Manual

Contents

System requirements

To use Bakery you'll need:

- Windows (7 or higher) PC.
- Nvidia GPU. Minimum supported model is Kepler (GeForce 650 or newer. For Quadro cards check the specs).
- 64-bit Unity editor. Bakery was tested on all versions from 5.6 to 2020.

> 💡System requirements are for developer machine, not target platform. You can use lightmaps baked with Bakery anywhere.

Prerequisites

- Make sure your project does not have any script compilation errors, as it will prevent Bakery scripts from compiling as well.
- Make sure you have the latest GPU driver. Update it (https://www.nvidia.com/Download/index.aspx?lang=en-US) if needed.

Use the Game-Ready (standard) driver, not the "Studio" one.

Installation

1. Import Bakery to your project via Asset Store (https://assetstore.unity.com/packages/tools/level-design/bakery-gpu-lightmapper-122218).

2. Unity will show a list of files to import. If it's your first time using Bakery, it is recommended you just **click Import**. Experienced users may want to untick the *examples* folder for a more lightweight installation.

Files will be imported to Assets/Bakery and Assets/Editor/x64/Bakery. These folders can be moved later.

3. Unity will then import Bakery and compile the scripts. It may show a window like this:
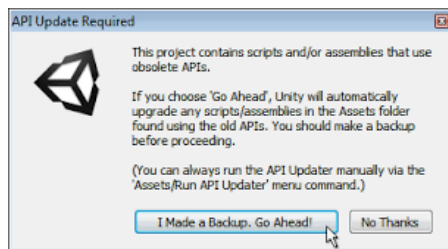
In which case you should click "Go Ahead".

You should now see Bakery menu added to the editor:
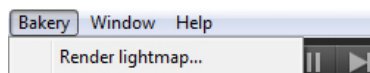
Quickstart

1. Add some model or primitive and mark it as Static.

Meshes must have non-overlapping UV for lightmapping. UV2 is used if present (otherwise UV1). If you did not unwrap models for lightmapping, make sure to check Generate Lightmap UVs (https://docs.unity3d.com/Manual/LightingGiUvs-GeneratingLightmappingUVs.htm l) on the asset. Unity primitives already have correct UV2.
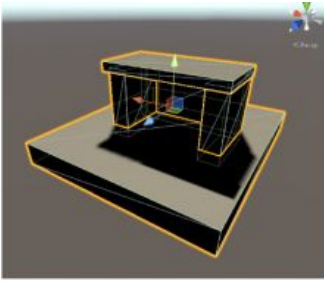
2. Select Directional Light (the one Unity created for you) and add **Bakery Direct Light** component to it.

> 💡 Bakery and Unity use separate light source components.

3. Bakery Direct Light has many options to tweak, but you can easily match it to Unity light. Click **Match lightmapped to real-time**. Now Bakery light should have the same yellowish color as Unity's default Directional.
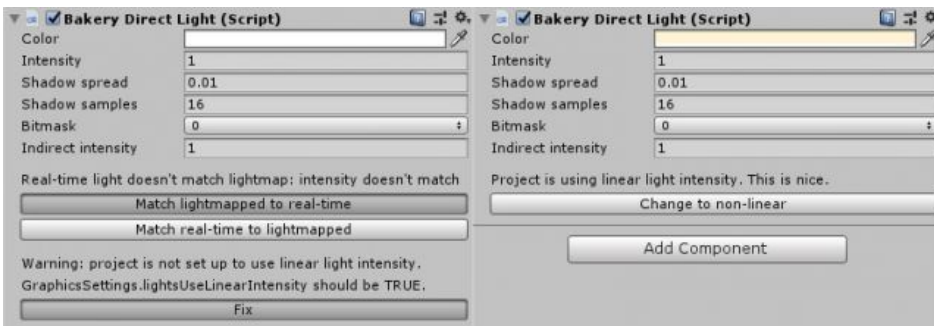
> 💡 Direct Light component may complain about project not using linear light intensity. The problem is that by default Unity applies gamma correction to light source colors incorrectly (https://twitter.com/guycalledfrank/status/1001108508614553600). Correct mode can be enabled by scripting (https://docs.unity3d.com/ScriptReference/Rendering.GraphicsSettings-lightsUseLinearIntensity.html), and that's what the "Fix" button does. If you already have many light sources set up in non-linear intensity mode, and you don't want to change them again, you may skip fixing. Even if you press "Fix", you can always revert the change anytime.

4. To get shadowed ambient lighting, create Skylight (Bakery->Create->Skylight).
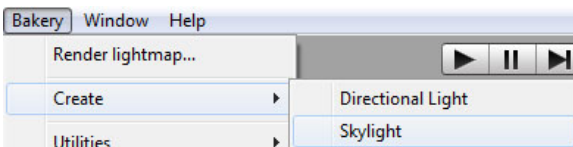
5. Select Skylight, give it some bluish color and click **Match scene skybox to this light**. This will make scene skybox match lighting exactly.

> 💡 Matching Unity skybox to Bakery skylight is optional but useful for correctness, as to ensure that visible environment and in-engine reflection probes match baked lighting. Currently this option only works for Standard render pipeline.
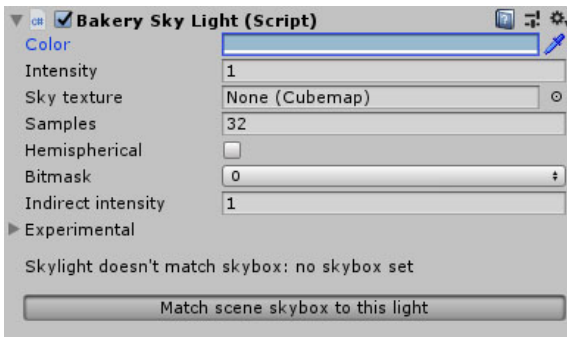
6. Click Bakery->Render Lightmap. By default, Full Lighting mode is active. It will bake both direct and indirect contribution from all lights. If you want to combine real-time shadows with baked GI, change it to **Indirect**.
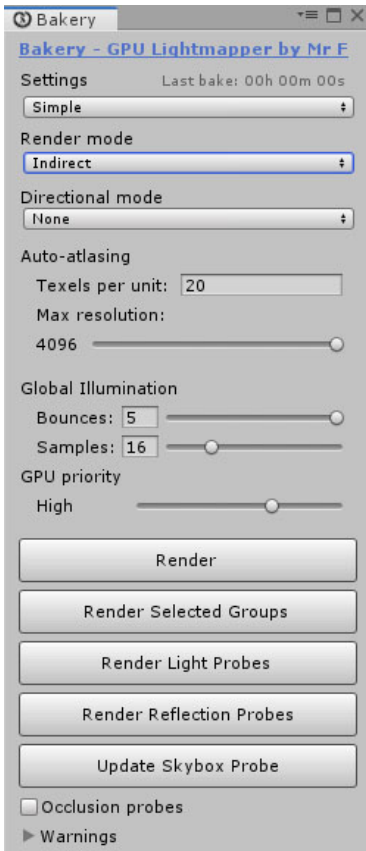
> 💡 Bakery may complain that you are using old gamma mode (https://docs.unity3d.com/Manual/LinearLighting.html) and suggest to change it. If you care about quality of lighting, using Linear mode is highly recommended. You may ignore it if you already have a big project set up with incorrect gamma and don't want to change it or when shipping to mobile devices that don't support it.

7. Click **Render**.

8. Done! You should now have baked GI:
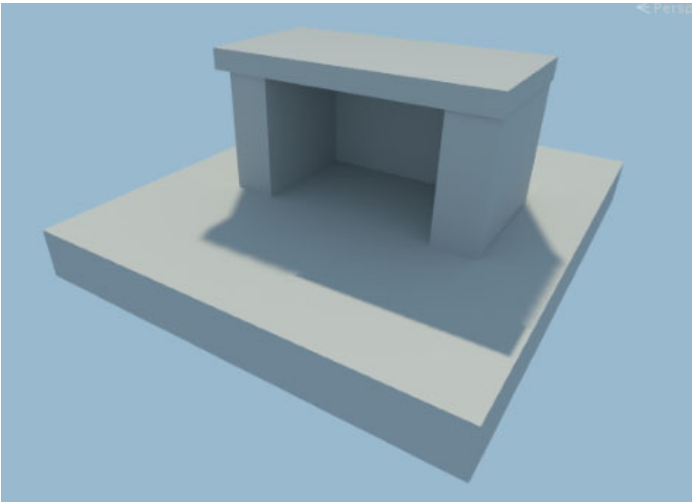
Render settings

All render settings can be accessed via Bakery->Render Lightmap. Settings are saved for every scene. The topmost option in Bakery window allows you to switch between Simple, Advanced and Experimental settings, each succeeding mode revealing more tweakable options. Simple mode is enough for most scenes and it's a good way to get started.

Render mode

Defines the type of lightmaps to bake.

Full Lighting

Complete direct and indirect lighting for all Bakery lights.

> 💡 On pre-2017.3 Unity versions if you have both Unity and Bakery components on the same object, you will need to disable Unity ones manually to avoid double brightness. On newer versions real-time effects are automatically disabled for baked lights, as with built-in lightmappers.

Indirect

Basic mixed mode. It will look at the Baked Contribution selector on every light: if it's set to *Direct And Indirect*, the light is baked as in Full Lighting mode. If it's set to *Indirect Only*, only indirect contribution (GI) from this light is baked. In latter case you should keep both Unity and Bakery lights on the object, one giving real-time direct contribution, and another for precomputed GI.
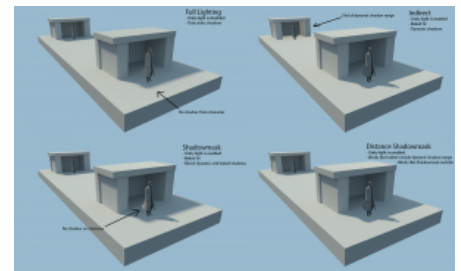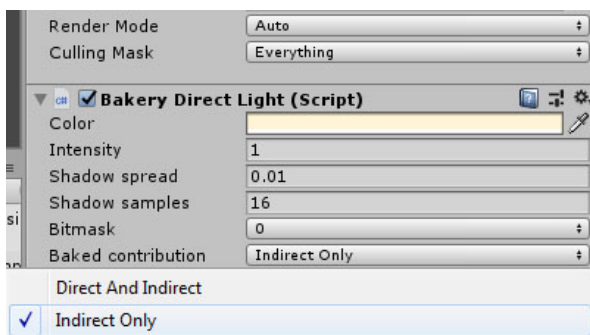
Comparison of different render modes supported by Unity

Shadowmask

A more advanced mixed mode. It works by generating two types of lightmaps - one with baked color (as in Indirect mode), and another with shadows from static objects. It has several benefits:

- Real-time shadows can render only a handful of dynamic objects, instead of the whole scene.
- Real-time and baked shadows blend together properly.
- Real-time lights can render bump, specular and other surface effects, while still being occluded by high-quality baked shadows.

