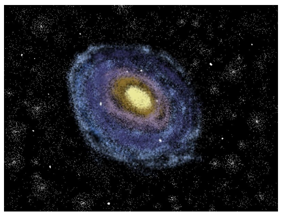

Team Andromeda

Version 1

Software Design Document

February 5, 2020

Clients - Dr. Audrey Thirouin and Dr. Will Grundy
Mentor - Isaac Shaffer
Members - Matthew Amato-Yarbrough, Batai Finley, Bradley Kukuk, John Jacobelli, and Jessica Smith

Table of Contents

1. Introduction

Since the time of Galileo, humans have been studying the universe. Currently, billions of dollars are spent every year on sending probes to other planets and small bodies in an attempt to understand what lies in the cosmos. This continuous research has driven technological development in areas not directly related to space. For example, home insulation, baby formula, and portable computers are a few of the byproducts of space exploration. Other valuable data and theories have been derived from space exploration as well, such as information about early planet formation. Many observatories like Lowell are gathering information daily to further our comprehension of areas in space such as the Kuiper Belt, which is a region of the solar system beyond the orbit of Neptune containing small bodies such as Pluto. Lowell and other observatories study how these small bodies form and interact with each other.

Our clients Dr. Audrey Thirouin and Dr. Will Grundy work for Lowell Observatory. Dr. Thirouin is a research scientist interested in the characteristics of small bodies within the Solar System, while Dr. Grundy is an astronomer that researches Kuiper Belt objects. Kuiper Belt objects are a region of leftover bodies from the solar system's early history, making them valuable for observing conditions similar to that of early planet formation. Together, Dr. Thirouin and Dr. Grundy focus on collecting and analyzing data about small bodies that reside far from Earth and are hard to directly observe. Currently, they are working on modeling binary systems in the Kuiper Belt and need to use techniques that make the most use of the data available to achieve this. For distant objects, only a pinpoint light source is observable. This can be used to determine the brightness of the object over time. These luminosity recordings can be combined to form a light curve, which is a graph of brightness values over a given set of time.

Light curves can be used to infer the rotational and physical properties of small bodies. For instance, an asteroid that is non-spherical will reflect more light when a larger portion of its surface is facing an observer. This is because more light from the Sun is being reflected to the observer. Since the object is reflecting more light at certain points in its rotation, the brightness will differ depending on when it is observed in its rotation. The various brightness values can then be graphed, generating the light curve. Information such as the period and amplitude of the curve, rotational period, and rough properties can then be found based on graph data. A broad number of other characteristics can be estimated using light curves by an adept astronomer.

Our clients use light curves to better understand binary systems. Dr. Thirouin and Dr. Grundy can model these systems with spheres and faceted objects using software developed in a previous iteration of the project. Our clients need upgraded software to improve and expand the application. They wish to do this through the addition of triaxial ellipsoid shapes, a Markov chain Monte Carlo (MCMC) algorithm, a graphical user interface (GUI), and a video generator. The solution for this iteration of the project will allow for more accurate modeling of binary systems and help users generate precise characteristics of these systems.

To implement this solution, the triaxial ellipsoids will need to accurately use ray tracing to simulate the ellipsoid-shaped system, as well as rotate the ellipsoid around a given pole. In addition, the Hamiltonian Monte Carlo (HMC) algorithm will need to calculate a range of values that are likely for a given parameter. Furthermore, the new GUI will increase ease-of-use for the API. Finally, the movie generator will need to compile images into a movie format for users to manipulate as needed.

2. Implementation Overview

Our solution will address our clients' problems by using the current API, frameworks, and math libraries to enhance performance and functionality. A framework will be used to provide a GUI that facilitates the entry of parameters for the forward model. A separate framework will be added to the GUI, allowing data plots. Our solution will also include a Shape subclass for the implementation of triaxial ellipsoids. Additionally, we will use an HMC API to handle the implementation of the HMC algorithm. Lastly, we will be creating a solution in C++ that will generate a video of the images created by the existing API.

