CS3895: VTK Homework Assignment

Jake Watson[1]

[1] Vanderbilt University, Nashville TN, 37212 USA

jacob.j.watson@vanderbilt.edu

Abstract. Create a rendering of a 3D brain data set with a colorful or Christmas-themed rendering that updates with a neighbor-vertex visitation method based on a timer callback function.

Keywords: Colorful, Brain.

1 Project

1.1 Introduction

The goal of this assignment is to make a Christmas-themed (or more colorful) color scheme such that a rendered brain is changing colors based on scalar values using a neighbor-vertex visitation method. A timer callback is implemented such that the scalar field gradually changes with the following neighbor-vertex visitation method:

1.2 Computational Efficiency: Decimation

A decimate mesh function [1, 2] was implemented to decimate/down-sample the mesh by 90% to increase computational efficiency and reduce computation time.

1.3 General Overview

The image data was read in using the vtkXMLPolyDataReader library [3]. The data was decimated using the decimate mesh function and used to create a mesh with the vtkPolyData library [4]. A mapper was implemented using the vtkPolyDataMapper library [5] to specify interface between the mesh data and graphics primitives. An actor was implemented [6] to represent the mesh in the rendering scene. A scalar array of type double [7] was created and assigned to the mesh for color rendering. Each scalar was initialized to the same value (0) to create a solid colored brain. A renderer [8], render window [9], and render window interactor [10] were implanted for rendering and visualization [1]. The color scheme, visitation method, and timer callback are discussed in further detail in the following sections.

1.4 Neighbor-Vertex Visitation Method

The following pseudocode describes the method implemented to visit and assess the scalar values of neighboring points, and to update (or not update) the scalar values, which changes the color. A random number generator [11] While the instructions say to render the initial scalar field to values of -1, I could not initially get the lookup table to accept negative scalar values, so the scalar field is initialized to 0 (unvisited), and my method follows the instructions of the assignment with the addition of changes to account for this. The class code [1] was referenced for retrieving neighboring vertices data. The following pseudocode describes the implemented neighbor-vertex visitation method:

```
1. Choose an initial seed vertex Vo. Set its scalar to $Xvo = 1$, indicating it was
   visited at iteration 0.
2. Each time the timer callback is triggered, the scalar field changes as follows.
   At iteration $i+1$, for each vertex V and its associated scalar value Xv,
     a. If vertex v currently has $Xv = 0$ (not yet visited), do nothing.
     b. If vertex v currently has $1 \leq Xv < i$ (already visited, and its neigh-
        bors previously handled), do nothing.
     c. If vertex X currently has $Xv = i$ (visited at the last iteration) identify
        all of V's neighboring vertices. Then, for each neighboring vertex
        V′:
          i. If neighboring vertex V′ has $Xv' \geq 1$ (already visited), do
             nothing.
          ii. If neighboring vertex V′ has $Xv' = 0$, set it to $Xv' = i+1$,
              such that it is visited at the current iteration.
```

1.5 Color Scheme: Randomized Rainbow and Christmas Colors

The brain was initially rendered to a scalar value for 0; The first value of the lookup table was manually set to red, which initially renders the brain in red. The background was set to pine green. These colors were implemented to the initial rendering to ensure the use of Christmas-theme colors. A lookup table [12] of random RGB values was implemented to create a very colorful rainbow-like scheme (for brownie points, according to the directions). The vtkMinimalStandardRandomSequence library [11] was used for random generation of red, green, and blue (RGB) values for each lookup table value [13].

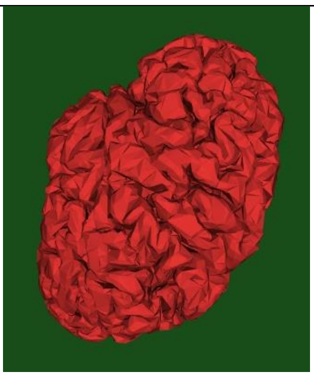
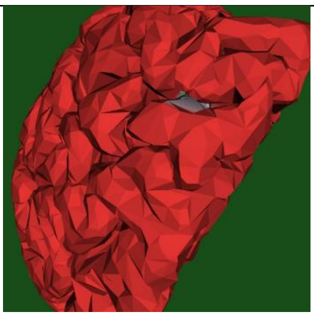
1.6 Timer Callback

