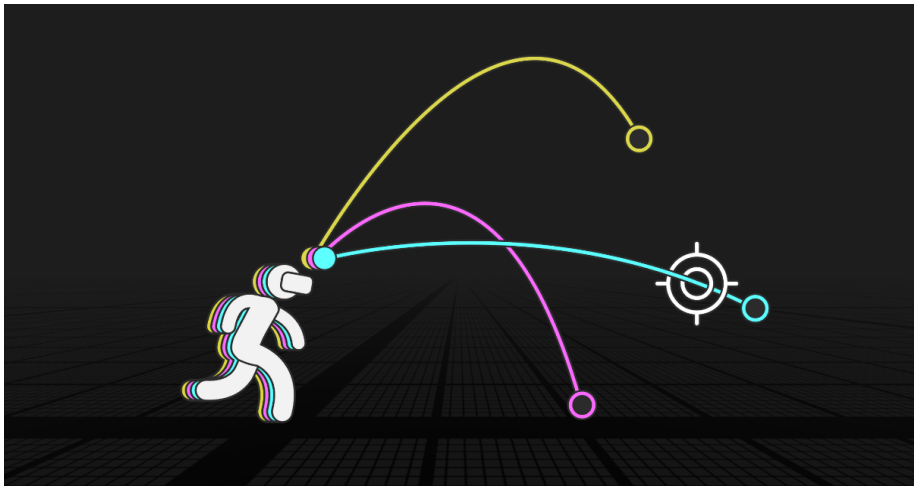
VR Throw Lab Manual

Version 1.1



Introduction

It can be tricky to get throwing mechanics to feel *just right* in VR, especially if you're asking the user to throw far and/or accurately. There isn't a one-size-fits-all solution because what feels right to the user is going to depend on tasks you're asking them to perform, the hardware they're physically holding, and their expectation of how thrown objects should behave in your virtual world.

The goal of VR Throw Lab is to provide tools you can use to create the throwing mechanics that are right for your VR game, and maintain a consistent experience across platforms. Most importantly, it's designed to be used while in the headset so you can fiddle with values without taking the headset on and off.

Quick Start

SteamVR

Open the package *Assets\ThrowLab\SteamVR\ThrowLab_SteamVRIntegration.unitypackage.*
Open the scene *Assets\ThrowLab\SteamVR\Lab_SteamVR.*

The **ThrowLab_SteamVRPlayer** is based on SteamVR's **Player** prefab with three important components added to each **Hand**:

SteamVR_GrabThresholdModifier
SteamVR_DeviceDetector
ViveUILaserPointer

As well as a **VRInputModule** added as a top-level child object. If you would like to use an existing player rig from your project, just make these additions on your own rig. The **VRInputModule** and **ViveUILaserPointer** are only necessary for interacting with the Lab UI and can be removed when no longer needed. If you have your own system for interacting with world-space canvases, you can also substitute this laser pointer system with your own.

There are two throwable prefabs in the SteamVR folder: **SVRThrowableBall** and **SVRThrowableCube.** These are based on SteamVR **Throwables** with two changes: a **ThrowHandle** was added and **Throwable** was replaced with **ThrowLabThrowable.** To set up your own throwable objects to work with the Throw Lab system, simply make these two changes (if you put the inspector into Debug mode, you can drag **ThrowLabThrowable** onto the **Throwable** script field and not lose any associated data). Be aware that some trajectory-related options in **ThrowLabThrowable** will not have any effect and are simply inherited from **Throwable.**

WARNING: **SteamVR_GrabThresholdModifier.cs** uses SteamVR_Actions in a very non-standard way, and should be removed when no longer needed for experimentation to prevent conflict with the SteamVR bindings system.

Oculus

Import the package *Assets\ThrowLab\Oculus\ThrowLab_OculusIntegration.unitypackage*
Open the scene *Assets\ThrowLab\Oculus\Lab_Oculus*

The **ThrowLab_OVRCameraRig** is based on Oculus's **[CameraRig]** prefab with three important components added to each **Hand**:

Oculus_GrabThresholdModifier
Oculus_DeviceDetector
OVRUILaserPointer

As well as a **VRInputModule** added as a top-level child object. If you would like to use an existing player rig from your project, just make these additions on your own rig. The **VRInputModule** and **OVRUILaserPointer** are only necessary for interacting with the Lab UI and can be removed when no longer needed. If you have your own system for interacting with world-space canvases, you can also substitute this laser pointer system with your own.

