Passthrough Camera API Setup Guide for Quest 3

This guide will help you set up Unity Passthrough Camera API for Meta Quest 3 devices. The Unity-PassthroughCameraAPISamples project helps Unity developers access Quest Camera data via the standard WebCamTexture and Android Camera API. You'll learn how to create mixed reality experiences that blend digital content with the real world using the Quest 3's passthrough cameras.

What You'll Learn:

- How to set up Unity for Quest 3 development
- How to configure passthrough camera access
- How to create a simple camera feed demo
- How to handle permissions and troubleshoot common issues

Requirements

Hardware Requirements

- **Meta Quest 3 or Quest 3S** running Horizon OS v74 or higher
- PC with USB-C cable for development
- Sufficient storage space (2GB+ recommended)

Software Requirements

- **Unity Version:** Unity 2022.3.58f1 or Unity 6000.0.38f1 (recommended)
- **Android SDK:** Latest version
- **Git LFS:** Required for downloading the project files

Important Permissions

Your app will need these permissions:

- `android.permission.CAMERA` - Required by WebCamTexture
- `horizonos.permission.HEADSET_CAMERA` - Custom Meta permission for passthrough camera access

Unity Project Setup

Step 1: Create New Unity Project

1. Open Unity Hub
2. Click **"New Project"**
3. **For Unity 2022:** Select **"3D (Built-in Render Pipeline)"** template
4. **For Unity 6:** Select **"3D"** template (you can switch to URP later if needed)
5. Name your project (e.g., "Quest3PassthroughDemo")
6. Choose project location and click **"Create Project"**

Step 2: Switch to Android Platform

1. Go to **File > Build Settings**
2. Select **"Android"** from the platform list
3. Click **"Switch Platform"**
4. Wait for Unity to finish switching (this may take a few minutes)

Step 3: Configure Android Settings

1. In Build Settings, click **"Player Settings"**
2. Under **"Android Settings"**:
    - Set **"Company Name"** to your preferred name
    - Set **"Product Name"** to your app name
    - Set **"Package Name"** (e.g., `com.yourcompany.quest3demo`)
    - Set **"Minimum API Level"** to **Android 10.0 (API level 29)** or higher
    - Set **"Target API Level"** to **Android 13.0 (API level 33)** or higher

Installing Meta XR SDK

Method 1: Using Package Manager (Recommended)

1. Open **Window > Package Manager**
2. Click the **"+"** button in top-left
3. Select **"Add package from git URL"**
4. For Unity 6, add: `com.unity.xr.meta-openxr@2.1.0`
5. For Unity 2022, you can use Oculus XR Plugin or Meta XR SDK

Method 2: Download Passthrough Camera API Samples

1. **Install Git LFS** first:

```
git lfs install
```

2. **Clone the repository:**

```
git clone https://github.com/GTamilSelvan07/Unity-PassthroughCameraApiSamples
```

3. **Open the project** in Unity instead of creating a new one

Required Packages

Make sure these packages are installed:

- **Meta MRUK** (com.meta.xr.mrutilitykit, v74.0.0 or higher)
- **Unity Sentis** (com.unity.sentis, v2.1.1) - for object detection features
- **XR Plugin Management**
- **OpenXR Plugin** (for Unity 6)
- **AR Foundation** (if using mixed reality features)

Project Settings Configuration

Step 1: XR Plugin Management Setup

1. Go to **Edit > Project Settings**
2. Navigate to **XR Plug-in Management**
3. **For Unity 6:**
    - In the Android tab, under Enabled Interaction Profiles, add Oculus Touch Controller Profile
    - Under OpenXR Feature Groups, enable the Meta Quest feature group
    - Check **"OpenXR"** under **"Plug-in Providers"**
4. **For Unity 2022:**
    - Check **"Oculus"** under **"Plug-in Providers"**

