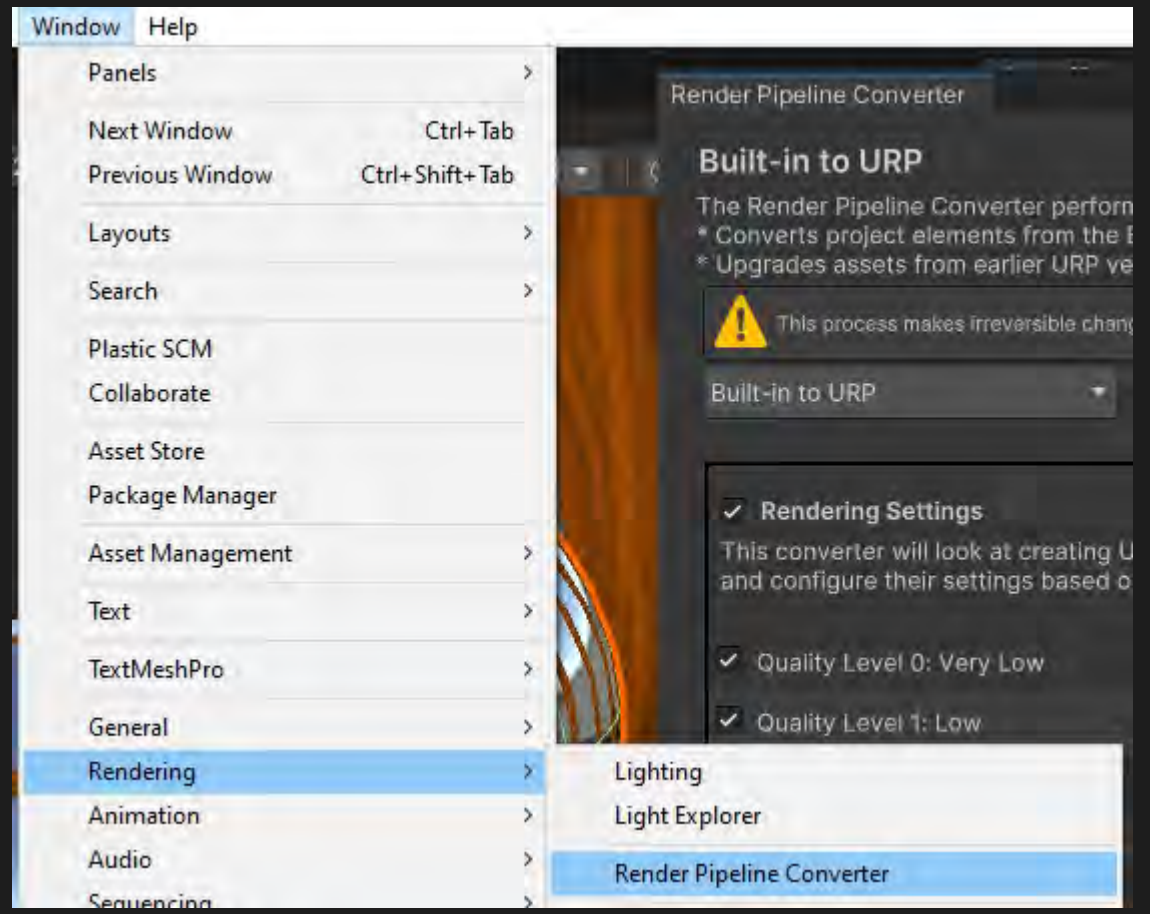
Lighting

Unity 2021

- Follow using Unity 2021.2+
- Install Probuilder with URP support

- To update BIRP material to URP material: Window/Rendering/RenderPipelineConverter
- Check also ReadonlyMaterialConverter to be able to convert also old legacy readonly Standard shader

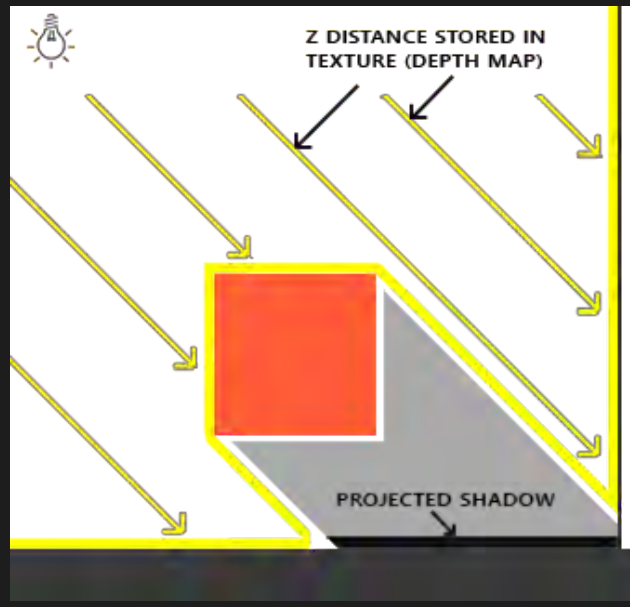


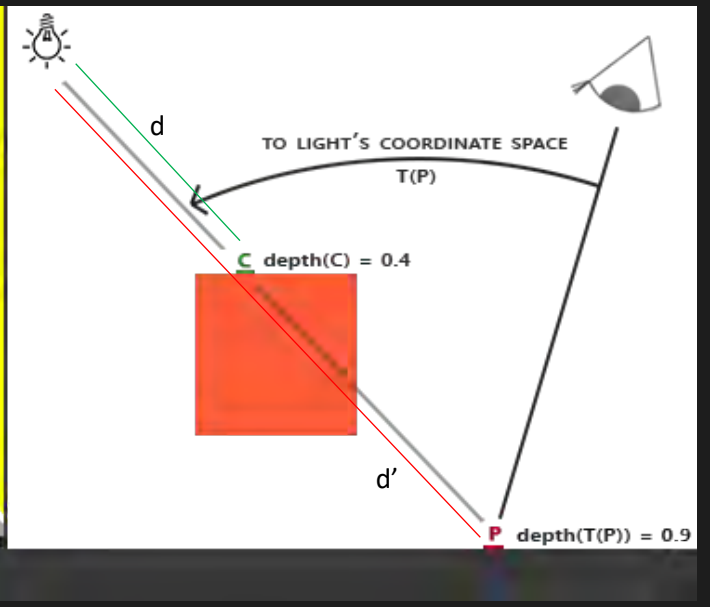
Shadow mapping

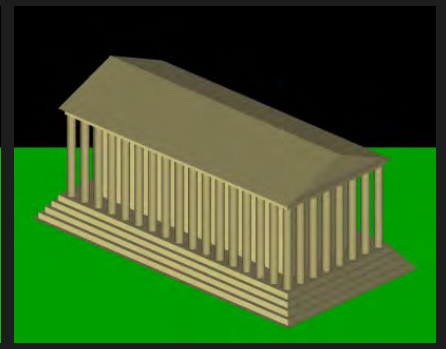
1. Create the depth map by rendering the scene (from the light's perspective) using a view and projection matrix specific to that light source
2. T Matrix transforms any 3D position to the light's visible coordinate space
3. A directional light doesn't have a pos (it's considered to be infinitely far away). However, for the sake of shadow mapping we need to render the scene from a light's perspective
    - Dir light camera is orthographic, and Unity renders the scene from a position somewhere along the lines of the light direction, where all the scene objs seen from the standard render camera are visible

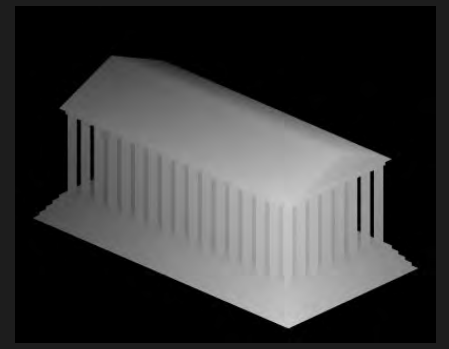
Render a fragment at point P for which we have to determine whether it is in shadow

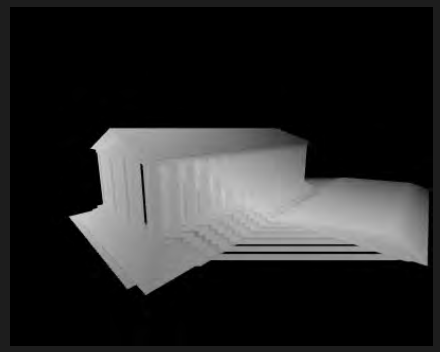
1. To do this, we first transform point to the light's coordinate space using T
2. Since point P is now as seen from the light's perspective, its z coordinate corresponds to its depth which in this example is 0.9
3. Using point P we can also index the depth map to obtain the closest visible depth from the light's perspective which is at point with a sampled depth of 0.4
4. Since indexing the depth map returned a depth smaller than the depth at point we can conclude point is occluded and thus in shadow

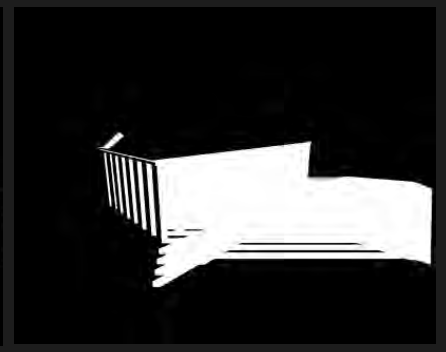


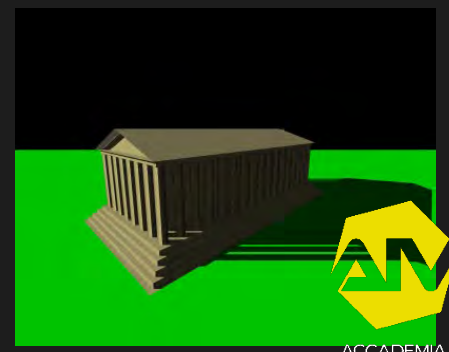








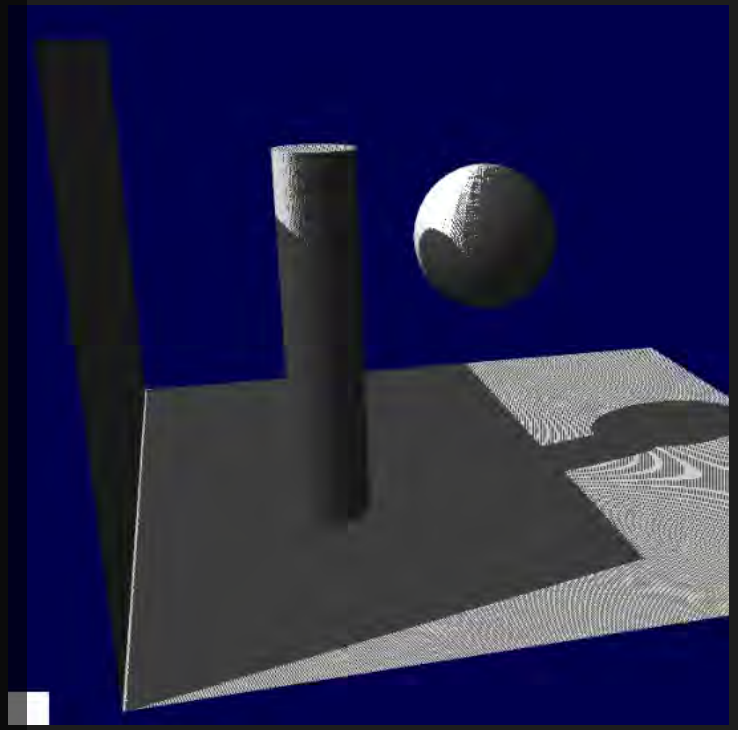


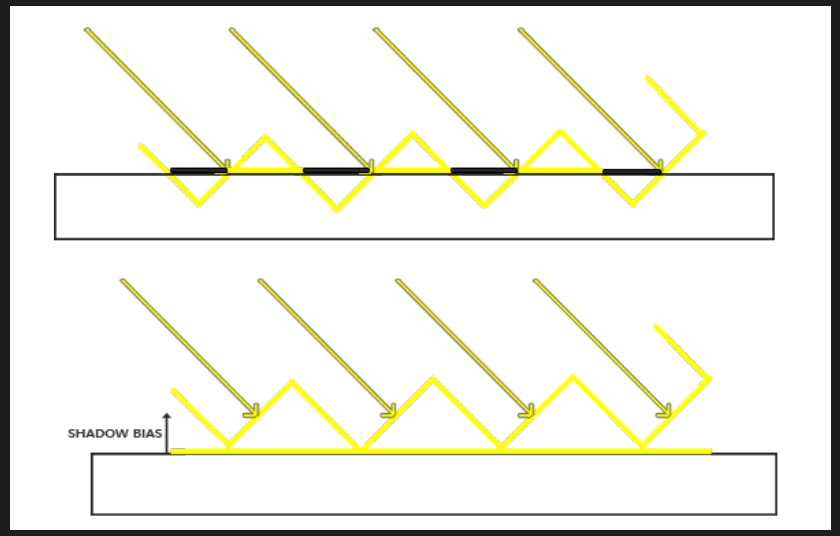


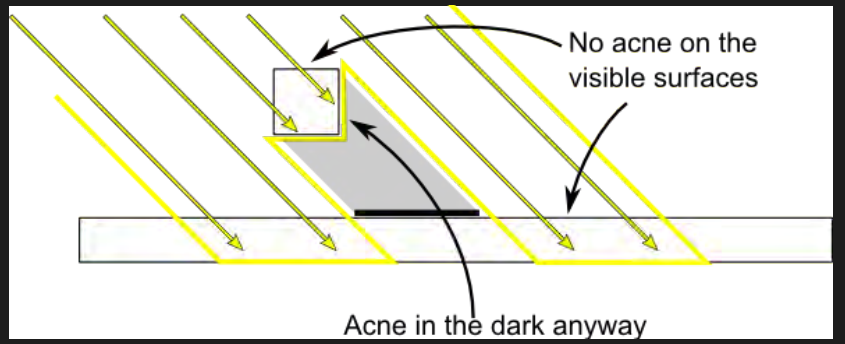
Shadow Acne

- The usual fix is to add an error margin: we only shade (put the fragment in shadow) if the current fragment's depth d (in light space) is far away from the lightmap value ld, plus a depth value called shadow bias
  - w/o s. bias: shade if d > ld
  - w s. bias: shade if d + bias > ld
- Another trick is to render only the back faces in the shadow map. This forces us to have a special geometry (see Peter Panning) with thick walls, but at least, the acne will be on surfaces which are in the shadow

[Shadows_01]



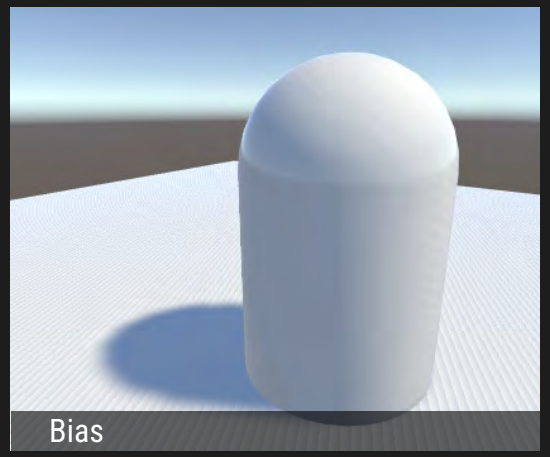


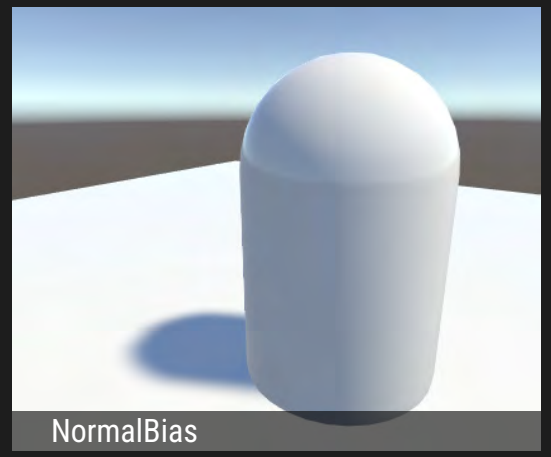


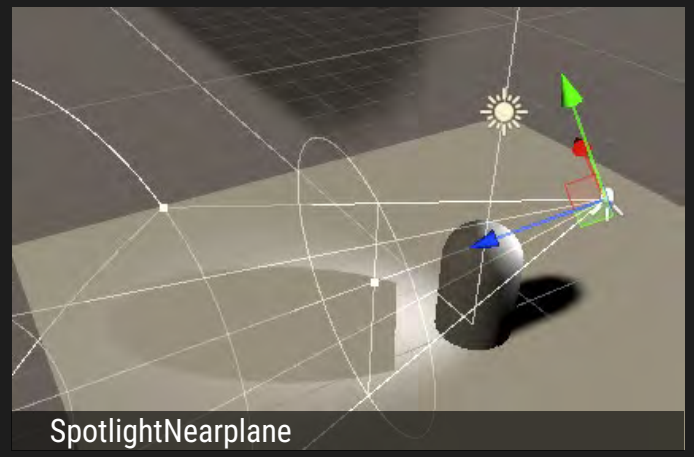
Shadow Bias

- URPAsset/DepthBias Reduces the Shadow Acne Problem
  - Don't be too much precise with this value: take into account the player perspective: if the game has a camera tha is distant 10 from the ground, tweak this value from that distance, not closer
- URPAsset/NormalBias Shadow caster surfaces are pushed inwards along their normals by this amount, to help prevent self-shadowing artifacts. Typically values between 0.3-0.7 work well.
- Light/NearPlane GameObjects closer than this distance to the light do not cast any shadows

[ShadowBias_01]

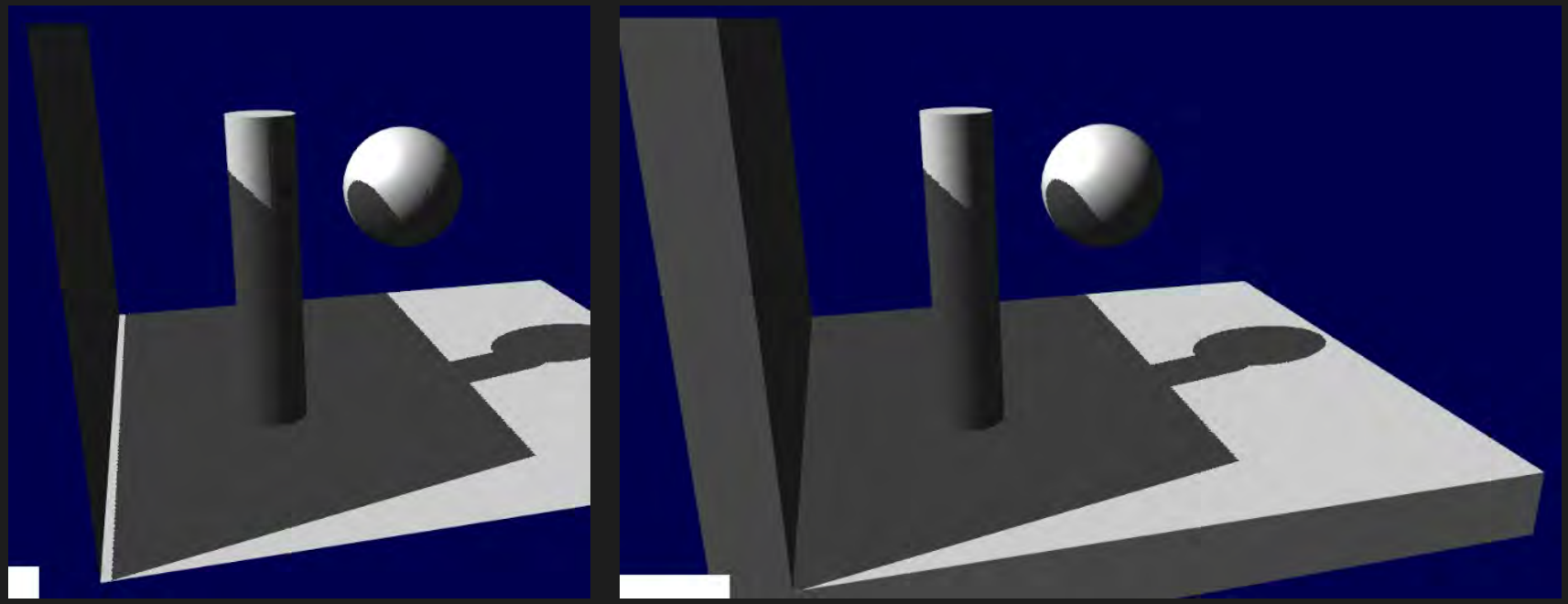

Bias


NormalBias


SpotlightNearplane

Peter Panning

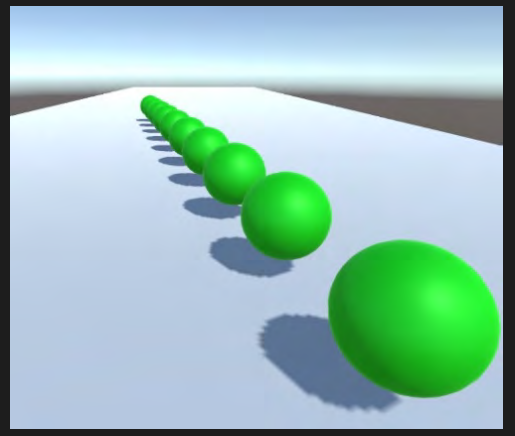
- Often fix shadow acne leads to this kind of problem: wrong shading of the ground, making the wall to look as if it's flying ("Peter Panning")
- Easy to fix : simply avoid thin geometry
  - It solves Peter Panning : if the geometry is more deep than your bias, you're all set
  - You can turn on backface culling when rendering the lightmap, because now, there is a polygon of the wall which is facing the light, which will occlude the other side, which wouldn't be rendered with backface culling

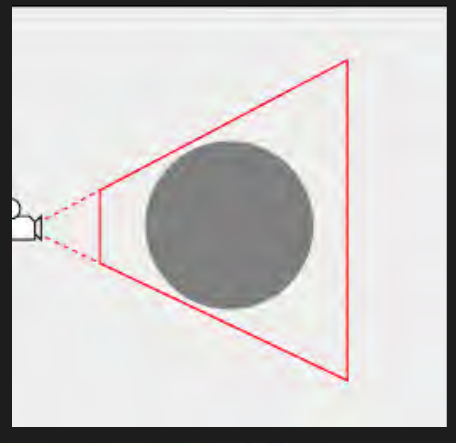


PCF (Percentage Close Filtering)

- When you sample the shadowmap once, Unity will in fact also sample the neighboring texels, do the comparison for all of them, and return a float in [0,1] with a bilinear filtering of the comparison results
  - 0.5 means that 2 samples are in the shadow, and 2 samples are in the light
- NB: PCF doesn't perform a single sampling of a filtered depth map. A comparison always returns true or false; PCF gives a interpolation of 4 "true or false"
- Result: shadow borders are smoother

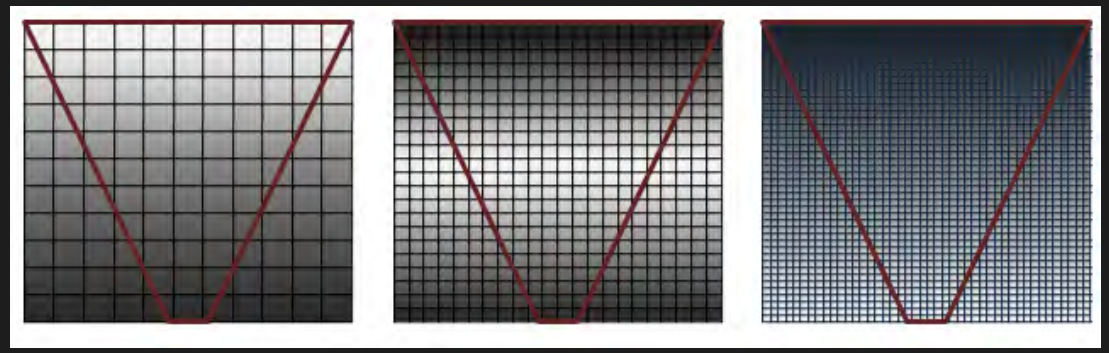
CSM (Cascade Shadow Maps)





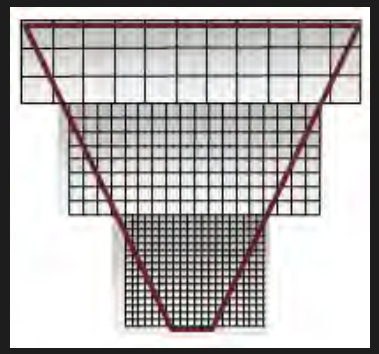
Cascade Shadow Maps

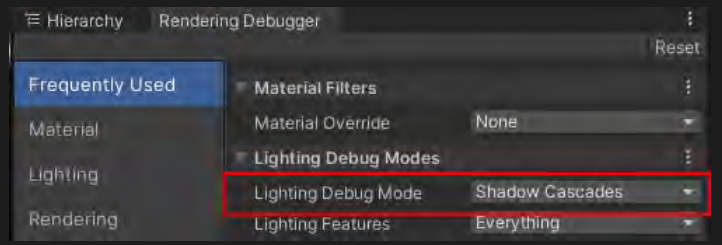
- SM pixels seen from the camera look enlarged if close to the camera, compared to those farther away: Shadows are of the highest quality (white pixels in the imgs) when there is a 1:1 ratio mapping pixels in light space to texels in the shadow map
- Use a higher resolution for the whole map reduce the problem, but uses more memory
- The zone near the camera can use a separate shadow map with higher resolution – CSM
    - Render the scene from the light POV multiple times is better than using a very high resolution for the SM texture

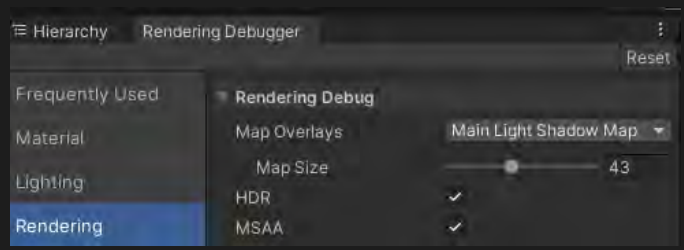


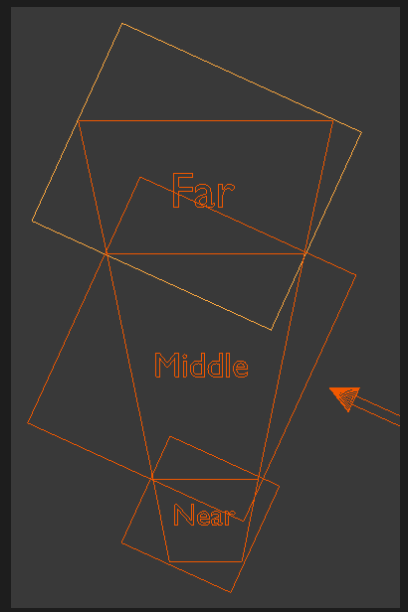
Good Rule

- Keep ShadowDistance as low as possible
    - Shadows that are far away often don't increase image quality
    - Add Fog to the scene if necessary





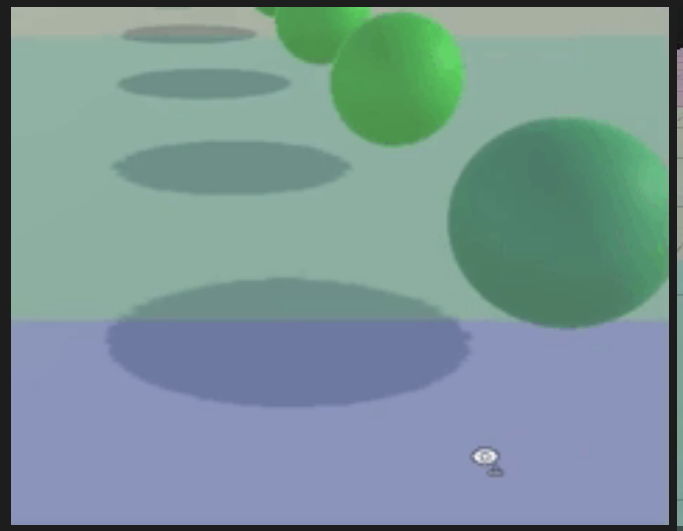


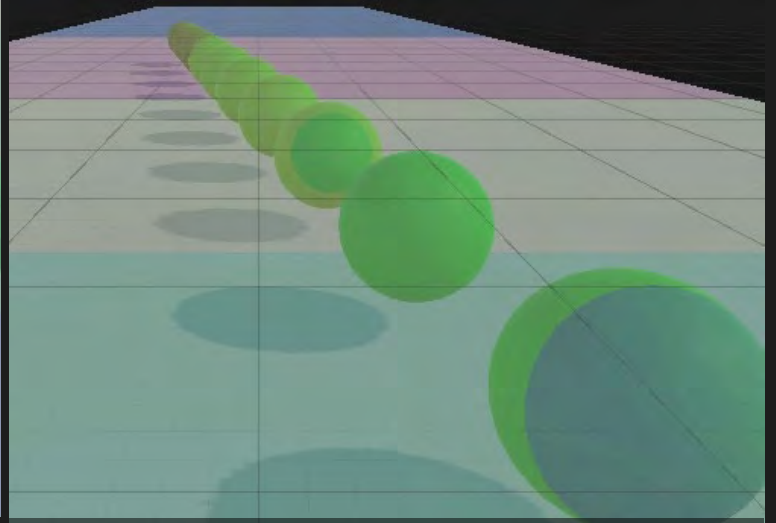


[BIRP] CSM

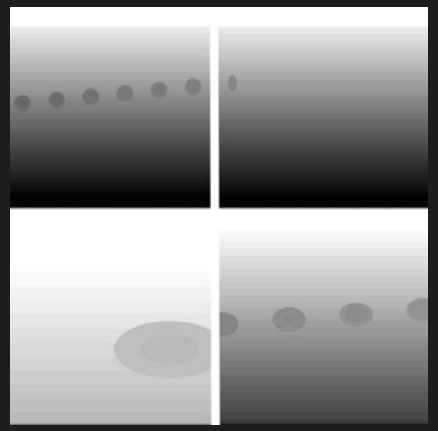
- Cascade splits percentages are referred to ShadowDistance value

ProjectSettings/Quality
- ShadowmaskMode – We'll see this later
- ShadowProjection (To show them, disable Cascades, use hard shadows and set ShadowResolution Low)
  - CloseFit If you rotate cam along Y, you'll see changing shadows borders
  - [URP in research] StableFit If you rotate cam along Y, you'll see stable shadows borders
- If CSM is used:
  - CloseFit Choose CSM band based on Cam depth buffer
  - [URP in research] StableFit Choose CSM band based on Cam pos distance

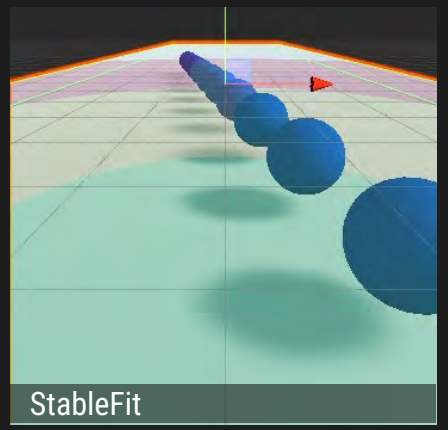




SceneView / Miscellaneous / ShadowCascades View (CloseFit)



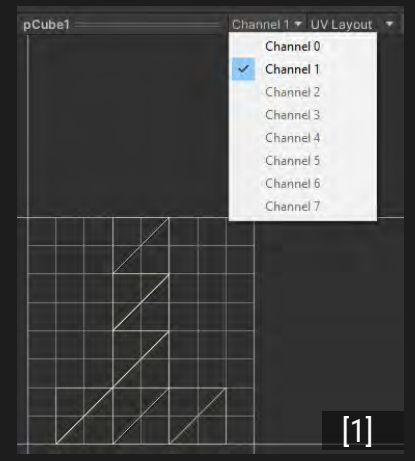
Cascade splits percentages are referred to ShadowDistance value

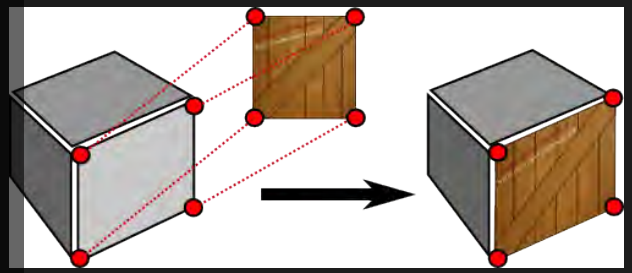


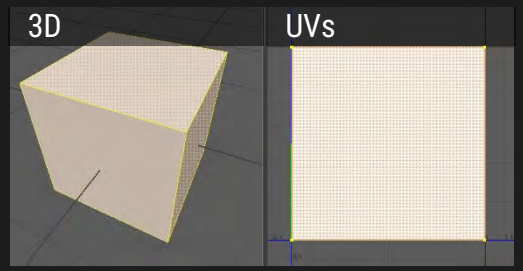
StableFit

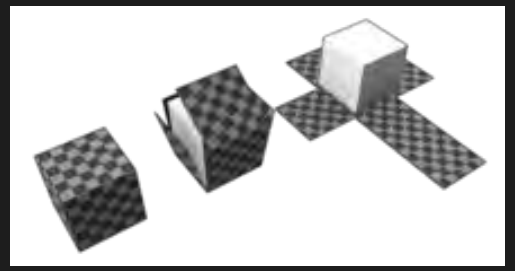
Lightmap UVs

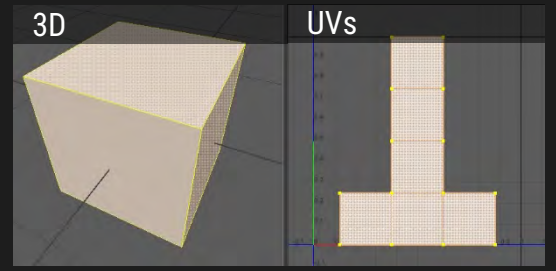
- A Lightmap is a texture that contains light/shadow info and is wrapped around our obj in the same way as Albedo Textures
- BUT Albedo Texture UVs usually use optimized projection
- What happens if we use the same UVs set?
- We need 2 UVs sets: one for Texturing, one for Lightmapping
- You can inspect mesh UVLayout in import settings (You must select the mesh obj INSIDE the FBX prefab) [1]

