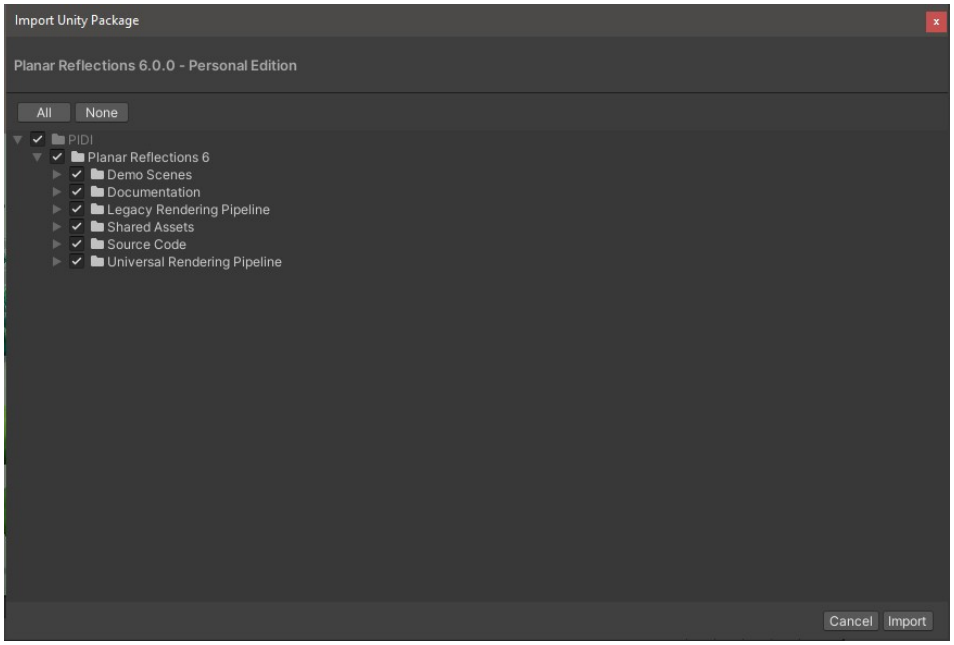
Thank you for buying PIDI Planar Reflections 6. This offline version of the documentation is provided for your convenience, but you are heavily encouraged to read the online version instead as it contains the most up-to-date information about the product, its features and capabilities.

Index

Getting Started

Installation

To install the asset, open the Package Manager and from the drop-down menu select My Assets. Then, search for PIDI Planar Reflections 6. Download the asset and prepare to import it into your project. You will see the screen showing all the contents of the asset (in the following picture we show the contents of the Personal Edition):



If you are using the Universal RP edition of the asset, you can simply import all the contents of the package into your project as there is no content from other pipelines available and the Package Manager will have downloaded the version of the asset that better matches your Unity version. If you are using the Personal Edition, please read the following section.

Personal Edition and multiple rendering pipelines

If you are using the Personal Edition of the asset, remember to deselect all the folders designed for other pipelines to reduce the amount of content installed into your project and avoid any potential errors. Inside the folders for each pipeline you can find several different unitypackage files designed to match different Unity versions. The right order to install these additional Unity packages should always be Source & Shaders -> Additional Resources (such as Amplify Shader Editor Nodes) -> Demos.

Every time you update the asset to a newer version, simply import the most recent unitypackage within the folder that matches the rendering pipeline you are using. We recommend you back up older versions of the asset in case that you want to roll back any upgrade / update.

Every time you update the asset to a newer version, simply import the most recent unitypackage within the folder that matches the rendering pipeline you are using. We recommend you back up older versions of the asset in case that you want to roll back any upgrade / update.

Upgrading from older versions

If you are upgrading from Version 5.x.x of Planar Reflections you just need to remove it entirely from your project before installing version 6.x.x. Once installed, version 6.x.x will, under normal circumstances, replace all references to the previous version without any issues. There may be some small differences in performance and / or visuals due to several brand new settings and internal changes added for version 6, but these should be minimal and can be tweaked within minutes.

Planar Reflection Renderer

The Planar Reflection Renderer is the script responsible for rendering the real-time reflections and their corresponding depth pass.

General Settings

The **General Settings** tab contains the essential features and settings of a reflection renderer.

The **Reflect Layers** drop out menu allows you to select which rendering layers will be rendered by the reflection.

Reflection Background allows you to control the background (or Clear Flags) of the reflection. It can be copied from the camera looking at the “mirror” surface, it can be set to be the Skybox, a Solid color or Transparent (useful for AR).

The Camera Filtering Mode setting is useful to define which cameras will trigger a reflection-rendering pass. By default, **all cameras** (except preview and internal reflection cameras) will trigger a reflection-rendering pass. The other options are **By Component** (which makes only the cameras with the Planar Reflection Camera component attached trigger a pass) and **By Prefix**, which filters the cameras based on their name.

Optionally, the reflection can be rendered to an **External Render Texture** provided by you. This is useful when the texture needs to be manipulated by external shaders for unique effects or when additional control over its format / properties and access is required.

Sometimes it is necessary to render a reflection’s depth besides its color. This depth can be very useful when working in AR as it allows to mask any reflected objects and remove their background very

precisely. It can also be used to simulate contact reflections (reflections that look sharper the closer the object is to the reflective surface), etc. This however requires an additional render pass, which is expensive. In most cases, it should not be necessary to render the depth pass.

If the **Show Advanced Settings** toggle is enabled, additional properties of the Reflection Renderer will become available.

In the Universal Rendering Pipeline it is sometimes required to use additional and custom Render Passes as part of your rendering pipeline. Some of these passes may be quite expensive or may, in some cases, be incompatible with Planar Reflections.

