SOFTWARE ARCHITECTURE

for

Mesh Smoothing Tool

Release 1.0

Version 1.0 approved

Prepared by Jan Jose Hurtado Jauregui
Pontifícia Universidade Católica do Rio de Janeiro

Contents

Revision History

Date	Version	Description	Author
18/04/2017	1.0	Software Architecture 1.0	Jan Hurtado

SOFTWARE ARCHITECTURE: Mesh Smoothing Tool

1 Introduction

1.1 Purpose

The purpose of this document is to provide a comprehensive architectural overview of the Mesh Smoothing Tool software. We present detailed information of how the software will be implemented.

1.2 Scope

3D shapes and surfaces obtained from real world data usually present undesired noise. This kind of problems are treated using smoothing techniques whose key aspects are denoising and fairing. Denoising is related to high-frequency noise removal, and fairing is related to obtain the smoothest version of the input regarding an energy function. This smoothing step is very important in a typical geometry processing pipeline.

Surfaces are commonly represented as triangular meshes due to its simplicity and easy processing. The smoothing task over meshes is called mesh smoothing, and is related to the modification of the geometric properties of the mesh (e.g. vertex positions). There are some important considerations in mesh smoothing algorithms such as detail-preserving, low normal variation, volume preservation, etc. Depending on the application these features determine robustness.

For example, in the case of medical data, several techniques are used to obtain a volume which represents the anatomy of the patient (e.g. X-ray radiography, medical ultrasound, Magnetic Resonance Imaging (MRI), etc.). Using a subset of this volume, a surface model can be reconstructed to represent the target region. Each step of the reconstruction process can introduce noise to the final surface.

To remove this kind of noise, several mesh processing frameworks introduce a mesh smoothing tool for this purpose.

1.3 Overview

This document is structured as follows. In Section 2 we introduce the architecture and its corresponding criteria. In Section 3 we present all architecture

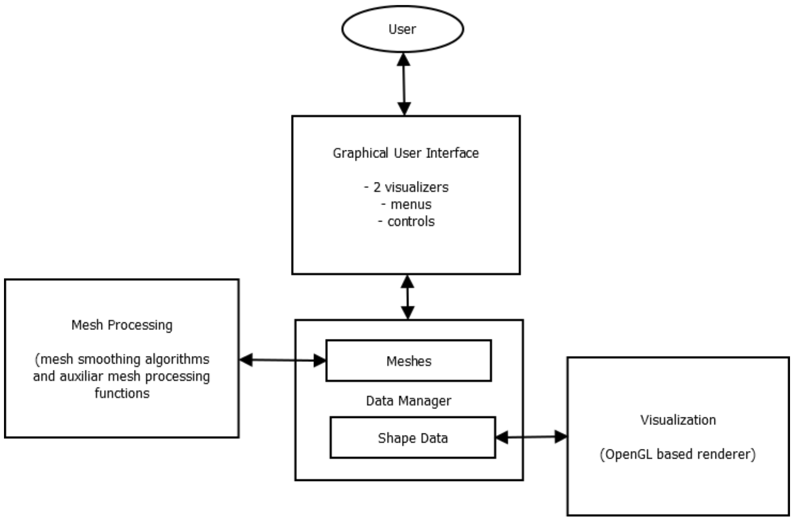

Figure 1: Overview of architecture

diagrams.

2 Architecture

As we considered in the requirement specification document, we will implement the software entirely in C++ language. Summarizing all requirements we can consider that the software should be able to: load input mesh, save output mesh, perform global smoothing, perform focalized smoothing, configure parameters of smoothing tools, visualize the meshes and configure the corresponding visualization We define the software as a composition of four main modules or components:

- Graphical User Interface: allow the user to interact with the software. It includes widget objects such as menus, buttons, sliders, 3D visualizers, etc. It will be based on the QT library.
- Data manager: this component manages input and resulting data for smoothing and visualization.
- Mesh processing: contains a set of tools for mesh processing, including denoising (used for smoothing). It will be based on OpenMesh library.
- Visualization: used for rendering of 3D data, in specific, of triangular meshes. It is based on OpenGL.

