→ E-BOOK

Lighting and environments in the High Definition Render Pipeline

(Unity 6 edition)

Unity®

Contents

Introduction

A game world is more than a backdrop – it's an interactive stage where every moment of your game unfolds.

When you start worldbuilding in Unity 6 with the High Definition Render Pipeline (HDRP), you can create breathtaking environments that push the boundaries of real-time graphics. HDRP delivers physically accurate lighting, high-fidelity materials, and cinematic effects to make your scenes more immersive.

Add depth to your scenes with volumetric fog and shadows. Make your landscapes come alive with SpeedTree's dynamic vegetation, while decal projectors age surfaces with dirt, cracks, and wear. Let ray tracing capture a new level of realism, from the soft sheen of brushed metal to the intricate dance of light through layered glass.

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 skies above with Cloud Layers and transition in real-time from day to night with Lighting Scenarios. HDRP's advanced water system can then fill in the oceans and rivers, complete with realistic waves, caustics, and foam.

What story will your world tell? Let HDRP help bring that vision to life.

HDRP allows you to build realistic game worlds. The top and bottom images are from Unity's high-end demo *Time Ghost*.

How to use this guide

This book is a modular reference guide, designed to help you navigate HDRP's extensive set of tools. You can jump directly to a topic of interest, whether it's lighting, environment effects, rendering techniques, or optimization strategies.

While each chapter is standalone, note that many topics are interconnected. For example, HDRP's lighting system spans multiple sections, covering physical light units, Volumes, and post-processing techniques.

Similarly, environment-related features like terrain, water, sky, and fog systems, often work together to create immersive worlds. To get the most out of this guidebook, try exploring related chapters together for a more complete understanding of HDRP's capabilities.

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 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 to go through the tutorial, Getting started with the High Definition Render Pipeline.

Time Ghost

Time Ghost is a real-time cinematic demo developed 2024 by the team behind projects like *The Blacksmith*, *Adam*, *Book of the Dead*, *The Heretic*, and *Enemies*.

This project demonstrates the latest advancements in HDRP, from improved lighting with Adaptive Probe Volumes (APVs) and Scenario Blending to performance enhancements using the GPU Resident Drawer. Leveraging Unity's Data-Oriented Technology Stack (DOTS), which includes the Unity Entities package built on Entity Component System (ECS) architecture, Time Ghost renders up to 12 million instances of vegetation.

Time Ghost used HDRP to create its realistic lighting.

The new **GPU Resident Drawer** in Unity 6 improves batching and instancing efficiency for rendering the demo's vast environments with minimal CPU overhead.

Time Ghost also makes extensive use of **VFX Graph** and **Shader Graph**. These tools empowered the team to create realistic explosions, dynamic fog, and intricate environmental effects.

VFX Graph powered Time Ghost's visual effects.

You can watch *Time Ghost* here. A behind-the-scenes session from Unite Barcelona 2024 offers a deep dive into the Unity Originals team's approach to lighting, environments, and character development.

Two sample packages on the Asset Store can help you explore the art and technology behind Time Ghost:

— **Time Ghost: Environment:** This package leverages **DOTS ECS** and procedural tools to manage a large number of scene entities and allocate resources.

— **Time Ghost: Character:** The demo's protagonist showcases **Sentis AI** for real-time character animation and physics-driven interactions.

These assets are available to download and can be imported into Unity 6 projects, allowing you to explore the advanced techniques featured in *Time Ghost*.

Time Ghost's main character is available for download.

HDRP Updates in Unity 6.1

HDRP in Unity 6.1 introduces several new features and improvements; here are the highlights for those moving from HDRP in 2022 LTS (which the previous version of this e-book is based on).

Lighting

LightBaker: HDRP's new backend LightBaker v1.0 takes a "snapshot" of the Scene state when starting a bake instead of continuously checking for scene changes every frame. The bake runs uninterrupted unless manually canceled, reducing unnecessary recalculations and improving Editor performance.

Baking Profile: The Lighting window now provides a GPU Baking Profile for the GPU Lightmapper. This feature divides lightmaps into tiles, with presets optimizing for either faster baking or lower memory usage.

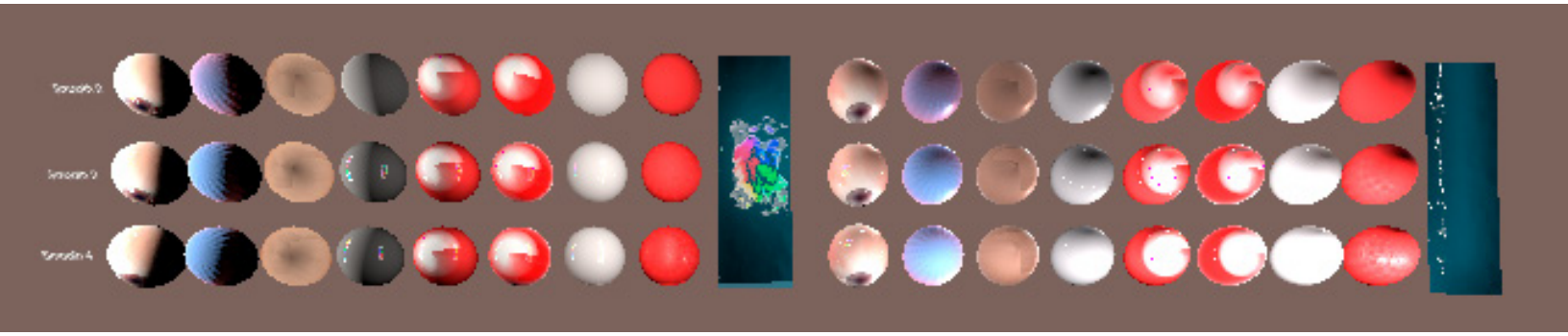
Adaptive Probe Volume (APV) updates: LightBaker now allows you to bake Light Probes and APVs independently from your lightmaps. APVs now also support sky occlusion baking, allowing GameObjects to adjust their lighting dynamically using the sky color from the ambient probe. This allows you to simulate a day-night cycle using APVs.

Ray tracing improvements: Ray tracing now includes some new APIs to provide more control over raytracing acceleration structures. The `RayTracingAccelerationStructure` includes some new methods:

- `UpdateInstanceGeometry` allows you to update the **Bottom-Level Acceleration Structure (BLAS)** manually using the new `RayTracingMode.DynamicGeometryManualUpdate` mode allows you to mark specific instances as "dirty" and then manually trigger an acceleration structure rebuild, rather than automatically updating the geometry each frame. This can improve performance for dynamic objects in ray-traced scenes.
- `AddInstancesIndirect` adds multiple ray tracing instances to the **Ray Tracing Acceleration Structure (RTAS)** using a GraphicsBuffer for transformation matrices. This allows the GPU to manage those instances, reducing CPU overhead and improving performance. Instances can be mesh-based or procedural.
- `RemoveInstances` enables removal of ray tracing instances based on LayerMasks or raytracing modes. This can optimize performance by dynamically updating the acceleration structure as the scene changes at runtime.
- `CullInstances` adds filtering and culling parameters, making it easier to update acceleration structures based on specific criteria.

Bicubic sampling for lightmaps: This feature can blur and smooth lightmaps, making them appear higher quality even at low-resolution. It's beneficial for large environments that use baked lighting, where memory and performance constraints may require lower-resolution lightmaps. Enable it in **Project Settings > Graphics > Use Bicubic Lightmap Sampling**.

Area light improvements: Area lights now use a "pillow" windowing function for range attenuation. This allows them to fade out more smoothly and work better with different material types while reducing CPU memory usage.



Area lights use a "pillow" windowing for range attenuation.

Path tracing improvements: Disc- and tube-shaped area lights are now supported with path tracing. **Box lights** have also been fixed to ensure correct illumination behavior. Additionally, **volumetric fog controls** have been added to area lights, allowing you to fine-tune their influence for more realistic atmospheric effects.

Path tracing supports disc and tube-shaped area lights.

Environment effects

Planet parameters: A shared set of parameters in the Visual Environment override now affect various environment effects like volumetric clouds, fog and physically based sky.

Physically Based Sky Volume: The PBR Sky Volume has been optimized by removing some precomputation steps, allowing real-time sky updates with minimal performance impact. A new ozone layer has been added, improving sky color accuracy near the horizon for more realistic twilight effects.

The ozone layer enhances color accuracy near the horizon.

Also, aerial perspective can now simulate light absorption by particles in the atmosphere when looking at distant objects, such as mountains or clouds.

The Physically Based Sky Volume can simulate atmospheric scattering.

Volumetric Clouds: Clouds are no longer clipped by the far plane, and earth curvature control has been centralized in the Visual Environment settings.

Water system: The water system enhances support for GameObject transforms on water surfaces, including translation, rotation, and negative scaling. Unity 6.1 introduces a Water Decal target for Shader Graph to output foam and horizontal water deformations (like rolling waves).

Volumetric Fog: Volumetric lighting performance has been optimized.

Performance and optimizations

Dynamic resolution: Unity 6.1 introduces **Spatial-Temporal Post-Processing (STP)**, a new, high-quality image upscaling technique that combines spatial upscaling (enhancing details from nearby pixels) with temporal upscaling (using previous frames to refine edges).

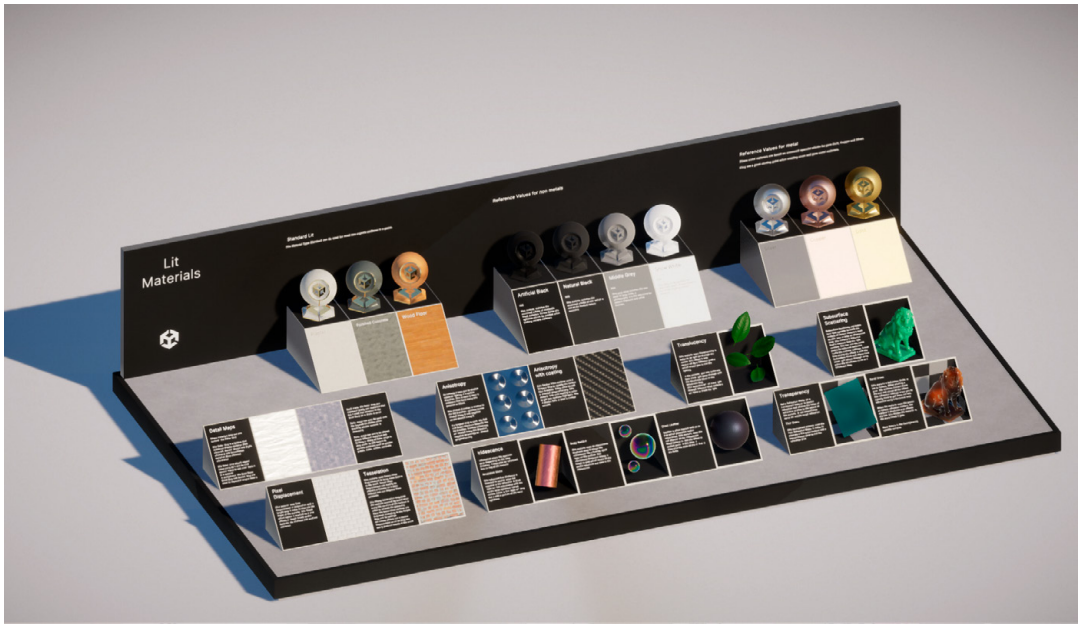
GPU Resident Drawer: The new GPU Resident Drawer leverages the Batch Render Group API for significant performance boosts when rendering complex scenes.

Variable Rate Shading (VRS): Variable Rate Shading (VRS) now supports Custom Passes, allowing dynamic shading resolution adjustments in specific screen areas to reduce GPU workload while maintaining visual quality.

Package Manager samples

The HDRP sample projects in the Package Manager have been reworked for consistency, with enhanced explanations and shared resources across pipelines:

- **Lit Material samples:** All materials in this scene use the **HDRP/Lit shader**, demonstrating various material types (e.g., Standard, Subsurface Scattering, Anisotropy, etc.). Each sample highlights how HDRP handles realistic lighting over different surfaces.
- **Transparent Material samples:** This scene demonstrates how HDRP handles transparency, including refractive materials, stacking transparent objects, and shadows from transparent materials. Switch between rasterization, ray tracing, and path tracing to explore the different rendering techniques.
- **Volumetric samples:** This scene showcases volumetric fog, 3D textures, Shader Graph effects, and blending modes. The samples show how to recreate several fog effects, procedural noise, and smoke elements.
- **Environment samples:** This island scene demonstrates several different cloud setups over an island terrain and ocean water system.
- **Water samples**: This project includes a new water **Cave** sample scene. This scene features Custom Passes and Local Volumetric Fog sampling the water system caustics buffer to showcase advanced water rendering techniques. The **Rolling Wave** scene shows how to implement horizontal deformations, enabling effects like rolling waves.



The Package Manager includes sample packages.

Lens flares

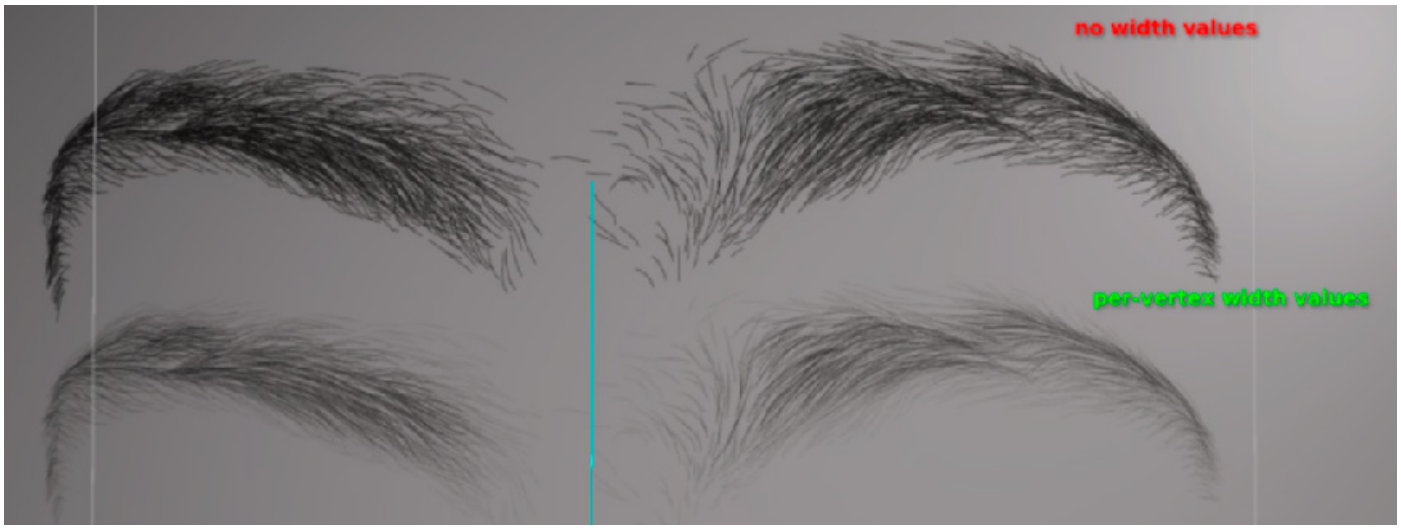
In Unity 6, lens flares have been enhanced with several new features to improve visual fidelity and workflow:

- **Improved XR support:** Lens flares now offer better compatibility with XR applications, ensuring consistent flare effects across various devices.
- **Ring Procedural Shape:** A new ring-shaped procedural flare allows you to create concentric circular flares, useful for bright light sources.
- **Recursive Lens Flare assets:** Lens flare assets can now be elements within other lens flares, enabling more complex and layered flare effects.
- **Radial Gradient support:** Procedural shapes now support radial gradients, providing more natural-looking flares.
- **Light Override functionality:** A new light override feature allows multiple flares to be influenced by a single light source. This streamlines setting up lens flare effects for multiple lights.

Unity 6 enhances lens flares.

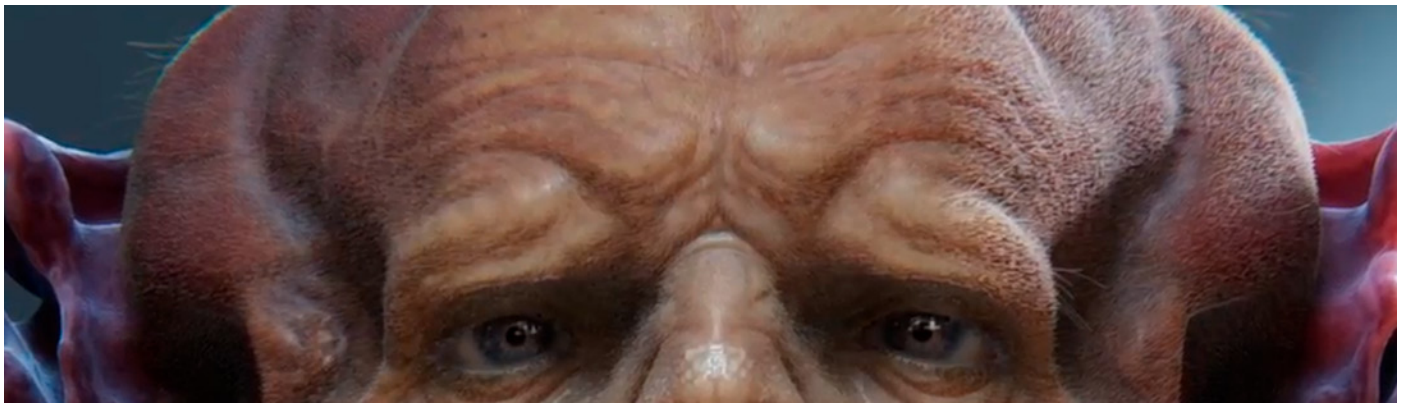
Hair

The **High-Quality Line Rendering Volume override** now allows users to define hair width in centimeters per vertex, improving control and accuracy.



Per-vertex width values improve control and accuracy.

Here's an example of creating peach fuzz using a uniform width of 0.01 cm.



Per-vertex width values with uniform width of 0.01 cm creates peach fuzz.

Additionally, a new screen coverage-based LOD mode dynamically culls hair strands based on viewport visibility. Artists can use an animation curve to control strand density, with the x-axis representing screen space coverage and the y-axis defining the percentage of strands rendered. This improves performance and visual quality.

Installing HDRP

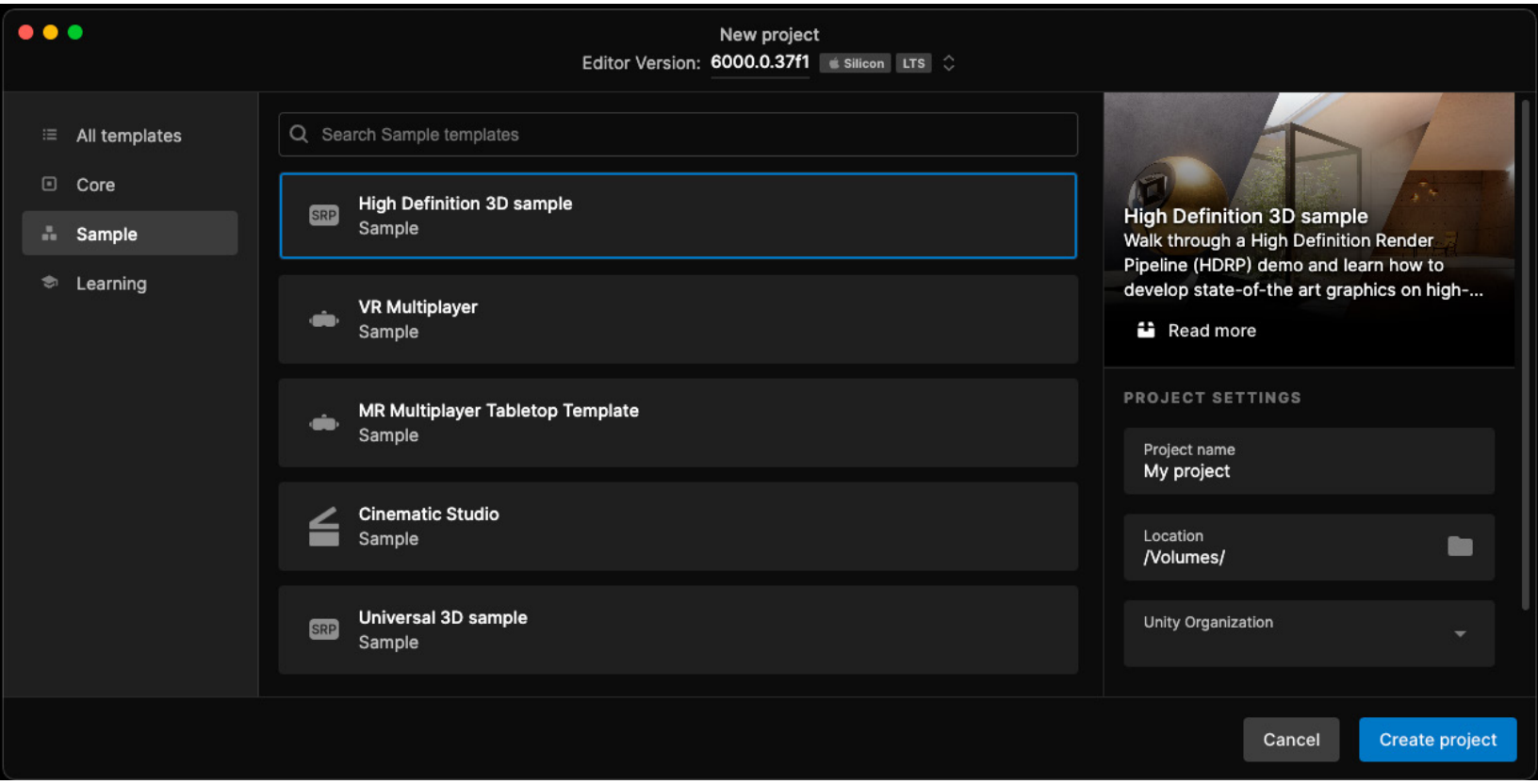
To set up HDRP in your project, start with an **HDRP Template Project** or manually install the **High Definition RP package**.

The HDRP template includes physically based lighting setups, decals, Volumes, and accurate materials, making it a great starting point for high-end visual projects.

Creating an HDRP project with the template

Open **Unity Hub** and go to the **Projects** tab. Click **New Project** and navigate to the **Sample** tab and select **High Definition 3D Sample Scene** as your template.

Enter a **Project Name** and click **Create Project**. Unity will automatically install HDRP and its dependencies.



Create an HDRP project from the Hub.

Installing HDRP manually

If you want to add HDRP to an existing project, open your Unity project and navigate to **Window > Package Manager**.

In the **Package Manager**, select **Unity Registry** from the drop-down menu. Find and select **High Definition RP** from the package list.

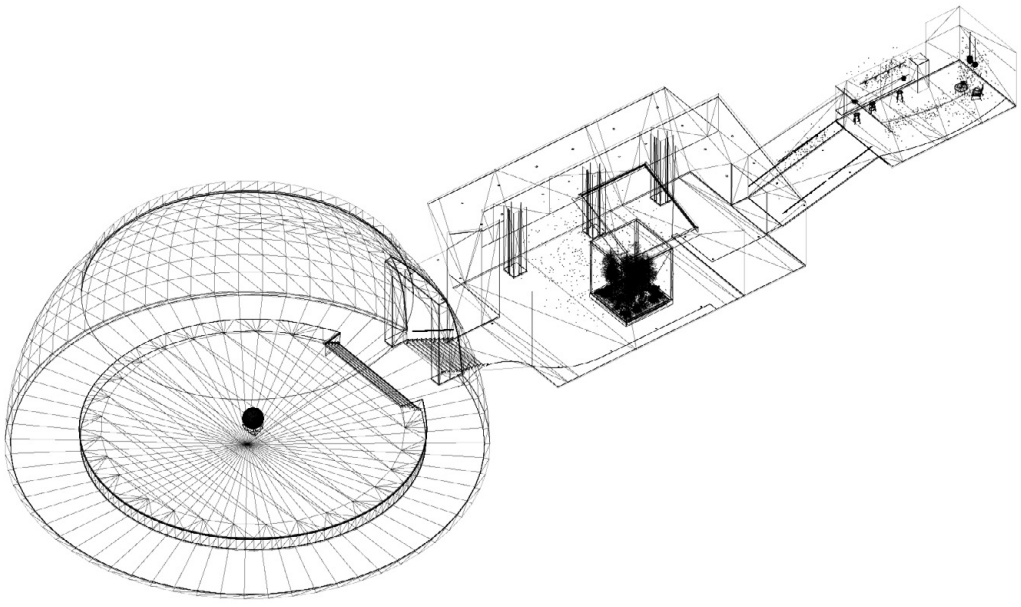
Click Install. If you encounter errors, use the HDRP Wizard to fix configuration issues.

For more information about setting your project for HDRP, see Getting Started with the High Definition Render Pipeline in the documentation.

HDRP sample scene

The 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.

— **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 vegetation inside the glass case.

— **Room 3** showcases interior artificial lighting and emissive materials.

The sample scene consists of three rooms.

For a deeper look at the HDRP 3D Sample Scene, check out this blog post, which describes the template scene in more detail.

More HDRP sample content

Let's look at other projects you can find helpful once you're done exploring the samples.

The Time Ghost Environment sample package contains one of the environments created for the Unity cinematic demo, *Time Ghost*. Get a closer look at how the Unity Originals team used the Entities package in DOTS and additional tooling for the creation of this environment scene. The demo leverages the latest improvements in Unity 6 and HDRP, including APV lighting, Scenario Blending, and the GPU Resident Drawer.

The *Time Ghost* Environment

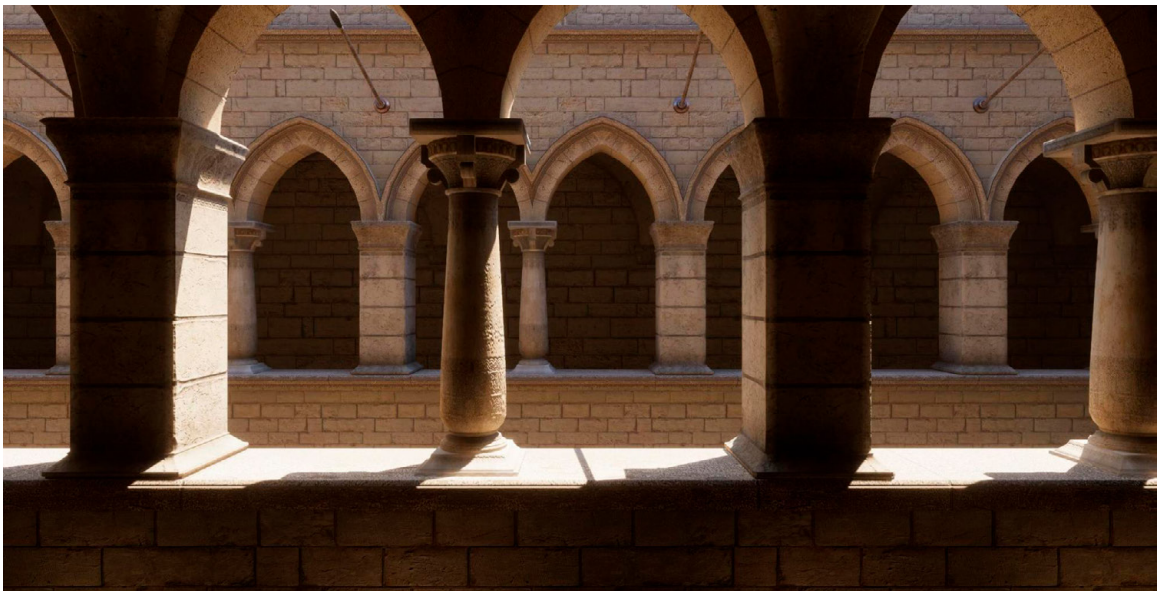
The *Time Ghost*: Character sample package features the character created for *Time Ghost*. This character demonstrates a machine learning model for cloth deformation and builds on the Hair System seen in The Heretic and Enemies. Learn more about Making of Time Ghost in this Unity Discussions post.

The *Time Ghost* character features cloth.

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.

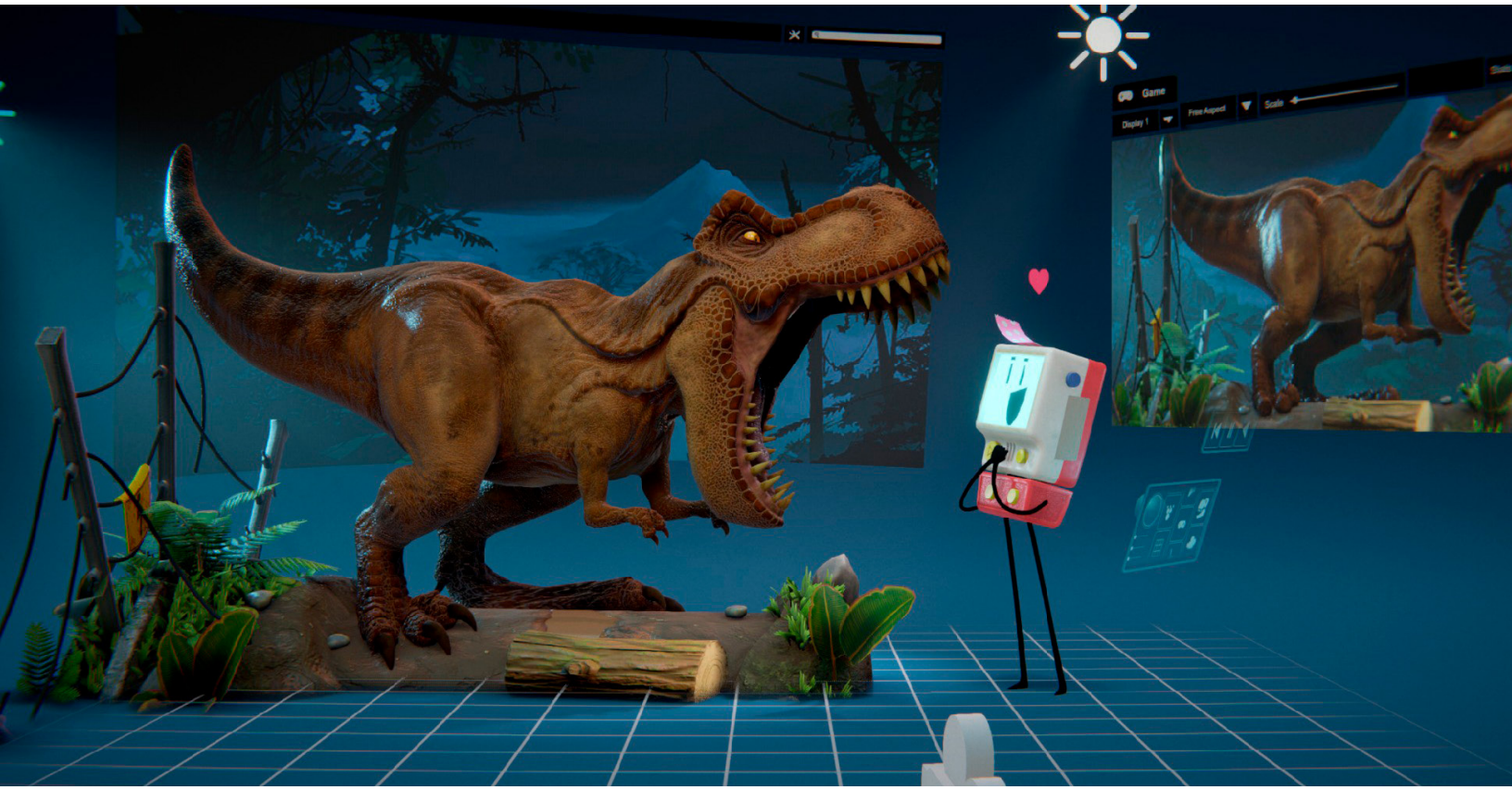
The Enemies demo project shows off an elaborate interior.

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

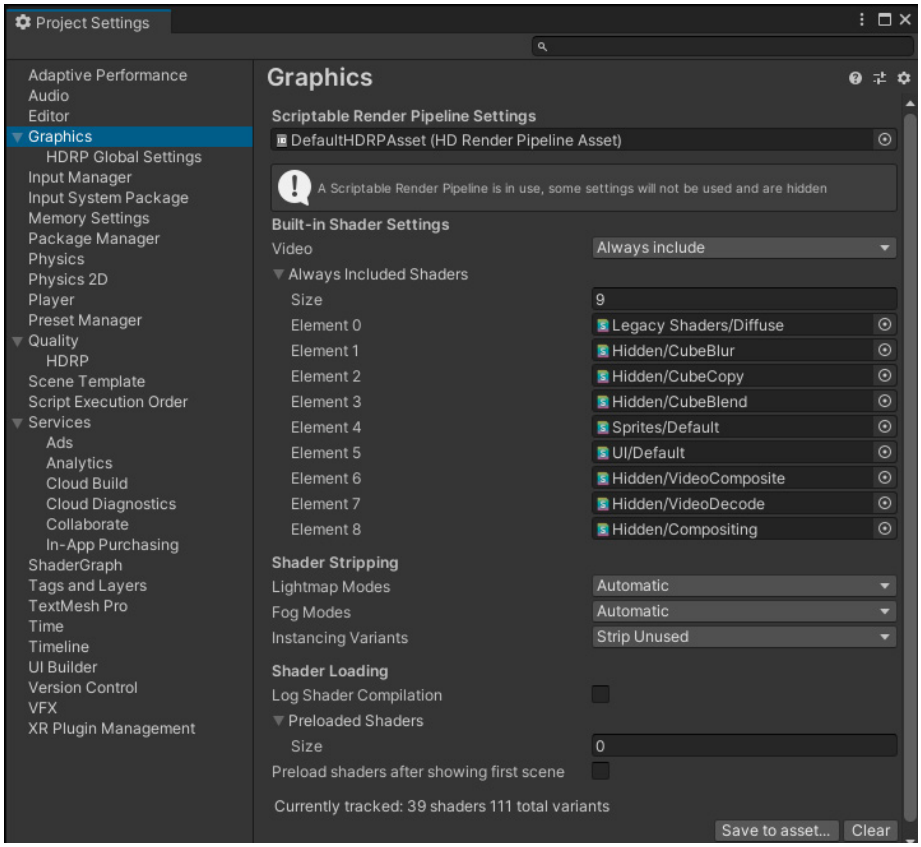
Install the Cinematic Studio Template from the Asset Store to learn how to make cinematics or animated films. The template teaches how to set up and light shots using a funny short movie called Mich-L, which mixes stylized and photoreal rendering.



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**.



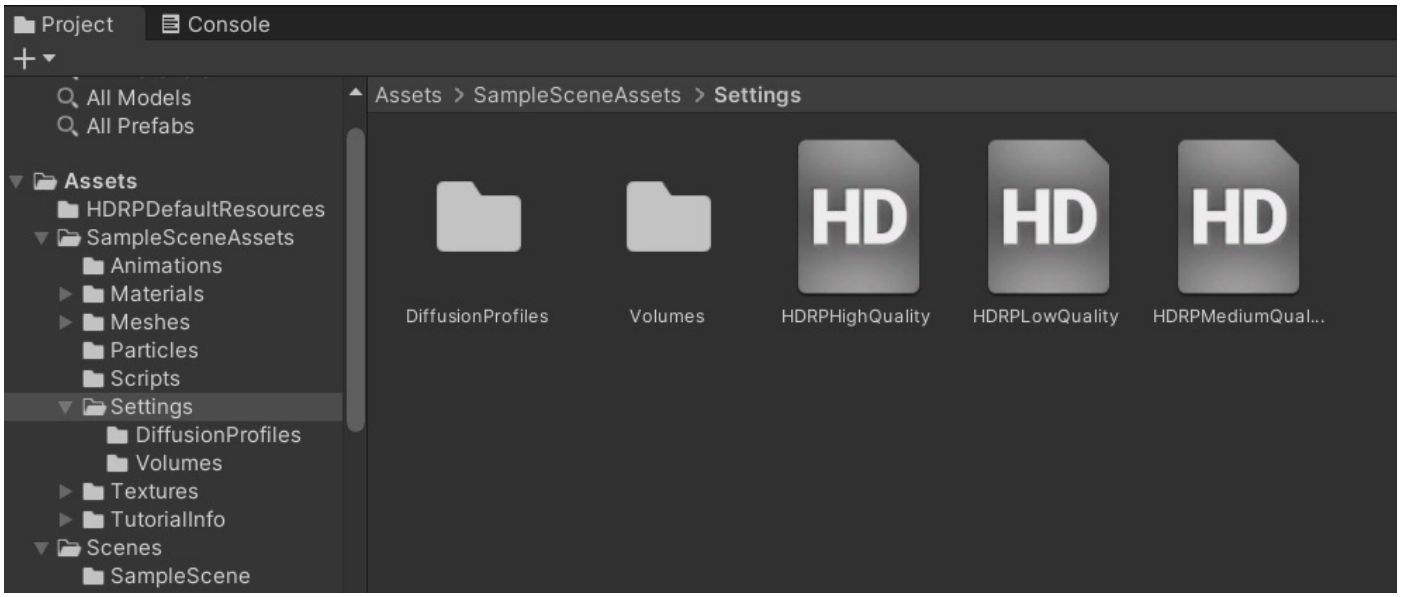
Project Settings

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, or they could also represent different visual quality levels that the player could swap at runtime.

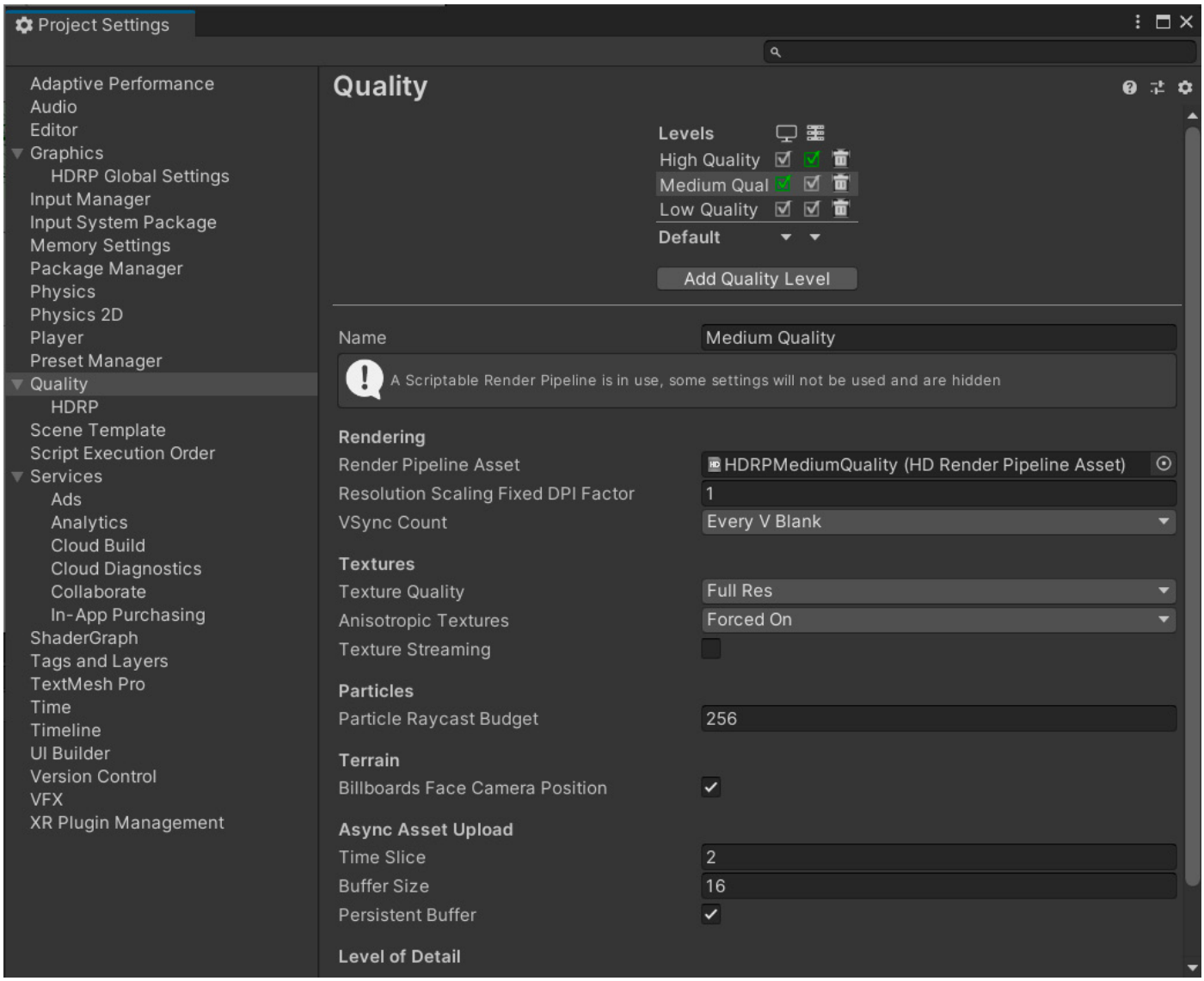
The **3D sample scene** begins with several Pipeline Assets in the Settings folder: **HDRPHighQuality**, **HDRPLowQuality**, and **HDRPMediumQuality**. There is also a **HDRPDefaultResources** folder containing a **DefaultHDRPAsset**.



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.



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).

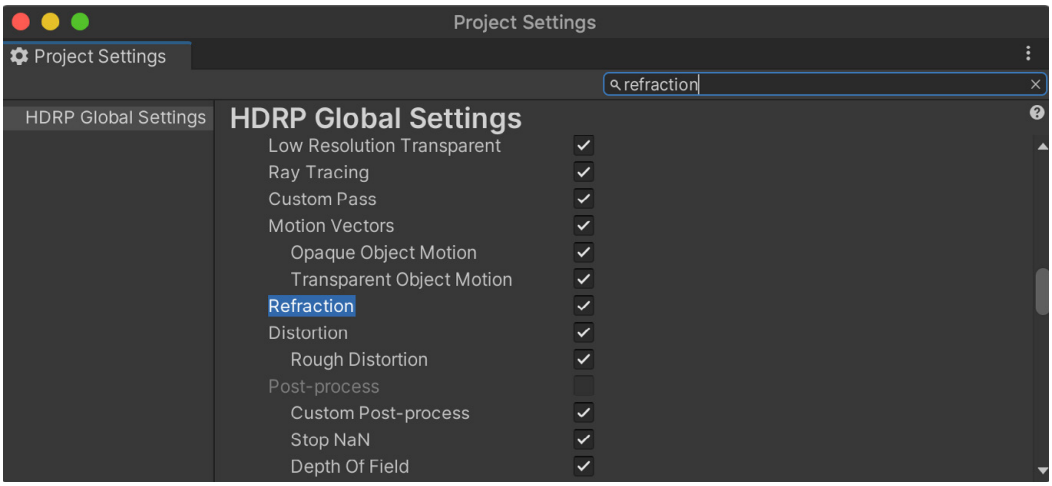
Global Settings save in their own separate Pipeline Asset defined at the top field. Set up the default rendering and post-processing options here.

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.



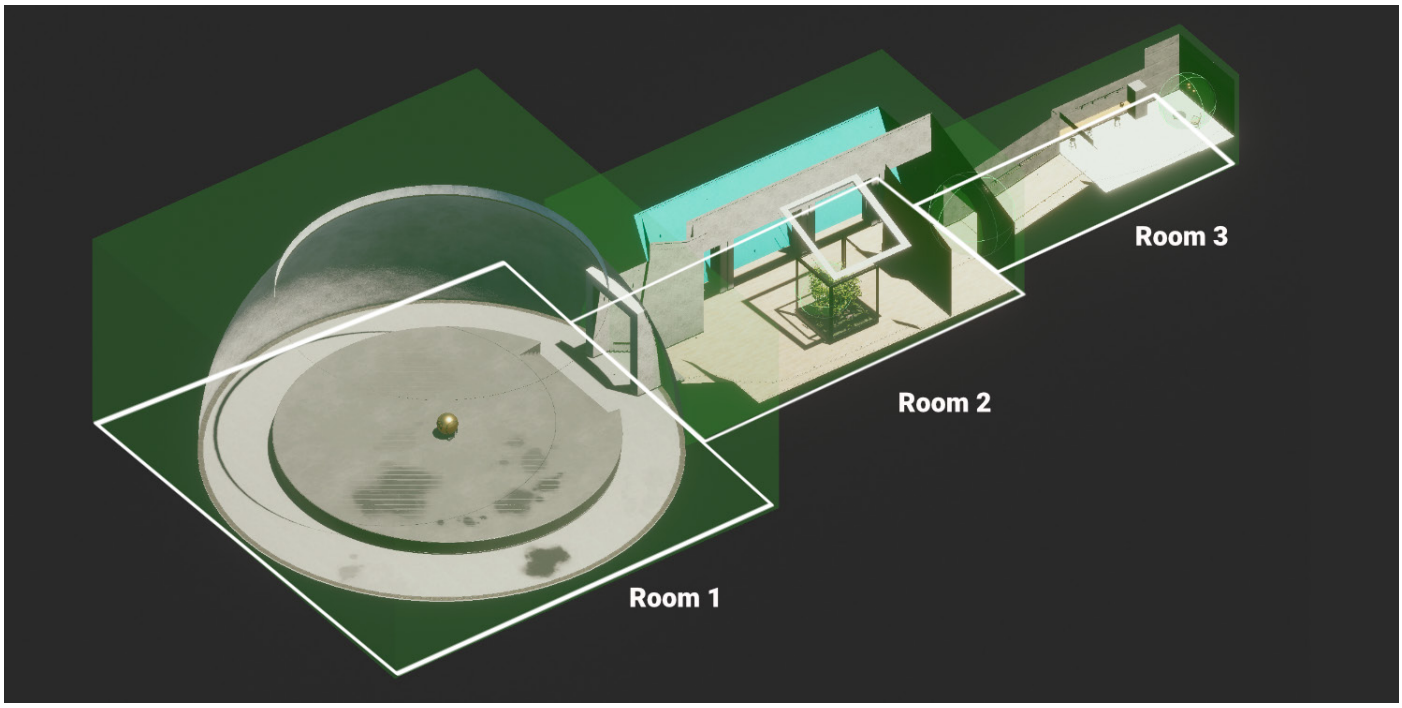
Search for HDRP features.

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 > Volumetric**s.

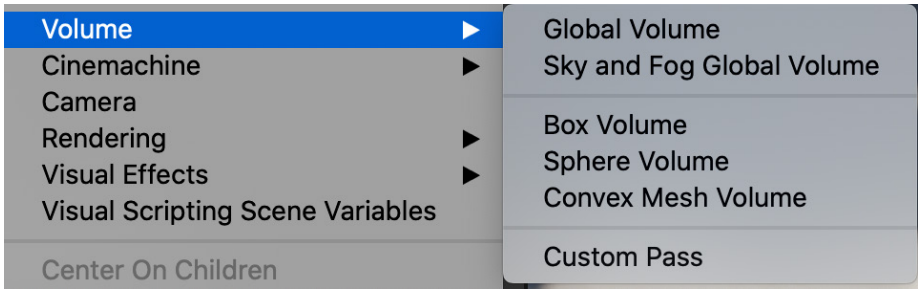
Volumes

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.



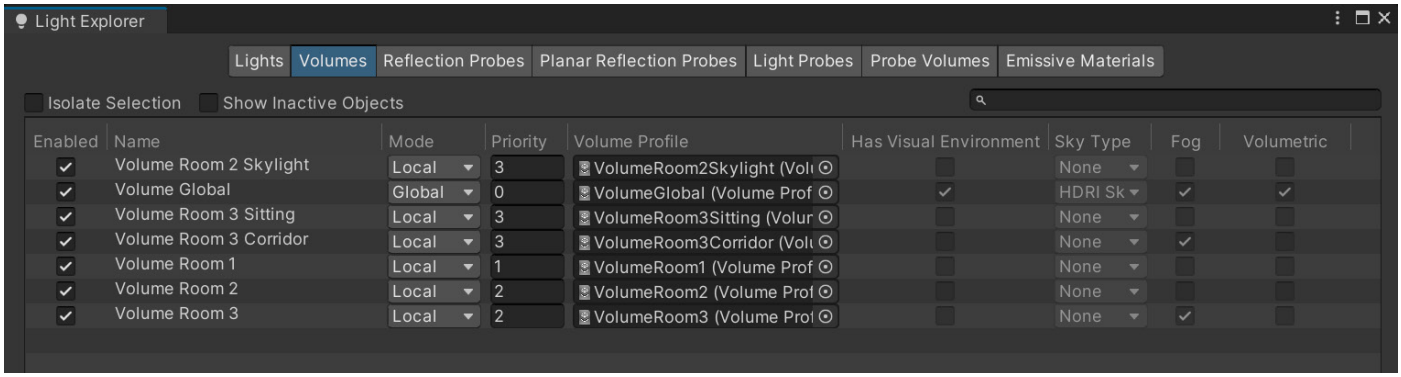
Volumes cover spaces with distinct lighting conditions.

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.

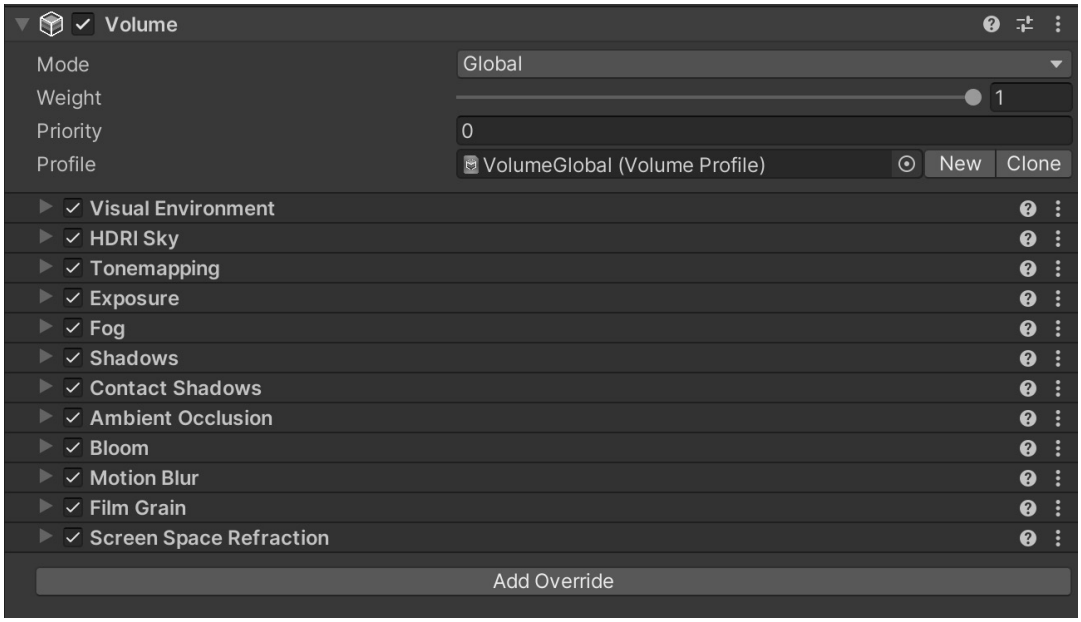


The Light Explorer can list all the Volumes in the open scene(s).

Local and Global

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

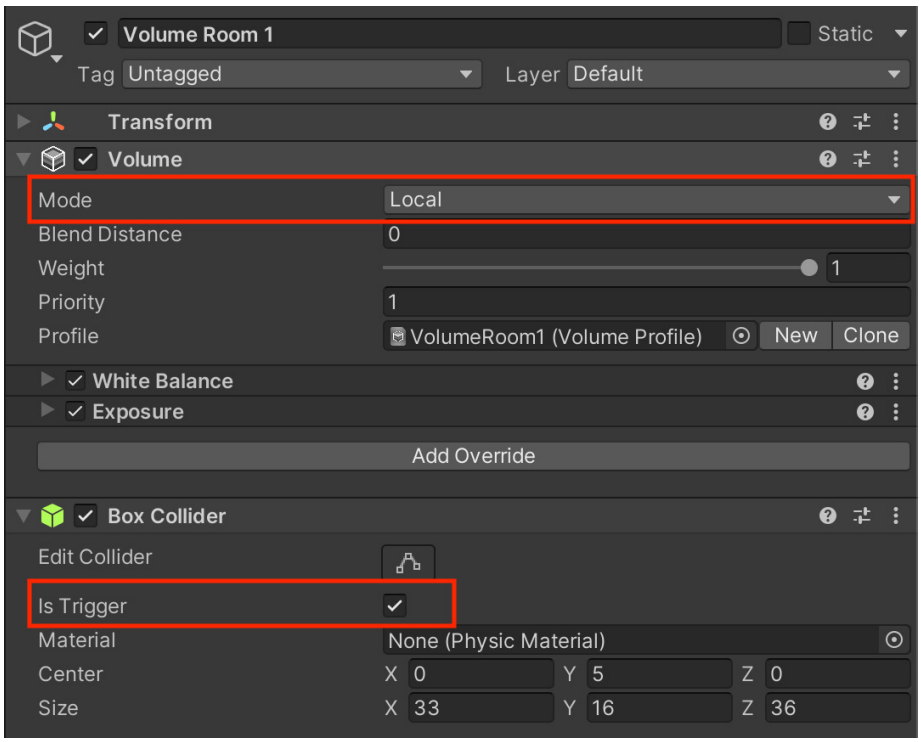
A global Volume works as a "catch-all" without any boundaries, and it affects all cameras in the Scene. In the HDRP template scene, the VolumeGlobal defines an overall baseline of HDRP settings for the entire level.



The Global Volume overrides

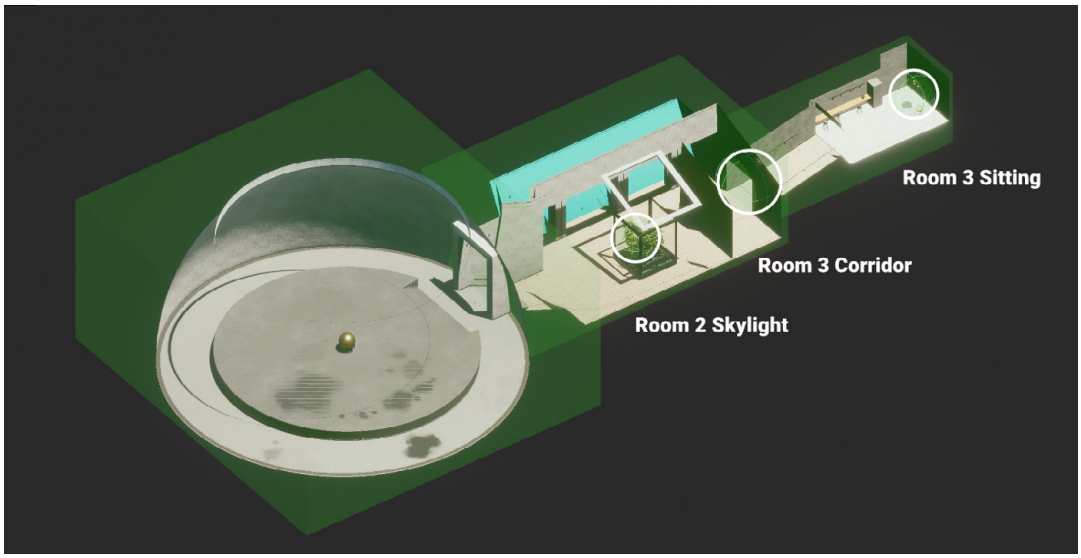
A local Volume defines a limited space where its settings take effect. It uses a Collider component to determine its boundaries. Enable **IsTrigger** if you don't want the Collider to impede the movement of any physics bodies like your FPS player controller.

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.

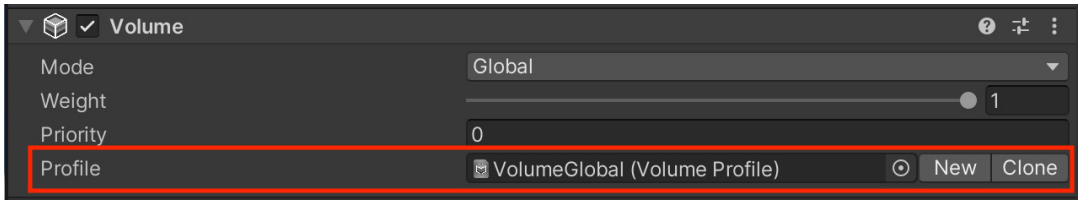
As your camera moves around the scene, the global settings take effect until your player controller bumps into a local Volume where those settings take over.

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.

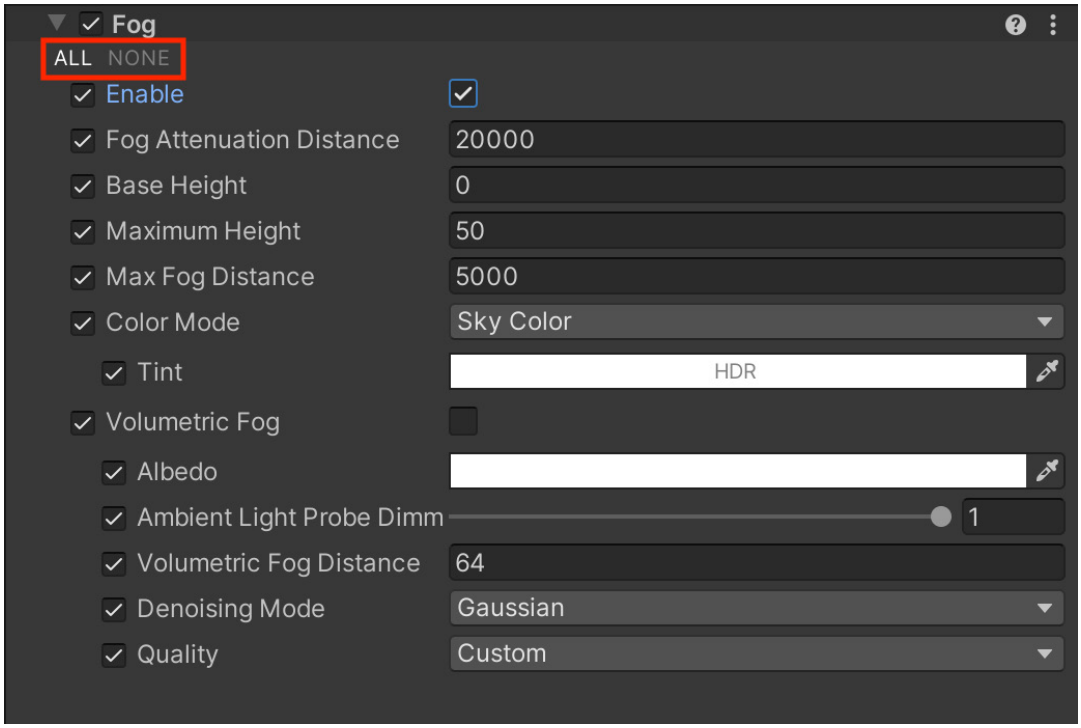
You can also switch to another Profile you already have saved. Having the Volume Profile as a file makes it easier to reuse previous settings and share Profiles between your Volumes.

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

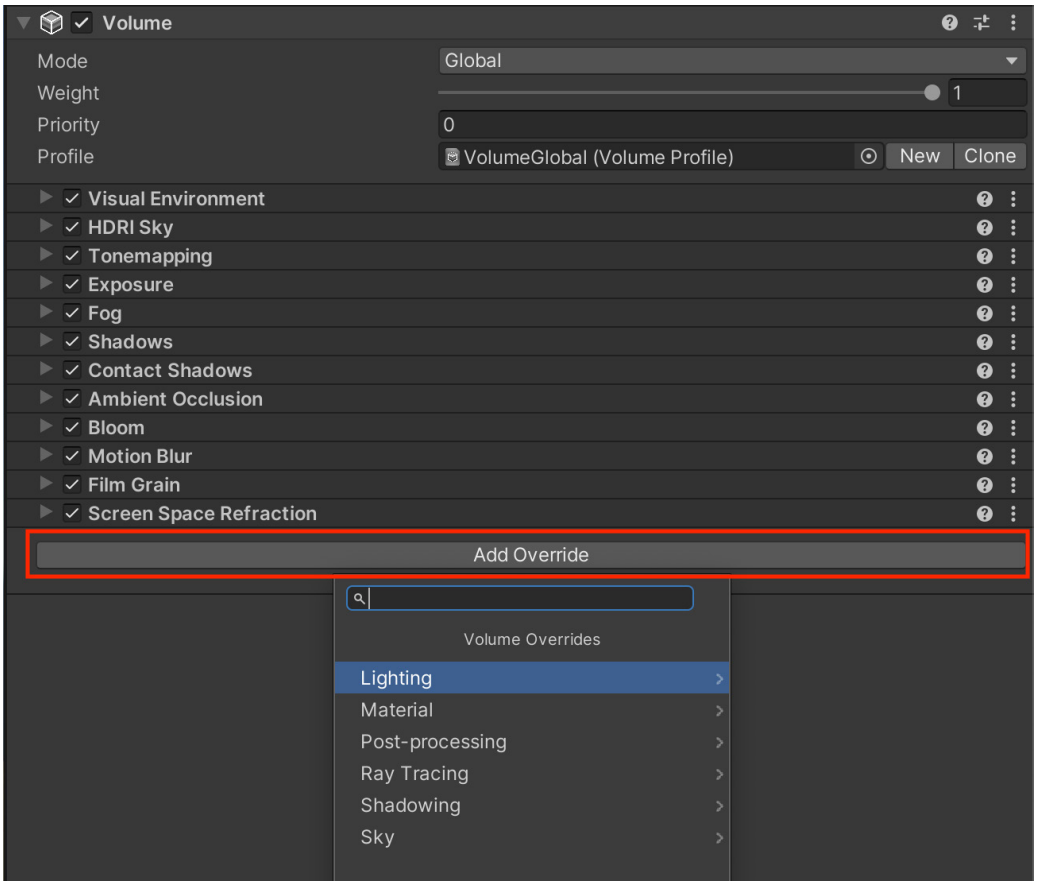
Each of the Volume Override's properties has a checkbox at the left, which you can enable to edit that property. Leaving the box disabled means HDRP uses the Volume's default value.

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.

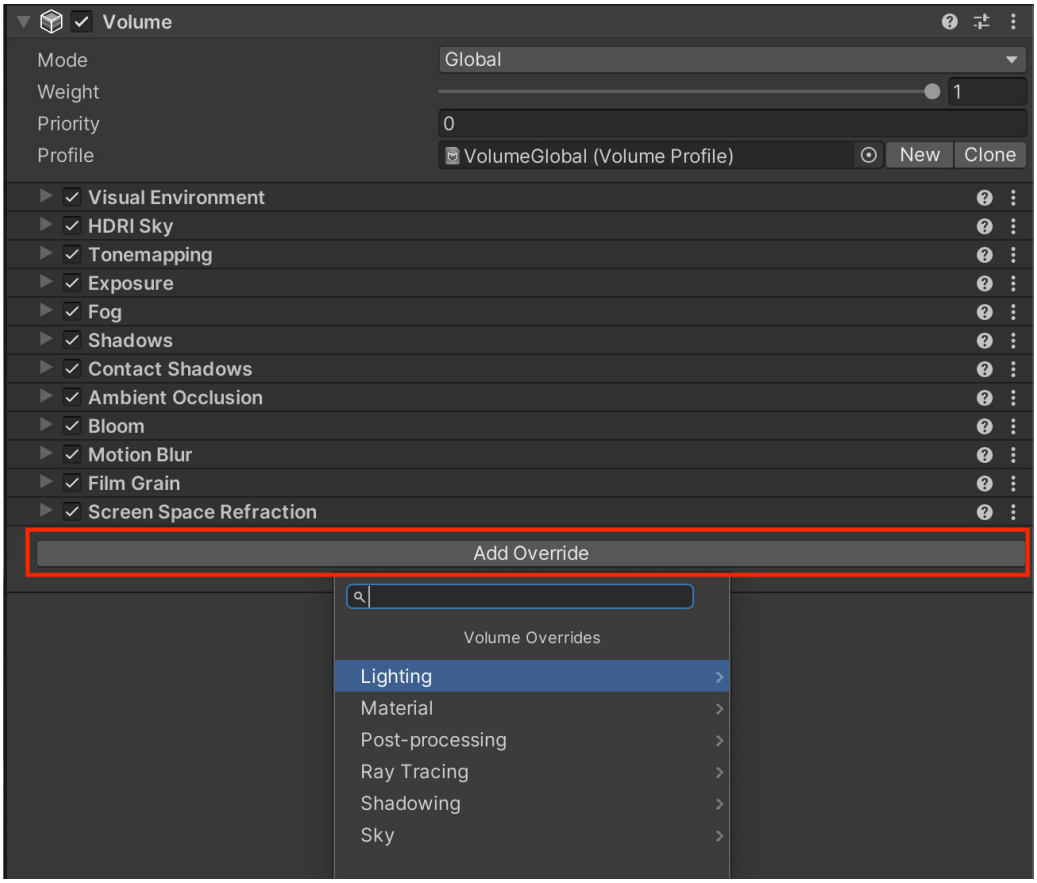
The higher-level Volume settings serve as the defaults for lower-level Volumes. Here, the HDRP Default Settings pass down to the global Volume. This, in turn, serves as the “base” for the local Volumes.



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

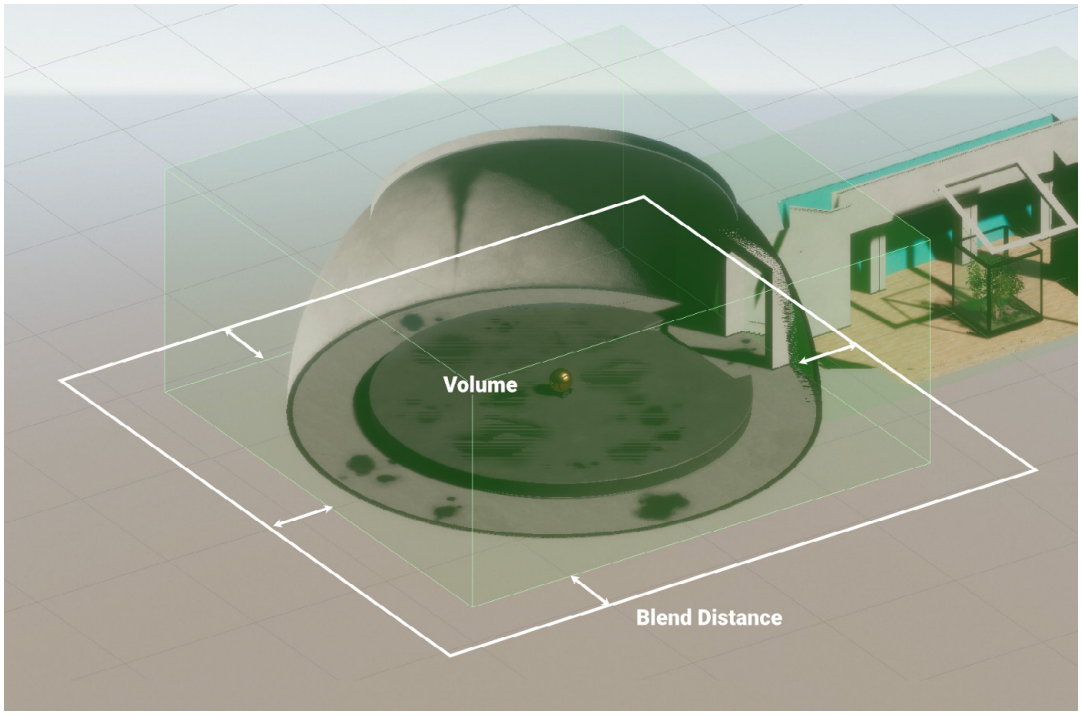
See the Volume documentation page in the HDRP documentation for more details.

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.

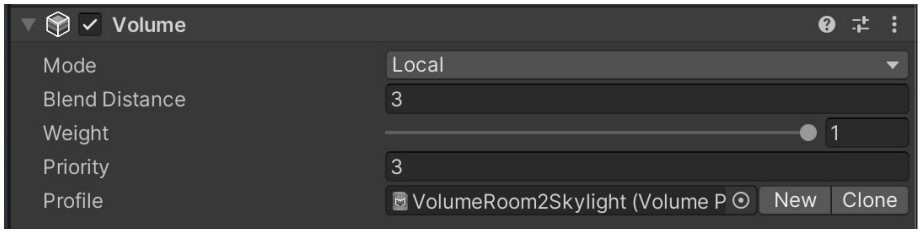
Blend Distance determines how far outside the Volume's Collider to begin fading on or off. A value of 0 for Blend Distance means an instant transition, while a positive value means the Volume Overrides begin blending once the camera enters the specified range.



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.



