MFSSGI

By Michiel Frankfort - © 2022

Table of contents

- Table of contents
  - Introduction
    - Contact information:
  - Quick setup
    - The Basics
    - Build-settings
    - Core-settings
    - 2D sampling search radius
  - Realtime Reflection-probes & other scene Camera’s
  - Quality & Performance
    - Quality profiles
    - Multi-frame vs Single frame rendering
    - Screen banding
    - Mobile platforms & performance optimizations
    - VR
  - Post-processing volume override
    - Override setup
    - Additive & Pre-multiply compositing
    - Encoded light-directions
  - Shadows
  - Object thickness
  - SSGI-Object component
    - Available overrides
    - Lightstrips & Emissive objects
    - Flickering and ‘Twinkles’
    - Shadow flickering
    - Alpha & Depth clipping
  - Advanced settings
    - Raymarching
  - Mobile & WebGL build settings
  - Tips & Tricks

Introduction

Welcome!
I want to thank those who have supported me over the past months, indulging my crazy working hours to make this happen. Shout out to Unity forum-members supporting & helping me with great feedback and questions.

The reason I started working on SSGI is because at Little Chicken it became apparent that light-baking isn't. Also the out-of-the-box light setup in URP leaves much to be desired…

So MF.SSGI was born and I spent a couple of months investigating, building & testing what's required to get to this point. Big thanks for purchasing this asset! It keeps the lights on and I hope you will enjoy this asset!

Contact information:

For quick (support related) questions: Send me a Unity-forum DM or contact me on Discord:

- Unity forum: https://forum.unity.com/members/michiel-frankfort.683900/
- Discord: Michiel#6374

For collaborations, business related questions, etc, please send me a mail:

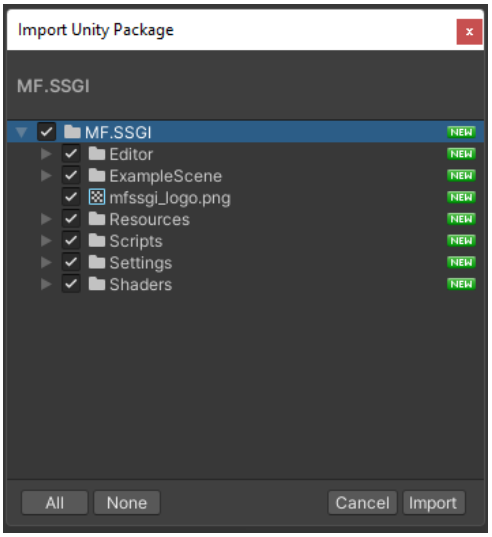
- Email: mf.ssgi@frankfortsoftware.com

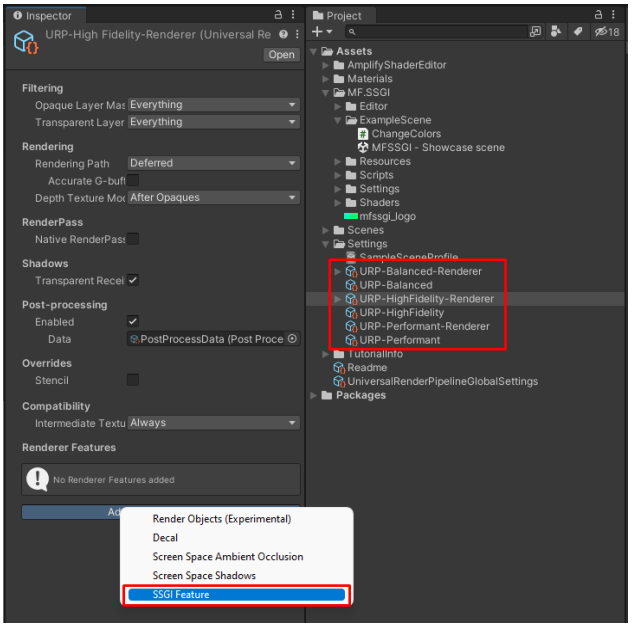
Quick setup

The Basics

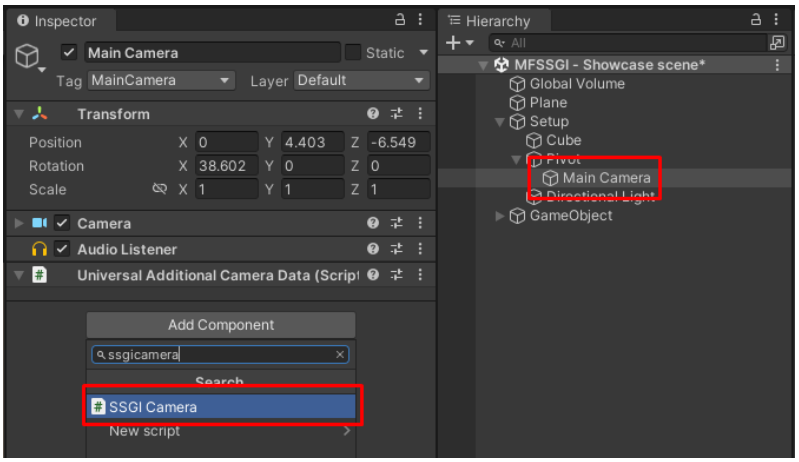
The best way to learn how to do a quick-setup is by **watching this video.** You can also keep reading and use the following steps below.

Start by importing the asset (which you most likely already done as you are reading this) and slect the desired UniversalRendererData-objects in the scene and add the SSGI-feature



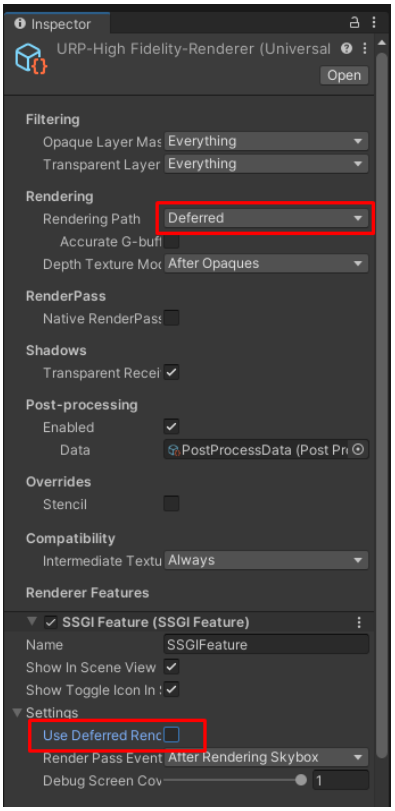


You will notice that the currently active Camera does not render SSGI just yet. In order to make that happen you need to assign the SSGICamera-component to the Camera you want to target.



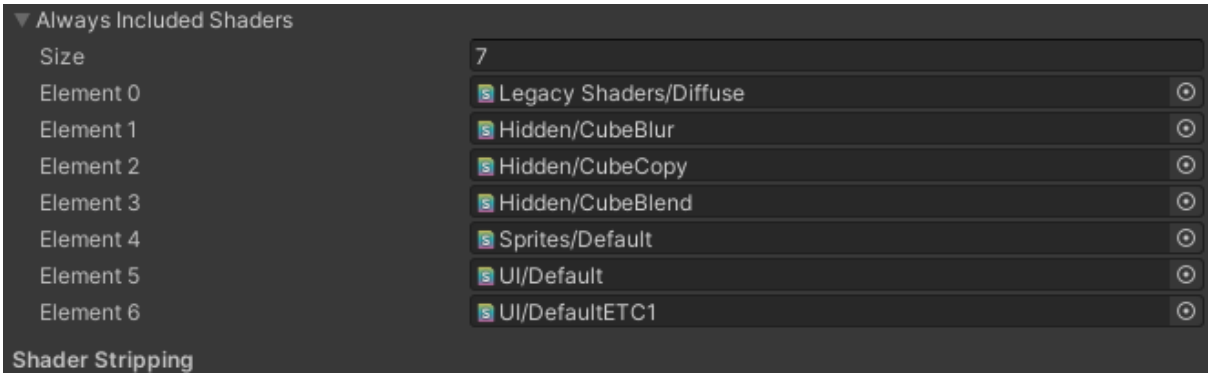
This is done to make sure no unneeded cameras are rendering SSGI as all active render-features are often also used for UI, minimaps, etc. Unity even uses them to render Reflection-probe, something you do not necessarily want.

Make sure to enable ‘Use Deferred Rendering’ if the URP rendering-path is also set to Deferred. **I highly recommend using Deferred** as it allows for far greater image quality when using SSGI and other post-process effects. Long story short: It allows to properly deal with Metallic/Smooth objects and has greater image detail in darker areas due to the use of GBuffers. Please note that not all target-platforms support Deferred, such as Mobile and Web-GL.

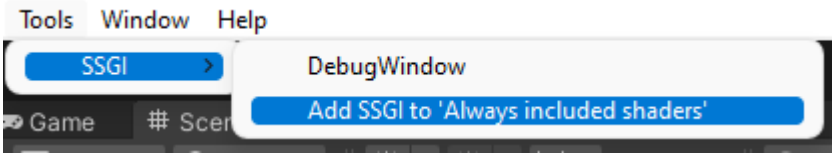


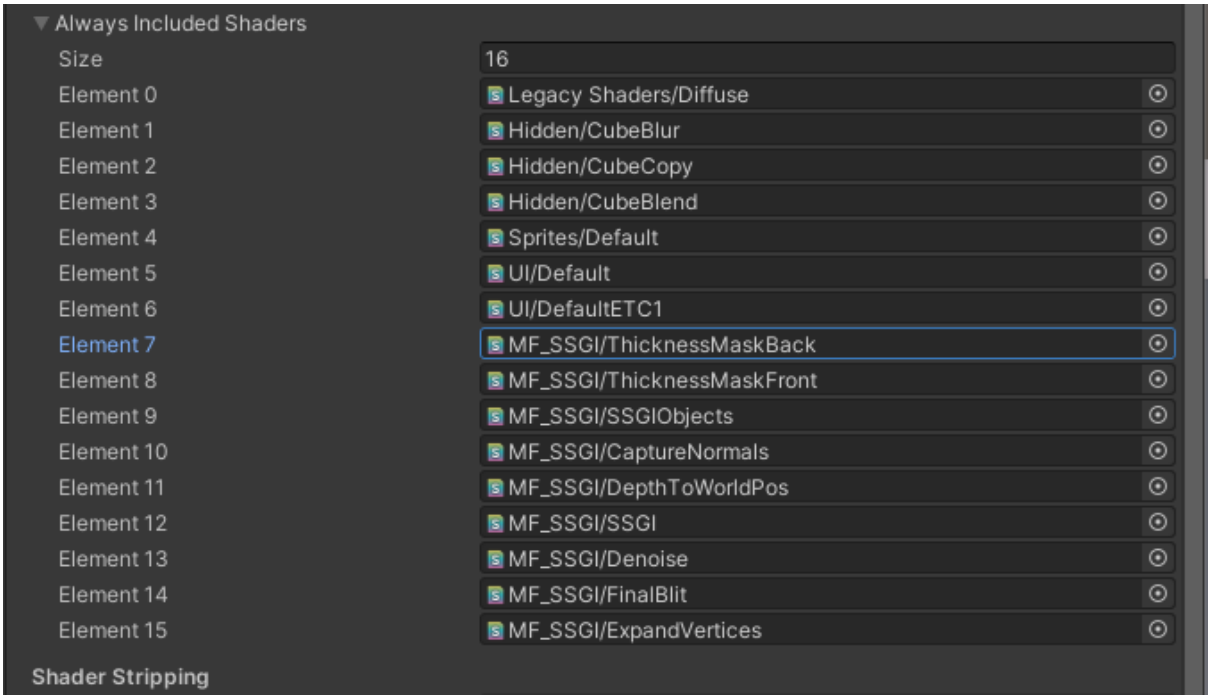
Build-settings

If you want to start making builds with MF.SSGI, we need to make sure the shaders used are included into the build. This has been automated, so you do not need to do that manually. To check if all went well, open “Edit -> Project settings -> Graphics” and you will find the list with “Always included shaders”.



