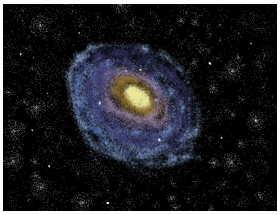

Team Andromeda

Technological Feasibility Assessment

November 8, 2019

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

We are Team Andromeda and our project is the Three-Dimensional Simulation and Visualization of Binary Asteroids. Our clients, Dr. Will Grundy and Dr. Audrey Thirouin, are astronomers at Lowell Observatory that research the Kuiper Belt and binary asteroids. The Kuiper belt is a region of the solar system beyond the orbit of Neptune which contains many comets, asteroids, and other small bodies made largely of ice. Many of the Kuiper belt residents are artifacts of the universe's beginning and there is much to be learned from them. For example, binary objects are a frequent occurrence in the belt. Binary asteroids are a system of two asteroids that are within the orbit of each other. The formation of these systems have only been explained in theories so far[1].

When our clients observe binary asteroids, they can only gather information about an object's brightness due to the distance it is observed at. The astronomers can use this brightness to generate a light curve. A light curve is a graph representing the intensity of light reflected from a celestial object over time. The best way for our clients to infer information about the object is by using lightcurves.

1.1 Purpose

Understanding how the universe around us functions is vital to space exploration, and objects in the Kuiper Belt give us insight into early formations of the solar system. Our clients use light curve graphs when they compare the generated light curve with observed data. By observing these light curves, they can begin to hypothesize how these objects work. They need a sophisticated way to model these objects, as well as a way to find a light curve model with similar parameters to observed systems.

[1] https://en.wikipedia.org/wiki/Binary_asteroid

1.2 Problem

We have been tasked by Dr. Will Grundy and Dr. Audrey Thirouin at Lowell Observatory to develop several new modules that would improve or add functionality to their solutions infrastructure. These modules would work towards solving the following issues.

- The implementation used to produce simulated light curves now requires the user to manually enter over fifty parameters into an IDL command line. This is an arduous and time-consuming process due to IDL running slow for our clients.
- The solution currently being used is limited by the efficiency at which it can render shapes. Thus, a means to improve the efficiency of the light curves produced by the forward model is necessary.
- Currently, the implemented solution has the ability to produce light curves that are similar to those found in observed data. However, our clients need these produced light curves to be more accurate than they are now. Therefore, a way to filter out data that is reducing the accuracy of produced light curves is needed.

1.3 Solution

We have decided to implement a set of modules that build upon the existing solution. The modules we plan to implement will focus on solving the aforementioned problems.

- We will develop a GUI that will support user input for the parameters used by the forward model. This GUI will also display rendered movies from images produced by the model.
- A new shape subclass will be implemented to allow for the modeling of triaxial ellipsoids.
- Lastly, a Hamiltonian MCMC wrapper for the forward model will be added to provide parameter estimates for light curves similar to those found in observed data.

2. Technical Challenges

2.1 Triaxial Ellipsoids versus GPU

Implementing a triaxial ellipsoid shape to the list of renderable objects, as well as optimizing the GPU so that all objects can render faster are two of our main goals for this project. We believe that we would have enough time to implement one of the above goals, but not the other. Since there are many facets to both implementations, we will break them down by reviewing the features necessary for their implementation.

2.2 Hamiltonian Monte Carlo (HMC) Algorithm

To improve the efficiency of our clients' or users' workflow, we need to implement a Hamiltonian MCMC algorithm using a pre-existing API that includes HMC specific functions. The most important thing we will have to consider is how the API is implemented. This way, we can use the algorithm effectively so that it can produce "best fitting" parameters that can be used to create light curves similar to the observed light curves. We will examine the workflow necessary to use the algorithm, along with how effectively an HMC algorithm can be integrated using an API.

2.3 Hamiltonian Monte Carlo (HMC) API Command Line Interface (CLI)

Multiple considerations need to be made when choosing the right package or framework that will serve as the interface for the HMC API. The most important of these considerations is whether our clients require a simple or advanced interface for use with the HMC API.

2.4 GUI Framework

The implementation of a new GUI framework needs to be designed in a way that allows efficient parameter manipulation and responsive displays of data. There are several different programming languages that can be used to achieve these goals, which will be analyzed in the sections below.

2.5 Video Generator

Our project needs to have the capability to take a series of rendered images of a modeled system and compile them into movie viewable through the GUI. The movie should be made after all the images for a given system have been generated.

3. Technology Analysis

3.1 Triaxial Ellipsoids versus GPU

3.1.1 Introduction to Triaxial Ellipsoids and GPU

Two important features that our clients are interested in implementing are the modeling of triaxial ellipsoids and the addition of GPU parallelization to the ray tracer of the forward model. These features would improve existing components of the program. Rendering triaxial ellipsoids would require further development of the sphere object in the program, while implementing GPU parallelization would require modifications to the ray tracer of the software.

Our main concern is the feasibility of being able to implement both of these within the given timeframe of the project. Ultimately, we believe that we only have time to work towards one of these goals. The sections hereafter breakdown and analyze the feasibility of implementing these features, and which one we feel would benefit our clients the most. After reviewing the practicality of both components, we will discuss which one we think should be considered for the program.

3.1.2 Factors

Our main factors when considering the implementation of triaxial ellipsoids and GPU APIs are compatibility, processing speed, and familiarity. The factors are presented in order of most importance to least importance.

Compatibility

Compatibility is important because we want to focus our time working with technology that is already compatible with the program we have. This way we can spend more time working on the project, and less time worrying about how the given improvement will integrate with the program. Compatibility will be measured by seeing whether integration is possible with the current program, and whether doing so would be difficult.

Processing Speed

Processing speed plays a large role in our choice, as our clients wish to have the program run as fast as possible. For the model, our clients will be switching between speed and accuracy as needed. Since our implementation of triaxial ellipsoids can give our clients a slightly more advanced model than a sphere, it is vital to maintain an optimal render time. Because our clients want the model to render as fast as possible, optimizing GPU speed is integral as well. The speed analysis of the triaxial ellipsoid implementation and the GPU APIs will be measured by comparing render speeds in separate technology groups. For example, triaxial ellipsoids' render speed will not be compared to a GPU API, but rather another render shape option such as a sphere.

Familiarity

Familiarity will play a deciding factor in which implementation we choose as well. The team feels that it is better to choose an approach that we are more familiar with so that less time is spent learning. We wish to try to dedicate a majority of our time to implementing these improvements to their fullest. To measure this factor, each candidate will have a table with how familiar the team members are with said candidate. The team will rate their familiarity on a scale from 0-5, and an average will be given at the bottom of the table.

