VR ASSIGNMENT

Virtual Reality and 3D Games

Contents

Introduction and How to Play..... 1
Complexity and Sophistication of Scenes and Sounds, Including Menus and Instructions..... 2
Underwater Lighting Effect..... 2
Underwater Particle Effects..... 2
Water Surface Shader..... 3
Water Caustics..... 4
Waypoint/Path system..... 4
Teleport Reticle..... 6
3D Modelling and Animation..... 6
Collection Menu..... 10
Sounds..... 11
Movement and Collision Detection..... 11
Movement in VR..... 11
Item retrieval..... 11
Interactive Features..... 12
Performance Evaluation Report..... 12
Models..... 12
Textures..... 12
Rigidbody Collision Detection..... 12
Colliders..... 13
Level of Detail (LOD)..... 13
Miscellaneous Settings..... 13
Final Performance..... 13
Sources..... 13
Third-party Assets..... 14

Introduction and How to Play

The player can navigate the scene using the left-hand controller. Move the controller into a position where the ray cast from it is colliding with a valid surface, indicated by a green colour of the ray, and then press the grip button to teleport there.

The player can pick up objects using the right-hand controller. Move the controller into a position where the ray cast from it is colliding with an object that can be picked up, indicated by a green colour of the ray, and then press and hold the grip button to pick it up.

Both controllers can be used to interact with menus, but it is recommended to use the right-hand controller due to the shape of the ray cast. Pointing the ray at the menu and using the trigger button will interact with the menu buttons.

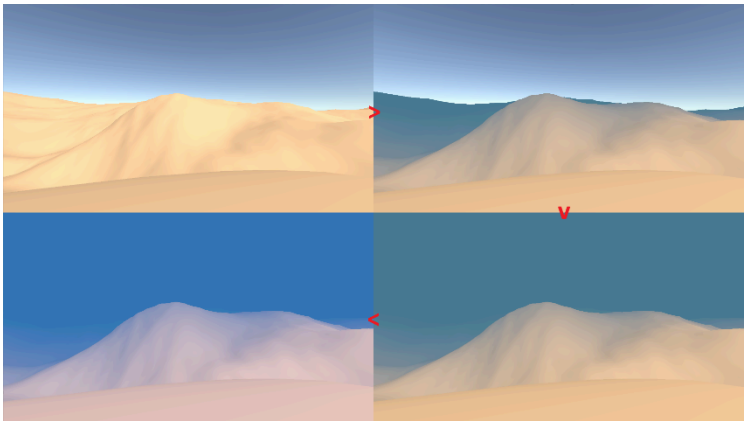
The aim of the game is to earn as high a score as possible by taking photographs of animals, whilst avoiding taking photos of waste products. The player can pick up the camera object, and then press the trigger button to use it to take a photo and will be scored based on what they included in the photo. Taking pictures of the same objects repeatedly will reduce the score on each subsequent photo, encouraging the player to explore to find more animals.

To aid in achieving a higher score, the player can also deposit waste items into the waste bin, removing them from the map. Both waste items and the waste bin can be picked up and moved.

When a new animal is photographed it is unlocked in the collection menu, allowing the player to view information about this animal.

Complexity and Sophistication of Scenes and Sounds, Including Menus and Instructions

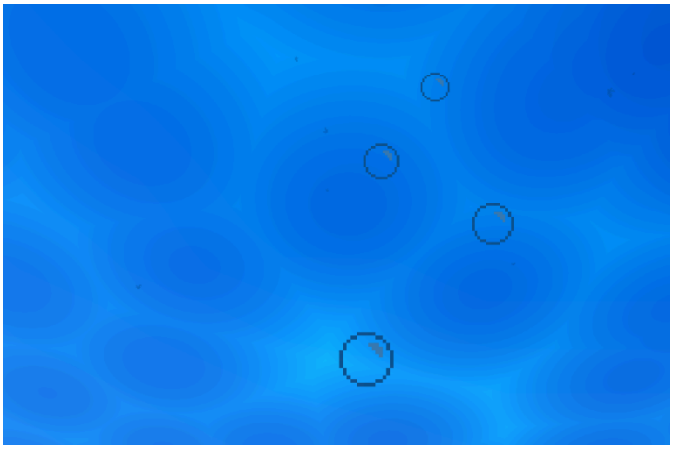
Underwater Lighting Effect



The underwater effect is a combination of a few features. Firstly, fog is enabled in the Lighting>Environment window, and configured to have a blue colour. However, the horizon is still visible through the fog, so to remove that a box constructed of planes was placed around the entire scene, which hides the horizon. Next, a global volume for post processing was added, which applies a blue filter. A feature of light underwater is that it does not create specular highlights, therefore on all materials used, specular highlights were disabled.

Underwater Particle Effects

particulates of ocean debris: a particle emitter was attached to the main camera with a hemisphere emitter, which produces particles in a radius of the player which grow and shrink over time.

