数据可视化 HW7

钟诚 16307110259

June 10, 2020

1 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.

通过 VTK 官网，我学习了 VTK 在 Python 上的应用，图片渲染一般遵循以下几个步骤，source->filter(MC 算法)->mapper->actor->render->renderwindow->interactor 代码实现如下：

2 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。

2.1 等值面渲染

通过 MatchingCube 算法，我们可以对体数据进行等值面渲染，因为在第三题中，也要用到相应的 nii 数据进行碎片化面单元的清除，所以在第二题中我也使用了这个体数据，渲染代码如下：

```
def isosurfaceRendering():
    # source->filter(MC算法)->mapper->actor->render->renderwindow->interactor
    # 提取出nii.gz其中的数据，并作为Inputdata
    img1 = nib.load('./nii_image/image_lr.nii.gz')
    img1_data = img1.get_data()
    dims = img1.shape
    spacing = (img1.header['pixdim'][1], img1.header['pixdim'][2],
        img1.header['pixdim'][3])
    image = vtk.vtkImageData()
    image.SetDimensions( dims[0], dims[1], dims[2])
    image.SetOrigin(0,0,0)

    if vtk.VTK_MAJOR_VERSION <= 5:
       image.SetNumberOfScalarComponents(1)
       image.SetScalarTypeToDouble()
    else:
       image.AllocateScalars(vtk.VTK_DOUBLE, 1)

    for z in range(dims[2]):
        for y in range(dims[1]):
            for x in range(dims[0]):
                scalardata = img1_data[x][y][z]
                image.SetScalarComponentFromDouble(x, y, z, 0, scalardata)



    #利用封装好的MC算法抽取等值面，对应filter
```

```
Extractor = vtk.vtkMarchingCubes()
Extractor.SetInputData(image)
Extractor.SetValue(0, 150)

# 平滑操作
'''
smoother = vtk.vtkSmoothPolyDataFilter()
smoother.SetInputConnection(Extractor.GetOutputPort())
smoother.SetNumberOfIterations(500)
'''
# 剔除旧的或废除的数据单元，提高绘制速度，对应filter
Stripper = vtk.vtkStripper()
Stripper.SetInputConnection(Extractor.GetOutputPort())
#Stripper.SetInputConnection(smoother.GetOutputPort())

# 建立映射，对应mapper
mapper = vtk.vtkPolyDataMapper()
mapper.SetInputConnection(Stripper.GetOutputPort())

# 建立角色以及属性的设置，对应actor
actor = vtk.vtkActor()
actor.SetMapper(mapper)
# 角色的颜色设置
actor.GetProperty().SetDiffuseColor(1, 1, 0)
actor.GetProperty().SetOpacity(0.8)
actor.GetProperty().SetAmbient(0.25)
actor.GetProperty().SetDiffuse(0.6)
actor.GetProperty().SetSpecular(1.0)

# 设置颜色
mapper.ScalarVisibilityOff()
# 定义舞台，也就是渲染器，对应render
renderer = vtk.vtkRenderer()
renderer.SetBackground(1, 1, 1)
renderer.AddActor(actor)

# 定义整个剧院(应用窗口)，对应renderwindow
rewin = vtk.vtkRenderWindow()

# 将舞台添加到剧院中
rewin.AddRenderer(renderer)
rewin.SetSize(250, 250)
interactor = vtk.vtkRenderWindowInteractor()
interactor.SetRenderWindow(rewin)
interactor.Initialize()
rewin.Render()
interactor.Start()
```