There are two throwable prefabs in the Oculus folder: **OVRThrowableCube** and **OVRThrowableBall.** These are based on Oculus **OVRGrabbables** with two changes: a **ThrowHandle** was added and **OVRGrabbable** was replaced with **ThrowLabOVRGrabbable.** To set up your own throwable objects to work with the Throw Lab system, simply make these two changes (if you put the inspector into Debug mode, you can drag **ThrowLabOVRGrabbable** onto the **OVRGrabbable** script field and not lose any associated data).

Unity XR Toolkit

Import the package *Assets\ThrowLab\UnityXR\ThrowLab_UnityXRIntegration.unitypackage*
Open the scene *Assets\ThrowLab\UnityXR\Lab_UnityXR*

The **ThrowLab_UnityXRRig** is based on Unity's **XRRig_Demo** prefab from their their XR-Interaction-Toolkit-Examples project with two important components added to each **XRController**:

UnityXR_GrabThresholdModifier
UnityXR_DeviceDetector

As well as a **LaserControllers** added alongside the **BaseControllers** to allow interaction with world-space canvases. Note that **TrackedDeviceGraphicRaycasters** were added to each Canvas in the Lab to support this system. If you would like to use an existing player rig from your project, just make these additions on your own rig.

There are two throwable prefabs in the UnityXR folder: **UXRThrowableCube** and **UXRThrowableSphere.** These are based on Unity's **XRGrabbableInteractables** with two changes: a **ThrowHandle** was added and **XRGrabbableInteractable** was replaced with **ThrowLabXRGrabbableInteractable.** To set up your own throwable objects to work with the Throw Lab system, simply make these two changes (if you put the inspector into Debug mode, you can drag **ThrowLabXRGrabbableInteractable** onto the **XRGrabbableInteractable** script field and not lose any associated data). Be aware that some trajectory-related options in **ThrowLabXRGrabbableInteractable** will not have any effect and are simply inherited from **XRGrabbableInteractable.**

NOTE: Grab Threshold Modifications will not persist past runtime. To make permanent changes, you will need to add/modify the Press Interaction for the Select action in your Input Actions asset (Unity Manual) and modify the Button Release Threshold under Project Settings > Input System Package.

