3D interactive visualisation of the solar system using object-oriented programming

Garry Logan

UP935745

School of Computing

3rd Year

Engineering project

Abstract

This project aims to produce a realistic visualisation of our solar system. In order to differentiate from other systems it hopes to achieve high accuracy with regards to the planets' positions and properties while also presenting features for increased clarity in 3D space. The artefact developed will involve some degree of user interaction and could be utilised as a tool for educational purposes. Used by teachers or students, it can be a helpful tool for teaching and learning. The model, however, does not attempt to simulate real-world physics and will mainly prioritise realistic visual design. The following report documents the development process of this system. As such, it will cover the research, design and implementation choices made throughout the making of the artefact as well as its deviation from other similar projects.

Acknowledgements

I would like to express my gratitude to Dr Dalin Zhou for his commitment to assisting me with this project as my supervisor. His willingness to provide feedback even on the day of the deadline for this dissertation was greatly appreciated.

I would also like to thank my friends who gave me helpful advice and critique and motivated me to persist in the completion of this report.

Table of Contents

List of Figures

List of Tables

1. Project Introduction

1.1 Problem

It is clear to see that the topic of this report has been previously extensively explored by many developers. Thus, there exist a plethora of excellent solutions developed by experienced programmers and animators. However, a lot of such animations tend to simplify the model by utilising just the structure of the system to achieve purely aesthetic resemblance. The problem investigated within this report involves the design and implementation of a mostly up-to-scale solar system viewer. The solution will aim to achieve high accuracy with regards to the proportions of the dimensions and relative distances between the objects within the solar system.

1.2 Context and motivation

Having very limited experience, the author of this report primarily aims to understand the development process of animation and common coding tropes utilised in three-dimensional animation programming. As such, developing a model that supersedes existing systems in terms of functionality may be out of the scope of this report. However, several high precision systems that do exist on the web tend to either sacrifice aesthetic design or the accuracy of the model. Some such systems can be seen to over saturate the user with information on the UI screen – reducing the accessibility of the program.

The overall motivation behind this project is the creation of an intuitive 3D viewer that could be embedded into a web page for educational purposes. Preserving the accuracy of the mathematical dimensions while remaining simple is key.

1.3 Project Aim and Objectives

The aim of this project is to develop an interactive model of our solar system that will operate in a realistic fashion. The model will aim to be constructed in an environment such that real time user input will affect the ongoing animation. The visualisation implemented will hope to achieve as close approximation to realism as possible while retaining clarity and visibility of the animation as well as ease of operability.

The following are the broad objectives that would aid in the accomplishment of this aim:

1. Research of the similar existing systems and their specifications. Specifically, the research of the most popular sources that utilise a 3D solar system model.
2. Analysis of the significant design choices implemented in existing models. It is likely the case that sources will have varying objectives and requirements. Thus, there would be specific design choices / trade-offs that the development team would make to achieve these requirements.
3. Derivation of model requirements. Based on the aim of the problem explored in this report, and the review of the existing systems, a set of requirements must be defined.
4. Decision on the development tools. The technology used throughout is an important factor of the product design. It dictates the potential functionality that can be implemented as well as the limitations of the artefact.
5. The development of the working prototype.
6. Incremental improvement until basic requirements are met.
7. Texture rendering and polishing of the objects within the system.
8. The implementation of UI and extra features as dictated by the user and system requirements.
9. Identifying areas of improvement

1.4 Project Management

The project is planned and managed using a timeline gantt chart. Presented in appendix C, it shows the breakdown of time allocation for specific tasks during the development of the artefact and the writing of this report.

1.5 Constraints

There are a few constraints when it comes to the development of this project. Firstly, due to the limited time that the project must be completed in, the amount of new software or frameworks that the author chooses to implement must be restricted. The competent use of new software requires extensive learning and practice, thus some aspects of the development of the artefact must be constricted to what the author is already capable of performing. Consequently, the problem specifies the use of object-oriented programming and the software that utilises it.

Another constraint is the nature of the project and the potential use of new 3D graphics tools, libraries, and research into their documentation. This may lead to increased overall development time as learning could impede progress in each stage of development. Finally, the venture into new technologies may present issues with inefficient resource allocation and / or inaccurate predictions being made. As a result features could suffer underdevelopment or have flaws.

2. Literature Review

The contents of the following literature review section will mainly involve the research of the existing models that attempt to solve a similar problem. This will involve a discussion of design features of each system and the isolation of specific decisions made with reference to their requirements. Furthermore, the author will attempt to find a gap in the design that can be feasibly improved upon within the artefact being made during the writing of this paper. The sources chosen for this review were based on several factors including popularity, functionality, relevance to the topic and project size.

2.1 Existing Solutions / Systems

2.1.1 Theskylive.com

Theskylive.com is currently one of the most visited solar simulation websites as per similarweb.com (*Similarweb*, 2022) data. One of its main unique features is the live accurate location of the stars and comets present in our solar system as well as their current animated trajectories and future movement. This system is built upon three.js – a JavaScript based application programming interface (API) for 3D graphics on the web.

The main requirement of this model seems to be the accurate position and tracking of "most interesting"(*TheSkyLive - Your Guide to the Solar System and the Night Sky*, n.d.) objects within the solar system. Reflecting this, the design of the model consists of displaying simple geometric shapes (mostly spheres) and icons of similar size flowing across the shown orbits. The user interface has a simple and straight-to-business approach. The user has control over the start and stop of the animation as well as its speed – varying the speed with reference to real time (e.g., 1 second = 6 hours). The user may also zoom in and out as well as choose which object the camera is "hinged" on in terms of various camera rotations. The 3D model viewer implements a milky-way galaxy image as background.

The visual design appears to be trivialised as all objects are simple two-dimensional shapes. Their precise positional movement is prioritised over the realistic animation of the planets themselves. Thus, the presence of some secondary (in terms of perceived importance) objects is ignored. Some of the ignored objects include asteroids and moons. However, this model does track the position of some objects of interest to the design of this system such as comets. Figure 1 presents this simplistic UI.

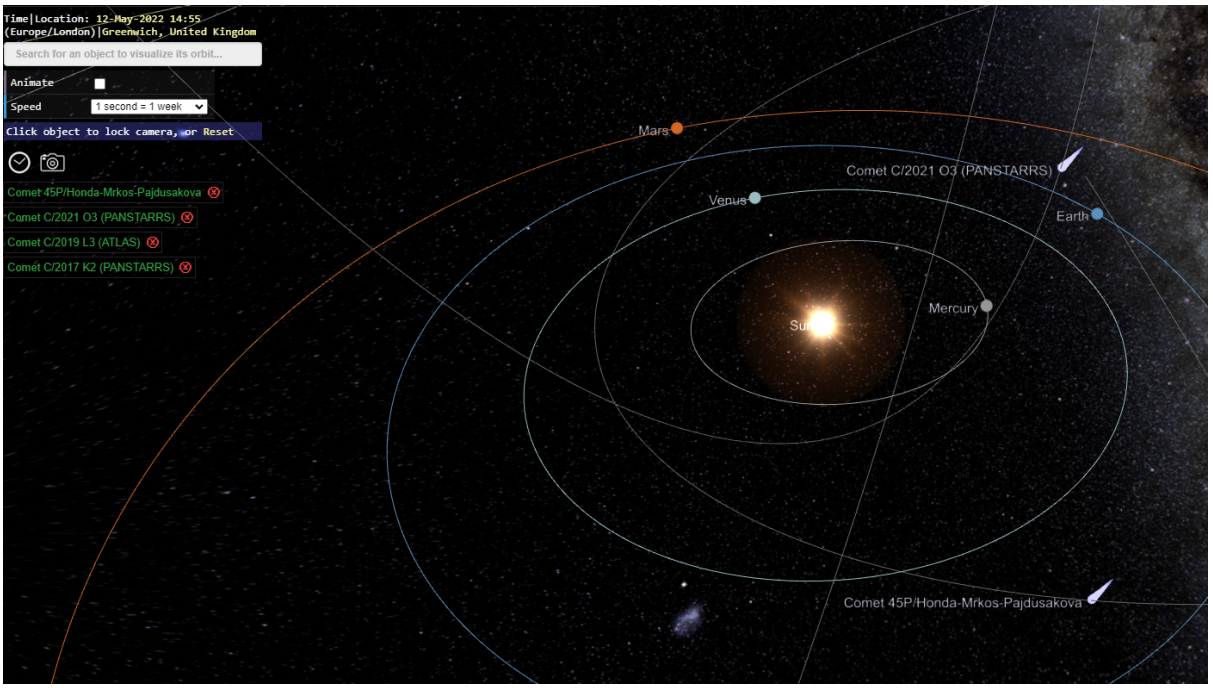


Fig 1. The Interface and visual design of the skylive.com 3D viewer

Overall, this model's focus is precise, live location of most notable objects within our solar system. Therefore, its realistic features are limited to position, object revolution along its orbit and their speed relative to user time reference input.

2.1.2 Solarsystemscope.com

Solarsystemscope.com (*Solar System Scope*, 2011) is another interactive solar system visualisation website. As opposed to the above model, this project makes an emphasis on the visual design and aesthetic appeal. This is exhibited in both the 3D viewer as well as the user interface. This particular solar system interface has been developed through the use of the Unity3D game engine.

Similarly, to the previous model, this design allows for control of the animation speed control (start / stop / speed), however also offering the backwards time controls whereby the animation is reversed. The website makes a concentrated effort on customizability of the system – allowing the user to hide or display different space objects, nameplates,

constellations, orbits, asteroids, and others. There exist multiple modes such as realistic planet sizes and the close-up exploration of each object. The size of the interactable universe extends beyond our solar system as the user can minimise (zoom out) the model several times up until the point where the milky-way galaxy is seen as a whole.

Visually, upon launch all the possible objects are displayed. Planets are shown to be of similar size and have colour rendered models true to their real-life counterparts. The online model, furthermore, utilises some lighting and shading effects. The lighting utilised complements another one of the realistic features implemented in this project: the planets' revolution around their own axii. Due to the high number of objects and visual effects drawn in this viewer, the visibility of the main solar system is somewhat obstructed as shown in Figure 2.

Fig 2. Interface and visual design of solarsystemscope.com model

Overall, it is clear to see that the main requirements for this design have been the highly customisable interface with exaggerated effort made on visual accuracy and aesthetic appeal. As with the previous project, some of the realistic features have been sacrificed for better user experience (such as making all objects roughly the same size for clarity).

2.1.3 Open-Source Project by Connor Gaskell

This is an open-source GitHub project (Connor Gaskell, 2021) that the author has chosen to include in this part of the report due to its high relevance to the problem outlined above. The developer of this program has used java3D API for the purposes of this project.

Functionally, this solar system visualisation, similarly to others, presents a three-dimensional view if the solar system completes with all planets orbiting the sun with their respective trajectory paths. The camera rotation and position are freely changed with the user's mouse and are fixed around the centre of the graphical window (the sun). The model does not include Pluto with its off-horizontal axis of rotation. The user control is limited to the control and zoom of the camera as there are no visible, on-screen settings or sliders. The developer specifies that the distances and planet sizes are not to scale and can be edited within the files if the user chooses to do so.

Visually, as mentioned above, the animation prioritises clarity of observation and the basic structure of the solar system as opposed to exact realism. The planets are all rendered in realistic fashion as displayed in Figure 3..

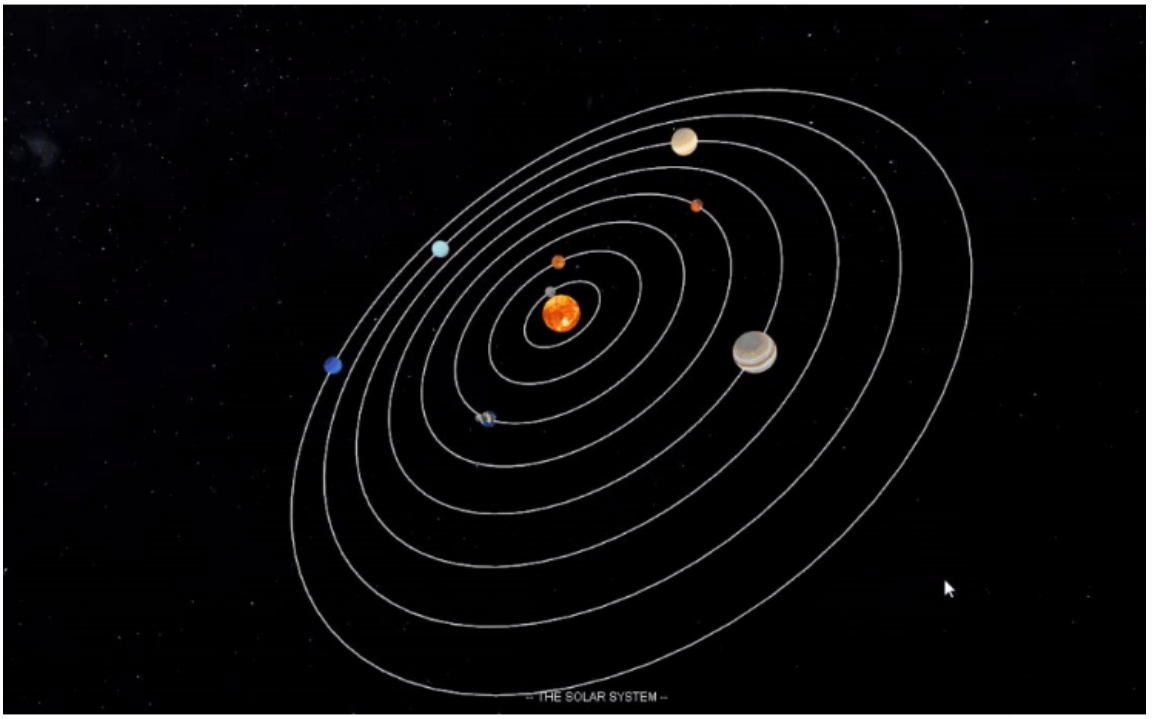


Figure 3. The Visual design of Connor Gaskell's open source java 3D projects

2.2 Analysis of Features Implemented

2.2.1 Functionality

All the discussed artefacts implement the basic functionality expected of a solar system model. This includes the relative positions of space objects, their revolution around the sun and other planets (in case of the Moon) and their respective travel paths. Additionally, all the models include a free motion camera that allows for control of the perspective position through zooming in and out as well as rotation of the camera around the centre of the graphical window. Some of the discussed systems (1 and 2) have the option of “unlocking” the camera or changing the point around which the camera rotations are performed.

Two of the three (1 and 2) models implement the time screen which displays the date and time dependent on the position and rotation of objects in the system.

Due to the different development environments, varying requirements, and scope of the projects some features may be seen in some projects but not the others. An example of such a feature is the texture rendering and animation of tertiary objects such as notable moons of each planet and the asteroid belt (present in project 2). Two of the three researched models implement a time system whereby the user may select the speed of animation and see the realistic motion of objects relative to the input time (i.e., 1 second of animation = 24 hrs)

2.2.2. Design

Looking at the design decisions of each model, it should be noted that all models (at least on default settings) agree on some aspects of visual representation. One such decision is prioritising visual clarity over complete and exact dimensions and distance of each object relative to others. This behaviour exhibits itself through the increased size of each planet and decreased perceived distance between the orbits

The specific requirements and goals set by each developer for their models dictate the trivialisation or exaggeration of certain aspects of the design. For example, theskylive.com is

not concerned with increased visual complexity of each object, and instead opts for a simplistic, clear and unconvoluted model.

2.2.3 Development Platforms

Theskylive.com and three.js

Theskylive.com is developed using the Three.js. Three.js is a JavaScript library and an API that allows the programming of 3D graphics and their embedment in webpages. Three.js can be characterised as an extension of WebGL (a controller API for OpenGL – a high level shading language in C) which allows for the integration of 3D graphics for most browsers without the use of specialised plug-ins (WebGL Fundamentals, 2012). Hence, the utilisation of this development tool seems highly suitable for embedding the running model into the website aimed for constant real-life monitoring of cosmic objects.

Solarsystemscope.com and unity

Looking into this project, the system here has been developed using the Unity game engine. Game engines such as Unreal and Unity are well established code bases used for 2D and 3D projects such as games, filmmaking and engineering. Their real usefulness is ascribed to allowing the creation of these projects to be available to a wider range of developers. The unity engine, while still having a steep learning curve, eases the requirements for the development of these artefacts. This is due to the existence of an API that does not require any sort of coding experience. Having been established since 2007 this software can be trusted to be a well-tuned comprehensive tool for many projects.

