Section 1 - Introduction

Welcome to the Alloy Shader Framework Documentation! In this doc you will find a quick introduction to most relevant concepts of Physically-based Shading, and the Alloy Framework. Following this in section 2, there is a breakdown of the Alloy shader families and their variants. Finally, in section 3 you will find a detailed list of all of the parameters you will find in the various shader families, what they mean, with some tips for usage.

Alloy has been a passion project, nay an obsession for myself, and the exceptionally talented and dedicated Josh Ols, who two years ago approached me with the crazy idea of dragging Unity kicking and screaming into the world of Physically-based shading. Since then we here at RUST have made several games, a slew of prototypes, and the DX11 Contest winning Museum of the Microstar, all using various permutations of Alloy. We now gladly bring the fruit of our labors to the wider Unity community, and can't wait to see what incredible creations you all produce using Alloy.

-Anton Hand, Beard Warrior at RUST LTD.

Section 1.A - Core Concepts of Use

Energy Conservation: Alloy shaders use energy conserving equations. What this means in practice is that the total light output (excluding vfx elements like incandescence and added rim lighting) will never exceed the amount of incoming light energy from ambient and direct sources. There is no separate 'specular intensity' map or slider present here, which might take some getting used to. What this ensures is that regardless of your settings, the resulting material is likely to be physically-plausible, and will behave consistently in a range of lighting environments.

Alloy Packed Map: All Alloy shaders use a custom packed data map. This map is composed of 4 black and white channels, each which represent a different type of data used by the shader, and described in detail below. Note that if you generate this packed map yourself (which we don't recommend), you must ensure it is set to 'Bypass SRGB Sampling' in the Unity Texture Import settings.

Metallic (Red Channel of Alloy Packed Map): In the real world, metals interact with light in a very different manner than other materials. Internally, they have an albedo that is completely Black, and have a specular color that is very high. Nonmetal materials on the other have primarily a diffuse color, with a fairly low, and monochrome specular color.

Instead of having the user swap black into the relevant texture channel based on a material being metal/nonmetal, Alloy shaders have a 'Metallic' parameter, both in global slider and per-pixel form. This value basically feeds a Lerped value of your Base Color, and Black into the appropriate parts of the PBR equations, allowing you to easily make a material behave as a metal, or nonmetal object. Values between '0' and '1' of metallic are incredibly uncommon for real-world materials, but we've left the option open to you to do, in the case that you need some sort of weird exotic sci-fi material characteristics.

Occlusion (Green Channel of Alloy Packed Map): This channel serves double-duty in the Alloy model, as it is used not only for ambient occlusion, but as specular occlusion as well. This allows you to have areas of very smooth metals that can still be darkened by this map. Always remember that baked Occlusion/Cavity information should _always_ be placed in this channel and _never_ pre-multiplied into your base color maps.

Specularity (Blue Channel of the Alloy Packed Map): This channel behaves a little differently than the specularity you might be used to. Following the model laid out in Disney's 'Principled BRDF', this channel changes the specular intensity for dielectric(non-metal) materials, across a range exhibited by all real-world materials with exception to diamond. The

material reference graphs document contains common values for what should be used here for realistic values. If however, you don't want to worry about tinkering with it, you can almost always safely set this to middle-grey (which is how all standard PBR Materials work in Unreal Engine 4).

Roughness (Alpha Channel of the Alloy Packed Map): This channel controls the micro-surface information of the material, modulating from smooth to rough (maximum roughness being 1.0). In general, most materials are going to occupy the upper half (.5-1.0) of the roughness spectrum, with only very polished/smooth materials being lower.

Reflection Modes: Alloy shaders have several different reflections modes, available in a dropdown at the bottom of each shader. The shader may be in RSRM mode, which angularly tints the ambient contribution, and provides a generalized specular reflection based on roughness. When in RSRMCube mode, a standard cubemap can be provided which is lerped to for specular reflection based upon material roughness. The exposure of the cubemap may also be boosted. Lastly, all Alloy shader can be switched to Skyshop mode (either standard or box projection), in which case their ambient lighting will be provided by Skyshop SH, and their specular reflection will come from the Skyshop filtered specular cubemap. A custom specular cube may also be inserted.

