1 Lighting

For this tutorial, we will be looking at the lighting system in Unity.

2 Realtime Lighting

By default, lights in Unity - directional, spot and point, are realtime. This means that they contribute direct light to the scene and update every frame. As lights and GameObjects are moved within the scene, lighting will be updated immediately. This can be observed in both the scene and game views. Realtime lighting is the most basic way of lighting objects within the scene and is useful for illuminating characters or other movable geometry.

Unfortunately, the light rays from Unity’s realtime lights do not bounce when they are used by themselves. In order to create more realistic scenes using techniques such as global illumination we need to enable Unity’s precomputed lighting solutions.

3 Baked Lighting

When “baking” a “lightmap”, the effects of light on static objects in the scene are calculated and the results are written to textures which are overlaid on top of scene geometry to create the effect of lighting. These “lightmaps” can include both the direct light which strikes a surface and also the “indirect” light that bounces from other objects or surfaces within the scene. This lighting texture can be used together with surface information like color (albedo) and relief (normals) by the “Shader” associated with an object’s material.

With baked lighting, these light textures (lightmaps) cannot change during gameplay and so are referred to as “static”. Realtime lights can be overlaid and used additively on top of a lightmapped scene but cannot interactively change the lightmaps themselves.

With this approach, we trade the ability to move our lights at gameplay for a potential increase in performance, suiting less powerful hardware such as mobile platforms.
4 Precomputed GI lighting

Whilst traditional, static lightmaps are unable to react to changes in lighting conditions within the scene, Precomputed Realtime GI does offer us a technique for updating complex scene lighting interactively.

With this approach it is possible to create lit environments featuring rich global illumination with bounced light which responds, in realtime, to lighting changes. A good example of this would be a time of day system - where the position and color of the light source changes over time. With traditional baked lighting, this is not possible.

In order to deliver these effects at playable framerates, we need to shift some of the lengthy number-crunching from being a realtime process, to one which is “precomputed”.

Precomputing shifts the burden of calculating complex light behaviour from something that happens during gameplay, to something which can be calculated when time is no longer so critical. We refer to this as an “offline” process.

So how does this work?

Most frequently it is indirect (bounced) light that we want to store in our lightmaps when trying to create realism in our scene lighting. Fortunately, this tends to be soft with few sharp, or “high frequency” changes in color. Unity’s Precomputed Realtime GI solution exploits these “diffuse” characteristics of indirect light to our advantage.

Finer lighting details, such as crisp shadowing, are usually better generated with realtime lights rather than baking them into lightmaps. By assuming we don’t need to capture these intricate details we can greatly reduce the resolution of our global illumination solution.

By making this simplification during the precompute, we effectively reduce the number of calculations we need to make in order to update our GI lighting during gameplay. This is important if we were to change properties of our lights - such as color, rotation or intensity, or even make change to surfaces in the scene.

To speed up the precompute further Unity doesn’t directly work on lightmaps texels, but instead creates a low resolution approximation of the static geometry in the world, called “clusters”.

Traditionally when calculating global illumination, we would “ray trace” light rays as they bounce around the static scene. This is very processing intensive and therefore too demanding to be updated in realtime. Instead, Unity uses ray tracing to calculate the relationships between these surface clusters beforehand - during the “Light Transport” stage of the precompute.

By simplifying the world into a network of relationships, we remove the need for expensive ray tracing during the performance-critical gameplay processes.

We have effectively created a simplified mathematical model of the world which can be fed different input during gameplay. This means we can make modifications to lights, or surface colors within the scene and quickly see the effects of GI in scene lighting update at interactive framerates.
The resulting output from our lighting model can then be turned into lightmap textures for rendering on the GPU, blended with other lighting and surface maps, processed for effects and finally output to the screen.

Only static geometry is considered by Unity’s precomputed lighting solutions. To begin the lighting precompute process we need at least one GameObject marked as “static” in our scene. This can either be done individually, or by shift-selecting multiple GameObjects from the hierarchy panel.

