Guide

So, you got the SDK, and imported a model of a level you ~~ripped from your favorite game~~ lovingly crafted in 3D modeling software. You set it up to the best of your ability, put it in Bonelab and... yikes. It looks terrible. Things are falling through the ground. It's running at 10 frames per minute. What did you do wrong?

Fear not friend, Lava Gang is here to help you out.

Before we begin: this is not a how-to guide. This guide assumes that you already know the basics of Unity and ideally some rudimentary 3D modeling, and how to import and utilize the SDK. You should know basic game development terms like *rigidbody* and *mesh collider*. And I'm not going to teach you level design. What this guide *is* for is getting the most out of your scene visually, getting it running well, and the things to make it feel good for physics and VR. I will update this guide as more of our internal tools get added to the SDK, but for now it will only cover what is available by default in Unity 2021.

Colliders

The first and most important thing you'll need to do after importing or building your level is to get it collidered. For a lot of modders, this will mean slapping a Mesh Collider on every mesh and calling it a day. Please, for the love of Ford, don't do this.

- Mesh Colliders are paper thin, and when used as the ground will cause physics items and often the player to move right through them.

- Mesh Colliders are heavy. Not only are they more complex than necessary from a geometry standpoint, but the way Unity calculates physics means it's doing extra work anytime a rigidbody is inside the bounds of a concave Mesh Collider.

How do I collider my scene then?

Using Primitive Colliders

The best way to collider a scene is to do it with primitives: boxes, capsules, and spheres. This can be tedious and take a lot of work, but it is well worth the effort. It's a major performance optimization, and will give you much more control over surface materials (which we will go over later), and other features that we or other modders may implement down the line. Doing this will also help you alleviate most of the physics bugs that come with concave Mesh Colliders such falling through the ground and physics explosions.

Do:

- Place each collider on its own Game Object. Colliders can't be rotated, but a Game Object can! This also helps a lot with organization.

- Make a bit of overlap where colliders meet. This helps avoid physics objects finding their way through the seams.

- Regardless of organization, save some time by adding the primitive collider straight on the game object with the mesh: Unity will try to automatically size the collider to the mesh. Then you can copy and paste that collider onto a child game object for organization, or just leave it if you're a masochist.

Try to avoid:

- Thin colliders. Thin colliders (<10 cm) have a high likelyhood of causing physics issues, especially at high velocity or low framerate. Ground planes and walls should aim for being 10-50 cm thick. If the player is hitting it at velocity, like a wall in a gokart track or the ground after a high fall it should be thicker, ideally 1m.

- Scaling. Use the Collider's Size and Center values to get the shape. Scaling the collider using the Transform values has a score of issues it can cause, especially with grips and other Marrow1 interactions.

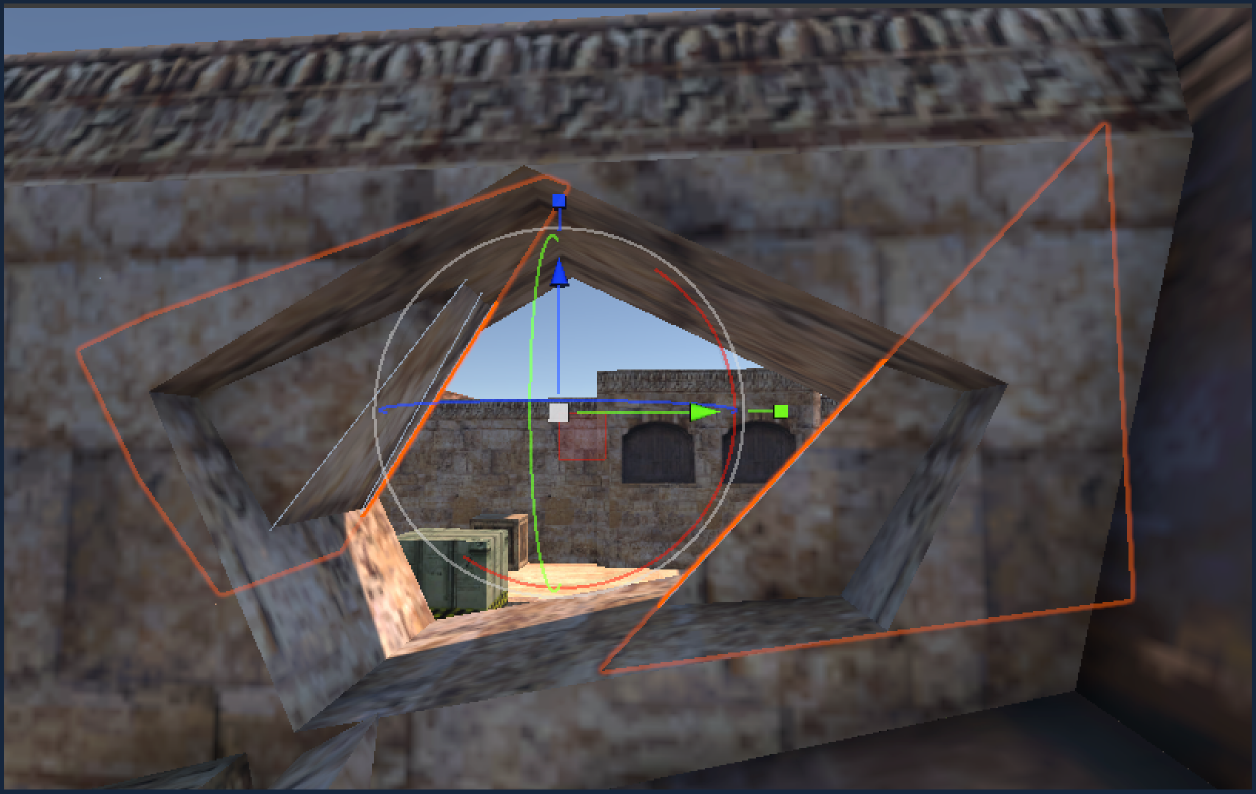
- Being too precise. If you're throwing 100 Box Colliders on an object to make it collidered accurately, a concave Mesh Collider may be more performant in the end (but will still have the physics problems).

