Optimizing performance when using physics in Virtual Reality

Advanced Tools

Saxion University of Applied Sciences

06.07.2022

Teacher: Yvens Rebouças Serpa

Name: Mihai Dascalescu
Student number: 463257

Table of Contents

1.Introduction 3
2.Problem Statement 3
3.Developing the platform for testing 3
3.1 Optimization Techniques 3
3.2 Basic project set-up 4
3.3 First Prototype 4
3.4 Second Prototype 7
3.5 Third Prototype 9
4.Testing and Result Analysis 11
4.1 Testing tools and set-up 11
4.2 Results and analysis 12
5.Version Control 16
6. Conclusion 17
7. References 17
8. List of Figures 17

1. Introduction

Because Virtual Reality technologies are constantly evolving in both hardware and software, the demand for Virtual Reality applications has consistently grown on the market (Marr, 2022). That is why it is important to analyze the current state of the technology and seek ways to improve the workflow of the developers.

This paper aims to present a way to optimize performance when using physics objects for Virtual Reality. To showcase this, Unity Engine was used, and Oculus Quest 2 was set as a target platform.

2. Problem Statement

Performance is one of the key features of Virtual Reality applications. To validate an application on the Oculus Quest Store the developer needs to verify a list of VRCs “Virtual Reality Checks/Conditions” (Oculus). These conditions refer to locomotion, audio, functionality, audio, and other parts of the application. One of them is also performance, to launch the application on Oculus App Labs, it needs to run at a constant 60 fps (frames per second) and to launch it to the Oculus Store, it needs to run at a constant 72 fps.

Developers need to optimize their applications to reach the standard set by the publisher of the applications. To do this, multiple solutions were tested and will be presented in the following chapters.

3. Developing the platform for testing

In order to test the different optimization techniques, multiple prototypes were set-up.

3.1 Optimization Techniques

The optimization techniques used for this project are:

- choosing the right type of collider depending on the purpose of the objects.
- reducing the polygon count of the mesh.
- using LODs (level of detail) to change the mesh based on the distance of the user.

To showcase how the performance is affected for all the techniques multiple tests were conducted for each of the cases.

3.2 Basic project set-up

The project was set-up in Unity Engine version 2020.3.32f1, this version was chosen because it allows for using the newest versions of the V.R. packages. The V.R. packages imported in the project are Unity X.R. Management, which manages the rendering of the application to the device. And Unity X.R. Interaction Toolkit, which is an SDK, (Software development kit) that helps the developer in creating content for virtual reality.

After the essential packages were imported a basic scene was set-up, where a V.R. controller was created, representing the player controller. Furthermore, with older code and prefabs, hands and rays were set-up, so that the user can interact with the environment and the U.I. (user interface) elements.

The environment consists of a closed room where the physics simulations will take place. It is used throughout the project to keep the same conditions for the testing. Moreover, it is closed so that objects cannot exit it accidentally. The objects could fly over the walls of the room, and they will not affect the performance anymore.

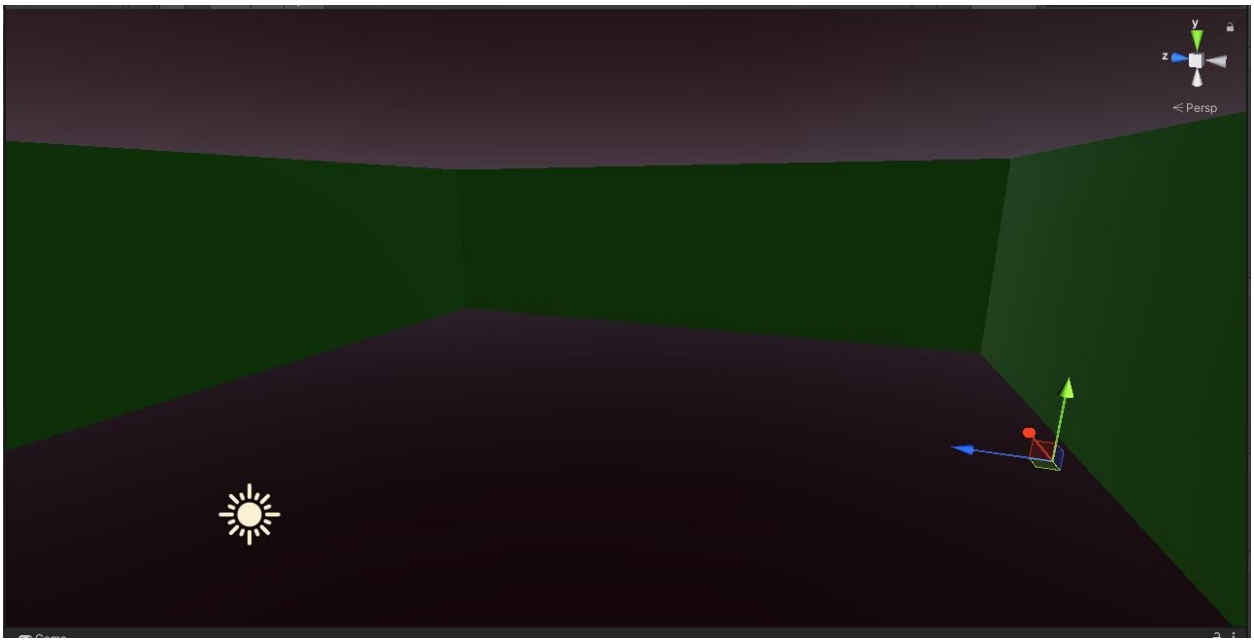

Figure 1 – The room in the scene.

3.3 First Prototype

For the first test conducted, a verification of the performance based on the collider component was required. Furthermore, the collision type of said colliders was tested. The collider components tested were the “Sphere Collider” and “Mesh Collider”. The difference between the two is that Mesh Colliders require more time to calculate compared to the primitive

colliders (sphere collider, box collider, capsule collider), as they can also map collisions for convex shapes, and it automatically takes the shape of the mesh. The collision type can either be discrete or continuous. Discrete collision detection is done by checking when two objects collide at their current position. This approach works for slowly moving objects, but faster moving objects need continuous collision. Continuous collision checks more positions of the object that is moving, managing to give better results when using fast moving objects.