From the Inspector panel, the Static checkbox can be selected (Inspector>Static). This will set all of the GameObject’s “static options”, or “flags”, including navigation and batching, to be static, which may not be desirable. For Precomputed Realtime GI, only “Lightmap Static” needs to be checked.

For more fine-grained control, individual static options can be set from the drop-down list accessible to the right of the Static checkbox in the Inspector panel. Additionally, objects can also be set to Static in the Object area of the lighting window.

If your scene is set to Auto (Lighting>Scene>Auto), Unity’s lighting precompute will now begin automatically. Otherwise it will need to be started manually as described below.

5 Ambient Lighting

An important contributor to the overall look and brightness of a scene is “ambient lighting”. This can be thought of as a global light source affecting objects in the scene from every direction.

Ambient light can be useful in a number of cases, depending upon your chosen art style. An example would be bright, cartoon-style rendering where dark shadows may be undesirable or where lighting is perhaps hand-painted into textures. Ambient light can also be useful if you need to increase the overall brightness of a scene without adjusting individual lights.

6 Directional Lights

“Directional Lights” are very useful for creating effects such as sunlight in your scenes. Behaving in many ways like the sun, Directional Lights can be thought of as distant light sources which exist infinitely far away.

Light rays emitted from Directional Lights are parallel to one another and do not diverge like those from other light types. As a result, shadows cast by Directional Lights look the same, regardless of their position relative to the source. This is useful to us, especially when lighting outdoor scenes. As Directional Lights do not have a source position, they can be placed anywhere.
in your scene without changing the effect of the light. Rotating the light however does greatly affect
the visual result.

With other light types where there is an obvious source position, such as Spotlights, character
shadows will change as the character moves closer to the light source. This can be a problem when
trying to generate character shadows in interior levels. Directional lights are advantageous in these
situations as shadows remain consistent regardless of proximity to the light source.

Directional Lights do not diminish over distance. As they affect all surfaces in your scene (unless
culled).

7 Point Lights

A Point Light can be thought of as a point in 3D space from which light is emitted in all directions.
These are useful for creating effects like light bulbs, weapon glow or explosions where you expect
light to radiate out from an object.

The intensity of of Point Lights in Unity diminishes quadratically from full intensity at the
centre of the light, to zero at the limit of the light’s reach defined by the “Range” property of the
component in the Inspector. Light intensity is inversely proportional to the square of the distance
from the source. This is known as “inverse square law” and is similar to how light behaves in the
real world.

Enabling shadows for Point Lights can be expensive and so must be used sparingly. Point
Lights require that shadows have to be rendered six times for the six world directions and on slower
hardware this can be an unacceptable performance cost.

When adding Point Lights to a scene it’s worth noting that currently they do not support indirect
bounce light shadowing. This means that the light created by Point Lights will continue through
objects and bounce on the other side unless attenuated by range. This can lead to light “leaks?”
through walls and floors and therefore lights must be carefully placed to avoid such problems. This
however is not a problem when using Baked GI.
8  **Spotlights**

Spotlights project a cone of light in their forward (+Z) direction. The width of this cone is defined by the light’s “Spot Angle” parameter. Light will “falloff” from the source position towards the extent of the light’s range, where it will eventually diminish to zero. Light also diminishes at the edges of the Spotlight’s cone. Widening the Spot Angle increases the width of the cone and with it, increases the size of this fade, known as the “penumbra”.

![Spotlight diagram](image)

Spotlights have many useful applications for scene lighting. They can be used to great effect as street lights, wall downlights or used dynamically, for creating effects like a flashlight. As their area of influence can be precisely controlled, Spotlights are extremely useful for creating focus on a character or for creating dramatic stage lighting effects.