More details (https://docs.unity3d.com/Manual/LightMode-Mixed-Shadowmask.html)

To enable this behaviour, you must have both Unity and Bakery lights on the same object, with Baked Contribution set to *Shadowmask and Indirect*. Unmarked lights will be baked as in Indirect mode.

Left: completely baked light. Center: dynamic light with baked GI. Right: capsule shadow, bump and specular are dynamic, but cube shadow and GI are baked.

Distance Shadowmask

This checkbox is only visible if you chose shadowmask and simply toggles equally named setting (https://docs. unity3d.com/Manual/LightMode-Mixed-ShadowmaskMode.html) in project's Quality Settings. More details. (https://docs.unity3d.com/Manual/LightMode-Mixed-DistanceShadowmask.html)

> 💡 Checkbox is not visible on Unity 5.6.x due to the lack of corresponding scripting API. Instead you can toggle it in Window->Lighting->Mixed Lighting->Lighting Mode.

> 💡 On Unity versions < 2017.3 rendering Shadowmask lightmaps will clear the light probes due to API limitations. They will need to be re-rendered again after bake. It shouldn't happen on newer versions.

When camera gets far away enough or if dynamic shadows are disabled, the same scene will look like this

Subtractive

Enables Subtractive (https://docs.unity3d.com/Manual/LightMode-Mixed-Subtractive.html) lighting mode.

This option does not do anything special to lightmaps and, in fact, works just like Full Lighting.

The only difference is that it also sets up real-time Unity lights to work with Subtractive mode (as it cannot be done through UI).

You will need to additionally set up global subtractive parameters (like the global shadow color) in Unity lighting window (https://docs.unity3d.com/Manual/GlobalIllumination.html#MixedLighting).

Ambient Occlusion Only

Only bakes AO. May be useful if you know what you're doing. Make sure to set up AO options.

Directional mode

Defines how directional information is baked. Standard lightmaps only store a single color per texel, while directional lightmaps give shaders a hint of how lighting changes over a hemisphere around the texel. This data is required to combine fully lightmapped areas with normal maps. Plugging it into the built-in Bakery shader can also produce approximate specular response.

None

No directional data, single color per texel.
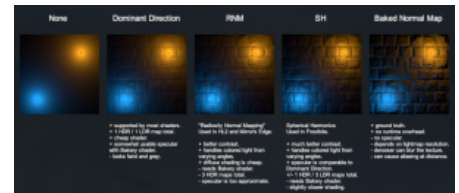
Comparison of different directional modes

Baked Normal Maps

Still no directional data, but normal maps are taken into account when rendering the lightmap. There is no additional runtime overhead. Since lightmaps usually have lower resolution than normal maps, the result may look blurry. Other problems include aliasing at distance due to lack of mipmapping and denoising step potentially smudging the detail. To learn how to use custom shaders with procedural normals in this mode, read Normal Mapping section.

Dominant Direction

This mode is similar to what Enlighten and Progressive bake in Unity. It is compatible will most shaders, only generates one additional map and the runtime overhead is minimal. The downside is that bump-mapping looks rather faint and gray-ish and can be quite different comparing to the same object under real-time lighting.

RNM

Based on Radiosity Normal Mapping technique originally invented for HL2 (slides (https://steamcdn-a.akamaihd.net/apps/valve/2004/GDC2004_Half-Life2_Shading.pdf)) and later used in many games (e.g. Mirror's Edge). It generates 3 HDR maps in total, being the most memory-demanding mode of all. Runtime overhead is still relatively low. This mode is more precise than Dominant Direction. It is better at reproducing surface contrast and handling colored lights affecting normal maps from different angles.

SH

Based on "Precomputed Global Illumination in Frostbite" paper (https://media.contentapi.ea.com/content/dam/eacom/frostbite/files/gdc2018-precomputedgiobalillumination infrostbite.pdf). This is the highest quality mode, giving much better surface contrast and representing differently colored lighting coming from different directions. Generates 4 maps in total, only one of them being HDR, therefore takes less memory than RNM. Runtime overhead is slightly higher than that of RNM.

MonoSH

Same as SH, but using a monochromatic L1 directional coefficients. Similarly good surface contrast, although not accurately representing different colors shining from different directions at one texel; but good enough for many scenes. Only generates one additional map, so the VRAM usage/bandwidth is similar to Dominant Direction; in fact it's even smaller, due to Dominant Direction using RGBA maps, while MonoSH only needing RGB. The map is fed via the built-in directional lightmap system, but *won't look correct with standard shaders*. Make sure to use a compatible shader.
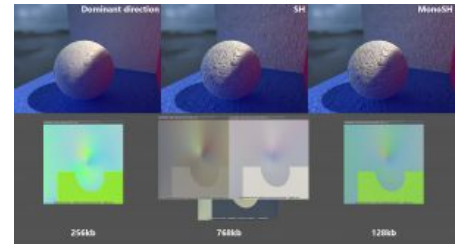
Size/quality comparison of Dominant Direction, full SH and MonoSH

Limitations

- RNM and SH/MonoSH can be only used with Bakery shader or require adjustments to your shaders.
- In RNM and SH modes, standard color lightmaps are not created. You need to either have Bakery shader on all materials in the scene, or use Lightmap Groups to separate RNM/SH objects from the rest.
- RNM and SH maps are applied to objects using MaterialPropertyBlock (https://docs.unity3d.com/ScriptReference/MaterialPropertyBlock.html) and so may be not shown completely in Unity's Lighting window.
- Dominant Direction, RNM and SH/MonoSH modes may require slightly more samples for GI and Light Meshes to get comparable quality.

Texels per unit

Approximate amount of lightmap texels per world unit. Affects the amount and resolution of generated lightmaps.

Example values to get you started:

- Large outdoor area (a city): 1 - 5
- Medium outdoor area (a few alleys): 10-20
- High quality interior: 100

1 vs 10 texels per unit

It is assumed that scene scale is roughly 1 unit = 1 meter. Such scale is generally recommended when working in Unity for better navigation and for physics simulation. If your scale is different, multiply Texels accordingly.
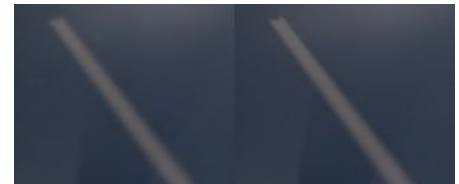
Keep in mind that Texels Per Unit is base resolution, but every object can be additionally tweaked using Scale in Lightmap (https://docs.unity3d.com/Manual/class-MeshRenderer.html) option on Mesh Renderers as well as Scale per map type in Bakery window.

> 💡 Objects with Scale in Lightmap set to 0 will not be baked, but will still cast shadows and affect GI.

Max resolution

Maximum lightmap size limit. If objects can't fit in a single lightmap (given current Texels Per Unit value), additional lightmaps will be allocated. Same 4 square objects can take four 512x512 maps or one 1024x1024.

Bounces

Defines how many times light rays should bounce off surfaces. Usually lower values are sufficient for outdoor scenes (e.g. a city), while higher values are required for more closed scenes (interiors, caves).

While the slider is limited by 5 bounces, it is possible to type any amount of bounces into the number field.

Samples

Affects quality of GI. Typical values are from 16 to 32.

> 💡 It is possible to type in any number, beyond the maximum slider value, but be aware that this number represents a square root of the final ray count, i.e. 32 is 1024 rays, 64 is 4096 rays, etc.

Note how more bounces bring more light to closed spaces.

GPU Priority

While GPU is working on lightmaps, OS and other software may become less responsive. This option allows you to balance between baking speed and system responsiveness.

Render

Bakes lightmaps for all opened scenes. If Light Probe Mode is set to L1 or L2, also bakes light probes.

Left: samples = 4, right: sample = 16. Denoising is off for illustrative purposes.

Render Light Probes

Bakes light probes (https://docs.unity3d.com/Manual/LightProbes.html) for all opened scenes.

Note that to get correct mixed light shadowing on dynamic objects in Shadowmask mode you also need to enable occlusion probes.

Render Reflection Probes

Bakes reflection probes (https://docs.unity3d.com/Manual/class-ReflectionProbe.html) for all opened scenes. This button is just for convenience and will call built-in engine reflection probe update.

Update Skybox Probe

Bakes global diffuse and reflection probe for the current skybox. As with "Render Reflection Probes", it just calls built-in engine functions.

Occlusion probes

When rendering light probes, lets Unity bake occlusion probes using currently selected built-in lightmapper. Occlusion probes is additional data stored inside regular light probes, and it prevents dynamic objects from getting lit in shadowed areas. Currently there is no way to use custom occlusion probes in Unity, and it has to call its own lightmappers instead to do the job.

Warnings

These options will validate the scene after one of the Render buttons is pressed and show warning dialog boxes asking to continue or stop the baking process.

- Suppress all popups: don't show any dialog boxes after pressing Render.
- UV validation: will check if all models have fully correct lightmapping UVs - specifically if they lie within 0-1 range and don't overlap.
- Overwrite check: will tell which lightmap files are going to be overwritten.
- Memory check: will tell a very approximate amount of video memory required.
- Sample count check: will check if any lights, GI or AO have unreasonable sample counts that can make GPU go out of available resources.
- Lightmapped prefab validation: validates Lightmapped Prefabs and notifies if some prefabs are going to be overwritten.

Advanced render settings

Light probe mode

Changes the way light probes are baked.

- Legacy: use Render Light Probes button to generate the probes. Point and directional lights are calculated within the lightmapper, while area/sky/indirect lighting is gathered by rendering a cubemap at each probe position. Results are stored as L2 spherical harmonics (https://docs.unity3d.com/ScriptReference/Rendering.SphericalHarmonicsL2.html). The downsides are slow cubemap rendering performance and possible mismatch between lightmaps and probes in cases where shaders in your game do not physically represent lit surfaces or your project is set up for mobile (Unity can clip high intensity values away).
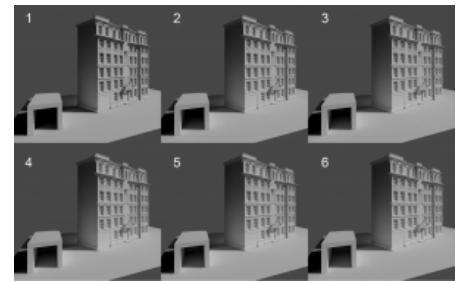
- L1: light probes will be rendered together with lightmaps when you click Render. This option provides superior baking performance and will guarantee that probe lighting matches the lightmaps. Results are stored as L1 spherical harmonics (can be still used by regular shaders). Results can be further improved by using Non-Linear Light Probe SH option in Bakery shader.

