

# Data Visualization on Compute Canada's Systems

Cedar and Graham clusters

Alex Razoumov  
alex.razoumov@westgrid.ca

WestGrid / Compute Canada

copy of these slides and other files at <http://bit.ly/remotecedar>  
➡ will download `remote.zip`

# Outline

- **Interactive client-server visualization on Cedar and Graham**
  - ▶ rendering on a cluster's GPU
  - ▶ software rendering on a single CPU
  - ▶ parallel software rendering on multiple CPUs
- **State of VNC (remote desktop)**
- **Running off-screen visualization scripts on Cedar and Graham**
  - ▶ batch jobs on CPU nodes
    - serial interactive jobs – purely for debugging or testing your scripts
    - serial (non-interactive) batch jobs
    - parallel (non-interactive) batch jobs
  - ▶ batch jobs on GPU nodes
- **Visualization on virtual machines on Arbutus** (part of CC cloud)

# Why remote visualization?

- Dataset could be **too big to download**
- Dataset and its analysis workflow **cannot fit into desktop's memory**
- Desktop rendering is **too slow** (limited CPU/GPU power)
- **In-situ visualization** = instrumenting a simulation code on the cluster to output graphics and/or connect to a visualization frontend (ParaView, VisIt) on the fly
- Required visualization software is **licensed** on Compute Canada systems (only for commercial packages)

# Dataset

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

$$f(x_1, x_2, x_3) = \sum_{i=1}^2 \left[ \frac{\sin^2 \left( \sqrt{\xi_{i+1}^2 + \xi_i^2} \right) - 0.5}{[0.001(\xi_{i+1}^2 + \xi_i^2) + 1]^2} + 0.5 \right], \text{ where } \xi_i \equiv 30(x_i - 0.5)$$

discretized on a  $100^3$  Cartesian grid

- You’ll find the code `buildSine.c` inside `remote.zip` (see the first slide)

```
$ export LD_RUN_PATH=/path/to/netcdf/lib
$ gcc buildSine.c -o buildSine -I/path/to/netcdf/include \
    -L/path/to/netcdf/lib -lnetcdf
$ ./buildSine
```

- This will produce the file `sineEnvelope.nc` which ParaView can read (also included in `remote.zip`)

# Cedar and Graham clusters

- General-purpose clusters for a variety of workloads
- Entered production in June 2017
- Respectively:
  - ▶ located at SFU and UofWaterloo
  - ▶ 27,696 and 32,136 CPUs
  - ▶ 584 and 320 NVIDIA P100 Pascal GPUs (12GB/16GB on-board memory)
  - ▶ specs at <https://docs.computecanada.ca/wiki/Cedar> and <https://docs.computecanada.ca/wiki/Graham>
- Multiple types of nodes, with 128GB/256GB/0.5TB/1.5TB/3TB memory
- Batch-oriented environment for parallel and serial jobs, use Slurm scheduler and workload manager
- Identical software setup  
[https://docs.computecanada.ca/wiki/Available\\_software](https://docs.computecanada.ca/wiki/Available_software)

# Perhaps you don't need 3D?

Off-screen Matplotlib example on Cedar/Graham

- In Python's matplotlib can script the entire workflow without opening windows – use a non-interactive backend
- More details at [http://matplotlib.org/faq/usage\\_faq.html#what-is-a-backend](http://matplotlib.org/faq/usage_faq.html#what-is-a-backend)
- See <http://matplotlib.org/gallery.html> for plotting examples covering many 1D/2D use cases

```
import matplotlib as mpl
mpl.use('Agg')      # for PNG; could also use PS or PDF backends
import matplotlib.pyplot as plt
from numpy import *
x = linspace(0,3)
y = 10.*exp(-2.*x)
plt.figure(figsize=(10,8))
plt.plot(x, y, 'ro-')
plt.savefig('tmp.png')
```

