



# 3D点云数据处理和可视化实践(入门)

## Lecture-pyvista篇

国新院实验实践课  
工程师通用技术科教平台



教师：欧阳真超 助教：xxx



邮箱：ouyangkid@buaa.edu.cn



学期：2025年春季

# 目录

## Contents

01 PyVista简介

02 课程内容

- **PyVista**: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK)
- 其他可视化库
  - Matplotlib (Hunter, 2007)
  - Mayavi (Ramachandran & Varoquaux, 2011)
  - yt Project (Turk et al., 2010)
  - Visualization Toolkit (VTK) (Schroeder, Lorensen, & Martin, 2006) # python 绑定了所有的C++接口
- 提供处理大规模、特定数据结构的数据集处理、友好API的开源工具。
- **Pyvista**致力于进一步简化标准网格创建与绘图例程，同时确保不牺牲基于 C++ 的 VTK 后端运行速度。
- 解决传统VTK的python binding流程复杂的问题。

结论 **既要VTK的功能  
又要简化调用实现流程**

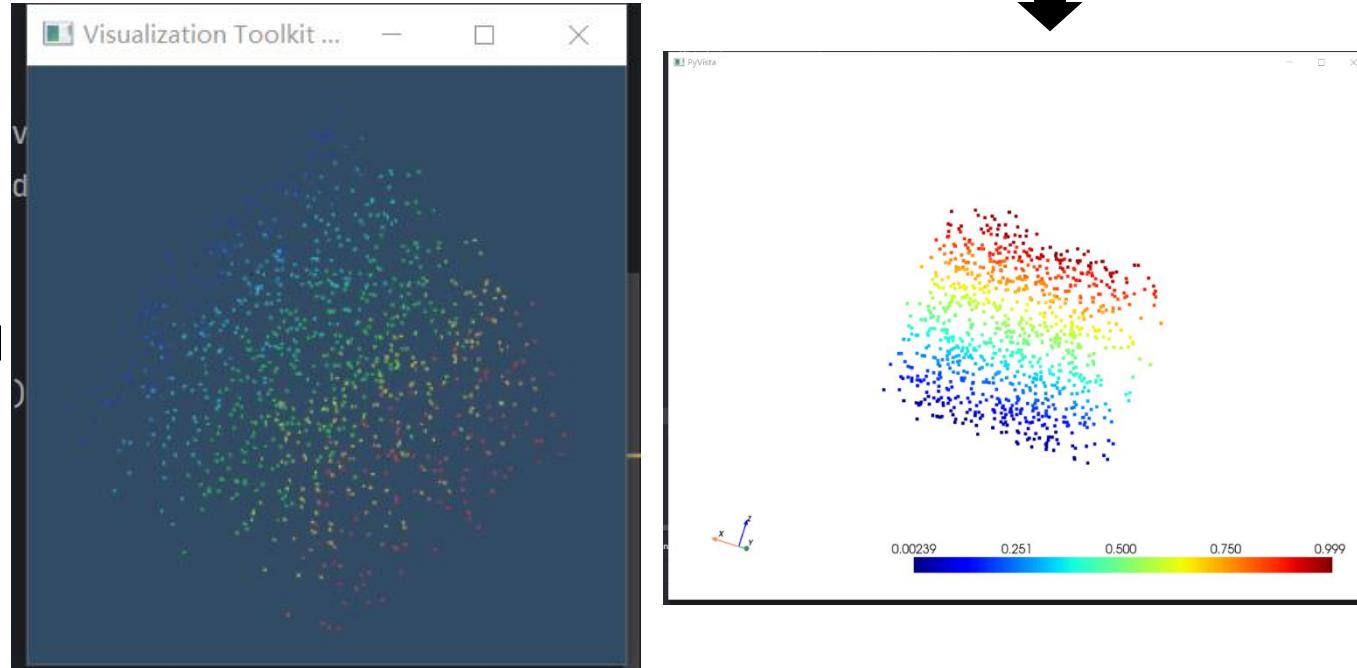
### VTK调用：初始化、添加点、清除点

```
1 import vtk
2 from numpy import random
3
4 # 解释 | 添加注释 | X
5 class VtkPointCloud:
6     # 解释 | 添加注释 | X
7     def __init__(self, zMin=-10.0, zMax=10.0, maxNumPoints=1000):
8         # 解释 | 添加注释 | X
9         def addPoint(self, point): ...
10        # 解释 | 添加注释 | X
11        def clearPoints(self): ...
12
13        pointCloud = VtkPointCloud()
14        for k in range(1000):
15            point = 20*(random.rand(3)-0.5)
16            pointCloud.addPoint(point)
17        pointCloud.addPoint([0,0,0])
18        pointCloud.addPoint([0,0,0])
19        pointCloud.addPoint([0,0,0])
20        pointCloud.addPoint([0,0,0])
21
22        # Renderer
23        renderer = vtk.vtkRenderer()
24        renderer.AddActor(pointCloud.vtkActor)
25        renderer.SetBackground(.2, .3, .4)
26        renderer.ResetCamera()
27
28        # Render Window
29        renderWindow = vtk.vtkRenderWindow()
```

### PyVista：直接使用numpy数据结构

```
# 着色点云可视化,
points = np.random.random((1000, 3))
pc = pv.PolyData(points)
pc.plot(scalars=points[:, 2], point_size=5.0, cmap='jet')
```

00\_vistest.py



- 1. 简洁易用的接口

➤ **简化 VTK 的复杂 API:** VTK 的原生 Python 接口直接绑定 C++ 调用，代码冗长且复杂。

PyVista 通过封装 VTK 的功能，提供了更简洁、直观的 API，降低了用户的学习门槛。

➤ **类似 Matplotlib 的语法:** PyVista 的绘图函数设计借鉴了 Matplotlib 的风格，支持直观的关键字参数，使得 3D 数据可视化更加容易上手。

- 2. 高效的原生 VTK 支持

➤ **基于 VTK 的强大后端:** PyVista 直接调用 VTK 的 C++ 后端，继承了其高性能和丰富的功能，同时通过 Python 接口实现了快速原型开发。

➤ **无缝集成 NumPy:** PyVista 的数据对象与 NumPy 数组兼容，支持直接访问和操作数据，方便科学计算和数据处理。

- 3. 丰富的内置功能

➤ **常用过滤算法:** PyVista 为数据集提供了大量内置的过滤方法（如裁剪、平滑、切片等），用户可以通过简单的函数调用实现复杂的网格操作。

➤ **支持多种数据格式:** 能够轻松读取和写入 VTK 支持的文件格式（如.vtk、.stl、.ply 等），简化了数据导入导出流程。

- 4. 面向对象的扩展设计

➤ **封装 VTK 数据类型:** PyVista 扩展了 VTK 的数据类型，为对象添加了便捷的方法和属性，例如快速访问标量数组、检查数据集属性等。

➤ **方法链式调用:** 支持链式调用过滤器和绘图方法，代码更简洁，逻辑更清晰。

- 5. 适用于大规模数据

- 高性能处理：得益于 VTK 的 C++ 后端，PyVista 能够高效处理大型空间数据集，适合科学计算和工程应用。
- 地理空间数据支持：特别适合处理地理科学数据（如地质模型、温度场等），支持复杂场景的集成渲染。

- 6. 社区与生态支持

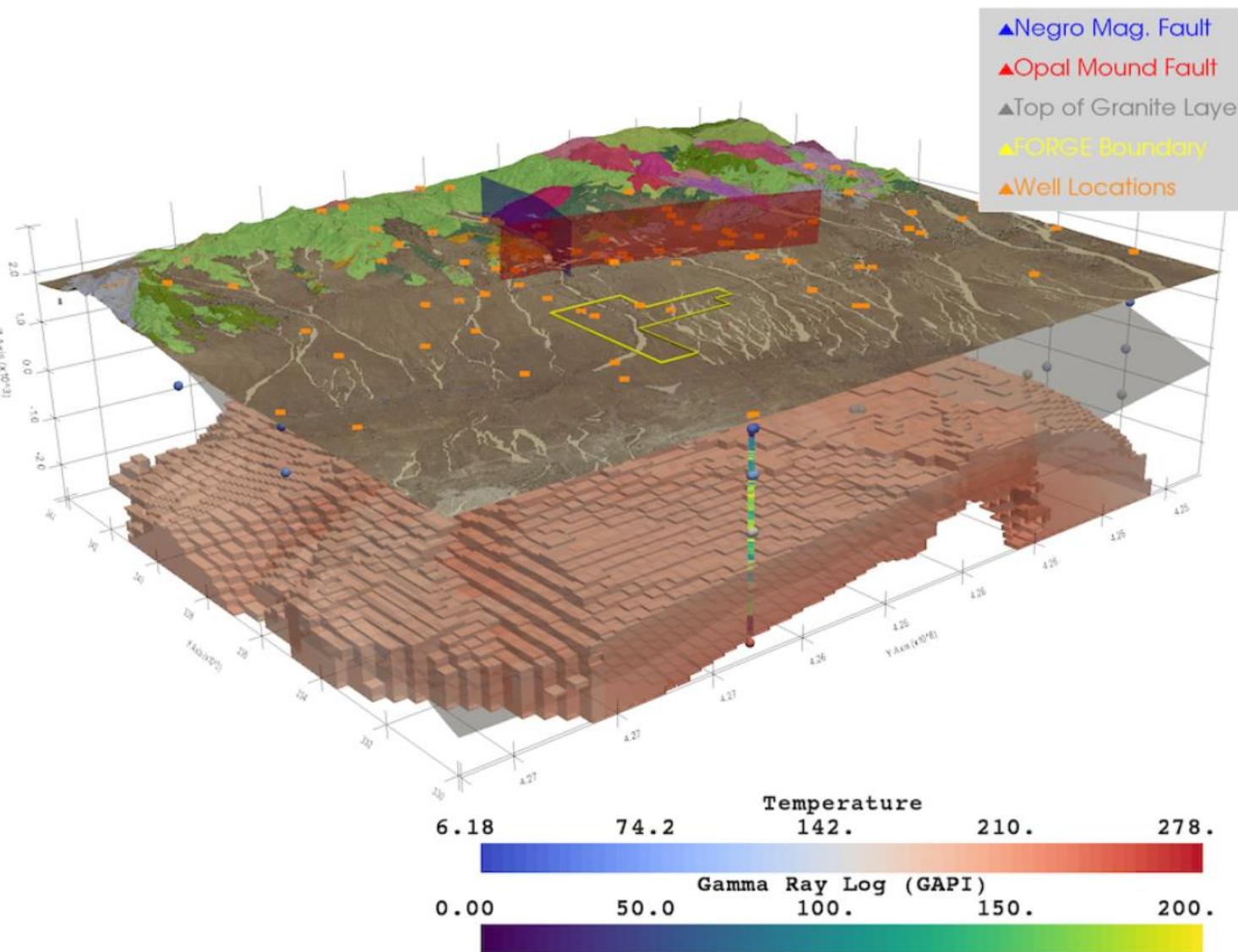
- 广泛的实际应用：被多个研究机构（如美国空军研究实验室 AFRL）用于科研可视化，并在多篇论文中作为主要工具。
- 与其他工具的集成：与 PVGeo 等库结合，进一步扩展了地质科学数据的分析和可视化能力。

- 7. 文档与可复现性

- 完善的文档：提供详细的教程和示例，帮助用户快速掌握功能。
- 促进科研复现：通过简洁的代码实现复杂的可视化，便于研究成果的分享和复现。

- 8. 开源与许可

- 开源免费：遵循 CC-BY 4.0 许可，用户可以自由使用、修改和分发。
- 活跃的社区：代码托管在 GitHub，用户可以通过社区贡献和反馈不断改进功能。



### PyVista的应用示例

美国能源部：空间地理数据的可视化

多层次三维渲染：

1. 数字陆地表面，包含覆盖的卫星图像、地质图；
2. 地下温度模型；
3. 体素化展示的 采样散点温度值（灰）
4. 地球物理学日志数据
5. 基于GIS的围栏边界划分
6. 可解释的断层信息

[在线demo \(需翻墙\)](#)

<https://gdr.openei.org/forge>

物理世界的三维属性，加上数据的立体结构复杂性，需要我们通过三维可视化进行展示。从模拟人眼视觉感知延伸到其它非可见信息的**具象化**，有助于我们进行理解和分析。

# 目录

## Contents

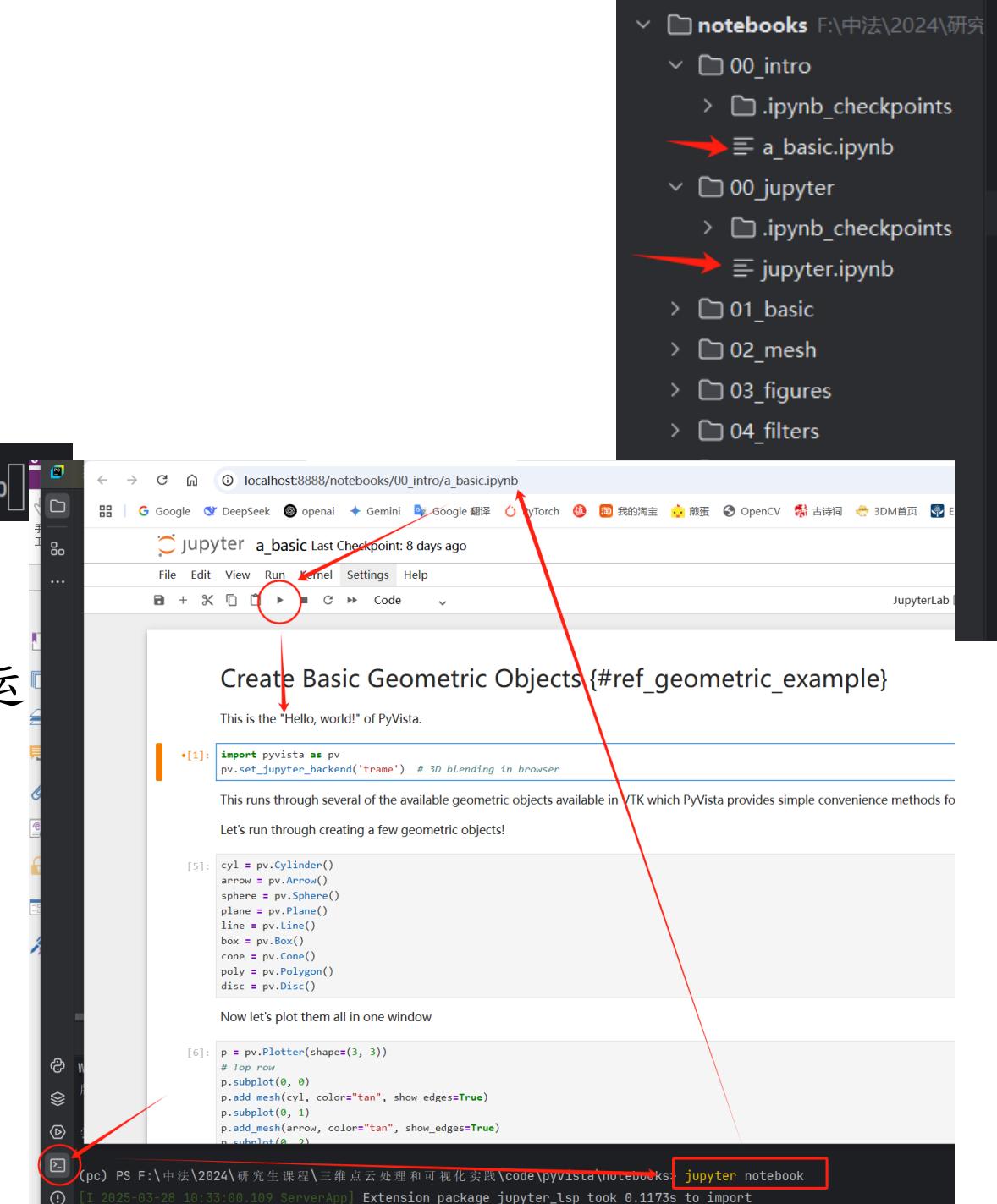
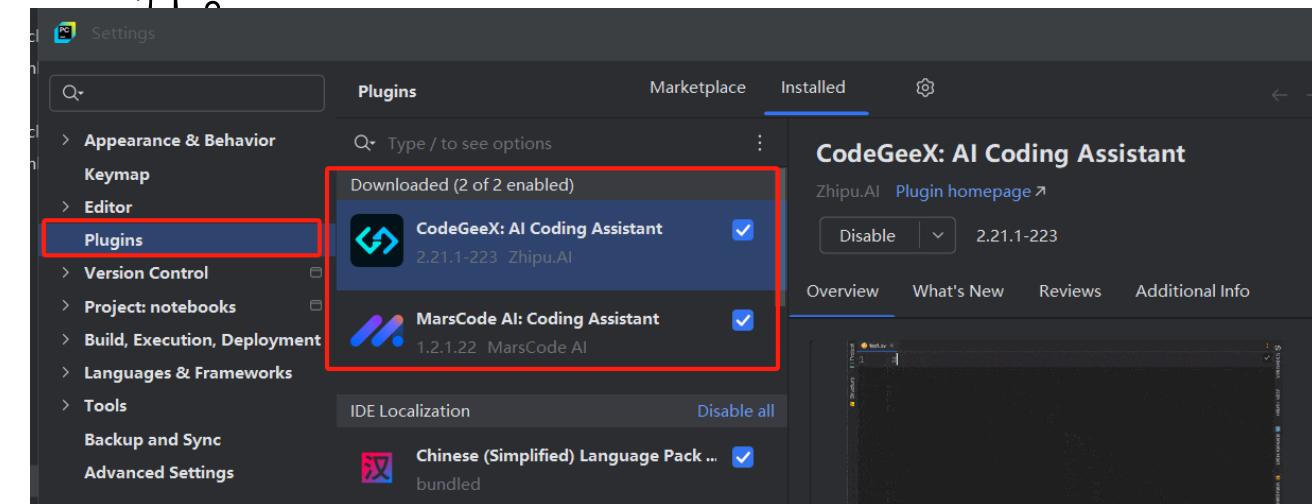
**01** PyVista简介

**02** 课程内容

# Part 2 | 课程内容

## » 操作指引

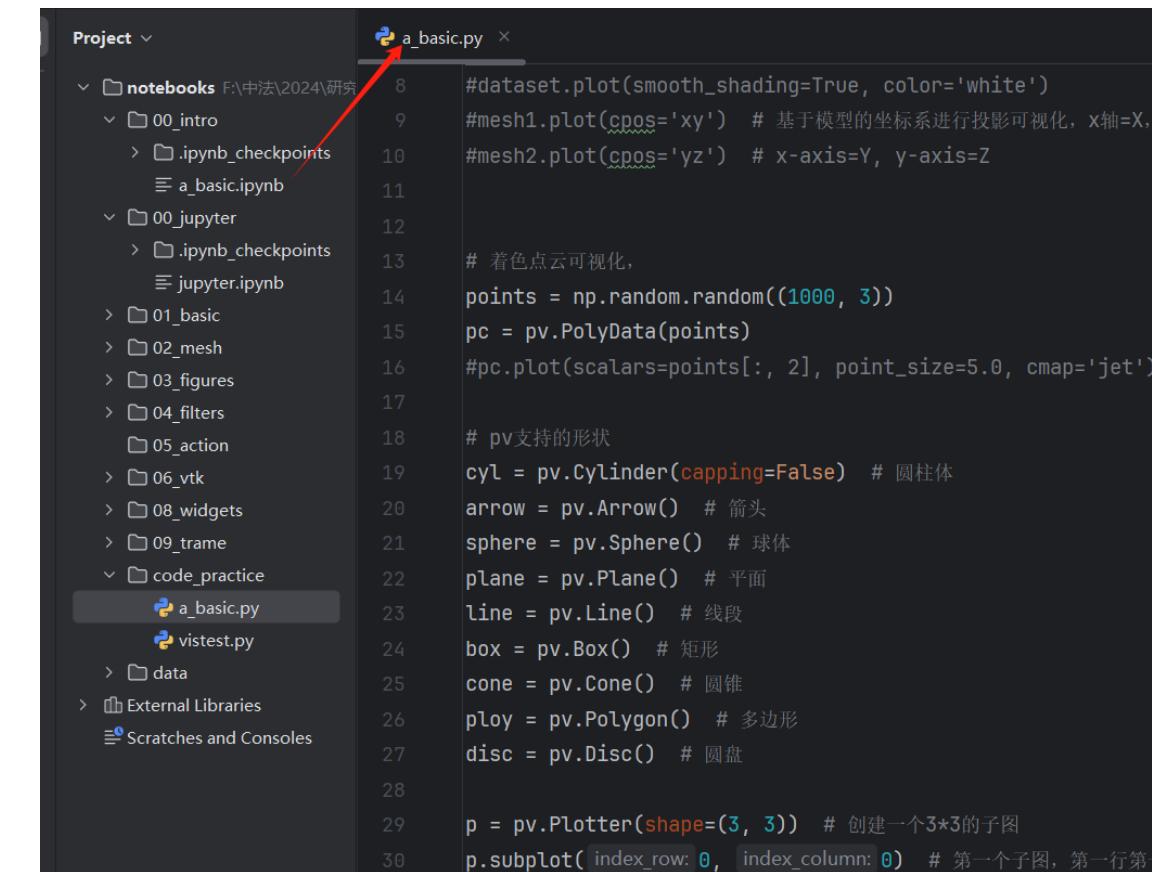
- 先安装anaconda+pycharm的编译环境。
- 建议安装AI辅助插件：MarsCode AI或CodeGeeX等
- Pip安装pyvista  
`> pip install 'pyvista[all,trame]' jupyterlab`
- 先通过环境配通运行test例程。
- 每个模块都有可通过jupyter交互编程的网页代码，通过jupyter notebook在网页端启动运行。



# Part 2 | 课程内容

## » 任务

- 建议在ipython notebook交互编辑器的基础上，通过本地化，撰写一次代码（类似背单词的时候眼看、脑想、手写、嘴念），将每个例程转写为\*.py文件。
- 尝试在敲击过程中，与AI编程助手进行交互，包括但不限于：
  - 咨询函数
  - 代码注释
  - Debug
  - ...