For this prototype, the goal is to arrange a set of tests that can check the difference between the two types of colliders and the two types of collision. To achieve this a “Level Manager” class was made in which an enum is created that stores all the test cases that will be needed.

```
private enum LevelTypes
{
    Blank,
    PhysicsSphereColliderFiveHundredObjects = 1,
    PhysicsSphereColliderOneThousandObjects = 2,
    PhysicsSphereColliderTwoThousandObjects = 3,
    PhysicsSphereColliderThreeThousandObjects = 4 ,
    PhysicsSphereColliderContinousDetectionFiveHundredObjects = 5,
    PhysicsSphereColliderContinousDetectionOneThousandObjects = 6,
    PhysicsSphereColliderContinousDetectionTwoThousandObjects = 7,
    PhysicsSphereColliderContinousDetectionThreeThousandObjects = 8,
    PhysicsMeshColliderDiscreteFiveHundredObjects = 9,
    PhysicsMeshColliderOneThousandObjects = 10,
    PhysicsMeshColliderTwoThousandObjects = 11,
    PhysicsMeshColliderThreeThousandObjects = 12 ,
    PhysicsMeshColliderContinousDetectionFiveHundredObjects = 13,
    PhysicsMeshColliderContinousDetectionOneThousandObjects = 14,
    PhysicsMeshColliderContinousDetectionTwoThousandObjects = 15,
    PhysicsMeshColliderContinousDetectionThreeThousandObjects = 16,
    PhysicsSphereColliderFourThousandObjects = 17,
    PhysicsSphereColliderFiveThousandObjects = 18,
    PhysicsSphereColliderSevenThousandObjects = 19,
    PhysicsSphereColliderTenThousandObjects = 20,
    PhysicsSphereColliderContinousDetectionFourThousandObjects = 21,
    PhysicsSphereColliderContinousDetectionFiveThousandObjects = 22,
    PhysicsSphereColliderContinousDetectionSevenThousandObjects = 23,
    PhysicsSphereColliderContinousDetectionTenThousandObjects = 24,
    PhysicsMeshColliderFourThousandObjects = 25,
    PhysicsMeshColliderFiveThousandObjects = 26,
    PhysicsMeshColliderSevenThousandObjects = 27,
    PhysicsMeshColliderTenThousandObjects = 28,
    PhysicsMeshColliderContinousDetectionFourThousandObjects = 29,
    PhysicsMeshColliderContinousDetectionFiveThousandObjects = 30,
    PhysicsMeshColliderContinousDetectionSevenThousandObjects = 31,
    PhysicsMeshColliderContinousDetectionTenThousandObjects = 32,
}
```

Figure 2 – Enum with test cases.

This test cases were then set-up in a method “OnLevelChanged”, that contains a switch that goes through all the cases from the figure above. This method was made to be mapped to an U.I. screen in the scene, so that the tester can switch the test when he wants to.

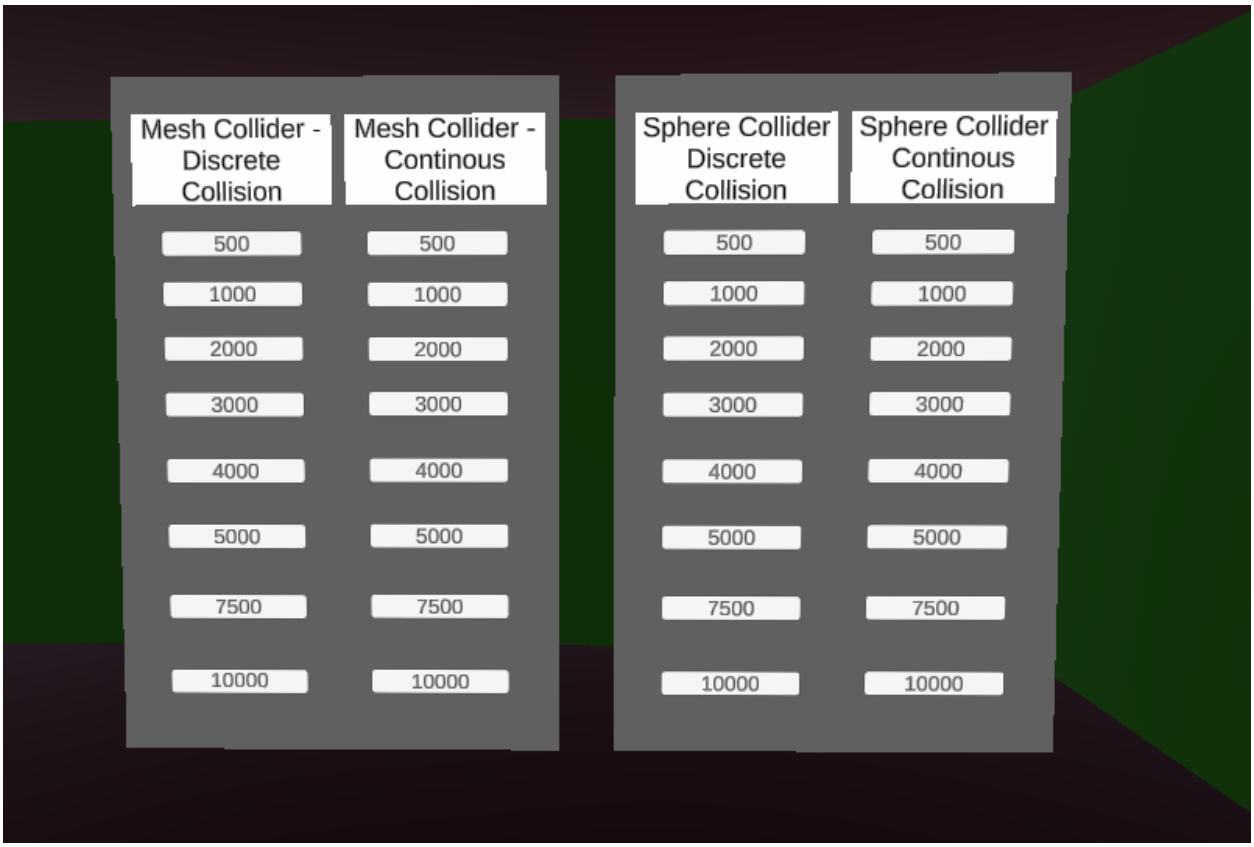

Figure 3 – U.I. for the first prototype.

In this prototype when pressing a button, it will instantiate the number of objects written in the text on the button. To do this, multiple methods were created inside the level class, one that instantiates the prefabs based on the length of the array of objects at a random position, and one that deletes the elements of the array and clears the scene for the next test. It was designed like this, so that no previously instantiated objects exist, when starting a new test.

The prefab used for this prototype can be seen in Figure 4. The components needed for the object to work as intended are either a sphere collider or a mesh collider based on the case, then a rigidbody to react to the physics. Moreover, a custom script called “StartForce” which in the “Start” method pushes the ball in a certain direction, chosen by the developer. In this case on the Z axis by the value of 2. Additionally, the ball has a physics material used to give the ball bouncing. Furthermore, the friction is set to 0 inside the physics material, so that the ball never stops bouncing.

