Requirements Specification

Version 1

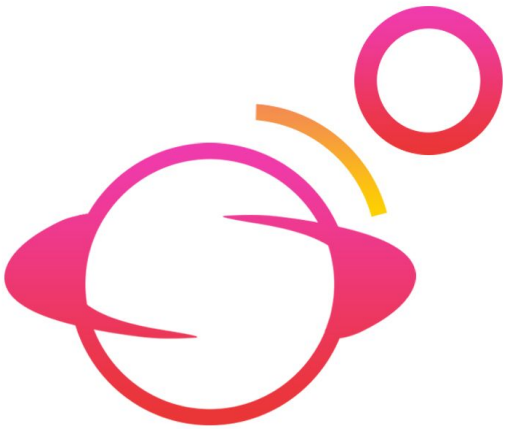
11/26/18

Sponsor
Lowell Observatory

Clients
Audrey Thirouin, Will Grundy

Team
Paired Planet Technologies
Zach Kramer, Brian Donnelly, Matt Rittenback

Mentor
Isaac Shaffer



Accepted as baseline requirements for the project:

For the client: _____ _____ *For the team:* _____ _____
Signature *Date* *Signature* *Date*

Table of Contents

1. Introduction

1.1 The Domain

Space exploration has always enthralled humanity. Every year, billions of dollars are spent trying to understand what is in the Solar system. Humanity sends probes to other planets and even to the vast space between stars. Humans have not stopped looking for answers in space since the time of Galileo. This search has driven technological development in all areas. Velcro, computers and GPS are just some of the technological byproducts of space exploration. Every day, observatories like Lowell are gathering information to expand the understanding of the space humanity lives in.

1.2 The Clients

The clients, Dr. Audrey Thirouin and Dr. Will Grundy, work with Lowell Observatory to analyze astronomical data. They focus on analyzing data about smaller objects farther from Earth in space which are harder to observe directly. Right now, they are working on modeling binary systems out in the Kuiper Belt. To do this, they need to use special techniques which make the most use of the data available. For most objects that far away in the Solar system from Earth, only a single pixel of light can be observed. That single pixel can be used to determine a luminosity at a given point in time. These luminosity recordings can be combined together to form something called a light curve. A light curve is a graph of brightness values over time.

1.3 Light Curves

Light curves can be used to infer properties about the objects that generate them. For example, an asteroid that is non-spherical will reflect more light when a larger amount of surface area is reflecting light from the Sun to the observer. Since the object is reflecting more light at certain points in its rotation, the brightness will be different depending on when it is observed in its rotation. Those brightness values can then be graphed. In most cases this will make the light curve sinusoidal. The rotational speed can then be found based on the period of the curve. The amplitude of the curve can be used to roughly guess at the proportions of the object. A large number of other characteristics can be found using light curves by a clever astronomer.

1.4 Binary Asteroid Systems

The clients want to make use of light curves to better understand binary systems. A binary system is composed of two objects that orbit each other about a point in space called a barycenter. The gravity from both objects affect the other, sometimes causing tidal locking and precession. Binary systems introduce new challenges but also opportunities for light curve modeling. The two objects will cast shadows on each other which can be used to determine surface features and shapes of the objects.

1.5 Modeling

The clients want software that can model these binary systems and generate light curves. They plan on using the solution from this project to come up with models that fit the observed data of binary asteroid systems. Currently, the clients are modeling with fragmented code that is slow and lacks some functionality. This project's solution will allow the modeling of binary systems quickly and accurately.

2. Problem Statement

2.1 Problem Overview

The clients at Lowell Observatory lack a comprehensive software solution for modeling binary systems. What they need is software that can take in properties of the asteroid system and return a theoretical light curve. They can then take the theoretical light curve and compare it to an observed light curve from the same time period. The clients can compare the two light curves and determine if the input system generated a correct light curve. If the light curve is correct, then they know that the asteroid properties they ran the model with are possible properties of the actual binary system. If the light curve model does not closely resemble the observed light curve, then they can change some of the properties and try again. By using minimization techniques (such as MCMC or Metropolis-Hastings) they can adjust the parameters until they generate an accurate light curve model.

