Introduction

This file provides documentation on how to use the included prefabs and scripts along with a breakdown of the example scenes.

Prefabs

A collection of pre-defined usable prefabs have been included to allow for each drag-and-drop set up of common elements.

FramesPerSecondCanvas

Overview

This canvas adds a frames per second text element to the headset. To use the prefab it must be placed into the scene then the headset camera needs attaching to the canvas:

- Select `FramesPerSecondCanvas` object from the scene objects
- Find the `Canvas` component
- Set the `Render Camera` parameter to `Camera (eye)` which can be found in the `[CameraRig]` prefab.

There are a number of parameters that can be set on the Prefab. Expanding the `FramesPerSecondCanvas` object in the hierarchy view shows the child `FramesPerSecondText` obejct and clicking on that reveals additional paramters which can be set via the `FramesPerSecondViewer` script (which can be found in `SteamVR_Unity_Toolkit/Scripts/Helper/FramesPerSecondViewer`)

Inspector Parameters

- **Display FPS:** Toggles whether the FPS text is visible.

- **Target FPS:** The frames per second deemed acceptable that is used as the benchmark to change the FPS text colour.
- **Font Size:** The size of the font the FPS is displayed in.
- **Position:** The position of the FPS text within the headset view.
- **Good Color:** The colour of the FPS text when the frames per second are within reasonable limits of the Target FPS.
- **Warn Color:** The colour of the FPS text when the frames per second are falling short of reasonable limits of the Target FPS.
- **Bad Color:** The colour of the FPS text when the frames per second are at an unreasonable level of the Target FPS.

Example

`SteamVR_Unity_Toolkit/Examples/018_CameraRig_FramesPerSecondCounter` displays the frames per second in the centre of the headset view. Pressing the trigger generates a new sphere and pressing the touchpad generates ten new spheres. Eventually when lots of spheres are present the FPS will drop and demonstrate the prefab.

ObjectTooltip

Overview

This adds a UI element into the World Space that can be used to provide additional information about an object by providing a piece of text with a line drawn to a destination point.

There are a number of parameters that can be set on the Prefab which are provided by the `SteamVR_Unity_Toolkit/Scripts/VRTK_ObjectTooltip` script which is applied to the prefab.

Inspector Parameters

- **Display Text:** The text that is displayed on the tooltip.
- **Font Size:** The size of the text that is displayed.
- **Container Size:** The size of the tooltip container where `x = width` and `y = height`.
- **Draw Line From:** An optional transform of where to start drawing the line from. If one is not provided the the centre of the tooltip is used for the initial line position.
- **Draw Line To:** A transform of another object in the scene that a line will be drawn from the tooltip to, this helps denote what the tooltip is in relation to. If no transform is provided and the tooltip is a child of another object, then the parent object's transform will be used as this destination position.
- **Line Width:** The width of the line drawn between the tooltip and the destination transform.
- **Font Color:** The colour to use for the text on the tooltip.
- **Container Color:** The colour to use for the background container of the tooltip.
- **Line Color:** The colour to use for the line drawn between the tooltip and the destination

transform.

Example

`SteamVR_Unity_Toolkit/Examples/029_Controller_Tooltips` displays two cubes that have an object tooltip added to them along with tooltips that have been added to the controllers.

ControllerTooltips

Overview

This adds a collection of Object Tooltips to the Controller that give information on what the main controller buttons may do. To add the prefab, it just needs to be added as a child of the relevant controller e.g. `[CameraRig]/Controller (right)` would add the controller tooltips to the right controller.

If the transforms for the buttons are not provided, then the script will attempt to find the attach transforms on the default controller model in the `[CameraRig]` prefab.

If no text is provided for one of the elements then the tooltip for that element will be set to disabled.

There are a number of parameters that can be set on the Prefab which are provided by the `SteamVR_Unity_Toolkit/Scripts/VRTK_ControllerTooltips` script which is applied to the prefab.

Inspector Parameters

- **Trigger Text:** The text to display for the trigger button action.
- **Grip Text:** The text to display for the grip button action.
- **Touchpad Text:** The text to display for the touchpad action.
- **App Menu Text:** The text to display for the application menu button action.
- **Tip Background Color:** The colour to use for the tooltip background container.
- **Tip Text Color:** The colour to use for the text within the tooltip.
- **Tip Line Color:** The colour to use for the line between the tooltip and the relevant controller button.
- **Trigger:** The transform for the position of the trigger button on the controller (this is usually found in `Model/trigger/attach`.
- **Grip:** The transform for the position of the grip button on the controller (this is usually found in `Model/lgrip/attach`.
- **Touchpad:** The transform for the position of the touchpad button on the controller (this is usually found in `Model/trackpad/attach`.
- **App Menu:** The transform for the position of the app menu button on the controller (this is usually found in `Model/button/attach`.

Example

`SteamVR_Unity_Toolkit/Examples/029_Controller_Tooltips` displays two cubes that have an object tooltip added to them along with tooltips that have been added to the controllers.

RadialMenu

Overview

This adds a UI element into the world space that can be dropped into a Controller object and used to create and use Radial Menus from the touchpad.

If the RadialMenu is placed inside a controller, it will automatically find a `VRTK_ControllerEvents` in its parent to use at the input. However, a `VRTK_ControllerEvents` can be defined explicitly by setting the `Events` parameter of the `Radial Menu Controller` script also attached to the prefab.

The RadialMenu can also be placed inside a `VRTK_InteractableObject` for the RadialMenu to be anchored to a world object instead of the controller. The `Events Manager` parameter will automatically be set if the RadialMenu is a child of an InteractableObject, but it can also be set manually in the inspector. Additionally, for the RadialMenu to be anchored in the world, the `RadialMenuController` script in the prefab must be replaced with `VRTK_IndependentRadialMenuController`. See the script information for further details on making the RadialMenu independent of the controllers.

There are a number of parameters that can be set on the Prefab which are provided by the `SteamVR_Unity_Toolkit/Scripts/Controls/2D/RadialMenu/RadialMenu.cs` script which is applied to the `Panel` child of the prefab.

Inspector Parameters

- **Buttons:** Array of Buttons that define the interactive buttons required to be displayed as part of the radial menu. Each Button has the following properties:
  - **ButtonIcon:** Icon to use inside the button arc (should be circular).
  - **OnClick():** Methods to invoke when the button is clicked.
  - **OnHold():** Methods to invoke each frame while the button is held down.
  - **OnHoverEnter():** Methods to invoke when button is first hovered over.
  - **OnHoverExit():** Methods to invoke when button leaves the hovered state.
- **Button Prefab:** The base for each button in the menu, by default set to a dynamic circle arc that will fill up a portion of the menu.
- **Button Thickness:** Percentage of the menu the buttons should fill, 1.0 is a pie slice, 0.1 is a thin ring.
- **Button Color:** The background color of the buttons, default is white.
- **Offset Distance:** The distance the buttons should move away from the center. This creates space between the individual buttons.

- **Offset Rotation:** The additional rotation of the Radial Menu.
- **Rotate Icons:** Whether button icons should rotate according to their arc or be vertical compared to the controller.
- **Icon Margin:** The margin in pixels that the icon should keep within the button.
- **Hide On Release:** Whether the buttons should be visible when not in use.
- **Execute On Unclick:** Whether the button action should happen when the button is released, as opposed to happening immediately when the button is pressed.
- **Base Haptic Strength:** The base strength of the haptic pulses when the selected button is changed, or a button is pressed. Set to zero to disable.
- **Menu Buttons:** The actual GameObjects that make up the radial menu.
- **Regenerate Buttons:** Button to force regeneration of the radial menu in the editor.

Example

`SteamVR_Unity_Toolkit/Examples/030_Controls_RadialTouchpadMenu` displays a radial menu for each controller. The left controller uses the `Hide On Release` variable, so it will only be visible if the left touchpad is being touched. It also uses the `Execute On Unclick` variable to delay execution until the touchpad button is unclicked. The example scene also contains a demonstration of anchoring the RadialMenu to an interactable cube instead of a controller.

VRTK_IndependentRadialMenuController

This script inherited from `RadialMenuController` and therefore can be used instead of `RadialMenuController` to allow the RadialMenu to be anchored to any object, not just a controller. The RadialMenu will show when a controller is near the object and the buttons can be clicked with the `Use Alias` button. The menu also automatically rotates towards the user. To convert the default `RadialMenu` prefab to be independent of the controllers: * Make the `RadialMenu` a child of an object other than a controller. * Position and scale the menu by adjusting the transform of the `RadialMenu` empty. * Replace `RadialMenuController` with `VRTK_IndependentRadialMenuController`. * Ensure the parent object has the `VRTK_InteractableObject` script. * Verify that `Is Usable` and `Hold Button to Use` are both checked. * Attach `VRTK_InteractTouch` and `VRTK_InteractUse` scripts to the controllers.

VRTK_IndependentRadialMenuController - Inspector Parameters

- **Events Manager:** If the RadialMenu is the child of an object with VRTK_InteractableObject attached, this will be automatically obtained. It can also be manually set.
- **Add Menu Collider:** Whether or not the script should dynamically add a SphereCollider to surround the menu.
- **Collider Radius Multiplier:** This times the size of the RadialMenu is the size of the collider.
- **Hide After Execution:** If true, after a button is clicked, the RadialMenu will hide.
- **Offset Radius Multiplier:** How far away from the object the menu should be placed, relative to the size of the RadialMenu.
- **Rotate Towards:** The object the RadialMenu should face towards. If left empty, it will automatically try to find the Camera (eye) object within the SteamVR CameraRig.

ConsoleViewerCanvas

Overview

This canvas adds the unity console log to a world game object. To use the prefab, it simply needs to be placed into the scene and it will be visible in world space. It's also possible to child it to other objects such as the controller so it can track where the user is.

It's also recommended to use the Simple Pointer and UI Pointer on a controller to interact with the Console Viewer Canvas as it has a scrollable text area, a button to clear the log and a checkbox to toggle whether the log messages are collapsed.

Inspector Parameters

- **Font Size:** The size of the font the log text is displayed in.
- **Info Message:** The colour of the text for an info log message.
- **Assert Message:** The colour of the text for an assertion log message.
- **Warning Message:** The colour of the text for a warning log message.
- **Error Message:** The colour of the text for an error log message.
- **Exception Message:** The colour of the text for an exception log message.

Scripts

This directory contains all of the toolkit scripts that add VR functionality to Unity.

- VRTK_ControllerEvents
- VRTK_ControllerActions
- VRTK_DeviceFinder
- VRTK_SimplePointer
- VRTK_BezierPointer
- VRTK_UIPointer
- VRTK_BasicTeleport
- VRTK_HeightAdjustTeleport
- VRTK_HeadsetCollisionFade
- VRTK_PlayerPresence
- VRTK_TouchpadWalking
- VRTK_RoomExtender
- VRTK_InteractableObject
- VRTK_InteractTouch
- VRTK_InteractGrab
- VRTK_InteractUse

Controller Events (VRTK_ControllerEvents)

Overview

The Controller Events script deals with events that the game controller is sending out.

The Controller Events script is attached to a Controller object within the `[CameraRig]` prefab and provides event listeners for every button press on the controller (excluding the System Menu button as this cannot be overriden and is always used by Steam).

When a controller button is pressed, the script emits an event to denote that the button has been pressed which allows other scripts to listen for this event without needing to implement any controller logic. When a controller button is released, the script also emits an event denoting that the button has been released.

The script also has a public boolean pressed state for the buttons to allow the script to be queried by other scripts to check if a button is being held down.

Inspector Parameters

- **Pointer Toggle Button:** The button to use for the action of turning a laser pointer on/off.
- **Pointer Set Button:** The button to use for the action of setting a destination marker from the cursor position of the pointer.
- **Grab Toggle Button:** The button to use for the action of grabbing game objects.
- **Use Toggle Button:** The button to use for the action of using game objects.
- **UI Click Button:** The button to use for the action of clicking a UI element.
- **Menu Toggle Button:** The button to use for the action of bringing up an in-game menu.
- **Axis Fidelity:** The amount of fidelity in the changes on the axis, which is defaulted to 1. Any number higher than 2 will probably give too sensitive results.

Class Variables

- `public bool triggerPressed` - This will be true if the trigger is squeezed about half way in.
- `public bool triggerTouched` - This will be true if the trigger is squeezed a small amount.
- `public bool triggerHairlinePressed` - This will be true if the trigger is squeezed a small amount more from any previous squeeze on the trigger.
- `public bool triggerClicked` - This will be true if the trigger is squeezed all the way until it clicks.
- `public bool triggerAxisChanged` - This will be true if the trigger has been squeezed more or less.
- `public bool applicationMenuPressed` - This will be true if the application menu is held down.
- `public bool touchpadPressed` - This will be true if the touchpad is held down.

- `public bool touchpadTouched` - This will be true if the touchpad is being touched.
- `public bool touchpadAxisChanged` - This will be true if the touchpad touch position has changed.
- `public bool gripPressed` - This will be true if the grip is held down.
- `public bool pointerPressed` - This will be true if the button aliased to the pointer is held down.
- `public bool grabPressed` - This will be true if the button aliased to the grab is held down.
- `public bool usePressed` - This will be true if the button aliased to the use is held down.
- `public bool uiClickPressed` - This will be true if the button aliased to the UI click is held down.
- `public bool menuPressed` - This will be true if the button aliased to the application menu is held down.