Triaxial Ellipsoid

The addition of an Ellipsoid class to the existing API will enhance our user's research. We will be tinkering with the Ellipsoid branch that was previously instantiated by Paired Planet Technologies. The methods that we are working with are built in conjunction with Math.hpp and the existing Hapke model. The methods that will help improve the existing API are Orient, SetCenter, setScale, and Hit.

Hamiltonian Monte Carlo
We will be building the Hamiltonian Monte Carlo submodule using the programming language R. Using R allows us to use a framework called RStan which will help ease the addition of the Hamiltonian Monte Carlo Algorithm. Once implemented, RStan will allow the user to specify ranges of values for certain parameters and plot the correlation of the parameters.

Forward Model Graphical User Interface
The Forward Model Graphical User Interface will rely on several different frameworks and APIs to create the best experience for our users. For the frontend development, we will be using Tkinter, a Python 3.8.0+ framework that allows for quick and easy development of the front end.

For the backend, the interface will be built with Python 3.8.0+ and will use the ctypes integrated module. Ctypes allows us to call functions and use data from an external C++ shared library where the forward model will be called and manipulated.

For the plotting functionality of the interface, we will be using a Python module named Plotly. This allows the user to take in a data set with certain constraints and will generate the lightcurve for the user. This will allow users to observe, compare, and analyze the data set.

Movie Generator
Our solution for the movie generator will be the implementation of a C++ script that will gather the images generated from the existing API and sow them into a .MP4 file. Users will be able to create these movies for analysis and visual representation. We will be using a library called OpenCV that will allow for easy video generation.

3. Architectural Overview

In this iteration of the project, we plan to add two modules and two submodules to the preexisting six modules implemented last year by Paired Planet Technologies. These modules and submodules will be designed to either add or extend functionalities from the previously implemented modules. By doing so, we uphold the preexisting modular design, while implementing additional functionalities within our iteration of the project. Below is the architectural design our team plans to follow for our iteration:

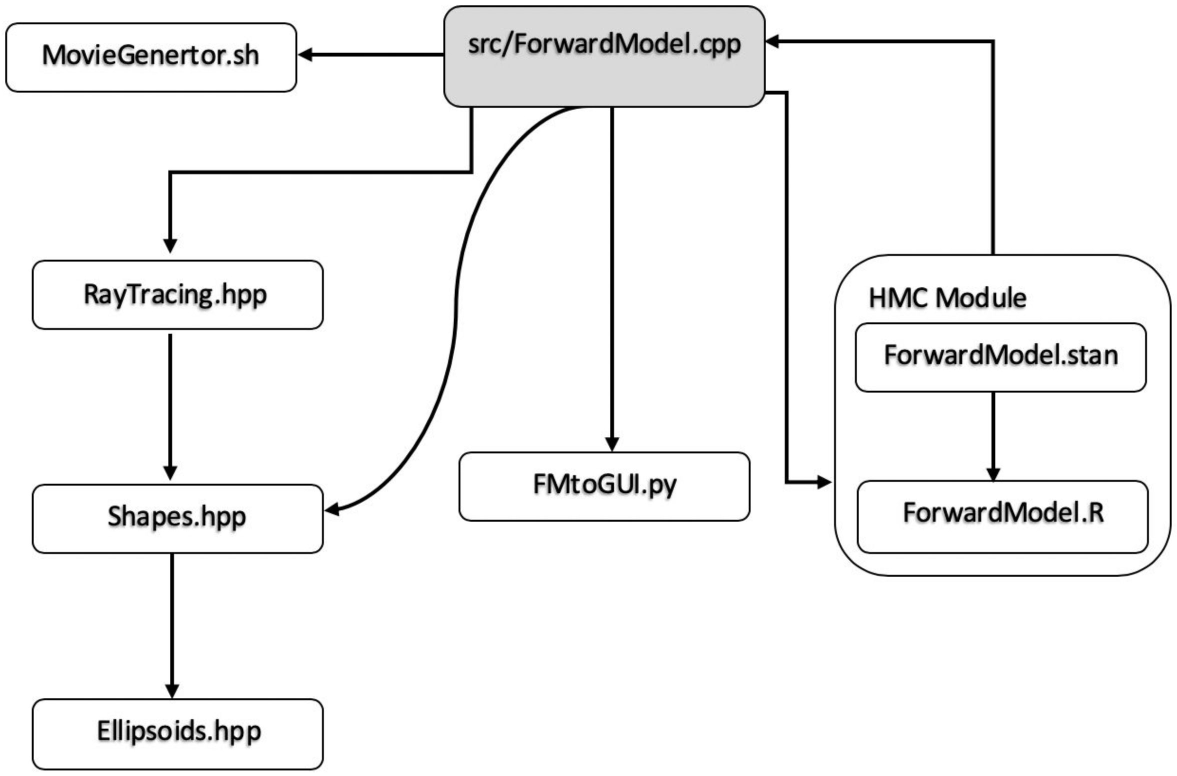

Figure 1: The dependencies of the conceptual modules

Triaxial Ellipsoid
The triaxial ellipsoid submodule adds functionality to the previously implemented shape module. When given the appropriate parameters, this new submodule will be capable of performing calculations for the forward model using triaxial ellipsoid objects. Additionally, it has the ability to display a triaxial ellipsoid object when calling the forward model. Both functionalities are created by using the forward model module in conjunction with the shape and ray tracing modules. The aforementioned parameters will be supplied by the users and will dictate how the triaxial ellipsoid object is shaped and displayed.

Hamiltonian Monte Carlo (HMC)
The HMC module will extend the functionality of the previously implemented forward model module. This new module will use two submodules to serve the purpose of finding the best set of estimated parameters to match the observed light curve to the predicted one. Additionally, the module will show how well our users have pinned down the values that they are interested in, and will visualize this cloud of solutions using a model.

The parameters that are being estimated will be determined by the users. Furthermore, the data for the observed light curve will be provided to the HMC module by the users, while the predicted light curve will be produced using the forward model with the constant parameters provided by the users.

Graphical User Interface (GUI)
The GUI module will extend the functionality of the previously implemented forward model module. This new module will allow our users to enter parameters into the forward model from the GUI rather than entering them at the command line. 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.

Movie Generator
The movie generator submodule will add new functionality to the forward model in the form of a movie generation script. When the script is run, it will take in a collection of images produced using the forward model and condense them down to a .MP4 file. The script will be accessible from the folder previously mentioned where the collection of images will be stored.

These modules and submodules work together with the previously implemented software solution to provide additional, beneficial functionalities for users. The dependencies, use cases, and design of each module are thoroughly detailed in the following section.

4. Module and Interface Descriptions

In an effort to retain modularity, the overall design involves separate modules and submodules. Each new major module or submodule remains semi-independent and only loosely interfaces with the others. Since there is minimal coupling, the way the modules are linked is loosely defined. Figure 2 shows the comprehensive blueprint of the modules and submodules previously implemented, what we plan to implement, and how they depend on each other.