Connor Gaskell open source project – java3D

Similarly to Three.js java3D is another API that was originally built on OpenGL, however has since moved to Java OpenGL – a Java specific library for OpenGL (Friesen, 2008). One of the main advantages of this coding environment for this project is the fact that the 3D application developed is entirely programmed in java and does not require any additional skill set. However, some issues exist such as the noticeable lowered performance during the operation of the Java Garbage Collector (a submodule that deals with removal of objects during an animation) (Selman, 2002). Additionally, this development environment runs into some complications upon the distribution of the artefact to end users. It is difficult to embed into a website as well as requiring the end user to go through the process of a series of downloads should they want to run the original source code.

3. Methodology

Project management planning allows for steady development with regular progress assessment as well as planned processes and coding cycles. It is important for most projects to utilise some methodological approach to maintain focus and produce timely delivery of product features.

Conceptually, the most common software development models can be categorised into 2 types: linear and Agile. Generally, linear (sequential) methodologies involve a direct step-by-step process with no overlap between each development phase. They include methodologies like the Waterfall and V. Contrasting the sequential models, the Agile methods usually involve some form of iteration within the process. The following section will briefly discuss the considered methodologies in each category and give the reasoning behind their acceptance or denial.

3.1. Linear / Sequential development methodologies.

The waterfall approach makes an emphasis on the design and development of the whole system before any of the testing is performed. This incurs a significant risk of numerous faults occurring within the submodules of the program. This is viewed as acceptable due to the size of the team and the comprehensive documentation that is a prerequisite for this management framework.

Similarly, to the Waterfall model, **the V model** requires the consecutive design, implementation and testing of the entire system. However, unlike the waterfall, the V model plans testing parallel to each stage of production. Although this directly addresses one of the major disadvantages of the waterfall framework, it also increases the required time resources.

Evaluation (Rejection)

Due to the time constraint and the small development team, it is risky to assume that all features will be implemented. Thus, attempting to design the entire system with complete interaction of all sub-modules may lead to failure. Failure to complete 1 module on time may impede the function of others. The linearity of such processes also does not provide the room for prototyping – dynamic change becomes hard to achieve. Therefore, for the purposes of this dissertation the **sequential methodologies are considered unsuitable.**

3.2 Agile Scrum

Agile development frameworks allow for quick delivery of features as well frequent review permitting dynamic response to changing requirements.

Scrum is one of the most popular methodologies of its type. It is widely used in both large and small projects due to its high flexibility. One of its main advantages is increased productivity. It is a result of set deadlines for each of the product deliverables. The frequent progress analysis (daily scrum meeting) is not fully applicable due to the team consisting of

one person; therefore, the full framework may not be utilised to its full capacity. Additionally, setting each sprint duration may prove to be challenging, resulting in either an overestimation or an underestimation of the required development time. This is caused by relative inexperience constraints discussed earlier in this report.

Evaluation (accepted for consideration)

While the full methodology may not be used, scrum remains a highly compatible framework for the proposed product. In that capacity, parts of it could be integrated into the development cycle to take advantage of its high flexibility and fast delivery.

3.3 Incremental

Incremental methodology breaks the system down into smaller components that are built up in increments, leading to a complete artefact. Implementing this development model makes it easier to focus on each core feature at a time, while gradually adding extra feature to the whole system. The testing occurs after the completion of the coding of each increment – fixing problems as they occur. The major product requirement can be easily identified for this project. Therefore, it is simple to define and plan the initial increments, while still permitting room for design flexibility in later increments. Given this, the time deadlines are not specified for each increment which could potentially result in slow production.

Evaluation (accepted for consideration)

Overall, this model seems suitable for the purpose. The potentially new technologies / software utilised throughout the development of the artefact could impede development of the entire system. The breakdown of the problem into smaller increments simplifies the

process. However, should this model be used, deadlines for each increment need to be set to avoid productivity issues.

3.4 Overall Decision

Overall, the management methodology chosen for this project is **incremental with integrated scrum elements**. The incremental approach allows the breakdown of the problem into easily digestible components and the steady development of the complete system. The integration of scrum elements into this model will set specific tasks that need to be solved for each increment as well as add a timeline during which each increment needs to be complete. The sprint backlog is merged with the design and development stage of the incremental model. The merged design of the chosen utilised development model is detailed in Figure X below. Generally, a set of goals for the first few increments are defined prior to any development. Then as the production proceeds into increment specific design stages, the design for the implementation of those increment specific goals is planned and carried out.

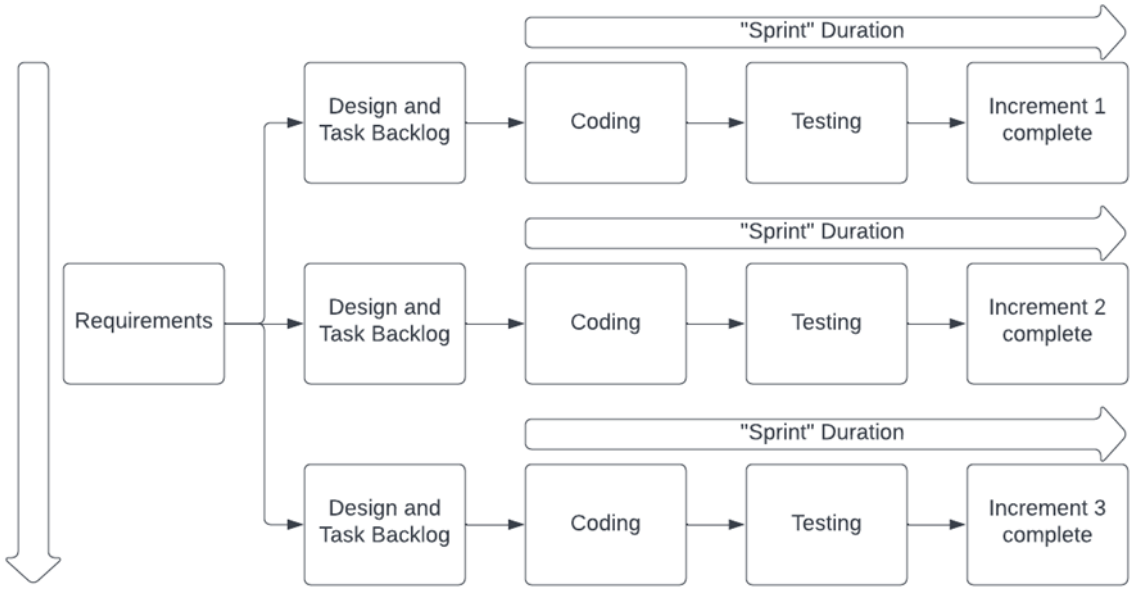


Figure 4. Diagram showing the chosen methodology method - drawn in lucidchart

4. Requirements

The following section will cover the specification of the overall requirements of the system. It will discuss the important aspects of the system, break down how the requirements were derived from them, describe the non-functional and functional requirements as well as attempt to solve any arising conflicts between them.

4.1. Requirement Elicitation

As part of section 2, the reviewed sources were analysed to see how their potential requirements were addressed with the design and functionality of the source's model. As a lot of the basic objectives and requirements of the discussed systems overlap with the artefact developed in this project, it is justified to adopt those here. Such requirements are the basic structure of the visualisation (i.e., structure of the planets, distances, dimensions, and general visual design). However, it is important to recognise where the requirements of the review material diverge from the goals of the solution developed in this report. Below subsections will discuss some of the most important aspects of the proposed solution that will be taken into consideration during the development of the product.

Model

The basic model requirements are relatively easy to derive. It is the most overlapping aspect of the previously reviewed sources and the artefact examined here. Simply put, the most basic and essential requirements of the proposed model are the working and accurate object movement and position in 3D space.

Realism

One of the main aims of this project is to create an educative model that can realistically portray the structure of our solar system. Therefore, it is important to follow specific relative sizes, distances, and movement of the objects within the model. 2 of the reviewed sources have only partially implemented this feature – mainly with the distances. The other source has the option for realistic object sizes; however, it takes a considerable effort to find it in the settings.

UX design and UI

When it comes to the user experience design and interface, the model must remain clearly visible and easy to operate. The user must be able to locate and inspect planets easily and be able to operate the camera angle and zoom to obtain the desirable perspective. The model is solely focused on our solar system so other potential objects outside the system should not complicate or obstruct the view. Such practice may be seen in solarsystemscope.com whereby upon launch the presence and high density of objects creates a close to indiscernible interface. The interactive interface itself must be simple and must be designed in an intuitive manner avoiding unnecessary additional windows of UI.

Some of the potential extra features that can be implemented in the consequent increments may involve the texture rendering of the objects as well as lighting / shading effects. This can improve the aesthetics of the system and therefore add to the positive user experience.

UX vs Realism

It should be noted that the previous subsections conflict with each other. Creating a model true to its real-world counterpart creates issues that affect the user experience. One such issue is the relative dimensions of planets. The diameter of the sun is over 100 times larger than the diameter of the earth. Developing a model of such a system where all the objects remain visible is impossible due to the size of the sun. Hence, implementing strictly accurate sizes of planets may be problematic from a user point of view. A decision needs to be made on which aspect of the design should be prioritised or whether a compromise can be made.

4.2. Functional Requirements

The functional requirements outlined in the below table are categorised into 3 main priority groups:

1 – Highest priority. This requirement must be satisfied first and foremost before any other features are added. The most basic system functions fall into this category.

2 – Medium priority. This category includes system functionality that is also essential to model operation and is built on top of the basic components. These should be fulfilled by the end of full system completion upon several system iterations (increments).

3 – Low priority. These are the extra submodules that should only be developed once the requirements from categories 1 and 2 have been fulfilled. These are intended for further expansion of the model and are usually aimed at improved user experience.

The features are assigned priority levels based on a few factors. One is the importance of the aspect or feature of the system to the development of others. For example, The creation and animation of objects is a prerequisite for the development of UI. Another factor is the importance of a feature with regards to the aim of the project. Some features like high accuracy are part of the core objectives of this report and as such rank high in priority.

Index	Requirement	Description	Priority
1	Complete implementation of all significant objects	The model must contain the 3D objects of all the planets, significant moons, and the sun.	1

2	Object accurate movement	Objects must behave realistically. This entails a close approximation to their movement and trajectory in the real world. This includes rotation around the correct axis and axis angle. The objects themselves should rotate as well.	1
3	Object Trajectories	The model should display the trajectory of most significant objects by default.	2
4	Accurate Dimensions & Distances	Planets must have correct diameters relative to each other and must be positioned at approximately correct distances.	1
5	Intuitive camera movement	Camera should be able to move freely with mouse drag. Zooming in and out should be implemented	2
6	Planet Identifiers	The user must be able to see planet names	2
7	Basic model options	The model should provide some basic options to the user such as being able to make the planets uniform in size or decrease / increase size of chosen objects	3
8	Advanced Camera Movement	User should be able to pan the camera in such a way that pans its focus on a particular object (via clicking, double clicking or using the UI options)	3

Table 1. Functional Requirements

4.3 Non-Functional Requirements

Index	Requirement	Priority
1	Model must be realistic	1
2	The model should be clearly visible	1
3	The system must be accessible and intuitive to use	1
4	The user should be able to interact with the system via operating the camera perspective and UI	2
5	The UI must be simplistic and non-intrusive to the displayed model by default	2
6	The visualisation should be able to be easily embedded into a website	3
7		

Table 2. Non-Functional Requirements

5. Design

The design section of this report will document the design related decisions made prior to the beginning of the development of the artefact. These decisions will preface the implementation and lay the groundwork for this project, allowing for efficient start.

It will, firstly, cover the software and technologies chosen for this project and justify their use in the context of requirements and limitations. Secondly, it will discuss the choices made with regards to the aesthetic design of the model and the user interface aspects of the program. Lastly, this section will cover the order of development and will outline the goals for the first few increments of implementation.

5.1 Software

The software that was chosen for this project is Processing. Processing is a programming tool for development of visual programs in multiple languages. By default, it is set to the java language – which is the preferred language for this project. One of its main features is the incorporation of the visualisation window into the IDE. As such any written code can be executed and consecutively shown in the “sketch” window without the need to interconnect multiple software applications together. It has extensive and highly comprehensive online documentations making it easy to find the required functionality quickly and easily.

Considering one of the limitations of this project being the relative inexperience of the author in the 3D graphics programming field, this is highly useful for future development. Additionally, due to the existence of P5.js – a software for the interpretation of processing for the web, any processing sketch can be easily compiled into JavaScript code and embedded into an HTML webpage. This falls in line with one of the main objectives of this project – an easily accessible visualisation for education purposes.

Other development platforms that were reviewed in the literature review section are deemed not appropriate for this project. Java3D, while having a large history of use, remains an older piece of technology that is considered outdated. Additionally, the difficulty of learning it may pose a serious hindrance to the project progress. Unlike java3D, game engines such hold a significant place in modern animation and are widely used. However, even more-so than Java 3D the learning curve for such software is steep and may impede progress.

5.2 Design Decisions

There are multiple aspects of the proposed solar system visualisation that need to be considered prior to the development. Mainly, these are the specifications of the model itself (which objects are to be displayed; how should they move), the accuracy of the system relative to the real-world figures and the user interface aesthetic and functional properties.

5.2.1 Model Specifications

The program developed will only be concerned with the main points of interest within our solar system. These will include the 8 planets, the sun and Pluto as well as some notable objects such as the moon. Depending on the progress of the development more objects can be integrated into the system, however they are not prioritised. As a basic requirement the following objects must be included within the system.

- Sun
- Mercury
- Venus
- Earth
- Mars
- Jupiter
- Saturn
- Uranus
- Neptune
- Pluto
- Earth's Moon

The system should implement their presence as globes or spheres in rotation around the sun and themselves. All objects must be visible either upon execution of the program or the further user interaction. The objects should follow a visible trajectory / path around the 3D space.

5.2.2 Realism

One of the main aims of this project is to produce a “realistic” solar system. Specifically, this concerns the question of how accurate the dimensions and relative distances need to be to be considered realistic.

The system will aim to achieve as close approximation to the exact proportions as possible with the exception of the sun. It should be noted that due to the significant difference between the size of the sun relative to other planets, the complete adherence to the exact size will affect the user experience of the program. As it is, the sun would cover the majority of the 3D space, leaving the other objects effectively invisible. Therefore, during the design of the system the sun must be significantly reduced in size. Specifically, by default, the system will implement the sun to be of slightly larger diameter than the largest object within the model. It should, however, be possible to increase the size of the sun to its correct dimension should the user choose to.

The orbital inclination is in other words an angle of the planet’s orbit to the reference axis. Most objects display some degree of angular rotation. However, in most such cases the orbital inclination is almost zero degrees and therefore portrays no visible difference to zero. As a result of this, it is enough to only consider significant inclinations of over 10 degrees (>10 degrees) for the model to achieve realism in this aspect. This would primarily affect Pluto.

No other changes to the dimensions of the objects are made. The rest of the system should emulate the exact rotation speed, direction, and angle of orbit.

5.2.3 UI and Functionality

In the literature review section of this document, it has been observed that the over saturation of the user interface with information can be undesirable. By default, upon the launch of the program only the necessary objects should be displayed on the screen.

The program should involve a simple, intuitive camera movement and / or a pre-set position of the camera at an optimal angle. The user should be able to freely rotate the camera with mouse movements and be able to reset the camera to default perspective.

The UI must be non-invasive and simple. It should consist of several buttons and sliders at the top left of the sketch screen. The diagram X below shows the UI design.

