UnBox3D:
Software Design Document

Version
2.0.0

Group 1: Alexander Ramirez, Vivian Cases, Jacky Lim,
Nicholas Sisneros, Vince Wang

Contents

Revision History

Name	Date	Reasons for Changes	Version
Alexander Ramirez	2025-06-01	First Draft	1.0.0
Vivian Casas	December 11, 2025	Update for snapshot 2	2.0.0

1 Introduction

1.1 Purpose

This document shows the general structure and specific implementation details to provide clarity on the project's direction and targeted features. Version 2 reflects the second checkpoint, where the import funciton is integrated and models in '.obj' format can be parsed and rendered in the viewport. It will ensure developers and stakeholders alike agree upon the direction of the program's development.

1.2 Intended Audience

Our intended audience includes:

- Software Developers
- All stakeholders
- Project Managers
- Quality Assurance Teams/Testers

1.3 System Overview

The program now allows importing '.obj' models into a scene, parsing them with Assimp, and rendering them using OpenGL through the OpenTK wrapper. This marks the transition from a placeholder interface to a functional prototype. Although exports and simplification features remained planned for later, the user will be satisfied with their changes that they can export, make cuts, and folds to the models in a scalable vector graphics file (.svg).

2 System Architecture

This section provides a high-level overview of UnBox3D's architectural design.

2.1 System Purpose and Goals

The program seeks to:

- Allow simplifying models
- Automate the unfolding process
- Provide visual feedback during simplification

2.2 General Workflow:

1. **Import Model:** Users can import a model into the scene that is in .obj format.
2. **Model Rendering:** The model is rendered in the viewport for the user to begin the simplification process.
3. **Simplification:** The user has the option to simplify the model by removing objects that are smaller than a certain threshold, or by Blender's decimation modifier where a given object has its geometry simplified while keeping the topology similar to before based on faces' angle constraints.
4. **Export:** The model is passed to Blender where it is unfolded using a Blender extension that automatically unfolds geometry islands and marks the cuts and folds for reassembly.

2.3 Data Flow:

1. When the user imports a model, an Assimp mesh is created
2. The Assimp mesh passes its data to a g3 mesh (DMesh3) so we can modify the data during simplification
3. Every frame, the data is passed from the g3 mesh to an AppMesh (a custom data container created by us)
4. The AppMesh data is passed into buffers for the GPU to access, with the help of OpenTK

3 User Interface

This sections describes the UI and how it is implemented.

3.1 Overview

There are 2 main screens in this program: the splash screen and the viewport.

- **Splash Screen:** The logo is shown as the program loads
- **Viewport:** The viewport features an Import button, a region where the scene is rendered, and an Export button for unfolding.

3.2 Viewport Refreshing

The program is able to refresh the viewport effectively thanks to OpenTK, a wrapper for OpenGL. It takes the data from AppMesh instances and constantly updates the buffers for the GPU to render objects accurately at every frame.

4 Glossary

- UI - User Interface
- Assimp - Open Asset Import Library
- OpenTK - Open Toolkit Library (a C# wrapper for OpenGL)

5 References

- Software Design Document (SRS): link will be added when available
- Original GitHub: Link