Introduction

This file provides documentation on how to use the included prefabs and scripts.

Prefabs (VRTK/Prefabs)

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

Frames Per Second Canvas (VRTK_FramesPerSecondViewer)

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.

This script is pretty much a copy and paste from the script at: http://talesfromtherift.com/vr-fps-counter/ So all credit to Peter Koch for his work. Twitter: @peterept

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

`VRTK/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.

Object Tooltip (VRTK_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 `VRTK/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.
- **Draw Line From:** An optional transform of where to start drawing the line from. If one is not provided 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.

Class Methods

Reset/0

```
public void Reset()
```

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

The Reset method resets the tooltip back to its initial state

Example

`VRTK/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.

Controller Tooltips (VRTK_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 `VRTK/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`.

Class Methods

ToggleTips/2

```
public void ToggleTips(bool state, TooltipButtons element =
TooltipButtons.None)
```

- Parameters
  - `bool state` - The state of whether to display or hide the controller tooltips, true will display and false will hide.
  - `TooltipButtons element` - The specific element to hide the tooltip on, if it is `TooltipButtons.None` then it will hide all tooltips. Optional parameter defaults to `TooltipButtons.None`
- Returns
  - *none*

The ToggleTips method will display the controller tooltips if the state is `true` and will hide the controller tooltips if the state is `false`. An optional `element` can be passed to target a specific controller tooltip to toggle otherwise all tooltips are toggled.

Example

`VRTK/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.

Radial Menu (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.

Inspector Parameters

- **Buttons:** An array of Buttons that define the interactive buttons required to be displayed as part of the radial menu.
- **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 colour of the buttons, default is white.
- **Offset Distance:** The distance the buttons should move away from the centre. 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.
- **Is Shown:** Whether the buttons are shown
- **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.

Example

`VRTK/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.

Independent Radial Menu Controller

(VRTK_IndependentRadialMenuController)

extends RadialMenuController

Overview

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.

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 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 Headset Camera.

Console Viewer Canvas (VRTK_ConsoleViewer)

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.

Class Methods

SetCollapse/1

```
public void SetCollapse(bool state)
```

- Parameters
  - `bool state` - The state of whether to collapse the output messages, true will collapse and false will not collapse.
- Returns
  - *none*

The SetCollapse method determines whether the console will collapse same message output into the same line. A state of `true` will collapse messages and `false` will print the same message for each line.

ClearLog/0

```
public void ClearLog()
```

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

The ClearLog method clears the current log view of all messages

---

Abstract Classes (VRTK/Scripts/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.

- Destination Marker
- World Pointer

Destination Marker (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).

Unity Events

Adding the `VRTK_DestinationMarker_UnityEvents` component to `VRTK_DestinationMarker` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnDestinationMarkerEnter` - Emits the DestinationMarkerEnter class event.
- `OnDestinationMarkerExit` - Emits the DestinationMarkerExit class event.
- `OnDestinationMarkerSet` - Emits the DestinationMarkerSet class event.

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/2

```
public virtual void SetInvalidTarget(string name, VRTK_TagOrScriptPolicyList
list = null)
```

- Parameters
  - `string name` - The name of the tag or class that is the invalid target.
  - `VRTK_TagOrScriptPolicyList list` - The Tag Or Script list policy to check the set operation on.
- Returns
  - *none*

The SetInvalidTarget method is used to set objects that contain the given tag or class matching the name as invalid destination targets. It can also accept a VRTK_TagOrScriptPolicyList for a more custom level of policy management.

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.

World Pointer (VRTK_WorldPointer)

extends VRTK_DestinationMarker

Overview

This abstract class provides any game pointer the ability to know 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

- **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.
- **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.
- **Target Tag Or Script List Policy:** A specified VRTK_TagOrScriptPolicyList to use to determine whether the play area cursor collisions will be acted upon. If a list is provided then the 'Ignore Target With Tag Or Class' parameter will be ignored.
- **Pointer Visibility:** Determines when the pointer beam should be displayed.
- **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 Variables

- `public enum pointerVisibilityStates` - States of Pointer Visibility.
  - `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.

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.

Highlighters (VRTK/Scripts/Highlighters)

This directory contains scripts that are used to provide different object highlighting.

- Base Highlighter
- Material Colour Swap
- Outline Object Copy

Base Highlighter (VRTK_BaseHighlighter)

Overview

The Base Highlighter is an abstract class that all other highlighters inherit and are required to implement the public methods.

As this is an abstract class, it cannot be applied directly to a game object and performs no logic.

Inspector Parameters

- **Active:** Determines if this highlighter is the active highlighter for the object the component is attached to. Only 1 active highlighter can be applied to a game object.

Class Methods

Initialise/2

```
public abstract void Initialise(Color? color = null, Dictionary<string,
object> options = null);
```

- Parameters
  - `Color? color` - An optional colour may be passed through at point of initialisation in case the highlighter requires it.
  - `Dictionary<string, object> options` - An optional dictionary of highlighter specific options that may be differ with highlighter implementations.
- Returns
  - *none*

The Initalise method is used to set up the state of the highlighter.

Highlight/2

```
public abstract void Highlight(Color? color = null, float duration = 0f);
```

- Parameters
  - `Color? color` - An optional colour to highlight the game object to. The highlight colour may already have been set in the `Initialise` method so may not be required here.
  - `float duration` - An optional duration of how long before the highlight has occured. It can be used by highlighters to fade the colour if possible.
- Returns
  - *none*

The Highlight method is used to initiate the highlighting logic to apply to an object.

Unhighlight/2

```
public abstract void Unhighlight(Color? color = null, float duration = 0f);
```

- Parameters
  - `Color? color` - An optional colour that could be used during the unhighlight phase. Usually will be left as null.
  - `float duration` - An optional duration of how long before the unhighlight has occured.
- Returns
  - *none*

The Unhighlight method is used to initiate the logic that returns an object back to it's original appearance.

GetOption/2

```
public virtual T GetOption<T>(Dictionary<string, object> options, string key)
```

- Type Params
  - `T` - The system type that is expected to be returned.
- Parameters
  - `Dictionary<string, object> options` - The dictionary of options to check in.
  - `string key` - The identifier key to look for.
- Returns
  - `T` - The value in the options at the given key returned in the provided system type.