Once the clients have a set of parameters that generates a theoretical light curve that is within a set tolerance of the observed light curve, they evaluate the asteroid system parameters and determine if the solution is realistic. If the solution is not realistic, they can redo the modeling until they come up with parameters that they think are accurate.

2.2 Current Implementation

Currently, the clients lack the software to take in asteroid system parameters and generate a theoretical light curve. They have pieces of software that they can combine with hand work to generate a partial solution. This method is not capable of handling large amounts of data or complex binary asteroid systems. The fragmented software also needs a number of small scripts to be written every time it is run to account for changes in how the software fragments need to be used. The lack of a cohesive software structure makes it nearly impossible for other astronomers to use. Astronomers would need to rewrite extensive portions of the code to make it usable for their research.

The modeling can be computationally expensive for a few reasons. First, ray tracing needs to be done. The ray tracing has to check and see if the light from the Sun will be reflected at the correct angle for a telescope to observe it. The clients' current implementation does not make use of a framework and is written in an interpreted language. This makes the current ray tracing slow and bottlenecks the whole program. Second, with two binary objects in orbit about a barycenter, it is possible that they can cast shadows on eachother. These shadows need to be accounted for when the ray tracing is done.

All of the problems that the clients are running into can be solved by the solution. The solution looks to solve the integration, performance and modeling problems efficiently and modularly.

3. Solution Vision

3.1 Solution Steps

The solution for modeling binary systems is to create a modular API. The clients want an API that can be called from within IDL, so C has been the chosen language for this project. The C API that will be developed will need to be able to do the following:

1. Take in input parameters and accuracy settings
2. Create objects in a coordinate space based on the Sun
3. Set objects into the correct orbital rotations for the given time
4. Perform ray tracing on the objects to generate a matrix of brightness for each tile of the objects
5. Save what tiles are visible to the observer and their brightness for rendering
6. Sum up the total brightness of every tile and save it
7. Advance the time and repeat steps 3-7 for each time interval
8. Take all of the brightness values and graph with their corresponding time stamps

9. If a rendering is wanted, use the saved information from step 5 to render a frame for each time stamp, then stitch them together for a short video.

3.2 Solution Methods

The 9 step solution will tie together all of the fragmented components that the clients already have. It will also use a framework for ray tracing that adapts to the hardware available and maximizes the efficiency. The chosen technologies for this project include: Vulkan for ray tracing and rendering, Eigen libraries for vector math, and OpenMP for parallelizing sections of the code.

The Vulkan API will be used for ray tracing and rendering in steps 4 and 9. This API will handle the majority of the work with ray tracing. The math for ray tracing is done in function calls to the API, and is already optimized. Since the objects will be tiled for the ray tracing math, the rendering can be done with the same object later on. Vulkan also runs on whatever hardware is available.

The math in steps 2, 3, and 6 of the solution steps are all vector equations. The Eigen libraries will be used to do the vector math because it is already optimized and parallelized internally. OpenMP will be used throughout the solution for both data and task parallelism. This will make the most use out of whatever idle CPU cores are available, and increase the performance of the solution.

The solution will be well documented. Each function call and object will have an explanation and clear commenting to allow the code to be modified in the future. The documentation will also allow users other than the clients to make use of the API without having to fully understand the code. This solution meets the requirements of the clients which are detailed in the next section.

4. Project Requirements

The solution will be a modular C API with a variety functions designed to work together optimally. This API will take in user input containing orbital parameters to generate a theoretical light curve. Additionally the features of this API will need to be documented to be accessible by other astronomers.

In developing this solution, the clients have granted significant freedom in most of the design decisions for the project. However, there are a few core requirements established by the clients to structure the project. These requirements can be broken down into 3 subcategories: Functional, Performance and Environmental. The following sections will outline how each requirement impacts the project and how the solution will have to consider each requirement.