3.1.3 Introduction to GPU

In the current implementation of the project, the software is parallelized for the CPU at two levels. The first is at the observation level and the ray level (where ray tracing occurs). While CPU parallelization does improve the performance of the software at the observation level, it is far less efficient at the ray level. This is where the application for GPU parallelization is valuable, as it would improve the performance of the ray tracer. Consequently, this would greatly cut down on the individual times needed to render and return the results of each observation that would otherwise add up very quickly across thousands of simulations using different sets of parameters.

There are two options we will explore while looking into the GPU problem. CUDA and OpenCL are both used for parallel processing, but the two platforms have some distinct performance differences. By comparing compatibility, processing speed, and familiarity, we will decipher whether optimizing the GPU is feasible within our time constraint.

3.1.4 CUDA

Introduction

CUDA is a NVIDIA owned and operated parallel processing framework. Since CUDA is propriety, it has full support of NVIDIA Corporation and therefore is well optimized. The CUDA framework is only able to work with NVIDIA graphics cards, which is limiting considering what GPU our client may have.

Analysis Methods

In order to accurately capture the metrics used to compare our candidates, CUDA was analyzed using the following methods for each factor:

- **Compatibility**: To judge if CUDA is compatible with our clients' GPU.
- **Processing Speed**: To gain an accurate representation of how well each CUDA performs, we refer to a study demonstrating the processing speed of CUDA when processing thousands of photons.
- **Familiarity**: For a smooth workflow, our project would benefit from our team's current framework knowledge. Our team will rank (0-5) our familiarity with CUDA.

Analysis Results

Compatibility

CUDA is proprietary and therefore is restrained to NVIDIA GPUs. NVIDIA offers a full range of support. In the case that our client has anything other than an NVIDIA GPU, CUDA would not be an applicable framework to use.

Processing Speed

A study involving Monte Carlo photon transport simulation showcased CUDA performance in a relatable way to our project. The three-dimensional simulation ran up to 40,000 photons per millisecond to test CUDA on NVIDIA cards. On the GTX 1080Ti, CUDA rendered about 32,000 photons per millisecond. Other, lower performance cards such as the GTX 1080, GTX 980 Ti, TITAN X, GTX 590, CUDA was still able to render high levels of photons per second. However, the GTX 1050Ti card gave an average photon render time[2].

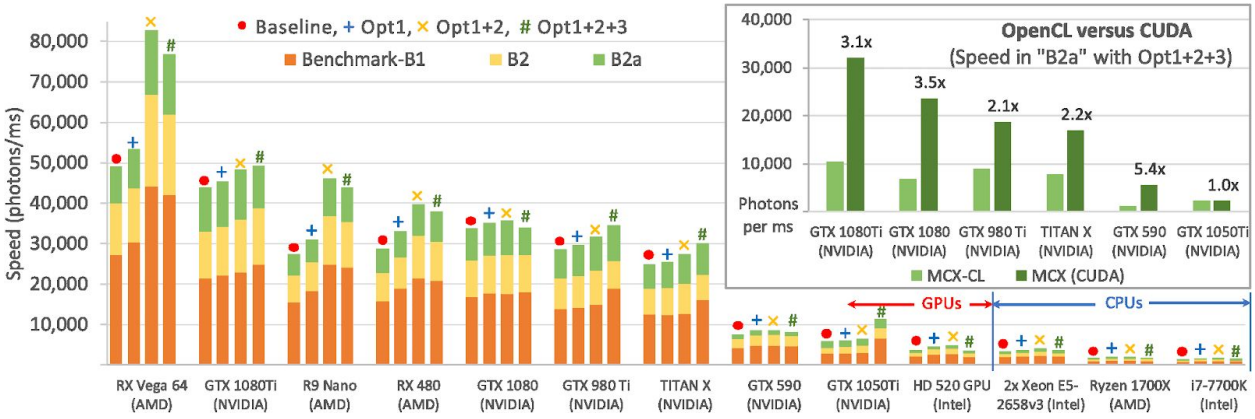

Figure 1: The top right graph shows the photons rendered per millisecond[2]

Familiarity

Our group has no experience working directly with CUDA. We have extensively researched its API however and discovered that it is a C-like framework. Our average familiarity levels out to 0.4 on a zero to five scale with five people.

[2] https://www.spiedigitallibrary.org/journals/journal-of-biomedical-optics/volume-23/issue-01/010504/Scalable-and-massively-parallel-Monte-Carlo-photon-transport-simulations-for/10.1117/1.JBO.23.1.010504.full?SSO=1

Familiarity with CUDA	
Team member	Familiarity (0-5)
Batai Finley	1
Bradley Kukuk	0
Matthew Amato-Yarbrough	0
Jessica Smith	1
John Jacobelli	0
Average:	0.4

Table 1: Our team's current familiarity with GPU APIs

3.1.5 OpenCL

Introduction

OpenCL is a framework that facilitates parallel processing across heterogeneous systems, which are systems that use more than one kind of processor. Furthermore, OpenCL is not restricted to one vendors brand of processors which allows for it to be compatible with a wider range of CPUs and GPUs. Additionally, OpenCL is an open source platform facilitating community support and optimization across a variety of different processors.

Analysis Methods

In order to accurately capture the metrics used to compare our candidates, GPU APIs were analyzed using the following methods for each factor:

- **Compatibility**: To judge if OpenCL is compatible with our clients' GPU.
- **Processing Speed**: To gain an accurate representation of how well each OpenCL performs, we refer to a study demonstrating the processing speed of OpenCL when processing thousands of photons.
- **Familiarity**: For a smooth workflow, our project would benefit from our team's current framework knowledge. Our team will rank (0-5) our familiarity with OpenCL.

Analysis Results

Compatibility

As noted previously, OpenCL is an open source framework and as a result is supported on a multitude of GPUs and CPUs. Therefore, implementing OpenCL into our project will not be an issue as our clients brand of hardware will be supported.

Processing Speed

In the same study examining Monte Carlo photon transport simulations, OpenCL's performance was measured as well. Under OpenCL, the simulation ran up to 10,000 photons per millisecond when testing on the GTX 1080 Ti. The GTX 1080, 980 Ti, Titan X and 1050 Ti performed worse with the 980 Ti coming closest to the 1080 Ti at around 9000 photons per millisecond[2].

Familiarity

Our team has no experience with OpenCL aside from what we have gathered in our research on the framework. However, we have come to learn that OpenCL can be called from programs written in C and C++.

Familiarity with OpenCL	
Team member	Familiarity (0-5)
Batai Finley	1
Bradley Kukuk	0
Matthew Amato-Yarbrough	0
Jessica Smith	1
John Jacobelli	0
Average:	0.4

Table 2: Our team's current familiarity with GPU APIs

3.1.6 Triaxial Ellipsoids

Introduction

Triaxial ellipsoids are spheres that have been deformed on 3 different axises, meaning they may look similar to a hamburger or potato. Our clients would like to have a feature that can model triaxial ellipsoids. This feature will allow them to attain a more accurate representation of different objects that exist in the Kuiper Belt.

