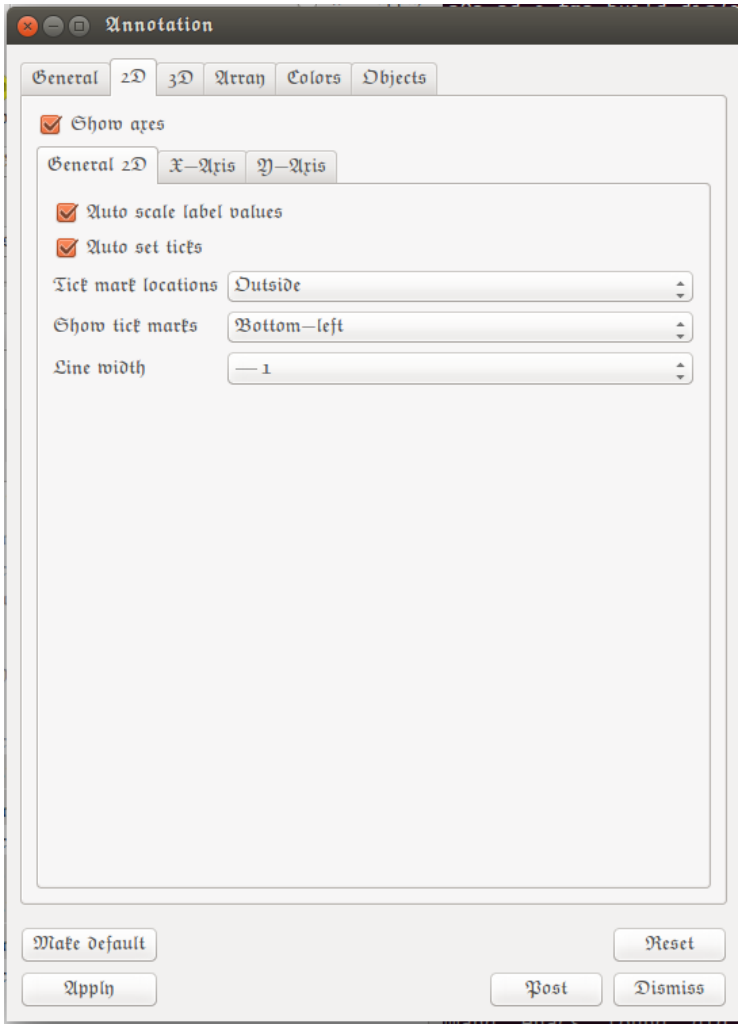
Controls - Annotation - General

The annotation menu pretty much controls all text in the window, almost always needs modifying to produce plots for publication or presentation



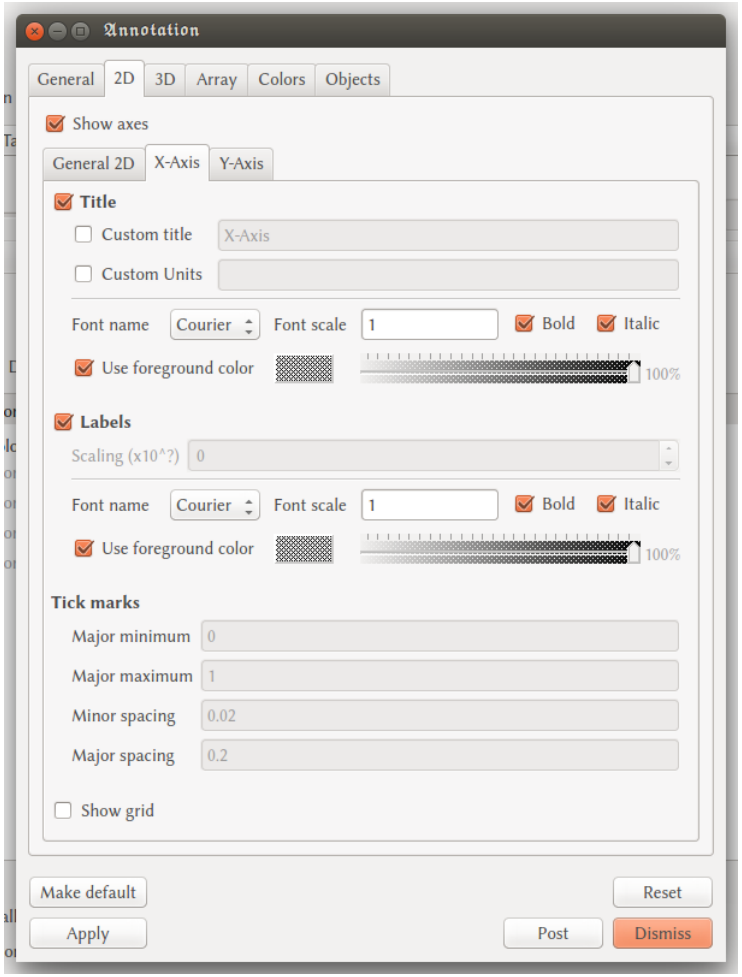
- **Legend** - this can toggle *all* legends on or off
- **Database** - displays the filename, cycle (often the code or output iteration) and time (if available from the output file) - useful for identifying plots, but looks a bit messy
- **User Information** - On by default, and gives your username, e.g. crsID - why?!

Controls - Annotation - 2D - General 2D



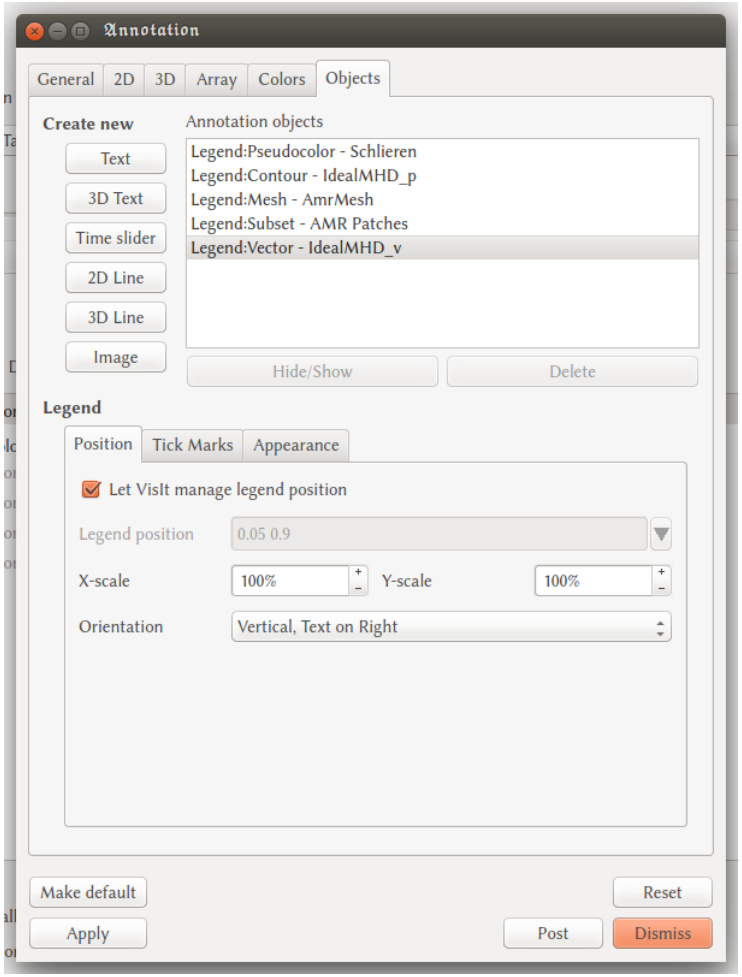
- The functionality of 2D and 3D is very similar
- **Auto scale label values** - VisIt will choose the best format for displaying numbers, if deselected, each axis can be chosen independently
- **Auto set ticks** - default is to divide the axis into 5 major regions, each of 10 minor regions, and not marking the axis extremes (e.g. (0,0)), again this can be chosen independently for each axis if necessary
- **Line width** - this controls the width of both the bounding box and the ticks (but not the length of the ticks)
- In 3D, **Show triad** and **Show bounding box** options are also available

Controls - Annotation - 2D - X-Axis



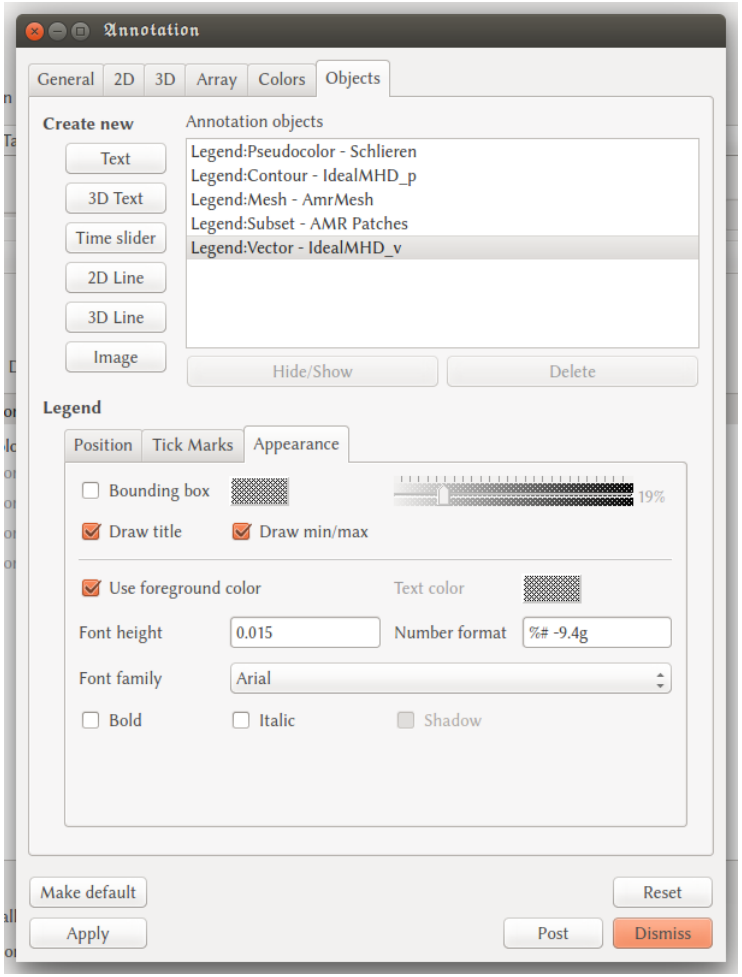
- All axis menus are functionally identical (2D and 3D)
- **Title** - set the desired axis label
- **Labels** - The numerical values on the axis, font options are always available, scaling options only if VisIt is not doing this automatically
- **Tick marks** - If not automatically set, they are chosen here
- **Show grid** - overlay a grid based on the major tick spacing - needs to be done for each axis, individually, only a series of lines are plotted

