



THE DEFINITIVE GUIDE TO

Lighting and environments in the High Definition Render Pipeline

Contents

Introduction	9
HDRP lighting and environments	10
Installation	12
System requirements	12
Unity Hub	13
Package Manager installation	14
HDRP 3D sample	16
More HDRP sample content	18
Project Settings	21
Graphics Settings	22
Quality Settings	22
Optimizing HDRP	24
HDRP Global Settings	24
Enabling HDRP features	24
Forward vs Deferred rendering	26
Customizing the render path	27
More about rendering paths	27
Forward rendering	27
Deferred shading	28
Anti-aliasing	30
Multisample anti-aliasing (MSAA)	30
Post-processing Anti-aliasing	32
Volumes	34
Local and Global	35

Performance tip.	37
Volume Profiles	37
Volume Overrides	37
Overrides workflow.	38
Blending and priority.	39
Exposure	41
Understanding exposure value	41
Exposure value formula	43
Exposure override.	44
Fixed mode	44
Automatic mode	44
Metering mode options	45
Automatic Histogram	46
Curve Mapping	47
Physical Camera	48
Additional Physical Camera parameters	49
Lights	51
Light types.	51
Shapes	52
Color and temperature	53
Additional properties	53
Light Layers.	54
Light Anchors	56
Physical Light Units and intensity	57
Units	57
Common lighting and exposure values	58
IES Profiles and Cookies.	59

HDRP global illumination	60
Understanding global illumination	60
HDRP global illumination features	61
Baked global illumination	62
Lightmapping workflow	63
Optimizing lightmaps	65
GPU lightmapping	66
Lightmap UVs	66
Real-time global illumination	67
Enlighten GI	67
Using Enlighten Realtime GI	68
Screen Space Global Illumination	69
Light probes	70
Light Probe Groups	70
What is the Sponza Palace?	72
Adaptive probe volumes	72
Environment lighting	76
HDRI Sky	77
Animating HDRI skies	78
Gradient Sky	79
Physically Based Sky	79
Color tip	80
Ray tracing and path tracing	81
Setup	81
Overrides	82
Performance	86
Path tracing	87

DirectX 12	89
Fog and atmospheric scattering	90
Global fog	90
Volumetric Lighting	93
Tips for volumetric lighting and shadows	93
Local Volumetric Fog	95
Shadows	97
Shadow maps	97
Shadow Cascades	98
Contact Shadows	100
Micro Shadows	101
Area light soft shadows	102
Reflections	103
Screen Space Reflections	104
Reflection Probes	105
Optimization tips	106
Planar Reflection Probe	106
Sky reflection	107
Reflection hierarchy	107
Reflection Proxy Volumes	108
Real-time lighting effects	109
Screen Space Ambient Occlusion	109
Screen Space Refraction	111
Post-processing	112
Post-processing overrides	113
Tonemapping	113
Shadows, Midtones, Highlights	114

Bloom	115
Depth of Field	116
White Balance	118
Color Curves	118
Color Adjustments	119
Channel Mixer	119
Lens Distortion	120
Vignette	121
Motion Blur	121
Lens Flare	122
Dynamic resolution	125
NVIDIA DLSS (for NVIDIA RTX GPU and Windows)	125
AMD FSR (cross-platform)	126
TAA Upscale (cross-platform)	127
Rendering Debugger	128
Color monitors	131
Support for HDR10 screens	132
Runtime Frame Stats	132
Shaders and Materials	133
Materials samples	133
Material Variants	134
Material properties	135
Transparency	136
Subsurface scattering and translucency	137
Decals	139
Shader Graph	140
HDRP Master Stacks	141

Volumetric Shader Graph fog.....	142
Fullscreen Shader Graphs	144
Terrain	145
Creating terrains	146
Sculpting	146
Texturing and detailing.....	147
Trees and vegetation	147
SpeedTree integration	148
Terrain Tools package	149
Painting Terrain.....	149
Noise Editor.....	151
Terrain Toolbox	152
Ray tracing support for terrains.....	153
HDRP Terrain Demo	153
Clouds	155
Cloud Layer	156
Atmospheric and sun-based lighting	157
Volumetric clouds	158
HDRP clouds presets blending	159
HDRP water system.....	160
Introducing the water system	160
Getting started	162
Unity 2023 new features	162
Water Surface component.....	163
HDRP Water samples	164
Physically based shading.....	166
Simulating waves and wind	168

Swells, agitation, and ripples	168
Current Maps.	169
Procedural rendering	169
Deforming a water surface.	170
Adding foam	171
Surface foam.	171
Foam generators.	172
Decals and masking	173
Decals	173
Water Mask	173
Caustics.	174
Water exclusion.	175
Rendering underwater scenes.	176
Waterline effects and custom pass.	176
Water scripting	177
Performance and optimization.	178
More water system demos.	179
Island scene.	179
Navigating the demo scenes	180
River scene	180
Pool scene	180
Next steps	181
More resources	182



Introduction

Book of the Dead used HDRP to create its atmospheric lighting.

Building a game world means unleashing your creativity on an epic scale.

With the cutting-edge real-time 3D graphics capabilities from the High Definition Render Pipeline (HDRP), artists and developers can take players to visually stunning environments that push the boundaries of game design.

Imagine a sprawling futuristic megacity, glowing with hyperrealistic ray traced reflections and emissive surfaces. Or picture an ancient rainforest, its dense foliage created with [SpeedTree](#) and backlit with subsurface scattering. With physically based rendering, you can paint your scenes with cinematic lighting and capture them with real-world camera exposure.

Need atmosphere or mood? Use volumetric fog and shadows to add depth and immersion to your scenes. Then, layer in fine details with decal projectors, and finish them with professional color grading and post-processing effects like Depth of Field and Bloom.

We've updated this HDRP guide to include our latest suite of worldbuilding tools. Move mountains – literally – or carve canyons with the Terrain tools. Adorn the sun-dappled skies above them with Cloud Layers, and animate the seas, lakes, and rivers below using the new water system.

What story do you want to tell? Let HDRP help bring your vision to life.

HDRP lighting and environments

HDRP extends Unity's existing lighting system with a variety of features to make rendering your scene resemble real-world lighting:

- **Physical light units and advanced lighting:** HDRP uses real-world lighting intensities and units. Match the specs from known light sources and set exposures using physical cameras. Take control over light placement with additional shape options for spot and area lights. Apply real-time effects like the Screen Space Global Illumination and Screen Space Refraction.
- **Skyscapes:** Generate natural-looking skies with varied techniques. Use the Physically Based Sky Volume override to simulate planetary atmosphere procedurally, add volumetric clouds, cloud layers, or apply HDRIs to simulate static skies.
- **Terrains:** Create realistic landscapes with HDRP-enhanced sculpting tools, texture layering, and custom brushes. Paint topography with heightmaps and add vegetation that can sway in the wind to create rich, dynamic scenes.
- **Water system:** HDRP's new Water System simulates interactive water surfaces. Add physically accurate wave simulations, currents, and foam to make your scenes come alive.
- **Fog:** Add depth and dimension to your scenes with fog. Enable volumetrics to integrate fog effects with your foreground objects and render cinematic shafts of light. Maintain per-light control of volumetric light and shadows and use the Local Volumetric Fog component for fine control of fog density with a 3D mask texture.

- **Volume system:** HDRP features an intuitive system that lets you block out different lighting effects and settings based on camera location or by priority. Layer and blend volumes to allow expert-level control over every square meter of your scene.
- **Post-processing:** HDRP post-processing is controlled by a series of Volume Overrides on top of the existing Volume system. Add anti-aliasing, tonemapping, color grading, bloom, depth of field, and a host of other effects.
- **Advanced shadows:** HDRP offers advanced artistic and performance control over shadows. Modify their tint, filtering, resolution, memory budget, and update modes. Accentuate small details and additional depth with contact shadows and micro shadows.
- **Advanced reflections:** Reflective surfaces can use several techniques to render. Reflection Probes offer a traditional reflection mapping approach, with Planar Reflection Probes giving you more advanced options for flat surfaces. Screen-space reflection (SSR) adds a real-time technique using the depth buffer.
- **Extensibility:** HDRP is built on Unity's [Scriptable Render Pipeline](#). Experienced technical artists and graphics programmers can extend the pipeline even beyond what's available out of the box.

Completely new to HDRP? Make sure you read [Getting started with the High Definition Render Pipeline](#).

Installation

Unity 2022 LTS and above includes the HDRP package with the installation to ensure that you're always running on the latest verified graphics code. When you install the most recent Unity release, it also installs the corresponding version of HDRP.

HDRP package version	Compatible Unity version
15.x	2023.1
14.x	2022.3 (used in this guide)
13.x	2022.1

Tying the HDRP graphics packages to a specific Unity release helps ensure compatibility. However, you can also switch to a custom version of HDRP by overriding the [manifest](#) file.

System requirements

HDRP is currently compatible with the following target platforms:

- Windows and Windows Store, with DirectX 11 or DirectX 12 and Shader Model 5.0
- Modern consoles (minimum Sony PlayStation®4 or Microsoft Xbox One)
- MacOS (minimum version 10.13) using Metal graphics
- Linux and Windows platforms with Vulkan

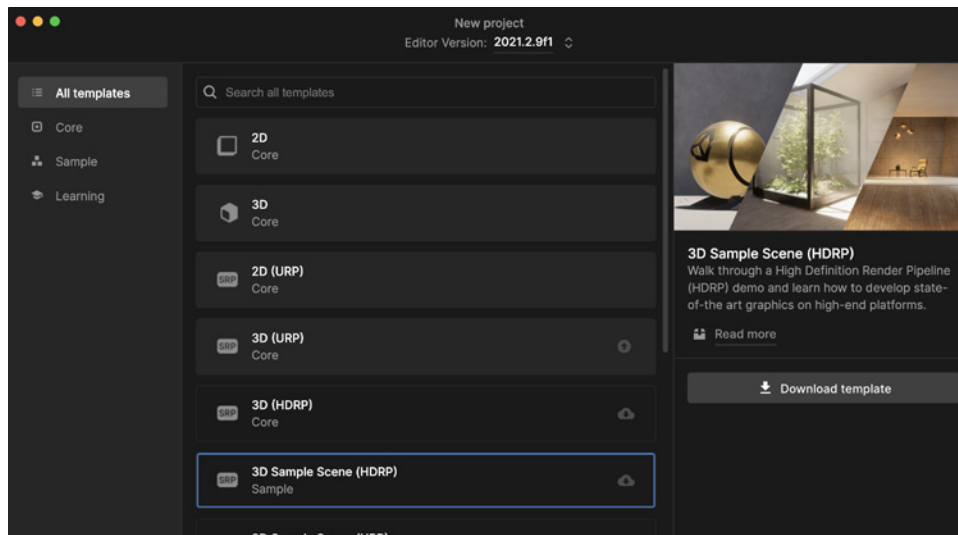
HDRP only works on console and desktop platforms that support compute shaders. HDRP does not support OpenGL or OpenGL ES devices. Refer to the documentation for more complete [requirements and compatibility](#) information.

See “[Virtual Reality in the High Definition Render Pipeline](#)” to learn about supported VR Platforms and devices.

Unity Hub

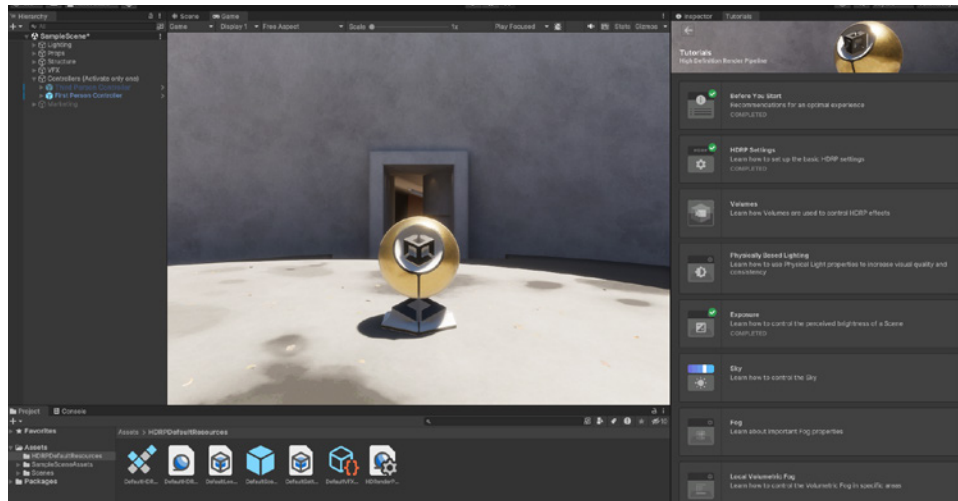
The Unity Hub is the simplest way to set up an HDRP project.

To get started, create a new project. Select either the **3D (HDRP)** empty template or **3D Sample Scene (HDRP)** from the available templates (also called High Definition RP in older versions of the Hub). Choose the latest to import the HDRP package with some example presets.



Select the 3D Sample Scene (HDRP) template.

Load the **Sample Scene**. You should see something like this:

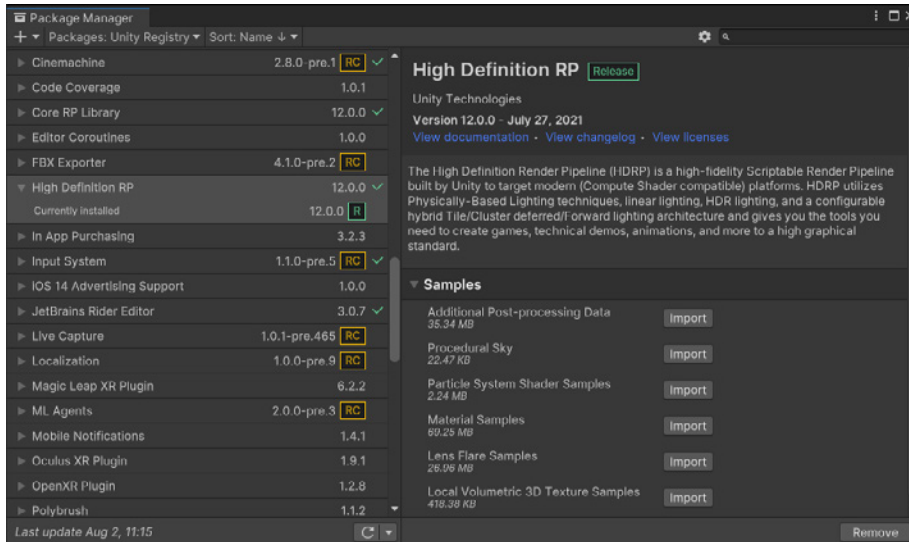


3D Sample Scene project setup

Package Manager installation

If you create your project with the **3D Core template**, Unity uses the older [Built-in Render Pipeline](#). You can migrate the project to HDRP manually from the **Package Manager (Window > Package Manager)**.

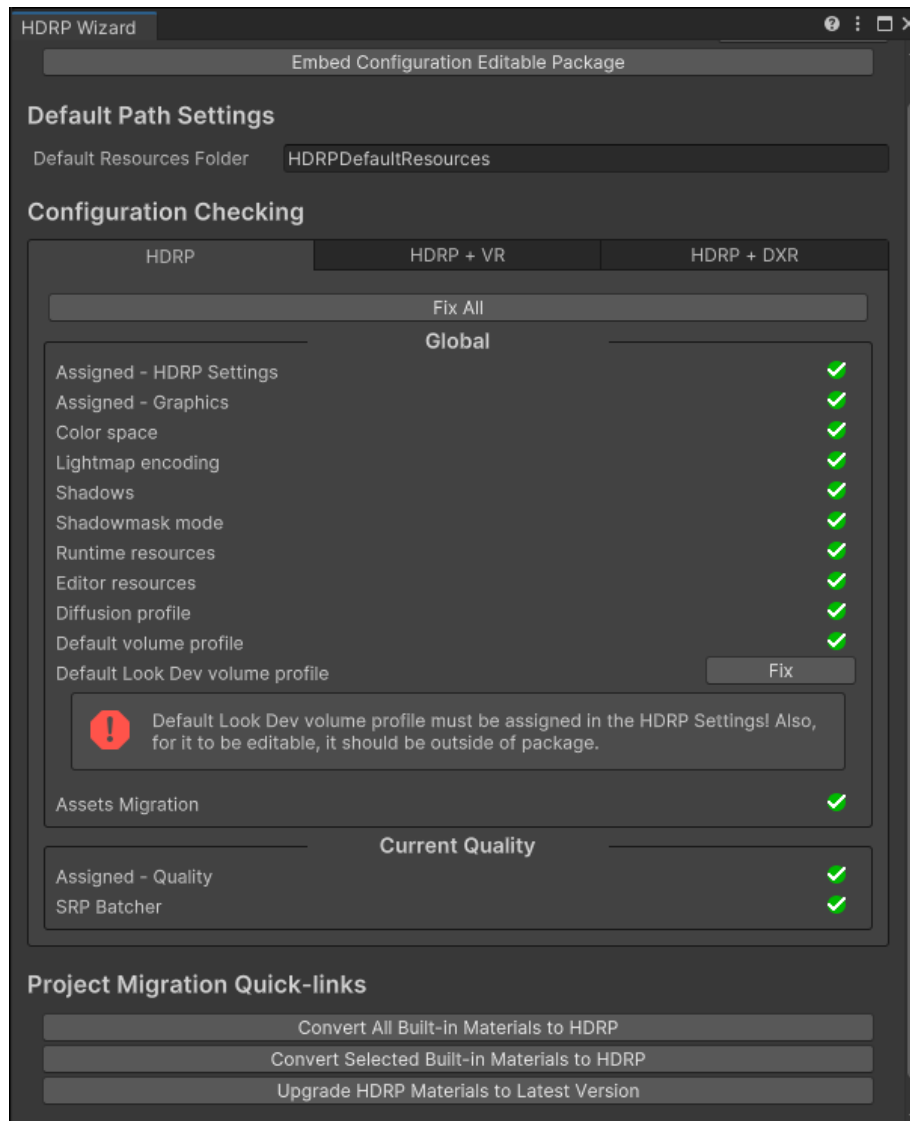
Find the High Definition RP package in the Unity Registry (or use the Search field to locate it), and install.



Installing the Package Manager

If there is a conflict with the current Project Settings, the **HDRP Render Pipeline Wizard** will appear to help you troubleshoot (also found under **Window > Rendering > HDRP Wizard**).

Click **Fix All** under **Configuration Checking**, or click **Fix** for each issue to repair individually. This checklist can help you migrate from a non-SRP project.



The HDRP wizard

When the wizard finishes, a prompt will ask you to create a new HDRP Pipeline Asset. This is a file on disk that will hold your specific pipeline settings. Select Create One to add a new Render Pipeline Asset and assign the file here.

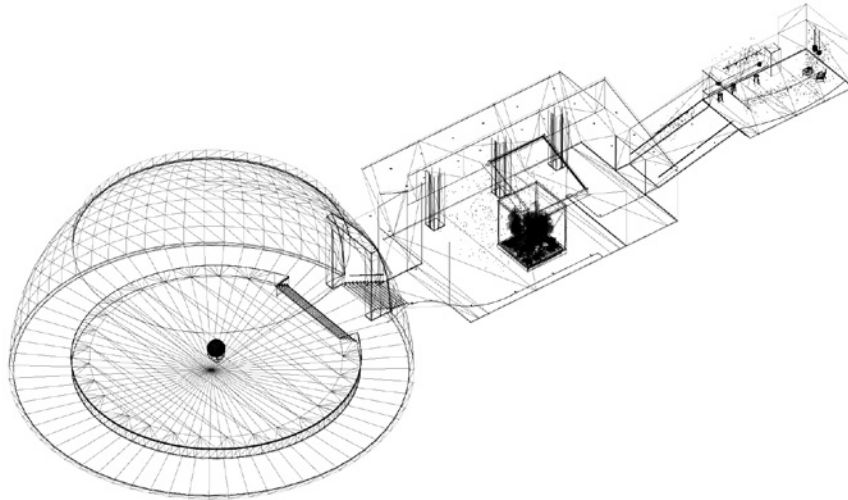
Once HDRP is functioning properly, all of the checkboxes in the Configuration Checking should be green, and the background environment may noticeably change color.

Note that manual installation from a blank project does not import the **3D Sample Scene (HDRP)**. Use the 3D Sample Scene template if you want access to the example assets shown in this guide.

HDRP 3D Sample

The HDRP 3D Sample scene available from the Unity Hub is a template project that helps you get started with HDRP. This lightweight project represents a small game level that you can always load quickly for reference.

We'll use this project periodically to demonstrate many of the HDRP's features in this guide.



A wireframe representing the 3D Sample Scene environment

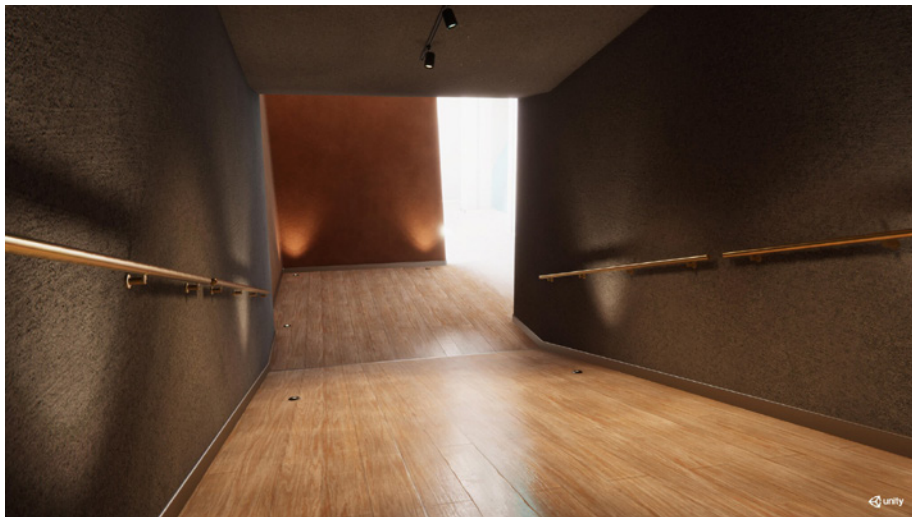
The small, multi-room environment demonstrates three distinct areas with different lighting setups. The directional light representing the sun has a real-world intensity of 100,000 lux, and each location corrects the camera's exposure to match the lighting environment.

Use the WASD keys and mouse to drive the FPS Controller around the level.



The 3D Sample Scene consists of three rooms.

- **Room 1** is a round platform lit by the overhead sunlight. Decals add grime and puddles of water to the concrete floor.
- **Room 2** adds volumetric shafts of light from the skylight, as well as advanced materials for the tree inside the glass case.
- **Room 3** showcases interior artificial lighting and emissive materials.



The 3D Sample is a lightweight project demonstrating HDRP features.

For a deeper look at the HDRP 3D Sample Scene, please check out this [blog post](#) from Unity technical artist Pierre Yves Donzallaz, which describes the template scene in more detail.

More HDRP sample content

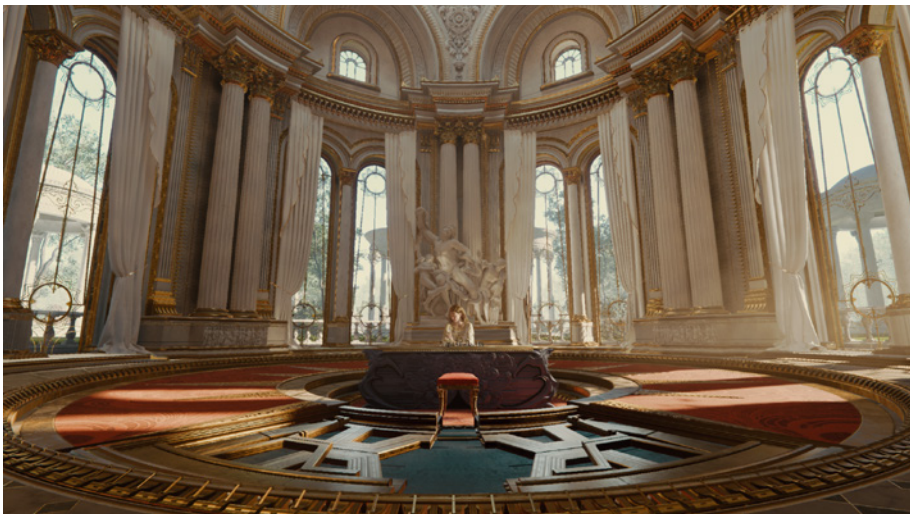
You might find some other projects helpful once you're done exploring the samples.

The [Book of the Dead: Environment](#) is an HDRP interactive demo created to showcase rendering high-end visuals for game productions. The assets in the demo are photogrammetry-scanned real-world objects and textures from an outdoor natural environment. This package has been updated to Unity 2022 LTS and is available in the Asset Store.



Book of the Dead: Environment

The [Enemies demo](#) project not only showcases a digital human character but also includes a mind-bending animated background environment. The demo leverages Probe Volumes, ray traced effects, and integrates native support for NVIDIA's Deep Learning Super Sampling (DLSS), making it possible to operate at 4K resolution.



The Enemies demo project shows off an elaborate interior.

The [Sponza Palace Atrium](#) is widely used by graphics programmers and artists. It provides an ideal lighting test environment, as it features both indoor and outdoor areas. This version has been remastered in HDRP.



The Sponza Atrium

If you're using HDRP with VR, you'll appreciate the [VR Alchemist Lab](#). This project showcases interactive effects in a small medieval laboratory.



VR Alchemist Lab demo

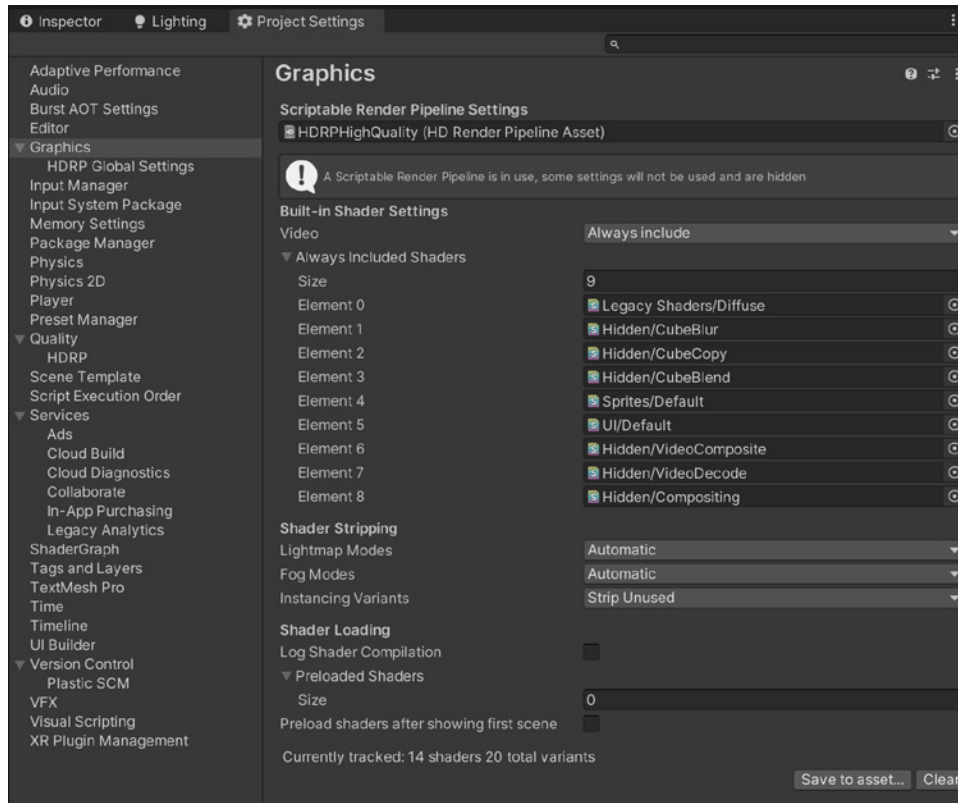
Install the [Cinematic Studio Template](#) from the Unity Hub or Asset Store to learn how to make cinematics or animated films. The template teaches [how to set up and light shots](#) using a funny short movie called *Mich-L*, which mixes stylized and photoreal rendering.



Cinematic Studio Sample

Project Settings

You'll find a few essential settings in the **Project Settings (Edit > Project Settings)** under **Graphics**, **HDRP Global Settings**, and **Quality**.



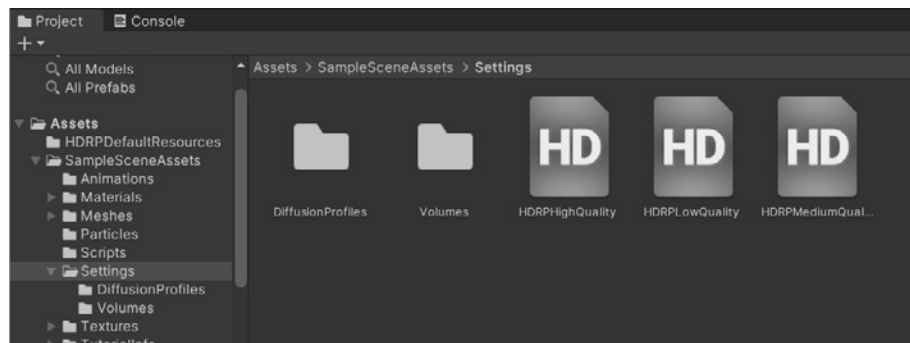
Project Settings

Graphics Settings

The top field, the **Scriptable Render Pipeline Settings**, represents a file on disk that stores all of your HDRP settings.

You can have multiple such Pipeline Assets per project. Think of each one as a separate configuration file. For example, you might use them to store specialized settings for different target platforms (Xbox, PlayStation, and so on), or they could also represent different visual quality levels that the player could swap at runtime.

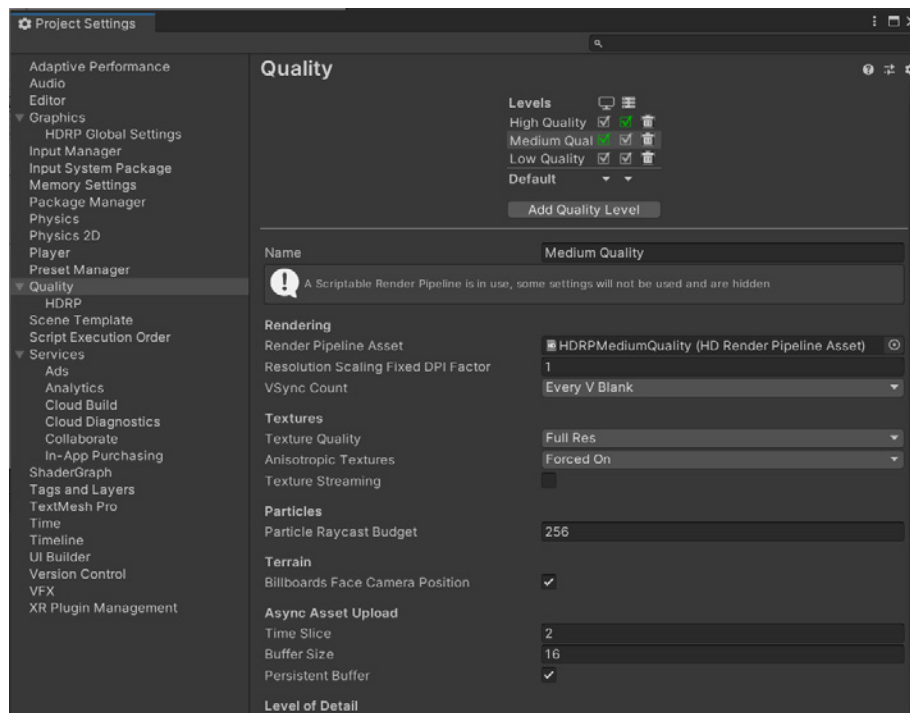
The **3D sample scene** begins with several Pipeline Assets in the Settings folder: **HDRPHighQuality**, **HDRPLowQuality**, and **HDRPMediumQuality**. There is also a **HDRPDefaultResources** folder containing a **DefaultHDRPAsset**.



The 3D Sample Scene includes low-, medium-, and high-quality Pipeline Assets.

Quality Settings

The Quality Settings allows you to correspond one of your Pipeline Assets with a predefined quality level. Select a **Level** at the top to activate a specific **Render Pipeline Asset**, shown in the **Rendering** options.

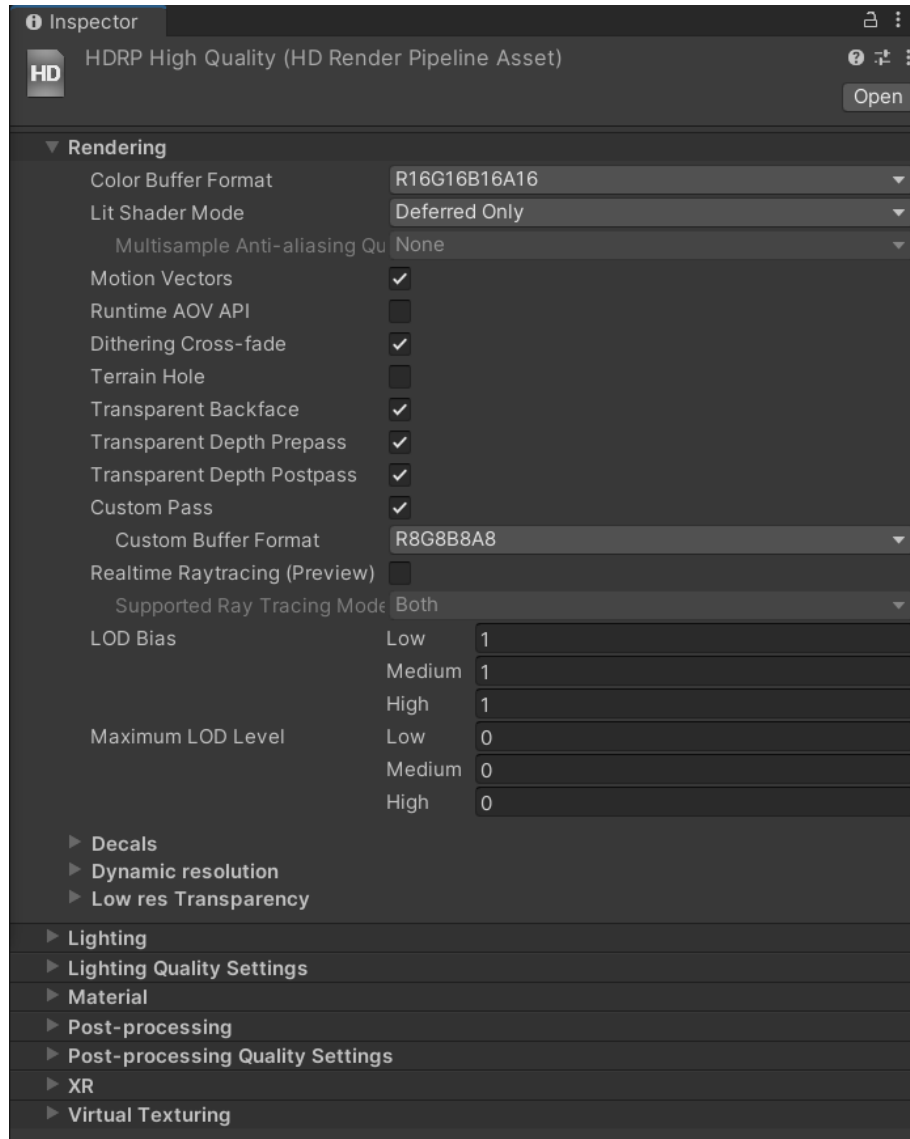


Select a quality level at the top to activate a Pipeline Asset.

You can customize the defaults or create additional Quality Levels, each paired with additional Pipeline Assets.

A Quality Level represents a specific set of visual features active in the pipeline. For example, you could create several graphics tiers within your application. At runtime, your players could then choose the active Quality Level, depending on hardware.

Edit the actual pipeline settings in the **Quality/HDRP** subsection. You can also select the Pipeline Asset in the Project view and edit the settings in the Inspector.



Editing the Pipeline Asset

Optimizing HDRP

Be aware that enabling more features in the Pipeline Asset will consume more resources. In general, optimize your project to use only what you need to achieve your intended effect. If you don't need a feature, you can turn it off to improve performance and save resources.

Here are some typical features that you can disable if you don't use them:

- In the **HDRP Asset**: Decals, low-res transparency, transparent backface / depth prepass / depth postpass, SSAO, SSR, contact shadows, volumetrics, subsurface scattering, and distortions
- In the camera's **Frame Settings** (Main Camera, cameras used for integrated effects like reflections, or additional cameras used for custom effects): Refraction, Post-Process, After Post-Process, Transmission, Reflection Probe, Planar Reflection Probe, and Big Tile Prepass

Read more in this [blog post](#) about getting acquainted with HDRP settings for enhanced performance.

HDRP Global Settings

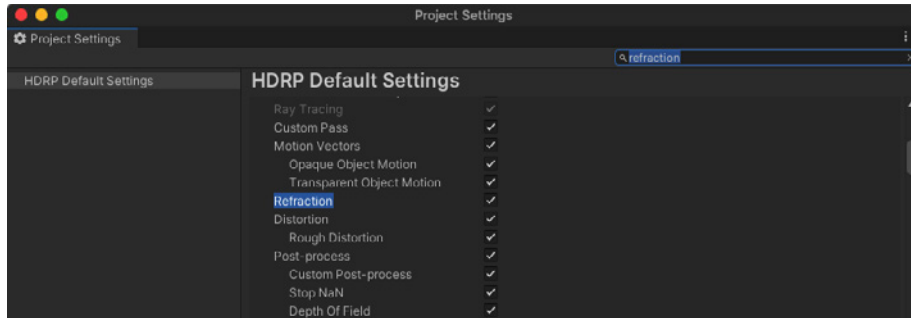
The **HDRP Global Settings section** (or **HDRP Default Settings** prior to version 12) determines the baseline configuration of the project. You can override these settings in the scene by placing local or global Volume components, depending on the camera position (see Volumes below).

Global Settings save in their own separate Pipeline Asset defined at the top field. Set up the default rendering and post-processing options here.

Enabling HDRP features

As you develop your project, you might need to return to the **Global** settings to toggle a specific feature on or off. Some features will not render unless the corresponding checkbox in **HDRP Global Settings** is enabled. Make sure you only enable features you require because they might negatively impact the rendering performance and memory usage. Also, certain settings will appear in the **Volume Profiles**, while other features appear in the **Frame Settings**, depending on usage.

While familiarizing yourself with HDRP's feature set, make use of the top right Search field in the Project Settings. This will only show you the relevant panels with the search terms highlighted.

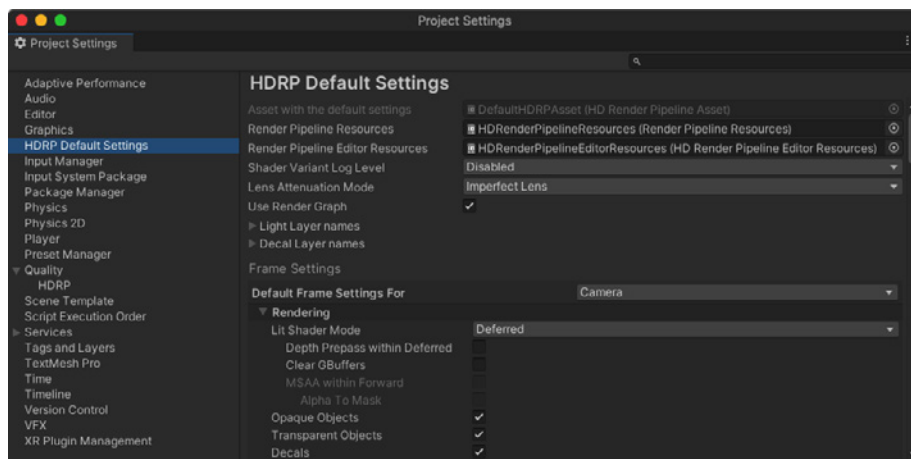


Search for HDRP features

Enabling a feature in the HDRP Global Settings does not guarantee it can be rendered at any time by any camera. You must ensure that the Render Pipeline Asset whose Quality level is selected under **Projects Settings > Quality** supports that feature as well. For instance, to ensure cameras can render Volumetric Clouds, you must toggle them under **HDRP Global Settings > Frame Settings > Camera > Lighting** and in the active Render Pipeline Asset, under **Lighting > Volumetrics**.

Forward vs Deferred rendering

When configuring your HDRP settings in the Pipeline Asset, you will usually start with the **Lit Shader Mode** under **Rendering**. Here you can choose between **Deferred**, **Forward**, or **Both**. These represent the rendering path, a specific series of operations related to how the pipeline will render and light the geometry.



Modifying the default HDRP settings

Customizing the render path

Choose **Forward** or **Deferred** in the **Lit Shader Mode** to set your default rendering path.

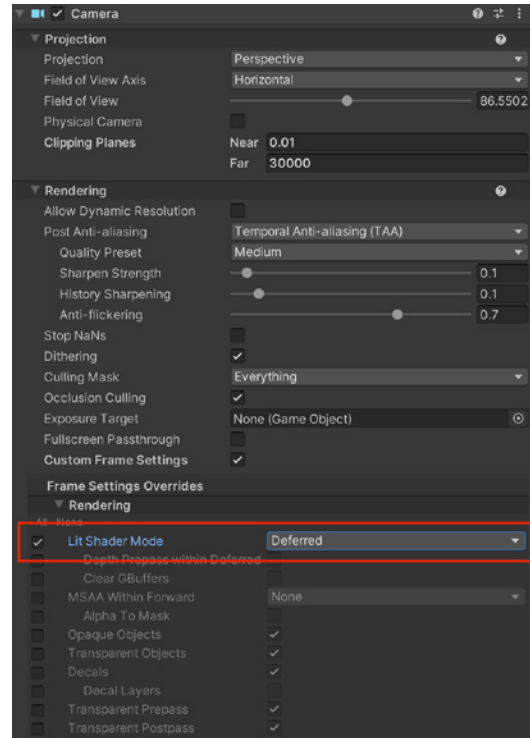
HDRP is flexible and also allows you to choose **Both**. This option lets you use one render path for most rendering and then override it per camera. However, this approach uses more GPU memory. In most cases, it is better to choose either Forward or Deferred.

- To affect all cameras by default, go to **HDRP Default Settings** and locate **Default Frame Settings**. This can apply for a **Camera**, **Baked** or **Custom Reflection**, or **Realtime Reflections**.

In the **Rendering** group, set the render path in the **Lit Shader Mode**.

- For a specific camera, check its **Custom Frame Settings** to override it.

Then, in the **Rendering** group, override and change the rendering path of the **Lit Shader Mode**.



Modifying the custom frame settings for a camera

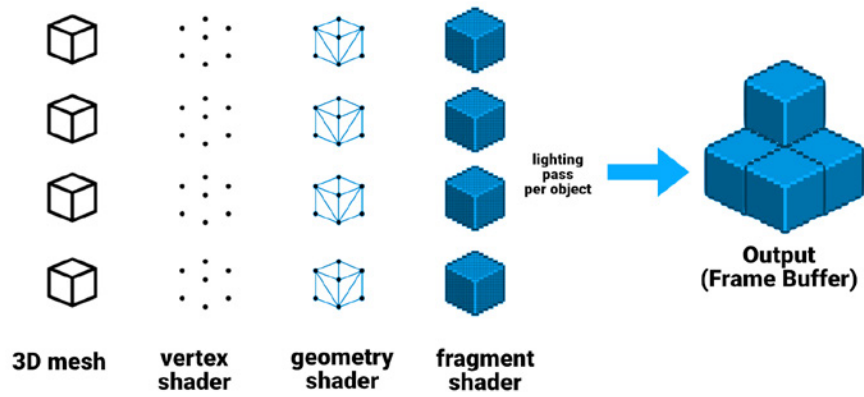
More about rendering paths

You may want to understand how these rendering paths work to see how Lit Shader Mode will impact the other settings in our pipeline.

Forward rendering

In Forward rendering, the graphics card splits the on-screen geometry into vertices. Those vertices are further broken down into fragments, or pixels, which render to screen to create the final image.

Each object passes, one at a time, to the graphics API. Forward rendering comes with a cost for each light. The more lights in your Scene, the longer rendering will take.



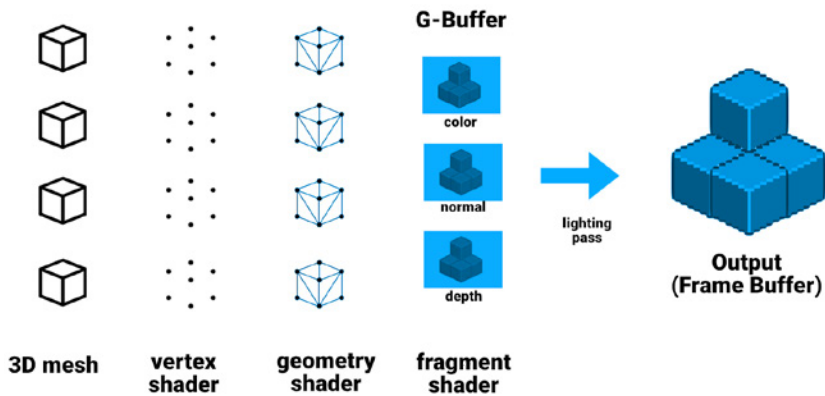
Forward rendering path

Forward rendering draws lights in separate passes. If you have multiple lights hitting the same GameObject, this can create significant overdraw, slowing down when a lot of lights and objects are present.

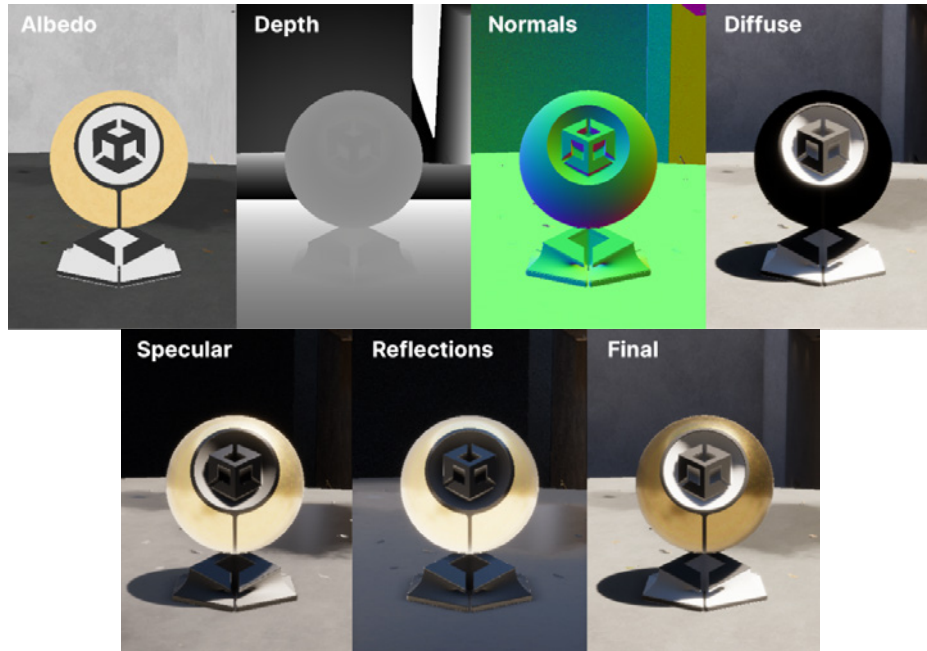
Unlike traditional forward rendering, HDRP does add some efficiencies to the forward renderer. For example, it culls and renders several lights together in a single pass per object material. However, it's still a relatively expensive process. If performance is an issue, you may want to use Deferred Shading instead.

Deferred shading

HDRP can also use deferred shading, where lighting is not calculated per object. Instead deferred shading postpones heavy rendering to a later stage and uses two passes.



Deferred shading path



Deferred shading applies lighting to a buffer instead of each object. Each of these passes contributes to the final rendered image.

In the first pass, or the [G-buffer](#) geometry pass, Unity renders the GameObjects. This pass retrieves several types of geometric properties and stores them in a set of textures (e.g., diffuse and specular colors, surface smoothness, occlusion, normals, and so on).

In the second pass, or [lighting pass](#), Unity renders the Scene's lighting after the G-buffer is complete. Hence, it *defers* the shading. The deferred shading path iterates over each pixel and calculates the lighting information based on the buffer instead of the individual objects.

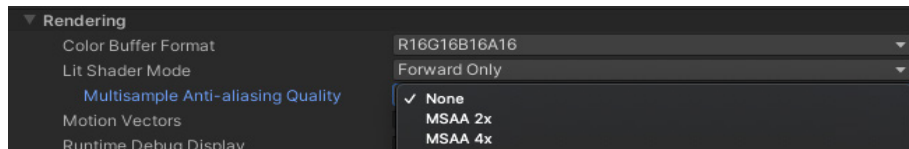
For more information about the technical differences between the rendering paths, see [Forward and Deferred rendering](#) in the HDRP documentation.

Anti-aliasing

The rendering path in the Lit Shader Mode influences how you can use anti-aliasing to remove the jagged edges from your renders. HDRP offers several anti-aliasing techniques, depending on your production needs.

Multisample anti-aliasing (MSAA)

[Multisample anti-aliasing \(MSAA\)](#) is a popular anti-aliasing method among PC gamers. This is a high-quality hardware method that smooths the edges of individual polygons, and it only works with forward rendering in Unity. Most modern [GPUs](#) support 2x, 4x, and 8x MSAA samples.



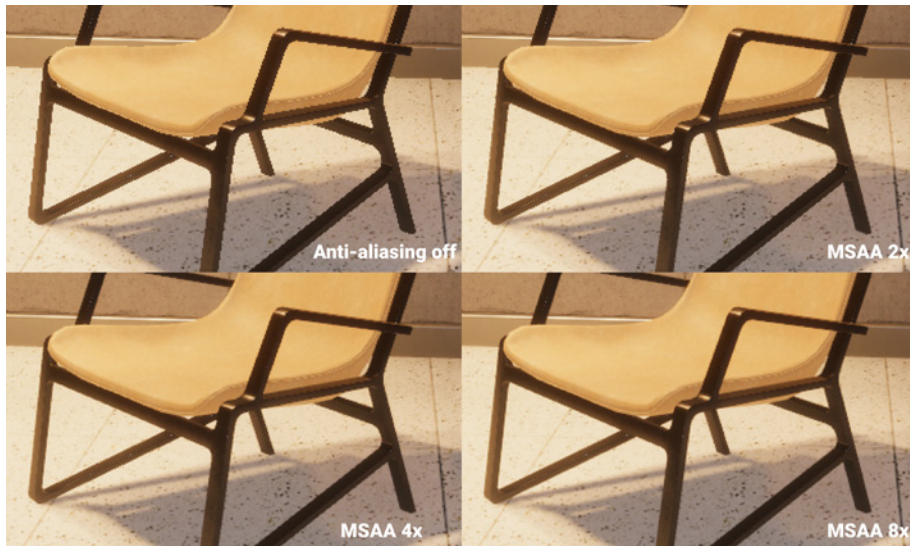
MSAA quality settings

In your active Pipeline Asset, set the Lit Shader Mode to **Forward Only**. Then select **MSAA 2x**, **MSAA 4x**, or **MSAA 8x** for the **Multisample Anti-aliasing Quality**. Higher values result in better anti-aliasing, but they are slower.

We can see this more clearly when we zoom into the camera view.



