Scientific visualization with ParaView Part 2

Alex Razoumov
alex.razoumov@westdri.ca

Digital Research Alliance of Canada

SFU SIMON FRASER UNIVERSITY

- ✔ slides, data, codes at `https://bit.ly/paraviewzipp`
  - the link will download a file `paraview.zip` (~36MB)
  - unpack it to find `codes/`, `data/` and `slides{1,2}.pdf`
  - command line: `wget https://bit.ly/paraviewzipp -O paraview.zip`
- ✔ install ParaView 5.11.x on your laptop from `http://www.paraview.org/download`

Exporting scenes (pre-computed polygons)

ParaView Glance

https://kitware.github.io/paraview-glance

PV Glance is an open-source standalone web app for in-browser 3D sci-vis

- very easy to use, ideal for sharing pre-built 3D scenes via the web
- no server ⇒ up to medium-size data (server support planned in future versions)
- interactive manipulation of pre-computed polygons
  - volumetric images, molecular structures, geometric objects, point clouds
- written in JavaScript and vtk.js + can be further customized with vtk.js and ParaViewWeb for custom web and desktop apps
- source and installation instructions
  https://github.com/kitware/paraview-glance

---

1. Create a visualization with several layers, make **all layers visible in the pipeline**
2. Many options in File → Export Scene... ⇒ save as VTKJS to your laptop
3. Open https://kitware.github.io/paraview-glance/app
4. Drag the newly saved file to the dropzone on the website
5. Interact with individual layers in 3D: rotate and zoom, change visibility, representation, variable, colourmap, opacity

Automatically load a visualisation into Glance

https://discourse.paraview.org/t/customise-pv-glance/2831

- Use the query syntax
  GLANCEAPPURL?name=FILENAME&url=FILEURL
  to pass name and url to the web server
- E.g. using ParaView Glance website
  https://kitware.github.io/paraview-glance/app?name=sineEnvelope.vtkjs&url=https://raw.githubusercontent.com/razoumov/publish/master/data/sineEnvelope.vtkjs
  - shortened to https://bit.ly/2KtPWNf
- You can parse long strings with JavaScript (next slide)

Embed your vis into a website with an iframe (`embed.html`)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sine envelope function</title>
  </head>
  <body>
    <h1>3D sine envelope function</h1>

    <script>
      var app = "https://kitware.github.io/paraview-glance/app";
      var dir = "https://raw.githubusercontent.com/razoumov/publish/master/data/";
      var file = "sineEnvelope.vtkjs";
      document.write("<iframe src='" + app + "?name=" + file + "&url=" +
                     dir + file +
                     "' id='iframe' width='1100' height='900'></iframe>");
    </script>

    <p>More stuff in here</p>
  </body>
</html>
```

- JavaScript here only to parse long strings

Animation in ParaView

Animation methods

1. Use ParaView's built-in animation of any property of any pipeline object
   - easily create snazzy animations, somewhat limited in what you can do
   - in Animation View: select object, select property, create a new track with "+", double-click the track to edit it, press "▶"

Animation methods

1. Use ParaView's built-in animation of any property of any pipeline object
   - easily create snazzy animations, somewhat limited in what you can do
   - in Animation View: select object, select property, create a new track with "+", double-click the track to edit it, press "▶"
2. Use ParaView's ability to recognize a sequence of similar files
   - time animation only, very convenient
   - try loading `data/2d*.vtk` sequence and animating it (visualize one frame and then press "▶")

Animation methods

1. Use ParaView's built-in animation of any property of any pipeline object
   - easily create snazzy animations, somewhat limited in what you can do
   - in Animation View: select object, select property, create a new track with "+", double-click the track to edit it, press "▶"
2. Use ParaView's ability to recognize a sequence of similar files
   - time animation only, very convenient
   - try loading `data/2d*.vtk` sequence and animating it (visualize one frame and then press "▶")
3. Script your animation in Python (covered in next section)
   - steep learning curve, very powerful, can do anything you can do in the GUI
   - typical usage scenario: generate one frame per input file
   - a simpler exercise without input files: see next slide

Exercise: animating function growth

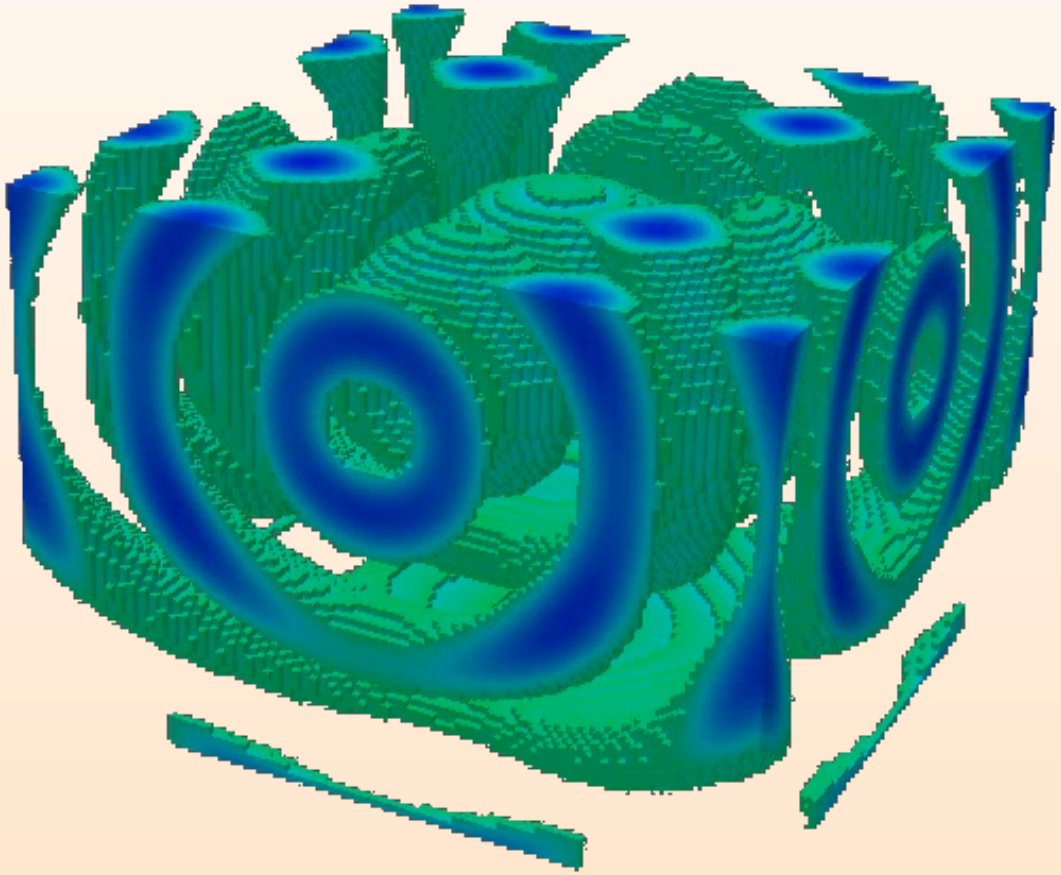
- 3D sine envelope wave function defined inside a unit cube ($x_i \in [0, 1]$)

$$f(x_1, x_2, x_3) = \sum_{i=1}^{2} \left[\frac{\sin^2\left(\sqrt{\xi_{i+1}^2 + \xi_i^2}\right) - 0.5}{\left[0.001(\xi_{i+1}^2 + \xi_i^2) + 1\right]^2} + 0.5 \right], \text{ where } \xi_i \equiv 15(x_i - 0.5)$$

- Reproduce the movie on the screen

  `https://vimeo.com/248501176`

  or `hidden/growth.mp4` on presenter's laptop



Exercise: animating function growth (cont.)

To visualize a single frame of the movie:

1. load `data/sineEnvelope.nc` (discretized on a 100^3 grid)
2. apply Threshold keeping only data from 1.2 to 2
3. apply Clip: origin $O = (49.5, 15, 49.5)$, normal $N = (0, -1, 0)$
4. colour by the right quantity

Two possible solutions:

1. bring up **Animation View** to animate Clip's O_2 from 0 to 99, for best results save animation as a sequence of PNG files
2. covered in the next section: Start/Stop Trace to record the workflow, save the corresponding **Python script**, enclose parts of it into a loop changing O_2 from 0 to 99 and writing a series of PNG screenshots, run it inside ParaView to produce 100 frames

   in either case, merge PNGs using a 3rd-party tool, e.g.