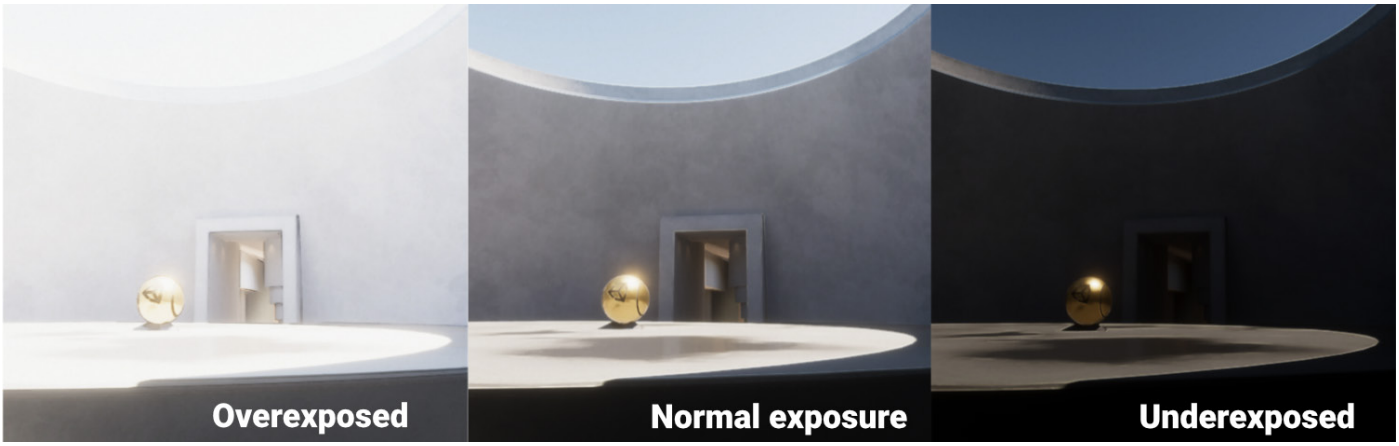
Use Blend Distance, Weight, and Priority when overlapping local Volumes.

Exposure

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

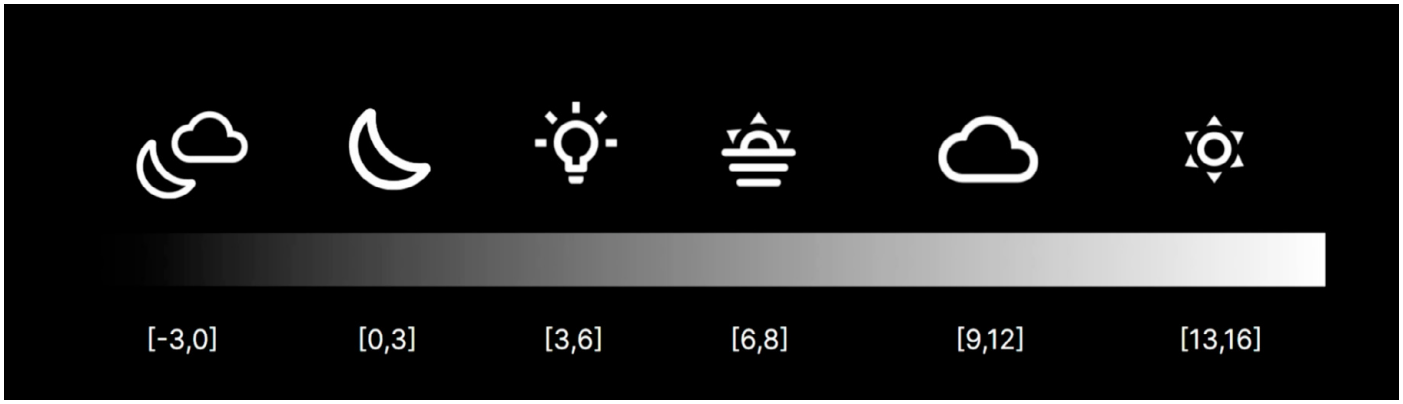
Understanding exposure value

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.

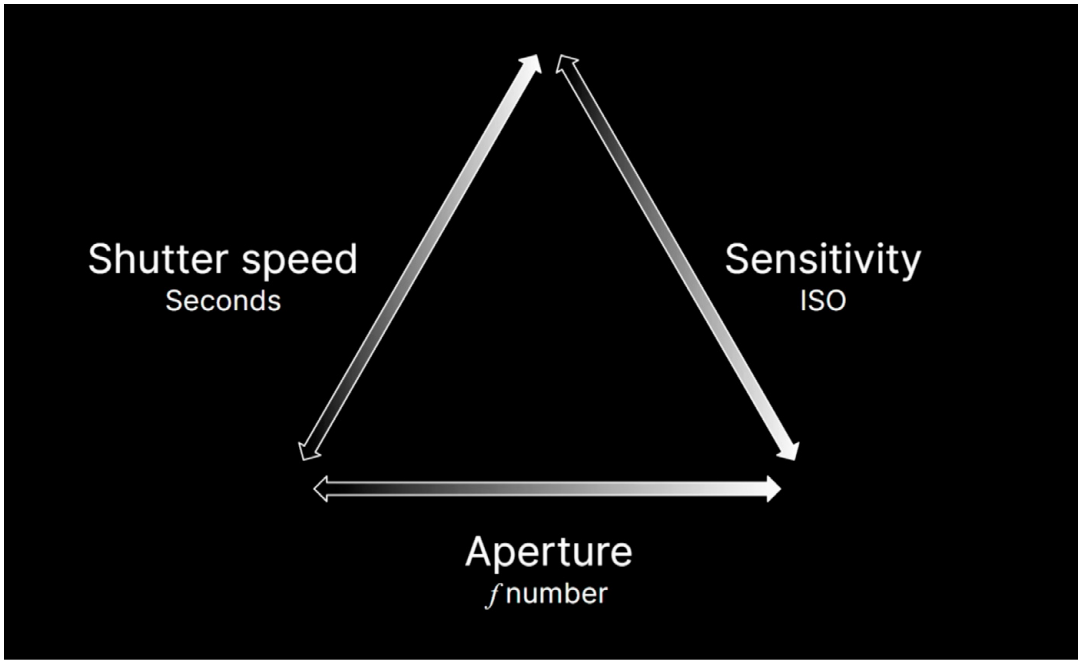
You can vary a number of factors in an actual camera's settings to modify your Exposure value:

— 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 EV100, which fixes the sensitivity to that of 100 International Standards Organization (ISO) film.

Exposure value formula

This formula actually calculates exposure value.

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

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

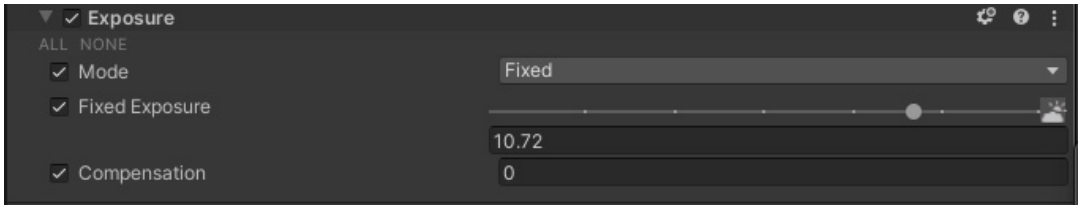
In HDRP, Exposure is a Volume Override. Add it to a local or global Volume to see the available properties.

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.

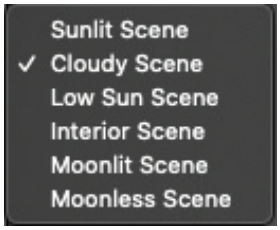
Fixed mode

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

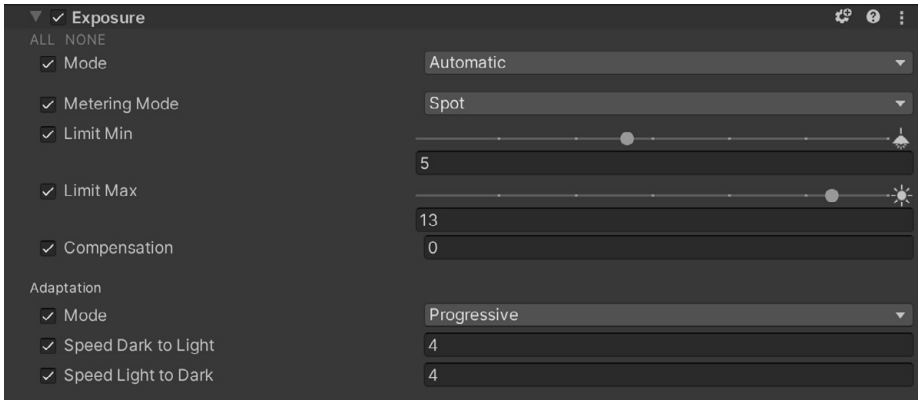
Fixed mode is simple but not very flexible. It usually only works if you have a Volume or Scene with relatively uniform lighting, where one exposure value can work throughout.

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.

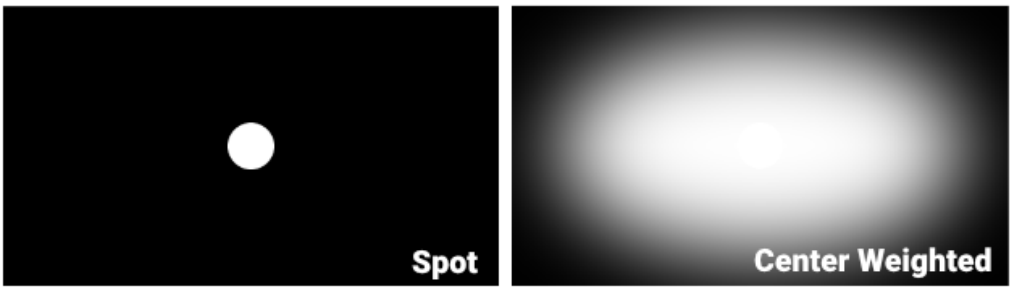
The **Adaptation mode** controls how the autoexposure changes as the camera transitions between darkness and light, with options to adjust the speed. Just like with the eye, moving the camera from a very dark to a very light area, or vice versa, can be briefly disorienting.

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.

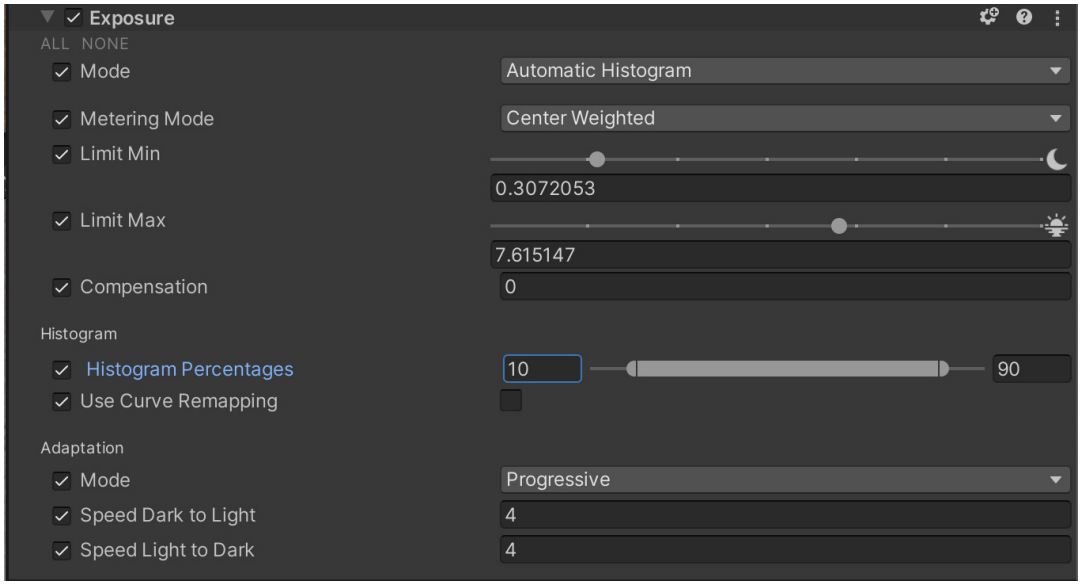
- **Spot:** The camera only uses the center of the screen to measure exposure.
- **Center Weighted**: The camera favors pixels in the center of the image and feathers out toward the edges of frame.
- **Mask Weighted:** A supplied image (Weight Texture Mask) determines which pixels are most important 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.

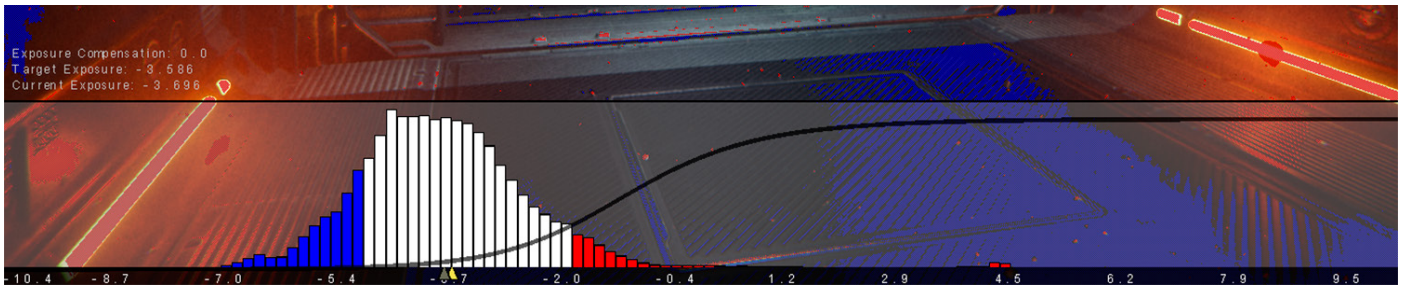


Automatic Histogram mode

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).

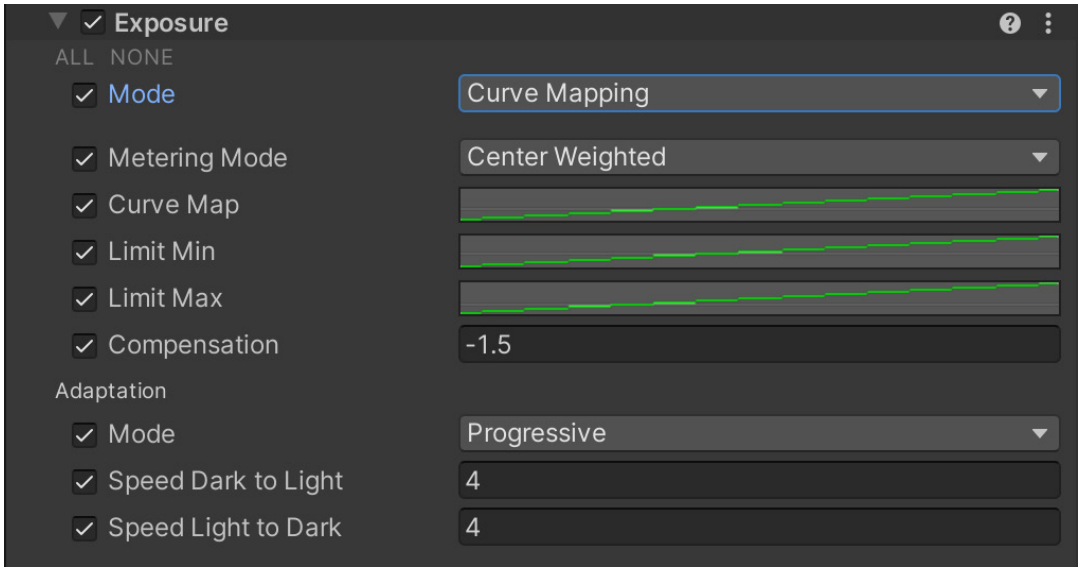
Curve Remapping lets you remap the exposure curve as well (see Curve Mapping below).



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

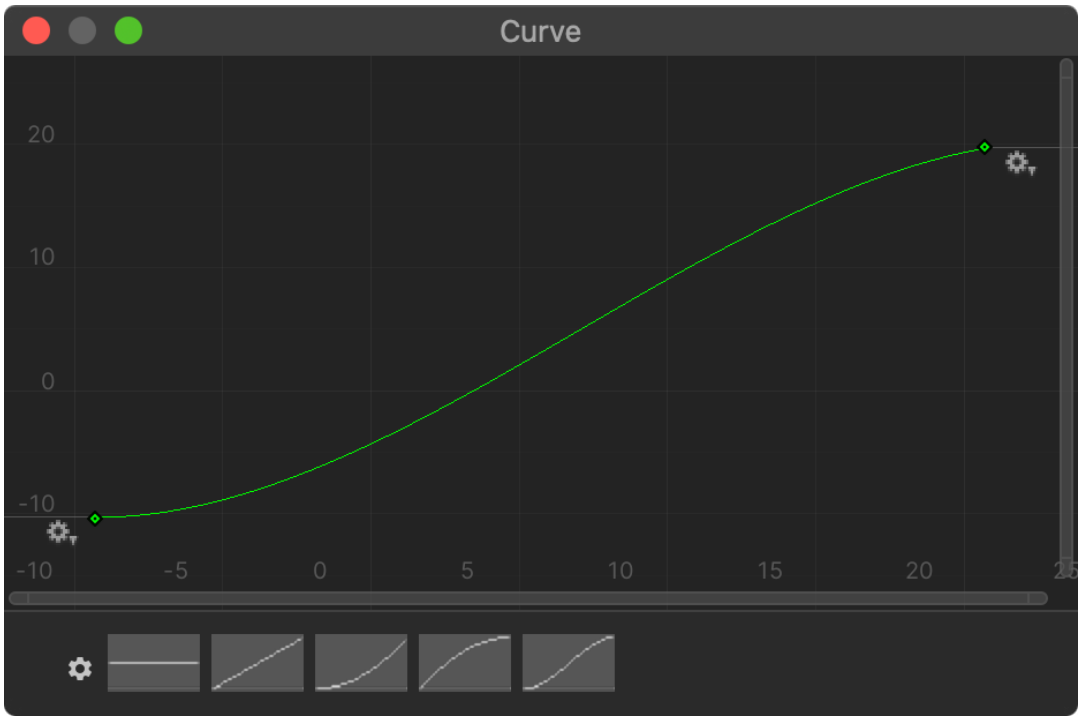
Curve Mapping

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.

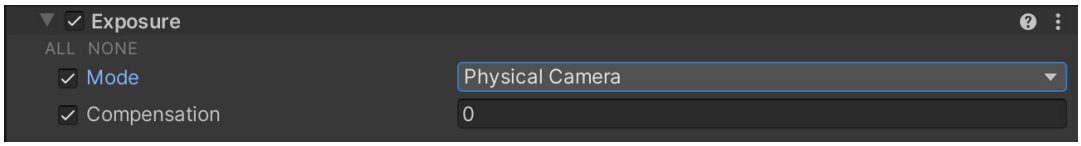


Adjust the exposure using a curve.

Physical Camera

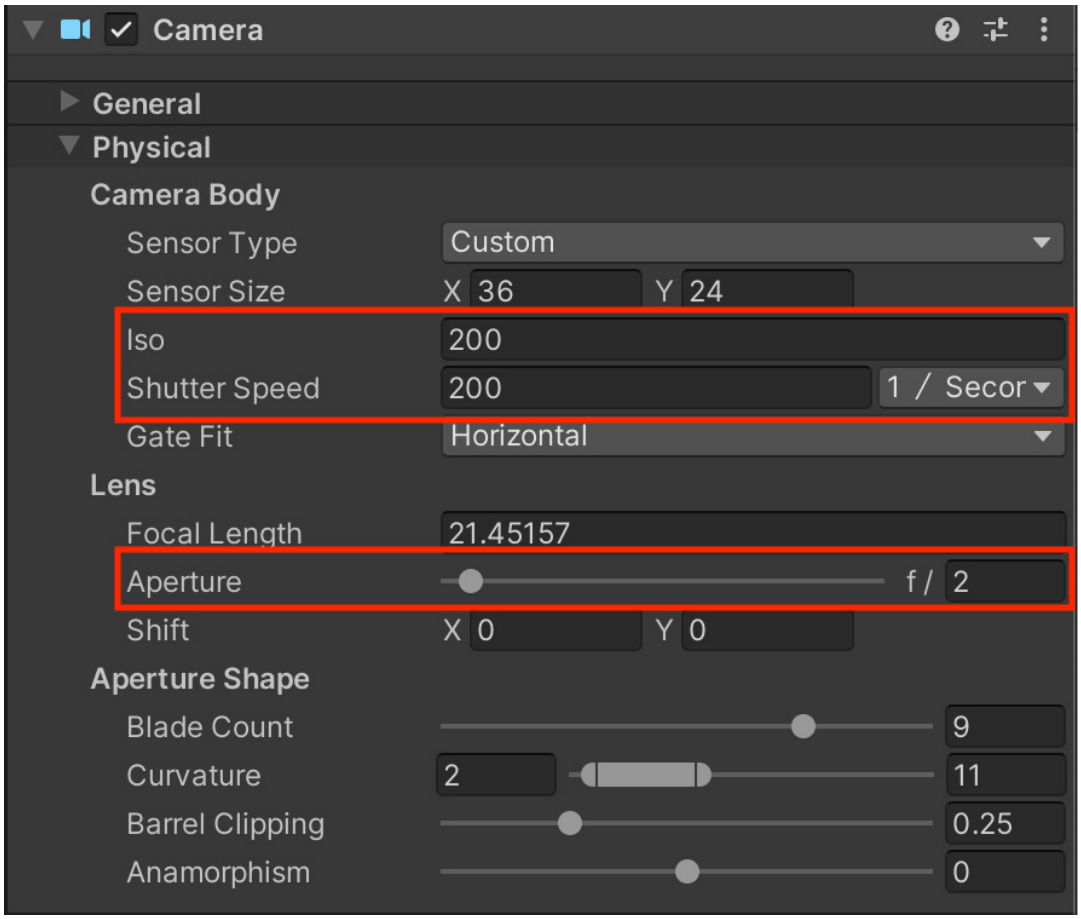
Those familiar with photography may find Physical Camera mode helpful for setting camera parameters.

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

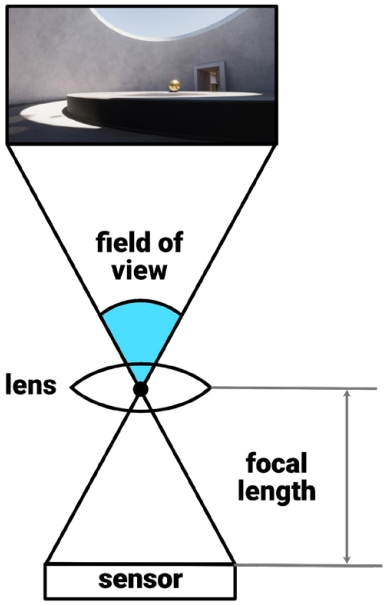
Important to exposure are the **ISO** (sensitivity), **Aperture** (or f-number), and **Shutter Speed**. If you are matching reference photos, copy the correct settings from the image's Exif data. Otherwise, this table can help you guesstimate Exposure Value based on f-number and shutter speed.

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.



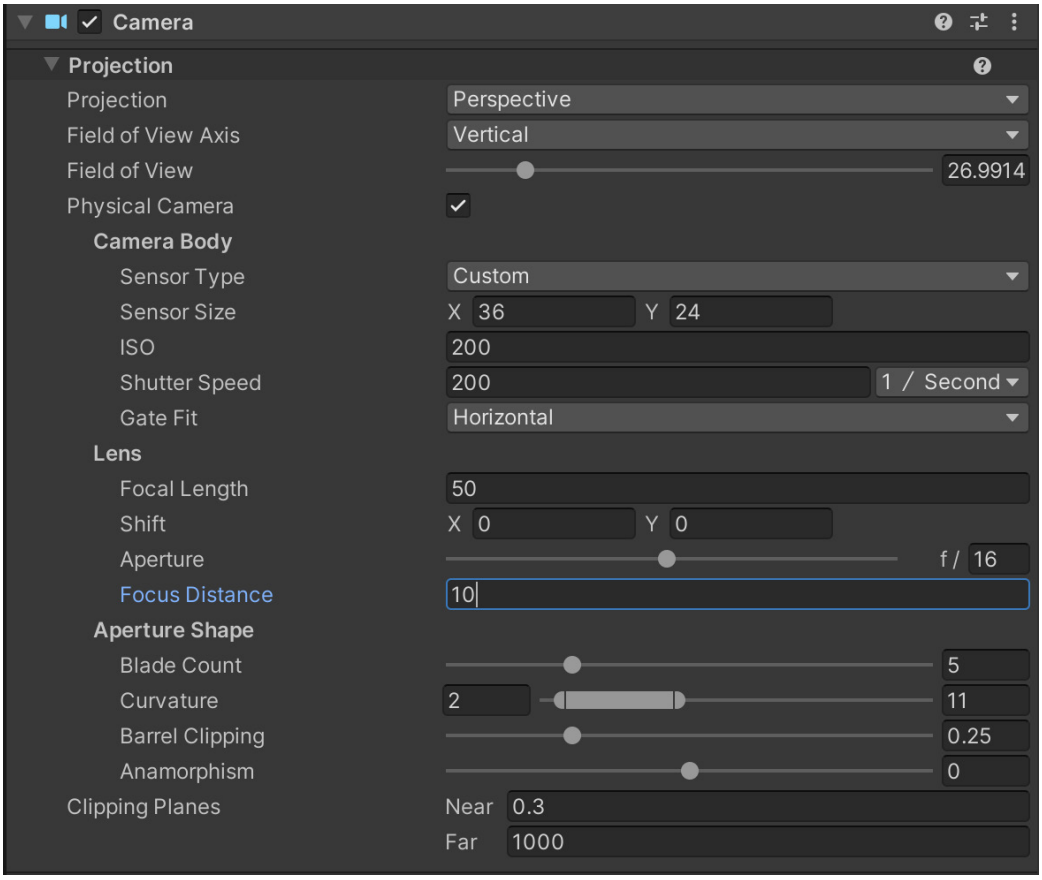
The relationship between focal length, sensor size, and field of view

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.

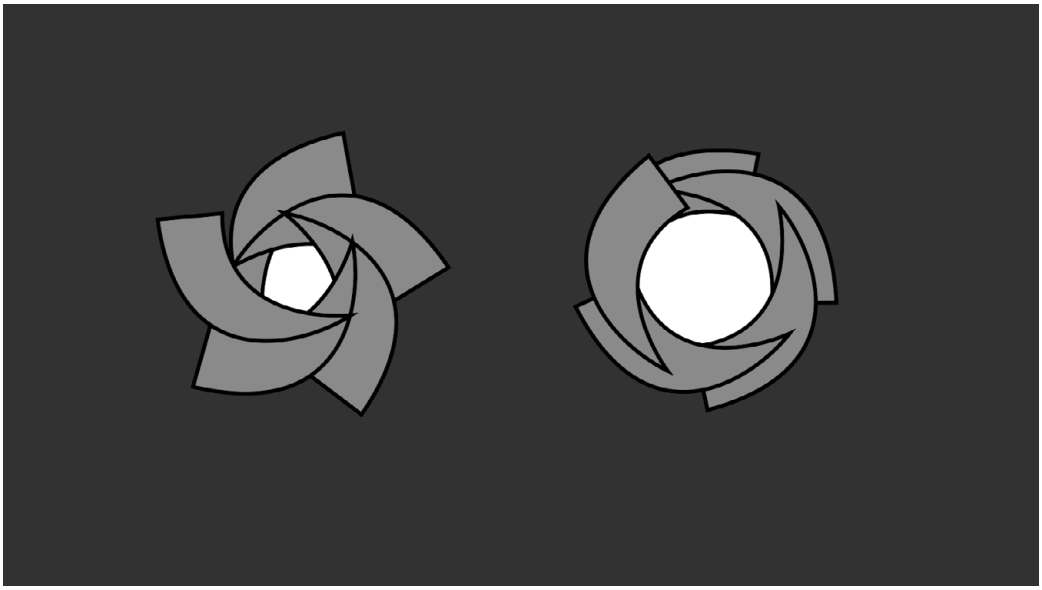
Note: You may need to search for the exact sensor dimensions on the manufacturer's website once you have the camera make and model from the metadata. This article includes an estimate of common image sensor formats.

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

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

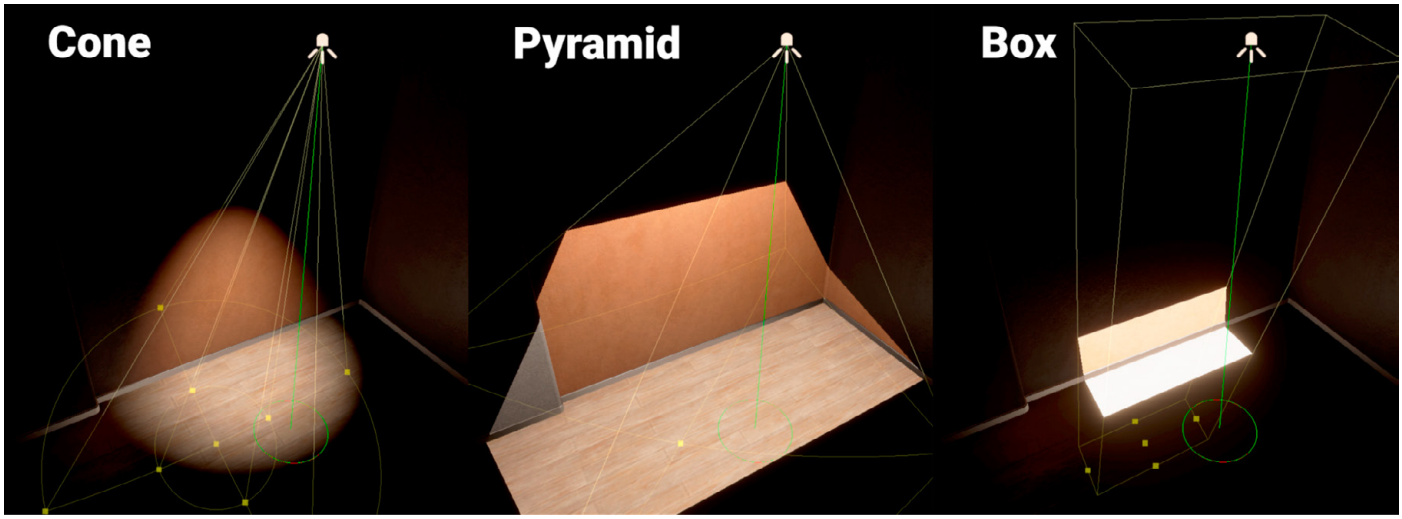
Spot and area lights have additional shapes for controlling how each light falls off.

HDRP spot lights can use three shapes:

— A **Cone** shape 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.

— A **Pyramid** shape projects light from a single point onto a square base. Adjust the pyramid shape with the **Spot Angle** and **Aspect Ratio**.

— A **Box** shape projects light uniformly across a rectangular volume. An X and Y size determine the base rectangle, and the Range controls the Y dimension. This light has no attenuation unless **Range Attenuation** is checked and can be used to simulate sunlight within the boundary of the box.



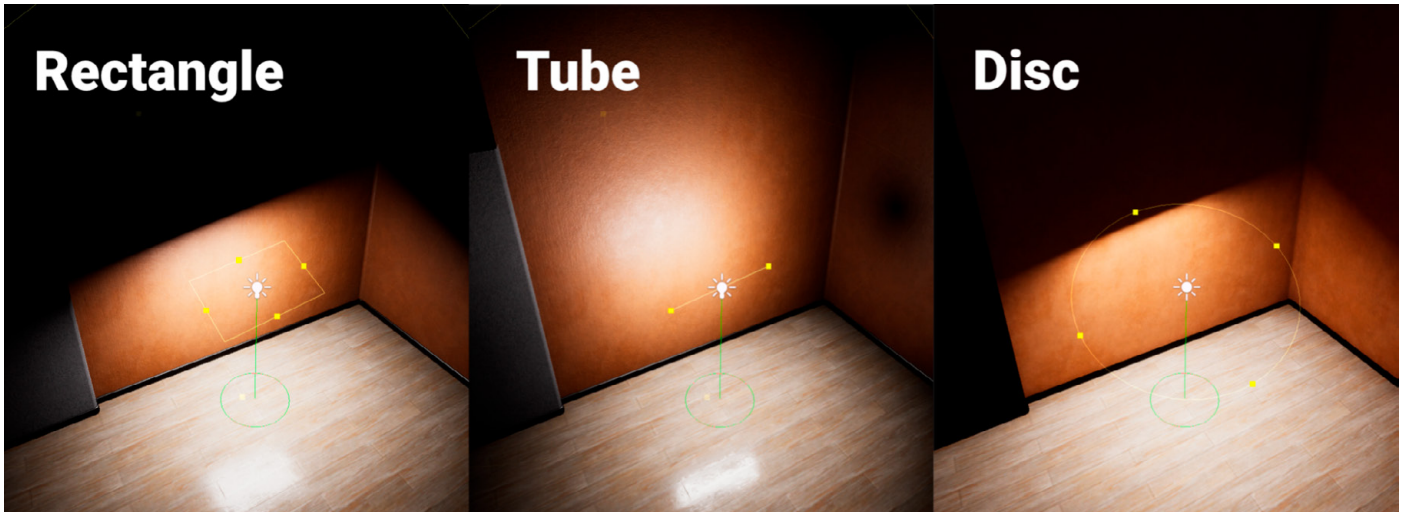
HDRP spot light shapes

HDRP area lights can use three shapes:

— A **Rectangle** shape projects light from a rectangle shape in the local, positive Z direction out to a defined Range.

— A **Tube** shape projects light from a single line in every direction, out to a defined Range. This light only works in Realtime Mode.

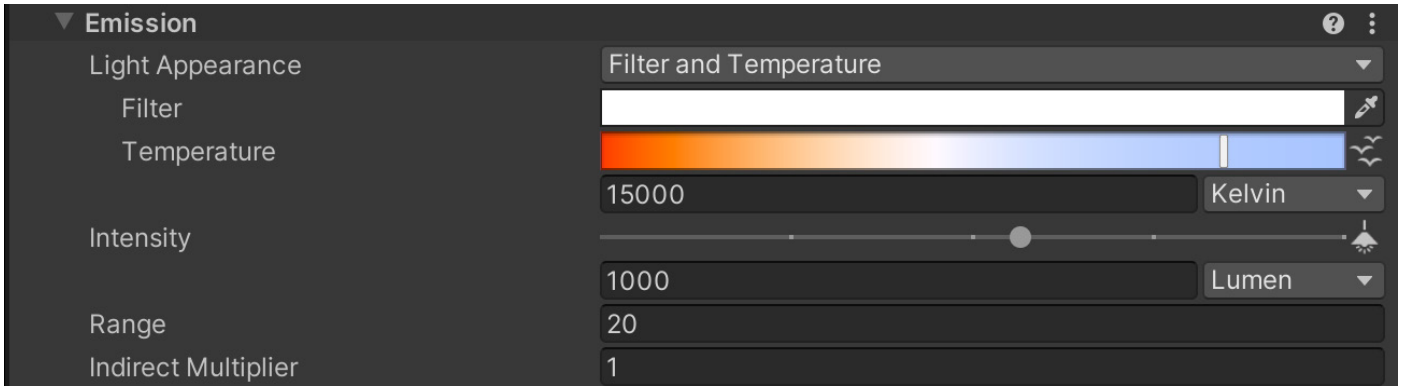
— A **Disc** shape projects light from a disc shape in the local positive Z direction, out to a defined Range. This light only works in Baked Mode.



HDRP area light shapes

Color and temperature

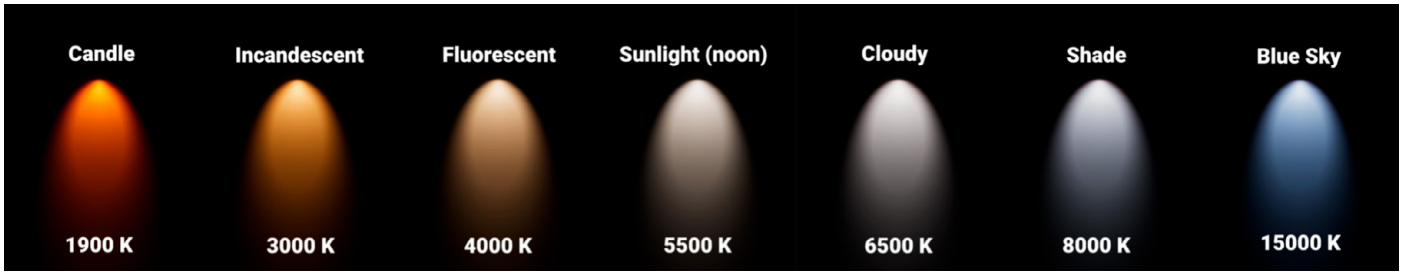
All HDRP Light types have **Emission** properties that define the light's appearance.



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

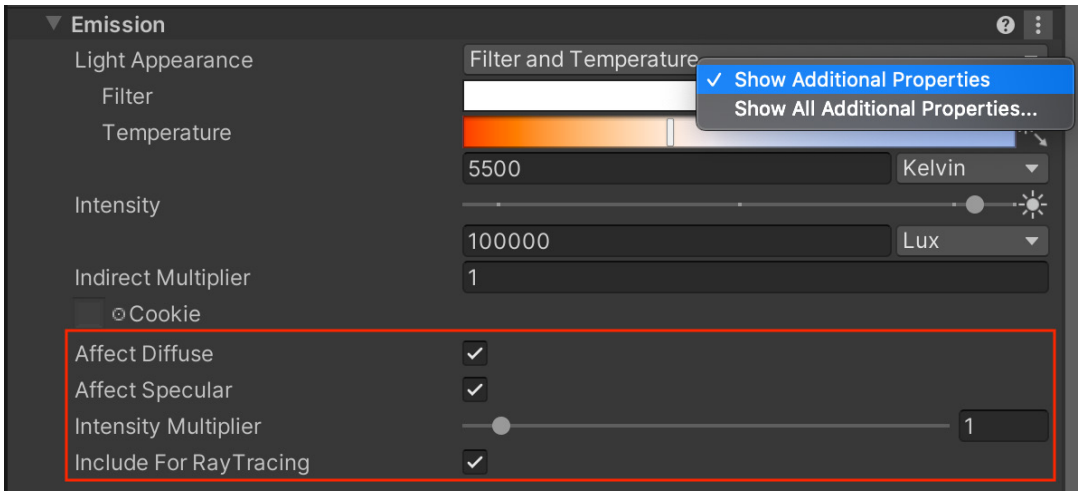
You can also add another color that acts like a **Filter**, tinting the light with another hue. This is similar to adding a color gel in photography.

Additional properties

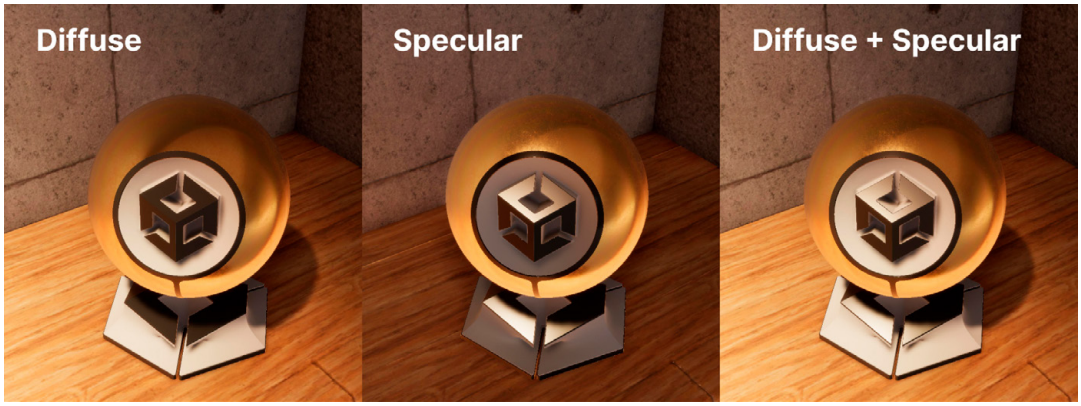
HDRP also includes some advanced controls under the **More Items menu (⋮)** at the top right of the Inspector properties. Select **Show Additional Properties** to see extra options.

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.

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.



Each Light has additional properties.

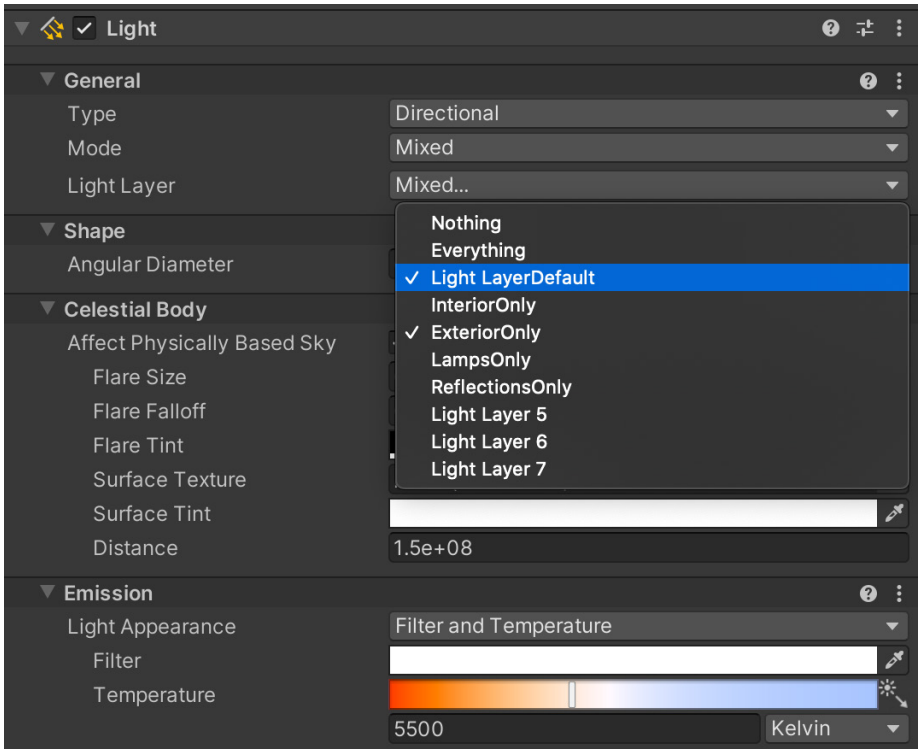


Toggle diffuse and specular lighting for additional control.

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.

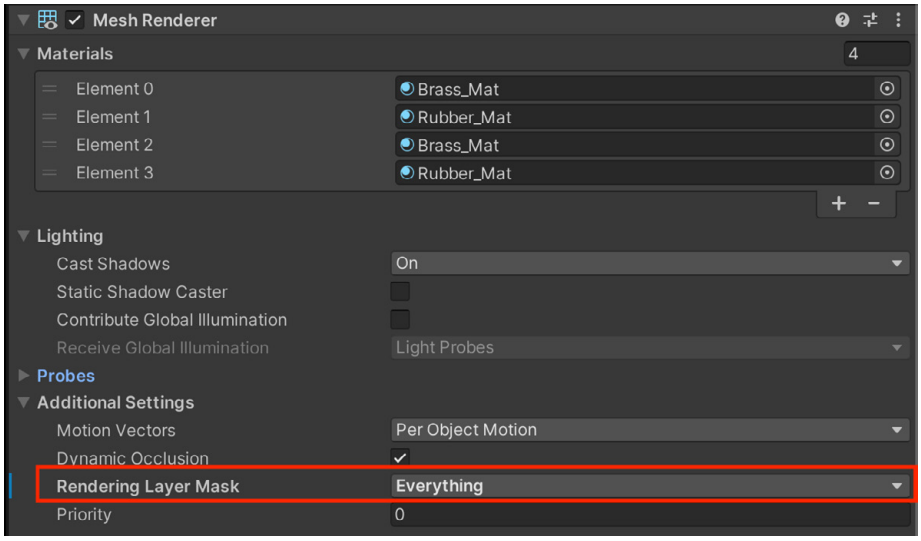
In the Light properties, click the **More Options** button (⋮). This displays the **Light Layer** dropdown under **General**. Choose which Layer Masks you want to associate with the Light.



Select a Light Layer.

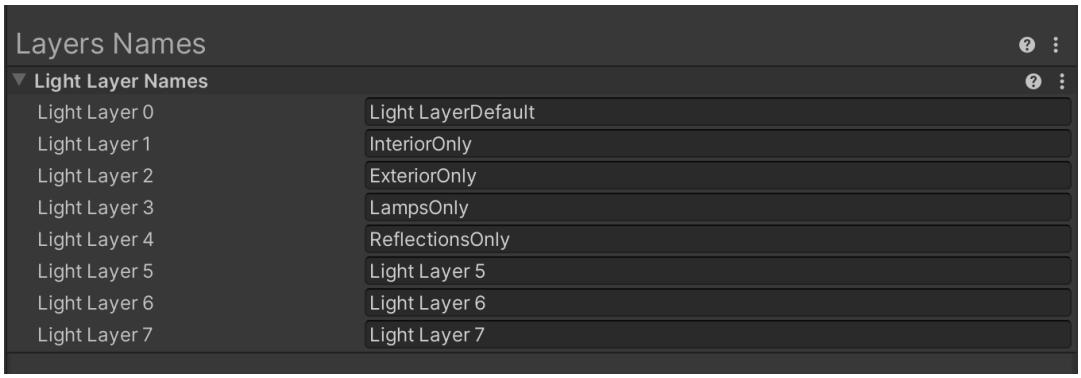
Next, set up the MeshRenderers with the **Rendering Layer Mask**. Only Lights on the matching LayerMask will affect the mesh. This feature can be invaluable for fixing light leaks, making sure that lights only strike their intended targets. It can also be part of the workflow to set up cutscene lighting, 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 6 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.

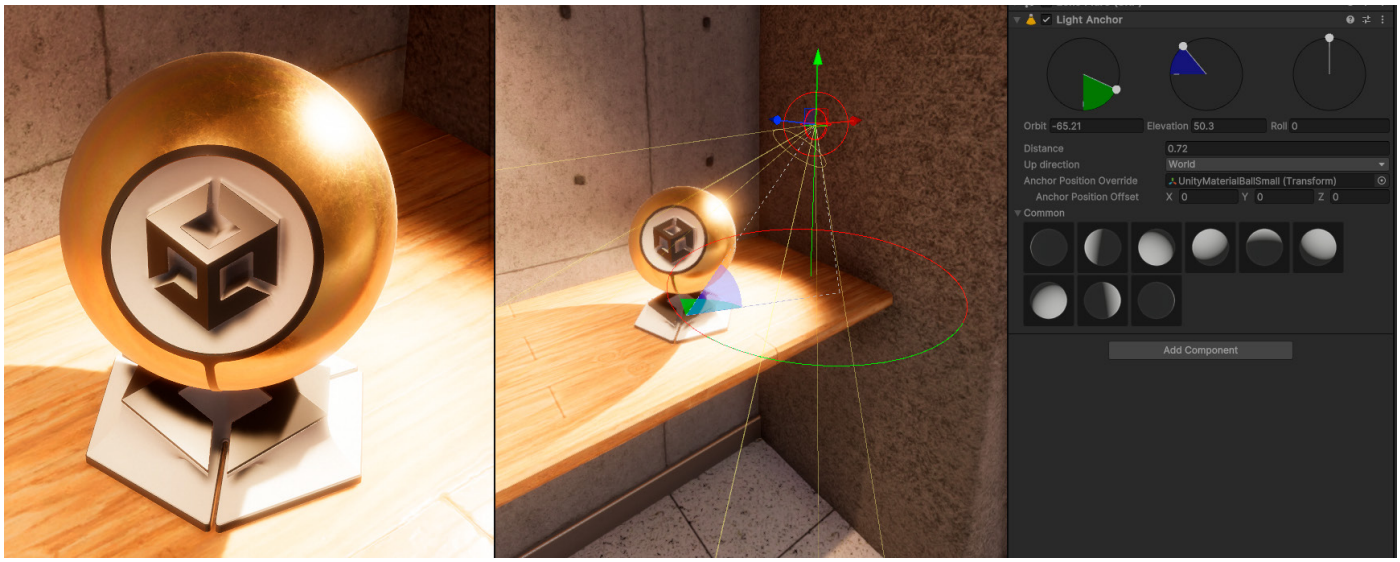
If you need to light a scene, a product, or a shot in a cinematic using multiple lights around characters or props, the Light Anchor component offers fast light manipulation in screen space around an anchor target.

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.

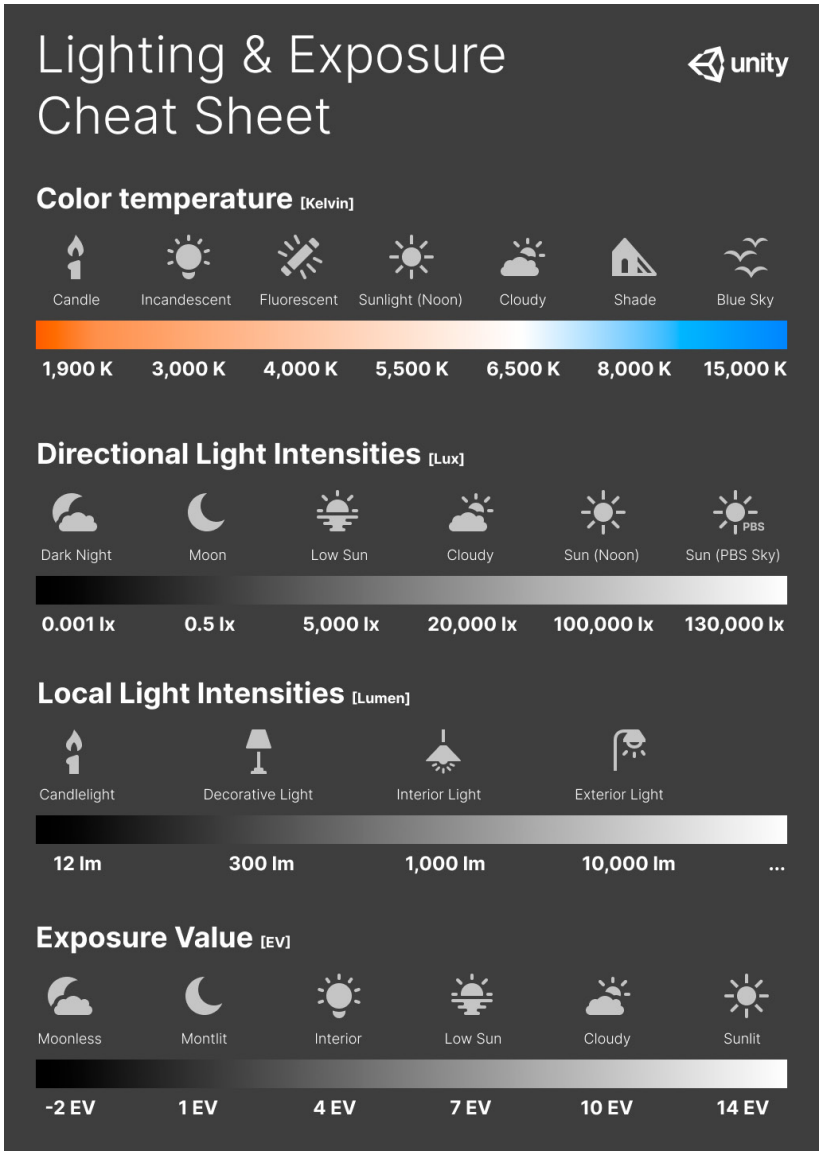
— **EV100:** This uses an intensity corresponding to EV100, which is an exposure value with 100 ISO film (see exposure value formula above). Incrementing the exposure results in the doubling of the lighting, due to the logarithmic behavior.

— 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.

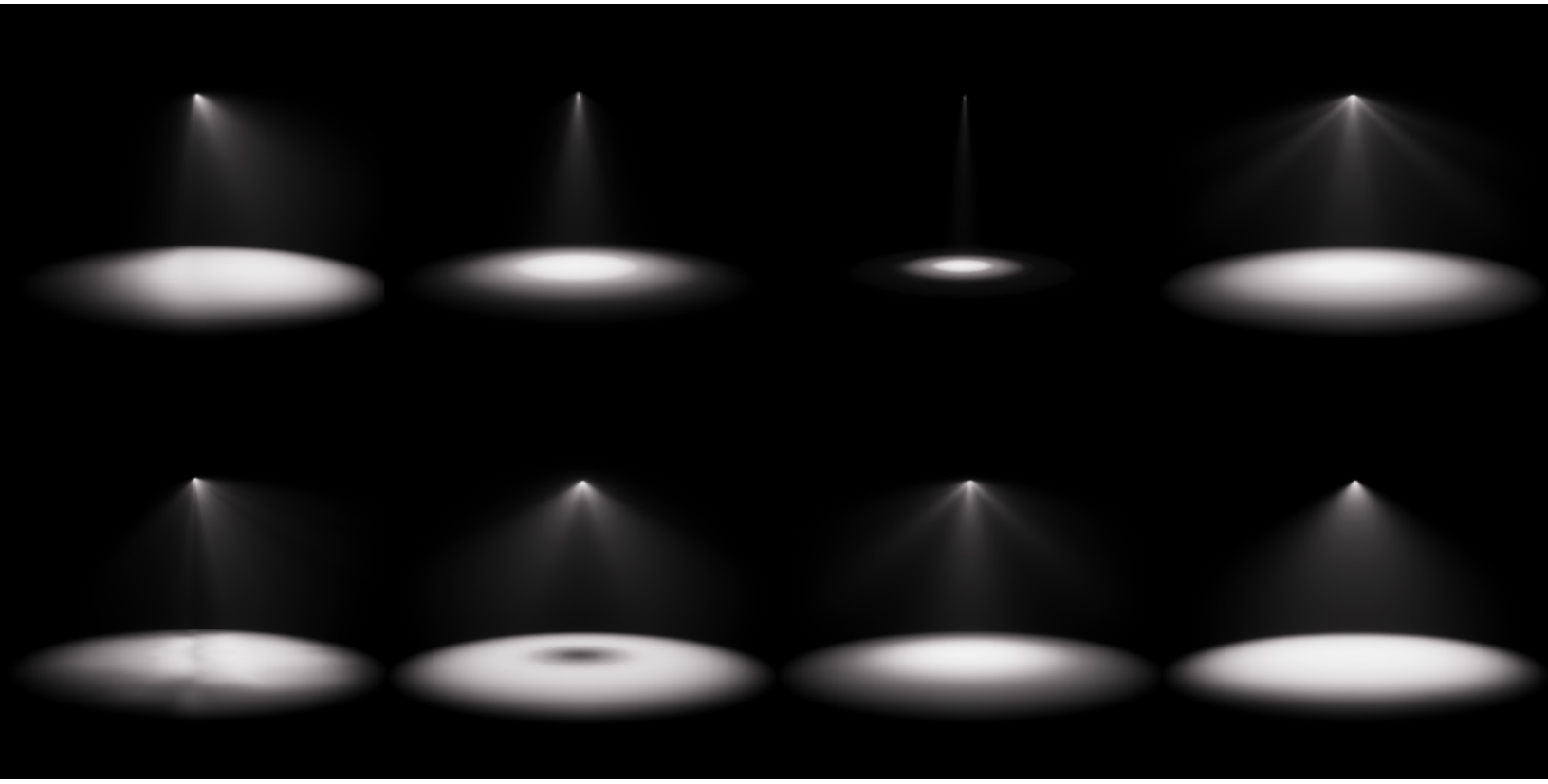


Guidance for lighting and exposure levels

You can find a complete table of common illumination values in the Physical Light Units documentation.

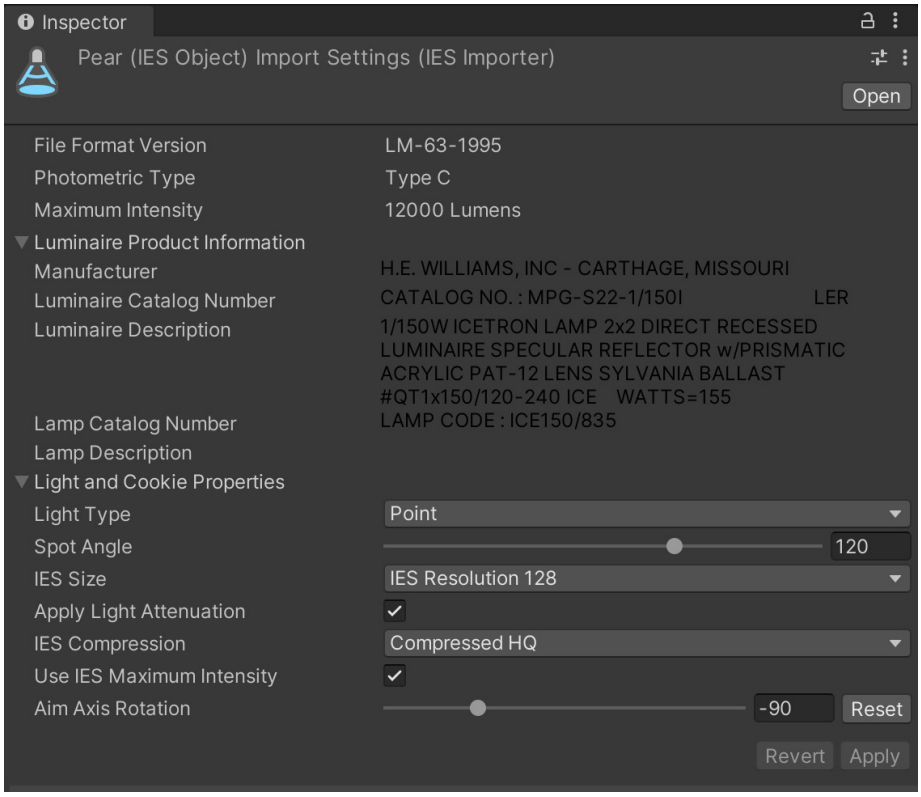
IES Profiles and Cookies

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.



IES Profile and Import Settings

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.

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.

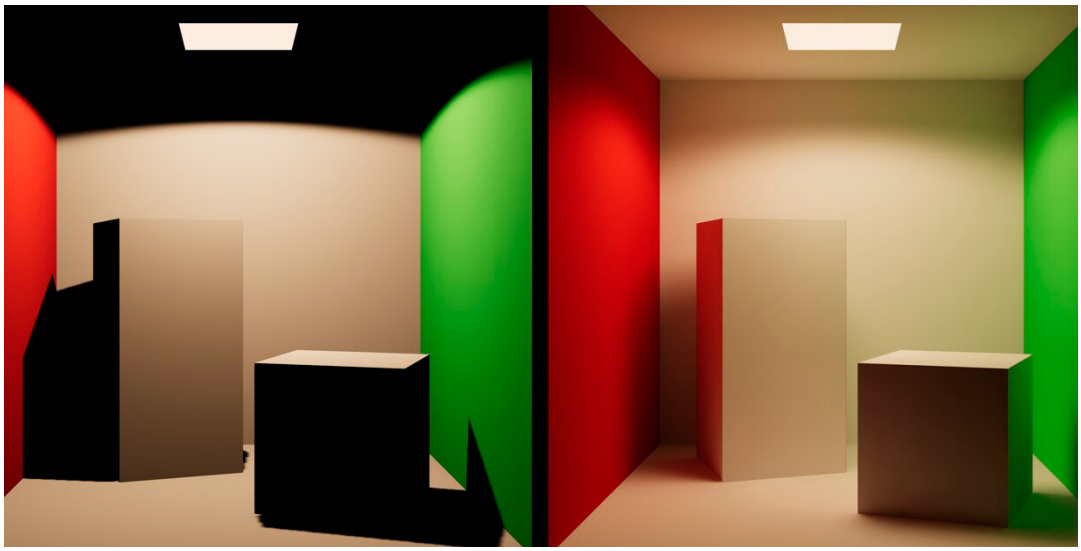
That's what global illumination (GI) tries to imitate. Without GI, areas outside of direct light are dark by default. GI attempts to approximate diffuse reflections; like in the real world, colored light transfers from one surface to another. This bounced, indirect lighting helps ground your game world 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 global illumination (baked lightmaps):** Precomputes indirect light for static objects, storing the results for runtime use
- **Real-time global illumination:** Computes indirect light in real-time, adapting to scene changes
  - Screen Space Global Illumination (SSGI): For real-time light propagation
  - Ray-Traced Global Illumination (RTGI): For physically accurate light bounces



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

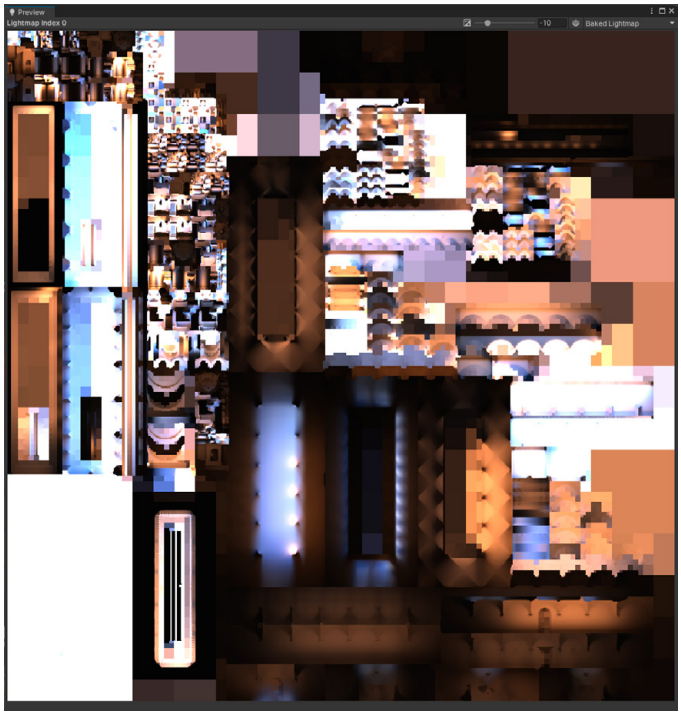
- **Light probes and Adaptive Probe Volumes (APVs):** Stores precomputed lighting information, allowing dynamic objects to receive indirect lighting without real-time calculations
  - Light Probes: Capture and interpolate baked lighting at manually placed points
  - APVs: Dynamically adjust probe density and support occlusion for more accurate large-scale lighting
- **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.

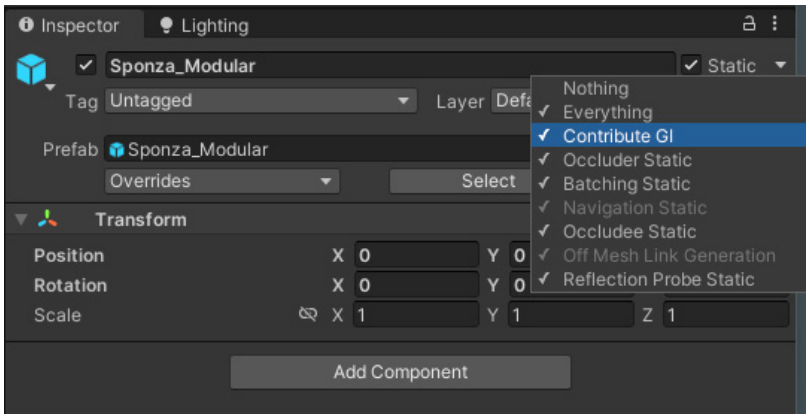


A baked lightmap

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

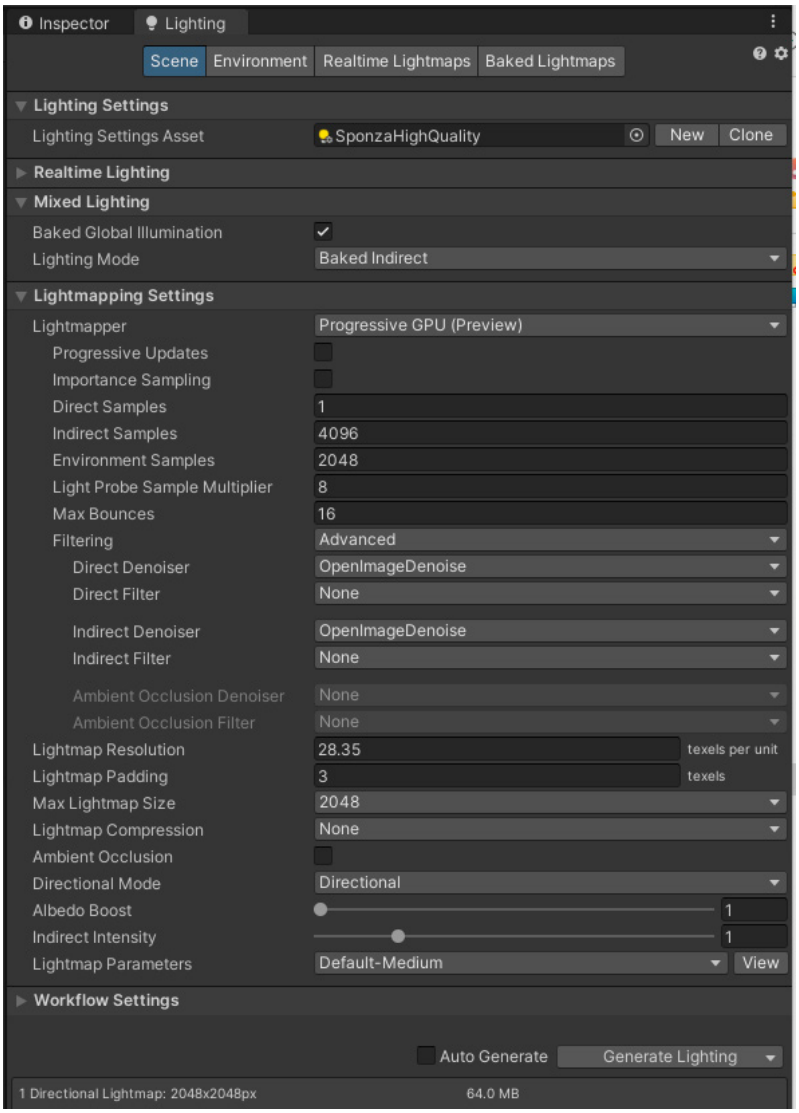
When lightmapping a scene with baked GI, follow these steps:

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).

If available, the GPU Lightmapper offers an order of magnitude faster lighting data generation compared to CPU Lightmapper. The GPU Lightmapper in Unity 6 now is production-ready and introduces a few key improvements:

— **LightBaker v1.0**: This new backend captures an instant snapshot of the Scene state when baking, making the process more predictable and stable. If you change the scene during baking, the bake process no longer restarts and the Editor remains more responsive.

— **GPU Baking Profile**: Use a GPU Baking Profile to balance performance with GPU memory usage.

— **Smaller GPU memory requirements:** In Unity 6, the new baking backend reduces the minimum GPU memory requirement to 2 GB VRAM.

The LightBaker backend improves the handling of lightmaps, light probes, shadowmasks, and ambient occlusion textures. This new architecture is internally more modular, making it easier to maintain and debug. These improvements are designed to speed up lighting iteration time while maintaining high-quality results.

Note that **Auto** mode still uses the previous progressive baking feature.

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

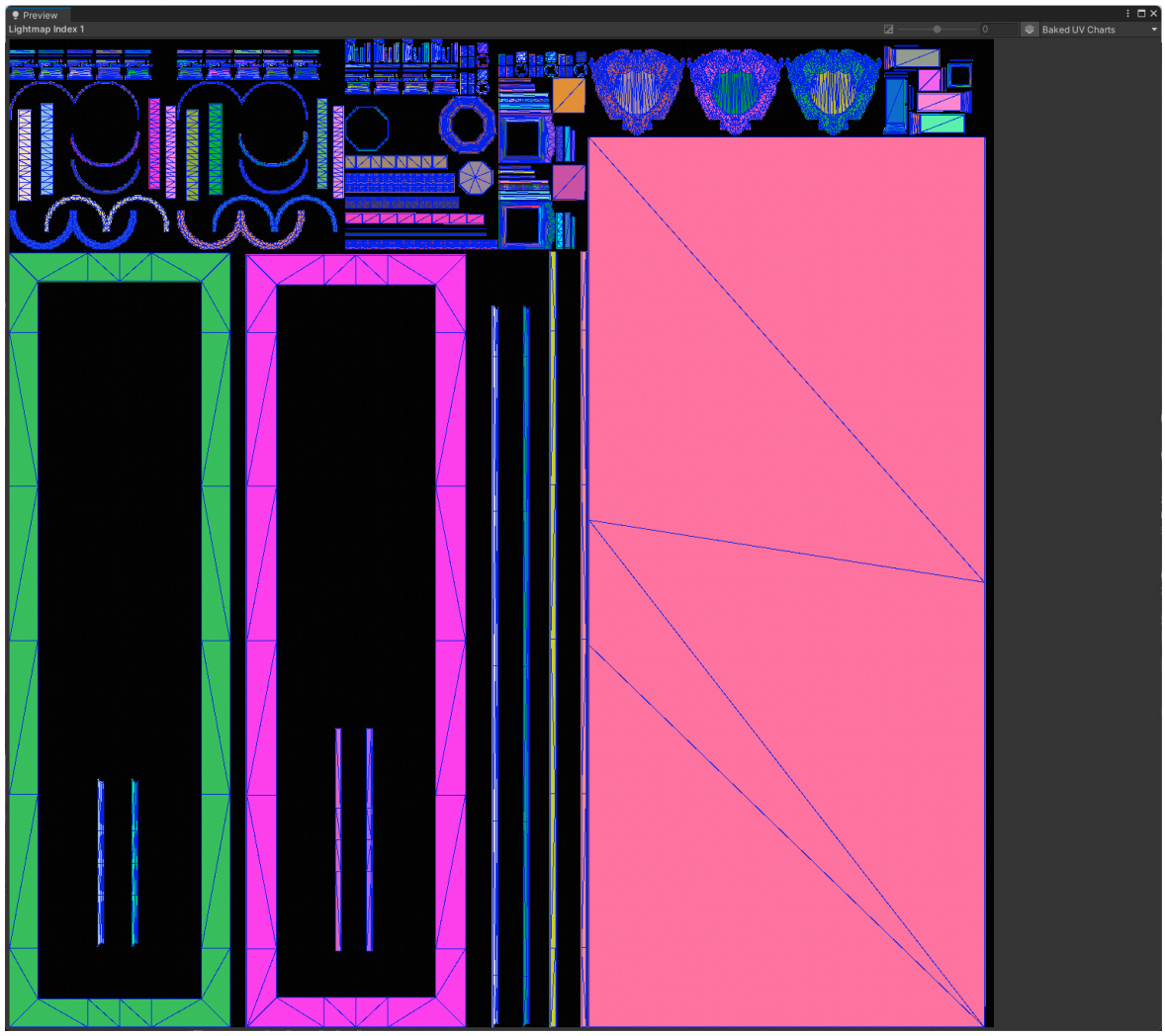
Lightmaps are textures, so Unity needs UVs to correctly use them in your scene.