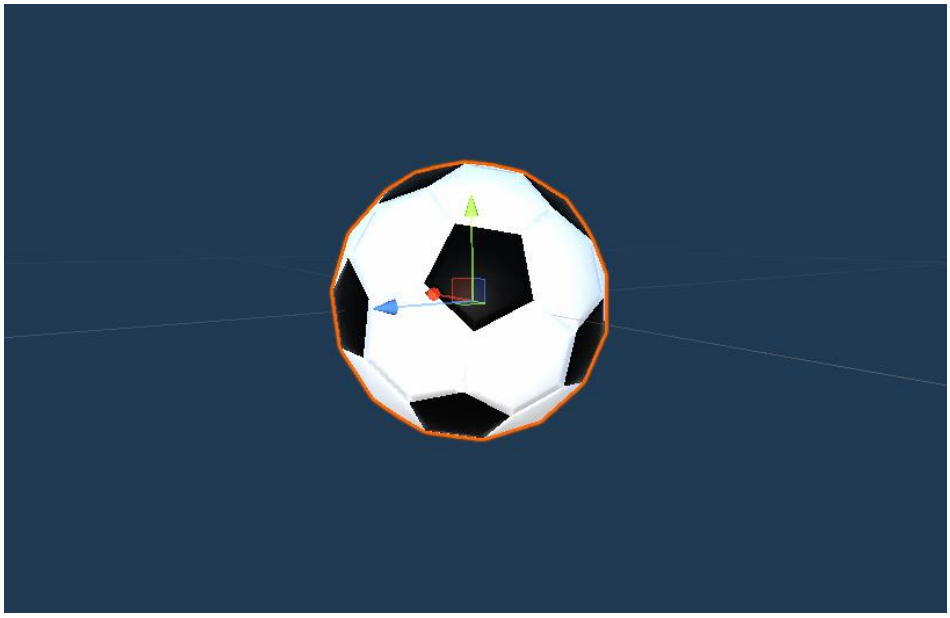

Figure 4 – Soccer ball used as a prefab.

After setting up all the assets for the “Level Manager” class in the inspector the scene was ready to be tested.

3.4 Second Prototype

For the second prototype, the test conducted would be to analyze the performance based on the triangles count of the objects and the type of collider (mesh, sphere colliders). To do this, three different spheres with different poly counts were created in Blender.

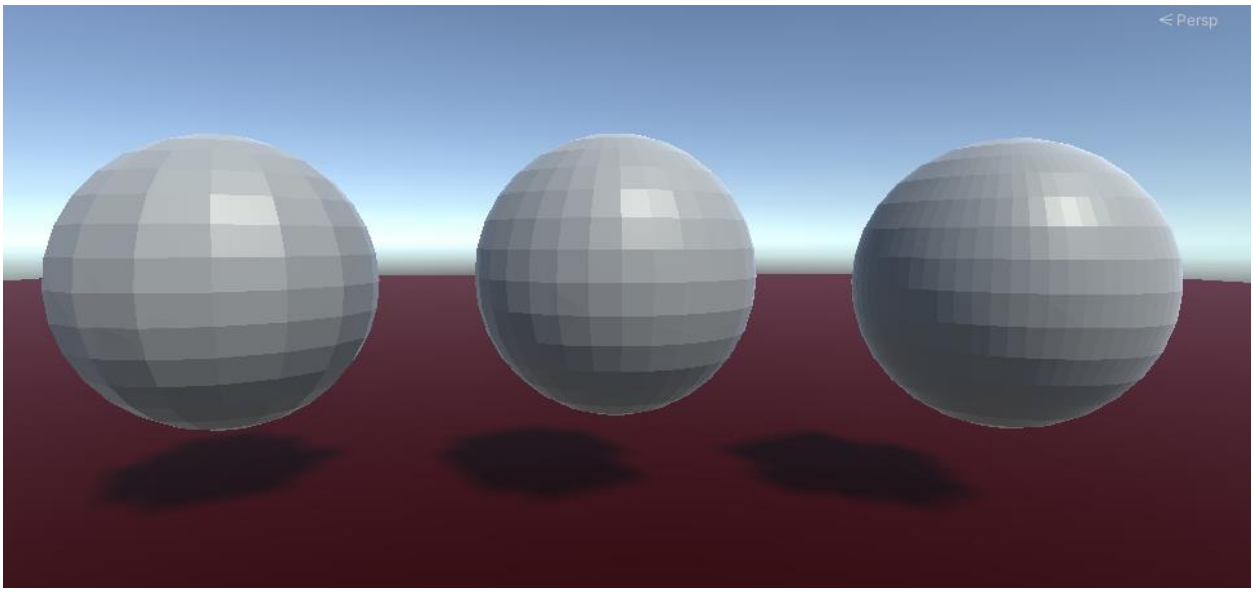

Figure 5 – Sphere Meshes from left to right low poly to high poly.

The low poly sphere has 480 triangles, the medium poly sphere has 1984 triangles, and the high poly sphere has 8064 triangles. After importing them to Unity, prefabs were made for all of them. There were two types of prefabs for each of the balls, one that had a mesh collider and

one that had a sphere collider. Furthermore, different color was assigned to each of the balls, so that they are easy to recognize when testing.

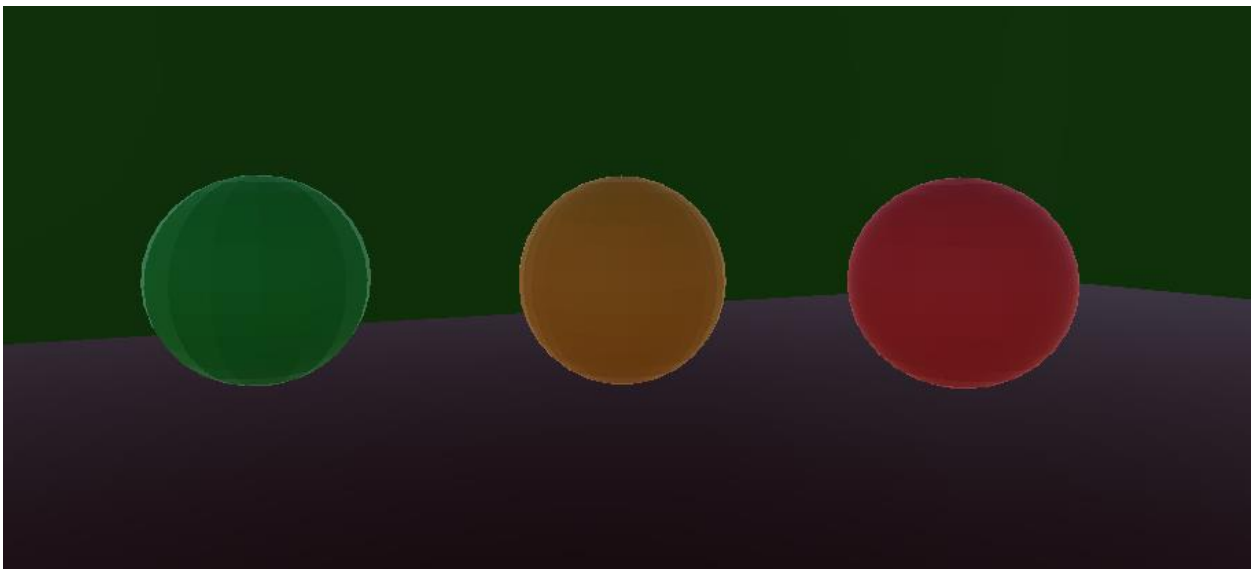

Figure 6 – Ball prefabs with the assigned color from left to right low poly to high poly.