Press “Tools -> SSGI -> Add SSGI to ‘Always included shaders’”.

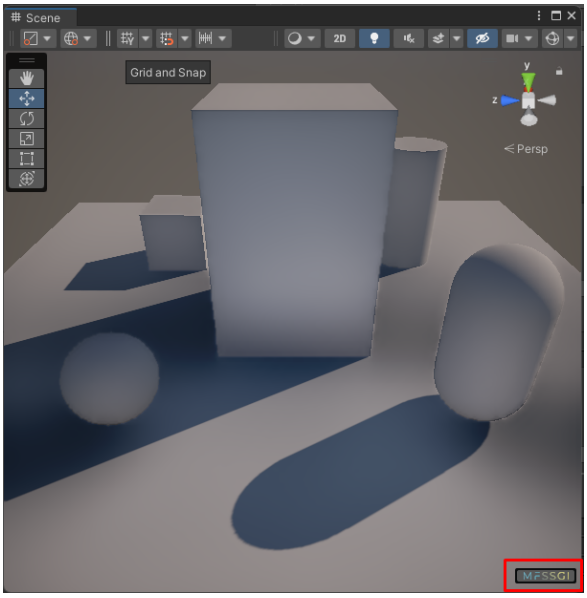


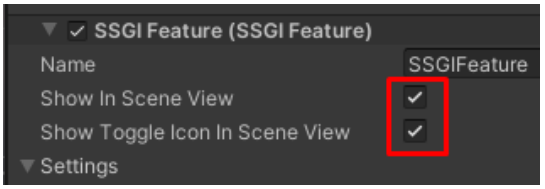


You will now find that a whole bunch of MF_SSGI shaders has been added to the list. Make sure to save the project and you are ready to build!

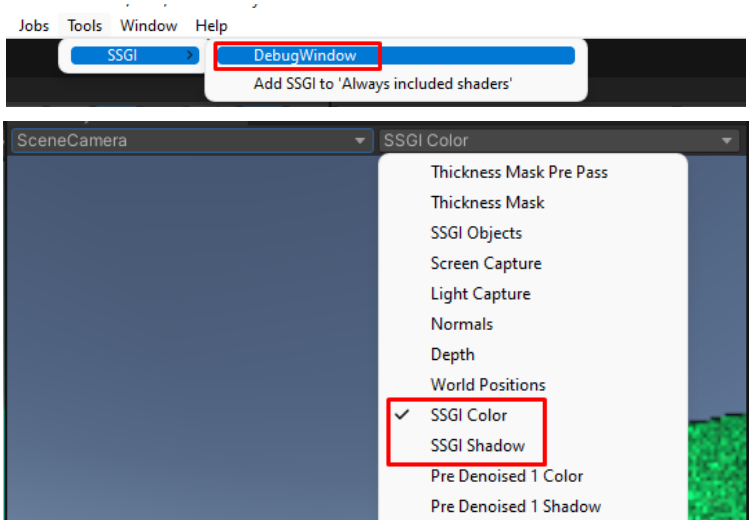
Core-settings

You now have the ability to toggle SSGI On/Off using the Scene-view button over at the bottom-right corner. This allows for easy tweaking and comparison. You can hide the button and/or disable the scene-view rendering over at the SSGI Rendering Feature settings.

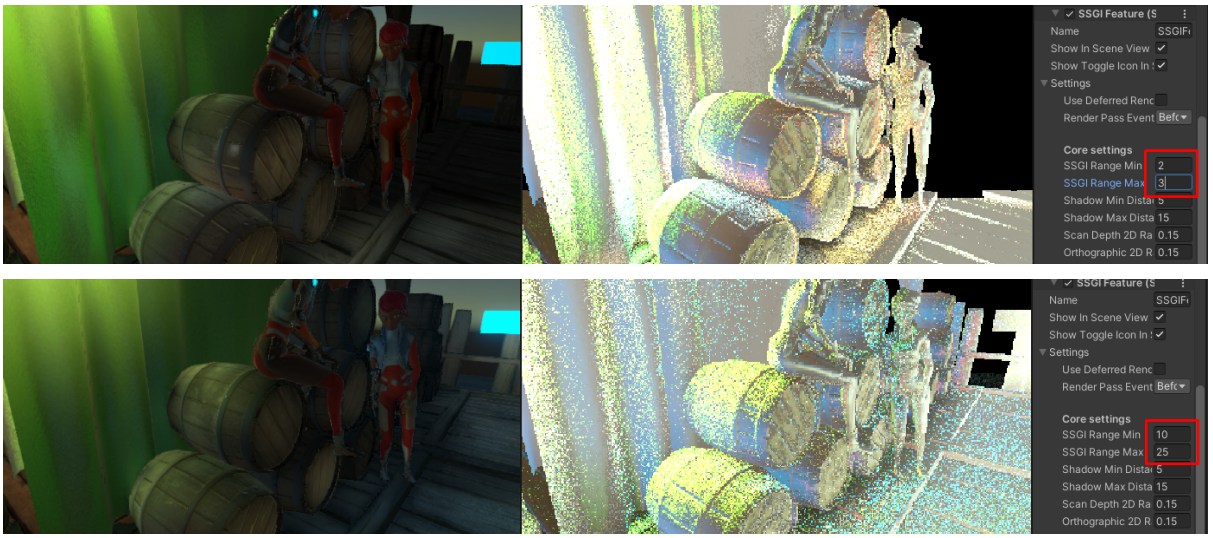




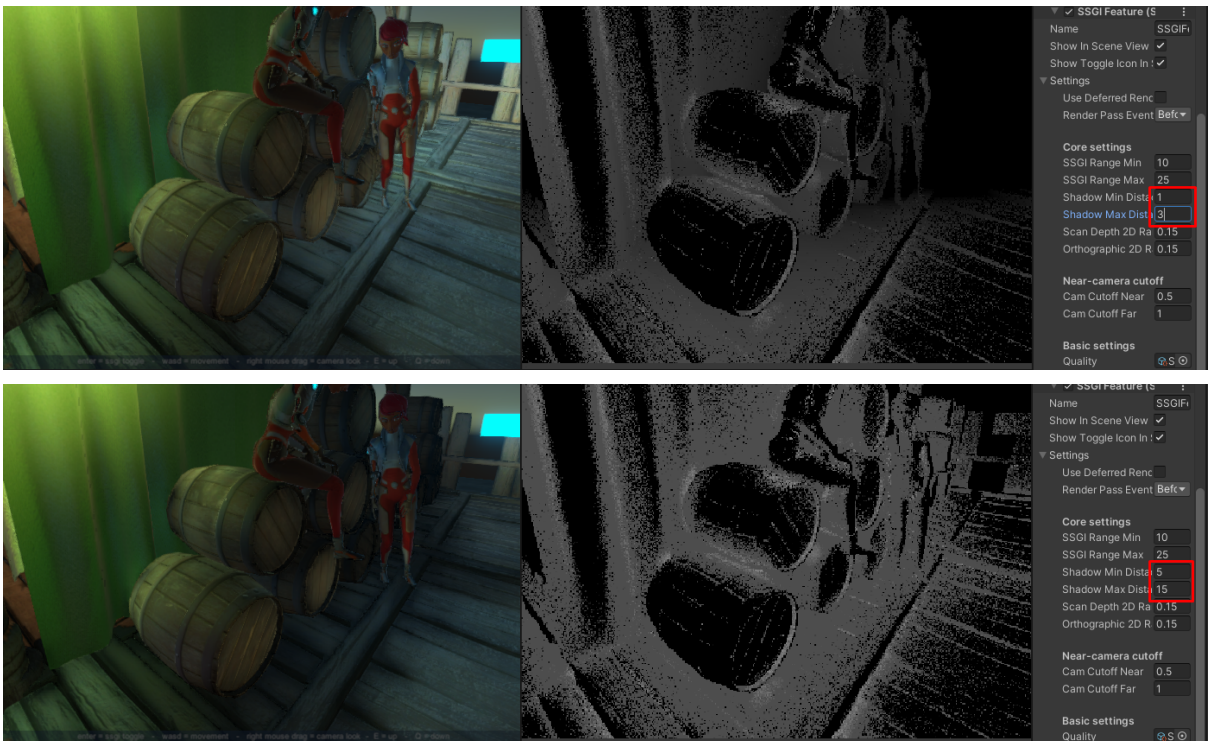
There are a few core-settings that you need to set up first**.** The best way to do this is by opening the SSGI Debug Window. This allows you to view and debug all the render passed individually. Let's start by selecting the ‘SSGI Color’ pass.



You first need to setup the scene-size. The question here is: How far does SSGI need to be rendered? For example: in the VIking-village SSGI is very nice on nearby objects, but the mountains in the background don't need to receive any GI lighting. Make sure the Minimum distance is always lower than the maximum distance. Between the Min/Max values the original screen-color fades back in. Beyond the Max-value, SSGI stops rendering all together. You can test this using the debugger as visualized below.



The same applies to the shadows: you can use the Min/Max distance to set the maximum range to which shadows are rendered. This is especially important when you want to limit flickering in the distance and/or save performance.



2D sampling search radius

This is most likely **the most important settings** to tweak before touching Quality and Lighting-settings.

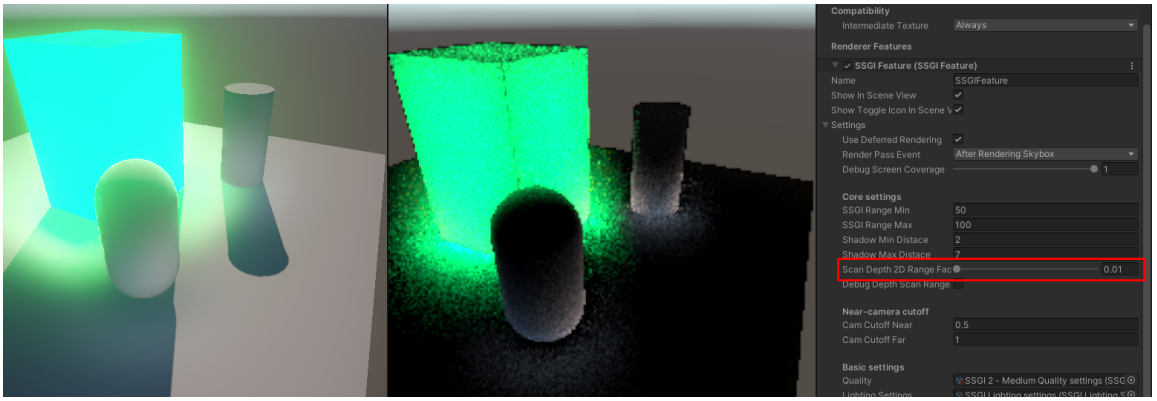
An important concept to the quality and image stability of SSGI is the distance used (in 2D-screen-space) to sample the scene. Since this is a post-process effect, think of it as a ‘blob of pixels’ being scanned in order to light 1 individual pixel. The more samples we use, the higher the quality and GPU cost.

