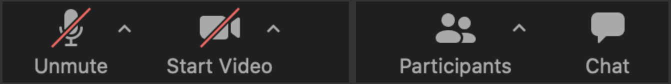
Image-based approach to large-scale visualization (Cinema Science)

ALEX RAZOUMOV
alex.razoumov@westdri.ca

Zoom controls

- Please mute your microphone and camera unless you have a question
- To ask questions at any time, type in Chat, or Unmute to ask via audio
  - please address chat questions to "Everyone" (not direct chat!)
- Raise your hand in Participants



- Email training@westdri.ca
- Our fall training schedule `https://bit.ly/wg2023b`
  - webinars, local workshops, autumn school

Why Cinema?

- Modern parallel simulations can produce huge amounts of data $\Rightarrow$ hard to visualize interactively, as each frame may take a while to render
  - client-server parallel rendering (ParaView, VisIt) may somewhat alleviate the problem, but you don't have a large computer at your interactive disposal at all times
  - client-server: interactive exploration, creating ParaView Python scripts
  - batch rendering for production visualization
  - one easy way to reproduce this problem: turn on OSPRay ray tracing, enable SamplesPerPixel=30 to reduce noise, try to rotate your object

Why Cinema?

- Modern parallel simulations can produce huge amounts of data $\Rightarrow$ hard to visualize interactively, as each frame may take a while to render
  - client-server parallel rendering (ParaView, VisIt) may somewhat alleviate the problem, but you don't have a large computer at your interactive disposal at all times
  - client-server: interactive exploration, creating ParaView Python scripts
  - batch rendering for production visualization
  - one easy way to reproduce this problem: turn on OSPRay ray tracing, enable SamplesPerPixel=30 to reduce noise, try to rotate your object
- Image-based visualization: instead of live rendering, pre-render all images for your full set of viewing parameters (viewing angles, time, features on/off, etc.), and then explore these images in a Cinema-enabled environment as if you were rendering live
  - can pre-render via parallel batch jobs on an HPC cluster
  - store images in a specially formatted database

Cinema history

- The original Cinema project `https://github.com/Kitware/cinema` ("An Image-based Approach to Extreme Scale In Situ Visualization and Analysis") was released in 2014
  - JavaScript package, write visualization pages to open in a web browser
  - last updated in March 2015, no longer maintained?
- Cinema Science `https://cinemascience.github.io` is a project from the "Data Science at Scale" group at LANL
  - Cinemasci Python toolkit `https://github.com/cinemascience/cinemasci`
  - documentation `https://cinemasciencewebsite.readthedocs.io`
  - last updated in July 2022
- Cinema Engine v2.0 `https://github.com/cinemascience/pycinema` is a Python toolkit for creating, filtering, transforming and viewing Cinema databases
  - introduces the concepts of *filter graphs* and *workspaces* to Cinema
  - authored mostly by the "Data Science at Scale" group at LANL
  - documentation `https://pycinema.readthedocs.io`
  - sample datasets `https://github.com/cinemascience/pycinema-data` (1.1G download)
  - last updated in May 2023

Spec D Cinema database

- Latest (4^{th}-generation) specification
- The database is a directory `databaseName.cdb` with a file `data.csv` (exactly this name) listing all parameters and related image/data filenames in the CSV format
- The database directory may be flat or may contain other subdirectories with data inside
- Files can be images or data
- Let's check a very simple example:

```
cd ~/tmp/pycinema-data/
tree sphere.cdb
bat sphere.cdb/data.csv
```

Cinemasci Python toolkit

- Installation

```
virtualenv ~/cinemasci-env
source ~/cinemasci-env/bin/activate
pip install --upgrade pip
pip install cinemasci        # will install most of its dependencies too, including jupyter
pip install opencv-python    # needed for some examples in their tutorial
python -m ipykernel install --user --name=cinemasci --display-name "cinemasci"   # optional
jupyter nbextension install --py widgetsnbextension --user                       # optional
jupyter nbextension enable widgetsnbextension --user --py                        # optional
...
deactivate
```