- L2: superior quality (full L2 spherical harmonics). As with L1, automatically rendered together with lightmaps. Negligible performance impact compared to L1. Should look OK with all shaders. Probe contrast/directionality/ringing removal can be tweaked via Project Settings -> Bakery -> "Ringing removal for L2 light probes".

There seems to be a bug in Unity 2019.3 preventing Legacy light probe colors from being properly saved IF occlusion probes option is enabled. L2 mode is now default and recommended when using occlusion probes.

Asset UV Processing

Configures UV padding adjustment for assets. Possible values:

- Don't change: don't touch the assets.
- Adjust UV padding: will look for models with auto-generated UVs ("Generate Lightmap UVs" on the asset) and adjust them further to have proper padding between UV islands per-mesh. Model-wide Pack Margin in importer settings is ignored. Optimal values are calculated instead, given area of each mesh and lightmap resolution.
- Adjust UV padding for new meshes only: same, but skips any models that were previously adjusted at least once.
- Remove UV adjustments: reverts all previous UV adjustments, makes auto-unwrapped models look they way Unity originally unwrapped them.

Denoiser

Sets the desired denoiser. Possible values:

- Optix 5: uses OptiX 5.1 AI denoiser (previously known as "Legacy denoiser"). Runs on the GPU. Supported on everything from Kepler (https://en.wikipedia.org/wiki/Kepler_(microarchitecture)) (typically GeForce 6xx) to Turing (https://en.wikipedia.org/wiki/Turing_(microarchitecture)) (typically Geforce 20xx). Not supported on Ampere (https://en.wikipedia.org/wiki/Ampere_(microarchitecture)) (30xx). OptiX 5.1 uses a static training dataset which is embedded into it, meaning it is not affected by the driver implementation.
- Optix 6: uses OptiX 6.0 AI denoiser (previously the default option). Runs on the GPU. Seems to fail on Kepler (6xx) GPUs, but runs on everything newer, including Ampere (30xx). Since 6.0 the training dataset for the OptiX denoiser is in the driver. Before driver v442.50 its behaviour was similar to OptiX 5.1; however, after it NVIDIA has changed the dataset (https://forums.developer.nvidia.com/t/optix7-denoiser-different-results-with-different-version-of-drivers/117989/2), and results can be different. The updated dataset may sometimes produce grid-like patterns and more bright edges (still fixable with "Denoise: fix bright edges"), but it runs faster.
- Optix 7: uses OptiX 7.2 AI denoiser. Behaves similarly to OptiX 6.0, but might be better supported on Ampere (30xx).
- OpenImageDenoise: uses Intel Open Image Denoise library. Runs on the CPU (any CPU that supports SSE 4.1). Can be slightly slower than OptiX, but the quality is comparable.
- OpenImageDenoise2: uses Intel Open Image Denoise 2, CUDA mode. Runs on the GPU, performance is comparable to OptiX 6/7, quality is identical to the CPU version.

Adjust sample positions

Find best sample positions to prevent lighting leaks. Details of the algorithm are outlined here (https://ndotl.wordpress.com/2018/08/29/baking-artifact-free-lightmaps/#leaks). In some cases (usually large and very low poly geometry with smooth normals) it may produce incorrect results, in which case you can disable it.

Unload scenes before render

Unloads Unity scenes before baking to free up video memory. Complex scenes can take a few gigabytes of VRAM by simply being shown in editor.

Denoise

If enabled, will apply denoising algorithm. Bakery uses Nvidia's AI denoiser (https://developer.nvidia.com/optix-denoiser).
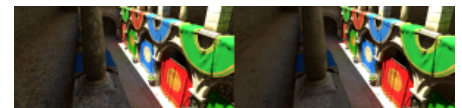
Left: denoising off, right: denoising on.

Fix Seams

If enabled, will attempt to blend seams created by UV discontinuities. Useful for smooth geometry, including Unity's default sphere.

Split by scene

If multiple scenes are loaded at once, and this option is enabled, each scene will have its own set of lightmaps, not shared with others. It can be useful for limiting the amount of textures loaded when scenes are streamed at runtime.

Hole filling

If Atlas Packer is set to xatlas, will try to fill every hole, resulting in more efficient atlases. For scenes with very complex geometry it can increase scene export time, but otherwise recommended.

Left: seams, right: seams are fixed.

Min resolution

Minimum lightmap size limit. Can be used to balance between many small but fully occupied lightmaps vs few incompletely filled.

Scale per map type

Allows you to scale the resolution of color/shadowmask/direction maps differently. For example, a common case would be to have low resolution indirect color but highly detailed shadowmask. Note that scaling is applied after the lightmaps are rendered, so it won't save baking time. If "Adjust UV padding" is on, padding will be based on the lowest resolution map to prevent texel leaking.

Hole filling off vs on.

Checker preview

If Show checker checkbox is on, Scene View will render a checkerboard pattern on top of visible objects to demonstrate lightmap texel size. This is useful to make sure you are using adequate values for Texels Per Unit and other resolution-affecting settings before you bake.

Enabling checker preview will force Bakery to perform atlas packing. It can take some time, but shouldn't be longer than a few seconds. Press Refresh checker to re-atlas the scene after you changed something to see changes.

Checker preview also uses random colors to show how the scene will be split into different lightmaps.

> 💡 Checker preview currently does not show correct texel sizes for Terrains.

Checker preview in action

Atlas preview

Atlas preview button is visible when Checker preview is on. Clicking it will open a new window where the UV layouts of future lightmaps are shown. Click "<" ">" buttons to browse the lightmaps. This is useful for checking out packing quality and lightmap count before baking.

Emissive boost

Multiplies any surface emission by this number.

Indirect boost

Multiplies all bounced lighting by this number. Same as Indirect Intensity on light components, but global.

Atlas preview in action

Backface GI

Determines how much light is passed through front faces to back faces and then bounced off by GI. This is especially useful for thin translucent surfaces like leaves. Values are in 0-1 range.

Backface GI

Ambient occlusion

Simple non-physical ambient occlusion you can apply over final scene lighting for aesthetic purposes.

- Intensity: controls visibility of the AO effect. Value of 0 disables the effect.
- Radius: determines the ray distance used in the AO effect. Smaller values give localized occlusion (corners, wrinkles), while larger values make it more similar to Skylight, giving shadows from distant objects.
- Samples: affects the quality of ambient occlusion. Typical values are from 4 to 32.

Dither transparency

Enables semi-transparent shadows by automatically dithering alpha textures. See how Bakery interprets opacity maps.

Left: backface GI = 0, right: backface GI = 1. Note how shadowed areas receive more green lighting.

RTX Mode

Enables RTX hardware acceleration. Only RTX GPUs will benefit from this option. Minimum supported driver version is 418.

Drivers can emulate RTX mode on most non-RTX Nvidia cards, but the result is usually slower.

RTX mode **must** be enabled on Ampere (3xxx) cards.

Left: AO intensity = 0. Right: AO intensity = 1.

Export terrain trees

If enabled, painted terrain trees will affect lighting. Trees themselves will not be baked. Note that highest possible LOD level is used for every tree during baking. It is not recommended to use this option for rendering multi-kilometer forests with highly detailed models.

Terrain optimization

If enabled (default), Terrains use separate ray tracing technique to take advantage of their heightfield geometry. Otherwise they are treated like any other mesh. Terrain optimization significantly reduces memory footprint required for high-resolution landscapes. In some cases it can also make terrain baking faster, in other cases (especially with simple low-resolution landscapes) it might make sense to disable it.

> 💡 Terrain Optimization is not currently compatible with painted terrain holes. Disable it to make them work.

Light Mesh batch limit

If this value is above 0, combines many Light Meshes into one, given they have identical settings and affect the same Lightmap Groups, but limiting their total sample count to this value (I.e. if the limit is 64, it can combine 2 lights with samples=32 or 4 lights with samples=16, etc). This will increase performance in scenes with many Light Meshes. Be careful setting this value too high, as the GPU may hit the OS driver timeout while processing too many samples at once.

Compress volumes

If enabled, applies texture compression to volume 3D textures and switches Bakery shaders to a corresponding sampling mode. Not recommended for very low resolution volumes. Volume size may be increased to be a multiple of 4.

Notes:

- Currently it can only compress to BC6H/BC7 texture formats (desktop/consoles).
- Volume compression is only supported on Unity 2020.1 or newer.
- Currently only Bakery shader for the standard rendering pipeline supports using compressed volumes.

Samples multiplier

Multiplies all shadow and GI samples by the specified factor. Use it to quickly change between draft and final quality.

Tile size

Same as GPU Priority, but instead of abstract priority you set tile size. Bakery splits lightmaps into smaller tiles and updates one at a time. Smaller size means more GPU work interruptions, and can make system more responsive.

Temp path

Temporary files folder. Bakery may require a few gigabytes of free space during rendering. Having this folder on SSD can make rendering slightly faster comparing to HDD. At the moment, this folder is not cleaned automatically. You can delete its contents anytime (except during rendering).

Output path

Lightmap folder. This is where all lightmap, and also lightprobe and vertex color assets will be saved. This path is relative to your Assets folder.

Use scene named output path

If this option is enabled, output path will be automatically set to a folder with the same name as the currently active scene (Assets/CurrentSceneName/). This is similar to how built-in Unity lightmappers behave.

Render Selected Groups

Only bakes Lightmap Groups containing selected objects. Not only selected objects will be redrawn, but all Lightmap Groups they belong to. Other lightmaps will not be updated. If objects are not a part of some manually assigned group, then a part of the scene using the same lightmap will be rebaked.

Beep on finish

If enabled, Bakery will play a sound when the bake is finished.

Experimental render settings

Unwrapper

If Adjust UV Padding is enabled, defines the unwrapper that will be used.

- Default: standard Unity unwrapper. Bakery will call Unwrapping.GenerateSecondaryUVSet (https://docs.unity3d.com/ScriptReference/Unwrapping.GenerateSecondaryUVSet.html) for every mesh with different padding parameters.
- xatlas: uses xatlas (https://github.com/jpcy/xatlas) by jpcy (https://github.com/jpcy), a modified version of thekla_atlas (https://github.com/Thekla/thekla_atlas) by Ignacio Castaño (https://github.com/castano). thekla_atlas was used in The Witness (blog post (http://the-witness.net/news/2010/03/graphics-tech-texture-parameterization/)).

Atlas Packer

Selects the algorithm to use when packing different object UV layouts into large lightmap atlases.

