Table of contents

1 Getting started

This chapter covers everything you need to do to fully integrate this asset into your own project. The provided demo scene can be run after the project is correctly set up.

Note that this asset was developed with Unity's Universal Render Pipeline (URP). Therefore, all provided material is based on URP. However, other render pipelines are also supported by this asset. Refer to section "Other render pipelines or custom shaders" below for more information.

1.1 Project setup

Find the asset in Unity's Package Manager (*Window > Package Manager*) and press "Import".

This asset is using Unity's Input System. If you are using a version below Unity 6, a warning pops up on import stating that the asset has Package Manager dependencies. Press "Install/Upgrade" to install missing dependencies including Unity's Input System. If you have not had the latter in your project activated already, another warning will show up asking you to activate it. This setting can later be manually changed under *Edit > Project Settings > Player > Active Input Handling*.

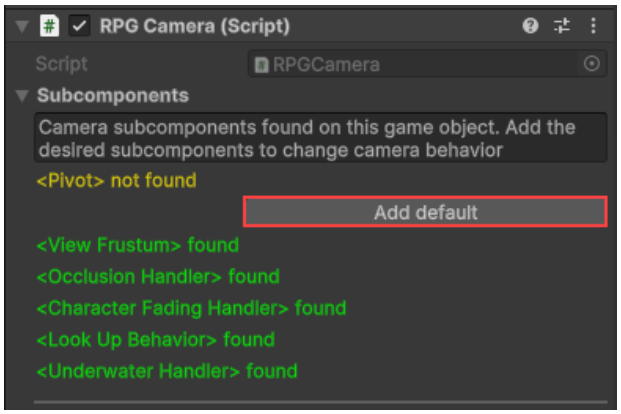
1.2 Scene setup

In the following, you find the steps for correctly setting up your scene with my asset in it:

1.2.1 Attaching the scripts

All scripts which are related to the character can be found inside *Scripts > Character*. Copy one of the provided character prefabs or attach the scripts of your choice to your character game object, usually its root:

i. RPGCamera: This script contains all general camera functionality like yaw, pitch or zoom. Additional functionality is based on the subcomponents you add to the game object – either manually or by adding a default via button:



Note that not every subcomponent is required for the camera to work!

ii. RPGController: This script uses device input to control the RPGCamera. Without this script, the camera will not move.

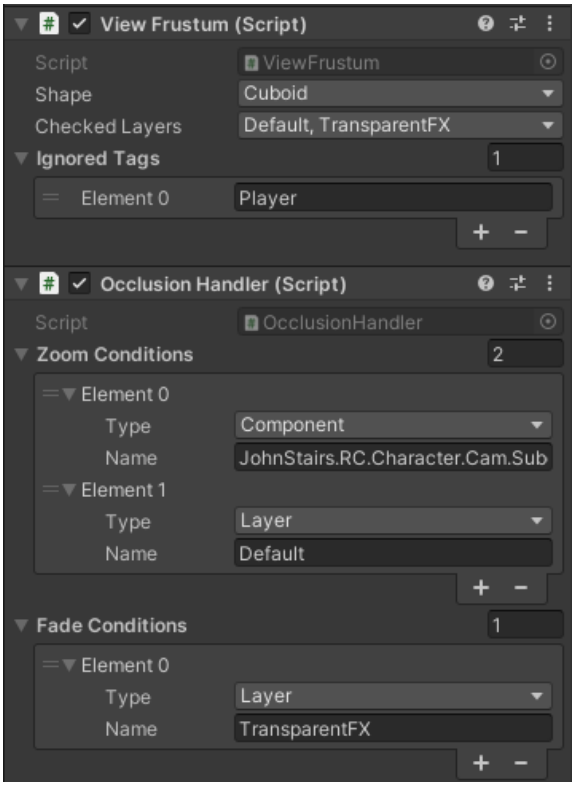
iii. PlayerInput: This component is automatically added with the RPGController. Open it and assign the RPGInputActions inside folder *Input* to the “Actions” property. Optionally, set the Behavior to “Invoke C Sharp Events”.

iv. (Optional) CursorHandler: This script is used by the RPGController to control cursor behavior. Add it if you need some cursor handling on top of camera functionality, e.g. hiding and letting it stay in place when camera orbiting starts.

v. (Optional) RPGMotorExample: Example character motor implementation based on Unity's Character Controller documentation. Used as a proof of concept for working camera and character interaction.

1.2.2 Assigning the right layers

Some of the provided camera subcomponents work with layer masks to filter processed game objects. There are explanation tooltips for each variable that are displayed when hovering over a variable label. Nevertheless, a short example does not harm:



The View Frustum describes what is in view between the camera and the camera pivot (defined by the Pivot subcomponent). With the setup above, only game objects inside layers “Default” and “TransparentFX” are recognized. However, game objects inside these layers with tag “Player” are ignored.

