Reflection Probes and You

Unity's reflection probe system is a little confusing and convoluted at first. Things that would seem to be good practice turn out to be horrible mistakes for newcomers. After passing around lots of similar advice for years, I've decided to collate it all into one page.

I'll cover how they work, and more importantly, how they should be used in worlds.

- The basics of reflection probes
- How to set up reflection probes
- Tips and tricks

The basics of reflection probes

A reflection probe captures an image of the world around it and stores it into a "cubemap" - a six-sided texture that can fill every side of a cube.

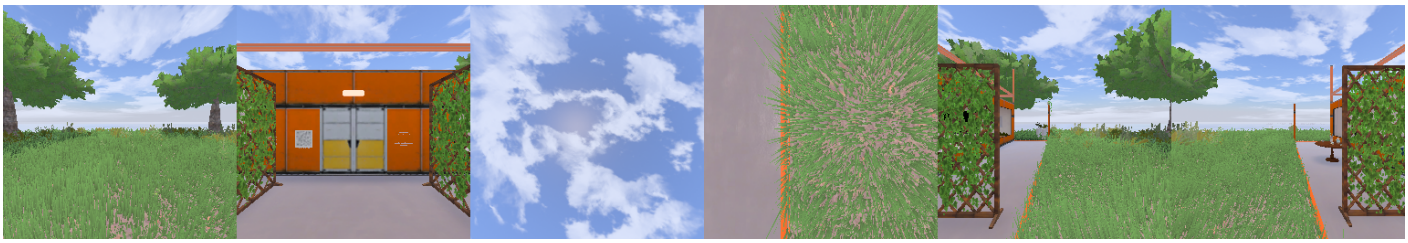
Each side of the texture is square, and of the same resolution. This means a "512" resolution probe is six 512x512 textures, compacted together. This takes up 2MB of space and 2MB of memory.

Reflection probes also store lower-resolution versions of their textures. These lower-resolution copies are blurred, and allow for the representation of rough or uneven surfaces realistically.

As objects are rendered in the scene, they will be asking Unity "what reflection probe am I in?". Unity will provide them the texture and location of the two closest probes to their anchor position, and the shader will blend between them. If only one probe is available, Unity only will send one.

In addition to reflection probe objects, each scene has an "environment reflection", which is provided when no reflection probes are nearby. This is typically generated from the skybox material, but can be overridden manually.