The ‘Scan Depth 2D Range Factor’ is a multiplier on the ‘SSGI Range Max’ value (i.e. if set to 0.1, 10% of the SSGI-Rane-Max is used). This value determines at which distance the entire screen needs to be sampled. Beyond that distance, the 2D sample-range tapers off equal to the amount of perspective projection.

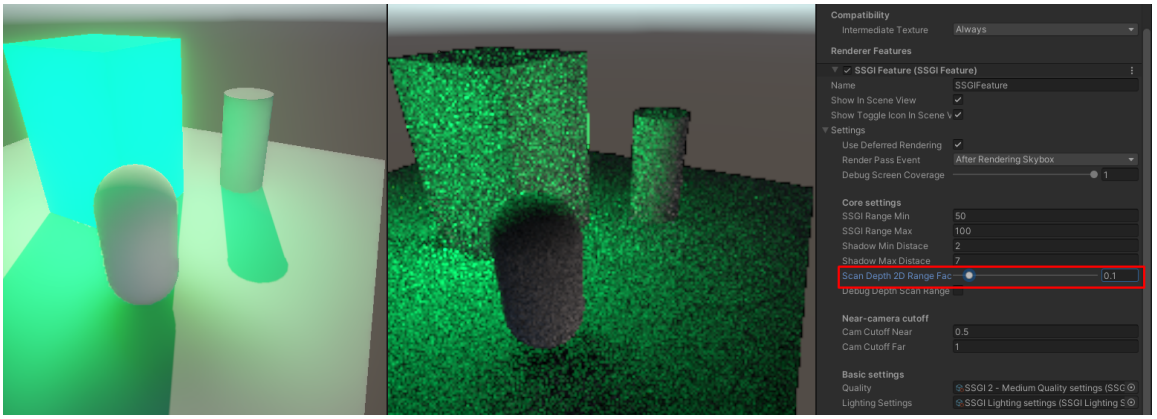
A general rule of thumb is:
The lower, the sharper as more samples pack closer together (i.e. best used for top-down).
The higher, the more ‘natural’ light falloff looks (i.e. best used for 1st-person perspective).

So start low, increase this value by tiny amounts to the point where the light scattering reaches a distance that makes sense to your project setup. This is very scene/project specific, so therefore I was not able to automate this properly and it needs to be set by hand.

Note: this doesn’t replace physical light attenuation, it's just a soft limiting factor to the 2D search range.



Set to 0.01



Set to 0.1

Realtime Reflection-probes & other scene Camera’s

The issue:

Unity uses the Default-renderer for EVERYTHING! To name a few: Scene-camera, UI, and most importantly: Realtime reflection-probes.

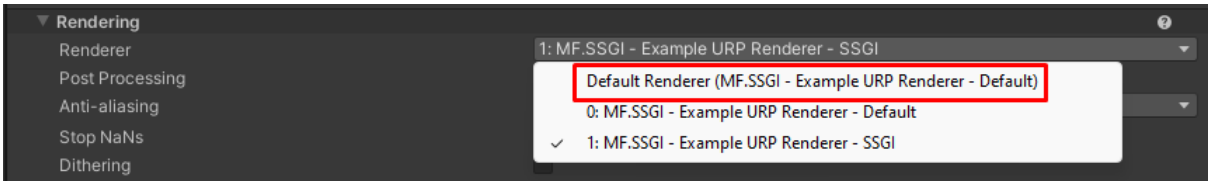
This means that if you add MF.SSGI (or any other post-effect for that matter) to the Default-renderer, they will be applied regardless! Of Course you still need to add the SSGICamera component, but that’s just to allow for easy Enabling/Disabling of the effect.

Unity still renders Depth/Motion-vectors/Normals for any post-effect that is attached to the renderer, regardless if it’s really needed. Yes, motion-vectors are even rendered on UI and real-time Reflection-probes, god knows why… (well it actually makes sense, it's just obfuscated and totally unnecessary).

So make sure to create a separate Renderer, attach all the post-effects there, and keep the Default-renderer as clean as possible. Use the profiler and Frame-debugger to test and see if the Realtime reflections and other cameras are ‘clean’.

The solution:

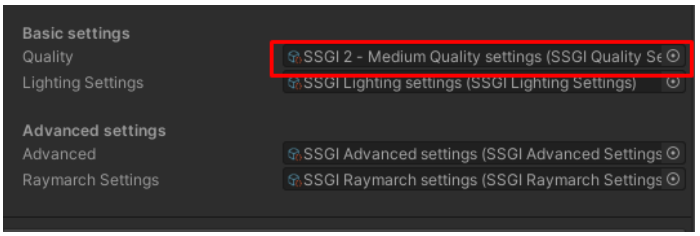
So make sure to create a separate Renderer, attach all the post-effects there, and keep the Default-renderer as clean as possible.



Quality & Performance

Quality profiles

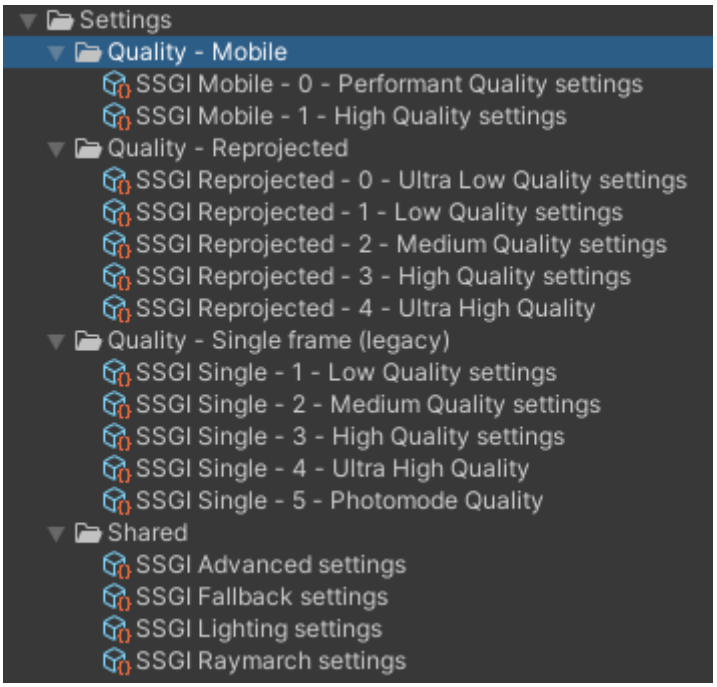
In order to honor Unity's Quality levels, I decided it's best to fully detach artistic-settings (like colors, intensities, etc) from quality-settings. I often see post-process effects that have all settings in one giant list/stack. So if you then set up multiple RenderPipeline-assets, you have to manually copy/paste not just quality-settings, but also artistic-settings. This makes it hard to tweak your look and feel as it is prone to errors. So that's why SSGI does things differently and stores everything in Scriptable-objects or in the Post-process volume override. This allows the user to hot-swap the Quality settings without any issues.



By default the package has multiple quality presets. They primarily differ in the amount of samples used for each pixel and resolution at which SSGI is rendered.

Since SSGI supports single-frame GI, you can use the Photomode to get high-quality single-frame renders for offline rendering. On my laptop GTX 3070 it's still real time, but very slow.

The low-quality settings can be used, although lots more artifacts will be visible. If you are considering using it for Mobile, please visit our 'Mobile platforms' section below before resorting to the lowest possible profile.

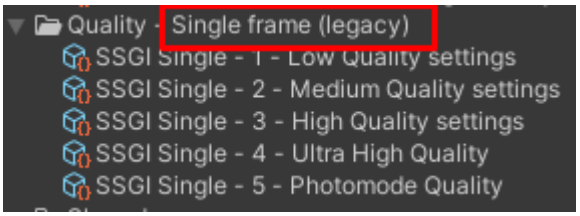


Multi-frame vs Single frame rendering

In V1.1 MF.SSGI supports multi-frame rendering. This means that GI from previous frames are ‘recycled’, allowing to spend more GPU time on single ‘new’ pixels. This greatly improves image quality while at the same time lowering the GPU cost. To compare: V1.0 Android build ran 30 Fps with far fewer samples then the new V1.1 which ran at 60 Fps.

Some of those performance gains were spent on improved Light-direction-encoding, making the visuals much nicer at a little more overhead. Some of the performance gains were also spent on an increased render-scale. Overall the WebGL and Android versions still run 2X the framerate as V1.0, including greatly improved visuals.

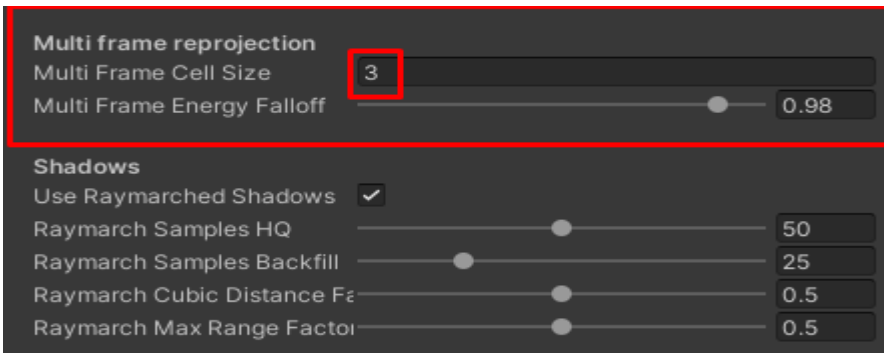
Since Single-frame rendering is not really needed any more (only if you have super-fast moving objects), these profiles are now moved to the ‘Legacy’ folder.



How it works:

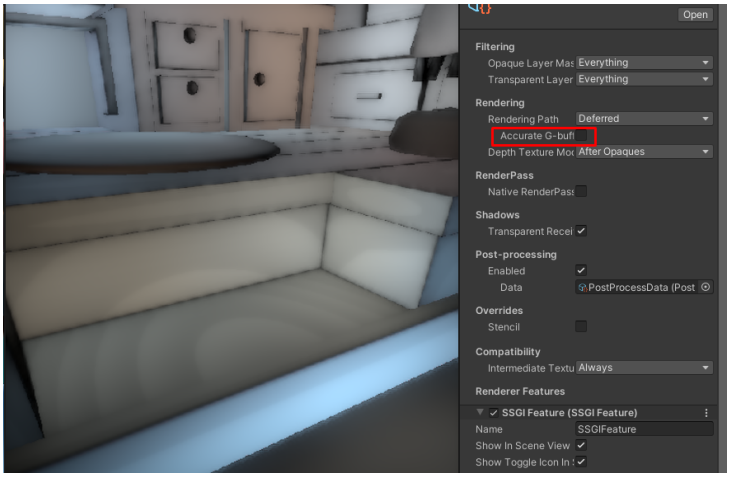
If you want to disable Multi-frame rendering, set the ‘Multi Frame Cell Size’ to zero. This will also disable the need for Motion-vectors, saving some drawcalls. However, as mentioned in general Single-frame rendering is more costly.

The cell (in this case 3x3 = 9 pixels) is updated 1 pixel per frame. Although the pattern is randomized, it is in a fixed order, making sure all pixels are re-rendered every cellsize*cellsize frames. The ‘Multi Frame Energy Falloff’ is a multiplier on previous frame, darkening the previous pixels, making sure that light+shadows from moving objects nicely fade away. If you increase the cell-size, it will take more frames to fully update, creating more ‘ghosting’. Note: a Cell-size of 2 or smaller will cause image instability.