Class Events

- `TriggerPressed` - Emitted when the trigger is squeezed about half way in.
- `TriggerReleased` - Emitted when the trigger is released under half way.
- `TriggerTouchStart` - Emitted when the trigger is squeezed a small amount.
- `TriggerTouchEnd` - Emitted when the trigger is no longer being squeezed at all.
- `TriggerHairlineStart` - Emitted when the trigger is squeezed past the current hairline threshold.
- `TriggerHairlineEnd` - Emitted when the trigger is released past the current hairline threshold.
- `TriggerClicked` - Emitted when the trigger is squeezed all the way until it clicks.
- `TriggerUnclicked` - Emitted when the trigger is no longer being held all the way down.
- `TriggerAxisChanged` - Emitted when the amount of squeeze on the trigger changes.
- `ApplicationMenuPressed` - Emitted when the application menu button is pressed.
- `ApplicationMenuReleased` - Emitted when the application menu button is released.
- `GripPressed` - Emitted when the grip button is pressed.
- `GripReleased` - Emitted when the grip button is released.
- `TouchpadPressed` - Emitted when the touchpad is pressed (to the point of hearing a click).
- `TouchpadReleased` - Emitted when the touchpad has been released after a pressed state.
- `TouchpadTouchStart` - Emitted when the touchpad is touched (without pressing down to click).
- `TouchpadTouchEnd` - Emitted when the touchpad is no longer being touched.
- `TouchpadAxisChanged` - Emitted when the touchpad is being touched in a different location.
- `AliasPointerOn` - Emitted when the pointer toggle alias button is pressed.
- `AliasPointerOff` - Emitted when the pointer toggle alias button is released.
- `AliasPointerSet` - Emitted when the pointer set alias button is released.
- `AliasGrabOn` - Emitted when the grab toggle alias button is pressed.
- `AliasGrabOff` - Emitted when the grab toggle alias button is released.
- `AliasUseOn` - Emitted when the use toggle alias button is pressed.
- `AliasUseOff` - Emitted when the use toggle alias button is released.
- `AliasMenuOn` - Emitted when the menu toggle alias button is pressed.
- `AliasMenuOff` - Emitted when the menu toggle alias button is released.
- `AliasUIClickOn` - Emitted when the UI click alias button is pressed.
- `AliasUIClickOff` - Emitted when the UI click alias button is released.

Event Payload

- `uint controllerIndex` - The index of the controller that was used.

- `float buttonPressure` - The amount of pressure being applied to the button pressed. `0f` to `1f`.
- `Vector2 touchpadAxis` - The position the touchpad is touched at. `(0,0)` to `(1,1)`.
- `float touchpadAngle` - The rotational position the touchpad is being touched at, 0 being top, 180 being bottom and all other angles accordingly. `0f` to `360f`.

Class Methods

GetVelocity/0

```
public Vector3 GetVelocity()
```

- Parameters
  - *none*
- Returns
  - `Vector3` - A 3 dimensional vector containing the current real world physical controller velocity.

The GetVelocity method is useful for getting the current velocity of the physical game controller. This can be useful to determine the speed at which the controller is being swung or the direction it is being moved in.

GetAngularVelocity/0

```
public Vector3 GetAngularVelocity()
```

- Parameters
  - *none*
- Returns
  - `Vector3` - A 3 dimensional vector containing the current real world physical controller angular (rotational) velocity.

The GetAngularVelocity method is useful for getting the current rotational velocity of the phyiscal game controller. This can be useful for determining which way the controller is being rotated and at what speed the rotation is occuring.

GetTouchpadAxis/0

```
public Vector2 GetTouchpadAxis()
```

- Parameters
  - *none*
- Returns
  - `Vector2` - A 2 dimensional vector containing the x and y position of where the touchpad is being touched. `(0,0)` to `(1,1)`.

The GetTouchpadAxis method returns the coordinates of where the touchpad is being touched and can be used for directional input via the touchpad. The `x` value is the horizontal touch plane and the `y` value is the vertical touch plane.

GetTouchpadAxisAngle/0

```
public float GetTouchpadAxisAngle()
```

- Parameters
  - *none*
- Returns
  - `float` - A float representing the angle of where the touchpad is being touched. `0f` to `360f`.

The GetTouchpadAxisAngle method returns the angle of where the touchpad is currently being touched with the top of the touchpad being 0 degrees and the bottom of the touchpad being 180 degrees.

GetTriggerAxis/0

```
public float GetTriggerAxis()
```

- Parameters
  - *none*
- Returns
  - `float` - A float representing the amount of squeeze that is being applied to the trigger. `0f` to `1f`.

The GetTriggerAxis method returns a float that represents how much the trigger is being squeezed. This can be useful for using the trigger axis to perform high fidelity tasks or only activating the trigger press once it has exceeded a given press threshold.

GetHairTriggerDelta/0

```
public float GetHairTriggerDelta()
```

- Parameters
  - *none*
- Returns
  - `float` - A float representing the difference in the trigger pressure from the hairline threshold start to current position.

The GetHairTriggerDelta method returns a float representing the difference in how much the trigger is being pressed in relation to the hairline threshold start.

AnyButtonPressed/0

```
public bool AnyButtonPressed()
```

- Parameters
  - *none*
- Returns
  - `bool` - Is true if any of the controller buttons are currently being pressed.

The AnyButtonPressed method returns true if any of the controller buttons are being pressed and this can be useful to determine if an action can be taken whilst the user is using the controller.

Example

`SteamVR_Unity_Toolkit/Examples/002_Controller_Events` shows how the events are utilised and listened to. The accompanying example script can be viewed in `SteamVR_Unity_Toolkit/Examples/Resources/Scripts/VRTK_ControllerEvents_ListenerExample.cs`.

Controller Actions (VRTK_ControllerActions)

Overview

The Controller Actions script provides helper methods to deal with common controller actions. It deals with actions that can be done to the controller.

Class Methods

IsControllerVisible/0

```
public bool IsControllerVisible()
```

- Parameters
  - *none*
- Returns
  - `bool` - Is true if the controller model has the renderers that are attached to it are enabled.

The IsControllerVisible method returns true if the controller is currently visible by whether the renderers on the controller are enabled.

ToggleControllerModel/2

```
public void ToggleControllerModel(bool state, GameObject grabbedChildObject)
```

- Parameters
  - `bool state` - The visibility state to toggle the controller to, `true` will make the controller visibile - `false` will hide the controller model.
  - `GameObject grabbedChildObject` - If an object is being held by the controller then this can be passed through to prevent hiding the grabbed game object as well.
- Returns
  - *none*

The ToggleControllerModel method is used to turn on or off the controller model by enabling or disabling the renderers on the object. It will also work for any custom controllers. It should also not disable any objects being held by the controller if they are a child of the controller object.

SetControllerOpacity/1

```
public void SetControllerOpacity(float alpha)
```

- Parameters
  - `float alpha` - The alpha level to apply to opacity of the controller object. `0f` to `1f`.
- Returns
  - *none*

The SetControllerOpacity method allows the opacity of the controller model to be changed to make the controller more transparent. A lower alpha value will make the object more transparent, such as `0.5f` will make the controller partially transparent where as `0f` will make the controller completely transparent.

HighlightControllerElement/3

```
public void HighlightControllerElement(GameObject element, Color? highlight,
float fadeDuration = 0f)
```

- Parameters
  - `GameObject element` - The element of the controller to apply the highlight to.
  - `Color? highlight` - The colour of the highlight.
  - `float fadeDuration = 0f` - The duration of fade from white to the highlight colour. Optional parameter defaults to `0f`.
- Returns
  - *none*

The HighlightControllerElement method allows for an element of the controller to have it's colour changed to simulate a highlighting effect of that element on the controller. It's useful for being able to draw a user's attention to a specific button on the controller.

UnhighlightControllerElement/1

```
public void UnhighlightControllerElement(GameObject element)
```

- Parameters
  - `GameObject element` - The element of the controller to remove the highlight from.
- Returns
  - *none*

The UnhighlightControllerElement method is the inverse of the HighlightControllerElement method and resets the controller element to it's original colour.

ToggleHighlightControllerElement/4

```
public void ToggleHighlightControllerElement(bool state, GameObject element,
Color? highlight = null, float duration = 0f)
```

- Parameters
  - `bool state` - The highlight colour state, `true` will enable the highlight on the given element and `false` will remove the highlight from the given element.
  - `GameObject element` - The element of the controller to apply the highlight to.
  - `Color? highlight` - The colour of the highlight.
  - `float fadeDuration = 0f` - The duration of fade from white to the highlight colour. Optional parameter defaults to `0f`.
- Returns
  - *none*

The ToggleHighlightControllerElement method is a shortcut method that makes it easier to highlight and unhighlight a controller element in a single method rather than using the HighlightControllerElement and UnhighlightControllerElement methods seperately.

ToggleHighlightTrigger/3

```
public void ToggleHighlightTrigger(bool state, Color? highlight = null,
float duration = 0f)
```

- Parameters
  - `bool state` - The highlight colour state, `true` will enable the highlight on the trigger and `false` will remove the highlight from the trigger.
  - `Color? highlight` - The colour to highlight the trigger with.

    - `float fadeDuration = 0f` - The duration of fade from white to the highlight colour. Optional parameter defaults to `0f`.
- Returns
    - *none*

The ToggleHighlightTrigger method is a shortcut method that makes it easier to toggle the highlight state of the controller trigger element.

ToggleHighlightGrip/3

```
public void ToggleHighlightGrip(bool state, Color? highlight = null, float
duration = 0f)
```

- Parameters
    - `bool state` - The highlight colour state, `true` will enable the highlight on the grip and `false` will remove the highlight from the grip.
    - `Color? highlight` - The colour to highlight the grip with.
    - `float fadeDuration = 0f` - The duration of fade from white to the highlight colour. Optional parameter defaults to `0f`.
- Returns
    - *none*

The ToggleHighlightGrip method is a shortcut method that makes it easier to toggle the highlight state of the controller grip element.

ToggleHighlightTouchpad/3

```
public void ToggleHighlightTouchpad(bool state, Color? highlight = null,
float duration = 0f)
```

- Parameters
    - `bool state` - The highlight colour state, `true` will enable the highlight on the touchpad and `false` will remove the highlight from the touchpad.
    - `Color? highlight` - The colour to highlight the touchpad with.
    - `float fadeDuration = 0f` - The duration of fade from white to the highlight colour. Optional parameter defaults to `0f`.
- Returns
    - *none*

The ToggleHighlightTouchpad method is a shortcut method that makes it easier to toggle the highlight state of the controller touchpad element.

ToggleHighlightApplicationMenu/3

```
public void ToggleHighlightApplicationMenu(bool state, Color? highlight =
null, float duration = 0f)
```

- Parameters
  - `bool state` - The highlight colour state, `true` will enable the highlight on the application menu and `false` will remove the highlight from the application menu.
  - `Color? highlight` - The colour to highlight the application menu with.
  - `float fadeDuration = 0f` - The duration of fade from white to the highlight colour. Optional parameter defaults to `0f`.
- Returns
  - *none*

The ToggleHighlightApplicationMenu method is a shortcut method that makes it easier to toggle the highlight state of the controller application menu element.

ToggleHighlightController/3

```
public void ToggleHighlightController(bool state, Color? highlight = null,
float duration = 0f)
```

- Parameters
  - `bool state` - The highlight colour state, `true` will enable the highlight on the entire controller `false` will remove the highlight from the entire controller.
  - `Color? highlight` - The colour to highlight the entire controller with.
  - `float fadeDuration = 0f` - The duration of fade from white to the highlight colour. Optional parameter defaults to `0f`.
- Returns
  - *none*

The ToggleHighlightController method is a shortcut method that makes it easier to toggle the highlight state of the entire controller.

TriggerHapticPulse/1

```
public void TriggerHapticPulse(ushort strength)
```

- Parameters
  - `ushort strength` - The intensity of the rumble of the controller motor. `0` to `3999`.
- Returns
  - *none*

The TriggerHapticPulse/1 method calls a single haptic pulse call on the controller for a single tick.

TriggerHapticPulse/3

```
public void TriggerHapticPulse(ushort strength, float duration, float
pulseInterval)
```

- Parameters
  - `ushort strength` - The intensity of the rumble of the controller motor. `0` to `3999`.
  - `float duration` - The length of time the rumble should continue for.
  - `float pulseInterval` - The interval to wait between each haptic pulse.
- Returns
  - *none*

The TriggerHapticPulse/3 method calls a haptic pulse for a specified amount of time rather than just a single tick. Each pulse can be separated by providing a `pulseInterval` to pause between each haptic pulse.

Example

`SteamVR_Unity_Toolkit/Examples/016_Controller_HapticRumble` demonstrates the ability to hide a controller model and make the controller vibrate for a given length of time at a given intensity.

`SteamVR_Unity_Toolkit/Examples/035_Controller_OpacityAndHighlighting`demonstrates the ability to change the opacity of a controller model and to highlight specific elements of a controller such as the buttons or even the entire controller model.

Device Finder (VRTK_DeviceFinder)

Overview

The Device Finder offers a collection of static methods that can be called to find common game devices such as the headset or controllers, or used to determine key information about the connected devices.

Class Methods

ControllerByIndex/1

```
public static SteamVR_TrackedObject ControllerByIndex(uint index)
```

- Parameters
  - `uint index` - The index of the tracked object to find. Must be of type `ETrackedDeviceClass.Controller`.

- Returns
  - `SteamVR_TrackedObject` - The object that matches the given index.

The ControllerByIndex method is used to find a SteamVR_TrackedObject by it's generated index. This is useful for finding controllers when only the index is known.

GetControllerIndex/1

```
public static uint GetControllerIndex(GameObject controller)
```

- Parameters
  - `GameObject Controller` - The controller object to check the index on.
- Returns
  - `uint` - The index of the given controller.

The GetControllerIndex method is used to find the index of a given controller object.

TrackedObjectByIndex/1

```
public static SteamVR_TrackedObject TrackedObjectByIndex(uint
controllerIndex)
```

- Parameters
  - `uint index` - The index of the tracked object to find.
- Returns
  - `SteamVR_TrackedObject` - The object that matches the given index.

The TrackedObjectByIndex method is used to find a SteamVR_TrackedObject by it's generated index. This method will loop over all SteamVR_TrackedObjects in the scene until the relevant index is found so it is less efficient than using `ControllerByIndex/1` to find a controller, but is useful for finding other tracked objects.

GetControllerHandType/1

```
public static ControllerHand GetControllerHandType(string hand)
```

- Parameters
  - `string hand` - The string representation of the hand to retrieve the type of. `left` or `right`.
- Returns
  - `ControllerHand` - An enum representing either the Left or Right hand.

The GetControllerHandType method is used for getting the enum representation of ControllerHand from a given string.