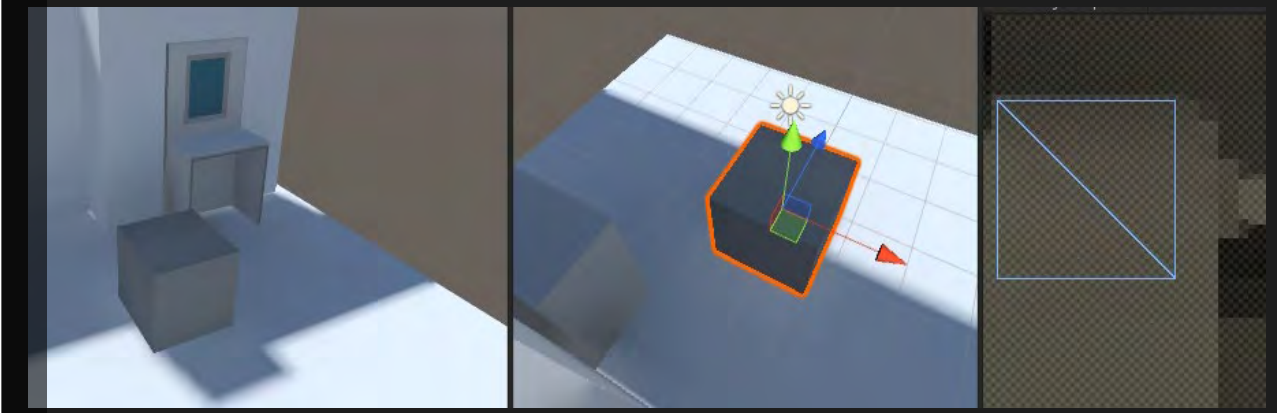


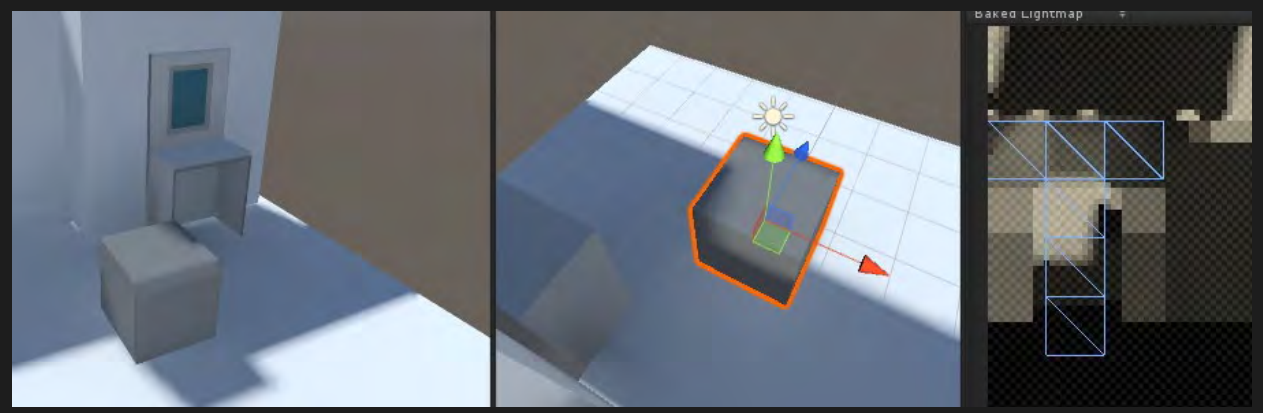




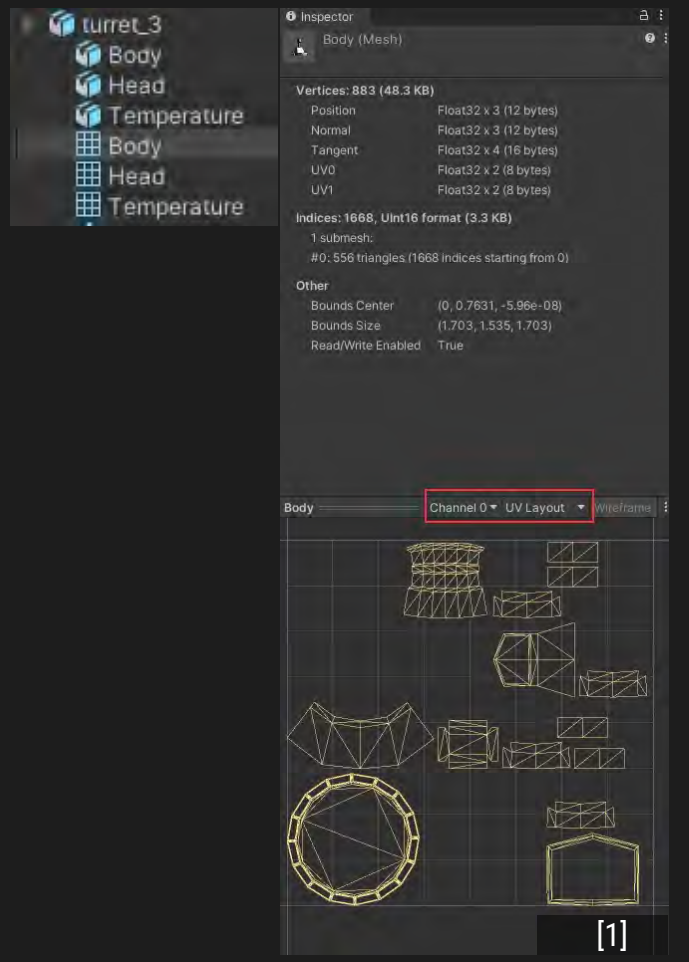








UVs channels



[1]

Unity supports up to 4 UVs channels

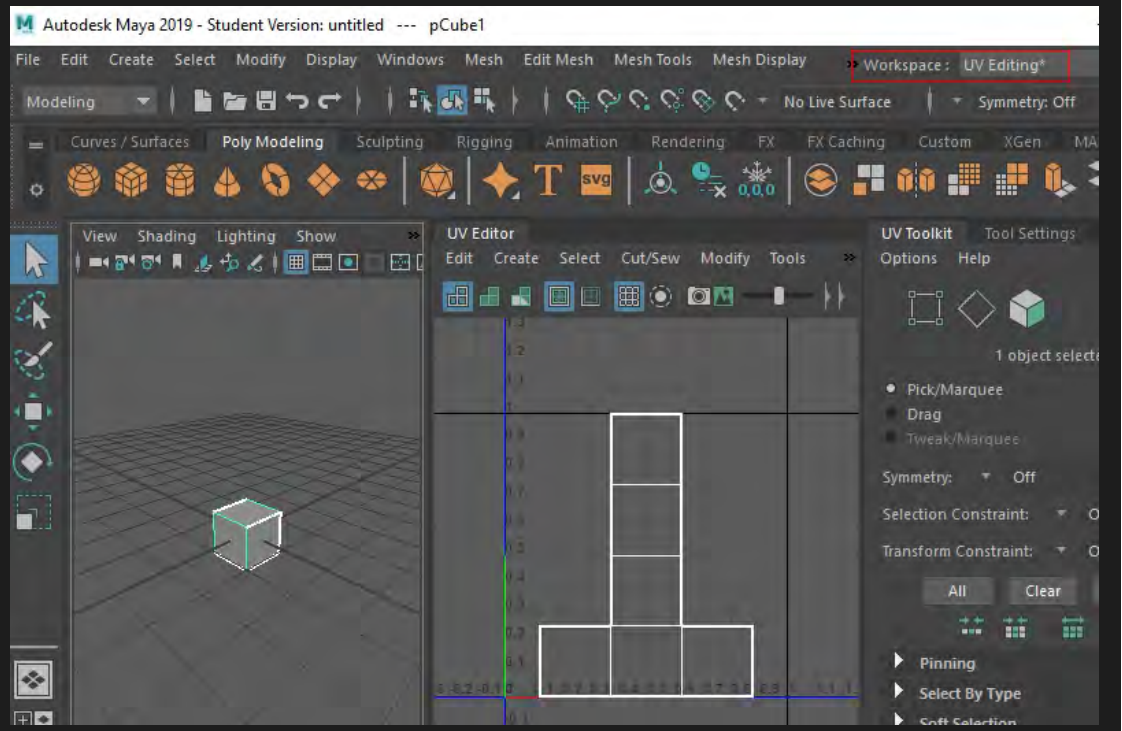
- Texturing UVSets
- Baked GI (Mixed lighting) 3DAsset Import settings/GenerateLightMapUVs
  - Direct lighting
  - Indirect lighting
  - AO
- Real-time GI Mesh Renderer Attribute/OptimizeRealtimeUvs
  - Indirect lighting only (direct light is rendered in realtime)
  - Low resolution
- Other

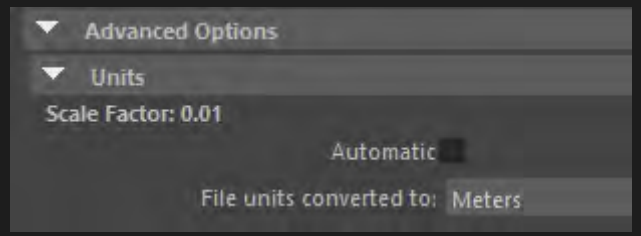
You can inspect 3D Mesh UVLayout channels selecting the mesh inside the imported FBX [1]

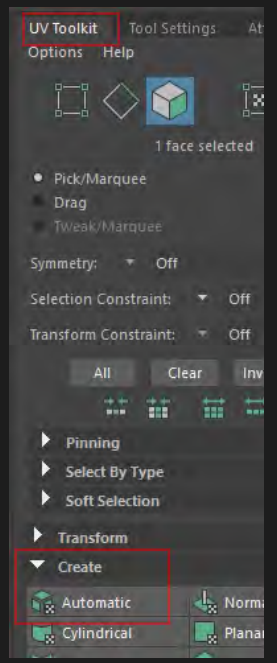
Ordinal	Mesh class property	Shader Code	Used for
First	mesh.uv	UV0	Diffuse, Metal/Spec etc.
Second	mesh.uv2 (& old .uv1)	UV1	Baked lightmap
Third	mesh.uv3	UV2	Realtime GI data
Fourth	mesh.uv4	UV3	Whatever you like

Baked Lightmap UVs

- Create a cube Cube_TFoldUV.fbx
- Workspace/UV Editing
- Leave the UV unwrap to standard T-Fold
- Export FBX / AdvancedOptions / Units: Meters
- Create another cube Cube_SameFacesUV.fbx with all faces mapped on the same UV space
  - RClick/Faces
  - Select each face (alt+drag to rotate cube) and choose UVToolkit/Create/Automatic







Baked Lightmap UVs

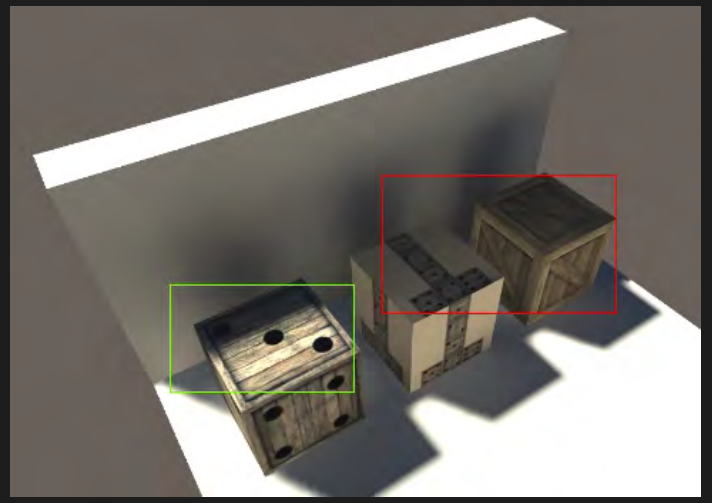
- Apply woodBox_SingleFace material to Cube_SameFacesUV
- Apply woodBox_TFold material to Cube_SameFacesUV
- Apply woodBox_TFold material to Cube_TFoldUV
- Set DirectionalLight to Realtime and move the Wall
- Set Lighting to
  - RTGlobalIllumination OFF
  - MixedLighting ON
  - Lightmap Resolution 10
  - Lightmapper Progressive CPU
  - DirectionalLight Baked
  - All the objs to Static
- Press GenerateLighting and move the Wall / rotate DirectionalLight
  - Light and shadows are now baked on the objs: they remain attached to the objs even if we move the wall!

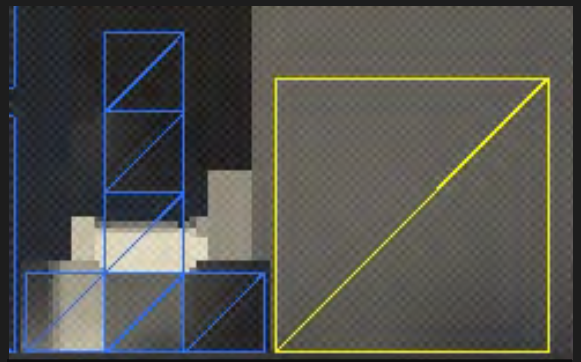
[LightingOnCubes_00]

Baked Lightmap UVs

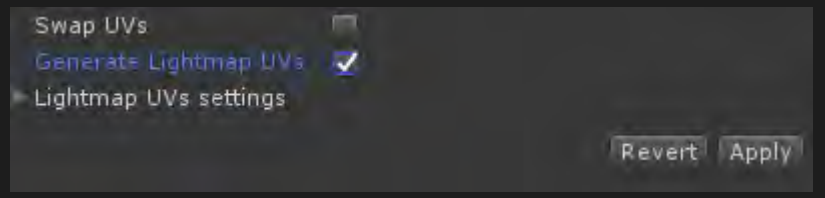
- Cube_SameFacesUV doesn't have UV1 proper coordinates > Cubes have wrong lightmap
  - Select the wrong cubes and click on preview lightmap to see their final UV coordinates on the lightmap
  - What's happening? UV0 is copied into UV1, and UV0 coordinates are not good for lightmapping
- You can delegate to Unity the generation of UV1 coordinates
  - Cube prefab / ImportSettings / GenerateLightmapUVs
    - Not always a good result
    - UVMap set 0 remains the same! If you apply a Crane material, every face will have the same texture!

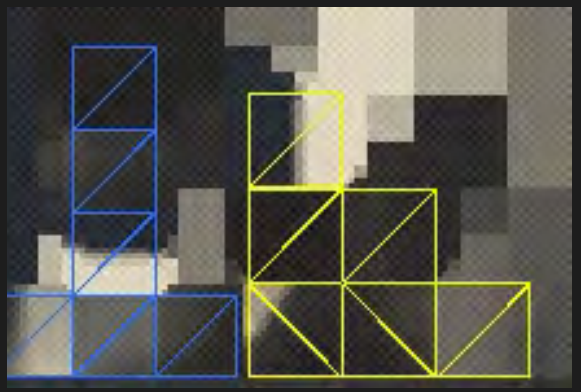
[LightingOnCubes_00]





Cube_TFoldUV (left) and Cube_SameFacesUV (right) uv coordinates inside the lightmap

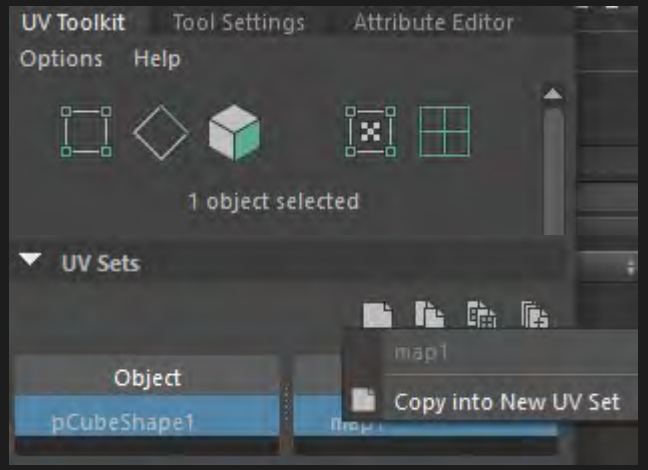


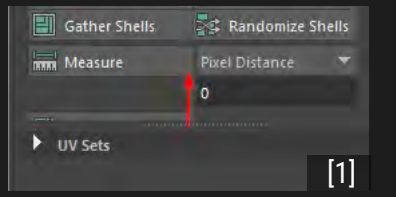


Baked Lightmap UVs

- Cube_SameFacesUV doesn't have UV1 proper coordinates >
- What if you want a UV0 like Cube_SameFacesUV to optimize textures inside your project, but you want to provide correct UV1 for lightmapping?
  - Show UVToolkit/UVSets
  - NB: UVToolkit/UVSets C(NB: If you don't see this panel, could be at the bottom of the UVToolkit column) [1]
  - Rclick on map1UVSet/Duplicate
  - Set map1UVSet as SameFace UVSet (select each faces/Create/Automatic)
- Export Cube_SameFaceUV0_TFoldUV1.fbx, import into your scene and mark it as Static
- Apply Woodbox_Tfold to it, to be sure we are using the same face for albedo texture (UV0)
- GenerateLighting > now everything works, because the cube has proper UV1 coordinates!

[LightingOnCubes_00]





[1]

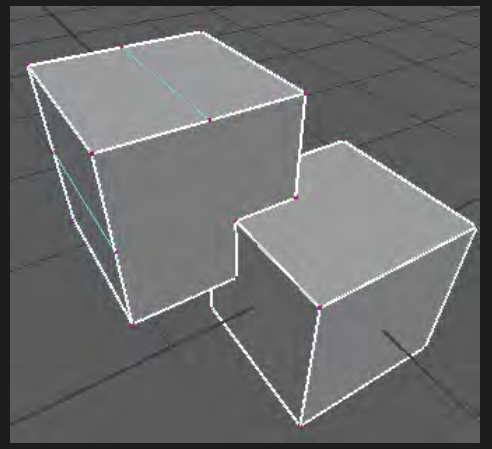
Cube_TFoldUV (left) UV0 Tfold, UV1 Copied from UV0 by Unity
Cube_SameFacesUV (center) UV0 sameFace, UV1 Autogenerated from import settings
Cube_SameFaceUV0_TFoldUV1 (right) UV0 sameFace, UV1 TFold

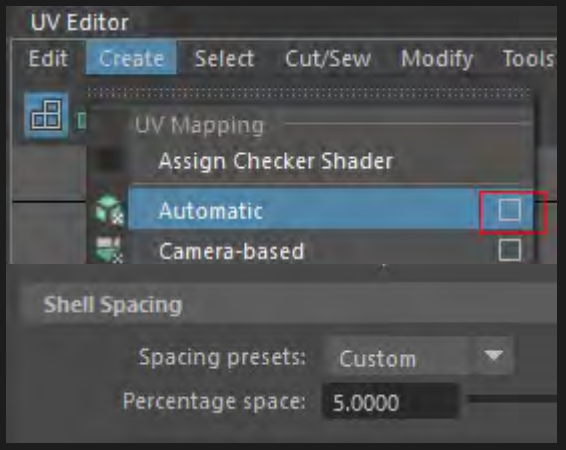
Baked Lightmap resolution and UVs artefacts

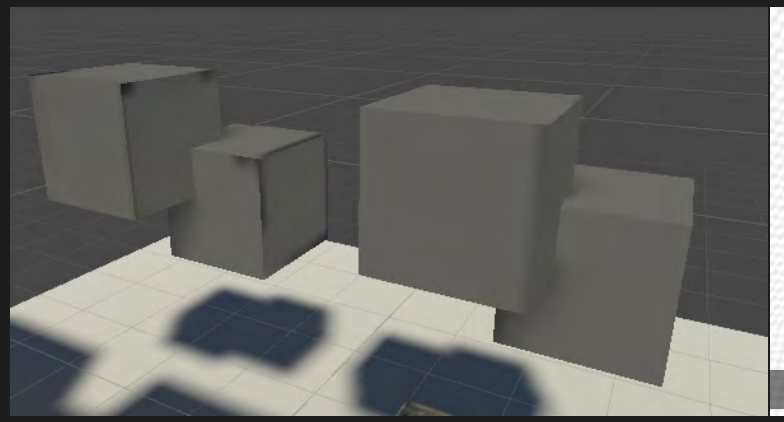
- Use GPU Progressive Lightmapper, if available on your Graphic card

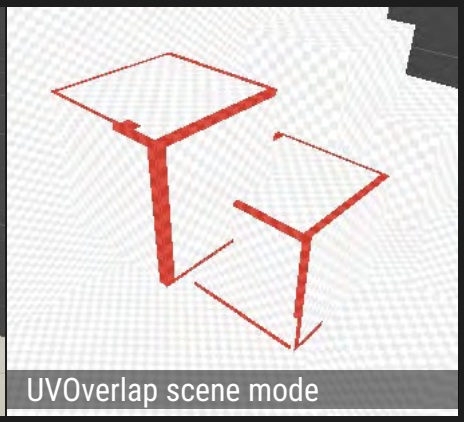
- Create 2 intersecting cubes in Maya
- Merge them in one mesh: Mesh/Boolean/Union
- Use UV Automatic mapping with ShellSpacing 0%
- Export as 2Cubes_Shell0.fbx
- Do the same with ShellSpacing 5%
- Export as 2Cubes_Shell5.fbx
- Import into Unity and set its import size properly
- Set them as static and see their differences in the scene lightmap

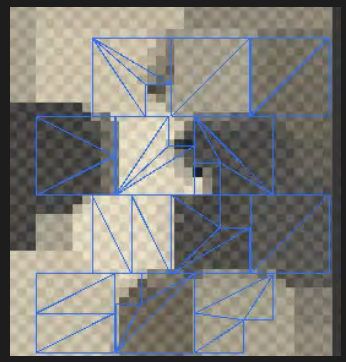
- If you are generating automatic UV1 set, you can achieve the same effect using
  - Duplicate 2Cubes_Shell0.fbx inside Unity and rename in 2Cubes_Shell0_AutoShell5.fbx
  - In the import settings, set GenerateLightMapUV ON and PackMargin Value to 40 or higher
  - If you open the object Lightmap in Chart visualization, you can see in realtime the UV distancing from themselves, creating the space between them that you need
  - Generate lighting again: you can see that the result is similar to 2Cubes_Shell5.fbx

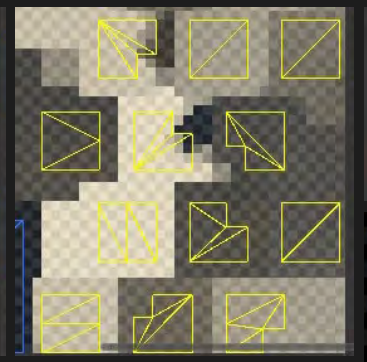


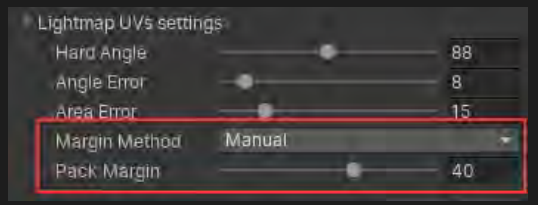


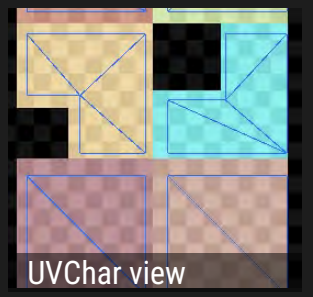










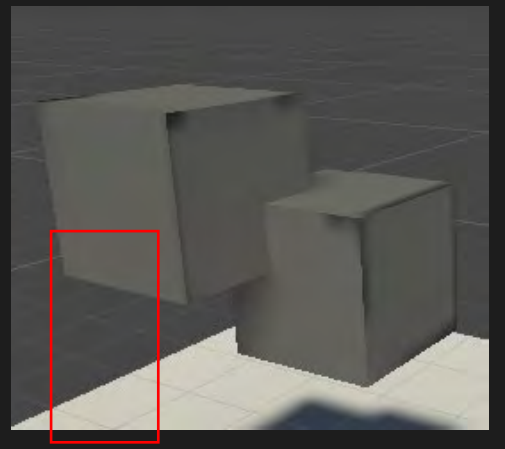


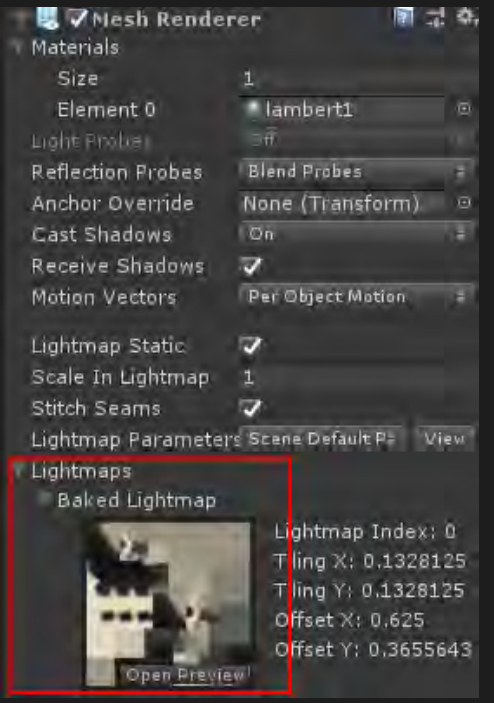
Hightlight UVs problems via UV Chart

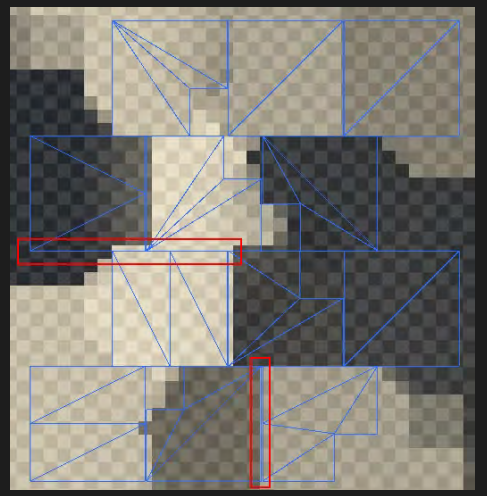
- The light leaking problems we have with shellspacing = 0 intersecting cubes are more visible if you switch in UV-Chart view (remember to use CPU Progressive lightmapper)

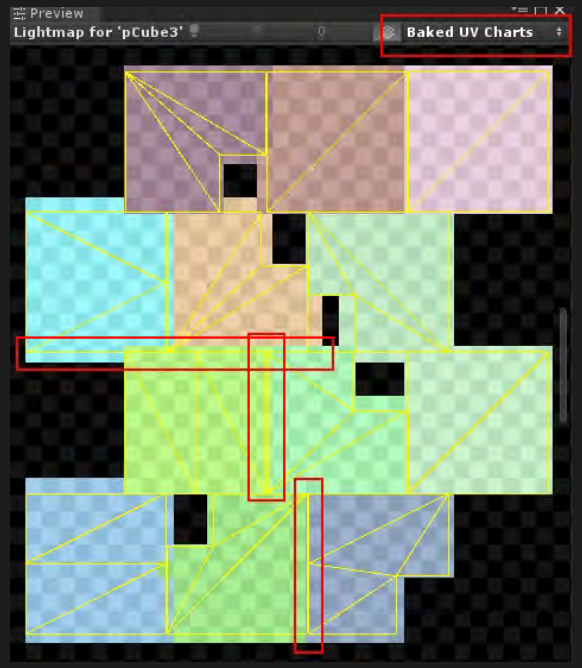
- To view it
  - Select GameObject
  - MeshRenderer / Open preview lightmap
  - Switch to BakedUVCharts

- Your goal is to avoid light leaking between UVCharts









Baked Lightmap parameters

Resolution

- How many texels does your object that is N Units large take?
- Always bake lightmaps at the largest atlas size possible. Fewer lightmaps require fewer Material state changes
  - NB: You don't need to include every last GameObject in the lightmap (which happens when setting GameObjects to Contribute-GI). While the above advice is generally true - you should mark all non-moving GameObjects as Static - you should omit small objects (gravel, cups, books), because adding them forces Unity to create another lightmap if there is not enough space. Small objects can look great when you light them with Light Probes

Size

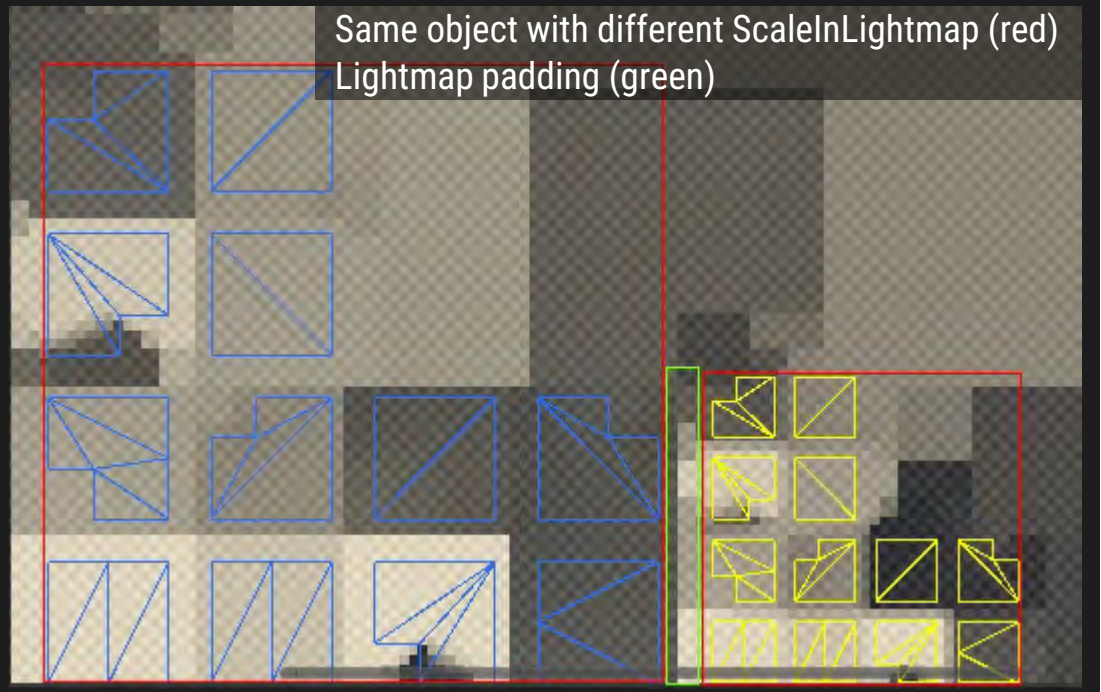
- This is the max resolution of the lightmaps of the scene. Every scene has one or more lightmaps, depending on how many scene objects are baked
- Eg. If Resolution is 10 and Size is 1024, and we have 2 quads of 100x100 units, they take 1000x1000 texels in a lightmap > We need 2 lightmaps of size 1024

Padding

- The # of texels between 2 different objs in the lightmaps
- This is different from PackMargin (that is the distance between 2 faces of the same obj)!

Scale in Lightmap

- If you need more detail for one particular object, you can tweak the ScaleInLightmap parameter
- Object lighmap resolution doesn't change if Resolution is 10/unit and the obj has Scale 1, OR Resolution is 5/unit and obj Scale is 2: it all depends on the default value you want for the scene objs



Cornell Box

- Someone at Cornell University 30 years ago built their Box, a Real box made out of simple geometry, with a single light source above the ceiling
- They probably had no idea that this box would stay around so long
- The idea was to photograph this box and try to achieve similar result in CG Graphic, simulating GI calculations
- This experiment illustrate was how important bounce lighting is when we see real physical light

Try it
- Bake the light

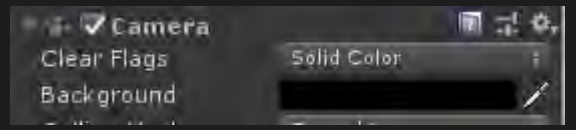
[http://www.graphics.cornell.edu/online/box/history.html]

[CornellBox_03]

Light types

- Start with no light contribution (see Lighting and Camera settings on the slide image)
- Directional light
  - Can be linked to procedural Skybox (default scene behavior)
- PointLight
  - [Built-int RP] Halo as a component (you can move slightly to adjust right position)
- Area light
  - Only in Baked
  - Light emitted in all directions uniformly across their surface area, from one side
  - Usually used with Unlit/Color shader

[CornellBox_03]

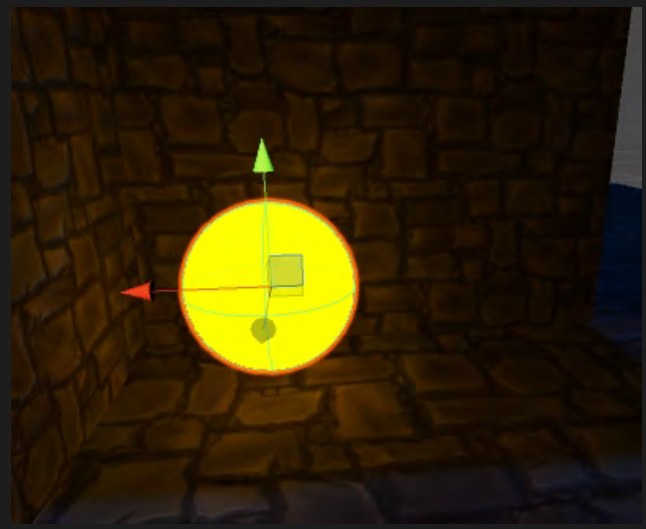


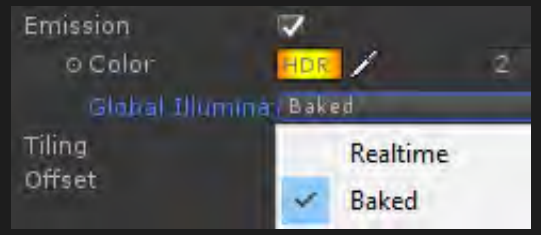
Light types

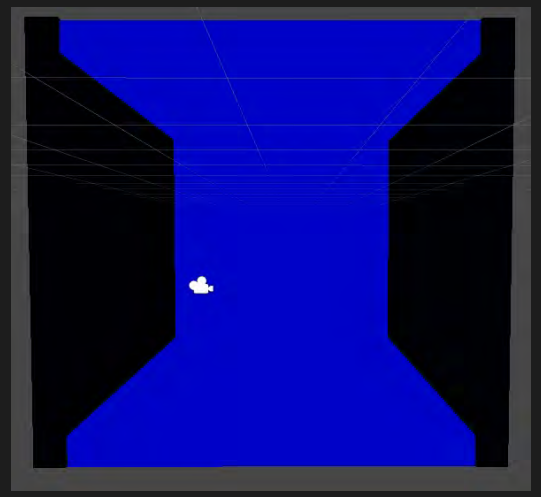
- Emissive
  - You can find Emission property in the URP/Lit shader
  - Visible only if GI is active
  - Emission will be received only by lightmapped static objects
  - Differences with Area light
    - Doesn't have IndirectMultiplier
    - Can emit light in both Baked and Realtime GI

- Ambient
  - Lighting window settings
  - NB: this color is multiplied by the final color: if we use ONLY the ambient color Blue with a Cornell Box, Red and Green Walls becomes Black (0,0,1) multiplied by (1,0,0) and (0,1,0)!

