医 学 影 像 分 析（实验二）

题目：实验二：基于 VTK 的医学图像可视化算法

学生姓名：王天也

学　　号：21S010051

所在学院：经济与管理学院

专业名称：管理科学与工程

2022 年 5 月

目　录

1 实验二，基于 VTK 的医学图像可视化算法

1.1 实验目的

编程实现基于 VTK 的医学图像可视化算法，熟悉 VTK 框架和几种可视化算法实现方法。

1.2 VTK 医学图像

可视化工具包（VTK）是一个开源、免费提供的软件系统，用于 3D 计算机图形学、建模、图像处理、体积渲染、科学可视化和 2D 绘图。它支持各种可视化算法和高级建模技术，并分别利用线程和分布式内存并行处理来提高速度和可扩展性。VTK 被设计为与平台无关。这意味着它几乎可以在任何地方运行，包括在 Linux，Windows 和 Mac 上。在网络上; 和移动设备上。

VTK 采用 Kitware 的质量软件流程，包括 CMake，CTest，CDash 和 CPack 来构建，测试和打包系统。结合强大的分布式开发人员社区，结果是非常高质量，健壮的代码。VTK 的核心功能以 C++ 编写，以最大限度地提高效率。此功能被包装到其他语言绑定中，以将其公开给更广泛的受众。与 Python 的互操作性特别完善。作为开源软件，VTK 可以自由用于任何目的。从技术上讲，VTK 具有 BSD 风格的许可证，该许可证对开放和闭源应用程序施加的限制最小。

VTK 最初是教科书 The Visualization Toolkit An Object-Oriented Approach to 3D Graphics 的一部分。Will Schroeder、Ken Martin 和 Bill Lorensen——三位图形和可视化研究人员——从 1993 年 12 月开始，在当时的雇主 GE RD 的法律许可下，在他们自己的时间编写了这本书和配套软件。这本书的动机是与其他研究人员合作，开发一个用于创建尖端可视化和图形应用程序的开放框架。VTK 源于作者在 GE 的经历，特别是 LYMB 面向对象图形系统。其他影响包括 Schroeder 等人开发的 VISAGE 可视化系统。al; 伦斯勒理工学院开发的发条机面向对象的计算机动画系统; 以及 Bill Lorensen 合著的《面向对象建模与设计》（Object-Oriented Modeling and Design）一书。在 VTK 的核心编写完成后，世界各地的用户和开发人员开始改进并将系统应用于现实世界的问题。特别是 GE 医疗系统和其他 GE 业务为该系统做出了贡献，Penny Rheinghans 博士等研究人员开始用这本书进行教学。其他早期倡导者包括洛斯阿拉莫斯国家实验室的 Jim Ahrens 以及慷慨的石油和天然气支持者。为了解决正在成为一个庞大，活跃和世界范围的社区，Ken 和 Will 以及 Lisa Avila，Charles Law 和 Bill Hoffman 于 1998 年离开 GE，成立了 Kitware，Inc.。从那时起，数百名额外的开发人员已经将 VTK 变成了现在世界上首屈一指的可视化系统。例如，桑迪亚国家实验室（Sandia National Laboratories）一

直是 VTK 中 2D 图表和信息可视化的坚定支持者和共同开发者。

1. 3 实验原理

VTK 应用程序使用过滤器处理数据。每个过滤器都会检查它接收到的数据并生成派生数据。一组连接的过滤器形成一个数据流网络。可配置的网络将原始数据转换为视觉上更易于理解的格式。

VTK 在底层图形库（大部分是 OpenGL）上添加了一个渲染抽象层。这个更高的级别简化了创建引人注目的可视化的任务。

VTK 的核心数据模型能够代表几乎所有与物理科学相关的现实世界问题。基本数据结构特别适合涉及有限差分和有限元解决方案的医学成像和工程工作。

交互帮助您了解数据的内容、形状和含义。在 VTK 中，3D 小部件、交互器和 2D 小部件库（如 Qt）的接口使您能够将全面的用户交互添加到您的程序中。

VTK 具有用于表格数据的全套 2D 绘图和图表类型。VTK 的拣选和选择功能可帮助您以交互方式查询数据。此外，VTK 与 Python 的互操作性非常好，包括 Matplotlib。

VTK 对 MPI 下的可扩展分布式内存并行处理具有出色的支持。更重要的是，许多 VTK 过滤器通过 vtkSMP（用于粗粒度线程）和 vtk-m（用于多核和 GPU 架构上的细粒度处理）实现更细粒度的并行性。

1. 4 实验流程

因本人为初学者，所以仅创建一个最简单的圆锥体。

以下为使用到的 vtk 程序库中的函数：vtkmodules.vtkRenderingOpenGL2 vtkmodules.vtkCommonColor vtkmodules.vtkRenderingCore vtkmodules.vtkFiltersSources

1. 第一步为背景选择颜色

代码中使用到的函数：vtkNamedColors() 该函数用于创建一个 vtkNamedColors 的实例，我们将使用此选项为对象和背景选择颜色。

2. 创建圆锥，设置圆锥属性值

vtkConeSource() 该函数表示生成圆锥数据，其中，圆锥数据包含几个部分：（1）底面圆中心（2）底面半径（3）圆锥角度（和半径相似的功能）（4）圆锥高度 SetCenter: 设置圆底圆底底面中心位置 SetRadius: 设置圆底面半径 SetAngle: 设置圆锥顶角，单位为 ° SetHeight: 设置圆锥高度 SetDirection；设置圆锥方向