The GetOption method is used to return a value from the options array if the given key exists.

Material Colour Swap (VRTK_MaterialColorSwapHighlighter)

extends VRTK_BaseHighlighter

Overview

The Material Colour Swap Highlighter is a basic implementation that simply swaps the texture colour for the given highlight colour.

Due to the way the object material is interacted with, changing the material colour will break Draw Call Batching in Unity whilst the object is highlighted.

The Draw Call Batching will resume on the original material when the item is no longer highlighted.

This is the default highlighter that is applied to any script that requires a highlighting component (e.g. `VRTK_Interactable_Object` or `VRTK_ControllerActions`).

Inspector Parameters

- **Emission Darken:** The emission colour of the texture will be the highlight colour but this percent darker.

Class Methods

Initialise/2

```
public override void Initialise(Color? color = null, Dictionary<string,
object> options = null)
```

- Parameters
  - `Color? color` - Not used.
  - `Dictionary<string, object> options` - A dictionary array containing the highlighter options:
    - `<'resetMainTexture', bool>` - Determines if the default main texture should be cleared on highlight. `true` to reset the main default texture, `false` to not reset it.
- Returns
  - *none*

The Initialise method sets up the highlighter for use.

Highlight/2

```
public override void Highlight(Color? color, float duration = 0f)
```

- Parameters
  - `Color? color` - The colour to highlight to.
  - `float duration` - The time taken to fade to the highlighted colour.
- Returns
  - *none*

The Highlight method initiates the change of colour on the object and will fade to that colour (from a base white colour) for the given duration.

Unhighlight/2

```
public override void Unhighlight(Color? color = null, float duration = 0f)
```

- Parameters
  - `Color? color` - Not used.
  - `float duration` - Not used.
- Returns
  - *none*

The Unhighlight method returns the object back to it's original colour.

Example

`VRTK/Examples/005_Controller_BasicObjectGrabbing` demonstrates the solid highlighting on the green cube, red cube and flying saucer when the controller touches it.

`VRTK/Examples/035_Controller_OpacityAndHighlighting` demonstrates the solid highlighting if the right controller collides with the green box or if any of the buttons are pressed.

Outline Object Copy (VRTK_OutlineObjectCopyHighlighter)

extends VRTK_BaseHighlighter

Overview

The Outline Object Copy Highlighter works by making a copy of a mesh and adding an outline shader to it and toggling the appearance of the highlighted object.

Inspector Parameters

- **Thickness:** The thickness of the outline effect
- **Custom Outline Model:** The GameObject to use as the model to outline. If one isn't provided then the first GameObject with a valid Renderer in the current GameObject hierarchy will be used.
- **Custom Outline Model Path:** A path to a GameObject to find at runtime, if the GameObject doesn't exist at edit time.

Class Methods

Initialise/2

```
public override void Initialise(Color? color = null, Dictionary<string,
object> options = null)
```

- Parameters
  - `Color? color` - Not used.
  - `Dictionary<string, object> options` - A dictionary array containing the highlighter options:
    - `<'thickness', float>` - Same as `thickness` inspector parameter.
    - `<'customOutlineModel', GameObject>` - Same as `customOutlineModel` inspector parameter.
    - `<'customOutlineModelPath', string>` - Same as `customOutlineModelPath` inspector parameter.
- Returns
  - *none*

The Initialise method sets up the highlighter for use.

Highlight/2

```
public override void Highlight(Color? color, float duration = 0f)
```

- Parameters

    - `Color? color` - The colour to outline with.
    - `float duration` - Not used.
- Returns
    - *none*

The Highlight method initiates the outline object to be enabled and display the outline colour.

Unhighlight/2

```
public override void Unhighlight(Color? color = null, float duration = 0f)
```

- Parameters
    - `Color? color` - Not used.
    - `float duration` - Not used.
- Returns
    - *none*

The Unhighlight method hides the outline object and removes the outline colour.

Example

`VRTK/Examples/005_Controller_BasicObjectGrabbing` demonstrates the outline highlighting on the green sphere when the controller touches it.

`VRTK/Examples/035_Controller_OpacityAndHighlighting` demonstrates the outline highlighting if the left controller collides with the green box.

Scripts (VRTK/Scripts)

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

- Controller Events
- Controller Actions
- Device Finder
- Simple Pointer
- Bezier Pointer
- UI Pointer
- Basic Teleport
- Height Adjust Teleport
- Headset Collision
- Headset Fade
- Headset Collision Fade

- Teleport Disable On Headset Collision
- Player Presence
- Touchpad Walking
- Room Extender
- Interactable Object
- Interact Touch
- Interact Grab
- Interact Use
- Object Auto Grab
- Player Climb
- Dash Teleport
- Tag Or Script Policy List
- Simulating Headset Movement
- Adaptive Quality

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 overridden 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.

- **Trigger Click Threshold:** The level on the trigger axis to reach before a click is registered.

Class Variables

- `public enum ButtonAlias` - Button types
  - `Trigger_Hairline` - The trigger is squeezed past the current hairline threshold.
  - `Trigger_Touch` - The trigger is squeezed a small amount.
  - `Trigger_Press` - The trigger is squeezed about half way in.
  - `Trigger_Click` - The trigger is squeezed all the way until it clicks.
  - `Grip` - The grip button is pressed.
  - `Touchpad_Touch` - The touchpad is touched (without pressing down to click).
  - `Touchpad_Press` - The touchpad is pressed (to the point of hearing a click).
  - `Application_Menu` - The application menu button is pressed.
  - `Undefined` - No button specified
- `public bool triggerPressed` - This will be true if the trigger is squeezed about half way in. Default: `false`
- `public bool triggerTouched` - This will be true if the trigger is squeezed a small amount. Default: `false`
- `public bool triggerHairlinePressed` - This will be true if the trigger is squeezed a small amount more from any previous squeeze on the trigger. Default: `false`
- `public bool triggerClicked` - This will be true if the trigger is squeezed all the way until it clicks. Default: `false`
- `public bool triggerAxisChanged` - This will be true if the trigger has been squeezed more or less. Default: `false`
- `public bool applicationMenuPressed` - This will be true if the application menu is held down. Default: `false`
- `public bool touchpadPressed` - This will be true if the touchpad is held down. Default: `false`
- `public bool touchpadTouched` - This will be true if the touchpad is being touched. Default: `false`
- `public bool touchpadAxisChanged` - This will be true if the touchpad touch position has changed. Default: `false`
- `public bool gripPressed` - This will be true if the grip is held down. Default: `false`
- `public bool pointerPressed` - This will be true if the button aliased to the pointer is held down. Default: `false`
- `public bool grabPressed` - This will be true if the button aliased to the grab is held down. Default: `false`
- `public bool usePressed` - This will be true if the button aliased to the use is held down. Default: `false`
- `public bool uiClickPressed` - This will be true if the button aliased to the UI click is held down. Default: `false`
- `public bool menuPressed` - This will be true if the button aliased to the application menu is held down. Default: `false`