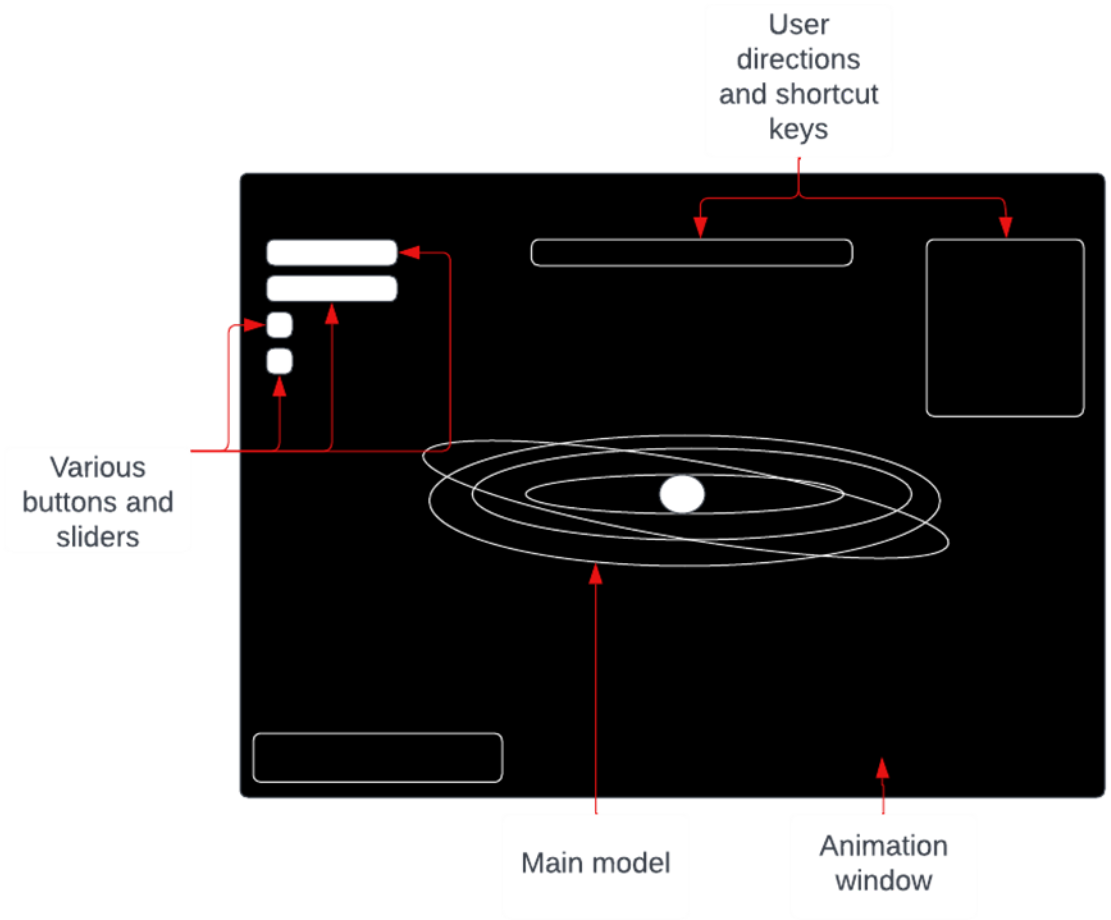


Figure 5. The proposed UI idea upon running the program

The following simple functionality should be implemented upon successful completion of the base model:

Name	Button / Slider / Click	Functionality
show/hide names	Button	A button that would display or hide the names of the drawn objects. By default, the names will be displayed on the screen.
Show/hide trajectories	Button	A button that would show or hide the path along which the planets orbit the sun.
match planet sizes	Button	A button that increases the sizes of smaller planets to match the size of bigger ones to make them more visible
stop/start rotation	Button	A button that stops / resumes the planet rotation around the sun
increase / decrease rotation speed	Slider	A slider that allows the user to vary the speed of the animation (faster or slower)
Reset Camera Perspective	Button	Reset the camera position to its initial position on launch.
Centre the Camera on a specific object	Click / Double Click	On click or double click of a particular project during the animation the camera centres around the clicked object at some distance away.

Table 3. UI functionality

5.3 Incremental Objectives

As was discussed earlier in the methodology section, an incremental approach with the elements of scrum will be carried out. Therefore, some rough goals need to be set for the first few increments to establish the workflow. This will help outline the first basic objectives and shape the base of code on which further development can be done. Depending on the issues that arise during the implementations of these increments, further undefined increments can be more easily analysed and planned.

Considering the employed methodology, this can be said to represent the analysis and design stage of the process. More concrete ideas can be defined for the first increment; however further planning needs to be rougher to allow for dynamic change. Considering the limitations of learning new technology, such malleability is important.

5.3.1 Increment 1

The first increment should focus on the basic implementation of the sketch window and the object classes. The main goal of the first increment is the creation of the base of code on which further development can be built. The following are a list of objectives to be achieved by the end of the first increment.

- Create the main generation file. This should include the definition of the sketch window and any prerequisite functions that it entails.
- Create the object class / classes file. This file concerns the generation of objects within the model and their basic properties.
- Write functions that allow basic rotation of one object around another. This should emulate the circular rotation of a planet around the sun.

5.3.2 Increment 2

The second increment should complete the object class / classes and introduce the major elements into the program. It will aim to develop a code backbone that will satisfy the high priority requirements and will permit further development. The following are the minimum deliverables aimed to be produced in this increment.

- Complete generation of most or all objects listed in 5.2.1.
- Complete emulation of accurate relative distances and sizes as per real-world data with the exception of the sun.
- Realistic circle rotation of objects around the sun.

5.3.3 Increment 3

It is yet unclear if or what issues will come up during the process of development of the first two increments. Therefore, it is imperative to allow room for change, especially while planning far ahead into the production. The immediate issues that occur are dealt with within the increment that they occur in. However, depending on the issue, this may result in the reconstruction of the previously finished features. This can lead to previously planned methods of implementation becoming obsolete. Hence, it is important to employ more generalised goals.

- Adapt existing code against the occurred issues (if any)
- Implement object trajectories for major objects. These include the 8 planets and Pluto.
- Starting to implement variables and functions for use in the UI.
- Implement lighting features
- Improve the aesthetic look of the model. Specifically focus on acquiring and rendering the textures of the main objects within the model.

5.3.4. Further Increments

By this point in the production, the main features should already be achieved. Therefore, further increment will focus on the development of the UI, optimising of the code and the addition of any extra tertiary features. As such, the following are the design goals for further increments leading to the completion of the artefact.

- Full completion of all UI features
- Completed lighting such that it enables clarity of the animation as well as improved aesthetic design.
- Optimisation of the code and removal of any obsolete functions.

6. Implementation

Implementation chapter of this report documents the process of development of the artefact. This section first briefly outlines the fundamental prerequisites that the code has to adhere to. It then covers the stage-by-stage progress of the implementation and looks to elaborate on specific design choices made during the production. The previously discussed plans for increments will be used as guidelines and short-term goals on a per increment basis. This section can be regarded as the main content of this paper and as such hopes to acquire the final product by the end of it.

6.1 Processing fundamentals

In the design section of this report, it is stated that the software for 3D graphics coding that will be utilised in this project is Processing. Processing features an almost identical programming language to java. The code written is easily converted to java itself and executed.

The main aspect of processing that a developer needs to be aware of to be able to use this software is the two basic functions that need to exist within the program. These functions

are **setup()** and **draw()**. The code will not be executed correctly unless these two functions appear in the main file.

The setup() function block is executed once when the program is run. It is typically used to initialise objects and load the required assets. This is the part of the code where, commonly, a “sketch” window properties such as size are declared, and various objects are generated.

The draw() block operates as the backbone of the animation. The draw() function is effectively a recurring loop that runs until the code window is closed. By default, the block is executed top to bottom at a rate of 60 times a second.

Figure 6 Basic Processing interface.

6.2 3D vectors

Throughout the development the concept of positional vectors will be used. Positional vectors are effectively a pointer to an exact location in 3D space. They take the values of the x,y,z axes as parameters and draw a directional line to that position. Positional vectors in processing are useful as they possess a number of methods such as vector.mult(float value) and v.cross(PVector vector). These functions are used extensively to manipulate the positions of objects on the screen. The cross product method v.cross is particularly useful for the development of the artefact. The cross product of two vectors is a vector that is

perpendicular to both vectors specified. Using this characteristic, it is possible to define an exact vector around which the planets created can rotate within the animation.

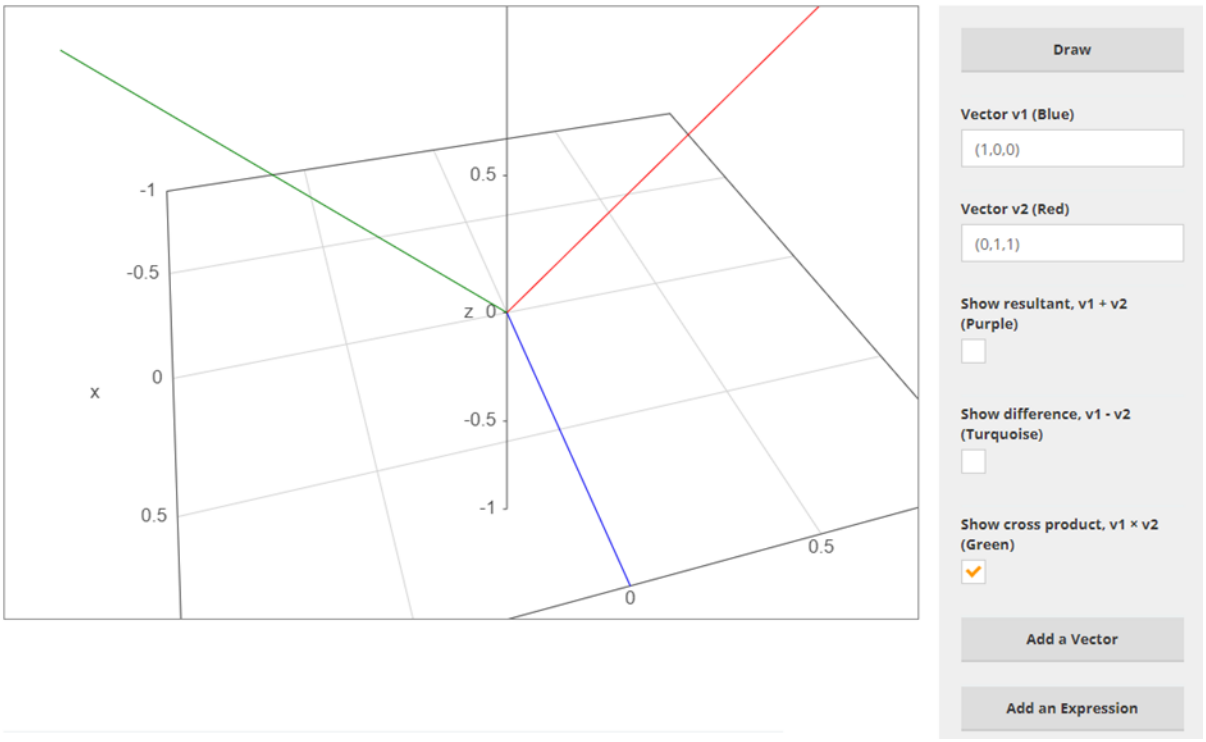


Figure 7. Graphical Representation of a cross product vector – displayed on academo.org 3d-vector-plotter

6.3 Increment 1

Increment 1 of the program establishes the first code base.

In this increment two main classes are created. The main “Generation” class and the object class “planet”. The main class should generate the sketch window, declare and initialise the objects of class Planet, and execute any relevant functions.

6.3.1 Planet file and class

The planet class is an object class that is dedicated to the creation and management of most objects within the created model. The name “planet” is used loosely here as all spherical objects (the Sun, moons, Pluto) will be categorised as part of this class. These objects must

have certain parameters the variation of which can differentiate them from one another. The parameters deduced for this class are shown in the table below.

Type of Data	Name	Description
float	radius	defines the radius of the sphere being generated and therefore its overall size.
float	angle	angle in radiance. Angle variable is important as the rotation functions within processing must have an angle variable.
float	distance	distance from the centre of the animation window.
float	rotationSpeed	rotationSpeed is a variable that specifies the speed of the objects orbiting the sun
PVector	v	v is a variable that denotes a precise location of an object in 3D space. It has x,y and z parameters.

Table 4. Initial Planet class design

The variables radius, distance and rotationSpeed are the variables by which the program will be able to reference real world figures and therefore accurately visualise the distances, sizes and speeds of the planets and moons within our solar system. The constructor that will allow the creation of such objects will take those three variables as parameters.

```
class planet {

  float radius; // radius of sphere - size of planet
  //angle variable need to exist for rotate function to be used. angle needs to be
specified in radians i.e. [0, 2pi]
  float angle;
  float distance;     // distance factor
  float rotationSpeed;  // increment used to increase / decrease angle
  PVector v; // 3d positional vector

  planet[] planets;

  planet(float r, float d, float s){
    v = PVector.random3D(); // randomly directed vector
    radius = r;
    angle = random(TWO_PI);
    distance = d;
    rotationSpeed = s;
    // unit vector can be regarded as length = 1, multiplying by distance
simulates the actual distance between objects
    v.mult(distance);

  }
```

To draw the objects created on the screen it is essential to have a method that draws the objects as shapes in the animation window. This is the method that would be run recursively in the draw() function in the main file. Every object must be visibly displayed in an animation, therefore a show() function is incorporated into the planet class. This function will translate the object generated from the centre of the window to the desired coordinates (vector), create a sphere object with the input variables and rotate it around the cross vector. It should also loop through the created planets array for each object and show each of their generated moons.

```
  void show() {

    pushMatrix();
    noStroke();
    fill(255);

    PVector v2 = new PVector(1,0,1);
    PVector p = v.cross(v2);

    rotate(angle, p.x, p.y, p.z);
    stroke(255);
    line(0, 0, 0, v.x, v.y, v.z);
    //line(0, 0, 0, p.x, p.y, p.z);
    translate(v.x,v.y,v.z);
    noStroke();
```

```
    sphere(radius);

    fill(255);

    if (planets != null) {
      for (int i = 0; i < planets.length; i++){
      planets[i].show();
      }
    }
    popMatrix();
  }
```

Next, to ensure and practice the rotation of the objects around one another a function to generate the moons around the planets is written. The idea behind this function is that it is going to create an input number of moons with random parameters that will orbit the object via which the method is called. These new objects are added into a “planets” array so that they can be accessed for modification. This will ensure the show() method operates correctly and can be adapted later to be used for the creation of moons around different planets within the solar system.

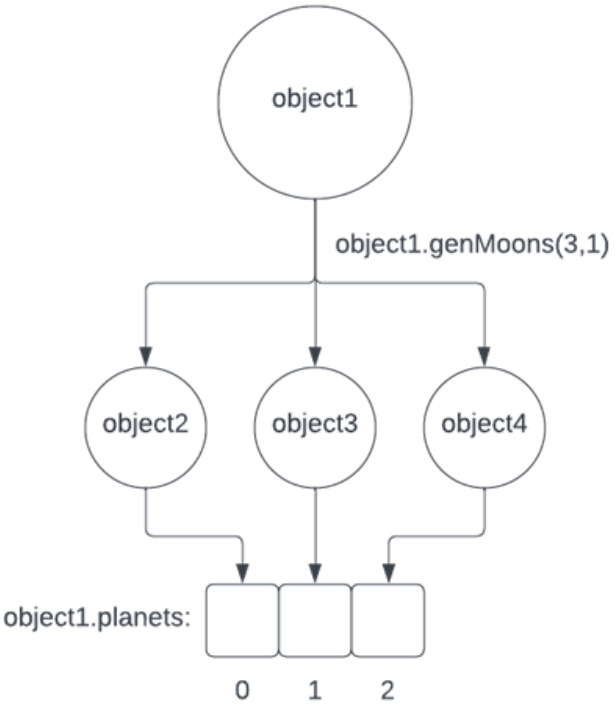


Figure 8. Structural design of genMoons function

```
  void genMoons(int number, int level) {
    planets = new planet[number];
    for (int i = 0; i < planets.length; i++){
      float r = radius/(level*2); // every recursion of moons' sizes reduced from
original i.e. 1/2, 1/4, 1/6 ...
      float d = random(radius + r, (radius + r)*2);
      float o = random(-0.01, 0.01);    // forward and backward rotation
      planets[i] = new planet(r, d, o);
      // for below specified level of recursion, each "moon" will generate its own
"num" number of moons.
      if (level < 1){
        int num = 1;
        planets[i].genMoons(num, level+1);
      }

    }
  }
```

At this point, this function is only used for testing and thus does not need any specific values entered as parameters. The values for object variables here are random.

Lastly, to visualise the dynamic rotation (orbiting) of the objects another method needs to be implemented. Utilising the recursive loop of the draw() function in the main class and the rotationSpeed variable, an orbit() method is created. This method will simply add the rotationSpeed value to the angle repeatedly as the execution of the program loops draw().

```
  void orbit(){
    angle = angle + rotationSpeed;

    if (planets != null){
      for (int i = 0; i < planets.length; i++){
        planets[i].orbit();
      }
    }

  }
```

6.3.2 Generation file

The generation file for this increment consists of initialising a sphere planet object and testing the rotation of the generated moon objects via the genMoons() function.

```
planet sun;

void setup() {
  //Initialise window and 3D module
  size(600,600, P3D);
  sun = new planet(50, 0, 0);
  sun.genMoons(3, 1);
}

void draw() {
  background(0);
  //By default objects are drawn at (0,0,0)
  //Translate to the centre of the window.
  translate(width/2, height/2);
  lights();
  sun.show();
  sun.orbit();
}
```

Upon execution, the functions appear to work correctly as the sphere is drawn in the centre of the screen and the 3 “moons” orbit it at random angles.