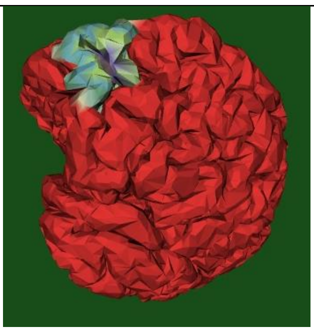
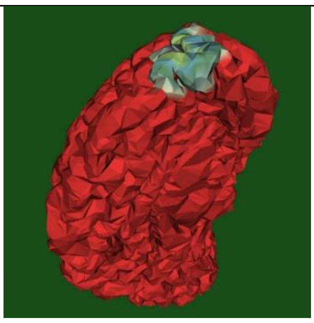
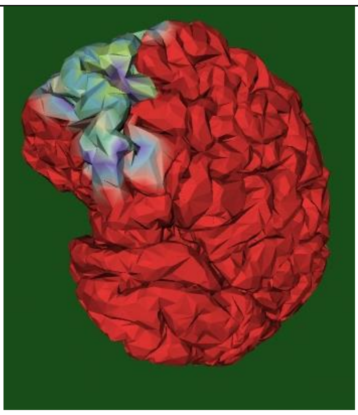
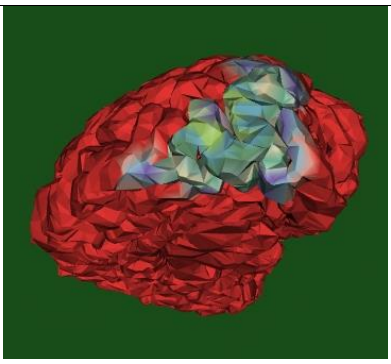
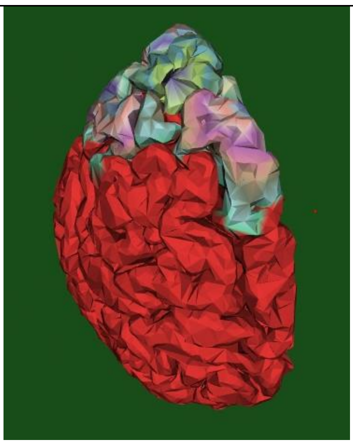
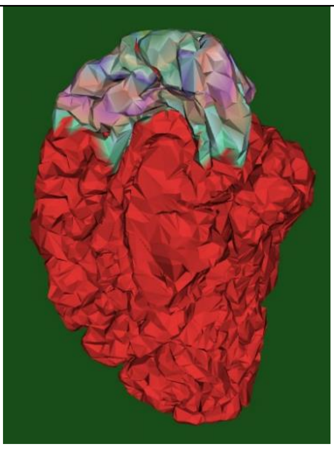
A timer callback [1, 14, 15] was implemented for the neighbor-visitation method and to update the visualization accordingly. The timer was set to update the rendering window every 5 seconds as iterations advanced based on the value of the incrementing timer.

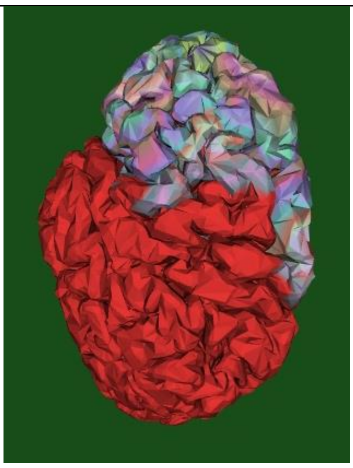
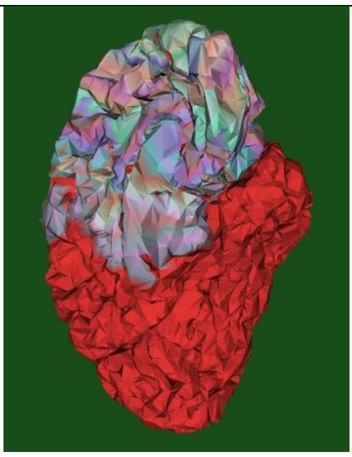
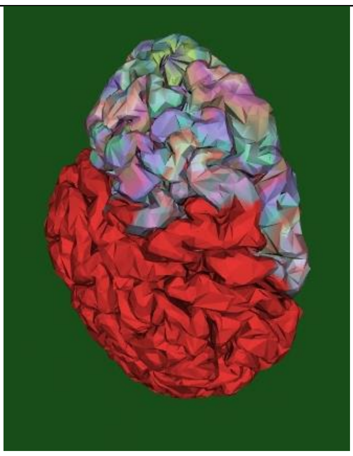
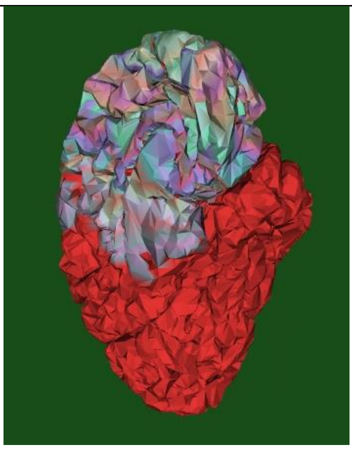
1.7 Results