- Not being precise enough. Remember, it's way more obvious in VR than in flatscreen gaming when something is floating off the ground, or your hand hits an invisible barrier. Within reason, try to be accurate to your geo. Just consider what things the player is or isn't likely to interact closely with.

When to use Mesh Colliders

But I get it: your scene has complex geometry that can't just be broken down neatly into boxes and capsules. Using mesh colliders sparingly is okay.

One thing you can do is break your scene's models down into smaller pieces, ideally pieces that will allow you to use convex Mesh Colliders rather than concave. Convex Mesh Colliders are much better for performance than concave, and are not two-dimensional so they minimize physics bugs. Convex Mesh Colliders are not quite as good as using primitives, but they're close.



These two separate rocks would be awkward to build out with Box Colliders, but are excellent candidates for a convex Mesh Collider.

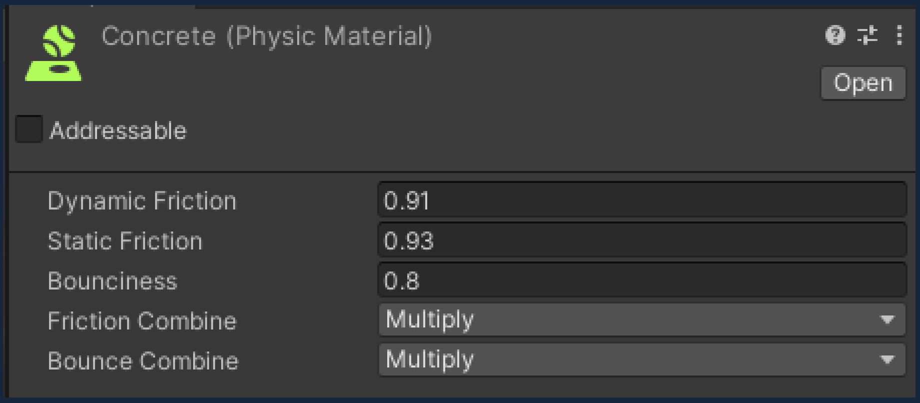
The fidelity of interaction in VR means people are much more likely to notice when your colliders don't match the geometry, but that doesn't mean everything has to be exact. For many objects, using a convex Mesh Collider may save you a lot of time and effort and be close enough. Use your discretion.

The same applies to concave Mesh Colliders; sometimes it's worth the time saved to just slap them on an object and call it a day. Breaking apart your scene's meshes as mentioned earlier will help a lot with this. One object in a scene having its own concave Mesh Collider is a minimal performance hit, as its bounds for calculating physics will be much smaller. You'll want to watch out though if the object is something the player is likely to hit at velocity, such as an obstacle in a gokart track. The faster the physics interaction, the more likely that it will cause unwanted behavior with a concave Mesh. The same goes for if the player will be interacting with the object with a thin collider of their own, such as using a crowbar to slide down a zipline. Thin colliders have a high likelihood of passing through each other.

Physic Materials

This is a lesser known feature, but every collider can be given its own unique physics properties such as bounciness and friction. Most colliders will be fine not being given one: it will default to the properties of concrete. But if you want a slippery, bouncy, or sticky surface you can change them here. We like using Multiply rather than Average, and we use real-world physics values for various materials. These values can be found by Googling.

Physic materials are much more important on physics objects than static. If you don't add them to anything else, at least add them to your props.



Concrete (Physic Material)

Addressable

Dynamic Friction	0.91
Static Friction	0.93
Bounciness	0.8
Friction Combine	Multiply
Bounce Combine	Multiply

Speaking of which, you may want different values for your dynamic and static materials. These are the values we use for static concrete (as it's the actual irl values), but can feel pretty goofy on a dynamic concrete block, which we set to a Bounciness of 0.05.

Lighting

General Tips

- Don't go nuts on the intensity value. Most lights should just have an intensity of 1 to start with. If you need to increase the range, make sure you increase the range as well as the intensity. The range clips off the light but doesn’t make it more intense (unless you make it *too* big, in which case Unity handles it poorly). It does matter for realtime lights because a smaller range means less things affected which means less calculations, but that being said...