GetControllerHand/1

```
public static ControllerHand GetControllerHand(GameObject controller)
```

- Parameters
  - `GameObject controller` - The controller game object to check the hand of.
- Returns
  - `ControllerHand` - An enum representing either the Left or Right hand.

The GetControllerHand method is used for getting the enum representation of ControllerHand for the given controller game object.

IsControllerOfHand/2

```
public static bool IsControllerOfHand(GameObject checkController,
ControllerHand hand)
```

- Parameters
  - `GameObject checkController` - The actual controller object that is being checked.
  - `ControllerHand hand` - The enum representation of a hand to check if the given controller matches.
- Returns
  - `bool` - Is true if the given controller matches the given hand.

The IsControllerOfHand method is used to check if a given controller game object is of the hand type provided.

HeadsetTransform/0

```
public static Transform HeadsetTransform()
```

- Parameters
  - *none*
- Returns
  - `Transform` - The transform of the VR Headset component.

The HeadsetTransform method is used to retrieve the transform for the VR Headset in the scene. It can be useful to determine the position of the user's head in the game world.

HeadsetCamera/0

```
public static Transform HeadsetCamera()
```

- Parameters
  - *none*
- Returns
  - `Transform` - The transform of the VR Camera component.

The HeadsetCamera method is used to retrieve the transform for the VR Camera in the scene.

Simple Pointer (VRTK_SimplePointer)

> extends VRTK_WorldPointer

Overview

The Simple Pointer emits a coloured beam from the end of the controller to simulate a laser beam. It can be useful for pointing to objects within a scene and it can also determine the object it is pointing at and the distance the object is from the controller the beam is being emitted from.

The laser beam is activated by default by pressing the `Touchpad` on the controller. The event it is listening for is the `AliasPointer` events so the pointer toggle button can be set by changing the `Pointer Toggle` button on the `VRTK_ControllerEvents` script parameters.

The Simple Pointer script can be attached to a Controller object within the `[CameraRig]` prefab and the Controller object also requires the `VRTK_ControllerEvents` script to be attached as it uses this for listening to the controller button events for enabling and disabling the beam. It is also possible to attach the Simple Pointer script to another object (like the `[CameraRig]/Camera (head)`) to enable other objects to project the beam. The controller parameter must be entered with the desired controller to toggle the beam if this is the case.

Inspector Parameters

- **Enable Teleport:** If this is checked then the teleport flag is set to true in the Destination Set event so teleport scripts will know whether to action the new destination. This allows controller beams to be enabled on a controller but never trigger a teleport (if this option is unchecked).
- **Controller:** The controller that will be used to toggle the pointer. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.
- **Pointer Material:** The material to use on the rendered version of the pointer. If no material is selected then the default `WorldPointer` material will be used.
- **Pointer Hit Color:** The colour of the beam when it is colliding with a valid target. It can be set to a different colour for each controller.
- **Pointer Miss Color:** The colour of the beam when it is not hitting a valid target. It can be set to a different colour for each controller.

- **Show Play Area Cursor:** If this is enabled then the play area boundaries are displayed at the tip of the pointer beam in the current pointer colour.
- **Play Area Cursor Dimensions:** Determines the size of the play area cursor and collider. If the values are left as zero then the Play Area Cursor will be sized to the calibrated Play Area space.
- **Handle Play Area Cursor Collisions:** If this is ticked then if the play area cursor is colliding with any other object then the pointer colour will change to the `Pointer Miss Color` and the `WorldPointerDestinationSet` event will not be triggered, which will prevent teleporting into areas where the play area will collide.
- **Ignore Target With Tag Or Class:** A string that specifies an object Tag or the name of a Script attached to an obejct and notifies the play area cursor to ignore collisions with the object.
- **Pointer Visibility:** Determines when the pointer beam should be displayed:
  - `On_When_Active` only shows the pointer beam when the Pointer button on the controller is pressed.
  - `Always On` ensures the pointer beam is always visible but pressing the Pointer button on the controller initiates the destination set event.
  - `Always Off` ensures the pointer beam is never visible but the destination point is still set and pressing the Pointer button on the controller still initiates the destination set event.
- **Hold Button To Activate:** If this is checked then the pointer beam will be activated on first press of the pointer alias button and will stay active until the pointer alias button is pressed again. The destination set event is emitted when the beam is deactivated on the second button press.
- **Activate Delay:** The time in seconds to delay the pointer beam being able to be active again. Useful for preventing constant teleportation.
- **Pointer Thickness:** The thickness and length of the beam can also be set on the script as well as the ability to toggle the sphere beam tip that is displayed at the end of the beam (to represent a cursor).
- **Pointer Length:** The distance the beam will project before stopping.
- **Show Pointer Tip:** Toggle whether the cursor is shown on the end of the pointer beam.
- **Custom Pointer Cursor:** A custom Game Object can be applied here to use instead of the default sphere for the pointer cursor.
- **Layers To Ignore:** The layers to ignore when raycasting.

Example

`SteamVR_Unity_Toolkit/Examples/003_Controller_SimplePointer` shows the simple pointer in action and code examples of how the events are utilised and listened to can be viewed in the script `SteamVR_Unity_Toolkit/Examples/Resources/Scripts/VRTK_ControllerPointerEvents_ListenerExample.cs`

Bezier Pointer (VRTK_BezierPointer)

> extends VRTK_WorldPointer

Overview

The Bezier Pointer emits a curved line (made out of game objects) from the end of the controller to a point on a ground surface (at any height). It is more useful than the Simple Laser Pointer for traversing objects of various heights as the end point can be curved on top of objects that are not visible to the user.

The laser beam is activated by default by pressing the `Touchpad` on the controller. The event it is listening for is the `AliasPointer` events so the pointer toggle button can be set by changing the `Pointer Toggle` button on the `VRTK_ControllerEvents` script parameters.

The Bezier Pointer script can be attached to a Controller object within the `[CameraRig]` prefab and the Controller object also requires the `VRTK_ControllerEvents` script to be attached as it uses this for listening to the controller button events for enabling and disabling the beam. It is also possible to attach the Bezier Pointer script to another object (like the `[CameraRig]/Camera (head)`) to enable other objects to project the beam. The controller parameter must be entered with the desired controller to toggle the beam if this is the case.

> The bezier curve generation code is in another script located at `SteamVR_Unity_Toolkit/Scripts/Helper/CurveGenerator.cs` and was heavily inspired by the tutorial and code from Catlike Coding.

Inspector Parameters

- **Enable Teleport:** If this is checked then the teleport flag is set to true in the Destination Set event so teleport scripts will know whether to action the new destination. This allows controller beams to be enabled on a controller but never trigger a teleport (if this option is unchecked).
- **Controller:** The controller that will be used to toggle the pointer. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.
- **Pointer Material:** The material to use on the rendered version of the pointer. If no material is selected then the default `WorldPointer` material will be used.
- **Pointer Hit Color:** The colour of the beam when it is colliding with a valid target. It can be set to a different colour for each controller.
- **Pointer Miss Color:** The colour of the beam when it is not hitting a valid target. It can be set to a different colour for each controller.
- **Show Play Area Cursor:** If this is enabled then the play area boundaries are displayed at the tip of the pointer beam in the current pointer colour.
- **Play Area Cursor Dimensions:** Determines the size of the play area cursor and collider. If the values are left as zero then the Play Area Cursor will be sized to the calibrated Play Area space.
- **Handle Play Area Cursor Collisions:** If this is ticked then if the play area cursor is colliding with any other object then the pointer colour will change to the `Pointer Miss Color` and the `WorldPointerDestinationSet` event will not be triggered, which will prevent teleporting into areas

where the play area will collide.
- **Ignore Target With Tag Or Class:** A string that specifies an object Tag or the name of a Script attached to an obejct and notifies the play area cursor to ignore collisions with the object.
- **Pointer Visibility:** Determines when the pointer beam should be displayed:
  - `On_When_Active` only shows the pointer beam when the Pointer button on the controller is pressed.
  - `Always On` ensures the pointer beam is always visible but pressing the Pointer button on the controller initiates the destination set event.
  - `Always Off` ensures the pointer beam is never visible but the destination point is still set and pressing the Pointer button on the controller still initiates the destination set event.
- **Hold Button To Activate:** If this is checked then the pointer beam will be activated on first press of the pointer alias button and will stay active until the pointer alias button is pressed again. The destination set event is emitted when the beam is deactivated on the second button press.
- **Activate Delay:** The time in seconds to delay the pointer beam being able to be active again. Useful for preventing constant teleportation.
- **Pointer Length:** The length of the projected forward pointer beam, this is basically the distance able to point from the controller potiion.
- **Pointer Density:** The number of items to render in the beam bezier curve. A high number here will most likely have a negative impact of game performance due to large number of rendered objects.
- **Show Pointer Cursor:** A cursor is displayed on the ground at the location the beam ends at, it is useful to see what height the beam end location is, however it can be turned off by toggling this.
- **Pointer Cursor Radius:** The size of the ground pointer cursor. This number also affects the size of the objects in the bezier curve beam. The larger the raduis, the larger the objects will be.
- **Beam Curve Offset:** The amount of height offset to apply to the projected beam to generate a smoother curve even when the beam is pointing straight.
- **Custom Pointer Tracer:** A custom Game Object can be applied here to use instead of the default sphere for the beam tracer. The custom Game Object will match the rotation of the controller.
- **Custom Pointer Cursor:** A custom Game Object can be applied here to use instead of the default flat cylinder for the pointer cursor.
- **Layers To Ignore:** The layers to ignore when raycasting.
- **Valid Teleport Location Object:** A custom Game Object can be applied here to appear only if the teleport is allowed (its material will not be changed).
- **Rescale Pointer Tracer:** Rescale each pointer tracer element according to the length of the Bezier curve.

Example

`SteamVR_Unity_Toolkit/Examples/009_Controller_BezierPointer` is used in conjunction with the Height Adjust Teleporter shows how it is possible to traverse different height objects using the curved pointer without needing to see the top of the object.

`SteamVR_Unity_Toolkit/Examples/012_Controller_PointerWithAreaCollision` shows how a Bezier Pointer with the Play Area Cursor and Collision Detection enabled can be used to traverse a game area but not allow teleporting into areas where the walls or other objects would fall into the play area space enabling the user to enter walls.

`SteamVR_Unity_Toolkit/Examples/036_Controller_CustomCompoundPointer' shows how to display an object (a teleport beam) only if the teleport location is valid, and can create an animated trail along the tracer curve.

Unity UI Pointer (VRTK_UIPointer)

Overview

The UI Pointer provides a mechanism for interacting with Unity UI elements on a world canvas. The UI Pointer can be attached to any game object the same way in which a World Pointer can be and the UI Pointer also requires a controller to initiate the pointer activation and pointer click states.

It's possible to prevent a world canvas from being interactable with a UI Pointer by setting a tag or applying a class to the canvas and then entering the tag or class name for the UI Pointer to ignore on the UI Pointer inspector parameters.

The simplest way to use the UI Pointer is to attach the script to a game controller within the `[CameraRig]` along with a Simple Pointer as this provides visual feedback as to where the UI ray is pointing.

The UI pointer is activated via the `Pointer` alias on the `Controller Events` and the UI pointer click state is triggered via the `UI Click` alias on the `Controller Events`.

Inspector Parameters

- **Controller:** The controller that will be used to toggle the pointer. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.
- **Ignore Canvas With Tag Or Class:** A string that specifies a canvas Tag or the name of a Script attached to a canvas and denotes that any world canvases that contain this tag or script will be ignored by the UI Pointer.
- **Activation Mode:** Determines when the UI pointer should be active:
  - `Hold_Button` only activates the UI Pointer when the Pointer button on the controller is pressed and held down.
  - `Toggle_Button` activates the UI Pointer on the first click of the Pointer button on the controller and it stays active until the Pointer button is clicked again.
  - `Always_On` the UI Pointer is always active regardless of whether the Pointer button on the controller is pressed or not.

Class Events

- `UIPointerElementEnter` - Emitted when the UI Pointer is colliding with a valid UI element.
- `UIPointerElementExit` - Emitted when the UI Pointer is no longer colliding with any valid UI elements.

Event Payload

- `uint controllerIndex` - The index of the controller that was used.
- `GameObject currentTarget` - The current UI element that the pointer is colliding with.
- `GameObject previousTarget` - The previous UI element that the pointer was colliding with.

Class Methods

SetEventSystem/1

```
public VRTK_EventSystemVRInput SetEventSystem(EventSystem eventSystem)
```

- Parameters
  - `EventSystem eventSystem` - The global Unity event system to be used by the UI pointers.
- Returns
  - `VRTK_EventSystemVRInput` - A custom event system input class that is used to detect input from VR pointers.

The SetEventSystem method is used to set up the global Unity event system for the UI pointer. It also handles disabling the existing Standalone Input Module that exists on the EventSystem and adds a custom VRTK Event System VR Input component that is required for interacting with the UI with VR inputs.

SetWorldCanvas/1

```
public void SetWorldCanvas(Canvas canvas)
```

- Parameters
  - `Canvas canvas` - The canvas object to initialse for use with the UI pointers. Must be of type `WorldSpace`.
- Returns
  - *none*

The SetWorldCanvas method is used to initialise a `WorldSpace` canvas for use with the UI Pointer. This method is called automatically on start for all editor created canvases but would need to be manually called if a canvas was generated at runtime.

PointerActive/0

```
public bool PointerActive()
```

- Parameters
  - *none*
- Returns
  - `bool` - Returns true if the ui pointer should be currently active.

The PointerActive method determines if the ui pointer beam should be active based on whether the pointer alias is being held and whether the Hold Button To Use parameter is checked.

Example

`SteamVR_Unity_Toolkit/Examples/034_Controls_InteractingWithUnityUI` uses the `VRTK_UIPointer` script on the right Controller to allow for the interaction with Unity UI elements using a Simple Pointer beam. The left Controller controls a Simple Pointer on the headset to demonstrate gaze interaction with Unity UI elements.

Basic Teleporter (VRTK_BasicTeleport)

Overview

The basic teleporter updates the `[CameraRig]` x/z position in the game world to the position of a World Pointer's tip location which is set via the `WorldPointerDestinationSet` event. The y position is never altered so the basic teleporter cannot be used to move up and down game objects as it only allows for travel across a flat plane.

