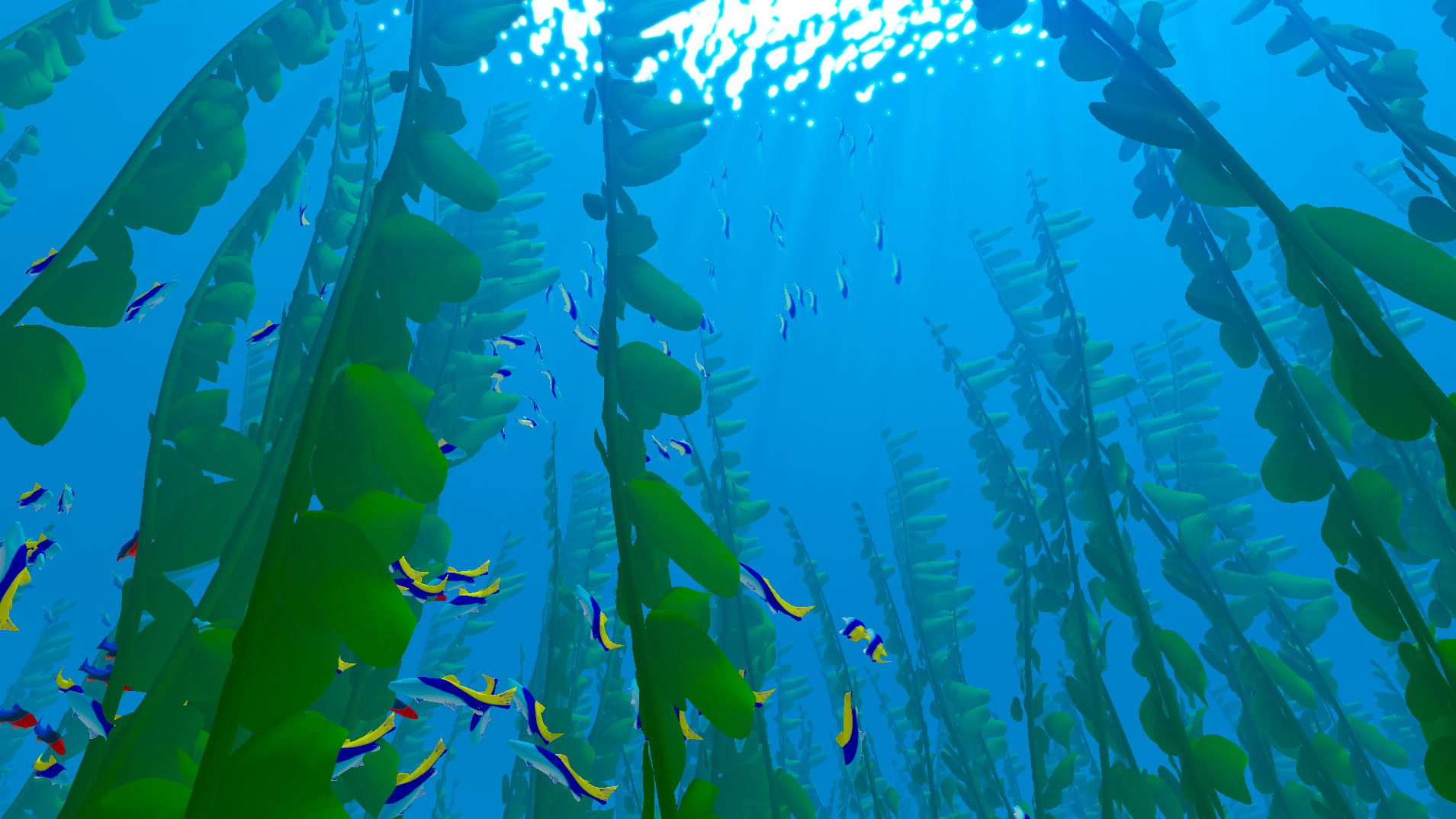
