Final Capstone Report — Daydream

Group 9

Janet Chen (j985chen)

Sophie Liu (s657liu)

Kevin Hu (k56hu)

Grace Gao (g34gao)

Richard Xu (r248xu)

Table of Contents

Project Specification

Introduction

Purpose: This document specifies the requirements for Daydream, an interactive projection art installation project, designed to create dynamic cloud animations and scenery based on user interactions. The intended audience of this document is the project developers, potential exhibition venues, and the instructors of our Capstone course.

Scope

- The system will detect user input through LIDAR sensors installed in the ceiling, which indicate the presence and coordinates of people in the room.
    - We will draw a trail of cloud particles behind users with a nonzero velocity.
    - For users that are at a zero velocity for a sufficient amount of time, we will generate a distinctly-shaped cloud (e.g. like a rubber duck, elephant, bunny, etc.) underneath them on the floor.
    - We will create a vortex of cloud particles between users who are standing close to each other.
- We will use Unity to generate and manage the cloud graphics, sensors to detect visitors, and projector displays.
- We will not handle arbitrary changes to the projection (ex. “change the lighting conditions to sunset”).

- Daydream will be installed in a physical venue, a medium-sized room to fit at most 40 people, at the EYEPOOL room in THEMUSEUM in downtown Kitchener.

Glossary

Daydream — The interactive art installation in its entirety.

Visitors — People who enter the room where Daydream is installed (i.e. users of the installation).

Cloud trail / particle system — A collection of cloud-shaped images which follow the movement of the user, and which disappear after a short time.

EYEPOOL — The exhibition space in THEMUSEUM where Daydream is hosted.

Unity — A cross-platform, real-time development platform that supports building 3D games and experiences. Unity 6000 was the main development environment used to build Daydream.

Overall Description

Product Perspective: The system will be integrated with a multi-projector display system, which will project our output onto the walls or ceiling of the exhibition space and handle inter-projection synchronization of content on its own backend. Daydream will be used in EYEPOOL, a public space, for group engagement by children and adults of all backgrounds.

User Needs

- The primary users of this system are the visitors. Their needs are to be provided with a source of entertainment which is simple and engaging to interact with and provide input to, and which provokes imagination, enjoyment, and connection with the community.
- The secondary users of this system are the setup facilitators. Their needs are to get the installation installed and running in the venue, in a way that requires little to no maintenance. They should also be able to monitor the system during usage.

Specific Requirements

System Features

Graphic Projections

The system will display high-resolution cloud shapes. The floor, as well as the lower portion of the four walls, will be taken up by projections of cloud graphics, while the upper portion of the walls will be taken up by a projection of the sky. The experience should be as immersive as possible to closely emulate the experience of cloud-gazing.

The main effects that occur in the graphic projections are:

1. **Shaped clouds floating across the walls.** Clouds that are shaped like animals and objects (such as rubber ducks and elephants) will float upwards from behind the clouds on the walls at random time intervals and from random locations across the room.
2. **Gradual changes in scene.** Initially, the projection should create a sunrise scene, with an orange and yellow sky. Over time, the scene should transition gradually to a daytime scene with a blue sky and fluffy white clouds, and then to an evening sunset scene with a red and orange sky, before looping back to the sunrise. A sun will travel across the sky to mirror this transition. The scene loop should occur within a 5-minute period, so that a visitor can feasibly stay at the exhibit for long enough to experience all the phases.

User Interactions

The system will detect the presence and location of users via LIDAR sensors installed in the ceiling of the venue. The two interactions that the system will provide are:

Cloud trails. When a visitor walks around the exhibit, cloud trails will follow behind em.

1. The trails will appear fluffy to evoke the appearance of a cloud. As the scene changes from sunrise to daytime to sunset, the colouring of the cloud will shift to match the colouring of the sky, such that the clouds are white during the

day, but as the sun moves lower in the sky the colours will shift from yellow to orange and finally to red.