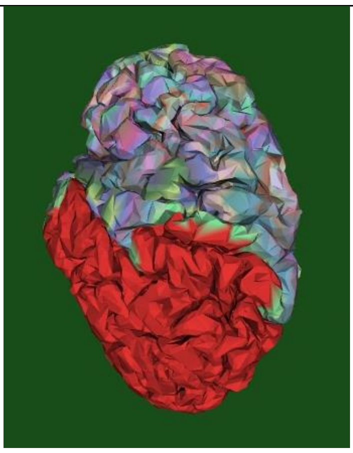
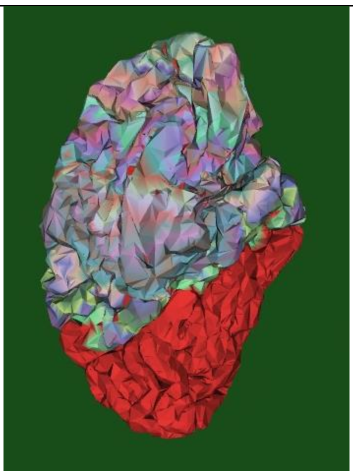
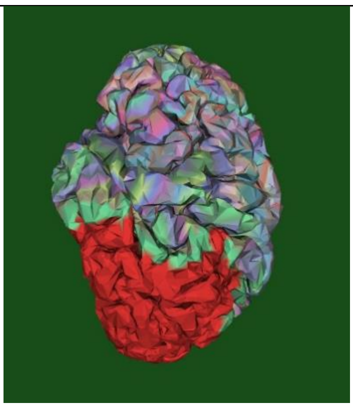
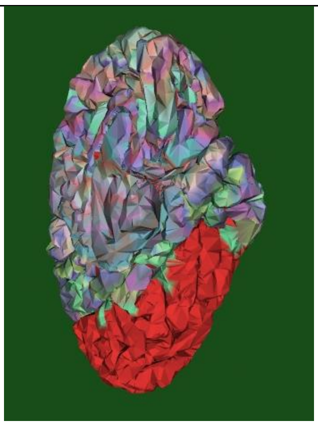
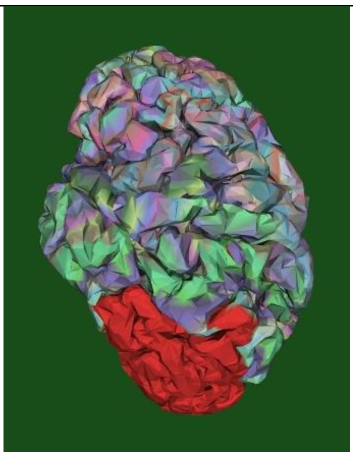
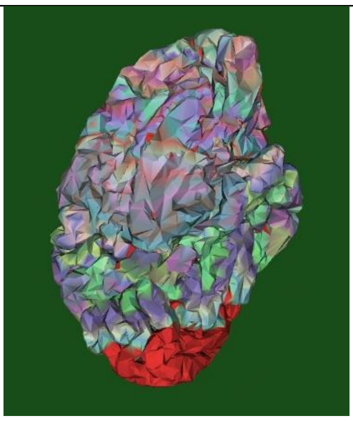
Figure 1 shows the brain changing color as it advances through iterations. As the iterations advance, for each vertex whose scalar value matches the value of the current iteration, the scalar values of its neighboring vertices are updated such that they have been ‘visited’. The neighboring ‘rings’ of the same color indicate that the visitation method functions properly. The brain was fully colored within 82 iterations.