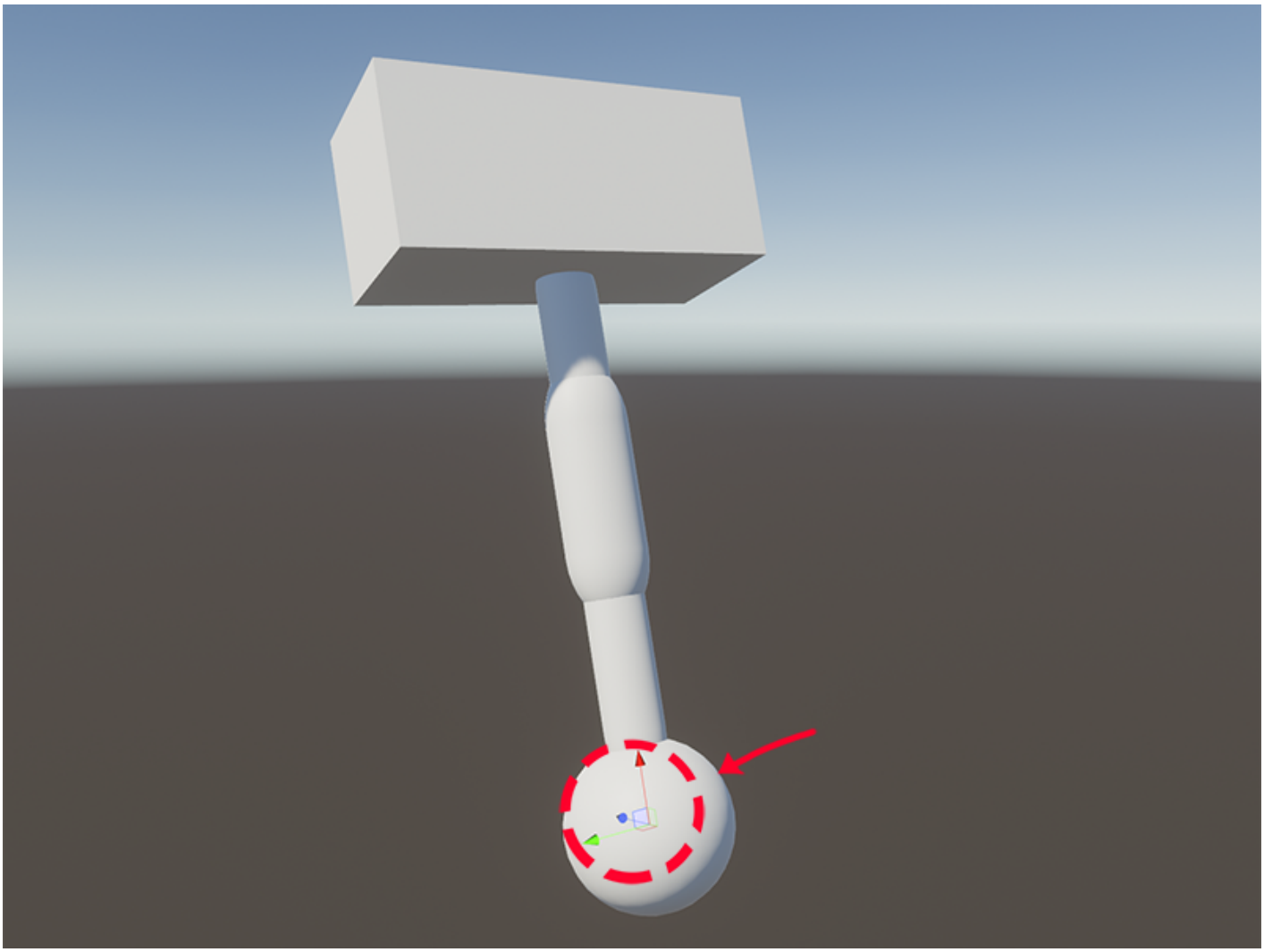
An example of a reflection probe texture



In this image, you can see the contents of a reflection probe. Typically, reflection probes are stored uncompressed with all their faces laid out like this, but Unity will automatically combine them when importing them or any other cubemap texture laid out the same way. You can see that the probe in this example contains a building, a sky above, and an exterior area. Even though the probe is only 512 pixels on each side, it is still pretty sharp and contains a reflection that looks accurate and sharp in realtime.

How do they work for dynamic objects, like players?

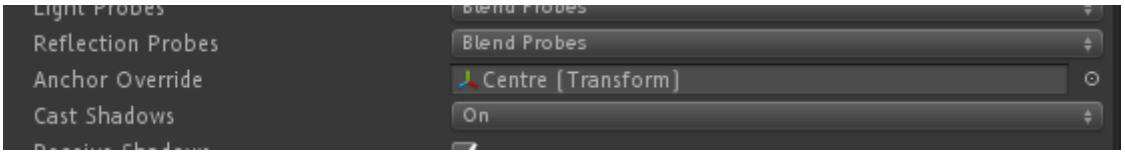
Dynamic objects use the probes closest to their anchor position on their mesh.



In this image, you can see the anchor point of an object. It's at the root of the object, rather than the middle as you might expect.

This object's anchor is, in fact, where the red arrow is pointing - even though the centre is elsewhere. Unity will use that position to pick the reflection probes for the model. This position is, by default, the "origin" of the mesh - the point at (0, 0, 0) that every vertex is placed around.

You can override this by setting a GameObject in the "Anchor Override" field on the Mesh Renderer.



How do they work for static objects, like the map?

They work the same as for dynamic objects. This, however, can cause some problems.

Unity has static batching, which combines multiple models into a single mesh, and renders them in a single draw call.

However, Unity will not combine objects affected by different probes.

You can partially work around this with Anchor Overrides. Objects using the same Anchor Override will sometimes be batched together. Unity will often fail to batch them together, though, so in these cases you'll be required to merge the meshes manually to regain performance.

What do shaders do with reflection probes?

