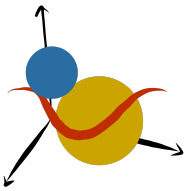
Version 1.1

Software Design Document

February 15, 2021

Team Triaxis



Sponsors:
Dr. Will Grundy, Dr. Audrey Thirouin

Faculty Mentor:
Sambashiva Kethireddy

Team Members:
Eleanor Carlos
Reyna Orendain
Andres Sepulveda
Brandon Visoky

Overview

This document outlines and details the architectural design of the Team Triaxis' project.

Table of Contents

1. Introduction

Exploring our galaxy is a journey that often feels like one that is beyond any of our lifetimes. Something that can only be achieved through science fiction, television shows, and novels. While this may be true at present, there is amazing work being done every day by scientists and researchers to observe the galaxy and turn what often feels like fiction, into reality. One of our only ways to study and research celestial bodies is to visualize them based on observed data. This is especially true for objects that are far too small and distant from Earth to even consider sending a man-made craft to study them. Of particular interest to our project, researching asteroids relies nearly entirely on different forms of visualization because of their distance from our planet.

We are Team Triaxis and we are working with our clients, Dr. Will Grundy and Dr. Audrey Thirouin, on the project “Complex Asteroid Shapes in Modeling of Binary Asteroid Systems.” We are responsible for delivering the goals put forth by our clients and Lowell Observatory. Of these goals, Dr. Will Grundy and Dr. Audrey Thirouin are focused on modeling binary systems in the Kuiper Belt, and the Licht-cpp software that we are continuing work on helps them visualize and model the data from light curves in ways that would otherwise be impossible.

This is the third year and third iteration of this project given to us by Lowell Observatory. Team Triaxis looks forward to continuing the development of this project as left to us by our former peers, Paired Planet Technologies and Team Andromeda.

Our iteration of this project includes implementing a new module to render binary systems observed by Lowell Observatory. Our primary objective is to render the components of these systems as triaxial ellipsoids, a more complex model than the current spheres they are rendering them as now. While our primary objective is to implement the ellipsoid module, we will also be expanding upon previous work to create a module for fluid equilibrium, as well as a graphical user interface (GUI) to vastly improve the functionality of licht-cpp as it was given to us.

Part of delivering the highest quality product we are capable of delivering for our clients is to develop and expand on the already robust testing suite that has already been developed for Licht-CPP. The framework for unit testes has already been developed by previous semesters, and thus our largest goal is to incorporate the new features and fixes we have developed into the existing suite of tests that already exist.

2. Implementation Overview

The end goal of this phase of this project is to deliver a fully functioning and stable release of licht-cpp to our clients at Lowell Observatory. We will be delivering several bug fixes as well as some quality of life improvements for our clients, which includes: fixing the ellipsoid module, implementing a module that renders objects in fluid equilibrium, and if all goes according to plan, we will develop a graphical user interface for our clients to interface with the NLM as was developed in the previous semester.

Triaxial Ellipsoid

The triaxial ellipsoid class first developed during the first phase of this project with Team Paired Planet, and then later completed by Team Andromeda during the second phase. The second iteration was, unfortunately, nonfunctional by the end of their semester, and so it is our main goal to have it fixed by the end of our time with this code. The Ellipsoid module will provide much-needed functionality to our clients, by allowing them to render more complex shapes than a sphere, while also rendering at a much faster speed than the faceted shapes module would also provide.

Fluid Equilibrium

The fluid equilibrium submodule will depend on the triaxial ellipsoid submodule since an object in fluid equilibrium is a triaxial ellipsoid whose shape changes over time.

- a submodule that will be based on the structure of existing shape modules
- will add functions that will allow shape/radii to change over time based on parameters given by the user

Graphical User Interface (GUI)

The GUI will rely on wxWidgets 3.1.4+ to render the interface for the user. Both frontend and backend development, as well as the interface itself, will be dependent on C++ (VERSION), to simplify the number of dependencies for current and future modules of the licht cpp program.

wxWidgets renders a user interface in a platform-agnostic manner, therefore the GUI would be able to be rendered regardless of the operating system. The licht cpp program does require the use of Ubuntu, so it is highly recommended to use a virtual machine if not an actual installation.

