# Lighting up your graphics

With Unity, you can achieve realistic lighting that's suitable for a wide range of art styles. This e-book outlines some of the tools and features you'll find useful as you get started with the Universal Render Pipeline.

unity

# Introduction

Lighting is one of the most important visual elements of your game. Unity helps you achieve your desired look while keeping your game optimized for performance. Throughout this e-book, we'll be drawing practical examples from the *Boat Attack* sample game, a project created by the Unity R&D team to showcase what can be achieved with the Universal Render Pipeline (URP).

Today, it's assumed that a game will run on many different kinds of hardware. Players expect the game to be aesthetically pleasing and to play smoothly across a broad range of specs, including low-end devices.

With Unity's Universal Render Pipeline, you can achieve great-looking graphics using artist-friendly workflows that let you quickly create optimized graphics for different platforms, from mobile to high-end consoles and PCs. The key is in the many optimizations performed behind the scenes, crunching every factor that goes into a perfect frame.

In this e-book, we'll discuss those factors, introduce some real-time concepts, and walk you through common lighting configurations in Unity to help bridge the gap between aesthetics and performance using an in-house game demo to see these effects at work. By the end, you'll come away with concrete advice and valuable lighting tips that you can bring to your own projects.

Let's get started.


### *Boat Attack* game demo

We're using *Boat Attack*, a vertical slice demo of a boat-racing game, to illustrate the lighting concepts and techniques described in this e-book.

*Boat Attack* is a real-world boat racing game that plays beautifully across platforms including desktop, mobile, and consoles.

The demo was developed in-house by the Unity R&D team to drive product development and showcase the features and benefits of the URP.

You can download the demo here.

1 —

# Getting started

To achieve best-in-class visual quality and performance, you need to consider certain tasks in your lighting flow. Here are the first four approaches that will help you push your project further when working in Unity.

**1. Choosing a render pipeline**

The first thing to know before you can think about lighting for your project: you need to select your Scriptable Render Pipeline, as this forms the basis of the technical decisions you will make from there on.

The pipeline you choose will perform a series of operations that may not translate if you switch to a different pipeline mid-project. You have to commit to a pipeline right from the beginning.

Each of the available pipelines has different strengths, and choosing the best one for the job depends on your goals and circumstances. For *Boat Attack*, the idea was to deploy the game across many different devices. Since the goal of this e-book is to help you achieve high-fidelity lighting with maximum performance and platform reach, the optimal pipeline to work in is Unity's Universal Render Pipeline. URP is one of the two highly customizable, C# scriptable rendering path templates that make up Unity's Scriptable Render Pipeline offerings. The goal of the URP is to provide optimized real-time rendering on a wide selection of high- to lower-power devices, which is achieved by constraining options and guiding users towards performant rendering choices with regard to lighting and shading.

It's easy to set up the URP in Unity through the Unity Hub or Package Manager. You can also learn more about URP features in this blog post.

**2. Organizing with Volumes**

In URP, many rendering and post-processing properties are driven by a Volume framework (and stored in the Volume's associated profiles). Your settings will essentially be organized in categories attached to the Volume component. This approach means that the properties that control rendering can be applied spatially as the game camera moves around its world.

In *Boat Attack*, individual Volumes can be given a priority and weighting that determine how their attached properties blend with others as the boat travels between darker caves and open sky. Working with Volumes at different scales (globally, locally, or per shot), provides fine-grained control over rendering for a variety of contexts.

This workflow means that Volumes are perfect for experimentation, since you can duplicate a Volume profile asset and play around with its configuration, then revert to the original Volume with little consequence.

Get to know Volumes in Unity for maximum flexibility and the ability to fine-tune how your real-time scenes appear and perform.

**3. Embracing collaboration**

One of the greatest advantages of creating projects in Unity is that multiple people who need to handle the same scene can work on it without blocking one another's progress, instead of waiting for the previous task to complete.

