Scientific Programming

Assignment: Brain Mesh, use a class.

Anand Kamble

amk23j@fsu.edu

10th October 2023

---

1. Introduction

This report provides a comprehensive analysis of the C++ code designed for performing analysis on a brain mesh. Mesh is loaded from a VTK file. The code consists of several components.

1.1 Code Modules

The code is organized into the following modules:
• `main.cpp`: The main driver code responsible for loading the brain mesh, computing surface area, and saving results to files.
• `brain mesh.h/.cpp`: Class definition and methods for the `BrainMesh` object, which encapsulates mesh data and provides analysis methods.
• `triangle area.h/.cpp`: Functions for calculating triangle area, particularly the `areaOfTriangle()` function.

1.2 Program Flow

The overall program flow is as follows:
1. Instantiate a `BrainMesh` object in `main.cpp`.
2. Load the VTK file using the `BrainMesh::Load()` method.
3. Compute the total surface area using `BrainMesh::TotalSurfaceArea()`.
4. Calculate and save edge lengths using `BrainMesh::calculateEdgeLengths()`.
5. Calculate and save surrounding area per vertex using `BrainMesh::SurroundingAreaForVertex()`.

1.3 Formatting Conventions

Casing: Camel-case

Indentation: 4 spaces

Line Break: CRLF

2. BrainMesh Class

The `BrainMesh` class serves as the core component, encapsulating the mesh data and providing essential methods for loading, analysing, and writing results. Most of the methods in this class are written as lambda function [1] wrapped inside the error handler, which allows us to catch any errors and gracefully exit the program if anything goes wrong during runtime.

2.1 Properties

The key properties of the `BrainMesh` class are:
• `id`: A string identifier.
• `debugMode`: A flag to enable or disable debug messages.
• `numberOfPoints`: The number of vertices in the mesh.
• `numberOfPolygons`: The number of polygons in the mesh.
• `points`: A vector containing vertex coordinates.
• `polygons`: A vector containing polygon indices.

2.2 Public Methods

The main methods provided by the `BrainMesh` class include:
• `Load()`: This function reads vertex and polygon data from a VTK file specified by the given filename. It expects the VTK file to have a specific structure with POINTS and POLYGONS sections. The loaded data is stored in the \`points\` and \`polygons\` vectors respectively.

• `TotalSurfaceArea()`: This function iterates through the polygons of the mesh, calculates the area of each triangle, and accumulates the total surface area. The area of each triangle is computed using the Heron's formula based on the vertices' coordinates by calling the function `areaOfTriangle`.

• `calculateEdgeLengths()`: This method calculates the edge lengths of each polygon in the mesh and saves the lengths to the specified file. It iterates through the polygons, computes the lengths of edges by invoking the edgeLength function, and writes the results to the specified output file. The progress of the calculation is displayed in the console.

• `SurroundingAreaForVertex()`: This method calculates the total surface area of all polygons surrounding each vertex in the mesh and writes the results to the specified output file. It iterates through each vertex, computes the total surface area of polygons connected to the vertex, and saves the area to the output file. The progress of the calculation is displayed in the console.

2.3 Private Methods

The private methods in the `BrainMesh` class include:

• `debug()`: Utility function which prints the given message to the standard output if the debug mode is enabled.

• `clone()`: This function copies the debug mode setting, number of points, and number of polygons from another BrainMesh object, allowing the current object to mirror the properties of the source object.

• `destroy()`: This function performs a cleanup operation, resetting the BrainMesh object by clearing its points and polygons, resetting the number of points and polygons to zero, turning off debug mode, and clearing the ID string. After calling this function, the BrainMesh object is restored to its initial state, ready for reuse or destruction.

• `split()`: Splits the provided string into parts separated by the given separator.

• `errorHandler()`: This function takes a callback function [2] as a parameter and executes it. If the callback function encounters an exception of type std::exception, it catches the exception, prints an error message including the ID of the BrainMesh object, and calls the `destroy` method to clean up the object's resources before terminating the program. The error message provides context about the object involved in the error and the specific exception message received. This function ensures graceful error handling for various operations within the BrainMesh class.

• `edgeLength()`: This function computes the length of the edge (distance) between two vertices represented by the given 3D coordinates.

4. MakeFile

This makefile is a concise build script for a C++ project. In this MakeFile:

- Compiler and flags are defined.
- Directories for source, object, library, and binary files are specified.
- Source files, object files, and static libraries are named and organized.
- Targets include 'all' for the executable and 'clean' to remove generated files.
- Dependencies between source files, object files, and libraries are established.
- The build steps include compilation, library creation, and executable linking.

The makefile is designed for easy maintenance and modular code, with potential improvements in dependency tracking and additional comments. [3] I have also added a target named `test` which builds the test version of the program.

5.Execution

The `Makefile` is included with this code. You can run the command `make` to compile the program. After successful compilation, you can find the executable named `main.out` inside the `bin` folder. Run this executable by `./bin/main.out`. The outputs generated by the program can be found in files `Surrounding areas.txt` and `edge lengths.txt`. To clean the generated folders and files, use the command `make clean`

6. Testing

For testing purposes, we will compile the file named `test.cpp` instead of `main.cpp`. The test program is designed to import the `BrainMesh` class and load the `test.vtk` file using it. Upon successful execution, it proceeds to compare the obtained results with the known results. The program then prints the outcomes of each function within the class, indicating whether the test for each function was successful or not.

To execute the tests, build the test version of the program by running following command in terminal,

```
make test
```

For testing purposes, I've updated the Makefile to build a test version of the program. This update involves introducing a new target named "test." [4]

References

[1] T. c.-h. v.-k. M. o. m. g. and S. , “Lambda expressions in C++,” Microsoft, 19 February 2023. [Online]. Available: https://learn.microsoft.com/en-us/cpp/cpp/lambda-expressions-in-cpp?view=msvc-170. [Accessed 8 October 2023].

[2] Pixelchemist, “Callback functions in C++,” Stack Overflow, 24 February 2015. [Online]. Available: https://stackoverflow.com/a/28689902/22647897. [Accessed 8 October 2023].

[3] OpenAI, “ChatGPT,” OpenAI, Microsoft Corporation, [Online]. Available: https://chat.openai.com/share/4de328ab-60f6-4cff-ae2c-f19896634da0. [Accessed 8 October 2023].

[4] D. Lin, “How can I configure my makefile for debug and release builds?,” Stack Overflow, 3 July 2009. [Online]. Available: https://stackoverflow.com/a/1080180/22647897. [Accessed 8 October 2023].

[5] Bearboat Software, “Area of Triangle in 3D,” 10 October 2023. [Online]. Available: https://bearboat.net/TriangleArea/Triangle.html. [Accessed 8 October 2023].