# AARToolkit v0.3 Documentation

*Unity toolkit for development and prototyping of a hybrid projector / AR system.*

## Overview

The purpose of this toolkit is to provide developers with the tools to rapidly create and test ideas for hybrid projector-AR HMD systems.

The toolkit consists of four main areas.

1. AR-Projector System
2. Calibration-Hardware Interface
3. Spatial Awareness System
4. Rendering Engine

## Installation

Everything required to run this version is contained here in the zip

Download all files with the provided link.

1. Create a new Unity project, (**Use version 2018.4 LTS**).
2. Import Microsoft MixedReality.Toolkit.Unity.Foundation.2.3.0 package. Note, it is important to set this up first before proceeding. See: https://microsoft.github.io/MixedRealityToolkit-Unity/README.html
3. Import all items from AARrealityToolkit_v0.x into Unity.
4. Go to "Player Settings" and turn VR renderer mode to **Multi-pass** under XR settings. And turn on “unsafe code” in General Settings
5. Open Unity's Layers and add these immediately after Postprocessing:
    - AARVirtualObjects
    - AARVirtualEnvironments
    - AARVirtualTextures
    - AARBlendable
    - AARProjectorOnly
    - AARHololensOnly

### Configuring Scene

Inside folder `AugmentedAugmentedReality/Scene,` open up the SampleScene and then:

1. Make sure the MRTK prefabs are loaded (accessed through toolbar).
    - MixedRealityToolkit
    - MixedRealityPlayspace
2. Click on MixedRealityToolkit and configure the project for Hololens 1.
    - Might have to clone the profiles to configure it properly.
    - ⚠️ Important ⚠️: Go to the Input tab, and disable Eye Gaze under Input Data Providers
3. Add the AAR* prefabs contained in AugmentedAugmentedReality/Prefabs
4. Click on the ConfigurationManager scene object, and load in the profile provided in the folder above.
    - Initialize the plugin.

# AR-Projector System

Provides tools to control the projector and the servo motors for movement relative to the Hololens and encapsulates the general physical structure of the hardware.

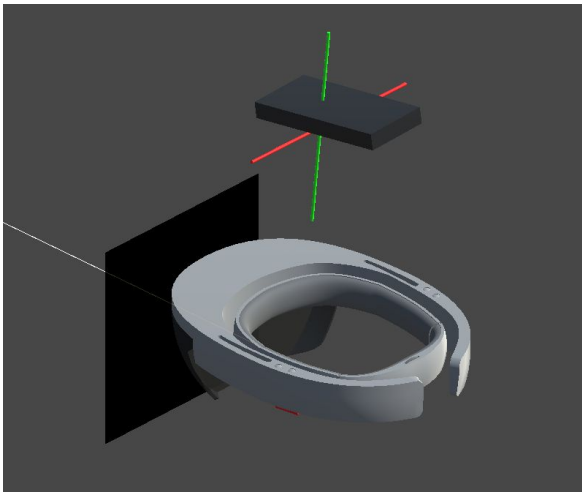


***Image**: Depicts the virtual Hololens-Projector structure as represented inside Unity. The red and green axes represent the servo motors positions and axis of rotation.*

## Projector Movement API

**File**: AARCameraProjectorRig.cs
**Prefab**: AARCameraProjectorRig
**Accessor**: Singleton

Invoke methods through singleton accessor:

```
AAR.AARCameraProjectorRig.Instance
```

The movement API provides high-level functionality to move the projector so that the centre of its frustum points toward world-space coordinates, or to move the frustum relative to the Hololens' current view.

Movement relative to the Hololens current view is based on where the frustum of the Hololens intersects the real-world geometry.  There are currently 5 different locations described by the **ScreenLocation** enum.

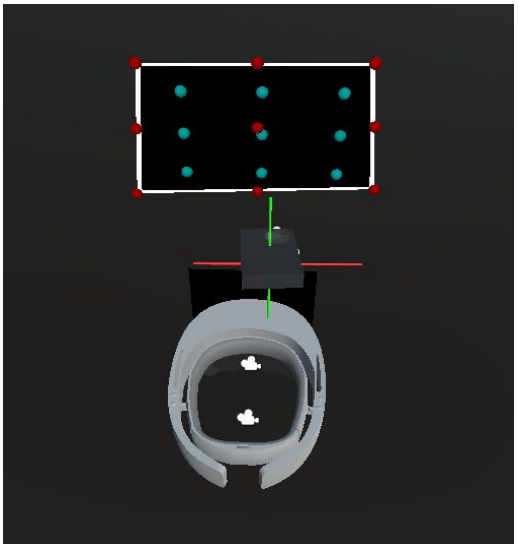
```
public enum ScreenLocation
{
    AAR_SCREEN_CENTER = 0,
    AAR_SCREEN_LEFT,
    AAR_SCREEN_RIGHT,
    AAR_SCREEN_UP,
    AAR_SCREEN_DOWN,

    // Total
    AAR_SCREEN_COUNT
}
```

**Center Position** (`AAR_SCREEN_CENTER`)

Positions the projectors frustum (red) so that it is pointing towards the centre of the Hololens' view (blue).