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.”

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.



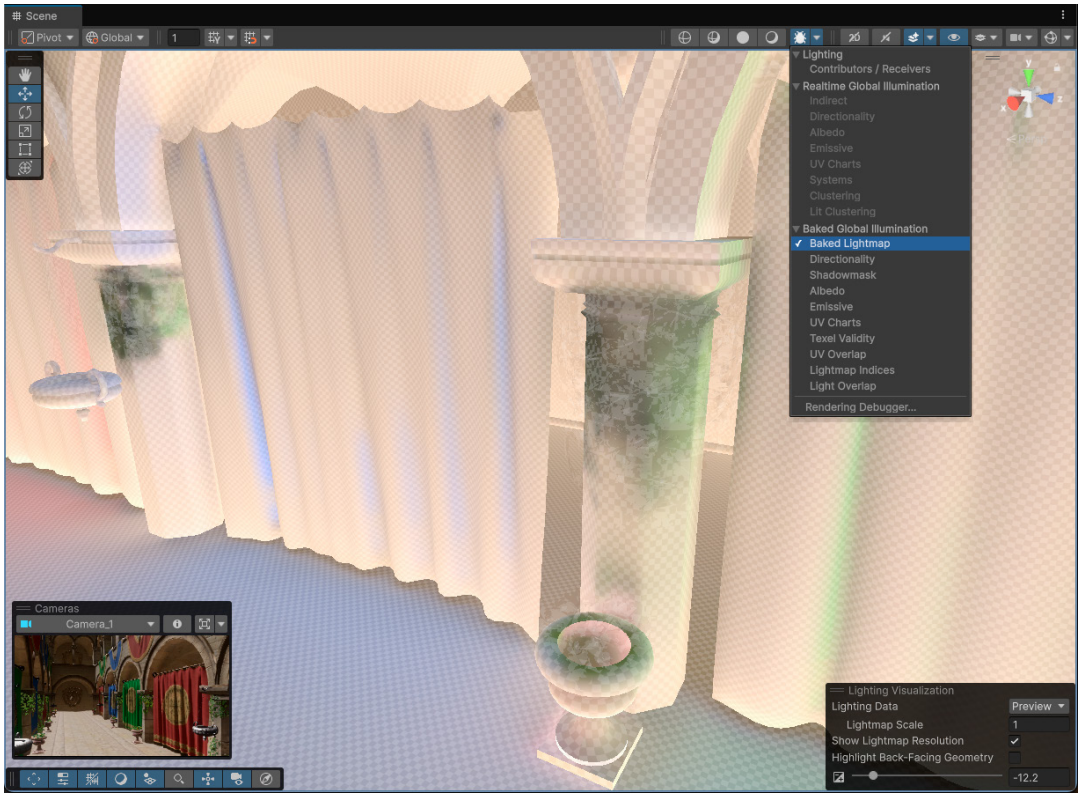
Lightmaps need UVs to display correctly.

For more technical information about Lightmap UVs, refer to the documentation page.

Interactive Preview for baking lighting (Unity 6.1)

Unity 6.1 introduces interactive **GI Debug Preview Mode** in the Scene view. This new feature provides a real-time preview of baked lighting, showing how changes to the scene affect global illumination without committing to a full rebake.

Integrated directly into Scene view draw modes, it displays updates interactively without changing existing baked data. To use it, select a debug draw mode under the **Baked Global Illumination** in the dropdown.



The GI Debug Preview Mode is available in the Scene view.

This feature replaces the previous **Auto Generate** system for baked global illumination (GI).

Dynamic global illumination

While baked GI provides high-quality indirect lighting, it requires setup time and can be costly to compute. It also comes with one major limitation – it's static. Baked lightmaps cannot respond to changing light conditions, moving objects, or time-of-day transitions. To overcome this, Unity 6 HDRP offers several dynamic global illumination (GI) techniques that provide real-time or near-real-time lighting updates.

Screen Space Global Illumination (SSGI) estimates diffuse light bounces using the depth and color buffer of the screen. This allows for real-time indirect lighting without precomputed lightmaps.

Ray-traced global illumination (RTGI) extends SSGI by using hardware-accelerated ray tracing (DXR) for more physically accurate light bounces. When ray tracing is enabled, the available properties in the Inspector change to reflect RTGI settings.

Light probes and Adaptive Probe Volumes (APVs) use probe-driven indirect lighting to approximate global illumination. APVs adaptively place probes to provide real-time or near-real-time indirect lighting updates, making them suitable for large-scale environments with dynamic lighting changes. For more information about light probes and APVs, see the section below.

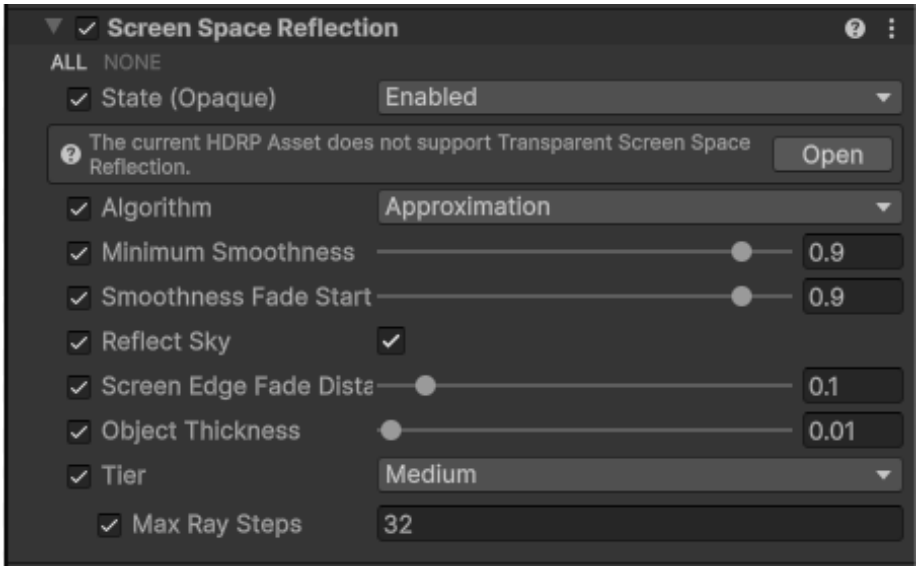
Both SSGI and RTGI replace all lightmap and light probe data. If enabled in a Volume affecting the Camera, light probes and the ambient probe will no longer contribute to indirect lighting for GameObjects.

Choose the technique that best suits your project's performance requirements and dynamic lighting needs. These techniques can complement or replace baked lighting, depending on your project's performance requirements and dynamic lighting needs.

Screen-Space Global Illumination (SSGI)

SSGI approximates indirect lighting by using the depth and color buffer to estimate how light bounces off surfaces. It can be helpful if you have fast-changing lights and cannot precompute lightmaps.

Note that SSGI only works with on-screen data; off-screen objects don't contribute to the global illumination.



Add Screen Space Global Illumination as a Volume override.

Add the **Screen-Space Global Illumination** override in the Volume system (**Lighting > Screen Space Global Illumination**). Enable this feature in **Frame Settings** and the HDRP Asset's **Lighting** section.

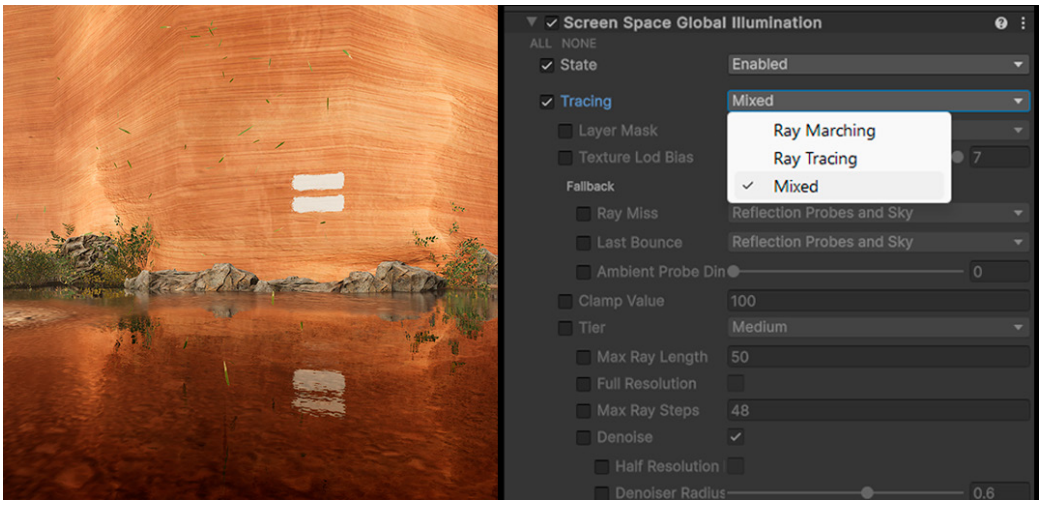
Screen Space Global Illumination casts realistic indirect light.

Tracing modes

Both SSGI and Screen Space Reflection use Tracing modes, which determine what properties appear in the Inspector:

- **Ray Marching:** This mode uses a screen-space, ray marching solution. Ray marching determines lighting for screen-space objects visible in the GBuffer.
- **Ray Tracing:** This mode uses ray tracing. Ray tracing calculates lighting for objects outside the screen view or beyond the ray marching limit. See Ray Tracing and Path Tracing (below) for more information.
- **Mixed tracing mode:** This combines ray tracing and ray marching. This mode captures additional effects like particles, vertex animations, and decals, which wouldn't be possible with ray tracing alone. Mixed Tracing mode requires **Performance** mode and the **Lit Shader Mode** to be set to **Deferred**. This only affects objects in the GBuffer rendered with deferred rendering.

  In Unity 6, Mixed Tracing mode for transparent Screen Space Reflections now mixes both tracing modes as expected, instead of only using ray traced reflections. For example, in this Scene, it can include reflections for the leaf particles, the white decal, and GameObjects that aren't visible in the cliff face's non-deformed geometry.



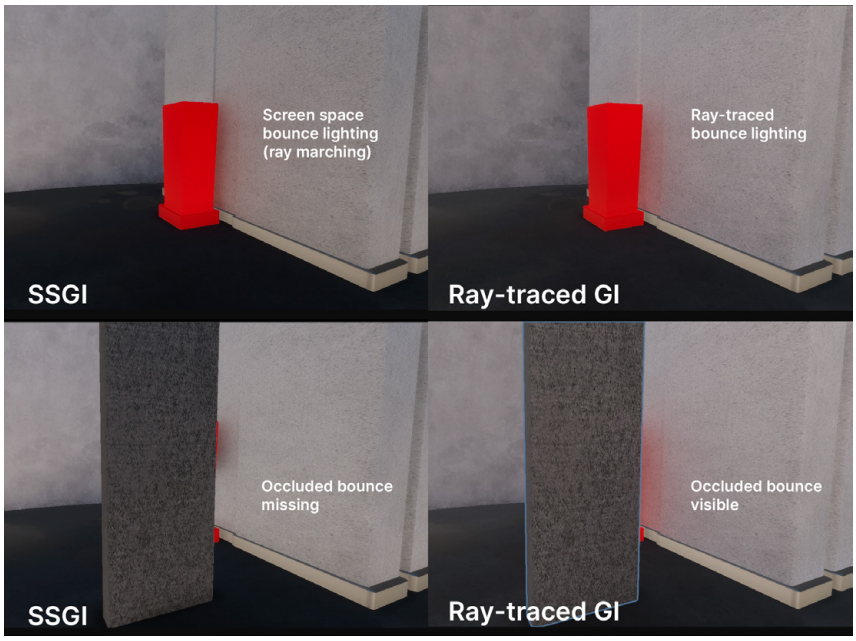
Mixed tracing mode

Ray-traced Global Illumination (RTGI)

RTGI uses hardware-accelerated ray tracing to calculate indirect light. This provides higher accuracy than SSGI and can work for off screen objects but is computationally more expensive.

Ray tracing works best for high-end real-time rendering on platforms with RTX-cable GPUs.

Enable **Ray-traced global illumination** in HDRP settings, along with DXR support in the HDRP Asset.



Ray-traced global illumination works for off-screen objects.

See the Ray tracing chapter for more details.

Enlighten Realtime GI deprecation

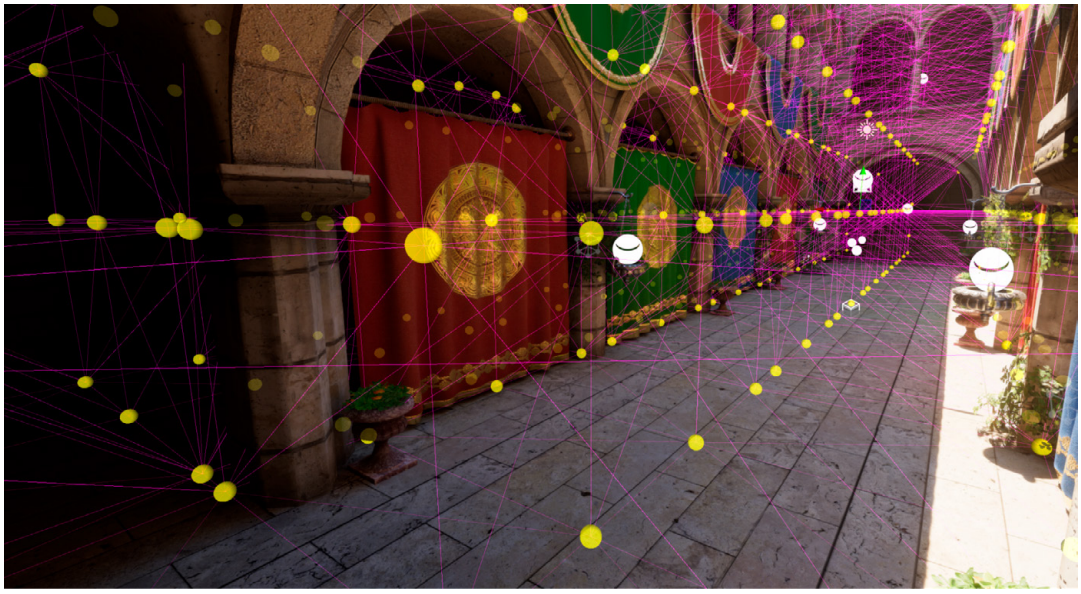
Note that Unity 6 is the last supported release for **Enlighten Realtime GI** (available under the **Lighting > Realtime Global Illumination** settings). You can find more details in this previous forum post, Update on Global Illumination 2021.

Light probes and Adaptive Probe Volumes

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 probes sample the lighting in the empty spaces.

Light probes versus SSGI/RTGI

If you enable Screen Space Global Illumination or Ray-traced global illumination on a Volume override that affects the Camera, Light Probes and the ambient probe stop contributing to lighting for GameObjects within that Volume (see Dynamic global illumination for more details).

However, Adaptive Probe Volumes (APVs) can complement SSGI and RTGI by providing indirect lighting information for areas not covered by screen space effects.

Light Probe Group

You can examine a basic light probe setup using the Light Probe Group component.

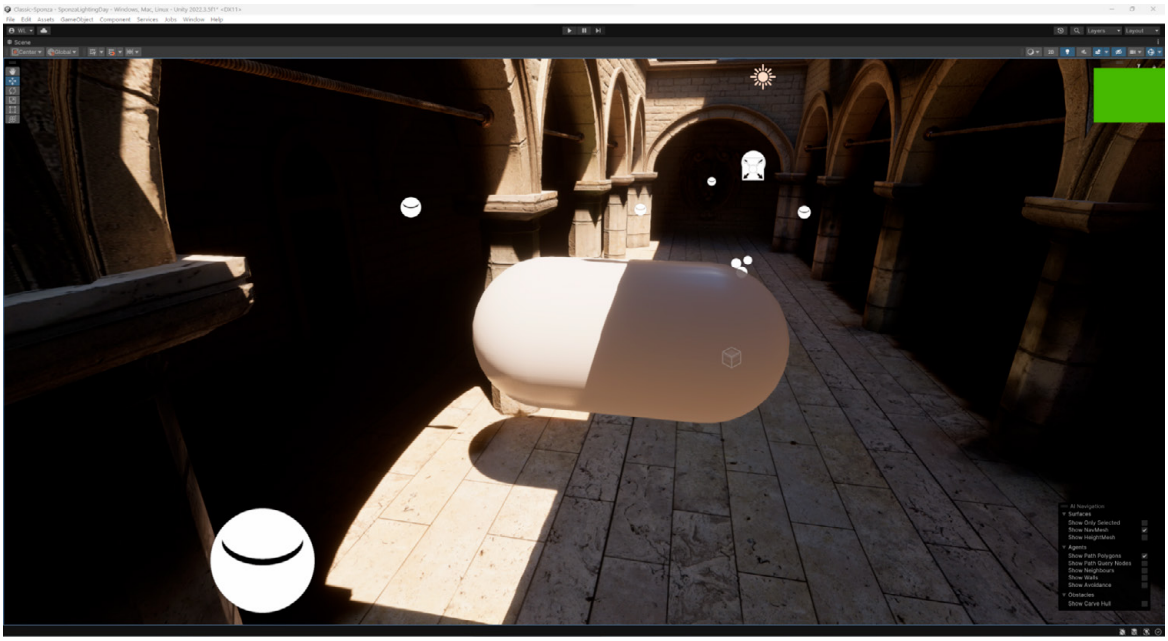
To place light probes in your scene, add a Light Probe Group component (**Component > Rendering > Light Probe Group**) to an empty GameObject. Then, use the component settings to edit the positions of the probes, add more, or delete probes.

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

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 labor intensive. Because of this, consider using Adaptive Probe Volumes (below) instead of individually arranging light probes in the scene.

Refer to the light probes documentation page for more about Light Probe Groups.

Moving light probes at runtime

Unity 6 also includes some new features to make working with light probes more streamlined for production.

The new LightProbes API allows you to reposition traditional light probes at runtime using a script. This is useful if your application uses modular or procedurally generated environments.

For example, if your game world is split into multiple scenes and loaded additively at runtime, you may need to move different parts of the environment into place. Their corresponding light probes must also be repositioned. Otherwise, their lighting wouldn't match.

However, you can't simply reposition light probes at runtime by adjusting the Transform of the Light Probe Group, as that only affects their positions for baking. Instead, the new LightProbes API lets you clone probe data, update the probe positions, and then rebuild the probe network via script. The API also includes a few performance optimizations for efficiently moving multiple probes at once.

If you're creating an environment from smaller scenes, this means you can pre-bake lighting and reposition modular pieces of your game world at runtime while maintaining lighting consistency.

For an example implementation, see the Unity Light Probes Runtime Movement Documentation.

Adaptive Probe Volumes

Adaptive Probe Volumes (APVs) in Unity 6 automate light probe placement based on scene geometry, eliminating the need for manual placement in Light Probe Groups. Unlike per-object probes, APVs use per-pixel sampling, resulting in smoother lighting transitions and greater consistency across GameObjects.

Because APVs work per pixel, they avoid many of the lighting issues caused by manually placed light probes. The comparison below illustrates this difference between Light Probe Groups (left) and APVs (right).

Light Probe Group versus an APV

In the Light Probe Group, each object samples a single interpolated probe, leading to mismatched lighting between separate parts. An APV uses per-pixel sampling, making the lighting transitions more seamless.

The table below compares the features of Light Probe Groups with APVs.

Feature	Light Probe Groups	Adaptive Probe Volumes (APV)
Probe selection	Per GameObject	Per pixel
Optimize memory use with streaming	No	Yes
Place probes automatically	No	Yes
Blend between different bakes	No	Yes
Place probes manually	Yes	No

Advantages of APVs:

- **Per-pixel lighting:** APVs ensure smoother transitions and fewer seams between objects.
- **Improved fog accuracy:** APVs produce more precise lighting variations in volumetric fog.
- **Automatic probe placement:** Baking APVs adjusts probe density based on scene complexity for finer detail in key areas.
- **Multi-scene support:** Baking Sets allow you to bake multiple scenes together.
- **Streaming mode:** This feature optimizes memory usage for large open worlds.
- **Sky occlusion updates:** APVs can dynamically adjust for lighting from the sky at runtime.
- **Debugging tools:** The Rendering Debugger visualizes probe layouts and density levels for easier troubleshooting.

Limitations of APVs:

- **Fixed probe placement:** Light probe positions cannot be manually adjusted, which may cause artifacts near walls or boundaries.
- **No conversion from Light Probe Groups:** Existing Light Probe Groups cannot be transferred to APVs.

Compare this APV volume rendering with manually placed light probes. For many scenes, you can set up light probes in a matter of seconds.

Probe volumes improve light probe rendering.

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

APVs distribute probes automatically through the scene using “bricks” of 64 probes (4 × 4 x 4). They use a volume-based system that adapts to the geometry density in your scene.

The spacing between these 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 sparse areas, probes are spaced farther apart. This concentrates resources where they are needed.

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

The spacing of Light probes in an APV can vary.

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.

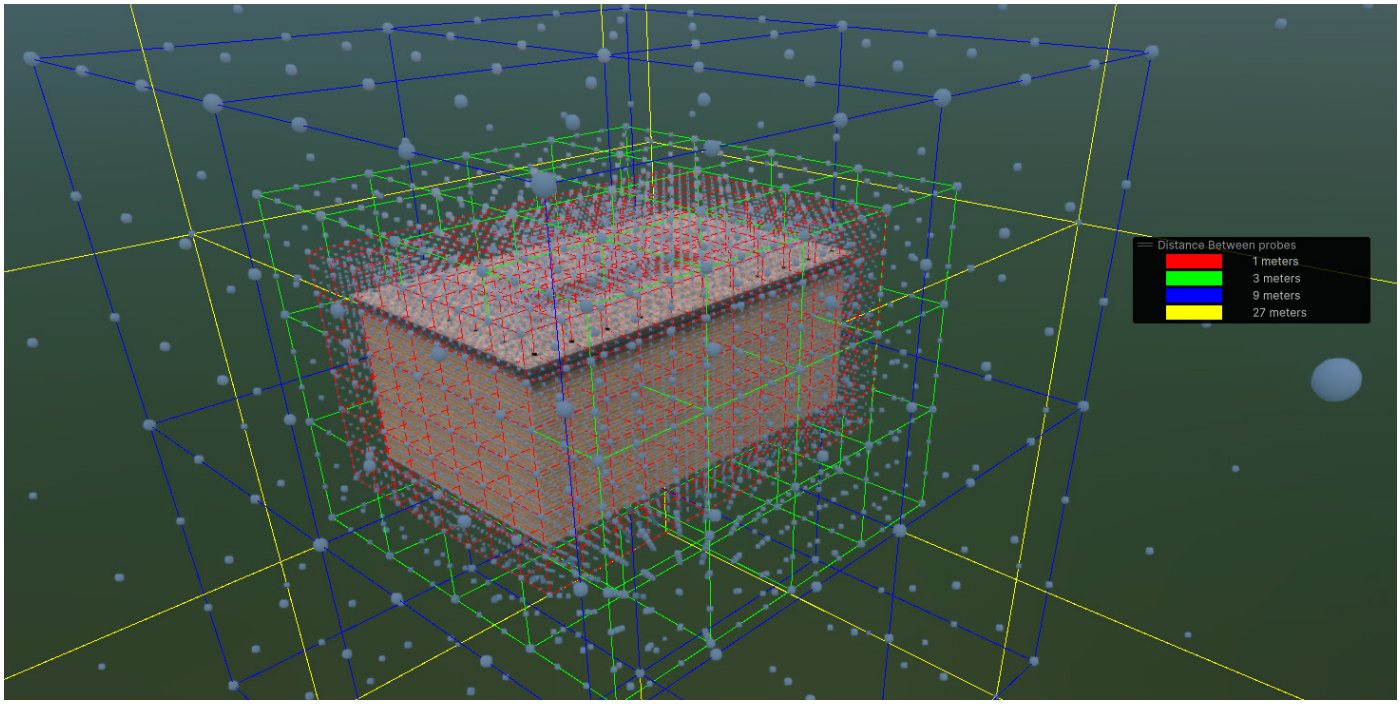
Working with Adaptive Probe Volumes

In the Hierarchy window, right-click and select **Light > Adaptive Probe Volume** (APV).

Set the Mode to **Global** and use the defaults or choose a specific **Subdivision Override**. Select a new value range in **Override Probe Spacing**.

Bake the volume by pressing **Bake Probe Volumes**. Probes are then placed in the scene based on the geometry, packing more densely where there is more geometry.

To visualize the light probes themselves, open **Analysis > Rendering Debugger**. Select **Probe Volumes** and enable **Display Probes**. To view the different resolutions choose **Display Bricks**.



Use the Rendering Debugger to visualize APVs.

Note that you can't manually adjust the individual locations of light probes inside an APV. This means that its probes may not always align perfectly with wall or light boundaries. However, you can use multiple APVs with different subdivisions to better control probe placement and density.

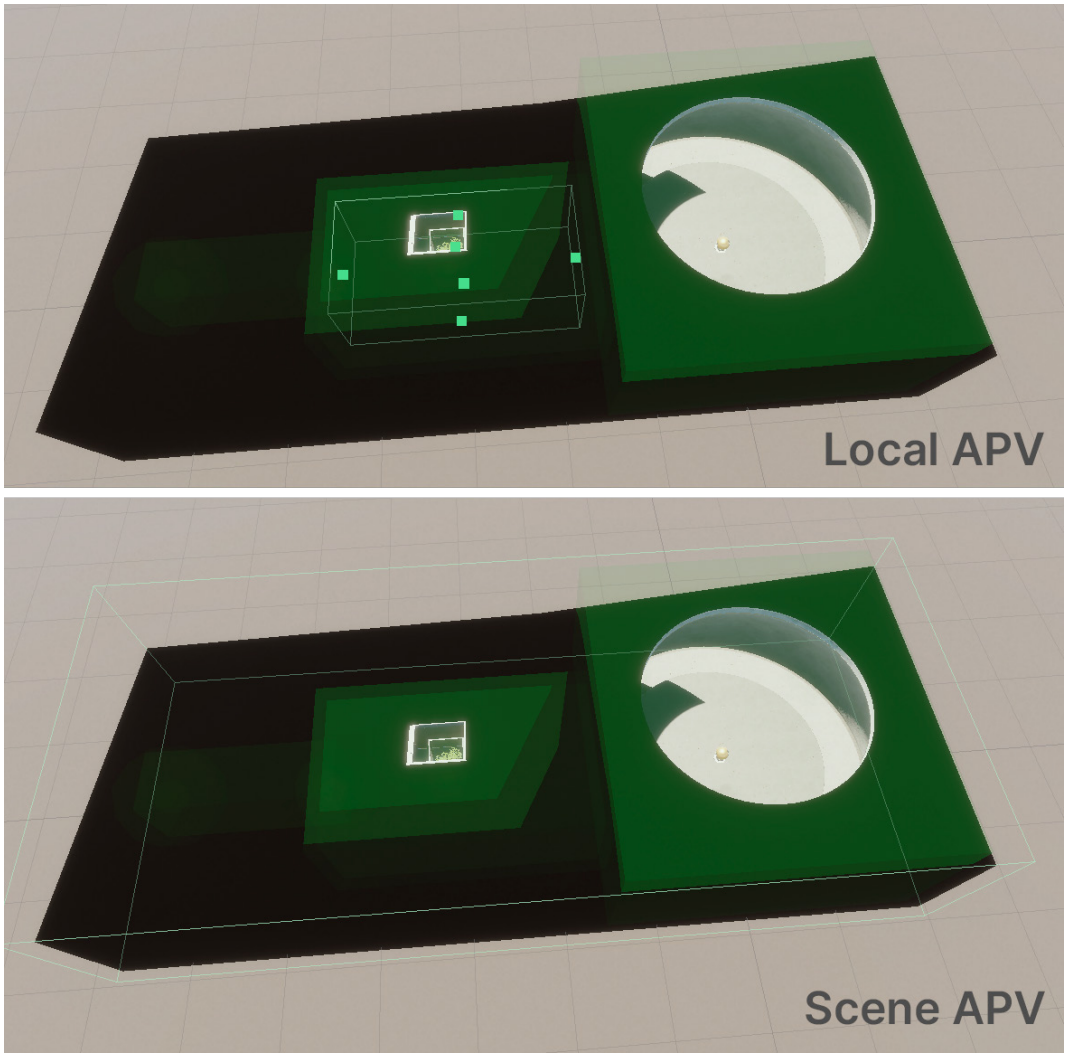
The HDRP Sample template scene uses a similar setup. It includes two Probe Volumes in the scene. An APV uses a Mode to control how HDRP determine the volume's size and which renderers influence it:

— **Global:** The APV resizes automatically to include all renderers in the Baking Set or scene marked with Contribute Global Illumination. HDRP recalculates this volume size every time you save or regenerate lighting.

— **Scene:** The APV resizes automatically to include all renderers in the current scene.

— **Local:** APV does not resize automatically. Instead, use the Size to set the volume manually.

In the Sample template, one APV that encloses the entire level has a **Mode** set to **Scene**, while another APV for the second room has its **Mode** set to **Local**.

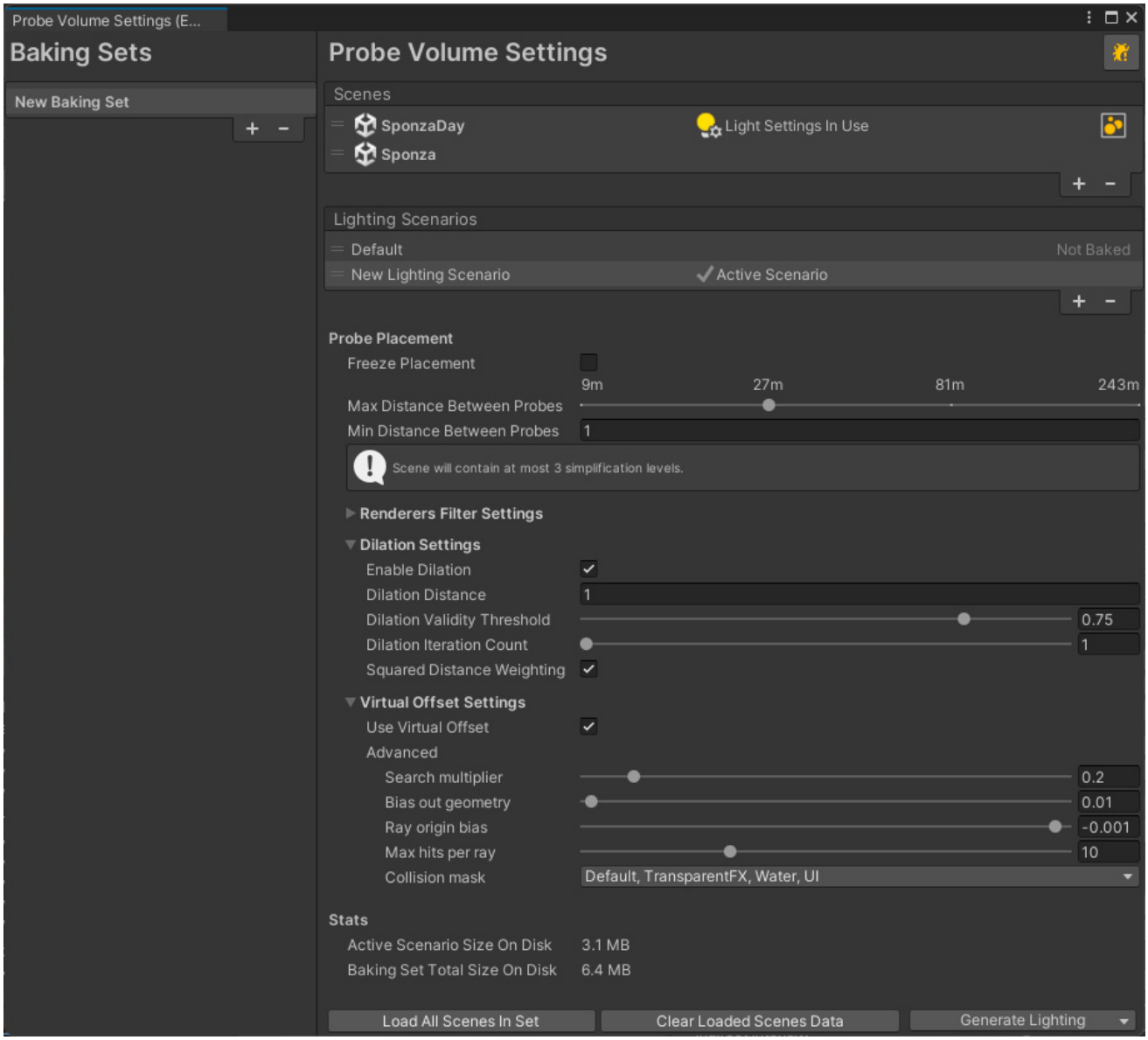


Use multiple APVs in your scene for precise control.

Bake the Probe Volumes from either:

— **The Adaptive Probe Volume component**: Select **Bake Probe Volumes** to only bake the APVs from a specific GameObject.

— **The Lighting window:** Select **Generate Lighting** to bake APVs along with reflection probes and other baked lighting.



The settings for probe volumes

In some cases, switching entirely to APVs – and removing lightmaps entirely – can simplify your workflow by using a single, consistent lighting system for all scene objects.

HDRP splits the baked data into multiple parts for efficiency. When using multiple Lighting Scenarios, probe volumes don't duplicate baked data on disk as long as probe placement and geometry remain the same between bakes.


Baked lightmaps (left) versus APVs only (right)

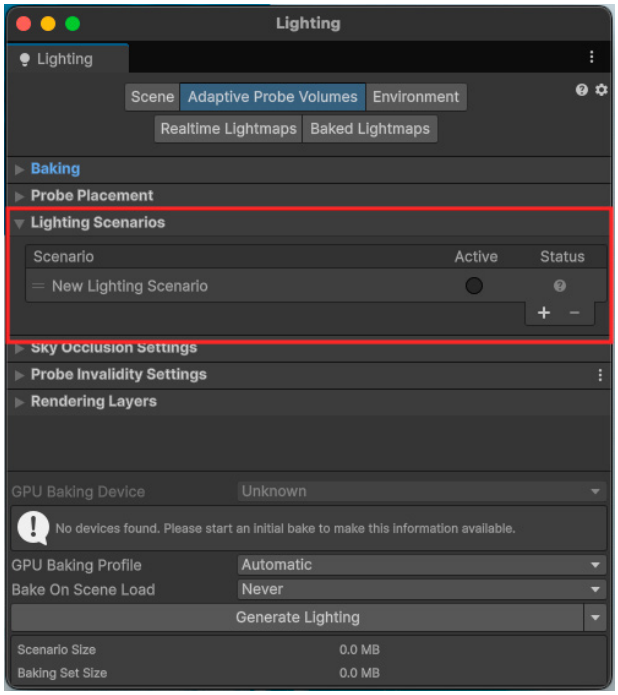
While probe volumes might not capture all of the nuances of baked lightmapping, you can combine them with real-time effects, such as Screen Space Ambient Occlusion to help add back details. This is one approach to indirect lighting without relying on lightmaps.

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 APVs to your scenes.

Lighting Scenario asset

APVs can also switch between indirect lighting data using a **Lighting Scenario** asset. Each Lighting Scenario stores a baked lighting state. You can bake different lighting setups into different Lighting Scenarios, and change which one HDRP uses at runtime. A Lighting Scenario can match either a scene or a Baking Set.

For example, you can create one Lighting Scenario for day, and another one for night. At runtime, you can switch or blend between the two using a script. In the Editor, you can preview the effect using the Rendering Debugger.



Apply Lighting Scenarios in the Lighting window.

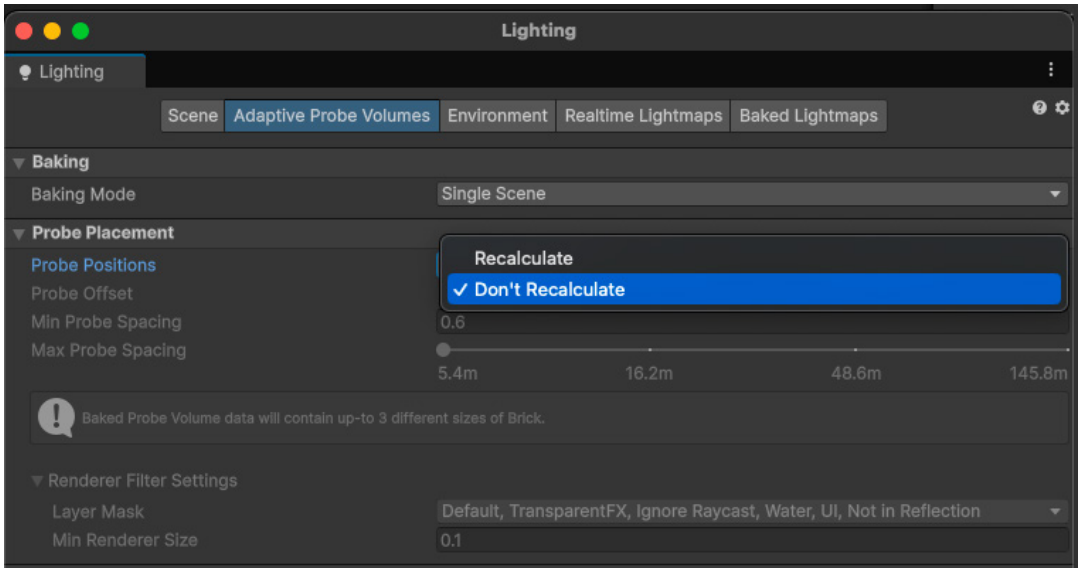
Rather than switching lighting instantly, you can use **Scenario Blending** to gradually transition between two Lighting Scenarios, making the change more natural. This avoids abrupt shifts and allows for a more immersive day-night cycle or lighting state changes.

The same environment using different Lighting Scenarios.

To use a Lighting Scenario asset, navigate to the active HDRP Asset and enable **Lighting >**

Light Probe Lighting > Lighting Scenarios. Then, you can create a new Lighting Scenario asset to store baking results inside:

1. Open the **Adaptive Probe Volumes** panel in the **Lighting** window.
2. In the **Lighting Scenarios** section, select the **Add** (+) button to add a Lighting Scenario asset.
3. In the **Lighting** window, under the **Adaptive Probe Volume** tab, make sure the **Probe Positions** are set to **Don't Recalculate**. This ensures that Unity will only rebake lighting without changing the probe positions, which could otherwise invalidate previously baked scenarios.



Don't recalculate probe positions when using Lighting Scenarios.

Select a Lighting Scenario from the list to make it active and select **Generate Lighting**. HDRP stores the baking results in the active Lighting Scenario.

You can also use the dropdown button next to **Generate Lighting** to only **Bake Probe Volumes** if you're not using lightmaps.

Set which Lighting Scenario is currently in use at runtime using the ProbeReferenceVolume API and blend them using the BlendLightingScenario API.

You can also use the Rendering Debugger (**Window > Analysis > Rendering Debugger**) to preview transitions between Lighting Scenarios. Set the Scenario Blend Target to a Lighting Scenario and use a Scenario Blending Factor to check the blending effect.

The Scenario Blending Factor controls the interpolation between two baked lighting states. This can be used for dynamic time-of-day effects, indoor-outdoor transitions, or different lighting moods within the same scene. Scenario Blending ensures a smooth and gradual transition without a sudden "pop" when changing Lighting Scenarios.

You can see an example of Scenario Blending in the Time Ghost: Environment asset. Here, a MonoBehaviour blends between different Lighting Scenarios to simulate a smooth transition between times of day.

Lighting Scenario manager

If you change the active Lighting Scenarios at runtime, HDRP changes only the indirect lighting data in the light probes. You might still need to use scripts to move geometry, modify lights or change direct lighting.

See the `ScenarioManager` script in the TimeGhost Environment sample for reference.

Fixing issues with APVs

Because the APVs automatically place the probes on a grid, placement can sometimes cause rendering errors.

Invalid probes occur when a probe is inside geometry and its sampling rays hit the unlit backfaces, preventing accurate lighting.This might create dark areas that should be light or vice versa.

In this example, invalid probes placed below the floor of the 3D Sample scene cause a dark artifact at the bottom of the capsule.

Invalid probes cause a dark artifact.

The Editor provides several tools to let a technical artist identify and fix these issues.

— **Rendering Debugger:** Navigate to **Probes Volumes > Debug Probe Sampling** to see which probes influence a specific pixel.

— **Probe Invalidity Settings:** Navigate to the **Adaptive Probe Volumes** panel in the **Lighting** window. Use these settings to offset the capture point of each probe for better sampling.

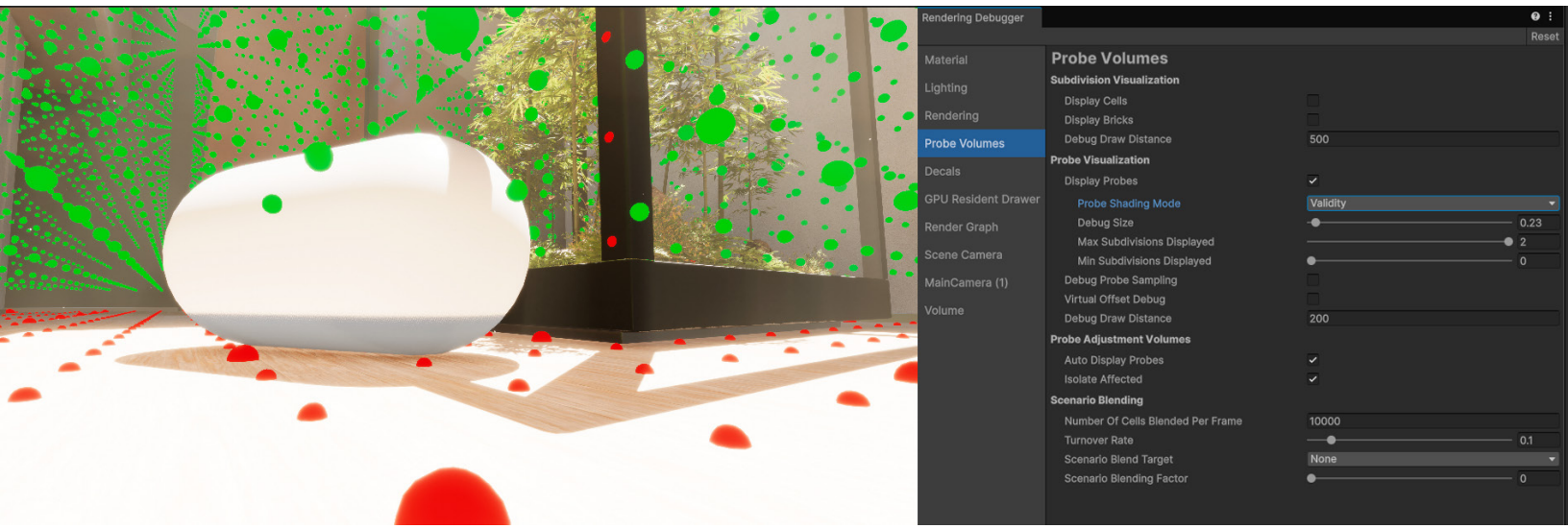
Let's explore a few common troubleshooting issues and solutions with APVs.

Fixing invalid probes

One common issue is probes being generated inside geometry. Probes that sample backfaces within objects may lead to incorrect lighting and are considered invalid.

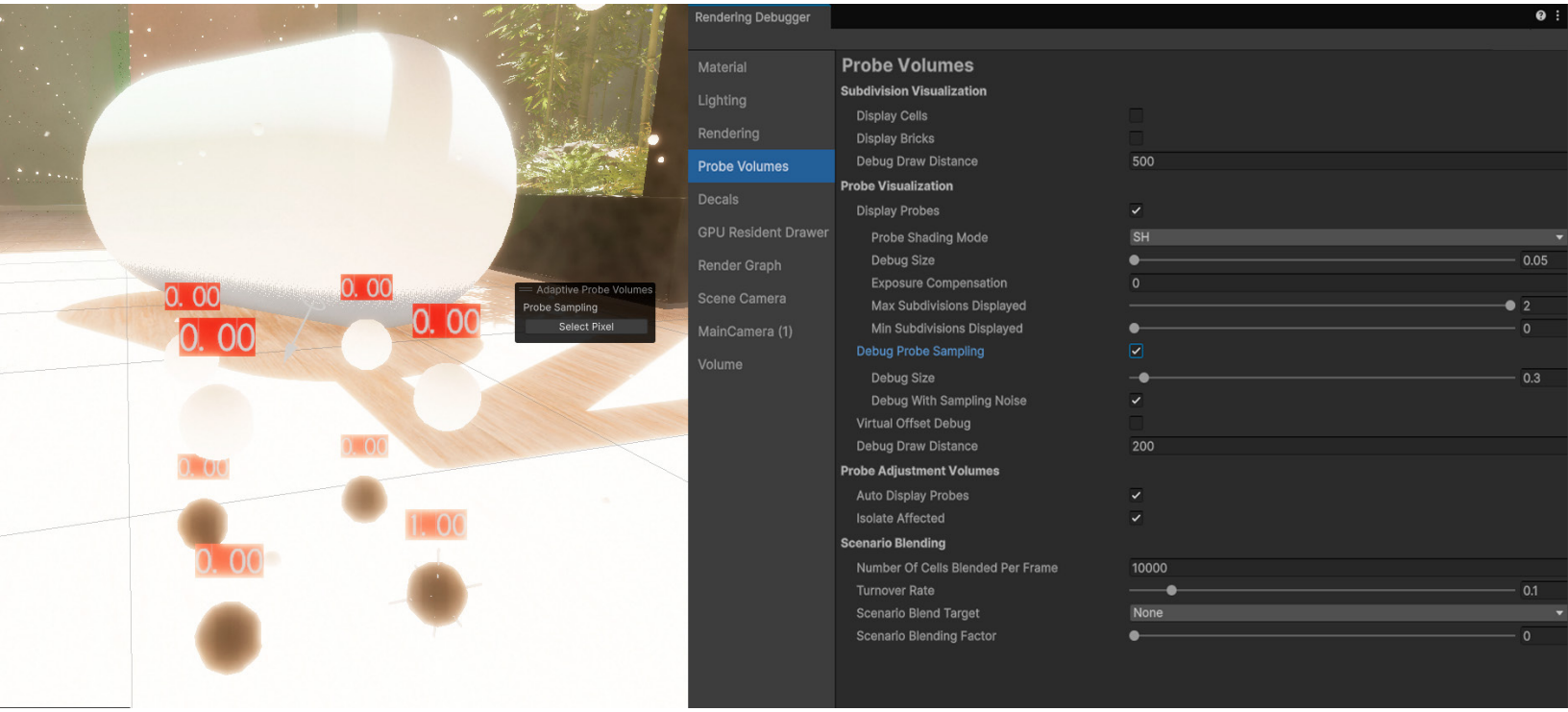
Use the **Rendering Debugger** (**Window > Analysis > Rendering Debugger > Probe Volumes**) to help troubleshoot. Enable the **Display Probes** option and set the **Probe Shading Mode** to **Validity** to visualize invalid probes.

For the above example, the Game view now shows the individual probes of the APV. Valid and invalid probes appear as green and red, respectively.



The Rendering Debugger shows invalid probes in red.

To isolate what might be causing the specific artifact, enable **Debug Probe Sampling**. In the Scene view, use **Select Probes** and then click on the underside of the capsule. This debug view shows which probes affect the given pixel.



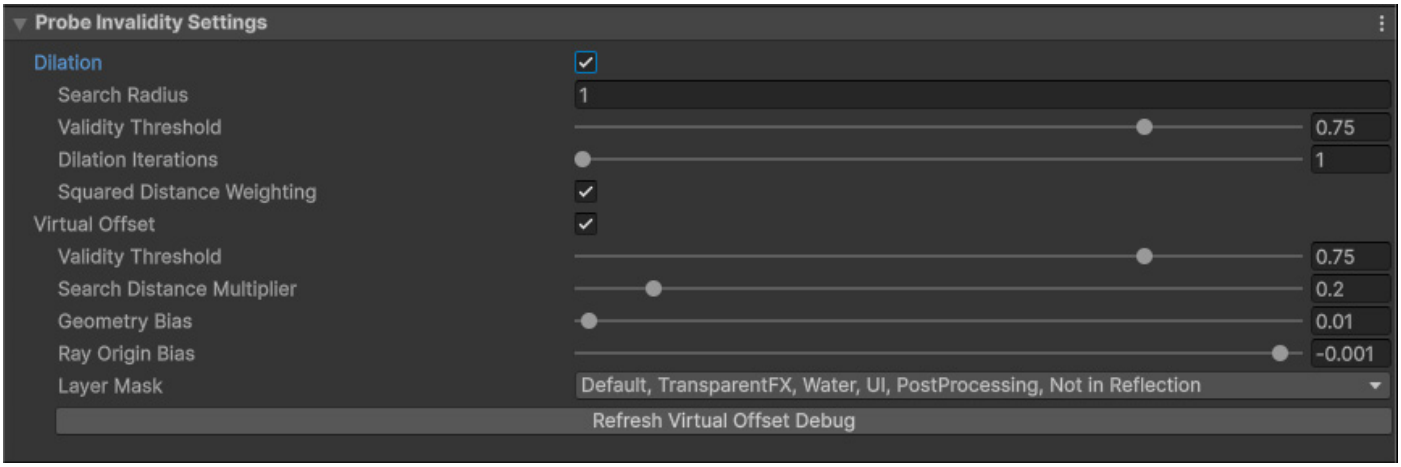
The Debug Probe Sampling option shows which probes cause the issue.

Adjust the **Probe Invalidity Settings** (**Lighting** window > **Adaptive Probe Volumes** panel) to help fix dark areas caused by invalid probes:

— **Virtual Offset** shifts each probe's capture position by casting rays to find a valid spot outside geometry. Adjust settings like **Validity Threshold, Search Distance Multiplier**, and **Layer Mask**. to improve probe placement. **Geometry Bias** pushes probes away from nearby surfaces, while **Ray Origin Bias** adjusts where probe rays start for better placement accuracy.

— **Dilation** settings help correct invalid probes by borrowing lighting data from nearby valid ones then filling in the missing data. Fine-tune this process using **Search Distance**, **Validity Threshold**, **Iterations**, and other dilation settings to control how strictly probes are validated. The **Squared Distance Weighting** can blend the borrowed lighting data more naturally.

Experiment with the settings and re-bake the Probe Volumes until the dark streaks disappear. Both Virtual Offset and Dilation only affect the baking process and do not impact runtime performance.



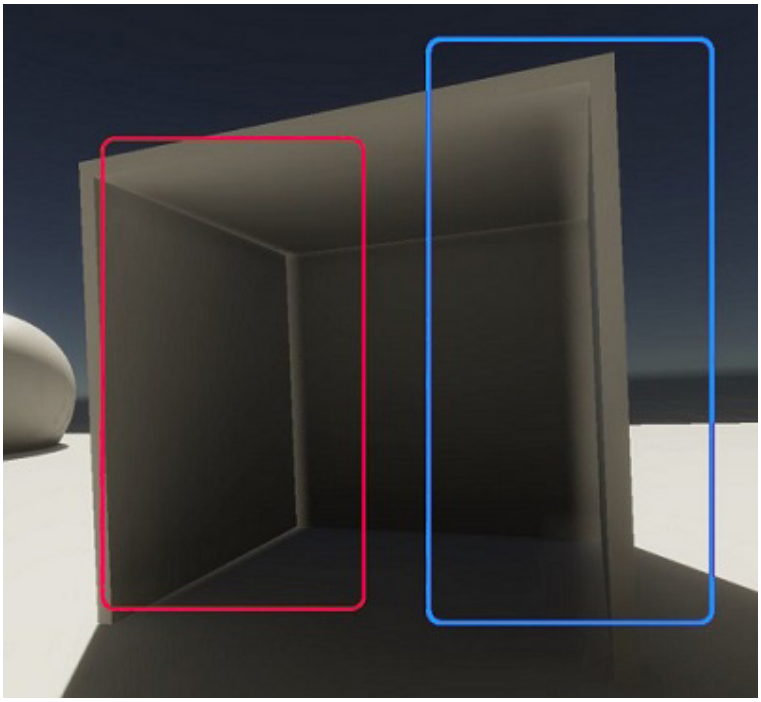
Use the Probe Invalidity Settings to fix dark areas caused by invalid probes.

The artifact is removed after adjusting Virtual Offset and Dilation settings.

Fixing light leaks

Similar to the dark streaks from invalid probes, “light leaks” are artifacts that can occur when geometry receives light from a Light Probe that isn’t visible to the geometry (e.g., the light probe is on the other side of a wall).

This happens when the APV density and wall thickness in your environment mismatch, causing interpolation that makes exterior light appear as if it’s passing through what should be solid walls.



Light leaks appear as artifacts.

Light leaks manifest themselves as areas that are too light or dark, often in the corners of a wall or ceiling. APVs use regular grids of light probes, so light probes might not follow walls or be at the boundary between different lighting areas.

Use these strategies to fix light leaks:

— Increase wall thickness to ensure better probe placement. Adjust walls so their width is closer to the distance between probes in the local brick.

— Use an Adaptive Probe Volumes Options override (below) to adjust sampling positions.

— Enable Rendering Layer Masks (below) to prevent objects from using probes on incorrect layers.

— Modify Baking Set properties to optimize probe spacing and search distances.

Using Probe Adjustment Volumes

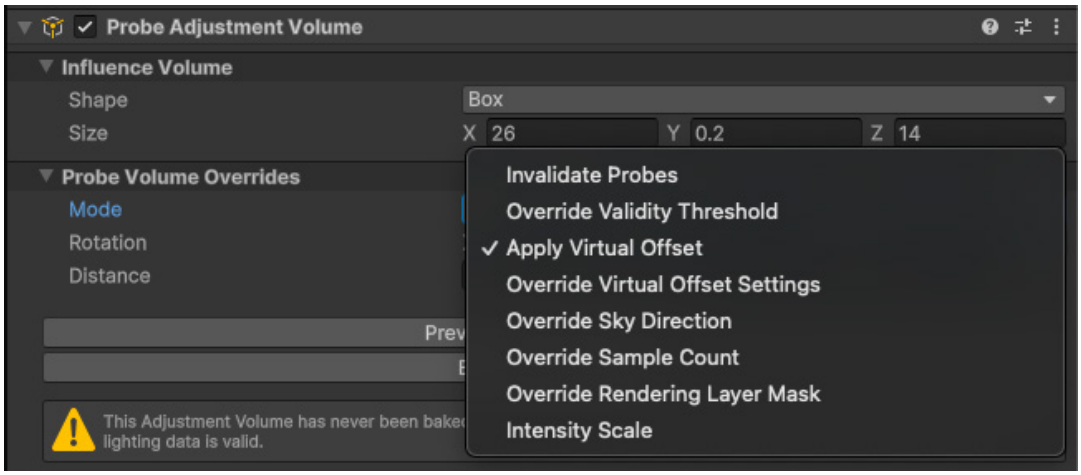
The Volume system includes a **Probe Adjustment Volume Option override**. This override provides localized control over light probes and APVs, allowing you to adjust probe-based lighting in specific areas of a scene.

Use the **Influence Volume** settings to define a **Box** or **Sphere** to affect. Then, choose a **Mode** to adjust intensity, sampling, sky contribution, and/or probe validity:

— **Invalidate Probes:** This mode marks probes as invalid, preventing them from contributing to lighting calculations.

- **Override Validity Threshold:** This changes how probes are validated, reducing artifacts from invalid probes.
- **Apply Virtual Offset:** This overrides the Lighting Window's Rotation and Distance for local virtual offsets.
- **Override Virtual Offset Settings:** This mode overrides the Lighting Window settings for **Validity Threshold**, **Geometry Bias** (pushes probe away from nearby surfaces), and **Ray Origin Bias** (adjusts where the probe rays starting point).
- **Intensity Scale:** This mode adjusts the brightness within the volume.
- **Override Sky Direction:** This forces probes to use a specific sky direction.
- **Override Layer Mask:** This allows the probes to use a different Layer Mask for selecting which probes are affected.

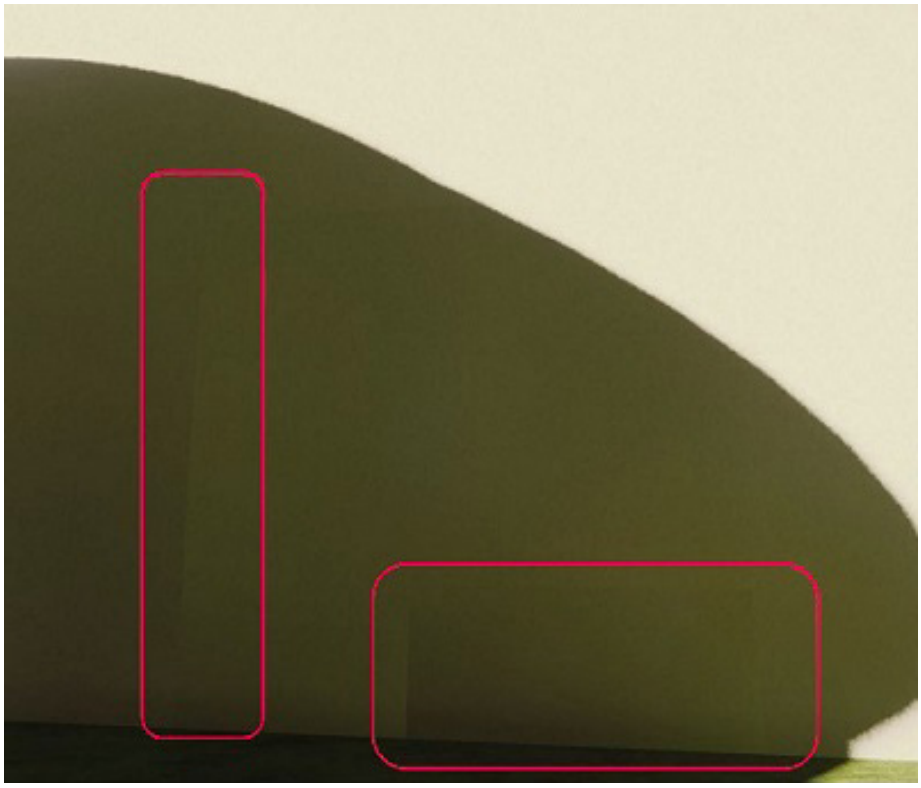
Volumes only affect the scene if the camera is near or inside the volume. Refer to Understand volumes and Probe Volumes Options Override reference for more information on **Probe Volumes Options** settings.



This override provides localized control of light probes and APVs.

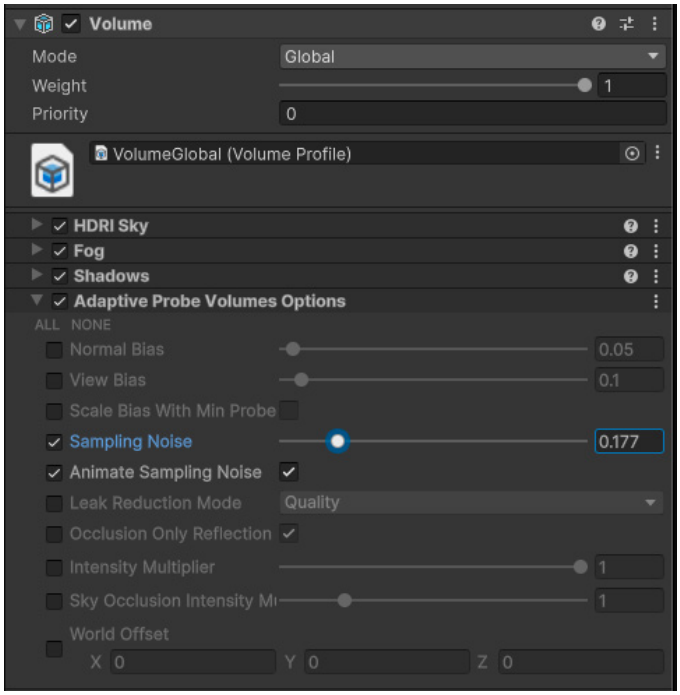
Fixing seams

Seams appear when two adjacent probe bricks have different densities, creating abrupt lighting transitions. Though HDRP automatically fixes seams, if it persists, you can enable **Sampling Noise** in an Adaptive Probe Volumes Options override to help soften the transition.



An example of a seam between probe bricks.

Adjusting Baking Set properties and probe density can further refine results. Most APV issues appear during baking, so make adjustments before finalizing the lighting for the most efficient scene rendering.



Adjust the Sampling Noise to fix a seam.

Rendering Layer Masks

Another way to fix issues with APVs is to use Rendering Layer Masks. Rendering Layer Masks act as filters to ensure objects sample lighting from relevant probes, reducing light leaks. APVs support up to four Rendering Layer Masks.

During light baking, Unity automatically assigns a Rendering Layer Mask to each probe based on nearby objects. Probes near objects with a specific Rendering Layer Masks (e.g. “Interior”) will inherit that layer.

At runtime, objects only sample lighting from probes that share their Rendering Layer Mask. If no matching mask is found, they sample from all valid surrounding probes.

A common cause of light leaks is when pixels blend between interior and exterior probes, with dramatically different lighting. To prevent this, you can create separate Rendering Layer Masks for the interior and exterior.

Note that this feature requires **Light Layers** to be enabled in the HDRP Asset.

To enable Rendering Layer Masks for APVs:

1. Assign **Rendering Layer Masks** to your renderers in the **Mesh Renderer** component.
2. Open the **Lighting** window and navigate to **Adaptive Probe Volumes** > **Rendering Layer Masks**.
3. Assign up to four Rendering Layer Masks to the list. When baking, the light probes will be assigned to the mask based on nearby objects.

In HDRP, you can use the Rendering Debugger to visualize which layers are assigned to an object:

— Go to the Material tab.

— Set the **Material** field to **Common > Rendering Layers**.

Sky occlusion in APVs

Sky occlusion in APVs dynamically adjusts indirect lighting by reducing sky light contribution in shadowed or covered areas (e.g., tunnels, rooms, or beneath overhangs). This creates more realistic indirect lighting and helps prevent excessive brightness in areas that should receive less sky influence.

Since the ambient probe can update at runtime, this allows for real-time lighting transitions, such as smoothly shifting from daylight to night. This allows you to simulate a day-night cycle using only APVs.

Enable sky occlusion

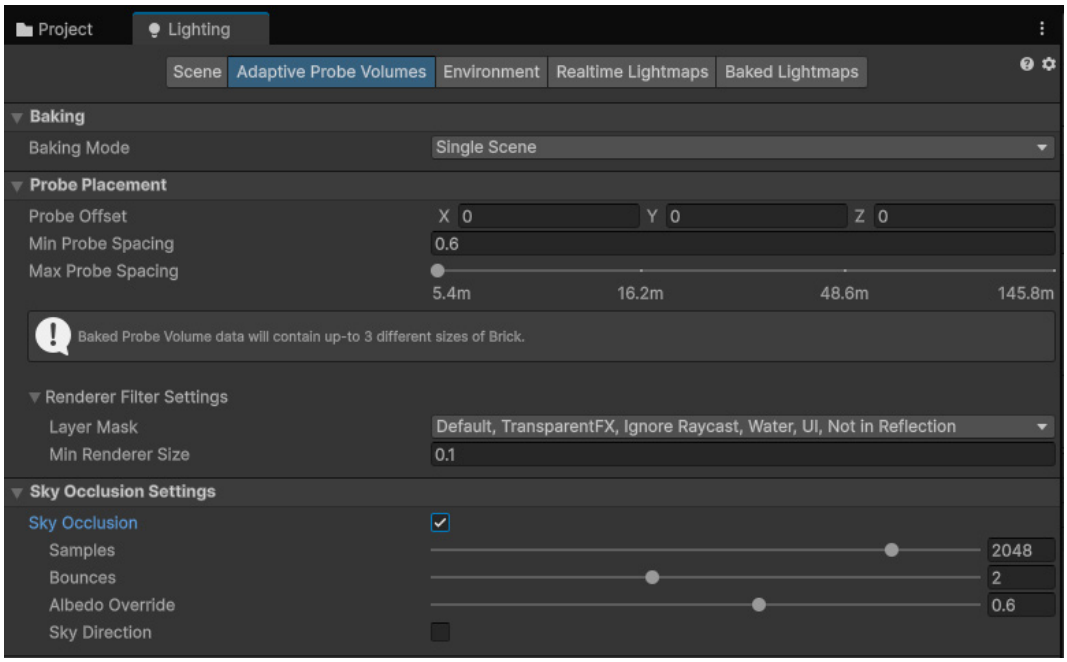
To enable sky occlusion:

— Enable the **Progressive GPU Lightmapper** in the Lighting window. This feature is not supported if you use Progressive CPU.

— Enable **Sky Occlusion** in the Adaptive Probe Volumes panel and rebake the volume.

When enabled, Unity bakes an additional static sky occlusion value into each probe, representing the indirect sky light received. At runtime, Unity combines this with the ambient probe's sky color, which updates dynamically when the sky changes.

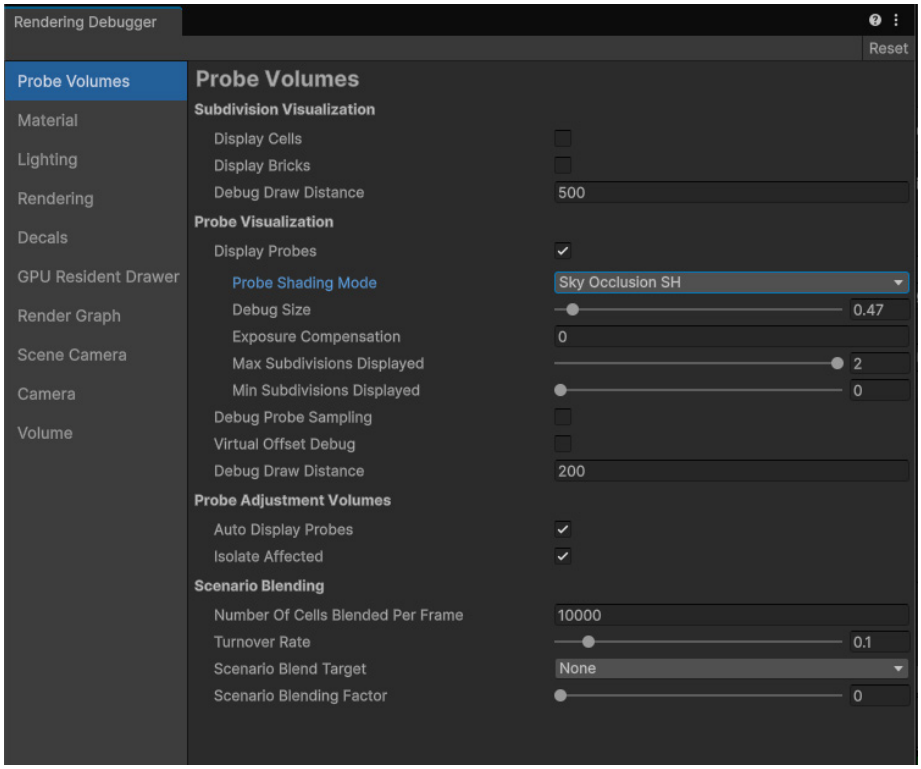
To ensure that the ambient probe updates at runtime, open the **Volume Profile Asset** and navigate to **Visual Environment > Sky section.** Set the **Ambient Mode** to **Dynamic**.



Enable the Sky Occlusion option in the Lighting window.

Debugging sky occlusion

You can visualize how sky occlusion affects APV lighting in the **Rendering Debugger**. In the **Probe Volumes** section, enable **Display Probes** and set **Probe Shading Mode** to **Sky Occlusion SH.**



Debug sky occlusion in the Rendering Debugger.

Compare the following images with **Sky Occlusion** enabled (left) and disabled (right). With Sky Occlusion enabled, probes in open areas appear brighter, while probes blocked by scene geometry are darker, correctly reducing indirect sky lighting. With Sky Occlusion disabled, all probes receive uniform sky lighting, making shadowed areas appear unnaturally bright.

With and without the Sky Occlusion setting enabled (left to right)

This difference is most noticeable in areas shielded from the sky. Once the sky occlusion is baked, the scene lighting updates based on ambient probe changes. In HDRP, the ambient probe is updated in real-time only when using the Color or Gradient Mode (not the Skybox mode).

Sky direction

By default, Unity uses an object's surface normal as the sky direction when sampling the ambient probe. This can be inaccurate in enclosed spaces where the sky light bounces off other objects.

Enable **Sky Direction** in the **Adaptive Probe Volumes** panel to allow Unity to calculate and store precise sky lighting directions per probe. This can take bounce lighting into account in areas where probes don't have a direct line of sight to the sky or when the light comes from a specific direction, such as through a window. This improves lighting accuracy at the cost of additional bake time and memory.

Optimizing APV data loading

To optimize APV data loading at runtime, you can either stream APV data or load it from AssetBundles/Addressables.

Stream APV data

Streaming allows large-scale APV lighting by loading only the necessary data as the camera moves. This reduces memory usage and enables baking APV data larger than available CPU/GPU memory. HDRP streams data from the `StreamingAssets` folder and loads only the cells within the camera's view frustum.