```
ffmpeg -r 30 -i frame%04d.png -c:v libx264 -pix_fmt yuv420p \
       -vf "scale=trunc(iw/2)*2:trunc(ih/2)*2" movie.mp4
```

Camera animation in the GUI

Good introductory resource `https://www.paraview.org/Wiki/Advanced_Animations`

1. Start with any static visualization
2. Click on 'Adjust Camera' icon (one of the left-side icons on top of the visualization window)
   - adjust / write down Camera Focal Point
3. Bring up Animation View (or erase all previous timelines)

(3a) In Animation View:

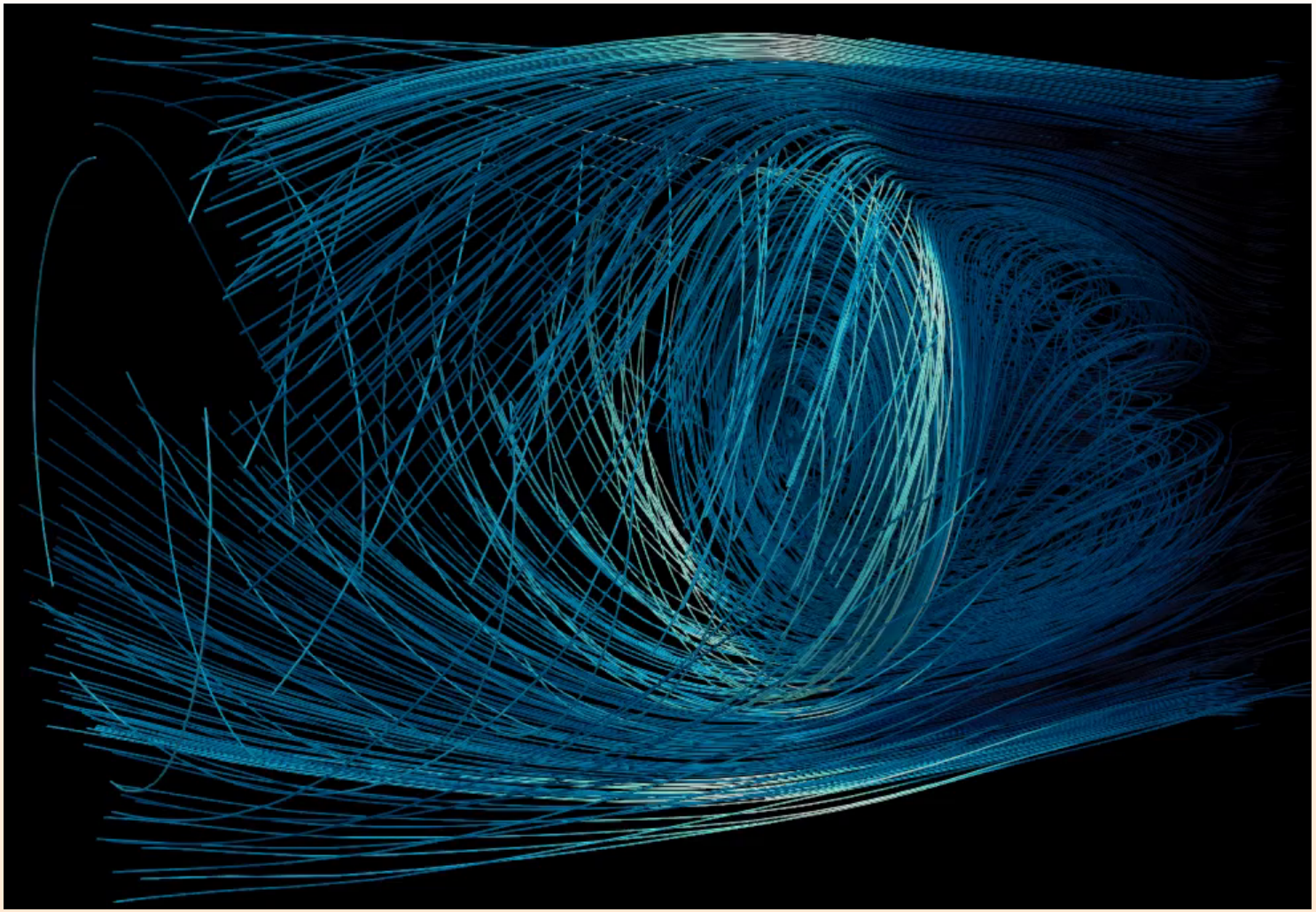
- select Camera - Orbit
- click "+" to create a new timeline
- set Center = Camera Focal Point, for the rest accept default settings
- adjust the number of frames

(3b) In Animation View:

- select Camera - Follow Path
- click "+" to create a new timeline
- double-click on the white (or black) timeline
- double-click on Path... in the right column
- click on Camera Position
  - a yellow path with spheres will appear
  - drag the spheres around
- also can change Camera Focus and Up Direction

4. Click "▶"

Animating stationary flow: streamlines through a slice

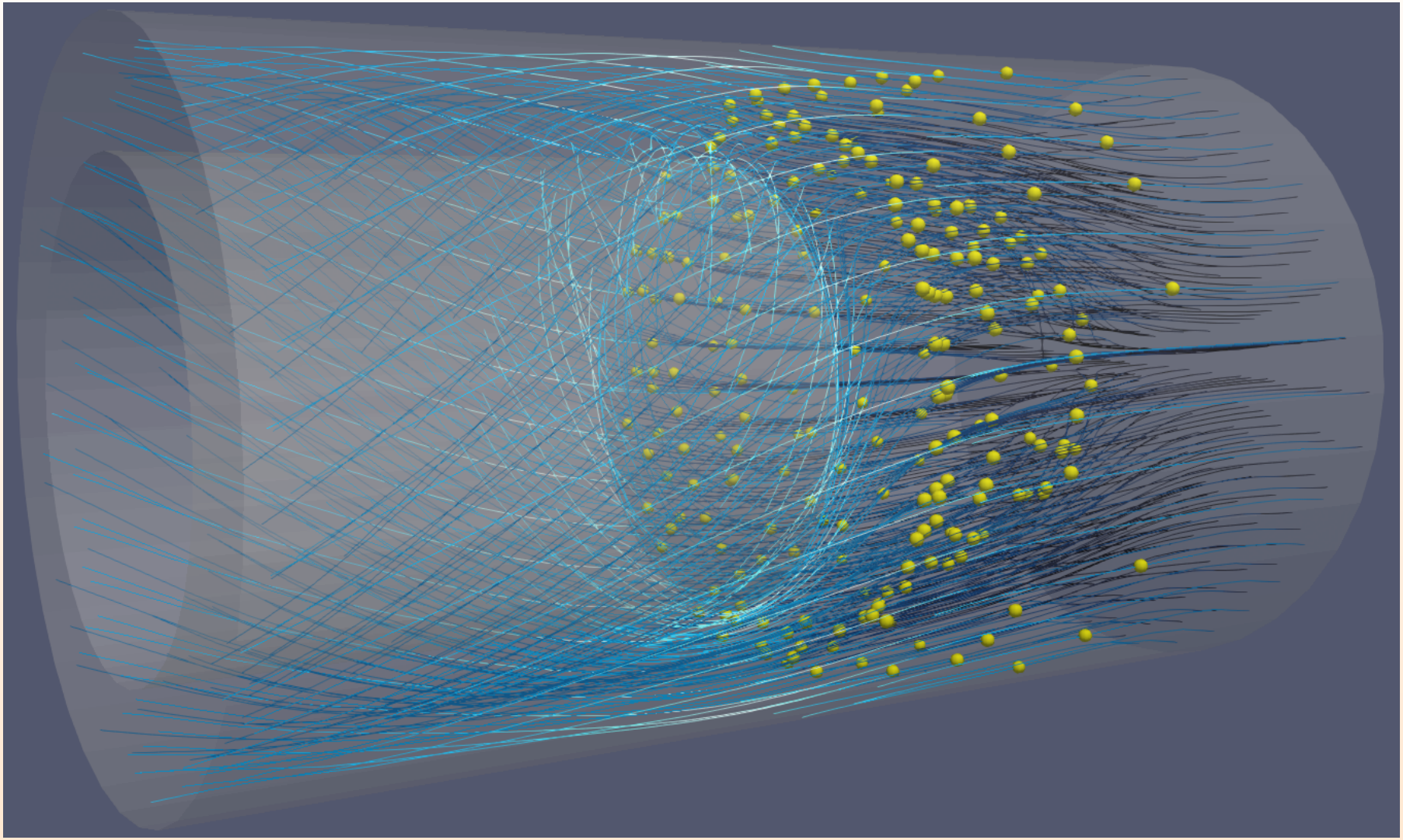


- `https://vimeo.com/248501893` or `hidden/radialSlice.mp4` on presenter's laptop
- `https://vimeo.com/248502086` or `hidden/xySlice.mp4` on presenter's laptop

Animating stationary flow: streamlines through a slice (cont.)

1. Load `disk_out_ref.ex2` making sure to load velocity
2. Draw a radius-z plane slice through the center, origin $O = (0, 0, 0)$ and normal $N = (1, 0, 0)$
3. Stream Tracer With Custom Source: *input*=`disk_out_ref.ex2`, *seedSource*=`Slice1`
4. Tube filter with $r = 0.015$
5. Animation View: animate Slice's O_0 from -1 to 1 (full range [-5.75,5.75])
6. Use 100 frames, black background, blue2cyan colourmap, colour with vorticity
7. Unselect "Show Plane"
8. Save animation as PNGs, encode at 10 fps

Animating a stationary flow: time contours



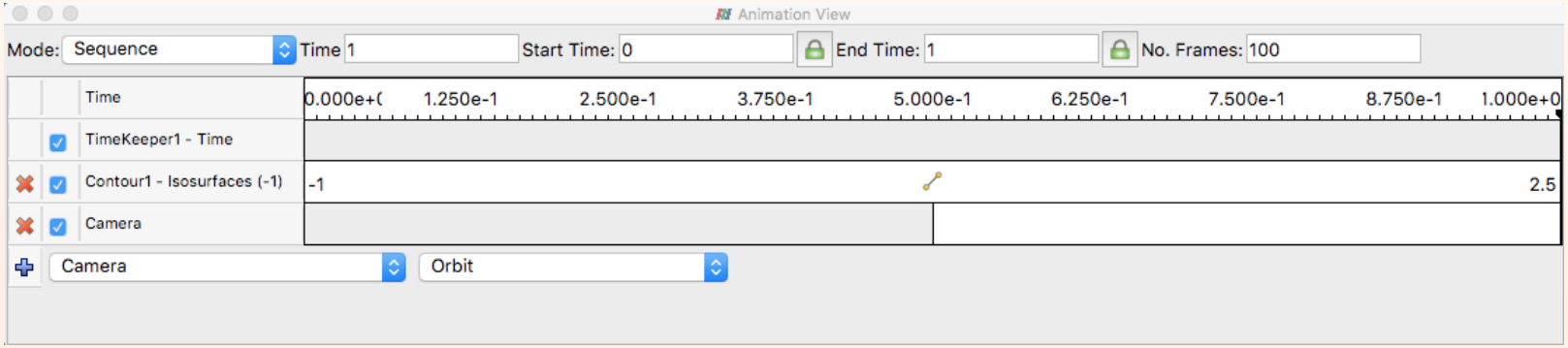
`https://vimeo.com/248509153` or `hidden/timeContours.mp4` on presenter's laptop

Animating a stationary flow: time contours (cont.)

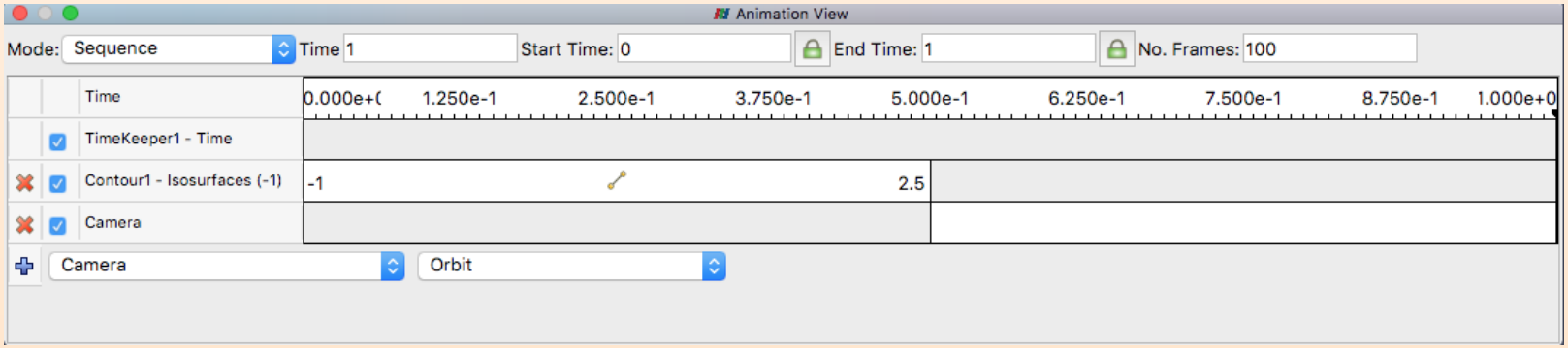
1. Start with the streamtracer lines, however drawn
2. Apply a Countour filter to the output of Streamtracer
