图像处理与可视化 Homework 08

学号: 21307140051
姓名: 雍崔扬

Data

本次作业我们使用以下数据:

- ① elearning 心脏 CT 图像 (`image_lr.nii`)
    - ② OASIS Brains 数据集 (Open Access Series of Imaging Studies)
      这是一个在阿尔茨海默病及其他与年龄相关的认知衰退研究中广泛使用的神经影像学数据集. 该数据集提供了全面的神经影像数据，包括结构性 MRI 扫描，以及参与者的临床和人口统计学信息.
      我们使用 OASIS-1 RAW DATA Disc 1 的 0001 号数据的第 1 MAPAGE 转换成的 NIFTI 格式图像.
      (`OAS1_0001_mpr-1.nii` 和 `segmented_brain.nii`)

Problem 1

阅读了解 VTK (The Visualization Toolkit)，学习在某个编程环境下调用 VTK 库进行可视化.
调用可视化渲染引擎库 VTK，实现三维体数据完整的渲染过程 (如光照模型，颜色设置等)

- ① 等值面渲染
- ② 实现体渲染

1.1 等值面渲染

我们使用**光线遍历 (Ray Traversal)** 的**最大强度投影 (MIP)** (直接体绘制的一种变体)
对光线在整个路径上的强度值取最大值，用以突出强度最高的区域.
Python 代码:

```
option = 1
smoothing_enabled = False # Set this to False to disable smoothing

# Load the NIfTI image and extract data
if option == 1:
    image = nib.load('./nii_files/image_lr.nii')
    iso_value = 140
    smoothing_iterations = 1000
    # shape: (124, 124, 73)
    # spacing: (1.5, 1.5, 1.5) (mm)
elif option == 2:
    image = nib.load('./nii_files/segmented_brain.nii')
    iso_value = 500
    smoothing_iterations = 300
    # shape: (256, 256, 20)
    # spacing: (1, 1, 1.25) (mm)
elif option == 3:
    image = nib.load('./nii_files/OAS1_0001_mpr-1.nii')
    iso_value = 500
```

```
    smoothing_iterations = 100
    # shape: (256, 256, 128)
    # spacing: (1, 1, 1.25) (mm)

data = np.array(image.get_fdata(), dtype=np.float64)
print("Min value:", np.min(data))
print("Max value:", np.max(data))
dims = image.shape
spacing = tuple(image.header['pixdim'][1:4])

# Create vtkImageData object and set its properties
image = vtk.vtkImageData()
image.SetDimensions(dims)
image.SetSpacing(spacing)
image.SetOrigin(0, 0, 0)
image.AllocateScalars(vtk.VTK_DOUBLE, 1)  # Allocate scalar data (1 component)

# Convert the data to a 1D array that matches the correct order
flat_data = np.array(data, dtype=np.float64).transpose(2, 1, 0).ravel()  #
Transpose to (z, y, x) order and flatten

# Set the scalar values for the vtkImageData
image.GetPointData().GetScalars().SetVoidArray(flat_data, flat_data.size, 1)

# Marching cubes to extract the isosurface
extractor = vtk.vtkMarchingCubes()
extractor.SetInputData(image)
extractor.SetValue(0, iso_value)  # Set the contour value for the isosurface

# Create a stripper to connect triangles into strips
stripper = vtk.vtkStripper()
stripper.SetInputConnection(extractor.GetOutputPort())

# Map the polydata to an actor
mapper = vtk.vtkPolyDataMapper()

# Optional smoothing step
if smoothing_enabled:
    smoother = vtk.vtkSmoothPolyDataFilter()
    smoother.SetInputConnection(stripper.GetOutputPort())
    smoother.SetNumberOfIterations(smoothing_iterations)  # Set number of
iterations for smoothing
    smoother.Update()
    poly_data = smoother.GetOutput()  # Get the smoothed polydata
    mapper.SetInputData(poly_data)
else:
    mapper.SetInputConnection(stripper.GetOutputPort())  # Without smoothing

actor = vtk.vtkActor()
actor.SetMapper(mapper)
actor.GetProperty().SetColor(1, 1, 0.1)  # yellow
actor.GetProperty().SetOpacity(0.95)  # Set opacity
actor.GetProperty().SetAmbient(0.05)  # Set ambient lighting
actor.GetProperty().SetDiffuse(0.5)  # Set diffuse lighting
actor.GetProperty().SetSpecular(1.0)  # Set specular lighting

# Set up the rendering window and interactor
renderer = vtk.vtkRenderer()
```

```
renderer.SetBackground(1, 1, 1)  # White background
renderer.AddActor(actor)

render_window = vtk.vtkRenderWindow()
render_window.AddRenderer(renderer)
render_window.SetSize(750, 750)

interactor = vtk.vtkRenderWindowInteractor()
interactor.SetRenderWindow(render_window)
interactor.Initialize()

# Start rendering
render_window.Render()
interactor.Start()
```