4.1 Functional Requirements

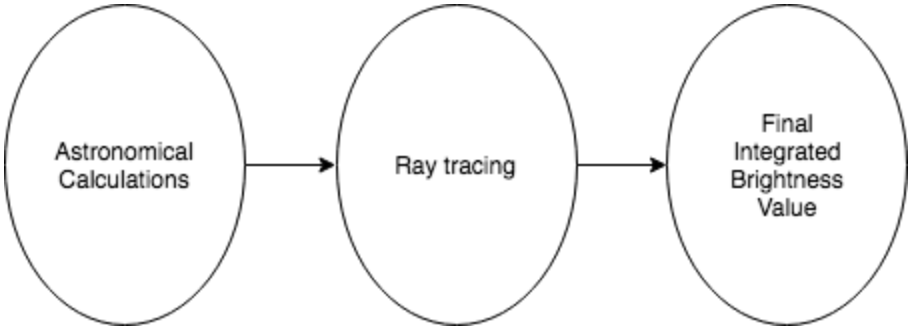


Figure 1: The Forward Model

For this project there is one major use case that must be satisfied, the Forward Model, which is displayed in Figure 1 above.

Implementing the Forward Model involves 3 major steps: performing astronomical calculations, ray tracing the scene, and producing an integrated brightness value. This set of functions will be called multiple times in succession from a minimization routine with different or unchanging time values entered at each iteration. The final iteration of this minimization routine will produce a final matrix of brightness values which represents a light curve.

The user must also have the option to request a rendered image of the binary system modeled in the ray tracing step.

4.1.1 Accept User Input

The software must provide the capability to read in large input files with a predefined format similar to the format of existing ephemeris tables. Although currently the format in the clients' solution is not strongly established, ephemeris tables from NASA will be used as a base reference in terms of complexity and types of data. These tables provide the majority of the input data necessary to perform the astronomical calculations in the first step of the Forward Model.

Input files will be located through strings passed into the API. Additional values can be passed into the API to provide additional data not found in ephemeris tables, such as the times to model, or to adjust the accuracy of the model. The latter includes modifying the resolution of facets used to create the objects in the ray tracing. All input data presently accepted in the clients' existing solution must be supported by the API, either through input files or function calls.

4.1.2 Adjustable Accuracy

The client needs options that can adjust accuracy to have the ability to run quick tests to determine if a model is a rough approximation of their observed system. Then tests with higher accuracy can be run afterwards to find a higher resolution model.

1. Object resolution

The largest impact on accuracy will be the resolution of the object models used for the Forward Model. This parameter will be taken in when the Forward Model is called. The resolution will determine how many facets are generated to define the object. An increased resolution will cause the generated object to have more facets. The increase in facets will allow the ray tracing to calculate reflection angles with a higher level of precision.

2. Accuracy presets

The software will provide a number of preset accuracy settings. Different accuracy settings will enable the clients to run various scenarios for testing purposes. For this solution the accuracy settings will include 3 different options:

1. Best Performance
2. Best Quality
3. Balanced

The level of accuracy used to run the functions inside API will vary on the setting selected by the user. These settings will adjust various internal parameters used for both the astronomical calculations and ray tracing. Parameters to be adjusted inside the calculations include the level of tolerance for Newton's method and using lookup tables over calculating values with trigonometric functions.

Best Performance will adjust the parameters to produce the quickest result at the cost of accuracy in the results.

Best Quality will adjust the parameters to produce the highest resolution result with no reduction of accuracy. This setting will be the most time expensive of the 3 settings.

Balanced will adjust the parameters to produce a result that minimizes accuracy loss while trying to optimize runtime in the calculations. The ray tracing will produce the best resolution possible with the optimized calculation results.

The speed difference between these settings will not vary greatly when compared in a single iteration. However, since the Forward Model will be called numerous times and perform these calculations for each iteration, the difference in runtimes will add up over time.