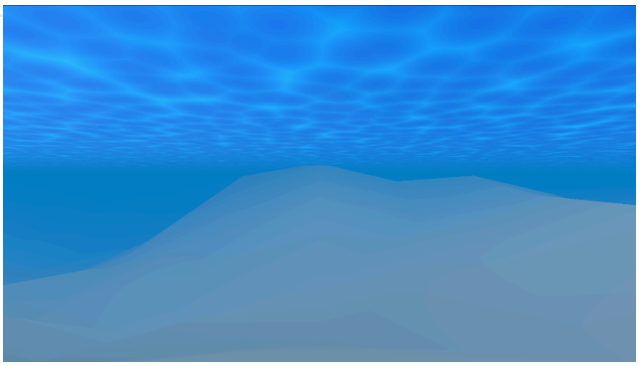


Player breathing bubbles: an emitter was attached similarly to the main camera, with bubble particles on a lifetime of 30 seconds, and a maximum particles of 7. This causes the particles to emit a short burst every 30 seconds. By using the

velocity over lifetime control with world space, the particles can be made to always move upwards, instead of being based on the orientation of the emitter.

Water Surface Shader

Unity's shader graph feature was used to build a simple water shader based on Voronoi noise.



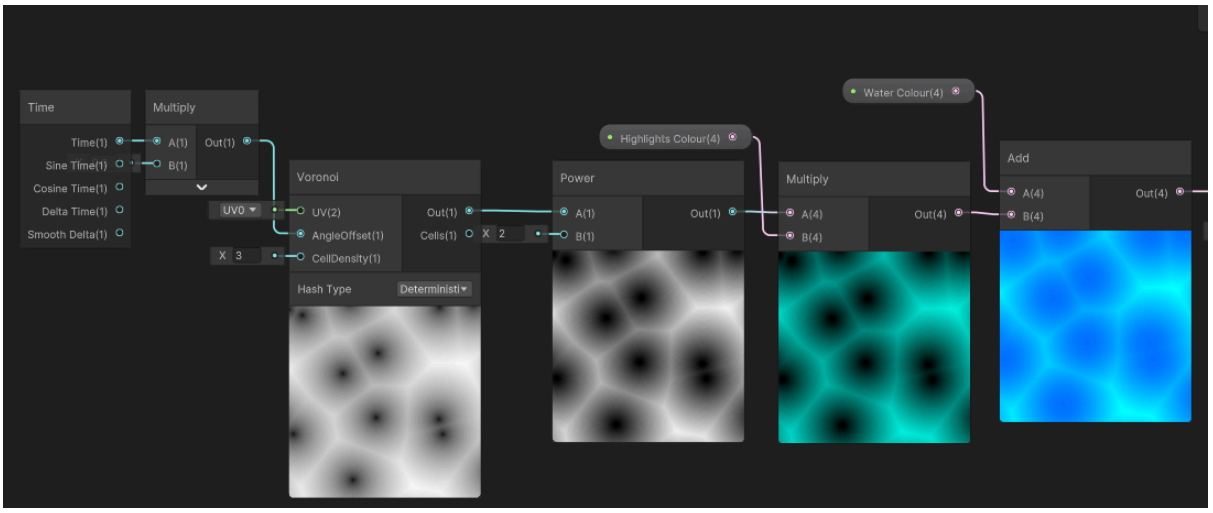
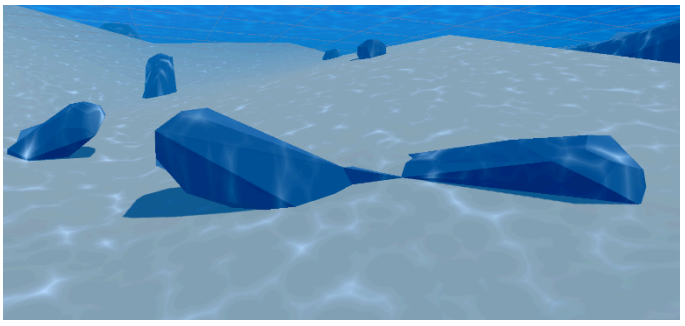


Figure 4, Water surface shader graph.

Water Caustics

A similar shader was built to display the caustics underwater, which was applied to the entire scene using a decal projector. This allows for the caustics to render not just on the ground, but also on any other objects.