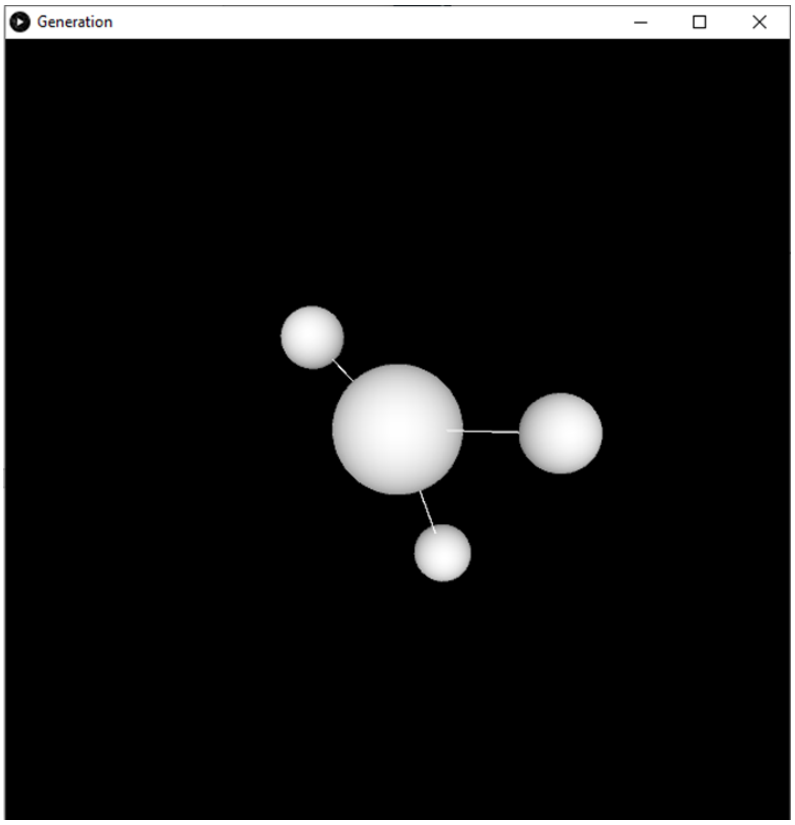


Figure 9. Increment 1 complete

6.4 Increment 2

Increment 2 goals include the complete delivery of the highest priority functionality. Referencing the established requirements, the highest priority functions are the complete implementation of all of the planned objects and their realistic movement. This increment will obtain real world data, make configurative adjustments to the classes and implement the desired objects.

6.4.1 Parameter Adjustments

To start generating the geometrically correct planet objects a few changes need to be made to the planet class. It should be noted that the positional vectors that are used in this program show a direct distance from the centre of the window to the coordinates of the specified vector. Creating a sphere (the sun) in the middle of the sketch window and another sphere (generated planet) reduces the visible distance between the 2 spheres. The radii of the 2 spheres would be included in the total distance between the two objects as the spheres are drawn from the centre of the input coordinates. Graphically, this is represented by the figure below.

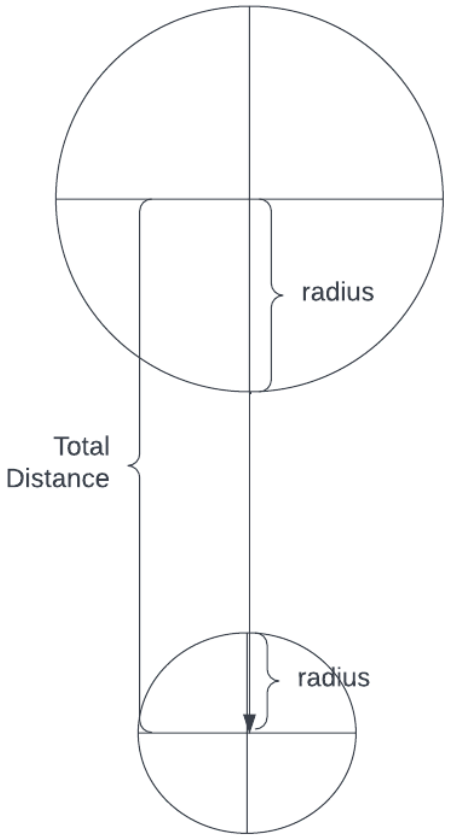


Fig 10. Graphical representation of the added distance due to radius.

Therefore an **offset** parameter needs to exist within the planet class constructor to account for the extra distance covered by the 2 radii.

Most objects within the solar system structurally lie on the same axis X. This means that for most objects it is easy to set up a vector cross product. In processing the axis are set as shown below.

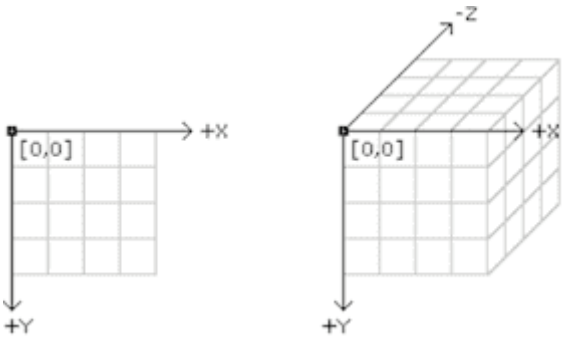


Fig 11. Layout of the axis in a Processing graphical window (*P3D*, n.d.)

Therefore, it is enough for most objects to contain a vector v1 = (1,0,0) and a vector v2 = (0,0,1) vector to obtain a cross product vector of p = (0,1,0). Vector v1 is defined in the planet class constructor and vector v2 is set as a **reference** vector in the show() method. Vector p is effectively a vertical line denoting the Y axis. Using a rotation function in processing with this vector and an angle will program the rotation around the Y axis as it occurs in the real world. Note that by previous design, the object is built with a distance variable which multiplies the x,y and z coordinates of the positional vector to effectively represent the real-world distance. For that purpose, the vector (1,0,0) is too small and will result in a number of objects clashing with each. A decision is made to increase this unit vector to (100, 0, 0) to create some visible space between the objects.

However, some objects like the moons and Pluto orbit at an angle. In this case just having the vector (100,0,0) becomes insufficient. There needs to exist a way of inputting a precise vector v1 such that, with the reference vector v2, a cross product vector may be generated at an exact angle to the Y-axis. The method of calculating such a vector is discussed later in this section. The solution for this problem is the definition of multiple constructors with different parameters that could be used for different objects. The code denoting the 2 new constructors is detailed below.

```
planet(float r, float d, float s, float o){
  v = new PVector(100, 0, 0);  // increase to 1000 for larger visible distance
  radius = r;
  angle = random(TWO_PI);
  distance = d;
  speed = s;
  v.set(v.x * distance + o, v.y, v.z);
}

planet(float r, float d, float s, float o,float x, float y, float z){
  v = new PVector(x, y, z);
  radius = r;
  angle = random(TWO_PI);
  distance = d;
  speed = s;
  v.set(v.x * d + o, v.y, v.z);
}
```

6.4.2 Data Acquisition

During the development of this increment, in order to satisfy the realism and accuracy requirements, some data has to be acquired about the solar system and converted to be used in the program environment. This data includes: the relative diameters of the objects in space, their distances from the sun, their orbital speeds and their orbital inclinations.

The data regarding the precise properties of each of the planets can be easily found on NASA's website. The following reference sheet has been used as reference raw data.

Solar System Sizes and Distances

Distance from the Sun to planets in astronomical units (au):

Planet	Distance from Sun (au)
Mercury	0.39
Venus	0.72
Earth	1
Mars	1.52
Jupiter	5.2
Saturn	9.54
Uranus	19.2
Neptune	30.06

Diameter of planets and their distance from the Sun in kilometers (km):

Planet	Diameter (km)	Distance from Sun (km)
Sun	1,391,400	-
Mercury	4,879	57,900,000
Venus	12,104	108,200,000
Earth	12,756	149,600,000
Mars	6,792	227,900,000
Jupiter	142,984	778,600,000
Saturn	120,536	1,433,500,000
Uranus	51,118	2,872,500,000
Neptune	49,528	4,495,100,000

Fig 12 Table showing the relative distances and diameters of planets

Data retrieved from *NASA Jet Propulsion Laboratory (JPL)*
https://www.jpl.nasa.gov/edu/pdfs/scaless_reference.pdf

Planetary Fact Sheet - Metric

	MERCURY	VENUS	EARTH	MOON	MARS	JUPITER	SATURN	URANUS	NEPTUNE	PLUTO
Mass (10^{24}kg)	0.330	4.87	5.97	0.073	0.642	1898	568	86.8	102	0.0130
Diameter (km)	4879	12,104	12,756	3475	6792	142,984	120,536	51,118	49,528	2376
Density (kg/m^3)	5429	5243	5514	3340	3934	1326	687	1270	1638	1850
Gravity (m/s^2)	3.7	8.9	9.8	1.6	3.7	23.1	9.0	8.7	11.0	0.7
Escape Velocity (km/s)	4.3	10.4	11.2	2.4	5.0	59.5	35.5	21.3	23.5	1.3
Rotation Period (hours)	1407.6	-5832.5	23.9	655.7	24.6	9.9	10.7	-17.2	16.1	-153.3
Length of Day (hours)	4222.6	2802.0	24.0	708.7	24.7	9.9	10.7	17.2	16.1	153.3
Distance from Sun (10^6 km)	57.9	108.2	149.6	0.384*	228.0	778.5	1432.0	2867.0	4515.0	5906.4
Perihelion (10^6 km)	46.0	107.5	147.1	0.363*	206.7	740.6	1357.6	2732.7	4471.1	4436.8
Aphelion (10^6 km)	69.8	108.9	152.1	0.406*	249.3	816.4	1506.5	3001.4	4558.9	7375.9
Orbital Period (days)	88.0	224.7	365.2	27.3*	687.0	4331	10,747	30,589	59,800	90,560
Orbital Velocity (km/s)	47.4	35.0	29.8	1.0*	24.1	13.1	9.7	6.8	5.4	4.7
Orbital Inclination (degrees)	7.0	3.4	0.0	5.1	1.8	1.3	2.5	0.8	1.8	17.2
Orbital Eccentricity	0.206	0.007	0.017	0.055	0.094	0.049	0.052	0.047	0.010	0.244
Obliquity to Orbit (degrees)	0.034	177.4	23.4	6.7	25.2	3.1	26.7	97.8	28.3	122.5
Mean Temperature (C)	167	464	15	-20	-65	-110	-140	-195	-200	-225
Surface Pressure (bars)	0	92	1	0	0.01	Unknown*	Unknown*	Unknown*	Unknown*	0.00001
Number of Moons	0	0	1	0	2	79	82	27	14	5
Ring System?	No	No	No	No	No	Yes	Yes	Yes	Yes	No
Global Magnetic Field?	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	Unknown
	MERCURY	VENUS	EARTH	MOON	MARS	JUPITER	SATURN	URANUS	NEPTUNE	PLUTO

Fig 13 - NASA reference sheet (NASA, 2018)

The distance from the sun (au) listed above does not necessitate further conversion. It is to be used as raw data. As mentioned above, the positional vector (100,0,0) is used for most planets as a unit line that can be multiplied by these figures to accurately represent the relative positions of the planets. **Pluto** is discussed later in this chapter.

The diameter of planets has been downsized to smaller usable numbers as shown in the tables below. The converted value column contains the figures used as parameters for generation.

Object name	Diameter (km)	radius (km)	Converted Value
Mercury	4,879	2,4395	2.4
Venus	12,104	6,052	6.1
Earth	12,756	6,378	6.4
Mars	6,792	3,396	3.4
Jupiter	142,984	71,492	71.5
Saturn	120,536	60,268	60.3
Uranus	51,118	25,559	25.6
Neptune	49,528	24,764	24.8
Pluto	2,370	1,185	1.2
Moon	3,474	1,737	1.7
Sun	1,391,400		100

Table 5. Raw data conversion to usable data

Note: As was mentioned in the design section the sun is downsized to retain clarity of the model. It has been made larger than all of the other objects, however not so large that the planets immediately next to it become indiscernible.

Orbital velocities of planets translate into the animation as the angle increment of object rotation per frame. Though there are plans to implement UI options to affect these parameters, the model by default should operate in a smooth manner. In order to achieve this, there must be visible difference between the object speeds, however not so much so that the fastest object moves faster than the eye can follow. A reference speed should be established. For this purpose, mercury will be used. As seen in Table 5 above, Mercury has the fastest orbital velocity of 47.4 km/s. Based on previous testing in increment 1, it was deduced that the maximum angle increment should not be larger than | 0.01 | for most objects. Therefore, we can calculate the relative angle increments for all objects based on equating the reference speed of 47.4 to 0.01. The moons of planets are excepted from this

as they move relative to the planet they are a moon of. The object specific increments are calculated using the following formula:

$$(planet\ velocity\ /\ reference\ velocity)\ *\ max\ increment\ =\ increment$$

The Moon's orbital velocity is calculated relative to the Earth's. The Moon orbits the Earth 13 times in a year while the earth orbits the Sun once. Therefore, to calculate the Moon's orbital speed, the Earth's increment value should be multiplied by 13.
This produces the following table of results:

Object	Orbital Speed (km/s)	Reference Speed (km/s)	Maximum Increment	Increment Value
mercury	47.87	47.87	0.01	0.010
venus	35.02	47.87	0.01	0.007
earth	29.78	47.87	0.01	0.006
mars	24.077	47.87	0.01	0.005
jupiter	13.07	47.87	0.01	0.003
saturn	9.69	47.87	0.01	0.002
uranus	6.81	47.87	0.01	0.001
neptune	5.43	47.87	0.01	0.001
pluto	4.74	47.87	0.01	0.001
moon (mean)	1.02	47.87	0.01	0.0002

Table 6. Orbital Velocity raw data conversion.

6.4.3 Planet Generation

To reiterate the implemented design in increment 1, the generation and animation of an object generally follows this pattern:

1. Create an object through a constructor
2. Translate from the centre of the window to the coordinates of the vector (100 * distance factor)
3. Rotate the object around the cross product vector by an increment angle (recurred)

Using the above obtained data and the new constructors outlined in 6.4.1 the following planets can be generated.

```
sun = new planet(100, 0, 0, 0);
mercury = new planet (2.4, 0.39, -0.01, sun.getRadius() + 2.4);
venus = new planet (6.1, 0.72, -0.007, sun.getRadius() + 6.1);
earth = new planet (6.4, 1, -0.006, sun.getRadius() + 6.4);
mars = new planet (3.4, 1.52, -0.005, sun.getRadius() + 3.4);
jupiter = new planet (71.5, 5.2, -0.003, sun.getRadius() + 71.5);
saturn = new planet (60.3, 9.54, -0.002, sun.getRadius() + 60.3);
uranus = new planet (25.6, 19.2, -0.001, sun.getRadius() + 25.6);
neptune = new planet (24.8, 30.06, -0.001, sun.getRadius() + 24.8);
```

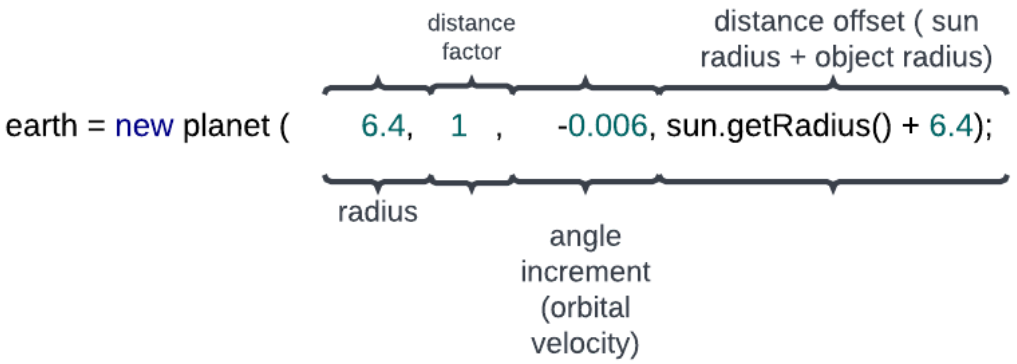


Fig 14 planet declaration annotated

These planets are, for this model, considered to have 0 orbital inclination and generally lie on the X axis and rotating around the cross product vector axis Y. This allows for the use of the first constructor where the positional vector only includes an X coordinate (100, 0, 0) which is multiplied by the distance. Pluto, on the other hand, has a relatively steep 17 degree inclination. Generation of pluto will require the use of a second constructor to specify the exact vector such that this inclination is accounted for. Such a vector is calculated using the trigonometry rules. Figure 15 helps demonstrate the above conditions.