Controls - Annotation - Objects - Legend



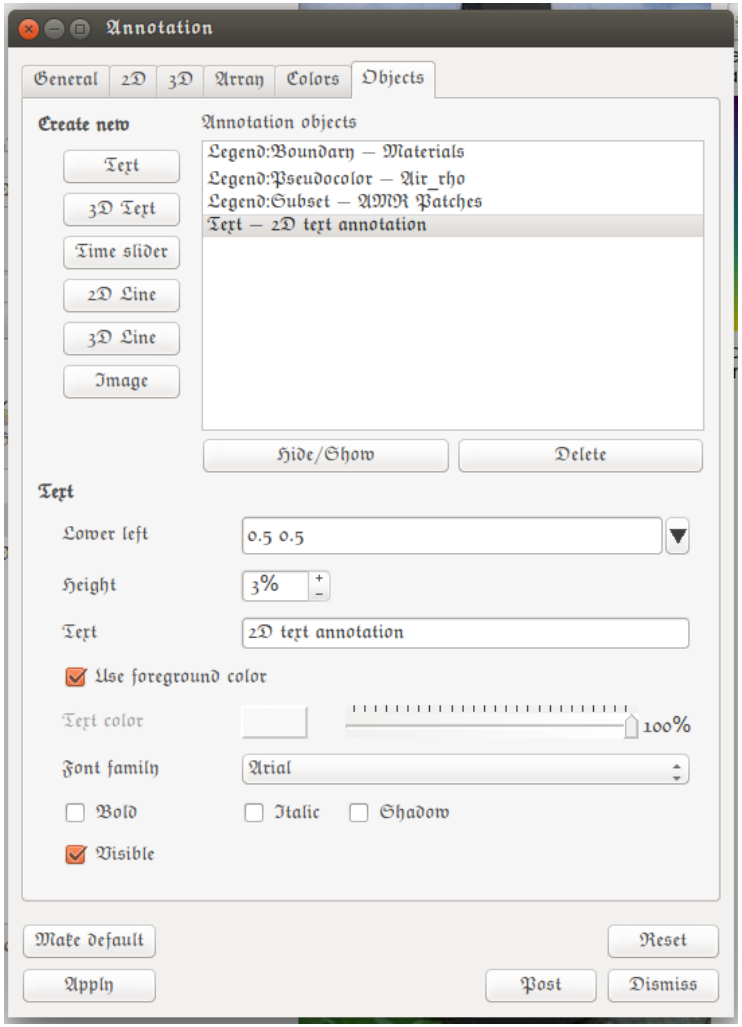
- The objects tab can control a lot, including legend positions
- **Position** - VisIt attempts to place legends sensibly (beside the plot), based on the chosen **Orientation**, which doesn't always look good for three or more
- Deselecting the checkbox gives you control, the coordinate position is the top-left corner of the legend object, as a fraction of the window
- The arrow box gives you a cross-hair location tool
- Note - VisIt does not consider manually placed legends when auto-placing others

Controls - Annotation - Objects - Legend



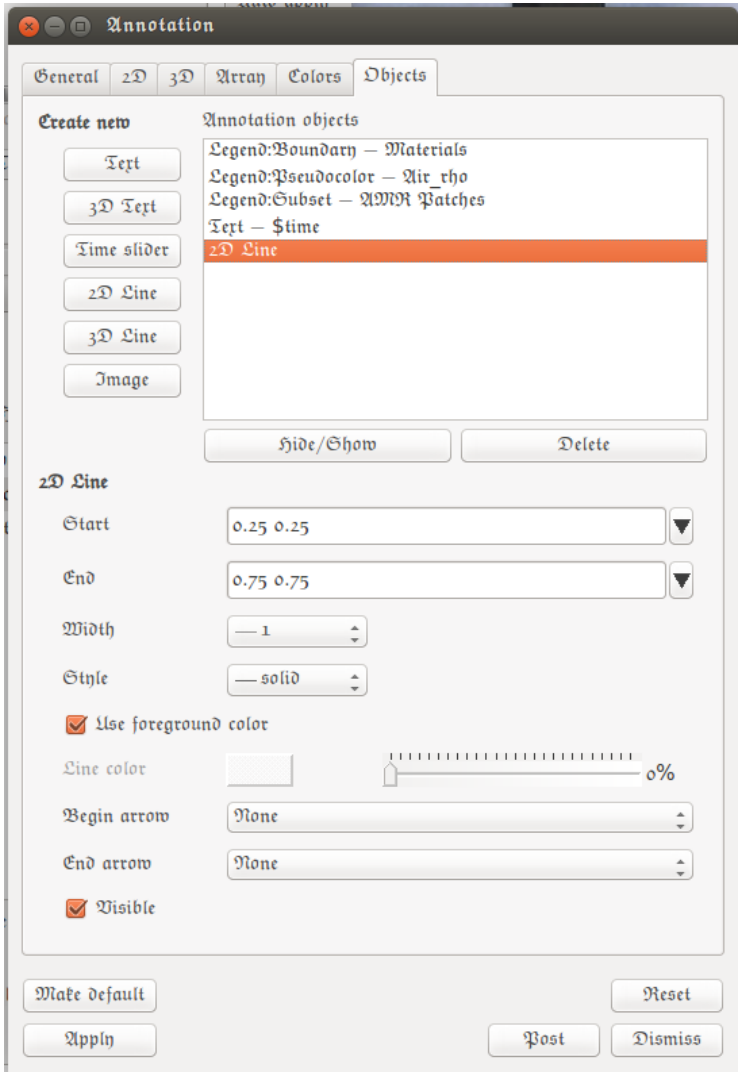
- **Appearance** - some control over how the legend is displayed
- **Bounding box** - produces a coloured box behind the legend, with transparency, useful if legend needs to be on the plot
- **Draw Title** - toggles a legend title comprising plot type and (VisIt) variable name, contents cannot be changed
- **Draw min/max** - Default is to have absolute variable extents (not plot-limited) beneath the legend, not usually necessary
- The font control is similar to previous examples, size and format often need changing for readability

Controls - Annotation - Objects - Text



- New objects can be created, in addition to the legends - VisIt will always ask for a name, in the GUI environment, this name is irrelevant
- **Text** - text can be placed on plots, e.g. titles, labels, custom legend heading
- Position is a fraction of the view window, and, as the name suggests, measured from the **Lower left** of the text
- Font controls as other cases
- Certain variables can be entered as text, e.g. **$time** will output the simulation time (avoids having to plot full database information)