3. Architectural Overview

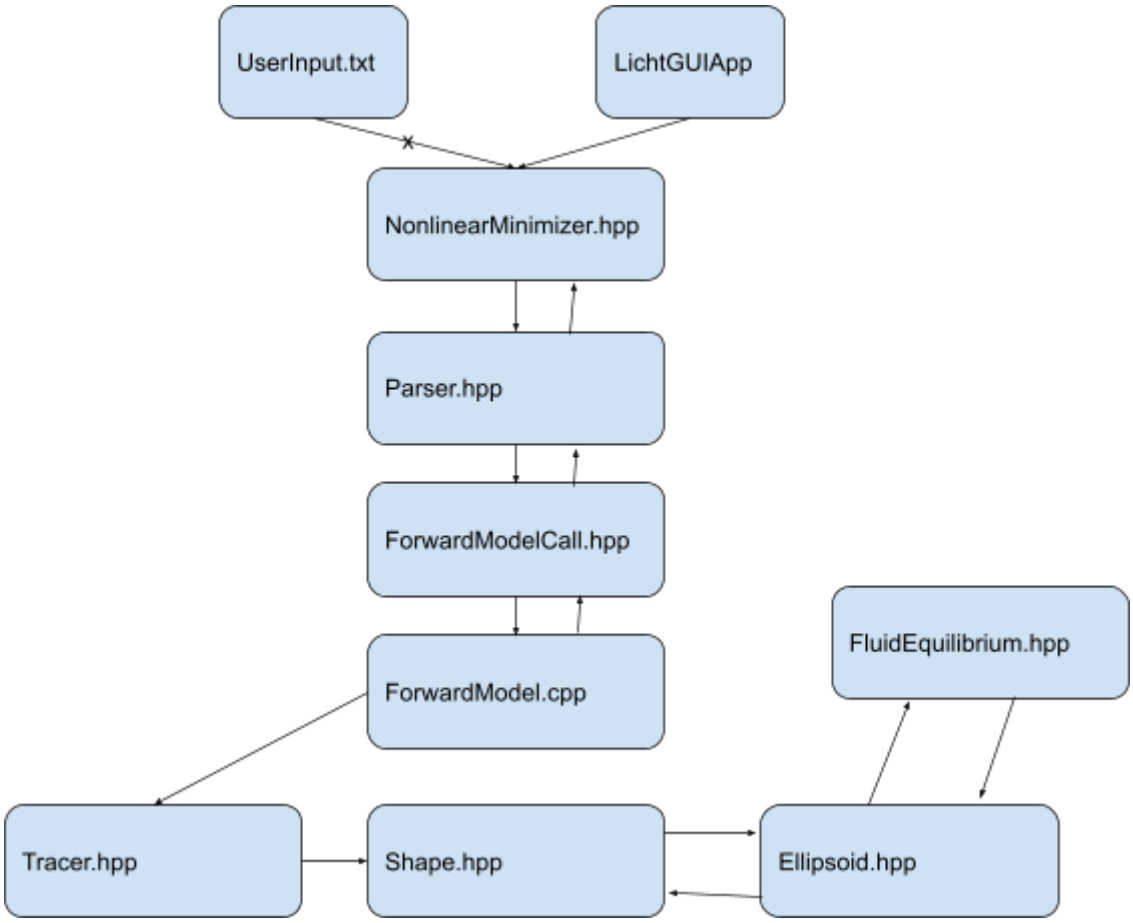


Figure 1: Dependencies of the conceptual modules

To keep a modular structure for this project, licht-cpp is broken into several different modules that all work together to bring the project into a singular functional state. For the goals in this phase of the project, the modules we will be working with are the following:

Triaxial Ellipsoid

The Triaxial Ellipsoid module focuses entirely on the rendering of the ellipsoid shape for use within the forward model. This module finished development during the last iteration of the project but was left with some minor bugs and flaws within the system which we are to fix. Currently, the module renders lighting incorrectly when the shape is expanded to more extreme values, and also spins at the wrong axis.

Fluid Equilibrium

This module will be responsible for determining how the radii of objects in equilibrium will change over time. Since this module depends on triaxial ellipsoids, all it will do is change the current radii as if it was a triaxial ellipsoid and leave the original radii variable unchanged. Then it will be added as an option in the forward model, which will be called by the NLM so that users can generate objects in fluid equilibrium by adjusting its parameters.

