

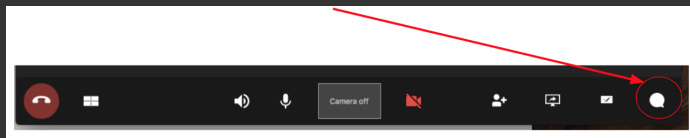
Photorealistic rendering with ParaView and OSPRay

ALEX RAZOUMOV
alex.razoumov@westgrid.ca



To ask questions

- Websteam: email **info@westgrid.ca**
- Vidyo: use the GROUP CHAT to ask questions



- Please mute your microphone unless you have a question
- Feel free to ask questions via audio at any time

This presentation is geared at **scientific visualization** practitioners who would like to introduce some elements of realism into their 3D renderings

Scientific Visualization Application (ParaView, VisIt, VMD) + VTK

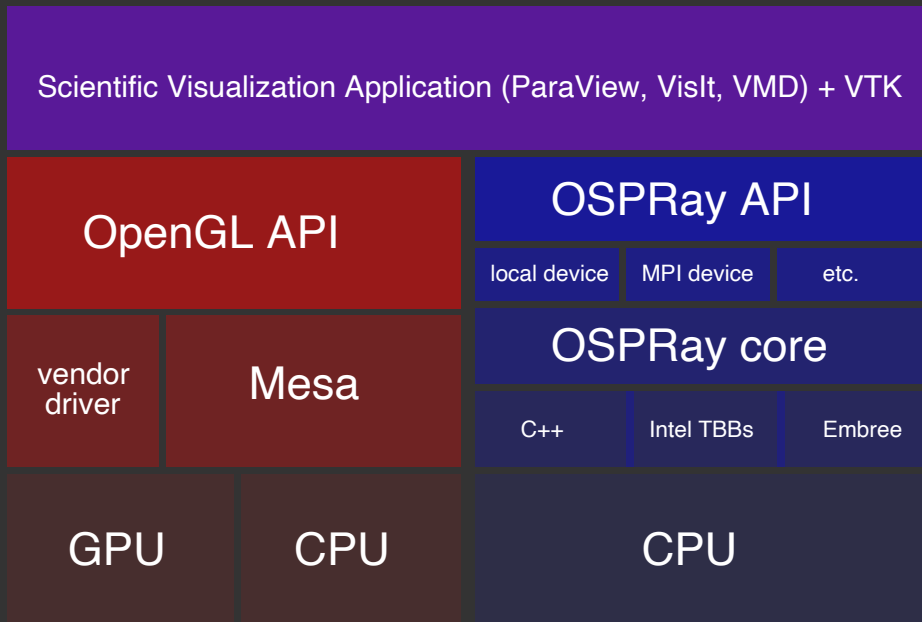
OpenGL API

vendor
driver

Mesa

GPU

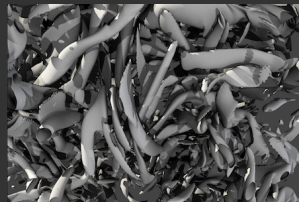
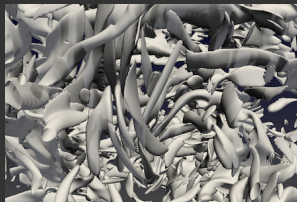
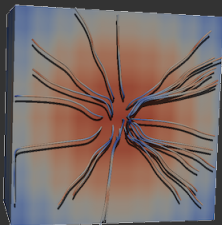
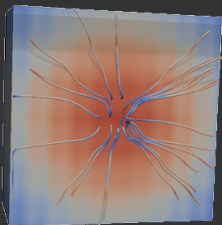
CPU



OSPRay

- Fast, open-source, end-user ray tracing library from Intel
- Same spirit as OpenGL (both surfaces and volumes), but different API
- Built on top of Intel's *Embree* low-level ray-tracing kernels (surface geometry rendering)
 - ▶ provides $\sim 1.5 - 6X$ speedup compared to earlier software renderers
 - ▶ small memory footprint: $10^{8.5}$ triangles, $10^{9.5}$ particles on a single workstation
- Can work with non-polygonal surfaces: cones, spheres, streamlines, cylinders
- It's ray tracing \Rightarrow provides ambient occlusion and shading
- *ParaView* (since 5.1), *VisIt*, *VMD* already use OSPRay

-
- Webinar "CPU-based rendering with OSPRay" in September 2016
<https://westgrid.github.io/trainingMaterials/tools/visualization>



OSPRay webinar conclusions in 2016:

- OSPRay + shadows was very good to highlight three-dimensionality and **depth effect in intricate scenes**
- OSPRay was much faster than Mesa-llvm, almost as fast as OpenGL ⇒ **great for GPU-less systems**
- OSPRay worked for both surface and volume rendering
- MPI backend did not work ...

Fast forward to 2020:

- MPI backend works well
- client-server OSPRay rendering works well
- choice of renderers!

OSPRay in ParaView

In ParaView 5.7 in Ray Traced Rendering → Back End you will find two OSPRay options:

1. **OSPRay raycaster** (earlier called SciVis renderer): fast ray tracer for scientific visualization with surface/volume rendering, ambient occlusion, light bouncing of hard surfaces, optional hard shadows, can't use materials
2. **OSPRay pathtracer**: supports soft shadows, indirect illumination (bounced light from other surfaces), realistic materials
3. if compiled into ParaView and NVIDIA GPU(s) present ⇒ CUDA-based OptiX pathtracer (closed source)

