Marine Ecology Data Visualization in VR

Brad Coles, Ethan Adkins, Judah Rowe, and Justin Morera

April 27, 2025

***Abstract*—We have developed a virtual reality experience that allows users to immerse themselves directly inside a marine biodiversity dataset based on real-life ecological survey data. The experience offers hands-on data visualization and immersion for both educational and entertainment purposes. The goal of the project is to expand on traditional two-dimensional views of large datasets by offering unique angles and more personal interaction with the data. The environment features three main scenes for data visualization: an aquarium with a scrolling side view to examine organisms in the data at the depths they were observed in, a habitat scene that allows users to get up-close-and-personal with the data by literally swimming around with the data, and a dock with a fun fishing 'mini-game' where users catch organisms based on their actual proportion in the dataset and can see their progress on a graph that filters by taxonomic level. The organisms modeled in the environment come from a structured JSON data file that can be automatically generated from a Python script using a CSV file with each organism's taxonomic info, common name, observation count, observed depth range, and 3D model location. During the experience, these data can be seen in a straightforward window by selecting specimens with the right controller. Additionally, the user can filter the population density and the types of organisms shown based on depth range, maximum observation count, and taxonomy.**

I. Introduction/Motivation

OUR team started with just the simple idea of creating something practical but beautiful to use in virtual reality. One member, with a keen interest in marine ecology wanted to create an application centered around visualizing marine animal observations in a realistic way. The entire team worked together to grow this idea into something more concrete and adaptable by developing different ways a user could visualize this data. Each team member proposed what ultimately became the different scenes used in this final product.

We wanted the experience to be modular and adaptable to different datasets with varying taxonomic groups of organisms, so even though the version showcased in this report features only a regional selection of marine animals, the application is set up to work with organisms from any domain of life so long as the user has supplied models to represent each group in the dataset.

Each member took on different development roles based on their interests and level of experience with Unity and VR development. Brad Coles was in charge of specimen spawning and managing how each specimen held their own representative's data; he also developed the specimen filters tab and underlying logic for the pause menu. Ethan Adkins was responsible for the fishing dock with its fishing activity and progress-tracking graph, as well as the starting hub for scene transitions; he also improved the level of detail for models to drastically improve the framerate. Judah Rowe, a first-time VR developer, developed both the aquarium scene with scrolling depth window and top-down view, as well as the immersive habitat scene with swimming locomotion; he also added much of the aesthetic details to all scenes to improve immersion quality. Justin Morera, a novice at Unity development, was responsible for ecological data procurement and management, as well as representative model procurement, selection logic for all scenes, and UI design for both the floating wrist menu and the pause menu; he also consulted on much of the data linking logic and the biology behind what the data represented.

II. Background/Related Work

VR seems to be an effective tool for offering new ways to visualize data beyond what is possible with traditional 2D data visualization [1], [2], [3]. To this end, we wanted to leverage VR for visualization of marine ecology data in a way that would give users a more emotional and direct experience with the data than traditional 2D tables and graphs.

III. Technical Approach and Concept

In the following section, we will describe the overall design and approach in a high-level conceptual manner.

A. Intended Use-case

When preparing and designing our project, we had several use cases in mind. Primarily, we intended for biologists to be able to visualize the data they obtain in either ecological surveys or datasets of other specimen distributions. This allows these users to better understand what their data means in an interactive and engaging way. With this we intended for the data being visualized to be modular and easily swapped out in order to visualize a wide array of specimen data.

Another, perhaps more popular, use case is that of education in, say, a museum environment. Younger students or children in general would be able to use our application to understand what marine animals inhabit certain parts of the world while also building their taxonomy skills.

B. Models/Dataset

As mentioned previously, models were found representative of all groups (and mainly the family level). Forty-five models

Fig. 1: Representative Models

were used in the environment as seen in Figure 1 below. The marine data was obtained from a dataset published by Diveboard, which consisted primarily of Caribbean Sea sightings of animals from the years 2012 to 2014 [4]. This data was grouped by both the observation count (based on species) and the minimum and maximum depths. We also found common names for each of the 186 species to allow potential users better understanding of what each instance was.