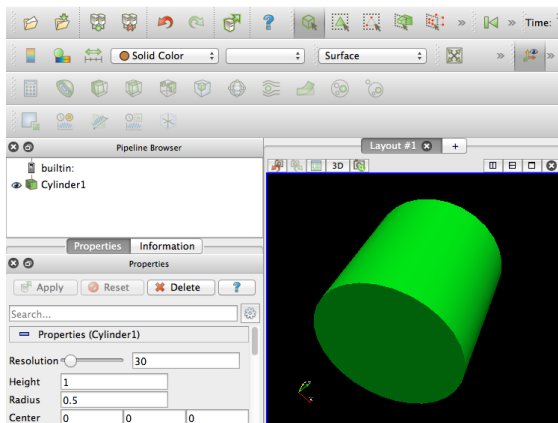
```
$ module load python/2.7.13
$ pip install --user matplotlib    # will install into ~/.local/
$ python simple.py                 # should produce tmp.png
```

# 3D general-purpose vis: VisIT and ParaView

- Open source, under active development, multi-platform (Linux/Mac/Windows)
  - Scalar, vector, tensor fields
  - Wide variety of discretizations (structured, unstructured, particles, irregular in 2D/3D)
  - Support extremely large datasets (GBs to TBs)
  - Scale to large ( $10^3 - 10^5$  cores) systems via MPI
  - Nice interactive GUI and Python scripting
  - Client-server mode
  - Support over 100 input data formats
  - Support parallel I/O
  - Huge array of visualization features
  - VTK support, OpenGL graphics
  - We teach both (full-day, in-person workshops)
- In this webinar all examples will use ParaView
- Same general ideas apply to remote VisIt visualization (*client-server and batch rendering*)  $\Rightarrow$  let me know if you want more details for VisIt

# I assume you are already familiar with ParaView basics

- Main elements of the GUI
- How to load a dataset (and convert your data into a Paraview-readable format)
- How to switch between different views (representations)
- How to colour visualization by a variable
- How to use filters to build a pipeline



If not, please see our regular ParaView workshop materials at <http://bit.ly/paraviewzip> (slides/data/codes) and the official docs <http://www.paraview.org/documentation>



# Few words on X11 forwarding

From cluster's login or compute node to your laptop

- Could connect with X11 forwarding (`ssh -Y`) and run the script on the login (GPU-less) node

```
$ ssh -Y cedar.computecanada.ca # or graham.computecanada.ca
$ module load paraview/5.3.0
$ paraview
```

or even on a compute node inside an interactive job

```
$ ssh -Y cedar.computecanada.ca # or graham.computecanada.ca
$ module load paraview/5.3.0
$ salloc --x11 --time=0:30:0 --ntasks=1 ...
$ paraview
```

- Not a good idea for a number of reasons!
  - ▶ login nodes are shared by many users
  - ▶ X11 connection requires **lots of round trips and is not very efficient** (designed in mid-1980s!)  $\Rightarrow$  **slow response** and unnecessary network traffic
  - ▶ requires an X11 server on your laptop
  - ▶ instead we suggest using client-server or batch scripts (more on VNC later)

# OpenGL context for off-screen rendering on a GPU

To render on a GPU from an OpenGL application such as ParaView, you need:

- (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 (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 to render graphics on a GPU without an X server; currently compiled into a module `paraview-offscreen-gpu/5.4.0` 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

# ParaView's distributed parallel architecture

Three logical components inside ParaView – these units can be embedded in the same application on the same computer, but can also run on different machines:

- **Data Server** – The unit responsible for data reading, filtering, and writing. All of the pipeline objects seen in the pipeline browser are contained in the data server. The data server can be parallel.
- **Render Server** – The unit responsible for rendering. The render server can also be parallel, in which case built-in parallel rendering is also enabled.
- **Client** – The unit responsible for establishing visualization. The client controls the object creation, execution, and destruction in the servers, but does not contain any of the data, allowing the servers to scale without bottlenecking on the client. If there is a GUI, that is also in the client. The client is always a serial application.