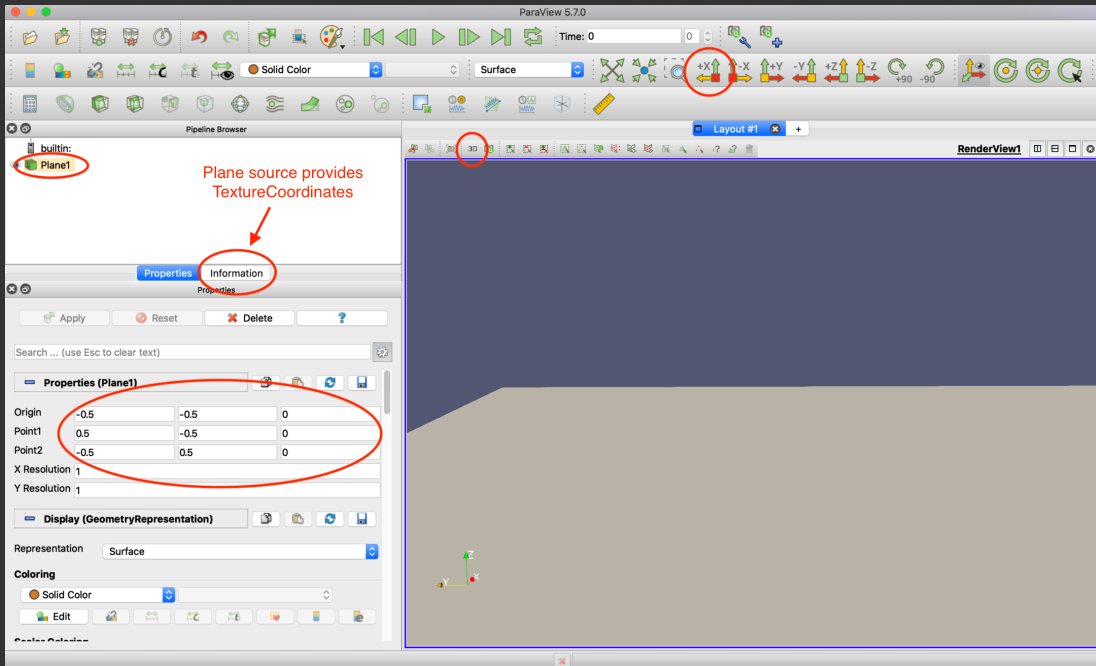
This presentation was inspired by December 2018 blog article “[Virtual tour and high-quality visualization with ParaView 5.6 + OSPRay](#)”

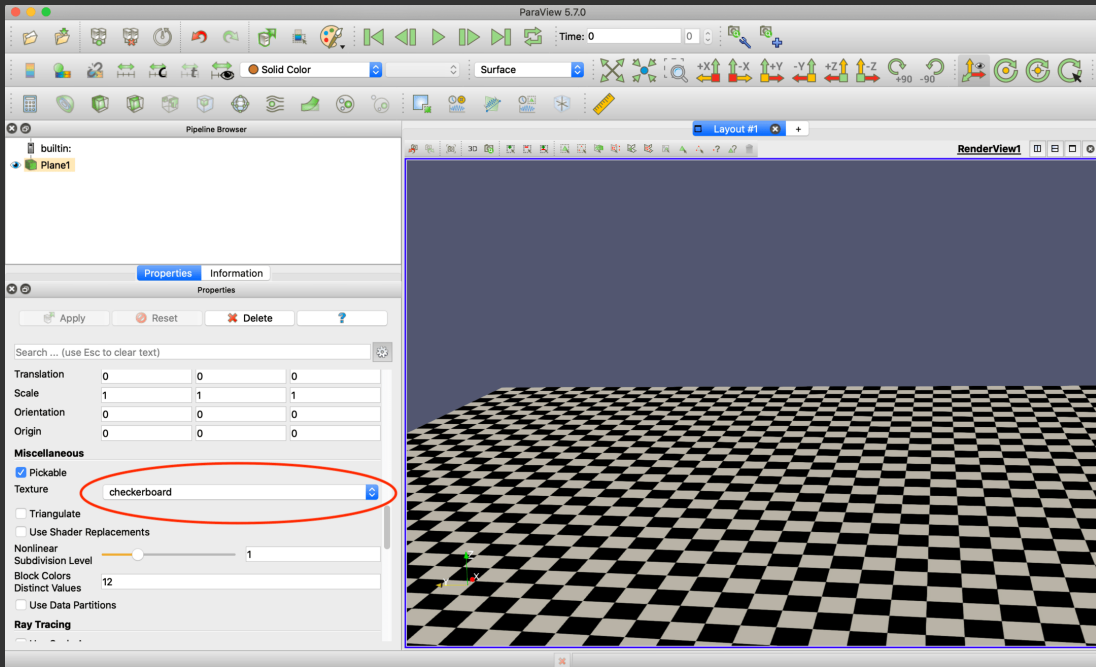
- Virtual visit of a power plant that mixes high-quality rendering and scientific visualization
- Their 1080p/30fps video contains 600 frames rendered with 300 samples per pixel
- Took ~35 hours to render on a Google Cloud compute node with 72 vCPUs