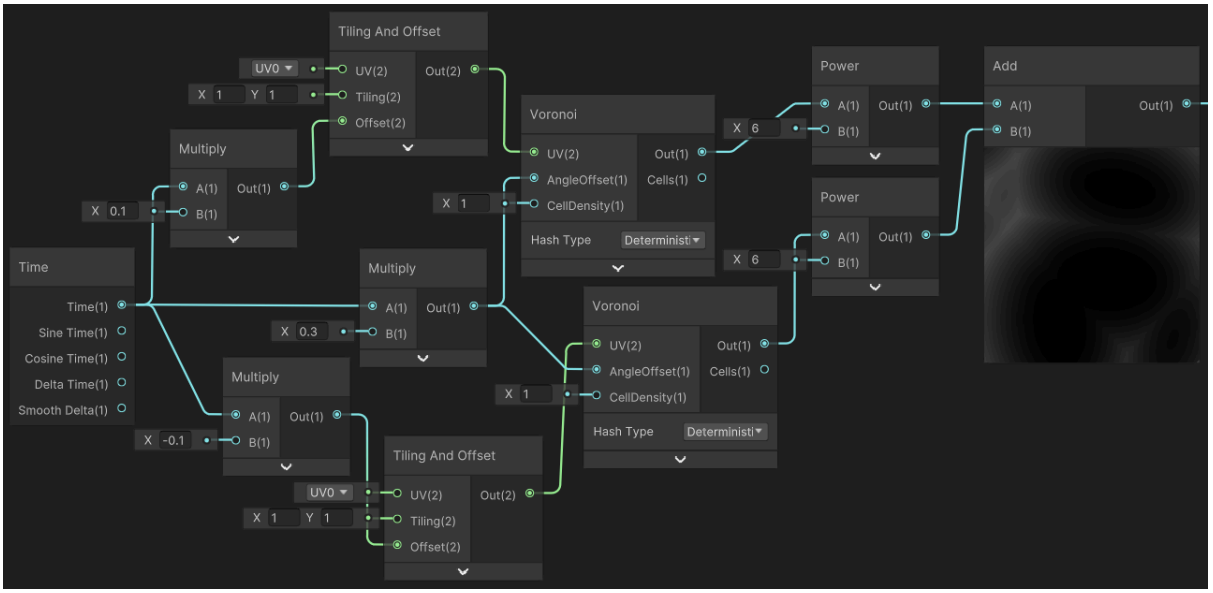


Figure 6, Water caustics shader graph.

Waypoint/Path system

I developed a dynamic system that allows for any number of waypoints to be placed, and an object can be made to smoothly traverse between these in a cycle. Another usage of this system is to have just one waypoint, and the object will orbit it, or to make one object follow another moving object.

Another feature of this system is a random mode, which picks one of the waypoints at random each time. This can be used to make an object move randomly around a region filled with a grid of waypoints.

In case the user sets a rotation value too low, and an object gets stuck orbiting a single waypoint, there is a value for the maximum time trying to reach the same waypoint and if this is reached the next waypoint is automatically selected.

This system can be used creatively to get different behaviours. For example, the first figure below shows a shark which travels between three waypoints, with a group of fish that follow a waypoint with the shark as its parent.

Figure 7, Shark and small fish movement using waypoints.

The next figure shows a set of 3 moving waypoints, these waypoints move together between 3 waypoints and act as targets for multiple fish. The fish are set to randomly select a waypoint of the group which creates a schooling effect.

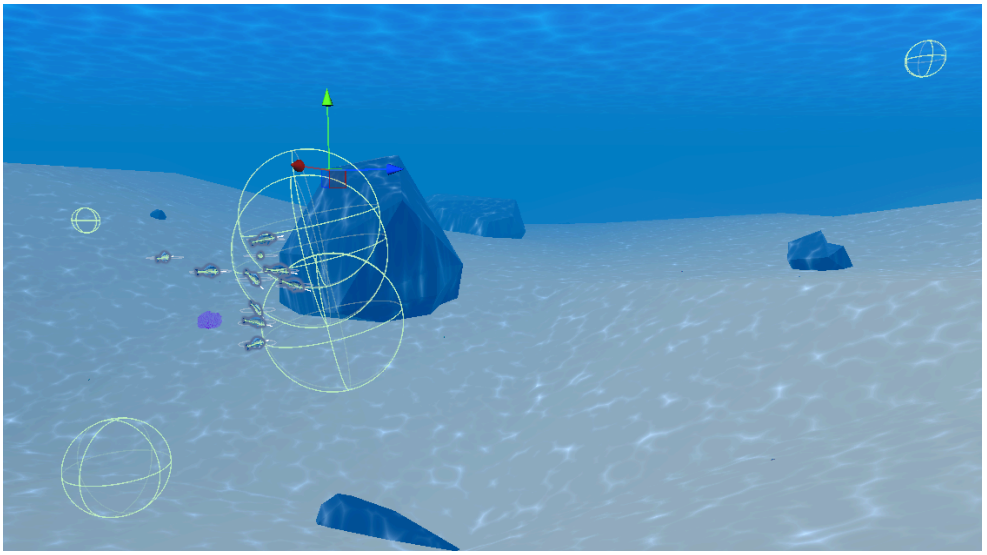


Figure 8, School of fish controlled using waypoints.