The screenshot shows a code editor with a dark theme. On the left is a project tree titled 'Project' under 'notebooks'. It contains several subfolders: '00\_intro', '00\_jupyter', '01\_basic', '02\_mesh', '03\_figures', '04\_filters', '05\_action', '06\_vtk', '08\_widgets', '09\_trame', and 'code\_practice'. Inside 'code\_practice', there are two files: 'a\_basic.py' (which is currently selected) and 'vistest.py'. The right pane displays the contents of 'a\_basic.py'. The code uses the Paraview API (pv) to perform various visualizations:

```
#dataset.plot(smooth_shading=True, color='white')
#mesh1.plot(cpos='xy') # 基于模型的坐标系进行投影可视化, x轴=X, y轴=Y
#mesh2.plot(cpos='yz') # x-axis=Y, y-axis=Z
# 着色点云可视化,
points = np.random.random((1000, 3))
pc = pv.PolyData(points)
#pc.plot(scalars=points[:, 2], point_size=5.0, cmap='jet')

# pv支持的形状
cyl = pv.Cylinder(capping=False) # 圆柱体
arrow = pv.Arrow() # 箭头
sphere = pv.Sphere() # 球体
plane = pv.Plane() # 平面
line = pv.Line() # 线段
box = pv.Box() # 矩形
cone = pv.Cone() # 圆锥
ploy = pv.Polygon() # 多边形
disc = pv.Disc() # 圆盘

p = pv.Plotter(shape=(3, 3)) # 创建一个3*3的子图
p.subplot(index_row: 0, index_column: 0) # 第一个子图, 第一行第一列
```

查询函数的两种方式：

- 1) 通过F4 进入函数直接看
- 2) 通过Print (pv.core.utilities.reader.get\_reader) # 完整调用路径在terminal查看

# Part 2 | 课程内容

## » Jupyter\_AI 自行折腾

- 注: **jupyter** 中与AI交互
- <https://jupyter-ai.readthedocs.io/en/latest/users/index.html>

```
pip install 'jupyter-  
ai[all]'
```

- 注意重新启动浏览器，以及jupyter
- 同时，需要配置大语言模型，有些需要自己注册**API\_key**，有些需要本地进行部署（依赖本地算力，不完全推荐）

### Model providers

Jupyter AI supports a wide range of model providers and models. To use Jupyter AI with a particular provider, you must install its Python packages and set its API key (or other credentials) in your environment or in the chat interface.

Jupyter AI supports the following model providers:

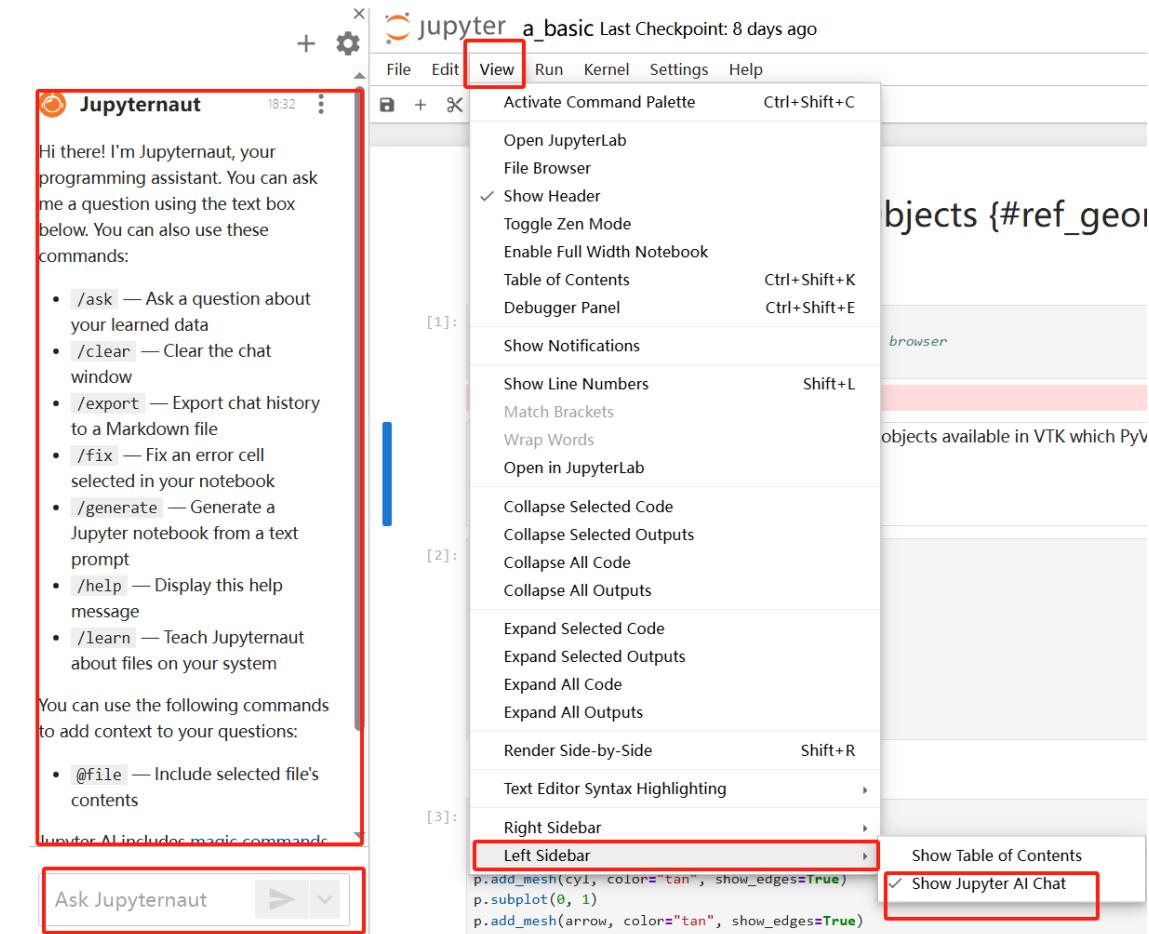
Provider	Provider ID	Environment variable(s)	Python package(s)
AI21	ai21	AI21_API_KEY	ai21
Anthropic	anthropic	ANTHROPIC_API_KEY	langchain-anthropic
Anthropic (chat)	anthropic-chat	ANTHROPIC_API_KEY	langchain-anthropic
Bedrock	bedrock	N/A	langchain-aws
Bedrock (chat)	bedrock-chat	N/A	langchain-aws
Bedrock	bedrock-custom	N/A	langchain-aws

On this page

- Prerequisites
- Installation
- Uninstallation
- Model providers**
- The chat interface
- The `%ai` and `%%ai` magic commands
- Configuration

This Page

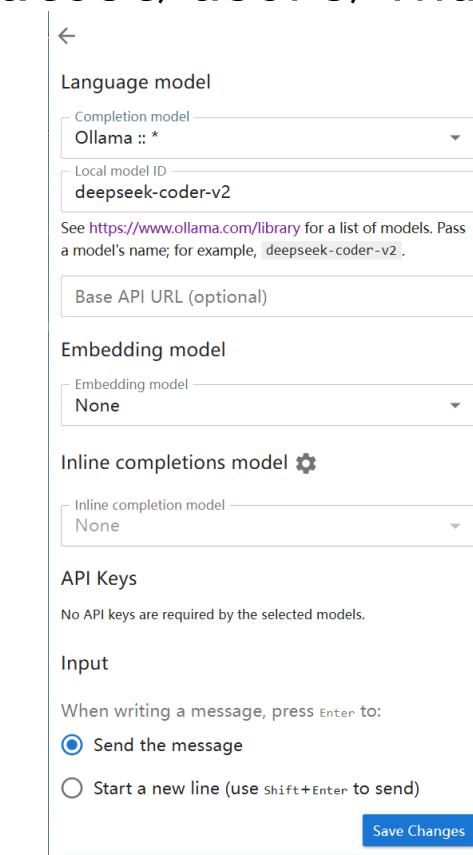
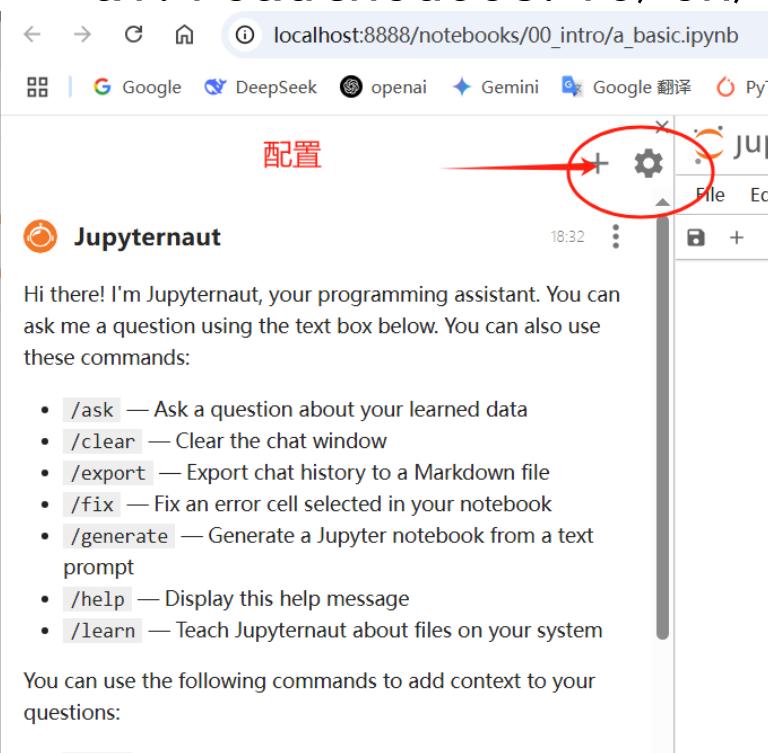
- [Show Source](#)



# Part 2 | 课程内容

## » Jupyter\_AI 自行折腾

- 注: **jupyter** 中与AI交互
- <https://jupyter-ai.readthedocs.io/en/latest/users/index.html>



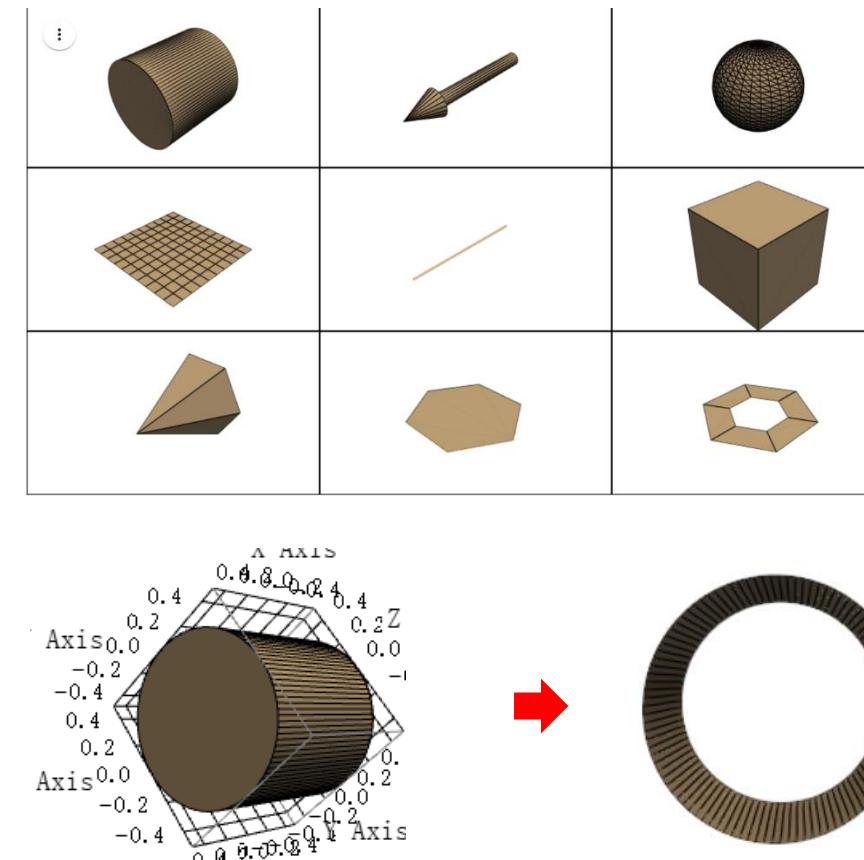
- 有兴趣可自行摸索配置
- 目前尚不支持国产大模型（在国内的）云端 API
- 需要在本地配置
- 此外，交互窗口也一般
- 推荐采用 IDE+插件+联网调用的方式

# Part 2 | 课程内容

## » 基础几何类 00\_a\_basic.py

- 主要是PyVista中可以定义和绘制的基础三维网格结构，包括如下：

```
# pv支持的形状
cyl = pv.Cylinder() # 圆柱体
arrow = pv.Arrow() # 箭头
sphere = pv.Sphere() # 球体
plane = pv.Plane() # 平面
line = pv.Line() # 线段
box = pv.Box() # 矩形
cone = pv.Cone() # 圆锥
ploy = pv.Polygon() # 多边形
disc = pv.Disc() # 圆盘
```



了解相应函数，将鼠标光标定位到相应的类函数，如Cylinder()，通过F4进入类定义，查看默认参数定义，包含了圆柱体的：中心坐标、朝向（roll pitch yaw），半径，高度，分辨率，是否封闭。

```
解释 | 添加注释 | X
def Cylinder(
    center=(0.0, 0.0, 0.0),
    direction=(1.0, 0.0, 0.0),
    radius=0.5,
    height=1.0,
    resolution=100,
    capping=True,
):
    """Create the surface of a cylinder.
```

函数定义后是参数的注释，通过修改默认参数，调整绘制结果，如capping=False。

# Part 2 | 课程内容

## » 基础几何类

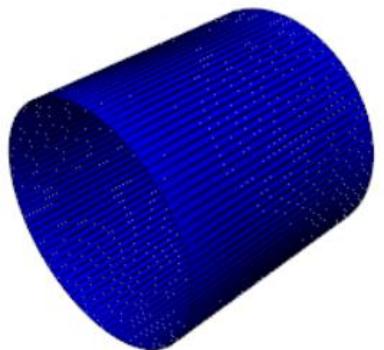
```
p = pv.Plotter(shape=(3, 3)) # 创建一个3*3的子图  
p.subplot(0, 0) # 第一个子图，第一行第一列，0, 0  
p.add_mesh(cyl, color='tan', show_edges=True) # 输入参数的数据结构十分长，多元化可控性高
```

- 关键函数为 `add_mesh`, 参数列表相当长

```
解释 | 添加注释 | X  
def add_mesh(  
    self,  
    mesh,  
    color=None,  
    style=None,  
    scalars=None,  
    clim=None,  
    show_edges=None,  
    edge_color=None,  
    point_size=None,  
    line_width=None,  
    opacity=None,  
    flip_scalars=False,  
    lighting=None,  
    n_colors=256,  
    interpolate_before_map=None,  
    cmap=None,  
    label=None,  
    reset_camera=None,  
    scalar_bar_args=None
```

- 了解参数的定义，尝试修改参数，显示绘制结果。

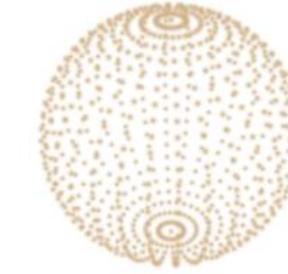
`color='blue'`



`line_width=5.`  
3



`style='points_gaussian'`



# Part 2 | 课程内容

## » Jupyter

- Web端的 3D可视化渲染与交互依赖 Trame的后端
- <https://kitware.github.io/trame/>

- 分为 服务端渲染: server-side rendering mode

- `plotter.show(jupyter_backend='trame')` #逐次对每个画布申明

- 或者在初始申明 `pv.set_jupyter_backend('trame')`

- 客户端渲染client-rendering

```
pl = pv.Plotter()
pl.add_mesh(pv.ParametricRandomHills().elevation())
pl.show(jupyter_backend="client")
```

```
pl = pv.Plotter()
pl.add_volume(pv.Wavelet())
pl.show(jupyter_backend="server")
```

```
: import pyvista as pv
# Set/enable the backend
pv.set_jupyter_backend("trame")
```

```
: pl = pv.Plotter()
pl.add_mesh(pv.ParametricKlein())
pl.show()
```

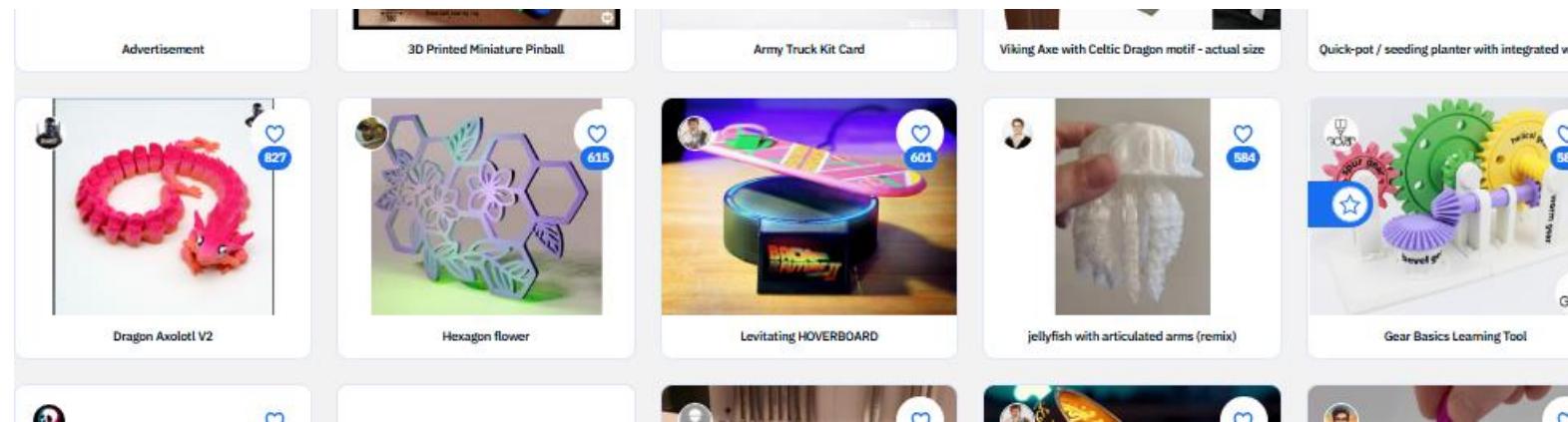
本地做实验时，无需区分，因为本机的python虚拟环境即server。

上述模式主要是，当在本地服务器开启jupyter服务，学生端仅通过浏览器环境无Trame时，方便学生无需安装环境使用

# Part 2 | 课程内容

## » 基本用途

- 数据源获取和绘制
  - 使用pyvista在线数据: `from pyvista import examples;`  
`examples.download_pine_XX`
  - 下载满足格式的第三方三维数据: 如3D打印网站的建模
  - 使用本地传感器采集数据, 并将数据转化为pyvista的数据结构。

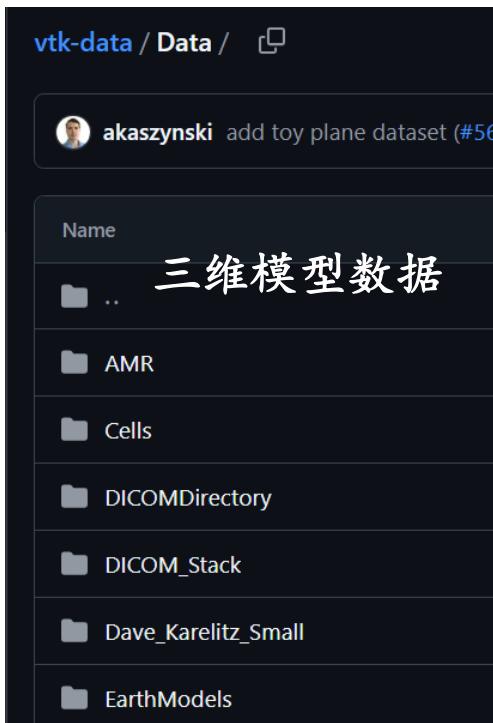


# Part 2 | 课程内容

## » 基本用途: pyvista数据

- 使用默认下载函数, 下载内置数据。相关数据维护在<https://github.com/pyvista/vtk-data/>, github访问可能偶尔需要挂梯子。

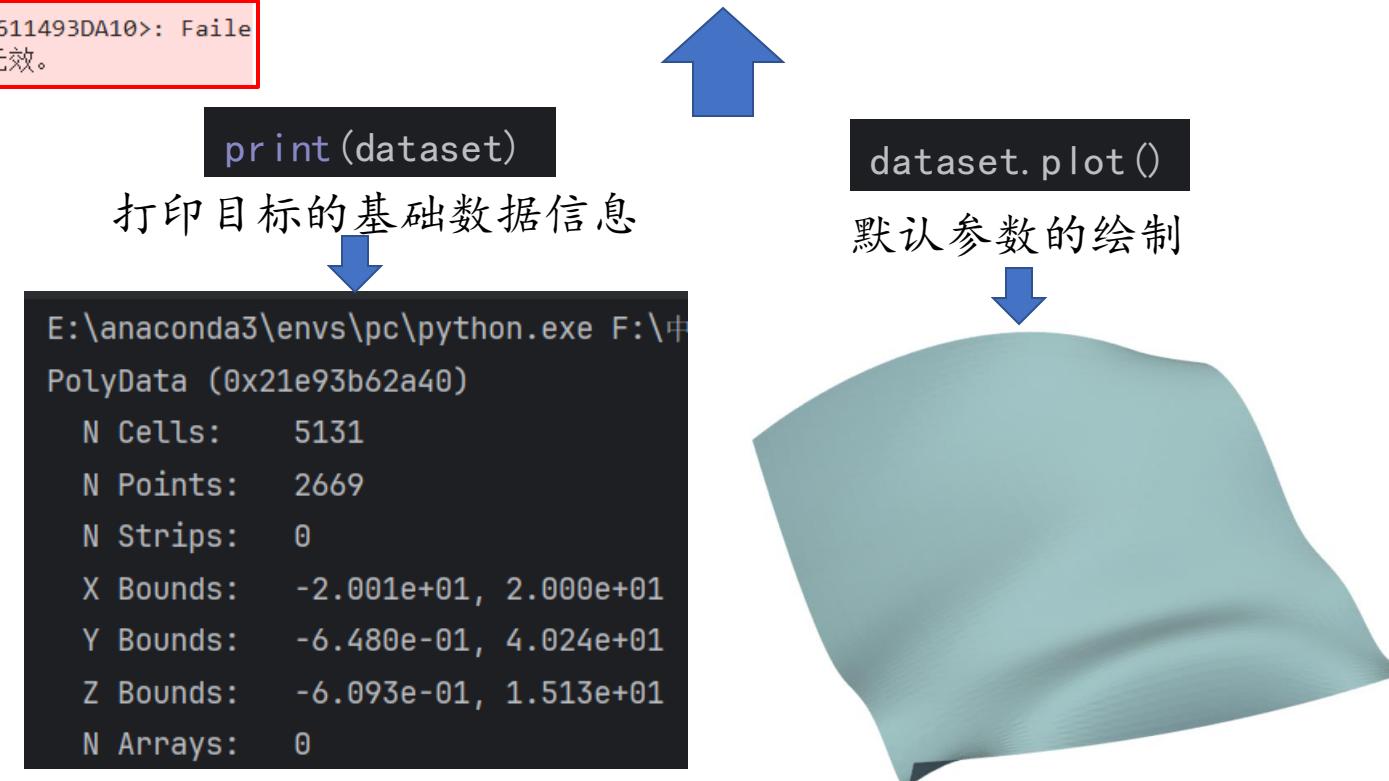
```
NewConnectionError: <urllib3.connection.HTTPSConnection object at 0x000001611493DA10>: Failed to establish a new connection: [WinError 10049] 在其上下文中, 该请求的地址无效。
```



<https://docs.pyvista.org/api/core/objects>

Two pieces of information are required: **the data's geometry**, which describes where the data is positioned in space and what its values are, and **its topology**, which describes how points in the dataset are connected to one another.