We spoke with Brian Donnelly, a member of the team working on the project last year, about the features created previously. Brian stated that spheres and faceted shapes had been implemented. Faceted objects are objects rendered using many different shapes in order to create the model. Brain also mentioned that spheres have a run time of 1-2 seconds while faceted objects run for 20-35 seconds. While providing a more realistic object, faceted rendering is much slower than rendering a single shape.

Brian added that his team had started working on implementing triaxial ellipsoid rendering, but did not finish adding it to their program. He touched on the issues that they had run into, which mainly involved the rotation of these objects. Brian talked about the problem being a minor one that would only take a few weeks to solve, but stated that his team had been on a time crunch and could not include the feature. Brain explained that triaxial ellipsoids can be modeled using a single shape similar to a sphere. The ability to model these triaxial ellipsoids would greatly improve the accuracy of modeling binary systems while retaining the run time of a sphere.

Analysis Methods

In order to accurately capture the metrics used to compare our candidates, triaxial ellipsoids were analyzed using the following methods for each factor:

- **Compatibility**: Evaluate whether the methods used to implement triaxial ellipsoids are compatible with the current API.
- **Processing Speed**: Talk to Brian Donnelly, an expert on the program, and gain his opinion on how fast triaxial ellipsoids can be run in comparison to currently implemented shapes.
- **Familiarity**: Our team will rank (0-5) how familiar they are with the codebase, as the triaxial ellipsoid will be made using an extension of a currently implemented module.

Analysis Results

Compatibility

Compatibility should not be an issue for implementing triaxial ellipsoids since the module for using triaxial ellipsoids would be an extension of the sphere module. This module was written in C++, meaning that the triaxial ellipsoid would also be implemented in C++. As the triaxial ellipsoid module would not be from an outside library or source, compatibility would not be a concern. The module that will be written for the triaxial ellipsoids will already be in C++ and therefore be naturally compatible.

Processing Speed

Speed efficiency is important for our clients because they will be generating thousands of these models in order to try and match an observed light curve. Waiting twenty times longer for an object to run would inhibit the efficiency at which our clients could check light curves. This is important to our clients because they would like to be able to choose between a faster run time or a more accurate model. The triaxial ellipsoid object, like spheres, would assist in keeping run times low while faceted objects would be used for more accurate models. The impact of the render speeds over time can be seen better in Table 3 and Figure 2 below.

Object Render Speed	
Shape	Time (seconds)
Sphere	~1-2
Triaxial Ellipsoids	~1-2 (projected)
Faceted Shapes	~20-35

Table 3: Render speed of objects using different shapes

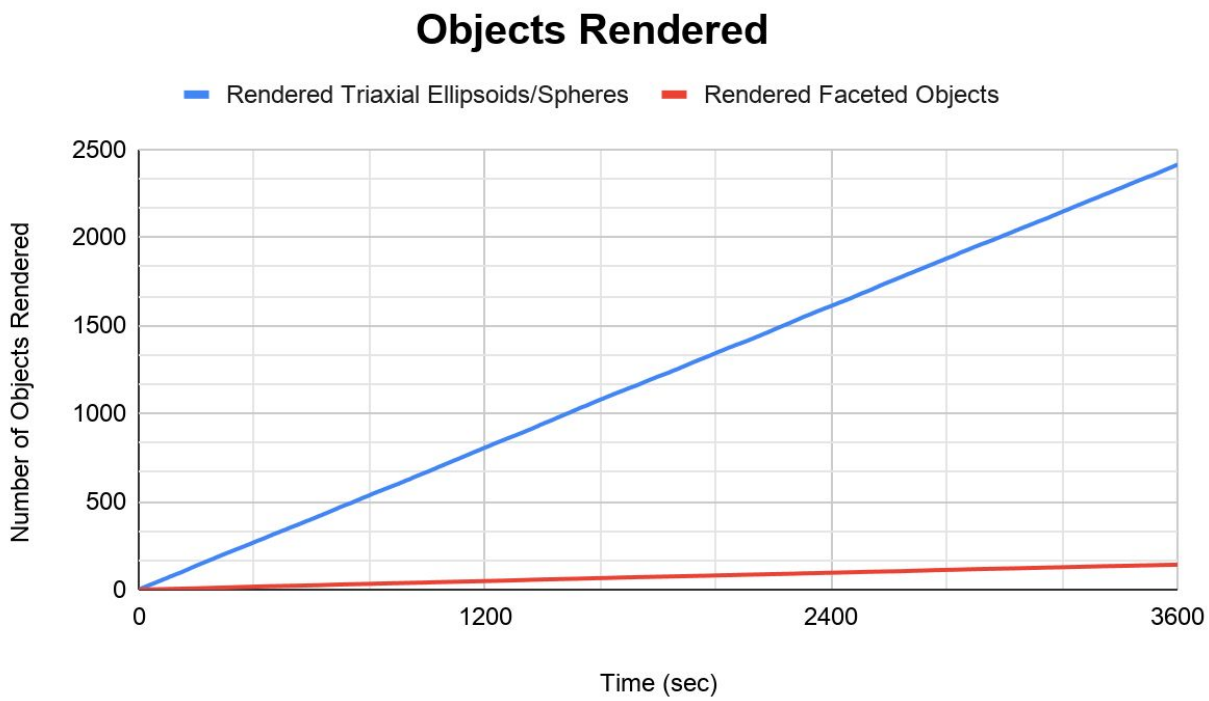

Figure 2: Number of objects rendered for an hour using triaxial ellipsoids vs faceted objects

Familiarity

Our team has begun to review the code and started to understand the inner workings of the program. We've also gone over detailed explanations of how the modules interact with each other in order to get a better understanding of the API we are working with. While we are not experts, we are somewhat familiar with the codebase we will be working from. The team's familiarity with the program is seen in Table 4 below.

Familiarity with Program	
Team member	Familiarity (0-5)
Batai Finley	2
Bradley Kukuk	3
Matthew Amato-Yarbrough	2
Jessica Smith	2
John Jacobelli	2
Average:	2.2

Table 4: Our team's current familiarity with our clients' program

3.1.7 Summary

Overall, we believe that the triaxial ellipsoid feature would be much more feasible. Brian is currently researching CUDA at the graduate level and advised that it would take a full-time person who is familiar with GPU APIs around 2 to 3 years to implement the problem we are facing. With this advice and with our lack of GPU API knowledge, GPU optimization does not have a feasible implementation time frame.

Triaxial ellipsoids benefit our client and allow them to have a more accurate model to simulate without costing them render time. This shape can also be implemented within a reasonable time frame. Since the module for the triaxial ellipsoid object would be an extension of the sphere module, it would also be compatible with the current API.

The table below overviews our candidates and compares them with our analysis methods. Due to these factors outlined in Table 5, we believe that adding triaxial ellipsoids into the existing API would be feasible and extremely useful for our clients/users.