We can say that visualization and mesh processing components are independent regarding the others. Each one works with different representation of triangular meshes. In the case of visualization it is not necessary to perform a mesh navigation, for this reason the data of the vertices and the connectivity are represented using simple arrays. In fact, the OpenGL API works considering the latter. Also we use penGL Extension Wrangler Library (GLEW) to manage OpenGL and OpenGL Mathematics to perform mathematic operations. In the case of mesh processing it is necessary the usage of an efficient data structure to navigate in the mesh. For example, in mesh denoising it is important to obtain neighborhoods of each vertex. We adopt a half-edge based data structure implemented in OpenMesh library for navigation and geometry modification. Also we will use vector operations implemented in OpenMesh.

The component data manager will work as a container of the two different representations of triangular meshes of the current input and output meshes. Also it will update the data when a change is done or when new data is loaded. So it depends on I/O and update operations of mesh processing and visualization components. All mesh processing algorithms and visualization features will be performed over the data contained in this module. The main interface will only show this data and manage the corresponding interactions. In Figure 1 we show a general overview of the architecture.

Being more specific, these are all third party libraries we will use in the implementation:

- QT library 5.8 (Core, GUI, Widgets and OpenGL modules)
- OpenMesh 6.3 (Core)
- OpenGL 4.5.0
- GLEW 2.0.0
- GLM 0.9.8

All software tasks will be executed following a sequential scheme with the exception of the global smoothing task. We want to perform this task using another thread without blocking the visualization and interaction of partial results. Due to the iterative nature of the global smoothing, we will implement it with two iteration level. An internal iteration which is going to be executed in parallel and an external iteration which is going to work as an updater for the partial results.

For the architecture diagrams we will adopt a UML based definition. We will present a use case diagram, a component diagram and a class diagram, which we consider are the most important for the implementation. Also we define an automata for global smoothing implementation considering the threaded execution.

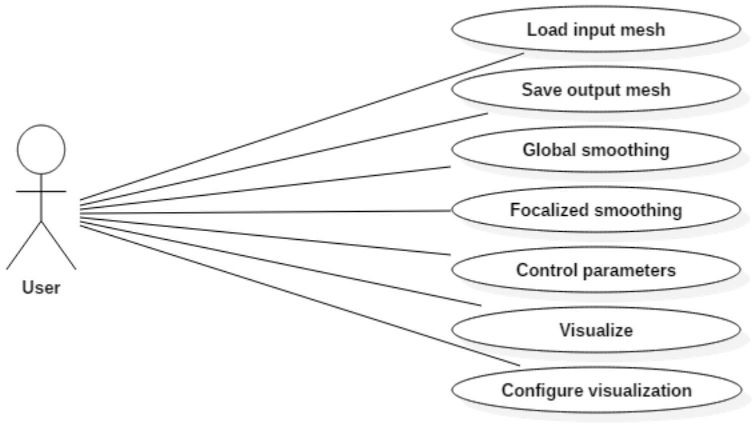

Figure 2: Use case diagram

3 Diagrams

3.1 Use Case Diagram

As we said before, we can summarize all requirements to have a more compact representation. Based on this idea we can define the use case diagram presented in Figure 2.

3.2 Component Diagram

In the component diagram we define each component and the corresponding dependencies. It is important to remark that data manager will only depend on I/O and update operations of visualization and mesh processing components. The diagram is presented in Figure 3.

3.3 Class Diagram

Based on an object-oriented solution we can define a class diagram considering these main classes:

- TriMesh – Triangular mesh representation using a half-edge based data structure. It will support all mesh processing operations.
- ShapeData – Triangular mesh representation for OpenGL based visualization.
- DataManager – Data container for algorithm execution and GUI manipulation.
- Renderer – OpenGL based renderer.