   - contour by Integration Time
   - probe the range of values that works best
3. Apply Glyph filter to the output of Countour
4. Animation View: animate Contour | Isosurfaces
5. This video was recorded with 2000 frames at 60 fps
   - such high resolution only for the final production video
   - debugging animation with 100 frames is perfectly Ok

Exercise: several timelines in one animation

1. Start with the previous integration-time-contour animation
2. Add the second timeline to the animation: Camera - Orbit from $t = 0.5$ to $t = 1$ (while the first animation is still playing for its second half)

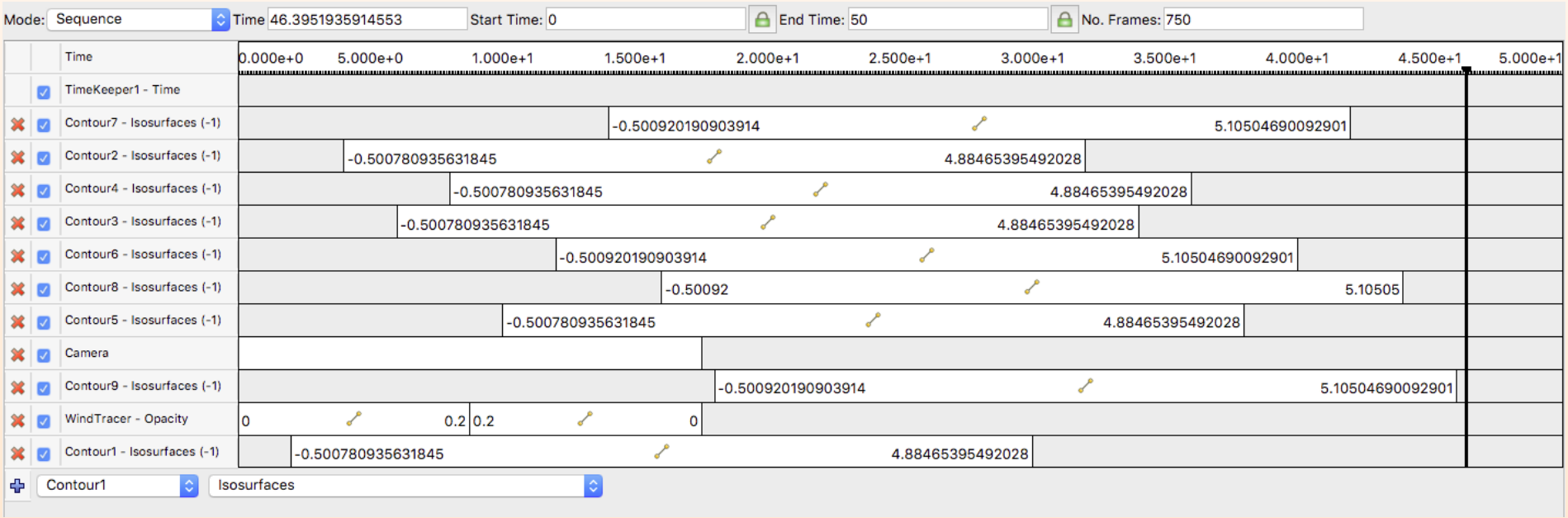


3. Now complete integration-time-contour animation before rotation



Combining many timelines in one animation (cont.)

- In principle, can add as many timelines (with their individual time intervals and variables!) to the animation as you want
- Here is an example from WestGrid's 2017 *Visualize This* competition submission by Nadya Moisseeva (UBC)



`hidden/complexAnimation.mp4` on presenter's laptop

Python scripting in ParaView

Batch scripting for automating visualization

Official documentation at `https://www.paraview.org/Wiki/ParaView/Python_Scripting`

- Why use scripting?
  - automate mundane or repetitive tasks, e.g., making frames for a movie
  - document and store your workflow
  - use ParaView on clusters from the command line and/or via batch jobs
- In the GUI: View | Python Shell opens a Python interpreter
  - write or paste your script there
  - use the button to run an external script from a file
- *[/usr/bin/ /usr/local/bin/ /Applications/Paraview*.app/Contents/bin/]* **pvpython** will give you a Python shell connected to a ParaView server (local or remote) without the GUI
- *[/usr/bin/ /usr/local/bin/ /Applications/Paraview*.app/Contents/bin/]* **pvbatch** *--force-offscreen-rendering script.py* is a serial (on some machines parallel) application using a local ParaView server ✐ make sure to save your visualization
- *[/usr/bin/ /usr/local/bin/ /Applications/Paraview*.app/Contents/MacOS/]* *paraview --script=codes/displayWireframe.py* to start ParaView GUI and auto-run the script

First script

- Bring up View | Python Shell
- "Run Script" `codes/displaySphere.py`

displaySphere.py