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.
- `ControllerEnabled` - Emitted when the controller is enabled.
- `ControllerDisabled` - Emitted when the controller is disabled.

Unity Events

Adding the `VRTK_ControllerEvents_UnityEvents` component to `VRTK_ControllerEvents` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnTriggerPressed` - Emits the TriggerPressed class event.
- `OnTriggerReleased` - Emits the TriggerReleased class event.
- `OnTriggerTouchStart` - Emits the TriggerTouchStart class event.
- `OnTriggerTouchEnd` - Emits the TriggerTouchEnd class event.
- `OnTriggerHairlineStart` - Emits the TriggerHairlineStart class event.
- `OnTriggerHairlineEnd` - Emits the TriggerHairlineEnd class event.
- `OnTriggerClicked` - Emits the TriggerClicked class event.
- `OnTriggerUnclicked` - Emits the TriggerUnclicked class event.

- `OnTriggerAxisChanged` - Emits the TriggerAxisChanged class event.
- `OnApplicationMenuPressed` - Emits the ApplicationMenuPressed class event.
- `OnApplicationMenuReleased` - Emits the ApplicationMenuReleased class event.
- `OnGripPressed` - Emits the GripPressed class event.
- `OnGripReleased` - Emits the GripReleased class event.
- `OnTouchpadPressed` - Emits the TouchpadPressed class event.
- `OnTouchpadReleased` - Emits the TouchpadReleased class event.
- `OnTouchpadTouchStart` - Emits the TouchpadTouchStart class event.
- `OnTouchpadTouchEnd` - Emits the TouchpadTouchEnd class event.
- `OnTouchpadAxisChanged` - Emits the TouchpadAxisChanged class event.
- `OnAliasPointerOn` - Emits the AliasPointerOn class event.
- `OnAliasPointerOff` - Emits the AliasPointerOff class event.
- `OnAliasPointerSet` - Emits the AliasPointerSet class event.
- `OnAliasGrabOn` - Emits the AliasGrabOn class event.
- `OnAliasGrabOff` - Emits the AliasGrabOff class event.
- `OnAliasUseOn` - Emits the AliasUseOn class event.
- `OnAliasUseOff` - Emits the AliasUseOff class event.
- `OnAliasUIClickOn` - Emits the AliasMenuOn class event.
- `OnAliasUIClickOff` - Emits the AliasMenuOff class event.
- `OnAliasMenuOn` - Emits the AliasUIClickOn class event.
- `OnAliasMenuOff` - Emits the AliasUIClickOff class event.
- `OnControllerEnabled` - Emits the ControllerEnabled class event.
- `OnControllerDisabled` - Emits the ControllerDisabled class event.

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 physical game controller. This can be useful for determining which way the controller is being rotated and at what speed the rotation is occurring.

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.

IsButtonPressed/1

```
public bool IsButtonPressed(ButtonAlias button)
```

- Parameters
  - `ButtonAlias button` - The button to check if it's being pressed.
- Returns
  - `bool` - Is true if the button is being pressed.

The IsButtonPressed method takes a given button alias and returns a boolean whether that given button is currently being pressed or not.

Example

`VRTK/Examples/002_Controller_Events` shows how the events are utilised and listened to. The accompanying example script can be viewed in `VRTK/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.

The highlighting of the controller is defaulted to use the `VRTK_MaterialColorSwapHighlighter` if no other highlighter is applied to the Object.

Inspector Parameters

- **Model Element Paths:** A collection of strings that determine the path to the controller model sub elements for identifying the model parts at runtime. The paths will default to the model element paths of the selected SDK Bridge.
  - The available model sub elements are:
  - `Body Model Path`: The overall shape of the controller.
  - `Trigger Model Path`: The model that represents the trigger button.
  - `Grip Left Model Path`: The model that represents the left grip button.
  - `Grip Right Model Path`: The model that represents the right grip button.
  - `Touchpad Model Path`: The model that represents the touchpad.
  - `App Menu Model Path`: The model that represents the application menu button.
  - `System Menu Model Path`: The model that represents the system menu button.
- **Element Highlighter Overrides:** A collection of highlighter overrides for each controller model sub element. If no highlighter override is given then highlighter on the Controller game object is used.
  - The available model sub elements are:
  - `Body`: The highlighter to use on the overall shape of the controller.
  - `Trigger`: The highlighter to use on the trigger button.
  - `Grip Left`: The highlighter to use on the left grip button.
  - `Grip Right`: The highlighter to use on the right grip button.

- `Touchpad`: The highlighter to use on the touchpad.
- `App Menu`: The highlighter to use on the application menu button.
- `System Menu`: The highlighter to use on the system menu button.

Class Events

- `ControllerModelVisible` - Emitted when the controller model is toggled to be visible.
- `ControllerModelInvisible` - Emitted when the controller model is toggled to be invisible.

Unity Events

Adding the `VRTK_ControllerActions_UnityEvents` component to `VRTK_ControllerActions` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnControllerModelVisible` - Emits the ControllerModelVisible class event.
- `OnControllerModelInvisible` - Emits the ControllerModelInvisible class event.