- Don't use real-time lights. Like, ever. Seriously these are extremely heavy and one or two will tank framerate even on PC. A single real-time light with shadows can barel run on Quest. If you *must*, try a Mixed light, or a realtime light with no shadows. Still, use them extremely sparingly, or preferably not at all.

- Emissive maps *do* bake light into the lightmap. We generally put an Emissive Multiplier value of 5 for actual lights with an emissive map. This value often makes it so we don't even have to add a light to the scene, the emissive map will do the work. Make sure your emissive materials are set to Baked Global Illumination to see them in the bake.

- Spot Lights are kind of busted in this version of Unity. You may need to really crank up the values to get them to look right in the bake. A Spot Light that looks good with Intensity 2 and Range 50 might need to be Intensity 40 and Range 300 to look the same in the bake. It is what it is.

- Try to use Area Lights over Point Lights whenever it makes sense. You have much more control over them. Point lights look tacky unless the source needs to cast omnidirectionally. Just make sure you increase the range so you don't get unnatural cutoffs. Area Lights can have a pretty high range and still look natural.

- Directional Lights should be at least an angle of 0.5 (that's the size of the sun in the sky).

- Hard Shadows look janky. Use Soft Shadows, and for shadows you want to be crisp just use a very low Baked Shadow Radius, like 0.1.

- In the "Environment" tab of the Lighting window, you can add a Skybox and have it contribute to the lighting. If your scene is indoors, make sure no skybox is assigned here or it will mess up your bake. In this case, also set Environment Lighting to "Color" and make that color black. In most cases, "Environment Reflections" should be set to "Custom".

- For outdoor scenes, change your light bake settings from "Baked Indirect" to "Shadowmask", and add one Directional Light set to Mixed lighting mode. If you use another mode, you'll have either light leaking indoors because of the lack of shadows on quest or reduced quality direct lighting and lack of shadows on PC.

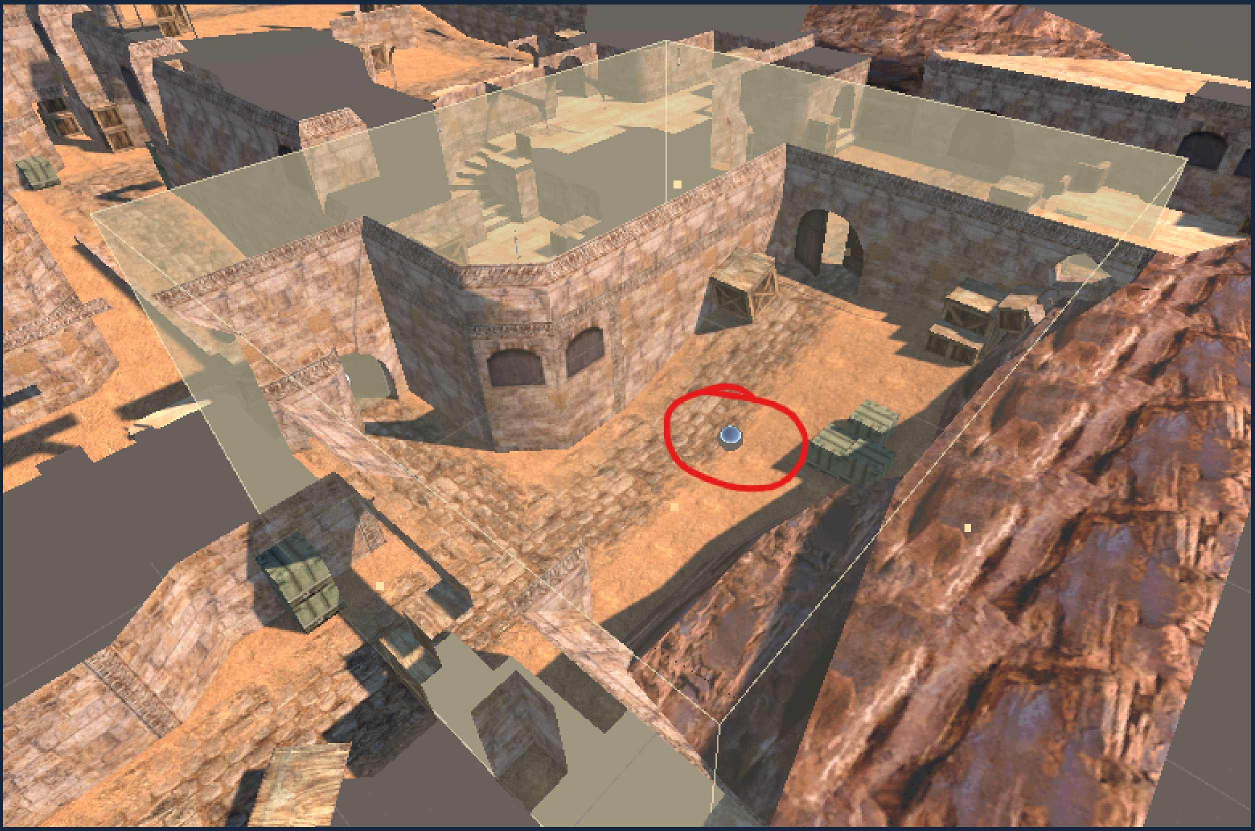
Probes

A lot of people are going to forget this step, but it's crucial. Often you'll be doing everything else right, but your scene will still look awful and you won't know why. These are why.