To enable APV streaming:

1. Navigate to **Edit > Project Settings > Quality > HDRP**.
2. Select a **Quality Level**, then expand **Lighting > Light Probe Lighting**.

You can now enable two types of streaming:

— Select **Enable GPU Streaming** to stream from CPU memory to GPU memory.

— Select **Enable Disk Streaming** to stream from disk to CPU memory. You must enable **Enable GPU Streaming** first.

Streaming settings can be further configured in the HDRP Asset settings.

The smallest section HDRP loads and uses is a cell, which is the same size as the largest brick in an Adaptive Probe Volume. You can influence the size of cells in an APV by adjusting the density of light probes. Use the Rendering Debugger to visualize APV cells and debug streaming.

Load APV data from AssetBundles or Addressables

To load only required APV data at runtime, store it in AssetBundles or Addressables instead of Streaming Assets.

1. Enable AssetBundle-Based APV Loading, navigate to **Project Settings > Graphics > Pipeline Specific Settings > HDRP.**
2. Enable **Probe Volume Disable Streaming Assets** (disables disk streaming).
3. Bake your lighting.
4. Add the scene to an AssetBundle.

Note: Disabling Streaming Assets may increase memory usage, as Unity keeps all Baking Set lighting data in memory, even if some scenes are not loaded.

Baking light probes at runtime

Unity 6 introduces a new API for baking light probes, offering more control, efficiency, and flexibility compared to previous methods. You can now control how many probes are baked at a time, allowing you to balance execution time versus memory usage.

The API features a high-performance **GPU-accelerated backend**, leveraging **OpenCL** for probe baking. This system extracts necessary lighting data directly from the scene before baking and does not support procedural geometry.

See these documentation pages for more information:

— RadeonRaysContext

— RadeonRaysProbeIntegrator

— RadeonRaysProbePostProcessor

Watch the GDC 2023 session, Efficient and impactful lighting with Adaptive Probe Volumes, for more about Adaptive Probe Volumes. You can also see the Adaptive Probe Volume documentation.

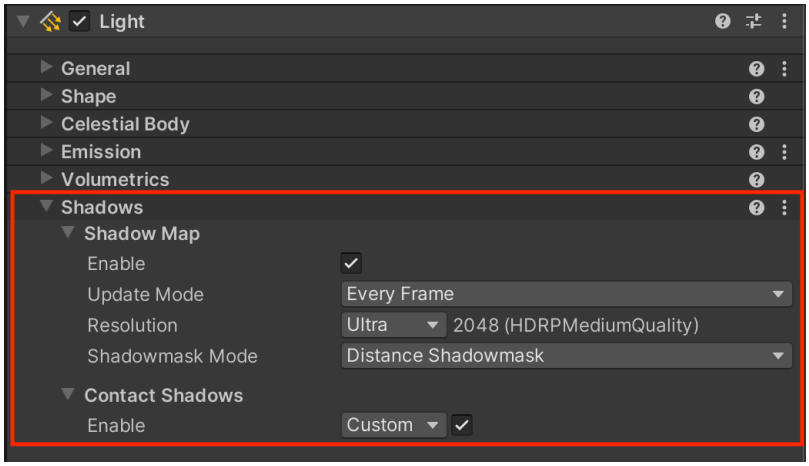
Shadows

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.

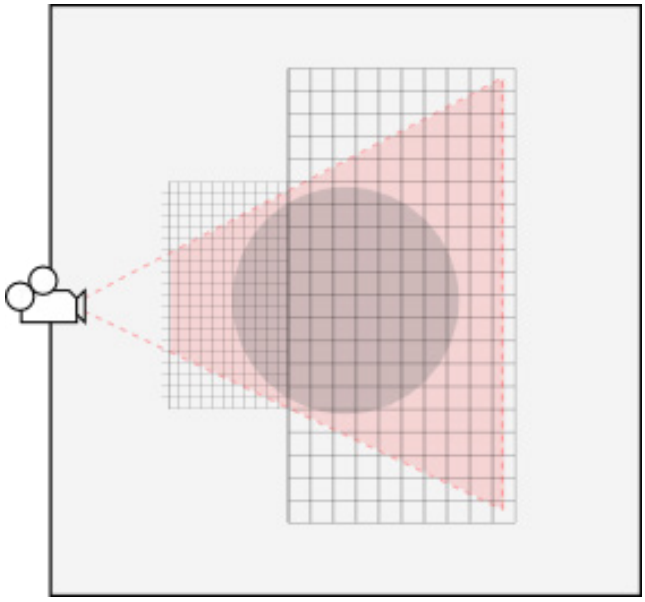


Shadow settings per light

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

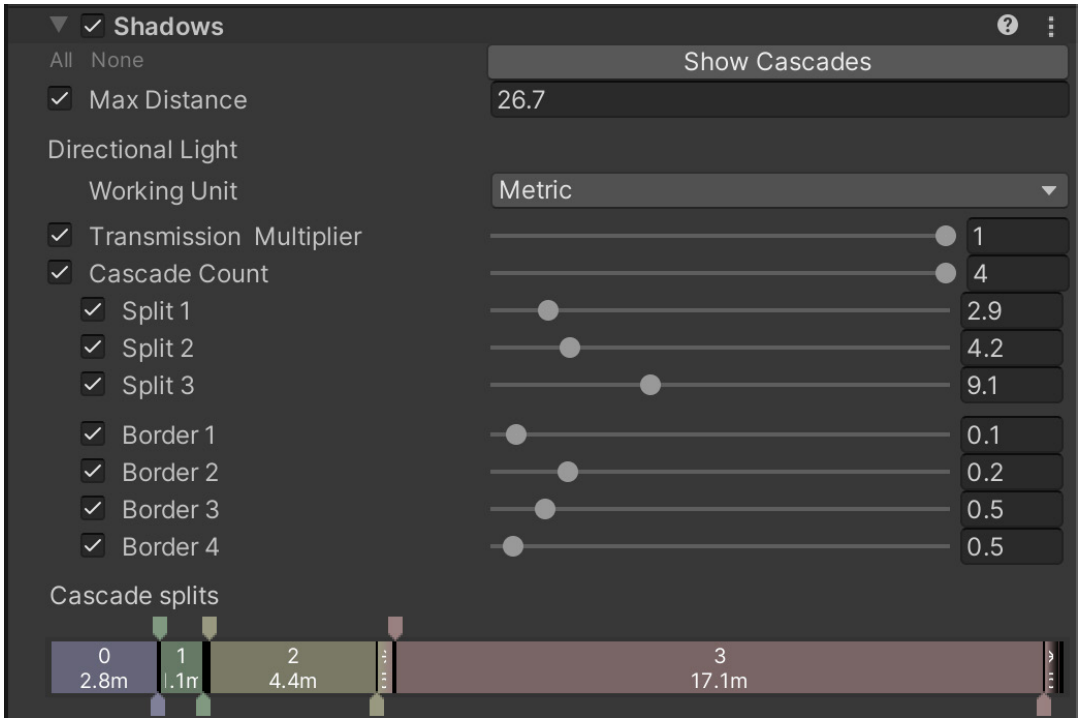


Unity solves this with cascaded shadow maps. It splits the camera frustum into zones, each with its own shadow map. This reduces the effect of perspective aliasing.

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

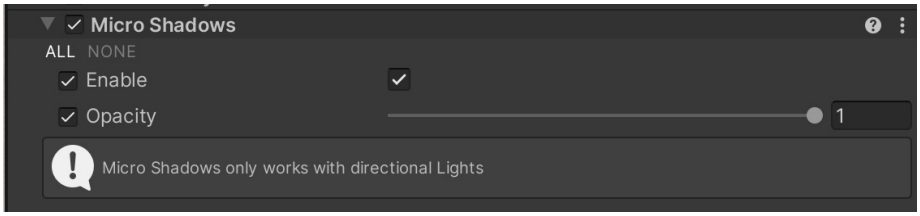
Contact Shadows

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. The **Micro Shadows** override uses the normal map and ambient occlusion map to render really fine surface shadows without using the mesh geometry itself.



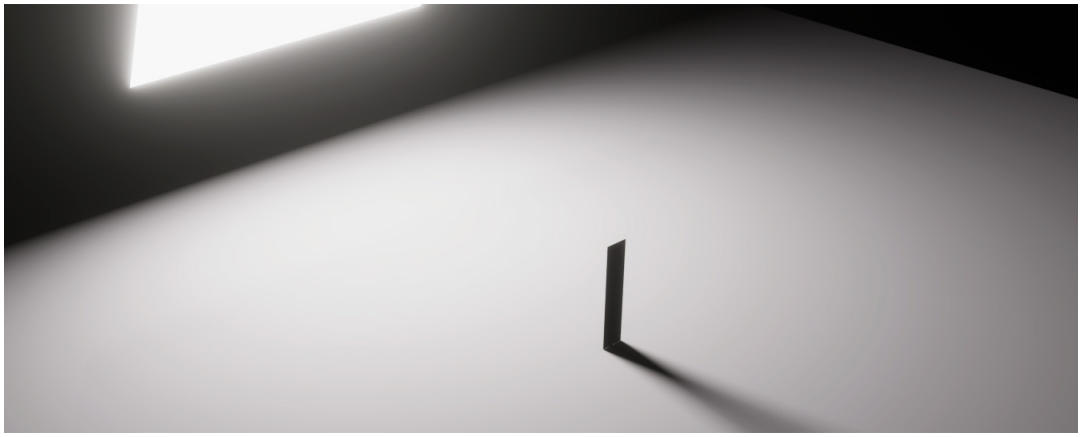
Micro Shadows override

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

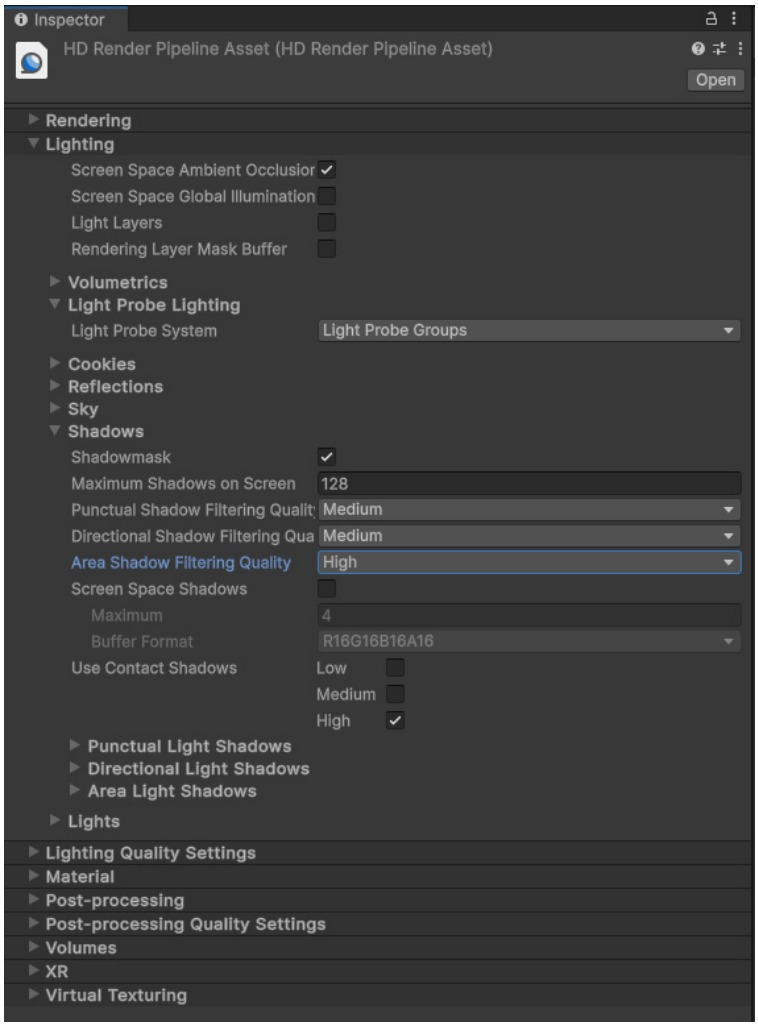
Micro shadows add extra contrast to this bed of foliage.

Area light soft shadows

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.



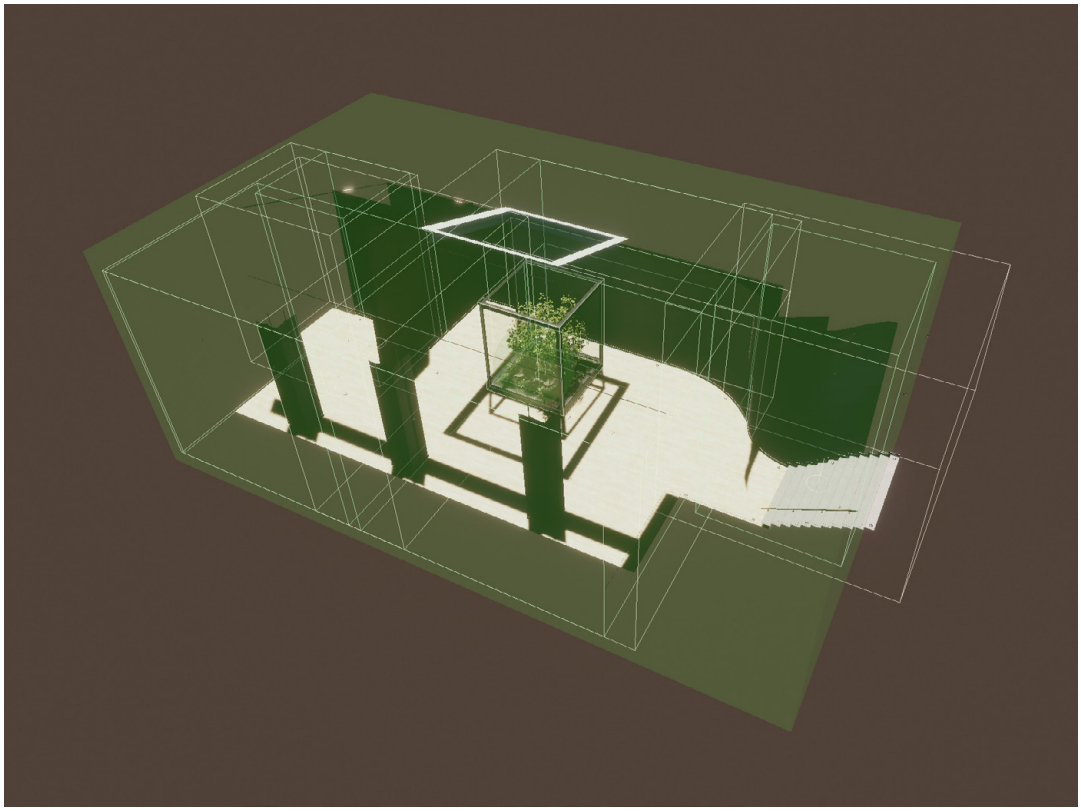
Area Shadow Filtering Quality

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.



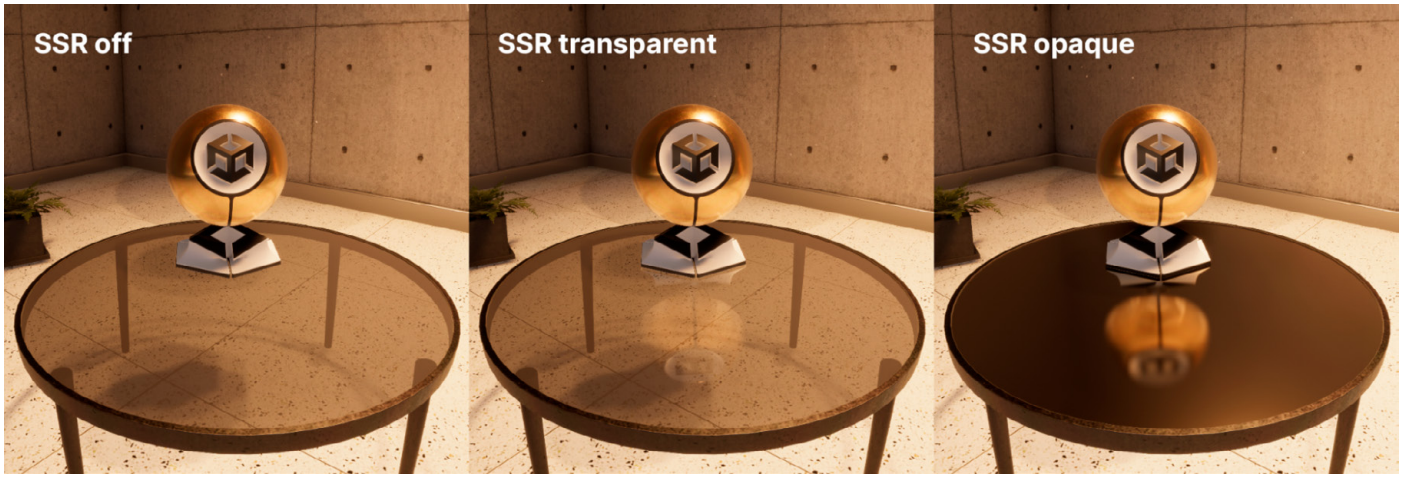
Influence Volumes determine where reflection probes create reflections.

Screen space reflections

Screen space reflections (SSR) 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.

The Screen Space Reflection override



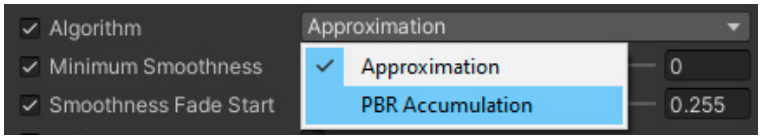
Screen space reflections work for transparent and opaque materials.

Make sure you have the **Screen Space Reflection** setting 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 SSRs. 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 the Screen Space Reflection override 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 SSRs.

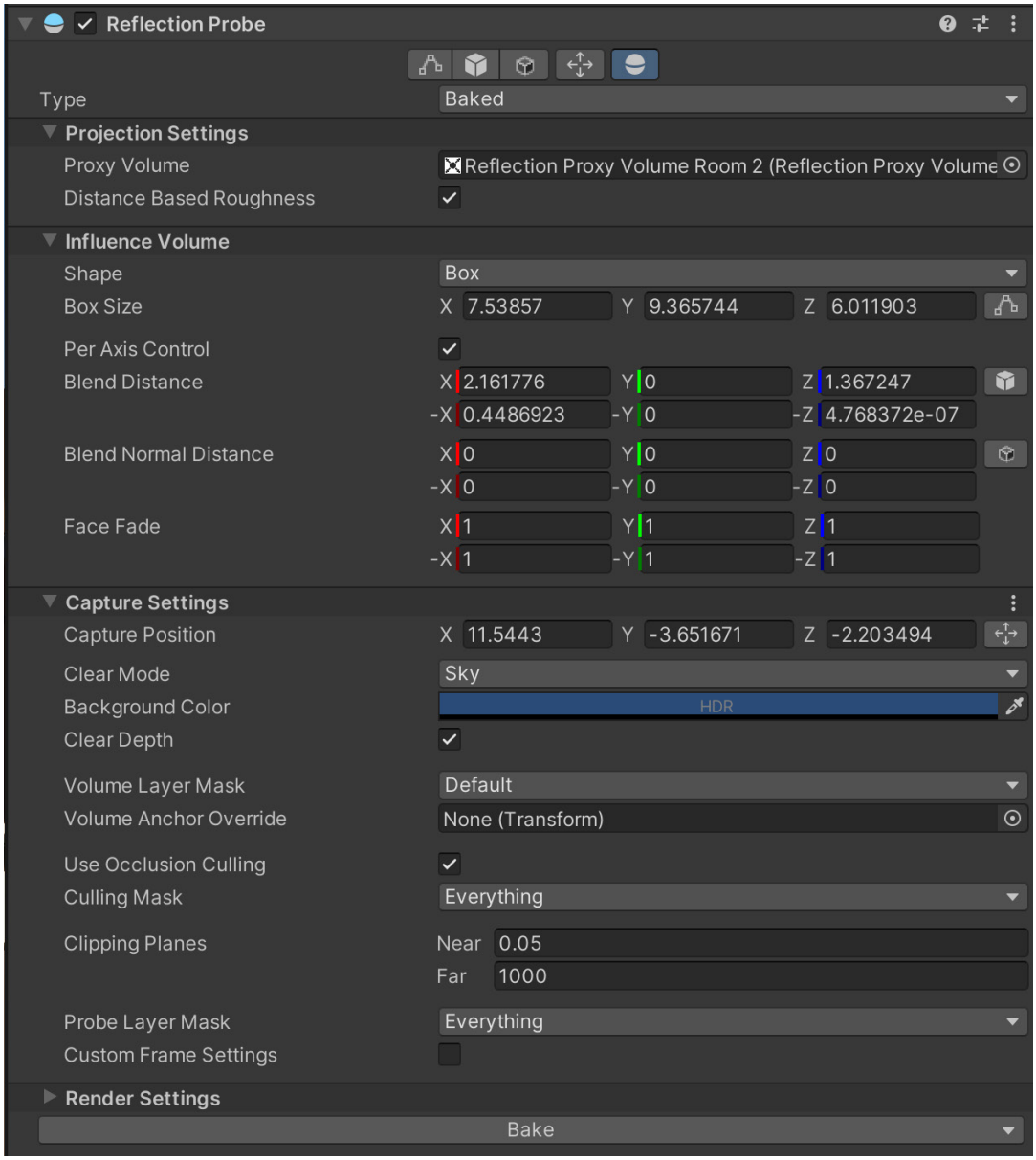
Reflection probes

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.



The 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. You can do this by **overriding** camera settings globally or per probe. You can also create a script to time slice the update.

The **Time Slicing** property was added to the Reflection Probe component in Unity 2022. To see this property, set the reflection probe's **Type** to **Realtime**. Instead of updating the entire Cubemap at once, this feature updates one face per frame, spreading out the processing load and improving efficiency. This can save on performance costs.

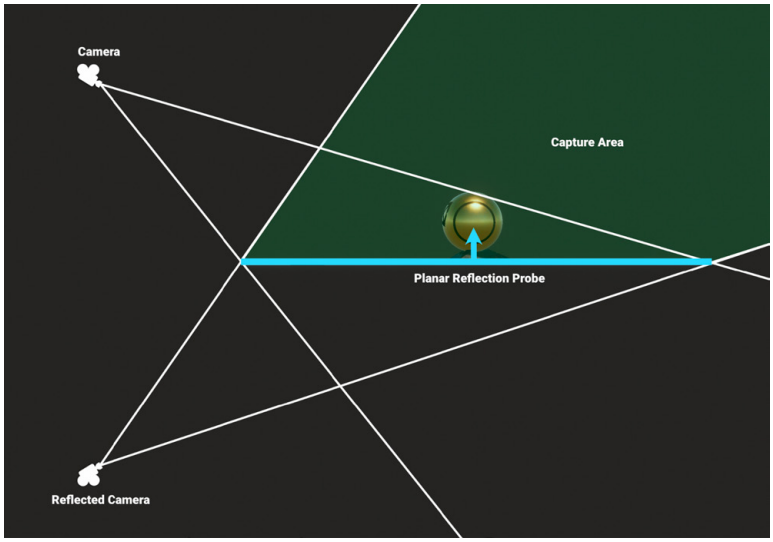
HDRP now replaces the cube reflection probe cache array with a **2D texture atlas cach**e in octahedral projection. This allows you to control the resolution for each reflection probe to save memory.

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

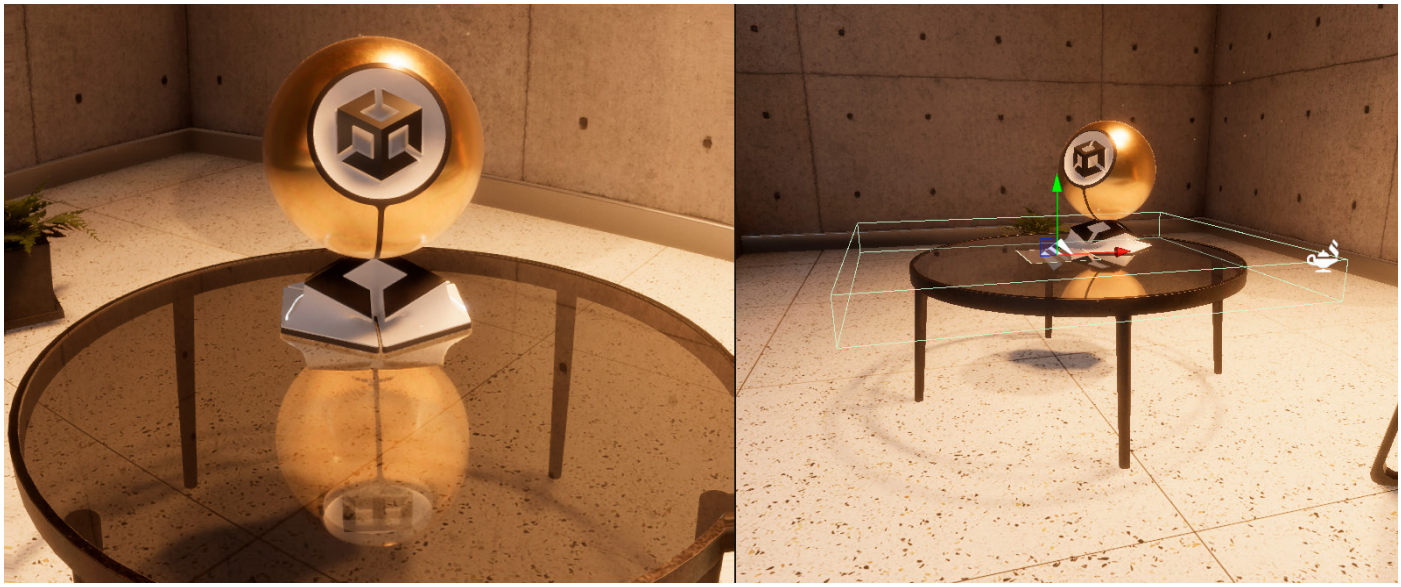
The Planar Reflection Probe component allows you to recreate a flat, reflective surface, taking surface smoothness into account. This is perfect for a shiny mirror or floor.

Though a planar reflection probe shares much in common with a standard reflection probe, there are slight differences between how the two probe types work. Rather than capture the environment as a cubemap, a planar reflection probe recreates the view of a camera reflected through the probe's mirror plane.



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

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



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 fall back to the sky reflection.

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

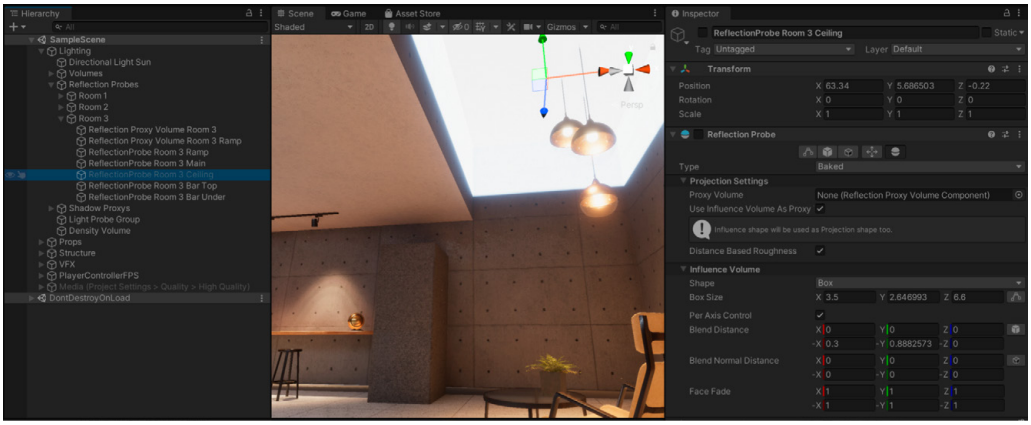
Reflection hierarchy

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 SampleScene. 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.



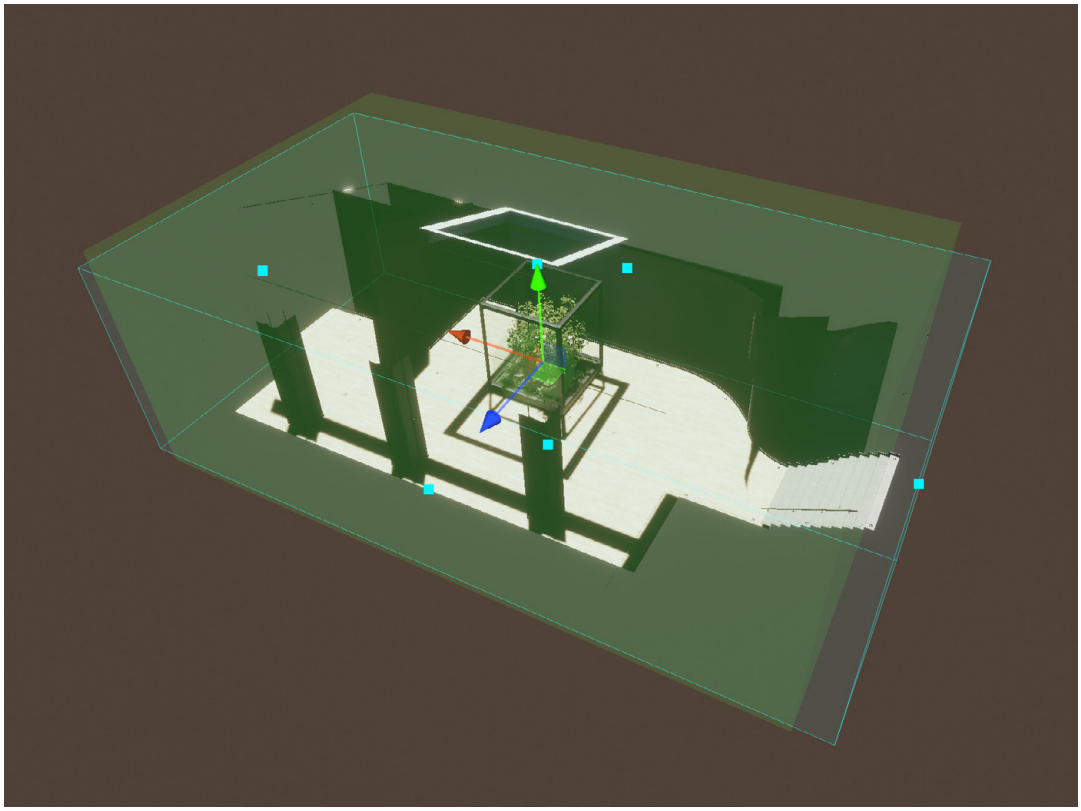
Disabling the reflection probe on the Room 3 ceiling causes unwanted light leaking.

For more details about determining Reflection hierarchy, see the Reflection in the HDRP documentation page.

Proxy Volume

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



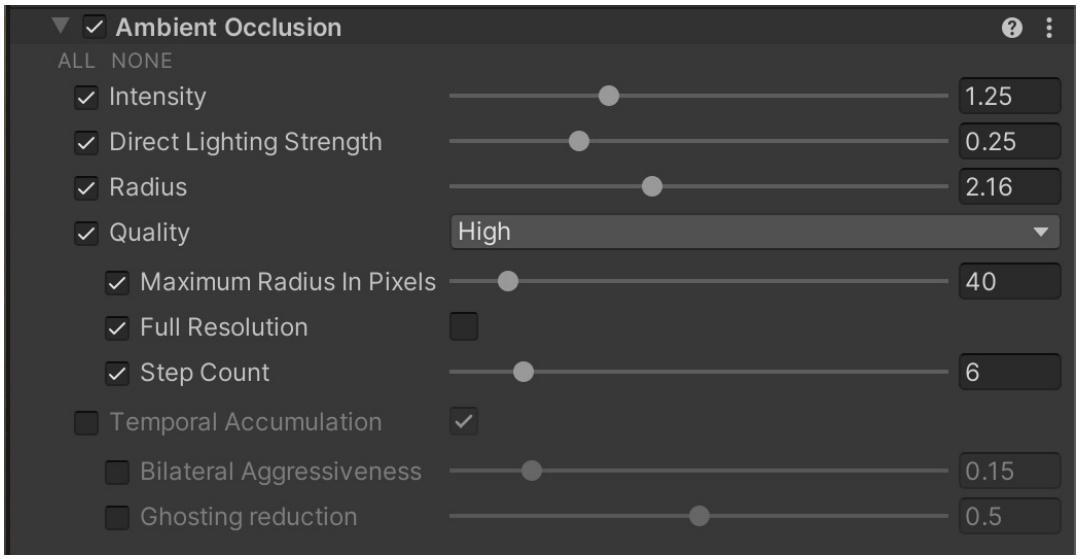
A Proxy Volume reprojects the cubemap to match the room's world space position.

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** override 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**.

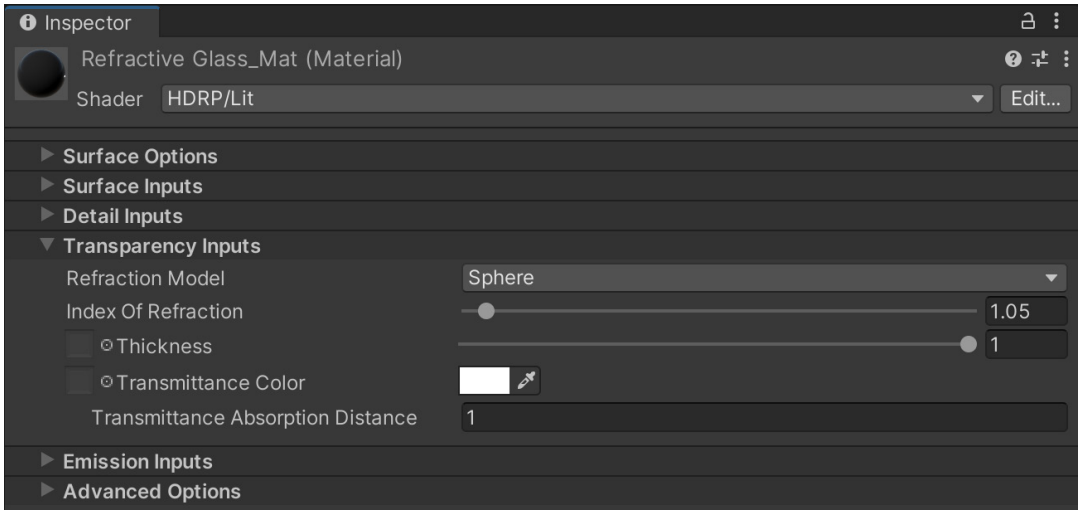
Screen space refraction

The Screen Space Refraction override helps to simulate how light behaves when passing through a denser medium than air. Screen space refraction in HDRP uses the depth and color buffer to calculate refraction through a transparent material like glass.

The 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

Light rendering layers

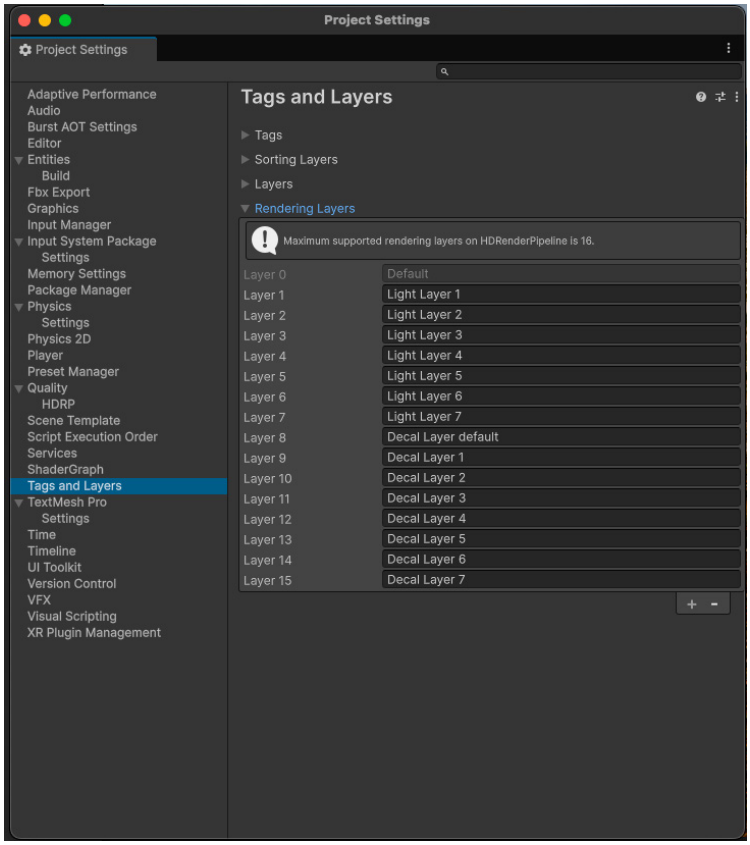
Rendering layers in HDRP allow you to control which lights, decals, and effects influence specific Renderers in your scene. Rendering layers are LayersMasks which limit Lights or effects so that they only affect specific Renderers.

16 versus 32 rendering layers

While any renderer (mesh or object) can belong to up to 32 rendering layers, HDRP lights and effects only support the first 16 layers. You can assign the renderers to any of the 32 layers for scripting or organization purposes, but only the first 16 layers will influence lighting, decal, or other HDRP effects.

Configuring rendering layers

To create and name rendering layers, navigate to **Edit > Project Settings > Tags and Layers**. Open **Rendering Layers** and assign custom names to layers for easier management.



Enable the Rendering Layers feature.

To enable rendering layers for Lights, navigate to **Edit > Project Settings > Graphics > Pipeline Specific Settings > HDRP**. Enable Light Layers in the **Frame Settings (Default Values)** under Lighting.

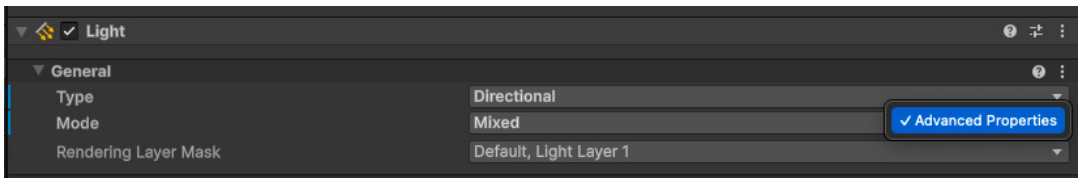
Then, enable **Light Layers** in the **HDRP Asset**.

To use rendering layers with decals, refer to the **Decal Layers** documentation.

Using rendering layers with lights

Once **Light Layers** are enabled, you can assign them to lights and Mesh Renderers.

In the **Inspector**, select a light, then enable **Advanced Properties** in the **General** section from the options menu (⋮) in the Inspector.



Reveal the rendering layers in the Advanced Properties.

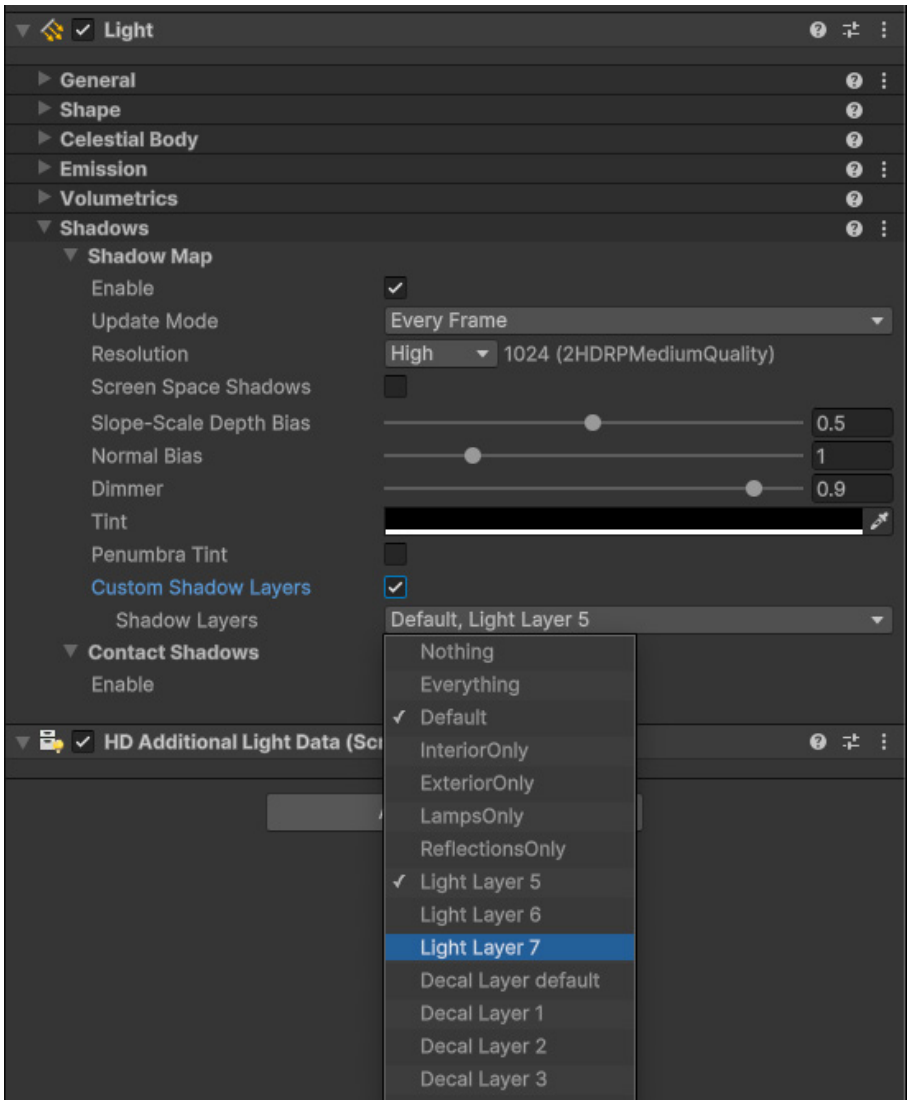
Use the **Rendering Layer Mask** dropdown to specify which **Light Layers** the light affects.

To assign a Mesh Renderer or Terrain, use the **Rendering Layer Mask** dropdown in its **Inspector**. A light only affects a Mesh Renderer or Terrain if both use the same rendering layer.

Using rendering layers for shadows

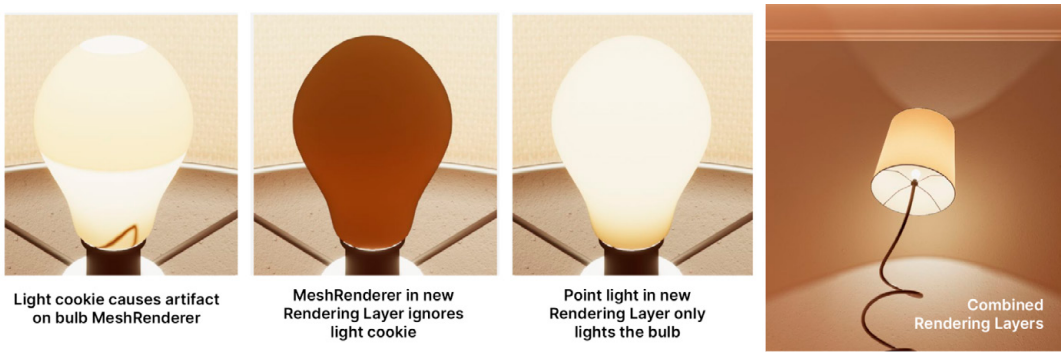
HDRP's default behavior is to link a light's rendering layer with its shadow rendering. This means that anything the light illuminates will cast a shadow based on that same layer.

If you need to separate the shadows from the lighting, you can enable **Custom Shadow Layers** in the Light's **Shadows** settings. This feature allows you to assign a different rendering layer specifically for shadows.



Isolate a light effect using Rendering Layers.

This example illustrates how you might use shadow rendering layers in practice. Here, a light cookie causes unwanted self-shadowing on a bulb (left). By assigning the bulb's Mesh Renderer to a separate Rrendering layer, the light – and light cookie – no longer affect the bulb (center). Then, a separate Point light is added to the same rendering layer as the bulb and restores the intended lighting (right).



Use rendering layers to customize lighting and shadows.

For more details, refer to this expert guide on high-quality light fixtures in Unity.

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 acceleration has now made ray tracing possible for real-time applications.

HDRP uses a hybrid ray tracing system, blending ray traced and rasterized rendering. It supports DirectX 12 and requires compatible NVIDIA RTX™ and AMD Radeon™ PRO GPUs.

See Getting started with ray tracing for a specific list of system requirements.

New in Unity 6

- **Production-ready ray tracing:** HDRP supports ray tracing for DirectX 12 and specific console platforms. Consult console-specific documentation for more information.
- **Enhanced ray traced effects:** Improvements in ray traced ambient occlusion, global illumination, reflections, and shadows provide more accurate and realistic lighting and shading.
- **Solid-angle culling**: You can now enable Solid Angle culling, to discard smaller and distant instances from the acceleration structure. This is enabled via setting in the HDRP Ray Tracing Volume setting, as well as the scripting API via EnableSolidAngleCulling. Enabling solid-angle culling is a quick way to improve ray tracing performance.
- **Inline ray tracing and shader queries:** You can now leverage hardware-accelerated ray queries via inline ray tracing and shaders. This unlocks new possibilities for custom effect authoring, such as custom ray traced shadows.

— **Decal support:** Decals are now fully integrated into ray traced reflections and global illumination. For example, a large yellow decal will now tint nearby surfaces with warm bounced light for more realistic renders.

  Decal projectors can now be used with the HDRP Path Tracer, allowing you to add water puddles, dirt, bullet holes, and other material effects without modifying the underlying surface.

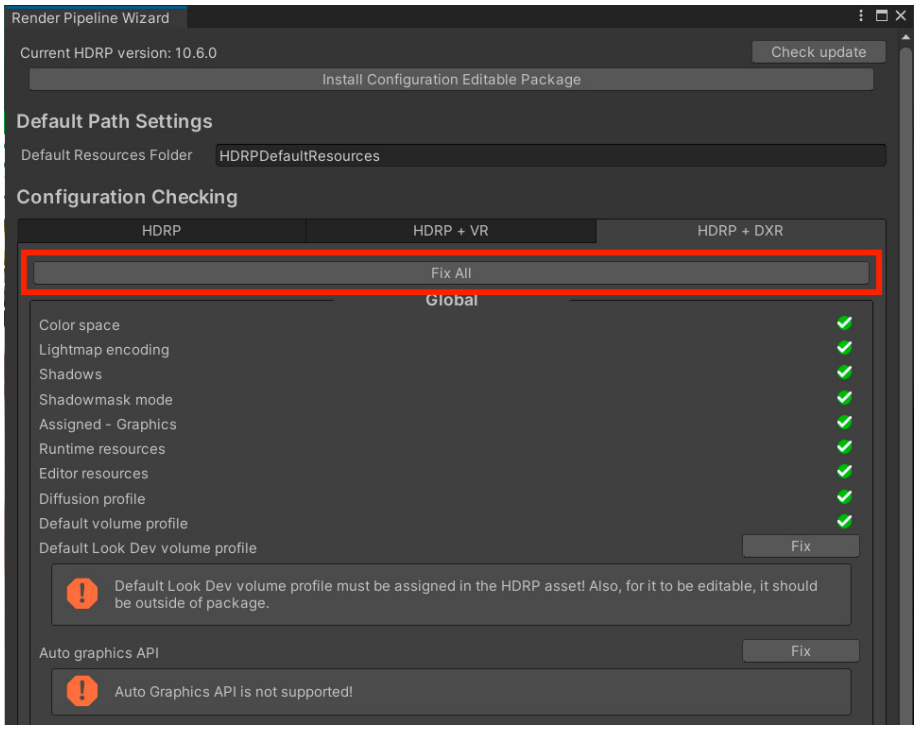
— **Optimized Acceleration Structures:** Ray tracing now features reduced memory usage for complex scenes, improving real-time performance. This is a result of multiple optimizations, including a new small BLAS allocator (reduced memory waste for small meshes), BLAS compaction (reducing memory usage for static meshes) and a new minimize memory flag.

— **Expanded Hardware Support:** Full ray tracing hardware acceleration is available on NVIDIA GeForce RTX GPUs, NVIDIA RTX PRO GPUs, select AMD Radeon™ RX and PRO Series GPUs, on DirectX 12 and DXR capable GPUs for Windows PC, as well as specific console platforms (consult console-specific documentation for more information).

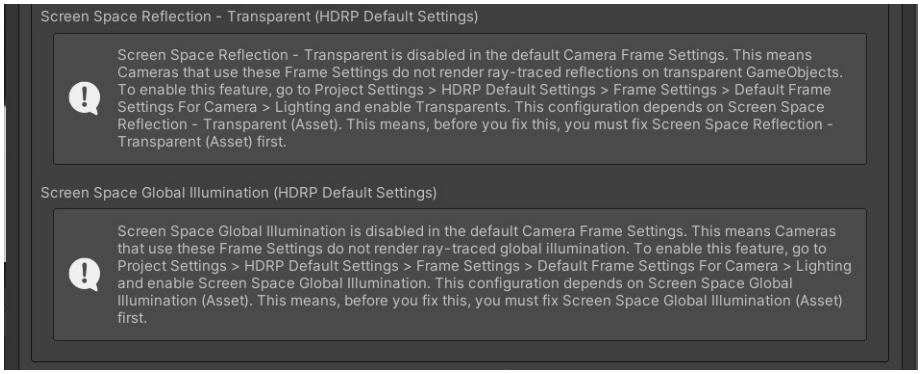
Setup

In order to enable ray tracing, 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.

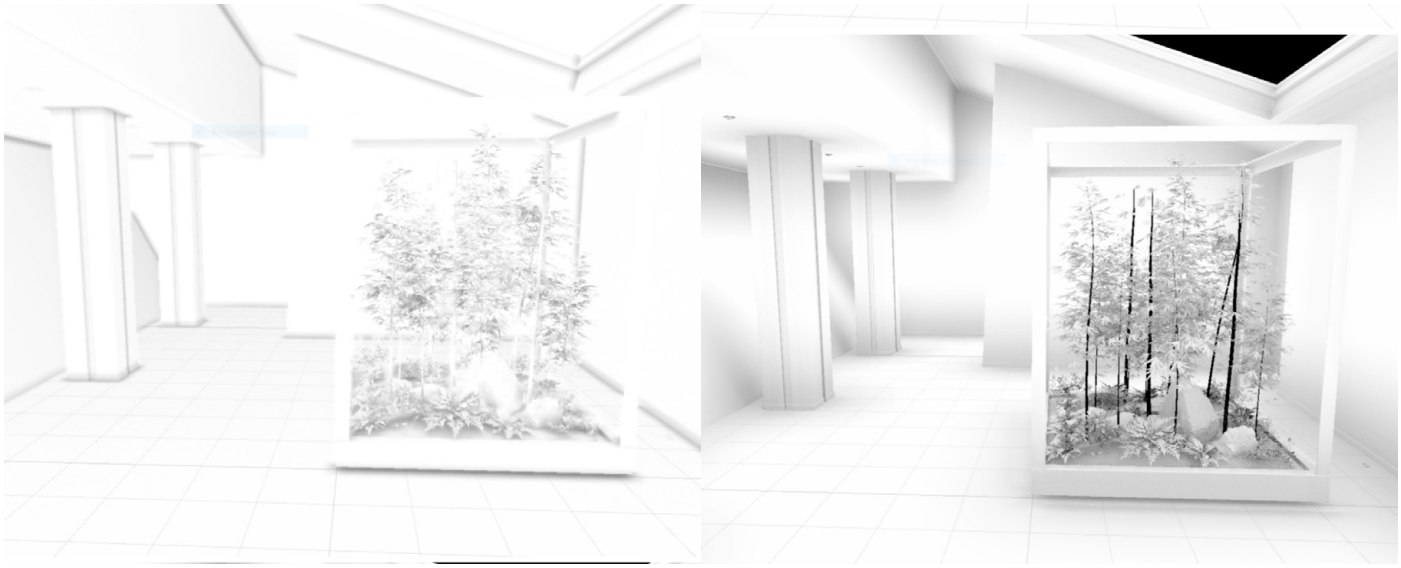
You can also set up ray tracing manually.

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.



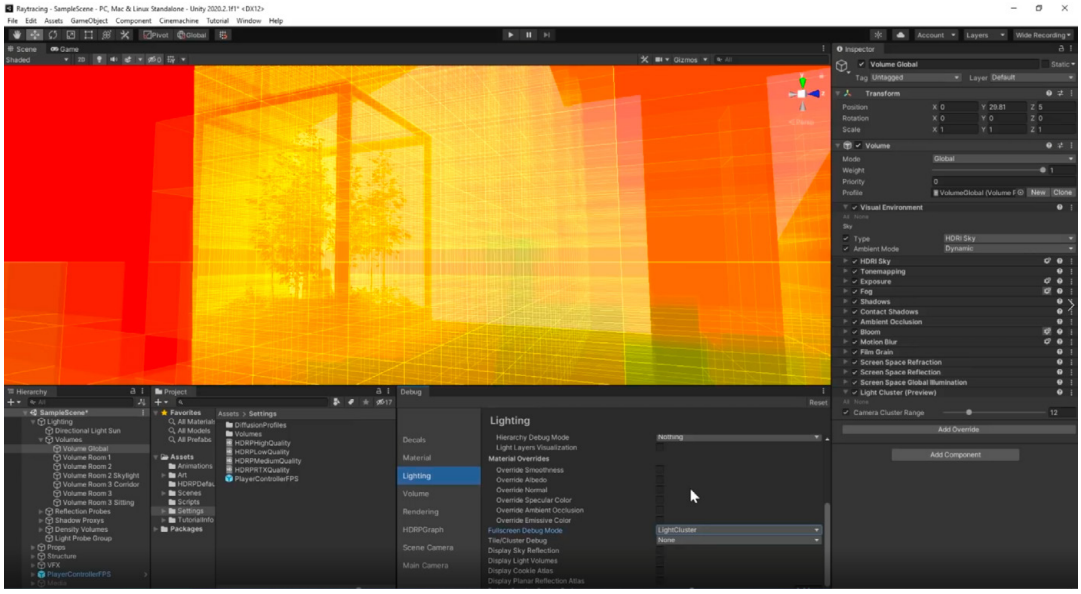
Screen space ambient occlusion (left image) vs ray traced ambient occlusion (right image)

— **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.

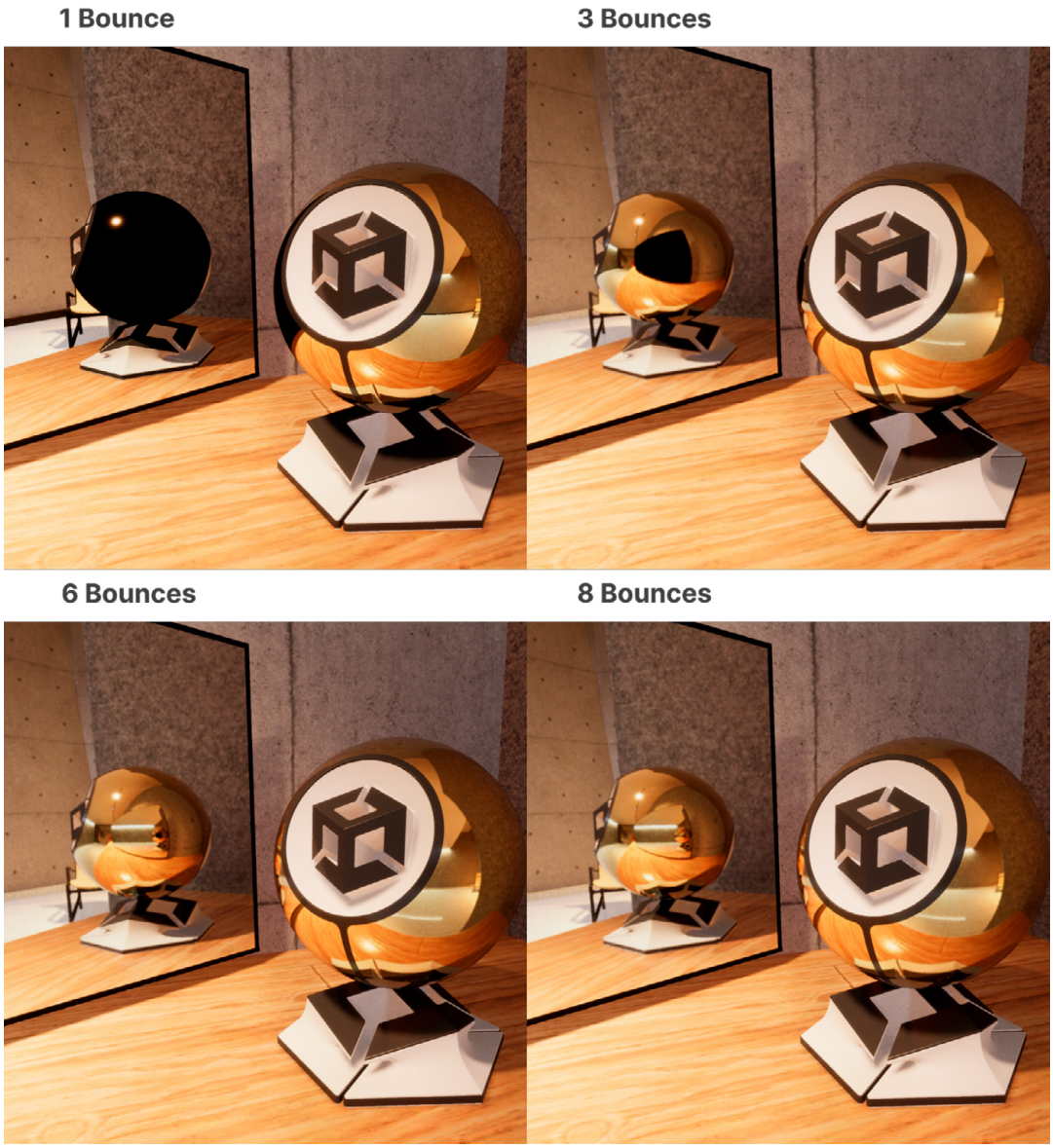
Use the Quality setting for complex interior environments that benefit from multiple bounces and samples. Performance mode (limited to one sample and one bounce) works well for exteriors, where the lighting mostly comes from the primary directional light.

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

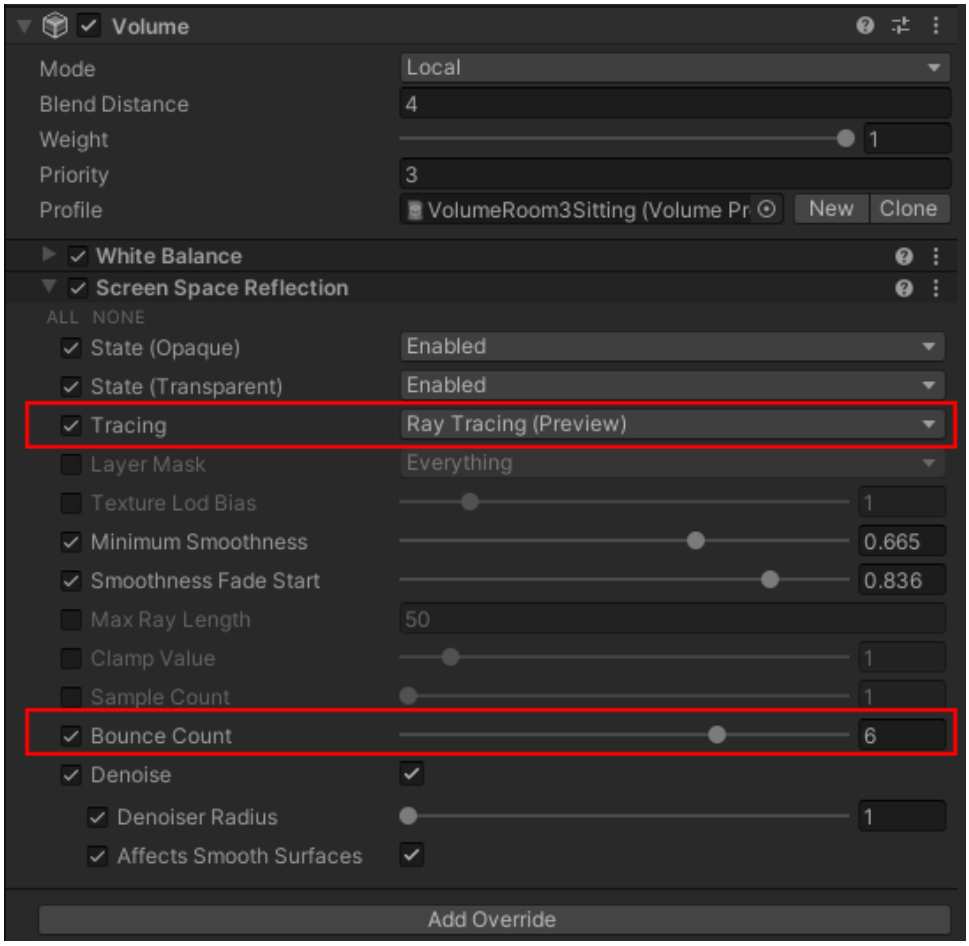
— **Ray traced reflections:** With ray traced reflections, you can achieve higher-quality reflections than using reflection probes or screen space reflections. Offscreen meshes appear correctly in the resulting reflections.

  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 traced reflections can include offscreen objects.



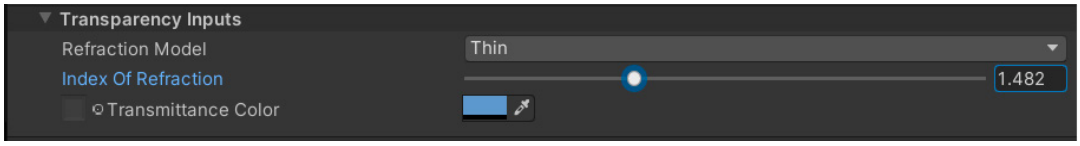
Ray tracing can improve screen space reflections.

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

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.

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.

Inline ray tracing support (PC and consoles)

The Unity Ray Tracing API has been extended to provide inline ray tracing support across all shader stages, when targeting Windows and console platforms.

Ray tracing pipeline

HDRP's various ray tracing effects, such as shadows and reflections, are all implemented via the Unity ray tracing API and using the traditional "ray tracing pipeline".

The ray tracing pipeline works by creating a ray tracing acceleration structure, which represents the scene's geometry and helps the GPU determine ray intersections efficiently.

When a ray hits an object, a "hit shader" calculates shading. If the ray misses, a "miss shader" defines the background or environment. Shader tables manage these interactions dynamically, binding the correct shaders for each case.

This system is well-suited for complex scenes but adds processing overhead and isn't ideal for general-purpose compute shaders. For simpler effects, using dynamic shader tables can introduce unnecessary complexity.

Inline ray tracing

DXR 1.1 introduces inline ray tracing, which allows hardware-accelerated ray queries inside compute and rasterization shaders without relying on the traditional ray tracing pipeline.

This provides an alternative method of performing hardware accelerated ray tracing, and unlocks new and exciting possibilities for shader and custom pipeline effect authoring.

In Unity 6, this new functionality is now fully supported by the ray tracing API across compute and rasterization shader stages, when targeting DXR1.1 capable Windows platforms and consoles.

Please check DirectX specification for DXR 1.1 and RayQuery object for more information on using ray queries in shaders. And check the official Discussions post for more details on inline ray tracing support.

Inline ray tracing creates these custom soft shadows.

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 tips help you balance visual quality with performance:

— **Dynamic resolution:** Because ray tracing scales with the number of pixels, rendering at a lower resolution saves on performance. Modern upscalers (e.g., NVIDIA DLSS) allow rendering at a reduced resolution, then intelligently upscale to full resolution with minimal visual fidelity loss.

  For example, you can activate DLSS and use a forced screen percentage of 75%. This improves frame times, especially in CPU-bound scenarios where reducing GPU load helps overall performance.

- **Pick your effects**: Not all ray traced effects need to be activated all the time. You can be selective where to deploy them in your game application.

  In a racing game, for example, you might add ray traced reflections to improve the rendering of the players' cars. In an open-world adventure game, ray traced shadows and global illumination could help make the environments more immersive.

- **Use ray tracing only when necessary:** For multiple effects like ray traced reflections and global illumination, you can try Mixed tracing mode first. This uses fast ray marching first and only falls back on ray tracing when necessary.

  To further reduce expensive long-ray calculations, use alternative techniques like Probe Volumes, Sky Volume overrides, or reflection probes, which provide efficient lighting and reflections without the full cost of ray tracing.

- **Optimize the evaluation**:

  - On some effects, use Layer Masks to prevent ray traced effects from evaluating unnecessary objects that don't need them.
  - Adjust quality settings. Customize ray length, sample count, and resolution to balance performance and noise. Remember that longer rays require more samples to reduce noise, while lowering sample count reduces GPU cost but increases noise.
  - For reflections, reduce the number of bounces if objects don't need to be reflected inside of other reflections. Tweak the smoothness parameters to apply ray tracing only where needed.
  - For ray-traced effects that use textures, apply a Texture LOD Bias to fetch lower-resolution MIP levels. This reduces GPU memory usage and lowers the cost of texture fetches during material evaluation. This can improve overall performance without significantly impacting visual quality.

- **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 Rendering Debugger (**Window** → **Render Pipeline Debug**) when in Play mode. This provides a set of markers to track the cost of every ray tracing effect.

The **CPU timings RT** measures the time (in milliseconds) the CPU takes to process ray tracing tasks, including attribute binding and C# execution. Meanwhile, the **GPU timings RT** tracks the GPU's execution time per ray-traced effect.

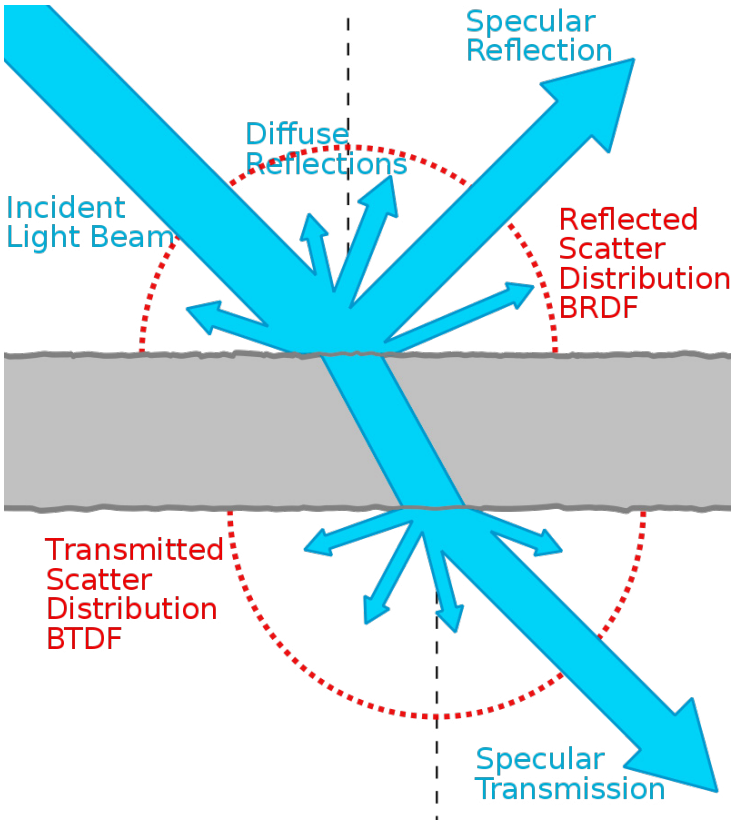
Path tracing

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.

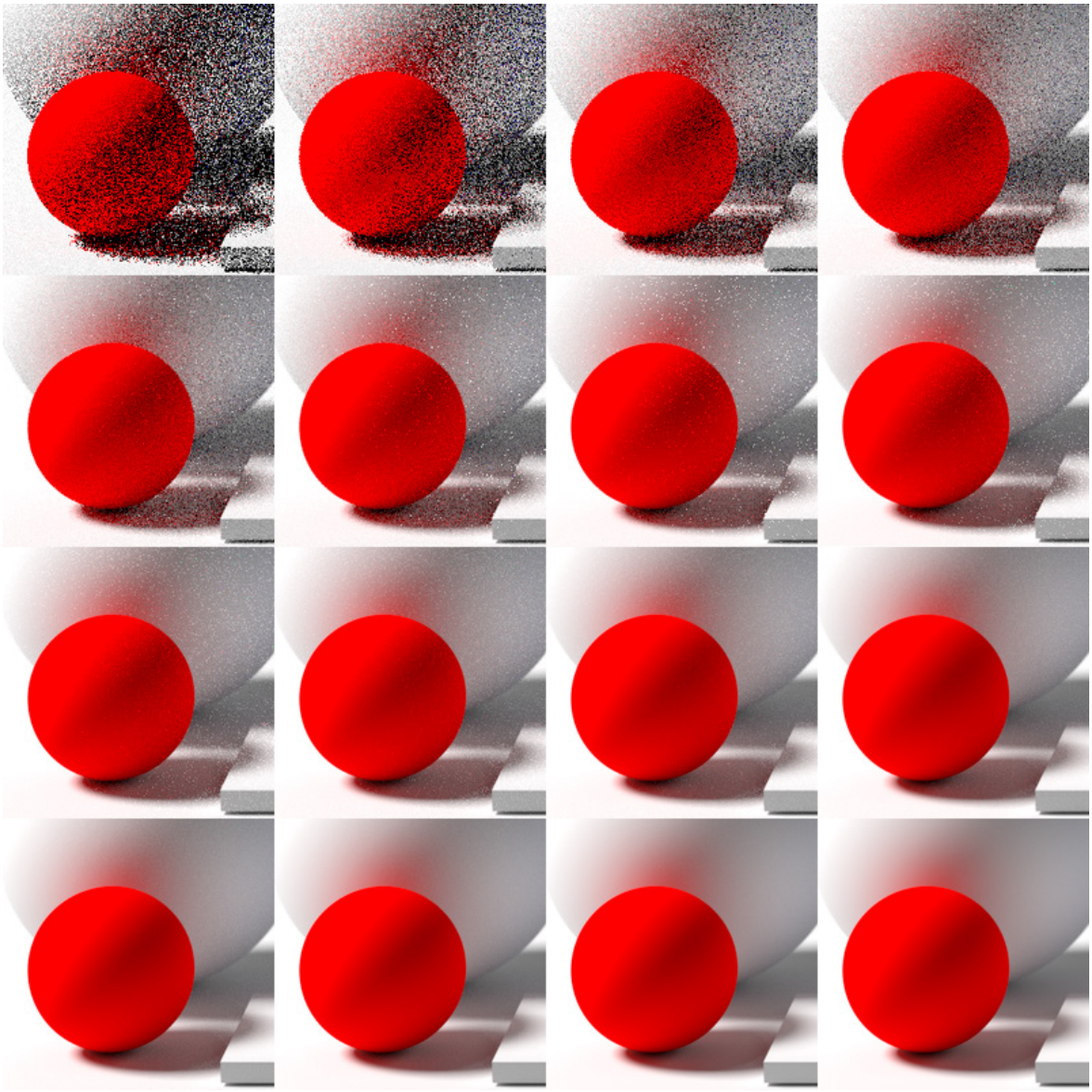
At each bounce, the renderer gathers light information, taking into account both direct and indirect illumination. Color information from all these bounces is accumulated to determine the final color of a pixel.