Original scene



MSAA settings applied to an image

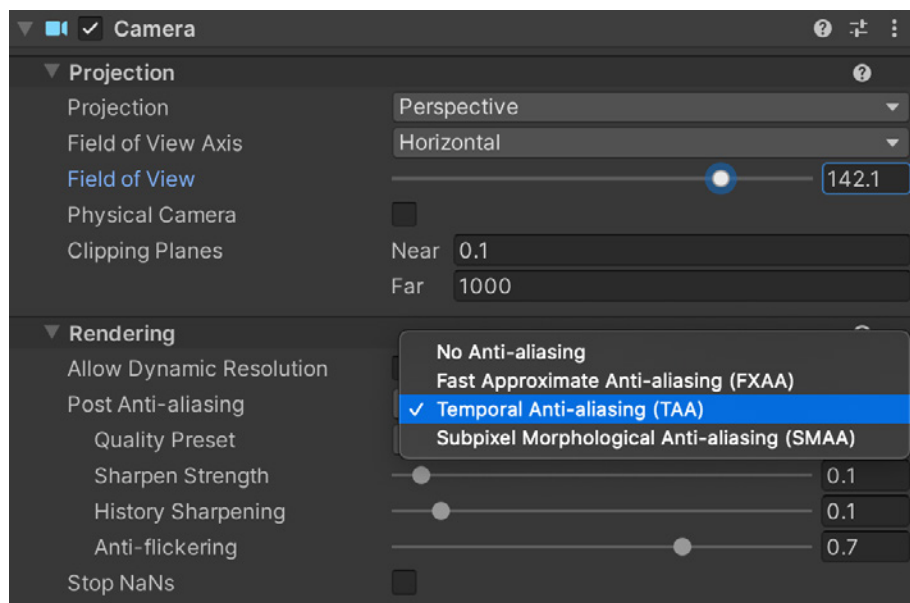
Note these limitations:

- MSAA is incompatible with deferred shading's G-buffers, which store the scene geometry in a texture. Thus, deferred shading requires one of the Post-processing Anti-aliasing techniques (below).
- Because MSAA only deals with polygon edge aliasing, it cannot prevent aliasing found on certain textures and materials hit by sharp specular lighting. You may need to combine MSAA with another Post-processing Anti-aliasing technique (below) if that is an issue.

Post-processing anti-aliasing

Your camera also allows you to apply anti-aliasing as a post-processing technique with the Post Anti-aliasing setting:

- **Temporal Anti-aliasing (TAA)** combines information from past frames and the current frame to remove [jaggies](#) in the current frame. You must enable [motion vectors](#) in order for it to work. TAA generally produces great results, but it may create ghosting artifacts in some situations (e.g., a GameObject moving quickly in front of a contrasting surface). HDRP10 introduced improvements to cut down on typical TAA artifacts. Unity's implementation reduces ghosting, improves sharpness, and prevents flickering found in other solutions.
- **Fast Approximate Anti-aliasing (FXAA)** is a [screen-space anti-aliasing](#) algorithm that blends pixels between regions of high contrast. It is a relatively fast technique that does not require extensive computing power, but it can reduce the overall sharpness of the image.
- **Subpixel Morphological Anti-aliasing (SMAA)** detects borders in the image, then looks for specific patterns to blend. This produces sharper results than FXAA, and it works well with flat, cartoon-like, or clean art styles.



Adjust Post Anti-aliasing on your camera when using deferred shading.

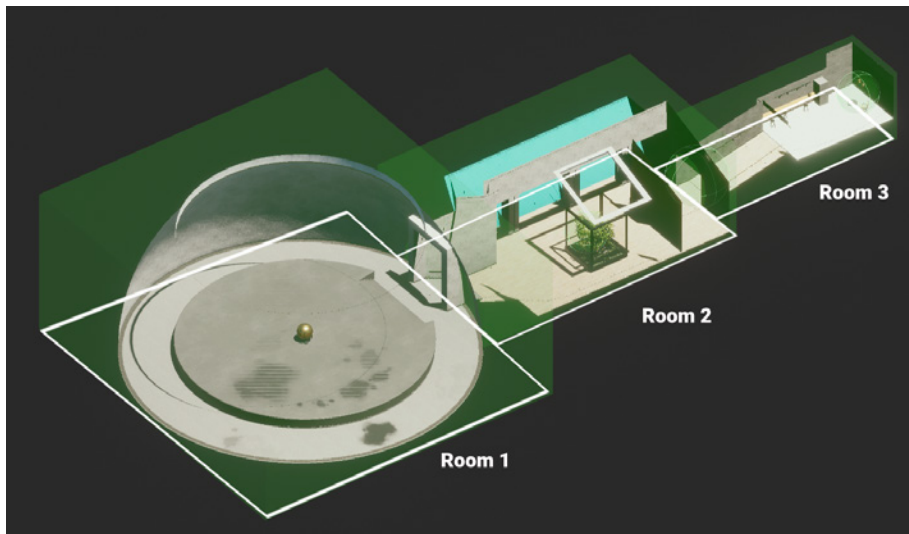


Post-processing anti-aliasing – compare the results of FXAA, SMAA, and TAA settings.

Note: When combining Post-processing Anti-aliasing with Multisample Anti-aliasing, be aware of the rendering cost. As always, optimize your project to balance visual quality with performance.

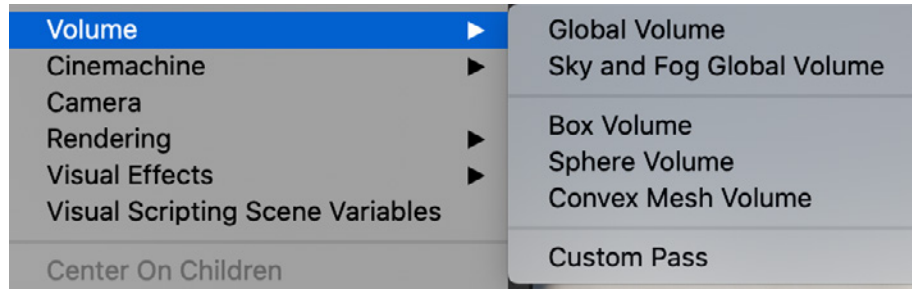
Volumes

HDRP uses a [Volume framework](#). This system allows you to split up your Scene and enable certain settings or features based on camera position. For example, the HDRP template level contains three distinct parts, each with its own lighting setup. Thus, we have different Volumes encompassing each room.



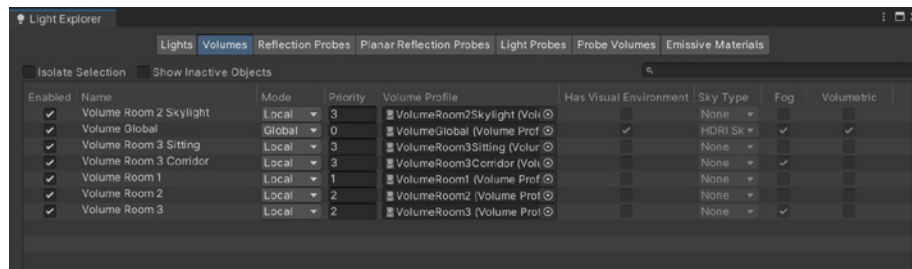
Volumes cover spaces with distinct lighting conditions.

A Volume is just a placeholder object with a Volume component. You can create one through the **GameObject > Volume** menu by selecting a preset. Otherwise, simply make a GameObject with the correct components manually.



Creating a Volume object using the presets

Because Volume components can be added to any GameObject, it can be difficult to find them via the Hierarchy. The Light Explorer (**Window > Rendering > Light Explorer > Volumes**) can help you locate the volumes in the loaded Scenes. Use this interface to make quick adjustments.

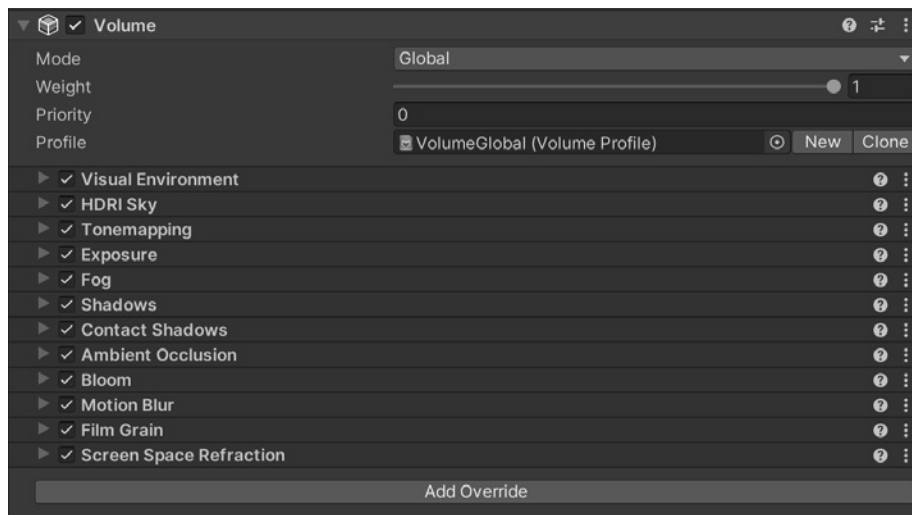


The Light Explorer can list all the Volumes in the open Scene(s).

Local and Global

Set the Volume component's **Mode** setting to either **Global** or **Local**, depending on context.

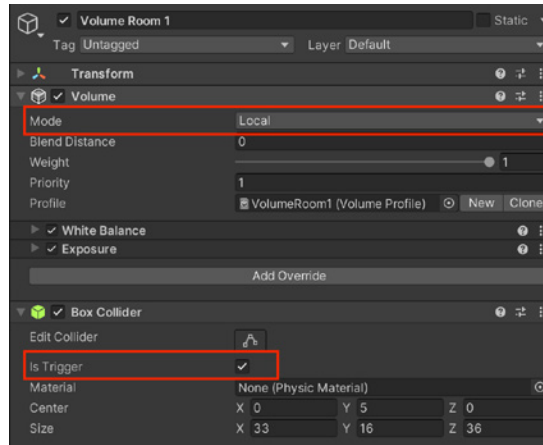
A global Volume works as a “catch-all” without any boundaries, and it affects all cameras in the Scene. In the HDRP template scene, the VolumeGlobal defines an overall baseline of HDRP settings for the entire level.



The Global Volume overrides

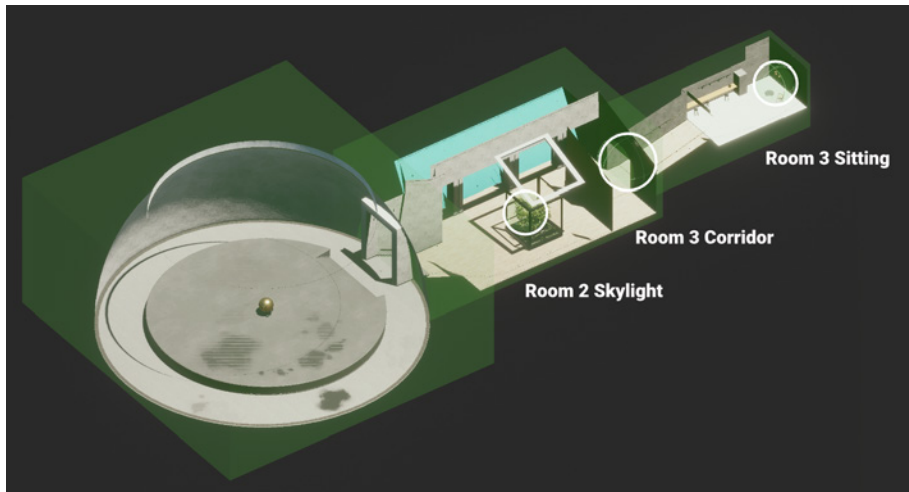
A local Volume defines a limited space where its settings take effect. It uses a Collider component to determine its boundaries. Enable **IsTrigger** if you don't want the Collider to impede the movement of any physics bodies like your FPS player controller.

In the template scene, each room has a local Volume with a BoxCollider that overrides the global settings.



Each room has a local Volume with a Collider set to IsTrigger.

Room 2 has a small, spherical Volume for the bright center next to the glass case. Likewise, Room 3 has smaller Volumes at its entrance corridor and at the seated area below the pendant lights.



Use smaller Volumes for special lighting conditions.

In the SampleScene, the local Volumes override White Balance, Exposure, and/or Fog. Anything not explicitly overridden falls back to the global defaults.

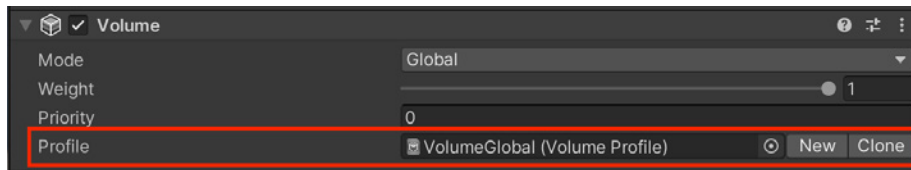
As your camera moves around the scene, the global settings take effect until your player controller bumps into a local Volume where those settings take over.

Performance tip

Don't use a large number of Volumes. Evaluating each Volume (blending, spatialization, override computation, and so on) comes with some CPU cost.

Volume Profiles

A Volume component itself contains no actual data. Instead, it references a [Volume Profile](#), a [ScriptableObject](#) Asset on disk that contains HDRP settings to render the scene. Use the Profile field to create a new Volume Profile with the **New** or **Clone** buttons.



Use the Profile field to switch Volume Profiles or create a new one.

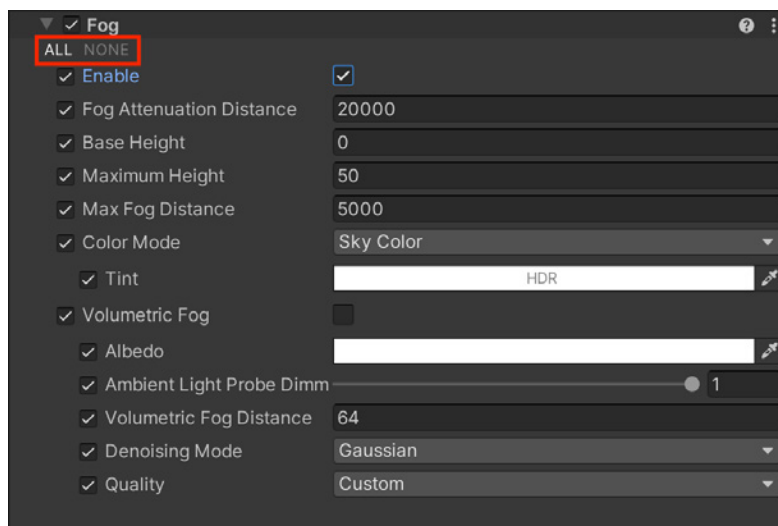
You can also switch to another Profile you already have saved. Having the Volume Profile as a file makes it easier to reuse previous settings and share Profiles between your Volumes.

Note that changes done to Volume Profiles in Play mode will not be lost when leaving said mode.

Volume Overrides

Each [Volume Profile](#) begins with a set of default properties. To edit their values, use [Volume Overrides](#) and customize the individual settings. For example, Volumes Overrides could modify the Volume's Fog, Post-processing, or Exposure.

Once you have your **Volume Profile** set, click Add **Override** to customize the Profile settings. A Fog override could look like this:



An example of Fog as a Volume Override

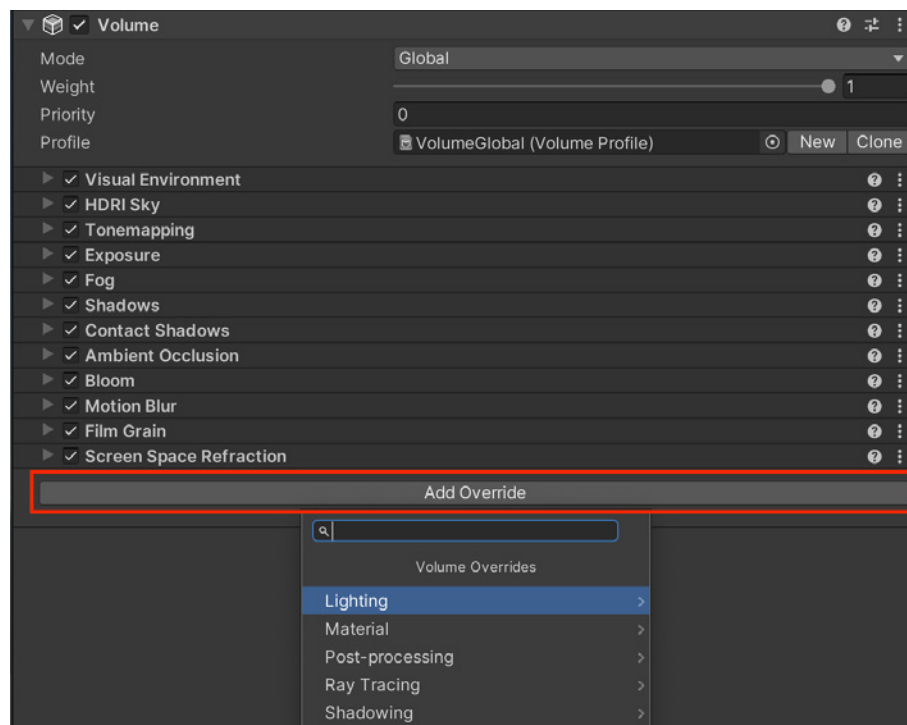
Each of the Volume Override's properties has a checkbox at the left, which you can enable to edit that property. Leaving the box disabled means HDRP uses the Volume's default value.

Each Volume object can have several overrides. Within each one, edit as many properties as needed. You can quickly check or uncheck all of them with the **All** or **None** shortcut at the top left.

Overrides workflow

Adding overrides is a key workflow in HDRP. If you understand the concept of [inheritance from programming](#), Volume Overrides will seem familiar to you.

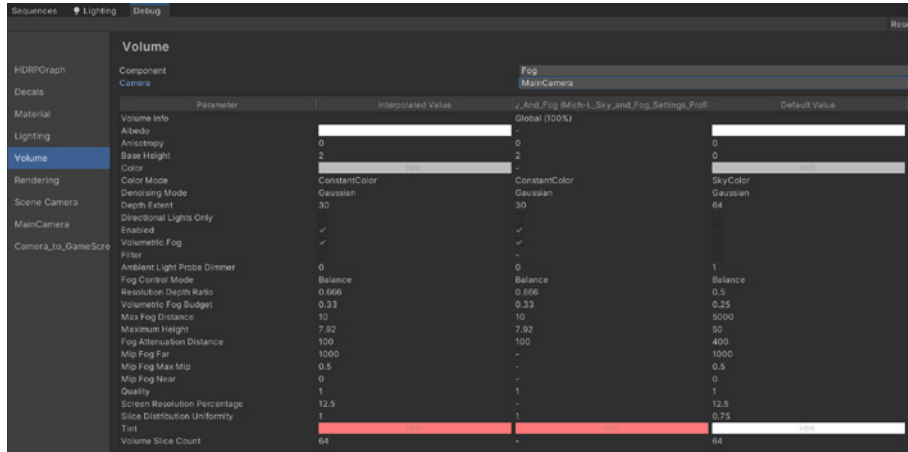
The higher-level Volume settings serve as the defaults for lower-level Volumes. Here, the HDRP Default Settings pass down to the global Volume. This, in turn, serves as the "base" for the local Volumes.



Adding HDRP features using Volume Overrides

The Global Volume overrides the HDRP Default Settings. The Local Volumes, in turn, override the Global Volume. Use the **Priority**, **Weight**, and **Blend Distance** (outlined below) to resolve any conflicts from overlapping Volumes.

To debug the current values of a given Volume component, you can use the Volume tab in the [Rendering Debugger](#).



Debugging a Volume

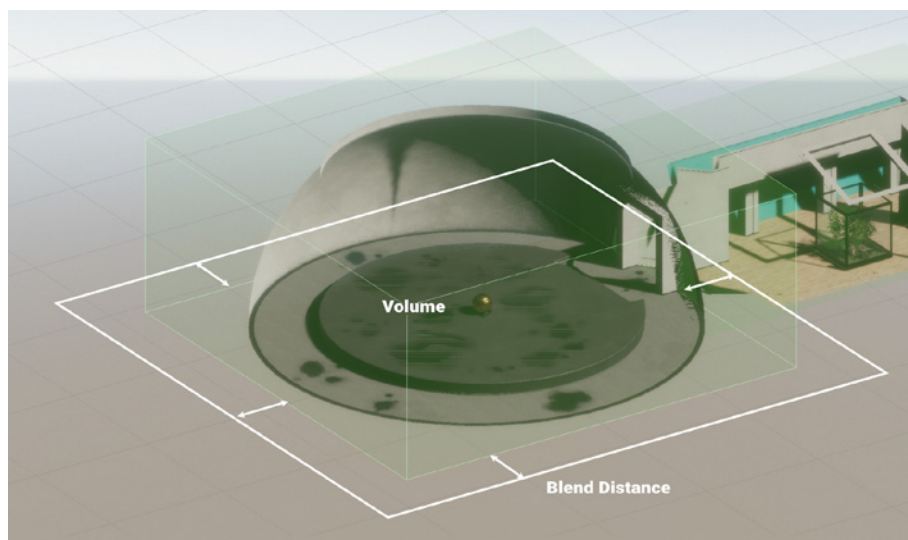
You can find a complete [Volume Overrides List](#) in the HDRP documentation.

Blending and priority

Because you often need more than one Volume per level, HDRP allows you to blend between Volumes. This makes transitions between them less abrupt.

At runtime, HDRP uses the camera position to determine which Volumes affect the final HDRP settings.

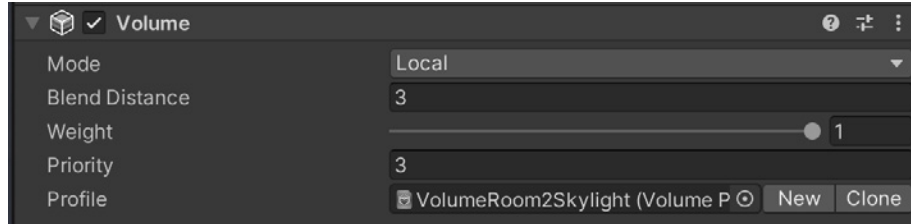
Blend Distance determines how far outside the Volume's Collider to begin fading on or off. A value of 0 for Blend Distance means an instant transition, while a positive value means the Volume Overrides begin blending once the camera enters the specified range.



Blend Distance defines a transition zone around the Volume.

The Volume framework is flexible and allows you to mix and match Volumes and overrides as you see fit. If more than one Volume overlaps the same space, HDRP relies on **Priority** to decide which Volume takes precedence. Higher values mean higher priority.

In general, set your Priority values explicitly to eliminate any guesswork. Otherwise, the system will use creation order as the Priority “tiebreaker,” which may lead to unexpected results.



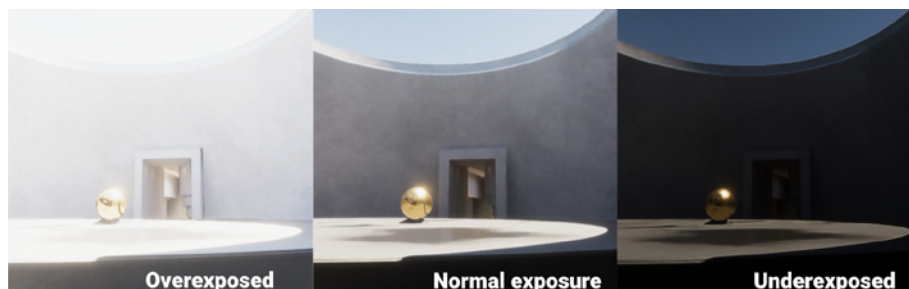
Use Blend Distance, Weight, and Priority when overlapping local Volumes.

Exposure

HDRP uses real-world lighting models to render each scene. As such, many properties are analogous to their counterparts in traditional photography.

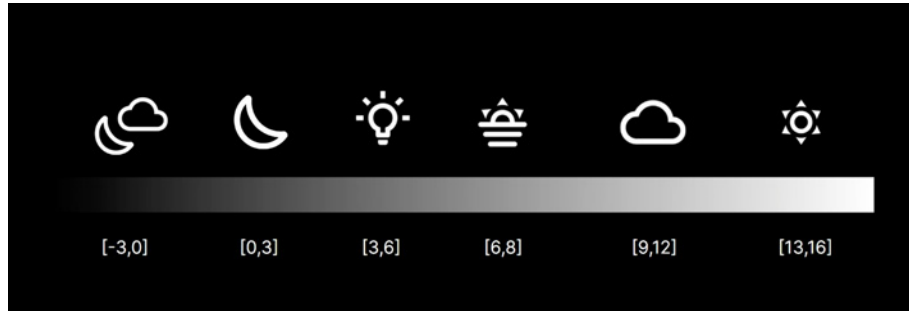
Understanding exposure value

Exposure value (EV) is a numeric value that represents a combination of a camera's [shutter speed](#) and [f-number](#) (which determines the size of the lens opening, or aperture). You need to properly set [exposure](#) to reach ideal brightness, capturing high levels of detail in both the shadows and highlights. Otherwise, overexposing or underexposing the image leads to less-than-desirable results.



Compare overexposed, underexposed, and balanced images

Your exposure range in HDRP will typically fall somewhere along this spectrum:



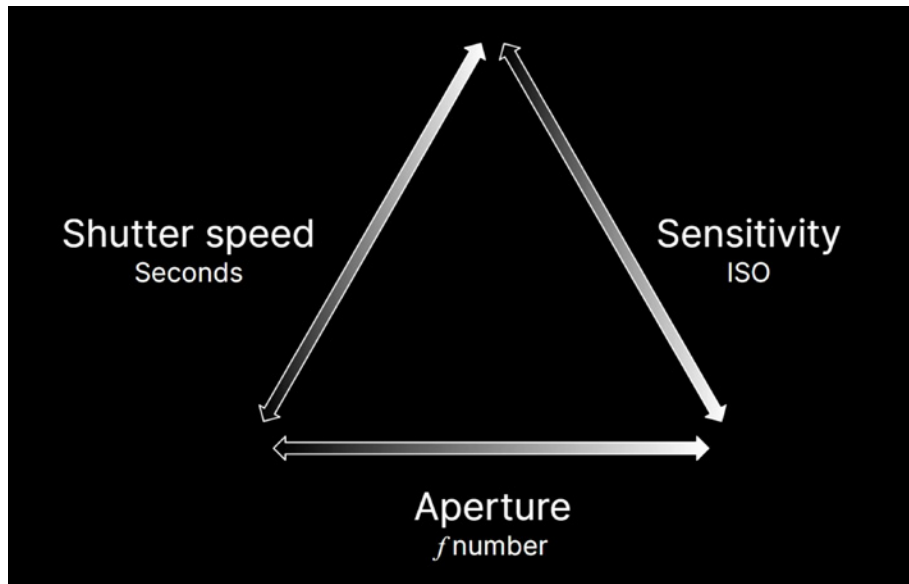
Exposure range, from a moonless night to a bright, sunny day

Greater exposure values allow less light into the camera and are appropriate for more brightly lit situations. Here, an EV value between 13 and 16 is suitable for a sunny daytime exterior. In contrast, a dark, moonless, night sky might use an EV between -3 and 0.

You can vary a number of factors in an actual camera's settings to modify your Exposure value:

- The shutter speed, the length of time the image sensor is exposed to light
- The f-number, or the size of the aperture/lens opening
- The ISO, or sensitivity of the film/sensor

Photographers call this the exposure triangle. In Unity, as with a real camera, you can arrive at the same exposure value using different combinations of these numbers.



Exposure triangle

HDRP expresses all exposure values in EV_{100} , which fixes the sensitivity to that of [100 International Standards Organization \(ISO\) film](#).

i Exposure value formula

This formula actually calculates exposure value.

$$EV = \log_2 \left(\frac{f \text{ number}^2 / \text{shutter speed}}{ISO / 100} \right)$$

Exposure formula

It's a logarithmic base-2 scale. As the exposure value increases 1 unit, the amount of light entering the lens decreases by half.

HDRP allows you to match the exposure of a real image. Simply shoot a digital photo with a camera or smartphone. Grab the metadata from the image to identify the f-number, shutter speed, and ISO.



Use the digital photo's Exif data to match exposure.

Then, calculate the exposure value using the formula above. If you use the same value in the Exposure override (see below), the rendered image should fall in line with the real-world image exposure.

In this way, you can use digital photos as references when lighting your level. While your goal isn't necessarily to recreate the image perfectly, matching an actual photograph can take the guesswork out of your lighting setups.

Exposure override

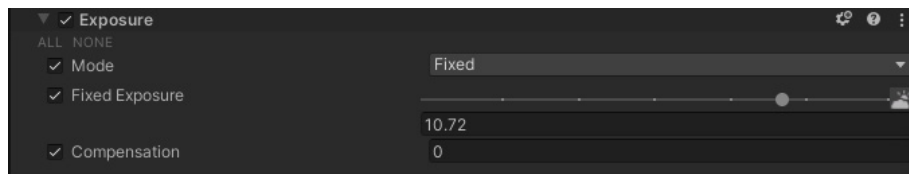
In HDRP, Exposure is a Volume Override. Add it to a local or global Volume to see the available properties.

In the **Mode** dropdown, you can select one of the following: **Fixed**, **Automatic**, **Automatic Histogram**, **Curve Mapping**, and **Physical Camera**.

Compensation allows you to shift or adjust the exposure. You would typically use this to apply minor adjustments and “stop” the rendered image up and down slightly.

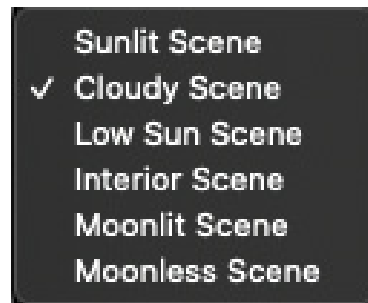
Fixed mode

Fixed mode lets you set the exposure value manually.



Fixed mode exposure

Follow the graduation marks on the **Fixed Exposure** slider for hints. The icon to the right also has a dropdown of Presets (e.g., 13 for a Sunlit Scene down to -2.5 for a Moonless Scene). You can also set the field directly to any value.



Fixed exposure presets

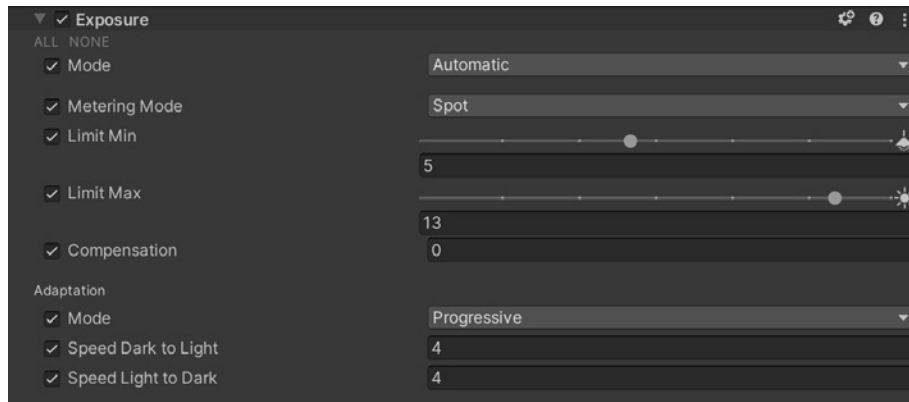
Fixed mode is simple but not very flexible. It usually only works if you have a Volume or Scene with relatively uniform lighting, where one exposure value can work throughout.

Automatic mode

Automatic mode dynamically sets the exposure depending on the range of brightness levels onscreen. This functions much like the [human eye adapts to varying levels of darkness](#), redefining what is perceived as black.

While Automatic mode will work under many lighting situations, it can also unintentionally overexpose or underexpose the image when pointing the camera at a very dark or very bright part of the scene.

Use the **Limit Min** and **Limit Max** to keep the exposure level within a desirable range. Playtest to verify that your limits stay within your expected exposure throughout the level.



Automatic mode exposure

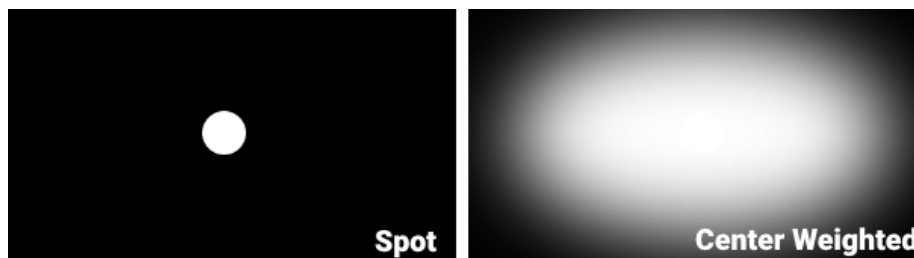
Metering Mode, combined with mask options, determines what part of the frame to use for autoexposure.

The **Adaptation mode** controls how the autoexposure changes as the camera transitions between darkness and light, with options to adjust the speed. Just like with the eye, moving the camera from a very dark to a very light area, or vice versa, can be briefly disorienting.

Metering mode options

Automatic, Automatic Histogram, and Curve Mapping modes use Metering mode to control what part of the frame to use when calculating exposure. You can set the Metering mode to:

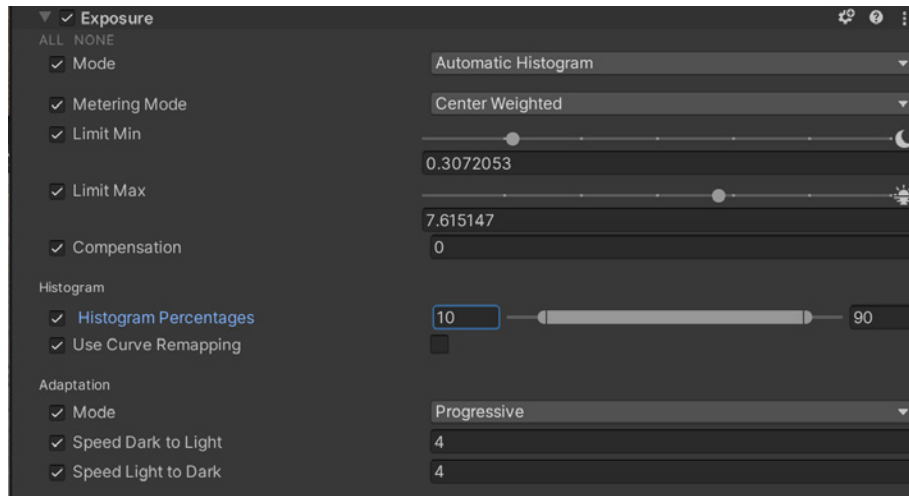
- **Average:** The camera uses the entire frame to measure exposure.
- **Spot:** The camera only uses the center of the screen to measure exposure.
- **Center Weighted:** The camera favors pixels in the center of the image and feathers out toward the edges of frame.
- **Mask Weighted:** A supplied image (Weight Texture Mask) determines which pixels are most important when determining exposure.
- **Procedural Mask:** The camera evaluates exposure based on a procedurally generated texture. You can change options for the center, radius, and softness.



Spot and Center Weighted metering modes

Automatic Histogram

Automatic Histogram mode takes Automatic mode a step further. This computes a [histogram](#) for the image and ignores the darkest and lightest pixels when setting exposure.

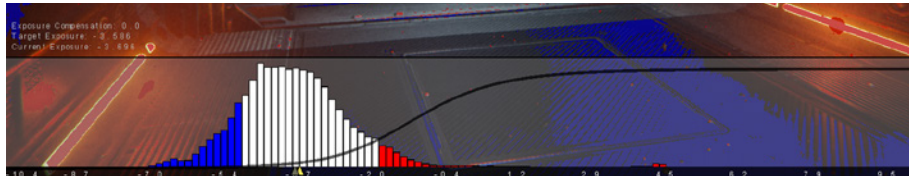


Automatic Histogram mode

By rejecting very dark or very bright pixels from the exposure calculation, you may experience a more stable exposure whenever extremely bright or dark pixels appear on the frame. This way, intense emissive surfaces or black materials won't underexpose or overexpose your rendered output as severely.

The **Histogram Percentages** setting allows you to discard anything in the histogram outside the given range of percentages (imagine clipping the brightest and darkest pixels from the histogram's leftmost and rightmost parts).

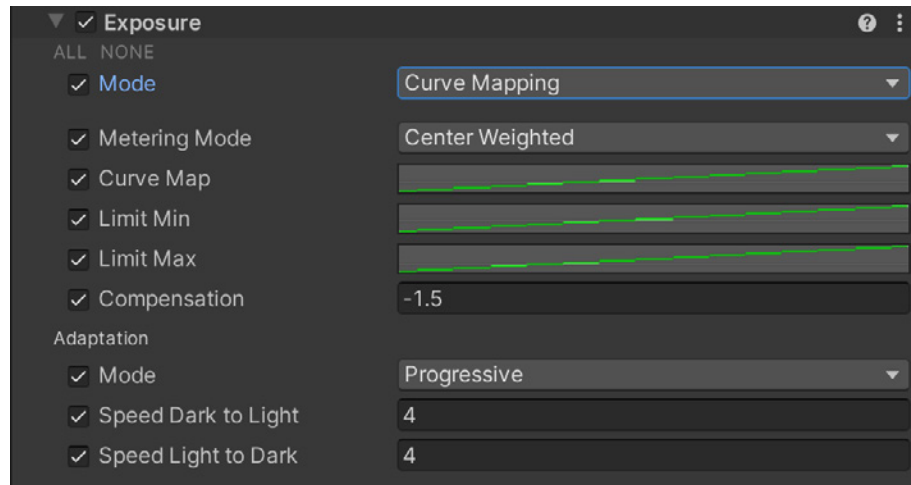
Curve Remapping lets you remap the exposure curve as well (see Curve Mapping below).



Automatic Histogram mode uses pixels from the middle of the histogram to calculate exposure.

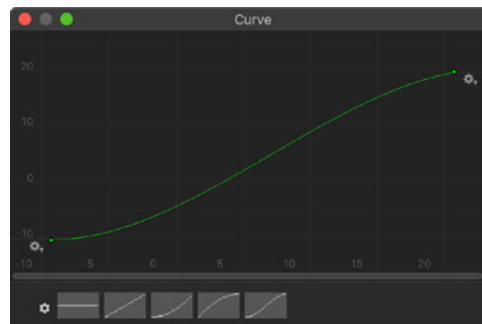
Curve Mapping

The Curve Mapping mode is another variant of [Automatic](#) mode.



Curve Mapping mode

Here, the x-axis of the curve represents the current exposure, and the y-axis represents the target exposure. Remapping the exposure curve can generate very precise results.

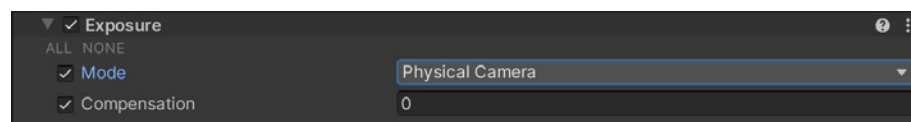


Adjust the exposure using a curve.

Physical Camera

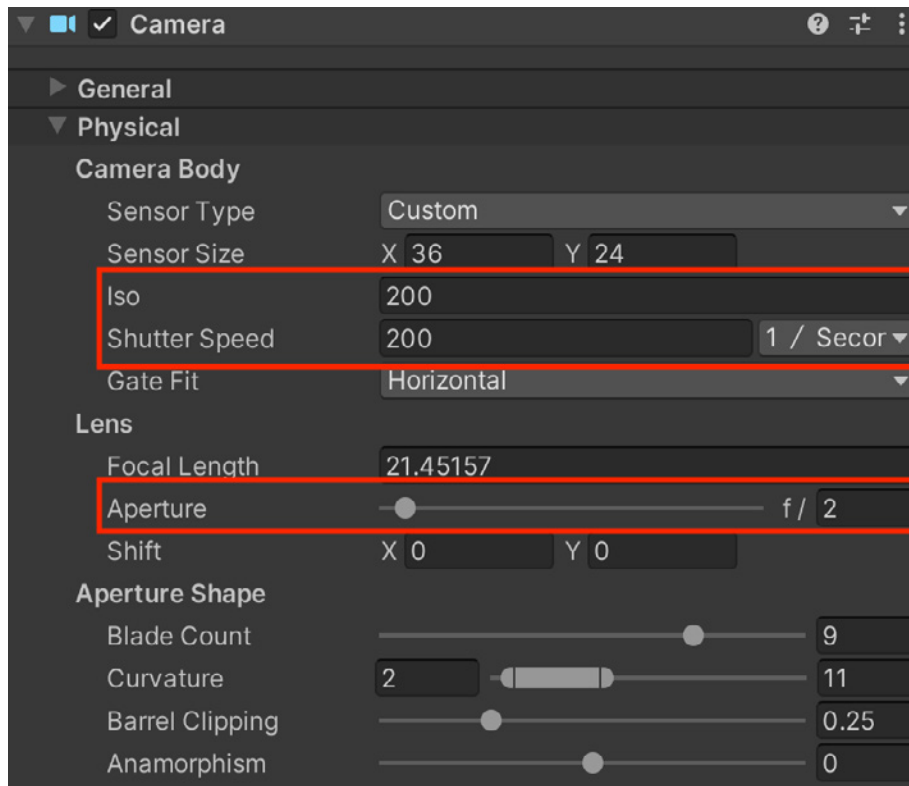
Those familiar with photography may find [Physical Camera](#) mode helpful for setting camera parameters.

Switch the Exposure override's **Mode** to **Physical Camera**, then locate the Main Camera.



Physical Camera mode

Enable **Physical Camera**. The Inspector shows the following properties.



Physical Camera properties on a camera

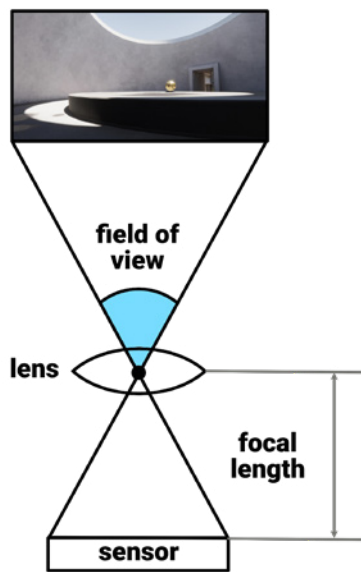
Important to exposure are the **ISO** (sensitivity), **Aperture** (or f-number), and **Shutter Speed**. If you are matching reference photos, copy the correct settings from the image's [Exif](#) data. Otherwise, [this table](#) can help you guesstimate Exposure Value based on f-number and shutter speed.

Additional Physical Camera parameters

Though not related to exposure, other [Physical Camera](#) properties can help you match the attributes of real-world cameras.

For example, we normally use **Field of View** in Unity (and many other 3D applications) to determine how much of the world a camera can see at once.

In real cameras, however, the field of view depends on the size of the sensor and focal length of the lens. Rather than setting the field of view directly, the Physical Camera settings allow you to fill in the **Sensor Type**, **Sensor Size**, and **Focal Length** from the actual camera data. Unity will then automatically calculate the corresponding Field of View value.



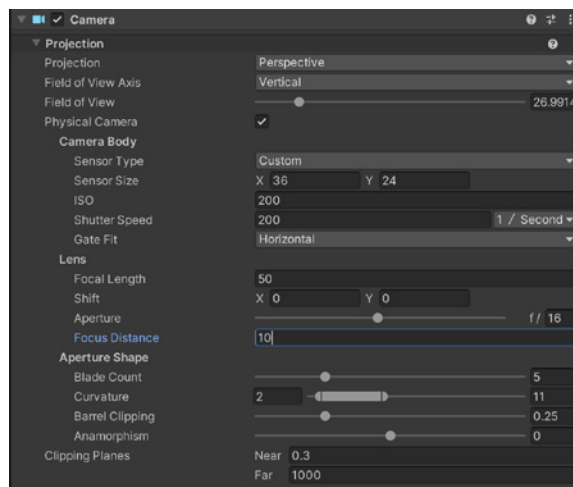
The relationship between focal length, sensor size, and field of view

Rely on the camera metadata included with the image files when trying to match a real photo reference. Both Windows and macOS can read the Exif data from digital images. You can then copy the corresponding fields to your virtual camera.

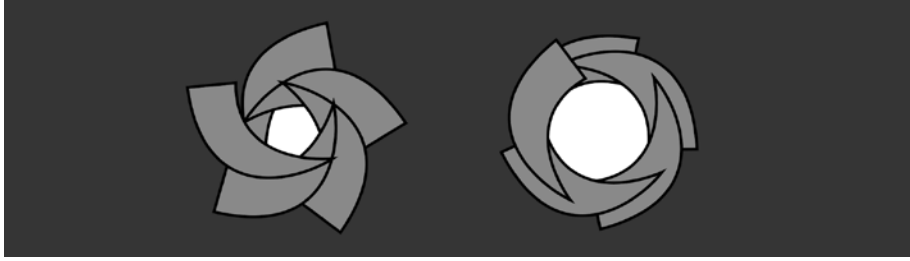
Note: You may need to search for the exact sensor dimensions on the manufacturer's website once you have the camera make and model from the metadata. [This article](#) includes an estimate of common image sensor formats.

Several of the bottom parameters influence the Depth of Field Volume.

In Unity 2021 LTS and up, you can control the **Focus Distance** from the Camera's Inspector. In the Depth of Field Volume component, set the **Focus Mode** and the **Focus Distance Mode** to **Physical Camera**.



Blade Count, Curvature, and Barrel Clipping change the camera aperture's shape. This influences the look of the bokeh that results from the [Depth of Field Volume](#) component.



Use the physical camera parameters to reshape the aperture. The five-bladed iris can show a pentagonal (left) or circular (right) bokeh.

Lights

HDRP includes a number of different Light types and shapes to help you control illumination in your scene.

Light types

These Light types are available, similar to the other render pipelines in Unity:

- **Directional:** This behaves like light from an infinitely distant source, with perfectly parallel light rays that don't diminish in intensity. Directional lights often stand in for sunlight. In an exterior scene, this will often be your key light.
- **Spot:** This is similar to a real-world spotlight, which can take the shape of a cone, pyramid, or box. A spot falls off along the forward z-axis, as well as toward the edges of the cone/pyramid shape.
- **Point:** This is an omnidirectional light that illuminates all directions from a single point in space. This is useful for radiant sources of light, like a lamp or candle.
- **Area:** This projects light from the surface of a specific shape (a rectangle, tube, or disc). An area light functions like a broad light source with a uniform intensity in the center, like a window or fluorescent tube.

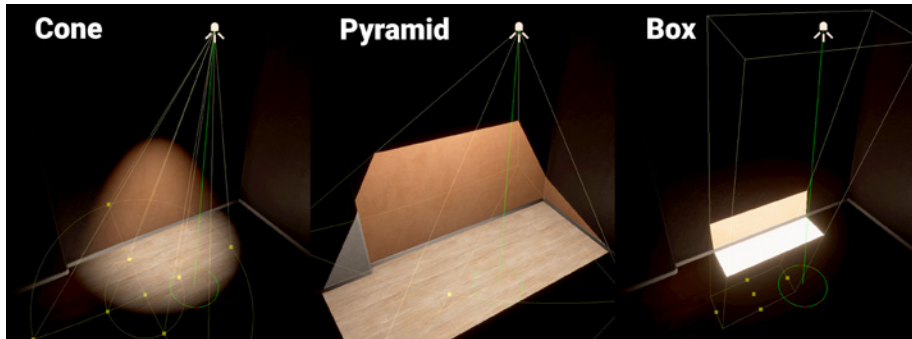
Modify how the spot, point, and area lights fall off with the **Range**. Many HDRP Lights diminish using the [inverse square law](#), like light sources in the real world.

Shapes

Spot and area lights have additional shapes for controlling how each light falls off.

HDRP spot lights can use three shapes:

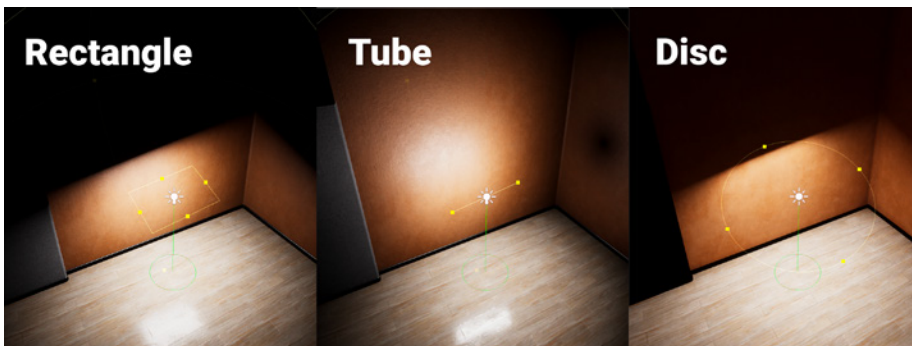
- **Cone:** Projects light from a single point to a circular base. Adjust the **Outer Angle** (degrees) and **Inner Angle** (percentage) to shape the cone and modify its angular attenuation.
- **Pyramid:** Projects light from a single point onto a square base. Adjust the pyramid shape with the **Spot Angle** and **Aspect Ratio**.
- **Box:** Projects light uniformly across a rectangular volume. An X and Y size determine the base rectangle, and the Range controls the Z dimension. This light has no attenuation unless **Range Attenuation** is checked and can be used to simulate sunlight within the boundary of the box.



HDRP spot light shapes

HDRP area lights can use three shapes:

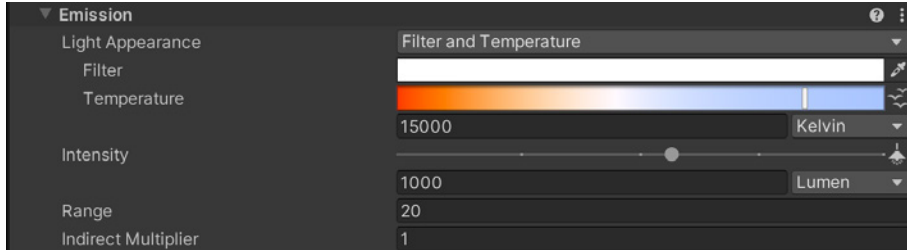
- **Rectangle:** Projects light from a rectangle shape in the local, positive Z direction out to a defined Range.
- **Tube:** Projects light from a single line in every direction, out to a defined Range. This light only works in Realtime Mode.
- **Disc:** Projects light from a disc shape in the local positive Z direction, out to a defined Range. This light only works in Baked Mode.



HDRP area light shapes

Color and temperature

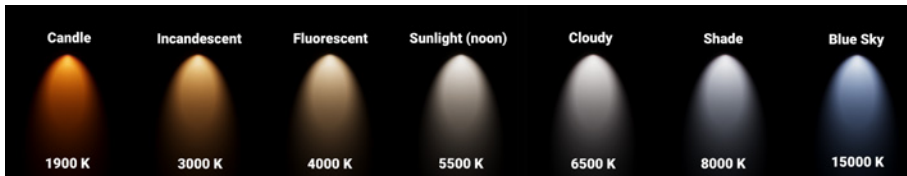
All HDRP Light types have [Emission](#) properties that define the light's appearance.



Modify Light Appearance properties under Emission

You can switch the **Light Appearance** to **Color** and specify an RGB color. Otherwise, change this to **Filter and Temperature** for more physically accurate input.

Color temperature sets the color based on [degrees Kelvin](#). See the [Lighting and Exposure Cheat Sheet](#) for reference.



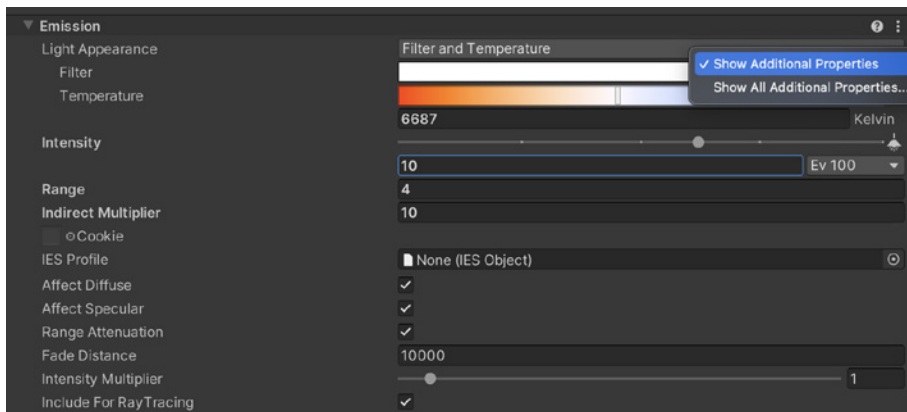
Color temperature on the Kelvin scale

You can also add another color that acts like a **Filter**, tinting the light with another hue. This is similar to adding a [color gel](#) in photography.