Reflection Probes

Once your scene's geometry is at least roughly complete, you'll need to add reflection probes. This step is absolutely mandatory for the scene to look good.

Each room or area will need its own probe. Set the probe at roughly player height, and adjust the bounds until they extend slightly past the walls or edges of the area, to account for blending. We like to have Box Projection checked. Reflection probes can be baked individually, independent of the lightbake. But baking lights will automatically bake all reflection probes in the scene.



A map like de_dust suits itself well to the same methods for probing that you'd use indoors, but for a big wide-open map like Tuscany or Wuhu Island, you'll want to go back to that Environment tab and change Environmental Reflections back to Skybox. Probe up areas in and around buildings normally, but then make some much larger areas for outdoors.

Light Probes

Just as essential as Reflection Probes. Dynamic objects like props, NPCs, and even the player won't adapt to the light without them. Your workflow for probing a scene is up to you, just make sure you have them. Have a higher probe density where the player is likely to be, less dense in areas they're not likely go, such as near the ceiling. But you *still need to have them everywhere,* except outside of map bounds, or inside walls.

Lightprobes interpolates between locations. Because of this, place more probes at locations with a high rate of change. For instance, place more probes near a shadow boundary so you can see items transition into shadow and light.

Baking Lights

Thanks Dall-E but that's not quite what I meant.

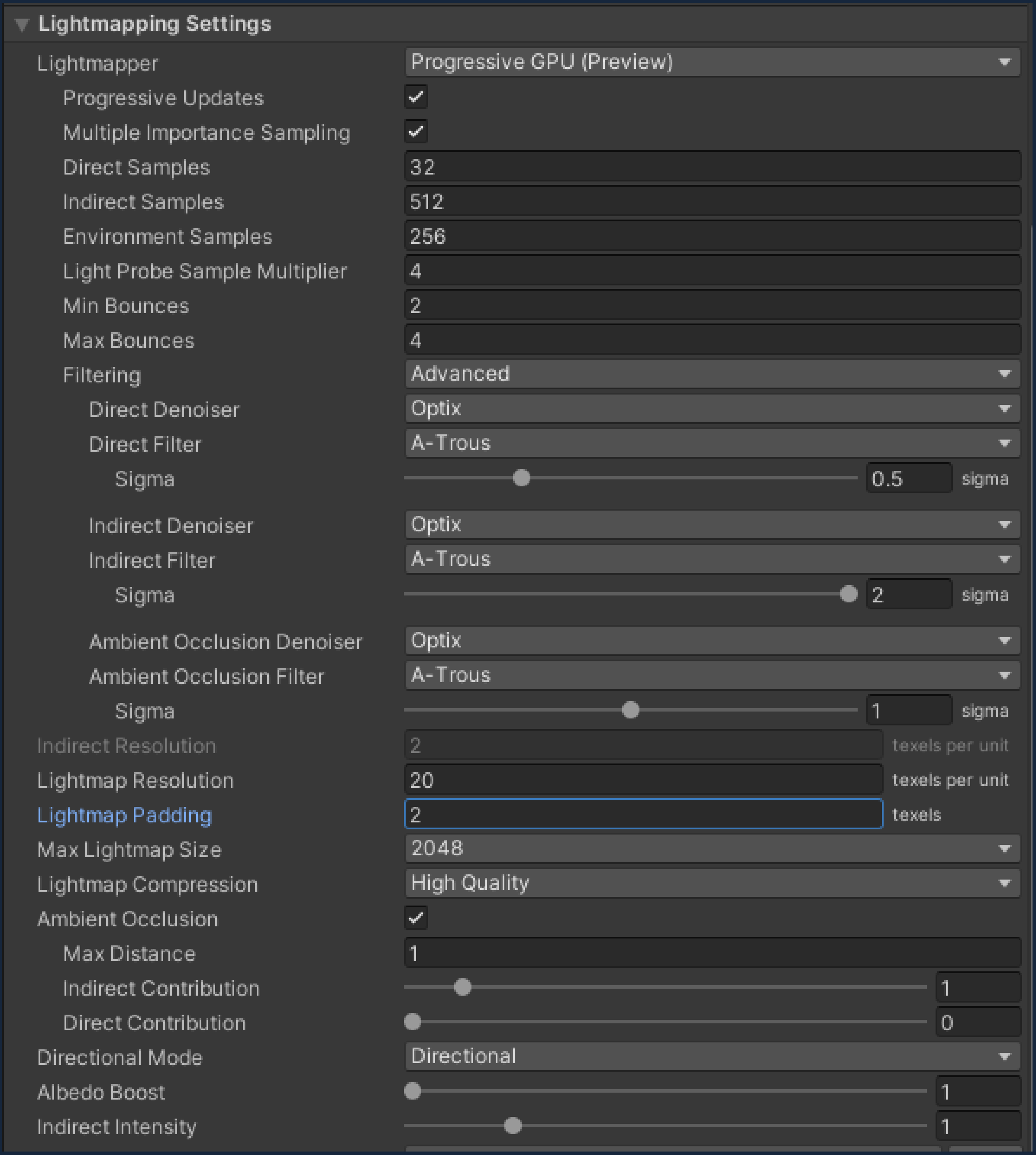
Unity's built-in lightbake system is actually pretty robust. Especially with the new GPU bake, you can bake an entire scene in minutes. Make sure you make lightmaps UVs when modeling, or if importing models, check the "Generate Lightmap UVs" checkbox on the model's import settings. Also, only Static items or items with the "Contribute GI" box checked under Static will be included in the bake.