Geometry in PyVista is represented as points and cells.



# Part 2 | 课程内容

## » 基本用途：本地雷达点云数据

00\_a\_basic.py

pv.read() 所支持的格式包括VTK 和 Meshio(右图)  
相对比较常见的有OBJ和PLY

Meshio定义的Mesh包含 顶点 points 和 cells

Points为坐标构成的数组2维或3维

Cells为元组列表

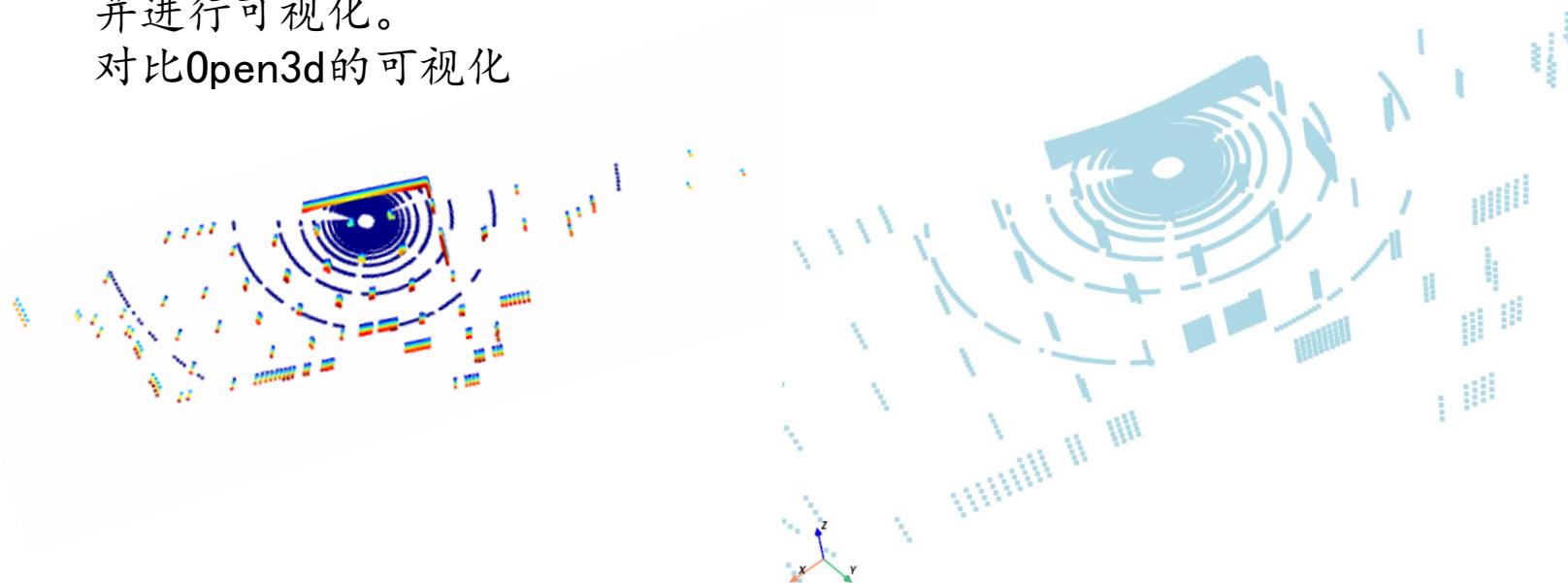
```
# two triangles and one quad
points = [
    [0.0, 0.0],
    [1.0, 0.0],
    [0.0, 1.0],
    [1.0, 1.0],
    [2.0, 0.0],
    [2.0, 1.0],
]
cells = [
    ("triangle", [[0, 1, 2], [1, 3, 2]]),
    ("quad", [[1, 4, 5, 3]]),
]

mesh = meshio.Mesh(
    points,
    cells,
    # Optionally provide extra data on points, cells, etc.
    point_data={"T": [0.3, -1.2, 0.5, 0.7, 0.0, -3.0]},
    # Each item in cell data must match the cells array
    cell_data={"a": [[0.1, 0.2], [0.4]]},
)
```

Abaqus (.inp), ANSYS msh (.msh), AVS-UCD (.avs), CGNS (.cgns), DOLFIN XML (.xml), Exodus (.e, .exo), FLAC3D (.f3grid), H5M (.h5m), Kratos/MDPA (.mdpa), Medit (.mesh, .meshb), MED/Salome (.med), Nastran (bulk data, .bdf, .fem, .nas), Netgen (.vol, .vol.gz), Neuroglancer precomputed format, Gmsh (format versions 2.2, 4.0, and 4.1, .msh), OBJ (.obj), OFF (.off), PERMAS (.post, .post.gz, .dato, .dato.gz), PLY (.ply), STL (.stl), Tecplot.dat, TetGen.node/.ele, SVG (2D output only) (.svg), SU2 (.su2), UGRID (.ugrid), VTK (.vtk), VTU (.vtu), WKT (TIN) (.wkt), XDMF (.xdmf, .xmf).

([Here's a little survey](#) on which formats are actually used.)

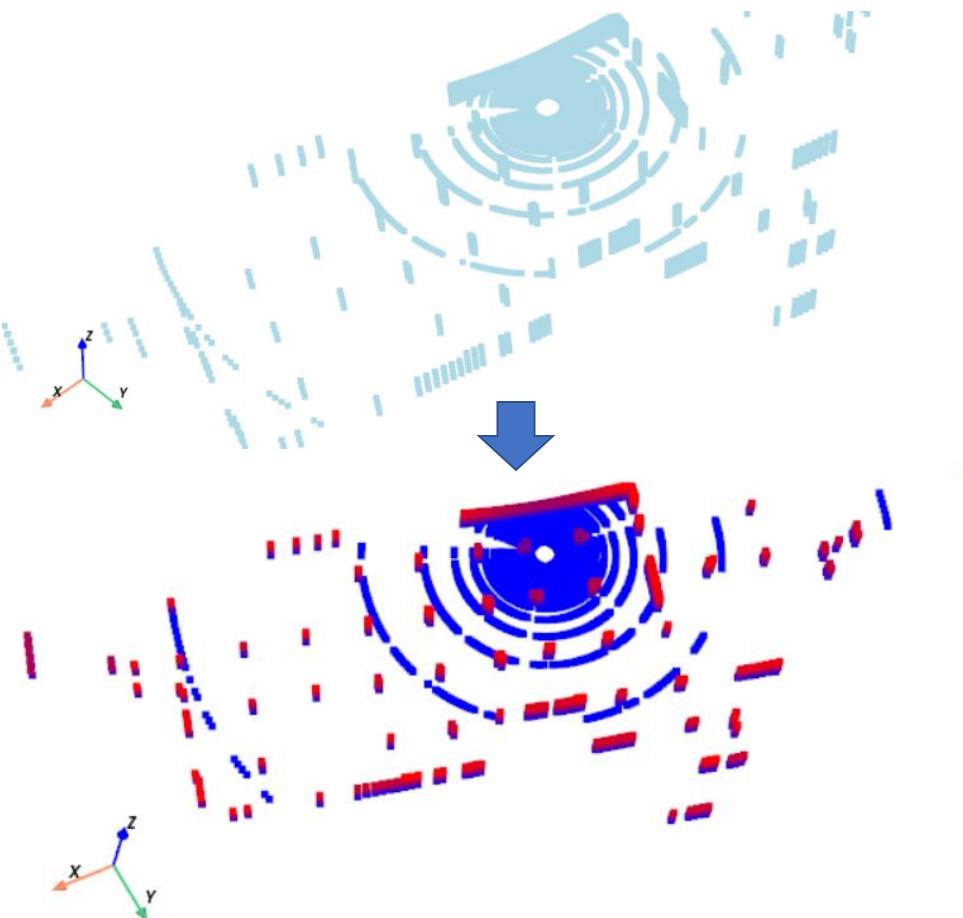
实验：尝试将KITTI的bin文件，转化成可被读取的mesh格式文件  
并进行可视化。  
对比Open3d的可视化



# Part 2 | 课程内容

## » 基础几何类

- 类似open3d 对pyvista的点进行着色， RGB值基于Z-轴/高度。



```
cells = [("vertex", np.arange(len(lidar)).reshape(-1, 1))]
z_values = lidar[:, 2]
z_normalized = (z_values - z_values.min()) / (z_values.max() -
z_values.min())
color = np.zeros((len(lidar), 3))
color[:, 0] = z_normalized
color[:, 2] = 1 - z_normalized

mesh = meshio.Mesh(points=lidar, cells=cells, point_data={"colors": color})
```

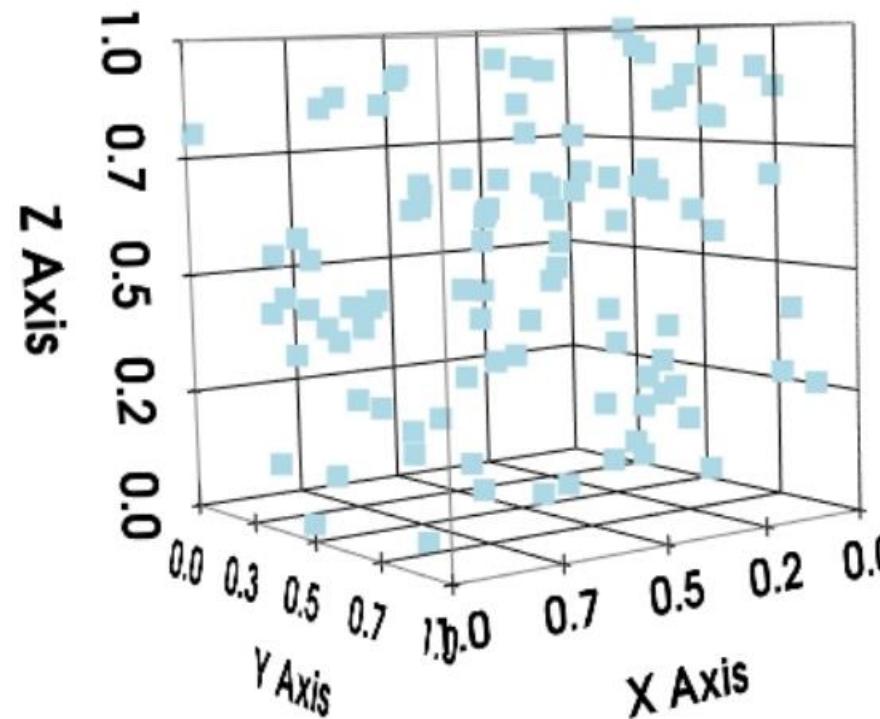
# Part 2 | 课程内容

## » Mesh的基本数据结构

02\_basic data structure of mesh.py

- Points

直接基于numpy的数组构建N\*3矩阵，  
添加到mesh中。



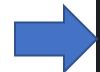
```
points = np.random.rand(300, 3)
print(points[:5, :]) # 打印前5个点的坐标
print(points.shape) # 打印点的数量和维度
# vertices
mesh = pv.PolyData(points) # no connection between
points
print(mesh)
mesh.plot(point_size=10, style='points')
```

```
[[0.01225607 0.61658534 0.98448689]
 [0.42999768 0.95451637 0.02553598]
 [0.76528746 0.35510359 0.96257639]
 [0.80900929 0.78076612 0.07470049]
 [0.17071116 0.14825986 0.61132915]]
(300, 3)
PolyData (0x199b3676aa0)
N Cells: 300
N Points: 300
N Strips: 0
X Bounds: 7.061e-04, 9.989e-01
Y Bounds: 2.553e-03, 9.998e-01
Z Bounds: 8.467e-03, 9.924e-01
N Arrays: 0
```

# Part 2 | 课程内容

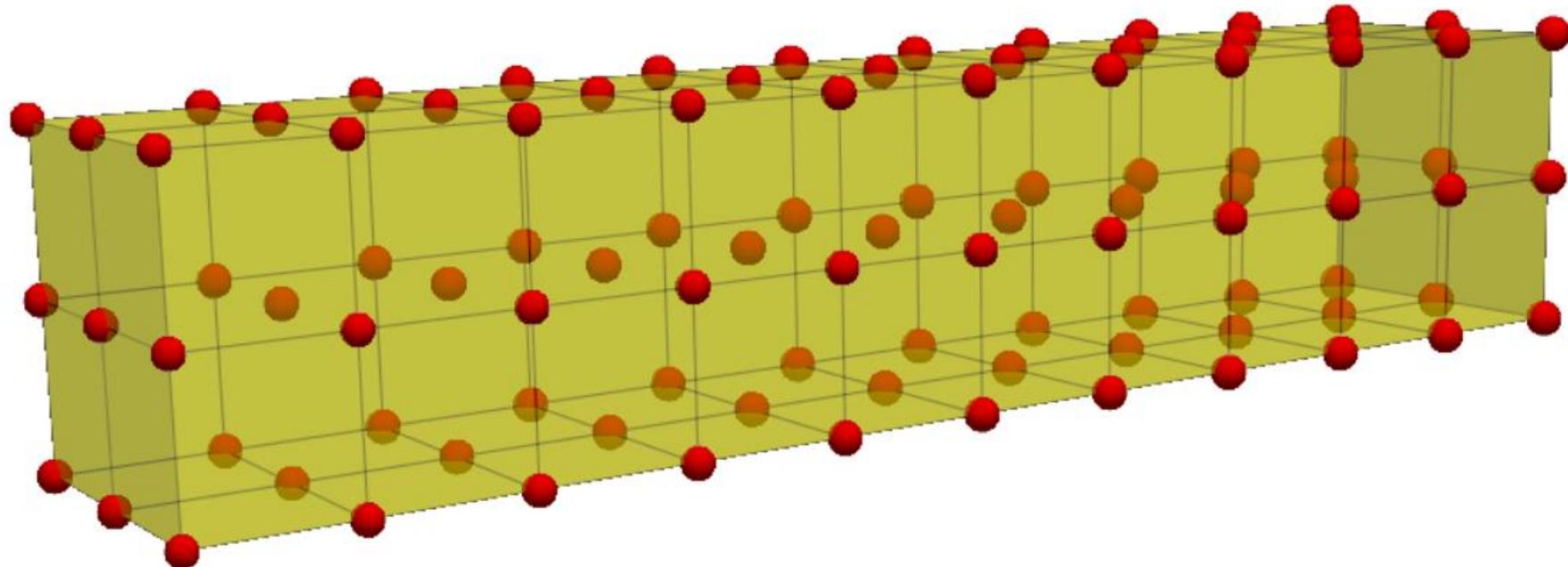
## » 基础几何类

- 可视化meshgrid中的顶点
  - 依次在画布中添加mesh
  - 以及顶点的绘制风格



```
# meshgrid and points
mesh_example = examples.load_hexbeam()
cpos = [(6.2, 3., 7.5), (0.16, 0.13, 2.65), (-0.28, 0.94,
-0.21)]

pl = pv.Plotter()
pl.add_mesh(mesh_example, style='surface',
show_edges=True, color='yellow', opacity=0.5) #,
render_lines_as_tubes=True)
pl.add_points(mesh_example.points, color='red',
point_size=20, render_points_as_spheres=True)
pl.camera_position = cpos
pl.show()
```

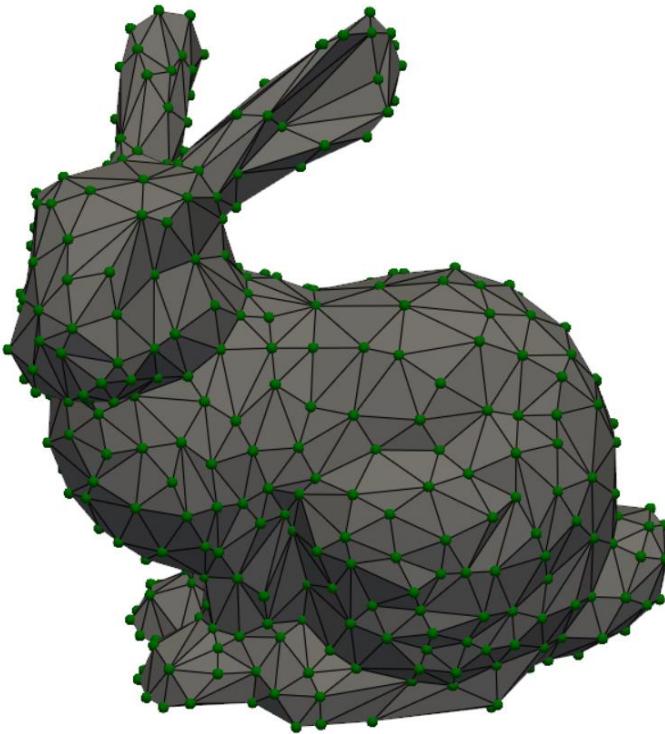


- Mesh使用黄色、透明、显示边
- Point使用红色，20，球状显示

# Part 2 | 课程内容

## » 基础几何类

- 另一个示例，使用兔子，依次添加mesh和points，并可视化



```
# bunny_coarse =
examples.download_bunny_coarse() # vpn
bunny_coarse = pv.read('../data/Bunny.vtp')
pl = pv.Plotter()
pl.add_mesh(bunny_coarse, show_edges=True,
color='gray') # 三角剖分的颜色
pl.add_points(bunny_coarse.points, color='green',
point_size=10, render_points_as_spheres=True)
pl.camera_position = cpos = [(0.02, 0.3, 0.73),
(0.02, 0.03, -0.022), (-0.03, 0.94, -0.34)]
pl.show()
```

# Part 2 | 课程内容

## » 基础几何类

- Cell边的绘制
  - cell 定义mesh中points的连接性，也就是拓扑几何关系
  - 实际上就是通过edges关联点
  - Single\_cell为cell\_example的最后一个单元，单独进行着色和绘制。

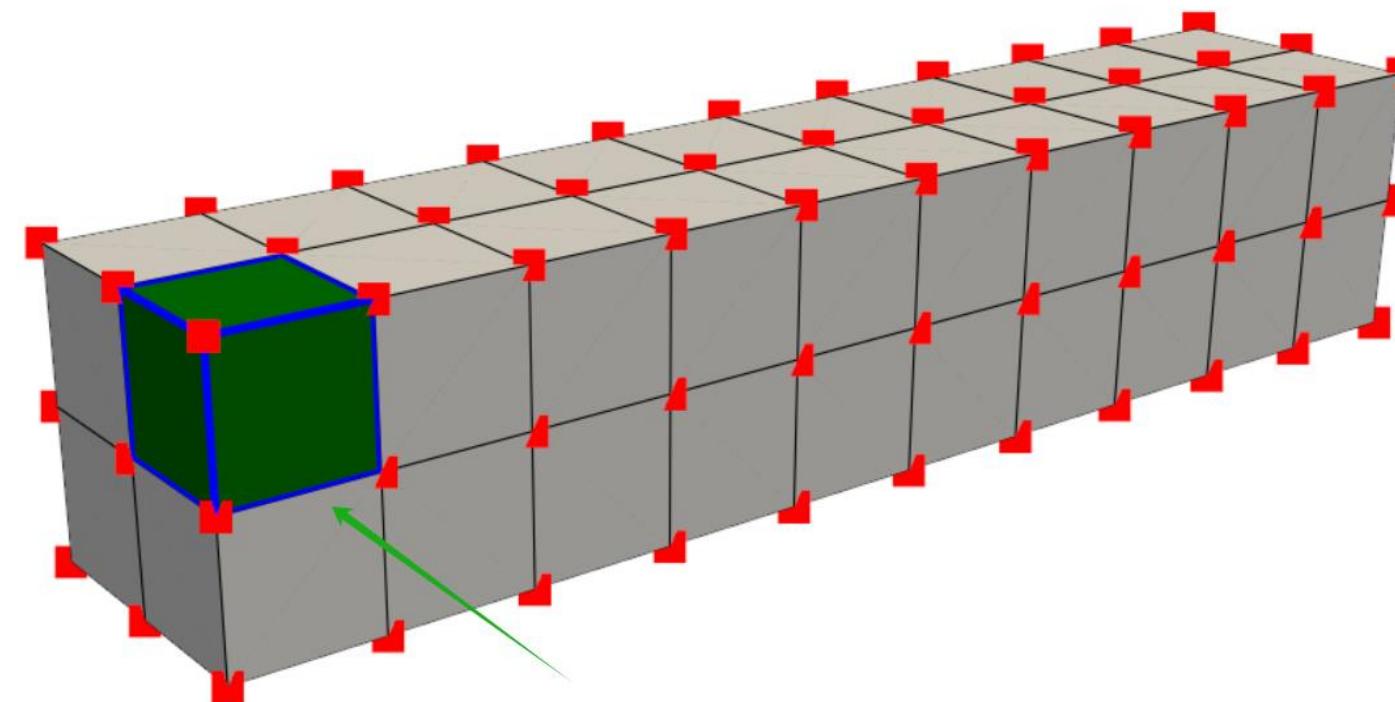


```
cell_example = examples.load_hexbeam()

pl = pv.Plotter() # new plotter
pl.add_mesh(cell_example, show_edges=True, color='white') # 三角剖分的颜色
pl.add_points(cell_example.points, color='red', point_size=20)

single_cell = cell_example.extract_cells(cell_example.n_cells - 1) # 提取最后一个cell
pl.add_mesh(single_cell, color='green', edge_color="blue", line_width=5,
show_edges=True) # , opacity=0.9) # 渲染效果似乎有点问题，单独显示最后一个cell

pl.camera_position = cpos = [(6, 2, 3, 7, 5), (0, 16, 0, 13, 2, 65), (-0, 28,
```