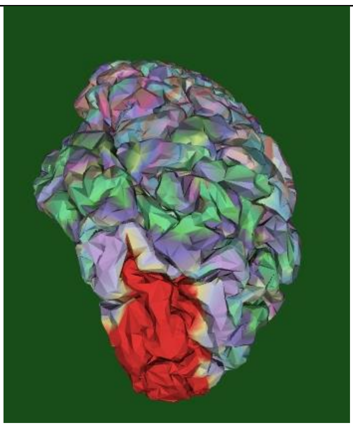
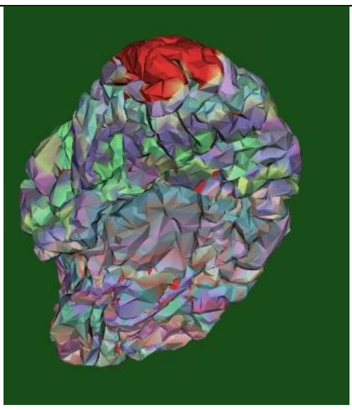
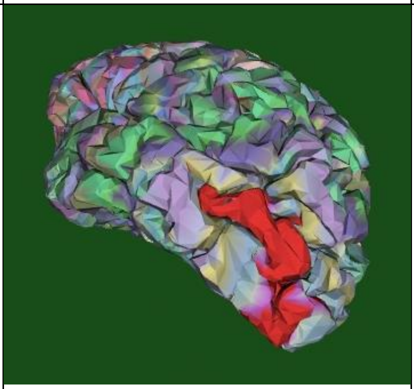
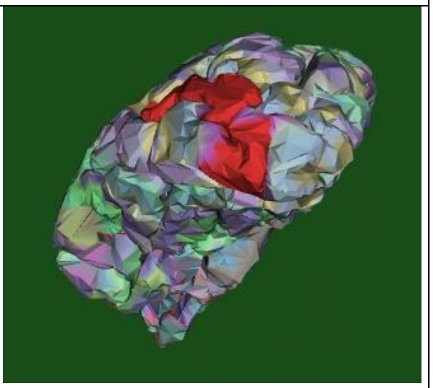
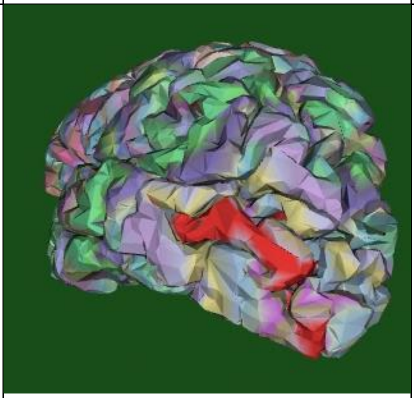
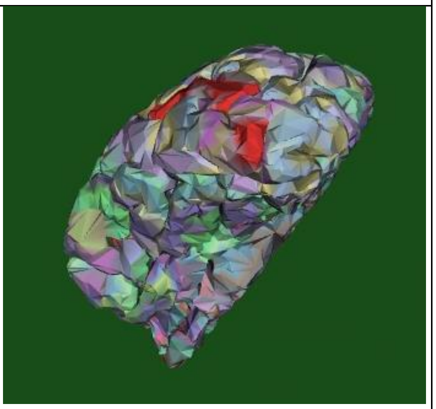
Figure 1. Views of the brain rendering as timer iterations progress.