Event Payload

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

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 visible - `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` - 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 its 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 its 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 duration` - The duration of fade from white to the highlight colour.
- 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 separately.

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 duration` - The duration of fade from white to the highlight colour.
- 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 duration` - The duration of fade from white to the highlight colour.
- 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 duration` - The duration of fade from white to the highlight colour.
- 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 duration` - The duration of fade from white to the highlight colour.
- Returns
  - *none*

The ToggleHighlightApplicationMenu method is a shortcut method that makes it easier to toggle

the highlight state of the controller application menu element.

ToggleHighlighBody/3

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

- Parameters
  - `bool state` - The highlight colour state, `true` will enable the highlight on the body and `false` will remove the highlight from the body.
  - `Color? highlight` - The colour to highlight the body with.
  - `float duration` - The duration of fade from white to the highlight colour.
- Returns
  - *none*

The ToggleHighlighBody method is a shortcut method that makes it easier to toggle the highlight state of the controller body 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 duration` - The duration of fade from white to the highlight colour.
- 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.

InitaliseHighlighters/0

```
public void InitaliseHighlighters()
```

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

The InitaliseHighlighters method sets up the highlighters on the controller model.

Example

`VRTK/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.

`VRTK/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 Variables

- `public enum Devices` - Possible devices.
  - `Headset` - The headset.
  - `Left_Controller` - The left hand controller.
  - `Right_Controller` - The right hand controller.
- `public enum ControllerHand` - Controller hand reference.
  - `None` - No hand is assigned.
  - `Left` - The left hand is assigned.
  - `Right` - The right hand is assigned.

Class Methods

TrackedIndexIsController/1

```
public static bool TrackedIndexIsController(uint index)
```

- Parameters
  - `uint index` - The index of the tracked object to find.
- Returns
  - `bool` - Returns true if the given index is a tracked object of type controller.

The TrackedIndexIsController method is used to determine if a given tracked object index belongs to a tracked controller.

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 GameObject TrackedObjectByIndex(uint index)
```

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

The TrackedObjectByIndex method is used to find the GameObject of a tracked object by its generated index.

TrackedObjectOrigin/1

```
public static Transform TrackedObjectOrigin(GameObject obj)
```

- Parameters
  - `GameObject obj` - The GameObject to get the origin for.
- Returns
  - `Transform` - The transform of the tracked object's origin or if an origin is not set then the transform parent.

The TrackedObjectOrigin method is used to find the tracked object's origin.

TrackedObjectOfGameObject/2

```
public static GameObject TrackedObjectOfGameObject(GameObject obj, out uint
index)
```

- Parameters
  - `GameObject obj` - The game object to check for the presence of a tracked object on.
  - `out uint index` - The variable to store the tracked object's index if one is found. It returns 0 if no index is found.
- Returns
  - `GameObject` - The GameObject of the tracked object.

The TrackedObjectOfGameObject method is used to find the tracked object associated with the given game object and it can also return the index of the tracked object.

DeviceTransform/1

```
public static Transform DeviceTransform(Devices device)
```

- Parameters
  - `Devices device` - The Devices enum to get the transform for.
- Returns
  - `Transform` - The transform for the given Devices enum.

The DeviceTransform method returns the transform for a given Devices enum.

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` - A ControllerHand 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` - A ControllerHand 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.

GetControllerLeftHand/0

```
public static GameObject GetControllerLeftHand()
```

- Parameters
  - *none*
- Returns
  - `GameObject` - The left hand controller.

The GetControllerLeftHand method retrieves the game object for the left hand controller.

GetControllerRightHand/0

```
public static GameObject GetControllerRightHand()
```

- Parameters

- *none*
- Returns
  - `GameObject` - The right hand controller.

The GetControllerRightHand method retrieves the game object for the right hand controller.

IsControllerOfHand/2

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

- Parameters
  - `GameObject checkController` - The actual controller object that is being checked.
  - `ControllerHand hand` - The 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.

PlayAreaTransform/0

```
public static Transform PlayAreaTransform()
```

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

The PlayAreaTransform method is used to retrieve the transform for the play area 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

- **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

`VRTK/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 `VRTK/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 `VRTK/Scripts/Helper/CurveGenerator.cs` and was heavily inspired by the tutorial and code from Catlike Coding.

Inspector Parameters

- **Pointer Length:** The length of the projected forward pointer beam, this is basically the distance able to point from the controller position.
- **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 radius, the larger the objects will be.
- **Pointer Cursor Match Target Rotation:** The pointer cursor will be rotated to match the angle of the target surface if this is true, if it is false then the pointer cursor will always be horizontal.
- **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.
- **Beam Height Limit Angle:** The maximum angle in degrees of the controller before the beam curve height is restricted. A lower angle setting will prevent the beam being projected high into the sky and curving back down.
- **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

`VRTK/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.

`VRTK/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.

`VRTK/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.

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.
- **Activation Mode:** Determines when the UI pointer should be active.
- **Attempt Click On Deactivate:** Determines whether the UI click action should be triggered when the pointer is deactivated. If the pointer is hovering over a clickable element then it will invoke the click action on that element.
- **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.
- **Canvas Tag Or Script List Policy:** A specified VRTK_TagOrScriptPolicyList to use to determine whether any world canvases will be acted upon by the UI Pointer. If a list is provided then the 'Ignore Canvas With Tag Or Class' parameter will be ignored.

Class Variables

- `public enum ActivationMethods` - Methods of activation.
  - `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.

Unity Events

Adding the `VRTK_UIPointer_UnityEvents` component to `VRTK_UIPointer` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnUIPointerElementEnter` - Emits the UIPointerElementEnter class event.
- `OnUIPointerElementExit` - Emits the UIPointerElementExit class event.

Event Payload

- `uint controllerIndex` - The index of the controller that was used.
- `bool isActive` - The state of whether the UI Pointer is currently active or not.
- `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 initialise 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

`VRTK/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 Teleport (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.

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.
- **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 object 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.

- **Target Tag Or Script List Policy:** A specified VRTK_TagOrScriptPolicyList to use to determine whether destination targets will be acted upon by the Teleporter. If a list is provided then the 'Ignore Target With Tag Or Class' parameter will be ignored.
- **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.

Unity Events

Adding the `VRTK_BasicTeleport_UnityEvents` component to `VRTK_BasicTeleport` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnTeleporting` - Emits the Teleporting class event.
- `OnTeleported` - Emits the Teleported class event.

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

InitDestinationSetListener/2