To create the test cases for this prototype a similar approach to the first prototype was used. An enum with the test cases was created, from which a switch statement will define what happens in each of the cases. Furthermore, the cases were then mapped to the buttons of the Canvas that manages the test cases in the scene (see Figure 7).

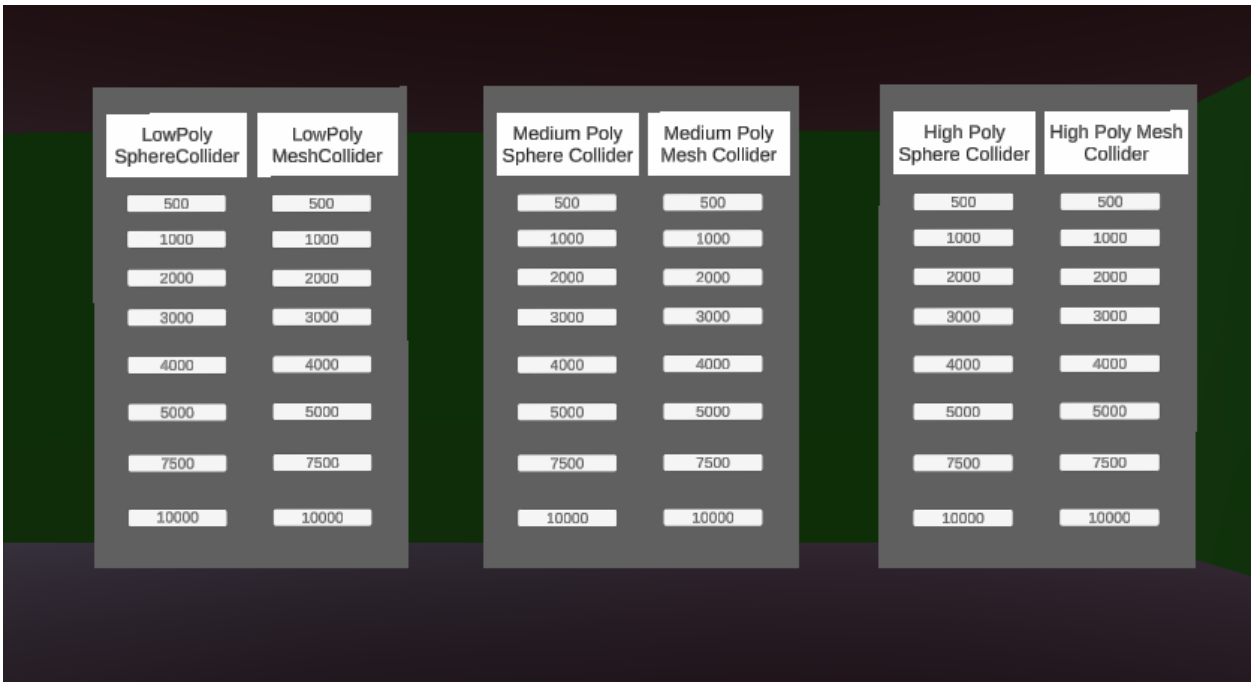

Figure 7 – The canvas for the second prototype.

3.5 Third Prototype

The third prototype was designed to analyze the performance when using LODs (Level of detail) for the balls. “LOD Group” is a component used to manage the level of detail of a game object. This component is used to improve performance by changing the mesh or the texture of an object depending on the distance from the object to the camera. Therefore, by using this component the developer might save a few fps, by using a mesh that has lower polygons or a lower resolution texture.

This prototype was set-up in the same type of scene (see Figure 1) as the previous two. What changed is the level manager object, which, has a different set of test cases (see Figure 9). Furthermore, in Blender, two more sphere models were created with even less polygons, to use as LODs for the “MediumPolySphere” and for the “LowerPolySphere”. The new spheres have 24 triangles and 112, respectively (see Figure 8). The “HighPolySphere” has as LODs the “MediumPolySphere” and the “LowPolySphere”.

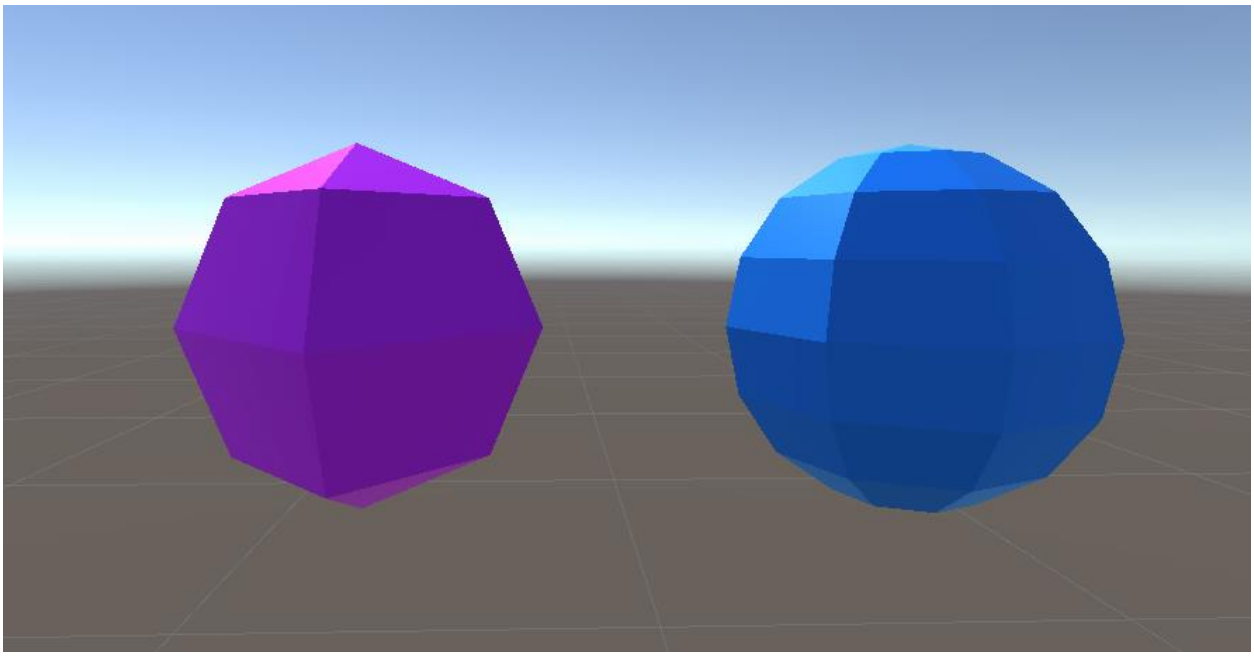

Figure 8 – Low poly spheres, lowest from left to right.

```
public enum LevelTypes
{

    Blank,
    LowPolyLODFiveHundredObjects = 1,
    LowPolyLODOneThousandObjects = 2,
    LowPolyLODTwoThousandObjects = 3,
    LowPolyLODThreeThousandObjects = 4 ,
    LowPolyLODFourThousandObjects = 5,
    LowPolyLODFiveThousandObjects = 6,
    LowPolyLODSevenThousandObjects = 7,
    LowPolyLODTenThousandObjects = 8,
    MediumPolyLODFiveHundredObjects = 9,
    MediumPolyLODOneThousandObjects = 10,
    MediumPolyLODTwoThousandObjects = 11,
    MediumPolyLODThreeThousandObjects = 12,
    MediumPolyLODFourThousandObjects = 13,
    MediumPolyLODFiveThousandObjects = 14,
    MediumPolyLODSevenThousandObjects = 15,
    MediumPolyLODTenThousandObjects = 16,
    HighPolyLODFiveHundredObjects = 17,
    HighPolyLODOneThousandObjects = 18,
    HighPolyLODTwoThousandObjects = 19,
    HighPolyLODThreeThousandObjects = 20,
    HighPolyLODFourThousandObjects = 21,
    HighPolyLODFiveThousandObjects = 22,
    HighPolyLODSevenThousandObjects = 23,
    HighPolyLODTenThousandObjects = 24,


}
```

Figure 9 – Test Cases for the third prototype.