2. When two or more visitors stand sufficiently close to each other, their cloud trails will combine to create a circular vortex that surrounds them.
3. There should be one trail per visitor. Trails should accurately follow the positions of visitors with no obvious jittering or discontinuous jumping.

Appearance of object-shaped clouds beneath visitors. When a visitor stands still for over 3 seconds, a fluffy cloud that is shaped like an animal or object that evokes childhood memories, such as an elephant, a rabbit, or a cupcake, will expand beneath them.

1. The shape of the cloud will be chosen randomly from a set of pre-created models.
2. The cloud will remain on the ground for 30 seconds after it has been created in the original location where it was spawned, even if the visitor moves away from that location. When 30 seconds have elapsed, the cloud will fade from view.
3. When two or more visitors stand sufficiently close to each other, one large cloud will spawn beneath them, shaped like an object that typically comes in a pair (such as boots or mittens).
4. Similar to the particle trails, the colouring of the clouds should match the overall scene, with the clouds appearing white during the day, and shifting from yellow to orange and to red as the sun moves lower in the sky.

Wind effects in the clouds. The speed of the wind effects in the exhibit, which animate the background clouds on the floor and walls, changes subtly with the movement of the visitors.

1. When the visitors perform larger or faster movements, the wind (and therefore the clouds) will also move faster.
2. When the visitors are more stationary, the clouds will move slower.

External Interface Requirements

User Interfaces

The system will allow users to interact with based on two categories:

- Active: visitor movement within the exhibit, walking across the floor v.s. standing still or sitting down.
- Passive: visitor location, number of visitors, other factors that are not intentionally set by visitors

Hardware Interfaces

The system will interface with Augmenta LIDAR motion sensors using a USB connection to detect user movements. The system will also interface with a projector via HDMI to display visual content.

Software Interfaces

The **Augmenta** sensors that are used to track visitor positions also provides a full software interface that allows sensor data to be easily imported and read by the software system via the OSC protocol, while also enabling us to simulate data for local development.

Non-Functional Requirements

Performance

The system must minimize latency to give the best user experience. Cloud generation should happen almost instantaneously, and seamlessly. The cloud trails should not be jittery and should accurately follow the visitors. To not break the illusion of immersion, interactions must be smooth and fast. In other response, Input-to-Action latency should be 100ms or less. The system frame rate should be 60 FPS to allow for smooth user viewing.

Usability

The system must provide an intuitive and user-friendly interface for both interaction and configuration. The interactions of the system should be easily intuited by users, to make the experience smooth and seamless.

Reliability and Availability

The system must have 99% uptime, and must handle up to 20 simultaneous users without delays, and up to 30 without crashing.

Security

Since the system has high visibility and reflects user input data, it must be secure from unauthorized access.

Additionally, the system will make sure no personal data gets stored or transmitted. All data will be handled within the system on a temporary basis.

Compliance

The system must adhere to the Personal Information Protection and Electronic Documents Act (PIPEDA) which stipulates how it may use personal user data. At a high level: obtain consent for data collection, limit data usage to stated purposes, and implement appropriate security measures.

Scalability

The system should be scalable to support a larger projection array or more visitors with minimal reconfiguration. The scaling is expected to be on similar surfaces, i.e. if the final product is designed to be projected onto a wall, we do not expect the system to scale well to a ceiling installation.

Maintainability

- It should be easy to add additional models to Daydream that clouds can assume the shape of, without needing to modify code.

