Ocean Simulator: Iteration 1

https://github.com/i12345/ocean-simulator/tree/f0d4c7b4b86193571ed0fc8349a64b3a0f2f9c85

Project Members: Shaina Ayer, Christian Blundell, Isaac Valdez

Vision:

The ocean simulator will be a video game that lets players virtually "swim" in an ocean with marine life using motion-control.

Target Users:

The target users will be an adolescent-focused group that enjoys things such as new technologies, games, science, animals, exercise, and exploration. A sample of this target audience is Jacob, a family member of group member Isaac Valdez, who falls into this target user category and is used for feedback. Jacob wants a simulator game that runs smoothly, has realistic graphics, and extensive gameplay.

Competitors:

Similar video games and interactive projects featuring marine life already exist, such as Ultimate Ocean Simulator, Ocean Mammals: Blue Whale Mari, Fish Abyss: Aquarium Simulator, and David Attenborough's Great Barrier Reef. They have many kinds of animals, and Ultimate Ocean Simulator lets the player "play" as one of the marine animals. David Attenborough's Great Barrier Reef has many educational videos and articles as well. Besides these, there are many non-interactive underwater experiences available on the Internet, such as regular and 3D videos. However, these games lack motion-controlled input.

Move! lets a person play a video game with motion-control, but it doesn't feature the ocean.

Requirements:

The major requirement for this project is to develop a motion control for the user to navigate through the simulator, with the user's webcam detecting movement and passing it out as movement in the simulation. The other major requirement is to design 3D models of marine life that can be discovered by the user while moving in the simulation. The marine life should have simple AI for generation and recognition of the type of marine life generated. Other requirements include generating AI patterns for certain types of marine life (schools of fish, shark behaviors, rarity of marine life) and dynamic generation of marine life. If possible, other requirements could include developing more in-depth patterns for certain types of marine life based on their natural behavior as well as fluidity of the water. Requirements will not include active interaction with marine life, or any simulation based outside of an ocean/sea environment.

Core Features:

- Navigation between main menu, "set up camera", and playing in the simulation
- 3D model / rendering for animals and diver
- Motion-control for the diver
- AI controls the animals' behavior

Project Plan:

The plan is to first develop motion control for the diver. This will be done with NodeJS and NPM. The motion control foundation has been developed and tested during this iteration. For the next iteration we anticipate improving the motion control for certain body poses and adjusting the pose recognition to ensure the body pose is always recognized while in movement. We are developing this to work on a webcam, and we expect this simulator to work on any device that has access to a webcam. There may not be the option to develop a simulator that is connected via screencast, which is shown by the limited users in the target audience. This should take about one month to complete, finishing at iteration 3. The 3D modeling of the fish has also already begun, with 3D images of fish collected to be implemented into the simulator. We plan on getting more 3D models of various types of marine life to be generated (potentially dynamically) into the simulator. We anticipate that this will take about a month until iteration 3. The other requirements will be based on the completion of the previously mentioned steps and will take less time to implement. The simulator starting screen and user control setup will be developed during the second and third iterations. We anticipate 20 hours of implementation of core features for each iteration, including testing.

Core Features I/O Table:

Input	Output
User: Selects Start Game (Home Screen)	**Program: Begins calibration for motion control**
Webcam: User moves using motion control	**Program: Defines movement to mirror movement of simulator**
User: Selects marine life found in simulator	**Simulator: Identifies fish based on value defined based on its 3D model**
Program: Generates marine life	**User: Sees marine life in simulated environment**

Use Cases:

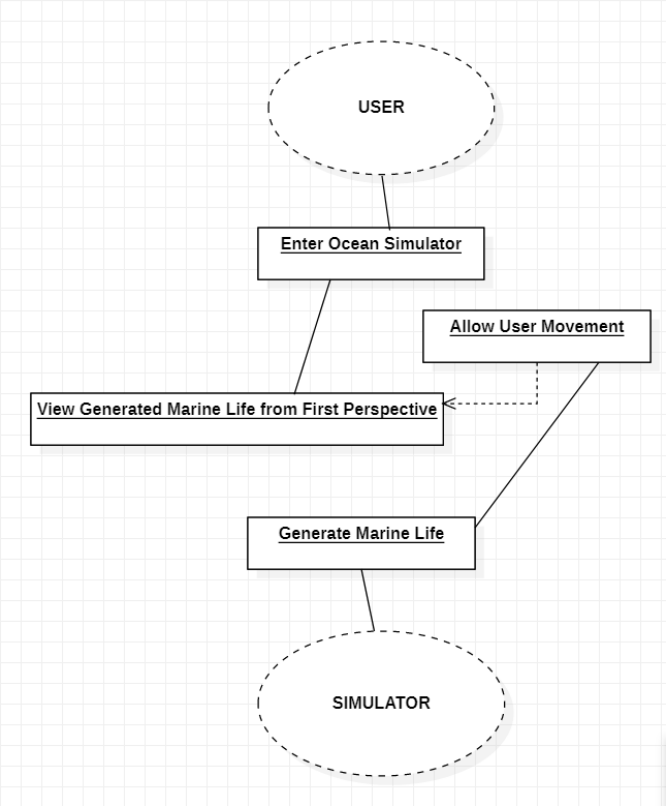


Figure 1: Use Case for General Ocean Simulator Project

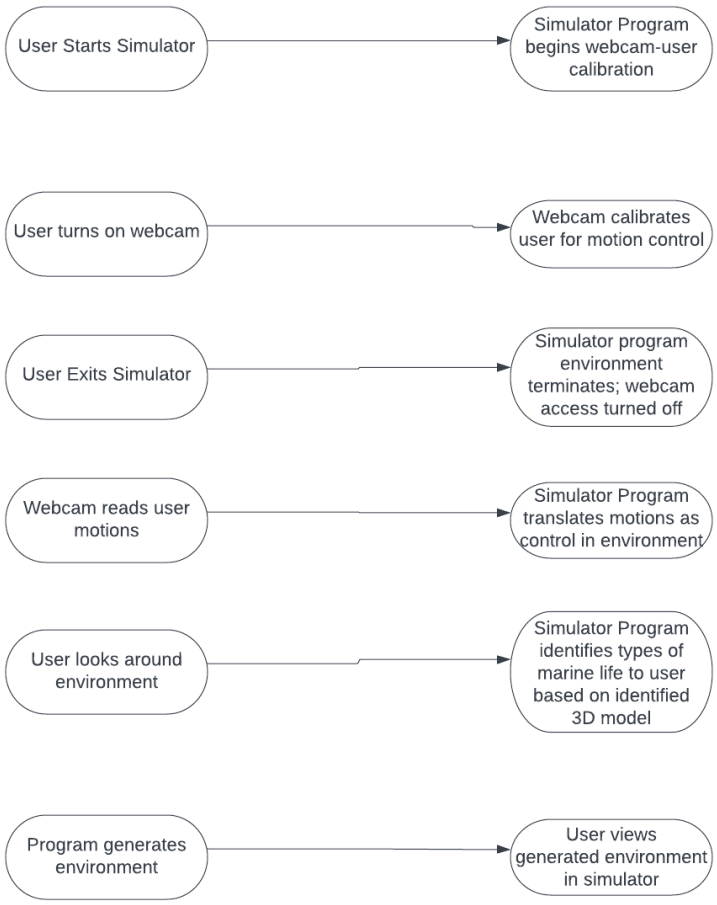


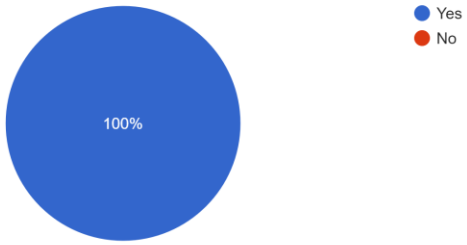
Figure 2: Use Cases for each aspect of simulator setup/gameplay

Risks and Risk Mitigation:

- **TV/Large monitor implementation: 100% x total project hours = Effect on total project (60 hr)**
    - TV/Monitor implementation difficult to achieve, only able to test and develop on single monitor setups.
    - Market Study: 20 people that fit target audience demographic took market study; results confirmed that they do not have a setup that justifies the implementation of large monitor device for motion control

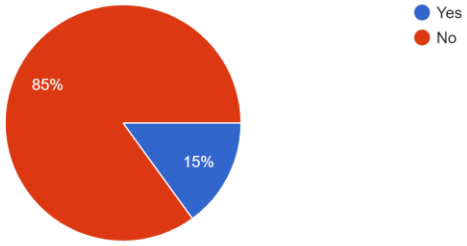
Do you have access to a device with a webcam (not including mobile device)?