C. Scenes

This project consists of four distinct parts and scenes that each have unique views. The following section will give a high-level and conceptual view of the usage and approach to each of these scenes.

a) Central Hub: From an early point, we decided it would be best to have a central area that would allow the user to decide which part of the application they would like to visit. The central hub area consists of doors to the aquarium, fishing, and habitat scenes/views that brings the user to each one respectively.

b) Aquarium: The aquarium is intended to serve as the main scene for visualizing animals in the dataset. Our original concept for the aquarium was to take a marine environment and let the user get a side view of all of the animals in it. This means, unlike a normal aquarium, there is only one "tank" to look at. Furthermore, we wanted the tank to feel like an actual ocean environment, which would require the tank to be extremely tall. To this end, our draft for the aquarium also included a "scrolling" feature which would allow the user to easily get a grasp of the frequency of certain animals at specific depths. Finally, we planned on adding various exhibit signs and other details to make it feel like a real aquarium. While this last part of the concept could not be fully realized, we were able to design an immersive aquarium scene that captures the data.

The aquarium scene is one spacious room with a tank on the opposite wall from where the user spawns. The user spawns on an elevated platform with the exit behind them. The image of a real aquarium we used for inspiration included a similarly elevated platform. We decided to include it mostly for aesthetic purposes. It also helps fill the room, as it would have felt more empty. It needs to be as wide and long as it is so that the user can get a full view of the aquarium without the animal models being overcrowded. For some final details to add immersion, the room is lit up with soft blue lighting and "Aquarium Ambience" sounds are on loop in the background.

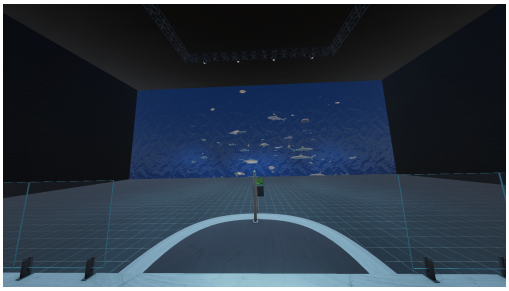


Fig. 2: Aquarium Scene Wide View

The main thing for the user to interact with in the aquarium is the *Depth Display*. This display—located down the stairs in the center of the spacious aquarium room—is the mechanism by which the user can traverse up and down through the environment. The depth display has two arrow-shaped buttons, one pointing downwards and another pointing upwards. By either 1) physically pressing down the buttons, or 2) pressing the grip button while the controller's raycast is attached to one of them, the user can move the tank up or down in the scene. The depth display also features a gauge that provides the current depth the tank is currently sitting at, updating instantaneously as the user moves it. The building is 20m tall, so the range of the two numbers displayed will always be 20m. In addition, the tank is about 54m wide and 20m long (in the Z direction). The dimensions were settled on after test runs proved that making the tank too long hurt the user experience. The animals would swim too far away, and the overhead scene took too long to fully traverse.

Our initial plans for the application also included an "Overhead" scene, intended to give you a view of the animals from above. In the process of developing the aquarium scene, we decided to implement the overhead view into it. It works seamlessly with the scrolling feature. The user can scroll up to the top of the surface until the depth display reads "-20.0m - 0.0m". At this point, the user can freely walk along the top of the tank. While it's a bit harder to get an accurate sense of depth from this view (Fig. 3), it is still valuable. It is much harder or sometimes just impossible to get a view similar to this one in any real aquarium or ocean environment. Note that when the user scrolls up to the surface, the background audio changes to "ocean surface" sounds that can also be heard in the fishing scene.

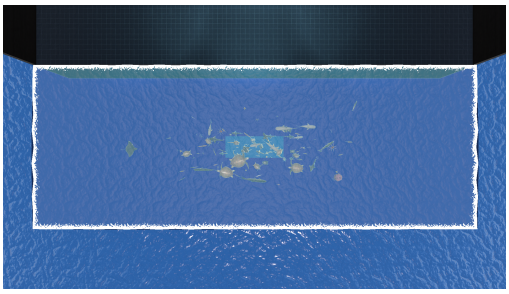


Fig. 3: Aquarium Scene Overhead View

c) Fishing: This view is intended to give a fun/interactive way of visualizing how rare certain animals in the dataset are, while also allowing users to closely inspect the 'caught' specimen. When a user enters the fishing view, they are teleported to a fishing dock in the middle of the ocean. To exit, the user can then head to the boat, which will take them back to the central hub. Further up on the dock, there are two components: A fishing rod and a graph.