9  **Area Lights**

Area Lights can be thought of as similar to a photographer’s softbox. In Unity they are defined as rectangles from which light is emitted in all directions, from one side only - the object’s +Z direction. Presently only available in Baked GI, these Area Lights illuminate uniformly across their surface area. There is no manual control for the range of an Area Light, however intensity will diminish at inverse square of the distance as it travels away from the source. Area Lights are useful in situations where you wish to create soft lighting effects. As light is emitted in all directions across the surface of the light, the rays produced travel in many directions - creating a diffuse lighting effect across a subject. A common use for this might be a ceiling striplight or a backlit panel.

In order to achieve this, we must fire a number of rays from each lightmap texel in the world, back towards the light in order to determine whether the light can be seen. This means that Area Lights can be quite computationally expensive and can increase bake times. However, used well,
they can add a great depth of realism to your scene lighting and this extra precomputation may be justified. Note that as they are baked only, gameplay performance is not affected.

10 Emissive Materials

Whilst Area Lights are not supported by Precomputed Realtime GI, similar soft lighting effects are still possible using “Emissive Materials.” Like Area Lights, emissive materials emit light across their surface area. They contribute to bounced light in your scene and associated properties such as color and intensity can be changed during gameplay.

“Emission” is a property of the Standard Shader which allows static objects in our scene to emit light. By default the value of “Emission” is set to zero. This means no light will be emitted by objects assigned materials using the Standard Shader. The HDR color picker can be used to select colors with intensities beyond the 0-1 range in order to create bright light effects similar to those of Area Lights.

There is no range value for emissive materials but light emitted will again falloff at a quadratic rate. Emission will only be received by objects marked as “Static” or “Lightmap Static” from the Inspector. Similarly, emissive materials applied to non-static, or dynamic geometry such as characters will not contribute to scene lighting.

However, materials with an emission above zero will still appear to glow brightly on-screen even if they are not contributing to scene lighting. This effect can also be produced by selecting “None” from the Standard Shader’s “Global Illumination” Inspector property. Self-illuminating materials like these are a useful way to create effects such as neons or other visible light sources.

Emissive materials only directly affect static geometry in your scene. If you need dynamic, or non-static geometry - such as characters, to pick up light from emissive materials, Light Probes must be used. Changing emission values at gameplay will update Light Probes interactively and the results will be visible on any objects presently receiving light from those probes.
11 Light Probes

Only static objects are considered by Unity’s Baked or Precomputed Realtime GI systems. In order for dynamic objects such as interactive scene elements or characters to pick up some of the rich bounced light that our static geometry receives, we need to record this lighting information into a format which can be quickly read and used in our lighting equations during gameplay.

We do this by placing sample points in the world and then capturing light from all directions. The color information these points record is then encoded into a set of values (or “coefficients”) which can be quickly evaluated during gameplay. In Unity, we call these sample points, “Light Probes”.

Light Probes allow moving objects to respond to the same complex bounced lighting which is affecting our lightmaps regardless of whether Baked GI or Precomputed Realtime GI is used. An object’s mesh renderer will look for the Light Probes around its position and blend between their values. This is done by looking for tetrahedrons made up by the position of Light Probes, and then deciding which tetrahedron the object’s pivot falls into. This allows us to place moving characters in scenes and have them appear properly integrated. Without Light Probes, dynamic objects would not receive any global illumination and would appear darker than surrounding, lightmapped geometry.

By default there are no Light Probes in a scene so these will need to be placed using a Light Probe Group (GameObjects>Light>Light Probe Group).

If the “Auto” box is checked at the bottom of your scene precompute settings (Lighting>Scene>Auto), Light Probes will update whenever changes are made to the scene lighting or static geometry. Otherwise they will be updated when the Build button is clicked.

12 Try it

We can also perform scripting on the lights using the following code.
Figure 1: Use Baked GI, Emissive material on a plane for the light, and different color materials

```csharp
using UnityEngine;
using System.Collections;

public class lightanim : MonoBehaviour {
    public Light light;
    public Transform obj;
    // Use this for initialization
    void Start () {
    }

    // Update is called once per frame
    void Update () {
        light.intensity = Mathf.PingPong(Time.time, 2.0f);
        light.transform.LookAt (obj.transform.position);
    }
}
```