GUI

The GUI module will extend the functionality of the previously implemented forward model module. This new model will allow the user to enter parameters into the forward model from the GUI rather than entering them at the command line, or via .txt file. Additionally, users will be able to see the predicted light curve directly on the screen and have the option to compare the predicted light curve to the observed light curve.

4. Module and Interface Descriptions

To retain modularity as previous teams have done, we shall keep the overall design in separate modules and submodules. The new modules, fluid equilibrium, and the graphic user interface, remain semi-independent and interact with each other when necessary. As for the triaxial ellipsoid, it will remain the same, keeping its dependencies unless the solution to the lighting and rotation issues affect the way it works.

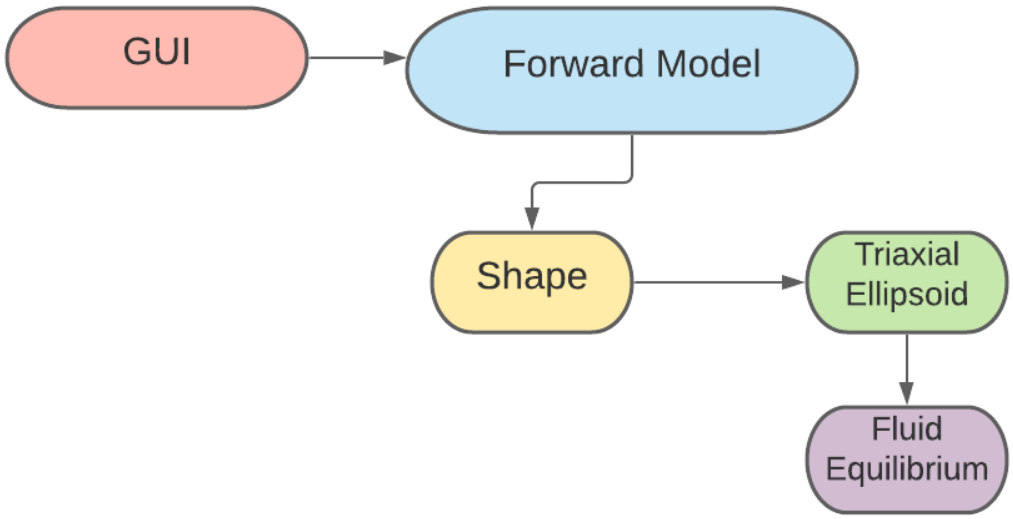


Figure 2. Conceptual Modules

4.1 Triaxial Ellipsoids

The Triaxial Ellipsoid submodule is the third and final extension of the Shape module. And it will be the module for the Fluid Equilibrium submodule. The Triaxial Ellipsoid inherits and defines all functions from the Shape interface. It also uses the Math module for precise calculations.

Dependencies

- Math.hpp
- Shape.hpp
- HapkeModel.hpp

Use cases

- After the forward model has been called, the Shape module uses the calculations from the Ellipsoid submodule to generate the shape. The shape is then manipulated by the ray tracer.

