Technical Design Document:

Love on a Plate

Project Client:

Love on a Plate

Yangos Hadjiyannis, Artur Kononuk, Juan Ramón Sánchez Velar

CatFlix Developer Team:

Lead developer: Zimo Wu

Developers: Anh Vu, Sally Park

Project Advisor:

Center for Digital Media

Bill Zhao

Project time:

5/6/2024 ~ 8/1/2024

Table of Content

Introduction and Overview

The prototype, developed by Team CatFlix for the innovative startup Love on a Plate, is an immersive mixed reality experience designed to be showcased in Apple Vision Pro using Unity. This 6-7 minute experience dives into the culinary journey of Chef Nagata, highlighting the intricate process of cooking crème brûlée. The project leverages cutting-edge mixed reality technology to blend digital and physical worlds, offering users an engaging and educational culinary narrative. This document outlines the technical design and specifications necessary to develop, implement, and deliver the project successfully. The intended audience for this document includes developers and stakeholders involved in the project.

Revision History

Version 1.0

Date: 24 July 2024

Description of Changes: Initial creation of the document.

Author: Zimo Wu & Sally Park & Anh Vu

Agreed Deliverables

- Proof of Concept Prototype: A 5 min mixed reality prototype (Catflix Team committed to a full-story prototype with beginning- middle-end.) showcasing the integration of spatial video, user interactions, and 3D assets.
    - Experimentation with Mixed Reality: Blurring lines between digital and physical worlds.
    - 3D Asset Creation: Crafting minimal yet impactful 3D elements.
    - User Interactions: Focused on gaze and hand gestures.
    - Spatial Video Integration: Meaningful transitions between video content and 3D elements.
- Documentation: Detailed documentation covering research findings, development guide and design structures.

Project Scope and Changes

Initial Project Scope

The initial project scope, as outlined in the client brief, includes the development of a 45-second to 1-minute mixed reality prototype. This prototype was to incorporate spatial video and demonstrate potential interactions using passthrough and hand movements on next-generation headsets, with a preference for the Vision Pro. The key elements of the initial scope were:

- Experimentation with Mixed Reality to blur the boundaries between the physical and digital, creating seamless, immersive prototypes.
- Crafted minimal yet compelling 3D assets that enhance the storytelling aspect of the culinary experiences.
- Intuitive user interactions centered around gaze and hand gestures, making the experience accessible and engaging for users of all technical backgrounds.
- Integration of a spatial video player within the scene, focusing on meaningful transitions between video content and 3D elements to enrich the narrative. Our developer team will actively support this aspect as it is a quite novel feature.
- Comprehensive documentation outlining research findings, mockups, brainstorming sessions, and potential risks, ensuring a well-rounded understanding of the project's development process.

Final Project Scope

The final project scope, as refined and agreed upon in the project charter, extends the initial scope to develop a more comprehensive 6-7 minute mixed reality experience. This experience tells a full narrative with a clear beginning, middle, and end.

Technical Research & Limitations

Xcode & Unity versions

Unity PolySpatial and its support for visionOS require Unity 2022.3 (LTS) or later. Versions of Unity before 2022.3 cannot be supported.

Compiling for visionOS currently requires Xcode 15 beta 2 and visionOS 1.0 SDK or newer.
You must currently use an Apple Silicon (M1/M2) Mac in order to compile for visionOS.

visionOS Xcode setup

Download Xcode 15 beta 2 following the instructions in this link.

RealityKit and Immersive Spaces

To present RealityKit content, we typically use a RealityView.
There are three styles of immersive spaces using the immersionStyle scene modifier:

- Mixed Style: Blends virtual objects with passthrough, placing them in the user's real surroundings.
- Full Style: Displays only your content, turning off passthrough, ideal for transporting users to entirely new environments.
- Progressive Style: Replaces passthrough in part of the display, allowing users to stay grounded in the real world while viewing another world.

When an immersive space is opened, all windows from your app are displayed, but windows from other apps are hidden. Only one immersive space can be displayed at a time across all apps.

Unity visionOS PolySpatial Tools setup

Install PolySpatial and visionOS support following the instructions here. We also created a detailed set-up guide in the visionOS Development Section below.

Mixed Reality Apps

To develop mixed reality (MR) apps for visionOS, you need to install the PolySpatial packages along with the visionOS platform module and XR packages:

- Install the visionOS platform module:
    1. Add from Unity Hub: Go to Installs → Gear icon → Add Modules → Select visionOS Build Support → Install.
- Install the XR packages:
    1. Create a new project, then navigate to Window → Package Manager → Update XR Plug-in Management to version 4.4.1 or later.
    2. Go to Edit → Project Settings → XR Plug-in Management section → visionOS tab → Enable Apple visionOS.
- Install the PolySpatial packages:
    1. Open the Project Settings window (menu: Edit > Project Settings).
    2. Select the Apple visionOS settings under XR Plug-in Management.
    3. Change the App Mode setting to Mixed Reality - Volume or Immersive Space.

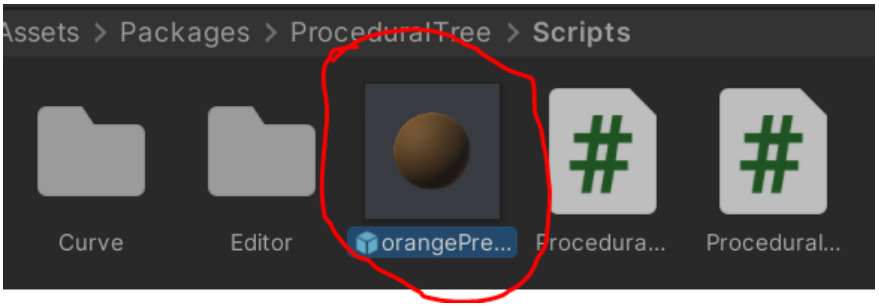
To run the project on simulator (instead of vision pro for testing)

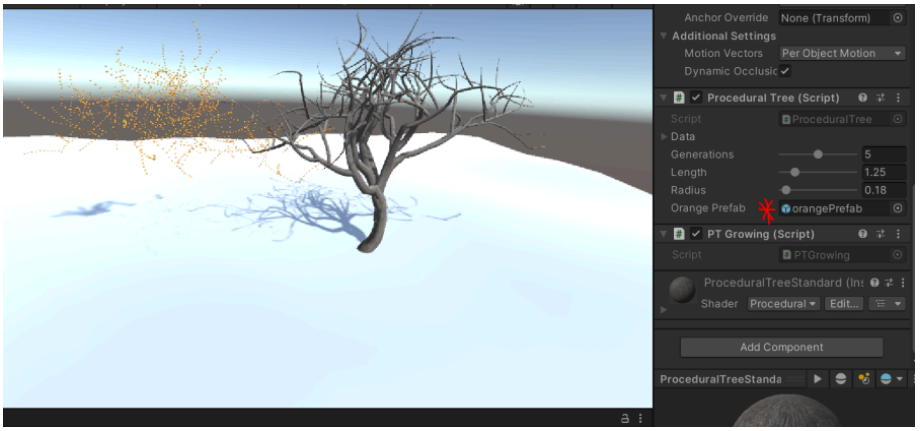
1. Go into Edit → Project Settings → Player
2. for the Target SDK setting set it to Simulator SDK
3. Build and run

Orange Tree Research & Limitations

We are currently using this github repo: https://github.com/mattatz/unity-procedural-tree

For adding oranges, we tried creating an orangePrefab (orange sphere from Unity provided assets) and appending it to the trees after they are done growing. However, this means the oranges aren't growing automatically with the tree.



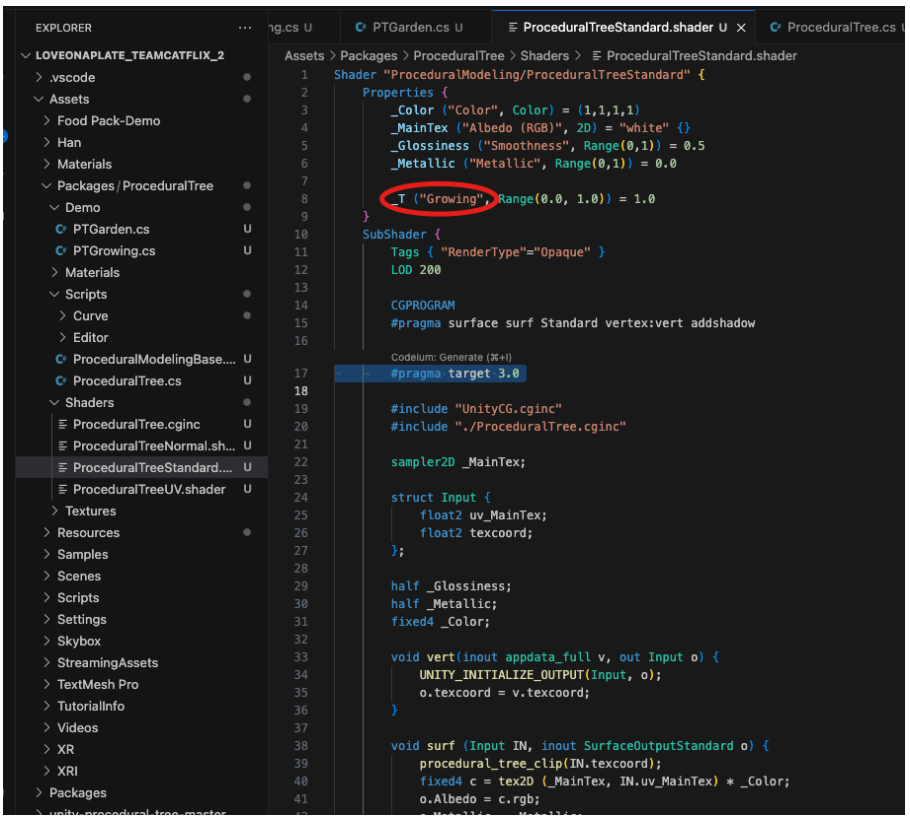


Also there's an issue seen in the demo where the oranges generated aren't set where the tree is, but rather at around position 0.

Later, we tried importing the tree repo into our own repo. However, because we're using MR, the "growing" effect of trees is lost (the trees grow fine in VR). Trees now instantly appear when hitting play : video here

We suspect that the reason is because Apple Vision currently doesn't support custom shaders and the trees use these low-level custom shaders: https://docs.unity3d.com/Packages/com.unity.polyspatial.visionos@1.2/manual/ShaderGraph.html

From looking at the shader scripts, it seems they contain instructions for growing :



We tried researching whether shaderGraph can fix this issue, and shaders are used to do calculations for displaying things on the screen.

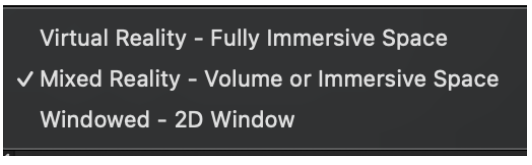
On June 11th, we reached a dead end due code limitation. The artists team will continue with making a growing orange tree with shadergraph.

Code side we tried to instantiate oranges under some segments of each branch (using probability) when play scene, use parentTransform and making oranges children of the tree so they appear relative to the position of the tree in the scene, and create a script that's used by the orange prefab to delay rendering by x seconds.

The issue with that was the oranges persisted after exiting the scene. So the trees will have more and more oranges the more scenes you play.

MR VR Transitions

Our client wants smooth transitions between mixed reality (MR) and virtual reality (VR), but Unity does not support these kinds of transitions. It only supports the entire VR full immersive mode or Mixed reality mode.



To address this issue, we have developed a solution that effectively bridges the gap between MR and VR experiences. Our approach involves leveraging Unity's volume camera system for MR and implementing a creative workaround for simulating VR.

1. Utilizing Volume Cameras for MR Experience:

- Unity offers two types of volume cameras: unbounded and bounded.
    - Unbounded Cameras
        - These cameras are not constrained by specific boundaries and provide a more immersive experience by allowing users to engage with the environment without distractions from other applications.
        - We opted for the unbounded camera setup to maintain a consistent and distraction-free MR experience, ensuring fixed spatial values between users and the mixed reality content.
    - Bounded Cameras
        - These cameras are restricted to a predefined boundary and are used for more controlled environments.
        - We chose not to use bounded cameras in this case to avoid potential interruptions and maintain the desired MR/VR experience. Bounded Camera also cannot make the following VR skybox work.

2. Simulating VR Experience:

- For the VR portion of the project, we are leveraging Unity's MR mode, but with a creative workaround to simulate a VR environment. We constructed a large sphere within the scene, employing a fake skybox texture that is applied to the sphere's interior surface. A custom script, invertNormals, is attached to this sphere to ensure that the texture appears correctly inside the sphere, effectively creating the illusion of a VR space.
- By enclosing all elements within this giant sphere, we create a controlled environment that mimics the VR experience while continuing to use Unity's MR mode.

Fading Spatial Videos

Fading the plane to which the VisionOSVideoComponent renders to does not directly affect the spatial video's alpha channel. As a workaround, another plane was placed in front of the video plane, creating the fading effect by altering the alpha of the second plane.

Hand gestures Detection

XR Interaction Toolkit

The XR Interaction Toolkit is a Unity package that provides a comprehensive framework for building XR (Extended Reality) applications. It includes tools and components for creating immersive and interactive experiences across various XR platforms, including AR (Augmented Reality), VR (Virtual Reality), and MR (Mixed Reality). Key features include:

- Interaction Components: Ready-to-use scripts and prefabs for common interactions like grabbing, pointing, and selecting objects.
- Locomotion Systems: Supports various movement types, including teleportation and direct movement.
- Interactivity: Provides a system for creating interactive objects and UI elements that respond to user input.
- Cross-Platform Support: Ensures compatibility with different XR devices and platforms.

It serves as the foundation for building interactive experiences in VisionOS. It provides a suite of tools that allow developers to implement a wide range of interactions without having to build them from scratch. This toolkit includes prefabs and scripts for common interaction patterns like grabbing, selecting, and manipulating objects in the virtual space. By leveraging these tools, developers can focus on crafting the specific interactions and gestures unique to their VisionOS application.

XR Hands

XR Hands is a subsystem within the XR Interaction Toolkit focused on hand tracking and hand-based interactions. It leverages the capabilities of devices that support hand tracking (e.g., Oculus Quest, HoloLens) to provide more natural and intuitive ways to interact with the virtual environment. Key features include:

- Hand Tracking: Detects and tracks the position, orientation, and gestures of the user's hands.
- Hand-based Interactions: Enables interactions like pinching, grabbing, and pointing using the user's hands.
- Hand Poses and Gestures: Recognizes and responds to specific hand poses and gestures to trigger actions in the virtual environment.

The XR Hands subsystem is specifically designed to handle hand tracking and gestures. For VisionOS, this involves using the system's hand tracking capabilities to interpret and respond to the user's hand movements. Developers can define custom gestures, such as pinching, waving, or specific hand poses, and map these gestures to actions within the application. This allows for a natural and intuitive way for users to interact with the virtual environment. The XR Hands system can recognize hand positions, track movements in real time, and trigger interactions based on predefined gestures.

XR Origin

XR Origin is a concept and a set of tools within Unity that helps manage the player's position and orientation in an XR environment. It is crucial for ensuring a consistent and immersive experience across different devices and scenarios. Key features include:

- Origin Point Management: Defines a central point of reference for the player's position and orientation in the virtual space.
- Camera Rigging: Manages the configuration and movement of cameras to match the player's head movements and positional tracking.
- Scaling and Room Setup: Adjusts the virtual environment's scale and layout to match the physical space and ensure a comfortable experience.

The XR Origin component is essential for managing the player's position and orientation within the VisionOS environment. It ensures that the virtual experience aligns with the physical movements of the user. By setting up a central reference point, or origin, developers can create a consistent spatial relationship between the user and the virtual world. This helps maintain immersion and ensures that hand gestures and interactions feel natural and responsive. The XR Origin component handles camera rigging and positional tracking, adjusting the virtual environment to match the user's movements and providing a seamless experience.

Customization

With the help of our advisor, and addressing the specific requirements of our client, we customized XR interaction toolkit and related scripts to implement the grabbing gesture for our story. The related process and scripts will be discussed in the VisionOS Development Guide and the Class Overview section.

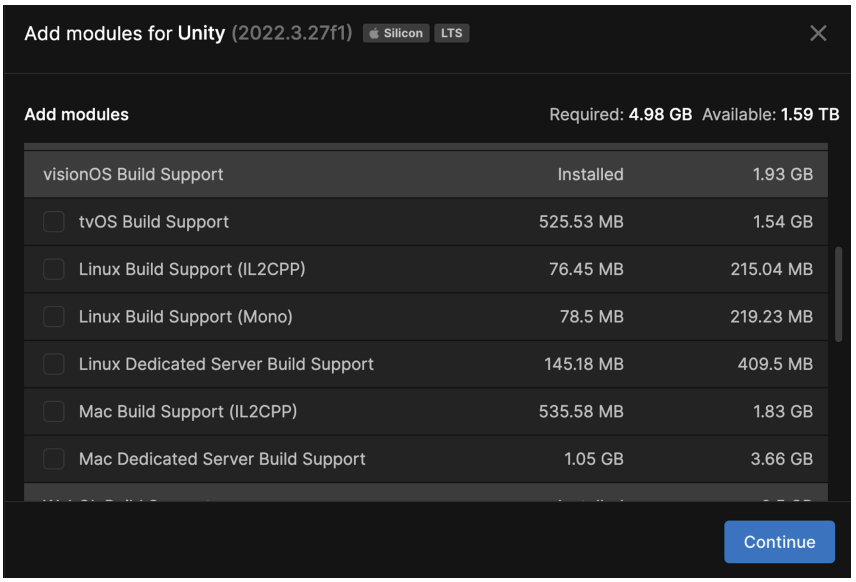
VisionOS Development Guide

Set up Guide

You will need an Apple Silicon (M1/M2) Mac to compile for visionOS.