```
from paraview.simple import *

sphere = Sphere() # create a sphere pipeline object

print(sphere.ThetaResolution) # print one of the attributes of the sphere
sphere.ThetaResolution = 16

Show() # turn on visibility of the object in the view
Render()
```

- Can always get help from the command line

```
help(paraview.simple)     # will display a help page on paraview.simple module
help(Sphere)
help(Show)
help(sphere)   # to see this object's attributes
dir(paraview.simple)
```

Using filters

- "Run Script" `codes/displayWireframe.py`

displayWireframe.py

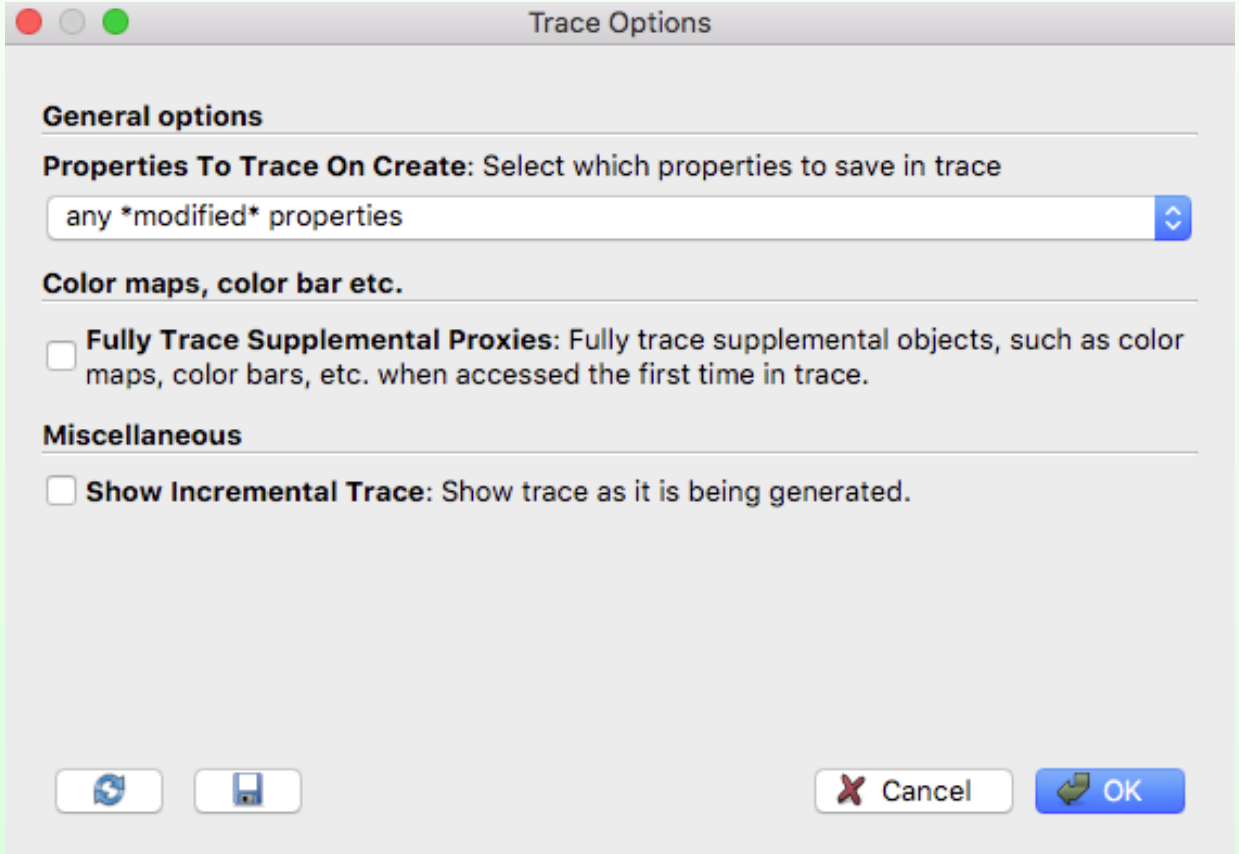
```
from paraview.simple import *

sphere = Sphere(ThetaResolution=36, PhiResolution=18)

wireframe = ExtractEdges(Input=sphere) # apply Extract Edges to sphere

Show() # turn on visibility of the last object in the view
Render()
```

- Try replacing Show() with Show(sphere)
- Also try replacing `Render()` with `SaveScreenshot('/path/to/wireframe.png')` and running via `pvbatch`

Trace tool

Generate Python code from GUI operations

- Newer ParaView: Tools | Start / Stop Trace
- Older ParaView: Tools | Python Shell | Trace | Start / Stop / Show Trace



Passing information down the pipeline

... and other useful high-level workflow functions

- `GetSources()` gets a list of pipeline objects
- `GetActiveSource()` gets the active object
- `SetActiveSource()` sets the active object
- `GetRepresentation()` returns the *view representation* for the active pipeline object and the active view
- `GetActiveCamera()` returns the active camera for the active view
- `GetActiveView()` returns the active view
- `CreateRenderView()` creates standard 3D render view
- `ResetCamera()` resets the camera to include the entire scene but preserve orientation (or does nothing ☺)

There is quite a bit of overlap between these two:

```
help(GetActiveCamera())
help(GetActiveView())
```

Camera animation with scripting

1. Let's load `data/sineEnvelope.nc` and draw an isosurface at $\rho = 0.15$

2. Compare the focal point to the center of rotation (must be the same for object to stay in view)

```
v1 = GetActiveView()
print(v1.CameraFocalPoint)
print(v1.CenterOfRotation)
```

   if not $\Rightarrow$ `ResetCamera()`

3. Look up azimuthal rotation

```
dir(GetActiveCamera())
help(GetActiveCamera().Azimuth)
```

4. Rotate by 10° around the view-up vector

```
camera = GetActiveCamera()
camera.Azimuth(10)
Render()
```

Camera animation: full rotation

✐ Can paste longer commands from `clipboard.txt`

5. Do full rotation and save to disk

```
nframes = 360
for i in range(nframes):
    print(v1.CameraPosition)
    camera.Azimuth(360./nframes)    # rotate by 1 degree
    SaveScreenshot('/path/to/frame%04d'%(i)+'.png')
```

6. Merge all frames into a movie at 30 fps

```
ffmpeg -r 30 -i frame%04d.png -c:v libx264 -pix_fmt yuv420p \
       -vf "scale=trunc(iw/2)*2:trunc(ih/2)*2" spin.mp4
```

Camera animation: flying towards the focal point

1. Optionally reset the view manually or with `ResetCamera()`
2. Now let's fly 2/3 of the way towards the focal point

```
initialCameraPosition = v1.CameraPosition[:]   # force a real copy
nframes = 100
for i in range(nframes):
    coef = float(i+0.5)/float(1.5*nframes)  # runs from 0 to 2/3
    print(coef, v1.CameraPosition)
    v1.CameraPosition = [((1.-coef)*a + coef*b) \
      for a, b in zip(initialCameraPosition,v1.CameraFocalPoint)]
    SaveScreenshot('/path/to/out%04d'%(i)+'.png')
```

3. Create a movie

```
ffmpeg -r 30 -i out%04d.png -c:v libx264 -pix_fmt yuv420p \
       -vf "scale=trunc(iw/2)*2:trunc(ih/2)*2" approach.mp4
```

Exercise: write and run a complete off-screen script

1. Mac/Linux/Windows: create a script with standalone ParaView GUI
   - use Start/Stop Trace
   - load `data/sineEnvelope.nc` and draw an isosurface at $\rho = 0.15$
   - save the image as PNG
2. Test-run your script with `pvbatch` on your laptop

```
$ pvbatch --force-offscreen-rendering script.py
```

   - Linux: `pvbatch` should be in one of your system's `bin` directories
   - Mac: `pvbatch` should be in `/Applications/ParaView*.app/Contents/bin`
   - Windows: `pvbatch` does not exist (or so I am told), but you can use `pvpython`
     ➠ you will need to locate it yourself
   - those of you with a Compute Canada account can run this script on one of our HPC clusters with

```
$ module load gcc/9.3.0 paraview-offscreen/5.10.0
$ pvbatch --force-offscreen-rendering script.py
```

3. Modify the script to create some animation

Extracting data from VTK objects

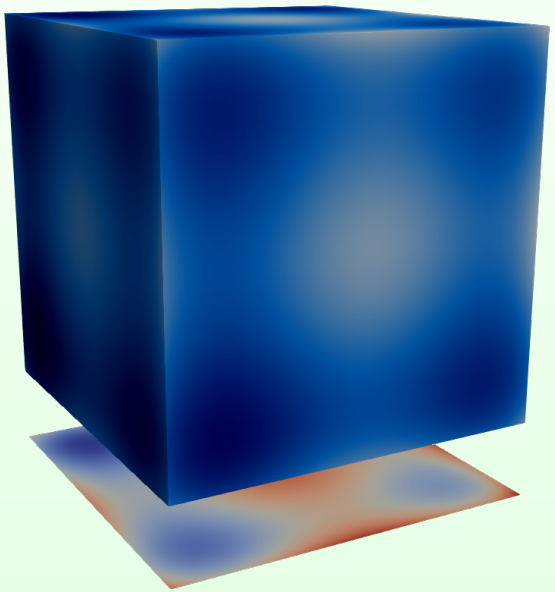
Do this from *View | Python Shell* or from *pvpython* (either shell will work)

```
# codes/extractValues.py
from paraview.simple import *

dir = '/Users/razoumov/training/paraviewWorkshop/data/'
data = NetCDFReader(FileName=[dir+'stvol.nc'])
local = servermanager.Fetch(data) # get the data from the server
print(local.GetNumberOfPoints())

for i in range(10):
    print(local.GetPoint(i))   # coordinates of first 10 points

pd = local.GetPointData()
print(pd.GetArrayName(0))   # the name of the first array

result = pd.GetArray('f(x,y,z)')
print(result.GetDataSize())
print(result.GetRange())

for i in range(10):
    print(result.GetValue(i))   # values at first 10 points
```

This is useful for post-processing, e.g., feeding these into **numpy arrays** and doing further calculations in a Python script

Creating/modifying VTK objects

Let's say we want to plot a projection of a cubic dataset along one of its principal axes, or do some other transformation for which there is no filter



- Calculator / Python Calculator filter cannot modify the geometry ...

Programmable filter

Watch our webinar `https://bit.ly/programmablefilter`

1. Apply Programmable Filter with OutputDataSetType = vtkUnstructuredGrid
2. Paste the following code `codes/projectionUnstructured.py` into the filter (this code was tested in ParaView 5.10.1)