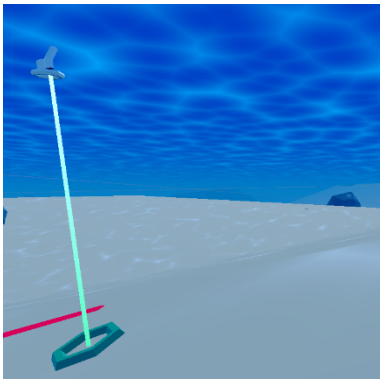
In order to have fluid movement between waypoints, the rotation needs to be gradual instead of instantly snapping to face the target like transform.LookAt() does. Lerp can be used to do this by interpolating a rotation at some point between the current rotation of the object, and the rotation they would need to be facing the target. The rotation of the object can then be updated to this new value each frame, resulting in a smooth rotation.

```
// gradually rotate to face target
Vector3 targetDirection = targetWaypoint.transform.position - transform.position;
Quaternion targetFacingRotation = Quaternion.LookRotation(targetDirection);
transform.rotation = Quaternion.Lerp(targetFacingRotation, transform.rotation, Mathf.Pow(0.5f,rotationSpeed * Time.deltaTime));
// move forwards
transform.Translate(Vector3.forward * movementSpeed * Time.deltaTime);
```

Figure 9, rotation and movement section of MovementPath script.

Teleport Reticle

A script was included (TeleportReticleVisuals) that casts a ray down to whatever is below the teleport reticle and gets the normal information of the surface. This information is used to set the rotation of the reticle so that it is flush with the surface.



This script also handles the other visual effects of rotating the reticle and causing it to bob up and down making it more visible.

3D Modelling and Animation

For this project I used Blender to create the 3D models I needed, as well as rigging and animating where necessary.

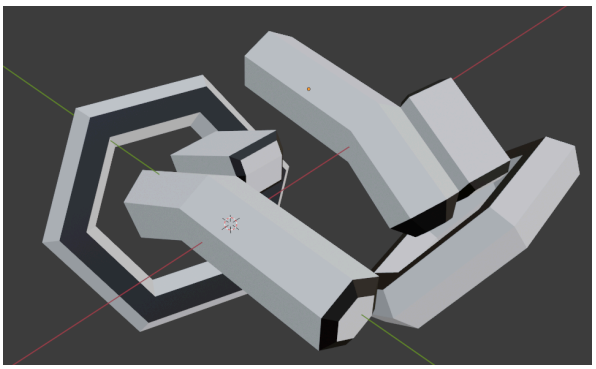


Figure 11, Controllers (168 Triangles Each)

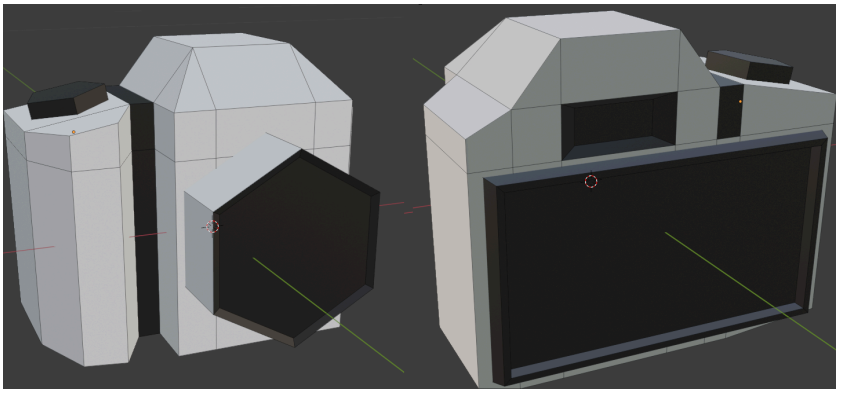


Figure 12, Camera (191 Triangles)

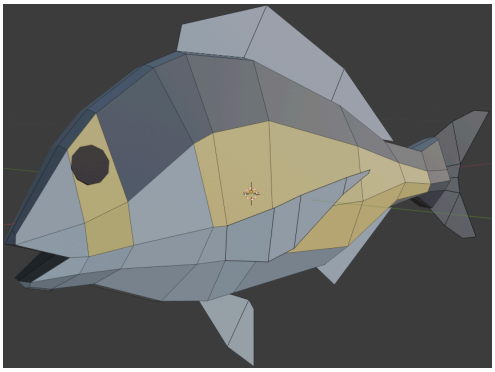


Figure 13, Gilt-head Bream (196 Triangles)