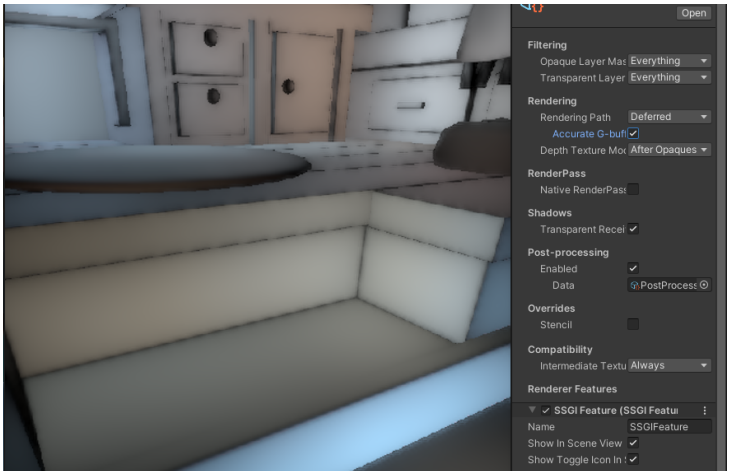


Screen banding

If you are experiencing banding issues, usually this is caused by Unity rendering the normal-pass at a LDR quality. This can be fixed by enabling ‘Accurate G-buffer normals’.



‘Accurate G-buffer normals’ disabled



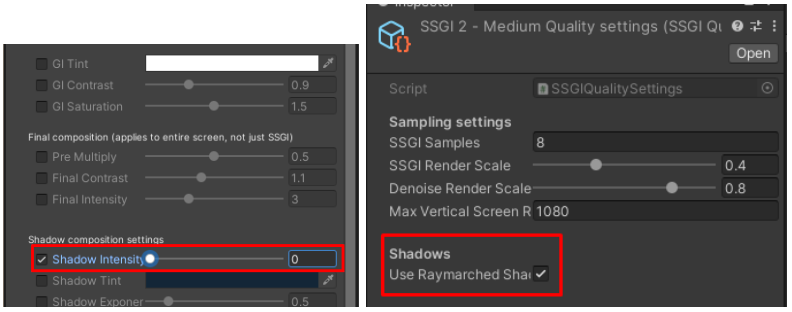
‘Accurate G-buffer normals’ enabled

Mobile platforms & performance optimizations

SSGI is very very memory intensive. Not in amounts of memory used, but memory-throughput constraint. Basically we need to sample textures millions of times each frame and perform some serious light-theory magic to make it look good.

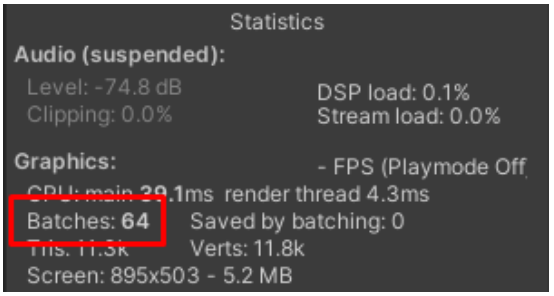
Therefore on mobile I advise you to disable shadows as they require vast amounts of additional samples per pixel rendered. For example, if the shadow-samples are set to 30 and the SSGI-samples set to 8, foreach pixel roughly 8x8x30 texture-lookups are performed. When disabled its just 8*8 samples, 30x times less samples required.

Disabling shadows can be achieved by setting the Shadow-intensity to zero, but that requires different post-processing volumes per platform. It's easier to head over to the Quality-profile used for mobile and disable it there.This way you can have shadows on PC, but have them disabled on Mobile.



In my experience Mobile platforms are greatly draw-call limited. Although SSGI is a post-process effect, if set up incorrectly, a load of more draw calls can be produced due to SSGIObjects-components in the scene. If you are using shadows, the ‘Thickness Mask Layer’ (over at the Raymarch settings) needs to be checked as well, to prevent Unity rendering objects that are not needed.

You can test this by opening the stats-window in your Game-view and toggling SSGI On/Off. If set up incorrectly, you could easily see it double or even triple in some rare cases.



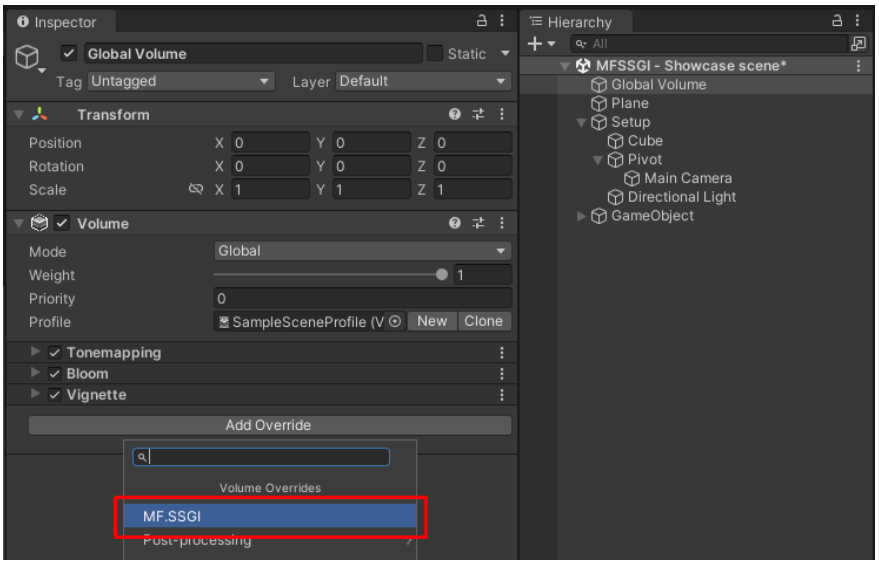
VR

At the moment VR is not supported. Even if it was, I believe the high requirements of current VR platforms (i.e. Oculus Quest: 2x Camera’s - FHD - 90 fps + on basically an IPhone 7) are not suitable for SSGI. Perhaps in the future this will change, but for now serious VR-developers know the struggle of achieving the required performance compliance.

Post-processing volume override

Override setup

In order to change Lighting-settings, make sure a post-process volume is in the scene and press the ‘Add Override’ button. Select MF.SSGI.



Additive & Pre-multiply compositing

SSGI creates light information that can be added to the final result. In case of just bounce-lighting this makes most sense as indirect light adds to the final brightness of the image.

However, the SSGI pass sometimes contains beautiful shading that simply is lost when just adding it to the final result. Instead by pre-multiplying the SSGI pass one can ‘reshade’ the image, creating an improved sense of realism and contrast. By doing so the image gets darker, therefore it is advised to compensate for this by increasing the ‘final-intensity’.

SSGI disabled

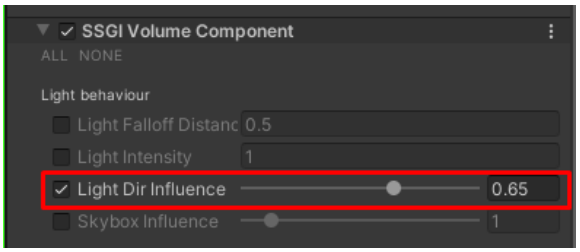
SSGI enabled, but only additive

SSGI enabled, premultiplied and additive

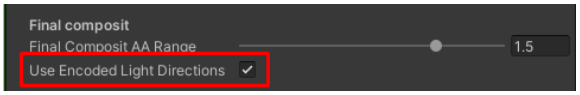
The only downside to fully pre-multiplying it is the loss of detail. As the SSGI is rendered at a lower resolution, pre-multiplying makes the low-res artifacts more visible. This however fully depends on the scene setup as dark and gritty scenes don’t show it as much as bright and clean scenes.

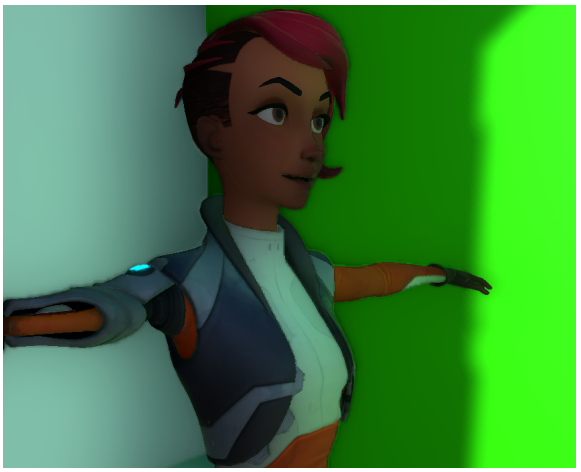
Encoded light-directions

In order to increase realism and boost normal-maps you can play with

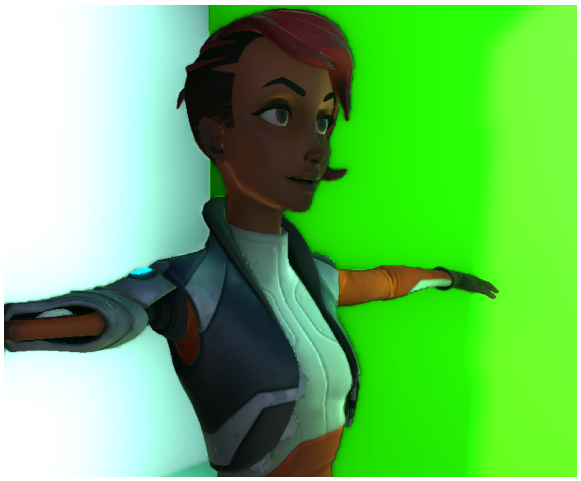


When turned off, SSGI switches to Half-precision HDR and does not encode the light-directions. This saves 5-10% performance, but will result in flat-shaded visuals. It can also be turned off over at the Quality-settings so you can override this per-platform.

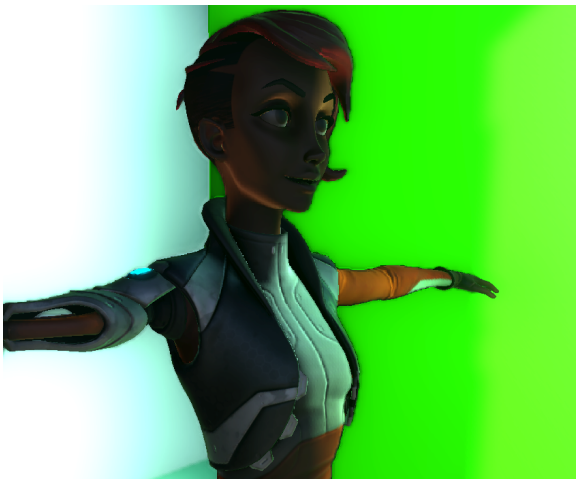




Value set to Zero:
Very soft, flat and toon-like shading. It does lose detail in normal-maps.



Default value: Nicely boosts the normals but keeps it soft, steady and natural looking



Maximum value: Pure Lambert-shading, boosting normal directions but looks ‘hard’ and moves around a lot as the camera moves around.

