UNITY XR ASSESSMENT

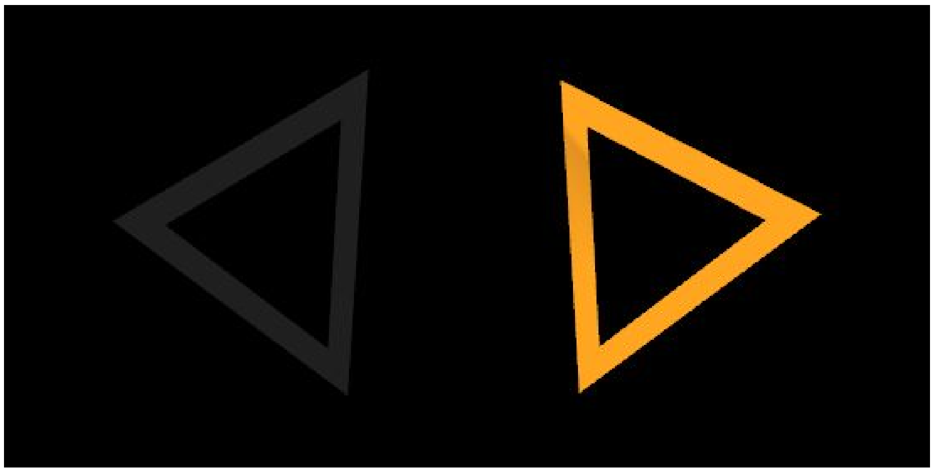
Usage

This project is used to test Unity's XR features. Tests in this project are designed to be simple and platform agnostic. The behavior of specific features can be targeted by isolating the systems under test, therefore the scenes have been constructed to limit interaction between different systems and components. Furthermore, display and input systems have been implemented in a generic way so that the project does not rely on the platform specific features of plugins. The project should be used to verify behavior of player configurations, compare features between Unity releases, and test for changes in performance.

Test Scenes

Scene Navigation

Gaze based controls have been implemented for scene navigation to allow the project to run on a variety of platforms without platform specific input dependencies. Gaze at the gray arrows in any scene to transition to the next or previous scene. The arrows will progressively fill for several seconds as long as gaze focus is maintained. The scene will transition once the arrow is completely filled.



Configurations

These tests are designed to be run in the built-in renderer with multi-pass, single-pass, and single-pass instanced stereo rendering modes. Scenes should be tested to make sure they are displayed the same in the left eye, right eye, Editor GameView, and the player's standalone mirror (if applicable). All combinations of settings supported by the target platform should be tested:

Stereo Rendering Modes
Multi Pass
Single Pass
Single Pass Instanced

Graphics APIs
Direct3D11
Direct3D12 (Experimental)
OpenGL
Vulkan
OpenES2
OpenES3
Metal

Players
Editor
Standalone

Building

Tests can be run manually in the editor or in a built player. Batch building functionality has been included in this project to provide an easy-to-use script and build configuration system that can build a common set of configurations for testing. In addition, this system allows for multiple build configurations to be built at once to reduce overhead of managing multiple test build configurations. Finally, utilization of the batch builds will maintain a consistent set of build settings across test passes to reduce configuration errors.

Usage

The build scripts located in the root of this project are available to build the most common configurations of the project for testing across supported platforms. Scripts are available for Windows (.bat) and macOS (.command) in the *BuildScripts* folder. A Build Windows UI is available in the Editor to run batch builds and allow customization of the build configurations

Pass the location of the Unity executable to run from the command line:

Windows Example:
BuildWindows.bat *<location of Unity.exe>*

MacOS Example:
BuildmacOS.command *<location of Unity.app>/Content/Unity...*

Build Location

The builds will be located in:

<Project Directory>\Builds\<Build Target>\<VR SDK>\<Stereo Rendering Method>-<Graphics API>

For example,
...\Builds\StandaloneWindows64\Oculus\Instancing-Direct3D11.exe
...\Builds\StandaloneWindows64\Oculus\MultiPass-Direct3D11.exe

Materials

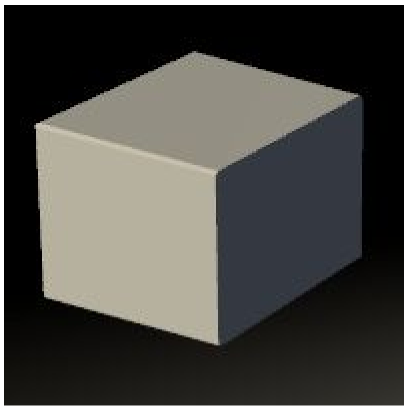
Purpose: These tests target the Standard Surface shader, Render Textures, render buffers, and other common material setups.

Test Setup: Each rotating cube is an individual test identified by the label in front of each cube.

Test Method: Compare the materials to the images and descriptions below.

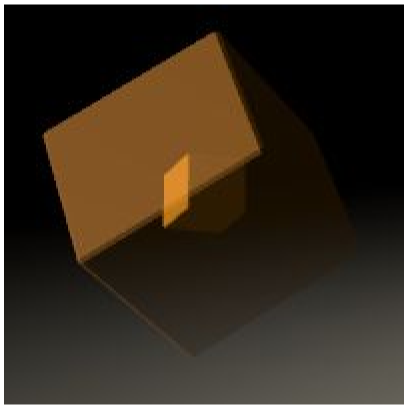
Standard

Expected Result: The cube's surface should be gray and specular highlights should be apparent as the object rotates.



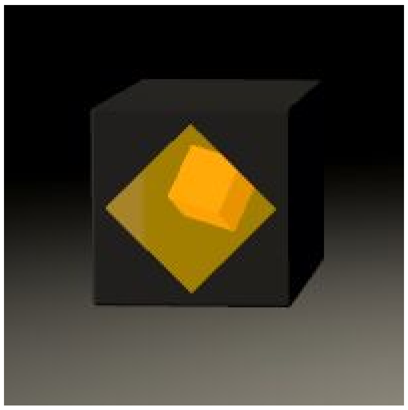
Transparent

Expected Result: A small opaque orange cube should be visible inside of a larger transparent orange cube.



