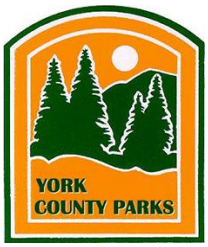

Visualization Technical Report

Team members: Derrek Herr and Jordan King

YCAS Radio Telescope Project
Senior Software Design Project I, Fall 2022
Prof. Donald J. Hake II

York College of Pennsylvania

Abstract

Team Saturn's contribution to the radio telescope project for the York College Astronomical Society (YCAS) involves the development of two Unity programs.

The first program is a game that consists of a scale model of John C. Rudy County Park where the radio telescope will be located with a model of the telescope in place. This game can be played using either traditional mouse and keyboard controls or by using any virtual reality headset. In this game, the player can manipulate the telescope model and highlight different parts of the telescope to learn what each of them do. The purpose of this game is to be used as an educational tool and for community outreach to create interest in astronomy.

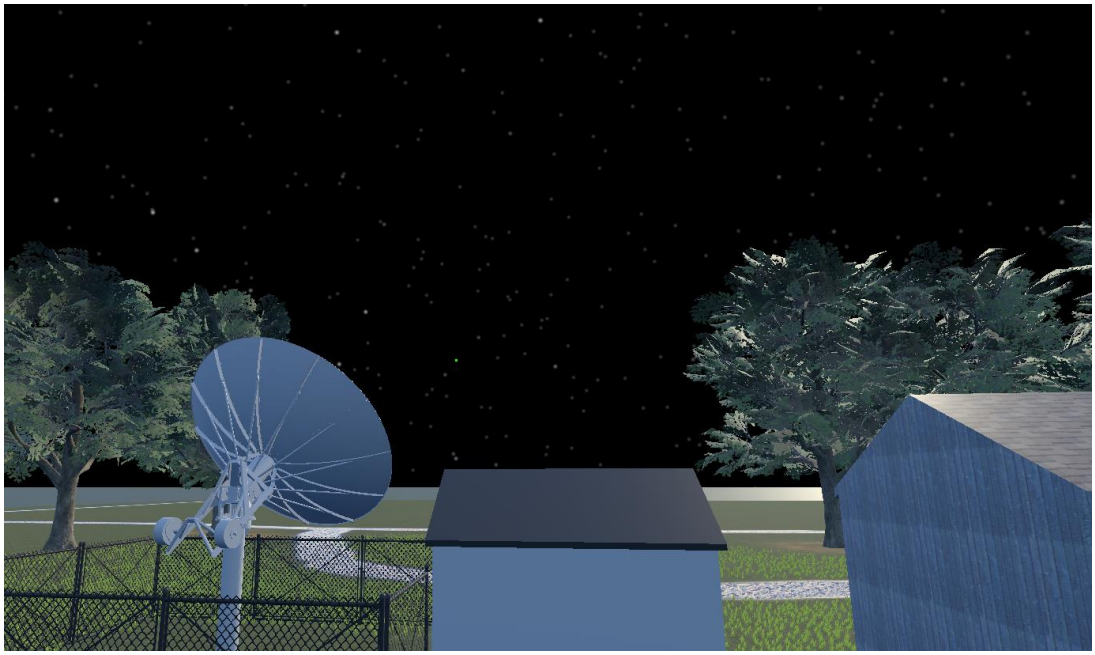

Figure 1: A screenshot of the educational game. (2019)

Figure 2: A screenshot of the unfinished control panel (2022)

Our new member added some new basic elements within the project to expand on their experience within Unity. These new elements regard more interaction within the environment to invoke immersion within the program. Also, additions have been made to integrate tools for the player to create date specific visualizations. It encourages exploration by providing the tools to aid in this endeavor. This is in the form of the control panel, as shown in figure 2.

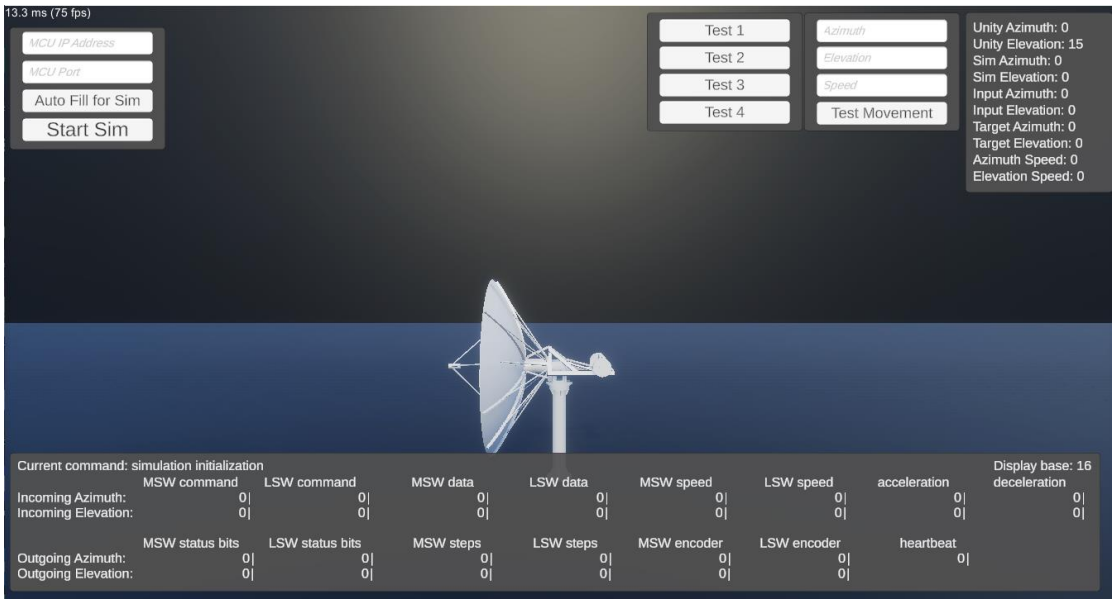

Figure 3: The hardware Simulation after it has finished initialization.