- Area & Emissive light intensity will diminish at inverse square of the distance
- Area light IndirectMultiplier
  - 0 no indirect light
  - <1 ray light intensity decrease every bounce
  - >1 ray light intensity doesn't decrease







Light Explorer

- Window/Rendering/LightExplorer

Cookies

- Environment effects (Cinema, Theatre)

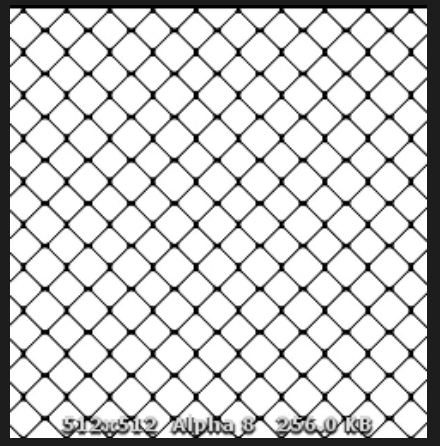
1. Set texture as Cookie Asset
2. If it doesn't have alpha, set AlphaSource FromGreyscale
3. If Pointlight, Specify what kind of light and mapping do you prefer
    1. MirroredBall / 6 Frames / LatLong / Auto
4. Set Appropriate WrapMode
    - For Firectional light probably you want WrapMode Repeat

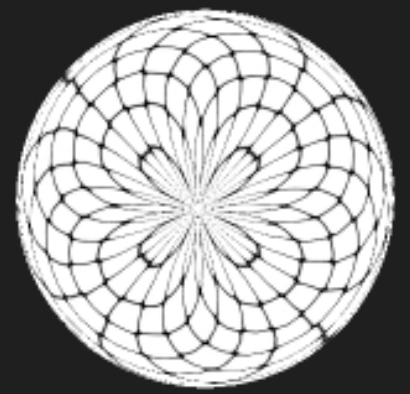
If DirectionalLight, we also have CookieSize Attribute on the light

Used for
- Change the shape of a light
    - A dark tunnel with striplights along the ceiling
    - Monitor screen glow should be restricted to a small box shape
- Can also incorporate grayscale levels. Useful for simulating dust in the path of the light
    - Torch light glass usually contains ridges that create caustic patterns

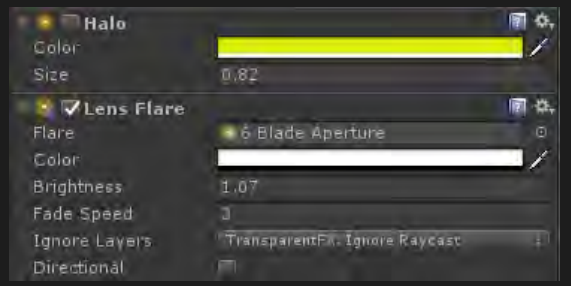
Try it
- activate PointLightCookie
- Import Other/CageCookie to the lantern PointLight
- TextureShape/Cube, Mapping/6FramesLayout
- Activate LanternRoot/LerpLoop.cs to move the lantern

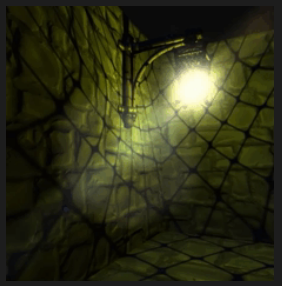
[CornellBox_03]





Halo / LensFlare



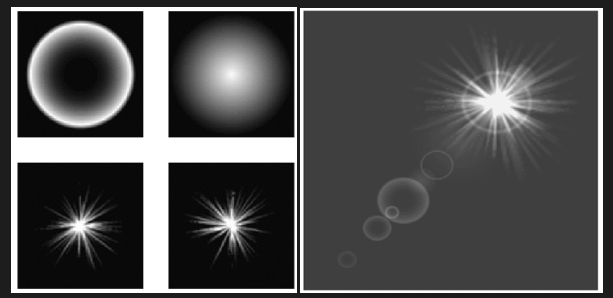


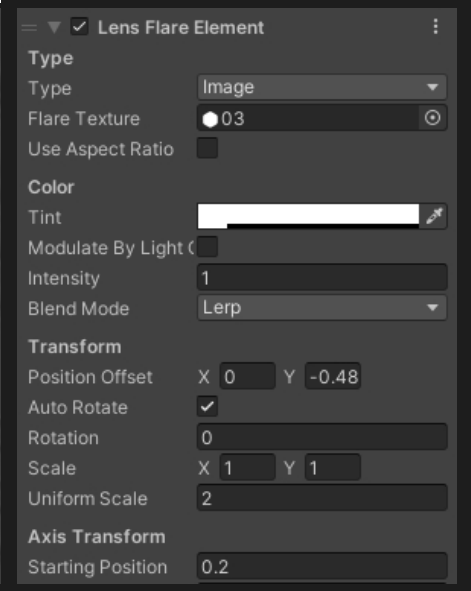
Halo
- [BIRP] Halo effect size and color is linked to light parameters
  - To override this use Halo component

Flare
- Imitates a bright light source seen through optical glass
- Depends on the intensity, shape, orientation of the light

- [BIRP] Use LensFlare component
  - FadeSpeed How quickly the flare will fade
  - Directional The flare will be oriented along positive Z axis of the game object. It will appear as if it was infinitely far away
  - Lens Flares are blocked by Colliders, even if the Collider does not have a Mesh Renderer.
    - If the Collider is a Trigger, it will block the flare if ProjectSettings/Physics/QueriesHitTriggers is true

- [URP] Use LensFlare (SRP) component
  - Create a LensFlare(SRP) asset and add [/Other/LensFlareSRP/00,01,02,03 elements with images values
  - To know more about this, open URP/Samples/LensFlares sample package
  - Occlusion
    - Use OcclusionRadius to increase fade in/out times





LightMapper components

Precompute

- Executed only for Realtime GI
- Calculates light transport in a scene: doesn't directly work on lightmaps texels, but instead creates a low resolution approximation of the static geometry in the world, called clusters
- Only depends on the static geometry and not the materials or light (Based on concept of visibility)
    - Packing -> Clustering -> Compositing the light transport

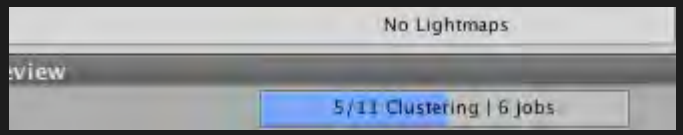
Real-time solver

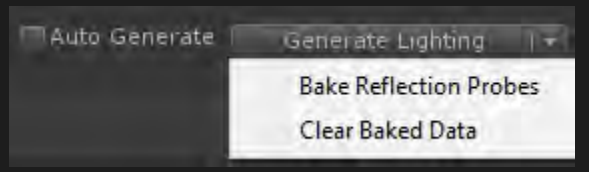
- Executed only for Realtime GI
- Combines the precalculated data with the material and light information to produce realtime GI light maps @ real time

Light map baker

- Produces baked light maps for direct and indirect light and also AO
- Uses Path Tracing method

- AutoGenerate flag use an internal Cache. Always use Manual Generate Lighting before building for all Scenes. Unity will save lighting data as Asset files in your project folder
- To Clean GI cache: Preferences/GI Cache





Baked GI –Progressive Lightmapper

- **Setup**
  - **Create a new project with packages Probuilder, PProcessing**
  - Realtime GI OFF
  - Mixed Lighting ON
  - IndirectIntensity 1
  - AlbedoBoost 1
- Directional Light
  - Intensity 2
  - IndirectMultiplier 1
  - Baked
  - BakedShadowAngleAngle 0

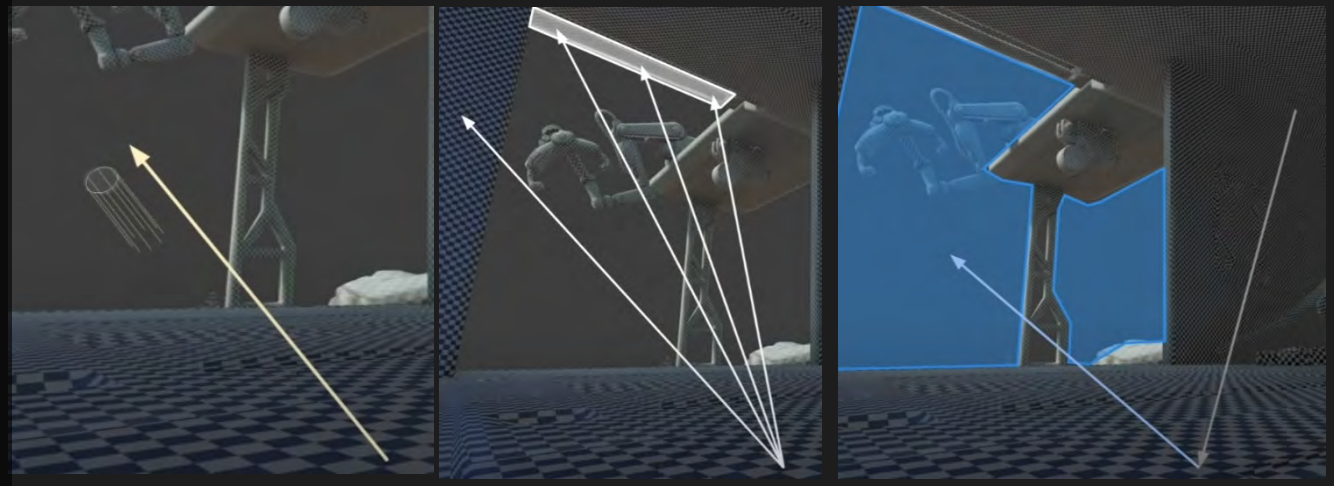
- Unlike Enlighten, CPU/GPU Progressive lightmapper gives instant feedback: at the moment you start baking you see changes happening and you can iterate faster
- Better estimation of the result and better baking time
- Enlighten results isn't the same, because of the different ways that these light mappers handle light map calculation

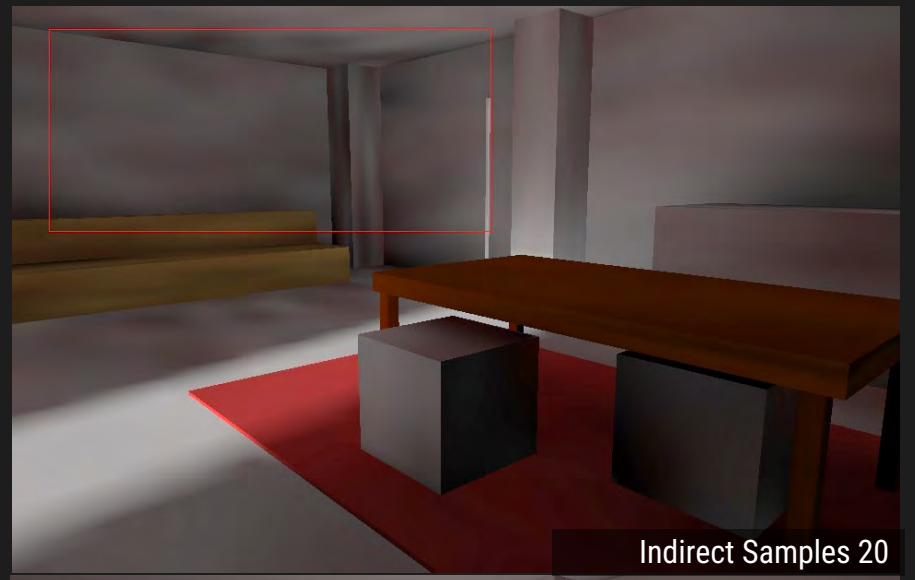
[Room_07]

Baked GI

- Direct/Indirect samples Number of samples used in calculating direct and indirect lighting. The higher the sample value the higher the quality and the lesser the noise
  - Outdoor scenes with directional lights < 100 samples
  - Indoor scenes > 100 samples to avoid noise
- imagine N rays pointing out from each texel of the lightmap
  - Direct light:
    - ray hits the sky AND has a direction coherent with the directional light => this texel is lit by the directional light
    - 10 rays hit an area light => this texel is lit with a high intensity from that area light
  - Indirect light
    - Ray hits the floor, and after one bounce hits the sky, with the same direction of the directional light => this texel will be a little bit blueish
- Min/Max Bounces Some rays will bounce Max times, some Min times

[Room_07]





Indirect Samples 20

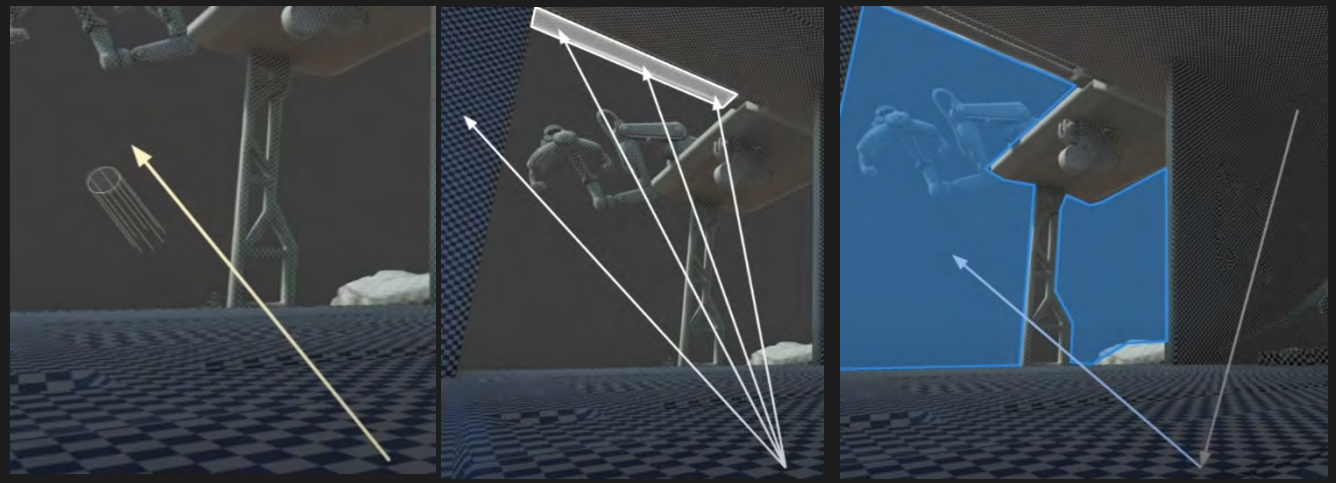
Indirect Samples 2000

Baked GI

- In Unity < 2019, the environment used to be sampled as part of the indirect samples. Everytime an indirect ray was shot from a lightmap texel, or bounced off a surface, and didn't hit anything, the environment was evaluated. The problem with this approach is that high frequency environment maps would lead to a lot of noise due to undersampling.
- With the latest changes the first difference is that environment sampling has been split into direct and indirect environment sampling. Direct environment sampling is controlled via the new sampling parameter in the UI and determines how many occlusion rays are shot into the environment from a lightmap texel. If you have noise coming from the parts of the environment that are directly visible by a lightmap texel, increasing this sample value will give better results.
- For indirect rays, everytime the ray bounces off a surface, an additional environment occlusion ray is also shot. This also means that if you are getting noise from the environment via bounce lighting, you have to increase the indirect sample count to get better results. Increasing "Environment Samples" in the UI will not make a difference in this case. So whereas before, for one indirect sample, you could get at most one sample of environment contribution, with the latest changes you get an environment ray at each bounce plus the environment rays spawned directly at the lightmap texel.

Lighting multiple importance sampling

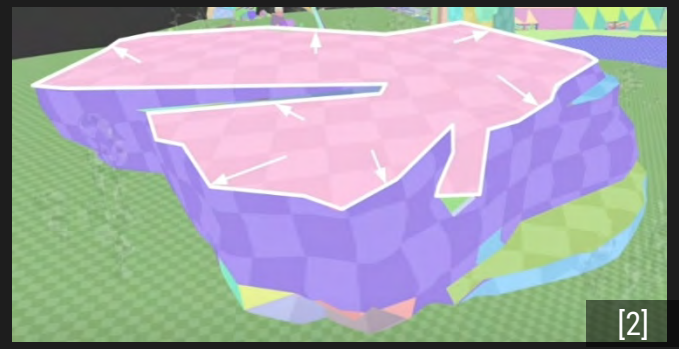
- The second difference is how the directions for the environment rays are determined. Before, the ray direction was more or less a random direction from the hemisphere. This works alright when the environment map has low frequencies in it, but breaks down with high frequencies (think HDR map with a small bright spot for the sun at the horizon). When enabling MIS, we analyze the environment map and find spots of high influence in it. When generating the ray directions, we alternate between either picking a random direction like before, or picking a direction from the pre-calculated points.
- MIS works best for high frequency environments, but will produce noisier results, the smoother the environment gets. The worst case is an environment of uniform color.

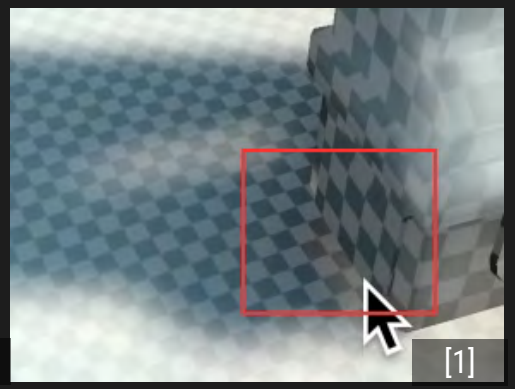


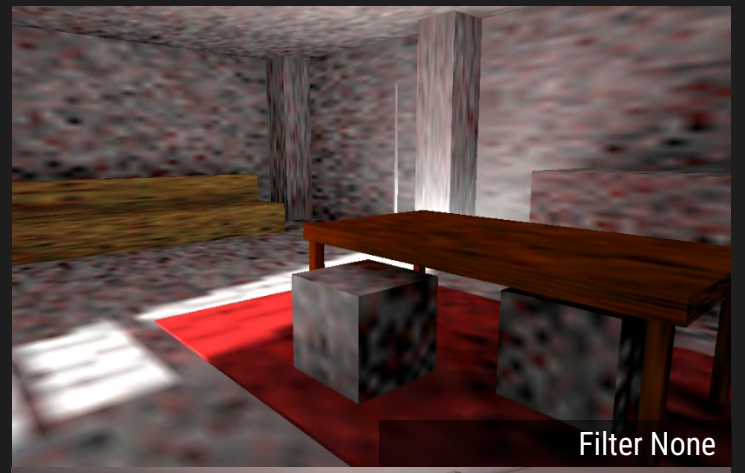
Baked GI

Filtering Configure the way the Progressive Lightmapper applies post-processing to lightmaps to limit noise

- Rays per texels goes in random direction. Sometimes the samples find a lot of light, sometimes less. And this randomness creates differences in color, which results in noise
- You could potentially up the number of the samples so high that you get no noise, but the cost of that would be too expensive
- Depending on your GPU and SO, it allows to apply different Filtering techniques
- Filtering will be performed only between texels that belong to the same UVChart [2]
- Gaussian filter applies a bilateral Gaussian filter on the lightmap. This blurs the lightmap and reduces the visible noise
  - It blurs also texels on different geometries: in [1] there is a blur leaking orange color between the vertical texels and the floor
- A-Trous filter minimizes the amount of blur while it removes visible noise in the lightmap
  - Reads the geometry and tries not to blend texels that belong to different geometries
- Denoiser They are basically AI based filters: they don't just filter, but they run an algorithm to decide what to filter and what not


[2]


[1]


Filter None


Filter Advanced Gaussian radius 1


Filter Advanced Gaussian radius 4

Denoisers

Optimize a lot performances

- 20/50/50 Samples with a Filter and a Denoise: 3 min
- 1000 Samples with only Filter: 13 min

The final quality is the same!

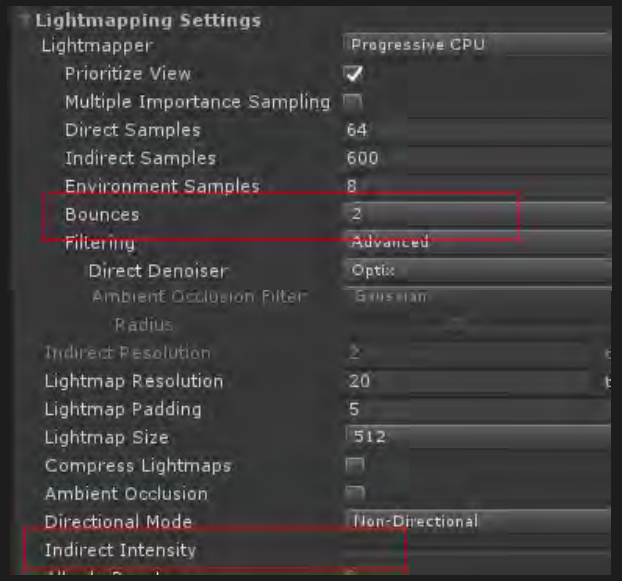
Name	Works on
Open Image Denoiser	All platforms
Optix	Nvidia
Radeon Pro	AMD

Baked GI

- Bounces The number of times the light bounces before it dies off
  - The higher the number of bounces
    - The higher the chance for it to capture color bleeding (E.g. from a red wall to a white floor)
    - The brighter the scene
  - Every time the light bounces off, the color bleeding amount is calculated based on Indirect intensity value

Try it

- With 0 bounces, there is only direct light!



Baked GI

LightMap resolution The number of texels/unit in the lightmap occupied by the object

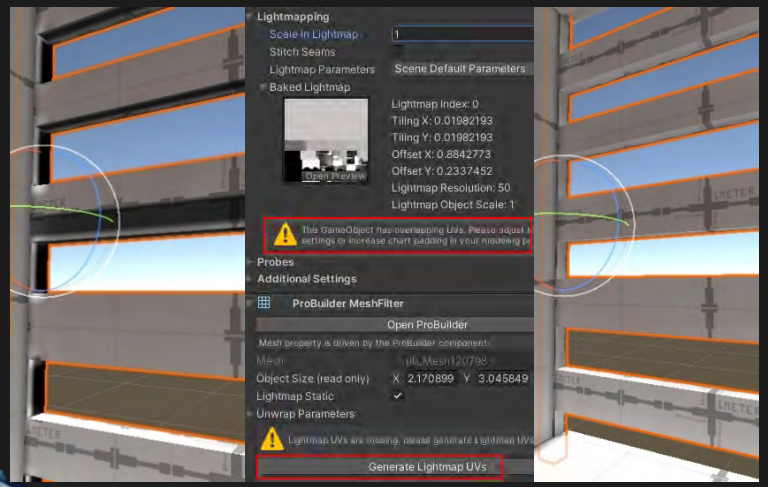
- This number is multiplied by the ScaleInLightmap parameter in the MeshRenderer component
- Try it
  - Activate Grid Gobj
  - Bake Light with a LMResolution 20 and 50, and see the differences

LightMap size The max size of the lightmap

- If there are too many objects in the scene and they don't fit into one lightmap, another lightmap is allocated

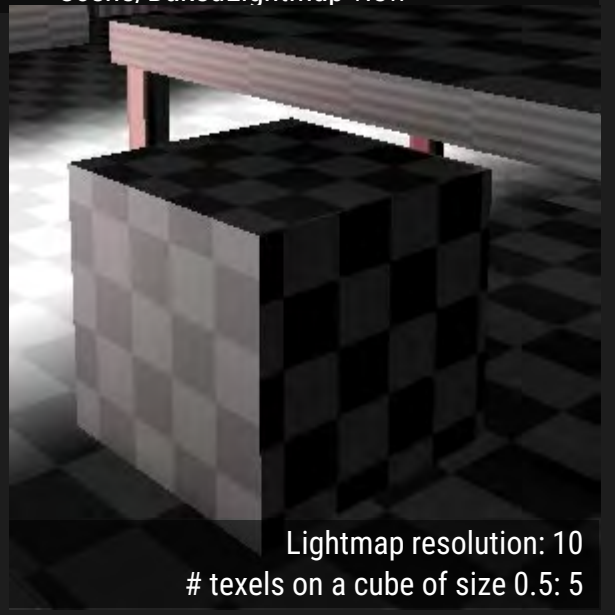
Try it

- In [1a/b] you can see the difference activating the LigthMapResolution(Grid)/Grid GameObj in the hierarchy
  - That obj has bad UVs: it is a Probuilder Obj. Press Generate LightmapUVs



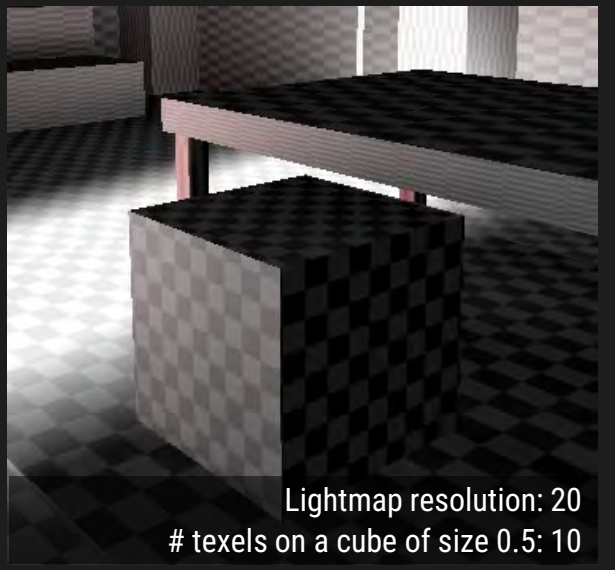
[1a] Lightmap resolution: 20

Scene/BakedLightmap view



Lightmap resolution: 10
texels on a cube of size 0.5: 5

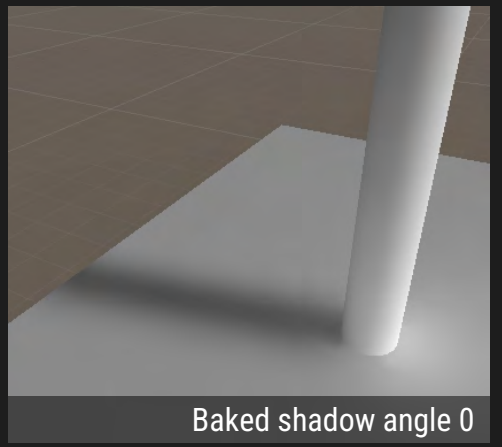
[1b] Lightmap resolution: 50



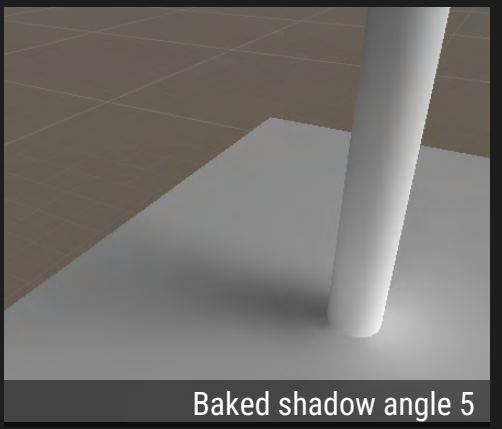
Lightmap resolution: 20
texels on a cube of size 0.5: 10

Baked GI

- BakedShadowsAngle/Radius The amount of softness penumbra to apply to Directional/Spotlight lights
- Try it
  - Activate BakedShadowAngle Gobj on the Roof
  - Change BakeShadowAngle value (0, 5) and see the differences



Baked shadow angle 0



Baked shadow angle 5

Baked GI

Don’t waste Lightmap texels on

- Surfaces in your scene with a low light variation on them, such as a surface lit by one type of light, or in the shadow of such a light
- Surfaces which receive indirect light only, because light bounced off a diffuse surface is much lower frequency than the direct lighting, and storing it doesn’t require that many texels
- Small objects in your scene, such as stones
- Thin objects, for example, wires. The texels get crammed into a tiny space and don’t contribute much to the quality of the final image
- Surfaces that are hidden from the player

Baked Lighting

Useful for

- local ambience: these lights are pre-calculated before run time. They are not included in any run-time lighting calculations

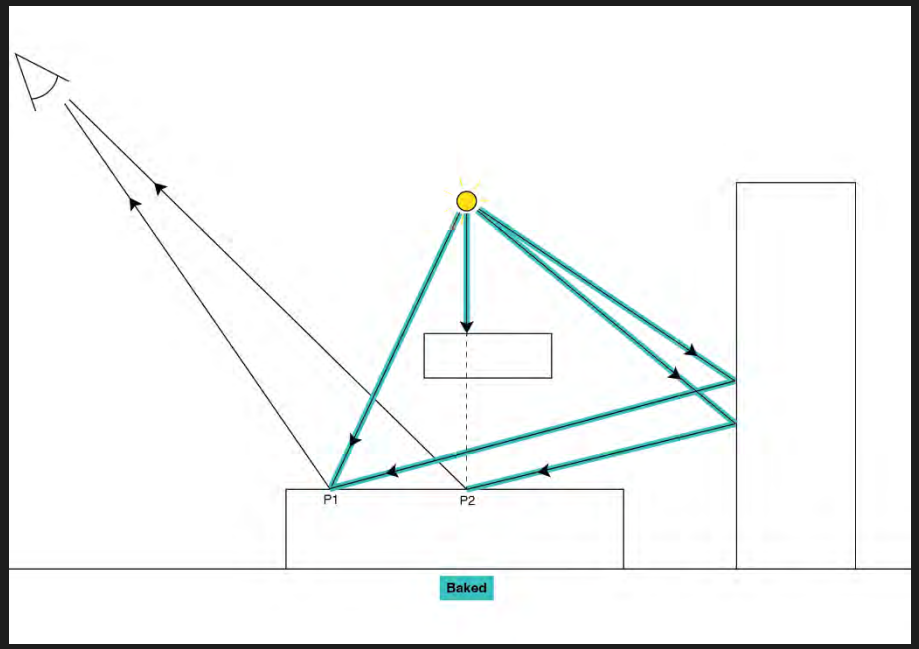
PROs

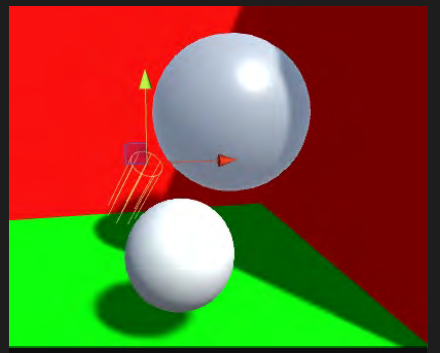
- High-quality shadows from statics GObjs on static GObjs
- Lighting for static GObjs is one Texture fetched from the light map in the Shader

CONs

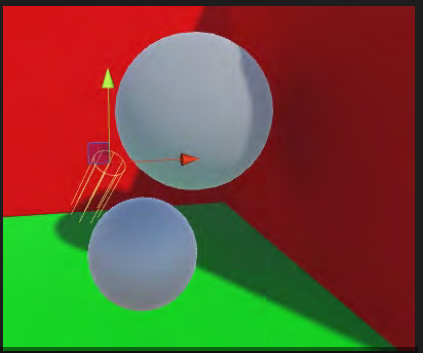
- No real-time direct lighting (no specular lighting effects - light direction information is not available to Unity at run time)
- No shadows from dynamic GObjs on static GObjs
- You only get low-resolution shadows from static GObjs on dynamic GObjs using Light Probes

[UnityLightingModesReferenceCard/Baked]





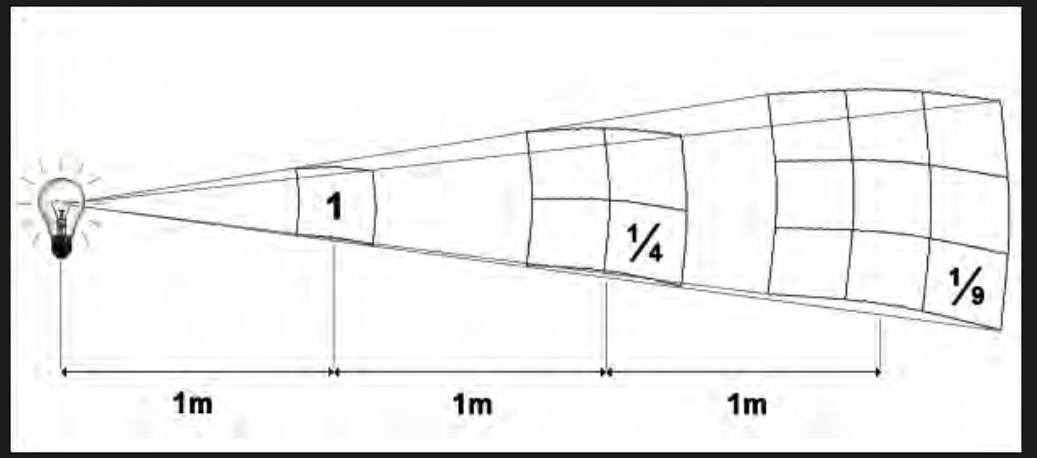
Realtime Light
Shadows and Specular

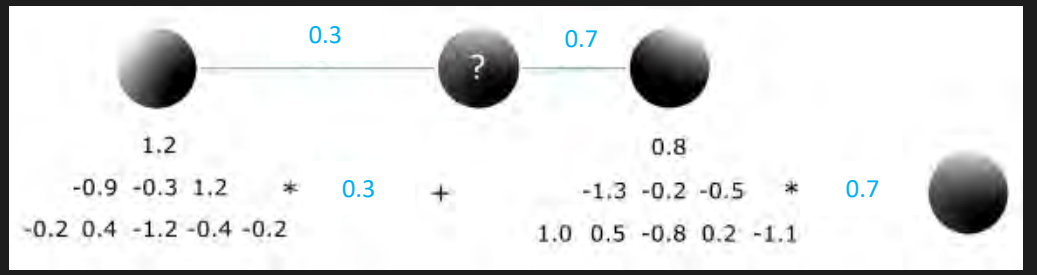


Baked Light
No Shadows from dynamic Objs and No Specular info

Light Probes

- Lightmaps store info about light hitting the surfaces, Light Probes store info about light passing through a point in space
  - provide high quality lighting (including indirect bounced light and shadows) on dynamic objs
- Irradiance
- A LightProbe use third order polynomials, also known as L2 Spherical Harmonics, L2 SH, to store directionality of RGB ray values
  - These are stored using 27 floating point values, 9 for each color channel
- Interpolating two probes can be done just by interpolating their coefficients