20 responses



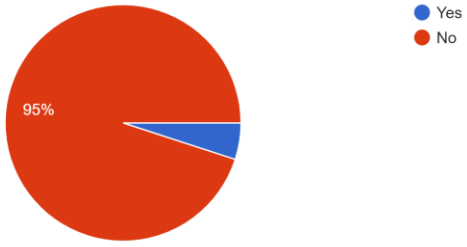
Do you have a TV/monitor that is capable of connecting to a computer using screencast?

20 responses



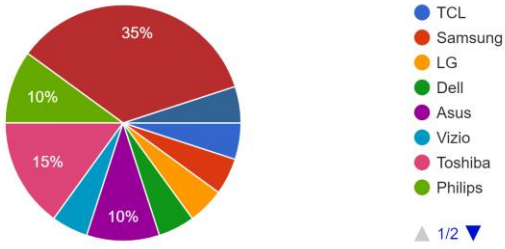
Do you have a console that has a webcam attachment for the TV/monitor?

20 responses



What brand is your TV/monitor?

20 responses



- Mitigation: We will implement and test this project using a single webcam setup on a computer as it is the most available for our target audience. We will potentially develop

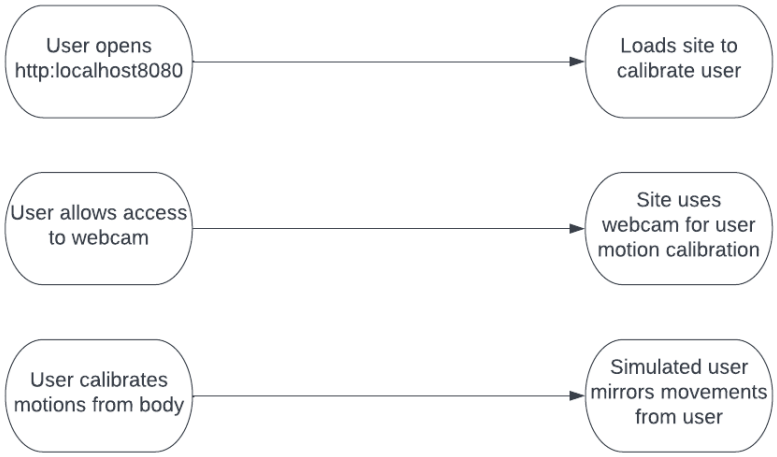
a simple prototype for multi-device setups that work on all types of large monitors/TV that support screen casting. (We will not make large monitors a must-have requirement for our project's final delivery)

- **Unrealistic Schedule: 60% x total project hours (60 hr) = 36 hours**
    - The time frame for each iteration may not be enough to complete each feature to the highest quality anticipated. This will result in a potentially incomplete product for the final delivery.
    - Mitigation: Schedule each feature for each iteration properly and set realistic goals for each feature. In the project plan, we have anticipated that we will complete the most necessary features by iteration 3 so that the product will have the minimum functionalities we anticipate demonstrating.
- **Failure in Pose Detection Functionality: 40% x total project hours (60 hr) = 24 hours**
    - If pose detection technology is too technical and abstract to implement, core features of the product will not be available, leaving an unpolished product as the resulting effect.
    - Mitigation: Analyze pose detection and implementation in earlier iterations to ensure that pose detection remains a key part of the product. (Currently working on pose detection for this iteration)

Current Progress:

We have a foundation for the motion control diver, as we have set up the code and test cases for calibrating the webcam to detect user motion. The arms and joints of the arms can be detected using NodeJS and NPM implementation. The test can be run by opening http local host and allowing access to the device's webcam.