- Usage: several options

```
source ~/cinemasci-env/bin/activate
cd ~/tmp

(1) python -m cinemasci.server --port 8200 --viewer view --data pycinema-data/sphere.cdb

(2) cinema view --viewer view -d pycinema-data/sphere.cdb --browser firefox

(3) jupyter notebook
--- start cinemasci notebook
import cinemasci.pynb
viewer = cinemasci.pynb.CinemaViewer()
viewer.load("pycinema-data/sphere.cdb")
```

- Cinema:View (`--viewer view`) shows an interactive 3D view with variable sliders
  - sometimes shows an empty page for me, even when there is no problem with the database ... can be traced to a broken pipe error inside Python ...
- Cinema:Explorer (`--viewer explorer`) presents individual images on a grid with an interactive parallel coordinates graph at the top
  - quite often does not load for me at all ... ⇒ won't show it here

---

- Another standalone viewer `https://github.com/cinemascience/cinema_view`
  - also allows you to compare several databases side by side (must have same number of files)
  - works great every time!
  - must enable local file access in your web browser `https://github.com/cinemascience/cinema_view`

```
git clone https://github.com/cinemascience/cinema_view
cd cinema_view
add your database to ./cinema/view/1.1/databases.json
allow local file access in your web browser
open cinema_view.html
```

- Another standalone viewer `https://github.com/cinemascience/cinema_scope`
  - Qt-based ⇒ need Qt to compile and use it

Pycinema Python toolkit

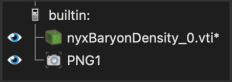
- Installation

```
virtualenv ~/pycinema-env
source ~/pycinema-env/bin/activate
pip install --upgrade pip
pip install pycinema     # will install most of its dependencies too, including jupyter
python -m ipykernel install --user --name=pycinema --display-name "pycinema"   # optional (but see below)
...
deactivate
```

- Usage
  - trying to run any of the included `examples/ipynb/*.ipynb` leads to internal errors ...
  - the command-line tool works really well

```
source ~/pycinema-env/bin/activate
cinema view ~/tmp/pycinema-data/sphere.cdb
cinema explorer ~/tmp/pycinema-data/sphere.cdb
cinema supportedPythonScript.py    # see script examples in pycinema/examples/theater
                                   # in their source code on GitHub; more details at
                                   # https://pycinema.readthedocs.io/en/latest/scripts.html
```

Cinema databases from ParaView Extractors



Let's start with a demo using the Cinema tutorial dataset from SC'20

```
git clone https://github.com/cinemascience/cinema_tutorial_2020-SC
```

- In ParaView: File | Open `~/tmp/cinema_tutorial_2020-SC/data/nyxBaryonDensity/*.vti` all 18 timesteps as a collection, hit Apply
- Surface view, colour by baryonDensity
- Extractors | Image | PNG, in Properties set Camera Mode = Phi-Theta, use default (=6) Phi / Theta Resolution for now, click Apply $\Rightarrow$ this one will create $6^2 * 18 = 648$ images
- Creare the database directory

```
mkdir -p ~/tmp/case01/nyxBaryonDensity.cdb
```

- File | Save Extracts..., set Extracts Output Directory = `~/tmp/case01/nyxBaryonDensity.cdb`, click Generate Cinema Specification, wait a couple of mins to write all 648 images into our 3D (time+Phi+Theta) database
- Check the images and `data.csv`

```
source ~/pycinema-env/bin/activate
cinema view ~/tmp/case01/nyxBaryonDensity.cdb
```

PNG Extractor Properties

- If Camera Mode = Static and Trigger=TimeStep
  $\Rightarrow$ generate a 1D time sequence (no rotation)
- As far as I can tell, Trigger=TimeStep and Trigger=TimeValue produce the same output
- Camera Mode = Python is undocumented
  - probably a placeholder for future development?

Scripting the Extractor

1. Repeat the previous workflow up until (but not including) File | Save Extracts...
   - create a visualization + a PNG Extractor
2. File | Save State... to a Python script `export.py`
3. These should already be there:

```
pNG1 = CreateExtractor('PNG', renderView1, registrationName='PNG1')
pNG1.Trigger = 'TimeStep'
pNG1.Writer.FileName = 'RenderView1_{timestep:06d}{camera}.png'
pNG1.Writer.ImageResolution = [1920, 1080]
pNG1.Writer.Format, pNG1.Writer.ResetDisplay = 'PNG', 1
pNG1.Writer.CameraMode = 'Phi-Theta'
pNG1.Writer.PhiResolution, pNG1.Writer.ThetaResolution = 10, 10
```

Ideal for running on HPC!