```
public void InitDestinationSetListener(GameObject markerMaker, bool register)
```

- Parameters
  - `GameObject markerMaker` - The game object that is used to generate destination marker events, such as a controller.
  - `bool register` - Determines whether to register or unregister the listeners.
- 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.

ToggleTeleportEnabled/1

```
public void ToggleTeleportEnabled(bool state)
```

- Parameters
  - `bool state` - Toggles whether the teleporter is enabled or disabled.
- Returns
  - *none*

The ToggleTeleportEnabled method is used to determine whether the teleporter will initiate a teleport on a destination set event, if the state is true then the teleporter will work as normal, if the state is false then the teleporter will not be operational.

Example

`VRTK/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 Adjust Teleport (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.

Inspector Parameters

- **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.
- **Play Space Fall Restriction:** An additional check to see if the play space fall should take place. If the selected restrictor is still over the current floor then the play space fall will not occur. Works well for being able to lean over ledges and look down. Only works for falling down not teleporting

up.
- **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.
- **Floor Height Tolerance:** The amount the `y` position needs to change by between the current floor `y` position and the previous floor `y` position before a change in floor height is considered to have occurred. A higher value here will mean that a `Drop To Floor` teleport event will be less likely to happen if the `y` of the floor beneath the user hasn't changed as much as the given threshold.

Class Variables

- `public enum FallingRestrictors` - Options for testing if a play space fall is valid
  - `No_Restriction` - Always play space fall when the headset is no longer over the current standing object.
  - `Left_Controller` - Don't play space fall if the Left Controller is still over the current standing object even if the headset isn't.
  - `Right_Controller` - Don't play space fall if the Right Controller is still over the current standing object even if the headset isn't.
  - `Either_Controller` - Don't play space fall if Either Controller is still over the current standing object even if the headset isn't.
  - `Both_Controllers` - Don't play space fall only if Both Controllers are still over the current standing object even if the headset isn't.

Example

`VRTK/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.

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

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

Headset Collision (VRTK_HeadsetCollision)

Overview

The purpose of the Headset Collision is to detect when the user's VR headset collides with another

game object.

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

Inspector Parameters

- **Ignore Target With Tag Or Class:** A string that specifies an object Tag or the name of a Script attached to an object and will be ignored on headset collision.
- **Target Tag Or Script List Policy:** A specified VRTK_TagOrScriptPolicyList to use to determine whether any objects will be acted upon by the Headset Collision. If a list is provided then the 'Ignore Target With Tag Or Class' parameter will be ignored.

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.

Unity Events

Adding the `VRTK_HeadsetCollision_UnityEvents` component to `VRTK_HeadsetCollision` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnHeadsetCollisionDetect` - Emits the HeadsetCollisionDetect class event.
- `OnHeadsetCollisionEnded` - Emits the HeadsetCollisionEnded class event.

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).

Class Methods

IsColliding/0

```
public virtual bool IsColliding()
```

- Parameters
  - *none*
- Returns
  - `bool` - Returns true if the headset is currently colliding with a valid game object.

The IsColliding method is used to determine if the headset is currently colliding with a valid game object and returns true if it is and false if it is not colliding with anything or an invalid game object.

Example

`VRTK/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.

Headset Fade (VRTK_HeadsetFade)

Overview

The purpose of the Headset Fade is to change the colour of the headset view to a specified colour over a given duration and to also unfade it back to being transparent. The `Fade` and `Unfade` methods can only be called via another script and this Headset Fade script does not do anything on initialisation to fade or unfade the headset view.

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

Class Events

- `HeadsetFadeStart` - Emitted when the user's headset begins to fade to a given colour.
- `HeadsetFadeComplete` - Emitted when the user's headset has completed the fade and is now fully at the given colour.
- `HeadsetUnfadeStart` - Emitted when the user's headset begins to unfade back to a transparent colour.
- `HeadsetUnfadeComplete` - Emitted when the user's headset has completed unfading and is now fully transparent again.

Unity Events

Adding the `VRTK_HeadsetFade_UnityEvents` component to `VRTK_HeadsetFade` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnHeadsetFadeStart` - Emits the HeadsetFadeStart class event.
- `OnHeadsetFadeComplete` - Emits the HeadsetFadeComplete class event.
- `OnHeadsetUnfadeStart` - Emits the HeadsetUnfadeStart class event.
- `OnHeadsetUnfadeComplete` - Emits the HeadsetUnfadeComplete class event.

Event Payload

- `float timeTillComplete` - A float that is the duration for the fade/unfade process has remaining.
- `Transform currentTransform` - The current Transform of the object that the Headset Fade script is attached to (Camera).

Class Methods

IsFaded/0

```
public virtual bool IsFaded()
```

- Parameters
  - *none*
- Returns
  - `bool` - Returns true if the headset is currently fading or faded.

The IsFaded method returns true if the headset is currently fading or has completely faded and returns false if it is completely unfaded.

IsTransitioning/0

```
public virtual bool IsTransitioning()
```

- Parameters
  - *none*
- Returns
  - `bool` - Returns true if the headset is currently in the process of fading or unfading.

The IsTransitioning method returns true if the headset is currently fading or unfading and returns false if it is completely faded or unfaded.

Fade/2

```
public virtual void Fade(Color color, float duration)
```

- Parameters
  - `Color color` - The colour to fade the headset view to.
  - `float duration` - The time in seconds to take to complete the fade transition.
- Returns
  - *none*

The Fade method initiates a change in the colour of the headset view to the given colour over a

given duration.

Unfade/1