Additional properties

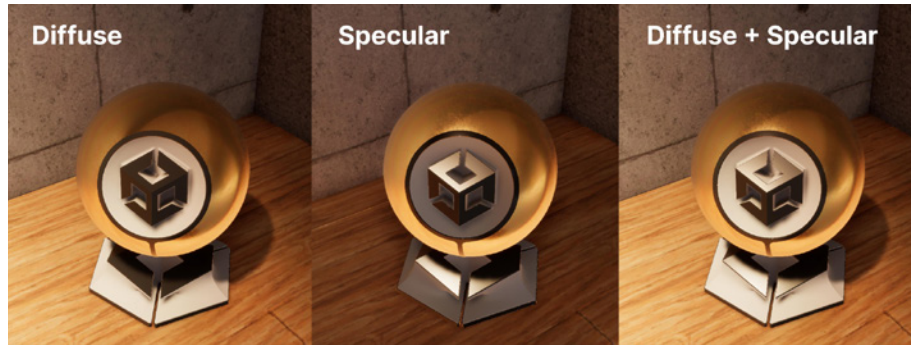
HDRP also includes some advanced controls under the **More Items menu** (:) at the top right of the Inspector properties. Select **Show Additional Properties** to see extra options.

These include toggles for **Affect Diffuse** and **Affect Specular**. In cutscene or cinematic lighting, for example, you can separate Lights that control the bright shiny highlights independently from those that produce softer diffuse light.



Each Light has additional properties.

You can also use the **Intensity Multiplier** to adjust the overall intensity of the light without actually changing the original intensity value. This is useful for brightening or darkening multiple Lights at once.



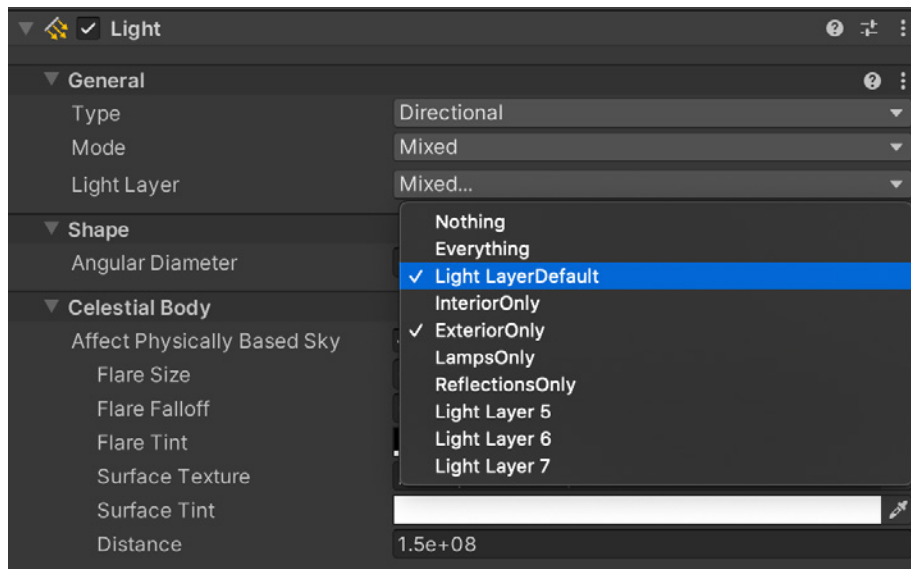
Toggle diffuse and specular lighting for additional control.

Light Layers

HDRP allows you to use **Light Layers** to make Lights only affect specific meshes in your Scene. These are LayerMasks that you can associate with a Light component and MeshRenderer.

In the Light properties, click the **More Options** button. This displays the **Light Layer** dropdown under **General**. Choose which LayerMasks you want to associate with the Light.

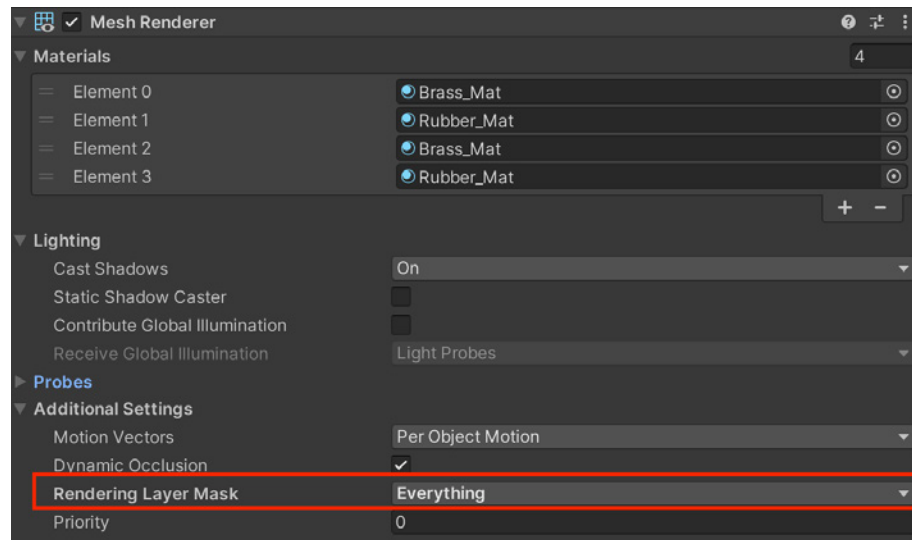
In the Light properties, select **Show Additional Properties** from the **More Items** menu (:). This displays the **Light Layer dropdown** under **General**. Choose what LayerMasks you want to associate with the Light.



Select a Light Layer

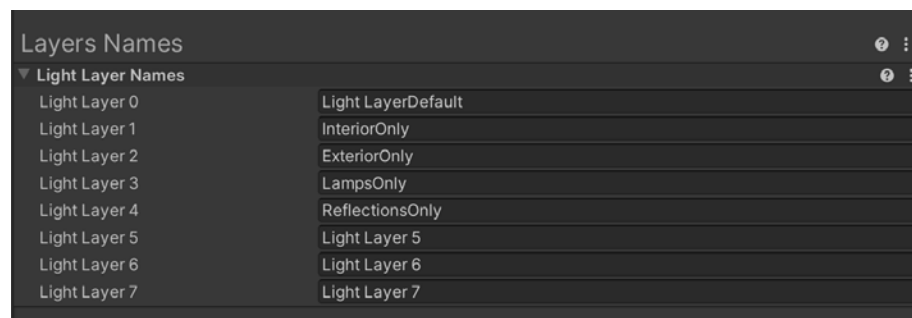
Next, set up the MeshRenderers with the **Rendering Layer Mask**. Only Lights on the matching LayerMask will affect the mesh. This feature can be invaluable for fixing light leaks, making sure that lights only strike their intended targets. It can also be part of the workflow to set up cutscene lighting, so that characters only can receive dedicated cinematic lights.

For example, if you wanted to prevent the lights inside of a building from accidentally penetrating the walls to the outside, you could set up specific Light Layers for the interior and exterior. This ensures that you have fine-level control of your Light setups.



Set the Rendering Layer Mask so that only specific Lights affect the mesh.

To set up your Light Layers, go to the **HDRP Default Settings**. The **Layers Names** section lets you set the string name for **Light Layer 0** to **7**.



Layers Names in the HDRP Default Settings

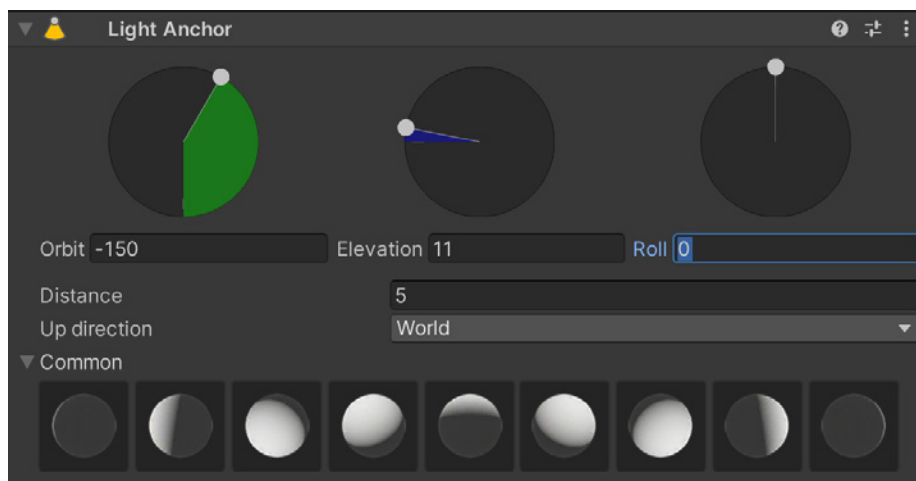
For more information, including the full list of Light properties, see the [Light component documentation](#).

Light Anchors

Unity 2021 LTS and above includes a **Light Anchor** system to help you set up lights quickly, by controlling the angle and distance between the camera and subject. It also enables you to select common lighting angles via nine presets.

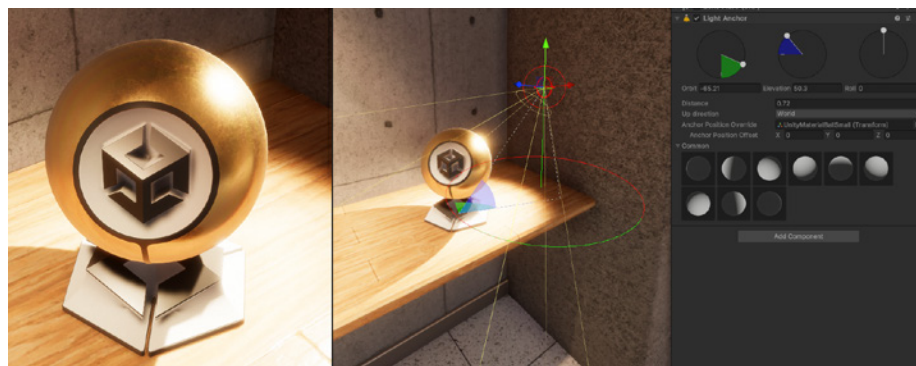
If you need to light a scene, a product, or a shot in a cinematic using multiple lights around characters or props, the Light Anchor component offers fast light manipulation in screen space around an anchor target.

First, make sure your Camera is tagged as MainCamera for the Light Anchor to work. Then, add a **Light Anchor** component on the **Spot** Light you want to control. Align the light on the subject; its position is now the anchor point of the Spot Light. Increase the **Distance** between the anchor point and the Spot Light. You can now adjust the position of the light around the anchor point by tuning the **Orbit**, **Elevation**, and **Roll** of the light within the Game View, rather than having to manually adjust the Transform of the light in the Scene view.



The Light Anchor component

For more information, see this presentation introducing [Light Anchors](#).



Aim your lights with a Light Anchor component instead of changing their transforms directly.

Physical Light Units and intensity

HDRP uses Physical Light Units (PLU) for measuring light intensity. These match real-life [SI](#) measurements for illuminance, including candela, lumen, lux, and nits. Note that PLU expects 1 unit in Unity to equal 1 meter for accuracy.

Units

Physical Light Units can include units of both *luminous flux* and *illuminance*. Luminous flux represents the total amount of light *emitted* from a source, while illuminance refers to the total amount of light *received* by an object (often in luminous flux per unit of area).

Because commercial lighting and photography may express units differently depending on application, Unity supports multiple Physical Light Units for compatibility:

- **Candela:** One unit is equivalent to the luminous flux of a wax candle. This is also commonly called *candlepower*.
- **Lumen:** This is the SI unit of luminous flux defined to be the 1 candela over a solid angle (steradian). You will commonly see lumens on commercial lightbulb specifications. Use it with Unity spot, point, or area lights.
- **Lux:** A light source that emits 1 lumen onto an area of 1 square meter has an illuminance of 1 lux. Real-world light meters commonly read lux, and you will often use this unit with directional lights in Unity.
- **Nits:** This is a unit of luminance that is the equivalent of 1 candela per square meter. Display devices and LED panels (televisions or monitors, for example) often measure their brightness in Nits.

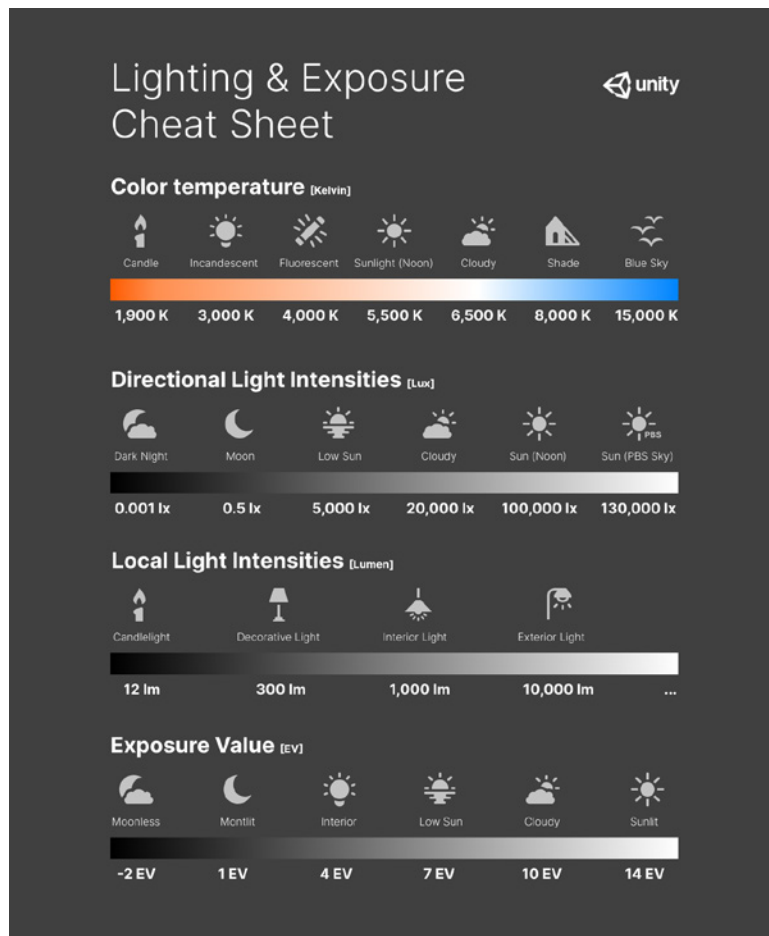
- **EV₁₀₀**: This uses an intensity corresponding to **EV₁₀₀**, which is an exposure value with 100 ISO film (see exposure value formula above). Incrementing the exposure results in the doubling of the lighting, due to the logarithmic behavior.

For recreating a real lighting source, switch to the unit listed on the tech specs and plug in the correct luminous flux or luminance. HDRP will match the Physical Lighting Units, eliminating much of the guesswork when setting intensities.

Click the icon to choose presets for **Exterior**, **Interior**, **Decorative**, and **Candle**. These settings provide a good starting point if you are not explicitly matching a specific value.

Common lighting and exposure values

The following cheat sheet contains the color temperature values and light intensities of common real-world [light](#) sources. It also contains [exposure](#) values for different lighting scenarios.

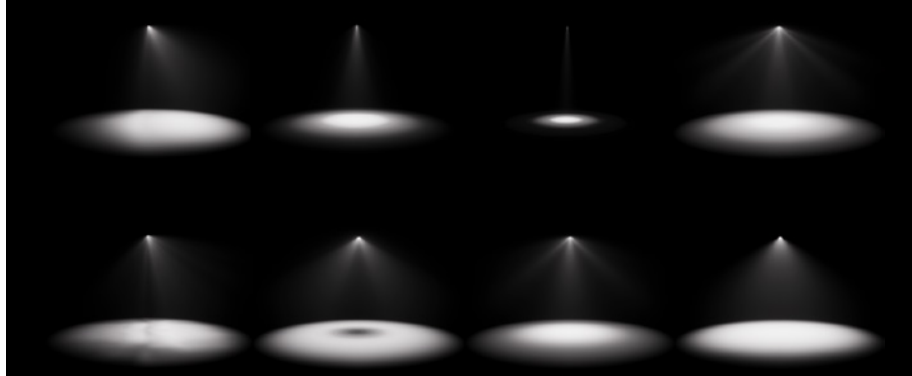


Guidance for lighting and exposure levels

You can find a complete table of common illumination values in the Physical Light Units [documentation](#).

IES Profiles and Cookies

Make your point, spot, and area lights more closely mimic the falloff of real lights using an **IES profile**. This works like a **light cookie** to apply a specific manufacturer's specs to a pattern of light. IES profiles can give your lights an extra boost of realism.



IES profiles applied to various lights

Import an IES profile from **Assets > Import New Asset**. The **importer** will automatically create a **Light Prefab** with the correct intensity. Then just drag the Prefab into the Scene view or Hierarchy and tweak its color temperature.

Here are some good sources for IES profiles:

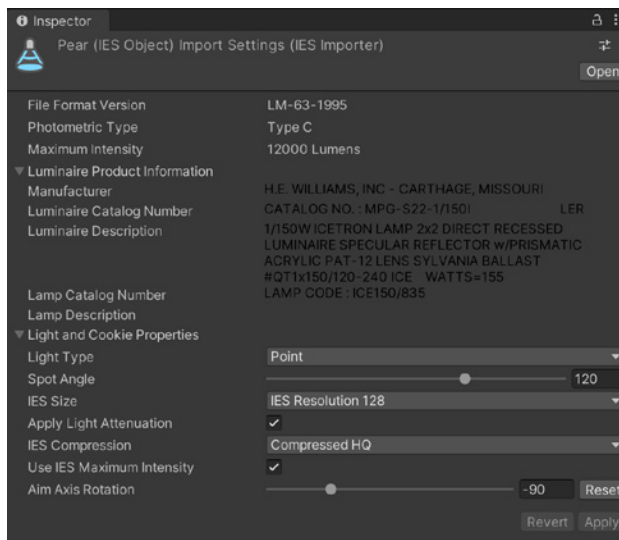
Real-world manufacturers

- Philips
- Lithonia Lighting
- Efficient Lighting Systems
- Atlas
- Erco
- Lamp
- Osram

Artist sources

- Renderman

For information on the IES profile importer, see the [documentation](#).



IES Profile and Import Settings

HDRP global illumination

Understanding global illumination

When light hits a surface in the real world, it doesn't just stop; it bounces, refracts, and scatters. Nothing short of a black hole can capture those amazing photons as they disperse through the environment. Thus, even the dullest materials – like concrete, sand, or stone – reflect light.

The human eye is attuned to these subtle behaviors of light. We recognize a rendering as “realistic” when it accurately depicts how light interacts with a complex variety of materials.

That's what global illumination (GI) tries to imitate. Without GI, areas outside of direct light are dark by default. GI attempts to approximate diffuse reflections; like in the real world, colored light transfers from one surface to another. This bounced, indirect lighting helps ground your game world in reality.

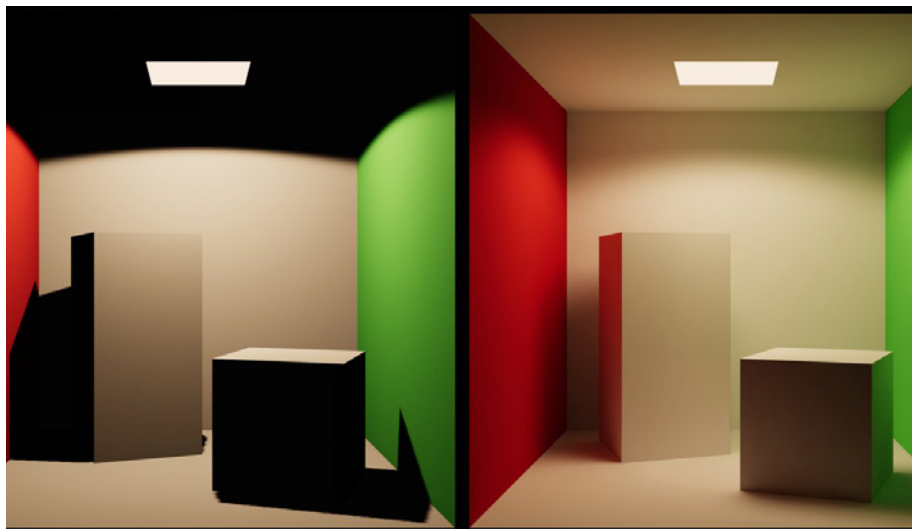


Global illumination can produce realistic results. Source: ArchVizPro Vol. 10.

HDRP global illumination features

GI isn't a single technique but rather a set of algorithms to help account for this phenomenon. Unity includes a whole ecosystem of tools that recreate how light behaves in a physical environment. You'll often use a combination of these techniques to produce realistic renders:

- **Baked GI (baked lightmaps):** Precomputes indirect light for static objects, storing the results for runtime use
- **Precomputed real-time GI:** Calculates indirect light bounces in advance but applies final lighting at runtime, allowing for dynamic changes to lighting (e.g., day-night cycle)



Global illumination (right) simulates diffuse reflections or indirect lighting.

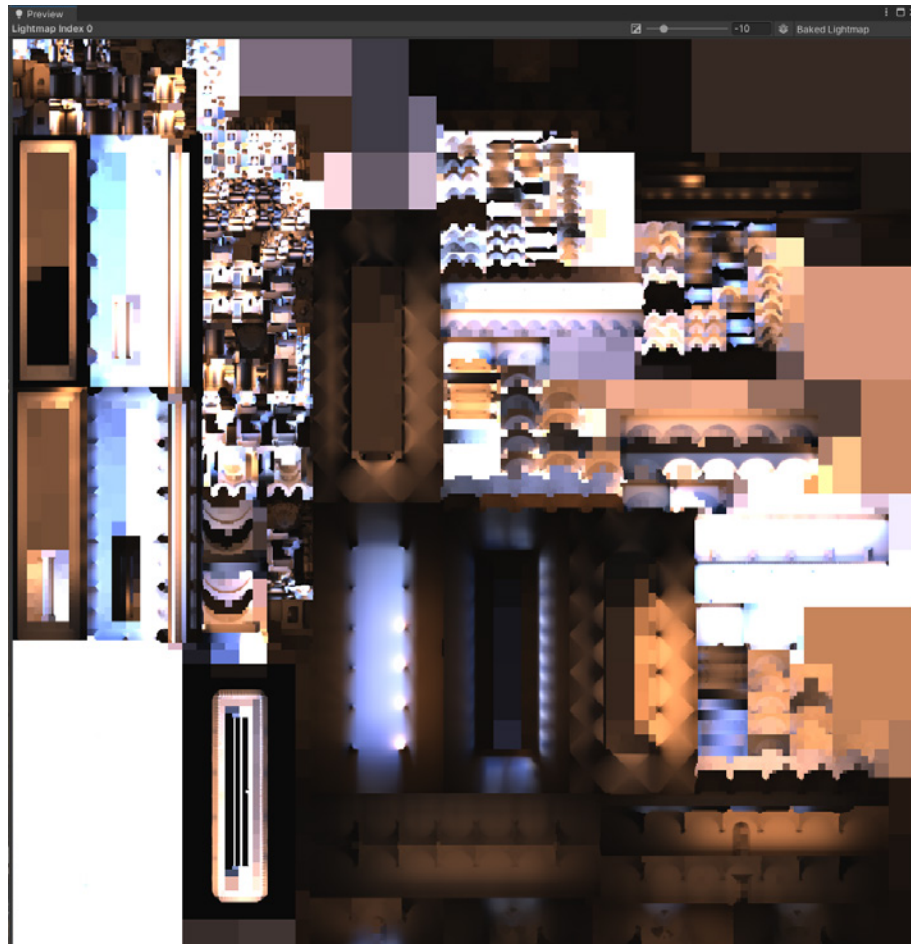
- **Light probes:** Small points in the scene that capture and interpolate baked lighting information, usually for moving objects; this enables dynamic objects to blend seamlessly with the precomputed lighting of the static environment
- **Environment lighting:** Uses high dynamic range images (e.g., HDRI skybox) to simulate environment lighting using image-based techniques, helping to provide realistic ambient light
- **Real-time ray tracing:** Produces photorealistic interactions of light with materials, capturing detailed reflections and shadows
- **Path tracing:** Recreates rays from the camera to allow HDRP to compute various effects (like shadows, reflections, refractions, and indirect illumination) in a unified process

HDRP's global illumination can support both static and dynamic environments. This can help make your game worlds feel alive and responsive.

Watch this video [“4 techniques to light environments in Unity”](#) for more tips.

Baked global illumination

Baked global illumination (baked GI) precomputes light interactions within a scene and stores the results as textures called lightmaps. This method captures how light reflects and refracts off surfaces but without trying to calculate this at runtime.

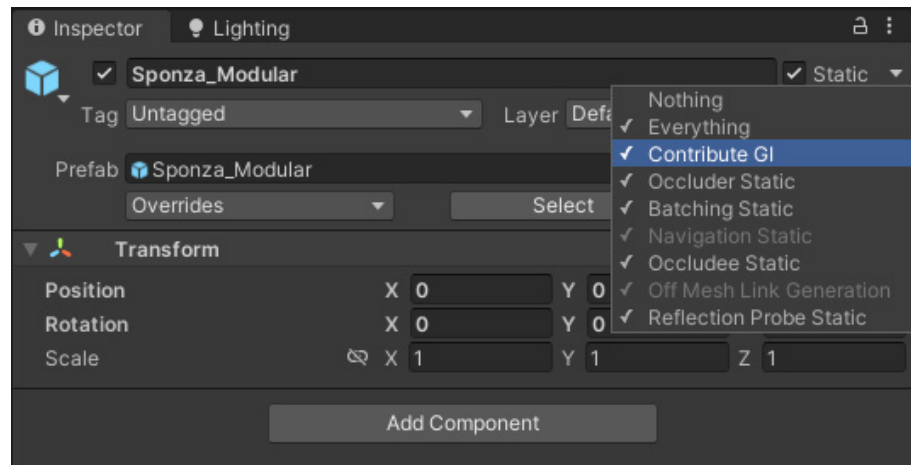


A baked lightmap

When developing for mobile platforms, this is often a common strategy for adding realistic lighting to the game environment. When baking lighting, the intensive computations are performed offline, just once.

This means there are no additional performance costs associated with these lighting calculations at runtime.

Baked GI offers several distinct advantages, starting with performance; precalculating the light computations drastically reduces demands during runtime. This process can capture intricate nuances like soft shadows and subtle diffuse reflections. It also eliminates many visual artifacts commonly associated with real-time lighting calculations.



Lightmapping only works for static objects.

However, prebaked lighting captures the global illumination only for static elements. Dynamic or moving elements in a scene won't fully benefit from the precomputed lighting data.

Note that lightmaps require additional memory and disk space, much like texture assets. If you're working on platforms with limited memory resources (e.g., mobile), be aware of this requirement.

Finally, baking GI can be quite time-intensive, depending on the scene complexity and your specific hardware. Very large environments with lots of static geometry can take minutes, or even hours, to lightmap.

Lightmapping workflow

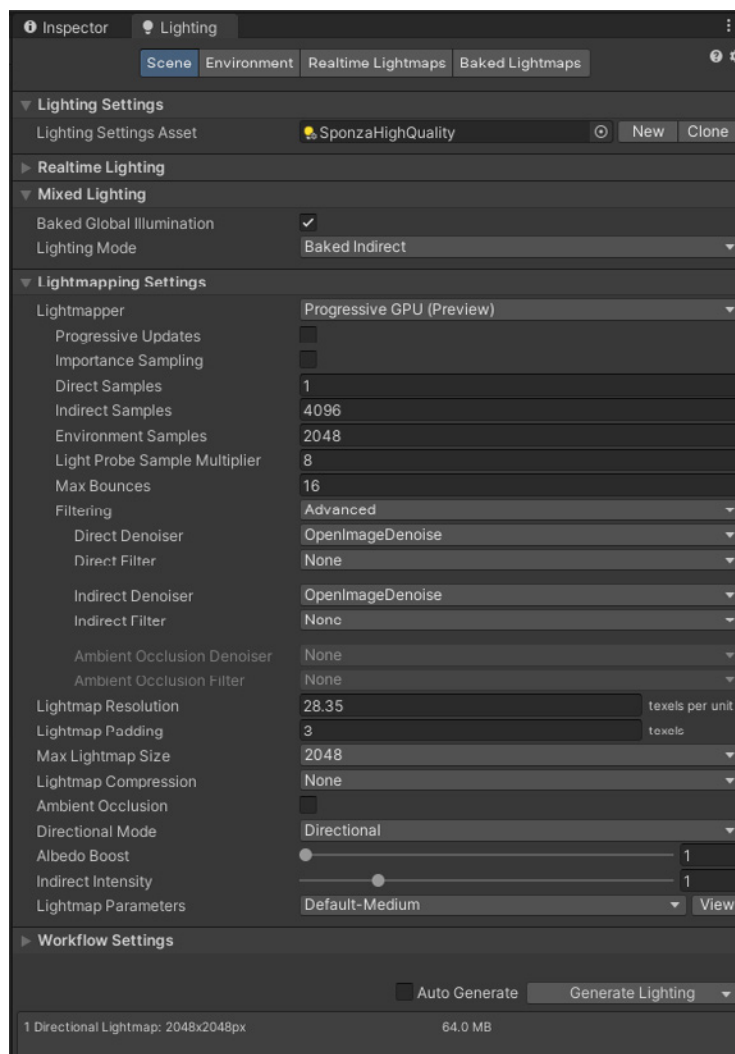
When lightmapping a scene with baked GI, follow these steps:

1. **Set light Modes:** Each light should be set to either **Baked** or **Mixed** to be included in the baking process.
2. **Mark objects as lightmap static:** Additionally, any objects that are intended to be part of the baking process should be flagged as **Contribute GI** or **Static Everything** from the [Static dropdown](#) menu.

3. **Create a new Lighting Settings Asset:** Use the Lighting window to generate a new Lighting Settings Asset, which represents a saved instance of the [LightingSettings](#) class. This stores precomputed lighting data for the baked GI (and Enlighten Realtime GI, below).
4. **Choose a Lighting Mode:** In the Lighting Settings Asset, select **Bake Indirect**, **Shadowmask**, or **Subtractive**. See [this guide](#) to determining the [Lighting Mode](#).
5. **In the Lightmap Settings, adjust basic parameters:** Use Lightmap Resolution and Sample Count to influence the accuracy and quality of the bake. Higher resolution and more samples typically yield a higher-quality result but can increase the baking time and consume more resources.
6. Click **Generate Lighting** to start lightmapping.

Baked lightmapping uses the [Progressive Lightmapper](#), which progressively refines the lightmaps as it calculates.

Preview the baking process as it happens. Interrupt the bake, adjust the settings, and rebake as necessary. This iterative process makes for a more interactive lighting workflow. The results appear in the **Baked Lightmaps** tab of the Lighting window.



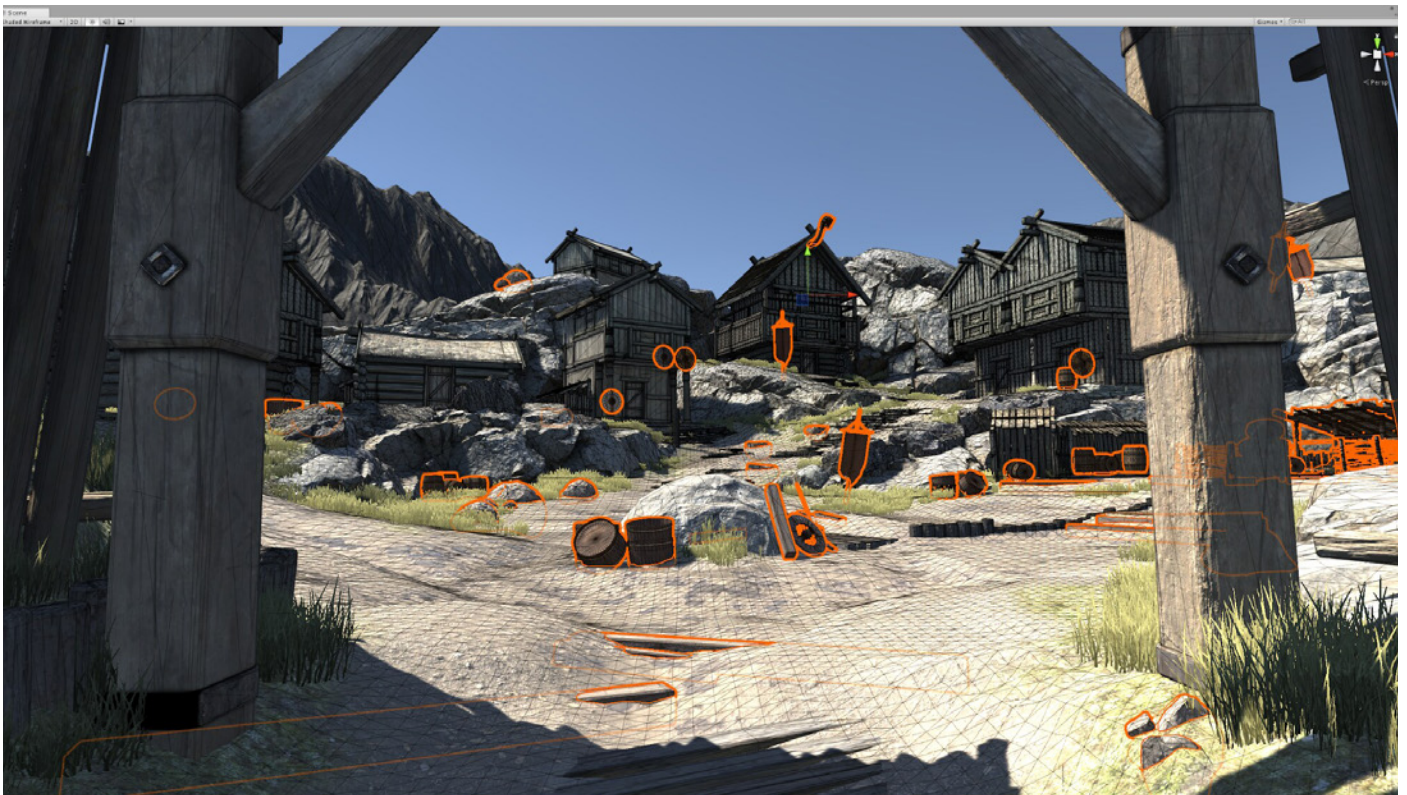
Enable the Progressive Lightmapper

Optimizing lightmaps

Optimizing baked GI is a strategic balance of visual quality with computational efficiency and memory management.

Here's a general list of tips for lightmapping:

- **Lightmap resolution:** Higher resolutions capture more detail but increase memory usage. Prioritize larger resolutions for hero objects, and reduce the resolutions for elements in the background.
- **Don't waste texels:** Small or thin objects, like pebbles or wires, can disproportionately use lightmap resources. Disable **Contribute Global Illumination** in either the Static menu or MeshRenderer to exclude those objects from GI calculations unless they significantly influence scene lighting (e.g., they are brightly colored or have emissive materials).



Don't waste texels on small or thin objects.

- **Sampling:** The number of samples directly influences the light bake's quality. More samples yield richer lighting details but extend bake times.
- **Denoising:** In certain conditions like low lighting, baking can introduce visual noise. Choose **Auto** to allow HDRP to choose a denoising algorithm automatically. Otherwise, select **Advanced** to select the Direct Denoiser and Indirect Denoiser.
- **Lightmap compression:** Compression techniques can decrease memory usage but with a potential minor loss in quality.
- **Anti-aliasing:** To optimize performance, consider reducing the anti-aliasing level. For instance, switch from 8x Multi Sampling to 2x Multi Sampling in **Project Settings > Quality**.

GPU lightmapping

You can choose between two backends for the Progressive Lightmapper, the CPU or the GPU. The [Progressive GPU Lightmapper](#) accelerates the generation of baked lightmaps with your computer's GPU and Dedicated Video Ram (VRAM).

When using the GPU Lightmapper, consider these suggestions to optimize bake speed:

- Close other GPU-accelerated applications, especially those that use VRAM
- Switch to a CPU-based denoiser, like Intel Open Image, to free up VRAM
- If you have multiple GPUs, allocate one for rendering and another for baking
- Reduce the number of Anti-aliasing samples, especially for lightmap sizes of 4096 or above

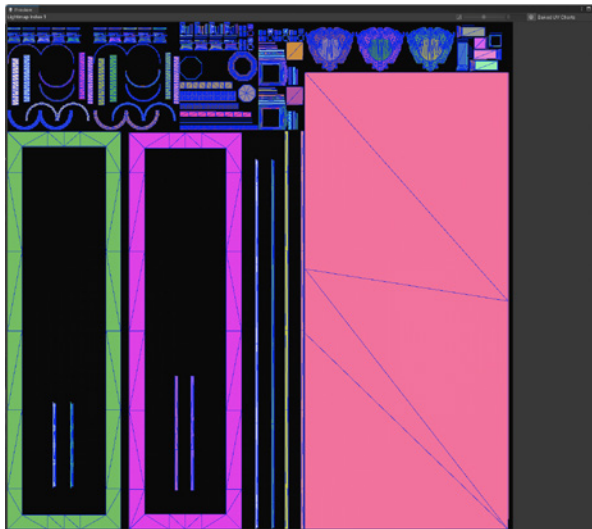
See [these performance guidelines](#) from the manual.

Lightmap UVs

Lightmaps are textures, so Unity needs UVs to correctly use them in your scene.

Unity uses separate sets of lightmap UVs for baked and real-time global illumination due to differences in instance grouping and mesh scaling. UVs are per-mesh, meaning all instances of the same mesh share the same UVs.

Unity can either generate these UVs upon model import or use a set of existing UVs within the model. Unity stores baked lightmap UVs in the Mesh.uv2 channel. This channel maps to the TEXCOORD1 shader semantic and is commonly called "UV1."



Lightmaps need UVs to display correctly.

Unity repacks real-time lightmap UVs to ensure each chart's boundary has a small amount of padding to reduce graphical issues like bleeding (when light leaks from one UV island to the next).

The UVs' calculation depends on the instance's scale and lightmap resolution. Unity optimizes this when possible, allowing Mesh Renderer components with the same mesh, scale, and lightmap resolution to share UVs.

For more technical information about Lightmap UVs, refer to the [documentation page](#).

Real-time global illumination

Unity also includes some more dynamic solutions for real-time global illumination. These won't provide the highest-quality indirect lighting compared to baked lightmapping, but they may be helpful to your application.

Enlighten GI

Unity employs a middleware solution known as Enlighten Realtime Global Illumination. [Enlighten](#) is ideal for lights that change slowly and have a significant visual impact, such as light from the sun's movement.



Baked GI (left) versus Enlighten GI (right).

This feature is best suited for mid- to high-end PC systems, consoles, and some high-end mobile devices. For optimal performance on mobile devices, keep scenes small and maintain a low resolution for real-time lightmaps.

To enable Enlighten, create a new Lighting Settings Asset (from the Lighting window or Create context menu), and enable **Realtime Global Illumination** in the Lighting window.

Enlighten GI splits the scene into small surface patches and determines their visibility to one another. It then approximates how the real-time lights bounce based on this precomputed visibility. Since it's computationally expensive, updating real-time lightmaps happens across several frames.

Enlighten Realtime GI is not suitable for rapidly changing light sources like a flickering neon sign. Also, it may not capture intricate lighting details as effectively as baked GI in certain scenarios.

Using Enlighten Realtime GI

Note these changes when Enlighten Realtime GI is enabled:

- **Light probes:** Light probes (see below) behave differently when Enlighten Realtime GI is enabled. They sample lighting iteratively at runtime.
- **Shadows:** If a light casts shadows, Unity renders both dynamic and static GameObjects into the light's shadow map. Enlighten Realtime GI results also include soft shadows. Shadow Distance settings can be modified in the project settings.
- **Performance:** Enlighten Realtime GI increases memory requirements. The number of shader calculations also increases because it samples an additional set of lightmaps and light probes.
- **Optimization:** If Enlighten Realtime GI doesn't respond quickly enough to changes in scene lighting, you can reduce the real-time lightmap resolution or increase the CPU Usage setting for Realtime GI in the Quality Settings window.

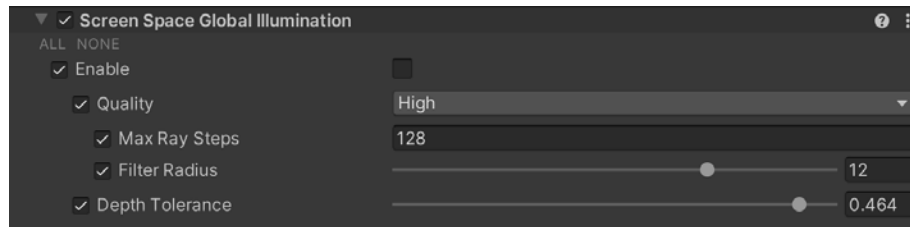
Combine Enlighten Realtime GI with other lighting techniques for optimal results.

Note: Baked GI with Enlighten is deprecated in Unity 2023.1, but Enlighten Realtime GI remains unaffected.

Screen Space Global Illumination

[Screen Space Global Illumination \(SSGI\)](#) uses the depth and color buffer of the screen to calculate bounced, diffuse light. Much like how lightmapping can bake indirect lighting into the surfaces of your static-level geometry, SSGI accurately simulates how photons can strike surfaces and transfer color and shading as they bounce.

SSGI is available from the Volume system. Add an override and then select **Lighting > Screen Space Global Illumination**. See [Volumes](#) for more about HDRP's Volume framework.



Screen Space Global Illumination override

In Room 2 of the sample scene, you can see the green of the moving tree leaves transfer through bounced light onto the wall with SSGI enabled.



SSGI captures the bounced light from the foliage in real-time.

Like other effects that depend on the frame buffer, the edges of the screen become problematic, since objects outside the camera's field of view cannot contribute to the global illumination. This dilemma can be partially improved using reflection probes that will provide a fallback when ray marching outside the frame buffer.

SSGI is enabled in the **Frame Settings** under **Lighting** and must be enabled in the Pipeline Asset's **Lighting** section as well.

Light probes

Baked and real-time global illumination can add realistic lighting to your static environments. Moving objects, however, can use a different lighting technique called light probes.

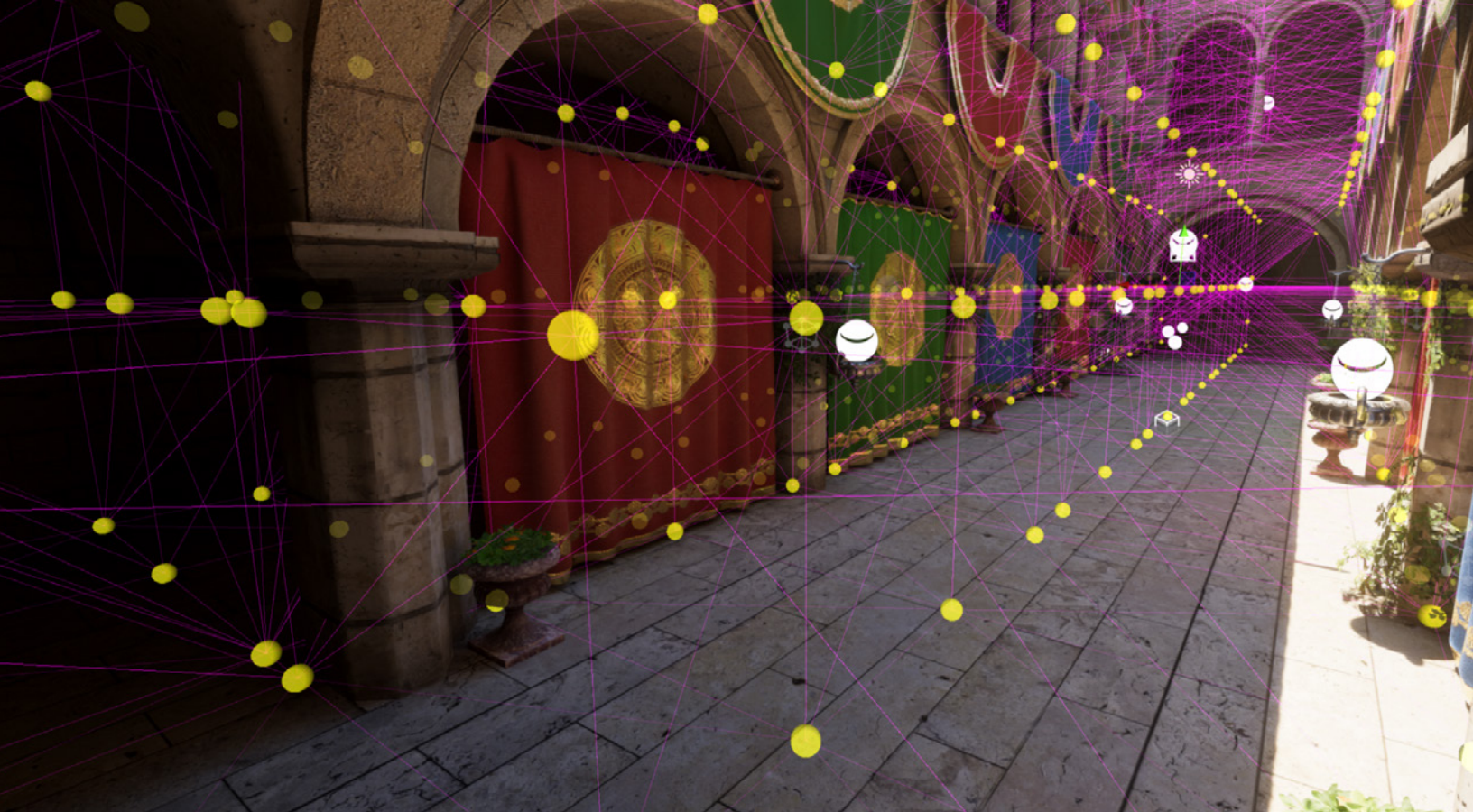
Similar to lightmaps, light probes store “baked” information about lighting in your scene. The difference is that while lightmaps contain information about light hitting the surfaces in your scene, light probes hold information about light passing through empty space.

Use light probes to light moving objects or static scenery using the Level of Detail (LOD) system. Probes capture high-quality light, including indirect bounced light.

Light Probe Groups

To place light probes in your scene, add a Light Probe Group component (**Component > Rendering > Light Probe Group**) to an empty GameObject. Then, use the component settings to edit the positions of the probes, add more, or delete probes.

Arrange light probes in grids or clusters in areas where dynamic objects might move. It's more efficient to place probes around areas where there's a pronounced change in lighting (e.g., near doorways, between indoor and outdoor areas).

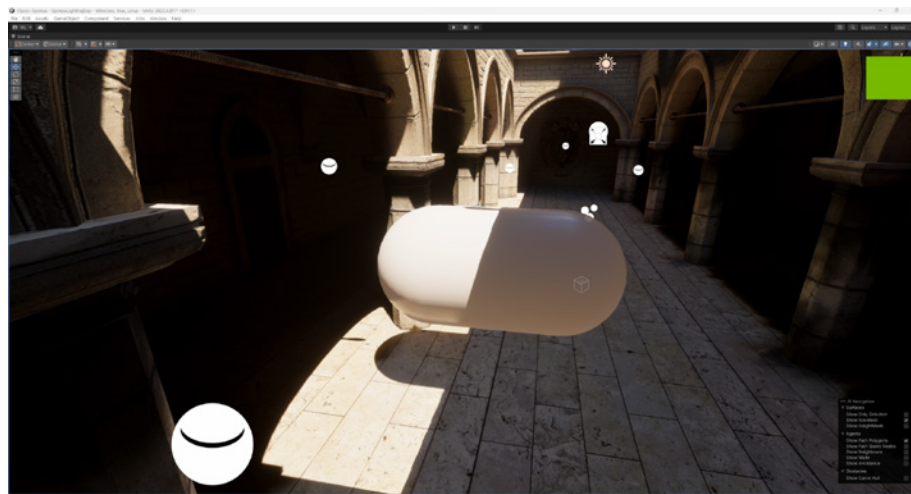


Light probes sample the lighting in the empty spaces.

Once placed, the light probes need to be baked to capture the scene's ambient light. Use the **Generate Lighting** button in the Lighting window, just like with baked lightmaps.

Probe lighting is relatively inexpensive at runtime and quick to precompute. Light probes don't require lightmaps, which can be time consuming to unwrap and can use significant memory.

However, objects lit by light probes blend the results between the four closest probes, and this can result in some inaccurate lighting. In the image below, moving the capsule through the light probes reveals some unnatural illumination, especially when transitioning between light and dark parts of the scene.



Light probes are inexpensive but can produce imprecise results.

While they are fairly quick to calculate, light probes require manual placement. Sampling the lighting environment is an iterative process and can be laborious for the artist. When using HDRP, consider using probe volumes instead of individually arranging light probes in the scene.

Refer to the Light Probes [documentation page](#) and [this blog post](#) for more about light probes and Light Probe Groups.

What is the Sponza Palace?

The 16th-century Croatian palace is the basis for the Sponza Atrium 3D computer graphics model, a widely used reference model for global illumination rendering. We have remastered it in HDRP, and you can find it in this [GitHub repository](#).

Adaptive probe volumes

Hand-placing light probes can be tedious and imprecise. Adaptive probe volumes (named **Probe Volumes** in the Editor) offer a more accurate solution by automating light probe positions. This results in higher-quality lighting that works per pixel, not per object. Compare the probe volume rendering with manually placed light probes.



Probe volumes improve light probe rendering.

Enable probe volumes in the HDRP Quality settings or HDRP asset under **Lighting > Light Probe System**. Enabling **Probe Volumes** will disable **Light Probe Groups** and vice versa.

As the name implies, probe volumes are a volume-based system that can adapt to the geometry density in your scene. A probe volume can distribute probes automatically through the scene.

Visualizing the individual probes in a baked probe volume looks something like this.



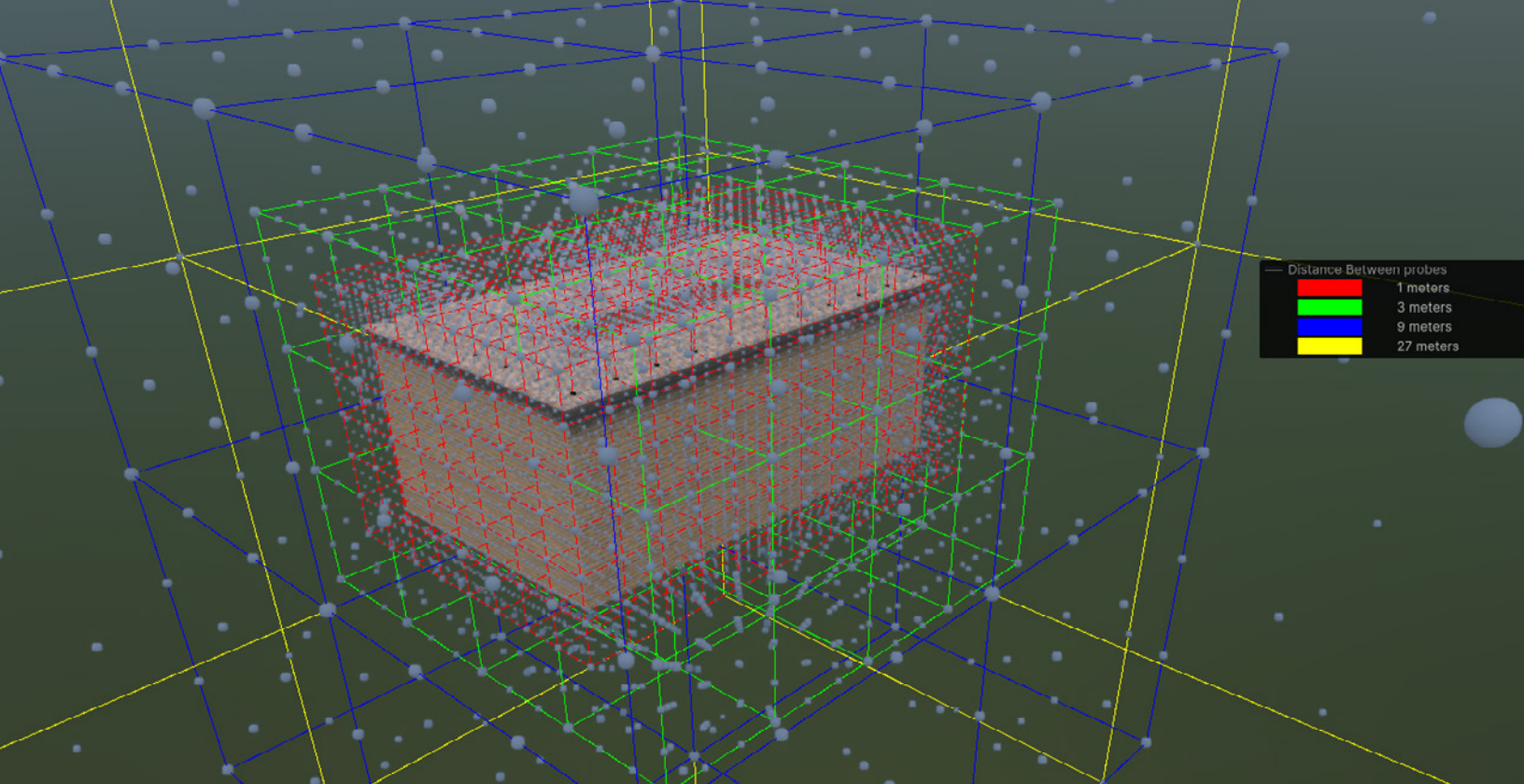
Probe volumes adapt to the geometry density.