To start, here's some starter settings I'll bake most scenes with:

Turn off Realtime Global Illumination.

Indoor: Set the lighting mode to "Baked Indirect".

Outdoor: Set the lighting mode to “Shadowmask” with a mixed directional light in the scene.



Lightmapping Settings

Lightmapper	Progressive GPU (Preview)
Progressive Updates	✓
Multiple Importance Sampling	✓
Direct Samples	32
Indirect Samples	512
Environment Samples	256
Light Probe Sample Multiplier	4
Min Bounces	2
Max Bounces	4
Filtering	Advanced
Direct Denoiser	Optix
Direct Filter	A-Trous
Sigma	0.5 sigma
Indirect Denoiser	Optix
Indirect Filter	A-Trous
Sigma	2 sigma
Ambient Occlusion Denoiser	Optix
Ambient Occlusion Filter	A-Trous
Sigma	1 sigma
Indirect Resolution	2 texels per unit
Lightmap Resolution	20 texels per unit
Lightmap Padding	2 texels
Max Lightmap Size	2048
Lightmap Compression	High Quality
Ambient Occlusion	✓
Max Distance	1
Indirect Contribution	1
Direct Contribution	0
Directional Mode	Directional
Albedo Boost	1
Indirect Intensity	1

- Bounces can gently control the overall brightness of your scene. It mainly affects how much lights bleed into different areas of your scene. Realistically this is infinite, but increasing bounces will also increase bake time. Turn down Max Bounces if your computer is struggling to bake a scene.

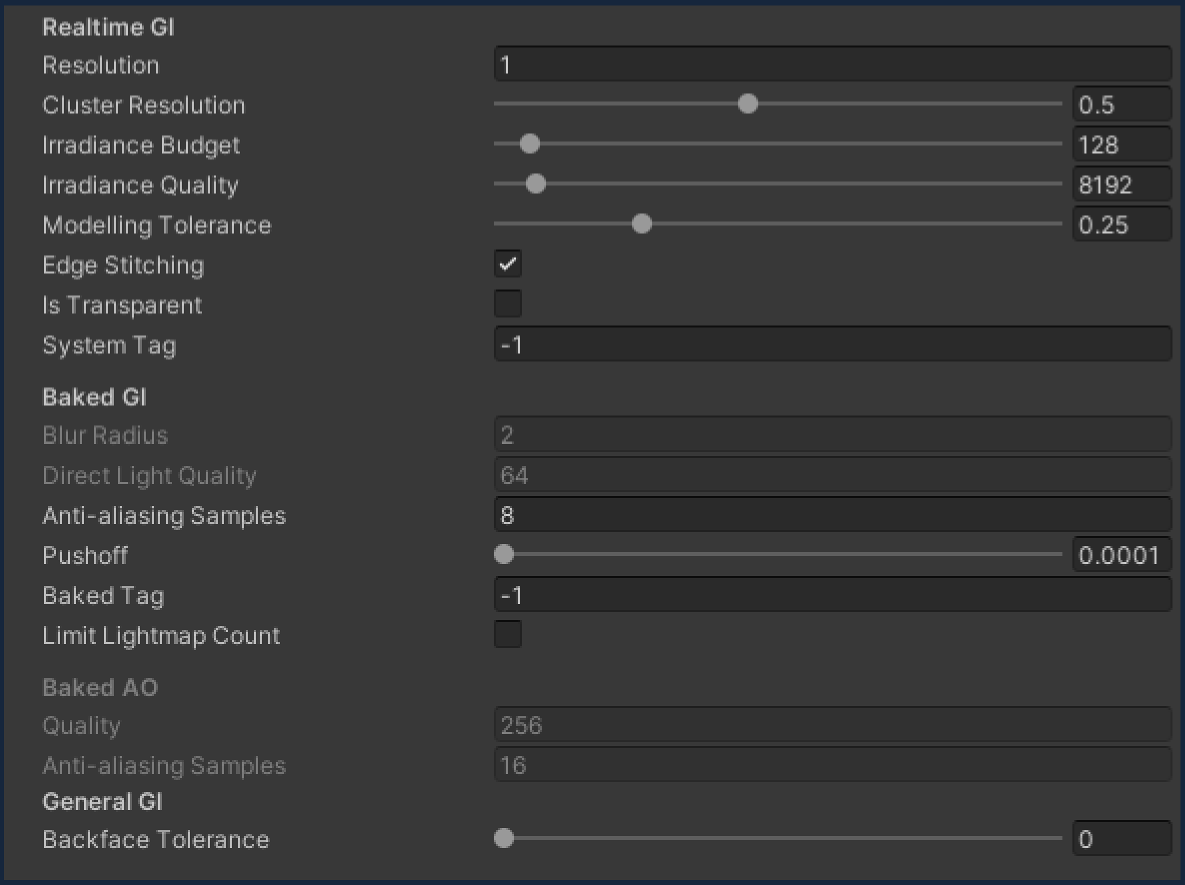
- Lightmap Resolution is different from the Lightmap Size: it determines how sharp the lightmaps will be. I usually use 10-15 for drafts, and 20-30 for final. Mesh Renderers also have “scale in lightmap” which is just a multiplier of this resolution. For very large surfaces, you can increase this value for sharper lightmaps, and conversely you can decrease the size for small objects that don't need as much detail. "