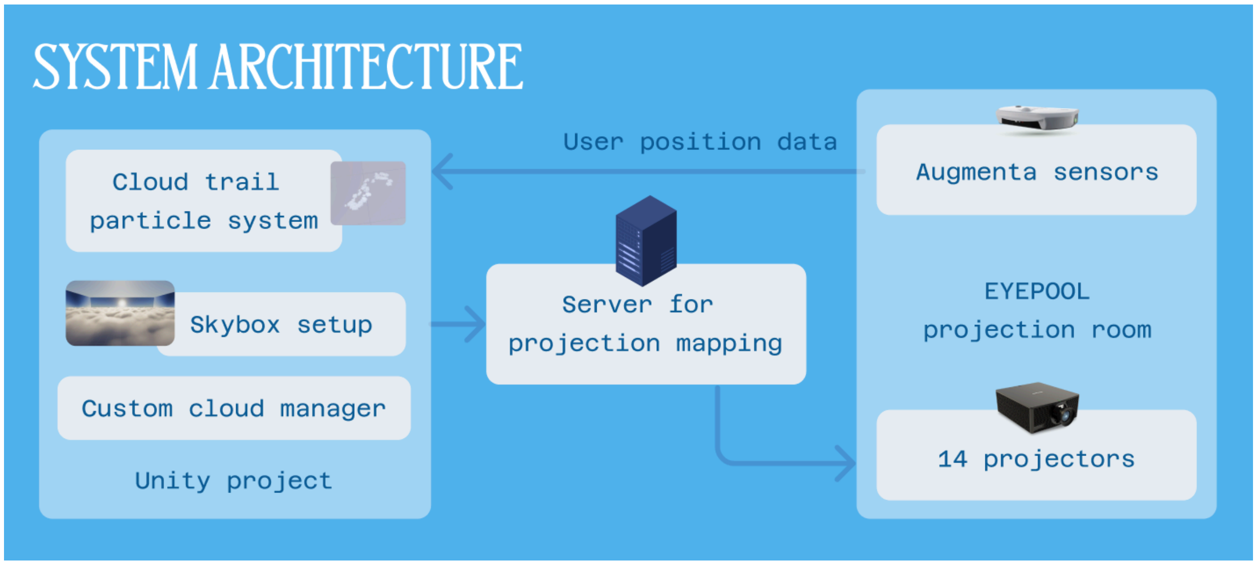
- Any detected user interactions with the system and cloud-related events (transformation, drift direction change, current detected guess) will be logged to assist developers in troubleshooting and debugging the system.
- The system will follow a modular, component-based architecture, where different subsystems (e.g., user interface, projection system, sensors, interaction logic) are designed as separate modules with well-defined interfaces.

Portability

Daydream is not required to be easily portable between different venue locations, but should be configurable to support more than one venue. For instance, camera-projector calibrations and network setups will have to be redone for new venues, but once the communication channels between modules are working properly, interactions with Daydream should function as desired.

Design Doc

Overview



To implement our interactive projection art installation exhibit, we are very grateful to be able to partner with THEMUSEUM and showcase our project in their space called EYEPOOL. The room setup has a LIDAR sensor network, which we use as inputs to our system, and then our Unity displays are connected to their projector displays as output.

All of our software runs in Unity, where the exhibit itself is represented as a 3D scene, and all of the components, including the particle systems, the sun, the cameras, and all of the clouds, are GameObjects that are positioned within the scene. This means that all the exhibit logic is programmed into C# scripts, including the interactive cloud trails, generating clouds, and the 5 minute day cycles. The shader code that is used for visual effects also lives in Unity.

The code and assets are then built together as a game that we run on EYEPOOL's server PC. Our design decision to use Unity helped us to convincingly visualize and manipulate 3D objects in our project while leveraging in-house tools that Unity provides such as easily-configurable particle systems, background volumetric clouds, and other visual effects.

For more in-depth technical discussion about project implementation, please refer to the following key terms in Unity:

- **GameObject** — The fundamental building block of a Unity scene to represent an object in the game world. It can be transformed and manipulated within the scene by scripts and other settings.
- **Scene** — A scene represents an environment within a Unity project. It contains GameObjects, assets, and other elements that define the game area.
- **Script** — A script is a piece of code written in C# that defines the behaviour and logic of a GameObject. They make take in values or references as parameters and control how objects interact with each other and the environment.

Shader Programming

For our cloud implementation, we explored multiple approaches, including particle systems to generate clusters of cloud-like particles in the shapes of different objects, or surface shaders built using Unity shader graph, but ultimately found that volumetric shaders provided the most realistic appearance while giving us precise control over cloud shapes.

