Visualization Technical Report

Team Members: Derek Herr

YCAS Radio Telescope Project
Senior Software Design Project II, Spring 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 an HTC Vive Virtual Reality (VR) 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.

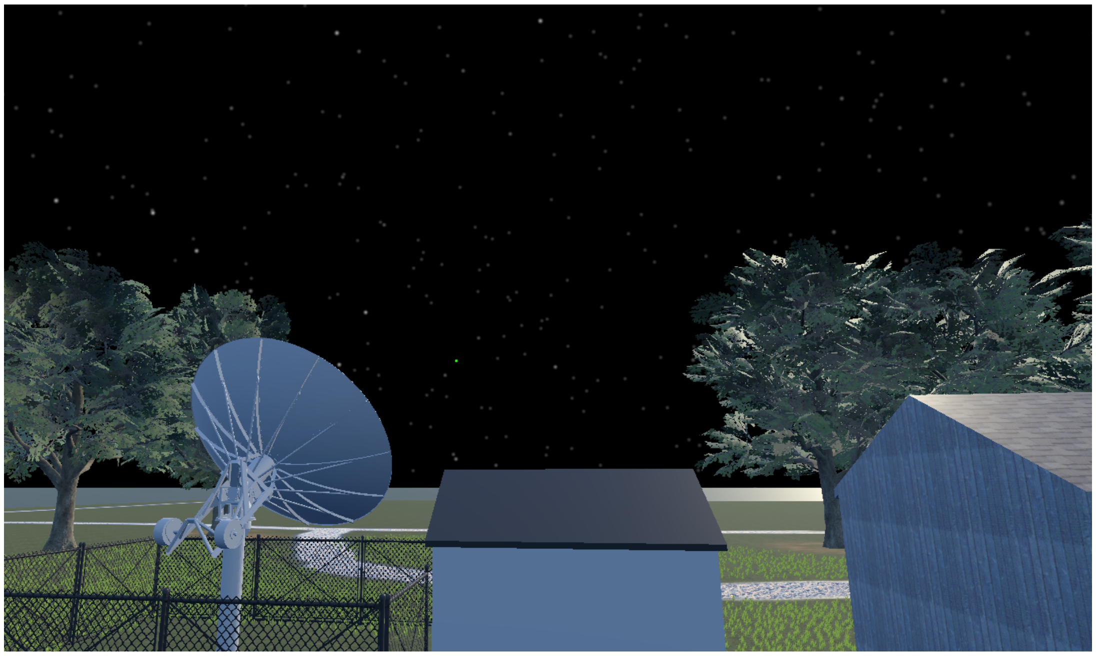
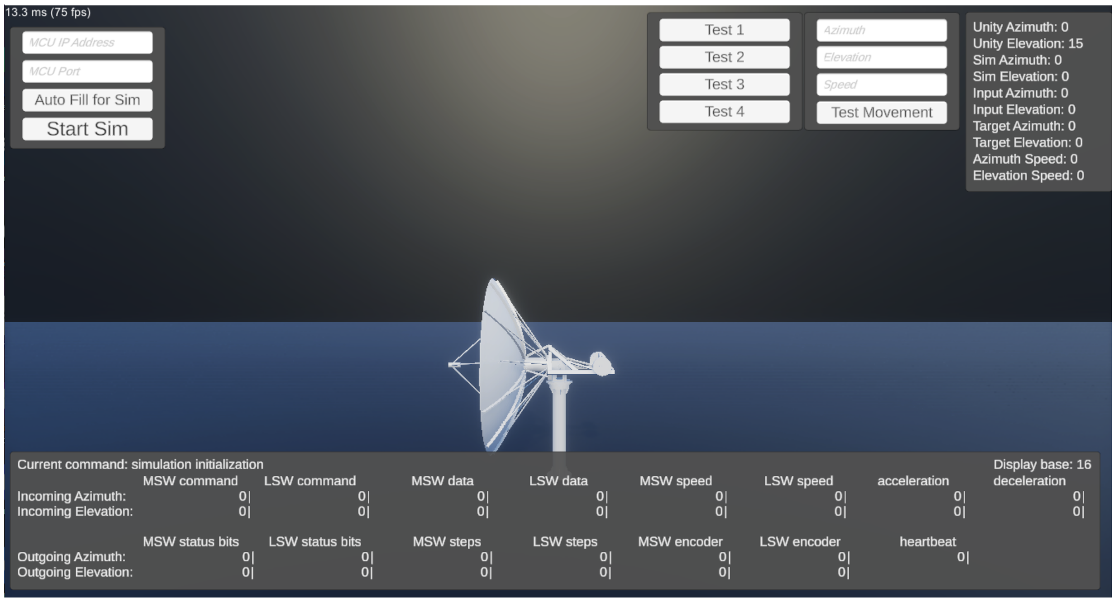


Figure 1: A screenshot of the educational game. (2019)

The second program is a Simulation of the radio telescope hardware. This program acts as an interface that interacts with the Control Room, receiving commands from the Control Room as the actual hardware would receive them and executing them on a virtual telescope model as the hardware would execute them. The purpose of this hardware Simulation is to allow other teams to test their changes without requiring that they have access to the hardware, greatly increasing turnaround times on new features and allowing for much easier troubleshooting.



```
Unity Azimuth: 0
Unity Elevation: 15
Sim Azimuth: 0
Sim Elevation: 0
Input Azimuth: 0
Input Elevation: 0
Target Azimuth: 0
Target Elevation: 0
Azimuth Speed: 0
Elevation Speed: 0
```

```
Current command: simulation initialization                                                                  Display base: 16
                      MSW command   LSW command   MSW data   LSW data   MSW speed   LSW speed   acceleration   deceleration
Incoming Azimuth:              0|            0|         0|         0|          0|          0|             0|             0|
Incoming Elevation:            0|            0|         0|         0|          0|          0|             0|             0|

                  MSW status bits  LSW status bits  MSW steps  LSW steps  MSW encoder  LSW encoder   heartbeat
Outgoing Azimuth:              0|               0|         0|         0|           0|           0|          0|
Outgoing Elevation:            0|               0|         0|         0|           0|           0|
```

Figure 2: The hardware Simulation after it has finished initialization.

The team's primary focus for the Spring 2022 semester was on the continued development of the virtual reality and PC visualization programs. The goals for this semester were to completely update the visualization programs to be fully shippable by the end of the semester. New features along with multiple optimizations were added to both the virtual reality program as well as the mouse and keyboard program.

Introduction

Team Saturn this semester primarily consisted of a single member, with help being provided by the mechanical engineers where necessary when it came to updating the telescope model.

This semester continued the work started in 2019 on the educational game. At the beginning of the semester, this educational game was roughly functional, however it had some massive optimization problems as well as outdated models and frameworks. The primary goal of this semester was to fix any game breaking bugs and optimize the program to the point of being fully functional and shippable to any VR device as well as any PC. Throughout the semester, multiple accessibility additions were made, as well as some new features. A main struggle was getting the program operational in terms of performance, thus, many changes were made to optimize the program. An updated version of the model was also provided in conjunction with the mechanical engineers. Lastly, many accommodations were made so the program could be compatible with any VR device, including the Quest 2, which also resulted in a portable Quest 2 APK project being developed for quick demonstration purposes.

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 Fall 2022[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 are able to have scripts attached to them. These scripts are written in C#, capable of using standard C# libraries as well as Unity's scripting library[6] to manipulate how game objects act and interact with each other and the player. The VR toolkit (VRTK) library[7] is also used for the purposes of handling 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 makes use of 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.

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

Unity is somewhat different in that it uses GameObjects. 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.

Performance Optimizations

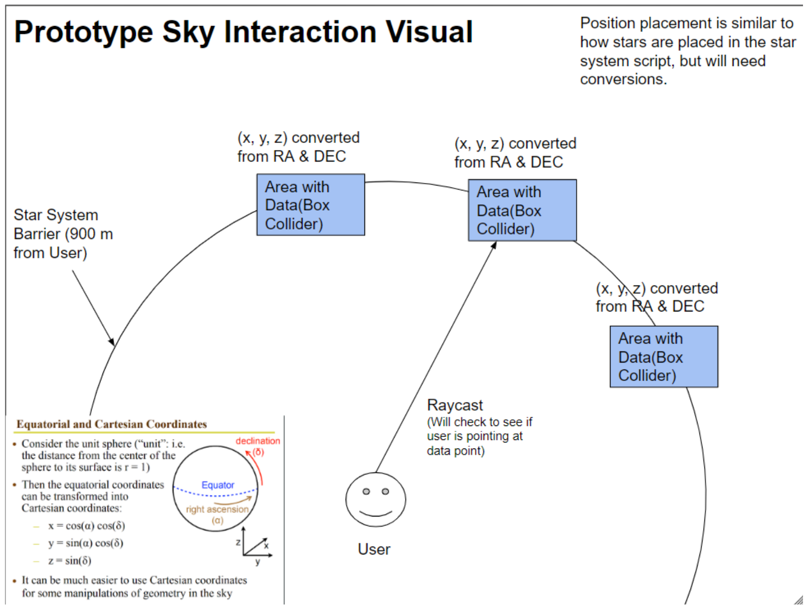
Upon reviewing the 2019 project, multiple red flags were evident in terms of performance and optimization. Most notably, the “Starfield” script, which is responsible for reading the star positions from a CSV file and displaying them as particles was both plagued with bugs and consuming much more frametime than was acceptable. In addition, the grass models were exponentially expanding the triangle count by a degree that was unacceptable. These two issues were the highest priority in development as they would prevent the completion of a shippable product.

Upon reviewing the “Starfield” script, it was noticed that the CSV was being read on every frame, as was the particle system. The current design was set up in a way that had the Unity Particle System initializing a particle for each star every frame. Since the stars themselves are not changing positions, the strategy for fixing this bug was to restructure the Unity Particle System so that the stars themselves were only initialized once, which would only require the CSV to be only read once instead of every frame. This would eliminate the constant particle initialization and limit the CSV file reader to initializing only once.

In addition, it was also noticed that the grass models(of which there were hundreds) were 3D models instead of 2D models. Because of this, the triangle count of the program was extremely high(around 1.2 million triangles). This was significantly bogging down the framerate of the program and represented a critical problem. The solution to solving this issue was to address the triangle count by creating a new grass model. Originally it was planned to use a free Unity asset as a simple substitution, however because there were none available, an original grass model was created in Blender[12]. The design of this model focused on being 2D and simplistic to make it easy to apply texturing. The 2D factor reduced the triangle count considerably and improved performance(proof shown in Implementation section).

Star Interaction System

The Star Interaction system is a system proposed to visually display the telescope's data points within the 3D space. The player should be able to interact with the points in the sky to gain visual information on that data point. In addition, this system can be used to label stars and virtually label any point in the sky. A rough design for this system is shown below:


Prototype Sky Interaction Visual

Position placement is similar to how stars are placed in the star system script, but will need conversions.

Equatorial and Cartesian Coordinates

- Consider the unit sphere ("unit": i.e. the distance from the center of the sphere to its surface is r = 1)
- Then the equatorial coordinates can be transformed into Cartesian coordinates:
  - $x = \cos(\alpha)\cos(\delta)$
  - $y = \sin(\alpha)\cos(\delta)$
  - $z = \sin(\delta)$
- It can be much easier to use Cartesian coordinates for some manipulations of geometry in the sky

(x, y, z) converted from RA & DEC
Area with Data(Box Collider)
(x, y, z) converted from RA & DEC
Area with Data(Box Collider)
(x, y, z) converted from RA & DEC
Area with Data(Box Collider)
Star System Barrier (900 m from User)
Raycast
(Will check to see if user is pointing at data point)
User
declination (δ)
Equator
right ascension (α)
z
x
y

Figure 3: Sky Interaction Rough Design Visual

Upon designing the Sky Interaction System, the choice was made to initialize a new raycast[13] using an existing raycast system to select data points. Upon selection, a new user interface would appear to display that data point's information. The data within the data points(along with their position) would be read from a CSV, similar to the "Starfield'' script.

Actually creating the data points was the challenge in this system, most notably the positioning of the data points. Because there was currently no system to convert RA(Right Ascension) and DEC(declination) value[14], a new one needed to be created. That process included using trigonometry and Unity's vector system to manipulate the spawns of the data points.

The overall system reads a CSV full of the data points' positions and their data, initialize all data points in their correct positions as new objects, initialize a new raycast object that detects data points on user input, and gather data point info and send it to the user interface and display it on screen.

Telescope Model Accuracy Update

It was quite obvious that the current Telescope model was out-of-date in both the PC and VR educational game. To fully update the model, the program required modifications to the existing models, the addition of new models, and the animation of the new models.

In order to efficiently update the telescope model, close coordination with the mechanical engineers was required. The method for updating the telescope was to hold multiple meetings with the mechanical engineers to identify missing parts and incorrect parts, acquiring new .STL models of missing parts, using Blender to convert those models to be compatible with Unity using .OBJ, and using the help of the mechanical engineers to correctly place and scale those new models. From there, all that was required was animating the new parts by altering their respective transformation positions, which was accomplished using an existing system. New part descriptions were also acquired from the mechanical engineers.

Virtual Reality Optimizations

The main goal for this semester in regards to the VR simulation was to get it up to date and operational with the updated telescope model and add controls to the newly implemented sky interaction system. This included modifications to the existing control scheme and the addition of new UI elements. The VR selection of parts and stars was somewhat buggy and inconsistent. The design to fix this issue was to create a visual pointer that showed the player what they were selecting and making the selection a button press instead of a continuous loop.

In addition, Oculus Quest 2 support was highly desirable for the portability, as well as ease of development and deployment. The main method of addressing those aspects was to utilize the already existing SteamVR API. By registering a developer account and making small modifications to the Quest 2 and updating the SteamVR api, Quest 2 support(along with any other SteamVR compatible device) is now supported for use in the project.

Addition of Post-Processing and Lighting Effects

The original project did not have any form of post-processing or a proper setup of the lighting. Which became a goal to add to the project to improve the look of the project and hopefully improve the visual aspect of the project.

The plan was to use Unity's standard post-processing api to add the post processing to the program. By testing each and every setting we would be able to apply the correct effects that fit with the program without impacting the performance. These effects included; anti-aliasing with FXAA[17], introduction of shadows, grain, motion blur, ambient-occlusion[18], etc.

Also, we would use Unity's built-in lightmap baker to optimize and visually improve the lighting effects. This would be done by tagging gameobjects that were static to produce a lightmap of baked lighting effects to be used instead of real time lighting.

Oculus Quest 2 Demonstration Project

One valuable asset that the Quest 2 provides is the ability to be completely standalone and wireless. The design plan to support this was to create an entirely new Unity Project and import the existing assets and scenes(much like a copy). The difference being in the project settings; instead of using the high definition render pipeline we would use the universal render pipeline which supports the VR development to apk, replace high poly models with mobile models for performance, utilize the Oculus OpenXR api for VR controls, and create a small animation showcasing the telescope without user input.

Without user input, the program could wirelessly be shown to an audience without explaining the control scheme and without the risk of motion sickness. In addition, it makes displaying the telescope much more efficient and accessible. Also, the Quest 2 is completely standalone with this program, meaning a supercomputer is not required for exhibition.

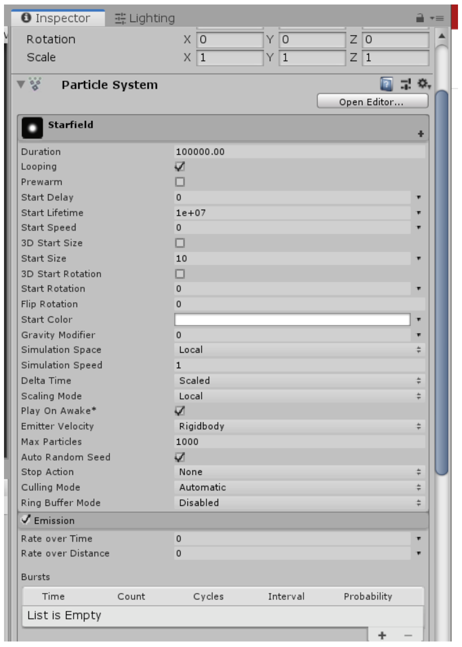
Addition of New Geometry

As a small addition, we wanted to add some new geometry and foliage to the scene to make it seem more lively. The main method of doing this was to use the images of John C. Rudy County Park along with google maps to attempt to replicate the feel of the park with premade free Unity assets. Also, a new main menu screen was added for the implementation of a graphical settings system. The main menu was added, however the settings system was not.

Implementation

Performance Optimizations

In order to eliminate the performance hit of the star system script, a new function to read the star CSV was created with the existing code so the star CSV would ultimately only be read at the beginning of the program. This ultimately broke the Unity particle system that was being used to spawn the stars, so accommodations were made so that the particle system would work with this new approach.



Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Starfield

Duration	100000.00
Looping	✓
Prewarm	☐
Start Delay	0
Start Lifetime	1e+07
Start Speed	0
3D Start Size	☐
Start Size	10
3D Start Rotation	☐
Start Rotation	0
Flip Rotation	0
Start Color	⬜
Gravity Modifier	0
Simulation Space	Local
Simulation Speed	1
Delta Time	Scaled
Scaling Mode	Local
Play On Awake*	✓
Emitter Velocity	Rigidbody
Max Particles	1000
Auto Random Seed	✓
Stop Action	None
Culling Mode	Automatic
Ring Buffer Mode	Disabled

✓ Emission

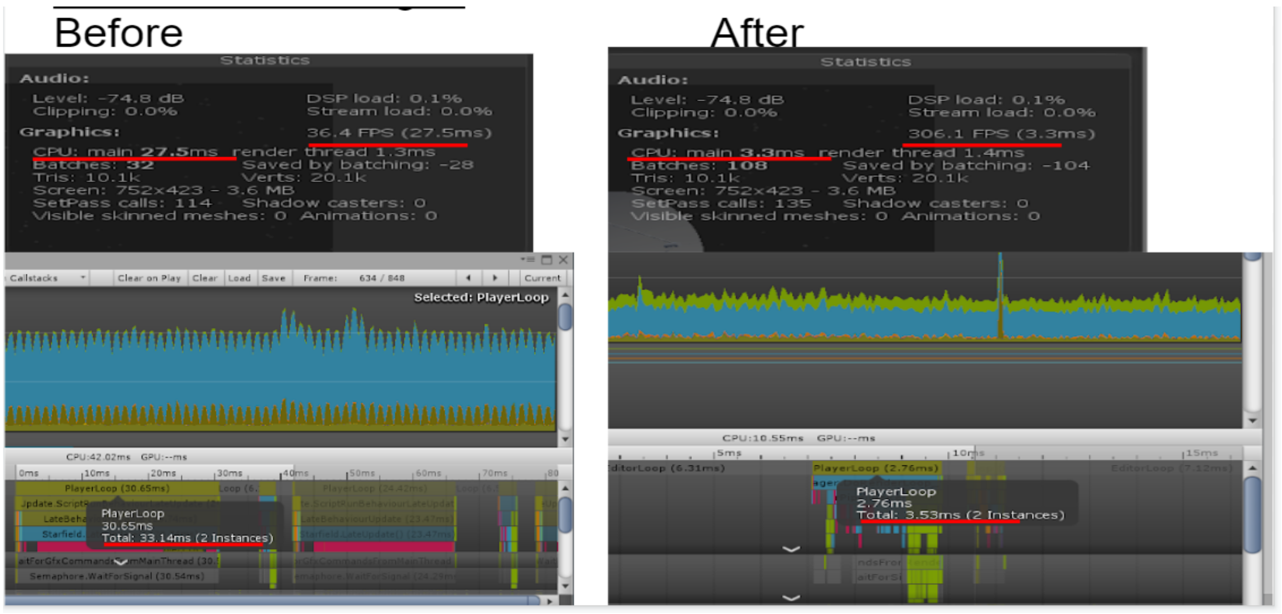
Rate over Time	0
Rate over Distance	0

Bursts

Time	Count	Cycles	Interval	Probability
List is Empty				

Figure 4: Altered Unity Particle System Adjusted for Fixed “Starfield” Script

This new approach improved the runtime cost of the “Starfield” script from 33.14ms to 3.52ms in the playerloop(note that the playerloop includes all scripts within the scene, this change was noted after fixing the “Starfield'' script). It was also able to improve the framerate from 36.4 fps to 306.1 fps, which ultimately reduced the frametime from 27.5ms to 3.3ms on the PC version. (these changes also affect the VR version).



```
Before
Statistics
Audio:
  Level: -74.8 dB              DSP load: 0.1%
  Clipping: 0.0%               Stream load: 0.0%
Graphics:                      36.4 FPS (27.5ms)
  CPU: main 27.5ms  render thread 1.3ms
  Batches: 32          Saved by batching: -28
  Tris: 10.1k          Verts: 20.1k
  Screen: 752x423 - 3.6 MB
  SetPass calls: 114   Shadow casters: 0
  Visible skinned meshes: 0   Animations: 0

PlayerLoop
30.65ms
Total: 33.14ms (2 Instances)
```

```
After
Statistics
Audio:
  Level: -74.8 dB              DSP load: 0.1%
  Clipping: 0.0%               Stream load: 0.0%
Graphics:                      306.1 FPS (3.3ms)
  CPU: main 3.3ms  render thread 1.4ms
  Batches: 108         Saved by batching: -104
  Tris: 10.1k          Verts: 20.1k
  Screen: 752x423 - 3.6 MB
  SetPass calls: 135   Shadow casters: 0
  Visible skinned meshes: 0   Animations: 0

PlayerLoop
2.76ms
Total: 3.53ms (2 Instances)
```

Figure 5: Unity profiler view of before and after “Starfield” script fix. (grass was disabled)

In order to fix the grass models, the most obvious solution was to just find an alternative model that did not have an extremely high poly count like the existing one. However, after searching the Unity assets we were unable to find a sufficient asset that could replace the current one, so one had to be created. To do this we focused on creating a very minimalistic grass model consisting of just a couple of 2d planes meshed together using Blender. After this, the UV mappings of the model were created so the grass texture would be oriented correctly.

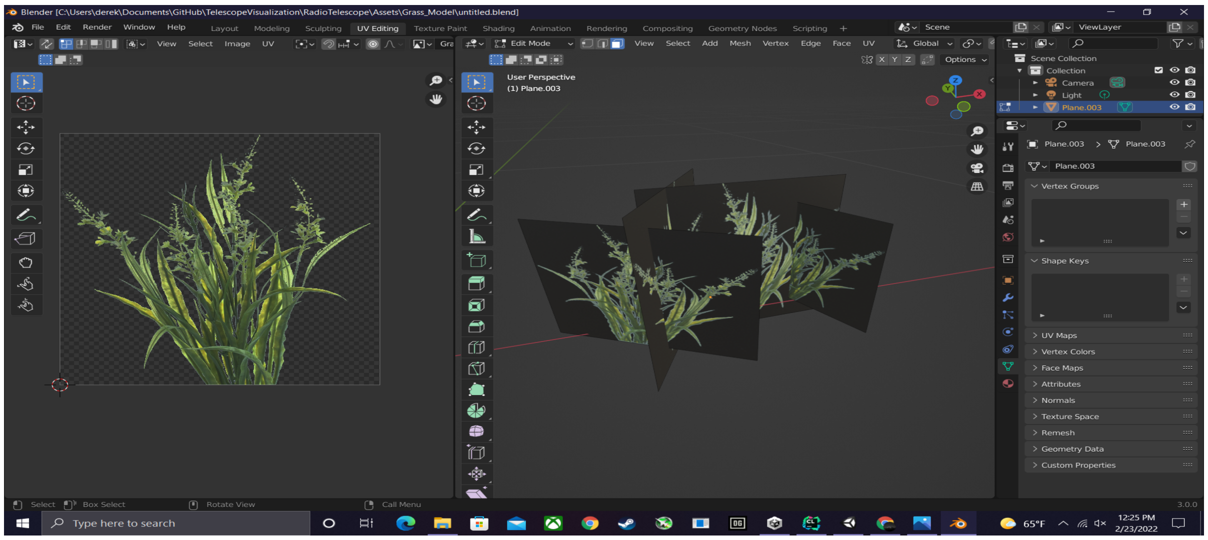
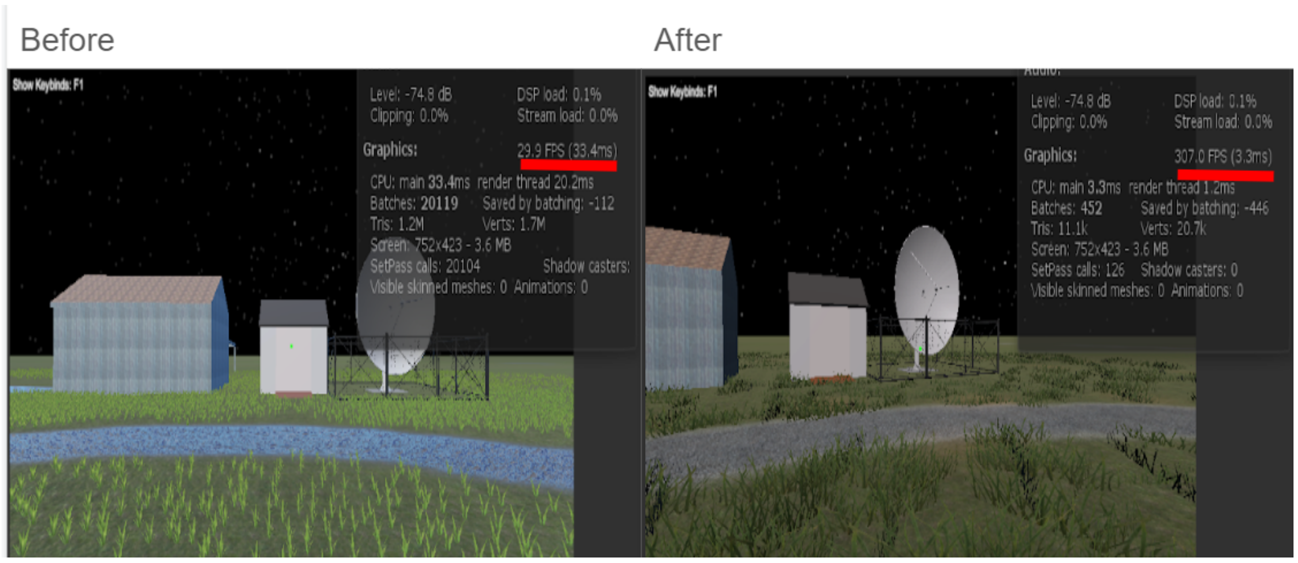


Figure 6: Finished rendition of the grass model and correct UV mappings with an example grass texture.

After creating a finished model of the new grass, the method of inclusion was to use Unity's terrain system to "paint" the model around the 3D environment. However, because the Unity version was outdated and the current render pipeline did not support 3D mesh painting, these models were simply hand placed into the environment. This replacement caused a significant performance improvement as noted in the diagram below.

In addition, a new terrain texture was created in Blender with generated normal maps in order to fix a bug with the pavement being highly reflective like glass.



```
Level: -74.8 dB            DSP load: 0.1%
Clipping: 0.0%             Stream load: 0.0%
Graphics:                  29.9 FPS (33.4ms)
CPU: main 33.4ms  render thread 20.2ms
Batches: 20119     Saved by batching: -112
Tris: 1.2M         Verts: 1.7M
Screen: 752x423 - 3.6 MB
SetPass calls: 20104          Shadow casters:
Visible skinned meshes: 0  Animations: 0
```

```
Level: -74.8 dB            DSP load: 0.1%
Clipping: 0.0%             Stream load: 0.0%
Graphics:                  307.0 FPS (3.3ms)
CPU: main 3.3ms  render thread 1.2ms
Batches: 452       Saved by batching: -446
Tris: 11.1k        Verts: 20.7k
Screen: 752x423 - 3.6 MB
SetPass calls: 126   Shadow casters: 0
Visible skinned meshes: 0  Animations: 0
```

Figure 7: Before and after visualization of the new grass model and new pavement terrain texture. Notably the frametime improved from 33.4ms to 3.3ms. Notably, the triangle count of the program reduced from 1.2 million triangles to a measly 11.1k triangles, which is fantastic.

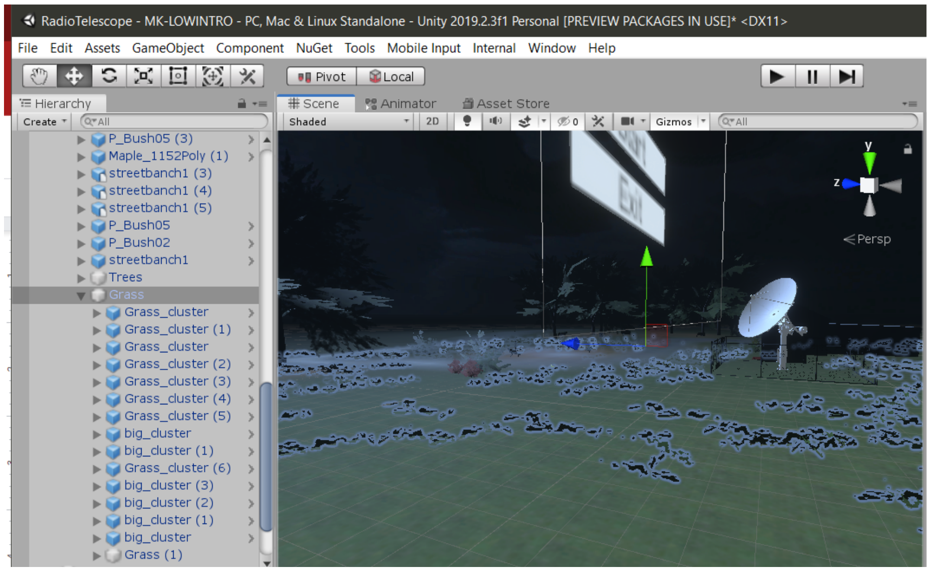


Figure 8: Placement of grass models, grouped with clusters and embedded within the Foliage object.

Star Interaction System

The first implementation of the star interaction system was to create the actual interactable object itself. This object would later be spawned and store CSV gathered information. A triangle design was chosen because it was simplistic to create within Blender and would be able to highlight an area in the sky without being too much of an eyesore. A new raycast object named “SkyRay” was created with an accompanying script, which was modeled after the existing “Highlight_Target” script(raycast). As a little addition, an animation was created so that when the triangle object detects a raycast, it enables an enlargement animation to indicate that the player can select the object.

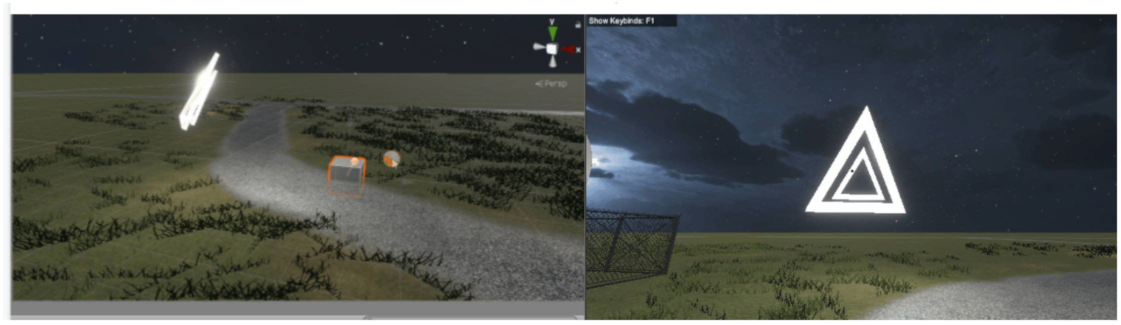


Figure 9: First rendition of the triangle star object. On the left it can be seen that the object is always facing an object of focus(cube). On the right the triangle object is expanded on a mouse hover.

The next and most difficult part of the implementation was reading the CSV data, spawning the triangle objects in their correct position in relation to the "Starfield" script, and storing the data within the newly spawned star objects.

The first step was creating a new "Sky_Spawner" script reads a corresponding CSV named "Sky_Data" using a C# System.IO filereader. This data was then used to spawn the triangle objects in their respective positions and store the information contained within the CSV for that data point.

Firstly, "Sky_Spawner" reads the CSV line by line with each line representing a new triangle object; this was done so that the CSV can be easily manipulated within Excel and is very similar to how the "Starfield" script works. The script then uses the gathered RA and DEC(which are cartesian coordinates) values to place the triangle object in the correct position. However, because Unity is a 3D program that uses polar coordinates, this was incompatible. In order to convert the cartesian coordinates to polar coordinates a new method was created that used an existing sequence of equations (Trigonometry). After obtaining the converted coordinates, the position vectors are then normalized and expanded to the same distance as the "Starfield" stars. After initializing the parent of each triangle object to the "Starfied" object, the position of each triangle object is accurate and rotates with the stars.

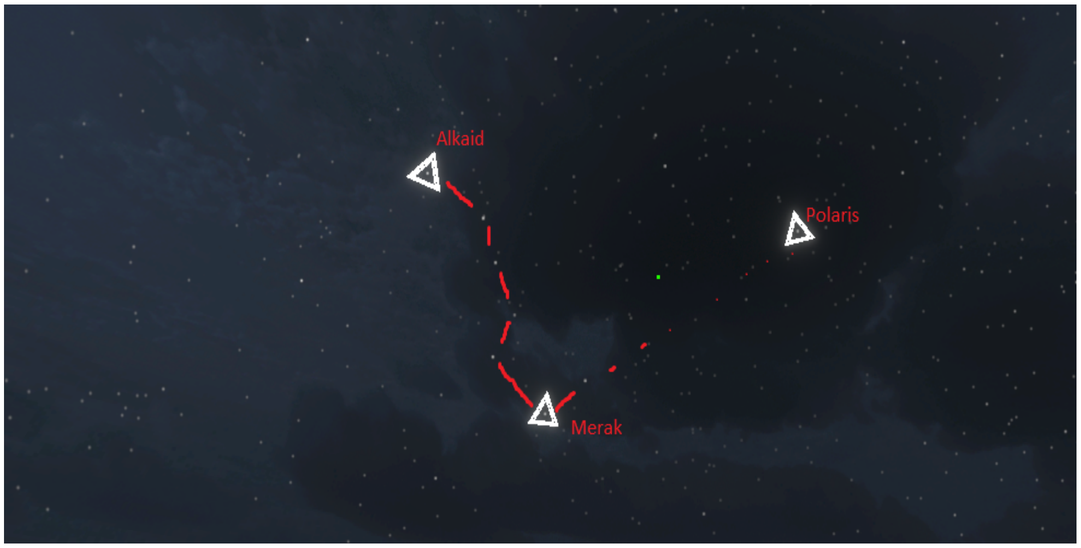


Figure 10: Visualization of triangle object spawns in relation to the star system. Highlighted is the big dipper, with triangle objects successfully spawned on both edges.

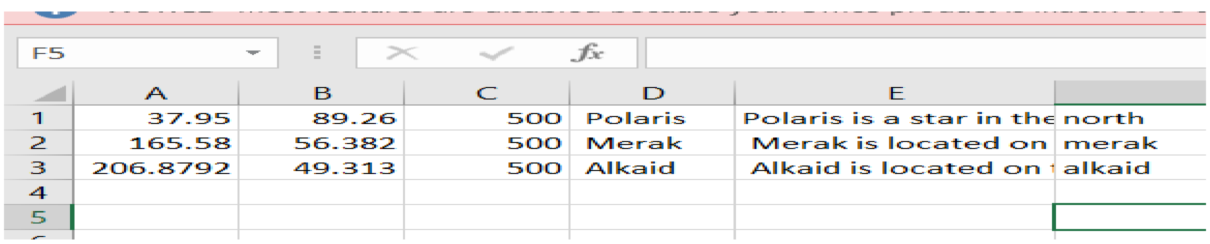
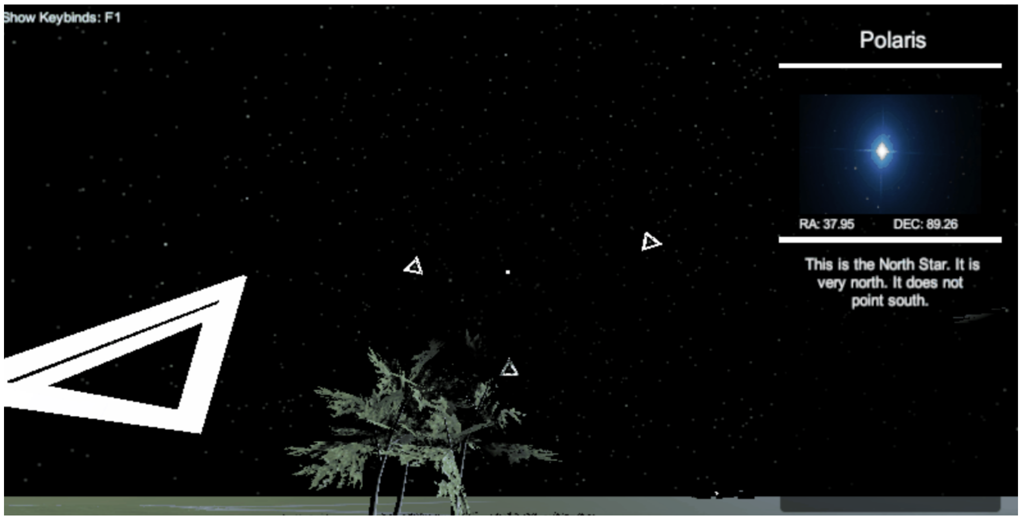

	A	B	C	D	E	
1	37.95	89.26	500	Polaris	Polaris is a star in the	north
2	165.58	56.382	500	Merak	Merak is located on	merak
3	206.8792	49.313	500	Alkaid	Alkaid is located on	alkaid
4						
5						

Figure 11: Excel Sheet of "Sky_Data": RA, DEC, DIST(lightyears), Label, Desc, Image Name.

After successfully spawning the triangle objects in the correct positions, the next goal was to create a user interface that would display the information contained within the CSV and also the corresponding .jpg image. The design of this UI can be seen below in figure 12.

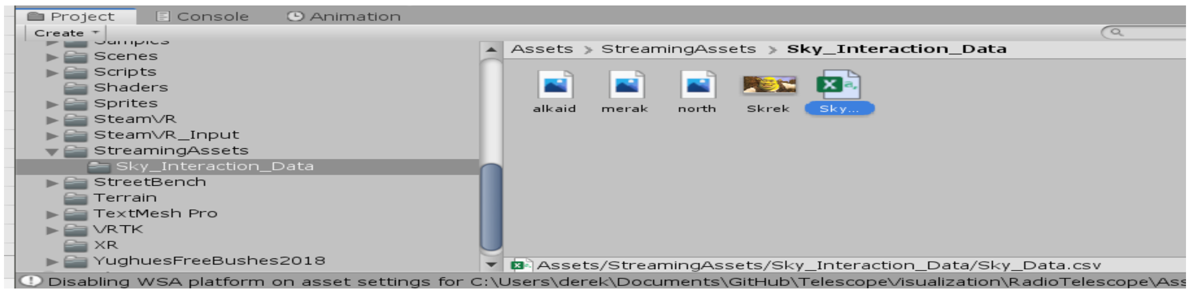


This is the North Star. It is very north. It does not point south.

Figure 12: Visual of sky interaction UI. On the right is the UI display after selecting the triangle object for the star "Polaris". The label can be seen at the top, the corresponding .JPG in the middle, the RA and DEC values right below, and the description is then listed at the bottom.

In order to list all the CSV data onto the GUI, a script named "Star_Object" is attached to each initialized triangle object. This script contains the CSV information when the triangle object is spawned from "Sky_Spawner''. It also contains the logic for when a user hovers over and clicks the triangle object. When clicked, a GUI object named "StarObjectUi'' is sent all the corresponding information, which is then listed into the text objects and displayed to the user.

One complicated part of this was the .JPG image. In order to display the corresponding .JPG image for each triangle object, the .JPG name is searched within a folder named “Sky_Interaction_Data” which contains all the .JPG images and the “Sky_Data” CSV file. After finding the .JPG image, the bytes are added to an image array and displayed on screen.



Assets/StreamingAssets/Sky_Interaction_Data/Sky_Data.csv

Disabling WSA platform on asset settings for C:\Users\derek\Documents\GitHub\TelescopeVisualization\RadioTelescope\Ass

Figure 13: The “Sky_Interaction_Data” folder contained within the “StreamingAssets” Unity folder.

We felt that our client would want to change this information(addition and deletion of data points) so we enabled and utilized a special folder in Unity named the “StreamingAssets” folder. Within this folder absolutely nothing is modified or compiled on a Unity build. What this means is that the user can edit and add to the CSV so that new points are displayed without compiling an entirely new build from the Unity project.

Telescope Model Accuracy Update

In order to accurately update the existing telescope model we met with mechanical engineers to identify; what parts are outdated, what parts are missing, and what parts needed to be removed. After identifying those specifics, we moved along to getting the new part models and removing/modifying the existing part models.

In terms of additions, most of the top of the telescope had to be reimported as much of it was outdated and did not match with the newly imported parts. Those parts included; dish, supports1, supports2, main dish support, frame, embedded systems box, electrical cabinet, and the funnel pipe assembly. All of those parts were given as a .STL file from one of the mechanical engineers. These .STL files were then imported into Blender and then exported as a .OBJ file. Then these files were imported and scaled into Unity. The mechanical engineers were able to provide part descriptions for each new part. After correctly animating each part using an existing expansion script, the model was completely updated.

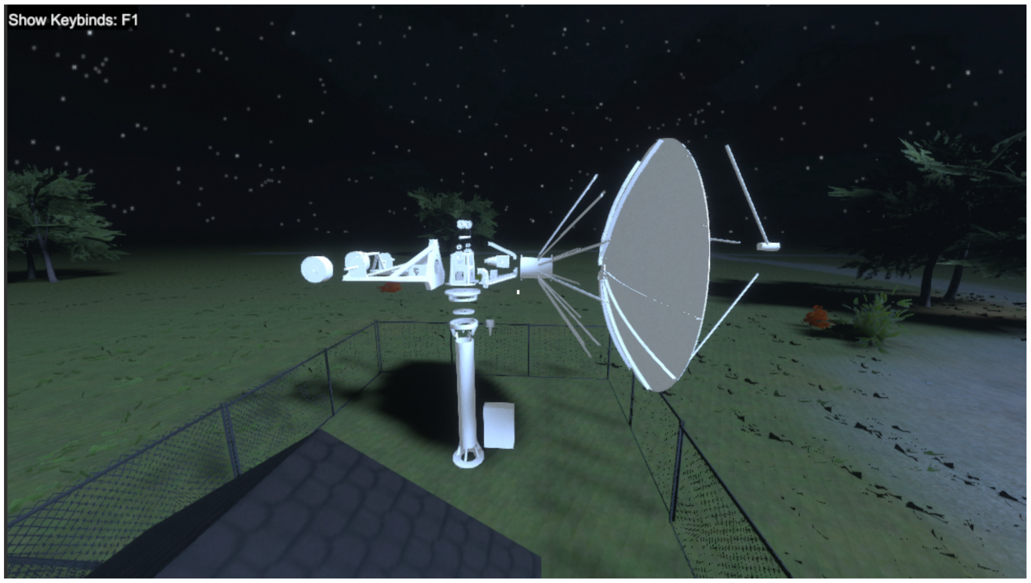
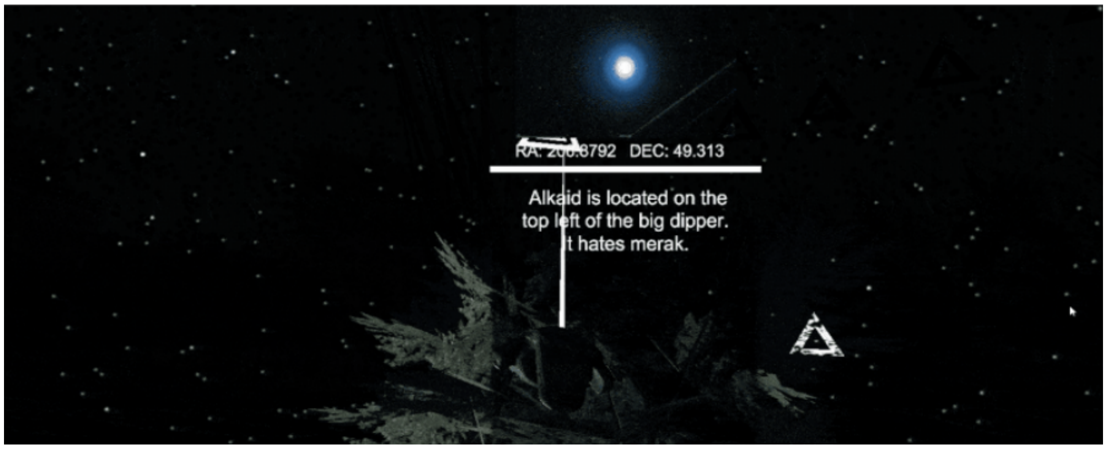


Figure 14: Visual expansion of all updated telescope parts.

Virtual Reality Optimizations

In order to optimize the VR experience, numerous modifications had to be made in regards to the UI and control scheme.

For the VR version, a separate sky interaction GUI was copied and altered to fit within the eyesight of a VR space. In addition, a selection indicator was modified so that it would always be visible for selection purposes. Also, the material/shader was fixed so that it now renders correctly. Also, the control scheme was modified so that the trigger input would only be checked when pressed instead of within the game loop. That meant that the selection process was easier since it would activate only on one button press instead of a button hold. This required a new function and was done mainly as an accessibility modification.


RA: 200.8792 DEC: 49.313

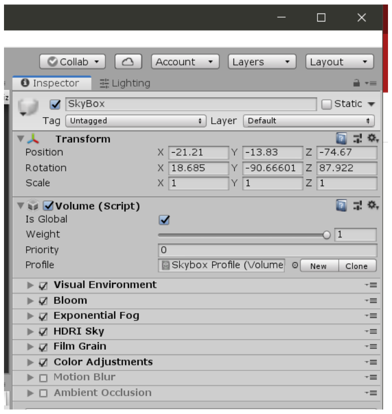
Alkaid is located on the top left of the big dipper. It hates merak.

,

Figure 15: VR controller with line renderer indicator, selection of star object is shown.

Addition of Post-Processing and Lighting Effects

The main implementation of the post-processing effects[15] was the addition of the Unity post-processing api. This versatile addition allowed the quick inclusion of multiple different types of post-processing.



SkyBox ☐ Static
Tag Untagged Layer Default

Transform

Position	X	-21.21	Y	-13.83	Z	-74.67
Rotation	X	18.685	Y	-90.66601	Z	87.922
Scale	X	1	Y	1	Z	1

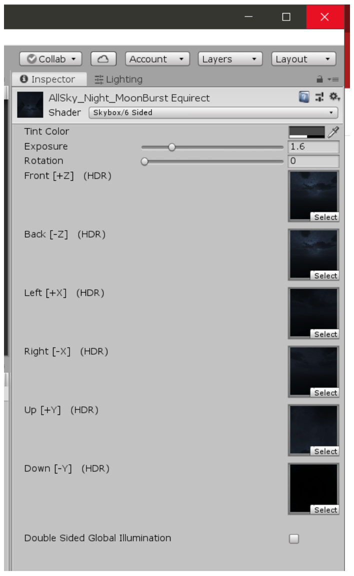
Volume (Script)

Is Global	✓
Weight	1
Priority	0
Profile	Skybox Profile (Volume

- ✓ **Visual Environment**
- ✓ **Bloom**
- ✓ **Exponential Fog**
- ✓ **HDRI Sky**
- ✓ **Film Grain**
- ✓ **Color Adjustments**
- ☐ Motion Blur
- ☐ Ambient Occlusion

Figure 16: A summary of the post-processing volume that was added. The included post-processing effects were; Visual Environment(Skybox), Bloom, Exponential Fog, HDRI Sky(Skybox), Film grain, and Color Adjustments.

In addition to the post-processing effects, a new skybox cubemap was created in photoshop. This consisted of 6 images bound together to form a new skybox for the background of the program.



AllSky_Night_MoonBurst Equirect
Shader Skybox/6 Sided

Tint Color
Exposure 1.6
Rotation 0
Front [+Z] (HDR)
Back [-Z] (HDR)
Left [+X] (HDR)
Right [-X] (HDR)
Up [+Y] (HDR)
Down [-Y] (HDR)
Double Sided Global Illumination

Figure 17: The newly created skybox images meshed together to form a skybox. These images were originally a dark cloudy sky that was darked and altered to give off a less constant color profile in the sky.

Also, shadows and baked lightmaps[15] were added for performance related purposes as well as visual additions. The baked lightmaps were created using Unity's lighting system, the only modifications that had to be made were marking static objects and then baking the lightmaps themselves. Lightmaps are basically a faster way to render lighting by prerendering lightness values into images. Also, a new telescope material was created to match the white paint.

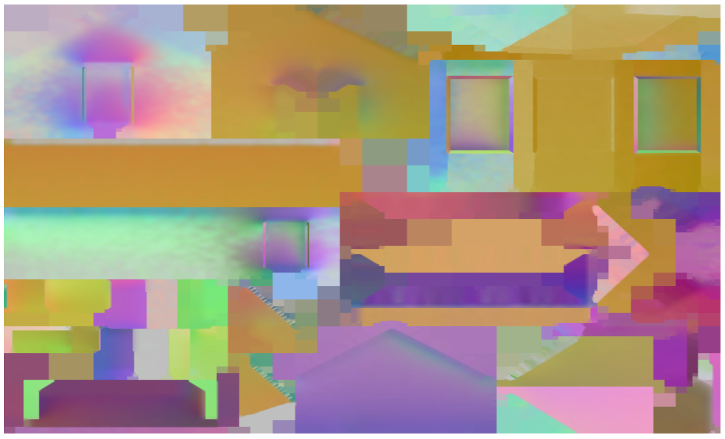


Figure 18: Example of one of the baked lightmaps. These are read and applied to objects as a means of lighting them without doing the computations in real time.

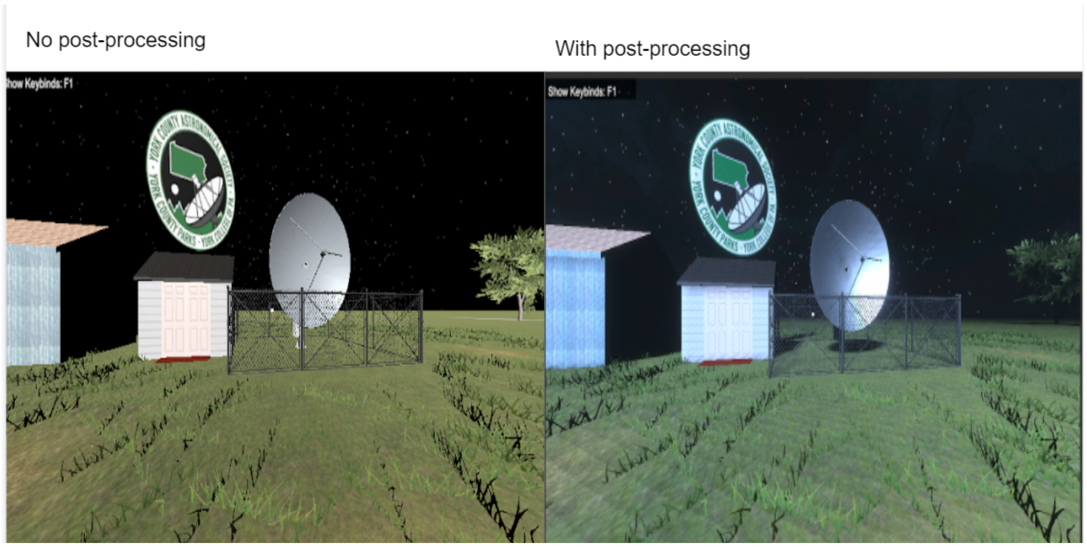


Figure 19: Before and after applying all post-processing effects, shadows, and lighting effects.

After applying and adding all of these effects, using the Unity quality system, two separate scenes were created; MK-HIGH & MK-LOW. MK-HIGH is the high quality version of the mouse and keyboard program(this includes all post-processing at highest quality) and MK-LOW(most post-processing is disabled or at lowest quality).

Oculus Quest 2 Demonstration Project

In addition to the main Unity project, a separate Unity project was created for development of an Oculus Quest 2 apk. This project uses the universal render pipeline and the Oculus OpenXR[16] api to support the Quest 2. Through this project, a small visual demo was created as a quick introduction to VR and the project. This demo was created by exporting the previous project's main assets and then animating a small visual that would display the telescope as well as its parts.

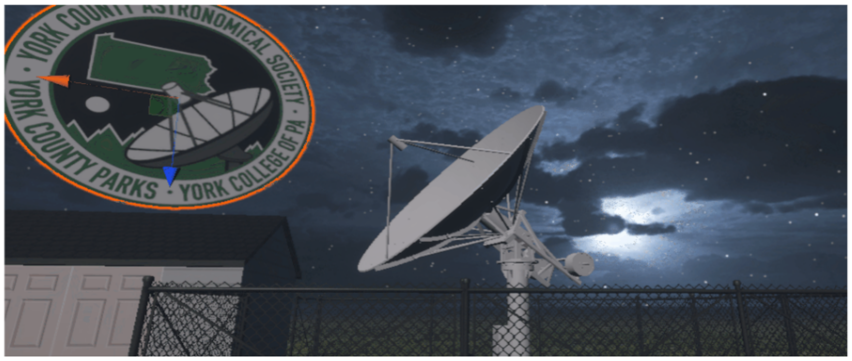


Figure 20: The first rendition of the Oculus Quest 2 apk demonstration.

Addition of New Geometry and Menu

Also, new tree, bush, and bench models were added to simulate John C. Rudy Country Park. These models were obtained from the Unity Asset Store and are 100% free use. Some of these new additions are shown below:

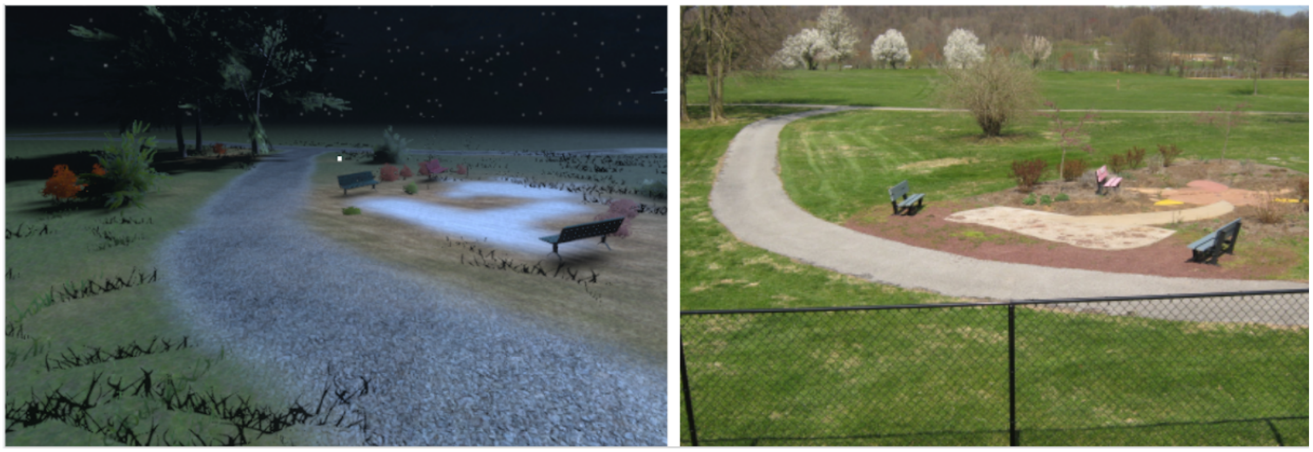


Figure 21: The rendered Unity scene with added tree, grass, and bench models(left). A reference image of John C. Rudy County Park(right).

Lastly, a new main menu was added to the PC version of the program. This was originally going to allow a quality settings adjustment menu, however there was not enough time budgeted to finish this aspect. Instead, there are two builds of the game labeled MK-HIGH(high quality) and MK-LOW(low quality). The only difference between these two builds is that MK-HIGH has post-processing effects turned on and is rendered at the highest quality setting in Unity. MK-Low has no post-processing effects and is rendered at the lowest quality possible in Unity.

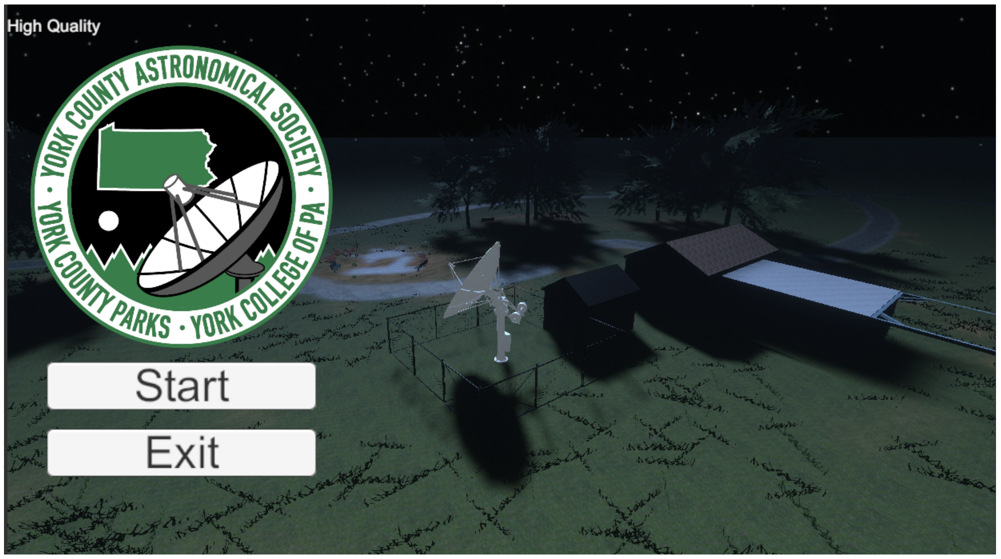

High Quality
YORK COUNTY ASTRONOMICAL SOCIETY
YORK COUNTY PARKS • YORK COLLEGE OF PA
Start
Exit

Figure 22: An added main menu for the PC version of the educational game. The quality version is listed in the top left corner.

Future Work

Update Unity Project to a more stable LTS version

Updating the current project to a new Unity version would open up the project to more features and more importantly fix some of the existing bugs with the VR portion(shadow bug). Also, updating this to a new render pipeline would also improve performance and allow apk development on the Quest 2, this would allow the standalone use of the Quest 2 without any cables, similar to how the Quest 2 demonstration project works. This would require a significant amount of changes to the VR controls and some modifications to the materials and lighting. Most of the scripts and functionality would be unaffected.

Additions of New Scenes and Demonstrations

Time did not allow for a visual demonstration of the telescope within the actual visualization program. A cinematic that goes through the telescope operation and all its parts would be a pretty useful learning tool. Also, with the newly added main menu a settings option could be added on the PC program to adjust the quality settings instead of two separate builds.

Polish of New and Old Systems

We still feel that some additional improvements could be made to both the old and new systems. With the sky interaction system, a program could be built so it is easier to add and delete data points, either through user input or through reading a database of read data points from the real telescope. Also, error checks need to be added to the sky interface system, for instance, a correct field needs to be entered into the CSV for the system to work (what if they don't want a description?).

The VR controls still seem to need some improvements, particularly with the movement of the player character. Currently, the movement is bound to a touch sensor on the VR controllers, which often pops up annoyingly without the meaning to move. Also the movement in general is just disorienting in VR due to the movement sometimes.

Suggested Features That Have Not Yet Been Implemented

Many proposed features both from this semester and from previous semesters have never been implemented. This includes but is not limited to; inclusion of the sun, inclusion of the moon, a daytime mode, a daytime to nighttime transition, a telescope the player can pick up and zoom with, tracking of a data point with the telescope, a control tutorial, transitions, and a sound system.

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

[15] Post-processing https://en.wikipedia.org/wiki/Video_post-processing

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

[17] FXAA
https://en.wikipedia.org/wiki/Fast_approximate_anti-aliasing#:~:text=Fast%20approximate%20anti%2Daliasing%20(FXAA,a%203%2Dclause%20BSD%20license.

[18] Ambient-Occlusion https://en.wikipedia.org/wiki/Ambient_occlusion