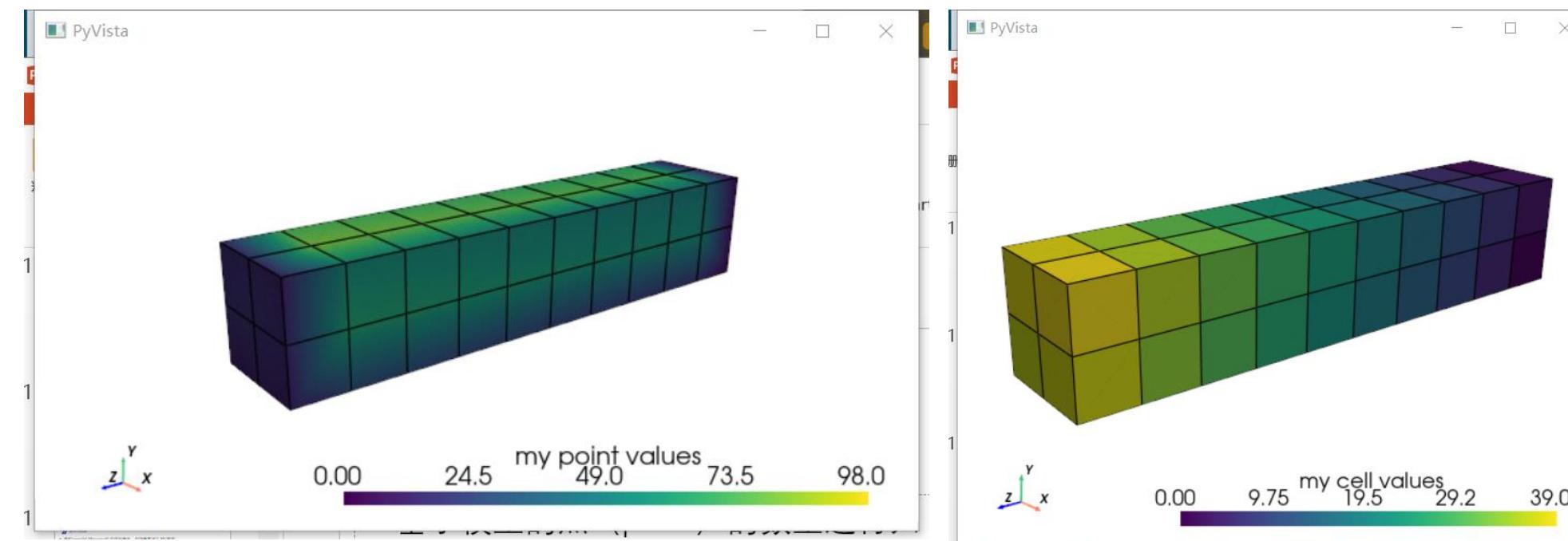


# Part 2 | 课程内容

## » 基础几何类

- 基于模型的点 (point) 的数量进行归一化，渐进着色。 $11*3*3=99$
- 基于模型的cell的数量进行归一化，渐进着色。 $10*4=40$

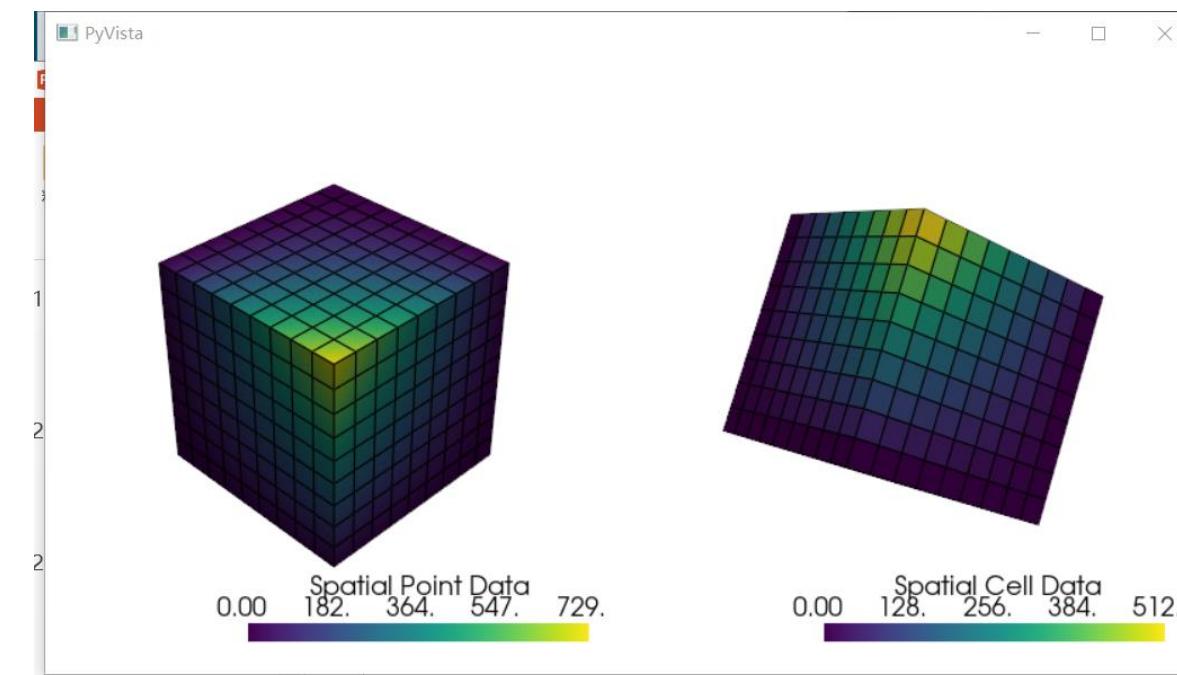
```
cell_example.point_data['my point values'] =  
np.arange(cell_example.n_points)  
cell_example.plot(scalars='my point values', cpos=cpos,  
show_edges=True)  
  
cell_example.cell_data['my cell values'] =  
np.arange(cell_example.n_cells) # 每个cell的值  
cell_example.plot(scalars='my cell values', cpos=cpos,  
show_edges=True)
```



# Part 2 | 课程内容

## » 基础几何类

- 分别对uniform的三维模型，基于点的数量
- 基于cell的数据进行渐进
- pl.link\_views() # 关联多个子视图的相机姿态



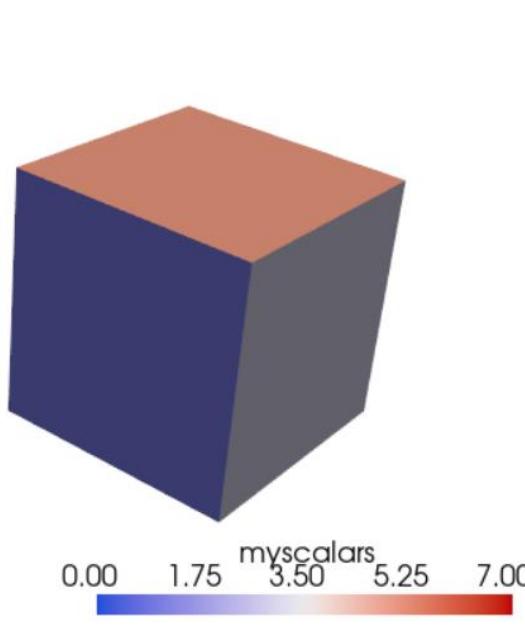
```
uni = examples.load_uniform()
pl = pv.Plotter(shape=(1, 2), border=False)
pl.add_mesh(uni, scalars='Spatial Point Data',
show_edges=True)
pl.subplot(0, 1)
pl.add_mesh(uni, scalars='Spatial Cell Data',
show_edges=True)
# pl.link_views()
pl.show()
```

# of points=  $(n+1)^3$   
# of cells =  $n^3$

# Part 2 | 课程内容

## » 基础几何类

- 单个Cube的顶点为8，面为6
- 基于这两个值对可视化进行渐进着色。



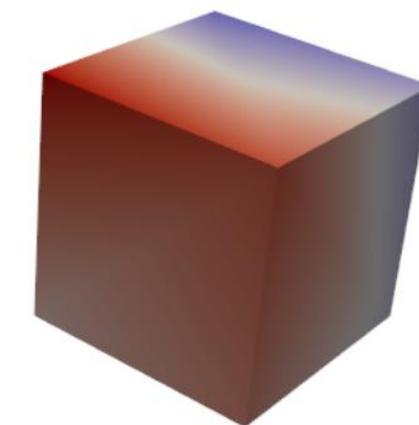
```
PolyData (0x20f34b132e0)
N Cells: 6
N Points: 8
N Strips: 0
X Bounds: -5.000e-01, 5.000e-01
Y Bounds: -5.000e-01, 5.000e-01
Z Bounds: -5.000e-01, 5.000e-01
N Arrays: 3
```

```
cube = pv.Cube()
print(cube)
cube.cell_data['myscalars'] = range(6)

other_cube = cube.copy()
print(other_cube)
other_cube.point_data['myscalars'] =
range(8)

pl = pv.Plotter(shape=(1, 2),
border_width=1)
pl.add_mesh(cube, cmap='coolwarm')

pl.subplot(0, 1)
pl.add_mesh(other_cube, cmap='coolwarm')
pl.link_views()
pl.show()
```



# Part 2 | 课程内容

## » Mesh着色

02\_mesh\_exercises.py

```
data = points[:, -1] # z-axis
point_cloud["elevation"] = data # 定义为高度

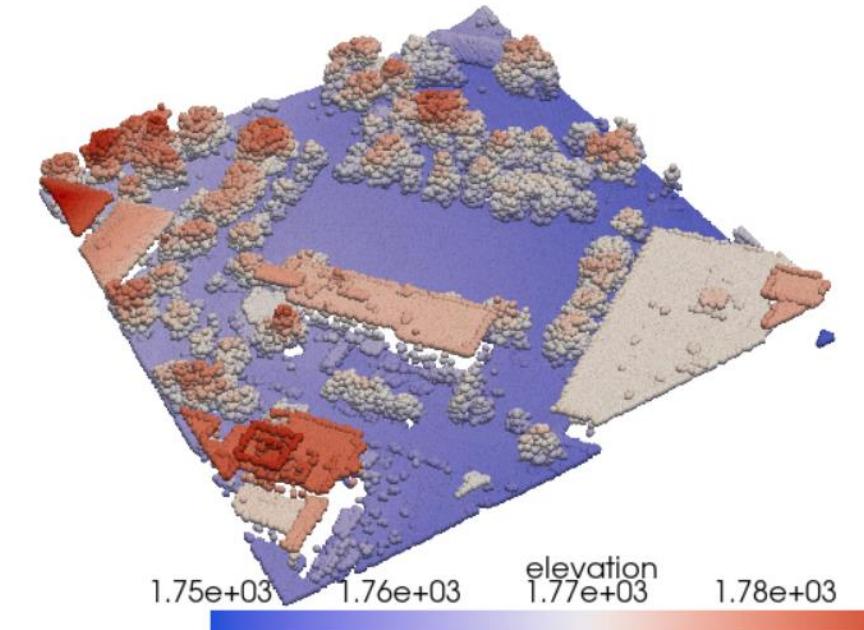
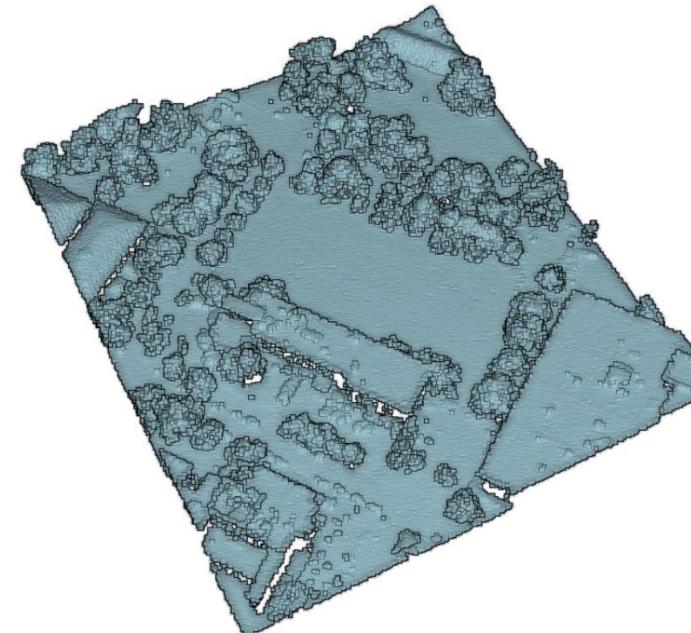
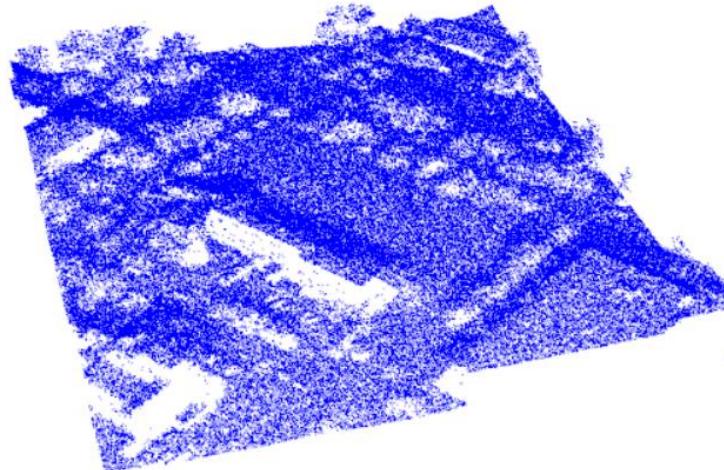
point_cloud.plot(render_points_as_spheres=True, point_size=5,
cmap="coolwarm",
scalars="elevation")
```

- 基于点云数据进行可视化

- 使用航空雷达数据，取子集：examples.download\_lidar()

- EDL可视化

```
point_cloud.plot(eye_dome_lighting=True)
```



02\_mesh\_exercises.py

# Part 2 | 课程内容

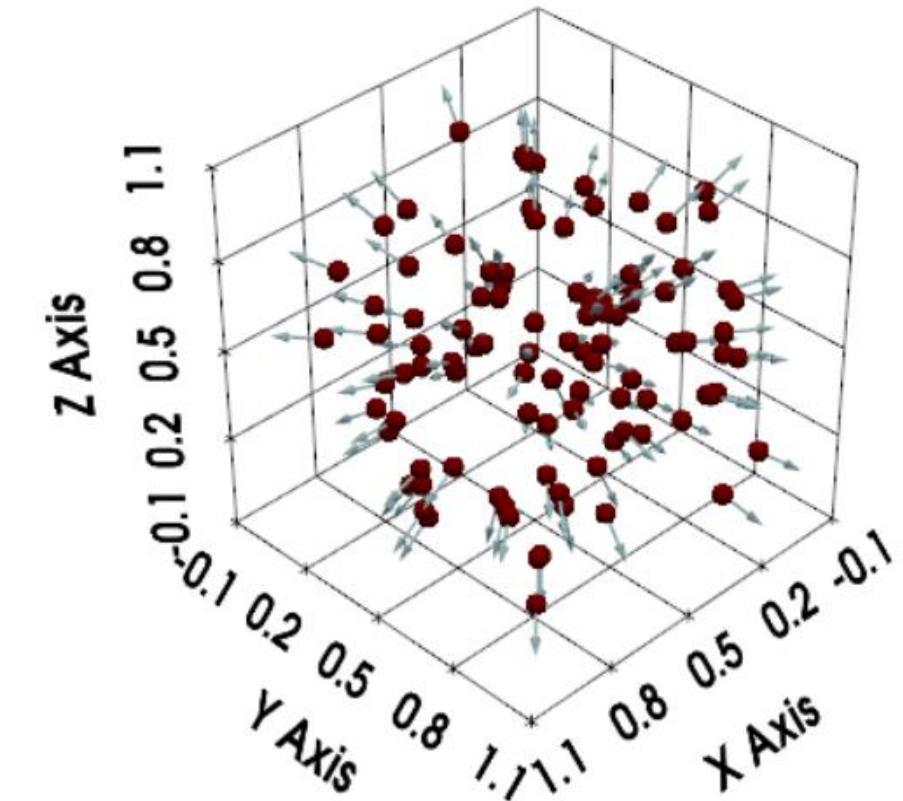
## » Mesh

- 生成随机3D点
- 基于中心（均值）生成向量，中心-点延长线朝向
- 基于向量朝向生成箭头
- 绘制
  - 增加点
  - 增加箭头

```
points = np.random.rand(100, 3)
point_cloud = pv.PolyData(points)

解释 | 添加注释 | X
def compute_vector(mesh): 1 usage
    origin = mesh.center
    vectors = mesh.points - origin # 中心归一化
    return vectors / np.linalg.norm(vectors, axis=1)[:, None]

vectors = compute_vector(point_cloud)
```

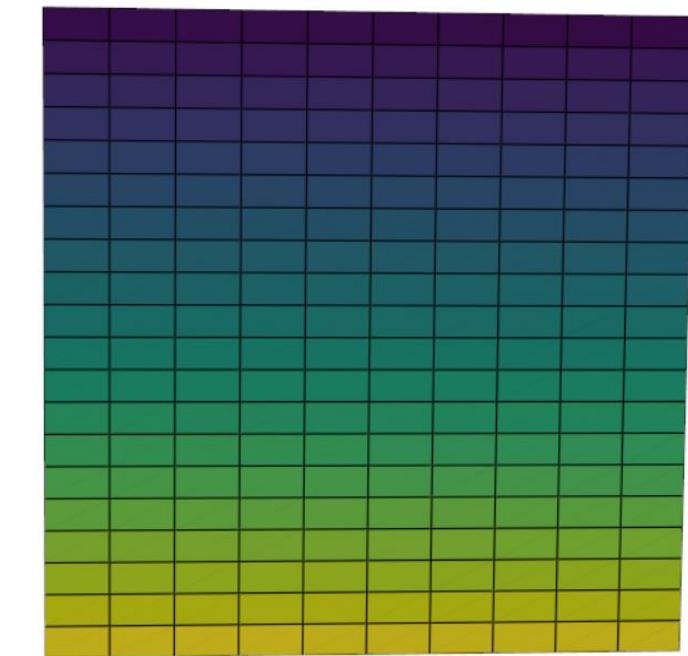
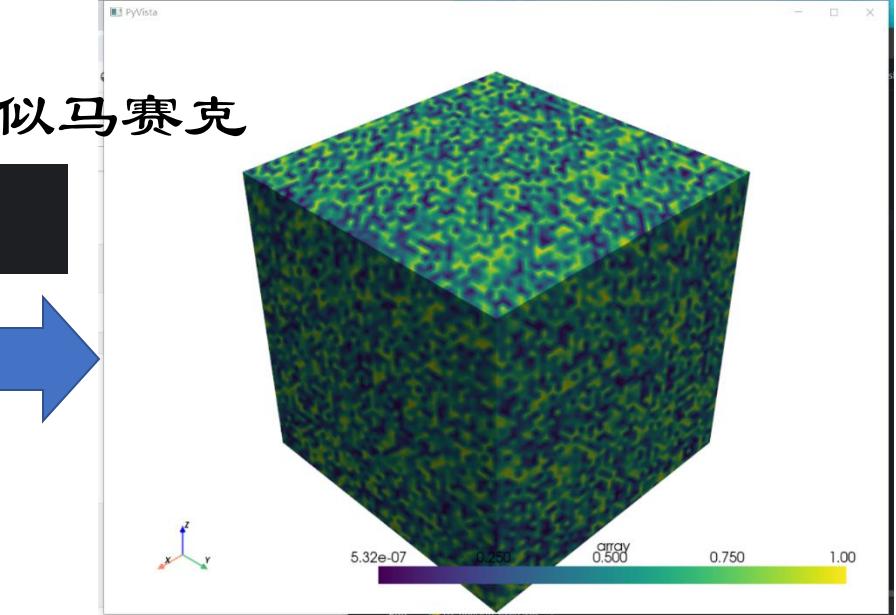
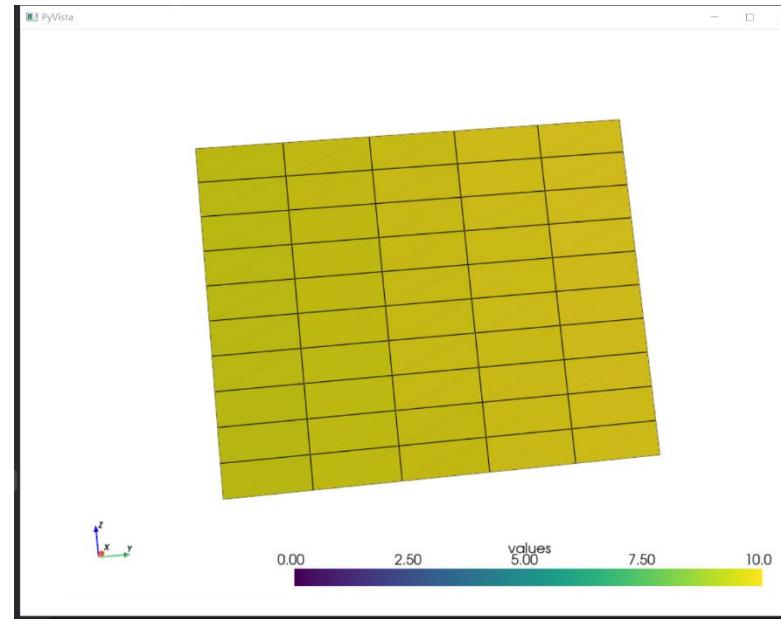
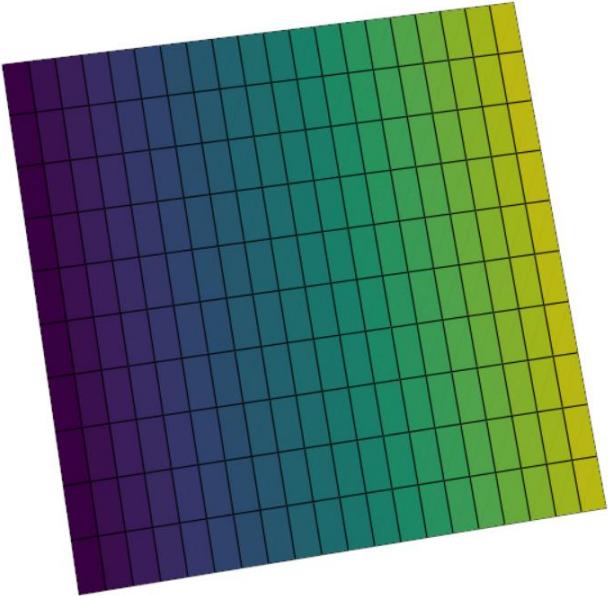


# Part 2 | 课程内容

## » Mesh - 均匀栅格

```
values = np.linspace(0, 10, 1000).reshape((20,  
5, 10))
```

- 顺序划分 $20 \times 5 \times 10$ 的矩阵



02\_uniform\_exercises.py

# Part 2 | 课程内容

## » Mesh - 非均匀栅格映射

- 对膝盖三维可视化，通过定制化透明度呈现对比sigmoid函数和非线性递增



使用预设sigmoid函数

0.00 63.8 SLCImage 128. 191. 255.