The test cases were then mapped to an U.I. like the ones showed before. The LODs were setup so that each of the ball prefabs has three levels of detail. To make it more complex to calculate for the application the levels of detail were set-up to use the highest level of detail for most of the time, as in the setup from figure 10.

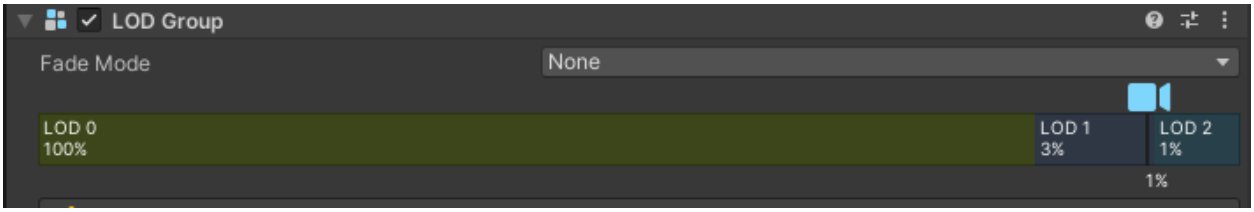

Figure 10 – LOD setup.

Therefore, all the scenes were set-up for testing. However, for a better usability in Virtual Reality, a scene manager object was created, that allows the user to switch freely between the scenes, so that the application is easy to test.

4. Testing and Result Analysis

4.1 Testing tools and set-up

To install the .apk (extension used for executable files for android) on the device, “SideQuest” was used. This application allows developers and users to install unofficial applications to their Oculus devices. The application installed can then be found on the device under the “Unknown Sources” folder.

For the collection of data “OVR Metrics Tool” was used. This tool allows the user to see different parameters for the application that is running. With this tool the fps values were collected. To collect the data, tests for all the prototypes were conducted and the data displayed on the menu in Figure 11 was collected. The data would only be collected after the fps levels stabilized. If the data would be collected when the object were generated, the performance values would be unreliable, because the script that generates it also takes up from the performance.

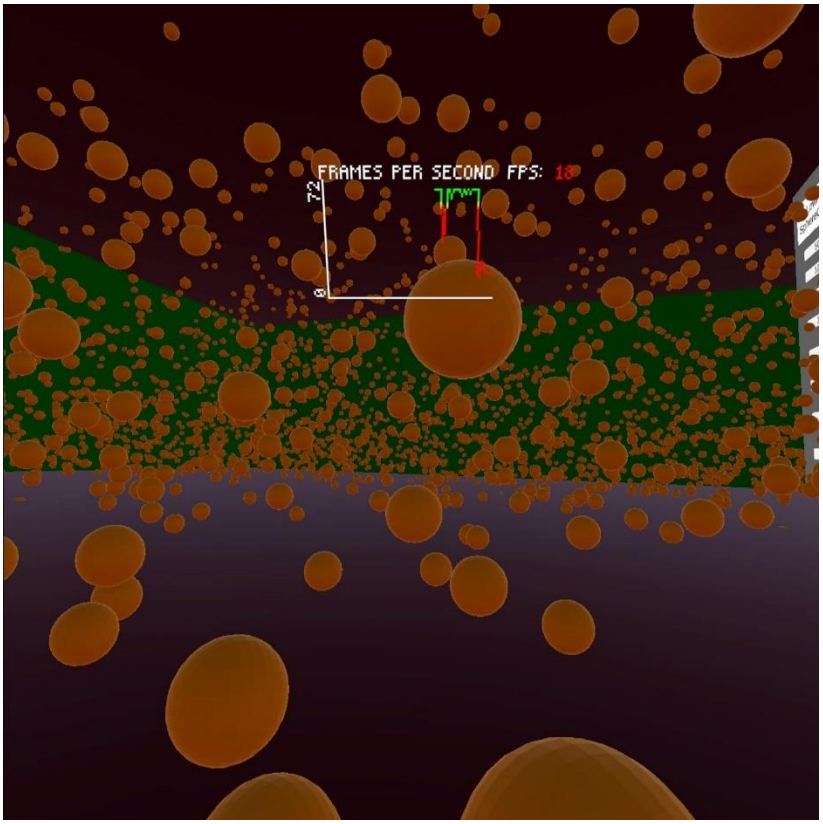

Figure 11- Test setup in device.

In total there were one hundred and four different test cases. The data was collected in an excel document after every test. In the next chapter, the results will be presented in the forms of graphs, and they will be analyzed.

4.2 Results and analysis

The first test consisted in testing the performance for objects with “Sphere Collider” plus discrete collision, objects with “Sphere Collider” plus continuous collision and objects with “Mesh Collider” plus discrete collision and objects with “Mesh Collider” plus continuous collision. The results were as expected, because the continuous collision is more performance consuming than the discrete collision (see Figure 12, 13). Moreover, the objects with a “Sphere Collider” had better performance, than the ones with “Mesh Colliders” because for them more calculations are needed. The comparisons illustrated in the graphs below are:

- between the objects with “Sphere Collider” and discrete collision and objects with “Sphere Collider” and continuous collision (Figure 12).
- between the objects with “Mesh Collider” and discrete collision and objects with “Mesh Collider” and continuous collision (Figure 13).
- between all the types of objects (Figure 14).

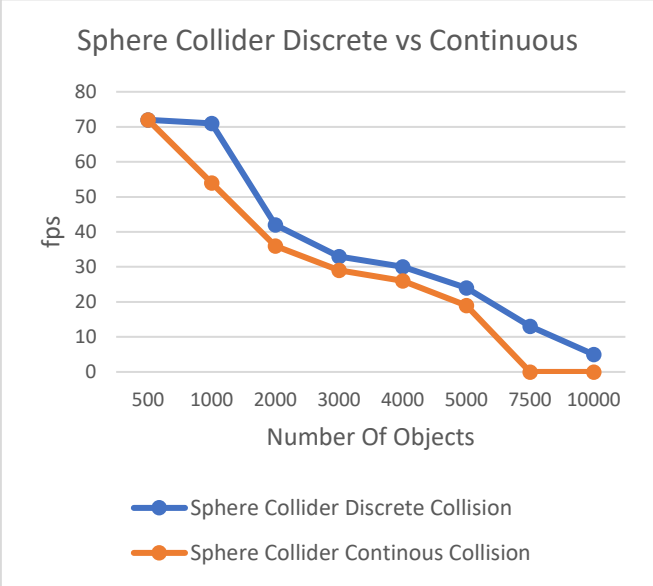

Figure 12 – Sphere Collider Discrete vs Continuous Graph.