4. Optionally can control the start/end timesteps:

```
# pNG1.Trigger.UseStartTimeStep, pNG1.Trigger.UseEndTimeStep = 1, 1
# pNG1.Trigger.StartTimeStep, pNG1.Trigger.EndTimeStep = 0, 2
```

5. Add the following:

```
SaveExtracts(ExtractsOutputDirectory='/Users/razoumov/tmp/case06', GenerateCinemaSpecification=1)
```

6. Run the script: `pvpython export.py`

Creating custom databases

- Extractors are very limited: only time + Phi + Theta
- What if you want other variables? What if you want to turn on/off layers or switch representations via sliders?

Creating custom databases

- Extractors are very limited: only time + Phi + Theta
- What if you want other variables? What if you want to turn on/off layers or switch representations via sliders?

- For 1D databases, you can use keyframe animation

Creating custom databases

- Extractors are very limited: only time + Phi + Theta
- What if you want other variables? What if you want to turn on/off layers or switch representations via sliders?

- For 1D databases, you can use keyframe animation
- For multidimensional databases, you can code everything in Python with very little effort!

1D Cinema database via Animation

1. In ParaView load `sineEnvelope.nc`, apply Contour filter, rescale to custom range [0,2]
2. View | Animation View, animate Contour | Isosurfaces, click + to create a timeline, set the range [0,2], set 100 frames
3. File | Save Animation... to `~/tmp/case03/frame*.png`, select full HD resolution
4. Edit and then run the following script:
    - use fixed precision for all variables in the database for smooth sliders!

```
#! /usr/bin/env python
import sys, os, pandas as pd, numpy as np
from glob import glob

if len(sys.argv) == 4:
    inputDir = sys.argv[1]
    startValue = float(sys.argv[2])
    endValue = float(sys.argv[3])
else:
    sys.stderr.write("Usage: generateCinemaDB.py "+
                     "imageDir startValue endValue\n")
    sys.exit(0)
```

```
print('Converting the directory to Cinema database ...')

files = glob(inputDir+"/*.png")
files.sort()
files = [os.path.basename(f) for f in files]

density = np.linspace(startValue, endValue, len(files))
density = ["{:7.4f}".format(rho) for rho in density]

df = pd.DataFrame({'density': density, 'FILE': files})
df.to_csv(inputDir+"/data.csv", index=False)
```

```
~/Documents/10-cinema/generateSingleVariable.py ~/tmp/case03 0 2
```

3D Cinema database via Python scripting

Animating 2 angles and the isosurface

1. In ParaView load `sineEnvelope.nc`, apply Contour filter
2. File | Save State... as a Python script
3. Simplify the script
4. Add custom lines at the end (see next slide) and then run the script:
   `pvpython generateMultiVariable.py`
5. On presenter's laptop the complete database is in `~/tmp/case04`

3D Cinema database via Python scripting (cont.)

```
import numpy as np, pandas as pd
nphi, ntheta, ncontour = 50, 3, 5;     counter, tilt = 0, 25
phi, theta, contour, files = [], [], [], [] # these will form dataframe columns
density = np.linspace(0.1, 1.9, ncontour);    camera = GetActiveCamera()
for j in range(ntheta):
    if j==0:
        elevation = 0              # "camera-centred" track
    if j==1:
        camera.Elevation(-tilt)    # "camera-below" track
        elevation = -tilt
        camera.SetViewUp(0,np.cos(np.radians(tilt)),np.sin(np.radians(tilt)))    # view-up (rotation) vector
    if j==2:
        camera.Elevation(70)       # "camera-above" track
        elevation = tilt
        camera.SetViewUp(0,np.cos(np.radians(tilt)),-np.sin(np.radians(tilt)))    # view-up (rotation) vector
    for i in range(nphi):
        if i==0:
            azimuth = 0
        else:
            camera.Azimuth(360./(nphi-1))
            azimuth += 360./(nphi-1)
        print("camera = %3.4f %3.4f %3.4f"%(camera.GetPosition()), "  frame = %1d/%1d"%((counter+1,nphi*ntheta*ncontour)))
        for k in density:
            contour1.Isosurfaces = [k]
            counter += 1
            filename = 'frame%04d'%(counter)+'.png'
            SaveScreenshot(dir+filename)
            phi.append("{:7.4f}".format(azimuth))    # fixed-length string
            theta.append(elevation)
            contour.append("{:6.4f}".format(k))      # fixed-length string
            files.append(filename)

df = pd.DataFrame({'phi': phi, 'theta': theta, 'contour': contour, 'FILE': files})
df.to_csv(dir+"data.csv", index=False)
```