Light Probes

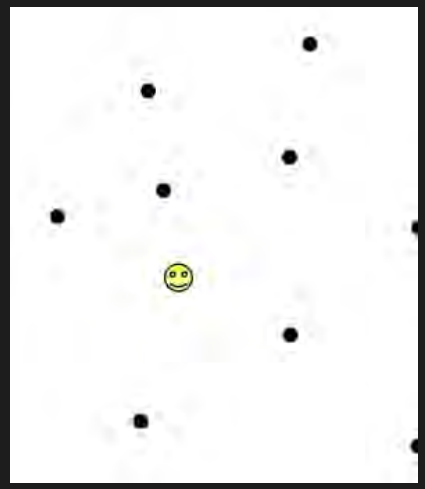
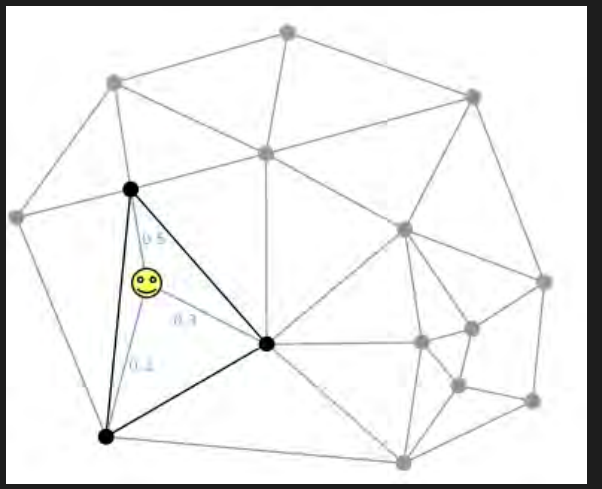
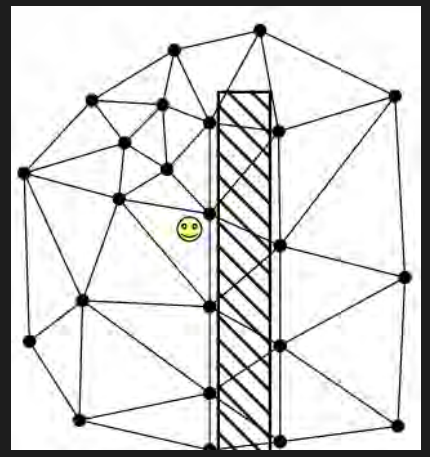
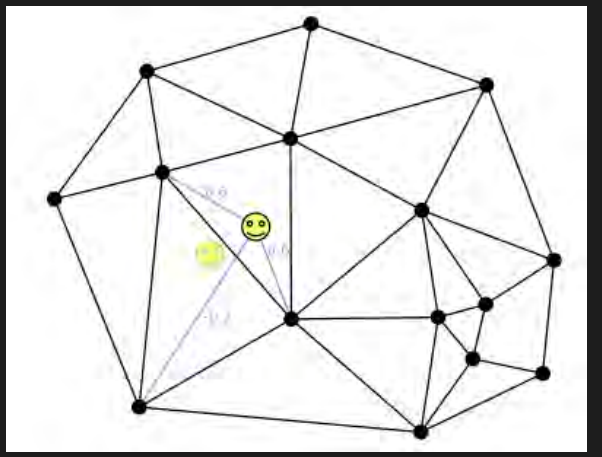
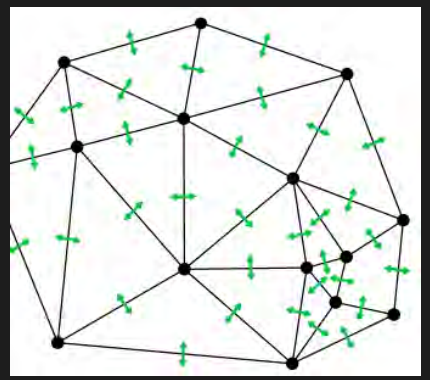
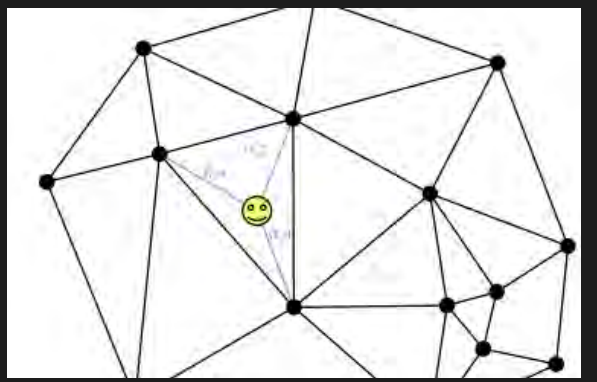
- How many pts do I need to define a circle?
- Delaunay Triangulation (Every circle doesn't contains other vertices)
- 2D > 3D Delaunay Tetrahedralization

1. Cache the tetrahedron index from the previous frame
2. Calculate barycentric coordinates Bc
    1. Bc > 0 => We are inside
    2. Bc < 0 => The obj moved the most negative coordinate
3. Each tetrahedron has exactly 4 neighbors. Find the right one (the one with only positive weights)

Data needed

- probe positions probe_count * 3 floats
- SH coefficients probe_count * 27 floats
- hull rays hull_probe_count * 3 floats
- Tetrahedron indices 4 vertices + indices 4 neighbours

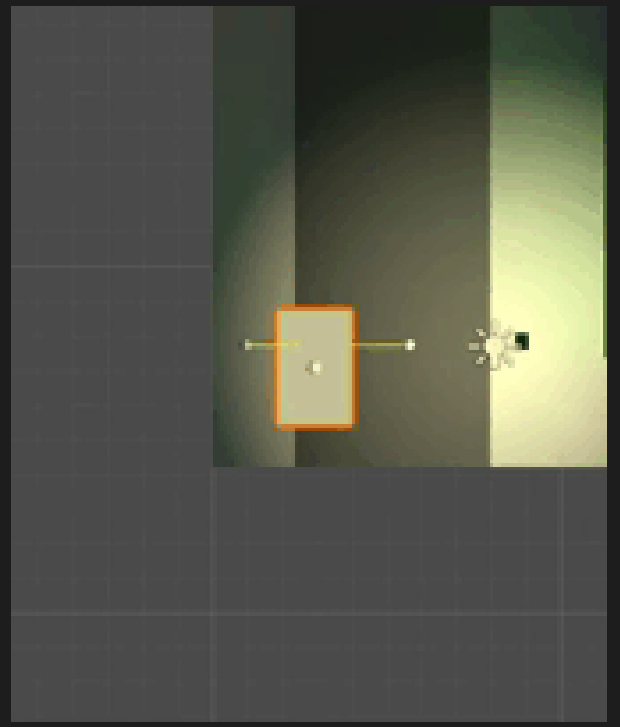
- The unwanted influence of the probes from the right side of the wall can be completely avoided by placing a couple of probes along the wall
- The influence of each probe will be limited to a roughly circular area of around 6 triangles originating from that probe
- A probe's influence area can be limited by adding probes around it

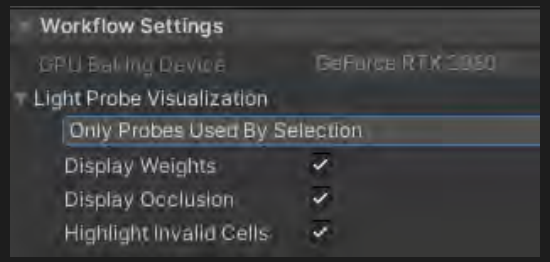







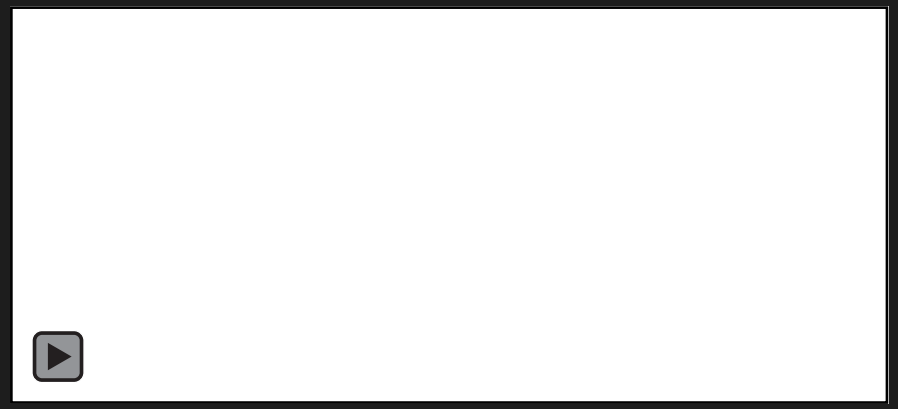
Light Probes - Placement

- Best to add it to a new empty GameObject
- Light probe info resolution = how closely packed are the probes
- Good rule: Condensed pattern around areas that have complex or highly contrasting light
- Never use a 2D disposition (even for driving on a road games). At least two vertical “layers” of points in your group of probes: this will allow to calculate sensible tetrahedral volumes from the probes
- Dynamic Objs MeshRenderer LightProbes
  - BlendProbes
  - [BIRP] UseProxyVolumes + lightProbeProxyVolume large moving objs
  - Anchor override This may be useful when a GameObject contains two separate adjoining meshes; if both meshes are lit individually according to their bounding box positions then the lighting will be discontinuous at the place where they join. This can be prevented by using the same Transform (for example the parent or a child object) as the interpolation point
- See [LightProbeGroup/LPAdder.cs] script to know how to access light probes on a light probe group. NB: Light probes positions are RELATIVE to the LPGroup position!

- To see them, setup LightProbeVisualization under Lighting/WorkflowSettings



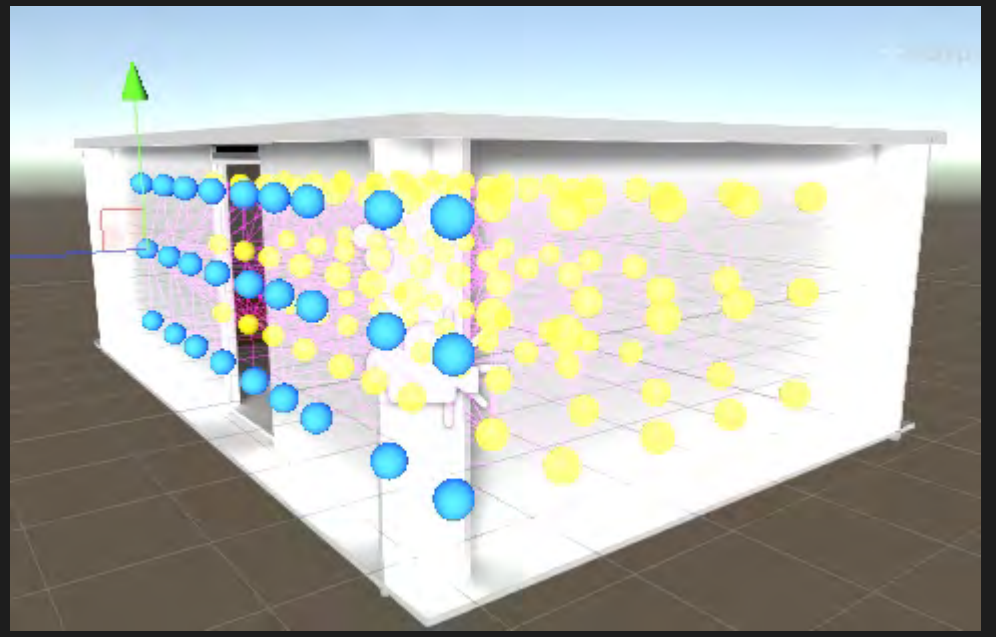


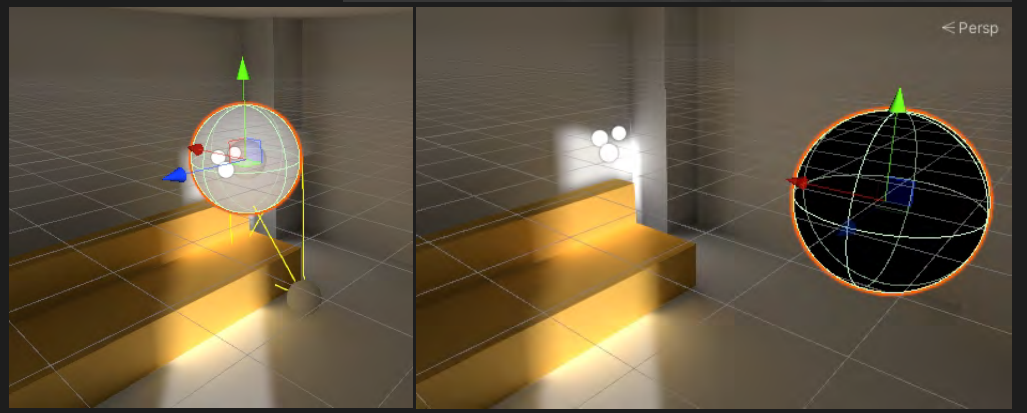


Light Probes - Placement

- Try it
  - Activate LightProbes/LightProbeGroupToAdd
  - Add LightprobeGroup and manually distribute lightprobes around the room
  - Generate Lighting
  - Activate DynamicSphereLightProbesOFF: it will be black, because no GI is contributing to its color
  - Activate DynamicSphereLightProbes: it will be lit by lightprobes

[Room_07]





[BIRP] LP Proxy Volume

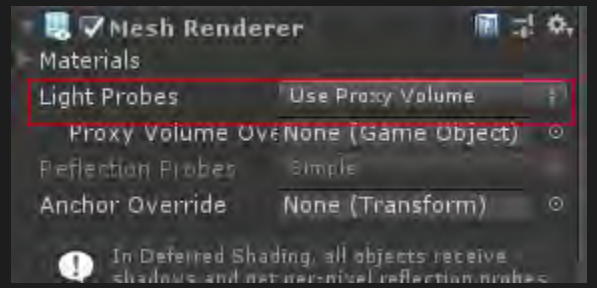
- [URP] are in research
- Large moving objs
- Particles

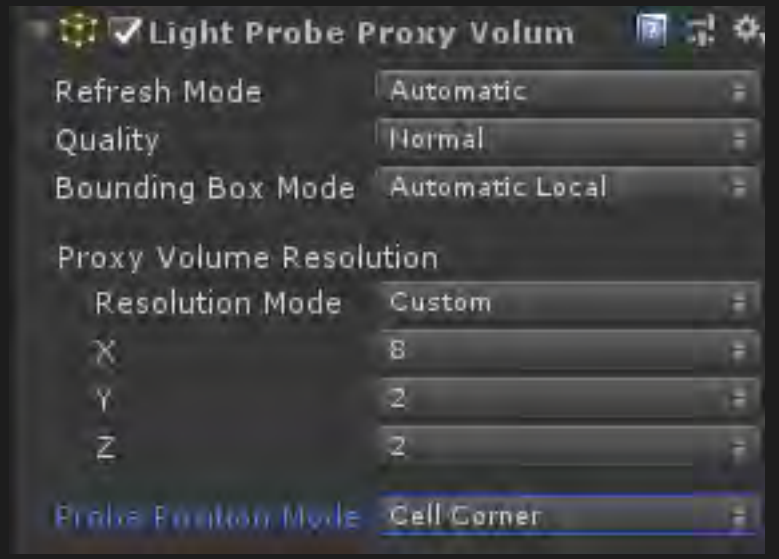
- UseProxyVolumes + lightProbeProxyVolume
  - ProxyVolumeResolution
  - ProbePositionMode

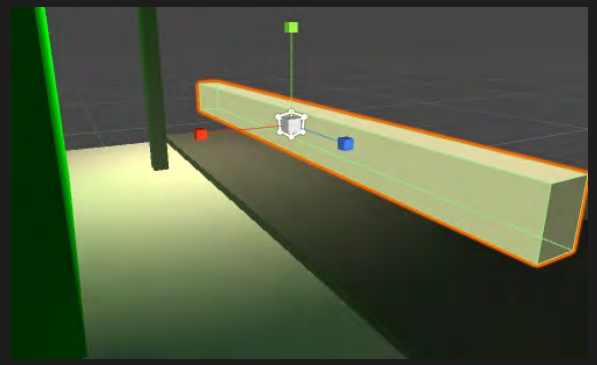
- Try it
  - Activate LongDynamicCube
  - Add LightProbeProxyVolume component
  - ResolutionMode Custom

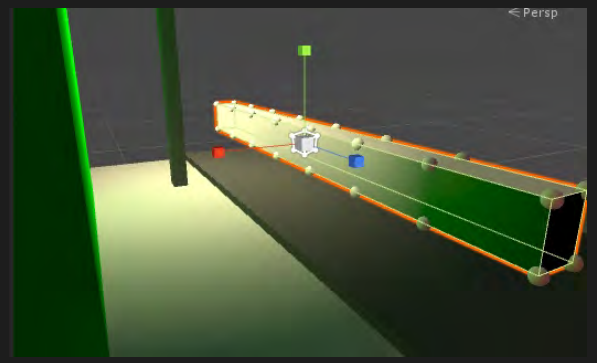
- Refresh mode via script
  - LightProbeProxyVolume.Update()

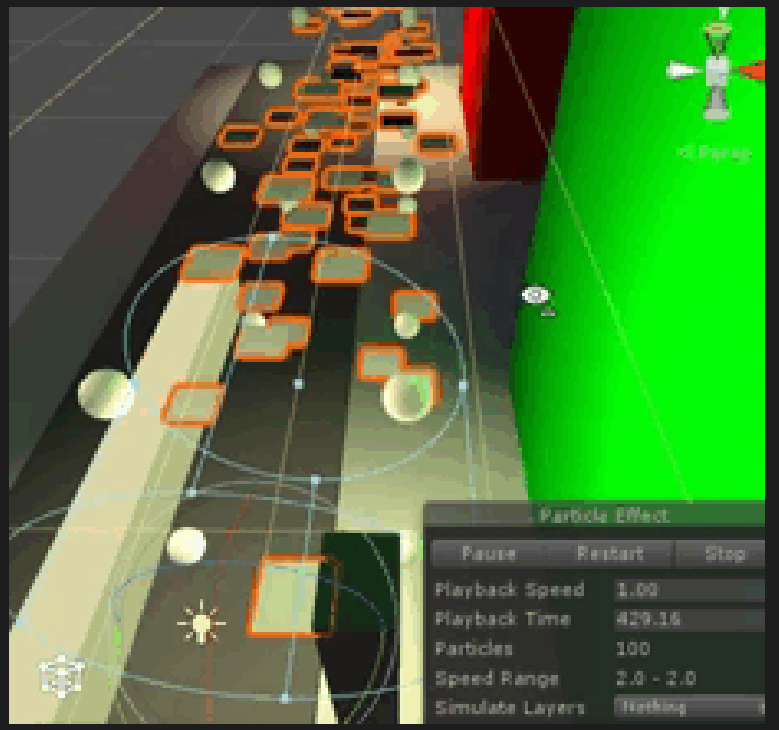
- Try it
  - Activate LongDynamicCubeLPPV
  - Set RefreshMode to ViaScripting
  - Enter PlayMode (otherwise le LPPV will be automatically updated)
  - Use LPPVUpdater.cs to update via script







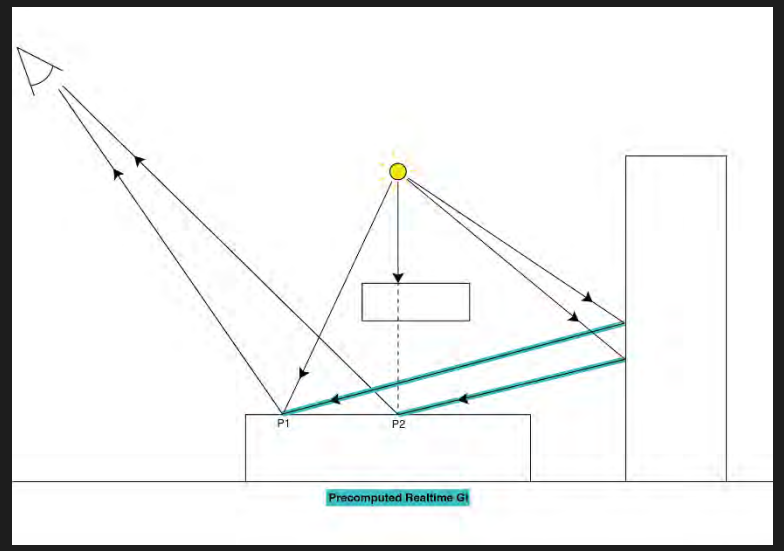




[LongCube/LPPVUpdater.cs]

Realtime GI

- Unlike Baked GI, Realtime GI lightmaps are dynamic, they change in realtime
- RT GI is useful for lights that change slowly and have a high visual impact on your Scene, such as the sun moving across the sky, or a slowly pulsating light in a closed corridor. Realtime GI is not intended for Lights that change quickly or for special effects, due to performance cost and latency.
- The resolution of Realtime GI lightmap MUST be low
  - We need another lightmap texture, with different UV cords: UV2
- To update the RT GI Lightmap, we need to perform a lot of calculations before the app is running, starting from the Precompute/Packing step



Realtime GI vs Baked GI

- When working with Precomputed Realtime GI, we need to calculate the bounces of light around the static geometry within a Scene in the Unity Editor and storing this data for use at run time. This process reduces the number of lighting calculations that must be performed at run time, allowing Realtime bounced lighting while maintaining interactive framerates
- When using Baked GI, traditional lightmap textures are generated offline during the precompute process. These textures then exist as assets within the project and cannot be changed at run time. Precomputed Realtime GI does not create lightmap assets in the same way. Instead, lighting data is saved as a Lighting Data Asset which contains the information needed to generate and update a set of low resolution lightmaps interactively, at run time

- **Setup**
  - Realtime GI ON
  - Mixed Lighting OFF
  - Activate Directional Light Realtime
    - Intensity 1
    - Indirect Multiplier 2
  - Generate Lighting
  - In [URP], there is no need to enter Play-mode to calculate RT-GI in the right way

- Try it
  - Rotate Direction light around Y axis and see the GI changing in RT

Precompute/Packing

- In Unity's Lightmaps
  - a Chart is an area of a lightmap texture to which we map the lightmap UVs of a given Scene object
  - A Single Scene object can span its triangles over multiple charts

- In BakedLighting, you can see UVCharts in SceneView/BakedGlobalIllumination/UVCharts
  - How many texels are inside a chart depend on
    - LightmapResolution
    - Scale in LightMap

- In RTLighting, you can see UVCharts in SceneView/GlobalIllumination/UVCharts
  - How many texels are inside a chart depend on
    - IndirectResolution
    - LightMap Parameters used by the object

- Large numbers of UVCharts in a Realtime Lightmap = high realtime GI precompute times

- Baked Lightmap Charts are spaced using the Padding value
  - No padding is needed for realtime GI because Unity clamps the lightmap UVs to give a half texel border inside the Chart during the packing stage of the mesh import pipeline, in order to prevent bleed caused by texture filtering
  - Each RT UVChart has a minimum of 4x4 texels

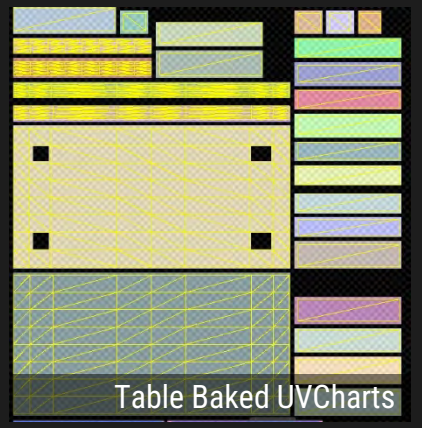
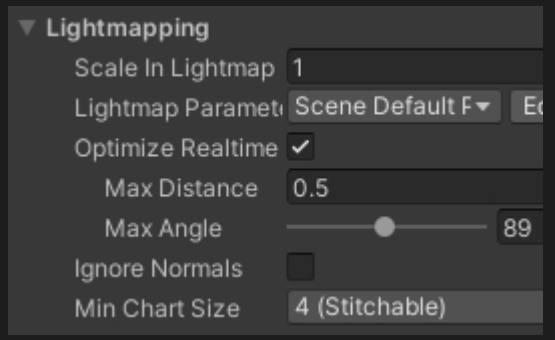


Table Baked UVCharts


Table RT UVCharts

MinSize of a RT UVChart

Precompute/Packing



How to create UV2?

A - Not optimized

- Usually produce artifacts, because of the low resolution lightmap
- creates one chart for each group of triangles in UVs that share vertices

B - Optimized

- MaxDistance a value of 0.5 will give acceptable results. For large objects with large faces, it may be necessary to increase it. This is to prevent suitable candidate UVs from being excluded from selection by the stitching algorithm. Increasing it will reduce the number of Charts required by the selected object. Decreasing it is useful in situations where there is visible stretching of lightmap texels and we may actually need more Charts in order to get the required texel coverage. The results of these changes can be easily assessed using the checkerboard overlay in the UV Charts Scene draw mode. Finding the right balance can often require a little experimentation
- MaxAngle defines the maximum angle permitted between faces sharing a UV edge and is calculated using the internal angle. If the angle between the backfaces is greater than this amount, these UV shells will not be considered for stitching. Increasing it will make it more likely that lighting UVs will be combined by Unity's unwrapping algorithm. This can be a good way to reduce the number of Charts required by a selected object. However, sometimes stretched lightmaps can occur. Decreasing it will make the unwrapper less likely to combine UV edges, which will result in more Charts but less distortion
- IgnoreNormals In certain cases the mesh importer may decide to split geometry. This will also affect Charts. If a mesh has an extremely high triangle count it may be more performant for Unity to split it into separate sub-meshes, such as reducing the number of triangles in each draw call. These splits occurs on hard edges. Charts can potentially be split during this process as edges that fall within a Chart may be separated, resulting in multiple shells which in turn will require additional Charts. In these cases, enabling the Ignore Normals checkbox will prevent the Charts being split for Precomputed Realtime GI lighting
- MinChartSize MUST be at least 4 if you want try stitching them

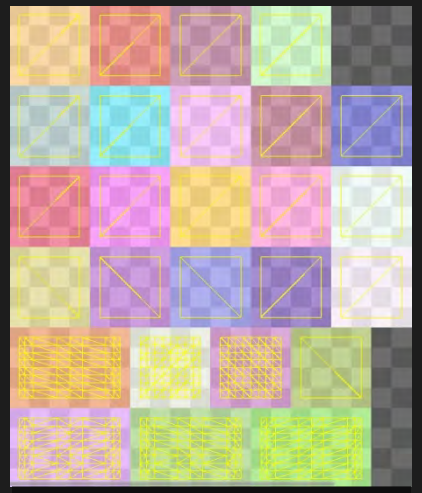


Table RT UVCharts
Not optimized

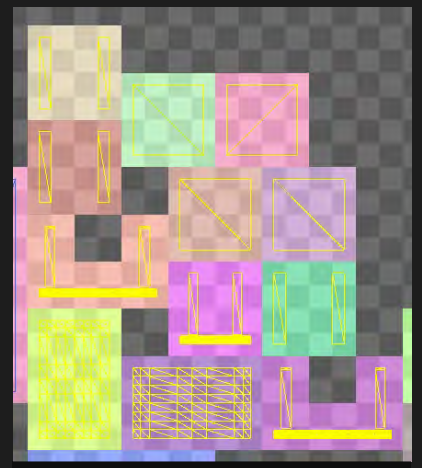
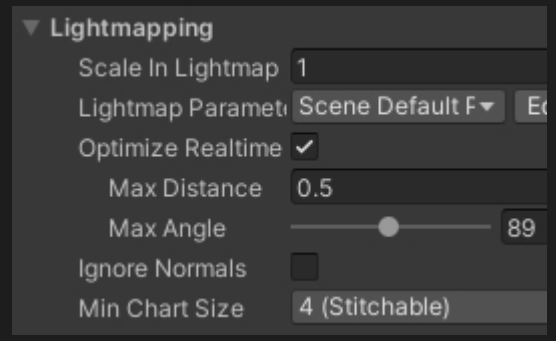


Table RT UVCharts
Optimized

Precompute/Packing



- Inappropriate Charting is the major culprit for lighting precomputes not completing or taking too long
- The number of Charts that an object requires is therefore largely determined by the number of UV shells (pieces) needed to unwrap the object in question

Packing main tasks

1. Identifies Charts in UV1 (for baked lightmapping)
2. Create UV2: Pack the Charts into lightmap used for real-time GI

Precompute/Packing

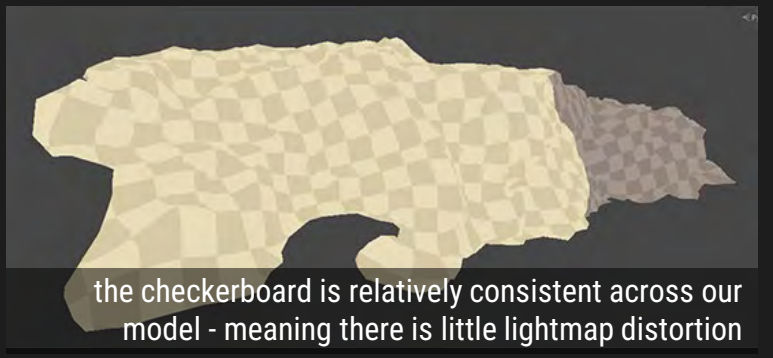
- If you are using RTimeGI, use Scene/UVCharts to check the auto unwrapping distortion

Try it: analyze HouseBig02 [RealTimeGI_Optimization]
- Keep Max Angle to a value of 180
- Ignore normal TRUE
- Max Distance
  - 10: All faces are inside this distance: there is only one chart
  - 0.8: now the result is much better

[RealTimeGI_Optimization_2]

MaxDistance 10: low cluster num, high distortion

MaxDistance 8: high cluster num, low distortion



the checkerboard is relatively consistent across our model - meaning there is little lightmap distortion

Stretching and warping of the checkerboard indicates that there is some inconsistency in the distribution of lightmap texels

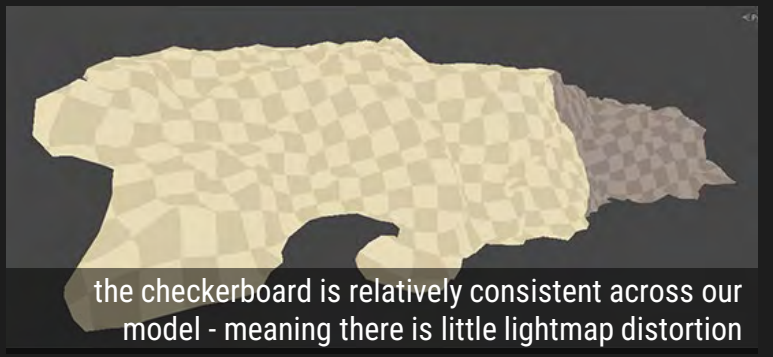
Precompute/Packing

- If you are using RTimeGI, use Scene/UVCharts to check the auto unwrapping distortion

Try it: analyze HouseBig02 [RealTimeGI_Optimization]

- Max Angle Parameter
  - 100: Too few charts: at least 2 house faces have the same UVChart
  - 0: too much charts
  - 90: this is ok, but the house has strange behavior on edges [1]
- MaxDistance 0.1 / MaxAngle 0 is the best result, but it could be too expensive[2]: you should try a tradeoff

[RealTimeGI_Optimization_2]

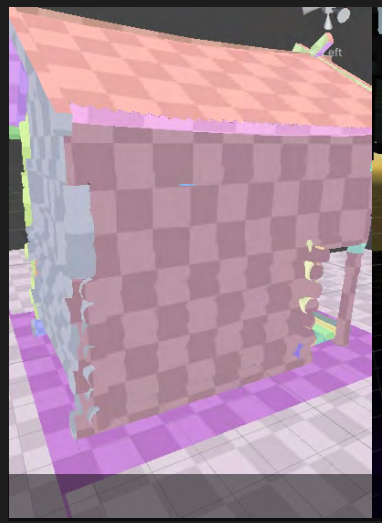


the checkerboard is relatively consistent across our model - meaning there is little lightmap distortion

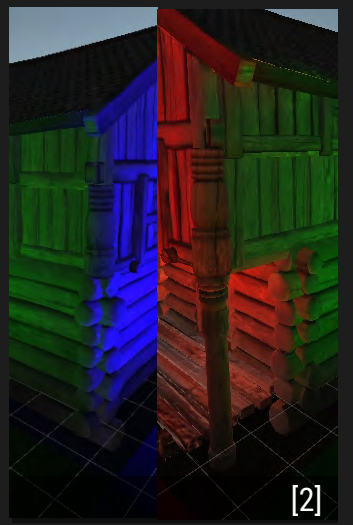
Stretching and warping of the checkerboard indicates that there is some inconsistency in the distribution of lightmap texels

MaxAngle 0

MaxAngle 100



[1]



[2]

Precompute/Packing

Unwrap settings: per-instance or per-Prefab?

