VRIL - Virtual Interaction Library

Installation and In-Depth Description of VRIL

Author:
Andreas Roither

March 4, 2019

Contents

1 Installation

VRIL can be integrated by either importing the package from the Asset Store or by manually downloading the latest version from Github VRIL - Github and placing it in the Asset folder in your project.

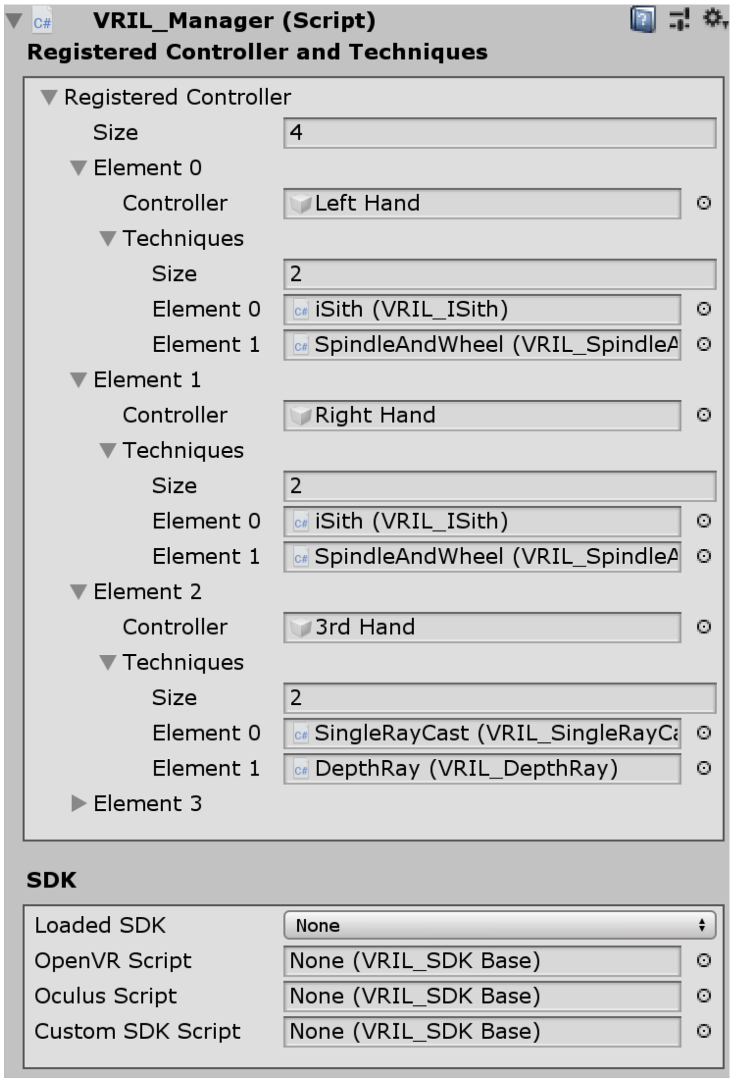
2 Quick Start

Example scenes
To start as soon as possible please look at the included example scenes under *VRIL/ExampleScenes*. In this folder scenes with the implemented techniques have been set up. There are two different scene examples: Scenes for keyboard and mouse, and SteamVR. To use the SteamVR example scenes please install the SteamVR package from the Unity Asset Store first and then generate default actions via the SteamVR Input window.

Custom scenes
To set up a new scene by yourself, please attach the *VRIL_Manager* script to any game object you want in the scene. Afterwards assign game objects as controllers. You can assign interaction techniques to these controllers, assigning multiple techniques is possible. Please attach any technique that you want to use in your scene to a game object in the scene and add an interaction technique script to a specific controller game object at the manager. The interaction technique script has to be derived from the *VRIL_InteractionTechniqueBase*. Interaction techniques have button mapping available to them, which has to be set if a technique has been newly added to the scene. Lastly, if an object should be interactable, the *VRIL_Interactable* script has to be attached to the game object.

In **Figure 1** below you can see an example how registered controllers and their techniques at the manager look like in the Unity Inspector.

In **Figure 2** the example technique iSith shows how a technique can be set up.


VRIL_Manager (Script)

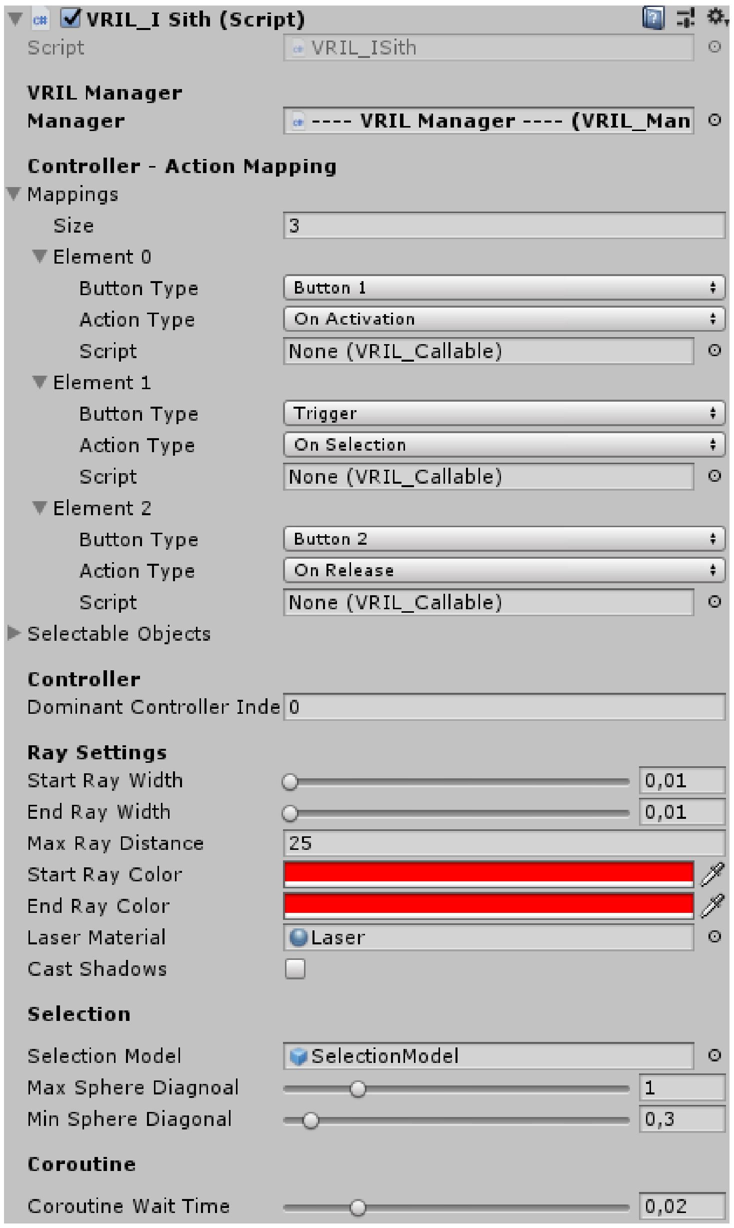
Registered Controller and Techniques

Registered Controller
Size 4
Element 0
Controller Left Hand
Techniques
Size 2
Element 0 iSith (VRIL_ISith)
Element 1 SpindleAndWheel (VRIL_SpindleA
Element 1
Controller Right Hand
Techniques
Size 2
Element 0 iSith (VRIL_ISith)
Element 1 SpindleAndWheel (VRIL_SpindleA
Element 2
Controller 3rd Hand
Techniques
Size 2
Element 0 SingleRayCast (VRIL_SingleRayCa
Element 1 DepthRay (VRIL_DepthRay)
Element 3