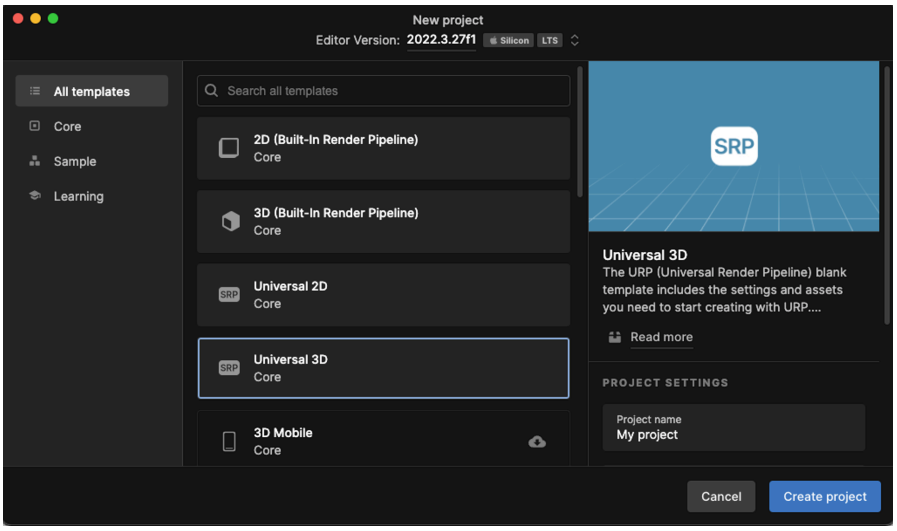
We are using Unity version 2022.3.27 and Xcode 15.3 for this project. You also need Unity Pro or a student account for Unity to work on visionOS.

- Install the visionOS platform module
    - add from Unity Hub (installs → gear icon on Unity 2022.3.27 → add modules → select visionOS Build Support → install)

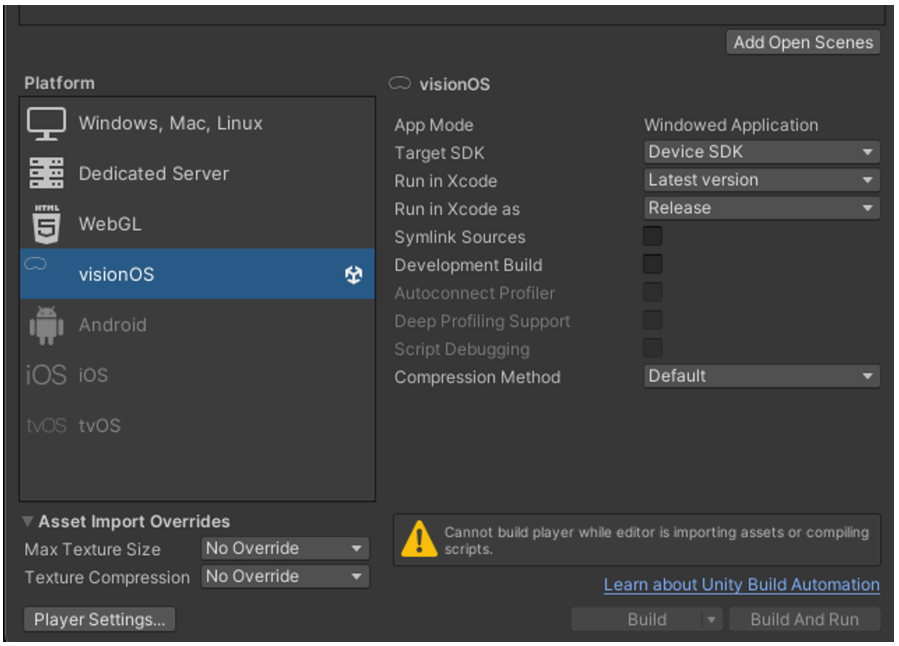


After you install the correct version of Unity, you can create a new Universal 3D project using the Universal Render Pipeline.

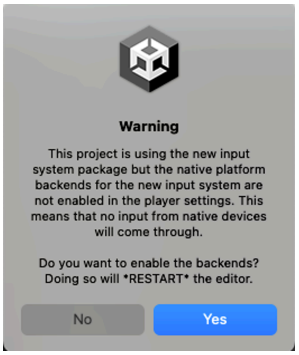
Note: You can either use the Universal Render Pipeline (URP) or the Built-in Render Pipeline. It's recommended to use URP since it allows you to use tools like Shader Graph and access all PolySpatial features.



- Build Settings → Switch Platform to visionOS

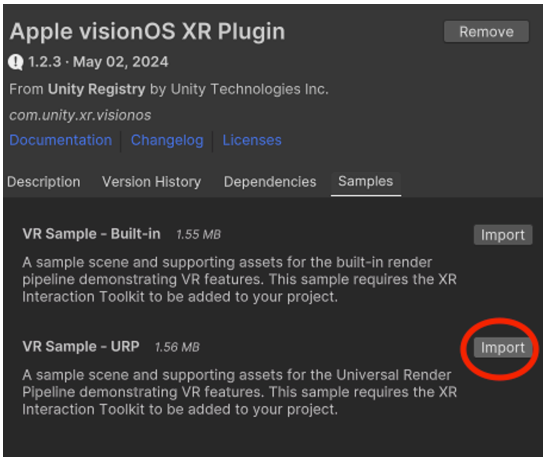


- Package Manager
    1. add by name:
        - com.unity.xr.visionos

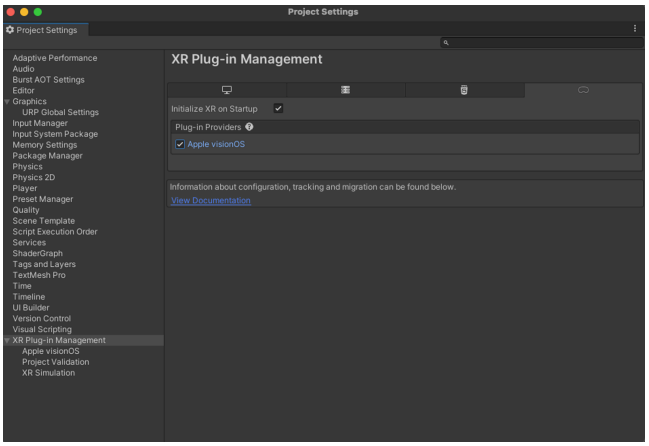


Choose Yes otherwise it will appear every time you enter Unity.

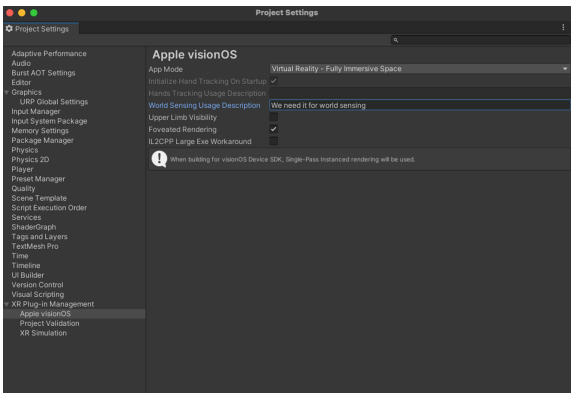
2. import VR Sample - URP



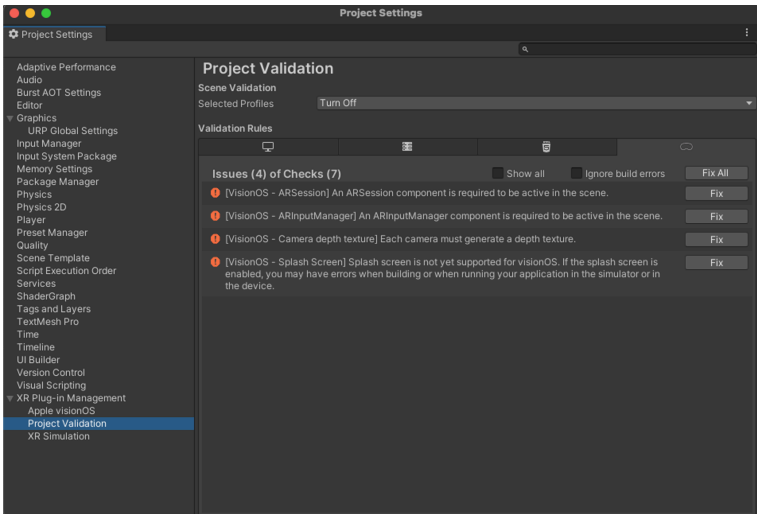
3. Player Settings / XR Plug in management, click on Apple visionOS



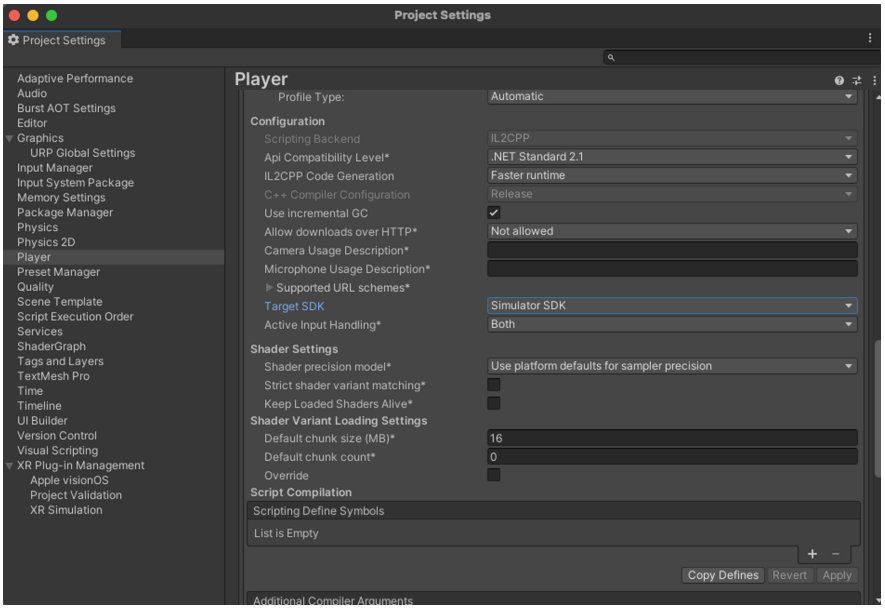
Fill in - this one not necessary because can be do later



Fix all - make sure to fix all the things

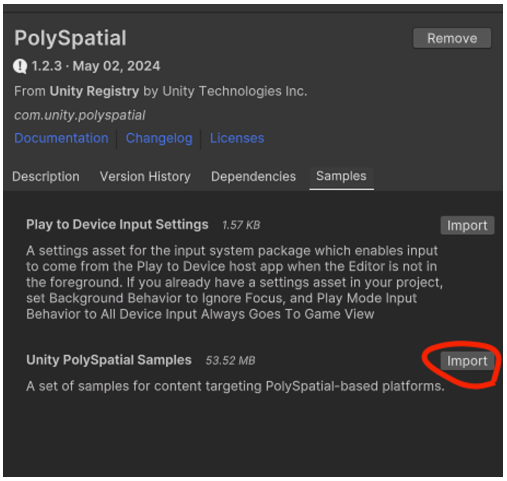


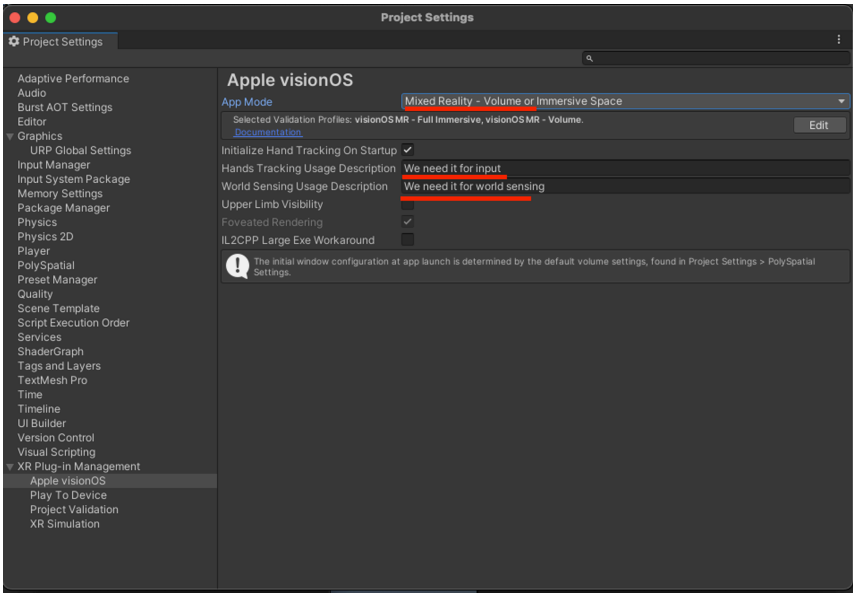
Change to simulator if no vision pro device



4. Package Manager

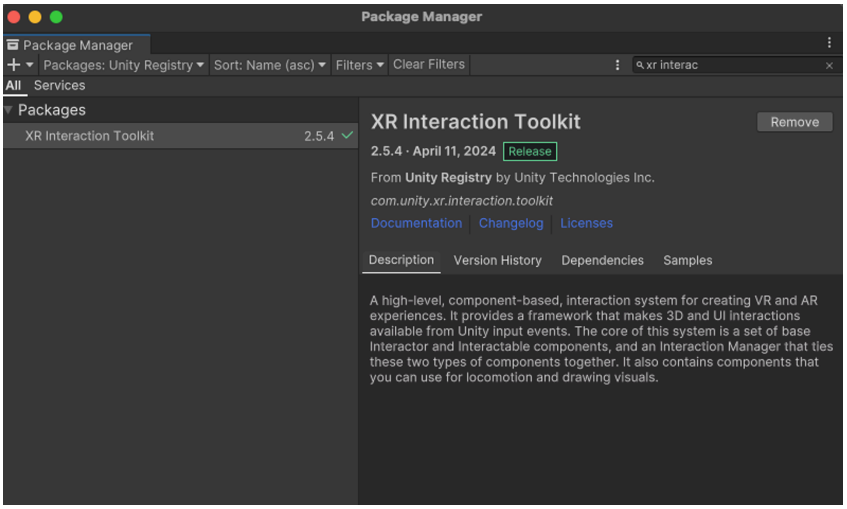
- add by name:
  - com.unity.polyspatial
  - com.unity.polyspatial.visionos
  - com.unity.polyspatial.xr
- import Unity PolySpatial Samples



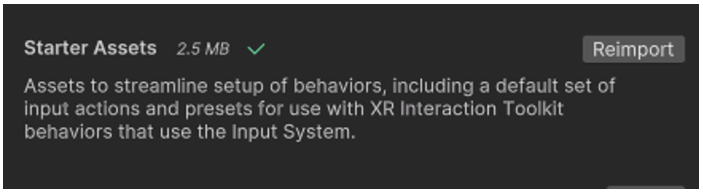


XR Interaction Toolkit

com.unity.xr.interaction.toolkit



import starter assets

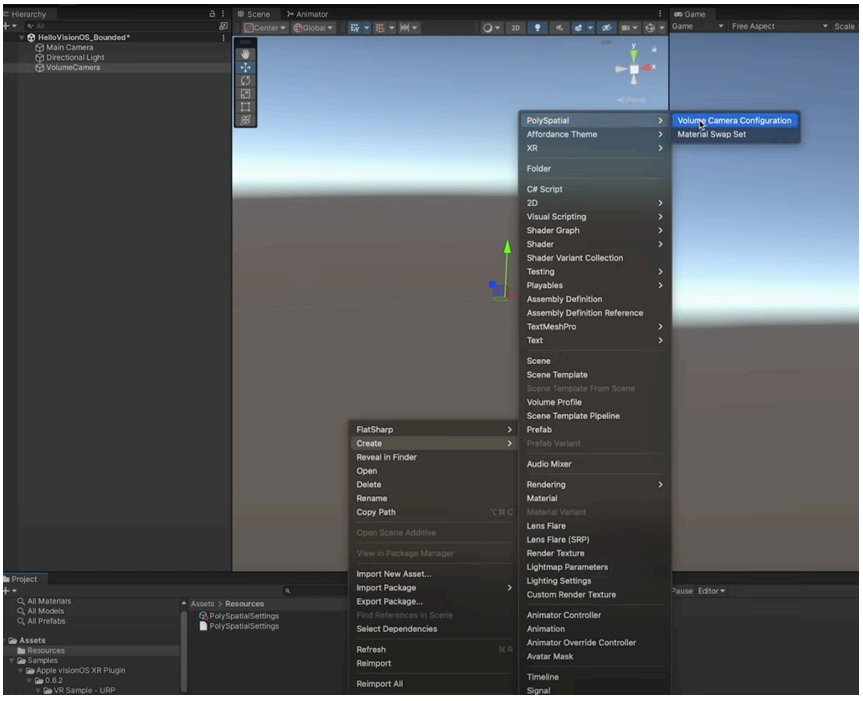


Open a PolySpatial sample scene

Assets/Samples/ PolySpatial/ Scenes

Create a new scene

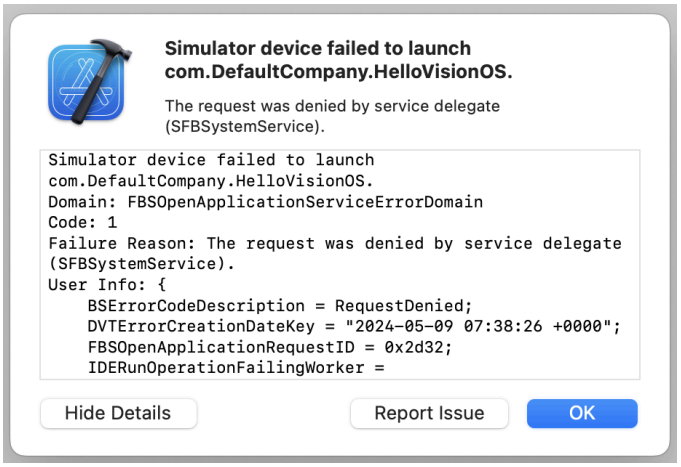
1. Create a volume camera
2. go to resources