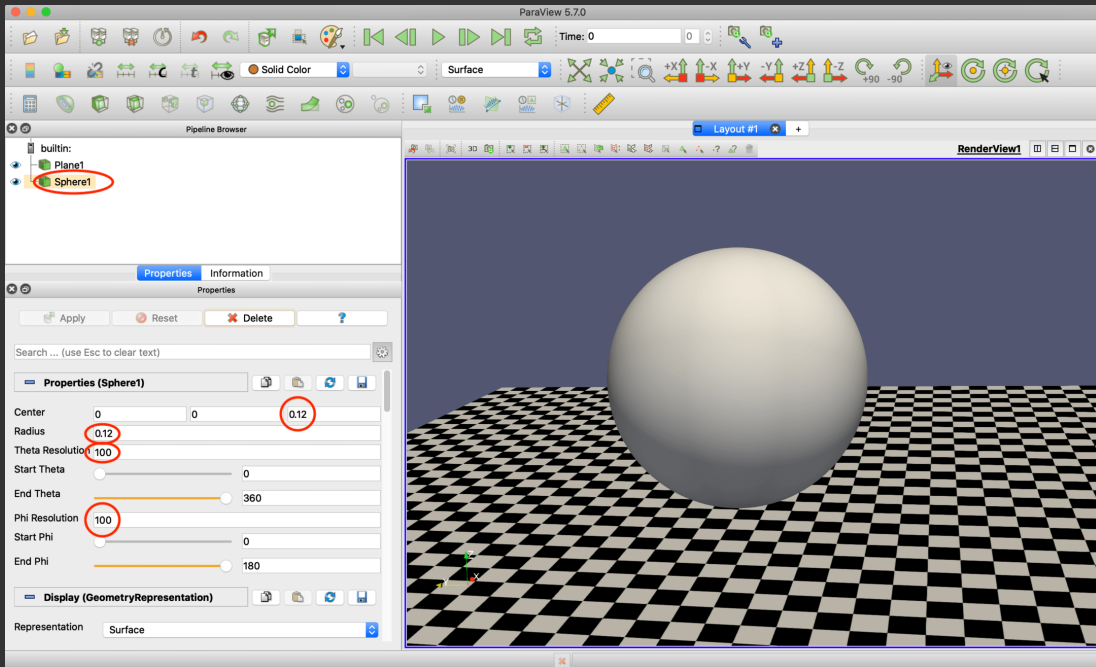


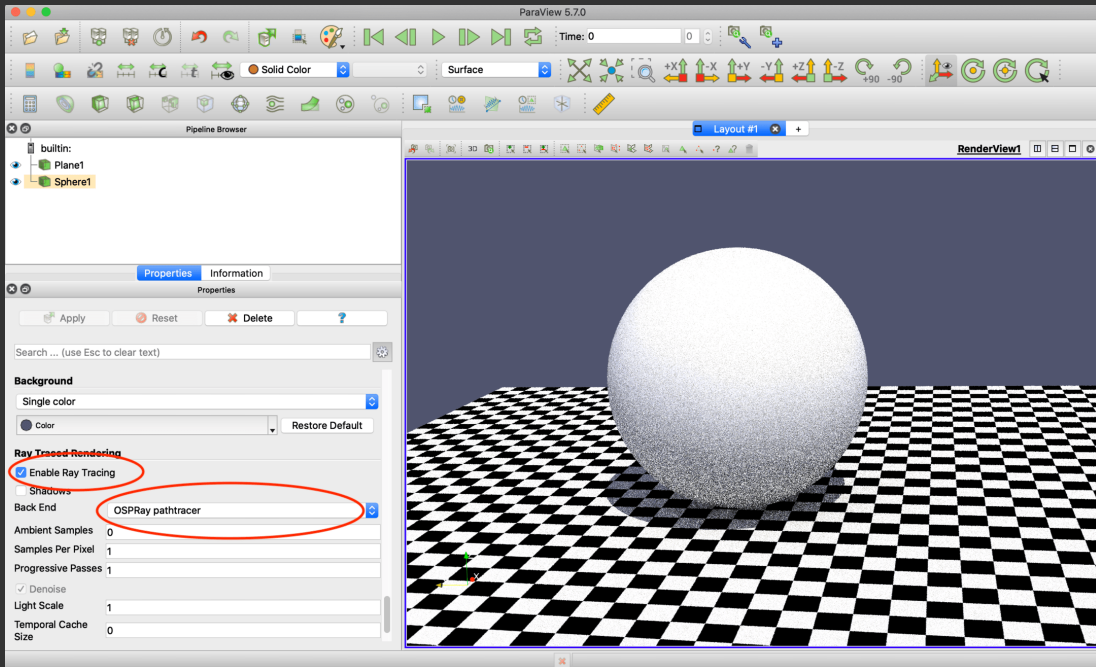
Detailed instructions for this setup and the power plant dataset (single step only) link can be found in the tutorial presentation “[ParaView and OSPRay Walk-through](#)” by Aaron Knoll and Bruce Cherniak (Intel)

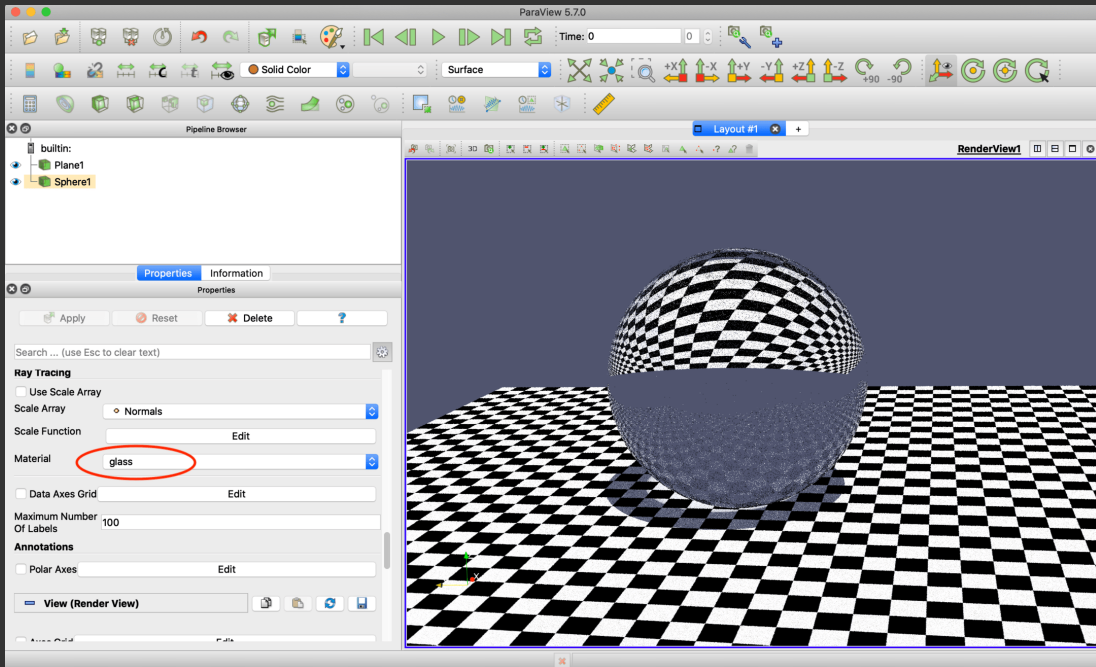
Let's start with a much simpler example
creating a simple object and assigning it an OSPRay material