SDK

Loaded SDK None
OpenVR Script None (VRIL_SDK Base)
Oculus Script None (VRIL_SDK Base)
Custom SDK Script None (VRIL_SDK Base)

Figure 1: VRIL Manager



VRIL_I Sith (Script)

Script	VRIL_ISith
VRIL Manager	
Manager	---- VRIL Manager ---- (VRIL_Man
Controller - Action Mapping	
Mappings	
Size	3
Element 0	
Button Type	Button 1
Action Type	On Activation
Script	None (VRIL_Callable)
Element 1	
Button Type	Trigger
Action Type	On Selection
Script	None (VRIL_Callable)
Element 2	
Button Type	Button 2
Action Type	On Release
Script	None (VRIL_Callable)
Selectable Objects	
Controller	
Dominant Controller Inde	0
Ray Settings	
Start Ray Width	0,01
End Ray Width	0,01
Max Ray Distance	25
Start Ray Color	🟥
End Ray Color	🟥
Laser Material	Laser
Cast Shadows	☐
Selection	
Selection Model	SelectionModel
Max Sphere Diagnoal	1
Min Sphere Diagonal	0,3
Coroutine	
Coroutine Wait Time	0,02

Figure 2: iSith Technique

3 In-Depth

For a more detailed explanation how the components work and UML diagrams please read this chapter.

3.1 Concept Design

Before you can use the library by your own you have to understand how the library works first. **Figure 3** shows the conceptual design of the library.

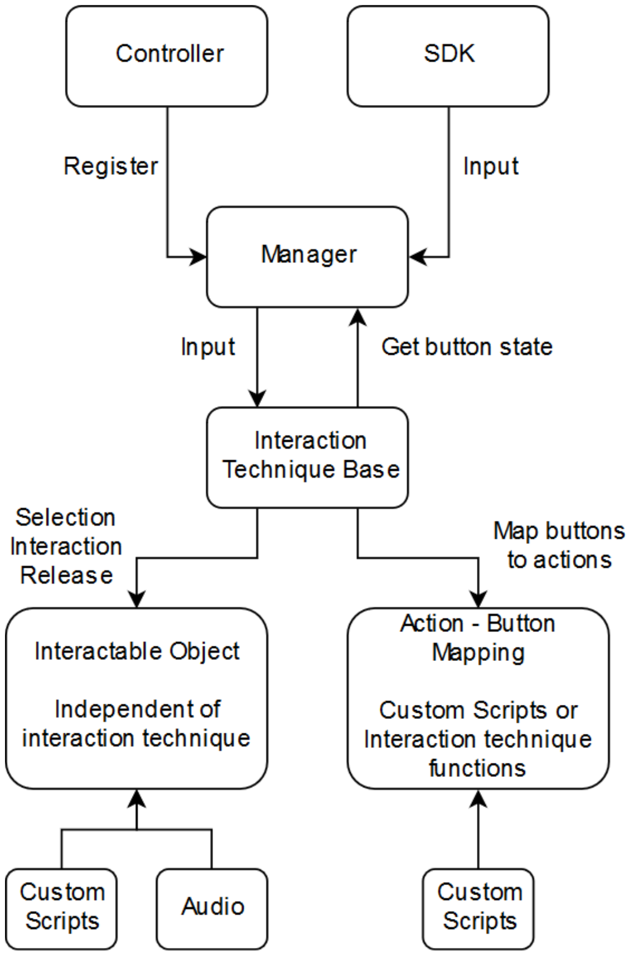


Figure 3: Concept Design

3.2 Architecture

The manager has a list of registered techniques and controllers. A controller is mapped to a specific interaction technique which allows for diverse use of multiple controllers with multiple interaction techniques. Additionally, it also contains a list of SDKs with the possibility to request a button state of the registered controller if additional input for an interaction technique is ever needed. A registered controller can be any desired game object, the registered technique for the controller has to be derived from the interaction technique base. A registered controller does no literally have to be a controller, it is just the game object that is used by the interaction technique to accomplish its task. For example, the camera can be used for a gaze-based system as a controller as well.

The interaction technique base has a list of registered controllers that is provided by the manager since many interaction techniques depend on the position and rotation of the controller. Additionally, the base class has a list of action mappings. This allows the user to map a specific controller action to one of the available functions of the base class or a custom script.

The interactable object has a variety of options should it be selected, released or manipulated. These options include playing audio, using a custom script or moving to a desired position like the controller position.

3.3 Interaction Library Components

3.3.1 Manager

The *VRIL_Manager* (UML **Figure 4**) acts as an intermediate between any SDK or custom script used and the interaction technique. As the interaction techniques should be reusable even if the hardware changes, the manager offers the possibility to use custom SDK scripts. Custom SDK scripts have to implement the interface *VRIL_SDKBase* which provides a function to return the current state of a button which can be requested from an interaction technique.

All used controllers or rather the representing game objects register at the manager. To further specify which controller uses which interaction technique a list of said techniques can be added to each controller. One interaction technique can be referenced multiple times and will work for multiple controllers (the mappings have to be adjusted however). All techniques will retrieve the registered controllers at the beginning of their lifetime and save them. Since the manager has a reference to all registered techniques, controller actions can be relayed by the manager as well. The function *OnControllerInteraction* can be used to relay the pressed button as well as information in which form the interaction happened. Possible options are:

