Light Baking and You

Lighting in Unity is a complicated topic. It's also very slow. It can feel like a ton of trial and error. To remove some of that trial and error, this book summarizes a lot of important information that's good to know about light baking.

This is based on my guide at
https://gitlab.com/s-ilent/SCSS/-/wikis/Other/Light-Baking

- Why bake lights?
- What's Bakery?
- How expensive is unbaked lighting?
- Realtime lighting and "Importance"
- What's a lightmap?
- How to make lightmap UVs
- What are light probes?
- Tools for automatically placing light probes
- A Beginner's Guide To Using The In-Built Unity Lightmappers
- A Beginner's Guide To Using Bakery
- Advanced Bakery Notes

Why bake lights?

If you put a light in a scene, it works, so why do we need to bother with baking? There's two reasons.

First, each realtime light needs to draw everything it touches again. It means lights are really expensive to render, compared to a lightmap which is just stored as a texture.

Second, realtime lights don't have bounced light. Bounce lighting is all around us in the real world, with your average sunny day being lit just as much from the light bouncing off the Earth as from getting lit by the sun itself. Bounce lighting is the reason a window can make a whole room lit.

These two reasons make a really good argument for baking lighting. The only downside is that it'll make your maps larger, due to needing to store that baked data somewhere. Depending on what kind of map you're making, though, it's generally worth the cost. However, for larger maps, expect the lightmap to take 50-60mb of the filesize when optimised down.

What's Bakery?

Bakery is a tool for baking lights with Unity. However, it only works on NVIDIA GPUs. For VRchat, it's really useful because it's

- high quality, using raytracing and AI denoising
- supports physically accurate lighting, but also options for stylisation like fake ambient occlusion
- really fast, much much faster than Unity's light baking
- can bake different types of lightmaps that look better than Unity's default
- can pack lightmaps tighter than Unity using xatlas
- real-time preview addon (purchased seperately) lets you test your lighting before baking it, inspect issues normally invisible even using the builtin tools

What's the catch? It costs $55. The Real-Time Preview addon is an extra $35. So, I can only really recommend it if your time is valuable. (Note that Bakery periodically goes on sale.) Also, you do need some knowledge of the regular Unity lightmap baking workflow even when using it.

Some parts of this guide will still apply to the built-in Unity lightmap rendering, but no guarantees. There are lots of guides to baking lightmaps with Enlighten on the internet, and lots of guides to general lighting theory - I'd recommend looking for some even if you primarily plan on using Bakery, because it's still Unity. I'll add some resources at the end of this guide.

How expensive is unbaked lighting?

This section covers exactly what the cost of each type of light is.

As we're measuring cost, our currency is draw call passes. A draw call is Unity sending data to the graphics card to render. We won't consider the cost of actually shading pixels in this section - if it doesn't add passes, we'll call it **free**. Otherwise, we'll note the number of draw call passes it incurs.

Let's consider a totally blank scene with no lights.

The first light we add isn't a real light, but the **Skybox**. This provides ambient light, and is **free**. The lighting from the skybox goes into the light probes, which we'll cover more later.

The second light we add is a **Directional Light**. This provides the main lighting, and is **free**. Every lit shader will render one directional light, no matter what.

However, that's assuming it casts no shadows. If we **turn the shadows on**, we reach our first additional cost. Unity must render the shadow cascades - a set of textures that contain the shadows for everything in view. VRchat will render **4 splits**, each of which can contain everything visible from the light's perspective, which in the worst case will render everything 4 times. (Also, shadows do not use occlusion culling.)

There is an additional cost. When shadows are enabled, Unity will need to render a **depth buffer** which renders a depth pass for everything visible. This is rendered beside the main shadow pass. The depth buffer is really useful for shader effects, but it is an extra cost.

The third light we add is a **Point Light**. This adds **1 pass** to everything within its radius, and renders everything it touches again.

Our Point Light has no shadows. If we **enable shadows**, Unity will render a cubemap - a six sided texture - from the light's perspective with the shadows. This means things within the light's radius can be rendered **up to 6 times**.

The fourth light we add is a **Spot Light**. It works the same as a Point Light, adding **1 pass** to everything within its radius.

If we **enable shadows** on the Spot Light, it will render **1 pass** for the shadows. It only needs one because it's a Spot Light.