RSRMs: Alloy shaders use a Look-up-texture called a Radially Symmetric Reflection map, which is an in-house solution we've put together for providing generalized reflectance information, for when generating and managing cubemaps is impractical. This map has pre-computed values for our full range of specular powers in it, and creates the appearance of a generic-horizon-line style range of luminance. What this ensures, is that light-mapped Alloy materials that are not being struck by _any_ real-time lighting will still appear reflective, and metal will still look like metals, without looking like mirrors.

Ambience: For Alloy shaders to exhibit their full range of visual character (such as grazing-angle Fresnel), it is critical that all objects have some source of ambient lighting. For that reason, scenes using lightmapped objects should also have a corresponding set of Light-probes for all dynamic objects to reference.

Lightmapping: While Alloy shaders are fully compatible with all three forms of Lightmapping found in Unity, we heartily recommend that you use Directional lightmaps for best results. If you do not do so, you will find that many matte materials feel fairly flat. Dual lightmaps may be cheaper at runtime, but for about the same storage cost, directional lightmaps yield a dramatically higher visual quality.

Section 1.B Performance Considerations

The setup that we've chosen for Alloy has been tuned to extract the greatest performance out of using a Physically-based setup as possible. However, it is important to understand where certain costs lay, in the features and data-set sizes that you will often find yourself using alongside Alloy:

- Deferred Rendering: which you pay a fixed cost for, which I of course feel is worth it 100% of the time for the ability to use lots of dynamic lights. Plus being able to use point-light and spot-light shadows is a must.
- Per-pixel Operations: Normal mapping, per-pixel spec, rim lighting, etc. aren't free of course. While Alloy contains plenty of exotic variations, you should use these will discretion as with any complex shader.
- Texture Data: Between directional lightmaps, and fat texture sets, you're likely going to find yourself using tons of texture data (our Museum of the Microstar project was 80% texture data on disc, but we used 4096s on everything). Don't expect most games using Alloy to be small downloads.
- Translucent and Grab-Pass Variants: Be aware that these shaders don't benefit from a lot of engine optimizations, and can get heavy FAST.

Section 1.C Project Setup

Before using this shader set, there are a couple things you _must_ set up in your project. If you do not set up these options, you will get visual artifacts caused by broken math.

1. Open Edit->Project Settings->Player

2. Open the 'Other Settings' rollout

3. Set 'Color Space' to 'Linear'

4. If you want to use lots of dynamic lights, set 'Rendering Path' to 'Deferred Lighting'

5. Select your camera in your scene

6. Check the 'HDR' box

7. Now save a scene, and CLOSE UNITY COMPLETELY **This is necessary for the custom deferred shader to be loaded to override Unity's default deferred shader. This is what allows us to use a BRDF in deferred mode.**

8. Open Unity and your project back up.

Section 2 - Shaders

Alloy contains several shader families, each containing a set of variants for RSRM and Cube-mapped reflectance. This section of the documentation is designed to give you a conceptual overview of the framework's content, and where and when it is best to use various shaders in the family.

Section 2.A - Shader Families

Core: The shaders found in this family are the work-horses of the Alloy set. You will find the simplest shaders here, which you will likely use for a majority of your objects. You will find variants with glowy rim lighting, detail mapping, decal mapping and masked incandescence in this family.

Nature: This category is currently empty. We're still working the kinks out of our terrain shader. We'll replace it soon!!

Particles: The shaders found here aren't Physically-based like the rest of the Alloy set, but are instead a couple useful HDR-compliant particle shaders that we've found ourselves using commonly alongside Alloy. These have intensity values that can go up to '8' for soooper glowy particles :-)

Self-Illum: The shaders found in this family have one core purpose. Namely they are configured specifically to allow for objects that you want to be emissive AND have that emission be sent to the Beast Lightmapper. If you simply want glowy bits on characters/vfx/etc. you likely want to use the shaders found in the Core family.