```
numPoints = inputs[0].GetNumberOfPoints()
side = int(round(numPoints**(1./3.)))    # round() in this Python returns float type
layer = side*side
rho = inputs[0].PointData['density']     # 1D flat array
points = vtk.vtkPoints()       # create vtkPoints instance, to contain 100^2 points in the projection
proj = vtk.vtkDoubleArray(); proj.SetName('projection')   # create the projection array
for i in range(layer):         # loop through 100x100 points
    x, y = inputs[0].GetPoint(i)[0:2]
    z, column = -20., 0.
    for j in range(side):
        column += rho.GetValue(i+layer*j)
    points.InsertNextPoint(x,y,z)        # also points.InsertPoint(i,x,y,z)
    proj.InsertNextValue(column)         # add value to this point

output.SetPoints(points)                 # add points to vtkUnstructuredGrid
output.GetPointData().SetScalars(proj)   # add projection array to these points

quad = vtk.vtkQuad()                # create a cell
output.Allocate(side, side)         # allocate space for side^2 'cells'
for i in range(side-1):
    for j in range(side-1):
        quad.GetPointIds().SetId(0,i+j*side)
        quad.GetPointIds().SetId(1,(i+1)+j*side)
        quad.GetPointIds().SetId(2,(i+1)+(j+1)*side)
        quad.GetPointIds().SetId(3,i+(j+1)*side)
        output.InsertNextCell(vtk.VTK_QUAD, quad.GetPointIds())
```

Using 3rd-party libraries from ParaView's Python

- `pvpython` includes few common 3rd-party libraries such as `numpy,scipy,pandas`
- What if you want to use other libraries that were not bundled with ParaView?

Using 3rd-party libraries from ParaView's Python

- `pvpython` includes few common 3rd-party libraries such as `numpy`, `scipy`, `pandas`
- What if you want to use other libraries that were not bundled with ParaView?

---

1. Let's assume you work on a CC cluster; check your ParaView's Python version

```
module load gcc/9.3.0 paraview/5.10.0
pvpython   # let's assume it says Python 3.9.6
```

2. Load the closest Python module, create a virtual env. and install your library there

```
module avail python   # python/3.9.6 is one of them
module load python/3.9.6
virtualenv --no-download astro   # this will install a new virtual environment into ~/astro
source ~/astro/bin/activate
pip install --no-index --upgrade pip
pip install --no-index xarray   # install an external package into this new environment
```

3. Next time you log in to the cluster, start `pvpython`:

```
module load gcc/9.3.0 paraview/5.10.0
pvpython
```

4. Load your new virtual environment directly from Python:

```
filename = '/home/username/astro/bin/activate_this.py'
exec(open(filename).read(), {'__file__': filename})
from paraview.simple import *
import xarray     # this xarray comes from your new virtual environment
```

Remote and distributed visualization

Visualizing remote data

If your dataset is on a remote cluster, there are several options:

✘ **download data** to your desktop and visualize it locally
- limited by the dataset size and your desktop's CPU/GPU + memory

✘ run ParaView remotely on a larger machine via **X11 forwarding**
- your desktop $\xrightarrow{\text{ssh -Y}}$ larger machine running ParaView
- remote OpenGL apps with either (1) software rasterizer on the cluster (usually the default) or (2) on your laptop's GPU (need to re-enable INdirect GLX inside X11 server and set `LIBGL_ALWAYS_INDIRECT=1`)

✔ run ParaView remotely on a larger machine **via remote desktop**
- your desktop $\xrightarrow{\text{VNC}}$ larger machine running ParaView
- you can always start a VNC server on an interactive cluster compute node by hand as described in our documentation `https://bit.ly/startVNC`
- remote OpenGL apps will run either (1) using software rasterizer on the cluster (usually the default) or (2) on cluster's GPU(s) via VirtualGL wrapper (see our VNC docs)
- the VNC slide is coming up

✔ run ParaView in **client-server mode**
ParaView client on your desktop $\rightleftharpoons$ ParaView server on larger machine

✔ run ParaView via a GUI-less **batch script** (interactively or scheduled)
- render server can run with GPU rendering or purely in software
- data/render servers can run on single-core, or across several cores/nodes with MPI
- for interactive GUI work on clusters you should schedule interactive jobs, as opposed to running on the login nodes

Special remote vis cases

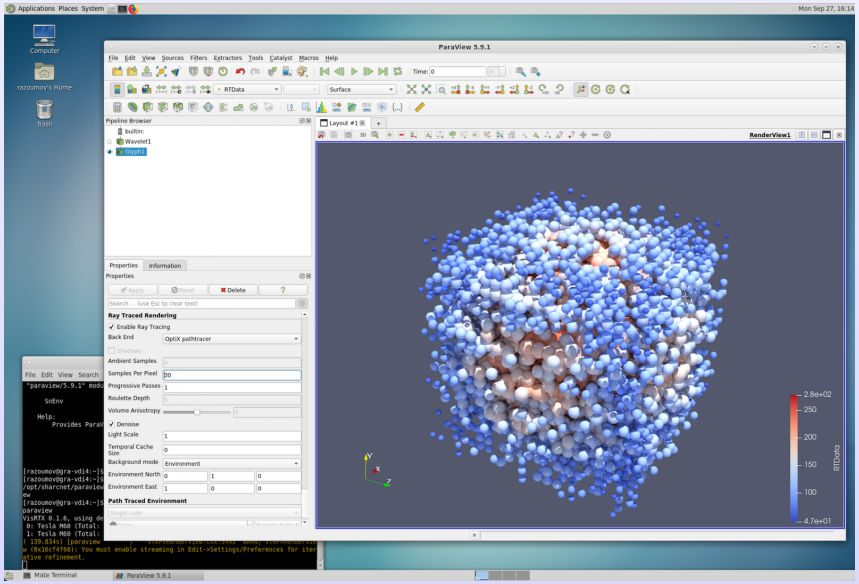
1. In-situ visualization = instrumenting a simulation code on the cluster to
   1.1 output graphics and/or
   1.2 act as on-the-fly server for a visualization frontend (ParaView/VisIt client on your laptop)
   - need to use a special library (ParaView's Catalyst or VisIt's libsim)
   - very advanced topic for another time
2. Web-based visualization with data served from another location

ParaView via remote desktop

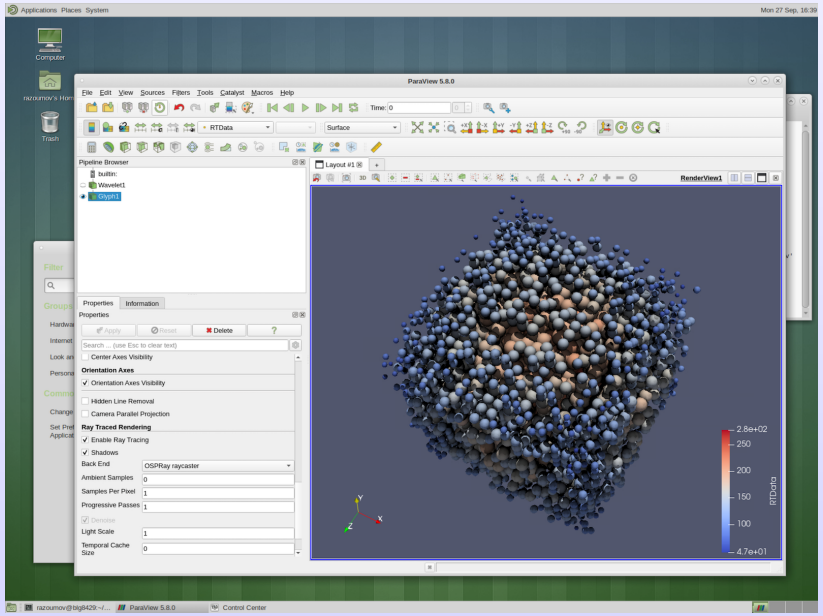
https://docs.alliancecan.ca/wiki/VNC or https://docs.alliancecan.ca/wiki/JupyterHub

You have several options:

(1) run a VNC server on compute nodes with SSH tunnelling, connect via a VNC client
https://docs.alliancecan.ca/wiki/VNC#Compute_Nodes



(2) VNC on gra-vdi.computecanada.ca, connect via a VNC client



(3) Remote Desktop via JupyterHub on Béluga, point your web browser at https://jupyterhub.beluga.computecanada.ca

Cedar, Graham, Béluga, Narval clusters

- General-purpose CC clusters for a variety of workloads
  - entered production in phases since June 2017
  - located at SFU, UofWaterloo, École de technologie supérieure (Montreal)
  - 101,568 / 44,444 / 39,120 / 80,720 CPUs
  - many hundred NVIDIA GPUs (with 12GB/16GB/32GB on-board memory)
  - multiple types of nodes, with 128GB/256GB/0.5TB/1.5TB/3TB memory
  - specs at `https://docs.alliancecan.ca/wiki/Cedar` (replace `Cedar` with `Graham` or `Beluga` or `Narval`)
- Batch-oriented environment for parallel and serial jobs $\Rightarrow$ use Slurm scheduler and workload manager
- Identical software setup `https://docs.alliancecan.ca/wiki/Available_software`

Interactive jobs on Cedar / Graham / Béluga

- **Client-server workflow** is by definition interactive
- On Cedar interactive jobs should automatically go to one of Slurm's interactive partitions (CPU or GPU)

```
$ sinfo -p cpubase_interac
    # will list nodes and their states (idle, mixed, allocated, ...)
```

- `salloc` without a script name will start an interactive shell inside a submitted job on a compute node