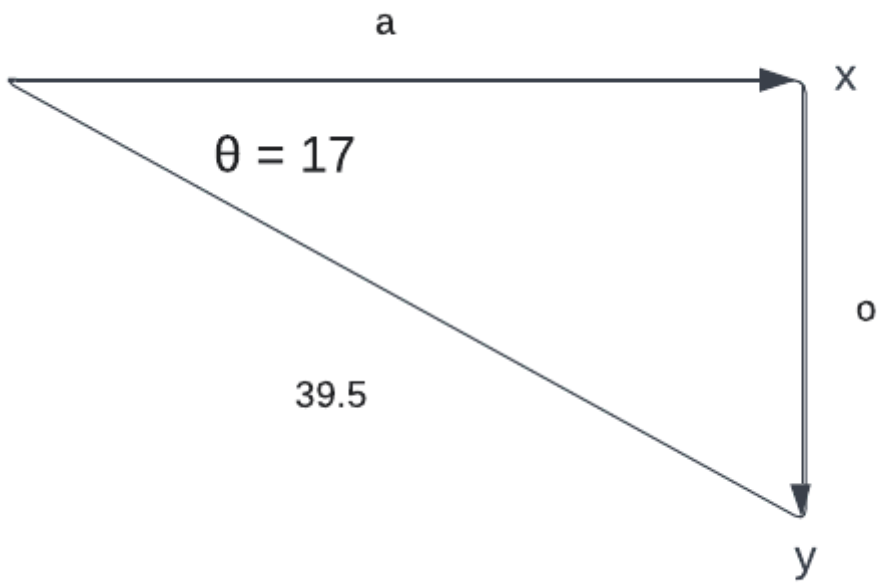


Fig 15. Diagram representing the position vector of Pluto

$A = 39.5 * cos(17) = 37.8\ (rounded\ to\ 1\ d.p.)$

$O = 39.5 * cos(17) = 11.5 (rounded\ to\ 1\ d.p.)$

These valued upscaled (multiplied by 100) for the program give us the final vector for the Generation of Pluto.

```
pluto = new planet(1.2, 39.5, -0.001, 1.2 + sun.getRadius(), 100, 1150, 0);
```

The last object left to generate is the Moon. Using a similar logic as explained above a vector for the moon is calculated. The genMoons() function in the planet class is adapted to generate the Moon as per the following code.

```
void genMoon(){
  planets = new planet[1];
    float r = 1.7;
    float d = 0.0025;
    float s = -0.078;    // 0.006 earth orbital speed * 13
    float o = 6.4 + 1.7;
    planets[0] = new planet(r, d, s, o, 1, -2, 0);

}
```

The genMoon() function is then called as an earth object method within the setup() block inside the Generation file.

6.4.4 Camera

For the purposes of testing and the future additional implementation a camera is introduced. Processing supports basic camera methods that can, by default, be set to any position and distance away from the centre of the window. However, it does not support free and intuitive camera movement with mouse drags and clicks.

Here, a camera library is chosen to be imported into this project. Peasy camera (*Peasycam*, n.d.) library, once imported, can be freely rotated and tilted around during the animation with simple mouse clicks and drags. This directly jumpstarts and sets a base for the upcoming implementation of “quality of service” functionality (last on requirements priority list) changes.

Furthermore, importing this library allows for save of time resources that can be dedicated to faster development of other features. This library can be later combined with UI for the complete functional delivery.

6.4.5 Increment 2 result

The numerous changes detailed in this section have yielded a model complete with all the objects in their up to scale sizes, positions and orbital orientations.

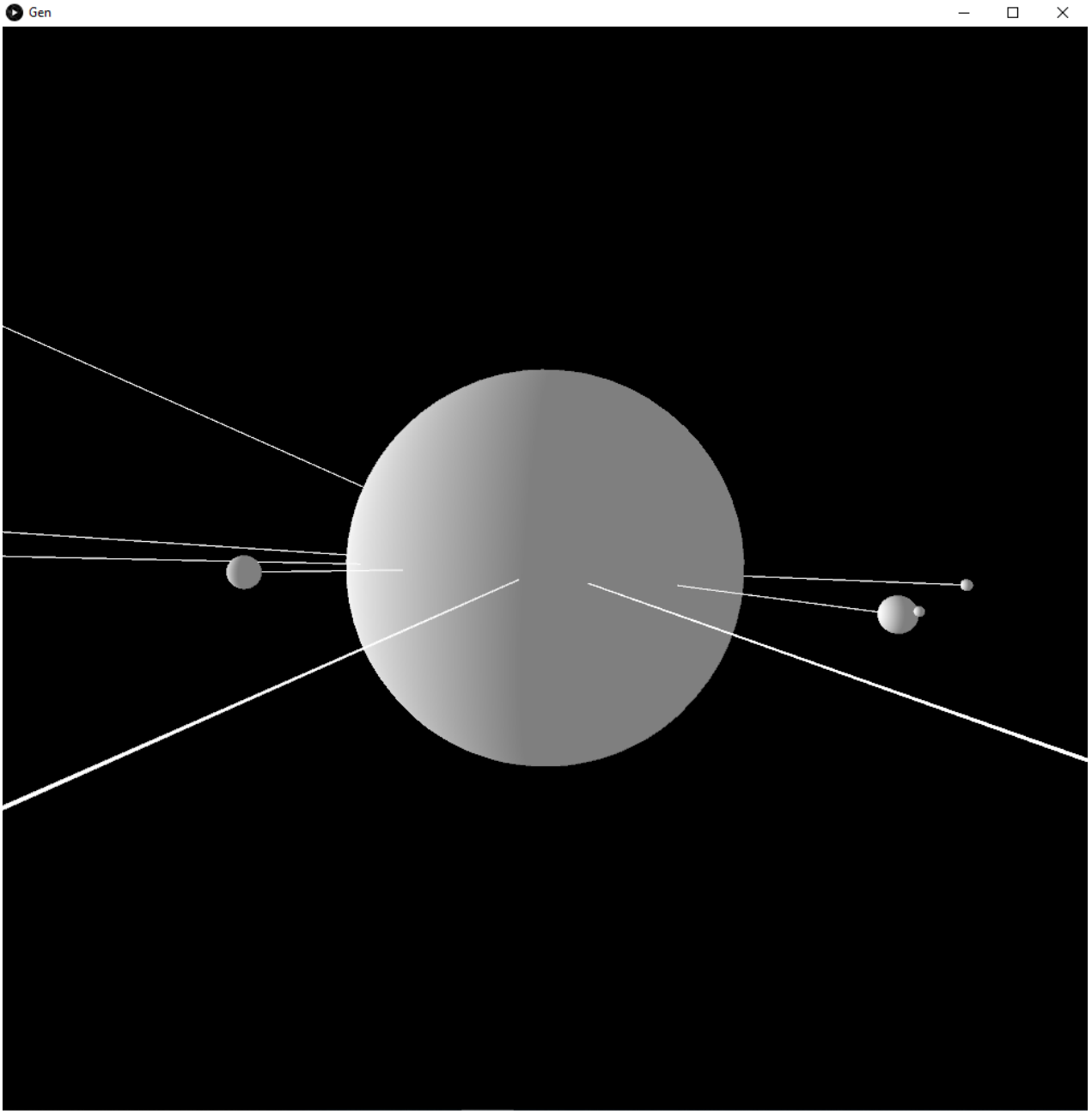


Fig 16. A close up of the model and its nearby planets including the Earth and its Moon

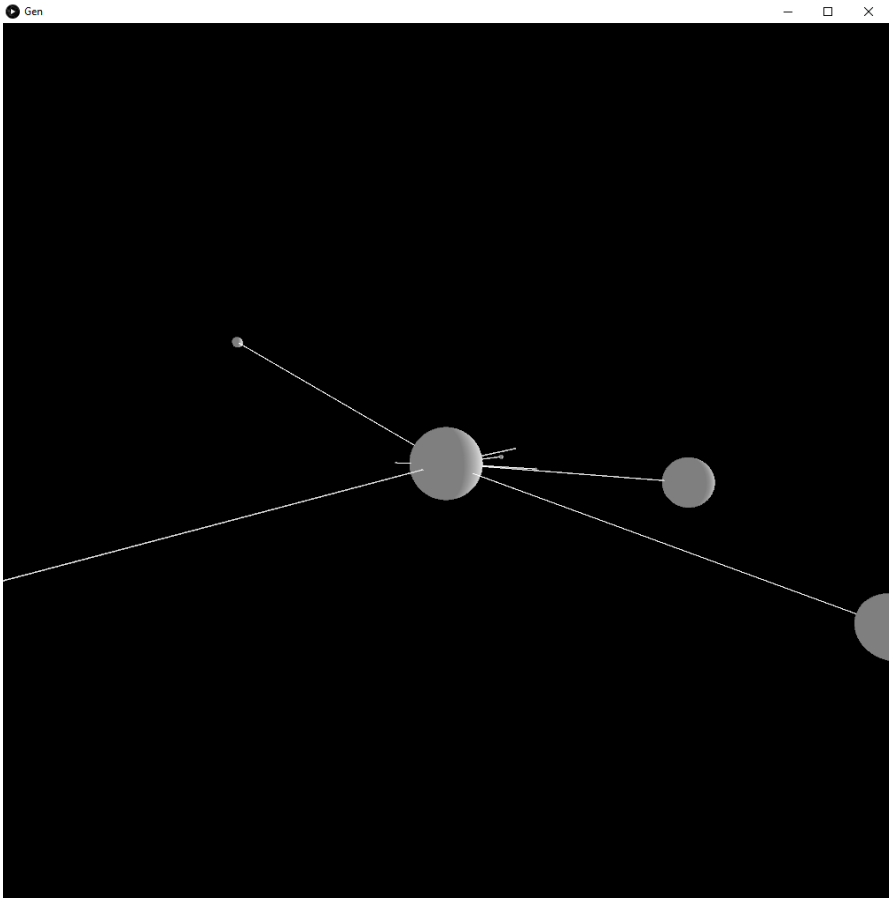


Fig 17. The model some distance away from the centre and displaying most of the objects

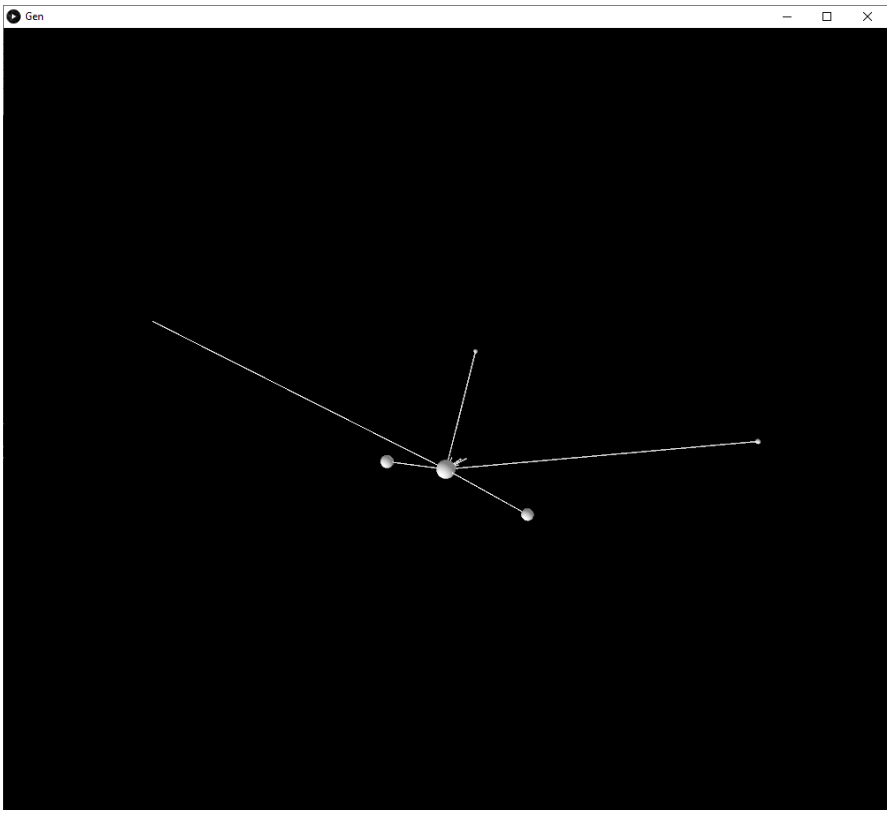


Fig 18. The model from a large distance away and displaying all the generated objects.

6.5 Increment 3

In increment 3 the development is planned to reach a stage at which it could be considered operational from a user's perspective. It focuses on improving the visibility of the model, highlighting the objects' differences and establishing a pleasant aesthetic look as well as beginning the UI development. Specifically these include the implementation of the object trajectories, the improvement of the lighting function, the rendering of object textures and the establishment of a base of code for UI functionality.

6.5.1 Object Trajectories and Names

Just observing the animation from increment 2, it can be seen that, from a user's perspective it is difficult to identify precisely how objects move in space. Due to the black background and the drawn vector lines the objects appear disorganised - moving without any sort of structure.

For the objects to be recognised as moving along the same plane, a trajectory line is implemented. The aligned trajectories can help the user identify the exact way that the planets move in the real world. This can be easily achieved through the creation of the circular objects within the 3D space and translating them such that the objects move along their circumference. A function path() is coded within the planet class such that those conditions are met.

```
void path(){

 pushMatrix();

   stroke(255);
   noFill();
   rotate(PI/2, v.x, v.y, v.z);
   ellipse(0,0, v.x*2, v.x*2);

popMatrix();
}
```

This function, when called for each planet in the draw() method, will create an ellipse, rotate it along the vector axis on which the planet object exists by an PI/2 or 90 degrees.

As this project is partially aimed at producing a product that can be used for educational purposes, it is important to add text into the animation to name the existing objects within it. By default, all of the names of the planets should be shown upon the execution of the code. Therefore, a line creating a text object at the slightly altered position of the planet is created.

As the name can, depending on the camera zoom, be obstructing other objects - a UI option for the user to be able to hide them must exist. As this will be required during UI development a function hideName() is included into the planet class. To this effect, a trigger boolean variable showName is added to the class which will act as a checking mechanism to see if the user has or has not already clicked the button. If the boolean value is “true”, the function will change the planet name to a single space character, which will appear invisible within the animation.

```
void hideName(){

  String savedName = Pname;
  String hidden = " ";

  if(showName == true){
    showName = false;
    Pname = hidden;
  }
  else {
    showName = true;
    Pname = savedName;
  }

}
```

Similar functions are written for planets’ trajectories for further implementation in UI.
The drawn lines to the coordinates of the positional vectors have mostly been removed with the exception of pluto. Pluto, having the smallest relative size, appears invisible when observing the whole model. It is presumed to be beneficial for the user to track where Pluto is without having to adjust the camera angle.

6.5.2 Texture and lighting

In order to achieve realistic effect, the planet objects created must appear to the user having the same look as the real word planets. This can be using the Processing's texturing methods. This is performed via the importation of the image file through the project's data folder and its stretch along the object surface. However, this requires a few changes within the planet object class. Firstly, not every created object is able to have an image applied to it. If an object is to be rendered it must be of the type PShape. It is possible to create and upload any shape into processing, however as the project primarily involves sphere, a preset SPHERE PShape is configured. A PShape object can have a PImage object associated with it. PImage is an uploaded image that is "stretched" onto the PShape. Thus, the planet class constructors are altered to contain PShape and PImage variables. The constructors now require a PImage as a parameter for generation and use it to access the PShape.texture() method.

```
  planet(String name, float r, float d, float s, float o,float x, float y, float
z, PImage texture){
    v = new PVector(x, y, z);
    radius = r;
    angle = random(TWO_PI);
    distance = d;
    speed = s;
    Pname = name;
    v.set(v.x * d + o, v.y, v.z);

    noStroke();
    noFill();
    globe = createShape(SPHERE, radius);
    globe.setTexture(texture);
  }
```

The PImage objects must be declared and initiated with images in the generation file.
The images for these objects are obtained from Planet Pixel Emporium webpage (*Planet Texture Map Collection*, n.d.) webpage. PShape object is generated within the constructor with a preset SPHERE shape and using the radius parameter.

The lighting is also improved. Previously a default lights() function was deployed which provides ambient light on all objects. A lighting() is written which defines a combination of point and ambient light methods within processing.

```
void lighting() {

  pointLight(255, 255, 255, 0, 0, 0);
  lightFalloff(0, 0, 0.00001);
  ambientLight(255, 255, 255);

}
```

The pointLight() defines a coordinate at which the light source is located. The light produced is reduced with distance covered as dictated by the lightFalloff() processing method. The ambient light is added to the entire animation to illuminate slightly the sides of objects not covered by the point light.

