Assignment 8

Junyi Liao 20307110289

2022 年 12 月 21 日

1 等值面渲染及移动立方体法

1.1 算法概述

三维数据场的可视化方法主要分为面渲染和体渲染，而等值面渲染 (isosurface rendering) 则是面渲染中最为重要和常用的一类方法.

对于给定的三维标量 (scalar) 数据场，若存在某一曲面内所有点对应的数据取值均为常数 c，则该曲面是一个等值面. 等值面渲染是将三维数据场中某一数值所对应的等值面进行可视化，从而提取出该数据场的某些特征. 例如给定人体的数据场，可以利用等值面渲染提取出人的骨骼、皮肤等特征.

移动立方体法 (marching cubes) 是一类常用的求解等值面的算法，这一算法简单、且计算上具有高效性. 该方法假定输入的三维标量数据场离散且规则排列，可以看作三维网格的格点，进一步，一个体元 (voxel) 可以看作由相邻层上8个顶点组成的小正方体. 为提取某一常数 c 对应的等值面，我们可以根据顶点数据是否大于 c 将顶点区分为正类 $(x > c)$ 或负类 $(x \leq c)$；若相邻顶点属于不同类别，则等值面穿过这两个顶点所连成的棱边，并且可应用线性插值的方法求出交点的位置；将交点连成三角面，构成等值面片，遍历数据场中所有体元，所有等值面片的集合便构成了最终的等值面.

另外，在单个体元中，移动立方体法只能通过插值求出等值面与体元交点，但可能存在多种方式将这些交点连接为三角形（歧义性问题），因此在构造等值面片时，还要注意使相邻体元之间的等值面片相互匹配，确保最终形成的等值面是封闭、连续的.

1.2 基于VTK库的实现

数据简介 本例中我们使用的体数据为 `image_lr.nii.gz`，这是一个尺寸为 $124 \times 124 \times 73$ 的标量场数据，储存了心脏CT扫描结果.

渲染流程 在接下来的步骤中，我们将调用可视化渲染引擎库VTK实现三维体数据的等值面渲染. VTK的渲染过程大致如下：

(1) 读入源数据 (Source)；

(2) 映射器 (Mapper) 将源数据映射到三维空间中，得到对应的几何图形；

`vtk.vtkPolyDataMapper()`

(3) 演员 (Actor) 将映射器封装起来，相当于空间中的一个3D对象 (3D Object)；

`vtk.vtkActor()`

(4) 渲染器 (Renderer) 在特定的光照和视角下对三维空间中的场景和对象 (Actor) 进行渲染，

相当于摄影，使三维物体在二维屏幕上成像；

`vtk.vtkRenderer()`

(5) 渲染窗口 (Render Window) 展示渲染器的渲染结果；

`vtk.vtkRenderWindow()`

(6) 交互器 (Interactor) 提供窗口与交互界面；

`vtk.vtkRenderWindowInteractor()`

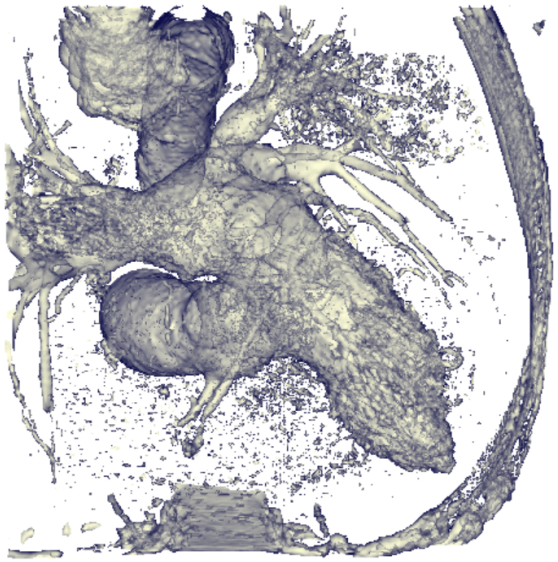
另外，我们调用了VTK库中的`vtk.vtkMarchingCubes()`计算三维标量数据场中的等值面，这一类对象基于移动立方体算法提取等值面，返回一个`vtk.vtkPolyData`对象；随后我们调用了`vtk.vtkStripper()`形成三角面片，以构造完整的等值面. 具体代码实现如下.

```
import vtk
import nibabel as nib


# Read in the data.
def read_nib(path='./data/image_lr.nii.gz'):
    # Read in the scalar field data.
    heart_nib = nib.load(path)
    heart_voxel = heart_nib.get_fdata()
    print(f'Shape: {heart_voxel.shape}')

    # Construct the vtk image data.
    heart = vtk.vtkImageData()
    heart.SetDimensions(heart_voxel.shape)
    heart.SetSpacing(heart_nib.header['pixdim'][1:4])
    heart.SetOrigin(0, 0, 0)
    heart.AllocateScalars(vtk.VTK_DOUBLE, 1)
    for z in range(heart_voxel.shape[2]):
        for y in range(heart_voxel.shape[1]):
            for x in range(heart_voxel.shape[0]):
                heart.SetScalarComponentFromDouble(x, y, z, 0, heart_voxel[
    x, y, z])
    return heart


# Compute the isosurface.
def isosurface_compute(data, iso=150):
    # Use marching cube to compute the isosurface.
```