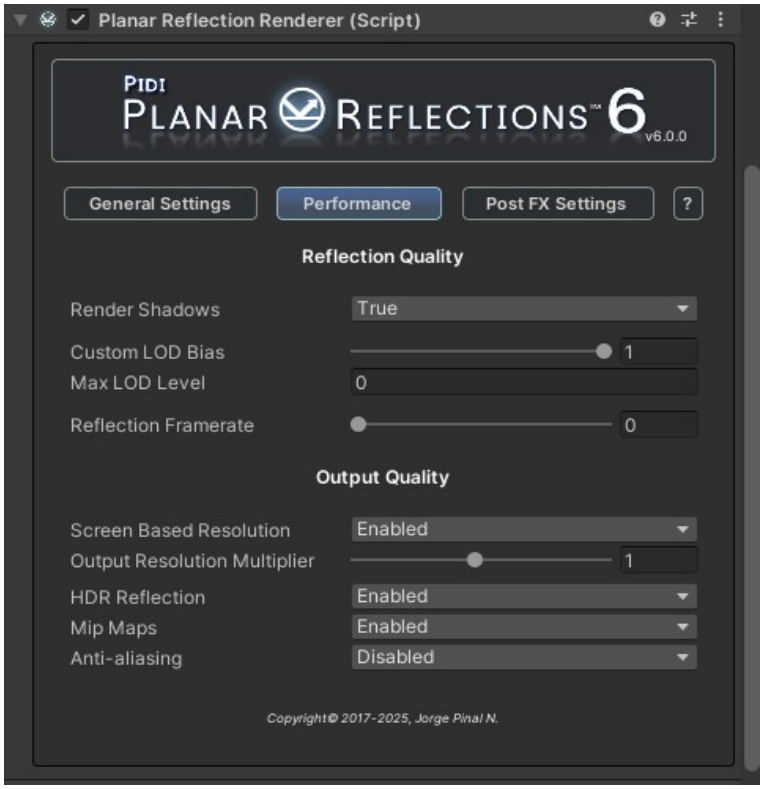
In those cases, it is useful to create an additional Renderer for your Universal Rendering Pipeline Asset and remove from it any unnecessary passes, then assign its index to the **Renderer Override** variable. It can also be useful for rendering reflections with custom effects that should not be present in the main camera view.

A reflection is, by default, **Previewable in the Scene View**. This means that all reflections will be visible at Edit and Play time on the Scene view, allowing for an easier and much better workflow. However, in complex scenes, this may cause some performance issues. For those cases, you can disable it and run the reflections only on the Game View.

The **Accurate Matrix** toggle makes sure that the projection of the Reflection Camera is adjusted to match the surface of the reflective plane. This prevents objects that are partially or totally behind the reflective surface from being rendered. In some rare cases, it may also cause visual glitches in some very specific projection-dependent effects, or disable them entirely.

Reflection Fog is a legacy, now obsolete setting that adds a custom fog pass to the reflections. In most cases reflections should display fog properly without any additional tinkering.

Performance Settings



PIDI Planar Reflections 6 offers a lot of different options to fine tune the performance of the effect, as well as many internal optimizations compared to previous releases.

Shadows can be disabled on a per-reflection basis, and a **Custom LOD Bias** and **Maximum LOD** Levels can be set to further reduce the visual fidelity of the reflected image compared to the main game's view, further reducing the performance hit.

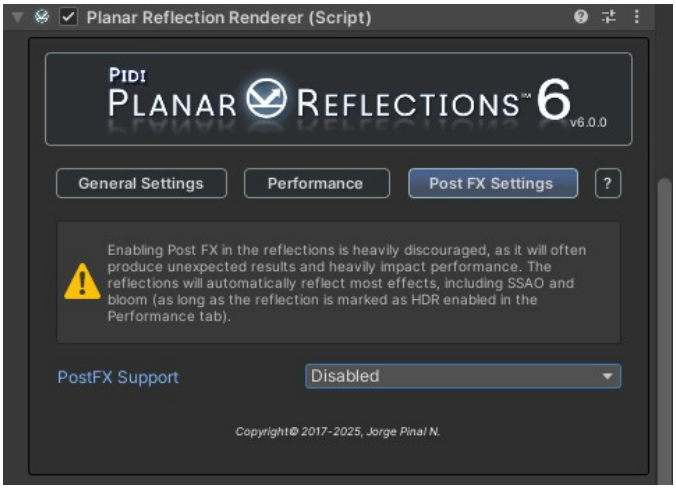
Reflections can be further optimized by **reducing their framerate**. Depending on your game's style and target platforms it may be desirable to use a lower framerate in your reflections to save some performance. However, by default, this is disabled and the framerate is set to zero instead (which means that reflections are updated every single frame).

The final resolution of the reflection can have a huge impact in performance, especially in scenes with complex shaders and visual effects. The resolution of the reflection can be set manually, or it can be based on the actual screen size the game is being rendered at, to ensure that the reflections keep a consistent quality at different screen resolutions.

Additionally, certain features applied to the output can help ensure a higher quality with a lower performance cost. Mip Maps paired with Anti-aliasing can allow for the reflections to be blurred with PBR enabled shaders as well as improve the look on lower resolution outputs, while HDR reflections will automatically be compatible with Bloom and other post process effects without the overhead of actually computing Post FX on the reflection itself.

Post FX Settings

Each Reflection Renderer component can set up support for Post Process FX with their unique volume layer masks. However, in most cases, it is heavily discouraged to enable Post FX support for the reflections themselves. SSAO and most basic effects will be reflected without issues in URP out of the box. This is mostly a legacy feature, now obsolete.