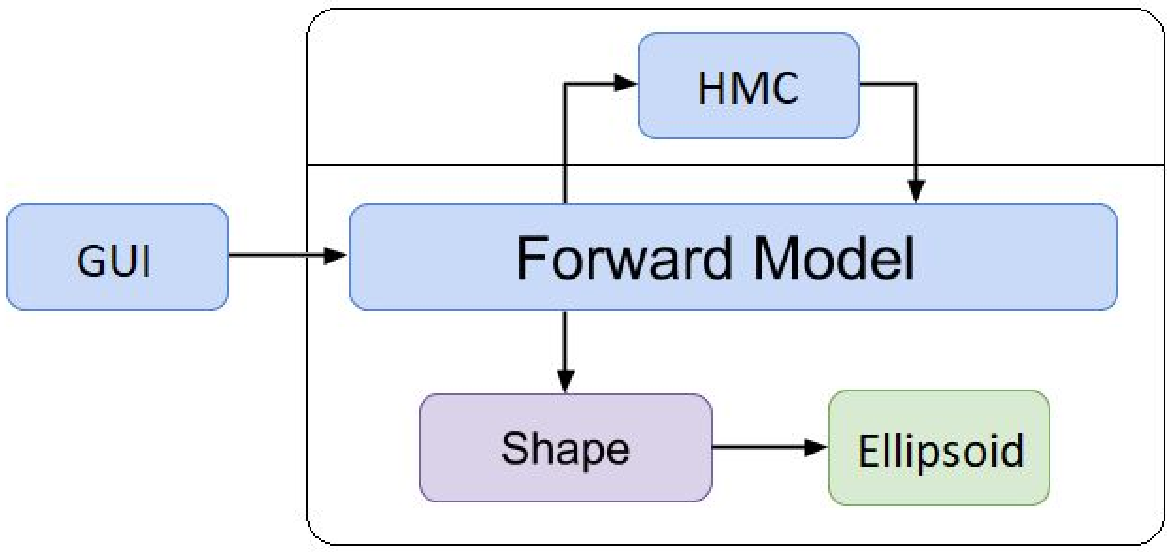

Figure 2: How the code base is split into conceptual modules

‘

4.1 Ellipsoid

The Ellipsoid submodule is the third and final extension of the Shape module. It holds the necessary equations to form a triaxial ellipsoid shape which is generated by the Shape module.

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

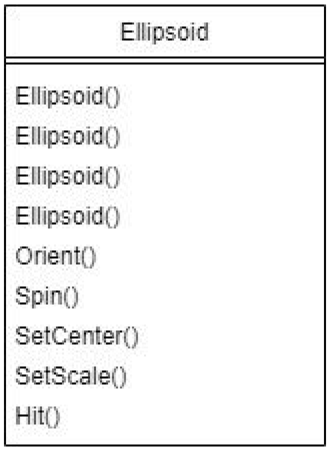

Figure 3: UML diagram of Ellipsoid.hpp

The Ellipsoid submodule defines how an ellipsoid is visually displayed and how it can be manipulated. There are four separate constructors and five methods used to help accommodate these manipulations, as seen in Figure 3.

<u>Constructors</u>

- Given a radius
- Given a radius and pole
- Given center and radius
- Given center, radius, and pole

Methods

- Orient
  - Realigns the ellipsoids via rotation about a given pole so that it is oriented along the pole.
- Spin
  - Rotates the ellipsoid on the shortest axis to simulate its rotation, using rotation speed, epoch, and time.
- SetCenter
  - Using a given center, sets the ellipsoid's center which dictates where the object is simulated.
- SetScale
  - Using the original radii given, calculates scale based off given scaling factor and saves into radiusX, radiusY, and radiusZ; also sets the matrix with new radii
- Hit
  - Calculates the point of origin of the ellipsoid using the ray direction and the triaxial ellipsoid's matrix; sends this information to hitRecord which then calculates whether the object is hittable

Variables

- radiusX
- radiusY
- radiusZ
- originalRadiusX
- originalRadiusY
- Eigen::Matrix3d M
- Eigen::Matrix3d rz
- Vector3d pole

4.2 Hamiltonian Monte Carlo (HMC)

The HMC module provides the required tools for computing parameter estimates and displaying those estimates to our users. It consists of the following submodules: ForwardModel.stan and ForwardModel.R.

4.2.1 Forward Model Stan Model

The ForwardModel.stan submodule provides the Stan Model necessary for using the RStan package.

Dependencies

- RStan package
- External C++ header file

Use cases

- Provide a method for dictating which parameters our users wish to keep constant and which to estimate.
- Provide a method for dictating how the data will be modeled.