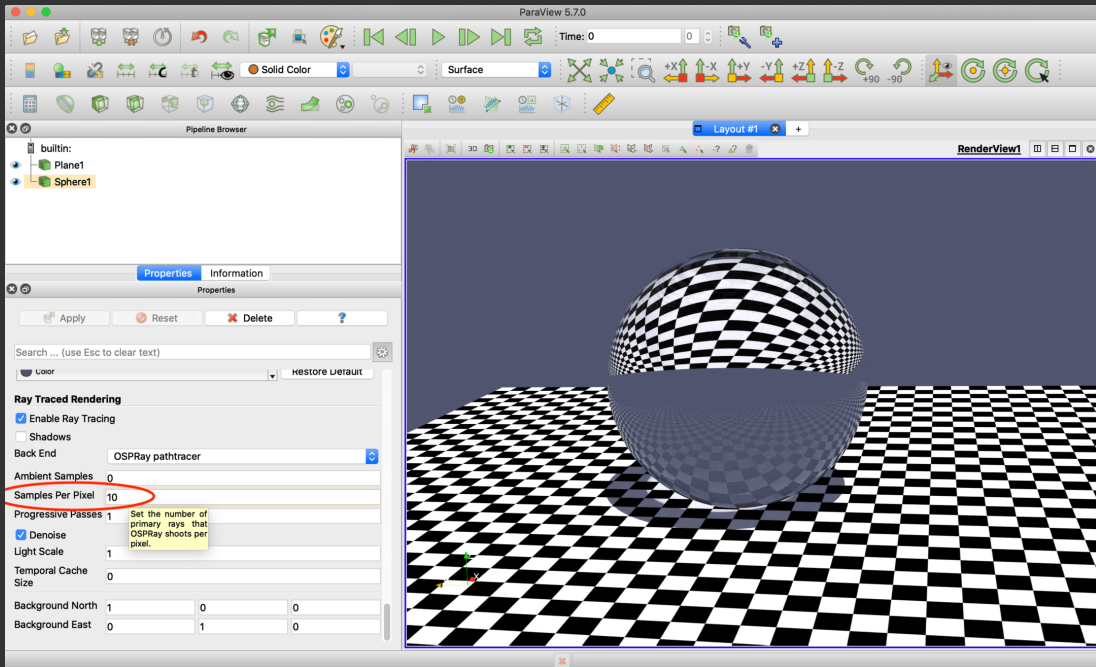




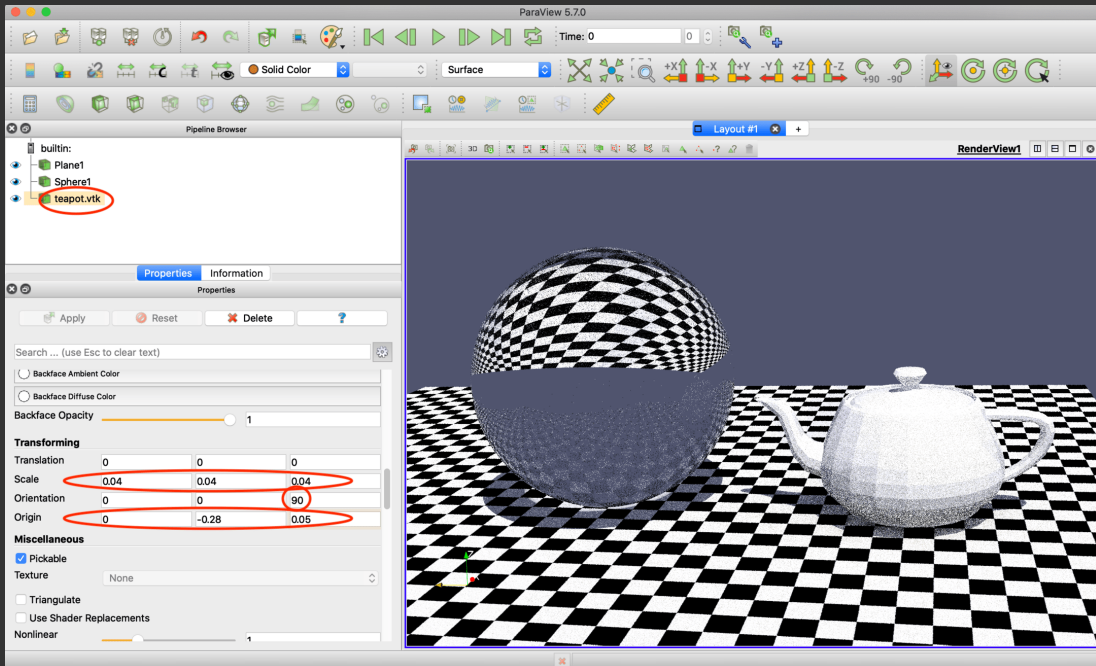


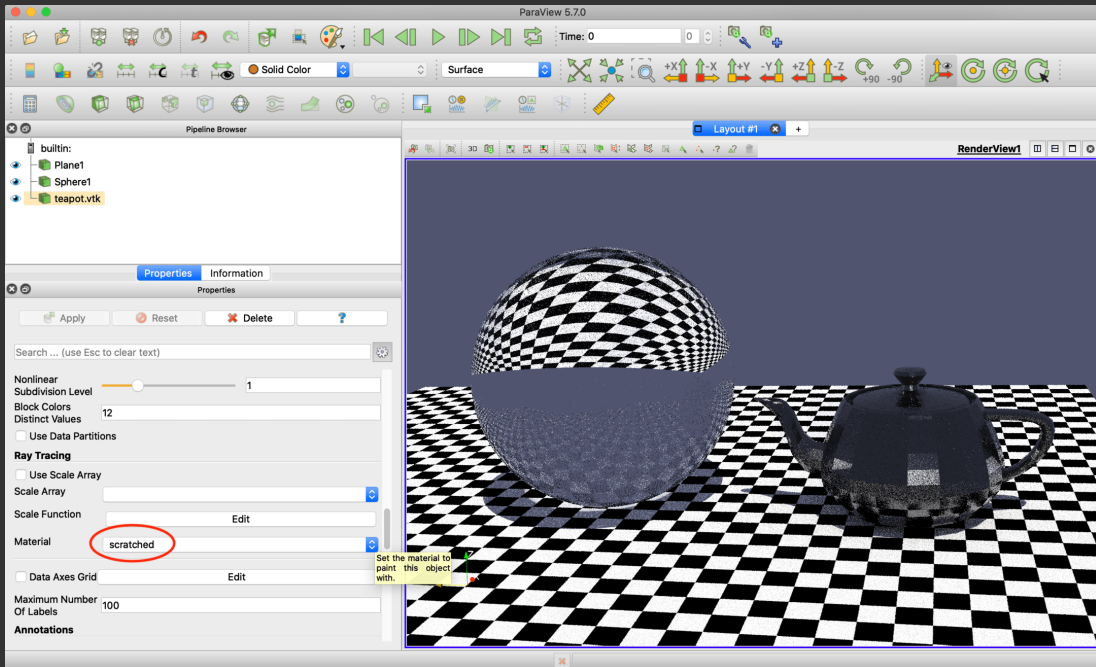


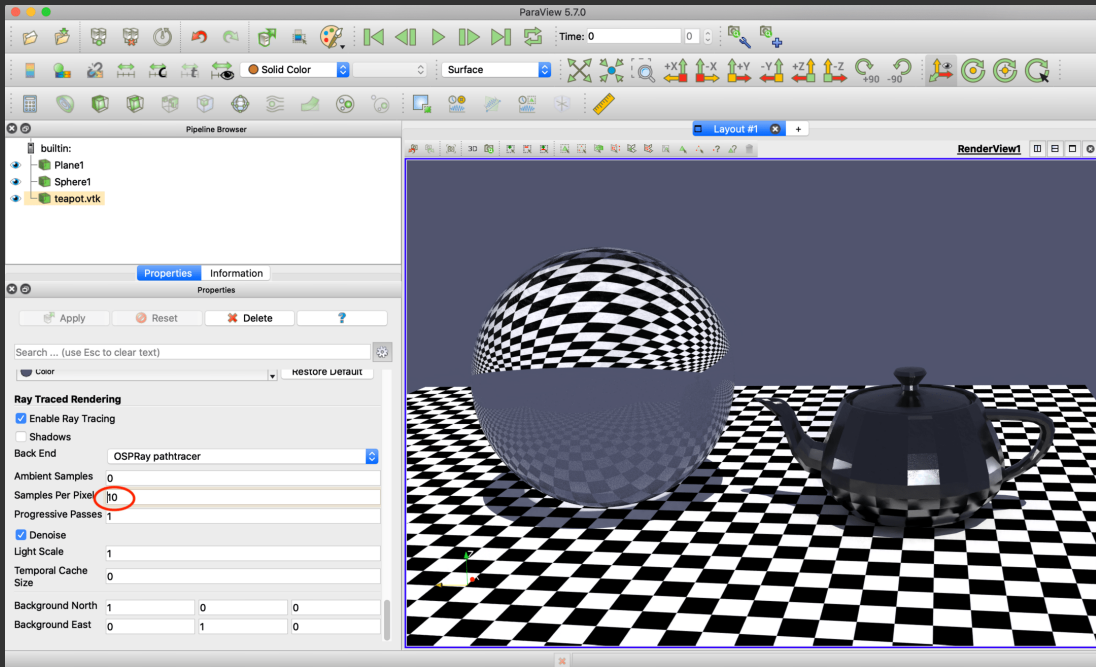




Let's add a second object
from a VTK data file







Live demo

```
$ paraview --state=checkSphereTeapot.pvsm # on presenter's laptop only
```

1. Adjust Samples Per Pixel: really meant for the production rendering (GUI and scripting)
2. Turn off ray tracing temporarily for interactive manipulation (switch back to familiar OpenGL rasterization)
 - ▶ very useful for any scene adjustment
 - ▶ crucial for setting and testing keyframe animation in the GUI

```
$ paraview --state=checkSphereTeapot.pvsm --enable-streaming
```

3. Adjust Progressive Passes: helps with interactive visualization with ray tracing ON
 - ▶ fast, low-pass rendering while interacting with the scene
 - ▶ slow, high-quality rendering when not interacting

OSPRay materials

1. Precompiled ParaView contains a set of default materials, e.g.

/Applications/ParaView-5.7.0.app/Contents/materials/ospray_mats.json (Mac) or
ParaView-5.7.0-...-64bit/share/paraview-5.7/materials/ospray_mats.json (Linux)

2. You can load 3rd-party materials File → Load Path Tracer Materials...

▶ e.g. check this [large OSPRay material database](#) from Marston “mconti”

3. Or you can create your own materials

-
- OSPRay materials reference <http://bit.ly/2FGcckW> on their GitHub page
 - OSPRay documentation <https://www.ospray.org/documentation.html>
 - ▶ most things (but not everything!) are enabled in ParaView’s OSPRay
 - Another good resource <https://discourse.paraview.org/search?q=materials>

OSPRay materials (cont.)

To test different materials, I wrote `testMaterials.py`:

- unfortunately, I can't find PV's Python equivalent of `File` → `Load Path Tracer Materials...` (Trace returns nothing)
- simple hack: add your materials to ParaView's default material database (will be loaded every time)

```
from paraview.simple import *
import paraview.servermanager as sm

renderView1 = GetActiveViewOrCreate('RenderView')
renderView1.ViewSize = [900, 800]
renderView1.InteractionMode = '3D'
renderView1.EnableRayTracing = 1
renderView1.BackEnd = 'OSPRay pathtracer'
renderView1.SamplesPerPixel = 5 # 10-30 for high quality
renderView1.CameraPosition = [-0.74, -0.14, 0.53]
renderView1.CameraFocalPoint = [1.62, 0.3, -0.82]
renderView1.CameraViewUp = [0.49, 0.036, 0.87]
renderView1.CameraParallelScale = 0.71

texture = sm.rendering.ImageTexture()
texture.FileName = sys.argv[1] # plane texture

planel = Plane()
```

```
planelDisplay = Show(planel, renderView1)
planelDisplay.Representation = 'Surface'
planelDisplay.Texture = texture
planelDisplay.OSPRayMaterial = 'None'

sphere1 = Sphere()
sphere1.Center = [0.0, 0.0, 0.15]
sphere1.Radius = 0.15
sphere1.ThetaResolution = 30
sphere1.PhiResolution = 30

sphere1Display = Show(sphere1, renderView1)
sphere1Display.Texture = None

sphere1Display = Show(sphere1, renderView1)
sphere1Display.Representation = 'Surface'
sphere1Display.OSPRayMaterial = 'greenGlass'

SaveScreenshot(sys.argv[2], renderView1,
               ImageResolution=[1800, 1600])
```

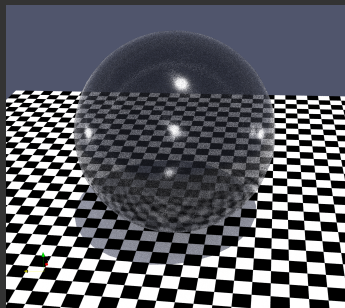
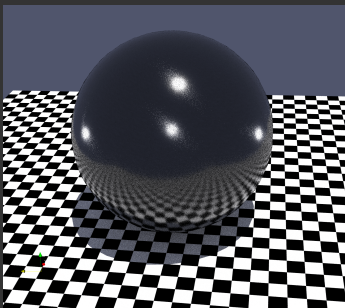
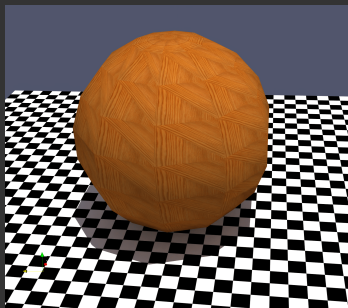
1. edit `/path/to/ospray_mats.json`
2. run: `pvbatch testMaterials.py checkerboard.jpg sphere.png && open sphere.png`

OSPRay materials (cont.)