4.1.3 Perform Astronomical Calculations

The first step of the Forward Model is to perform numerous astronomical calculations to generate the data necessary to build a model in the ray tracing step. For the purposes of the ray tracing, all calculations and data will be computed with the Sun as the central point. The math in this step can be broken down into 3 major functions:

1. Work out the times at the binary system

One iteration of the calculations will revolve around a specific time and provide essentially a snapshot of the object's exact location in space. This time value will serve as a key to find all relevant orbital data for the objects of the binary system in the input file provided. The specific time will be either entered directly by the user inside an API call or calculated by the API itself. Calculating the specific time for a binary system will be based on the light travel time reflected off the objects towards the observer.

2. Calculate binary system location

The location of the objects in the binary system must be calculated using a specific time determined from the previous step of the calculations. Keplerian orbital elements must be used in conjunction with the specific time to determine the exact location of system.

3. Calculate orientation of bodies in binary system

In binary systems, the two objects can be in a variety of orientations relative to each other. Utilizing user input that provides the rotational axis, rotational velocity and starting orientation of the binary system along with a specific time value, the orientation of the two objects at an exact point in their rotation will be determined.

4.1.4 Ray Tracing

After the first step of the Forward Model has been completed, the data necessary for the second step has been generated. The second step will use ray tracing to construct and position a model of the binary system based on the calculated data and user input. The ray tracing within this step of the Forward Model can be broken down into five steps:

1. Generate objects composed of triangular facets

To start the ray tracing, two objects composed of triangular facets will need to be generated and these will serve as a simulation of the binary system. By default, the two objects will be spheres, however there will be other basic spherical shapes offered as options through an API function call.

2. Position the binary system

The objects simulating the binary system must be placed inside the ray tracing model relative to the light source, which corresponds to how the actual binary system is positioned relative to the Sun. When positioning the objects in the binary system inside the ray tracing model, the Sun has to be used as the central point of the coordinate system. This central point will be critical in building an model with the objects properly aligned with the Sun and the observer.

3. Trace rays from light source

A light source will be defined as the central point within the ray tracing model. This central point will cast out rays towards the binary system and these rays will be tracked individually to trace their path and identify what the rays collide with.

4. Test if rays intersect a facet

Each facet of an object in the binary system will be evaluated to determine if a ray intersected the surface of an object. This evaluation determines what parts of the objects are lit up and will eventually display potential irregularities on the surface of the asteroids.

5. Check and record facets reflecting light towards observer

Light reflected off the binary system may not reflect towards the observer and therefore the solution will have to distinguish which of the facets reflect light towards the observer. These facets are relevant to calculating the integrated brightness value and rendering an image.

4.1.5 Implement Hapke Modeling

The solution will implement a function to perform Hapke modeling and calculate the light-scattering properties of the surface of the objects in the binary system. This function will help identify and determine the irregular surface of the objects.

4.1.6 Generate Integrated Brightness Values

The final step of the Forward Model will be to return a matrix of brightness values to the user, corresponding to the input time values. This brightness values will be calculated based off the results of the ray tracing steps. A matrix of these values can be interpreted to generate a light curve graph.

4.1.7 Render an Image

The user will have the option to request additional output from the Forward Model call in the form of a rendered picture. This output will be generated through the use of ray tracing. Additionally, this output must be modifiable by input parameters to adjust the resolution or pixel count of the image.

A single specific time or range of times will be passed in from the user and an image, or set of images, of the binary system will be rendered based on the entered time.

The image will display the two objects of the binary system, with two key characteristics. First the shapes of the objects will be variable based on the shape the user selected for each object. Second the facets on the objects will display varying greyscale values based on how much light the facet reflects. Lighter colored facets represent parts of the asteroid that reflect more light than the darker colored facets which represent parts that do not reflect as much light.

4.2 Performance Requirements

Currently the clients' solution is not complete and requires a substantial amount of overhead time to set up the data to be parsed. A script is needed to produce a light curve or to render an image. Each time the clients want to model a new binary system, they have to recreate the necessary data and rewrite the script. The ray tracing in the clients' solution is written in an interpreted language which does not run efficiently compared to using C libraries for ray tracing.