In this scene, the user can grab the fishing rod and attempt to catch fish. The fish able to be caught is represented by a shadow in the water directly in front of the user. Originally, this shadow moved around quicker and further away, but we found that doing this was not fun for the user and also detracted from being able to visualize the caught fish on the graph (which were our original use-case priorities).

After a user has caught at least one fish, the graph to their left will populate with a bar chart with counts of the caught fish according to their phylum. The user then has the option to select different taxonomical levels below the graph and update the plot accordingly. As previously mentioned, we wanted this scene to allow a user to visualize the rarity/distribution of specimens in a specific dataset, and this graph-based approach aligns with that goal.

As well as populating the graph's bar chart, catching a fish spawns in the model representation of the caught fish before the user. They can then grab the fish model either up close or from afar and inspect it however they would like. This approach lets users see what makes up each specimen and how it looks. Once done, the user can then release the fish back into the water, where it will disappear after a splash sound, or place it on the dock to keep.

An important component that was worked on was attempting to keep the user grounded and avoid motion sickness (as the moving water could cause this). The solution to this problem was implementing several signs as well as an island off in the distance. The first sign shows where the exit is by the boat. The second sign updates according to which fish was caught, but stays in the bottom left of the user's view and is attached to the dock in an attempt to let the user know where they are looking. The island is off to the right of the user's vision but on the horizon. It is a bright green/grass color with a dock, and also includes a representation of the aquarium on top to further make the experience seem more unified. Figure 4 displays this below.

d) Habitat: The habitat scene drops the user directly in the middle of the ocean environment. This scene best utilizes the VR nature of this project. In it, the user can swim up close to both friendly and dangerous marine animals to get a closer look at them.

In terms of data visualization, this scene is great for getting a more realistic feel of the density levels of the dataset. While the aquarium has a lot of depth, it is not very wide or tall. The habitat spawns animals within a 150x100x150m volume by default. As the user looks around in all directions, it gives them a more natural feeling of a bustling (or barren, depending on what the user set the density to) ocean environment.

Additionally, the users can closely examine all of the beautiful details of the models we found. We found many of our models from sources that took 3D scans of actual dead fish and converted them into high-quality models. However as will be elaborated upon further, our models use level of detail optimizations so that the application runs smoothly. In exchange, the models of the animals reduce to lower quality when viewing them from a distance. In the habitat scene, there is no restriction on the user's movement. They can freely travel as close to the animals as they want and view them in their highest quality.

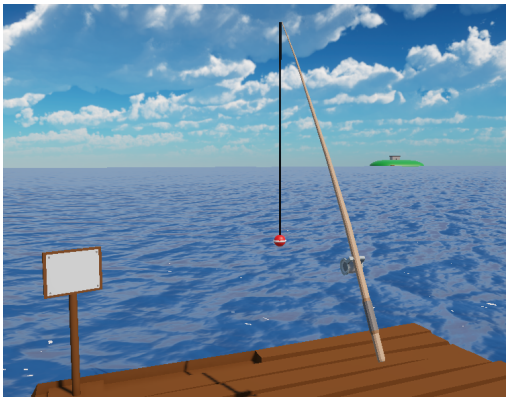


Fig. 4: Fishing Scene Dock View

Overall, the scene is very simple. There is the skybox that gives the feeling of being underwater, and a semi-transparent plane of sand at the bottom of the environment that subtly provides the user of where the bounds of the animal spawning is. Note that the bounds are almost all visual, though. Besides the fact that you cannot swim directly through the plane of sand, you can swim anywhere you want in this scene. You can swim either with standard joystick movement or with the unique swimming locomotion we implemented. To add to the immersion, we also play underwater sounds on loop (which is subtly different to the audio heard in the aquarium scene). We also added swimming sounds that adjust dynamically based on the type of movement the user chooses to use.