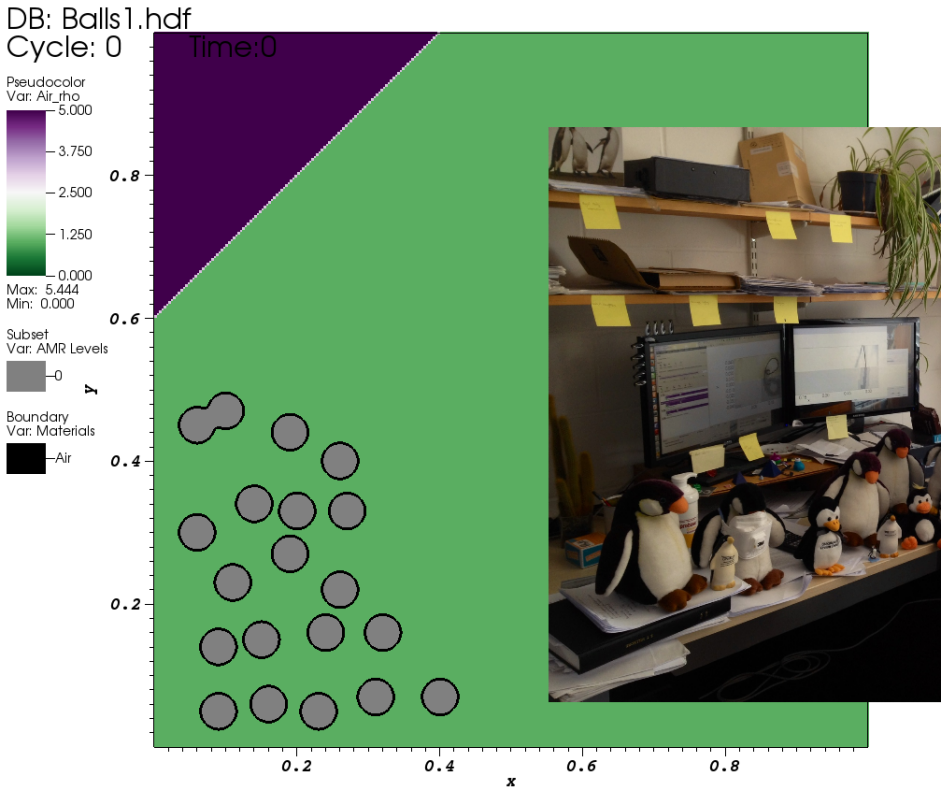
Controls - Annotation - Objects - Line



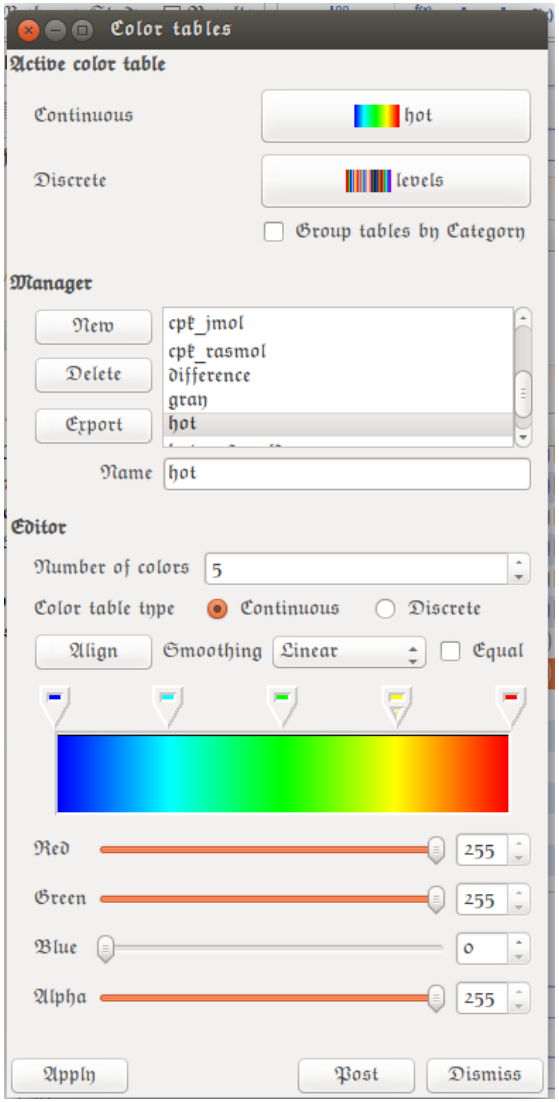
- **Line** - lines can be drawn on plot, including as arrows, useful for labelling
- Start and end, as with other objects, are fractions of the view window
- Additionally, there is **3D Text** and **3D Line**, for adding these features to 3D plots. My little experience of 3D text is that is a bit of a pain to work with

Controls - Annotation - Objects - Image

- Adding additional images to a plot could actually be useful for comparing plots to those of a paper
- Applying transparency can be a graphically intensive operation

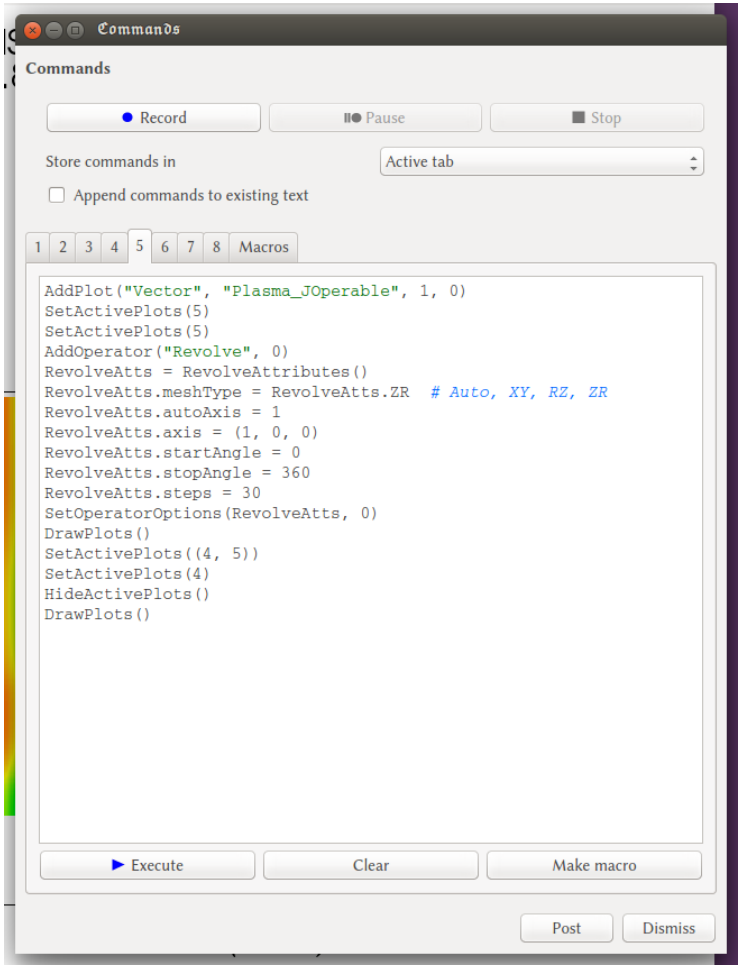


Controls - Color tables



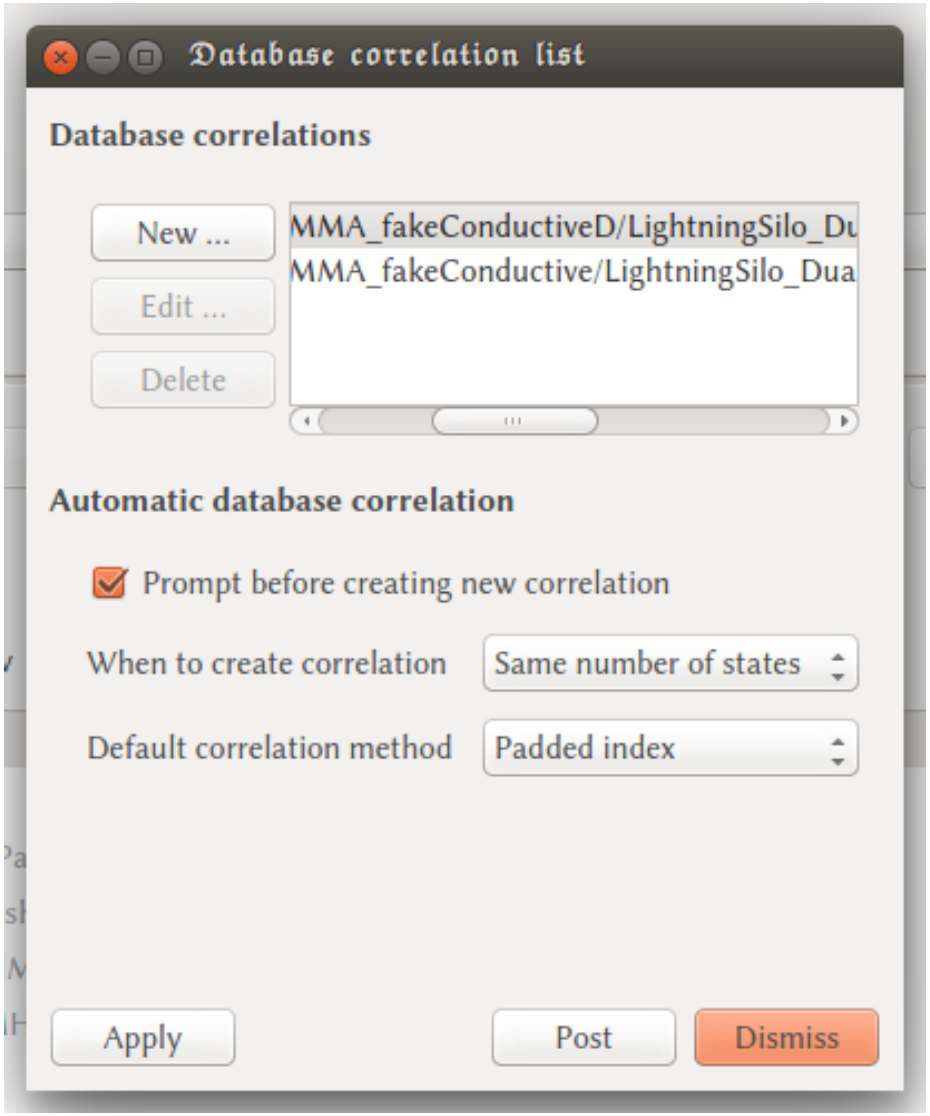
- You can make your own colour scheme, e.g. for matching another plotting program's results
- Each triangle can slide, and have its colour set, and VisIt will interpolate between them (in continuous mode)
- Modifying the existing colour tables is possible, at least within a single plot