Note also that if you want colored emission to be sent to Beast, that color MUST be represented on the appropriate pixels of the Base Color texture.

Transparent: The shaders in this family follow the combinatorial logic of the Core family, except that they are all translucent.

To start with, there are a couple things you should understand about translucent objects in a Physically-based model. The first of these is that the energy-conserving nature extends to the way translucence is handled. Merely setting the Alpha value of the Base map/tint will not be enough to make a material fully transparent. The Metalness of the material will also affect how transparent the material will appear, as it controls the relative balance of _reflection_ and _refraction_.

The easiest way to think about this is that alpha darkens albedo, but doesn't affect f0. f0 darkens albebo, and raises alpha to opaque. Thus diffuse + refraction + reflection <= 1.0. If this is confusing to you, please play with the sliders on the sample glass materials found in the Demo Scene to see what effect they have.

The second important thing is that all translucent shaders in Alloy are Forward-rendering only. This means you will only see dynamic specular highlights from a number of lights equal to the 'max pixel lights' parameter found in your quality settings. I tend to keep this value at 5-6.

Transparent Cutout: The Alpha-cutout shaders can also be found in this set, which you can also use for shadow-casting particles!

Section 3 - Shader Parameter Sets

Alloy Shaders have sets of parameters which are broken up into tabbed groups. This section will cover those parameters, what their values mean in abstract, and some workflow advice for ranges that will produce good results.

Section 3.A – Primary Texture Properties

1. Main Color: This color picker will be multiplied against the Base texture below. Note that it's alpha value will only be utilized on transparent variants. This will affect the bounce and emission color for lightmap generation.

2. Base Color (RGB): This texture represents the base tone of the material. As it is the only per-pixel color map for the core shaders, this map plays 'double duty' when it comes to reflective surfaces. Essentially, it will inform both the 'diffuse' and 'specular' coloration of the material based upon the settings below. Note that the texture repeats/offsets values set here will determine the mapping for ALL of the other common texture maps. None of the other texture repeat/offset boxes will function. This map will determine the bounce and emission colors for lightmap generation.

3. Cutoff: This slider, found on all cutout variants, is used to control the interpretation of the Alpha channel of the Base texture, in determining which pixels of the texture are made 100% transparent. Note that cutout shaders ARE deferred-compatible, and will cast dynamic shaders in-game and when Lightmapped. Delicious

4. Normal Map: Nothing special here, just your run-of-the-mill normal map.

5. Material Map (RGBA): This texture represents 4 packed data-channels, as is best created using the included Alloy Material Map Packer (from input textures already imported into your project). The Material Map’s channels each represent Metallic(R), Occlusion(G), Specularity(B), and Roughness(A). If you do pack this map yourself, please ensure it is set to ‘Bypass SRGB Sampling’, or it will not function as intended.

Section 3.B – Physical Properties

1. Metallic (0-1): This slider is multiplied against the per-pixel Metallic value found in the packed map. It is not meant to commonly used (more for testing really), unless you have a need to dynamically change this value during use.

2. Specularity (0-1): This slider is multiplied against the per-pixel Specularity value found in the packed map. It is not meant to commonly used (more for testing really), unless you have a need to dynamically change this value during use.

3. Roughness (0-1): This slider is multiplied against the per-pixel Roughness value found in the packed map. It is not meant to commonly used (more for testing really), unless you have a need to dynamically change this value during use.

Section 3.C – Distort Properties

1. Transparency LM: This value is for passing a color value to the beast lightmapper (in the rare case that you would need to bake illumination on a distortive surface). This is needed only because of the peculiar requirements of Beast in Unity. It can be ignored in all other circumstances.

2. Distort Weight (0-1): This value is a multiplier for the power of the distortive effect overall. Its usage is for raising and lowering the power of the distortion while maintaining the relative relationship between Normal Map and Geo Normals that you have established below.