You'll see this capability in action with multi-scene editing. Using multi-scene editing, one artist can make a change – for instance, in response to a modification that someone else made in the same scene – without stepping on another person's work. Unity handles each artist's data through version control and merges all their changes behind the scenes.

With multi-scene editing, you can store the base art direction lighting (such as Global Illumination) in one scene and store a separate scene with settings that are dynamic and shot-specific so that artists can safely modify the shot.

**4. Saving time with Prefabs**

Another benefit of using Unity is that the platform offers you quick ways to get started – templates you can use so you're not forced to manually set up every detail in a scene, start all over again in the next scene, and then waste time tracking down elements when you need to change something small.

Multi-scene editing enables collaborative workflow. You can have lighting, environments and animations in separate scenes.

The *Boat Attack* demo features rich, performant graphics with nuanced lighting that displays beautifully on almost any device.

Unity's Prefabs are one of those shortcuts. These templates are great for grouping objects that need to be reused multiple times. A Prefab lets you modify the composition of your group, and all instances will follow automatically. Prefab instances also let you easily and selectively revert any derivation you've made, or apply it back to the original Prefab and propagate changes instantaneously. This alone can save you hours.

You can also include Prefabs inside other Prefabs. This is called nesting Prefabs. Nested Prefabs were used to create the island level in *Boat Attack*, where various parts, such as the rocks and cliffs, were edited without disturbing the overall design of the island.

Another useful Prefab concept is Prefab Variants. A Prefab Variant inherits the properties of another Prefab, called the base. Overrides made to the Prefab Variant take precedence over the base Prefab's values. There are several types of boats in *Boat Attack*, all based on the same basic Prefab. However, each of the boats moves at a different speed, has a different paint color, and emits distinctive sound effects, so Prefab variants were extremely helpful in creating them.

Now let's get lighting in Unity with these familiar principles: Global Illumination, reflections, and direct lighting.

2 —

# Real-time direct lighting

The process starts with real-time direct lighting, which lays the foundation when you begin lighting your scene. In URP, direct lights have a limited subset of features to keep them performance-oriented and ensure the ability to target as many platforms as possible. URP has one Main Light, which is the directional light, and Additional Light, which are local lights with a limited count that supplement the Main Light (local lights are analogous to pixel lights in the Built-in Render Pipeline). In *Boat Attack*, we use only the directional light, with no Additional Light. The following are some of the controls and settings you'll find in Unity to configure them.

The same scene with directional light (top) and without (bottom).

### Directional light

The Main Light is useful for creating effects such as sunlight in your scenes. This is a directional light, so all objects in the scene are illuminated as if the light is always coming from the same direction. The distance of the light from the target object is not defined, so the light does not diminish.

To enable additional lighting, you need to select the render pipeline asset and increase its number. If you do not do this, your Main Light will disable all other local lights.

### Shadows in moderation

Shadows are far from free. Any GameObject that might cast shadows must first be rendered into the shadow map; then that map will be used to render objects that might receive shadows. For directional light, the shadow map covers a large portion of the scene, which can lead to a problem called perspective aliasing. Perspective aliasing means that shadow map pixels close to the camera look enlarged and chunky compared to those farther away. URP solves this using cascaded shadow maps. It breaks the camera frustum's depth into regions that can be covered with shadow maps proportional to the screen estate they occupy. This increases the rendering overhead – but not as much as if you were to use a high-resolution map across the whole shadow. Only the Main Light lets you have cascade shadows. Any additional directional lights are non-shadowed and will be classed as Additional Light.

*As a note, real-time shadows for point lights are in development, but they are not supported at this time.*

URP allows for realistic interplay of light, shadows and reflections, even on low-end devices.

URP lets you achieve stunning vegetation, structures, clouds, and more.

