VR

Assignment 4: Flight Simulator

Released: Feb 19

Due: March 26th @ 4:00 PM

Please start early as this is long assignment with a lot of details.
We simply want to make sure that you have started the assignment, and we want to provide some feedback to you in time. Furthermore, you will be able to see some good designs from other students on Piazza.
REMEMBER TO SUBMIT YOUR EXECUTABLE, UNITYPACKAGE, UnityPlayer DLL, InputManager, AND DATA FOLDER.

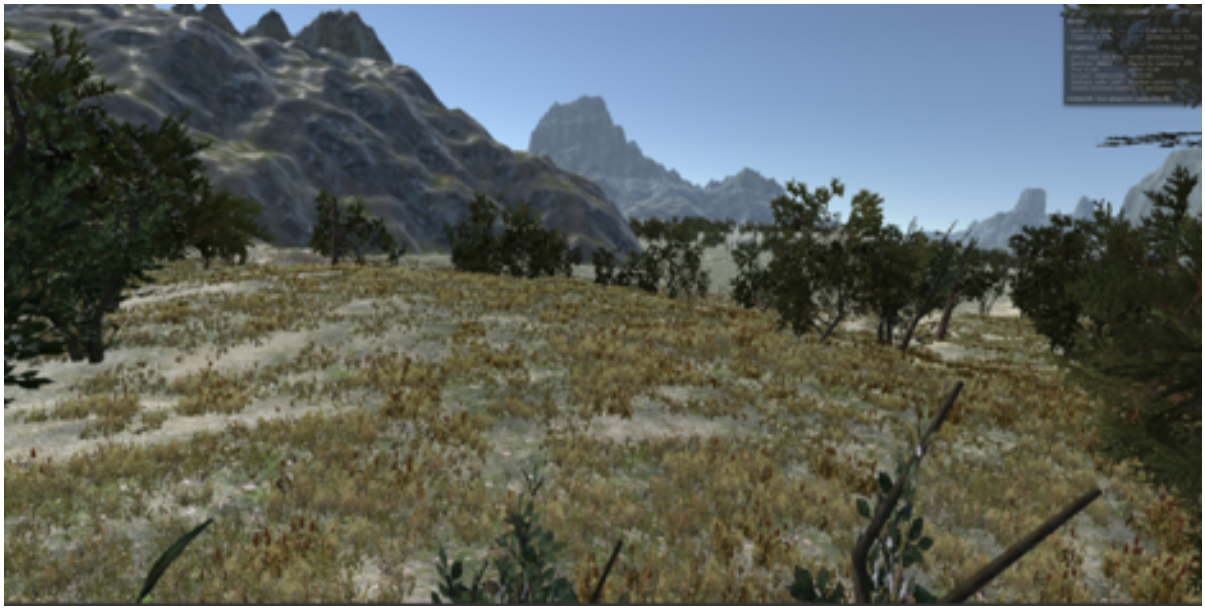
Overview

Welcome to Assignment 4! In this assignment, you are going to build a flight simulator in Unity. You are going to construct a new environment, write script to support your plane, make a UI, and optimize the game to fulfill requirements from the Oculus Best Practices Guide! (What is the Oculus Best Practices Guide? See MP 3.)

PLEASE REGULARLY SAVE YOUR WORK IN THIS MP AS UNITY MIGHT CRASH AND YOU COULD LOSE UNSAVED DATA!

Make sure to read the FAQ at the end of the document.

Part 1: Environment



First of all, we need a unique environment for your flight simulator. The environment you build should have at least a sky, water, and a terrain. Remember, the purpose of this part is to let you play with Unity graphics and make your environment look good. Therefore, you are always welcome to add other things into your scene besides these requirements! **Please go over all of the requirements before you start and take a look at the reference screenshots at the end of the document to get an idea of what we expect.**

All the techniques you need for this part of the assignment can be found online. There are tons of useful tutorials on Youtube too.

1. When working on your terrain, make sure that you use at least 3 real life photos (or concept art) as your reference images, and consider the following questions:

Where should you place the trees? What is the height of the trees compared to the mountain? How does your fog look, e.g. color, density? etc.

2. Use a Skybox for the sky (other than the default). It is even better if you combine it with other techniques, such as importing the cloud meshes into the game or write a script to simulate the moving of real clouds.

3. Use Unity's Terrain Editor (or some other tool) and create a terrain, at least 1000 x 1000 units, in the game (Similar to the terrain shown in the picture above). The terrain should **at least**:
    a. Have a variety of elevations (canyons, mountains, hills, etc.)
    b. Have 3 different types of textures (search online or use the one in the package: /assets/import/environment)
    c. Apply a normal map for the each texture.
    d. Have plants (/assets/import/environment), with shadows

    You should create your own terrain. Do not directly import a terrain mesh.

4. Unity's fog can be controlled in "Lighting". Play with it, such as the color and density. However, you must be very careful that effects can be extremely computationally expensive and ruin your game. You are likely to come back to fix your fog effect when you optimize your game.

5. Add some water to your scene (/assets/import/environment). It can be an ocean, a lake, or a stream.

6. Use your imagination and make the scene look good!

Part 2: Flight

A flight simulator must take flight. You are going to build a plane in this part and write scripts that allow your plane to move, rotate, and shoot. You can start with a simple cube, and later create your own planes and replace this box with your modeled plane.

The next step is to write some scripts to make the plane fly.

1. You should use an Xbox controller instead of keyboard. Of course, you can implement both of them during your development, but we are only going to test the Xbox controller version.

2. Your plane should be able to accelerate/decelerate, and yaw, pitch, and roll. You should implement these functions in your own way with the Xbox controller and Oculus Rift. Think carefully: What is the best way to rotate the plane to minimize discomfort? Should you use buttons or sticks? How fast should the rotation be? Remember to consult the Oculus Best Practice for suggestions.
    a. You can also consider existing flight games and try to replicate their controls.

3. Ray Casting is a very interesting technique. We are going to use it as the weapon of your plane, so that when you press the right trigger, your flight can “shoot” a ray. You should decide where the ray should shoot from.

4. Your simulator looks good, but we can make it better! You should scatter some spheres in the sky, so when your flight shoot at a ball, it should disappear. Also, these spheres should reappear after 5s after you shoot them. You decide the texture and the color of the spheres, but these spheres must be transparent, e.g. transparency from 0.4 to 0.7 is acceptable.

Part 3: Make it a Game!

UI & Tutorial

Interface is an important part of a program. Before implementing, think carefully: should the UI/Tutorial follow head movement? Should the UI/Tutorial stay on the ground? How to present your UI/Tutorial so that people can see it in the Rift and don't feel interrupted? Remember to consult the Oculus Best Practices Guide and share ideas with your teammates. Remember to ground your UI in world space. Don't attach it to the camera If you are still confused, think about the games you have played before, and check this.

1. You should have a start menu, so that players can start by selecting that option or press a specific button.
2. Your Tutorial should explain to your players how to play the game. It can be either in game or in the sub menu, or both.
3. Your UI should show the score to players. After hitting a ball in your scene, your score should increase. There are multiple ways of displaying the score.

It is easy to implement a UI/Tutorial (some simple words), but in order to receive full points, your UI/Tutorial needs to be clear and self-explanatory. If your user doesn't know what to do when looking at your UI/Tutorial, then it is probably not a good design. Ask other people to try your design if possible. If you think words are insufficient to express your ideas, you can choose to display images to your users instead. (Is audio a good choice?)

You should assume that the user doesn't know anything about Xbox controller.
You should assume that it is user's first time using Oculus to play the game.
You can assume that the user knows something about video games.

Please devote some effort into the UI and menus, as you can easily transfer these skills to your final project.

Sound

Sound is essential to a simulation. Add sounds to your simulator, such as ambient

sound, engine noise, or background music. The sounds should enhance the experience.

Overall Design

"Overall Design" is a vague definition, but we are going to evaluate it along the following dimensions:

1. The detail of your scene. A generic terrain with very few trees and low quality textures will lose you points (unless it is your art style). Check Piazza if you are still confused. Please also note that sometimes screenshots cannot reflect the quality of your scene. Put simply, just make your game look good!
2. Your interface/instruction system works as expected: a first time player without any knowledge of your simulator should be able to know how to play the game.
3. The overall experience of the flight: How is the rotation? What is the limit of the speed? Is collision implemented? Is my plane acting like a real plane (not a car, a box, etc.) (hint: it is difficult for a plane to perform one kind of rotation.)?

"Extra" credit opportunity (20 points):

Remember the ray cast in part 2? Now, you are going to replace the ray cast with real missiles. When pressing the fire button, your missile will be fired from your plane to the target, and when they hit, the ball will disappear.

These factors will be considered when giving you the extra credits, but generally, the more realistic the better:

1. The speed of the missile: It should not be linear.
2. The shape of the missile. You can use a box, but be careful about the size of the box.
3. It is not necessary to implement the explosion effect, but you will rock if you do it!
4. Add trails to your missile?

Optimization and Comfort

Our lab computers are powerful, but the Oculus is very demanding on hardware.

You should optimize your game so that the frame rate rarely drops below 60 FPS (frames per second). According to the Oculus Best Practices Guide, we need a stable

frame rate to maximize comfort. If your simulator has overall FPS above 60, and only very rarely drops below then you are probably fine. Unity has an in-built FPS counter. In the game screen, at the upper right, click “stats”, and a window with game statistics, FPS included, will pop up.

You will need to consider the following strategies to increase the frame rate:

1. Delete some of your assets or lighting in the simulator.
2. Lower the effect (shadow, fog) of your scene
3. Check pixel error of terrain (see terrain settings)
4. Are there some assets in your scene that have too many polygons?
5. See the Unity manual page here

Finally, make sure your controls are comfortable in VR. Are rotations violent, or smooth? How much acceleration do you allow?

Submission instructions: (remember the data folder!)

Step 1: Create a .unitypackage file

1) Save your Unity scene in the Assets folder with the title “CS498HW2”

2) Using the editor, find the created scene in the Project menu

3) Right click on the scene and select Export Package…

4) Export the file using default settings (“Include dependencies” should be checked by default)

Step 2: Create a standalone game build

1) Go to Edit → Project Settings→ Player. Make sure the “Virtual Reality Supported” box under Other Settings is checked.

2) Go to File → Build Settings

3) Click “Add Current”. This will add the current scene to the build. You must have saved the scene to the Assets folder for this to work (you should do that anyways).

4) Hit “Build”. Save the project to C:\Users\student’s netid\project name, rather than your networked folder.

5) This should create an executable (.exe) for running the build, as well as a folder containing your scene data. Make sure this executable runs correctly on the Rift before submitting.

Step 3: Copy the Input Manager

1. Shut down your project, and navigate to Your_Project_Folder→ProjectSettings
2. Copy the “InputManager.asset” file, and copy it to your submission folder. This will allow us to replicate any new gamepad buttons or joysticks you mapped.

Step 4: Zip the files and submit them through Compass

1) Create a zip file containing 5 items:

a) The .unitypackage created in Step 1

b) The .exe, .dll, and data folder created in Step 2

c) The inputmanager.asset copied in Step 3

d) A README.txt file containing any special instructions or notes you think are relevant for evaluating your assignment.

e) Your reference photos that you modeled your terrain after

2) Name the file by separating NetIDs with underscores._cs498sl_HW4.zip EXAMPLE: If john1 and carmack2 worked together, the file should be called john1_carmack2_cs498sl_HW4.zip

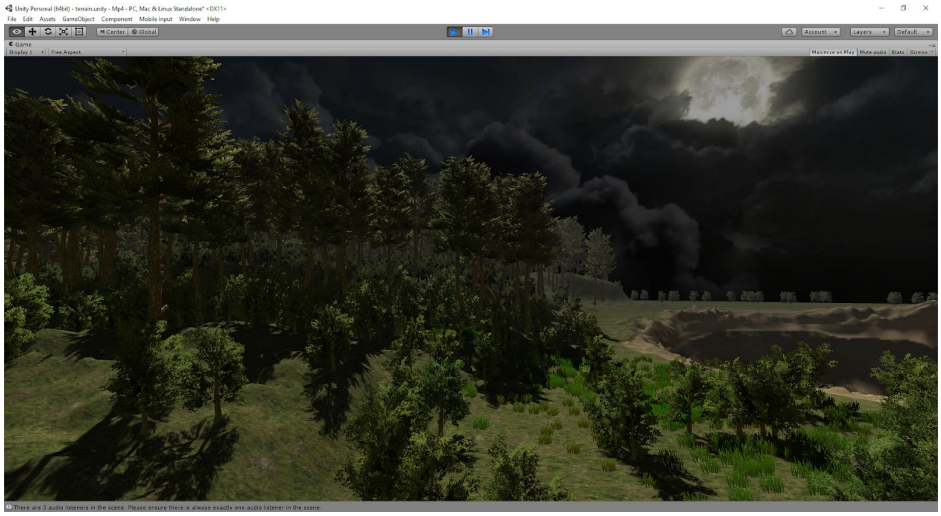
DO NOT SUBMIT YOUR ENTIRE PROJECT FOLDER

REMEMBER TO ALSO SUBMIT YOUR UNITYPACKAGE, AND DATA FOLDER.

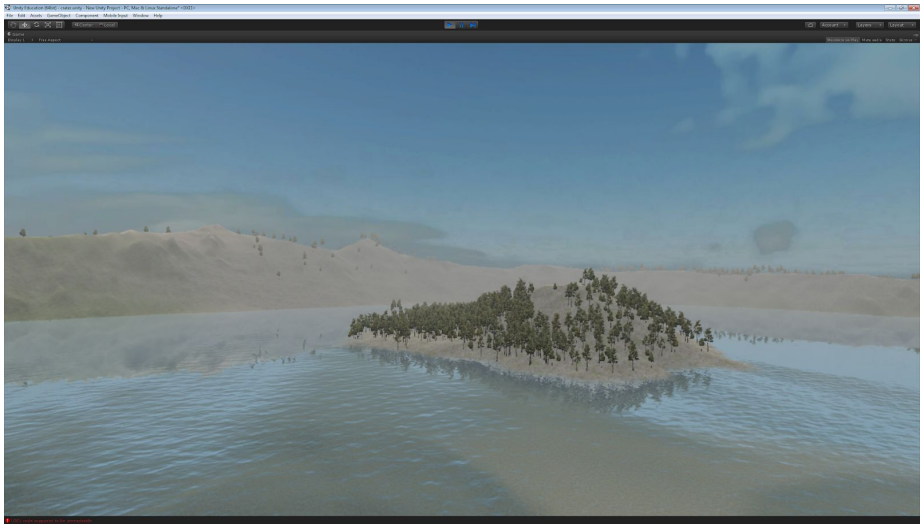
Sample terrain screenshots

Fine

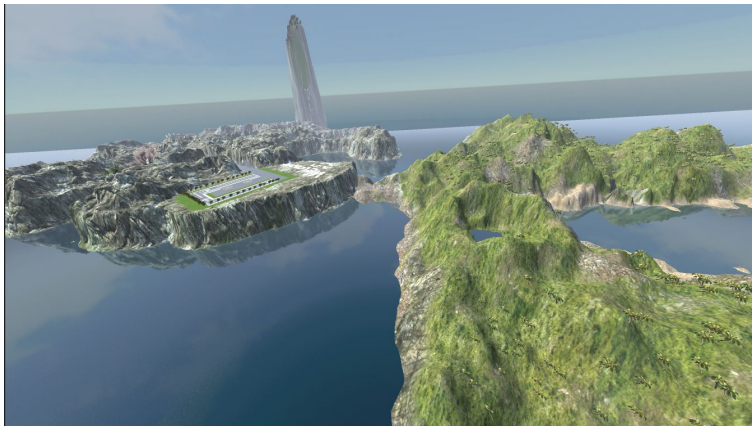
Scale seems a little off in some places but good otherwise



Good



Acceptable



Good

Q&A

Q: Can you debug my code on Piazza?
A: NO

Q: There are not enough textures and maps inside the package provided by Unity.
A: Google and download the maps and textures. Also check the Unity asset store.

Q: I don't know what a map is
A: Google

Q: Why there are there points for optimization?
A: When people start to play your VR game and your game's FPS makes them sick (Yes, 'some' game companies usually mess up on this point), their interest in the game will greatly decrease, especially when there are tons of other alternatives out there.

Q: Can I develop them in Unreal/CryEngine instead of Unity?
A: Good Idea! You are highly encouraged! You can learn some Unreal/CryEngine stuff and develop in the new environment. Note that Unreal's coding standard is different from the C++ programs we write. And in fact, a large majority of people who uses Unreal only uses its Blue Print visual script, which simplifies program's logic into nodes, so you do not need to "write" any code. Read this article before considering learning new stuff. As for CryEngine, we do not recommend it to people who has deadline of around two weeks -- it is not that friendly to new users. However, if you are more experienced in CryEngine, you are more than welcome to use CryEngine.

Rubric

Name	Points	Description
Skybox	5	Game has a custom skybox
Terrain and Details	30	Terrain is at least 1kmx1km, and is detailed and varied
Fog effects	5	Terrain has some fog effects
Water	5	Terrain has some amount of water
Reference photos	10	Zip has three reference photos that influenced terrain
Plane	5	Game has a controllable plane
Acceleration/Deceleration	10	Plane can accelerate and decelerate comfortably
Rotation	10	Plane can yaw, pitch, and roll
Raycasting	10	Pressing a button shoots out a raycast from the plane
Spheres	5	There are spheres around the terrain
Shooting Game	25	Player can shoot spheres, which disappear, then reappear
Tutorial/UI	10	Game has in-world tutorial and UI
Start menu	5	Game has start menu
Displaying score	5	Score is displayed in-world
Sound	10	Game has some element of sound in it
Framerate/Comfort	50	Framerate rarely drops below 60 fps, and plane acceleration, deceleration, and rotations are comfortable
Missiles	20 EC	Plane fires physical missiles, instead of raycast
Total	200+20	