Help & Support

The last tab of the UI includes a small guide so that you can request support for our asset in a more efficient way, detailing steps to contact us and the information we require. It also contains a button that will take you to the latest version of this documentation file

Planar Reflection Caster

The **Planar Reflection Caster** component is the script that takes a reflection and assigns it to a material. It must always be attached to a Mesh Renderer component, and it will automatically track all of its materials and enable per-material settings to define whether the material uses the **Reflection Color** texture (_ReflectionTex), the **Reflection Depth** texture (_ReflectionDepth) and whether an additional Blur Pass should be used to blur the final output of the reflection.

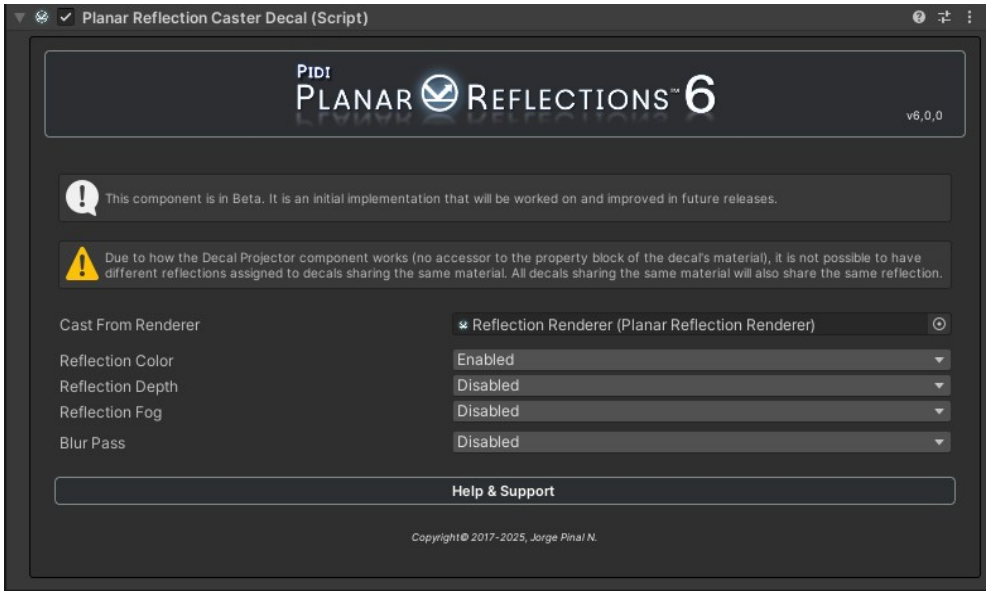
This blur pass should not be confused with the automatic blur applied through PBR shaders. Because of this, we recommend you disable the blur pass in most cases. It is provided as a form of backwards compatibility with shaders from previous versions.

The Reflection Fog pass can also be enabled from here, to simulate and closely match the fog present in the scene within the reflection itself. If you are working with URP, Reflection Fog can be disabled in most cases.

Planar Reflection Caster Decal

This feature is available only in the Universal Rendering Pipeline

The **Planar Reflection Caster Decal** is a special class that allows you to add real-time planar reflections to a decal with a compatible shader. A demo decal shader with support for reflections is provided.



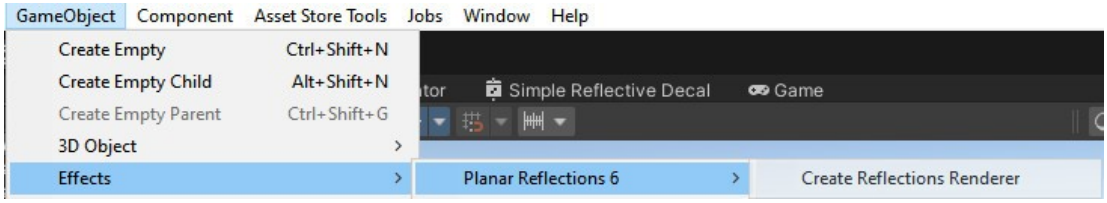
Its functionality is in essence the same as a normal Planar Reflection Caster component, but due to the way that Decal Projectors work you can only assign a single reflection to a single material. If you want different decals with the same material to show different reflections (for example, multiple puddles in different rooms / floors) you will need to use material variants to ensure that the reflections assigned to one decal do not override the reflection of another one with the same material.

Also, because Decals do not invoke the **OnWillRenderObject** function in Unity (as they are not really meshes nor usual renderers) having a reflective decal in the scene tied to a reflection renderer will disable the visibility optimizations in said Reflection Renderer, triggering its update whether decals and other reflective objects are visible or not.

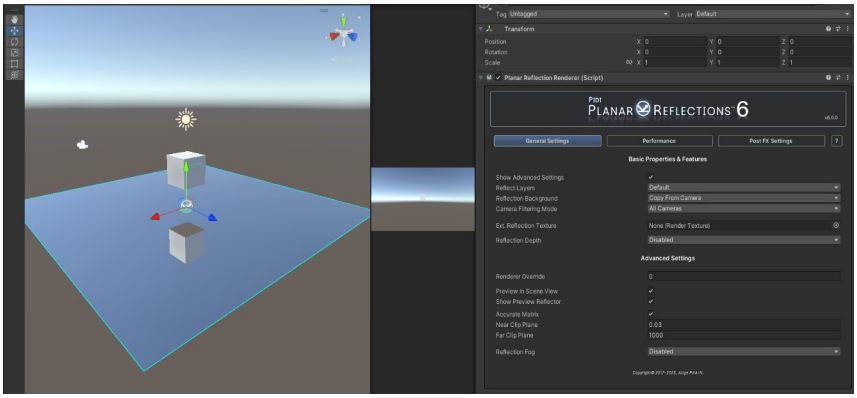
Adding Reflections to a scene

Adding reflections to a scene with PIDI Planar Reflections 6.x.x is a very simple process that takes only a few minutes. To simplify the workflow of adding reflections in real-time and ensuring that they are as re-usable as possible within a level the process has been split into two main components, the Reflection Renderer which generates a reflection and its depth pass and a Reflection Caster which assigns it to a material and actually displays it in the scene.