Each probe volume contains “bricks” of light probes, with each brick comprising 64 light probes structured in a $4 \times 4 \times 4$ grid. The spacing between these light probes can vary (between 1, 3, 9, or 27 units, dictating the brick’s overall size).

In areas with dense geometry, HDRP uses tightly packed bricks, resulting in higher-resolution lighting data. In sparser areas, probes are spaced farther apart. This concentrates resources where they are needed.

Here are some other features of probe volumes:

- **Per-pixel lighting:** Each pixel on an object receives the appropriate lighting based on its position and surroundings. This results in smoother transitions between light and shadow and improved overall accuracy.
- **Streaming mode:** Probe volumes also include a [Streaming](#) mode to provide high-quality lighting for games set in large open worlds.
- **Baking Sets:** Each scene that uses probe volumes must be part of a [Baking Set](#). This simply maps one set of settings to one or more scenes. If you don’t manually create a Baking Set, HDRP adds your scenes to a default Baking Set.
- **Debugging:** The Rendering Debugger offers a visualization of the brick and light probe layout. This example shows how colors mark different levels of probe density.



Probe volumes automatically create bricks of light probes.

Objects in the scene source their lighting information from the nearest eight light probes. You can fine-tune this to influence which probes an object references.

Set up the probe volumes in **Window > Rendering > Probe Volume Settings**:

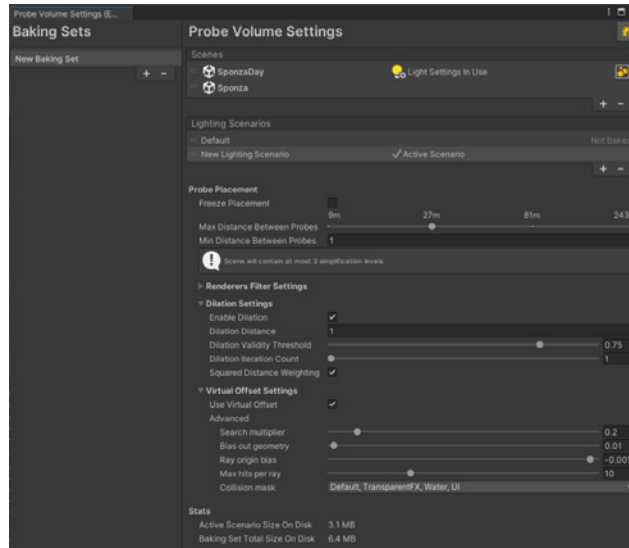
- A **Baking Set** can help manage baking probe volumes. This maps a single set of settings to one or more scenes.

Choose **Baking Mode: Bakings Sets (Advanced)** to choose your own Baking Sets. Otherwise, Unity will automatically add your scenes to a default Baking Set.

- Baked probe volume results reside inside a **Lighting Scenario**. You can switch between Lighting Scenarios at runtime, akin to changing lighting setups. For example, you can use one Lighting Scenario for daytime lighting and another for nighttime lighting.

To create a new Lighting Scenario, select a Baking Set and select + to add a Lighting Scenario. The Lighting Scenario displays **Active Scenario**.

Select **Generate Lighting** to store the baking results in the Lighting Scenario.



HDRP splits the baked data into multiple parts for efficiency. As a result, probe volumes don't duplicate baked data on disk when you use multiple Lighting Scenarios (as long as each maintains the same probe placement and geometry between bakes).

The settings for probe volumes

In some cases, consider switching entirely to probe volumes instead of baked lightmapping. This could reduce your workflow to a single, consistent lighting system for all scene objects.



Baked lightmaps (left) versus probe volumes only (right)

While probe volumes might not capture all of the nuances of baked lightmapping, pairing them with real-time effects, such as Screen Space Ambient Occlusion, offers another approach for lighting your environments.

Refer to this [complete list](#) of probe volume settings and properties. Also, be sure to watch “[Four techniques to light environments in Unity](#)” on how to apply baked GI and probe volumes to your scenes.

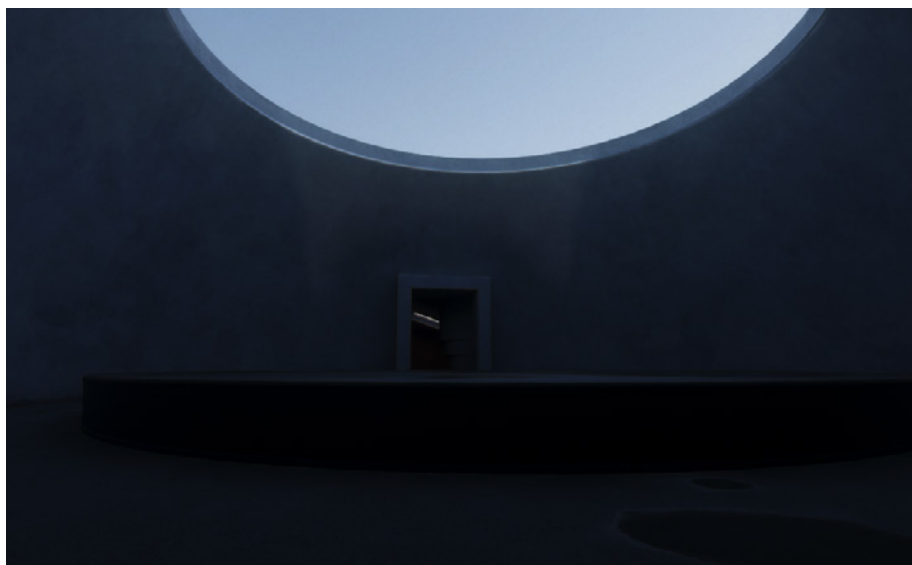
Environment lighting

Because light reflects and scatters around us, the sky and ground in the real world contribute to environment lighting. This is the result of random photons bouncing between the atmosphere and earth and ultimately arriving at the observer.

In HDRP, you can use the **Visual Environment** override to define the sky and general ambience for a scene.

Use **Ambient Mode: Dynamic** to set the sky lighting to the current override that appears in the Visual Environment's **Sky > Type**. Otherwise, **Ambient Mode: Static** defaults to the sky setup in the **Lighting** window's **Environment** tab.

Even with your other light sources disabled, the SampleScene receives general ambient light from the Visual Environment.



Environment lighting only – with the sun directional light disabled, the sky still provides ambient light.

Adding the key light of the sun completes the general illumination of the scene. The environment light helps fill in the shadow areas so that they don't appear unnaturally dark.



Direct sunlight combined with the environment lighting

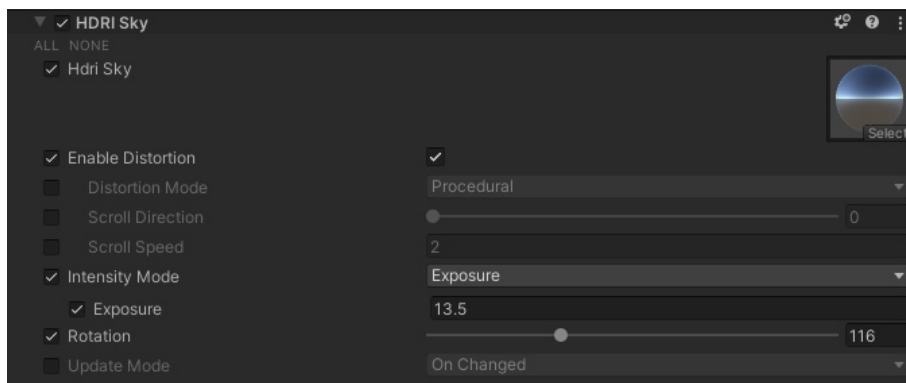
HDRP includes three different techniques for generating skies. Set the **Type** to either **HDR Sky**, **Gradient Sky**, or **Physically Based Sky**. Then, add the appropriate override from the Sky menu.

Applying a Visual Environment sky is similar to wrapping the entire virtual world with a giant illuminated sphere. The colored polygons of the sphere provide a general light from the sky, horizon, and ground.

HDR Sky

HDR Sky allows you to represent the sky with a cubemap made from [high dynamic range photographs](#). You can find numerous free and low-cost sources of HDRIs online. A good starting point is the [Unity HDR Pack](#) in the Asset Store.

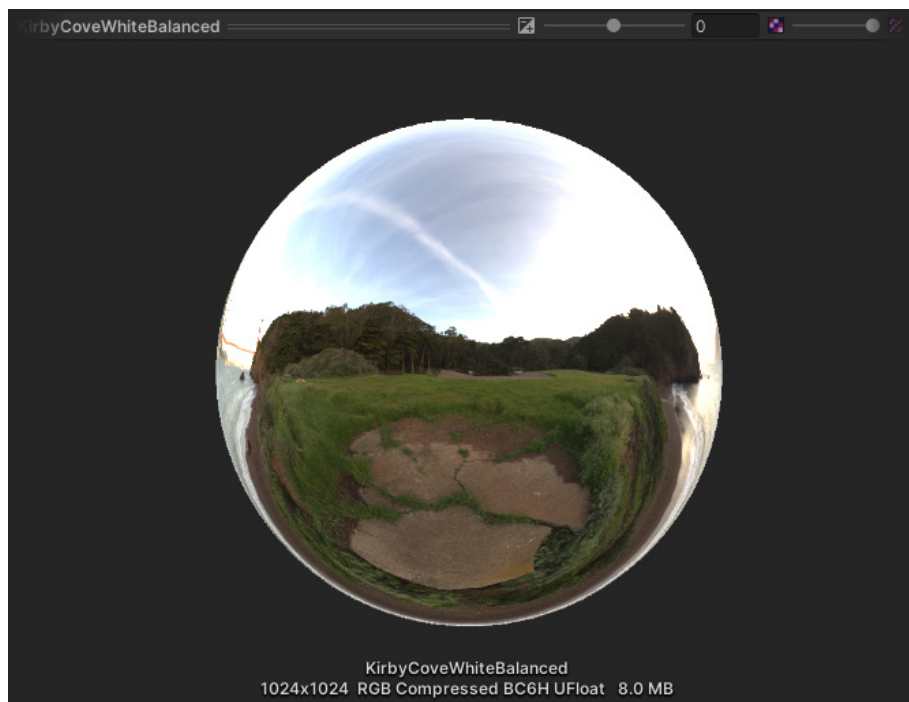
If you're adventurous, we also have a [guide to shooting your own HDRIs](#).



HDR Sky

Once you've imported your HDRI assets, add the **HDRI Sky** override to load the **HDRI Sky** asset. This lets you also tweak options for **Distortion**, **Rotation**, and **Update Mode**.

Because the sky is a source of illumination, specify the **Intensity Mode**, then choose a corresponding **Exposure/Multiplier/Lux** value to control the strength of the environmental lighting. Refer to the Lighting and Exposure Cheat Sheet above for example intensity and exposure values.



An HDRI Sky applied as a cubemap to the interior of a sphere

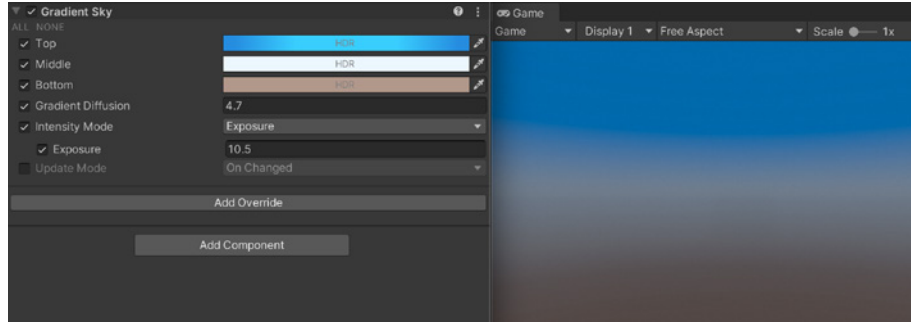
i Animating HDRI skies

You can animate your **HDRI Sky** by distorting the HDRI map either procedurally or with a flow map. This allows you to fake a wind effect on a static HDRI or to create more specific VFX.

Gradient Sky

Choose **Gradient Sky** in the Visual Environment to approximate the background sky with a color ramp. Then add the **Gradient Sky** override. Use the **Top**, **Middle**, and **Bottom** to determine colors for the gradient.

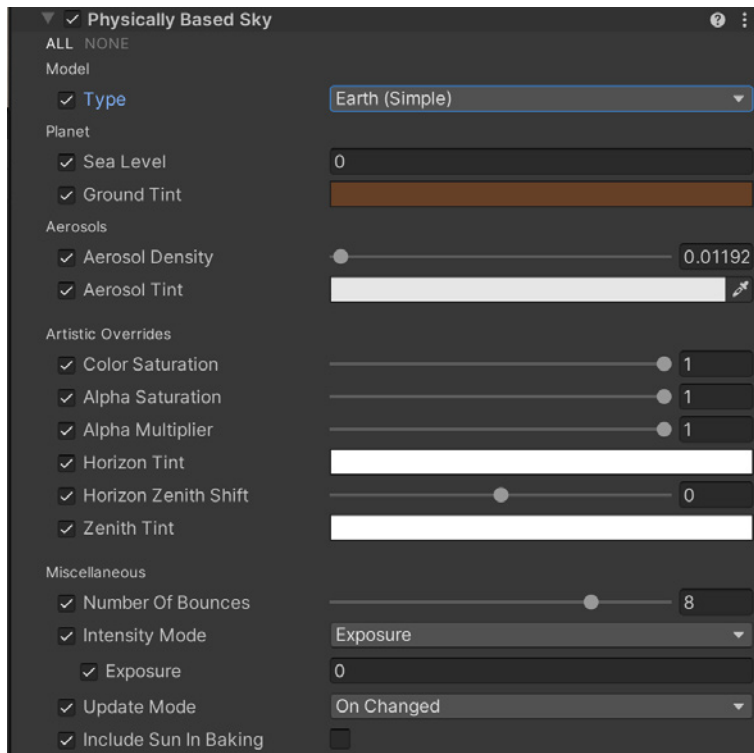
Blend the color ramp with **Gradient Diffusion**, and dial the **Intensity** for the strength of the lighting.



The Top, Middle, and Bottom colors blend into a Gradient Sky.

Physically Based Sky

For something significantly more realistic than a gradient, you can use the **Physically Based Sky** override. This procedurally generates a sky that incorporates phenomena such as Mie scattering and Rayleigh scattering. These simulate light dispersing through the atmosphere, recreating the coloration of the natural sky. Physically Based Sky requires a directional light for accurate simulation.



Physically Based Sky override



A procedurally generated sky from the *Fountainebleau* Demo

i Color tip

Choose the ground color according to the average color of your actual ground (e.g., terrain) where your objects won't be affected by reflection probes.

Ray tracing and path tracing

Ray tracing is a technique that can produce more convincing renders than traditional rasterization. While it has historically been expensive to compute, recent developments in hardware-accelerated ray intersection (or tracing) have now made ray tracing possible for real-time applications.

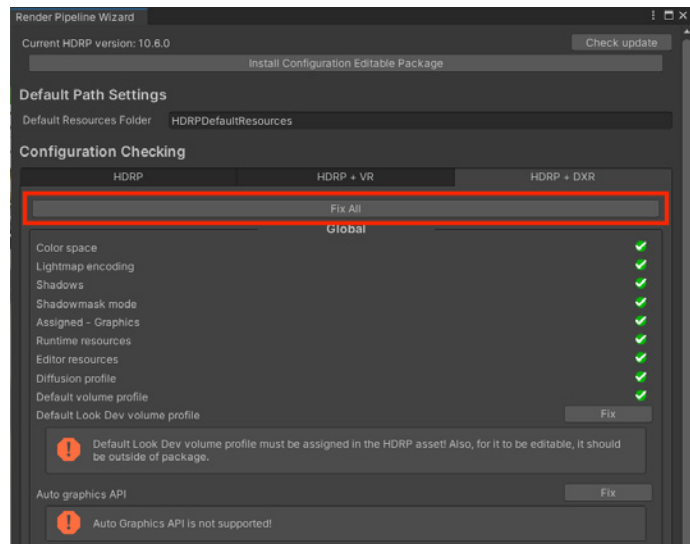
Ray tracing in HDRP is a hybrid system that still relies on rasterized rendering as a fallback and includes preview support for ray tracing with a subset of select GPU hardware and the DirectX 12 API. See [Getting started with ray tracing](#) for a specific list of system requirements.

Ray tracing leaves Preview status in Unity 2023.1 (HDRP 15 and above).

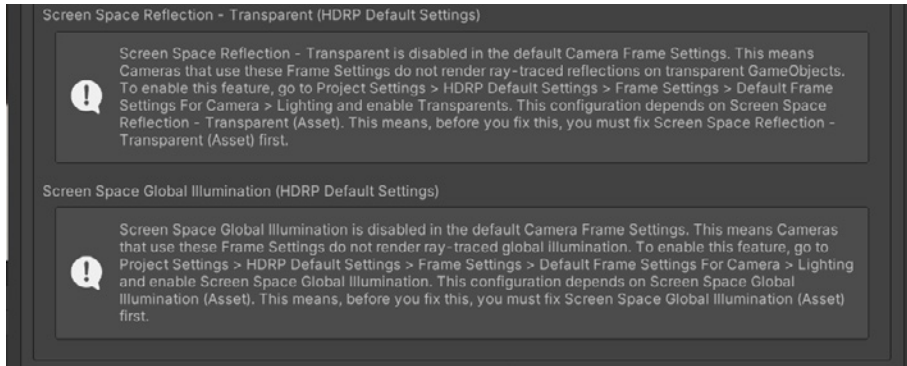
Setup

In order to enable ray tracing (in Preview), you need to change the default graphics API of your HDRP project to DirectX 12.

Open the Render Pipeline Wizard (**Window > Rendering > HDRP Wizard**). Click **Fix All** in the **HDRP + DXR** tab; you will be asked to restart the editor.



Enable ray tracing in the Render Pipeline Wizard.



Follow the instructions to fix any disabled features.

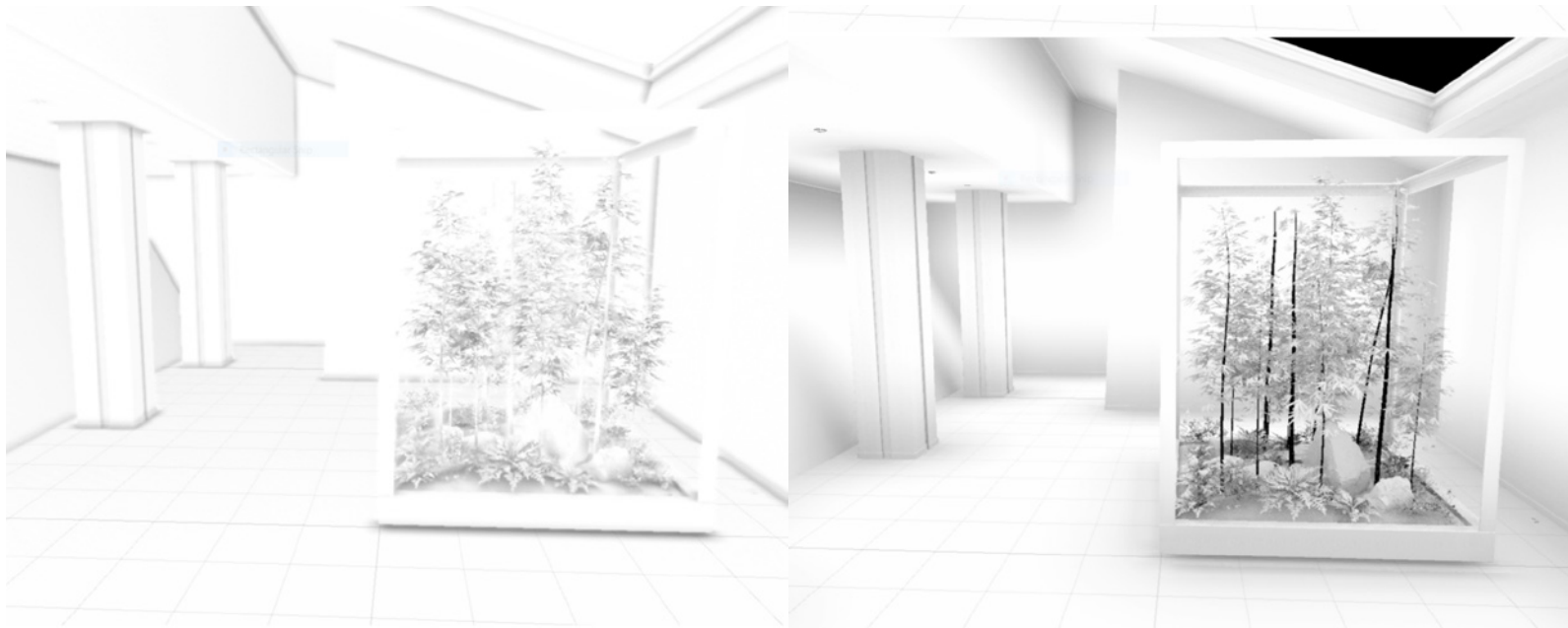
You can also set up ray tracing [manually](#).

Once you enable ray tracing for your project, check that your **HDRP Global** or **Camera Frame** Settings also has ray tracing activated. Make sure you are using a compatible 64-bit architecture in your **Build Settings** and validate your scene objects from **Edit > Rendering > Check Scene Content for HDRP Ray Tracing**.

Overrides

Ray tracing adds some new Volume overrides and enhances many of the existing ones in HDRP:

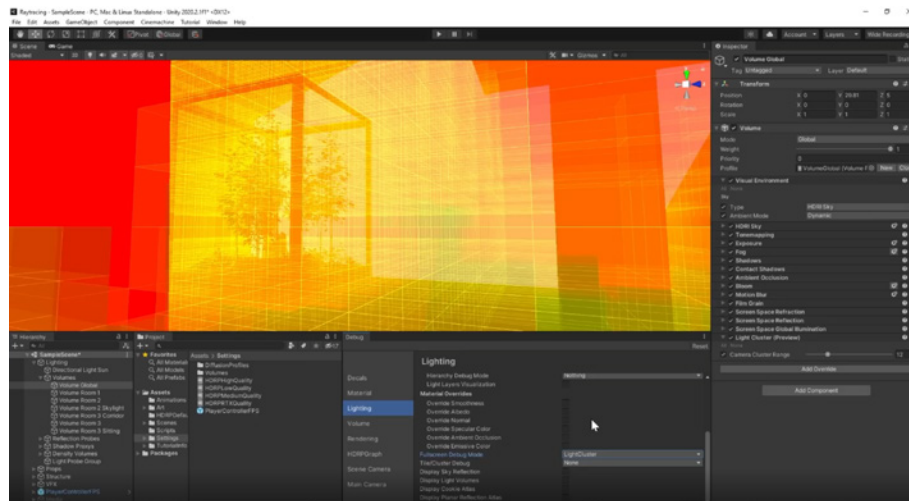
- **Ray Traced Ambient occlusion:** Ray Traced Ambient Occlusion replaces its screen space counterpart (see [Real-time lighting effects below](#)). Unlike SSAO, Ray Traced Ambient Occlusion allows you to use off-screen geometry to generate the occlusion. This way, the effect does not disappear or become inaccurate toward the edge of frame.



Screen Space Ambient Occlusion (left image) vs Ray Traced Ambient Occlusion (right image)

- **Light clusters:** When using ray traced reflections, GI, SSS, and recursive rendering, you need to make sure the lights are efficiently stored in the light cluster to avoid artifacts and optimize performance. HDRP divides your scene into an axis-aligned grid centered around the camera. HDRP uses this structure to determine the set of local lights that might contribute to the lighting whenever a ray hits a surface. It can then compute light bounces for certain effects (Ray Traced Reflections, Ray Traced Global Illumination, and so on). Use the **Camera Cluster Range Volume Override** to alter the range of this structure to make sure it encompasses the GameObjects and lights that need to be considered.

You can use an **HDRP Debug** mode available via **Windows > Analysis > Rendering Debugger > Lighting > Full Screen Debug mode**. It helps visualize the Light Clusters Cells highlighted in red, indicating where the light count has reached the **Maximum Lights per Cell** in the HDRP asset. Adjust this setting to reduce unwanted light leaking or artifacts.



Ray tracing light clusters in Debug mode

- **Ray Traced global illumination:** This is an alternative to SSGI and light probes for simulating bounced, indirect lighting. Ray Traced global illumination is calculated in real-time, and it allows you to avoid the lengthy offline process of baking lightmaps while yielding comparable results.

Use the **Quality** setting for complex interior environments that benefit from multiple bounces and samples. **Performance** mode (limited to one sample and one bounce) works well for exteriors, where the lighting mostly comes from the primary directional light.

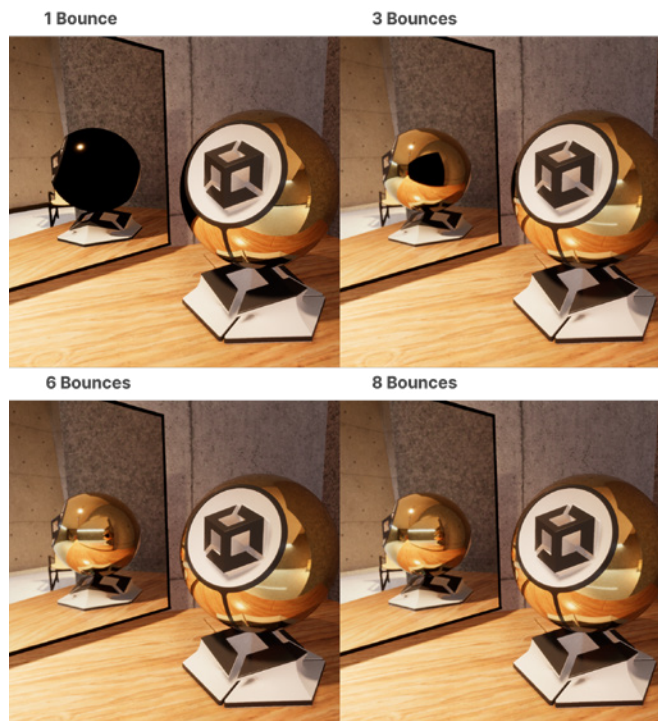


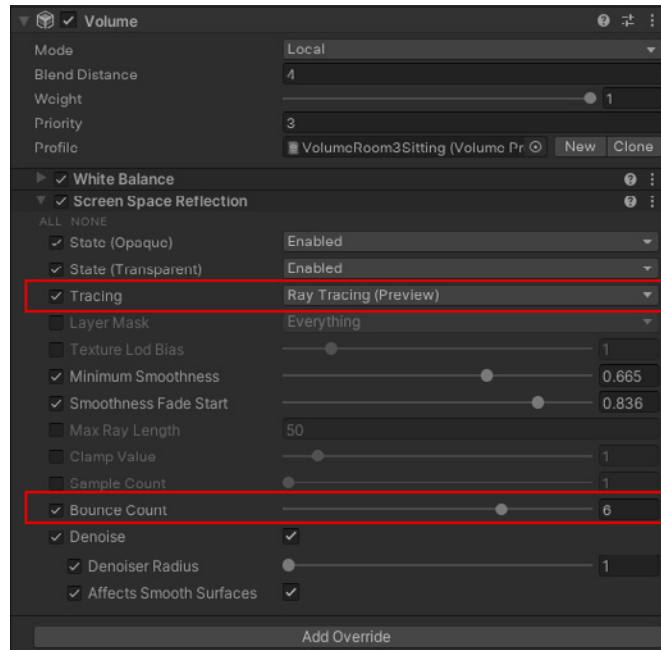
Ray traced global illumination shows bounced lighting in real-time.

- **Ray traced reflections:** With ray traced reflections, you can achieve higher-quality reflections than using reflection probes or screen space reflections. Off-screen meshes appear correctly in the resulting reflections.

Adjust the **Minimum Smoothness** and **Smoothness Fade Start** values to modify the threshold at which smooth surfaces start to receive ray traced reflections. Increase the **Bounces** if necessary, but be aware of the performance cost.

The below example demonstrates the effect of ray traced bounces in an “infinite mirror” setup (e.g., two mirrors reflecting each other). The series shows the progression of reflections as they become more complex with 1, 3, 6, and 8 bounces. The increasing number of bounces creates a sense of depth as the reflections appear to recede into the distance.





Ray tracing can improve screen space reflections.

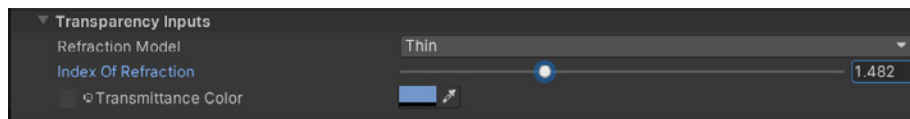
- **Ray traced shadows:** Ray traced shadows for directional, point, spot, and rectangle area lights can replace shadow maps from any opaque GameObject. Directional lights can also cast ray traced shadows from transparent or translucent GameObjects.



Ray traced shadows soften as they fall farther from the caster to achieve a different effect than with shadow mapping. Directional, point and spot lights can also generate semi-transparent shadows.

Ray tracing can produce natural-looking shadows that soften as their distance from the caster increases, like in real life.

HDRP's directional lights can also generate semi-transparent, colored shadows. In the image below you can see that a glass surface casts a realistically tinted shadow on the floor.



Use Transmittance Color for transparent shadow casters.



Directional lights can generate ray traced colored shadows (right).

Watch [Activate ray tracing with HDRP](#) for a walkthrough of HDRP's ray tracing features. See the ray tracing documentation on the HDRP microsite for more information.

Performance

Ray tracing comes with additional performance cost. Building and updating the ray traced world on the GPU will scale with the complexity of the world and how often it needs to be updated, while sending rays and applying effects will scale linearly with the number of pixels affected.

Here are a few tricks, systems, and settings to help you optimize performance:

- **Dynamic resolution:** Because ray tracing scales linearly with the number of pixels, rendering at a lower resolution saves a lot on performance. Most content using ray tracing in real-time leverages modern upscalers (e.g., NVIDIA DLSS) in order to render the frame buffer at a sub resolution which will be upsampled to reach the final screen resolution with very limited visual fidelity loss. For example, you can start by activating DLSS and use a forced screen percentage of 75%. This will allow you to gain on the GPU time, and consequently on the frame time if you are CPU bound.
- **Pick your effects:** There's a strong chance that you can't afford to run all ray traced effects in a game. In a car game, for example, you might pick ray traced reflections to improve the car visual fidelity, while in an outdoor adventure game you might prefer ray traced directional shadows and global illumination to improve the immersion in the environment.
- **Use ray tracing only when necessary:** For multiple effects like Ray Traced Reflections and Global Illumination, you can try [Mixed tracing mode](#), which uses fast ray marching to resolve the rays on screen, and only falls back on ray tracing when necessary. You can also tweak your effects settings to use more of the fast fallbacks (Probe Volumes, the Sky Volume override, or Reflection Probes for reflections) to reduce the need for longer rays without reducing the quality of the final result.

- **Optimize the evaluation:**
 - On some effects, you can use a layer mask to exclude some objects when evaluating an effect.
 - Choose different quality settings, or build custom quality settings for ray length (the longer, the more noisy, the more one needs samples), denoiser settings (remember that using temporal anti-aliasing is key to reduce noise), sample count (the more samples, the less noisy, but the more expensive), and resolution (number of pixels per ray).
 - For reflections, you can reduce the number of bounces if reflective objects don't need to be seen inside of other reflective objects. You can also tweak the smoothness parameters to reduce the number of objects using ray traced reflections so it's only applied to those where ray tracing provides a significant visual improvement.
 - Finally, for ray traced effects requiring textures, a texture LOD bias allows the use of a lower-texture MIP to reduce the cost of texture fetches during the material evaluation.
- **General Ray Tracing Settings:** These settings allow you to balance performance and visual quality, configuring rays, and the way the acceleration structure is built.

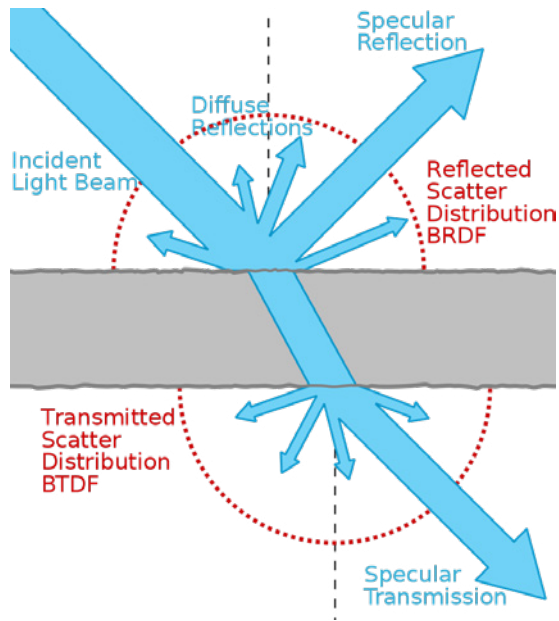
To analyze performance, use the [display stats](#) accessible through the Rendering Debugger (**Window → Render Pipeline Debug**) when in Play mode, and access a set of markers to understand the cost of every ray tracing effect. The “CPU timings RT” matches the cost in millisecond to execute on the CPU the target effect (attribute binding, C# code, etc.). On the other hand, the “GPU timings RT” matches the GPU execution time in milliseconds to evaluate the effect.

Path tracing

Path tracing is a type of ray tracing that simulates many possible paths for each light ray, including how it bounces off multiple surfaces. This allows it to capture more complex interactions between light and materials.

Like ray tracing, path tracing starts by shooting rays from the camera into the scene. However, instead of stopping after the first encounter with a surface, the ray continues to bounce around the scene multiple times, interacting with various objects.

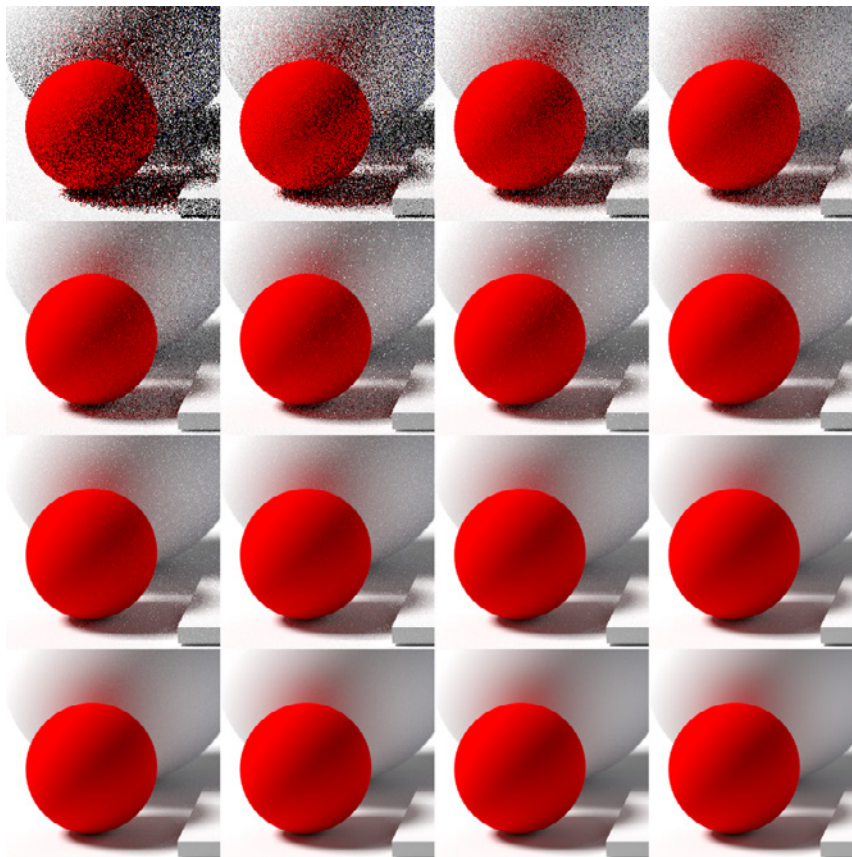
At each bounce, the renderer gathers light information, taking into account both direct and indirect illumination. Color information from all these bounces is accumulated to determine the final color of a pixel.



Path tracing scattering. Source: Wikipedia

Path tracing captures global illumination effects more naturally, including subtle diffuse reflections and soft shadows. This technique produces more physically accurate and realistic images compared to basic ray tracing.

However, path tracing is more computationally intensive than ray tracing due to its multiple bounces. Noise can be an issue, especially with the fewer samples necessary for real-time performance.



Path tracing creates noise with fewer samples. Source: Wikipedia

HDRP now includes new denoising techniques to mitigate this:

- NVIDIA Optix™ AI-accelerated denoiser
- Intel® Open Image Denoise (available as an opt-in package)

HDRP also adds the **Use AOVs** (Arbitrary Output Variables) setting to Material shaders that use path tracing to support the new path tracing denoisers. When you enable this setting, HDRP puts albedo and normal values into AOVs and can improve your path tracing results.

DirectX 12

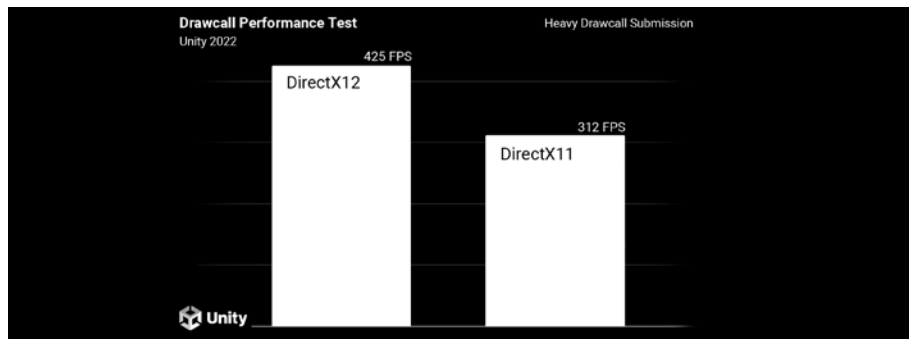
You can choose to set Direct X11 (DX11) or Direct X12 (DX12) as your default Graphics API in the Editor or Standalone Player. Starting with Unity 2022 LTS, DX12 is out of preview.

The Direct X12 (DX12) graphics backend in Unity has undergone notable improvements. In scenarios with many draw calls, DX12 demonstrates superior CPU performance in standalone builds.

Note, however, DX12 doesn't always outperform DX11. For example, DX11 drivers can reorder compute shader dispatch calls more efficiently than DX12, Metal, or Vulkan. Scenes with intensive GPU workloads and intricate compute shaders might perform better with DX11.

To address Editor performance, DX12 introduces the option to run native graphics jobs in the Editor. Though this feature remains in the experimental phase, users can activate it using the command line argument **-force-gfx-jobs native**.

See the blog post "[Reach more players over multiple platforms and form factors](#)" to learn more.



Performance test comparing Direct X12 and Direct X11 in Unity 2022 LTS.

For version Unity 2022 LTS and beyond:

- If your project is GPU-bound, opt for DX11.
- If it's CPU-bound, use DX12.
- For those pushing Editor performance boundaries, try DX12 with native graphics jobs in the Editor.

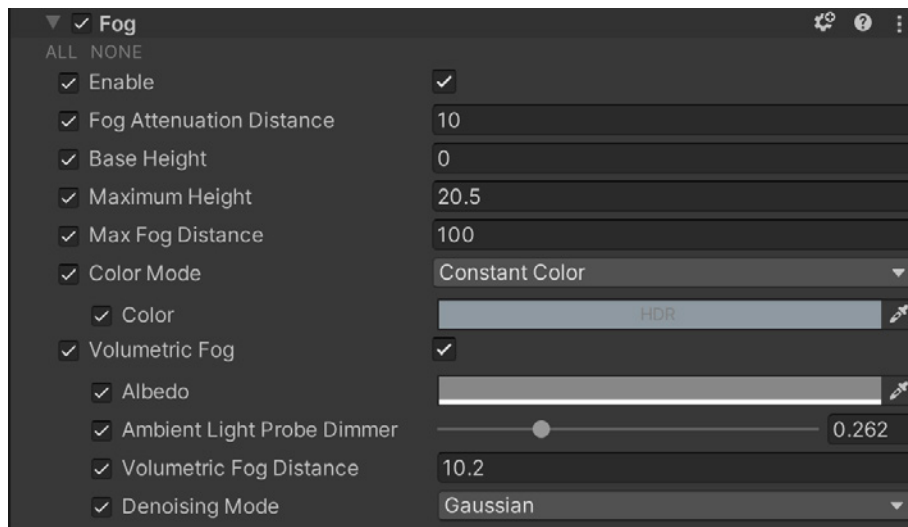
Fog and atmospheric scattering

Smoke, fog, and haze are traditional tools of cinematography. They can help add depth and dimension to stage lighting or create an atmospheric mood. You can use fog for a similar advantage in HDRP.

Its opacity depends on the object's distance away from the camera. Fog can also hide the camera's far clipping plane, blending your distant geometry back into the scene.

Global fog

HDRP implements global fog as a **Fog** override. Here, the fog fades exponentially with its distance from the camera and its world space height.



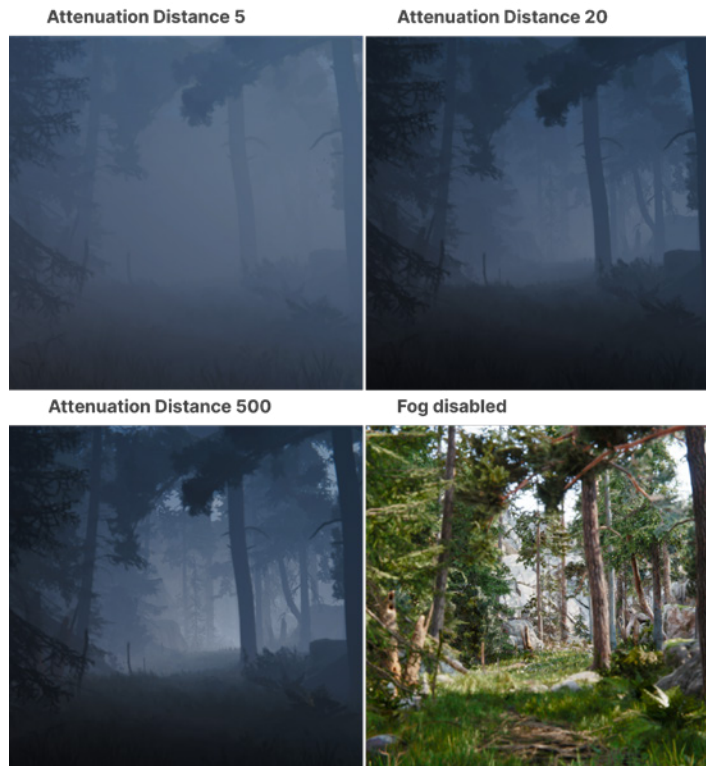
Fog override

Set up the Fog override on a Volume in your Scene. The **Base Height** determines a boundary where constant, thicker fog begins to thin out traveling upward. Above this value, the fog density continues fading exponentially, until it reaches the **Maximum Height**.



Use the Base Height and Maximum Height settings to make low-hanging fog.

Likewise, the **Fog Attenuation Distance** and **Max Fog Distance** control how fog fades with greater distance from the camera. Toggle the **Color Mode** between a **Constant Color** or the existing **Sky Color**.



The Fog Attenuation Distance setting modifies how fog fades into the background (Volumetric Fog shown).

Enable **Volumetric Fog** to simulate atmospheric scattering. Make sure to check **Fog** and Volumetrics in the **Frame Settings** (either under the camera or in HDRP Default Settings) under Lighting. Also, enable **Volumetric Fog** in the HDRP Pipeline Asset.

Volumetric Fog Distance sets the distance (in meters) from the Camera's Near Clipping Plane to the back of its volumetric lighting buffer. This fills the atmosphere with an airborne material, partially occluding GameObjects within range.



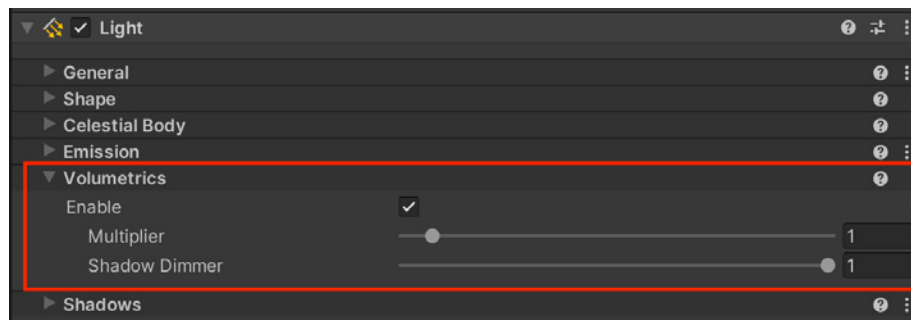
Volumetric Fog accurately holds out the foreground geometry.

Volumetric Lighting

[Volumetric Lighting](#) can simulate rendering dramatic sunbeams, like [crepuscular rays](#) behind the clouds at sunset or passing through foliage.

Each Light component (except area lights) has a **Volumetrics** group. Check **Enable**, then set the **Multiplier** and **Shadow Dimmer**. A **Real-time** or **Mixed Mode** light will produce “god rays” within Volumetric Fog.

The Multiplier dials the intensity, while the Shadow Dimmer controls how shadow casting surfaces cut into the light.



Volumetrics in the Light component

The shafts of light only appear within the Volumetric Fog, so adjusting the fog's Base Height and Maximum Height controls their falloff.

Tips for volumetric lighting and shadows

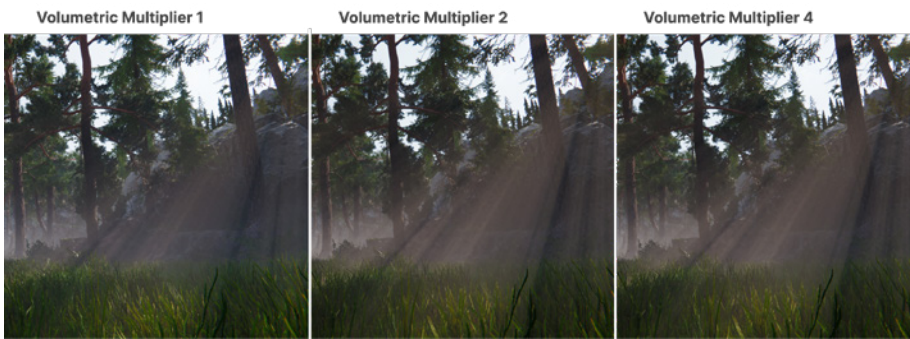
Need to add some dramatic flair with volumetric lighting and shadows? Consider these tips:

- Objects that obstruct the light, such as buildings or trees, will create shadows and allow the light shafts to become more visible around the edges of these objects.
- The appearance of the light shafts depends on the camera's position and angle relative to the light source. The rays will be more noticeable when the camera nearly aligns with the light direction.
- The brightness and color of the light affect the appearance of the god rays. Higher intensity and specific color choices can make the rays more prominent.
- Volumetric Fog, dust, or other particles in the scene will scatter the light and make the rays more pronounced. VFX Graph can add detail to each light shaft and make them more animated.



Enable the directional light's volumetric option to show light shafts within the Volumetric Fog.

These examples show the effects of the Volumetric Multiplier on a directional light and a spotlight.

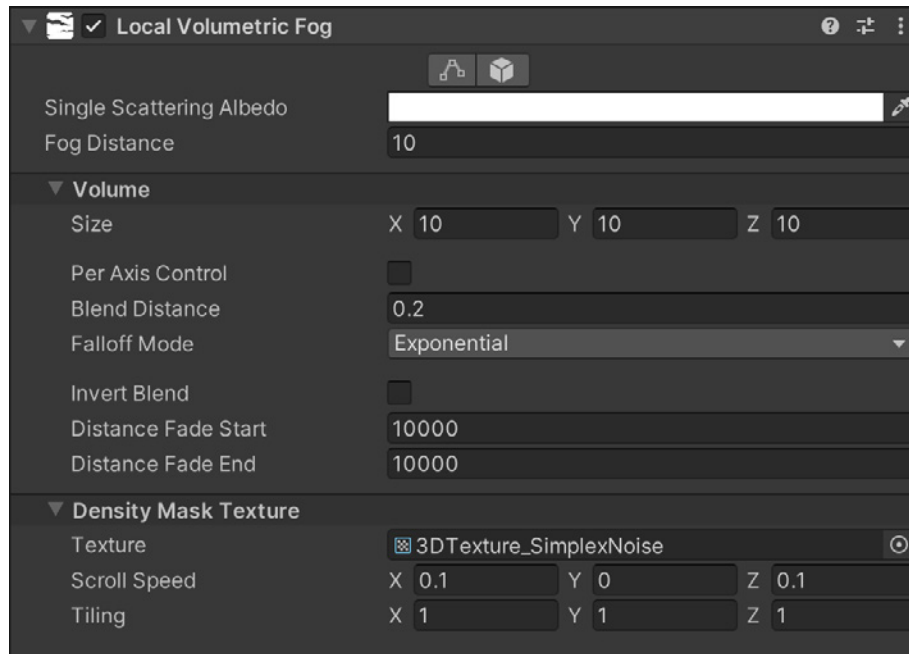


Volumetric lighting applied to the ceiling spotlights in the HDRP Sample

Local Volumetric Fog

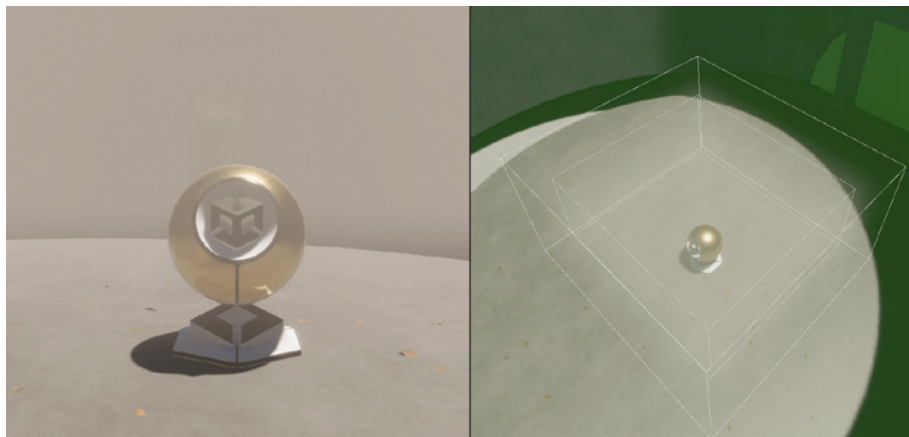
If you want more detailed fog effects than the Fog override can provide, HDRP additionally offers the [Local Volumetric Fog](#) component (formerly called a Density Volume in older versions of HDRP).

This is a separate component, outside the Volume system. Create a **Local Volumetric Fog** GameObject from the menu (**GameObject > Rendering > Local Volumetric Fog**) or right-click over the Hierarchy (**Rendering > Local Volumetric Fog**).