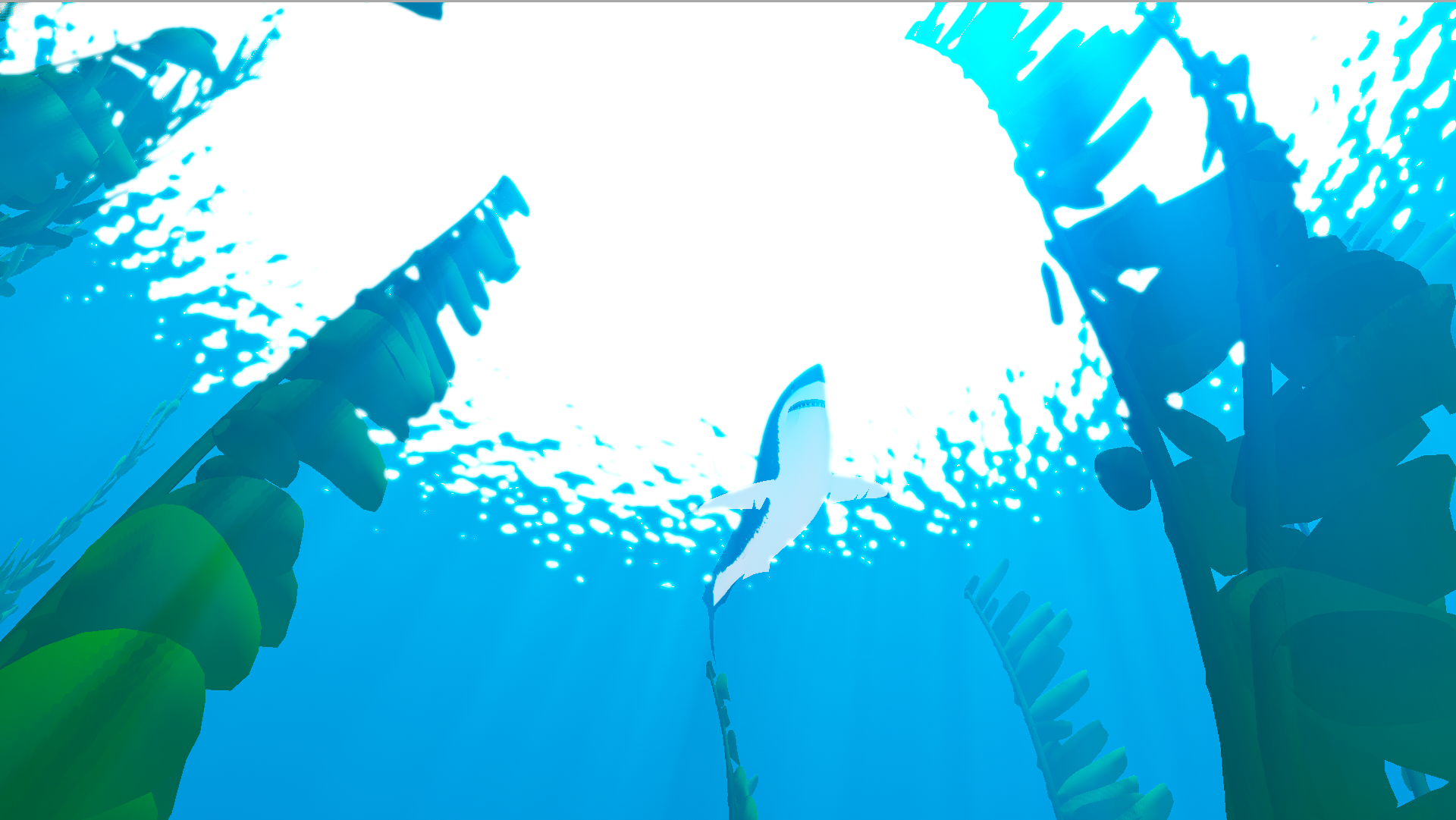
174C Project Report