The fifth light we add is an **Area Light**. **Nothing happens**, because Area Lights can only be baked.

If we set these lights to **Baked**, and bake the lighting, they will **no longer have a cost**. They'll be merged into the light probes, which previously contained only the ambient light.

Every non-static object can read the light probe data Unity provides based on its position. It is the cheapest form of lighting, even though it can represent an arbitrary number of lights and their bounces - which can't be represented normally.

Baked lights are great because they're cheap and look good.

Realtime lighting and "Importance"

Realtime lights also have an Importance setting. Unity will only render (in VRC) 8 lights as full extra pass lights, which places an upper limit on the number of times an individual object can be affected by lights and re-rendered.

Which lights it chooses is based on the Importance setting on the light.

When a light is Important, it's rendered as a realtime light with priority. Important lights can render shadows - either realtime shadows or mixed ones which combine a baked mask with realtime shadows.

When a light is set to Not Important, or the limit is reached, Unity will render the light as a vertex light. Vertex lights are much cheaper than regular lights. However, only 4 vertex lights can be rendered on a single object, and they're typically rendered per-vertex - so they won't look great. More annoying than that, though, is they typically won't show on objects with baked lightmaps, so you can't mix the two.

What's a lightmap?

Just as a model might have a basic albedo/diffuse map, a normal map, and etc., a light map is a texture applied to the surface of that model. Using Bakery or whatever, we pre-calculate the lighting using higher quality lighting calculations than we can afford in the actual scene, and then save it into a texture.

This means that your lightmapped models need a seperate set of UVs designed to hold the lightmap. These can be automatically generated in Unity by enabling "Generate Lightmap UVs" on the model file, or created in your modelling tool as a second UV channel. Blender can take an existing UV set and re-unwrap it as a lightmap UV to make this process easier.

Unity provides 3 options for lightmapping, which are exposed to Bakery. They are:

- **Baked Indirect** will bake the indirect lighting into a lightmap, allowing you to choose whether the main light is included or added on top seperately. This means you can have scenes where the lightmap has baked lights while still containing a realtime sun or moonlight with dynamic shadows. This is pretty expensive, as the main light will need to still render realtime shadows for each object, but also the best looking.

- **Shadowmask** will bake a lightmap containing the ambient light, and then a second map with the directional light shadow baked into it. This allows the light to use some of the benefits of a realtime light, like proper specular highlights, while avoiding the cost of realtime shadows for the baked part of the scene.
  Shadowmasks can then be combined with dynamic shadows, which looks nice and realistic. Or you can set the light to be fully baked with Occlusion Probes, in which case Unity will also bake a value into each light probe specifying whether it's being hit by the main directional light or not, making it look mostly the same but without player shadows. In Bakery, Occlusion Probes can be enabled from the bottom of the Bakery window.

- **Subtractive** mode will bake a lightmap containing all the lights in the scene, but dynamic objects like players will still cast shadows. Those shadows will cut holes in the lightmap corresponding to the shadow colour. It's very optimised, but it looks kind of ugly.

How to make lightmap UVs

Creating UV maps for objects is tough. And doing it again for lightmaps is even more work. But it makes a big difference to visual quality. Unity's automatic lightmap UV generation doesn't work too well with complex objects, and for simple ones, it can still place seams in totally obvious and ugly places. The solution? Make them ourselves!

A lightmap UV must sit entirely within the 0-1 range, and no parts of it can overlap. If parts of the lightmap UV overlap, the baking result will be broken because the rays from the overlapping regions will fight with each other. If you see black splotches on your lightmap, it can be caused by broken lightmap UV. (Bakery will warn you about overlapping UVs in the log.)

Try and give each surface a consistent texel density. In other words, make sure each part of the object gets about the same amount of lightmapping space, such that if the model's texture was a checkerboard the squares would look a consistent size. Bakery will automatically smoothen out seams between your lightmaps where it can, but if the sides don't line up, it will end up looking a bit broke anyway.

With that in mind, let's go over some key points to keep in mind.

1. Don't overlap UVs.
As noted before. The lightmap can't be generated correctly if the UVs overlap.

2. Keep the texture coordinates within 0-1 space.
As noted before. The lightmap can't be generated correctly if the UVs are outside this range. The actual resolution on lightmapped objects is set inside Unity.