# Interactive jobs on Cedar/Graham

- Client-server workflow is by definition interactive
- Interactive jobs should automatically go to one of Slurm interactive partitions (CPU or GPU)

```
$ sinfo | grep 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 ... --account=def-user-role  
$ echo $SLURM_...    # access Slurm variables, or set your environment  
$ ./serial  
$ srun ./mpi         # run an MPI code  
$ exit               # terminate the job (go back to the login node)
```

# Interactive client-server rendering on a cluster's GPU

Details in <http://bit.ly/2wrSvKV>

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

```
$ salloc --time=0:30:0 --ntasks=1 --gres=gpu:1 \  
--mem-per-cpu=4000 --account=def-razoumov-ac
```

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 paraview-offscreen-gpu/5.4.0  
$ unset DISPLAY      # so that PV does not attempt to use X11 rendering context  
$ pvserver           # --egl-device-index=0 not needed: first 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

... continued

- (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 5.4.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*

# Interactive client-server rendering on a cluster's GPU

... continued

- ParaView's client and server must have matching major versions (5.4.x)
- Occasionally during client-server connection might get an error *"Only EGL 1.4 and greater allows OpenGL as client API"*
  - ▶ the GPU is stuck in a strange state  $\Rightarrow$  need to reboot the node (let us know!)
- In ParaView's preferences can set Render View  $\rightarrow$  Remote/Parallel Rendering Options  $\rightarrow$  Remote Render Threshold (beyond which rendering will be remote)
  - ▶ **default 20MB**  $\Rightarrow$  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**  $\Rightarrow$  all rendering (including rotation) will be remote, so you will be really using the cluster's GPU for everything
    - good for large data processing
    - not so good for interactivity
  - ▶ experiment with the threshold to find a suitable value

# Serial client-server software rendering

Details in <http://bit.ly/2x5zIB6>

- (1) On Cedar or Graham **submit a serial interactive job**:

```
$ salloc --time=0:30:0 --ntasks=1 --account=def-razoumov-ac
```

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

- (2) On the compute node inside the job **start a ParaView server**

```
$ module load paraview-offscreen/5.3.0
$ pvserver --use-offscreen-rendering --mesa-swr-avx2
```

The pvserver command will return something like

```
Waiting for client...
Connection URL: cs://cdr544.int.cedar.computecanada.ca:11111
Accepting connection(s): cdr544.int.cedar.computecanada.ca:11111
```

- (3) On your desktop **set up ssh forwarding** to the ParaView server port:

```
$ ssh username@cedar.computecanada.ca -L 11111:cdr544:11111
```

- (4) On your desktop **start ParaView 5.3.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*



# Parallel client-server software rendering

Details in <http://bit.ly/2x5zIB6>

- (1) On Cedar or Graham **submit a parallel interactive job**:

```
$ salloc --time=0:30:0 --ntasks=4 --mem-per-cpu=2000 --account=def-razoumov-ac
```

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

- (2) On the compute node inside the job **start a parallel ParaView server**

```
$ module load paraview-offscreen/5.3.0  
$ srun pvserver --use-offscreen-rendering
```

The pvserver command will return something like

```
Waiting for client...  
Connection URL: cs://cdr544.int.cedar.computecanada.ca:11111  
Accepting connection(s): cdr544.int.cedar.computecanada.ca:11111
```

- (3) On your desktop **set up ssh forwarding** to the ParaView server port:

```
$ ssh username@cedar.computecanada.ca -L 11111:cdr544:11111
```

- (4) On your desktop **start ParaView 5.3.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*

# Parallel client-server software rendering

... continued

- Works faster on 4 cores!
- Switch to surface view, pass your dataset through Filters - ProcessIdScalars, colour by ProcessId to check that rendering is happening in parallel

# VNC: no solution in place yet

- **Cedar:** at first one of standard login nodes (no GPUs) will be dedicated to software rendering
  - ▶ secure VNC setup with user authentication, not under the scheduler
  - ▶ more such nodes might be added later
  - ▶ next few months
- **Graham:** several GPU login nodes will be added to the system, to be used for visualization
  - ▶ secure VNC setup with user authentication, not under the scheduler
  - ▶ multiple users sharing individual physical GPUs
  - ▶ no ETA yet
- Wide variety of VNC-like setups on legacy systems across the country
  - ▶ on legacy WestGrid clusters users start a VNC server manually inside a submitted job, set up VNC port forwarding, connect a client
  - ▶ same can be done right now on Cedar and Graham, but the setup is a little bit cumbersome
  - ▶ no user-space VNC servers are allowed on login nodes