- None
- Pressed
- NearTouch
- Touched
- PressedOnce
- TouchedOnce
- Released

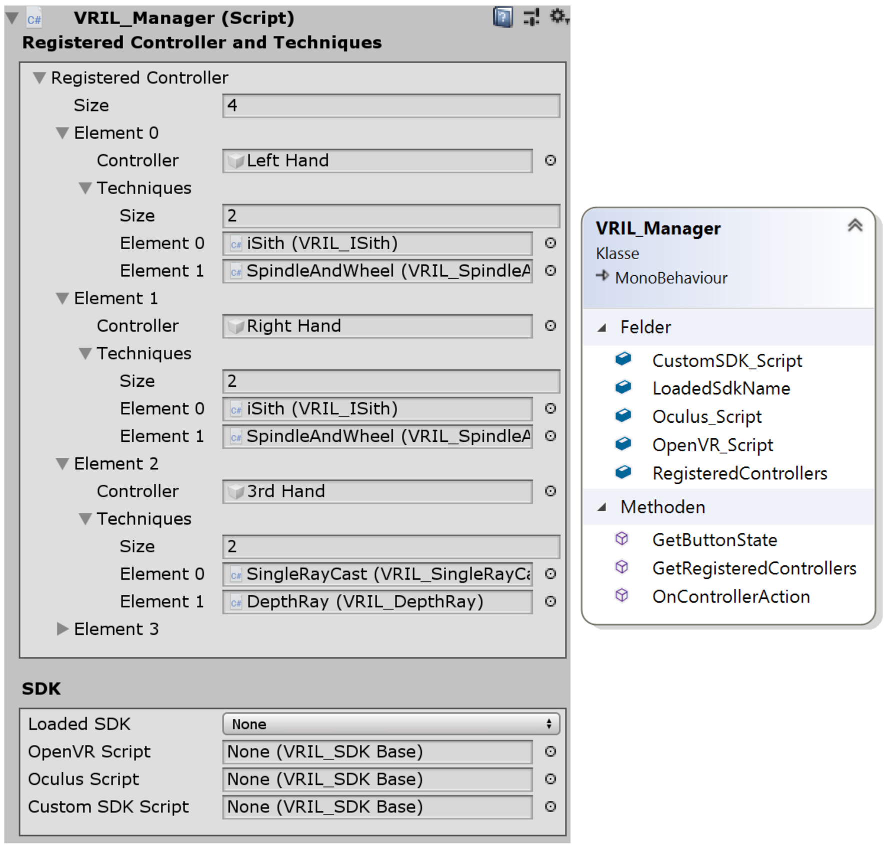
Available options for a controller interaction include:

- None
- Button1
- Button2
- ButtonStart
- Grip
- Trigger
- Touchpad

If a controller action happened (function OnControllerInteraction was called) the manager then proceeds to pass on the specified action to every interaction technique script attached to the specified controller.

The interaction technique can then react to the input. Techniques that are not registered at the controller will not get notified. The manager also provides a function to check a button state for a specific controller as mentioned above. This function, however, is not necessarily needed by every interaction technique but can be used to check for additional input. Lastly, the manager implements a function that returns a list of registered controllers for an interaction technique since most techniques need some form of position or rotation to work with (for example iSith or ray-casting) in order to function properly.

As more and more properties are added to a script the user interface will get cluttered soon. In order to provide better usability, the manager has a custom interface with sections and boxes to ease the use of the manager as seen in **Figure 4**. The custom interface may not play a big role for the manager as the manager does not have that many properties, but the custom interface will be crucial for *VRIL_Interactable* objects.



```
VRIL_Manager (Script)
Registered Controller and Techniques
▼ Registered Controller
    Size                4
  ▼ Element 0
      Controller        Left Hand
    ▼ Techniques
        Size            2
        Element 0       iSith (VRIL_ISith)
        Element 1       SpindleAndWheel (VRIL_SpindleA
  ▼ Element 1
      Controller        Right Hand
    ▼ Techniques
        Size            2
        Element 0       iSith (VRIL_ISith)
        Element 1       SpindleAndWheel (VRIL_SpindleA
  ▼ Element 2
      Controller        3rd Hand
    ▼ Techniques
        Size            2
        Element 0       SingleRayCast (VRIL_SingleRayCa
        Element 1       DepthRay (VRIL_DepthRay)
  ▶ Element 3

SDK
Loaded SDK              None
OpenVR Script           None (VRIL_SDK Base)
Oculus Script           None (VRIL_SDK Base)
Custom SDK Script       None (VRIL_SDK Base)
```

```
VRIL_Manager
Klasse
→ MonoBehaviour

Felder
  CustomSDK_Script
  LoadedSdkName
  Oculus_Script
  OpenVR_Script
  RegisteredControllers
Methoden
  GetButtonState
  GetRegisteredControllers
  OnControllerAction
```

Figure 4: VRIL Manager UML Diagram

3.3.2 Interaction Technique Base

The interaction technique base (UML **Figure** 16) is the base class for all other interaction techniques and implements basic functionality to communicate with the registered manager and interactable objects. All interaction techniques that derive from this class should call the *Initialize* function at the start. Like the manager, the base class offers a list of registered controllers that will be requested from the manager when the *Initialize* function is called. Controllers that are requested from the manager are returned in the order that they are registered. This has to be kept in mind when working with the *DominantControllerIndex* property that specifies which controller is the primarily used controller (left or right handed user).

Any controller action relayed from the manager can be mapped to the functions of the base class or even a custom script.
These functions include:

- OnActivation
  is used to start or enable an interaction technique and should not be used to initialize a technique
- OnSelection
  is used to select interactable objects that are available for selection
- OnInteraction
  is used to interact with an object
- OnRelease
  is used to release all selected objects
- OnStop
  is used to stop or halt an interaction technique

The mapping is implemented by another class, the *ActionMapping* class. Mappings are saved in a list and are automatically handled by the base class. Mapping an action twice will not result in an action to be ignored which creates the possibility to call a base class function and a custom script at the same time. This allows for easier development where the base functionality of the desired mapping can be left to the base class while still adding additional options in form of the custom script. The custom script has to implement the *VRIL_Callable* interface in order for the base class to invoke the *OnCall* function.

The interaction technique base also provides basic functionality for notifying an interactable object upon selection, release or interaction. Any of these functions can be overwritten by a derived class if different actions are intended for an interaction technique. Interactable objects are game objects that have the *VRIL_Interactable* script attached to them in the Unity editor. All interactable objects can decide, based on their settings, what action should be taken for either of the previously mentioned cases should it be notified.

3.3.3 ActionMapping

The action mapping class (UML **Figure 5**) is used by the interaction technique base to map a specific controller action to a function of the interaction technique base or a custom script. If a custom script is used the function *OnCall* will be invoked. *ActionType* includes the following options:

- None
- OnActivation
- OnSelection
- OnInteraction
- OnRelease
- OnStop
- CustomScript

ButtonType includes the following options:

- None
- Button1
- Button2
- ButtonStart
- Grip
- Trigger
- Touchpad

Mapping the buttons in the right way is entirely up to the user. Since the manager just relays any controller action to the registered techniques without any checks if the right button was pressed, changing button assignments can be done at either the interaction techniques or at the SDK that registers controller actions and sends it to the manager.

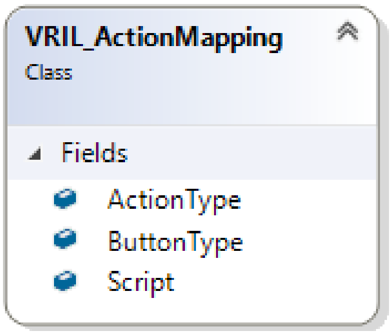


Figure 5: VRIL_ActionMapping UML diagram

3.3.4 Callable

All custom scripts that can be used by an interaction technique or an interactable object must implement the *VRIL_Callable* interface (UML **Figure 6**). The *OnCall* method will be the function that will be invoked if the custom script should be executed. Additionally, a *OnStop* method is provided if the custom script needs to cancel ongoing processes. The *OnStop* method though will only ever be called if it's mapped correctly at the interaction technique. Calling the method in any other function is possible as well.