Step 2: Graphics API Configuration

1. In **Project Settings > Player > Android Settings**
2. Expand **"Other Settings"**
3. Under **"Graphics APIs"**:

- Meta recommends that you use the Vulkan Graphics API in your project, as some of the newer features for Meta Quest devices are only supported with that API
- Remove **"OpenGLES3"** if present
- Add **"Vulkan"** and move it to the top
- Your list should be: `Vulkan, OpenGLES3`

Step 3: Quality Settings

1. Go to **Project Settings > Quality**
2. Select the **"Android"** tab
3. Choose **"Medium"** or **"Low"** quality level for Quest 3
4. Disable **"Soft Particles"**
5. Set **"Texture Quality"** to **"Full Res"**
6. Disable unnecessary features like terrain holes and additional lights

Step 4: Universal Render Pipeline (If Using URP)

If you're using URP, you need to optimize settings:

1. Locate your project's Universal Render Pipeline Asset. One way to do this is to type t:UniversalRenderPipelineAsset into the Project window's search bar
2. In the Inspector:
   - Under the Rendering header, disable Terrain Holes
   - Under the Quality header, disable HDR
3. For Universal Renderer Data:
   - Under the Post-processing header, uncheck Enabled
   - Under the Compatibility header, set the Intermediate Texture value to Auto

---

Build Settings for Quest 3

Step 1: Android Manifest Configuration

1. Create an **"Android"** folder in **Assets/Plugins/** if it doesn't exist
2. Create **"AndroidManifest.xml"** in that folder
3. Add the required permissions:

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:tools="http://schemas.android.com/tools">
    <!-- Camera permissions -->

    <uses-permission android:name="android.permission.CAMERA" />

    <uses-permission android:name="com.horizonos.permission.HEADSET_CAMERA" />



    <!-- VR permissions -->

    <uses-feature android:name="android.hardware.vr.headtracking" android:required="true" />

    <uses-feature android:name="oculus.software.handtracking" android:required="false" />



    <application android:theme="@style/UnityThemeSelector">

        <activity android:name="com.unity3d.player.UnityPlayerActivity"

                  android:exported="true"

                  android:launchMode="singleTask"

                  android:screenOrientation="landscape"

android:configChanges="mcc|mnc|locale|touchscreen|keyboard|keyboardHidden|navigation|orie
ntation|screenLayout|uiMode|screenSize|smallestScreenSize|fontScale|layoutDirection|density"
>