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)

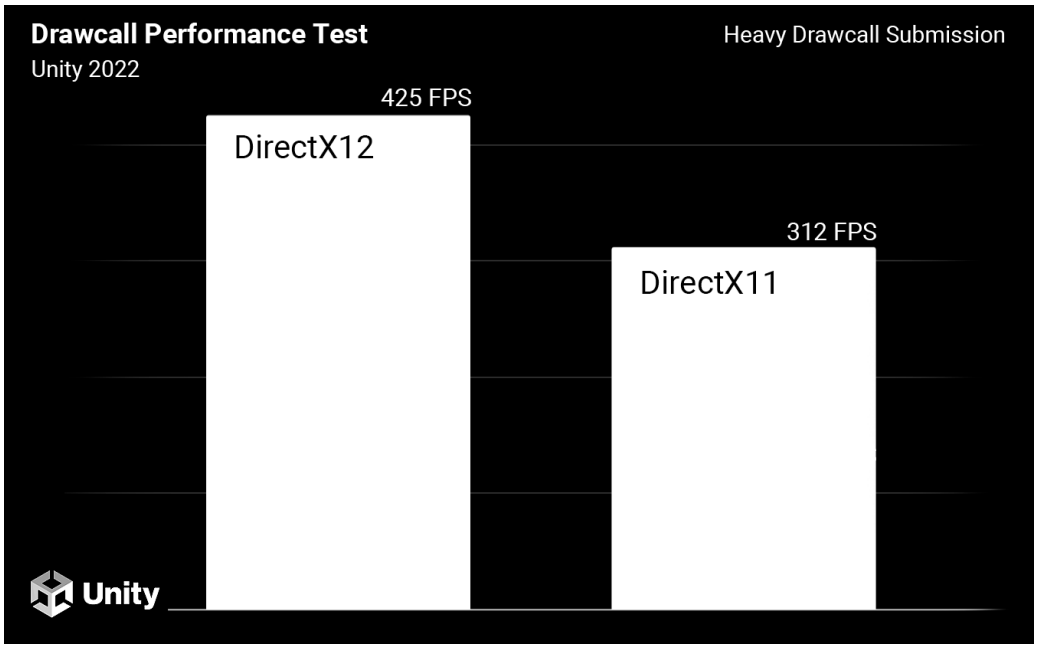
— HDRP also adds the **Use AOVs** (Arbitrary Output Variables) setting to material shaders that use path tracing to support the new path tracing denoisers. When you enable this setting, HDRP puts albedo and normal values into AOVs and can improve your path tracing results.

DirectX 12

DirectX 12 (DX12) is now the default graphics API on Windows with DirectX 11 (DX11) fallback if needed. The DirectX 12 (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. Users can activate it using the command line argument **`-force-gfx-jobs native`**.



Performance test comparing DirectX 12 and DirectX 11 in Unity 2022

— 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.

Samples in Unity 6

For a great way to see ray tracing in action, check out the new Transparency samples from the Package Manager in Unity 6. This sample scene showcases ray-traced transparency, reflections, and lighting interactions. Enable ray tracing effects to render glass, water, and other transparent materials more realistically.

The Transparent samples showcase ray tracing effects.

Environment lighting

In the real world, light bounces between surfaces, scattering through the atmosphere and reflecting off the ground. This indirect illumination plays a role in defining the overall brightness and color of a scene.

In HDRP, environment lighting simulates this effect, ensuring that objects receive ambient light from the sky, even when direct light sources are not present. This is the result of random photons bouncing between the atmosphere and earth and ultimately arriving at the observer.

In Unity, a sky is a type of background that a camera draws before it renders a frame. This type of background provides a sense of depth and makes the environment seem much larger than it actually is. Unity can also use a sky to generate realistic ambient lighting in your scene.

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

Default Lighting Data Asset

In Unity 6, the previous SkyManager system has been replaced with a new **default Lighting Data Asset**, now embedded in the Editor.

When you add a new scene, Unity assigns this asset automatically. It provides default environment lighting using a hidden ambient probe and reflection probe, which capture environment lighting from Unity's built-in Default Skybox. This new default asset doesn't appear in the Project window.

Note that the hidden probes contain fixed, precomputed data that does not update. If you simply change the Skybox Material in the Lighting window, these probes will not capture the new skybox.

To stop using the default Lighting Data Asset, select **Generate Lighting** or **Clear Baked Data** in the Lighting window. Unity then creates and uses a new default Lighting Data Asset and a new baked reflection probe (the ambient reflection probe). These assets appear in the Project window, and update when you select Generate Lighting.

Previously, while the SkyManager could ensure that environment lighting affects your scene by default, it could also sometimes cause inconsistencies between the Editor and the built Player. The new default Lighting Data Asset ensures a more predictable workflow.

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

Visual Environment

In most cases, you'll replace the default sky with a custom setup that better suits your scene's lighting and artistic goals.

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

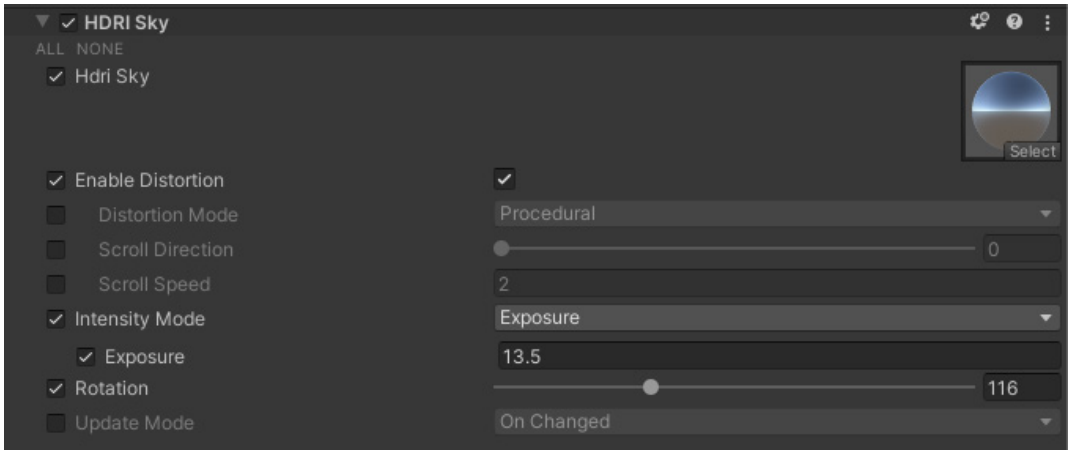
Use **Ambient Mode: Dynamic** to set the sky lighting to the current override that appears in the Visual Environment's **Sky > Type**. Otherwise, **Ambient Mode: Static** defaults to the sky setup in the **Lighting** window's **Environment** tab.

HDRP includes three different techniques for generating skies. Set the **Type** to either **HDRI 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.

HDRI Sky

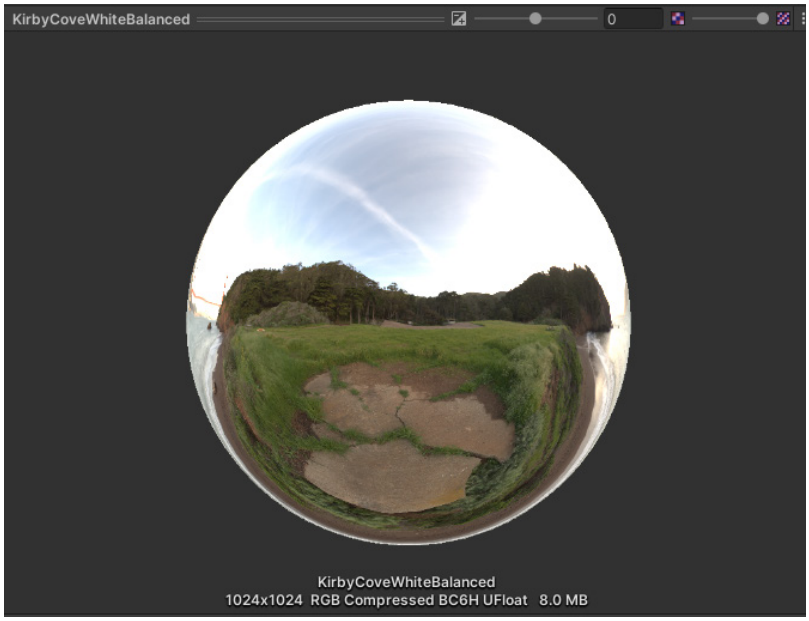
The HDRI Sky override allows you to represent the sky with a cubemap made from high dynamic range photographs. You can find numerous free and low-cost sources of HDRIs online. A good starting point is the Unity HDRI Pack in the Asset Store.
If you're adventurous, we also have a guide to shooting your own HDRIs.



The HDRI Sky asset

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

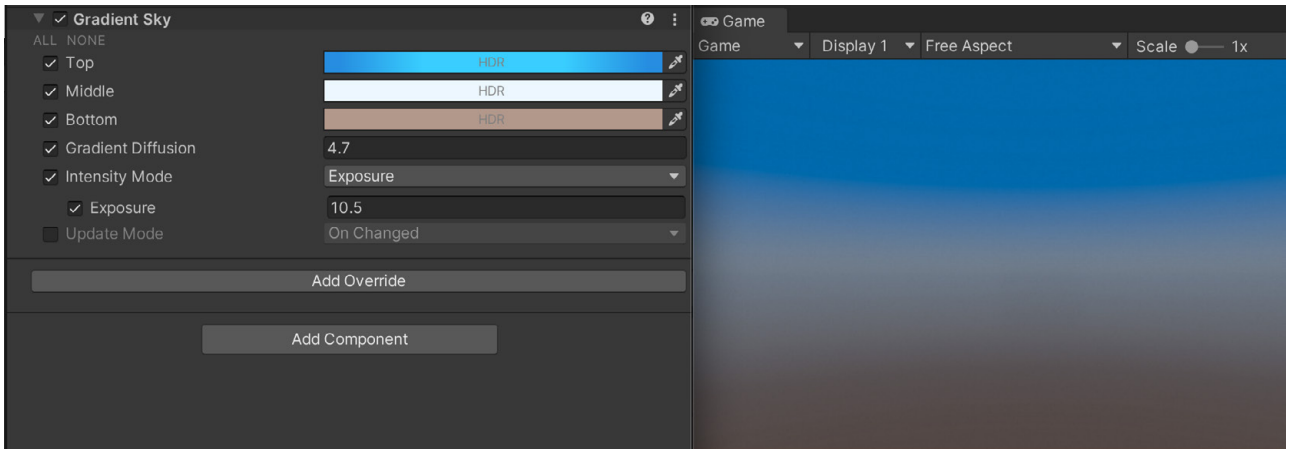
Animating HDRI skies

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 override 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.

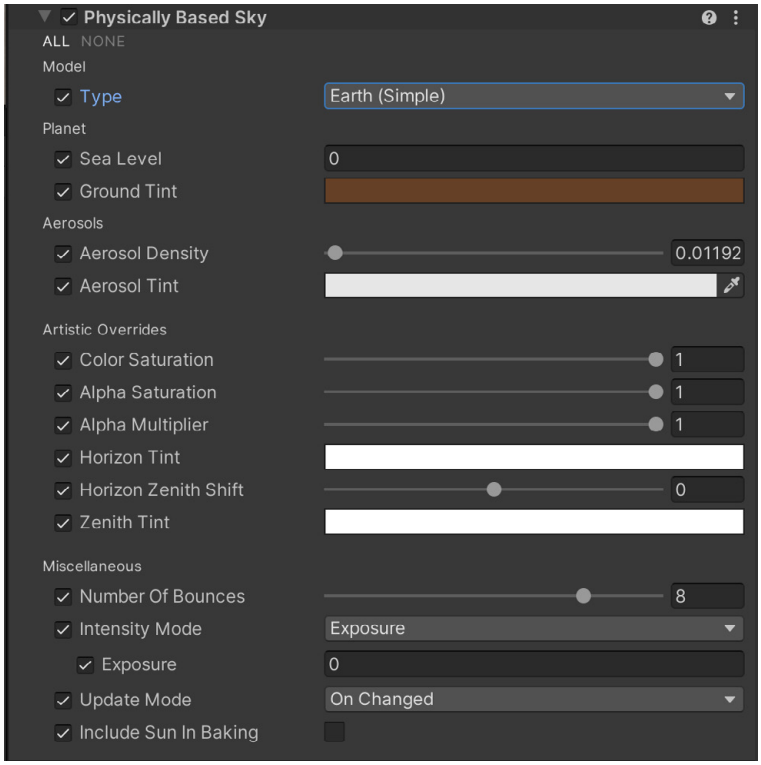


The Top, Middle, and Bottom colors blend into a Gradient Sky.

Physically Based Sky

For something significantly more realistic than a gradient, you can use the **Physically Based Sky** (PBR) override.

The Physically Based Sky override in HDRP simulates a two-part atmosphere composed of air and aerosol particles. This can be adjusted in the Physically Based Sky override using the Volume system.



The Physically Based Sky override

This override procedurally generates a sky that incorporates phenomena such as Mie and Rayleigh scattering. These simulate light dispersing through the atmosphere, recreating the coloration of the natural sky. A PBR sky requires a directional light for accurate simulation. Your Physically Based Sky override can also serve as an environment lighting source.

A PBR sky from *Time Ghost*

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.

Unity 6 updates for PBR skies

PBR skies are now more efficient, with reduced memory usage and improved performance. Multi-scattering no longer requires certain precomputation steps, improving light scattering when the sun is below the horizon.

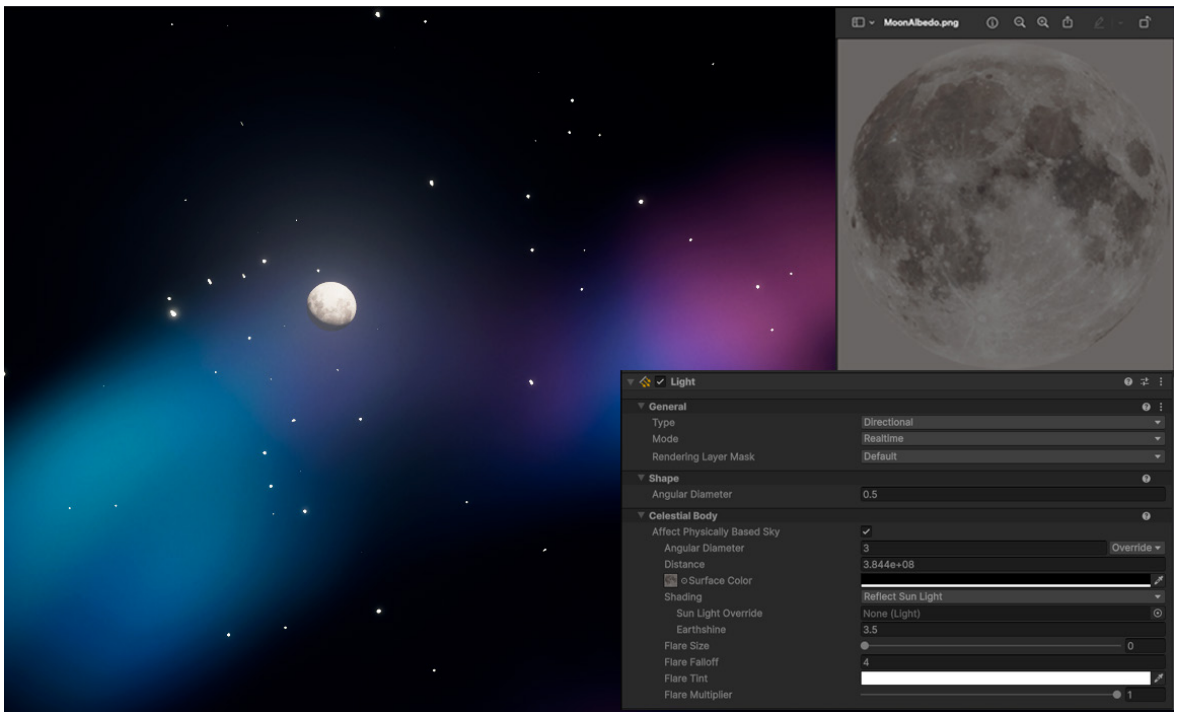
The PBR sky's atmosphere model now includes an ozone layer. This is noticeable when the sun is close to the horizon (e.g. dusk), subtly tinting sunlight with a purplish hue as it passes through the thicker atmosphere at low angles.

Ozone tints the sky at sunset.

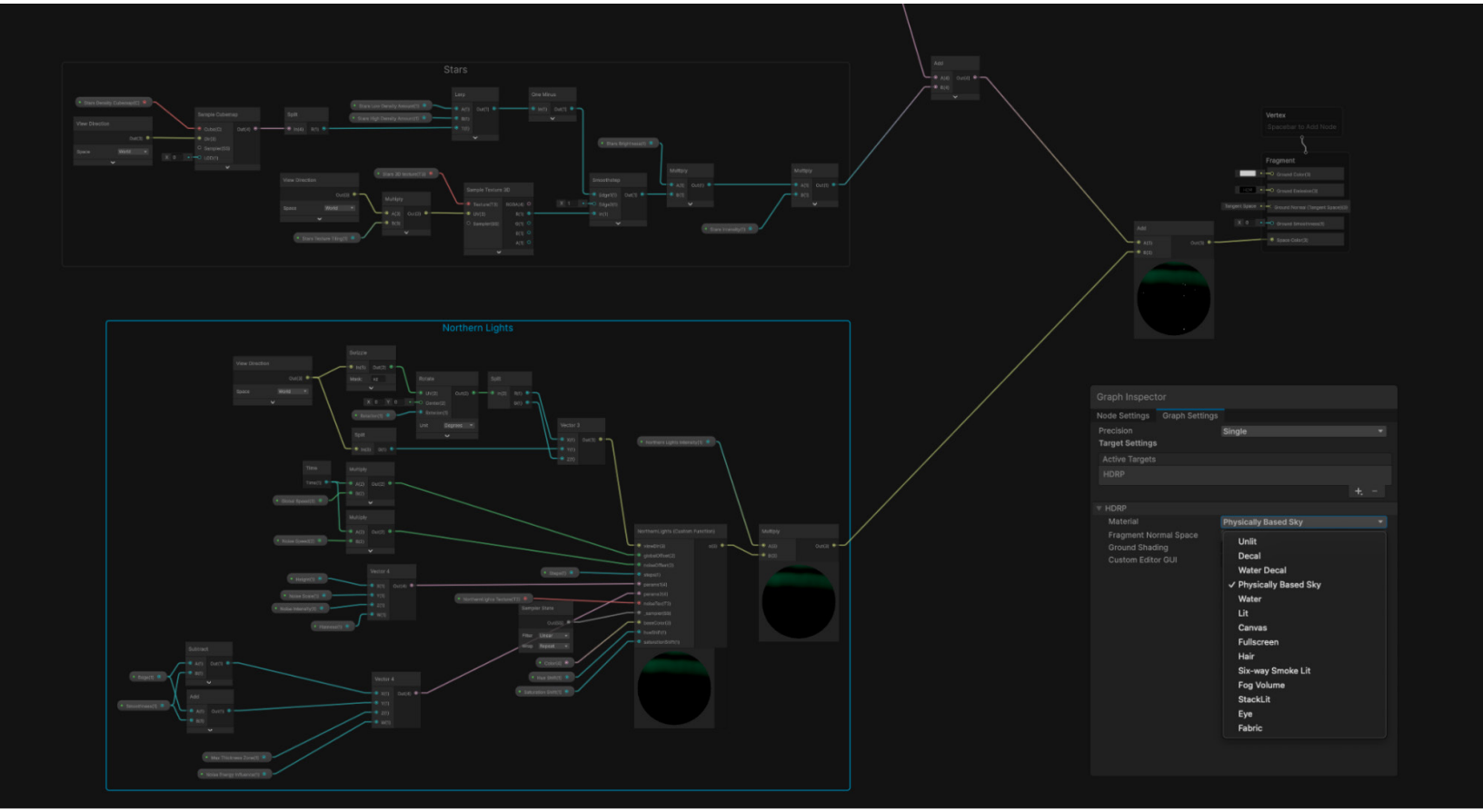
Aerial perspective can now simulate light absorption by particles in the atmosphere, naturally fading out objects in the distance like mountains or clouds.

Atmospheric scattering renders clouds more realistically.

Unity 6 includes new features to simplify night sky rendering. A **PBRSky material type** in Shader Graph enables effects like procedural stars, and a new option allows celestial bodies to be marked as moons, ensuring they receive lighting from the main directional light.



A celestial body can represent the moon.



Shader Graph enables effects like procedural stars or the northern lights.

Atmospheric scattering

To optimize performance, atmospheric scattering is applied to the sky, simulating how blue light scatters across distant objects and clouds while avoiding the cost of volumetric fog at high altitudes. The system models three key interactions:

— **Rayleigh scattering** (air molecules) creates the sky's blue color.

— **Mie scattering** (aerosols) introduces haze, fog, and pollution effects.

— **Ozone absorption** alters the sky's color gradient without contributing to scattering.

Sky rendering now simulates atmospheric scattering for distant objects and clouds, improving how they interact with ambient light. This is useful for time-of-day transitions, preventing unnaturally dark clouds at dusk when shifting from evening to night.

The result is smoother lighting transitions and more natural-looking skies without requiring large values for fog distance. In addition, aerosol scattering has been refined, ensuring more realistic atmospheric depth and consistency.

Rendering Space

The **Rendering Space** setting in the Visual Environment determines how the planet and sky simulation behave:

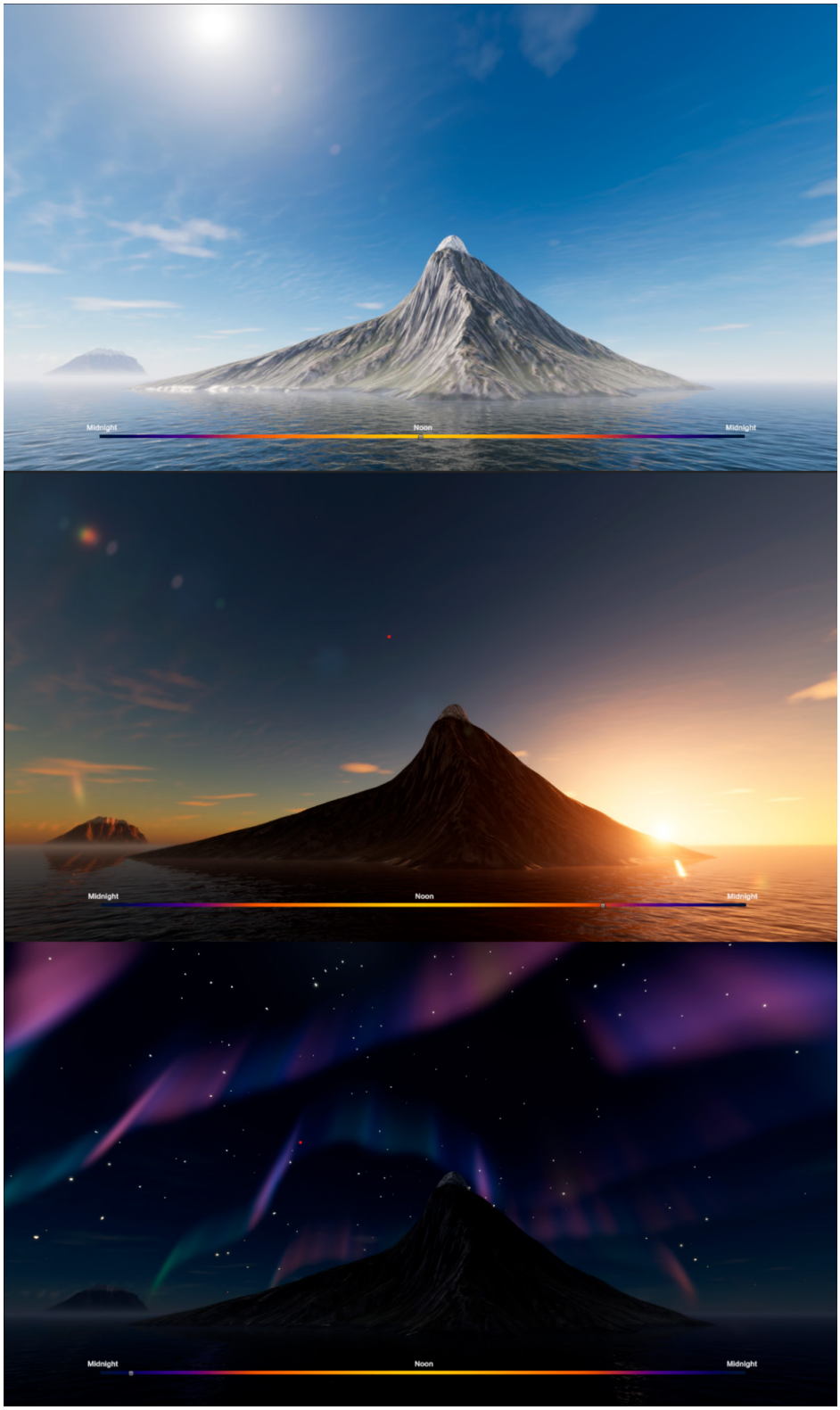
— **World mode**: The planet's surface remains fixed at the world origin, with its center derived from the Radius property. Atmospheric scattering is precomputed for all possible light and viewing angles; then it's stored in 3D textures, which Unity resamples at runtime. This mode allows the camera to move freely, even into outer space.

— **Camera mode**: The planet moves with the camera, keeping the surface always beneath it. This mode precomputes sky scattering for a specific set of lights rather than for the entire scene.

Note that the planet surface does not render in the depth buffer, so it won't occlude lens flares or behave correctly with motion blur.

Environment Sample

To see these features in action, import the **Environment Sample** from the **Package Manager**. This includes an island scene with skies and clouds using a time-of-day system, transitioning between midnight, noon, and dusk.



The Environment Sample shows a sky setup.

The Environment Sample uses the Terrain and Clouds from HDRP's worldbuilding tools – topics we'll explore in upcoming chapters.

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.

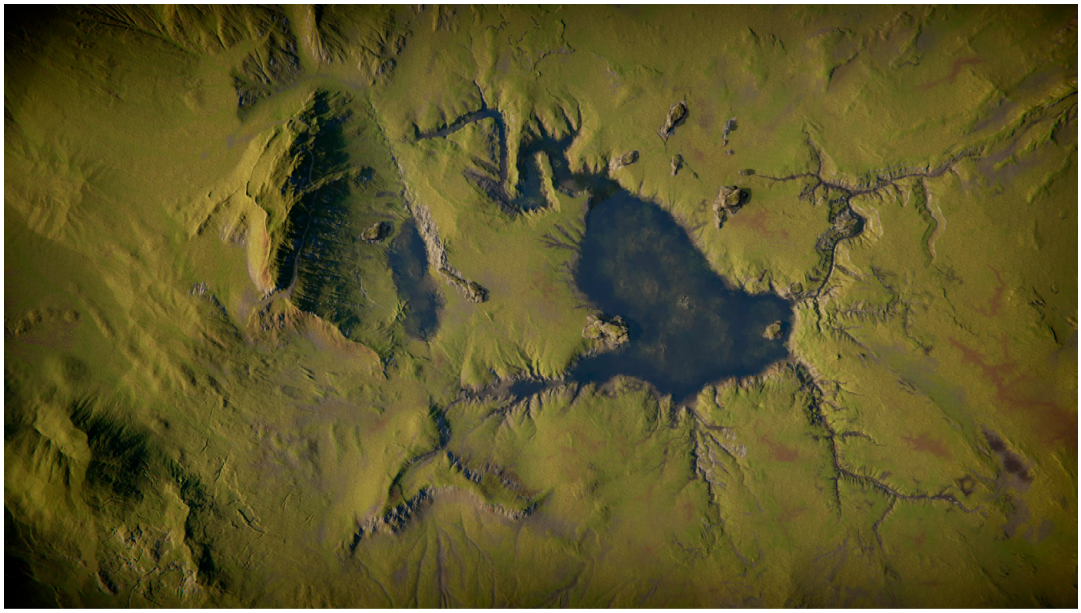
The Terrain system works with the Cloud Layer Volume component override.

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.

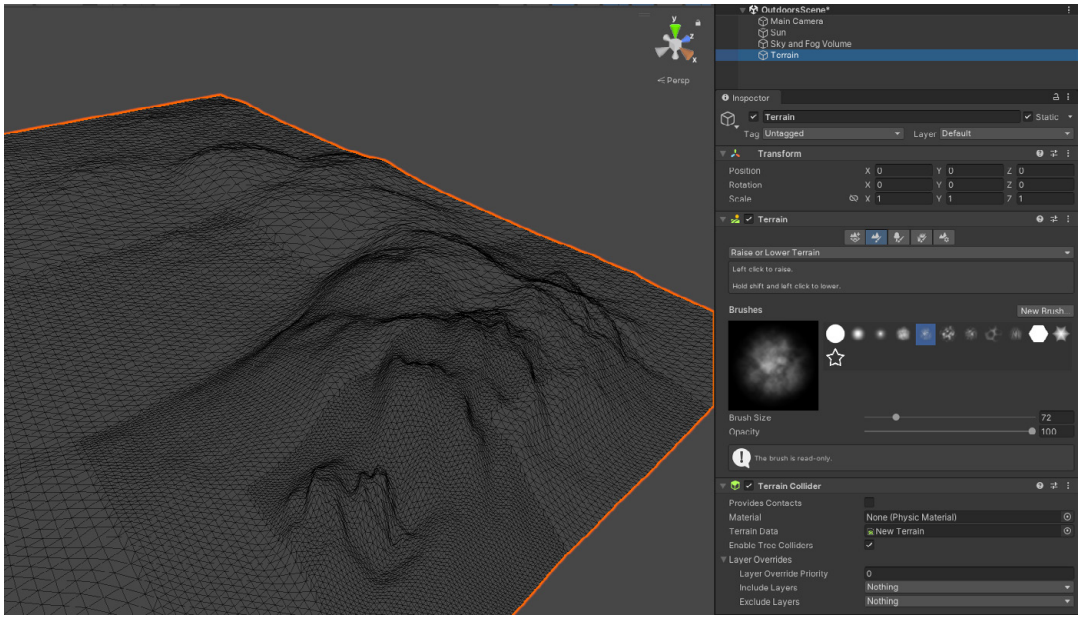


Paint on a heightmap to modify the Terrain.

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.

Unity 6.1 improves Terrain Inspector performance by deferring full-resolution brush texture creation until a brush is actively used. This reduces memory usage and makes the Terrain Inspector faster and more performant.

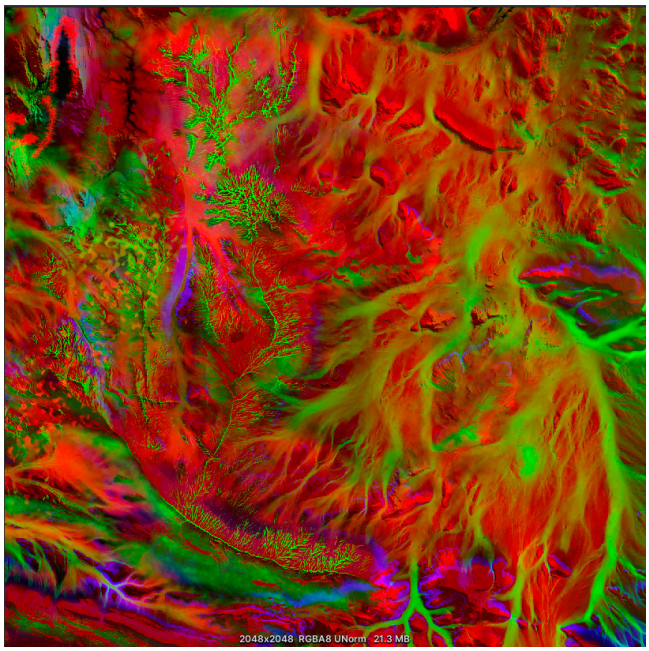


Use brushes to sculpt terrain.

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.

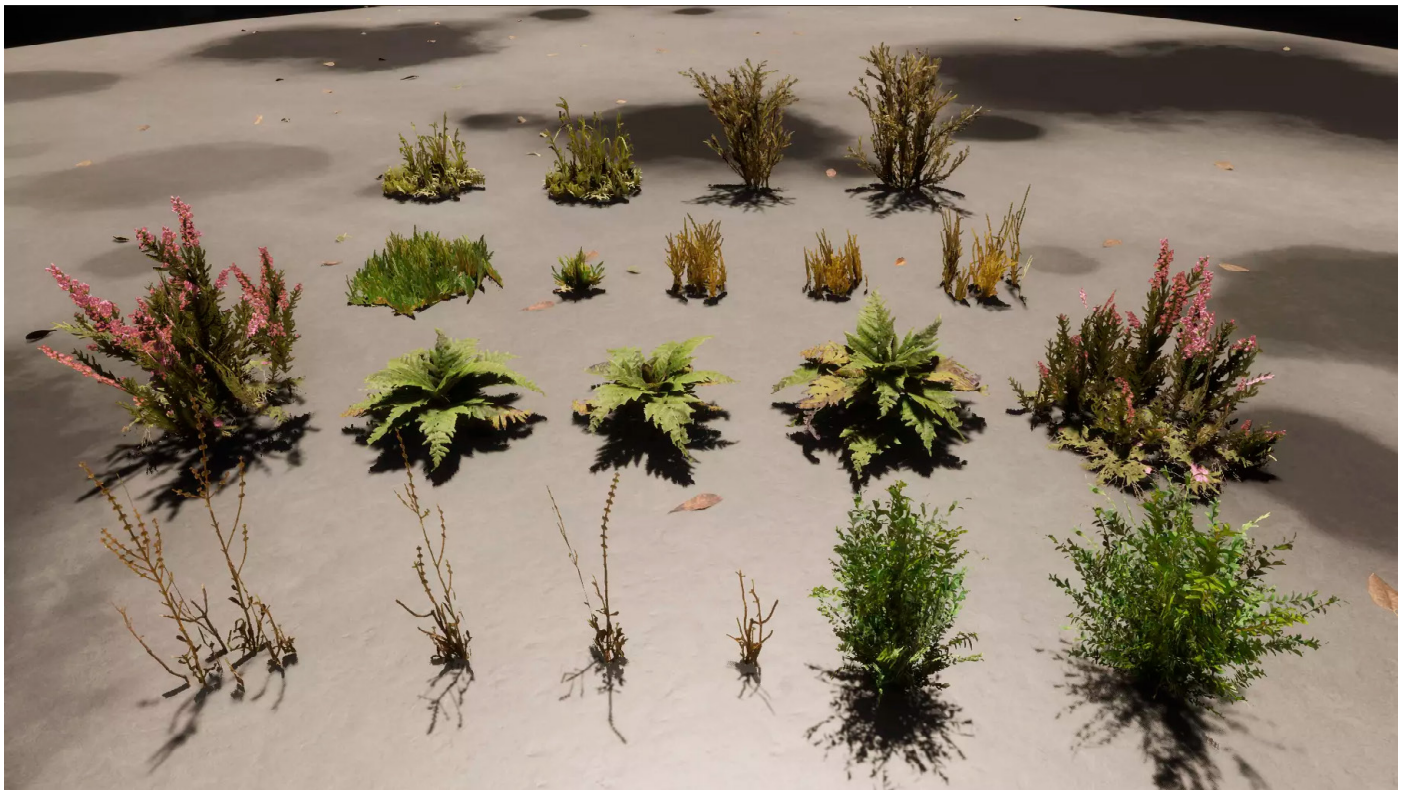
Then, use **detail maps** for finer details like grass, flowers, or small rocks. Paint these directly onto the terrain.



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 a Wind Zone component to make your trees and vegetation sway with the wind, adding life to the scene.

SpeedTree integration

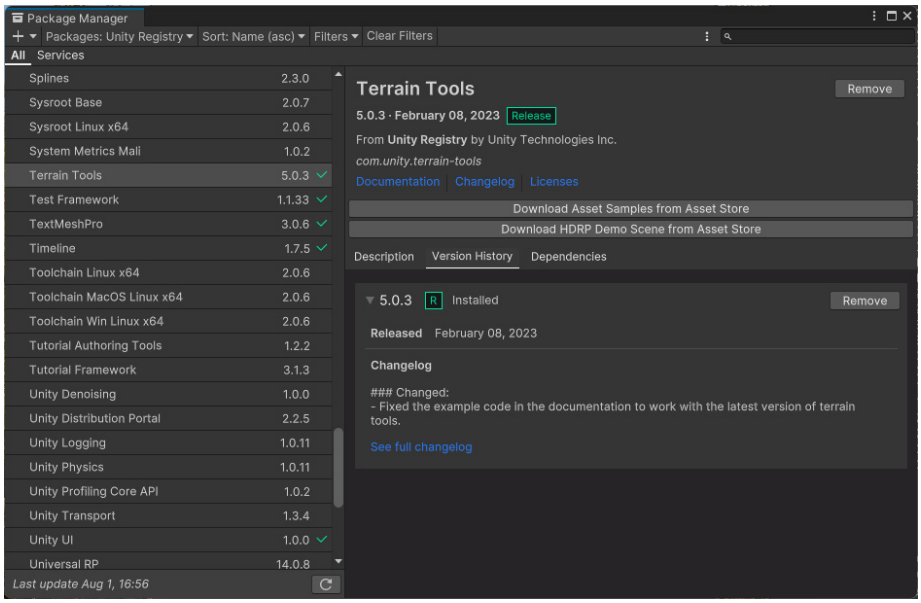
Unity also supports the direct import of SpeedTree models (.SPM or .ST files), treating them like standard GameObjects. Use the Terrain Editor to paint SpeedTree vegetation directly onto terrains. Painted trees can automatically receive colliders to interact with other objects at runtime.

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.



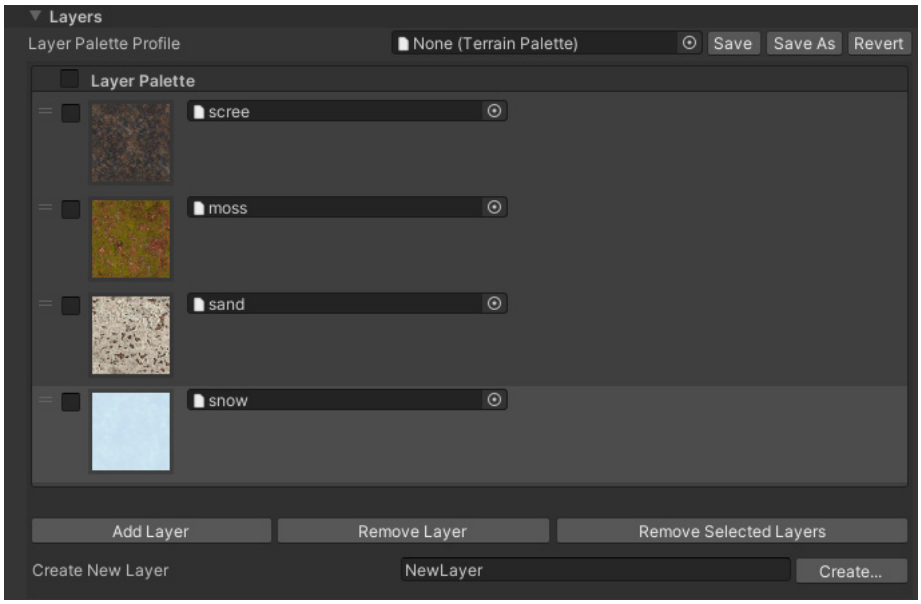
Add the Terrain Tools package for advanced features.

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

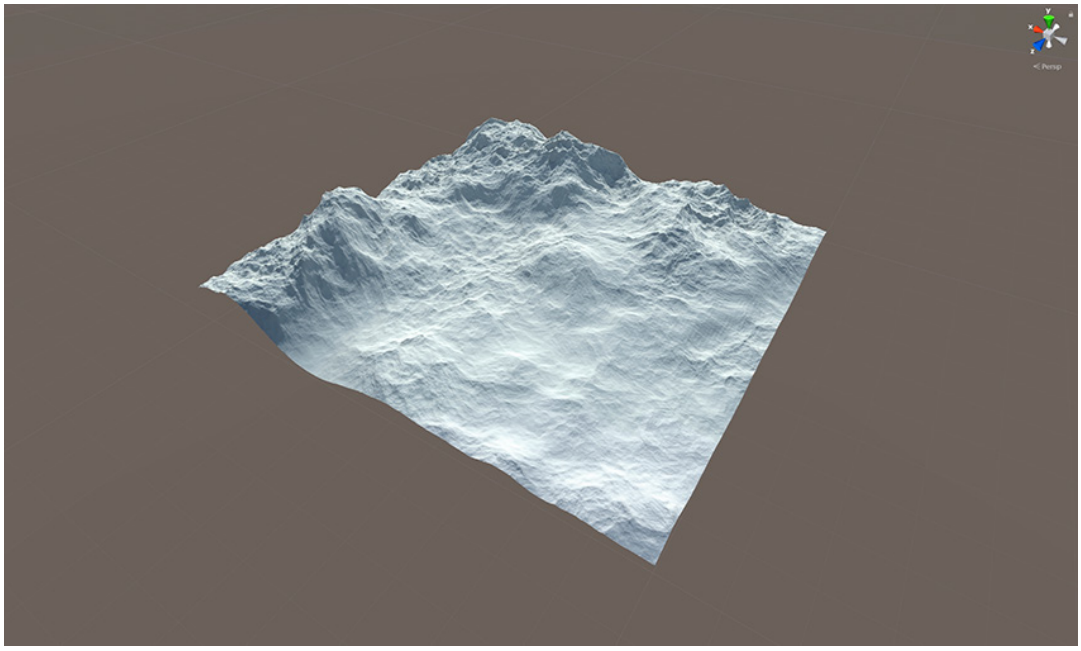
Sculpt tools alter the shape of the terrain with additive and subtractive methods:

— 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.

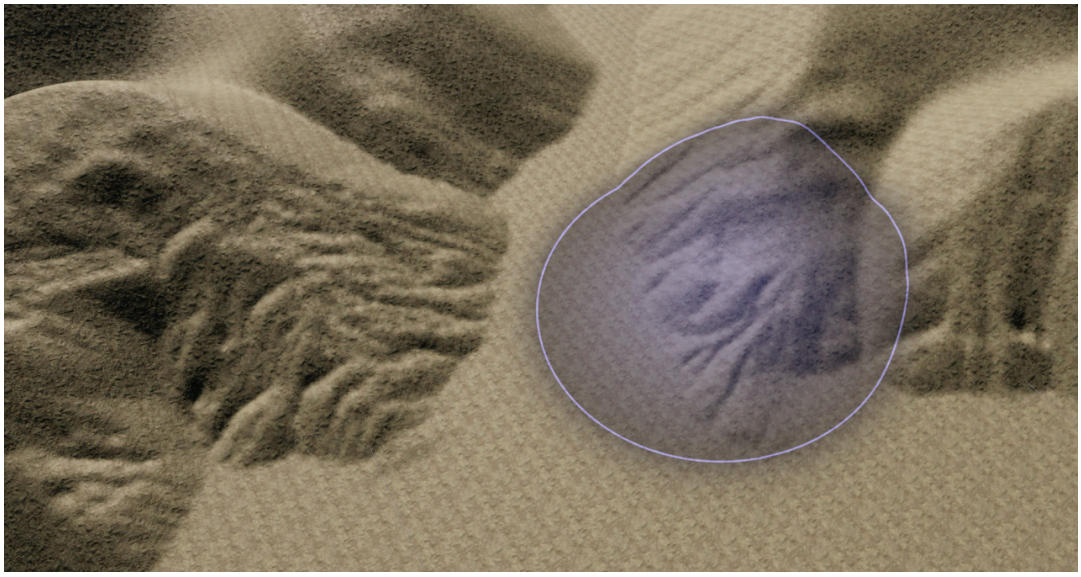
— Slope Flatten flattens the terrain while maintaining the average slope.

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.

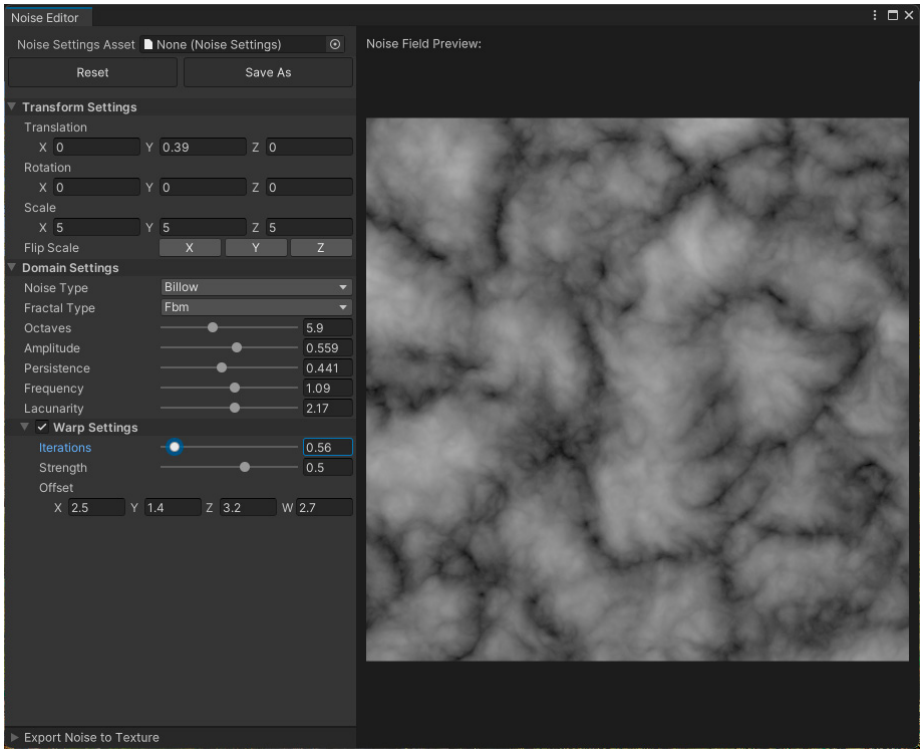
— Twist rotates terrain features around the center of the brush, along the path of the brush stroke.

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.

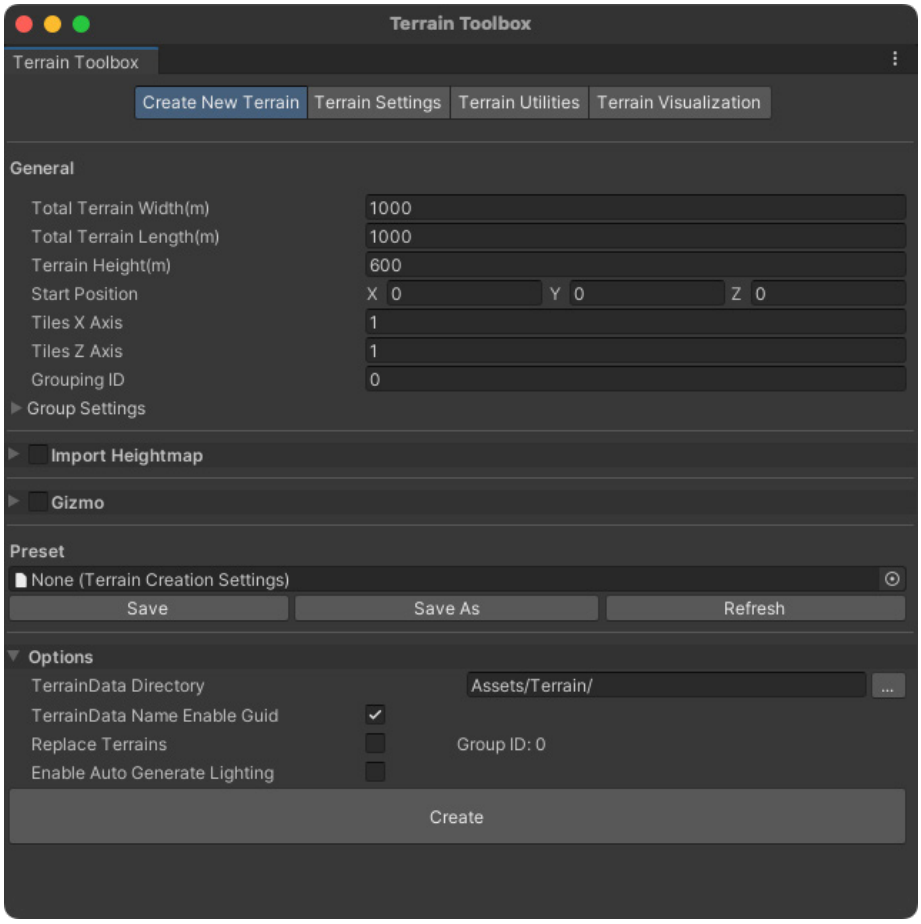


The Noise Editor

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

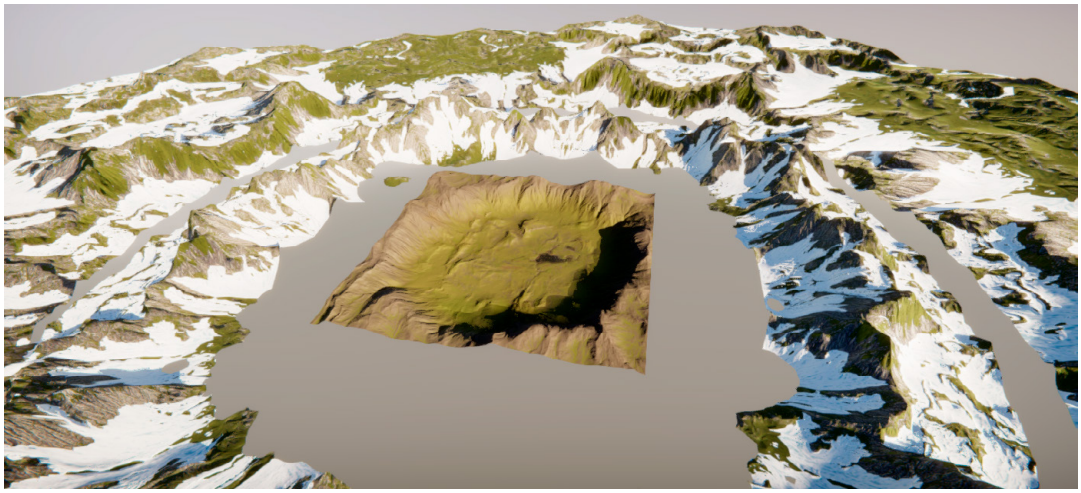
Specifically, the terrain is omitted in ray traced reflections. Instead, you'll need to use planar reflections when necessary (e.g., the surface of a lake). Also, ray traced ambient occlusion and global illumination do not affect the terrain.

HDRP Terrain Demo

This package is available on the Unity Asset Store and 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.

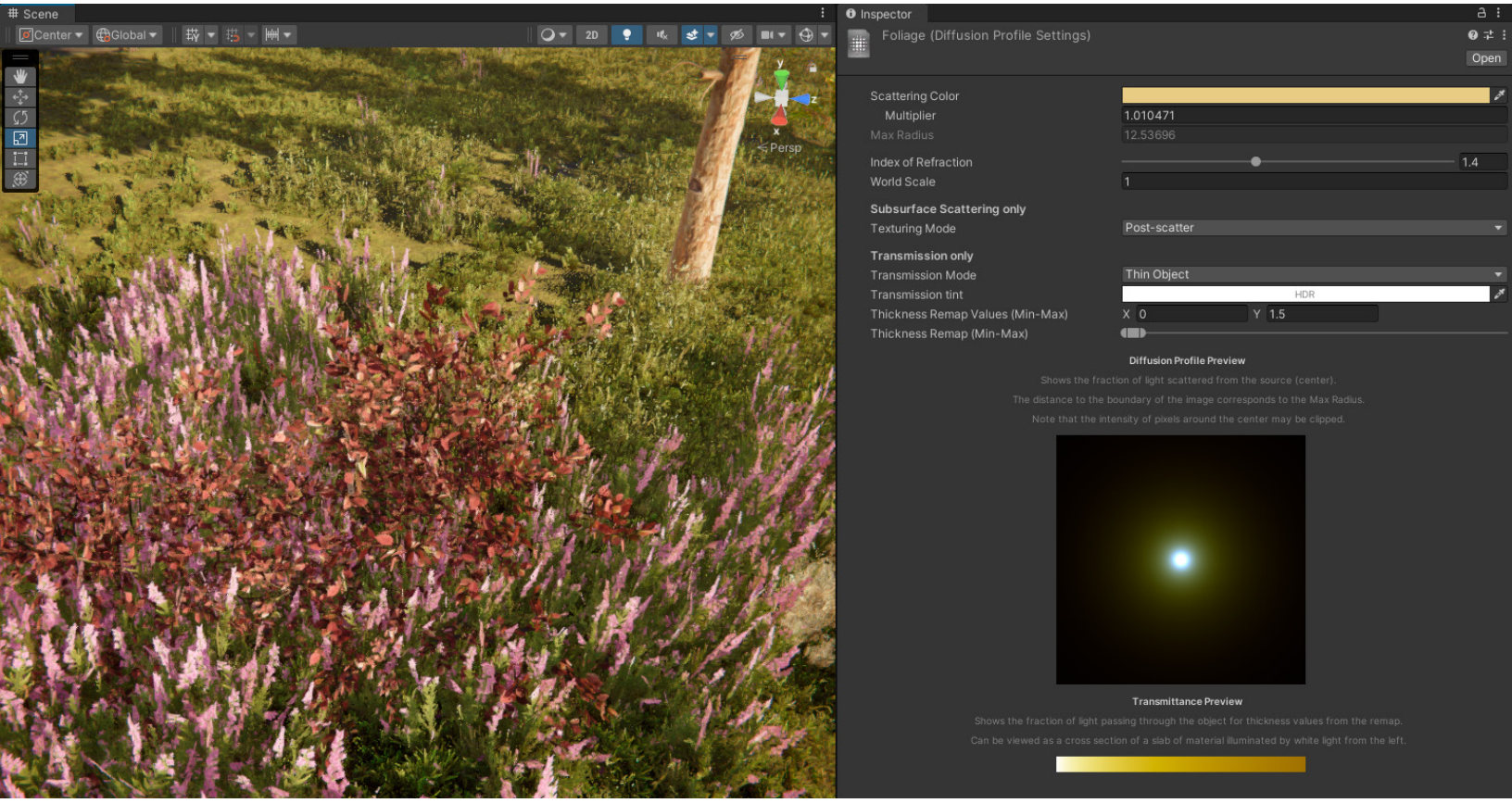
The HDRP Terrain Demo is available via the Package Manager.

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.

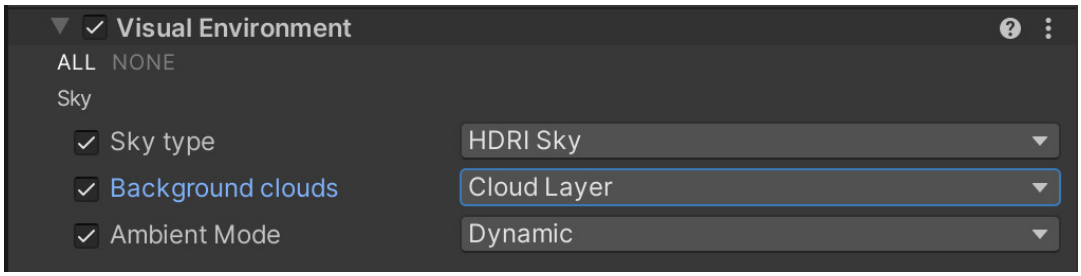
Clouds

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.

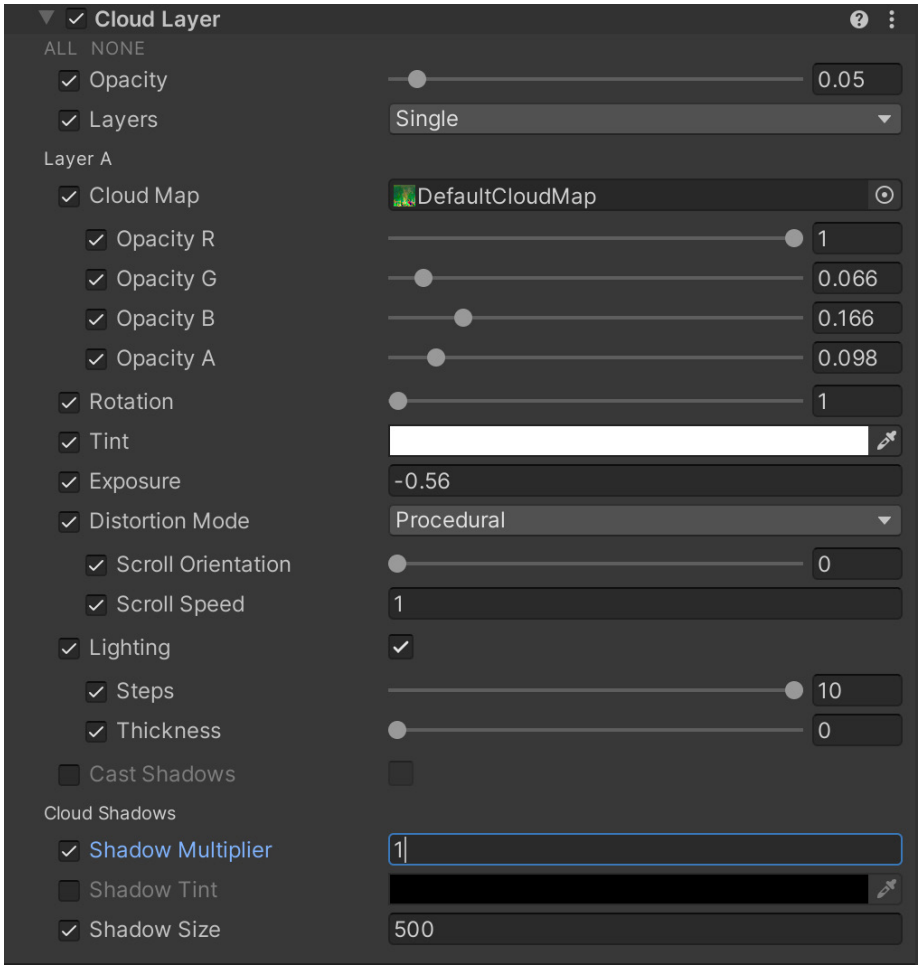
Cloud Layer

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.



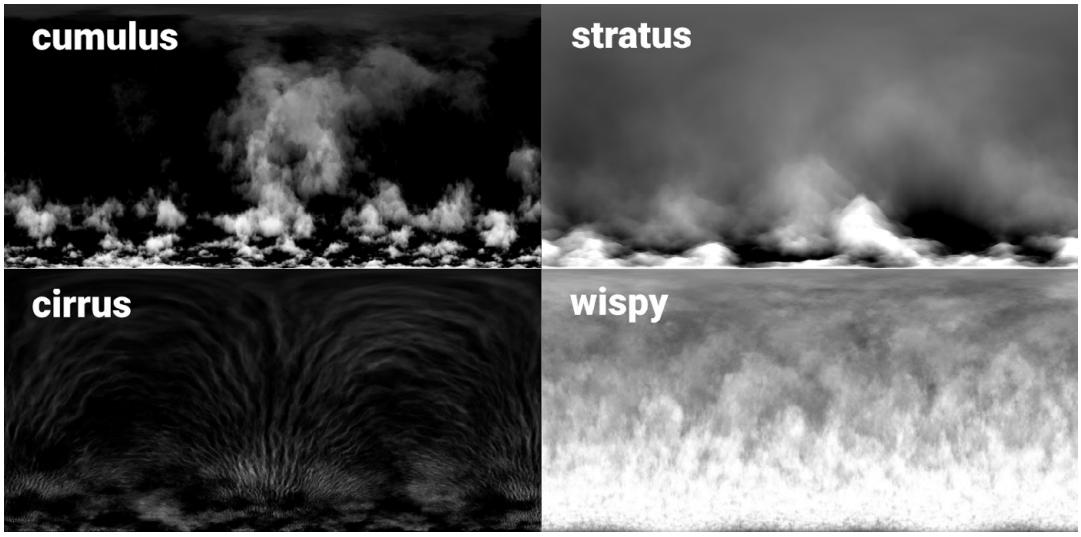
Enable the Cloud Layer override in the Visual Environment override.

In a local or global Volume, enable **Background clouds** in the Visual Environment override, 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 cloudscapes.



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 6 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

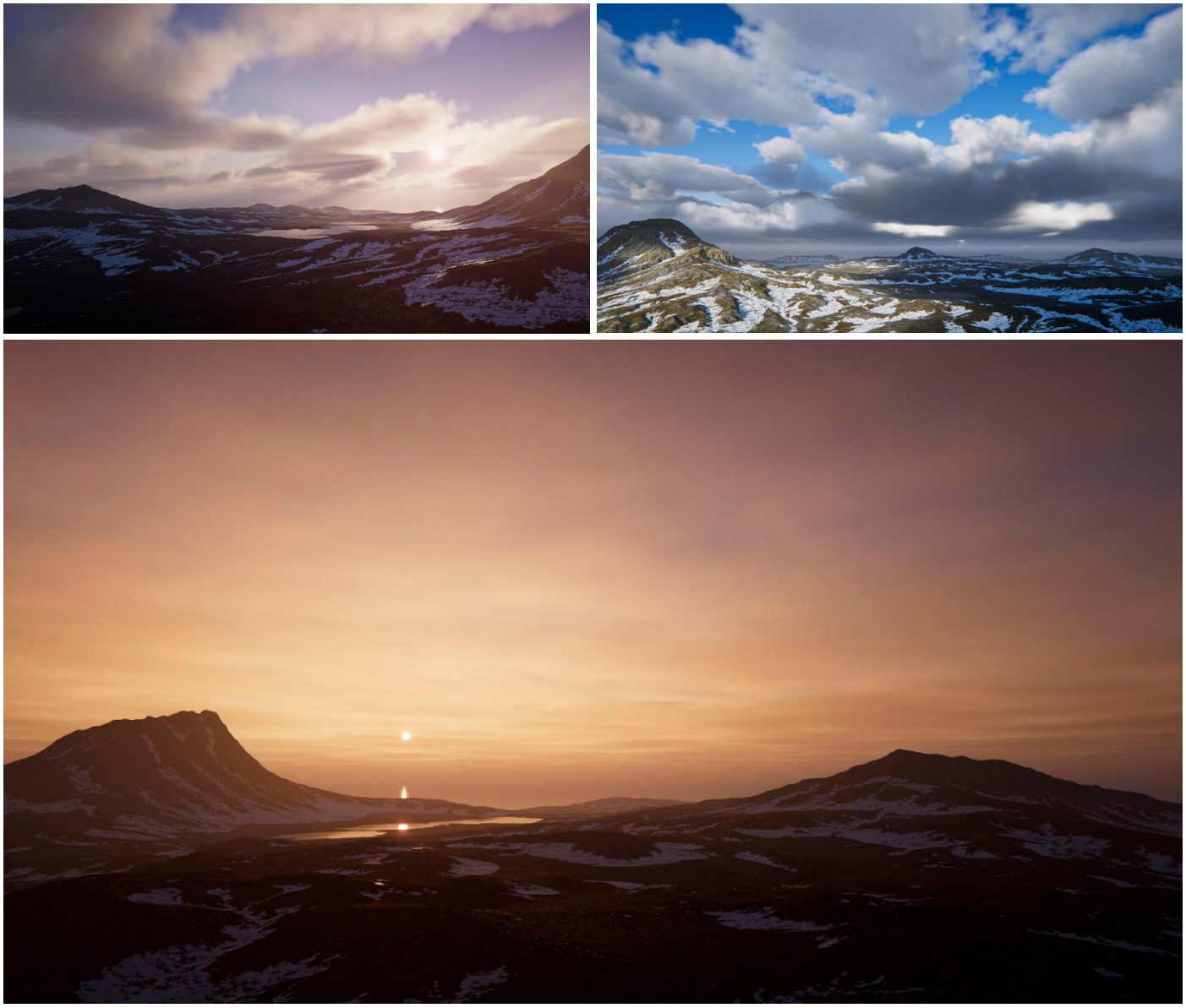
Volumetric clouds

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:

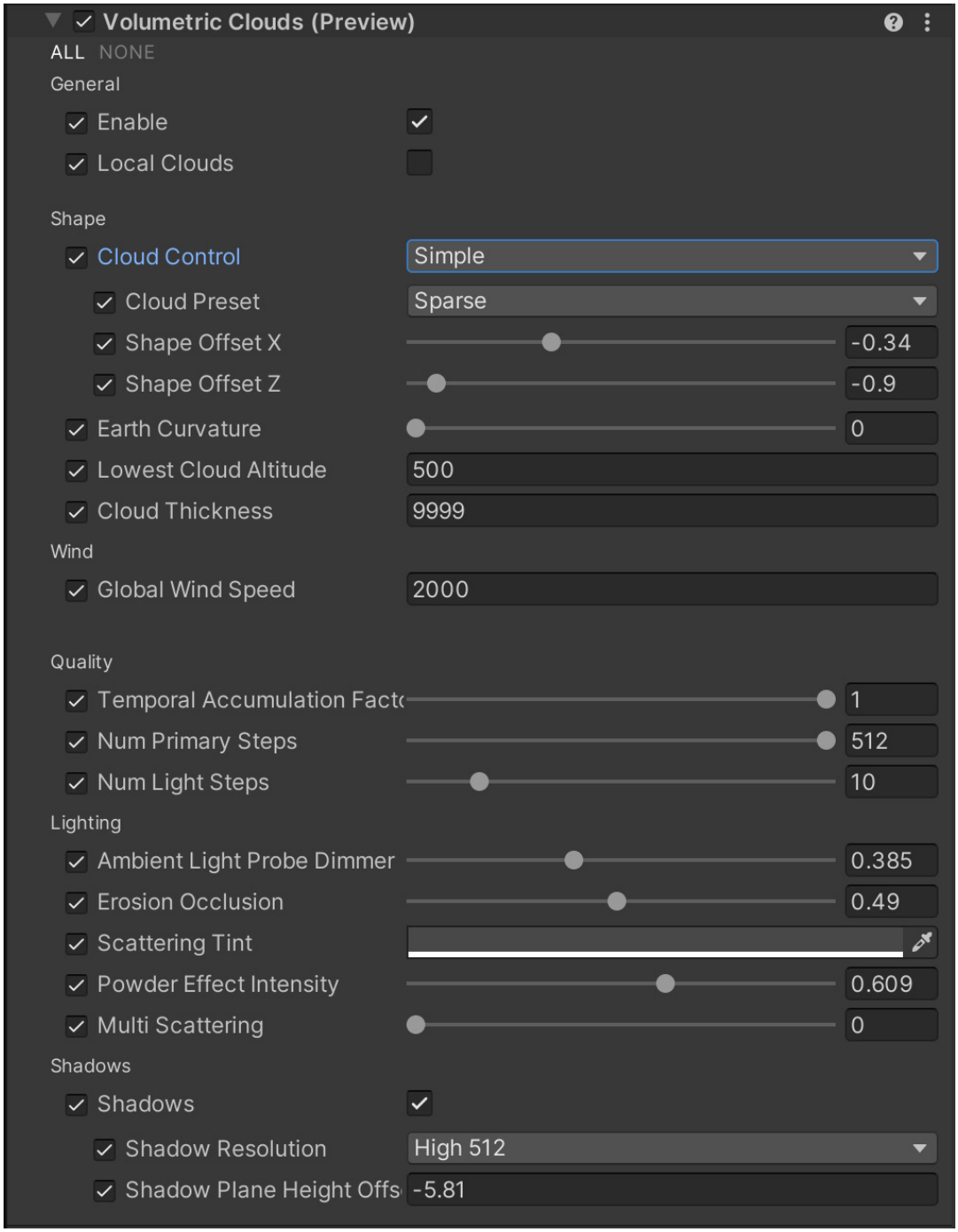
— In your HDRP Asset: Enable **Lighting > Volumetric Clouds > Volumetric Clouds**.