自定义数组

```
p.add_volume(vol, cmap='bone',
            opacity='sigmoid')

opacity = [0, 0, 0, 0.1, 0.3, 0.6, 1]
p.add_volume(vol, cmap='bone', opacity=opacity)
```

# 不透明度映射 (Opacity mapping) 是计算机图形学和可视化中的一种技术，用于根据标量数组的值来调整物体的不透明度。

# 这种方法常用于科学可视化，例如在医学成像、气象模拟和流体动力学等领域，以便更清晰地展示数据中的不同结构和特征。

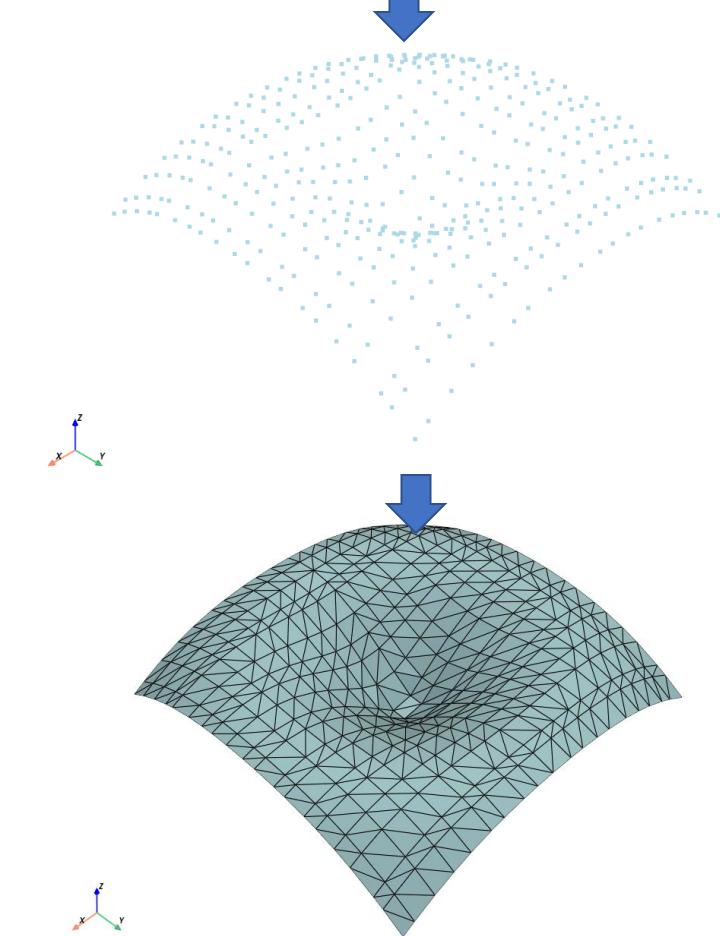
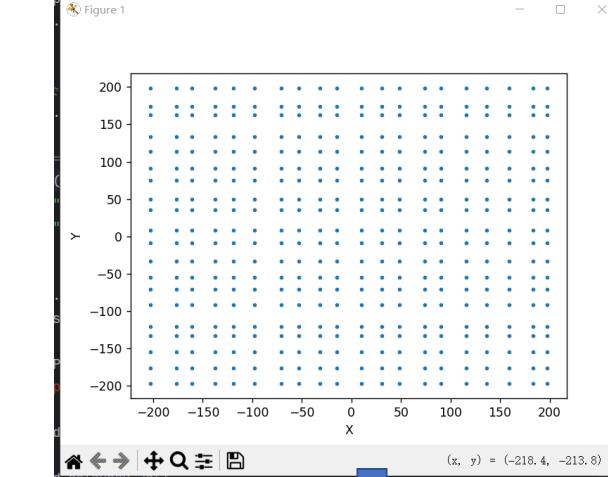
**希望非目标，如肉体的透明度增加；目标的透明度降低，剔除非骨骼组织对成像的影响。**

# Part 2 | 课程内容

## » Triangle surface

- 将三维离散点进行关联，也就是添加边，形成meshgrid，依赖德劳内三角化 (Delaunay)。

- 生成离散随机二维坐标x-y
- Z轴为归一化的sin函数  $zz = A * \sin \sqrt{xx^2 + yy^2} / b$
- 构建三维点云数据 N\*3
- 进行德劳内三角化，生成meshgrid



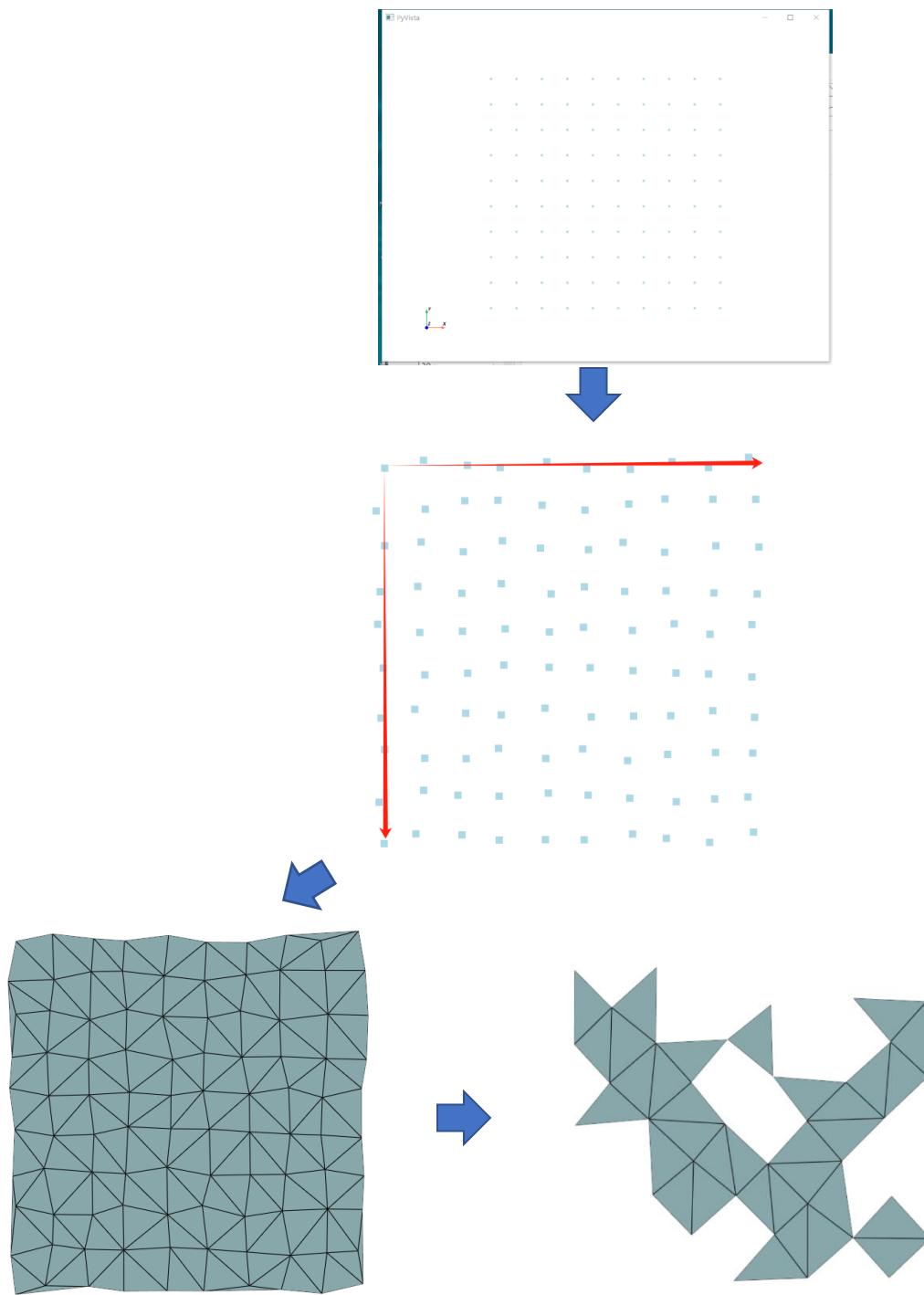
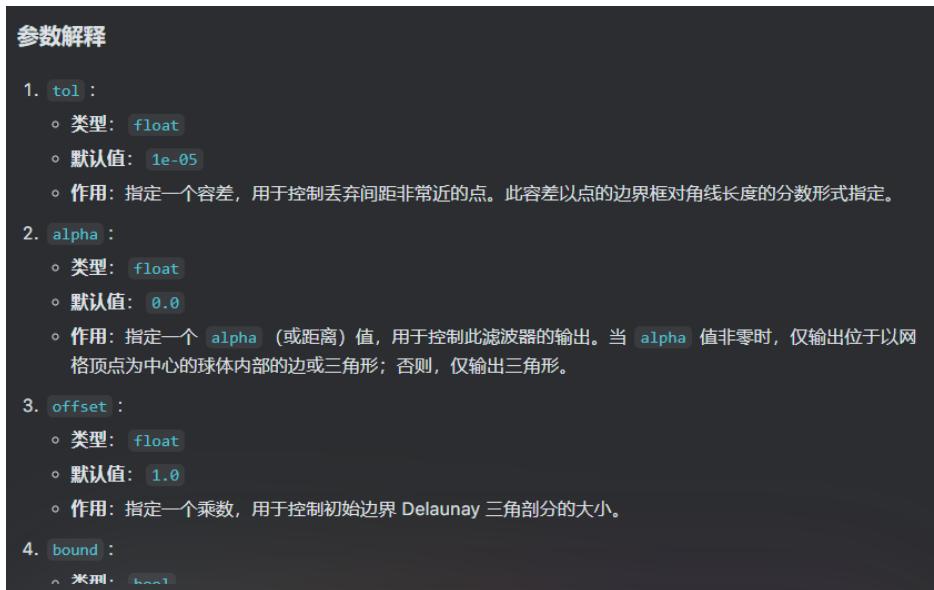
# Part 2 | 课程内容

## » Triangle surface

- 基于栅格meshgrid，降维抽取前两维 (zz=[0])

```
x = np.arange(10, dtype=float)
xx, yy, zz = np.meshgrid(x, x, [0])
```

- 加噪声
- 2d 三角化
- 过滤Triangle：基于条件策略（参数咨询AI）



# Part 2 | 课程内容

## » 读取文件

- 主要是测试对不同类型文件的可视化：

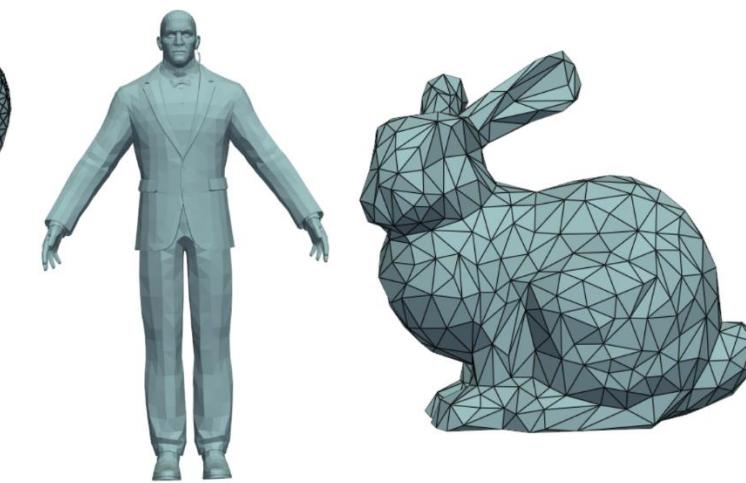
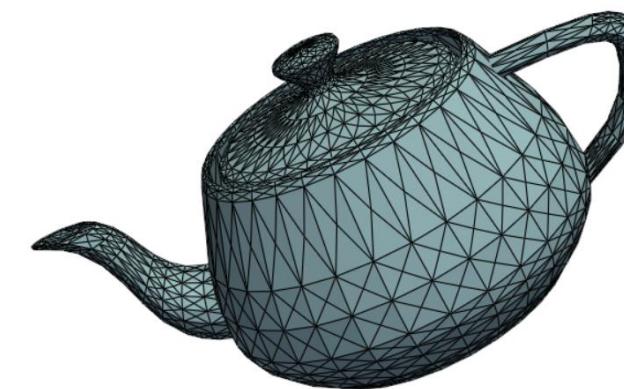
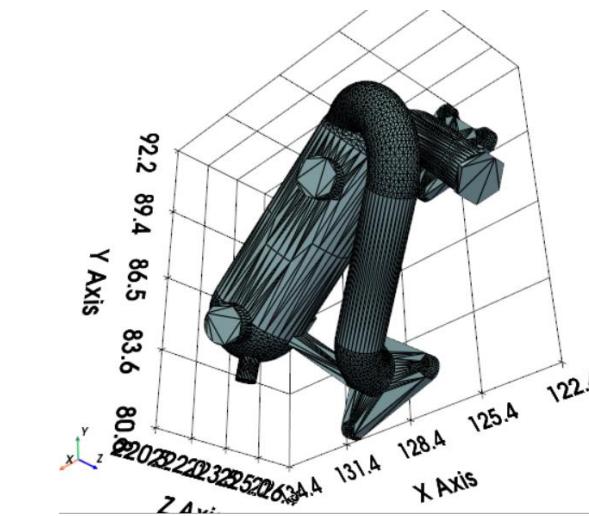
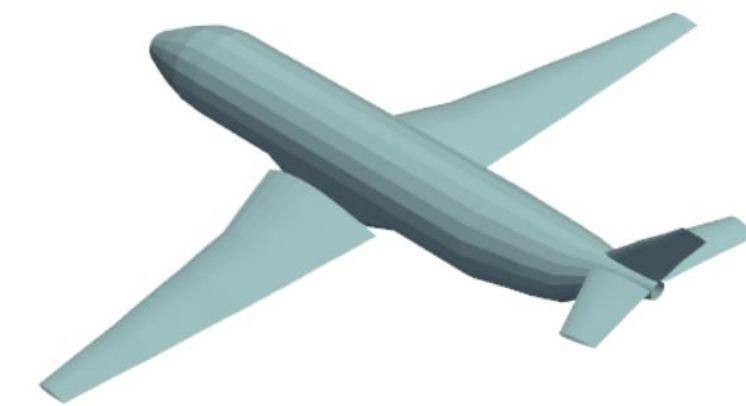
- filename = examples.planefile # 从安装的pyvista库中读取（本地虚拟环境）

```
E:\anaconda3\envs\pc\Lib\site-packages\pyvista\examples\airplane.ply
```

- 从pyvista的github上下载examples.download\_cad\_model()
- 从本地文件路径读取pv.read(path)

需要大家自己在[网络上下载不同后缀格式的3D文件，并进行可视化截图发群里。](#)

包括：PLY、STL、OBJ、VTP等

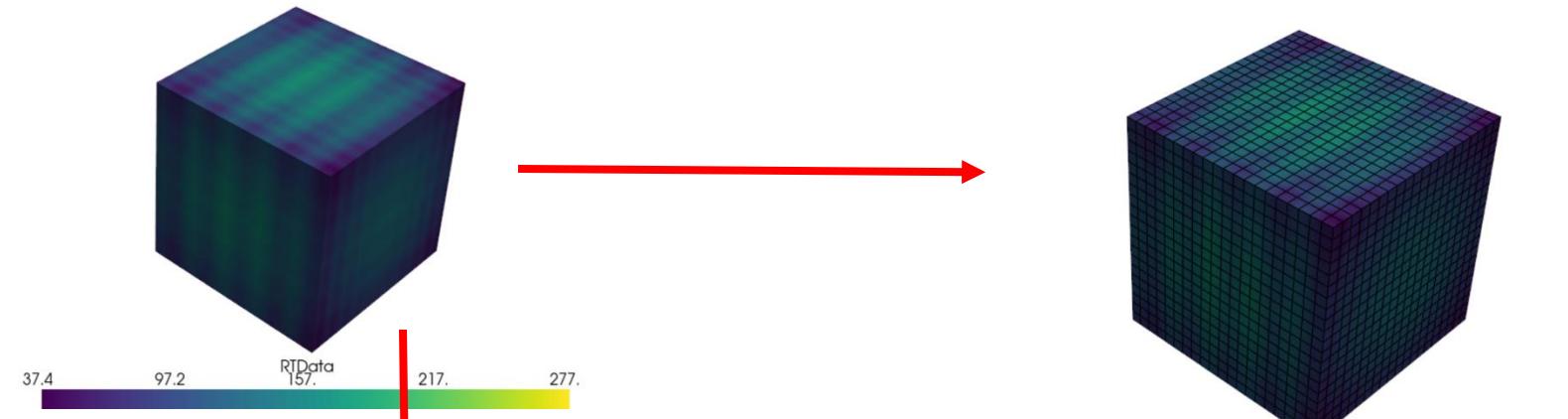


# Part 2 | 课程内容

## » 画板属性Plot

- 在画布中添加mesh并绘制

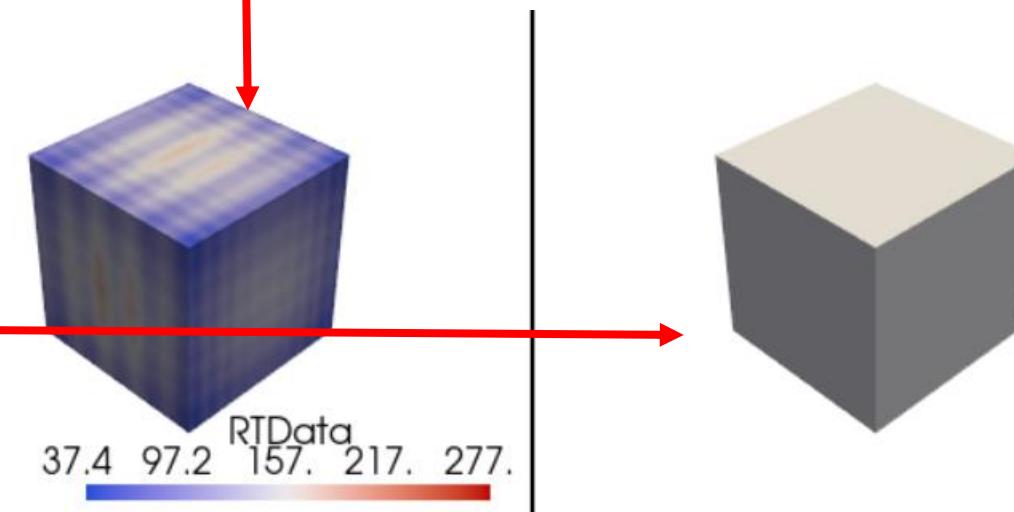
```
p = pv.Plotter()  
p.add_mesh(mesh)  
p.show()
```



- 修改mesh的配色：

- cmap: 单一维度到预定于的RGB三通道0-255映射；
- color: 单一配色

```
p = pv.Plotter(shape=(1, 2))  
p.subplot(index_row: 0, index_column: 0)  
p.add_mesh(mesh, cmap='coolwarm')  
  
p.subplot(index_row: 0, index_column: 1)  
p.add_mesh(mesh, color='w')  
p.link_views()  
p.show()
```



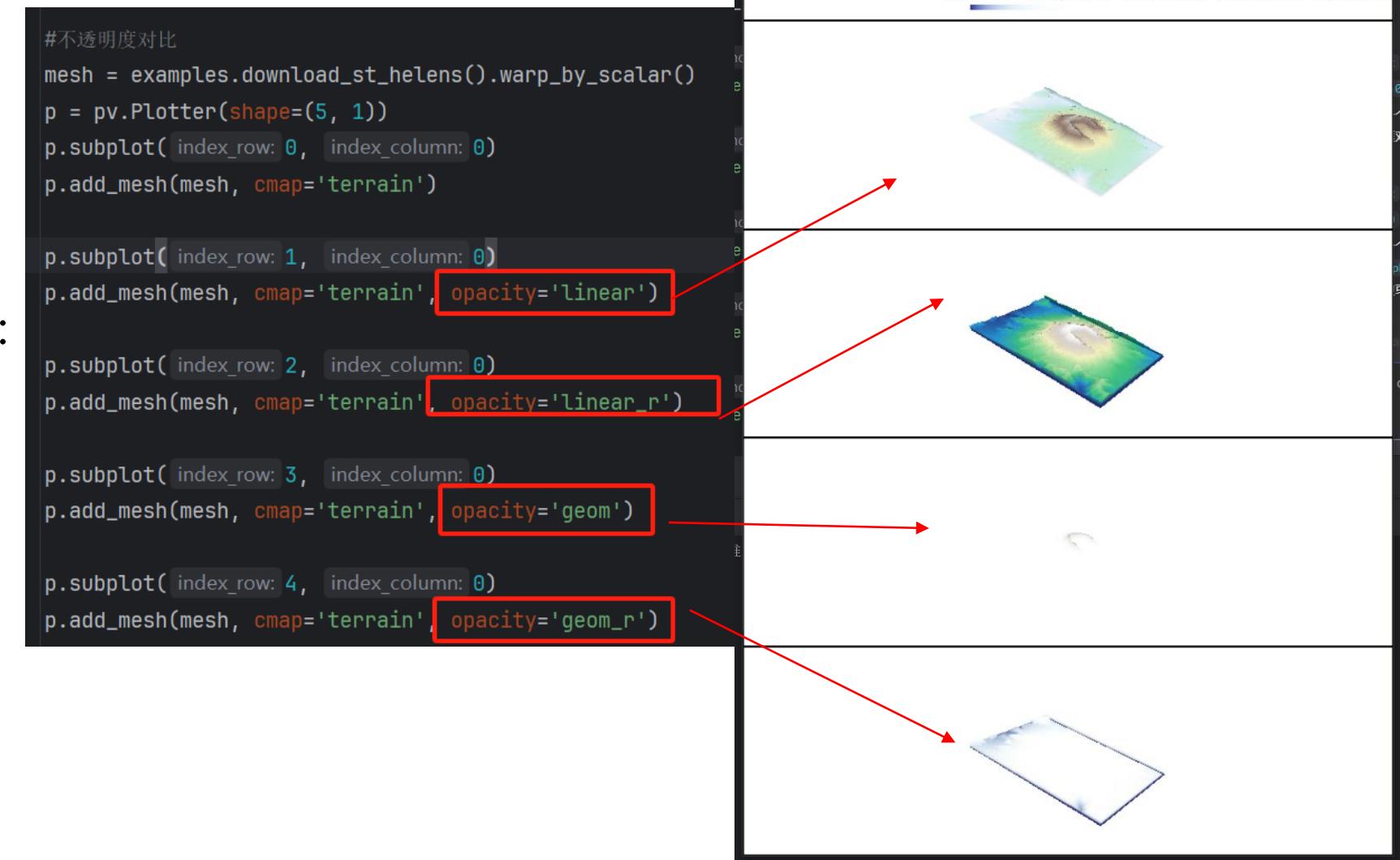
- 显示mesh的边缘

```
p.add_mesh(mesh, show_edges=True)  
p.show()
```