Candidate Feasibility			
Candidate	Compatibility	Processing Speed	Familiarity (Average)
Triaxial Ellipsoids	High	High	Medium (2.2)
CUDA	Low	High	Low (0.4)
OpenCL	High	Low	Low (0.4)

Table 5: Feasibility of each candidate and how they compare to each other

3.2 Hamiltonian Monte Carlo (HMC) API

3.2.1 Introduction to Hamiltonian Monte Carlo API

In an effort to increase the efficiency of our clients' or users' workflow, we aim to incorporate a Markov Chain Monte Carlo (MCMC) algorithm into our project. More specifically, we will be working with a variant of MCMC algorithms called the Hamiltonian Monte Carlo (HMC) algorithm. With the inclusion of this algorithm, our clients/users will be able to obtain likely parameters for observed binary systems that can then be used in conjunction with the forward model.

Below is a workflow illustrating how the HMC algorithm will be used in order to create light curves with increased accuracy.

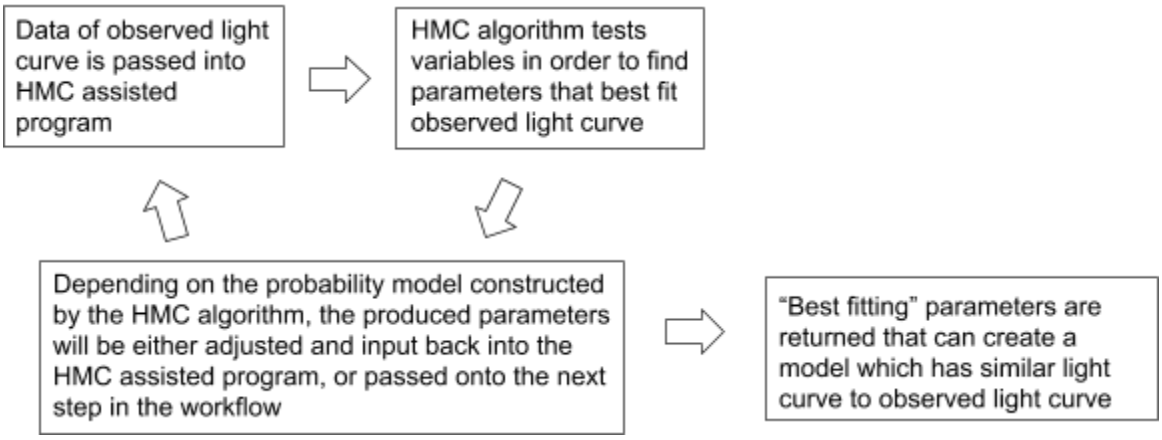

Figure 3: Workflow involved with using the HMC algorithm

In order to satisfy this workflow, we will be using pre-existing APIs that include HMC specific functions in order to create probabilistic models. These APIs include Stan, an open-source probabilistic programming API written in C++, and Pymc3, an open-source probabilistic programming API written in Python. Both APIs support a wide range of probability modeling, including HMC. Furthermore, both APIs work for continuous parameters while also offering methods of masking continuous parameters as categorical in order to handle these types of parameters within the API. For each API, an analysis will be done exploring key factors that will determine each API's feasibility within this project.

3.2.2 Factors

In order to be considered feasible, these APIs will need to satisfy four factors: efficiency, familiarity, level of support offered, and compatibility. The factors are presented in order of most importance to least importance.

Efficiency

The efficiency of the API is crucial in determining its feasibility; the chosen API must be able to maintain a balance between speed and accuracy. When measuring the efficiency of an algorithm based API, the size of the problem being solved is an important factor to consider. The problem size encapsulates factors such as the type of data being input, the number of parameters incorporated in the problem, how the data is related, etc. However, constant factors such as startup times and the speed in which computation is done are also important factors to consider. When analyzing each candidate, an analysis will be done considering the API's method of computation. Specifically, this factor will consider the optimization library used by the API.

Familiarity

In order for our team to be able to implement the chosen API, we must be familiar with the programming language that the API is written in. This will affect the difficulty and required time associated with implementing the API into our project. For example, if our team is only somewhat familiar with the language that the API uses, it will be more difficult and require more time to write code using the libraries within the API. Conversely, if our team has a strong familiarity with the language, it will be less difficult and will take less time overall. The familiarity that our team has with the language that each API uses will be determined by a rating of 0-5 from each teammate in regard to their familiarity with the API's programming language.

Level of Support Offered

The amount of resources that can be found on each API's website is important in determining the difficulty associated with using the API. These resources can be in the format of tutorials, documentation, community support, examples, books and videos. These resources will give us the references needed in order to utilize the APIs, and more specifically, the HMC aspect of the APIs. The level of support offered will be determined by examining whether these resources are readily available on the API's website.

While additional information can be found on websites other than the API's website (such as YouTube), this information will not be considered when exploring this factor. This is due to the differing levels of accuracy associated with information found outside the API's website; differing API versions, incorrect tutorials, etc. While this additional information could be considered viable, it varies depending on the source of the information.

Compatibility

The API that we choose to work with for this section of the project will need to call the forward model currently implemented in the codebase of the project. As such, it is important that the programming languages of each API be compatible with the languages currently used in the codebase of the project: C, C++, and IDL. Moreover, the language for the projected GUI for the forward model will be written Python, and as such, it will be important to consider this language as well.

By ensuring compatibility with these languages, we will ensure that there will not be any compatibility errors associated with using the API in conjunction with the forward model. The level of compatibility between the API and the languages used within the current and planned iteration of the project will be examined by determining whether the language that the API is written in is compatible with the aforementioned languages.

Additionally, each API will need to be able to run on all operating systems (OSs): Linux, Mac, and Windows. This will ensure that the API can be utilized by our clients/users, regardless of the OS they are running the API on. Compatibility with these major platforms will be explored by determining whether the API can or cannot be used with a specific OS.

3.2.3 Stan[3]

Introduction

Stan is a modern, free and open-source API written in C++ and is used for modeling and high-performance statistical computation. It is used by thousands of users for statistical modeling, data analysis, and prediction within various fields. It can be used with most data analysis languages such as R, Python, and Matlab. Stan was initially released in August 2012, and the current stable version of Stan is 2.21.0.

Analysis Methods

In order to accurately capture the metrics used to compare our candidates, Stan was analyzed using the following methods for each factor:

- **Efficiency**: Examine the efficiency of Stan in regard to the optimization library that it uses to construct models.
- **Familiarity**: Explore the familiarity that our team has with C++, since Stan is written in this programming language.
- **Level of Support Offered:** Identify the available support on Stan's website that can be used to assist in using the API.
- **Compatibility**: Determine whether the programming language that Stan is written in is compatible with the pre-existing and projected codebase.