— 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

Unity 6 updates

Note these updates to Volumetric Clouds in Unity 6:

- Volumetric clouds are no longer clipped by the far plane and fully render regardless of camera distance.
- Planet settings are now managed as part of the **Visual Environment** override, eliminating the need for separate earth controls. This replaces the need for the previous Local Clouds option.
- Improved atmospheric scattering enhances lighting on distant clouds. This reduces the need for high fog distances that previously impacted performance.
- Volumetric clouds render with improved self-shadowing through better shadow map integration. This results in more detailed and realistic cloud formations.
- The updated cloud layer system supports blending between two distinct cloud setups, enabling smooth transitions (e.g., sparse to overcast skies via volume blending). Curve blending in the volume framework allows seamless transitions between volumetric clouds.
- This update reduces visual artifacts like flickering or shimmering (seen with the previous presets, which relied on Look-Up Tables).
- Volumetric Clouds respect the **Rendering Space** setting in the **Visual Environment**. Use **Camera Space** to prevent the camera from entering the clouds.

Refer to HDRP documentation for the Cloud Layer and Volumetric Clouds overrides.

Environment sample

The Environment sample from the Package Manager Samples tab includes several sample clouds. You can use these as a starting point to create a variety of clouds.

This sample includes different **Volumetric Cloud** setups. Each prefab demonstrates a different setup to define the cloud formations:

Simple mode: This uses predefined density and erosion settings to generate general cloud types. Adjust parameters to modify cloud coverage and shape without the need for a custom texture.

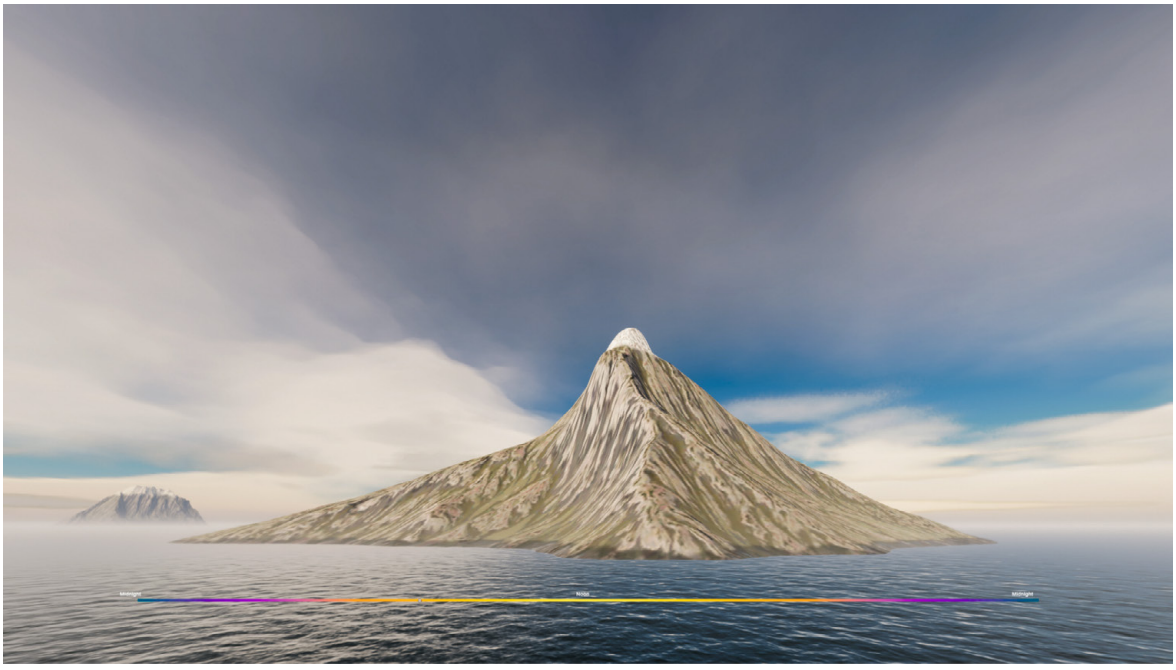
Advanced mode: This places predefined cloud types (Cumulus, Altostratus, Cumulonimbus) at specific world-space positions. A procedurally generated **Cloud LUT** defines the cloud shape, altitude, and density.

Manual mode: This allows full customization by defining a **Cloud LUT** with specific cloud types. A **Cloud Map** determines placement by sampling different colors in the blue channel to position the clouds. Use the Inspector UI to scroll through the available prefabs:

Simple - Clear Skies: This sparse cloud preset only has some subtle stratus clouds.

Simple Clear Skies

Simple - Cloudy: A cloudy preset adds an extra cloud veil for denser coverage.

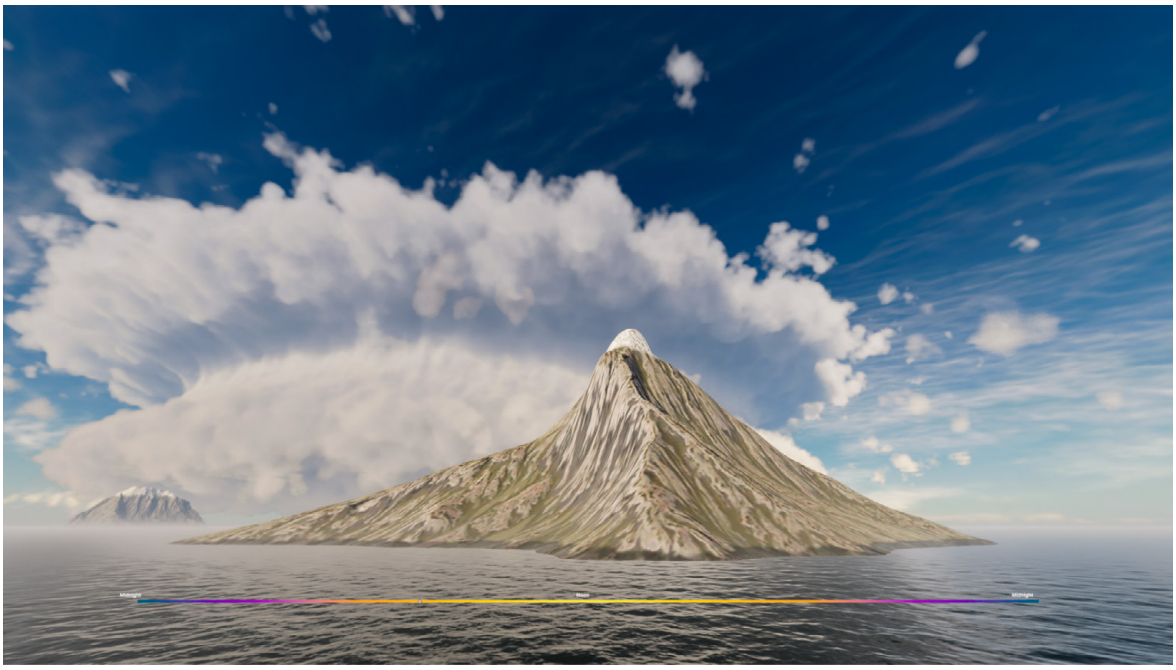


Simple Cloudy

Advanced Clouds: This places a cumulus over the island with layered stratus and cumulonimbus clouds at different altitudes.

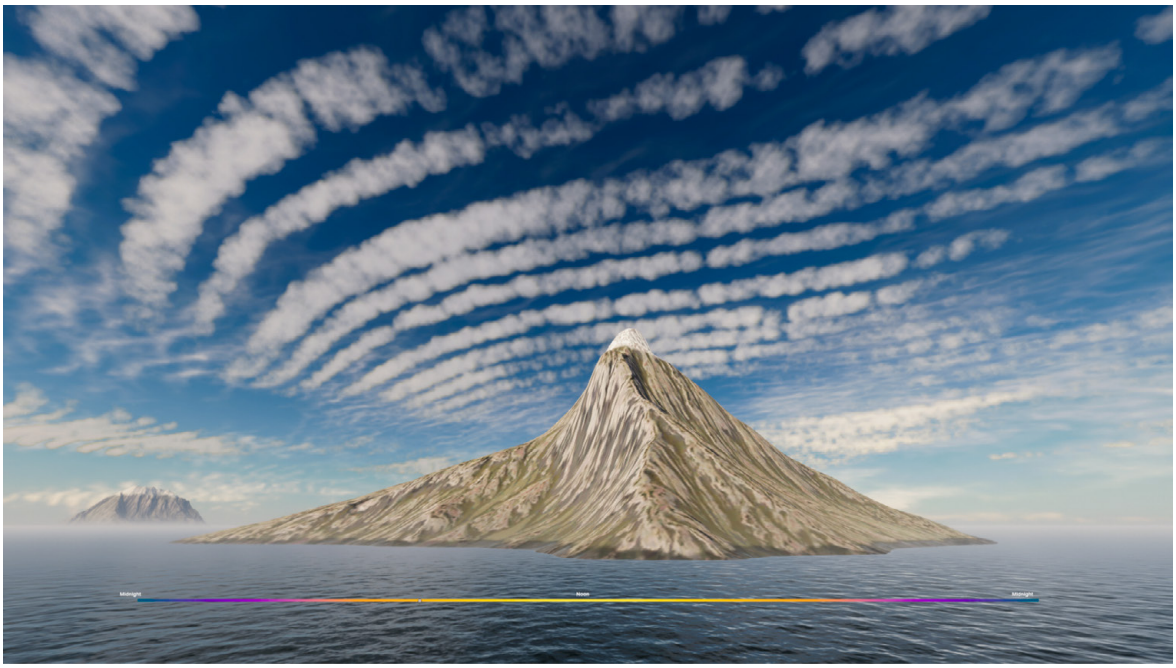
Advanced Clouds

Manual - Cumulonimbus: This prefab uses a custom **Cloud LUT** to place a single cumulonimbus cloud for a dramatic sunset effect.



Manual Cumulonimbus

Manual - Cirro Stratus: This creates a cirrostratus clouds with a deformed gradient and noise texture for a natural variation.



Manual - Cirro Stratus

Manual - Complete LUT: This is a full **Cloud LUT** with multiple cloud types sampled via a **Cloud Map**, using gradients and noise textures for diversity.

Manual - Complete LUT

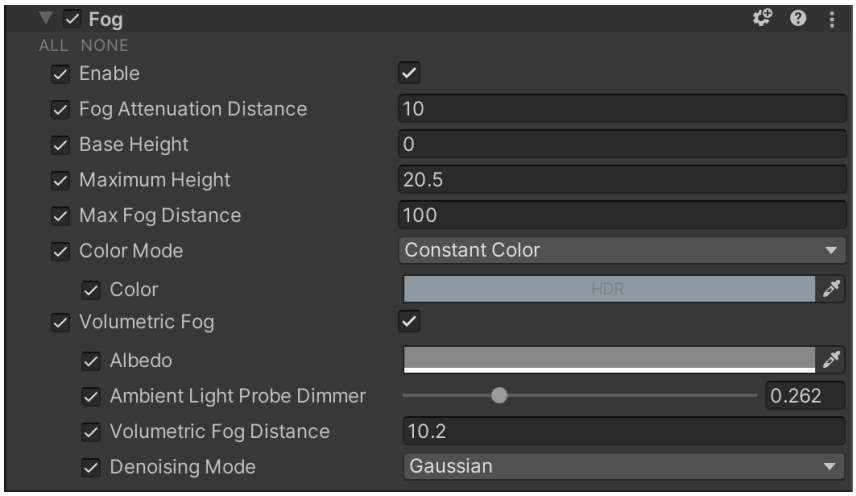
Fog and atmospheric scattering

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.

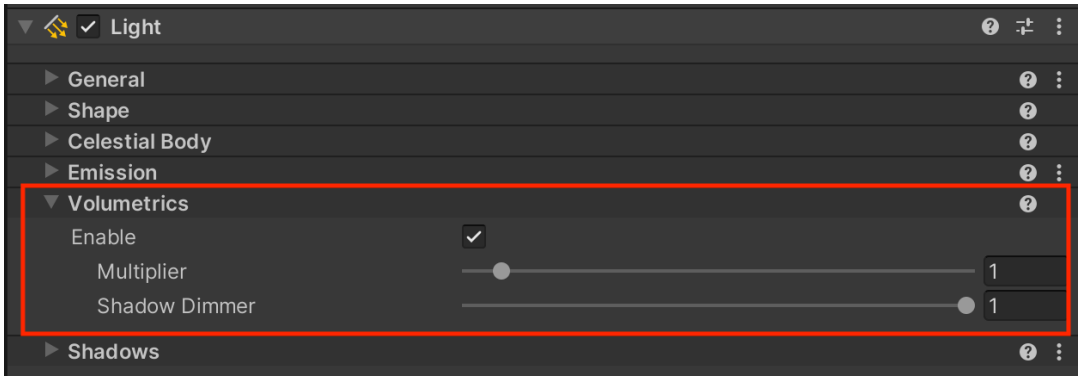
Volumetric Fog accurately holds out the foreground geometry.

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.

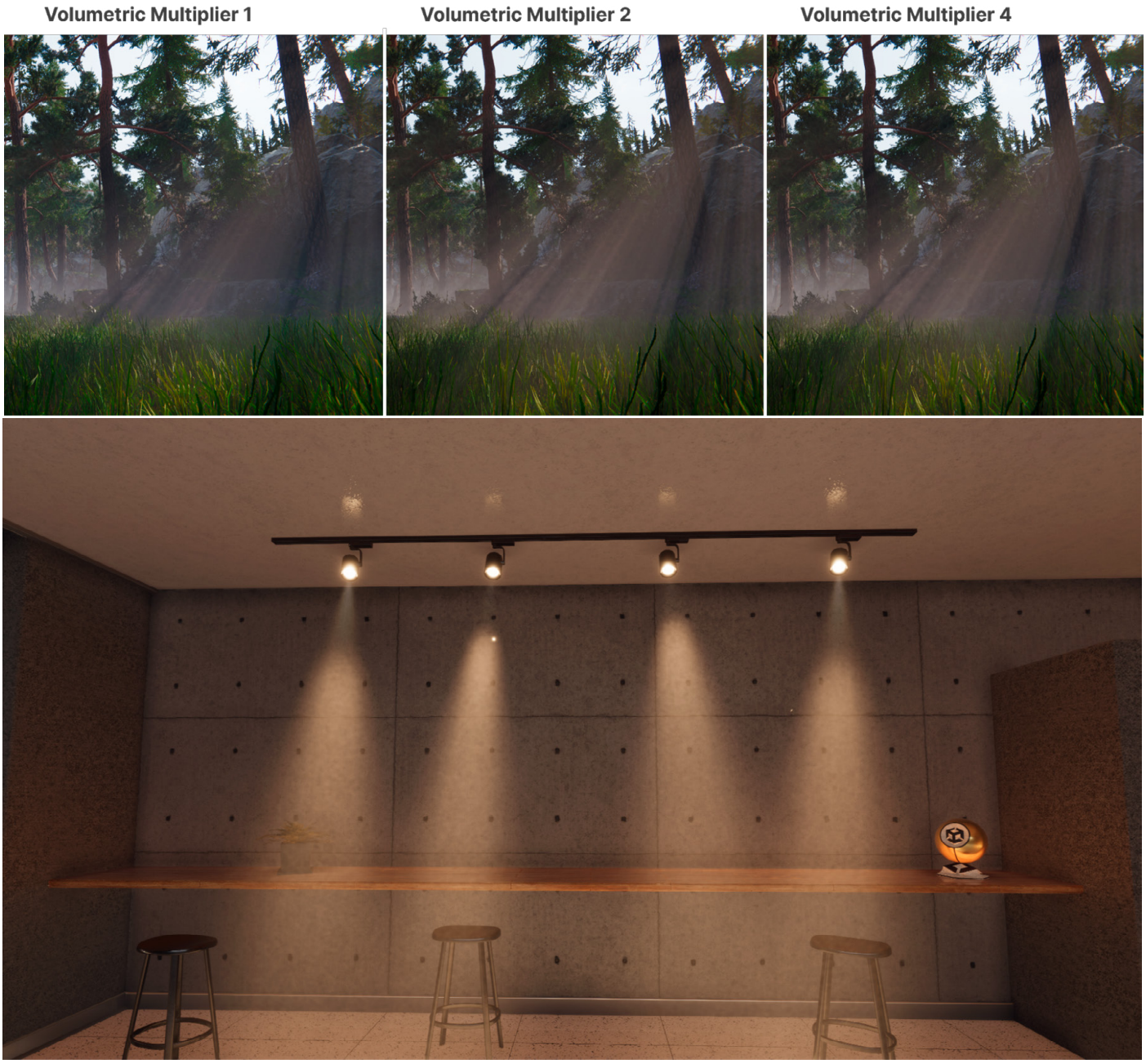
Tips for volumetric lighting and shadows

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.

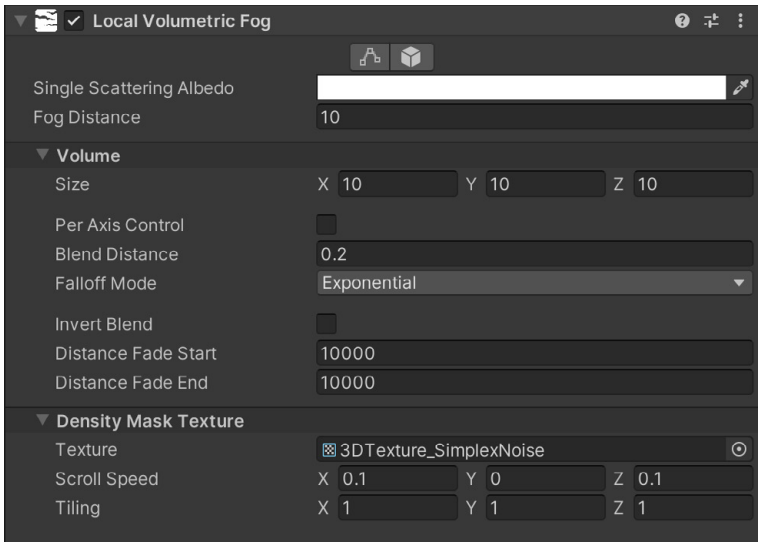


Volumetric lighting applied to the ceiling spotlights in the HDRP sample

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**).



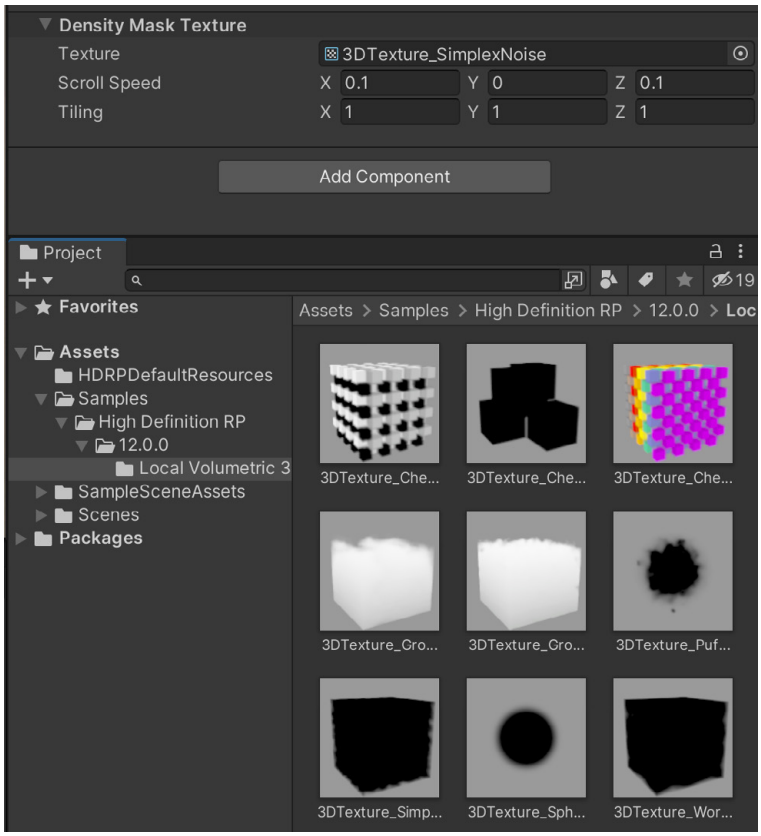
The Local Volumetric Fog component

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.

HDRP 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.

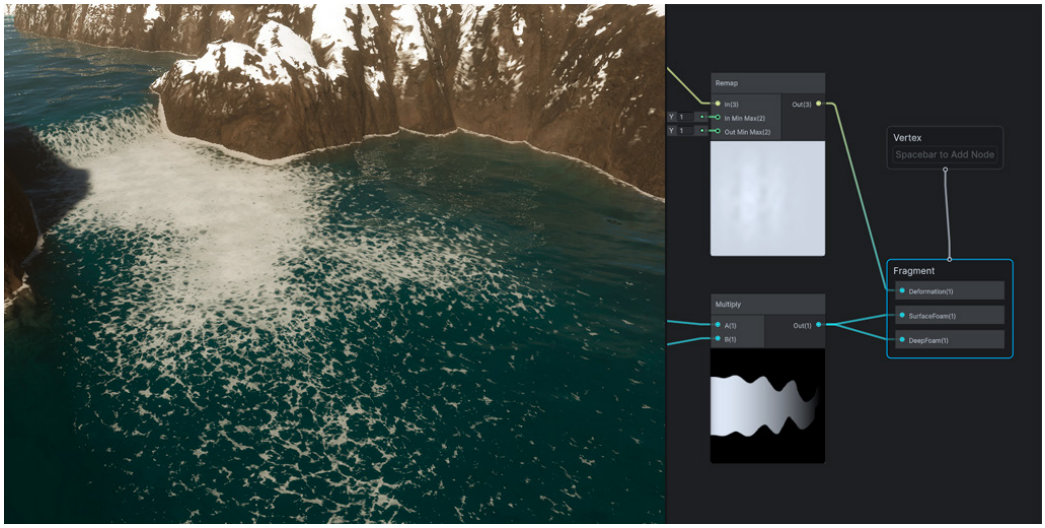
The water system is flexible and customizable.

Unity 6.1 new features

We recommend starting with Unity 6.1 (HDRP 17), which includes several new features for use with the water system.

- **Improved Transform support for water surfaces:** Infinite Oceans and Instanced Quad surfaces now support translation, rotation, and negative scaling, allowing flipping of the surface.
- **Water decal ShaderGraph target:** A new ShaderGraph target enables direct output of water deformation and foam to the atlas. This eliminates the need for a Custom Render Texture. Foam animation now follows the water's direction for a more dynamic effect.
- **Enhanced underwater rendering:** Volumetric fog now supports god rays and light shafts, enhancing the immersive feel of underwater environments.

— **New Camera Height Node in ShaderGraph:** A new ShaderGraph node can sample the camera's height relative to the water surface. This enables effects like water droplets on the camera lens (see the **Underwater** sample scene below).



A new ShaderGraph target enables foam output.

You'll also need Unity 6.1 or higher to explore the **Cave** and **RollingWav**e sample scenes.

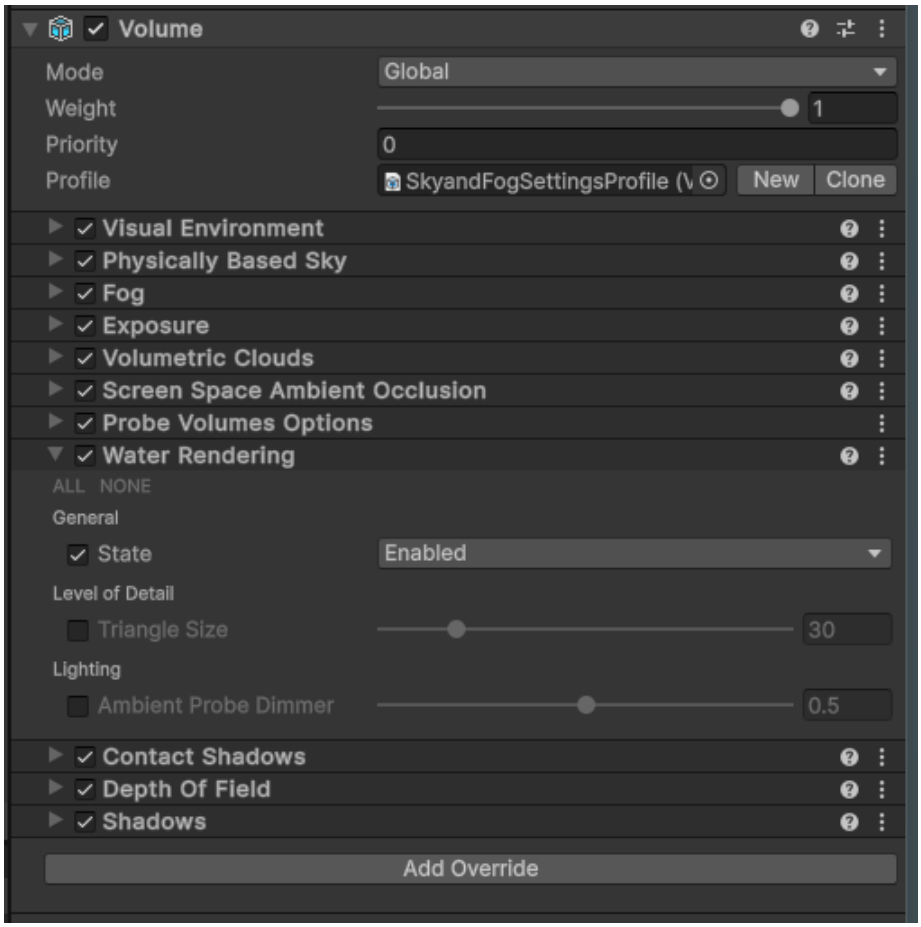
Get started with the water system

To get your project ready for the water system:

— Activate and configure water for each Render Pipeline Asset (per quality level that requires water).

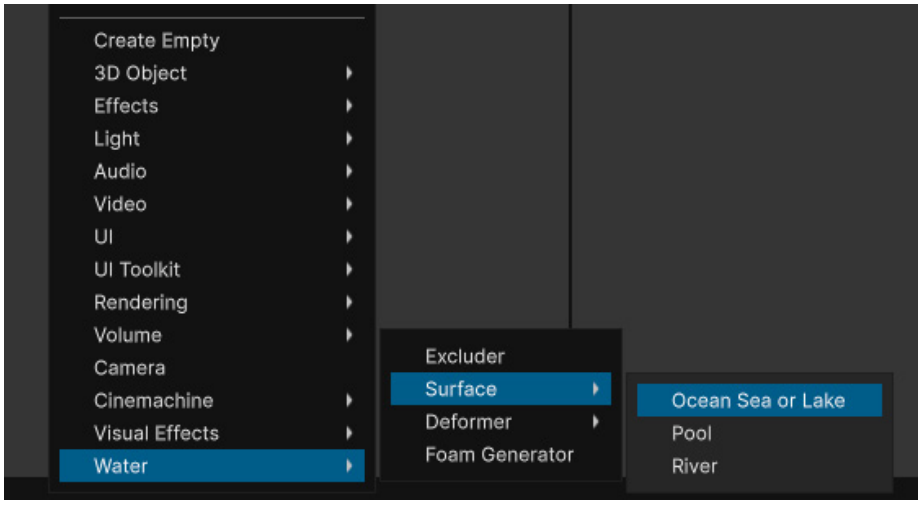
— Enable water in the Frame Settings of your camera(s) (**Edit** > **Project Settings > Graphics > HDRP Global Settings**).

— 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.

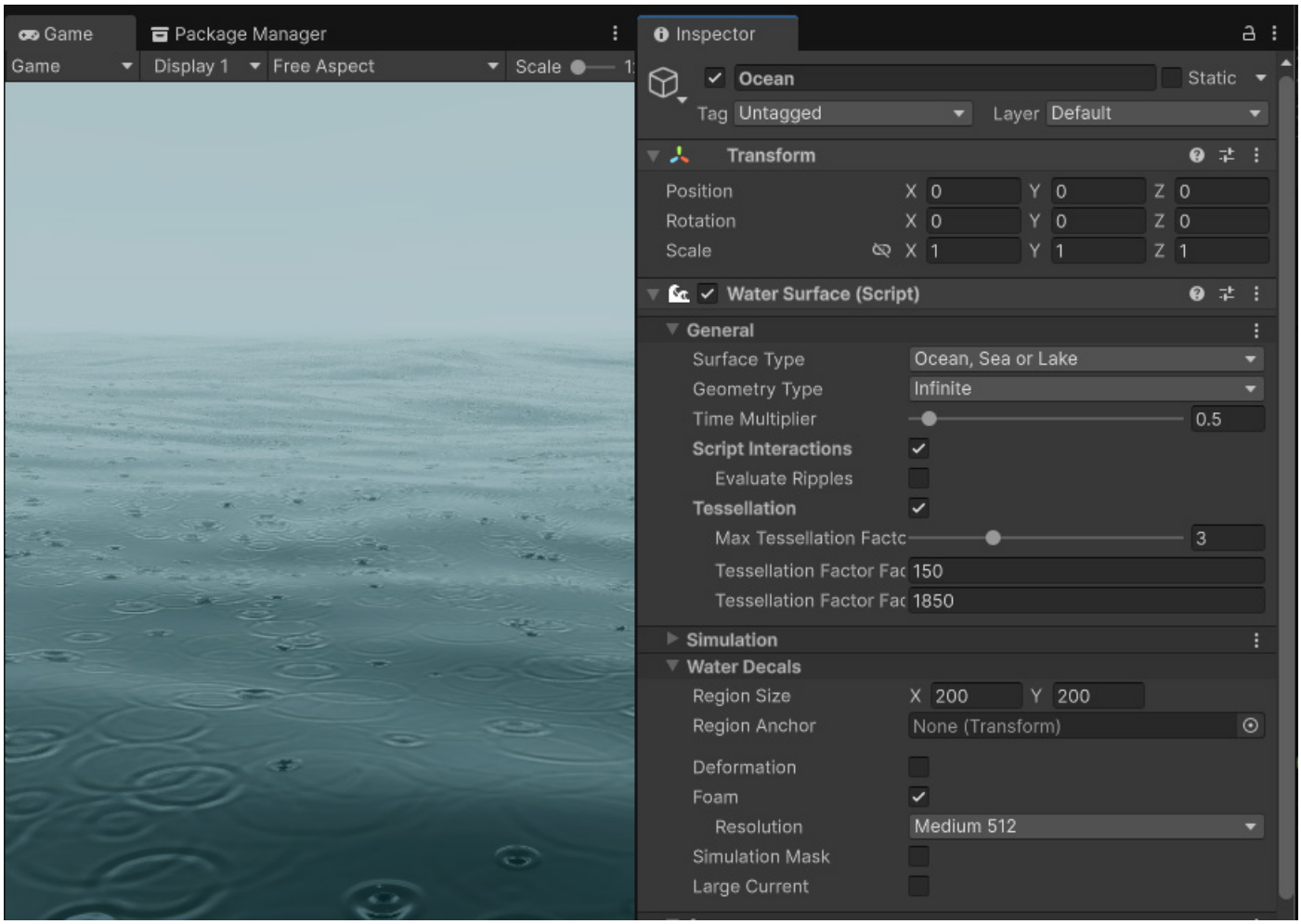
Once water is active, you can add preconfigured water bodies from the GameObject menu. HDRP provides three water surface types: Pwool, river, and ocean.



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

— **Water Decals:** Influences the center and size of the water decals (see below); use these settings to toggle deformation, foam, simulation mask, and currents, as well as set their texture resolution

- **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

Water decals (Unity 6.1)

Unity 6.1 adds fine-grained control over water effects using water decals. A water decal is a Shader Graph Master Stack applied in world space. It has several applications:

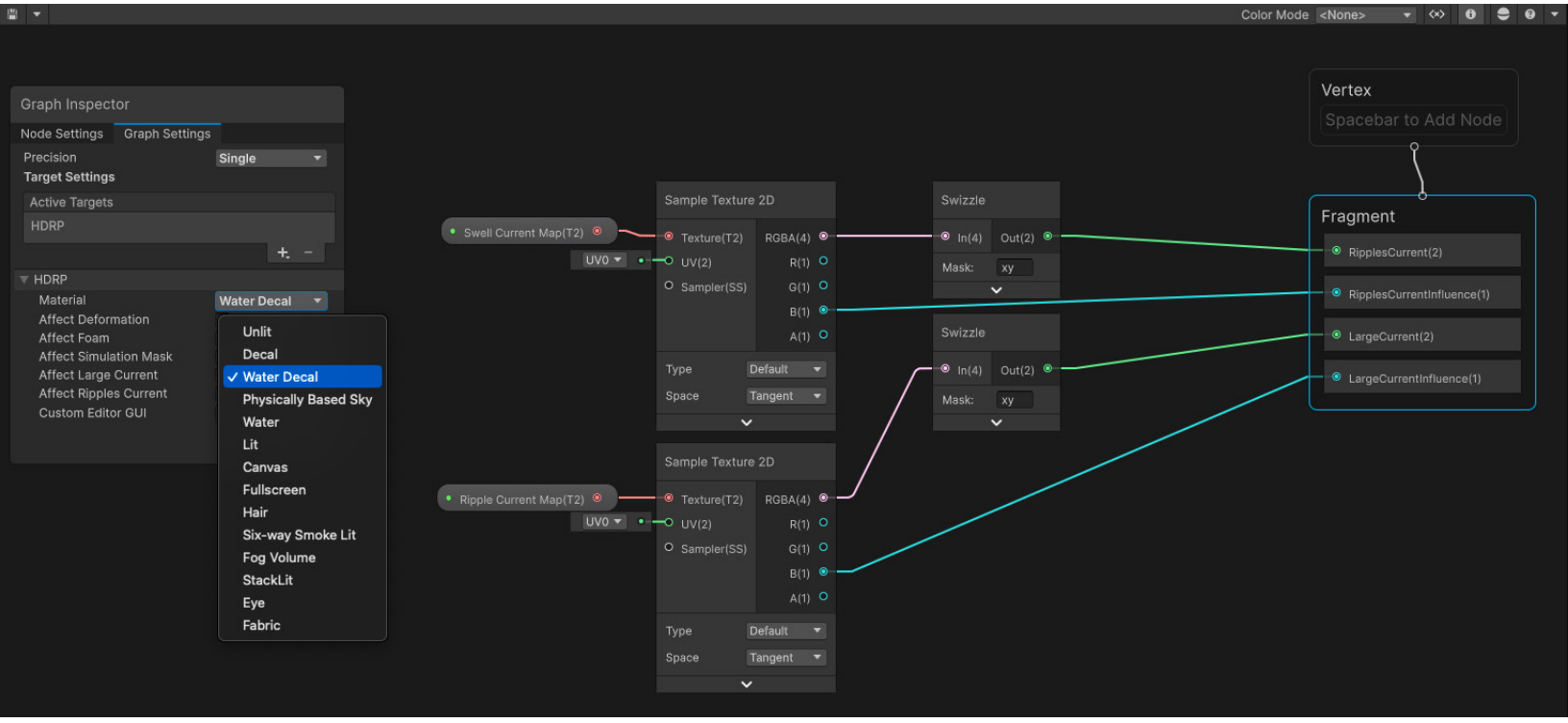
- **Currents:** The decal can define the localized flow of water.
- **Deformation:** Modify the height and displacement of the water surface.
- **Foam:** Add whitewater effects where water interacts with objects or terrain.

To create a water decal, navigate to the **Create** or **GameObject** menu and select **Water > Water Decal**. HDRP will add a GameObject to your scene. Move the decal to the area where you want to apply the effect.

Use the settings under the **Water Decal** property on the Water Surface component to modify the texture resolution or to toggle specific

Edit the Water Decal's Shader Graph to determine how the decal affects the water. Once you have enabled mask and current water decals, you can add the following water features through the Graph Inspector:

- HorizontalDeformation
- SimulationMask
- SimulationFoamMask
- LargeCurrent
- LargeCurrentInfluence
- RipplesCurrent
- RipplesCurrentInfluence



A water decal effect uses a Shader Graph Master Stack.

By default, water decal regions are anchored to the camera. You can also anchor them to a GameObject. See the **RollingWave** scene in the Water package samples for more information about how to use water decals.

Water samples in Unity 6

The best way to explore the HDRP water system is through sample scenes that demonstrate different use cases. In the Package Manager, select the **High Definition Render Pipeline (HDRP)** package and install the **Water Samples** from **Samples** tab.

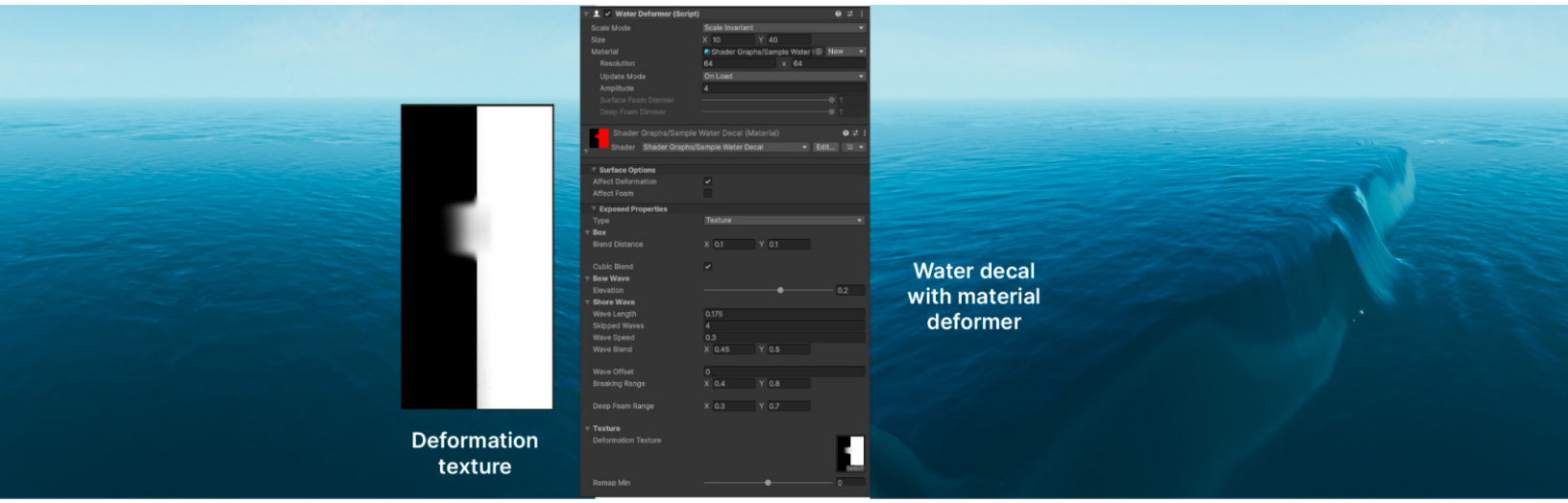
Updated for Unity 6.1, these samples provide a starting point for integrating realistic water with your project.

The glacier

The glacier sample demonstrates how to transition from a river to a larger body of water like an ocean. Updated for Unity 6.1, it now uses the **Mask and Current Water Decal** workflow (enabled in **Project Settings > Graphics**).

Water decals replace the previous water deformers system and now handle all water deformations.

A water decal with a material deformer shapes the central waterfall; this elevates the height of the water based on a deformation texture. Meanwhile, a Decal Projector component with a Custom Render Texture creates the animated texture that simulates falling water.



Water decals deform the water surface.

A **Water Simulation decal** prevents the ocean swell along the river and generates foam at the river's starting point. Meanwhile, the **Water Current decal** directs the flow of water from the river to the ocean.

The glacier sample connects a river with an ocean.

Foam is added along the riverbank by comparing pixel depth values. When the difference between the riverbed and water surface is small, foam is applied.

Foam appears at the edges of the river.

Caustics effects enhance the realism of water interactions by simulating light refraction. Caustics on ice walls are projected using a Decal Projector component, which updates in real time through the GetCausticsBuffer function.

For the riverbed, caustics are enabled directly on the Water Surface component, though they remain unaffected by Current maps, ensuring consistent lighting effects regardless of water movement.

Caustics and ice chunks

Floating ice chunks move with the water currents using the `FloatingIceberg` script, which simulates buoyancy against the water surface. Each chunk also includes a Foam Generator, creating a trailing foam effect as it drifts.

The island

This scene demonstrates open water with shore interactions and floating objects. A single water decal serves several purposes:

— The decal masks part of the swell and agitation using the Affect Simulation Mask option.

— This same simulation mask texture is used in the Water Decal Shader Graph to mask the foam.

— A Current Map texture changes the current direction of the water surface using the **Affect Large Current** option.

The island sample showcases breaking shore waves.

To enhance realism near the shore, shore waves simulate breaking along the coastline. A third-person capsule controller can move through the water, creating a dynamic wake that disturbs the surface. Meanwhile,a **Water Excluder** GameObject ensures that water does not appear inside floating objects, such as the boat in the scene.

The third-person capsule controller moves through the water.

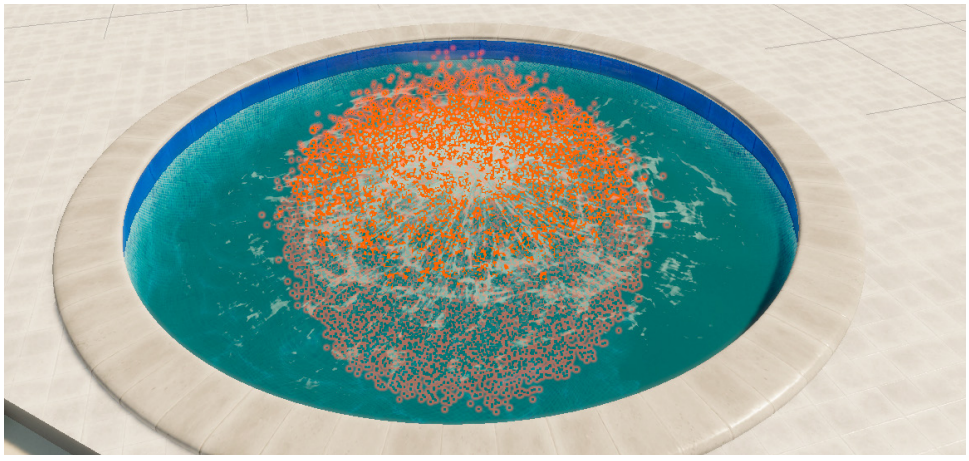
The pool

This scene showcases buoyancy physics and detailed reflections. Drop beach balls into the pool to see how each impact triggers a local deformation and splash effects along the surface of the water.

The resulting beach balls float on the surface using custom buoyancy physics, taking into account mass and volume.

Buoyant objects float on the Pool sample.

To help show off the various reflections on the water, **Screen Space Reflection on Transparent** is enabled in the HDRP asset. Also, note how the Directional Shadow in the Caustics settings dims the caustics effect in the shaded parts of the underwater pool.



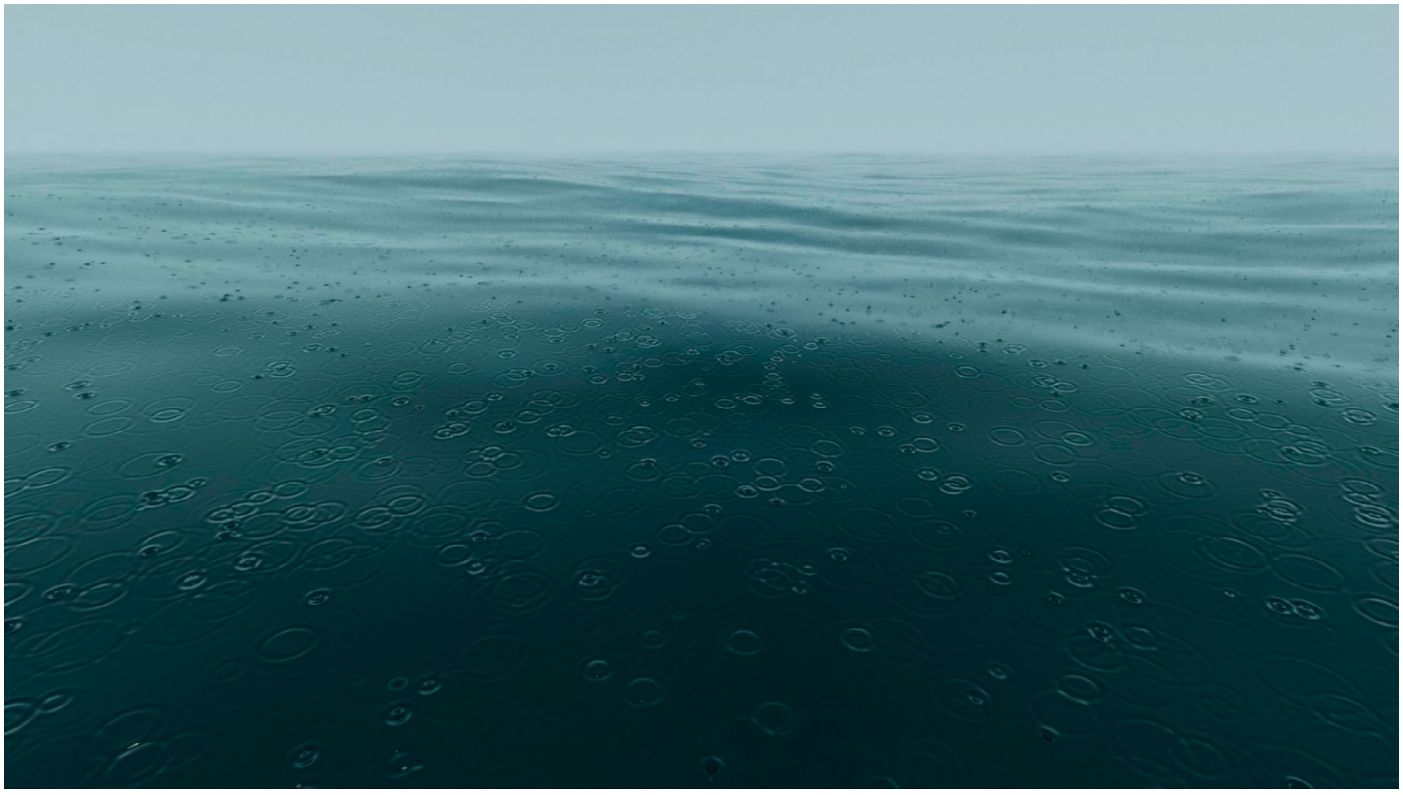
VFX Graph and decals create hot tub bubbles.

Meanwhile, the secondary hot tub simulation includes a Bubbles GameObject. This creates the effect of the moving water using a Decal Projector and an attached VFX Graph.

The rain

This scene simulates rain interacting with an ocean surface. A Custom Render Texture generates rain ripples in real time. A Shader Graph material applies the ripples dynamically to the water surface.

Use the adjustable settings to control raindrop speed, direction, and intensity.



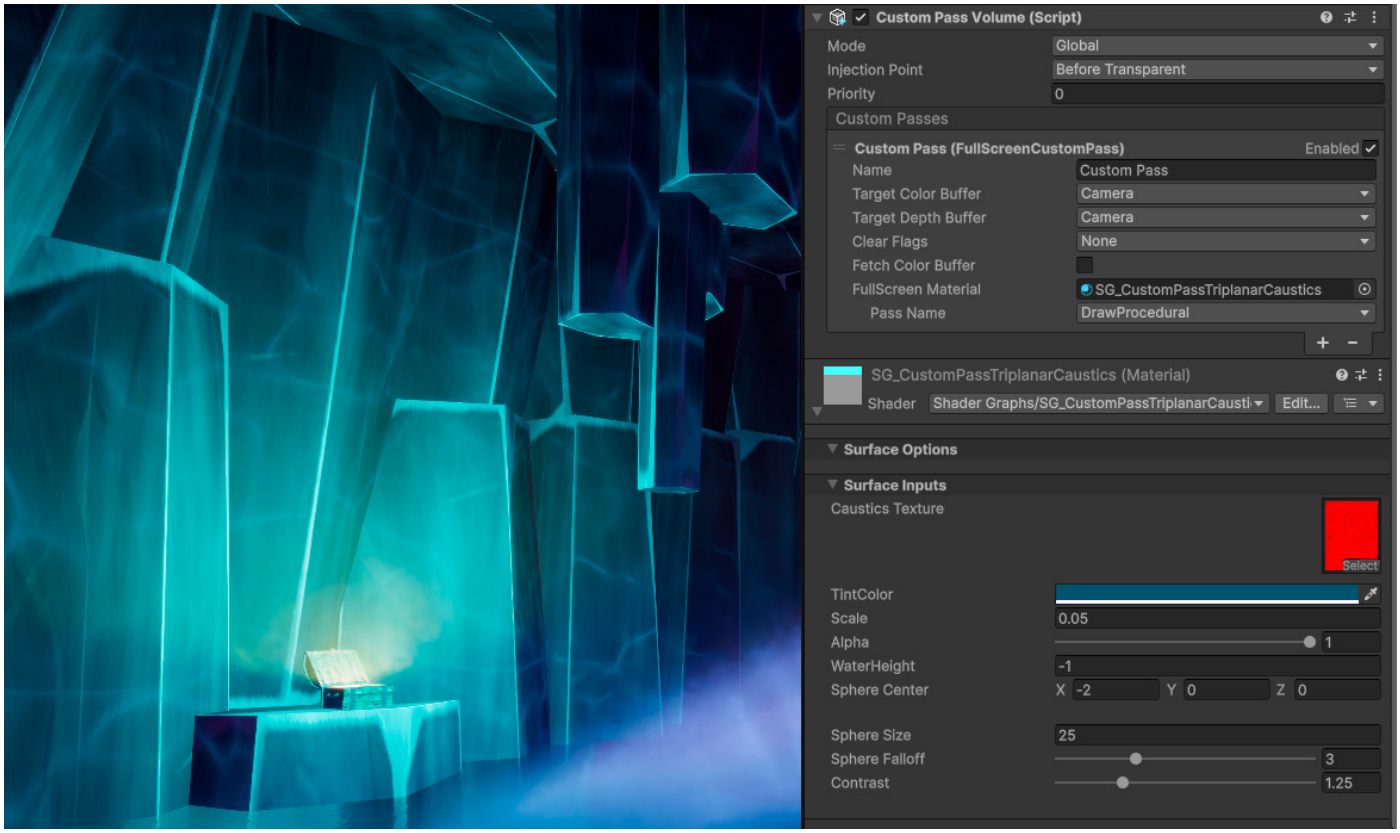
A Custom RenderTexture generates rain ripples.

The following samples require Unity 6.1 or higher, as they showcase new workflows specific to that version.

The cave

This scene demonstrates different uses of caustics textures in Unity 6.1 or higher. Here, we apply lighting effects that might be seen in a cave environment. These can simulate how light bends and focuses through water to create shifting patterns on nearby surfaces.

A Custom Pass projects caustic lighting effects on the cave walls and ceiling using a full screen material and Shader Graph. The shader samples a caustic texture from the Water Surface. You can adjust the **SG_CustomPassTriplanarCaustics** material settings to fine-tune the look.



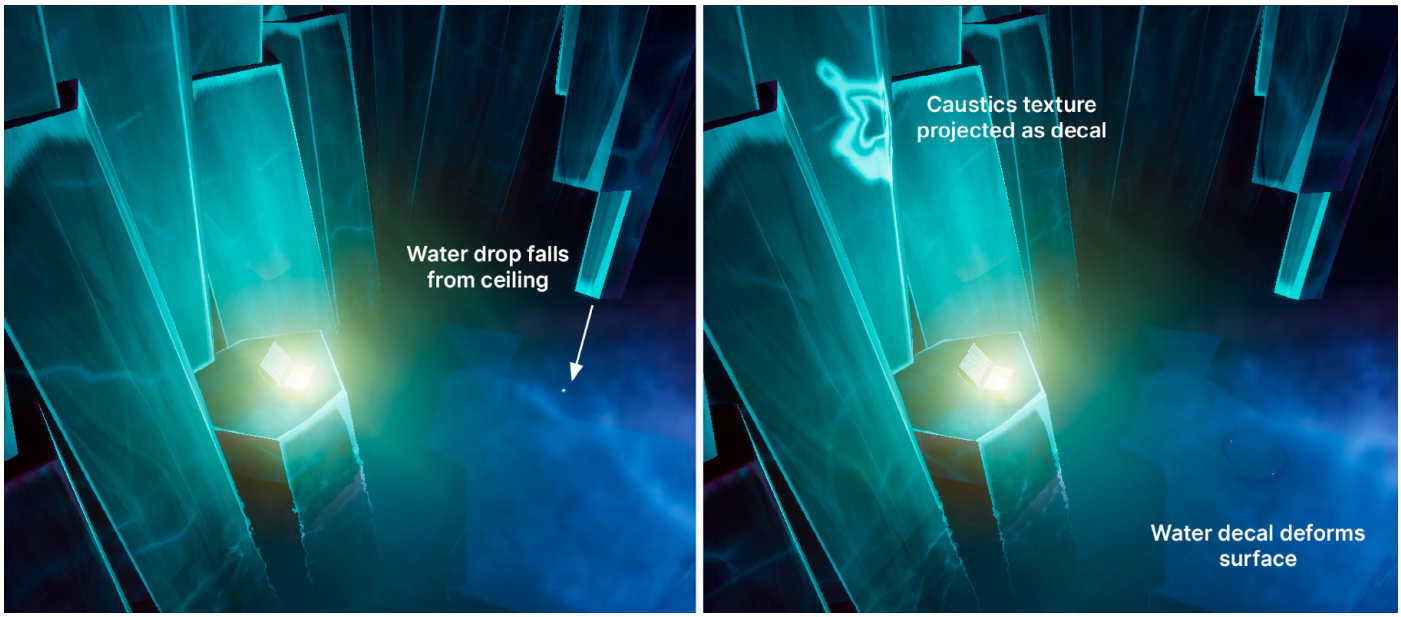
A Custom Pass projects caustic lighting effects.

Edit the **Shader Graph** and to see how it projects the caustics texture along each axis. The shader attenuates the effect over distance and in the -Y direction to avoid caustics appearing on flat, upward-facing surfaces. It also prevents caustics from appearing below the water surface, where the Water System already handles them.

The Cave demonstrates caustics projection and volumetric fog.

For the shaft of light, a **Local Volumetric Fog** with a custom material simulates light patterns bouncing off the water surface. The Shader Graph samples a caustics texture from the Water Surface component to create the soft, atmospheric effect.

Finally, an animated water drop falls from the ceiling, creating ripples on the surface of the water. A Decal Projector simulates the caustics texture that appears on the nearby wall.



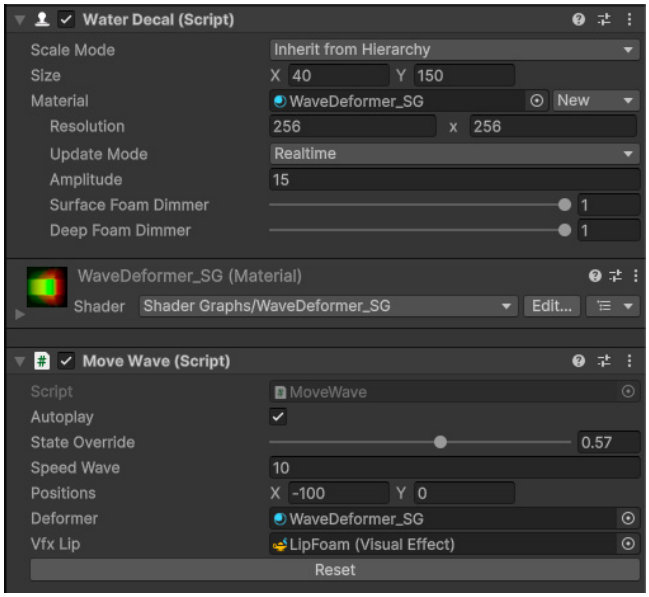
A water drop deforms the surface and projects a caustic on the wall.

The rolling wave

This scene showcases a large rolling wave using deformation textures, using Unity 6.1's water decal deformations. It creates a wave using a Shader Graph and custom movement script.

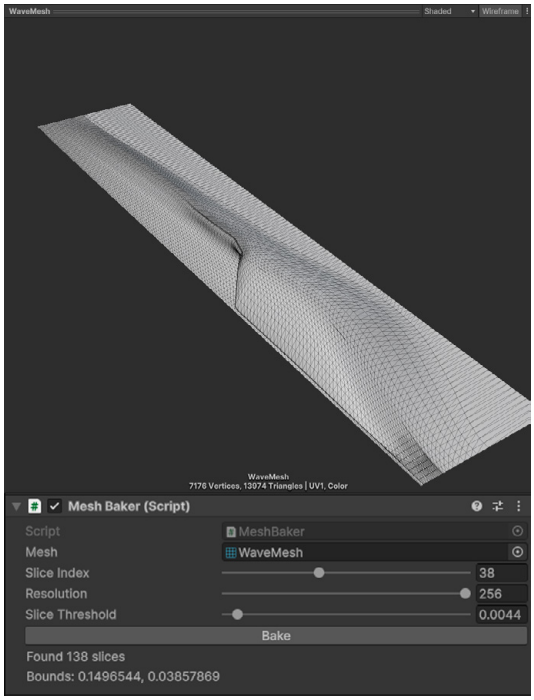
A deformation texture creates a wave.

To create the wave, a deformation texture is sampled in Shader Graph to define the wave's shape. The shader adjusts UVs for symmetry and smooths the wave back into the surrounding water using a blending region. A `MoveWave` script then animates the wave by shifting it along the +X axis.



A Water Decal script deforms the water.

The deformation texture itself is generated from a premade mesh made up of multiple slices, each representing a different wave state. A `MeshBaker` script processes these slices, recording vertex positions and storing them in a texture. Each texture row corresponds to a slice, with texture channels encoding X, Y, and Z deformations.



Generate the deformation texture using a custom Baker.

To test the script, enable the **Mesh Baker** GameObject, adjust settings, and press **Bake** in the Inspector.

Underwater

The Underwater sample demonstrates what happens when the camera dips below the surface of the water. Here, a waterline separates where the water meets the air. By default, this is a sharp line that separates the above-water and underwater views. To make it look more natural, a **Custom Pass** can soften this transition.

This effect uses a **Full Screen Shader Graph** that applies a blur around the waterline. The shader detects which pixels are underwater by sampling a special underwater buffer. It then uses the **HD Scene Color** node to blur only the edges of the waterline, making the transition smoother.

The shader blurs the waterline.

To make the effect look more realistic, the shader slightly stretches the image above the waterline for a meniscus effect.

A Custom Pass adds onscreen water droplets.

The **Custom Pass** also adds water droplets that distort the screen UVs before sampling the **HD Scene Color** node. A **Custom Render Texture** then shifts each frame to create a dripping effect.

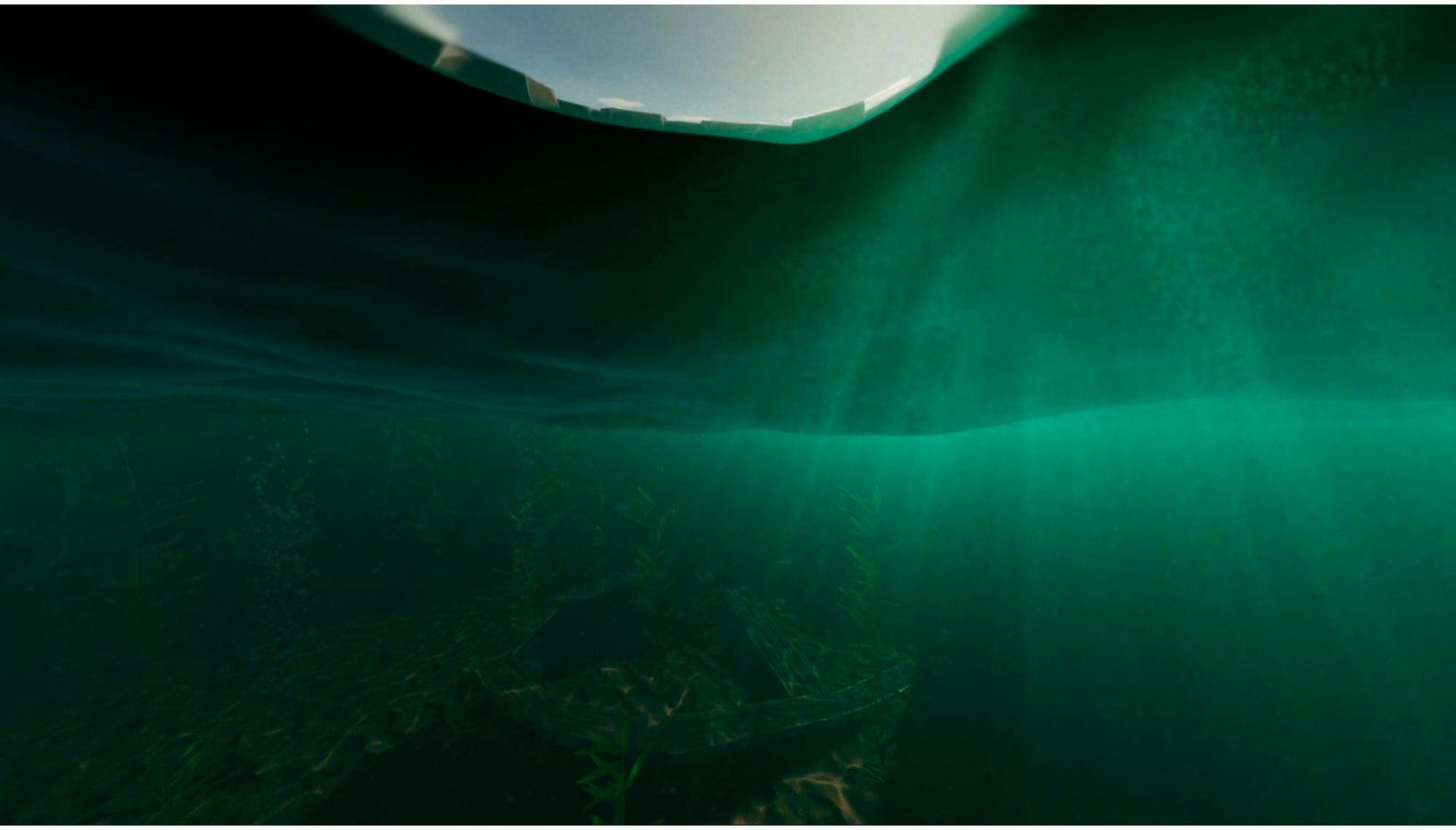
The scene also features VFX Graph effects to bring the background to life:

— Plants use vertex colors and sine waves to animate.

— A school of fish swims dynamically within the water volume.

— Bubbles use the **Sample Water Surface** node to ensure they don't appear above the water.

— Floating particles simulate the turbid underwater atmosphere.



Volumetric shadows add underwater realism.

Volumetric Fog simulates how light scatters in water. It samples the caustics texture from the Water Surface component to create the effect of light rays penetrating the water, adding some dramatic volumetric shadows.

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 included Shader Graphs make use of the Master Stacks (see below) like the Lit Shader Stack, Fabric Master Stack, Decal Master Stack, and others.

The Fabric, Hair, and Eye master nodes usually require detailed artist work within Shader Graph. These samples serve as a good starting point for your own shaders.

The Fabric samples

The Eye samples use carefully designed meshes with specialized UV setups and specific import scale factors. To create eyes of similar quality, examine these eye meshes in 3D modeling software to understand their construction and UV layouts.

Note that some materials require a GPU that supports ray tracing for proper visualization.

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.

An example of Material Variants

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.

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 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.

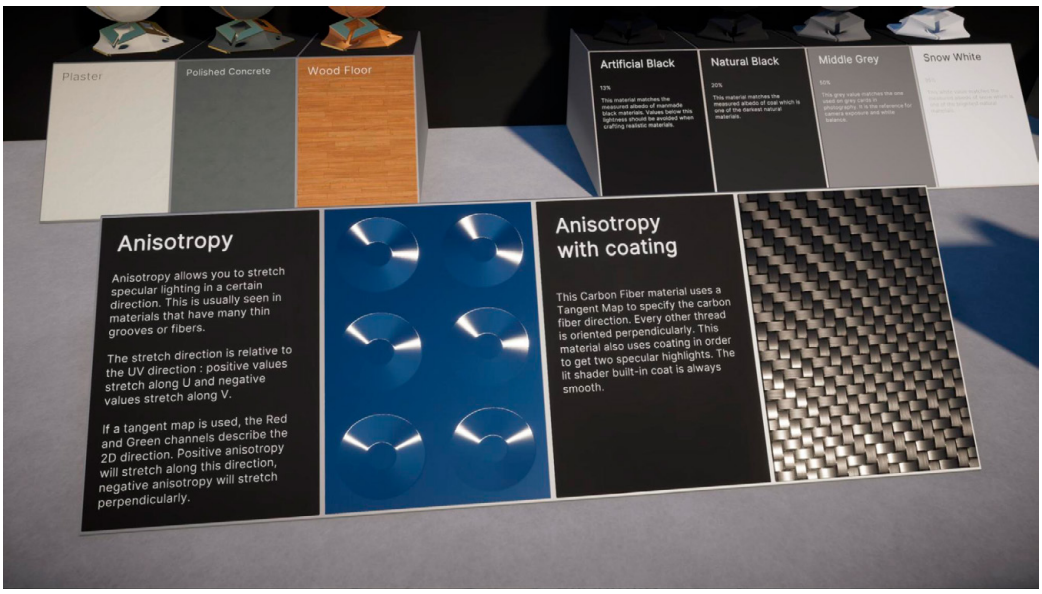
Base Color: Controls both the color and opacity of your material

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.

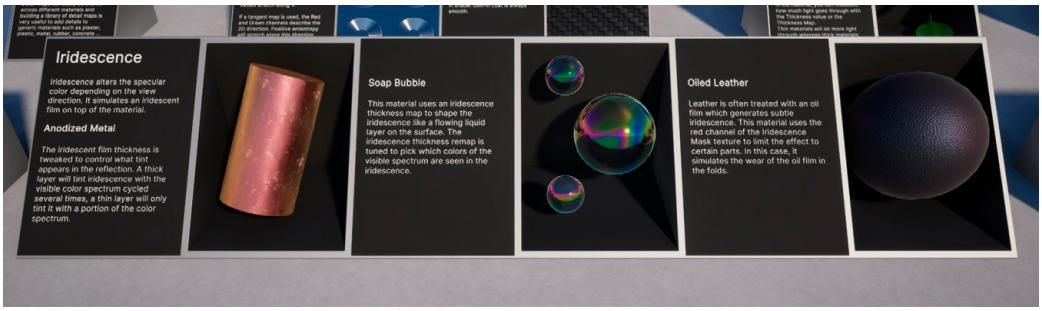
Thickness: Typically used with subsurface scattering (SSS) to give the impression of light penetrating the surface of a translucent material and scattering within

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)

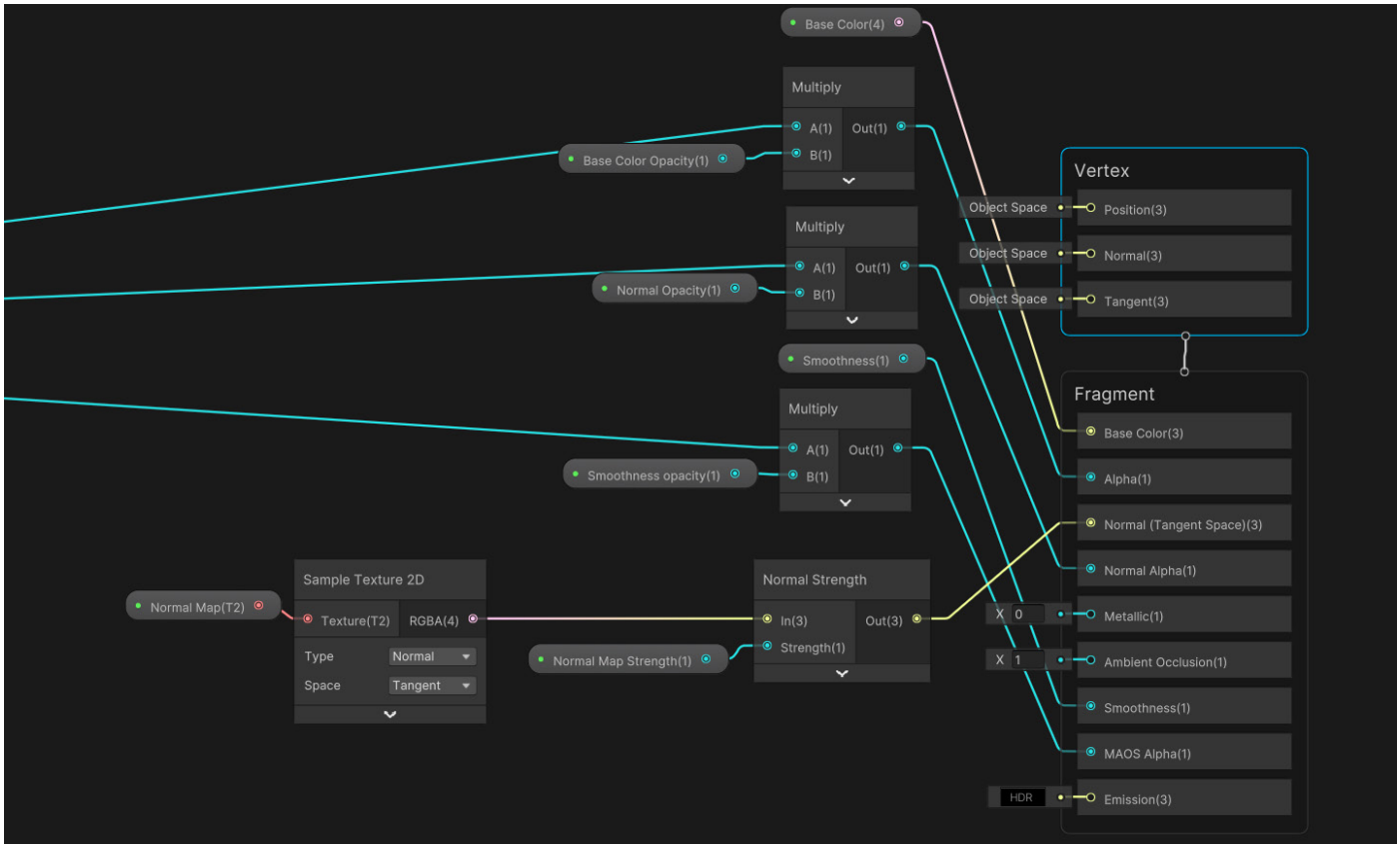


Iridescent objects change color with the viewing angle.