本实验设置圆锥的高度为 3.0，底面半径为 1.0，分辨率为 10

3. 创建映射器

我们使用映射器进程对象终止管道。（中间筛选器（如 vtkShrinkPolyData）可以插入到源和映射器之间。我们创建一个 vtkPolyDataMapper 实例，以将多边形数据映射到图形基元中。我们将锥体源的输出连接到此映射器的输入。

4. 创建执行组件

创建一个执行组件来表示圆锥体。执行组件协调映射器的图形基元的呈现。参与者还通过 vtkProperty 实例引用属性，并包括内部转换矩阵。我们将这个动画的映射器设置为我们上面创建的锥形映射器。

5. 创建渲染器

创建渲染器并向其分配角色。渲染器就像一个视口。它是屏幕上窗口的一部分或全部，我负责绘制它所拥有的动画。我们还在此处设置了背景色。

6. 创建循环语句，宣传椎体全部渲染

现在，我们循环超过 360 度，每次都渲染锥体。

7. 算法完成

运行完毕实验图，如图所示展示圆锥体的不同角度的形态

图 1.1　形态 1

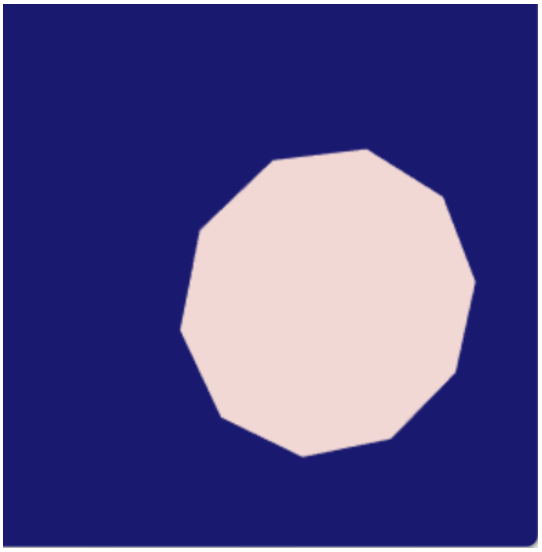

图 1.2　形态 2

图 1.3　形态 13

1. 5 3D 圆锥体代码块

Listing 1.1: python 代码

```python
#!/usr/bin/env python




import vtkmodules.vtkInteractionStyle
# noinspection PyUnresolvedReferences
import vtkmodules.vtkRenderingOpenGL2
from vtkmodules.vtkCommonColor import vtkNamedColors
from vtkmodules.vtkFiltersSources import vtkConeSource
from vtkmodules.vtkRenderingCore import (
    vtkActor,
    vtkPolyDataMapper,
    vtkRenderWindow,
    vtkRenderer,
    vtkRenderWindowInteractor

)


def main(argv):
    #
    # 接下来，我们创建一个 vtkNamedColors 的实例，我们将使用此选项为对象和背景选择颜色。
    #
    colors = vtkNamedColors()
    bkg =  map(lambda x: x / 255.0, [26, 51, 102, 255])
    colors.SetColor("BkgColor", *bkg)
    #
    # 现在，我们创建一个 vtkConeSource 的实例并设置其一些属性。
    # vtkConeSource “cone” 的实例是可视化管道的一部分（它是一个源进程对象），
    # 它生成其他过滤器可以处理的数据（输出类型为vtkPolyData）。
    #
    cone = vtkConeSource()
    cone.SetHeight(3.0)
    cone.SetRadius(1.0)
    cone.SetResolution(10)
```

```
37
38      #
39      # 在此示例中，我们使用映射器进程对象终止管道。
40      #  （中间筛选器（如 vtkShrinkPolyData）可以插入到源和映射器之间。
41      #  我们创建一个 vtkPolyDataMapper 实例，
42      #  以将多边形数据映射到图形基元中。
43      #  我们将锥体源的输出连接到此映射器的输入。
44      #
45      coneMapper = vtkPolyDataMapper()
46      coneMapper.SetInputConnection(cone.GetOutputPort())
47
48      #
49      # 创建一个执行组件来表示圆锥体。执行组件协调映射器的图形基元的呈现。
50      # 参与者还通过 vtkProperty 实例引用属性，并包括内部转换矩阵。
51      # 我们将这个演员的映射器设置为我们上面创建的锥形映射器。
52      #
53      coneActor = vtkActor()
54      coneActor.SetMapper(coneMapper)
55      coneActor.GetProperty().SetColor(colors.GetColor3d('MistyRose'))
56
57      #
58      #创建渲染器并向其分配角色。渲染器就像一个视口。
59      # 它是屏幕上窗口的一部分或全部，
60      # 负责绘制它所拥有的演员。
61      # 我们还在此处设置了背景色。
62      #
63      ren1 = vtkRenderer()
64      ren1.AddActor(coneActor)
65      ren1.SetBackground(colors.GetColor3d('MidnightBlue'))
66
67      # 最后，我们创建渲染窗口，该窗口将显示在屏幕上。
68      # 我们使用 AddRenderer 将渲染器放入渲染窗口中。我们还
69      # 将大小设置为 300 x 300 像素。
70      #
71      renWin = vtkRenderWindow()
72      renWin.AddRenderer(ren1)
73      renWin.SetSize(300, 300)
74      renWin.SetWindowName('Tutorial_Step1')
75
76      iren = vtkRenderWindowInteractor()
```

```
77      iren.SetRenderWindow(renWin)
78
79      iren.Initialize()
80      #
81      # 现在，我们循环超过 360 度，每次都渲染锥体。
82      #
83      for i in  range(0, 360):
84          # 渲染图像
85          renWin.Render()
86          # 将活动相机旋转一度。
87          ren1.GetActiveCamera().Azimuth(1)
88      iren.Start()
89
90
91  if __name__ == '__main__':
92      import sys
93
94      main(sys.argv)
```