Design

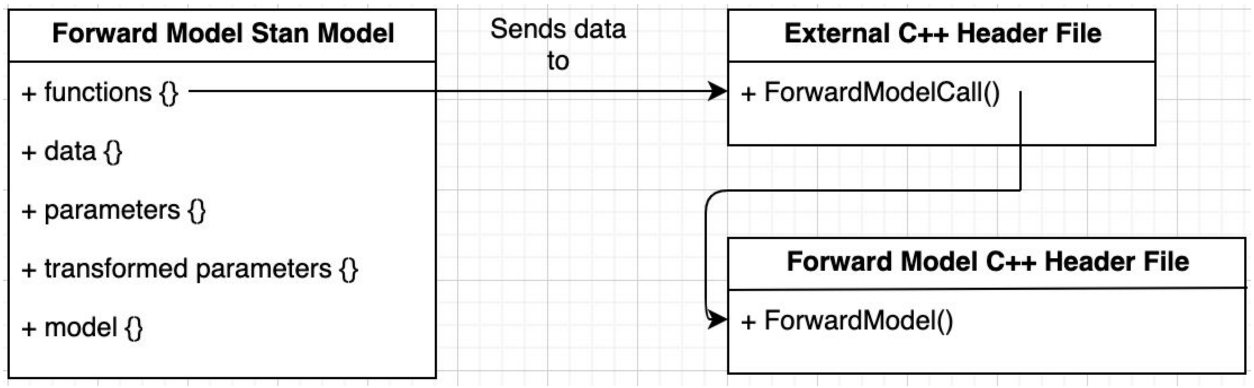

Figure 4: UML diagram of ForwardModel.stan

A Stan program is organized into a sequence of named program blocks, the bodies of which consist of variable declarations, with some blocks occasionally containing statements.

<u>Program Blocks</u>

- Functions block

    - This is where function definitions are included to be used within the user-defined functions.
- Data block
    - The data supplied to the Stan model is stored in the variables declared in this block. This is where the user declares the data types, their dimensions, any restrictions (i.e. upper = or lower =, which act as checks for Stan), and their names. Any names the user gives to their Stan program will also be the names used in other blocks.
- Parameters block
    - This is where the user indicates the parameters they want to model (estimate), their dimensions, restrictions, and name.
- Transformed parameters block
    - This is where the user includes variables to be defined in terms of data and parameters that may be used later and will be saved.
- Model block
    - Statements declared here are used to help define the model. This is where the user includes any sampling statements, including the "likelihood" (model) that they plan to use. The model block is where the user indicates any prior distributions they want to include for their parameters.

Send Data to the External C++ header file

The functions block will send data to an external C++ header file, which in turn calls the licht-cpp shared library. This shared library contains the compiled C++ files necessary to run the forward model in conjunction with the Forward Model Stan Model.

Parameters

There are a total of 56 parameters required to use the forward model. Since the function block within the ForwardModel.stan file will be sending these parameters to an external C++ header file (which in turn calls the licht-cpp shared library), these parameters will be required as input with the ForwardModel.stan file. For a full review of all 56 parameters, please review the Software Design document[1] created for the previous iteration of this project.

[1] https://www.cefns.nau.edu/capstone/projects/CS/2019/PairedPlanet-S19/docs/Software_Design_FD.pdf

4.2.2 Forward Model R Script

The ForwardModel.stan submodule provides an interface necessary for fully utilizing the RStan package.

Dependencies

- R compiler
- RStan package
- RStudio
- External C++ header file

Use cases

- Provide an environment for which the Stan model can be executed, and used to display data produced from the Stan model.