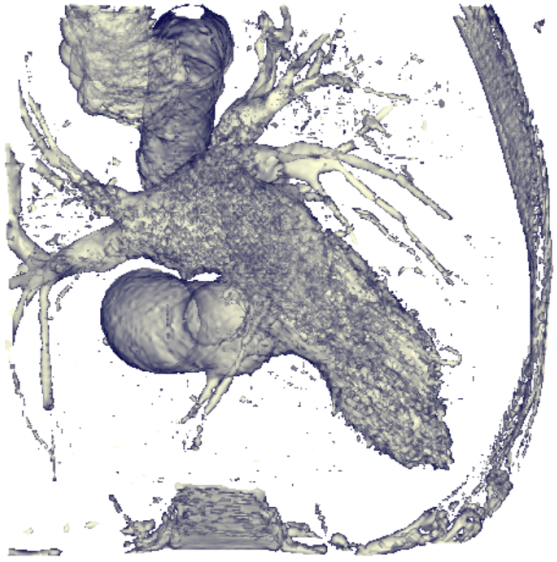
```
28      extractor = vtk.vtkMarchingCubes()
29      extractor.SetInputData(data)
30      extractor.SetValue(0, iso)
31
32      # Apply Laplacian smoothing.
33      smoother = vtk.vtkSmoothPolyDataFilter()
34      smoother.SetInputConnection(extractor.GetOutputPort())
35      smoother.SetRelaxationFactor(0.01)
36      smoother.SetNumberOfIterations(300)
37
38      # Generate triangle strips.
39      # Not smoothed.
40      stripper1 = vtk.vtkStripper()
41      stripper1.SetInputConnection(extractor.GetOutputPort())
42      # Smoothed.
43      stripper2 = vtk.vtkStripper()
44      stripper2.SetInputConnection(smoother.GetOutputPort())
45      return stripper1, stripper2
46
47
48  # Render the isosurface.
49  def render(stripper):
50      # Render the isosurface.
51      # Maps data to graphic primitives.
52      mapper = vtk.vtkPolyDataMapper()
53      mapper.SetInputConnection(stripper.GetOutputPort())
54
55      # Set the actor.
56      actor = vtk.vtkActor()
57      actor.SetMapper(mapper)
58      actor.GetProperty().SetColor(1, 1, 0)
59      actor.GetProperty().SetOpacity(0.95)
60      actor.GetProperty().SetAmbient(0.25)
61      actor.GetProperty().SetDiffuse(0.6)
62      actor.GetProperty().SetSpecular(1.0)
63
64      # Set the renderer, window, and interactor.
65      renderer = vtk.vtkRenderer()
66      renderer.SetBackground((1, 1, 1))
```

```
    renderer.AddActor(actor)

    window = vtk.vtkRenderWindow()
    window.AddRenderer(renderer)

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

    # Render.
    interactor.Initialize()
    window.Render()
    interactor.Start()


if __name__ == '__main__':
    heart = read_nib(path='./data/image_lr.nii.gz')
    isosurface, isosurface_smoothed = isosurface_compute(heart, iso=150)
    render(isosurface)
    render(isosurface_smoothed)
```

分别设置 iso 为 150 和 200，可以对应数值的等值面提取结果，图片如下.



(1) 等值面渲染结果1，iso=150



(2) 等值面渲染结果2，iso=200

2 碎片化等值面的去除

2.1 去除方法

观察上述渲染结果，我们发现在周围背景中有较多碎片化等值面（尤其是在 iso=150 的等值面中），在影响视觉效果的同时，可能还会干扰部分关键信息的获取. 为此，我们将采取相关方法去除这些碎片化的等值面.

用移动立方体算法构造的等值面为多个顶点及其构成三角面片的集合，可以看作一个多面体的表面；我们可以应用 Laplace 平滑的方法对等值面进行处理，此方法借用了机器学习中的 $k-$近邻方法 (k-nearest neighbors, kNN) 的思想，假设构成等值面的所有顶点集合为 $\mathcal{S} = \{x_i\}_{i=1}^N$，且距离顶点 x_i 最近的 k 个顶点构成的集合为 $\mathcal{N}_k(x_i)$，即

$$\mathcal{N}_k(x_i) = \underset{\mathcal{N}\subset\mathcal{S},\ |\mathcal{N}|=k,\ x_i\notin\mathcal{N}}{\arg\min} \sum_{x_j\in\mathcal{N}} \|x_i - x_j\|_2^2;$$