3. Don't pack individual UV islands together too tightly.
Shadows from other parts of the model will leak if they're too close. To avoid light bleeding, there must be a sufficient amount of space between each island.

4. Hide seams as much as you can.
Make sure seams are inconspicuous, because depending on the lightmap resolution they can be pretty obvious.

5. Don't place UVs diagonally when you could align them to an axis.
Diagonal UVs will stand out badly, and they're wasteful compared to axis-aligned rectangles. Avoid them as much as possible.

6. Connecting UV islands isn't always a good idea.
On the one hand, connecting islands together makes sense. It reduces seams and ugly gaps. But on the other hand, if the resolution for an area isn't high enough, the texels for one area of the mesh will bleed across to the other parts, leading to incorrect shading.

The disadvantage tends to be especially noticeable in buildings, since they are generally flat planes.

7. Put more texel resolution into corners where the angle of the surface changes.
If your model has bevelled edges, giving the bevelled regions extra space on the lightmap avoids both bleeding and seams.

Actually, for VR especially, it's a good idea to give the edges of hard-surface models a bevelled edge. This removes the obviously fake CG-look that super sharp edges have... but it also makes the lightmapping look better.

If you have hard edges on a model, splitting the lightmap there looks right, but most props and objects in the real world won't have a super hard edge. If you connect the edge there, though, the lightmap bleeds across it in an ugly way. So smooth that edge out with a bevel! The bevel will let the light fall across the model smoothly, adding in more perceived detail and a natural appearance.

Notes:
* http://tsumikiseisaku.com/blog/unity-japan-office-2/
This info was mainly based on this blog post, which is written in Japanese. Check out the original for some nice demonstrative images and insight into the modelling process for the Unity Japan Office project.

* https://docs.unity3d.com/Manual/LightingGiUvs-GeneratingLightmappingUVs.html
The Unity documentation on generating lightmap UVs, which goes into great detail about how the auto generation process can fail.

* https://docs.unity3d.com/Manual/ProgressiveLightmapper-UVOverlap.html
Unity's page on how to debug overlapping UVs, and the kinds of issues they cause.

* https://docs.unity3d.com/Manual/GIVis.html
Unity's page on the various visualisation modes that can display issues with your lightmap.

What are light probes?

Lightmaps cover the environment, but if you just bake a lightmap, dynamic objects won't be affected. You also need light probes!

Light probes are points in space that capture an image of the surrounding lighting from all directions. When there are enough probes to create a tetrahedron between them, that space will affect the dynamic objects inside it and send them the data from the nearest probes, blending between the points of the tetrahedron.

Light probes have a major impact on how dynamic objects receive the lighting from your scene. Because of this, getting their placement wrong will make dynamic objects look totally wrong. However, light probes are *extremely* cheap. You can have tens of thousands of baked light probes in a single scene with practically no performance impact. Conversely, placing tens of thousands of probes would be awfully tedious, which is why it's recommended to use tools to do the job for you.

In this case, I recommend reading the Unity page on light probes to get an understanding of how they work and should be placed.

Tools for automatically placing light probes

This section covers information on light probe placement tools.

LightProbesVolumes (free)

Light Probes Volumes is a free tool that places probes down based on the scene collision. It's easy to use and quick to set up. Simply place down volumes and have it fill them.

Magic Light Probes

Magic Light Probes is a paid asset on the Asset Store. It costs about as much as Bakery, and goes on sale about as often. It uses some algorithms to place probes along where light is varied and where lights are. It can be used to place a lot of probes down at once, following the map geometry really closely, in a way that can represent lighting surprisingly faithfully. The setup procedure is complicated, but it can produce really good results. However, it's prone to bugs.

AutoProbe

AutoProbe is a paid asset on the Asset Store. It costs half as much as Bakery, but goes on sale fairly often too. It uses simple methods to scatter light probes around the scene evenly within volumes. It also has an optimisation function to remove unnecessary probes after the scene is baked. It produces good results.

A Beginner's Guide To Using The In-Built Unity Lightmappers

(Unfortunately, this is really more a *cynicist's* guide.)

This section will go over using the in-built Unity bakers. Unity provides three options for lightmap baking.

- Enlighten is tried, true, terribly outdated, and depreciated in newer Unity versions.
- The Progressive bakers are buggy with their own set of idiosyncrasies, especially in Unity 2018.

