Assignment #3: 3D Scalar Field Visualization

Due on October 7th, before midnight

Goals and Requirements:

The goals of this assignment include: (1) complete the cut planes for 3D volume data, (2) extract and visualize iso-surfaces from volumetric scalar fields, and (3) perform ray-casting direct volume rendering.

Two 3D volumetric data sets in the format of image data (or uniform structured grid) are provided. A skeleton code in Python is also available to help you get started.

You should submit your source code **and a writing report in a single .zip file** via Blackboard before the deadline.

Your report (required) should include your answers to the writing questions **AND** high-quality images captured from the individual visualization tasks listed below for the **two data sets** given. Also, please **specify the parameters** used to produce the corresponding visualization (like the iso-value for the iso-surface and the configuration file for the transfer function design), **especially for the bernard3D_Q data set**.

Tasks:

1. Writing questions (20 points)

1.1 (5 points) What are the potential issues during the computation of Marching Cubes?

1.2 (5 points) What is the fundamental difference between Raycasting and Splatting?

1.3 (5 points) What are the limitations of Raycasting? How will you improve Raycasting?

1.4 (5 points) Why designing proper transfer function for DVR is challenging?

2. 3D slicing (30 points)

Visualize three axis-aligned cut planes based on the user-specified locations. You can safely assume that these locations are integers corresponding to the indices of the XY, YZ, and XZ planes in the provided data. For example, Z=10 defines an XY plane with all the 3D points (or voxels) that have Z index 10.

You can use the following code to get the dimensions of the image data, `dim = [xdim, ydim, zdim]`

```
dim = reader.GetOutput().GetDimensions()
```

The following code shows how to specify an XY cut plane for a loaded 3D image data

```
# Create a mapper and assign it to the corresponding reader
xy_plane_Colors = vtk.vtkImageMapToColors()
xy_plane_Colors.SetInputConnection(reader.GetOutputPort())
xy_plane_Colors.SetLookupTable(bwLut)
xy_plane_Colors.Update()
```

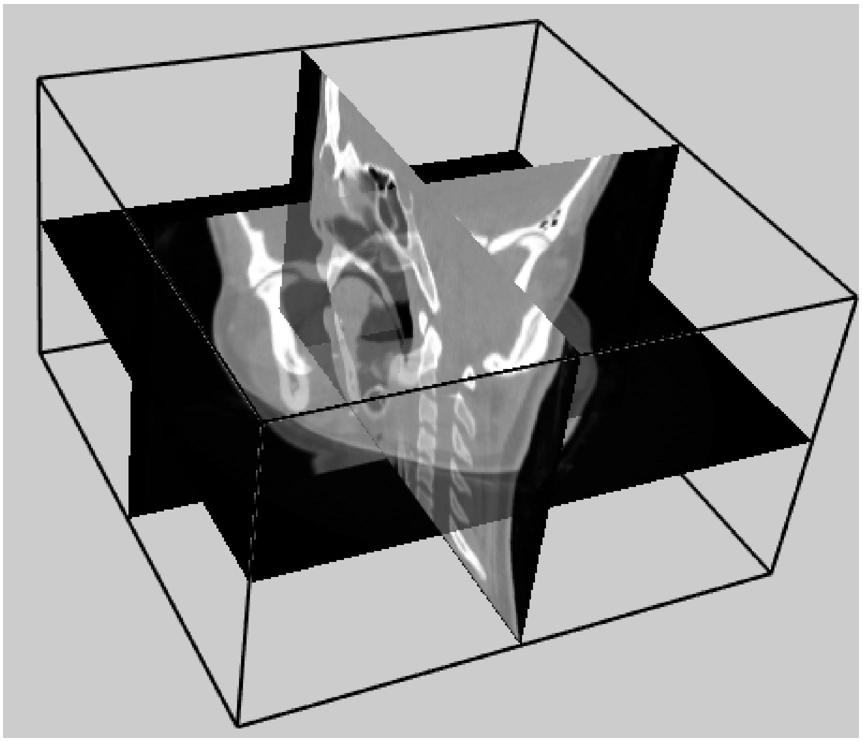
Note that `bwLut` is a black/white look up table that can be setup as follows. You can **re-use the color scale look up tables you created in Assignment 2** here, but this look up table must be setup first before it can be used. A place to set this look up table is when you load the data file (e.g., in the `open_vtk_file()` function).

```
# Create a black/white lookup table
bwLut = vtk.vtkLookupTable()
# YOU need to adjust the following range to address the dynamic range issue!
bwLut.SetTableRange(0, 2)
bwLut.SetSaturationRange(0, 0)
bwLut.SetHueRange(0, 0)
bwLut.SetValueRange(0, 1)
bwLut.Build()

# Create an actor for the XY plane
xy_plane = vtk.vtkImageActor()
xy_plane.GetMapper().SetInputConnection(xy_plane_Colors.GetOutputPort())
xy_plane.SetDisplayExtent(0, xdim-1, 0, ydim-1, current_zID, current_zID)
# Current_zID is a user-input integer within the range of [0, zdim-1]

# Add the actor to the renderer (ren is a vtkRenderer object)
ren.AddActor(xy_plane)
```