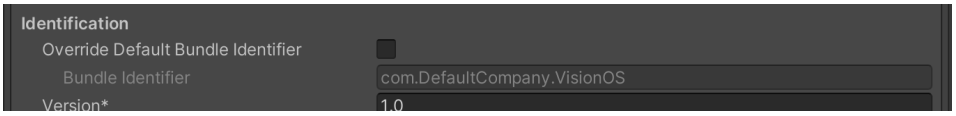
3. Assets/Samples/PolySpatial/Shared/Models

Bugs

- Materials sometimes don't match between Unity Editor & Simulator
- Unity rendering issue - objects not showing up on simulator



- Solution: Player setting → Uncheck override default bundle identifier:



Play to Device

💻 Tech Stack

- PolySpatial Version 1.2.3
- Unity version 2022.3.27f1
- Xcode 15.3 & higher

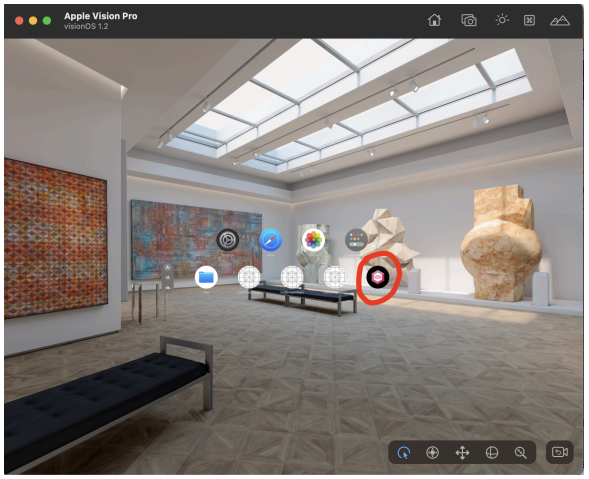
Check the compatibility matrix on the Unity documentation page for the corresponding versions

1. https://docs.unity3d.com/Packages/com.unity.polyspatial.visionos@1.2/manual/PlayToDevice.html

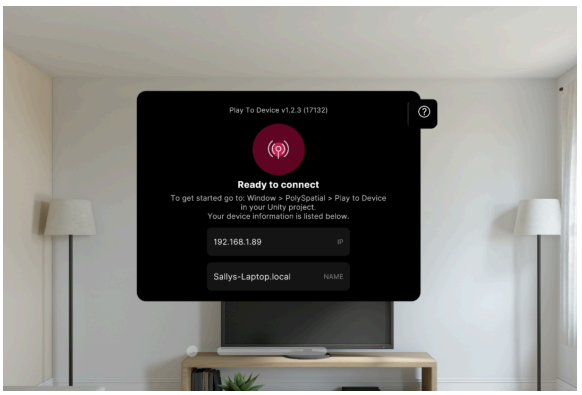
PolySpatial Version	Supported Unity Versions	Required Xcode Versions	Required Firmware Version	Xcode .App Link (Apple Silicon)	Device TestFlight Link
1.2.3	2022.3.19f1 and higher	Xcode 15.3	visionOS 1.1 SDK	Download Link 709	Join the Unity Play to Device Host beta - TestFlight - Apple 632
2.0.0-pre.3	6000.0.0f1 and higher	Xcode 15.3	visionOS 1.1 SDK	Download Link 709	Join the Play To Device v2 beta - TestFlight - Apple

Set Up For VisionOS Simulator (Macbook Pro/Mac)

1. Download the Xcode .App files using the link https://drive.google.com/drive/u/0/folders/1jwF8mlARTVs43z0Xm3vO2WW-9-T2bdr9
    a. Download the "PlayToDeviceHost.app.zip" (See the Compatibility Matrix to identify the right version given your PolySpatial version)
2. Extract the downloaded zip file
3. Start the visionOS simulator: "Xcode > Open Developer Tool > Simulator"
4. Drag the extracted "PlayToDeviceHost.app" from Finder into the simulator
5. After a few seconds, you should see "PlayToDeviceHost" appear as one of the app icons on the home screen



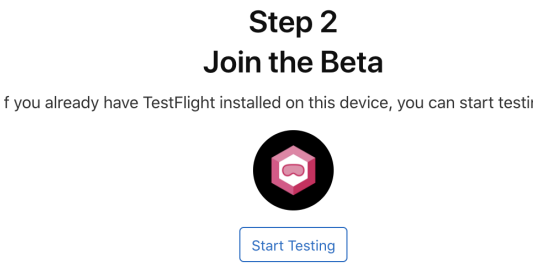
6. Launch the PlayToDeviceHost app



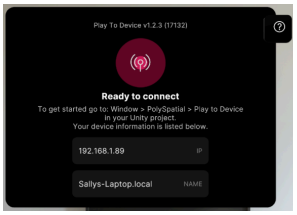
Set Up For Vision Pro Device

1. In the device, open the TestFlight Link that corresponds to your PolySpatial version from the compatibility matrix
    a. PolySpatial 1.2.3: https://testflight.apple.com/join/FVMH8qiG
2. Install the TestFlight app on the Vision Pro

3. Once installed, click start testing



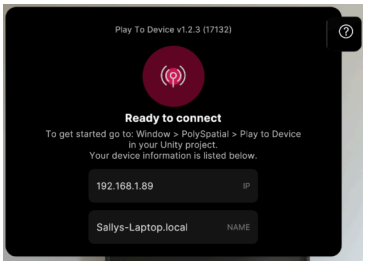
4. In the TestFlight app, install the PlayToDevice Host app
5. Once completed you should see this screen



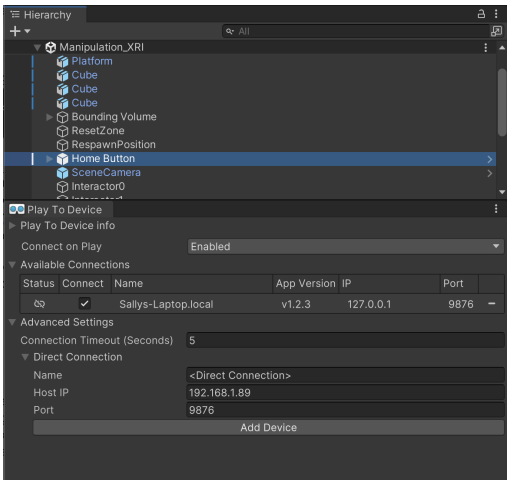
Launch Your Project From Unity

1. In the Unity Editor, open the Play to Device Editor window: Window > PolySpatial > Play to Device.

2. Enable Connect on Play
3. Add your device
    a. In the Advanced Settings, type in the Host IP in the host app on your simulator/device



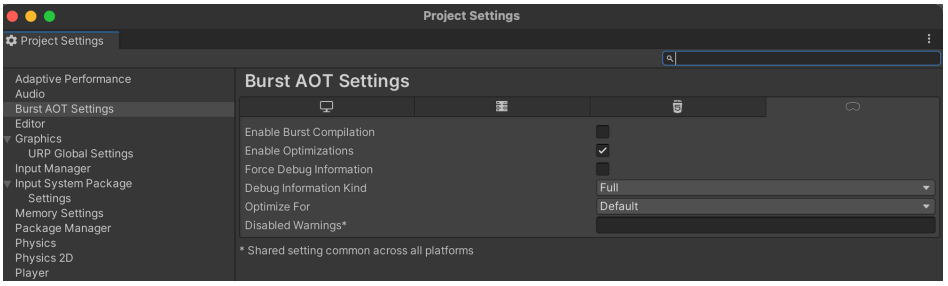
    b. set a Host Name
    c. click 'Add Device'
4. Select your device to connect to by checking the Connect toggle in the Available Connections list



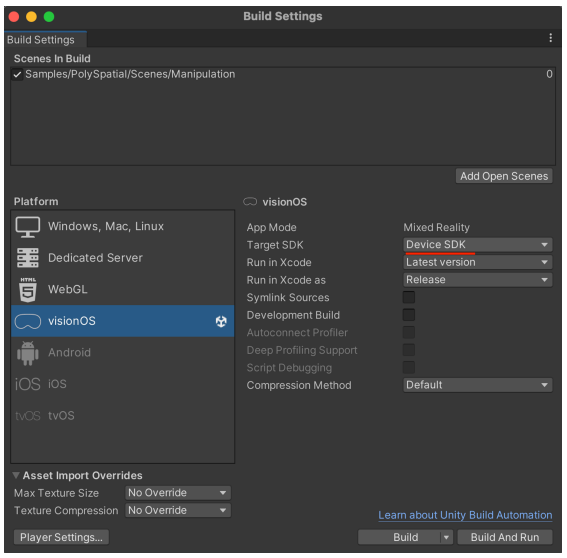
5. Enter Play mode in the Unity Editor

Deploying to Vision Pro

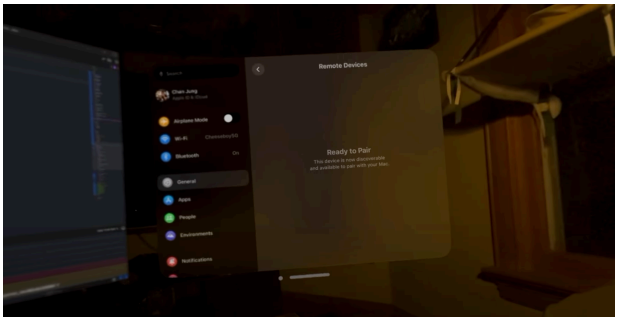
1. Go to Unity/Project settings/Burst AOT Settings, flipping to the visionOS tab, and disabling Enable Burst Compilation



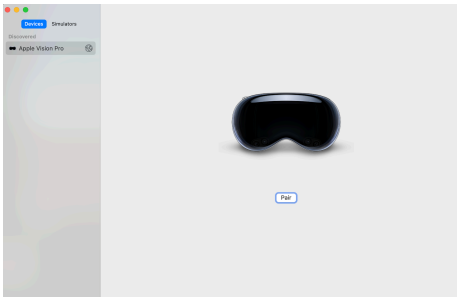
2. Make sure Xcode is updated to the newest version and reboot computer
3. Make sure vision pro and mac and Xcode are using the same apple ID
4. Turn Unity settings to Device before Build



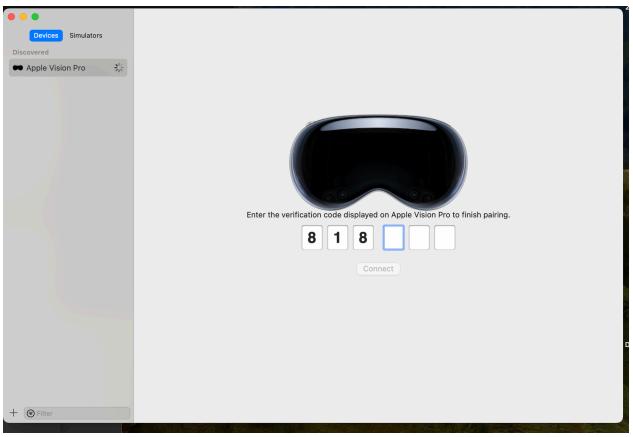
5. Build the project
6. Open Xcode file
7. Manage
8. Inside Vision Pro, Go to Settings/General/Remote Devices, ready to pair



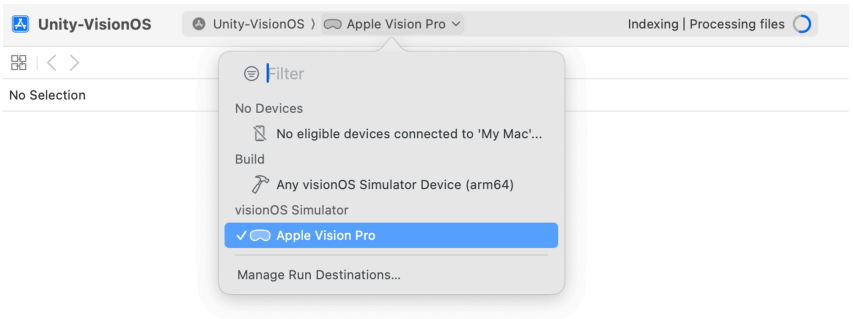
9. Use USB to connect Vision pro and mac
10. Make sure vision pro and mac are connected to the same wifi
11. Vision Pro should show up on X-code



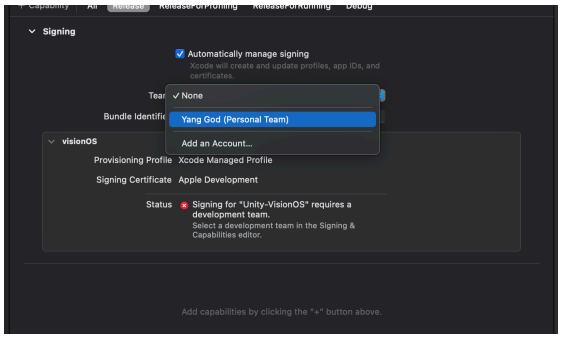
12. click on connect
13. A number will appear in vision pro, fill in the number in Xcode

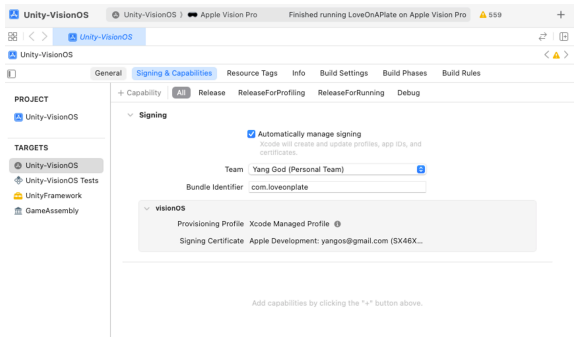


14. Choose apple vision pro



15. Go to settings in Xcode, and change
    a. Go to Unity-VisionOS and Unity-VisionOS Tests, enable automatically signing, Team, etc.





16. Build

If vp not showing up

reboot everything if following not working:

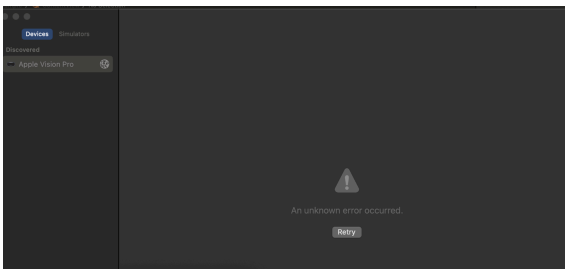
- disable wifi on computer
- enable
- make sure Vision Pro is connected to the same wifi

- if still not showing, restart Vision Pro

If there are errors,

To open your current directory in Finder from Terminal, type open .

Unknown error



Reset the network configuration on the Mac.

- cd to /Library/Preferences/SystemConfiguration
- remove all .plist file except com.apple.Boot.plist
- reboot mac
- reconnect your Apple Vision Pro and Mac

Burst compiler errors

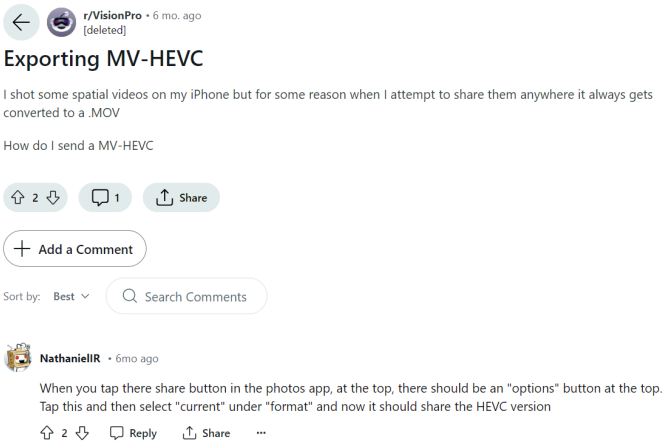
sudo xcode-select -s /Applications/Xcode-15.2-beta1.app

sudo xcode-select -s /Applications/Xcode.app

See video for the whole process.

Spatial Videos

To display spatial videos on visionOS with PolySpatial, the video should be in MV-HEVC format.



You can use either the VisionOSVideoComponent or Unity's default VideoPlayer. Each has its own pros and cons.

