Visualisation scientifique avec ParaView
2e partie

Pier-Luc St-Onge
et Alex Razoumov

Digital Research Alliance of Canada

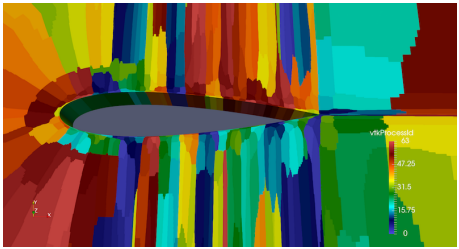
✔ Diapositives, données et codes : `https://folio.vastcloud.org/ecolehivernale`

- le lien "Fichier ZIP ..." permet de télécharger `paraview.zip` (∼30 Mo)
- une fois le contenu extrait, on obtient `codes/`, `data/` et `slides2.pdf` (en anglais)

✔ Installez ParaView 6.0.x sur votre ordinateur à partir de `http://www.paraview.org/download`

Plan de l'atelier

- Scripter des visualisations
  - voir les façons d'exécuter des scripts
  - tester quelques exemples de scripts
  - sauvegarder une trace des opérations manuelles
  - utiliser des filtres et des sources programmables



SCRIPTER DES VISUALISATIONS VIA PARAVIEW

Des scripts pour automatiser la visualisation (1/2)

Tutoriel : `https://docs.paraview.org/en/latest/Tutorials/SelfDirectedTutorial/batchPythonScripting.html`

- Pourquoi utiliser des scripts? Pour :
  - automatiser des opérations fastidieuses ou répétitives, par exemple la création d'images pour une vidéo
  - documenter et enregistrer un flux de travail
  - utiliser ParaView via la ligne de commande et/ou via des tâches de calcul sur une grappe
- Dans ParaView, l'option *View* → *Python Shell* ouvre un interpréteur Python
  - dans l'invite >>>, on peut y écrire ou copier un script
  - le bouton *Run Script* permet de charger et d'exécuter un script à partir d'un fichier

Des scripts pour automatiser la visualisation (2/2)

Tutoriel : https://docs.paraview.org/en/latest/Tutorials/SelfDirectedTutorial/batchPythonScripting.html

- Ensuite, pour utiliser ParaView à partir d'un terminal régulier, la variable d'environnement `$PATH` doit avoir un ou deux chemins, selon le système :
  - Linux : `/usr/local/bin` ou autre répertoire d'installation
  - MacOS : `/Applications/Paraview*.app/Contents/bin`
  - Windows : `C:\Program Files\ParaView*\bin`
- `pvpython` vous ouvre un interpréteur Python connecté à un serveur ParaView (local ou distant) sans interface graphique
- `pvbatch --force-offscreen-rendering script.py` est une application séquentielle (ou parallèle sur certaines machines) utilisant un serveur ParaView local. ✐ Assurez-vous d'enregistrer votre visualisation
- `paraview --script=codes/displayWireframe.py` permet de démarrer l'interface graphique de ParaView et d'exécuter automatiquement le script

Autres façons d'utiliser des scripts Python dans ParaView

1. Filtre *Python Calculator* (versus le *Calculator* régulier) pour créer de nouveaux tableaux de données
   - permet de traiter les données comme des tableaux NumPy
   - permet d'utiliser la syntaxe courte `0.5*rho*mag(V)**2` ou longue `0.5*inputs[0].PointData['rho']*inputs[0].PointData['V']**2`
   - permet d'accéder aux données de points ou de cellules à partir de plusieurs entrées et à plusieurs pas de temps
   - permet de contrôler la précision ou le type des données, mais pas leur discrétisation
2. Source ou filtre programmable pour générer une nouvelle géométrie
   - pour créer de nouvelles discrétisations ou des objets VTK
   - utile pour créer des objets personnalisés à partir de zéro, par exemple projeter un volume sur un plan
   - une autre façon de créer votre propre lecteur de fichiers personnalisé (si le format n'est pas pris en charge nativement)
3. Fichiers d'état en Python : plus robustes que les fichiers `*.pvsm` pour les flux de travail complexes

Un premier script

- Faites afficher l'interpréteur avec *View* → *Python Shell*
- Cliquez sur *Run Script* et sélectionnez `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()                           # make it visible in the view
Render()
```

- Dans le *Python Shell*, on peut obtenir de l'information sur les différents noms :