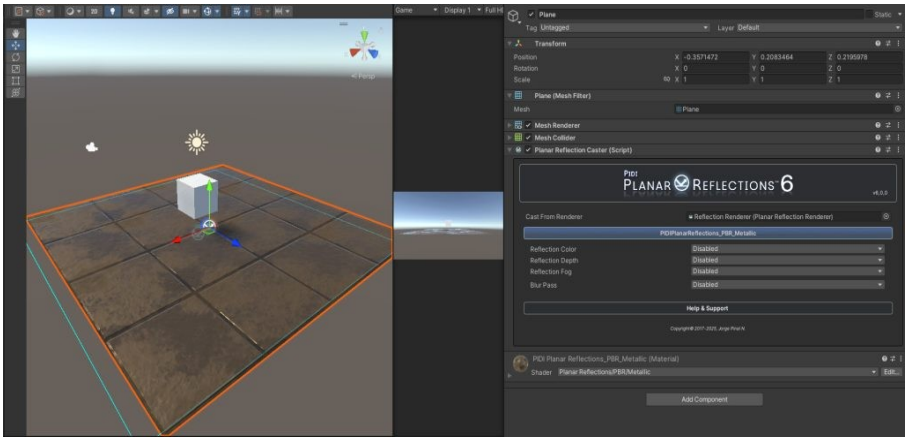
To add a Reflection Renderer to the scene go to the GameObject tab on the top of the Unity Editor, then to Effects, Planar Reflection 6, Create Reflections Renderer.



This will create an empty reflection renderer in the middle of the scene. By default, it will not display the preview reflection renderer but you can enable it easily on the **General Settings** tab of the Reflection Renderer component.

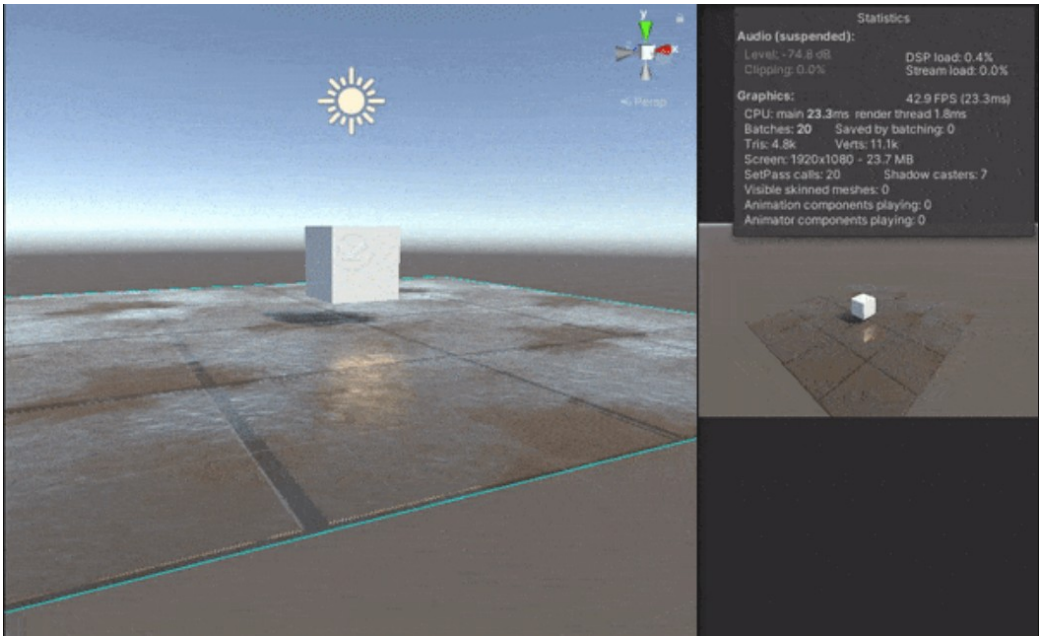


The reflection will not be visible at all on the Game View as it is not being displayed on any mesh at the moment. Let's add a plane to the scene and use one of the demo materials included with the asset on it. We will also add a Planar Reflection Caster component to it.



Then, as the last step, assign your Reflection Renderer to the Cast From Renderer slot and enable the Reflection Color feature in order to display the Reflection Color texture over the surface.

With this, the reflection is ready and will be displayed without issues in both the Game and Scene views. With Planar Reflections 6 you can also use our accurate roughness / smoothness based blur out of the box, integrated within the shader itself, without needing to enable the Blur Pass in the Reflection Caster component and with better performance.



Advanced Topics

In this section we will cover some advanced uses of the PIDI Planar Reflections 6.x asset, including how to create custom shaders that support the reflections generated by the tool using ShaderLab and ShaderGraph as well as the specific utilities provided for each of them.

We will also go over some optimization tips when using reflections on your scenes, what is a depth pass and a blur pass and how you can integrate them into your own shaders.

Depth Pass

Each Planar Reflection Renderer component can have its own optional depth pass. This depth pass stores the distance between the mirror's surface and the objects reflected on it based on the depth of the virtual camera used to render it. This depth pass can be used to create complex fading effects or, in the case of the included PBR based shaders, to simulate contact reflections (that is, reflections that are sharper the closer an object gets to the reflective surface).