VisionOSVideoComponent:

- Setup: Add to a GameObject, set properties like Target Material Renderer (MeshRenderer) and Clip (Video Clip asset).
- Requirements: The video clip must be manually copied to ../StreamingAssets/VisionOSVideoClips/ folder within the project.
- Advantages: Utilizes visionOS's native video player, supports 3D spatial video (MV-HEVC video), and offers better performance.
- Limitations: Limited public API, lacks complex functionality (e.g., Events and Delegates), no access to video texture through Unity, and incompatible with debugging tools like PlayToDevice.

Unity VideoPlayer:

- Setup: Set Render Mode to Render Texture, provide a RenderTexture, apply it to a material and renderer, and add a script to call PolySpatialObjectUtils.MarkDirty() each frame.
- Advantages: Access to extensive methods, events, and delegates, allows streaming over a network, and compatible with PlayToDevice. Video texture can be used in shader graph materials.
- Limitations: Performance cost due to the need to update render texture each frame.

Comparison:

- VisionOSVideoComponent: Better performance and 3D spatial video support but limited API and functionality.
- Unity VideoPlayer: Greater functionality and flexibility but higher performance costs.

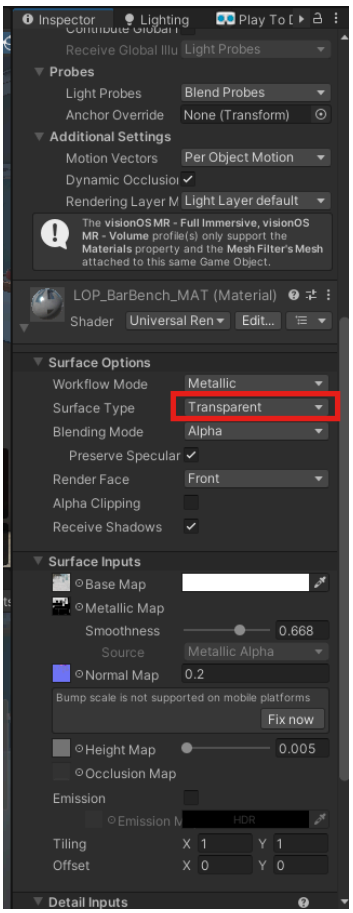
Fade in/out using Unity animator

Select the object that you want to fade in/out

In the inspector, under material, change the surface type to “Transparent”.

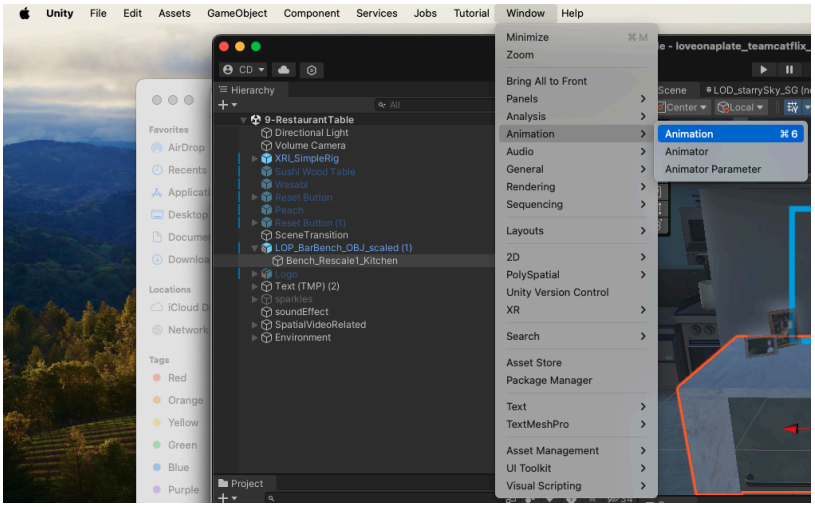
“Transparent” enables the fading effect effect for objects when you change the alpha color value.

The alpha value determines how see-through the object is, where 1 is fully visible, while 0 is transparent (but still visible).



To create animations:

Go to Windows → Animation → Animation, then select the object to create an animation for it:



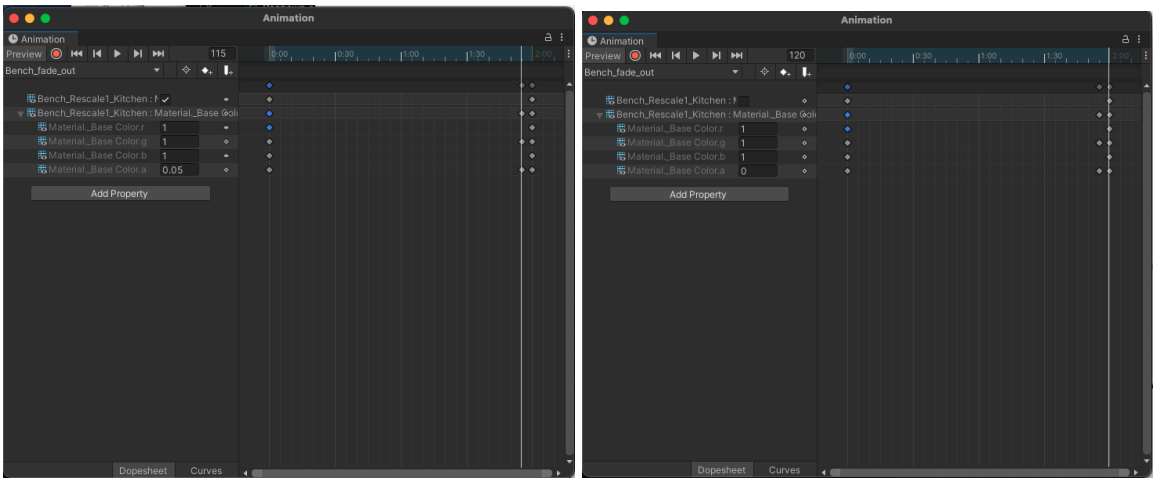
Click the red record button, then Add Property, then select Enabled and Base Colors from the Mesh Renderer.

Right click on the page and Add Key. You'll need 3 keys for fading out:

1st: set Enabled on and alpha value 1 → Object is fully visible

2nd: after x seconds, set alpha value to 0.05 or 0 → Object is transparent but still visible

3rd: set Enabled off to make the object disappear



Press play to check if the object is fading out

Do the opposite to create a fading in animation

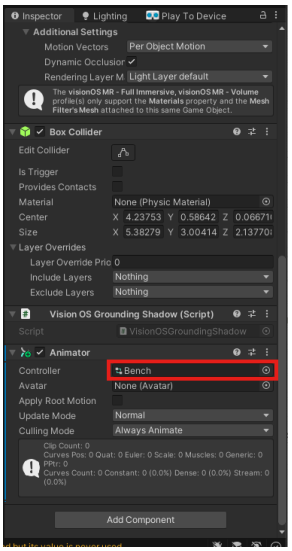
Animation Controller:

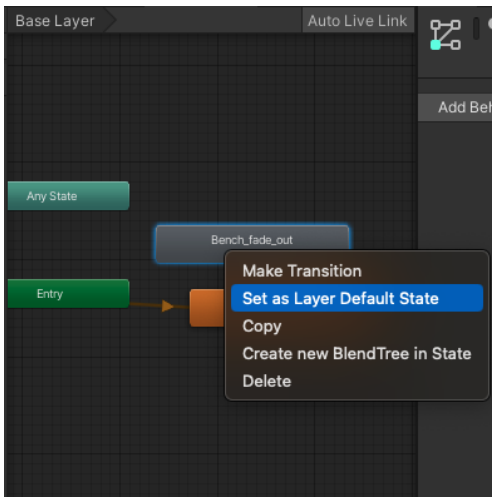
You need an animation controller to manage all the animations created

Select the animation controller in the Inspector:

In the controller, you'll have the fade out animation and a default animation. The default animation is the object being visible over x seconds. Check the Loop Time so the object continues being visible.

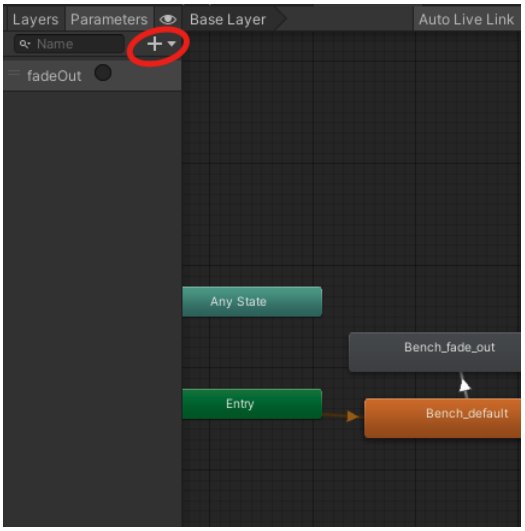
Set default as the starting state:



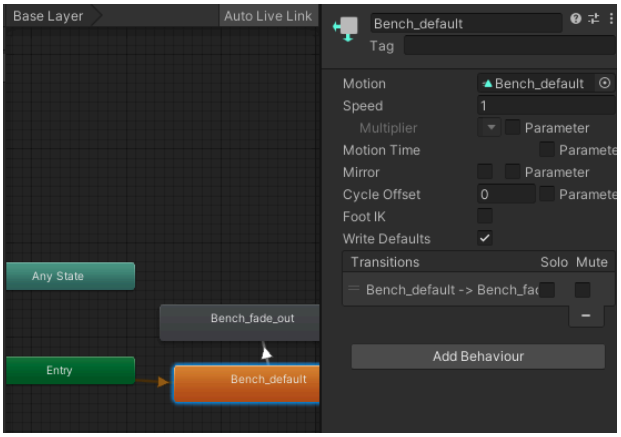


Link default to fading out by right clicking on default and “Make Transition”

To go from default to fading out, you need a “Trigger”. Go to “Parameters” → create a new Trigger using the “+”:



Now select the default state, add the trigger in the transition section:



In code add the object's animator to the script:

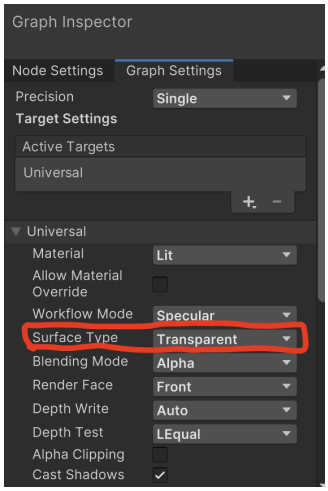
```
pulbic Animator AnimatorObj;
```

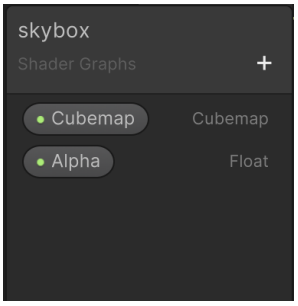
Then when you want the object to fade out, call:

```
AnimatorObj.SetTrigger("TriggerName");
```

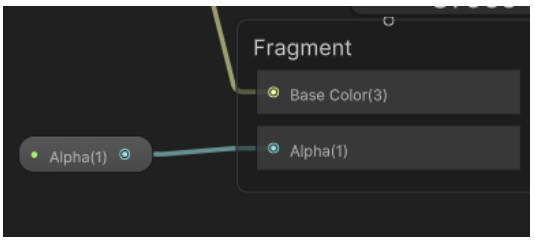
Fade in/out using script and ShaderGraph

Fade in / Fade out game objects that use shader graph materials

1. Ensure that the Surface Type is set to Transparent in the Graph Settings in Shader Graph
    a. 

2. Control material alpha channel from inspector:
    a. open shader graph
    b. press the "+" button
        i. 

    c. add a float property
    d. name it Alpha
    e. drag the Alpha property to the main graph
    f. attach it to the alpha in the master node

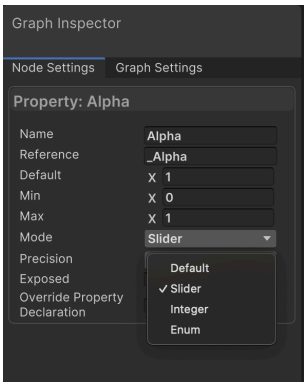
i. 

ii. open/look at the graph inspector

iii. tick on the “Exposed” property to see the property in the inspector

iv. change this to a slider

1. Mode -> slider:

a. 

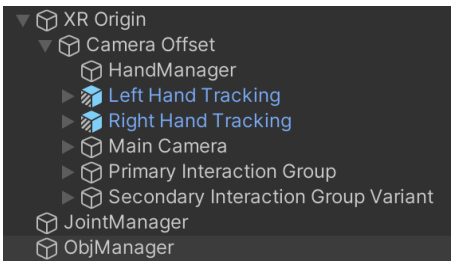
3. Attach the material to the gameobject
4. Fade the object using a script by accessing the alpha property (_Alpha)
    a. use the SetFloat(String property, float value) method to change the alpha value
        i. example: material.SetFloat(“_Alpha”, value)
            1. material is of type Material

XR Interaction Toolkit, XR Hands & modifications

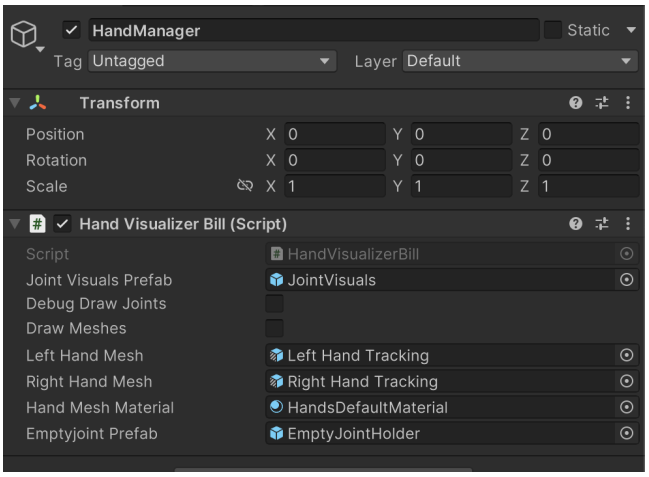
As outlined in the Technical Research & Limitations section, our primary approach to implementing hand grabbing poses for user interactions involves using the XR Interaction Toolkit along with XR Hands and XR Origin. However, due to the lack of full support for Vision Pro and the inability of the original SDK to accurately detect grabbing gestures, we customize the SDK’s HandVisualizer code with assistance from our advisor, Bill. The key scripts involved in this customization include: HandVisualizerBill, JointObjVisionPro, JointManagerVisionPro, and ObjManagerLeftAndRight. These scripts will be detailed further in the class overview.

How to reuse the feature Grab

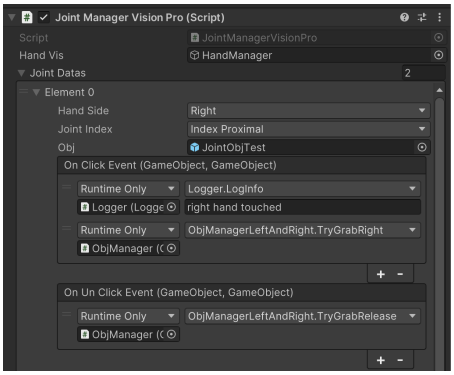
Inside a Unity scene, you will need to have all the things in the following screenshot:

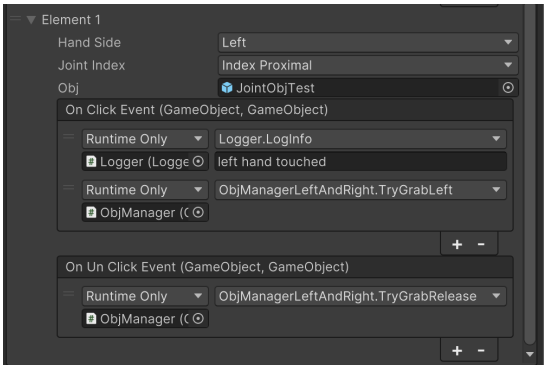


1. XR Origin with a customized script called HandVisualizerBill attached to the HandManager GameObject

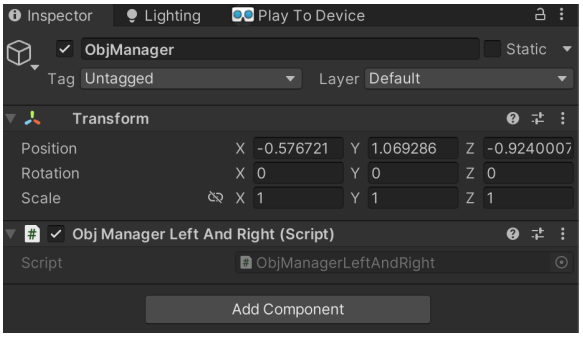


2. JointManager

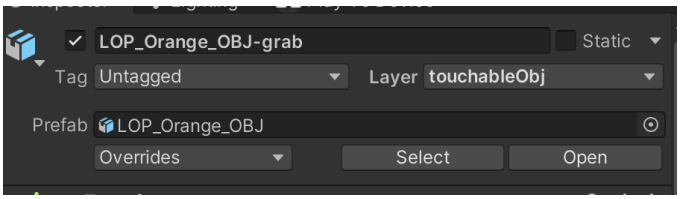




3. an ObjManager GameObject with the script ObjManagerLeftAndRight attached

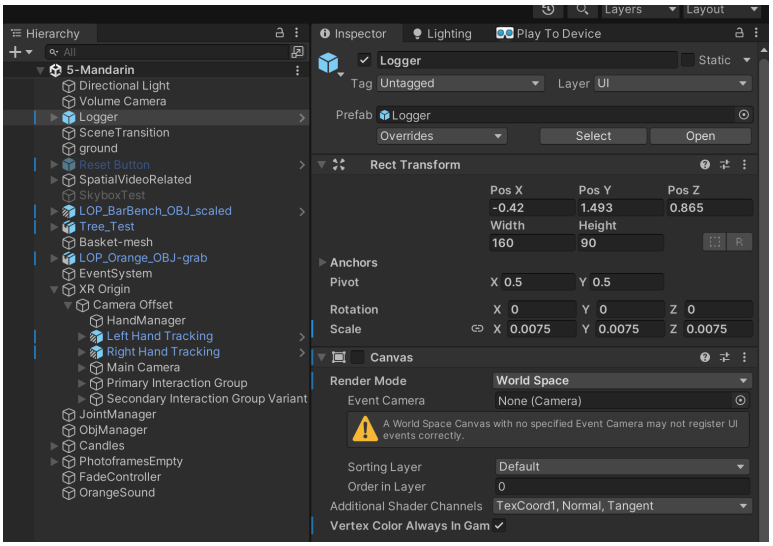


4. The object that the users can interact should have the Layer “touchableObj”



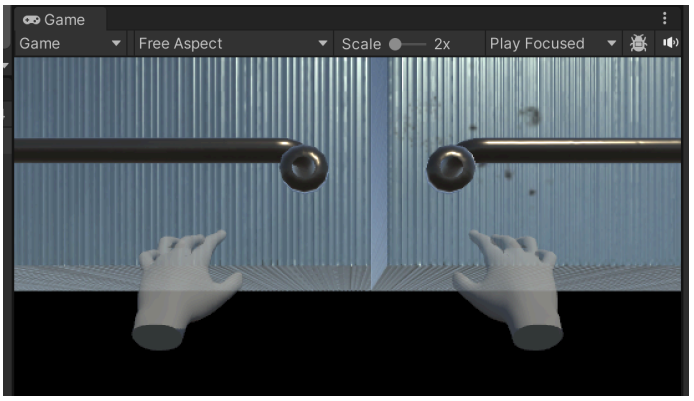
Logger

The Logger can be opened for debugging purposes by enabling the Canvas on the Logger object.