```
help(paraview.simple)  # affiche une page d'aide du module paraview.simple
dir(paraview.simple)   # affiche uniquement les noms disponibles
help(Sphere)
help(sphere)  # pour voir les attributs de l'objet
help(Show)
```

Utiliser des filtres

- Via le bouton *Run Script*, exécutez `codes/displayWireframe.py`

displayWireframe.py

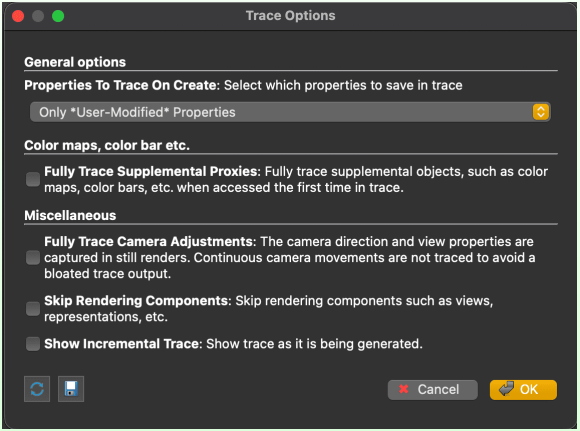
```
from paraview.simple import *

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

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

Show()                                    # make the last object visible in the view
Render()
```

- Essayez de remplacer `Show()` par `Show(sphere)`
- Essayez aussi de remplacer `Render()` par `SaveScreenshot('wireframe.png')` et d'exécuter le script avec `pvbatch`

L'outil *Trace*

Cet outil génère un script Python à partir des opérations dans l'interface graphique

- *Tools* → *Start Trace*
- *Tools* → *Stop Trace*



Exercice : créer un script et l'exécuter en mode *offscreen*

1. Créez un script `script.py` avec l'interface graphique de ParaView
   - utilisez *Tools* → *Start Trace*
   - chargez `data/sineEnvelope.nc` et affichez une isosurface à $\rho = 0.15$
   - enregistrez la vue en PNG (*File* → *Save Screenshot*) et faites *Tools* → *Stop Trace*
2. Essayez votre script avec la commande `pvbatch(.exe)` sur votre ordinateur

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

   - Linux: `pvbatch` devrait être dans un des répertoires `bin` de votre système d'exploitation
   - Mac: `pvbatch` devrait être dans `/Applications/ParaView*.app/Contents/bin`
   - Windows: `pvbatch.exe` devrait être dans `C:\Program Files\ParaView*\bin`
3. Si vous avez un compte à l'Alliance de recherche numérique du Canada, vous pourriez importer le script et les données sur une grappe de calcul, adapter les chemins dans le script et exécuter le script avec :

```
$ module load paraview/6.0.0
$ pvbatch --force-offscreen-rendering script.py
```

Transmission d'informations le long du pipeline

... et d'autres fonctions de haut niveau utiles pour le flux de travail

- `GetSources()` retourne une liste d'objets de pipeline
- `GetActiveSource()` retourne l'objet actif (sélectionné)
- `SetActiveSource()` pour spécifier l'objet à rendre actif
- `GetRepresentation()` retourne la *représentation* de l'objet actif selon la vue active
- `GetActiveCamera()` retourne la caméra active de la vue active
- `GetActiveView()` retourne la vue active
- `CreateRenderView()` crée une nouvelle vue de rendu 3D standard
- `ResetCamera()` réinitialise la caméra pour inclure la scène entière tout en préservant l'orientation (ou ne fait rien ☺)

Il y a un certain chevauchement entre ces deux éléments :

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

Animation de la caméra via un script

1. Chargez `data/sineEnvelope.nc` et affichez une isosurface à $\rho = 0.15$
2. Dans le *Python Shell*, vérifiez que le point focal est bien au centre de l'objet
   - dans une scène 3D, le point focal est la coordonnée de l'espace vers laquelle la caméra regarde et autour de laquelle elle tourne

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

   - au besoin, recentrez le point focal sur les objets en vue avec $\Rightarrow$ `ResetCamera()`
3. Recherchez la fonction de rotation azimutale

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

4. Faites une rotation de 10° autour d'un axe pointant l'azimut

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

Animation de la caméra : rotation complète

5. Programmez une rotation complète et sauvegardez chaque rendu

```
import os
os.chdir(os.path.expanduser('~/Desktop'))
os.makedirs('pv-frames', exist_ok=True)

nframes = 60
angle = 360. / nframes  # degrees per image
for i in range(nframes):
    print(v1.CameraPosition)
    camera.Azimuth(angle)
    SaveScreenshot('pv-frames/frame%04d'%(i)+'.png')
```

6. (Optionnel) Créez une vidéo à 30 ips

```
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
```

Animation de la caméra : voler vers le point focal

1. Au besoin, réinitialisez la vue manuellement ou avec `ResetCamera()`
2. Ensuite, faites avancer la caméra le 2/3 de la distance vers le point focal

```
initialCameraPosition = v1.CameraPosition[:]   # force a real copy
nframes = 100
for i in range(nframes):
    coef = (i + 0.5)/nframes * 2./3  # runs between 0 and 2/3
    v1.CameraPosition = [((1.-coef)*a + coef*b) \
      for a, b in zip(initialCameraPosition, v1.CameraFocalPoint)]
    print(f'{coef:.5f}', v1.CameraPosition)
    SaveScreenshot('pv-frames/out%04d'%(i)+'.png')
```

3. (Optionnel) Créez une vidéo