Controls - Command

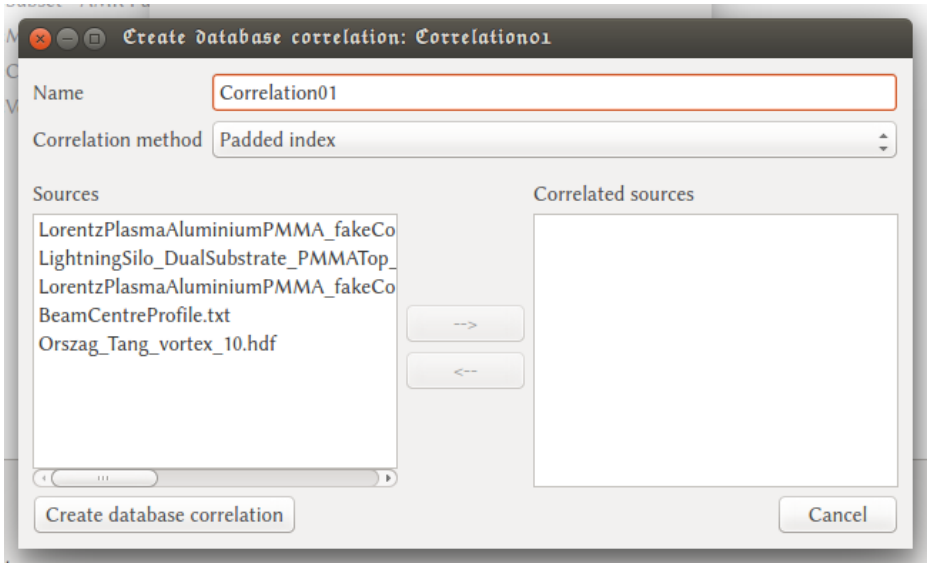


- The command menu is primarily used with the python interface (see later)
- It can identify the commands which actually add, edit and draw plots
- Pressing **Record** will start recording every operation that you do through mouse clicks, and they will be printed when **Stop** is pressed
- The commands are also editable (then press **Execute**), so could offer a quick way of setting multiple desired features in a new plot
- There are a few commands which don't get recorded well, e.g. **Pick**

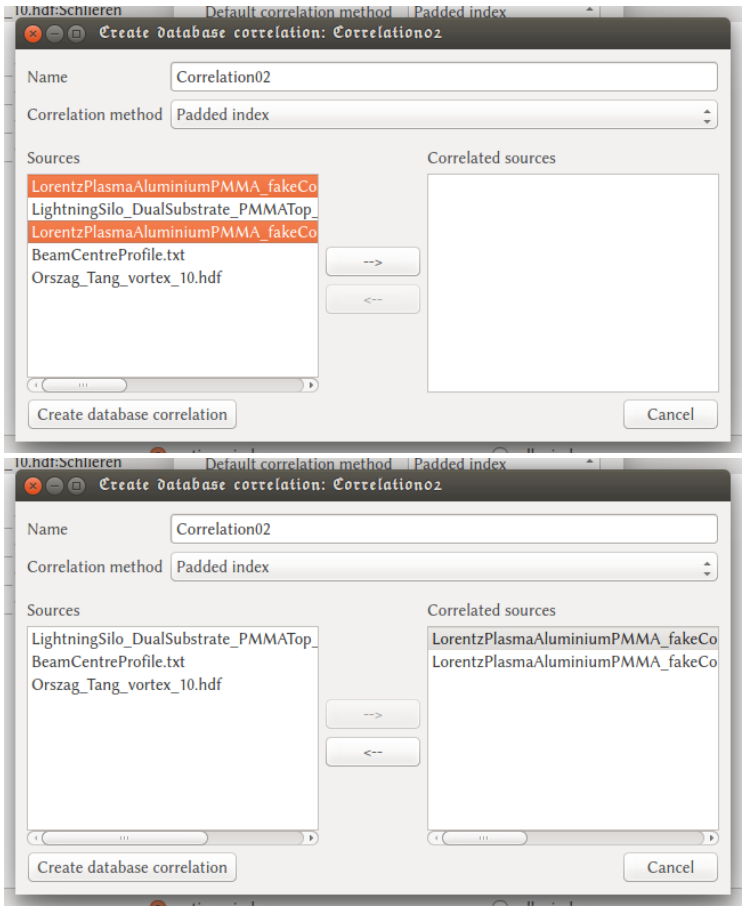
Controls - Database correlations



- Allows two or more databases to be controlled by a single time slider
- Select **New**, and a new window appears allowing you to name your correlation (as it will appear in the **Active source** menu of the main window

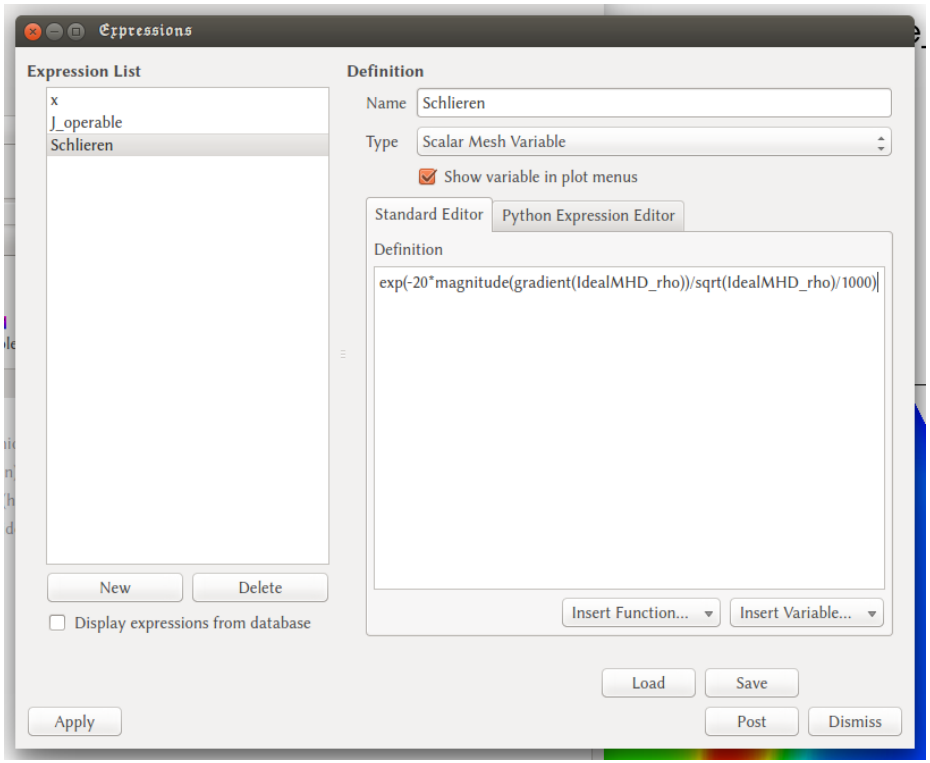


Controls - Database correlations



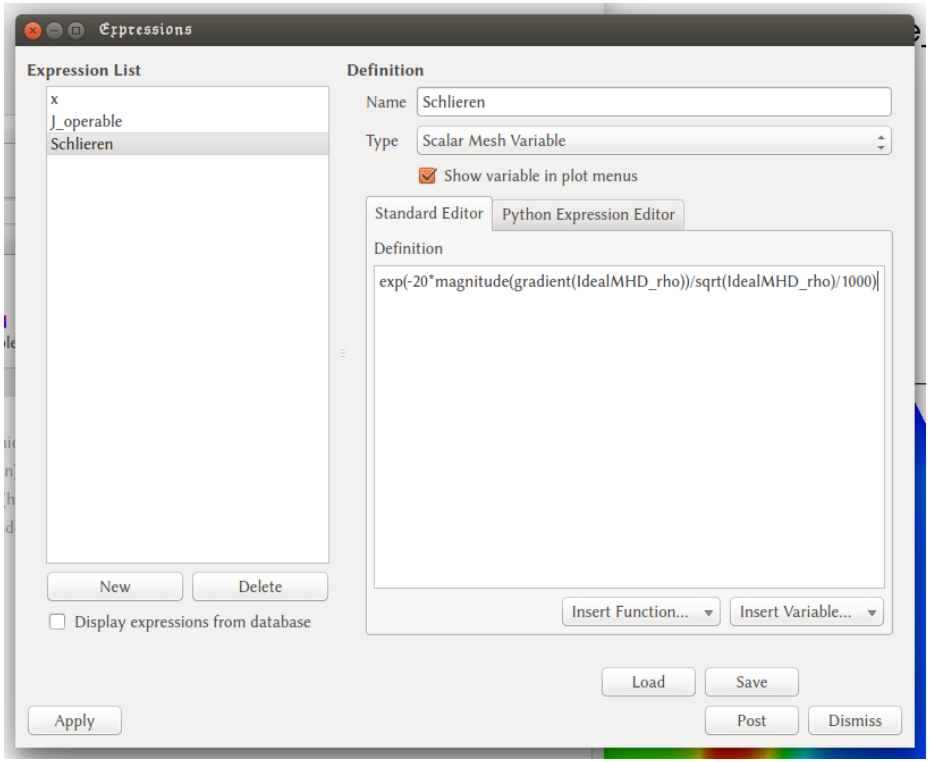
- Click on as many databases as you wish to correlate, this highlights them in the left box
- Pressing the arrow adds them to the correlation (if you made a mistake, you can remove them again)
- Sometimes, when plotting two databases on the same window, VisIt will offer to create a correlation for you
- If the databases have different numbers of outputs, this is handled ok, on set of results will simply stop advancing whilst the other continues to