The Local Volumetric Fog component

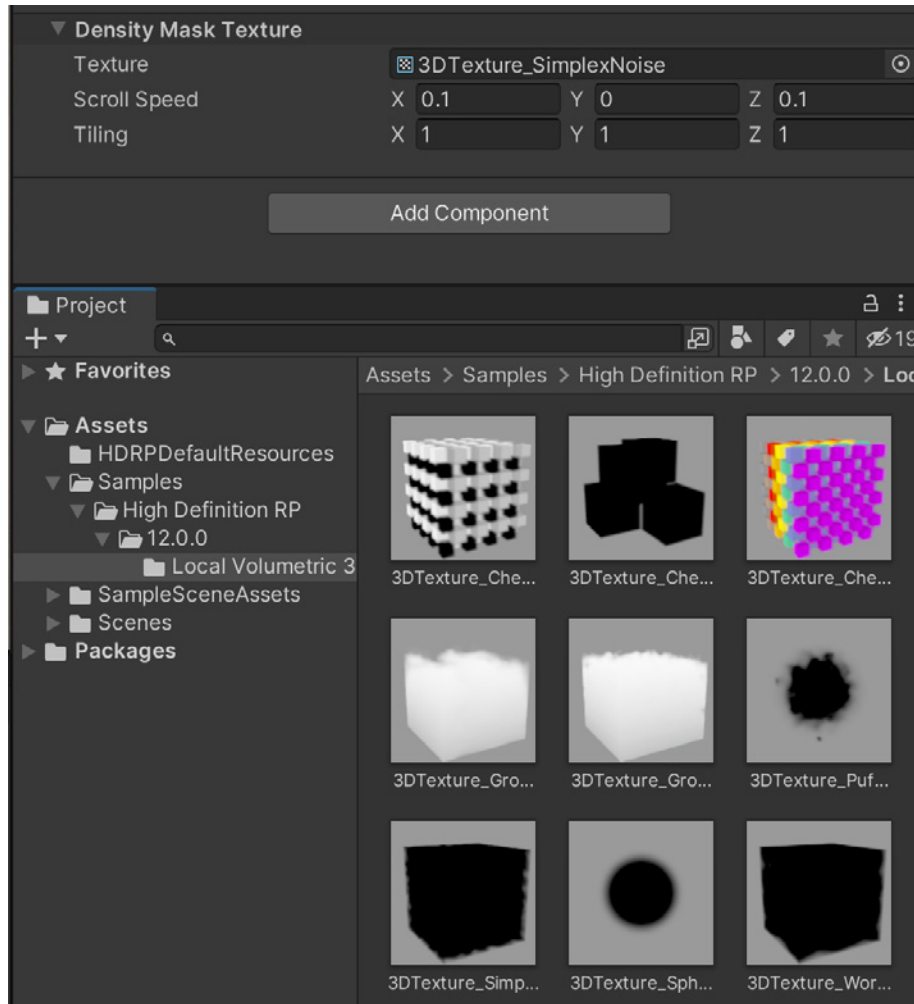
This generates a fog-filled bounding box. Adjust the size, axis control, and blending/fading options.



Local Volumetric Fog appears in a bounding box.

By default, the fog is uniform, but you can apply a 3D **Texture** to the Texture field under the **Density Mask Texture** subsection. This gives the user more flexibility to control the look of the fog. Download examples from the Package Manager's **Local Volumetric 3D Texture Samples** or follow the [documentation procedures](#) to create your own Density Masks.

Add some **Scroll Speed** for animation and adjust the **Tiling**. Your volumetric fog then can gently roll through the scene.



Density Mask Texture from the Local Volumetric 3D Texture Samples

Note: HDRP voxelizes Local Volumetric Fog to enhance performance. However, the voxelization can appear very coarse. To reduce aliasing, use a Density Mask Texture and increase the Blend Distance to soften the fog's edges. You can enable local volumetric resolutions up to 256×256×256 in your HDRP Pipeline Asset, allowing for more precise, large-scale effects.

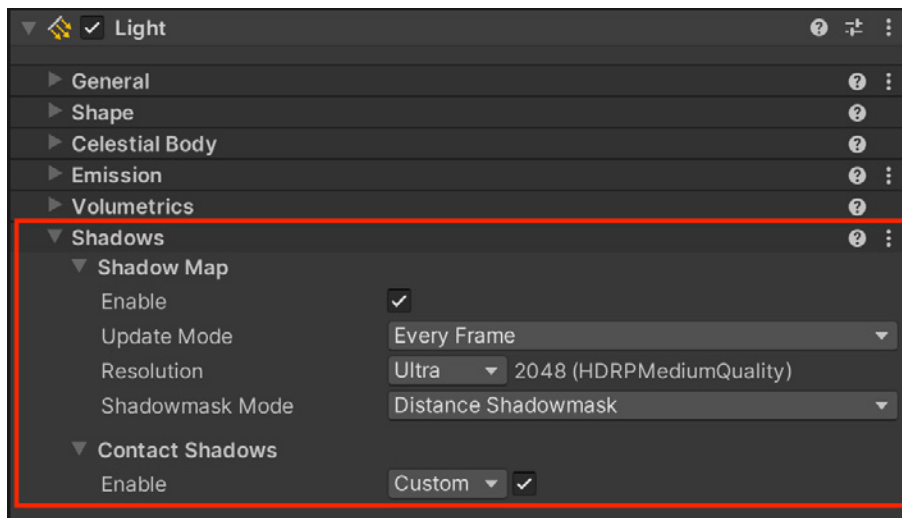
Shadows

We can't perceive light without darkness. Well-placed shadows in your scene add as much interest as the lighting itself and can imbue your scenes with extra depth and dimension. HDRP includes a number of features to fine-tune your shadows and prevent your renders from looking flat.

Shadow maps

Shadows render using a technique called [shadow mapping](#), where a texture stores the depth information from the light's point of view.

Locate the Shadows subsection of the Light component to modify your shadow mapping **Update Mode** and **Resolution**. Higher resolutions and update frequency settings cost more resources.



Shadow settings per light

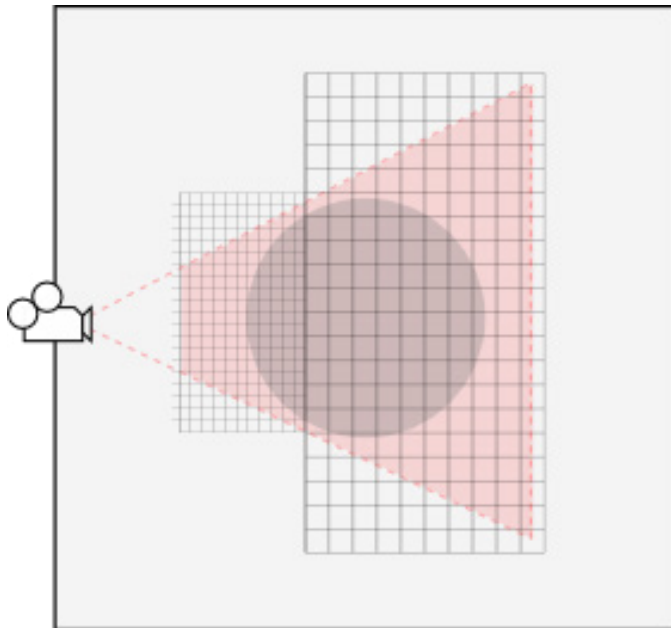
Shadow Cascades

For a directional light, the shadow map covers a large portion of the scene, which can lead to a problem called *perspective aliasing*. Shadow map pixels close to the camera look jagged and blocky compared to those farther away.



Perspective aliasing with blocky shadows

Unity solves this with cascaded shadow maps. It splits the camera frustum into zones, each with its own shadow map. This reduces the effect of perspective aliasing.

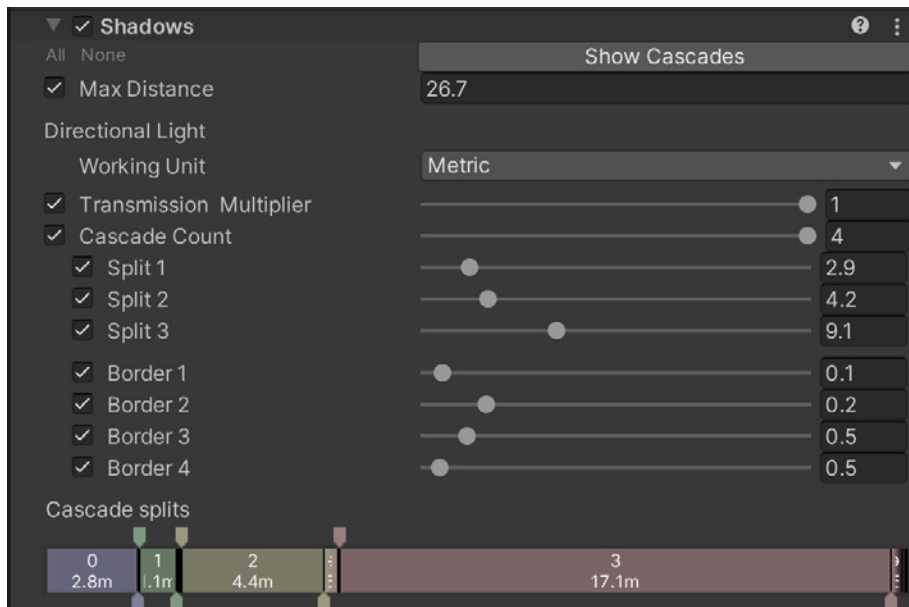


Shadow cascades break the camera frustum into zones, each with its own shadow map.



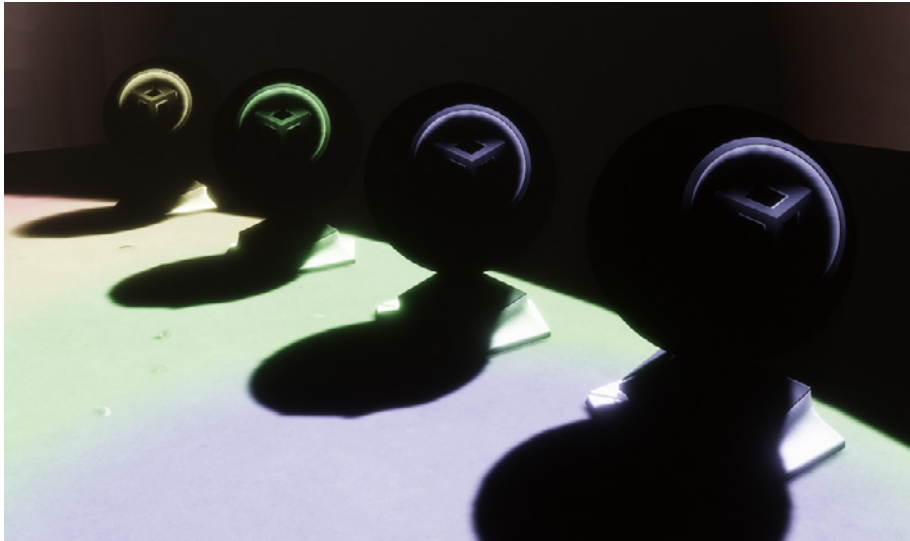
Shadow Cascades reduce perspective aliasing.

HDRP gives you extra control over your shadow cascades with the [Shadows](#) override. Use the cascade settings per volume to fine-tune where each cascade begins and ends.



The Shadows override can adjust the shadow cascades.

Toggle the **Show Cascades** button to visualize the cascade splits more easily. With some tweaking, you can keep perspective aliasing to a minimum.



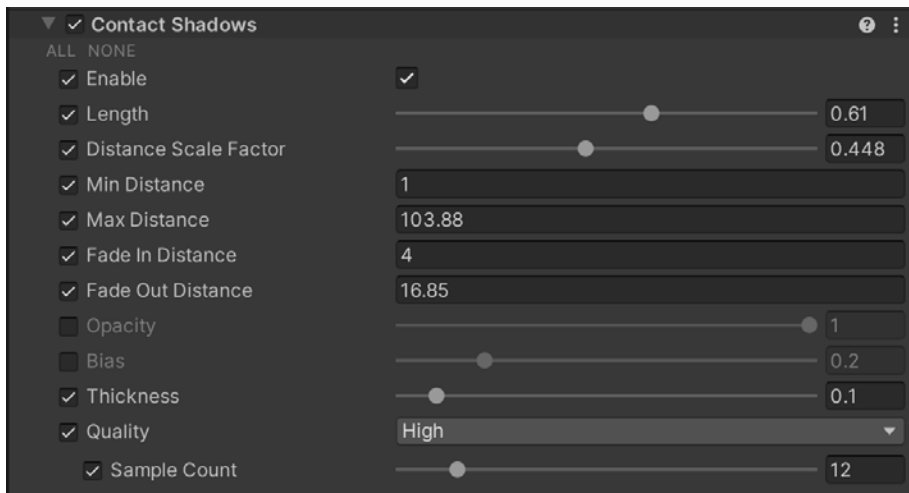
Click the Show Cascades option to visualize the cascade splits.

Contact Shadows

Shadow maps often fail to capture the small details, especially at discernible edges where two mesh surfaces connect. HDRP can generate these contact shadows using the **Contact Shadows** override.

Contact Shadows are a screen space effect that rely on information within the frame in order to calculate. Objects outside of the frame do not contribute to contact shadows. Use them for shadow details with a small onscreen footprint.

Make sure that you enable **Contact Shadows** in the **Frame Settings**. You can also adjust the **Sample Count** in the Pipeline Asset under **Lighting Quality Settings**.



Contact Shadows override



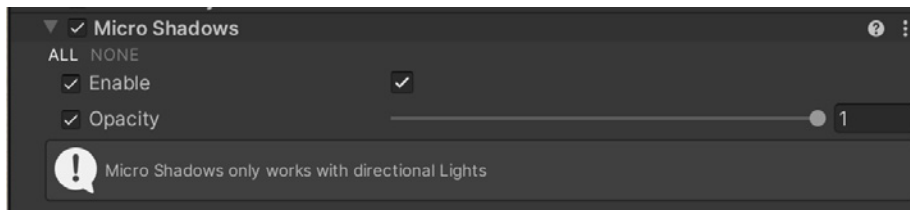
Contact Shadows

Adjust the settings for the shadow thickness, quality, and fade.

This feature was improved to work better with Terrain and Speedtree. Read more on [the blog](#).

Micro Shadows

HDRP can extend even smaller shadow details into your Materials. **Micro Shadows** use the normal map and ambient occlusion map to render really fine surface shadows without using the mesh geometry itself.



Micro Shadows override

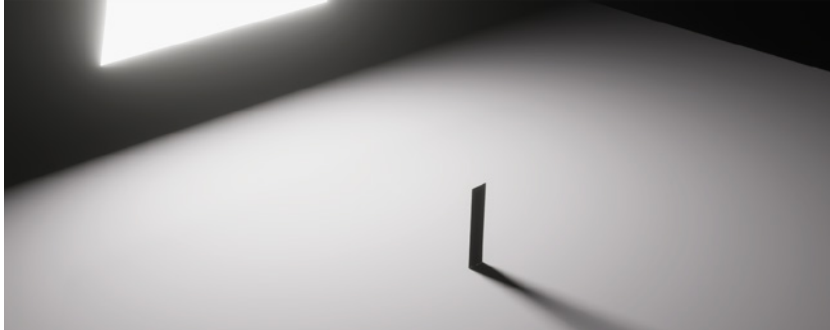
Simply add the **Micro Shadows** override to a Volume in your Scene and adjust the Opacity. Micro Shadows only work with directional lights.



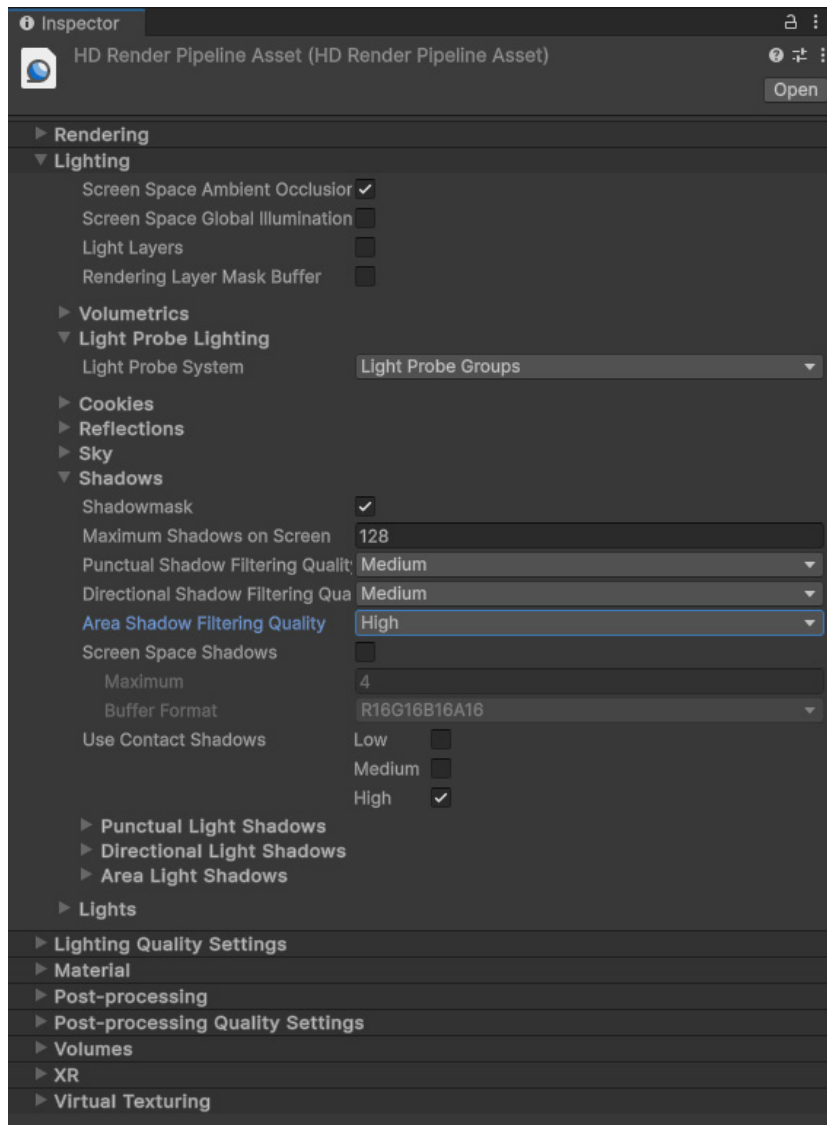
Micro Shadows add extra contrast to this bed of foliage.

Area light soft shadows

Soft area shadows are more accurate in Unity 2022 LTS and above. Set the **Area Shadow Filtering Quality** in the HDRP Asset to High for improved soft shadows with area lights.



Area lights show improved soft shadows.



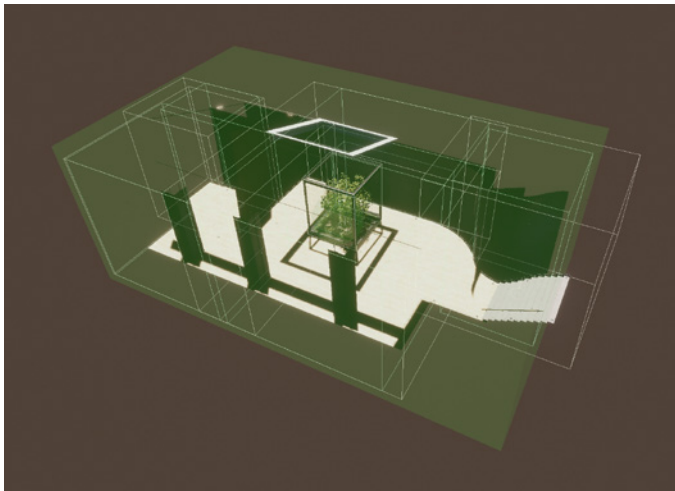
Area Shadow Filtering Quality

Reflections

Reflections help integrate your GameObjects with their surrounding environment. Though we normally associate reflections with smooth and shiny surfaces, even rough materials need to receive correct reflections in a PBR workflow. HDRP offers multiple techniques to generate reflections:

- Screen space reflections
- Reflection probes
- Sky reflections

Each reflection type can be resource-intensive, so select the method that works best for your use case. If more than one reflection technique applies to a pixel, HDRP blends the contribution of each reflection type. Bounding surfaces called Influence Volumes partition the 3D space to determine which objects receive the reflections.

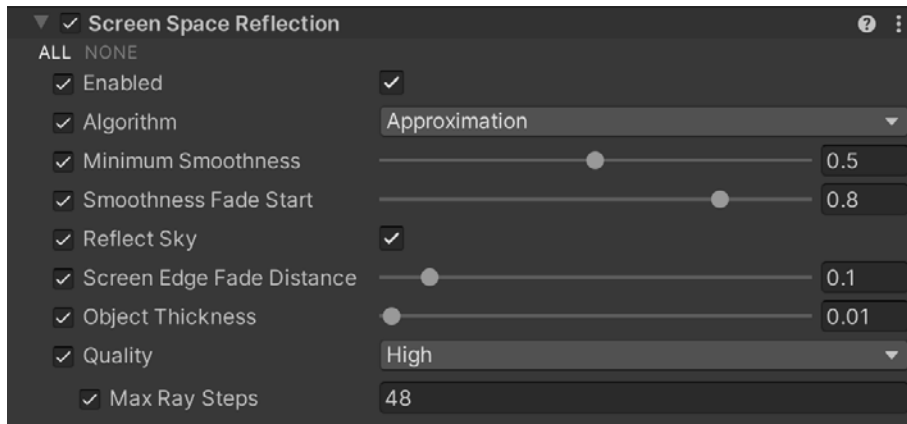


Influence Volumes determine where reflection probes create reflections.

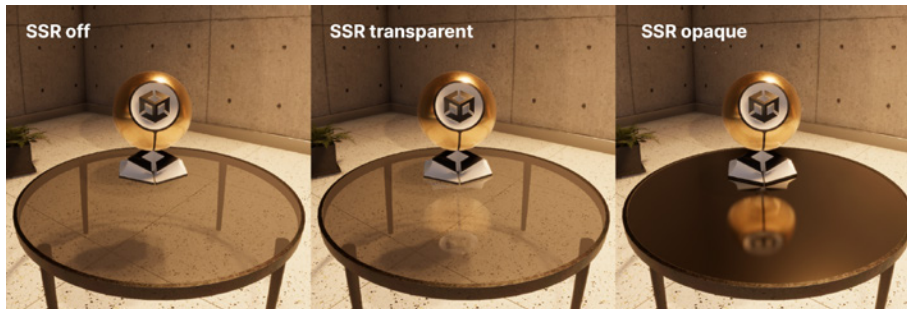
Screen Space Reflections

Screen space reflections use the depth and color buffer to calculate reflections. Thus, they can only reflect objects currently in camera view and may not render properly at certain positions on screen. Glossy floorings and wet planar surfaces are good candidates for receiving screen space reflections.

Screen space reflections ignore all objects outside of the frame, which can be a limitation to the effect.



Screen Space Reflection override



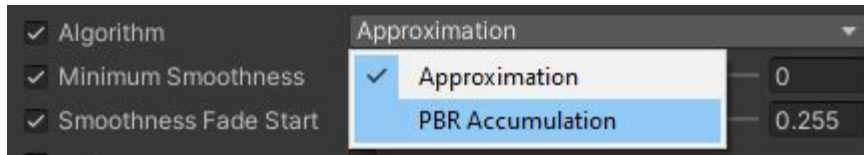
Screen space reflections work for transparent and opaque materials.

Make sure you have **Screen Space Reflection** enabled in the **Frame Settings (HDRP Default Settings or the Camera's Custom Frame Settings)** under **Lighting**. Then, add the **Screen Space Reflection** override to your Volume object.

Material surfaces must exceed the **Minimum Smoothness** value to show screen space reflections. Lower this value if you want rougher materials to show the SSR, but be aware that a lower Minimum Smoothness threshold can add to the computation cost. If screen space reflection fails to affect a pixel, then HDRP falls back to using reflection probes.

Use the **Quality** dropdown to select a preset number of **Max Ray Steps**. Higher Max Ray Steps increase quality but come with a cost. As with all effects, balance performance with visual quality.

Note: You can choose between a physically based algorithm using accumulation or a less accurate approximate algorithm (default).



Material surfaces must exceed the Minimum Smoothness value to show Screen Space Reflections.

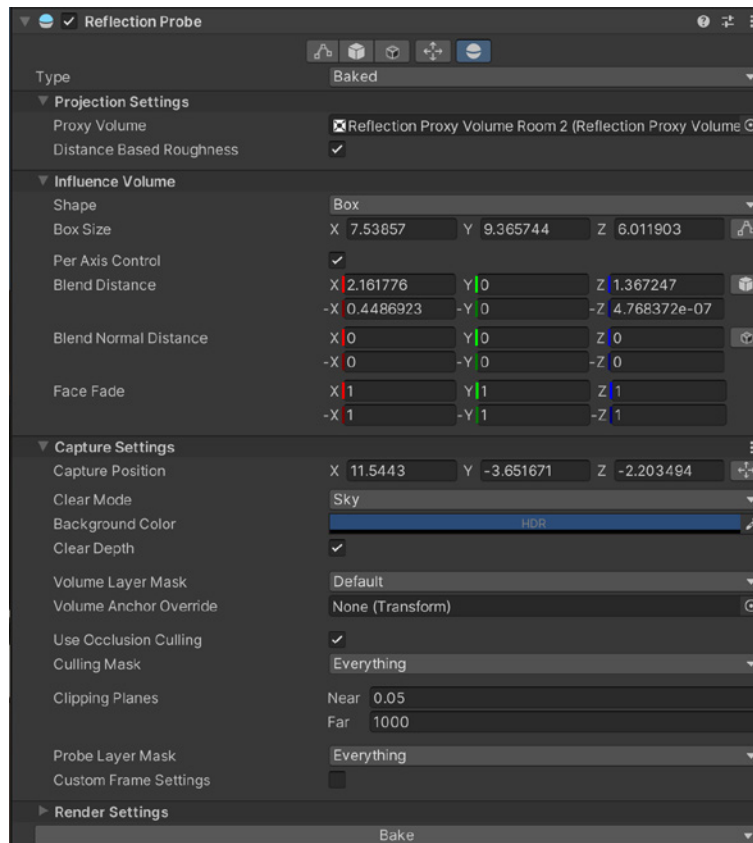
Reflection Probes

Reflection probes generate reflections using an image-based technique. A probe captures a spherical view of its surroundings in all directions and stores the result in a cubemap texture. A shader uses that cubemap to approximate a reflection.

Each Scene can have several probes and blend the results. Localized reflections then can change as your camera moves through the environment.

Set the **Type** of each probe to **Baked** or **Real-time**:

- *Baked* probes process the cubemap texture just once for a static environment.
- *Real-time* probes create the cubemap at runtime in the Player rather than in the Editor. This means that reflections are not limited to static objects, but be aware that real-time updates can be resource-intensive.



Reflection Probe component

Optimization tips

To optimize real-time reflection probes, disable any rendering feature which is not significantly affecting the visual quality of reflections by overriding general or per reflection probe camera settings. You can also create a script to time slice the update.

Unity 2022 LTS adds the **Time Slicing** property to the Reflection Probe component. To see this property, set the reflection probe's **Type** to **Realtime**. Use this to update the Cubemap and perform convolution across several frames (one face per frame instead of updating the whole probe at once). This can save performance for real-time reflection probes.

HDRP 14 replaces the cube reflection probe cache array with a **2D texture atlas cache** in octahedral projection. This allows you to control the resolution for each reflection probe to save memory.

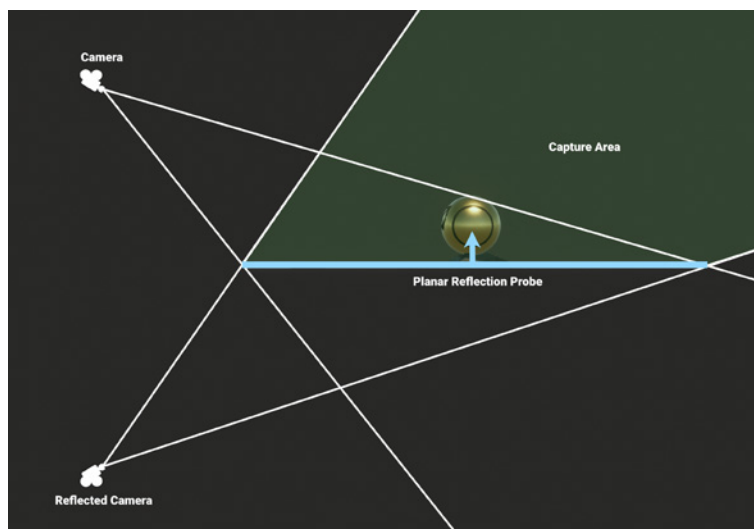
The **Influence Volume** determines the 3D boundaries where GameObjects will receive the reflection, while the **Capture Settings** let you customize how the Reflection Probe takes a snapshot of the cubemap.

Planar Reflection Probe component

The [Planar Reflection Probe](#) component allows you to recreate a flat, reflective surface, taking surface smoothness into account. This is perfect for a shiny mirror or floor.

Though a planar reflection probe shares much in common with a standard reflection probe, it does work slightly differently. Rather than capture the environment as a cubemap, it recreates the view of a camera reflected through the probe's mirror plane.

The probe then stores the resulting mirror image in a 2D RenderTexture. Drawn to the boundaries of the rectangular probe, this creates a planar reflection.



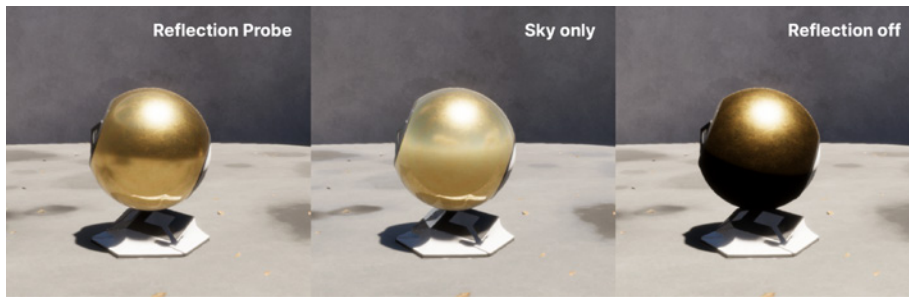
A planar reflection probe captures a mirror image by reflecting a camera through a plane.



A planar reflection probe creates a reflection for a flat object.

Sky reflection

When an object is not affected by a nearby reflection probe, it will fallback to the sky reflection.



A Reflection Probe shows the surrounding room whereas a sky reflection reflects the Gradient Sky.

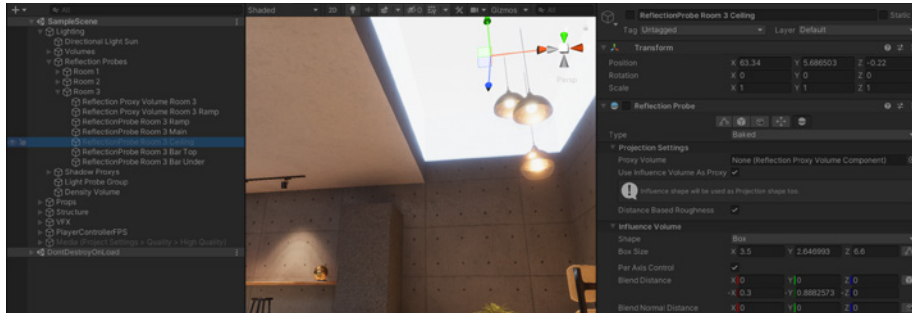
i Reflection hierarchy

To produce the highest-quality reflections, HDRP uses the technique that gives the best accuracy for each pixel and allows it to blend with the other techniques. HDRP checks the three reflection methods (SSR, reflection probes, sky) using a weighted priority. This specific sequence for evaluating reflections is called the [Reflection hierarchy](#).

When one technique does not fully determine the reflection at a pixel, HDRP falls back to the next technique. In other words, screen space reflection falls back to the reflection probes, which in turn fall back to sky reflections.

It's important to set up your Influence Volumes for your reflection probes properly. Otherwise, you could experience light leaking from unwanted sky reflections.

This is apparent in Room 3 of the Sample Scene. Disabling one of the reflection probes or shifting its Influence Volume forces the reflection to fall back to the sky. This causes the bright HDRI sky to overpower the scene with its intense reflections.



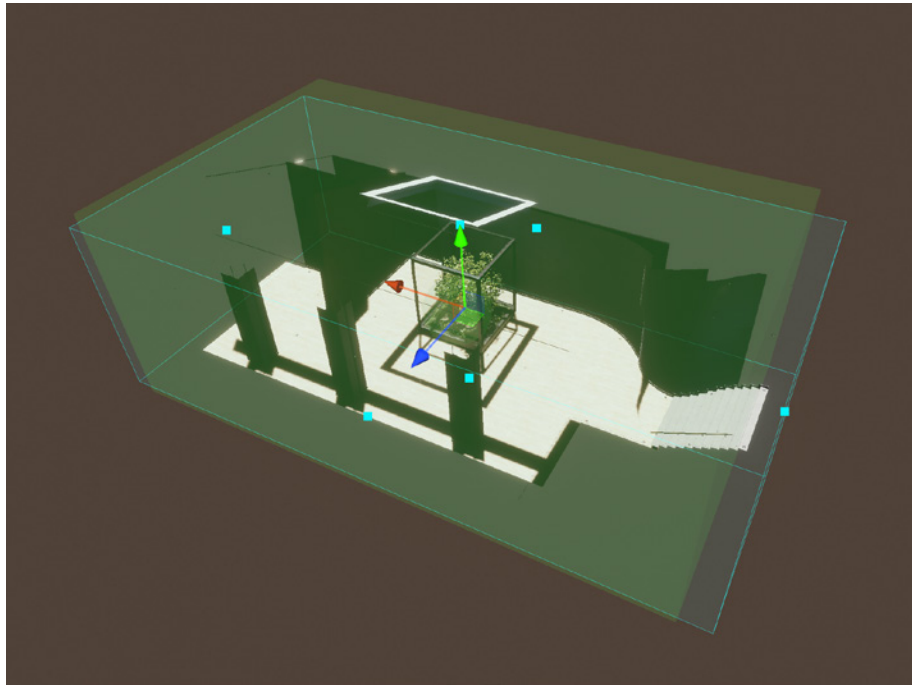
Disabling the Reflection Probe on the Room 3 ceiling causes unwanted light leaking.

For more details about determining Reflection Hierarchy, see the [Reflection in the HDRP](#) documentation page.

Reflection Proxy Volumes

Because the capture point of a Reflection Probe is fixed and rarely matches the Camera position near the Reflection Probe, there may be a noticeable perspective shift in the resulting reflection. As a result, the reflection might not look connected to the environment.

A Reflection Proxy Volume helps you partially correct this. It reprojects the reflections more accurately within the proxy Volume, based on the Camera position.



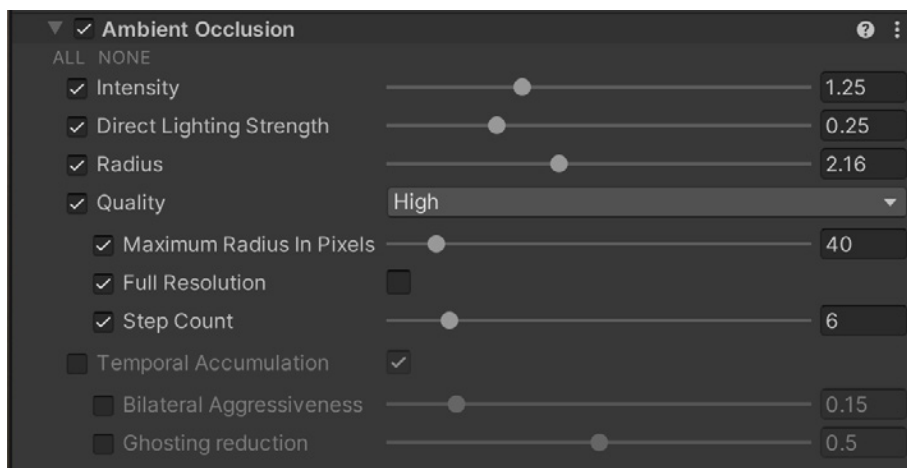
A Reflection Proxy Volume reprojects the cubemap to match the room's world space position.

Real-time lighting effects

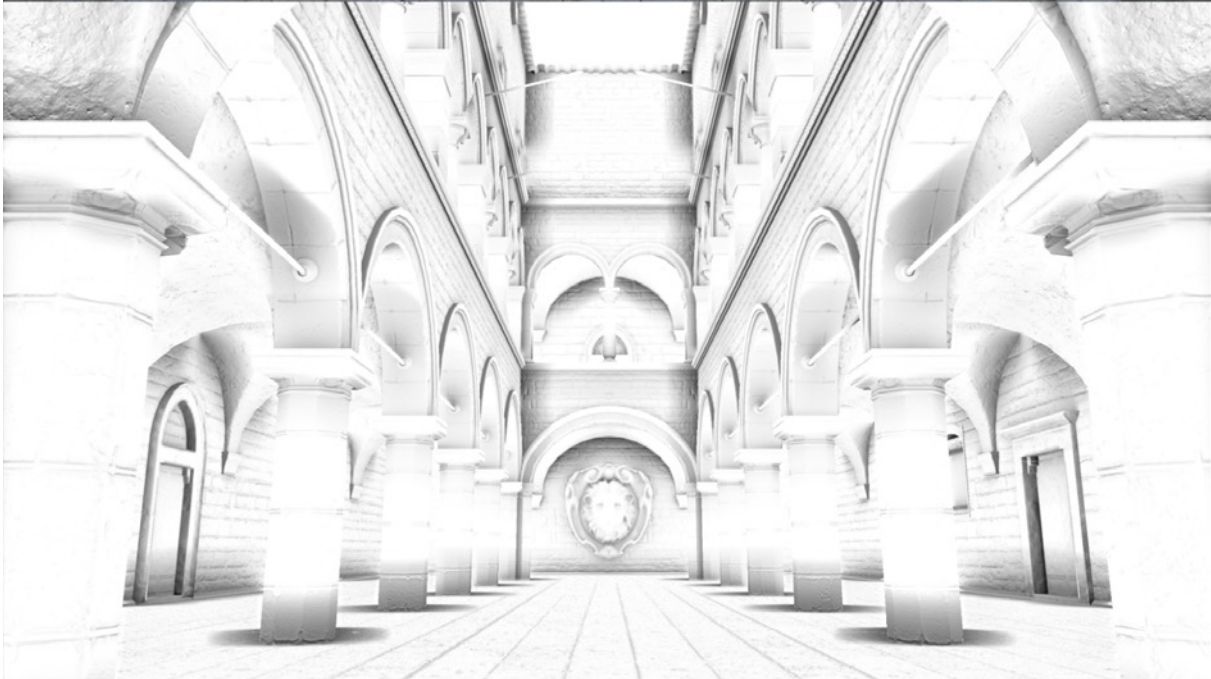
HDRP also features some real-time lighting effects available through the Volume system. Select a local or global Volume, then add the appropriate effect under **Add Override > Lighting**.

Screen Space Ambient Occlusion

Ambient occlusion simulates darkening that occurs in creases, holes, and surfaces that are close to one another. Areas that block out ambient light appear occluded.



Ambient Occlusion override



Screen Space Ambient Occlusion visualization in the Sponza Atrium

Though you can bake ambient occlusion for static geometry through Unity's Lightmapper, HDRP adds an additional **Screen Space Ambient Occlusion**, which works in real-time. Because this is a screen space effect, only information from within the frame can contribute to the effect produced. SSAO ignores all objects outside of the camera's field of view.

Enable Screen Space **Ambient Occlusion** in the **Frame Settings** under **Lighting**. Then, **Add Override** on a local or global Volume and select **Lighting > Ambient Occlusion**.

Screen Space Refraction

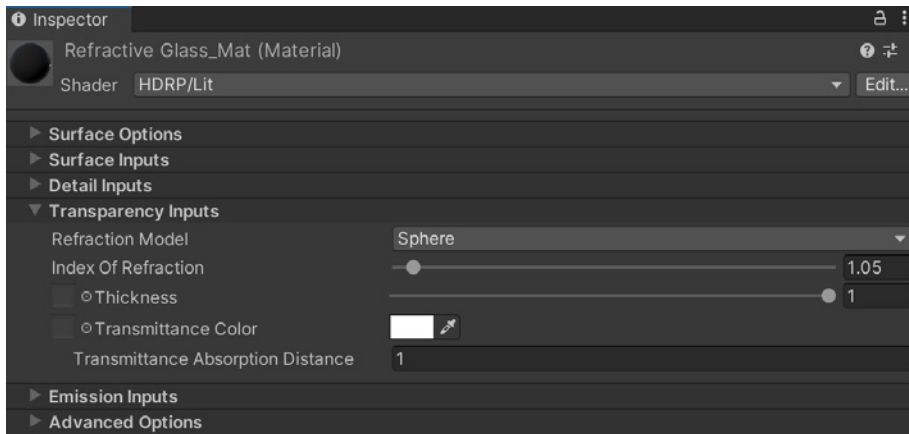
The [Screen Space Refraction](#) override helps to simulate how light behaves when passing through a denser medium than air. HDRP's Screen Space Refraction uses the depth and color buffer to calculate refraction through a transparent material like glass.



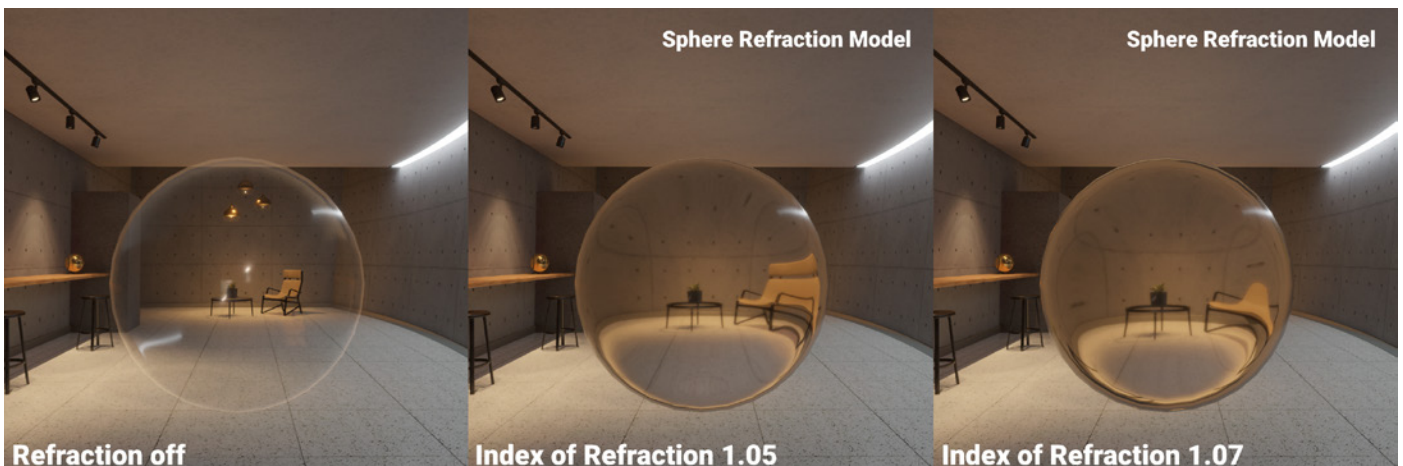
Screen Space Refraction override

To enable this effect through the **HDRP/Lit shader**, make sure your material has a Surface Type of **Transparent**.

Then choose a [Refraction Model](#) and [Index of Refraction](#) under **Transparency Inputs**. Use the **Sphere** Refraction Model for solid objects. Choose **Thin** (like a bubble) or **Box** (with some slight thickness) for hollow objects.



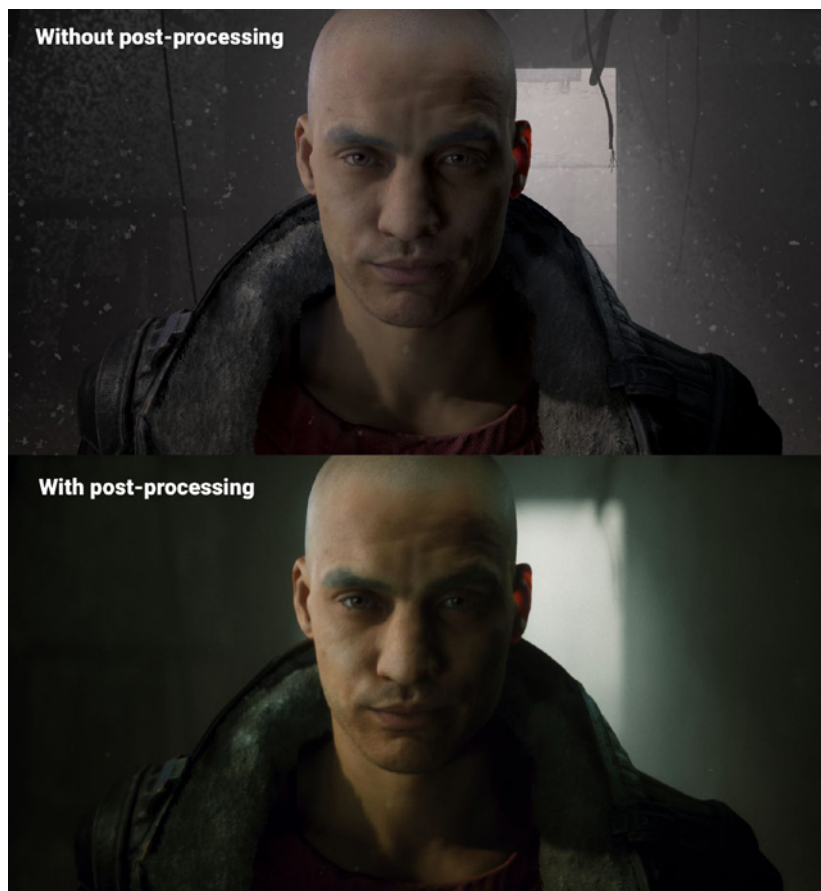
Transparency Inputs to control refraction



Screen Space Refraction

Post-processing

Modern high-end graphics would be incomplete without post-processing. While we can't always "fix it in post," it's difficult to imagine our rendered images without the filters and full-screen image effects that make them more cinematic. Thus, HDRP comes bundled with its own built-in post-processing effects.

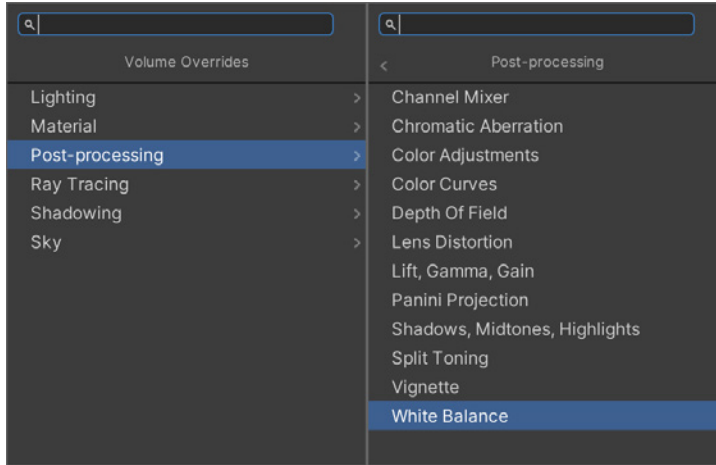


Post-processing effects make your renders more cinematic.

HDRP post-processing uses the Volume system to apply the image effects to the camera. Once you know how to add overrides, the process of applying more post effects should already be familiar.

Post-processing overrides

Many of these post-processing overrides for controlling color and contrast may overlap in functionality. Finding the right combinations of them may require some trial and error.



Post-processing overrides

You won't need *every* effect available. Just add the overrides necessary to create your desired look and ignore the rest.

Refer to the Volumes in the HDRP Sample Scene (available in the Unity Hub) for example usage.

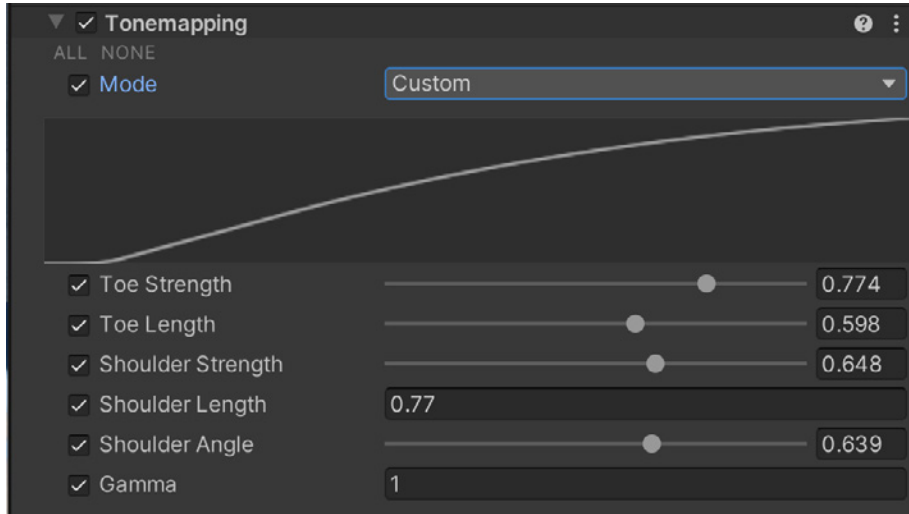
Tonemapping

Tonemapping is a technique for mapping [high dynamic range](#) colors to the more limited [dynamic range](#) of your screen. It can enhance the contrast and detail in your renders.



ACES vs Neutral tonemapping

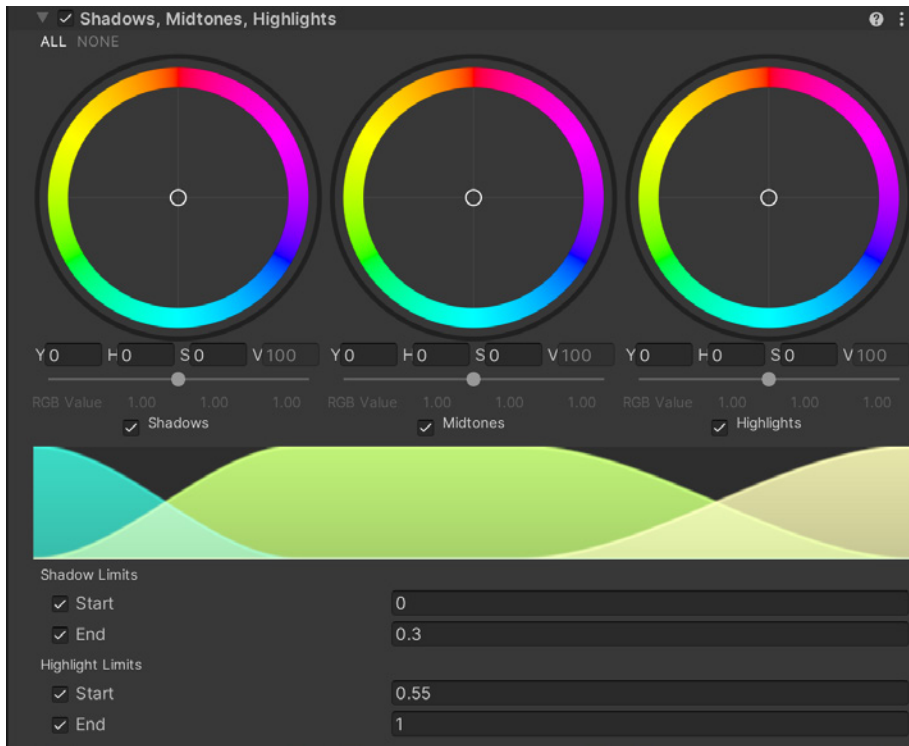
If you want a filmic look, set the **Mode** to the industry standard **ACES** ([Academy Color Encoding System](#)). For something less saturated and contrasted, select **Neutral**. Experienced users also have the option of choosing **Custom** and defining the tonemapping curve for themselves.



Tonemapping with a custom curve

Shadows, Midtones, Highlights

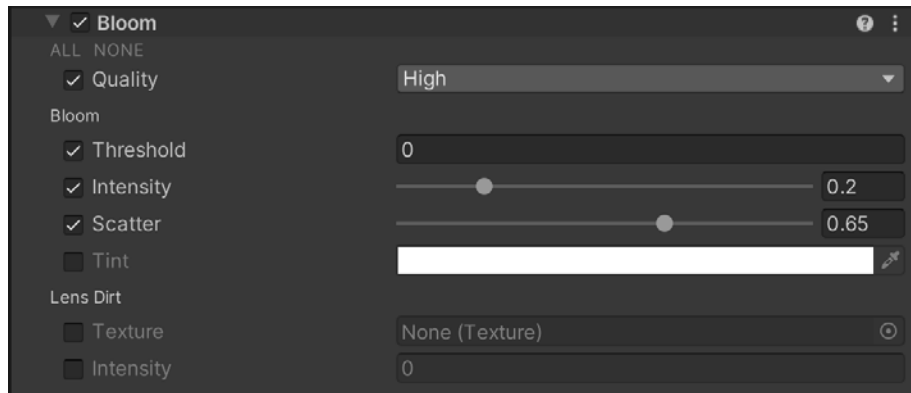
The Shadows, Midtones, Highlights override separately controls the tonal and color range for shadows, midtones, and highlights of the render. Activate each trackball to affect the respective part of the image. Then, use the **Shadow** and **Highlight Limits** to prevent clipping or pushing the color correction too far.