3. Distort Intensity (0-120): This value will determine the amount by which pixels behind the material will be distorted in screen-space by the normal map on this material. NOTE that materials will this parameter are using a grab-pass for their effect, so use them judiciously, as multiple grab-passes can obliterate the performance of complex scenes.

4. Distort Geo Weight (0-1): This value is a multiplier for how much you want the geometry normal of the object to influence the distortion direction. A '0' here will ensure that only the object's normal map influence the distortion.

Section 3.D – Self Illumination Properties

1. Illum Map (A): This texture mask will set what pixels of the Base texture will be counted as emissive, both for on-screen effects, and for what is sent to the Beast Lightmapper.

2. Illum Tint: This color picker will tint the emissive contribution of the material. The alpha value does nothing. Note that this tint will NOT affect the values sent to Beast.

3. Illum Weight (0-1): This value is a multiplier over the illumination effect that allows you to modulate its power while maintaining the relative relationship between tint and intensity that you have established.

4. Illum Intensity (0-n): This value will determine the intensity of the emissive contribution of the material. Note that this will NOT affect the values sent to Beast.

5. Emission LM: This float value will determine the intensity of the emissive contribution that will be sent to the Beast Lightmapper.

Section 3.E – Decal Mapping Properties

1. Decal Texture (RGBA): This texture map is alpha-blended on top of the base textures of your material. Note that the texture offsets on this input are used for this texture, as well as the decal normal and decal material map.

2. Decal Bump Map: Nothing special here, just another normal map. IMPORTANTLY, unity will not complain if you plug a texture in here that hasn't yet been set to 'normal map' in its import settings. Always check that you have done this before putting a map in here, or you'll get straaange results.

3. Decal Material Map (RGBA): This map is similar to the packed map, but has two core differences. The first is that it does not have a metallic or specular value for itself. The second is that the roughness stored is not actual roughness, but a variance generated by the material packer based on the decal normal map.

Section 3.F – Detail Mapping Properties

1. Detail Texture (RGB): This texture will be multiplicatively blended over the base color. Note that this will make materials using this map appear MUCH darker, so it is recommended that you use a very bright/subtle texture here. Note that the texture offsets on this input are used for this texture, as well as the decal normal and decal material map.

2. Detail Bump Map: Nothing special here, just another normal map. IMPORTANTLY, unity will not complain if you plug a texture in here that hasn't yet been set to 'normal map' in its import settings. Always check that you have done this before putting a map in here, or you'll get straaange results.

3. Detail Material (RGBA): This map is similar to the packed map, but has two core differences. The first is that it does not have a metallic or specular value for itself. The second is that the roughness stored is not actual roughness, but a variance generated by the material packer based on the detail normal map.

Section 3.G – Incandescence Properties

1. Incandescence Mask (RGB): This texture allows for full-color masking off parts of your Incandescence map, as it uses the Base Texture's texture repeats/offsets. Conceptually, what this allows if for you to have 'zones' on your model that are Incandescent that line up with the Base map at all times, but then scroll the Incandescence Map (which would in this case be a seamless texture with a nice color range), giving the impression of moving energy/shields/fire, or other effects. Several examples of this can be found in the Demo Scene.

2. Incandescence Map (RGB): This texture allows for full-color incandescence, which is multiplied by the Intensity parameter below. Note that the texture repeat/offset parameters fully work and are independent of the rest of the material's. Note that this form of Incandescence will NOT be sent to the Beast lightmapper.

3. Incandescence Tint: This color picker will tint/multiply against the values of the Incandescence map below. The alpha value does nothing.

4. Incandescence Weight (0-1): This value is a multiplier used for modulating the power of the effect, while maintaining the relative relationship of Tint and Intensity you have specified.

5. Incandescence Intensity (0-20): This float (which defaults to '0'), will multiply the contribution of the Incandescence map below. While fully HDR compliant (you can set this to up to '20'), it WILL shift the hue of your map, so you may have to compensate by using the tint above to get the final bloomed color to be at the hue you want.