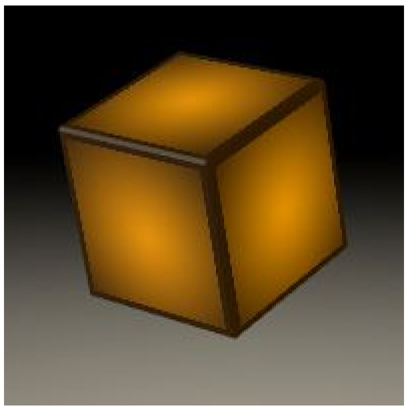
Stencil

Expected Result: An opaque dark gray cube should have a square window facing the user. The inside of the cube should appear to be orange. A second smaller opaque orange cube should be visible through the window inside of the dark gray cube.



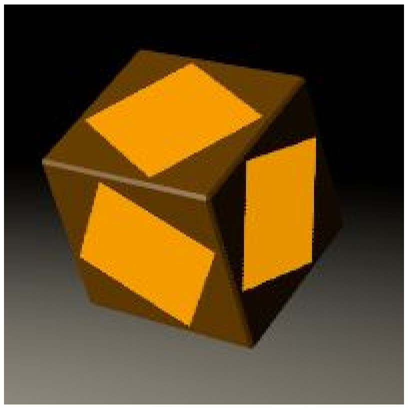
Set Target Buffers

Expected Result: Each face should have a thin uniform border while the interior should contain a gradient from orange at the center to black near the edges. The intensity of the interior faces should pulsate - starting completely black and transitioning to an orange gradient.



Render Texture

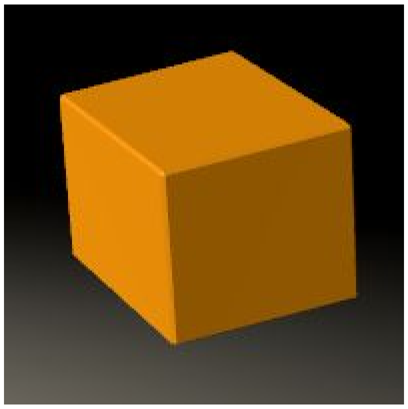
Expected Result: An orange square should rotate clockwise on each face of a cube.



GPU Instancing

Expected Result: A large orange cube should be visible in both eyes.

Common Defects: The object is only visible in one eye.

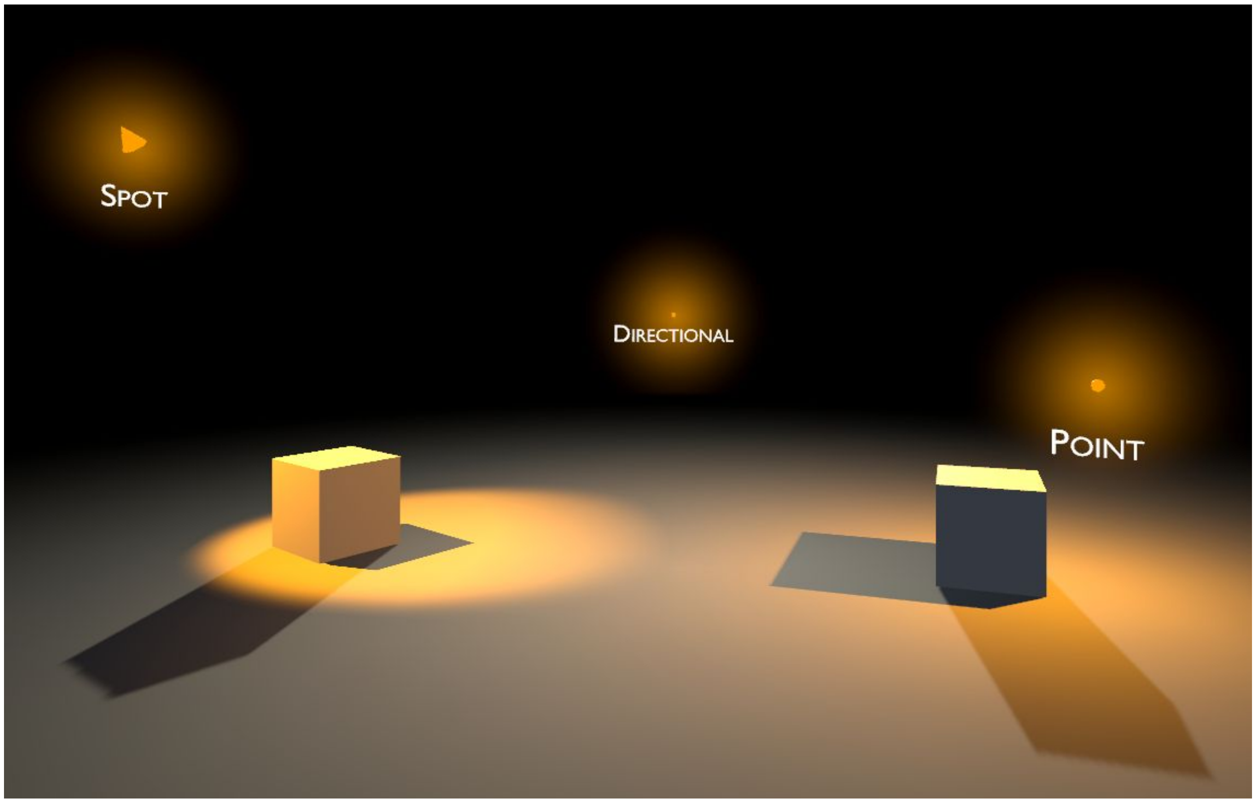


Baked Lighting

Purpose: Lightmaps in this scene are generated from the progressive lightmapper. Three light types are tested: spot, directional, and point lights.

Test Setup: All objects and lights in this scene are static. The location of light sources is indicated by orange objects surrounded by diffuse halos. The type of light can be identified by the label near each light.

Test Method: Ensure that the lighting matches the image below. Compare the directionality of shadows, saturation of light of surfaces, hue of light on surfaces, and softness of shadow edges. Shadows and lighting should appear the same in the left eye, right eye, Editor GameView, and the player's standalone mirror (if applicable). This test should be run in the Editor and Player.