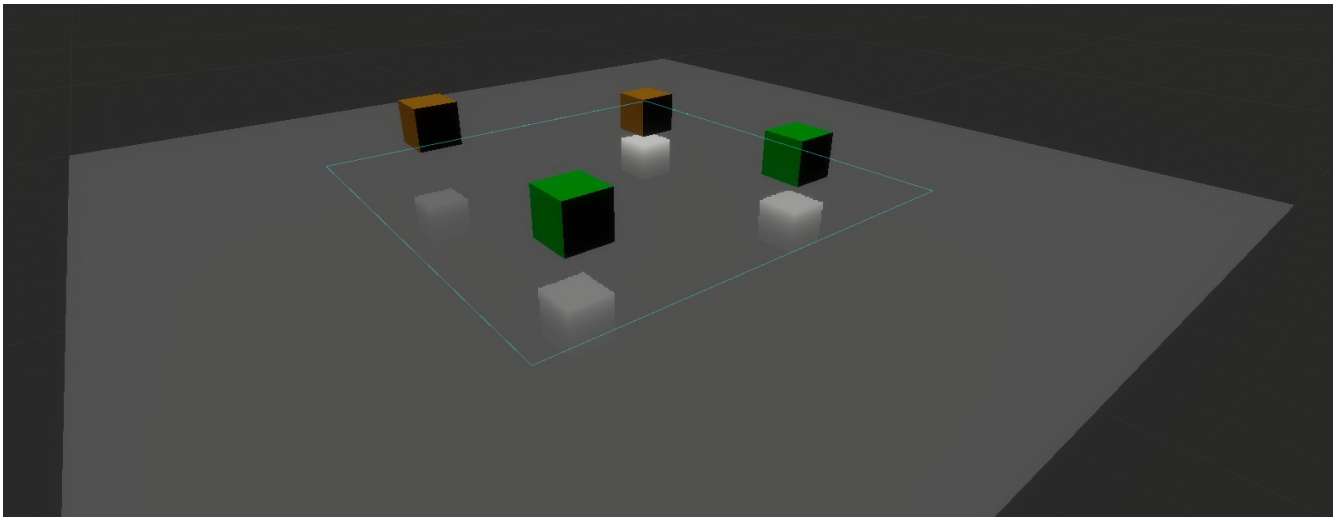
There are two steps to enable depth in your reflections. First, you must enable the depth pass in the Reflection Renderer to ensure that the pass is generated at all:

Then you have to enable the depth pass in your Reflection Caster for every material that will use it, so that the corresponding texture is sent to the material.

Once you've done this, your shaders can access the property called "_ReflectionDepth" and use its Red Channel, which now contains a default depth texture, and use it for all sorts of effects.

Please remember that not all platforms support depth textures and that some pipelines (like Universal RP) need to have the depth rendering feature enabled from their general settings.

For shader programming purposes, when depth is disabled in either the Reflection Renderer or the Reflection Caster it is set to black by default.



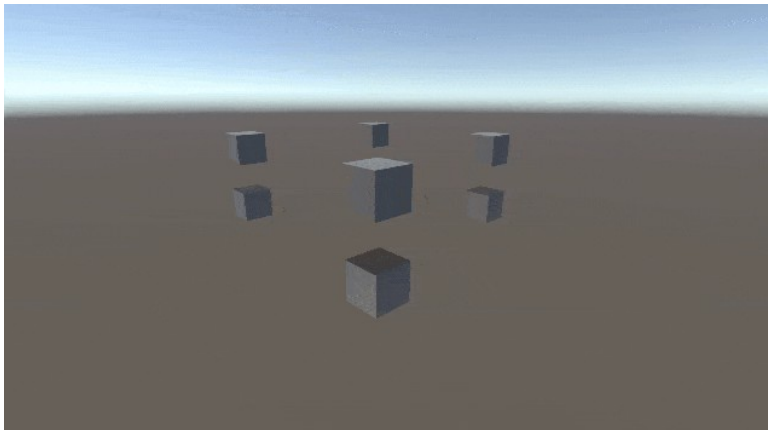
Depth pass debug

Reflection Only Mode

In some cases you may want to composite reflections on top of other effects, custom backgrounds or real video-feedback (for example, in the case of AR). For these cases there are a few shaders provided that automatically remove any backgrounds from the reflective plane leaving only the actual objects being reflected. This is achieved by testing the alpha value of the solid color that the reflection camera is clearing against or, if more precision is needed, by testing against the depth pass of the reflection.

To use them, simply set the **Reflection Background** on your Planar Reflection Renderer component to “Transparent”:

And make sure you are using a Reflection Only shader on your reflective plane. If needed, enable the Depth features in both the Caster and the Renderer. For additional flexibility, the included shader allows you to control the opacity of the final reflections so that you can more easily integrate it with any background you choose.



Custom Legacy RP Shaders

While several different shaders are provided for the Legacy Rendering Pipeline that cover a wide variety of uses including PBR materials, water shaders and basic mirror-like shaders, there may be a need to create your own for the specific project you are working on.

A CGInclude file is provided with the asset that contains several helpful functions that should simplify the integration of the asset with custom made shaders. Helpers for getting the reflection UVs as well as for generating a fully PBR adjusted reflection color (with and without contact depth) are provided.

half2 screen2ReflectionUVs(float4 screenPosition)

This function turns the screen position (in Unity shaders IN.screenPosition) into a set of usable UVs to unwrap the reflection data (both color and depth).

half4 PBRBasedBlur(half4 albedo, half smoothness, half4 screenPosition, half maxBlur, half3 viewDir, half3 normal)

&

half4 PBRBasedBlurDepth(half4 albedo, half smoothness, half4 screenPosition, half maxBlur, half3 viewDir, half3 normal)

This function takes several properties from the PBR shader (the same that will be sent to the corresponding PBR outputs) such as albedo, smoothness and normal, as well as the reflectionUVs, a maxBlur value (to define how much will the reflection blur on very rough surfaces) and a view direction to apply a simple fresnel effect. It can be used as follows:

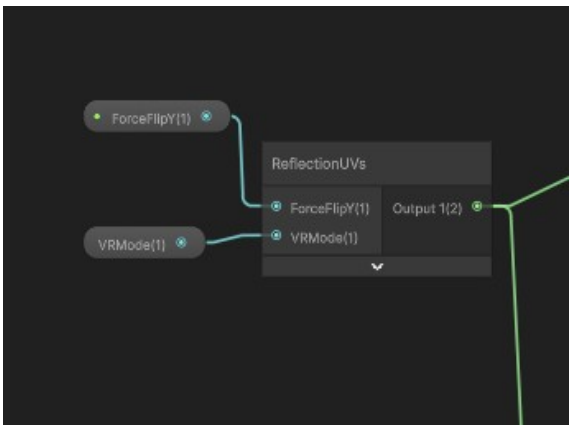
```
half4 e = tex2D( _EmissionMap, mainUV );
e.rgb *= _EmissionColor.rgb * 16 * _EmissionColor.a;

#if !_USE_DEPTH
half4 reflectionColor = PBRBasedBlur( o.Albedo, o.Smoothness, IN.screenPos, 32, IN.viewDir, o.Normal );
#else
half4 reflectionColor = PBRBasedBlurDepth( o.Albedo, o.Smoothness, IN.screenPos, 32, IN.viewDir, o.Normal );
#endif
o.Emission = e.rgb + lerp( reflectionColor, reflectionColor * (1-e.a), _EmissionMode );
```

This is an example of how to add accurate reflections to a PBR shader using the Specular workflow in the Legacy RP (it is based around the Specular shader included with the asset). Using the functions included in the CGInclude file as well as a shader_feature called USE_DEPTH to define whether a depth pass will be used or not (which can save some performance at the shader level by reducing the amount of texture operations).

ShaderGraph Nodes

In the Universal RP you can easily create custom shaders that support real-time reflections by using the included ShaderGraph sub-graphs that handle reflection UVs (for simple reflections) and full PBR based workflows. These graphs can be edited and extended as you see fit in order to adapt to your project's purposes.

ReflectionUVs node

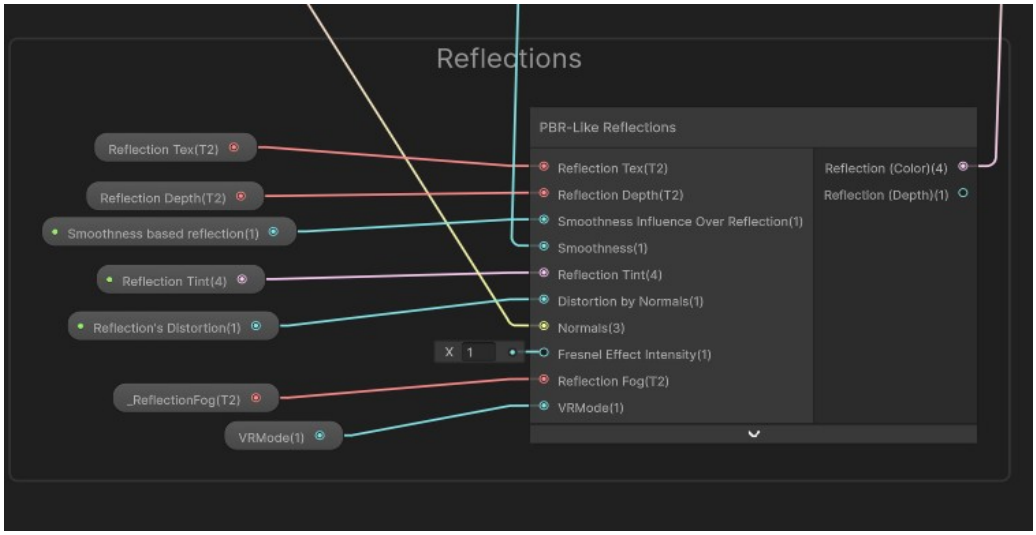


This node automatically generates a Vector2 type output containing ready to use UVs to use on either the Reflection color or Reflection depth textures.

The optional ForceFlipY value can either be 0 or 1 and defines whether the reflection should be flipped vertically.

The VRMode value should be a non-exposed float property called _VRMode, which will be set automatically by the Planar Reflection Caster when a camera using VR requires a reflection pass, triggering all the necessary conversions internally so that the reflection can work as expected in stereoscopic mode.

PBR Like Reflections node



The PBR-Like reflection node generates a PBR based reflection that takes into account the smoothness of the material, its normal and a simple fresnel effect.

It takes as inputs the **Reflection Tex** and **Reflection Depth** textures, a value to define how much the smoothness of the material will affect the sharpness and intensity of the reflection, the smoothness of the material as a float value, a color for the reflection's final tint, a value controlling how much the normals of the material will distort the reflection and a vector3 containing the normal data of the material (the same value that will be sent to the Normal output of the shader), and finally a value specifying the intensity of the fresnel effect.

