第六次作业报告

邓文 17307130171

一、阅读了解 VTK（VTK - The Visualization Toolkit, www.vtk.org），学习某个编程环境下调用 VTK 库进行可视化。该题只需要回答已经学习（完成作业）或没有学习（未完成作业）。

已完成作业，并学习了在 python 编程环境下运用 VTK 进行可视化。

除了学习 elearning 上 **14.1 Demo_surface_volumerender.pdf** 文件之外，还阅读了以下网站的相关教学资源，分享在这里：

1. https://lorensen.github.io/VTKExamples/site/VTKBook/ VTK 官网 book（英文版，阅读太复杂，仅看了几页就放弃了）

2. https://blog.csdn.net/shenziheng1/article/list/13?t=1 VTK 修炼之道系列（中文讲解，内容细致齐全!! 推荐）

3. https://blog.csdn.net/www_doling_net//article/list/1 VTK 教程（中文讲解，但内容并不是特别全）

4. https://lorensen.github.io/VTKExamples/site/Python/ VTK 官网 python examples（含有 python 相关的很多例子，可以用来练手，感受 VTK 的绘制流程）

5. https://vtk.org/doc/nightly/html/index.html VTK 官网文档（里面有各种类和函数的定义，写代码时可以通过查看具体的定义来了解接口的详情，以及参数的意义。）

二、调用可视化渲染引擎库（如 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）。需要实现的渲染过程包括：(1) 等值面渲染，(2) 体渲染。请自己找一个体数据进行测试和结果展示。提交作业需要对使用数据进行说明，并提交源数据（或数据下载的网上链接）。

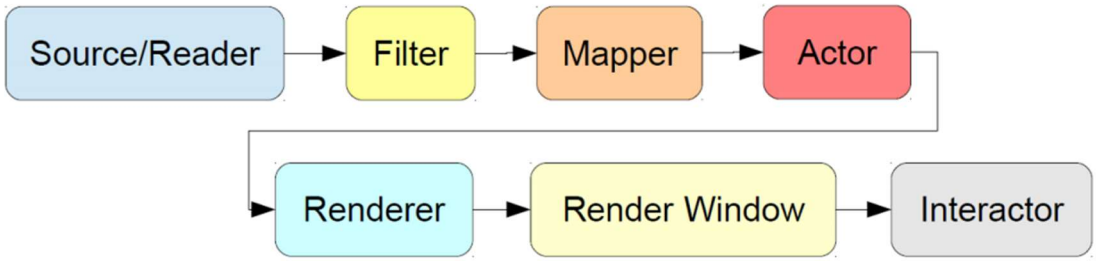
数据说明：——**backpack.vti**
此数据为我在网上下载的一个安检时**背包**的体数据，具体的可视化图片我会在后面放出。

1. 等值面渲染 – Marching Cubes

Marching Cubes 的算法原理：

- 首先，假定原始数据是离散的三维空间规则数据场，（断层扫描仪 CT 及核磁共振仪 MRI 产生的图像均属于这一类型);
- 其次，给出所求等值面的值(为了在这一数据场中构造等值面);
- 最后，找出等值面经过的体元位置，求出该体元内的等值面并计算出相关参数(以便由常用的图形软件包或图形硬件提供的面绘制功能绘制等值)

VTK 的使用流程图如下：



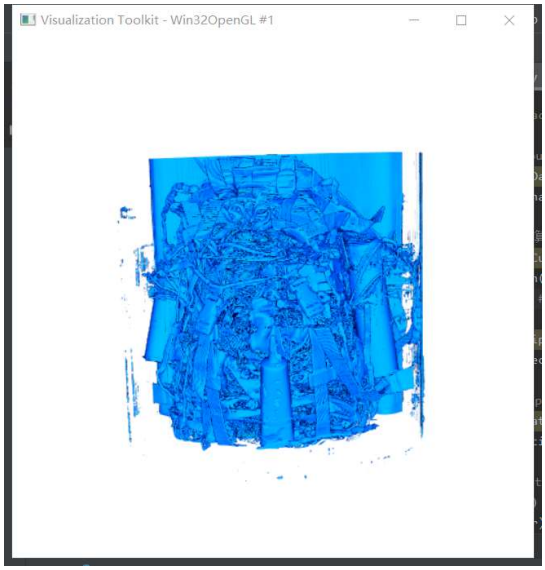
在等值面渲染的过程中，在 Filter 步骤中，使用了 Marching Cubes 的算法来计算等值面，并将结果传递给 mapper 过程。
具体的类的定义及使用方法在 https://vtk.org/doc/nightly/html/classvtkMarchingCubes.html 中有详细介绍。