- Default: original algorithm Bakery used before v1.7.
- xatlas: uses xatlas.

Some features are only supported with one atlas packer:

	Default	xatlas
Override resolution	Yes	No
Efficient LOD packing	No	Yes
Hole filling	No	Yes

Note that atlas packer can be also chosen separately for every Lightmap Group.

Export geometry and maps

If enabled, Bakery will export the scene to its format before rendering. If you are sure that geometry and textures of the scene and lightmap resolution settings were not changed (e.g. you are only tweaking GI or light settings), you can disable this checkbox to make next rendering faster.

Update unmodified lights

If enabled, Bakery will recalculate light sources that didn't change since last rendering. If you are only tweaking one light and don't want to wait for other lights to re-render, you can disable this checkbox.

Update modified lights and GI

If enabled, Bakery will recalculate GI and lights that were changed since last rendering.

UV padding: increase only

Only visible if Adjust UV padding is on. By default, optimal UV padding for a model asset is calculated only based on currently loaded scenes. If there are multiple instances of the same model using different lightmap resolution, the smallest one will define padding, so spacing between UV charts is large enough to prevent them from leaking over each other. However, baking the same model in 2 different isolated scenes will possibly break UVs in previous scenes while optimizing for the new one. This checkbox allows to prevent such behaviour by never decreasing the padding value, so it will be always optimized for the lowest resolution instance ever baked.

Denoise: fix bright edges

Only visible if Denoise is on. Sometimes the neural net used for denoising may produce bright edges around shadows, like if a sharpening effect was applied. If this option is enabled, Bakery will try to filter them away. Denoising stage may get slightly slower, when enabled.

This feature may also be used to filter "fireflies", i.e. occasional bright dots from the lightmap.

Left: a bright edge is visible after denoising; Right: fixed.

Post-packing

Sometimes when rendering a scene, Bakery can split the lightmap into multiple sub-lightmaps, e.g. when baking different LODs or terrains. Post-packing tries to minimize final atlas count by combining these sub-lightmaps in one texture. It is recommended to keep this option enabled.

Combine with Enlighten real-time GI

If enabled, then when the Render button is pressed, it will first try to bake with Enlighten to calculate real-time GI. After that, regular Bakery lightmapping process will happen. Both baked and real-time GI will work together.

Bake on remote server

Enables network baking.

Presets

Currently shown under Render/Update buttons when settings are in Experimental mode.

- Click **New** to save all current settings to a preset file.
- By default, when a preset is active, all settings are grayed out to prevent overwriting the preset.
- Click **Modify preset** to be able to change settings and overwrite the preset.
- Presets can be swapped via the preset field. Disable preset usage by setting it to None.

Components

Bakery Lightmap Group Selector

Specifies which **Lightmap Group** to use for the object and all of its children. Lightmap Group is a term Bakery uses for a collection of objects sharing one lightmap. Groups have properties telling Bakery how to pack the objects, which lights should affect them, should the result be baked into a texture or vertices. By default all static objects are automatically packed into multiple lightmaps groups (atlases), so you don't have to worry. Defining groups manually should be only used for special purposes, for example:

- Marking certain objects use per-vertex lightmap instead of textures.
- Baking a lightmap using exact unscaled UVs as they were in a modeling package.
- Grouping certain areas of the level to use single lightmap to facilitate resource streaming or to switch different baked lighting at runtime via scripts.
- Using Render Selected Groups button to only update grouped objects.

To manually define a group, you create **Lightmap Group Assets**, either by using Assets->Create menu, or using the Create New button in the component. When using the Create New button, new asset will be created based on these forms:

- Name: name of the Lightmap Group Asset to create.
- Packing mode: this selector defines the packing mode of the Lightmap Group. There are 3 modes:
  - Original UV: object and its children will be baked with unmodified UVs. No packing is performed. This is useful for models when multiple meshes share the same already packed UV layout.
  - Pack Atlas: object and its children are packed into a dedicated texture atlas. Before v1.3 every packable child needed to have the component, but it's no longer necessary. Every child will be packed as its own rectangle, except when Bakery auto-detects a child having non-overlapping sub-children, in which case

multiple objects can use a single rectangle.

- Vertex: object and its children will use vertex-colored baked lighting instead of textures. Note that you'll need a custom shader for this to work, like Bakery Shader. One simple shader ("Bakery/Simple Vertex Lightmapped") is also included for demonstration and reference. Learn more about vertex lightmaps.

- Directional mode: allows you to override directional mode on the group. Options are the same except for Auto, which will simply use the global setting.
- Resolution: desired lightmap resolution.
- Auto-resolution: use Texels Per Unit to determine closest power-of-two resolution.
- Atlas packer: select atlas packing algorithm for this group.
- Bitmask: a list of toggles used to exclude light sources from affecting the lightmap. Every Bakery light source also has a bitmask setting. Lights only affects Lightmap Groups with which they share at least one of these toggles. Default settings mean all lights affect all lightmaps.

If you have a Lightmap Group with the Pack Atlas mode assigned, additional settings will appear on the component:

- Override resolution: override the resolution this object and its children occupy in the lightmap.
- Resolution: manually defined resolution. For example, if you have a lightmap with a resolution of 512 and it is assigned to 4 objects with override resolution set to 256, Bakery will generate a single 512x512 lightmap where each object takes exactly 256x256 square.

If you select a Lightmap Group asset, additional experimental settings will be visible:

Transparent selfshadow OFF vs ON: note how some single-sided but back-facing polygons become properly lit

- Subsurface scattering: (click to read more)
- Normal offset: allows you to offset rays from the surface. This is rarely useful and mostly present for experimental purposes.
- Transparent selfshadow: start rays behind the surface so it doesn't cast shadows on self. Might be useful for translucent foliage.
- Vertex sampling density (only if Packing Mode is set to Vertex): by default per-vertex bakes only compute 1 value for every vertex; if sampling density is > 0, it will instead average many samples scattered across triangles, yielding super-sampled, smoother results.

Bakery Lightmapped Prefab

This component allows prefabs to store lightmapping metadata. Such prefabs can be then instantiated in any scene, both in editor and at runtime. The component should be added directly to the root object of the prefab. Nested prefabs are not supported. Because the prefab is going to be overwritten after bake (the same way as when the "Apply" button is pressed), it is required that the prefab does not have any unapplied changes (child objects shouldn't be moved, materials and script parameters should be unchanged, etc). If such changes are detected, an error will be printed in the component UI and no metadata will be saved after bake. If Lightmapped prefab validation warning is enabled, all prefab errors will be also shown in a dialog box before the bake.

Lightmapped prefabs support all Bakery features, such as LODs, terrains, directional lightmaps, RNM, SH, per-vertex modes and shadowmasks. For shadowmasks to work, you need lights to also be a part of the same prefab.

💡 Note that prefab shadowmasks only work on Unity 2017.4 or newer due to API limitations.

After baking, an object named "BakeryPrefabLightmapData" will be added to prefab. This object holds a script with all necessary data to apply the lightmaps.

Additionally, Lightmapped Prefab allows to save current render settings and load them back to any currently open scene, using two buttons:

- Save current render settings to prefab: stores a copy of current Render Settings on this prefab.
- Load render settings from prefab: sets current Render Settings to those stored on the prefab.

When baking a Lightmapped Prefab, it is recommended to check that:

Applying the prefab

- Prefab is "Applied". On older Unity versions, click "Apply" button on the top right corner on the Inspector. On newer versions click the "Overrides" list and then "Apply all".
- Prefab component UI does not show any errors.
- [Recommended] "Lightmapped prefab validation" checkbox is enabled in the main window. Clicking Render you doesn't show any warning dialog boxes about the prefabs.
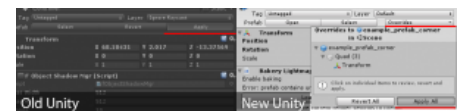
💡 Lightmaps are named after the scene they are baked in or a Lightmap Group they use (if any). When baking different prefabs in the same scene, make sure to either use a differently named Lightmap Group or change the Output Path.

💡 When baking multiple prefabs using identical models one by one, it is recommended to set Asset UV Processing to *Don't Change*, to prevent alteration of shared model UVs.

💡 Bakery's prefab workflow was implemented before Unity had a "double-click into prefab" view feature. Thus, lightmaps may be not correctly applied in the isolated prefab view, and the component may show a "This GameObject is not a prefab" message. It should work as expected when working in a normal scene.

💡When unpacking previously baked prefabs and unlinking them from existing lightmaps, make sure to remove the BakeryPrefabLightmapData object which holds the prefab-wide lightmap/renderer connections.

Bakery Direct Light

Infinitely distant directional light (e.g. Sun).

- Color: color.
- Intensity: linear color multiplier.
- Shadow spread: size of the light source or, simply put, shadow blurriness. 0 = max sharp shadows, 1 = max blurred shadows. Technically it interpolates from a single ray to a wider cone of rays.
- Shadow samples: affect shadows quality. The smaller the Shadow Spread value, the less samples are needed for a clean image. Typical values for sun shadows are from 1 to 16. Setting samples to 0 will make shadows from this light disappear.
- Bitmask: a list of toggles used to exclude this light source from affecting particular Lightmap Groups.
- Baked contribution: determines what kind of lighting data should be baked. Only visible in Indirect or Shadowmask modes. Possible values:

  - Direct and Indirect: both direct and indirect contribution will be completely baked.
  - Indirect only: a realtime light is supposed to provide direct contribution; only indirect lighting is baked.
  - Shadowmask and indirect: a realtime light is supposed to provide direct contribution, but shadows from static geometry will be baked to a separate map. Dynamic and static shadows will be mixed together. Indirect lighting will be baked as usual.
  - Direct, Indirect, Shadowmask (custom lighting only): same as Direct and Indirect, but the Shadowmask is also produced and assigned to use with custom shaders.

- Denoise shadowmask: determines if denoising should be applied to this light's shadowmask. Usually it's not necessary as shadows don't exhibit much noise, but can be useful for very wide blurry shadows (large shadow spread).
- Indirect intensity: non-physical GI multiplier for this light. Should be 1 for natural lighting, but can be modified for more stylized scenes.
- Anti-alias: performs super-sampling for the shadow.
- Texture projection: projects a multiplicative tiled texture over lighting. Can be used to emulate distant cloud shadows.

Pure direct light with GI
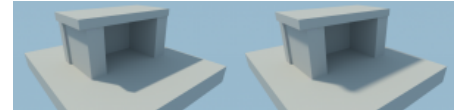
Left: shadow spread = 0.01, right: 0.05. Note how shadows start sharp and get gradually blurrier with distance.