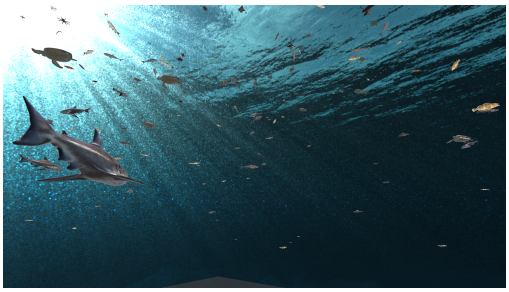


Fig. 5: Habitat Scene

D. Menus

Our design incorporated various menus that users will interact with during operation.

a) Pause: The pause menu (see fig. 6 includes the ability for users to change scenes, see the currently active scene, reset the current scene, and quit the application. It also includes a filters tab that activates the filter menu.

Fig. 6: Pause Menu

b) Filter: the filter menu (fig. 7) is intended to allow users to filter the visible specimens in the environment by taxonomic group membership, observed depths, and observed counts. We implemented a basic filtering mechanism. The menu allowed filtering to membership of one specific taxonomic group such as Kingdom Animalia or Phylum Chordata, filtering by the minimum observed depth and maximum observed depth creating a depth range which must entirely contain the depth range of the species for it to be displayed, and filtering by a maximum value on observed count allowing filtering to remove more common species from view if the user wants to find rarer species. The filter used an 'and' relationship between the three categories of filters, meaning that any specimen had to meet all enabled filters to be displayed. Figure 8 shows a before and after shot of filtering. The filter menu also includes the ability to adjust the density of the spawned specimen adjusting the approximate amount of models to display, when density is less than the total observation count of the dataset the amount of spawned instances may not match the desired number due to the inability to spawn half a fish.

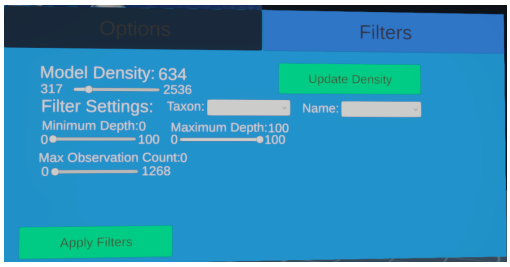
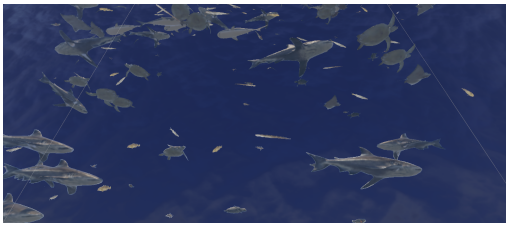


Fig. 7: Filter Menu

c) Information Panel: The information panel (see fig. 9) displays information about a selected specimen within the environment. Users can select specimen to view their information as they explore the environment.

IV. Results and Technical Details

There were several obstacles to overcome while developing this experience, but a few should be highlighted for the uniqueness of their solution. The main work we are particularly proud of includes the pipeline developed to process any CSV dataset into a JSON structure ready to be linked to the experience, the specific way models that represent multiple species were able to be used as higher taxon representatives, the method utilized to programmatically lower models' level of detail to



(a) Before

(b) After

Fig. 8: Filter before and after

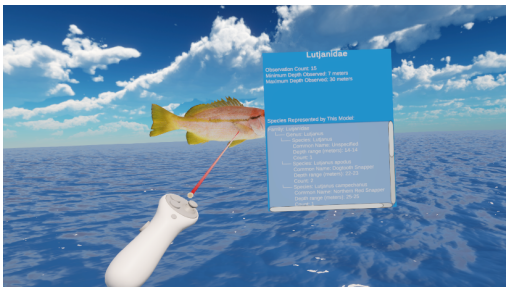


Fig. 9: Specimen Information Panel

boost framerate, and the algorithm developed for spawning specimens randomly within their respective depth ranges. This section discusses several clever tricks and solutions as mentioned previously.

A. Data Processing

The initial dataset used had more than 226'000 observations and had to be manually filtered by taxonomy, region, and time of observation. The final subset of observations still contained 186 separate species with 1'268 total observations. After manual collection of species' common names and models, a Python script was written to iterate through each row of the CSV and create a large nested data structure in JSON format nested by taxonomic level. In our case, every specimen stemmed from a single kingdom, Animalia, and branched outward by phylum, class, order, family, genus, and finally, species. Each higher taxon aggregates the observation count and depth range of all organisms held within it, which allows easier access to the information of that group if the representative model is set to that level.