具体的代码实现：

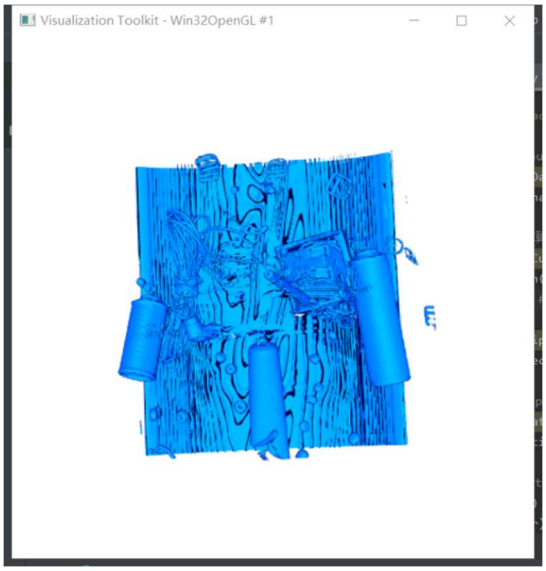
```
# the process of following code:
# source/reader → filter → mapper → actor → renderer → renderWindow → interactor

file_name = "./data/backpack.vti"

# 读取数据文件  -----source/reader
img = vtk.vtkXMLImageDataReader()
img.SetFileName(file_name)

# 利用 marching cubes 算法提取等值面  -------filter
iso = vtk.vtkMarchingCubes()
iso.SetInputConnection(img.GetOutputPort())
iso.SetValue(0, 1300)  # 设置第一个等值面的阈值
# 第一个参数为 i 个等值面，  第二个参数为该等值面的 value

stripper = vtk.vtkStripper()  # 通过vtkStripper在等值面上产生纹理或三角面片
stripper.SetInputConnection(iso.GetOutputPort())

# 设置 mapper ------mapper
mapper = vtk.vtkPolyDataMapper()
mapper.SetInputConnection(stripper.GetOutputPort())

# 设置actor   ------actor
actor = vtk.vtkActor()
actor.SetMapper(mapper)
actor.GetProperty().SetColor(0., 0.75, 1.)  #  设置acotr 的颜色 和 光照 等属性
actor.GetProperty().SetAmbient(0.25)
actor.GetProperty().SetDiffuse(0.6)
actor.GetProperty().SetSpecular(1)

# 设置渲染器  -------renderer
ren = vtk.vtkRenderer()
ren.SetBackground(1, 1, 1)
ren.AddActor(actor)

# 设置窗口  -------renderWindow
renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren)
renWin.SetSize(512, 512)
# 设置 Interactor  -------interactor
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
renWin.Render()
iren.Start()
```

代码均有详细注释，如果不清晰可以查看具体的 code。

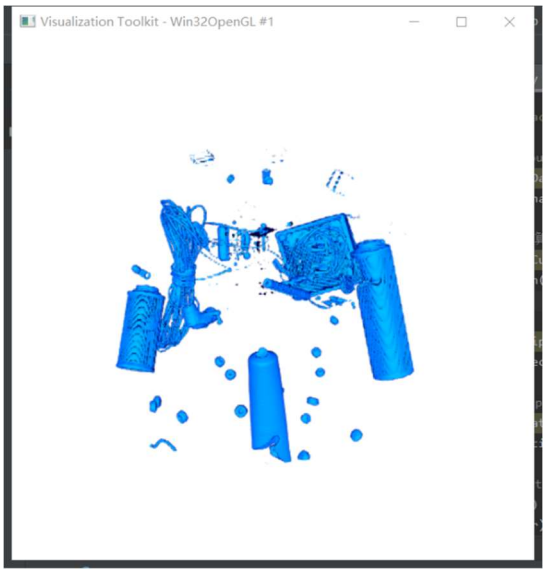
最终的结果如下：



图一



图二



图三

这是将 MarchingCubes 第一个等值面阈值设为 200(图一)，800(图三)，1300(图三)的结果。

结果分析:

我们可以看到，起初还能够看到书包的外部轮廓的和细节，后面随着将阈值不断增大，书包的外表已经被消失，并且放置书包的类似“传送带”的物体也逐渐消失，我们能够较为清楚到看见书包里面水瓶，牙膏，电线，充电器等物品。

因此，通过对等值面的绘制，我们可以可视化出输出的三维模型，并且通过对等值面的阈值的设定，我们可以展示不同的感兴趣的图片要素。

2. 体渲染 --Ray Casting

体渲染是将三维空间的离散数据直接转换为最后的立体，图像而不必生成中间几何图元(面绘制需要)，其中心思想是为每一个体素指定一个不透明度,并考虑每一个体素对光线的透射、发射和反射作用。
Ray Casting 原理：从图像平面的每个像素都沿着视线方向发出一条射线，此射线穿过体数据集，按一定步长进行采样，由内插计算每个采样点的颜色值和不透明度，然后由前向后或由后向前逐点计算累计的颜色值和不透明度值，直至光线完全被吸收或穿过物体。该方法能很好地反映物质边界的变化，使用 Phong 模型，引入镜面反射、漫反射和环境反射能得到很好的光照效果，在医学上可将各组织器官的性质属性、形状特征及相互之间的层次关系表现出来，从而丰富了图像的信息。

具体的实现如下：

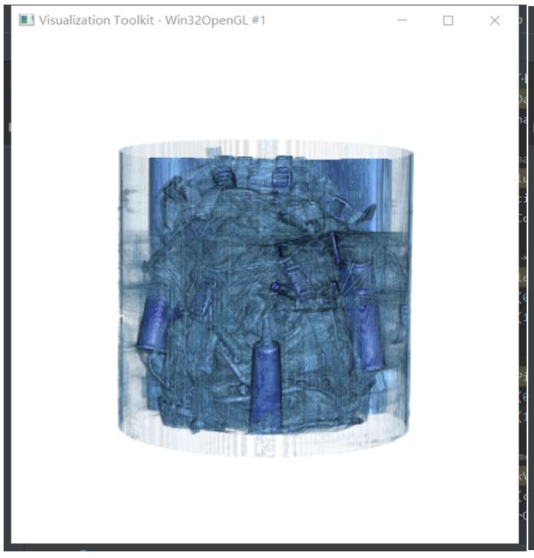
```
file_name = "./data/backpack.vti"

# 读取数据文件  -----source/reader
img = vtk.vtkXMLImageDataReader()
img.SetFileName(file_name)

# 设置 ray cast 体渲染mapper
mapper = vtk.vtkGPUVolumeRayCastMapper()
mapper.SetInputConnection(img.GetOutputPort())
mapper.SetBlendModeToComposite()

# 设置颜色映射  scalar → RGB value
color_map = vtk.vtkColorTransferFunction()
color_map.AddRGBPoint(0,     0.5, 0.8, 1.0)
color_map.AddRGBPoint(15000, 0.0, 0.0, 0.55)

# 设置透明度映射 scalar → opacity
opacity_map = vtk.vtkPiecewiseFunction()
opacity_map .AddPoint(0, 0.0)
opacity_map .AddPoint(15000, 1)

# 将以上属性添加进 volume property 中
vol_property = vtk.vtkVolumeProperty()
vol_property.SetColor(color_map)
vol_property.SetScalarOpacity(opacity_map)

# 其他属性设置
vol_property.SetInterpolationTypeToLinear()  # 插值方法为线性插值
vol_property.ShadeOn()  # 阴影开启
vol_property.SetAmbient(0.)  # 环境光系数
vol_property.SetDiffuse(0.9)  # 散射光系数
vol_property.SetSpecular(0.5)  # 反射光系数

# 设置 3D 对象 --- 类似之前的actor
volume = vtk.vtkVolume()
volume.SetMapper(mapper)
volume.SetProperty(vol_property)
```

```
# 渲染器
ren = vtk.vtkRenderer()
ren.SetBackground(1, 1, 1)
ren.AddVolume(volume)
light = vtk.vtkLight()
light.SetColor(1,1,1) #光的颜色
#设置灯光类型为相机灯光，灯光会自动随相机移动
light.SetLightTypeToCameraLight()
ren.AddLight(light)

# 渲染器窗口
renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren)
renWin.SetSize(512, 512)

# 设置交互器
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)

# 启动
iren.Initialize()
renWin.Render()
iren.Start()
```