We represented our 3D meshes, which were created by hand in Blender, as signed distance fields (SDFs). SDFs represent the distance from any point in space to the surface of the model. Using Unity's in-house mesh to SDF baking tool, we converted our 3D models to SDFs. It was necessary to represent our objects as SDFs, as they provided the depth information necessary to implement volumetric raymarching in our shader.

We implemented a raymarching algorithm that casts rays through the cloud volume, sampling points along each ray to accumulate density and light interactions. This approach allows us to simulate how light scatters within the cloud volume, creating self-shadowing effects that give our clouds depth and dimension.

Our shader uses a lighting model that calculates how directional light interacts with the cloud at different sample points. We take multiple shadow samples along the light direction to create soft, realistic shadows within the cloud volume. This self-shadowing effect is used to create the illusion of volume and depth, making clouds appear darker when the light source is at an angle to the viewing direction, and brighter when directly illuminated.

Additionally, we incorporated noise to introduce randomness into the cloud density, while fractal Brownian motion was used to create smooth, continuous animation of the noise throughout the cloud structure.

The shader also includes a "cloud fatness" parameter that allows us to control the volumetric thickness of the clouds independently of their overall shape. Rather than simply scaling the entire cloud, this parameter modifies the signed distance field by applying a gradient that's strongest at the cloud's core and gradually diminishes toward the edges. This approach ensures that increasing the fatness parameter results in clouds that appear more voluminous and substantial, without altering the base meshes themselves.

For realistic lighting, we implemented a CloudLightTracker script that continuously updates the shader with the current position, direction, and color of the directional light that represents the sun in our exhibit. This allows our clouds to respond dynamically to changes in lighting conditions, creating realistic time-of-day effects including sunset coloration when the sun is low on the horizon by simulating Rayleigh scattering. The lighting system preserves the light's color while applying appropriate shadowing based on the cloud's density and the angle of incident light.

Projector Setup

The projector setup in Unity was a complex and challenging task. Our first iteration started with a 1:1 recreation of the EYEPOOL space in Unity. This involved creating a scaled version of a 28′ x 29′ room with 14 projectors. Each wall has 2 cameras pointing towards it and is connected to their own viewport in Unity. Similarly, the floor has 6 cameras pointing towards it, with each camera connected to their own viewport.

Upon recreating the EYEPOOL space, our second iteration required each camera frustum to overlap perfectly at the wall boundary so that projectors could display a smooth, continuous picture. This was done by setting the cameras to have 45-degree frustums, allowing corner cameras to be perfectly tangential with one another, enabling corner continuity.

Because there are multiple cameras pointing at a surface, there are portions where camera frustums overlap. The overlap in frustums leads to duplicate scene view at infinity. For instance, we ran into an issue of having two suns on a wall due to the parallax effect of the two cameras. To solve this issue, our third iteration involved creating 4 additional cameras to see past the wall boundary and output what they saw to a render texture placed at the wall boundary.

Once the room is recreated, in the final iteration, we place the room inside an active cloud environment, and the cameras simply pick up footage in all directions; as if the cloud environment is an extension of the room itself.

Game Logic

To handle scene logic, all code was written in C# scripts and attached to the relevant GameObjects in Unity. The interactive elements of the exhibit used the Augmenta plugin to receive real-time position data from the sensor network. This data was made available through an `AugmentaManager` GameObject that could be passed into other scripts.

Interactive elements

To implement cloud trails, a particle system prefab (a reusable GameObject) was created and configured to resemble clouds. A particle manager script

`ParticleManager.cs` was implemented using this prefab to generate and update particle systems to match position of visitors in the exhibit.

For spawning clouds on the floor, we created another script `ObjectManager.cs` to process position data and detect when users are standing still. When they are stationary for more than 3 seconds, we generate a new cloud GameObject and assign *a* random material (ie. random cloud shape) to it. This shaped cloud will then live for 15 seconds, with subtle rotation and scaling animations during its lifetime managed by another script for `TemporaryVisualEffect.cs`.