Design

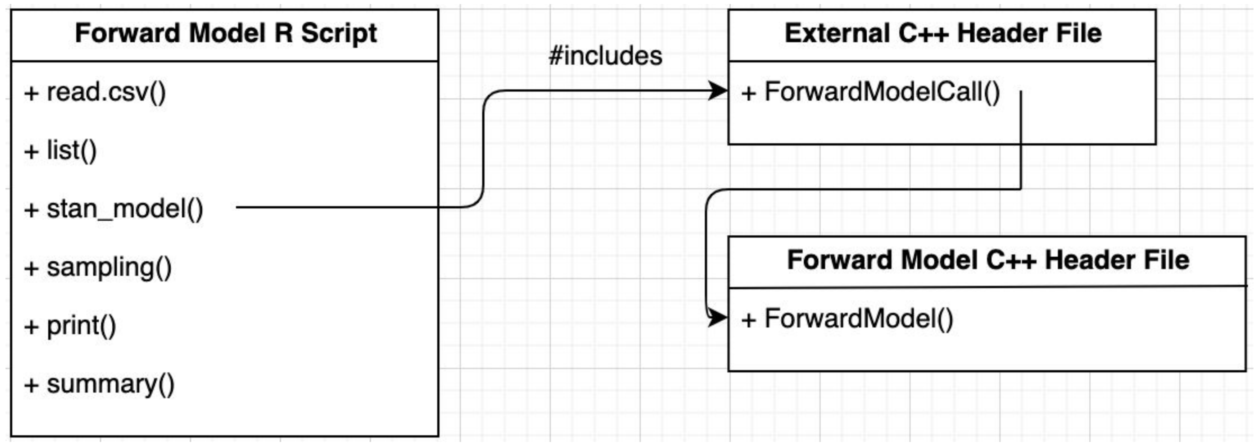

Figure 5: UML diagram of ForwardModel.R

The R script is organized into a series of function calls that allow for the import of theStan model, its necessary external C++ header file, and the storing and sampling of data provided to the model.

<u>Functions</u>

- read.csv()
    - Reads in data for the Stan model into an R object from a specified .csv file.

- list()
    - Parses the data that are contained in the R object returned by read.csv().
- stan_model()
    - Builds an instance of the specified Stan model. The Stan code used to build the stan model may be contained within the R script, however, this is not necessary and instead may be pointed to in a separate file. An external C++ header is included to provide the implementation of the "ForwardModelCall" declared in the function block.
- sampling()
    - Samples the Stan model and stores the returned data within an R object. The number of iterations, chains and other diagnostic arguments for the Stan model are set here.
- print()
    - Prints a summary of the data contained in the R object determined by sampling(). Included in the summary is model specific information(number of chains, number of total iterations, number of warmup iterations, etc.), quantiles, means, standard deviations (sd), effective sample sizes (n_eff), and split Rhats for all parameters. Date and time samples are drawn, and a brief explanation of n_eff and Rhat is provided.
- summary()
    - Creates an R object that contains specific information selected by the user from the R object returned from sampling().

Include the External C++ header file

The stan_model() function will include the external C++ header file, which in turn can be used by the ForwardModel.stan file.

4.3 Forward Model Graphical User Interface

The Forward Model Graphical User Interface 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:

- Python Version 3.8.0+
- Ctypes Python Module
- Tkinter Python API
- External C++ Shared Library
- Plotly Python API
- ForwardModel.hpp

Use Cases:

- Provides an alternative method for parameter input for the forward model.
- After the forward model has run, it provides the ability to graph light curves for data analysis.

Design:

The Forward Model Graphical User Interface module uses the functionality of the forward model. It uses 4 main functions and 1 constructor.

<u>Functions:</u>

- 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 Interface 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. It takes in the data set and creates a lightcurve through the Plotly API.

Constructor:

- createGUI
    - This constructor is used to create the graphical user interface object. The reason the interface must be an object is so that data can be passed to the object parameter fields.

Parameters:

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

4.4 Movie Generator

The movie generator will allow users to stitch together .jpeg files into a .MP4 file through the use of the OpenCV library. The movie generator will allow users to look at how the predicted system behaves and moves within space.