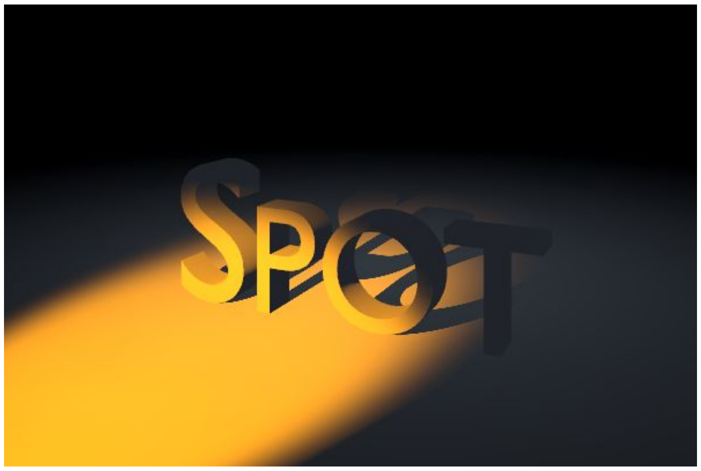
Real-time Lighting

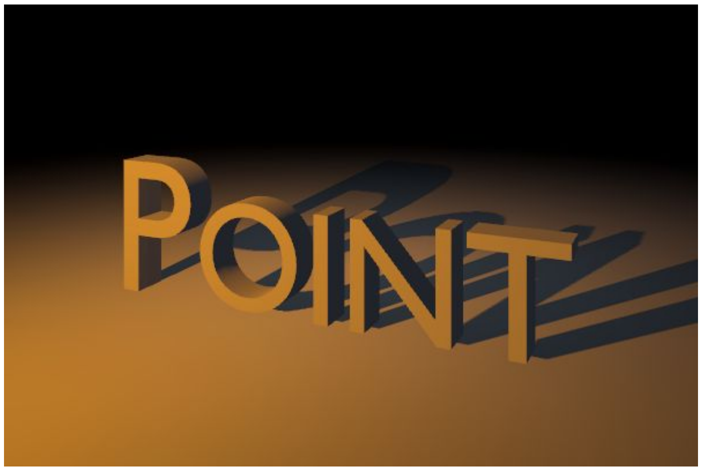
Purpose: Shadows and lighting from three realtime light types are tested in this scene.

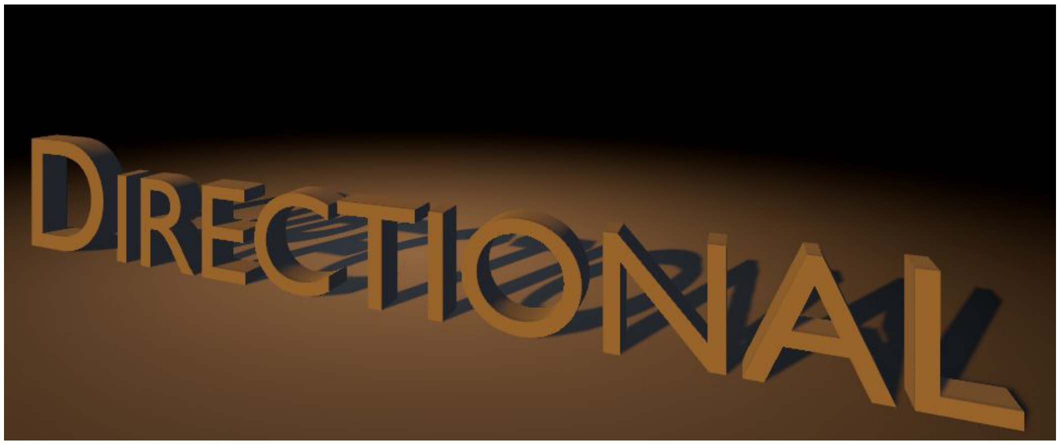
Test Setup: Directional, point, and spot lights are cycled every few seconds. A mesh indicating the current type of light is placed in the center of the scene. The lights orbit the mesh to allow the shadows to be observed from different angles.

Test Method: Verify that the shadows and lighting in the scene is correct. Pay close attention to the softness of the shadow edges and the location where the shadows intersect the text geometry.

Common Defects: Shadows are only visible in one eye. Shadows appear stretched and no longer match their shadow casting source.







Effects

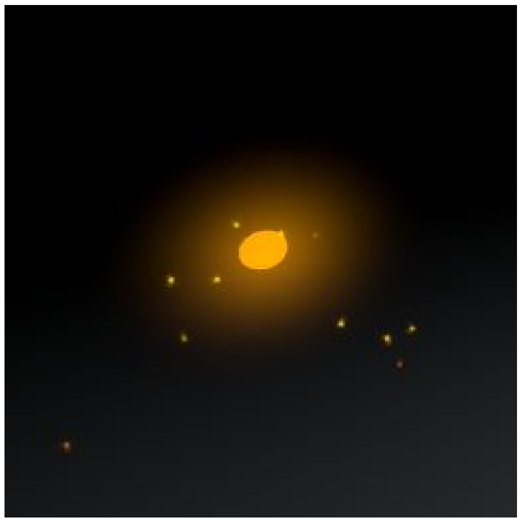
Purpose: This scene tests various effect components.

Test Setup: The five effects in this scene can be identified by the text labels.

Test Method: Compare the effects to the images and descriptions below.

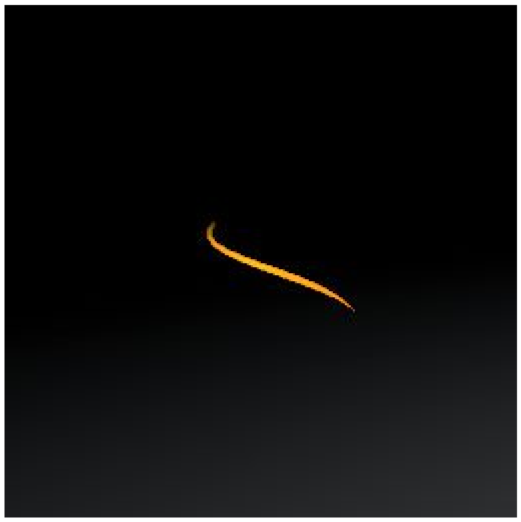
Particles

Expected Result: Particles are emitted from the orange sphere. The particles should appear to be affected by gravity and transition from orange to red before fading out. The particles should bounce when colliding with the ground plane.