- **OBJ Material:** workhorse material
- **Principled:** most complex material, capable of reproducing many different physical surfaces
- **CarPaint:** specialized version of Principled for rendering different types of car paints
- **Metal:** physical metal
- **Alloy:** similar to Metal, but with more intuitive colour control
- **Glass**
- **ThinGlass:** primarily for windows
- **MetallicPaint:** with a base coat with optional flakes

- Most of these material types can be fully customized via parameters
- Many of these types allow textures

OSPRay materials (cont.)

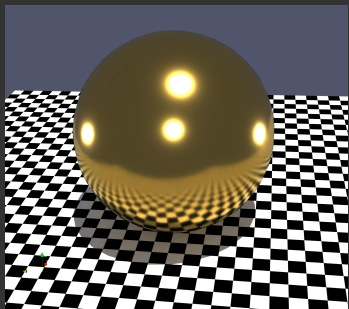


```
"wood": {
  "type": "OBJMaterial",
  "textures": {
    "map_kd": "wood.jpg"
  }
}
```

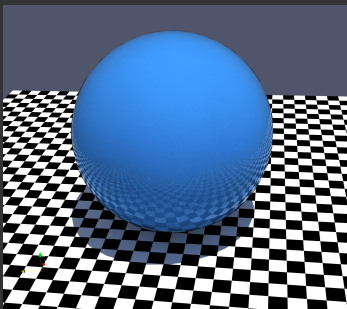
```
"scratched": {
  "type": "Principled",
  "doubles": {
    "metallic": [1.0]
  },
  "textures": {
    "normalMap": "metal_Normal.jpg",
    "baseColorMap": "metal_Base_Color.jpg",
    "roughnessMap": "metal_Roughness.jpg"
  }
}
```

```
"thinMetal": {
  "type": "Principled",
  "doubles": {
    "metallic": [1.0],
    "opacity": [0.5],
    "sheen": [0.8]
  },
  "textures": {
    "normalMap": "metal_Normal.jpg",
    "baseColorMap": "metal_Base_Color.jpg",
    "roughnessMap": "metal_Roughness.jpg"
  }
}
```

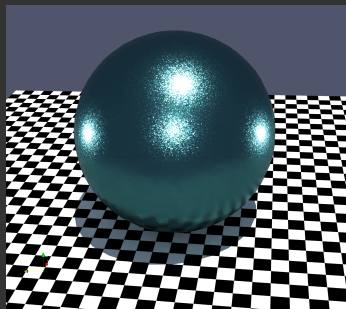
OSPRay materials (cont.)



```
"gold": {
  "type": "Metal",
  "doubles": {
    "eta": [0.07, 0.37, 1.5],
    "k": [3.7, 2.3, 1.7],
    "roughness": [0.2]
  }
}
```

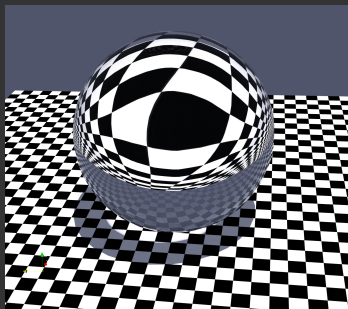


```
"carPaint1": {
  "type": "CarPaint",
  "doubles": {
    "baseColor": [0.2, 0.5, 0.8]
  }
}
```

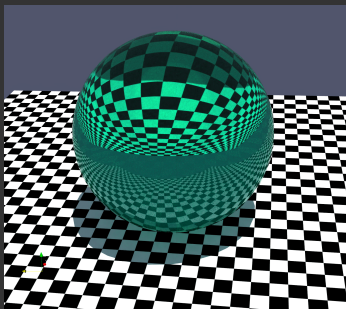


```
"carPaint2": {
  "type": "CarPaint",
  "doubles": {
    "coatColor": [0.3, 0.6, 0.6],
    "coat": [1],
    "coatRoughness": [0.3],
    "coatThickness": [0.5],
    "flakeDensity": [1],
    "flakeScale": [1000]
  }
}
```

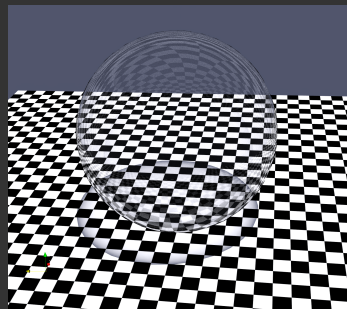
OSPRay materials (cont.)



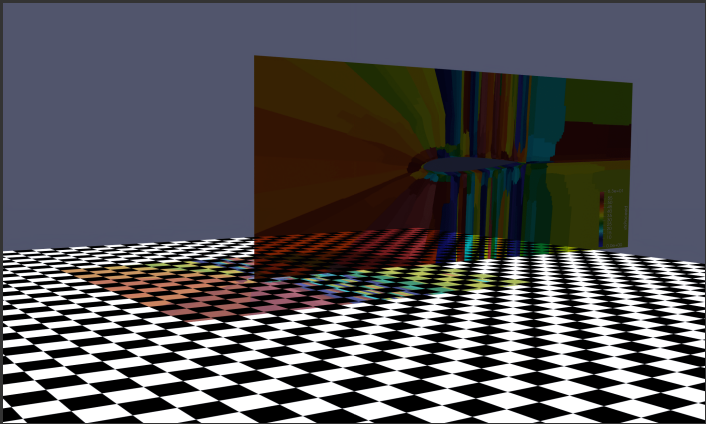
```
"glass": {
  "type": "Glass"
}
```



```
"greenGlass": {
  "type": "Glass",
  "doubles": {
    "attenuationColor": [0, 1, 0.3],
    "eta": [1.5]
  }
}
```



```
"thin glass": {
  "type": "ThinGlass"
}
```



```
"window": {
  "type": "ThinGlass",
  "textures": {
    "map_attenuationColor": "processorID.png"
  }
}
```

```
$ paraview --state=stainedGlass.pvsm # on presenter's laptop only
```