XR Origin

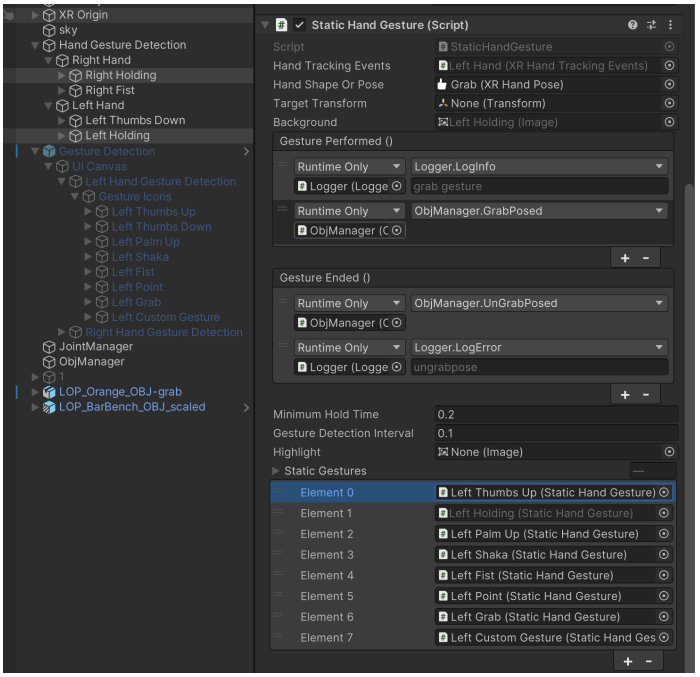
To ensure accurate alignment of the hand mesh with the user's real hands, the XR Origin and its CameraOffset should be configured to the position (0, 0, 0). Any modification of this position for improved visibility in Unity may result in the loss of the grabbing functionality in VR, as the hand mesh will appear displaced. We have to use this setting for now for the grabbing functionalities, but this definitely can be dived deeper in the future as it's hard to debug in unity.



Previous Version

Initially, we utilized the grab and hold gesture from Unity's Sample scene. When this gesture is detected, and the hand collides with an object, the collision is recognized by the

JointManagerVisionPro Script. Consequently, the object is attached as a child of the hand. If the distance between the user's index finger and thumb decreases and the ungrab pose is detected, the object is detached from the hand and subjected to gravity, causing it to fall.



It also needs the Hand Gesture Detection and Gesture Detection in the hierarchy. Although Gesture Detection is disabled, the static Gestures in the Left Holding or Right Holding refers to them so do not delete them.

However, during testing and debugging, we encountered challenges with detecting the grab pose within the SDK, along with issues such as conflicts between the left and right hands. As a result, we decided to remove Unity's default grab condition and instead detect a grab based solely on the proximity of the index finger and thumb.

Shader Graphs

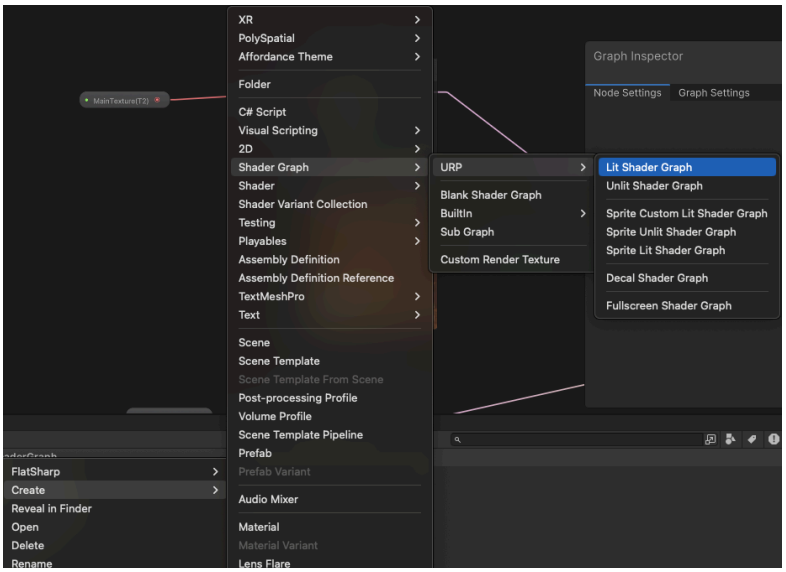
Shader Graph is a tool in Unity that allows you to create shaders using a visual interface and nodes rather than code

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

1. Prerequisites:

1. Unity 2019.1 or later.
2. Install Shader Graph:
    a. Go to Window → Package Manager
    b. Search for Shader Graph, and click Install.
3. Set Up URP/HDRP: Install and configure URP or HDRP using the Package Manager
    a. guide to installing URP to an existing project: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/InstallURPIntoAProject.html

2. Creating a Shader

1. Create a Shader Graph:
    - Right-click in the Project window.
    - Navigate to Create → Shader → ___
        i. select the shader type you need
        ii. 

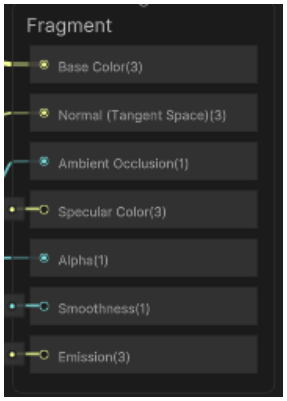
2. Double-click the created shader graph to open the Shader Graph editor
3. You can create and connect nodes to other nodes
    - connect the output node to an input node

3. Applying Your Shader To GameObjects

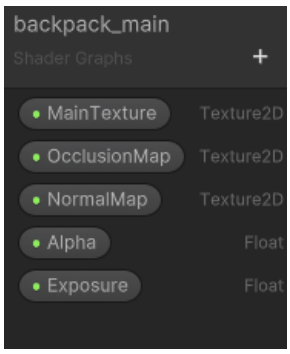
1. Right-click the Shader Graph and select Create → Material
2. Apply the material to a 3D object in your scene

4. Nodes and Components in Shader Graph

Master Node

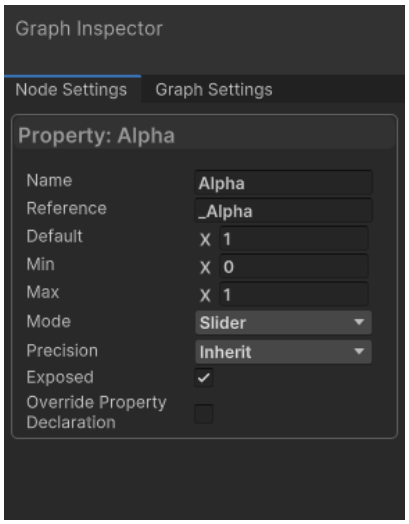


Property Nodes

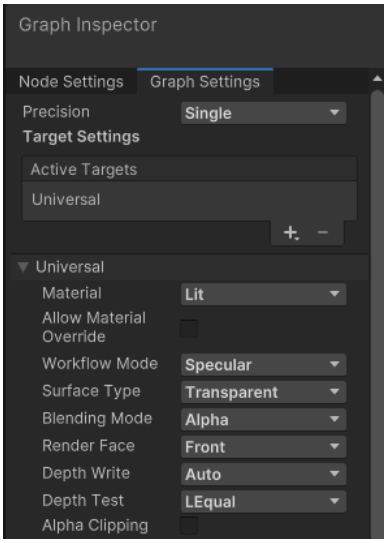


1. you can create your own property nodes by clicking the “+” button and selecting the type (Float, Vector2, Vector3, Bool, etc.)
2. these can be connected to a node in the Graph and can be exposed in the inspector

Graph Inspector

1. Node settings for a float-type property
   a. 

   b. Exposed: displays the property/node in the inspector under the shader
   c. Mode: display mode of the property
2. Graph settings

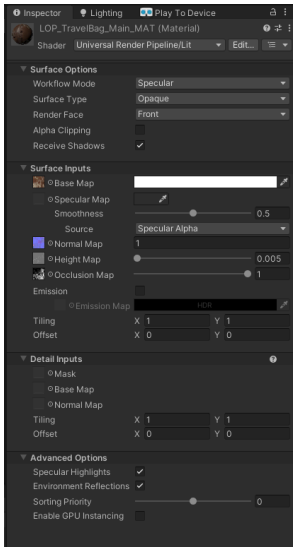


- allows configuration of global settings and rendering pipeline, customising your shader

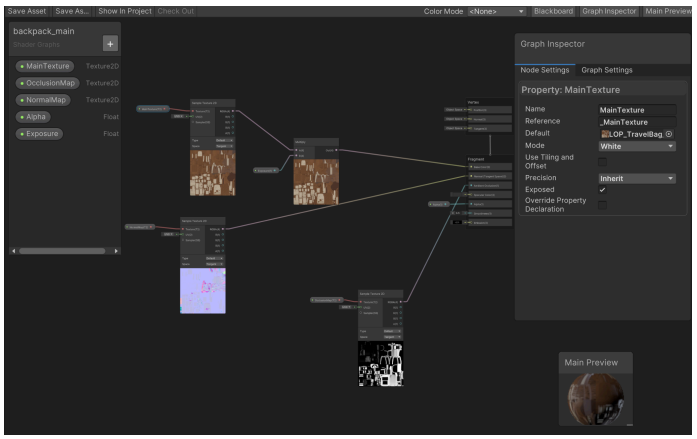
Utility Nodes

1. Add: Adds two values
2. Multiply: Returns the result of input A multiplied by input B
    a. can handle different data types, such as scalar (float), vector (Vector2, Vector3, Vector4), and textures
3. Lerp: Linearly interpolates between two values
4. Texture: Samples a texture
5. Time: Provides time-based values for animation
6. More Nodes Here: https://docs.unity3d.com/Packages/com.unity.shadergraph@6.9/manual/Node-Library.html

Example

- converting a Material to Shader Graph Material
- URP Lit:
    - 

- Shader Graph version:



Hints and Tips for working with Vision Pro

- Force Quit
    - hold the two buttons at the top of the device to force quit an application.
- Align
    - hold the right digital crown to re-align.
- Screen Record
    - if looking up cannot see the drop down arrow or the green dot, you can press the left button first to go to the camera and then look up at the green dot.
- Eyes & Hands setup
    - if you press the left button four times, it will enter eyes & hands setup
    - you can also find it in the settings
- You need to click trust in the VPN settings when you first deploy a project to vision pro using an apple account

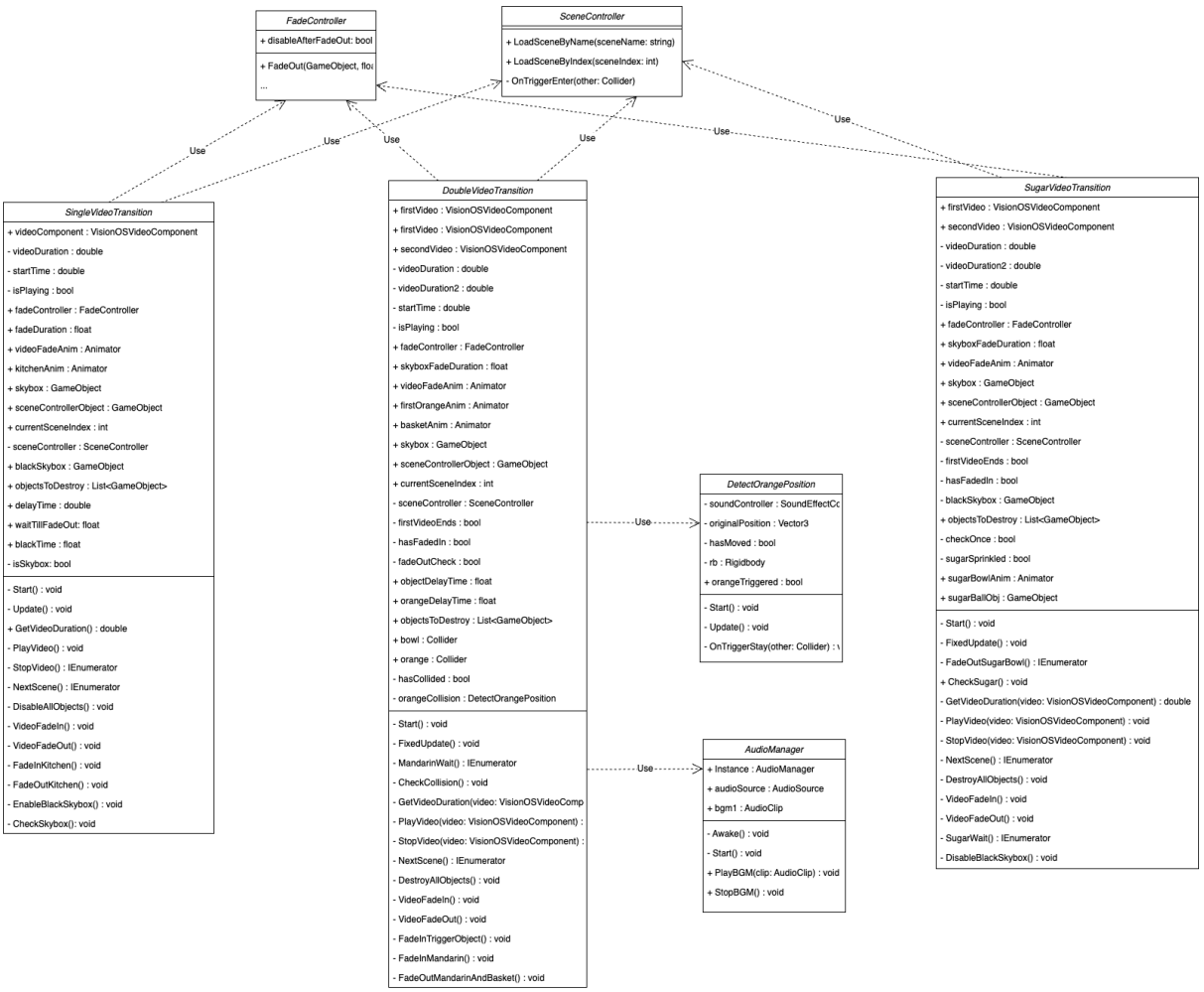
Class Relationship

Most of our classes are designed to be straightforward, each tailored to the specific scenes to meet the client's story requirements. The primary reusable classes are FadeController and SceneController. FadeController manages the skybox fading effect for each scene, while SceneController handles transitions between scenes.

We also have three scripts related to playing spatial videos and managing scene events: SingleVideoTransition, DoubleVideoTransition, and SugarVideoTransition. These scripts were initially intended to be reusable. SingleVideoTransition is designed for scenes that play a single video and is used in all such scenes. DoubleVideoTransition is meant for scenes with two sequential videos, where the second video triggers a transition to the next scene upon completion. However, due to differences in how user interactions are handled—collision detection in the orange scene versus a flag set to true when the user grabs sugar in the sugar scene—we created a new script, SugarSceneTransition, which has a similar structure to DoubleVideoTransition but is specifically for the sugar scene. As a result, DoubleVideoTransition (which would be more appropriately named MandarinVideoTransition) now only handles the orange scene.