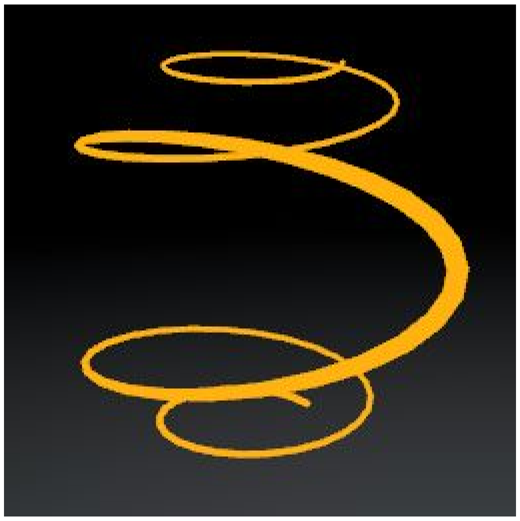
Trail Renderer

Expected Result: A single trail should slowly spiral around the Y axis. The width should change along the trail's length, becoming narrower at the ends. The alpha of the trail should fade out along the following end.



Line Renderer

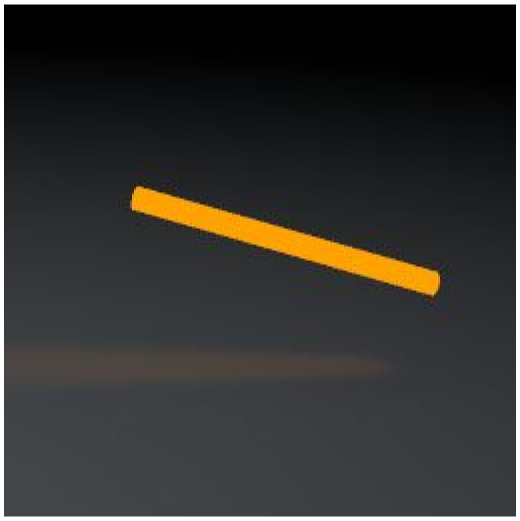
Expected Result: The length of the spiral and number of windings should change over time. The entire spiral should rotate over time. The ends of the spiral should be capped and appear smooth.



Projector

Expected Result: The cylindrical bar should appear to light the ground plane as it moves. The lighting on the ground should be rectangular with the same alignment as the cylindrical bar.

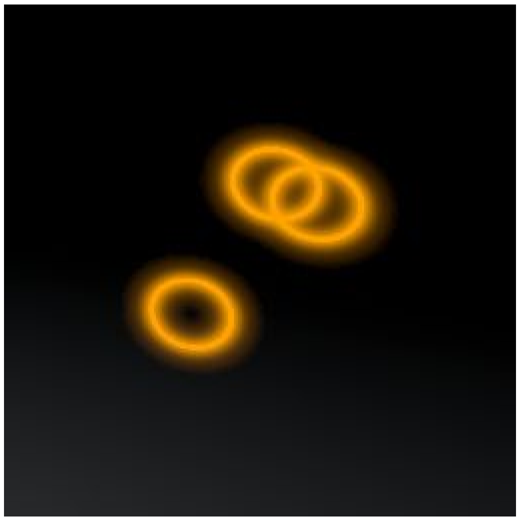
Common Defect: Projected texture on the ground is only visible in one eye.



Sprites

Expected Result: The rings should always face the user and should be visible through the interior of the rings when they overlapping.

Common Defects: Sprites are visible in only one eye and/or flickers.

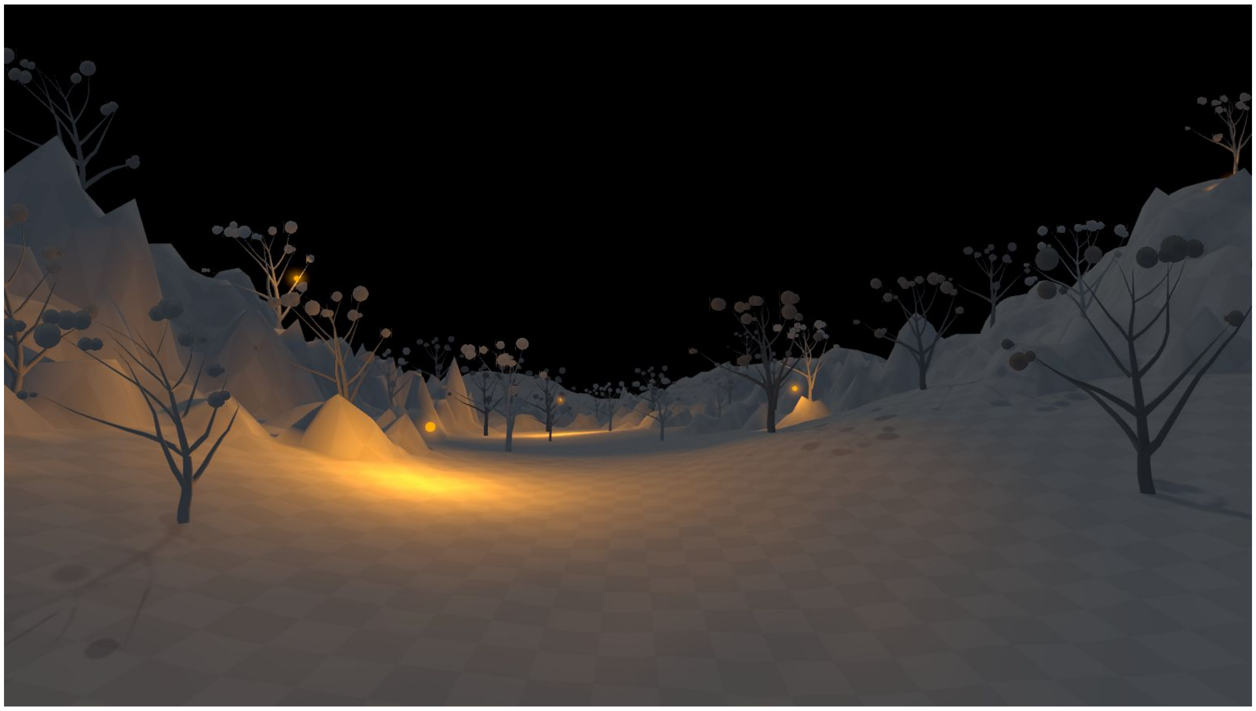


Terrain

Purpose: This test targets assets created with the terrain creation toolset and several associated subsystems.