（具体代码参考了老师给的 ppt，以及官网的 Documention https://vtk.org/doc/nightly/html/classvtkOpenGLGPUVolumeRayCastMapper.html）

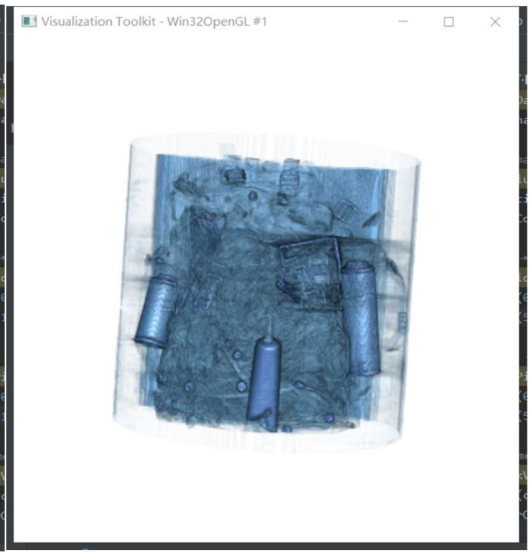
结果展示：

说明：我将图像的 scalar 的值分为了两个区间，第一个区间的颜色为浅蓝色，且几乎透明，第二个区间的颜色为深蓝色，且几乎不透明。



图一

“区间：[0~1300] & [1300,+∞）”



图二

“区间：[0~5000] & [5000,+∞）”

图三

“区间：(0, 15000) & (15000, +∞)”

结果分析，从实验结果我们可以看出，将第一个区间的宽度不断提高，使得书包的外表信息以及外部传送带的信息不断地变浅，而书包里的物体，水瓶，牙膏等不断变得明显。

因此，通过 Ray Casting 的方法，我们对 3d 数据进行了体绘制。为每一个体素指定一个不透明度,并考虑每一个体素对光线的透射、发射和反射作用，通过这样的方法，我们就可以突出我们感兴趣的部分。

同时，我们也能通过调整 actor 里面的光照属性等参数来改变可视化的效果。

三、 请设计一个方法消除等值面渲染结果中的碎片化的面单元，如下图所示，并使用数据进行测试并且展示可视化结果。心脏 CT 图像：image_lr.nii.gz

为了消除等值面渲染中的碎片化的单元，我想到了图像处理中的过滤的操作。因此，通过阅读 VTK 的官方文档，我尝试了以下两种不同的方法进行过滤：

1. 利用 **vtkSmoothPolyDataFilter** 进行平滑过滤：**（这一步是在 marching cubes 算法之后）**

介绍：这个类主要通过 laplacian smoothing 来对数据进行平滑。
该算法进行如下。对于每个顶点 v，执行拓扑和几何分析，以确定哪些顶点连接到 v，哪些单元连接到 v。然后，为每个顶点构造一个连通性数组。 （连接性数组是直接连接到每个顶点的顶点列表的列表。）接下来，一个迭代阶段开始于所有顶点。对于每个顶点 v，v 的坐标根据连接的顶点的平均值进行修改。 （**松弛因子**可用于

控制 v 的位移量)。对每个顶点重复该过程。遍历顶点列表是一次迭代。重复许多迭代（**通常大约 20 次**），直到获得所需的结果。
具体的介绍参考官方文档:
https://vtk.org/doc/nightly/html/classvtkSmoothPolyDataFilter.html#details

2. 利用 **vtkImageGaussianSmooth** 进行平滑过滤**(这一步是在 marching cubes 算法之前)**

介绍：这类主要对输入的图像进行高斯卷积，至此 1~3 维的数据。
具体的介绍参考官方文档:
https://vtk.org/doc/nightly/html/classvtkImageGaussianSmooth.html#details

3. 具体的代码实现:

- **数据的读入**

有两个方法：一种是 **python + ninabel + vtkImageData** 读取，如下所示:

```
image_lr = nib.load('./data/image_lr.nii.gz')  # 使用 nibabel 读取数据
np_data = np.asanyarray(image_lr.dataobj)  # 转成 numpy 的格式
dims = np_data.shape  # 三个维度的大小
# pixdim 中储存的信息是每个维度网格的间距 grid spacing ， unit per dimension
# more info in https://brainder.org/2012/09/23/the-nifti-file-format/
grid_spacing = image_lr.header["pixdim"][1:4]

# 初始化一个对象 用于储存 上面读入的数据
# more info in https://vtk.org/doc/nightly/html/classvtkImageData.html
image = vtk.vtkImageData()
# 设置维度
image.SetDimensions(dims[0], dims[1], dims[2])
# 设置网格间隔
image.SetSpacing(grid_spacing[0], grid_spacing[1], grid_spacing[2])
# 设置数据格式 以及 number of components
image.AllocateScalars(vtk.VTK_INT, 1)

# 最后一步 将三维体中的每一个 scalar 数值填jin 上面创建的 image 对象中的对应位置
for x in range(dims[0]):
    for y in range(dims[1]):
        for z in range(dims[2]):
            image.SetScalarComponentFromDouble(x, y, z, 0, np_data[x, y, z])
```

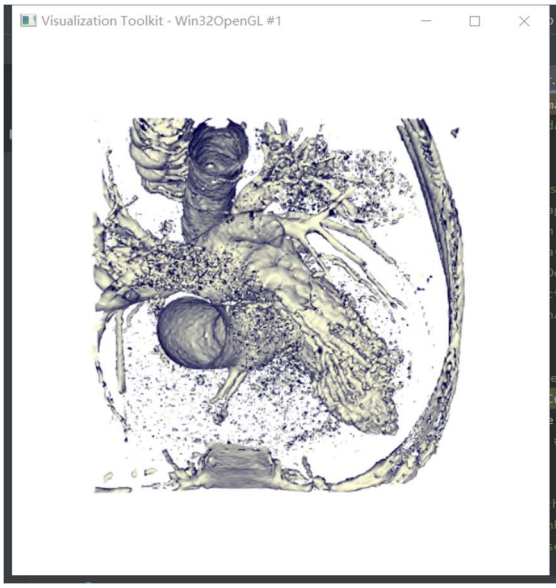
另外一种方法是在学习 VTK 官方文档时发现的：**直接使用 vtkNIFTIImageReader 进行读取**，代码如下:

```
image = vtk.vtkNIFTIImageReader()
image.SetFileName("./data/image_lr.nii.gz")
image.Update()
```

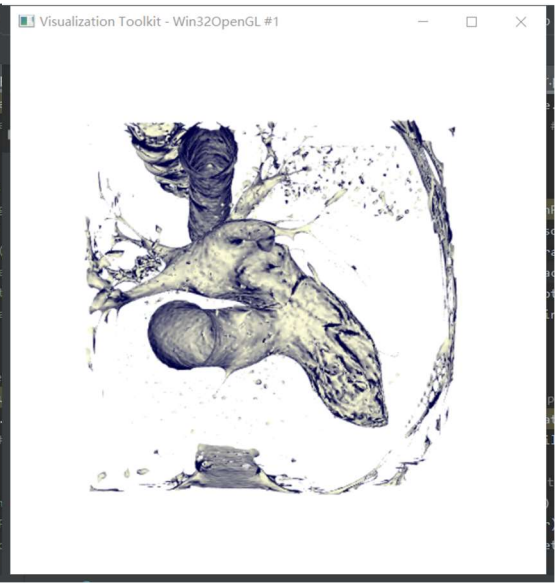
- **其他代码**如下:

```
# 对输入图像进行 Gaussian Smooth
filterImg = vtk.vtkImageGaussianSmooth()
# set input connection from the reader
filterImg.SetInputData(image.GetOutput())
filterImg.SetStandardDeviation(1, 1, 1)
filterImg.SetRadiusFactors(3, 3, 3)
filterImg.SetDimensionality(3)
filterImg.Update()

# 2. 利用 marching cubes 算法提取等值面  -------filter
iso = vtk.vtkMarchingCubes()
iso.SetInputData(filterImg.GetOutput())
iso.SetValue(0, 150)  # 设置第一个等值面的阈值
# 第一个参数为 i 个等值面，  第二个参数为该等值面的 value
iso.Update()

# 用laplacian smooth
Filter = vtk.vtkSmoothPolyDataFilter()
Filter.SetInputData(iso.GetOutput())
Filter.SetNumberOfIterations(20)
Filter.SetRelaxationFactor(1)
Filter.FeatureEdgeSmoothingOff()
Filter.BoundarySmoothingOff()
Filter.Update()


# 设置 mapper ------mapper
mapper = vtk.vtkPolyDataMapper()
mapper.SetInputData(Filter.GetOutput())

# 设置actor   ------actor
actor = vtk.vtkActor()
actor.SetMapper(mapper)
actor.GetProperty().SetColor(1, 1, 0)  # 设置acotr 的颜色 和 光照 等属性
actor.GetProperty().SetAmbient(0.25)
actor.GetProperty().SetDiffuse(0.6)
actor.GetProperty().SetSpecular(1.0)

# 设置渲染器  -------renderer
ren = vtk.vtkRenderer()
ren.SetBackground(1, 1, 1)
ren.AddActor(actor)

# 设置窗口  -------renderWindow
renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren)
renWin.SetSize(512, 512)

# 设置交互器  -------interactor
iren = vtk.vtkRenderWindowInteractor()

# 启动
iren.SetRenderWindow(renWin)
iren.Initialize()
renWin.Render()
iren.Start()
```