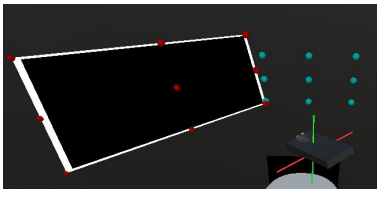
*Note: the projector's FoV is slightly larger than the Hololens'.*

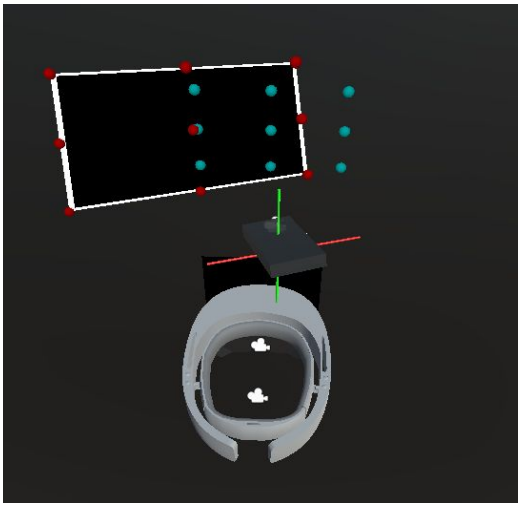


**Screen Left** (`AAR_SCREEN_LEFT`)

Positions the projector's frustum (red) to the centre of the far left position of the Hololens' view (blue).

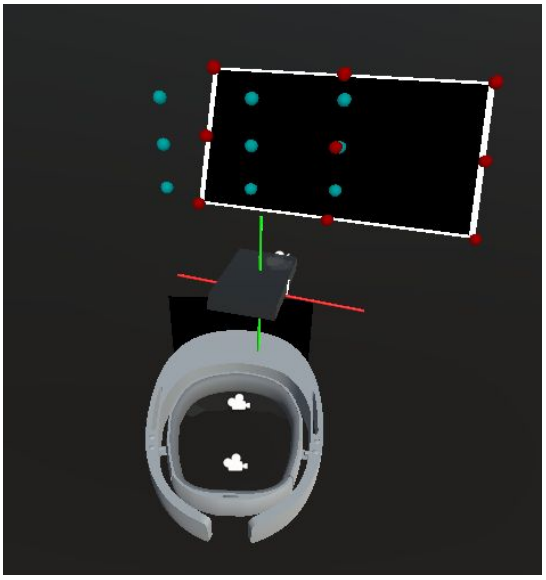
A variation will place it further, such that it touches the end of the Hololens' view.





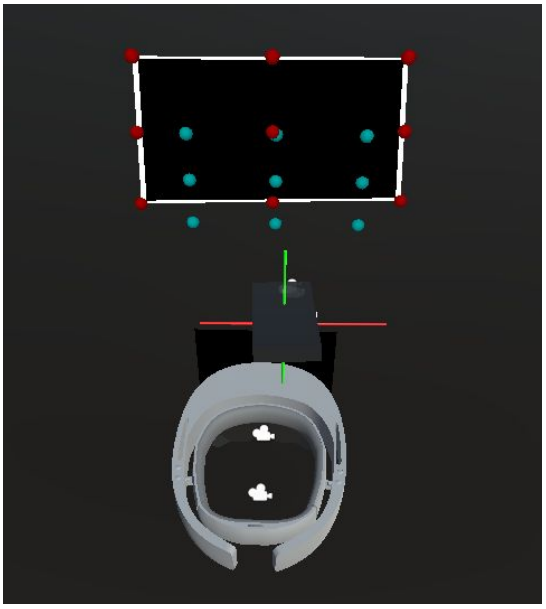
**Screen Right** (`AAR_SCREEN_RIGHT`)

Positions the projector's frustum (red) to the centre of the far-right position of the Hololens' view (blue).



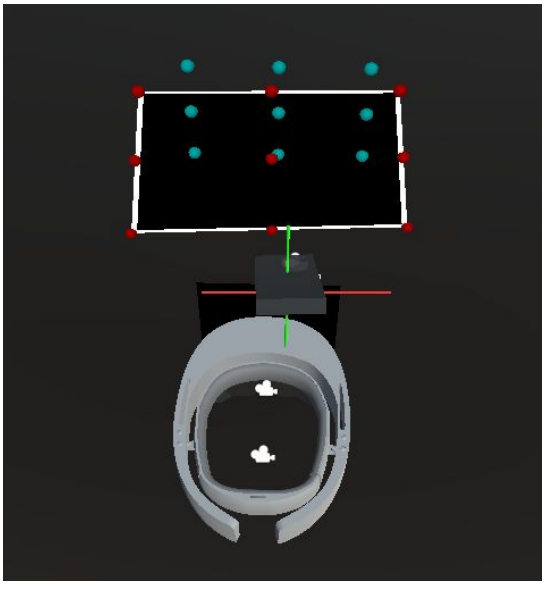
**Screen Up** (`AAR_SCREEN_UP`)

Positions the projector's frustum (red) to the middle-up location on the Hololens' view (blue).