3 —

# The big picture: Global Illumination

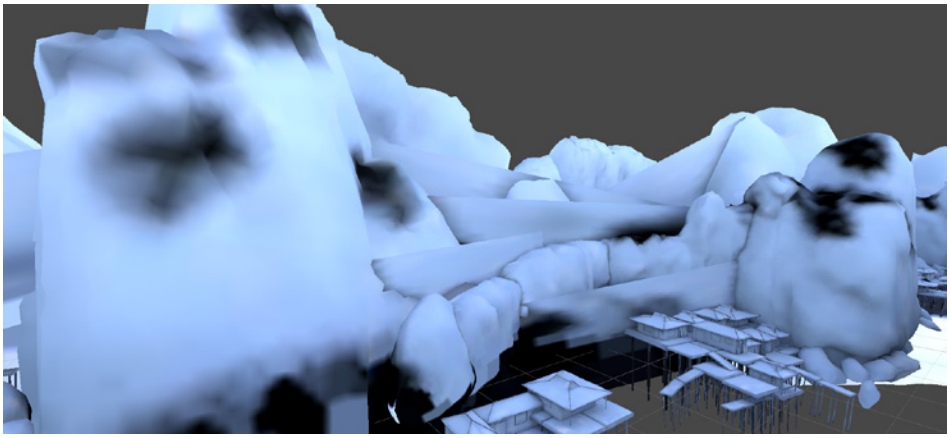
In an offline renderer, light scattering is normally achieved through lengthy offline calculations, but these would be prohibitively expensive to perform in real-time. In order to achieve similar results with playable frame rates, game engines must perform these calculations during Edit time, and encode the results in a way that can be quickly read back during play. This is a process known as “baking.”

Unity’s baked Global Illumination system makes realistic bounced lighting feasible in-game by calculating the light transport offline and storing the results using a number of techniques: lightmaps, Light Probes and Reflection Probes. All of these are ways that complex lighting characteristics can be stored and easily decoded by shaders.

**Lightmapping**

Lightmaps are essentially textures that store lighting. Direct lighting (light which is received directly from the source with no bounce), indirect lighting (light which has been diffused by bouncing around the scene), and shadowmasks (which explicitly store baked shadows separate from other lighting components) can all be encoded into lightmaps. These can then be used to light static objects in the scene, such as cliff walls and beach floors. In *Boat Attack*, we used the Baked Indirect option. This setting bakes the directional light to create real-time lighting along with high-

fidelity directional lighting, which is especially useful since this option allowed us to move the directional light around without fixing its direction during the game development process.

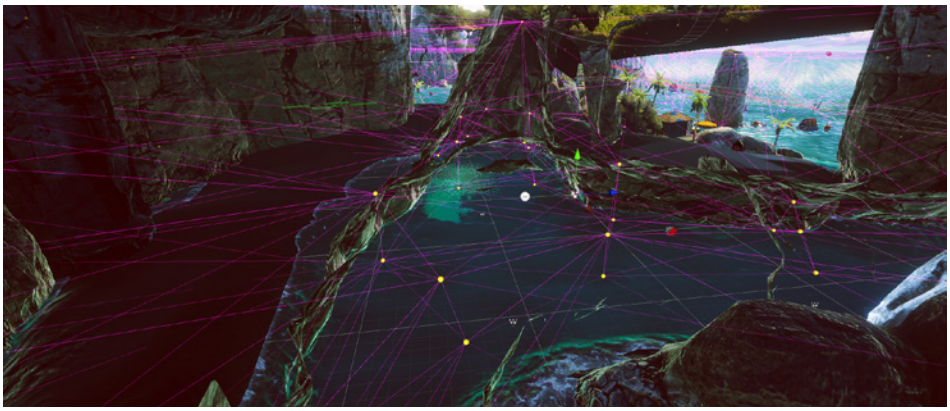


The Baked Lightmap option in Scene view helps you to visualize contact shadows and bounce lights.

