Orbital Mechanics Simulator

Lucius Kwok (VC1B)
<lk@felttip.com>
Supervisor: Brian Papa
<bpapa@icloud.com>

Project Purpose

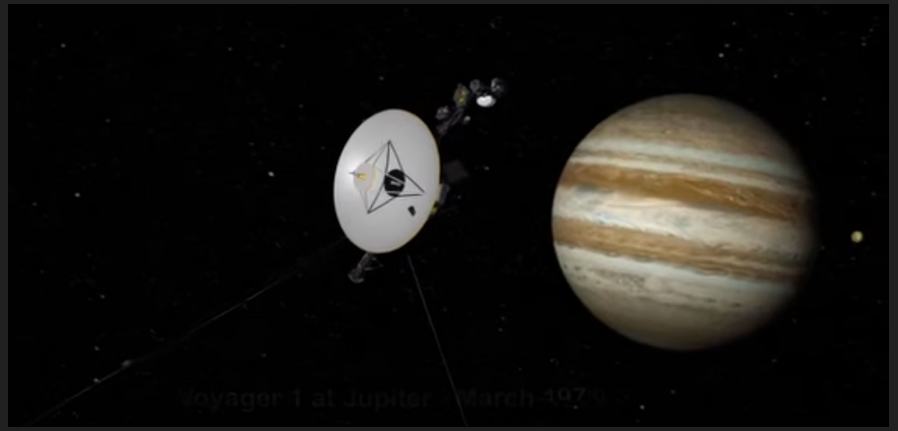
This 3D spaceflight simulator will be part of a web page where users will learn about the orbital mechanics topic within the topic of spaceflight. It will allow students and people without a background in physics to understand the physics required to go from an Earth orbit to a Moon orbit. It allows users to plan out and execute a series of velocity change maneuvers to go from an Earth orbit to a Moon orbit based on an approximation of realistic orbits of celestial bodies called patched conics, which uses multiple spheres of influence to represent discrete gravity wells for each body. It will also present some historical context in the form of the Apollo Moon missions, the Voyager missions, and more recent robotic missions to the Moon.

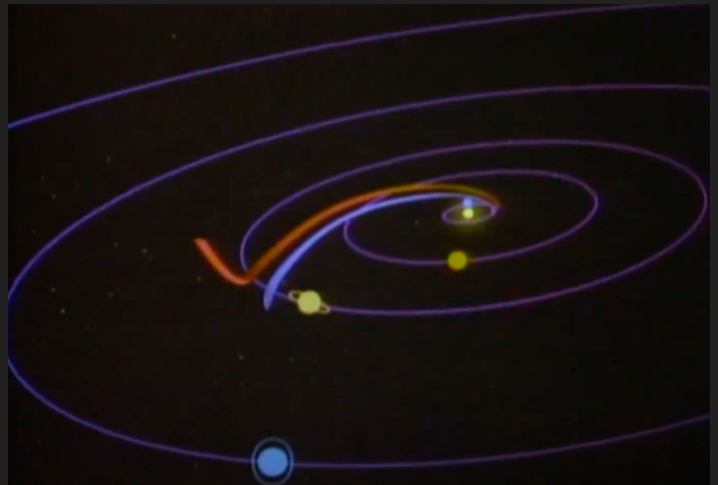
Target Audience

This experience is targeted at users who are:

- College-level students interested in intro-level physics and astronomy, but not necessarily having a physics background
- Casual gamers who enjoy playing spaceflight simulators such as Kerbal Space Program
- Learners who watch documentaries about historic spaceflight missions to the Moon and beyond, such as those about Apollo or Voyager missions

Concept Art: Voyager missions



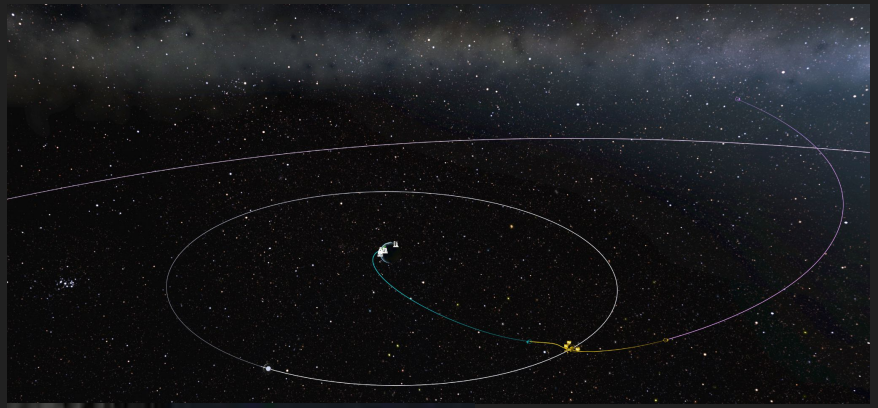


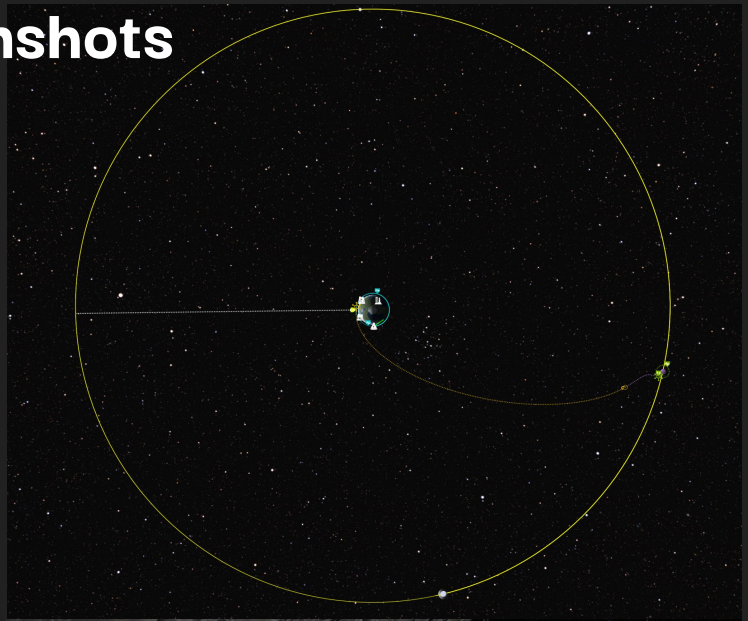
Reference Example:

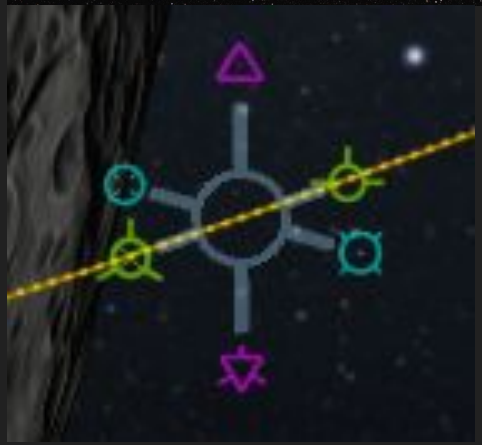
KSP (version 1):

- Started in 2010 as a project by one person, HarvesteR (Felipe Falanghe)
- Uses Unity engine
- Continuously developed by a team until about 2021, to focus on KSP 2
- Changes in ownership and direction led to failure of KSP 2 in the market and closure of the game studio that was developing it
- Vacuum left by implosion of KSP 2 means there is a market for a new spaceflight simulator to take its place.