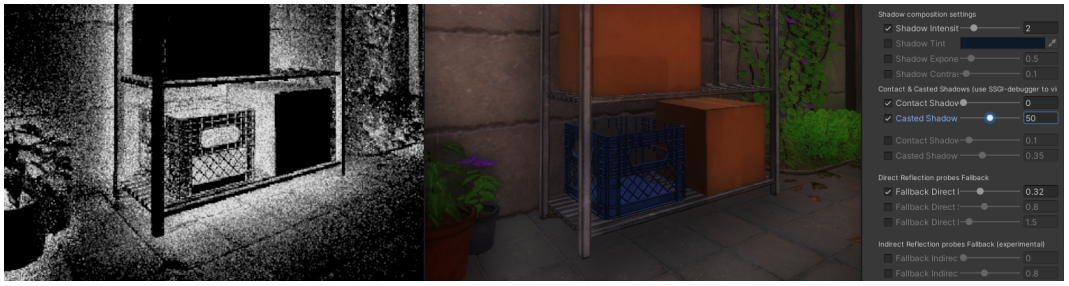
Shadows

Contact & Casted shadows

SSGI supports two types of raymarched shadows, both calculated in one go so they can be used at the same time without any performance hit. Note: Please visit ‘Quality & Performance’ down below for more information.

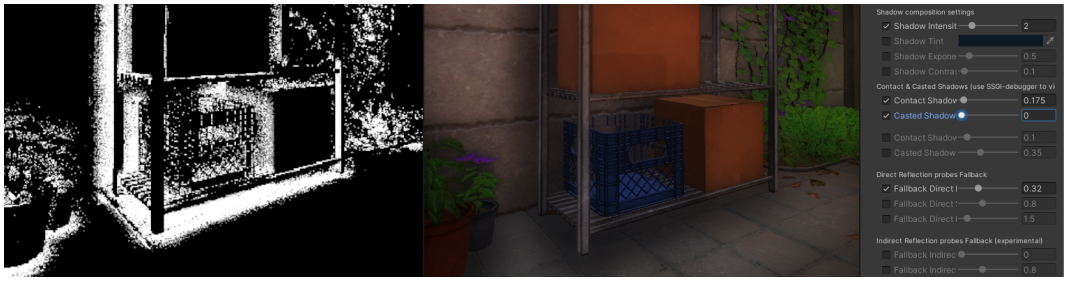
Casted shadows:

Each pixel functions as a light-source. So when light is blocked, SSGI will not illuminate that pixel. The casted shadows are the subtracted difference between SSGI-without-raymarching and SSGI-with-raymarching. By subtracting the two we get a nice soft-shadow-pass that can be used to occlude the already illuminated surfaces (by direct lights or lightmaps), originating from pixels that actually emit/bounce light.



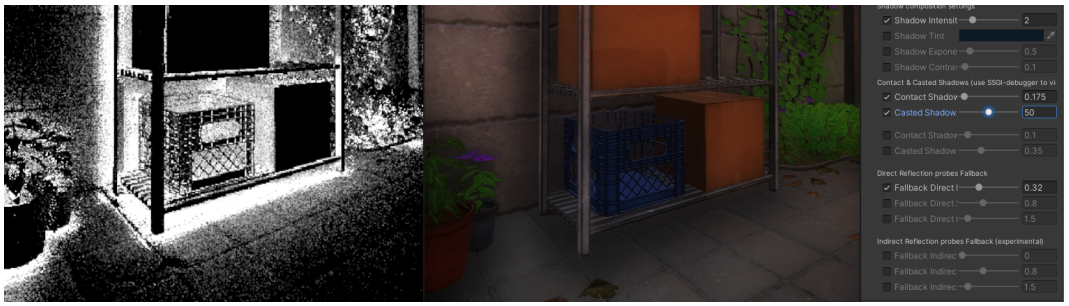
Contact-shadows:

They use the same ‘rays’ as the casted shadows, but there is a distance-check applied. This results in shadows much like traditional Ambient Occlusion, but since they originate from actual light-source pixels, the result feels more physically correct.



Combined Contact & Casted shadows:

In this case the Casted-shadows have soft-knee, making sure they appear less intense then the Contact-shadows. Using the soft-knee values you can mix it to your liking.

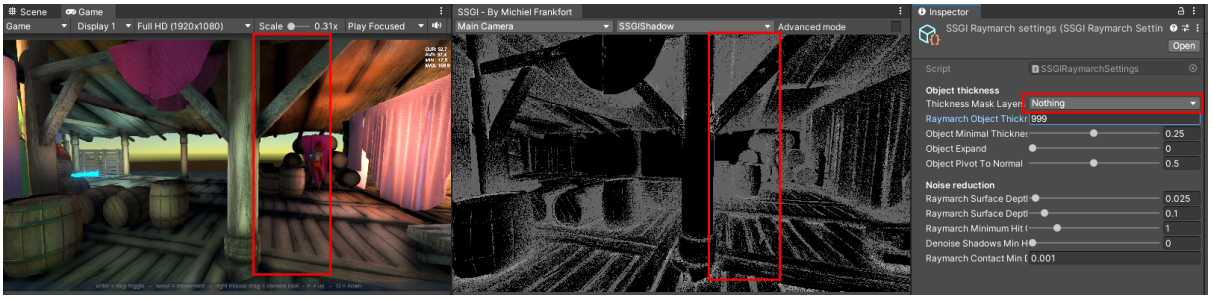


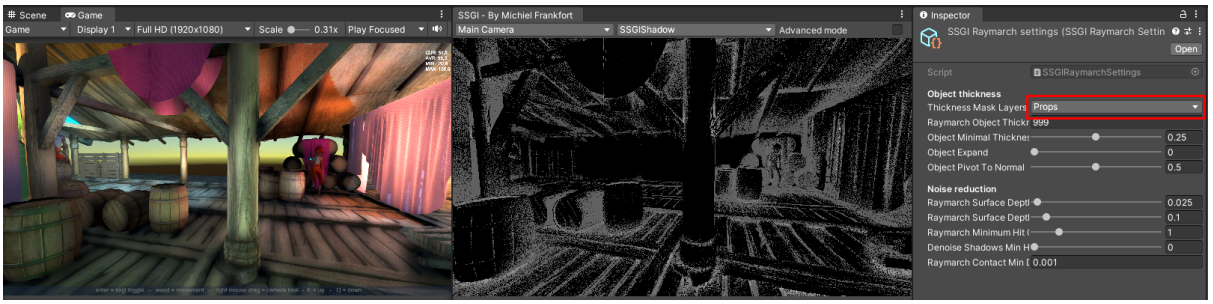
Object thickness

Thickness Mask Layers: Why are they important?

SSGI is a screen-space effect, therefore the shaders are unable to tell how ‘Thick’ objects are in 2D space. In the example below, the light that comes from the left-side of the screen is blocked by the pillar in the foreground. This should not be!

The pillar is not an infinitely deep/thick wall, so light should be able to pass behind it. By assigning the pillars to the props-layer - and by selecting the props layer to the Thickness-mask - SSGI is now able to tell that this a relatively thin object and light will not be blocked behind the object.





Limitations:

Be careful with what you select. By default the layer is set to Nothing. Selectively add objects that you really need to be ‘unblocking’ the shadows. Each object in the layer will be rendered to the thickness-layer and will have a negative effect on drawcalls.

The mask itself is 2D as well, meaning that only the object nearest to the camera will have thickness information available. This can be debugged using the SSGI-debugwindow.

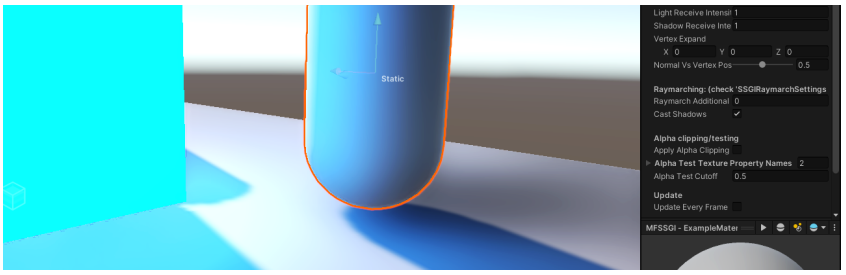
SSGI-Object component

Available overrides

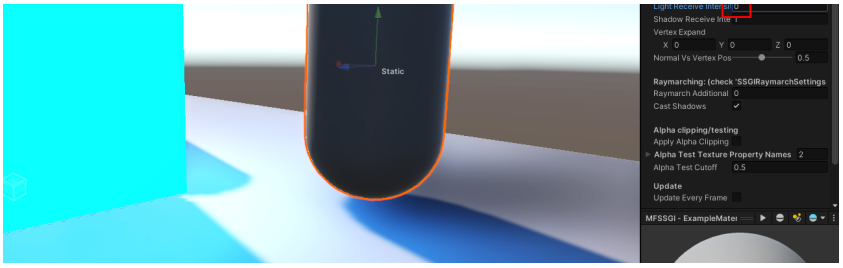
The SSGI-Object component allows for in-depth control of individual objects. Sometimes you don't want objects to cast/receive light, or you want them to be ignored by the shadow-pass completely. This is all possible thanks to this component, added a unique feature to MF.SSGI that others do not have.

Notes:

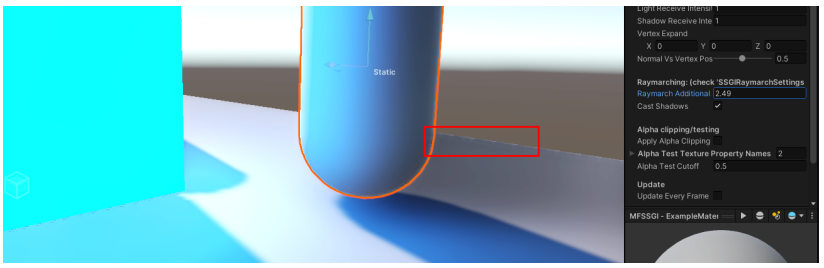
- Negatively scaled objects will be ignored as they cause graphic glitches.
- Change the 'Default Clip Depth Bias' over at the Advanced-settings when z-fighting issues occur. You can also override this per-component by changing 'Clip Depth Bias'



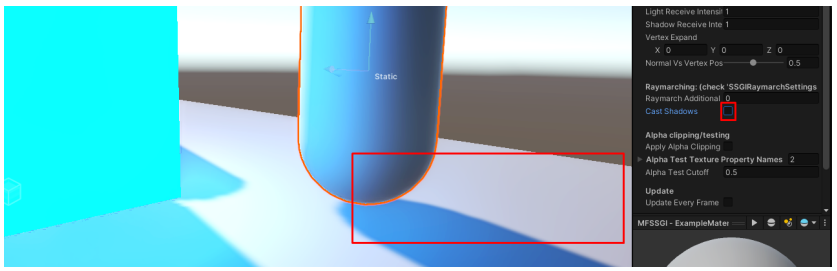
Default SSGI-component: Nothing changes



Light Receive: set the multiplier to zero to exclude,
or set above 1 to boost the light received