[3] https://mc-stan.org/

Analysis Results

Efficiency

Stan uses its own optimization library in order to solve mathematical expressions. The optimization library can be configured to use either the computer's GPU or CPU. By utilizing the computer's GPU over its CPU, the mathematical calculations performed by Stan allows for superior processing power. This is due to the multitude of GPU cores that can be used to handle multiple functions at the same time, with less of a cost to the overall speed of computation.

Familiarity

Stan is based on the programming language C++. While our team is familiar with C++, our knowledge on the language is limited, as reflected by the metrics in Table 6 below. This means that we can read and understand C++ source code, but will need to spend a large amount of time referencing online C++ resources in order to determine the correct syntax that would need to be used in order to construct a stable C++ program.

Familiarity with C++	
Team member	Familiarity (0-5)
Batai Finley	1
Bradley Kukuk	1
Matthew Amato-Yarbrough	1
Jessica Smith	1
John Jacobelli	1
Average:	1

Table 6: Our team's current familiarity with C++

Level of Support Offered

As with most well documented and actively maintained APIs, an extensive amount of support can be found on Stan's website, as noted below in Table 7. Support in regard to how to get started using Stan, the math behind the statistical models that can be created, and how to implement it into various interfaces are just a few resources that are readily available.

Level of support offered					
API	Tutorials	Documentation	Community Support	Examples	Books + Videos
Stan	Yes	Yes	Yes	Yes	Yes

Table 7: Level of support offered on Stan's website

Compatibility

Both the forward model of the project and Stan API are written in the same programming language, C++, and as such are natively compatible. However, in order for the API to be compatible with IDL and Python, a wrapper will need to be used. Compatibility is demonstrated in Table 8 below. Entries with an asterisk note that a wrapper will need to be used.

Compatibility with C, C++, IDL, and Python				
API	C	C++	IDL	Python
Stan	Yes	Yes	Yes*	Yes*

Table 8: Compatibility Stan has with languages used in codebase

3.2.4 Pymc3[4]

Introduction

Pymc3 is a free and open-source API written in Python and is used for Bayesian statistical modeling and probabilistic machine learning. Due to its flexibility and extensibility, it is an applicable solution for a large variety of scientific fields such as astronomy, chemistry and ecology. Due to its widespread use, Pymc3 has amassed a large number of users. Pymc3 was initially released in May 2013, and the current stable version of Pymc3 is 3.7.

Analysis Methods

In order to accurately capture the metrics used to compare our candidates, Pymc3 was analyzed using the following methods for each factor:

- **Efficiency**: Examine the efficiency of Pymc3 in regard to the optimization library that it uses to construct models.
- **Familiarity**: Explore the familiarity that our team has with Python, since Pycm3 is written in this programming language.
- **Level of Support Offered:** Identify the available support on Pycm3's website that can be used to assist in using the API.
- **Compatibility**: Determine whether the programming language that Stan is written in is compatible with the pre-existing and projected codebase.

Results

Efficiency

Pymc3 is built on top of a powerful optimization library called Theano[5], which allows for the efficient computation of complex mathematical expressions. Theano's speed is derived from 2 important factors:

1. Use of the computer's GPU: data-intensive mathematical expressions are computed using the computer's GPU rather than its CPU.
2. Dynamic C code generation: evaluates mathematical expressions using dynamic C code.

[4] http://deeplearning.net/software/theano/
[5] https://docs.pymc.io/

The benefits of utilizing the GPU for complex mathematical calculations are explored in the efficiency results of the analysis results in Section 3.2.3. In short, ability to use the GPU over the CPU allows for greater computation speed. Furthermore, the use of dynamic C code within Theano allows for higher performance standards than its static C code counterpart.

Familiarity

Pymc3 is based on the programming language C++. On average, our team is between familiar and proficient with Python, as reflected by the metrics in Table 9 below. This means that we can read, understand and implement Python code without needing to refer to online Python resources. However, we will need to refer to online resources in order to implement complex functions within a Python program.

Familiarity with Python	
Team member	Familiarity (0-5)
Batai Finley	2
Bradley Kukuk	4
Matthew Amato-Yarbrough	3
Jessica Smith	3
John Jacobelli	3
Average:	3

Table 9: Our team's current familiarity with Python

Level of Support Offered

An ample amount of support can be found on Pymc3's website, as noted below in Table 10. An example of support on the website includes beginner tutorials, documentation on implementing HMC model computation, and videos covering examples of Pymc3 programs. Additionally, links to online books regarding Bayseian data analysis can be found on the website, free of charge.

Level of support offered					
API	Tutorials	Documentation	Community Support	Examples	Books + Videos
Pymc3	Yes	Yes	Yes	Yes	Yes

Table 10: Level of support offered on Pymc3's website

Compatibility

Both the projected GUI for the forward model of the project and Pymc3 are written in Python, and as such are natively compatible. However, in order for the API to be compatible with C, C++ and IDL, a wrapper will need to be used. Compatibility is demonstrated in Table 11 below. Entries with an asterisk note that a wrapper will need to be used.

Compatibility with C, C++, IDL, and Python				
API	C	C++	IDL	Python
Pymc3	Yes*	Yes*	Yes*	Yes

Table 11: Compatibility Pymc3 has with languages used in codebase

3.2.5 Summary

After performing an analysis of both the Stan and Pymc3 APIs, we concluded the API that we are going to use to implement the HMC algorithm within the project will be Pymc3. The efficiency, compatibility and level of support offered for Stan and Pymc3 are similar, as shown in Table 12 below. Both APIs use highly optimized libraries in order to solve mathematical expressions, and they are both compatible with the current programming languages used by the existing codebase. Moreover, both APIs provide an ample level of support on their websites. As such, both are feasible options in terms of efficiency, compatibility and level of support offered.

However, the familiarity that our team has with the programming language used by Pymc3 makes it the more feasible candidate for this project. This metric is compiled in Table 12 below. The use of Python as Pymc3's programming language means that it will be less difficult and more time efficient to be implemented into the project by our team.

Candidate Feasibility				
Candidate	Efficiency	Familiarity (Average)	Level of Support Offered	Compatibility
Stan	High	Low (1.0)	High	High
Pymc3	High	Medium (3.0)	High	High

Table 12: Feasibility of each candidate and how they compare to each other

3.3 Hamiltonian Monte Carlo (HMC) API Command Line Interface (CLI)

3.3.1 Introduction to Hamiltonian Monte Carlo API Command Line Interface

In order to utilize the HMC API that we to plan implement into our project, a Command Line Interface (CLI) package or framework must be considered. This CLI package or framework will augment the HMC API and function as a way to input parameters into the API using the command line. This will allow our clients/users a way to interact with the HMC API in order to produce the data they need, while also limiting the time needed in order to input data into the API.