For the most part, shaders will follow what the Unity Standard shader does. Unity offers some functions that let any shader use parts of Standard to provide the same reflections as Standard does, or access the data to do it themselves.

As noted before, Unity will provide them the texture and location of the two closest probes to their anchor position, and the shader will blend between them using some handy functions Unity provides which most shaders will make use of.

In Standard, the occlusion map is used to define areas that can't normally be reached by light, which will make those parts reflect less light and dampens the effect of reflections.

How to set up reflection probes

Reflection probes capture an image of their surroundings into a texture. This means a few things

- A higher resolution probe is much sharper, but much much bigger, due to containing six textures.
- The texture is static. It can't change. It won't be updated if the surroundings change.
- They can only represent a single perspective, which is shifted around to give the illusion of depth.
- Despite all this, nobody's going to notice an inaccurate probe if it roughly fits the lighting of the area.

So, here's my method for determining how to place probes in a scene.

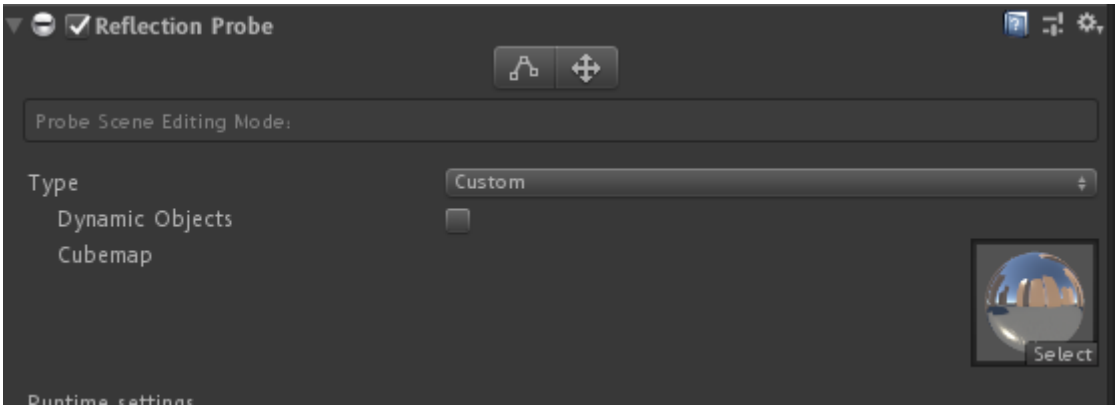
1. Basic reflection probe

Even if you don't have a reflection probe, Unity will capture the skybox material into a reflection probe and hand it out. This reflection probe is given to everything by default.

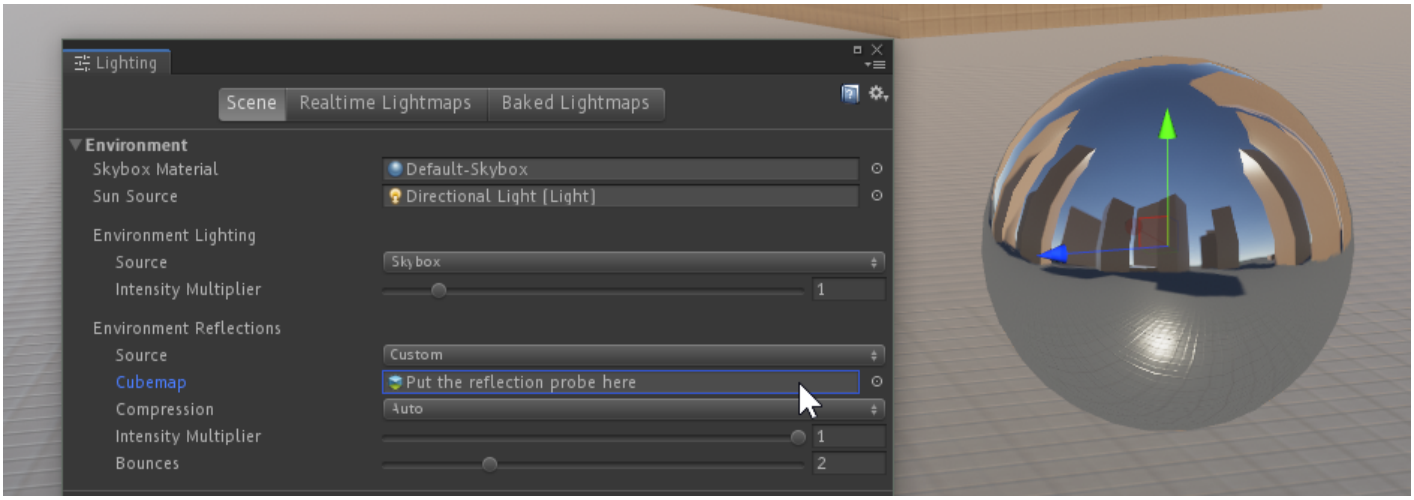
Is the Unity skybox really representative of your scene? It probably isn't, so let's change that to something we set ourselves.