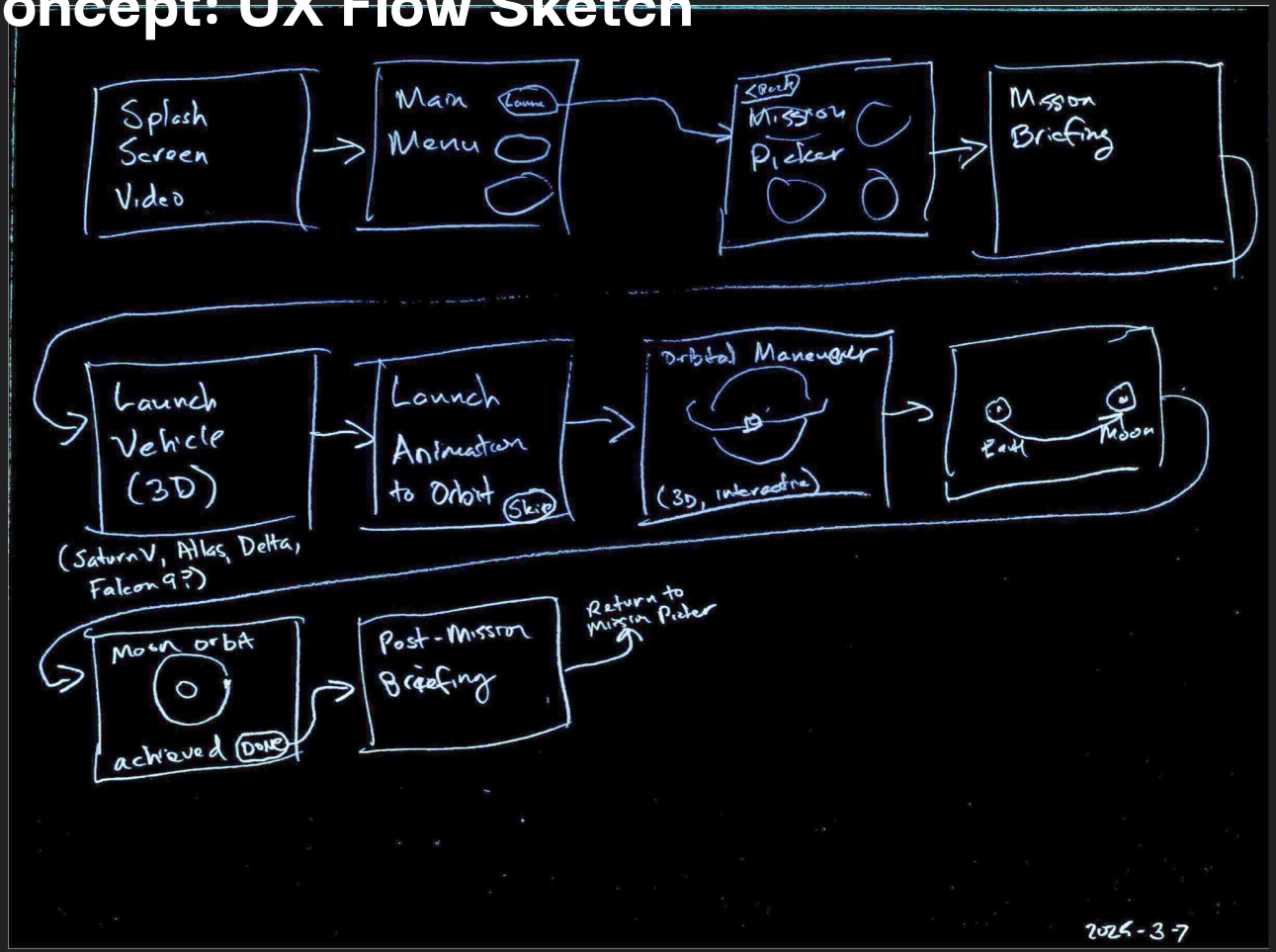
Reference Example: KSP Screenshots



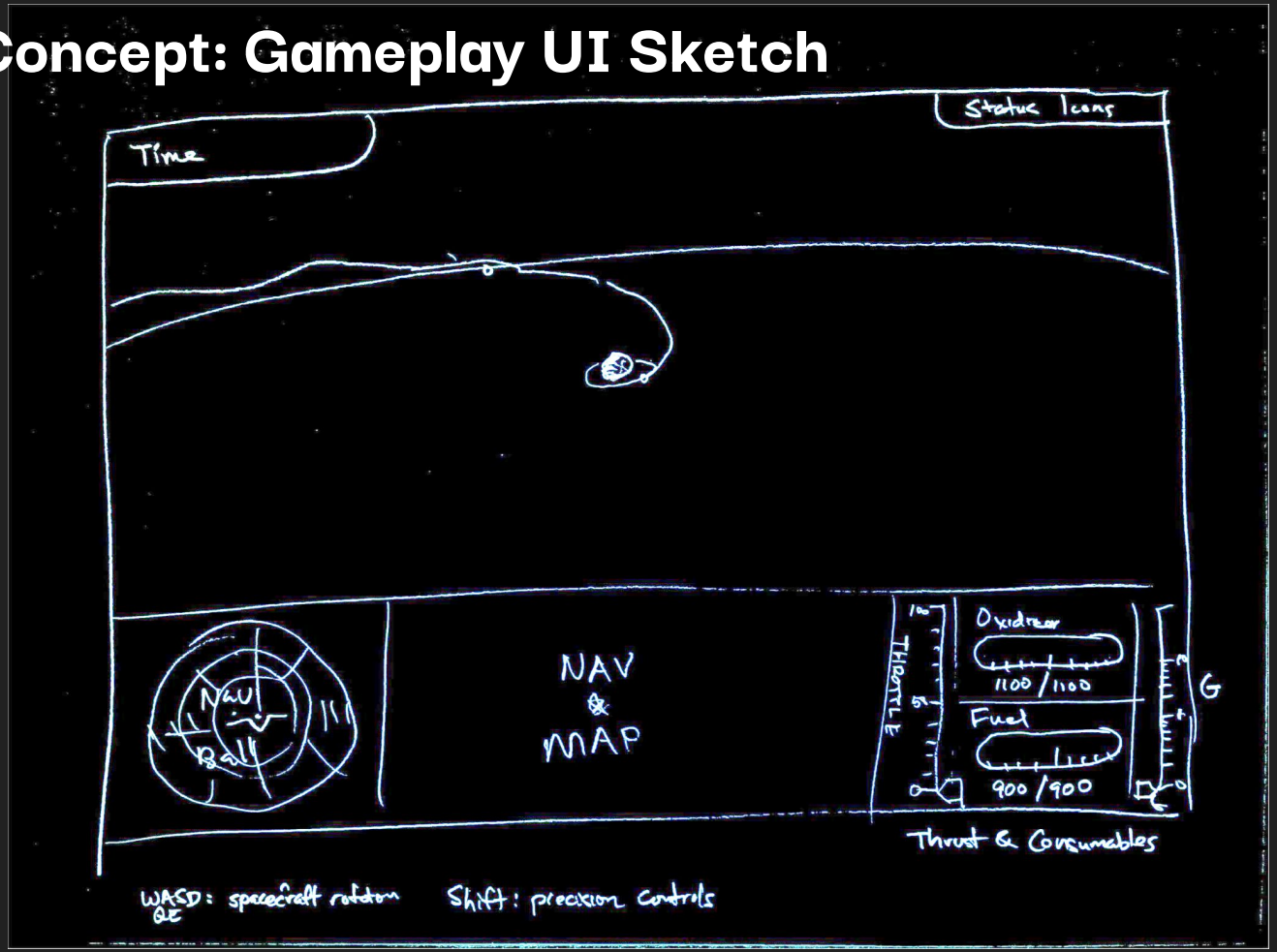




Initial Concept: UX Flow Sketch



Initial Concept: Gameplay UI Sketch

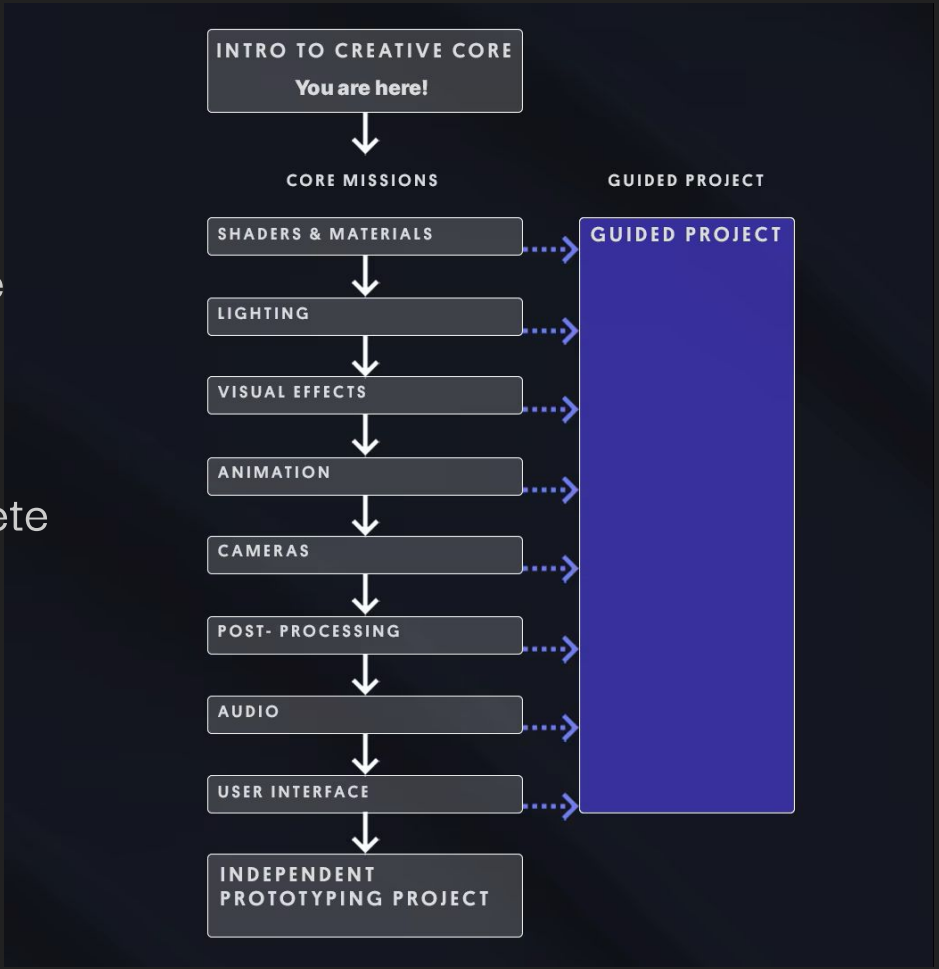


Challenges & Roadblocks: Scope

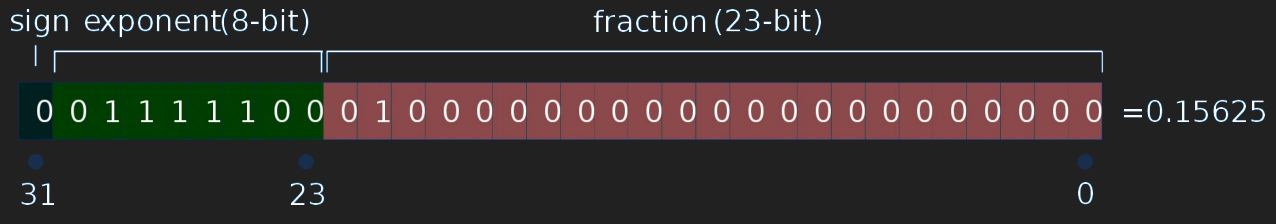
- KSP was developed over 10 years, I have only 15 weeks.
- I have experience only with making productivity apps, not with making games or using Unity engine.
- Game vs. Simulation: I want to lean towards a simulation with realistic physics, but still making an engaging and enjoyable experience.
- Technical challenges such as 32-bit float precision and 3D calculations complicate the project.