When texture projection is set, its scale/offset can be manipulated using "Tiling" and "Offset" fields. Alternatively, click "Tweak projection in Scene View" and use additional gizmo to move and scale the projection visually. Click "Render reference frame" to save a PNG of "light's point of view". This is useful to paint cloud shadows over the scene.

If the same object has both Unity and Bakery light sources enabled, and they don't match, two buttons will appear:

- Match lightmapped to real-time: copy common settings from Unity light to Bakery light.
- Match real-time to lightmapped: copy common settings from Bakery light to Unity light.

> 💡 In physical terms, Bakery's "Intensity" for directional lights is Lux divided by Pi.

Bakery Sky Light

Infinitely distant shadowed ambient light (sky).

- Color: color.
- Intensity: linear color multiplier.
- Sky texture: optional cubemap asset, e.g. HDRI panorama to affect lighting colors from different directions.

> 💡 Rotating sky light's GameObject will rotate lighting from the cubemap accordingly.

- Samples: affects shadows quality. Typical values are from 8 to 32.
- Hemispherical: if enabled, lighting will only come from above (upper hemisphere). Otherwise from all directions.
- Bitmask: a list of toggles used to exclude this light source from affecting particular Lightmap Groups.
- Baked contribution: determines what kind of lighting data should be baked. Only visible in Indirect or Shadowmask modes. Possible values:

  - Direct and Indirect: both direct and indirect contribution will be completely baked. This is the default and recommended behaviour for all Sky Lights, as there is no real-time version to emulate them in Unity.
  - Indirect only: a realtime light is supposed to provide direct contribution; only indirect lighting is baked.

Pure sky light with GI

If sky light's settings and current scene skybox don't match, two buttons will appear:

- Match this light to scene skybox: copy common settings from active skybox material to Bakery sky light. Currently it can only match skyboxes using the "Skybox/Cubemap" or "Skybox/Bakery skybox" shaders.
- Match scene skybox to this light: copy common settings from Bakery sky light to scene skybox.

Bakery Light Mesh

Emissive mesh of any shape. Should be used together with the Mesh Renderer component or with a Light component set to Area mode.

- Color: color

- Intensity: linear color multiplier.
- Cutoff: maximum lighting distance. Bakery will additionally attenuate correct inverse-squared falloff near the limit. It works best when cutoff value is just where physical falloff gets very dim. Cutoff can improve baking performance, but you also can set it to a very high (unreachable) value for complete correctness.
- Self Shadow: determines if light mesh itself casts shadows. This option also enables a more precise lighting algorithm.
- Samples Near: affect lighting quality near the mesh. Typical values are from 16 to 64. This option is only available when Self Shadow is on.
- Samples Far: affects lighting quality far away from the mesh. Typical values are from 4 to 4096.

Left: Samples Near = 0, Samples Far = 4096.
Middle: Samples Near = 0, Samples Far = 128.
Right: Samples Near = 16, Samples Far = 128.
Note how the lighting appearance is similar, yet less combined samples are used.

Bakery mixes two different algorithms for area lights based on emissive surface proximity. Close to the light it works similar to the GI algorithm, while at a distance the light is approximated as a cloud of virtual point lights.

💡

- Light Meshes don't receive lighting on their own and don't get lightmapped.
- One Light Mesh with many polygons is faster to bake than many simple Light Meshes.
- If you keep getting dirty/noisy results, try setting Samples Near to 0. Then only the VPL algorithm is used. It is also useful for very simple lights where precision is not important (e.g. window lights on a building).

- Bitmask: a list of toggles used to exclude this light source from affecting particular Lightmap Groups.
- Baked contribution: determines what kind of lighting data should be baked. Only visible in Indirect or Shadowmask modes. Possible values:
  - Direct and Indirect: both direct and indirect contribution will be completely baked. This is the default and recommended behaviour for all Light Meshes, as there is no real-time version to emulate them in Unity.
  - Indirect only: a realtime light is supposed to provide direct contribution; only indirect lighting is baked.
  - Shadowmask and Indirect: a realtime light is supposed to provide direct contribution, but shadows from static geometry will be baked to a separate map. Dynamic and static shadows will be mixed together. Indirect lighting will be baked as usual. Currently only HDRP (and custom renderers) support mixed area lights.
  - Direct, Indirect, Shadowmask (custom lighting only): same as Direct and Indirect, but the Shadowmask is also produced and assigned to use with custom shaders.
- Indirect intensity: non-physical GI multiplier for this light. Should be 1 for natural lighting, but can be modified for more stylized scenes.
- Shadowmask with falloff: bakes complete light attenuation into the shadowmask (only useful with custom shaders).

A curved Light Mesh

If mesh material and light settings do not match, two buttons will appear:

- Match light to material: copy common settings from unlit mesh material or area light to Bakery light.
- Match material to light: copy common settings from Bakery light to unlit mesh material or area light.

Differences between Light Mesh and emissive materials

Emissive material:

- Propagates light during GI calculation, therefore is "free", no matter how many meshes are emissive.
- Small and bright lights can give inaccurate noisy results, depending on GI sample count.
- Has to occupy space in lightmap due to the way GI in Bakery works, therefore possibly wasting it. Also it means that emissive objects require valid UVs and have "Cast shadows" enabled..

Light Mesh:

- Lighting is calculated separately for each light, therefore is slower than emissive materials.
- Designed to give clean results for small and bright lights, even very far away from them.
- Doesn't take space in lightmaps.

Bakery Point Light

Point light, doesn't have area. Despite physical impossibility it is useful for games and the only way to match Unity's point and spot lights.

💡Spot lights are just point lights with additional projection mask.

Left: Shadow Spread = 0.5, right: Shadow Spread = 1.

- Color: color.
- Intensity: linear color multiplier.
- Shadow spread: shadow blurriness. Although point lights don't have area, it is often desired to still have blurred shadows from them. For this purpose shadows are simulated as if they were cast from a spherical area light. This parameter defines the radius of such virtual sphere, directly affecting blurriness.
- Physical falloff: if enabled, will use correct inverse-squared falloff with additional attenuation near the limit. Otherwise will try to mimic Unity's falloff.
- Falloff min size: learn more in Point Light Attenuation section.
- Range: if physical falloff is disabled, equals to Unity's light range. If it's enabled, will only slightly fade away physical falloff at the edge.
- Samples: affects shadows quality. Typical values are from 1 to 512. Setting samples to 0 will make shadows from this light disappear.

- Projection mask: type of mask/cookie. Options:

    - Omni: no mask, equals to Unity's Point Light.
    - Cookie: cookie texture projection. Additional options:

        - Cookie texture: 2D texture asset. Bakery includes the original Unity's Spot Light texture (named ftUnitySpotTexture) that you can use to mimic it.
        - Angle: texture projection angle (similar to Unity's Spot Light).
        - Correct distortion: corrects cookie distortion visible on straight lines at wide angles.
    - Cubemap: cubemap cookie projection. Additional options:

        - Projected cubemap: cubemap asset.
    - IES: lighting is modulated by the IES file data. Additional options:

        - IES file: file with .ies extension.

Left to right: Omni, Cookie (default spot texture), Cubemap, IES.

- Bitmask: a list of toggles used to exclude this light source from affecting particular Lightmap Groups.
- Baked contribution: determines what kind of lighting data should be baked. Only visible in Indirect or Shadowmask modes. Possible values:

    - Direct and Indirect: both direct and indirect contribution will be completely baked.
    - Indirect only: a realtime light is supposed to provide direct contribution; only indirect lighting is baked.
    - Shadowmask and Indirect: a realtime light is supposed to provide direct contribution, but shadows from static geometry will be baked to a separate map. Dynamic and static shadows will be mixed together. Indirect lighting will be baked as usual.
    - Direct, Indirect, Shadowmask (custom lighting only): same as Direct and Indirect, but the Shadowmask is also produced and assigned to use with custom shaders.
- Indirect intensity: non-physical GI multiplier for this light. Should be 1 for natural lighting, but can be modified for more stylized scenes.
- Shadowmask with falloff: bakes complete light attenuation (except for distance) into the shadowmask (only useful with custom shaders).
- Legacy sampling: use Bakery's original more biased shadow sampling strategy. Produces noise-free shadows, but wide penumbras can exhibit banding. If disabled, an unbiased, but noisier technique is used.

If the same object has both Unity and Bakery light sources enabled, and they don't match, two buttons will appear:

- Match lightmapped to real-time: copy common settings from Unity light to Bakery light.
- Match real-time to lightmapped: copy common settings from Bakery light to Unity light.

> 💡 In physical terms, Bakery's "Intensity" for points lights is (Candela/Pi).

Bakery Volume

Volumes generate 3D textures that store light probes per voxel (L1 spherical harmonics and shadowmasks) making them a viable replacement for regular Unity light probes.

	Unity Light Probes	Unity LPPV	Bakery Volumes
Manually placing each probe	Yes	Yes	No
Interpolate multiple probes on one object	No	Yes	Yes
Movable at runtime	No	No	Yes
Can put in a prefab	No	No	Yes
Full scripting API	No	No	Yes
Ringing	Possible	Possible	Fixed
Denoising	No	No	Yes
Runtime overhead	CPU: low; GPU: low.	CPU: medium; GPU: medium.	CPU: none; GPU: medium.

Bakery Volume is a bounding box linked to a set of 3D textures (3 SH textures and one shadowmask, if needed), making it very easy to use and script. Volumes can be trivially swapped, moved, loaded or unloaded at runtime or linked to prefabs. They also do not need any Light Probe Groups and manual probe placement; the whole volume is filled with a uniform grid of probe voxels. Light/shadow leaking is automatically fixed the same way it is done for lightmaps (although thin double-sided walls may be a problem with large voxel size, but it can be avoided by using different volumes on different sides). Because of the uniform nature of voxel grids, it is also possible to apply lightmap-like denoising to them, which is not possible with regular light probes. Additionally, Bakery shaders use the Geomerics sampling trick (http s://drive.google.com/file/d/1MK6Clw93q8ljijRjb4gQOzzaIjn1uF9Y/view) to completely get rid of the "ringing" artifact spherical harmonics can sometimes produce with high-contrast HDR lighting.

Because of that, Volumes can produce clean, detailed, high-quality baked lighting for dynamic objects (that is, the lighting that comes from the static objects and affects dynamic objects, not the other way around).

The only downside of the Volume approach is the higher per-pixel shading cost. The cost is similar to LPPV (scaling with the amount of shaded volume-affected pixels on the screen) but unlike LPPV, the Volumes themselves are never updated once they are baked. Multiple volume blending can be avoided by slightly overlapping them instead.