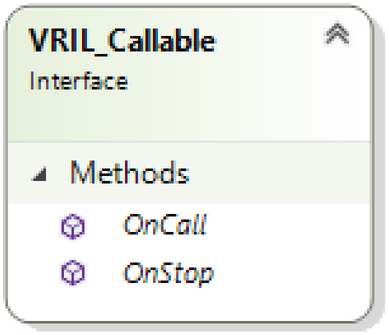


Figure 6: VRIL_Callable UML diagram

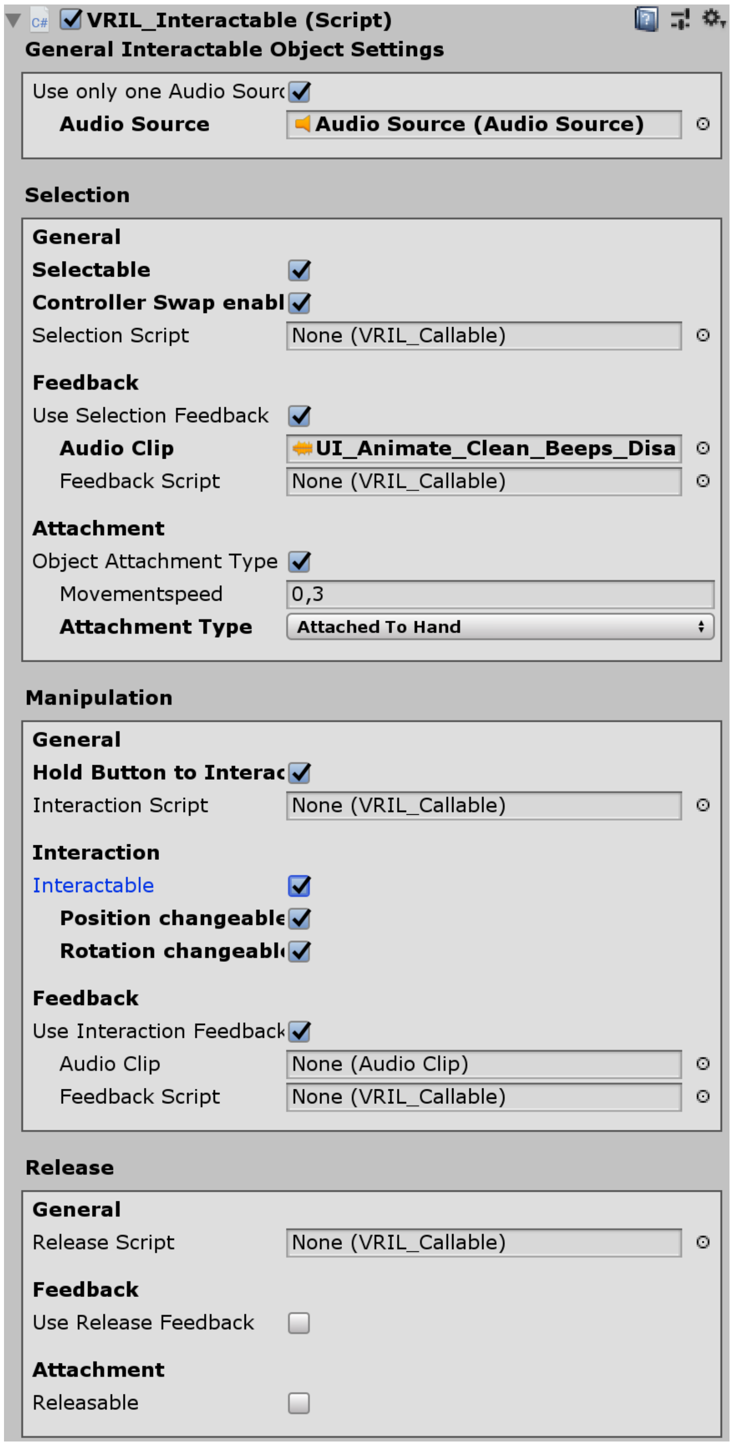
3.3.5 Interactable Object

The interactable object class (UML **Figure 17**) provides an extensive list of options for every category since an interactable object should mostly manage itself. Each category has it's own audio source, audio clip or custom script that will be played or started if the object is notified from the interacting technique. Additionally, a general audio source is provided as well but has to be selected explicitly in order for the audio source to be used. If this option is selected only one audio source will be used by the object in case any music should be played. Since an interactable object is highly customizable with many different options a new user interface in Unity is needed in order to provide better usability. The interface can be seen in **Figure 7**. All these options mentioned above belong to one of the following categories:

Selection
The selection category provides options if the object is selectable, uses selection feedback, is swappable between controllers or if the object should attach itself to a specific location. Moreover, the speed at which the object moves to its final location can be adjusted as well.

Interaction
The interaction category provides the same options as the selection category but with additional options specific to interactions. These additional options include the ability to select if the object's position or rotation can be changed. The object itself does not check at runtime if it has been moved as these options should be used by the interaction technique that manipulates the object.

Release
The release category provides the same options as the selection category. If an interactable object should be selected, the script provides functions that move and rotate the object to the desired position (new position or initial position) and play or stop the music if any audio source or audio clip is attached and the feedback option is set.



VRIL_Interactable (Script)

General Interactable Object Settings

Use only one Audio Sourc	✓
Audio Source	**Audio Source (Audio Source)**

Selection

General

Selectable	✓
Controller Swap enabl	✓
Selection Script	None (VRIL_Callable)

Feedback

Use Selection Feedback	✓
Audio Clip	**UI_Animate_Clean_Beeps_Disa**
Feedback Script	None (VRIL_Callable)

Attachment

Object Attachment Type	✓
Movementspeed	0,3
Attachment Type	Attached To Hand

Manipulation

General

Hold Button to Interac	✓
Interaction Script	None (VRIL_Callable)

Interaction

Interactable	✓
Position changeable	✓
Rotation changeabl	✓

Feedback

Use Interaction Feedback	✓
Audio Clip	None (Audio Clip)
Feedback Script	None (VRIL_Callable)

Release

General

Release Script	None (VRIL_Callable)