Challenges & Roadblocks: Unity

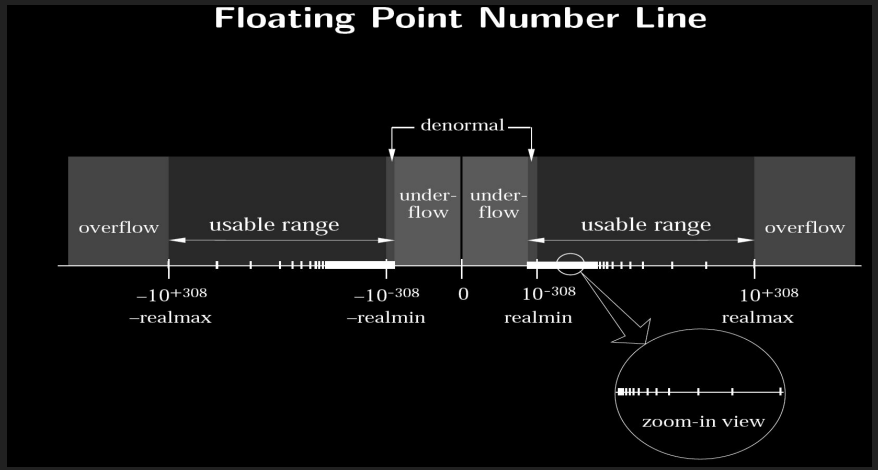
- Unity is a very complex game engine with a steep learning curve.
- I have almost no experience in it.
- Spent 3 weeks / 50 hours to complete Creative Core pathway tutorials.
- Will need to spend as much time to complete the Junior Programmer pathway tutorials.



Technical Challenge: 32-bit Floating Point Accuracy



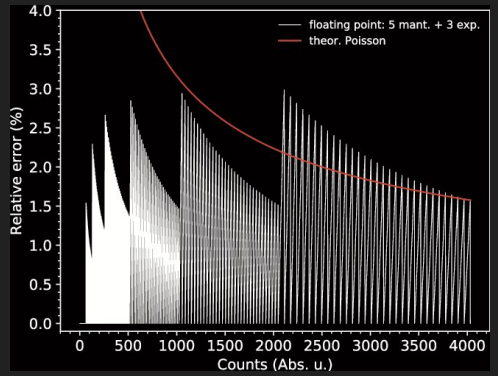
There is problem where the precision of a floating-point number decreases as the value is farther from zero. When representing a location in space, the positions of points far away from the origin of the world become less precise.