The team's primary focus for the Fall 2022 semester was on the continued development of the virtual reality and PC visualization programs. The goals for this semester were to update the systems built in the last semester and make the visualization programs fully shippable by the end of the semester. This semester mostly consisted of the development of new features and additions, rather than any optimizations as the program seems to be optimized at this point.

Introduction

Team Saturn this semester primarily consisted of two members, with help being provided by those in the control room where necessary.

This semester continued the work started in 2019 on the educational game. At the beginning of the semester, this educational game was mostly functional, however it lacked some of the features that the client was requesting. The primary goal of this semester was to create a working star positioning system, update the existing star interaction system, and create a brand new cinematic that would help explain the functions of the telescope. The star positioning system needed to be created so the user would be able to see all visible points in the sky. This meant creating a system that would take the date from the user, and rotate the existing star system to that specific date in time. The star interaction system, while mostly operational, lacked the primary feature of being able to hold multiple data points. This addition required multiple UI changes, a rewrite of the existing system, and some .CSV database changes.

A working gate system for the player was also added. Now they can open and close the gate with the click of their mouse when hovering over the entrance. In addition, a tool was added to the program in the form of a console that can manipulate time. Thus, the player can manipulate the position of stars by hovering over the buttons on the console and clicking them. The buttons themselves control the increase/decrease of the date unit and switch between date units. As a result, this allows the player to manipulate the sky for the preferred date and time to gaze at the stars.

Given that no development work has been done on the simulation this semester, this technical report will not focus on it. For information on the Simulation, see the technical report from Spring 2021[3]. An example image of the Simulation can be found above in figure 3.

Background

The programs that Team Saturn develops are all Unity[5] projects. Unity is a powerful game engine capable of working with both traditional mouse and keyboard (M&K) controls as well as VR controls. Unity projects consist of a list of "game objects," each game object having a list of properties associated with it. These game objects can be invisible and only work as logic handlers, or can be visible models such as the radio telescope model that we use.

Game objects can have scripts attached to them. These scripts are written in C#, capable of using standard C# libraries and Unity's scripting library[6] to manipulate how game objects act and interact with each other and the player. For example, the project utilizes raycasting[13], an object is set to produce a line of a certain length which, when collided with particular objects, can retrieve information about that object. The VR toolkit (VRTK) library[7] is also used to handle VR controls for the VR educational game, with SteamVR[8] being used as a means of connecting the VR headset to the Unity program. The VR headset that Team Saturn uses for the VR educational game is the HTC Vive[9], a VR headset supported by Unity, the VRTK library, and SteamVR. Thanks to the compatibility provided by SteamVR, the program also runs on an Oculus Quest 2 and any other SteamVR-supported device. In addition, the project now uses Unity's Cinemachine[21] for the Cinematic additions.

Unity is used not only for its VR support and game development features for the purposes of the educational game, and its GUI for the purposes of the Simulation, but also for the ease of integration with the Control Room. Since both the Control Room and Unity scripts are written in C#, communication between the two is relatively simple, making use of a TCP connection and Modbus registers to pass information back and forth, the exact same connection that the Control Room would make with the hardware; this common

interface is what allows the Simulation to act as a proxy for the hardware to test the Control Room's behavior and how the hardware would react to its commands.

Build instructions, how to run the Simulation, and how to connect it to the Control Room can all be found on the Simulation's Github repository wiki section[10].

Design

Unlike your typical IDE, Unity uses a unique system of GameObjects that operate very similar to classes. GameObjects can have scripts attached to them which are then compiled together and run when enabled. Basically, all interactions between classes are done through GameObjects. GameObjects are added to the world and then have scripts attached to them. Scripts are like classes, and GameObjects are like instances. So for a class to have an instance of another class, it must be a script attached to an object and have a reference to an object (which can be the same object that it is attached to) that has the script that it wants access to attached.

Character Environment Interaction

A gate system was added to the project in order to improve immersion. The main way for the player to interact within the environment is by clicking/hovering over objects. As a result, a new raycasting script was needed in order to interact with the object without damaging any features. Next, the object was made a child to a parent object to simulate the desired rotation. Then, the motion of the gate was developed by a simple boolean if-statement system for determining if it's closed or open. Thus, the system checks the current rotation of the parent and determines if the gate is in the open or closed state. As a result, it can determine what direction the gate will be rotated based on its current state.

Star Positioning Script

In order to see all possible star constellations and points in the sky, A system needed to be created that would allow the adjustment of the star system based on a specified user date and time. So, a new function was created within the star system script that takes a specified date and time, and then rotates the star system to that specified point. Upon research, it was found that stars operate on a different timescale, specifically, they are roughly 4 minutes faster on a day to day basis. This difference in time accumulates throughout the year, causing the stars in the night sky to change from day to day, this is why there are different constellations depending on the season. Because of this, the calculations would mostly be based on sidereal time[17], which is the natural 23-hour, 56 minute, and 4 second rotation of the earth.