B. Model Representations

Due to the lack of species-level representation for all but the most popular animals, most models were confined to represent specimen groups at the genus, or even the family level. However, this was expected, and so the code for the

application allows setting models at different levels of the taxonomic hierarchy. The menus and graphs are specifically designed to descend the phylogenetic tree of each specimen's 'root' level in order to present the information nested under that representative. If a user is able to provide models for all specimens at the species level, then the 'root' will be species for each specimen, and each model would appear different visually and would represent a different data point in the actual data. This implementation was done to avoid having visually identical models that all represent different species in the dataset, making them impossible to differentiate. Our implementation allows users to visually identify separate taxon groups while still being able to select them to see the underlying information of the taxonomy that is not directly shown in the environment.

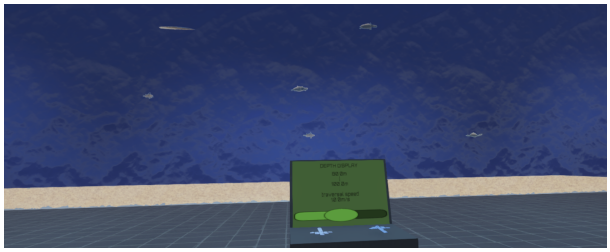
C. Level of Detail

One of the largest issues we faced while working on this project was the models' extreme detail. More specifically, while the models represented the specimen well, the overall performance of the application would struggle. For example, one of the models by itself consisted of 1.5 million triangles on its own. Since each of these models was loaded at this level of detail constantly (never culled/changed) the user would often feel motion sick or be unable to perform activities (as the lag/framerate drops would make it hard to move or select objects).

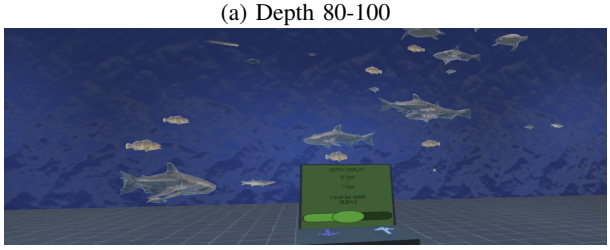
The solution Ethan came up with for the framerate/model problem was implementing a variable level of detail. What this does is, as the user walks away, it will reduce the model quality up to three times and then finally disable the renderer entirely if too far away (or looking away from the model). The specific implementation was a several-forked version of the official Unity AutoLOD package since we are using a newer version of Unity [5]. This package allowed us to automatically reduce the main model polygon count and include lower resolution versions for when the user is not near them. After the changes were made the average triangle count visible on the aquarium scene went from 550 million to around 40 million at a single instance, allowing the application to become a lot more playable and lessening motion sickness.

D. Spawning Algorithm

Specimen spawning required that specimen spawn randomly throughout the environment but exist only within observed depths for that specimen as shown in figure 10. To accomplish this spawning we needed to ensure all data and models were loaded, then we could calculate the relative distribution of each species based on the ratio of observed count to total count and use this distribution to determine how many instances to spawn based on the set density parameter (approximate amount of models to spawn). Each model also needed to be spawned within the confines of the environment, since we wanted a random spawn this required use of a buffer zone for allowable spawn points. In order to most effectively make use of the space we used a dynamic per model buffer that used the bounds of the model plus some small padding of



(a) Depth 80-100



(b) Depth 57.6 - 77.5

Fig. 10: Depth based spawn example

1 world unit as the buffer. Solving the spawn to avoid out-of-bounds spawns and ensuring depth-based spawning while managing spawn efficiently proved a challenge due to the size of the data. Initial spawn was not able to be implemented in a way that did not block render as doing so caused an issue with spawning within the correct depths, we were unable to resolve this before the end of the project.

Beyond initial spawn it was important to have a respawn mechanism for when users adjust the density parameter for the environment. We wanted this respawn to operate efficiently and without blocking render. To accomplish this we decided that the best method was to use the distribution values and the new density setting to calculate the expected number of occurrences for each species and then adjust the current amount by adding or removing instances until the desired number is reached, rather than the simpler method of deleting and spawning from scratch. The amount of instance additions and removals was limited per frame update to provide a better experience for users by not blocking render.