1. Add PNG texture to the thin glass material definition
2. Copy stainedGlass.png to /Applications/ParaView-5.7.0.app/Contents/materials/
3. Set custom directional lighting

Encoding materials with a multi-value scalar field

- So far, each of our pipeline objects was assigned a unique material
- You can also assign a unique material to each spatial region (a collection of data elements) with a constant data field
 - ▶ RegionId of a multi-block, e.g. check out this multiblock dataset example <https://discourse.paraview.org/t/parallel-and-material/3094/2>
 - ▶ load any complex scene from <https://sketchfab.com> (I'll show an example later)
 - ▶ here I'll use a small Unstructured Grid dataset with a multi-value scalar field saved as a VTK file
⇒ each value will be assigned its own material

createCells.py below writes a trio of 3D cells:

```
import pyevtk as pe, numpy as np
arr = np.array                                # function alias

x = arr([0.,0.2,0.2,0,0,0.2,0.2,0])          # define a cube
y = arr([0.,0,0.2,0.2,0,0,0.2,0.2])
z = arr([0.,0,0,0,0.2,0.2,0.2,0.2])

x = np.concatenate((x,arr([-0.1,0,0.2,0.1]))) # add first tetrahedron
y = np.concatenate((y,arr([-0.1,-0.3,-0.2,-0.2])))
z = np.concatenate((z,arr([0.,0.,0.,0.15])))

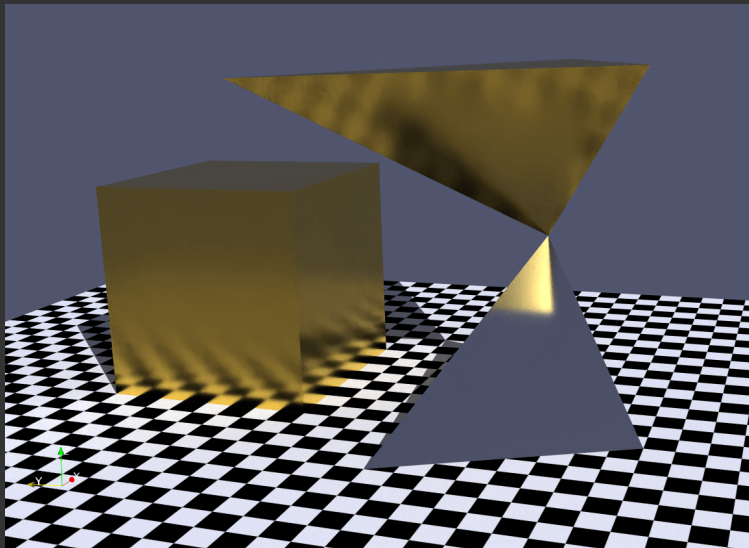
x = np.concatenate((x,arr([-0.1,0,0.2,0.1]))) # add second tetrahedron
y = np.concatenate((y,arr([0,-0.3,-0.2,-0.2])))
z = np.concatenate((z,arr([0.3,0.3,0.3,0.15])))

conns = np.arange(len(x)) # array of vertices
offs = arr([8,12,16])     # indices of the last vertex of each element in the connectivity array
ctypes = arr([pe.vtk.VtkHexahedron.tid, pe.vtk.VtkTetra.tid, pe.vtk.VtkTetra.tid])
cdata = {'density': arr([1., 2, 1])} # density at cell centres

filename = pe.hl.unstructuredGridToVTK('cells', x, y, z, connectivity=conns,
                                       offsets=offs, cell_types=ctypes, cellData=cdata)

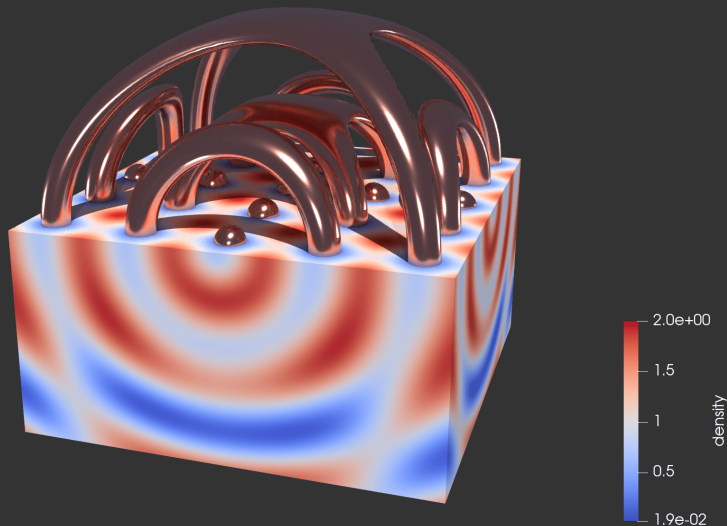
$ python createCells.py                        # this writes cells.vtu
$ paraview --state=cellsInitial.pvsm --enable-streaming # on presenter's laptop only
```

Encoding materials with a multi-value scalar field (cont.)



1. Colour by the cell array
2. Enable Ray Tracing, select OSPray Pathtracer
3. Choose raytracing material: Value Indexed
4. Edit Colour Map, check Interpret Values as categories
5. In the annotation, set:
value 1 is gold,
value 2 is aluminum

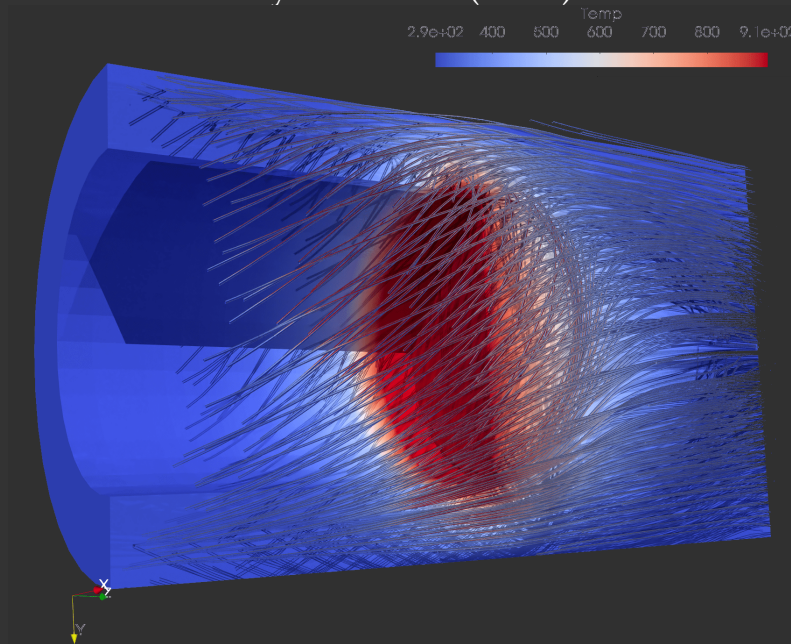
Mixing sci-vis with OSPRay materials



```
$ paraview --state=mixSciReal.pvsm  
$ open mixSciRealSpin.mp4
```

```
# on presenter's laptop only  
# on presenter's laptop only
```

Mixing sci-vis with OSPRay materials (cont.)