6.5.3 UI conflict

It has been previously assumed that the UI and several of its components such as buttons and sliders can be implemented aside the Peasycam imported library. The assumption was that these UI elements would move with the movement of the camera by the user. It is, in fact, the case that the supported elements of the user interface such as buttons are classes as objects within the graphics window similar to the other generated objects. This means that, while the camera is being manipulated by the user, these elements of the interface remain local to the specified point of where they are drawn. This is a significant problem as it means that a new camera class must be created and configured in order to incorporate camera movement as well as the UI.

6.5.4 Model at the end of increment 3

It is at this point that the development of the artefact ceases. Due to the UI conflict, the estimated time of completion has been extended beyond the allotted time resources for this project.

At the end of increment 3, the designed artefact has fully achieved the base model complete with all the planets, their realistic movement, the texture rendering for all objects and started to implement a few UX improvements such as the lighting and UI code

components. Some snapshots of the artefact at this stage are displayed below.

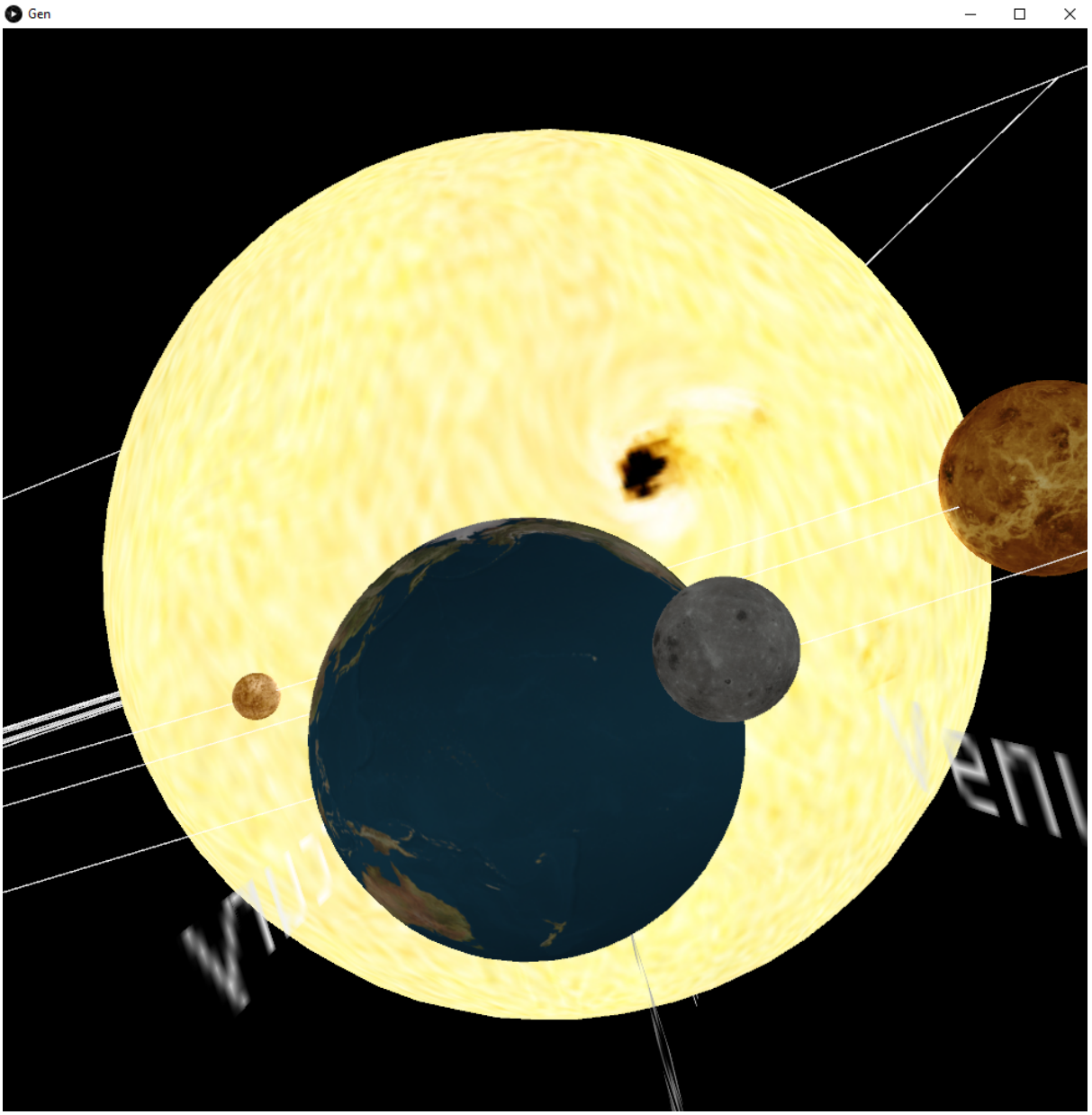


Fig 19. The model screenshot close up

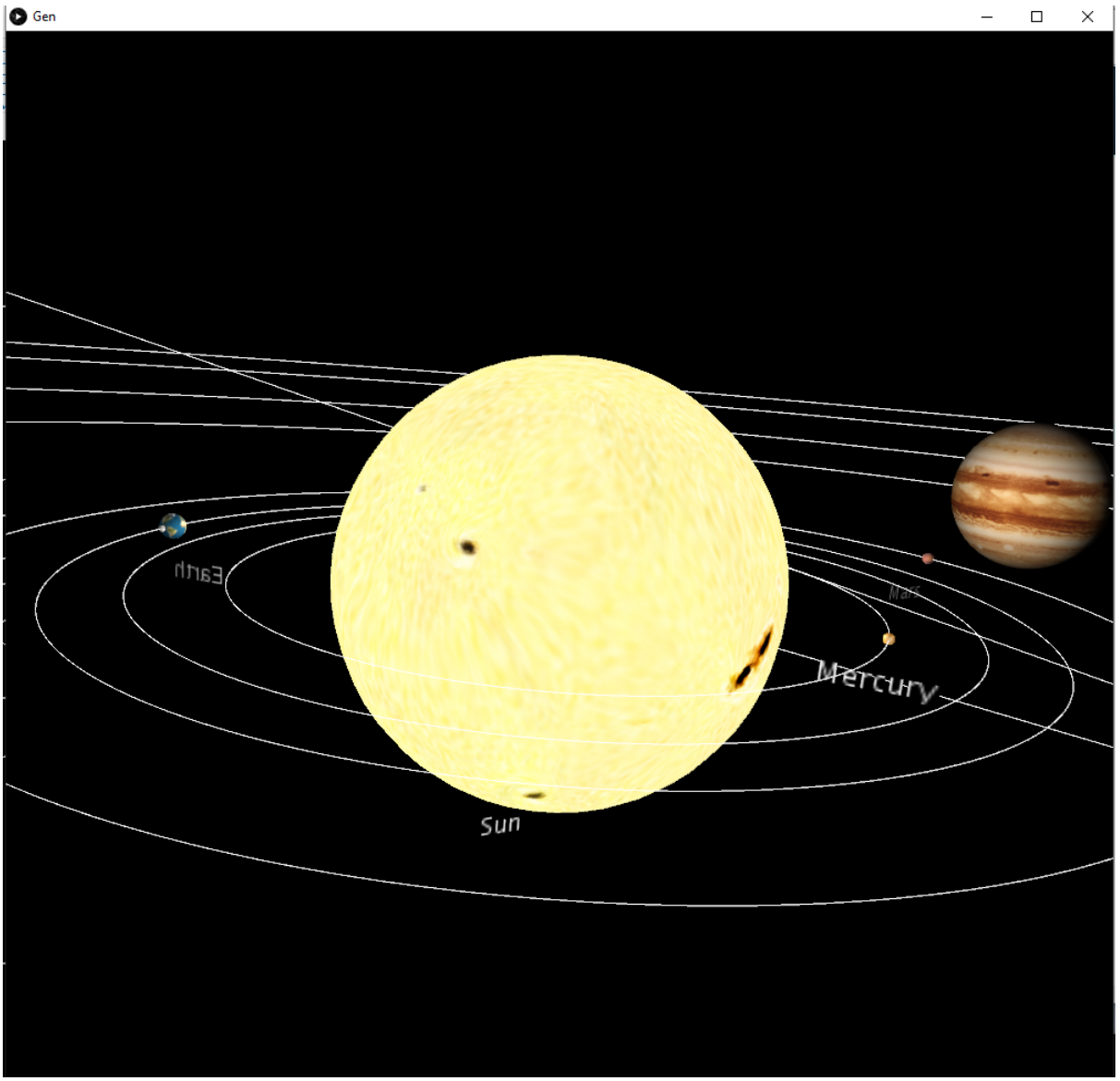


Fig 20. The model screenshot within the first few orbits

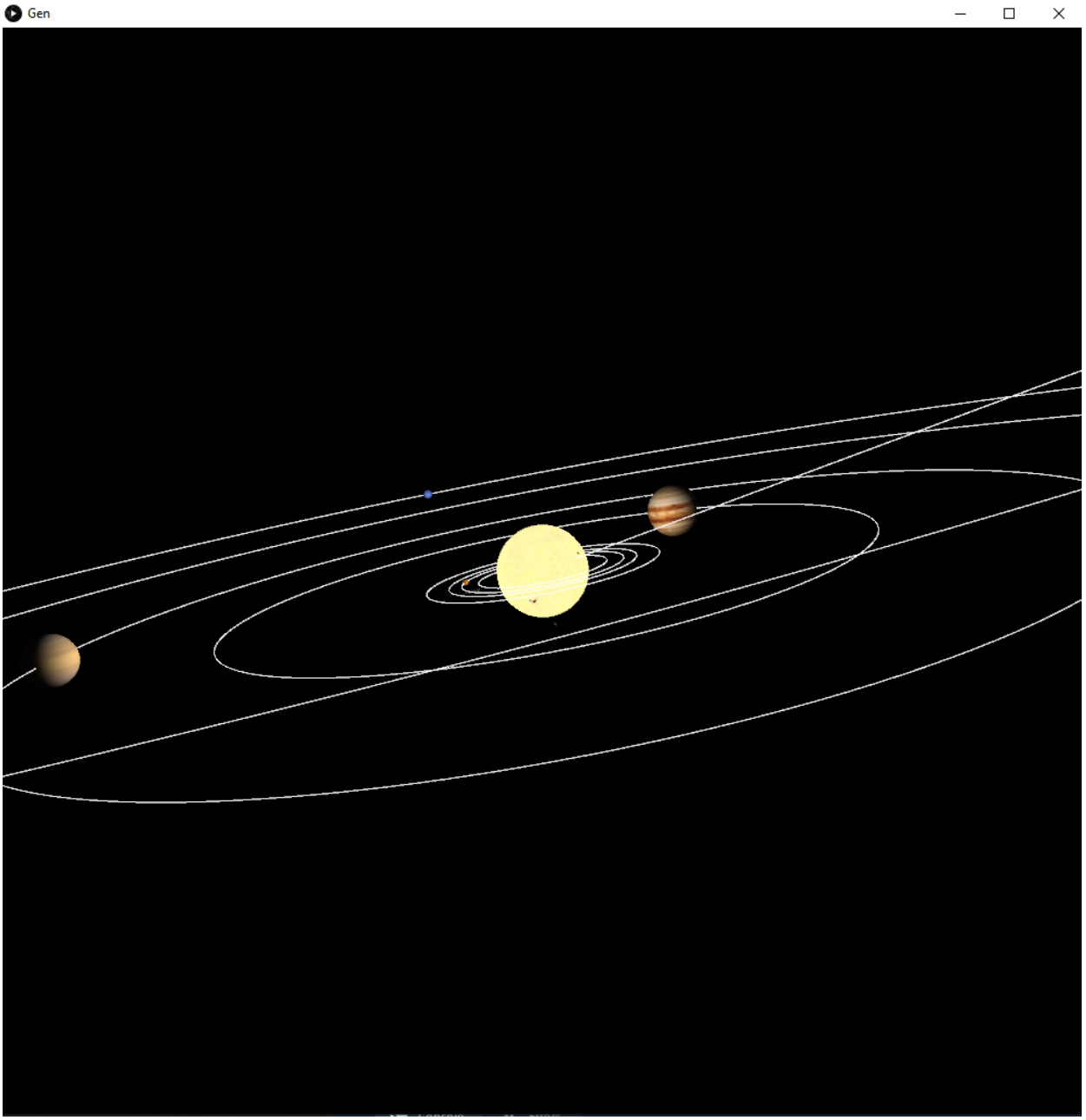


Fig 21. The model screenshot at some distance away

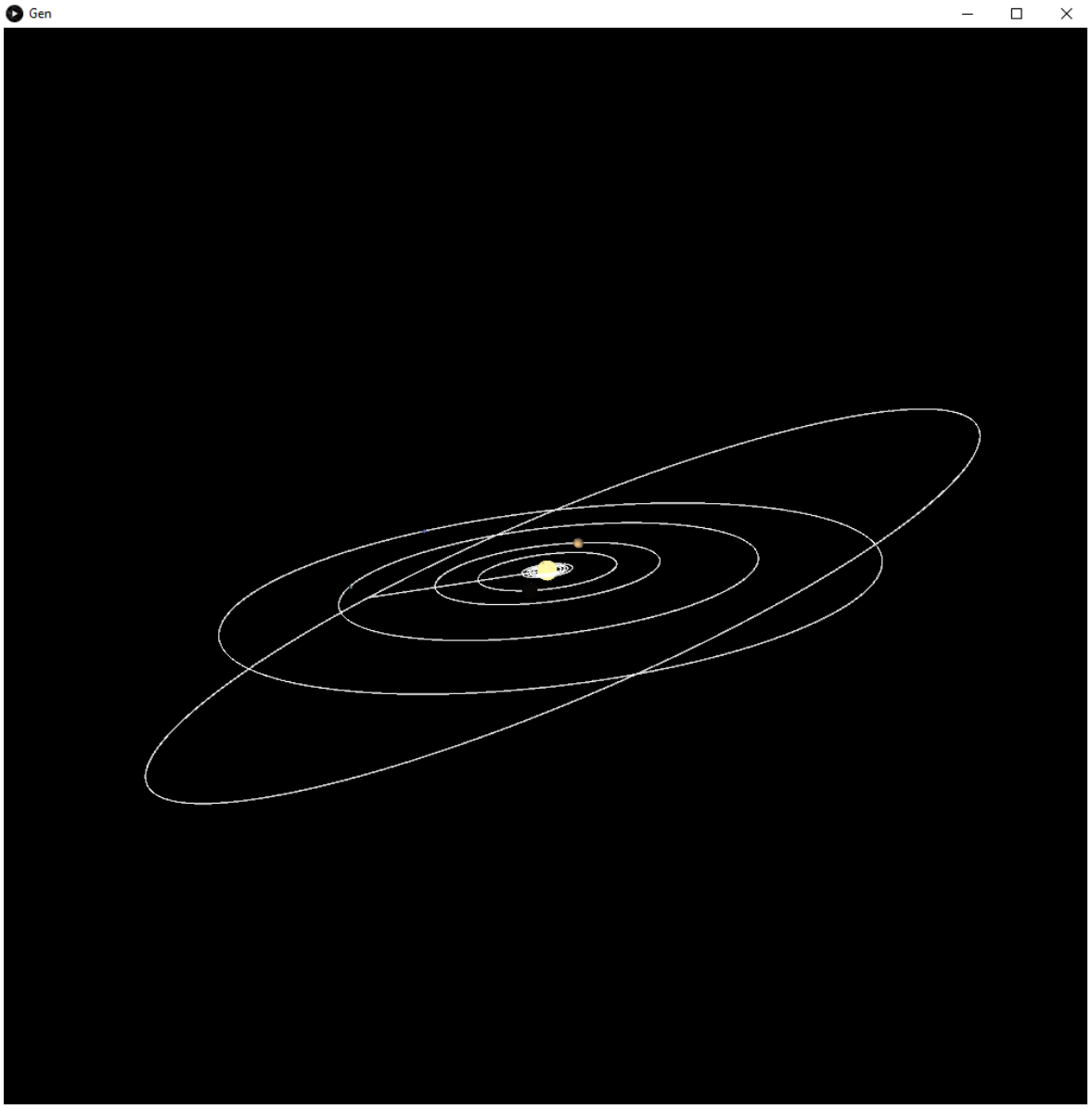


Fig 22. The whole model at a large distance away

7. Testing

In accordance with the chosen methodology for the development of the artefact, most of the testing is performed within each of the incremental sprints. This usually involves direct execution of the code and its consecutive analysis during the animation or the lack thereof. As such, most debugging and verification must be performed and completed before the development moves on to the next increment. Each increment presumes the complete functional delivery of the goals set for the previous increment.