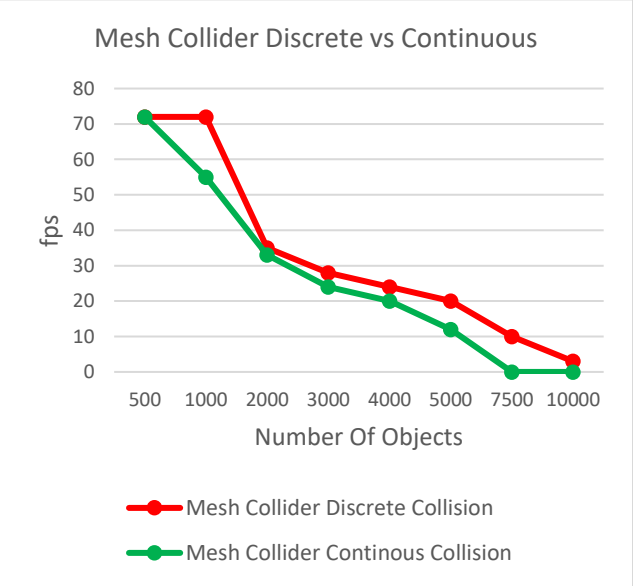

Figure 13 – Mesh Collider Discrete vs Continuous

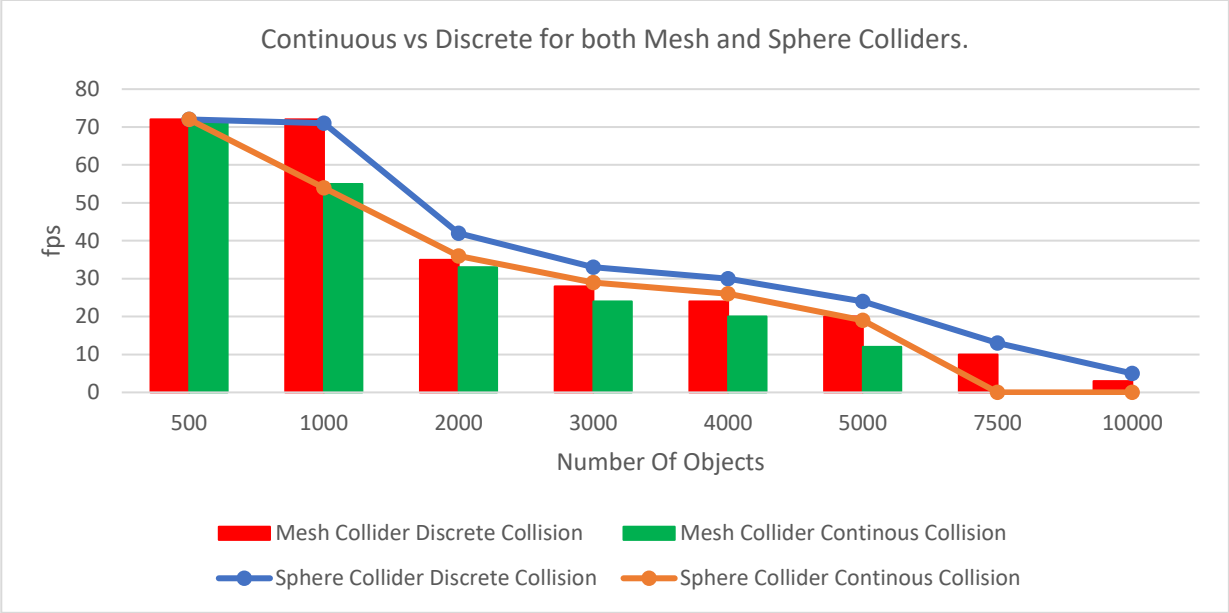

Figure 14 – Continuous vs Discrete for both Mesh and Sphere Colliders.

As it can be seen in both *Figure 12* and *Figure 13* there is a big difference in performance between discrete and continuous collision, in some of the cases the difference in performance was much better for the discrete, managing to achieve more than ten fps in the same test case. The biggest drops in performance were registered after going above one thousand instances of the same object, because the amount of calculation has doubled by a thousand, instead of five hundred. Thus, there are much more interactions as before.

In the *Figure 14* the differences between the “Sphere Collider Discrete” and “Mesh Collider Discrete” are illustrated. While, not as significant as for the other comparison, it is still a difference which for Virtual Reality is important as even one less fps can make the created application be denied publishing. Therefore, it is important to use “Mesh Colliders” when necessary and otherwise the developer should use primitive colliders as they affect the performance less.

The second test was for the second prototype, in which the performance based on the triangle count of the mesh and of the type of collider was tested. For this test the expected results were that the object with the least triangles will have a better performance and that the objects which had a sphere collider instead of a mesh collider will also yield a better performance. The expected result is confirmed by the data collected from the testing.

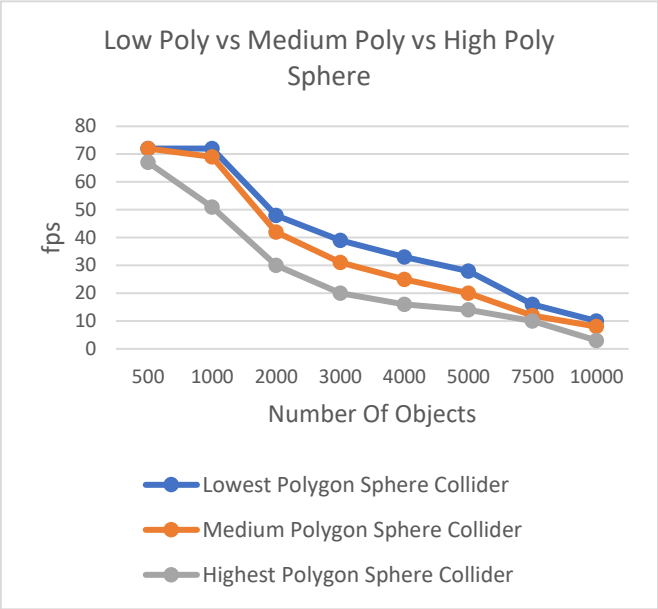

Figure 15 – Low Poly vs Medium Poly vs High Poly with Sphere Colliders.