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.

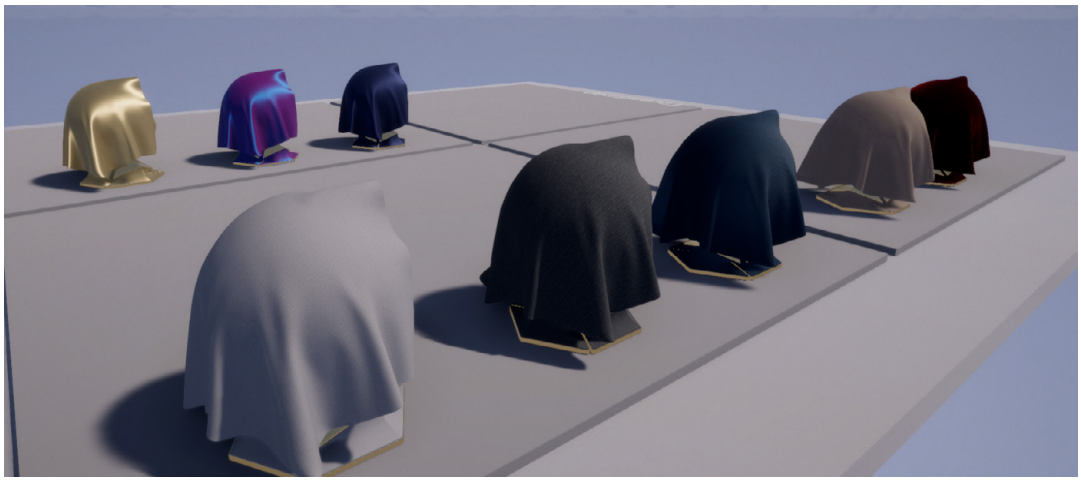
HDRP Master Stacks

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.

The Eye Master Stack helps render realistic eyes.

— **Hair Master Stack:** This simulates hair and fur, with control over color, specular highlights, and how light scatters through individual strands.

The Hair Master Stack simulates how light scatters through strands of hair.

— **Canvas Master Stack:** Use this for UGUI shaders for user interface elements.

— **Fog Volume Master Stack:** This master stack provides specific settings and contexts for fog volumes. Use this to create realistic environmental effects like localized smoke or mist.

— **Fullscreen Master Stack:** Use this for full-screen effects, such as post-processing shaders.

The *Enemies* demo features a digital human character.

See the *Enemies* demo project or the Time Ghost: Character asset to see the Eye Master and Hair Master stacks in action.

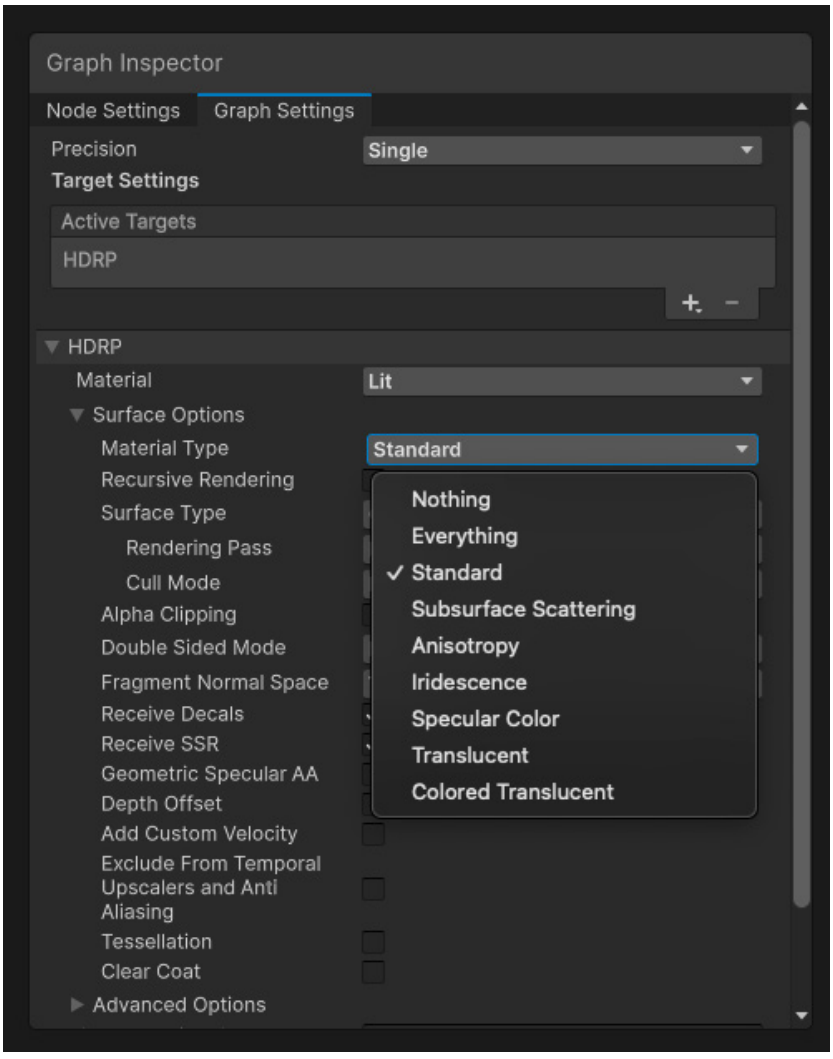
Some Shader Graph nodes also offer HDRP-specific functionality. These can include Diffusion Profile, Emission, Exposure, HD Scene Color, HD Scene Depth, and so on. The Eye Master Stack or Water Master Stack, in particular, have many nodes specific to their stacks. See the complete list of HDRP-specific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

Material Type property

In Unity 6.1, Shader Graph allows you to expose the **Material Type** property (e.g., Standard, Subsurface Scattering, Translucent, etc.) directly within your shader.

Within the Shader Graph, you can specify which Material Types are exposed, enabling you to select the Material Type dynamically in the material's Inspector.

This makes it possible to use a single Shader Graph across multiple Material Types and material variants.



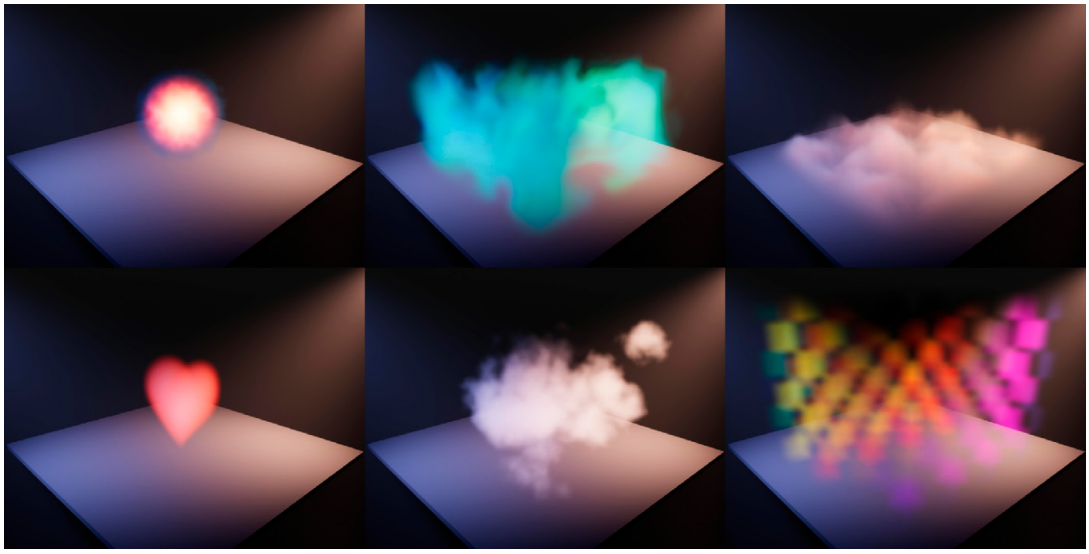
Volumetric Shader Graph fog

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.

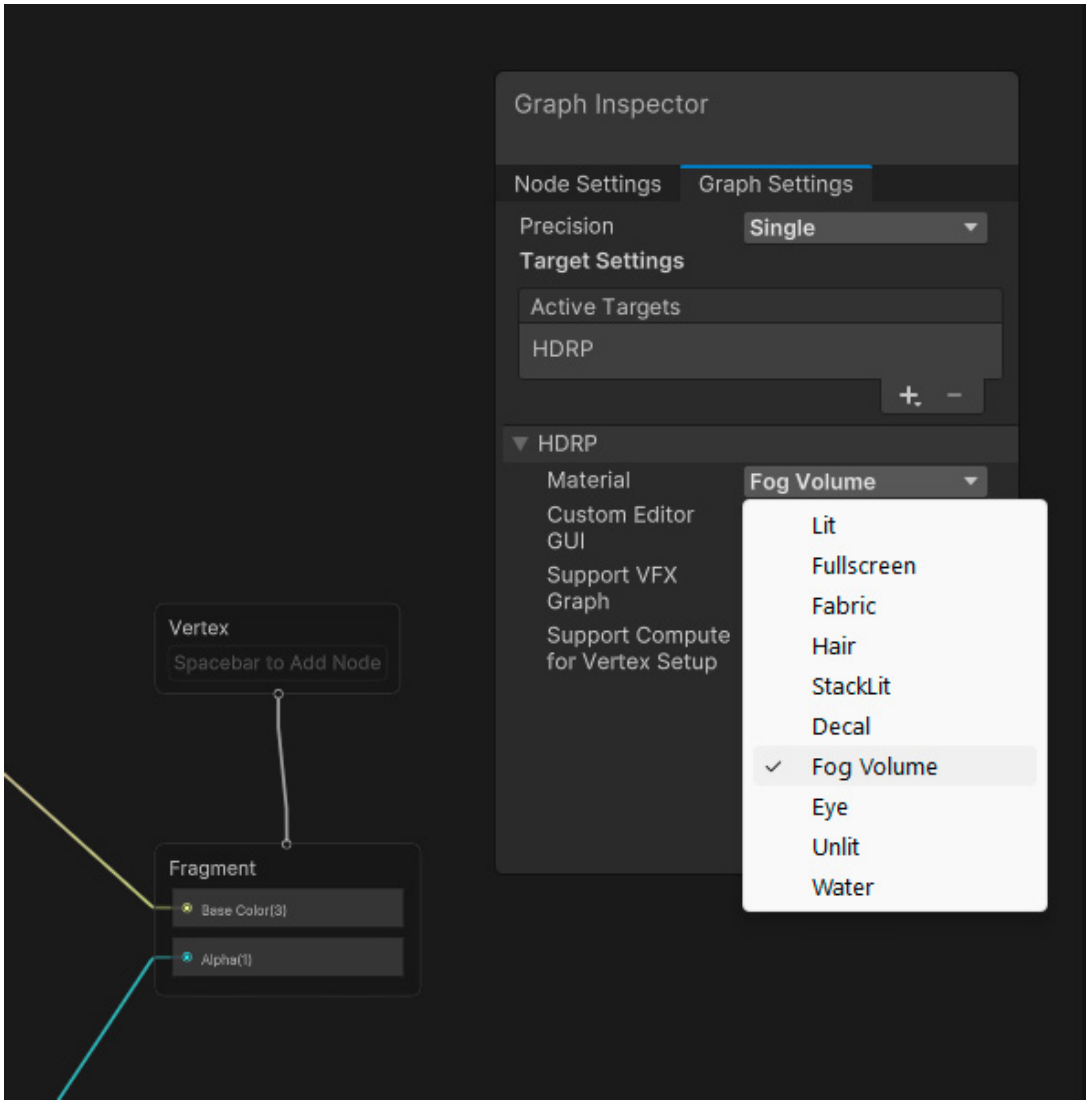
An example of volumetric procedural fog.

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:

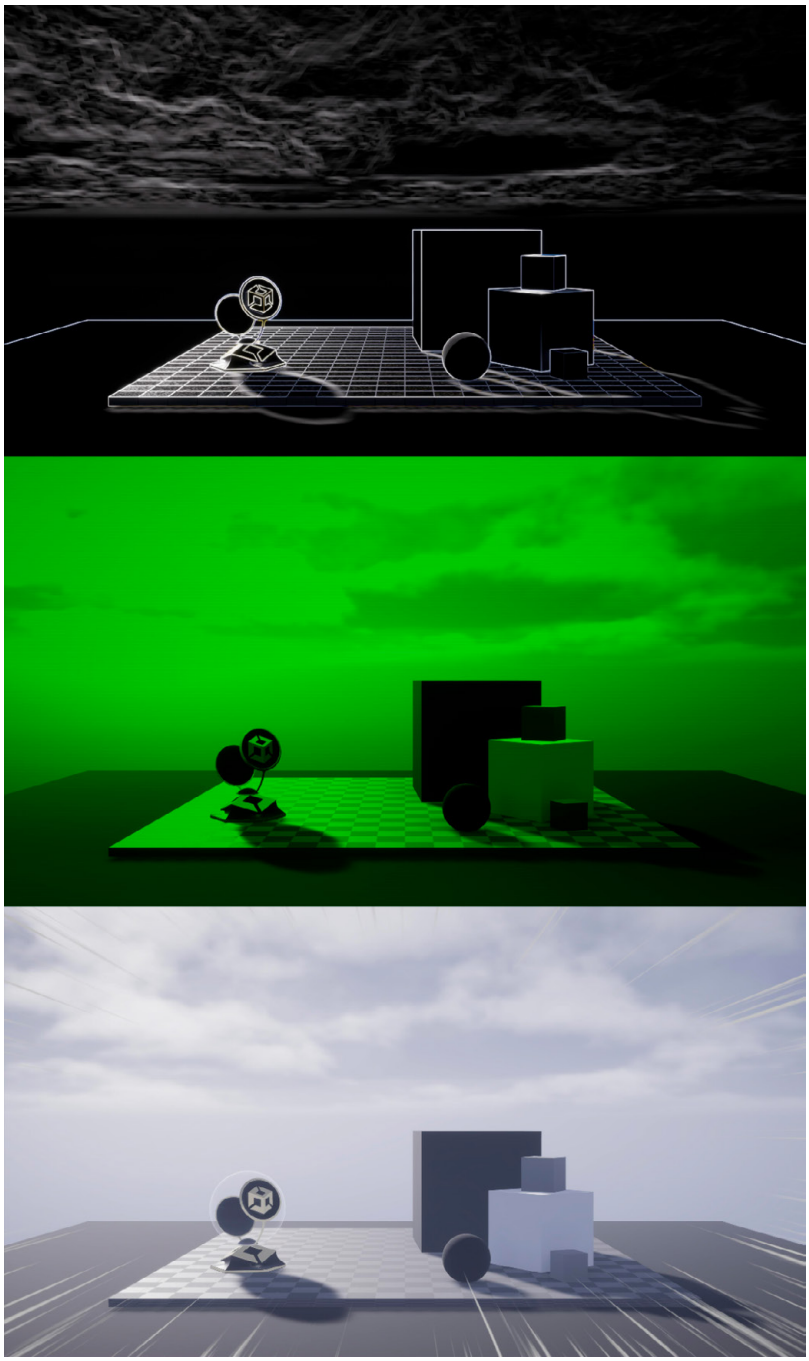
— To create a custom pass effect

— 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.

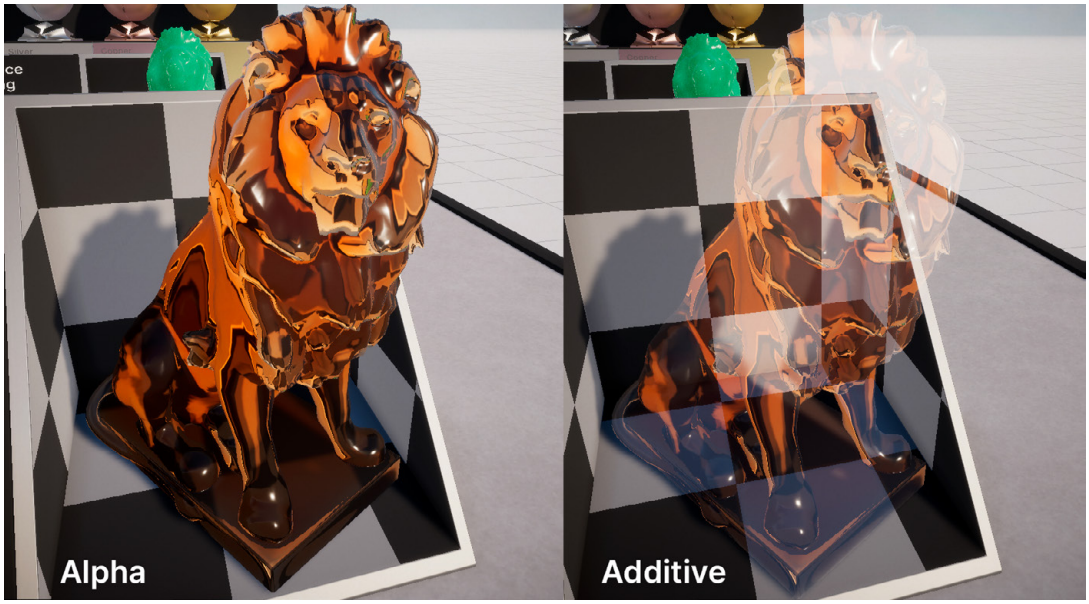


Fullscreen Shader Graph samples

Transparency

Transparency is the quality in a material that allows light to pass through it, 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.

HDRP supports several transparency modes: **Alpha**, **Premultiply**, and **Additive**. Alpha uses the alpha value directly from the texture, while Premultiply multiplies the color with the alpha value before applying it. Additive adds the material's color to the background, which can be useful when making a glowing effect.



Compare Alpha with Additive blending modes.

Lit Material samples

The **Lit Material** samples (available in the Package Manager) show examples of how to set up transparent materials. When using a Transparent Surface Type, select a Refraction Model and set an **Index of Refraction** suitable for the material.

Use the **Thin Glass** Refraction Model to simulate a thin glass surface like a window. The **Transmittance Color** tints the refraction.

For a solid glass object, use the **Sphere** or **Planar** Refraction Model. The Sphere Refraction Model can approximate a gemstone or glass marble. The Planar Refraction Model works for something flat like an ice cube.

Because the renderer doesn't know the shape and size of the object, you can specify a **Thickness Map** with these models to make the refraction more realistic. The thicker the material is in the map, the more tinted it appears.

The Lit Material samples show different Refraction Models.

Transparent Material samples

Unity 6 also includes **Transparent** samples (available separately in the Package Manager) to see the effects of transparency in more detail. These materials demonstrate how to set up shadows, stacking, and refraction when working with transparent surfaces.

The Transparent samples

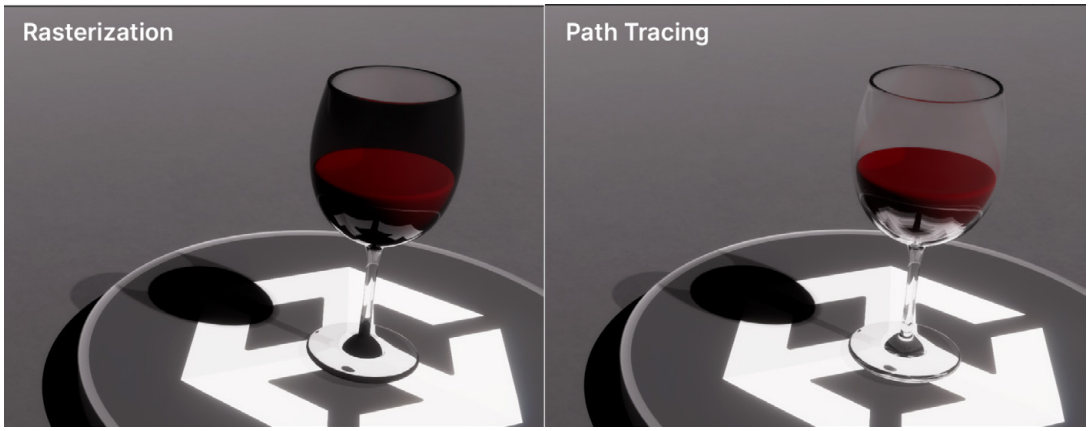
Note that you can switch between several different rendering modes that affect how the transparent materials appear in the scene:

— **Rasterization:** In the default rasterization mode, HDRP approximates refraction effects for performance at the cost of physical accuracy.

— **Recursive rendering:** This mode handles multiple light bounces within transparent materials, producing more realistic refraction and internal reflection. Use this for complex transparent objects where basic rasterization alone is not enough.

— **Ray traced shadows:** This mode uses standard rasterization for the transparent materials but then calculates the shadows with ray tracing. The transparent shadows can then appear more accurate without the complete expense of path tracing.

— **Path tracing:** Path tracing simulates how light travels through a transparent medium. While this produces the most physically accurate results, it's also computationally expensive. Use it to favor quality over performance.



Compare the rendering modes

Shader Graph nodes

By default, Shader Graph includes some nodes that can be used for custom transparency when using rasterization (not compatible with ray tracing). These nodes can be used to simulate refraction and depth-based transparency:

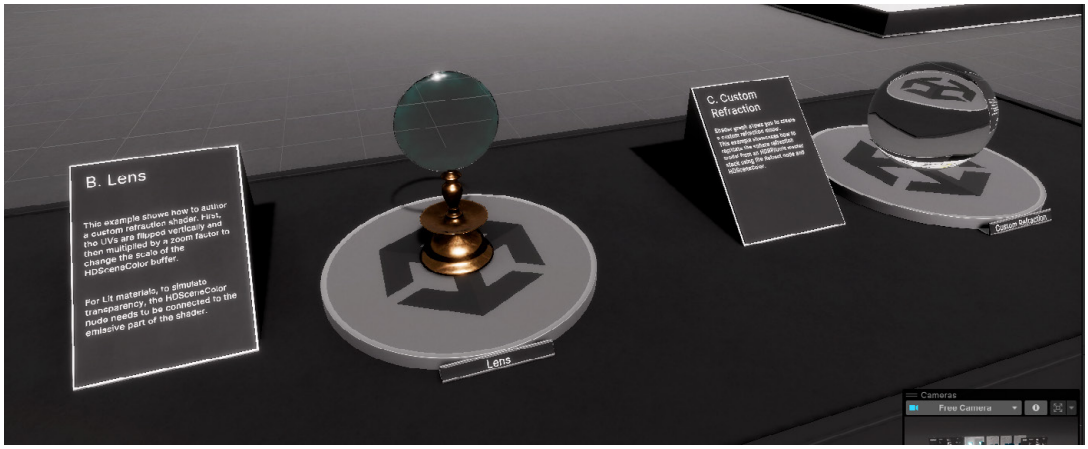
— **Refract**: This node simulates light bending through transparent materials (e.g., glass, water, etc).

— **Scene Depth**: This node captures depth information behind transparent surfaces for depth-based transparency and blending.

— **Scene Depth Difference**: This node measures depth variation between a transparent object and the background. Use this for soft edges and distortions.

— **HD Scene Color**: This retrieves background color data for other transparency, blending, and refraction effects.

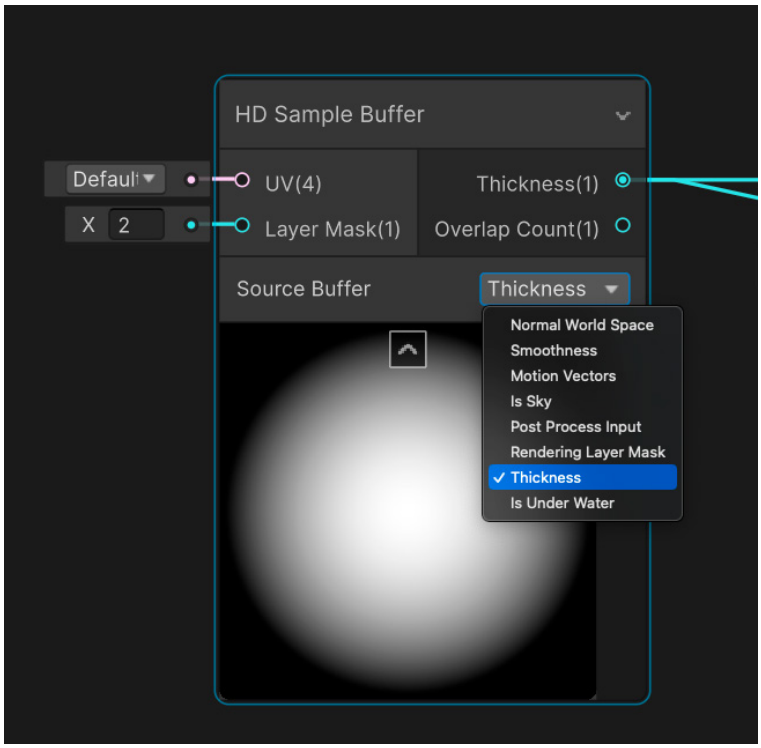
You can combine these nodes to make custom transparency effects in your scenes. These examples show how manipulation of the UVs and HDSceneColor buffer can recreate refraction.



The Lens and Custom Refraction samples

Compute Thickness

Enabling **Compute Thickness** in the HDRP asset allocates a buffer containing the thickness information of objects within the specified LayerMask. Sample this buffer in Shader Graph using the HD Sample Buffer node to affect the color and influence the resulting refraction.



Sample the Compute Thickness in the HD Sample Buffer.

For example, an x-ray shader applied to the gummy bears uses the Compute Thickness buffer to reveal the gummy bear skeleton. The Shader Graph exposes some additional controls in the Inspector to alter the look and appearance of the effect.

The X-Ray shader uses the Compute Thickness buffer.

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.

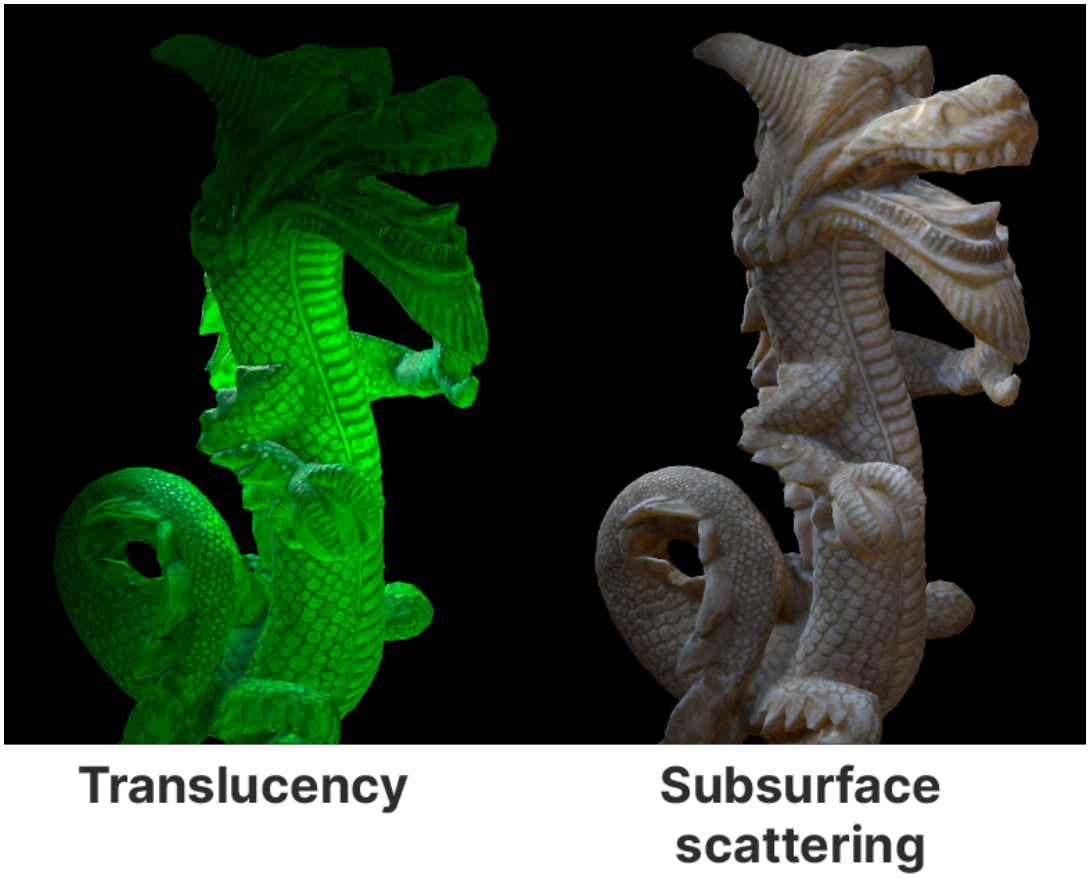
When light hits a translucent material, not all of it reflects off the surface. A portion penetrates the interior and scatters within the material, bouncing around internally. Some scattered light eventually exits the material at a different point from where it entered.

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 only accounts for transmission. Subsurface scattering adds a screen-space blur effect.

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.

SpeedTree and Transmission Masks

A Transmission Mask is useful when working with SpeedTree vegetation to ensure that only leaves receive translucency while bark and twigs remain opaque. Use the Transmission Mask to prevent applying subsurface scattering where it should not exist.

The SpeedTree 8 Sub Graph Assets (HDRP/Nature/SpeedTree8.shadergraph) 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.

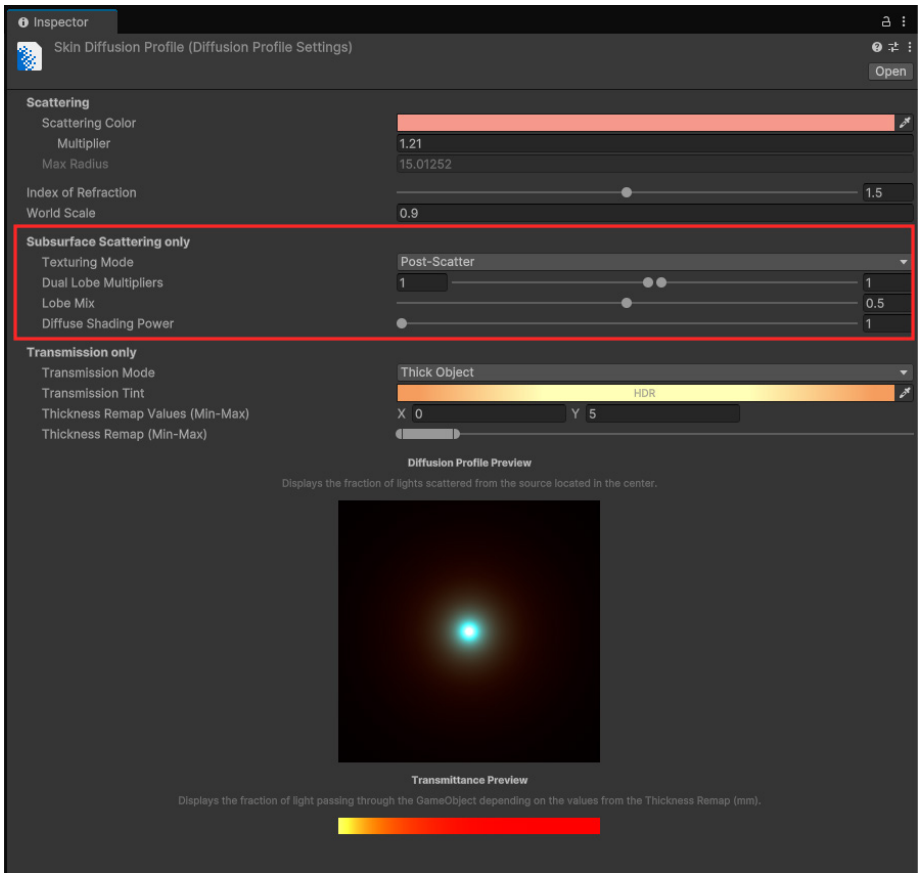
New in Unity 6: Improved skin rendering

The Diffusion Profile in a translucent material controls how light scatters within the surface. Unity 6 introduces new parameters in the Diffusion Profile to improve skin rendering:

— **Dual lobe** specular reflection simulates the thin oily layer on the skin's surface, creating both sharp highlights from the oil and softer reflections from deeper layers.

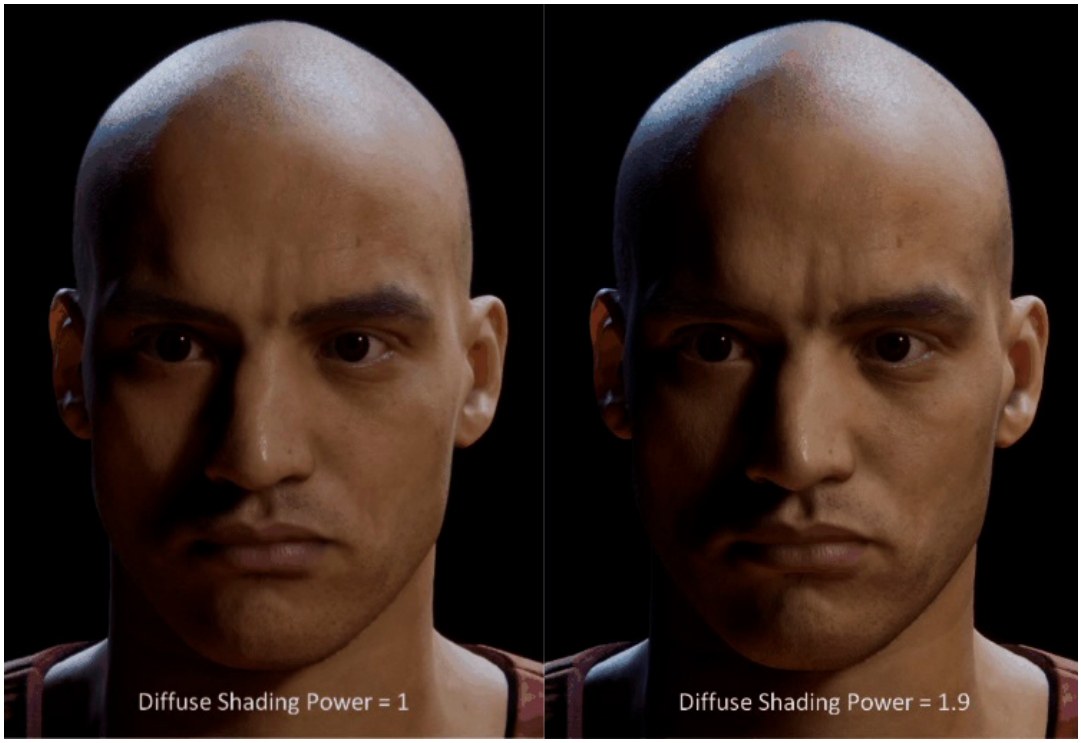
— **Diffuse power** improves how light spreads in materials with strong subsurface scattering (like skin).

These updates make skin materials look more natural under different lighting conditions.



Unity 6 adds parameters to the Diffusion Profile.

Dual lobe captures the specular reflection of human skin.



The Diffuse Shading Power controls how light spreads over the skin material.

Subsurface scattering renders skin more realistically.

Decals

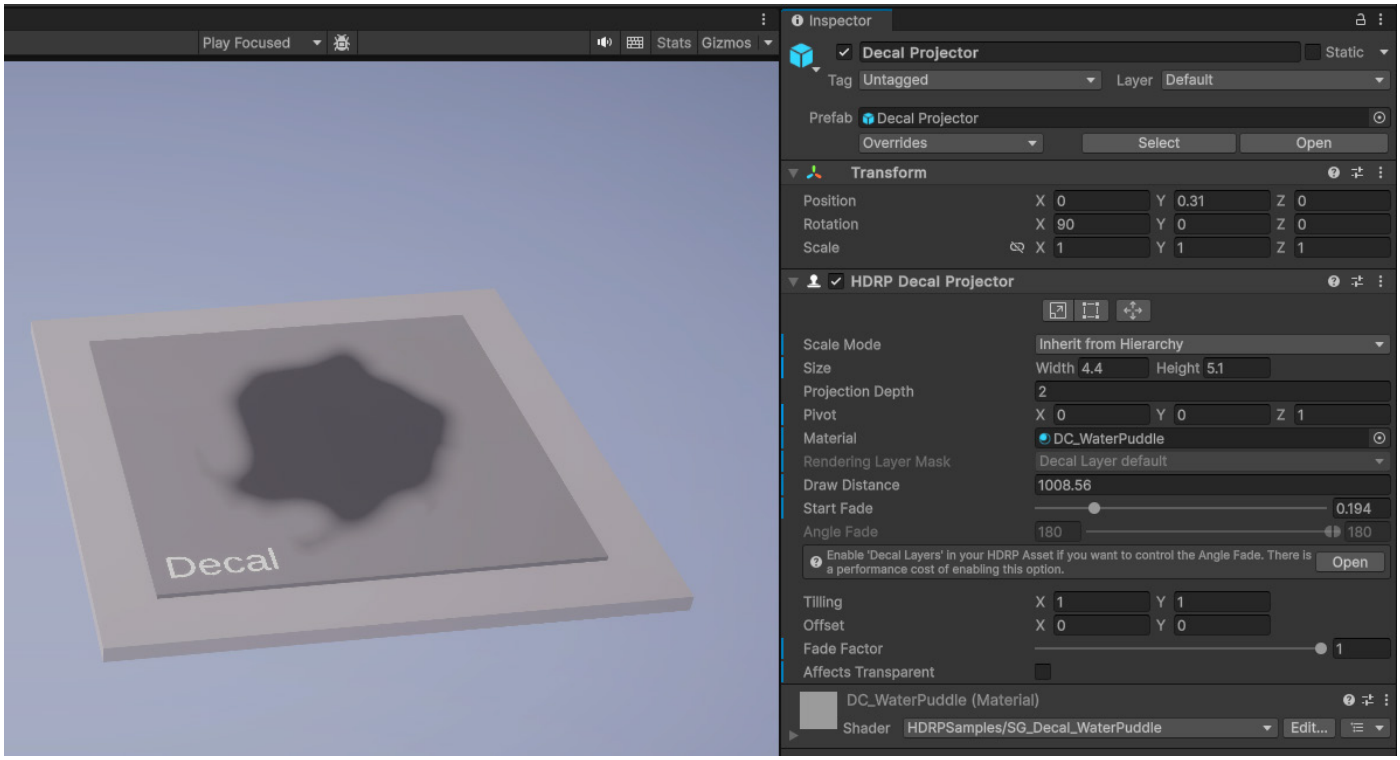
Decals are materials that use the Decal Shader or Decal Master Stack. They add detail to materials and surfaces as graphical overlays without adding additional geometry.

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.

In Unity 6.1, decals created with Shader Graph now affect transparent objects. Use this to build procedural effects like rain drops, ripples, custom engravings, dirt effects on glass and more.

Shaders: Fur and hair

You can use the Hair shader or the Hair Shader Graph as a starting point for rendering hair and fur in HDRP. To create a realistic looking hair effect, the Hair shader and the Hair Shader Graph use layers called “hair cards.”

Each hair card represents a different section of hair. If you use semi-transparent hair cards, you must manually sort them so that they’re in back-to-front order from every viewing direction.

The Hair Master Stack supports two material types for use with hair and fur:

— **Approximate Material Type:** This hair material type is suitable for hair card geometry and based on the Kajiya-Kay hair model. When you use this type, you need to adjust the blocks in the Fragment context to suit the lighting environment in your scene.

— **Physical Material Type:** This hair material type is suitable for hair strand geometry (thin tubes or ribbons) and based on the Marschner hair model. This type is suitable for all light environments.

  Strand-based hair is more detailed and realistic but requires greater computational resources. This makes it ideal for main characters or high-quality close-ups. The Physical Material Type also supports multiple scattering, simulating realistic light interactions within hair strands.

HDRP can help you create realistic hair and fur.

New features in Unity 6.1:

— **High Quality Line Rendering:** Rendering thin lines using traditional hardware rasterization can lead to aliasing and sorting issues, especially for hair and fur. High Quality Line Rendering delivers higher-quality lines with better transparency and anti-aliasing. See Use high quality line rendering for more information.

— **Screen Coverage LOD Mode:** This mode dynamically adjusts the amount of hair strands rendered based on their screen space coverage, improving performance.

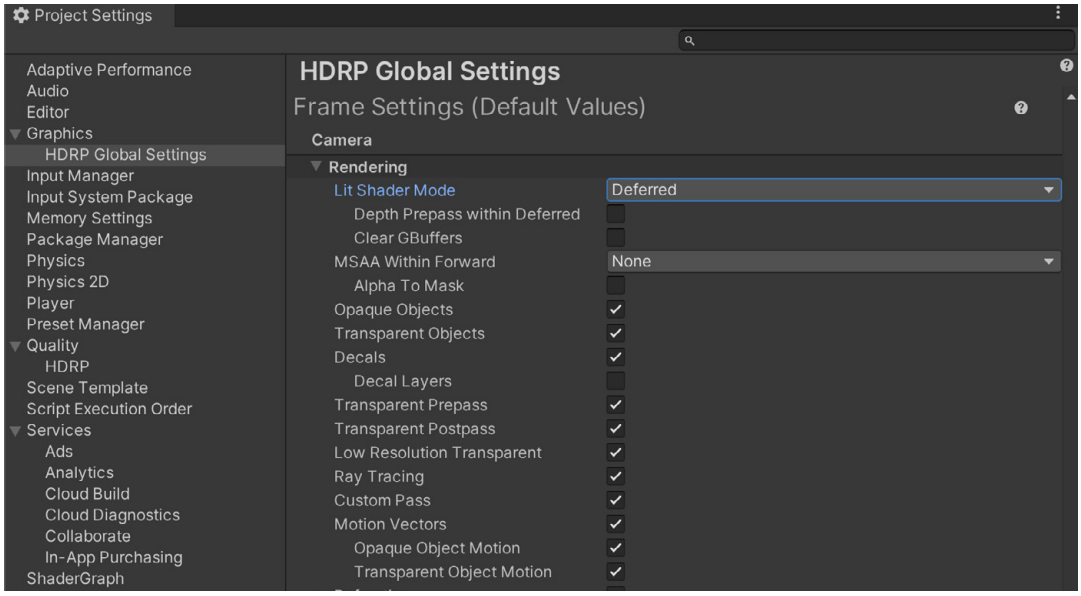
— **Performance improvements:** Strand based hair rendering uses a software rasterizer that can optimize performance, visual quality, and anti-aliasing when rendering a lot of strands.

Refer to Hair and fur documentation pages for more information. If you're new to setting up hair or fur in Unity, see Get Started with Hair Simulation for an introduction to the hair system.

High-quality line rendering renders with better transparency and anti-aliasing

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.

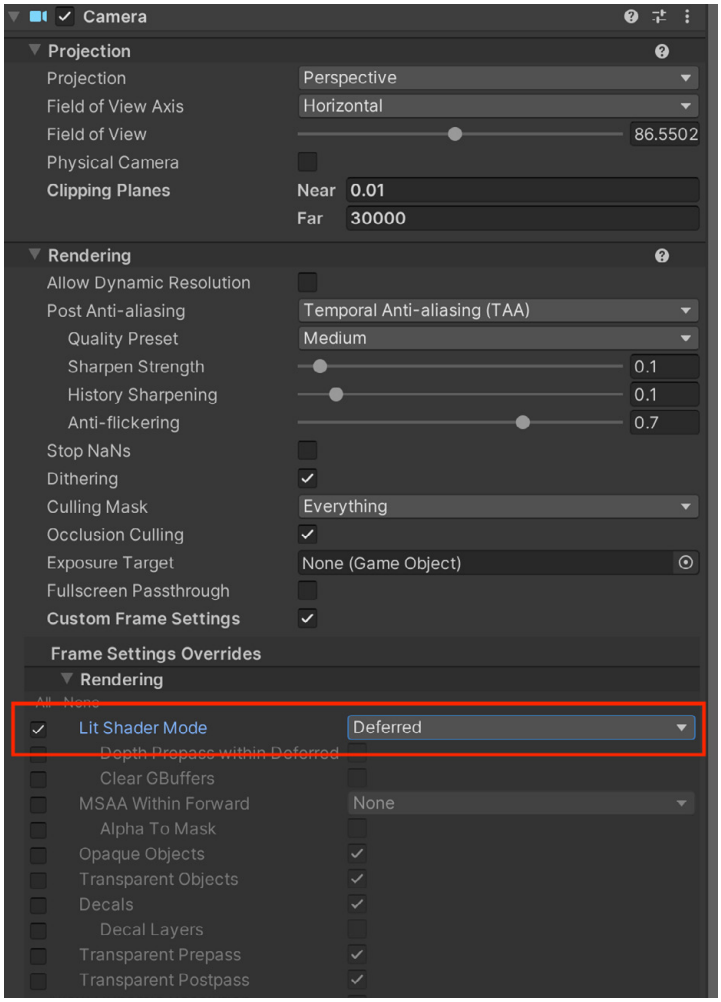
HDRP is flexible and also allows you to choose **Both**. This option lets you use one render path for most rendering and then override it per camera. However, this approach uses more GPU memory. In most cases, it is better to choose either Forward or Deferred.

— 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

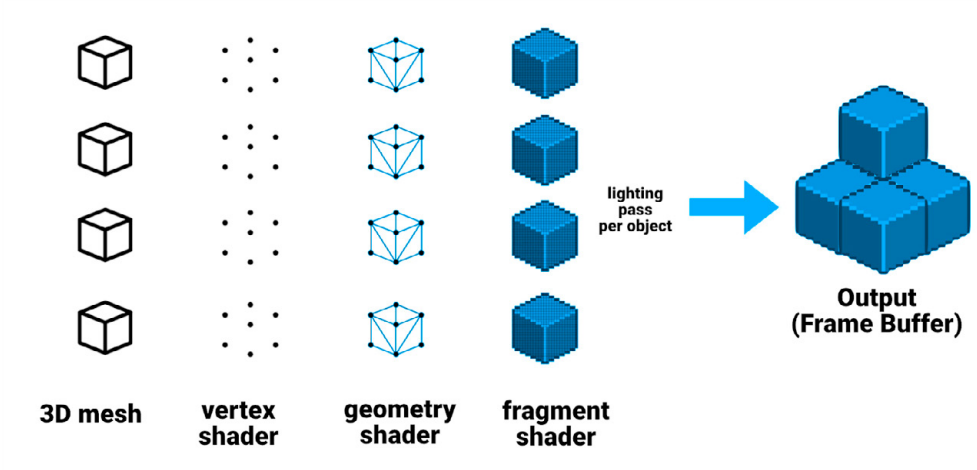
More about rendering paths

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.



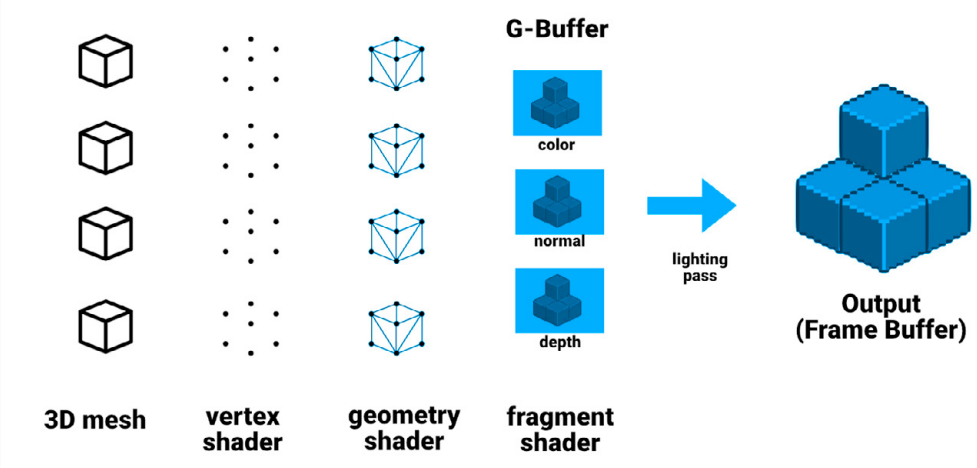
Forward rendering path

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.

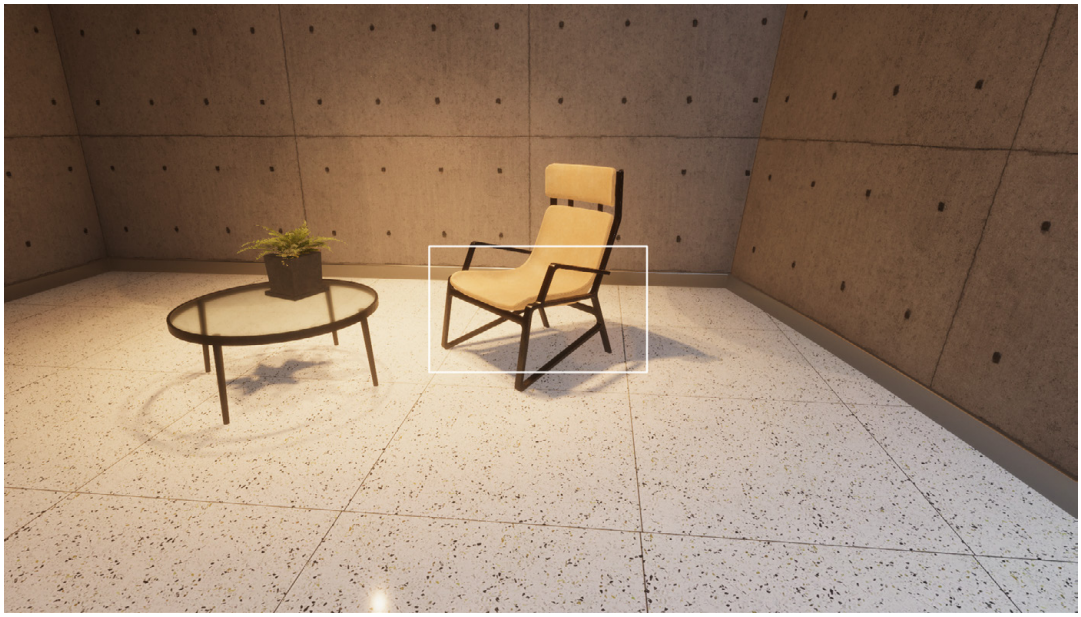
Multisample anti-aliasing (MSAA)

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.

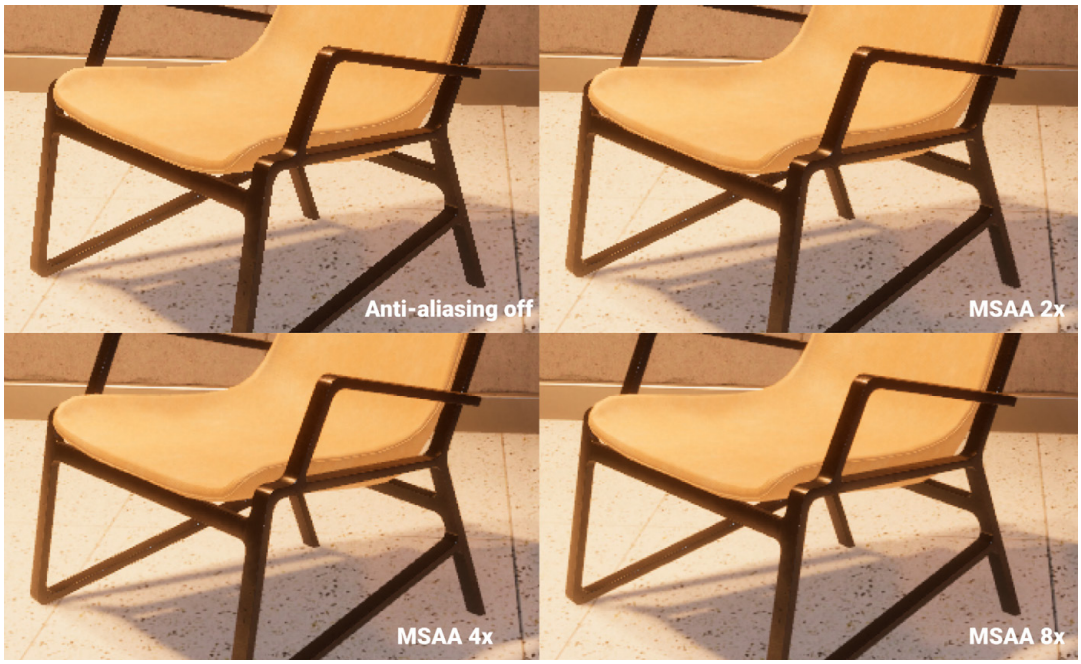
MSAA quality settings

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.

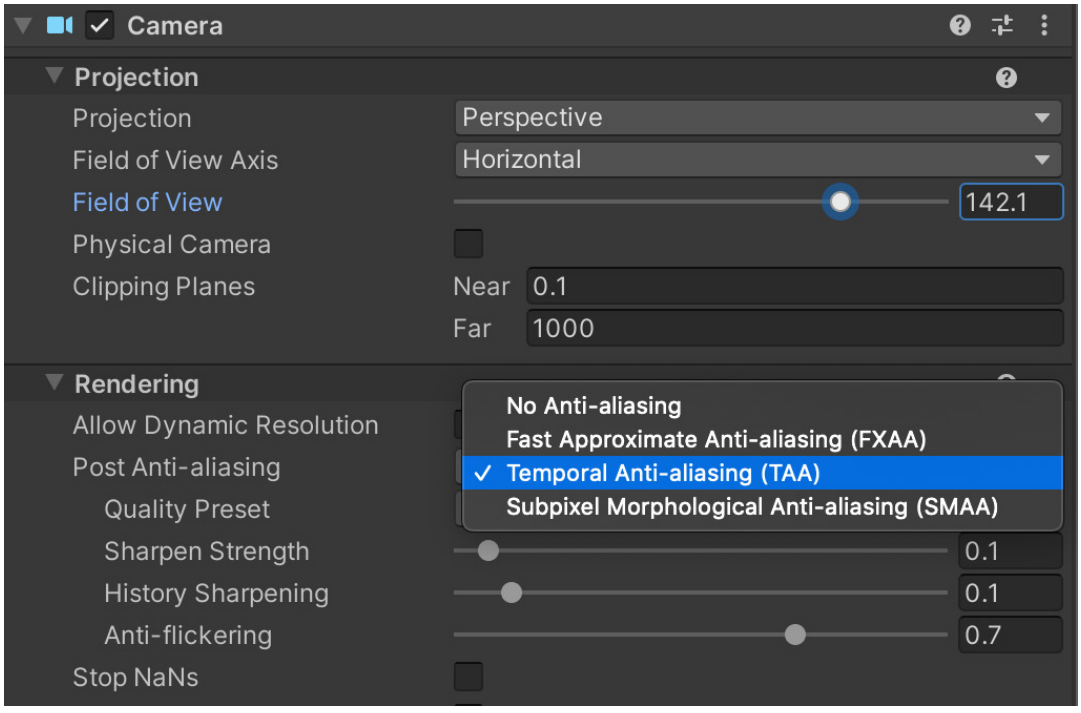
Post-processing anti-aliasing

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

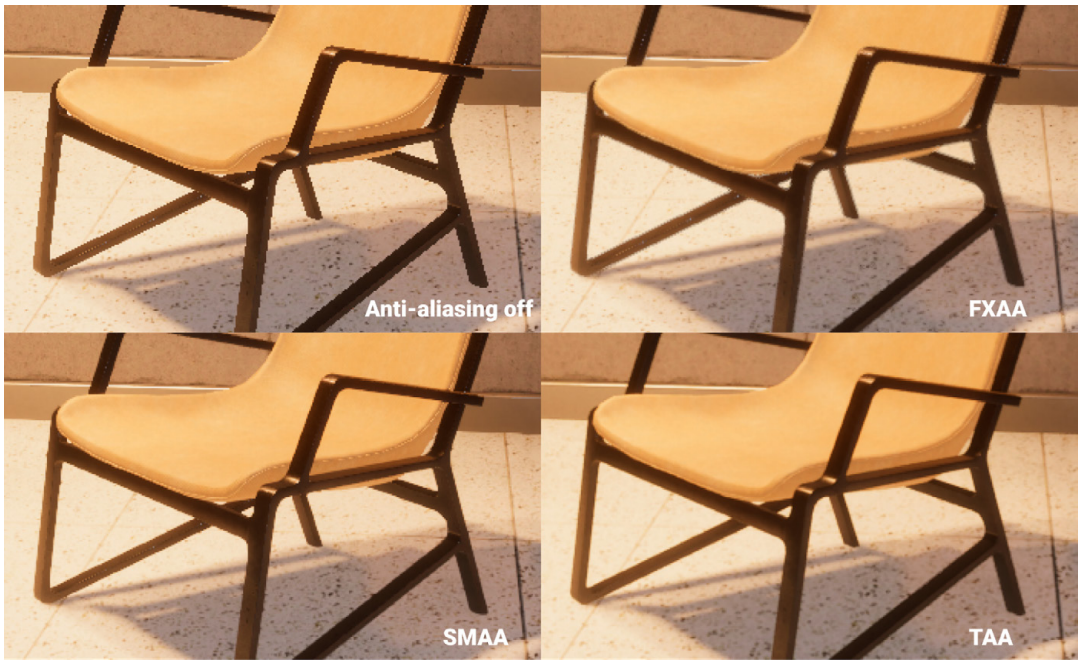
— **Temporal anti-aliasing (TAA)** combines information from past frames and the current frame to remove jaggies in the current frame. You must enable motion vectors in order for it to work. TAA generally produces great results, but it may create ghosting artifacts in some situations (e.g., a GameObject 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.

Post-processing

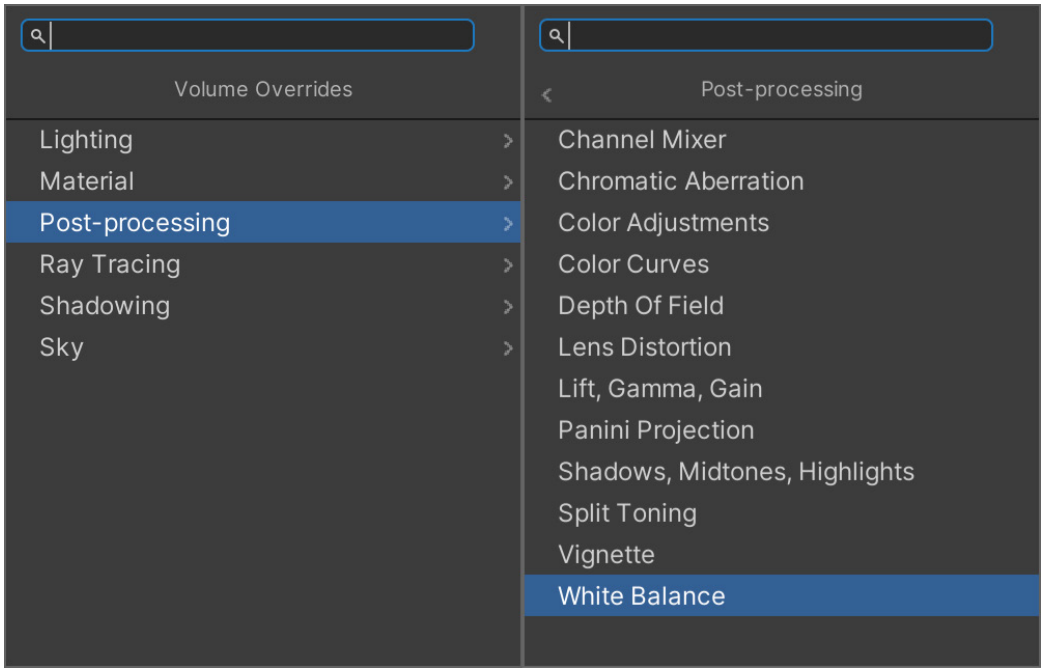
Modern high-end graphics would be incomplete without post-processing. While we can't always "fix it in post," it's difficult to imagine our rendered images without the filters and full-screen image effects that make them more cinematic. Thus, HDRP comes bundled with its own built-in post-processing effects.

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.

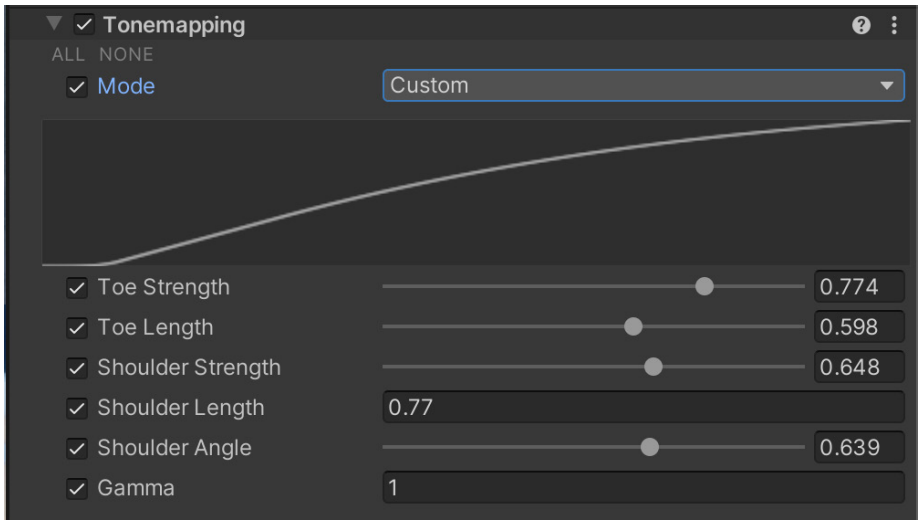
Refer to the Volumes in the HDRP Sample Scene (available in the Unity Hub) for example usage.

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

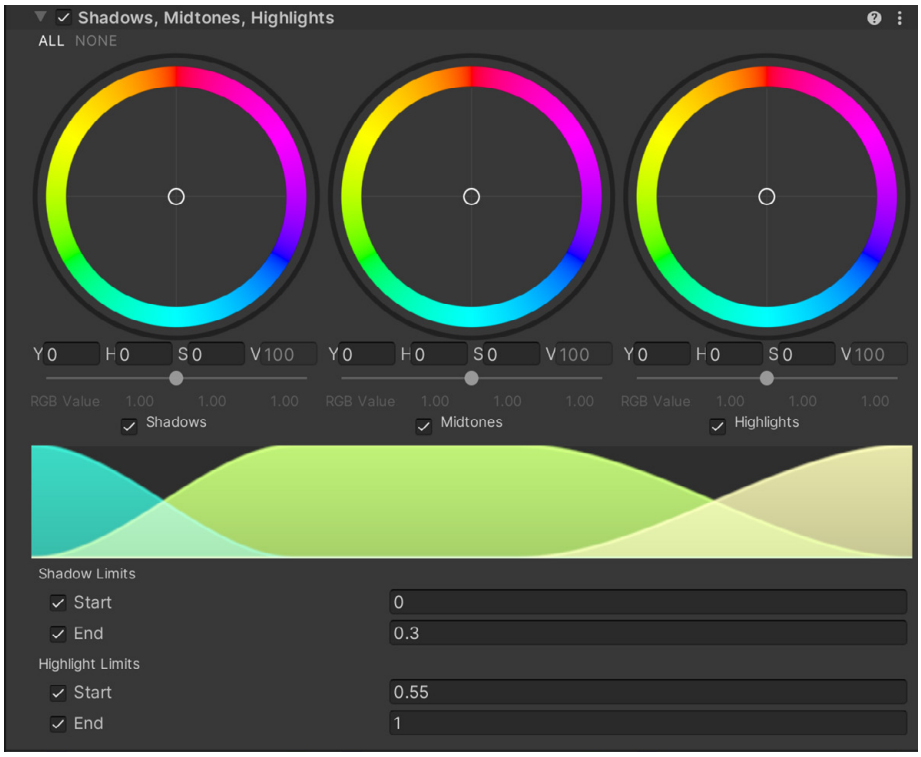
If you want a filmic look, set the **Mode** to the industry standard **ACES** (Academy Color Encoding System). For something less saturated and contrasted, select **Neutral**. Experienced users also have the option of choosing **Custom** and defining the tonemapping curve for themselves.



Tonemapping with a custom curve

Shadows, Midtones, Highlights

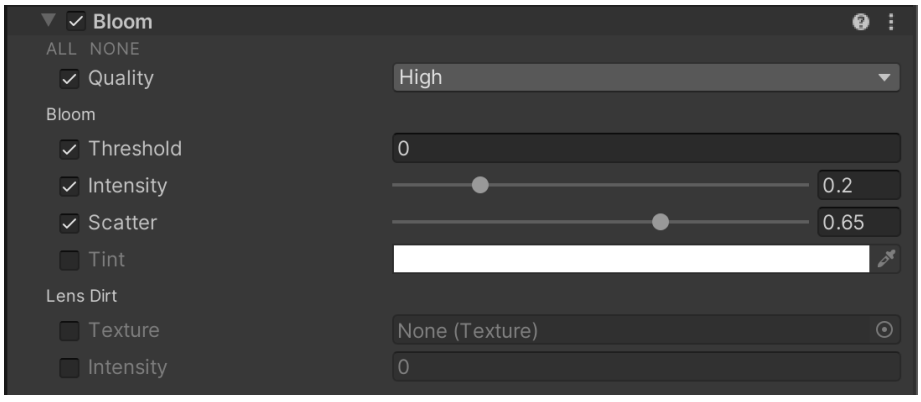
The Shadows, Midtones, Highlights override separately controls the tonal and color range for shadows, midtones, and highlights of the render. Activate each trackball to affect the respective part of the image. Then, use the **Shadow** and **Highlight Limits** to prevent clipping or pushing the color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



Shadows, Midtones, Highlights

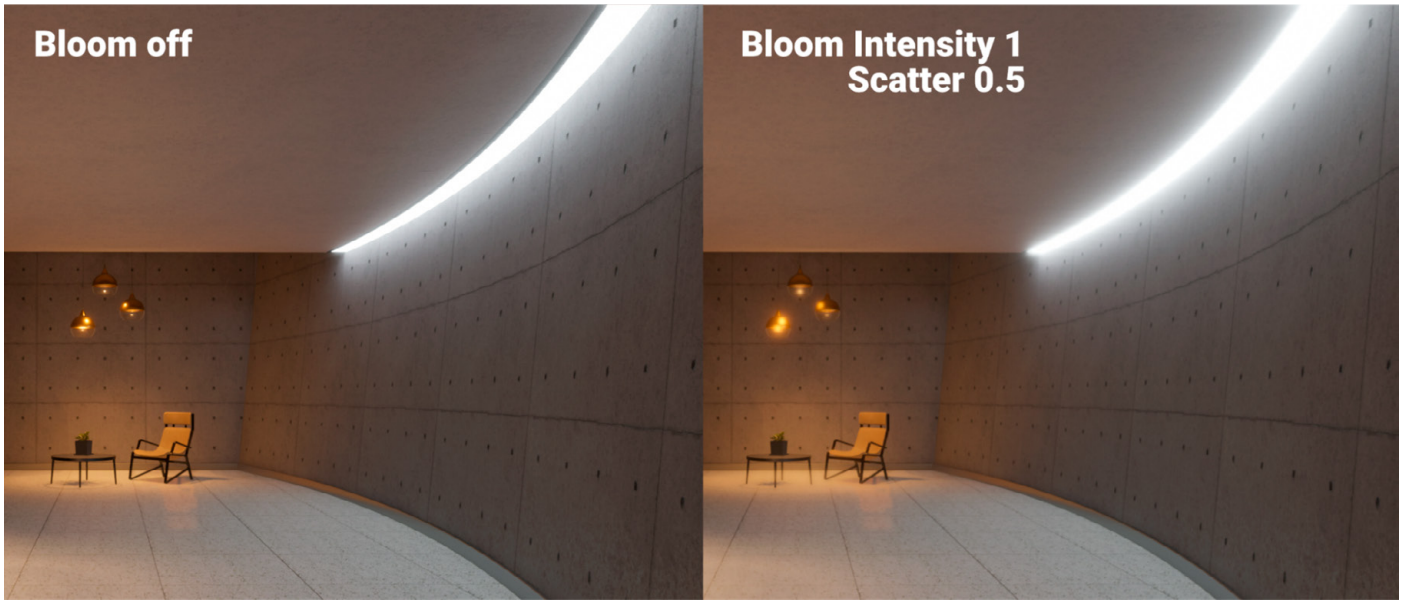
Bloom

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.