Another detail we implemented is for the special case where two visitors are standing next to one another in the exhibit. In this case, the `ParticleManager.cs` script will position the particle systems to move in a circular orbit around the two visitors to create a playful sense of connectedness. We also added a case in `ObjectManager.cs` to detect this case and generate bigger, different cloud shapes such as mittens or pears to reflect this idea.

Finally, one more interesting interaction involved adjusting the wind speed to match the average speed of users in the exhibit. To do so, we tracked and calculated user speeds from updates in real-time position data through the script `DynamicWindController.cs`. We also applied data smoothing and enforced minimum and maximum speeds to ensure smooth changes in ambience that reflect user movement.

Environment elements

The script `SunMovement.cs` rotates the position of the sun from sunrise to sunset over a period of 5 minutes, with a curved time parameter that maximizes aesthetic experience (ie. longer sunrises/sunsets). As the lighting of the scene changes during the day, we also added logic in `CloudLightTracker.cs` and `ParticleManager.cs` to set the the colouring of the clouds and particle trails to match the time of day.

Most scripts included adjustable parameters to tweak the behaviour, which was very important when user testing the overall exhibit experience.

User Manual

This manual will focus on the operating instructions of the system as a user on THEMUSEUM staff. No manual is necessary for users experiencing the Daydream exhibit, as it is intended to be an intuitive and exploratory experience.

Operating Instructions

Prerequisites. Before starting, ensure all systems are powered on and configured correctly within the EYEPOOL space. Ensure that the final Daydream build exists on the server PC. All assets and executables are stored in a folder located at `Desktop/Tests/daydream`.

Launch the exhibit. To launch Daydream, open the executable within the folder called `daydream6000.exe`. This action will also connect the Unity program to the sensor network, the projector displays, and the speaker system.

Verify setup. Verify that all four walls and the floor are completely covered by the projector. Verify that music playing through the speakers. Verify that position-tracking is correct by walking across the floor and seeing that cloud trails follow under your feet.

Takedown. To stop the exhibit, use the hotkey `esc` on the keyboard to exit the program.

Deployment Information

Daydream was designed in Unity 6000 and was deployed as part of the EYEPOOL exhibit, an interactive art space located within the museum.

Venue Specifications

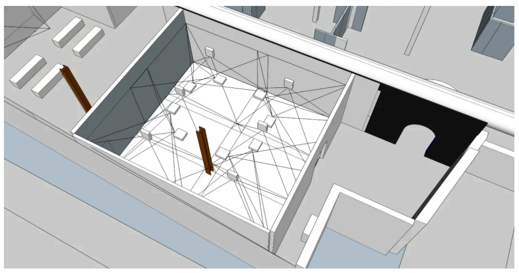
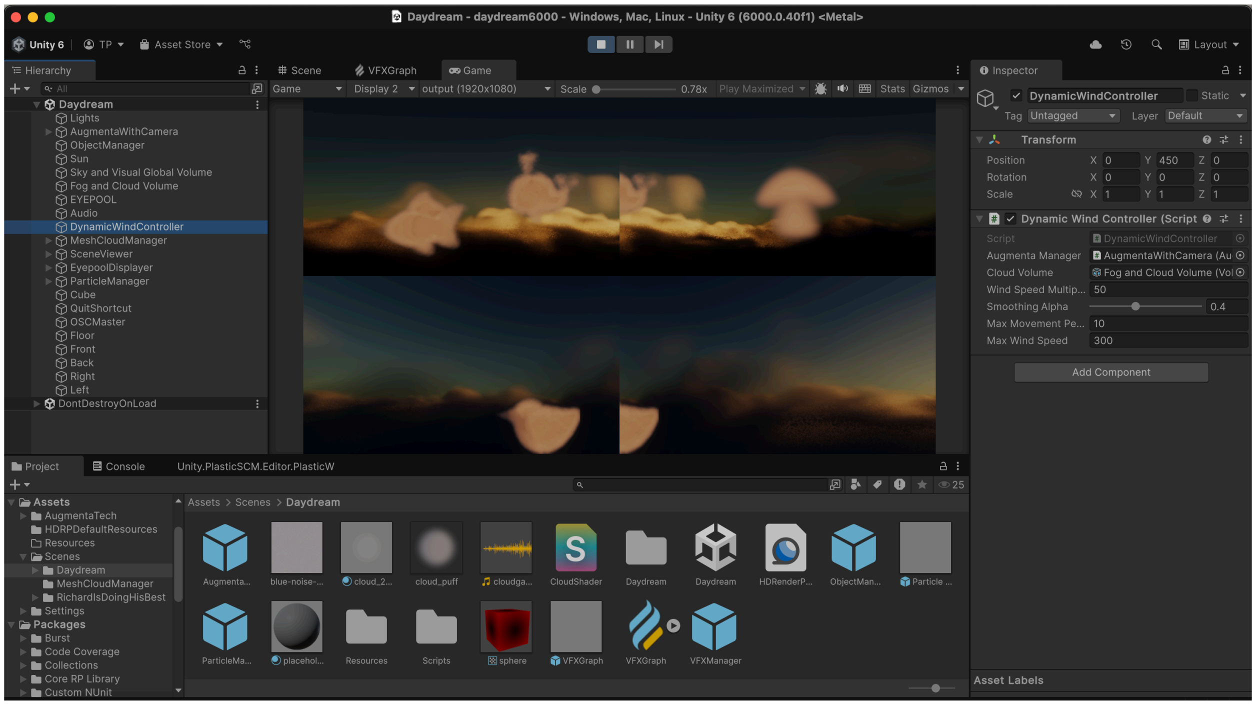