Aleksei Martynchik(605546832), Edan Abiri (705699909), Mason Bloomfield (404964553), Dennis van Ee (004831758)

**Describe theme and scene design including some figures or screenshots:**

The theme that we chose for our project is an underwater environment simulation with schools of fish, roaming predators, wandering crabs, and various types of ocean flora. We focused our efforts on making a visually cohesive and pleasing scene with dynamic elements.

In order to achieve this we spent the time to create a more robust deferred renderer to support the easy creation and integration of various types of scene objects. We also spent time working on more advanced lighting algorithms such as basic Cook-Torrance based PBR and semi-physically based volumetric lighting.



The rendering backend was created so that the addition of objects to the scene would be as simple as possible. To that end we created a system where a generic SceneObject class was created with simple draw and update functions. Every element in the scene was then pushed to an array of SceneObjects and as such all draw calls, render passses, and updates could be done agnostically on all of them.

The renderer itself contains 10 passes. In order they are: shadow pass -> deferred geometry pass -> point lighting pass -> directional lighting pass -> ambient lighting pass -> generic forward pass -> depth fog pass -> volumetric fog pass -> bloom pass -> and finally tone-mapping and gamma correction.

Deferred rendering was based on a simple system with a 4 element gbuffer, albedo, normal, position, and specularColor. Each is a 16-bit float point RGBA color buffer. The specularColor buffer is misleadingly named and actually stores the roughness, metalness and ambient occlusion values for each pixel. The deferred lighting passes use a version of a Cook-Torrance BRDF as used in learnopengl which was modified to look better underwater.

**Specify additional configurations to run the code**

We do not require any additional configurations to run the code. All the user has to do is launch the host.bat file which will start the server.py server allowing them to access the simulation through localhost:8000. From there the player can freely move around and observe the fish and objects that we created.

**What computer animation algorithms were used and how did we implement them? Did we use any external resources?**

We implemented our animation algorithms through several different objects. The first object that we implemented algorithms within is one of the types of kelp. We created this kelp using a combination of Catmull Rom splines and visco elastic springs. This allowed us to create fluid movement of the kelp giving them a rocking motion while not creating any breaks within the tube or other discrepancies. The mesh for each kelp object was created by making a catmull rom spline using the particles in the visco-elastic ropes as control points and constructing a tube around the path.

The second type of kelp was implemented using a model downloaded from the internet. These kelp were drawn using GPU instancing, a method where the model is only uploaded to the GPU once and drawn multiple times. A kelp controller object is created at initialization that seeds a vertex attribute with 1650 random xz coordinates and adds that to the other vertex attributes of the mesh. The instanced drawing itself was done by editing the tiny-graphics shader activation code to support instanced element drawing. The code was edited so that vertex attributes ending with “\_1” were treated as instanced and had their vertexAttribDivisor set to 1. Furthermore, any vertex attribute with “Transform” in the name was assumed to be an instanced matrix and treated as such (this was used for the boids). The animation for this kelp was done completely in the vertex shader by panning a sin wave along the z direction to simulate the leaves blowing in the current, as well as the composition of forces acting along the x y and z directions based on perlin noise seeded by the kelp’s x and z position in world space. These kelp were also given light checkerboard rendering when close to the camera in order to appear semi translucent and allow the player a clear view of the scene.

Boids were implemented using a basic particle simulation where each boid was represented as a particle whose position and velocity were tracked. Forces were added to the boids to drive their actions. Because of the limitations of webgl and javascript (single-threaded-ness and lack of compute shaders) we separated the boids into schools of 35 fish so that any calculations that needed to loop over all boids were limited to relatively small groups. This allowed us to forego more complicated acceleration structures such as octrees or k-d trees. Despite this optimization however we were still limited in the amount of fish we could realistically draw on screen by a balance of two factors. For non-instanced boids drawing too many fish was impossible because opengl became overloaded by more than a few hundred draw calls. The easy solution to this is to implement instancing and draw all the fish in one call, but in that case we were limited by the copying and uploading of the transformation matrices for all the fish. In the end, we decided to go with a moderate amount of fish (700) drawn in 4 batches (to allow for different types of fish). There is a potential way to speed this up and allow for more fish by uploading only position and direction and then calculating the orientation and translation matrices from within the vertex shader, but we didn’t have the time to test that implementation. The animation of the boids was done strictly in the vertex shader with a set of panning rotations around the z and x axes centered slightly behind the fish’s head and growing in amplitude towards the tail. A modified version of this vertex shader was used on the sharks as well. The models and textures for the fish were created by us.