```
public virtual void Unfade(float duration)
```

- Parameters
  - `float duration` - The time in seconds to take to complete the unfade transition.
- Returns
  - *none*

The Unfade method initiates the headset to change colour back to a transparent colour over a given duration.

Example

`VRTK/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.

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.

The Headset Collision Fade uses a composition of the Headset Collision and Headset Fade scripts to derive the desired behaviour.

> **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 object and will prevent the object from fading the headset view on collision.
- **Target Tag Or Script List Policy:** A specified VRTK_TagOrScriptPolicyList to use to determine whether any objects will be acted upon by the Headset Collision Fade. If a list is provided then the 'Ignore Target With Tag Or Class' parameter will be ignored.

Example

`VRTK/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.

Teleport Disable On Headset Collision (VRTK_TeleportDisableOnHeadsetCollision)

Overview

The purpose of the Teleport Disable On Headset Collision script is to detect when the headset is colliding with a valid object and prevent teleportation from working. This is to ensure that if a user is clipping their head into a wall then they cannot teleport to an area beyond the wall.

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 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 collider.
- **Ignore Grabbed Collisions:** If this is checked then any items that are grabbed with the controller will not collide with the player presence collider. 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 collider.
- **Reset Position On Collision:** If this is checked then if the Headset Collision script is present and a headset collision occurs, the CameraRig 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.

Unity Events

Adding the `VRTK_PlayerPresence_UnityEvents` component to `VRTK_PlayerPresence` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnPresenceFallStarted` - Emits the PresenceFallStarted class event.
- `OnPresenceFallEnded` - Emits the PresenceFallEnded class event.

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.

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

`VRTK/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 Walking (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

- **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.
- **Move On Button Press:** If a button is defined then movement will only occur when the specified button is being held down and the touchpad axis changes.
- **Device For Direction:** The direction that will be moved in is the direction of this device.

Example

`VRTK/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.

Room Extender (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 attached 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.
- **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.

Class Variables

- `public enum MovementFunction` - Movement methods.
  - `Nonlinear` - Moves the head with a non-linear drift movement.
  - `LinearDirect` - Moves the headset in a direct linear movement.

Example

`VRTK/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 centre 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.

The highlighting of an Interactable Object is defaulted to use the `VRTK_MaterialColorSwapHighlighter` if no other highlighter is applied to the Object.

Inspector Parameters

- **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 colour (for instance on the `VRTK_InteractTouch` script).
- **Rumble On Touch:** The haptic feedback on the controller can be triggered upon touching the object, the `Strength` denotes the strength of the pulse, the `Duration` denotes the length of time.
- **Allowed Touch Controllers:** Determines which controller can initiate a touch action.
- **Hide Controller On Touch:** Optionally override the controller setting.
- **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.

- **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.
- **Grab Override Button:** If this is set to `Undefined` then the global grab alias button will grab the object, setting it to any other button will ensure the override button is used to grab this specific interactable object.
- **Rumble On Grab:** The haptic feedback on the controller can be triggered upon grabbing the object, the `Strength` denotes the strength of the pulse, the `Duration` denotes the length of time.
- **Allowed Grab Controllers:** Determines which controller can initiate a grab action.
- **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 its central point. Not required for `Precision Snap`.
- **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 its central point. Not required for `Precision Snap`.
- **Hide Controller On Grab:** Optionally override the controller setting.
- **Stay Grabbed On Teleport:** If this is checked then the object will stay grabbed to the controller when a teleport occurs. If it is unchecked then the object will be released when a teleport occurs.
- **Grab Attach Mechanic:** This determines how the grabbed item will be attached to the controller when it is grabbed.
- **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. Only required for `Fixed Joint`, `Spring Joint`, `Track Object` and `Rotator Track`.
- **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. Only required for `Spring Joint`.
- **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. Only required for `Spring Joint`.

- **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.
- **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 is currently being grabbed.
- **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.
- **Use Override Button:** If this is set to `Undefined` then the global use alias button will use the object, setting it to any other button will ensure the override button is used to use this specific interactable object.
- **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 `Strength` denotes the strength of the pulse, the `Duration` denotes the length of time.
- **Allowed Use Controllers:** Determines which controller can initiate a use action.
- **Hide Controller On Use:** Optionally override the controller setting.

Class Variables

- `public enum GrabAttachType` - Types of grab attachment.
  - `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 meaning 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 resistance 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` - 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` - 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.

- `public enum AllowedController` - Allowed controller type.
  - `Both` - Both controllers are allowed to interact.
  - `Left_Only` - Only the left controller is allowed to interact.
  - `Right_Only` - Only the right controller is allowed to interact.
- `public enum ControllerHideMode` - Hide controller state.
  - `Default` - Use the hide settings from the controller.
  - `OverrideHide` - Hide the controller when interacting, overriding controller settings.
  - `OverrideDontHide` - Don't hide the controller when interacting, overriding controller settings.
- `public int usingState` - The current using state of the object. `0` not being used, `1` being used. Default: `0`

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.

Unity Events

Adding the `VRTK_InteractableObject_UnityEvents` component to `VRTK_InteractableObject` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnTouch` - Emits the InteractableObjectTouched class event.
- `OnUntouch` - Emits the InteractableObjectUntouched class event.
- `OnGrab` - Emits the InteractableObjectGrabbed class event.
- `OnUngrab` - Emits the InteractableObjectUngrabbed class event.
- `OnUse` - Emits the InteractableObjectUsed class event.
- `OnUnuse` - Emits the InteractableObjectUnused class event.

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.

- 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.

AttachIsKinematicObject/0

```
public bool AttachIsKinematicObject()
```

- Parameters
  - *none*
- Returns
  - `bool` - Is true if the grab attach mechanic sets the object to a kinematic state on grab.

The AttachIsKinematicObject method is used to determine if the object has kinematics turned on at the point of grab.

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.

AttachIsUnthrowableObject/0