The Basic Teleport script is attached to the `[CameraRig]` prefab and requires an implementation of the WorldPointer script to be attached to another game object (e.g. VRTK_SimplePointer attached to the Controller object).

Inspector Parameters

- **Blink Transition Speed:** The fade blink speed can be changed on the basic teleport script to provide a customised teleport experience. Setting the speed to 0 will mean no fade blink effect is present. The fade is achieved via the `SteamVR_Fade.cs` script in the SteamVR Unity Plugin scripts.
- **Distance Blink Delay:** A range between 0 and 32 that determines how long the blink transition will stay blacked out depending on the distance being teleported. A value of 0 will not delay the teleport blink effect over any distance, a value of 32 will delay the teleport blink fade in even when the distance teleported is very close to the original position. This can be used to simulate time taking longer to pass the further a user teleports. A value of 16 provides a decent basis to simulate this to the user.
- **Headset Position Compensation:** If this is checked then the teleported location will be the position of the headset within the play area. If it is unchecked then the teleported location will always be the centre of the play area even if the headset position is not in the centre of the play area.
- **Ignore Target With Tag Or Class:** A string that specifies an object Tag or the name of a Script

attached to an obejct and notifies the teleporter that the destination is to be ignored so the user cannot teleport to that location. It also ensure the pointer colour is set to the miss colour.
- **Nav Mesh Limit Distance:** The max distance the nav mesh edge can be from the teleport destination to be considered valid. If a value of `0` is given then the nav mesh restriction will be ignored..

Class Events

- `Teleporting` - Emitted when the teleport process has begun.
- `Teleported` - Emitted when the teleport process has successfully completed.

Event Payload

> The event payload is reused from the VRTK_DestinationMarker Event Payload

Class Methods

InitDestinationSetListener/1

```
public void InitDestinationSetListener(GameObject markerMaker)
```

- Parameters
  - `GameObject markerMaker` - The game object that is used to generate destination marker events, such as a controller.
- Returns
  - *none*

The InitDestinationSetListener method is used to register the teleport script to listen to events from the given game object that is used to generate destination markers. Any destination set event emitted by a registered game object will initiate the teleport to the given destination location.

Example

`SteamVR_Unity_Toolkit/Examples/004_CameraRig_BasicTeleport` uses the `VRTK_SimplePointer` script on the Controllers to initiate a laser pointer by pressing the `Touchpad` on the controller and when the laser pointer is deactivated (release the `Touchpad`) then the user is teleported to the location of the laser pointer tip as this is where the pointer destination marker position is set to.

Height Adjustable Teleporter (VRTK_HeightAdjustTeleport)

> extends VRTK_BasicTeleport

Overview

The height adjust teleporter extends the basic teleporter and allows for the y position of the `[CameraRig]` to be altered based on whether the teleport location is on top of another object.

Like the basic teleporter the Height Adjust Teleport script is attached to the `[CameraRig]` prefab and requires a World Pointer to be available.

Inspector Parameters

- **Blink Transition Speed:** The fade blink speed on teleport.
- **Distance Blink Delay:** A range between 0 and 32 that determines how long the blink transition will stay blacked out depending on the distance being teleported. A value of 0 will not delay the teleport blink effect over any distance, a value of 32 will delay the teleport blink fade in even when the distance teleported is very close to the original position. This can be used to simulate time taking longer to pass the further a user teleports. A value of 16 provides a decent basis to simulate this to the user.
- **Headset Position Compensation:** If this is checked then the teleported location will be the position of the headset within the play area. If it is unchecked then the teleported location will always be the centre of the play area even if the headset position is not in the centre of the play area.
- **Ignore Target With Tag Or Class:** A string that specifies an object Tag or the name of a Script attached to an obejct and notifies the teleporter that the destination is to be ignored so the user cannot teleport to that location. It also ensure the pointer colour is set to the miss colour.
- **Nav Mesh Limit Distance:** The max distance the nav mesh edge can be from the teleport destination to be considered valid. If a value of `0` is given then the nav mesh restriction will be ignored..
- **Play Space Falling:** Checks if the user steps off an object into a part of their play area that is not on the object then they are automatically teleported down to the nearest floor. The `Play Space Falling` option also works in the opposite way that if the user's headset is above an object then the user is teleported automatically on top of that object, which is useful for simulating climbing stairs without needing to use the pointer beam location. If this option is turned off then the user can hover in mid air at the same y position of the object they are standing on.
- **Use Gravity**: allows for gravity based falling when the distance is greater than `Gravity Fall Height`.
- **Gravity Fall Height**: Fall distance needed before gravity based falling can be triggered.
- **Blink Y Threshold:** The `y` distance between the floor and the headset that must change before the fade transition is initiated. If the new user location is at a higher distance than the threshold then the headset blink transition will activate on teleport. If the new user location is within the threshold then no blink transition will happen, which is useful for walking up slopes, meshes and terrains where constant blinking would be annoying.

Example

`SteamVR_Unity_Toolkit/Examples/007_CameraRig_HeightAdjustTeleport` has a collection of varying height objects that the user can either walk up and down or use the laser pointer to climb on top of them.

`SteamVR_Unity_Toolkit/Examples/010_CameraRig_TerrainTeleporting` shows how the teleportation of a user can also traverse terrain colliders.

`SteamVR_Unity_Toolkit/Examples/020_CameraRig_MeshTeleporting` shows how the teleportation of a user can also traverse mesh colliders.

Headset Collision Fade (VRTK_HeadsetCollisionFade)

Overview

The purpose of the Headset Collision Fade is to detect when the user's VR headset collides with another game object and fades the screen to a solid colour. This is to deal with a user putting their head into a game object and seeing the inside of the object clipping, which is an undesired effect.

The reasoning behind this is if the user puts their head where it shouldn't be, then fading to a colour (e.g. black) will make the user realise they've done something wrong and they'll probably naturally step backwards.

If the headset is colliding then the teleport action is also disabled to prevent cheating by clipping through walls.

> **Unity Version Information** * If using `Unity 5.3` or older then the Headset Collision Fade script is attached to the `Camera (head)` object within the `[CameraRig]` prefab. * If using `Unity 5.4` or newer then the Headset Collision Fade script is attached to the `Camera (eye)` object within the `[CameraRig]->Camera (head)` prefab.

Inspector Parameters

- **Blink Transition Speed:** The fade blink speed on collision.
- **Fade Color:** The colour to fade the headset to on collision.
- **Ignore Target With Tag Or Class:** A string that specifies an object Tag or the name of a Script attached to an obejct and will prevent the object from fading the headset view on collision.

Class Events

- `HeadsetCollisionDetect` - Emitted when the user's headset collides with another game object.
- `HeadsetCollisionEnded` - Emitted when the user's headset stops colliding with a game object.

Event Payload

- `Collider collider` - The Collider of the game object the headset has collided with.

- `Transform currentTransform` - The current Transform of the object that the Headset Collision Fade script is attached to (Camera).

`SteamVR_Unity_Toolkit/Examples/011_Camera_HeadSetCollisionFading` has collidable walls around the play area and if the user puts their head into any of the walls then the headset will fade to black.

Player Presence (VRTK_PlayerPresence)

Overview

The concept that the VR user has a physical in game presence which is accomplished by adding a collider and a rigidbody at the position the user is standing within their play area. This physical collider and rigidbody will prevent the user from ever being able to walk through walls or intersect other collidable objects. The height of the collider is determined by the height the user has the headset at, so if the user crouches then the collider shrinks with them, meaning it's possible to crouch and crawl under low ceilings.

Inspector Parameters

- **Headset Y Offset:** The box collider which is created for the user is set at a height from the user's headset position. If the collider is required to be lower to allow for room between the play area collider and the headset then this offset value will shorten the height of the generated box collider.
- **Ignore Grabbed Collisions:** If this is checked then any items that are grabbed with the controller will not collide with the box collider and rigid body on the play area. This is very useful if the user is required to grab and wield objects because if the collider was active they would bounce off the play area collider.
- **Reset Position On Collision:** If this is checked then if the Headset Collision Fade script is present and a headset collision occurs, the Camera Rig is moved back to the last good known standing position. This deals with any collision issues if a user stands up whilst moving through a crouched area as instead of them being able to clip into objects they are transported back to a position where they are able to stand.
- **Falling Physics Only:** Only use physics when an explicit falling state is set.

Class Events

- `PresenceFallStarted` - Emitted when a gravity based fall has started
- `PresenceFallEnded` - Emitted when a gravity based fall has ended

Event Payload

- `float fallDistance` - The total height the player has dropped from a gravity based fall.

Class Methods

SetFallingPhysicsOnlyParams/1

```
public void SetFallingPhysicsOnlyParams(bool falling)
```

- Parameters
  - `bool falling` - Toggle the physics falling on or off.
- Returns
  - *none*

The SetFallingPhysicsOnlyParams method will toggle the `fallingPhysicsOnly` class state as well as enable or disable physics if needed.

IsFalling/0

```
public bool IsFalling()
```

- Parameters
  - *none*
- Returns
  - `bool` - Returns if the player is in a physics falling state or not.

The IsFalling method will return if the class is using physics based falling and is currently in a falling state.

GetHeadset/0

```
public Transform GetHeadset()
```

- Parameters
  - *none*
- Returns
  - `Transform` - The transform for the object representing the VR headset.

The GetHeadset method returns the transform of the object representing the VR headset in the game world.

StartPhysicsFall/1

```
public void StartPhysicsFall(Vector3 velocity)
```

- Parameters
  - `Vector3 velocity` - The starting velocity to use at the start of a fall.

- Returns
  - *none*

The StartPhysicsFall method initializes the physics based fall state, enable physics and send out the `PresenceFallStarted` event.

StopPhysicsFall/0

```
public void StopPhysicsFall()
```

- Parameters
  - *none*
- Returns
  - *none*

The StopPhysicsFall method ends the physics based fall state, disables physics and send out the `PresenceFallEnded` event.

Example

`SteamVR_Unity_Toolkit/Examples/017_CameraRig_TouchpadWalking` has a collection of walls and slopes that can be traversed by the user with the touchpad but the user cannot pass through the objects as they are collidable and the rigidbody physics won't allow the intersection to occur.

Touchpad Movement (VRTK_TouchpadWalking)

Overview

The ability to move the play area around the game world by sliding a finger over the touchpad is achieved using this script. The Touchpad Walking script is applied to the `[CameraRig]` prefab and adds a rigidbody and a box collider to the user's position to prevent them from walking through other collidable game objects.

If the Headset Collision Fade script has been applied to the Camera prefab, then if a user attempts to collide with an object then their position is reset to the last good known position. This can happen if the user is moving through a section where they need to crouch and then they stand up and collide with the ceiling. Rather than allow a user to do this and cause collision resolution issues it is better to just move them back to a valid location. This does break immersion but the user is doing something that isn't natural.

Inspector Parameters

- **Left Controller:** If this is checked then the left controller touchpad will be enabled to move the play area. It can also be toggled at runtime.

- **Right Controller:** If this is checked then the right controller touchpad will be enabled to move the play area. It can also be toggled at runtime.
- **Max Walk Speed:** The maximum speed the play area will be moved when the touchpad is being touched at the extremes of the axis. If a lower part of the touchpad axis is touched (nearer the centre) then the walk speed is slower.
- **Deceleration:** The speed in which the play area slows down to a complete stop when the user is no longer touching the touchpad. This deceleration effect can ease any motion sickness that may be suffered.

Example

`SteamVR_Unity_Toolkit/Examples/017_CameraRig_TouchpadWalking` has a collection of walls and slopes that can be traversed by the user with the touchpad. There is also an area that can only be traversed if the user is crouching. Standing up in this crouched area will cause the user to appear back at their last good known position.

Play Space Extension (VRTK_RoomExtender)

Overview

This script allows the playArea to move with the user. The `[CameraRig]` is only moved when at the edge of a defined circle. Aims to create a virtually bigger play area. To use this add this script to the `[CameraRig`] prefab.

There is an additional script `VRTK_RoomExtender_PlayAreaGizmo` which can be attachted to the `[CameraRig`] to visualize the extended playArea within the Editor.

Inspector Parameters

- **Movement Function:** This determines the type of movement used by the extender and can be one of the following options:
  - **Nonlinear:** Moves the head with a non linear drift movement.
  - **LinearDirect:** Moves the headset in a direct linear movement.
- **Additional Movement Enabled:** This is the a public variable to enable the additional movement. This can be used in other scripts to toggle the `[CameraRig]` movement.
- **Additional Movement Enabled On Button Press:** This configures the controls of the RoomExtender. If this is true then the touchpad needs to be pressed to enable it. If this is false then it is disabled by pressing the touchpad.
- **Additional Movement Multiplier:** This is the factor by which movement at the edge of the circle is amplified. 0 is no movement of the `[CameraRig]`. Higher values simulate a bigger play area but may be too uncomfortable.
- **Head Zone Radius:** This is the size of the circle in which the playArea is not moved and everything is normal. If it is to low it becomes uncomfortable when crouching.

- **Debug Transform:** This transform visualises the circle around the user where the `[CameraRig]` is not moved. In the demo scene this is a cylinder at floor level. Remember to turn of collisions.

Example

`SteamVR_Unity_Toolkit/Examples/028_CameraRig_RoomExtender` shows how the RoomExtender script is controlled by a VRTK_RoomExtender_Controller Example script located at both controllers. Pressing the `Touchpad` on the controller activates the Room Extender. The Additional Movement Multiplier is changed based on the touch distance to the center of the touchpad.

Interactable Object (VRTK_InteractableObject)

Overview

The Interactable Object script is attached to any game object that is required to be interacted with (e.g. via the controllers).

The basis of this script is to provide a simple mechanism for identifying objects in the game world that can be grabbed or used but it is expected that this script is the base to be inherited into a script with richer functionality.