Complex scenes

1. Prepare your scene geometry and variables (> 95% of the work)
 - ▶ numerical simulation?
 - ▶ AutoCAD or a 3D paint tool or some 3D mesh generator?
 - ▶ 3D scan or photogrammetry?
2. Define your materials
3. Define your lights and test your shadows
 - ▶ View → Light Inspector
4. Only at the end you go into the path tracing mode and render

Complex scenes demo

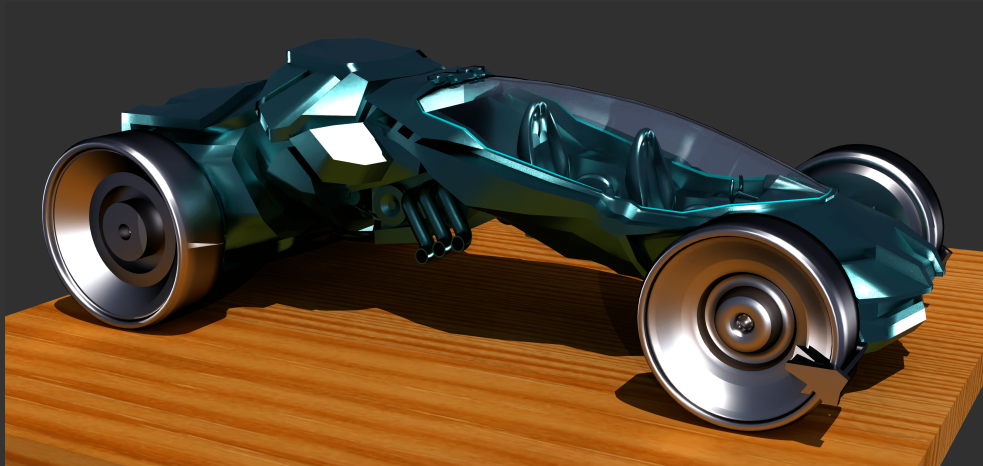
- Visit <https://sketchfab.com> (popular commercial hub for 3D models):
- Download free 3D model **Future Car** by *3DHaupt* <http://bit.ly/2T2szAc>
- Created in Blender's native 3D scene description file format
- Downloaded as glTF (new advanced file format for 3D scenes and models using JSON)
- Under Creative Commons license

On presenter's laptop only:

1. Open ParaView 5.7
2. Tools → Manage Plugins... → Load GLTFReader
3. File → Load State... → `car.pvsm`
4. Colour with `vtkBlockColors`
5. Turn off ray tracing for interactivity

```
"carPaint": {
  "type": "CarPaint",
  "doubles": {
    "coatColor": [0.3, 0.6, 0.6],
    "coat": [1],
    "coatRoughness": [0.3],
    "coatThickness": [0.5],
    "flakeDensity": [1],
    "flakeScale": [1000]
  }
}

"OBJ_black_bright": {
  "type": "OBJMaterial",
  "doubles": {
    "kd": [0.00, 0.00, 0.00],
    "ks": [0.90, 0.90, 0.90],
    "d": [1.00]
  }
}
```



- Watch `carSpin.mp4` on presenter's laptop
 - ▶ 3624x2552 image with 30 samples/pixel took $\sim 4^m$ on my laptop (four cores)
 - ▶ 300 frames at 1812x1276 with 5 samples/pixel took $\sim 1^h19^m$ on my laptop (four cores)
- Turn off ray tracing when setting keyframe animation: make sure everything works

Parallel rendering on Compute Canada clusters

- On Compute Canada clusters, regular ParaView modules do not have OSPRay: requires a rather custom installation procedure
- On Cedar, I compiled ParaView server 5.7 with OSPRay + materials in `/home/razoumov/paraviewcpu571`
 - ▶ materials in `share/paraview-5.7/materials`
- `serial.sh` job submission script:

```
#!/bin/bash
#SBATCH --time=00:15:00    # walltime hh:mm:ss
#SBATCH --mem=3600         # in MB
#SBATCH --job-name="serial rendering"
/home/razoumov/paraviewcpu571/bin/pvbatch testMaterials.py checkerboard.jpg benchmark.png
```

- `parallel.sh` job submission script:

```
#!/bin/bash
#SBATCH --ntasks=16        # number of MPI processes
#SBATCH --time=0-00:15     # walltime d-hh:mm
#SBATCH --mem-per-cpu=3600 # in MB
#SBATCH --job-name="parallel rendering"
mpirun -np $SLURM_NTASKS /home/razoumov/paraviewcpu571/bin/pvbatch testMaterials.py \
checkerboard.jpg benchmark.png
```

Parallel rendering on Compute Canada clusters

Using `testMaterials.py` to generate 1800x1600 PNG on Cedar:

SamplesPerPixel	1	30	100
serial	00 ^m 26 ^s	02 ^m 03 ^s	06 ^m 05 ^s
4 cores		01 ^m 13 ^s	03 ^m 32 ^s
16 cores			01 ^m 23 ^s

Summary

- OSPRay materials reference <http://bit.ly/2FGcckW> on their GitHub page
- OSPRay documentation <https://www.ospray.org/documentation.html>
- ParaView Discourse
<https://discourse.paraview.org/search?q=materials>
- Watch “[ParaView Photorealistic Rendering for Data Visualization](#)” presentation by Jean M. Favre (Swiss National Supercomputing Centre) at
<https://www.youtube.com/watch?v=ifK3vXYJm6Y&t>
- Email me “alex.razoumov@westgrid.ca” if you need help

Questions?