Section 3.H – Rim Emission Properties

1. Rim Tint: This color picker will determine the color of the rim lighting. The alpha value does nothing.

2. Rim Texture (RGB): This texture allows you to have rim emission modulated by a full color texture. It also has its own set of scale/velocity inputs so that you can have this texture animate in UV space.

3. Rim Weight (0-1): This value is a multiplier used for modulating the power of the effect, while maintaining the relative relationship of Tint and Intensity you have specified.

4. Rim Intensity: This float value will determine the intensity of rim lighting. This parameter is HDR-compliant, so if you want your rim lighting to 'glow' ensure that this value exceeds the Bloom Threshold of your post-fx.

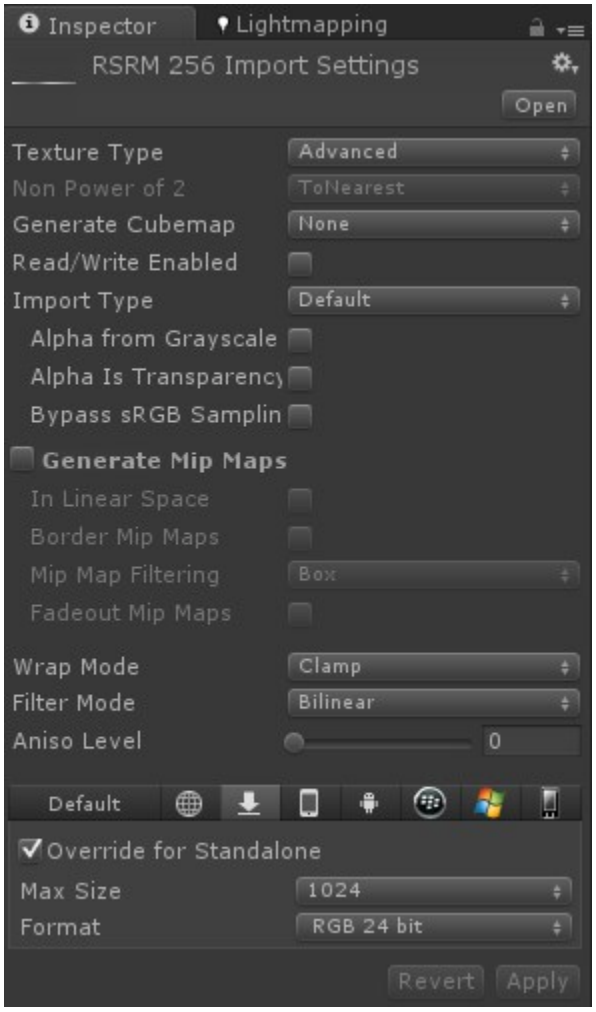
5. Rim Bias (0-1): This value is a multiplier which pulls the effect towards the inverse-view-direction (covers the object more). Really useful for some animated spell effects.

6. Rim Power: This float value will determine the breadth/width of the rim lighting, with lower values spreading the rim lighting out further towards the view direction.

Section 3.I – Reflection Properties

1. Reflection Mode: This dropdown allows you to choose between the 4 reflection mode options: RSRM, RSRMCube, Skyshop and SkyshopBox.

2. Radially-Symmetric Reflection Map: RUST's homebrew solution for generalized horizon-style reflectivity. This is a pre-computed Look-Up-Texture that MUST be set to the included RSRM 256 texture found in the Shader directory. Please ensure the import settings for this texture are setup to match the following image or you will have problems.



3. Reflection Cube Map: When the reflection mode is set to 'RSRMCube' , you will be able to plug a regular Unity Cubemap into this channel. Note that as this system was designed around sharp cubemaps, this map will be cross-faded with the output of the RSRM based upon the Smoothness of the material (as only mirror-smooth surfaces should perfectly reflect their environment).

4. Custom Specular Cube: If using Skyshop or SkyshopBox mode, you can input a custom specular cube here.