Shadows, Midtones, Highlights

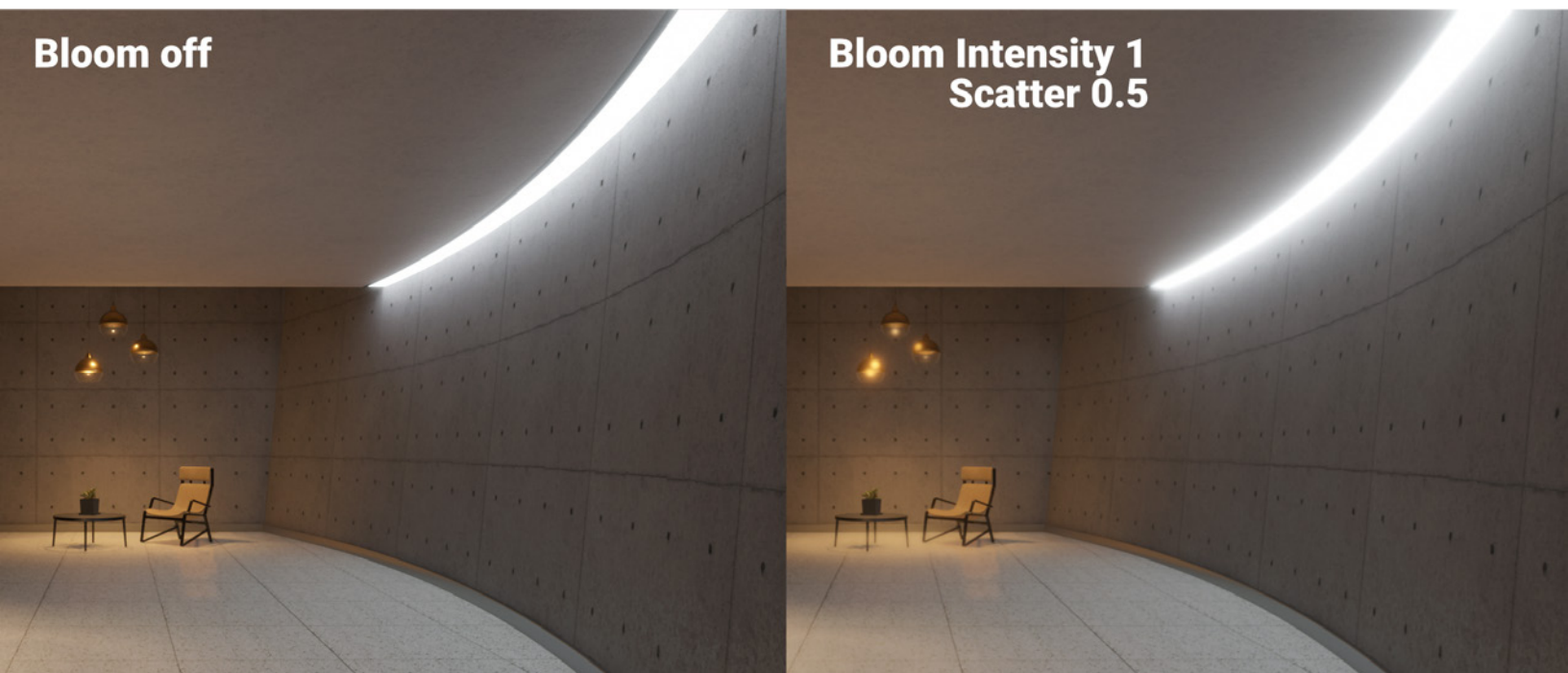
Bloom

Bloom creates the effect of light bleeding around the light source. This conveys the impression that the light source is intensely bright and overwhelming the camera.



Bloom override

Adjust the **Intensity** and **Scatter** to adjust the Bloom's size and brightness. **Lens Dirt** applies a texture of smudges or dust to diffract the Bloom effect. Use the **Threshold** to keep sharpness on non-bright pixels.



The effect of Bloom

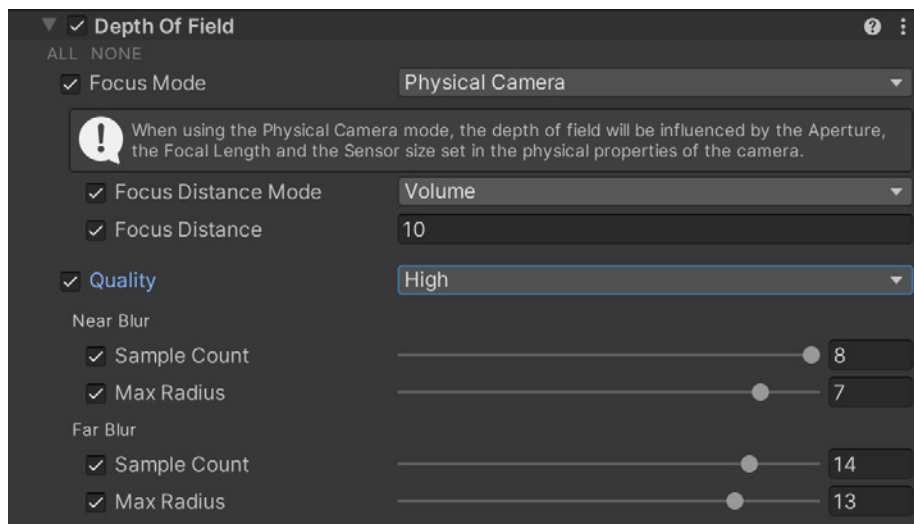
Depth of Field

Depth of Field simulates the focus properties of a real camera lens. Objects nearer or farther from the camera's focus distance appear to blur.

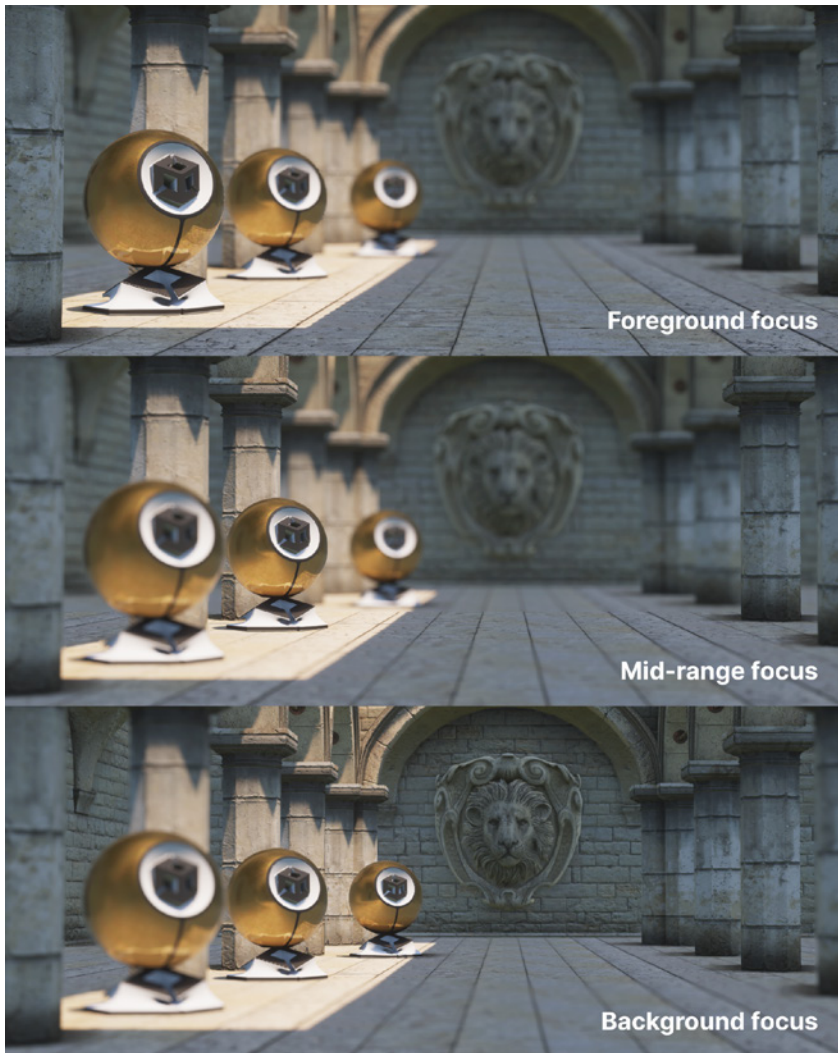
You can set focus distance from:

- The Volume Override with the **Manual Ranges** Focus Mode: Here, the Volume itself controls the focus distance. For example, you could use this to make your camera intentionally blurry based on location (e.g., underwater scene).
- Using a Cinemachine camera with the Volume Settings Extension: This lets you follow and autofocus on a target.
- The **Physical Camera** properties using **Physical Camera** Focus Mode: This allows you to animate the **Focus Distance** parameter from the Camera component.

When Depth of Field is active, an out-of-focus blur effect called a bokeh can appear around a bright area of the image. Modify the camera aperture's shape to change the appearance of the bokeh (see Additional Physical Camera Parameters above).

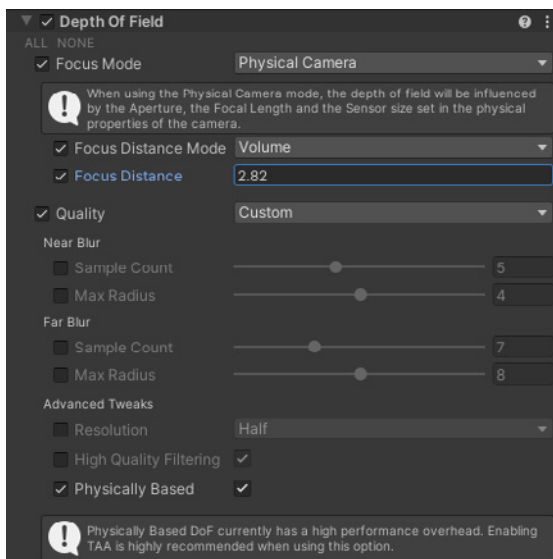


Depth of Field simulates the focus distance of real cameras.



Depth of Field simulates the focus distance of real cameras.

For cinematics or offline rendering, you can try choosing a more expensive but **Physically Based** depth of field by enabling Additional Settings and **Custom Quality**.



Enabling Custom Quality for depth of field

White Balance

The White Balance override adjusts a Scene's color so that the color white is correctly rendered in the final image. You could also push the **Temperature** to shift between yellow (warmer) and blue (cooler). **Tint** adjusts the color cast between green and magenta.

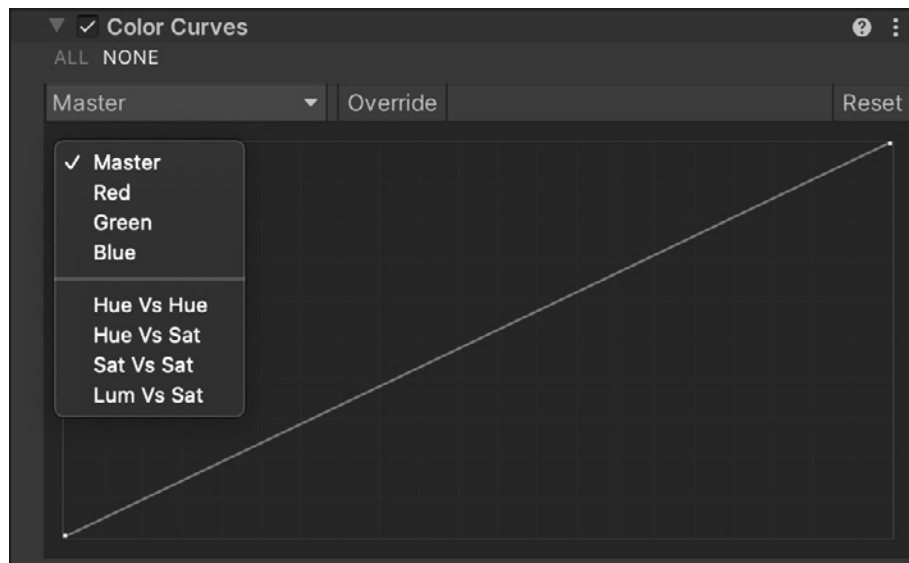
In the HDRP Sample Project, the local Volumes include White Balance overrides for each room.



White Balance

Color Curves

[Grading curves](#) allow you to adjust specific ranges in hue, saturation, or luminosity. Select one of the eight available graphs to remap your color and contrast.



Color Curves

Color Adjustments

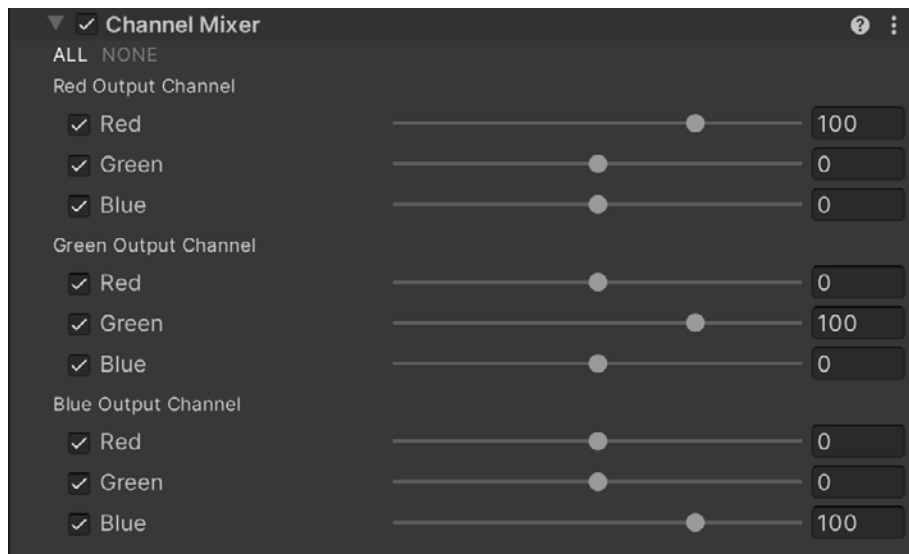
Use this effect to tweak the overall tone, brightness, hue, and contrast of the final rendered image.



Color Adjustments

Channel Mixer

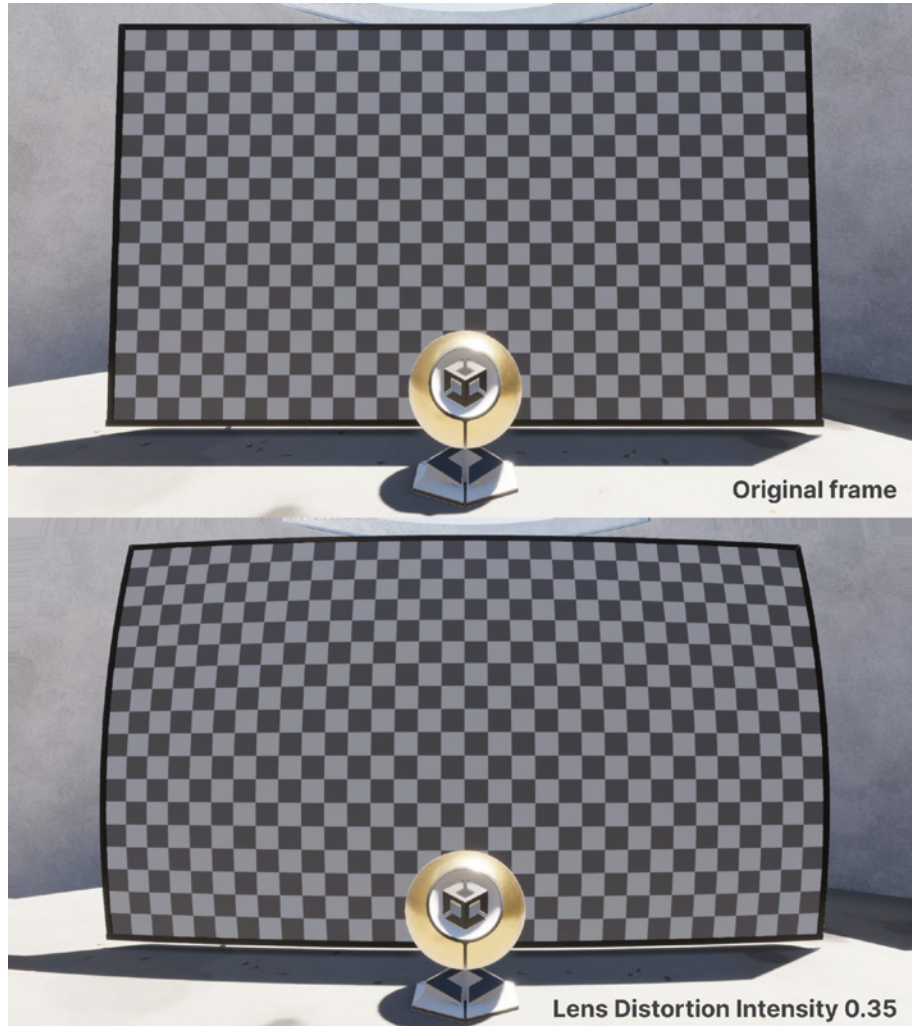
The Channel Mixer lets one color channel impact the “mix” of another. Select an RGB output, then adjust the influence of one of the inputs. For example, dialing up the Green influence in the Red Output Channel will tint all green areas of the image with a reddish hue.



Channel Mixer

Lens Distortion

Lens Distortion simulates radial patterns that arise from imperfections in the manufacture of real-world lenses. This results in straight lines appearing slightly bowed or bent, especially with zoom or wide-angle lenses.



Lens Distortion warps the image in a radial pattern.

This “fisheye” effect can help create a specific mood or style. For example, you can use it to emphasize a flashback cut sequence or to show an altered state of mind. This can also enhance a feeling of tension or disorientation.

Vignette

Vignette imitates an effect from practical photography, where the corners of the image darken and/or desaturate. This can occur with wide-angle lenses or result from an appliance (a lens hood or stacked filter rings) blocking the light. This effect can also be used to draw the attention of the viewer to the center of the screen.



A Vignette darkens the edges of the frame.

Motion Blur

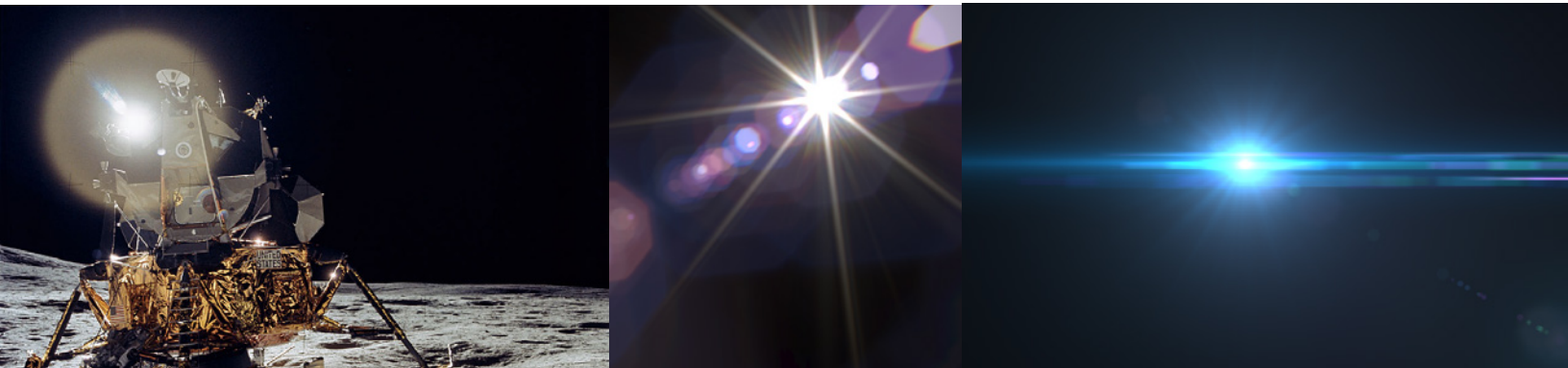
Real-world objects appear to streak or blur in a resulting image when they move faster than the camera exposure time. The Motion Blur override simulates that effect.

To minimize performance cost, reduce the Sample Count, increase the Minimum Velocity, and decrease the Maximum Velocity. You can also reduce the Camera Clamp Mode parameters under the Additional Properties.

Lens Flare

A lens flare is an artifact that appears when bright light is shining onto a camera lens. A lens flare can appear as one bright glare or as numerous colored polygonal flares matching the aperture of the camera.

In real life, flares have an unwanted effect, but they can be useful for a narrative or artistic purpose. For example, a strong lens flare can be used to grab the player's attention or change the mood of a setting or scene.



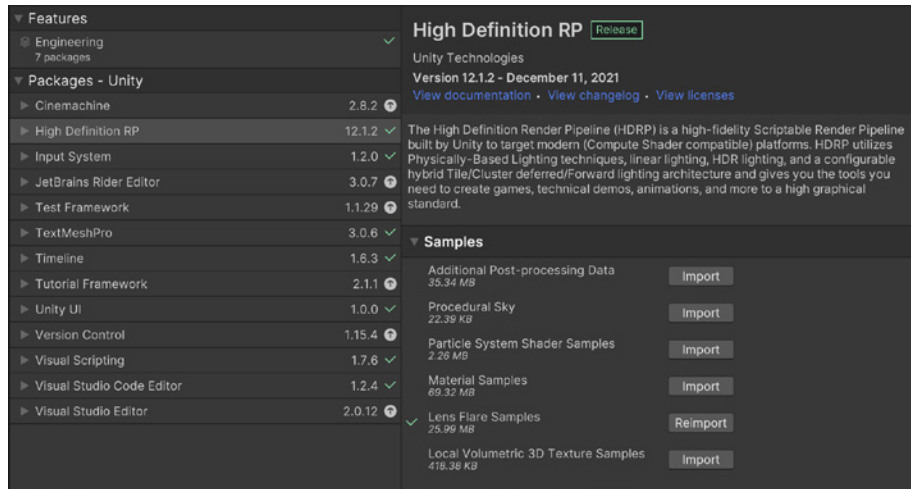
Lens flares from actual photographs. Learn more about the physical aspect of lens flares in this [Wikipedia article](#)

Lens flares are essentially a post-processing effect since they are rendered in the later stages of the rendering process.

The appearance of the flare is indicated in the [Flare asset](#), but in order to render the effect, you need to add the component [Lens Flare \(SRP\)](#) to an object in the Scene view. For instance, a light source or an object generating a flare.

The component controls the effect's general intensity, scale, and occlusion parameters. It also provides the option to run offscreen when the flare is outside the camera view, but it should still project some flare effects visible in the scene, for example when the flare comes from a static source.

The best way to get familiar with Lens Flares is by installing the samples from the Package Manager. This will add a set of predefined Flare assets and modify HDRP Sample Scene to include Lens Flare effects. It also contains a test scene to browse Lens Flares and help you build your own.



You can find Lens Flare samples in the Package Manager under High Definition RP

If you select the **Directional Light Sun** in the project, you can find the component Lens Flare (SRP) attached to the light, and a Lens Flare data asset. If you change the asset, you can observe different flare effects.

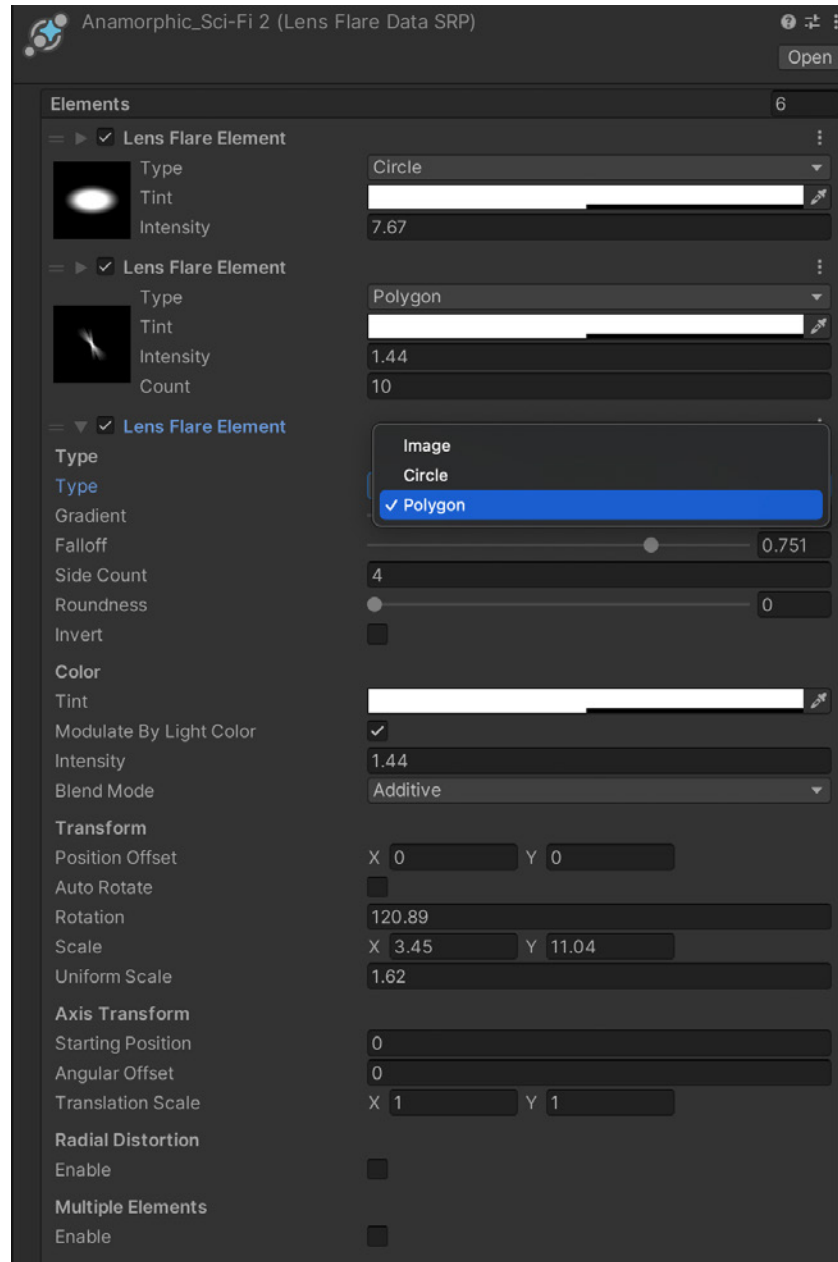


The Lens Flare samples come with presets ranging from realistic lens effects to more stylized looks.

Lens Flares are made out of Lens Flare Elements. Each element represents the different artifacts that the flare produces. The element's shape can be a polygon, circle, or a custom image.

The element parameters allow you to tweak the **Color**, **Transform** position, and deformation, or how it's positioned, colored, and scaled when the flare element is part of a Lens Flare effect with multiple elements. If it's attached to a light source, the flare elements can use the tint color, making it possible to reuse the same Flare asset with many different light sources.

Watch [this SIGGRAPH talk](#) to learn more about how Lens Flares work.



A Flare asset with several elements and their available parameters; you can also make your own library of custom flare effects to create the right atmosphere for your game.

Dynamic resolution

If you are not CPU-bound, resolution can strongly affect performance. Dynamic resolution reduces the rendering resolution and scales the result to the output screen resolution. Filters to perform this upscale allow you to render at 70% or lower resolution with little visual loss.

In Unity 2022 LTS, HDRP offers multiple choices with some of the latest super sampling technologies.

NVIDIA DLSS (for NVIDIA RTX GPU and Windows)

NVIDIA DLSS (Deep Learning Super Sampling) is natively supported for HDRP in Unity 2022 LTS. NVIDIA DLSS uses advanced AI rendering to produce image quality that's comparable to native resolution while only conventionally rendering a fraction of the pixels.

With real-time ray tracing and NVIDIA DLSS, you can create beautiful worlds running at higher frame rates and resolutions on NVIDIA RTX GPUs. DLSS also provides a substantial performance boost for traditional rasterized graphics. Learn more about it on NVIDIA's [Unity developer page](#), this [blog post](#), and the Unity [documentation](#).



Read this [blog](#) to learn more about the inner workings of the DLSS technology that allows games, such as *Nakara Bladepoint* by 24 Entertainment, to run at 4K.

AMD FSR (cross-platform)

AMD FidelityFX™ Super Resolution (FSR) includes built-in support for HDRP and URP. You can use FSR by enabling dynamic resolution in HDRP assets and cameras, then selecting **FidelityFX Super Resolution 1.0** under the Upscale filter option.

AMD FidelityFX Super Resolution (FSR) is an open source, high-quality solution for producing high-resolution frames from lower-resolution inputs. It uses a collection of algorithms with a particular emphasis on creating high-quality edges, giving performance improvements compared to rendering at native resolution directly.

FSR enables practical performance for costly render operations, such as hardware ray tracing. Learn more about it on AMD's Unity [web page](#) and on the [Unity forum](#).



Try FSR technology in the *Spaceship* demo, available on [Steam](#).

TAA Upscale (cross-platform)

Temporal Anti-Aliasing (TAA) Upscale uses temporal integration to produce a sharp image. Unity performs this method alongside the normal anti-aliasing. HDRP executes this upscale filter before post-processing and at the same time as TAA.

This means you can only use the TAA anti-aliasing method, since this filter is not compatible with other anti-aliasing methods. Temporal Anti-Aliasing (TAA) Upscale performs anti-aliasing on each frame, meaning it also runs when you enable dynamic resolution, even when the screen percentage is at 100% resolution. For more information, see the documentation section [“Notes on TAA Upscale.”](#)



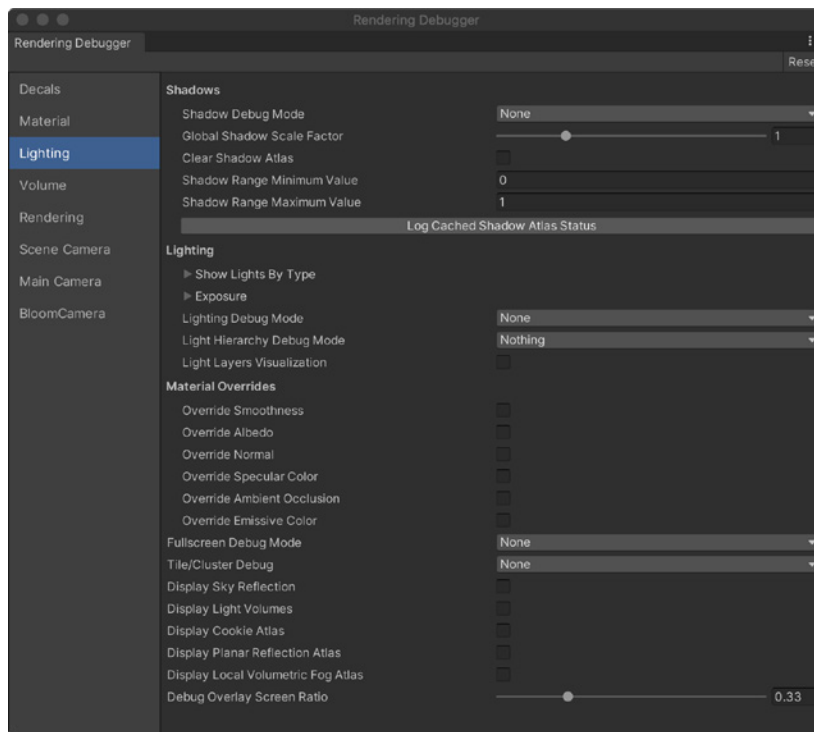
Image (A) to the left, using the Temporal Anti-Aliasing Upscale method, is sharp and well-defined. Image (B) to the right, which uses the Catmull-Rom upscale method, appears less-defined.

You can either force the scaling in the HDRP asset or code your logic to adjust the scale dynamically.

To set up dynamic resolution in your project and get guidance on how to choose the best algorithm for your needs, please [read the documentation](#).

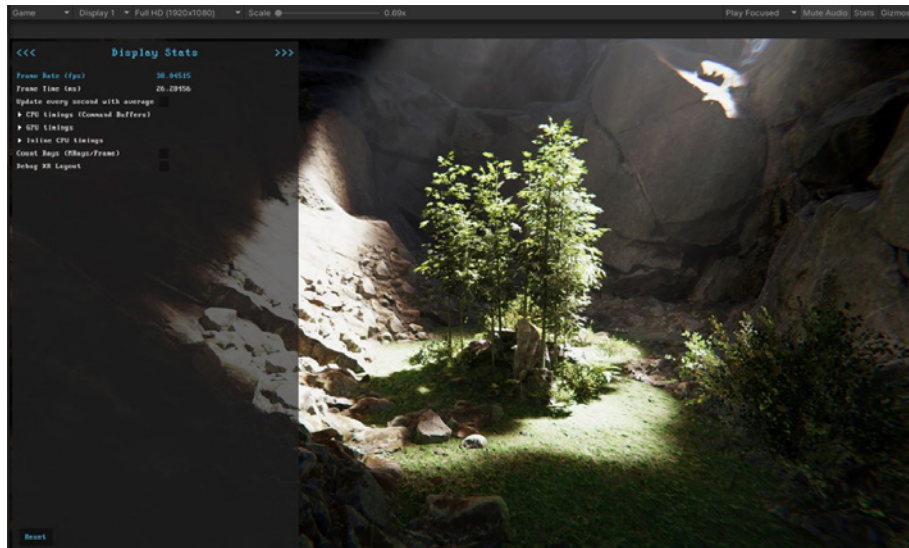
Rendering Debugger

The Rendering Debugger window (**Window > Analysis > Rendering Debugger**) contains debugging and visualization tools specific to the Scriptable Render Pipeline. The left side is organized by category. Each panel allows you to isolate issues with lighting, materials, volumes, cameras, and so on.



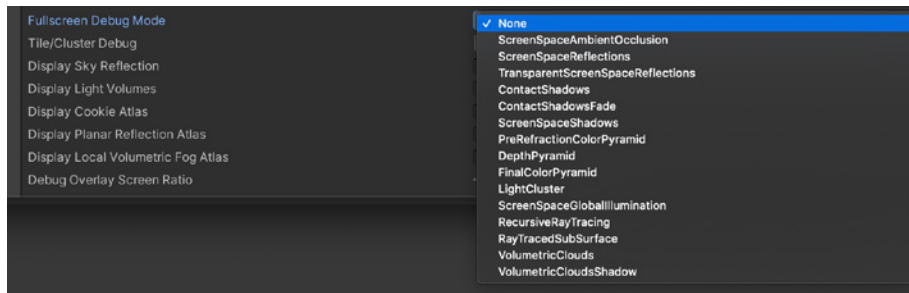
Rendering Debugger

The Rendering Debugger is also available at runtime in the Game view in Play mode or in a Player built in the Development build. You can open its menu by using **Ctrl+Backspace** or pressing the two sticks of a game controller.



Rendering Debugger window in Game view or Player

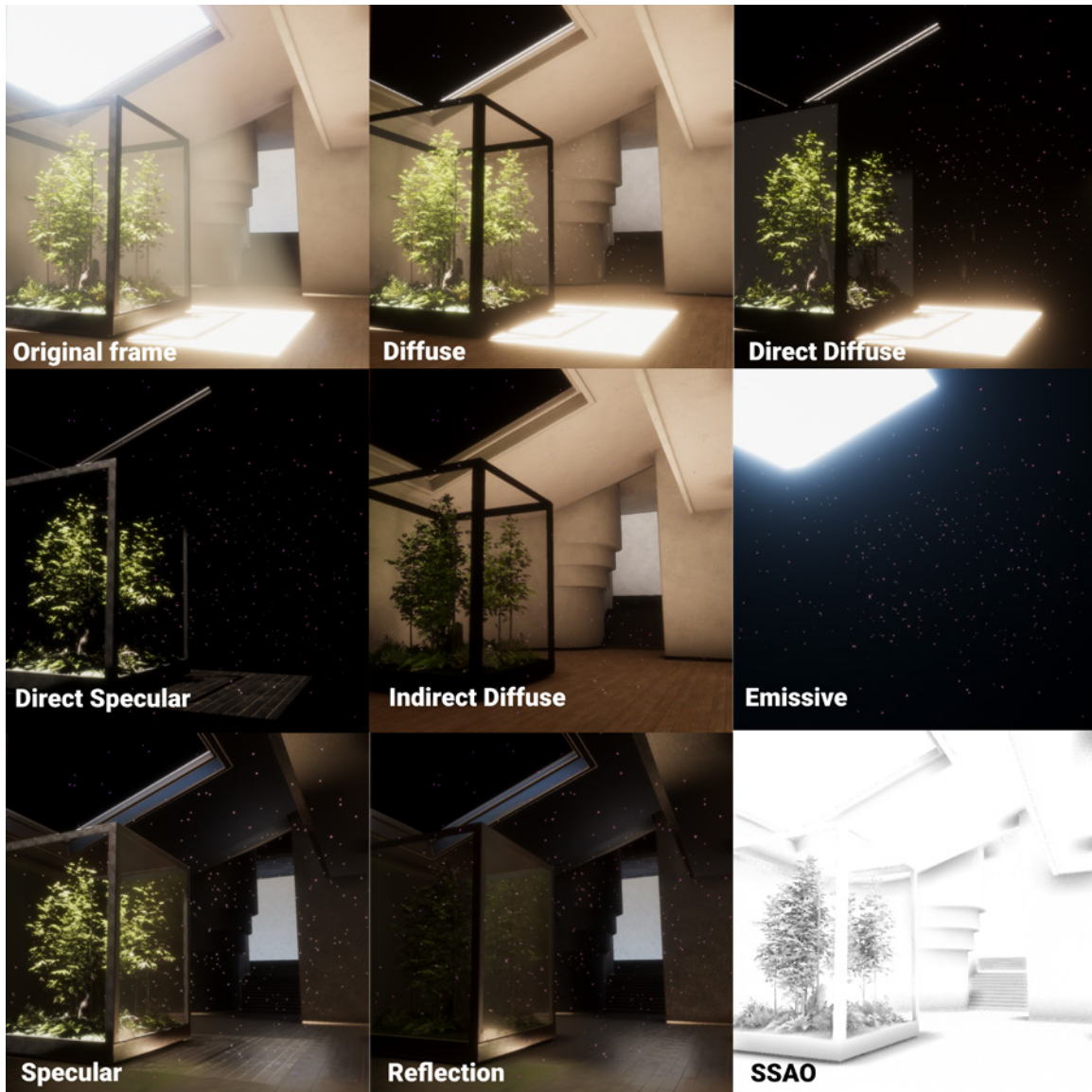
The Debugger can help you troubleshoot a specific rendering pass. On the Lighting panel, you can enter **Fullscreen Debug Mode** and choose features to debug.



Fullscreen Debug Mode options

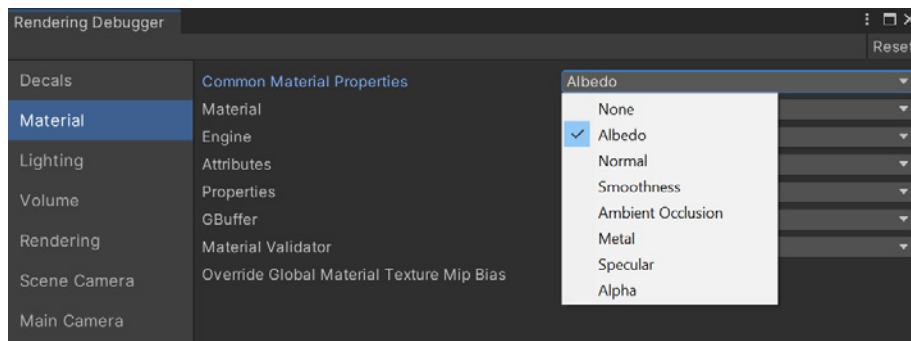
These Debug modes let you play “pixel detective” and identify the source of a specific lighting or shading issue. The panels on the left can show you vital statistics from your cameras, Materials, Volumes, and so on, to help optimize your render.

With the fullscreen Debug mode active, the Scene and Game views switch to a temporary visualization of a specific feature. This can serve as a useful diagnostic.

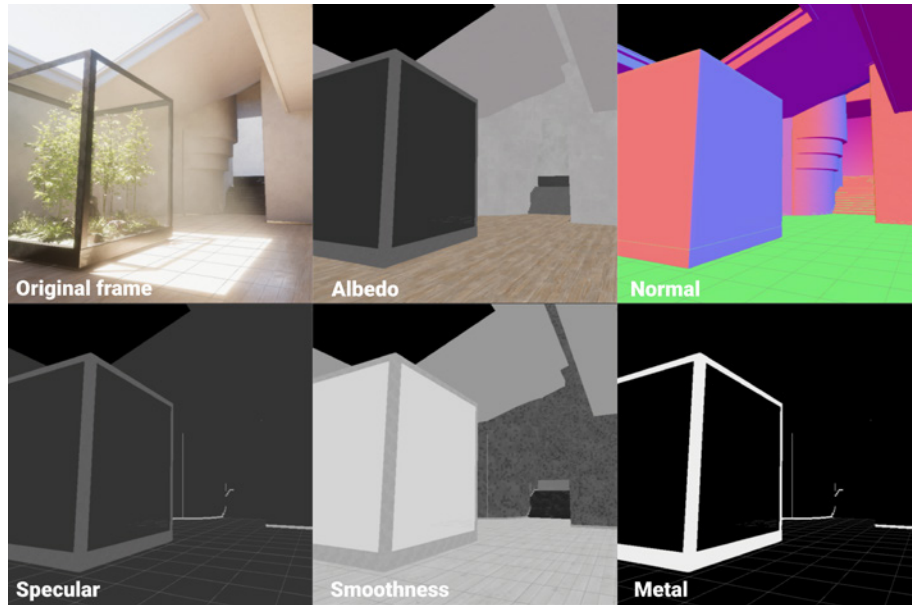


Lighting Debug Mode or Fullscreen Debug Mode can help you understand the sources of illumination in your scene.

You can also debug several common material properties. On the Material screen, select from the Common Material Properties: albedo, normal, smoothness, specular, and so on.



Common Material Properties



Use the Render Pipeline Debugger to troubleshoot materials.

Color monitors

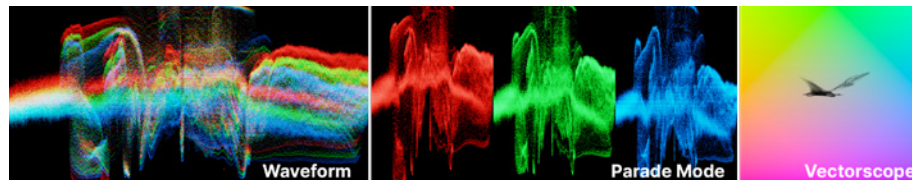
Unity 2022 LTS and above includes a set of industry-standard color monitors that you can use to control the overall look and exposure of a scene. You can find these monitors in the **Rendering Debugger** window (**Windows > Analysis > Rendering Debugger**) in the Rendering tab.

The **Waveform** monitor provides a graphical representation of the luminance (brightness) values of a rendered image.

Parade Mode splits the waveform image into red, green, and blue channels, making it easier to identify and correct color imbalances or discrepancies.

The **Vectorscope** monitor provides a circular graph that measures the hue (color) and saturation (intensity of color) of an image. This tool is particularly useful for ensuring skin tones are accurate and for identifying any dominant color cast.

Incorporating these monitors into your workflow can ensure that colors are accurate and balanced.



The color monitors.

Support for HDR10 screens

Unlike standard dynamic range displays, High Dynamic Range (HDR) displays can reproduce a broader spectrum of luminance levels, closer to what the human eye perceives in natural environments.

While HDRP has long been able to render high dynamic range images, it previously lacked a dedicated tonemapper to output specifically to HDR displays. This unoptimized output could have led to lost details in the brightest and darkest parts of an image.

With Unity 2022 LTS, HDRP has the option to enable HDR10 output, which uses 10 bits per color channel for a more extensive color palette and finer brightness levels. If you're watching your HDRP content on an HDR10 display, you'll enjoy more vibrant colors as well as improved details in both shadows and highlights.

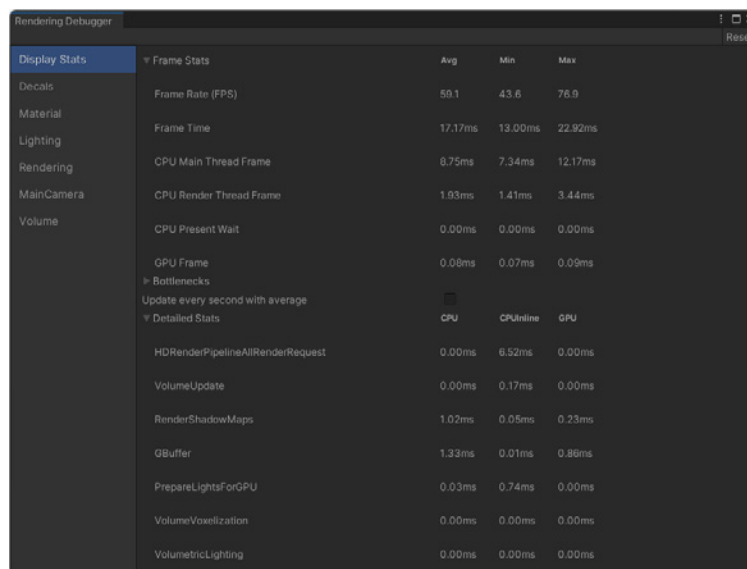
Runtime Frame Stats

SRPs have access to the Runtime Frame Stats panel in the Profiler. This tool is available both in Editor and players to easily get an idea of where most of the GPU time is spent.

This can help artists, technical artists, and developers learn more about per-frame performance with HDRP.

Note: Profiling in a player build instead of the Editor is recommended for accuracy.

Enable **Frame Timing Stats** in the Player Settings. Then, the [Display Stats](#) screen will be accessible in Play mode from the **Rendering Debugger (Window > Analysis > Rendering Debugger)** in Editor, or **Ctrl + Backspace** in Player).



Display Stats		Avg	Min	Max
Decals	Frame Rate (FPS)	59.1	43.6	76.9
Material	Frame Time	17.17ms	13.00ms	22.92ms
Lighting	CPU Main Thread Frame	8.75ms	7.34ms	12.17ms
Rendering	CPU Render Thread Frame	1.93ms	1.41ms	3.44ms
MainCamera	CPU Present Wait	0.00ms	0.00ms	0.00ms
Volume	GPU Frame	0.08ms	0.07ms	0.09ms
Bottlenecks				
Update every second with average				
Detailed Stats		CPU	CPUinline	GPU
	HDRRenderPipelineAllRenderRequest	0.00ms	6.52ms	0.00ms
	VolumeUpdate	0.00ms	0.17ms	0.00ms
	RenderShadowMaps	1.02ms	0.05ms	0.23ms
	GBuffer	1.33ms	0.01ms	0.86ms
	PrepareLightsForGPU	0.03ms	0.74ms	0.00ms
	VolumeVoxelization	0.00ms	0.00ms	0.00ms
	VolumetricLighting	0.00ms	0.00ms	0.00ms

The Runtime Frame Stats when in Play mode

See the [Render Pipeline Debugger](#) documentation for complete details.

Shaders and Materials

Materials play a crucial role in rendering by determining how an object reflects or emits light. They can make an object look like metal, glass, wood, or even something abstract or magical.

A [material](#) contains a reference to a [shader object](#). If that shader object defines [material properties](#), then the material can also contain data, such as colors or texture references.

The shader itself is a program that runs on the GPU to determine the color of each pixel. It performs its calculations based on input data, which might include textures, lighting information, and material properties.

Materials samples



HDRP includes a Materials sample in the Package Manager.

The [HDRP Materials sample](#) includes various examples of materials and shaders specific to HDRP. The samples showcase effects such as subsurface scattering, displacement, and anisotropy. The Shader Graphs make use of the [Master Stacks](#) (see below) like the [Lit Shader Stack](#), [Fabric Master Stack](#), [Decal Master Stack](#), and others.

Note that some materials require a GPU that supports [ray tracing](#) in order to visualize properly.

Material Variants

Ever wondered how to efficiently manage and apply changes to complex material libraries in Unity? With Material Variants, you can create templates or material prefabs based on regular materials.

Variants can share common properties with the template material and then override just what's needed. Much like Prefabs work for GameObjects, Material Variants allow you to lock certain properties if you don't want them overridden.

For example, if you wanted to set up several wood materials in your project, you could start with a Wood Base material, and make variations like Oak or Cherry with different textures. Either of those, in turn, could have its own Material Variants (e.g., White Oak or Red Oak) that override some other property like the Base Color tint.



An example of Material Variants.

Material Variants allow you to create a versatile library of base materials for use across projects, with the ability to override as needed. Thus, artists can avoid duplicating materials for minor tweaks, reducing visual bugs and performance issues, especially in larger projects.

See [this blog post](#) and [documentation page](#) for more details about Material Variants.

Material properties

Many of your scene's materials will use the [HDRP Lit Shader](#). Here are some of its most important properties, which may also appear in other HDRP shaders as well.

Base Color: Controls both the color and opacity of your material

Metallic: Determines how “metal-like” your surface appears; a higher value creates a more metallic sheen, while a lower value gives a more non-metallic (or dielectric) appearance

Specular: Determines the reflectivity of non-metal materials; for metallic materials, the specular highlight color is derived from the base color

Smoothness: Determines the clarity of reflections on the material; higher smoothness results in shinier, smoother surfaces, while lower values result in more matte, rougher materials

Anisotropy: Creates a visual effect where the highlight of the material is elongated in one direction (e.g., carbon fiber or brushed metal)



Anisotropy elongates the highlight in one direction.

Thickness: Typically used with subsurface scattering (SSS) to give the impression of light penetrating the surface of a translucent material and scattering within

Emission: Allows materials to “glow,” which is useful for objects that function as sources of illumination (e.g., neon lights)

Normal Map: A special texture type that allows you to add detailed surface characteristics such as bumps and grooves

Detail Map: Allows for additional texture layering on top of the base map, granting more precise control over how textures look up close

Iridescence: Describes a change in color as the viewing angle changes (e.g., seen in soap bubbles or certain gemstones)



Iridescent objects change color with the viewing angle.

Transparency

Transparency is the quality that allows light to pass through it, commonly seen with materials like glass or clear plastic.

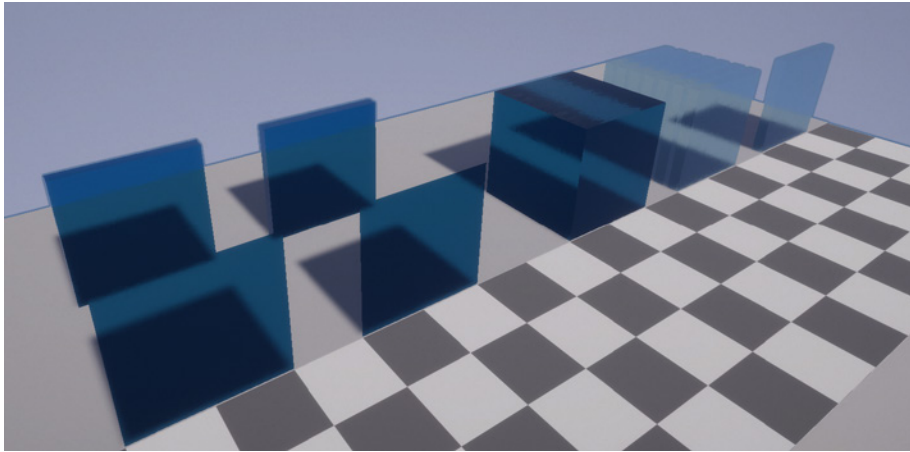
In HDRP, transparency is handled by adjusting the alpha value of the base color of a material. A lower alpha value makes a material more transparent, while an alpha value of 1 means the material is fully opaque.



Transparent materials allow light to pass through.