# Off-screen batch rendering with scripts

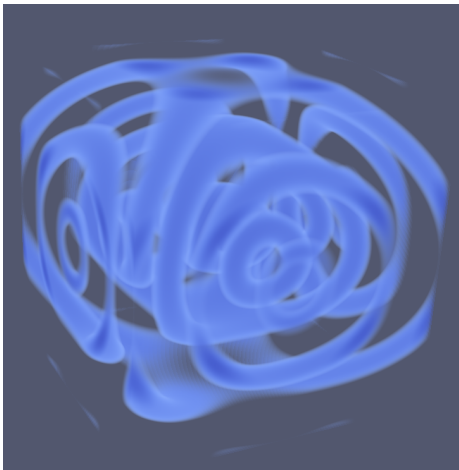
- Very large renderings cannot be done interactively (they take too long!)
  - ⇒ submit a batch rendering job and come back to a nice visualization in a few hours or the next day
    - ▶ can be performed on any combination of GPUs or CPUs, but details vary
- Automate mundane or repetitive tasks, e.g., making multiple frames of a movie

Workflow in any Linux-compatible visualization tool with a programming interface (in a compiled or interpreted language) can be scripted on a cluster

# Open-source 3D vis. tools with scripting interfaces

- Many domain-specific packages support scripting, e.g., VMD (Visual Molecular Dynamics) provides Python and Tcl interfaces
  - ▶ to get batch visualization help with any domain-specific tool, please contact us
- General-purpose tools
  - ▶ VTK (Visualization Toolkit) library has C++, Tcl/Tk, Java and Python interfaces – can be used as a standalone renderer
  - ▶ Mayavi2, a serial 3D interactive scientific data visualization package, has an embedded Python shell
  - ▶ both **VisIT** and **ParaView** provide Python scripting (and compiled language interfaces for in-situ visualization)

# Writing ParaView Python scripts

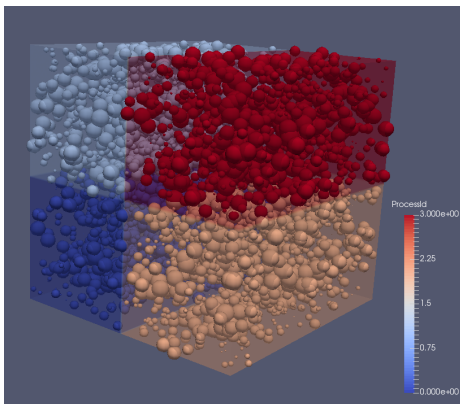


- (1) Use Tools → Start/Stop Trace to produce a volume rendering script
- (2) Debug the script either in client-server or standalone on your laptop
- (3) Before multi-hour batch renderings, make sure the script works with cluster's ParaView:
  - ▶ file paths
  - ▶ cluster's ParaView version (its Python library is evolving quickly!)
  - ▶ support for third-party libraries, formats, etc.

# Running ParaView scripts

- I put an optimized version of the script `static.py` inside `remote.zip`
- Many ways to run this script:
  - ▶ open ParaView's built-in Python interpreter Tools → Python Shell, load the script with Run Script
  - ▶ `paraview --script=static.py`
  - ▶ `pvpython` will give you a Python shell (connected to a ParaView server without the GUI) into which you can copy and paste the script
  - ▶ `pvbatch --use-offscreen-rendering static.py`

# Another quick script to test parallel rendering



The script `spheres.py`

- 1 uses `ProcessIdScalars` filter to assign a unique scalar (0, 1, 2, ...) to each subdomain of the dataset according to which processor it resides on,
- 2 colours the corresponding subdomain with a unique colour and renders it as a semi-transparent volume,
- 3 drops 5000 random spheres and scales them by the value of the function at that location,
- 4 colours these spheres by the colour of its host subdomain