运行结果:

- ① `option == 1` 对应心脏 CT 图像的面渲染:

- ② `option == 2` 对应大脑 CT 图像冠状切片的面渲染:

- ③ `option == 3` 对应大脑 CT 图像的面渲染:

1.2 体绘制

我们使用**光线遍历 (Ray Traversal)** 的方法，模拟从观察者出发发射光线穿过三维体数据.
每条光线与体数据的体素相交，计算每个交点的颜色和透明度，并根据传输函数进行映射.
最后所有这些信息沿光线方向进行累积，生成最终的体积图像.
Python 代码:

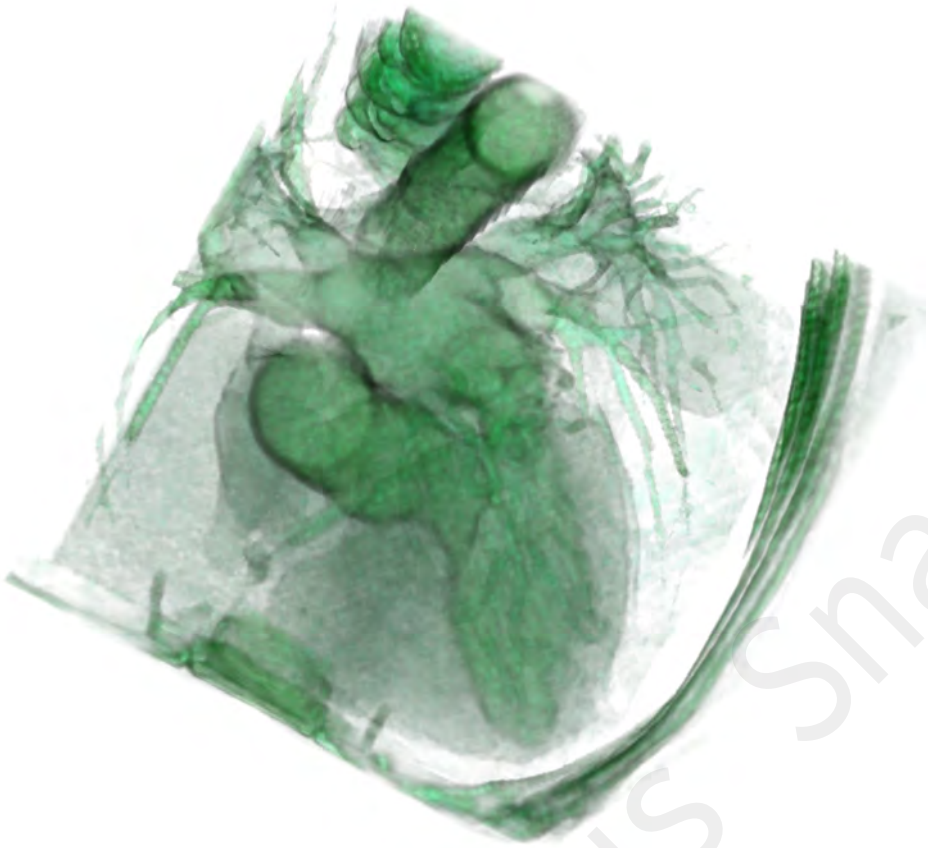
```
option = 1

# Load the NIfTI image and extract data
if option == 1:
    image = nib.load('./nii_files/image_lr.nii')
    opacity = [50, 400]
    color = [0, 400]
    gradient = [150, 600]
```

```
    # shape: (124, 124, 73)
    # spacing: (1.5, 1.5, 1.5) (mm)
elif option == 2:
    image = nib.load('./nii_files/segmented_brain.nii')
    opacity = [500, 1000]
    color = [0, 1000]
    gradient = [500, 1000]
    # shape: (256, 256, 10)
    # spacing: (1, 1, 1.25) (mm)
elif option == 3:
    image = nib.load('./nii_files/OAS1_0001_mpr-1.nii')
    opacity = [900, 1000]
    color = [0, 1000]
    gradient = [800, 1000]
    # shape: (256, 256, 128)
    # spacing: (1, 1, 1.25) (mm)

# Get image dimensions and spacing
data = np.array(image.get_fdata(), dtype=np.float64)
dims = image.shape
spacing = tuple(image.header['pixdim'][1:4])

# Create vtkImageData object and set its properties
vtk_image = vtk.vtkImageData()
vtk_image.SetDimensions(dims)
vtk_image.SetSpacing(spacing)
vtk_image.SetOrigin(0, 0, 0)
vtk_image.AllocateScalars(vtk.VTK_DOUBLE, 1)