Setting volume transform

When BakeryVolume is added to a GameObject, a bounding box gizmo will appear in scene view. Dragging it by the little squares will scale it. There are also other ways to change volume size and position:
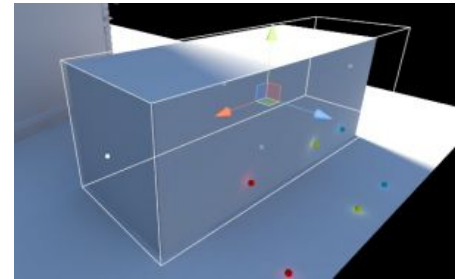
- Drag any renderable GameObject into "Wrap to object" field. The volume will wrap around this object (and its children) to encapsulate them.
- If there is a Box Collider added to the same object, "Set from box collider" and "Set to box collider" buttons will appear. These can be used to quickly copy values between Volume/Collider.

Volume gizmo

Resolution

When "Adaptive resolution" is enabled, the amount of voxels is governed by "Voxels per unit". Disabling it allows to manually enter the resolution. For most games with 1 unit = 1 meter, a reasonable "Voxels per unit" value is 2, but there are cases (e.g. dynamic doors) when using a smaller, but higher resolution volume is beneficial.

Other settings

- Enable baking: should the volume be (re)computed? Disable to prevent overwriting existing data.
- Denoise: apply denoising after baking the volume. Recommended for high-resolution volumes. Avoid using for very low-resolution volumes covering many lights, as it will be hard for denoisers to tell actual content from noise.
- Global: automatically assign this volume to all volume-compatible shaders, unless they have overrides. Internally it will call Shader.SetGlobalTexture()/SetGlobalVector().

Usage

Volumes can be assigned to objects using a volume-aware shader. Bakery comes with "Bakery Standard" shader that implements it, as well as a HDRP and URP graphs. See example_volumes scenes for various ways of applying a volume.

Hints

- Make sure to check the examples.
- If two volumes connect, make them overlap a bit. The size of the overlap should be equal to the size of the largest dynamic object. This way simply switching between volumes on the go will work.
- Only place volumes around areas dynamic objects can actually reach. E.g. use player height to limit volume height.

Rotating volumes

Volumes can be rotated around Y axis, if "Rotate around Y" option is enabled. Rotation is very cheap and only uses 2 extra floats of data (precomputed sin/cos of the angle).

There is also an additional option to rotate volumes at runtime (Experimental -> "Support rotation after bake" and a similar option in Bakery Shader), in which case they need to be baked with zero initial rotation.

In URP such dynamic rotation can be used with BakeryURPVolumeGraphRotatable.shadergraph.

For global volumes dynamic rotation will be updated when the component is (re)enabled or volume.SetGlobalParams() is called. Using local rotated volumes requires manually setting the rotation on the material or the MaterialPropertyBlock:

```
SetMatrix("_VolumeMatrix", volume.GetMatrix())
```

Technical information

Volumes can work automatically (check example scenes), but it is also possible to script the way they are used.

Volume data can be retrieved from the component via these public properties:

- bakedTexture0: stores L0 RGB coefficients and L1 Z red channel in alpha.
- bakedTexture1: stores L1 X RGB coefficients and L1 Z green channel in alpha.
- bakedTexture2: stores L1 Y RGB coefficients and L1 Z blue channel in alpha.
- bakedMask: stores the volumetric shadowmask.
- bounds: the bounding box of the volume.

All bakedTexture* maps use RGBAHalf (8 bytes per voxel) format and the mask is ARGB32 (4 bytes per voxel). Shadowmask can also use single-channel R8 format (1 byte per voxel).

Therefore volume byte size = width*height*depth*3*8 + (only if the shadowmask is used) width*height*depth*4 (or width*height*depth with an R8 mask).

Texture compression can be enabled on volumes to make them much smaller.

To sample the volume, a special shader is needed, and these are included in Bakery. To add volumes to a custom shader, check BakeryVolume_float() function in BakeryDecodeLightmap.hlsl which is included in HDRP/URP packages.

Bakery Light Filter

Marks a renderer as a Light Filter. Filters tint all direct/indirect rays with their color, similar to real-life optical filters, allowing for colored transparencies (e.g. stained glass). The only parameter is the texture used for tinting.

- Light Filter objects still receive lightmaps like everything else.
- Filters don't scatter/blur rays (for that try Subsurface Scattering or Backface GI).
- Filters can produce color-tinted shadows. If Shadowmask mode is used on the light, only R channel of the result will be saved (as Unity only supports single-channel masks). For fully baked shadows there are no limitations.

Bakery Sector

See **Partial scene baking**.

Bakery Always Render

If added to an object, Bakery will always take it into account when baking, even if its renderer is disabled.

Bakery Pack As Single Square

If added to an object, Bakery will treat UVs of this object and its children as a single combined square and never move them away from each other during automatic atlas packing or in a Lightmap Group with Pack Atlas mode. Whole children hierarchy will be packed as one.

Bakery Shared LOD UV

If added to a LODGroup, will mark all similarly-named LODs as sharing the same lightmap. By default, every LOD mesh receives a unique lightmap, taking space in the atlas; If, however, it is known that LODs of the same object have similar UV2 (or UV1, if it's non-overlapping, and UV2 is not present), they can map to the same part of the atlas, saving memory. This technique is also known as Lightmap Stealing (https://github.com/laurenth-personal/LODLightmapScripts/).

Material compatibility

Albedo and emission

Bakery supports most Unity materials by utilizing the Meta Pass (https://docs.unity3d.com/Manual/MetaPass.html) system. All built-in, surface, URP and HDRP shaders already have a correct meta pass defined.

If you have an Unlit shader, add meta pass manually, as described in Unity docs.

If the pass is not present, Bakery will assume **_MainTex**, **_BaseColorMap** or **_Color** (if no texture is set) properties as material albedo and **_EmissionMap** and **_EmissionColor** as emissive color.

In order for emission to work, Material.globalIlluminationFlags (https://docs.unity3d.com/ScriptReference/Material-globalIlluminationFlags.html) must be set to MaterialGlobalIlluminationFlags.BakedEmissive (https://docs.unity3d.com/ScriptReference/MaterialGlobalIlluminationFlags.BakedEmissive.html). Standard shaders set this value automatically.

Opacity

By default, Bakery always takes full resolution opacity from the alpha value of **_MainTex**, **_BaseColorMap** or **_BaseMap** and treats it as cutout (1 bit tranparency).

When **Dither Transparency** option is enabled, semi-transparent values are automatically converted to variable blue noise, which is then filtered/blurred, resulting in partial transparency (it was previously possible using custom manually dithered maps, but since v1.97 it is automatic).

In order for opacity to work, RenderType tag (https://docs.unity3d.com/Manual/SL-SubShaderTags.html) of the shader must be one of these values:

- **Transparent**
- **TransparentCutout**
- **TreeLeaf**

If a property named **_Cutoff** is present, it will affect the cutout (therefore Standard shader cutoff slider works automatically).

Without Dither Transparency, if material RenderType is **Transparent** or **TreeLeaf**, then alpha value of **_Color** will additionally modify the cutoff: *finalCutoff = 1 - (1 - cutoff) * alpha*

For colored transparency, use the Light Filter component.

For diffuse translucency effects read about Backface GI and subsurface scattering.

Alpha Meta Pass

Since v1.9 Bakery also supports specifying custom shader-based transparency, but it requires a compatible shader. To take advantage of *Alpha Meta Pass*, following steps must be taken:

- Add a property named **BAKERY_META_ALPHA_ENABLE** to the shader. For example:
```
[HideInInspector] BAKERY_META_ALPHA_ENABLE ("Enable Bakery alpha
meta pass", Float) = 1.0
```
  - Make sure that RenderType tag is set to one of the values above.
  - Declare a pass named **META_BAKERY**.
  - In that pass check if **unity_MetaFragmentControl.w** is non-zero, in which case, return the computed transparency value.

See *Baked_Alpha_meta.shader* included in the package as an example.

Note: currently Alpha Meta Pass requires Light Probe Mode to be set to L1 or L2.

Normal mapping

Custom surface normals can be used in conjunction with Baked Normal Maps mode. Unfortunately, Unity's Meta Pass does not output normal information, so by default normals are taken from a texture named **_BumpMap**, additionally transformed by **_MainTex_ST**.

Material parameters converted to cutout (default)

Semi-transparent values added via Dither Transparency option

Light Filter components used on red planes, producing colored shadows

It is however possible to make completely custom shaders with advanced normal mapping features (e.g. blending multiple layers) compatible with baking. To do that you'll need:

1. Define a Meta Pass in your shader, just as described in Unity docs (https://docs.unity3d.com/Manual/MetaPass.html#ExampleMetaPass).

2. Name it **"META_BAKERY"**, instead of just **"META"**.

3. Include **"BakeryMetaPass.cginc"**.

4. Add **#pragma vertex vert_bakerymt**

5. If **unity_MetaFragmentControl.z** variable is not 0, shader should return world-space normal. Returned normal should be encoded using **BakeryEncodeNormal** function.

Two example shaders implementing this extended Meta Pass are included in Assets/Bakery/examples/shaders.

You can copy and modify BakeryMetaPass.cginc to suit your needs, e.g. pass additional data from the vertex shader.

Front/back faces

No matter what culling mode is used on the shader, Bakery treats all back-faces as opaque, meaning any direct light ray will not pass through a back-face, just as it will not pass through a front-face. However, back-faces can partially emit lighting received by a front-face, if you use Backface GI.

"Lightmap Parameters" assets

Similar to Bakery's Lightmap Groups, Unity has Lightmap Parameters (https://docs.unity3d.com/Manual/class-LightmapParameters.html) that can be created as assets and assigned to Mesh Renderers. Their logic has differences and similarities:

- Lightmap Group's primary usage is to group multiple objects in one atlas. Additionally it allows overriding baking settings for this atlas.
- Lightmap Parameters are primarily used to override baking settings for an object. But additionally they can influence packing by using the "Baked Tag" option: objects with a different Baked Tag will never share the same atlas.
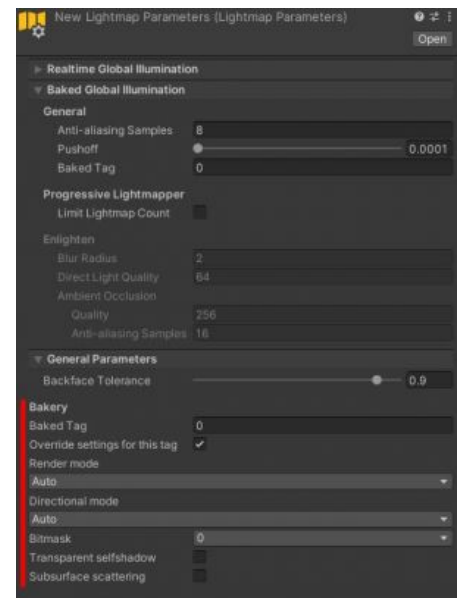
Since some users are more used to the Lightmap Parameters workflow, Bakery now supports and extends them:

- Baked Tag is respected by Bakery.
- Additional Bakery-specific options will be shown in the Lightmap Parameters UI. Refer to the Lightmap Group section for their meaning.

