Scientific Software Development with Python

Visualizing scientific data

CHALMERS
UNIVERSITY OF TECHNOLOGY

Simon Pfreundschuh
Department of Space, Earth and Environment

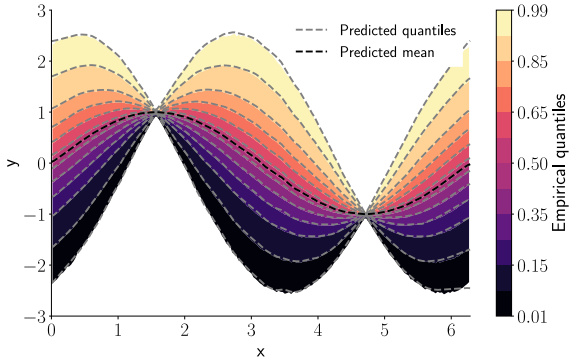
Agenda

1. Introduction

2. Visualizing 3D data with PyVista

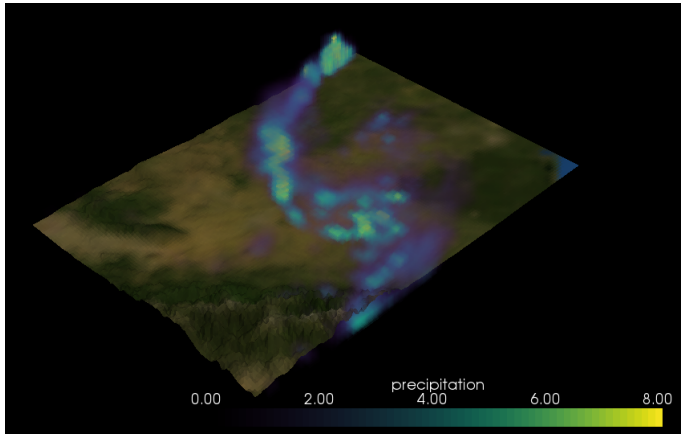
2D

- Matplotlib
  - Arguably one of the most popular scientific Python packages
  - Limited 3D capability
- plotly, bokeh
  - Web-based, interactive plotting



3D

- VTK
  - Powerful but complex
- MayaVi, PyVista
  - High-level interfaces to VTK



Agenda

1. Introduction

2. Visualizing 3D data with PyVista

Primitives

- Complex visualizations are typically built from basic, visual primitives.
- In 3D we have the following primitives:
  - points
  - lines
  - surfaces
  - volumes

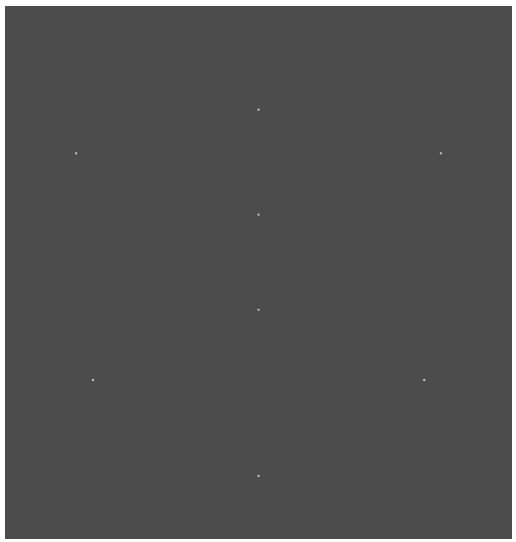
- `PolyData`: Represent surfaces points, lines and surfaces
- `UnstructuredGrid`:
  - Can represent surfaces and volumes
  - Connections between grid points must be added explicitly
  - Therefore seldomly used directly
- `StructuredGrid`:
  - Cells between neighboring points are implicitly assumed
  - Easiest way to represent gridded surfaces and volumes

Points

- The `PolyData` class allows us to build 3D shapes from these primitives
- To begin, we add a dataset with eight points.