Due to time constraints and the need to accommodate client-provided videos at the last minute, we did not have the opportunity to fully clean up the repository or refine the codebase for better reusability. This led to some code duplication and similar scripts that should be improved in the future.

The following is a UML class diagram with dependencies for our main scripts.



See Appendix Figure 1 for a class diagram with dependency for hand gesture detection part.

Detailed Design overview

Scenes Overview

To enhance clarity and streamline the development process, we established a naming convention for the scenes. This convention includes scene numbers and main identifiers, making it easier to differentiate and locate scenes within Unity. Given that our MR project incorporates both mixed reality and simulated VR environments through the use of fake skyboxes, we structured the scenes to reflect both narrative segments and significant environmental changes.
Scenes are categorized based on narrative progression and major environmental changes, such as skybox transitions. This structure enables developers to work on distinct segments of the project more effectively. It also simplifies the management of interactions, videos, and 3D assets, ensuring a more organized and efficient development workflow.
However, many scene names were initially set with placeholder titles for testing purposes. Due to time constraints towards the end of the project, we completed a general cleanup of the project but were unable to undertake a more comprehensive refinement and rename the scenes as initially planned.

1-Start

Description: Introduction scene setting the context for the experience

Key Elements:

- Love on a Plate Image
- Interactable start button
- Background music to set the mood

Main Controllers/Managers:

- SceneTransition(GameObject), SceneController(script) - transitions to the next scene

2-TestText

Description: A beginning text scene that introduces Chef Nagata and his story.

Key Elements:

- Two parts of texts
    - Crafted with Love
    - intro of Chef Nagata
- Background music continues

Main Controllers/Managers:

- SceneController - transitions to the next scene

- Text(TMP) (GameObject), TextFade(script) - controls the timing of fading in/out text and also changing text content

3-Bell

Description: A mixed reality scene with a virtual bench and a hotel bell, simulating a virtual bar waiting for food.

Key Elements:

- Virtual bench
- Hotel bell
- Background music changes

Main Controllers/Managers:

- SceneController - transitions to the next scene
- HotelBell (GameObject) - has two parts: a normal body and an EmergencyStopOffset which is the animated part
    - XRSimpleInteractable - the simplest Interactable object which provides events for interaction states like hover and select.
    - XR Poke Filter - filter component that allows for basic poke functionality and to define constraints for when the interactable will be selected.
    - XR Poke Follow Affordance - used to animate a pressed transform, such as a button (hotel bell in this case) to follow the poke position.

4-Kitchen

Description: A virtual environment of chef Nagata's kitchen where the chef starts preparing the creme brulee and talks about his story

Key Elements:

- Detailed 3D models of kitchen appliances and utensils
- Kitchen bench
- Spatial video
- 3D backpack model and animation of knives, book, map falling in
    - the animation begins when the chef begins talking about his travels

Main Controllers/Managers:

- EditedSpatialVideo(GameObject), SingleVideoTransition(script) - Transitions to the next scene when the video ends
- BackpackController - Controls the timing of the backpack animation
- SceneController - transitions to the next scene

5-Mandarin

Description: Introduction scene setting the context for the experience

Key Elements:

- Mandarin tree - with growing animation
- Mandarin
    - fade in when Chef in video points at the tree
    - mandarin sound effect
    - have gravity after grab/falls of the tree
    - freeze position and rotation after put into basket
    - fade out two seconds after the second video plays
- Basket
    - fade in after the mandarin
    - fade out two seconds after the second video plays
- Bench
- Mandarin field skybox
- Spatial video
- Background music continues

Main Controllers/Mangagers:

- SceneController - transitions to the next scene
- SpatialVideosController(GameObject), DoubleVideoTransition (script)
    - Plays two videos
    - after the first video ends, only user interaction of grabbing mandarin and put into basket will trigger the second video
    - transitions to the next scene when the second video ends
    - controls fade in/out skybox and objects
- HandManager(XR Origin/CameraOffset/...), HandVisualizerBill (script) - modifies the original HandVisualizer for visionOS
- JointManager(GameObject), JointManagerVisionPro - detects collision with both hands
- ObjManager(GameObject), ObjManagerLeftAndRight (Script) - controls grab and release objects

6-GrandmaPlace

Description: depicts the chef's grandma's place

Key Elements:

- Spatial video
- Kitchen bench
- Grandma skybox

Main Controllers/Managers:

- SingleVideoTransition - Transitions to the next scene when the video ends

- EditedSpatialVideo(GameObject), SingleVideoTransition(script) - Transitions to the next scene when the video ends

7-Photoframe

Description: user is enveloped in starry night sky while the chef talks

Key Elements:

- Spatial video
- Kitchen bench
- Starry night skybox
- Photoframes
- XR Hand Interaction

Main Controllers/Managers:

- JointManager - define interactions when the user grab the photo frames
- ObjManager
- SingleVideoTransition - Transitions to the next scene when the video ends

8-GrandmaPlace2

Description: Scene depicting the chef's grandma's place. Continuation of the first GrandmaPlace scene

Key Elements:

- Spatial video
- Interactive sugar model that allows users to sprinkle sugar to the creme brulee in the video
- Kitchen bench
- Grandma skybox

Main Controllers/Managers:

- SugarSystemController - Manages the sugar interaction
- SpatialVideoController(GameObject), SugarVideoTransition(script) - Manages the transition between the spatial videos in the scene as well as the timing of the sugar interaction animation

9-RestaurantTable

Description: the chef presents the final product, followed by text and assets fading out

Key Elements:

- Spatial video
- Kitchen bench
- Sparkles effect with sound

- Kitchen environment
- Text

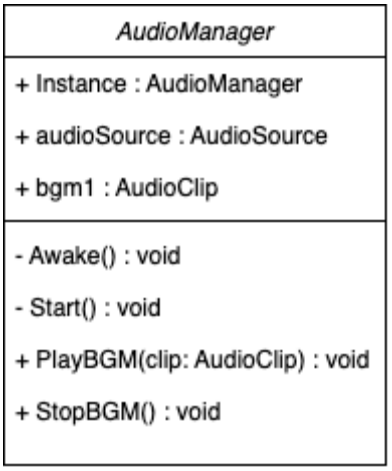
Main Controllers/Managers:

- TextAnimatorLastScene - script to control when everything fades out and the text fades in
- SingleVideoTransition - plays video

Class Overview

AudioManager

Class Function: manages the playback of audio. It ensures that only one instance of the audio manager exists throughout the game and provides methods to play and stop background music.



Methods Overview

- PlayBGM(AudioClip clip)
    - Description:plays a specific audio clip as the background music. If the audio clip passed as a parameter is different from the currently playing clip, it will set the new clip and start playing it. This ensures that the same clip does not get restarted unnecessarily.
    - Parameters:
        - clip (AudioClip): The audio clip to be played as background music.
- StopBGM()
    - Description: This method stops the currently playing background music. It is useful for situations where you want to halt the music, such as transitioning between scenes or changing game states.

DetectOrangePosition

Class Function: Manages the detection of the orange's movement and its interaction with a basket. Handles the activation of sound effects and the freezing of Rigidbody constraints when the orange is inside the basket.

DetectOrangePosition
- soundController : SoundEffectController - originalPosition : Vector3 - hasMoved : bool - rb : Rigidbody + orangeTriggered : bool
- Start() : void - Update() : void - OnTriggerStay(other: Collider) : void

Methods Overview

- Start()
    - Initializes the original position of the orange and retrieves the Rigidbody component.
- Update()
    - Checks if the orange has moved from its original position and triggers the sound effect if it has.
    - Enables gravity on the Rigidbody and sets hasMoved to true once movement is detected.
- OnTriggerStay(Collider other)
    - Checks if the orange is within a trigger collider tagged as "Basket".
    - Sets orangeTriggered to true and freezes the Rigidbody's position and rotation if a basket is detected.

DoubleVideoTransition

Class Function: Manages the transition between two videos, including video playback, collision detection, and scene transitions. Handles animation for fading in/out and manages objects and their visibility during the transition.
See Appendix Figure 2 for class diagram.

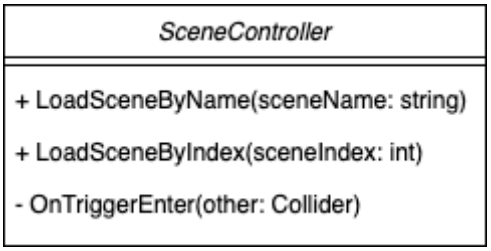
Methods Overview

- Start()
    - Initializes video durations and delays.
    - Sets up fading effects and triggers animations.

    - Starts playing the first video.
- FixedUpdate()
    - Regularly checks video playback status and collision detection.
    - Handles the transition between the first and second videos.
    - Manages the end of the second video and initiates scene transition.
- IEnumerator MandarinWait()
    - Waits for a short duration before fading out the mandarin and basket.
- CheckCollision()
    - Stops the first video and destroys objects if a collision with the basket is detected.
    - Starts playing the second video and triggers additional animations.
- GetVideoDuration(VisionOSVideoComponent video)
    - Calculates and returns the duration of a given video.
- PlayVideo(VisionOSVideoComponent video)
    - Plays the specified video with fade-in animation.
    - Sets the isPlaying flag and records the start time.
- StopVideo(VisionOSVideoComponent video)
    - Pauses the specified video with fade-out animation.
    - Resets the isPlaying flag.
- IEnumerator NextScene()
    - Waits for a short duration before loading the next scene.
- DestroyAllObjects()
    - Destroys all objects listed in objectsToDestroy and clears the list.
- VideoFadeIn()
    - Triggers video fade-in animation.
- VideoFadeOut()
    - Triggers video fade-out animation.
- FadeInTriggerObject()
    - Activates and triggers fade-in animation for the basket object.
- FadeInMandarin()
    - Activates and triggers fade-in animation for the mandarin object.
- FadeOutMandarinAndBasket()
    - Triggers fade-out animations for the mandarin and basket objects.

SceneController

Class Function: Handles scene loading based on either scene name or index and manages scene transitions triggered by specific colliders.



Methods Overview

- LoadSceneByName(string sceneName)
    - Loads a scene based on the provided scene name.
    - Validates that the scene name is not empty or null before attempting to load the scene.
- LoadSceneByIndex(int sceneIndex)
    - Loads a scene based on the provided scene index.
    - Ensures the scene index is within the valid range of scenes in the build settings.

SingleVideoTransition

Class Function: Manages the playback of a single video, handles scene transitions, and controls animations for objects in the scene. Includes functionality for fading in and out, handling skybox visibility, and managing object destruction or activation.

See Appendix Figure 3 for class diagram.

Methods Overview

- Start()
    - Initializes video playback and sets up initial conditions for skybox and object visibility based on the current scene index.
    - Plays the video and invokes methods for handling specific scene-related effects.
- Update()
    - Monitors video playback status and triggers video stopping and scene transition when the video ends.
- GetVideoDuration()
    - Calculates and returns the duration of the video based on its frame count and frame rate.
- PlayVideo()
    - Starts video playback with a fade-in effect and updates the playback status.
- IEnumerator StopVideo()
    - Handles stopping the video with a fade-out effect.
    - Manages skybox visibility and object states before transitioning to the next scene.
- DisableAllObjects()
    - Disables all objects listed in objectsToDestroy.
- FadeAllObjects()
    - Fades out and destroys all objects listed in objectsToDestroy.
- CheckSkybox()
    - Manages the fading of the skybox based on the current scene index and visibility flag.
- EnableBlackSkybox()
    - Activates the black skybox after a specified time.
- VideoFadeIn()
    - Triggers the fade-in animation for the video.

- VideoFadeOut()
    - Triggers the fade-out animation for the video.
- FadeInKitchen()
    - Triggers the fade-in animation for the kitchen object.
- FadeOutKitchen()
    - Triggers the fade-out animation for the kitchen object.

SugarVideoTransition

Class Function: Manages the playback of two sequential videos, controls skybox and object visibility, and handles animations for the sugar bowl and particles. Also handles scene transitions based on video playback and object interactions.

See Appendix Figure 4 for class diagram.

Methods Overview

- Start()
    - Initializes video playback and skybox fading based on the scene index.
    - Plays the first video and schedules the disabling of the black skybox if needed.
- FixedUpdate()
    - Monitors the video playback and transitions between the first and second videos.
    - Checks for the completion of video playback and handles the fading out of the sugar bowl.
- GetVideoDuration(VisionOSVideoComponent video)
    - Calculates and returns the duration of a given video based on its frame count and frame rate.
- CheckSugar()
    - Starts a coroutine to wait and then marks the sugar as sprinkled.
- IEnumerator SugarWait()
    - Waits for 4.5 seconds before setting the sugarSprinkled flag to true.
- PlayVideo(VisionOSVideoComponent video)
    - Starts video playback with a fade-in effect and updates playback status.
- StopVideo(VisionOSVideoComponent video)
    - Stops video playback with a fade-out effect and updates playback status.
- IEnumerator NextScene()
    - Waits for 2 seconds before loading the next scene using SceneController.
- DestroyAllObjects()
    - Disables all objects listed in objectsToDestroy.
- DisableBlackSkybox()
    - Deactivates the black skybox.
- VideoFadeIn()
    - Sets animation parameters to trigger the fade-in effect for the video.

- VideoFadeOut()
    - Sets animation parameters to trigger the fade-out effect for the video.
- IEnumerator FadeOutSugarBowl()
    - Waits for 4 seconds, then hides the sugar ball and triggers the fade-out animation for the sugar bowl.

FadeOutBell

Class Function: Handles the fading out animation of a bell object. The class waits for a specified duration before triggering the fade-out animation and then disables the bell object.

Methods Overview

- IEnumerator FadeOut()
    - Waits for a specified time (waitTillFadeOut).
    - Triggers the fade-out animation for the bell using bellAnim.
    - Waits for an additional 2.5 seconds before disabling the bell object.

ObjManagerLeftAndRight

Class Function: Manages the interaction between the VR hand gestures and objects. Handles grabbing and releasing objects with left and right hands based on finger tip distance and hand gestures.

Methods Overview

- Start()
    - Initializes the jointM if the JointManagerVisionPro instance is available.
- Update()
    - Call methods to check the distance between fingertips on both hands.
    - Manages object parenting and gravity based on hand gestures and finger closure status.
- TestFingerTipDistanceLeft()
    - Checks if the distance between the left index and thumb finger tips is less than 0.04 units.
    - Updates twoFingerClosedLeft based on the distance.
- TestFingerTipDistanceRight()
    - Checks if the distance between the right index and thumb finger tips is less than 0.04 units.
    - Updates twoFingerClosedRight based on the distance.
- TryGrabRelease(GameObject _initFingerObj, GameObject _touchedObj)
    - Releases the target object if it is currently grabbed and sets gravity back on its Rigidbody.
- TryGrabLeft(GameObject _initFingerObj, GameObject _touchedObj)

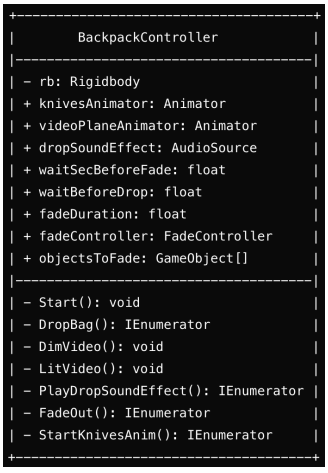
    - Attempts to grab an object with the left hand and sets targetObj, targetRigidBody, and initFingerJointObj accordingly.
- TryGrabRight(GameObject _initFingerObj, GameObject _touchedObj)
    - Attempts to grab an object with the right hand and sets targetObj, targetRigidBody, and initFingerJointObj accordingly.