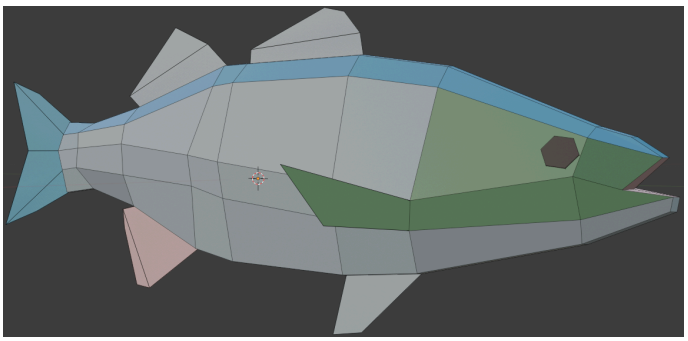


Figure 14, European Seabass (308 Triangles)

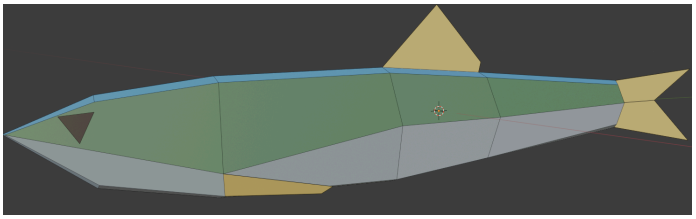


Figure 15, European Anchovy (114 Triangles)

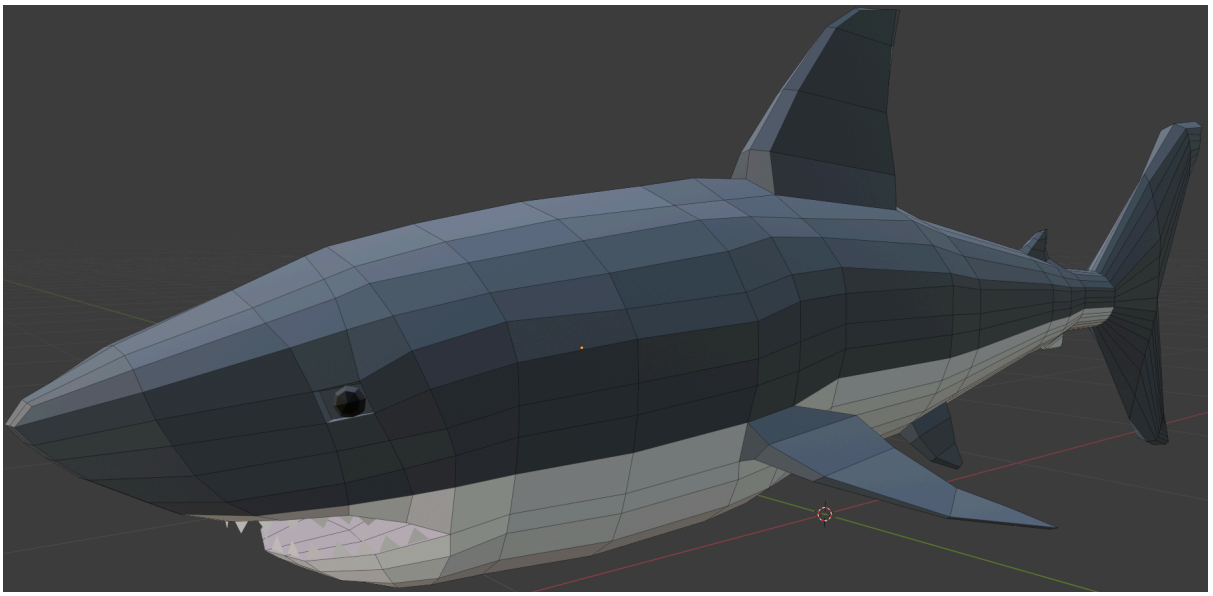


Figure 16, Great White Shark (1452 Triangles)

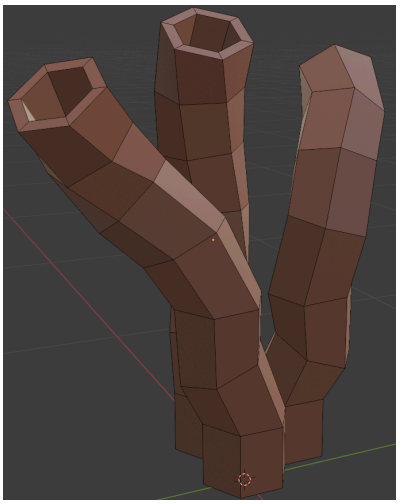


Figure 17, Brown Tube Sponge (564 Triangles)

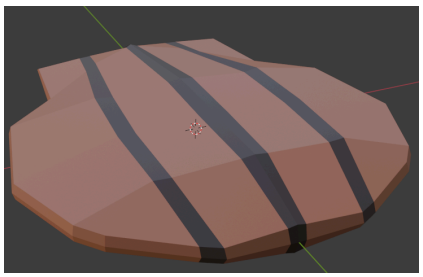


Figure 18, Atlantic Sea Scallop (264 Triangles)