In order to best fit our clients' needs, this section of the document will provide two solution candidates in the form of a package or a framework. In short, a package will provide the support needed to make a simple but effective CLI for inputting data. Conversely, a framework will provide the support needed to make an advanced CLI at the cost of additional compilation time and overall time to implement.

The CLI package and framework that meet these requirements and will be explored in this section of the document are Click and Cement. Both Click and Cement are Python based and they have been specifically created to help programmers make CLIs. Each can be imported into Python projects and ran within the code of the HMC API. An analysis will be conducted on both candidates to ensure that the correct one is chosen for development within this project.

3.3.2 Factors

For a CLI package or framework to be considered, it must satisfy three important factors: usability, level of support offered, and longevity. The factors are presented in order of most importance to least importance.

Usability

Since the CLI will act as the interface to the HMC API, it is important to consider the ease of use that our clients or users will experience when interacting with the interface produced. Although CLIs are limited in terms of possible use, there are differing levels of customizability that can be achieved for them. As such, we will need to examine the design limitations posed by each candidate in order to determine the level of usability that can be achieved by each.

Level of Support Offered

The amount of documentation available on each candidate's website will need to be considered, as it is a factor in determining the difficulty that comes with building the CLI. These resources can be in the format of documentation, examples, and community support. The level of support offered will be determined by examining whether these resources are readily available on the candidate's website. Moreover, similar to the level of support factor in Section 3.2.1, information found outside the candidate's website will not be considered when exploring this factor.

Longevity

Longevity of a program is important because of future integration and stability. To ensure that the CLI implemented for this section of the project remains viable in future iterations, it is important to examine the length of time that the candidate has been available and how actively maintained its codebase is.

3.3.3 Click[6]

Introduction

Click is a Python package for creating simple but effective command line interfaces in a composable way with as little code as necessary. It's highly configurable but comes with sensible defaults out of the box. It is a commonly used CLI package due to its simplicity and ease of implementing. Click was initially released in May 2012, and the current stable version of Click is 7.1.

Analysis Methods

In order to accurately capture the metrics used to compare our candidates, Click was analyzed using the following methods for each factor:

- **Usability**: Examine the design limitations of Click.
- **Level of Support Offered:** Identify the available support on Click's website that can be used to assist in building the CLI.
- **Longevity**: Identify the amount of time that Click has been available as a package and how actively maintained its codebase is.

Results

Usability

The main selling point of Click is its simplicity in implementation, which makes it exceptionally easy CLI to incorporate into any project. Fortunately, this simplicity does not come at a great cost in terms of how much usability customization it offers to its users. It offers functionalities such as the validation of values, integration with the terminal (colors, progress bar, etc.), and result callbacks. Overall, despite its limitations, Click offers solutions for problems that require a CLI.

Level of Support Offered

The amount of support that can be found on Click's website is limited to its documentation and the examples within its documentation. However, despite being limited in the varying options of support, as shown below in Table 13, the level of coverage provided by the documentation is extensive. Information regarding every aspect of Click's functionality is included, as well as a section on how to get started.

[6] https://click.palletsprojects.com/en/7.x/

Level of support offered			
CLI	Documentation	Examples	Community Support
Click	Yes	Yes	No

Table 13: Level of support offered on CLI's website

Longevity

Since it's first release on May 24th 2012, Click has been actively maintained with a steady flow of commits and version updates, as seen in Figure 4 below. This figure was created using the information regarding Click's repository, and was gathered using an online Github comparison tool[7]. Furthermore, the most recent commit was 9 days ago shows that the project is still actively maintained.

Figure 4: General health of the Click project repository

[7] https://bayne.github.io/github-compare/#!/

3.3.4 Cement[8]

Introduction

Cement is a standard and feature-full framework for both simple and complex command line applications. This flexibility is demonstrated in the large number of functionalities associated with the framework. Moreover, due to the flexibility of Cement, it provides a wide range of use cases and as such serves a wide variety of users. Click was initially released in Dec. 2009, and the current stable version of Click is 3.0.5.

Analysis Methods

In order to accurately capture the metrics used to compare our candidates, Cement was analyzed using the following methods for each factor:

- **Usability**: Examine the design limitations of Cement.
- **Level of Support Offered:** Identify the available support on Cement's website that can be used to assist in building the CLI.
- **Longevity**: Identify the amount of time that Cement has been available as a framework and how actively maintained its codebase is.

Results

Usability

The goal of Cement is to provide the functionalities needed to implement a CLI ranging from simple to complicated. As such, it offers a variety of functionalities such as:

- Temple handler for rendering content/file templates
- Log handler for logging output to console or file
- Cache handler for for improved performance through caching

In total, Cement offers 10 different handlers for processing and output various types of data. Overall, Cement provides all the basic functionalities one would expect to find a CLI package, along with additional advanced functionalities.

[8] https://docs.builtoncement.com/

Level of Support Offered

Cement's website has support, but it is limited to documentation and the examples within its documentation. Though the support may seem to be limited in quantity, as seen in Table 14 below, the quality of the documentation is extensive. Documentation about Cement encompasses all of Cement's functionality, and includes sections for beginners so that they can ease into the framework.

Level of support offered			
CLI	Documentation	Examples	Community Support
Click	Yes	Yes	No

Table 14: Level of support offered on Cement's website

Longevity

Since it's first release on Dec. 4th 2009, Cement has a steady flow of commits, but lacks the in regard to the number of version updates, contributors and pull requests, as seen in Figure 5 below. Furthermore, the most recent commit was 4 months ago, which shows that the project is still maintained, but not at an exceptionally high rate.

Figure 5: General health of the Cement project repository

3.3.5 Summary

After performing an analysis on both the Click package and the Cement framework, we concluded the alternative that we are going to use to act as the interface with the HMC API will be Click. The level of support offered for Click and Cement is similar, as shown in Table 15 below. Both alternatives provide extensive documentation in regard to the implementation of the various functionalities that each alternative has to offer.

However, each alternative differs in usability and longevity. Cement offers a larger number of functionalities than Click in the way of advanced customizability that cannot be achieved with Click. Conversely, Click offers a much higher level of longevity than Click, ensuring that it can be used in future iterations of the project and will remain stable. While Cement offers a higher level of usability, the additional functionalities that it provides are not required in this project.

Candidate Feasibility			
Candidate	Usability	Level of support offered	Longevity
Click	Medium	Medium	High
Cement	High	Medium	Low

Table 15: Feasibility of each candidate and how they compare to each other

Additionally, a possible stretch goal for this project would be to implement support for the HMC API directly into the GUI of the project. However, further research would need to be done before a design decision was concluded on this subject.

3.4 Graphical User Interface

3.4.1 Introduction to Graphical User Interface

Currently our clients' software does not have a GUI, and they must use a command line to run the software. Our clients asked that they would like our team to build them a GUI that uses their existing code base to improve efficiency. This existing code base is built in C/C++, so we have decided to use a Python framework to build our GUI.