# Software batch rendering

Off-screen batch visualization on CPU nodes

## (1) On Cedar or Graham prepare our environment

```
$ module load paraview-offscreen/5.3.0  
$ cd ~/remote && /bin/rm *.png
```

## (2) Start with an interactive job

```
$ salloc --time=0:30:0 --ntasks=1 --mem-per-cpu=2000 --account=def-razoumov-ac  
$ pvbatch --use-offscreen-rendering static.py # should produce volume.png  
$ exit # terminate the interactive job
```

- It would read data from disk, do rendering, and write the resulting image to disk – all without opening any windows

# Software batch rendering

... continued

## (3) Next run it as a serial batch (non-interactive) job

```
#!/bin/bash
#SBATCH --time=00:05:00    # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --job-name="quick test"
#SBATCH --mem=2000         # in MB
#SBATCH --account=def-razoumov-ac
pvbatch --use-offscreen-rendering static.py
```

```
$ /bin/rm *.png
$ sbatch sl.sh    # should produce volume.png
$ squeue -u razoumov
```

# Software batch rendering

... continued

## (4) Now run it as a parallel batch job

```
#!/bin/bash
#SBATCH --ntasks=4    # number of MPI processes
#SBATCH --time=0-00:05 # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --mem-per-cpu=2000 # in MB
#SBATCH --account=def-razoumov-ac
srun pvbatch --use-offscreen-rendering static.py

$ /bin/rm *.png
$ sbatch p1.sh # runs static.py on 4 cores, should produce volume.png
$ squeue -u razoumov
$ sbatch p2.sh # runs spheres.py on 4 cores, should produce regions.png
$ sbatch p3.sh # runs spheres.py on 8 cores, should produce regions.png
```

# GPU batch rendering on Cedar/Graham

Let's run a serial GPU job

```
#!/bin/bash
#SBATCH --gres=gpu:1          # GPUs per node
#SBATCH --mem=2000M          # memory per node
#SBATCH --time=0-05:00       # walltime in d-hh:mm or hh:mm:ss format
#SBATCH --account=def-razoumov-ac
unset DISPLAY
pvbatch static.py
```

```
$ /bin/rm *.png
$ module load paraview-offscreen-gpu/5.4.0
$ sbatch gpu.sh # should produce volume.png
$ squeue -u razoumov
```

# Extending the script

Typing commands inside ParaView's Python shell with an active view

```
>>> help(GetActiveCamera)
```

Help on function GetActiveCamera in module paraview.simple:

```
GetActiveCamera()
```

Returns the active camera **for** the active view. The returned **object** is an instance of vtkCamera.

```
>>> dir(GetActiveCamera()) # list all the fields and methods of the object
['AddObserver', 'ApplyTransform', 'Azimuth', 'BreakOnError', 'ComputeViewPlaneNormal', 'DebugOff', 'DebugOn', 'DeepCopy', 'Dolly', 'Elevation', 'FastDelete', 'GetAddressAsString', 'GetCameraLightTransformMatrix', 'GetClassName', 'GetClippingRange', 'GetCommand', 'GetCompositeProjectionTransformMatrix', 'GetDebug', 'GetDirectionOfProjection', 'GetDistance', 'GetEyeAngle', 'GetEyePlaneNormal', 'GetEyePosition', 'GetEyeSeparation', 'GetEyeTransformMatrix', 'GetFocalDisk', 'GetFocalPoint', 'GetFreezeFocalPoint', 'GetFrustumPlanes', 'GetGlobalWarningDisplay', 'GetLeftEye', 'GetMTime', 'GetModelTransformMatrix', 'GetModelViewTransformMatrix', 'GetModelViewTransformObject', 'GetOrientation', 'GetOrientationWXYZ', 'GetParallelProjection', 'GetParallelScale', 'GetPosition', 'GetProjectionTransformMatrix', 'GetProjectionTransformObject', 'GetReferenceCount', 'GetRoll', 'GetScreenBottomLeft', 'GetScreenBottomRight', 'GetScreenTopRight', 'GetThickness', 'GetUseHorizontalViewAngle', 'GetUseOffAxisProjection', 'GetUserTransform', 'GetUserViewTransform', 'GetViewAngle', ...]
```