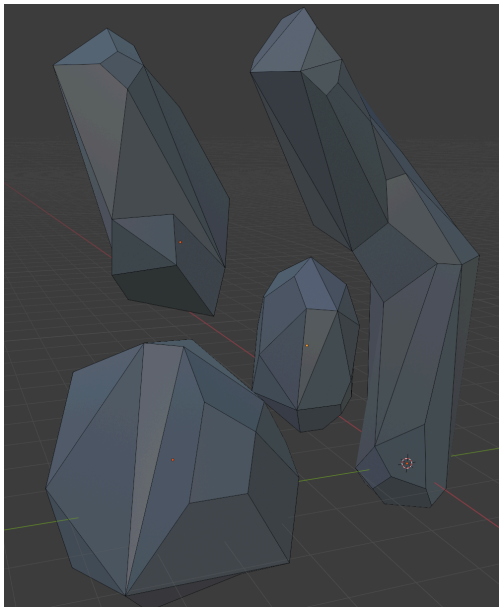


Figure 19, Rocks (44-110 Triangles each)

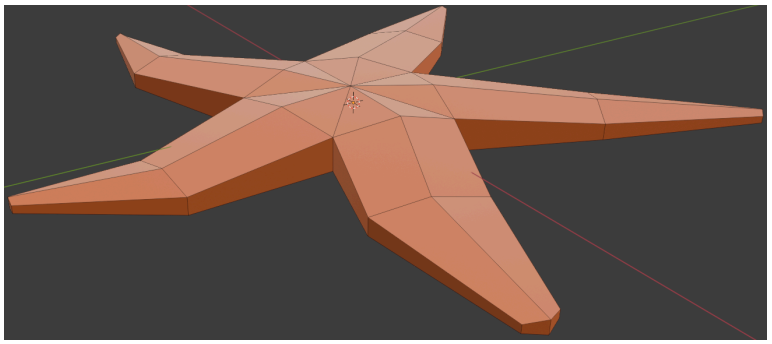


Figure 20, Common Starfish (158 Triangles)

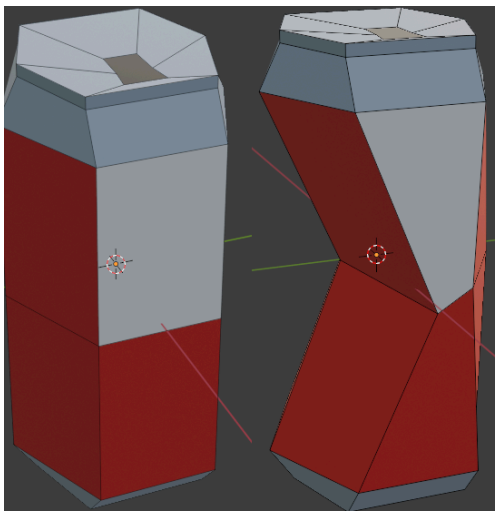


Figure 21, Drinks Can (80 Triangles)

Figure 22, Drinks Bottle (104 Triangles)

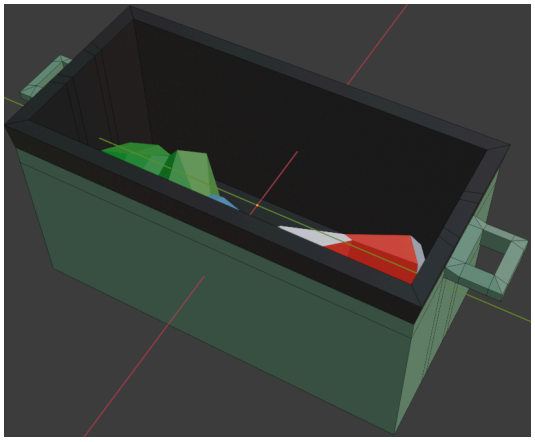


Figure 23, Waste bin (244 Triangles)

Collection Menu

Near the starting point there is an interactive canvas which displays all the animals that the player can find. Initially all of these will be hidden in a locked state but once the player has found an example of this animal in the world and photographed it, it becomes unlocked in the collection menu. Once an animal is unlocked, the player is able to select the animal on this menu to view more information about it.