Inspector Parameters

Touch Interactions

- **Highlight On Touch:** The object will only highlight when a controller touches it if this is checked.
- **Touch Highlight Color:** The colour to highlight the object when it is touched. This colour will override any globally set color (for instance on the `VRTK_InteractTouch` script).
- **Rumble On Touch:** The haptic feedback on the controller can be triggered upon touching the object, the `x` denotes the length of time, the `y` denotes the strength of the pulse. (x and y will be replaced in the future with a custom editor)
- **Allowed Touch Controllers:** Determines which controller can initiate a touch action. The options available are:
  - `Both` means both controllers will register a touch.
  - `Left_Only` means only the left controller will register a touch.
  - `Right_Only` means only the right controller will register a touch.
- **Hide Controller On Touch:** Optionally override the controller setting (hide when touched):
  - `Default` means using controller settings.
  - `Override Hide` means hiding the controller when touched, overriding controller settings.
  - `Override Dont Hide` means *not* hiding the controller when touched, overriding controller settings.

Grab Interactions

- **Is Grabbable:** Determines if the object can be grabbed.

- **Is Droppable:** Determines if the object can be dropped by the controller grab button being used. If this is unchecked then it's not possible to drop the item once it's picked up using the controller button. It is still possible for the item to be dropped if the Grab Attach Mechanic is a joint and too much force is applied to the object and the joint is broken. To prevent this it's better to use the Child Of Controller mechanic.
- **Is Swappable:** Determines if the object can be swapped between controllers when it is picked up. If it is unchecked then the object must be dropped before it can be picked up by the other controller.
- **Hold Button To Grab:** If this is checked then the grab button on the controller needs to be continually held down to keep grabbing. If this is unchecked the grab button toggles the grab action with one button press to grab and another to release.
- **Rumble On Grab:** The haptic feedback on the controller can be triggered upon grabbing the object, the `x` denotes the length of time, the `y` denotes the strength of the pulse. (x and y will be replaced in the future with a custom editor).
- **Allowed Grab Controllers:** Determines which controller can initiate a grab action. The options available are:
  - `Both` means both controllers are allowed to grab.
  - `Left_Only` means only the left controller is allowed to grab.
  - `Right_Only` means only the right controller is allowed to grab.
- **Precision_Snap:** If this is checked then when the controller grabs the object, it will grab it with precision and pick it up at the particular point on the object the controller is touching.
- **Right Snap Handle:** A Transform provided as an empty game object which must be the child of the item being grabbed and serves as an orientation point to rotate and position the grabbed item in relation to the right handed controller. If no Right Snap Handle is provided but a Left Snap Handle is provided, then the Left Snap Handle will be used in place. If no Snap Handle is provided then the object will be grabbed at it's central point.
- **Left Snap Handle:** A Transform provided as an empty game object which must be the child of the item being grabbed and serves as an orientation point to rotate and position the grabbed item in relation to the left handed controller. If no Left Snap Handle is provided but a Right Snap Handle is provided, then the Right Snap Handle will be used in place. If no Snap Handle is provided then the object will be grabbed at it's central point.
- **Hide Controller On Grab:** Optionally override the controller setting (hide when grabbed):
  - `Default` means using controller settings.
  - `Override Hide` means hiding the controller when grabbed, overriding controller settings.
  - `Override Dont Hide` means *not* hiding the controller when grabbed, overriding controller settings.

Grab Mechanics

- **Grab Attach Type:** This determines how the grabbed item will be attached to the controller when it is grabbed.
  - `Fixed Joint` attaches the object to the controller with a fixed joint meaning it tracks the position and rotation of the controller with perfect 1:1 tracking.
  - `Spring Joint` attaches the object to the controller with a spring joint meaing there is some

  flexibility between the item and the controller force moving the item. This works well when attempting to pull an item rather than snap the item directly to the controller. It creates the illusion that the item has resistence to move it.
  - `Track Object` doesn't attach the object to the controller via a joint, instead it ensures the object tracks the direction of the controller, which works well for items that are on hinged joints.
  - `Rotator Track` also tracks the object but instead of the object tracking the direction of the controller, a force is applied to the object to cause it to rotate. This is ideal for hinged joints on items such as wheels or doors.
  - `Child Of Controller` simply makes the object a child of the controller grabbing so it naturally tracks the position of the controller motion.
  - `Climbable` non-rigid body interactable object used to allow player climbing.
- **Detach Threshold:** The force amount when to detach the object from the grabbed controller. If the controller tries to exert a force higher than this threshold on the object (from pulling it through another object or pushing it into another object) then the joint holding the object to the grabbing controller will break and the object will no longer be grabbed. This also works with Tracked Object grabbing but determines how far the controller is from the object before breaking the grab.
- **Spring Joint Strength:** The strength of the spring holding the object to the controller. A low number will mean the spring is very loose and the object will require more force to move it, a high number will mean a tight spring meaning less force is required to move it.
- **Spring Joint Damper:** The amount to damper the spring effect when using a Spring Joint grab mechanic. A higher number here will reduce the oscillation effect when moving jointed Interactable Objects.
- **Throw Multiplier:** An amount to multiply the velocity of the given object when it is thrown. This can also be used in conjunction with the Interact Grab Throw Multiplier to have certain objects be thrown even further than normal (or thrown a shorter distance if a number below 1 is entered).
- **On Grab Collision Delay:** The amount of time to delay collisions affecting the object when it is first grabbed. This is useful if a game object may get stuck inside another object when it is being grabbed.

Use Interactions

- **Is Usable:** Determines if the object can be used.
- **Use Only If Grabbed:** If this is checked the object can be used only if it was grabbed before.
- **Hold Button To Use:** If this is checked then the use button on the controller needs to be continually held down to keep using. If this is unchecked the the use button toggles the use action with one button press to start using and another to stop using.
- **Pointer Activates Use Action:** If this is checked then when a World Pointer beam (projected from the controller) hits the interactable object, if the object has `Hold Button To Use` unchecked then whilst the pointer is over the object it will run it's `Using` method. If `Hold Button To Use` is unchecked then the `Using` method will be run when the pointer is deactivated. The world pointer will not throw the `Destination Set` event if it is affecting an interactable object with this setting checked as this prevents unwanted teleporting from happening when using an object with a

pointer.

- **Rumble On Use:** The haptic feedback on the controller can be triggered upon using the object, the `x` denotes the length of time, the `y` denotes the strength of the pulse. (x and y will be replaced in the future with a custom editor).
- **Allowed Use Controllers:** Determines which controller can initiate a use action. The options available are:
  - `Both` means both controllers are allowed to use.
  - `Left_Only` means only the left controller is allowed to use.
  - `Right_Only` means only the right controller is allowed to use.
- **Hide Controller On Use:** Optionally override the controller setting (hide when used):
  - `Default` means using controller settings.
  - `Override Hide` means hiding the controller when using, overriding controller settings.
  - `Override Dont Hide` means *not* hiding the controller when using, overriding controller settings.

Class Events

- `InteractableObjectTouched` - Emitted when another object touches the current object.
- `InteractableObjectUntouched` - Emitted when the other object stops touching the current object.
- `InteractableObjectGrabbed` - Emitted when another object grabs the current object (e.g. a controller).
- `InteractableObjectUngrabbed` - Emitted when the other object stops grabbing the current object.
- `InteractableObjectUsed` - Emitted when another object uses the current object (e.g. a controller).
- `InteractableObjectUnused` - Emitted when the other object stops using the current object.

Event Payload

- `GameObject interactingObject` - The object that is initiating the interaction (e.g. a controller)

Class Methods

CheckHideMode/2

```
public bool CheckHideMode(bool defaultMode, ControllerHideMode overrideMode)
```

- Parameters
  - `bool defaultMode` - The default setting of the controller (true=hide, false="don't hide").
  - `ControllerHideMode overrideMode` - The override setting of the object.
    - `Default` means using controller settings (returns `overrideMode`).
    - `OverrideHide` means hiding the controller anyway (even if `defaultMode`is `true`).
    - `OverrideDontHide` means *not* hiding the controller anyway (even if `defaultMode`is `false`).
- Returns
  - `bool` - Returns `true` if the combination of `defaultMode` and `overrideMode` lead to "hide controller".

The CheckHideMode method is a simple service method used only by some scripts (e.g.

InteractTouch InteractGrab InteractUse) to calculate the "hide controller" condition according to the default controller settings and the interactive object override method.

IsTouched/0

```
public bool IsTouched()
```

- Parameters
  - *none*
- Returns
  - `bool` - Returns `true` if the object is currently being touched.

The IsTouched method is used to determine if the object is currently being touched.

IsGrabbed/0

```
public bool IsGrabbed()
```

- Parameters
  - *none*
- Returns
  - `bool` - Returns `true` if the object is currently being grabbed.

The IsGrabbed method is used to determine if the object is currently being grabbed.

IsUsing/0

```
public bool IsUsing()
```

- Parameters
  - *none*
- Returns
  - `bool` - Returns `true` if the object is currently being used.

The IsUsing method is used to determine if the object is currently being used.

StartTouching/1

```
public virtual void StartTouching(GameObject currentTouchingObject)
```

- Parameters
  - `GameObject currentTouchingObject` - The game object that is currently touching this object.

- Returns
  - *none*

The StartTouching method is called automatically when the object is touched initially. It is also a virtual method to allow for overriding in inherited classes.

StopTouching/1

```
public virtual void StopTouching(GameObject previousTouchingObject)
```

- Parameters
  - `GameObject previousTouchingObject` - The game object that was previously touching this object.
- Returns
  - *none*

The StopTouching method is called automatically when the object has stopped being touched. It is also a virtual method to allow for overriding in inherited classes.

Grabbed/1

```
public virtual void Grabbed(GameObject currentGrabbingObject)
```

- Parameters
  - `GameObject currentGrabbingObject` - The game object that is currently grabbing this object.
- Returns
  - *none*

The Grabbed method is called automatically when the object is grabbed initially. It is also a virtual method to allow for overriding in inherited classes.

Ungrabbed/1

```
public virtual void Ungrabbed(GameObject previousGrabbingObject)
```

- Parameters
  - `GameObject previousGrabbingObject` - The game object that was previously grabbing this object.
- Returns
  - *none*

The Ungrabbed method is called automatically when the object has stopped being grabbed. It is also a virtual method to allow for overriding in inherited classes.

StartUsing/1

```
public virtual void StartUsing(GameObject currentUsingObject)
```

- Parameters
  - `GameObject currentUsingObject` - The game object that is currently using this object.
- Returns
  - *none*

The StartUsing method is called automatically when the object is used initially. It is also a virtual method to allow for overriding in inherited classes.

StopUsing/1

```
public virtual void StopUsing(GameObject previousUsingObject)
```

- Parameters
  - `GameObject previousUsingObject` - The game object that was previously using this object.
- Returns
  - *none*

The StopUsing method is called automatically when the object has stopped being used. It is also a virtual method to allow for overriding in inherited classes.

ToggleHighlight/1

```
public virtual void ToggleHighlight(bool toggle)
```

- Parameters
  - `bool toggle` - The state to determine whether to activate or deactivate the highlight. `true` will enable the highlight and `false` will remove the highlight.
- Returns
  - *none*

The ToggleHighlight/1 method is used as a shortcut to disable highlights whilst keeping the same method signature. It should always be used with `false` and it calls ToggleHighlight/2 with a `Color.clear`.

ToggleHighlight/2

```
public virtual void ToggleHighlight(bool toggle, Color globalHighlightColor)
```

- Parameters
  - `bool toggle` - The state to determine whether to activate or deactivate the highlight. `true` will enable the highlight and `false` will remove the highlight.
  - `Color globalHighlightColor` = The colour to use when highlighting the object.
- Returns
  - *none*

The ToggleHighlight/2 method is used to turn on or off the colour highlight of the object.

PauseCollisions/0

```
public void PauseCollisions()
```

- Parameters
  - *none*
- Returns
  - *none*

The PauseCollisions method temporarily pauses all collisions on the object at grab time by removing the object's rigidbody's ability to detect collisions. This can be useful for preventing clipping when initially grabbing an item.

AttachIsTrackObject/0

```
public bool AttachIsTrackObject()
```

- Parameters
  - *none*
- Returns
  - `bool` - Is true if the grab attach mechanic is one of the track types like `Track Object` or `Rotator Track`.

The AttachIsTrackObject method is used to determine if the object is using one of the track grab attach mechanics.

AttachIsClimbObject/0

```
public bool AttachIsClimbObject()
```

- Parameters
  - *none*
- Returns
  - `bool` - Is true if the grab attach mechanic is `Climbable`.

The AttachIsClimbObject method is used to determine if the object is using the `Climbable` grab attach mechanics.

AttachIsStaticObject/0

```
public bool AttachIsStaticObject()
```

- Parameters
  - *none*
- Returns
  - `bool` - Is true if the grab attach mechanic is one of the static types like `Climbable`.

The AttachIsStaticObject method is used to determine if the object is using one of the static grab attach types.

ZeroVelocity/0

```
public void ZeroVelocity()
```

- Parameters
  - *none*
- Returns
  - *none*

The ZeroVelocity method resets the velocity and angular velocity to zero on the rigidbody attached to the object.

SaveCurrentState/0

```
public void SaveCurrentState()
```

- Parameters
  - *none*
- Returns
  - *none*

The SaveCurrentState method stores the existing object parent and the object's rigidbody kinematic setting.

ToggleKinematic/1

```
public void ToggleKinematic(bool state)
```

- Parameters
  - `bool state` - The object's rigidbody kinematic state.
- Returns
  - *none*

The ToggleKinematic method is used to set the object's internal rigidbody kinematic state.

GetGrabbingObject/0

```
public GameObject GetGrabbingObject()
```

- Parameters
  - *none*
- Returns
  - `GameObject` - The game object of what is grabbing the current object.

The GetGrabbingObject method is used to return the game object that is currently grabbing this object.

IsValidInteractableController/2

```
public bool IsValidInteractableController(GameObject actualController,
AllowedController controllerCheck)
```

- Parameters
  - `GameObject actualController` - The game object of the controller that is being checked.
  - `AllowedController controllerCheck` - The value of which controller is allowed to interact with this object.
- Returns
  - `bool` - Is true if the interacting controller is allowed to grab the object.