Feedback

Use Release Feedback	☐

Attachment

Releasable	☐

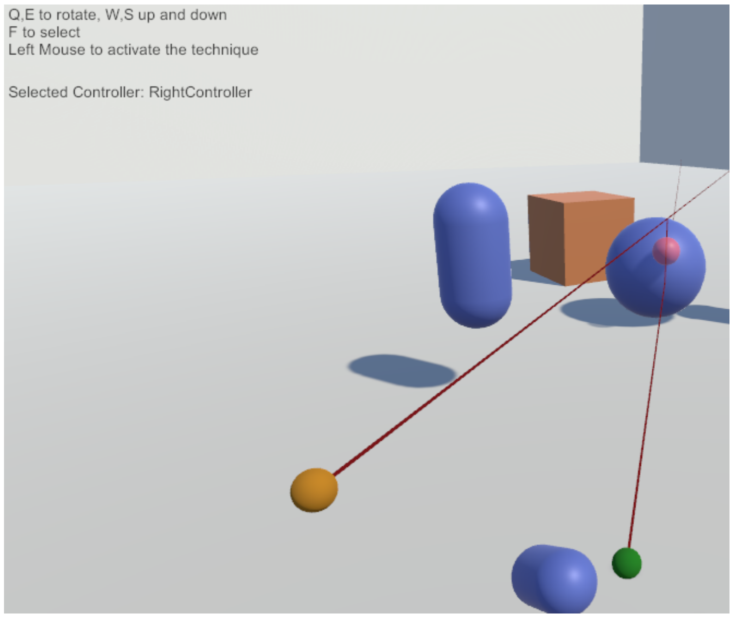
Figure 7: VRIL_Interactable object Unity interface

3.4 Example Integration

To show how the interaction library can be used, two example scripts are implemented. Scenes for each example have been set up in Unity that utilize said example scripts.

3.4.1 Keyboard and Mouse

The *VRIL_Keyboard* script allows the user to move, rotate, select and activate a technique. This script simply remaps any input from the keyboard or mouse and notifies the *VRIL_Manger*. Additionally, the script also shows which controller is currently selected which can be seen in **Figure** 8 on the top left side.



Q,E to rotate, W,S up and down
F to select
Left Mouse to activate the technique

Selected Controller: RightController

Figure 8: VRIL_Keyboard UI

3.4.2 SteamVR

The SteamVR[1] example script utilizes the default actions that are generated on start. Actions were introduced with SteamVR 2.0 and allow developers to subscribe to an action instead of checking input from the controller manually. To use said actions or create new ones, the “SteamVR Input“ window in Unity offers a variety of options (package has to be installed first, as well as the SteamVR software). If these actions have been generated, they can be bound to a specific controller input action like a button or a touchpad. After actions are bound and generated, developers can subscribe or attach a new listener to an action. The example script utilizes 4 of the default actions, that are auto-generated when opening the “SteamVR Input“ for the first time, and registers listeners to each one. After any input from the controller was detected, the *VRIL_Manager* will be notified. In **Figure 9**, an example for the *iSith* technique with SteamVR can be seen.

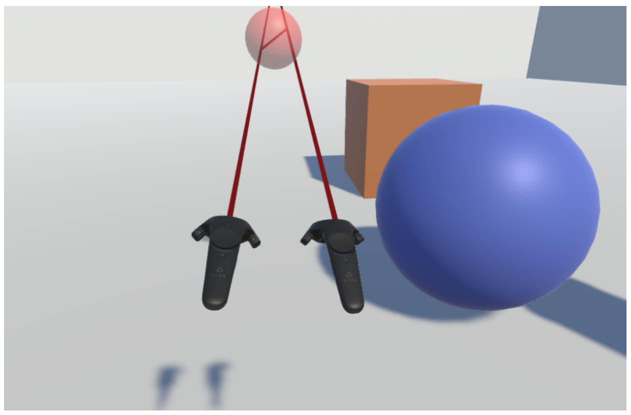


Figure 9: VRIL_SteamVR example

[1]https://steamcommunity.com/steamvr

3.5 Interaction Techniques

These are the interaction techniques that are implemented in the library. An example image demonstrates the use of the technique described.

3.5.1 Ray-Cast

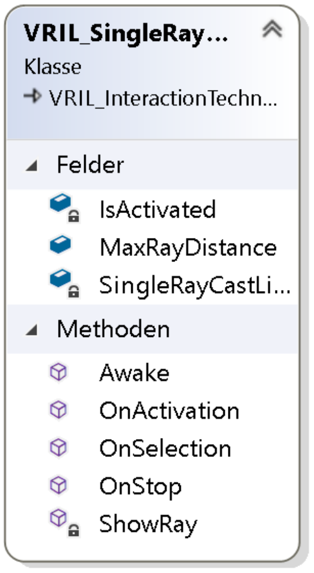
The Ray-Casting technique (earliest version mentioned by Bolt [Bol80], later again by Bowman and Hodges [BH97]) is an interaction technique that utilizes a single controller with a line to show the user what objects can be selected. Anything the line touches and is selectable can be selected. The ray-casting technique (UML **Figure 10**) is implemented by using a single line renderer[2]. By using a special attribute[3] from the Unity scripting API, a line renderer will automatically be attached to the parent game object of the script if no line renderer is present:

```
[RequireComponent(typeof(LineRenderer))]
```

The line renderer is used to create a single line moving forward from the first registered controller with a customizable maximum distance. Line width, starting and ending width as well as the color of the line can be adjusted in the automatically created line renderer script. Since the ray should be able to select interactable objects (objects that have the *VRIL_Interactable* component attached), a *Physics.Raycast*[4] from the Unity scripting API is used. The ray is provided with a point of origin, direction and a maximum length and any objects that collide with said ray will be selected.

An example of the Ray-Cast technique can be seen in **Figure 11**.

[2]https://docs.unity3d.com/ScriptReference/LineRenderer.html
[3]https://docs.unity3d.com/ScriptReference/RequireComponent.html
[4]https://docs.unity3d.com/ScriptReference/Physics.Raycast.html


VRIL_SingleRay...
Klasse
→ VRIL_InteractionTechn...

Felder
- IsActivated
- MaxRayDistance
- SingleRayCastLi...