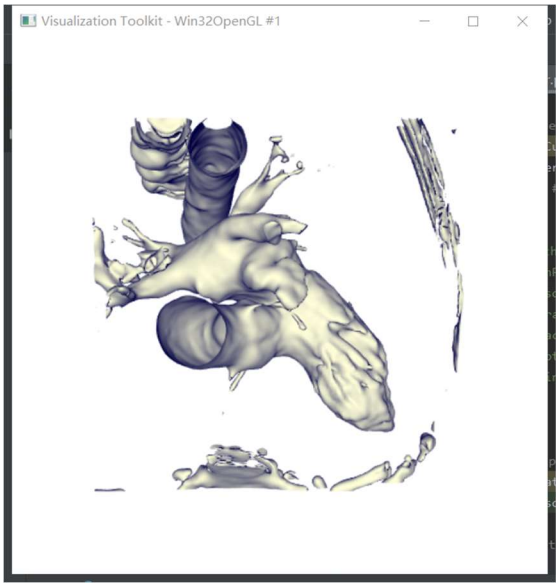
得到的可视化结果如下:



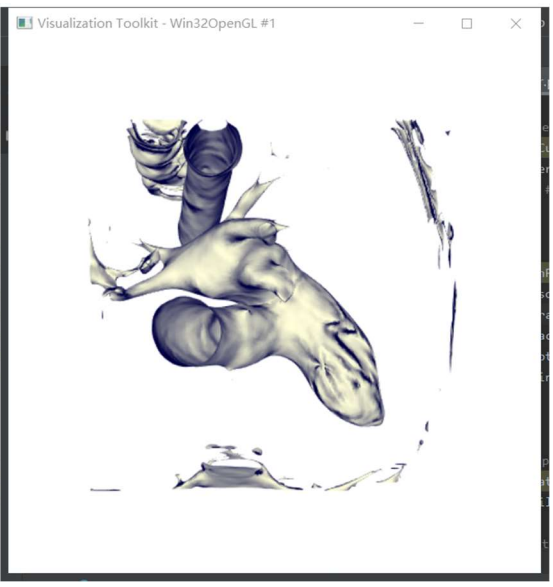
图一



图二



图三



图四

说明：

图一为 运用 marching cubes 算法得到的等值面渲染结果；

图二为 **仅**运用 vtkSmoothPolyDataFilter 算法得到的平滑结果；

图三为 **仅**运用 vtkImageGaussianSmooth 算法得到的平滑结果；

图四为 先运用了 vtkImageGaussianSmooth 后又运用了 vtkSmoothPolyDataFilter 算法得到的平滑结果。

结果分析：我们可以看到，在 matching cubes 之后使用 vtkSmoothPolyDataFilter 算法能够去除大部分的碎片化面单元，但仍会有所保留（**去除的效果可以由 relaxaion factor 以及 number of iteration 控制**）；

在 marching cubes 算法之前使用 vtkImageGuassianSmooth 能够对图像起到较好的光滑作用，几乎去掉了所有的碎片化面单元，**这里使用的是 方差为 （1，1，1），大小为 3x3x3 的高斯卷积核。**

虽然以上两个方法结合使用，能使图像更加平滑，但也同时丢失了更多的细节信息。