Source: https://courses.grainger.illinois.edu/cs357/fa2023/notes/ref-4-fp.html

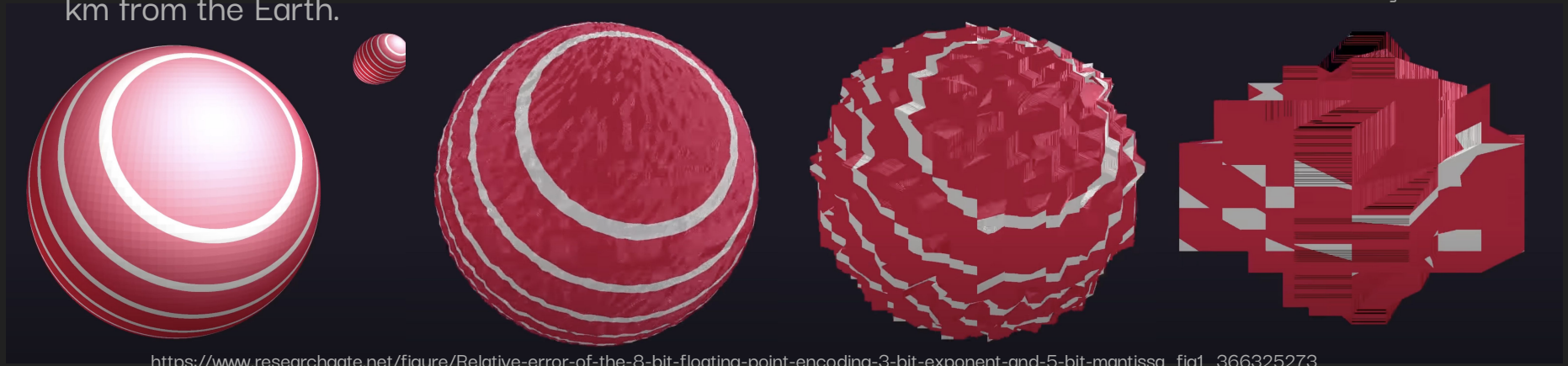
Technical Challenge: 32-bit Floating Point Accuracy

As objects move farther away from the origin of worldspace, their geometries develop glitches due to precision errors. For 32-bit floats, at a distance of 10km, the accuracy is only 1 to 10 meters. For reference, the Earth's diameter is 12,756 km, and the Moon is 382,500 km from the Earth.



Left: Rounding error in 8-bit floating point representation

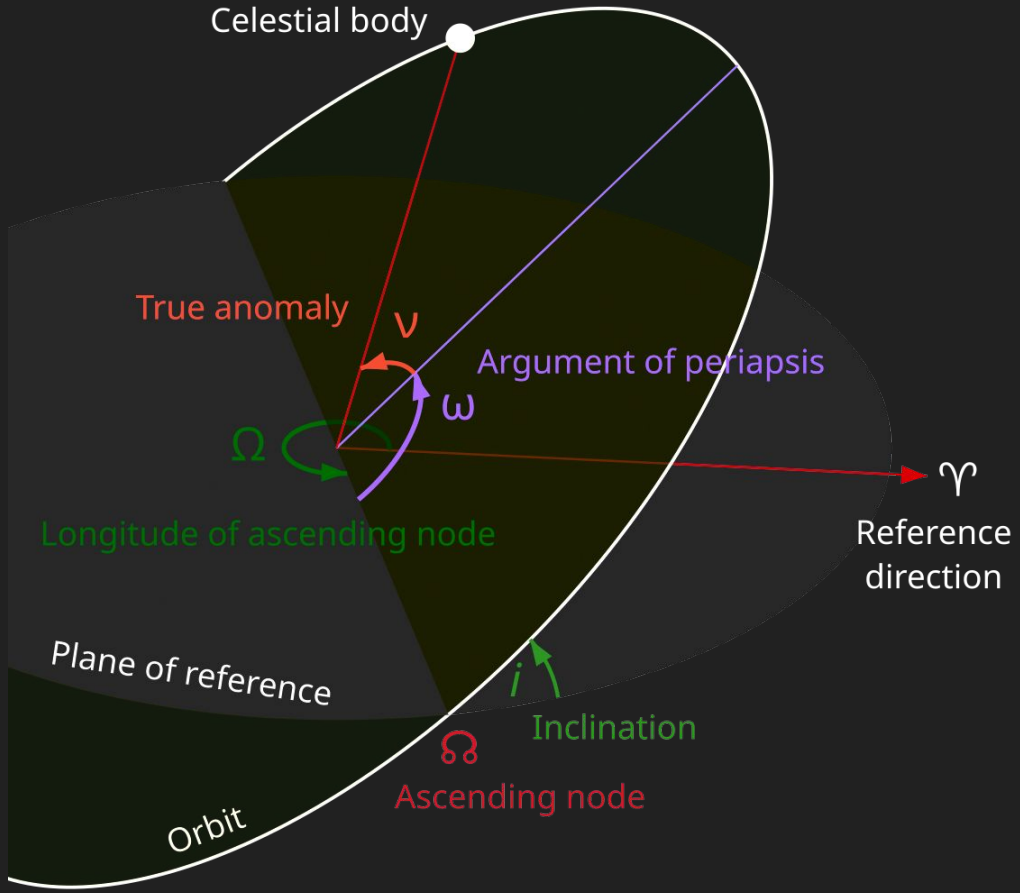
Below: Effects on 3D geometry as a sphere moves away from the origin.



https://www.researchgate.net/figure/Relative-error-of-the-8-bit-floating-point-encoding-3-bit-exponent-and-5-bit-mantissa_fig1_366325273

https://www.youtube.com/watch?v=wGhBjMcY2YQ

Technical Challenge: Orbital Elements in 3D Space



Orbital elements are parameters that define an orbit in 3D space.

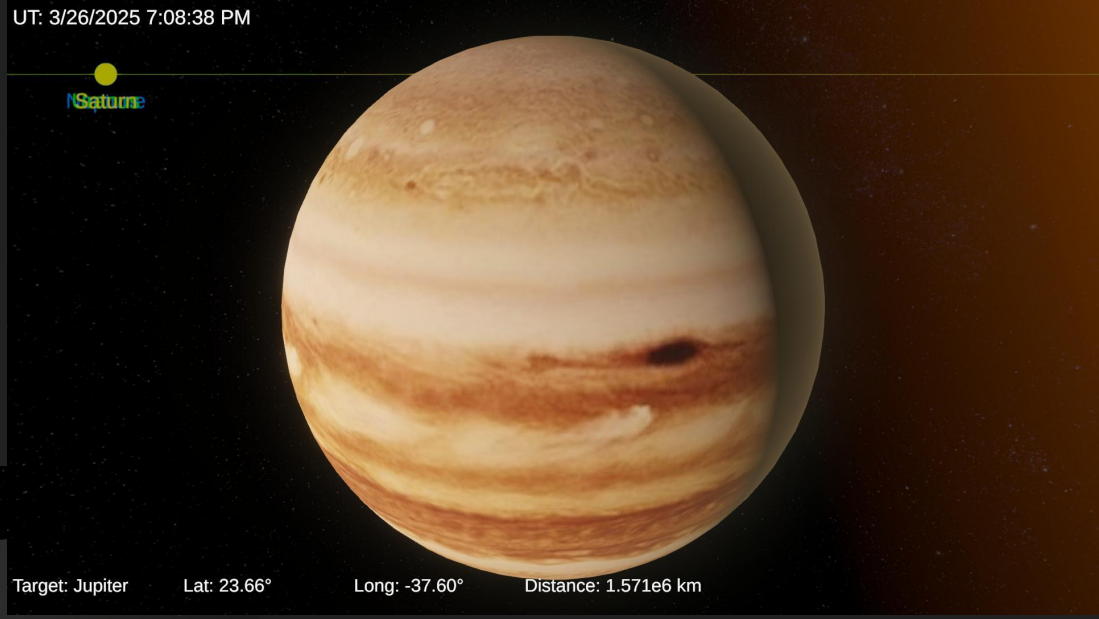
- Eccentricity (*e*)
- Semi-major axis (a)
- Inclination (*i*)
- Longitude of the ascending node (Ω)
- Argument of periapsis (ω)
- Orbital period (*P*)