Diagram of EYEPOOL projector setup

EYEPOOL is a space within THEMUSEUM that is designed to host fully immersive experiences. The 29′ x 28′ room is equipped with 14 Christie Digital Inspire Projectors, with warp and blend by Christie Twist on all surfaces (four walls and the floor). The room also has an 8-speaker audio system through Dante and a top-down LIDAR sensor network that uses Augmenta software to provide position data via the OSC protocol. The whole system is controlled by a powerful Windows server PC with a A6000 graphics card. This server PC runs the Unity program that powers the Daydream exhibit.

Deployment Environment

Local development. All development was carried out on personal macOS devices using Unity 6000. We implemented features in VSCode and tested our work locally within the Unity game editor, which allowed for streamlined testing and iteration. All changes were pushed to a shared Github repository, and standard development workflows (pull requests, etc.) were used to collaborate effectively. Visual and audio effects were mapped and tested through local macOS-native builds.

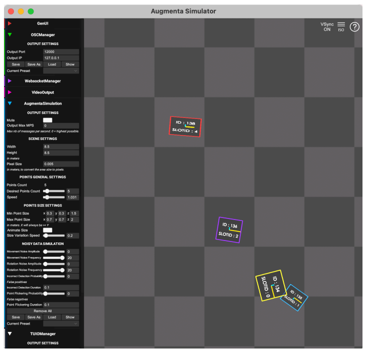


DynamicWindController — Static
Tag: Untagged — Layer: Default

Transform
Position X 0 Y 450 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Dynamic Wind Controller (Script)
Script: DynamicWindController
Augmenta Manager: AugmentaWithCamera (Au
Cloud Volume: Fog and Cloud Volume (Vol
Wind Speed Multip...: 50
Smoothing Alpha: 0.4
Max Movement Pe...: 10
Max Wind Speed: 300

Development environment in Unity 6000



```
OUTPUT SETTINGS
Output Port          12000
Output IP            127.0.0.1
Current Preset

OUTPUT SETTINGS
Mute
Output Max MPS       0
Max nb of messages per second. 0 = highest possible.
SCENE SETTINGS
Width                8.5
Height               8.5
in meters
Pixel Size           0.005
in meters, to convert the area size to pixels.
POINTS GENERAL SETTINGS
Points Count         5
Desired Points Count 5
Speed                1.031
POINTS SIZE SETTINGS
Min Point Size       x 0.3  y 0.3  z 1.5
Max Point Size       x 0.7  y 0.7  z 2
in meters. X will always be >= Y
Animate Size
Size Variation Speed 0.2
NOISY DATA SIMULATION
Movement Noise Amplitude       0
Movement Noise Frequency       20
Rotation Noise Amplitude       0
Rotation Noise Frequency       20
Incorrect Detection Probability 0
False positives
Incorrect Detection Duration   0.1
Point Flickering Probability   0
False negatives
Point Flickering Duration      0.1
Current Preset
```