First, add a new Reflection Probe. Set the Type to "Custom".

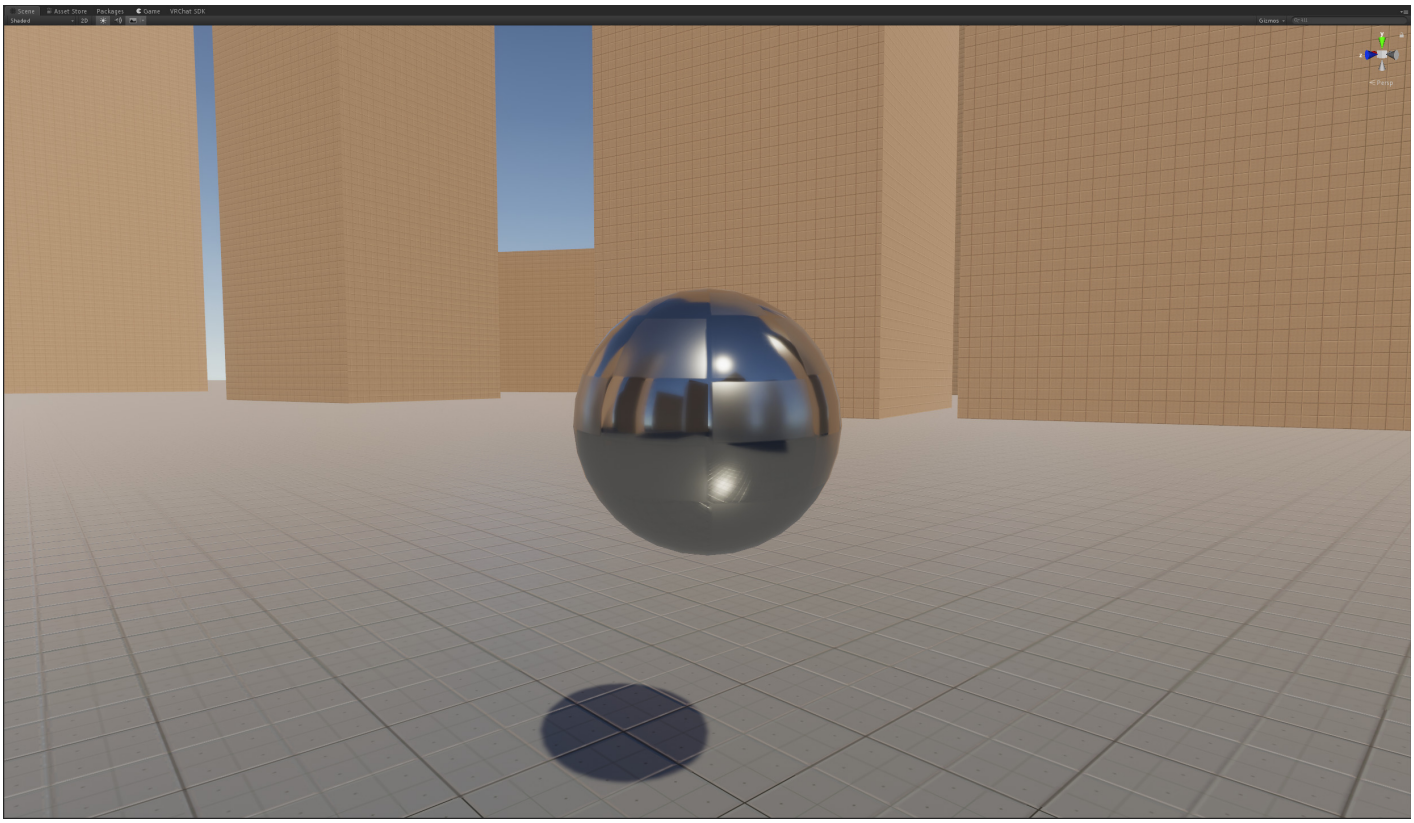
- If you've marked all the objects in your scene as static, don't turn Dynamic Objects on.
- If you haven't marked your scene as static (because you haven't merged it, or can't use static batching) turn Dynamic Objects on. Then select Bake at the bottom and place your Reflection Probe texture somewhere relevant to your scene.