            <intent-filter>

                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />

                <category android:name="com.oculus.intent.category.VR" />

            </intent-filter>
        </activity>

    </application>

</manifest>
```

Important for Unity 6: When updating the project to Unity 6, the Android Manifest will need to be updated. Android Manifest android:name="com.unity3d.player.UnityPlayerActivity" must be changed to android:name="com.unity3d.player.UnityPlayerGameActivity"

Step 2: Build Configuration

1. In **Build Settings**:
    - Ensure **"Android"** is selected
    - Set **"Texture Compression"** to **"ASTC"**
    - Enable **"Development Build"** for testing
    - Connect your Quest 3 and select it under **"Run Device"**

Step 3: Player Settings for Quest 3

1. **Configuration:** Set to **"Release"** for final builds
2. **Scripting Backend: "IL2CPP"**
3. **Target Architectures:** Check **"ARM64"** only
4. **Graphics APIs: "Vulkan"** (primary), **"OpenGLES3"** (fallback)

Passthrough Camera API Integration

Step 1: Setup Meta Project Tools

1. In Unity, go to **Meta > Tools > Project Setup Tool**
2. Click **"Fix All"** to resolve any configuration issues
3. This tool will automatically configure many Quest 3 specific settings

Step 2: Add WebCamTextureManager Prefab

1. To integrate Passthrough Camera API in your scene, drag and drop the WebCamTextureManagerPrefab prefab to your scene
2. The prefab contains:
    - **WebCamTextureManager** script for camera initialization
    - **PassthroughCameraPermissions** script for permission handling

Step 3: Configure Camera Settings

In the **WebCamTextureManager** component:

- **Camera Eye:** Choose **"Left"** or **"Right"** camera
- **Resolution:** Leave empty for highest resolution, or set to specific resolution like `1280x960`
- **Auto Start:** Check this to automatically start camera on scene load

---

Creating Your First Demo

Step 1: Setup Scene

1. Create a new scene: **File > New Scene**
2. Delete the default **"Main Camera"**
3. Add XR Rig:
    - **Unity 6:** Go to **GameObject > XR > XR Origin**
    - **Unity 2022:** Add **"OVR Camera Rig"** or **"XR Origin"**

Step 2: Configure XR Origin

1. Select the **XR Origin** GameObject
2. Expand to find the **"Main Camera"**
3. Set the Main Camera's:
    - **Clear Flags: "Solid Color"**
    - **Background Color:** Set alpha to **0** (transparent for passthrough)

Step 3: Add Camera Display

1. Create a **Canvas**: **GameObject > UI > Canvas**
2. Set Canvas **"Render Mode"** to **"World Space"**
3. Position the canvas in front of the user (e.g., position `0, 0, 2`)
4. Scale it appropriately (e.g., scale `0.01, 0.01, 0.01`)
5. Add a **Raw Image** as child of Canvas
6. This will display the camera feed

Step 4: Add the Passthrough Camera System

1. From the Passthrough Camera API samples, drag **"WebCamTextureManagerPrefab"** into your scene
2. The prefab will handle permissions and camera initialization automatically

---

Code Examples

Basic Camera Feed Display

Create a script to display the camera feed on a UI element:

```
using UnityEngine;

using UnityEngine.UI;

public class CameraViewer : MonoBehaviour

{

    [SerializeField] private RawImage displayImage;

    [SerializeField] private WebCamTextureManager webCamManager;


    void Start()

    {

        // Find the WebCamTextureManager if not assigned

        if (webCamManager == null)

        {

            webCamManager = FindObjectOfType<WebCamTextureManager>();

        }

    }


    void Update()

    {

        // Check if camera texture is available
        if (webCamManager != null && webCamManager.WebCamTexture != null)
        {
            // Assign the camera texture to the display
            displayImage.texture = webCamManager.WebCamTexture;
        }
    }
}
```

Camera to World Conversion Example

Convert 2D camera coordinates to 3D world positions:

```
using UnityEngine;

public class CameraToWorldExample : MonoBehaviour
{
    private void DetectObjectAt2DPoint(Vector2Int screenPoint)
    {
        // Convert 2D screen point to 3D ray in world space
        var ray = PassthroughCameraUtils.ScreenPointToRayInWorld(
            PassthroughCameraEye.Left, screenPoint);

        // Use environment raycast to find intersection with real world
        var environmentRaycastManager = FindObjectOfType<EnvironmentRaycastManager>();

        if (environmentRaycastManager != null &&
            environmentRaycastManager.Raycast(ray, out EnvironmentRaycastHit hitInfo))
        {
            // Place a virtual object at the hit point
            PlaceObjectAt(hitInfo.point, hitInfo.normal);
        }
    }

    private void PlaceObjectAt(Vector3 position, Vector3 normal)
    {
        // Create a simple cube at the detected position
        GameObject marker = GameObject.CreatePrimitive(PrimitiveType.Cube);
        marker.transform.position = position;
        marker.transform.rotation = Quaternion.LookRotation(normal, Vector3.up);
        marker.transform.localScale = Vector3.one * 0.1f;
    }
}
```

Permission Handling Example

```
using UnityEngine;
public class CustomPermissionHandler : MonoBehaviour
{
    void Start()
    {
        RequestPermissions();
    }


    private void RequestPermissions()

    {

        // Check if camera permission is granted

        if (!UnityEngine.Android.Permission.HasUserAuthorizedPermission(

            UnityEngine.Android.Permission.Camera))

        {

            UnityEngine.Android.Permission.RequestUserPermission(

                UnityEngine.Android.Permission.Camera);