More complex case: CPU-intensive rendering with OSPRay

Animating a layer on/off, material selection, azimuthal angle

- OSPRay is CPU intensive and may take a while at high quality ⇒ can be a miserable interactive experience (Progressive Rendering will help) ⇒ perfect case for pre-rendering
- Let's load the state file `glass.py` without the custom Cinema lines at the end
  - comment out the Cinema lines, set `renderView1.SamplesPerPixel=1`
  - explore the scene: contour and 2 clips
- Add custom lines at the end (see next slide) and then run the script: `pvpython glass.py`
- On presenter's laptop the complete database is in `~/tmp/case05`

More complex case: CPU-intensive rendering with OSPRay (cont.)

```
import numpy as np, pandas as pd
nphi = 50
counter = 0
phi, clip, material, files = [], [], [], [] # these will form dataframe columns
camera = GetActiveCamera()
for clipState in ['show', 'hide']:
    if clipState=='hide':
        Hide(clip1)
    for composition in ['Glass_Thick', 'Metal_Lead_brushed']:
        contour1Display.OSPRayMaterial = composition
        for i in range(nphi):
            if i==0:
                azimuth = 0
            else:
                camera.Azimuth(360./(nphi-1))
                azimuth += 360./(nphi-1)
            print("camera = %3.4f %3.4f %3.4f"%(camera.GetPosition()), "  frame = %1d/%1d"%((counter+1,nphi*2*2)))
            counter += 1
            filename = 'frame%04d'%(counter)+'.png'
            SaveScreenshot(dir+filename)
            phi.append("{:7.4f}".format(azimuth))    # fixed-length string
            clip.append(clipState)
            material.append(composition)
            files.append(filename)

df = pd.DataFrame({'phi': phi, 'clip': clip, 'material': material, 'FILE': files})
df.to_csv(dir+'data.csv', index=False)
```

General thoughts so far

- It makes sense to pre-render only those frames that are expensive to render, otherwise interactive live visualization will work just fine
- For multiple variables, the number of combinations/frames grows very quickly
  - consider $n_\phi = 30$, $n_\theta = 30$ (smooth rotation!) $\Rightarrow$ 900 frames per every combinations of the rest of your parameters $\Rightarrow$ this can easily grow to $10^{\sim 4.5}$ frames
  - not only will it take a very long time to render, but will use a lot of disk space as well ...
- My suggestion: use $n_\phi = 30$, $n_\theta = 1$ and few other parameters in moderation
- Litmus test: compare the size of the original dataset to the size of your Cinema database
- Use fixed precision for all variables in the database, otherwise the sliders in pycinema will become very choppy
- Can easily script everything on an HPC cluster and submit as a batch job

In-situ writing to a Cinema database via ParaView Catalyst

Watch our webinar "In-situ visualization with ParaView Catalyst2" from September 2022
`https://bit.ly/vispages`

1. Instrument your simulation code with the Catalyst library
2. Generate a representative dataset, e.g. `./simCode --output dataset-%04ts.vtpd` (if coded; otherwise, can create it by hand)
3. Load it into ParaView and create your visualization interactively
4. Apply Extractors | Image | PNG
5. File | Save Catalyst State to save it as `extract-image.py`
   - check "Generate Cinema specification to summarize generated extracts in a file named data.csv under the Extracts Output Directory"
6. Make sure `registrationName` in the script matches the data channel name in the simulation code
7. Run `./simCode extract-image.py` to generate PNG images