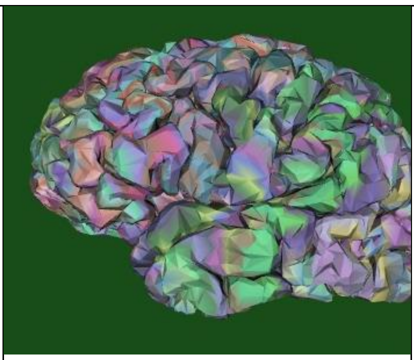
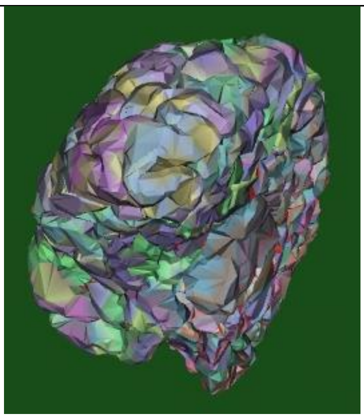
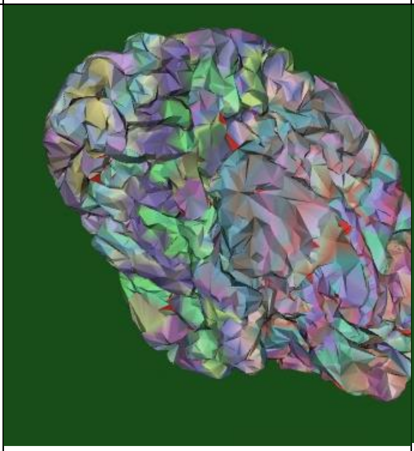
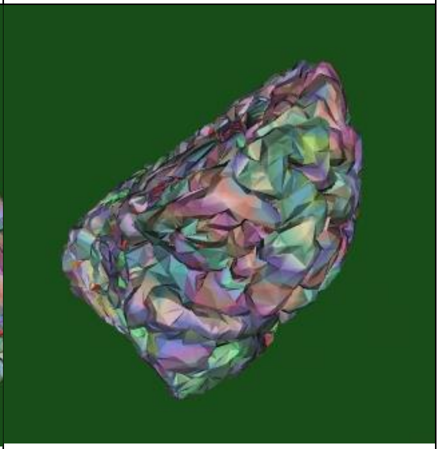
Itera-tion	Brain View 1	Brian View 2
0		
3		

10		
15		
22		

35		
42		

50		
58		
65		

70		
75		
77		

85		
More views of 85		

1.8 Challenges

Initial challenges stemmed from implementing the lookup table. Several functions in the lookup table library perform similarly to build the lookup table, and can overwrite each other. Using only the 'SetNumberOfTableValues' function resulting in the lookup table being built correctly. The timer callback was difficult to implement. I found little documentation for implementing a timer callback in the manners needed for this assignment. The images for **Figure 1** were taken with the timer callback removed from the code.

1.9 Conclusions

The neighbor-vertex visitation method and randomized RGB lookup table successfully rendered the brain into a rainbow color-scheme within 82 iterations. A timer callback was created and implemented; however, challenges prevented it from being used in the final brain rendering.

References

1. "VU-CS8395-fall2022." *GitHub*, https://github.com/VU-CS8395-Fall2022.
2. "VtkDecimatePro Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkDecimatePro.html.
3. "VTKXMLPOLYDATAREADER Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkXMLPolyDataReader.html.
4. "VTKPOLYDATA Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkPolyData.html.
5. "VtkPolyDataMapper Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkPolyDataMapper.html.
6. "VTKACTOR Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkActor.html.
7. "VTKDOUBLEARRAY Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkDoubleArray.html.
8. "VTKRENDERER Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkRenderer.html.
9. "VtkRenderWindow Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkRenderWindow.html.
10. "VTKRENDERWINDOWINTERACTOR Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkRenderWindowInteractor.html.
11. "VTKMINIMALSTANDARDRANDOMSEQUENCE Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkMinimalStandardRandomSequence.html.
12. "VTKLOOKUPTABLE Class Reference." *VTK*, https://vtk.org/doc/nightly/html/classvtkLookupTable.html.
13. "Colorcells." *Kitware, Inc.*, https://kitware.github.io/vtk-examples/site/Cxx/PolyData/ColorCells/.
14. "Timer." *Kitware, Inc.*, https://kitware.github.io/vtk-examples/site/Cxx/Utilities/Timer/.
15. Narula, Harsh Kumar, and Harsh Kumar NarulaHarsh Kumar Narula. "Stopping VTK Timer Callback." *Stack Overflow*, 1 July 1965, https://stackoverflow.com/questions/50680621/stopping-vtk-timer-callback.