Methoden
- Awake
- OnActivation
- OnSelection
- OnStop
- ShowRay

Figure 10: VRIL_SingleRayCast UML diagram

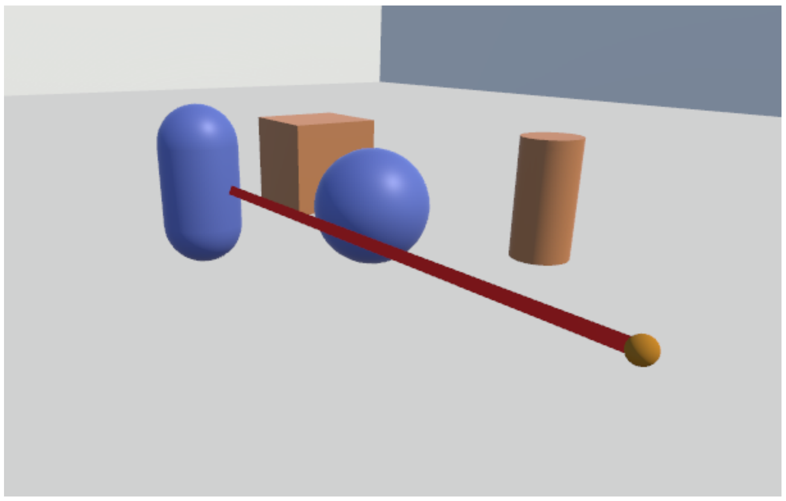


Figure 11: VRIL_SingleRayCast example

3.5.2 Depth-Ray

The Depth-Ray technique [GB06] is similar to the single Ray-Cast technique [Bol80] but instead of relying on the ray to select objects, a sphere is used to indicate the area where objects can be selected. Together with another controller, the distance between the two controllers determines the distance of the model on the ray. Linear mapping is used to determine the distance and can be adjusted freely. The greater the distance between the two registered controllers for this technique, the greater the distance of the model on the line to the controller where the ray originates. The selection model indicates the area in which an object can be selected. If another model with a different mesh is used like a quad or cylinder, the model will not accurately represent the selection area since a sphere ray cast is used to check for objects in the area. This technique provides a small set of options like the model size, initial sphere distance and linear distance mapping.

An example of this technique can be seen in **Figure 12** (UML diagram located at the image appendix **Figure 18**).

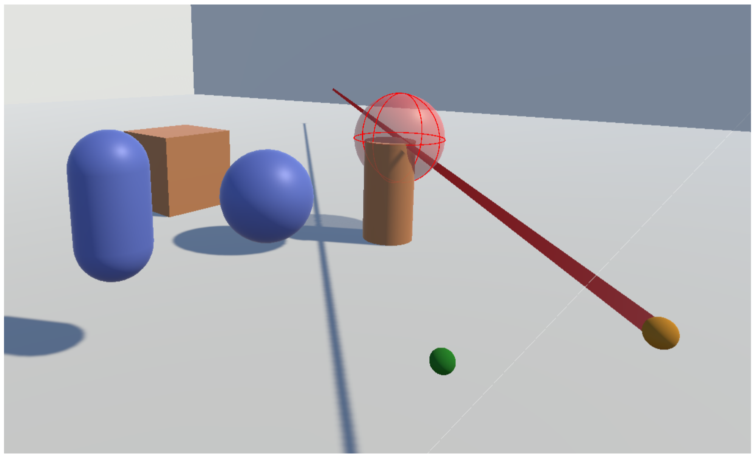


Figure 12: VRIL_DepthRay example

3.5.3 iSith

The iSith technique [WBB06] utilizes two controllers to select an object. A ray from each of the controllers is created that expands in front of the controller with a maximum distance. If those two intersect with one another a model is created (red sphere **Figure 13**). This model can grow or shrink in size depending on the distance between the two main rays from the controllers. The model indicates the region in which an object can be selected.

The iSith technique implements three line renderers. Since the line renderers are only instantiated at runtime (there is currently no way to require 3 instances of a line renderer via class attribute), a small list of options is provided for the ray: width, color, the line material, the shadow casting mode and the maximum distance of the ray. One line is created at each controller position moving forward with the options specified. The third line from the last renderer is used to indicate the shortest distance between the two forward vectors from each controller. The line will not be displayed if one vector is not in front of the other (one controller is pointing behind the other controller). If there is a shortest distance between the two vectors, a custom model of a transparent sphere is placed in the middle between the two vectors.

The provided model will be automatically resized based on the two options *MinSphereDiagonal and MaxSphereDiagonal*. As the rays are further apart from each other the model can be resized to fit the desired size. This model can be replaced with any other model as long as no component that creates a collider[5] around the model is attached to it. The model, however, does not indicate the space in which objects can be selected if it has a different mesh than a sphere. Every frame, the interaction technique will cast a sphere, which size corresponds to the *MinSphereDiagonal and MaxSphereDiagonal* options that are used to resize the used model. Any overlapping objects that have the *VRIL_Interactable* component attached will be selected by using a sphere ray-cast. To make development easier, the *OnDrawGizmos* function from the Unity scripting API is used to show the size of the actual sphere which can be seen in the scene view in Unity if activated.

[5]https://docs.unity3d.com/ScriptReference/Collider.html

Depending on the position and if there is a shortest distance between the two lines, the provided model will be set active or inactive to provide better performance instead of deleting it. Setting the model inactive means it will not be visible anymore. If the model is invisible no objects can be selected even if the rays point at objects as the main usage is to select objects with the sphere instead of the rays.

An example of the iSith implementation in action can be seen in **Figure 13**.

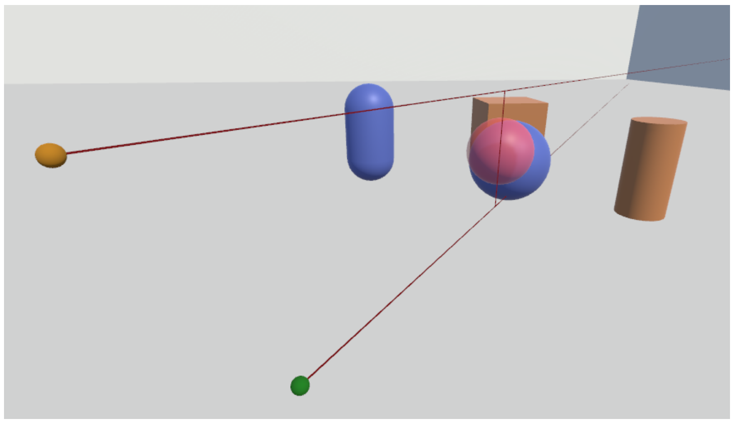


Figure 13: VRIL_iSith example

3.5.4 Spindle + Wheel technique

The Spindle + Wheel technique [CW15] is an extension of the original Spindle technique [MM95]. While the iSith technique [WBB06] focuses on the selection part of an interaction technique by providing a unique way of selection with two rays, the Spindle + Wheel technique focuses on the manipulation part after selecting an object.

An object is selected by using both controllers and a selection model similar to the iSith technique. But instead of using two rays only one ray is instantiated. At the middle between the two controllers, a transparent model is placed to indicate the area in which objects can be selected. After an object has been selected the technique can be switched to the manipulation mode which results in the selected object being placed in between the two controllers. The manipulation mode then allows users to manipulate an object's rotation, position and the scale:

- Scaling:
  The distance between the two controllers translates to the scale of the object.
- XYZ axis translation:
  Moving both controllers allow for the placement of the object along all axes.
- YZ axis rotation:
  Asymmetric hand movement result in yz rotations on the object.
- X rotations:
  Moving the dominant-hand controller results in changed x rotations on the object that translate to the controller rotation.

An example of this technique can be seen in **Figure 14** and **Figure 15** (UML diagram **Figure 20** located at the image appendix).