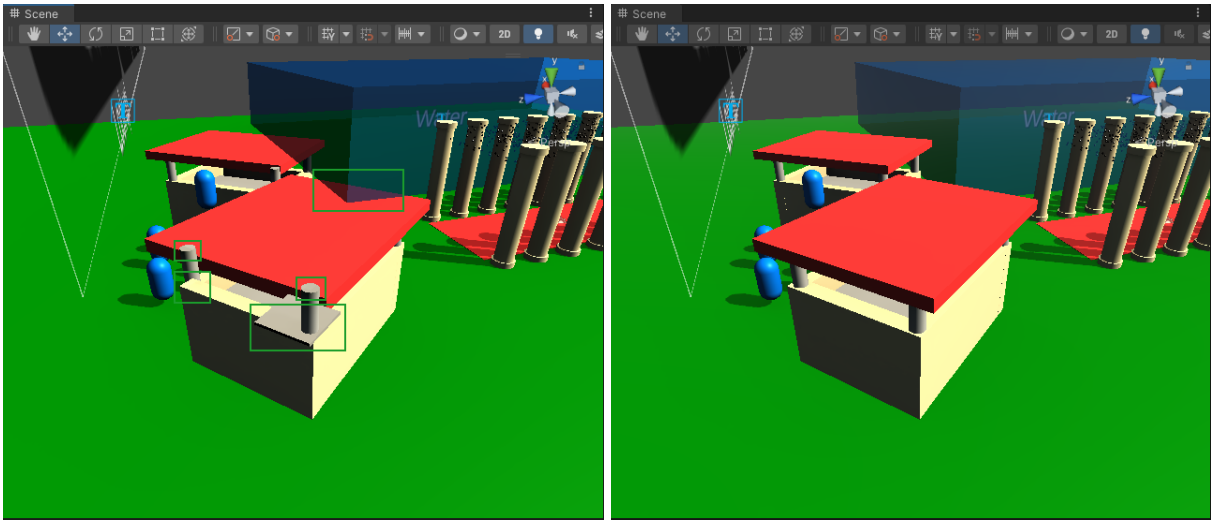
The Occlusion Handler describes how game objects inside the View Frustum are processed, i.e. if they force the camera to zoom in or if they should be faded out. Objects that are considered by the View Frustum but do not fulfill any of the conditions are not causing anything.

In combination, character game objects with layer “Default” and tag “Player” (like set up in the Character prefabs) do not cause a zoom in, even though the layer “Default” is configured as a zoom condition.

1.2.3 Using the correct shader (optional)

To make the fading of game objects work, they need to have a material with a Transparent shader assigned, e.g. Unity’s “Universal Render Pipeline/Lit” shader. Just check out which shaders are used in the provided demo scenes.

If the shader does not have ZWrite enabled by default (like the one mentioned above), it will be enabled when the material enters the view frustum and should be faded. However, for complex objects, ZWrite needs to be enabled on game start to be displayed properly. To accomplish this, you can assign the provided EnableMaterialZWrite script to the game object (see the “House” demo prefab as an example).



Left: ZWrite disabled. Right: ZWrite enabled.

1.2.4 Underwater effects (optional)

For leveraging the provided UnderwaterHandler script, two things are required for properly defining a water game object:

1. It must have the “Water” script attached
2. It must have a box collider attached that acts as a trigger

Check out the prefab “Water” in folder *Prefabs* for reference.

2 Useful interfaces

Below you find interfaces that you can implement to create custom components that seamlessly integrate with my asset.

2.1 RPGCamera subcomponents

Every subcomponent of the RPGCamera is based on an interface which can be found in the *Interfaces* subfolder. For a custom subcomponent, implement the corresponding interface and assign it to the game object. Make sure that there is only one component per interface

assigned. If the custom subcomponent was found, its entry will be displayed in green at the top of the RPGCamera script.

2.2 ICursorHandler

This interface is used by the RPGController to control cursor behavior and UI consideration. It provides methods to show and hide the cursor, or method “IsCursorOverUI” for checking if the cursor is over an UI element. Check out the provided CursorHandler which implements this interface. Depending on the result of “IsCursorOverUI”, the RPGController allows or disallows camera control.

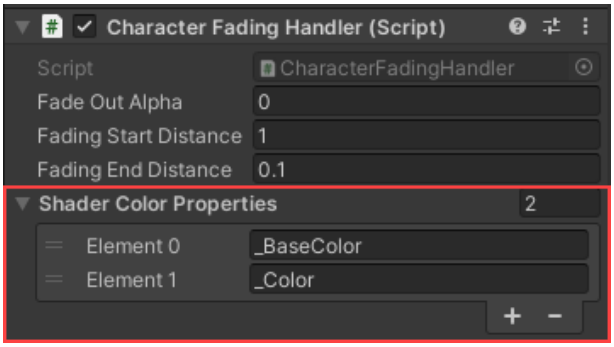
2.3 IMotor

Implement this interface if you want to use the RPGController for controlling character movement as well. An example implementation can be found in MotorExample.

Note: It is not required to have a component implementing this interface on the character game object to make the RPGCamera work. However, there are use cases where orchestration between camera and motor is required. Example: The Rotation Sync (character and camera rotating together) cannot work if the rotated degrees of the character is not known.

3 Other render pipelines or custom shaders

This asset is written based on URP. However, other pipelines and custom shaders are also supported. Every camera subcomponent that deals with the fading of objects has a list of shader color properties:



Checking from top to bottom, the first property that is found in the shader is treated as the color property. By default, “_BaseColor” is used by Unity’s URP/HDRP shaders and “_Color” is used by BRP shaders. If your own shader uses a deviating property for controlling its color, just add the property name to the list.

4 Version history

If you are interested in the release notes, please refer to my website johnstairs.com/rcc.

5 Got feedback, questions or problems?

Feedback, feature suggestions and asset reviews are highly appreciated.

In case of question which are not covered by the FAQs on johnstairs.com/rcc, feel free to send me an email to mail@johnstairs.com. Even if considered outdated, this is the most reliable way to directly reach out to me.

If you are facing issues, please attach at least a screenshot showing your scene, the used variable values and error messages which might occur in the console.

Best regards,

John Stairs