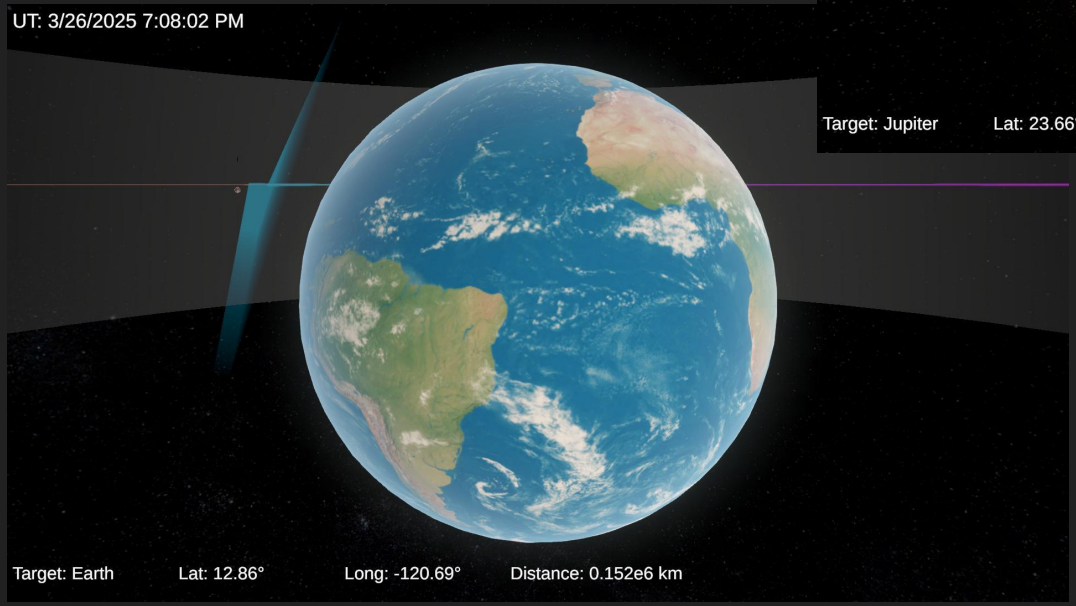
Understanding and applying physics in a 3D space is much more complicated than in 2D.

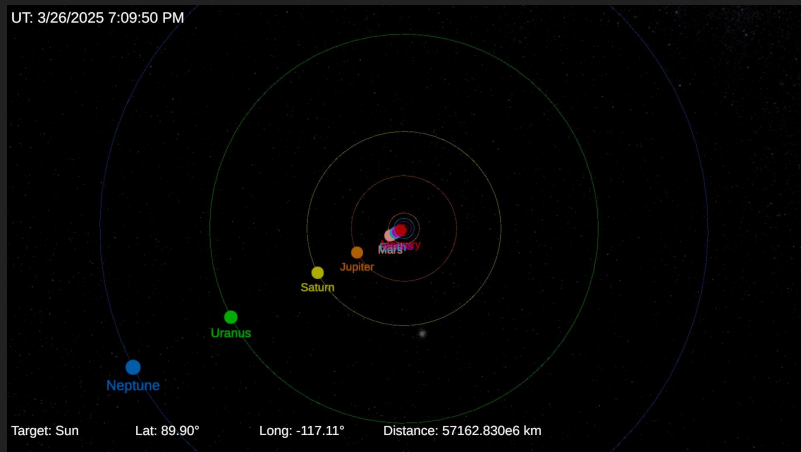
https://en.wikipedia.org/wiki/Orbital_elements

Midterm Project Status

- Running in Unity
- Displays planet shaders & orbit lines
- Camera control using mouse
- Switch between target planets







Midterm: Overcoming Challenges

- Scope: will need to focus on one core gameplay element at first (orbital mechanics and maneuver nodes).
- Realism: accurate simulation will be the goal, but some things may need to be simplified.
- Technical challenge: 32-bit float. Several solutions are possible, including a floating origin and multiple rendering contexts.
- Technical challenge: 3D vs 2D. May need to constrain orbits to 2D plane that exist in a 3D space.

Project Schedule (initial plan)

Week 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Project planning & research. Set up project repo & issue tracker.															
		Identify users, use cases, and proposed solutions. Sketch UI/UX.													
				Develop prototype/demo. User testing, presentation.					Further testing			Final testing			
					Refine and build out UI and features based on user and presentation feedback.										
								Triage features and issues to decide on MVP							
											Feature lock & focus on fixing issues.				
													Prepare final presentation & submission		

Forms of Media & Digital Representation

Multimedia Computing Learning Outcomes

This project uses various forms of media in their digital representations.

Images: Start out as photos, drawings, renderings, digital paintings, texture maps, etc. Can be digitized in raster or vector formats. Raster formats: PNG (lossless), EXR (high dynamic range), JPEG (lossy). Vector format: SVG.

Video: Can come from animations, digital renderings, digitized 70mm Apollo film footage, etc. Container format: MPEG, QuickTime.

Audio: Can be in the form of music, sound effects, foley, UI feedback, etc. Formats: AAC, MP3, WAV, FLAC, Ogg Vorbis, Midi.

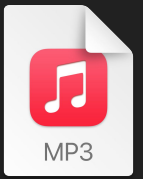
Encoding & Compression

Multimedia Computing Learning Outcomes

This project uses encoding and compression to reduce file sizes and bandwidth.

Images: typically in PNG format for lossless compression and JPEG for lossy compression. Considerations include: image size, pixel resolution (for raster), image quality for lossy formats.

Video: typically in the MPEG4 (.m4v) container format, and use the H.264 video encoding format. Considerations include: pixel resolution of each frame, whether to use high dynamic range, and the target bitrate of the file or stream.



Audio: formats used include: AAC, MP3, WAV, and Ogg Vorbis. Music and most other audio is typically in MP3 format, while short sound effects can be in the uncompressed WAV format.

Trade-Offs in Media Formats

Multimedia Computing Learning Outcomes

This project makes various trade-offs between file size, quality, and processing time.

Images: can constitute a large part of a project's size, depending on the number of files. Beyond their direct use, they are also used in 3D rendering for texture and effects. Trade-offs include image size, pixel resolution, and image quality for lossy compression vs. file size.

Video: contributes the most to project's file size, with higher pixel resolution, using HDR, and higher quality resulting in larger file sizes and higher bitrates. Processing time is also a concern on lower-end hardware, where a 4K HDR video might not be playable.

Audio: Trade-offs include: audio quality vs. file size, stereo vs. mono, and dynamic range. Processing time is generally not a concern with modern hardware.

