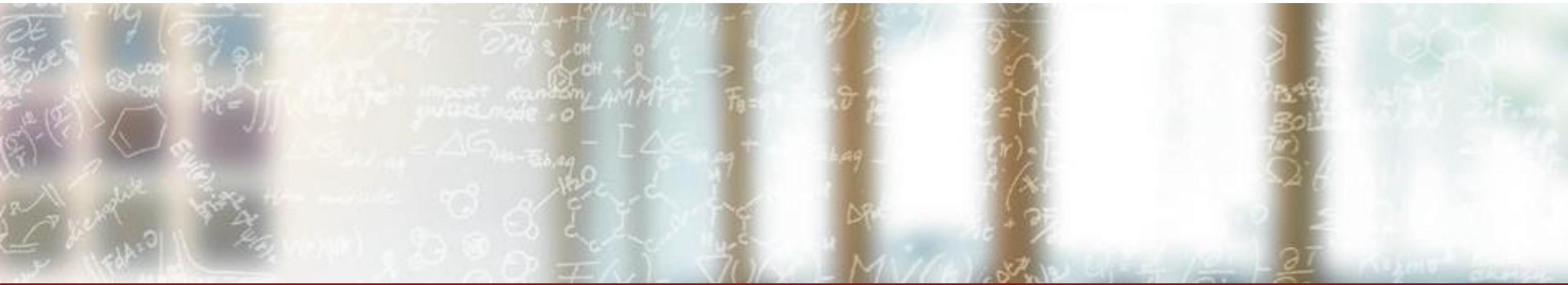




CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



ParaView in a Jupyter notebook

Dr. Jean M. Favre, CSCS

September 1, 2020

Foreword

- ParaView is a very mature 3D parallel visualization ecosystem in use at CSCS for many years.
- Usually, users would create a client-server connection from their remote desktop to a set of compute nodes on Piz Daint.
- ParaView uses an efficient and productive interface via Python scripts:
 - The client will read Python commands and the execution takes place [in parallel], on the server side
- A jupyter notebook can execute, stand-alone, or connected to a ParaView parallel server.

Overlook

Analyze data in a familiar, python-driven environment and create 3D interactive visualizations.

No need for a desktop ParaView client, and the [*sometimes complicated*] connection process in client-server mode.

Access to a GPU if you do not have a powerful desktop.

Outline

- Hello sphere ParaView program
- Hello sphere ParaView program + ipywidgets
- Hello sphere ParaView parallel program
- Local notebook connected to remote ParaView session on Piz Daint
- Numpy-to-Paraview
- Raytracing demo

Pre-requisites if you use the **Hybrid** partition

Edit your `$HOME/.jupyterhub.env`

```
module load PyExtensions h5py/2.8.0-CrayGNU-19.10-python3-serial
```

```
module load ParaView/5.8.0-CrayGNU-19.10-EGL
```

See the presentation by Tim Robinson@cscs for all generic details.

Pre-requisites if you use the **Multicore** partition

Edit your \$HOME/.jupyterhub.env

```
module load PyExtensions h5py/2.8.0-CrayGNU-19.10-python3-serial
```

```
module load ParaView/5.8.0-CrayGNU-19.10-OSMesa
```

See the presentation by Tim Robinson@cscs for all generic details.

Hello_Sphere-ParaView.0.ipynb

- Standard ParaView Python initialization
- Standard pipeline
 - ParaView Source
 - ParaView Representation
 - Render

+

- PVDisplay widget (contributed by NVIDIA)