4.2.1 Calculate Low-Resolution Light Curve

The solution needs to generate the data required to determine a low-resolution light curve. The clients gave a rough estimate of this task taking between 30 and 120 seconds on a workstation laptop. A workstation laptop will be defined for this project as a laptop with both a stronger CPU and faster GPU than that of an average notebook laptop. The steps to complete this task is composed mainly of quick mathematical calculations, thus this requirement should be easy to accomplish.

4.2.2 Render High-Resolution images

Generating a high-resolution image will involve ray tracing objects with a higher count of facets which requires more processing to analyze each facet to render the image. To render a high-resolution image of a calculated binary system, this should take the solution 3-5 minutes to accomplish on a workstation laptop.

The clients' focus is on a complete solution to save them time and money as the current solution has recurring overhead costs. The solution simply being complete will be a significant performance improvement over the clients' current implementation. However, this solution has to satisfy these performance requirements as well as certain environmental requirements covered in the following section.

4.3 Environmental Requirements

4.3.1 Linux Systems

The solution needs to be able to be compiled and run on Linux systems, specifically Ubuntu and Red Hat. It does not need to be able to run on any other type of operating system. Support for MacOS is a stretch goal.

4.3.2 Supports CPU and CPU-GPU Configurations

The solution must be able to run on computer that only has a CPU, but also make use of a GPU if one is available. Any code or framework that the solution has needs to be compatible with laptops and be able to run within the performance requirements without access to more powerful hardware. The software needs to adapt to the hardware available without any additional input from the user.

4.3.3 Callable From IDL

The solution needs to be callable from within an IDL environment. IDL is an older, interpreted language, mostly used by astronomers, and does not provide support for many modern languages. One language it does support, though, is C. The parameters passed through to the code will be primitive or IDL types and need to be handled accordingly.

4.4 Conclusion

The client needs to generate a light curve from binary asteroid system parameters. They need a documented API that has all of the functions necessary to generate a Forward Model. The API needs to have functions that are modular and a function that incorporates all of the work involved with creating a Forward Model.

The Forward Model generation is the major function that needs to be implemented by the API solution. This major function will call upon many other functions inside the API to complete astronomical calculations, ray tracing, and calculating a final brightness matrix.

The solution must be able to calculate a low-resolution light curve in between 30 and 120 seconds on a workstation laptop. It must also be able to generate a high-resolution image in 3-5 minutes. Environmentally, the solution has to run on Linux systems, support CPU and CPU-GPU configurations, and must be callable from IDL.

5. Potential Risks

These requirements naturally introduce certain challenges that need to be accounted for. Overall the requirements are relatively low-risk, since they are in the field of refactoring and optimizing, not creating a solution from scratch. However, there is a reason why the software needs to be rewritten and refactored, and there is potential to refactor in a way that does not support future requirements.

5.1 Scope Expansion

The first major risk is that the scope expands significantly to the point that it breaks the solution. It is also highly likely to happen. The solution that is being built is not a one-off solution, this is software that will be modified in the future. The clients have many ideas about future improvements and features, some of which rely heavily on utilizing existing code.

If the API was solely designed for the capstone requirements, it could need to be significantly refactored to add some of the new functionality that the clients may want. This is a normal risk for every software project that is not a one-off solution, but there is some insight that makes this a more specific risk. Specifically, the requirement is to implement a Forward Model. Eventually the clients will implement, on their own, an inverse model. An inverse model is essentially a fitting algorithm for the Forward Model.

5.1.1 Mitigation

A modular design can be used to account for a feature without knowing the implementation details. The currently translated code already partially does this. For example, there is a module for handling the orbits and a module for the ray tracing.

To incorporate a feature without knowing the implementation details, a black-box module can be utilized. This means that the solution will not need to know the implementation details of this module, just how it will interact with the other modules. By doing this the solution can prepare the existing modules to account for this new inverse model feature. This may mean breaking up the Forward Model into more functions than currently necessary, since the inverse model may need only specific functionality from the Forward Model.

There is no perfect prevention to scope creep -- there is no realistic way to account for all possible features in the future. However, by doing extensive requirements acquisition, the conceptual goals of the near future are outlined for clear understanding and preparation can begin for the in-development solution of such such features.