Dependencies

- Existing C++ Shared Library
- OpenCV C++ Library

Use Cases

- Allows the user to create movies of the images produced from the existing API.

Design

The movie generator will be built alongside the existing API. It will be a submodule that gathers the images to be sown together for the movie. It will take no parameters but will be generated by 1 main method and 1 constructor.

Functions:

- gatherImages
    - This function will be used to sow together the selected images to generate a movie.

Constructor:
- createVideo
    - This constructor will take in the images from gatherImages and use the OpenCV functionality to stitch the images together. This will return a .mp4 in the output files.

5. Implementation Plan

It is imperative that we section our project into reasonable milestones. The project has been strategically split up to break down triaxial ellipsoids, the HMC, the GUI, and the movie generator into realistic, specific goals. These goals will construct a plan for development that encompasses the rest of the semester. We have had active scheduling and task assignments in order to make sure the implementation of our solution is successful.

In the fall, our team broke down the project into four major sections. We considered our interests and experience to choose among working on triaxial ellipsoids, the HMC, the GUI, and the movie generator accordingly. Jes and John are assigned to work together on the triaxial ellipsoid problem because of their experience with mathematics and their interest in ray tracing. Batai and Matt are assigned to work on the HMC problem because of their experience with C and C++, and they strive to challenge themselves by learning new topics. Moreover, they are interested in the application of statistical computation and modeling-based. Brad is assigned to the GUI and movie generator because of his passion for creating GUIs, working with Python, and learning new frameworks.

The Gantt chart in Figure 6 makes use of colored bars to illustrate which tasks have been assigned to which team member. The purple tasks are assigned to John and Jessica, the blue tasks are assigned to Batai and Matt, and orange and green tasks are assigned to Brad.

As seen in the Gantt chart, the major phase titled “Alpha Prototype” is a major milestone upcoming programming task and once all of the major features are implemented, this large task is to be broken down and assigned to team members.

After the Alpha Prototype Demo has been completed, we will optimize our solution and implement the video generator as a stretch goal. We will dedicate time, after the core features of our additions have been implemented, to enhance the features of our implementations by optimizing the code. This optimization involves low-level changes, code organization and commenting, and parallelization for certain sections of the API.

6. Conclusion

Space is an exciting, mysterious field. Despite thousands of years of research, humankind has only begun to scratch the surface of understanding the universe. To this day, there are numerous objects floating in the Kuiper Belt that we have yet to explore. Up to 30% of those asteroids are considered to have at least one secondary in their orbit. Studying these binary systems gives the power to unlock answers for how they are formed, how their orbits work, and how they fit in to the solar system.

Our clients Dr. Audrey Thirouin and Dr. Will Grundy work at Lowell Observatory and observe these binary systems in the Kuiper Belt, which they accomplish via software that models binary systems. Since space voyages are time consuming and costly, telescopic observations from Earth are the most efficient way to form hypotheses of binary systems. Our clients can prepare observed light curves but cannot easily confirm the correctness of the characteristics formed from it. Our project is aimed to help Will Grundy and Audrey Thouirin at Lowell Observatory solidify these attributes.

The Hamiltonian Monte Carlo algorithm we are implementing will allow for computed estimations of parameters for observed light curves, allowing for easier gathering of characteristic information for a binary system. The addition of the triaxial ellipsoid shape will accelerate render speeds for visualization without a high cost of accuracy lost. The GUI will streamline the data input process for users, easing their use of the system. Finally, the video generator will compile a movie for users to use at will.

The formalization of a conceptual design instills confidence in both the developers and the clients. It provides a blueprint for how to implement our project and hence makes meeting the requirements more straightforward. It also mitigates the risk of running into issues during development. With this extensive planning, there are no foreseeable roadblocks and the independently testable modules ensure a robust solution. Team Andromeda is excited to implement this design and create an efficient, intuitive solution that exceeds the clients' needs.