**Screen Down** (`AAR_SCREEN_DOWN`)

Positions the projector's frustum (red) to the middle-lower location on the Hololens' view (blue).



## Methods

```
/// <summary>
/// Moves projector to look at point in world coordinates.
/// </summary>
/// <param name="_position">
///     World-space position.
/// </param>
public void LookAt(Vector3 _position)
```

```
/// <summary>
/// Projector locks onto a target (GameObject) and maintains lock
/// for duration object is active
/// </summary>
/// <param name="_followObject">
///     World-space position (Vector3)
/// </param>
public void Follow(GameObject _followObject)
```

```
/// <summary>
/// Removes any object the projector is currently locked onto.
/// </summary>
public void Unfollow()
```

```
/// <summary>
/// Moves the projector to locations relative to the frustum of the
/// Hololens view.
/// </summary>
/// <param name="_location">
///     An enum dictating where the projector should lock
/// </param>
/// <param name="_overlap">
///     Overlap determines if the projector should be adjacent or with
///     partial overlap.
///     The projector and hololens overlap 50 %
/// </param>
public void MoveTo(ScreenLocation _location, bool _overlap = false)
```

# Projector Rendering API

**File**: AARCameraProjectorRig.cs
**Prefab**:  AARCameraProjectorRig
**Accessor**: Singleton

Invoke methods through singleton accessor:
`AAR.AARCameraProjectorRig.Instance`

## Methods

```
/// <summary>
/// Returns the current rendering mode used on the projector. Can be one of three
/// different states:
///     1. Default_Projection
///        Represents the default rendering model. No projection mapping is used.
///        Working Layer Masks:
///             AARVirtualTextures
///             AARProjectorOnly
///     2. View_Dependent_Projection
///        View-dependent projection mapping is used to render all virtual objects
///        from the AARUserView's point-of-view (POV). Working Layer Masks:
///             AARVirtualTextures
///             AARVirtualObjects
///             AARProjectorOnly
///             AARBlendable
///     3. Static_Material_Projection
///        A static texture or material is rendered onto the projectors view.
///        No scene objects will be displayed.
///        Working Layer Masks:
///             Not Applicable
/// </summary>
public ProjectorRenderMode GetProjectorRenderMode()
```

```
/// <summary>
/// Resets rendering mode to default (Default_Projection)
/// </summary>
public void ResetProjectorRenderingToDefault()
```

```
/// <summary>
/// Enables view-dependent projection mapping
```

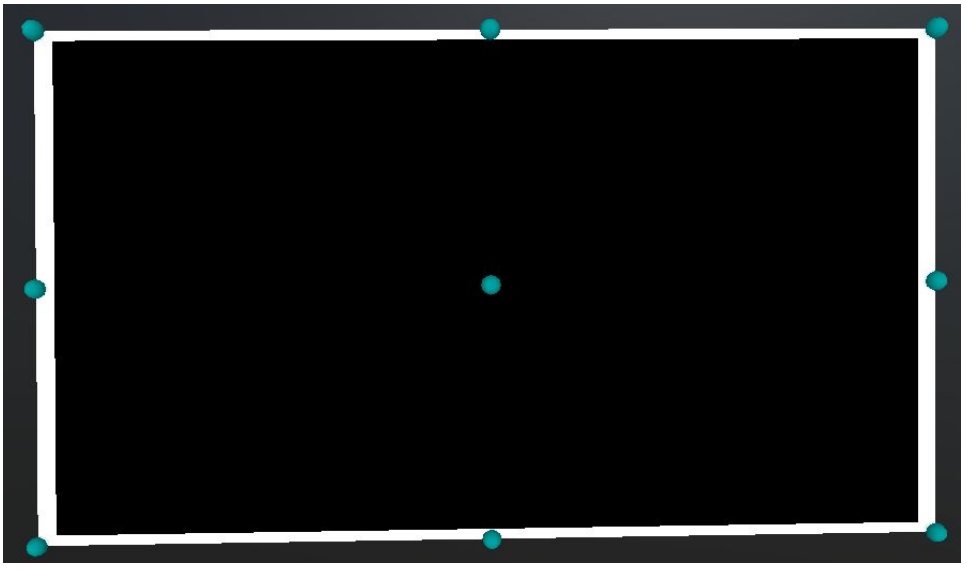
```
(View_Dependent_Projection)
/// </summary>
/// <param name="_state">
///     True:  Enables projection mapping
///     False: Sets mode to default
/// </param>
public void EnableProjectMappingRender(bool _state = true)
```

```
/// <summary>
/// Enables static texture or material rendering
/// </summary>
/// <param name="_staticMat">
/// The shader, texture, color, or material to render to the projector's
/// display.
/// </param>
public void EnableStaticMaterialRender(StaticMaterial _staticMat)
```