Example

```
import numpy as np
import pyvista as pv

points = np.array([0.0, 1.0, 2.0])
x, y, z = np.meshgrid(points, points, points)
point_coords = np.hstack([x.reshape(-1, 1),
                          y.reshape(-1, 1),
                          z.reshape(-1, 1)])
# Dataset has eight points.
points = pv.PolyData(point_coords)
points.plot()
```



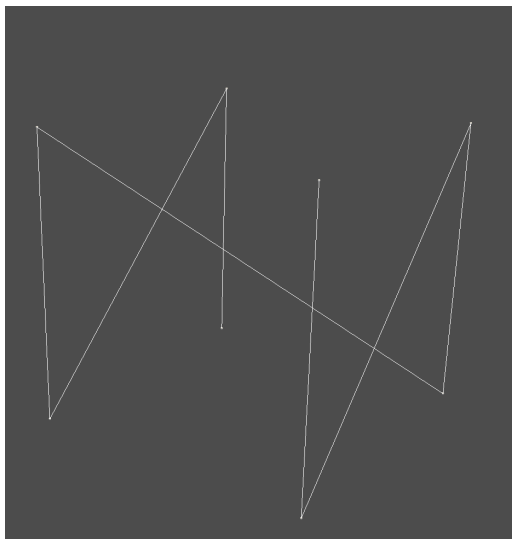
Adding lines

- Lines are described by arrays of the form:

```
[n, point_index_0, ..., point_index_n-1]
```

Example

```
# Line consists of 8 points.
lines = np.array([8] + list(range(8)))
points_and_lines = pv.PolyData(point_coords)
points_and_lines.lines = lines
```



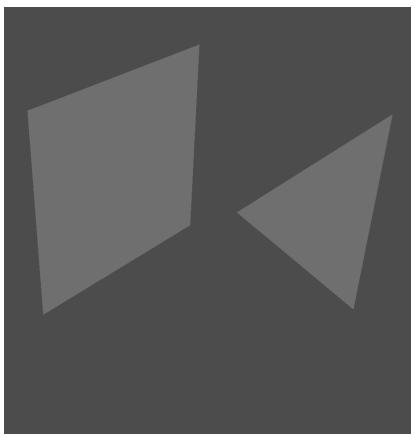
Adding faces

- Similar to lines, faces are described by arrays of the form:

```
[[n, point_index_0, ..., point_index_n-1], # First face
 [...],                                    # Second face
 ...
```

Example

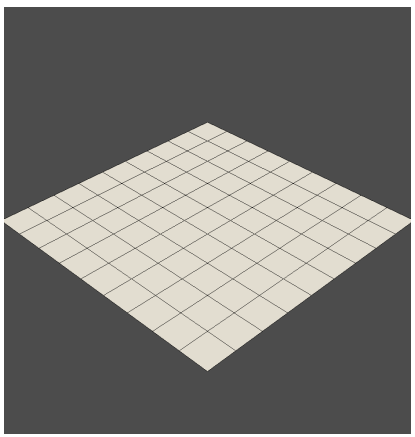
```
faces = np.hstack([[4, 0, 1, 3, 2], # A rectangle
                   [3, 4, 5, 7]],)   # A triangle
# Faces can be passed directly to the constructor.
points_with_faces = pv.PolyData(point_coords, faces)
```



- We could use `PolyData` to represent gridded surfaces, but that is typically too complicated.
- To create a gridded surface it is easier to use the `StructuredGrid` class:

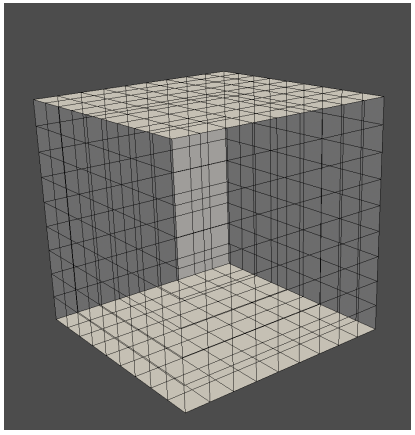
Example

```
x = np.arange(10)
y = np.arange(10)

x_coords, y_coords = np.meshgrid(x, y)
z_coords = np.zeros(x_coords.shape)
surface = pv.StructuredGrid(x_coords, # 10 x 10 array
                            y_coords, # 10 x 10 array
                            z_coords) # 10 x 10 array

plotter = pvqt.BackgroundPlotter()
plotter.add_mesh(surface, show_edges=True)
```



- Similarly, we can use the `StructuredGrid` to represent volumes:

```
x = np.arange(10)
y = np.arange(10)
z = np.arange(10)

x_coords, y_coords, z_coords = np.meshgrid(x, y, z)
volume = pv.StructuredGrid(x_coords, # 10 x 10 x 10 array
                           y_coords, # 10 x 10 x 10 array
                           z_coords) # 10 x 10 x 10 array
plotter = pvqt.BackgroundPlotter()
plotter.add_mesh(volume, show_edges=True)
```



Adding information

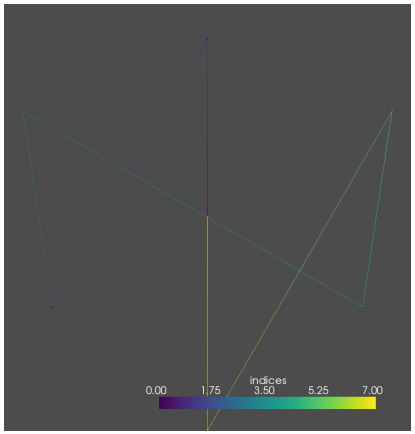
- So far we can represent geometries
- Next we will see how to associate these geometries with data

`PolyData`

- To display information we need to add data arrays to our mesh
- We can do this for each point

Example

```
points = pv.PolyData(point_coords)
points.point_arrays["indices"] = np.arange(8)
points.plot(scalars="indices")
```

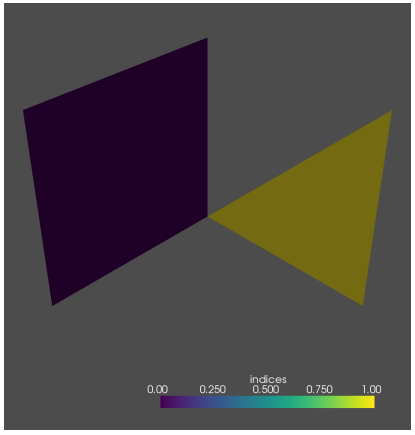


`PolyData`

- Or for each cell

Example

```
faces = np.hstack([[4, 0, 1, 3, 2],
                   [3, 4, 5, 7]],)
points_with_faces = pv.PolyData(point_coords, faces)
points_with_faces.cell_arrays["indices"] = [0, 1]

plotter = pvqt.BackgroundPlotter()
plotter.add_mesh(points_with_faces, scalars="indices")
```



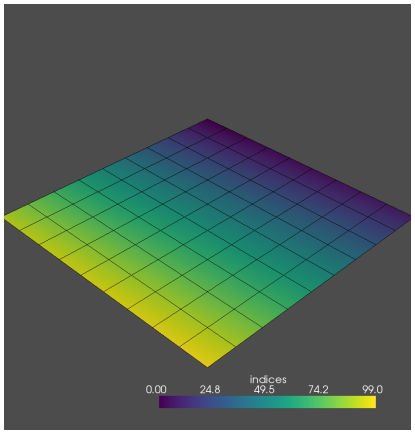
- We can use this to display surface elevation:

```
from typhon.topography import SRTM30
lats, lons, z = SRTM30.elevation(57.0, 11.5, 58.0, 12.5)
x, y = np.meshgrid(lats, lons)
surface = pv.StructuredGrid(x, y, z * 2e-4) # 10 x 10 x 10 array
surface.plot()
```

