Introduction to Batch Visualization

Alex Razoumov
alex.razoumov@westgrid.ca

WestGrid / Compute Canada

copy of these slides at `http://bit.ly/batchslides`

sample (longer) codes at `http://bit.ly/batchcode`
- will download batch.zip

Batch visualization

BATCH PROCESSING AND VISUALIZATION OF DATA WITH SCRIPTS ON REMOTE CLUSTERS

- Will be given from a WestGrid user perspective, showing examples on parallel.westgrid.ca GPU cluster
- The same ideas (in a simpler form) apply to batch visualization on your desktop
- With some work, the examples can be ported to any other cluster in Compute Canada, with or without GPUs
- If you need help with visualization on other systems, please ask us
  - local consortia help, e.g. *support@westgrid.ca* or *help@sharcnet.ca*
  - Canada-wide visualization support list *vis-support@computecanada.ca*

Batch visualization

BATCH PROCESSING AND VISUALIZATION OF DATA WITH SCRIPTS ON REMOTE CLUSTERS

- Will be given from a WestGrid user perspective, showing examples on parallel.westgrid.ca GPU cluster
- The same ideas (in a simpler form) apply to batch visualization on your desktop
- With some work, the examples can be ported to any other cluster in Compute Canada, with or without GPUs
- If you need help with visualization on other systems, please ask us
  - local consortia help, e.g. *support@westgrid.ca* or *help@sharcnet.ca*
  - Canada-wide visualization support list *vis-support@computecanada.ca*

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

Usual benefits of scripting

- Automate mundane or repetitive tasks, e.g., making mutiple frames for a movie
- Do visualization on clusters without any GUI elements, e.g., via a job scheduled from the command line

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 support for 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)

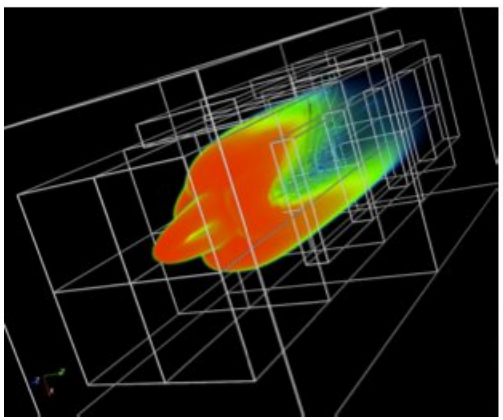
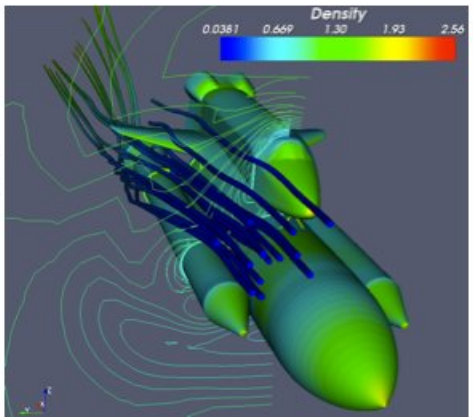
VisIT and ParaView

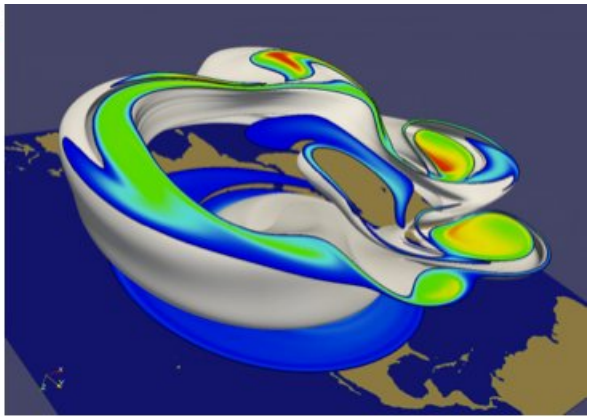
- Open source, under active development
- Scalar, vector, tensor fields
- Wide variety of discretizations (structured, unstructured, particles, irregular in 2D/3D)
- Support very large datasets (GBs to TBs)
- Scale to large ($10^3 - 10^5$ cores) systems via MPI
- Nice interactive GUI
- Client-server mode
- Support over 100 input data formats
- Support parallel I/O
- Huge array of visualization features
- VTK support