Controls - Expressions



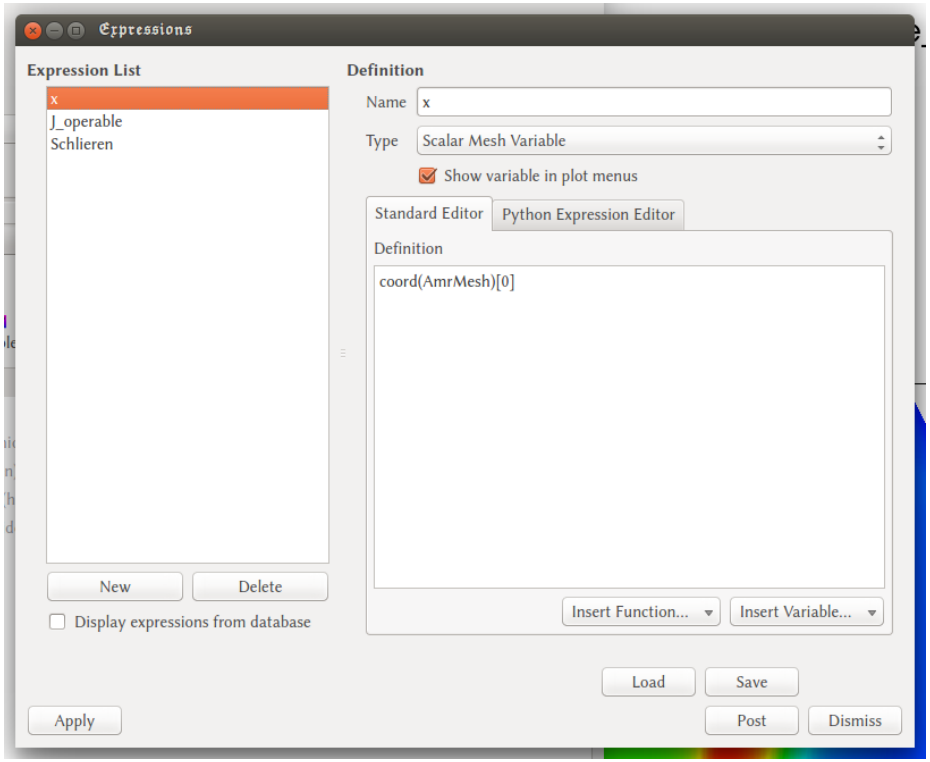
- Expressions are user defined functions and variables
- () defines function evaluation, or standard mathematical use, [] an array entry and {} a vector quantity
- $+$, $-$, $*$, $/$ and order of operations works as normal, other syntax is not necessarily straightforward
- Variable names, through **Insert variable**, are the VisIt names, and can include previously defined expressions

Controls - Expressions



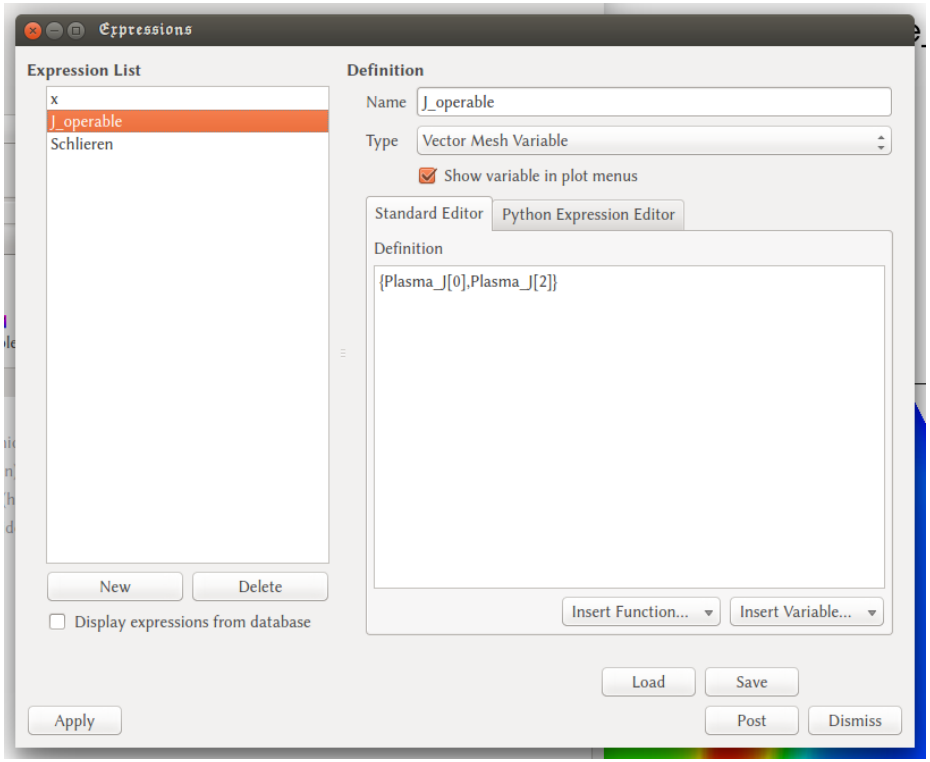
- **Insert function** provides a list of functions, separated into approximately useful categories
- Some functions are inserted with syntax hints, for example, selecting 'lt':
  lt(<var-LHS>, <var-RHS>)
- This is the less than (<) function
- Expressions can make scalars, vectors, and tensors, and can be constants, as well as functions

Controls - Expressions



- Perhaps surprisingly, VisIt has no inbuilt means to access coordinate information - a position expression must be created
- It does have a 'coord' function, though, this example defines the x-position
- The variable within the coord function should not matter
- The output of coord() is a vector, the x-component is the 0^{th} entry