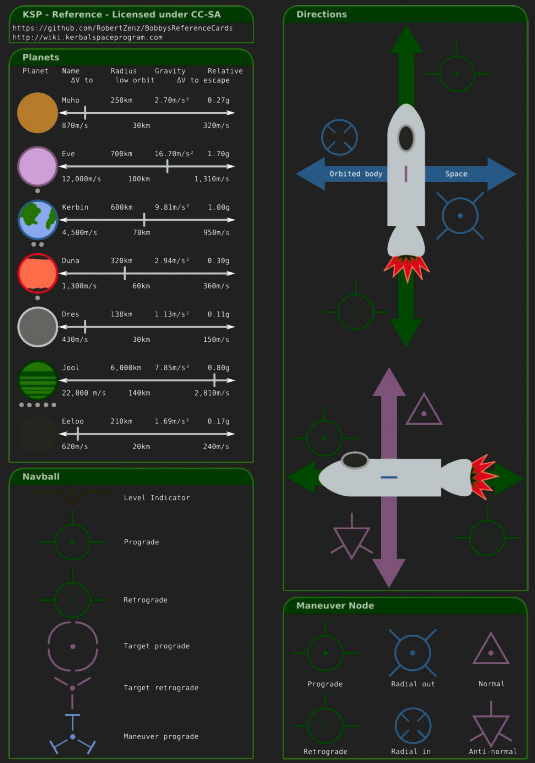
Trade-Offs in Simulations

Multimedia Computing Learning Outcomes

Another kind of trade-off happens in simulation programs, that between accuracy of the simulation vs. the real world.

A simulation cannot model everything with absolute fidelity. The scope must be limited and the computer itself has limitations in terms of its processing ability. One example is how floating point numbers are only approximations of real numbers, as shown in another slide.

Orbits: Real-world orbits are multi-body problems where there is no definite solution, while this simulation only models two-body problems and puts the planets “on rails” with fixed orbits. In the real world, the orbit of a moon or spacecraft is perturbed by the sun and all the other planets and moons in the solar system.



Why use the Unity Engine?

Multimedia Computing Learning Outcomes

The choice of the frameworks and engine is about trade-offs related to project creation time.

Frameworks and game engines are a layer of abstraction on top of platform-specific systems for graphics (Metal, DirectX, OpenGL), audio, files, etc., and for common tasks such as physics calculations (PhysX).

Unity is a real-time 3D engine which takes care of common tasks when creating a 3D game, letting you to focus on the unique parts of your project. For example. it has a built-in physics engine, and can build for multiple platforms easily. This should lead to shorter development times.

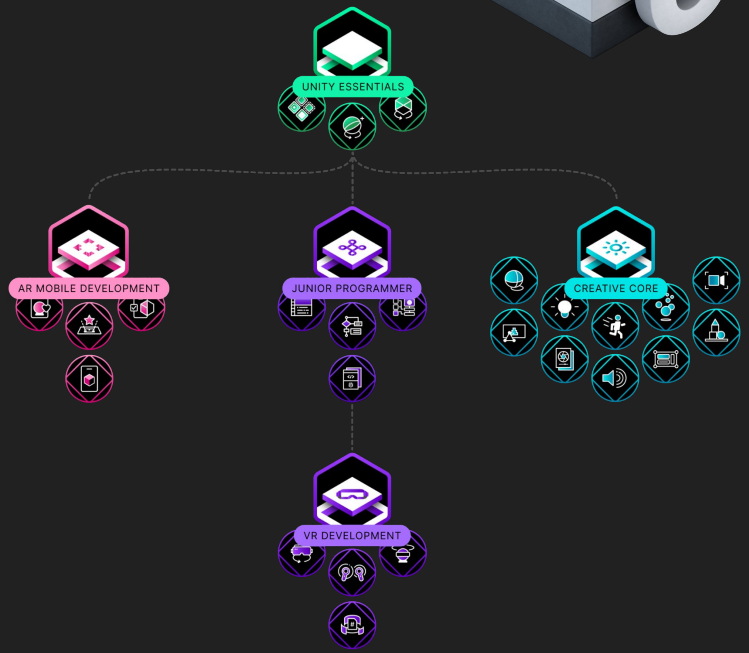
Trade-offs in using Unity Engine

Multimedia Computing Learning Outcomes

Unity has a steep learning curve.

The initial trade-off for someone who has never used Unity is the steep learning curve to understand how it works and the time needed to learn the basics before you can start your own project. The benefits only come after you have spent at least 100 hours in tutorials.

Another trade-off is between the kind of product that the engine is designed for and what kind of product you want to make. Unity is best at making a 1st or 3rd person 3D game like a shooter or walking simulator. It is not designed for spaceflight simulation, which results in more time and work to build functionality for it.



Methods and Tools: Part 1

Multimedia Computing Learning Outcomes

Unity: A real-time 3D engine which combines images, videos, and audio created and edited in other tools into a single, interactive multimedia product using WebGL or as a native executable program on Mac or PC.

Autodesk Maya: 3D modeling and basic animation. Can also be used for lighting and rendering 3D scenes into a sequence of images in PNG or EXR format.

Adobe After Effects: Taking the sequence of images from Maya, optionally combines them with audio and visual effects, and generating an encoded video file, for example, in H.263 or H.264 for video tracks and MPEG or AAC for the audio tracks.

Methods and Tools: Part 2

Multimedia Computing Learning Outcomes

Adobe Photoshop: Creating and editing raster image files for a wide range of purposes, including texture maps on 3D models and preparing images for the slide presentation, and saving in compressed formats such as JPG or PNG.

Sound Studio (Mac): creating and editing audio including music, ambience, and sound effects. Encoding and compressing in Ogg Vorbis, FLAC, WAV, MP3, and AAC.

Visual Studio Code: writing and debugging C# source code with Unity integration and the Monobehaviour class.

Methods and Tools: Part 3

Multimedia Computing Learning Outcomes

Panic Nova: HTML, CSS, JavaScript, Markdown, and other Web technologies.

GitHub: Git for both source code and media files, and GitHub Projects for project management.

Google Suite: for project notes, calculations in spreadsheets, and slide presentation.

Hardware: Developed on a Windows PC with a Core i7-11700 CPU & a NVIDIA GTX 1080 Ti GPU. Tested on a MacBook Air M3.

Hardware-Software Platform

Multimedia Computing Learning Outcomes

Initial platform: WebGL

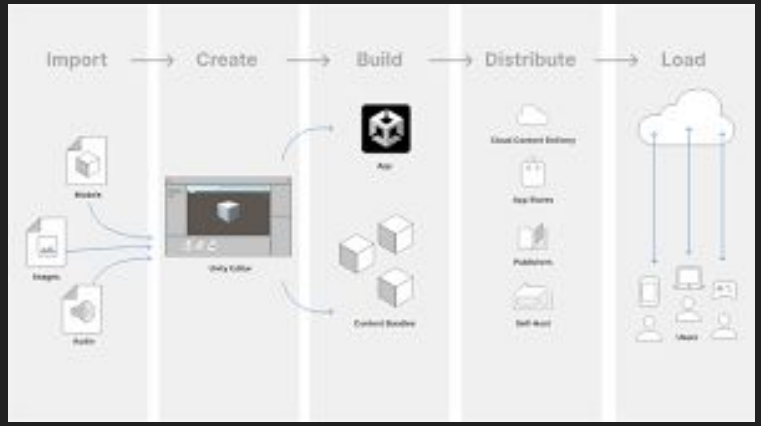
For maximum compatibility and portability, the initial version will run in a Web browser by using the WebGL platform build option in Unity. This has some trade-offs in terms of performance, but will allow most people to run it.