This webinar is an extension of our ParaView training

ParaView `http://www.paraview.org`

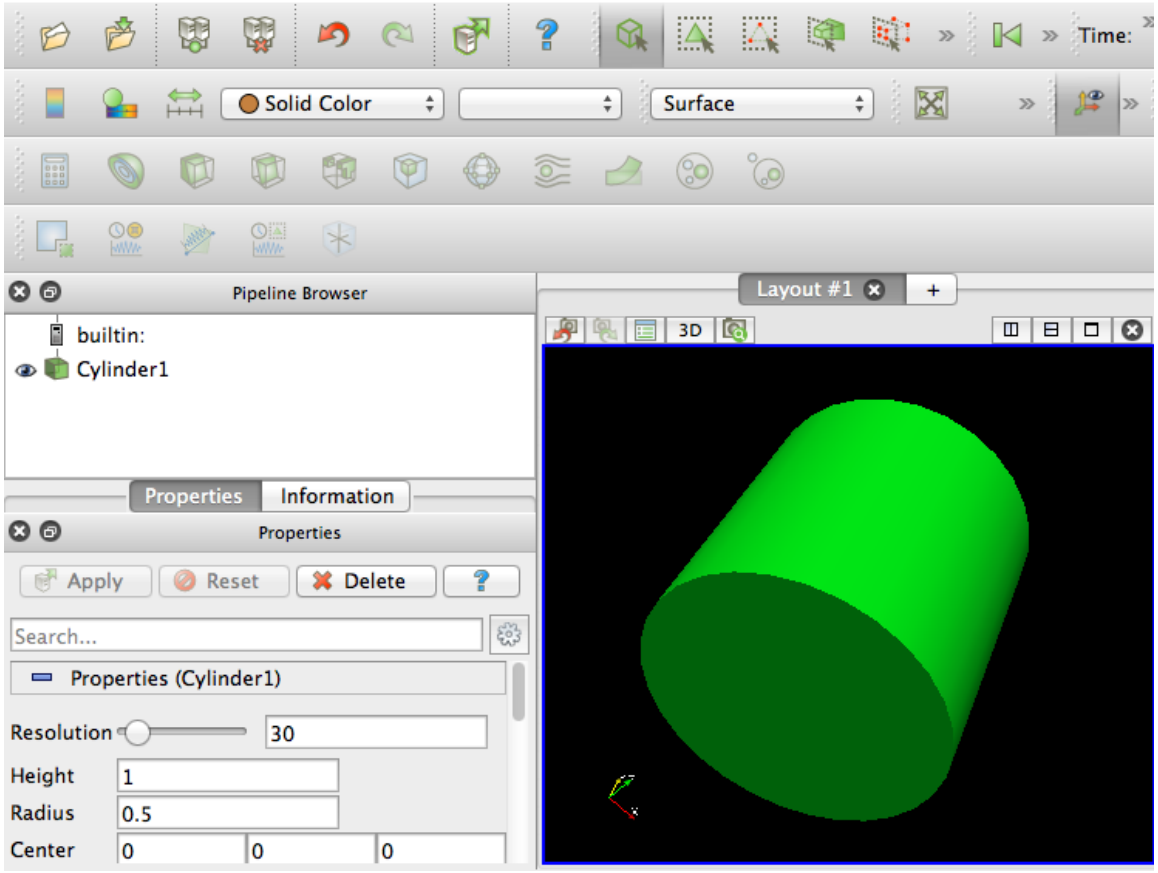
- Started in 2000 as a collaboration between Los Alamos National Lab and Kitware Inc., in 2005 joined by Sandia National Labs
- Latest stable release 4.3.1, binary downloads for Linux/Mac/Windows
- Built to visualize extremely large datasets on distributed memory machines, works equally well on a desktop
- Interactive GUI and Python scripting
- Uses MPI for distributed-memory parallelism on HPC clusters
- ParaView is based on VTK (developed by the same folks)



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, see our regular ParaView workshop material at
`http://bit.ly/pvslides` (slides)
and `http://bit.ly/pvarchive` (datasets and codes),
as well as `http://www.paraview.org/documentation`

Perhaps you don't need 3D?