The following shows a screenshot of the three selected cut planes for the *FullHead* data.



3. Iso-surface Extraction and Visualization (20 points)

Extract and visualize an iso-surface based on the user input iso-value. You can use the `vtkMarchingCubes()` filter to achieve that.

```
isoSurfExtractor = vtk.vtkMarchingCubes()
```

Once you create a marching cubes filter, you can use the similar function `SetValue(0, iso-value)` as you used in Assignment 2 with the contour filter, and pass the iso-value that the user inputs.

To accelerate the rendering of the extracted iso-surface(s), it is recommended to use the `vtkStripper` to convert the individual triangles in the iso-surface into triangle strip.

```
isoSurfStripper = vtk.vtkStripper()
isoSurfStripper.SetInputConnection(isoSurfExtractor.GetOutputPort())
isoSurfStripper.Update()
```

Next, you will pass this stripper object to a `vtkPolyDataMapper()` mapper. Remember to call the `ScalarVisibilityOff()` function **IF** you want to specify a color for the surface.

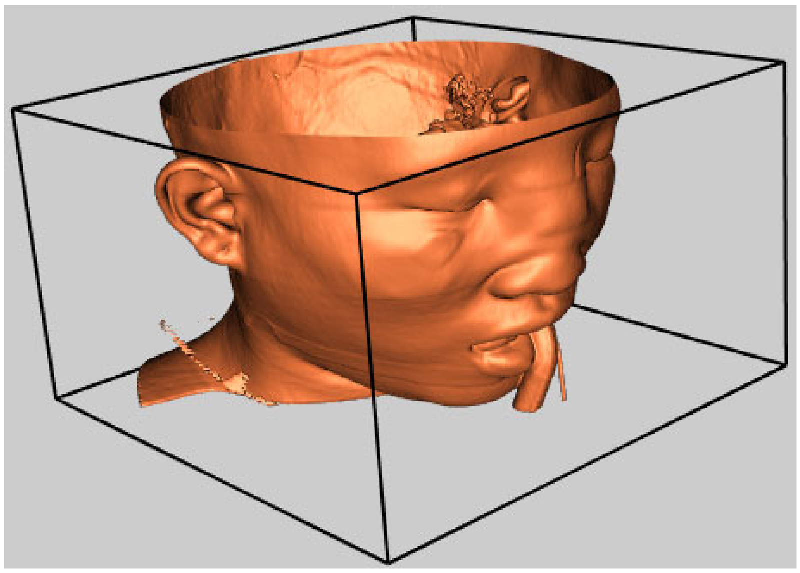
Next, create an actor for the iso-surface so that you can set its color and other materials. If you wish to use the lighting effect, you can use the following functions.

```
your_actor.GetProperty().SetDiffuseColor([the color you prefer])
your_actor.GetProperty().SetSpecular(.3)
your_actor.GetProperty().SetSpecularPower(20)
```

You can also specify the transparency of the extracted iso-surfaces, which can be useful when multiple iso-surfaces are shown at the same time.

```
your_actor.GetProperty().SetOpacity([your preferred opacity in [0, 1]])
```

The following shows a screenshot of the iso-surface corresponding to value 500 of the *FullHead* data. This iso-surface represents the skin of a patient.



4. DVR-Raycasting (30 points)

Compute and show the volume rendering using ray-casting. To achieve that, you can use the `vtkFixedPointVolumeRayCastMapper()` mapper or the `vtkSmartVolumeMapper()` mapper. The former is the standard raycasting DVR and the latter is the improved DVR. \alpha composition will be used. Note that for DVR, there is no filter needed as no geometry will be extracted from the data. So, you should directly connect the loaded data from the source to ray-casting mapper using the `SetInputConnection()` function of the mapper.