HDRP supports two transparency modes: Alpha and Premultiply. Alpha uses the alpha value directly from the texture, while Premultiply multiplies the color with the alpha value before applying it.

The Transparency scenes in the Material samples contain examples that demonstrate how to set up transparent materials using different rendering methods (Rasterization, Ray Tracing, Path Tracing).



The Transparency samples demonstrate how to set up different rendering methods.

Subsurface scattering and translucency

Subsurface scattering (SSS) is a phenomenon associated with translucent materials. Translucency is a special form of transparency where light can pass through a substance but is scattered in the process.

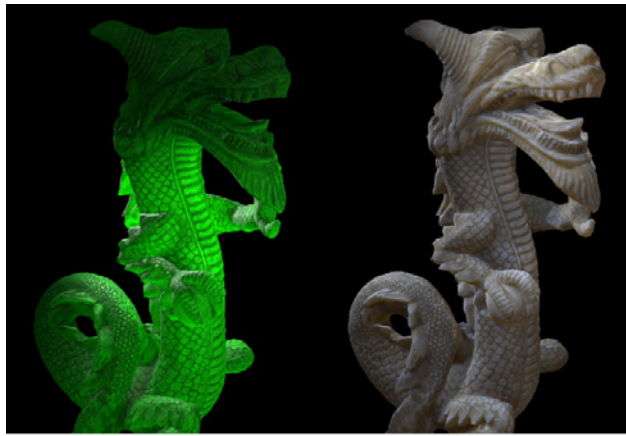
This effect is responsible for the soft glow of light that passes through certain organic materials, like skin or foliage, that makes them look smooth rather than rough and plastic-like.

When light hits a translucent material, not all of it reflects off the surface. A portion penetrates the interior and scatters within the material, bouncing around internally. Some scattered light eventually exits the material at a different point from where it entered.

HDRP implements subsurface scattering using a screen-space blur technique. It also handles transmission, when light penetrates GameObjects from behind to make them appear transparent. Transmission can also color the light as it passes through.

There are two material types for SSS in HDRP:

- **Subsurface scattering** implements both the screen-space blur effect and transmission.
- **Translucent** only accounts for transmission.



Translucency

Subsurface scattering

The Transparency samples demonstrate how to set up different rendering methods.

Enable subsurface scattering in the HDRP Asset and in the **Project Settings > HDRP Default Settings**. Then, create a [Diffusion Profile](#) Asset that stores the SSS settings. HDRP supports up to 15 custom profiles in view simultaneously, with override options.

Set each Material Type to **Subsurface Scattering** or **Translucent** as necessary and assign a Diffusion Profile.



SSS materials, like the drapes in this scene, use a Diffusion Profile.

The Diffusion Profile controls properties like the color of the scattered light, transmission tint, thickness, etc.

In each material, you also have two other options that affect the SSS:

- A **Thickness Map** controls the scattering and transmission effects across different parts of the object.
- A **Transmission Mask** controls the overall strength of the transmission e.g., a tree could use one shader for both its trunk and leaves.



Foliage renders more realistically with subsurface scattering.

Note: In Unity 2022 LTS, the [SpeedTree 8 Sub Graph Assets](#) (HDRP/Nature/SpeedTree8.shadergraph) now uses its Subsurface Map for the Transmission Mask node to remove the unintended light transmission from tree barks and twigs. This also fixes the overly bright billboard lighting which didn't match the 3D geometry's lighting.

Decals

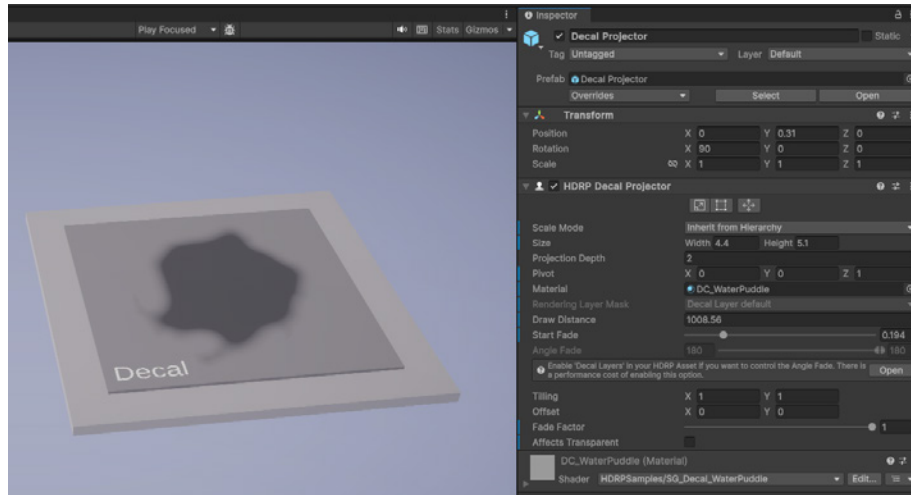
Decals are materials that use the [Decal Shader](#) or [Decal Master Stack](#). They add detail to materials and surfaces as graphical overlays without adding additional geometry.

Decals are essentially textures projected onto other objects in the scene, either as flat images (projected decals) or conforming to the shape of a 3D object (mesh decals).

A [Decal Projector](#) component is your primary tool for creating decals. Use this to control the size, depth, and other properties of the decal.

Use the **Start Fade** and **Draw Distance** to determine the decal's visibility by distance. Enable **Angle Fade** to fade by the decal orientation to the vertex normal.

Use decals to layer graffiti on top of city walls, create wear and tear on floors and machinery, add patterns and emblems to vehicles and objects, and more.

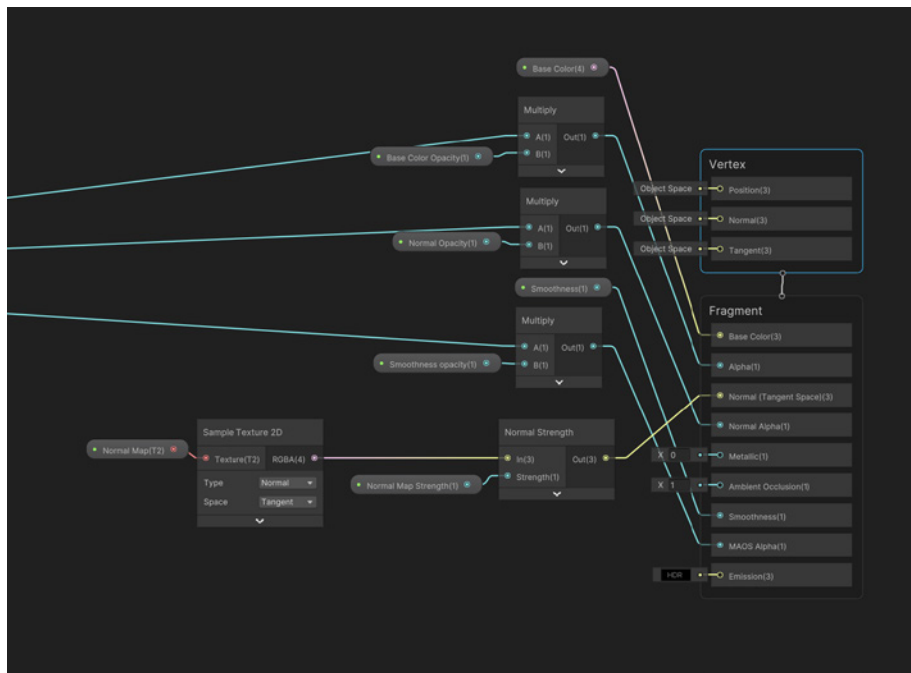


Decals are a method of adding details.

Shader Graph

Shader Graph is a powerful visual tool that enables you to build shaders visually rather than by writing code. It provides a node-based interface for creating complex shaders without the need to write, debug, and maintain shader code in a language like HLSL.

By connecting Shader Graph nodes, developers can visually construct complex shaders, making the process more accessible for those who may not be proficient in shader programming. Each node in the Shader Graph represents a mathematical operation, a function, or a shader property.



Build shaders graphically with the node-based Shader Graph.

Shader Graph is fully integrated with HDRP and provides a set of nodes and features specifically designed for it. This includes support for HDRP's physically based lighting model, advanced material types, and post-processing effects.

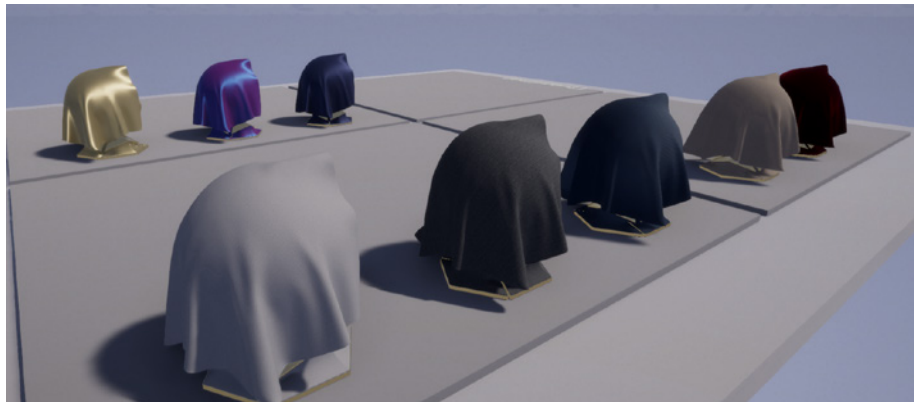
HDRP prioritizes high-definition visuals, so it includes a variety of supported shaders that can help you produce more realistic physically based materials. Each HDRP material type corresponds to a [Master Stack](#) in Shader Graph.

HDRP Master Stacks

The Master Stack is the end point of a Shader Graph that defines the final surface appearance of a shader. Your Shader Graph only contains one Master Stack.

The Master Stacks in HDRP include:

- **Lit Master Stack:** This creates general-purpose materials with physically based lighting. This is the most common Master Stack for realistic materials and supports a wide range of features, including subsurface scattering, translucency, and anisotropy.
- **StackLit Master Stack:** This is designed for complex materials with multiple layers, allowing for detailed control over how light interacts with each layer. For example, you would use a StackLit Master Stack for a car paint material that needs a base layer, a metallic flake layer, and a clear coat.
- **Unlit Master Stack:** This is ideal for materials that don't interact with lighting, providing full control over color and texture without considering light and shadow.
- **Decal Master Stack:** This adds graphical overlays onto surfaces. Use the resulting shader for adding details like dirt, scratches, or graffiti to a surface.
- **Water Master Stack:** This shader specializes in simulating water surfaces, including reflections, refractions, and other water-related visual effects. See the [Water System section](#) for more information.
- **Fabric Master Stack:** This simulates fabric materials. The corresponding shader can render fabric materials and includes options for sheen color and intensity.



Fabric materials have options for sheen color and intensity.

- **Eye Master Stack:** This helps render realistic eyes, including the cornea, sclera, and iris, with detailed control over reflections and refractions.
- **Hair Master Stack:** This simulates hair and fur, with control over color, specular highlights, and how light scatters through individual strands.



The *Enemies* demo features a digital human character.

See the [Enemies demo project](#) to see the Eye Master and Hair Master stacks in action.

Some Shader Graph nodes also offer HDRP-specific functionality. These can include Diffusion Profile, Emission, Exposure, HD Scene Color, HD Scene Depth, and so on. The Eye Master Stack or Water Master Stack, in particular, have many nodes specific to their stacks. See the complete list of [HDRP-specific nodes](#) in the documentation.

Volumetric Shader Graph fog

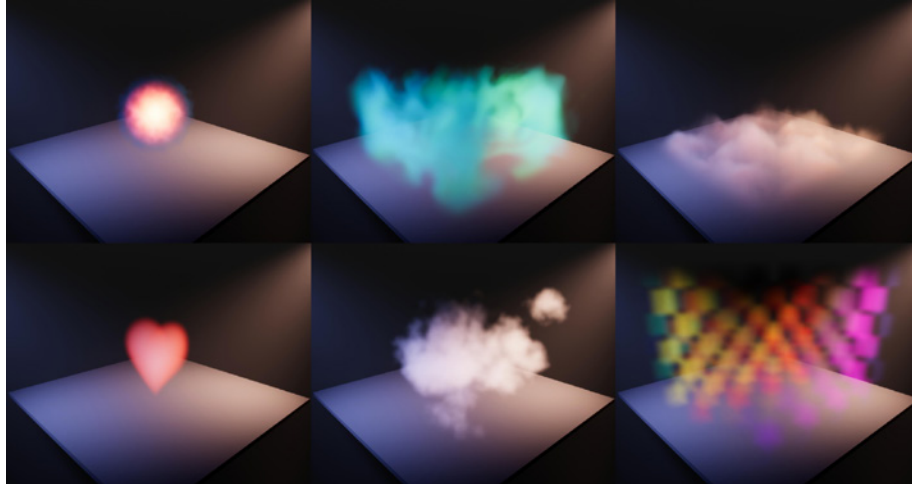
Create advanced procedural fog and volumetric effects with Volumetric Materials. These use a combination of a Shader Graph applied to any Local Volumetric Fog component.



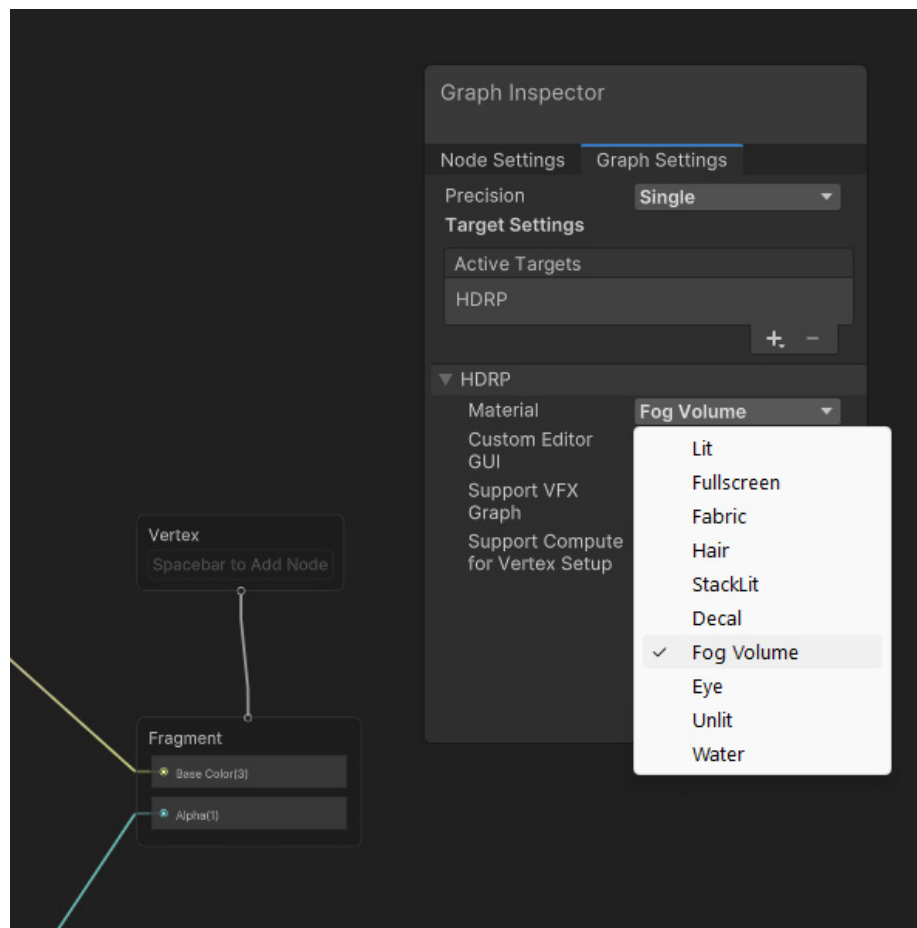
An example of volumetric procedural fog.

Use this to create custom ground fog or cloud effects. You can also create other natural phenomena for your scenes like sand storms, aurora borealis, and more.

Import the **HDRP Volumetrics samples** from the Package Manager for more examples. Here, the Shader Graph uses a **Fog Volume** as the HDRP material to add details and animation to the Local Volumetric Fog component.



The HDRP Volumetric samples collection is available in the Package Manager.



Select Fog Volume as the Shader Graph material.

Fullscreen Shader Graphs

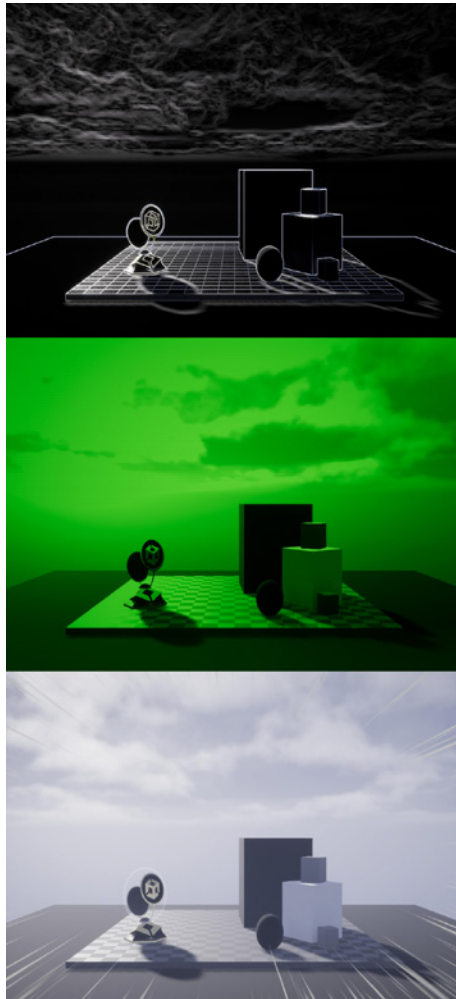
The Unity Fullscreen shader allows developers and artists to create custom effects that span the entire screen view. This can create effects such as a screen turning red when a character takes damage, or to make droplets of water appear on the screen.

The Fullscreen shader can be used in three ways:

- To create a custom pass effect
- To create a custom post-processing effect
- In a C# script with the **HDUtils.DrawFullscreen** or **Graphics.Blit** functions

To create a Fullscreen shader, create a new Fullscreen Shader Graph or modify an existing one. Make sure to include a Fullscreen Master Stack.

HDRP also includes sample Fullscreen shaders, which can be imported into your project through the Package Manager. These are representative examples of what you can do with the FullScreen Shader Graph.



Fullscreen Shader Graph samples

Terrain

Your game world starts beneath your players' feet. For that, Unity's Terrain Editor can help you build detailed and true-to-life landscapes, big or small.

A Terrain object starts as a plane that can be sculpted like a topographical map. By painting height values, you can raise mountains and carve valleys, assembling your virtual world from interconnected tiles. Then, finish the landscape with textures and vegetation – either procedurally or with an artist's brush.

The [Terrain Tools](#) package further improves the built-in workflow with additional sculpting brushes and a collection of utilities to help automate editing or setting up terrain assets.



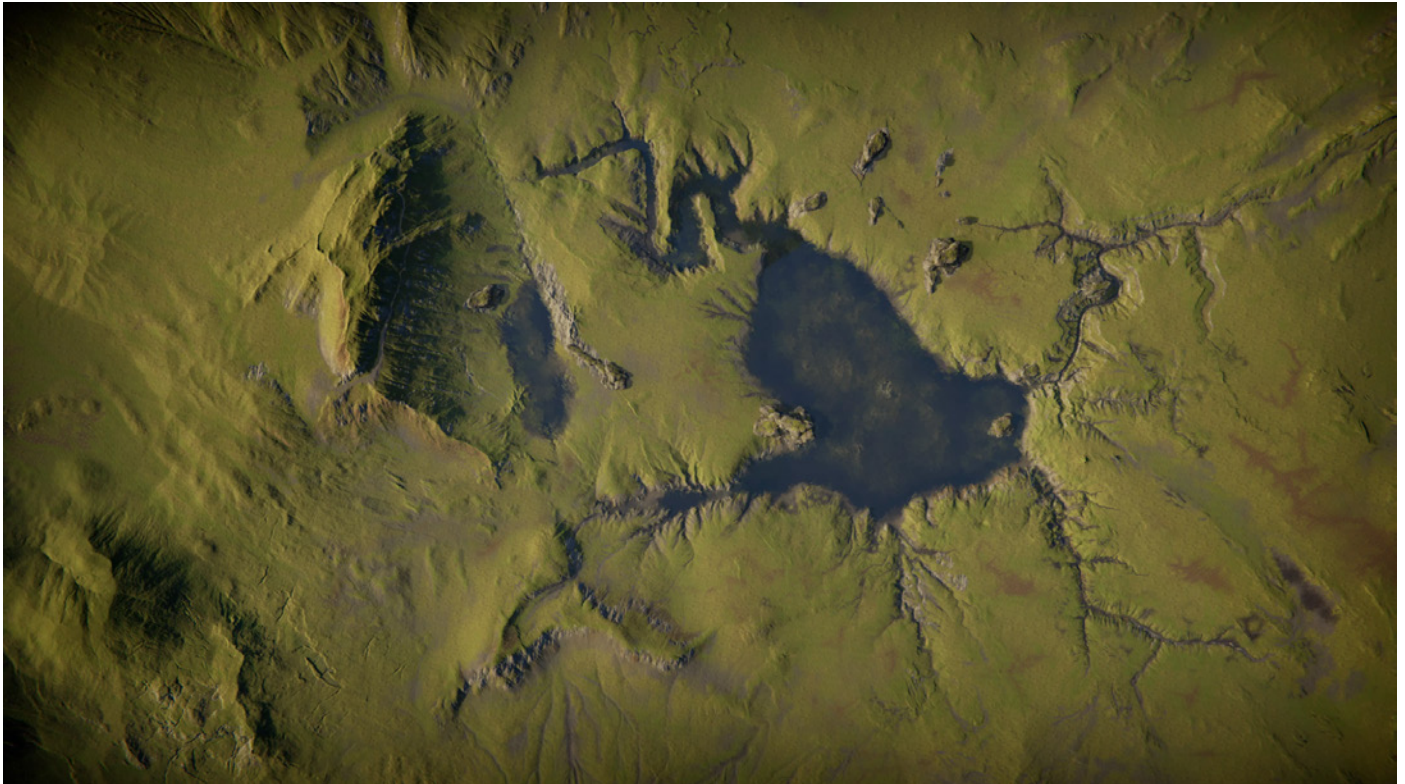
The Terrain system works with the Cloud Layer Volume component override.

Learn how to create and customize a terrain using specific tools and techniques with this [series of tutorials](#) from Unity Learn.

Creating terrains

When you create a new terrain in the Hierarchy, Unity adds a large flat plane to your scene. This grid of points can be modified at each vertex.

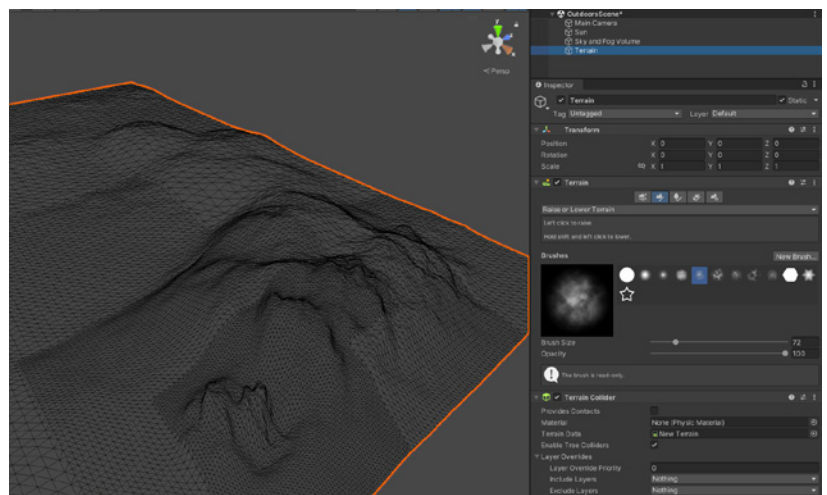
The shape of the terrain is determined by a grayscale **heightmap**. This grayscale map determines how to raise or lower each vertex on the grid. White represents the highest point and black is the lowest.



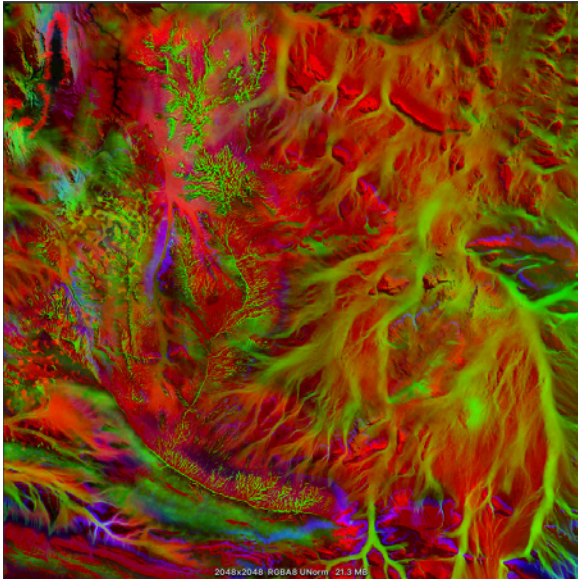
Paint on a heightmap to modify the Terrain.

Sculpting

The Terrain component includes various brushes that allow you to sculpt terrain. Use these to raise or lower parts of the terrain, smooth out areas, or even create custom brushes for specific shapes or patterns.



Use brushes to sculpt terrain.



Texturing and detailing

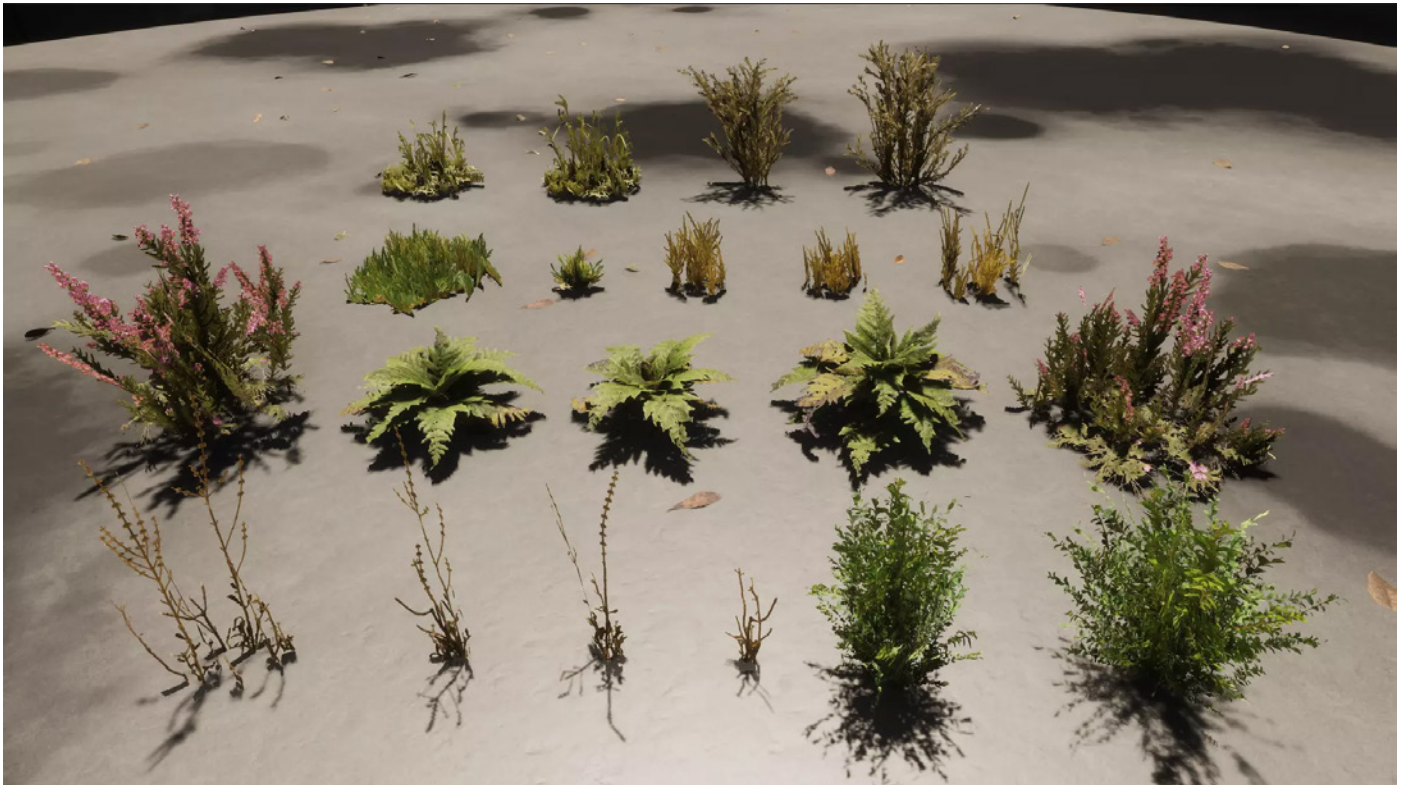
You can use [splatmaps](#) to paint different textures on the terrain. These are like layers in image editing software, allowing you to define and blend textures together.

Then, use **detail maps** for finer details like grass, flowers, or small rocks. Paint these directly onto the terrain.

Paint terrain textures using splatmaps.

Trees and vegetation

No terrain is complete without its flora. Unity provides a Tree Editor as part of the Terrain component. This is useful when you want to create detailed forests and jungles with different tree types and variations.



The Terrain system offers advanced vegetation features.

The Terrain component's tree palette can create a more natural look by mixing trees at different maturity stages. You can place trees individually or use mass placement tools. Grass and flower texture billboards can then be used to add layers of detail.

Add [Wind Zones](#) to make your trees and vegetation sway with the wind, adding life to the scene.

SpeedTree integration

Unity also supports the direct import of [SpeedTree models](#) (.SPM or .ST files), treating them like standard game objects. Use the Terrain Editor to paint SpeedTree vegetation directly onto terrains. Painted trees can automatically receive colliders to interact with other objects at runtime.

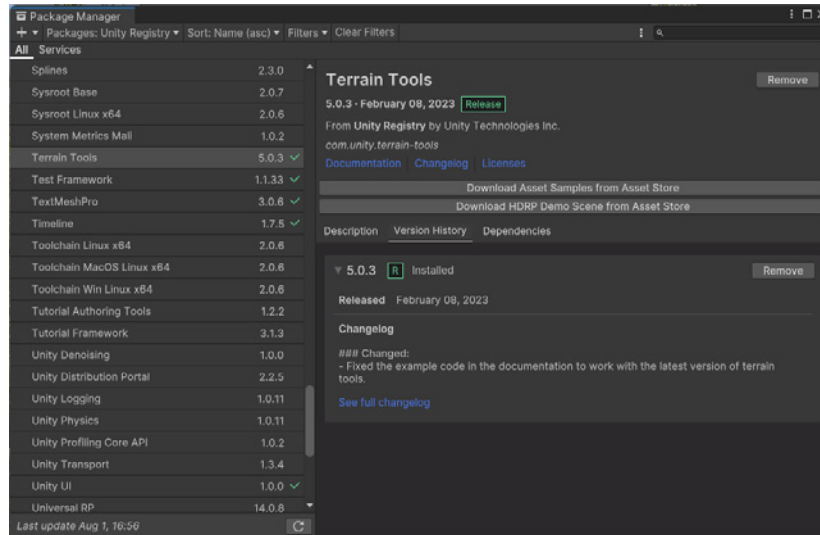
SpeedTree models come with built-in LODs for optimized performance. The rendering system batches the models for efficiency. SpeedTree vegetation can receive shadows and can contribute to global illumination.



SpeedTree vegetation.

Terrain Tools package

The [Terrain Tools](#) package adds additional Terrain sculpting brushes and tools to Unity. This add-on toolset is suitable if you require more control over the appearance of your Terrain and want to streamline Terrain workflows.



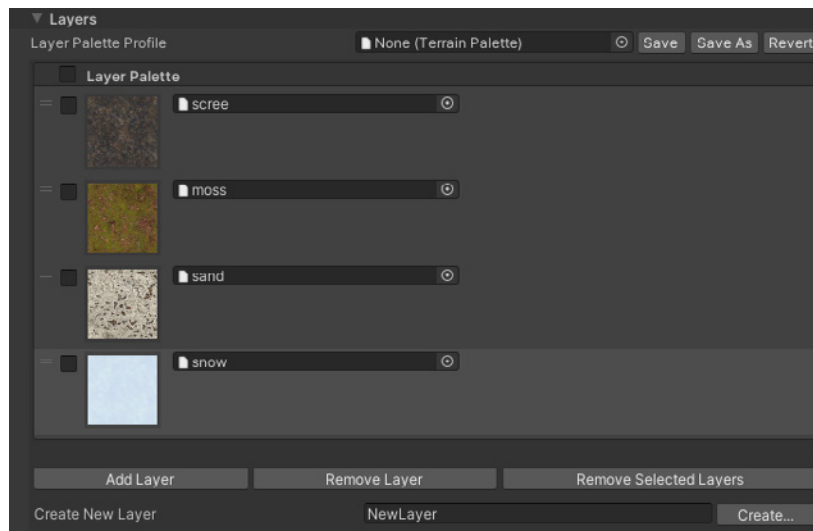
Add the Terrain Tools package for advanced features.

This package can help create more complex-looking terrain, or author terrain texture data in external digital content creation tools.

For more information, see [Getting started with Terrain Tools](#).

Painting Terrain

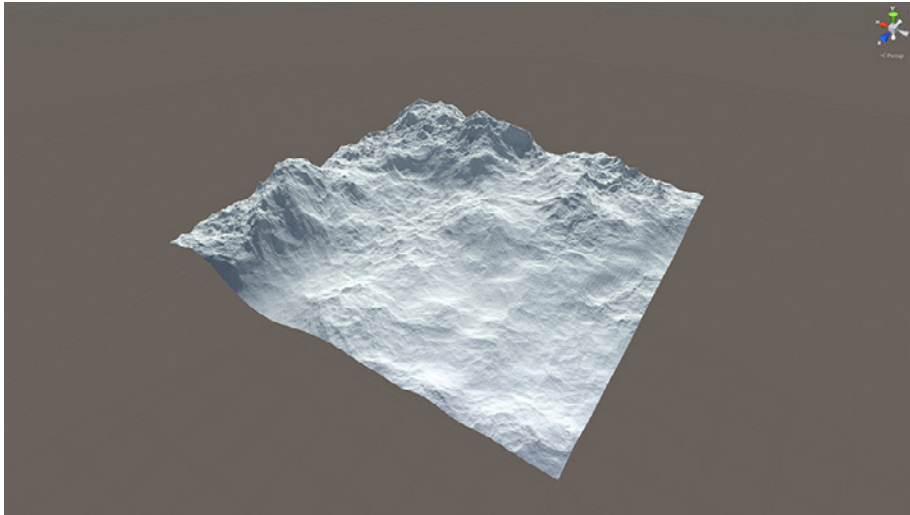
The Terrain Tools package adds additional tools to the [built-in](#) ones and improves the functionality of the built-in [Paint Texture](#), [Smooth Height](#), and [Stamp Terrain](#) tools. Refer to this [documentation page](#) for a complete list of [Paint Terrain tools](#).



Paint in layers using the Terrain Tools.

Sculpt tools alter the shape of the terrain with additive and subtractive methods:

- **Bridge** creates a brush stroke between two selected points to build a land bridge.
- **Clone** duplicates terrain from one region to another.
- **Noise** uses different noise types and fractal types to modify terrain height.
- **Terrace** transforms terrain into a series of flat areas like steps.



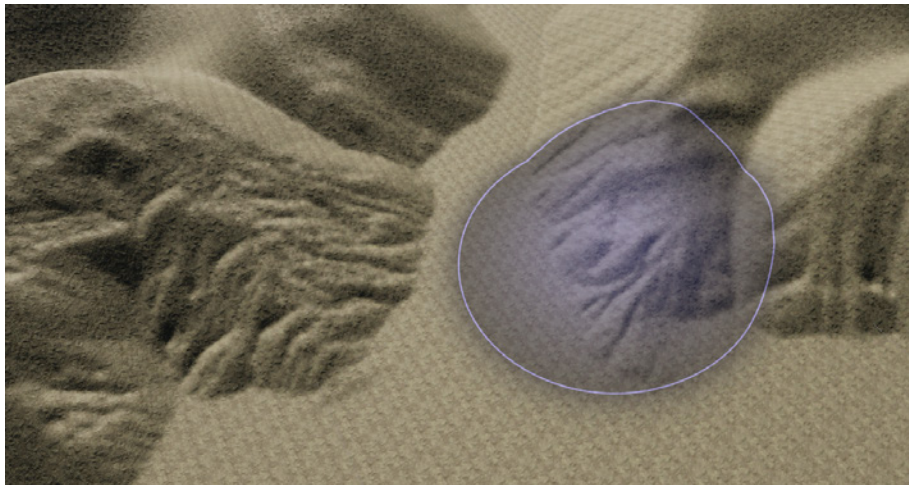
Noise modifies terrain height.

Effects tools modify the terrain based on its existing height:

- **Contrast** expands or shrinks the overall range of the terrain height.
- **Sharpen Peaks** can sharpen peaks or flatten already-flat parts of the terrain.
- **Slope Flatten** flattens the terrain while maintaining the average slope.

Erosion tools simulate the effect of flowing water or wind and the transport of sediment.

- **Hydraulic** simulates water erosion with sediment following a flow field. Use this tool to create valley and fluvial features.
- **Thermal** simulates the effect of sediment settling on the terrain while maintaining a natural slope.
- **Wind** simulates the effect of wind erosion and redistributing sediment.



Erosion tools simulate the effect of flowing wind or water and the transport of sediment.

Transform tools push, pull, and rotate terrain.

- **Pinch** pulls the height towards or bulges it away from the center of the brush.
- **Smudge** moves terrain features along the path of the brush stroke.
- **Twist** rotates terrain features around the center of the brush, along the path of the brush stroke.



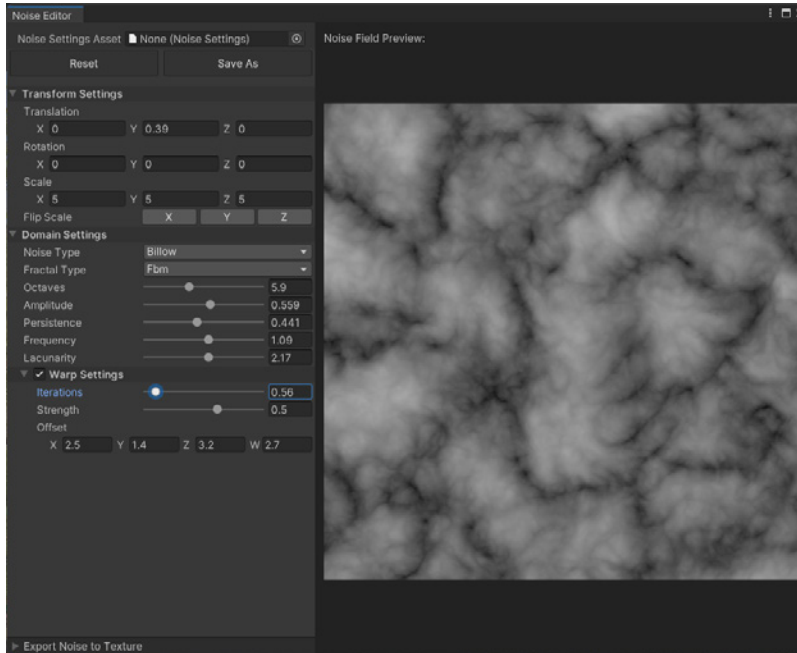
Twist rotates terrain features around the center of the brush.

You can also create your own custom terrain painting tools. For more information, see [TerrainAPI.TerrainPaintTool](#) and [Create a custom terrain tool](#).

Noise Editor

The **Noise Editor** allows you to author and manage Noise Settings assets, which you can use in the [Noise Height Tool](#) and [Noise Brush Mask Filters](#). You can also use the Noise Editor to generate procedural textures for external use.

To open the Noise Editor, select **Window > Terrain > Edit Noise** from the menu.

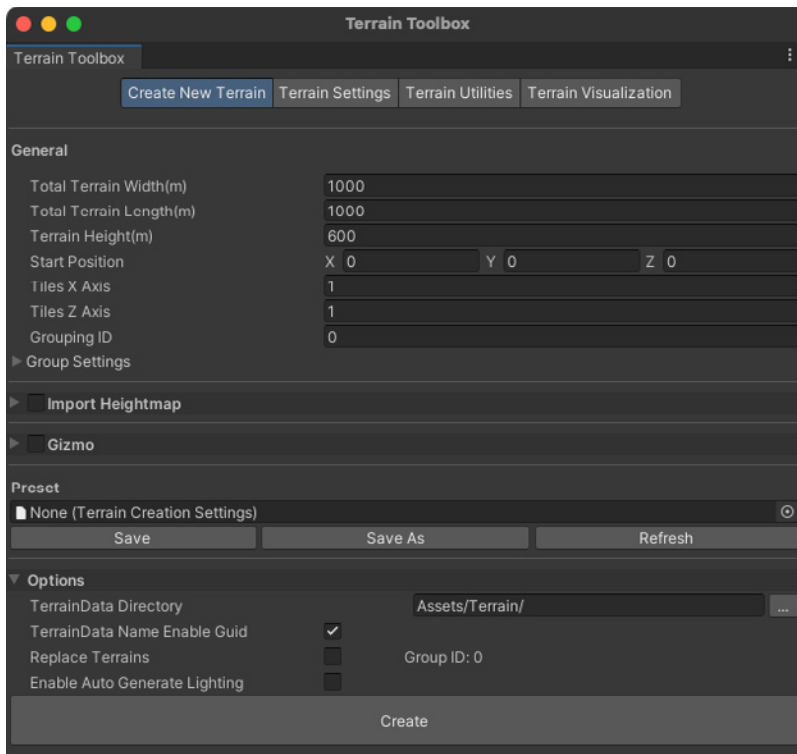


The Noise Editor

Terrain Toolbox

The Terrain Toolbox is an Editor window that contains useful tools to streamline terrain workflows. It allows you to create new terrain from preset settings or imported maps, batch change terrain settings, and import/export splatmaps and heightmaps.

To launch the Terrain Toolbox, select **Window > Terrain > Terrain Toolbox**.



The Terrain Toolbox improves workflow.

Ray tracing support for terrains

Unity 2022 LTS supports ray tracing terrain with some limitations. While terrain is influenced by ray traced effects, it doesn't actively contribute to them.

Specifically, the terrain is omitted in ray traced reflections. Instead, you'll need to use planar reflections when necessary (e.g., the surface of a lake). Also, Ray Traced Ambient Occlusion and Global Illumination do not affect the terrain.

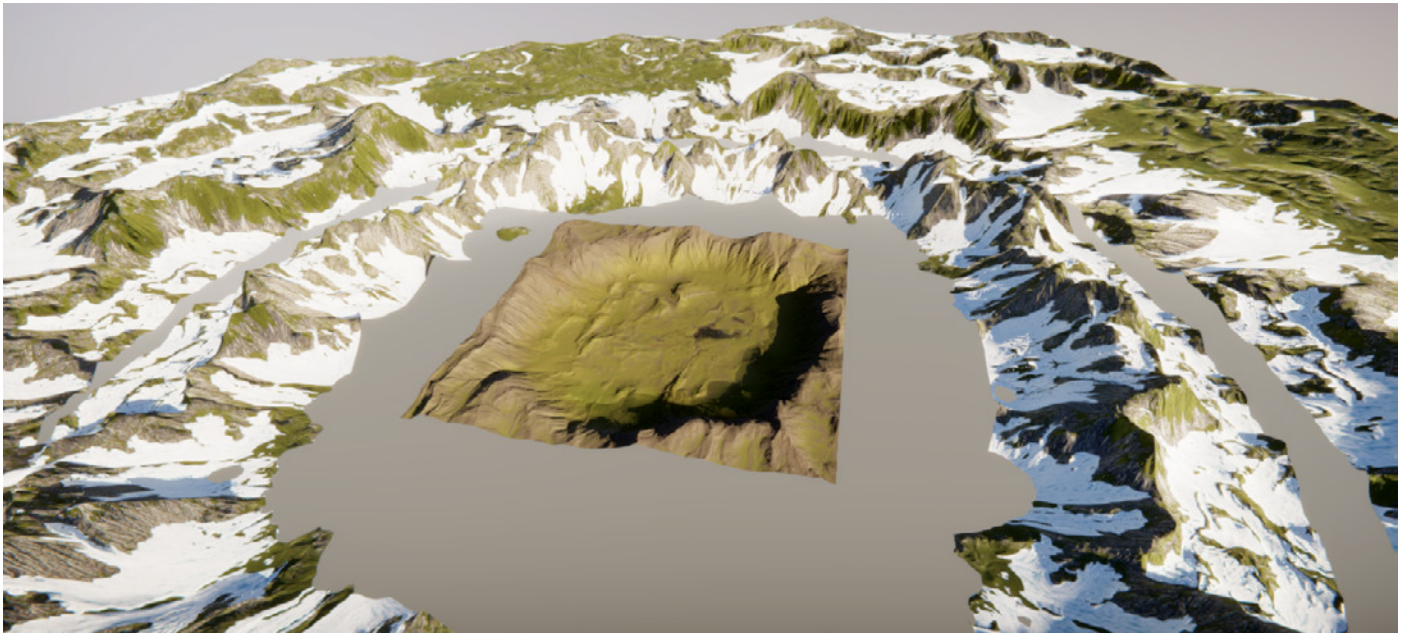
HDRP Terrain Demo

This package includes a mix of example content to help you get the most out of the Unity Terrain system. You can use the textures, brushes, and models in this sample pack for your own projects.



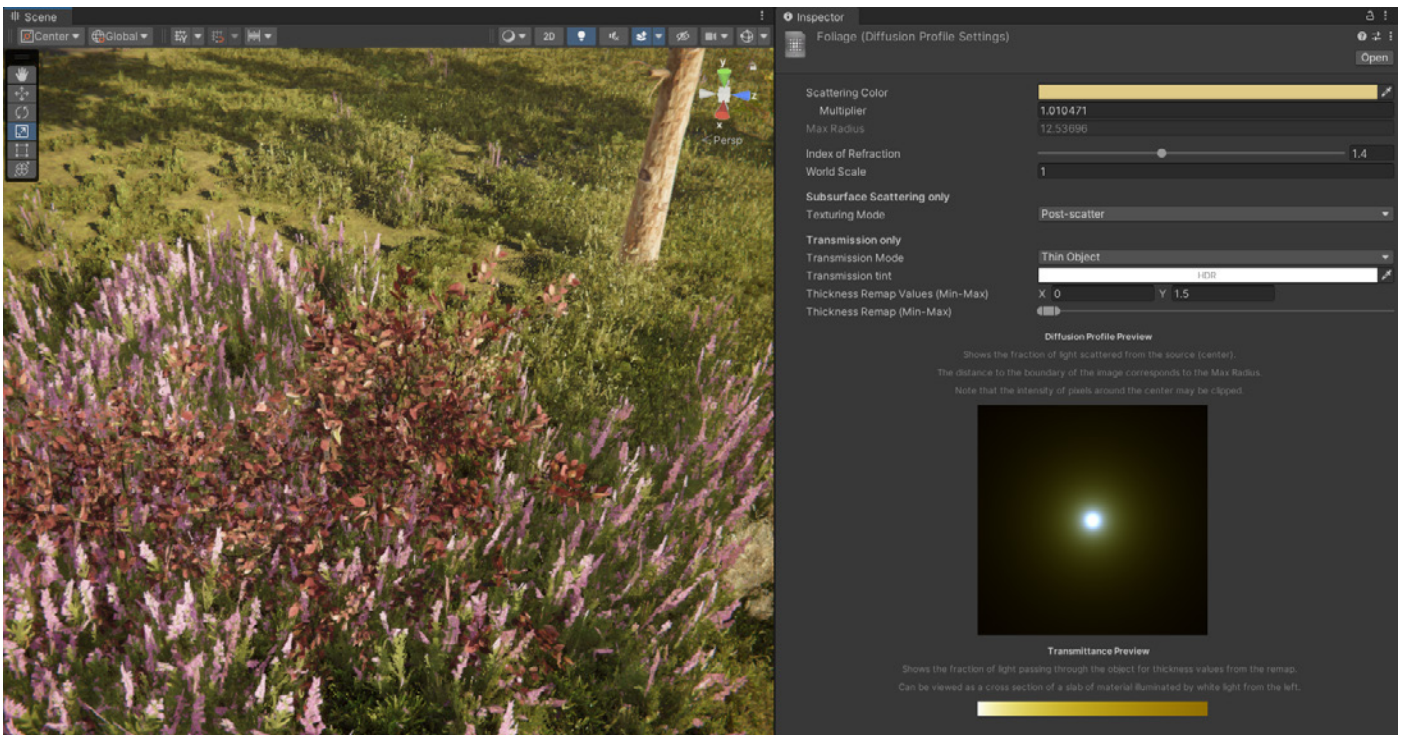
The HDRP Terrain Demo is available via the Package Manager.

This scene includes a series of bookmarked locations with notes and tips. Use the Buttons in the Inspector to focus the camera on points of interest and cycle through the bookmarks.



Terrain tiles in the HDRP Terrain Demo.

Foliage shaders used in this scene utilize the HDRP Diffusion Profile assets for subsurface scattering. This can help translucent organic materials look smooth and natural rather than rough and plastic-like.



Diffusion Profiles store subsurface scattering settings.

Clouds

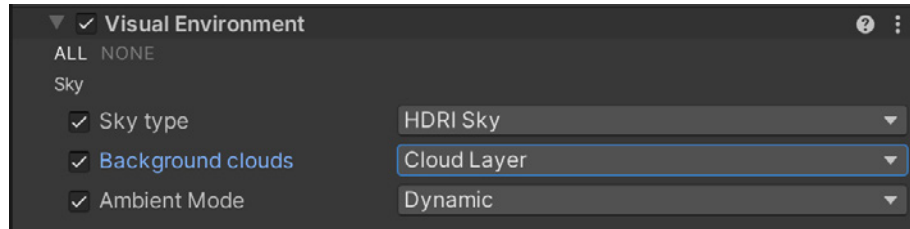
Skies would not look complete without clouds. The **Cloud Layer** Volume component override can generate natural-looking clouds to complement your **Sky** and **Visual Environment** overrides. **Volumetric Clouds** produce realistic clouds with an actual thickness that react to the lighting and wind.



A Cloud Layer appears in front of an HDRI sky.

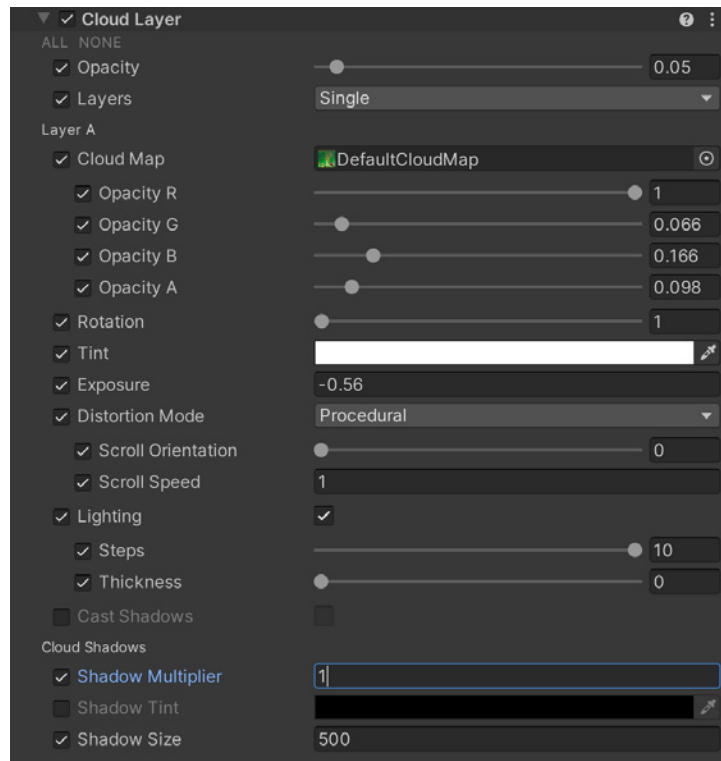
Cloud Layer

The **Cloud Layer** override is a 2D texture that you can animate with a flowmap, and which uses the red and green channels to control vector displacement. The clouds can add some slight motion to your skies in Play mode, making the background more dynamic. The cloud layer sits in front of the sky, with an option to cast shadows on the ground.