# Flatten the data to match VTK's expected input order (z, y, x)
flat_data = np.array(data, dtype=np.float64).transpose(2, 1, 0).ravel()

# Set the scalar values for the vtkImageData
vtk_image.GetPointData().GetScalars().SetVoidArray(flat_data, flat_data.size, 1)

# Set up transfer functions (opacity, color, gradient opacity)
opacity_transfer_function = vtk.vtkPiecewiseFunction()
opacity_transfer_function.AddPoint(0, 0.0)
opacity_transfer_function.AddSegment(opacity[0], 0.3, opacity[1], 0.5)
opacity_transfer_function.ClampingOff()

color_transfer_function = vtk.vtkColorTransferFunction()
color_transfer_function.AddRGBSegment(0, 0.0, 0.0, 0.0, 20, 0.2, 0.2, 0.2)
color_transfer_function.AddRGBSegment(color[0], 0.1, 0.1, 0, color[1], 1, 1, 0)

gradient_transfer_function = vtk.vtkPiecewiseFunction()
gradient_transfer_function.AddPoint(0, 0.0)
gradient_transfer_function.AddSegment(gradient[0], 0.1, gradient[1], 0.3)

# Set volume properties (how the data will be visualized)
volume_property = vtk.vtkVolumeProperty()
volume_property.SetScalarOpacity(opacity_transfer_function)
volume_property.SetColor(color_transfer_function)
volume_property.SetGradientOpacity(gradient_transfer_function)
volume_property.ShadeOn()
volume_property.SetInterpolationTypeToLinear()
volume_property.SetAmbient(1)
volume_property.SetDiffuse(0.9)
```