```
$ salloc --time=1:0:0 --ntasks=4 ... --account=def-someuser
$ echo $SLURM_...    # access Slurm variables
$ module load ...    # set your environment
$ ./serialCode
$ srun ./mpiCode     # run an MPI code
$ exit               # terminate the job (go back to the login node)
```

- You might need to specify `pvserver --server-port=11112` (etc.) if someone else is already using the default port 11111 on the same node

Question 1: should I use CPUs or GPUs for rendering?

- Can render on GPUs (*hardware acceleration*) or CPUs (*software rendering*) with both interactive and batch visualizations
  - GPUs have traditionally been faster for rendering graphics
  - in recent years better open-source software rendering libraries such as OSPRay (Intel's ray tracing) and OpenSWR (Intel's rasterizer) have largely closed the performance gap for many types of visualizations

  ⇒ I recommend starting with CPU rendering since you already likely have many CPUs! (see next slide)

- One might have to resort to software rendering if no GPUs are available, e.g., all taken by GP-GPU jobs

- I suggest doing **all hands-on exercises with CPU rendering**; also included slides on **GPU rendering** on the cluster

Question 2: how many CPUs/GPUs do I need?

- How many processors do we need? From *ParaView documentation*:
  - structured data (Structured Points, Rectilinear Grid, Structured Grid): one CPU core per ~20 million cells
  - unstructured data (Unstructured Points, Polygonal Data, Unstructured Grid): one CPU core per ~1 million cells
- Your main bottlenecks will be physical memory and disk read speed, and to a lesser extent CPU/GPU rendering time $\Rightarrow$ to simplify things, to decide on the number of CPU cores for initial dataset exploration, use the dataset size
  - consider 80 GB dataset
  - base nodes have 128 GB memory with 32 cores $\Rightarrow$ 3.5 GB/core (accounting for the OS, system tools, etc.) $\Rightarrow$ 23 cores for this dataset
  - need to account for filters (and other processing), MPI buffers $\Rightarrow$ minimum 32 cores
  - for comfortable processing with complex filters use $48 - 64$ cores
- On large HPC systems ParaView is known to scale to $\sim 10^{12}$ cells (Structured Points) on ~10,000 cores and beyond
- Always do a scaling study before attempting to visualize large datasets
- It is important to understand memory requirements of filters
  - a typical structured $\rightarrow$ unstructured filter increases memory footprint by $\sim$ 3X

Remote Render Threshold

In ParaView's preferences can set Render View → Remote/Parallel Rendering Options → Remote Render Threshold beyond which rendering will be remote

- default 20MB ⇒ small rendering will be done on your laptop's GPU, interactive rotation with a mouse will be fast, but anything modestly intensive (under 20MB) will be shipped to your laptop and might be slow
- 0MB ⇒ all rendering (including rotation) will be remote, so you will be really using the cluster's CPU(s)/GPU(s) for everything
  - good for large data processing
  - not so good for interactivity, especially on a slower connection
- experiment with the threshold to find a suitable value

Next few pages: remote rendering exercises

Short version:

1. create your visualization via interactive client-server using CPU rendering
2. save your visualization to PNG

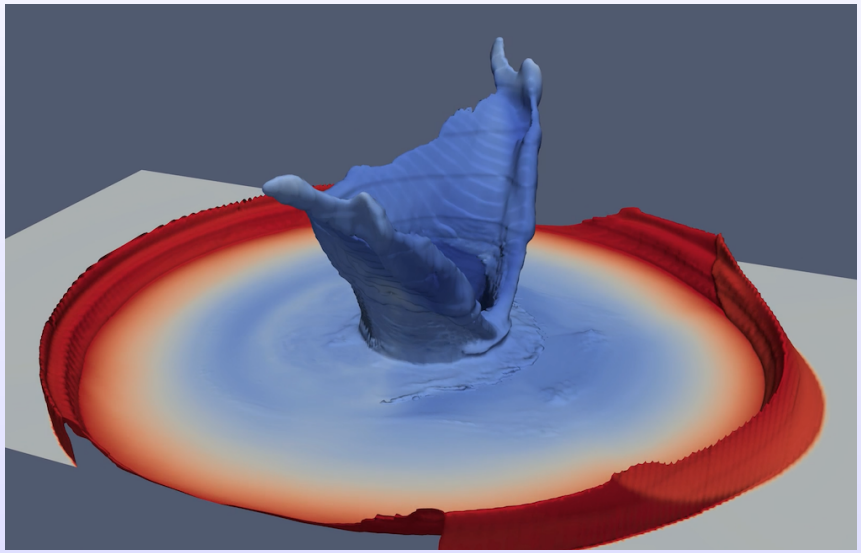
Long version:

1. create your visualization via interactive client-server using CPU rendering
2. save your visualization to PNG
3. convert this workflow into a Python script
4. upload this Python script to the cluster
5. try running the script inside an interactive (`salloc`) job; debug if needed
6. once happy with the result, write a Slurm job submission script and submit this rendering as a batch (`sbatch`) job

Exercise 1 (on Cedar): deep impact dataset

Dataset from IEEE 2018 SciVis Contest

- Dataset from *Deep Water Impact* simulation by John Patchett (LANL) and Galen Gisler (Univ. of Oslo)
  - dataset details at `https://bit.ly/2SXmjsq`
  - you can work with 269 low-resolution ($460 \times 280 \times 240$) snapshots in time
  - the original simulation is much higher resolution



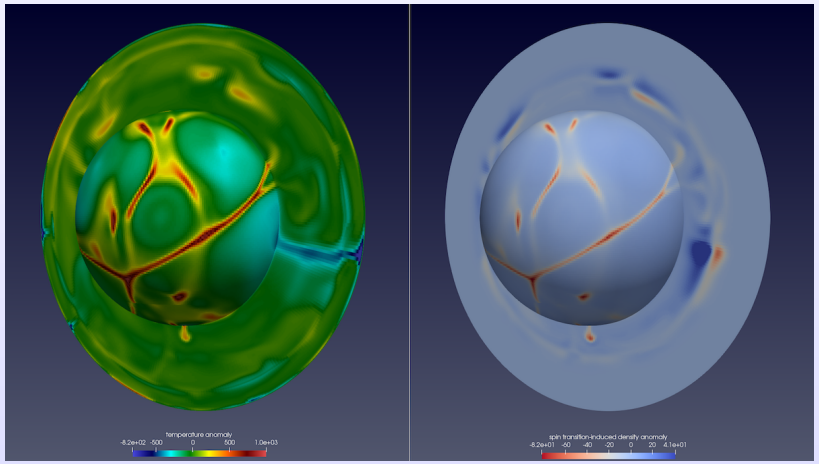
- You can render this dataset in serial
  - try to adapt the client-server instructions from "Parallel software rendering" slide (forward a few pages) to render on one CPU
- Data in `cedar:/project/6003910/razoumov/ieeevis2018/460x280x240` (115GB in total)
- To simplify navigating to the dataset in ParaView, I highly recommend creating a symbolic link:

```
[cedar]$ mkdir -p ~/data
[cedar]$ ln -s /project/6003910/razoumov/ieeevis2018/460x280x240/ ~/data/deepImpact
```

Exercise 2 (on Cedar): Earth's mantle convection

Dataset from IEEE 2021 SciVis Contest `https://scivis2021.netlify.app`

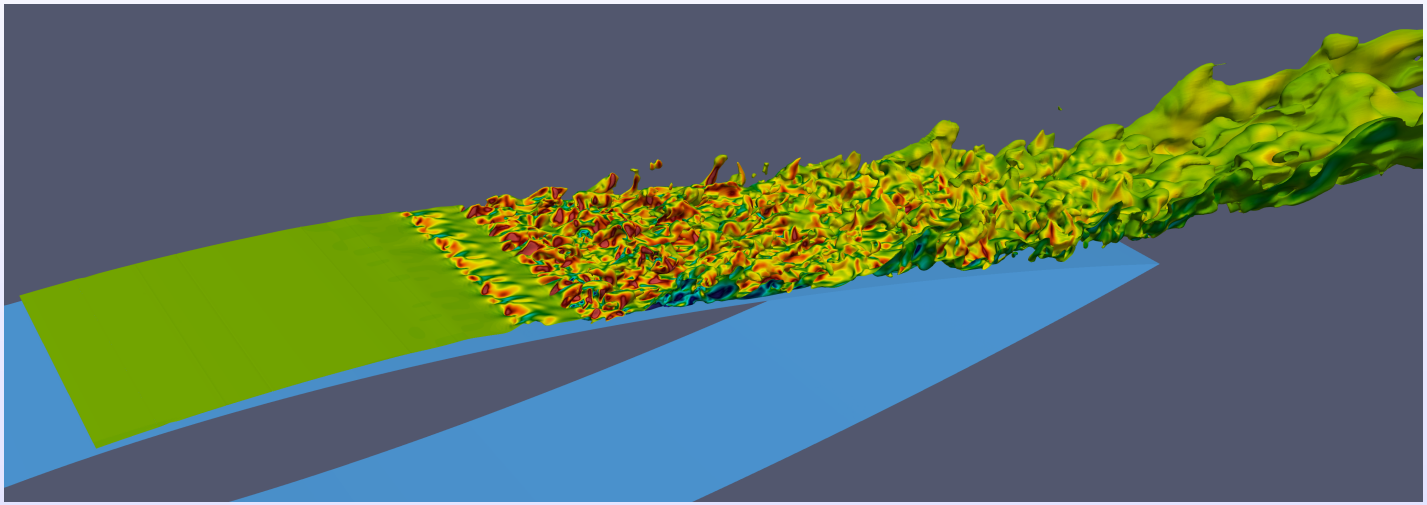
- Dataset from *Earth's Mantle Convection* simulation by Hosein Shahnas and Russell Pysklywec (U. of Toronto)
  - dataset details at `https://scivis2021.netlify.app/data`
  - 251 timesteps on a spherical $180 \times 201 \times 360$ grid
- You can render this dataset in serial
  - try to adapt the client-server instructions from "Parallel software rendering" slide (forward a few pages) to render on one CPU



- Data in `cedar:/project/6003910/razoumov/ieeevis2021/spherical` (89GB in total)
- Create a symbolic link to simplify navigating to the dataset in ParaView

Exercise 3 (on Cedar): airflow over a turbine blade

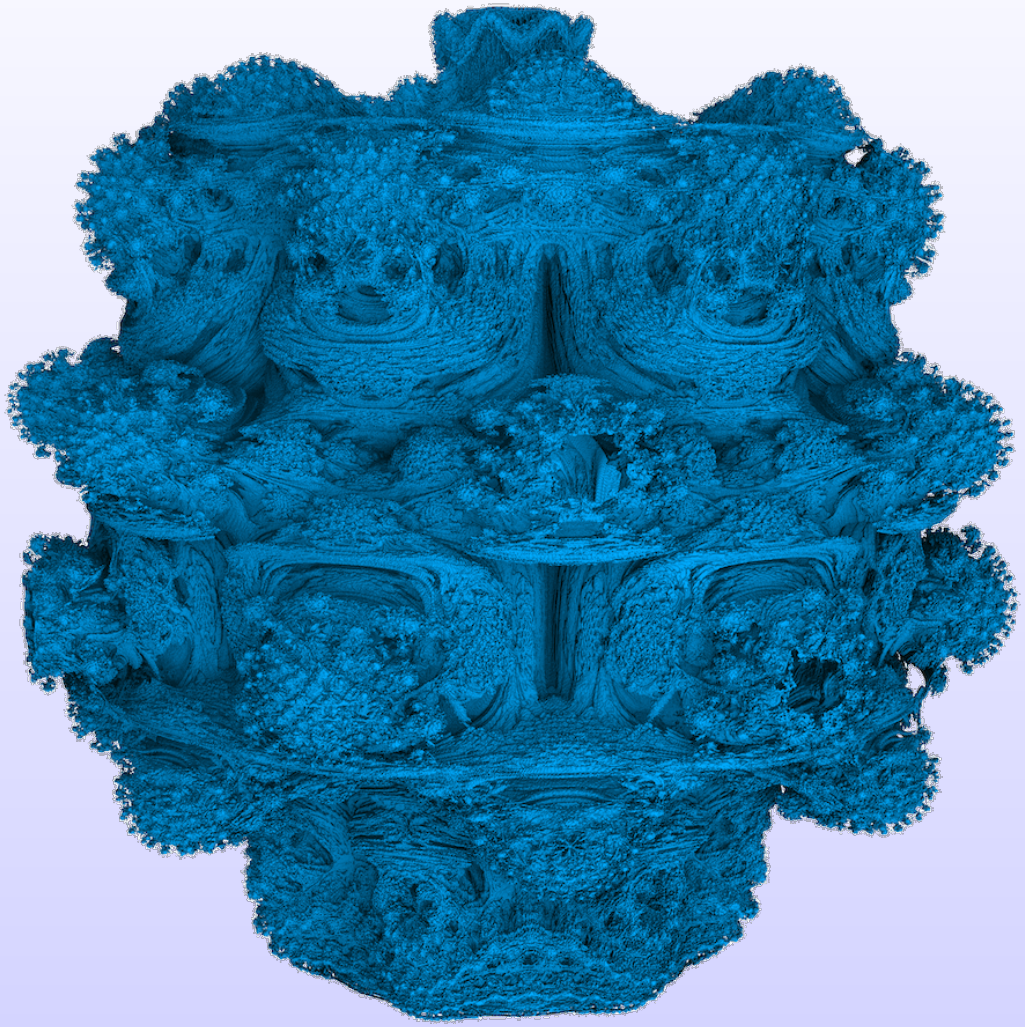
Dataset from WestGrid's 2019 `https://computecanada.github.io/visualizeThis`



- OpenFOAM *decomposed* dataset: 512 cores, 86 timesteps, 5 hydro variables, ∼1TB in total
  - kindly provided for this competition by Joshua Brinkerhoff (UBC Okanagan)
  - unstructured mesh ⇒ loading a single timestep from the 3D internal mesh requires 200GB+ physical RAM
  - the 2D airfoil mesh takes only 13.7 GB virtual memory for 1 timestep + 1 variable
  - data in `cedar:/project/6003910/razoumov/visThis2019`
- Image at the top shows the isosurface of constant air speed coloured by the Y-component of the vorticity, full animation rendering (86 timesteps) took 17 minutes on 128 Cedar CPU cores
- Create a symbolic link to simplify navigating to the dataset in ParaView

Exercise 4 (on the training cluster): Mandelbulb

- Visualize power-8 Mandelbulb
- Use the file `mandelbulb800.nc` – now sampled at 800^3
- Use 4–8 CPU cores on the training cluster via `salloc`
  1. consult the next three pages, use critical thinking – you will need to modify some of the commands!
  2. try to recreate the picture on the right: pay attention to the lights and shadows
  3. use View → Memory Inspector to keep an eye on memory usage
  4. optionally colour your dataset by `processID`



```
$ unzip /home/razoumov/shared/paraview.zip data/mandelbulb800.nc
$ ls -lh data/mandelbulb800.nc
```

Parallel software rendering

From interactive client-server debugging to remote batch rendering

1. On the cluster start remote parallel ParaView server:

```
$ cd scratch   # necessary on Cedar
$ module load StdEnv/2020 gcc/9.3.0 openmpi/4.0.3 paraview-offscreen/5.10.0
$ salloc --time=0:60:0 --ntasks=128 --mem-per-cpu=3600 --account=def-someuser
$ mpirun -np 128 pvserver
```

2. Wait for it to start waiting for incoming connection:

```
Waiting for client...
Connection URL: cs://cdr774.int.cedar.computecanada.ca:11111
Accepting connection(s): cdr774.int.cedar.computecanada.ca:11111
```

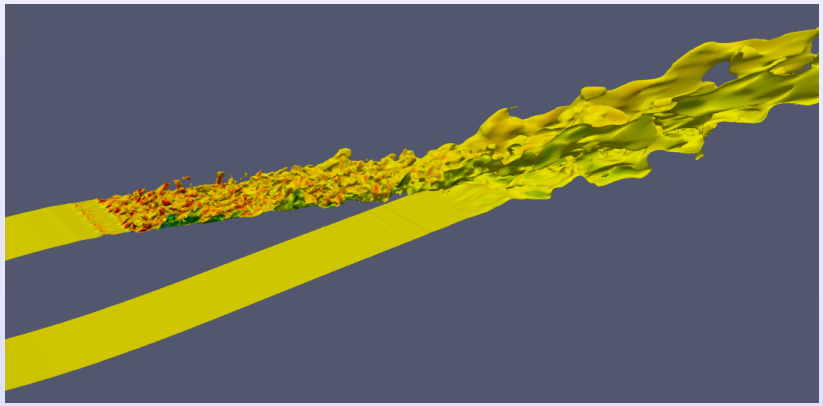
3. On your laptop start SSH port forwarding:

```
$ ssh cedar.computecanada.ca -L 11111:cdr774:11111  # use the actual compute node
```

4. On your laptop start ParaView 5.10.x, click Connect, then connect to `cs://localhost:11111`

Parallel software rendering (cont.)

5. Tools → Start Trace
6. Load OpenFOAM data, set Case Type = Decomposed
7. Apply Calculator: speed = mag(U)
8. Apply Contour at `speed=0.8`
9. Colour by $(\text{vorticity})_y$
10. Load *Rainbow Desaturated* colourmap
11. Save the image as a PNG file
12. Tools → Stop Trace
13. Save the generated script as `airflow.py` locally
    - edit it in a text editor, simplify (most generated lines will be setting defaults)
    - provide the correct output PNG path on the remote system



Parallel software rendering (cont.)

14. Upload the script to the cluster:

```
$ scp airflow.py cedar.computecanada.ca:scratch/
```

15. On the cluster try running it as a parallel interactive job:

```
$ cd ~/scratch
$ salloc --time=0:60:0 --ntasks=128 --mem-per-cpu=3600 --account=def-someuser
$ module load gcc/9.3.0 paraview-offscreen/5.10.0
$ mpirun -np 128 pvbatch --force-offscreen-rendering airflow.py
```

16. Once you are happy with the result, write a Slurm job submission script and submit it with `sbatch`

OpenGL context for off-screen rendering on a GPU

To render on a GPU from an OpenGL application such as ParaView, **traditionally you would require**:

1. OpenGL support in the GPU driver, and
2. an X server that handles windows and surfaces onto which client APIs can draw
   - run X11 server (typically started by root) on the GPU compute node, set DISPLAY=:0.$gpuindex (get GPU index from Slurm)

Latest NVIDIA GPU drivers include EGL (*Embedded-System Graphics Library*) support enabling creation of an OpenGL context for off-screen rendering without an X server.

- Your OpenGL application needs to be **recompiled with EGL support** $\Rightarrow$ use a special version of ParaView for GPU rendering without an X server; currently compiled into a module `paraview-offscreen-gpu/5.10.0` that provides both **`pvserver`** for client-server and **`pvbatch`** for batch rendering
- Unlike X11, EGL does not require any special setting to scale to very high resolutions, e.g., 4K (3840 $\times$ 2160) – simply ask it to render a 4K image

Interactive client-server rendering on a cluster's GPU

Details in `http://bit.ly/2wrSvKV`

1. On Cedar/Graham/Béluga **submit an interactive job** to the GPU partition, e.g., a serial job:

```
$ salloc --time=0:30:0 --ntasks=1 --gpus-per-node=[type:]count \
         --mem-per-cpu=3600 --account=def-someuser
```

   When the job starts, it'll return a prompt on the assigned compute node.

2. On the compute node inside the job **start the ParaView server** using a special version of ParaView with EGL support

```
$ module load gcc/9.3.0 paraview-offscreen-gpu/5.10.0
$ unset DISPLAY    # so that PV does not attempt to use X11 rendering context
$ pvserver         # --egl-device-index=0 not needed: first available GPU
                   #    is #0 inside the job
```

   For multiple GPUs can use