Design

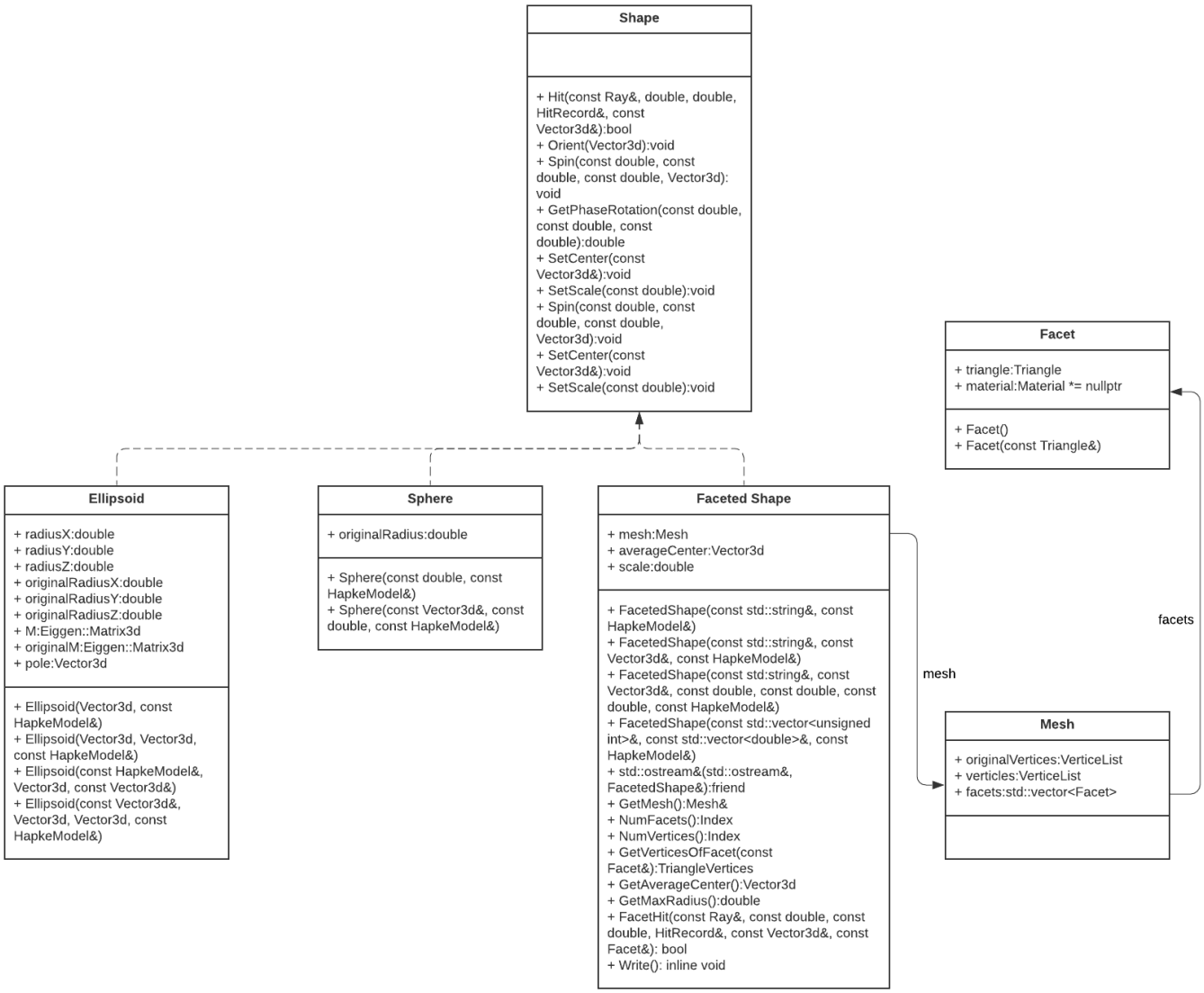


Figure 3. UML diagram describing the relationship of all shape classes

4.2 Fluid Equilibrium

The Fluid Equilibrium module is a submodule to the Triaxial Ellipsoid module. So, Ellipsoid is the only dependency for this module. It extends the Ellipsoid module by accepting a spin frequency and density which can both be used in the function for updating the Ellipsoid's radii.

Dependencies

- Ellipsoid.hpp

Use cases

- When a user uses the forward model, the Shape module calls ellipsoid, which calls fluid equilibrium to draw the shape