## Projector and Camera Frustum

**File**: AARGenerateFrustumIntersections.cs
**Prefab**:  N/A
**Accessor**: Per Object Instance

Provides utility functions for frustum bounds and intersection based on predetermined points around the frustum.



```
public enum FrustumPositions
{
    AAR_FRUSTUM_LEFTBOTTOM = 0,
    AAR_FRUSTUM_LEFTMIDDLE,
    AAR_FRUSTUM_LEFTTOP,
    AAR_FRUSTUM_TOPMIDDLE,
    AAR_FRUSTUM_RIGHTTOP,
    AAR_FRUSTUM_RIGHTMIDDLE,
    AAR_FRUSTUM_RIGHTBOTTOM,
    AAR_FRUSTUM_BOTTOMMIDDLE,
    AAR_FRUSTUM_CENTER,

    // TOTAL
    AAR_FRUSTUM_COUNT
}
```

## Methods

```
/// <summary>
/// Gets the current world-coordinate point for a particular location on
/// camera's frustum.
/// </summary>
/// <param name="_position">
///     Determines what point of the frustum will be returned.
/// </param>
/// <returns>
///     Vector3 in world-coordinates.
/// </returns>
public Vector3 GetCameraFrustumWorldPoint(FrustumPositions _position)
```

```
/// <summary>
/// Determines whether two frustums intersect eachother.
/// </summary>
/// <param name="_otherCam">
///     The camera / projector to check.
/// </param>
/// <returns>
///     True if frustums are intersecting
/// </returns>
public bool CheckIntersection(Camera _otherCam)
```

## Projector and Servo Structure

**File**: AARHololensSensor.cs
**Prefab**:  N/A
**Accessor**: Per Object Instance

Encapsulates the sensors on the Hololens. These include the PVCamera, VLC Cameras, and Depth sensors. Each sensor has its associated transformation relative to the Hololens origin point.

**File**: AARProjector.cs
**Prefab**:  N/A
**Accessor**: Per Object Instance

Encapsulates the projector and provides methods to get projector intrinsic parameters, positions, and frustum intersection points.

```
/// <summary>
/// Helper function that calculates the intersection of the projector's
/// frustum with another projector or camera.
/// </summary>
/// <param name="_otherCamera"></param>
/// <returns></returns>
public bool CheckIntersection(Camera _otherCamera)
```

```
/// <summary>
/// Gets a position on the projectors frustum in world coordinates.
/// </summary>
/// <param name="_position"></param>
/// <returns></returns>
public Vector3 GetFrustumPosition(FrustumPositions _position)
```

```
/// <summary>
/// Shows a visualization of what the projector "sees". This is projected
/// back into the world environment, simulating the physical projector.
/// </summary>
/// <param name="_enable"></param>
public void EnableProjectorVisualizer(bool _enable)
```

**File**: AARServo.cs
**Prefab**:  N/A
**Accessor**: Per Object Instance

Encapsulates the functionality of the physical servos. Provides methods to rotate servo towards a world coordinate position.

```
/// <summary>
/// Rotates the servo around a single axis, positiong it towards the
/// world coordinate vector.
/// </summary>
/// <param name="_worldPosition"></param>
/// <param name="_offset"></param>
public void LookAtWorldPosition(Vector3 _worldPosition, GameObject
_offset = null)
```

# Calibration and Hardware Interface

The calibration and hardware components are accessed through a native plugin written in C++. The interface handles all connection to and from the plugin and will accommodate missing hardware, simulating everything in the Unity environment.

## Configuration Manager

**File**: CSConfigurationManager.cs
**Prefab**: ConfigurationManager
**Accessor**: Singleton

Invoke methods through singleton accessor:

```
AAR.CSConfigurationManager.Instance
```

Builds the servo and projector structure based on the calibration input data. Sets up the AARPlayspace and instantiates all required components.