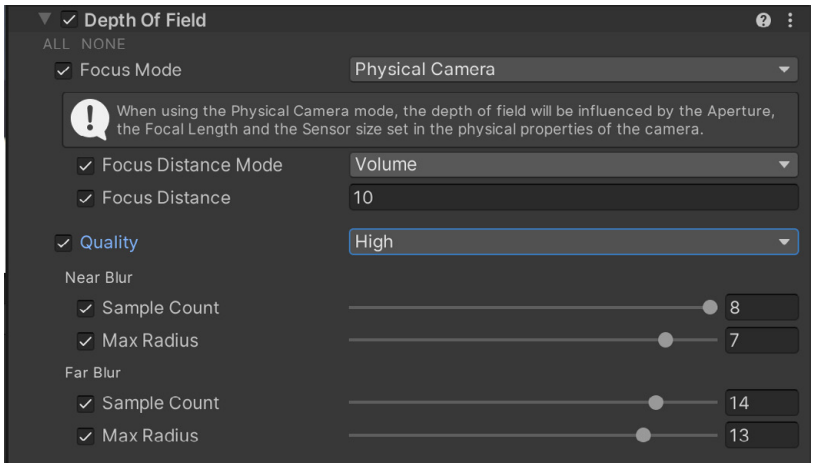
You can set focus distance from:

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).

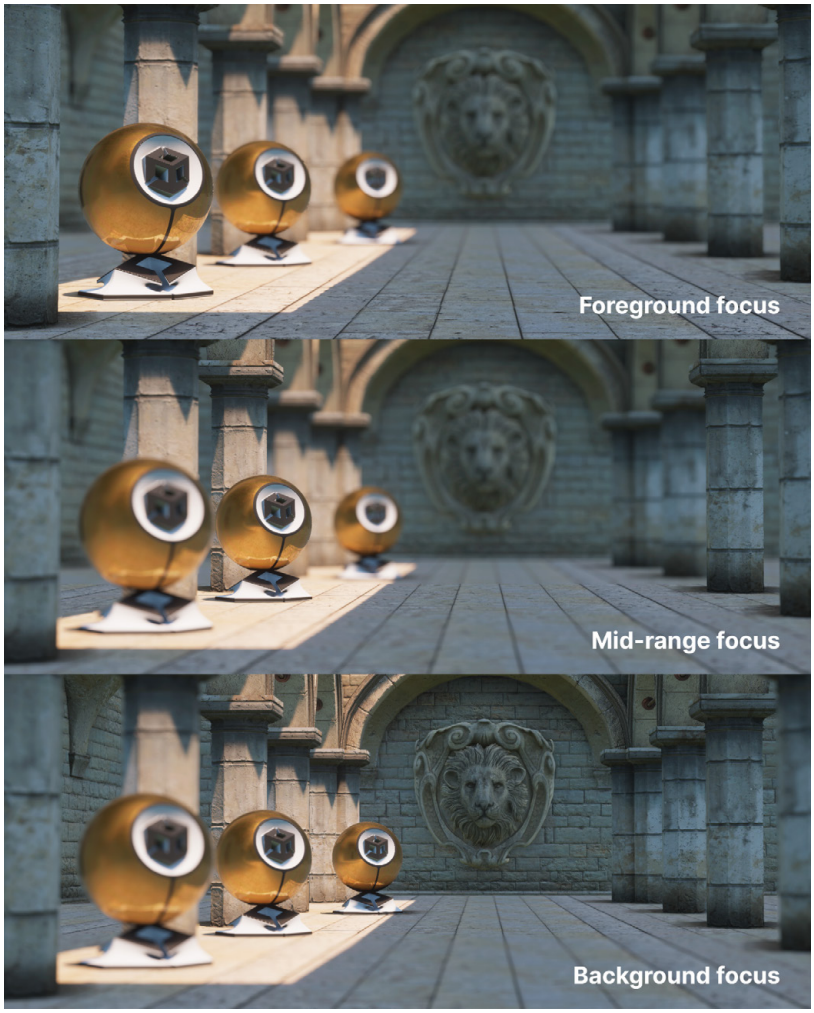
— Using a Cinemachine camera with the Volume Settings Extension: This lets you follow and autofocus on a target.

— 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).

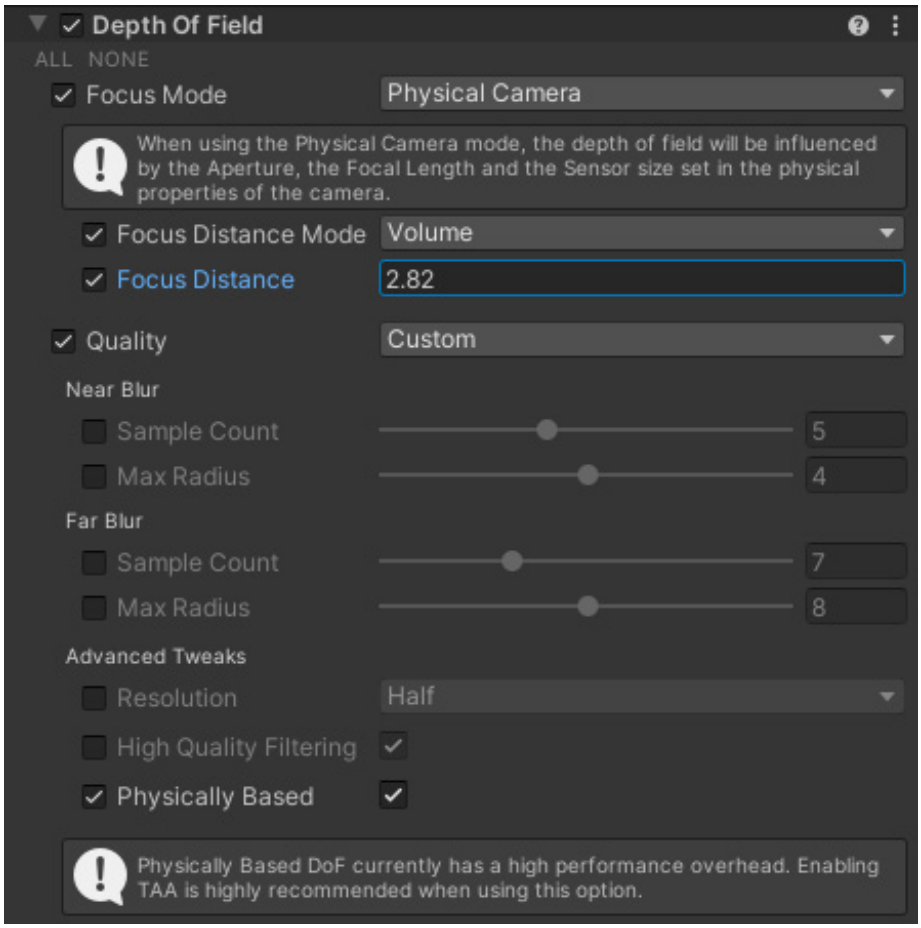


Depth of Field override



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**.



White Balance

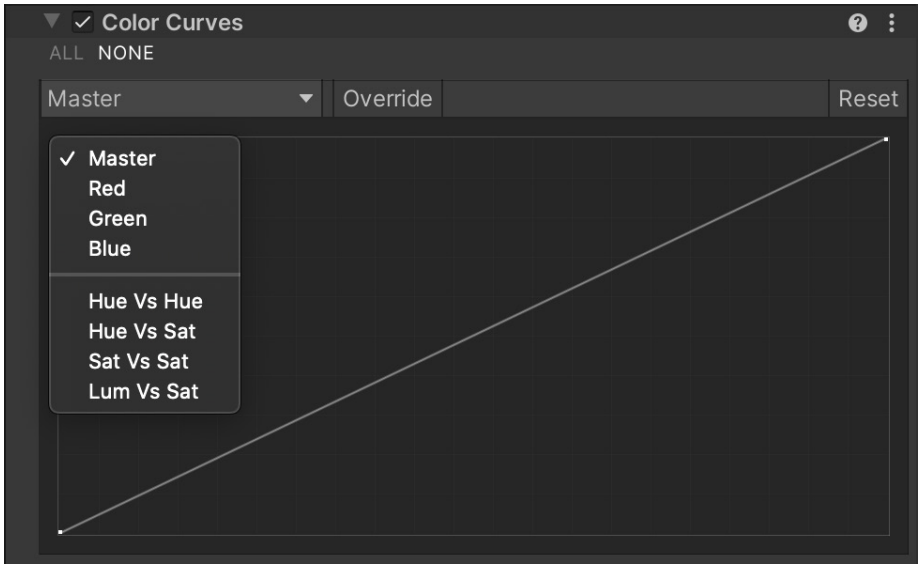
The White Balance override adjusts a Scene's color so that the color white is correctly rendered in the final image. You could also push the **Temperature** to shift between yellow (warmer) and blue (cooler). **Tint** adjusts the color cast between green and magenta.

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

White Balance

Color Curves

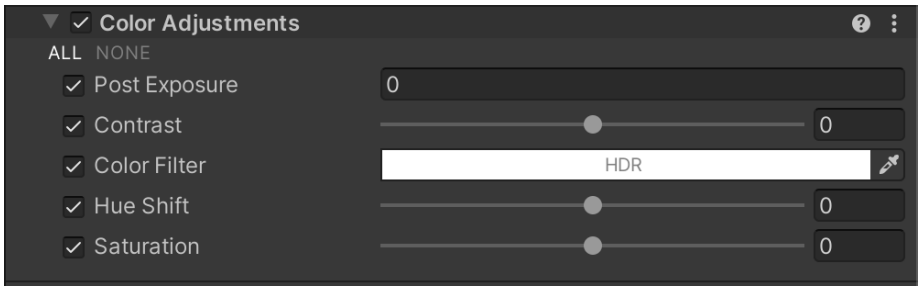
Grading curves allow you to adjust specific ranges in hue, saturation, or luminosity. Select one of the eight available graphs to remap your color and contrast.



Color Curves

Color Adjustments

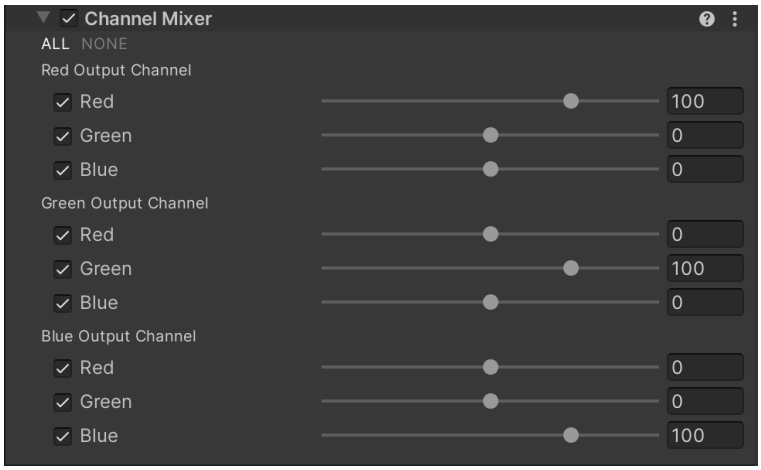
Use this effect to tweak the overall tone, brightness, hue, and contrast of the final rendered image.



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.

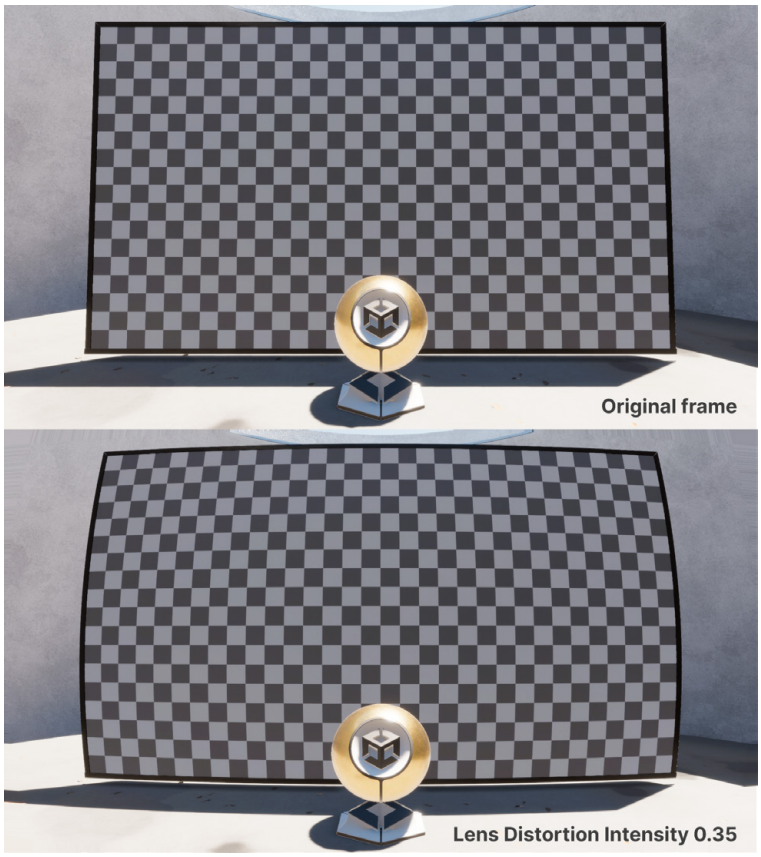


Channel Mixer

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.

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.



Lens Distortion warps the image in a radial pattern.

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 flares

A lens flare is an artifact that appears when bright light shines onto a camera lens. The flare can appear as one bright glare or as numerous colored polygonal shapes 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 grab the player's attention or change the mood of a setting or scene. Learn more about the physical aspect of lens flares in this Wikipedia article.

Lens flares from actual photographs (source: Wikipedia).

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

HDRP provides two complementary types of lens flares: the **Lens Flare (SRP)** component and **Screen Space Lens Flare** (new in Unity 6).

Lens Flare (SRP) component

The **Lens Flare (SRP)** component creates position-based flares from specific objects in your scene, typically light sources or bright elements. To use it, add the Lens Flare (SRP) component to any object and assign a Lens Flare Data asset.

Each Lens Flare Data asset consists of multiple elements, each with a shape such as a polygon, circle, or a custom image. You can layer these shapes together for complex effects. When attached to lights, they can inherit the light's color, making the same Flare asset reusable across different light sources.

The Lens Flare (SRP) controls the intensity, scale, and occlusion parameters.

The Lens Flare samples come with presets that range from realistic to stylized.

Screen Space Lens Flares

Unity 6 introduces a new post-processing technique, the Screen Space Lens Flare override (SSLF). This can generate flares from any bright surface, such as a bright specular highlight or an emissive mesh.

An SSLF is available as a Volume override (**Post-processing > Screen Space Lens Flare**). This type of flare effect works on bright areas in the scene, such as emissive surfaces, specular reflections, and onscreen lights. HDRP extracts these bright areas and applies effects like stretch, blur, and chromatic aberration to create the flares.

Screen Space Lens Flares use the same buffer as the Bloom effect to create several types of flares:

— Regular flares are distorted highlights from the bright areas of the screen.

— Reversed flares are regular flares flipped upside-down and reversed.

— Warped flares are regular flares transformed using polar coordinates, to mimic a circular camera lens.

— Streaks are flares stretched in one direction, to mimic an anamorphic camera lens.

To enable Screen Space Lens Flares, first activate them in your HDRP Asset and Frame Settings. Then add the **Screen Space Lens Flare** override to a Volume and set its Intensity above 0.

Note that if you already have an active Bloom override, it needs an intensity more than 0 for the Screen Space Lens Flares to appear.

You can combine a Screen Space Lens Flare with a Lens Flare (SRP) component for more control. Both systems are rendered as post-processing effects. The lens flares' occlusion integrates with environmental elements like clouds, volumetric fog, and water.

A Screen Space Lens Flare can add streaks.

Getting started

The best way to start adding your own lens flares is to install the samples from the Package Manager under High Definition RP. This will add a set of predefined Flare assets to the project. It also contains several test scenes to browse Lens Flares and help you build your own.

The predefined Flare Data assets recreate some commonly used flare effects. Explore the **Directional Light Sun** and **Point Light** scenes to take a closer look at how each lens flare is assembled:

— Each light source has its own Lens Flare (SRP) component.

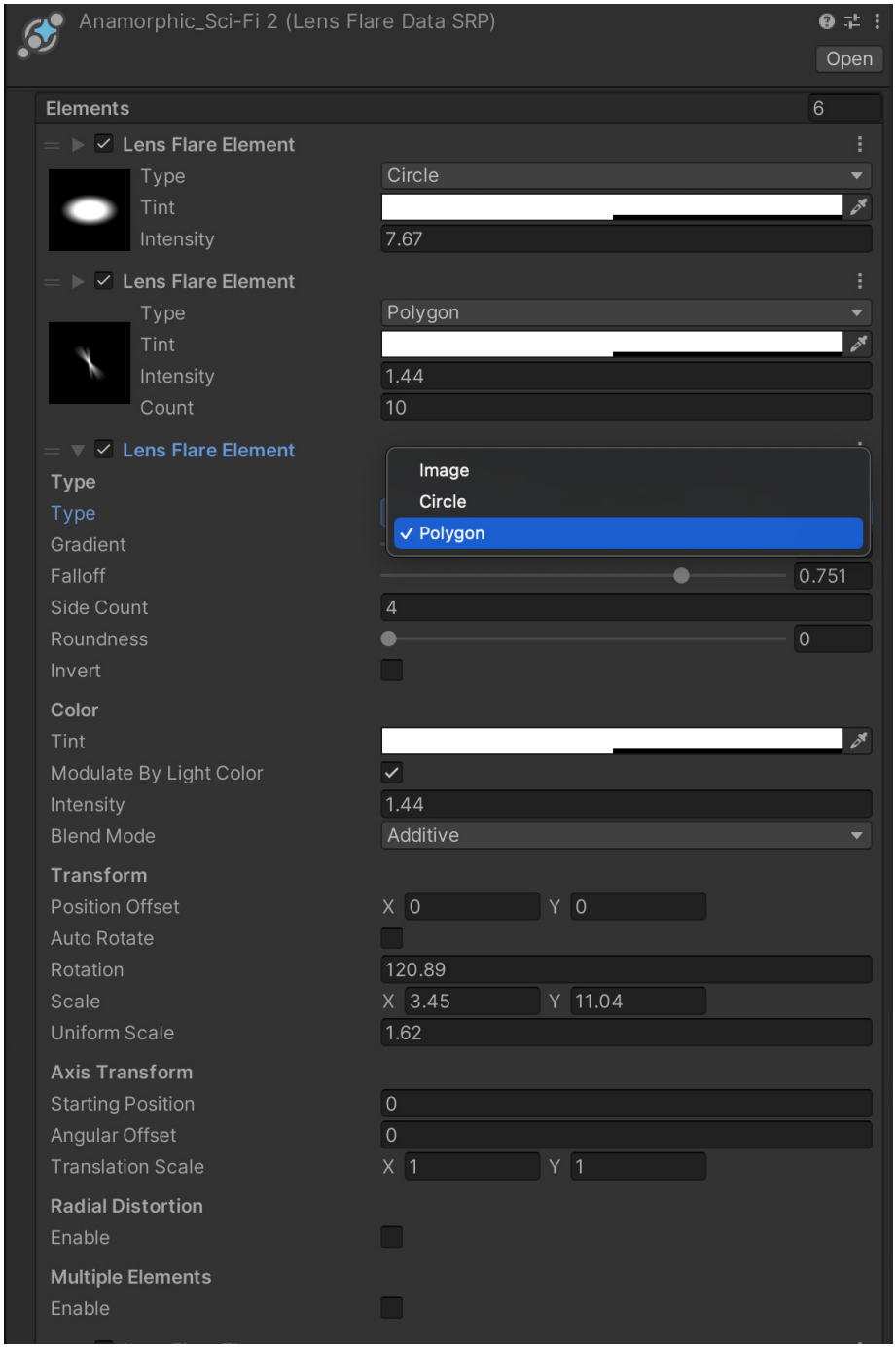
— This component references a Lens Flare data asset in the project.

You can browse the Lens Flares scene and help you build your own flare based on the presets, with the effect ranging from realistic to stylized.

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.

Combine lens flares to achieve the desired effect (intensity increased for illustration).

Watch this SIGGRAPH talk to learn more about how Lens Flares work.



Make your own library of custom flare effects.

Dynamic resolution

Upscaling improves performance by rendering a game at a lower resolution and then scaling it up to a higher resolution. This preserves visual quality while reducing GPU workload, allowing for higher frame rates without significant loss of detail.

Dynamic resolution is a feature in HDRP that adjusts the rendering resolution in real-time based on the GPU's performance. When enabled, HDRP allocates render targets for the maximum possible resolution and rescales the viewport to render at varying resolutions. At the end of each frame, HDRP upscales the rendered image to match the display's native resolution.

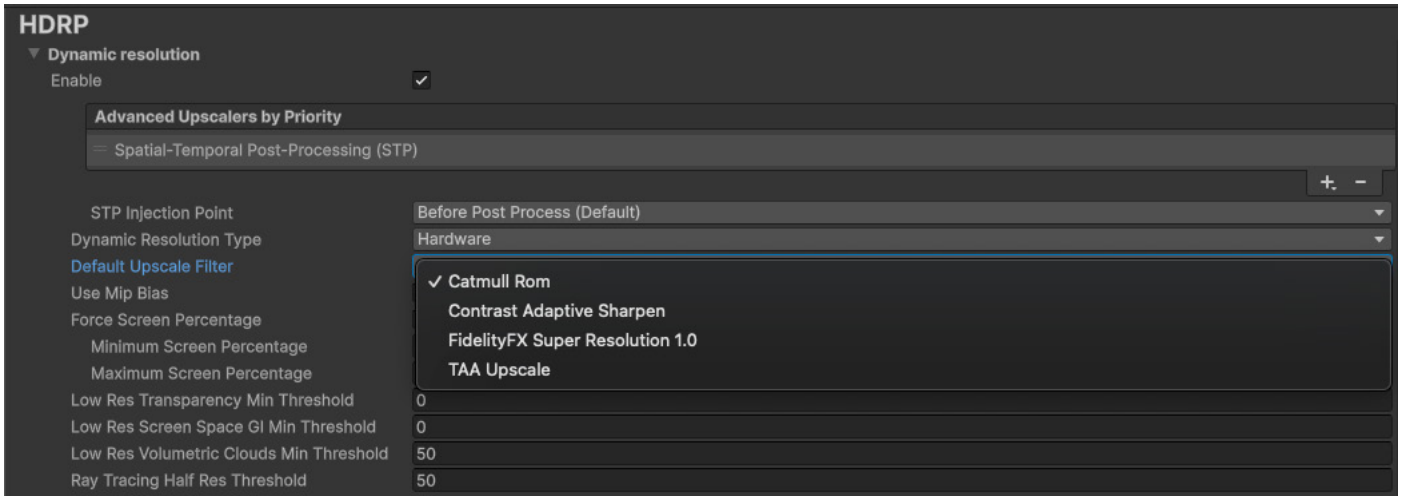
HDRP supports both hardware and software dynamic resolution. Hardware dynamic resolution is available on platforms like consoles, and PCs using DirectX 12, Vulkan, or Metal. If the platform doesn't support hardware dynamic resolution, HDRP defaults to software dynamic resolution.

Enable dynamic resolution in the HDRP Asset under **Rendering > Dynamic Resolution**. Then, in the Camera, check **HDRP Dynamic Resolution** under the **Rendering** settings.

HDRP now provides improved control over upscaling behavior across different hardware:

— **Advanced Upscalers by Priority**: This option allows you to specify a priority order for advanced upscalers (DLSS, FSR, STP). HDRP selects the highest-priority upscaler supported by the platform. If none are available, it falls back to the **Default Upscale Filter**.

— **Default Upscale Filter**: This sets the fallback upscaling method (e.g., TAAU, Catmull-Rom, CAS) when no advanced upscaler is available.



A new user interface specifies fallback preferences for upscaling.

Unity HDRP offers several upscale filter methods.

Catmull-Rom

Catmull-Rom uses four bilinear samples. This uses the least resources, but it can cause blurry images after HDRP performs the upscaling step. Catmull-Rom has no dependencies and runs at the end of the post-processing pipeline.

Contrast Adaptive Sharpen (CAS)

Contrast Adaptive Sharpen (CAS) uses FidelityFX (CAS) AMD. This method produces a sharp image with an aggressive sharpening step. Do not use this option when the dynamic resolution screen percentage is less than 50%. For information about AMD FidelityFX™ Super Resolution (FSR) and Contrast Adaptive Sharpening, refer to AMD FidelityFX Super Resolution (FSR).

Contrast Adaptive Sharpen (CAS) has no dependencies and runs at the end of the post-processing pipeline.

NVIDIA DLSS (for NVIDIA RTX GPU and Windows)

NVIDIA DLSS is natively supported for HDRP in Unity 6. NVIDIA DLSS is a suite of neural rendering technologies that uses AI to generate additional frames and construct images with quality rivaling that of 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, and the Unity documentation.

DLSS technology allows games like Nakara: Bladepoint by 24 Entertainment to run at 4K.

AMD FSR (cross-platform)

AMD FidelityFX™ Super Resolution (FSR) includes built-in support for HDRP. You can use FSR by enabling dynamic resolution in HDRP assets and cameras, then selecting **FidelityFX Super Resolution 2.0** 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 the AMD web page and on the Unity forum.

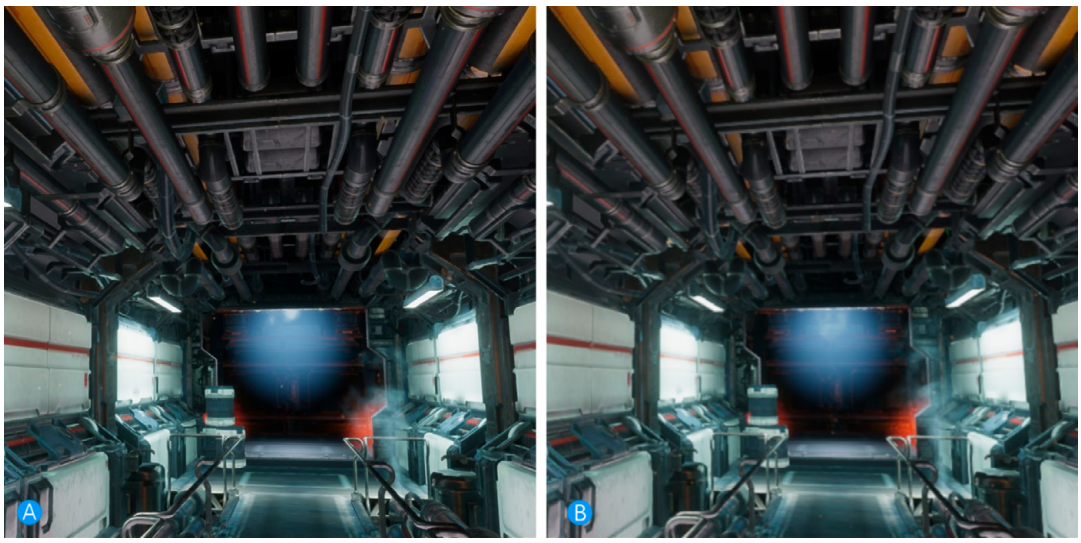
Try FSR technology in the *Spaceship* demo, available on Steam.

TAA Upscale (cross-platform)

Temporal Anti-Aliasing (TAA) Upscale enhances image sharpness by using temporal integration, running alongside normal anti-aliasing in HDRP. HDRP applies this upscale filter before post-processing and at the same time as TAA.

Since TAA Upscale is not compatible with other anti-aliasing methods, you must use TAA when enabling this feature. It also remains active when dynamic resolution is enabled, even at 100% screen resolution. You can set the upscale method in the HDRP Asset or dynamically adjust it via code.

For more information, see the documentation section Notes on TAA Upscale.



The image (A) on the left, using TAA Upscale, appears sharper and more defined. The image (B) on the right, using the Catmull-Rom upscale method, appears softer and less detailed.

Spatial-Temporal Post-Processing (cross-platform)

Spatial-Temporal Post-Processing (STP) is Unity 6's new built-in upscaler. STP renders at a lower resolution and then rebuilds a higher-resolution image using spatial and temporal data. This reduces GPU workload while maintaining visual fidelity.

STP uses an anti-aliasing technique that smooths out jagged edges using *both* spatial and temporal data. It can sharpen and refine details within a single frame (spatial) while stabilizing motion over multiple frames over time (temporal). This results in cleaner visuals with fewer artifacts.

STP reduces GPU workload while maintaining image quality.

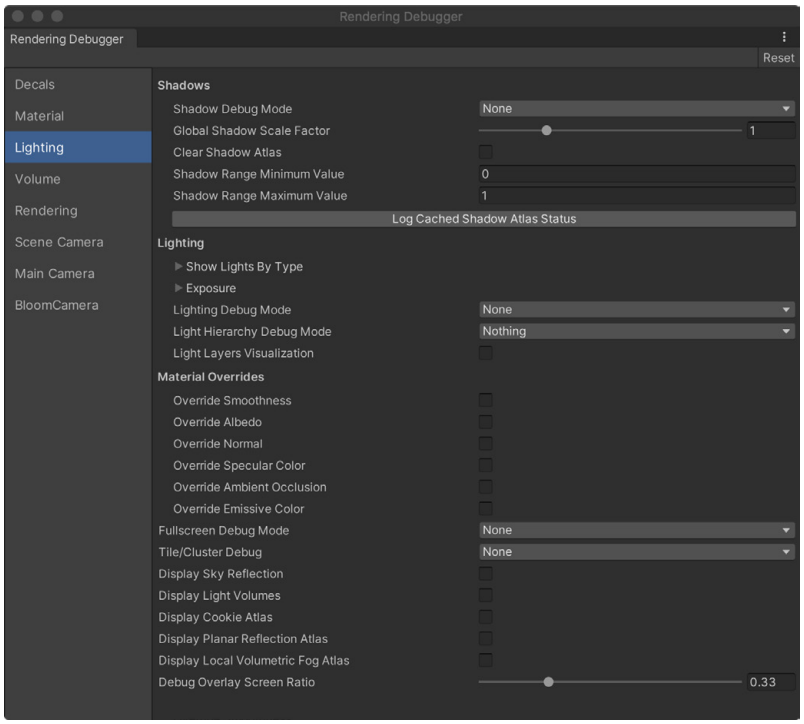
STP automatically adjusts its settings based on the platform. On high-performance devices (PCs and consoles), it uses higher quality image filtering and deringing logic during upscaling. On mobile devices, it opts for faster filtering methods to minimize performance impact while delivering visual quality comparable to DLSS2, FSR2, or XeSS.

By adjusting **Render Scale** in the **HDRP Asset**, you can fine-tune STP's balance between performance and quality based on your target platform.

Note that STP requires Shader Model 5.0 and only works with TAA (Temporal Anti-Aliasing). STP does not support dynamic resolution unless hardware support is available. If hardware dynamic resolution is unavailable, Render Scale must be set to a fixed value.

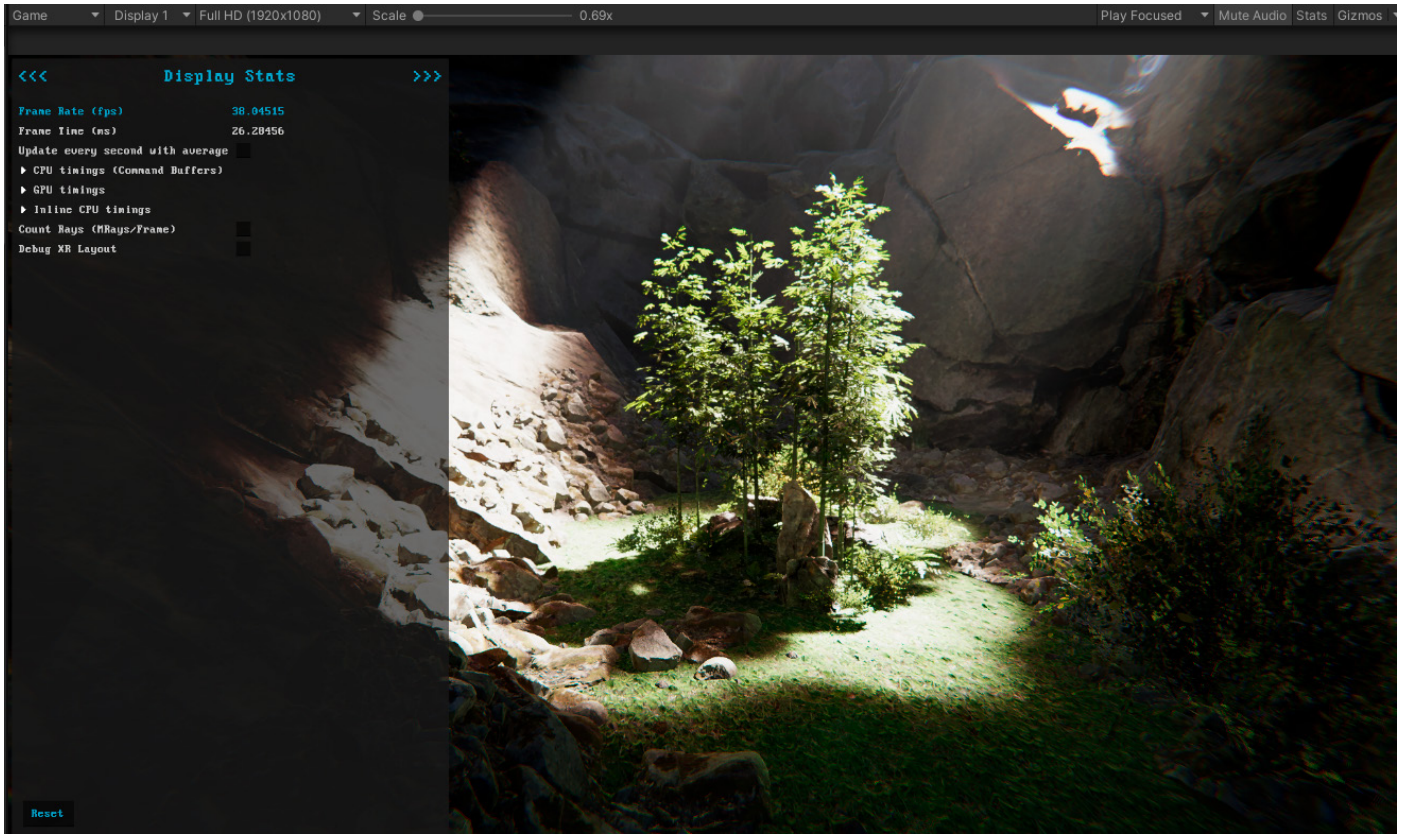
Rendering Debugger

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.



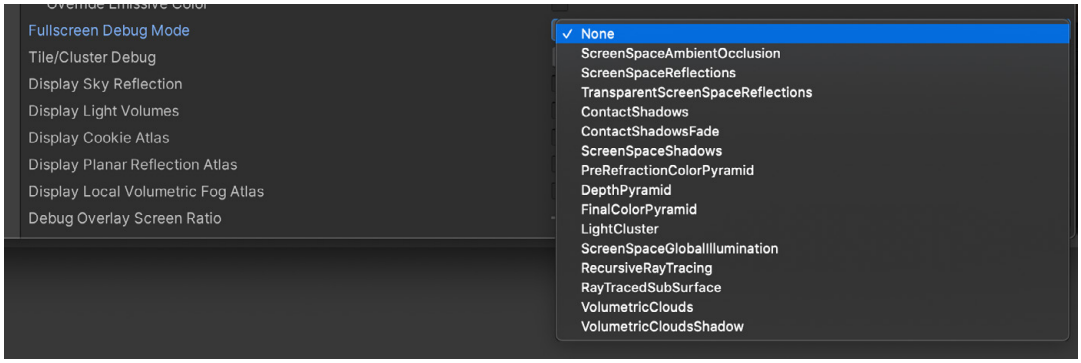
Rendering Debugger

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

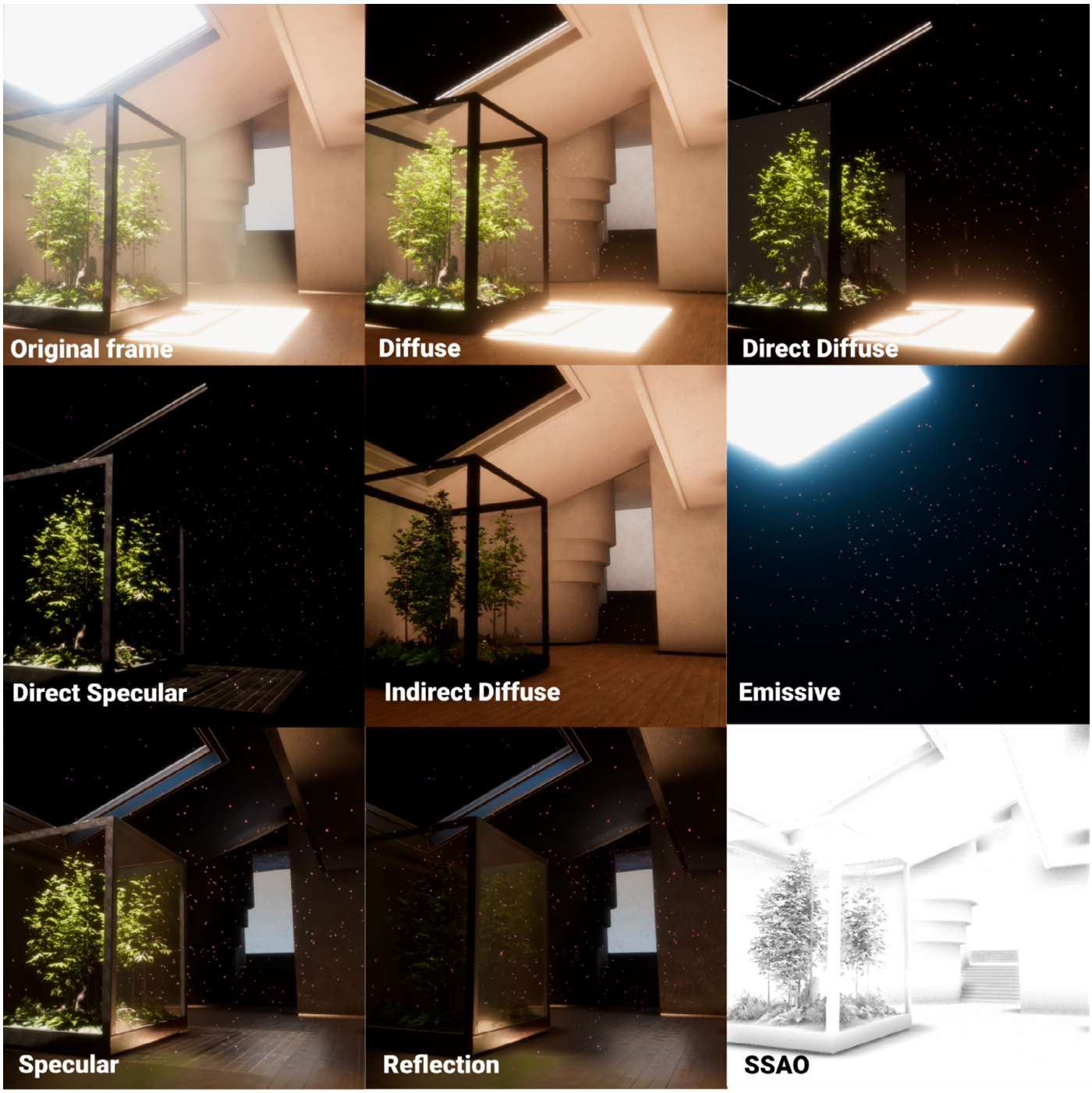
The Debugger can help you troubleshoot a specific rendering pass. On the Lighting panel, you can enter **Fullscreen Debug Mode** and choose features to debug.



Fullscreen Debug Mode options

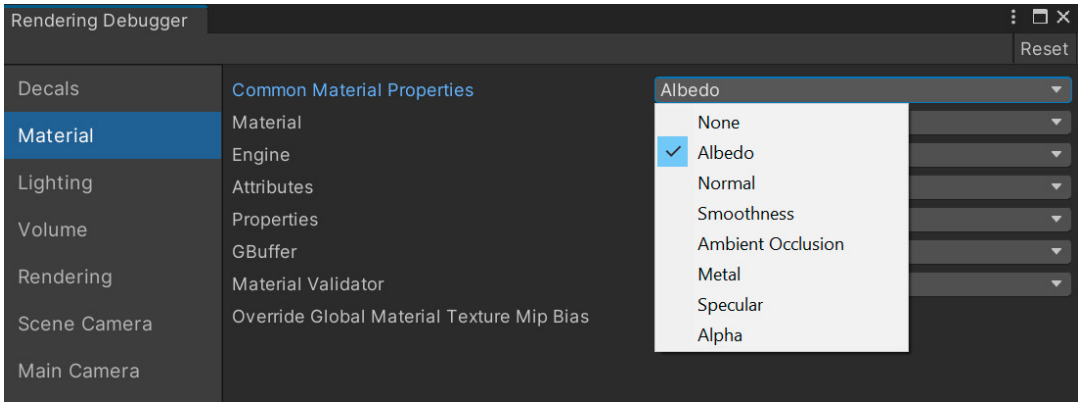
These Debug modes let you play “pixel detective” and identify the source of a specific lighting or shading issue. The panels on the left can show you vital statistics from your cameras, Materials, Volumes, and so on, to help optimize your render.

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.

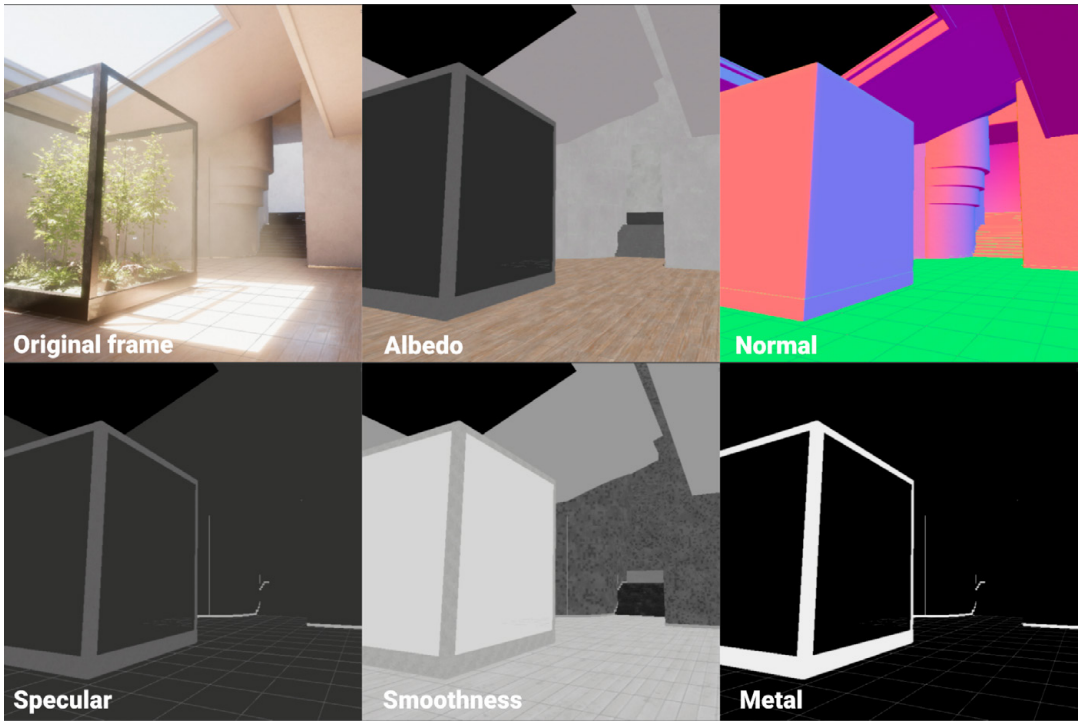


Lighting Debug Mode or Fullscreen Debug Mode can help you understand the sources of illumination in your scene.

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

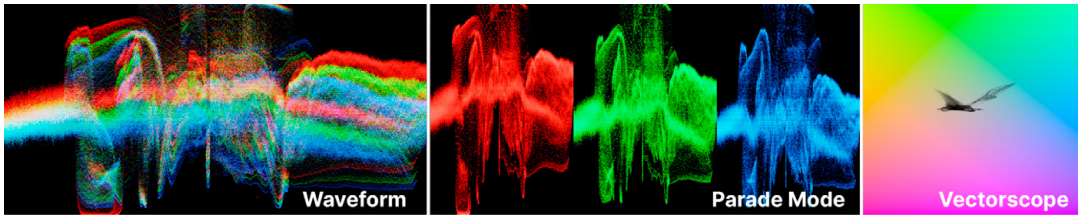
Unity includes a set of industry-standard color monitors that you can use to control the overall look and exposure of a scene. You can find these monitors in the **Rendering Debugger** window (**Windows > Analysis > Rendering Debugger**) in the Rendering tab.

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.

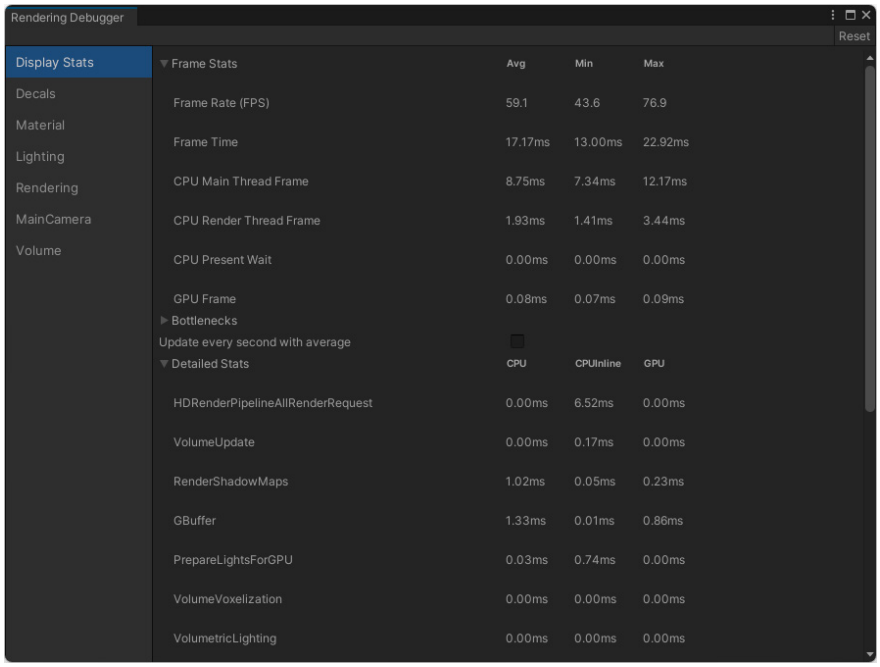
Runtime Frame Stats

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).



The Runtime Frame Stats when in Play mode

See the Render Pipeline Debugger documentation for complete details.

Support for HDR displays

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. Unity 6 now provides cross-platform HDR output support in both the Editor and Standalone Players.

As high dynamic range (HDR) displays are becoming increasingly available, Unity can take advantage of these displays and reproduce images with better contrast (highlights/shadows) and color saturation.

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.

Unity 6 expands HDR display support across all render pipelines and supported platforms, enabling native color space rendering and HDR tone mapping. HDR output allows for 10-bit or 16-bit color depth, with a color range up to 68 billion colors.

The Editor and Standalone Players now provide full HDR tone mapping and display support across all SRP render pipelines and compatible platforms. HDR rendering uses a floating-point framebuffer to store a wider range of color values. This improves lighting, post-processing, and overall visual accuracy compared to a standard definition range render.

Once you've implemented HDR into your project, you can also incorporate HDR calibration menus using the new HDR calibration template. This can help you avoid common challenges like under and over exposure and ensuring that text on screen remains legible.

To activate HDR output, navigate to **Project Settings > Player** > **Other Settings** and enable the following settings:

— **Allow HDR Display Output**

— **Use HDR Display Output**

Only enable **Use HDR Display Output** if you need the main display to use HDR Output.

HDR Output will be active only in Game View and in Player. Currently the feature is only available on DirectX 12.

With HDR you can turn on the option to take advantage of better quality image render outputs from URP on HDR displays. As a result, you can present final images with colors and contrasts that mimic natural lighting conditions better on these devices.

In addition to the existing desktop and console support that became available in Unity 2022, Unity 6 introduces mobile support for the following platforms:

— iOS players (iOS 16+, iPadOS 16+)

— Android players using Vulkan and GLES (Android 9+, depending on device capabilities)

Some common mobile devices that provide HDR display support include iPhone X and newer, Samsung Galaxy S10 and newer, Galaxy Note 10 and newer, and Galaxy Tab S6 and newer.

For more details, see this blog post and refer to High Dynamic Range (HDR) Output for additional information.

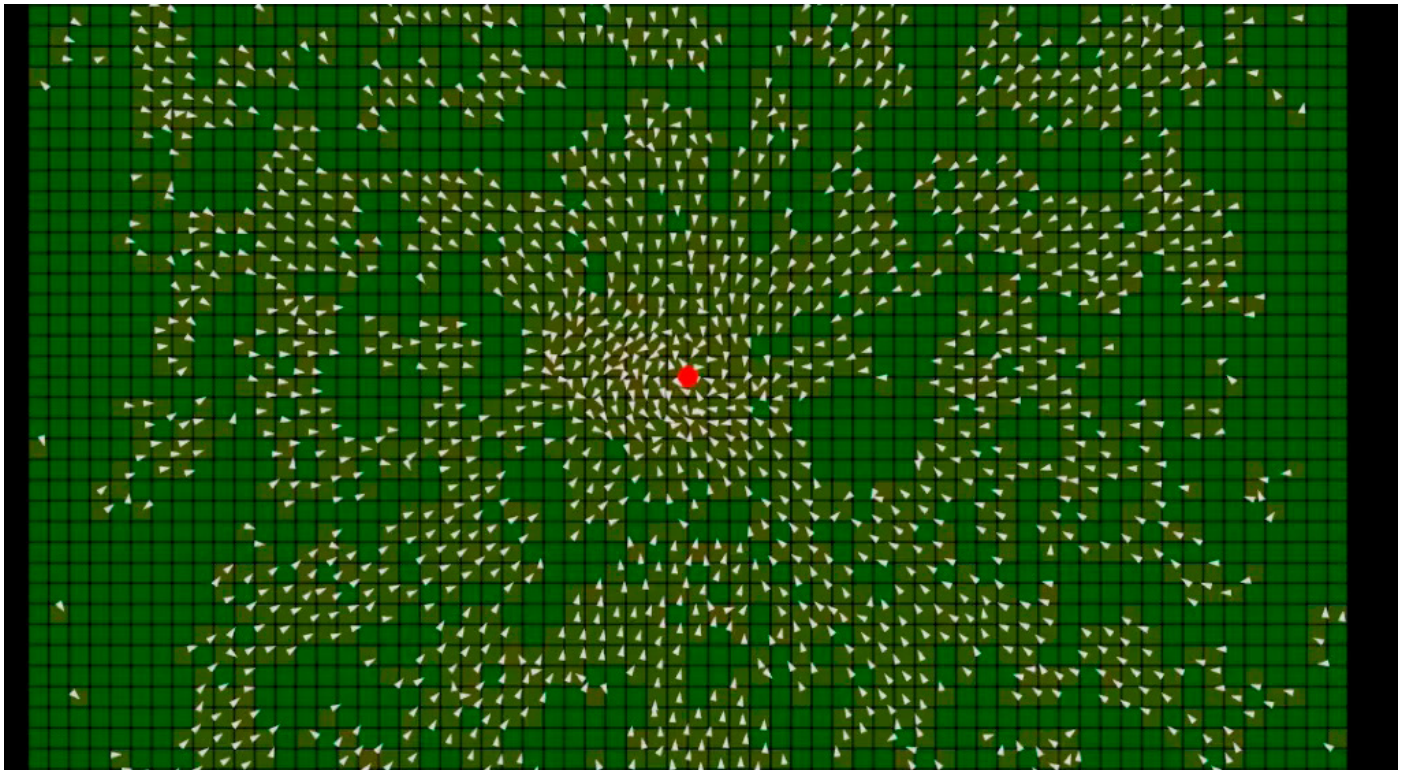
VFX Graph support

Unity 6 also includes VFX Graph features that work with the HDRP, for added flexibility and advanced visual effects. VFX Graph complements HDRP by enabling artists to create real-time visual effects that leverage HDRP's rendering features.

Custom HLSL Block and Operator

Unity 6 includes a new block and operator that you can use to write custom HLSL code directly in a VFX Graph. You can use this to script complex behavior in a single node. You can embed custom HLSL code in the node or load it from an external file to reuse the code.

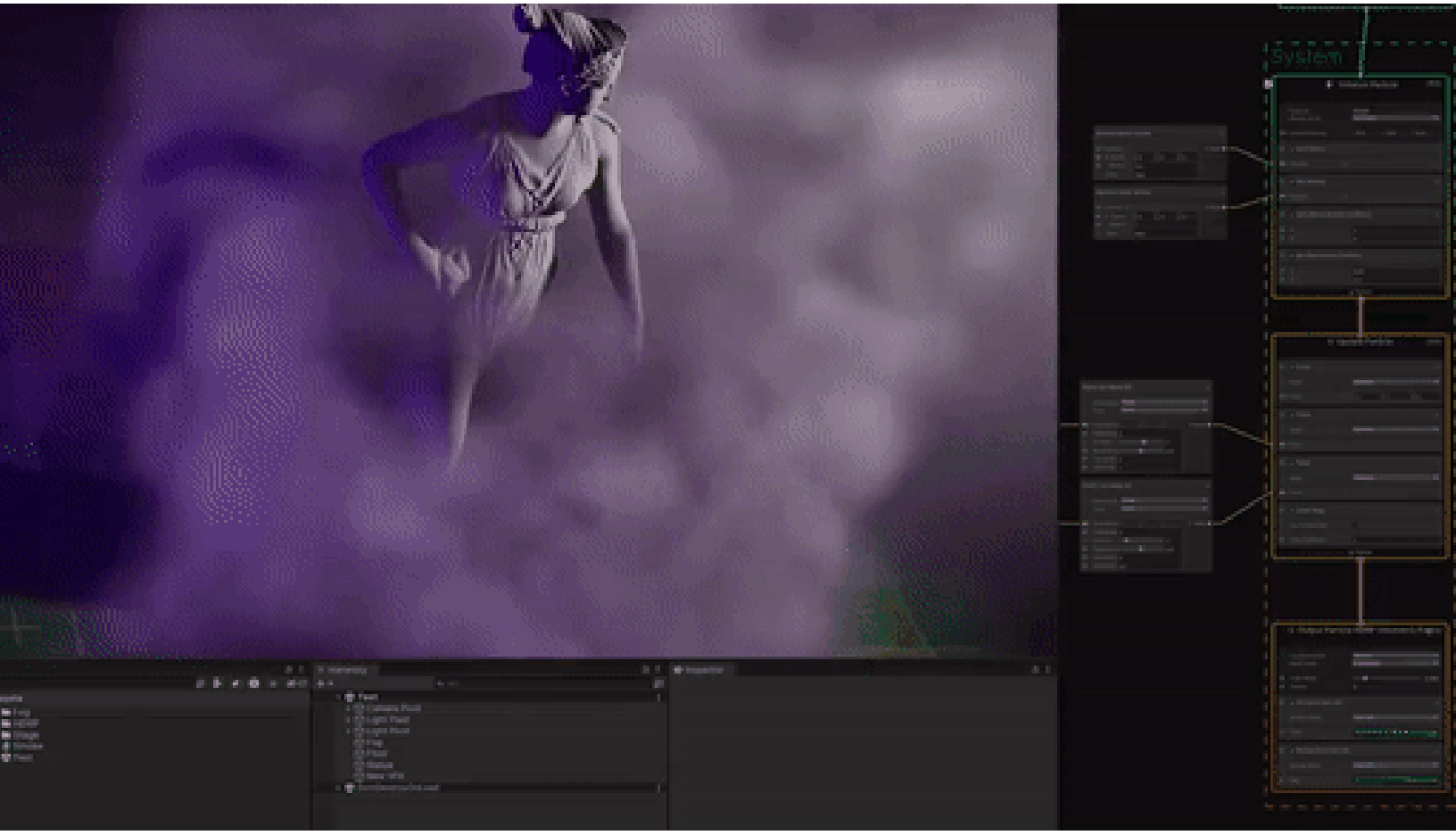
Use this to write custom shaders or extend Unity's built-in capabilities for unique effects. For instance, you can implement custom flocking behaviors using neighbor search algorithms or use buffer readbacks to sync visual effects with audio triggers.



An effect created with the HLSL Block.

Volumetric Fog Output

The **Output Particle HDRP Volumetric Fog** Context samples Local Volumetric Fog in HDRP, allowing VFX Graph to inject fog directly into volumetric light. The result is improved interaction between VFX Graph particles and scene lighting for added realism. Use it to create dynamic fog effects, realistic clouds, smoke, or fire.



The Output Particle HDRP Volumetric Fog output node

Ray tracing support

VFX Graph particles in Unity 6 now support the following ray tracing features in HDRP:

— Ray-traced Reflections

— Refraction, Shadows

— Ambient Occlusion

— Global illumination

You can enable ray tracing with VFX Graphs that use quads, triangles, and octagons. To use ray tracing in an HDRP scene, refer to Getting started with ray tracing. Note: VFX Graph does not support ray tracing with meshes or strips.

To explore how you can integrate VFX Graph into your HDRP projects, see the e-book, *The Definitive Guide to Creating Advanced Visual Effects in Unity (Unity 6 Edition)*.

Optimizations

Complex game environments can be demanding on performance, so optimizing the render pipeline is essential for your application. If your game uses modular elements or prefabs, you can reduce CPU load and improve rendering efficiency using various techniques.

Scriptable Render Pipeline Batch

The **SRP Batcher** optimizes draw calls by reducing the CPU overhead required to prepare and dispatch them. Instead of reducing the *number* of draw calls, the SRP Batcher minimizes *render-state changes* between them.

The SRP Batcher groups bind and draw GPU commands into **SRP batches**, improving performance for materials that share the same shader variant.

For optimal performance, each SRP batch should contain as many `bind` and `draw` commands as possible. Using fewer shader variants helps maximize batching efficiency while still allowing multiple materials to use the same shader.

See this documentation page for more information about how the SRP Batcher works.

BatchRendererGroups

BatchRendererGroup (BRG) is a rendering API in Unity designed for efficiently handling large numbers of objects in projects using the Scriptable Render Pipeline (SRP) and the SRP Batcher.

Instead of relying on traditional GameObject-based rendering, BRG allows direct control over rendering data. This makes it ideal for DOTS Entities, procedural environments, and custom terrains where traditional GameObjects would be too costly.

BRG optimizes performance by batching multiple objects into a single draw call, reducing CPU overhead. It keeps persistent GPU buffers, storing and updating object data without needing frequent CPU intervention.

To use BRG, a project must enable **SRP Batcher** compatibility, retain BRG shader variants (in **Project Settings > Graphics**), and allow unsafe code (via **Player Settings**).

BRG works well in scenes that require large-scale instancing, including dense vegetation, rendering many objects on-screen, or dynamic LOD handling. It integrates with the new GPU Resident Drawer.

GPU Resident Drawer

The GPU Resident Drawer in Unity 6 improves the rendering efficiency of GameObjects. This new feature uses the BatchRendererGroup API, which can batch and render MeshRenderers via a special fast path that handles better instancing. By reducing the per-object processing costs, the GPU Resident Drawer can alleviate CPU bottlenecks in scenes with heavy draw calls.

This feature is compatible with Scriptable Render Pipeline (both URP and HRDP) and requires graphics APIs that support compute shaders (not available on OpenGL/GLES).

Performance gains using the GPU Resident Drawer increase with scene complexity. The more detail in your scene – and more instanced objects – the bigger the benefit.

To enable the GPU Resident Drawer:

— Navigate to **Project Settings > Graphics**, and set **BatchRendererGroup Variants** to **Keep All**.

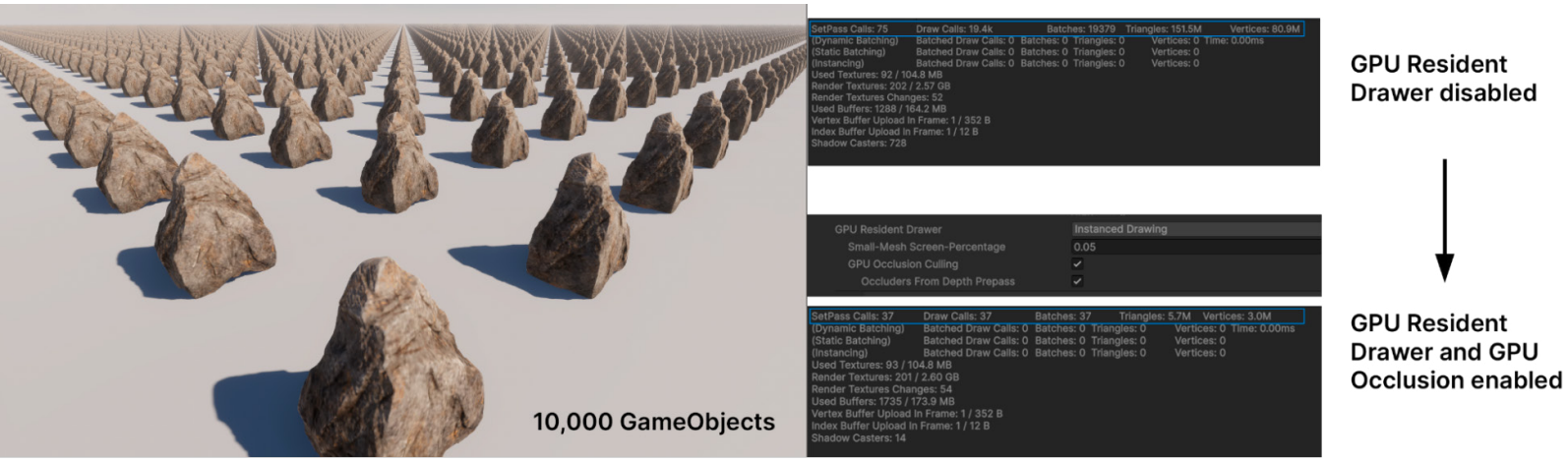
— Then, locate the **HDRP Render Pipeline Asset**. Find the **GPU Resident Drawer Mode**, and set it to **Instanced Drawing**.

— Make sure that the GameObject's material has **Enable GPU Instancing** checked.

To optimize the GPU Resident Drawer, disable **static batching** for the project (**Edit > Project Settings > Player**) or per instance in the Inspector. If static batching is enabled, the GPU Resident Drawer won't take effect.

Be aware that build times may increase due to compiling additional shader variants when using the GPU Resident Drawer. This feature works with standard MeshRenderers and does not handle skinned mesh renderers, VFX Graphs, particle systems, or other effects renderers.

Once you've enabled the GPU Resident Drawer, this can save draw calls for scenes where multiple GameObjects share the same mesh. Here is a comparison in the Profiler for a Prefab instanced 10,000 times:

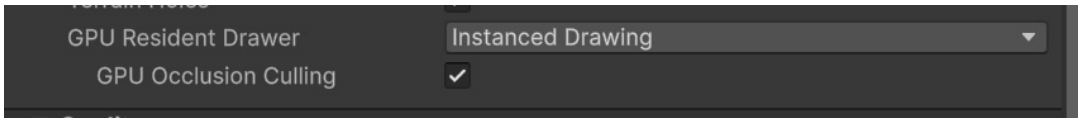


Compare stats with and without the GPU Resident Drawer.

GPU Occlusion Culling

If your scene is using the GPU Resident Drawer, you can also take advantage of **GPU Occlusion Culling** to remove any GPU instances that don't actually appear on-screen. This can improve GPU performance in scenes with heavy overdraw with just one setting.

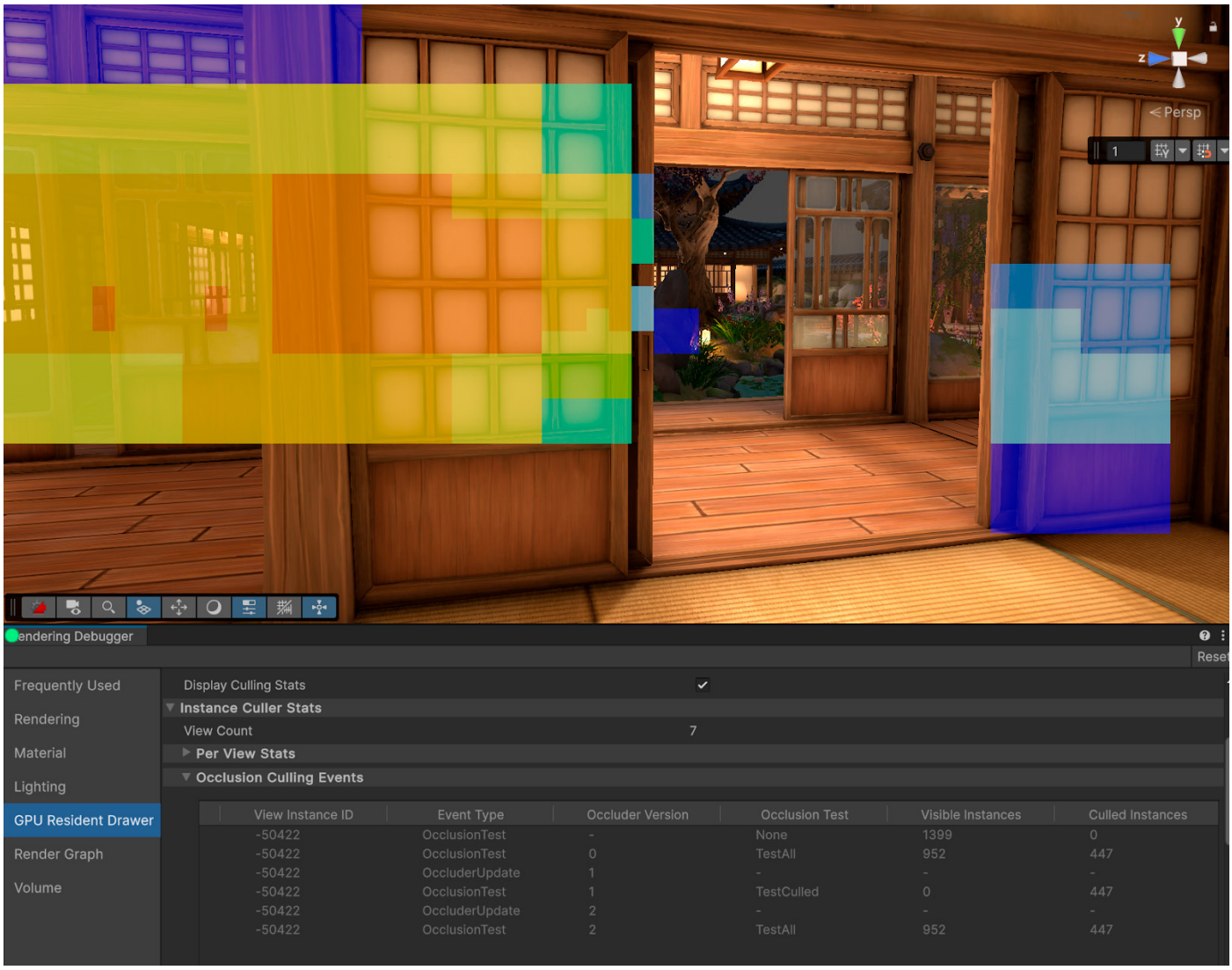
To enable it, make sure that you've activated the GPU Resident Drawer Mode and then locate the HDRP Asset. Navigate the **Rendering** section and enable **GPU Occlusion Culling.**



Enable GPU Occlusion Culling.

To see if GPU occlusion culling is effective for your scene go to **Window > Analysis > Rendering Debugger**, and select **GPU Resident Drawer > Occlusion Test Overlay**.

The RenderDebugger colors the screen with a heat map that shows which parts of the scene benefit the most. Areas shown in **blue** indicate few culled instances, while **red** represents many culled instances, with a gradient of colors in between.



The GPU Resident Drawing also works in URP.

See our other guidebook, *Introduction to URP for advanced creators*, for more details.

DX12 Graphics Jobs Editor support

Graphics Jobs are now supported in the Unity Editor when using DX12. This improves Scene View and Play Mode rendering, for improved performance when working in the Editor with complex environments.

To enable this behavior, set **DirectX 12** as the **Graphics API**. Then, check **Graphics Jobs** in the **Player** settings and enable **Allow Graphics Jobs in Editor** in the **Jobs** settings (**Preferences/ Settings > Jobs**).

Modern graphics APIs like DirectX 12 (DX12) improve CPU utilization by enabling multi-threaded recording and submission of command buffers. Unity's Graphics Jobs system optimizes this process by reducing CPU bottlenecks when handling draw calls.

The main thread queues draw commands, while the rendering thread spawns worker threads to convert and submit them as GPU command buffers. This update allows better CPU-GPU workload distribution for high-performance rendering and large-scale scenes.

DX12 Graphics Jobs are supported in the Editor on Windows platforms using DirectX 12. In runtime builds, Graphics Jobs are supported on Windows(DX12), and console platforms.

Variable Rate Shading (Unity 6.1)

Variable Rate Shading (VRS) lets you adjust how detailed the shading is across different parts of the screen. By controlling shading quality at the pixel or object level, games can boost GPU performance without significantly impacting visual quality.

For example, in a racing game, the player's car and detailed surroundings can use high-quality shading, while lower-quality shading is applied to less noticeable areas, like motion-blurred roads, to save resources.

Unity 6.1 introduces a new API for Variable Rate Shading, supporting Windows (DirectX 12), Vulkan, and compatible consoles.

Check the Shading Rate sample project for a practical demo for how Variable Rate Shading works. See this discussion page for more details.

Variable Rate Shading adjusts shading quality across different parts of the screen.

Next steps

We hope this guide inspires you to try HDRP for your next project. Remember that the **3D Sample Project** is always 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®