Extended Lightmap Parameters

Project Settings

Some global settings can be tweaked in Project Settings -> Bakery GPU Lightmapper (this option is only visible in Unity 2018.3 or newer; for older versions, edit BakeryProjectSettings.asset instead):

- Mipmap lightmaps: enables mipmapping on lightmap assets. Can cause leaks across UV charts as atlases get smaller; on the other hand, it reduces high resolution lightmap aliasing in the distance. Overall it is recommended to use separately baked LODs with lower resolution lightmaps for very distant objects instead.
- Color file format: format for regular HDR color lightmaps. Possible values:

  - HDR: Radiance HDR format (default). Can be opened in many image editors.
  - Asset: native Unity format. Can have custom mipmap count.
- Mask/Direction file format: format for additional LDR maps, such as shadowmasks or directional data. Possible values:

  - TGA: 8-bit Targa (default).
  - PNG: 8-bit PNG. Usually takes less space on disk compared to TGA.
  - Asset: native Unity format. Can have custom mipmap count.
- Compress lightmaps: defines lightmap asset compression behaviour. Compressed lightmaps take less VRAM, but may exhibit some visual artifacts (such as banding or colored blotches) depending on the type of compression (for mobile compression issues see this). Possible values:

    - Compress but allow overriding asset: new lightmaps will be created with compression enabled. After they are created, it is possible to change their compression settings in Inspector. These modified values will be kept after rebaking (default).
    - Force compress: always compress all lightmaps.
    - Force no compress: always produce uncompressed lightmaps.
- High quality direction: enables "high" quality compression for directional maps. Actual formats depend on platform. On PC, for example, it will change it from DXT1 to BC7, which gives higher quality, but doubles the VRAM usage.
- Texel padding (default atlas packer): defines how many empty texels to add between objects' UV layouts in lightmap atlases, when using the default atlas packer.
- Texel padding (xatlas packer): same, but when using xatlas.

- Alpha Meta Pass resolution multiplier: multiplies resolution of the Alpha Meta Pass allowing to have precise shader-based cutouts while baking low resolution lightmaps.
- Volume render mode: which render mode to use for all Volumes? Auto uses current active render mode (default).
- Delete previous lightmaps before bake: actually delete all previously rendered lightmap files from this scene before baking new ones.
- Log level: defines how much information is printed to console during the bake. Possible values:

  - 0: do not print anything.
  - 1: only print progress information.
  - 2: only print warnings.
  - 3: print everything.

- Alternative scale in lightmap: makes 'Scale in Lightmap' renderer property act more similar to built-in Unity behaviour.
- Align to texture compression blocks with xatlas: makes xatlas align charts to 4x4 block boundaries, which gives more compression-friendly results.
- Generate smooth positions: should sample positions be adjusted to prevent incorrect shadowing on low-poly geometry with smooth normals? Can be disabled in very rare cases when using geometry with wildly mismatching normals. See this article (https://ndotl.wordpress.com/2018/08/29/baking-artifact-free-lightmaps/#shadowterm) for technical explanation.
- Smooth positions per-triangle: should smooth/flat position be decided per-triangle? Can be disabled in very rare cases when using geometry with wildly mismatching normals.
- Use 'Receive GI' values: take 'Receive Global Illumination' values into account on renderers. Originally Bakery ignored it.
- Remove ringing in Legacy light probes: enables softer light probe convolution in Legacy mode to prevent artifacts in high-contrast areas.
- Always render reflection probes: automatically render reflection probes after every Render/Render Light Probes.
- Ringing removal for L2 light probes: applies de-ringing to L2 light probes. The higher the value, the smoother the probe lighting is; this smoothing reduces ringing (https://docs.unity3d.com/6000.2/Documentation/Manual/light-probes-troubleshooting.html) artifacts (if present).
- Put menu under Tools: puts Bakery menu under the Tools menu.

If any format is set to Asset, and mipmapping is enabled, an additional option is shown:

- Maximum mipmap count: defines maximum amount of mipmaps. For example, if set to 2, a 8K lightmap will only have two additional mipmaps: 4K and 2K. Lower resolutions will be omitted, thus limiting leaking artifacts during mipmapping.

Skinned mesh renderer support

Bakery supports skinned meshes since v1.65. However, there are some limitations:

- Some versions of Unity do not support UV1 on skinned meshes and silently ignore "Generate lightmap UVs". Make sure the actual mesh used in your scene has UV1 and use non-overlapping UV0 if possible;
- UV padding adjustment does not work on skinned meshes (Unity seems to crash when attempting to call Unwrap function on them);
- Skinned meshes can't have per-vertex lightmaps (engine has no support for additionalVertexStreams with skinning).

Shader Tweaks

Bakery offers optional shader modifications that can be applied to Unity shaders to improve lightmap sampling and match real-time lighting closer to lightmaps. Shader tweaking menu can be accessed via Bakery->Shader Tweaks.

These options will patch base Unity shader includes (from the CGIncludes folder) that are used by most built-in and third party shaders. All modifications will affect all projects opened with this version of editor, but can be also easily reverted back. Shader tweaks affect the way standard and surface shaders work. HDRP, LWRP and other SRPs are not affected.

All available modifications are shown as buttons. Press the button to apply modification, and it will remain pressed. Un-press the button to remove the modification. Currently available options:

Left: default bilinear filtering. Right: bicubic filtering.

> 💡 You may need to run Unity editor as admin to make CGIncludes patching work

- Use bicubic interpolation for lightmaps: bicubic interpolation will be used instead of default bilinear. This tweak fixes many jagged edges of low resolution lightmaps pretty well. Currently it only works on DX11 and modern consoles, while other platforms will remain using bilinear.

- Use multiplication for shadowmask: combine static and dynamic shadows using multiplication instead of minimum. For reasons unknown Unity decided to use minimum operator by default, and it produces artifacts that are not present with multiplication.

- Use physical light falloff (forward): replace default attenuation of real-time point/spot lights with a more physically correct inverse-squared. "Range" will only slightly fade it out near the edge (matches Bakery lights with Physical Falloff checkbox). Only affects forward renderer. See Point Light Attenuation for more details.

- Use physical light falloff (deferred): same as above, but only affects deferred renderer.

Left: both shadows are baked.
Center: spherical shadow is dynamic, and other shadow is static. Combined together with multiplication.
Right: same as center, but combined with default minimum.

Falloff tweaks may not fully work on > 2017 Unity versions. Also, SRPs have correct falloff by default.

Bakery shaders

Bakery includes its own shaders with support for vertex lightmapping, as well as RNM and SH directional modes. It also allows approximate baked specular to be calculated from directional data. Bakery shader is **not required** for regular color lightmaps, shadow masks and Dominant Direction mode, as these features are supported by most Unity shaders anyway.

Standard render pipeline

Bakery/Standard: extends regular Standard shader. Additional options:

- Allow Vertex Lightmaps: allows this material to be used with vertex lightmapping.
- Allow RNM Lightmaps: allows this material to be used with RNM directional mode.
- Allow SH Lightmaps: allows this material to be used with SH directional mode.

> 💡 All "Allow" toggles enable a certain code path in the shader. It is recommended to only use one "allow" option in a material for performance reasons.

- Enable Lightmap Specular: calculates baked specular using data from Dominant Direction, RNM or SH directional modes. Note that due to the lack of information, the effect is very approximate. If you ever used baked specular in Unity 4, it should be comparable. It looks best together with highly perturbed normal maps, not so good on flat surfaces.
- Enable VertexLM directional: if vertex lightmaps are present, specifies that Dominant Direction data is stored in vertices and should be used.
- Enable VertexLM SH: if vertex lightmaps are present, specifies that SH data is stored in vertices and should be used.
- Enable VertexLM Shadowmask: if vertex lightmaps are present, specifies that shadow mask is stored in vertices and should be used.
- Enable MonoSH: treat directional maps as MonoSH maps.

- Non-linear SH: in SH directional mode this option can enhance contrast (closer to ground truth), but it makes the shader a bit slower. This trick is based on "Reconstructing Diffuse Lighting from Spherical Harmonic Data" (https://drive.google.com/file/d/1MK6Clw93q8ljijRjb4gQOzzaIjn1uF9Y/view) paper by Geomerics.
- Non-linear Light Probe SH: similar to Non-linear SH, but works for light probes instead. In case of probes the most notable benefit is getting rid of incorrect negative values. This problem often happens in high-contrast scenes with bright HDR light sources. It's an inherent limitation of spherical harmonics and not specific to Bakery (i.e. you can get similar artifacts with built-in lightmappers as well).
- Force Bicubic Filter: enables bicubic filtering for all additional maps. For complete bicubic filtering (including the first color map) it is recommended to also enable it in Shader Tweaks.

Standard SH light probe vs non-linear

Bakery/Standard Specular: same as Standard, but for specular workflow.

HDRP & URP

SH specular effect

To extract additional HDRP/URP shaders, open Bakery_ShaderGraphHDRP.unitypackage or Bakery_ShaderGraphURP.unitypackage included with Bakery. Inside you will find Shader Subgraphs that you can use in your own Shader Graphs as well as some example materials.

Feature support across shaders

	Most Unity shaders	Most HDRP & URP shaders	Bakery Standard	HDRP graphs	URP graphs
HDR color lightmaps	Yes	Yes	Yes	Yes	Yes
Shadowmask	Yes	Yes	Yes	Yes	Yes (Since URP 10.1)
Dominant Direction	Yes	Yes	Yes	Yes	Yes
RNM	No	No	Yes	Yes	Yes
SH	No	No	Yes	Yes	Yes
MonoSH	No	No	Yes	Yes	Yes
Lightmap specular	No	No	Yes	Yes for dominant direction and SH	Yes for dominant direction (see link)
Bicubic filtering	Yes, with Shader Tweaks	No	Yes, with Shader Tweaks	No	No
Vertex lightmaps	No	No	Yes	Single color, no per-vertex SH, direction or shadowmasks (shader graph limitation)	Single color, no per-vertex SH, direction or shadowmasks (shader graph limitation)
Non-Linear Light Probes	No	No	Yes	Yes	Yes
Volumes	No	No	Yes	Yes	Yes
Volume shadowmask	No	No	Yes	No	Yes
Compressed volumes	No	No	Yes	Yes	Yes

> 💡 Everything except for regular color lightmaps requires at least Shader Model 3.0 to work.