Test Setup: A terrain asset with several Tree Creator assets and point lights are used to verify behavior. One directional shadow-casting light has been placed in the scene. A wind zone is used to affect the trees.

Test Method: Verify that the lighting on the terrain is correct by observing the light cast by the orange spheres. The trees near the camera should cast a single directional shadow and appear to sway in the wind. More distant trees are drawn as billboards and should appear static but have correct lighting.

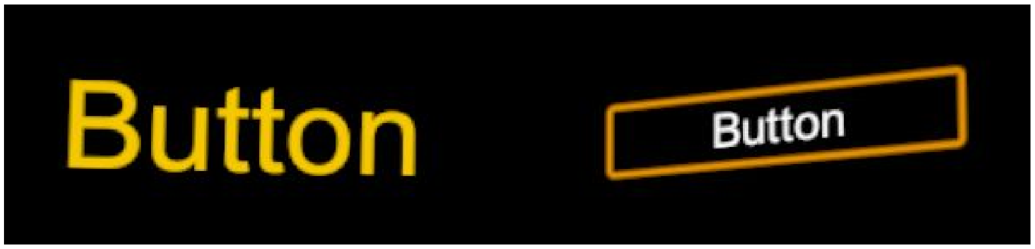


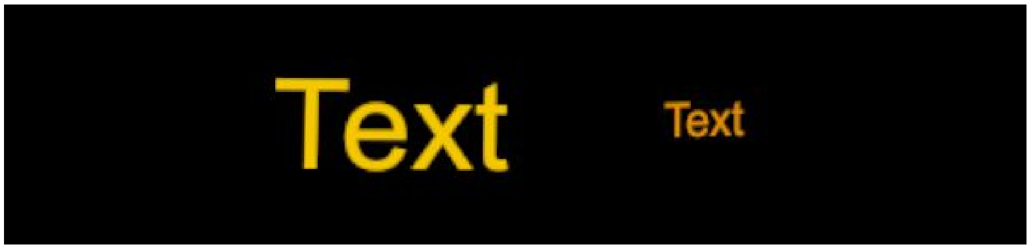
Canvas

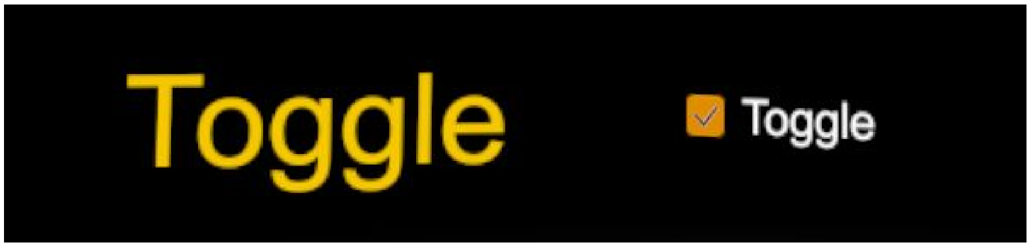
Purpose: User interface interaction is tested with the Canvas component and the associated event trigger callbacks. Button, text, toggle, and slider UI components are tested for correct rendering and event trigger behavior.

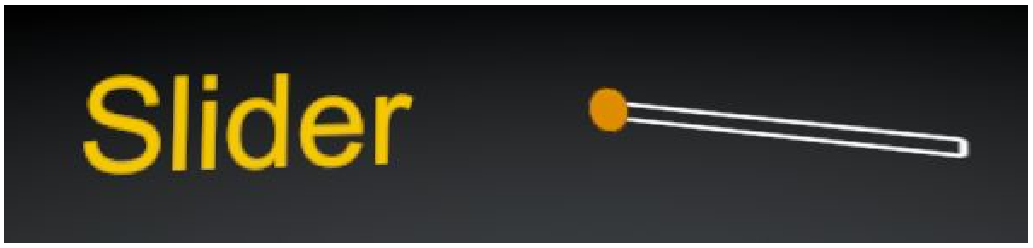
Test Setup: Button, text, toggle, and slider UI components in the scene have event triggers assigned that are called when the user's gaze enters or exits a UI element. The handler for the event triggers controls the color of the TextMesh objects in the scene.

Test Method: Gazing at the UI object on the right side should cause the UI object to become highlighted in orange while the TextMesh object to the left of the UI object should transition from gray to orange. Event triggers have been implemented for enter and exit events, so it is necessary to check that the UI object and TextMesh return to gray when they no longer have the user's gaze focus. A reticle should exist in the scene that remains at the center of the screen and follows the user's gaze. This reticle should be gray and become larger and orange when the user's gaze intersects a UI object.









API Check

Purpose: XRDevice contains details about the state of the active XR device, such as a headset, phone, or other device. XRSettings allows access to the global XR settings used for the player and editor.

Test Setup: Properties of XRDevice and XRSettings are queried at runtime to verify correct functionality.

Test Method: Read the text display and verify that all settings match the expected output for the device.