# Part 2 | 课程内容

## » 画板属性Plot

- 不透明度 Opacity
  - 赋值为浮点数时，全局对mesh的透明度进行调整，[0, 1]取值。
  - 可以指定预定义字符串：linear、linear\_r、geom、geom\_r
  - 用户自定义的映射函数



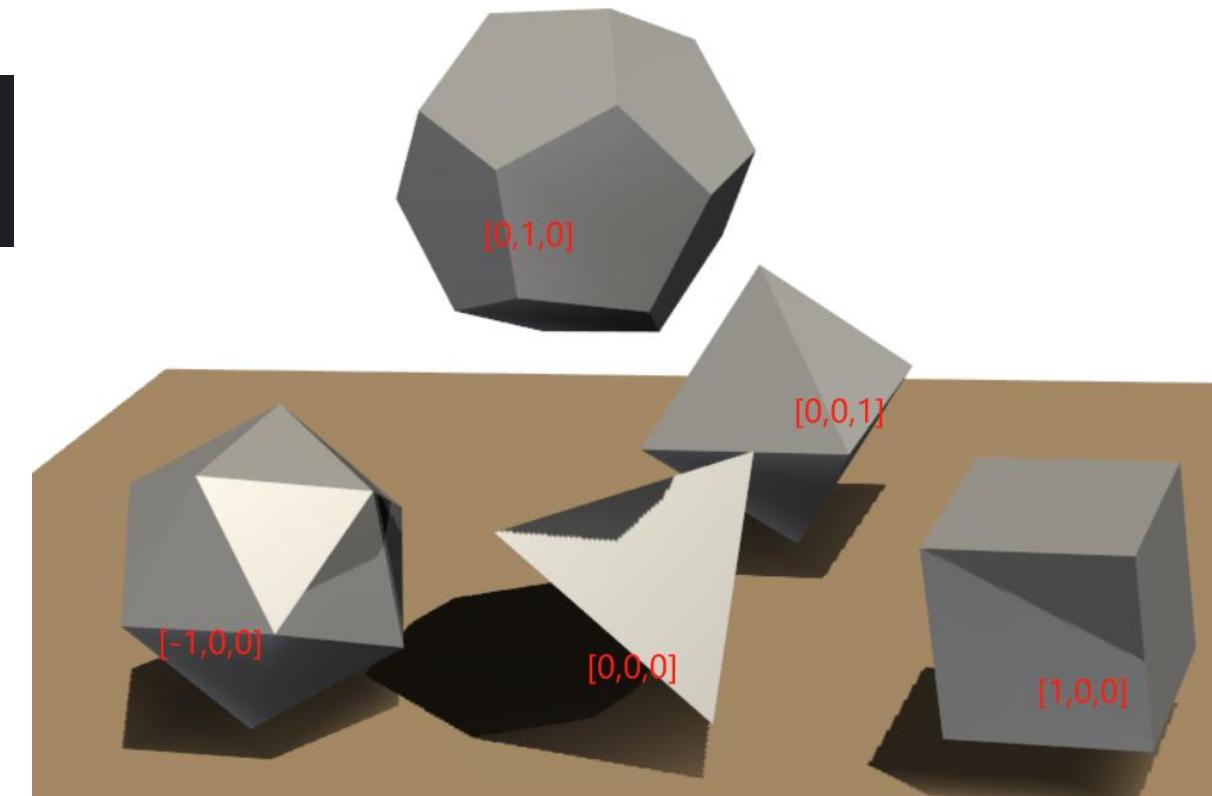
# Part 2 | 课程内容

## » 画板属性Plot

- 在同一画布添加多个不同的mesh并绘制。

```
p = pv.Plotter(window_size=[1000, 1000])
for _ind, solid in enumerate(solids):
    p.add_mesh(solid, color='silver', specular=1.0, specular_power=10)
```

```
kinds = ['tetrahedron', 'cube', 'octahedron', 'dodecahedron', 'icosahedron']
centers = [(0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1), (-1, 0, 0)]
```



# Part 2 | 课程内容

## » 画板属性Plot

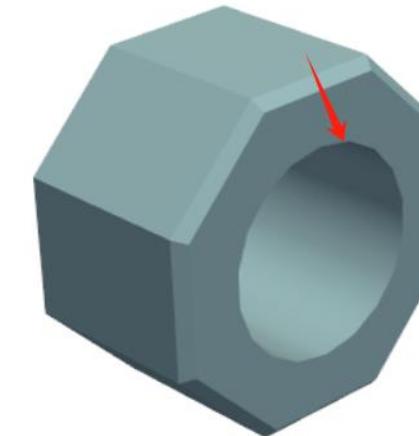
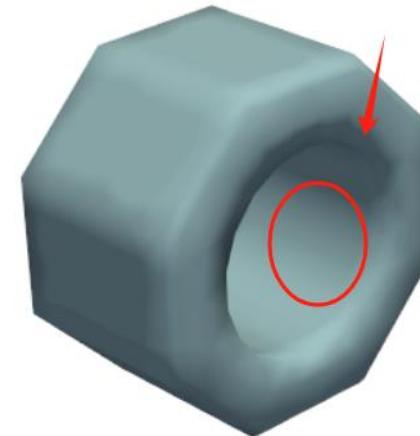
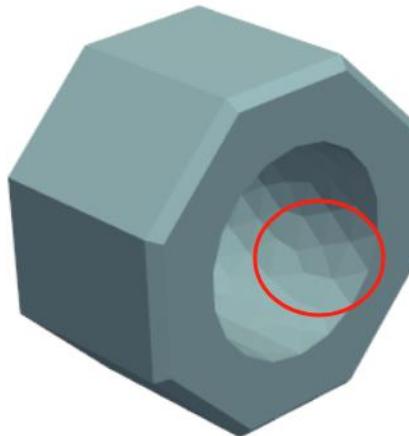
- 平滑策略

```
p.add_mesh(mesh, color="r",
           split_sharp_edges=True,
           pbr=True,
           metallic=0.5,
           roughness=0.5)
```

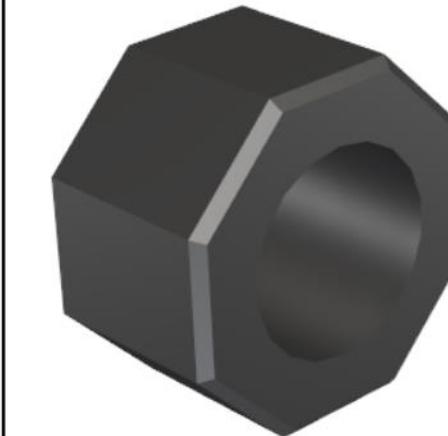
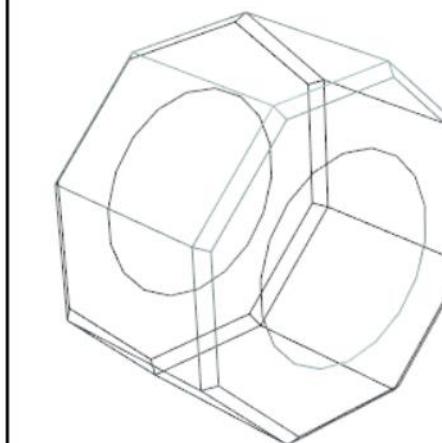
pbr启用物理渲染  
metallic真实材  
质渲染0-1  
配置反射率0-1

raw

`smooth_shading=True`



`mesh.extract_feature_edges`



`smooth_shading=True, split_sharp_edges=True`

## Part 2 | 课程内容

### » 画板属性Plot

- 将摄影图像投影到地里mesh上：  
因为本例程依赖预标定的经纬度  
信息，仅尝试可视化即可。



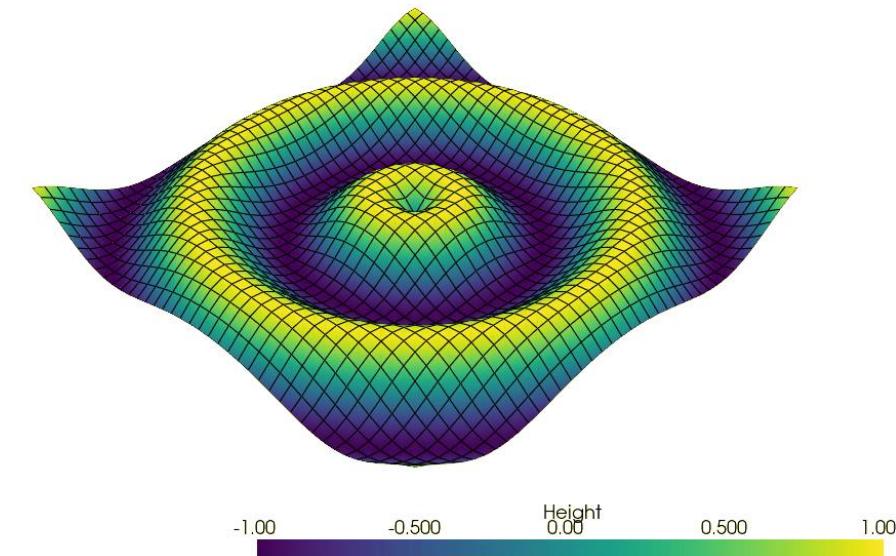
# Part 2 | 课程内容

## » gif

- 定义mesh-grid
- 定义画布
- 定义文件存储缓存plotter  
作为一个存储gif的帧缓存池
- 定义帧数量
- 刷新帧
  - 更新变量z (+ $2\pi/15$ )
  - 更新grid
  - 写入最新帧

```
x = np.arange(-10, 10, 0.5)
y = np.arange(-10, 10, 0.5)
x, y = np.meshgrid(*xi: x, y)
r = np.sqrt(x**2 + y**2)
z = np.sin(r)

grid = pv.StructuredGrid(x, y, z)
```



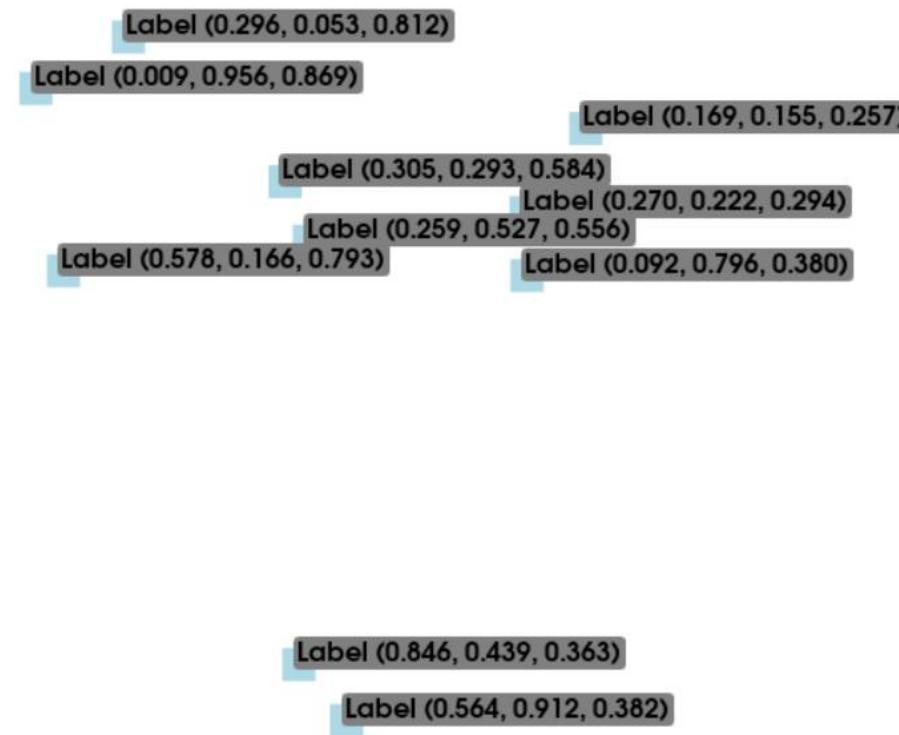
```
nframe = 15
#[0, 2*pi] / 15
for phase in np.linspace(start: 0, 2 * np.pi, nframe + 1)[:nframe]:
    print(phase)
    z = np.sin(r + phase)
    grid.points[:, -1] = z.ravel()
    # plotter.update_coordinates(pts, render=False)
    grid.point= pts
    plotter.write_frame()
```

# Part 2 | 课程内容

## » 可视化标签

- 通过add\_point\_labels添加标签

```
poly = pv.PolyData(np.random.rand(10, 3))
poly["mylabels"] = [f"Label ({', '.join(map(lambda x: f'{x:.3f}', tuple(poly.points[i])))})" for i in range(poly.n_points)]
```



# Part 2 | 课程内容

## » 可视化标签

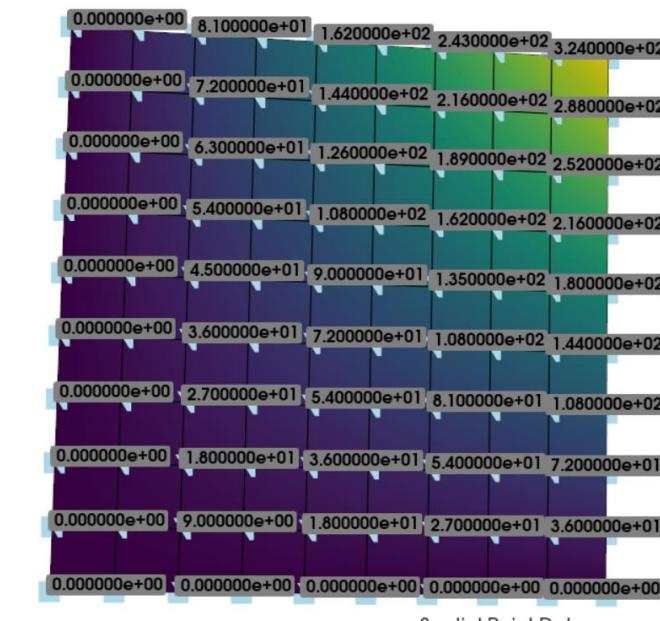
- 挑选其中 $x=0$ 的点的坐标作为标签

```
grid = pv.UnstructuredGrid(examples.hexbeamfile)
plotter = pv.Plotter()
plotter.add_mesh(grid, show_edges=True, color='tan')
points = grid.points
mask = points[:, 0] == 0 # 选择x=0的点云
plotter.add_point_labels(points[mask], points[mask].tolist(), point_size=20, font_size=16)
plotter.show()
```



- 以张量作为标签

```
mesh = examples.load_uniform().slice()
p = pv.Plotter()
p.add_mesh(mesh, scalars="Spatial Point Data", show_edges=True)
p.add_point_scalar_labels(mesh, labels="Spatial Point Data", point_size=20, font_size=16)
```



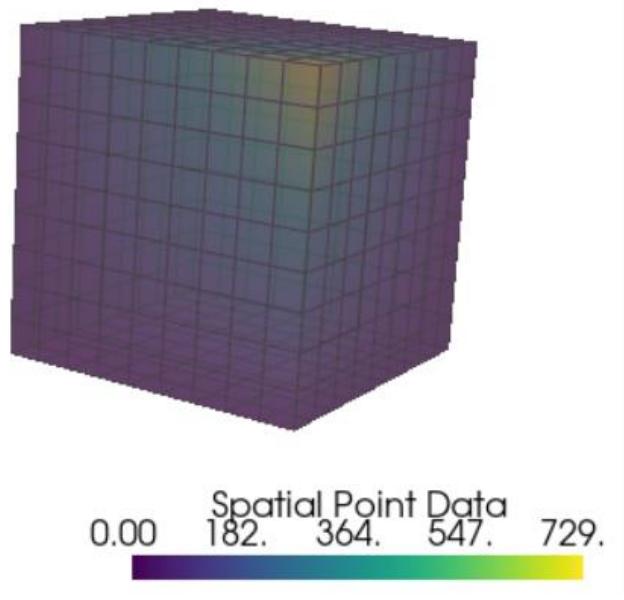
标签的自由度较高，对一帧点云和标签，将点云体素化之后 $0.5$ ，给每个体素打标签（众数—标签最多的）

# Part 2 | 课程内容

## » 过滤器 filter

```
threshold = dataset.threshold([100, 500])
```

- 过滤 scalar 范围内的点



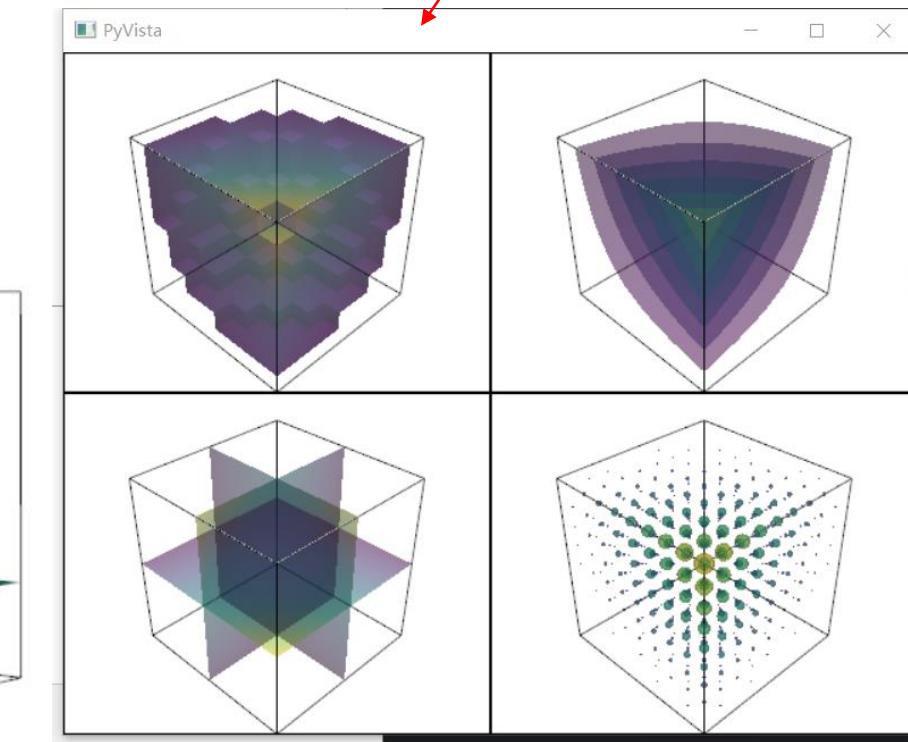
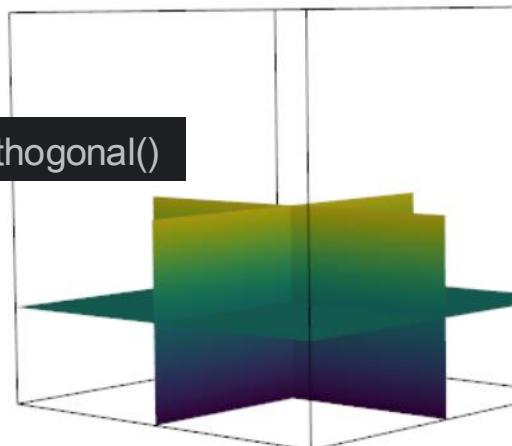
- 不同的过滤器效果 (后三种)

```
contours = dataset.contour(8)
slices = dataset.slice_orthogonal()
glyphs = dataset.glyph(factor=0.001, geom=pv.Sphere(), orient=False)
```

- Filter chain pipeline:

通过连续调用

```
result = dataset.threshold().elevation().clip(normal="z").slice_orthogonal()
```

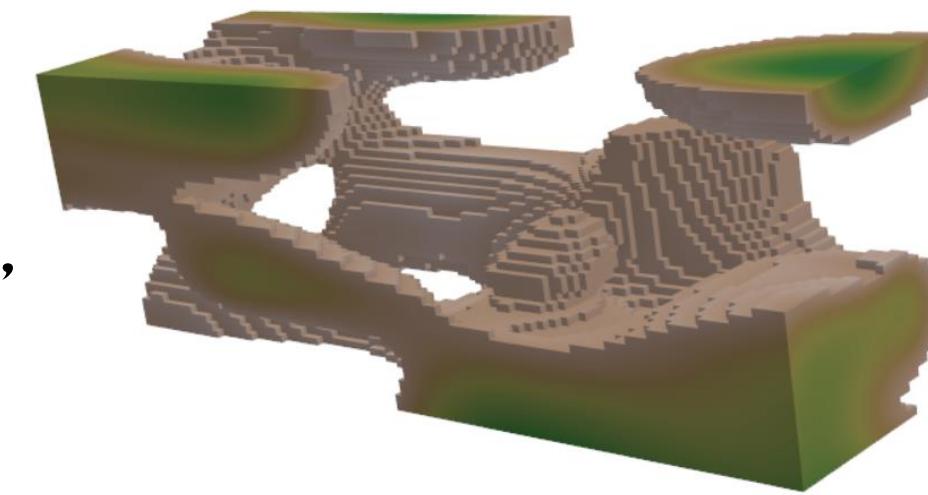


# Part 2 | 课程内容

## » Filter-Perlin noise

- [https://en.wikipedia.org/wiki/Perlin\\_noise](https://en.wikipedia.org/wiki/Perlin_noise)
- Perlin噪声 (Perlin noise, 又称为柏林噪声) 指由 [Ken Perlin](#)发明的自然噪声生成算法<sup>[1]</sup>, 具有在函数上的连续性, 并可在多次调用时给出一致的数值。在电子游戏中可以透过使用Perlin噪声生成具连续性的地形; 或是在艺术领域中使用Perlin噪声生成图样。
- 用随机产生的噪声图像和显然自然界物体的随机噪声有很大差别, 不够真实。1985年[Ken Perlin](#)指出<sup>[1]</sup>, 一个理想的噪声应该具有以下性质:
  - 对旋转具有统计不变性;
  - 能量在频谱上集中于一个窄带, 即: 图像是连续的, 高频分量受限;
  - 对变换具有统计不变性。

```
freq = (1, 2, 3) # Perlin 噪声的频率  
noise = pv.perlin_noise(amplitude=1, freq=freq, phase=(0, 0, 0))  
grid = pv.sample_function(noise, bounds=[0, 3.0, -0, 1.0, 0, 1.0], dim=(120, 40, 40))  
out = grid.threshold(0.02)
```