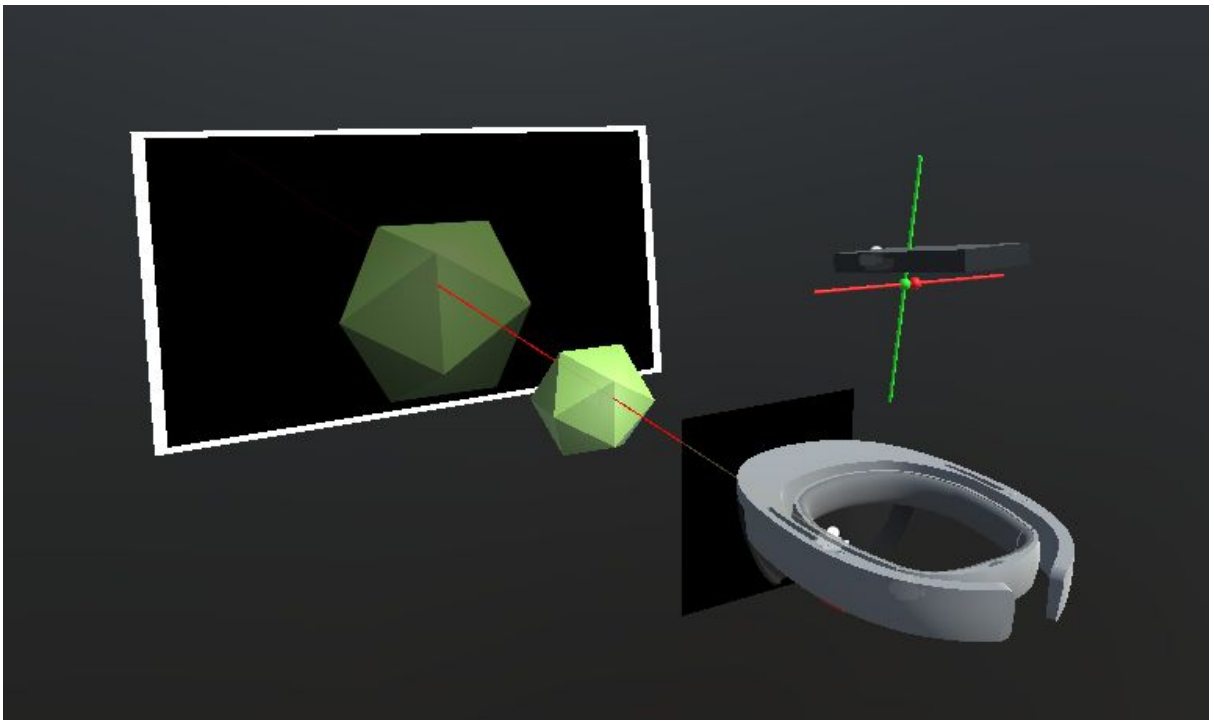
To configure:

1) Click on “Load” and select the calibration file.
2) Click on Initialize.

The configuration manager has to be initialized once per Unity instance. This will call into the native plugin DLL and allocate the projector and servo components.

**Show Projectors**: Will display the current index of the projector on the companion screen.
**Enable Fullscreen**: Expands projector to an external display. (Note: This may not work without hardware access.)
**Reset to Calibrated Centre View**: Initiates a calibration process to reset the hardware servos.

**Projector Visualizer**:
If enabled, will project the contents the servos see into the world environment. This will be from the projector’s point of view if projection mapping is turned off and from the AARUserview if projection mapping is on.



# Projector and Servo Controllers

**File**: CSProjectorController.cs / CSServoController.cs
**Prefab**:  N/A
**Accessor**: Per Object Instance

Provides an interface between the hardware and the software components of the projector and servo.