`StructuredGrid`

- Adding data to a structured grid works in the same way

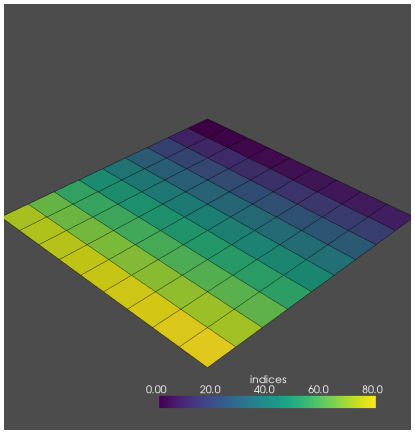
```
x = np.arange(10)
y = np.arange(10)

x_coords, y_coords = np.meshgrid(x, y)
z_coords = np.zeros(x_coords.shape)
surface = pv.StructuredGrid(x_coords, # 10 x 10 array
                            y_coords, # 10 x 10 array
                            z_coords) # 10 x 10 array

surface.cell_arrays["indices"] = np.arange(81)
plotter = pvqt.BackgroundPlotter()
plotter.add_mesh(surface, show_edges=True)
```



`StructuredGrid`

- Adding data to a structured grid works in the same way

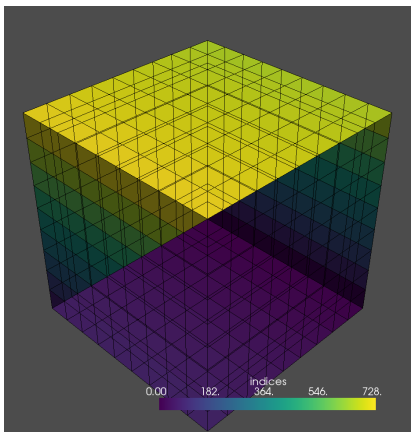
```
x = np.arange(10)
y = np.arange(10)

x_coords, y_coords = np.meshgrid(x, y)
z_coords = np.zeros(x_coords.shape)
surface = pv.StructuredGrid(x_coords, # 10 x 10 array
                            y_coords, # 10 x 10 array
                            z_coords) # 10 x 10 array

surface.cell_arrays["indices"] = np.arange(81)
plotter = pvqt.BackgroundPlotter()
plotter.add_mesh(surface, show_edges=True)
```



`StructuredGrid`

- Adding data to a 3D `StructuredGrid` works in the same way.

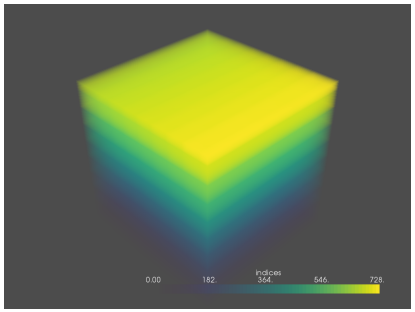
```
x = np.arange(10)
y = np.arange(10)
z = np.arange(10)

x_coords, y_coords, z_coords = np.meshgrid(x, y, z)
volume = pv.StructuredGrid(x_coords,  # 10 x 10 x 10 array
                           y_coords, # 10 x 10 x 10 array
                           z_coords) # 10 x 10 x 10 array

volume.point_arrays["indices"] = np.arange(10 * 10 * 10)
plotter = pvqt.BackgroundPlotter()
plotter.add_mesh(volume, show_edges=True)
```



Volume rendering

- Typically refers to displaying volumetric scalar data using transparency.
- In `pyvista` only works with a `UniformGrid`

```
volume = pv.UniformGrid((21, 21, 21), spacing=(1, 1, 1))
volume.cell_arrays["indices"] = np.arange(20 ** 3)
volume.plot(volume=True)
```



Exercise 1

- Exercise 1
- Time 30 minutes