# Part 2 | 课程内容

## » 裁剪 clip

- 基于平面裁剪

```
PolyData (0x2b07d462e60)
```

```
N Cells: 1502  
N Points: 872
```

```
N Strips: 0  
X Bounds: -1.316e-01, 1.802e-01  
Y Bounds: -1.205e-01, 1.877e-01  
Z Bounds: -1.430e-01, 9.851e-02  
N Arrays: 1
```

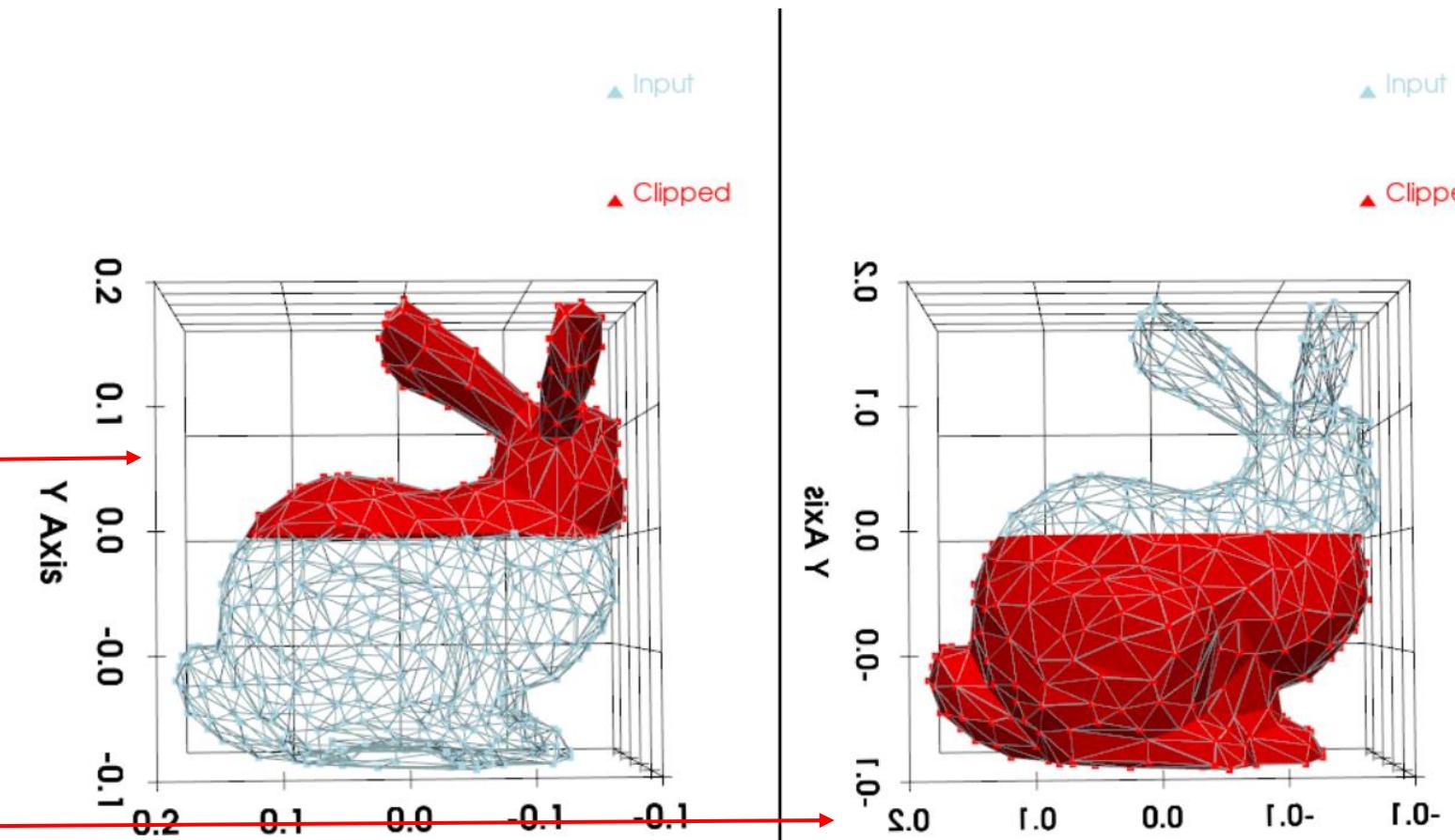
```
PolyData (0x2b07d462f20)
```

```
N Cells: 608  
N Points: 248  
N Strips: 0  
X Bounds: -1.316e-01, 1.326e-01  
Y Bounds: 3.360e-02, 1.877e-01  
Z Bounds: -1.430e-01, 8.721e-02  
N Arrays: 1
```

```
PolyData (0x2b07d462fe0)
```

```
N Cells: 1028  
N Points: 388
```

```
# 使用平面裁剪, 取y轴负方向的数据, True表示取反  
clipped = dataset.clip(normal: 'y', invert=False)  
inv_clipped = dataset.clip(normal: 'y', invert=True)
```

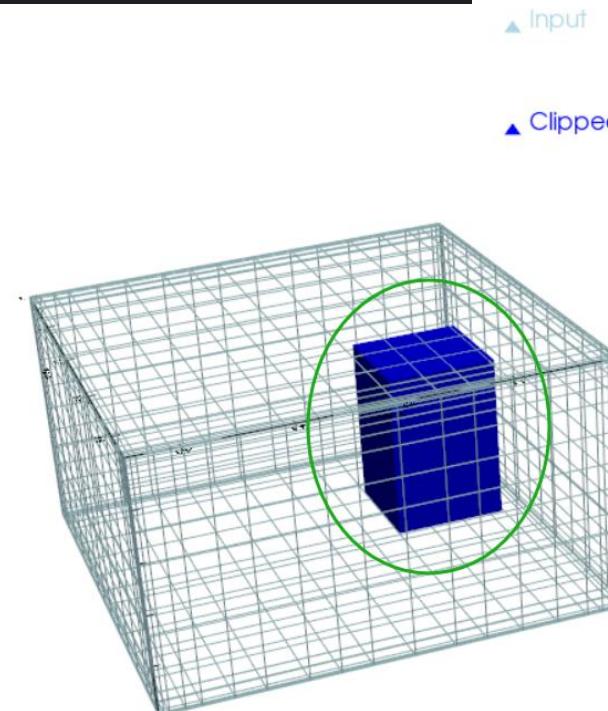
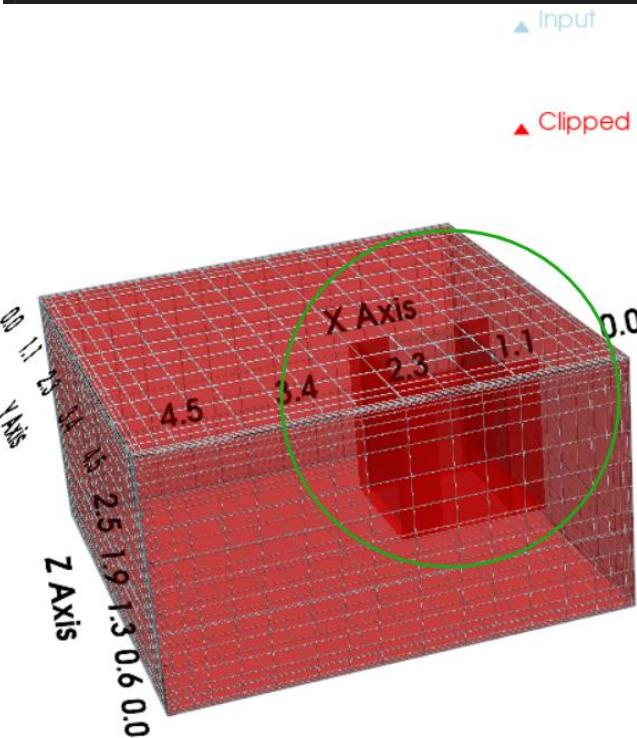


# Part 2 | 课程内容

## » 裁剪 clip\_box/ Cube+rotate()

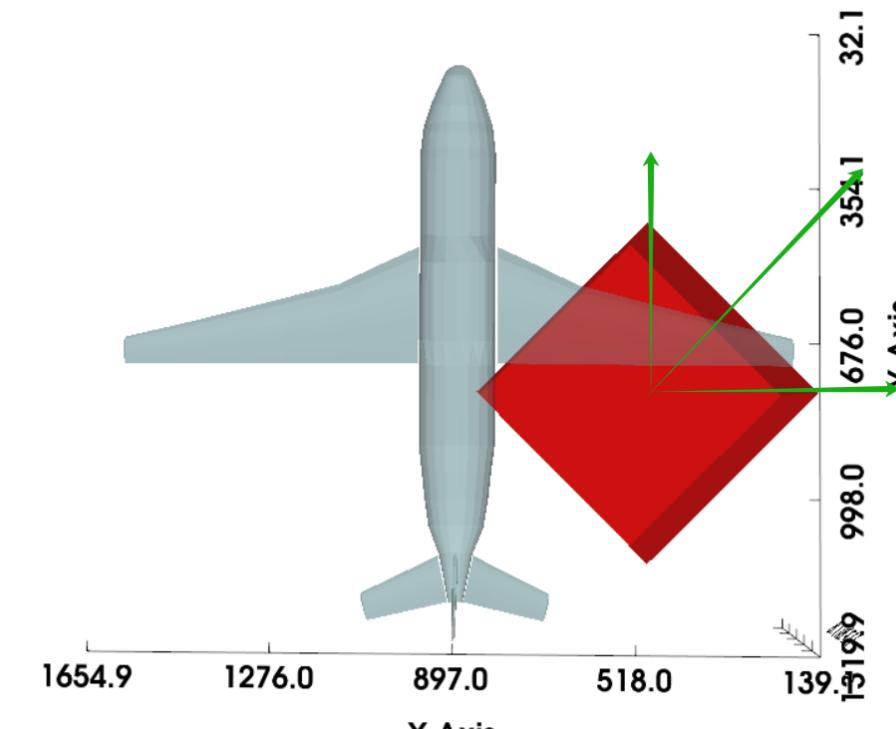
- 使用矩形进行裁剪

```
bounds = [1, 2, 3, 4, 1, 3] # xmin, xmax, ymin, ymax, zmin, zmax  
clipped = dataset.clip_box(bounds)  
inv_clipped = dataset.clip_box(bounds, invert=False)  
print(clipped)
```



```
# Use `pv.Box()` or `pv.Cube()` to create a region of interest  
# center, x_length, y_length, z_length  
roi = pv.Cube(center=(0.9e3, 0.2e3, mesh.center[2]), x_length=500, y_length=500, z_length=500)  
roi.rotate_z(angle: 45, inplace=True) # 旋转
```

- 旋转45度



# Part 2 | 课程内容

## » Clip时保持数据完整性

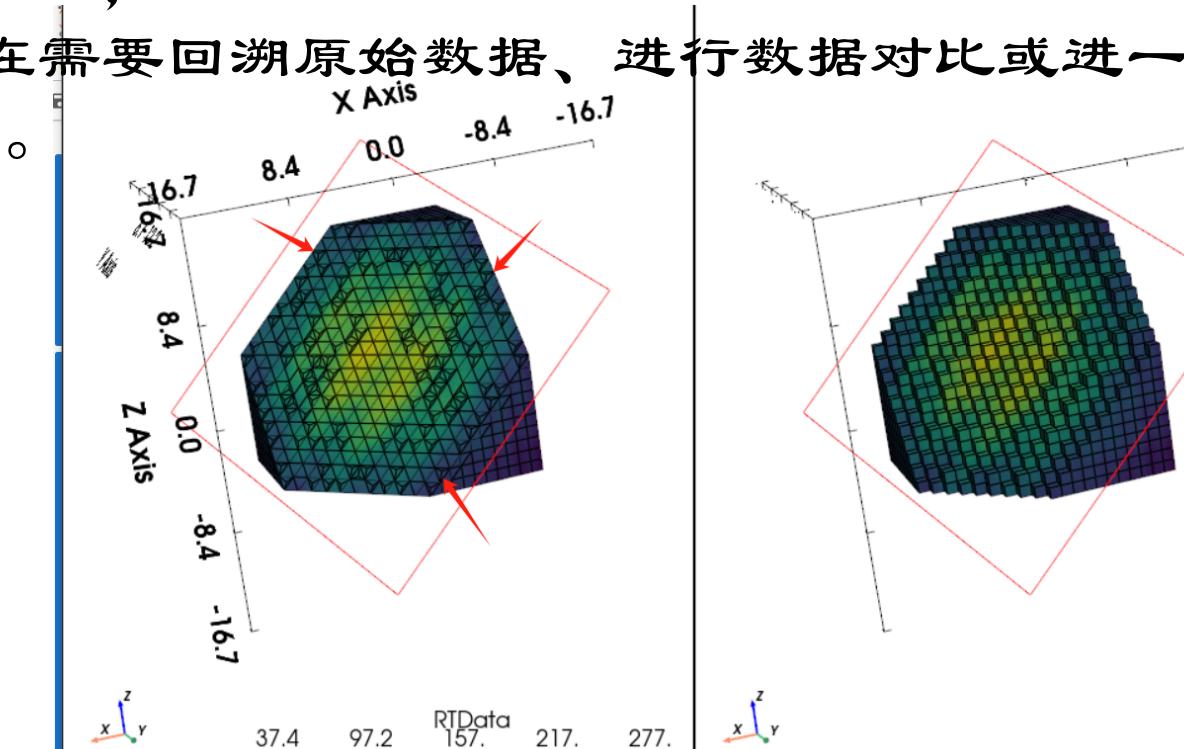
```
解释 | 添加注释 | X
def clip(
    self,
    normal='x',
    origin=None,
    invert=True,
    value=0.0,
    inplace=False,
    return_clipped=False,
    progress_bar=False,
    crinkle=False,
):
```

### 1. 保持数据完整性

当对三维网格数据进行裁剪操作时，通常会出现裁剪平面穿过某些单元格的情况。如果不使用 crinkle（即 crinkle=False），裁剪操作可能会将单元格切割成多个部分，这会破坏单元格的完整性。而将 crinkle 设置为 True 后，会提取完整的单元格，确保数据的完整性。

### 2. 追踪原始数据

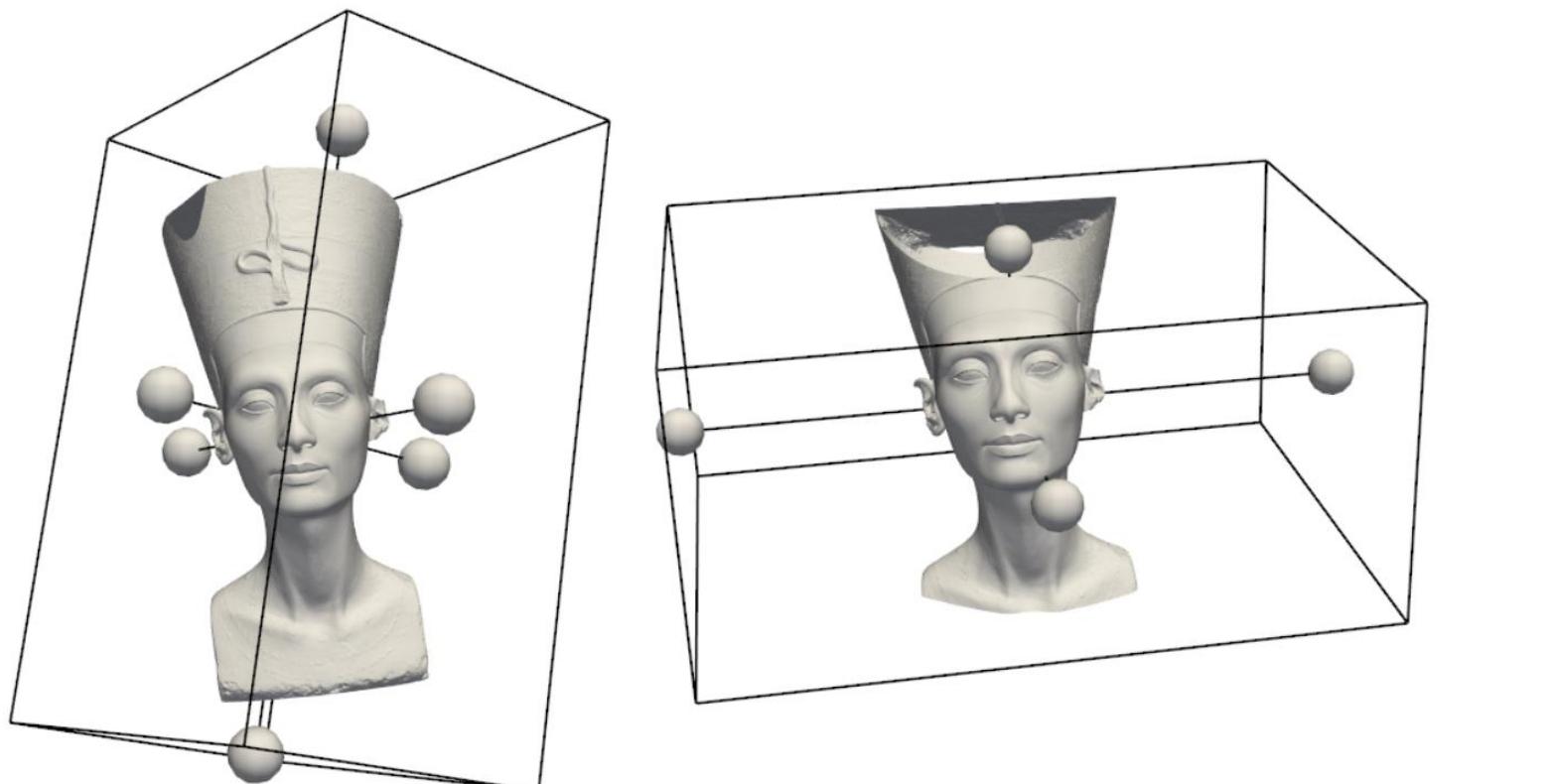
启用 crinkle 后，裁剪结果会在 cell\_data 中添加 "cell\_ids" 数组，该数组记录了裁剪后单元格在原始数据集中的 ID。这在需要回溯原始数据、进行数据对比或进一步分析时非常有用。



# Part 2 | 课程内容

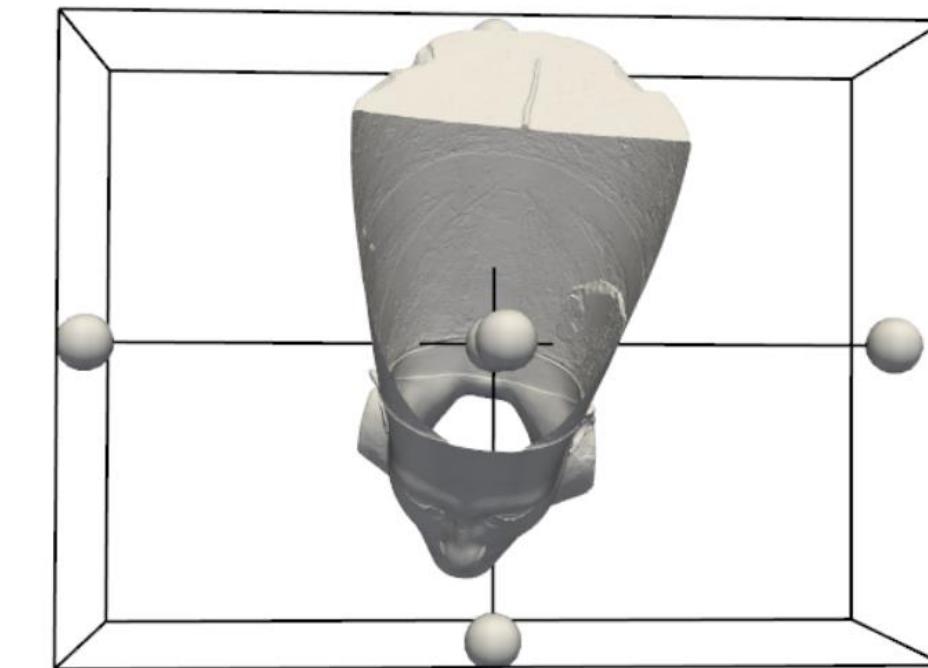
## » add\_mesh\_clip\_box

- # 选中 clip box 调整视角, 仅变换box
- # 选中 空白区域 调整视角, 变换两者, 渲染计算量增大。



### • 裁剪结果

```
[UnstructuredGrid (0x220cda202e0)
 N Cells: 1246663
 N Points: 1014571
 X Bounds: -1.194e+02, 1.194e+02
 Y Bounds: -1.813e+02, 1.813e+02
 Z Bounds: -2.473e+02, 2.473e+02
 N Arrays: 1]
```



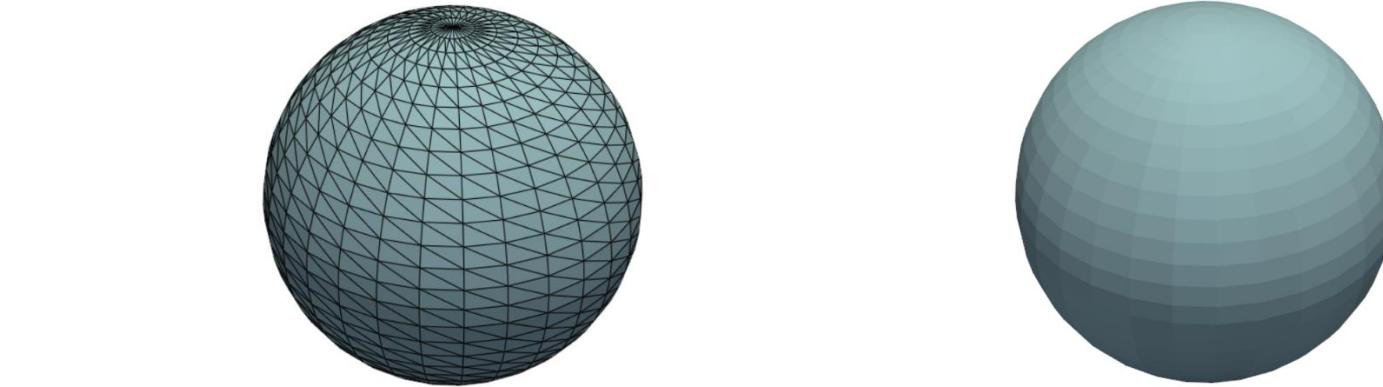
# Part 2 | 课程内容

## » checkbox

- 通过添加一个按钮来触发相应的事件，改变绘制结果。

```
def toggle_vis(flag) -> None: 1 usage
    actor.SetVisibility(flag)
解释 | 添加注释 | X
def toggle_wireframe(flag) -> None: 1 usage
    global actor
    p.renderer.remove_actor(actor)
    actor = p.add_mesh(mesh, show_edges=flag)
```