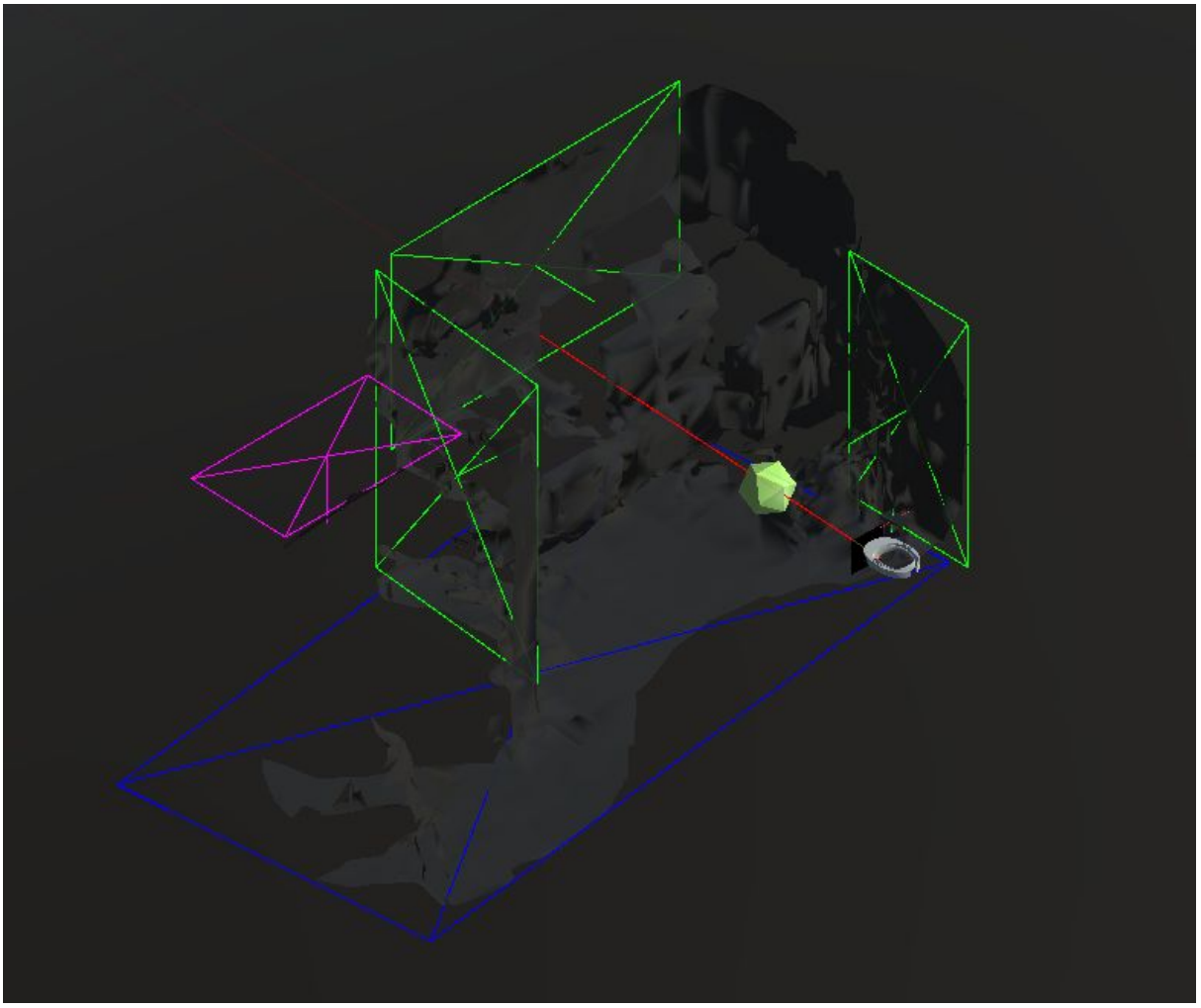
# Spatial Awareness System

**File**: AARSpatialAwarenessManager.cs
**Prefab**:  AARSpatialAwarenessManager
**Accessor**: Singleton

Invoke methods through singleton accessor:

`AAR.AARSpatialAwarenessManager.Instance`

Provides utility functions for plane finding and environment reconstruction.



**Plane Type Enum**

```
    /// <summary>
    /// All possible plane types that a SurfacePlane can be.
    /// </summary>
    [Flags]
```

```
    public enum PlaneTypes
    {
        Wall = 0x1,
        Floor = 0x2,
        Ceiling = 0x4,
        Table = 0x8,
        Unknown = 0x10
    }
```

## Methods

```
/// <summary>
/// Starts the mesh observations through the MRTK
/// </summary>
public void StartMeshObservations()
```

```
/// <summary>
/// Stops mesh observations through the MRTK
/// </summary>
public void StopMeshObservations()
```

```
/// <summary>
/// Set a flag to update the merged planes on next frame loop.
/// </summary>
public void UpdateMergedPlanes()
```

```
/// <summary>
/// Sets flag to update subplanes on next frame loop.
/// </summary>
public void UpdateSubPlanes()
```

```
/// <summary>
/// Searches the mesh provided through the MRTK spatial awareness system
/// and constructs planes as gameobjects.
/// </summary>
public void BuildEnvironmentPlanes()
```

# Rendering Engine

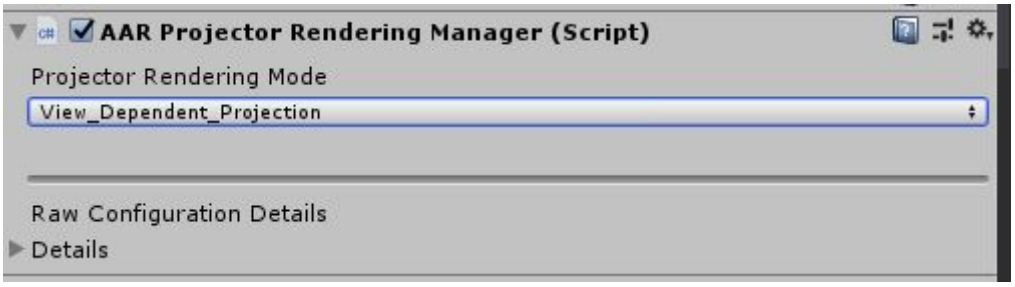
Provides rendering functionality for view-dependent projection mapping and for object blending between Hololens and projector views.

**File**: AARProjectorRenderingManager.cs
**Prefab**:  AARProjectorRenderingManager
**Accessor**: Singleton

Invoke methods through singleton accessor:

```
AAR.AARProjectorRenderingManager.Instance
```

There are three different rendering modes that work alongside the Layer Masks provided through Unity. The modes can be accessed through the **AARProjectorRenderingManager** prefab through a drop-down panel, but can also be changed dynamically at runtime.




```
public enum ProjectorRenderMode
{
    Invalid = -1,
    Default_Projection,
    View_Dependent_Projection,
    Static_Material_Projection
}
```

## Default Projection

Enum: `Default_Projection`

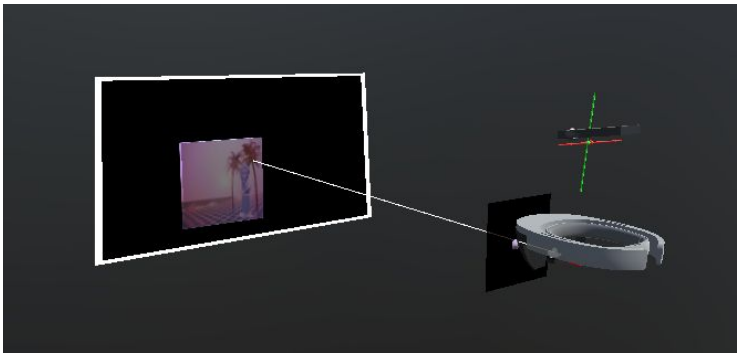
This mode renders scene objects from the projectors point of view. No projection mapping occurs and only objects with layer masks set to:

1) AARProjectorOnly
2) AARVirtualTexture

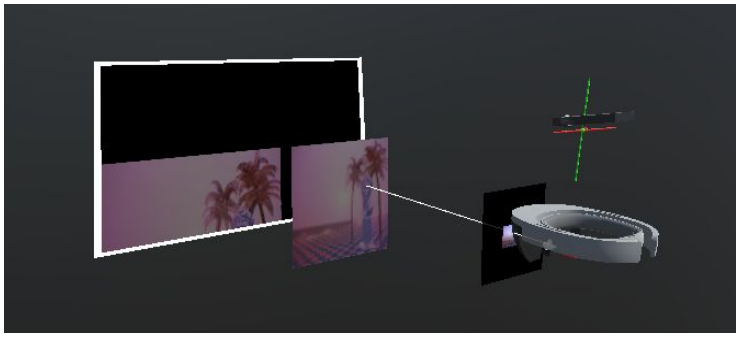
Will be visible by the projector. All other layers are culled during the rendering process.

This mode is usually only suitable for virtual textures that are aligned against a wall or mesh that represents the real-world geometry of the space. The reason to do this is to ensure there is alignment between the Hololens rendering of the object and the projectors form any point of view.

For example, if you have a texture on a wall, the virtual location of this object will be aligned with the physical location of the projection of that object. As represented by the image on the right.



However, if the object floats freely in the virtual environment, then there will be *misalignment* with the projected object in the real environment and the virtual representation of it.



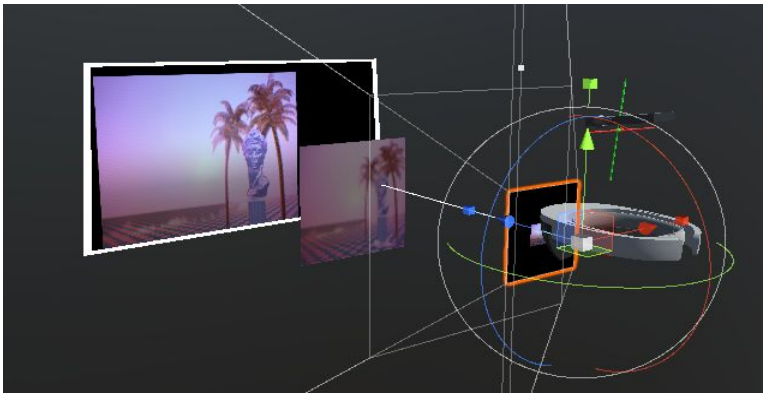
## View-Dependent Projection Mapping

Mode: `View_Dependent_Projection`

Projection mapping can get around some of the limitations listed above by rendering scene objects from a particular point of view. This lets you create hologram type illusions of 3D objects for a single viewpoint.

Like the default projection mode, this works with Unity's built-in layer mask system. Used layer masks:

1) AARPorjectorOnly
2) AARVirtualTexture
3) AARVirtualObject
4) AARBlendable (Details about this later)



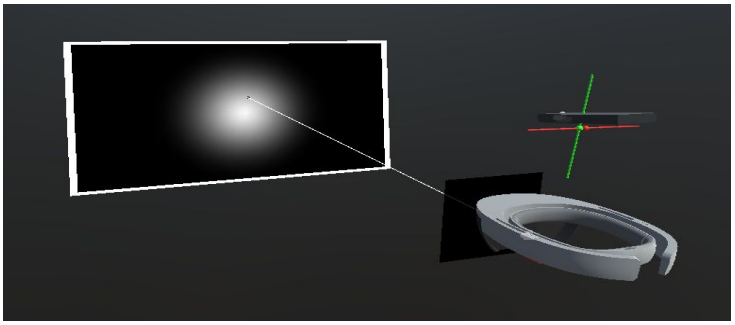
This is accomplished by rendering from the viewpoint of the **AARUserView** object in the scene. By default, this object is located at the Hololens origin point, rendering from the

viewpoint of the Hololens user, but other third-person perspectives could also be achieved by moving the **AARUserView** prefab.

## Static Material Projection

Mode: `Static_Material_Projection`

This projects a static texture or material shader from the projector's point of view. This can be used to project textures that are not scene-objects. For example, using the projector as a spotlight, or for some other type of effect.