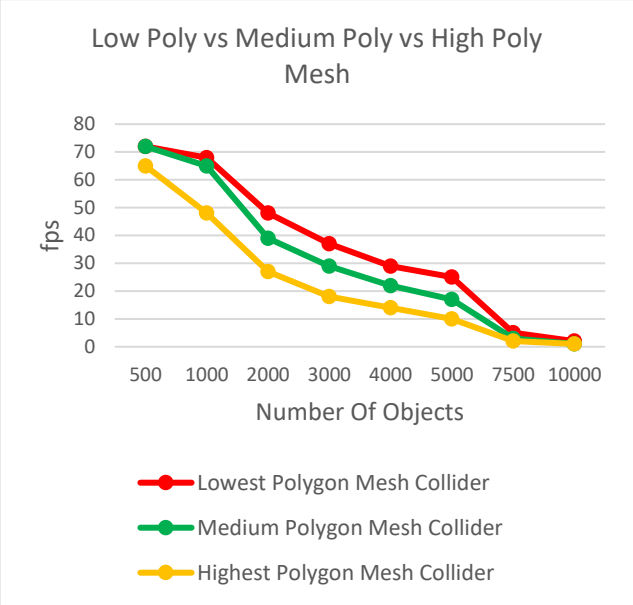

Figure 16 – Low Poly vs Medium Poly vs High Poly Mesh Colliders.

The two figures above (see Figure 15 and Figure 16) clearly illustrate that the object with a lower triangle (polygon/poly) count achieves the better performance with both “Sphere Collider” and “Mesh Collider”. This happens because there is less data to calculate when having an object with less triangles. In *Figure 16,* it can be observed that when passing a certain threshold, in this case the 7500 objects, all the polygon types lead to a similar result close to zero, because no matter the poly count, it is just too much to calculate.

This test also yielded other results that confirm the first test between the “Mesh Collider” and “Sphere Collider”. The result being that overall, “Sphere Colliders” are more efficient for performance (see Figure 17). The difference in performance stays similar throughout the graph.

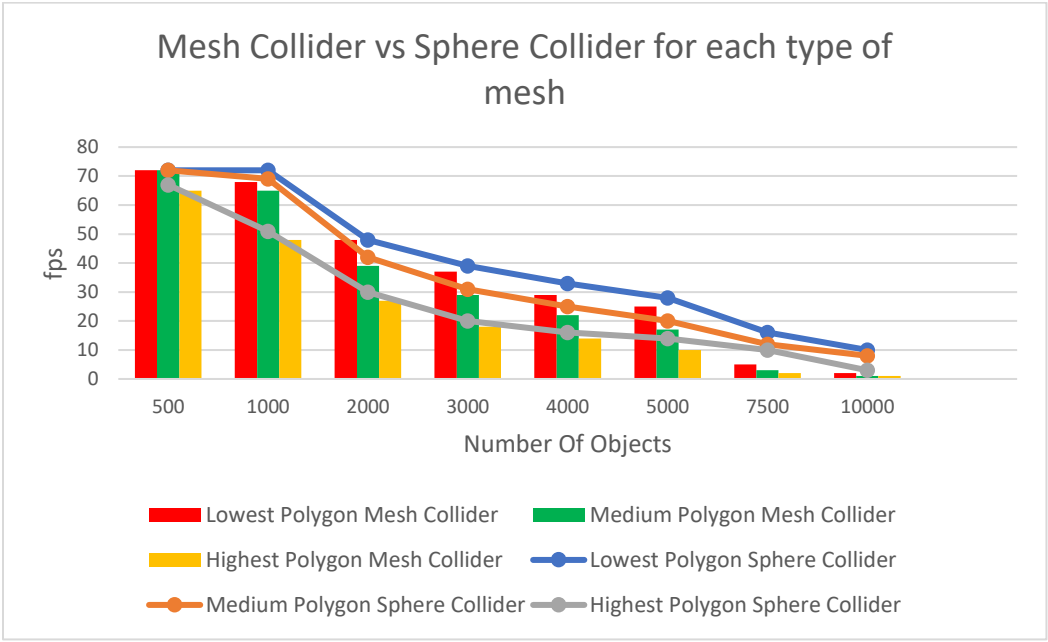

Figure 17 - Mesh Collider vs Sphere Collider for each type of mesh.

Based on the triangle count of the object and the type of collider it is visible that having more objects with a higher poly count will reduce the performance significantly more than having lower or medium sized (referring to the triangle count) objects. There are approximately ten fps difference between all the steps in the graph, excluding the first one and the last two.

The third prototype was used to conduct the third test. This prototype allows to test three different prefabs for each of the poly counts mentioned above with different LODs. The expected result for this test was that it will yield a better performance with LODs, than without them. The result partially confirms the hypothesis, because when reaching 10000 objects the objects with LODs have less performance, than the ones only being “Low Poly” with Sphere Colliders, same with “Medium Poly” (see Figure 19).

The objects with a lower polygon count scored a better performance when using LODs than the ones with a higher poly count (see Figure 18). The third step adds an interesting insight, because it shows that with the same number of objects, in this case two thousand, the performance is much better for the objects with lower polygons. But then as the calculations increase the difference in performance between the steps decreases.

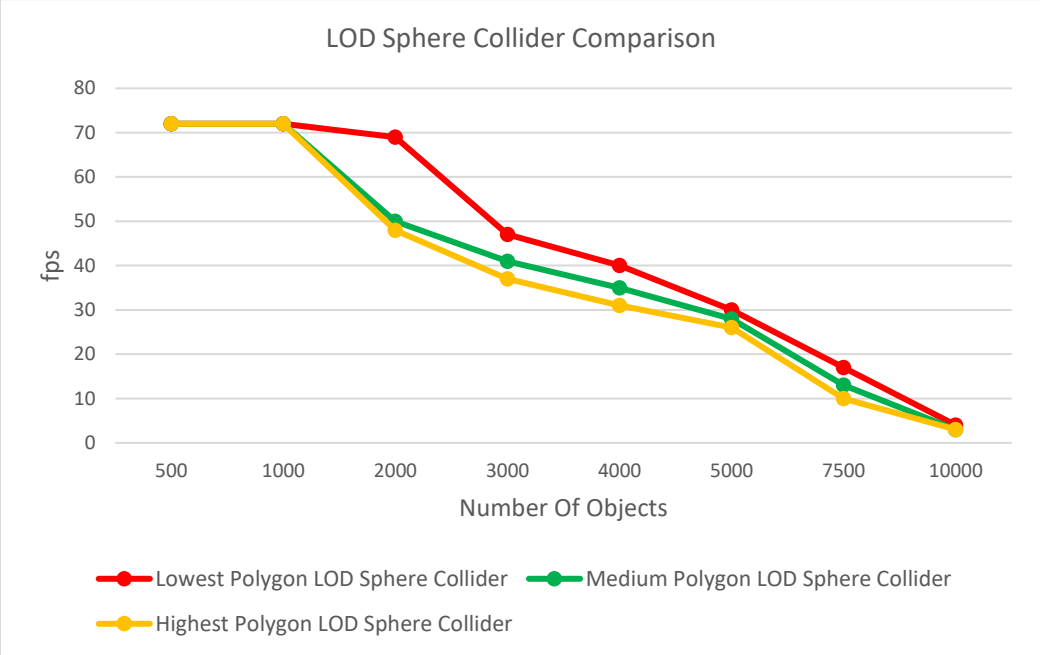

Figure 18 – LOD Sphere Collider Comparison.