- Lightmap Size is merely how large Unity will allow the texture of a lightmap to be. It will make as many as it needs to bake the lights, so this setting will not affect bake quality.

- Ambient Occlusion is completely optional. Not realistic but looks great if done with subtlety. If curved surfaces are getting shadows in their edges or you see other shadowy problems feel free to turn this off.

- Directional mode is a must. This is how you get your baked specular highlights!

At the bottom of the lightmap settings is Lightmap Parameters. I don't know why Unity has these as a separate thing but whatever. I tweak these far less often. The only setting that really matters is backface tolerance. You’ll want to set that to 0 to get rid of some artifacts. Here's the settings we used in the vast majority of scenes in Bonelab:



Realtime GI	
Resolution	1
Cluster Resolution	0.5
Irradiance Budget	128
Irradiance Quality	8192
Modelling Tolerance	0.25
Edge Stitching	✓
Is Transparent	
System Tag	-1
Baked GI	
Blur Radius	2
Direct Light Quality	64
Anti-aliasing Samples	8
Pushoff	0.0001
Baked Tag	-1
Limit Lightmap Count	
Baked AO	
Quality	256
Anti-aliasing Samples	16
General GI	
Backface Tolerance	0

Fog

In the Environment tab of the Lighting window, you'll likely want to turn Fog off for indoor scenes. That's because we're going to use Volumetric Fog. Hell yeah, we fancy. You can use the base fog if you want, but you get better mileage with Volumetrics. Large outdoor scenes work well with the standard fog though. Turn it on and set the mode to Exponential Squared. Those other modes are square and lame. Don't worry about the color, it'll automatically pull the skybox reflection for fog color. Just set the density. You can also use the fancy volume controls we'll talk about later to get more control.