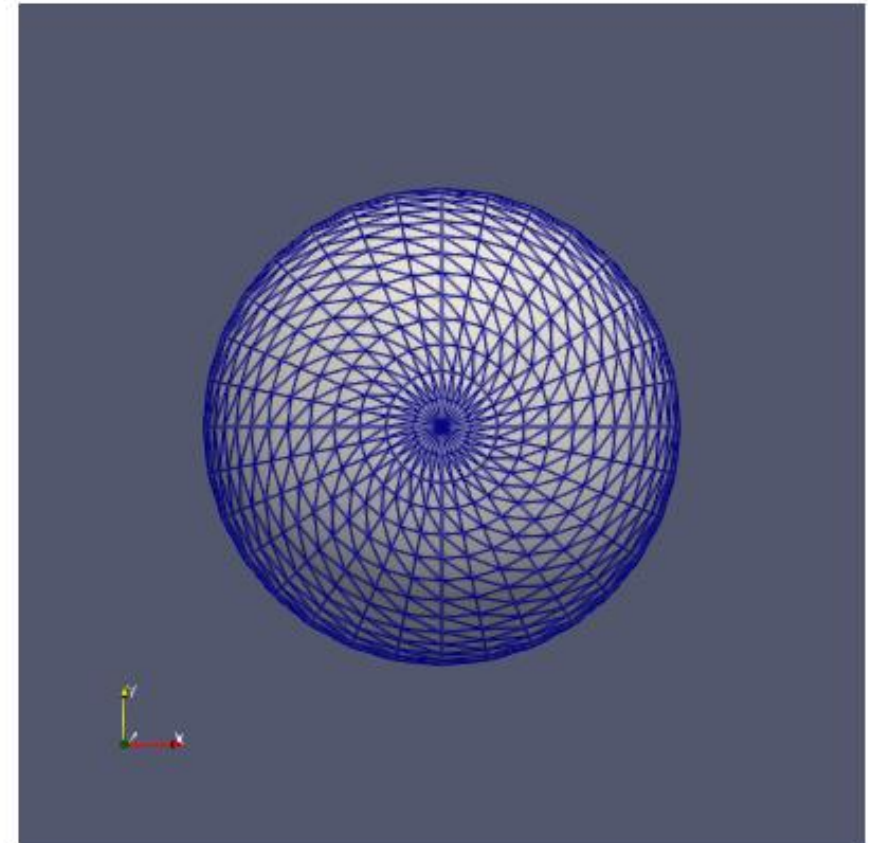
ParaView Hello Sphere Test

```
[1]: from paraview.simple import *

[2]: sphere = Sphere(ThetaResolution=32, PhiResolution=32)

    rep = Show()
    rep.Representation = "Surface With Edges"

[3]: from ipyparaview.widgets import PVDisplay
    disp = PVDisplay(GetActiveView())
    w = display(disp)
```



Hello World (Sphere) augmented with ipywidgets

sphere.ListProperties()

Attach PhiResolution and ThetaResolution to an IntSlider

```
['Center',  
'EndPhi',  
'EndTheta',  
'PhiResolution',  
'PointData',  
'Radius',  
'StartPhi',  
'StartTheta',  
'ThetaResolution']
```

Hello World (Sphere) augmented with ipywidgets

```
sphere.ListProperties()
```

Attach PhiResolution and ThetaResolution to an IntSlider

```
from ipywidgets import interact, IntSlider
```

```
# automatically triggers a pipeline update, and a render event
```

```
def Sphere_resolution(res):
```

```
    sphere.ThetaResolution = sphere.PhiResolution = res
```

```
    sphere.UpdatePipeline()
```

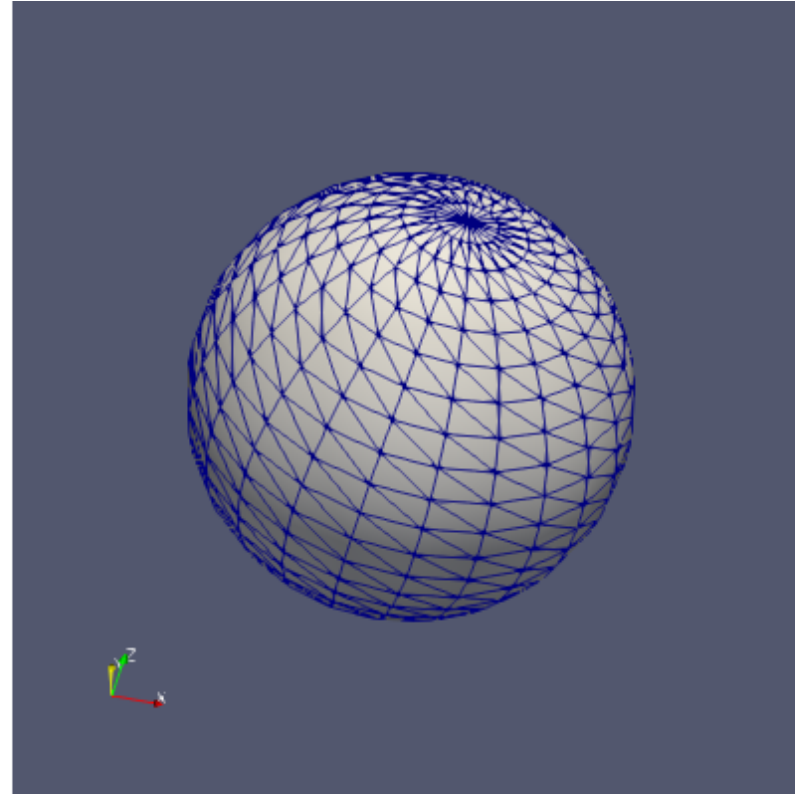
```
i = interact(Sphere_resolution,
```

```
            res=IntSlider(min=3, max=48, step=1, value=12)
```

```
)
```

```
['Center',  
'EndPhi',  
'EndTheta',  
'PhiResolution',  
'PointData',  
'Radius',  
'StartPhi',  
'StartTheta',  
'ThetaResolution']
```