Design

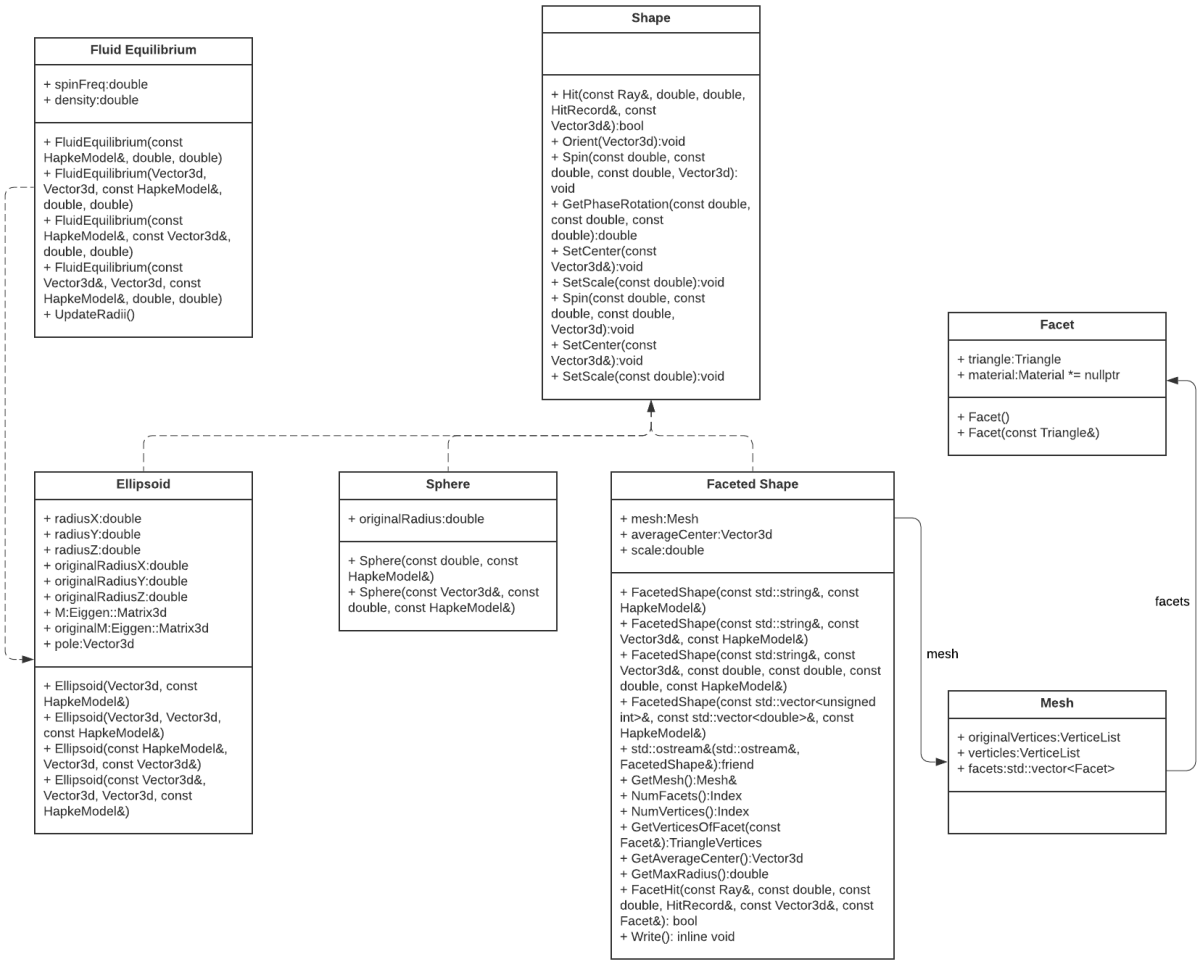


Figure 4. UML diagram describing the relation of Fluid Equilibrium within the Shape classes

Constructors

- Given Hapke, spin frequency, and density
- Given pole, Hapke, spin frequency, and density
- Given center, Hapke, spin frequency, and density
- Given center, pole, Hapke, spin frequency, and density

Method

- UpdateRadii: using the equation for objects in fluid equilibrium, update the radii for the triaxial ellipsoid

Variables

- spinFreq
    - The object's spin frequency
- density
    - The object's density

4.3 Graphical User Interface

The Graphical User Interface (GUI) provides the user the ability to run the forward model with parameter input from a dedicated interface. Once acceptable parameters are input, the forward model will generate an estimated light curve. Users can utilize this data to compare observed data to the estimated light curve and form characteristics about binary systems.

Dependencies

- C++
- Visual Studio 2019
- WxWidgets 3.1.4
- External C++ Shared Library

Use cases

- The user will be able to plot a graph of the lightcurve data
- The user will be able to output data to CLI for nlm / forward model rendering
- The user will be able to import / export data via .JSON file
    - Alternative: The program will be able to read the .txt files formatted by previous capstone groups.