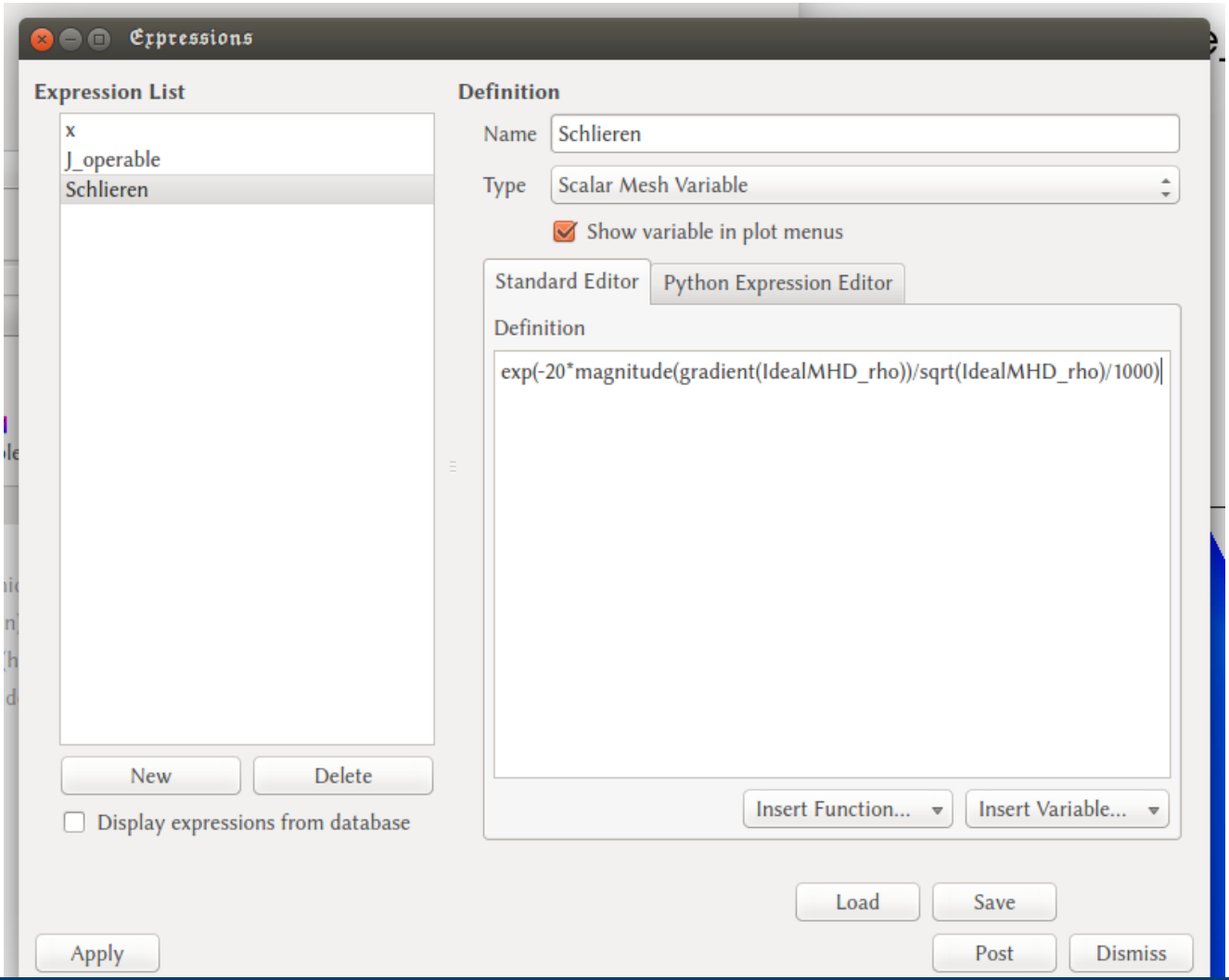
Controls - Expressions



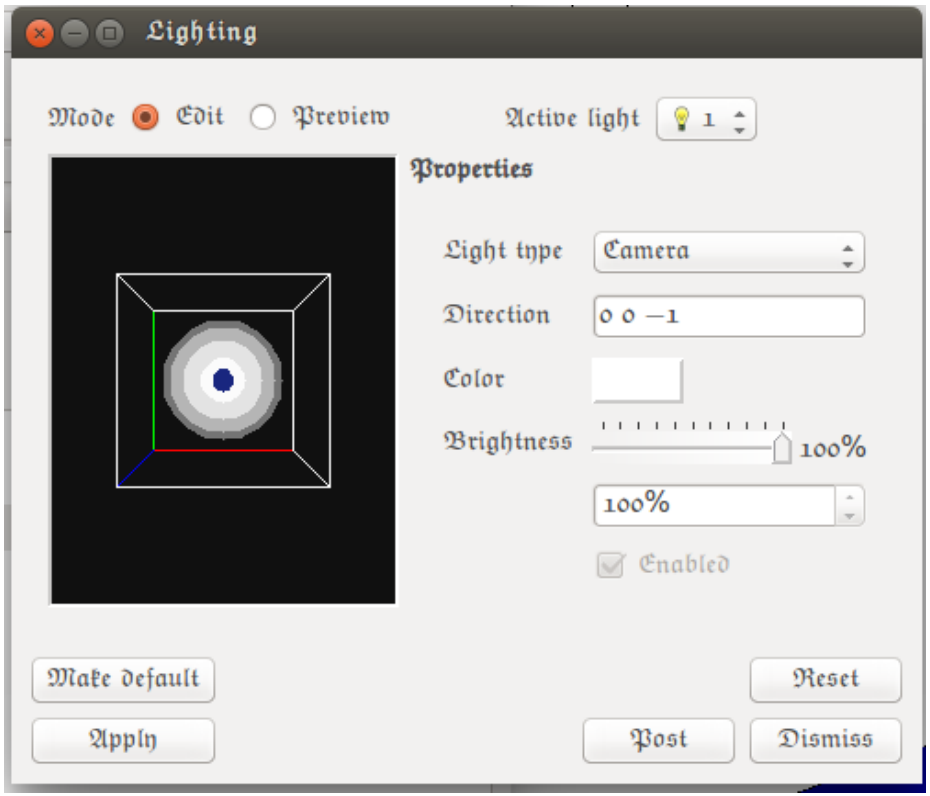
- Defining vector expressions is just a matter of putting each component within a comma separated list enclosed by {}
- VisIt will only use vector entries up to the dimension of the plot (i.e. the first 2 or 3)

Controls - Expressions

Another useful example - mock-Schlieren command



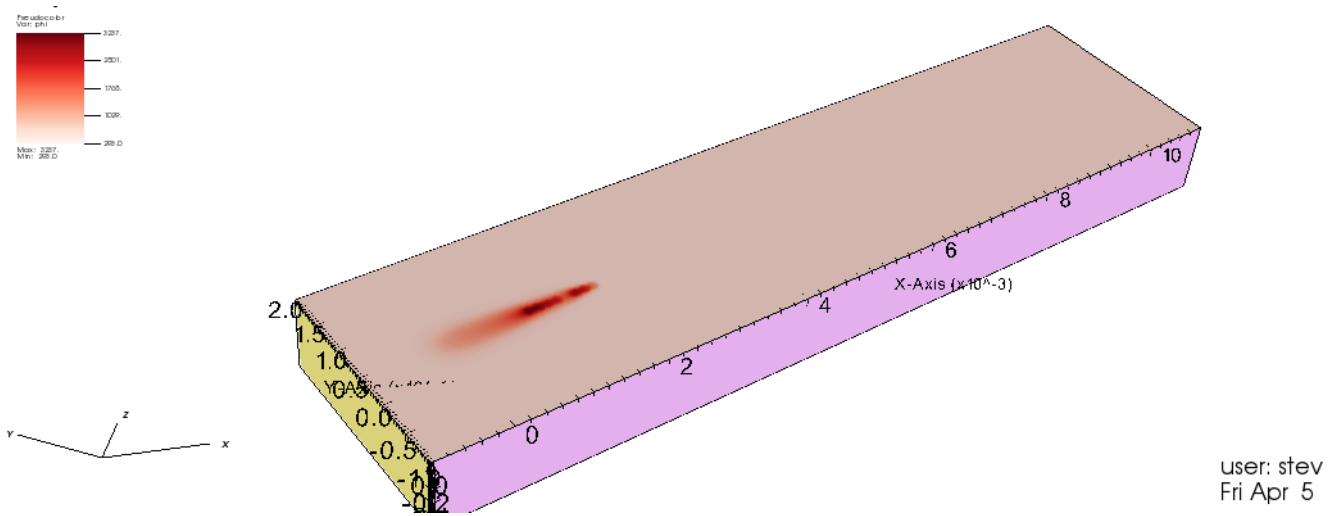
Controls - Lighting



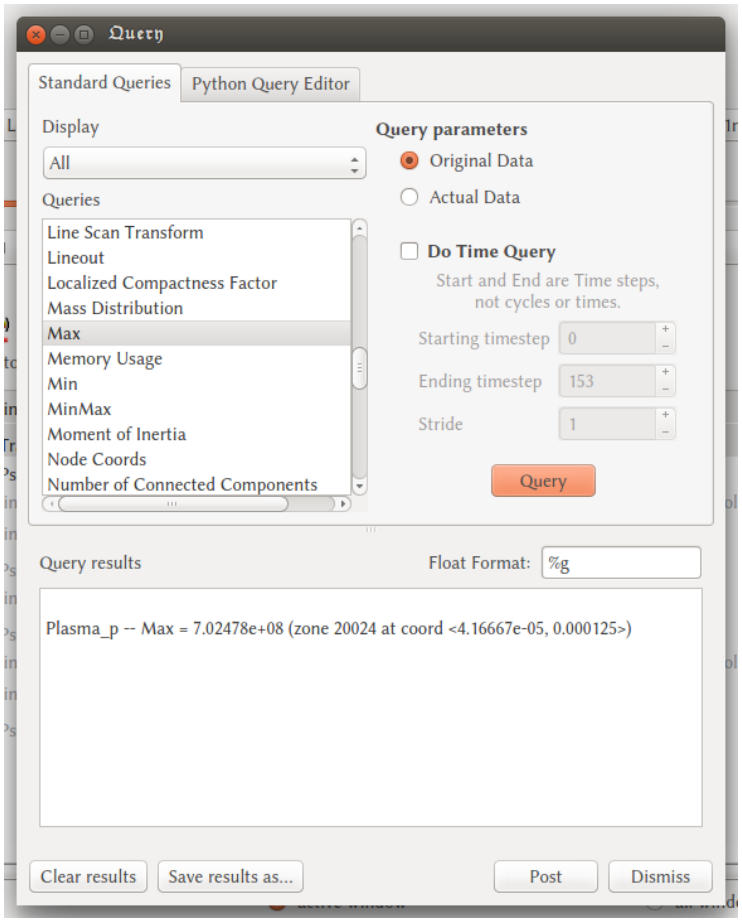
- In 3D, VisIt adds lighting effects, to attempt to improve visibility
- However, this can leave some parts of the plot in shadow, fortunately lighting can be customised
- Up to 8 lights can be **Enabled** - light 1 cannot be disabled
- Position can be controlled with click-and-drag the dot in the box, or through the **Direction** box
- Each light has its own colour, brightness and direction ('Camera' - an external source, 'Object' - from the object, 'Ambient' - no direction)

Controls - Lighting

Playing with lighting doesn't always produce good plots...