- 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`
- 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')
```

ParaView's distributed parallel architecture

Three logical units of 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.

Remote visualization

If your dataset is on cluster.consortium.ca ⇒ you have several options:

1. download data to your desktop and visualize it locally
2. run ParaView remotely on the cluster via X11 forwarding
   your desktop $\xrightarrow{\text{ssh } -X}$ larger machine running ParaView
3. run ParaView remotely on the cluster **via VNC**
   your desktop $\xrightarrow{\text{VNC}}$ larger machine running ParaView
4. run ParaView in **client-server mode**
   ParaView client on your desktop $\rightleftharpoons$ ParaView server on the cluster
5. run ParaView via a **GUI-less batch script** (interactively or scheduled)

For remote options (2) - (5) the setup details vary across the consortia

- render server can run with or without GPU rendering
- data/render servers can run on single-core, or across several cores/nodes with MPI
- for interactive GUI work on clusters it's best to schedule interactive jobs, as opposed to running on the login nodes

Example 1

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

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

discretized on a 100^3 Cartesian grid

- You'll find the code `example1.c` inside batch.zip (see the first slide)

```
$ export LD_RUN_PATH=/path/to/netcdf/lib
$ gcc example1.c -o example1 -I/path/to/netcdf/include -L/path/to/netcdf/lib \
    -lnetcdf
$ ./example1
```

- This will produce the file `sineEnvelope.nc` which ParaView can read

Demo time

1. Compile and run `example1.c` – this will produce `sineEnvelope.nc`
2. Load `sineEnvelope.nc` into ParaView
3. Play with several views: contours, volume ray tracing
4. Use Tools → Start/Stop Trace to produce the script for volume rendering

Running the script

- I put the optimized version of the script `spin.py` inside batch.zip
- There are many ways to run this script
  - Open ParaView's built-in Python interpreter Tools → Python Shell, load the script with Run Script
  - *[/usr/bin/ /usr/local/bin/ /Applications/paraview.app/Contents/bin/]* **pvpython** will give you a Python shell connected to a ParaView server without the GUI – can copy and paste commands here
  - *[/usr/bin/ /usr/local/bin/ /Applications/paraview.app/Contents/bin/]* **pvbatch** *spin.py* will run the script

Extending the script

Typing commands inside ParaView's Python shell after running the script

```
>>> 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.)

Let's replace the current line

```
SaveScreenshot('/Users/razoumov/Documents/03-webinar/test.png', magnification=2,
    quality=100, view=renderView1)
```

inside `spin.py` with the following:

```
camera = GetActiveCamera()
firstFrame = 90
lastFrame = 180
camera.Azimuth(firstFrame)    # rotate by 90 degrees
for i in range(lastFrame-firstFrame+1):
    camera.Azimuth(1)     # rotate by 1 degree
    SaveScreenshot('/Users/razoumov/Documents/03-webinar/frame%04d'%(i)+'.png',
        magnification=2, quality=100, view=renderView1)
```

and run the script again

```
/path/to/pvbatch spin.py
```

Creating a movie

- This will produce 91 files frame0000.png, ..., frame0090.png each rotated by a one degree compared to the previous one
- Can merge them into a movie with a third-party tool, e.g.

```
$ ffmpeg -r 10 -i frame%04d.png -c:v libx264 -pix_fmt yuv420p -vf "scale=trunc(iw
    /2)*2:trunc(ih/2)*2" cube.mp4
$ ls -lh cube.mp4
-rw-r--r--@ 1 razoumov  staff   421K 17 Mar 22:56 cube.mp4
```

Remote script execution on the login node

- Parallel.westgrid.ca is a cluster at UofCalgary with 60 twelve-core GPU nodes with 24GB memory and 3 NVIDIA Tesla M2070s GPUs each, used for GP-GPU computing and visualization – four of these nodes are open to the interactive queue (accepting jobs less than 3 hours)
- Could connect with X11 forwarding (ssh -X) and run the script on the login (GPU-less) node

```
/global/software/ParaView/ParaView-4.3.1-Linux-64bit/bin/pvbatch spin.py
```