Design (prototype)

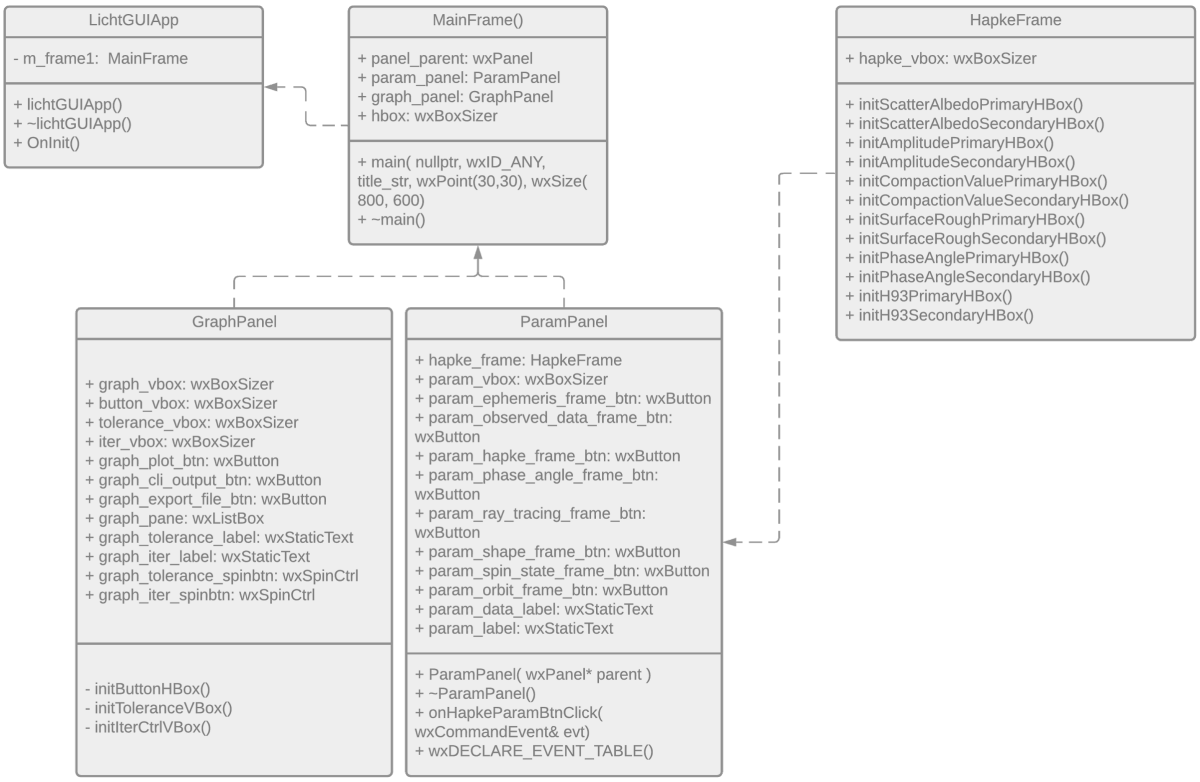


Figure 5. UML diagram of the Licht GUI app

Integration with Current Project

The GUI will use the functionality of the already built forward model when possible. Currently, it will use four main functions from the External C++ Shared Library:

- ForwardModelCall
    - Calls the forward model from the External C++ Shared Library with the parameters from the Interface object
- ParameterCollection
    - Takes in parameter input from the GUI and creates a dictionary with the variable name, and the data associated.
- FileInput
    - This function allows the user to input a data set from an external file to help compare the observed data to the predicted data.
- PlotData
    - This function plots the data from either the FileInput function or the ParameterCollection function.