In Unity, lightmapping is an integral part of the rendering engine that directly affects the quality of lighting in any given scene – from the smoothness of shadows to the blending between light and dark areas. With lightmapping, you'll configure how many lighting samples you need to retrieve lighting information, choose an appropriate texel density for your game type, optimize sample counts to achieve noiseless bakes in a reasonable amount of time, and denoise to clean up lower sample count bakes.

**Light Probes**

Only static objects are considered by Unity's baked GI systems. In order for dynamic objects such as the boat to pick up some of the rich bounced light that the static geometry receives, this lighting information must be recorded in a format that can quickly be read and used in lighting equations during gameplay. After establishing the ambient lighting and lightmapping settings, you can use a Light Probe Group to illuminate the remaining props and dynamic objects. The scene here uses an array of probes in the path that the boats take, following the water (and making sure no probes end inside an object).



Experiment with using multiple groupings of Light Probes to see how these affect performance in your scene.

4 —

# Reflections

Computer-generated films and animated console games commonly feature highly realistic reflections, which are important for creating a sense of connectedness among the objects in the game. However, the accuracy of these reflections comes at a high cost in processor time – and while this isn't a problem for high-end devices, it severely limits the use of reflective objects in low-end devices. Reflections can quickly deteriorate your frame rate, making it difficult to work in the Editor, so plan for them carefully.

**Static (baked) reflections**

Baked reflection probes store a reflection cubemap generated in the Editor for subsequent use in the game. For an object to show these reflections, its shader must have access to the images representing the cubemap. Each point of the object's surface can "see" a small area of the cubemap in the direction that the surface faces (i.e., the direction of the surface normal vector). A good rule of thumb is to place your capture point at the average height of your camera work.

Reflection probes are represented as cubes. You can place multiple reflection probes in your scene to improve the quality of reflections.

Some reflection probes are very specific to shadowed areas. Using more localized Reflection Probes ensures greater precision in the indirect reflections because they provide more accurate reflections in the less open areas. In *Boat Attack*, all reflections were baked. Reflection probes were kept to a minimum since these are loaded with the scene and risk increasing the loading time.

**Real-time reflections**

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 and can be updated in real-time to show changes in the scene. However, it takes considerable processing time to refresh the view of a probe, so it's wise to manage your updates carefully. Since the aim of this project is to also get optimal performance, there are no real-time reflections.

With Reflection Probes turned on, you can see the stones reflecting the water (top). With the Reflection Probes disabled, the stones look flat and out of place (bottom).

5 —

# Post-processing

Post-processing in Unity is the set of rendering effects that you apply, based on your existing rendered scene, before generating the final render. That means you get instant visual feedback for the effects you choose, instead of having to add finishing steps afterward, dramatically improving the scene without altering your existing content.

**Shadows Midtones Highlights**

The Unity platform includes highly controllable trackballs. Trackball controls are a familiar tool – they're easy to manipulate, and lighting pros really love them. You can use trackballs to help give your project a style or look that sets it apart.


With Processing Effects turned off, the game looks flat and bleached (top). Processing Effects provide a dramatic improvement to the game's visual quality (bottom).

### Vignette

Vignette is a powerful effect, but one that can easily be overused. It mimics an artifact of filming through a lens, visually representing the loss of light in the corner of the frame. For realism, this effect should always be rounded.

### Bloom

Bloom is a beautiful artifact created by strong light bleeding on your camera sensor and dirty lenses. You can simulate this effect in Unity, but, as with many effects, a subtle touch is crucial, and this technique should be used in moderation.

### Motion Blur

Motion Blur simulates the blurring of an image when objects filmed by a camera are moving faster than the camera's exposure time. This can be caused by rapidly moving objects or a long exposure time. Since *Boat Attack* is a fast-paced racing game, this effect is critical to give players the sense of speed.