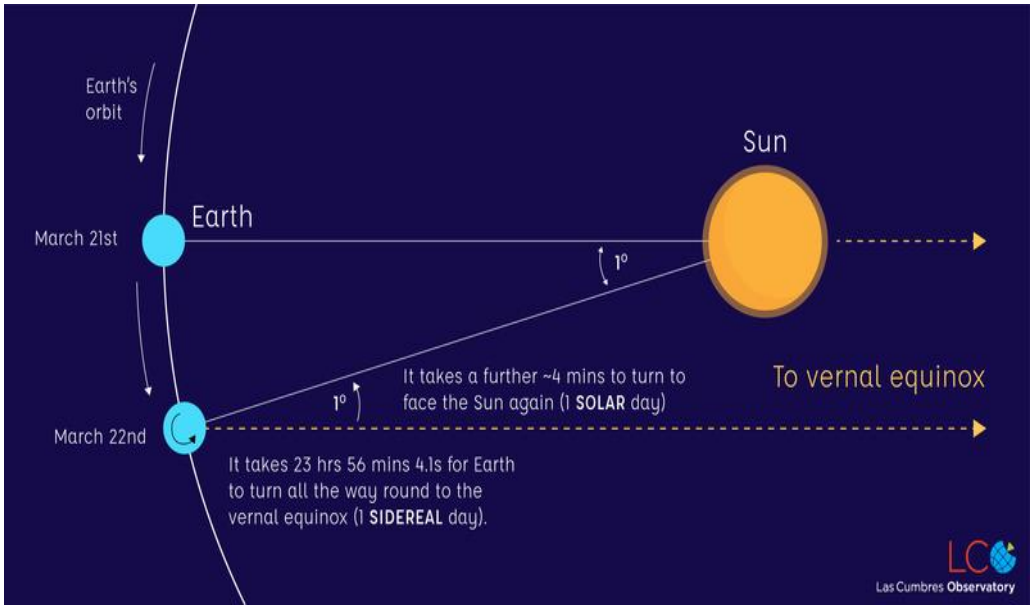

Figure 4: A graphical comparison between sidereal time and solar time[17].

The actual calculation would be based on the user specified data and times, as well as the sidereal time offset of the stars. In order to accurately check if the system was working properly, and to create any references, an external star database website Stellarium[18] was used.

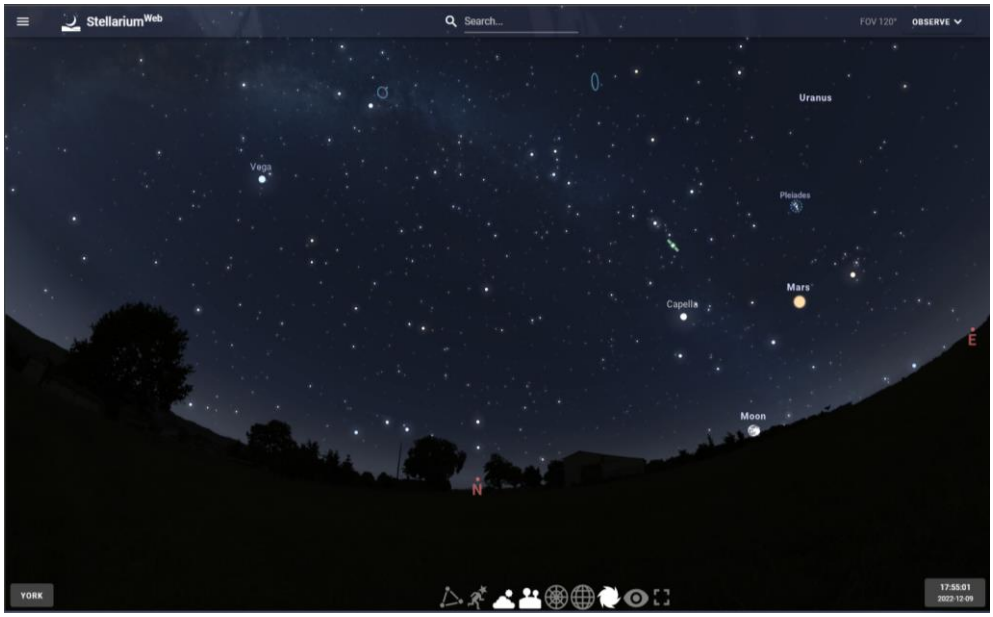

Figure 5: The user interface for Stellarium, an external star database website. This was used to fact check our star positions.

A control panel would be built to control the script and provide user interaction. The aim for the control panel is for the user to be able to change the date/time from the

panel itself with the various buttons. The collection of buttons on the console face is currently four, a set to control decrementing or incrementing of date units and a set to control the switching of date units. Within the script of these buttons they change the starfield directly for whatever date unit the player is currently on. Therefore, a way to switch between the date unit was needed. As a result, a new variable was created, DateUnitID, which resides within the Starfield script. Now, the buttons can easily change between the date units and DateUnitID can be referenced to see what date unit is being incremented or decremented.

Star Interaction System Updates for Multiple Data Points

Our client made clear that some changes needed to be made to the original star interaction system built in the Spring 2022[19] semester. The original system, while fully operational, did not account for multiple data objects per point(a data object being the data image, RA and DEC[14] values, and the date/time captured). This meant that a point in the sky, like the sun for example, could only have one data object. The client wanted to be able to cycle through multiple data objects per point. This adjustment required multiple additions and refactors of the original data code.

The process of making this addition included an addition to the .CSV database that would allow the addition of a datetime[20] string. This datetime string would be converted to a .NET datetime variable to be used in the sorting of data objects. A new struct was created for the data objects which included all the information present in the .CSV file

	A	B	C	D	E	F	G	H	I
1	2	31	89.26	500	Polaris	Polaris is a	north	10/19/2022 9:00	
2	11	1	56.382	500	Merak	Beta Ursa	merak	10/19/2022 10:00	
3	13	47	49.313	500	Alkaid	Alkaid; als	alkaid	10/19/2022 11:00	
4	13	47	49.313	500	Alkaid	Alkaid: Te	example1	10/19/2022 12:00	

Figure 6: Excel Sheet of "Sky_Data": RA, DEC, DIST(lightyears), Label, Desc, Image Name, Datetime String.