Second, open the Lighting window from the Windows menu at the top. From there, set Environment Reflections to Custom, and drag your new probe in.

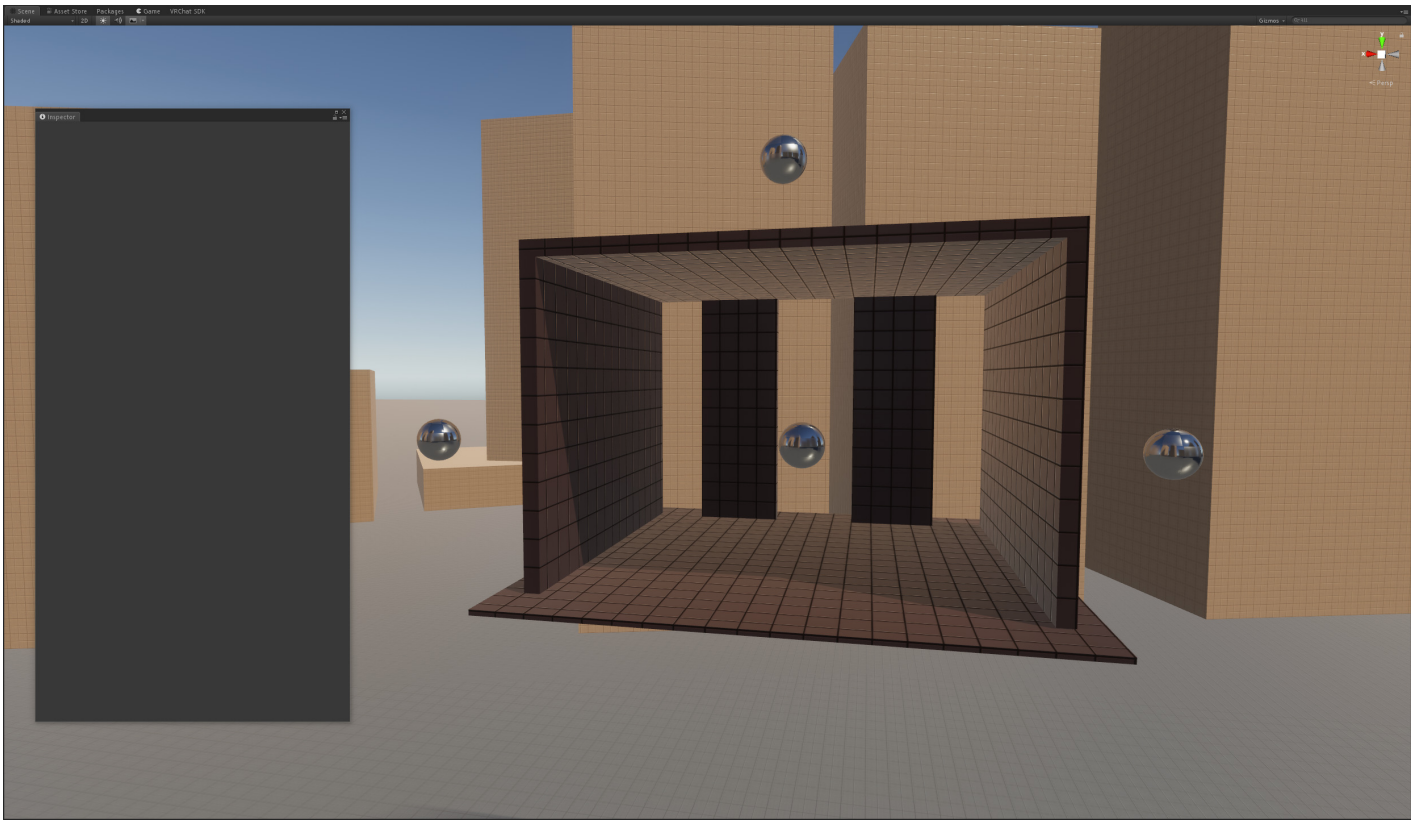


After doing this, you can disable the original probe. Now your scene has a default reflection that won't affect batching.