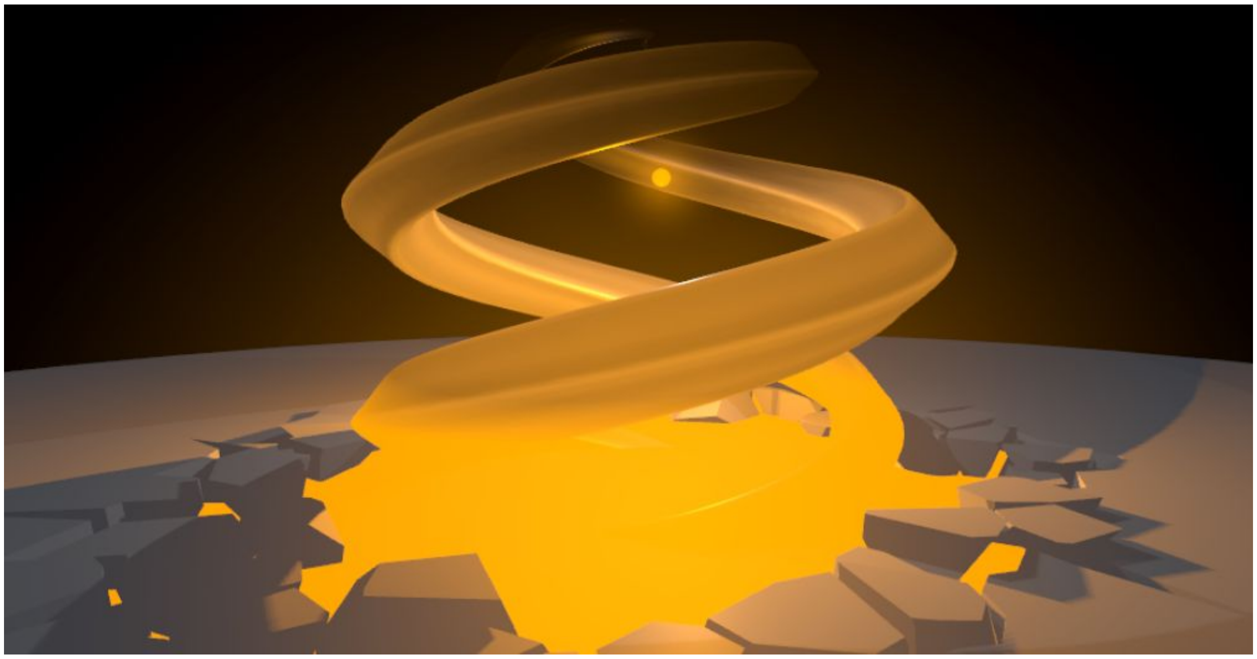
XRDevice.isPresent: True as long as a device is connected and in a good state.
XRDevice.model: Matches the connected device model.
XRDevice.refreshRate: The expected refresh rate of the device, note that this will be 0 if the device does not report refresh rate.
XRDevice.userPresence: Indicates whether the user is interacting with the device. This is hardware specific and some devices may not track user presence.
XRDevice.fovZoomFactor: A scaling factor applied to the field of view and should be 1 for this test.
XRSettings.eyeTextureHeight: Identical to the device's resolution height.
XRSettings.eyeTextureWidth: Identical to the device's resolution width.
XRSettings.eyeTextureResolutionScale: A scaling factor applied to the eye texture resolution and should be 1 for this test.
XRSettings.isDeviceActive: True if the device is connected and receiving input.
XRSettings.loadedDeviceName: This should match the name of the device.
XRSettings.occlusionMaskScale: A scale applied to the occlusion mesh. This should be 1.
XRSettings.renderViewportScale: A scale applied to the viewport. This should be 1.
XRSettings.supportedDevices: This should contain a list of all SDKs that were included in this build.

Performance

Purpose: XRStats tracks real-time performance statistics for the editor and player. These statistics are useful for performance profiling.

Test Setup: The scene contains two shadow casting points lights, 280000 triangles, 42 dynamic objects, and two sprites. A text display to the left of the spirals contains the output of XRStats as well as an average frame rate.

Test Method: Compare the XRStats and frame rate against previous version of Unity or other XR configurations to identify changes in performance. The two orange spirals should rotate along the Y axis. The rocks near the edge of the orange pit should slowly oscillate. Two shadows should be visible: one cast by the orange sphere and another cast by the orange pit.



Input

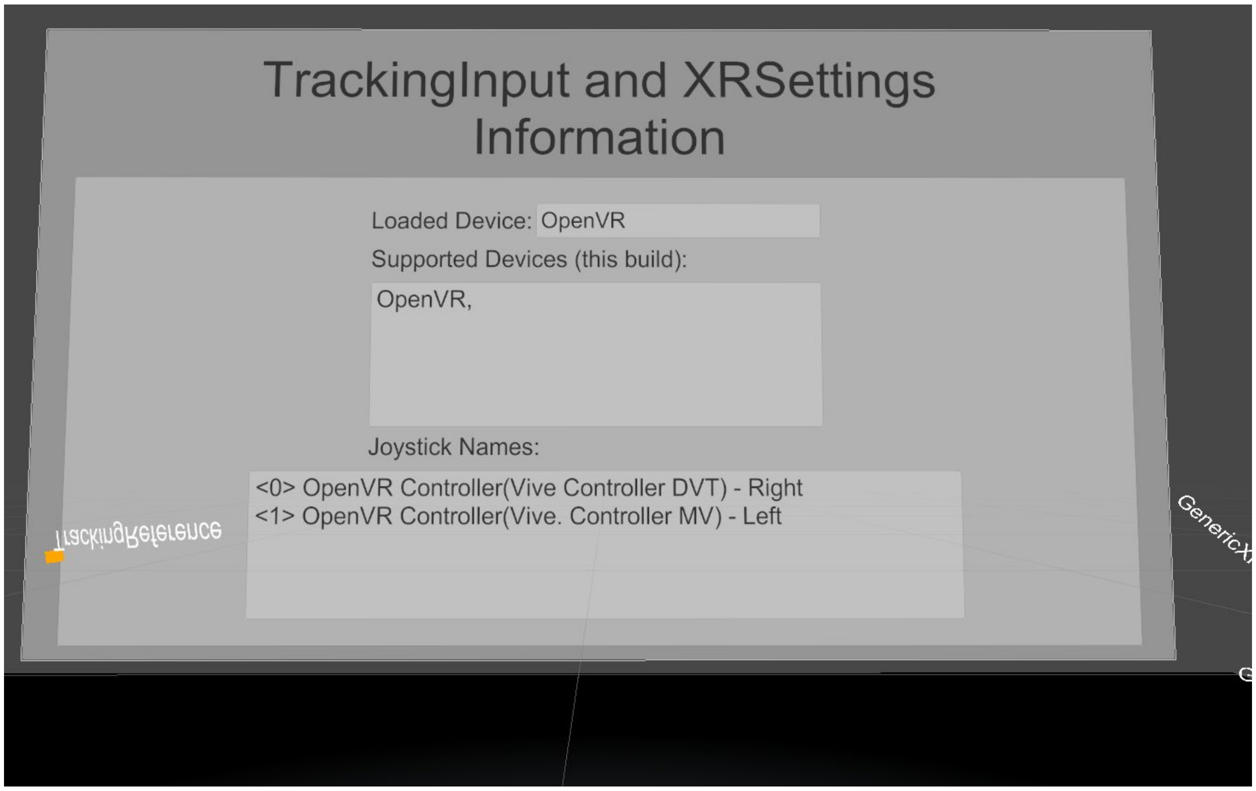
Purpose: The XR system provides access to device details through InputTracking, XRSettings, XRNodes, and the TrackedPoseDriver.