This new struct was then used to create multiple data objects from the .CSV file. These objects would then be allocated to each individual point in the sky. The data objects were then grouped together based on their proximity to each other.

Multiple UI changes were made to accommodate the additions of the datetime string and the function of cycling individual data objects per point. These additions were added on top of the old star interaction UI.

Cinemachine & Cinematic Addition

Our client expressed interest in a sort of cinematic that would be able to quickly teach participants of the functions and purpose of the radio telescope without user interaction. For this feature it was chosen that Unity's own cinematic system, Cinemachine[21] would be used for the development of this feature.

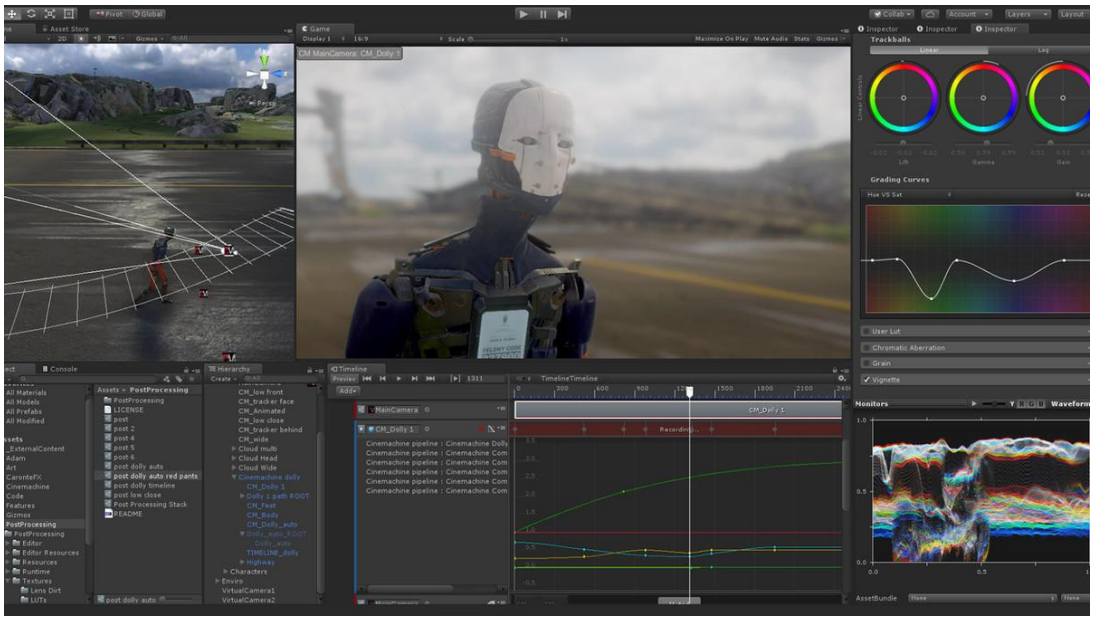

Figure 7: An example image of the Unity Cinemachine developmental workspace.

For the planning and design of the actual cinematic, a storyboard[22] was created and presented to the client. The client then made additions and changes to the overall cinematic scenes.

As for the development of the cinematic within Unity. It was found early on in development that the VR version would need multiple accommodations as Cinemachine was not fully compatible with VR. Also, Cinemachine did not have any form of text function, and so a wrapper class had to be created for subtitles. Also, Cinemachine did not have any form of fading transitions, and so a quick system was built for that as well. It was also found that any form of movement that was not within the user's control would induce nausea, so all scenes were made stationary with this in mind.

Because of the differences in the VR version, it was decided that an entirely different scene would be created for both the computer and VR programs. The process of initiation would involve the player clicking on an added cinematic button, which would load the new cinematic scene, play the cinematic, and then reload the original play scene with the telescope.

Implementation

Character environment interaction

Implementation of more character interaction within the environment was relatively easy. The process I utilized to create the gate and the console functionality would be creating a fundamental model. Therefore, avoiding the common mistake of making an object appealing without developing the core functionality. As a result, the gate's model was only a basic cube object, and the main focus was on opening/closing. When handling the script's development, it is advised to start and develop functionality in bits and pieces. Thus, the script to activate open/close was developed originally to move the gate with the activation from a key press. Therefore, this removes any possible outside factors affecting the desired performance and makes troubleshooting leagues easier. Moving onto the ray casting would be permitted, knowing the base functionality is sound. The challenges with the implementation were reasonably low due to having experience with the Unreal engine. However, there were issues with getting used to how Unity handles raycasting.

Star Positioning Script

Implementing the control panel was moderately challenging, considering that experience with the Unreal engine was present. However, there were caveats to implementing the raycasting and a terminology difference. For example, in Unreal, raycasting is referenced as line tracing, which needed to be clarified to research documentation for implementation in Unity. In addition, the implementation is different between Unity and Unreal; typically, one script would be needed for multiple objects in Unreal. However, for Unity, each object needs its own personal script, or it will activate the script as many objects have it. In addition, it was tackling a new feature, displaying text on a 3D object, which was challenging in its regard. Since the tricks and limitations were unknown and caused a period of long experimentation to understand the text object feature. The culprit was implementing a solution only that did not account for multiple text objects. However, thankfully further research into the subject revealed a better solution that accounted for multiple text objects. This solution was to have the text object reference itself rather than the TextMeshPro. Therefore, each text object was unique and could display various texts rather than having one object display something and the rest nothing.