### Tonemapping

Tonemapping is the process of remapping the high dynamic range (HDR) values of an image to a new range of values. Its most common purpose is to make an image with a low dynamic range appear to have a higher range. In Unity, tonemapping has two modes: ACES and Neutral. ACES gives a more contrasted and punchy image, but it's performance intensive. To deploy to lower-end devices, the *Boat Attack* team used the Neutral tonemapping mode because it's more performant, then leveraged color grading to achieve the same look as ACES would have created.

### Chromatic Aberration

Chromatic Aberration mimics the effect that a real-world camera produces when its lens fails to join all colors to the same point, creating fringes of color along the boundaries that separate darker and lighter parts of the image.

### Panini Projection

This effect helps you to render perspective views in scenes with a very large field of view. Panini Projection is a cylindrical projection, which means that it keeps lines straight and vertical. Unlike other cylindrical projections, Panini Projection also keeps radial lines through the center of the image straight.

### Depth of Field

Depth of Field simulates the focus properties of a camera lens. A real-world camera can only focus sharply on an object at a specific distance; objects nearer or farther from the camera will appear slightly out of focus. This effect is not used when *Boat Attack* is deployed to lower-end devices. Since there's a performance hit, it's only enabled for higher-end platforms.

# Conclusion

In this e-book, we have covered some best practices that a game artist should follow to get the most out of Unity's URP for great-looking lighting.

**Choose your path**

Ready to learn how Unity can help your artists? Unity is more accessible than ever to creators and technical directors. Take the Unity Learn free one-hour tutorial Introduction to Lighting and Rendering to get more comfortable with how lighting works in Unity.

If you're new to the art of lighting for real-time rendering, refer to the glossary on the following pages.

**Universal Render Pipeline and the High Definition Render Pipeline**

The URP does not replace or encompass the High Definition Render Pipeline (HDRP). The URP aims to be the future default render pipeline for Unity. It is more flexible, more extensible, and delivers higher performance than the Built-in Render Pipeline, with improved graphics quality.

HDRP delivers state-of-the-art graphics. HDRP is the best to use if your goal is more targeted – pushing graphics on high-end hardware, delivering performant powerful high-fidelity visuals.

You should choose the render pipeline to use for a project according to the features and platform requirements of your project.

**Additional resources**

- Find the *Boat Attack* game demo here.
- Discover how the Lightweight Render Pipeline is evolving in this blog post.
- Read this article for more information about how to achieve beautiful, scalable, and performant graphics with the URP.
- Watch the Unite Now session, "Evolving game graphics with the Universal Render Pipeline", on YouTube.
- Discover how the URP unlocks games for you in this video.

# Real-time lighting glossary

| Term | What it is | When you'll use it | Why it matters |
|---|---|---|---|
| **Baked Global Illumination (baked GI)** | Lighting baked into textures called lightmaps and into Light Probes. | When you've generated traditional lightmap textures offline during the precompute process, rather than inside Unity. | Better performance is realized even with extremely high-fidelity textures because these assets are not being computed at rendering time. |
| **Global Illumination (GI)** | A group of techniques that model both direct and indirect lighting to provide realistic lighting results. | To generate the indirect lighting in the scene, primarily as a function of the direct lighting. | GI brings a more realistic, balanced, cohesive look to a scene that's less labor-intensive and more true-to-life than the effect created by local lighting on individual objects. |
| **High Definition Render Pipeline (HDRP)** | HDRP is Unity's pre-built, scriptable pipeline for authoring high-fidelity projects using physically based lighting and advanced visual effects. | When you need advanced rendering and shading features for projects that require a high degree of visual fidelity using high-end hardware. | HDRP in Unity gives console, PC game studios and animation studios the ability to achieve graphical fidelity and visual realism in their productions. |
| **Light Probes** | Light Probes are positions in the scene where the light is measured (probed) to provide indirect lighting values for dynamic scene objects during rendering. | In areas between objects, where light is passing through empty space in your scene. | Light Probes are important tools in providing high-quality lighting (including indirect bounced light) on moving objects in your scene. |
| **Lightmaps** | A pre-rendered texture (overlaid on top of scene geometry) that contains the effects of light sources on static objects in the scene. | When using Unity's Baked Global Illumination system to provide realistic lighting results. | Lightmaps make it possible for much of the lighting to be entirely precomputed, so real-time performance is not compromised. |
| **Lightmapper** | Unity's underlying system to generate data for lightmaps and Light Probes by shooting light rays, calculating light bounce, and applying the resulting lighting into textures. | Only when the surface is made up of non-overlapping UVs with small area and angle errors and there is sufficient padding between the charts. | Various lightmappers will often produce different lighting looks, as they might rely on distinctive techniques to produce the lighting data. |
| **Multi-scene editing** | The ability in Unity's Editor to have multiple scenes open in the Editor simultaneously to enable multiple people to collaborate simultaneously in a scene within a seamless workflow. | When configuring a setting that affects multiple scenes or when more than one person needs to modify the scene, e.g., both the lighting artist and the environment artist are making changes. | An important time-saver when multiple artists need to access the same scene simultaneously, or when certain scene files (e.g., base lighting like GI) need to be reused across different compositions. |
| **Prefabs** | A Unity workflow system that allows you to create, configure, and store an object's properties as a reusable asset. | Whenever you want to reuse an object's configuration in multiple places in a scene, or across multiple scenes in a project instead of recreating a new one every time. | Save many hours on any scene by duplicating, modifying, and instantly propagating changes to a single instance of an asset's settings. |