Test Setup: Input and device information associated with input testing is verified by querying the XR system for device specifics, testing generic button an axis functionality, visualizing XRNode states, and using the TrackedPoseDriver. The tests are grouped into four sections.

Test Method: Perform tests by following the instruction in the four following sections.

Input Tracking and XR Settings

XR SDK and joystick information is displayed immediately forward in the scene. Verify that the Loaded Device field shows the correct XR SDK. The Supported Devices list should contain all XR SDKs included in this build. The Joystick Names list should contain all currently attached joysticks, including XR and non-XR joysticks, gamepads, and controllers that are connected.



TrackingInput and XRSettings Information

Loaded Device: OpenVR

Supported Devices (this build):

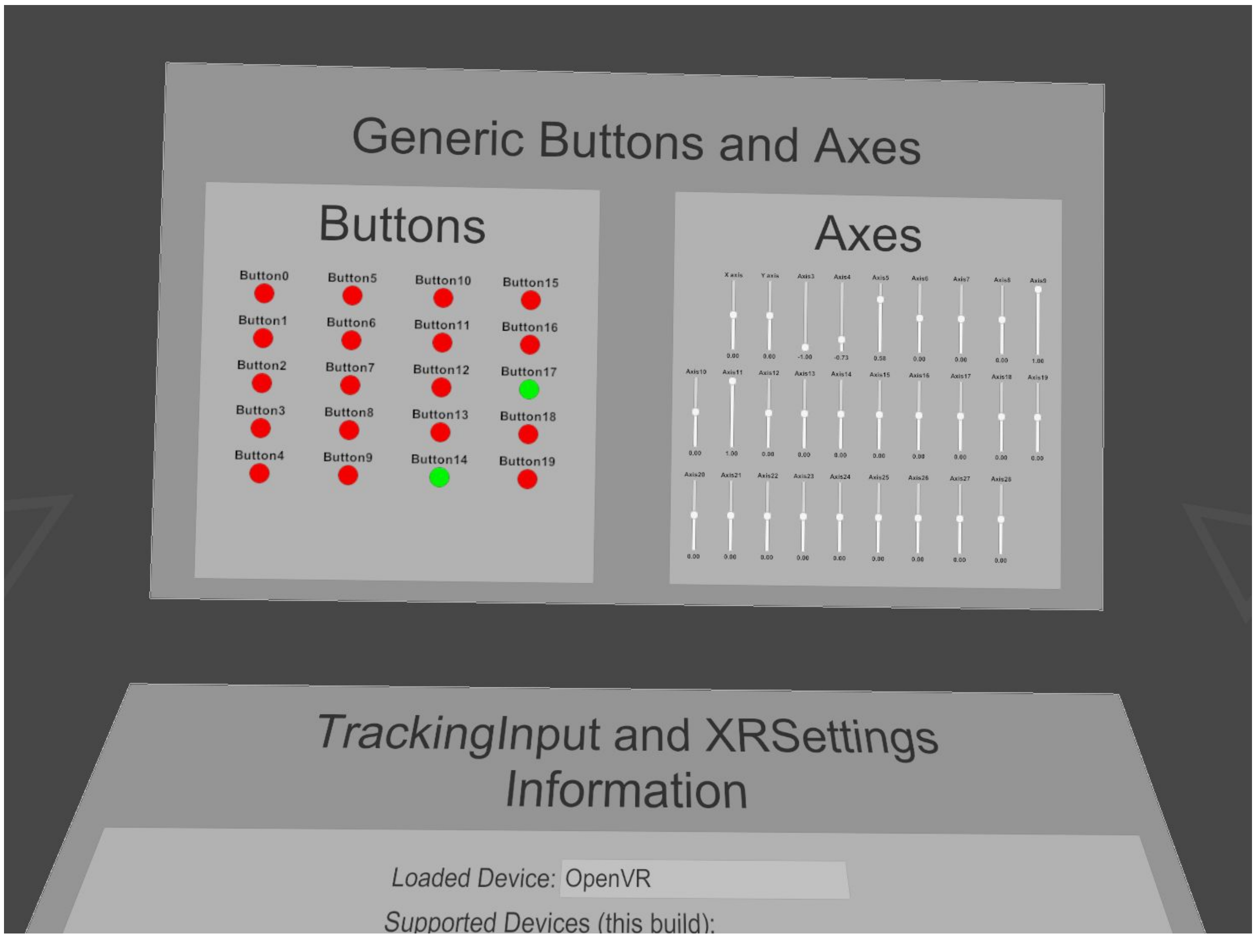
OpenVR,

Joystick Names:

<0> OpenVR Controller(Vive Controller DVT) - Right
<1> OpenVR Controller(Vive. Controller MV) - Left

Input Visualization, Generic Buttons, and Axes

The Generic Buttons and Axes panel is located above the Input Tracking and XR Settings panels. Verify that actuation of buttons, thumbsticks, touchpads, and triggers on an XR controller cause the appropriate response from a button or axis. Platform specific bindings can be found at https://docs.unity3d.com/Manual/XR.html for comparison.