> 💡 Vertex Lightmaps with SH mode require at least Shader Model 4.0 due to interpolator limit.

> 💡 Updated URP graphs with specular from Dominant Direction can be downloaded from the main wiki page.

See also: Integrating Bakery shader features to custom shaders

Notes on HDRP/URP

Once the package is imported into an HDRP or URP project, in general it can be used right away, but there are some things to note:

- Example scenes use both regular Standard and, in some cases, "Bakery Standard" shaders. HDRP and URP provide a mechanism to update project shaders from Standard to Lit, but they cannot upgrade from Bakery Standard. Replace Bakery Standard materials with the ones coming from Bakery_ShaderGraphHDRP.unitypackage or Bakery_ShaderGraphURP.unitypackage.

- In HDRP, the default HDRI sky intensity is high (11) and sky reflection is not being occluded by default. Together with default eye adaption it may result in some example scenes being hardly visible, as they are baked using much lower intensities. Reduce HDRI sky intensity to 1 in this case. HDRI settings are usually located in Assets/HDRPDefaultResources/DefaultSettingsVolumeProfile.

- In HDRP, if lights in example scenes are too bright, click "match real-time to lightmapped" on Bakery light components.

- Use OpenImageDenoise 2 (best) or OptiX 7 denoiser in HDRP, as it is better suited for high intensity values.

- In HDRP "distance" and regular Shadowmask can be switched on the Light component itself, not in global settings.

- Don't use "match" buttons on the Skylight, as HDRP uses a very specific procedural sky shader. Instead, bake it into a HDRI this way.

- It is recommended to always set Light Probe Mode to L2 or L1 when using HDRP, as HDRP's renderer is not optimized for the Legacy mode.

- Newest version of HDRP/URP have their own IES file importer. Because of the asset type mismatch, Unity will unplug them from Bakery IES lights saved using older versions, meaning they will appear as regular point lights instead. They can be plugged back and re-saved.

Bakery's example scenes are designed for the Built-in rendering pipeline, so there are a few steps to properly render them in HDRP/URP. Here are some tips; a fresh project with default HDRP/URP settings is assumed:

Viewing example scenes on HDRP

- After importing Bakery, upgrade standard materials to HDRP: Edit->Rendering->Materials->Convert All Built-in Materials To HDRP.
- Import Bakery_ShaderGraphHDRP.unitypackage into the project, as it contains special shader graphs implementing advanced Bakery features (not required for classic color/directional/mask lightmaps).
- Select DefaultSettingsVolumeProfile.asset. In HDRI Sky section, change Exposure Compensation to 1. Otherwise the default HDRP will be too bright compared to example scenes, and they will appear black due to this exposure adjustment.
- When selecting lights in example scenes, HDRP will initialize its own light data. Sometimes it may result in the light immediately becoming much brighter. Click "Match real-time to lightmapped" to synchronize.
- Set Light Probe Mode to L2 before rendering light probes.
- **example_directional**: change shaders to BakeryDirSpecGraph to see baked highlights (or just regular HDRP/Lit for diffuse-only normal mapping). Delete the sphere, as there is currently no shader graph supporting per-vertex direction.
- **example_directional_MonoSH**: change shaders to BakerySpecMonoSHGraph (or BakeryMonoSHGraph for a slightly cheaper version without highlights). Delete the sphere, as there is currently no shader graph supporting per-vertex MonoSH.
- **example_RNM**: change shaders to BakeryRNMGraph.
- **example_SH**: change shaders to BakerySpecSHGraph (or BakerySHGraph for a slightly cheaper version without highlights). Delete the sphere, as there is currently no shader graph supporting per-vertex SH.
- **example_mixed_directional_mode**: similarly a different graph is needed for MonoSH/RNM/SH planes.
- **example_lights**: change the shader of the vertex-lightmapped sphere to BakeryVertexColorGraph.
- **example_prefabs_bake**: no further adjustments needed.
- **example_sectors**: no further adjustments needed.
- **example_shadowmask**: Select the DirectLight and click "Match real-time to lightmapped" to fix brightness. Set Shadowmask Mode to Shadowmask on all HDRP lights.
- **example_sponza_day**: set OpenImageDenoise 2 (or Optix 7, if unsupported) as preferred denoiser.
- **example_sponza_evening**: set OpenImageDenoise 2 (or Optix 7, if unsupported) as preferred denoiser.
- **example_subtractive**: HDRP doesn't support the subtractive mode.
- **example_volumes_simple**: change train shader to BakeryVolumeSpecGraph (or BakeryVolumeGraph for a slightly cheaper version without highlights).
- **example_volumes_simple_shadowmask**: shadowmask volumes are not currently supported for HDRP.
- **example_volumes_advanced**: same as with example_volumes_simple.
- **example_translucency**: most of the scene should appear fine. For the "Dynamic sphere lit by volume", set shader to BakeryVolumeGraph.

Viewing example scenes on URP

- After importing Bakery, upgrade standard materials to URP: open Assets/Bakery/examples/content, select all materials by typing "t:material" and choosing to only search the "content" folder below. Click Edit->Rendering->Materials->Convert Selected Built-in Materials To URP.
- Import Bakery_ShaderGraphURP.unitypackage into the project, as it contains special shader graphs implementing advanced Bakery features (not required for classic color/directional/mask lightmaps).
- **example_directional**: change shaders to BakeryURPLit, enable "Bakery Lightmapped Specular" (or just keep regular URP/Lit for diffuse-only normal mapping). Delete the sphere, as there is currently no shader graph supporting per-vertex direction.
- **example_directional_MonoSH**: change shaders to BakeryURPLit, set Bakery Mode to MonoSH, enable "Bakery Lightmapped Specular". On the vertex-lit sphere also enable "Per-vertex".
- **example_RNM**: change shaders to BakeryURPLit, set Bakery Mode to RNM, enable "Bakery Lightmapped Specular".
- **example_SH**: change shaders to BakeryURPLit, set Bakery Mode to SH, enable "Bakery Lightmapped Specular". Delete the sphere, as there is currently no shader graph supporting per-vertex SH.
- **example_mixed_directional_mode**: similarly use BakeryURPLit with a different Bakery Mode (MonoSH, RNM, SH). Non-directional / dominant direction / baked normap maps can use regular URP/Lit.
- **example_lights**: no further adjustments needed.
- **example_prefabs_bake**: no further adjustments needed.
- **example_sectors**: no further adjustments needed.
- **example_shadowmask**: similarly to the Built-in pipeline, uncheck Distance Shadowmask in Bakery window (or project Quality Settings) to see actual baked/real-time shadow mixing near the camera.
- **example_sponza_day**: enable alpha clipping on the foliage material.
- **example_sponza_evening**: same as example_sponza_day.
- **example_subtractive**: no further adjustments needed.
- **example_volumes_simple**: change train shader to BakeryURPLit and set Bakery Mode to Volume. Optionally enable "Bakery Lightmapped Specular".
- **example_volumes_simple_shadowmask**: same as example_volumes_simple, but also enable "Volume Shadowmask".
- **example_volumes_advanced**: same as with example_volumes_simple.
- **example_translucency**: most of the scene should appear fine. For the "Dynamic sphere lit by volume", set shader to BakeryURPLit and Bakery Mode to Volume.

Upgrading Bakery

When it comes to upgrading the asset, it is recommended to follow these steps:

- Close Bakery windows.
- Close Unity.
- Open it again.
- Import the updated package.

Note that without first closing Unity importing may fail because of locked DLL files (printing messages like "alphabuffergen.dll: Access is denied.").

If you moved Bakery folders (Assets/Bakery and Assets/Editor/x64/Bakery) to other locations, you might need to move them back before updating to prevent getting two conflicting copies of the package.

Scripting API

Basic usage

Get and modify render settings:

```
ftLightmapsStorage storage = ftRenderLightmap.FindRenderSettingsStorage();
```

Full list of available settings can be seen in ftLightmapsStorage.cs.

Get lightmapper instance and (re)load settings:

```
ftRenderLightmap bakery = ftRenderLightmap.instance != null ? ftRenderLightmap.instance : new ftRenderLightmap();
bakery.LoadRenderSettings();
```

Render lightmaps (also probes if they're set to L1 or L2 mode):

```
bakery.RenderButton(bool showMsgWindows)
```

Render light probes only:

```
bakery.RenderLightProbesButton(bool showMsgWindows)
```

Render reflection probes only:

```
bakery.RenderReflectionProbesButton(bool showMsgWindows)
```

showMsgWindows tells if any dialog boxes/confirmations can be shown. Supressed, if set to false.

Check if baking is in progress:

```
ftRenderLightmap.bakeInProgress
```

For usage example check Batch Scene Baker script.

Additional functions

Copies all render settings from one ftLightmapsStorage to another:

```
ftLightmapsStorage.CopySettings(ftLightmapsStorage src, ftLightmapsStorage dest)
```

Gets global (per-project) storage object:

```
ftGlobalStorage gstorage = ftLightmaps.GetGlobalStorage()
```

Copies all render settings from ftLightmapsStorage (per-scene) to ftGlobalStorage (per-project):

```
ftLightmapsStorage.CopySettings(ftLightmapsStorage src, ftGlobalStorage dest)
```

Copies all render settings from ftGlobalStorage (per-project) to ftLightmapsStorage (per-scene):

```
ftLightmapsStorage.CopySettings(ftGlobalStorage src, ftLightmapsStorage dest)
```

Gets the path to the folder containing Bakery binaries:

```
ftLightmaps.GetEditorPath()
```

Additional variables

Gets the current temporary files path (ftRenderLightmap instance must be created first):

```
ftRenderLightmap.scenePath
```

Gets/sets UVGBuffer data compression (LZ4, if enabled, DDS if not):

```
ftRenderLightmap.compressedGBuffer
```

Gets/sets output lightmap data compression (LZ4, if enabled, DDS if not):

```
ftRenderLightmap.compressedOutput
```

Events

Scripts can subscribe to static events in ftRenderLightmap class.

Called before full Render:

```
EventHandler OnPreFullRender
```

Called before each light probe is rendered in Legacy light probe mode. ProbeEventArgs is a struct containing **pos** field with current light probe position:

```
EventHandler<ProbeEventArgs> OnPreRenderProbe
```

Called after Legacy light probes were rendered:

```
EventHandler OnFinishedProbes
```

Called after full Render was finished:

```
EventHandler OnFinishedFullRender
```

Called before reflection probes are rendered:

```
EventHandler OnPreReflectionProbeRender
```

Called after reflection probes were rendered:

```
EventHandler OnFinishedReflectionProbes
```

Retrieved from "https://geom.io/bakery/wiki/index.php?title=Manual&oldid=1760"

This page was last edited on 26 September 2025, at 19:43.