```
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
```

Exercice : créer un script et l'exécuter en mode *offscreen*

- Modifiez le script du précédent exercice pour créer une animation
  - essayez une animation de la caméra basée sur les précédents exemples, le tout à partir du terminal
  - essayez d'animer une isosurface en faisant varier ρ

Extraction de données à partir d'objets VTK (1/2)

Ajustez le `dir` et exécutez le script via *Python Shell* → *Run Script* ou `pvpython`

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

dir = '/Users/razoumov/training/paraviewWorkshop/data/'   # edit the path
data = NetCDFReader(FileName=[dir+'sineEnvelope.nc'])
print(type(data))                  # class 'paraview.servermanager.NetCDFReader'

local = servermanager.Fetch(data) # get this data from the server
print(local.GetNumberOfPoints())
print(local.GetDimensions())

# the following are populated with data only after servermanager.Fetch(data) is called
pointInfo = data.GetDataInformation().GetPointDataInformation()
narrays = pointInfo.GetNumberOfArrays()
print(narrays)

for i in range(narrays):
    print(pointInfo.GetArrayInformation(i).GetName())

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

Extraction de données à partir d'objets VTK (2/2)

Dans cette deuxième partie du script, l'intégration des données dans des tableaux NumPy permettrait d'effectuer des calculs de post-traitement

```
vtkArray = local.GetPointData().GetArray('density')
print(vtkArray.GetDataSize(), vtkArray.GetRange())

for i in range(10):
    print(vtkArray.GetValue(i))       # values at the first 10 points -- awkward to use

from vtk.numpy_interface import dataset_adapter as dsa

wrapped = dsa.WrapDataObject(local)   # wrap it in a NumPy-friendly object
flatArray = wrapped.PointData['density']
print(flatArray.shape)
print(flatArray)

import numpy as np

dims = local.GetDimensions()
array3d = np.reshape(flatArray, (dims[2], dims[1], dims[0]))
print(array3d.shape)
print(array3d)
```

Utiliser des bibliothèques tierces à partir de `pvpython`

- L'interpréteur `pvpython` inclut quelques bibliothèques tierces courantes telles que `numpy`, `scipy` et `pandas`
- Que faire si vous souhaitez utiliser d'autres bibliothèques qui n'ont pas été incluses avec ParaView?

Utiliser des bibliothèques tierces à partir de `pvpython`

1. Sur une grappe de l'Alliance, vérifiez la version Python de ParaView

```
module load paraview
pvpython                          # supposons que la sortie indique Python 3.13.2
```

2. Avec le module Python le plus proche, installez votre bibliothèque dans un environnement virtuel

```
module avail python               # python/3.13.2 est l'un des modules disponibles
module load python/3.13.2
virtualenv --no-download ~/env-astro   # nouvel environnement virtuel dans ~/env-astro
source ~/env-astro/bin/activate
pip install --no-index --upgrade pip
pip install --no-index xarray     # installe une bibliotheque externe dans cet environnement
```

3. La prochaine fois que vous vous connecterez à la grappe, démarrez `pvpython`

```
module load paraview
pvpython
```

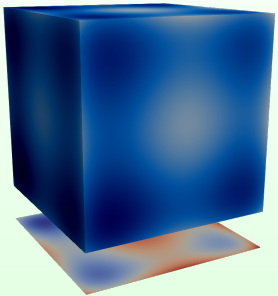
4. Chargez votre nouvel environnement virtuel directement depuis Python

```
venv_path = '/home/userXY/env-astro/lib/python3.13/site-packages'
import sys
sys.path.insert(0, venv_path)    # activer l'environnement virtuel a partir de Python
from paraview.simple import *
import xarray      # cette bibliotheque xarray provient de votre environnement virtuel
```

5. Dans vos scripts de tâches de calcul, remplacez `pvpython` par `pvbatch`

Création/modification d'objets VTK

Supposons que nous voulions tracer une projection d'un ensemble de données cubique (par exemple `data/stvol.nc`) le long de l'un de ses axes principaux, ou effectuer une autre transformation pour laquelle il n'existe pas de filtre



- Un filtre *Calculator* ou *Python Calculator* ne peut modifier la géométrie …

Filtre programmable

1. Ajoutez un *Programmable Filter* avec *OutputDataSetType* = `vtkUnstructuredGrid`
2. Copiez-collez le code de `codes/projectionUnstructured.py` dans la propriété *Script* du filtre (testé dans ParaView 6.0.1)

```
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, z, column = inputs[0].GetPoint(i)[0:2], -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())
```

Voir le webinaire : `https://training.westdri.ca/tools/visualization/#programmable`