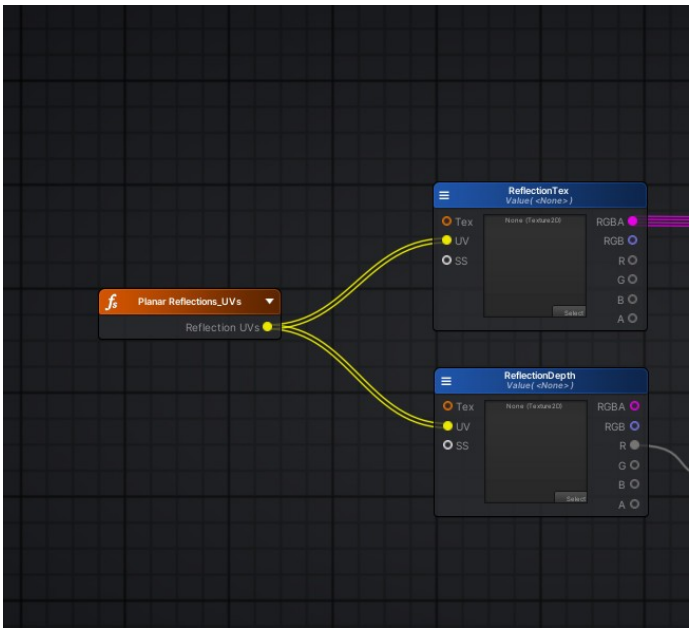
Just like before, the VRMode value should be a non-exposed float property called _VRMode, which will be set automatically by the Planar Reflection Caster when a camera using VR requires a reflection pass, triggering all the necessary conversions internally so that the reflection can work as expected in stereoscopic mode.

Bonus: Amplify Shader Editor Nodes

For users that create their shaders using Amplify Shader Editor we also provide a few custom nodes that will make the integration process with PIDI Planar Reflections much easier. To use these nodes in your project, first make sure that Amplify Shader Editor is installed in your project. Then go to theAdditional Resources folder included with the PIDI Planar Reflections 6.x.x asset and unpack the Unity package included in the Amplify Shader Editor Nodes folder.

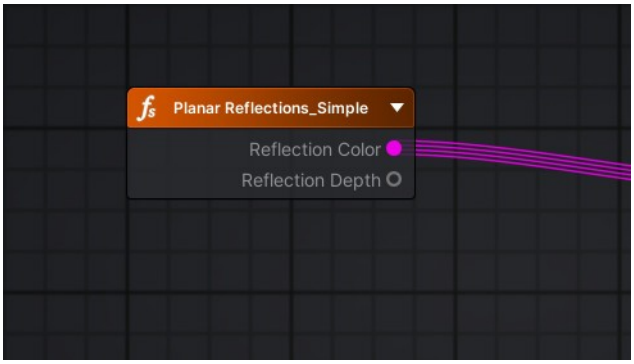
Planar Reflections_UVs

This function generates ready-to-use UVs to unwrap both the reflection's color and depth. They are compatible with VR and standard rendering modes.



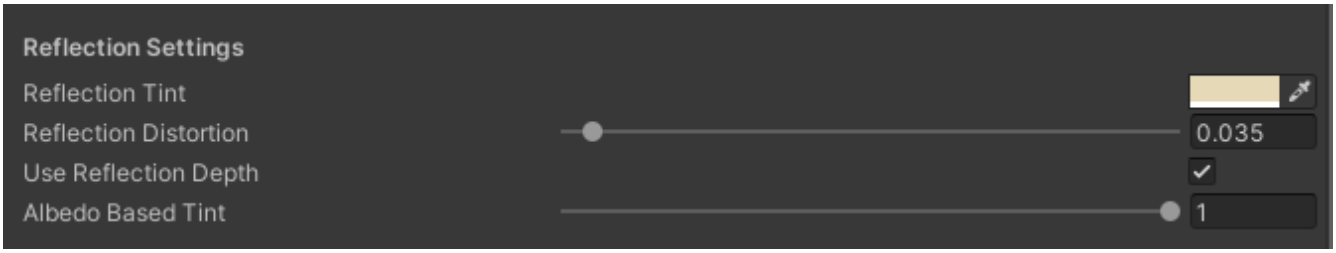
PlanarReflections_Simple

This node allows you to add simple reflection color and depth to your shaders without any needed extra inputs nor setup. Simply drop into an Amplify based shader and use it in any way you see fit. The depth output is not processed in any way nor adjusted for eye / camera based depth, it is the value in its raw format.



PlanarReflections_PBR

This node allows you to add PBR based reflections to your Amplify made shaders through an easy to use node that only takes the normal, smoothness and albedo values from your material and uses them to generate the appropriate Reflection Color and depth outputs. The Depth in this case has been processed and is ready to use for contact depth and other similar effects (which are also automatically handled for the reflection color). The node also adds automatically all the parameters needed to control the reflection to your shader, using appropriate headers and spacing to display them in the UI.



Bonus: Better Shaders Sample

A minimal sample of how to integrate our asset to Better Shaders made shaders is included with the asset, in the Additional Resources folder. The process is quite straightforward and very similar to the way it is handled in standard ShaderLab. Be aware that Better Shaders is not officially supported and you are expected to handle the implementation of our asset with your own custom shaders by yourself, and some proficiency in writing shaders is expected in order to integrate our asset and Better Shaders made shaders. This small guide is simply a starting point in how to do so.

First, you will need to have a _ReflectionTex property declared as a texture sampler. Then you will need to unwrap it in screen space, using the screen position coordinates as its UVs. If you want to support VR-enabled reflections you will also need a float parameter called _VRMode and to do some small modifications to the x-coordinate depending on which eye is currently rendering.