To compute the pixel colors, both color and opacity transfer functions need to be set.

The following shows an example code for the specification of the *color transfer function* for the FullHead data. The `vtkColorTransferFunction` is used. The `AddRGBPoint()` function of this class is used to add control points to the color transfer function. This function consists of four parameters, the first one specifies the iso-value and the last three provide the RGB color you want to assign to that value.

```
# The goal is to assign one color for flesh (between 500 and 1000)
# and another color for bone (1150 and over).
volumeColor = vtk.vtkColorTransferFunction()
volumeColor.AddRGBPoint(0, 0.0, 0.0, 0.0)
volumeColor.AddRGBPoint(500, 1.0, 0.5, 0.3)
volumeColor.AddRGBPoint(1000, 1.0, 0.5, 0.3)
volumeColor.AddRGBPoint(1150, 1.0, 1.0, 0.9)
```

The following shows an example code for the specification of the *opacity transfer function* (which is a 1D function) for the FullHead data. The `vtkPiecewiseFunction` is used. The `AddPoint` () function is used to add control points to this 1D function and provide the specific opacity values. Other opacity values for the data between these control points will be interpolated linear as the piecewise function is used here.

```
volumeScalarOpacity = vtk.vtkPiecewiseFunction()
volumeScalarOpacity.AddPoint(0, 0.00)
volumeScalarOpacity.AddPoint(500, 0.15) #Skin is more transparent
volumeScalarOpacity.AddPoint(1000, 0.15)
volumeScalarOpacity.AddPoint(1150, 0.85) #Bone get the highest opacity
```

NOTE that for both the color and opacity functions, the above locations of control points need to be adjusted by the user interactively to achieve the best rendering effect. Also, those scalar values (e.g., 500, 1000, and 1150) are data dependent. You must replace them based of the data range of the loaded data (e.g., for the bernard3D_Q data).

To facilitate the interactive specification of the above two transfer functions, a "rendering_config.txt" file is used that has the following format:

```
#Color
0, 0.0, 0.0, 0.0
500, 1.0, 0.5, 0.3
1000, 1.0, 0.5, 0.3
1150, 1.0, 1.0, 0.9
```

#Opacity
0, 0.00
500, 0.15
1000, 0.15
1150, 0.85

Each row under the #Color section corresponds to a control point for the color transfer function as explained earlier. The rows under the #Opacity section are similar. During run time, the user can open this rendering_config.txt file and modify the control points to adjust the volume rendering effect. Once the file is modified and save, it can be reloaded by pressing the "Load control points" button on the interface (see an example interface later) or uncheck and check the "Show Volume Rendering" checkbox as currently done in the skeleton code.

After you specify the above transfer functions, you need to add them to a `vtkVolumeProperty` object. The following shows an example of adding the previously defined transfer functions to the volume property object.

```
volumeProperty = vtk.vtkVolumeProperty()
volumeProperty.SetColor(volumeColor)
volumeProperty.SetScalarOpacity(volumeScalarOpacity)
volumeProperty.SetGradientOpacity(volumeGradientOpacity)
volumeProperty.SetInterpolationTypeToLinear()
```

If you wish to add illumination (or shading) effect to your volume rendering, you can do something like the following for your volume property object (after the above setting).

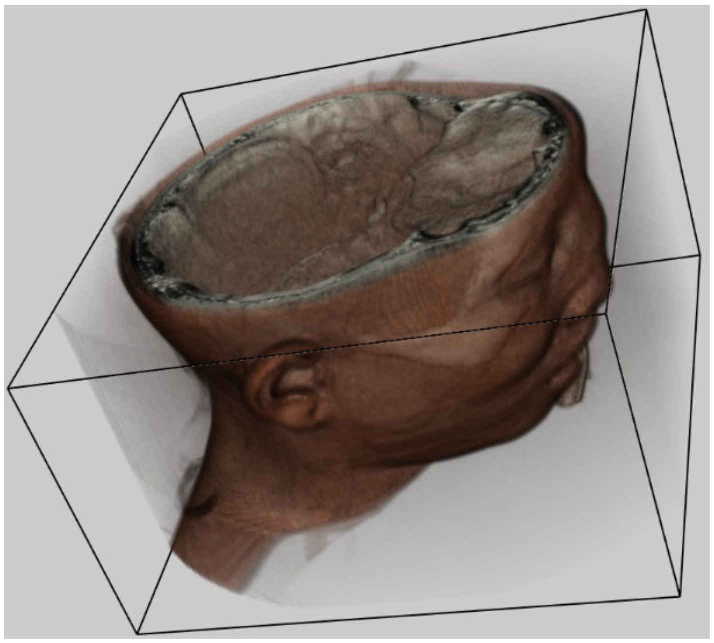
```
volumeProperty.ShadeOn()
volumeProperty.SetAmbient(0.4)
volumeProperty.SetDiffuse(0.6)
volumeProperty.SetSpecular(0.2)
```