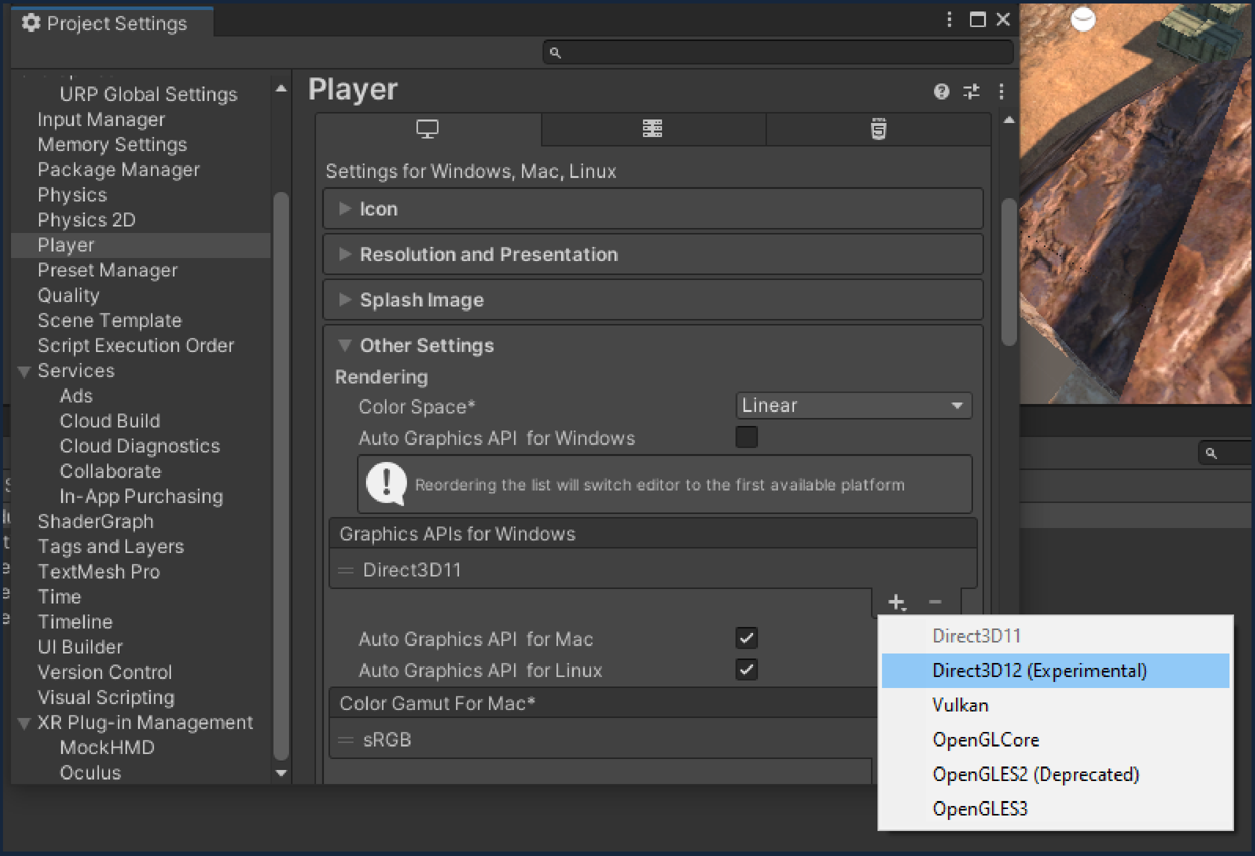
Volumetric Fog

First things first, we need to add a Volume for the global settings (naming conventions here get a bit screwy). Add a Volume to a game object, and Add Override for Volumetrics. Play around with these, as the settings will vary dramatically based on what kind of scene and vibe you're making. The fog view distance is the most important as it controls how dense the fog is. The light multiplier and albedo are unrealistic controls but can help tweak the look if used carefully. Mipfog and sky texture affect the fog and volumetrics.

Now that the scene has a Global setting, we make baked volumetric areas. For an outdoor scene this can just encompass the map* (kind of, see asterisk), but for indoor areas you need to add a Baked Volumetric Area for various rooms. As a rule of thumb, these can basically have the same frequency, size, and shape of your Reflection Probes.

*The volumetric baker will freeze up or crash when baking big volumes. Lowering the texel density of large volumes can fix this, but you may just need to break up large volumes into smaller chunks.

Once these areas have been made, it's time to bake. This can only be done in DX12, so switch your project settings while baking volumetrics.


Project Settings

URP Global Settings
Input Manager
Memory Settings
Package Manager
Physics
Physics 2D
Player
Preset Manager
Quality
Scene Template
Script Execution Order
Services
Ads
Cloud Build
Cloud Diagnostics
Collaborate
In-App Purchasing
ShaderGraph
Tags and Layers
TextMesh Pro
Time
Timeline
UI Builder
Version Control
Visual Scripting
XR Plug-in Management
MockHMD
Oculus

Player

Settings for Windows, Mac, Linux

Icon

Resolution and Presentation

Splash Image

Other Settings

Rendering

Color Space* Linear

Auto Graphics API for Windows ☐

Reordering the list will switch editor to the first available platform

Graphics APIs for Windows

Direct3D11

Auto Graphics API for Mac ✓

Auto Graphics API for Linux ✓

Color Gamut For Mac*

sRGB

Now, go to Stress Level Zero -> Volumetric Baking. For outdoor scenes, don’t worry about skybox contribution as the mipfog will account for it as long as you have a proper environmental reflection. For indoor scenes, it's likely that you won't have a Skybox at all anyway.

The bake is very fast. For small scenes it may not seem like it did anything. If you don't have any Baked Volumetric Areas, then it didn't. As long as those and the global Volume are set up, it should have worked. You can preview your volumetrics by clicking the Editor Tool button on your Tools bar and clicking "Show Volumetrics". This will bring up a window in your Scene view with a checkbox to preview volumetrics. Note that this view will differ somewhat from how it will look in-game, but is useful for seeing how your lights are interacting with the fog. This is where it will become apparent if you've gone too far with Intensity or other settings on your lights.

Physics

Okay, this is where things get interesting. You can disregard advice here as it pertains to the needs of your specific level. We're not the end-all-be-all experts of PhysX. But there are some really common troubles and mistakes people encounter when they don't really know what they're doing.

General tips

- Be realistic with Rigidbody weights. A gun or sword might be fine at the default 1kg, but a wooden crate will not. It's helpful to look up exact weights on Google so that you can stay consistent.

- Do not scale Rigidbodies. If you must, do so uniformly; don't scale differently on different axes. And for the love of Ford do not use inverse scale. If you need to change an object's size, it's best to have the mesh on a separate game object and scale that without scaling the rigidbodies or colliders at all. Definitely make sure you don't have cascading scales, by which I mean scaling a parent and then scaling the child again separately. This becomes a nightmare very fast.

- Avoid animated objects and moving kinematic rigidbodies. These things have infinite force, which not only is bound to cause massive physics bugs but also just isn't very ~~cash money of you~~ "Marrow". Instead use Configurable Joints or Spline Joints.