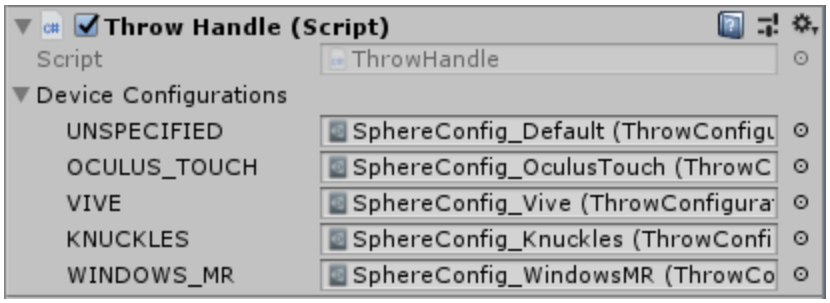
Lab Features

Device-Specific Throw Configurations

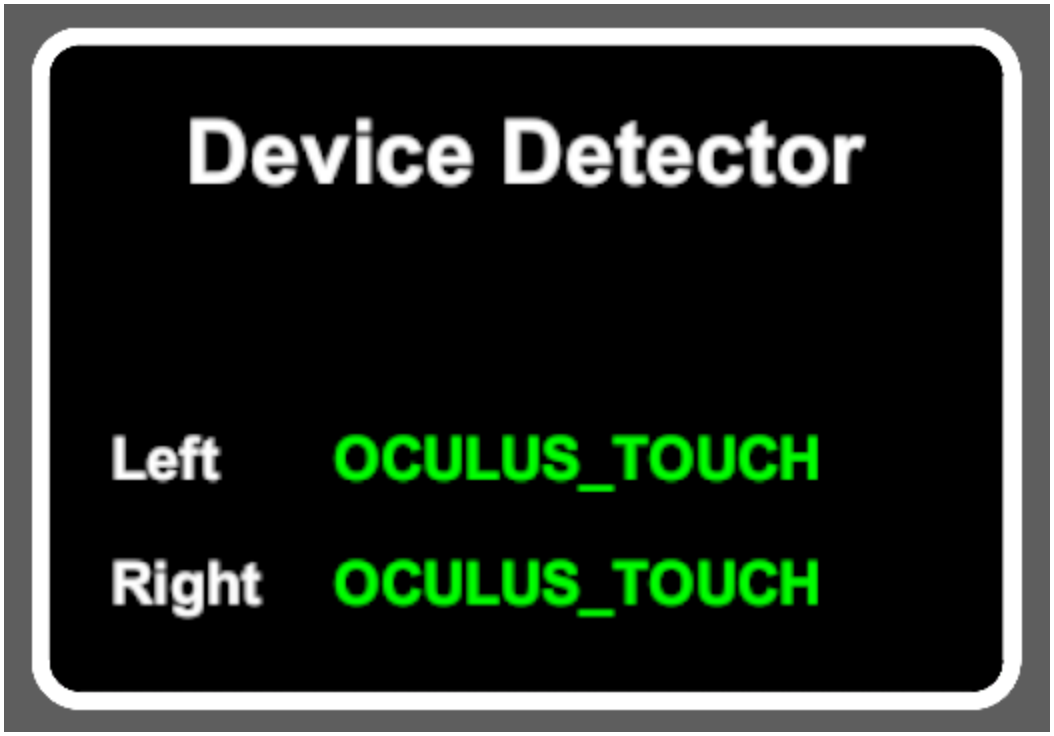
ThrowConfigurations are ScriptableObjects that define how an object should behave when thrown. You can create a new ThrowConfiguration using the asset creation menu:

Assets > Create > ThrowLab > ThrowConfiguration

Drop a ThrowConfiguration onto a ThrowHandle device slot to apply it to a throwable.

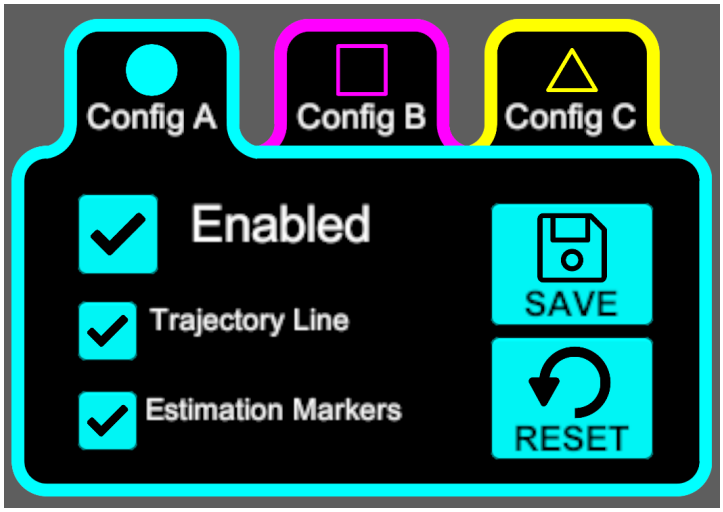


If the user's device is detected, the corresponding ThrowConfiguration will be automatically loaded as the active configuration. In the Lab, this panel will show what controller types have been detected.



Parallel Configurations

You can directly compare up to 3 different configurations by throwing multiple objects simultaneously. This is useful for visualizing the exact difference between two configurations. For instance, if you want to be able to tell how much Aim Assist will be applied to a given throw, this feature will let you throw two objects, one with Aim Assist enabled and one without, so you can compare the resulting trajectories.



These tabs on the UI can be used to change which variant configuration is currently being edited. Note that the color of the UI will reflect the active variant.