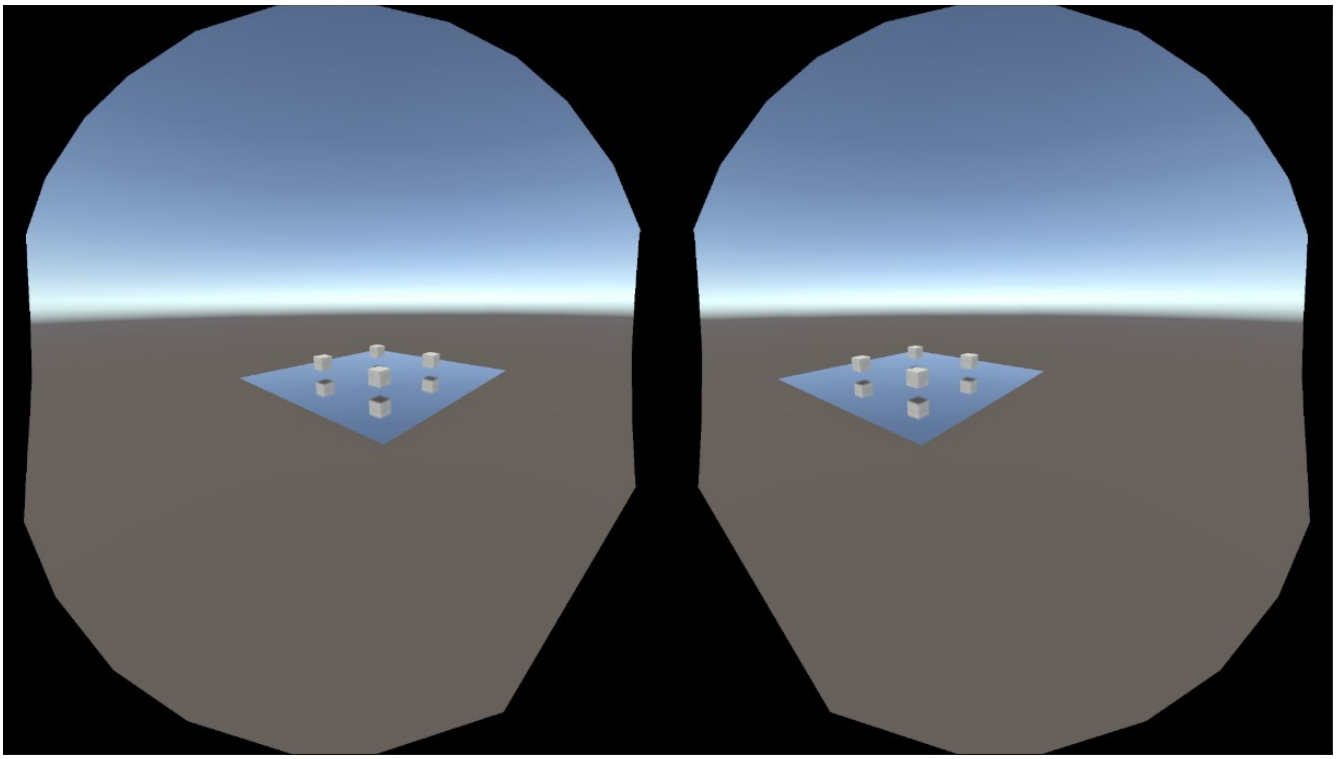
```
BEGIN_OPTIONS
END_OPTIONS
BEGIN_PROPERTIES
//Our property for Reflection textures has to be the Reflection Tex property
[HideInInspector]_ReflectionTex("Reflection Tex", 2D) = "white" {}
END_PROPERTIES
BEGIN_CBUFFER
END_CBUFFER
BEGIN_CODE
sampler2D _ReflectionTex;
float _VRMode;
void SurfaceFunction(inout Surface o, ShaderData d)
{
//The Reflection UVs will be the screen coordinates with an adjusted x coordinate for VR support
float2 refUV = d.screenUV;
refUV.x = lerp(refUV.x, refUV.x / 2.0 + 0.5 * unity_StereoEyeIndex, _VRMode );
o.Albedo = tex2D(_ReflectionTex, refUV).rgb;
}
END_CODE
```

Experimental: VR Support

With PIDI Planar Reflections 6 we have added support for VR devices but in an experimental fashion. This means that no specific devices are officially supported, bugs may not be fully solvable and the feature has not been tested on-device, so you will need to use it at your own risk.

It has been thoroughly tested in simulation mode with different plugin providers such as Open VR, Mock HMD, Meta's Quest simulator, etc. and in all tests it provided accurate reflections. There is no additional setup you need nor any settings to modify, reflections will automatically adapt and adjust when you set any of your cameras to be a stereo-enabled camera and the corresponding Unity.XR modules are enabled.

VR support works on both the Universal and the Legacy Rendering Pipeline and is tailored for Single-pass stereo rendering. It may be compatible with Multi-pass in the Legacy Pipeline, but this has not been thoroughly tested so Single-pass instanced is the recommended way to use VR reflections.



Optimization Tips

Rendering a reflection in real-time involves in most cases re-rendering your scene once for every single camera that views the reflection and for every reflection in the scene. This can cause performance to drop significantly if too many reflections are added to a scene and many of them are visible at once. While the actual cost of rendering the scene for a reflection is usually smaller than the cost of rendering the scene through the main camera there are some additional steps that you can take to improve performance even more.

Control Shadows & LOD

Use a lower LOD bias for your Reflection Renderer so that the reflections use lower quality models, reducing their performance impact. You can also change the Max. LOD value to ensure that only low poly versions of your models are reflected since, in most cases, players will not notice these smaller differences. Additionally, you can disable shadows in the reflection unless completely necessary to reduce the amount of draw calls generated by every mirror in scene.

Lower the reflection’s resolution

If your game uses heavy shaders and post-process FX or has a lot of overdraw then reducing the resolution of the reflections may be helpful. Using a slight blur pass on them may also reduce the sharp edges and make them appear softer and more natural. On top of this, you can also set the reflection’s resolution to depend on the main screen’s resolution so that players using older devices and switching to lower resolutions get lower quality reflections automatically.

Limit the reflection’s framerate

Lastly, you can limit the reflection’s framerate to gain some additional performance. This however is not recommended unless your game runs at a very high framerate by default as with a lower framerate the reflections may seem too choppy producing an unpleasant effect in the scene.