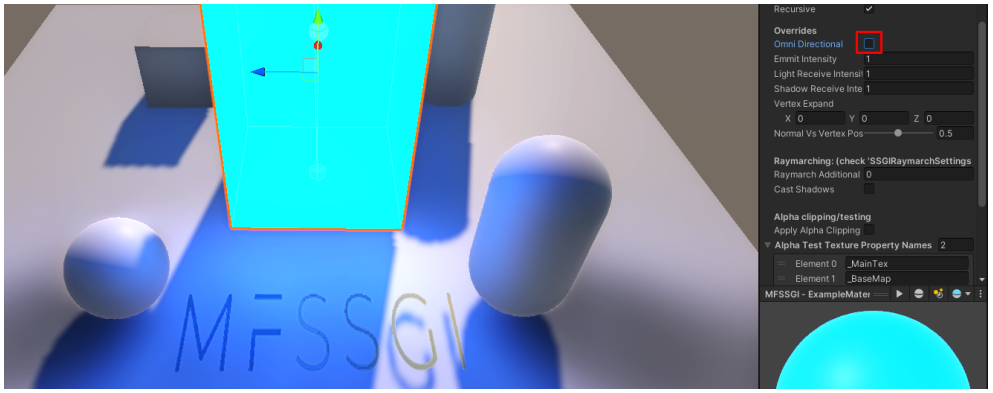


Raymarch additional thickness: Adds to the thickness-mask.
This can be useful on thin/single-sided objects

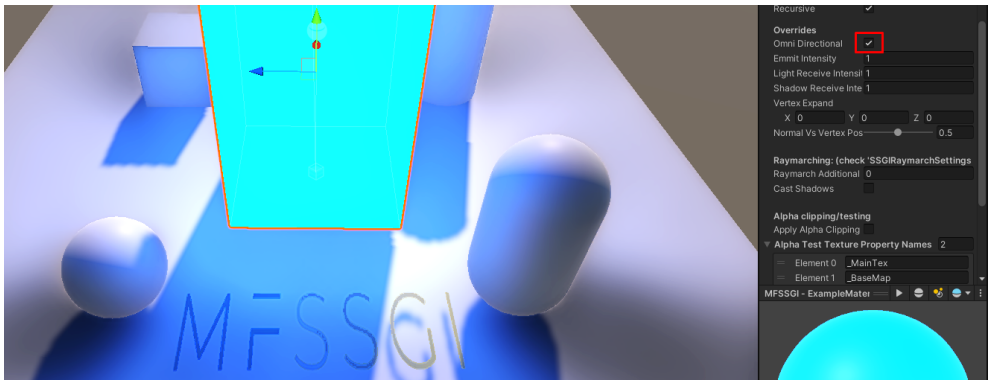


Cast shadows: Disabling this resolves unwanted shadows casted by this object

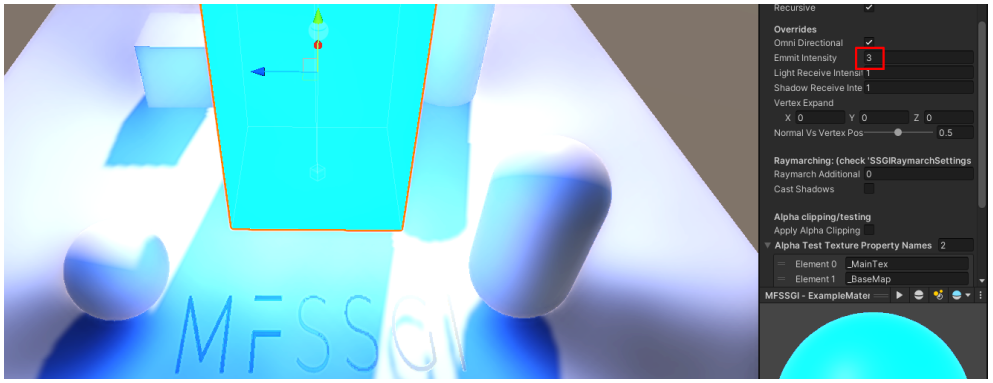
Emission:



Default SSGI-component: Nothing changes



Omni directional enabled: The surface-normal is now ignored and
the cube emits light on the objects in the background as well

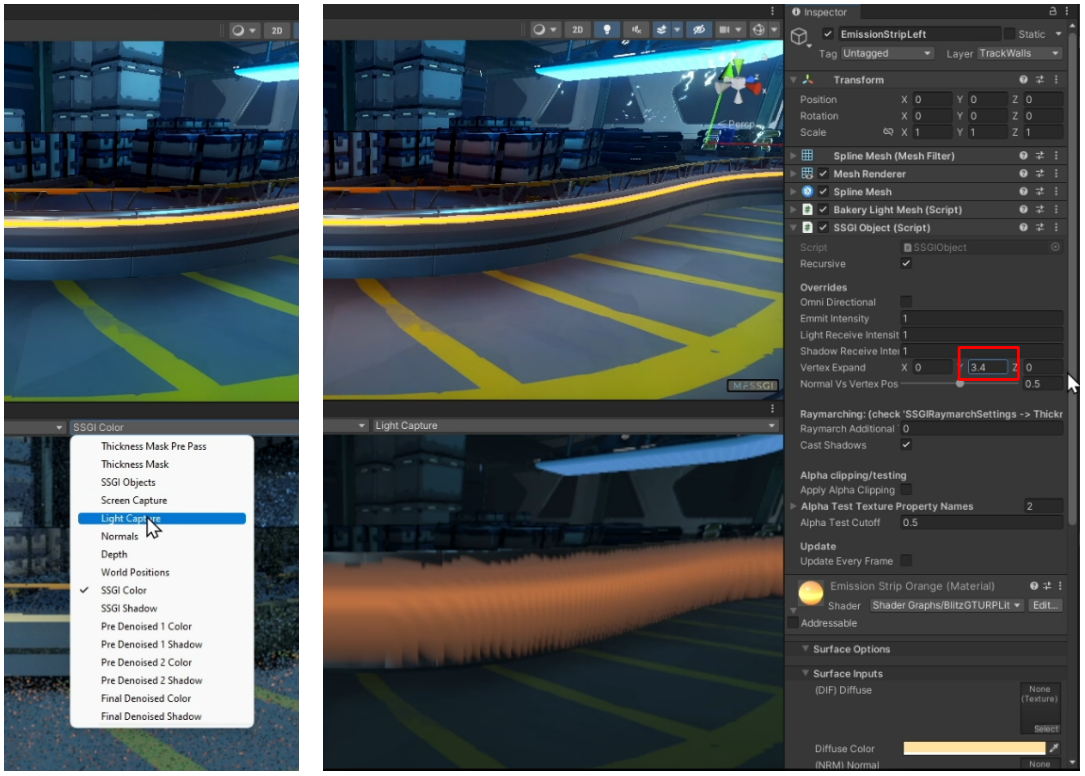


Emit intensity: set to 3

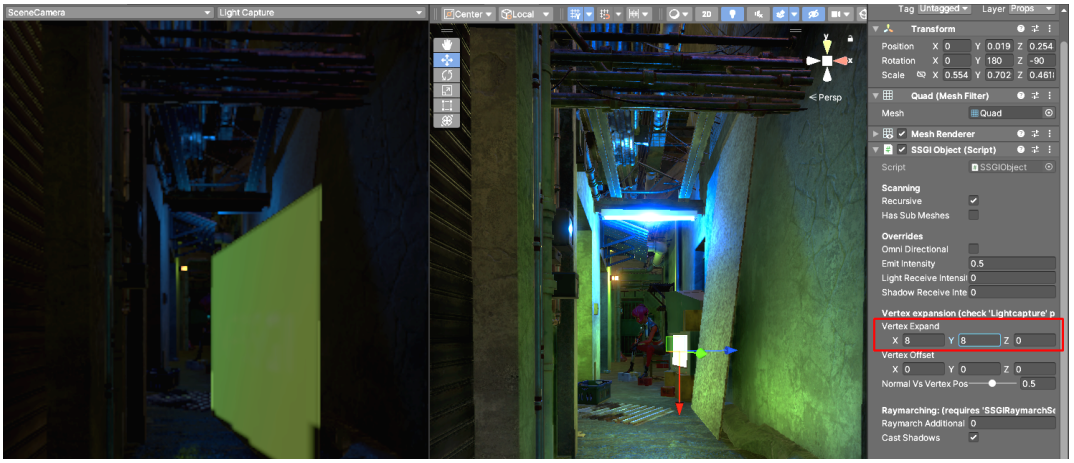
Lightstrips & Emissive objects

Lightstrips are amazing, but often very tiny and thin. This makes it hard for SSGI-samples to ‘find’ it in screenspace. In order to help the SSGI-feature to find it, we can make it virtually ‘bigger’.

The end-user won’t see this, but it helps a great deal with image stability and reduces flickering. To set it up, open the ‘Light Capture’ pass in the Debugger. Head over to the lightstrip, add an SSGIObject to it and head over to the ‘Expand vertices’ section.



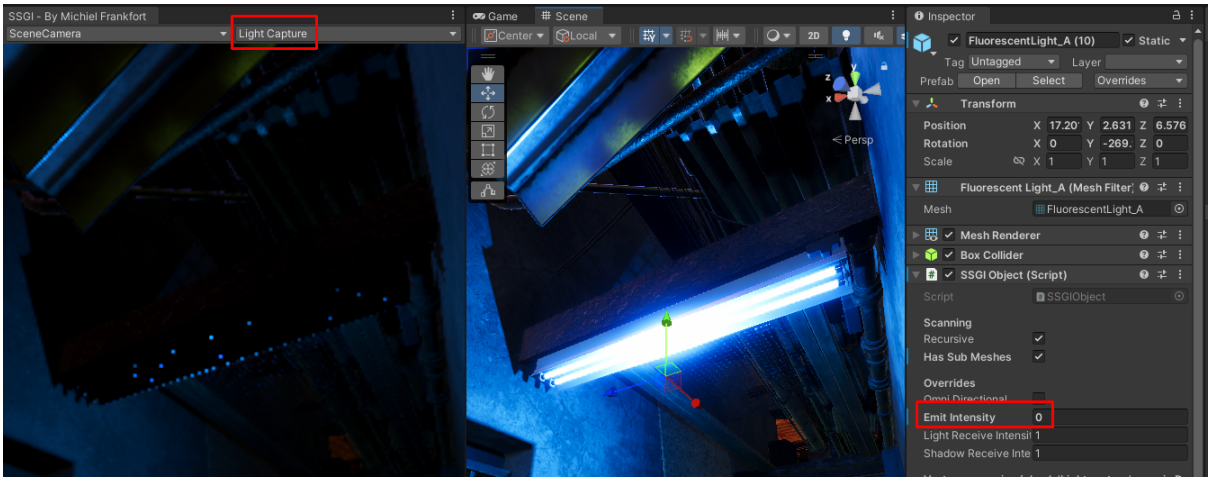
Another location where I used the same trick is in the Techdemo Alley. You might need to reduce the ‘Emit intensity’ to counter the additional light emitted by the grown object size.