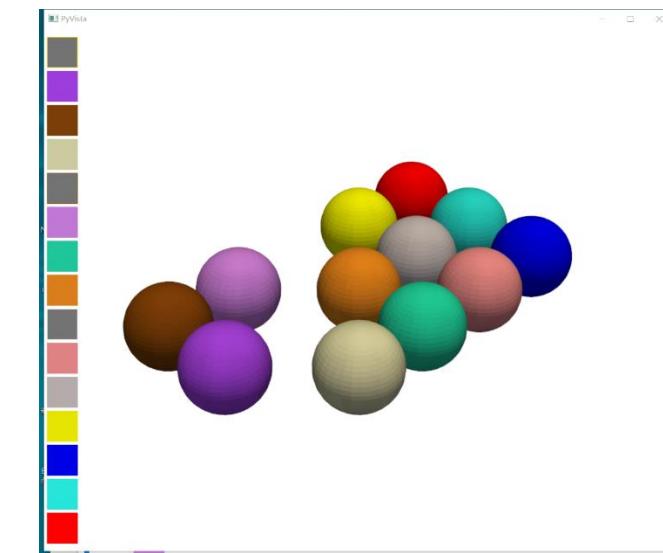
- 增加新按钮\*2，改变2项不同的mesh绘图属性。



Toggle Wireframe  
Toggle Visibility

Toggle Wireframe  
Toggle Visibility

- 控制多个mesh



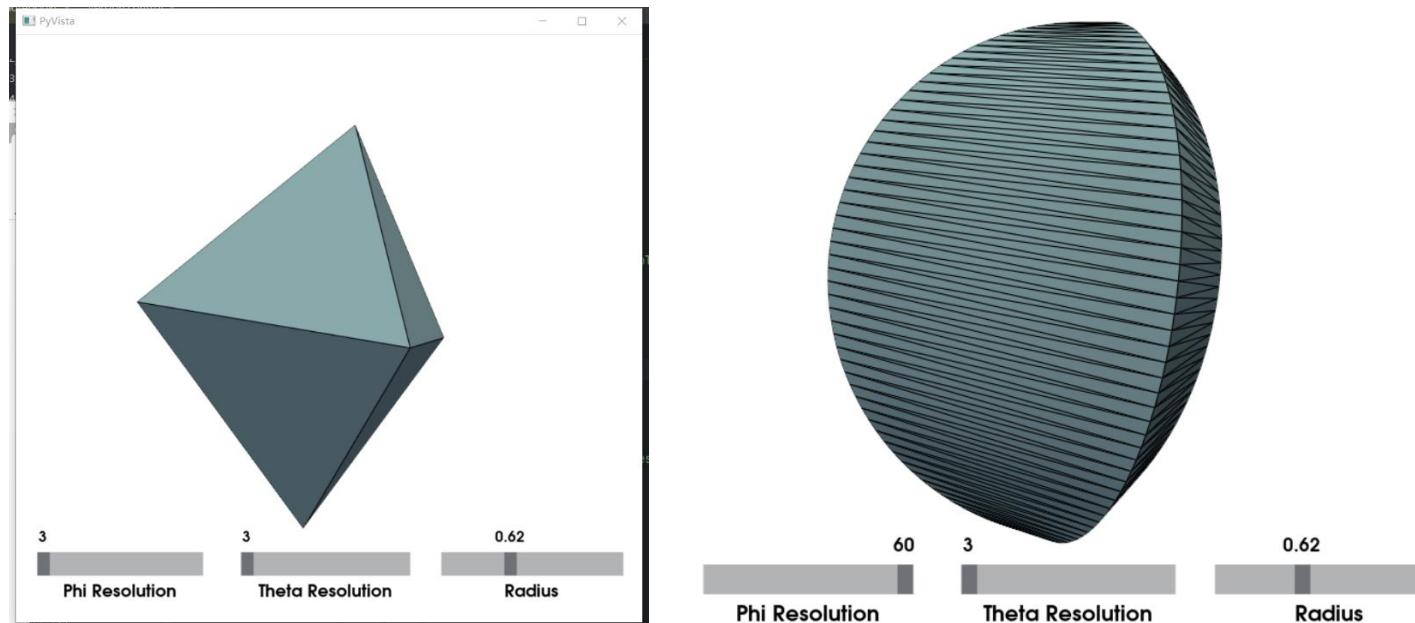
Toggle Wireframe  
Toggle Visibility

# Part 2 | 课程内容

## » 滑块调节slider组件

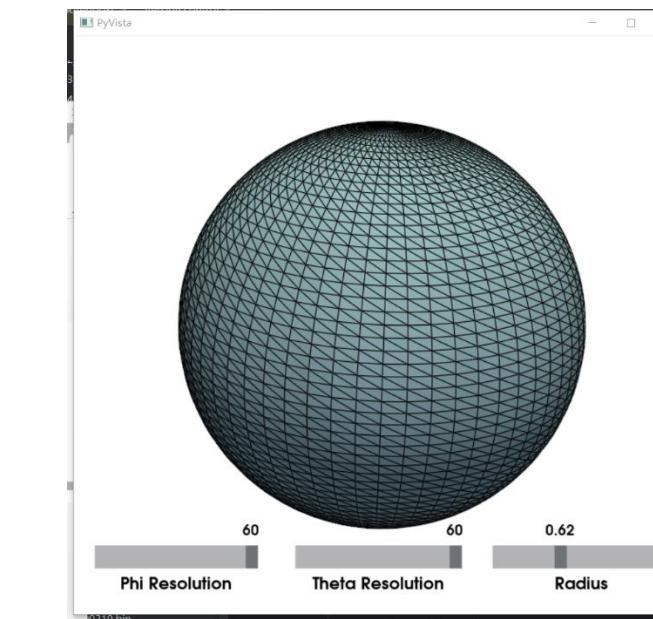
- 通过半径、经纬角控制球体分辨率。

```
self.output = mesh
self.kwargs = {
    "radius": 0.5,
    "theta_resolution": 30,
    "phi_resolution": 30,
}
```



- 配置滑块的参数调节区间。

```
p.add_slider_widget( #
    callback=lambda value: engine( param: "phi_resolution", int(value)),
    rng=[3, 60], # 范围
    value=30, # 初始值
    title="Phi Resolution",
    pointa=(0.025, 0.1), # slider的位置
    pointb=(0.31, 0.1), # slider的位置
    style="modern",
)
```

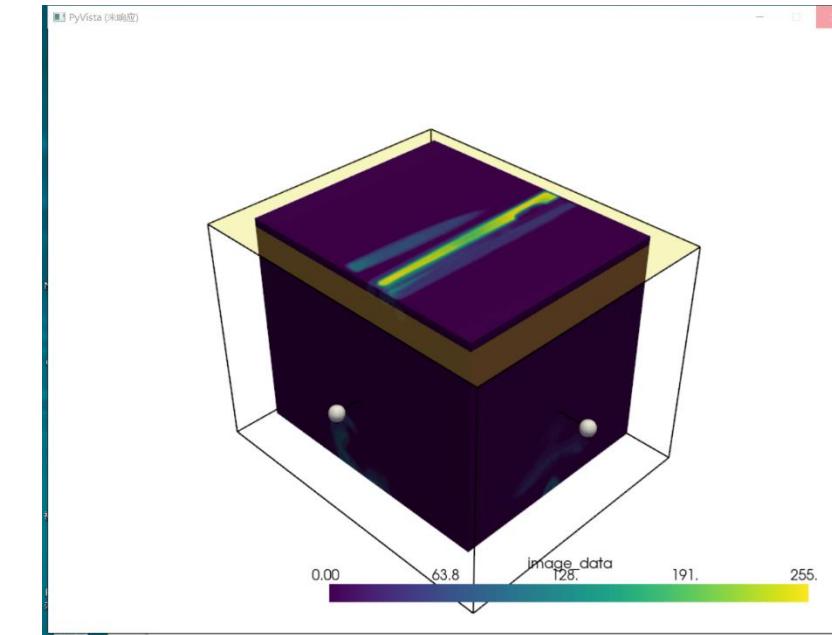


# Part 2 | 课程内容

## » 剖面add\_mesh\_clip\_box模拟医生看片

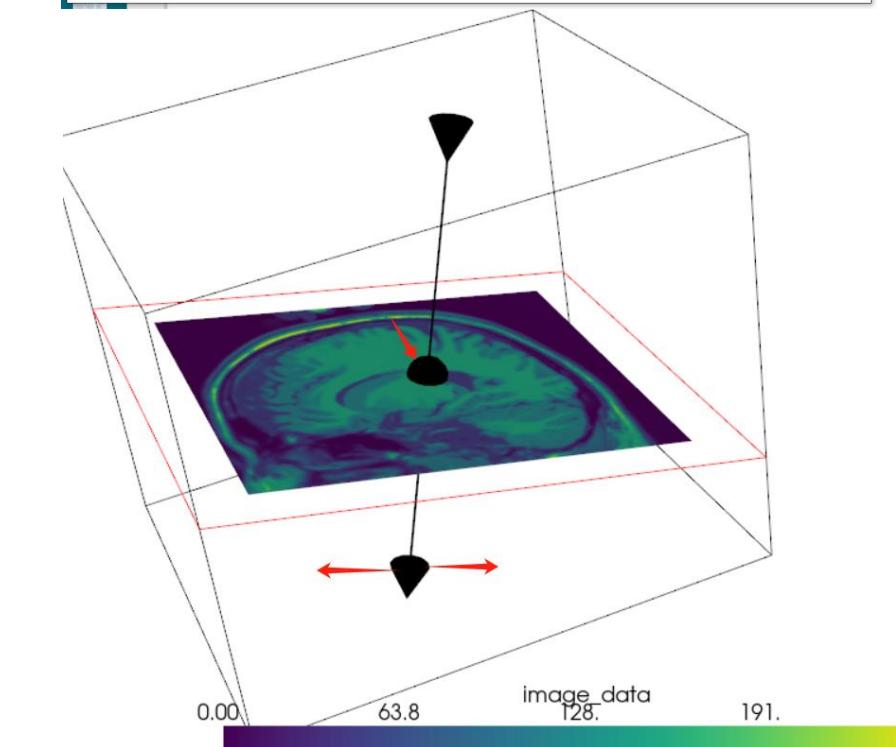
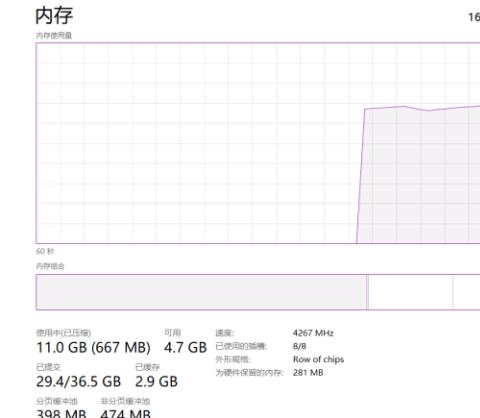
- 通过设定剖面，交互调节剖面，渲染剖面，适用于连续断层扫描目标（如CT），激光雷达等属于表面成像，非透射。

```
p = pv.Plotter()  
p.add_mesh_clip_box(vol)  
p.show()
```



- 通过设定剖面，交互调节剖面，只渲染剖面

```
p = pv.Plotter()  
p.add_mesh_slice(vol, normal='x')  
p.show()
```

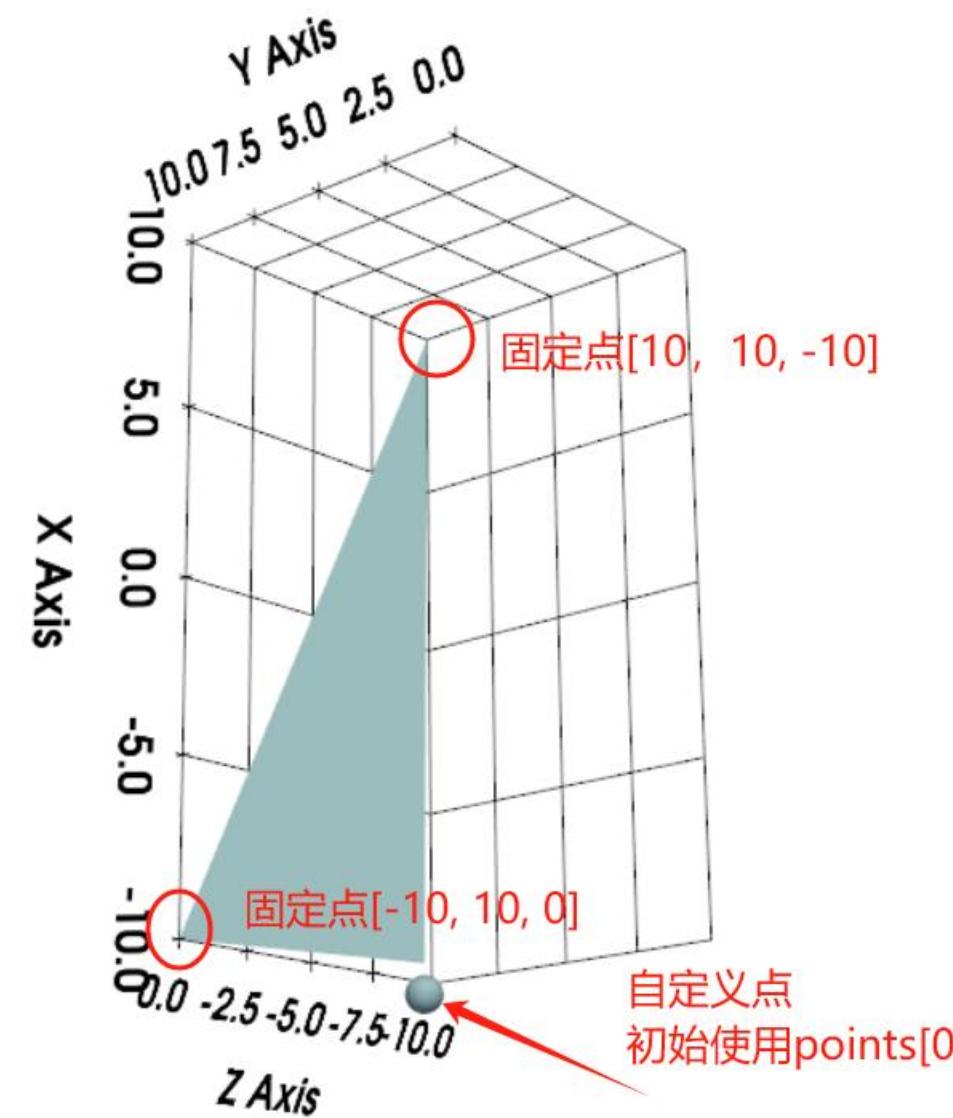


- 因为涉及复杂碰撞体积计算，对内存吃紧

# Part 2 | 课程内容

## » 自定义平面

- 定一个三角形，顶点的连接顺序。
- 通过回调函数在每次修改球时，刷新平面。

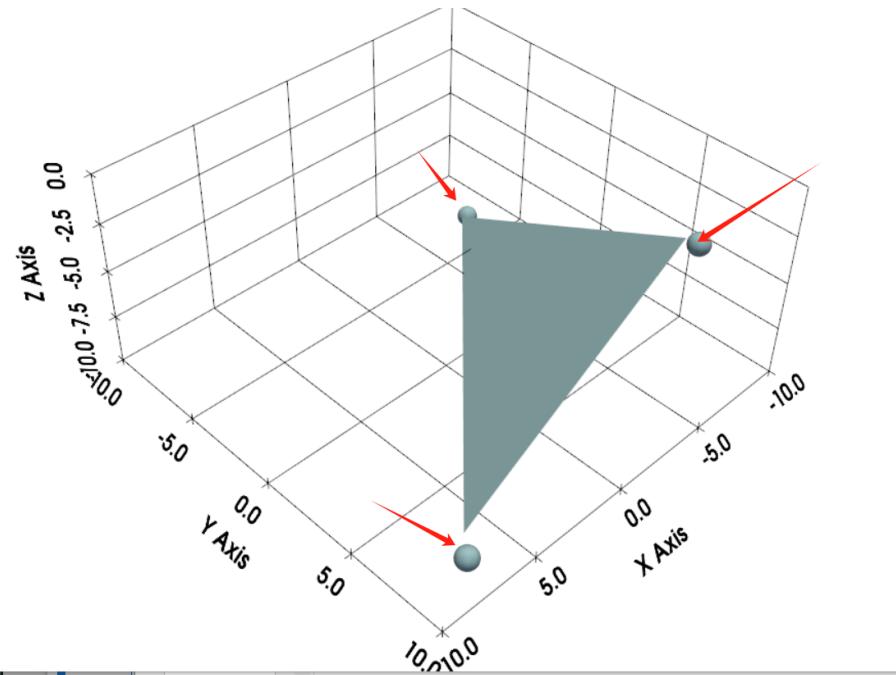


```
surf = pv.PolyData()  
surf.points = np.array([  
    [-10, -10, -10],  
    [10, 10, -10],  
    [-10, 10, 0],  
])  
surf.faces = np.array([3, 0, 1, 2])  
  
p = pv.Plotter()  
  
解释 | 添加注释 | X  
def callback(point) -> None:  
    surf.points[0] = point  
  
p.add_sphere_widget(callback)  
p.add_mesh(surf, color=True)
```

# Part 2 | 课程内容

## » 自定义平面

- 定一个三角形，顶点的连接顺序。
- 通过回调函数在每次修改球（三个）时，刷新平面。



```
surf = pv.PolyData()
surf.points = np.array([
    [-10, -10, -10],
    [10, 10, -10],
    [-10, 10, 0],
])
surf.faces = np.array([3, 0, 1, 2])

p = pv.Plotter()

解释 | 添加注释 | X
def callback(point) -> None:
    surf.points[0] = point

p.add_sphere_widget(callback)
p.add_mesh(surf, color=True)
```

def callback(point, i) -> None:

surf.points[i] = point

p.add\_sphere\_widget(callback, center=surf.points)

# Part 2 | 课程内容

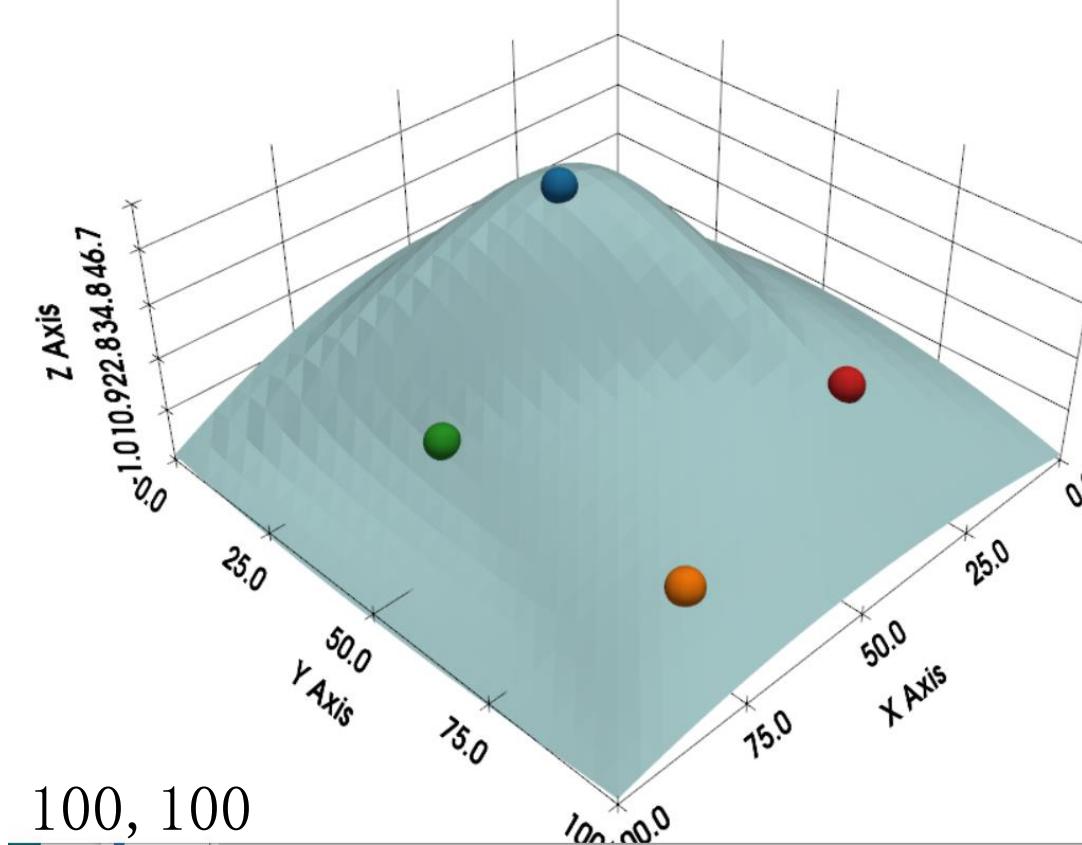
## » 自定义平面

- 类似调节贝塞尔曲线的控制点，通过平面mesh上的点调节平面曲率。
- 用到了scipy的插值函数对grid进行插值调节。

```
# Create a grid to interpolate to
xmin, xmax, ymin, ymax = 0, 100, 0, 100
x = np.linspace(xmin, xmax, num=25)
y = np.linspace(ymin, ymax, num=25)
xx, yy, zz = np.meshgrid(*xi: x, y, [0])
```



- 平面范围0, 0 - 100, 100



```
def update_surface(point, i) -> None: 1 usage
    points[i] = point
    tp = np.vstack((points, boundaries))
    # @ points @ data values @ Points at which to interpolate data. @ 插值算法
    zz = griddata(tp[:, 0:2], tp[:, 2], xi: (xx[:, :, 0], yy[:, :, 0]), method="cubic")
    surf.points[:, -1] = zz.ravel(order="F")
```



- 平面更新基于控制点坐标  
进行插值

# 开始实验

<https://github.com/hahakid/3D-point-cloud-processing-and-visualization-practice/tree/main/lecture5>



北京航空航天大学  
BEIHANG UNIVERSITY