As for the actual star system adjustments and calculations, the first step was to actually find a reference point that could be used for the rest of the script's adjustments. For this I used the external tool Stellarium to get my reference point positions. For the actual reference point, the Vernal Equinox of 2022[23] was chosen as this seems to be a common point in astronomy.

From this reference point a datetime is created and then referenced for all future calculations. After the user enters the appropriate hour, day, month, and year, another datetime is created, this is the time the user requests the stars to be positioned at. Next, the difference in minutes between these two points are calculated. From this difference, we can find the amount of time that has passed from the reference. Since the reference is correct, all that needs to be done is multiply this difference by the amount of degree change per minute. This can be given by calculating the amount of sidereal days from the difference, and multiplying the degree of change by the amount of minutes passed. This comes to around a 0.25 degree rotation every minute.

However, this system did not align entirely with the information that Stellarium was giving. To accurately check if the stars were aligned with their specified dates, a separate tool was created inside of Unity. Instead of having to calculate each star position for each test, a simple cube was created inside of Unity that would automatically face a chosen position. From this, the position values of the cube could be measured to determine the Azimuth and Elevation[24] values of whatever it was facing.

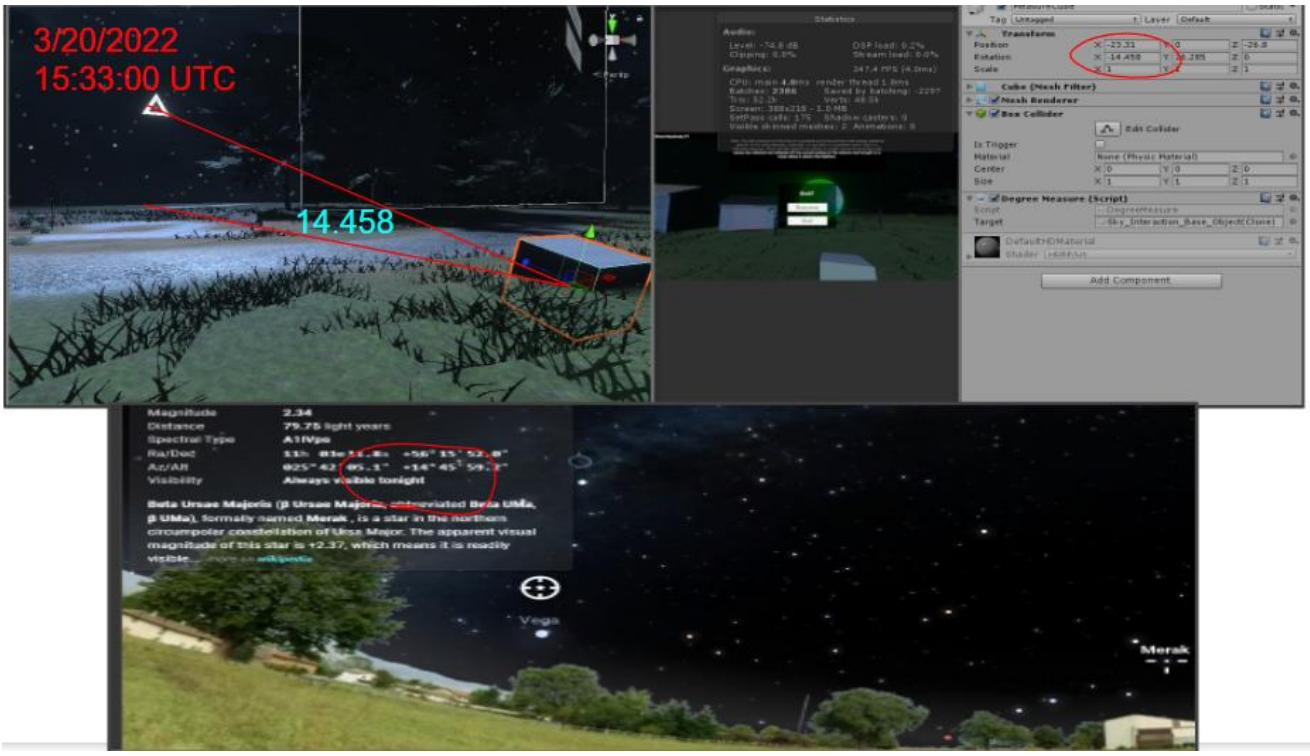

Figure 8: The measure cube was programmed to measure the position of a chosen star. This image takes place at the Vernal Equinox of 2022, our chosen reference point. The accuracy of this reference is nearly identical to the reference positions given by Stellarium.

To adjust for this inaccuracy, leap years are then calculated separately from the minute difference, while still using sidereal days. Based on the offset year of which the date

is from its closest leap year, an additional 0.25068 degree of difference is added on for every year that is ahead of the current leap year. After combining this leap year offset with the original yearly calculation, the final amount of degrees is set and the star system is then rotated when prompted by the user. This method of calculation is able to get the correct star positions to an accuracy of around 3 degrees of rotation, which is nearly invisible to the naked eye.