- It is often useful to configure a default unwrapping and lighting setup for your Prefabs. If the Prefabs are to be instantiated numerous times, it can save effort to have the unwrapping for that object preconfigured rather than having to specify this repeatedly throughout the Scene. On the other hand, it is frequently desirable to configure lighting settings based on the context in which an object is used within the overall Scene. For example, if an instance of a Prefab will be seen up close within the playable area then it makes sense to use higher fidelity setting on that instance. If the instance of the Prefab is far away, then it would not make sense to use our lighting budget maintaining the same settings. Here, we would likely want to lower the quality of our lightmaps and use more aggressive stitching parameters in our unwrap

Precompute/Clustering

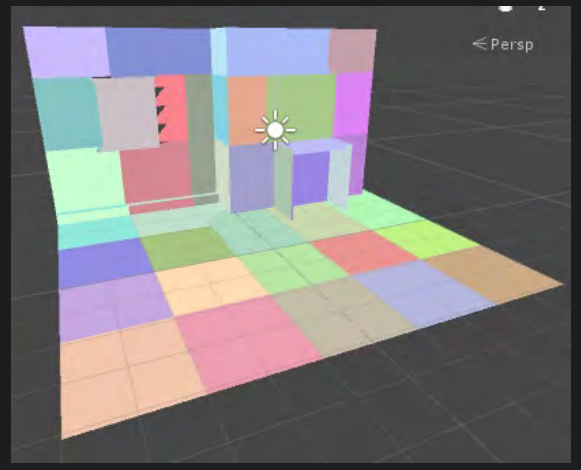
- Splits scene Geometry into clusters
- To calculate the size of each cluster, see next slides
- Use only triangles pos & orientation NOT UVs or Materials
    - Unity simplifies the calculations required by working on a voxelized version of the Static Scene. These voxels are called Clusters. Clusters are effectively surface patches (small tiles) mapped onto Static geometry which we use for lighting. Clusters are stored in a hierarchical relationships and are used for the complex irradiance calculations needed when precomputing Unity's diffuse global illumination solution. Although Clusters are mapped in a similar way to Charts, the two are actually independent

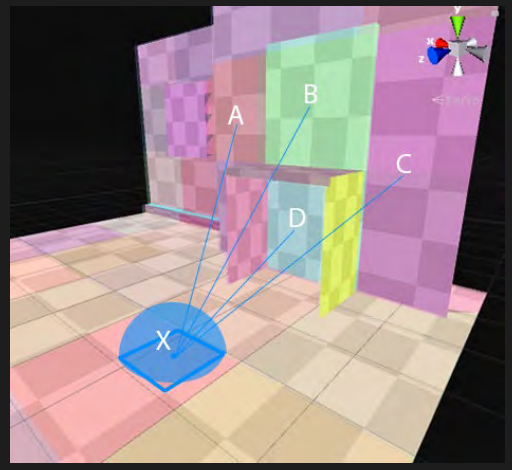
Benefits of reducing the number of Clusters

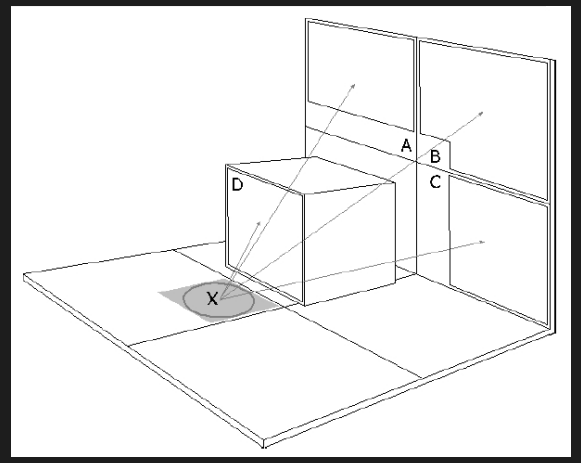
- Run time performance will be improved
- allow us to reduce the number of operations needed during later tasks in Unity's precompute process

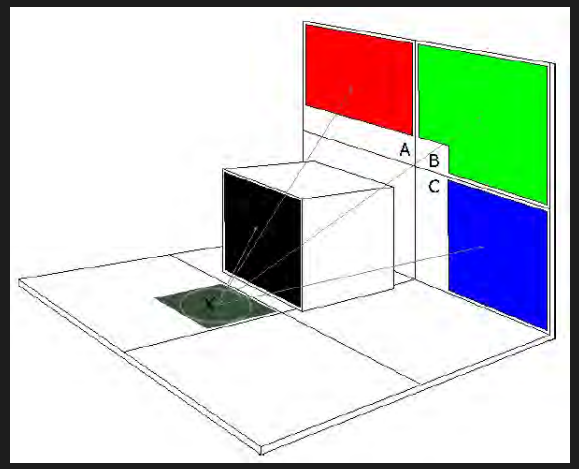
1. ∀ cluster in a real-time lightmap
    1. Cast rays in all directions of its hemisphere
    2. Calculates the visibility of the cluster (form factor)

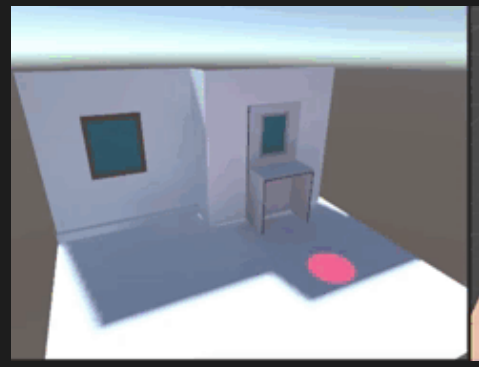
Lighting received from X = 0.05*A + 0.1*B + 0.05*C + 0.2*D

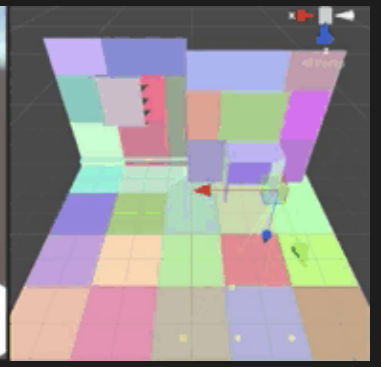












Precompute/Clustering

- Once the precompute has been completed, ambient (skybox) lighting along with light positions, intensities and colours can then be modified without needing to restart the precompute process. These lighting changes will bounce and permeate throughout the Cluster network, taking into consideration the underlying albedo and emission of the Scene's materials in the eventual output
- As this process is performed asynchronously on the CPU, the time it takes to refresh the global illumination solution is bound by the number of available worker threads
- Cluster Resolution can be considerably lower for very large objects, far from the Camera

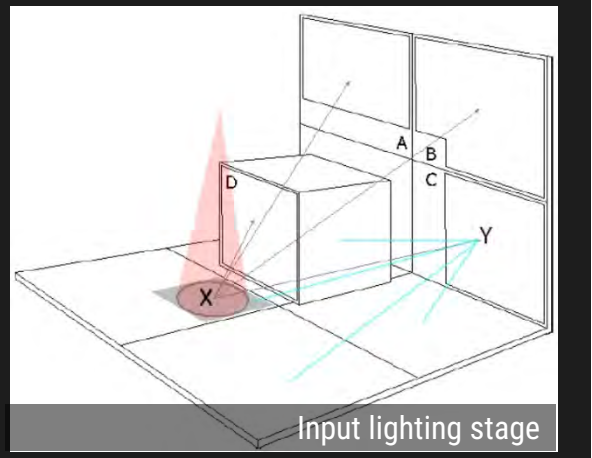
Real-time Solver

- Input lighting stage Direct lights contribute on the clusters, including dynamic lights
- Solve stage The solve stage sums up the cluster value multiplied by the stored form-factors, and stores the results in the light map
- If I want to calculate how much light cluster i intercept from clusters j-th around it:

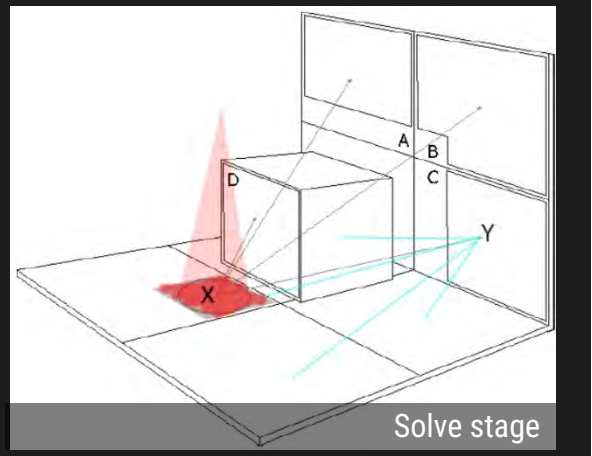
$$B_i = L_e + p_i \sum_{j=1}^{n} F_{ij} L_j$$

  - Bi Light from point i
  - Le Light emitted directly by cluster i
  - pi Material properties
  - n Irradiance Quality for the obj of cluster j (see later)
  - Fij Form factor (how much i can see j)
  - Lj Light from cluster j (sampled from last frame RT Lightmap)

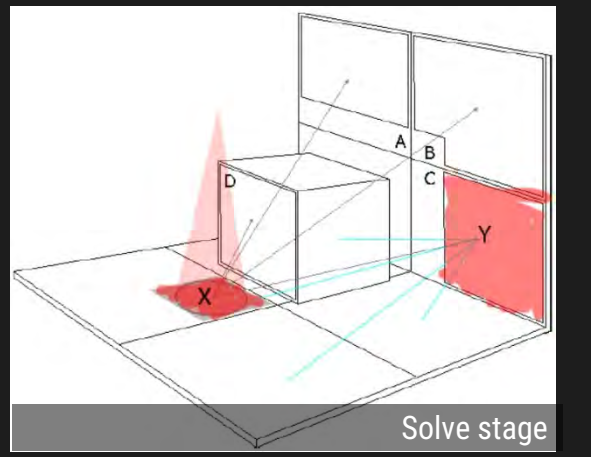
- Bounce stage reads back the values from the light maps and bounces light values back to the Clusters. This way simulates multiple light bounces
  - Light.IndirectMultiplier = Bounce Intensity



Input lighting stage



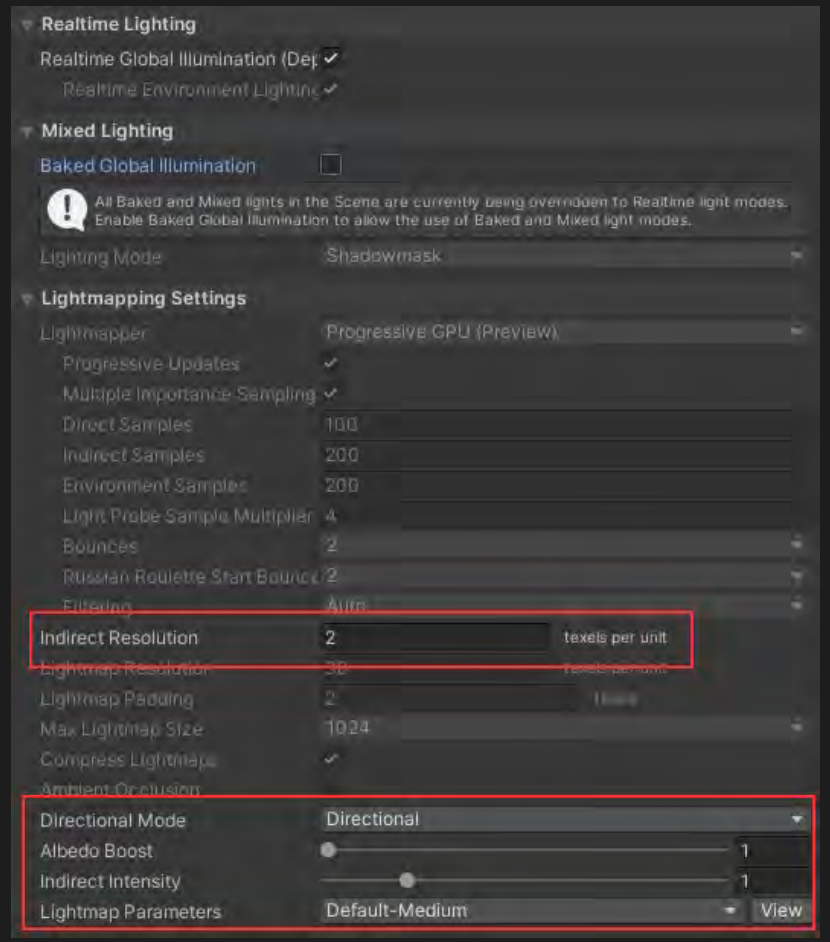
Solve stage

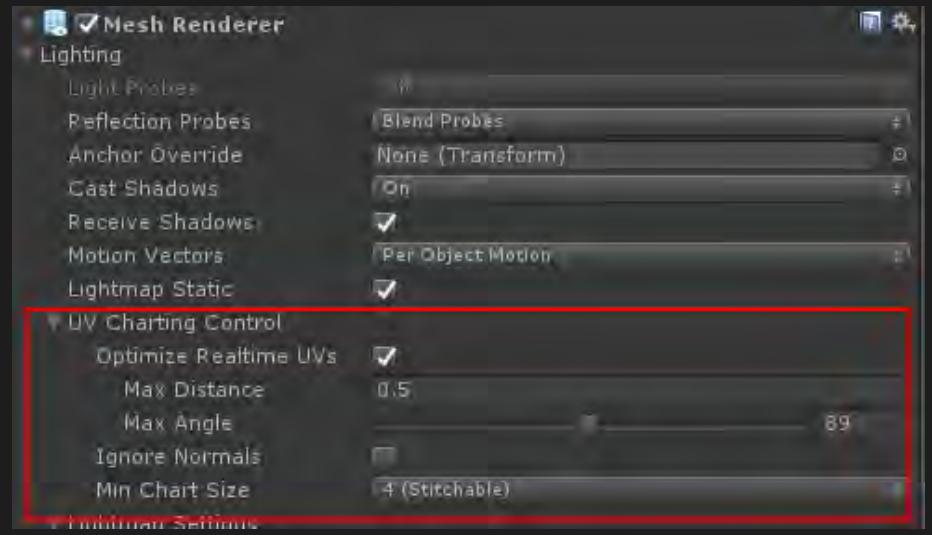


Solve stage

Realtime GI parameters

- Realtime GI parameters are common to Enlighten, Progressive CPU and Progressive GPU Lightmapper





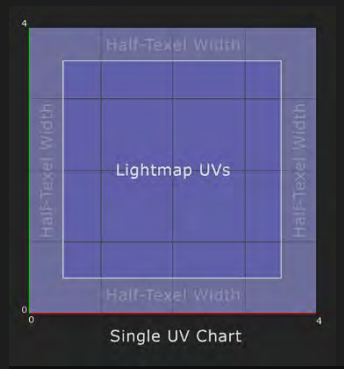
Realtime GI

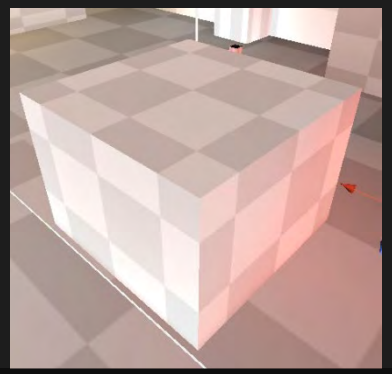
Indirect resolution number of texels per unit to use for your indirect lighting calculation (same as Lightmap resolution for Baked GI)

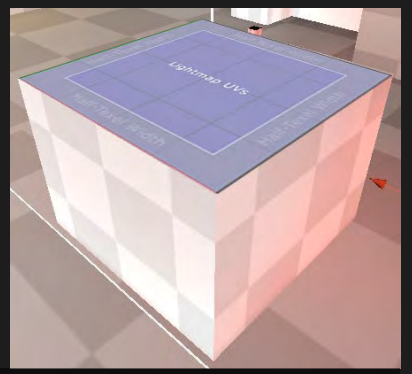
- Indoor 2-3 texels
- Outdoor 0.5-1 texels / 0.1-0.5 for terrain
- Don't go over 4 texels
  - This introduces CPU overhead, memory overhead and you start to see higher detail which also introduces more artifacts

Calculation Time
- 2 Texel = 1 min
- 10 Texels = 1 hour

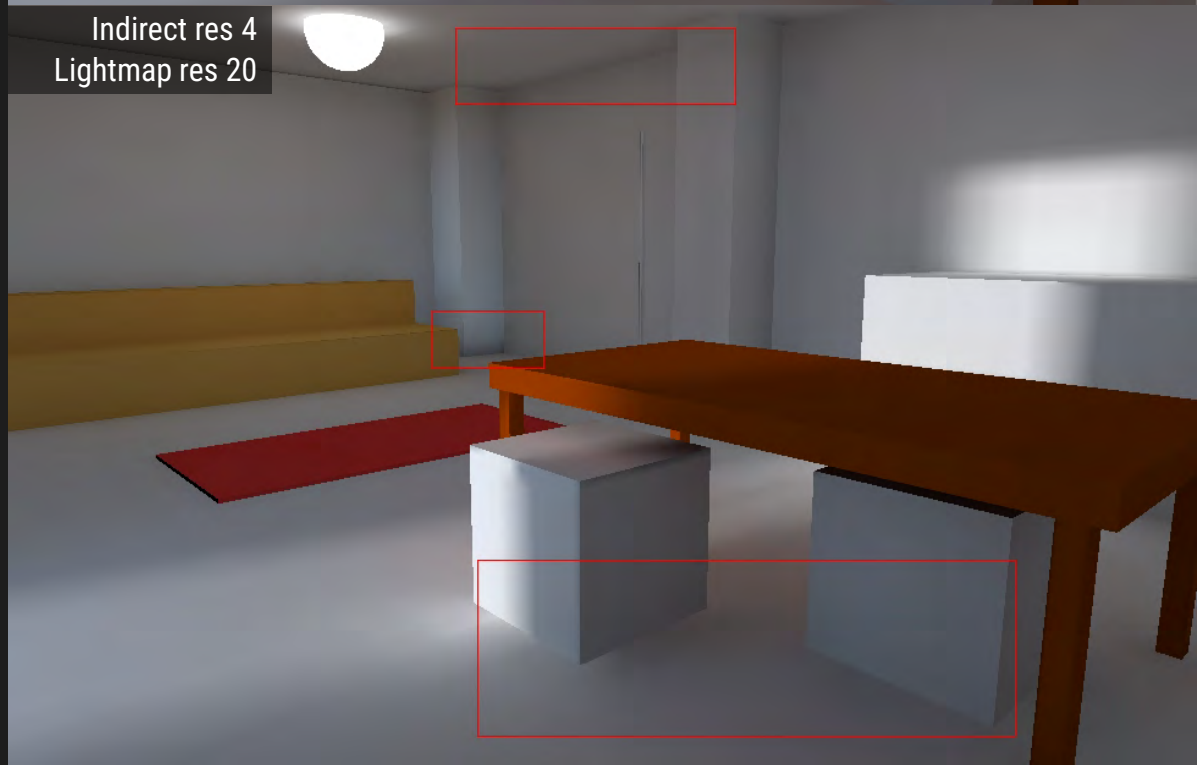
NB: Since the min chart size is 2x2 texels, on a 1 unit cube (but also on a 0.5 unit cube like the Chair obj) we'll have the same # of texels if indirect resolution is in range [1,4]







Indirect res 1 OR 2 on a 1 unit size cube



Realtime GI

Indirect intensity [Works in editor mode] The brightness of the indirect light

- Notice that the direct light intensity doesn't change, only the indirect lighting does

Realtime GI

Albedo boost [Works in editor mode] Artificially introduces more lights into your scene, increasing the number of bounce light of a surface

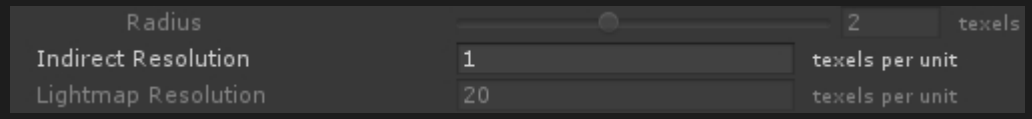
- The default value of 1 is correct; if you go over this value it is not physically correct, but sometimes it might be necessary
- Using Albedo boost, we lost shadow information (compare with Indirect intensity)
- If you want a brighter room, then adjust indirect intensity. If that it's not enough => tweak albedo boost

LightmapParameters / RT GI

Lightmap Parameters This is a global config that you apply to the entire scene

- Resolution A multiplier of your scenes Indirect resolution. E.g IndirectResolution = 2 and LightmapParameters.Resolution = 0.5 => 0.5 * 2 = you get overall 1 texels per unit for the entire scene
- Cluster resolution A multiplier of your LightmapParameters.Resolution

- Cluster size is related to a voxelization of the scene: the higher the cluster resolution, the more detail you will capture
  - If light map is not capturing variance in the indirect color bounces, consider increasing the cluster resolution (the size of the voxel may not be small enough to capture that variance)

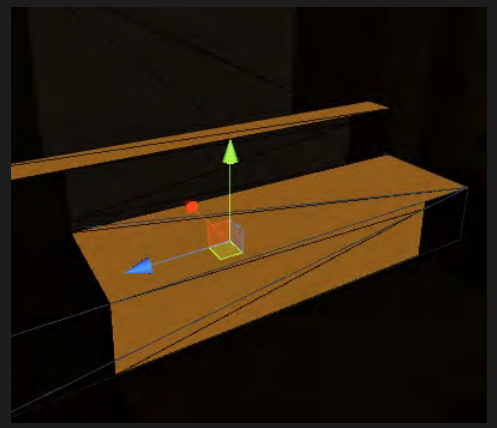
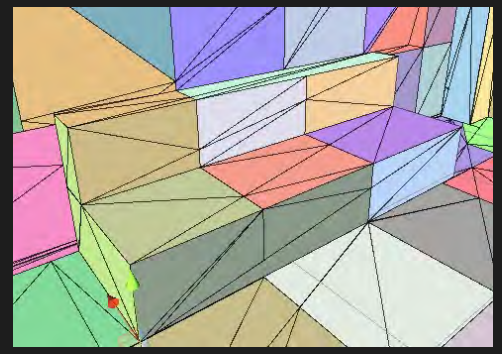


LightmapParameters / RT GI

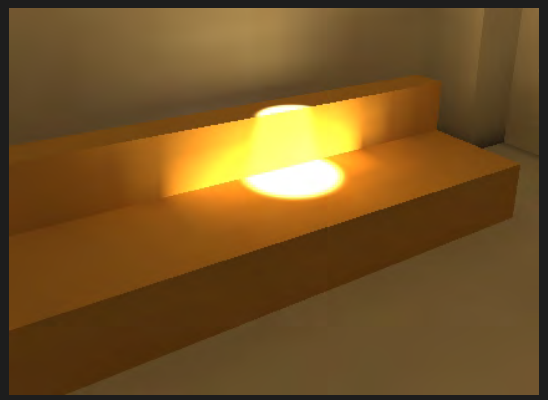
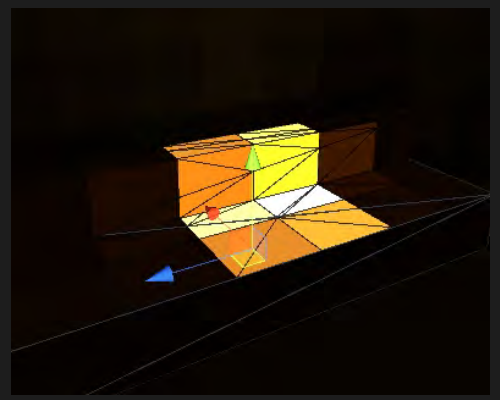
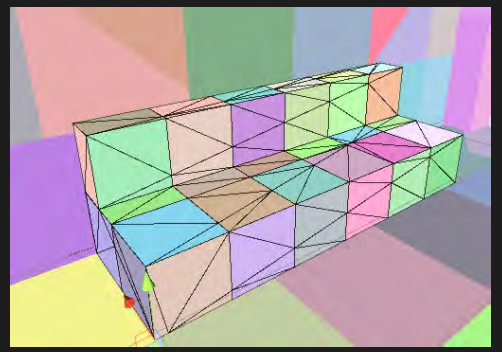
- Small cluster size (Higher resolution) lead to a higher detailed indirect light calculation

Try it

- Assign To Gobjs Sofa the LightMapParameter SofaLightmapParameterRTRes1RTCRes1
- Change ClusterResolution to 0.5: clusters should be halved, the resolution is half the previous one

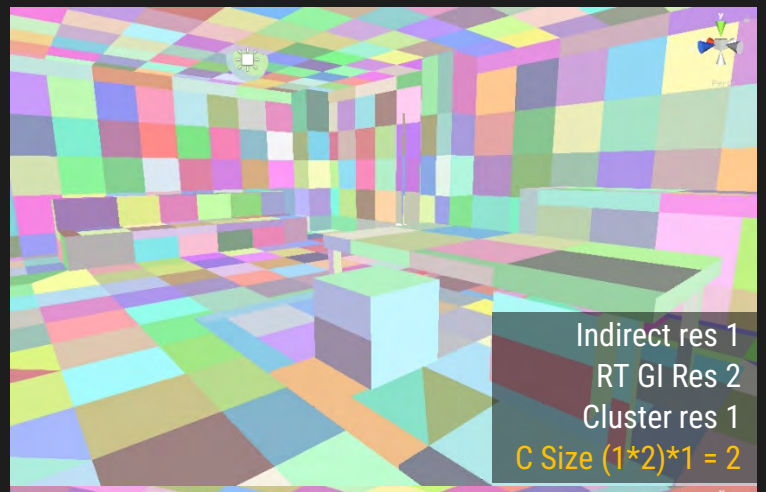
Big Cluster size (1meter = 1 cluster)
LightingPanel.IndirectResolution: 2
LightmapParameters.Resolution: 1
LightmapParameters.ClusterResolution: 0.5

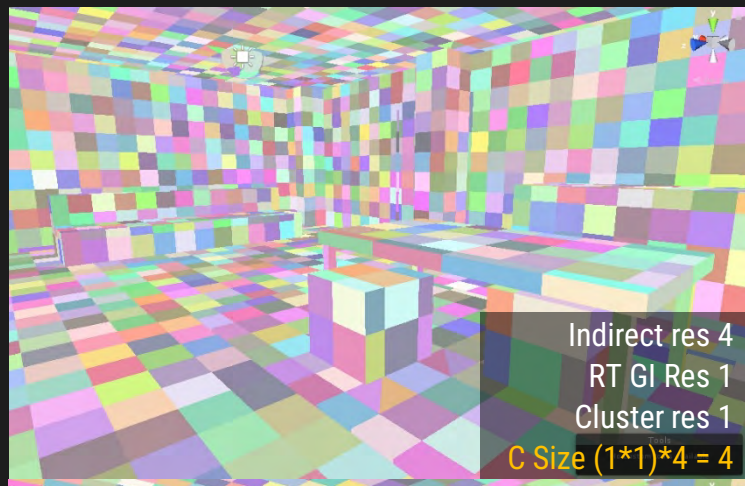




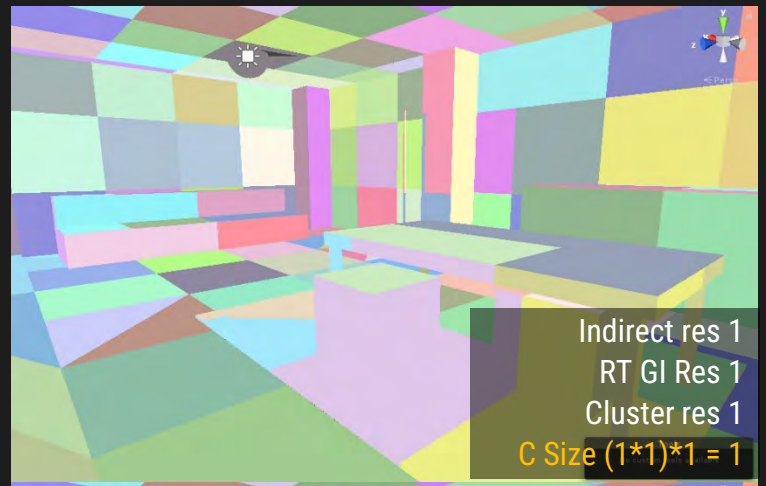
Small Cluster size (1 meter = 2 clusters)
LightingPanel.IndirectResolution: 2
LightmapParameters.Resolution: 1
LightmapParameters.ClusterResolution: 1

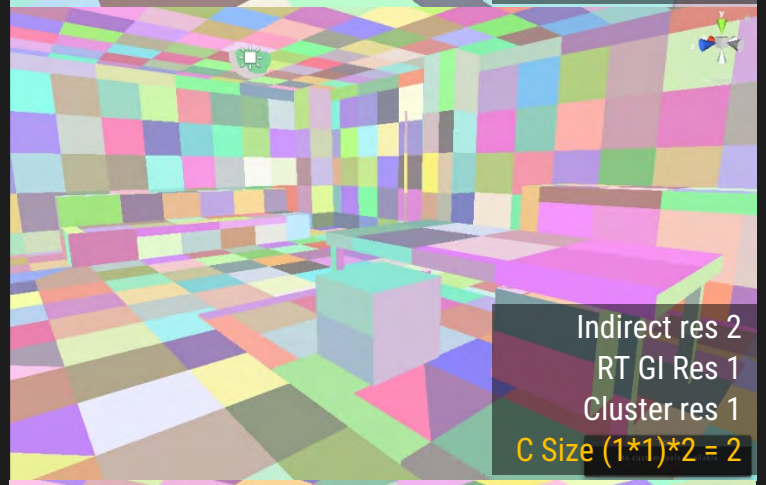
LightmapParameters RT GI

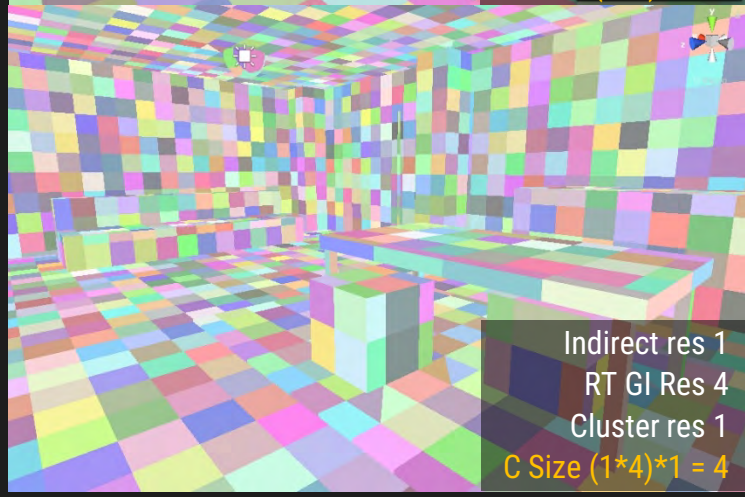
- How IndirectResolution, LightmapParameters.Resolution, LightmapParameters.ClusterResolution are related to the final cluster size

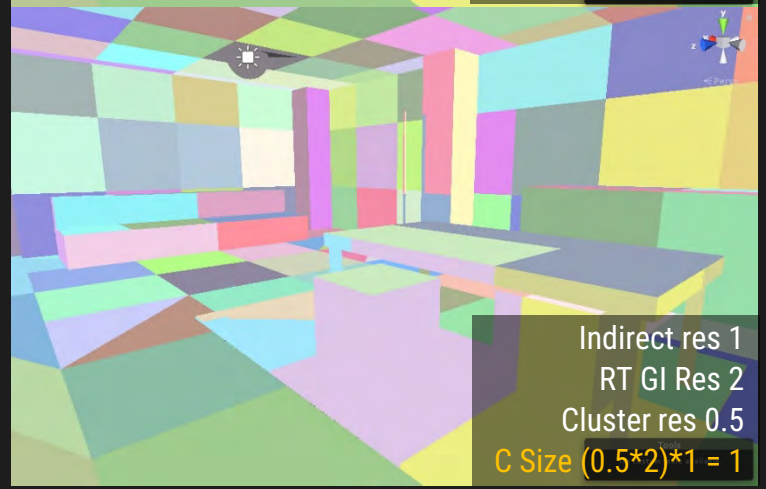


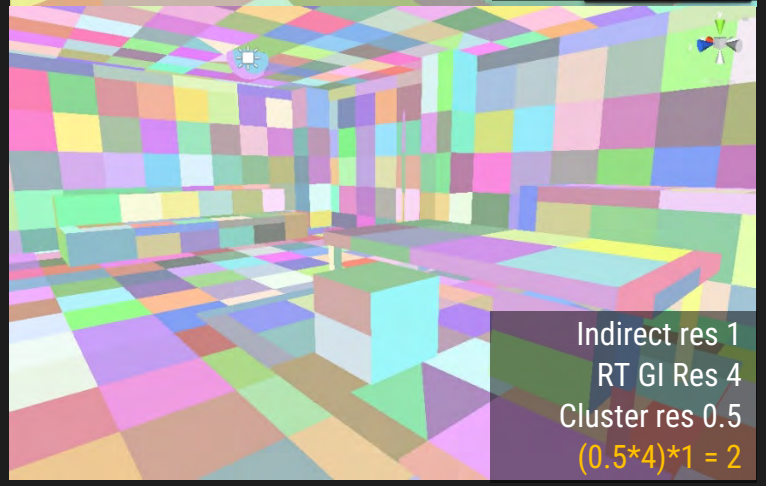












LightmapParameters / RT GI

- Irrandiance Budget The amount of memory used by each cluster when sampling the Cluster network. Determines the accuracy of the result. Higher value = more accurate GI = increase in memory usage and additional CPU overhead at runtime
- Irrandiance Quality The amount of rays the cluster is able to shoot into the scene in order to report the visibility of nearby clusters
    - E.g. If you have a pole which is a very slim object, and if your Irradiance Quality is not high enough then rays may miss this and misses the texels from the pole
- Is transparent Use this if the object has a transparent material (Fade or Cutout)
- System Tag A group of objects whose lightmap Textures are combined in the same lightmap atlas is known as a System. The Unity Editor handles this automatically (-1 value). It is sometimes useful to define separate systems yourself
    - E.g. To ensure that objects inside different rooms are grouped into one system per room, change the System Tag number to force new system and lightmap creation. The exact numeric sequence values of the tag are not significant

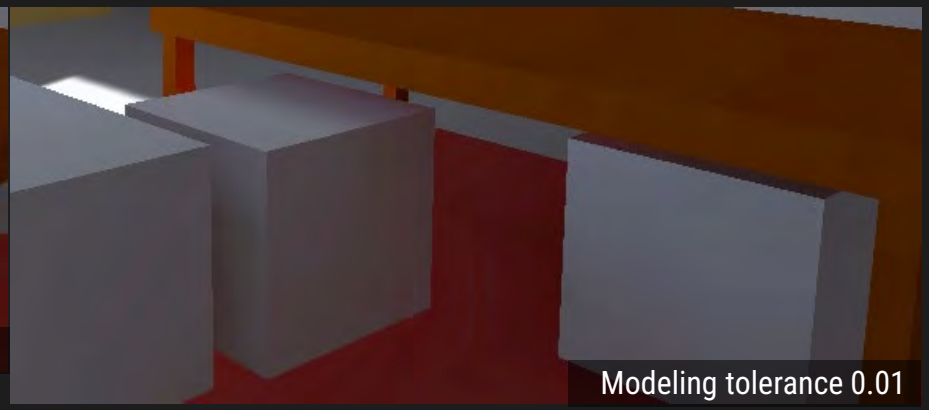
LightmapParameters / RT GI

Modeling tolerance Small values results in better indirect light calculations even if 2 surfaces are close each other

Try it

- Create a new LightMapParameters and assign it to all the 3 chairs Chair0/1/2
- Change BackFaceTolerance value to 0.3: Chairs should be black (wrong light calculation)

Modeling tolerance 0.2



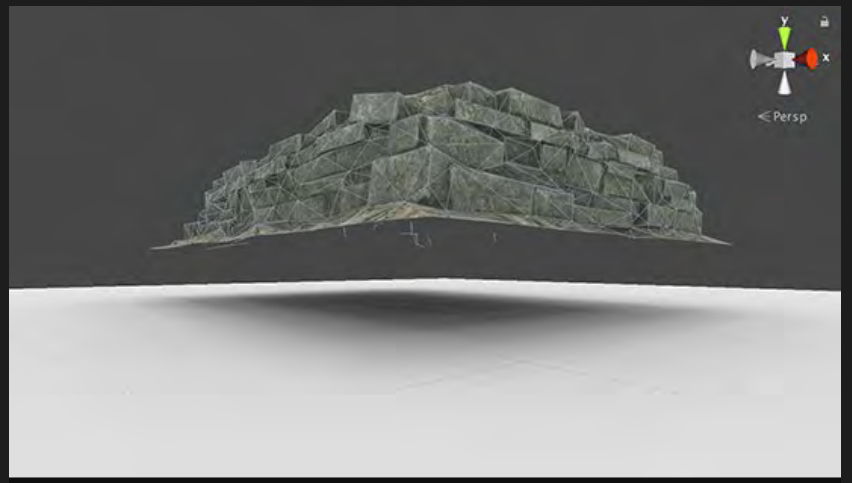
Modeling tolerance 0.01

Dark fringing artefacts such as there can often be fixed by increasing the Modelling Tolerance

LightmapParameters / General GI

BackfaceTolerance

- Backface Tolerance specifies a percentage of light that must come from front-facing geometry in order for a texel to be considered valid. If a lightmap texel fails this test, Unity will use the values of neighbouring texels to approximate a correct lighting result.
- Modifying Backface Tolerance will not have a great impact on the length of your precompute or affect run time performance. However is useful as a troubleshooting tool in situations where you have erroneous dark or light texels which remain despite increasing the Irradiance Budget



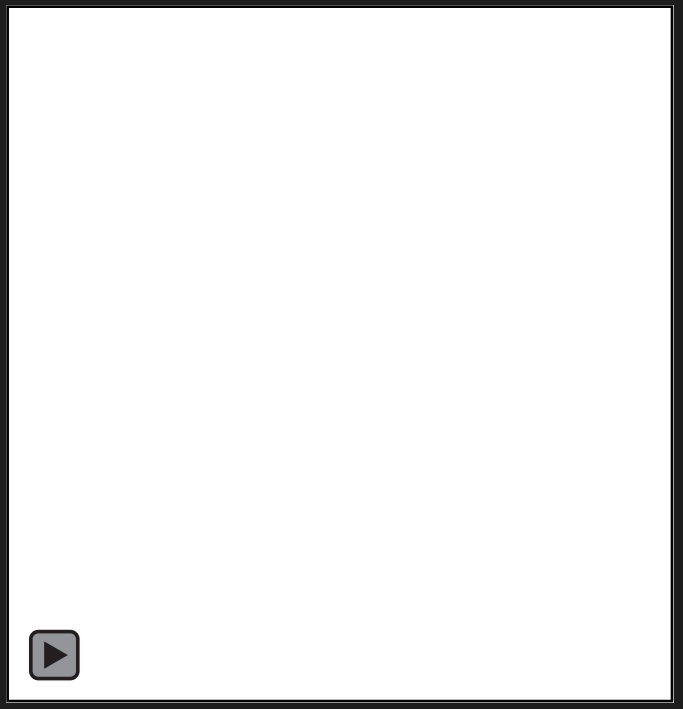
Dark lightmap texels on the floor are created by invalid backface hits during Unity's radiosity calculations. Increasing the Backface Tolerance settings can be a solution to this problem

Emissive RT GI

Try it

- Create an emissive cube CubeLight
- Set its emission to Realtime GI
- Why the emission is not working in RT GI?
- We don’t have form factors calculation yet
  - Make the cube static
  - Generate lighting
- Now try to switch off lighting in the scene and change the emission value of the cube
  - This works in realtime with RealtimeGI.UpdCol on CubeLight
    - DynamicGI.SetEmissive()
- Create an emissive cube TV
  - Import AbstractVideoArt.mp4 and use it in a VideoPlayer (activate loop)
  - This works in realtime with RendererExtensions.UpdateGIMaterials();