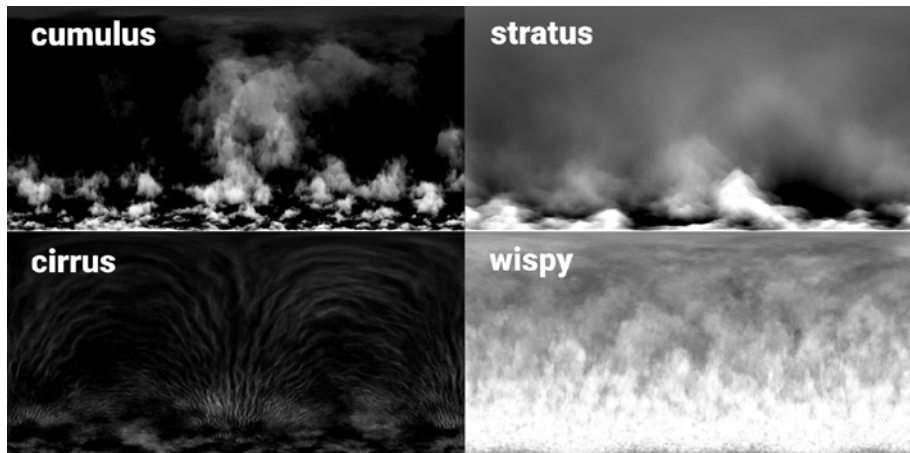
Enable the Cloud Layer override in the Visual Environment override.

In a local or global Volume, enable **Background clouds** in the Visual Environment, then add the **Cloud Layer** override.



The Cloud Layer override.

The cloud map itself is a texture using a [cylindrical projection](#), where the RGBA channels all contain different cloud textures (cumulus, stratus, cirrus, and wispy clouds, respectively). You can then use the cloud layer controls to blend each channel and build up your desired cloud formation. Two layers with four channels can simulate and blend from up to eight clouds.



The four channels of the DefaultCloudMap

Modify the cloud's animation, lighting, color, and shadows until you get a skyscape to your liking.

Atmospheric and sun-based lighting

Unity 2022 LTS improves the physically based rendering of cloud layers.

When using the cloud layer in combination with the Physically Based Sky override, the sunlight color now correctly accounts for atmospheric attenuation.

Additionally, the sunlight color will now always impact the color of the clouds, even if the ray marching is disabled. Improvements have also been made to the ray marching algorithms to improve scattering, and HDRP now shows more consistent results when changing the number of steps.



Cloud layers with a Physically Based Sky override

Note that you may have to tweak the density and exposure sliders to match the visuals with prior versions of HDRP.

Volumetric clouds

If you need clouds to interact with light, use the **Volumetric Clouds** override. These can render shadows, receive fog, and create volumetric shafts of light. Combine these with cloud layer clouds or add them separately.

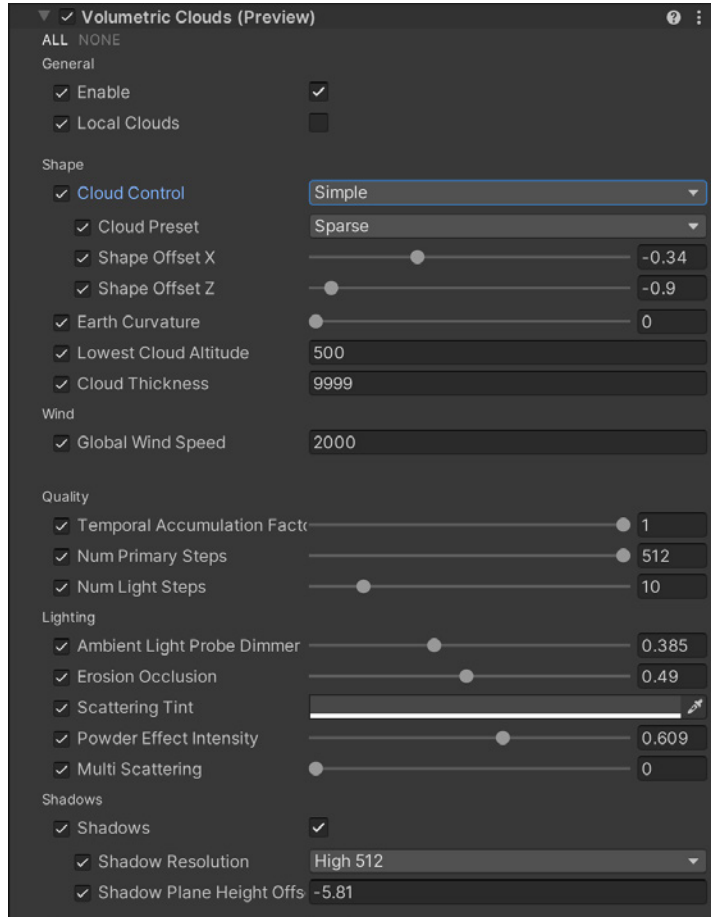
To enable the Volumetric Clouds override:

- In your HDRP Asset: Enable **Lighting > Volumetric Clouds > Volumetric Clouds**.
- In your local or global Volume: Add the **Volumetric Clouds** override.



Volumetric clouds can react to the main directional light.

The Advanced and Manual **Cloud Control** options allow you to define maps for each type of cloud.



Volumetric Clouds override

Refer to HDRP documentation for the [Cloud Layer](#) and [Volumetric Clouds](#) overrides.

HDRP clouds presets blending

The latest update to the cloud layer system supports blending between two distinct cloud setups. This allows for smoother transitions. For example, you can go from a sparse to an overcast sky by merely adjusting the volume blending between the two configurations.

The introduction of curve blending in the volume framework means that volume profiles with volumetric clouds can now be blended seamlessly. Furthermore, the revamped presets now offer increased micro details and performance enhancements.

This update reduces visual artifacts like flickering or shimmering (seen with the previous presets, which relied on Look-Up Tables).

HDRP water system

Introducing the water system

Water, water – it's everywhere in the real world, and now Unity's water system makes it easier than ever to integrate it into your game worlds as well. Designing a secluded tropical lagoon or a treacherous ice-covered fjord for your project? Adding water elements to your HDRP environments is just a few clicks away.



Use the water system to add oceans, rivers, and lakes.

Some key features of the water system include:

- **Physically Based water rendering:** The water system includes a physically based water shader with adjustable properties for smoothness, refraction, and light scattering.
- **Water movement:** The water system simulates water plane deformation, including swells, agitations, ripples, and currents.
- **Underwater effects:** When the camera submerges below the surface, underwater rendering accounts for water's physical properties, recreating light refraction and absorption.
- **Foam:** Foam can automatically form based on the water surface simulation and wind speed. Local foam generators can also simulate white water effects for smaller areas, like the wake of a boat.
- **Integration:** Connect water surfaces to the surrounding props and terrain with artist-friendly components like masks, decals, and deformers.
- **Performance:** The system is optimized for various platforms and offers several debugging modes for visualization.



The Water System is flexible and customizable.

Getting started

The Unity water system requires HDRP 14 (available in Unity 2022 LTS or above).

Unity 2023 new features

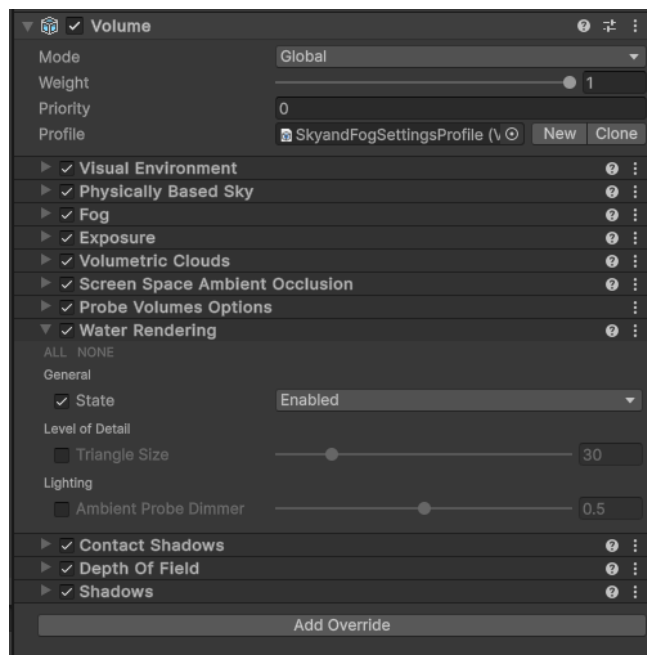
If you're new to the water system, we recommend starting with Unity 2023.1 (HDRP 15), which features several notable improvements from previous versions:

- Local currents
- Surface deformers
- Water excluders
- Local foam generators
- Water line and underwater features
- Extra Debug modes

You'll also need Unity 2023.1 or higher to explore the sample scenes.

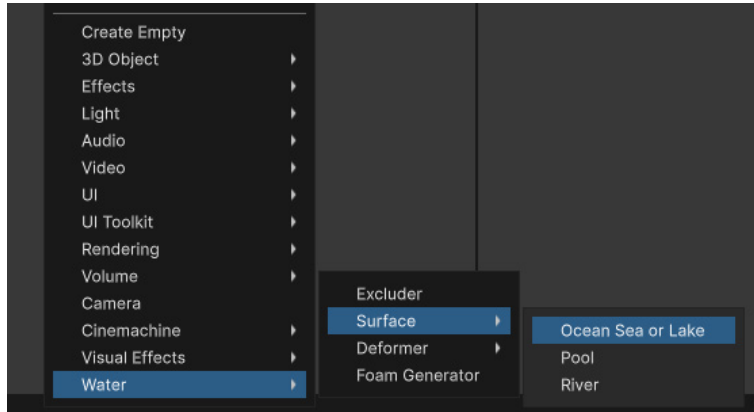
To get your project ready for the water system:

- Activate and configure water for each Render Pipeline Asset (per quality level that requires water).
- Enable water in the Frame Settings of your camera(s) (**Edit > Project Settings > Graphics > HDRP Global Settings**).
- Use the **Water Rendering** Volume override in the scene to control where water rendering is active, based on the camera's position.



Enable Water Rendering in the Volume overrides.

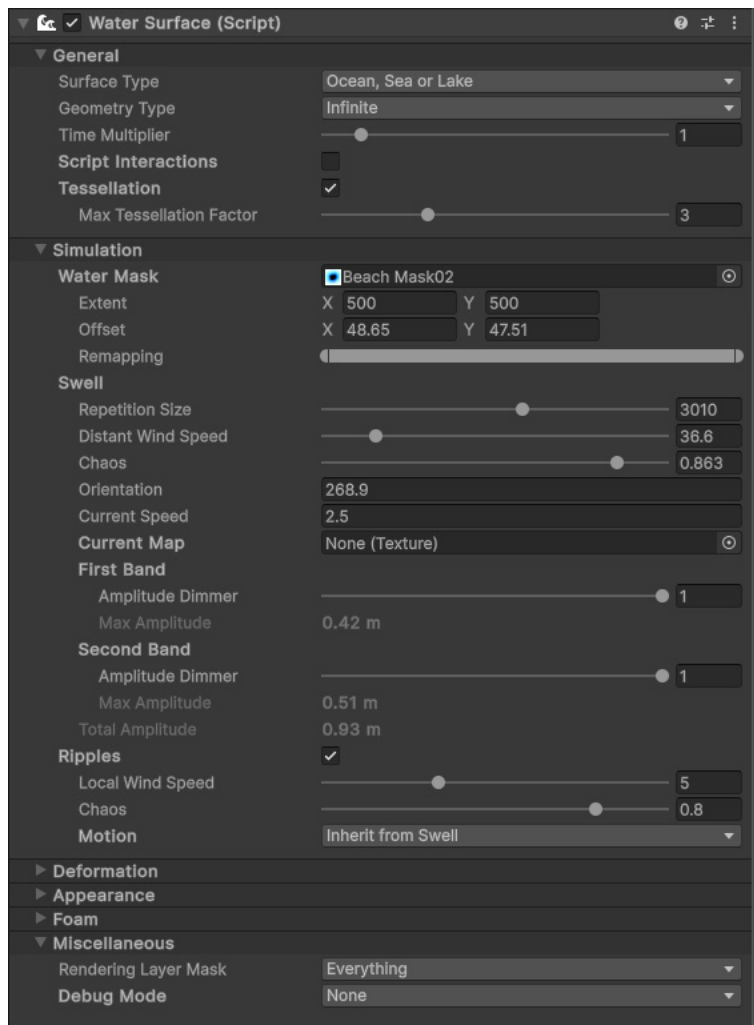
Once water is active, you can add preconfigured water bodies from the GameObject menu. HDRP provides three water surface types: pool, river, and ocean.



Add a water object from the GameObject menu.

Water Surface component

Water surface objects come with a Water Surface component script that controls the water's general parameters.



The Water Surface component

These parameters configure the water surface's simulation and rendering:

- **General:** Defines the overall type of water body (e.g., Ocean/Sea/Lake, River, Pool) and geometry used to render the water surface (e.g., Quad, Custom, InstancedQuads, Infinite)
- **Simulation:** Controls how the wave patterns and ripples form, emulating how the wind and moon affect the water's surface
- **Deformation:** Influences how a local deformation can raise or lower part of the water, useful for making waterfalls or other elevation changes
- **Appearance:** Determines the water's color, smoothness, refraction, and light scattering; caustics and special underwater settings enhance the physically based shading.
- **Foam:** Controls the appearance and behavior of foam at the crest of waves, around objects in the water, or along the shoreline
- **Miscellaneous:** Controls the Rendering Layer Mask and Debug modes

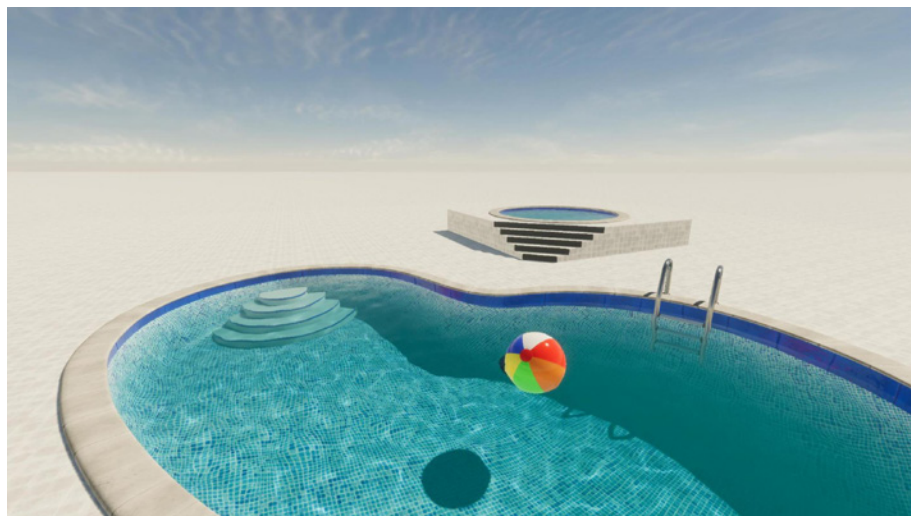
HDRP Water samples

The best way to familiarize yourself with all of the water system's properties is to see them in action. HDRP includes several sample scenes to demonstrate capabilities and provide starting points.

In the **Package Manager**, install them from the **Samples** tab of the **High Definition Render Pipeline**.

Let's take a look at each sample.

Swimming Pool: This scene shows multiple water surfaces for pools at different heights. It also demonstrates using custom meshes to achieve unique pool shapes beyond basic rectangles.



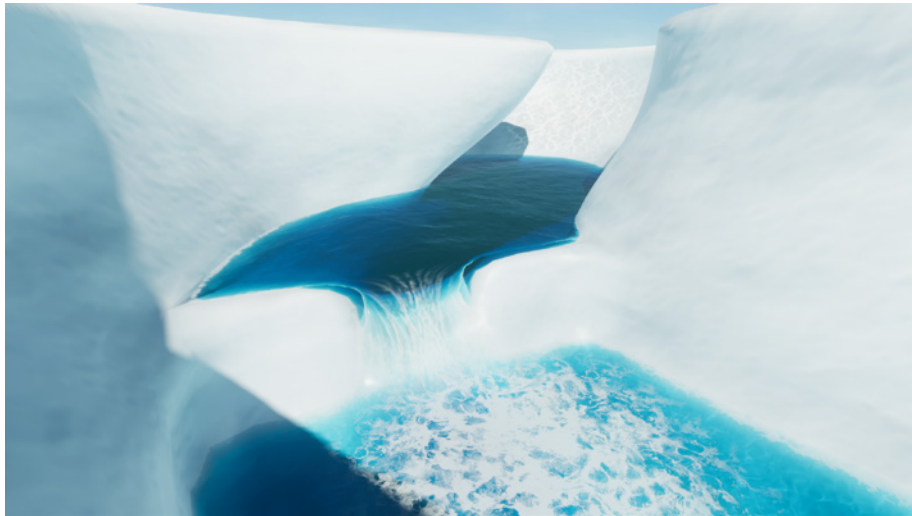
The Swimming Pool sample

Island: This scene features a small landmass surrounded by a body of water with waves. Water masks remove swell around the island, while deformers create waves. Decals and foam generators add foam for breaking waves. Physically simulated seagulls float on the water surface using the scripting API.



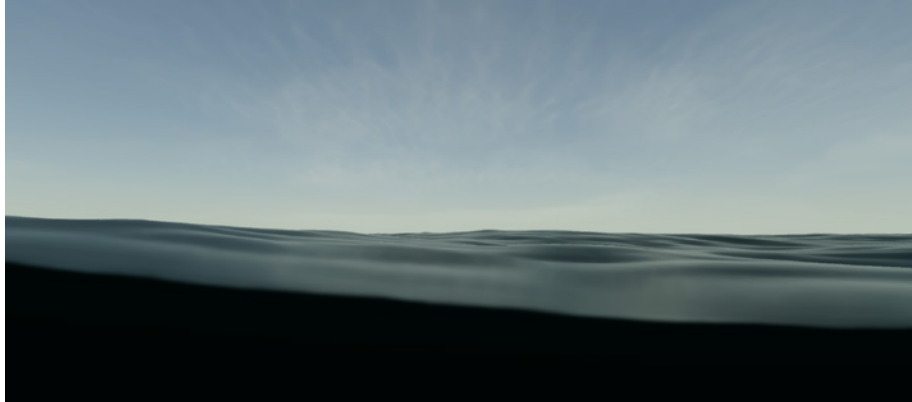
The Island sample

Glacier: This scene includes a river, waterfall deformer, current simulation to make the water flow, foam behind moving icebergs, decals for spray effects, and projected caustics.



The Glacier sample

Water Line: This scene modifies the water's surface level and underwater rendering using a custom pass, generating a larger blurry water line and simulating water on a camera lens.



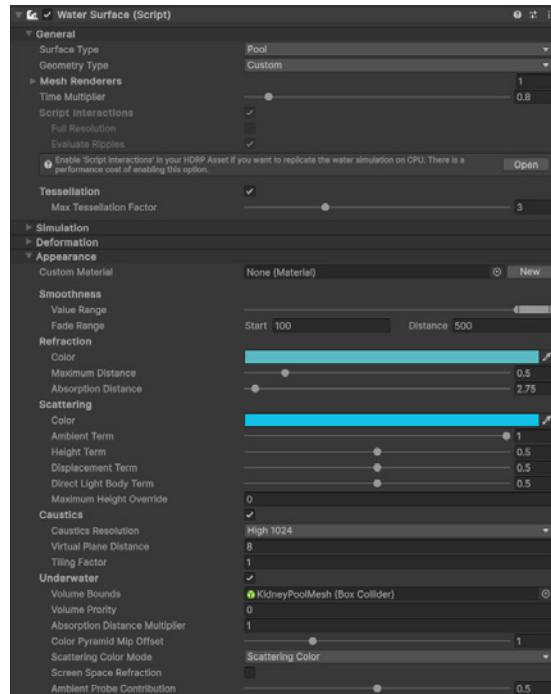
Rendering a water line

If you're new to the water system, use one of these samples as a starting template for a similar environment. Then, you can customize the look of the water to match your own game world.

Physically based shading

The water system uses a physically based water shader. Modify its smoothness, refraction, and light-scattering properties in the Appearance parameters.

The scattering works like the base color of the water, establishing a general feel. Then change the Absorption Distance and Refraction Color to control how transparent your water is. This tints the objects seen through the refracted water.



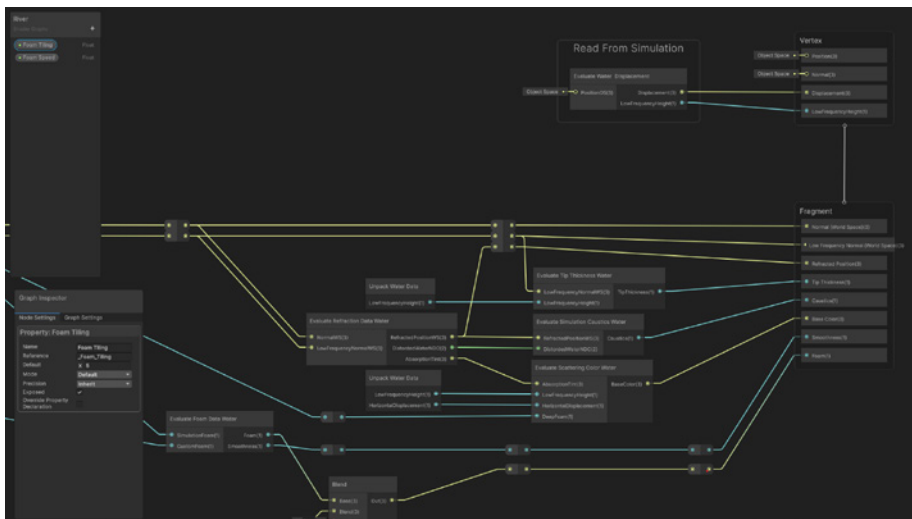
Customizing the Appearance parameters

For example, use a large Absorption Distance with a cyan scattering and refraction color if you want the water to appear a deep, clean Caribbean blue. For turbid water from a muddy river, opt for a dark brown scattering color with a small absorption distance.



Modifying the Absorption Distance parameter

You can also modify the water material using a custom Shader Graph. See the **River** Shader Graph asset in the samples as a reference.



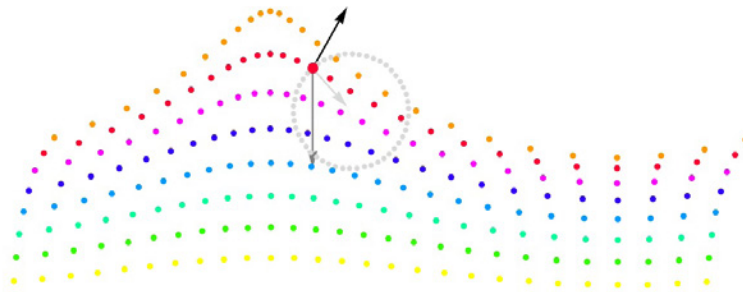
The River Shader Graph customizes the water material.

Simulating waves and wind

Each Water Surface component contains a number of simulation parameters that determine how the water reacts to factors like waves and wind. Waves are procedurally generated using a [Fast Fourier Transform \(FFT\)](#) simulation. This sums together different frequencies of simple waves to form a more complex one.

Swells, agitation, and ripples

One band of frequency produces broader waves called **Swells** (for sea, ocean, or lake surfaces). Swell parameters modify the amplitude, direction, and wavelength of larger [Gerstner Waves](#). These approximate the effects of distant wind and/or the pull of the moon.

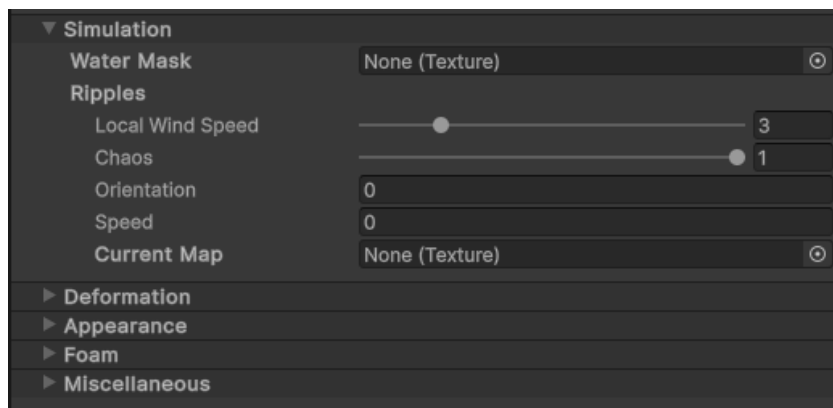


Larger waves determine the swells. Source: "[Trochoidal waves](#)," Wikipedia.

Agitation is the equivalent of swells but only for river surfaces. Whereas swells are optimized for larger bodies of water, agitation parameters simulate the more dynamic and turbulent conditions found in rivers.

Another band of higher frequencies simulates currents or local winds that produce **Ripples**. These are smaller waves that are close together and add fine detail to the water surface.

These parameters can be customized globally or per water body. Thus, different bodies of water within the same scene can have waves moving in different directions based on their local settings.



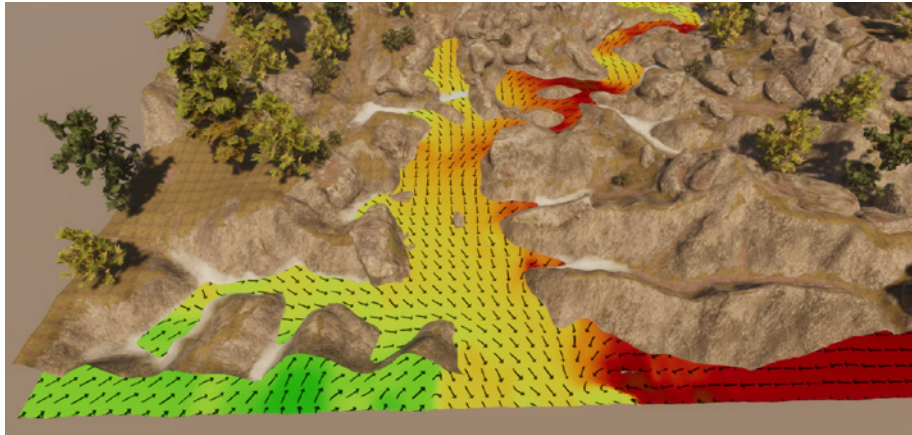
Simulation parameters control swells and ripples.

Current Maps

A **Current Map** is a 2D texture that dictates a water surface's current direction and speed. Here, the texture's Red and Green channels determine the direction, while the Alpha channel indicates its speed. Import the texture into Unity, then configure its settings for use in the water material or Volume override.

Follow these guidelines for creating your own Current Maps in third-party software like Photoshop or Krita.

Use the **Miscellaneous > Debug Mode** dropdown menu to preview the resulting current.

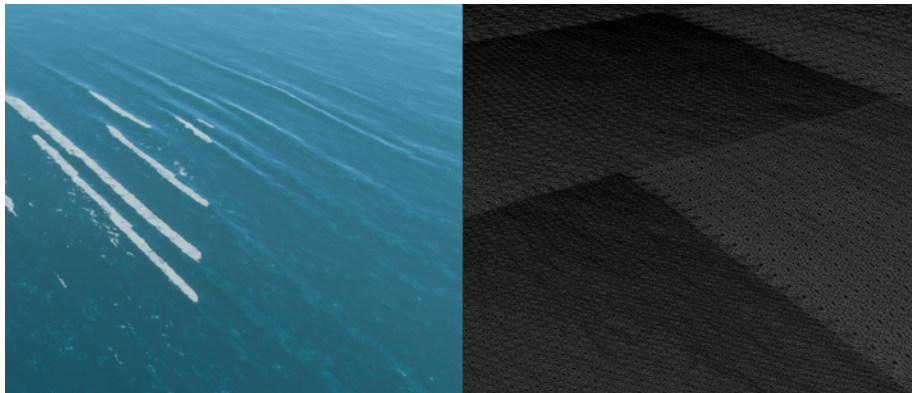


A Current Map shown through Debug mode

Procedural rendering

The water system uses vertex displacement for rendering water plane deformation. This procedural approach, similar to terrain rendering, works for vast expanses like infinite oceans or long rivers.

For the detailed rendering of minor ripples, HDRP uses GPU tessellation. This process subdivides triangles with specialized shaders, increasing triangle density near the viewer.

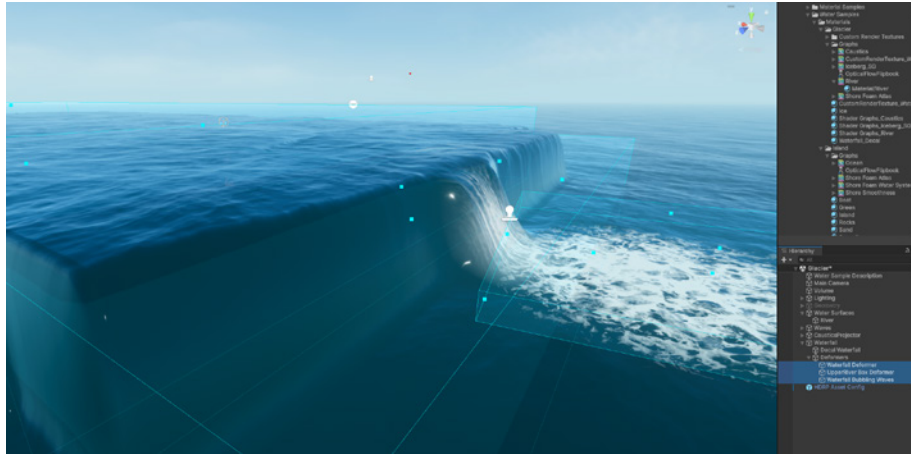


Water rendering uses GPU tessellation.

Deforming a water surface

A **Water Deformer** component can alter the water surface's shape. You can choose one based on predefined shapes or customize the deformation using a texture.

Water Deformer components can influence one another additively when they are placed in the same location. For instance, stacking two one-meter box deformer results in an equivalent to a single two-meter box deformer. This offers you extra control and flexibility for shaping the water surfaces.



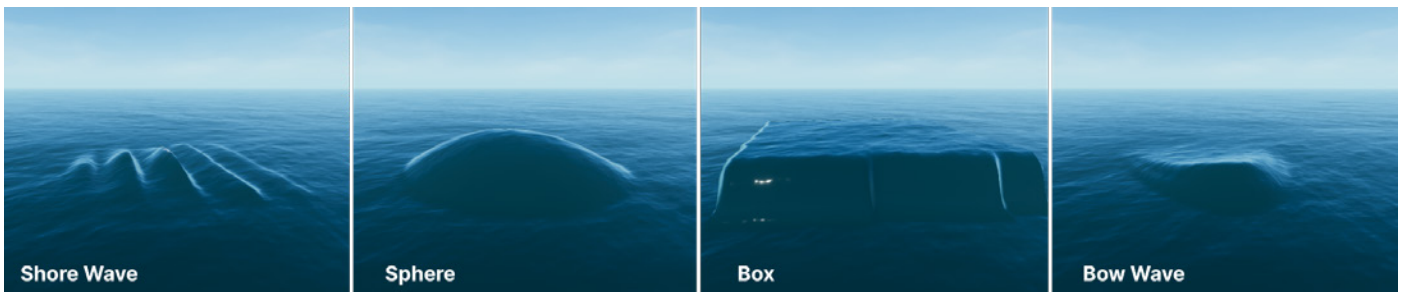
Add deformers to create a waterfall.

However, be aware of the maximum number of deformers that can be active simultaneously. Adjust this limit in the HDRP Asset via **Rendering > Water > Deformation > Maximum Deformer Count**.

Also, note that only a specific region of a water surface can undergo deformation. The size and offset of this confined region can be adjusted directly from the water surface's Inspector window. For debugging, select **Deformation** via **Miscellaneous > Debug Mode**.

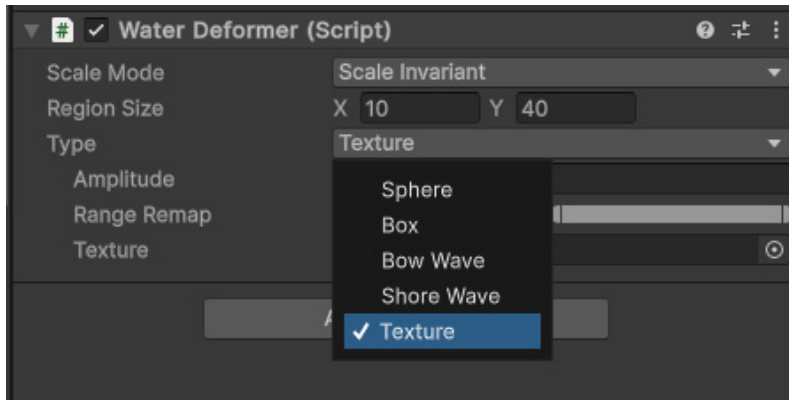
To create a deformation, navigate to **GameObject > Water > Deformer** and select the desired deformer type. For a water deformer to influence a water surface, the Water Surface must be enabled in the Deformation dropdown. Make sure this feature is also active in the Project's HDRP Asset and the Frame Settings.

The properties available in the Inspector vary based on the type of deformer chosen. Use Scale Mode, Region Size, and Amplitude to control the size and height.



Water deformers come in different shapes.

Each specific Deformer (Sphere, Box, Bow Wave, Shore Wave, and Texture) has a unique set of properties, described on this [documentation page](#).



The Water Deformer component

Adding foam

Foam simulates the white frothy water seen at the crests of waves or where water interacts with objects. This effect can help connect the water surface with objects making contact with it.

Surface foam

To activate foam for a GameObject, enable the corresponding checkboxes in the Water Volume component, and adjust the Foam parameters.

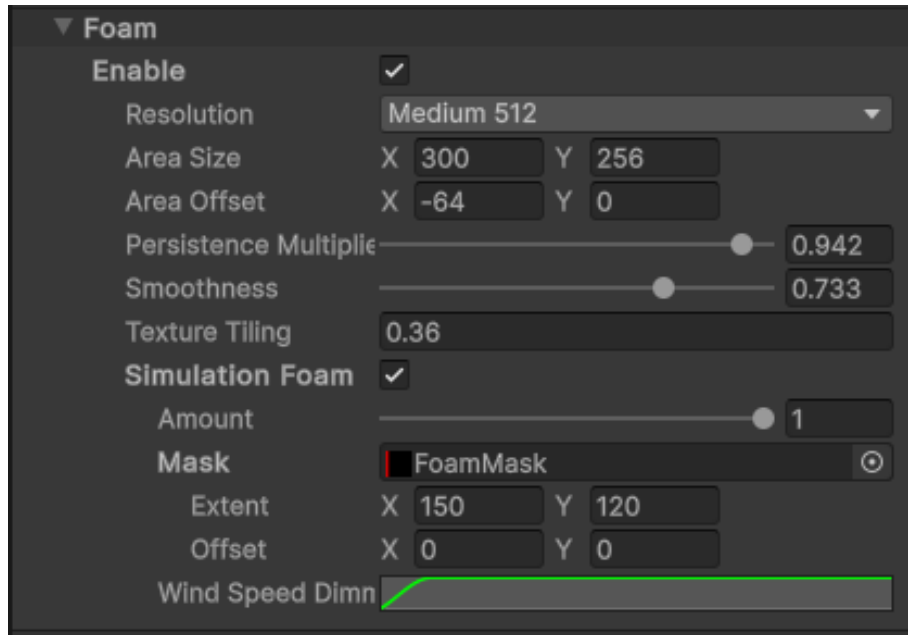
Currently, you only need to change two settings:

- **Foam Amount** determines foam patch size and controls the overall strength and prominence of the foam effect on the water surface. Increase its value to make the foam more pronounced, or reduce it for a subtler effect.
- **Wind Speed Dimmer** maps the amount of foam to the Distant Wind Speed parameter. Use the curve tool to determine the percentage of the Foam Amount to apply.



Surface Foam simulates frothy white water.

After fine-tuning these parameters, play the scene and observe how the foam behaves at runtime. Pay attention to where dynamic objects interact with the water or where the waves connect with the surroundings.



Adjusting the foam parameters a Water Surface

Foam generators

When objects or characters move through water, they often disturb the water's surface. You can also use a foam generator to stir up local foam to help integrate those objects. This can add white water for a boat trail or around rocks in open water.

The foam generator comes in three types: disk, rectangle, and texture-based. Adjust the parameters to control the foam's size, depth, and intensity.



A foam generator creates local foam around an object.

Decals and masking

Decals

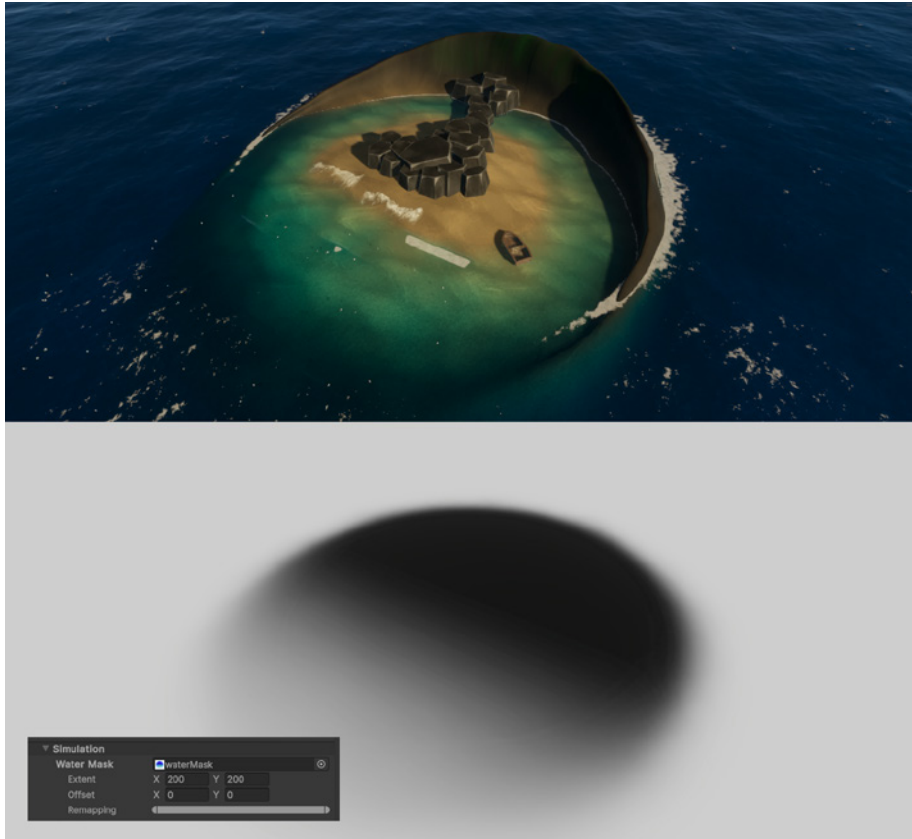
You can use a [decal](#) with a water surface in the form of a Decal Layer Mask, for example to imitate debris floating on the water.

Decals can add local foam, override the smoothness of the water, or simulate small local deformations such as droplets, impacts, or custom ripples by using a normal map. They can be also used to simulate wetness on sand or rocks (see the Island sample scene) or caustics on the walls (see the Glacier sample scene) independently from the water system.

Note that color cannot be changed since decals are only treated as grayscale. Global Opacity determines the amount of influence the decal has on the appearance of the water surface.

Water Mask

The Water Mask component lets you attenuate or remove ripples, swell, and foam in defined areas. This can be useful for directing the simulation for specific regions of the water surface and integrating the water with the surrounding terrain and props.



Water Masks remove or attenuate waves, ripples, and foam from defined areas.

Caustics

Caustics in Unity are patterns created when light rays are refracted or reflected by a curved surface and then projected onto another surface.

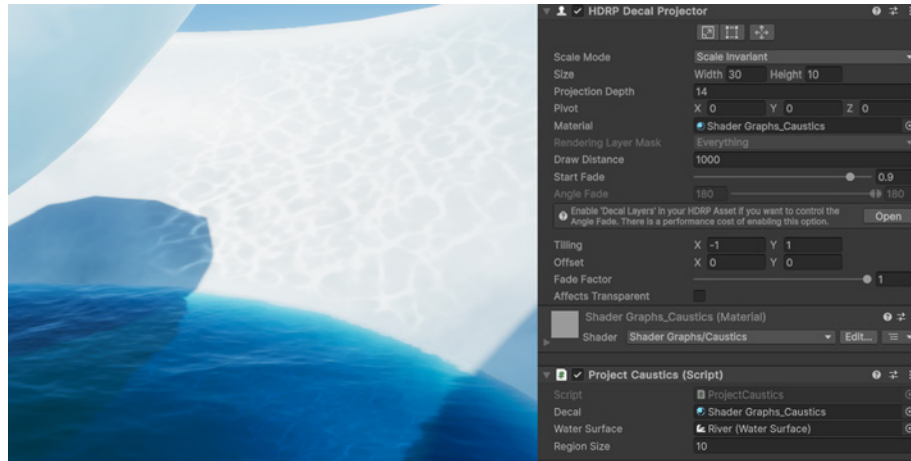
Lower the Absorption Distance to simulate cloudier waters, which would reduce the caustic intensity.



Caustics uses the Ripples Simulation Band by default.

By default, HDRP uses the Ripples property for caustics calculations. For rivers and oceans, switch the **Simulation Band** to generate caustics based on larger waves.

Note that caustics won't manifest above the water surface without custom scripting (e.g., the hull of a boat above the water line or the walls/ceilings above an indoor pool). Alternatively, use a Decal Projector to fake caustics that appear above the water surface (see the Glacier sample for an example).



Caustics approximated with a custom Decal Projector

Water exclusion

Sometimes you might want to prevent water from appearing on top of other surfaces. Though you can use a water mask for static environments, dynamic objects need a different solution.

Your moving objects can use a **water excluder**, a `GameObject` that marks part of the screen to not receive a water surface.

A common application of this is to eliminate the water surface within a floating object, such as the inside of a boat. This shows how an excluder can stop water from filling the hollowed portion of a floating object.



A boat with a water excluder.

Without a water excluder, the water surface might appear inside the boat. Using a custom mesh that conforms to the boat's interior shape as a water excluder, the water surface within the boat becomes invisible.

Make the new mesh a child of the moving model that needs to exclude the water surface. As it moves, the water exclusion mesh moves with it, maintaining the effect.

Note: Using the boat mesh itself in this example as a water excluder can introduce z-fighting. Thus, it's preferred to assign a separate, simplified mesh.

Rendering underwater scenes

Enable the Underwater option in the Water Surface component to render the water when the camera is below the water surface. This gives extra options for absorption, refraction, and scattering.



Underwater settings simulate reduced visibility.

To view non-infinite water surfaces (rivers and pools) from underwater, specify a collider as **Volume Bounds**. You can either use the Box Collider component that HDRP automatically provides or select a one in the scene for this purpose.

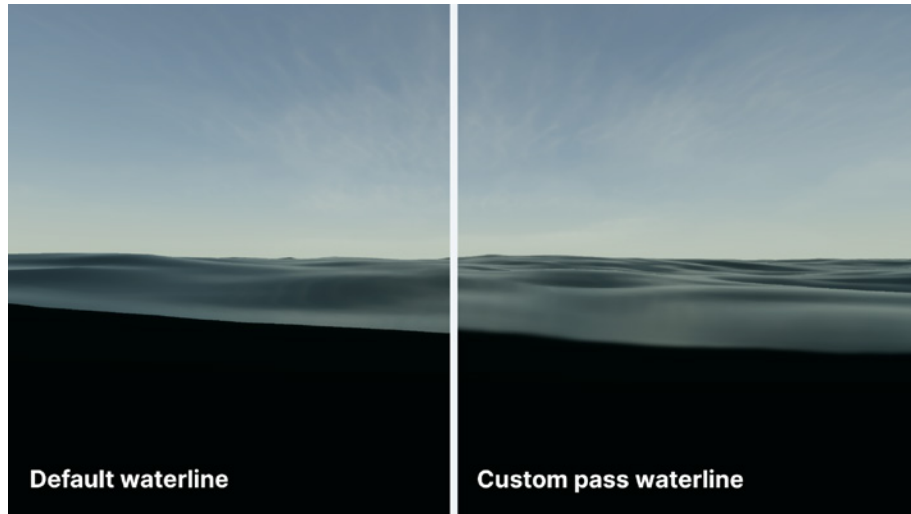
To view infinite water surfaces from underwater (oceans, seas, and lakes), specify a **Volume Depth**.

Waterline effects and custom pass

If your camera is partially submerged, you can see a “waterline” that appears between over and underwater rendering. By default, it appears as a very crisp border.

The Waterline scene shows how a custom pass can help here. It mainly uses two techniques:

- **Blur:** In real-life scenarios, the boundary between water and air (or a submerged object) is often not a sharp line due to factors like light refraction, tiny water droplets, and so on. A slight blur makes this more realistic.
- **Distorting the UVs:** A small distortion is added by stretching the UVs above the water line to create a meniscus-like effect.



Custom pass creates a more realistic waterline.

Water scripting

The water system runs its wave simulations primarily on the GPU for efficiency, but some processes can be mirrored on the CPU. This allows for water height and current sampling, useful for making objects float.

This is especially useful for scripting interaction. For example, you can query the height of the water surface and then make objects float on the water's surface.



Custom scripts make your GameObjects float.

To enable this, activate **Script Interactions** in the Water section of the HDRP Asset (**Project Settings > HDRP > Quality**) and in the Water Surface component.

The sample scenes include the example scripts **FitToWaterSurface.cs** and **FitToWaterSurfaceBurst.cs**. These show how to approximate buoyancy to a single object or an array of objects, respectively.

Be aware, however, of the CPU cost of script interaction with the water system. Use the [Burst compiler](#) when that's possible, and disable it when it's unnecessary.

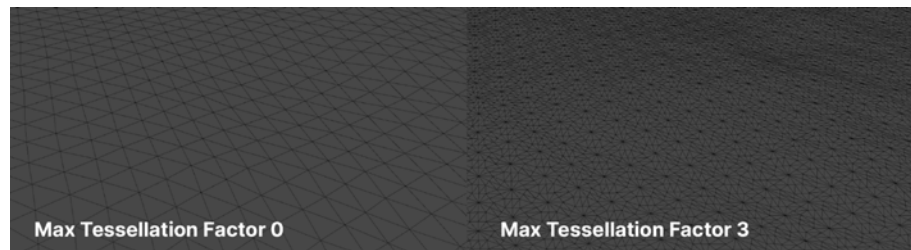
Also, note that Water Masks do not affect CPU simulations. As a result, buoyancy scripts might produce incorrect results for masked water surfaces.

Performance and optimization

Using the Glacier sample as a benchmark scene for the new water system, rendering time takes approximately 4 ms on the GPU on the latest generation console hardware, compared to 7 ms on previous generations.

Though your results can vary depending on quality settings and the complexity of the simulation, the system runs smoothly on various platforms.

Be aware that the system uses [deferred clustered](#) lighting to optimize shading and support a high number of light sources. Most of the rendering time is coming from the GBuffer pass due to the number of vertices required to get nice waves, even when they're in the distance.



Reduce tessellation settings to save resources.

When balancing visual quality with performance, consider adjusting these aspects of the water simulation:

- **Triangle Size:** Each Volume has the option to change the average size of a triangle onscreen. Smaller triangles provide more detail but are more computationally intensive.
- **GPU Tessellation:** The Water Surface includes options for Tessellation, which improves detail but increases the number of polygons. Note that surfaces like oceans, seas, or lakes require more calculations than pools due to more simulation bands. Lower the **Max Tessellation Factor**, **Tessellation Factor Fade Range**, and **Tessellation Factor Fade Start** to conserve memory accordingly.

- **Disable Script Interactions:** Only enable scripts if using water height data in scripts, as it increases calculations by running the simulation on both the GPU and CPU.
- **Smoothness Range:** To save resources, reduce the size between the Value Range or adjust the Start and Distance values of the Fade Range. This adjusts the distance from the camera where water surface detail begins to diminish.
- **Refraction Strength:** Lower the Maximum Distance property to save memory, which also decreases the visibility of the refraction effect.
- **Resolution:** Opt for lower resolutions for masks, caustics, and the simulation. Adjust settings in **Project Settings > Quality > HDRP** for **Simulation Resolution, Caustics Resolution**, and the resolution of source files for masks, decals, and custom foam textures.

Tweaking these settings can ensure smooth performance across different devices.

Remember to take advantage of the Debug Mode in the Miscellaneous section of each water surface when troubleshooting performance. This tool aids in visualizing attributes like current direction, deformation, foam, and applied masks.

More water system demos

While the sample available from the HDRP package should get you through the basics, we've also created a GitHub repository that shows the water system interacting with a more complex environment. This project includes large meshes and textures, a full post-process stack, and other elements typically found in a production example for a real game application.

This allows you to see water rendering in context, rather than in isolated samples. Clone the repository or download a zipped version using GitHub.

Island scene

This scene uses an infinite water surface to simulate the ocean. In addition, this scene uses water deformers and foam generators to improve the visual around the shoreline. There are also examples of how to use custom render textures to generate water deformers.



The Island demo scene

Navigating the demo scenes

Many of the demos feature a third-person controller to explore the environment. Use the WASD or arrow keys to move and the spacebar to jump.

Alternatively, use a gamepad to move the third-person controller.

River scene

This scene uses instanced quads for the water surface in addition to a current map to simulate the flow. In play mode, there are two fixed cameras. Use the tab key to switch between them.



The River demo scene

Pool scene

This scene demonstrates the pool parameters and how to use custom render textures for water deformers. In Play mode, a third-person controller allows you to walk the environment.



The Pool demo scene

Next steps

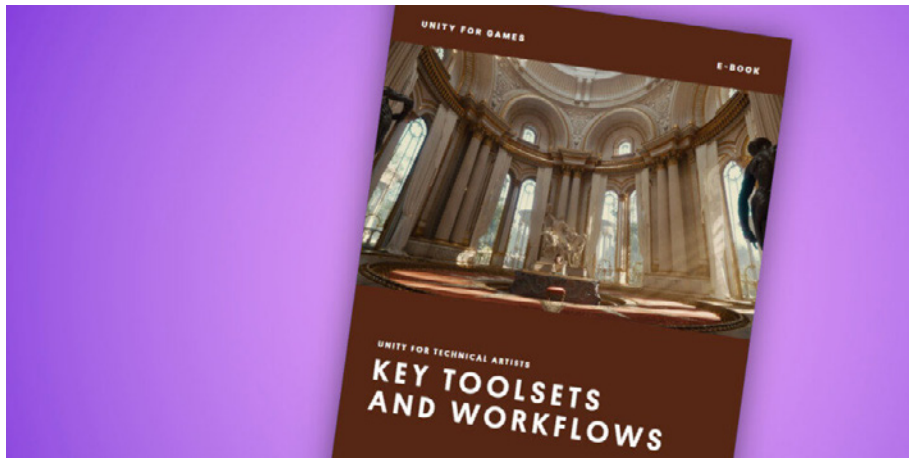
We hope this guide inspires you to try HDRP for your next project. Remember that the **3D Sample Project** is available from the Unity Hub when you want to explore further. If you have questions or feedback on this guide, please let us know in this [forum thread](#).

Be sure to check out the additional resources listed below, and you can always find tips on the [Unity Blog](#) or [HDRP community forum](#).

We want to empower artists and developers with the best tools to build real-time content. Remember that building game worlds and environments is both an art and a science – and where these meet, it's a little bit like magic.

More resources

- For a video introduction to HDRP's features, we recommend watching [“Achieving high-fidelity graphics with HDRP.”](#)
- If you're migrating from the Built-in Render Pipeline, see [this chart](#) for a detailed feature comparison between the two render pipelines.
- The [HDRP documentation](#) includes a complete breakdown of every feature in the pipeline.
- Delve into HDRP settings for enhanced performance with [this blog post](#).
- For more about the new water system, see [this blog](#) post or the water system [documentation](#). Also, be sure to watch this [introductory video](#) for an overview of its features.
- Want to learn more about [Unity development](#) and [technical art](#)? Access the [Unity best practices hub](#) for articles and e-books that provide a wealth of actionable tips and best practices to help you achieve more in less time.



Learn more from our technical e-books for programmers, artists, technical artists, and designers.



unity.com