The sharks are controlled using a similar system to that of the boids, except that they don’t have schools and act independently. They are attracted to the center of the nearest school of fish (which is computed by the boid controllers and stored once every update) and repelled by other sharks. The fish react to the positions of the sharks by scattering quickly away. The orientation of the sharks is controlled and tracked using quaternions so that we can slerp between their current and desired orientations by a small percentage each frame. This allows for the movement of the sharks to appear slower and more deliberate. The shark model was taken from the game Abzu.

Though not an animation algorithm, the volumetric lighting was a large part of the scene’s overall look. This was implemented by a render pass that samples the gBuffer and draws a ray from every point in the scene to the camera and uses mie scattering to sample how much light from the “sun” should scatter towards the camera. This fog was then blended additively with the scene color. Shadows were taken into account and light shafts were faked by scrolling an inverted caustics texture and pretending the highlights were shadows.

The animation algorithms used for crabs were catmull-rom splines and blend-shapes/morph targets. Catmull-rom splines were used to randomly generate paths for the crabs to follow along the ocean floor. The animation cycle that the crabs use is created by combining and uploading two separate poses of the same model into one shape. Within the vertex shader these two poses are then interpolated between on a vertex by vertex basis. The original model for the crab was downloaded from the internet, and the two poses used for the walk cycle were created by us.

For our last animation algorithm we created a water particle simulator which ripples the water as there is movement within it. We were originally planning to use IK for the crabs that can be found on the ocean floor, however, we decided to scratch that aspect from our implementation and animate the crabs differently, still giving them a similar movement but using a simpler technique to achieve it. The water sim is run on the GPU and uses the fragment shader to perform 3 operations: add drops to the water, update the water simulation step, and calculate the new normals of the water. As there aren’t any compute shaders in webgl, we had to use the fragment shader and render to a plane, where the coordinate of the operation is the interpolated position coordinate or the texture coordinate. The displacement and normals are fed into the water surface shader to add some level of dynamicness to it. For this simulation we used a talk from Nvidia at GDC.

External resources were used for inspiration in a few sections of the code, and attribution can be found from within the sections that were inspired by or taken from outside sources.

**Discuss division of work for each team member:**

Aleksei worked on the kelp creating a spline object that used the catmull rom algorithm to create a fluid simulation. We used a hermite spline similar to the one in assignment one adding control points and catmull-rom tangent calculations. Edan assisted greatly with the kelp by integrating the visco elastic springs in order to achieve a better random movement scheme and helped with overall optimization as large amounts of kelp objects significantly negatively affected the performance of our simulation. The copious amounts of calculations needed for tangents and movements for the giant kelp objects created lag. Edan also created the second type of GPU instanced kelp as well as the tube coral and anemones along the seafloor.

Edan also implemented the boids based fish and predators and created the general rendering backend for the project. This included the deferred renderer and all shaders (except those used for the water sim).

Mason implemented the pathfinding of the crabs, having them follow randomly generated catmull-rom spline paths. They also have randomized spawning orientations and positions. The speed at which the crabs move is dependent on the length of their specific spline path, meaning that each crab also moves at a speed relative to their unique path. This was done to ensure that the crabs move at the same speed that their legs move so they don’t appear to be gliding or floating. They follow the paths walking sideways like a crab, accomplished by using the idea of the cross product to find the angles needed to rotate them. Edan also helped with the crabs by implementing the blend-shape animation.

Dennis implemented the water simulation, writing the shaders and making sure that it worked interactively. He also integrated the result of the simulation into the water surface shader to add the correct refractions.

**If significant changes were made since the final demo mention them**

There were two changes since our final demo. The first was the integration of anemones and tube coral to the ocean floor (the models were taken from the internet, and the textures were created by us), and the second was the integration of the water sim into the main scene.