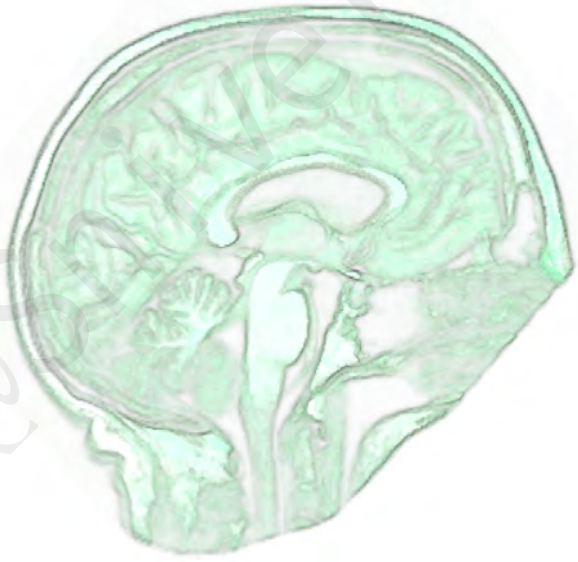
```
volume_property.SetSpecular(0.8)
volume_property.SetSpecularPower(10)

# Set up volume mapper and ray casting
volume_mapper = vtk.vtkFixedPointVolumeRayCastMapper()
volume_mapper.SetInputData(vtk_image)
volume_mapper.SetImageSampleDistance(5.0)

# Create a volume and apply the mapper and properties
volume = vtk.vtkVolume()
volume.SetMapper(volume_mapper)
volume.SetProperty(volume_property)

# Set up the renderer, render window, and interactor
renderer = vtk.vtkRenderer()
renderer.SetBackground(1, 1, 1)  # White background
renderer.AddVolume(volume)

render_window = vtk.vtkRenderWindow()
render_window.AddRenderer(renderer)
render_window.SetSize(750, 750)

interactor = vtk.vtkRenderWindowInteractor()
interactor.SetRenderWindow(render_window)

# Add light to the scene
light = vtk.vtkLight()
light.SetColor(0, 1, 1)
renderer.AddLight(light)

# Render the scene
render_window.Render()
interactor.Initialize()
interactor.Start()
```

运行结果:

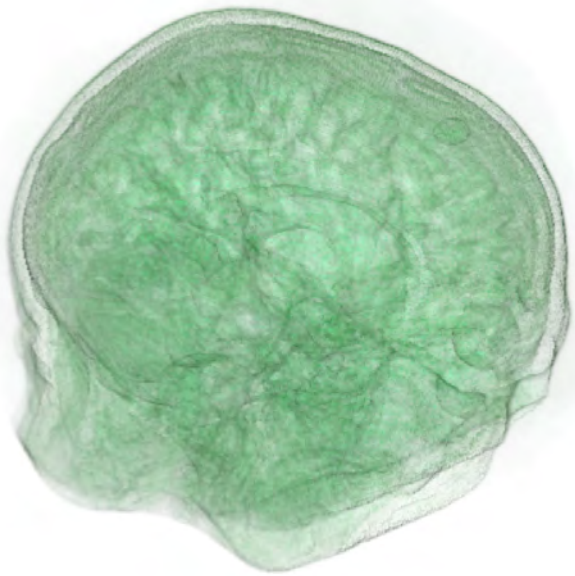
- ① `option == 1` 对应心脏 CT 图像的体绘制:



- ② `option == 2` 对应大脑 CT 图像冠状切片的体绘制:



- ③ `option == 3` 对应大脑 CT 图像的体绘制:



Problem 2

消除心脏 CT 图像 (`image_lr.nii`) 等值面渲染结果中的碎片化的面单元.