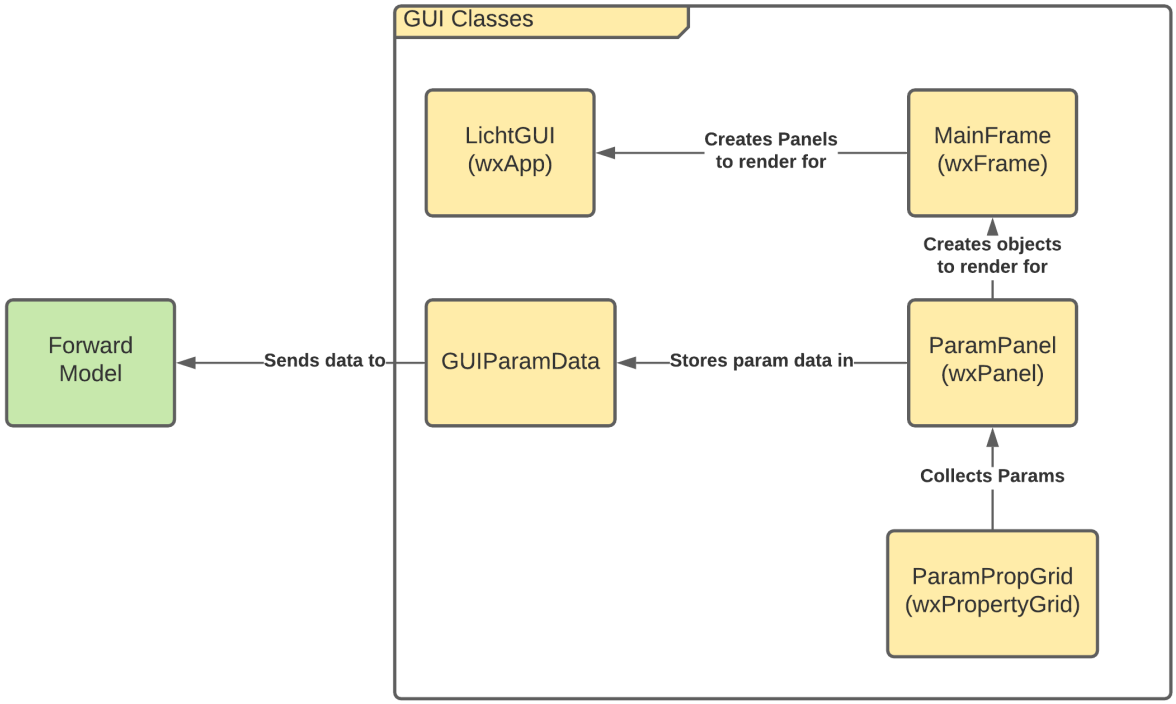


Figure 6. Visualization of GUI Structure and integration

Parameters

The parameters that will be used are the same parameters used for the forward model. They will be taken by the ParameterCollection function and sent to the ForwardModel_IDL function within the C++ Shared Library.

5. Implementation Plan

The implementation of this project starts with our immediate understanding of the material handed to us from previous semesters and combing through what needs to be fixed, expanded upon, or outright replaced. This is the first undertaking we will take in the overarching goal of this project, and can either be very simple or very complex depending on the scope of the bugs we need to fix.

Due to the overarching goal of this project revolving around the fixing of bugs, this will be the main timeframe that we must workaround and it is very hard to predict. Such bugs could be fixed in a short time or could take months of this working semester to search through. Due to this, any extensions of this project must be delayed until a grasp of the issues at hand are either figured out, or a fix has been deployed.

As such, the first few weeks of development will largely be spent with trial and error, looking through previous teams commit history, and discussing with our clients where the issues in the code may be. Onwards from there, we can begin to split the work among us to tackle different aspects of the project, such as building the fluid equilibrium into licht-cpp and working on a graphical user interface. Looking at the Gantt chart below, our major deadlines are based around the Alpha Prototype that is needed by March 1st.