E. Fishing Scene

a) Fishing Rod: When working on the fishing scene, we wanted, of course, a fishing rod for catching the fish. Ethan designed the fishing rod mechanics from scratch by taking advantage of different Unity joint components as well as the Unity line renderer. The bobber is connected to an invisible sphere on top of the fishing rod via a spring joint that then connects to another invisible sphere using a hinge joint. This is all brought together by a line renderer, which is attached to both the bobber and the fishing rod tip.

b) Graph: The bar chart/graph in the fishing scene was originally difficult to implement. We found, however, a graphics package "XCharts" which takes a lot of the overhead and does it automatically [6]. After adjusting some settings, we could then just pass in the total counts of the caught fish by routing the fish objects through the selected taxonomical level, and the package would handle the rest.

F. Water Shader

For the water in the aquarium and fishing scenes, we used BitGem's stylized water shader [7]. This was difficult to work with but it gave us a lot of flexibility in our scenes.

It operates based on a grid system, which made getting the dimensions to line up with the scenes harder than normal. More pressing, we ran into issues when it came to transparent objects being blocked by the renderer of the cubes of water in our scenes. The water shader blocked anything transparent from being viewed behind it. Unfortunately, this even included some of the animal models that did not use the typical URP shader. Also, this included the line renderers for the raycasting controls on the controller, which makes it a bit harder to know exactly which model you are selecting. Another issue was the "floater" script that was provided with the asset. It did not work properly, which led to us modifying the script ourselves. Still, it is not perfect. The last issue is CPU/GPU usage. The water is made up of many tiny tiles, so as the space you need to cover becomes larger, it will consume resources exponentially.

Despite these issues, we decided to stick with this water because of the flexibility it gave us. Its grid system proved powerful for creating distinct segments of water in the aquarium scene. Not only were they distinct in code, but we could adjust the look and tile size of the water to subtly show the difference to the user (so they know why they cannot move further across the water) and to reduce resource usage. Its shader settings allowed us to create water suited for the exact scene. I.e. we use different settings for the aquarium scene and the fishing scene because we only need to look at the surface of the water in the fishing scene.

G. Swimming Locomotion

Adding swimming locomotion that was based on hand/arm gestures rather than simple joystick movement was a challenge. We adapted code from [8], who adapted code from [9]'s "SwimmingController" script. The first issue was the age of these methods. Neither were created using the most recent standards of VR development, so we had to update them to match the current locomotion framework. After that was done, we could focus on the actual swimming gesture itself.

The swimming locomotion is intended as follows: 1) the user presses one or both of the grip buttons of their controllers, 2) the user makes a swimming motion with their arms, then 3) the user releases the grip button(s) as their arms pull back towards their body. The code we adapted from only supported two-handed swimming, which felt unnatural. You can push yourself with one hand while underwater, so we decided to add that to the locomotion. We also added swimming audio after each release of the grip button(s), with the volume based on how fast the controllers are moving (which also affects movement distance).

We got the swimming locomotion to a point where it felt both natural and immersive. Still, there are many improvements we can make in the future. Such as adding a buffer so that when the user releases both buttons within frames of each other there is no sudden shift in momentum after the second release.

V. Conclusion/Lessons Learned

Overall, we found that we were able to successfully visualize marine ecology data in a way that gives users the ability to explore and understand the data by selecting instances to view the statistics such as observed depths and counts, but also allows a more emotional connection by giving some sense of what being a part of the ecosystem is like with the distribution of specimen. We hope this emotional understanding of the data can provide a memorable experience to users while interacting with the data and thereby broaden their understanding of the data.

In order to provide users with a memorable experience, we learned the importance of optimizing operations per frame to mitigate cybersickness issues. Although there are still improvements we can make for cybersickness mitigation, such as improving the initial spawn, we managed to implement the LOD models as well as improve some efficiency of the data management to improve the frame rate for user experience. We also implemented a physical swimming movement to help with cybersickness, as users performing the swimming motion helps to trick the brain into believing the motion is occurring in the hopes of reducing the motion sickness experienced.