Current Use Case for Webcam Calibration:



Code for Pose Angle:

pose-angles.ts

src > motion-control > pose-angles.ts > ...

```ts
export function calcPoseAngles(pose: Partial<KeypointMap<Vec3>>): DeeplyPartial<PoseAngles> {
    let angles: DeeplyPartial<PoseAngles> = {}
    const blazePose = pose as Partial<KeypointMap<Vec3, BlazePoseKeypointIDs>>

    // L arm
    if (pose.left_shoulder && pose.right_shoulder && pose.left_hip) {
        const y = new Vec3().sub2(pose.left_shoulder, pose.left_hip)
        const z = new Vec3().sub2(pose.left_shoulder, pose.right_shoulder)
        const x = new Vec3().cross(y, z)
        const basis_upper = { x, y, z }

        angles.arms ??= {}
        angles.arms.left ??= {}

        if (pose.left_elbow) {
            angles.arms!.left!.upper = decomposeEulerXY(
                projectToBasis(
                    new Vec3().sub2(pose.left_elbow, pose.left_shoulder),
                    basis_upper
                )
            )
```

pose-angles.ts

src > motion-control > pose-angles.ts > ...

```ts
            if (pose.left_wrist) {
                const basis_lower = rotateBasis(basis_upper, angles.arms!.left!.upper! as EulerAngle)

                angles.arms!.left!.lower = decomposeEulerXY(
                    projectToBasis(
                        new Vec3().sub2(pose.left_wrist, pose.left_elbow),
                        basis_lower
                    )
                )

                if (blazePose.leftThumb) {
                    const basis_hand = rotateBasis(basis_lower, angles.arms!.left!.lower! as EulerAngle)

                    angles.arms!.left!.end = decomposeEulerXY(
                        projectToBasis(
                            new Vec3().sub2(blazePose.leftThumb, pose.left_wrist),
                            basis_hand
                        )
                    )
                }
            }
        }
    }
```

Code for Euler Angle Decomposition:

pose-angles.ts

src > motion-control > pose-angles.ts > ...

```ts
            if (pose.left_wrist) {
                const basis_lower = rotateBasis(basis_upper, angles.arms!.left!.upper! as EulerAngle)

                angles.arms!.left!.lower = decomposeEulerXY(
                    projectToBasis(
                        new Vec3().sub2(pose.left_wrist, pose.left_elbow),
                        basis_lower
                    )
                )

                if (blazePose.leftThumb) {
                    const basis_hand = rotateBasis(basis_lower, angles.arms!.left!.lower! as EulerAngle)

                    angles.arms!.left!.end = decomposeEulerXY(
                        projectToBasis(
                            new Vec3().sub2(blazePose.leftThumb, pose.left_wrist),
                            basis_hand
                        )
                    )
                }
            }
        }
    }
```

3D models have been found and will be implemented in the next iteration. Using 3D objects extracted from zip files we can use these models and identify them before implementing them into the simulator. Each 3D model will have a form of identification that can be used to identify the type of marine species for the user.

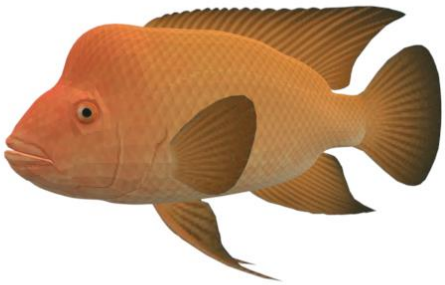


Figure of 3D fish model to be used in the Ocean Simulator.

Test Cases:

There are currently 13 test cases for motion diver calibration. 9 of the test cases are based on tests regarding the decompose identity rotation, and 4 test cases are based on the Euler angle rotation. For the "rotate by Euler angle" test: the test case presents a starting 3D vector and a Euler rotation (rotation around X, then Y, then Z axis), and an expected final 3D vector. For the "Decompose Identity Rotation" test, a given Euler angle is used to rotate the 3D vector [0, 0, 1], and then the decomposeEulerXY() method is tested to see if it can decompose this vector into a functionally equivalent Euler angle. Manual testing with the motion control scripts to make sure the pose of my arms would reasonably realistically rotate the diver's arms was done as well.

Instructions to Run Project:

1. Install nodejs
2. Clone ocean simulator repository
3. Run "npm install" in there
4. Run "npm run dev" in there
5. Navigate to http://localhost:8080 from the development computer and wait for it to load
6. Let web page access camera
7. Back up so the camera can see your arms
8. Test Movement

Feedback:

Jacob (Isaac's brother) is a member of the target user and is our group's representative of the general user. He gave these reviews after testing our current progress: Positives: It's responsive to joints, especially elbows and shoulders. Negatives: It needs to be responsive to all other joints, like wrists, legs, and possibly hands as well. We will take his feedback into consideration for the next iteration and

update the webcam calibration program to allow for more recognition of joints and limbs of the user's body.