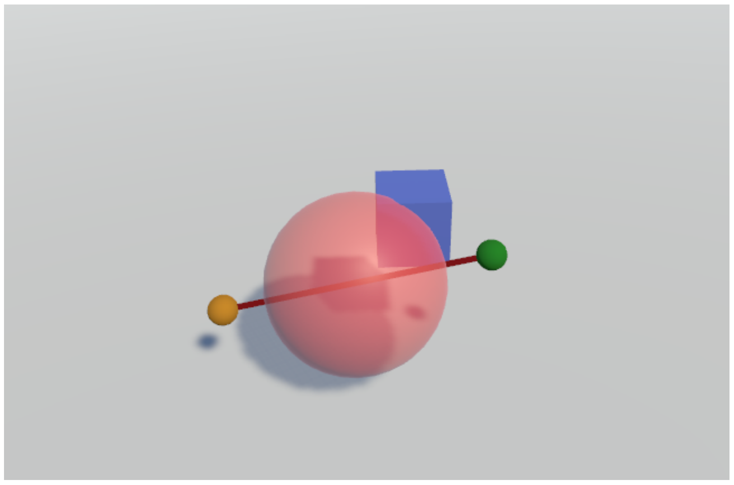


Figure 14: VRIL_SpindleAndWheel selection example

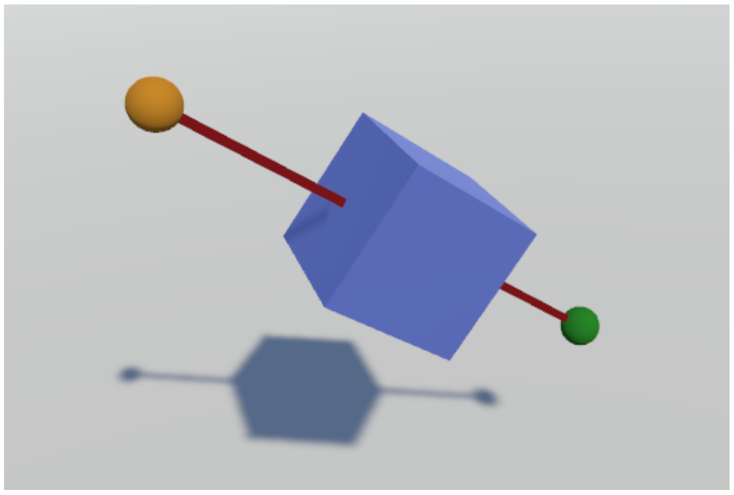


Figure 15: VRIL_SpindleAndWheel manipulation example

3.6 Extendability

Since the interaction technique base class is abstract new interaction techniques can easily be created while still providing an option to overwrite base class functions if any other desired functionality is wished for. Additionally, the mapping of controller actions to a custom script can be used to extend an existing interaction technique even more. The manager relays controller actions, which have to be mapped into the corresponding controller actions first (provided by the library). This allows the use of interchangeable SDKs or custom input frameworks. The SKDs have to invoke the function *OnControllerAction* of the manager with a custom *VRIL_ControllerActionEventArgs* object that is used to specify what controller action was triggered. Since almost every function from the base class can be replaced by the derived class, navigation techniques could also be implemented as well while technically not being an interaction technique that interacts with objects.

3.7 Software used

For the scripting environment, Visual Studio[6] is used. Visual Studio provides the option to debug scripts and is integrated well into the Unity development process.