- Not a good idea for a number of reasons!
  - login nodes are shared by many users, usually have restrictions
  - default ParaView on Linux requires an X11 server (your laptop) ⇒ unnecessary network traffic
  - `pvbatch --use-offscreen-rendering` requires recompiling ParaView against off-screen rendering OSMesa library (without recompiling --use-offscreen-rendering just won't open a window, still requires X11)
- On parallel.westgrid.ca we recommend using the GPU nodes for batch visualization

Batch script GPU visualization on WestGrid

Method 1: via a scheduled interactive job on parallel.westgrid.ca

```
qsub -q interactive -I -l nodes=1:ppn=1:gpus=1,walltime=0:30:00
          # wait for an interactive shell
firstgpu=$( head -n 1 "$PBS_GPUFILE" )
gpuindex=${firstgpu:(-1)}
export DISPLAY=:0.$gpuindex
/global/software/ParaView/ParaView-4.3.1-Linux-64bit/bin/pvbatch spin.py
```

- We'll use the X11 display associated with the compute node's GPU
- It would read data from disk, do rendering (inside a window on the cluster's GPU), and write an image (or many) to disk
- In my experience, it is easiest to debug a ParaView Python script on your laptop, but sometimes you might need to use the trace tool or debug in place on the remote system

Remote debugging via VNC

full details at `http://bit.ly/remotevnc`

1. Install TurboVNC from `http://www.turbovnc.org` on your desktop
2. Log in to parallel.westgrid.ca and optionally run the command `vncpasswd`, at the prompt set a password for your VNC server (you'll use it below)
3. **Submit an interactive job** to the cluster
   `qsub -q interactive -I -l nodes=1:ppn=1:gpus=1,walltime=1:00:00`
   When the job starts, it'll return a prompt on the assigned compute node.
4. On the compute node **start the vncserver**: `vncserver -geometry 1024x768`
   It'll produce *"New 'X' desktop is cn0553:1"*, where the syntax is `nodeName:displayNumber`
5. On your desktop **set up ssh forwarding** to the VNC port:
   `ssh username@parallel.westgrid.ca -L 5901:cn0553:5901`
   Here the port number is 5901 = 5900 (VNC's default) + displayNumber
6. **Start TurboVNC vncviewer** on your desktop, enter `localhost:displayNumber`, e.g. `localhost:1`, and then enter the password from item 2 above
7. A remote Gnome desktop will appear inside a VNC window on your desktop
8. Inside this desktop start a terminal, use it to **start ParaView with a VirtualGL wrapper**
   `vglrun /global/software/ParaView/ParaView-4.3.1-Linux-64bit/bin/paraview`
9. Work on your Python script

Batch script GPU visualization on WestGrid (cont.)

Method 2: via a scheduled batch job on parallel.westgrid.ca, using the same Python script

```
qsub -q interactive ./visualization.sh    # -q needed to send to an interactive
                                          # node (GPUs + max 3h runtime);
                                          # can also use -q gpu (might wait longer)
```

visualization.sh script

```
#!/bin/bash
#PBS -S /bin/bash
#PBS -q gpu
#PBS -l nodes=1:ppn=1:gpus=1
#PBS -l pmem=2000mb
#PBS -l walltime=00:30:00
#PBS -m b
#PBS -M yourEmailAddress
cd $PBS_O_WORKDIR
firstgpu=$( head -n 1 "$PBS_GPUFILE" )
gpuindex=${firstgpu:(-1)}
export DISPLAY=:0.$gpuindex
/global/software/ParaView/ParaView-4.3.1-Linux-64bit/bin/pvbatch spin.py
```

Batch script GPU visualization on WestGrid (cont.)