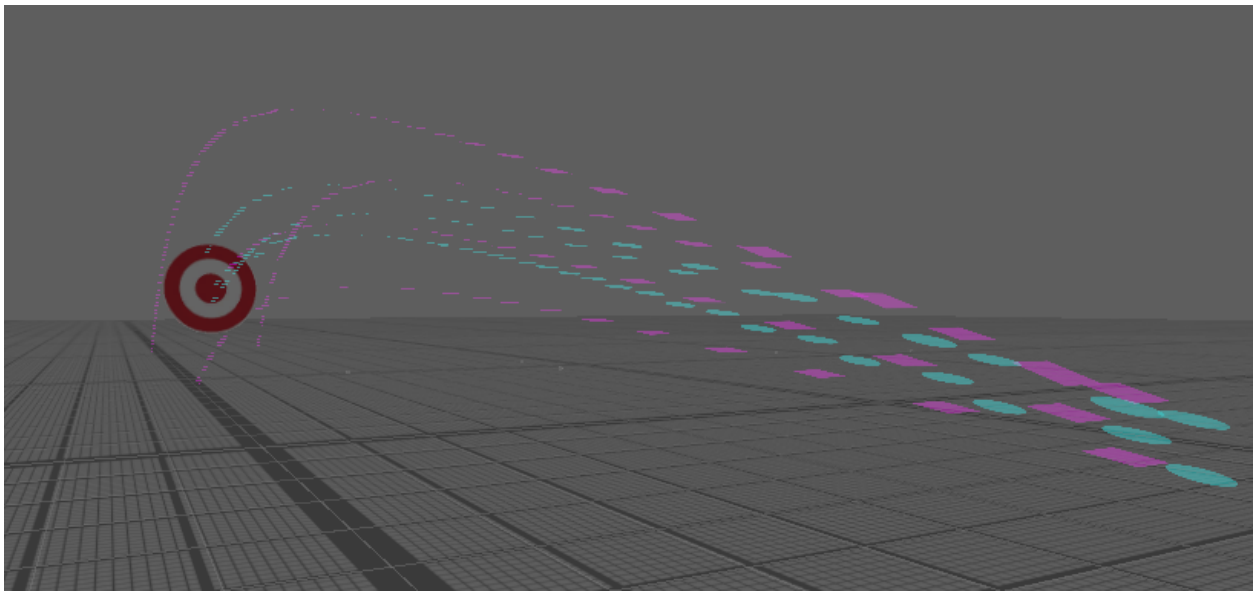
Enabled

Should an object with this configuration be attached to spawned throwables.

> NOTE: At least one configuration variant must be enabled to spawn throwables in Throw Lab.

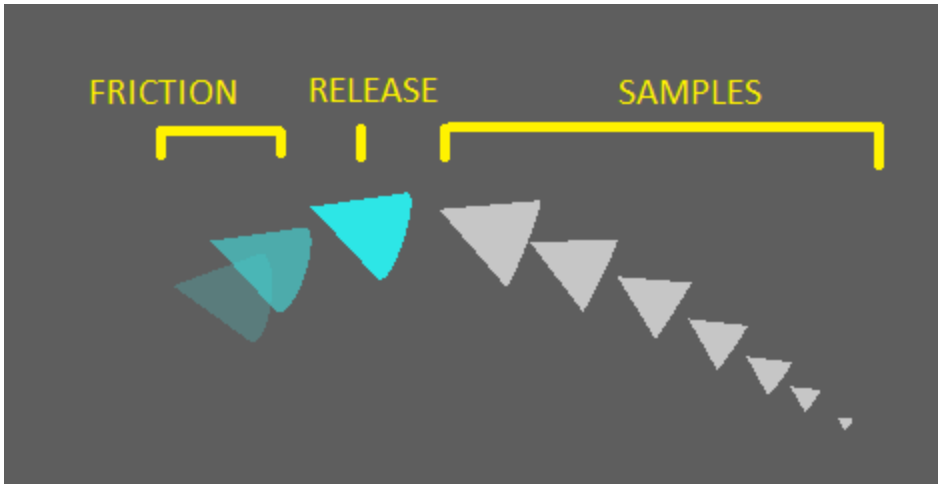
Trajectory Line

Trajectory lines will trace the path of a thrown object until it collides with something. If you would like to experiment with a configuration without cluttering the scene with colorful lines, you can uncheck this box to disable them for a particular configuration.



Estimation Markers

These markers can be used to visualize the samples that are being used to calculate the smoothed velocity, the moment of release, and influence being applied to the object after release.



The samples will appear white, and be scaled to reflect the sample's weight for smoothing purposes. On the frame that the object is released, a colored marker will be placed to show the position and direction of the launch velocity. If the configuration has Friction active, semi-transparent colored markers will be placed to show the diminishing influence of the hand over the ball.