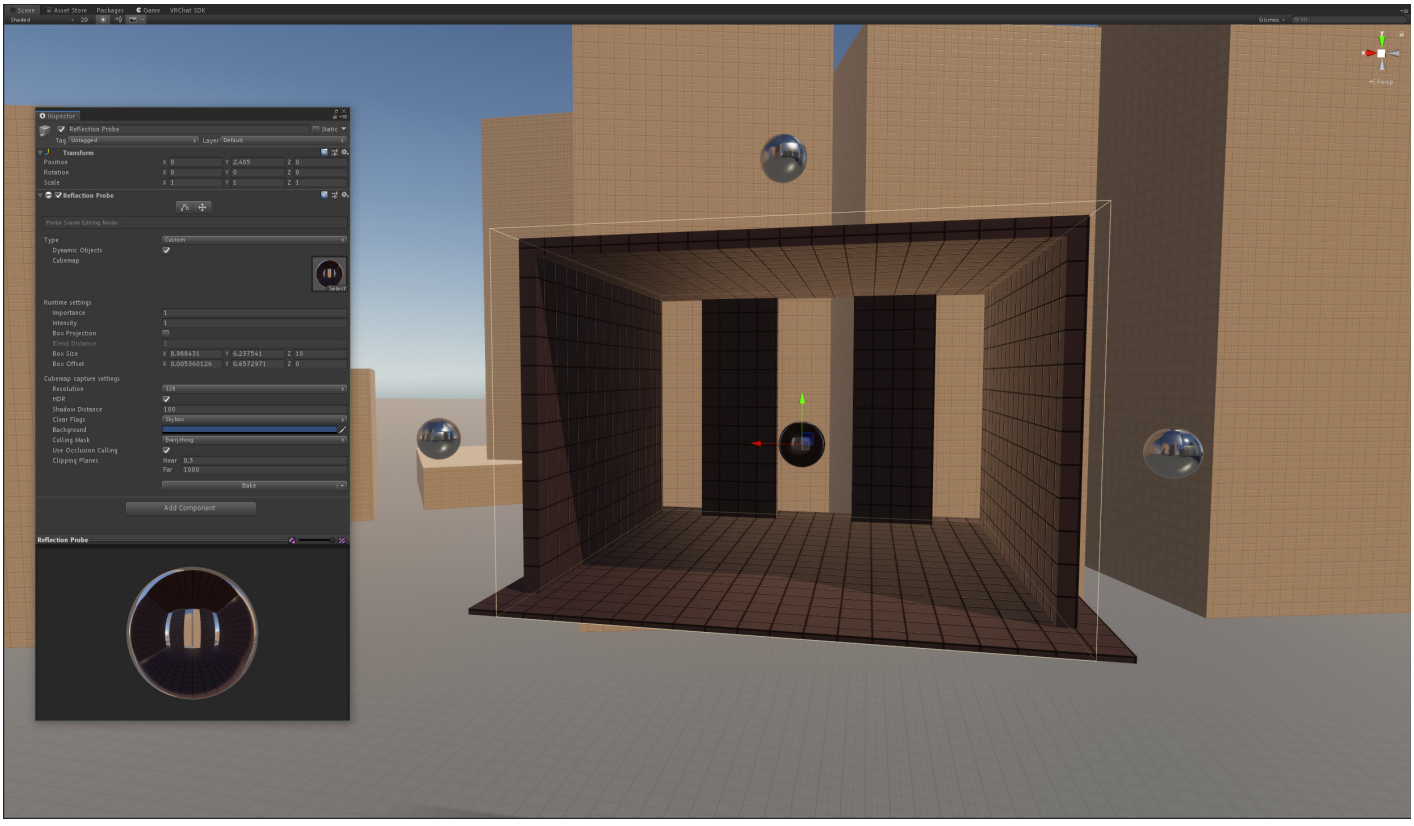
2. Complex reflection probes

One obvious problem with this is that there are places where your reflections will look wrong. Normally, this isn't a problem. If your reflections match the colour and shading of the area around them, they don't need to be totally accurate. But when the difference is too big, it can stand out in a bad way.