If you use Enlighten, you can also use Realtime GI. Realtime GI bakes a meta-lightmap that stores what each part of the lightmap can see, and then calculates the lighting while the scene is running in realtime, refreshing them whenever the lighting changes. This is great for when you want a cinema, or changing light source, but it requires keeping the size of the map small or else the time it takes to update the lightmap will be too slow to look good. Realtime GI also affects performance, even though it's refreshed so slowly. If you don't NEED it, turn it off. Keep that in mind for the next sections.

The Lighting Window

In the Lighting window, you can configure the most important settings for the in-built lightmappers. I'll cover them one-by-one.

For Environment Lighting, I recommend always using Skybox mode and always having a skybox in your scene that represents the ambient lighting you want. Always have the intensity multiplier at 1. Ambient Mode should match whether you want the skybox to change the lighting (use Realtime) or if the skybox is static (use Baked).

For Environment Reflections, it is best left to either Skybox, or a Custom probe referring to a probe in your scene. For more details, please see my Reflection Probe guide. Never leave this blank.

Realtime Lighting = do you want it? If not, make sure to disable this.

Mixed Lighting - as mentioned before, these are the mixed lighting modes. Choose carefully.

Lightmapping Settings - this is where you pick your poison. If you disable Realtime or Baked GI, different options will be disabled. These options mainly affect the end result. This is the options for Enlighten, because the Progressive ones are better explained in the Unity documentation.
* Indirect Resolution sets the lightmap size for Realtime GI. It has no effect on the baked static lightmap.
* Lightmap Resolution sets the texel size for the baked lightmap. You can view how many texels are applied to each object by selecting Baked Lightmap from the dropdown in the top-right corner of the Scene view.
* Lightmap Padding sets how big the gaps between each object on the lightmap are, but you rarely need to mess with it.
* Lightmap Size sets the maximum size a lightmap can be generated at. If an object has a big lightmap scale or scale in the scene, it will be clamped to fit in one of these textures, and if it's smaller than that, it'll be grouped with other small textures until they fill a lightmap. Bigger is better, but Unity isn't the best at filling these up tightly, so a bigger lightmap with small objects can often have big regions of wasted space.
* Compress Lightmaps should always be on. Without compression, lightmaps become incredibly large.
* Ambient Occlusion will add extra darkening to the corners of the lightmap, which can improve how it looks aesthetically even if it isn't exactly realistic.
* Final Gather will improve the quality of the final result and it looks nicer. However, it takes a bit extra to bake.
* Directional Mode sets whether directionality maps are generated. Directionality maps are required for normal maps to work with baked lighting.
* Indirect Intensity scales up and down how powerful indirect lighting is. The documentation suggests it should be a direct multiplier to the lightmap.
* Albedo Boost makes objects have a brighter base colour. The documentation suggests it makes things white, but that doesn't seem to actually be the case.
* Lightmap Parameters is a seperate settings file with even more options that are harder to explain. Please check the Unity documentation for more info.

A Beginner's Guide To Using Bakery

If you open the Bakery window, it'll change your lighting settings so that Unity won't try to automatically overwrite the lightmaps it bakes. Because Bakery is seperate from Unity's lighting system, you'll need to take note of the following:

- Don't use the "Generate Lighting" button in the Lighting window, or Unity will overwrite the Bakery lighting.
- Don't turn on "Auto Generate" in the Lighting window for the same reason.
- Don't use the "Bake" button on Reflection Probes, unless they're Custom. Clicking "Bake" on a reflection probe in "Baked" mode will start the whole Unity lightmap baking process.

Bakery doesn't use Unity's Light components. If you have a regular Light in your scene, it will always be rendered as a Realtime light. Instead, Bakery comes with its own set of Light components, which have different settings from Unity's own.

You can place a Unity light and a Bakery light on the same object. If you place a Bakery light onto a Unity light, it will provide options to copy the Unity light's parameters. This is handy for when you want to convert a lot of lights to Bakery lights - just select them all, add the corresponding Bakery Light component, and select "copy from realtime light".

When baking the light from Bakery, it will also set the Baked flag on the Unity light. This can be used for Mixed lighting, which has the same performance impact as a Realtime light, but looks nicer due to having bounced indirect lighting from the environment. Or you can use it to preview directional and point lights before you bake them, visualizing them as Realtime lights.