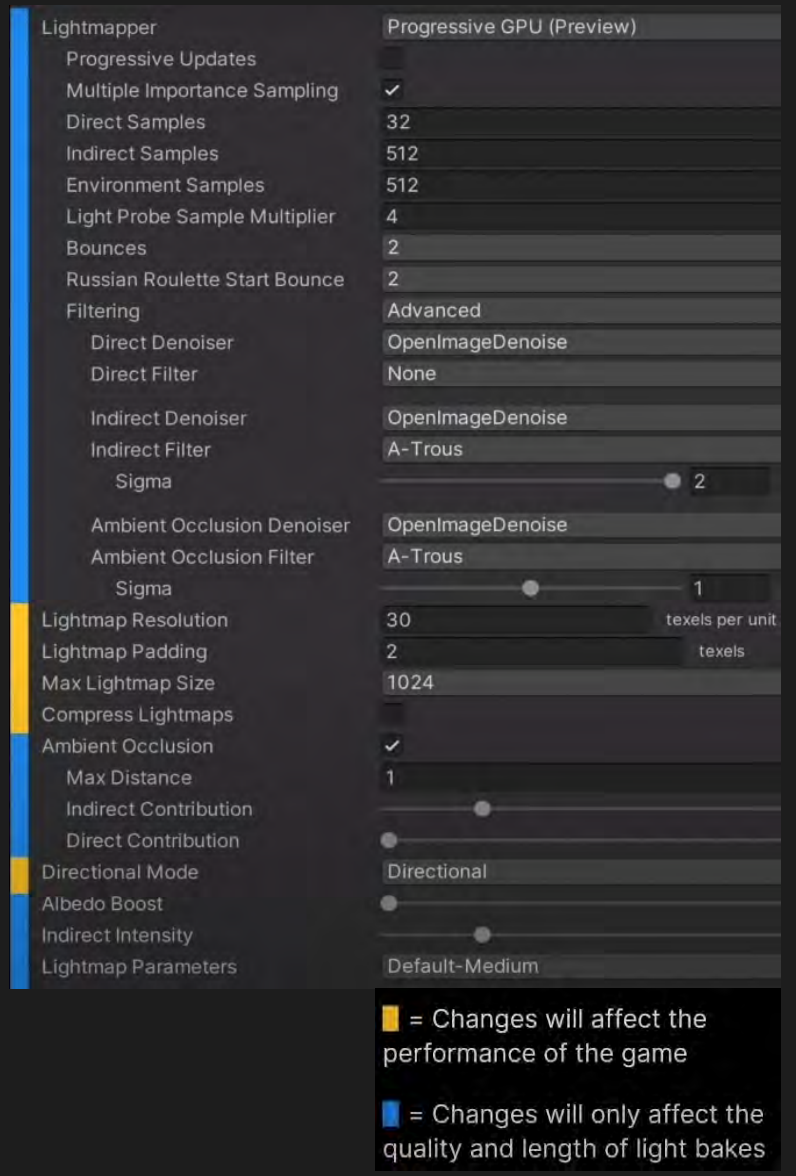
[AbstractVideoArt.mp4, RealtimeGI.cs]



Understanding the settings

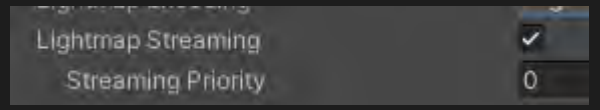
| Changes that will affect the performance of the game
- Lightmap resolution: more/less memory to be managed at realtime
- Directional lightmaps are additional maps to be sampled
- Settings to discuss with a technical artist/programmer

| Blue settings
- Once you are sure that yellow parameter values are good for your performance, you can tweak these values as long as you want



Other

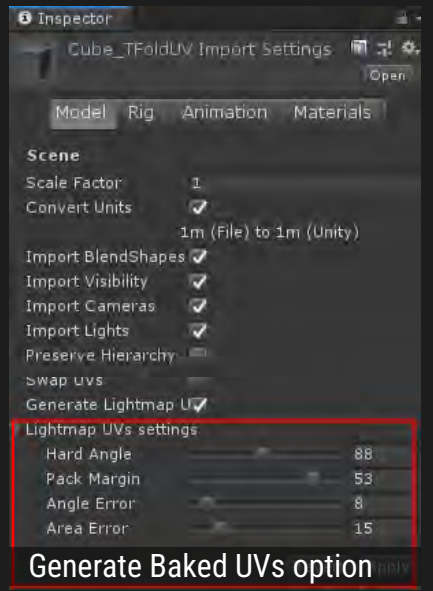
- Lightmaps are textures
  - Choose the right filter (bilinear, trilinear, anisotropic level)
  - Use Mipmaps
- ProjectSetting/Player/LightmapStreamingEnabled let you save RAM memory by loading only the Lightmap Mipmap you need



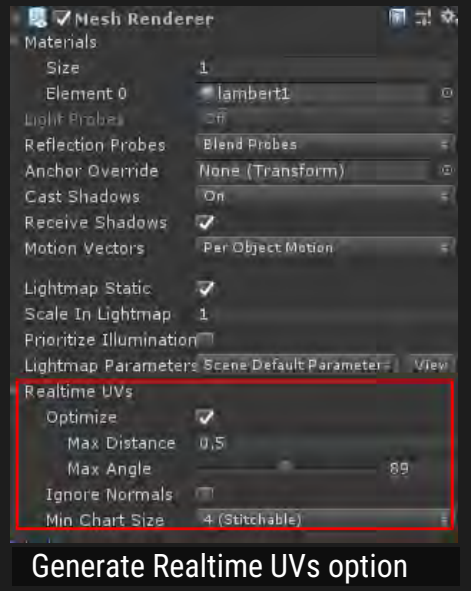
Generate Realtime UVs VS Generate Baked UVs

- If we want to let Unity generate UV1 (Baked lightmap UVs) for our GObj, we set it in the import settings
  - Once UV1 are calculated, they are scaled in the lightmap based on MeshRenderer.ScaleInLightmap parameter
  - Hence if we have a Margin of 3 texels in UVCharts with Scale in Lightmap 1, if I set Scale in Lightmap 2 there will be a Margin of 6 texels

- If we want to let Unity generate UV2 (Realtime lightmap UVs) for our GObj, we set it in the Mesh Renderer settings
  - Realtime Precompute/Repacking step guarantees a COSTANT half pixel boundary around the charts
  - Uvs are scaled dependending on the LightmapParameters.RealtimeGI.Resolution of the instance
  - Realtime UVs are per instance

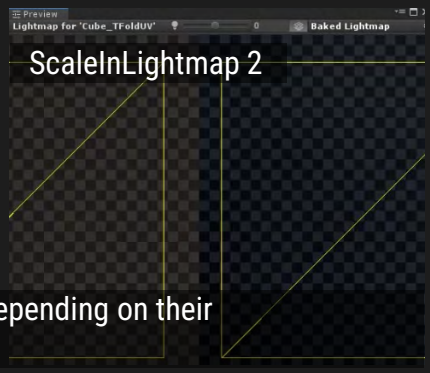


Generate Baked UVs option



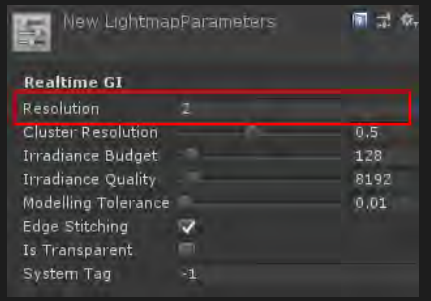
Generate Realtime UVs option

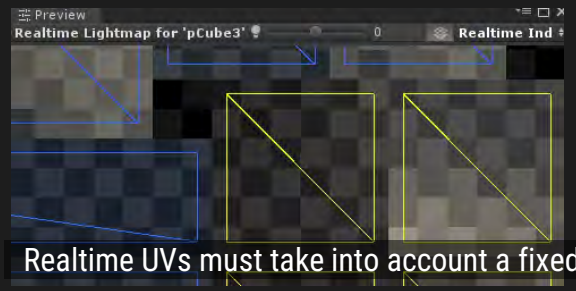
ScaleInLightmap 1

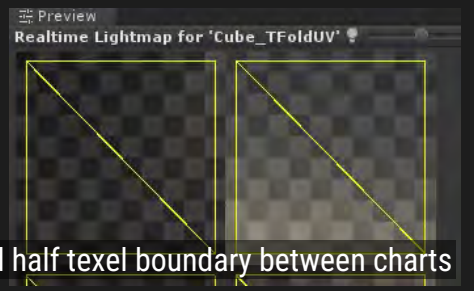


ScaleInLightmap 2

Baked UVs are simply scaled, depending on their ScaleInLightmap



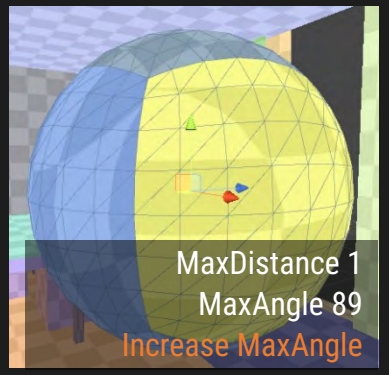


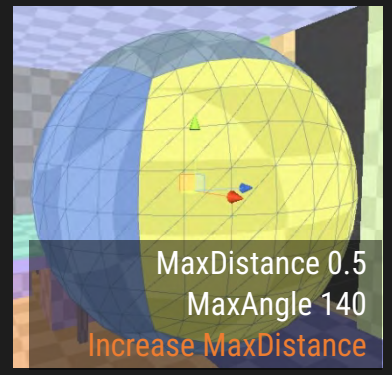


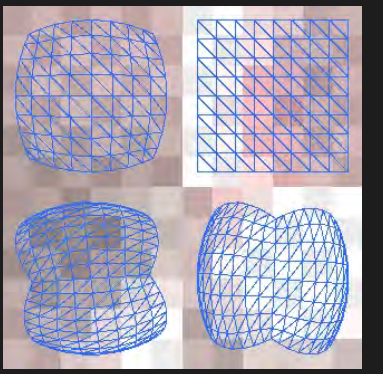
Realtime UVs must take into account a fixed half texel boundary between charts

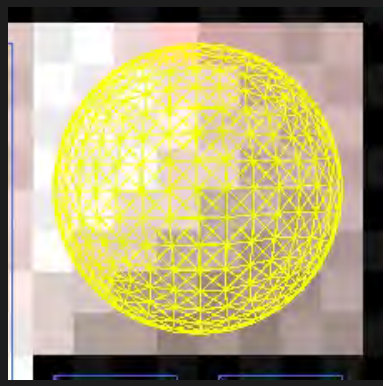
Lightmap seam stitching problem -RTGI

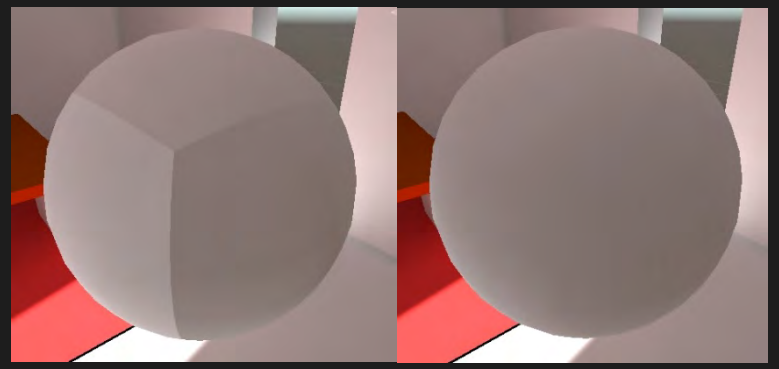
Setup
- Use the light DirectionalLight RT StitchingProblem
- Set RT GI ON / Mixed GI OFF
- Check UVCharts distribution
- Check in global LightmapParameters
  - RealtimeGI.EdgeStitching turned ON This property indicates that UV charts in the lightmap should be joined together seamlessly, to avoid unwanted visual artifacts
- In MeshRenderer OptimizeRealtimeUVs Options
  - MinChartSize 4
  - Increase MaxDistance/MaxAngle until all obj UVCharts are collapsed enough
  - IgnoreNormals ON could optimize the way the sphere Uvs are unwrapped
  - Use custom lightmap settings and Increase RealtimeGI.Resolution for the obj
- Try It
  - See the difference in MeshRenderer/LightMapParameters of RTGISphereBad/Good between RTGI Sphere Bad/Good

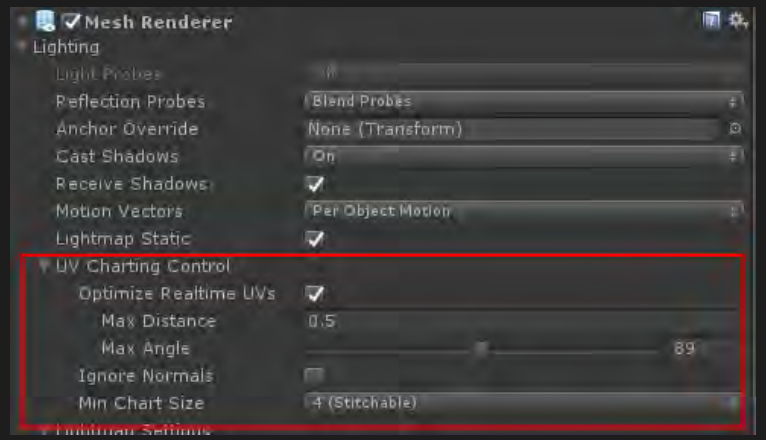












Lightmap seam stitching problem - BakedGI

Setup

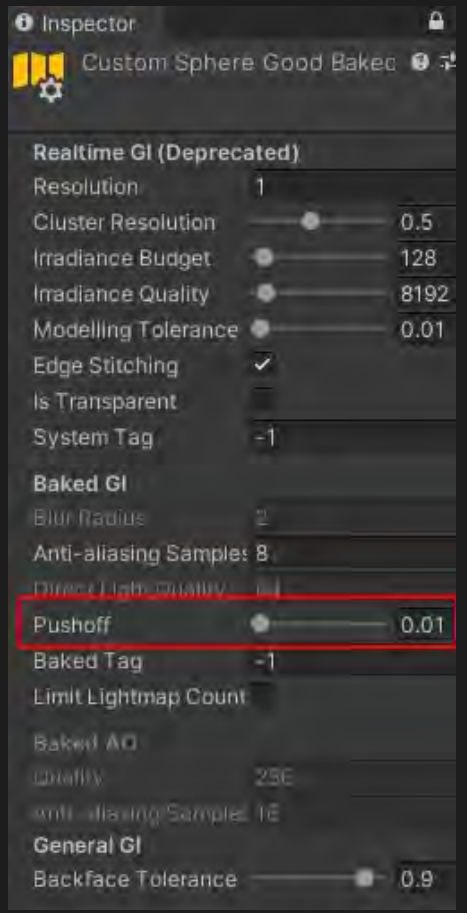
- Use the light DirectionalLight Baked StitchingProblem
- Set RT GI OFF / Mixed GI ON

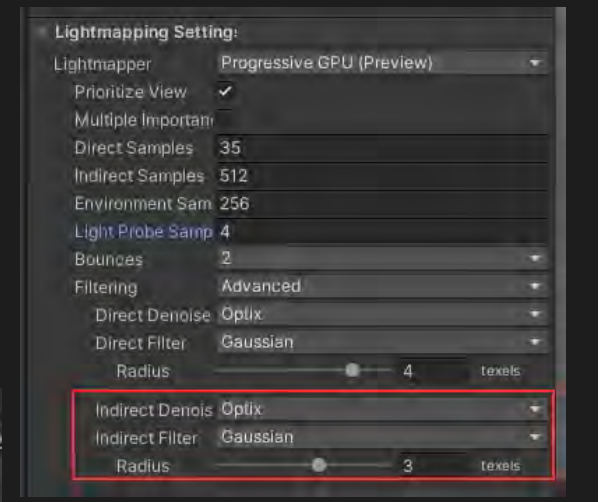
Mesh Render Settings

- MeshRenderer.StitchSeams - Turn ON this option
- Increase LightmapParameters/Pushoff The distance to push away from the surface geometry before starting to trace rays in modelling units. Useful for getting rid of unwanted AO or shadowing. Can solve problems where the surface of an object is shadowing itself, causing speckled shadow patterns to appear on the surface with no apparent source. You can also use this setting to remove unwanted artifacts on huge objects, where floating point precision isn't high enough to accurately ray-trace fine detail
- Increase Lighmapper quality settings, Direct/IndirectSamples, etc.
- If it is a 3D imported asset, try also to tweak GenerateLightmapUV/PackMargin value
- Provide your own UV Lightmap cords
- Tweak LightmappingSettings Filtering values (lower Indirect Radius value)

Try it

- Bake light with BakedGISphereBad on
  - StichSeams flag OFF, LightmapParameters Default
- Bake light again with BakedGISphereGood on
  - StichSeams flag ON, ScaleInLightMap 4, LightmapParameters CustomSphereGoodBakedGI





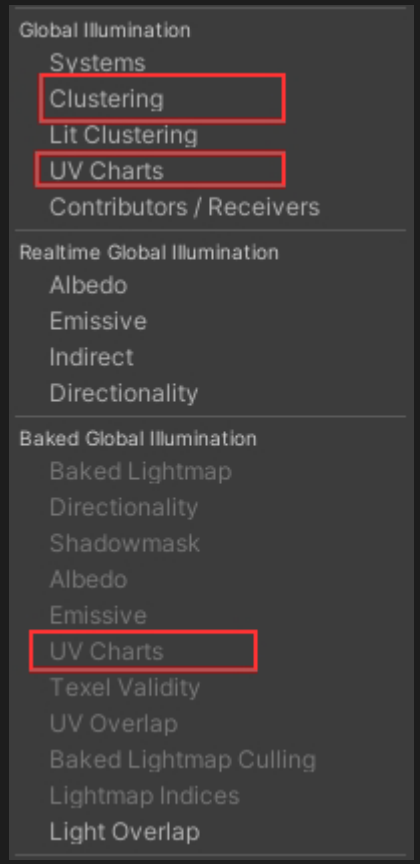
SceneView modes

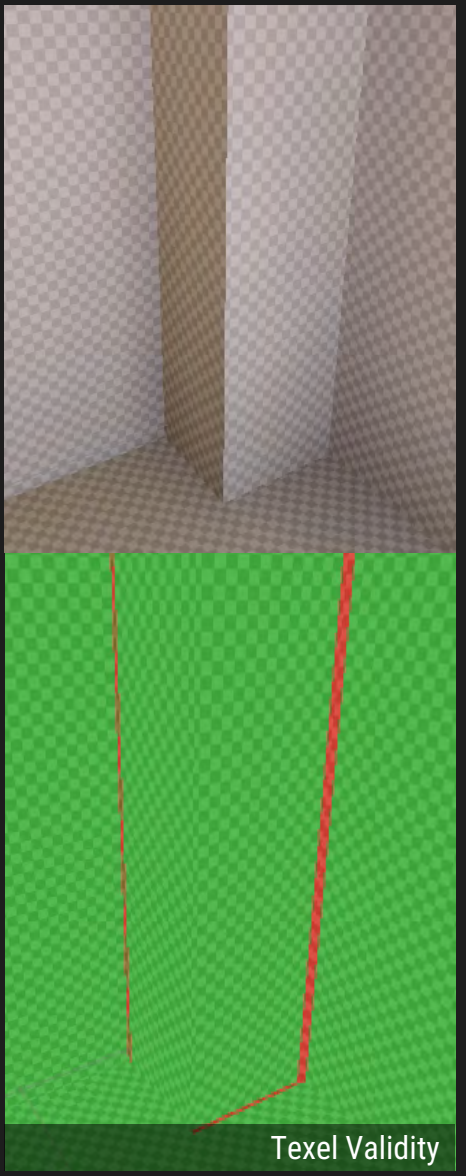
UVCharts
- RealTime – Based on UV2, not-optimized or Optimized
- Baked – Based on UV1, Imported or Auto-calculated

Clusters only for RT-GI

Systems SystemTag Is a value in LightmapParameters asset. Objs whose lightmap Textures are combined in the same lightmap atlas is known as a System. The Unity Editor handles this automatically (-1 value). It is sometimes useful to define separate systems yourself, E.g. To ensure that objects inside different rooms are grouped into one system per room, change the System Tag number to force new system and lightmap creation. The exact numeric sequence values of the tag are not significant

Texel Validity (See next slides) This mode shows which texels are marked invalid because they mostly "see" backfaces. During lightmap baking, Unity emits rays from each texel. If a significant portion of a texel's rays are hitting backface geometry, this texel is marked invalid. This is because the texel should not be able to see the backfaces in the first place. Unity handles this by replacing invalid texels with valid neighbours, otherwise there should be light leaking artifacts. You can adjust this behaviour using the Backface Tolerance parameter LightmapParameters/General GI





SceneView modes

UV Overlap If lightmap charts are too close together in UV space, the pixel values inside them might bleed into one another when the lightmap is sampled by the GPU. This can lead to unexpected artifacts

Indirect useful tool when evaluating Scene lighting as it shows your lighting results without the distraction of materials or non-Static objects which do not contribute to your Precomputed Realtime GI

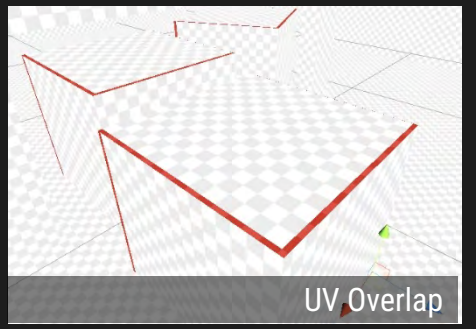
Albedo Shows at what detail Enlighten uses the color information from your scene

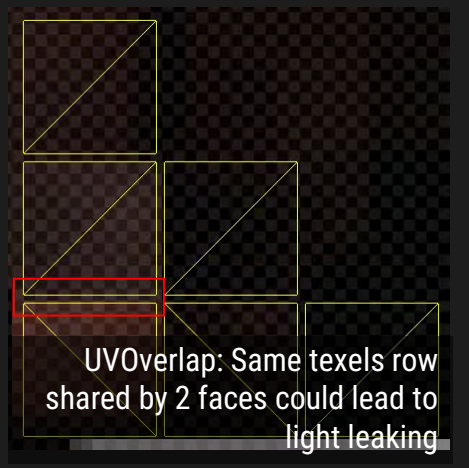
Emissive Color and intensity of the emitted light from emissive objects

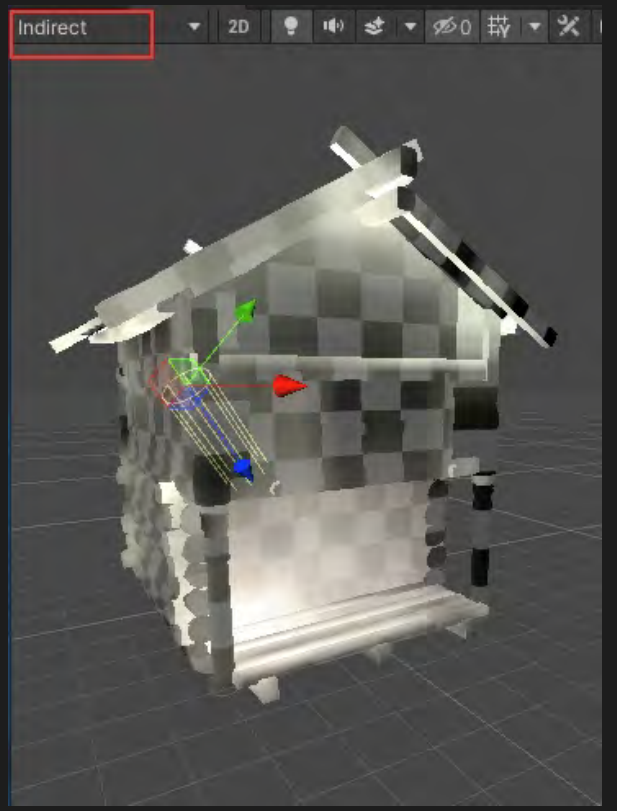
- Tip: we can use emissive objects with Enlighten to create area lights without any rendering cost

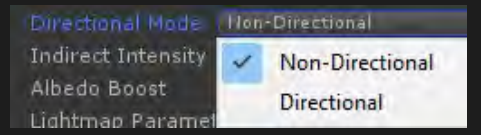
LightOverlap Useful in mixed Shodowmask lighting (see next slides)

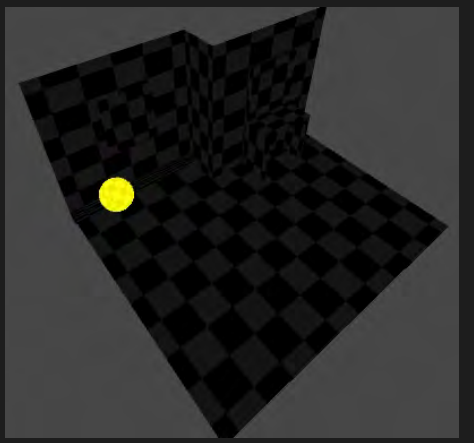
LightMapIndices What is the lightmap linked with this object? (Only if you have more than one lightmap in Lighting/BakedLightmaps)


UV Overlap


UVOverlap: Same texels row shared by 2 faces could lead to light leaking



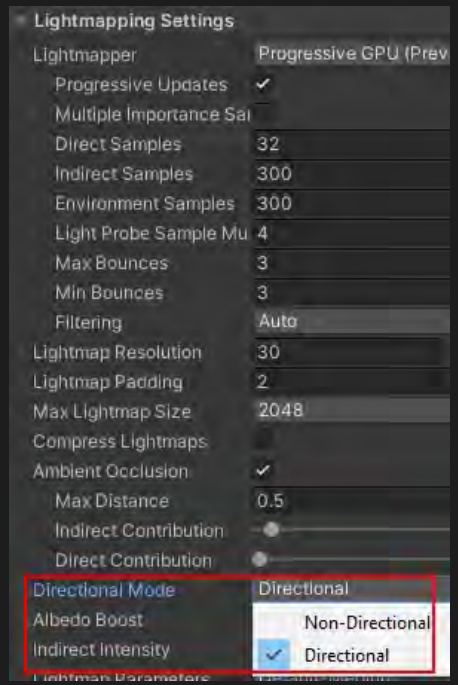


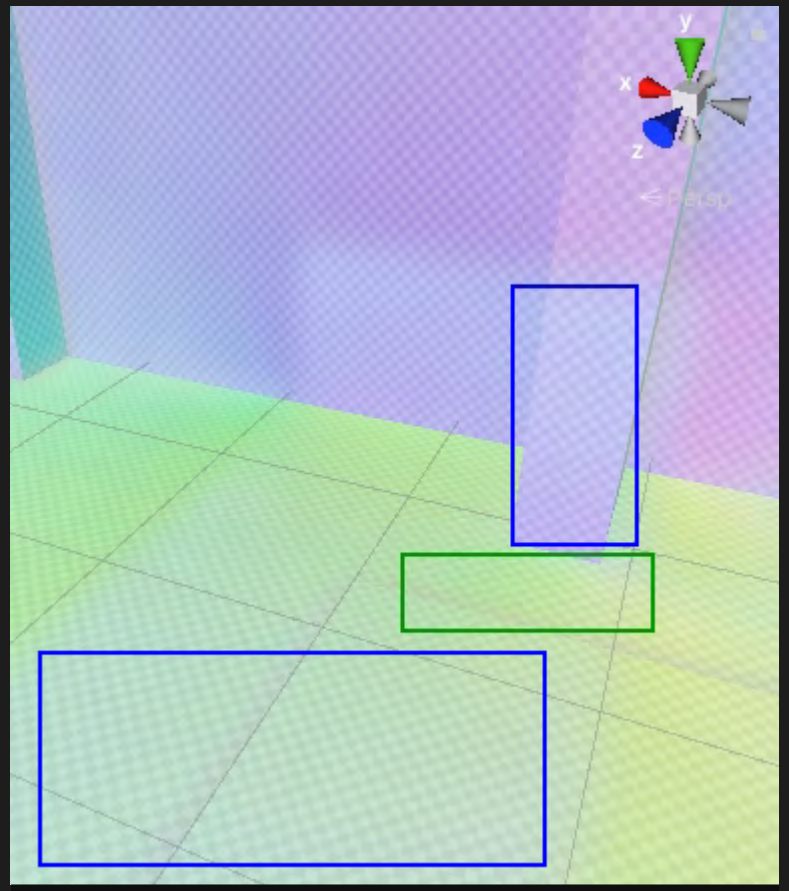


SceneView modes

Directionality Dominant light direction for each pixel

- Active if LightmapSettings/DirectionMode is Directional
- Used if the GI Shader use normal map info
- Adds a lightmap texture to store direction info





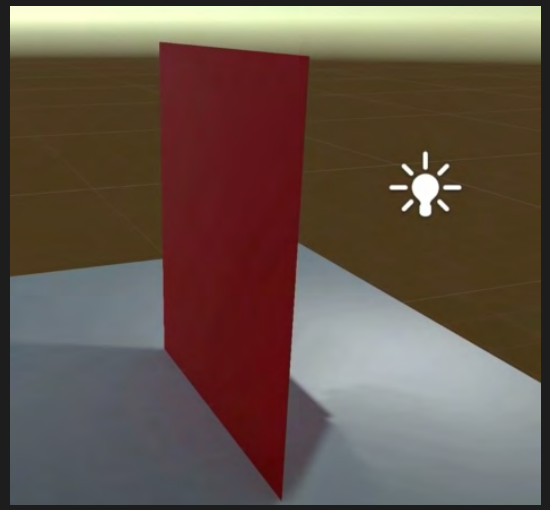
Bluish areas: they are receiving more direct light from the sun
Greenish areas: floor is receiving reflected light from the walls and columns

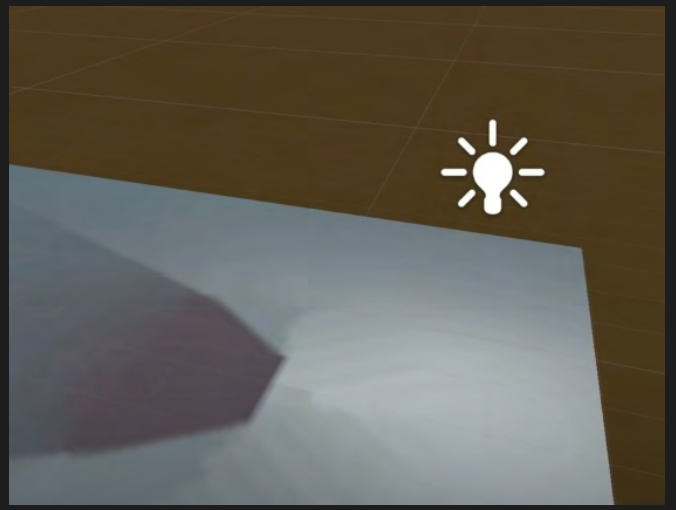
SceneView modes

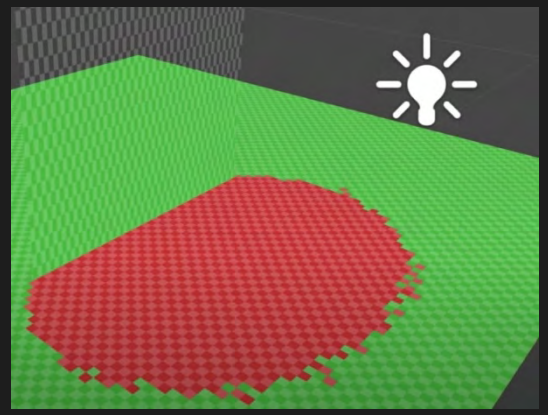
TexelValidity

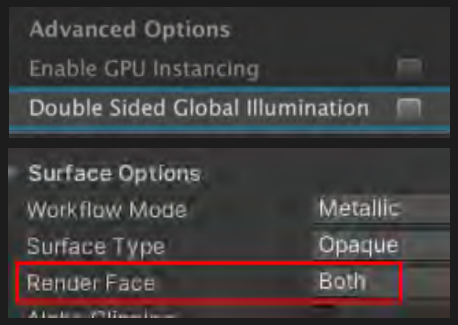
- Activate only light TexelValidity_PointLight
- If we have a flat surface and a lightsource on the backside, we might have some artifacts on the back side floor
- This is because back side floor texels are invalid (Texel validity scene view)
- To solve this (according to Unity version you're using)
  - activate doubleSideGlobalIllumination flag in the material Advanced options
  - Switch SurfaceOptions RenderFace/Both

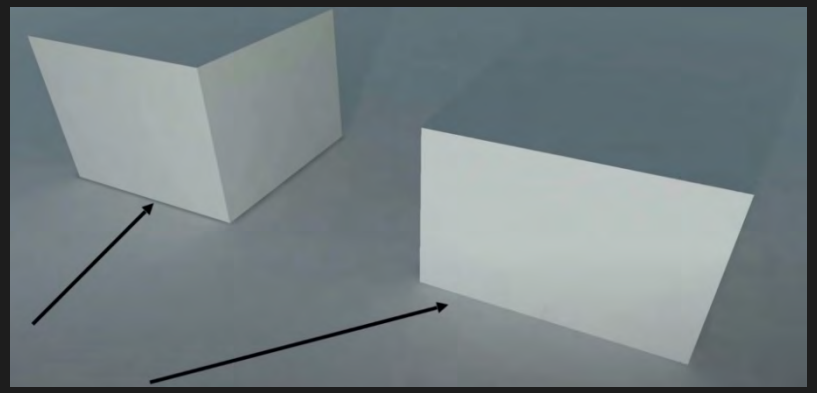
- Why don't leave this ON by default?
  - In some cases texels are under geometry, and if this flag is on, texels that shows outside the geometry might appear darker, generating a "floating mesh" effect