If the written program fails to execute, depending on the error given by the console, a few debug functions are written to output data into the console prior to the execute fail. Such functions are often methods for retrieving variable data of objects such as their vector coordinates or displaying all elements of arrays. Some of such functions remain in the code and can be inspected in the

Once the code is able to be executed and the model is displayed in the sketch window. Some user tests are run to ensure the correct operation of the interactive viewer. Some such tests for increment 3 were outlined in the table 7 shown below. These tests are elicited through the increment goals described in the design chapter.

Test Description	User Input	System Expected Output	Status	Comments
User loads the program and attempts to see the entire model via manipulating the camera	Once the program loaded, the user zooms out using the mouse wheel down input	The camera moves further away backwards increasing the amount of objects visible on the screen	Pass	
The user attempts to change perspective of the camera by rotating the angle form which the solar system is observed	The user clicks and drags the mouse left / right for left and right rotation respectively and up / down for a pitch motion	The camera pans, rotating around the vertical and horizontal axii relative to user input.	Pass	
User is able to see all names of visible objects on the screen	No user input required	All object names appear clearly from any perspective of the camera.	Fail	Names are affected by the light and therefore become invisible for the planets on the outer orbits.
The user attempts to change the centre of camera to look at a particular object	Double click on the desired object	Camera uses camera lookat() command to pan to the clicked object	Fail	Not implemented
The user attempts to reset the camera	Pressing a reset button at the top left of the screen	The camera reset to the default angle set when the program is launched	Fail	Not implemented due to PeasyCam conflict with UI integration

Table 7. User Tests for increment 3

8. Evaluation

In this chapter the overall outcomes of the project are assessed. The final iteration of the artefact is evaluated against the aims and objectives established at the beginning of this document as well as the requirements elicited throughout it. The project's pitfalls will be outlined and potential causes are investigated.

8.1 Evaluation against requirements

To evaluate the final artefact, a list of functional and non-functional requirements will be obtained from section 4. These requirements are what the design and implementation of the program was aimed at satisfying. Therefore, they are the primary way of defining the success of the project.

Tables 7 and 8 below show the requirements identified previously and assign a status of completion (achieved / not achieved / partially achieved) and briefly describe the reason behind the status evaluation.

Functional Requirement	Priority	Status	Justification
Complete implementation of all significant objects	1	Achieved	The model generated contains all 8 planets, Earth's Moon, Pluto and the Sun
Object accurate movement	1	Achieved	The model emulates the orbirting of all major objects around the sun in the correct direction with the accurate relative speed. This also includes the Moon.
Realistic Dimensions & Distances	1	Achieved	All objects generated with parameters based on real-world figures are converted to suit the program values taking into account the spheres' radii.

Object Trajectories	2	Achieved	The generated planets travel along their orbital paths indicated by a drawn circle outline.
Intuitive camera movement	2	Achieved	By scrolling the mouse wheel the user is able to zoom in and out. Clicking and dragging the mouse allows for rotation and tilting of the camera.
Planet Identifiers	2	Partially achieved	A name text exists below each planet (excluding the moon), however it is affected by the lighting and thus becomes invisible at larger distances from the sun.
Basic Model options	3	Not achieved	There is no UI present when the program is executed
Advanced Camera Movement	3	Not achieved	No extra features beyond simple drag and zoom have been implemented. The camera focus can not be changed to a different object other than the sun.

Table 8. Assessment of functional requirement delivery

Non-Functional Requirement	Priority	Status	Justification
Model must be realistic	1	Achieved	Most object movement has been replicated and functions in a realistic way
The model should be clearly visible	1	Partially achieved	No other objects besides the planets have been implemented. The model opted for the lack of background to preserve clarity. However, due to the lack of options to increase the objects sizes, objects like Pluto remain hard to observe - reducing the clarity of the model.
The system must be accessible and intuitive to use	1	Achieved	The model offers only the simplest intuitive movements like drag and zoom
The user should be able to interact with the system via operating the camera perspective and UI	2	Not achieved	Due to the lack of UI no extra features have been implemented to satisfy this requirement.
The UI must be simplistic and non-intrusive to the displayed model by default	2	Not achieved	No UI is present.
The visualisation should be able to be easily embedded into a website	3	Partially achieved	Due to the existence of p5.js - a processing library that allows the execution of sketches in JavaScript - it is possible to convert Processing code to a JavaScript language recognised by webpages. However, it can not be considered an easy way of integration as translation may take some time.

Table 9. Assessment of functional requirement delivery

8.2 Issues impeding full completion

One of the issues that caused the unfulfillment of the UI requirement was an oversight in increment development. As was mentioned earlier in section 6.5.3, the decision to integrate the Peasy cam library in increment 1 was based on the assumption that limited extra effort was required for the implementation of UI alongside it. This is an assumption that was regarded as true for the majority of the system development and was only recognised as untrue in increment 3. It was previously estimated that UI development would require a relatively small amount of time dedication. Upon further investigation into this issue, it appeared that the PeasyCam imported library is not able to be easily integrated with other UI modules.

This indicates a fundamental problem within the methodology employed or its application. This issue could have been prevented via a more thorough design stage process of the increments. It is important to verify the assumptions made prior to the integration of a third party library. It is also vital to carefully consider the estimated time required for the development of specific features. Taking the inexperience constraint into account, the aforementioned aspects of development must have been thoroughly researched before their application.

Due to the time constraints, a decision was made to prioritise the completion of this document and thus the cease of artefact development.

8.3 Summary

Overall, the final artefact satisfies most of the highest priority and some of the medium priority requirements. None of the low priority requirements have been met. This means that the program functions at the base level, however, it does not provide the high level features that would ultimately distinguish it from the other existing solutions. While this design does feature high precision relative positioning of the objects, the lack of UI implementation reduces the product's versatility.

The project code and Instructions for execectuion may be found at **https://github.com/UP935745/3D-Solar-System-Animation**

9. Conclusion

In this conclusion chapter, some final thoughts are given on the outcome of the project, the process of research and development as well as the success in achieving its aims.

Firstly, it should be said that the project was a partial success in terms of its aims. As is stated in section 1, its aim is the delivery of an interactive solar system visualisation that features close approximation to the behaviour of the real world. The final build of the code does satisfy the latter clause, however, does not present advanced interactivity. The objectives listed in section one were certainly of help and were followed, however the ineffective management of time and the oversight in the design impeded the completion of the artefact.

The approach to the development of the program was believed to be a structural success. Considering the type of program being developed and the several constraints such as the learning of 3D graphics technology, the incremental-agile methodology seems to have aided the implementation. The sprints of development between the increments have allowed for steady progress, while the incremental design helped define short term goals and break down the development into smaller, manageable problems. The application of this approach, however, was not perfect. The design decisions made within the development of each increment should have been given more thought with regards to their later integration with future iterations. An example of such fault is the UI conflict issue described in sections 6.5.3 and 8.2.

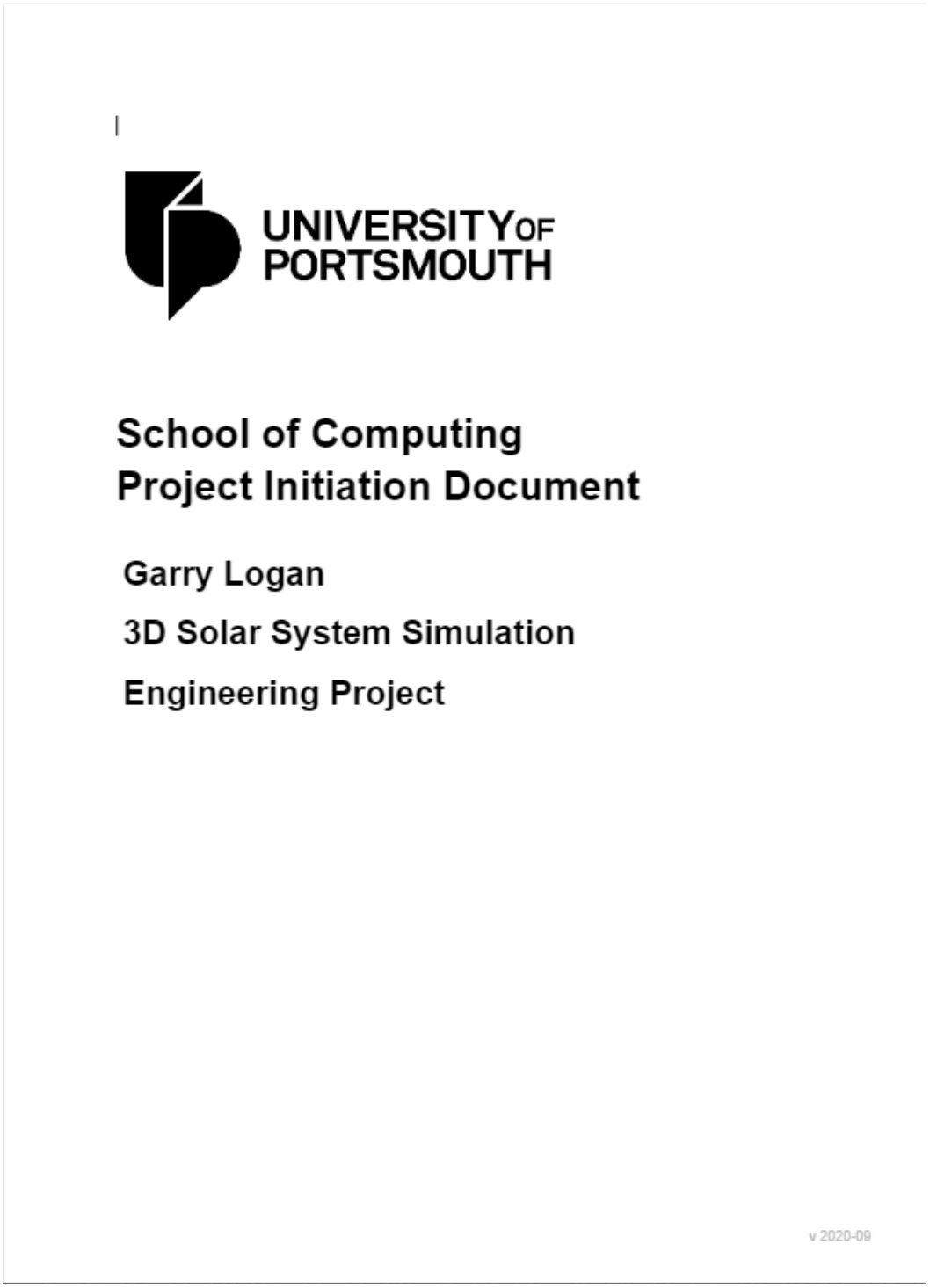
Critically evaluating the final build, the project was not able to breach the functionality level required for it to distinguish itself enough from similar projects discussed in section 2. If this project was to be carried out again, the UI implementation should be assumed to require almost as much development time as the implementation of the base mode. Given more time and the delivery of all intended components, the artefact could become a fully operational product.

10. References

1. *TheSkyLive - Your Guide to the Solar System and the Night Sky*. (n.d.). Theskylive.com. Retrieved May 5, 2022, from http://theskylive.com
2. *Similarweb*. (2022). Similarweb. https://www.similarweb.com/website/theskylive.com/#overview
3. *Solar System Scope*. (2011). Solar System Scope. https://www.solarsystemscope.com/
4. connorgaskell. (2021, April 28). *connorgaskell/solar-j3d*. GitHub. https://github.com/connorgaskell/solar-j3d
5. Gregg Tavares. (February 9th, 2012). *WebGL Fundamentals.* Retrieved May 5, 2022, from https://www.html5rocks.com/en/tutorials/webgl/webgl_fundamentals/
6. Friesen (2008, September 18). *Open source Java projects: Java Binding for OpenGL (JOGL)*. InfoWorld. https://www.infoworld.com/article/2077906/open-source-java-projects-java-binding-for-opengl-jogl.html
7. Selman. (2002). *Java 3D programming : [a guide to key concepts and effective techniques]*.
8. *3D Vector Plotter | Academo.org - Free, interactive, education.* (n.d.). Academo.org. https://academo.org/demos/3d-vector-plotter/
9. *NASA Jet Propulsion Laboratory (JPL) - Space Mission and Science News, Videos and Images*. (2019). NASA Jet Propulsion Laboratory (JPL). https://www.jpl.nasa.gov/
10. *P3D*. (n.d.). Processing. Retrieved May 5, 2022, from https://processing.org/tutorials/p3d
11. NASA. (2018). *Planetary Fact Sheet*. Nasa.gov. https://nssdc.gsfc.nasa.gov/planetary/factsheet/
12. *peasycam*. (n.d.). Mrfeinberg.com. Retrieved May 5, 2022, from https://mrfeinberg.com/peasycam/#about
13. *Planet Texture Map Collection*. (n.d.). Planetpixelemporium.com. Retrieved May 5, 2022, from http://planetpixelemporium.com/planets.htmlx

11. Appendices

Appendix A - PID



UNIVERSITY OF PORTSMOUTH

School of Computing
Project Initiation Document

Garry Logan

3D Solar System Simulation

Engineering Project

v 2020-09

1. Basic details

Student name:	Garry Logan
Draft project title:	3D Solar system simulation
Course:	Meng Computer Science
Project supervisor:	Dr Dalin Zhou
Client organisation:	
Client contact name:	

2. Degree suitability

In this project I aim to learn how programming code can be adapted for visual contexts and representation. I believe this is a suitable project for my course because it would allow me to achieve further depth in knowledge with regards to computer science. Particularly, in the areas of graphics programming and modeling.

3. Outline of the project environment and problem to be solved

The problem in question is: utilisation of object oriented programming in a graphical environment to simulate a real world scenario.

3D simulation is of great interest to me and I wish to apply my previously learned java programming skills to this task.Throughout the project, I will learn the basics of 3D graphics programming, the basics that could then be expanded upon in the future within the industry.

4. Project aim and objectives

My main aim is to figure out how a 3D code modeler operates and apply my knowledge of object-oriented programming to it, in order to simulate motion with created assets.
Some of my objectives that will help me achieve this are:

- Research into 3D processing software that could be used in this project
- Understanding the main premises of the chosen software / library
- Understanding how graphics units interact with Object-oriented programming
- Production of a testing environment where the project can be built
- Programming of objects relating to this scenario

- Testing of created objects in two dimensional environment ● Adaptation of 2D environment into 3D

5. Project deliverables

- Simulation Design Specification
- Progress / Issue Log
- The simulation code
- User Manual for use and adaptation into similar environments
- JavaDoc file
- GitHub page

6. Project constraints

The main constraint of this project is accuracy. It would be difficult to completely and accurately simulate the exact conditions of space with my current knowledge of the science and the programming skill. There are likely to be small discrepancies or inconsistencies.

Additionally the simulation code, by its nature, will only work with the specific language and libraries that it is written in. Hence, it would require additional efforts in order to port it to other systems.

7. Project approach

The first stage of my project will involve the research of technologies that could be used in simulation of 3D space environments compatible with java. Upon assertion of possible software I will assess their utility for this project, based on which I will make the final decision to use that technology. I, then, plan to read the documentation for the chosen software / library and begin the coding.

I will elicit my specifications through a review of the real world space environment. I will assess the size rations between the planets' sizes and respective distances from the sun. As well as their travel path trajectories.

As for the production of code I plan on using versioning control via GitHub to section-off functioning code and branching to implement changes. This will be done via the iterative and incremental approach whereby successive iterations will be improved upon and further functionality installed (such as 2D to 3D) port.

Software incompatible with home computer	Severe	Low	A new graphics engine has to be researched and used	Software / library is displaying errors on import
GitHub versioning error reversing progress to last version	Severe	Moderate	Always careful handling of commits and branching	Missing code

11. Project plan

Below is the google sheet file with the current project plan:

https://docs.google.com/spreadsheets/d/1GMQUqyynWWcSKYtj-FRkjGeXlm4hG078FOvbfA_5wd4/edit?usp=sharing

12. Legal, ethical, professional, social issues (mandatory)

There are almost no legal or ethical issues regarding this project as it is not concerned with handling personal data or generally impacting other people.

From the professional standpoint it is worth mentioning that this project or variation of this project is likely to have been already undertaken several times before. Therefore, the issue of plagiarism is a major factor in the scope of this project.
The aim of this project is, in part, to elevate my understanding of the field and thus referencing existing code should be avoided.
There is little reason for concern regarding the security of this project because, as stated previously, it does involve handling of sensitive information.
All students must complete the ethics review form at https://sums.soc.port.ac.uk/ethics at this time. Has your supervisor (and the FEC representative, if required) seen and approved your ethics form? ***Remember*** *– this is obligatory and must be completed* ***now****.*
The school's FEC representatives are Dr Matt Dennis and Dr Philip Scott.