Flickering and ‘Twinkles’

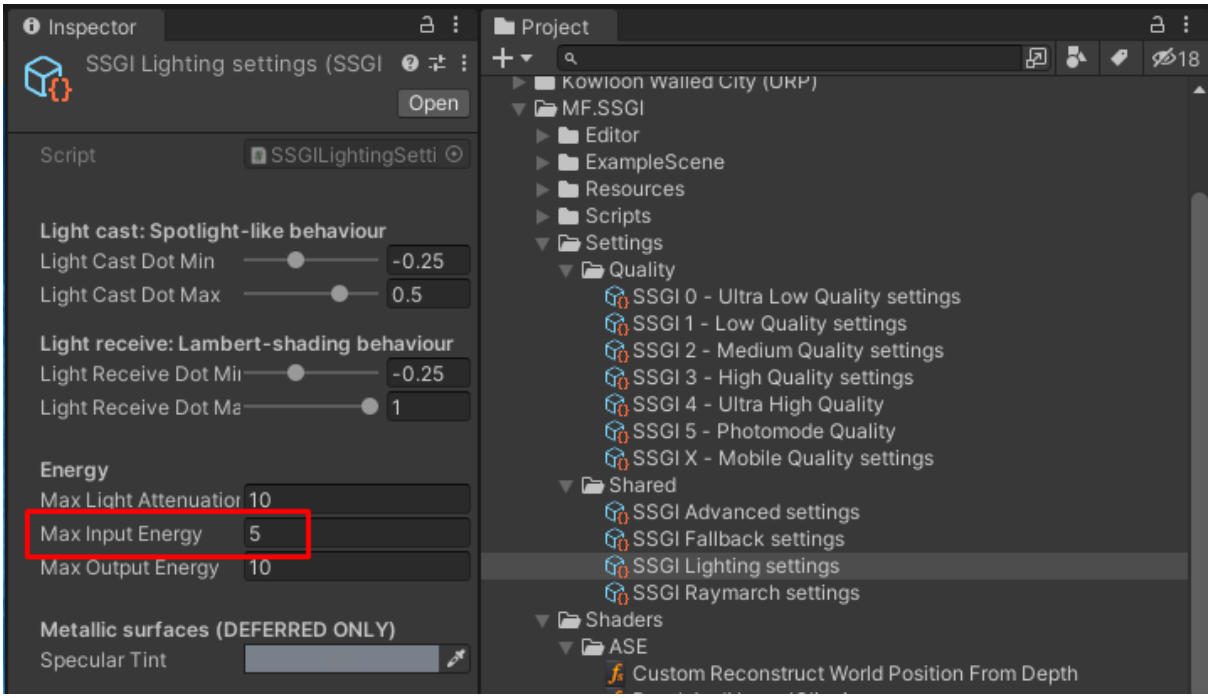
Very much related to the same issue as described above; the SSGI samples sometimes have a hard time ‘finding’ tiny but very bright objects. For example, in my Tech Demo I have these “Fluorescent lights” props that contain a point-light. Because the object itself is super bright, but often far away, I make sure it doesn’t emit any light itself.

This makes sure only indirect light bounces of the walls, not directly from the bright geometry of the prop.



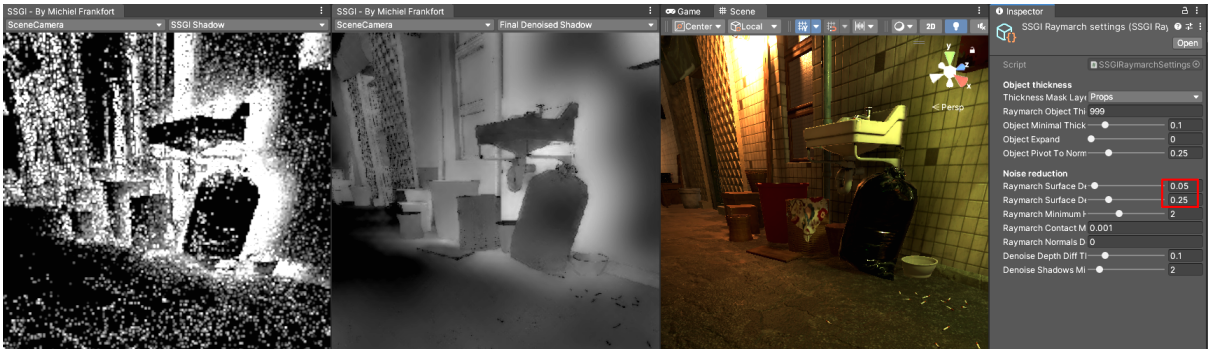
In the Alley the light suddenly becomes visible, casting a lot more light then intended and creating flickering because the mesh itself is so bright but quite small from afar. I solved this by simply setting the ‘Emit intensity’ to zero.

If this is not helping or you are looking for a different solution to counter ‘Twinkles’, you could also try lowering the ‘Max input energy’.

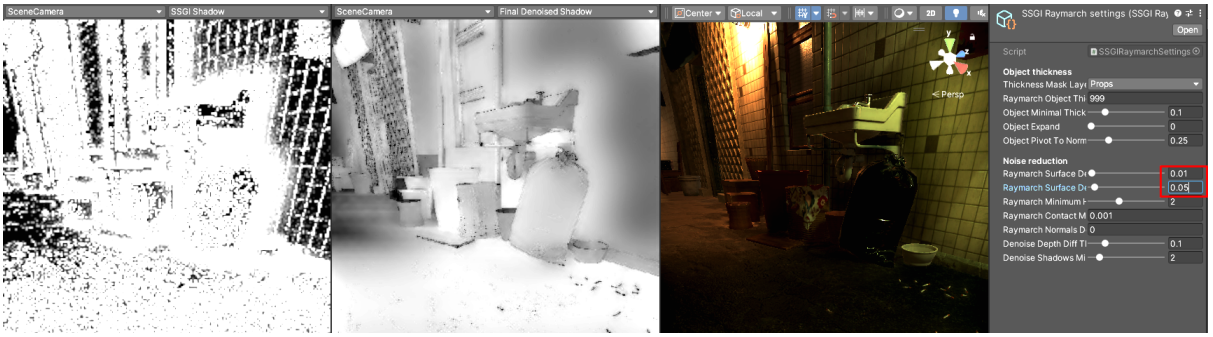


Shadow flickering

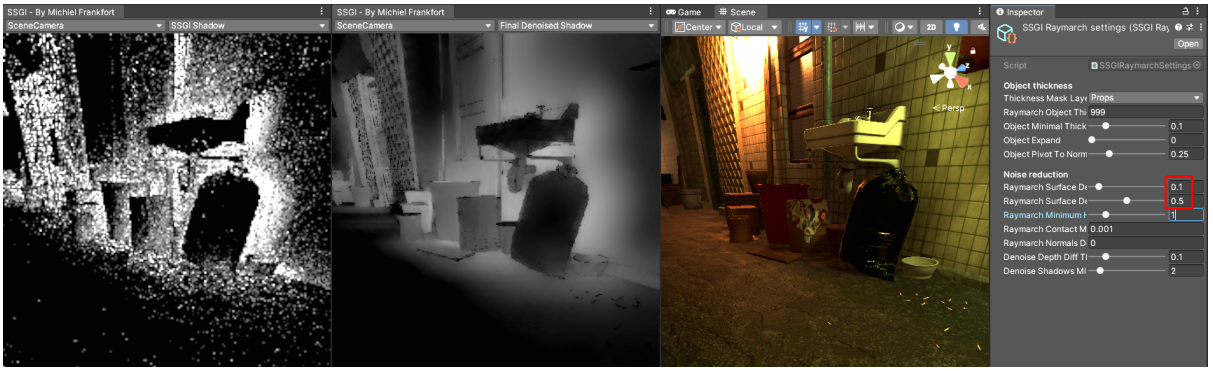
A lot of the flickering can be caused by shadows having a hard time staying ‘clean’. Individual sharp shadow-rays get denoised and blurred over a large area, creating flickering.



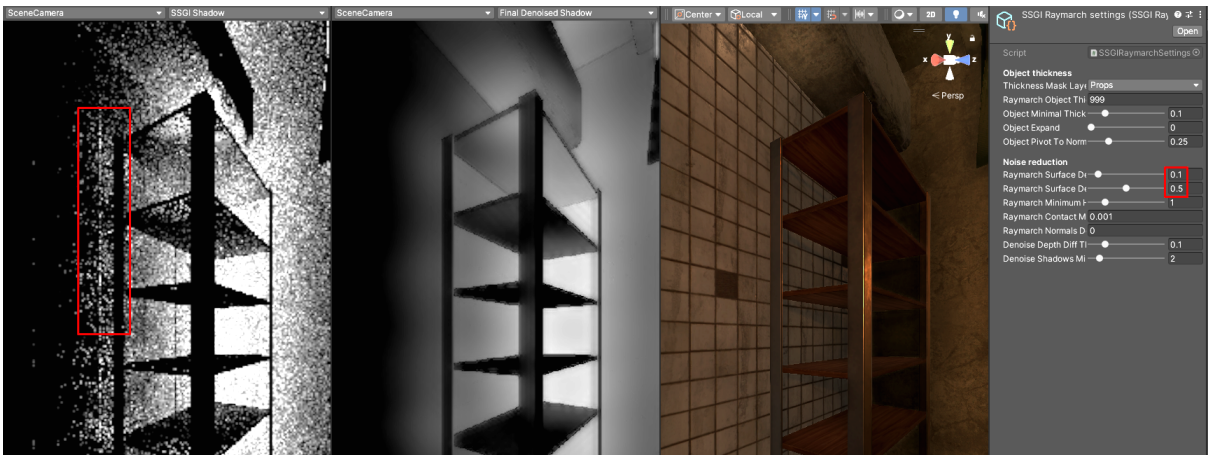
Default settings



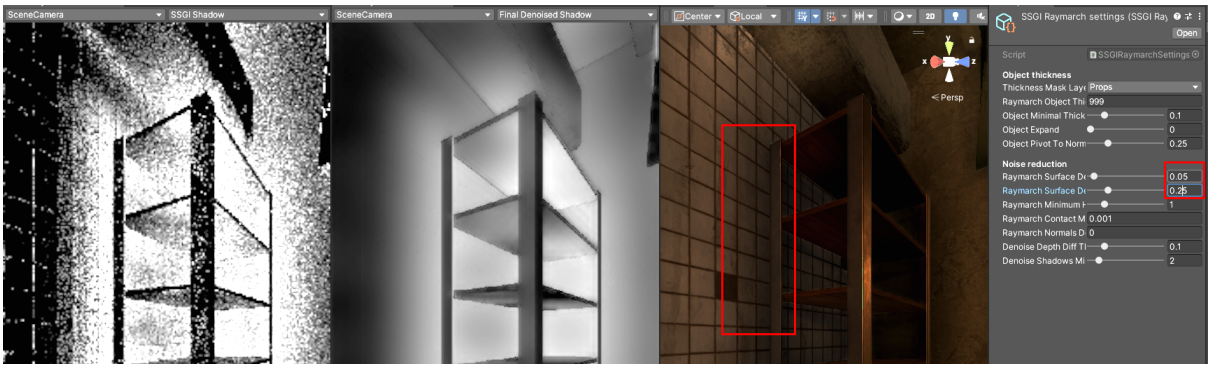
Extremely low settings: Works best with low-world-scale projects, where tiny details matter



Increased values cleans up the floor and tiny flickery details



However, these increased values makes it harder to find the thin shelf pillar



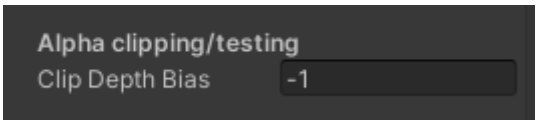
So finding a middle ground is key to your project

Alpha & Depth clipping

By default, thanks to some depth-testing-magic, Alpha clipped objects should be supported and SSGIObjects should nicely clip away if an object appears in front.