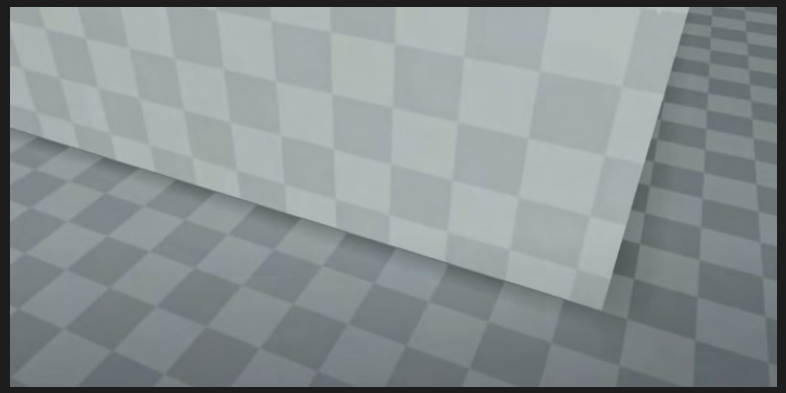


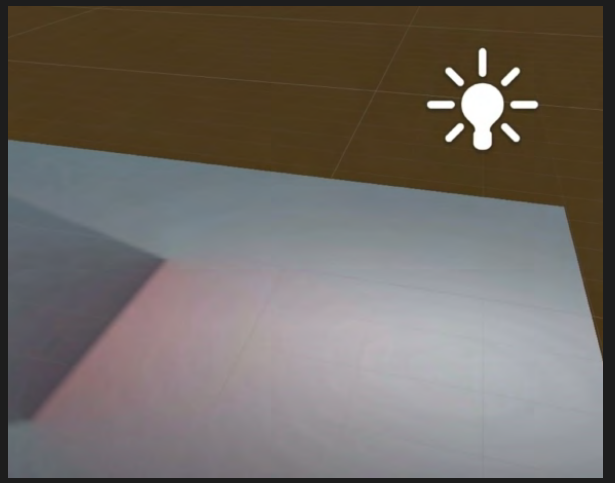




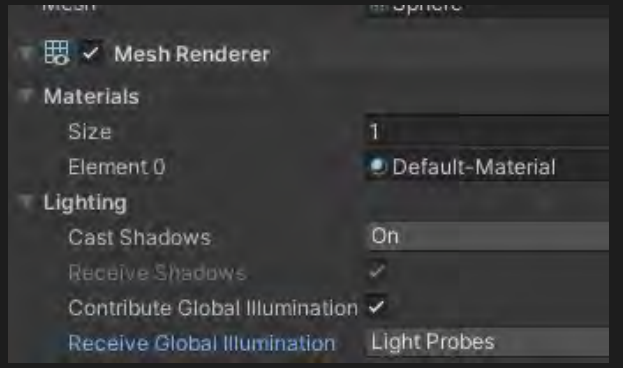






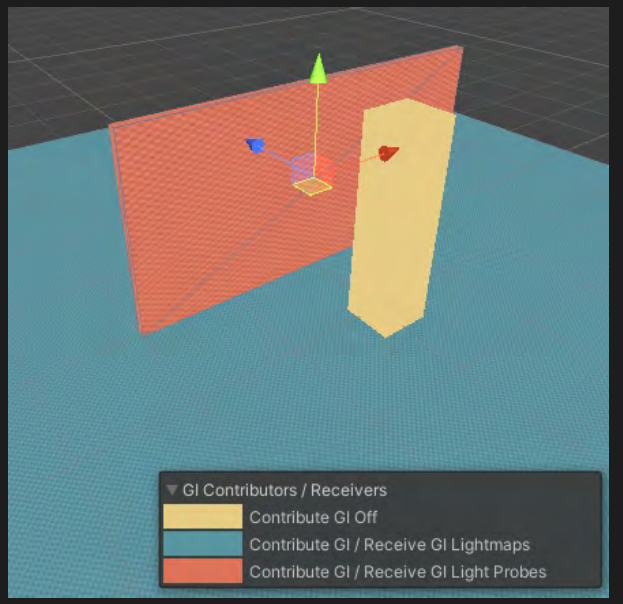


Contributors/Receivers



MeshRenderer properties

- An object can contribute to the GI or not
- If it doesn't contribute to GI
  - ContributeGI static flag is OFF
- [Built-in RP] A fast way to check contributors/receivers configuration is use Contributors/Receivers Scene visualization mode



- Otherwise, we can choose if
  - The mesh receives GI from lightmaps ReceiveGlobalIllumination/Lightmaps
  - The mesh receives GI from lightprobes ReceiveGlobalIllumination/Lightprobes
    - NB: One static object can contribute to GI but at the same time can receive GI from Lightprobes, to not waste lightmap size!
      - Before 2018.2, this effect was achieved with ScaleInLightmap 0

- [BIRP] MeshRenderer.PrioritizeIllumination should be ON every time we have a strongly emissive material and we are sure that other objects will be illuminated by this object

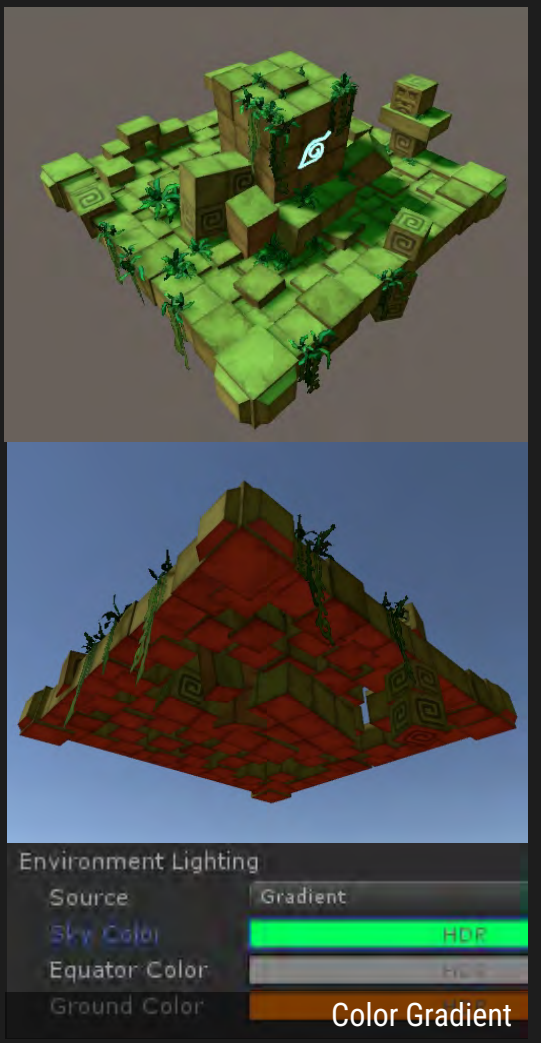
Environment Light

- Even if we don't use RealtimeGI or BakedGI, we need to use Lighting/GenerateLighting button to calculate Environment contribution into the scene

[EnvironmentLighting_06]


Color Black


Color Grey



Color Gradient

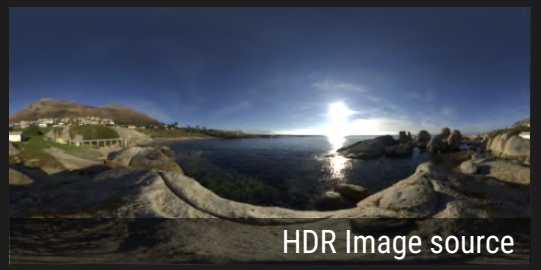
ImageBased Lighting

- 1999: Paul Debevec presented a lighting technique called Image-based lighting
- We need an HDR 360 image: instead to have 1 exposure for each photo, several exposures are taken
  - File extensions: HDR, EXR (created by Industrial Light Magic)
  - These additional exposures images are used by the renderer to know where the light comes from

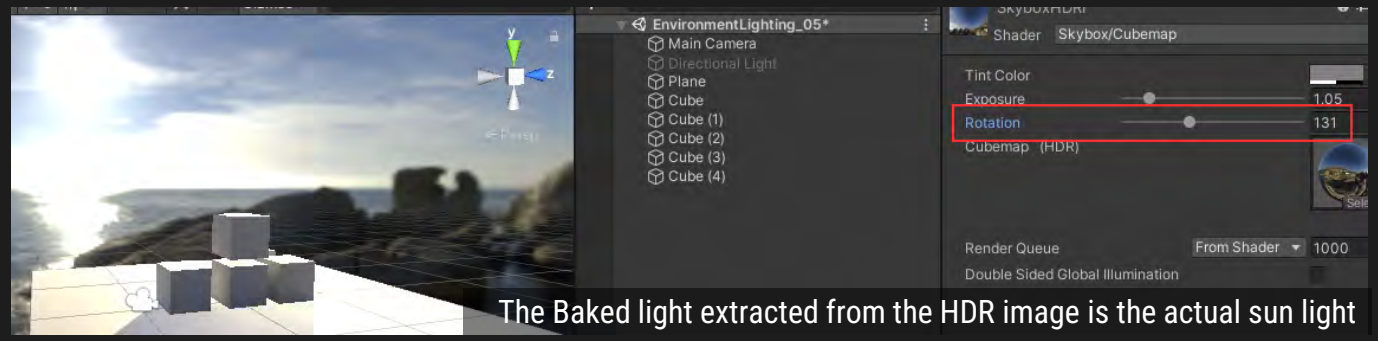
ImageBased lighting setup

- Import 360 photo / Set TextureShape to Cube
  - [simons_town_rocks_1k] in the scene example
- Create Skybox material / Shader: Skybox/Cubemap [skyboxHDRI]
- Add skybox to Environment lighting
- Turn off all scene directional lights
- Bake

[EnvironmentLighting_06]



HDR Image source



The Baked light extracted from the HDR image is the actual sun light

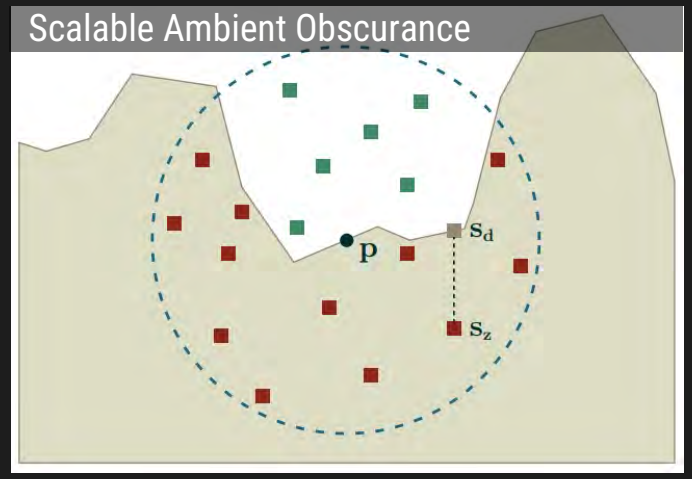
Ambient Occlusion

Baked AO

- MaxDistance How far the rays are traced before they are terminated
- Indirect/Direct how much the AO affects indirect/direct light

Only Environment

Environment +
AO (Max Distance 0.2)

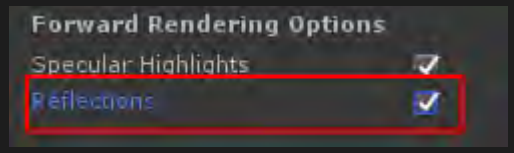


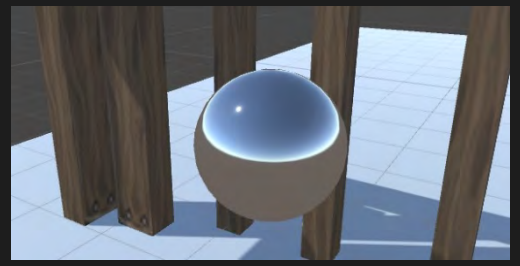
Reflections

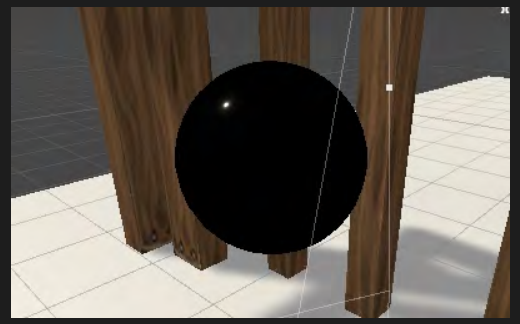
- Create a new material Reflections and apply it to Start/Sphere:
  - Smoothness 1, Metallic 1
  - [BIRP] Material/ForwardRenderingOptions/Reflections ON
  - [URP] Advanced/EvironmentReflections ON
  - No light is absorbed: Sphere is lit by Environment Light (cubemap) + Directional Light If you remove Skybox and Clear baked data, the sphere turns Black (Only directional light)
- Activate Stage: no changes. We need to bake the environment surrounding the sphere into a cubemap
  - To bake the light around a point we use LightProbes. to bake reflections around a point, we use ReflectionProbes

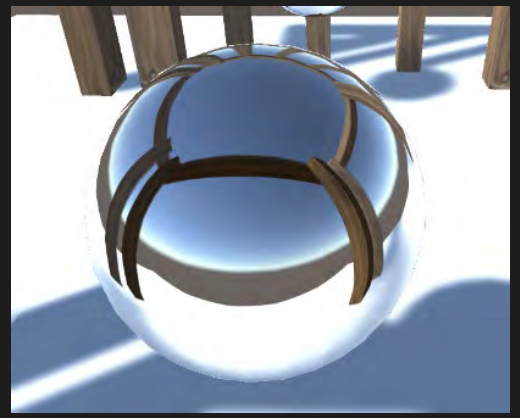
- To create a Reflection probe
  - RightClick/Light/ReflectionProbe
  - A 10x10x10 box appears
    - The reflection probe captures the environment by rendering a cube map. This means that it renders the scene six times, once per face of the cube
    - What is the meaning of the reflection probe box? Every reflective material inside that box will use the reflection probe cubemap!
  - Click Bake: it will bake all ReflectionProbe static objects. Now the sphere reflections are correct
    - A .exr file is created in a folder with the same name of Lighting/LightingSettings filename
  - Full metal materials are supposed to be fully reflective: no shadows on them (just like in real life, you can't cast a shadow on a mirror)

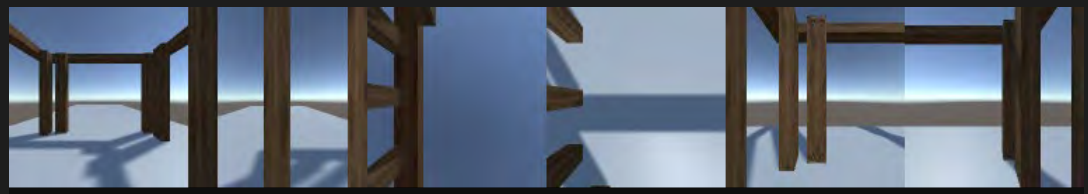
[Reflections_02]











The generated Reflection probe (.EXR file in Scene floder)

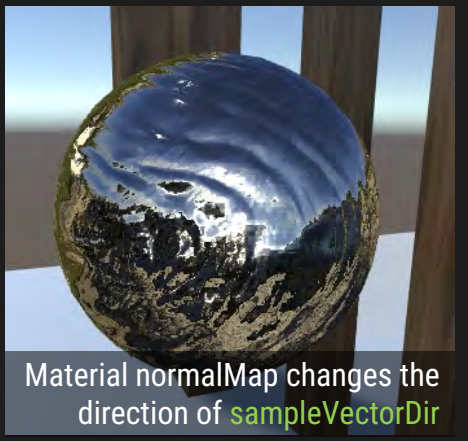
Reflection Probes

Type
- Baked Reflected objs must be ReflectionProbeStatic
- Custom Provide your own Cubemap (Eg. Portal to another environment)
- Realtime Reflected objs can be dynamic

- HDR Should High Dynamic Range rendering be enabled for the cubemap (save in PNG, EXR)?
- CullingMask What objs to consider in the reflection map (avoid small obects)
- Resolution

- The ReflectionProbe use the viewing direction and the mesh normal to calculate the sample point on the cubemap
  - sampleVectorDir = reflect(-viewDir, VertexMeshNormal)

- Generates blurred mipmaps to simulate reflections on dirty surfaces
- Can use Normal Maps
- Also dielectric materials with high Smoothness have reflections (metallic = 0)



Material normalMap changes the direction of sampleVectorDir

Reflections on dielectric materials

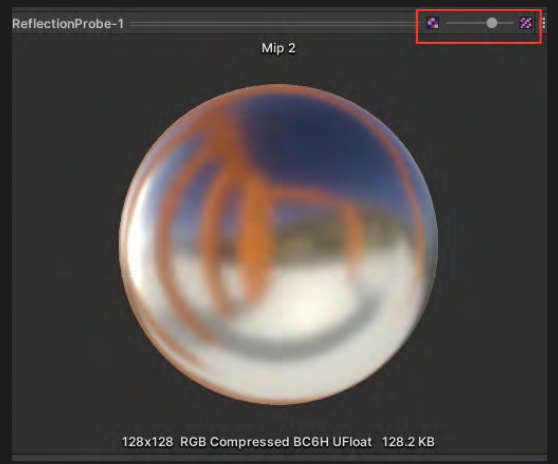
RP Realtime

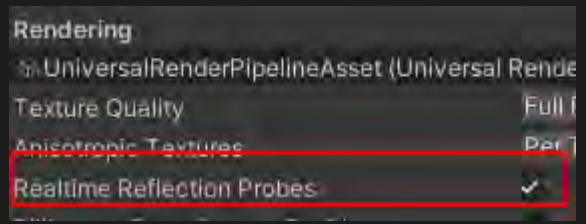
ProjectSettings/Quality/Rendering/Realtime ReflectionProbees
Realtime RefreshMode

- OnAwake() Useful if the scene is set up at run-time but does not change during its lifetime
- EveryFrame
- ViaScripting
  - ReflectionProbe.RenderProbe()
- Timeslicing
  - AllFacesAtOnce 6 cubemap faces rendered on the same frame, then blurring operation using mipmaps will follow (full updt in 9 frames)
    - Select the reflection probe and mode the Mipmap slider to see blurred mipmaps
  - IndividualFaces 1 cubemap face per frame (full updt in 14 frames)
  - NoTimeSlicing full updt in 1 frame. Very expensive!

Try it

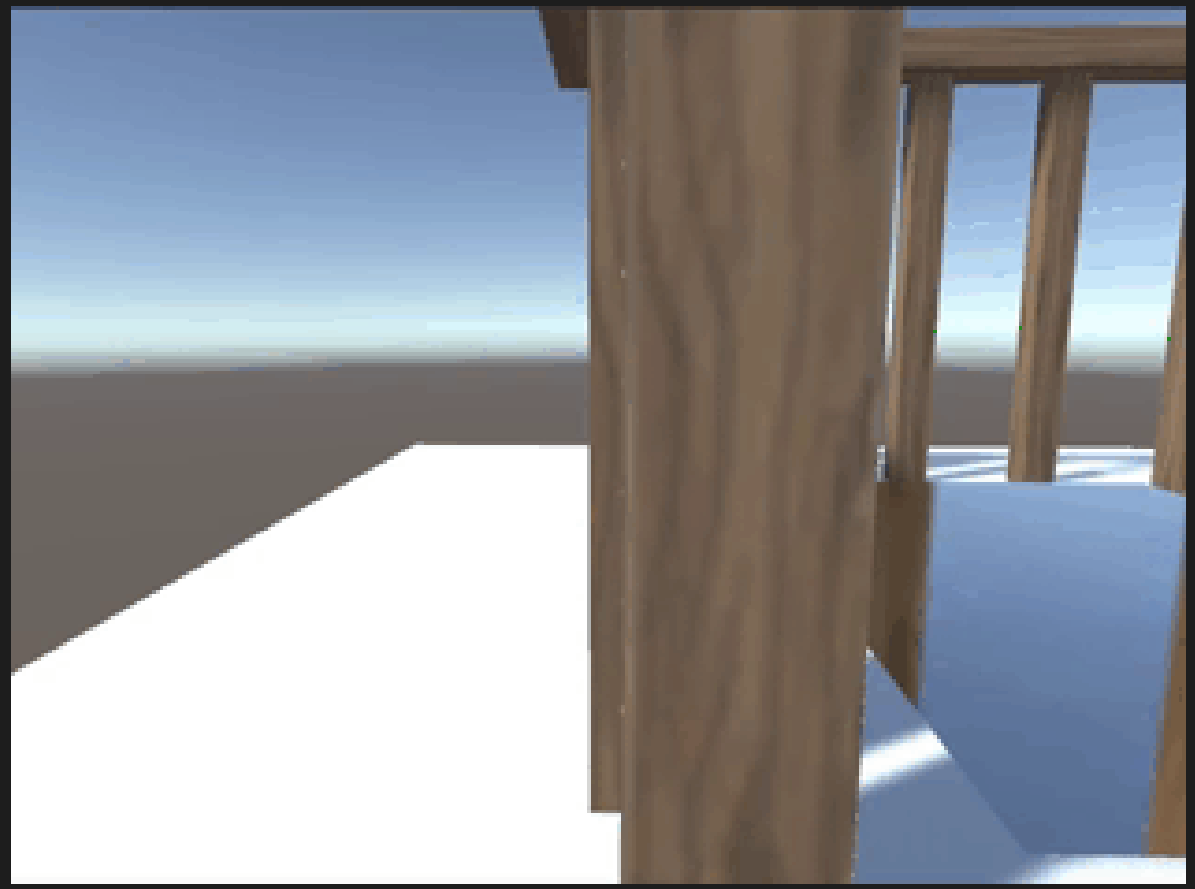
- [URP] Clear lightData before to try, otherwise the sphere will reflect also the prev baked RProbe

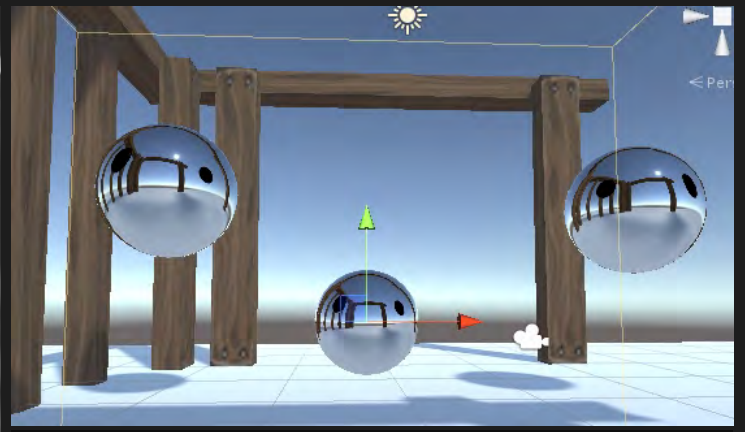




RP Box Projections

- [BoxProjection/2Spheres1PB] The same reflection probe may not work for different static objects with different scene positions!
- How to correctly setup Baked Planar reflections?
- [BoxProjection/GlassFloor] and try to setup a planar reflection
- Something is wrong. Why?



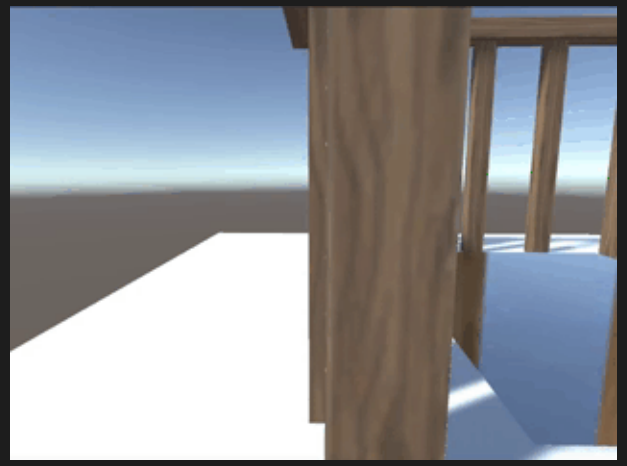


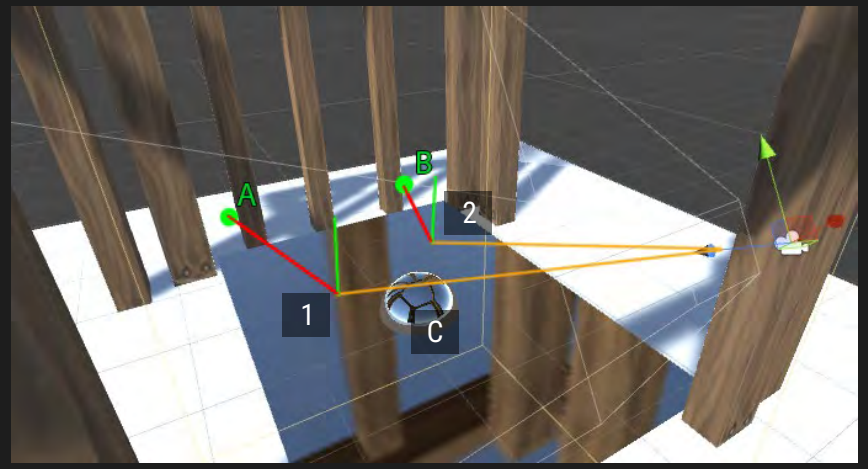
2 Static Spheres. 1 Reflection Probe

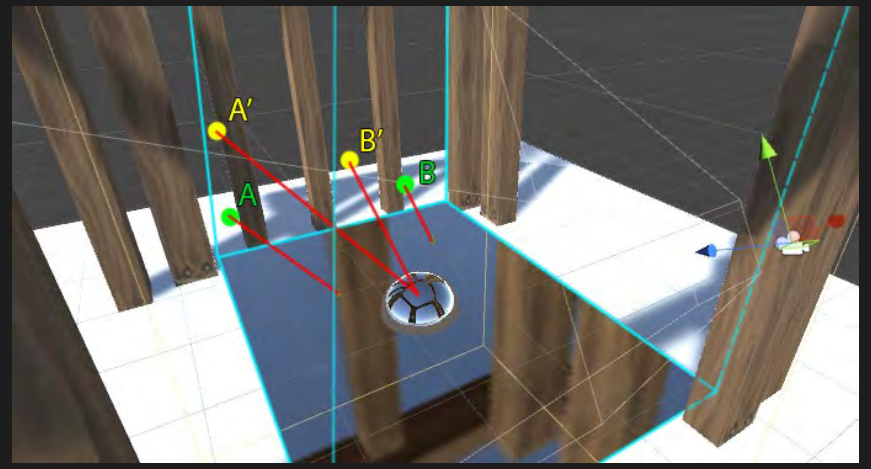
RP / Box Projections

- How to correctly setup Baked Planar reflections?
- To calculate the color on the 2 points I'm looking at [1,2], we know that the reflective floor perform a sample on the Reflection CubeMap from the reflection probe center [C] sampleVectorDir [red]
  - sampleVectorDir = reflect(-viewDir, VertexMeshNormal)
  - Where viewDir is yellow, VertexMeshNormal is green, sampleVectorDir is red in fig [X]
    - These vectors would be the correct calculation to perform. BUT the final color of [1,2] is [A', B'], NOT [A,B]! This is because the sampleVectorDir starts from the reflectionprobe pos, not from the point hitted by -viewDir!
    - This is obviously correct if we are in a sphere.. Hence the fact that we didn't notice this problem previously

- SOLUTION: sample the cubemap from the Reflection probe center [C] using C-A and C-B vectors as sampleVectorDir
  - In this way the final color will be [A,B]

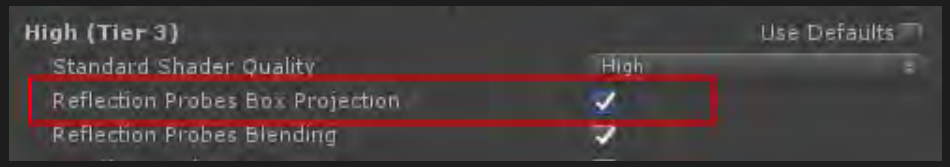


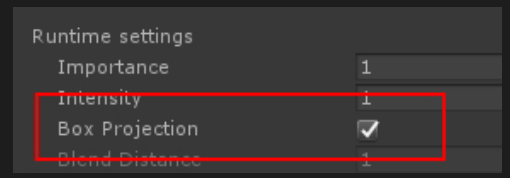


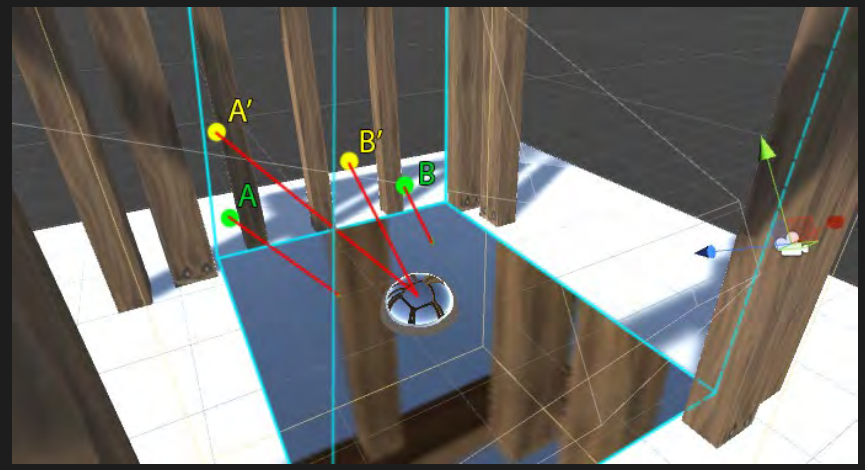


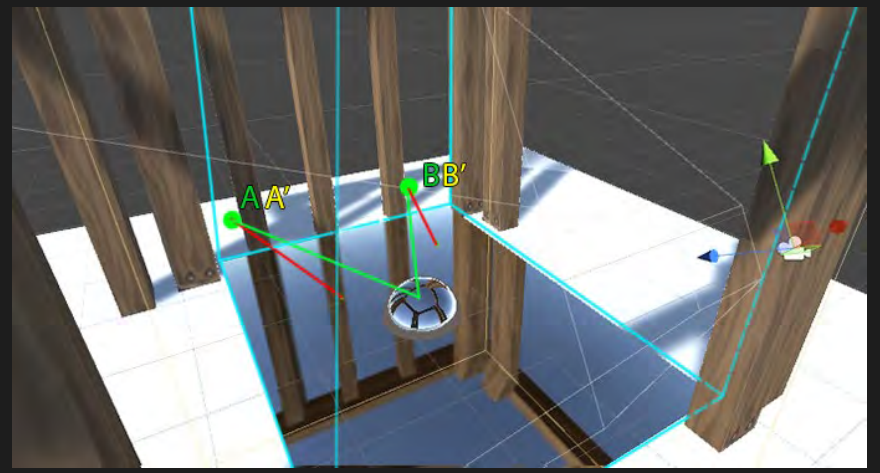
RP Box Projections

- The final color of [1,2] is [A', B'], while the correct color should be [A,B]
  - We want to sample the cubemap from the Reflection probe center [C] using C-A and C-B vectors as sampleVectorDir
    - In this way the final color will be [A,B]
- We must use BOX PROJECTION
- Moreover, the ReflectionProbe should be placed at position 0,0.1,0: a little above the floor



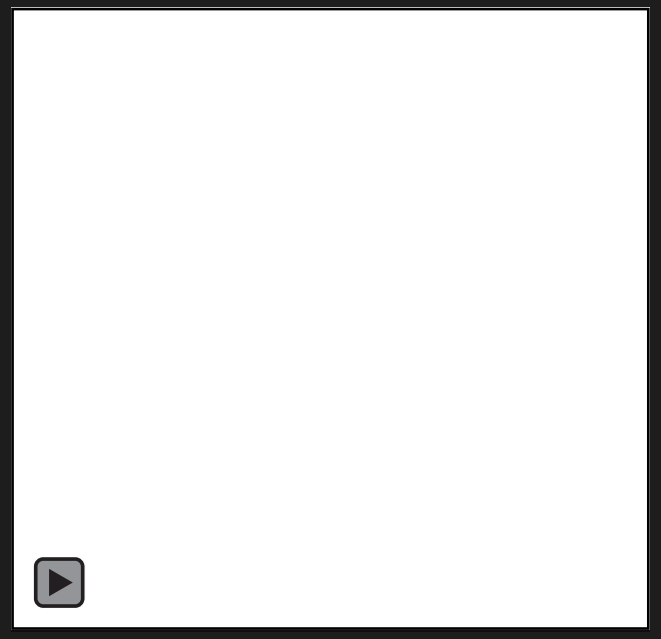






RP Box Projections

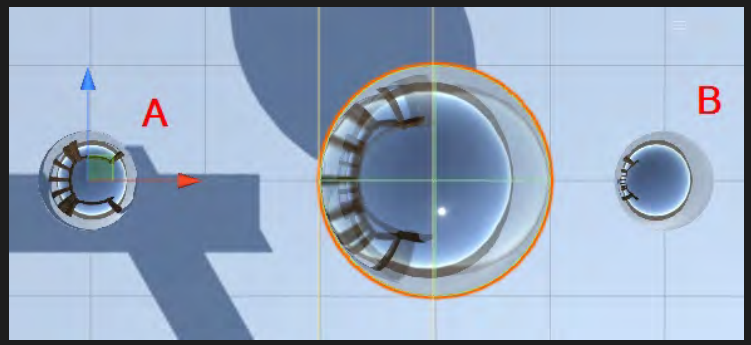
- RP Area size is always axis-aligned, it ignores all rotations and scaling. Unity uses these areas to
  - decide which probe is used when rendering an object
  - calculate the right cubemap direction sample if BoxProjection is enabled. A good Box projection around columns is: The box size should be big enough to include the entire columns

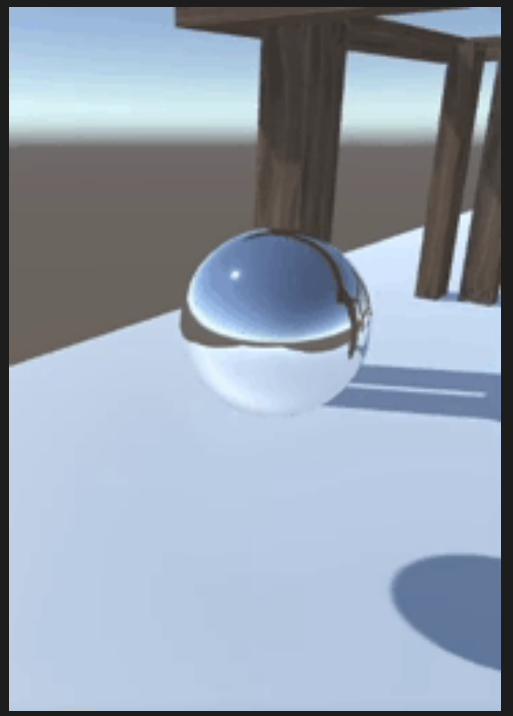


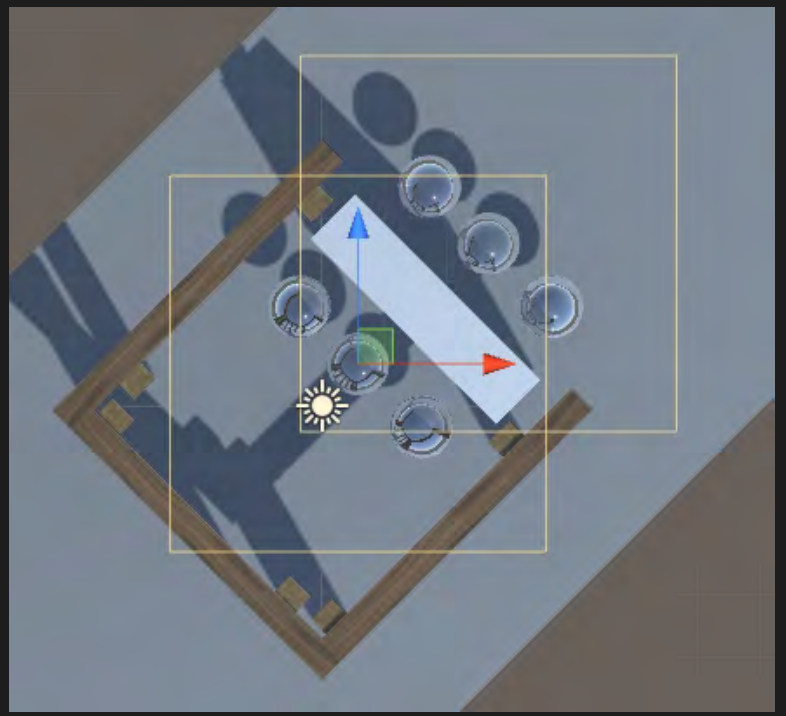
RP Interpolation

Interpolation

- Based on Importance value
- BlendProbes (only if between 2 probes, not between 1 probe and nothing)
  - The Sphere overlaps A by 1 unit, B by 2 units
  - A & B have same importance
  - ReflectionProbeContribution on Sphere = overlap / (Importance + # of probes)
    - Probe A 1.0 / (1.0 + 2.0) = 0.33
    - Probe B 2.0 / (1.0 + 2.0) = 0.67
- BlendProbes&Skybox
- Simple Doesn't blend probes (if between 2 probes, choose only one)
- AnchorOverride
  - [Built-in RP] Doesn't work in deferred mode
  - Useful for diagonal room spaces







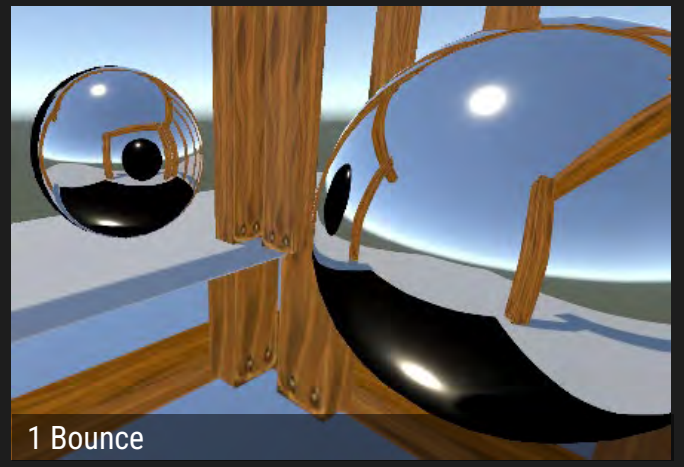
RP Bounces

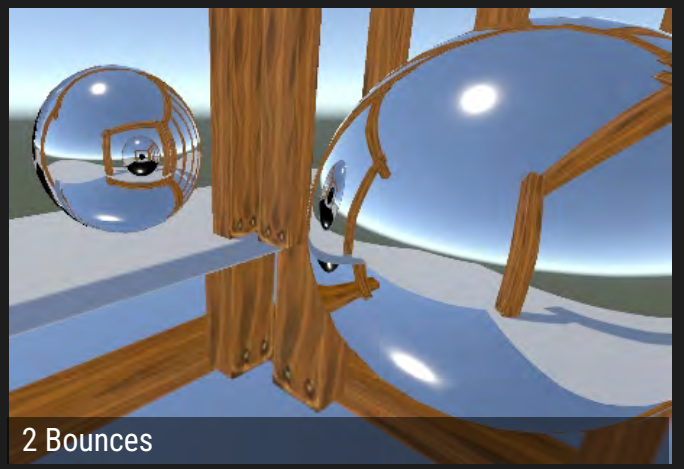
How to reflect the reflections?