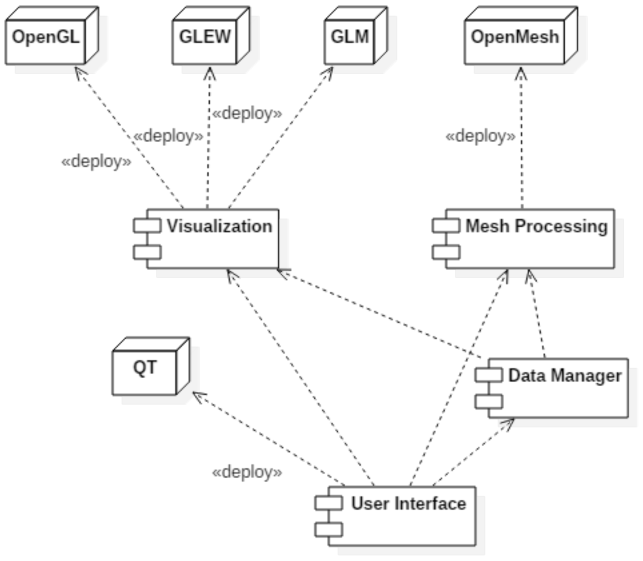

Figure 3: Component diagram

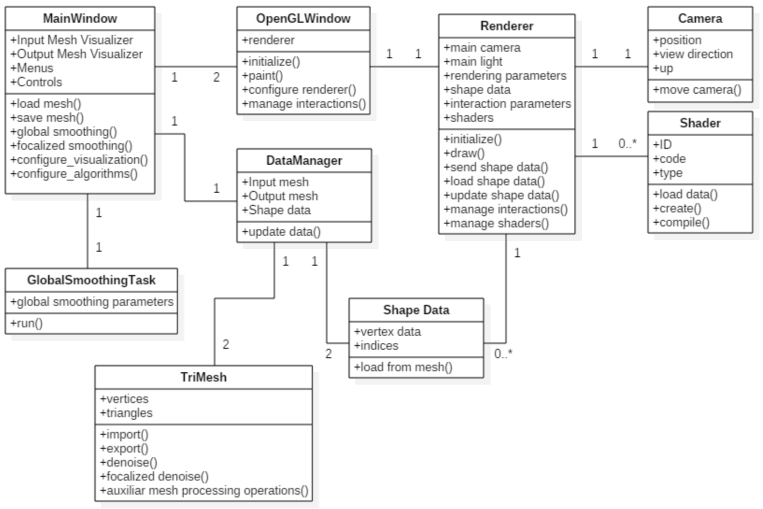

Figure 4: Class diagram

- Camera – A camera for the renderer.
- Shader – Shader data for the renderer.
- OpenGLWindow – QT based window for OpenGL visualization.
- GlobalSmoothingTask – Global smoothing manager (threaded).
- MainWindow – Main GUI.

The class diagram is presente in Figure 4

3.4 Global smoothing automata

To manage the synchronization of the global smoothing task we will consider the following states:

- INIT – The initial state.
- STARTED – When a global smoothing task was started.
- STOPPING – When the STOP button is pressed and the program is waiting for the global smoothing thread to finish (internal iterations).
- STOPPED – When global smoothing thread finish and the user can rerun a smoothing task again.
- CONTINUING – When a global smoothing task is restarted.

The corresponding transitions are the following:

- start – it happens when the user starts a global smoothing task (press START button)
- stop – it happens when the user stops a global smoothing task (press STOP button)
- wait – it happens when the user is waiting for synchronization (user can't press any button)
- continue – it happens when the user continues a global smoothing task (press CONTINUE button)
- restart – it happens when the user restarts a global smoothing task (press START button when task was stopped)

In Figure 5 we show the automata.

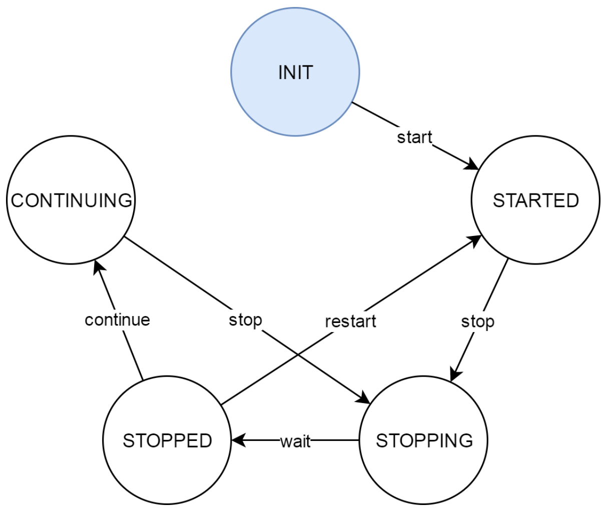

Figure 5: Global smoothing automata