Storage and distribution: The files are stored on a server and downloaded to the Web browser when the player runs the program, so an Internet connection is required.

Hardware-software interfaces: Using the Web for distribution and a Web browser for running the program solves many of the issues with getting it to work on many different types of computers.

Future Platforms & Distribution

Multimedia Computing Learning Outcomes



Future platforms: PC, Mac, and Linux native executables will allow for higher performance by running directly on the CPU and GPU that the player has. The program package can be distributed either online or offline, and then stored and run locally on the player's PC, so no Internet connection is required.

Itch.io is planned to be the primary distribution website for a public release.

Human Computer Interaction

Multimedia Computing Learning Outcomes

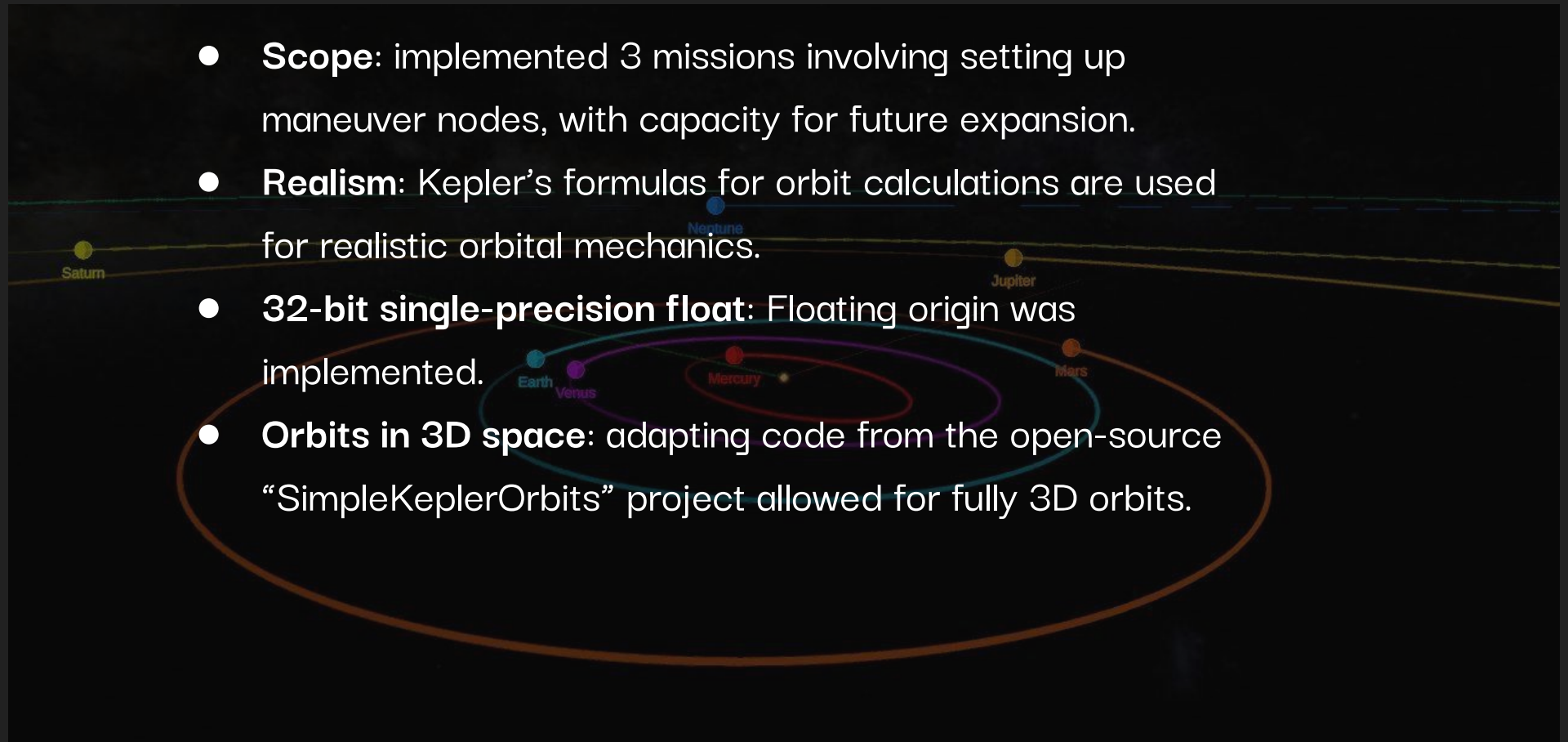
Considerations for UI/UX:

On-screen controls Because this program is aimed at users who are most likely using it for the first time, the controls such as buttons and sliders are visible to the player for point-and-click manipulation.

Keyboard and mouse control For more direct manipulation, some scenes allow the player to control the camera using keyboard and mouse. These are made visible with affordances and with instructions on how to use these controls.

Final: Solutions to Challenges

- **Scope**: implemented 3 missions involving setting up maneuver nodes, with capacity for future expansion.
- **Realism**: Kepler's formulas for orbit calculations are used for realistic orbital mechanics.
- **32-bit single-precision float**: Floating origin was implemented.
- **Orbits in 3D space**: adapting code from the open-source "SimpleKeplerOrbits" project allowed for fully 3D orbits.