# Extending the script (cont.)

```
>>> help(GetActiveCamera().Azimuth)  
Help on built-in function Azimuth:
```

```
Azimuth(...)  
  V.Azimuth(float)  
  C++: void Azimuth(double angle)
```

Rotate the camera about the view up vector centered at the focal point. Note that the view up vector **is** whatever was **set** via **SetViewUp**, **and is not** necessarily perpendicular to the direction of projection. The result **is** a horizontal rotation of the camera.

# Extending the script (cont.)

- Inside `static.py` let's replace the line

```
SaveScreenshot('/home/razoumov/remote/volume.png', renderView1)
```

with the following:

```
camera = GetActiveCamera()
numberOfFrames = 90
for i in range(numberOfFrames):
    camera.Azimuth(1)      # rotate by 1 degree
    SaveScreenshot('/home/razoumov/remote/frame%04d'%(i)+'.png',
                  view=renderView1)
```

and save the script as `spin.py`

- Run it as a single-processor batch job

```
#!/bin/bash
#SBATCH --time=00:15:00  # give it a little bit more time
#SBATCH --job-name="quick test"
#SBATCH --mem=2000      # in MB
#SBATCH --account=def-razoumov-ac
pvbatch --use-offscreen-rendering spin.py
```

```
$ sbatch s2.sh  # should produce frame{0000..0089}.png
```

# Creating a movie

- This produces 90 files `frame0000.png`, ..., `frame0089.png` each rotated by one degree compared to the previous one
- Can merge them into a movie with a third-party tool, e.g., `ffmpeg` from `nixpkgs/16.09` module:

```
$ which ffmpeg
/cvmfs/soft.computecanada.ca/nix/var/nix/profiles/16.09/bin/ffmpeg
$ 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
...
$ ls -l spin.mp4
-rw-rw-r-- 1 razoumov razoumov 220K Oct  1 16:43 spin.mp4
```

Very smooth three-second  $800 \times 800$  movie!



# Multiple files and garbage collection

- Very often a researcher deals with a sequence of files, e.g., with outputs from a time-dependent simulation at specific regular intervals
- For many file formats, ParaView has **built-in ability to recognize a sequence of similar files (or multiple variables in a single file) as a time sequence** and animate them without any special effort, producing a movie (\*.avi, \*.ogv, \*.png sequence)
- When this does not work, it can be useful to write the script for a single frame and then enclose it into a loop by hand
  - 1 read a data file #i
  - 2 produce visualization
  - 3 output an image #i
- It is important to delete all memory-intensive objects at the end of each loop

# Visualization on VMs in Compute Canada Cloud

Details in <http://bit.ly/2xaHWI0>

Arbutus system at the University of Victoria

- OpenStack cloud providing virtual machines (VMs)
- users have root access inside a VM, can install their own software stack
- now at 7,640 CPU cores / 500TB persistent storage / 76TB RAM
- in production since September 2016

Prerequisites for visualization in a VM:

- ➡ your own cloud VM  
<https://docs.computecanada.ca/wiki/CC-Cloud>
- ➡ system dependencies for compiling ParaView or VisIt
- ➡ a copy of ParaView or VisIt compiled with Python, Mesa (open-source OpenGL implementation supporting software rendering), support for your input file format – need to compile your own!

You can find all compilation and client-server usage instructions for both ParaView and VisIt in a cloud VM in <http://bit.ly/2xaHWI0>

## Summary: these orthogonal decisions are yours to make

### (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)
- no GPUs in the cloud system

# Questions?

- Webstream viewers: email [info@westgrid.ca](mailto:info@westgrid.ca)
- Vidyo viewers: unmute & ask question or use Vidyo Chat (chat bubble icon in Vidyo menu)



- Email me anytime [alex.razoumov@westgrid.ca](mailto:alex.razoumov@westgrid.ca)
- Support email [support@computecanada.ca](mailto:support@computecanada.ca)