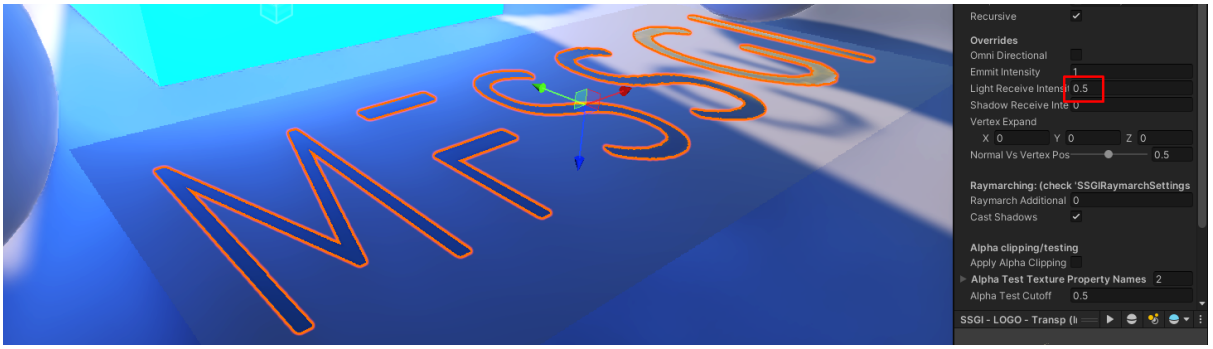
However, in case it doesn't work as intended, two options are available. First, in case of a non-alpha-clipped object, see if changing 'Clip Depth Bias' works. Values above 0 are used to test if the pixels should appear. The higher the value, the more leeway it has and fewer z-fighting artifacts will appear. However, when geometry is closely together, the SSGIObject will clip-through and be visible on unwanted objects. So please keep the value as low as possible.

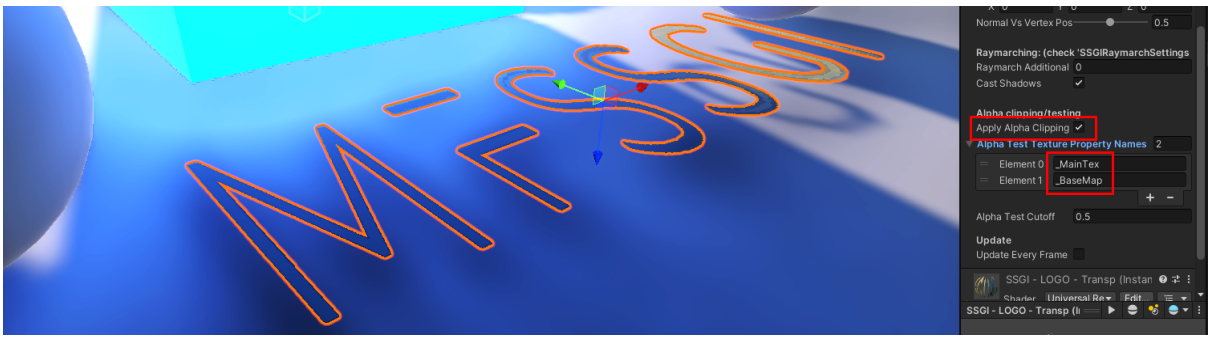
A value of -1 defaults to a clip-bias found over at the settings: "Advanced -> DefaultClipDepthBias"



If this solves the clipping issues for you, consider changing the global default setting instead of this individual object, as it will apply to all other objects in the scene as well.

In case of an alpha-clipped object, you can enable the 'Apply alpha clipping'-option. The materials will be scanned for textures assigned by the property names exposed (i.e. _MainTex). You can add more property-names to support custom shaders as well.

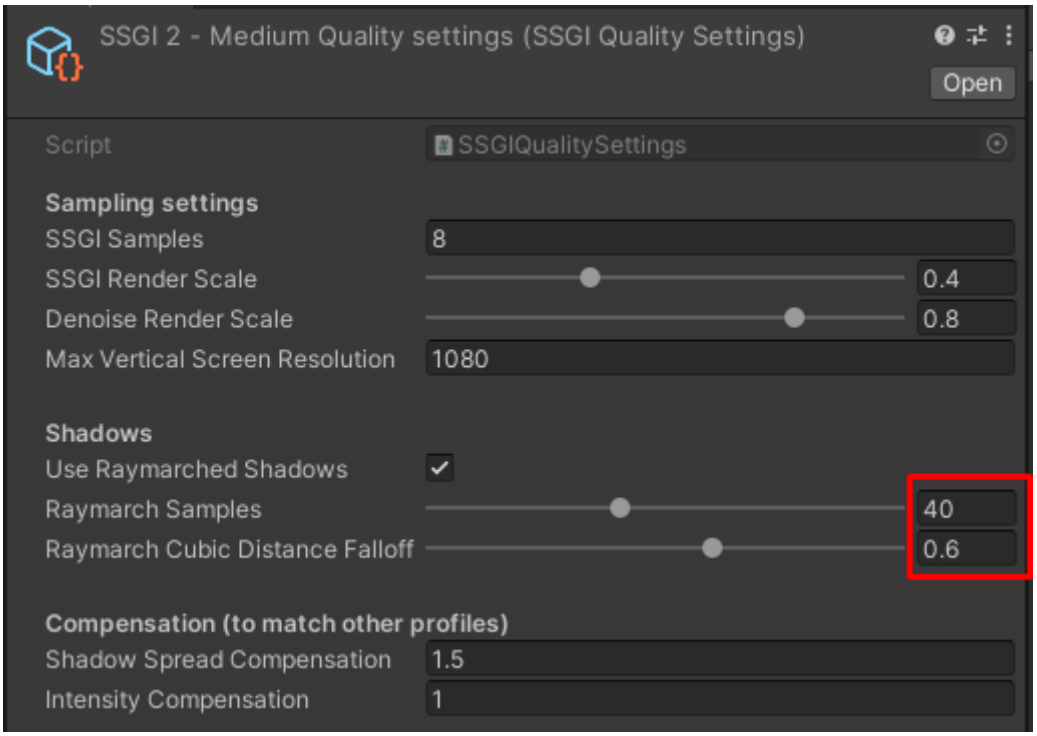




Advanced settings

Raymarching

A quick and basic summary of how this works: The way SSGI renders shadows is a ‘horizon-based’ approach where we can think of the depth-buffer as a 2.5D heightmap. So when a pixel on the left-side of the screen tries to light a pixel on the right-side, in 2D-space a ‘line’ is drawn. By comparing the depth-map foreach pixel on this line we can determine if the light was blocked. The amount of samples used to travel this ‘line’ can be found over at the SSGI Quality settings:



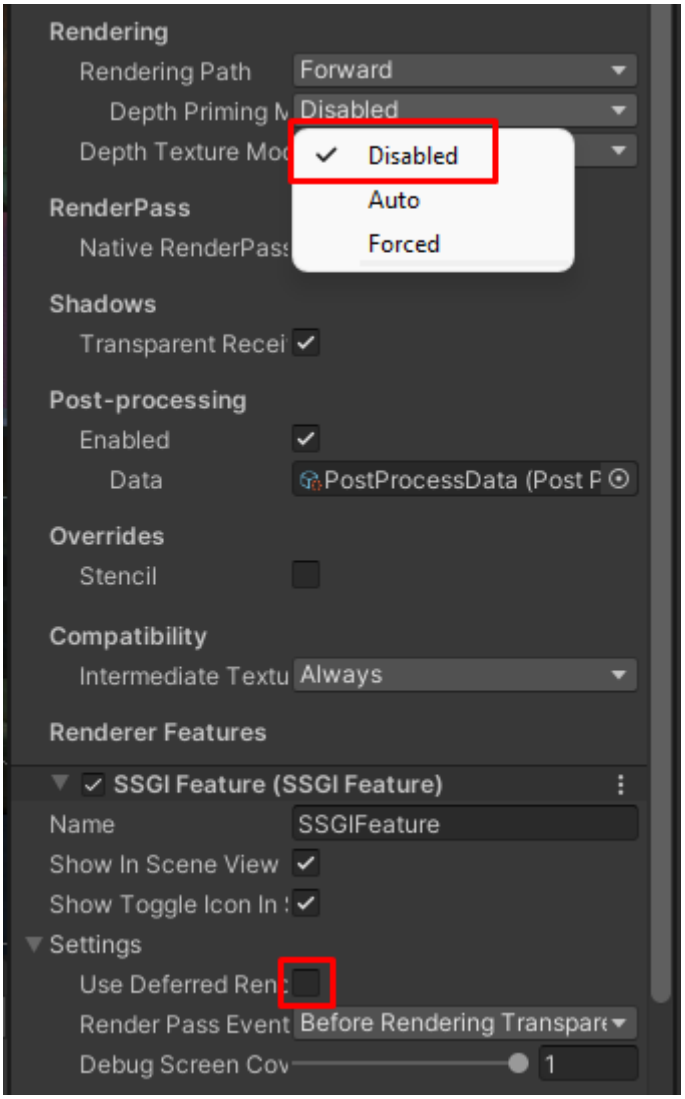
These samples are not evenly spread. As you can imagine a screen from left to right has far more pixels than 40, even at 0.4 renderscale. So we use the ‘Raymarch Cubic Distance’ to increase the samples near the target-pixel (the pixel receiving light) and decrease samples near the source-pixel (the pixel emitting light). The slider will blend between linear and cubic in favor of the target-pixel.

The tradeoff here is:
High cubic falloff: Sharper contact-shadows and more accurate nearby casted-shadows
Low cubic falloff: Longer stretchy casted-shadows, but less resolution available for contact-shadows.

Mobile & WebGL build settings

In case the SSGI is not visible, make sure to check the following settings:

- Depth Priming is a feature that doesn't work well in WebGL
- Mobile devices & WebGL fallback to Forward-rendering, so make sure to disable Deferred if enabled



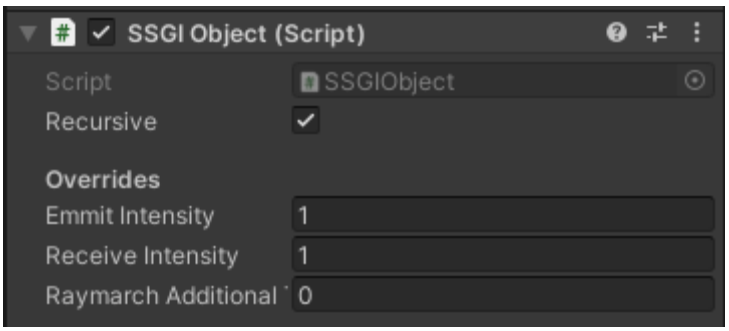
Tips & Tricks

Baked GI + SSGI

If your scene already has GI baked, in many cases full SSGI is just overkill and doesn't make a big difference to the environment. It is however a fantastic alternative to Light-probes and allows for animated shaders to emit light.

This is why SSGI is also used in BlitzGT, a fast-paced racegame where Emissive animated billboards and emissive strips on the floor emit light on the cars correctly. Because MF.SSGI is rendered in a single-frame, fast moving and/or changing emissive objects work just fine. Some other solutions use re-projection, which could yield higher quality, but has a lot of restraints as multiple frames are used to compile a single light solution. MF.SSGI does not need this, therefore you are not limited.

The way to set this up is to make use of the SSGIObject-component. By default mix your Lighting-settings in such a way that the original already baked GI is complimented and not overpowered. Once that looks good, head over to the scenery objects that you want to emit light (like billboards, large dynamic emissive objects like burning buildings, etc) and add a SSGIObject-component. Increase the **"Emit Intensity"** to a level that makes the scene look good.



Once the environment looks good, focus on dynamic objects and see if these need boosting. It could pay-off to fully disable Lightprobes and use SSGI instead. In that case it is advised to add SSGIObject-components to the dynamic-objects and boost the “**Receive Intensity**”