        }


        // Check if headset camera permission is granted

        if (!UnityEngine.Android.Permission.HasUserAuthorizedPermission(

            "com.horizonos.permission.HEADSET_CAMERA"))

        {

            UnityEngine.Android.Permission.RequestUserPermission(

                "com.horizonos.permission.HEADSET_CAMERA");

        }

    }

}
```

Troubleshooting Guide

Common Issues and Solutions

Issue 1: App Won't Launch on Quest 3

Solution:

- Verify that you are testing on supported hardware (Quest 3/3S running Horizon OS v74 or higher)
- Check that **"Developer Mode"** is enabled on your Quest 3
- Ensure USB debugging is authorized

Issue 2: Camera Permission Denied

Solution:

- Currently, if users click on Don't Allow for all permissions, they are unable to access the app even after changing Settings in the device. The only solution right now is to uninstall and re-install the app
- Alternatively, grant permissions via ADB:

```
adb shell pm grant {PACKAGE.NAME} com.horizonos.permission.HEADSET_CAMERA

adb shell pm grant {PACKAGE.NAME} android.permission.CAMERA
```

Issue 3: Black Camera Feed

Solution:

- Check that both android.permission.CAMERA and horizonos.permission.HEADSET_CAMERA are granted
- Verify the **WebCamTextureManager** is properly configured
- v74 currently needs to one frame delay before WebCamTexture.Play()

Issue 4: Unity 6 Upgrade Issues

Solution:

- When updating the project to Unity 6, the Android Manifest will need to be updated
- Use **Meta > Tools > Update AndroidManiest.xml** or manually update the manifest
- The horizonos.permission.HEADSET_CAMERA permission has to be added back into the Manifest manually after updating

Issue 5: Poor Performance

Solution:

- Processing high-resolution camera feeds and running on-device ML/CV models can impact performance, particularly on the main thread
- Use lower camera resolution (e.g., `640x480` instead of `1280x960`)
- Disable HDR in URP settings
- Process camera data asynchronously when possible

Issue 6: OpenXR vs Oculus XR Plugin Conflicts

Solution:

- XR Plug-in Management supports only one enabled plug-in provider at a time per build target
- For Unity 6: Use **OpenXR** plugin
- For Unity 2022: Use **Oculus XR** plugin
- Don't enable both simultaneously

Debug Steps

1. **Check Unity Console:** Look for error messages and warnings
2. **Check Device Logs:**

   ```
   adb logcat >> log.txt
   ```

3. **Verify Project Settings:** Use **Meta > Tools > Project Setup Tool**
4. **Test with Sample Scene:** Try running the provided sample scenes first
5. **Check Package Versions:** Ensure you're using compatible package versions

Performance Optimization Tips

1. **Use Appropriate Resolution:** Each camera supports resolutions of 320 x 240, 640 x 480, 800 x 600 and 1280 x 960
2. **Optimize Graphics Settings:** Disable unnecessary post-processing effects
3. **Handle Permissions Properly:** Ensure that all Android permission requests are managed from a single location
4. **Resource Management:** Ensure that resources such as the WebCamTexture are properly stopped and disposed of when no longer needed

Getting Help

If you continue to experience issues:

- Check the [Meta Developers Documentation]
- Post on Unity Forums or Meta Developer Community
- Include your Unity version, Quest device model, and Horizon OS version in support requests

Conclusion

You now have a complete setup for Unity Passthrough Camera API development on Quest 3! This setup enables you to:

- Access Quest 3 camera feeds in Unity
- Create mixed reality experiences
- Handle permissions properly
- Convert 2D camera coordinates to 3D world positions

Next Steps:

- Experiment with the provided sample scenes
- Try building object detection or tracking features
- Explore Unity Sentis for on-device machine learning
- Create your own mixed reality applications

Remember to test frequently on the actual Quest 3 device, as Meta XR Simulator and Meta Link app are currently not supported for passthrough camera features.