| Term | What it is | When you'll use it | Why it matters |
|---|---|---|---|
| **Reflection Probes** | A Reflection Probe captures a view of its surroundings and stores the results, which are then used on objects to produce accurate reflections of their surroundings. | To help you create accurate reflective materials and realistically reactive visuals for objects in your scenes. | Subtle and accurate reflections, balanced with controlling the area and detail covered, maximize a scene's believability without creating a drag on performance. |
| **Reflection Probe capture point** | The position of a Reflection Probe relative to the scene it's mirroring. | When placing Reflection Probes in a layout and setting local and world properties. | To ensure reflection angles and positions portray a precise and accurate view. |
| **Shadow cascades** | Splitting shadow maps into zones by proximity to the camera to reduce pixelation in the closer shadows. | When you're using a directional light that covers a large portion of the scene, leading to visible aliasing on closer shadow pixels. | Managing shadows with cascades is less of a drag on performance than using a high-resolution map across the whole shadow. |
| **Scriptable Render Pipeline (SRP)** | A Scriptable Render Pipeline allows you to control rendering with C# scripts, giving you a high degree of customization. There are two pre-built systems in Unity – High Definition Render Pipeline (HDRP) and the Universal Render Pipeline (URP) – that tailor the rendering process to your target platform so the project is optimized for specific hardware. | Choose the HDRP to deliver cinematic graphics for projects specifically targeted to high-end hardware with high-fidelity visuals.<br>Or, choose the URP when you don't need cinematic fidelity but want greater flexibility and extensibility than the built-in render pipeline and high-quality graphics across broad use cases. | Committing to a render pipeline at the start gives you access to a visual way of creating and editing sophisticated pipeline tools without having to write code. |
| **Universal Render Pipeline (URP)** | URP is a prebuilt Scriptable Render Pipeline that lets you quickly and easily create optimized graphics across a range of platforms. | When you need artist-friendly workflows to create projects across mobile to high-end consoles and PCs. | URP provides scalable graphics quality to meet device performance, so you get maximum-quality graphics on high-end devices and optimized performance on lower-end devices. |
| **Volume framework** | The Volume framework is how you partition your scene into global or local areas so that you can control lighting and effects at a finer level, rather than tuning an entire scene. | You can add as many Volumes to your scene as you want to create different spaces, then light each one individually for realistic effect. | Gives you fine scene-by-scene control (and even area control within scenes) over environment settings, such as fog color and density, that alter the mood of your scene in calculating the final render. |


unity