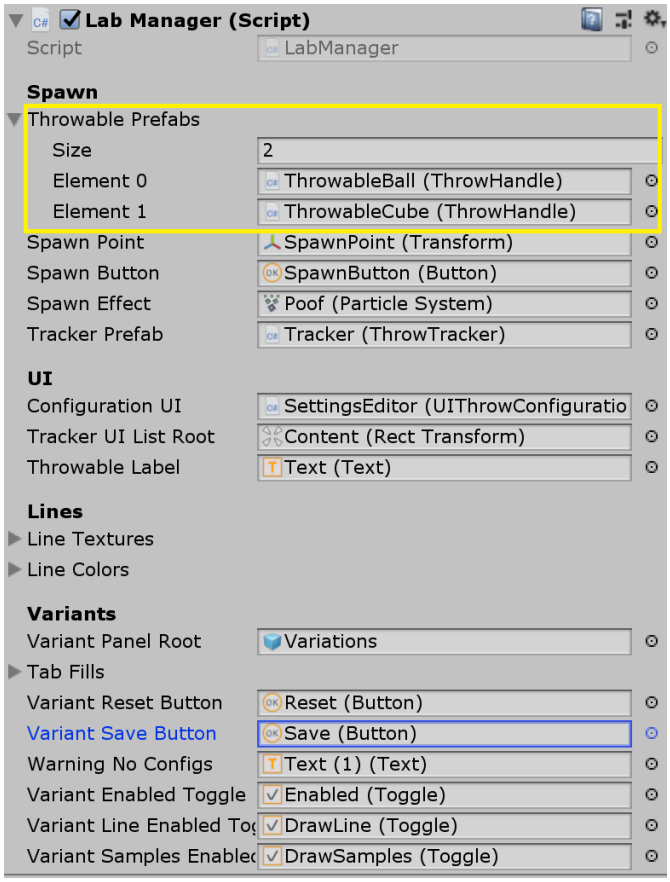
Save

When changing values on this UI, you will be making changes to the variants, not the original asset. If you're satisfied with one of the variants and would like to apply changes to the source ThrowConfiguration, use the SAVE button. The saving function will work in both the editor and standalone builds in case you would like to allow your users to tweak some of these settings as well.

Reset

The RESET button can be used to reset the values of the current variant to match the source ThrowConfiguration.

Throwables Playlist

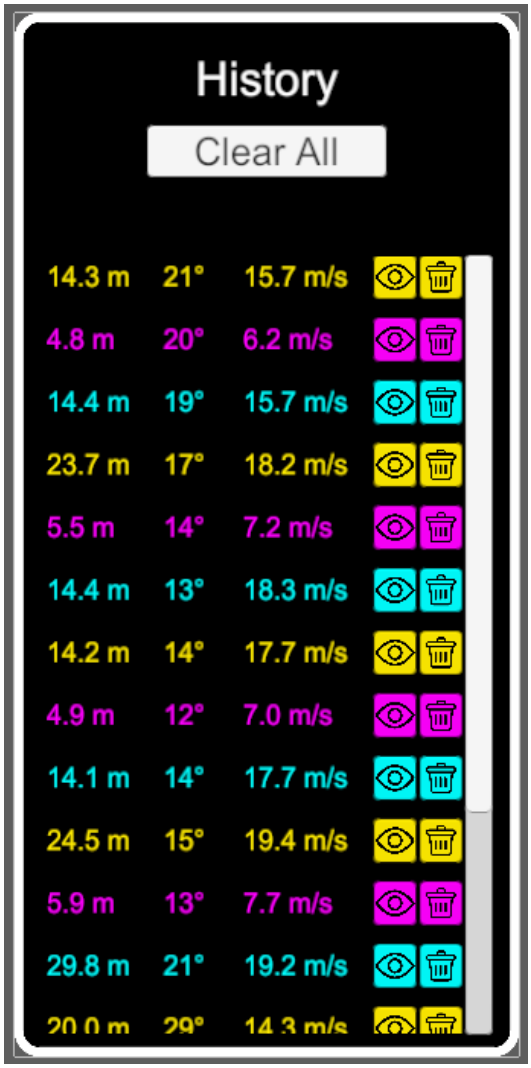


The LabManager has a List of ThrowHandles called *Throwable Prefabs*. Any objects that you would like to be accessible for tuning at runtime should be added to this list in the inspector.