5.2 Varying Ray Tracing Results

Another risk that has been identified is that when the ray tracing module is implemented, there is a high probably that the results will differ from the existing solution. This is important because the entire model's accuracy is affected by the ray tracing.

The ray tracing code that the clients are using is handwritten and specific to the purpose, whereas a framework is generalized (the low-level details are often hidden to the user). A reference needs to be established that shows how accurate or inaccurate the results are. It is unsafe to assume that they will be similar to the current solution.

5.2.1 Mitigation

To mitigate these risks, one needs to first establish references. There are two sources of references: the handwritten code and existing observations.

The easiest reference to establish is to run a data set through the existing handwritten code, then run the same dataset through the new module. Then one must identify if the results are even in a ballpark range. It may be challenging to debug the two solutions and see where specifically the calculations differ. The most realistic approach is to identify probable areas of code that could differ in results, such as one big function call to a framework versus dozens of lines of handwritten code.

A reference can also be established against known observations, though that is a larger task, since it would involve running through the entire Forward Model and not just the ray tracing. This reference is more so a backup in case the more accurate solution -- handwritten versus framework -- cannot be determined.

Once the references are established, unit tests should be written for the ray tracing module. Unit tests will constantly verify the results. This will occur after having already written an initial ray tracing implementation. Many iterations of refactoring and optimizing the ray tracing module will likely follow, since it is the bottleneck and meat of the code. During this phase, the unit tests will indicate when modifying the code has changed the accuracy.

In summary, establishing references along with writing unit tests will significantly mitigate the risk of the ray tracing module producing inaccurate or unexpected results.

6. Project Plan

This project started by first establishing how the team would work together and then moved immediately into assessing the project and gathering requirements. This can be seen in Figure 2, which is appended to the end of this document for readability purposes.

6.1 Current Work

One of the current tasks is translating the code. This means writing all of the core logic of the fragmented IDL solution into the language of choice, C. In reality this task belongs in semester two, since there is no immediate purpose for it. It does, however, provide a significant head-start. There already exists verified functionality, a deeply featured build system, a unit testing framework, and even continuous integration. The goal here is to be able to immediately go into the design phase next semester and work directly with C code.

Before a solution can be designed, there also needs to be demonstrations for the technologies that were chosen. There are three demonstrations. One for Vulkan, which performs ray tracing and rendering. One for the language of choice, showing it is compatible with IDL. And one for Eigen, which performs vector and matrix operations.

6.2 Future Work

The design phase begins in the second semester and consists of first conceptualizing a modular solution and then implementing it using the translated code. The language of

choice, C, is not object-oriented, but the architectural solution will still make use of modular design. From a user perspective, the API will just be a variety of functions, but internally it will be implemented in a modular manner. The implementation will involve direct refactoring of the already translated code, though the core logic will remain the same.

There is a big window of time for optimization, which was strategically planned for. Overall, a significant section of the programming work is already done. A complete solution will be delivered on time. The goal of this capstone is to deliver the best solution possible to the clients and meet their stretch goals, which in this case is optimization. This is simply because Paired Planet Technologies is passionate about the domain of space.