BackpackController

Class Function: Manages the timing of the backpack animation for scene '4-Kitchen' as well as the lighting of the video plane during the animation

Methods Overview

- DropBag(): Starts the backpack animation
    - calls the helper functions to trigger the animations for the video plane, backpack, and other assets
    - calls DimVideo(), FadeOut(), StartKnivesAnim(), PlayDropSoundEffect()
- FadeOut(): Starts the fade out animation on the backpack and lits up the video again
    - calls LitVideo()
- DimVideo(): Triggers the video plane dimming animation
- LitVideo(): Triggers the video plane lighting up animation
- StartKnivesAnim(): Triggers the knives, book, and map falling animation
- PlayDropSoundEffect(): Plays the sound effect for the backpack drop



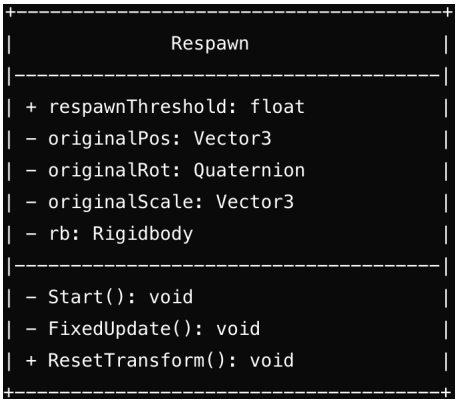
Respawn

Class Function: Respawns items that have fallen out of the scene

Methods Overview

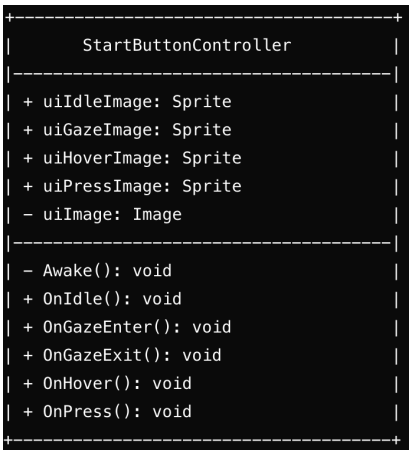
- Update():
    - Regularly checks if the gameobject's y-position is below the respawnThreshold value
- ResetTransform():

- Resets the gameobject's transforms (rotation, position, scale) and velocity to its initial state



StartButtonController

Class Function: Controls the start button's UI image upon interaction

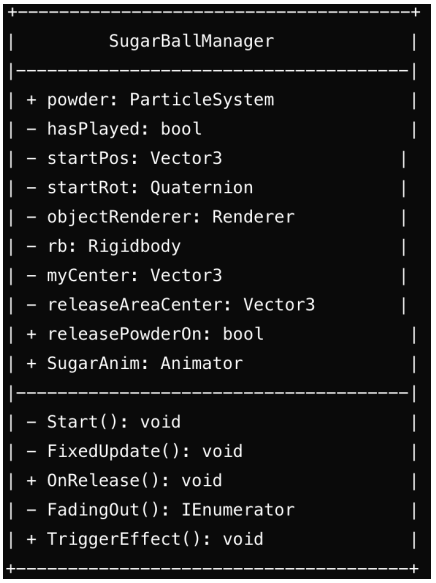


SugarBallManager

Class Function: Manages the interactable sugar ball object and falling sugar particles

Methods Overview

- FixedUpdate():
  - Regularly checks if the sprinkling has completed and destroys the gameobject on completion
- OnRelease():
  - Event handler method for release
- TriggerEffect():
  - Triggers the sprinkling effect sugar effect
  - calls FadingOut()
- FadingOut():
  - Starts the animation to fade out the sugar ball

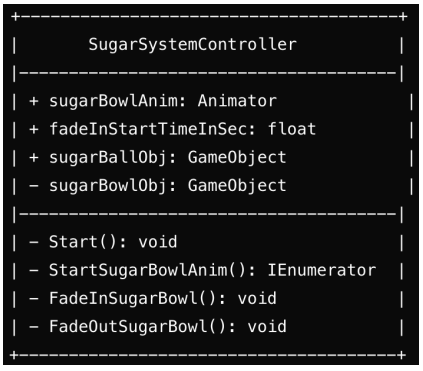


SugarSystemController

Class Function: Controls the fade in / fade out of the sugar bowl

Methods Overview

- Start():
    - calls StartSugarBowlAnim() to start the fade in animation
- StartSugarBowlAnim():
    - waits 'fadeInStartTimeInSec' seconds before fading in the sugar bowl



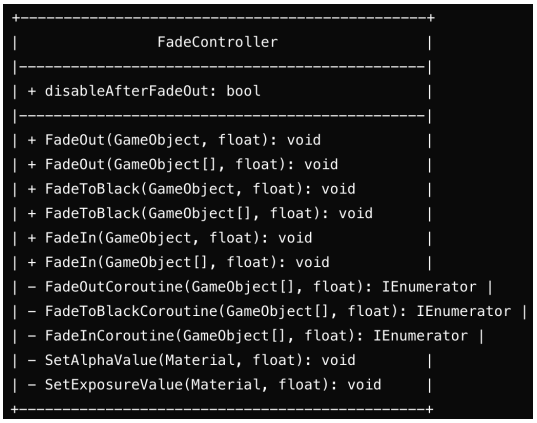
FadeController

Class Function: Controls the fade in / fade out of gameobjects with shadergraph-based materials

Methods Overview

- FadeOut(GameObject, float):
    - fades out the given gameobject for the given amount of seconds
- FadeIn():
    - fades in the given gameobject for the given amount of seconds
- FadeToBlack():

- fades the given gameobject to black for the given amount of seconds

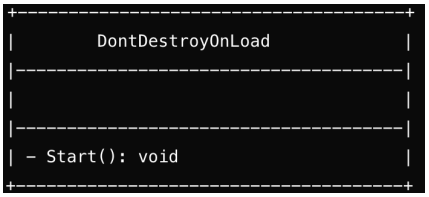


DontDestroyOnLoad

Class Function: Sets up objects to not be destroyed when loading a new scene. Attach the script to the gameobject you want to keep across scenes.

Methods Overview

- Start():
  - sets the gameobject to not be destroyed when switching scenes

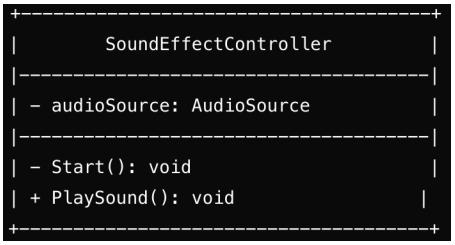


SoundEffectController

Class Function: Manages and plays sound effects using the AudioSource component to control audio playback

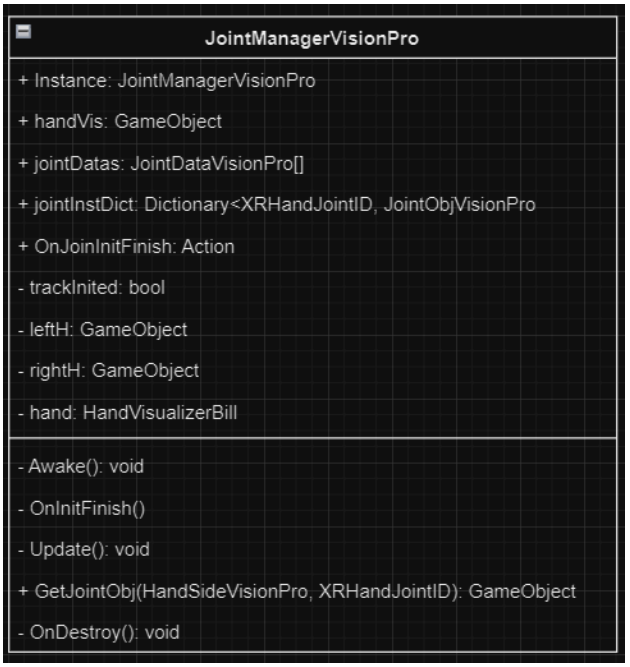
Methods Overview

- PlaySound():
  - plays the audio source in audioSource



JointManagerVisionPro

Class function: define data for different hands and interaction events



Class Function: initialize and set up hand joints

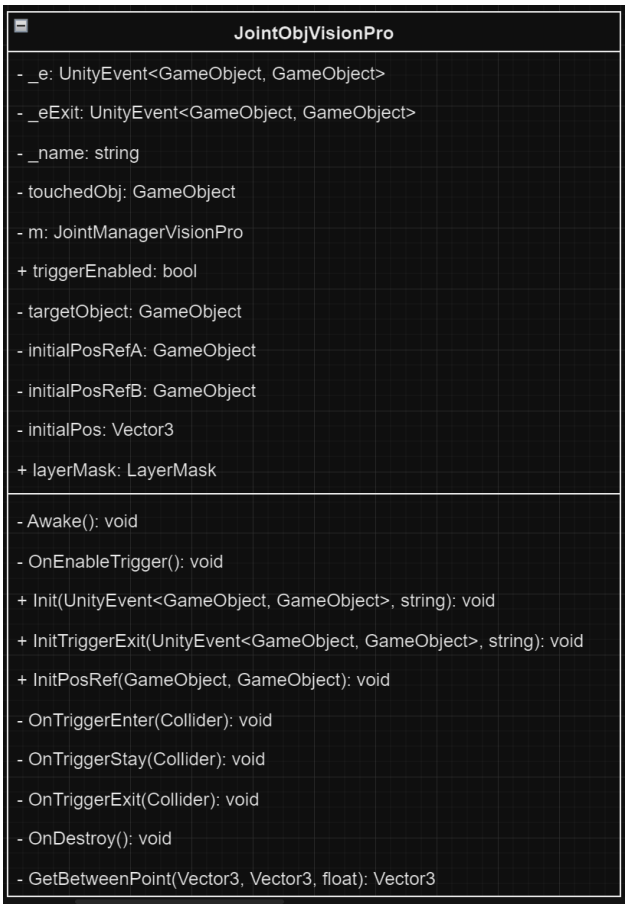
Methods Overview

- Awake():
    - Called when the script instance is being loaded.
    - initializes the Instance singleton and sets up the hand visualizer. If the hand visualizer (HandVisualizerBill) is found, it hooks into its OnInitFinish event.
- OnInitFinish():
    - This method is called when the hand visualizer initialization is finished.
    - iterates through jointDatas and instantiates associated objects at the joint positions.
    - If an object is specified, it sets up the object with the associated events (onClickEvent, onUnClickEvent).
    - assign the correct hand and index to leftH and rightH with GetJointObj()
- Update():
    - Called once per frame

    - checks if the tracking has been initialized. It checks if the left and right hand objects (leftH and rightH) are assigned and if they are sufficiently far apart, it triggers the OnJoinInitFinish event.
- GetJointObj():
    - A helper method that returns the GameObject corresponding to a specified hand side and joint index. It uses the HandVisualizerBill object to get the joint's object.
- OnDestroy():
    - This method is called when the object is destroyed.
    - unhooks the OnInitFinish event from the hand visualizer.

JointObjVisionPro

Class Function: handles interaction between a specific hand joint and other objects in a Unity scene. It manages initialization, triggers events for collisions (like entering, staying, and exiting contact), and filters interactions using a LayerMask.



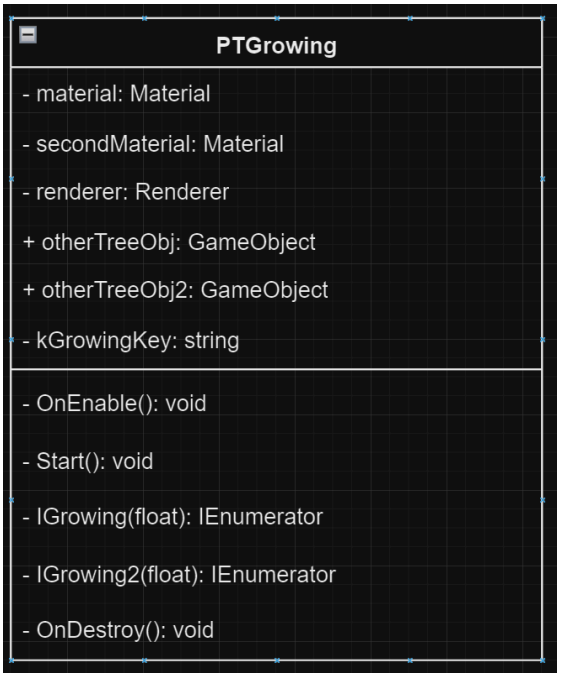
Methods Overview

- Awake():
    - is called when the script instance is being loaded.
    - sets up the JointManagerVisionPro instance reference and subscribes to the OnJoinInitFinish event. This event enables the triggers when the joint system is ready.

- OnEnableTrigger():
    - enables the triggerEnabled flag, allowing the joint object to start detecting collisions.
- Init(UnityEvent<GameObject, GameObject> e, string name):
    - Initializes the joint object with a specific interaction event (_e) and a name (_name). The name is also set to the GameObject's name for identification.
- InitTriggerExit(UnityEvent<GameObject, GameObject> e, string name):
    - Initializes the onUnClickEvent with a specific event (_eExit). The name parameter isn't used here, possibly a leftover from a similar method signature.
- InitPosRef(GameObject A, GameObject B):
    - Sets up two reference GameObjects (initialPosRefA and initialPosRefB), likely for positioning or alignment purposes.
- OnTriggerEnter(Collider other):
    - Triggered when another collider enters the joint object's collider. If triggerEnabled is true and the other object is within the specified layerMask, it invokes the _e event with the joint object and the touched object.
- OnTriggerStay(Collider other):
    - Called every frame while another collider stays within the joint object's collider. It behaves similarly to OnTriggerEnter, ensuring continuous interaction while the objects are in contact.
- OnTriggerExit(Collider other):
    - Triggered when another collider exits the joint object's collider. If the object is within the specified layerMask, it invokes the _eExit event.
- OnDestroy():
    - is called when the object is destroyed.
    - unsubscribes from the OnJoinInitFinish event to prevent potential memory leaks.
- GetBetweenPoint(Vector3 start, Vector3 end, float percent = 0.5f):
    - A utility method that calculates a point between two vectors (start and end). The percent parameter defines how far along the line between the two points the resulting point should be. This isn't used in the provided code but could be useful for positioning or movement calculations.

PTGrowing

Class function: controls a visual growth effect on a GameObject by animating shader properties over time.

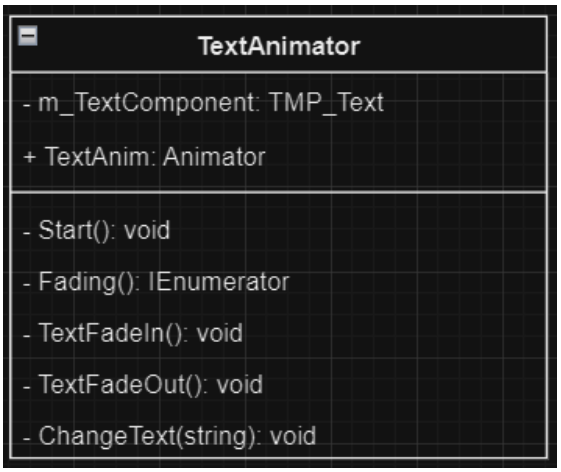


Methods Overview

- OnEnable():
  - Called when the GameObject becomes active.
  - initializes the renderer component and sets the _Growing property of the materials to their starting values.
  - If there are multiple materials, it also sets a property (_Growing2) on the second material.
- Start():
  - Called once when the script is first run.
  - starts the IGrowing coroutine, which animates the growing effect over a specified duration (6.5 seconds in this case).
- OnDestroy():
  - Called when the GameObject is destroyed.
  - ensures that any materials created during runtime are properly destroyed to avoid memory leaks.
- IGrowing(float duration):
  - Animates the _Growing property of the primary material from 0 to 1 over the specified duration. This likely corresponds to a visual growth effect in the shader. Once complete, it activates the otherTreeObj and otherTreeObj2 GameObjects.
- IGrowing2(float duration):
  - This coroutine, which is currently unused in the Start method, animates the _Growing2 property of the second material from a starting value of 2.5 to a value calculated over the duration. After the animation, it starts the IGrowing coroutine again, indicating a two-stage growing process.

TextAnimator

Class function: fade in/out text for scene 2



Methods Overview

- Start():
    - called when the script first runs.
    - Starts the Fading() coroutine
- Fading():
    - dictates when the text fades in/out using below helper functions
- TextFadeIn(): helper
- TextFadeOut(): helper
- ChangeText(string): helper

TextAnimatorLastScene

Class function: animate components for the last scene

Methods Overview

- Start():
    - called when the script first runs.
    - Call the Fading() coroutine
- Fading():
    - use the other helper functions in the diagram to fade in/out the components in the scene in a timely manner

InvertNormals

Class function: Inverts the normals of a mesh and swaps the vertex order of each triangle to ensure correct rendering with inverted normals. Make the skybox to be shown in the inside of a sphere.

Methods Overview

- Start()
    - Retrieves the MeshFilter component from the GameObject.
    - If the MeshFilter is present, it performs the following:
        - Inverts the normals of the mesh.
        - Swaps the vertex order of each triangle to maintain correct rendering.

HandVisualizerBill

Class function: Modified version that visualizes hand tracking data using prefabs for joints and hand meshes. Designed for use with VisionOS and XR hands.

Methods Overview

- OnEnable()
  - Updates rendering visibility when the component is enabled.
- OnDisable()
  - Unsubscribes from the hand subsystem and updates rendering visibility when the component is disabled.
- UnsubscribeSubsystem()
  - Unsubscribes from the hand subsystem events.
- OnDestroy()
  - Cleans up hand game objects on destruction.
- GetLeftJointsInfo() (GameObject[])
  - Returns an array of GameObjects representing the left hand joints.
- GetRightJointsInfo() (GameObject[])
  - Returns an array of GameObjects representing the right hand joints.
- Update()
  - Updates the component each frame, subscribing to the hand subsystem if necessary.
- SubscribeHandSubsystem()
  - Subscribes to the hand subsystem events.
- UnsubscribeHandSubsystem()
  - Unsubscribes from the hand subsystem events.
- UpdateRenderingVisibility(HandGameObjects handGameObjects, bool isTracked)
  - Updates the visibility of hand game objects based on tracking state.
- OnTrackingAcquired(XRHand hand)
  - Handles tracking acquired events for a hand.
- OnTrackingLost(XRHand hand)
  - Handles tracking lost events for a hand.
- OnUpdatedHands(XRHandSubsystem subsystem, XRHandSubsystem.UpdateSuccessFlags updateSuccessFlags, XRHandSubsystem.UpdateType updateType)
  - Handles updated hands events from the subsystem.

Events:

- OnInitFinish: Event invoked when initialization is finished.

Nested Class: HandGameObjects

- GetAllDrawJoints() (GameObject[])
  - Returns an array of GameObjects representing all draw joints.
- ToggleDebugDrawJoints(bool debugDrawJoints)
  - Toggles the visibility of debug draw joints.
- ToggleDrawMesh(bool drawMesh)

    - Toggles the visibility of the hand mesh.
- HandGameObjects(Handedness handedness, Transform parent, GameObject jointVisualsPrefab, GameObject meshPrefab, Material meshMaterial, GameObject emptyJointPrefab)
    - Constructor for initializing hand game objects.
- OnDestroy()
    - Cleans up joint visuals and draw joints on destruction.
- SetHandActive(bool isActive)
    - Sets the active state of the joint visuals parent.
- UpdateJoints(XRHand hand, bool areJointsTracked)
    - Updates the joint positions and visibility based on tracking data.
- UpdateJoint(XRHandJoint joint, ref Pose parentPose, ref int parentIndex, bool cacheParentPose = true)
    - Updates a single joint's position and visibility.