We also believe that VR and our data visualization is not limited to a HMD form factor. While we developed this for a HMD system, the environments and interactions could easily be translated to a CAVE system to allow users a different method of getting involved in the data. We envision that translating the system to another form factor such as a CAVE system could improve the use of the system for a museum environment where it could be an exhibit for multiple simultaneous users to get a sense of the data, although this would require some modifications for tracking in multi-user scenarios.

VI. Future Work

There is a multitude of improvements that can be made to this experience. The overall function of the experience is complete, but many things can be improved or streamlined. The most pressing is improving the latency of the models being loaded into the environment so that users can begin interacting with the specimens sooner after starting the game. Not only is the loading of the model files important for this latency, but we need to improve the initial specimen spawn as it currently blocks render. We were unable to solve this as of yet as some other issue caused spawn to spawn in incorrect depth zones with the fix to unblock render while setting up the initial spawn. Most other improvements are for convenience or increased utility; simple things, such as adding an exit door to the hub so that users don't feel 'trapped' in the facility and can escape without having to open the pause menu and press 'Quit App', as well as making the pause menu block raycasts so the user doesn't select specimens while interacting with the filtering options, in addition to a 'Clear Filters' button to reset all filters to their default values. Another pressing fix is to have filters and the graph data persist across scenes, as this was not implemented in the initial version, so all fishing progress and filter settings are reset when a scene is reloaded.

It was also originally planned to have additional information displayed for specimens, such as a 'Fun Facts' section taken from online sources to increase the educational value of interacting with the specimens; this was ultimately scrapped due to time constraints but has high utility value and should be relatively easy to implement in the future. It would also be worthwhile, however more difficult, to implement sessile organism logic for plants, which aren't in our demo, or sponges and coral, which are present, and can be seen swimming around like fish! While tolerable for the calibre of the project, it would be exciting to see a benthic gradient in the aquarium and habitat scenes where sedentary organisms could be clipped to at their respective depth. The sedentary logic is already implemented in the data, but was not utilized in the game logic, again due to time pressure.

Another area we can improve on is guiding the user to take certain actions. We could do this through a dedicated "Help/Tutorial" tab in the pause menu or another more seamless way. For example, there is no explanation of how the swimming locomotion is supposed to be done, let alone any indication that you can even move in the habitat scene without the use of the joystick. Another area where a tutorial would help is the fishing scene. Some users might not press the right button to grab the fishing rod. Even if they do, they might not realize that they can also grab the animals that they catch.

One final, highly worthwhile pursuit, would be to add accessibility features to cater to anyone with a scientific interest, regardless of physical ability. Adding visual, physical, and auditory accessibility features should be a must for all applications, especially those intended to offer educational value, for it is not up to developers to decide who has access to knowledge-building tools.

REFERENCES

[1] K. Challa, C. Jaiswal, A. Pendem, and B. Gokaraju, "A pipeline for immersive data visualization in cave systems," in *SoutheastCon 2025*, 2025, pp. 1531–1535.

[2] R. Sicat, J. Li, J. Choi, M. Cordeil, W.-K. Jeong, B. Bach, and H. Pfister, "Dxr: A toolkit for building immersive data visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 715–725, 2019.

[3] H. Nguyen, B. Ward, U. Engelke, B. Thomas, and T. Bednarz, "Collaborative data analytics using virtual reality," in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2019, pp. 1098–1099.

[4] A. A. Casassovici, "Diveboard - scuba diving citizen science observations," Jul 2022. [Online]. Available: https://doi.org/10.15468/tnjrgy

[5] YvesAlbuquerque, "Yvesalbuquerque/autolod: Fixed: Automatic lod generation + scene optimization," Apr 2024. [Online]. Available: https://github.com/YvesAlbuquerque/AutoLOD

[6] XCharts-Team, "Xcharts-team/xcharts: A charting and data visualization library for unity. unity." [Online]. Available: https://github.com/XCharts-Team/XCharts

[7] BitGem, "Urp stylized water shader," 2021. [Online]. Available: https://assetstore.unity.com/packages/vfx/shaders/urp-stylized-water-shader-proto-series-187485

[8] J. Barnett, "How to add swimming to your unity vr game," 2022. [Online]. Available: https://www.youtube.com/watch?v=ViQzKZvYdgE

[9] GameDevChef, "Gamedev/underwaterworldvr," 2020. [Online]. Available: https://github.com/GameDevChef/UnderwaterWorldVR