- A Hinge Joint is just a Configurable Joint with less settings. While it can be helpful to learn with them, eventually you're going to want to learn the full Configurable Joint.

- Do not leave physics-looking objects static in your scene. If you can move it in real life, it should have a rididbody.

Impacts

ImpactSFX and Impact Properties. This is not yet included in Marrow SDK. This will be updated if/when added. This is not a guarantee that will happen.

Configurable Joints

Configurable Joints are the single most powerful tool you will have as a modder. They're also a nightmare. The documentation is terrible and it seems like not even the people who made it fully understand what all the functions do. So I'm not going to teach you Configurable Joints here, you'll need to learn the basics yourself first. But here are some of the things I've learned while trying to make common things.

- "Spring" is used in two different contexts and does completely different things in each. Just keep this in mind.

- Forces are in newtons. That won't mean much to most of you. Ballpark, 10 newtons isn't a lot, 100 newtons is a decent amount, 1000 newtons is powerful, 10000+ is machine levels of strength.

- When troubleshooting, it can be helpful to add a ton of zeroes (to one value at a time). It may do ridiculous things, but it will help you understand the functions.

- If something is taking too long to get moving or come to a stop, use very high Spring/Damper forces and then cap its Maximum Force. This ensures the joint is always using its full power to reach the target.

- The X axis has more controls than Y and Z. If your object is only going to be moving along one axis, it can help to force it to be X. You can do this by setting the "Axis" to 1 for the one the joint needs to be moving along. So if you have a door that rotates on the Y axis, set Axis: Y=1, and now all the joint's settings will treat your Y axis as X. So you could use the X Limits, X Drives, etc. for rotating around the Y axis. A little confusing, you'll get the hang of it. Just make sure only one value is set to 1 or you'll have to start learning quaternions.

- Marrow has a built-in max break force, but it can still help to give most joints a break force (even if it's very high) to avoid physics bugs. Now that strong as hell avatars are in the game, this is especially important. 4000-6000 is a tough joint to break for a standard player.

- Perpetually moving items can be tricky, especially if they're not meant to be a strong force. Even if you set a Target Angular Velocity, the joint will frustratingly still be trying to move to your Target Rotation (likely 0,0,0). Setting a low Spring force relative to your Damper and having a high Max Force can help. The 250kg spinning Timber Roller obstacle in the Fantasy Arena has only 500 Spring force, with 1000000 Damper, and 50000 Max Force, with a Target Anglular Velocity of 1.35.

Miscellaneous

NavMesh

Bake one. Set the Agent Height to 1.85, Radius to 0.25. our AI can do slopes now, but not well. I wouldn't go above 30°. You can add a NavMesh Obstacle to dynamic objects that you want enemies to avoid, but be aware that these are fairly CPU-heavy.

Occlusion

Bake it.

There's more to it actually. Unity's built-in occlusion is quite scuffed but we work with what we've got. Occlusion is another big reason it's a good idea to break up your level mesh somewhat. If it's one giant piece, the entire thing will be rendering at once no matter where you're looking. Breaking it up lets things out of the player's view stop rendering. Occlusion is a balance between saving the GPU and hitting the CPU. If your occluder settings are too precise then your CPU is gonna have trouble, especiall on Quest. A Smallest Occluder setting of 2 is a good place to start, going as low as 1 if absolutely necessary. The smallest hole should be about 0.25. If you have smaller holes than that, consider making their surrounding objects non-Occluder Static. Not everthing *needs* to be an Occluder, but all Static objects will be by default. We've found that very long objects will create an unnecessary CPU burden while doing little to help the GPU. So set that pipe running the length of your hallway to be non-Occluder Static.

There's a lot more you could do, but that's the basics.

Performance

Meta has some guidelines on VR optimization here.

Adding to that, transparency is a big issue on Android platforms. This sucks, but avoid things like windows and decals as much as possible if you want your level to be playable on Quest. The more screen space they take up, the bigger the performance hit. This applies to any transparent material or shader.

Some Design Principles

I said I wouldn't teach you level design, but there are some things I'd like to get out there.

Remember, Marrow1 is a physics and interaction engine. Our philosophy is about freedom and no wrong answers. What this means is that things like invisible walls should be a no-go. If you don't want your player to cheese your map, ask yourself why. Would it be fun to jump over that wall instead of navigating the alley? Isn't it awesome to break a window and climb in rather than being forced to find a key? As often as possible, let the player do what they want.

This does make level bounds difficult. It's a big reason why we set our levels canonically inside of chambers with holographic walls. You can't just have a fence and say the player can't climb it. You've got to have physically consistent reasons why the player can't go where they want. This is a big design challenge, but an essential one for building in a pure-physics environment.

That's it for now. I'm going to add more as people's main struggle and missteps become more apparent, and as we add more features to the Marrow SDK. Feel free to hit me up with questions on the Lava Gang Discord server, #levels channel:
https://discord.gg/dhDFWP3Y

And hey, welcome home. Everything is possible with Lava Gang.