```
public bool AttachIsUnthrowableObject()
```

- Parameters
  - *none*
- Returns
  - `bool` - Is true if the grab attach mechanic is of a type that shouldn't be considered thrown when released.

The AttachIsUnthrowableObject method is used to determine if the object is using one of the grab

types that should not be thrown when released.

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

`VRTK/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.

`VRTK/Examples/013_Controller_UsingAndGrabbingMultipleObjects` shows multiple 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 multiple items can be turned on at the same time.

Interact Touch (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.

Unity Events

Adding the `VRTK_InteractTouch_UnityEvents` component to `VRTK_InteractTouch` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnControllerTouchInteractableObject` - Emits the ControllerTouchInteractableObject class event.
- `OnControllerUntouchInteractableObject` - Emits the ControllerUntouchInteractableObject class event.

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

`VRTK/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.

Interact Grab (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.

Unity Events

Adding the `VRTK_InteractGrab_UnityEvents` component to `VRTK_InteractGrab` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnControllerGrabInteractableObject` - Emits the ControllerGrabInteractableObject class event.
- `OnControllerUngrabInteractableObject` - Emits the ControllerUngrabInteractableObject class event.

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

`VRTK/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.

`VRTK/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.

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

Interact Use (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.

Unity Events

Adding the `VRTK_InteractUse_UnityEvents` component to `VRTK_InteractUse` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnControllerUseInteractableObject` - Emits the ControllerUseInteractableObject class event.
- `OnControllerUnuseInteractableObject` - Emits the ControllerUnuseInteractableObject class event.

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

`VRTK/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.

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

Object Auto Grab (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

`VRTK/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.

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.

Unity Events

Adding the `VRTK_PlayerClimb_UnityEvents` component to `VRTK_PlayerClimb` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnPlayerClimbStarted` - Emits the PlayerClimbStarted class event.
- `OnPlayerClimbEnded` - Emits the PlayerClimbEnded class event.

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

`VRTK/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.

Dash Teleport (VRTK_DashTeleport)

> extends VRTK_HeightAdjustTeleport

Overview

The dash teleporter extends the height adjust teleporter and allows to have the `[CameraRig]` dashing to a new teleport location.

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

The basic principle is to dash for a very short amount of time, to avoid sim sickness. The default value is 100 miliseconds. This value is fixed for all normal and longer distances. When the distances

get very short the minimum speed is clamped to 50 mps, so the dash time becomes even shorter.

The minimum distance for the fixed time dash is determined by the minSpeed and normalLerpTime values, if you want to always lerp with a fixed mps speed instead, set the normalLerpTime to a high value. Right before the teleport a capsule is cast towards the target and registers all colliders blocking the way. These obstacles are then broadcast in an event so that for example their gameobjects or renderers can be turned off while the dash is in progress.

Inspector Parameters

- **Normal Lerp Time:** The fixed time it takes to dash to a new position.
- **Min Speed Mps:** The minimum speed for dashing in meters per second.
- **Capsule Top Offset:** The Offset of the CapsuleCast above the camera.
- **Capsule Bottom Offset:** The Offset of the CapsuleCast below the camera.
- **Capsule Radius:** The radius of the CapsuleCast.

Class Events

- `WillDashThruObjects` - Emitted when the CapsuleCast towards the target has found that obstacles are in the way.
- `DashedThruObjects` - Emitted when obstacles have been crossed and the dash has ended.

Unity Events

Adding the `VRTK_DashTeleport_UnityEvents` component to `VRTK_DashTeleport` object allows access to `UnityEvents` that will react identically to the Class Events.

- `OnWillDashThruObjects` - Emits the WillDashThruObjects class event.
- `OnDashedThruObjects` - Emits the DashedThruObjects class event.

Event Payload

- `RaycastHit[] hits` - An array of objects that the CapsuleCast has collided with.

Example

`SteamVR_Unity_Toolkit/Examples/038_CameraRig_DashTeleport` shows how to turn off the mesh renderers of objects that are in the way during the dash.

Tag Or Script Policy List (VRTK_TagOrScriptPolicyList)

Overview

The Tag Or Script Policy List allows to create a list of either tag names or script names that can be checked against to see if another operation is permitted.

A number of other scripts can use a Tag Or Script Policy List to determine if an operation is permitted based on whether a game object has a tag applied or a script component on it.

For example, the Teleporter scripts can ignore game object targets as a teleport location if the game object contains a tag that is in the identifiers list and the policy is set to ignore.

Or the teleporter can only allow teleport to targets that contain a tag that is in the identifiers list and the policy is set to include.

Add the Tag Or Script Policy List script to a game object (preferably the same component utilising the list) and then configure the list accordingly.

Then in the component that has a Tag Or Script Policy List paramter (e.g. BasicTeleporter has `Target Tag Or Script List Policy`) simply select the list that has been created and defined.

Inspector Parameters

- **Operation:** The operation to apply on the list of identifiers.
- **Check Type:** The element type on the game object to check against.

Class Variables

- `public enum OperationTypes` - The operation to apply on the list of identifiers.
  - `Ignore` - Will ignore any game objects that contain either a tag or script component that is included in the identifiers list.
  - `Include` - Will only include game objects that contain either a tag or script component that is included in the identifiers list.
- `public enum CheckTypes` - The types of element that can be checked against.
  - `Tag` - The tag applied to the game object.
  - `Script` - A script component added to the game object.
  - `Tag_Or_Script` - Either a tag applied to the game object or a script component added to the game object.

Class Methods

Find/1