At runtime, you can cycle through the list using the left and right arrows in this UI.

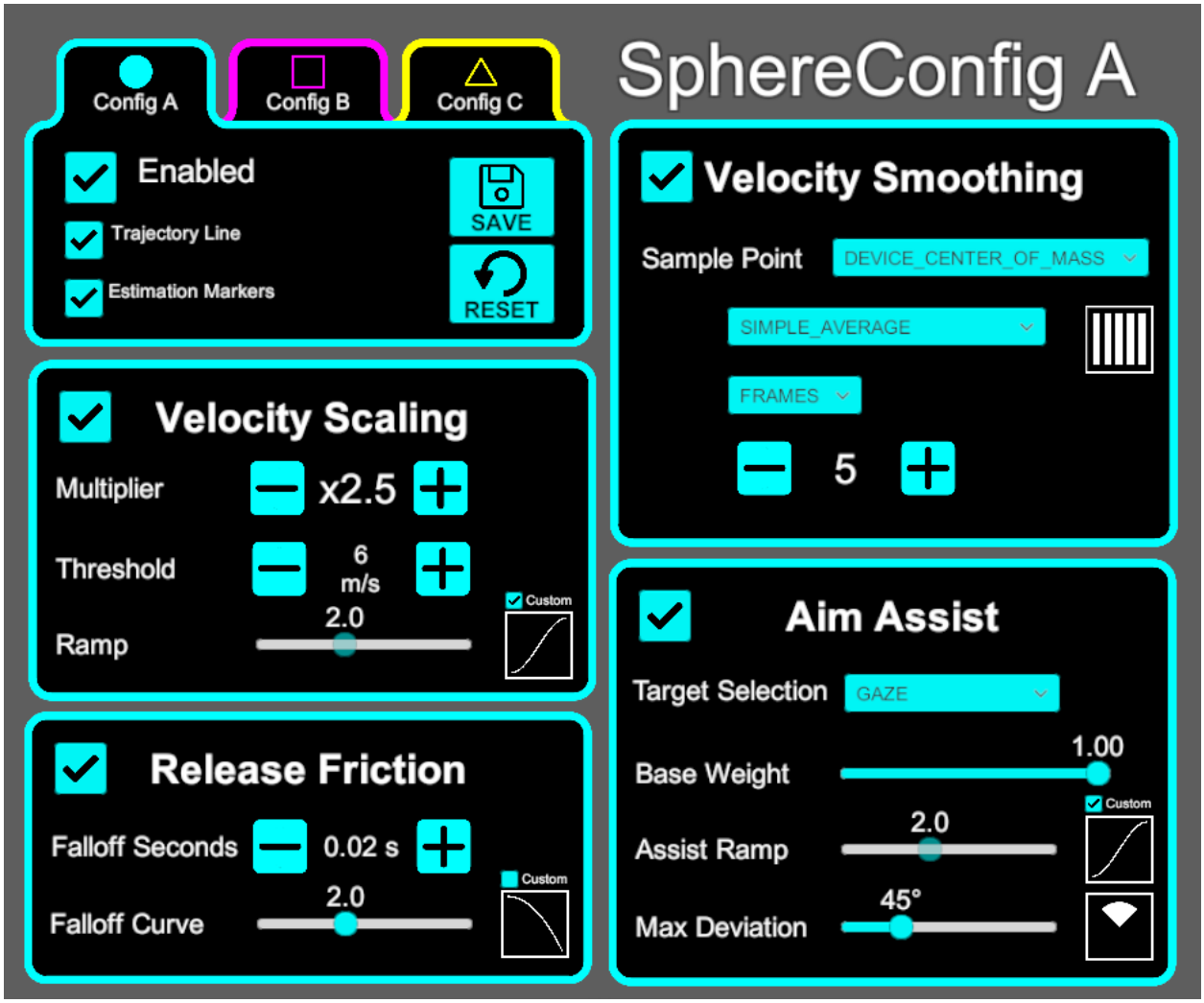
History

A record of previous throws is displayed on this Canvas. It lists the distance traveled before colliding with something, the initial speed, and the launch angle relative to the ground plane. There are also buttons to hide the visibility of any visualizations associated with the throw, and clear it from the scene completely. The Clear All button will clear everything at once for a clean slate.