The IsValidInteractableController method is used to check to see if a controller is allowed to perform an interaction with this object as sometimes controllers are prohibited from grabbing or using an object depedning on the use case.

ForceStopInteracting/0

```
public void ForceStopInteracting()
```

- Parameters
  - *none*
- Returns
  - *none*

The ForceStopInteracting method forces the object to no longer be interacted with and will cause a controller to drop the object and stop touching it. This is useful if the controller is required to auto interact with another object.

SetGrabbedSnapHandle/1

```
public void SetGrabbedSnapHandle(Transform handle)
```

- Parameters
  - `Transform handle` - A transform of an object to use for the snap handle when the object is grabbed.
- Returns
  - *none*

The SetGrabbedSnapHandle method is used to set the snap handle of the object at runtime.

RegisterTeleporters/0

```
public void RegisterTeleporters()
```

- Parameters
  - *none*
- Returns
  - *none*

The RegisterTeleporters method is used to find all objects that have a teleporter script and register the object on the `OnTeleported` event. This is used internally by the object for keeping Tracked objects positions updated after teleporting.

Example

`SteamVR_Unity_Toolkit/Examples/005_Controller_BasicObjectGrabbing` uses the `VRTK_InteractTouch` and `VRTK_InteractGrab` scripts on the controllers to show how an interactable object can be grabbed and snapped to the controller and thrown around the game world.

`SteamVR_Unity_Toolkit/Examples/013_Controller_UsingAndGrabbingMultipleObjects` shows mutltiple objects that can be grabbed by holding the buttons or grabbed by toggling the button click and also has objects that can have their Using state toggled to show how mutliple items can be turned on at the same time.

Touching Interactable Objects (VRTK_InteractTouch)

Overview

The Interact Touch script is attached to a Controller object within the `[CameraRig]` prefab.

Inspector Parameters

- **Hide Controller On Touch**: Hides the controller model when a valid touch occurs
- **Hide Controller Delay:** The amount of seconds to wait before hiding the controller on touch.
- **Global Touch Highlight Color:** If the interactable object can be highlighted when it's touched but no local colour is set then this global colour is used.
- **Custom Rigidbody Object:** If a custom rigidbody and collider for the rigidbody are required, then a gameobject containing a rigidbody and collider can be passed into this parameter. If this is empty then the rigidbody and collider will be auto generated at runtime to match the HTC Vive default controller.

Class Events

- `ControllerTouchInteractableObject` - Emitted when a valid object is touched
- `ControllerUntouchInteractableObject` - Emitted when a valid object is no longer being touched

Event Payload

- `uint controllerIndex` - The index of the controller doing the interaction
- `GameObject target` - The GameObject of the interactable object that is being interacted with by the controller

Class Methods

ForceTouch/1

```
public void ForceTouch(GameObject obj)
```

- Parameters
  - `GameObject obj` - The game object to attempt to force touch.
- Returns
  - *none*

The ForceTouch method will attempt to force the controller to touch the given game object. This is useful if an object that isn't being touched is required to be grabbed or used as the controller doesn't physically have to be touching it to be forced to interact with it.

GetTouchedObject/0

```
public GameObject GetTouchedObject()
```

- Parameters
  - *none*
- Returns
  - `GameObject` - The game object of what is currently being touched by this controller.

The GetTouchedObject method returns the current object being touched by the controller.

IsObjectInteractable/1

```
public bool IsObjectInteractable(GameObject obj)
```

- Parameters
  - `GameObject obj` - The game object to check to see if it's interactable.
- Returns
  - `bool` - Is true if the given object is of type `VRTK_InteractableObject`.

The IsObjectInteractable method is used to check if a given game object is of type `VRTK_InteractableObject` and whether the object is enabled.

ToggleControllerRigidBody/1

```
public void ToggleControllerRigidBody(bool state)
```

- Parameters
  - `bool state` - The state of whether the rigidbody is on or off. `true` toggles the rigidbody on and `false` turns it off.
- Returns
  - *none*

The ToggleControllerRigidBody method toggles the controller's rigidbody's ability to detect collisions. If it is true then the controller rigidbody will collide with other collidable game objects.

IsRigidBodyActive/0

```
public bool IsRigidBodyActive()
```

- Parameters
  - *none*
- Returns
  - `bool` - Is true if the rigidbody on the controller is currently active and able to affect other scene rigidbodies.

The IsRigidBodyActive method checks to see if the rigidbody on the controller object is active and can affect other rigidbodies in the scene.

ForceStopTouching/0

```
public void ForceStopTouching()
```

- Parameters
  - *none*
- Returns
  - *none*

The ForceStopTouching method will stop the controller from touching an object even if the controller is physically touching the object still.

ControllerColliders/0

```
public Collider[] ControllerColliders()
```

- Parameters
  - *none*
- Returns
  - `Collider[]` - An array of colliders that are associated with the controller.

The ControllerColliders method retrieves all of the associated colliders on the controller.

Example

`SteamVR_Unity_Toolkit/Examples/005_Controller/BasicObjectGrabbing` demonstrates the highlighting of objects that have the `VRTK_InteractableObject` script added to them to show the ability to highlight interactable objects when they are touched by the controllers.

Grabbing Interactable Objects (VRTK_InteractGrab)

Overview

The Interact Grab script is attached to a Controller object within the `[CameraRig]` prefab and the Controller object requires the `VRTK_ControllerEvents` script to be attached as it uses this for listening to the controller button events for grabbing and releasing interactable game objects. It listens for the `AliasGrabOn` and `AliasGrabOff` events to determine when an object should be grabbed and should be released.

The Controller object also requires the `VRTK_InteractTouch` script to be attached to it as this is used to determine when an interactable object is being touched. Only valid touched objects can be grabbed.

An object can be grabbed if the Controller touches a game object which contains the `VRTK_InteractableObject` script and has the flag `isGrabbable` set to `true`.

If a valid interactable object is grabbable then pressing the set `Grab` button on the Controller (default is `Grip`) will grab and snap the object to the controller and will not release it until the `Grab` button is released.

When the Controller `Grab` button is released, if the interactable game object is grabbable then it will be propelled in the direction and at the velocity the controller was at, which can simulate object throwing.

The interactable objects require a collider to activate the trigger and a rigidbody to pick them up and move them around the game world.

Inspector Parameters

- **Controller Attach Point:** The rigidbody point on the controller model to snap the grabbed object to (defaults to the tip).
- **Hide Controller On Grab:** Hides the controller model when a valid grab occurs.
- **Hide Controller Delay:** The amount of seconds to wait before hiding the controller on grab.
- **Grab Precognition:** An amount of time between when the grab button is pressed to when the controller is touching something to grab it. For example, if an object is falling at a fast rate, then it is very hard to press the grab button in time to catch the object due to human reaction times. A higher number here will mean the grab button can be pressed before the controller touches the object and when the collision takes place, if the grab button is still being held down then the grab action will be successful.
- **Throw Multiplier:** An amount to multiply the velocity of any objects being thrown. This can be useful when scaling up the `[CameraRig]` to simulate being able to throw items further.
- **Create Rigid Body When Not Touching:** If this is checked and the controller is not touching an Interactable Object when the grab button is pressed then a rigid body is added to the controller to allow the controller to push other rigid body objects around.

Class Events

- `ControllerGrabInteractableObject` - Emitted when a valid object is grabbed.
- `ControllerUngrabInteractableObject` - Emitted when a valid object is released from being grabbed.

Event Payload

- `uint controllerIndex` - The index of the controller doing the interaction.
- `GameObject target` - The GameObject of the interactable object that is being interacted with by the controller.

Class Methods

ForceRelease/0

```
public void ForceRelease()
```

- Parameters
  - *none*
- Returns
  - *none*

The ForceRelease method will force the controller to stop grabbing the currently grabbed object.

AttemptGrab/0

```
public void AttemptGrab()
```

- Parameters
  - *none*
- Returns
  - *none*

The AttemptGrab method will attempt to grab the currently touched object without needing to press the grab button on the controller.

GetGrabbedObject/0

```
public GameObject GetGrabbedObject()
```

- Parameters
  - *none*
- Returns
  - `GameObject` - The game object of what is currently being grabbed by this controller.

The GetGrabbedObject method returns the current object being grabbed by the controller.

Example

`SteamVR_Unity_Toolkit/Examples/005_Controller/BasicObjectGrabbing` demonstrates the grabbing of interactable objects that have the `VRTK_InteractableObject` script attached to them. The objects can be picked up and thrown around.

`SteamVR_Unity_Toolkit/Examples/013_Controller_UsingAndGrabbingMultipleObjects` demonstrates that each controller can grab and use objects independently and objects can also be toggled to their use state simultaneously.

`SteamVR_Unity_Toolkit/Examples/014_Controller_SnappingObjectsOnGrab` demonstrates the different mechanisms for snapping a grabbed object to the controller.

Using Interactable Objects (VRTK_InteractUse)

Overview

The Interact Use script is attached to a Controller object within the `[CameraRig]` prefab and the Controller object requires the `VRTK_ControllerEvents` script to be attached as it uses this for listening to the controller button events for using and stop using interactable game objects. It listens for the `AliasUseOn` and `AliasUseOff` events to determine when an object should be used and should stop using.

The Controller object also requires the `VRTK_InteractTouch` script to be attached to it as this is used to determine when an interactable object is being touched. Only valid touched objects can be used.

An object can be used if the Controller touches a game object which contains the `VRTK_InteractableObject` script and has the flag `isUsable` set to `true`.

If a valid interactable object is usable then pressing the set `Use` button on the Controller (default is `Trigger`) will call the `StartUsing` method on the touched interactable object.

Inspector Parameters

- **Hide Controller On Use:** Hides the controller model when a valid use action starts.
- **Hide Controller Delay:** The amount of seconds to wait before hiding the controller on use.

Class Events

- `ControllerUseInteractableObject` - Emitted when a valid object starts being used.
- `ControllerUnuseInteractableObject` - Emitted when a valid object stops being used.

Event Payload

- `uint controllerIndex` - The index of the controller doing the interaction
- `GameObject target` - The GameObject of the interactable object that is being interacted with by the controller

Class Methods

GetUsingObject/0

```
public GameObject GetUsingObject()
```

- Parameters
  - *none*
- Returns
  - `GameObject` - The game object of what is currently being used by this controller.

The GetUsingObject method returns the current object being used by the controller.

ForceStopUsing/0

```
public void ForceStopUsing()
```

- Parameters
  - *none*
- Returns
  - *none*

The ForceStopUsing method will force the controller to stop using the currently touched object and will also stop the object's using action.

ForceResetUsing/0

```
public void ForceResetUsing()
```

- Parameters
  - *none*
- Returns
  - *none*

The ForceResetUsing will force the controller to stop using the currently touched object but the object will continue with it's existing using action.

Example

`SteamVR_Unity_Toolkit/Examples/006_Controller_UsingADoor` simulates using a door object to open and close it. It also has a cube on the floor that can be grabbed to show how interactable objects can be usable or grabbable.

`SteamVR_Unity_Toolkit/Examples/008_Controller_UsingAGrabbedObject` which shows that objects can be grabbed with one button and used with another (e.g. firing a gun).

Auto Grabbing Interactable Objects (VRTK_ObjectAutoGrab)

Overview

It is possible to automatically grab an Interactable Object to a specific controller by applying the Object Auto Grab script to the controller that the object should be grabbed by default.

The Object Auto Grab script is attached to a Controller object within the `[CameraRig]` prefab and the Controller object requires the `VRTK_InteractGrab` script to be attached.

Inspector Parameters

- **Object To Grab:** A game object (either within the scene or a prefab) that will be grabbed by the controller on game start.
- **Clone Grabbed Object:** If this is checked then the Object To Grab will be cloned into a new object and attached to the controller leaving the existing object in the scene. This is required if the same object is to be grabbed to both controllers as a single object cannot be grabbed by different controllers at the same time. It is also required to clone a grabbed object if it is a prefab as it needs to exist within the scene to be grabbed.

Example

`SteamVR_Unity_Toolkit/Examples/026_Controller_ForceHoldObject` shows how to automatically grab a sword to each controller and also prevents the swords from being dropped so they are permanently attached to the user's controllers.

Testing Without Using The Headset (VRTK_Simulator)

Overview

To test a scene it is often necessary to use the headset to move to a location. This increases turn-around times and can become cumbersome. The simulator allows navigating through the scene using the keyboard instead, without the need to put on the headset. One can then move around (also through walls) while looking at the monitor and still use the controllers to interact.

The Simulator script is attached to the `[CameraRig]` prefab. Supported movements are: forward, backward, strafe left, strafe right, turn left, turn right, up, down.

Inspector Parameters

- **Keys:** Per default the keys on the left-hand side of the keyboard are used (WASD). They can be individually set as needed. The reset key brings the camera to its initial location.
- **Only In Editor:** Typically the simulator should be turned off when not testing anymore. This option will do this automatically when outside the editor.
- **Step Size:** Depending on the scale of the world the step size can be defined to increase or decrease movement speed.
- **Cam Start:** It can be very handy to start at a certain location instead of having to walk/teleport

there first. A good workflow can be to use the keys to navigate in the scene one time, then copy the transform of the `[CameraRig]`, stop the scene and paste the values to an empty game object that is then assigned here. To start at the original position again it is enough to deactivate the game object.

Player Climb (VRTK_PlayerClimb)

Overview

This class allows player movement based on grabbing of `VRTK_InteractableObject` objects that are tagged as `Climbable`. It should be attached to the `[CameraRig]` object. Because it works by grabbing, each controller should have a `VRTK_InteractGrab` and `VRTK_InteractTouch` component attached.

Inspector Parameters

- **Use Player Scale:** Will scale movement up and down based on the player transform's scale.
- **Use Gravity:** Will allow physics based falling when the user lets go of objects above ground.
- **Safe Zone Teleport Offset:** An additional amount to move the player away from a wall if an ungrab teleport happens due to camera/object collisions.

Class Events

- `PlayerClimbStarted` - Emitted when player climbing has started.
- `PlayerClimbEnded` - Emitted when player climbing has ended.