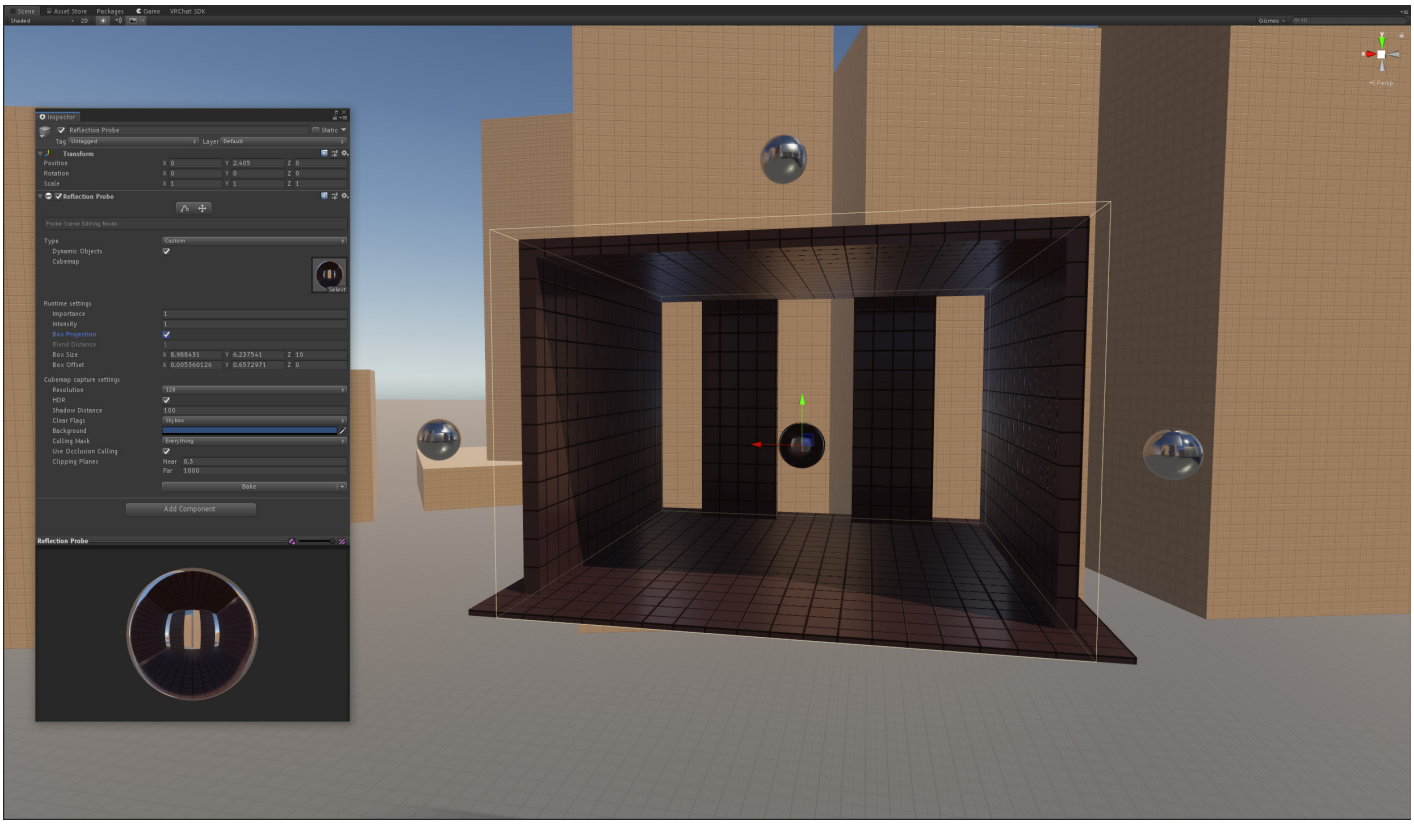
This isn't a worst case scenario, but it's pretty obvious that things being reflected aren't there. It's extra worse on the sphere in the room, because it doesn't reflect the walls around it.