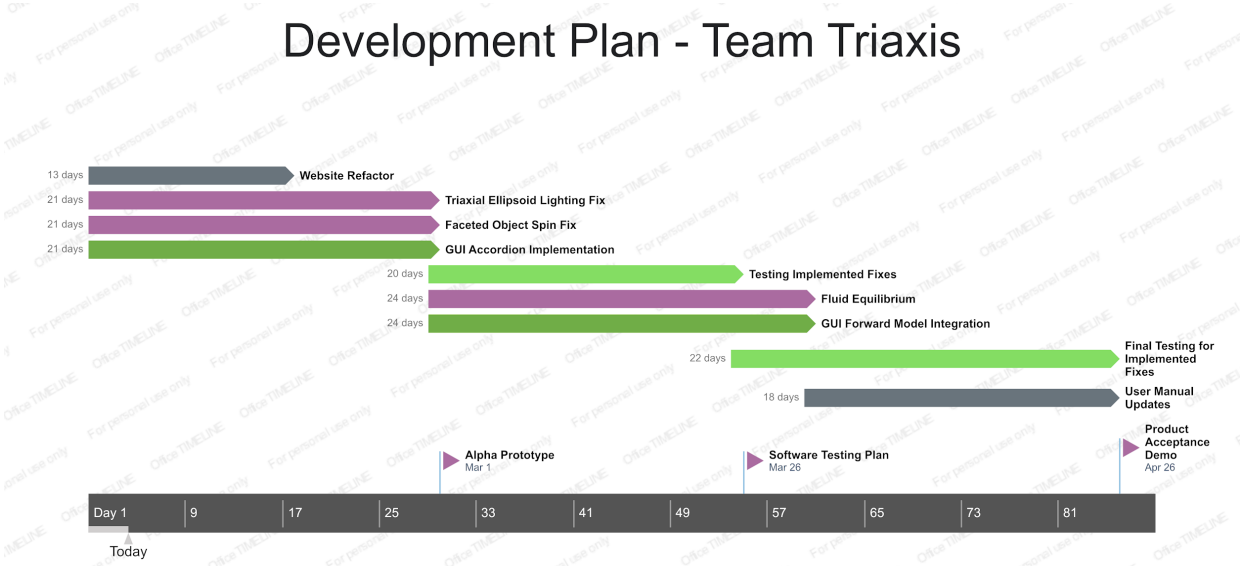
The first segment of time will be broken into four tasks, refactoring the website to better suit our mentor's requirements, fix the issue with the lighting of the ellipsoid module, fix the issue with the rotation of the faceted object, and then begin intermediate steps into developing the GUI further.

From there, the objective turns to testing our fixes, as well as implementing the fluid equilibrium module. If all has gone to plan, we will also begin the phase of building our GUI solution into the software as we have built it.

The final phase of the project is to implement more in-depth tests, as well as improve the user manual and Doxygen documentation to match what we have completed during this semester. At that point, our implementation of the project will be completed and we can approach our user acceptance tests with our clients.

Capstone / Development Plans

Below are two Gantt charts summarizing the development plan, as well as a big picture understanding of the tasks that need to be accomplished to successfully complete this capstone project. It is important we consider both plans as the tasks / goals outlined are to be concurrently accomplished throughout the semester.



Last Updated: 4 Feb 2021

6. Conclusion

Our knowledge and understanding of space is continuing to grow with the technology we develop to observe it. Thousands of years ago, humanity did not have a fraction of the understanding we have of outer space now. Imagine what we will know in another thousand, hundred, or even several years from now.

This growing knowledge pivots and relies on the thousands of astronomers, including our clients Dr. Will Grundy and Dr. Audrey Thirouin's, continued research. Of course, this research also depends on engineers and programmers who can design and build the specialized tools that are needed day to day.

Asteroids can be a very important and interesting field of study for astronomers for a multitude of reasons. Since they are much smaller than planets and far too numerous to land spacecraft on each one, researchers must rely on technology like licht-cpp to continue their research.

This is why we, Team Triaxis, are excited to pick up the torch of previous years' implementations of licht-cpp, a software designed specifically for our clients at Lowell Observatory to model the light curves of binary asteroid systems. Our primary goal is to implement a module to model binary systems using triaxial ellipsoids. Once this main requirement is met, we will be able to move forward with further improvements, such as implementing a GUI and GPU parallelization.

So far, we have been able to address minor bugs that our clients observed this semester pertaining to the spin states of faceted objects and how the users are prompted for the parameters for spin states. The names of the parameters asked for in the user input text file have been fixed and doubles are accepted instead of ints. These changes have been made known to our clients and are pending testing by our clients. Recently, we have also made progress with debugging the lighting of ellipsoids. In our investigations, we changed a calculation for the ray direction in the tracer function following a suggestion by our clients. The lighting seems to have improved, but the shadows on the shape now fall where they should not. We are currently continuing to work with our clients on this issue. We have also begun work on implementing the GUI. This progress is currently in another branch on our GitHub before being integrated with the master version of our project.

We hope that by the end of this project, we can deliver a functioning product with many quality of life changes and other improvements for our clients at Lowell Observatory. With each iteration of this project, our clients receive a better and more capable software to conduct their research with, and Team Triaxis is excited to further this tradition.