对于等值面上每一个顶点，拉普拉斯平滑将求取其 k 个最近邻的中心点，并将该顶点向近邻中心点方向移动一定距离，即

$$x_i' = (1-\lambda)x_i + \lambda \sum_{x_j\in\mathcal{N}_k(x_i)} \frac{x_j}{k};$$

其中 λ 作为松弛因子，决定平滑后顶点的偏移程度；这一方法可以实现对等值面的平滑，且在构成等值面顶点数目较多（远大于 k ）时，这一方法基本保留了等值面原本的结构；

对于较小的、碎片化的等值面，其结构中的顶点数目较少，若将每个顶点向 k-近邻中心移动，会导致整个等值面向中心收缩；通过选取合适的 k 并多次重复这一过程，最终会使碎片化等值面坍缩至中心点，由此实现了碎片化等值面的去除.

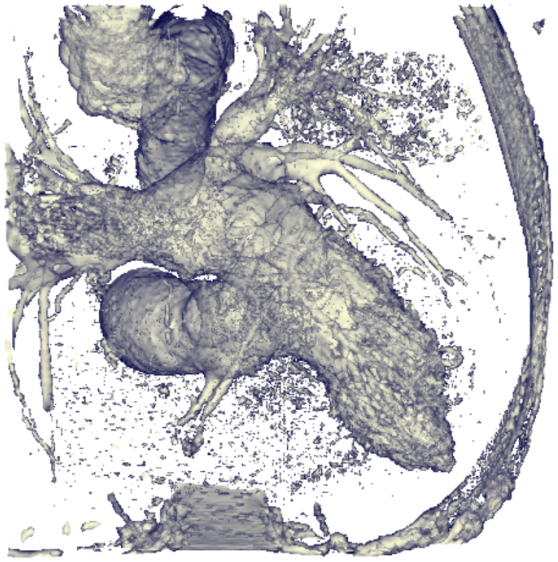
2.2 VTK实现

上述的 Laplace 平滑方法可以调用VTK库中的 vtkSmoothPolyDataFilter 实现，代码已在 **1.2** 中给出，注意平滑之后，还要使用 vtk.vtkStripper 构造三角面片和等值面；

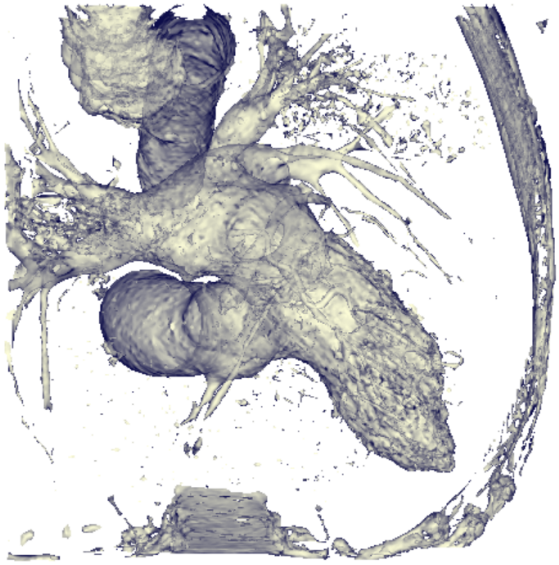
```
def isosurface_compute(data, iso=150):
    # Use marching cube to compute the isosurface.
    extractor = vtk.vtkMarchingCubes()
    extractor.SetInputData(data)
    extractor.SetValue(0, iso)

    # Apply Laplacian smoothing.
    smoother = vtk.vtkSmoothPolyDataFilter()
    smoother.SetInputConnection(extractor.GetOutputPort())
    smoother.SetRelaxationFactor(0.01)
    smoother.SetNumberOfIterations(300)

```

```
    # Generate triangle strips.
    # Not smoothed.
    stripper1 = vtk.vtkStripper()
    stripper1.SetInputConnection(extractor.GetOutputPort())
    # Smoothed.
    stripper2 = vtk.vtkStripper()
    stripper2.SetInputConnection(smoother.GetOutputPort())
    return stripper1, stripper2
```

我们针对iso=150的等值面进行碎片去除，设置松弛因子$\lambda = 0.01$，迭代次数$n = 300$，最终效果如下.



(3) 原等值面渲染结果，iso=150



(4) 去除碎片化等值面后

对比左右两图，可见平滑之后碎片化等值面得到了较好的去除.

3 附录

在本次作业中，我们的所有步骤均在IDE PyCharm Community Edition 2022.1.2 中完成，且均在conda虚拟环境中运行，对应的Python 版本为3.9.0.

除Python内置标准库之外，我们还用到了以下库，这些库可能需要您手动安装，您可以在对应虚拟环境下使用`pip install`命令进行安装.

`nibabel==4.0.2`用于读取体数据；

`vtk==9.2.2`用于计算体数据的等值面及可视化渲染.