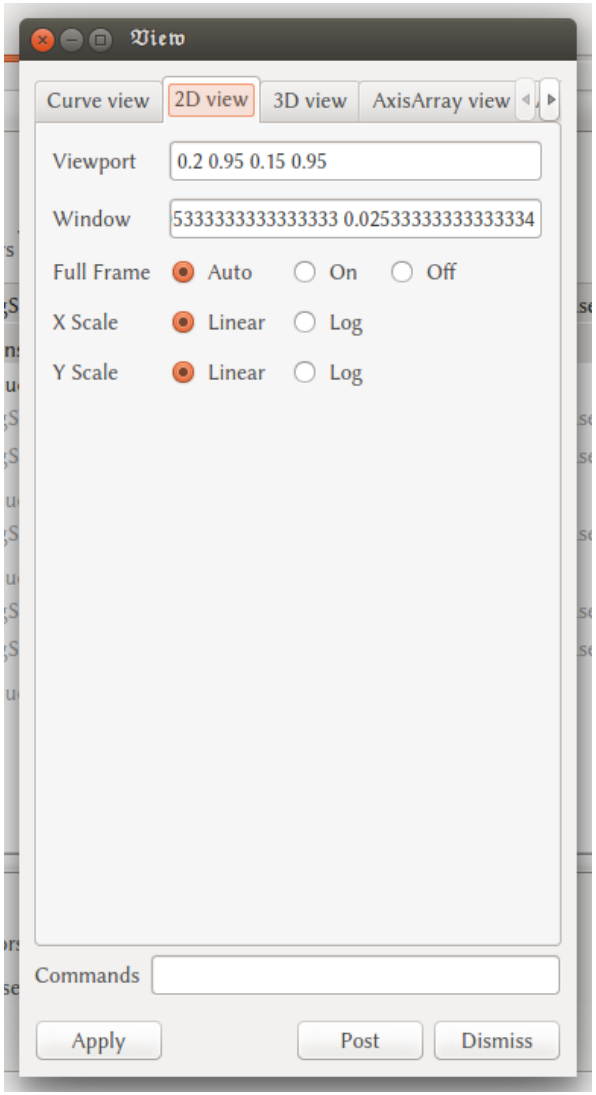


Controls - Query



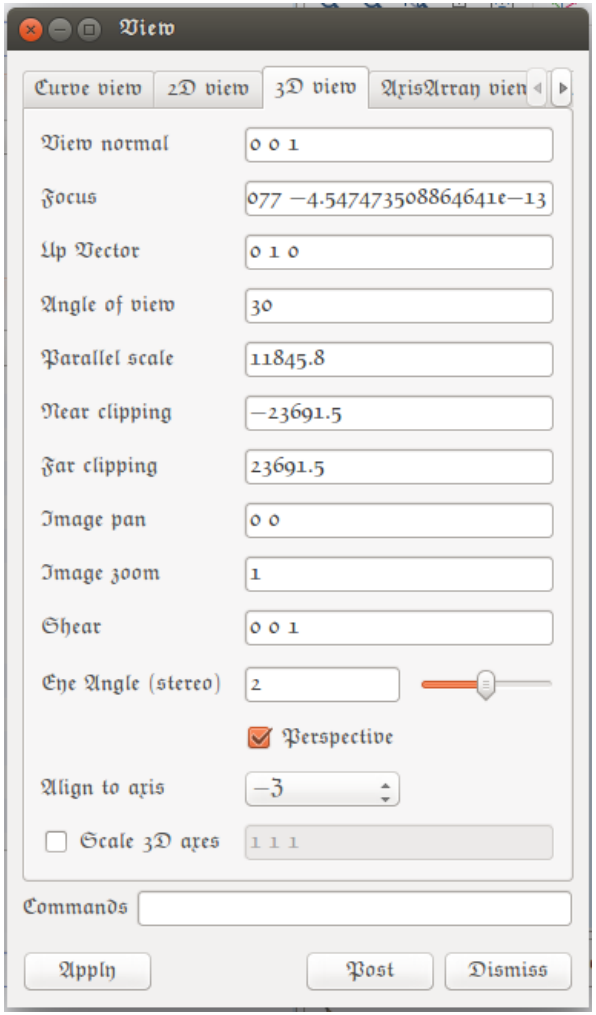
- The query window allows you to do some post-processing of the data
- The output is given in the **Query results** window, query will always act on the currently selected plot
- **Do Time Query** will produce a plot over time for the desired variable (not available for all queries)
- Depending on the data structure, the query results may not take account of geometric effects, e.g. cylindrical domain

Controls - View - 2D View



- The View menu customises the size of the plot relative to the size of the window, as well as the axis extents
- **Viewport** specifies the bottom-left and top-right corners of the plot, x first, then y
- The scaling cannot be changed by this, once one coordinate domain is full, the other will no longer be re-sized
- The **Window** corresponds to the axis ranges, allows for accuracy when zooming

Controls - View - 3D View



- 3D offers much more customisability, due to rotational freedom
- Personal preference - get roughly the right position through click-and-drag, then tweak through View menu
- Large files can be graphically expensive to do this, so the view menu can avoid 'in-between' rendering of two different views
- Some options are fairly obvious (**Image Zoom**, **Image pan**), some fiddly (**View normal**) and some best adjusted through trial and error when things don't look right!

Other options

- There are many other options in VisIt (plots, operators, controls), not covered here, mostly because I've never used them (or very rarely, a long time ago)
- For example, Controls → Data-Level Comparisons lets you do things such as subtract one database from another, and Controls → Material Options changes how boundaries are interpolated
- Other features relate to data types I've never used, or to a level of visualisation far beyond anything I've considered

Outline

Why script plots?

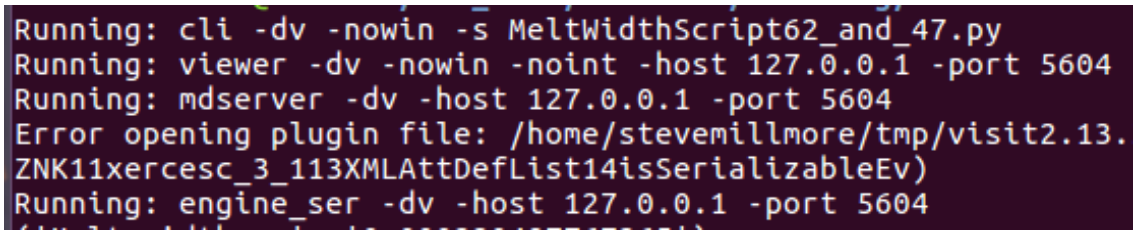
- In addition to the VisIt GUI, it can also be run remotely, through scripts
- This can speed up data processing, e.g. making similar plots from parameter study data
- It can be a safe way of generating the images for movies, especially if several minor changes are likely to be necessary, and you don't want VisIt to crash
- It can avoid excessive graphical rendering for 3D plots
- VisIt can be operated remotely, avoiding rendering over an SSH connection
- Post processing information can be automated

Command line interface

- VisIt supports a command line interface, in addition to the GUI, which can be used, even without running a script (I have never tried this)
- Assuming you have a script to run, this can be called through the command line by:

```
> visit -cli -nowin -s <script_name>.py
```

- This should start generating some default output text to the terminal



- The error here is VisIt trying to open a plugin for file reading (GDAL here), it does not cause a problem in this case, since I don't attempt to open a GDAL file
- Note: Errors in the script itself are not necessarily handled gracefully, and will leave you in VisIt's command line interface, `> exit()` will get you back to the terminal

What can be run?

- The command line interface basically runs through python, therefore, any python program can actually be run this way
- Not recommended - much slower than just running python
- However, this means that scripts can be a combination of regular, and VisIt-specific python, including from any desired libraries etc.
- We shall cover a few examples, and how to set up a file, in general, specific commands are best identified through the **Command** feature in the Controls menu

An example file

```
OpenDatabase("localhost:/data/hydra04-2/stm31/cns_amr/CNS_AMR_Multimaterial/output/oscillatingEllipseIdeal/surfaceTensionDroplet100_*.hdf database", 0)

AddPlot("Curve", "operators/Lineout/Air_LS", 1, 0)
LineoutAtts = LineoutAttributes()
LineoutAtts.point1 = (0, 1e-5, 0)
LineoutAtts.point2 = (0.5, 1e-5, 0)
LineoutAtts.interactive = 1
LineoutAtts.ignoreGlobal = 0
LineoutAtts.samplingOn = 0
LineoutAtts.numberOfSamplePoints = 10000
LineoutAtts.reflineLabels = 0
SetOperatorOptions(LineoutAtts, 0)
for state in range(TimeSliderGetNStates() ):
    SetTimeSliderState(state)
    DrawPlots()
    SaveWindowAtts = SaveWindowAttributes()
    SaveWindowAtts.outputToCurrentDirectory = 0
    SaveWindowAtts.outputDirectory = "/local/data/public/stm31/TexStuff/ValidationPaper/Figs/"
    SaveWindowAtts.fileName = "Ellipse0Levels"
    SaveWindowAtts.family = 1
    SaveWindowAtts.format = SaveWindowAtts.CURVE  # BMP, CURVE, JPEG, OBJ, PNG, POSTSCRIPT, POVRAY, PPM, RGB, STL, TIFF, ULTRA, VTK, PLY
    SaveWindowAtts.width = 1024
    SaveWindowAtts.height = 1024
    SaveWindowAtts.screenCapture = 0
    SaveWindowAtts.saveTiled = 0
    SaveWindowAtts.quality = 80
    SaveWindowAtts.progressive = 0
    SaveWindowAtts.binary = 0
    SaveWindowAtts.stereo = 0
    SaveWindowAtts.compression = SaveWindowAtts.PackBits  # None, PackBits, Jpeg, Deflate
    SaveWindowAtts.forceMerge = 0
    SaveWindowAtts.resConstraint = SaveWindowAtts.ScreenProportions  # NoConstraint, EqualWidthHeight, ScreenProportions
    SaveWindowAtts.advancedMultiWindowSave = 0
    SetSaveWindowAttributes(SaveWindowAtts)
    SaveWindow()

exit()
```

Opening and closing

- `OpenDatabase` is used to open either individual files, or entire databases

```
OpenDatabase("localhost:/data/hydra04-2/stm31/cns_amr/CNS_AMR_Multimaterial/output/oscillatingEllipseIdeal/surfaceTensionDroplet100_*.hdf database", 0)

for i in range(1,151):
    variableString = "localhost:"+locationString+inFileName+"/cavitationDropAluminiumVibrating"+str(i)+".hdf"

    OpenDatabase(variableString)
```

- Closing the database again is through `CloseDatabase`

```
DeleteAllPlots()

CloseDatabase(variableString)
```

- VisIt cannot close a database whilst it is being used for plots (same as GUI mode), `DeleteAllPlots()` ensures this is the case
- If plots are not closed, then they remain open until `exit()` is called
- In the case of looped opening, this can act as a memory leak

Focus

- By default, there is a focus (`SetActive...`) on the last file opened, plot created, window opened etc.
- This will be used for further functions, e.g. plots come from the active file, attributes are changed for the active plot
- This can be changed through: `SetActiveTimeSlider(<n>)` for the file, `SetActiveTimePlots(<n> [,<m>])` for the plots, and `SetActiveWindow(<n>)` for the window
- The features are numbered in the order they are created, starting from zero
- Note - multiple plots can be active at once (equivalent to highlighting multiple plot types in the GUI)
- For scripting, it is usually easiest to only modify the current plot, since commands can be placed exactly where you need them

Adding plots

```
AddPlot("Boundary", "AMRMaterials")
bdryAtts = BoundaryAttributes()
bdryAtts.colorType = bdryAtts.ColorBySingleColor
bdryAtts.singleColor = (0,0,0, 255)
bdryAtts.lineWidth = 3
bdryAtts.legendFlag = 0
SetPlotOptions(bdryAtts)

AddPlot("Pseudocolor", "Aluminium_J_magnitude")
tarPsAtts = PseudocolorAttributes()
tarPsAtts.minFlag = 1
tarPsAtts.maxFlag = 1
tarPsAtts.min = 0
tarPsAtts.max = 3e8
tarPsAtts.colorTableName = "hot_and_cold"
tarPsAtts.invertColorTable = 0
tarPsAtts.legendFlag = 0
tarPsAtts.scaling = 0
SetPlotOptions(tarPsAtts)

AddOperator("Threshold")
watThAtts = ThresholdAttributes()
watThAtts.lowerBounds = 1e-5
watThAtts.listedVarNames = "Aluminium_LS"
SetOperatorOptions(watThAtts)
```

- Adding plots is a matter of `AddPlot(''<Plot_name>'',''<Variable_name''`
- Altering the attributes of this plot requires an attributes object to be created
- Values are then edited within this object
- They are applied to the plot through a `Set...Options` call
- Once all plots are set, `DrawPlots()` will plot them (not that you'll actually see this...)