3.4.2 Factors

For the implementation for the Graphical User Interface for the Forward Model there are three main factors to consider, the first being compatible with C/C++ due to the Forward Model being built with these programming languages. The second factor that we must consider is the ability to use this software anywhere at anytime, with or without an internet connection. The last factor that we must consider is our familiarity with the possible framework for the Graphical User Interface.

3.4.3 Flask

Introduction

Flask is a Python microframework that is used to create web based software as well as hybrid applications that run on both the internet and locally on a device. It is often used to create websites due to its lack of dependency on other libraries.

Analysis Methods

In order to accurately capture the metrics used to compare our candidates for the Graphical User Interface, we analyzed each factor using the following methods:

- **Compatibility**: Are the methods used to implement a graphical user interface compatible with the current code base?
- **Usability**: Is this framework usable when there is no internet connection?
- **Familiarity**: Our team will rank (0-5) how familiar they are with the framework.

Analysis Results

Compatibility

Flask is compatible with C/C++ with the use of Cython and will be able to call the functions from the forward model.

Usability

Due to Flask being web-based, it will be able to be used to its full functionality when it has an internet connection. When used offline there will be problems that will not allow us to use the graphical user interface fully.

Familiarity

Our team has a good understanding of Python, and the table below lists our understanding of the Flask framework.

Familiarity with Flask	
Team member	Familiarity (0-5)
Batai Finley	1
Bradley Kukuk	5
Matthew Amato-Yarbrough	1
Jessica Smith	3
John Jacobelli	1
Average: 2.2	

Table 16: Familiarity with Flask for each team member

3.4.4 Django

Introduction

Django is a web-based framework that is used to create websites, and hybrid applications using Python. It has libraries that allow it to create and manage data storage, and is used primarily for data manipulation while using Python.

Analysis Methods

In order to accurately capture the metrics used to compare our candidates for the Graphical User Interface, we analyzed each factor using the following methods:

- **Compatibility**: Are the methods used to implement a graphical user interface compatible with the current code base?
- **Usability**: Is this framework usable when there is no internet connection?
- **Familiarity**: Our team will rank (0-5) how familiar they are with the framework.

Results

Compatibility

Django is compatible with C/C++ with the use of Cython and will be able to call the functions from the forward model.

Usability

Due to Django being web-based, it will be able to be used to its full functionality when it has an internet connection. When used offline there will be problems that will not allow us to use the graphical user interface fully.

Familiarity

Our team has a good understanding of Python, and the table below lists our understanding of the Django framework.

Familiarity with Django	
Team member	Familiarity (0-5)
Batai Finley	1
Bradley Kukuk	5
Matthew Amato-Yarbrough	1
Jessica Smith	2
John Jacobelli	1
Average: 2	

Table 17: Familiarity with Django for each team member

3.4.5 Kivy

Introduction

Kivy is a framework used to create mobile applications and other multitouch application software with a natural user interface. It is mostly used to create stationary software for desktops as well as hybrid applications.

Analysis Methods

In order to accurately capture the metrics used to compare our candidates for the Graphical User Interface, we analyzed each factor using the following methods:

- **Compatibility**: Are the methods used to implement a graphical user interface compatible with the current code base?
- **Usability**: Is this framework usable when there is no internet connection?
- **Familiarity**: Our team will rank (0-5) how familiar they are with the framework.

Results

Compatibility

Kivy is compatible with C/C++ with the use of Cython and will be able to call the functions from the forward model.

Usability

Kivy built software is able to be run anywhere and at any time. Its functionality will not suffer from running offline.

Familiarity

Our team has a good understanding of Python, and the table below lists our understanding of the Kivy framework.

Familiarity with Kivy	
Team member	Familiarity (0-5)
Batai Finley	1
Bradley Kukuk	5
Matthew Amato-Yarbrough	2
Jessica Smith	3
John Jacobelli	3
Average: 2.8	

Table 18: Familiarity with Kivy for each team member

3.4.6 Summary

All three candidates prove feasible in terms of usability, compatibility, and familiarity. Due to all candidates being Python based, implementation using any of these three frameworks will work. Due to Django and Flask being web based, we believe that Kivy is the best candidate due to its offline capabilities. This is beneficial for the client being able to use the software anywhere at any time, as well as Kivy being the framework that the team is most familiar with.

Candidate Feasibility			
Candidate	C/C++ Compatible	Usability Offline	Familiarity (Average)
Flask	Yes	No	2.2
Django	Yes	No	2.0
Kivy	Yes	Yes	2.8

Table 19: Feasibility of each candidate and how they compare to each other

3.5 Video Generator

3.5.1 Introduction to Video Generator

Our clients want to make the process of generating videos from their images more streamlined. Currently, the program will create images of the model, but does not produce a video. Our clients would like to generate a video of the images that the model would produce, which would be done when the program is run. This would be done using an image processing and management toolkit. We were interested in using a language such as Python to write the generator in, which would then be wrapped to work with the API.

The main issue for our client is the extra time it takes to convert images rendered by the program into a video. Our clients wish to expedite this process by having the program create a video of the images when it is run. This would replace the need to compile the images into a video externally. The two toolkits that we explore below are FFMPEG and OpenCV.

3.5.2 Factors

The main factors we examined for the implementation of the video generator were code efficiency, accessible and extensive documentation, and whether the tool was GUI or command line based. The factors are presented in order of most importance to least importance.

Code Efficiency

Our team was heavily focused at how efficient and simple the code for a given tool was. Since video generation is fairly simple, we wanted to be able to utilize the toolkit without writing extensive code. We looked into how many lines long a typical image to video program was to measure this.

Documentation

Documentation that explains how to use the toolkit is a key factor for us as well. Being able to use resources that can guide us through the process of creating the video generator is extremely helpful, and can be used to assist us if we fail to understand an aspect of the toolkit. This will be examined by whether a toolkit has documentation such as whether a wiki exists for the kit.

User Interface

The way a user interacts with the toolkit and uses it was important to us. To gauge the impact of this, we looked at how the user interacts with the toolkit.

3.5.3 OpenCV

Introduction

OpenCV is an open source computer vision and machine learning software library. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. While primarily used for image recognition and learning algorithms, it can also be used to perform simpler tasks such as video generation.

Analysis Methods

In order to accurately gather the information that was used to compare our candidates, OpenCV was examined using the following methods for each factor:

- **Code Efficiency**: The average amount of coded lines needed to run a typical image to video program will be looked at.
- **Documentation**: The documentation of the toolkit will be explored to see whether it has a wiki, forum, documentation of open source coding, or tutorials.
- **User Interface:** We examined the interface style of a tool to see whether it was GUI based, command line based, or other.

Analysis Results

Code Efficiency

On average, using OpenCV to write a program that will convert images to a video seems to be about 15-20 lines of code.[9] [10] [11] The toolkit is also primarily used for other image management functions, and therefore may be more than is necessary.