Hello_Sphere-ParaView.1.ipynb



```
[6]: # Interact from ipywidgets gives us a simple way to interactively control values  
# with a callback function  
from ipywidgets import interact, IntSlider  
  
# set the Theta and Phi resolution and trigger a pipeline update  
def Sphere_resolution(res):  
    sphere.ThetaResolution = sphere.PhiResolution = res  
    sphere.UpdatePipeline()  
  
i = interact(Sphere_resolution, res=IntSlider(min=3, max=48, step=1, value=12))
```

Caveat

It seems like the regular `SaveScreenshot()` no longer works

```
def SaveImage(filename):  
    from vtk import vtkPNGWriter  
    img_writer = vtkPNGWriter()  
    img_writer.SetInputConnection(dispatcher.GetOutputPort())  
    img_writer.SetFileName(filename)  
    img_writer.Write()
```

```
SaveImage("/users/jfavre/screenshot.png")
```



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Parallel visualization scenarios

Classic console output for client-server connection

Accepting connection(s): rancate:1100

#SBATCH --job-name=pvserver

#SBATCH --nodes=1

#SBATCH --ntasks-per-node=8

#SBATCH --ntasks=8

#SBATCH --time=00:20:00

#SBATCH --partition=debug

#SBATCH --constraint=gpu

srun -n 8 -N 1 --cpu_bind=sockets pvserver -rc -ch=daint103.cscs.ch -sp=1100

Submitted batch job 123456789

Hello_Sphere-ParaView-Parallel.ipynb (on-the-node parallelism)

This notebook is useful as a minimal example. It creates a synthetic data source (a sphere), and creates a polygonal display of it. Then, it creates a ParaView display widget showing the primary render view. The notebook further demonstrates how we may use interaction widgets (sliders), to change the resolution of the sphere.

```
[1]: from paraview.simple import *
     from paraview.modules.vtkRemotingCore import vtkProcessModule

[2]: # to run in parallel on-the-allocated node, one would issue an srun command
     # at the terminal:
     # module load ParaView/5.8.0-CrayGNU-19.10-EGL
     # srun -n 8 `which pvserver`
     # followed by a Connect() command

     Connect("localhost")

[2]: Connection (cs://localhost:11111) [2]

[3]: rank = vtkProcessModule.GetProcessModule().GetPartitionId()
     nbprocs = servermanager.ActiveConnection.GetNumberOfDataPartitions()
     info = GetOpenGLInformation(location=servermanager.vtkSMSession.RENDER_SERVER)
     print("nbprocs= ",nbprocs)

nbprocs= 8
```

```
jfavre@nid06882:~> module avail ParaView
```

```
----- /apps/daint/UES/jenkins/7.0.UP01/gpu/easybuild/modules/all
ParaView/5.7.0-CrayGNU-19.10-EGL(default) ParaView/5.8.0-CrayGNU-19.10-EGL
```

```
jfavre@nid06882:~> module load ParaView/5.8.0-CrayGNU-19.10-EGL
```

```
jfavre@nid06882:~>
```

```
jfavre@nid06882:~> srun -n 8 pvserver
```

```
Waiting for client...
```

```
Connection URL: cs://nid06882:11111
```

```
Accepting connection(s): nid06882:11111
```

```
Client connected.
```

```
□
```

Local jupyter lab (on your desktop) + parallel pv server on Piz Daint

Terminal window 1

```
from paraview.simple import *  
ReverseConnect("1100")
```

N.B. The client is put in wait mode with the call above, **before** issuing the srun command on compute node(s)

- get your userid on Piz Daint (mine is 1100)
- Replace the call `Connect("localhost")` by a `ReverseConnect(port)`
- Use id as port number
- `ReverseConnect("1100")`

Local jupyter lab (on your desktop) + parallel pv server on Piz Daint

Terminal window 1

```
from paraview.simple import *  
ReverseConnect("1100")
```

Terminal window 2

- open an ssh tunnel on port 1100.
- select one login node. Here we use daint101.cscs.ch

```
ssh -l jfavre -R 1100:localhost:1100 daint101.cscs.ch
```

```
module load daint-gpu
```

```
module load ParaView/5.8.0-CrayGNU-19.10-EGL
```

```
srun -C gpu -p debug -t 00:10:00 -n 8 -N 1 \
```

```
pvserver -rc -ch=daint101.cscs.ch -sp=1100
```

Questions?