With Unity, the skybox is automatically used to render ambient light in the skybox, but this isn't the case with Bakery. To render ambient light, you must add a Bakery Sky Light component. The Sky Light will cast lighting from all around the outside of your scene onto it using either an ambient colour or a cubemap. Without ones, shadows will be black.

Ideally, you should render a cubemap of your skybox using a Custom reflection probe, with everything but the sky hidden. Then use the resulting texture in the Sky Light. The result will look accurate and smooth.

Bakery DOES use Unity's Light Probe Group components to bake light probes. It provides two Light Probe Modes:

- L1, where Bakery will create the light probes with the lightmapping process. This is the default and fastest option. It has the minor disadvantage of seeing the scene as Bakery sees it, rather than how Unity sees it, so it won't capture special shader effects properly.
- Legacy, where Bakery will create the light probes after the lightmapping process. It generates the probes from rendering the scene in Unity into reflection probes, which ensures the results are accurate, but it's a bit slow.

If you use L1 mode, light probes are baked at the same time as the lightmap. If using Legacy mode, you'll need to bake them separately. Bakery will bake data into light probes when the "Generate Light Probes" button is clicked.

So, to review, when making lighting for a scene, you should:

- Disable Unity light components, unless your lights are Mixed.
- Add a Bakery Skyprobe, and give it a skybox cubemap to use for ambient light.
- Add a Bakery Direct Light, and set it up as you would the main directional light.
- Add Bakery Point Lights if you need them.
- Make sure your scene has lightmap UVs.
- Use the Show Checker option in Bakery to see the resolution of your lightmap on individual meshes, and adjust them if necessary.
- If you have an RTX capable graphics card, turn on RTX mode.
- If your scene uses normal maps, set the Directional Mode to Dominant Direction, which is the most compatible mode.

Advanced Bakery Notes

Mixed Lighting

Mixed lighting incurs the same costs as realtime lighting for dynamic objects. Because the first directional light is free, you can combine high quality baked lighting with real-time shadows using Shadowmask mode. Combine a Unity directional light in Mixed mode with a Bakery directional light set to "Shadowmask and Indirect" and you'll create a shadowmasked directional light. With this, you can get the look of dynamic shadows while only needing to render a depth buffer and dynamic object shadows.

To take full advantage of this, enable Occlusion Probes in Bakery. Occlusion Probes store the shadowing of the directional light into the light probes. Unity will darken the main directional light when objects are in shadow according to the probe values, as though they were receiving shade from the lighting.

Light Probe Ringing

Bakery will bake the full range of incoming lighting into light probes. However, due to the nature of the light probe calculation, it's possible for areas with high contrast lighting to get extremely strong shadows where there are high incoming levels of light from a single direction. This phenomenon is known as ringing. This mainly affects objects using Standard shading.

There are a few ways to avoid ringing. Objects in the scene can be set to use Bakery's Standard shader, which has an option to use "non-linear SH" which performs a filtering step to avoid the issue. However, objects where the shaders on them can't be controlled, like avatars, will need to incorporate the change seperately.

Mixed lighting with shadowmasks can avoid the issue by moving the high intensity directional light - a main cause of ringing - out of the light probes and into the shader calculations. However, this won't affect smaller sources of light. Alternatively, the problem can be avoided by avoiding high levels of contrast in scene lighting.

The situation may change if Bakery adds its own deringing filter to light probes.

Realtime GI

I recommend not using Bakery in the same project as a map that relies on Unity's Realtime GI, because Bakery will automatically turn it off for you. You can use Bakery and Realtime GI together if you tick the option to "combine with Enlighten realtime GI" in Bakery's settings. As the setting notes, this is using Enlighten, which means that you'll be waiting a while for Unity to finish baking it. Then Bakery's lightmap will be added on top.

https://github.com/MerlinVR/VRC-Bakery-Adapter

A script that handles converting Bakery RNM and SH directional lightmap bindings into a format that VRChat can process without the Bakery scripts being whitelisted.

http://techblog.sega.jp/entry/2019/04/25/100000

A guide to setting up believable physically based lighting in Unity.

https://gitlab.com/s-ilent/lightmap-quality-regions

My tool for setting up Lightmap Quality on models in regions. Compatible with Bakery.