Final: Finished Product

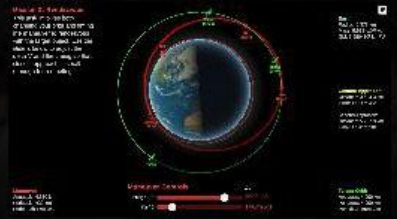
Title

Opening Video

Introduction

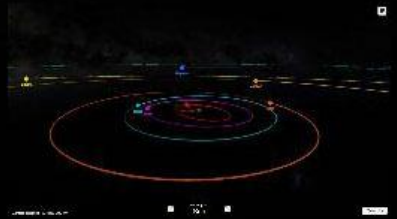
Launch to Orbit

Task 1: Change Orbit



Task 2: Rendezvous

Task 3: To the Moon

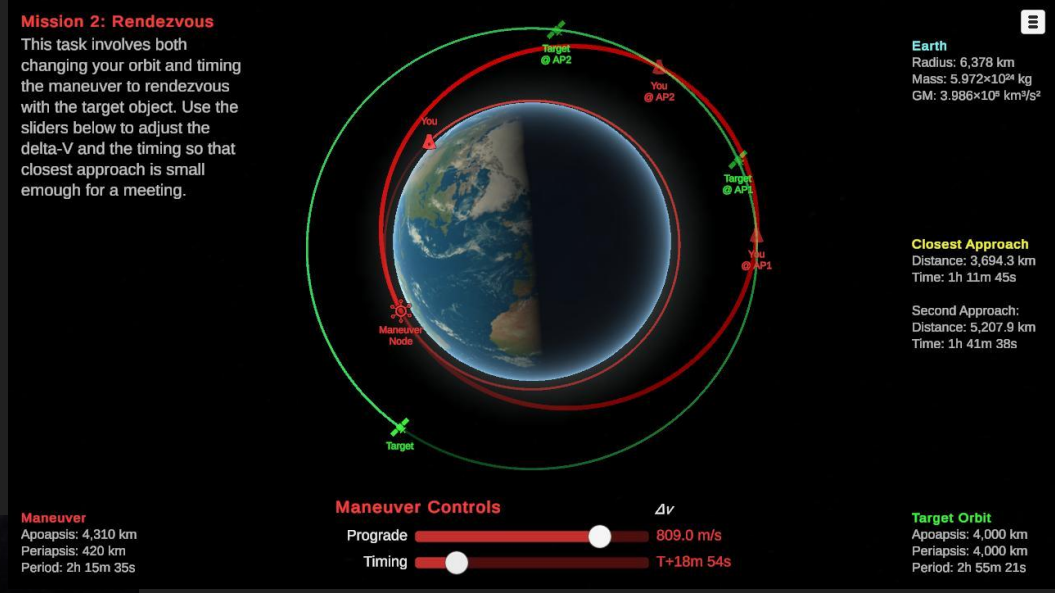


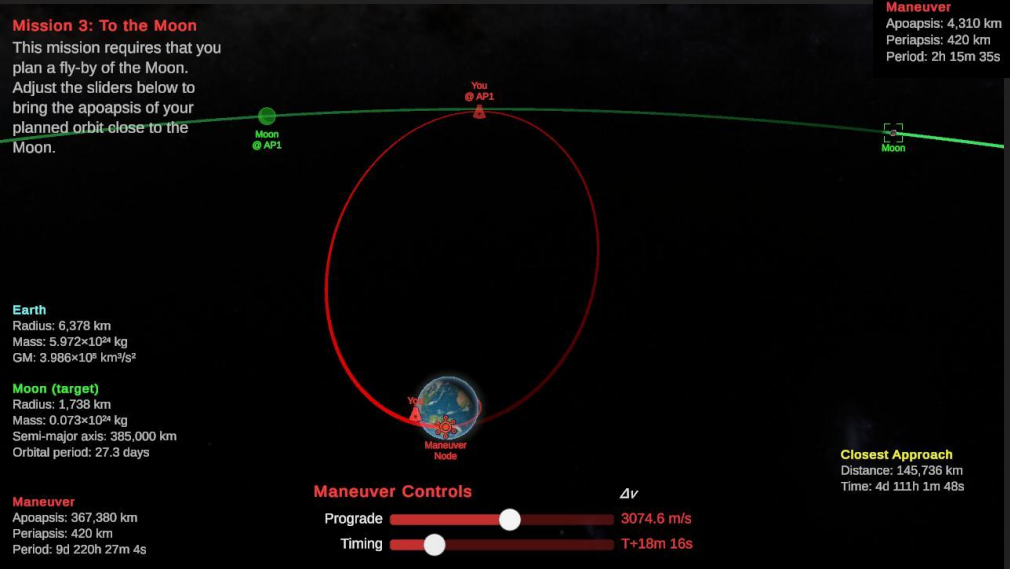
Solar System

End Credits

Final: Mission Detail

- Mission 2 teaches you how orbital rendezvous works





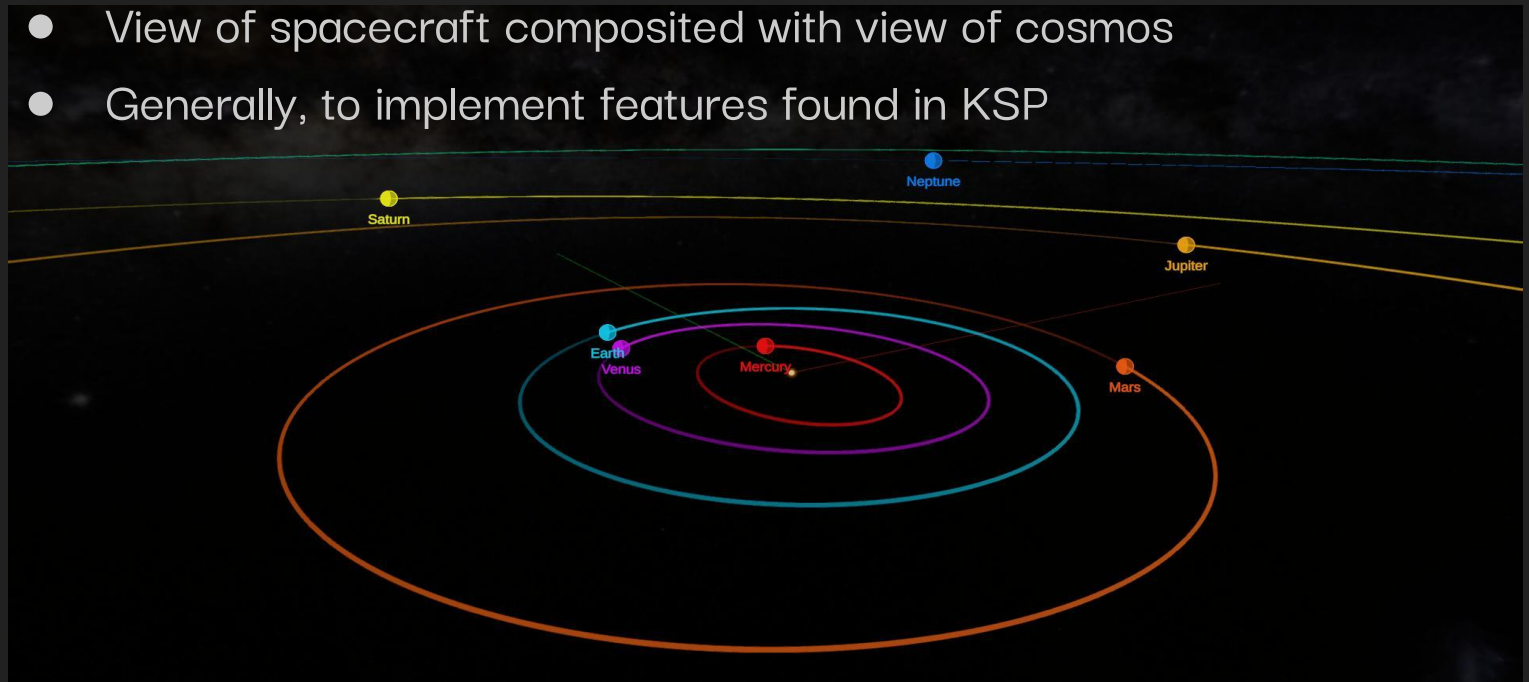
- Mission 3 teaches you how going to the Moon is like another kind of rendezvous

My Accomplishments & Contributions

- I created the concept, storyboard, and UI/UX design based on existing programs like KSP.
- Code: I wrote all the code except for orbital calculations adapted from Karth42's SimpleKeplerOrbits on GitHub.
- Graphics: I created the model and texture of the Saturn V rocket and its mobile launcher, and made the other images except for the planets, the sun, and the Moon from an asset pack.
- Sound: I chose the music and sound effects, with some editing to the audio, from royalty-free sources.
- I also learned how to use Unity Engine.

Future Plans

- Adding missions such as geostationary orbit, Hohmann transfers, plane change maneuvers, etc.
- Adding patched conics to transition between spheres of influence
- View of spacecraft composited with view of cosmos
- Generally, to implement features found in KSP



Endnotes

GitHub repository & project tracker:

https://github.com/luciuskwok/CISC-4900-Unity

https://github.com/users/luciuskwok/projects/2

Data sources:

Kerbal Space Program

NASA

Scott Manley

Unity Learn

University of Illinois

University of Texas

Wikipedia

Orbit Calculation

Karth42's SimpleKeplerOrbits

Saturn V-Apollo Specifications

Heroic Relics

Peter Alway