Next, you should create a volume object using `vtkVolume` and add the ray-casting mapper and the corresponding volume property object to this volume object as follows.

```
volume = vtk.vtkVolume()
volume.SetMapper(volumeMapper)
volume.SetProperty(volumeProperty)
```

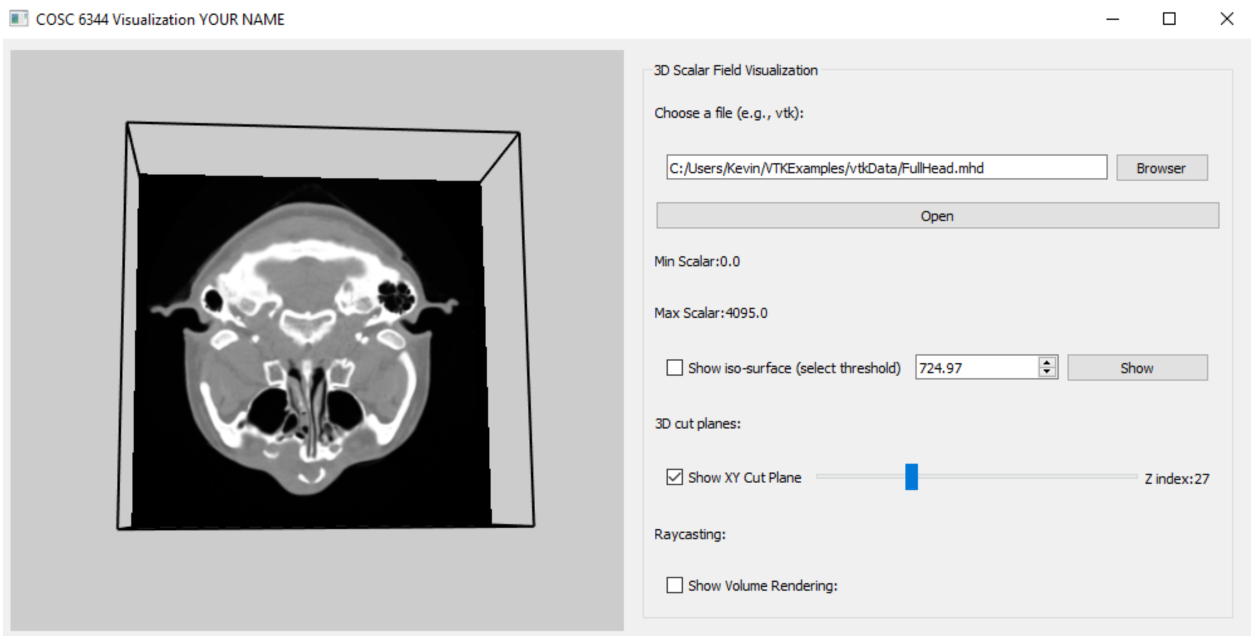
Finally, add this `volume` object to your renderer so that it can be rendered. The following provides a screenshot of the raycasting of the *FullHead* data with the above setting (and the rendering_config.txt file) for your reference.

Note that if you use the **`vtkSmartVolumeMapper`**`()`, which is recommended, you may see different (most likely better) rendering effect.