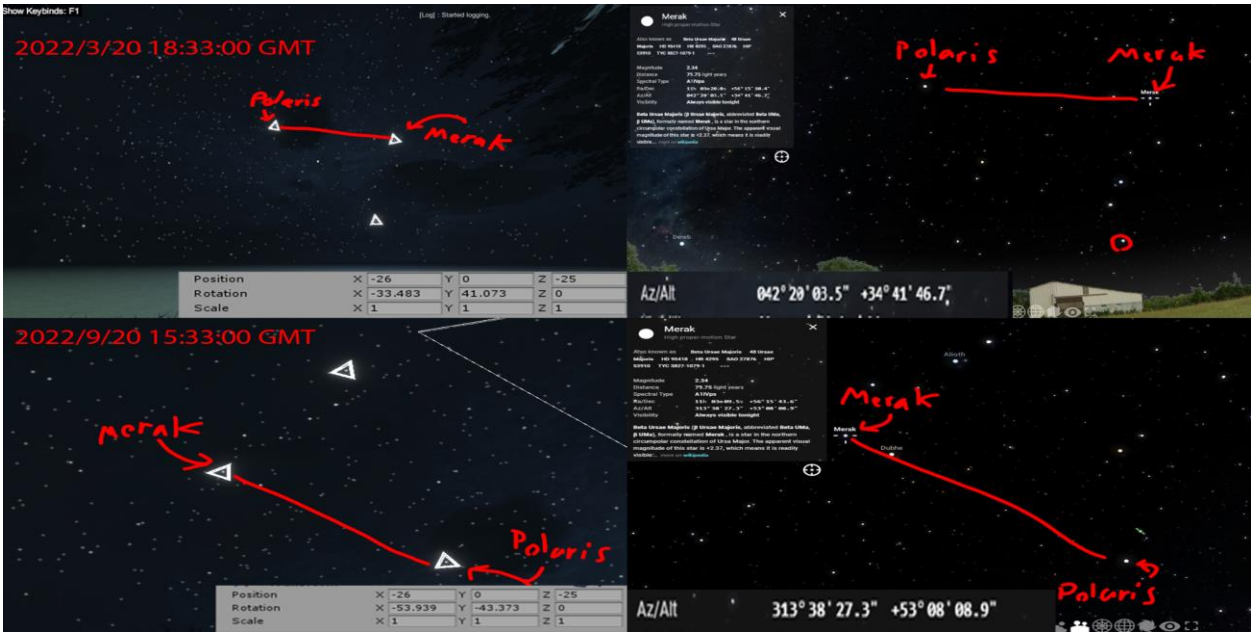

Figure 9: This image shows the comparison between the in-game star positions(left) compared with the Stellarium star positions(right). Star positions in this example are accurate to the naked eye.

Star Interaction System Updates for Multiple Data Points

Our client made clear that one of his most requested features was the ability to interact with the points in the sky that the telescope had received data from. Last semester the main functionality of this system was created, however it lacked the vital function of being able to handle multiple data points in one point in the sky. Since the telescope is mostly going to collect data from a few specific points, this feature had to be added to complete functionality.

First off, the .CSV file structure was altered to handle datetime strings; this is needed to not only sort the data points but also label them. This, along with all the other information contained within the .CSV is put into a list, which is then sorted by this datetime object. In order to group points that are the same, each point is checked based on it RA and DEC values, if a point is found that is already within 2 degrees, instead of a new point being spawned within the program, its data object (which contains all its

information) is added onto the already existing point. Effectively, this system completes the spawning of all data points, which have multiple data object functionality, in O(n log n) time complexity.

In order to get the multiple data points working, a new struct was created for the data objects. Instead of each point containing a singular points data, the points now contain a list of data objects, which contain the data for each line in the .CSV. This allows the user the ability to cycle through multiple dates worth of data, all on a single point in the sky.

Multiple new UI changes were added to accommodate the addition of date/time data, as well as indicators of which data object is currently visible, and if a point has multiple data objects available.

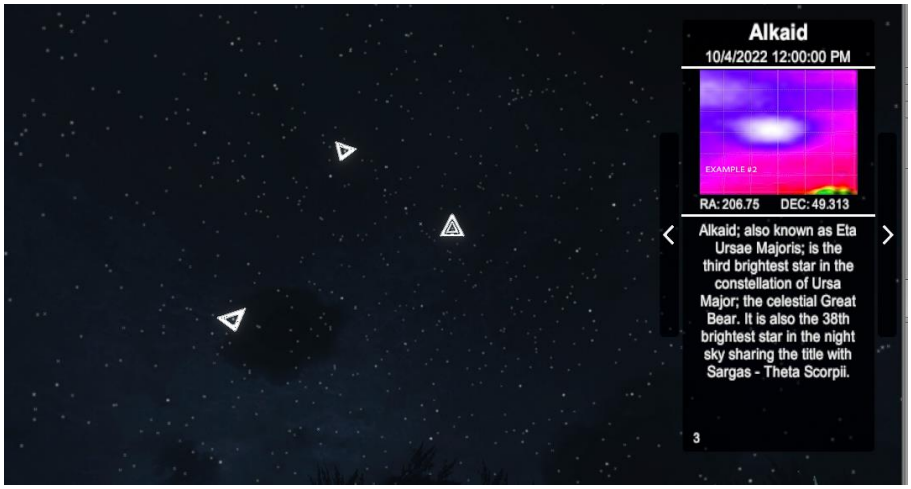

Figure 10: Visual of updated sky interaction UI. On the right is the UI display after selecting the triangle object for the star “Alkaid”. The label can be seen at the top, the datetime string right below the label, the corresponding .JPG in the middle, the RA and DEC values right below, and the description is then listed at the bottom. The left and right arrows indicate if the user can cycle to another data object, these arrows turn transparent if there is no corresponding data object in that direction.

In addition, the VR version of this system had its control scheme changed to accommodate the ability to cycle through multiple data objects.

Cinemachine & Cinematic Addition

To begin the development of an all new cinematic system, a storyboard was first created in order to plan out the development and progress of the cinematic. This storyboard was then presented to the client, of which many changes were made. With this done first, the expectations of the cinematic as well as its goals were clearly defined.