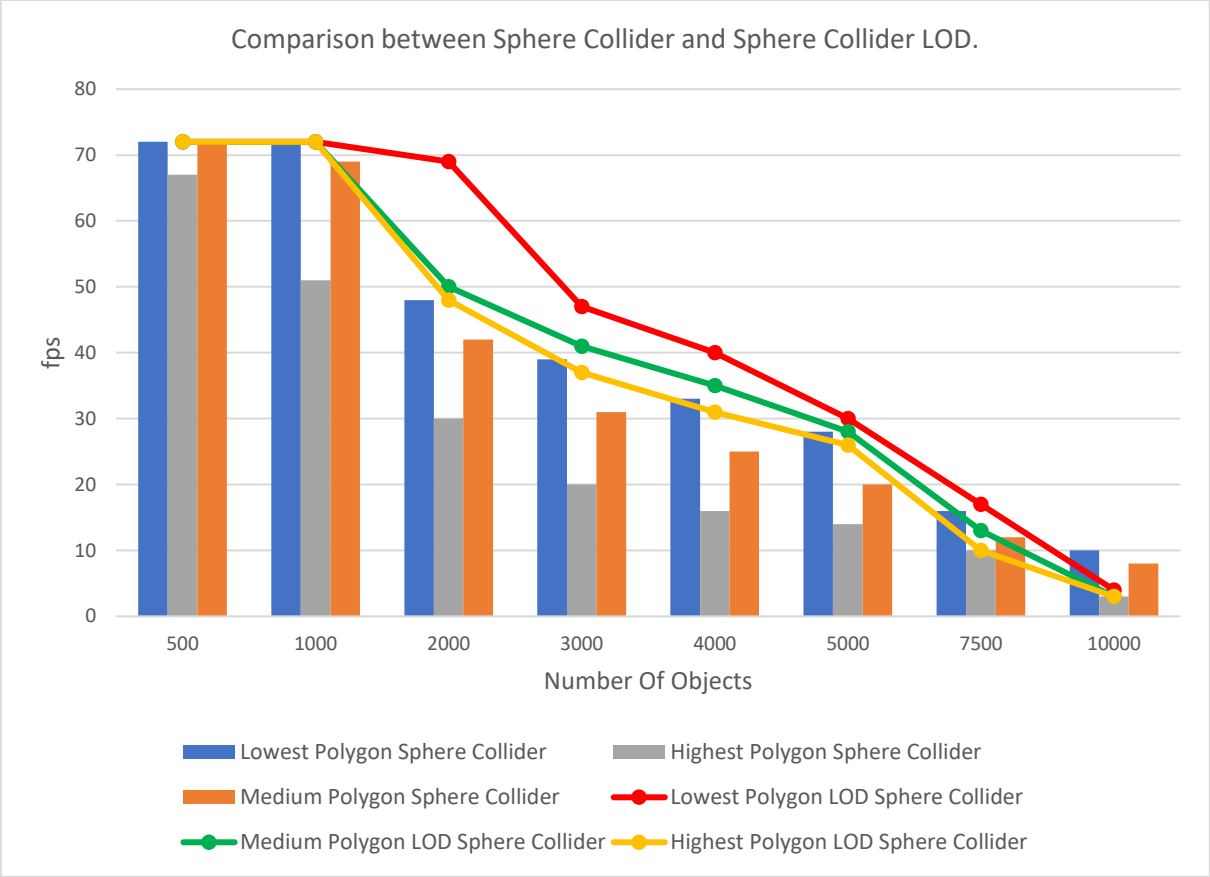

Figure 19 – Comparison between Sphere Collider and Sphere Collider LOD.

Overall, this test shows that LODs greatly improve the performance, but when that 7500 threshold is reached, it equalizes with the objects that have a “Sphere Collider”. This happens because the calculations for selecting the correct LOD also affect the performance, enough to go slightly under the object without LODs. However, it is still a big optimization that creates a difference of sometimes twenty fps between the steps.

5. Version Control

For version control “GitHub” was used for hosting the repository of the project and “Git Bash” was used as the repository controller. With “Bash”, the data was added, committed, and pushed to the repository. For the duration of the project, multiple commits were made, mainly when a scene was ready for testing. The work was conducted on only one branch for simplicity reasons.

The project is currently on GitHub and can be downloaded from this link: https://github.com/MihaiDascalescu/VRPhysicsPerformanceAnalysis.

6. Conclusion

To conclude, this paper explains multiple optimization techniques when using Virtual Reality, such as choosing the collision detection between discrete and continuous, choosing the collider type between mesh and sphere colliders, choosing a mesh that has as few triangles as possible and using LODs to improve performance. Furthermore, it explains how the prototypes work and assesses the difference in performance when using the techniques mentioned above. Moreover, shows how the optimizations made improve the performance on the target device.

7. References

Marr, B. (2022, February 22). *The top 10 tech trends in 2022 everyone must be ready for now*. Forbes. Retrieved June 6, 2022, from https://www.forbes.com/sites/bernardmarr/2022/02/21/the-top-10-tech-trends-in-2022-everyone-must-be-ready-for-now/?sh=4b05b6f2827d

Oculus. (n.d.). *VRC.Quest.performance.1: Oculus developers*. Developer Center. Retrieved July 6, 2022, from https://developer.oculus.com/resources/vrc-quest-performance-1/

8. List of Figures

1. 3.2. – Figure 1 – The room in the scene.
2. 3.3. – Figure 2 – Enum with test cases.
3. 3.3. – Figure 3 – U.I. for the first prototype.
4. 3.3. – Figure 4 – Soccer ball used as a prefab.
5. 3.4. – Figure 5 – Sphere Meshes from left to right low poly to high poly.
6. 3.4. – Figure 6 – Ball prefabs with the assigned color from left to right low poly to high poly.
7. 3.4 – Figure 7 – The canvas for the second prototype.
8. 3.5. – Figure 8 – Low poly spheres, lowest from left to right.
9. 3.5 – Figure 9 – Test Cases for the third prototype.
10. 3.5. – Figure 10 – LOD setup.
11. 4.1. – Figure 11 – Test setup in device.
12. 4.2. – Figure 12 – Sphere Collider Discrete vs Continuous Graph.
13. 4.2. – Figure 13 – Mesh Collider Discrete vs Continuous.
14. 4.2. – Figure 14 – Sphere vs Mesh Collider with Discrete Detection.
15. 4.2. – Figure 15 – Low Poly vs Medium Poly vs High Poly with Sphere Colliders.
16. 4.2. – Figure 16 – Low Poly vs Medium Poly vs High Poly Mesh Colliders.
17. 4.2. – Figure 17 - Mesh Collider vs Sphere Collider for each type of mesh.
18. 4.2. – Figure 18 – LOD Sphere Collider Comparison.
19. 4.2. – Figure 19 – Comparison between Sphere Collider and Sphere Collider LOD.