Method 3: via a batch script inside a VNC session

If for some reason (perhaps for debugging purposes) you prefer to work inside a remote graphical desktop:

1. Start the VNC session as described two slides ago
2. Inside a terminal in the VNC desktop:

```
vglrun /global/software/ParaView/ParaView-4.3.1-Linux-64bit/bin/paraview spin.py
```

Running batch visualizations on GPU-less nodes

- In the ideal world we would have several GPU visualization nodes on each cluster
- On GPU-less hardware ParaView needs to recompiled with OSMesa off-screen rendering library (software-based OpenGL without X11 dependency)
  - while at it, can also set PARAVIEW_USE_MPI=ON
  - will build *pvbatch*, *pvpython*, *pvserver*, but not the GUI client
  - Compute Canada tech staff will be happy to do it for you
- The command line in the PBS script will be one of:

```
/path/to/pvbatch --use-offscreen-rendering batch.py
```

```
mpiexec /path/to/pvbatch --use-offscreen-rendering batch.py
```

- Load balancing is handled automatically by ParaView for structured data, while unstructured data must be passed through D3 (Distributed Data Decomposition) filter

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

Example 2

- Data for our next example comes from the Physical Sciences Division of the Earth System Research Laboratory of the US National Oceanic & Atmospheric Administration
- Monthly measurements of the sea surface ice level from 1850/01 to 2013/12, at 1 degree latitude $\times$ 1 degree longitude resolution
- 164 years of data $\times$ 12 months/year $\times$ 360 $\times$ 180, one single-precision variable => 486 MB file
- Can be downloaded from `http://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/COBE2/icec.mon.mean.nc`
- ParaView can understand all 1968 variables inside one NetCDF file as a time sequence, however, for the purpose of our exercise I converted the data into 1968 separate files (`surface_0.vts`, ..., `surface_1967.vts`)
  1. loaded icec.mon.mean.nc into ParaView as "NetCDF files generic"
  2. File $\rightarrow$ Save Data as VTK StructuredGrid files, checked "Write all timesteps as file-series", selected Data Mode = Binary, Compressor Type = Zlib, used filename = "surface"

Demo time

1. Tools → Start Trace
2. Load `surface_0.vts` into ParaView
3. Colour by the ice sea level
4. Remove continents (ice level = 10^{20})
5. Set the range of displayed values to [0,1]
6. Save screenshot
7. Tools → Stop Trace

Running the script

- I put the optimized version of the script `evolve1.py` inside batch.zip
- Now let's put the main body of `evolve1.py` (after reading data and before writing the screenshot) into a loop:

```
for i in range(10):
    print 'processing surface_'+str(i)+'.vts'
    surface_0vts = XMLStructuredGridReader(FileName=['/Users/razoumov/Movies/data
        /surfaceIce/surface/surface_'+str(i)+'.vts'])
    ####################################
    # main body of evolve1.py goes here #
    ####################################
    SaveScreenshot('/Users/razoumov/Documents/03-webinar/frame%04d'%(i)+'.png',
        magnification=1, quality=100, view=renderView1)
```

and save it as `evolve2.py`

- Running the script will produce `frame0000.png`, ..., `frame0009.png` as expected, but it'll leave 20 objects in the pipeline browser (2 per file)
  - not a good idea if you have 1968 input files

Delete unused objects from the pipeline

Add the following at the end of the loop:

```
Delete(threshold1)
Delete(surface_0vts)
```

The script `evolve3.py` will produce `frame0000.png`, ..., `frame1967.png`

```
$ ffmpeg -r 10 -i frame%04d.png -c:v libx264 -pix_fmt yuv420p -vf "scale=trunc(iw
    /2)*2:trunc(ih/2)*2" iceCover.mp4
$ ls -lh iceCover.mp4
-rw-r--r--  1 razoumov  staff    15M 20 Mar 16:33 iceCover.mp4
```

Best way to produce a movie from the original file

The script `evolve4.py` produces `frame0000.png`, ..., `frame1967.png` without loops