Testing and Results

We conducted extensive testing throughout the development process to ensure the quality and usability of our prototype. This included both internal and external testing phases.

Internally, our team members, including developers, UX designers, and narrative designers, participated in a series of user and usability tests. These tests were crucial for identifying and addressing issues early on.

In addition to internal testing, we engaged in weekly testing sessions with our client, Yangos. His detailed feedback each week was instrumental in refining our prototype. His strong opinions and persistent emphasis on certain features, coupled with his tendency to reassess based on actual implementation, led to numerous modifications. This iterative process significantly influenced the development and evolution of the prototype.

We also conducted external user tests with individuals from diverse backgrounds to ensure broad usability and gather a range of perspectives. This comprehensive testing approach allowed us to enhance the prototype based on diverse user inputs and iterative feedback.

Controls

- Gaze & Pinch
    - UI - Start button
    - Sprinkle sugar - you can sprinkle sugar using the gaze & pinch feature.
        - gaze & pinch to get sugar from sugar bowl
        - pinch (and move your hands) to sprinkle sugar on the crème brûlée.
- Hand touch
    - UI - Start button
- Poke
    - Bell - you can use your finger to press down the bell slowly.
- Grab
    - Mandarin - grab the mandarin from the tree and put it into the basket.
        - as long as your index finger and thumb are close to each other AND your hand touches(collides) with the mandarin, you can “grab” the orange.
    - Photo frames - only in the starry night scene.
        - same as mandarin

Known Bugs, Issues and Workaround

A List of any known bugs or issues that were not fixed and/or provided any temporary workarounds.

1. Mandarin Gravity and Grab Issues:

- Description: When the user grabs the mandarin from the tree and slightly releases it, even a minimal finger movement causes the mandarin to fall slowly instead of immediately. This issue persists even if the user attempts to grab the mandarin again.
- Workaround: Currently under investigation. A potential solution may involve adjusting the physics interactions or implementing a more responsive grab mechanism.

2. Basket Collider Problem:

- Description: The use of a convex mesh collider for the basket results in visual inconsistencies. The mandarin may not appear correctly positioned at the bottom of the basket and can occasionally float due to the irregular shape of the collider.
- Workaround: Consider revising the collider type or modifying its configuration to improve interaction accuracy.

3. Video Transition Delay:

- Description: In the Kitchen scene, there is an extended delay during the transition from a video fade-out to the next scene, which affects the user experience.
- Workaround: Evaluate optimization options for video loading and scene transitions. Potential improvements could include preloading assets or refining transition timings.

4. Black Frame and Plane Issues:

- Description: The black frame between videos and the black plane appearing after a video ends detracts from the user experience. A fade-to-transparent effect is not currently feasible due to technical limitations.
- Workaround: Explore alternative visual solutions or consider implementing a less obtrusive transition method until a fade-to-transparent option becomes available.

Future Considerations

- Structure and modularize the code to enhance its reusability across various scenarios for future stories. Facilitate easier maintenance and adaptability to new requirements.
- Conduct thorough research and experimentation on smooth transitions and video fading effects to optimize visual performance. Alternatively, monitor for upcoming releases and updates that may offer improved capabilities in these areas.
- Design and implement more interactions related to the story. Enhance user engagement and create a more immersive and enjoyable experience.
- Design the system to incorporate cross-platform compatibility, enabling support for a diverse range of devices, including but not limited to Quest and Vision Pro.

Contact Information

For further information or queries related to this document or the prototype, please contact:

Developers: Zimo Wu & Sally Park & Anh Vu

Email: zimoyuhu@gmail.com & imsallypark@gmail.com & vuanhkhoa22002@gmail.com

Useful Resources

- Download Xcode
- Unity's PolySpatial Tools
- VisionOS SDK Developer Tools
- Setting up play to device
- Unity PolySpatial and VisionOS
- How to export spatial video
- Display spatial video
- API for spatial videos
- Integrating 3d video to skybox and display it in Unity
- Growing Flower in Unity
- Growing Trees
- Shader Graph Basics

Appendix

The following are the class relationships between the modified hand visualizer classes.

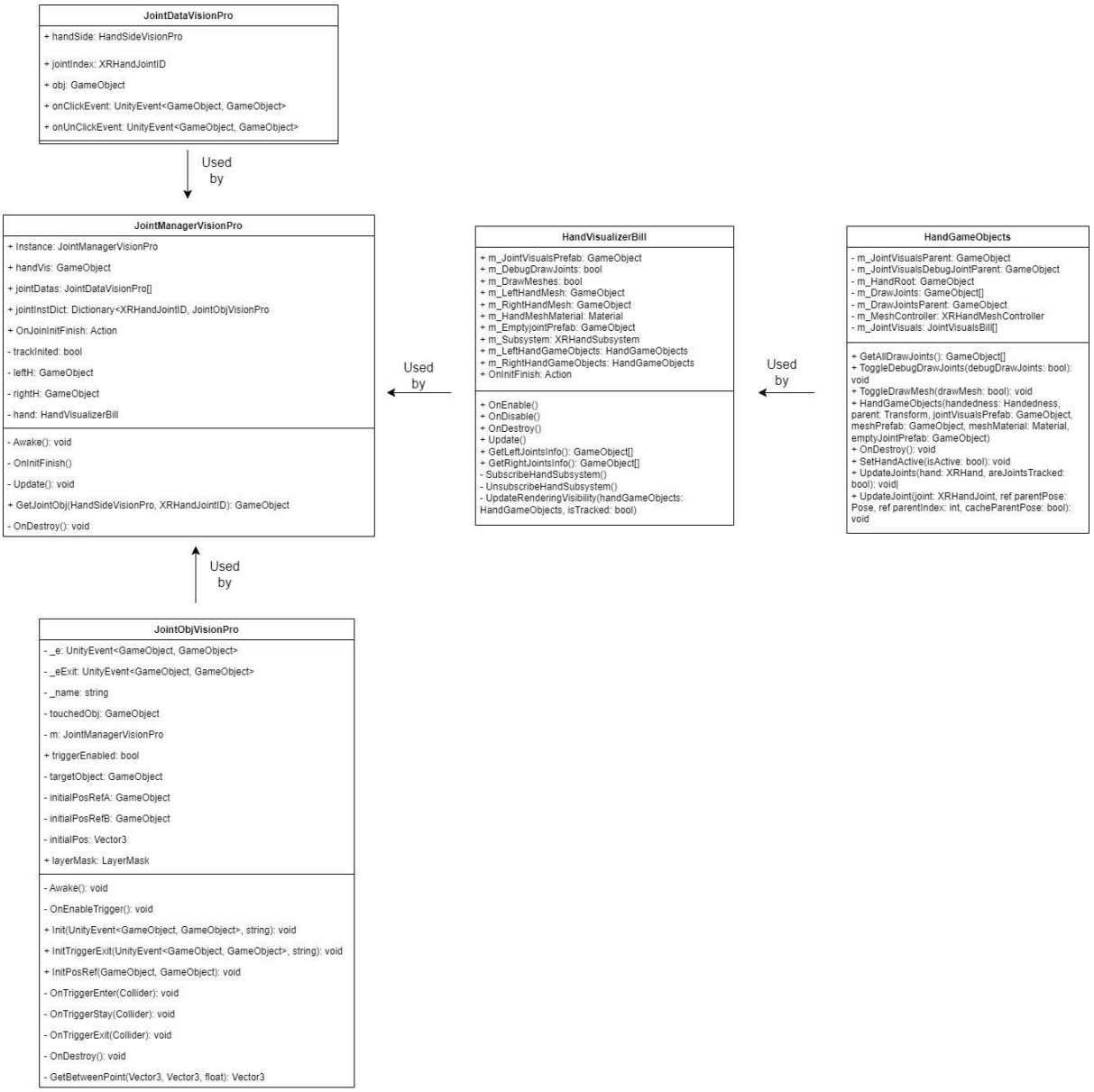


Figure 1 Dependency class diagram for hand visualizer

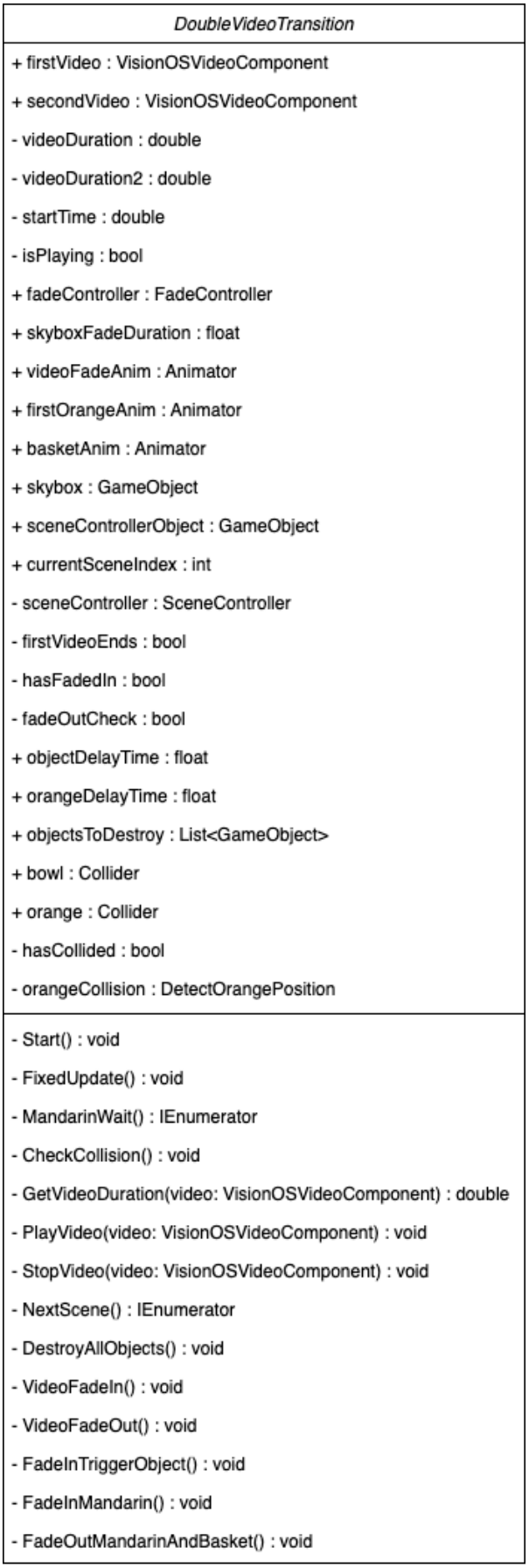


Figure 2 Class Diagram for DoubleVideoTransition

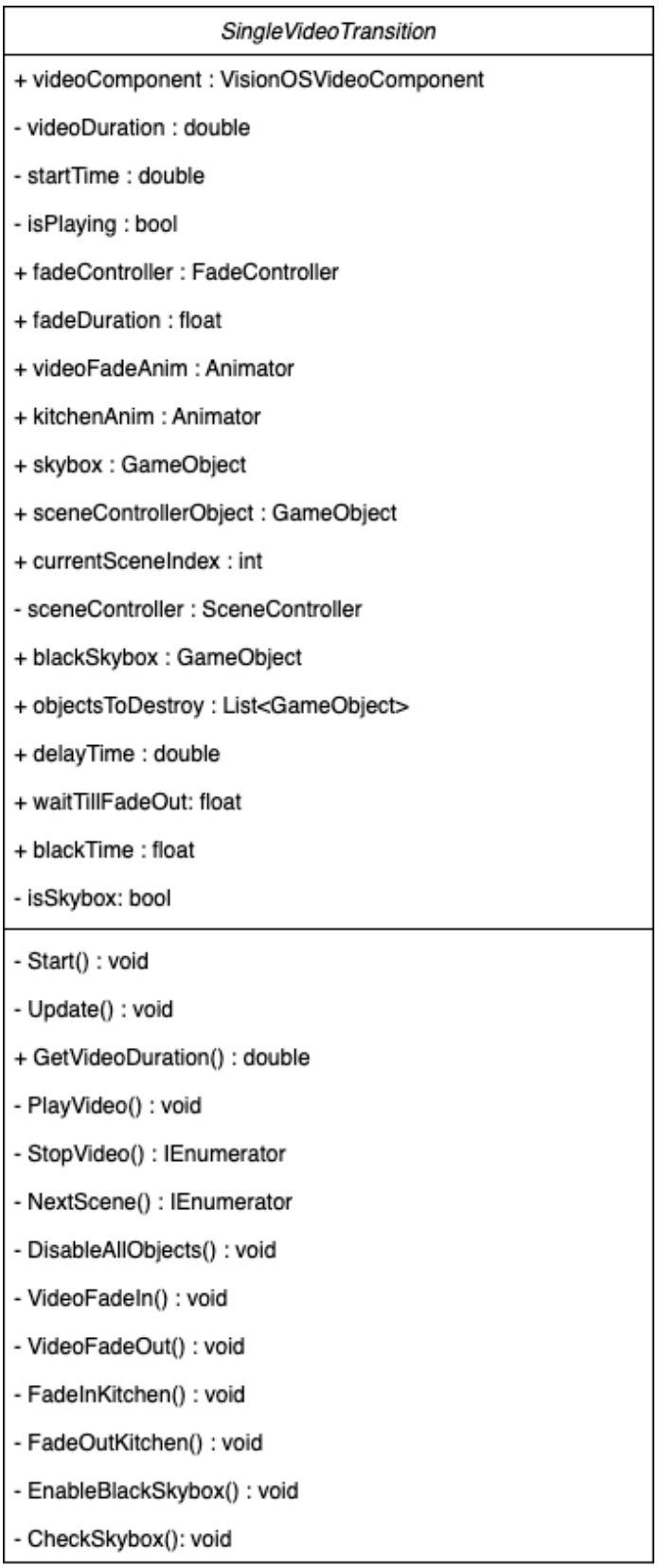


Figure 3 Class Diagram for SingleVideoTransition

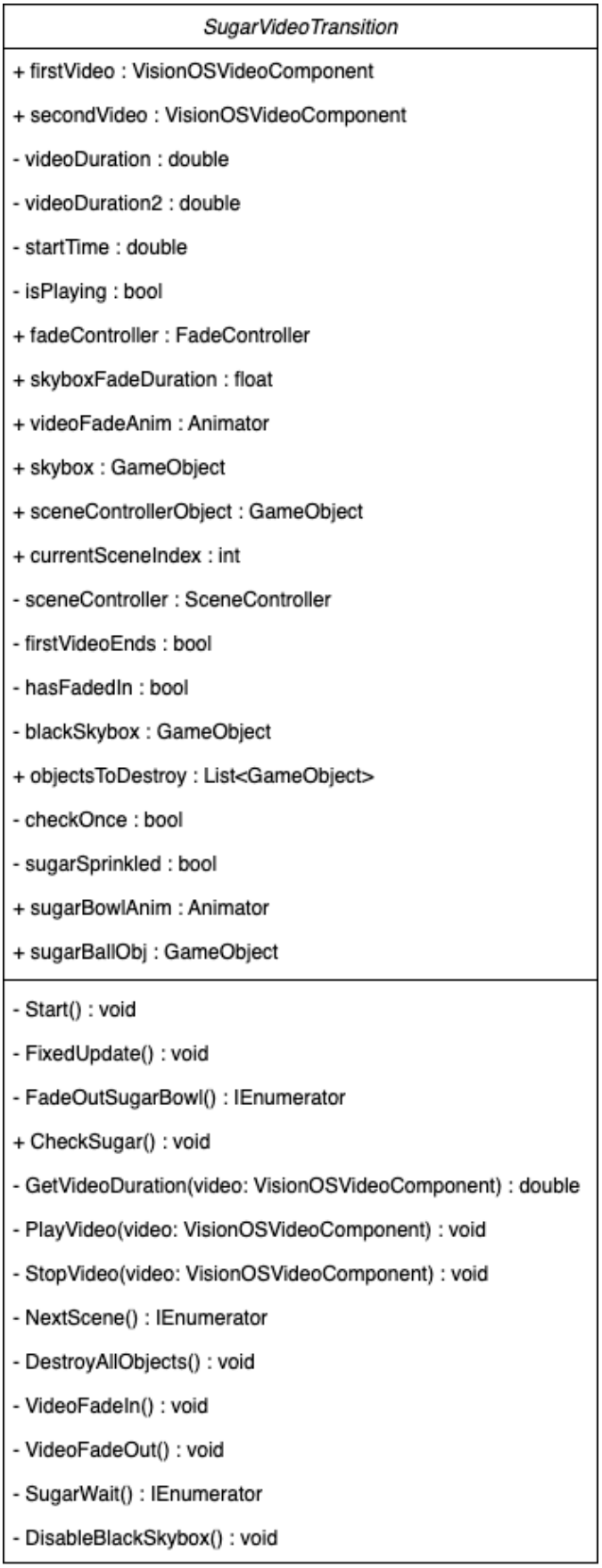


Figure 4 Class Diagram for SugarVideoTransition