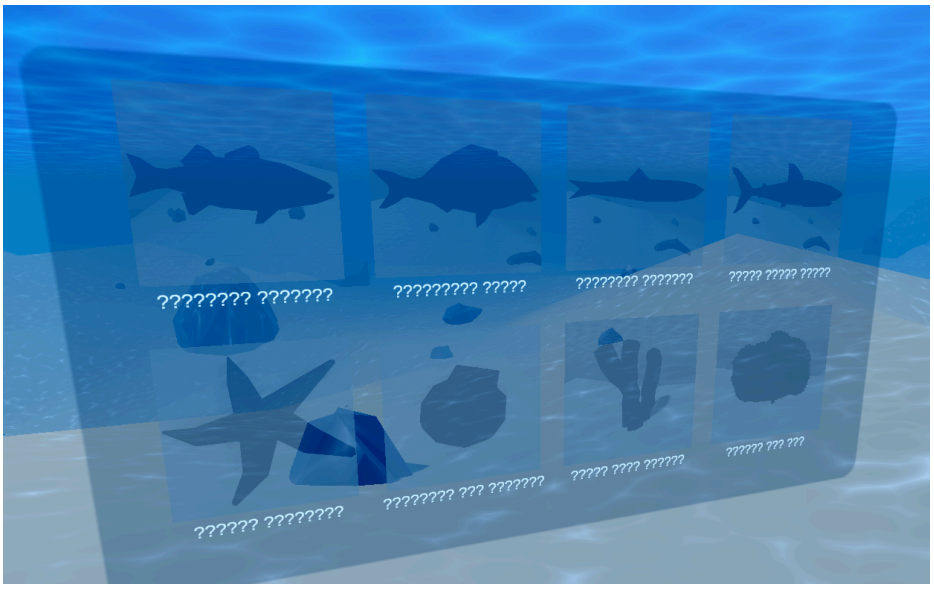


Figure 24, Collection menu.

Instructions Menu

Another menu appears near the starting point which lets the user learn the controls of the game, as well as the basic principles of how to play it. The menu has multiple pages which can be cycled through, as well as a button to hide/show the menu for when it is not needed.

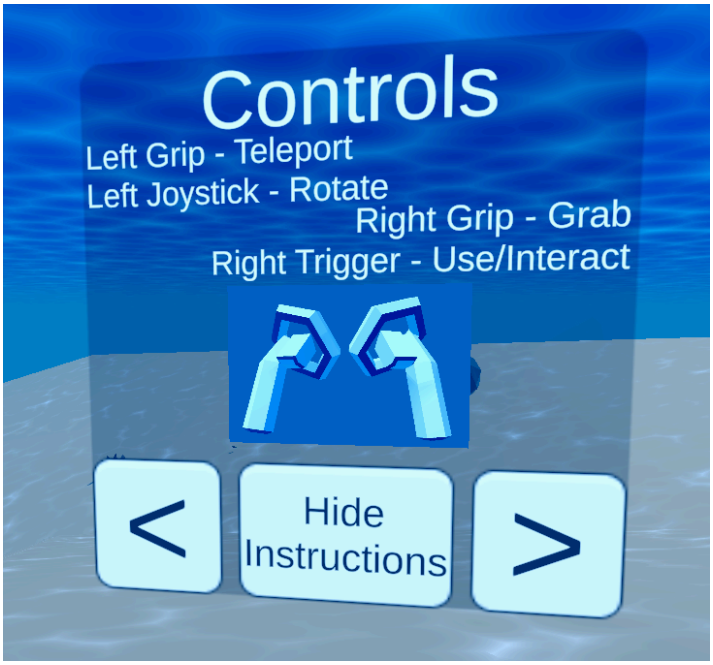


Figure 25, Instructions menu.

Sounds

For the background audio loop, a sound taken from the internet was used. The only change made to this sound was to reduce its duration down to 10 minutes from 6 hours to save on file size.

Whenever the player clicks on menu buttons or takes a photo with the camera, a click sound effect is played. This sound effect was created by me clicking together the metal ends of a charger and amplifying this audio in audacity.

On picking up the waste items, the waste collection bucket, and the camera, a pickup sound effect is played. This sound was made by me dropping a tin onto another tin. These two sound effects were originally created and used for the first assignment, but have been reused for this project as an alternative to using sounds from the internet.

Movement and Collision Detection

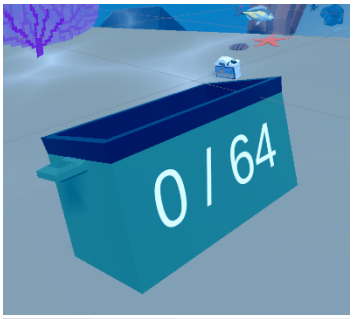
Movement in VR

The player can use the left-hand controller to move around the world via teleportation. The controller casts out a ray and if it is colliding with a surface that the player is allowed to stand on, the player can press a button to teleport to that location.

Item retrieval

There are occasions where collision detection can fail and an item may end up outside of where it should be, when these items are crucial to the experience, this can be a significant problem. To deal with this issue I implemented a simple item retrieval system, which consists of a trigger under the scene, so that if any item clips through the terrain, when it falls into this trigger it will be automatically teleported back into the area accessible by the player.

Interactive Features



The primary gameplay feature is a camera game, where the player can take photos, scoring points based on what is shown. Capturing wildlife gives positive points, and capturing waste gives negative points.