8. Literature review plan

- Java3D programming book by Daniel Selman http://index-of.co.uk/OReilly/3d%20Programming.pdf
- Introduction to Processing - https://processing.org/tutorials/overview
- Foundations of 3D graphics programming using JOGL https://www.researchgate.net/publication/220690441_Foundations_of_3D_graphics_programming_Using_JOGL_and_Java3D
- Java graphics - http://www.cs.um.edu.mt/~sspi3/AGraphicsAbstractionWithJAVA.pdf

9. Facilities and resources

No extra facilities need to be provided.

10. Log of risks

What risks will you encounter when doing your project? What backup plans do you have if identified things go wrong?
What is your plan for reviewing risks? Remember that risk probabilities, and hence priorities, will change over the course of the project, so this section should be maintained. Use a table like below.

Description	Impact	Likelihood	Mitigation	First indicator
Code does not function correctly on other systems	Severe	Moderate	Launch versions of code on other systems to confirm functionality	Failure to compile when code is launched on other computers
The scope of the project is underestimated	Severe	Moderate	Focus on the development of at least the 2D version	The estimated coding periods for each part are extended several times.
Wi-ficonnection inconsistency hindering progress	Low	Likely	Download required documentation for future reference	-

Appendix B - Ethics Form

Certificate of Ethics Review

Project title: 3D Solar Simulation using object oriented programming

Name:	Garry Logan	**User ID**:	935745	**Application date**:	02/03/2022 16:52:14	**ER Number**:	TETHIC-2022-102754

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are **Haythem Nakkas, David Williams**

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **School of Computing**
What is your primary role at the University?: **Undergraduate Student**
What is the name of the member of staff who is responsible for supervising your project?: **Dr Dalin Zhou**
Is the study likely to involve human subjects (observation) or participants?: No
Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No
Are there risks of significant damage to physical and/or ecological environmental features?: No
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No
Does the project involve animals in any way?: No
Could the research outputs potentially be harmful to third parties?: No
Could your research/artefact be adapted and be misused?: No
Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: No

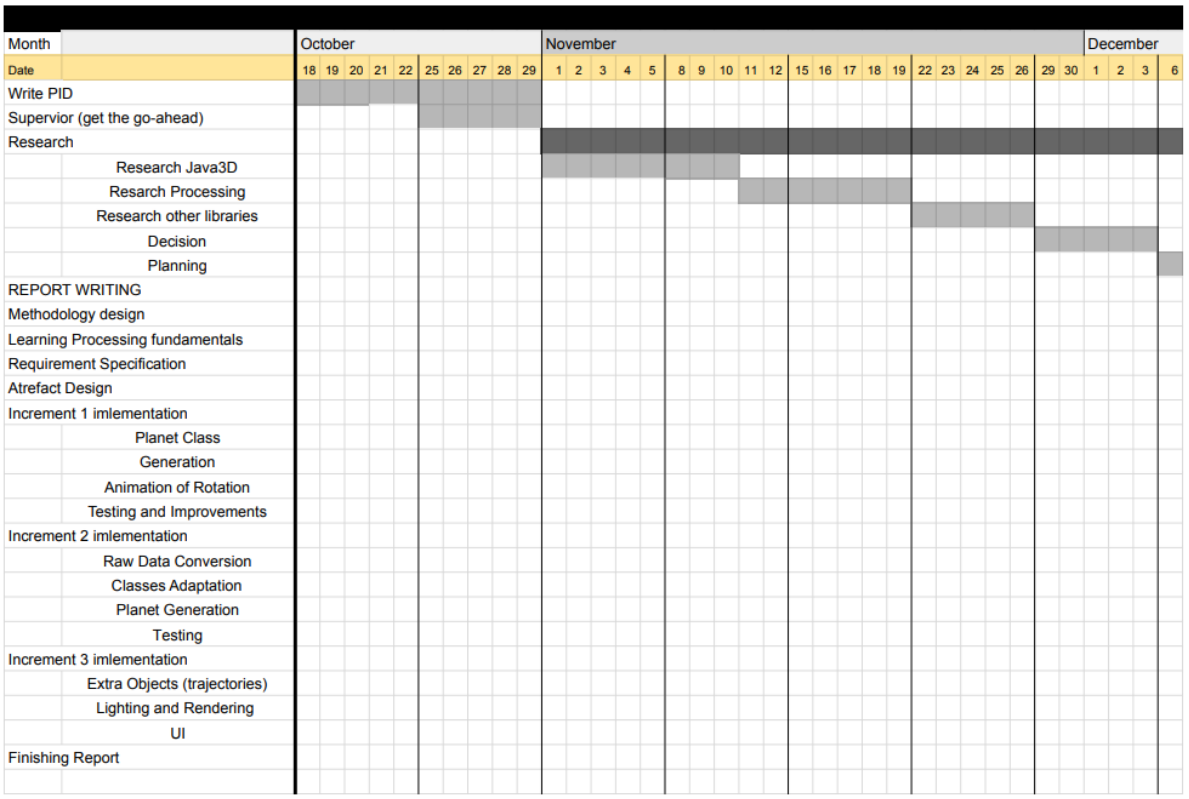
I confirm that I have considered the implications for data collection and use, taking into consideration legal requirements (UK GDPR, Data Protection Act 2018 etc)
I confirm that I have considered the impact of this work and and taken any reasonable action to mitigate potential misuse of the project outputs
I confirm that I will act ethically and honestly throughout this project

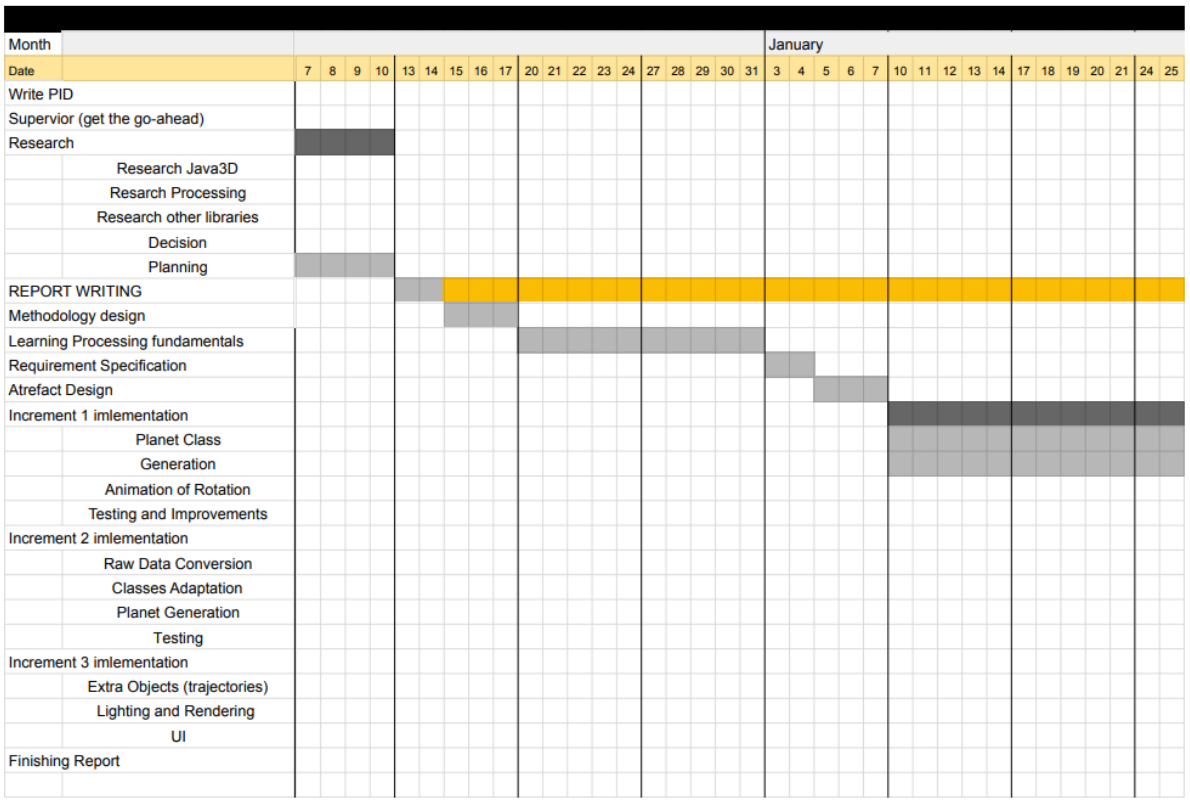
Supervisor Review

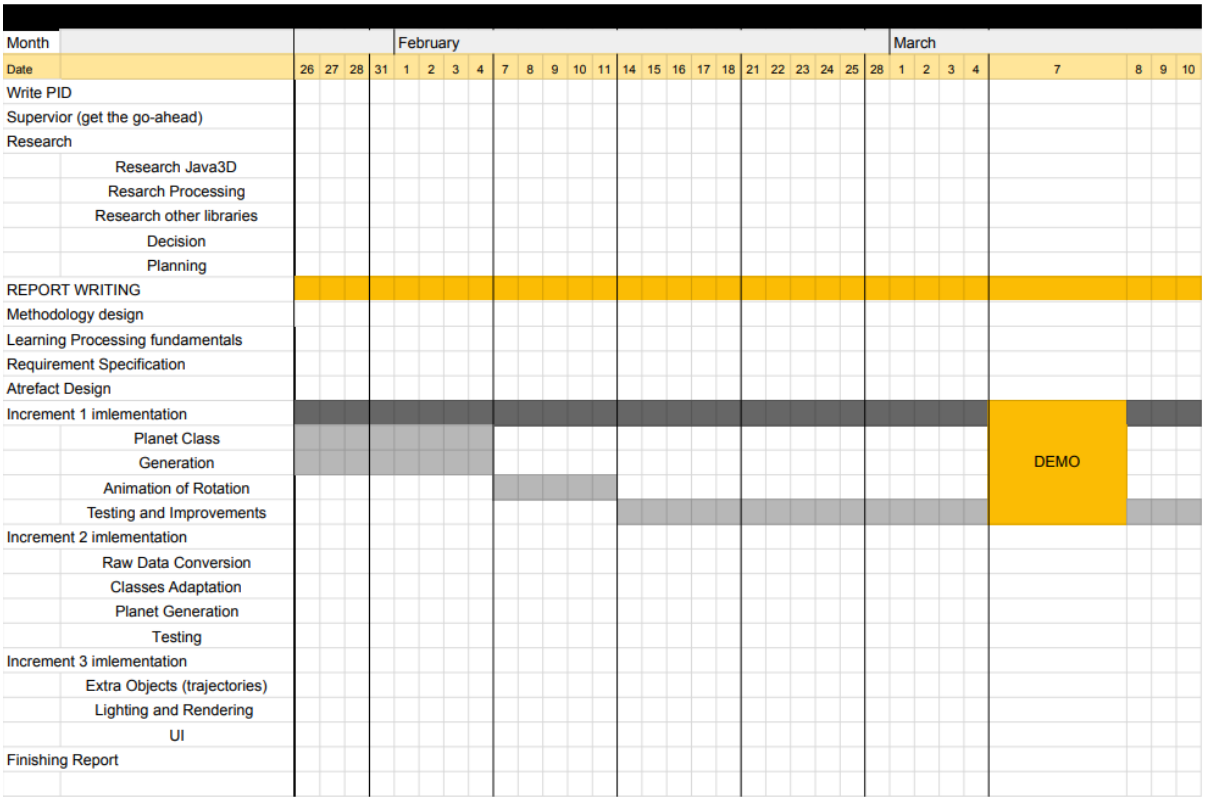
As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

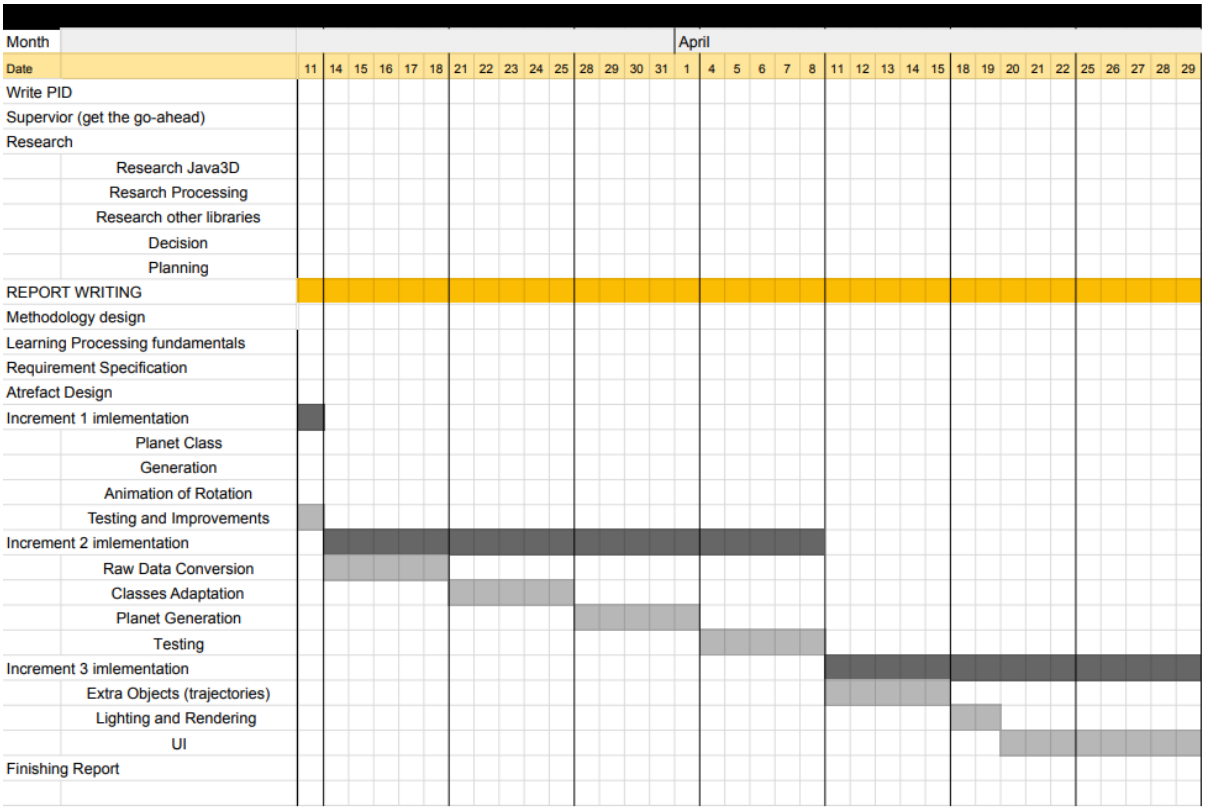
Supervisor's signature: Date:

Appendix C - Gantt Chart






Month
February
March
Date
26 27 28 31 1 2 3 4 7 8 9 10 11 14 15 16 17 18 21 22 23 24 25 28 1 2 3 4 7 8 9 10
Write PID
Supervior (get the go-ahead)
Research
Research Java3D
Resarch Processing
Research other libraries
Decision
Planning
REPORT WRITING
Methodology design
Learning Processing fundamentals
Requirement Specification
Atrefact Design
Increment 1 imlementation
Planet Class
Generation
Animation of Rotation
Testing and Improvements
DEMO
Increment 2 imlementation
Raw Data Conversion
Classes Adaptation
Planet Generation
Testing
Increment 3 imlementation
Extra Objects (trajectories)
Lighting and Rendering
UI
Finishing Report


Month
April
Date
11 14 15 16 17 18 21 22 23 24 25 28 29 30 31 1 4 5 6 7 8 11 12 13 14 15 18 19 20 21 22 25 26 27 28 29
Write PID
Supervior (get the go-ahead)
Research
Research Java3D
Resarch Processing
Research other libraries
Decision
Planning
REPORT WRITING
Methodology design
Learning Processing fundamentals
Requirement Specification
Atrefact Design
Increment 1 imlementation
Planet Class
Generation
Animation of Rotation
Testing and Improvements
Increment 2 imlementation
Raw Data Conversion
Classes Adaptation
Planet Generation
Testing
Increment 3 imlementation
Extra Objects (trajectories)
Lighting and Rendering
UI
Finishing Report

Month		May						
Date		30	1	2	3	4	5	6
Write PID								
Supervior (get the go-ahead)								
Research								
	Research Java3D							
	Resarch Processing							
	Research other libraries							
	Decision							
	Planning							
REPORT WRITING								
Methodology design								
Learning Processing fundamentals								
Requirement Specification								
Atrefact Design								
Increment 1 imlementation								
	Planet Class							
	Generation							
	Animation of Rotation							
	Testing and Improvements							
Increment 2 imlementation								
	Raw Data Conversion							
	Classes Adaptation							
	Planet Generation							
	Testing							
Increment 3 imlementation								
	Extra Objects (trajectories)							
	Lighting and Rendering						DEADLINE	
	UI							
Finishing Report								