- Use the chat for Q/A



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

There's more than “Hello World”

Numpy_to_ParaView.ipynb

- Select a VTK grid type
 - vtkImageData
 - vtkRectilinearGrid
 - vtkStructuredGrid
 - vtkUnstructuredGrid
- Attach the numpy arrays (coordinates, scalar and vector field) to the VTK Object (zero-memory copy)
- Make a ParaView object holding the VTK object
- Render

```
[ ]: from paraview.simple import *  
import numpy as np
```

Make numpy arrays

```
[ ]: dims = [150, 150, 150]  
np_data = np.random.rand(np.prod(dims))
```

Make a vtkImageData

```
[ ]: from vtk import vtkImageData  
from paraview import numpy_support  
  
ImageData = vtkImageData()  
ImageData.SetExtent(0, dims[0]-1, 0, dims[1]-1, 0, dims[2]-1)  
  
vtk_data = numpy_support.numpy_to_vtk(np_data)  
vtk_data.SetName("scalarA")  
ImageData.GetPointData().AddArray(vtk_data)
```

Make a ParaView object holding the vtkImageData

```
[ ]: trivialproducer = PVTrivialProducer()  
obj = trivialproducer.GetClientSideObject()  
obj.SetOutput(ImageData)
```

```
[ ]: rep = Show(trivialproducer)  
ColorBy(rep, ("POINTS", "scalarA"))  
rep.Representation= "Surface"
```

```
[ ]: from ipyparaview.widgets import PVDDisplay  
disp = PVDDisplay(GetActiveView())  
w = display(disp)
```

Raytracing.ipynb

```
cscs RayTracing Last Checkpoint: 2 minutes ago (unsaved changes) Logout Control Panel
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
renderView1.AdditionalLights = [light1, light2]
renderView1.OSPRayMaterialLibrary = materialLibrary1

In [6]: renderView1.ViewSize = [800, 600]

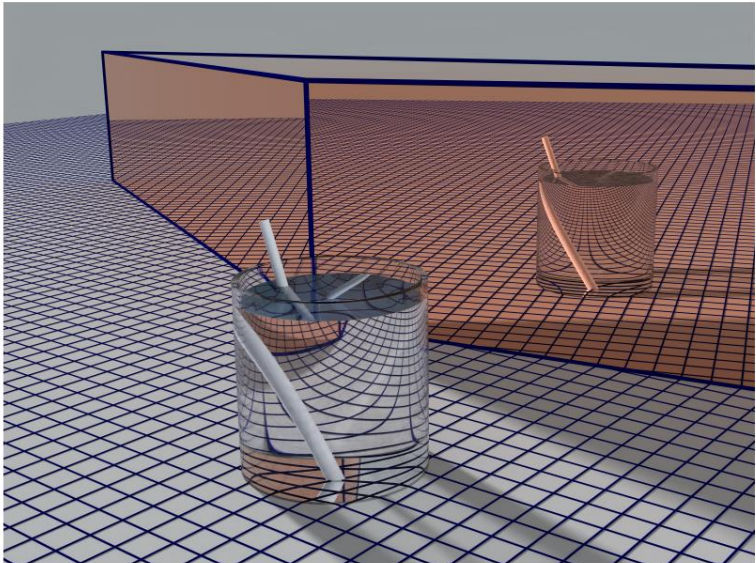
In [7]: # import the PVDisplay widget, then instantiate one for renV
from ipyparaview.widgets import PVDisplay
disp = PVDisplay(renderView1)

# show the widget once
display(disp)

In [8]: from paraview.modules.vtkPVClientServerCoreRendering import vtkPVOpenGLInformation

info = vtkPVOpenGLInformation()
info.CopyFromObject(None)
print("Vendor: %s" % info.GetVendor())
print("Version: %s" % info.GetVersion())
print("Renderer: %s" % info.GetRenderer())

Vendor: NVIDIA Corporation
Version: 4.6.0 NVIDIA 418.39
Renderer: Tesla P100-PCIE-16GB/PCIe/SSE2
```



- Ray-tracing is executed on the GPU
renderView1.BackEnd = 'OptiX pathtracer'
- Or runs on all available CPU threads
renderView1.BackEnd = 'OSPRay raycaster'
renderView1.BackEnd = 'OSPRay pathtracer'

updates

- ParaView v5.8 will become the default at the next maintenance
 - Move to 5.8 ASAP. It's much better anyway
- A 3-day Visualization Class [Instructor: Jean M. Favre]
 - Introductory + Advanced Visualization
 - ParaView in Jupyter Lab
 - Topics of interest (please send me emails)
 - October 5-7, 2020. Mark your calendar.

Your wish list?

What do you wish to have to improve your experience with ParaView (or 3D visualization) at CSCS?

Send me direct email jfavre@cscs.ch to discuss it further.



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