Augmenta simulator software

For local testing of exhibit interactions, position input data was simulated using the Augmenta simulator software and streamed into Unity via the OSC protocol. This allowed for efficient setup of test cases (ie. user behaviours that trigger certain interactions) and could seamlessly be replaced by actual position data.

Building with EYEPOOL. Upon deploying our code at THEMUSEUM, we needed to configure our program to use Windows build profiles (Windows x86_64) to ensure compatibility with the on-site hardware. The exhibition space EYEPOOL is controlled by a Windows-based server PC, and the Unity Windows mono plug-in was needed to compile and build for their systems.

After building the program in Unity, the final build folder was transferred manually to the EYEPOOL server PC using a USB storage device. This method ensured a reliable and secure transfer of all required executable files and assets. From the server PC, we then ran the executable Unity program to successfully launch the exhibit space. The final setup maintained the intended interactivity and visual fidelity, confirming a smooth transition from development to deployment.

Validation and Verification

Validation

We performed validation via user testing. Throughout the development process, as we iterated through different ideas for user interactions, we created MVPs using the p3.js web editor and conducted user surveys to determine which interactions felt most intuitive. Later on, after we finalized our venue, we also conducted user testing by polling users that visited the space to experience various iterations of our installation, as well as evaluating our own perceptions of the exhibit during the development process.

The user feedback we gathered helped us ensure our exhibit provided an experience that met our initial goals - an immersive projection art installation based around cloud graphics, which visitors would enjoy experiencing. Some examples of feedback we received include:

- **Matching the colours of the clouds to the sky**. Originally the shader that we used to render our clouds would produce yellow highlights and blue shadows regardless of the colour of the rest of the sky, which would shift based on the position of the sun. After we received this feedback, the shader was updated so that the clouds would appear red and orange during sunrise and sunset, and white during the daytime.
- **Making the sunrises and sunsets longer**. Our installation initially had the sun move at a constant velocity across the sky, so the exhibit shows a blue daytime sky most of the time. However, feedback suggested that users really liked the appearance of the sunrises and sunsets, so the animation curve of the sun's position was adjusted so that 25% of the day would be spent in sunrise, 50% in daytime, and the final 25% in sunset.
- **Decreasing wind effects.** Initial testing of the wind effects suggested that they were too dramatic, with any movement resulting in the clouds moving very quickly. With user feedback, we were able to adjust the strength of the wind effects until they were subtle enough that they were still noticeable but would not induce dizziness or motion sickness in visitors.

Verification

Verification was performed via two means - the Augmenta simulator, which allowed us to simulate, within Unity, the position and movement data that our real-life sensors would produce; and in-person testing performed at the EYEPOOL space.

Since in-person testing time at EYEPOOL was limited, most development was done locally on laptops running macOS. However, since the Unity project for the installation contained an in-project 3D representation of the exhibition space, and Augmenta allowed us to easily set up and run simulations of people walking around or standing still, and stream the simulation data directly into Unity, we were able to thoroughly test and verify the functionality of all interactions locally before deploying to the actual EYEPOOL space.

This meant that in-person testing could focus mainly on making sure that:

1. dimensions-wise, everything mapped correctly from the real-life Augmenta sensors in the space to the Unity scene, and then from the Unity scene to the 14 projectors;
2. interactions that we had implemented in Unity were intuitive and enjoyable in-person; and
3. the data smoothing that was applied to the Augmenta position data was sufficient to minimize jitter and the probability that a person that is standing still is incorrectly determined to be moving, or vice versa.