Configuration Editor

This UI can be used to experiment with **ThrowConfiguration** values. Three variants are generated for each **ThrowConfiguration** (A,B and C). Use the colored tabs to change which variant is currently being edited by the UI.



Velocity Smoothing

The motion of the controller during the single frame in which an object is released doesn't usually reflect the user's intended trajectory. By recording data points the motion of the controller over time, we can get a better approximation of the speed and direction that the user intended.

Smoothing Enabled	Enables velocity smoothing.
Sample Point	Point from which velocity should be estimated. [DEVICE_CENTER_OF_MASS: Sample velocity from the controller's center of mass. RECOMMENDED] [HAND_TRACKED_POSITION: Sample velocity from the controller's tracked position] [OBJECT_CENTER: Sample velocity from the origin of the held object.]

	[OBJECT_CUSTOM_OFFSET: Sample velocity from a custom point on the object. This point must be represented by a child GameObject named “CustomVelocitySensor”.]
Estimation Function	The algorithm used to calculate a launch velocity from data points collected over the course of the user’s throw. [SIMPLE_AVERAGE: All samples are weighted evenly.] [WEIGHTED_AVERAGE: Samples weights decay linearly. Newer samples will have a greater effect.] [EXPONENTIAL_AVERAGE: Sample weights decay exponentially. Newer samples will have a greater effect.] [CUSTOM_CURVE] Sample weights determined by an AnimationCurve set in the editor.
Sample Period Measurement	[TIME: Save samples over ***Period Seconds*** number of seconds. RECOMMENDED] [FRAMES: Save samples over ***Period Frames*** number of frames.]
Sample Time	[UNSCALED: Record samples in Update(). Save for a number of seconds in realtime. RECOMMENDED] [SCALED: Record samples in Update(). Save for a number of seconds in game time. May result in higher speed throws if Time.timeScale is less than 1.] [FIXED: Record samples in FixedUpdate(). Save for a number of seconds in realtime. May be useful if your throwable is moved by Physics.]
Period Frames	Number of frames to save a sample for if period type is FRAMES.
Period Seconds	Number of seconds to save a sample for if period type is TIME.

Velocity Scaling

There’s a limit to how fast the user can safely throw their arms around. If you want the user to be able to throw long distances, you’ll need to scale the speed of their throws.

This value will ramp up as the user’s throw approaches the **Threshold** speed so that gentle tosses are not affected by the same scaling as a full overhand throw.

Scale Enabled	Enables velocity scaling.

Scale Multiplier	The maximum increase to apply to the throw. A throw that
Scale Threshold	The threshold speed beyond which any throw will receive maximum scaling. Any throws with speed less than this will be scaled by some fraction of the full amount.
Scale Ramp Exponent	Exponent value for the exponential curve by which scaling increases as throw speed approaches ***Scale Threshold***. Higher values result in less scaling for gentler throws.
Use Scale Ramp Custom Curve	Use an AnimationCurve instead of an exponential curve for finer control.
Scale Ramp Custom Curve	Use instead of an exponential curve. Must be edited in the Inspector.

Aim Assistance

An imperceptible amount of assistance can dramatically improve how accurate throwing feels to the user. If you know that the user will be aiming for certain objects (eg. enemies) those objects can be tagged with a ThrowTarget component that will make them potential candidates for assisted targeting. If no ThrowTargets are found in the scene, no assistance can be applied.

This algorithm is designed to apply more assistance the more accurate the throw is. This increases precision for throws that are already fairly accurate, while throws that are wildly off target will not be affected.

NOTE: Aim assist will never modify the speed of a launched object, only the direction. A throw that is too slow to reach a target even if redirected will not receive any assist.

Assist Enabled	Enables aim assist.
Assist Weight	Can be used to soften the overall effect. Higher values result in more assistance. [0: No assistance]
Assist Range Degrees	The maximum deviation from a perfectly accurate launch trajectory that will still receive assistance. Higher values result in more assistance. [0: Only a perfect throw would be assisted.]