实验效果如下：
等值面渲染的某一个截图如下所示，通过运行作业中相应代码，你可以获得完整的三维图像：

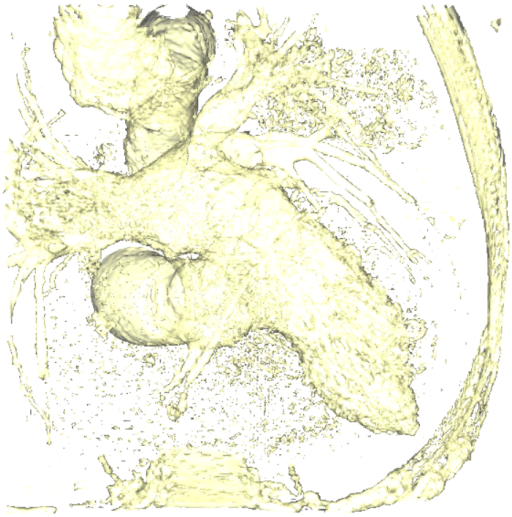


Figure 1: 等值面渲染

2.2 体渲染

与等值面渲染不同的是，体渲染需要我们通过分段函数定义基于像素强度的颜色、不透明度和阴影部分，从而在视觉上对体数据进行重建，代码实现如下：

```
def volumeRender():
    # Create the standard renderer, render window
    # and interactor.
    ren1 = vtk.vtkRenderer()

    renWin = vtk.vtkRenderWindow()
    renWin.AddRenderer(ren1)

    iren = vtk.vtkRenderWindowInteractor()
    iren.SetRenderWindow(renWin)

    # Create the reader for the data.
    reader = vtk.vtkStructuredPointsReader()
    reader.SetFileName('./ironProt.vtk')

    # Create transfer mapping scalar value to opacity.
    volumeOpacity = vtk.vtkPiecewiseFunction()
    volumeOpacity.AddPoint(50, 0)
    volumeOpacity.AddPoint(150, 0.4)
    volumeOpacity.AddPoint(200, 0.7)
    volumeOpacity.AddPoint(255, 1)

    # Create transfer mapping scalar value to color.
    volumeColor = vtk.vtkColorTransferFunction()
    volumeColor.AddRGBPoint(0.0, 25.0, 25.0, 25.0)
    volumeColor.AddRGBPoint(64.0, 100.0, 100.0, 100.0)
    volumeColor.AddRGBPoint(128.0, 150.0, 150.0, 150.0)
```

```
volumeColor.AddRGBPoint(192.0, 200.0, 200.0, 200.0)

# The property describes how the data will look like.
volumeProperty = vtk.vtkVolumeProperty()
volumeProperty.SetColor(volumeColor)
volumeProperty.SetScalarOpacity(volumeOpacity)
volumeProperty.ShadeOn()
volumeProperty.SetInterpolationTypeToLinear()

# render the data.
volumeMapper = vtk.vtkFixedPointVolumeRayCastMapper()
volumeMapper.SetInputConnection(reader.GetOutputPort())

# The volume holds the mapper and the property and
# can be used to position/orient the volume.
volume = vtk.vtkVolume()
volume.SetMapper(volumeMapper)
volume.SetProperty(volumeProperty)

colors = vtk.vtkNamedColors()
# render the data
ren1.AddVolume(volume)

# set the background
ren1.SetBackground(colors.GetColor3d("White"))
ren1.GetActiveCamera().Azimuth(0)
ren1.GetActiveCamera().Elevation(15)
ren1.ResetCameraClippingRange()
ren1.ResetCamera()

renWin.SetSize(600, 600)
renWin.Render()

iren.Start()
```

同样的，这里将会展示渲染结果的某个截面数据，你可以通过运行代码看到完整的渲染结果。

3. 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

在 vtk 中，可以使用 vtk.vtkSmoothPolyDataFilter() 对数据进行平滑并消除等值面渲染结果中的碎片化面单元，对此，我们需要加入这个类并对等值面渲染代码中的接口做一些更改，如下所示：

```
...
    #利用封装好的MC算法抽取等值面，对应filter
    Extractor = vtk.vtkMarchingCubes()
    Extractor.SetInputData(image)
    Extractor.SetValue(0, 150)
    # 平滑操作
    smoother = vtk.vtkSmoothPolyDataFilter()
    smoother.SetInputConnection(Extractor.GetOutputPort())
    smoother.SetNumberOfIterations(500)
```

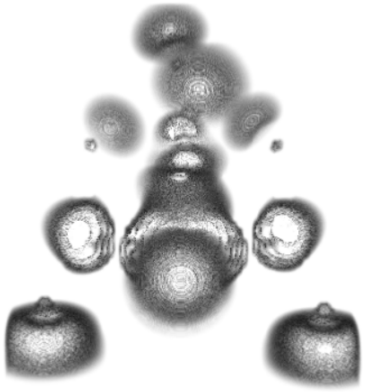
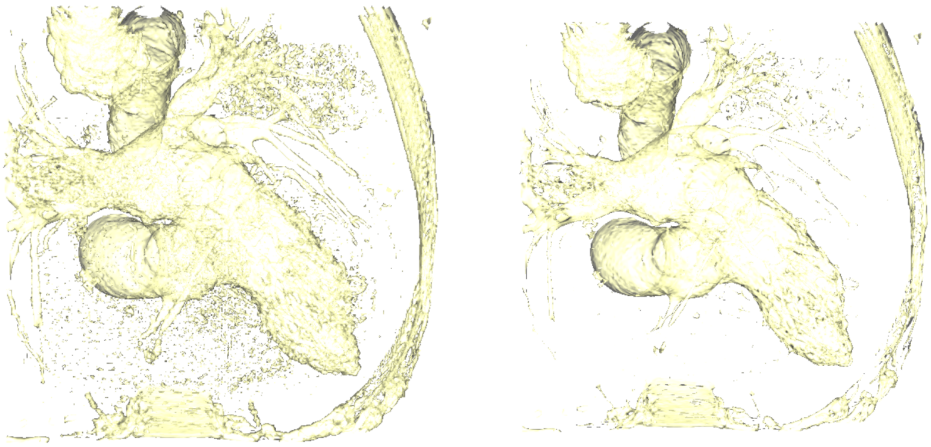


Figure 2: 体渲染

```
    # 剔除旧的或废除的数据单元，提高绘制速度，对应filter
    Stripper = vtk.vtkStripper()
    #Stripper.SetInputConnection(Extractor.GetOutputPort())
    Stripper.SetInputConnection(smoother.GetOutputPort())

    # 建立映射，对应mapper
    mapper = vtk.vtkPolyDataMapper()
    mapper.SetInputConnection(Stripper.GetOutputPort())
...
```

渲染结果对比如下，可见碎片化的面单元已被去除：



(a) 平滑前　　(b) 平滑后

Figure 3: 平滑前后图像的变化