Saving plots

```
    outputString = "PlasmaCarbonCurrent_VCPaper"+str(i)
    saveAtts = SaveWindowAttributes()
    saveAtts.outputToCurrentDirectory = 0
    saveAtts.outputDirectory = "/lsc/zeushome/stm31/TexStuff/
#    saveAtts.outputDirectory = "/home/stevemillmore/lsc_home
    saveAtts.fileName = outputString
    saveAtts.family = 0
    saveAtts.format = saveAtts.PNG
    saveAtts.height = 1024
    saveAtts.width = 2048
    saveAtts.resConstraint = saveAtts.NoConstraint
    saveAtts.screenCapture = 1
    SetSaveWindowAttributes(saveAtts)

    SaveWindow()
```

- Saving files requires the creation of `SaveWindowAttributes`, and setting the decided options
- Even though there is no screen, `screenCapture` can be used, with results at the resolution of the graphics output of the machine the script is run upon (I think)
- Once settings are saved, `SaveWindow` will save output to the desired location

Some examples - defining expressions

```
import math
import sys

#transparencyCutOff = 3.5e6

for i in range(0, 201):
#for i in [1,10,100,150]:

    variableString = "localhost:/media/AnotherHardDrive/cns_amr/PlasmaModel-
#    variableString = "localhost:/home/stevemillmore/cns_amr/PlasmaModel-str

    OpenDatabase(variableString)

    schlString = "coord(AMRMesh)[0]"
    DefineScalarExpression("x", schlString)#

    schlString = "coord(AMRMesh)[1]"
    DefineScalarExpression("y", schlString)#

    schlString = "{Plasma_J[0],Plasma_J[2]}"
    DefineVectorExpression("Plasma_JOperable",schlString)

    schlString = "{Aluminium_J[0],Aluminium_J[2]}"
    DefineVectorExpression("Aluminium_JOperable",schlString)

    print ("Plotting " + variableString)

    AddPlot("Boundary", "AMRMaterials")
    bdryAtts = BoundaryAttributes()
    bdryAtts.colorType = bdryAtts.ColorBySingleColor
    bdryAtts.singleColor = (0,0,0, 255)
    bdryAtts.lineWidth = 3
    bdryAtts.legendFlag = 0
    SetPlotOptions(bdryAtts)

    AddPlot("Pseudocolor", "Aluminium_J_magnitude")
    tarPsAtts = PseudocolorAttributes()
    tarPsAtts.minFlag = 1
    tarPsAtts.maxFlag = 1
    tarPsAtts.min = 0
    tarPsAtts.max = 3e8
    tarPsAtts.colorTableName = "hot_and_cold"
    tarPsAtts.invertColorTable = 0
    tarPsAtts.legendFlag = 0
    tarPsAtts.scaling = 0
    SetPlotOptions(tarPsAtts)
```

- In this example, files are manually looped over, rather than opened as a database (can avoid errors opening too many files and/or reduce memory usage)
- Expressions are easy to define, simply enter the same text as would be used in the **Controls** menu
- The assigned names can then be used in place of any other variable in the plot menus

Some examples - annotation and view

```
annAtts = AnnotationAttributes()
annAtts.userInfoFlag = 0
annAtts.databaseInfoFlag = 0
annAtts.axes3D.visible = 0
annAtts.axes3D.triadFlag = 0
annAtts.axes3D.bboxFlag = 0
annAtts.axes2D.xAxis.title.font.font = annAtts.axes2D.xAxis.title.font.Arial
annAtts.axes2D.xAxis.title.font.bold = 0
annAtts.axes2D.xAxis.title.font.italic = 0
annAtts.axes2D.xAxis.title.font.scale = 2
annAtts.axes2D.yAxis.title.font.font = annAtts.axes2D.yAxis.title.font.Arial
annAtts.axes2D.yAxis.title.font.bold = 0
annAtts.axes2D.yAxis.title.font.italic = 0
annAtts.axes2D.yAxis.title.font.scale = 2
annAtts.axes2D.xAxis.label.font.font = annAtts.axes2D.xAxis.label.font.Arial
annAtts.axes2D.xAxis.label.font.bold = 0
annAtts.axes2D.xAxis.label.font.italic = 0
annAtts.axes2D.xAxis.label.font.scale = 2
annAtts.axes2D.yAxis.label.font.font = annAtts.axes2D.yAxis.label.font.Arial
annAtts.axes2D.yAxis.label.font.bold = 0
annAtts.axes2D.yAxis.label.font.italic = 0
annAtts.axes2D.yAxis.label.font.scale = 2

SetAnnotationAttributes(annAtts)

view = View2DAttributes()
view.viewportCoords = (0.15, 0.98, 0.15, 0.95)
view.windowCoords = (0, 0.05, -0.01, 0.05)
SetView2D(view)


#view = View3DAttributes()
#view.viewNormal = (0.7, 0.65, 0.3)
#view.focus = (0, 0, -0.005)
#view.viewUp = (-0.2, -0.2, 0.95)
#view.parallelScale = 0.1
#view.nearPlane = -0.2
#view.farPlane = 0.2
#view.imagePan = (0, -0.12)
#view.imageZoom = 2.6
#SetView3D(view)
```

- The nested menus of the **Annotations** window lead to long variable names
- Some variables can be picked by name (e.g. `colorTableName` when plotting pseudocolors), the fonts available are not amongst them
- Here we show both 2D and 3D (commented) view setting options

Some examples - more selection and query

```
for i in range(9100, 9800):

    zi = str(i).zfill(5)
    variableString = "/home/stevemillmore/AMReX/heat-equation-in-amrex/Exec/run3d/N
st_"+zi+"/Header"
    if(not os.path.isfile(variableString)):
        continue

    it = it + 1

    OpenDatabase(variableString)

    AddPlot("Pseudocolor", "phi")
    DrawPlots()

    Query("Time")
    time = GetQueryOutputValue()

    outString = str(time) + " "

    SetQueryFloatFormat("%g")

    for dx in range(11):


        #    ZonePick(coord=(0.0252, 0.0248, 0.0029))
        #ZonePick(coord=(0.0252472, 0.025198, 0.0029 + float(dx) * 1e-5))
        ZonePick(coord=(0.0253008, 0.0250005, 0.0029 + float(dx) * 1e-5))
        pick = GetPickOutput().split()
        pickVal = 0.0
        #    print len(pick)
        for entry in range(len(pick)):
            #        print pick[entry]
            if entry > 2 and pick[entry-2] == "<zonal>":
                pickVal = float(pick[entry])
                #    print "time", time, "pick", pickVal

                if pickVal < 290:
                    print "Something has gone wrong ", pick
                    exit()


        outString = outString + str(pickVal) + " "
    outString = outString + "\n"
    historyFile.write(outString)
```

- If files are not consecutively numbered, python commands ensure missing ones are not opened
- Some queries, e.g. `Time`, output a single number, this is obtained as a floating point number through `getQueryOutputValue()`
- `ZonePick` can obtain information at a point, obtained through `GetPickOutput()`
- This is returned as a string, with text data included, hence needs manipulation before the desired output can be used

Some examples - more selection and query

```
for i in range(9100, 9800):

    zi = str(i).zfill(5)
    variableString = "/home/stevemillmore/AMReX/heat-equation-in-amrex/Exec/run3d/N
st_"+zi+"/Header"
    if(not os.path.isfile(variableString)):
        continue

    it = it + 1

    OpenDatabase(variableString)

    AddPlot("Pseudocolor", "phi")
    DrawPlots()

    Query("Time")
    time = GetQueryOutputValue()

    outString = str(time) + " "

    SetQueryFloatFormat("%g")

    for dx in range(11):


        #    ZonePick(coord=(0.0252, 0.0248, 0.0029))
        #ZonePick(coord=(0.0252472, 0.025198, 0.0029 + float(dx) * 1e-5))
        ZonePick(coord=(0.0253008, 0.0250005, 0.0029 + float(dx) * 1e-5))
        pick = GetPickOutput().split()
        pickVal = 0.0
        #    print len(pick)
        for entry in range(len(pick)):
            #        print pick[entry]
            if entry > 2 and pick[entry-2] == "<zonal>":
                pickVal = float(pick[entry])
                #    print "time", time, "pick", pickVal

                if pickVal < 290:
                    print "Something has gone wrong ", pick
                    exit()


        outString = outString + str(pickVal) + " "
    outString = outString + "\n"
    historyFile.write(outString)
```

```
Query("SpatialExtents", use_actual_data=1)
extents = GetQueryOutputValue()
strExtents = str(extents)
splitExtents = re.split(', |\(|\)', strExtents)
#        print("Extents are " + str(splitExtents))

#        print(splitExtents[3])
#        print(splitExtents[4])
meltWidth = float(splitExtents[4]) - float(splitExtents[3])
print("Melt width = ", str(meltWidth))

outString = str(i) + " " + str(meltWidth) + "\n"
widthFile.write(outString)
```

- Other queries take additional arguments to determine the range of the output
- Vector output is given in a braced format, which again needs processing

Summary

- Many of VisIt's options have been covered, hopefully some of them useful!
- This certainly isn't everything VisIt can do, but once you've played around with enough of the menus, additional things become more intuitive
- This presentation will be made available as reference
- VisIt 3.0 is due - hopefully this information doesn't become obsolete then...