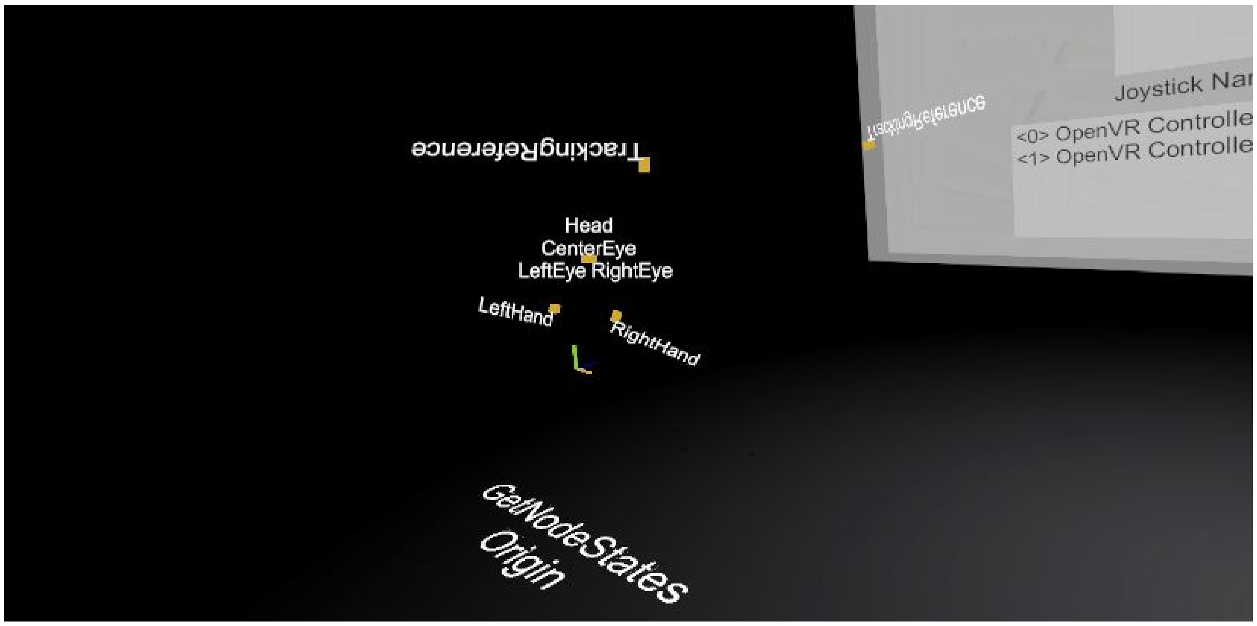
Loaded Device: OpenVR

Supported Devices (this build):

Node States

Orange cubes should appear to the left of the Tracking Input panel. The configuration of these cubes should mirror the current physical VR setup. These cubes represent XR Nodes queried from UnityEngine.XR.InputTracking.GetNodeStates(). Take note of the red, green, and blue axes and ensure that the axes origin matches the expected origin of the platform - either on the ground or where the head node visualizers begin during scene initialization. Tracking references such as tracking cameras or Vive Lighthouses are also visualized for some platforms.

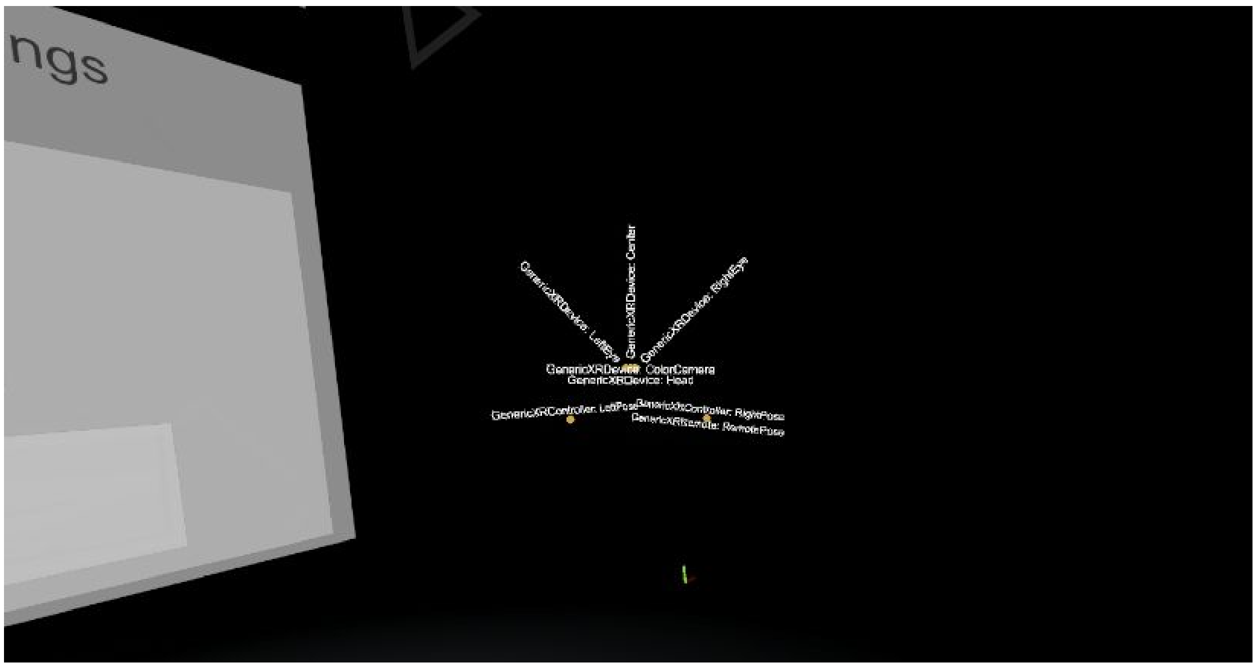
Compare the visual representation to the physical orientation of the hardware devices. Verify that the device's head, left eye, right eye, and center eye appear and track correctly. If tracking references exist, then ensure that they appear and track correctly. Finally, manipulate the controllers to verify that they appear and track as expected.



Tracked Pose Driver

Orange spheres should appear to the right of the Tracking Input panel. The configuration of these spheres should mirror the current physical VR setup. These spheres represent physical devices controlled by the TrackedPoseDriver component. Take note of the red, green, and blue axes and ensure that the axes origin matches the expected origin of the platform - either on the ground or where the head node visualizers begin during scene initialization.

Compare the visual representation to the physical orientation of the hardware devices. A visual representation will only exist for one of each: left controller, right controller, left eye, right eye, center eye, head, color camera, device pose. If a node is unused by the current XR SDK configuration then will either be positioned at the origin or alias to a similar device.


ngs
GenericXRDevice: LeftEye
GenericXRDevice: Center
GenericXRDevice: RightEye
GenericXRDevice: ColorCamera
GenericXRDevice: Head
GenericXRController: LeftPose
GenericXRController: RightPose
GenericXRRemote: RemotePose