If this is an important area of the map, just put an extra probe there!



In this screenshot, I've set up the probe area using the controls at the top of the Reflection Probe component. Because the probe area is defined, the steel spheres next to the structure are still using the outside reflection, while the sphere inside is reflecting the room. The room is reflecting itself, so the sky isn't visible, though the buildings are through the back wall.

If the area is box-shaped, we can fit our probes' bounds to the walls and enable Box Projection.



Box Projection is a powerful technique that will fit your reflection probe to a box. Used well, it can make reflections look really good, even if there's no perfectly reflective surface around to show it off.

Notes for box projection probes

Reflection probe bounds define what objects are affected by the probe. At the edges of the bounds, the objects will interpolate between the original probe and the most important, closest probe. For regular probes, this happens from the inside out. For box projection probes, though, this is outside-in.

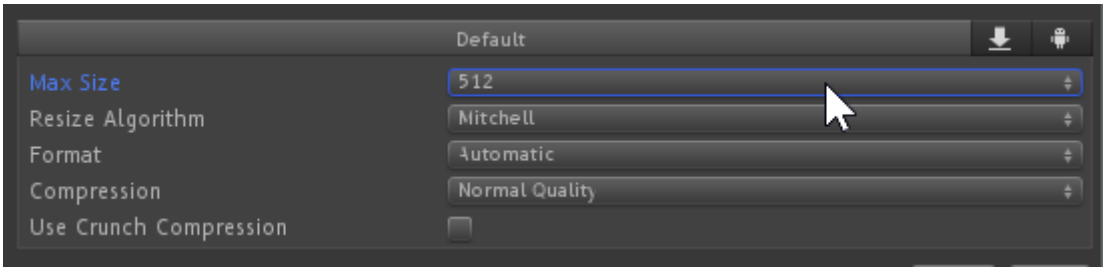
A video showing the expanding bounds of a box projection probe. Thanks ScruffyRules!

The bounds of box projection probes will actually expand to encompass the objects around them, based on the bounds of the objects' mesh renderers. This can cause big problems with large meshes, because a box projected probe that is too small for a large mesh but has a high priority will be stretched to fit the larger mesh.

Tips and tricks

Supersampled Probes

When baking a probe at a low resolution, the result can look a bit low-res. This is because the probe is rendered at that low resolution before it's saved. If you bake the probe at a higher resolution, and then set the Max Size lower on the texture, Unity will render the probe at the higher resolution first, and then squish it down - performing really high-quality anti-aliasing in the process.



Specular Highlights From Point Lights with SpecularProbes

Download SpecularProbes here.

When you bake lighting for a scene from point/spot lights, you'll often find that the result doesn't have the same specular highlights that it did with real-time lighting.

An example of such a scene.

This is because point lights don't have a physical presence in the environment rendered by reflection probes, and if there's no bright object where your point lights are, there'll be no shiny highlight to match it.

You can use this Unity tool to automatically add a glowing ball to each of your point lights, which will create a shiny highlight in your reflection probes.

An example of such a scene.

Note that there are other ways of getting specular highlights from baked light using special shaders. If you're using Bakery's "lightmap specular" mode, you don't need to use this.

Lighting Changing Over Time

Unlike baked lighting, reflection probes are tied to their components. If you disable a reflection probe, it'll stop affecting objects around it. Conversely, if you enable a reflection probe, it'll start affecting things around it.

So, if the lighting in your scene changes, you can enable and disable reflection probes containing different lighting. This allows you to keep the reflections correct between different lighting conditions.

The main catch is storing all those reflection probes. At 512x512, each one is 2MB. This can add up pretty quickly. An alternative is to set the probes to Realtime mode.

In Realtime mode, Unity will render the probe in-game, instead of saving it to a texture. Using a Realtime probe might be the best choice for your scene if major elements of the scene can change - like lighting shifting over time, or objects in the map changing appearance.