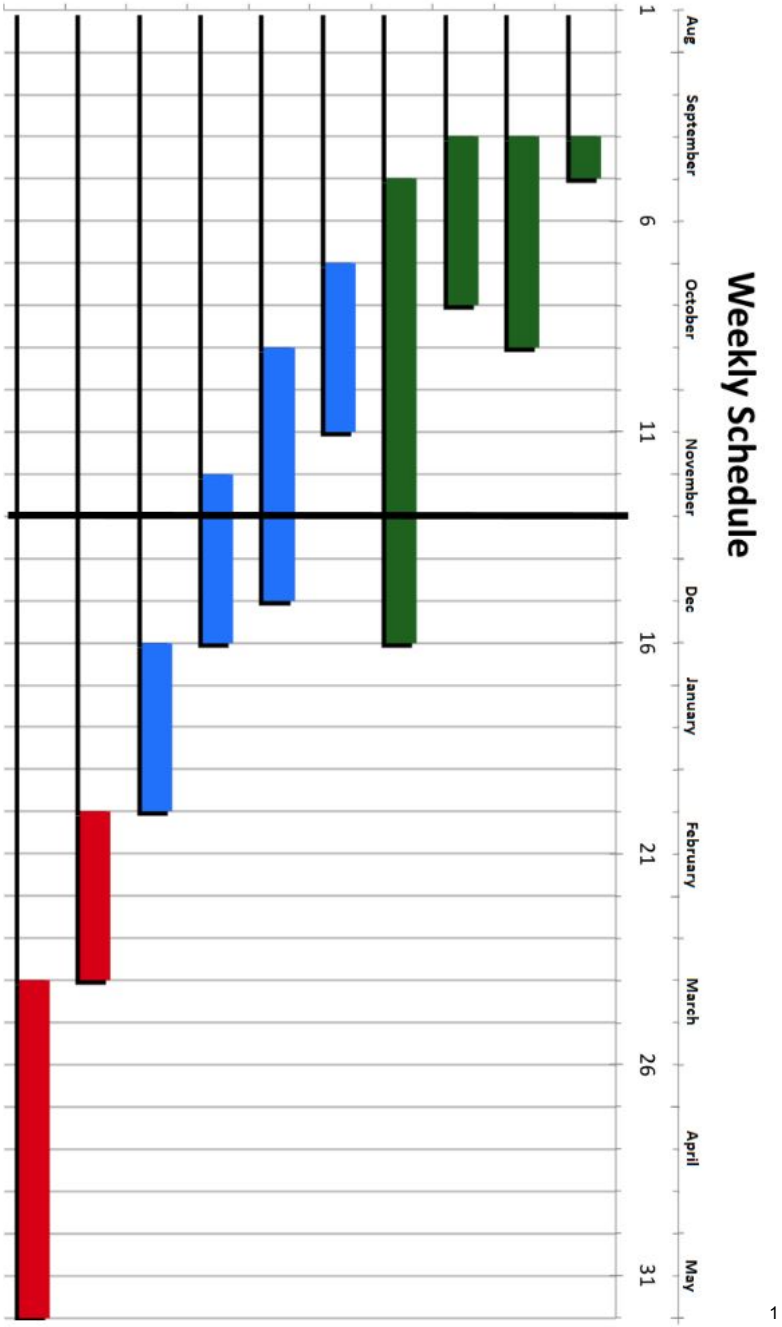
7. Conclusion

Space exploration has always enthralled humanity. Every year, billions of dollars are spent trying to understand what is in the Solar system. Binary asteroid systems have always been challenging to understand due to their small size. The clients, Dr. Audrey Thirouin and Dr. Will Grundy, work with Lowell Observatory to understand binary systems in the Kuiper Belt by using software to create models of them.

The issue is that these systems are so far away that only a single pixel of light can be observed. Thus, the software solution is relatively complex and the clients only have a partial solution for it. The current software is slow and fragmented. The clients would like an API that is significantly faster and integrated, meaning they can receive the model results in a single function call. The plan is to create a modular C API that uses Vulkan for ray tracing and rendering, and Eigen for vector and matrix mathematics.

Extensive requirements acquisition has been done for this document. This provides a contract with the clients stating the definition of a final solution. The details of ray tracing were dove into, which is the bottleneck of the current solution. This provides preparation for its implementation in the upcoming technical demonstration. The chosen solution was also laid out, providing a blueprint that solidifies the feasibility of the approach.

Overall the risks of the requirements were lower than anticipated, with both of the main risks falling under the category of “follow good software practices”. The performance requirements seem more than reasonable to meet and there are not many environmental requirements to meet. With a solid solution outline and a significant head-start for the second semester tasks, Paired Planet Technologies is confident that a top-quality solution will be delivered to the clients at the end of the second semester.



[1] Figure 2: Gantt chart showing previous, current, and future tasks