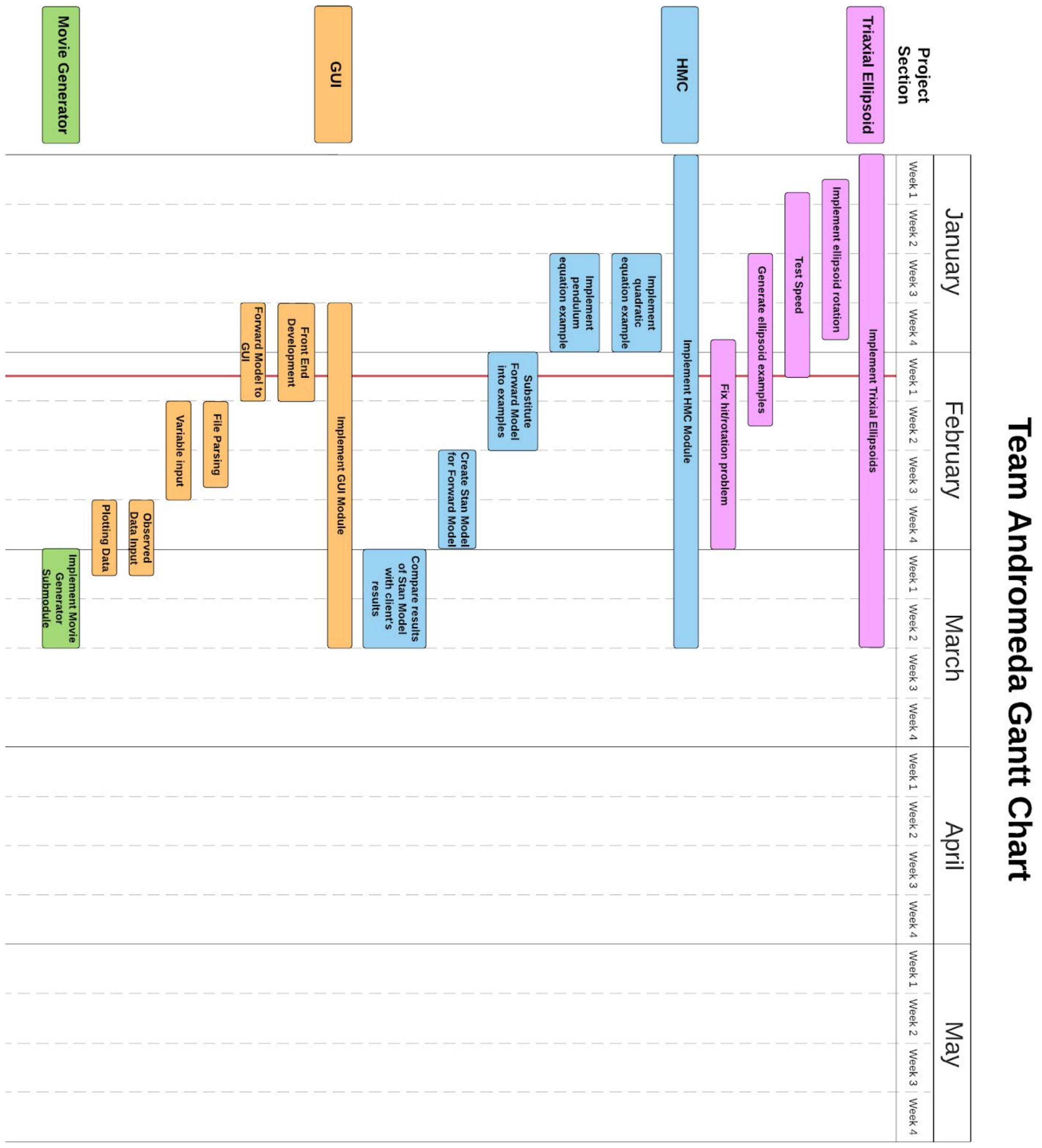

Figure 6: Gantt chart as of 2/5/2020