Event Payload

- `uint controllerIndex` - The index of the controller doing the interaction.
- `GameObject target` - The GameObject of the interactable object that is being interacted with by the controller.

Example

`SteamVR_Unity_Toolkit/Examples/037_CameraRig_ClimbingFalling` shows how to set up a scene with player climbing. There are many different examples showing how the same system can be used in unique ways.

3D Controls (Controls/3D)

In order to interact with the world beyond grabbing and throwing, controls can be used to mimic real-life objects.

A number of controls are available which partially support auto-configuration. So can a slider for example detect its min and max points or a button the distance until a push event should be triggered. In the editor gizmos will be drawn that show the current settings. A yellow gizmo signals a valid configuration. A red one shows that the position of the object should change or switch to manual configuration mode.

All 3D controls extend the following abstract class which provides common methods and events:

- VRTK_Control

The following UI controls are available:

- VRTK_Button
- VRTK_Chest
- VRTK_Door
- VRTK_Drawer
- VRTK_Knob
- VRTK_Lever
- VRTK_Slider

The following UI convenience scripts are available:

- VRTK_ContentHandler

VRTK_Control

Overview

All 3D controls extend the `VRTK_Control` abstract class which provides a default set of methods and events that all of the subsequent controls expose.

Class Events

- `OnValueChanged` - Emitted when the control is interacted with.

Event Payload

- `float value` - The current value in the context of the control extending this abstract class.
- `float value` - The normalized value in the range between 0 and 100 of the control extending this abstract class.

Class Methods

GetValue/0

```
public float GetValue()
```

- Parameters
  - *none*
- Returns
  - `float` - The current value of the control.

The GetValue method returns the current value/position/setting of the control depending on the control that is extending this abstract class.

GetNormalizedValue/0

```
public float GetNormalizedValue()
```

- Parameters
  - *none*
- Returns
  - `float` - The current normalized value of the control.

The GetNormalizedValue method returns the current value mapped onto a range between 0 and 100.

SetContent/2

```
public void SetContent(GameObject content, bool hideContent)
```

- Parameters
  - `GameObject content` - The content to be considered within the control.
  - `bool hideContent` - When true the content will be hidden in addition to being non-interactable in case the control is fully closed.
- Returns
  - *none*

The SetContent method sets the given game object as the content of the control. This will then disable and optionally hide the content when a control is obscuring its view to prevent interacting with content within a control.

GetContent/0

```
public GameObject GetContent()
```

- Parameters
  - *none*
- Returns
  - `GameObject` - The currently stored content for the control.

The GetContent method returns the current game object of the control's content.

VRTK_Button

> extends VRTK_Control

Overview

Attaching the script to a game object will allow the user to interact with it as if it were a push button. The direction into which the button should be pushable can be freely set and auto-detection is supported. Since this is physics-based there needs to be empty space in the push direction so that the button can move.

The script will instantiate the required Rigidbody and ConstantForce components automatically in case they do not exist yet.

Inspector Parameters

- **Connected To:** An optional game object to which the button will be connected. If the game object moves the button will follow along.
- **Direction:** The axis on which the button should move. All other axis will be frozen.
- **Activation Distance:** The local distance the button needs to be pushed until a push event is triggered.
- **Button Strength:** The amount of force needed to push the button down as well as the speed with which it will go back into its original position.

Class Events

- `OnPush` - Emitted when the button is successfully pushed.

Example

`SteamVR_Unity_Toolkit/Examples/025_Controls_Overview` shows a collection of pressable buttons that are interacted with by activating the rigidbody on the controller by pressing the grab button without grabbing an object.

VRTK_Chest

> extends VRTK_Control

Overview

Transforms a game object into a chest with a lid. The direction can be auto-detected with very high reliability or set manually.

The script will instantiate the required Rigidbody, Interactable and HingeJoint components automatically in case they do not exist yet. It will expect three distinct game objects: a body, a lid and a handle. These should be independent and not children of each other.

Inspector Parameters

- **Direction:** The axis on which the chest should open. All other axis will be frozen.
- **Lid:** The game object for the lid.
- **Body:** The game object for the body.
- **Handle:** The game object for the handle.
- **Content:** The parent game object for the chest content elements.
- **Hide Content:** Makes the content invisible while the chest is closed.
- **Max Angle:** The maximum opening angle of the chest.

Example

`SteamVR_Unity_Toolkit/Examples/025_Controls_Overview` shows a chest that can be open and closed, it also displays the current opening angle of the chest.

VRTK_Door

> extends VRTK_Control

Overview

Transforms a game object into a door with an optional handle attached to an optional frame. The direction can be freely set and also very reliably auto-detected.

There are situations when it can be very hard to automatically calculate the correct axis and anchor values for the hinge joint. If this situation is encountered then simply add the hinge joint manually and set these two values. All the rest will still be handled by the script.

The script will instantiate the required Rigidbodies, Interactable and HingeJoint components automatically in case they do not exist yet. Gizmos will indicate the direction.

Inspector Parameters

- **Direction:** The axis on which the door should open.
- **Door:** The game object for the door. Can also be an empty parent or left empty if the script is put

onto the actual door mesh. If no colliders exist yet a collider will tried to be automatically attached to all children that expose renderers.
- **Handles:** The game object for the handles. Can also be an empty parent or left empty. If empty the door can only be moved using the rigidbody mode of the controller. If no collider exists yet a compound collider made up of all children will try to be calculated but this will fail if the door is rotated. In that case a manual collider will need to be assigned.
- **Frame:** The game object for the frame to which the door is attached. Should only be set if the frame will move as well to ensure that the door moves along with the frame.
- **Content:** The parent game object for the door content elements.
- **Hide Content:** Makes the content invisible while the door is closed.
- **Max Angle:** The maximum opening angle of the door.
- **Open Inward:** Can the door be pulled to open.
- **Open Outward:** Can the door be pushed to open.
- **Snapping:** Keeps the door closed with a slight force. This way the door will not gradually open due to some minor physics effect. Does only work if either inward or outward is selected, not both.

Example

`SteamVR_Unity_Toolkit/Examples/025_Controls_Overview` shows a selection of door types, from a normal door and trapdoor, to a door with a catflap in the middle.

VRTK_Drawer

> extends VRTK_Control

Overview

Transforms a game object into a drawer. The direction can be freely set and also auto-detected with very high reliability.

The script will instantiate the required Rigidbody, Interactable and Joint components automatically in case they do not exist yet. There are situations when it can be very hard to automatically calculate the correct axis for the joint. If this situation is encountered simply add the configurable joint manually and set the axis. All the rest will still be handled by the script.

It will expect two distinct game objects: a body and a handle. These should be independent and not children of each other. The distance to which the drawer can be pulled out will automatically set depending on the length of it. If no body is specified the current object is assumed to be the body.

It is possible to supply a third game object which is the root of the contents inside the drawer. When this is specified the VRTK_InteractableObject components will be automatically deactivated in case the drawer is closed or not yet far enough open. This eliminates the issue that a user could grab an object inside a drawer although it is closed.

Inspector Parameters

- **Direction:** The axis on which the chest should open. All other axis will be frozen.
- **Body:** The game object for the body.
- **Handle:** The game object for the handle.
- **Content:** The parent game object for the drawer content elements.
- **Hide Content:** Makes the content invisible while the drawer is closed.
- **Snapping:** Keeps the drawer closed with a slight force. This way the drawer will not gradually open due to some minor physics effect.

Example

`SteamVR_Unity_Toolkit/Examples/025_Controls_Overview` shows a drawer with contents that can be opened and closed freely and the contents can be removed from the drawer.

VRTK_Knob

> extends VRTK_Control

Overview

Attaching the script to a game object will allow the user to interact with it as if it were a radial knob. The direction can be freely set.

The script will instantiate the required Rigidbody and Interactable components automatically in case they do not exist yet.

Inspector Parameters

- **Direction:** The axis on which the knob should rotate. All other axis will be frozen.
- **Min:** The minimum value of the knob.
- **Max:** The maximum value of the knob.
- **Step Size:** The increments in which knob values can change.

Example

`SteamVR_Unity_Toolkit/Examples/025_Controls_Overview` has a couple of rotator knobs that can be rotated by grabbing with the controller and then rotating the controller in the desired direction.

VRTK_Lever

extends VRTK_Control

Overview

Attaching the script to a game object will allow the user to interact with it as if it were a lever. The direction can be freely set.

The script will instantiate the required Rigidbody, Interactable and HingeJoing components automatically in case they do not exist yet. The joint is very tricky to setup automatically though and will only work in straight forward cases. If there are any issues, then create the HingeJoint component manually and configure it as needed.

Inspector Parameters

- **Direction:** The axis on which the lever should rotate. All other axis will be frozen.
- **Min Angle:** The minimum angle of the lever counted from its initial position.
- **Max Angle:** The maximum angle of the lever counted from its initial position.
- **Step Size:** The increments in which lever values can change.

Example

`SteamVR_Unity_Toolkit/Examples/025_Controls_Overview` has a couple of levers that can be grabbed and moved. One lever is horizontal and the other is vertical.

VRTK_Slider

extends VRTK_Control

Overview

Attaching the script to a game object will allow the user to interact with it as if it were a horizontal or vertical slider. The direction can be freely set and auto-detection is supported.

The script will instantiate the required Rigidbody and Interactable components automatically in case they do not exist yet.

Inspector Parameters

- **Direction:** The axis on which the slider should move. All other axis will be frozen.
- **Min:** The minimum value of the slider.

- **Max:** The maximum value of the slider.
- **Step Size:** The increments in which slider values can change. The slider supports snapping.

Example

`SteamVR_Unity_Toolkit/Examples/025_Controls_Overview` has a selection of sliders at various angles with different step values to demonstrate their usage.

VRTK_ContentHandler

Overview

Manages objects defined as content. When taking out an object from a drawer and closing the drawer this object would otherwise disappear even if outside the drawer.

The script will use the boundaries of the control to determine if it is in or out and reparent the object as necessary. It can be put onto individual objects or the parent of multiple objects. Using the latter way all interactable objects under that parent will become managed by the script.

Inspector Parameters

- **Control:** The 3D control responsible for the content.
- **Inside:** The transform containing the meshes or colliders that define the inside of the control.
- **Outside:** Any transform that will act as the parent while the object is not inside the control.

Example

`SteamVR_Unity_Toolkit/Examples/025_Controls_Overview` has a drawer with a collection of items that adhere to this concept.

Abstract Classes (Abstractions/)

To allow for re-usability and object consistency, a collection of abstract classes are provided which can be used to extend into a concrete class providing consistent functionality across many different scripts without needing to duplicate code.

The following abstract classes are available:

- VRTK_DestinationMarker
- VRTK_WorldPointer

VRTK_DestinationMarker

Overview

This abstract class provides the ability to emit events of destination markers within the game world. It can be useful for tagging locations for specific purposes such as teleporting.

It is utilised by the `VRTK_WorldPointer` for dealing with pointer events when the pointer cursor touches areas within the game world.

Inspector Parameters

- **Enable Teleport:** If this is checked then the teleport flag is set to true in the Destination Set event so teleport scripts will know whether to action the new destination.

Class Events

- `DestinationMarkerEnter` - Emitted when a collision with another game object has occurred.
- `DestinationMarkerExit` - Emitted when the collision with the other game object finishes.
- `DestinationMarkerSet` - Emitted when the destination marker is active in the scene to determine the last destination position (useful for selecting and teleporting).

Event Payload

- `float distance` - The distance between the origin and the collided destination.
- `Transform target` - The Transform of the collided destination object.
- `Vector3 destinationPosition` - The world position of the destination marker.
- `bool enableTeleport` - Whether the destination set event should trigger teleport.
- `uint controllerIndex` The optional index of the controller emitting the beam.

Class Methods

SetInvalidTarget/1

```
public virtual void SetInvalidTarget(string name)
```

- Parameters
  - `string name` - The name of the tag or class that is the invalid target.
- Returns
  - *none*

The SetInvalidTarget method is used to set objects that contain the given tag or class matching the name as invalid destination targets.

SetNavMeshCheckDistance/1

```
public virtual void SetNavMeshCheckDistance(float distance)
```

- Parameters
  - `float distance` - The max distance the nav mesh can be from the sample point to be valid..
- Returns
  - *none*

The SetNavMeshCheckDistance method sets the max distance the destination marker position can be from the edge of a nav mesh to be considered a valid destination.

SetHeadsetPositionCompensation/1

```
public virtual void SetHeadsetPositionCompensation(bool state)
```

- Parameters
  - `bool state` - The state of whether to take the position of the headset within the play area into account when setting the destination marker.
- Returns
  - *none*

The SetHeadsetPositionCompensation method determines whether the offset position of the headset from the centre of the play area should be taken into consideration when setting the destination marker. If `true` then it will take the offset position into consideration.

VRTK_WorldPointer

Overview

This abstract class provides any game pointer the ability to know the the state of the implemented pointer. It extends the `VRTK_DestinationMarker` to allow for destination events to be emitted when the pointer cursor collides with objects.

The World Pointer also provides a play area cursor to be displayed for all cursors that utilise this class. The play area cursor is a representation of the current calibrated play area space and is useful for visualising the potential new play area space in the game world prior to teleporting. It can also handle collisions with objects on the new play area space and prevent teleporting if there are any collisions with objects at the potential new destination.

The play area collider does not work well with terrains as they are uneven and cause collisions regularly so it is recommended that handling play area collisions is not enabled when using terrains.