```
public bool Find(GameObject obj)
```

- Parameters
  - `GameObject obj` - The game object to check if it has a tag or script that is listed in the identifiers list.
- Returns
  - `bool` - If the operation is `Ignore` and the game object is matched by an identifier from the list then it returns true. If the operation is `Include` and the game object is not matched by an identifier from the list then it returns true.

The Find method performs the set operation to determine if the given game object contains one of the identifiers on the set check type. For instance, if the Operation is `Ignore` and the Check Type is `Tag` then the Find method will attempt to see if the given game object has a tag that matches one of the identifiers.

Simulating Headset Movement (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:** An optional game object marking the position and rotation at which the camera should be initially placed.

Adaptive Quality (VRTK_AdaptiveQuality)

Overview

Adaptive Quality dynamically changes rendering settings to maintain VR framerate while maximizing GPU utilization.

The Adaptive Quality script is attached to the `eye` object within the `[CameraRig]` prefab.

There are two goals:

- Reduce the chances of dropping frames and reprojecting
- Increase quality when there are idle GPU cycles

This script currently changes the following to reach these goals:

- Rendering resolution and viewport (aka Dynamic Resolution)

In the future it could be changed to also change the following:

- MSAA level
- Fixed Foveated Rendering
- Radial Density Masking
- (Non-fixed) Foveated Rendering (once HMDs support eye tracking)

Some shaders, especially Image Effects, need to be modified to work with the changed render scale. To fix them pass `1.0f / VRSettings.renderViewportScale` into the shader and scale all incoming UV values with it in the vertex program. Do this by using `Material.SetFloat` to set the value in the script that configures the shader.

In more detail:

- In the `.shader` file: Add a new runtime-set property value `float _InverseOfRenderViewportScale` and add `vertexInput.texcoord *= _InverseOfRenderViewportScale` to the start of the vertex program
- In the `.cs` file: Before using the material (eg. `Graphics.Blit`) add `material.SetFloat("_InverseOfRenderViewportScale", 1.0f / VRSettings.renderViewportScale)`

Inspector Parameters

- **Active:** Toggles whether the render quality is dynamically adjusted to maintain VR framerate. If unchecked, the renderer will render at the recommended resolution provided by the current `VRDevice`.
- **Draw Debug Visualization:** Toggles whether to show the debug overlay. Each square represents a different level on the quality scale. Levels increase from left to right, and the first green box that is lit above represents the recommended render target resolution provided by the current `VRDevice`. The yellow boxes represent resolutions below the recommended render target resolution. The currently lit box becomes red whenever the user is likely seeing reprojection in the HMD since the application isn't maintaining VR framerate. If lit, the box all the way on the left is almost always lit red because it represents the lowest render scale with reprojection on.
- **Responds To Keyboard Shortcuts:** Toggles whether to allow keyboard shortcuts to control this script.
  - The supported shortcuts are:
  - `Shift+F1`: Toggle debug visualization on/off
  - `Shift+F2`: Toggle usage of override render scale on/off
  - `Shift+F3`: Decrease override render scale level
  - `Shift+F4`: Increase override render scale level
- **Responds To Command Line Arguments:** Toggles whether to allow command line arguments to control this script at startup of the standalone build.
  - The supported command line arguments all begin with '-' and are:

    - `-noaq`: Disable adaptive quality
    - `-aqminscale X`: Set minimum render scale to X
    - `-aqmaxscale X`: Set maximum render scale to X
    - `-aqmaxres X`: Set maximum render target dimension to X
    - `-aqfillratestep X`: Set render scale fill rate step size in percent to X (X from 1 to 100)
    - `-aqoverride X`: Set override render scale level to X
    - `-vrdebug`: Enable debug visualization
    - `-msaa X`: Set MSAA level to X
- **Msaa Level:** The MSAA level to use.
- **Minimum Render Scale:** The minimum allowed render scale.
- **Maximum Render Scale:** The maximum allowed render scale.
- **Maximum Render Target Dimension:** The maximum allowed render target dimension. This puts an upper limit on the size of the render target regardless of the maximum render scale.
- **Render Scale Fill Rate Step Size In Percent:** The fill rate step size in percent by which the render scale levels will be calculated.
- **Override Render Scale:** Toggles whether to override the used render scale level.
- **Override Render Scale Level:** The render scale level to override the current one with.

Class Variables

- `public readonly ReadOnlyCollection<float> renderScales` - All the calculated render scales. The elements of this collection are to be interpreted as modifiers to the recommended render target resolution provided by the current `VRDevice`.
- `public float currentRenderScale` - The current render scale. A render scale of 1.0 represents the recommended render target resolution provided by the current `VRDevice`.
- `public Vector2 defaultRenderTargetResolution` - The recommended render target resolution provided by the current `VRDevice`.
- `public Vector2 currentRenderTargetResolution` - The current render target resolution.

Class Methods

RenderTargetResolutionForRenderScale/1

```
public static Vector2 RenderTargetResolutionForRenderScale(float renderScale)
```

- Parameters
    - `float renderScale` - The render scale to calculate the render target resolution with.
- Returns
    - `Vector2` - The render target resolution for `renderScale`.

Calculates and returns the render target resolution for a given render scale.

BiggestAllowedMaximumRenderScale/0

```
public float BiggestAllowedMaximumRenderScale()
```

- Parameters
  - *none*
- Returns
  - `float` - The biggest allowed maximum render scale.

Calculates and returns the biggest allowed maximum render scale to be used for `maximumRenderScale` given the current `maximumRenderTargetDimension`.

ToString/0

```
public override string ToString()
```

- Parameters
  - *none*
- Returns
  - `string` - The summary.

A summary of this script by listing all the calculated render scales with their corresponding render target resolution.

Example

`VRTK/Examples/039_CameraRig_AdaptiveQuality` displays the frames per second in the centre of the headset view. The debug visualization of this script is displayed near the top edge 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 script.

3D Controls (VRTK/Scripts/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 `VRTK_Control` abstract class which provides common methods and

events.

- Control
- Button
- Chest
- Door
- Drawer
- Knob
- Lever
- Spring Lever
- Slider
- Content Handler

Control (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 Variables

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

Unity Events

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

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.

Button (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.

Unity Events

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

Example

`VRTK/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.

Chest (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

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

Door (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. Only works if either inward or outward is selected, not both.

Example

`VRTK/Examples/025_Controls_Overview` shows a selection of door types, from a normal door and trapdoor, to a door with a cat-flap in the middle.

Drawer (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 drawer 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

`VRTK/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.

Knob (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

`VRTK/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.

Lever (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 HingeJoint 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

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

Spring Lever (VRTK_Spring_Lever)

extends VRTK_Lever

Overview

This script extends VRTK_Lever to add spring force toward whichever end of the lever's range it is closest to.

The script will instantiate the required Rigidbody, Interactable and HingeJoint 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

- **Spring Strength:** Strength of the spring force that will be applied toward either end of the lever's range.

Slider (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.
- **Detect Min Max:** Automatically detect the minimum and maximum positions.
- **Min Point:** The minimum point on the slider.
- **Max Point:** The maximum point on the slider.

Example

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

Content Handler (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 re-parent 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

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