[6]https://visualstudio.microsoft.com

4 Image Appendix

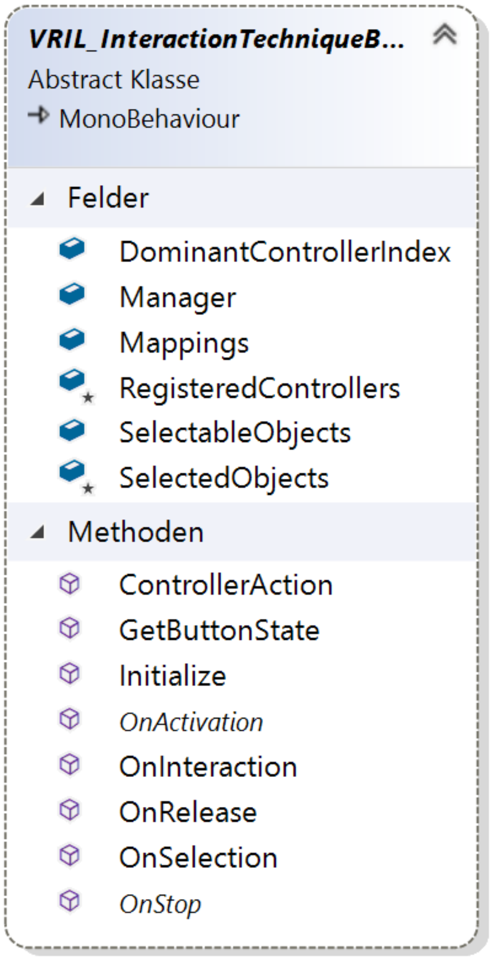


Figure 16: VRIL_InteractionTechniqueBase UML diagram

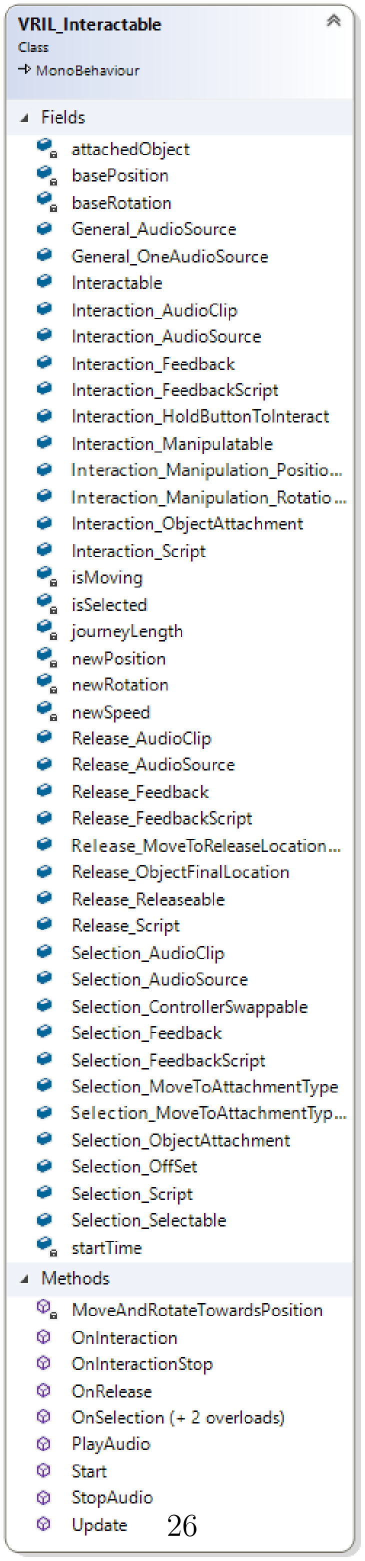


Figure 17: VRIL_Interactable UML diagram

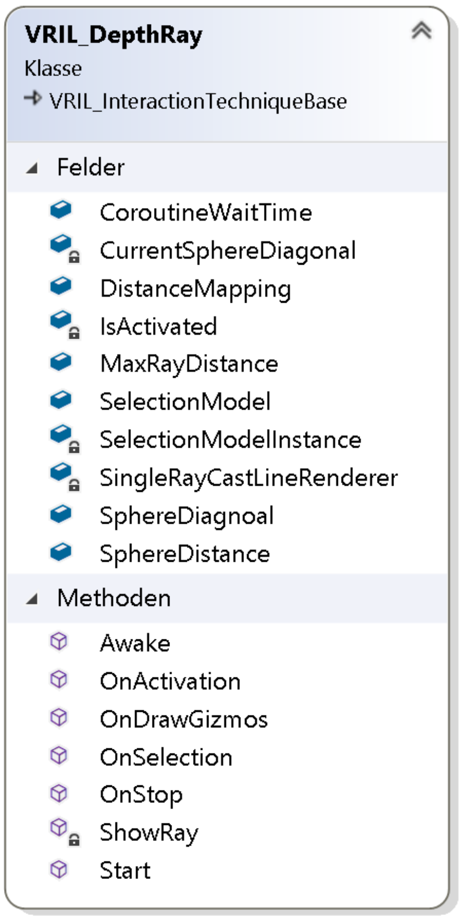
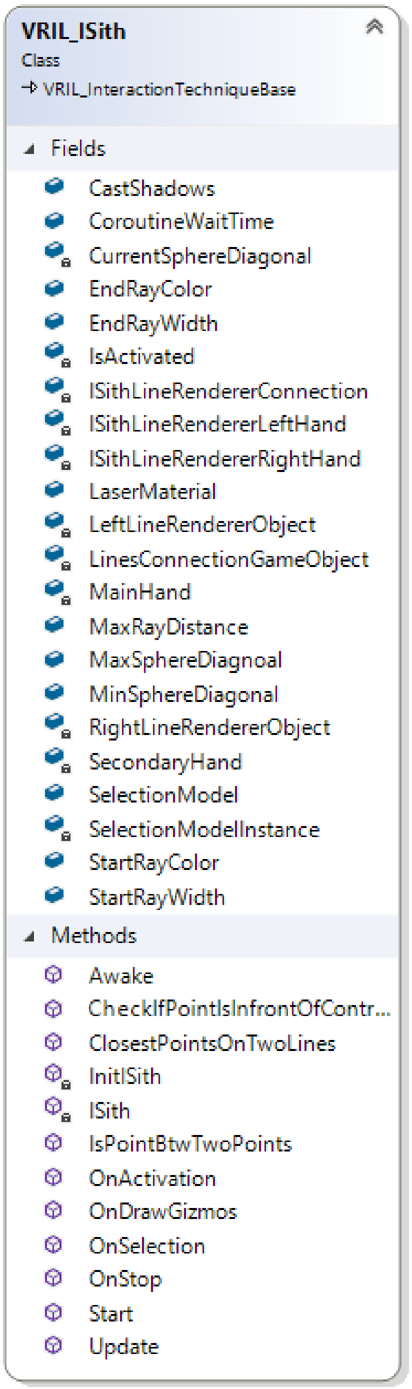


Figure 18: VRIL_DepthRay UML diagram


VRIL_ISith
Class
→ VRIL_InteractionTechniqueBase

Fields

- CastShadows
- CoroutineWaitTime
- CurrentSphereDiagonal
- EndRayColor
- EndRayWidth
- IsActivated
- ISithLineRendererConnection
- ISithLineRendererLeftHand
- ISithLineRendererRightHand
- LaserMaterial
- LeftLineRendererObject
- LinesConnectionGameObject
- MainHand
- MaxRayDistance
- MaxSphereDiagnoal
- MinSphereDiagonal
- RightLineRendererObject
- SecondaryHand
- SelectionModel
- SelectionModelInstance
- StartRayColor
- StartRayWidth

Methods

- Awake
- CheckIfPointIsInfrontOfContr...
- ClosestPointsOnTwoLines
- InitISith
- ISith
- IsPointBtwTwoPoints
- OnActivation
- OnDrawGizmos
- OnSelection
- OnStop
- Start
- Update

Figure 19: VRIL_iSith UML diagram

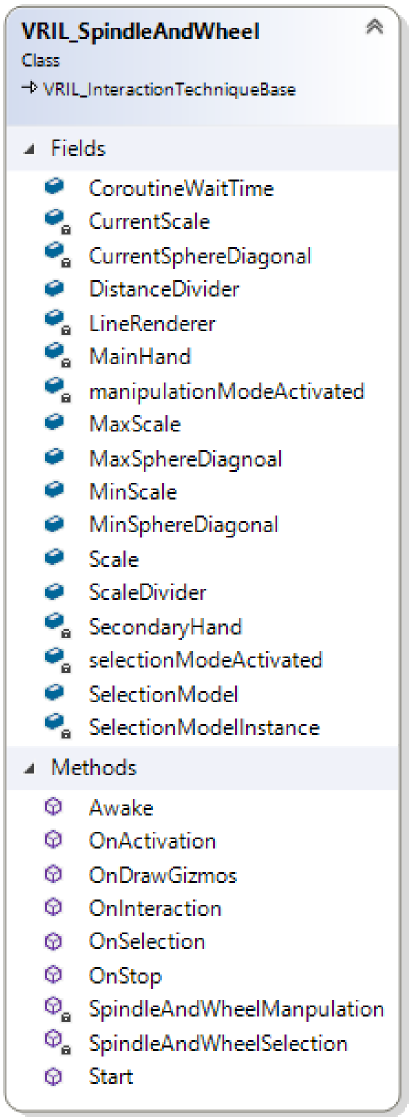


Figure 20: VRIL_SpindleAndWheel UML diagram

References

[BH97] Doug A. Bowman and Larry F. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, I3D '97, pages 35–ff., New York, NY, USA, 1997. ACM.

[Bol80] Richard A. Bolt. Put-that-there: Voice and gesture at the graphics interface. In *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '80, pages 262–270, New York, NY, USA, 1980. ACM.

[CW15] I. Cho and Z. Wartell. Evaluation of a bimanual simultaneous 7dof interaction technique in virtual environments. In *2015 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 133–136, March 2015.

[GB06] Tovi Grossman and Ravin Balakrishnan. The design and evaluation of selection techniques for 3d volumetric displays. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, pages 3–12, New York, NY, USA, 2006. ACM.

[MM95] Daniel P. Mapes and J. Michael Moshell. A two-handed interface for object manipulation in virtual environments. *Presence: Teleoper. Virtual Environ.*, 4(4):403–416, January 1995.

[WBB06] H. P. Wyss, R. Blach, and M. Bues. isith - intersection-based spatial interaction for two hands. In *3D User Interfaces (3DUI'06)*, pages 59–61, March 2006.