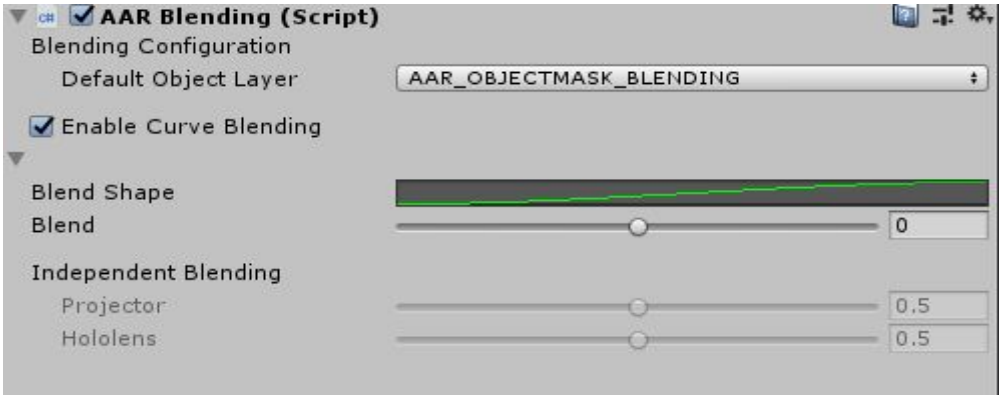
Static materials are encapsulated by the **StaticMaterial** class, which holds the texture (i.e. image), shader, and material that is used to project into the environment.

## Blendable Objects

Mode: `View_Dependent_Projection`

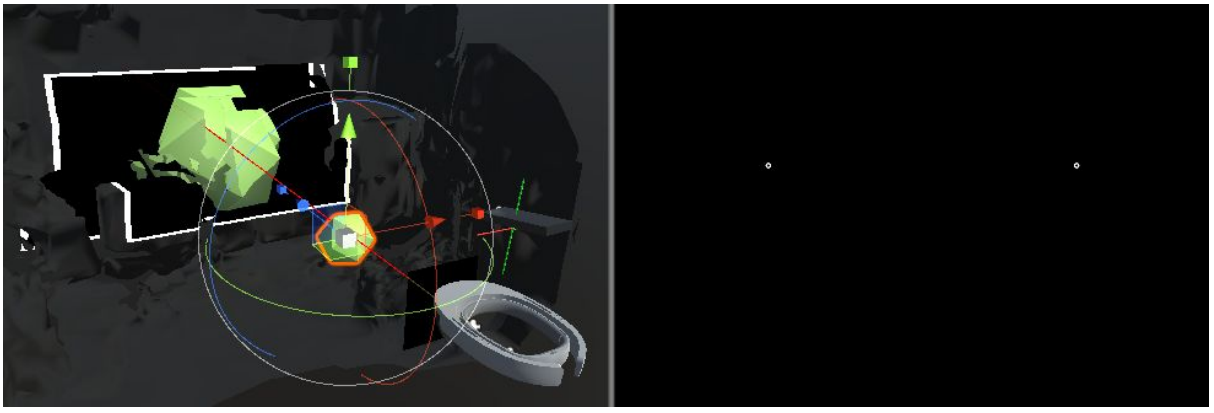
**File**: AARBlending.cs
**Prefab**: N/A
**Accessor**: Per Object Instance

A blendable object is a special type of object that can blend between the projector and Hololens view. The **AARBlending** script needs to be attached to the object containing the **Mesh Filter** component. The object must be set to **AARBlenable** mask layer.

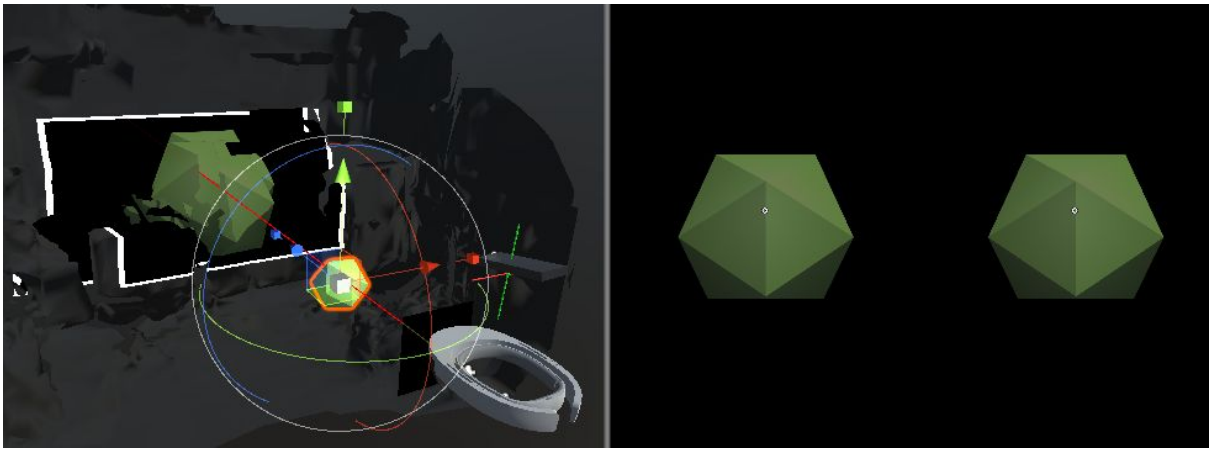


The blending can be controlled through the use of a curve or individually depending on the use case.

Projector:100 / Hololens: 0



Mixed: 50/50



Projector: 0 / Hololens: 100