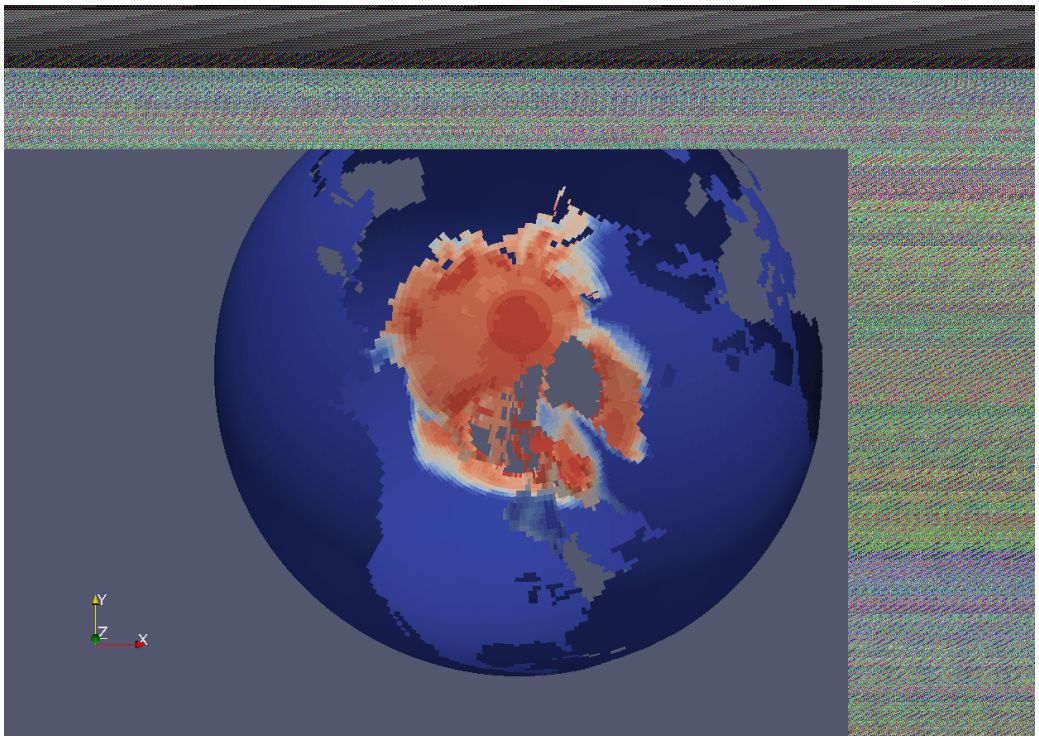
```
icecmonmeannc = NetCDFReader(FileName=['/Users/razoumov/Movies/data/surfaceIce/
    icec.mon.mean.nc'])
animationScene1 = GetAnimationScene()
# animationScene1.StartTime = -14824    # read time values from Properties of icec
    .mon.mean.nc in ParaView
# animationScene1.EndTime = -14794
animationScene1.UpdateAnimationUsingDataTimeSteps()
...
WriteAnimation('/Users/razoumov/Documents/03-webinar/111.png', Magnification=1,
    FrameRate=15.0, Compression=True)
```

Can run it remotely on the cluster

- For example, you can run it interactively inside a scheduled job:

```
qsub -q interactive -I -l nodes=1:ppn=1:gpus=1,walltime=0:30:00
          # wait for an interactive shell
firstgpu=$( head -n 1 "$PBS_GPUFILE" )
gpuindex=${firstgpu:(-1)}
export DISPLAY=:0.$gpuindex
/global/software/ParaView/ParaView-4.3.1-Linux-64bit/bin/pvbatch evolve4.py
```

- However, here is what you will likely get on parallel.westgrid.ca:



Virtual remote display settings

- Inside evolve4.py we have:

```
renderView1.ViewSize = [1246, 882]
```

- Problem lies with the limited virtual display resolution on the compute node (3 virtual screen for 3 GPU, respectively):

```
$ grep Virtual /etc/X11/xorg.conf

        Virtual     1024 768
        Virtual     1024 768
        Virtual     1024 768
```

- To fit into the virtual X11 display, account for the title bar and window borders $\Rightarrow$ the actual resolution of the image can't be higher than 1020×708
- If you need higher resolution, please let us know

ParaView Cinema

announced at `http://www.kitware.com/blog/home/post/734`

- New feature in ParaView 4.2.0 and higher (first released in 2014-Sep)
- Under active development, currently no documentation available ⇒ not exactly ready for end users yet
- Allows to render a large number of images in a batch ParaView's Python script, putting them into a specially formatted database, and then view these images from a web browser manipulating them as if it was live rendering
  - rotate, pan, zoom in/out, turn on/off different views (and filters)
- Everything to write databases is in `paraview.data_exploration` ParaView's Python module
- `https://github.com/Kitware/cinema` comes with Javascript code to play in a browser and many Python examples ➪ I'll show the output of generateDiskOutRefData.py
- Let me know if you are interested in exploring Cinema further

Questions?