Upon the research and prototyping of the new Unity Cinemachine, it became apparent that three things were problematic; the lack of text functionality, the incompatibilities with VR, and the lack of a proper transition system.

For the development of the cinematic, a test scene was created that contained some of the assets that the main game also contained. This was done so no conflicts would occur during development between the completed sections of the project and this new framework. An example image of this test scene is shown below in figure (11).

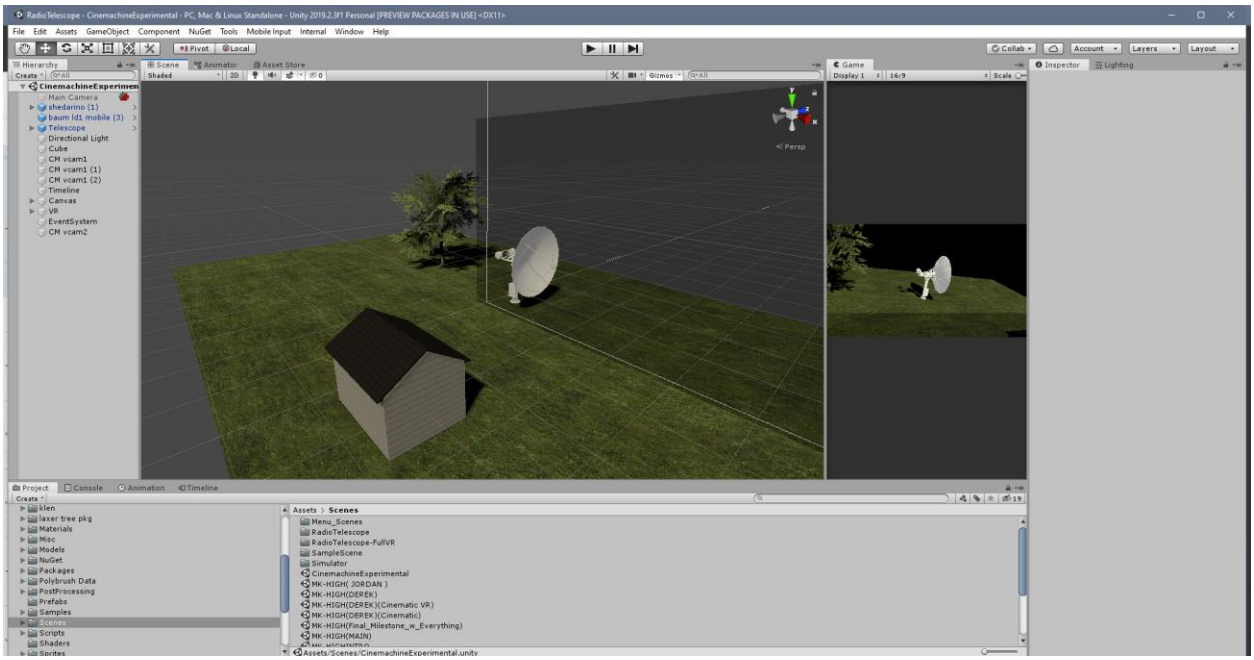

Figure 11: The test scene used to develop the cinematic. Cinemachine was added to this scene and tested without the risk of interfering with other game scenes.

The start of development first focused on creating a text wrapper class that would work within Unity's Cinemachine framework. This allowed the easy creation of text objects that would be shown on screen below each of the cinematic shots. Next, because there was no easy transition system already within Cinemachine, a work-around had to be created that would allow easy transitions. This required the creation of a shot that only displayed a black output, this shot would then be transitioned to between scenes to simulate a black fading transition effect.

In order to start the cinematic, a simple button was created within the main scene that, using a raycast, would start the cinematic if the player clicked on the button.

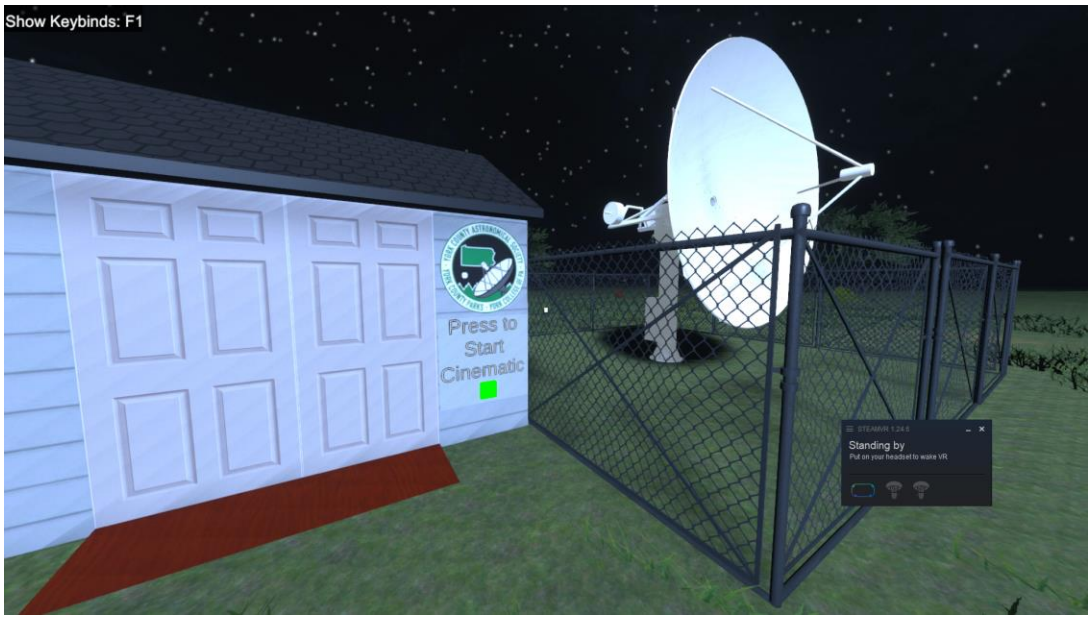

Figure 12: The Cinematic start button. The user simply clicks on the green button to start the cinematic. Once the cinematic is finished, the user is returned to the main play area.

