Scientific Visualization

VTK - Direct Volume Rendering

Anand Kamble
Department of Scientific Computing
Florida State University

1 Introduction

This report presents the implementation of direct volume rendering using the Visualization Toolkit (VTK) and Python to visualize a 3D scalar dataset of a mummy CT scan.

2 Implementation

The Python script for this assignment is written in the file ”main.py”. VTK version 9.0.1 and Python 3.6.8 were used for development and execution.

2.1 Execution

Make sure that you have the `mummy.120.vtk` file in the same folder where you are running command from listing 1.

Listing 1: bash

```
python3 main.py
```

2.2 Defining Colors

The colors which we will use for the visualization are defined in a dictionary.

Listing 2: main.py

```
Colors = {"SKIN": np.array([74, 194, 109]), "SKULL": np.array([159, 218, 58])}
```

2.3 Loading the Dataset

The mummy dataset in the ”mummy.128.vtk” file is loaded using the vtkStructuredPointsReader class as follows:

Listing 3: main.py

```
reader = vtk.vtkStructuredPointsReader()
reader.SetFileName("mummy.128.vtk")
```

2.4 Defining Transfer Functions

Transfer functions are defined to map scalar values to opacity and color values. The opacity transfer function assigns higher opacity to the scalar value range corresponding to the skin (70-90) and skull (100-120)

Listing 4: main.py

```
opacityTransferFunction = vtk.vtkPiecewiseFunction()
opacityTransferFunction.AddPoint(0, 0.0)
opacityTransferFunction.AddPoint(70, 0.2)
opacityTransferFunction.AddPoint(90, 0.5)
opacityTransferFunction.AddPoint(100, 0.7)
opacityTransferFunction.AddPoint(120, 0.9)
opacityTransferFunction.AddPoint(255, 1.0)
```

The color transfer function assigns different colors to the skin and skull:

Listing 5: main.py

```
colorTransferFunction = vtkColorTransferFunction()
colorTransferFunction.AddRGBPoint(0, 0.0, 0.0, 0.0)
colorTransferFunction.AddRGBPoint(70, *(Colors["SKIN"] / 255))
colorTransferFunction.AddRGBPoint(100, *(Colors["SKULL"] / 255))
colorTransferFunction.AddRGBPoint(255, 1.0, 1.0, 1.0)
```

2.5 Volume Rendering Pipeline

The volume rendering pipeline is set up by creating a volume mapper, volume property, and volume actor.

Listing 6: main.py

```
# Create the volume property
volumeProperty = vtkVolumeProperty()
volumeProperty.SetColor(colorTransferFunction)
volumeProperty.SetScalarOpacity(opacityTransferFunction)

# Create the volume mapper
volumeMapper = vtkFixedPointVolumeRayCastMapper()
volumeMapper.SetInputConnection(reader.GetOutputPort())

# Create the volume
volume = vtkVolume()
volume.SetMapper(volumeMapper)
volume.SetProperty(volumeProperty)
```

2.6 Rendering and Visualization

A renderer, render window, and interactor are created to visualize the volume rendering.

Listing 7: main.py

```
# Create the renderer
renderer = vtkRenderer()
renderer.AddVolume(volume)
renderer.SetBackground(0.0, 0.0, 0.0)

# Create the render window
renderWindow = vtkRenderWindow()
renderWindow.SetSize(800, 800)
renderWindow.AddRenderer(renderer)

# Create the interactor
interactor = vtkRenderWindowInteractor()
interactor.SetRenderWindow(renderWindow)
```

2.7 Revolving Camera

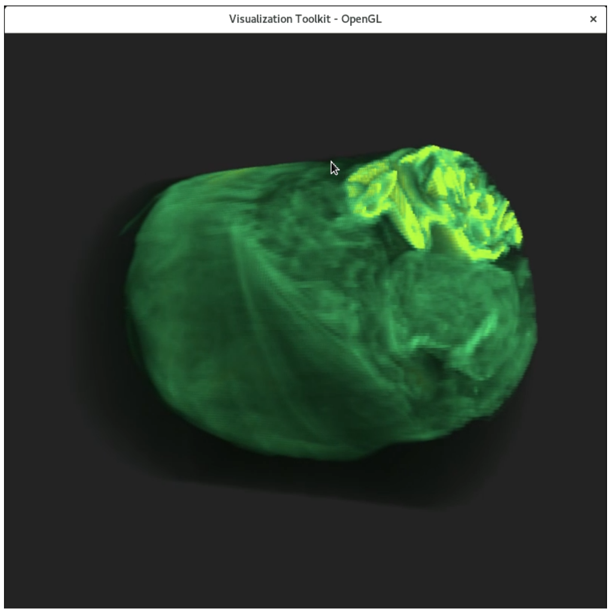
I have added a function to rotate the camera around the object which is done by using the observers from the interaction. Also, I'm using the Azimuth function to rotate the camera, which is a member function of the class vtkCamera.

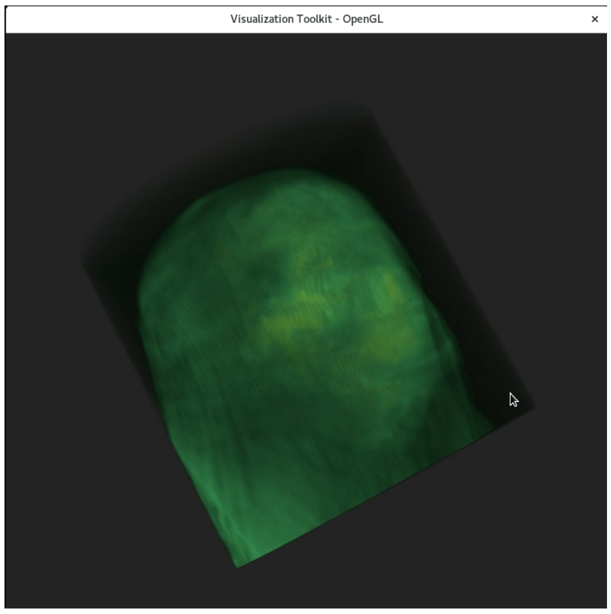
Listing 8: main.py

```python
timer_id = interactor.CreateRepeatingTimer(10)  # 10 milliseconds


def rotate_camera(obj, event):
    """
    Rotating the camera by a small amount every frame.
    Found the Azimuth function on:
    https://vtk.org/doc/nightly/html/classvtkCamera.html
    """

    camera = renderer.GetActiveCamera()
    camera.Azimuth(1.0)
    renderWindow.Render()


# Bind the rotate_camera function to the timer event
interactor.AddObserver(vtkCommand.TimerEvent, rotate_camera)
```

3 Results

After successful execution of the python script, we can see the following results.


Visualization Toolkit - OpenGL


Visualization Toolkit - OpenGL