	[180: Every throw would be assisted.]
Assist Ramp Exponent	Exponent value for the exponential curve by which assistance weakens as user accuracy approaches the ***Assist Range Degrees*** threshold. Higher values result in less assistance. [0: No falloff. All throws receive the exact same amount of assistance.]
Assist Target Method	[GAZE: will pick targets based on where the user's head is pointed. Good for targeting a specific object.] [NEAREST: will pick a target at the moment of release based on whichever the throw is closest hitting already.]
Use Assist Ramp Custom Curve	Use an AnimationCurve instead of an exponential curve for finer control.
Scale Assist Custom Curve	Use instead of an exponential curve. Must be edited in the Inspector.

Friction

When something is thrown it isn't released instantaneously, it rolls off the fingertips. This module is designed to simulate that smooth release by applying diminishing influence on the thrown object over a brief period of time after it has been released.

Friction Enabled	Enables friction.
Friction Falloff Seconds	Time period immediately following release that the hand should exert some amount of influence of the object's trajectory.
Friction Falloff Exponent	Exponent value for the exponential curve of diminishing friction with a thrown object.
Use Friction Falloff Custom Curve	Use an AnimationCurve instead of an exponential curve for finer control.
Friction Falloff Custom Curve	Use instead of an exponential curve. Must be edited in the Inspector.

Grab Threshold

Sets the thresholds for grabbing and releasing an object. For this feature to work, each Hand must must include a GrabThresholdModifier component (see Player Rig Integration)

NOTE: These values are not saved as part of a ThrowConfiguration. They should ultimately be set on the hands and apply universally to all interactables.

Custom SDK Integration

If your SDK isn't officially supported by ThrowLab, some scripting may be required to bridge the two systems.

Player Rig

There are three things that a player rig will need to make full use of VR Throw Lab.

1. **Input system for interacting with world-space Canvases.**

   The Lab UI is a basic world-space Unity Canvas. To do anything at all with it you'll need to be able to click buttons and drag sliders. There are included prefabs from wacki's

Unity-VRInputModule project. They provide a convenient way to drop this functionality onto SteamVR and Oculus rigs without needing to modify the UI, and are used on the default player rigs in the **Lab_SteamVR** and **Lab_Oculus** demo scenes. For the Unity XR Interaction Toolkit demo (2019.3+ only), Unity's own laser pointer solution was implemented as it is in their XR-Interaction-Toolkit-Examples project.

2. **Hands with GrabThresholdModifier components.**

   GrabThresholdModifier is an abstract class that can be implemented on a platform-specific basis to provide an access point for grab and release thresholds. If you don't care about being able to modify these values in the headset, you can ignore this step. These components can be removed when no longer needed for experimentation; most interaction systems will have a way to set your desired values in the Inspector once you've found them.

3. **Hands with DeviceDetector components**

   SteamVR_DeviceDetector uses OpenVR.System.GetStringTrackedDeviceProperty(..) to identify the user's hardware, while Oculus_DeviceDetector assumes the user will have Touch controllers. UnityXR_DeviceDetector uses XRController.inputDevice.name.

Throwable Objects

Each throwable object needs a ThrowHandle for trajectory calculation. A ThrowHandle needs to be aware of OnAttach() and OnDetach() events, as well as the grabbing Hand GameObject and, if different, the root GameObject of all hand colliders that may interfere with the thrown object after release (like fingers). Crucially, any attempt by other components to set the velocity of the throwable upon detachment must either be suppressed or immediately overwritten.

This will generally require extending the base functionality of the throwable scripts in your project, either through a subclass or editing the script itself. See ThrowLabOVRGrabbable.cs, ThrowLabThrowable.cs and ThrowLabXRGrabbable.cs for examples of how to override the stock behavior of your interaction system to use ThrowHandles.

Change Log

Version 1.1

- Add support for UnityXR ActionBasedController
- Phase out support for UnityXR Device based XRController
- Add Integration scripts for HurricaneVR and VRIF

Version 1.0.4

- Fix compatibility with new Unity XR Toolkit version

Version 1.0.3

- Compensate for root motion.
- Bug Fixes.

Version 1.0.2

- Bug Fixes

Version 1.0.1

- Add Unity XR Toolkit support.

Contact

Please send any comments, questions, bugs or feature requests to:

Marc Huet

marc@cloudfinegames.com