An example of what the cinematic looks like in the mouse and keyboard version is shown below in figure (13).

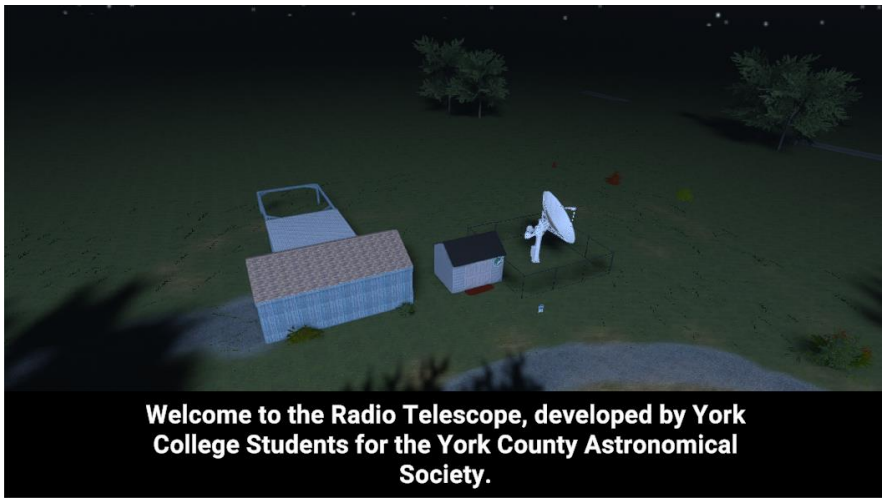

Figure 13: An example shot of the cinematic. The subtitles for each shot are displayed at the bottom .

In general, much of the cinematic system was completed, and is fully operational on the mouse and keyboard version. However, due to some compatibility issues with cinemachine, the VR portion of the cinematic remains unfinished. The player can interact and start the cinematic and see the text for each shot, but the particular positions of the

player are not aligned with the mouse and keyboard cinematic, often with the player facing incorrect positions when playing the cinematic. A polishing of the virtual reality shots would make the VR cinematic much more operational.

Future Work

As for future work, multiple additions were never made that were originally planned. Firstly, the cinematic for the VR is not fully complete. The setup for the scene, the text functionality, and the scene management are complete, however the virtual camera scenes still need to be polished as the VR shots do not align with the mouse and keyboard cinematic shots.

Also, the sun and moon were never added to the program. With this being one of the main focuses of the telescope, the addition of the sun and moon would be a logical and welcomed feature that has been requested by our client.

An interactive virtual appointment was also planned but was way out of scope for this semester. A virtual appointment would require some complicated additions and calculations, much of which would have taken too much time to fit into this semester.

Also, the main menu never got the graphical options that were originally planned for this semester. With the addition of the cinematic scenes, this may be more difficult than it was originally planned. At this point, this seemed to be pretty low on importance and was scrapped for this semester as our client agreed that the other tasks were more important. The continuation of this would make the .exe files more organized and would limit the amount of scenes required in the Unity build file.

Lastly, the Quest 2 demo program could be modified to include many of the new features and implementations that are present within the main program. This would require a lot of development time and so, it was not attempted this semester.

References

[1] Modbus protocol, https://modbus.org/

[2] *log4net* library, https://logging.apache.org/log4net/

[3] Spring 2021 Team Saturn Technical Report, https://docs.google.com/document/d/...

[4] Fall 2020 Team Saturn Technical Report, https://docs.google.com/document/d/...

[5] Unity, https://unity.com/

[6] Unity scripting library, https://docs.unity3d.com/Manual/index.html

[7] VR toolkit library, https://vrtoolkit.readme.io/docs

[8] SteamVR, https://store.steampowered.com/app/250820/SteamVR/

[9] HTC Vive, https://www.vive.com/us/

[10] Simulation GitHub repository wiki, https://github.com/YCPRadioTelescope/TelescopeVisualization/wiki

[11] Fall 2021 Control Room Technical Report, https://docs.google.com/document/d/...

[12] Blender https://en.wikipedia.org/wiki/Blender_(software)

[13] Raycast https://en.wikipedia.org/wiki/Ray_casting

[14]RA & DEC

https://www.celestron.com/blogs/knowledgebase/what-are-ra-and-dec#:~:text=RA%20(right%20ascension)%20and%20Dec,like%20latitude%20on%20the%20Earth

[16] OpenXR https://www.khronos.org/openxr/

[17] Sidereal time Solar Time vs. Sidereal Time | Las Cumbres Observatory (lco.global)

[18] Stellarium Stellarium Web Online Star Map (stellarium-web.org)

[19] Spring 2022 Tech Report Team Saturn Technical Report - Google Docs

[20] Datetime DateTime Struct (System) | Microsoft Learn

[21] Cinemachine Cinemachine Documentation | Package Manager UI website (unity3d.com)

[22] Telescope Storyboard Telescope StoryBoard - Google Slides

[23] Vernal Equinox of 2022 Enscript Output (weather.gov)

[24] Azimuth and Elevation for Stars homework.uoregon.edu/pub/emj/121/lectures/skycoords.html#:~:text=When the star is directly,to north (360 degrees).