```
$ nvidia-smi -L    # will return 0, 1, ...
```

   The `pvserver` command will return something like

```
Waiting for client...
Connection URL: cs://cdr347.int.cedar.computecanada.ca:11111
Accepting connection(s): cdr347.int.cedar.computecanada.ca:11111
```

Interactive client-server rendering on a cluster's GPU (cont.)

3. On your desktop **set up ssh forwarding** to the ParaView server port:

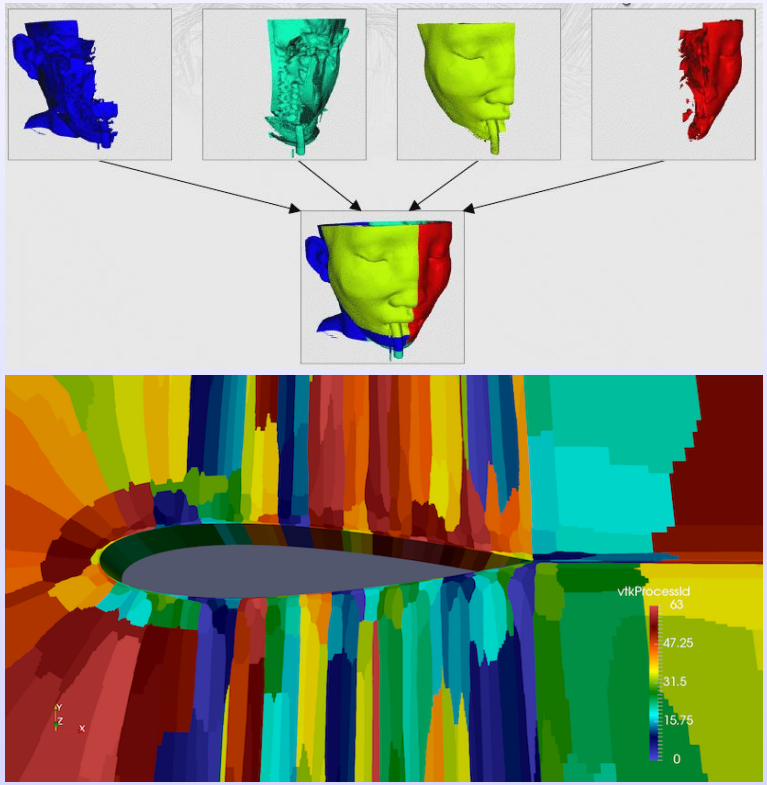
```
$ ssh username@cedar.computecanada.ca -L 11111:cdr347:11111
```

4. On your desktop **start ParaView 5.10.x** and **edit its connection properties** under *File - Connect - Add Server* (name = Cedar, server type = Client/Server, host = localhost, port = 11111), click *Configure* → *Manual* → *Save*, then select the server from the list and click on *Connect*

- ParaView's client and server must have matching major versions (5.10.x)

Data partitioning in parallel ParaView

- If loading unpartitioned data ⇒ dynamic load balancing is handled automatically for structured data:
  - structured points
  - rectilinear grid
  - structured grid
- Unpartitioned unstructured data will usually be read in serial, then must be passed through D3 (Distributed Data Decomposition) filter for dynamic load balancing:
  - particles/unstructured points
  - polygonal data
  - unstructured grid
- Some unstructured file formats can be read in parallel, e.g. the OpenFOAM reader will automatically read its unstructured data in parallel, distributing it among all available CPU cores
- After passing your unstructured data through D3, you can save it as parallel PVTU file ⇒ you'll get a statically distributed dataset that you can load next time with the same number of CPU cores
- Reading time ⇒ you can usually tell if your dataset is being read in serial or in parallel
- Look for `vtkProcessID` variable



Data partitioning in parallel ParaView (cont.)

If you have a large (many GBs) .vtu file:

1. Read your serial .vtu file into parallel ParaView on 16 cores - slow
   - and hope that it does not run out of memory on the reading core!
   - at this point the dataset is sitting in memory on one core
   - example: serial .vtu file at 9.1GB $\Rightarrow$ 1'49" reading time
2. Apply D3 filter to distribute the dataset - slowish (memory + MPI)
3. File $\rightarrow$ Save data as .pvtu with lz4 level-6 (fast) compression - fast
   $\Rightarrow$ 16 files + 1 header file
   - now you have a statically decomposed dataset
4. Restart parallel ParaView on 16 cores, read .pvtu from scratch into - fast!
   - at this point the dataset is distributed across all 16 cores
   - example: same (but now decomposed) .pvtu dataset at 5.1GB (fast compression) $\Rightarrow$ 11" reading time

- The same I/O speeds logic applies to .vti $\rightarrow$ .pvti (but there is no need for D3)

Exercise: parallel rendering of partitioned data

This is an extremely concise step-by-step guide for the turbine dataset:

1. Submit an interactive job
   `salloc --time=0:60:0 --ntasks=16 --mem-per-cpu=3600`
2. Start client-server ParaView session on 16 cores
3. Load all .vtm files (all 10 timesteps)
4. Apply Merge Blocks, output type = Unstructured Grid
5. Apply Cell Data to Point Data (so that you could use Contour)
6. Apply D3
7. Save data as `decomposed.pvtu`, write all timesteps as series, fast compression
8. Restart client-server ParaView session on 16 cores
9. Load all `decomposed.pvtu` files
10. Create visualization interactively
11. Save animation as 1000 × 800 PNG files ✍ this step should take ∼ 1min of processing time
12. Merge them into a movie with `ffmpeg`

Remote rendering summary: some orthogonal decisions

(1) interactive vs. batch

- interactive client-server for a quick look, exploration or debugging
  - another option is to download a scaled-down version of your dataset, debug a script locally on your laptop, and then run it as a batch job on the original full-resolution dataset on the cluster
- batch really preferred for production jobs and producing animations

(2) CPU vs. GPU

- in general, no single answer which one is better
  - you can throw many CPUs at your rendering job
  - modern software rendering libraries such as OSPRay (Intel's ray tracing) and OpenSWR (Intel's rasterizer) can be very fast, depending on your visualization
- might have to resort to software rendering if no GPUs are available (e.g., all are taken by GP-GPU jobs)
- for initial exploration, I would use the dataset size (GBs) to figure out the best number of CPU cores, and adjust from there

Summary

Further resources

- ParaView Discourse
  https://discourse.paraview.org
- Self-directed ParaView tutorial
  https://docs.paraview.org/en/latest/Tutorials/SelfDirectedTutorial/index.html
- ParaView User's Guide
  https://docs.paraview.org/en/latest/UsersGuide/index.html
- ParaView F.A.Q.
  http://www.itk.org/Wiki/ParaView:FAQ
- VTK wiki with webinars, tutorials, etc.
  http://www.vtk.org/Wiki/VTK
- VTK for C++/Python/Java/C#/JavaScript code examples
  https://kitware.github.io/vtk-examples
- VTK file formats (3rd-party intro)
  http://www.earthmodels.org/software/vtk-and-paraview/vtk-file-formats

Our visualization webinars

- ∼3-4 visualization webinars per academic year
  - ✐ keep an eye on our emails, Twitter, `https://westgrid.github.io/trainingMaterials/blog`
  - ▶ ∼50 mins + questions, usually on fairly specific or advanced topics
- Many past webinars are available with slides and screencasts at `https://bit.ly/vispages`
  - "In-situ visualization with ParaView Catalyst2"
  - "Highlights from the 2021 IEEE SciVis Contest"
  - "Remote visualization on Compute Canada clusters"
  - "Scientific visualization on NVIDIA GPUs"
  - "Workflows with Programmable Filter / Source in ParaView"
  - "The Topology ToolKit (TTK)"
  - "Web-based 3D scientific visualization" (ParaViewWeb, vtk.js, ParaView Glance)
  - "Photorealistic rendering with ParaView and OSPRay"
  - "Batch visualization on Compute Canada clusters"
  - "Molecular visualization with VMD" • "Intermediate VMD topics: trajectories, movies, scripting"
  - "Using YT for analysis and visualization of volumetric data" (part 1) • "Working with data objects in YT" (part 2)
  - "Scientific visualization with Plotly"
  - "Novel visualization techniques from 2017 VISUALIZE THIS competition"
  - "Camera animation in ParaView and VisIt"
  - "3D visualization on new Compute Canada systems"
  - "Using ParaViewWeb for 3D visualization and data analysis in a web browser"
  - "Visualization support in WestGrid / Compute Canada"
  - "Scripting and other advanced topics in VisIt visualization"
  - "CPU-based rendering with OSPRay"
  - "3D graphs with NetworkX, VTK, and ParaView" • "Graph visualization with Gephi"
- We are always looking for topic suggestions!

Documentation and getting help

- Visualization in the Alliance `https://ccvis.netlify.app` (online gallery)
- Official documentation `https://docs.alliancecan.ca/wiki/Visualization`
- Western Canada research computing visualization resources `https://bit.ly/vispages`
- Email **support@tech.alliancecan.ca** and mention *"visualization"* in the subject line (goes to our ticketing system)
- Email me **alex.razoumov@westdri.ca**
- ParaView documentation
  - official documentation `https://docs.paraview.org/en/latest`
  - wiki `http://www.paraview.org/Wiki/ParaView`
  - Python batch scripting `http://bit.ly/2wF5v0B`
  - VTK tutorials `http://www.itk.org/Wiki/VTK/Tutorials`