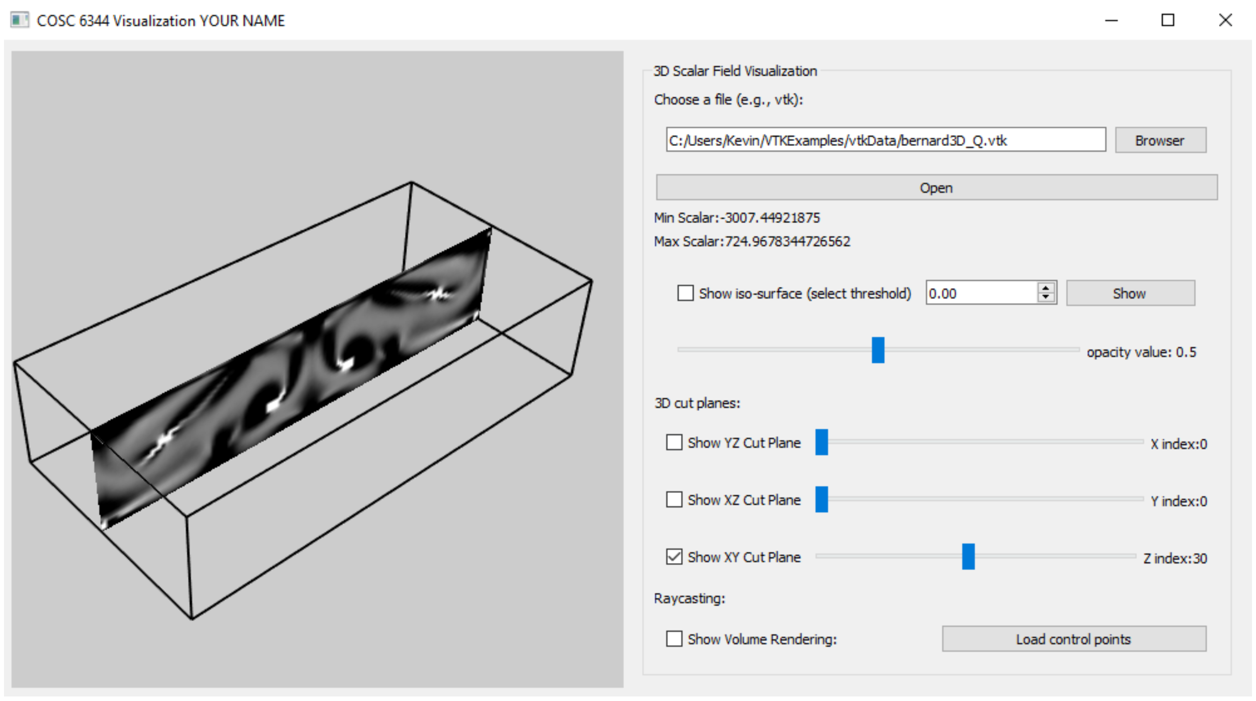
5. Graphical interface (GUI)

A skeleton code with a QT GUI is provided for you to get started. The interface of the skeleton program looks like below


COSC 6344 Visualization YOUR NAME
3D Scalar Field Visualization
Choose a file (e.g., vtk):
C:/Users/Kevin/VTKExamples/vtkData/FullHead.mhd
Browser
Open
Min Scalar:0.0
Max Scalar:4095.0
Show iso-surface (select threshold)
724.97
Show
3D cut planes:
Show XY Cut Plane
Z index:27
Raycasting:
Show Volume Rendering:

The interface for XY cut plane with the full functionality is provided to you as a reference. The interface for iso-surface extraction and volume rendering is also included in the skeleton code, but they currently do nothing as their respective functions are incomplete. You need to complete these functions as described earlier to see the respective visualization.

A more complete interface MAY look like the following


COSC 6344 Visualization YOUR NAME
3D Scalar Field Visualization
Browser
Open
Show
3D cut planes:
Raycasting:
Load control points

Choose a file (e.g., vtk): C:/Users/Kevin/VTKExamples/vtkData/bernard3D_Q.vtk

Min Scalar:-3007.44921875

Max Scalar:724.9678344726562

☐ Show iso-surface (select threshold) 0.00

opacity value: 0.5

☐ Show YZ Cut Plane X index:0

☐ Show XZ Cut Plane Y index:0

☑ Show XY Cut Plane Z index:30

☐ Show Volume Rendering:

But you should make your best decision and practice for the interface design.

Some simple instruction on PyQt and its widgets is provided in assignment 2 and the following small tutorial.

http://www2.cs.uh.edu/~chengu/Teaching/Fall2020/Assigns/PyQT_PyVTK%20Tutorial.pdf

A more detailed instruction and reference can be found below

https://doc.qt.io/qtforpython/contents.html

Grading rubric:

Tasks	*Total points*
1	20
2	30
3	20
4	30

The following shows a screen shot of a volume rendering for the bernard3D_Q data set for your reference. Again, you need to play with the transfer function designed via the rendering_config.txt file to achieve a volume rendering that reveals the characteristics of this data.