---

[9] https://medium.com/@iKhushPatel/convert-video-to-images-images-to-video-using-opencv-python-db27a128a481
[10] https://theailearner.com/2018/10/15/creating-video-from-images-using-opencv-python/
[11] https://stackoverflow.com/questions/44947505/how-to-make-a-movie-out-of-images-in-python/44948030

Documentation
There is extensive documentation for this toolkit. This comprises everything that was stated in the Analysis Methods section, which includes a wiki, forum, documentation of open source coding, or tutorials[12]. This would be beneficial to the team if help was needed.

User Interface
OpenCV can be used within Python, which means that the toolkit is GUI accessible.

3.5.4 FFMPEG

Introduction
FFmpeg is a leading multimedia framework that is able to decode, encode, play, and manipulate almost every form of media that has been created. It supports the formats that are very old and often unused, as well as formats that are current. As it is a media toolkit, it fits perfectly into the role which we would use it for.

Analysis Methods
In order to gather the information that was used to compare our candidates, FFMPEG was considered with the following methods for each factor:

- **Code Efficiency**: The average amount of coded lines needed to run a typical image to video program will be viewed.
- **Documentation**: The documentation of the toolkit will be explored to see whether it has a wiki, forum, documentation of open source coding, or tutorials.
- **User Interface:** We examined the interface style of a tool to see whether it was GUI based, command line based, or other.

[12] https://opencv.org/

Analysis Results

Code Efficiency

On average, FFMPEG takes about 1-2 lines of parameters to convert a series of images into a video[7] [13] [14]. This toolkit is also fairly lightweight as it does not deal with much besides multimedia manipulation.

Documentation

There is extensive documentation for this toolkit. Everything that was stated in the Analysis Methods section is included besides tutorials, which means there is a wiki, forum, and documentation of open source coding[15]. This would be beneficial to the team if help was needed.

User Interface

FFMPEG is primarily a command line based framework. Though, it can be used within Python which means that it can potentially have a GUI.

3.5.5 Summary

While OpenCV can do more than FFMPEG, FFMPEG is more lightweight than OpenCV in terms of code. Less lines of code are required to perform the same task, and the simplicity makes FFMPEG more desirable. For the language, we considered Python because of the use of it throughout the rest of our project. There are examples of using FFMPEG within Python despite it being a command line based tool[7].

We believe that FFMPEG would be the best image processing and management toolkit due to its simplicity. As they can both be called from within a language (primarily Python, which will be our focus for the video generator language), FFMPEG seems to be the best option. Using FFMPEG over OpenCV will save time due to less lines of code needing to be written, and being less complex overall. FFMPEG performs the video conversion task as well as OpenCV, and will be able to satisfy our clients' needs. This is reflected upon in the table below.

[13] https://hamelot.io/visualization/using-ffmpeg-to-convert-a-set-of-images-into-a-video/
[14] https://stackoverflow.com/questions/24961127/how-to-create-a-video-from-images-with-ffmpeg
[15] https://ffmpeg.org/

Candidate Feasibility			
Tool	Code efficiency	Documentation	Interface type
FFMPEG	1-2 lines to convert images to video	Wiki, forum, open source coding, and tutorials	Command line, but can be used in Python
OpenCV	15-20 lines to convert images to video	Wiki, forum, and open source coding	GUI, can be implemented in various languages, such as C/C++/Python

Table 20: Feasibility of each candidate and how they compare to each other

4. Technology Integration

4.1.1 Current Integration Issues

One of the problems we face is that our implementation has no GUI for the forward model. This an issue because a parameter does not have a label, meaning that it is unknown which variable it references without prior knowledge. Without knowing which parameter references which variable, data could be easily inputted incorrectly. This incorrect input could skew data, which would be less likely to occur if parameters were entered into a UI. This is why we plan to integrate our own UI via the Python Framework Kivy.

Moreover, we plan to integrate a Hamiltonian MCMC algorithm which can narrow down input parameters to more accurately represent observed data. The integration challenge here is determining how the HMC API will interact with the forward model. Following the workflow in the introduction of Section 3.2.1, we can see that the output of the HMC algorithm will need to be input into the forward model in order to create a model with a light curve similar to that of the observed light curve. This is an issue with the current iteration of the project since it does not have a GUI. However, with the implementation of a UI in the form of Kivy, it will be notably easier to send the data from the HMC algorithm to the forward model.

4.1.2 Future Integration Issues

We must also consider future integration for our program. The method used to implement the GPU may change for the projected iteration. If our clients would like to streamline GPU processes, integrating a different parallel framework would depend significantly on the clients' GPUs.

In addition, future integration problems may occur with differences in the user's hardware. If the program is running on different configurations of hardware, we will have to check compute capability of devices, driver versions, and a number of other parameters that may cause problems. Because of this, we have decided to not include hardware optimization in our initial solution. If we find that we have additional time and the clients are interested, we may attempt to implement some of these types of optimizations near the end of the project.

With the problem of integration addressed, we can anticipate future problems with added technologies. We can now make a conclusion about the feasibility of the methods and solutions for improving our program that models binary systems.

5. Conclusion

After researching our design decisions, we have a thorough comprehension of our expectations. Since we have solidified our project's tasks, we believe that our expectations can be realistically completed within our time constraints. In the table below, we have outlined our challenges, solutions, and confidence level that the solution will work as expected.

Challenges and Solutions		
Technical Challenge	Solution	Confidence
Implementing Triaxial Ellipsoid/GPU	Triaxial Ellipsoids	High
HMC Algorithm API	Pymc3	Medium
HMC API CLI	Click	Medium
GUI Framework	Kivy	High
Video Generator	FFMPEG	High

Table 21: Our challenges and solution briefly outlined

We decided that attempting the GPU problem would not be feasible for the amount of time we have. Therefore, we will be moving forward with adding the triaxial ellipsoid shape to the simulator and have full confidence that we can implement it. In addition, we will integrate the Pymc3 API to handle the HMC algorithm and the Click package to handle the HMC API CLI. Since our clients expressed how a GUI would ease their intense computations, we look forward to providing them a user-friendly interface. We are also eager to provide the function of a video processor to the GUI so our clients can generate videos based on the images that the model creates.

Team Andromeda is prepared to work with our clients to create a full-bodied solution. To ensure a successful outcome, we recognize that we will need to dive deep into linear algebra, Bayesian statistics, and unfamiliar frameworks.

Space has piqued the curiosity of humans since the first twinkle of light. Though we learn more about it everyday, it is still full of mystery and the unknown. Space exploration is an exciting and crucial step for humans to find the origins of the universe. The fact that our project will make an impact in gathering vital information about space thrills us. We are honored to be entrusted with such an important project and are confident that we have the abilities to overcome the challenges we face.