Inspector Parameters

- **Enable Teleport:** If this is checked then the teleport flag is set to true in the Destination Set event so teleport scripts will know whether to action the new destination. This allows controller beams to be enabled on a controller but never trigger a teleport (if this option is unchecked).
- **Controller:** The controller that will be used to toggle the pointer. If the script is being applied onto a controller then this parameter can be left blank as it will be auto populated by the controller the script is on at runtime.
- **Pointer Material:** The material to use on the rendered version of the pointer. If no material is selected then the default `WorldPointer` material will be used.
- **Pointer Hit Color:** The colour of the beam when it is colliding with a valid target. It can be set to a different colour for each controller.
- **Pointer Miss Color:** The colour of the beam when it is not hitting a valid target. It can be set to a different colour for each controller.
- **Show Play Area Cursor:** If this is enabled then the play area boundaries are displayed at the tip of the pointer beam in the current pointer colour.
- **Play Area Cursor Dimensions:** Determines the size of the play area cursor and collider. If the values are left as zero then the Play Area Cursor will be sized to the calibrated Play Area space.
- **Handle Play Area Cursor Collisions:** If this is ticked then if the play area cursor is colliding with any other object then the pointer colour will change to the `Pointer Miss Color` and the `WorldPointerDestinationSet` event will not be triggered, which will prevent teleporting into areas where the play area will collide.
- **Ignore Target With Tag Or Class:** A string that specifies an object Tag or the name of a Script attached to an object and notifies the play area cursor to ignore collisions with the object.
- **Pointer Visibility:** Determines when the pointer beam should be displayed:
  - `On_When_Active` only shows the pointer beam when the Pointer button on the controller is pressed.
  - `Always On` ensures the pointer beam is always visible but pressing the Pointer button on the controller initiates the destination set event.
  - `Always Off` ensures the pointer beam is never visible but the destination point is still set and pressing the Pointer button on the controller still initiates the destination set event.
- **Hold Button To Activate:** If this is checked then the pointer beam will be activated on first press of the pointer alias button and will stay active until the pointer alias button is pressed again. The destination set event is emitted when the beam is deactivated on the second button press.
- **Activate Delay:** The time in seconds to delay the pointer beam being able to be active again. Useful for preventing constant teleportation.

Class Methods

setPlayAreaCursorCollision/1

```
public virtual void setPlayAreaCursorCollision(bool state)
```

- Parameters
  - `bool state` - The state of whether to check for play area collisions.
- Returns
  - *none*

The setPlayAreaCursorCollision method determines whether play area collisions should be taken into consideration with the play area cursor.

IsActive/0

```
public virtual bool IsActive()
```

- Parameters
  - *none*
- Returns
  - `bool` - Is true if the pointer is currently active.

The IsActive method is used to determine if the pointer currently active.

CanActivate/0

```
public virtual bool CanActivate()
```

- Parameters
  - *none*
- Returns
  - `bool` - Is true if the pointer is able to be activated due to the activation delay timer being zero.

The CanActivate method checks to see if the pointer can be activated as long as the activation delay timer is zero.

ToggleBeam/1

```
public virtual void ToggleBeam(bool state)
```

- Parameters
  - `bool state` - The state of whether to enable or disable the beam.
- Returns
  - *none*

The ToggleBeam method allows the pointer beam to be toggled on or off via code at runtime. If true is passed as the state then the beam is activated, if false then the beam is deactivated.

Examples

This directory contains Unity3d scenes that demonstrate the scripts and prefabs being used in the game world to create desired functionality.

There is also a `/Resources/Scripts` directory within the `SteamVR_Unity_Toolkit/Examples` directory that contains helper scripts utilised by the example scenes to highlight certain functionality (such as event listeners). These example scripts are not required for real world usage.

Current Examples

001_CameraRig_VR_PlayArea

A simple scene showing the `[CameraRig]` prefab usage.

View Example Tour on YouTube

002_Controller_Events

A simple scene displaying the events from the controller in the console window.

View Example Tour on YouTube

003_Controller_SimplePointer

A scene with basic objects that can be pointed at with the laser beam from the controller activated by pressing the `Touchpad`. The pointer events are also displayed in the console window.

View Example Tour on YouTube

004_CameraRig_BasicTeleport

A scene with basic objects that can be traversed using the controller laser beam to point at an object in the game world where the user is to be teleported to by pressing `Touchpad` on the controller. When the `Touchpad` is released, the user is teleported to the laser beam end location.

View Example Tour on YouTube

005_Controller_BasicObjectGrabbing

A scene with a selection of objects that can be grabbed by touching them with the controller and pressing the `Grip` button down. Releasing the grip button will propel the object in the direction and velocity of the grabbing controller. The scene also demonstrates simple highlighting of objects when the controller touches them. The interaction events are also displayed in the console window.

View Example Tour on YouTube

006_Controller_UsingADoor

A scene with a door interactable object that is set to `usable` and when the door is used by pressing the controller `Trigger` button, the door swings open (or closes if it's already open).

View Example Tour on YouTube

007_CameraRig_HeightAdjustTeleport

A scene with a selection of varying height objects that can be traversed using the controller laser beam to point at an object and if the laser beam is pointing on top of the object then the user is teleported to the top of the object. Also, it shows that if the user steps into a part of the play area that is not on the object then the user will fall to the nearest object. This also enables the user to climb objects just by standing over them as the floor detection is done from the position of the headset.

View Example Tour on YouTube

008_Controller_UsingAGrabbedObject

A scene with interactable objects that can be grabbed (pressing the `Grip` controller button) and then used (pressing the `Trigger` controller button). There is a gun on a table that can be picked up and fired, or a strange box that when picked up and used the top spins.

View Example Tour on YouTube

009_Controller_BezierPointer

A scene with a selection of varying height objects that can be traversed using the controller however, rather than just pointing a straight beam, the beam is curved (over a bezier curve) which allows climbing on top of items that the user cannot visibly see.

View Example Tour on YouTube

010_CameraRig_TerrainTeleporting

A scene with a terrain object and a selection of varying height 3d objects that can be traversed using the controller laser beam pointer. It shows how the Height Adjust Teleporter can be used to climb up and down game objects as well as traversing terrains as well.

View Example Tour on YouTube

011_Camera_HeadSetCollisionFading

A scene with three walls around the play area and if the user puts their head into any of the collidable walls then the headset fades to black to prevent seeing unwanted object clipping artifacts.

View Example Tour on YouTube

012_Controller_PointerWithAreaCollision

A scene which demonstrates how to use a controller pointer to traverse a world but where the beam shows the projected play area space and if the space collides with any objects then the teleportation action is disabled. This means it's possible to create a level with areas where the user cannot teleport to because they would allow the user to clip into objects.

View Example Tour on YouTube

013_Controller_UsingAndGrabbingMultipleObjects

A scene which demonstrates how interactable objects can be grabbed by holding down the grab button continuously or by pressing the grab button once to pick up and once again to release. The scene also shows that the use button can have a hold down to keep using or a press use button once to start using and press again to stop using. This allows multiple objects to be put into their Using state at the same time as also demonstrated in this example scene.

View Example Tour on YouTube

014_Controller_SnappingObjectsOnGrab

A scene with a selection of objects that demonstrate the different snap to controller mechanics. The two green guns, light saber and sword utilise a Snap Handle which uses an empty game object as a child of the interactable object as the orientation point at grab, so the rotation and position of the object matches that of the given `Snap Handle`. The red gun utilises a basic snap where no precision is required and no Snap Handles are provided which does not affect the object's rotation but positions the centre of the object to the snap point on the controller. The red/green gun utilises the `Precision Snap` which does not affect the rotation or position of the grabbed object and picks the object up at the point that the controller snap point is touching the object.

View Example Tour on YouTube

015_Controller_TouchpadAxisControl

A scene with an R/C car that is controlled by using the Controller Touchpad. Moving a finger up and down on the Touchpad will cause the car to drive forward or backward. Moving a finger to the left or right of the `Touchpad` will cause the car to turn in that direction. Pressing the `Trigger` will cause the car to jump, this utilises the Trigger axis and the more the trigger is depressed, the higher the car will jump.

View Example Tour on YouTube

016_Controller_HapticRumble

A scene with a collection of breakable boxes and a sword. The sword can be picked up and swung at the boxes. The controller rumbles at an appropriate vibration depending on how hard the sword hits the box. The box also breaks apart if it is hit hard enough by the sword.

View Example Tour on YouTube

017_CameraRig_TouchpadWalking

A scene which demonstrates how to move around the game world using the touchpad by sliding a finger forward and backwards to move in that direction. Sliding a finger left and right across the touchpad strafes in that direction. The rotation is done via the user in game physically rotating their body in the place space and whichever way the headset is looking will be the way the user walks forward. Crouching is also possible as demonstrated in this scene and in conjunction with the Headset Collision Fade script it can detect unwanted collisions (e.g. if the user stands up whilst walking as crouched) and reset their position to the last good known position.

View Example Tour on YouTube

018_CameraRig_FramesPerSecondCounter

A scene which displays the frames per second in the centre of the headset view. Pressing the trigger generates a new sphere and pressing the touchpad generates ten new spheres. Eventually when lots of spheres are present the FPS will drop and demonstrate the prefab.

View Example Tour on YouTube

019_Controller_InteractingWithPointer

A scene which shows how the controller pointer beam can be used to toggle the use actions on interactable objects. Pressing the touchpad activates the beam and aiming it at objects will toggle their use state. It also demonstrates how a game menu could be created by using interactable objects snapped to a game object can trigger actions. Pressing the Application Menu button displays a cube connected to the controller which has menu options. Pointing the beam with the other controller at the cube will select the menu options accordingly.

View Example Tour on YouTube

020_CameraRig_MeshTeleporting

A scene with an object with a mesh collider to demonstrate how the Height Adjust Teleporter can be used to climb up and down onbjects with a mesh collider.

View Example Tour on YouTube

021_Controller_GrabbingObjectsWithJoints

A scene with a collection of Interactable Objects that are attached to other objects with joints. The example shows that Interactable Objects can have different attach mechanics to determine the best way of connecting the object to the controller. Fixed Joint works well for holding objects like cubes as they track perfectly to the controller whereas a Spring Joint works well on the drawer to give it a natural slide when operating. Finally, the Rotator Track works well on the door to give a natural control over the swing of the door. There is also a Character Joint object that can be manipulated into different shapes by pulling each of the relevant sections.

View Example Tour on YouTube

022_Controller_CustomBezierPointer

A scene that demonstrates how the Bezier Pointer can have complex objects passed to it to generate the tracer beam and the cursor of the pointer. In the scene, particle objects with rotations are used to demonstrate a different look to the bezier pointer beam.

View Example Tour on YouTube

023_Controller_ChildOfControllerOnGrab

A scene that demonstrates the grab mechanic where the object being grabbed becomes a child of the controller doing the grabbing. This works well for objects that need absolute tracking of the controller and do not want to be disjointed under any circumstances. The object becomes an extension of the controller. The scene demonstrates this with a bow and arrow example, where the bow can be picked up and tracked to the controller, whilst the other controller is responsible for picking up arrows to fire in the bow.

View Example Tour on YouTube

024_CameraRig_ExcludeTeleportLocation

A scene that shows how to exclude certain objects from being teleportable by either applying a named Tag to the object or by applying a Script of a certain name. In the scene, the yellow objects are excluded from teleport locations by having an `ExcludeTeleport` tag set on them and the black objects are excluded by having a script called `ExcludeTeleport` attached to them. The `ExcludeTeleport` script has no methods and is just used as a placeholder.

View Example Tour on YouTube

025_Controls_Overview

A scene that showcases the existing interactive controls, different ways how they can be set up and how to react to events sent by them.

View Example Tour on YouTube

026_Controller_ForceHoldObject

A scene that shows how to grab an object on game start and prevent the user from dropping that object. The scene auto grabs two swords to each of the controllers and it's not possible to drop either of the swords.

View Example Tour on YouTube

027_CameraRig_TeleportByModelVillage

A scene that demonstrates how to teleport to different locations without needing a world pointer and using the Destination Events abstract class on objects that represent a mini map of the game world. Touching and using an object on the map teleports the user to the specified location.

View Example Tour on YouTube

028_CameraRig_RoomExtender

A scene that demonstates the concept of extending the physical room scale space by multiplying the physical steps taken in the chaperone bounds. A higher multiplier will mean the user can walk further in the play area and the walk multiplier can be toggled by a button press.

View Example Tour on YouTube

029_Controller_Tooltips

A scene that demonstrates adding tooltips to game objects and to the controllers using the prefabs `ObjectTooltip` and `ControllerTooltips`.

030_Controls_RadialTouchpadMenu

A scene that demonstrates adding dynamic radial menus to controllers and other objects using the prefab `RadialMenu`.

031_CameraRig_HeadsetGazePointer

A scene that demonstrates the ability to attach a pointer to the headset to allow for a gaze pointer for teleporting or other interactions supported by the World Pointers. The `Touchpad` on the right controller activates the gaze beam, where as the `Touchpad` on the left controller activates a beam projected from a drone in the sky as the World Pointers can be attached to any object.

032_Controller_CustomControllerModel

A scene that demonstrates how to use custom models for the controllers instead of the default HTC Vive controllers. It uses two simple hands in place of the default controllers and shows simple state changes based on whether the grab button or use button are being pressed.

033_CameraRig_TeleportingInNavMesh

A scene that demonstrates how a baked NavMesh can be used to define the regions that a user is allowed to teleport into.

034_Controls_InteractingWithUnityUI

A scene that demonstrates how to interact with Unity UI elements. The scene uses the `VRTK_UIPointer` script on the right Controller to allow for the interaction with Unity UI elements using a Simple Pointer beam. The left Controller controls a Simple Pointer on the headset to demonstrate gaze interaction with Unity UI elements.

035_Controller_OpacityAndHighlighting

A scene that demonstrates how to change the opacity of the controller and how to highlight elements of the controller such as the buttons or even the entire controller model.

036_Controller_CustomCompoundPointer

A scene that demonstrates how the Bezier Pointer can display an object (teleport beam) only if the teleport location is valid, and can create an animated trail along the tracer curve. This scene provides a textured environment for testing the teleport, some active "plasma" spheres on the wall that can be activated with the pointer and another sphere that can be also grabbed and launched around.

037_CameraRig_ClimbingFalling

A scene that demonstrates how to set up the climbing mechanism with different activities to try it with. A `VRTK_PlayerClimb` object is needed on the `[CameraRig]`. `VRTK_HeightAdjustTeleport` is also added to the `[CameraRig]` to allow movement, but also to allow walking off edges with `UseGravity` enabled. Various objects with a `VRTK_InteractableObject` component are scattered throughout the level. They all have the `GrabAttachMechanic` set to `Climbable`.