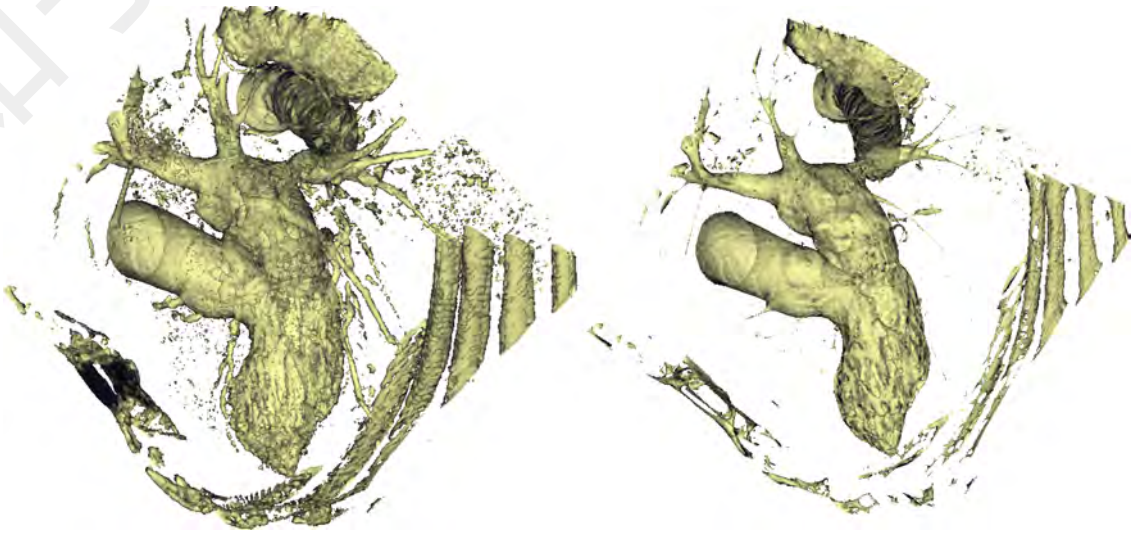
Solution:
我们使用 `vtkSmoothPolyDataFilter` 消除等值面渲染结果中的碎片化的面单元.
该平滑滤波器通过对每个顶点的邻域进行迭代平均处理来平滑网格.
其基本思路如下:

- **① 顶点平滑:**
  对每个顶点，根据其邻域顶点的坐标计算一个加权平均值.
  通过这种方式，顶点会向其邻居的几何中心移动，从而平滑整个表面.
  默认使用 Laplace 平滑.
- **② 加权平均:**
  每个顶点的新的位置是根据其邻域顶点的位置来计算的.
  邻域顶点的贡献是加权的，距离越近的邻居权重越大.

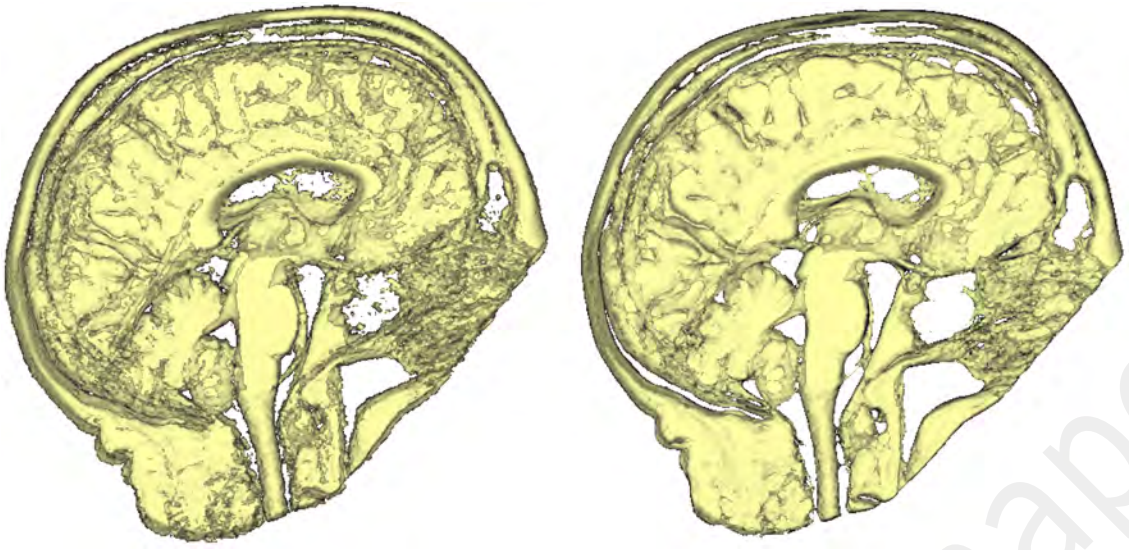
不断迭代以完成平滑过程.

运行结果:
设置 `smoothing_enabled = True` 消除碎片化的面单元.

- ① `option == 1` 对应心脏 CT 图像的面渲染:



- ② `option == 2` 对应大脑 CT 图像冠状切片的面渲染:



- ③ `option == 3` 对应大脑 CT 图像的面渲染: