Data Visualization - Homework 6 - Report

邓淇升 大数据学院 16307110232

1 阅读了解 VTK，学习 Python 环境下调用 VTK 库进行可视化。

已仔细阅读 VTK 相关资料，已掌握调用 VTK 库进行数据可视化的技术。

2 调用 VTK 库实现三维体数据的完整渲染过程（如光照模型，颜色设置等）。需要实现的渲染过程包括：(1) 等值面渲染；(2) 体渲染。

- **Python script code:**

(1) Isosurface Render:

2_isosurface_render.py

(2) Volume Render:

2_volume_render.py

- **Input & Output:**

(1) Isosurface Render:

Input:

brain.nii.gz

Output:

isosurface_render.png

isosurface_render.mp4（后期制作）

(2) Volume Render:

Input:

brain.nii.gz

Output:

volume_render.png

volume_render.mp4（后期制作）

- **Description:**

数据说明：

实验使用的体数据储存于文件 brain.nii.gz 中。体数据的内容为脑部 MRI 扫描图像，由 124 张大小为 255 * 255 的二维图像组成，采样间距为 0.9375 * 0.9375 * 1.5。三维图像重建中需要复现的部位为脑部的大脑皮层、大脑、小脑、脑干和胼胝体。为了展示三维图像重建效果，可视化处理中已将胼胝体脑区设置为高亮的颜色。

渲染过程：

等值面渲染中重建区域分为三个部分。外部为大脑皮层，等值面值为 10，光线模型为 0.3*环境光+0.6*散射光+0.8*反射光，不透明率为 0.2，颜色为浅红色；中部为大脑、小脑及脑干，等值面值为 500，光线模型为 0.4*环境光+0.6*散射光+0.6*反射光，反射光强为 1000，不透明率为 1.0，颜色为白色；内部为胼胝体，等值面值为 380，光线模型为 0.4*环境光+0.6*散射光+0.5*反射光，不透明率为 1.0，颜色为紫色。

体渲染中重建区域分为两个部分。外部为大脑皮层、大脑、小脑及脑干，光线模型为 1.0*环境光+1.0*散射光+1.0*反射光，反射光强为 100，颜色为浅蓝色；内部为胼胝体，光线模型为 0.8*环境光+0.8*散射光+0.8*反射光，反射光强为 10，颜色为红色。传输函数请查看代码。

- **Source Code:**

等值面渲染：

```
import os  # 文件

import vtk  # 可视化

########
# PATH #
########

# 注意：所有路径必须为全英文！！！
HOME_PATH = os.getcwd()  # 主目录
DATA_PATH = os.path.join(HOME_PATH, 'data')  # 数据目录
RESULT_PATH = os.path.join(HOME_PATH, 'result')  # 结果目录

##########
# SOURCE #
##########

source = vtk.vtkNIFTIImageReader()  # 数据源
source.SetFileName(os.path.join(DATA_PATH, 'brain.nii.gz'))  # 读取体数据

##########
# FILTER #
##########
```

```
########## inner ##########

# VOI
source_inner = vtk.vtkExtractVOI()  # 过滤器：感兴趣区域
source_inner.SetInputConnection(source.GetOutputPort())  # 读取数据源
source_inner.SetVOI(95, 160, 75, 180, 47, 76)  # 划定区域
source_inner.SetSampleRate(1, 1, 1)  # 设定采样率

# EXTRACTOR
filter_extractor_inner = vtk.vtkMarchingCubes()  # 过滤器：三角面片
filter_extractor_inner.SetInputConnection(source_inner.GetOutputPort())  # 读取数据源
filter_extractor_inner.ComputeNormalsOn()  # 计算等值面法向量以增强渲染效果
filter_extractor_inner.SetValue(0, 380)  # 设定内部等值面

# STRIPPER
filter_isosurface_inner = vtk.vtkStripper()  # 过滤器：等值面
filter_isosurface_inner.SetInputConnection(filter_extractor_inner.GetOutputPort())  # 读取过滤器：三角面片

########## median ##########

# EXTRACTOR
filter_extractor_median = vtk.vtkMarchingCubes()  # 过滤器：三角面片
filter_extractor_median.SetInputConnection(source.GetOutputPort())  # 读取数据源
filter_extractor_median.ComputeNormalsOn()  # 计算等值面法向量以增强渲染效果
filter_extractor_median.SetValue(0, 500)  # 设定中部等值面

# STRIPPER
filter_isosurface_median = vtk.vtkStripper()  # 过滤器：等值面
filter_isosurface_median.SetInputConnection(filter_extractor_median.GetOutputPort())  # 读取过滤器：三角面片

# CLIPPER
filter_plane_median = vtk.vtkPlane()  # 过滤器：切割面
filter_plane_median.SetOrigin(140, 127.5, 61.5)  # 设定原点
filter_plane_median.SetNormal(-1, 0, 0)  # 设定法向

########## outer ##########

# EXTRACTOR
filter_extractor_outer = vtk.vtkMarchingCubes()  # 过滤器：三角面片
filter_extractor_outer.SetInputConnection(source.GetOutputPort())  # 读取数据源
filter_extractor_outer.ComputeNormalsOn()  # 计算等值面法向量以增强渲染效果
filter_extractor_outer.SetValue(0, 10)  # 设定外部等值面

# STRIPPER
filter_isosurface_outer = vtk.vtkStripper()  # 过滤器：等值面
filter_isosurface_outer.SetInputConnection(filter_extractor_outer.GetOutputPort())  # 读取过滤器：三角面片

########## outline ##########

# OUTLINE
filter_outline = vtk.vtkOutlineFilter()  # 过滤器：边框
filter_outline.SetInputConnection(source.GetOutputPort())  # 读取数据源

##########
# Mapper #
##########

# STRIPPER: inner
mapper_isosurface_inner = vtk.vtkPolyDataMapper()  # 映射器：等值面
mapper_isosurface_inner.SetInputConnection(filter_isosurface_inner.GetOutputPort())  # 读取过滤器：等值面
mapper_isosurface_inner.ScalarVisibilityOff()  # 关闭颜色映射关系

# STRIPPER: median
mapper_isosurface_median = vtk.vtkPolyDataMapper()  # 映射器：等值面
mapper_isosurface_median.SetInputConnection(filter_isosurface_median.GetOutputPort())  # 读取过滤器：等值面
mapper_isosurface_median.ScalarVisibilityOff()  # 关闭颜色映射关系
mapper_isosurface_median.AddClippingPlane(filter_plane_median)  # 切割等值面

# STRIPPER: outer
mapper_isosurface_outer = vtk.vtkPolyDataMapper()  # 映射器：等值面
mapper_isosurface_outer.SetInputConnection(filter_isosurface_outer.GetOutputPort())  # 读取过滤器：等值面
mapper_isosurface_outer.ScalarVisibilityOff()  # 关闭颜色映射关系
```

```
# OUTLINE
mapper_outline = vtk.vtkPolyDataMapper()  # 映射器：边框
mapper_outline.SetInputConnection(filter_outline.GetOutputPort())  # 读取过滤器：边框

#########
# ACTOR #
#########

# STRIPPER: inner
actor_isosurface_inner = vtk.vtkActor()  # 演员：等值面
actor_isosurface_inner.SetMapper(mapper_isosurface_inner)  # 读取映射器：等值面
actor_isosurface_inner.GetProperty().SetColor(0.5, 0.5, 1.0)  # 设定颜色
actor_isosurface_inner.GetProperty().SetAmbient(0.4)  # 设定环境光系数
actor_isosurface_inner.GetProperty().SetDiffuse(0.6)  # 设定散射光系数
actor_isosurface_inner.GetProperty().SetSpecular(0.5)  # 设定反射光系数
actor_isosurface_inner.GetProperty().SetOpacity(1.0)  # 设定透射光系数
actor_isosurface_inner.RotateX(90)  # 设定X轴旋转
actor_isosurface_inner.SetScale(1.2, 1.2, 1.2)  # 设定缩放

# STRIPPER: median
actor_isosurface_median = vtk.vtkActor()  # 演员：等值面
actor_isosurface_median.SetMapper(mapper_isosurface_median)  # 读取映射器：等值面
actor_isosurface_median.GetProperty().SetColor(0.95, 0.95, 0.95)  # 设定颜色
actor_isosurface_median.GetProperty().SetAmbient(0.4)  # 设定环境光系数
actor_isosurface_median.GetProperty().SetDiffuse(0.6)  # 设定散射光系数
actor_isosurface_median.GetProperty().SetSpecular(0.6)  # 设定反射光系数
actor_isosurface_median.GetProperty().SetSpecularPower(1000)  # 设定反射光强
actor_isosurface_median.GetProperty().SetOpacity(1.0)  # 设定透射光系数
actor_isosurface_median.RotateX(90)  # 设定X轴旋转
actor_isosurface_median.SetScale(1.2, 1.2, 1.2)  # 设定缩放

# STRIPPER: outer
actor_isosurface_outer = vtk.vtkActor()  # 演员：等值面
actor_isosurface_outer.SetMapper(mapper_isosurface_outer)  # 读取映射器：等值面
actor_isosurface_outer.GetProperty().SetColor(1.0, 0.4, 0.4)  # 设定颜色
actor_isosurface_outer.GetProperty().SetAmbient(0.3)  # 设定环境光系数
actor_isosurface_outer.GetProperty().SetDiffuse(0.6)  # 设定散射光系数
actor_isosurface_outer.GetProperty().SetSpecular(0.8)  # 设定反射光系数
actor_isosurface_outer.GetProperty().SetOpacity(0.2)  # 设定透射光系数
actor_isosurface_outer.RotateX(90)  # 设定X轴旋转
actor_isosurface_outer.SetScale(1.2, 1.2, 1.2)  # 设定缩放

# OUTLINE
actor_outline = vtk.vtkActor()  # 演员：边框
actor_outline.SetMapper(mapper_outline)  # 读取映射器：边框
actor_outline.GetProperty().SetColor(0.5, 0.5, 0.5)  # 设定颜色
actor_outline.GetProperty().SetLineWidth(2.0)  # 设定宽度
actor_outline.RotateX(90)  # 设定X轴旋转
actor_outline.SetScale(1.2, 1.2, 1.2)  # 设定缩放

#############
# RENDERER #
#############

renderer = vtk.vtkRenderer()  # 渲染器
renderer.AddActor(actor_isosurface_inner)  # 读取演员：内部等值面
renderer.AddActor(actor_isosurface_median)  # 读取演员：中部等值面
renderer.AddActor(actor_isosurface_outer)  # 读取演员：外部等值面
renderer.AddActor(actor_outline)  # 读取演员：边框
renderer.SetBackground(0.8, 0.8, 0.8)  # 设定背景颜色

##########
# WINDOW #
##########

window = vtk.vtkRenderWindow()  # 窗口
window.AddRenderer(renderer)  # 读取渲染器
window.Render()  # 渲染窗口
renderer.GetActiveCamera().Azimuth(90)  # 设定视角
window.SetSize(750, 750)  # 设定窗口大小
window.SetWindowName('Isosurface Rendering')  # 设定窗口名称
```

```
########
# SAVE #
########

writer_filter = vtk.vtkWindowToImageFilter()  # 过滤器：截图
writer_filter.SetInput(window)  # 读取窗口
writer = vtk.vtkPNGWriter()  # 图像生成器
writer.SetFileName(os.path.join(RESULT_PATH, 'isosurface_render.png'))  # 设定图像路径
writer.SetInputConnection(writer_filter.GetOutputPort())  # 读取窗口截图
writer.Write()  # 保存图片

##############
# INTERACTOR #
##############

def key_handle(obj, event):  # 按键交互功能

    key = obj.GetKeySym()  # 获取按键值

    # 自动旋转
    # 注意：旋转过程结束前不能进行交互！
    if key == 'Return':  # Enter键
        for _ in range(360):
            renderer.GetActiveCamera().Azimuth(1)  # 顺时针旋转
            window.Render()  # 渲染窗口

    # 动态调整位置（调试用）
    elif key == 'Up':  # 上方向键
        actor_isosurface_inner.AddPosition(0, 1, 0)  # 偏置为一
        print('bias + 1')  # 打印调试信息
        window.Render()  # 渲染窗口
    elif key == 'Down':  # 下方向键
        actor_isosurface_inner.AddPosition(0, -1, 0)  # 偏置为负一
        print('bias - 1')  # 打印调试信息
        window.Render()  # 渲染窗口

    # 动态显示等值面值（调试用）
    elif key == 'Right':  # 右方向键
        iso_value = filter_extractor_inner.GetValue(0)  # 获取等值面值
        iso_value += 10  # 等值面值加十
        print('isosurface value =', iso_value)  # 打印调试信息
        filter_extractor_inner.SetValue(0, iso_value)  # 调整等值面值
        window.Render()  # 渲染窗口
    elif key == 'Left':  # 左方向键
        iso_value = filter_extractor_inner.GetValue(0)  # 获取等值面值
        iso_value -= 10  # 等值面值减十
        print('isosurface value =', iso_value)  # 打印调试信息
        filter_extractor_inner.SetValue(0, iso_value)  # 调整等值面值
        window.Render()  # 渲染窗口

interactor = vtk.vtkRenderWindowInteractor()  # 交互器
interactor.SetRenderWindow(window)  # 读取窗口
interactor.AddObserver('KeyPressEvent', key_handle)  # 绑定按键功能
interactor.Initialize()  # 初始化交互器
interactor.Start()  # 运行交互器
```

体渲染：

```
import os  # 文件

import vtk  # 可视化

########
# PATH #
########
```

```
# 注意：所有路径必须为全英文！！！
HOME_PATH = os.getcwd()  # 主目录
DATA_PATH = os.path.join(HOME_PATH, 'data')  # 数据目录
RESULT_PATH = os.path.join(HOME_PATH, 'result')  # 结果目录

##########
# SOURCE #
##########

source = vtk.vtkNIFTIImageReader()  # 数据源
source.SetFileName(os.path.join(DATA_PATH, 'brain.nii.gz'))  # 读取体数据

############################
# TRANSFER FUNCTION: OUTER #
############################

########## color transfer function ##########

color_transfer_function_outer = vtk.vtkColorTransferFunction()  # 颜色传输函数
color_transfer_function_outer.AddRGBPoint(0, 0.4, 0.6, 1.0)
color_transfer_function_outer.AddRGBPoint(300, 0.4, 0.6, 1.0)
color_transfer_function_outer.AddRGBPoint(560, 0.9, 0.9, 0.9)

########## opacity transfer function ##########

opacity_transfer_function_outer = vtk.vtkPiecewiseFunction()  # 透明度传输函数
opacity_transfer_function_outer.AddPoint(0, 0.0)
opacity_transfer_function_outer.AddPoint(150, 0.0)
opacity_transfer_function_outer.AddPoint(500, 0.9)
opacity_transfer_function_outer.AddPoint(560, 0.9)
opacity_transfer_function_outer.AddPoint(650, 0.0)

########## gradient transfer function ##########

gradient_transfer_function_outer = vtk.vtkPiecewiseFunction()  # 梯度传输函数
gradient_transfer_function_outer.AddPoint(0, 0.0)
gradient_transfer_function_outer.AddPoint(100, 0.8)
gradient_transfer_function_outer.AddPoint(500, 1.1)
gradient_transfer_function_outer.AddPoint(700, 0.2)

########## property ##########

volume_property_outer = vtk.vtkVolumeProperty()  # 性质：外部体元素
volume_property_outer.SetColor(color_transfer_function_outer)  # 设定颜色传输函数
volume_property_outer.SetScalarOpacity(opacity_transfer_function_outer)  # 设定透明度传输函数
volume_property_outer.SetGradientOpacity(gradient_transfer_function_outer)  # 设定梯度传输函数
volume_property_outer.SetInterpolationTypeToLinear()  # 增强渲染效果
volume_property_outer.ShadeOn()  # 打开阴影
volume_property_outer.SetAmbient(1.0)  # 设定环境光系数
volume_property_outer.SetDiffuse(1.0)  # 设定散射光系数
volume_property_outer.SetSpecular(1.0)  # 设定反射光系数
volume_property_outer.SetSpecularPower(100)  # 设定反射光强

############################
# TRANSFER FUNCTION: INNER #
############################

########## color transfer function ##########

color_transfer_function_inner = vtk.vtkColorTransferFunction()  # 颜色传输函数
color_transfer_function_inner.AddRGBPoint(0, 1.0, 0.3, 0.5)
color_transfer_function_inner.AddRGBPoint(1000, 1.0, 0.3, 0.5)
```

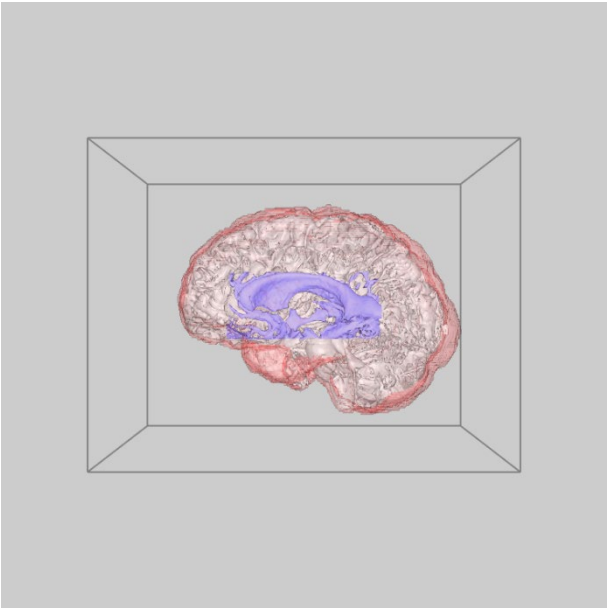
```
########## opacity transfer function ##########

opacity_transfer_function_inner = vtk.vtkPiecewiseFunction()  # 透明度传输函数
opacity_transfer_function_inner.AddPoint(0, 0.6)
opacity_transfer_function_inner.AddPoint(450, 0.6)
opacity_transfer_function_inner.AddPoint(451, 0.0)

########## gradient transfer function ##########

gradient_transfer_function_inner = vtk.vtkPiecewiseFunction()  # 梯度传输函数
gradient_transfer_function_inner.AddPoint(0, 0.0)
gradient_transfer_function_inner.AddPoint(100, 5.0)

########## property ##########

volume_property_inner = vtk.vtkVolumeProperty()  # 性质：内部体元素
volume_property_inner.SetColor(color_transfer_function_inner)  # 设定颜色传输函数
volume_property_inner.SetScalarOpacity(opacity_transfer_function_inner)  # 设定透明度传输函数
volume_property_inner.SetGradientOpacity(gradient_transfer_function_inner)  # 设定梯度传输函数
volume_property_inner.SetInterpolationTypeToLinear()  # 增强渲染效果
volume_property_inner.ShadeOn()  # 打开阴影
volume_property_inner.SetAmbient(0.8)  # 设定环境光系数
volume_property_inner.SetDiffuse(0.8)  # 设定散射光系数
volume_property_inner.SetSpecular(0.8)  # 设定反射光系数
volume_property_inner.SetSpecularPower(10)  # 设定反射光强

##########
# FILTER #
##########

# VOI
source_inner = vtk.vtkExtractVOI()  # 过滤器：感兴趣区域
source_inner.SetInputConnection(source.GetOutputPort())  # 读取数据源
source_inner.SetVOI(100, 155, 75, 180, 47, 70)  # 划定区域
source_inner.SetSampleRate(1, 1, 1)  # 设定采样率

# CLIPPER
plane_filter = vtk.vtkPlane()  # 过滤器：切割面
plane_filter.SetOrigin(140, 127.5, 61.5)  # 设定原点
plane_filter.SetNormal(-1, 0, 0)  # 设定法向

# OUTLINE
outline_filter = vtk.vtkOutlineFilter()  # 过滤器：边框
outline_filter.SetInputConnection(source.GetOutputPort())  # 读取数据源

##########
# MAPPER #
##########

# VOLUME: outer
volume_mapper_outer = vtk.vtkFixedPointVolumeRayCastMapper()  # 映射器：外部体元素
volume_mapper_outer.SetInputConnection(source.GetOutputPort())  # 读取数据源
volume_mapper_outer.AddClippingPlane(plane_filter)  # 切割体元素

# VOLUME: inner
volume_mapper_inner = vtk.vtkFixedPointVolumeRayCastMapper()  # 映射器：内部体元素
volume_mapper_inner.SetInputConnection(source_inner.GetOutputPort())  # 读取数据源

# OUTLINE
outline_mapper = vtk.vtkPolyDataMapper()  # 映射器：边框
outline_mapper.SetInputConnection(outline_filter.GetOutputPort())  # 读取过滤器：边框
```

```
#########
# ACTOR #
#########

# VOLUME: outer
volume_actor_outer = vtk.vtkVolume()  # 演员：外部体元素
volume_actor_outer.SetMapper(volume_mapper_outer)  # 读取映射器：外部体元素
volume_actor_outer.SetProperty(volume_property_outer)  # 读取性质：外部体元素
volume_actor_outer.RotateX(90)  # 设定X轴旋转
volume_actor_outer.SetScale(1.2, 1.2, 1.2)  # 设定缩放

# VOLUME: inner
volume_actor_inner = vtk.vtkVolume()  # 演员：内部体元素
volume_actor_inner.SetMapper(volume_mapper_inner)  # 读取映射器：内部体元素
volume_actor_inner.SetProperty(volume_property_inner)  # 读取性质：内部体元素
volume_actor_inner.RotateX(90)  # 设定X轴旋转
volume_actor_inner.SetScale(1.2, 1.2, 1.2)  # 设定缩放

# OUTLINE
outline_actor = vtk.vtkActor()  # 演员：边框
outline_actor.SetMapper(outline_mapper)  # 读取映射器：边框
outline_actor.GetProperty().SetColor(0.5, 0.5, 0.5)  # 设定颜色
outline_actor.GetProperty().SetLineWidth(2.0)  # 设定宽度
outline_actor.RotateX(90)  # 设定X轴旋转
outline_actor.SetScale(1.2, 1.2, 1.2)  # 设定缩放

############
# RENDERER #
############

renderer = vtk.vtkRenderer()  # 渲染器
renderer.AddVolume(volume_actor_outer)  # 读取演员：外部体元素
renderer.AddVolume(volume_actor_inner)  # 读取演员：内部体元素
renderer.AddActor(outline_actor)  # 读取演员：边框
renderer.SetBackground(0.8, 0.8, 0.8)  # 设定背景颜色

##########
# WINDOW #
##########

window = vtk.vtkRenderWindow()  # 窗口
window.AddRenderer(renderer)  # 读取渲染器
window.Render()  # 渲染窗口
renderer.GetActiveCamera().Azimuth(90)  # 设定视角
window.SetSize(750, 750)  # 设定窗口大小
window.SetWindowName('Volume Rendering')  # 设定窗口名称

########
# SAVE #
########

writer_filter = vtk.vtkWindowToImageFilter()  # 过滤器：截图
writer_filter.SetInput(window)  # 读取窗口
writer = vtk.vtkPNGWriter()  # 图像生成器
writer.SetFileName(os.path.join(RESULT_PATH, 'volume_render.png'))  # 设定图像路径
writer.SetInputConnection(writer_filter.GetOutputPort())  # 读取窗口截图
writer.Write()  # 保存图片

##############
# INTERACTOR #
##############
```

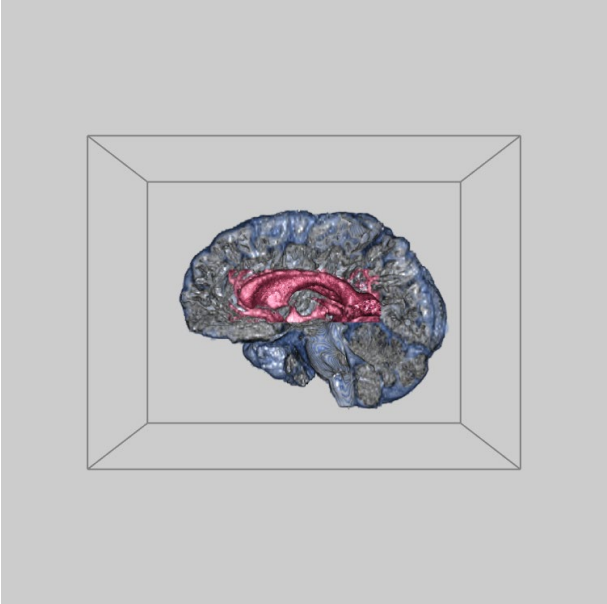
```
def key_handle(obj, event):  # 按键交互功能

    key = obj.GetKeySym()  # 获取按键值

    # 自动旋转
    # 注意：旋转过程结束前不能进行交互！
    if key == 'Return':  # Enter键
        for _ in range(72):
            renderer.GetActiveCamera().Azimuth(5)  # 顺时针旋转
            window.Render()  # 渲染窗口

    # 手动旋转（调试用）
    elif key == 'Up':  # Up键
        renderer.GetActiveCamera().Elevation(5)  # 向上旋转
        print('Up')  # 打印调试信息
        window.Render()
    elif key == 'Down':  # Up键
        renderer.GetActiveCamera().Elevation(-5)  # 向下旋转
        print('Down')  # 打印调试信息
        window.Render()
    elif key == 'Left':  # Up键
        renderer.GetActiveCamera().Azimuth(5)  # 向左旋转
        print('Left')  # 打印调试信息
        window.Render()
    elif key == 'Right':  # Up键
        renderer.GetActiveCamera().Azimuth(-5)  # 向右旋转
        print('Right')  # 打印调试信息
        window.Render()

interactor = vtk.vtkRenderWindowInteractor()  # 交互器
interactor.SetRenderWindow(window)  # 读取窗口
interactor.AddObserver('KeyPressEvent', key_handle)  # 绑定按键功能
interactor.Initialize()  # 初始化交互器
interactor.Start()  # 运行交互器
```

- **Results:**

等值面渲染：



体渲染：



结果分析：

程序运行后会生成交互界面，基于交互界面制作的自动旋转视角视频可查看对应的 mp4 文件。以上两图为交互界面初始状态的截图。比较两种方法，体渲染的结果比面渲染在视觉上更为精确，但对应的渲染时间更长。

3 设计算法消除等值面渲染结果中碎片化的面单元，使用数据进行测试并展示可视化结果。

- **Python script code:**

  3_fragment.py

  3_fragment_elimination.py

- **Input & Output:**

  Input:

  heart.nii.gz

  Output:

  fragment.png

  fragment_elimination.png

- **Description:**

3_fragment.py 文件中展示的是心脏 CT 图像重建结果，使用的渲染方式为等值面渲染，等值面值为 150，光线模型为 0.25*环境光+0.6*散射光+0.9*反射光，不透明率为 1.0，颜色为浅黄色。重建的结果中存在大量碎片化的面单元，为了消除多余的面单元，需要在 filter 阶段对体数据进行平滑滤波。3_fragment_elimination.py 文件中展示了消除碎片化面单元的重建结果，消除过程采用三个叠加的平滑滤波器，分别为中值滤波器、高斯滤波器以及扩散滤波器，其中高斯滤波器的滤波半径为 2.0，标准差为 0.48，扩散滤波器的阈值为 30，迭代次数为 20。

- **Source Code:**

```
import os  # 文件

import vtk  # 可视化

########
# PATH #
########

# 注意：所有路径必须为全英文！！！
HOME_PATH = os.getcwd()  # 主目录
DATA_PATH = os.path.join(HOME_PATH, 'data')  # 数据目录
RESULT_PATH = os.path.join(HOME_PATH, 'result')  # 结果目录

##########
# SOURCE #
##########

source = vtk.vtkNIFTIImageReader()  # 数据源
source.SetFileName(os.path.join(DATA_PATH, 'heart.nii.gz'))  # 读取体数据

##########
# FILTER #
##########

# MEDIAN SMOOTH
median = vtk.vtkImageHybridMedian2D()  # 过滤器：中值滤波
median.SetInputConnection(source.GetOutputPort())  # 读取数据源

# GAUSSIAN SMOOTH
gaussian = vtk.vtkImageGaussianSmooth()  # 过滤器：高斯滤波
gaussian.SetInputConnection(median.GetOutputPort())  # 读取过滤器：中值滤波
gaussian.SetDimensionality(3)  # 设定数据维度
gaussian.SetRadiusFactor(2.0)  # 设定滤波半径
gaussian.SetStandardDeviation(0.48)  # 设定滤波标准差

# DIFFUSION SMOOTH
diffusion = vtk.vtkImageAnisotropicDiffusion3D()  # 过滤器：扩散滤波
diffusion.SetInputConnection(gaussian.GetOutputPort())  #读取过滤器：高斯滤波
diffusion.SetNumberOfIterations(20)  # 设定迭代次数
diffusion.SetDiffusionThreshold(30)  # 设定扩散阈值
```

```
# EXTRACTOR
extractor = vtk.vtkMarchingCubes()  # 过滤器：三角面片
extractor.SetInputConnection(diffusion.GetOutputPort())  # 读取过滤器：扩散滤波
extractor.ComputeNormalsOn()  # 计算等值面法向量以增强渲染效果
extractor.SetValue(0, 150)  # 设定等值面

# STRIPPER
isosurface = vtk.vtkStripper()  # 过滤器：等值面
isosurface.SetInputConnection(extractor.GetOutputPort())  # 读取过滤器：三角面片

##########
# Mapper #
##########

mapper = vtk.vtkPolyDataMapper()  # 映射器：等值面
mapper.SetInputConnection(isosurface.GetOutputPort())  # 读取过滤器：等值面
mapper.ScalarVisibilityOff()  # 关闭颜色映射关系

#########
# ACTOR #
#########

actor = vtk.vtkActor()  # 演员：等值面
actor.SetMapper(mapper)  # 读取映射器：等值面
actor.GetProperty().SetColor(0.7, 0.7, 0.5)  # 设定颜色
actor.GetProperty().SetAmbient(0.25)  # 设定环境光系数
actor.GetProperty().SetDiffuse(0.6)  # 设定散射光系数
actor.GetProperty().SetSpecular(0.9)  # 设定反射光系数
actor.GetProperty().SetOpacity(1.0)  # 设定透射光系数

############
# RENDERER #
############

renderer = vtk.vtkRenderer()  # 渲染器
renderer.AddActor(actor)  # 读取演员：等值面
renderer.SetBackground(0.8, 0.8, 0.8)  # 设定背景颜色

##########
# WINDOW #
##########

window = vtk.vtkRenderWindow()  # 窗口
window.AddRenderer(renderer)  # 读取渲染器
window.Render()  # 渲染窗口
window.SetSize(750, 750)  # 设定窗口大小
window.SetWindowName('Fragment Elimination')  # 设定窗口名称

########
# SAVE #
########

writer_filter = vtk.vtkWindowToImageFilter()  # 过滤器：截图
writer_filter.SetInput(window)  # 读取窗口
writer = vtk.vtkPNGWriter()  # 图像生成器
writer.SetFileName(os.path.join(RESULT_PATH, 'fragment_elimination.png'))  # 设定图像路径
writer.SetInputConnection(writer_filter.GetOutputPort())  # 读取窗口截图
writer.Write()  # 保存图片
```

```
##############
# INTERACTOR #
##############

def key_handle(obj, event):  # 按键交互功能

    key = obj.GetKeySym()  # 获取按键值

    # 动态显示等值面值（调试用）
    if key == 'Right':  # 右方向键
        iso_value = extractor.GetValue(0)  # 获取等值面值
        iso_value += 10  # 等值面值加十
        print('isosurface value =', iso_value)  # 打印调试信息
        extractor.SetValue(0, iso_value)  # 调整等值面值
        window.Render()  # 渲染窗口
    elif key == 'Left':  # 左方向键
        iso_value = extractor.GetValue(0)  # 获取等值面值
        iso_value -= 10  # 等值面值减十
        print('isosurface value =', iso_value)  # 打印调试信息
        extractor.SetValue(0, iso_value)  # 调整等值面值
        window.Render()  # 渲染窗口

interactor = vtk.vtkRenderWindowInteractor()  # 交互器
interactor.SetRenderWindow(window)  # 读取窗口
interactor.AddObserver('KeyPressEvent', key_handle)  # 绑定按键功能
interactor.Initialize()  # 初始化交互器
interactor.Start()  # 运行交互器
```

此处仅展示消除碎片化面单元效果的代码，原图的代码与之相似，具体请查看文件。

- **Results:**

消除碎片化面单元的前后对比图如下。左图为消除前，右图为消除后。