The waste can be removed from the map by collecting it into the waste bin item and the player will be congratulated if they manage to collect all the waste from the map. Removing the waste from the map makes it easier to take higher scoring photos.

When a player takes a photo using the camera, a UI element will appear near to the player which shows the total points earned throughout the game, alongside the points earned from the last photo they took. There is also a list of all the subjects captured by the photo and how many of each and the points earned for them. When taking photos, to prevent players from just taking many photos of the same thing, the subjects reduce the score they provide every time they are photographed.

Performance Evaluation Report

Models

When creating the models for this project, I had performance in mind, ensuring to use as few polygons as possible to achieve the level of detail I wanted.

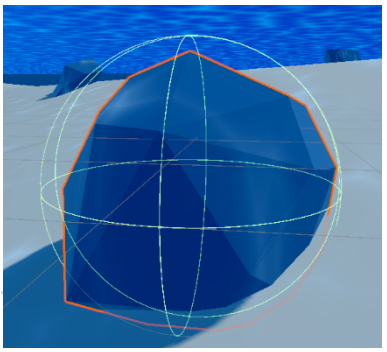
Textures

Instead of using conventional texturing methods which can use up a lot of VRAM when many high resolution textures are loaded, I used a single texture atlas which acts as a “colour palette”, and each polygon of the models are mapped onto the point of that atlas with the colour they need to be. This single atlas has a file size of under 100kb, so it requires a negligible amount of memory to keep loaded. Another feature of this method is that there is only one texture file that needs to be called for every model in the scene, so the number of draw calls is significantly reduced, even if there are many different models on screen.

Rigidbody Collision Detection

Whilst continuous dynamic can be used to prevent objects from falling through the map when thrown too fast, with the number of grabbable objects in my scene it would be very performance intensive to set them all up in this mode. Instead, I used discrete collision detection and relied on an item retrieval system which brings objects back into the map if they fall through the floor.

Colliders



Where colliders are used, I have ensured that they are very simple, only using one or two primitive colliders instead of the more complex mesh colliders. This allows for massive performance savings when scaled to many objects.

Level of Detail (LOD)

Using LOD groups allows for lower quality models to be swapped in depending on the distance of an object from the camera. It also allows for the objects to be culled at a certain distance as well. For the objects in the scene which had animations tied to them, I did not set up LODs as it would require every animation to be re-done for each of the new models. However, I did use LODs for the frequently appearing objects in the map which were not animated: The scallop, starfish, tube sponge, waste objects and rocks. By adding LODs to these objects, it has reduced the average tri count by about 15,000. The low detail models were created using the decimation modifier in Blender, with a ratio between 0.1 and 0.35 depending on what the lowest value was to maintain the general shape of the object.

Miscellaneous Settings

By using fog in this scene to create the effect of being underwater, Unity automatically culls any objects that are hidden by this fog which allows for significant performance savings. It also removes the necessity to use the culling feature of LOD groups.

The camera by default renders the skybox, even if it is not visible like in this environment. By setting the camera to render just a solid colour instead of the skybox, the tri count can be reduced by about 3,500 and 1 batch.

Turning off shadow casting by the terrain makes no visible difference in this scene, but it reduces tris by 2,000 and batches by 4.

In the settings for the terrain, reducing the heightmap resolution from 513x513 to 33x33 reduces tris by around 15,000 and batches by 17. Normally this would be too extreme to drop detail this much but in this project it is acceptable because of the low-poly style.

By marking all objects that do not move in the scene as static, they can be rendered in just a single batch. This optimisation step does not do anything to reduce the visual quality of the scene, but it saves around 70 batches.

Final Performance

After all the performance saving measures have been taken, the final performance metrics are as follows: The number of triangles ranges from 30k to 45k, the number of batches ranges from 160 to 230 and the frame rate on the computer used for development ranges from 80 to 90 fps.

Sources

https://docs.unity3d.com/Manual/index.html - Unity Documentation

https://learn.unity.com/course/create-with-vr - VR Basics Tutorial

Third-party Assets

https://www.nausicaa.fr/en/my-visit/animals/european-sea-bass - European Sea Bass Photograph

https://www.nausicaa.fr/en/my-visit/animals/gilthead-seabream - Gilt-head Bream Photograph

https://www.britannica.com/animal/white-shark - Great White Shark Photograph

https://www.fishi-pedia.com/fishes/engraulis-encrasicolus - European Anchovy Photograph

https://www.inaturalist.org/taxa/120130-Gorgonia-ventalina - Purple Sea Fan Photograph

https://manine-montessori.com/pages/starfish - Common Starfish Photograph

https://animalsandearth.com/featured/brown-tube-sponge-agelas-conifera-pete-oxford.html - Brown Tube Sponge Photograph

https://www.sciencephoto.com/media/374703/view/scallop - Atlantic Sea Scallop Photograph

https://www.youtube.com/watch?v=zrrZBoU7RVc – Underwater ambience