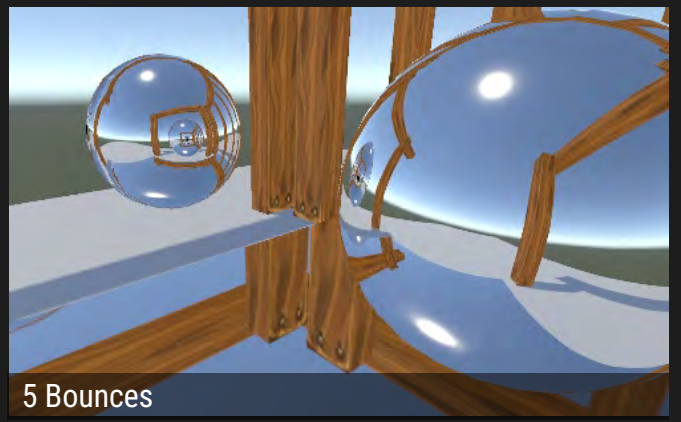
- Lighting/EnvironmentReflections/Bounces
- Very expensive, expecially if used on RT Reflection Probes

Try it

- Make the RT probe Realtime and move one sphere: if Bounces has max value, you should got bad performances


1 Bounce


2 Bounces


5 Bounces

Mixed lighting

- We want to use real-time lighting and baked lighting at the same time. How to achieve this?
- First bake all the environment, and then use real-time lighting for your characters

Setup

- Every Light must be set to Mixed
- Every light that caused GI bounces during baking, should be static
- [URP] URPAsset enable Lighting/MixedLighting

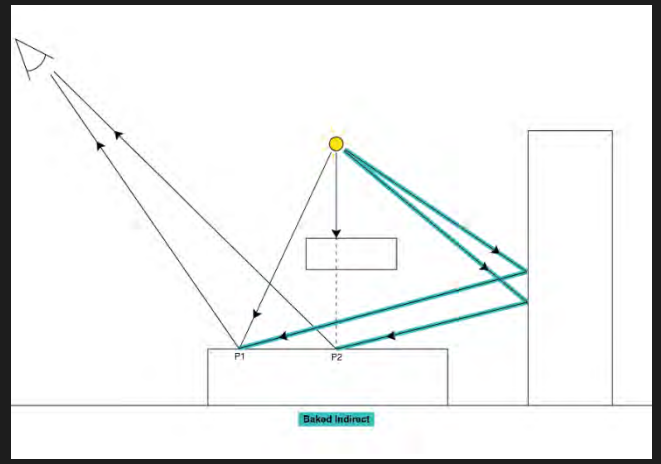
Mixed Lighting / BakedIndirect

Performance requirements
- mid-range PCs and high-end mobile devices

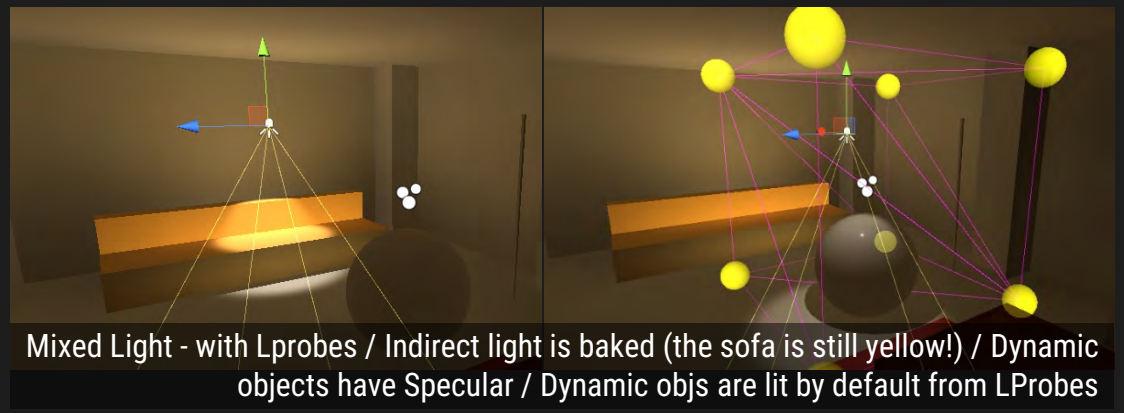
PROs
- Offers the same visual effect as real-time Lighting (Specular, RT-Shadows)
- Provides real-time shadows for all combinations of static and dynamic GObjs

CONs
- It has higher performance requirements relative to other Mixed Light modes, because uses Shadow mapping for shadow-casting static GObjs
- Shadows do not render beyond the Shadow Distance

Useful for
- Indoor game set in rooms. The viewing distance is limited, so everything that is visible should usually fit within the Shadow Distance
- Foggy outdoor scene: use the fog to hide the missing shadows in the distance

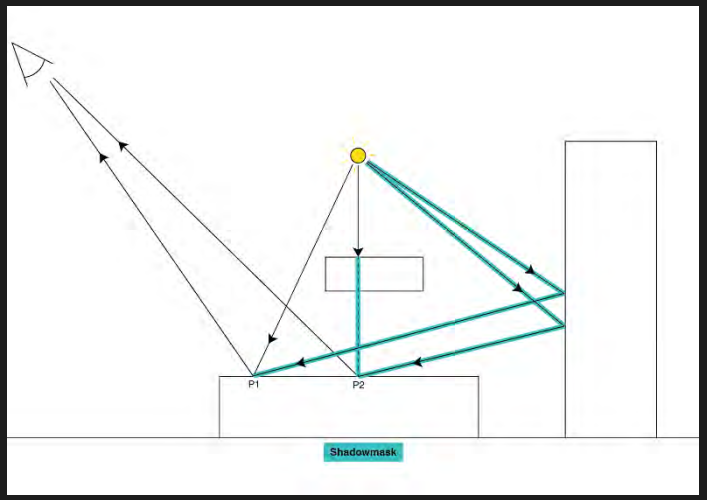


Mixed Light - no Lprobes / Indirect light is baked / Dynamic objects have Specular / No light on Dynamic objs if not directly lit



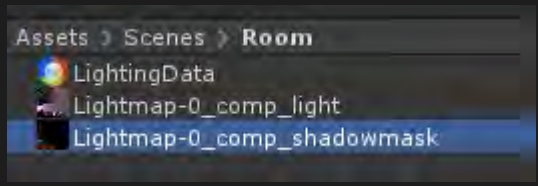
Mixed Light - with Lprobes / Indirect light is baked (the sofa is still yellow!) / Dynamic objects have Specular / Dynamic objs are lit by default from LProbes

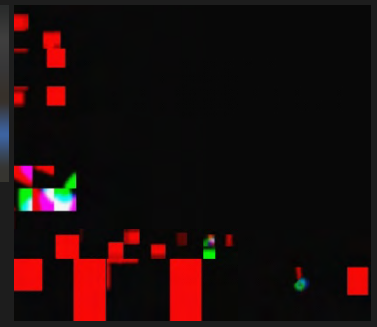
ML / Shadowmask (shadowmask)



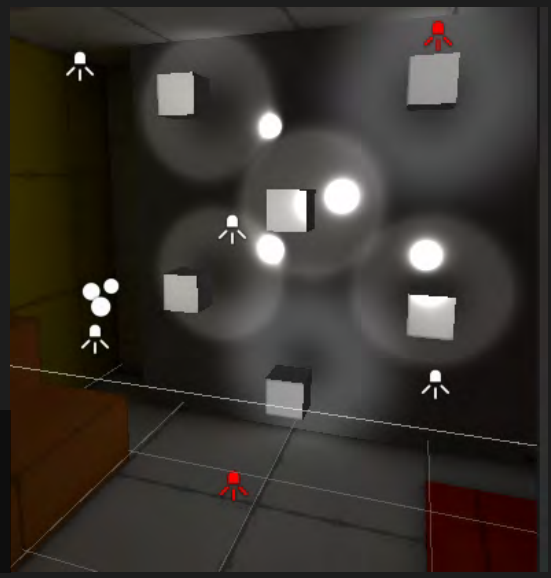
- Set Edit/ProjectSettings/Quality/ShadowMaskMode/Shadowmask

- To test Shadows, turn OFF Directional Light Mixed
- ShadowMasks (and Light Probes) stores information on up to 3 light sources: this enables shadow merging between dynamic and static objs. Above 3 lights, they will be fully baked
  - SceneView/LightOverlap helps to see when a Mixed light is fully baked





Performance requirements
- Mid



LightOverlap SceneView:
If we activate all spotlights, 3 of them will be mapped on Shadowmap, 1 is changed in realtime (no shadows for SR), the other (red spotlight gizmo) will be fully backed

[1]

ML / Shadowmask (shadowmask)

PROs
- Automatically composites overlapping shadows from static and dynamic Gobjs
- Offers less draw calls than distance shadowmask mode (no shadowmaps for static objs)
- Shadows cast by dynamic GameObjects can be correctly composited with precomputed shadows from static GameObjects, avoiding inconsistencies like double shadowing (RT shadows + Static shadows doesn't add each other).

CONs
- Low-resolution shadows from static GObjs onto dynamic GObjs, via Light Probes (because of the shadowmask)
- Lights can move only inside baked occlusion

Useful for
- Almost fully static Scene, using specular Materials, soft baked shadows and a dynamic shadow caster, not too close to the camera

ML / Shadowmask (Distance Shadowmask)

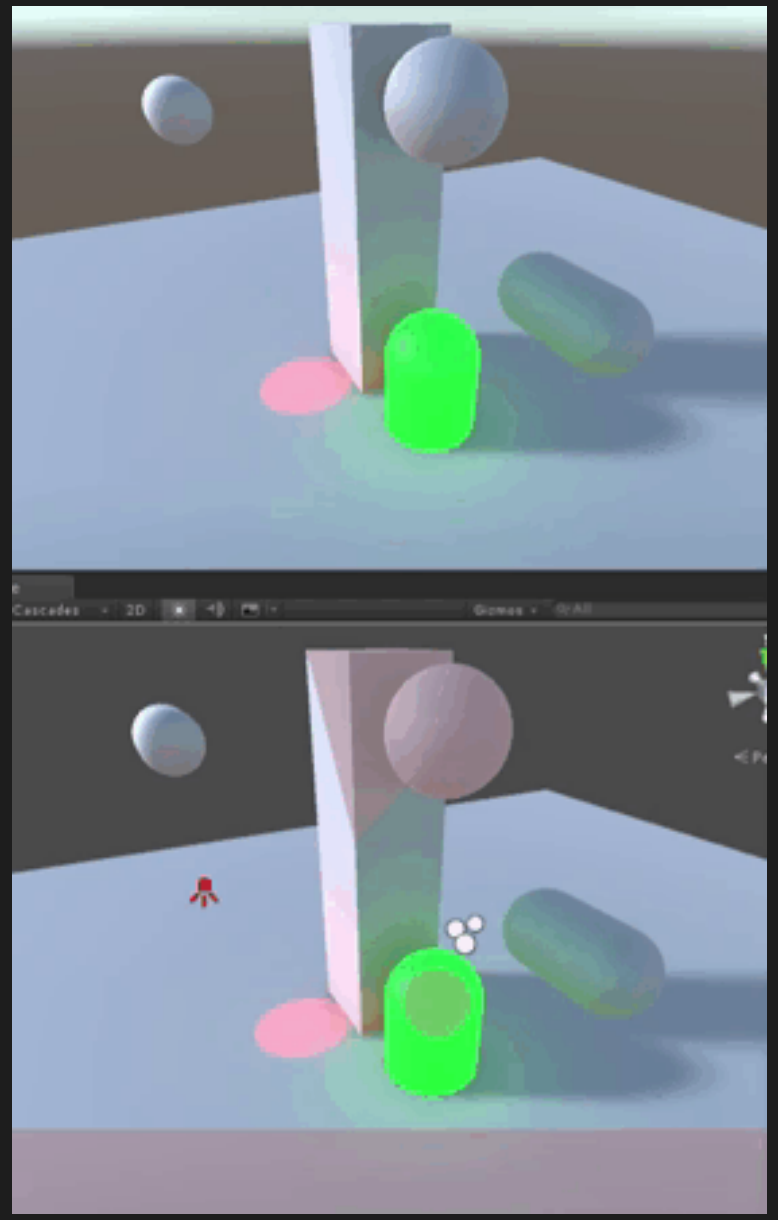
- Set Edit/ProjectSettings/Quality/ShadowMaskMode/DistanceShadowmask
- The Distance Shadowmask mode is a version of the Shadowmask lighting mode that includes high quality shadows cast from static GObjs onto dynamic GObjs within the shadow distance

Performance
- higher performance requirements than Shadowmask mode: both static & dynamic GObjs are rendered into the shadow map (demo video), every frame
- | If you bake lights in Shadowmask/DistanceShadowmask mode, Shadow masks are not used, but are rendered. Why?
  - Shadowmask are useful to know how to blend RTShadows with Baked shadows
  - You can always switch for performance reason to Shadowmask/Shadowmask mode @ realtime

PROs
- Same as ShadowMask Mode, plus:
- It provides real-time shadows from dynamic GObjs onto static GObjs, and static GObjs onto dynamic GObjs

CONs
- Same as ShadowMask Mode
- Dynamic Objects doesn't cast shadows if farther than shadowDistance

Useful for
- Open world Scene with shadows up to the horizon, and complex static Meshes casting real-time shadows on moving characters
- Shadowmask Lighting Mode provides the highest fidelity shadows among all the Lighting Modes, but has the highest performance cost and memory requirements. It is suitable for rendering realistic scenes where distant GameObjects are visible, such as open worlds, on high-end or mid-range hardware.



ML / Shadowmask - Distance Shadowmask switch

- You can change between these two mode in Quality panel: why not in Lighting/Scene panel?
  - Because it is possible to change the mode at run-time. For instance, you can use Shadowmask for in-door environment (e.g. to achieve soft shadows in a hangar) and switch to Distance Shadowmask for out-door environment within the same scene
    - QualitySettings.shadowmaskMode = ShadowmaskMode.Shadowmask
    - You can set various requirements for different hardware setups. For example, in a game, it can be exposed to Menu UI in which user can adjust the setting: you can rely on the regular Shadowmask for low-end hardwares and use Distance Shadowmask for high-end PCs

ML / Subtractive

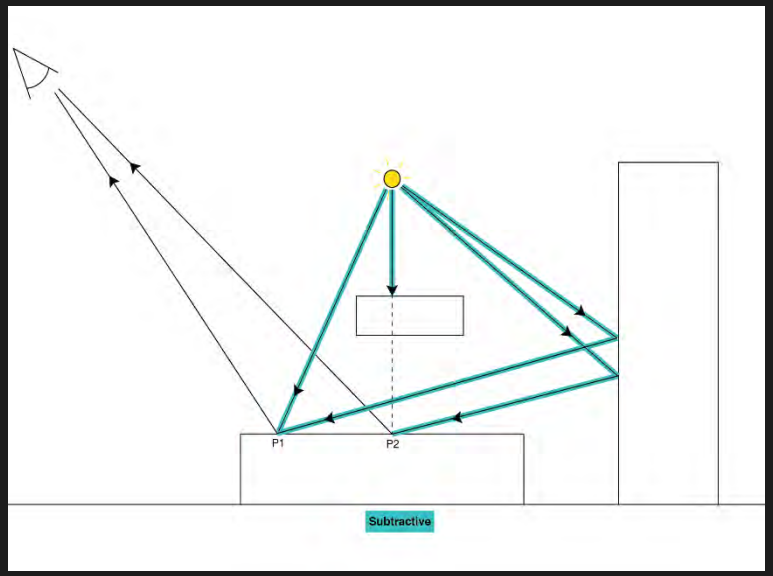
- The only Mixed lighting mode that bakes direct lighting into the light map: Unity cannot perform any direct lighting calculations at run time.
  - try to move one light @ RunTime – it'll remain baked in Intensity map
- Subtractive mode has a Realtime Shadow Color field. Unity uses this color in the Shader to composite real-time shadows with baked shadows, because there is no correct value that the engine can predetermine
  - In ShadowMask mode, Unity knows how to blend RT and baked shadows, because uses shadowmask information

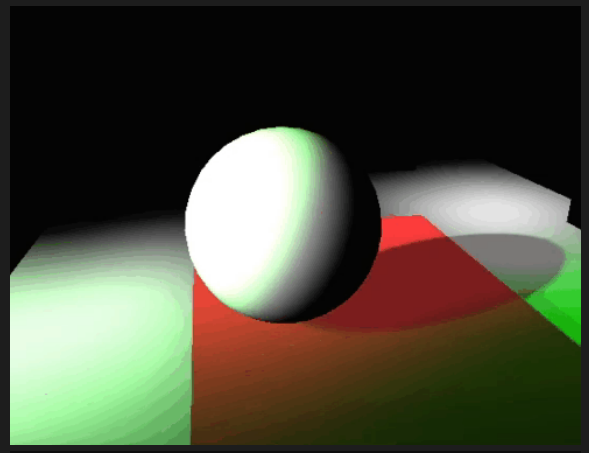
PROs

- Provides high-quality soft shadows between static Gobjs
- Dynamic objs can cast shadows from Realtime lights onto Static objs (but only for the main mixed light)

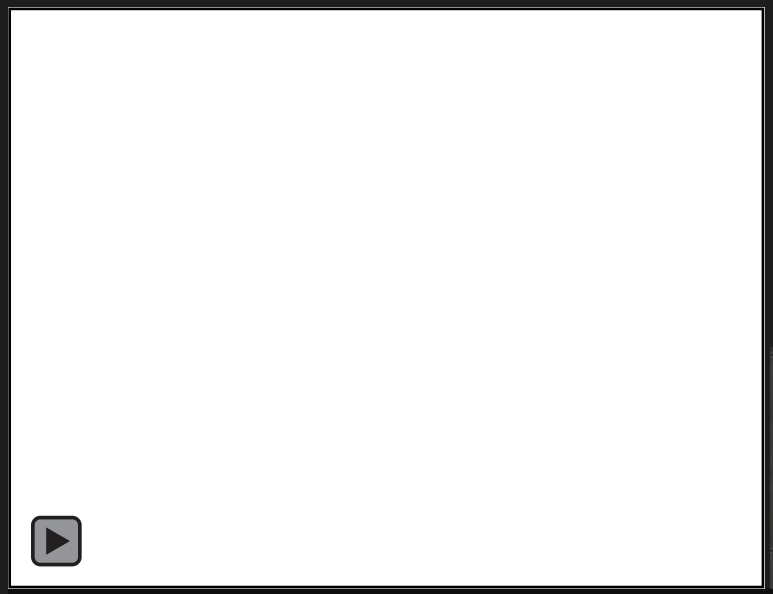
CONs

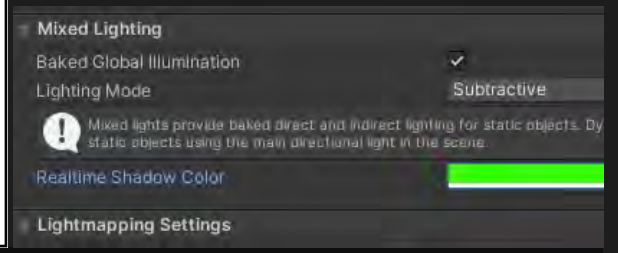
- No real-time direct lighting > No specular lighting
- Provides low-resolution shadows from static GObjs onto dynamic GObjs, via Light Probes
- Provides inaccurate composition of dynamic and static shadows
- Only the main Mixed light can cast shadows from dynamic objs onto Static objs (try to change intensity of the 2 directional lights, to change the main light)
- Designed for forward rendering and gamma color space: because it is relatively cheap, is targeted for low end devices such as mobile platform





Only the main Mixed light can cast shadows from dynamic objs to static objs





Since there is no ShadowMask, Realtime shadow from Mixed lights can't be merged perfectly. We must Choose a Realtime Shadow Color

QualitySettings

- Most of the Lighting settings we have seen is in ProjectSettings/Quality panel. Here you can configure different Quality settings for your game, and set the Default one
- You can change settings at Runtime using
  QualitySettings.SetQualityLevel(int level, bool applyExpensiveChanges);

- Changing the quality level can be an expensive operation if the new level has different anti-aliasing setting. It's fine to change the level when applying in-game quality options, but if you want to dynamically adjust quality level at runtime, pass false to applyExpensiveChanges so that expensive changes are not applied
- We can use a different number of player quality, depending on the final platform: levels that are not used for that platform are stripped. You should not expect a given quality setting to be at a given index

[Built-in RP] Pixel light / Vertex light

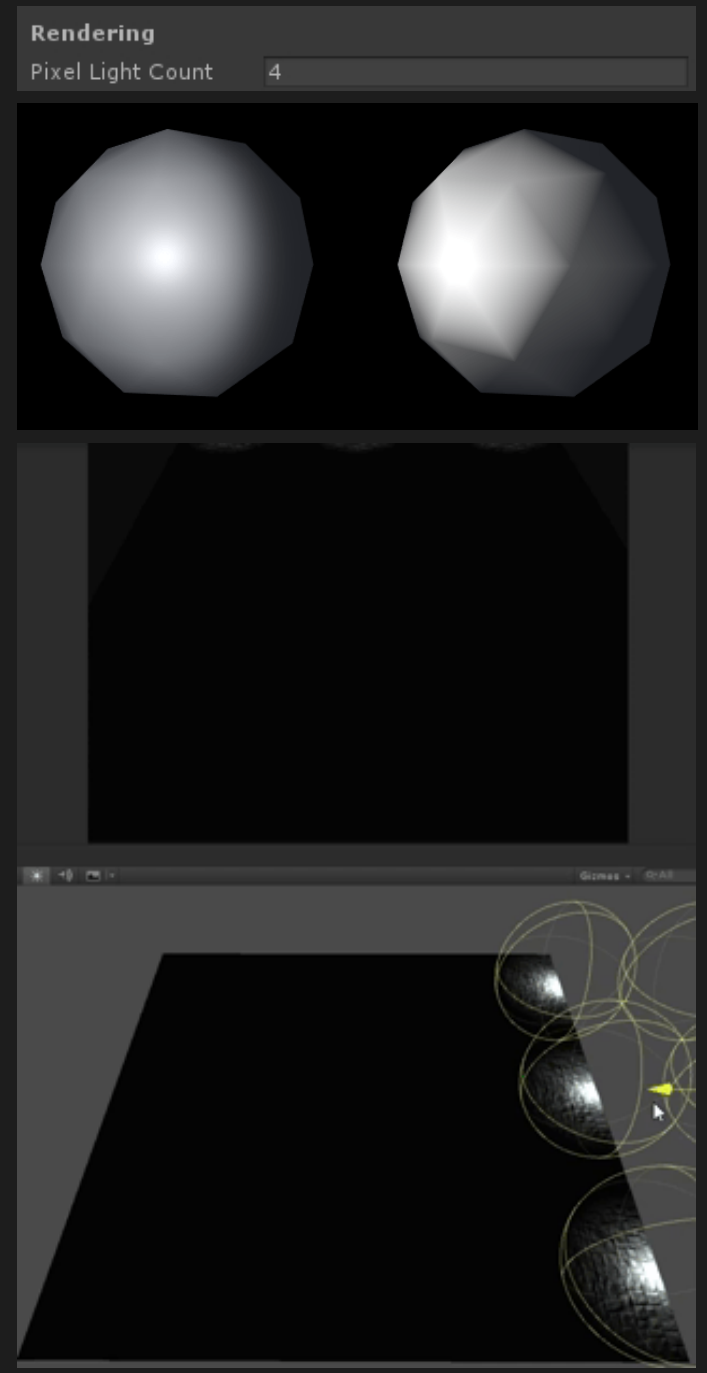
- Forward rendering mode
- Pixel lighting is mandatory for a lot of effects
  - Specular highlights
  - Normal mapping
  - Real-time shadows
  - Light cookie
  - Spotlight Shades

- QualitySettings/PixelLightCount Limit the Pixel light # in the scene
- Light RenderMode value
  - NotImportant Per vertex
  - Important Per pixel

Rules followed by Unity

1. Lights with Render Mode Not Important => always per-vertex or SH
2. Brightest directional light => always per-pixel
3. Lights with Render Mode Important => always per-pixel (regardless PixelLightCount settings)
4. If there are still less lights than current PixelLightCount => more per-pixel lights, in order of dec brightness

NB: SH are not calculated on static Meshes



[URP] Pixel light / Vertex light

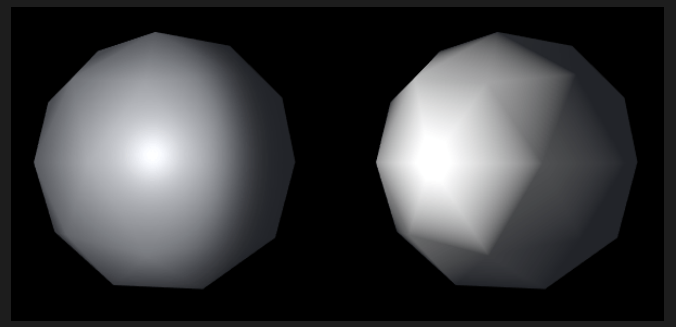
- Inside URPAsset properties you can tweak Main/Additional lights Pixel/Vertex shading
- Main light is the brightest Directional light. If no dir light is present, there is no main light

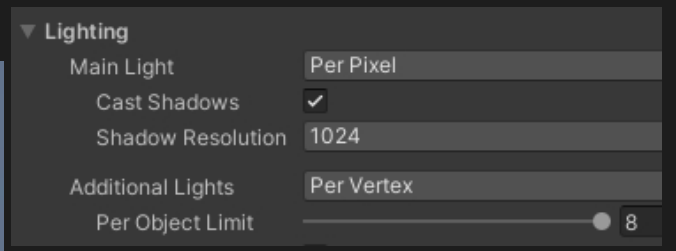
- Unity's BIRP forward rendering path can technically support unlimited number of lights => extreme performance penalty
- The URP is faster at doing 8 lights than the BIRP forward renderer is at doing 4
- The concept of deferred rendering is explicitly designed to allow for multiple lights. It is what it exists to handle

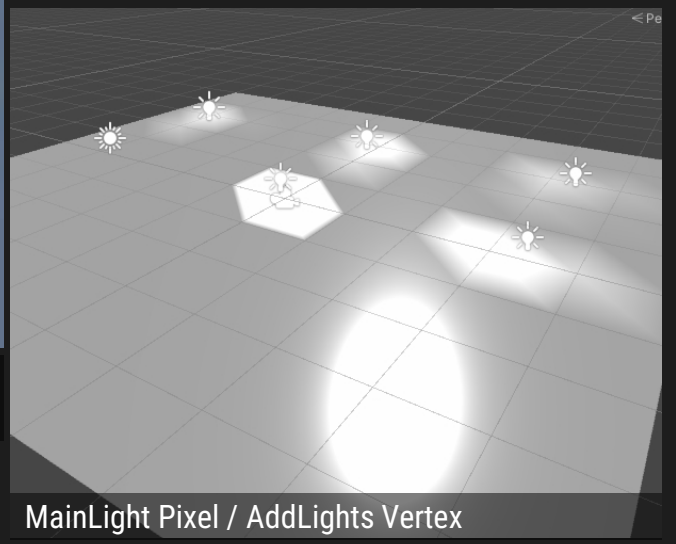
Try it
- Open [PixelVertex_URP_01]
- Via LightExplorer panel, disable all light.
- Switch to Top view isometric
- Enable the first 8 spot lights
- Add a directional light with shadows
- This is the URP limit

AddLight Pixel / PerObjLimit 8: Single cubes are lit by 4 pixel lights, plane is lit by a maximum of 8 point lights

[PixelVertex_URP_01]







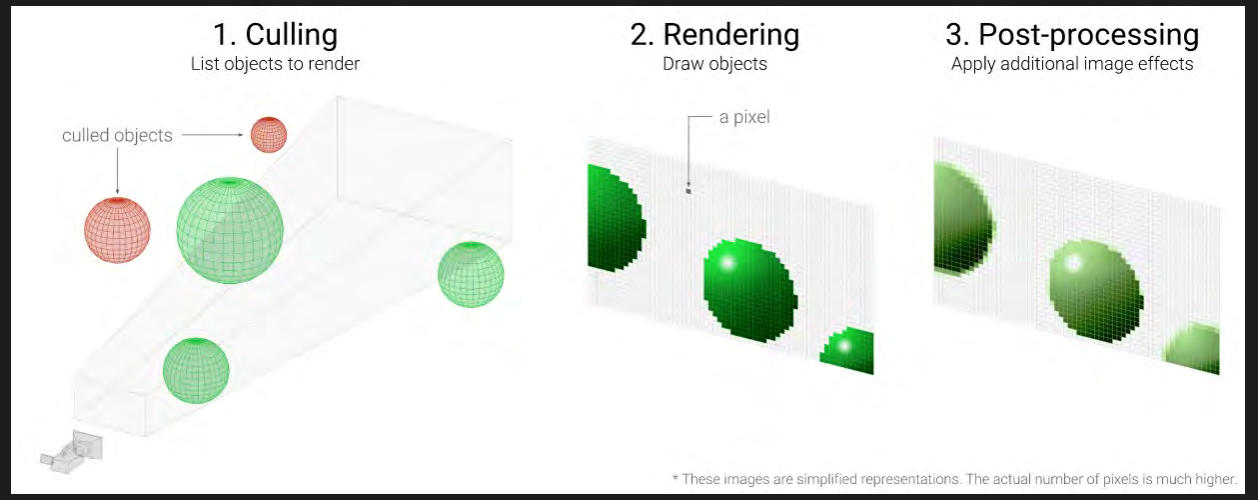
MainLight Pixel / AddLights Vertex

What is a Render Pipeline

- Determines how the objects in your scene are displayed, in three main stages
  - Culling; it lists the objects that need to be rendered, preferably the ones visible to the camera (frustum culling) and unoccluded by other objects (occlusion culling)
  - Rendering; it draws the objects, with the correct lighting and some of their properties, into pixel-based buffers
  - Post-processing operations can be carried out on these buffers, for instance applying color grading, bloom and depth of field

For each drawcall (when CPU sends Vertices to GPU)

1. VERTEX SHADER Scrip executed per vertex
2. RASTERIZER Culling / Interpolates vertices values between pixels / From 3D Frustom to 2D Rectagle screen / Sends candidates pixels to Pixel shader
3. PIXEL SHADER Lighting computation



Forward Rendering Path

In the Pixel shader

- Light calculations # = [geometry_fragments #] x [num_lights]

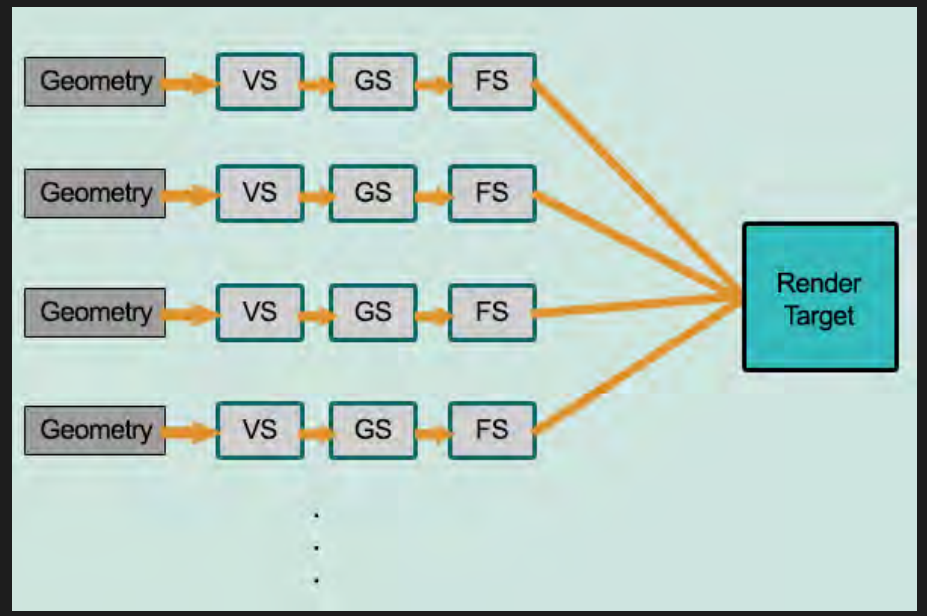
Performances

- Lighting calculations have to execute for each visible fragment of every polygon on the screen, regardless if it overlaps or is hidden by another polygon's fragments
- Screen resolution 1024x768 > nearly 800K pixels that need to be rendered > nearly 1M fragment operations every frame (taking into account overdraw)
  - A lot of the fragments will never make it to the screen because they were removed with depth testing > Lighting calculation time wasted on them
- It's not over: we have to render that scene again for each light
  - [num lights] x 1M fragment operations per frame
  - What if you had a long road with 100 lights? PShader lighting calculations: 100M

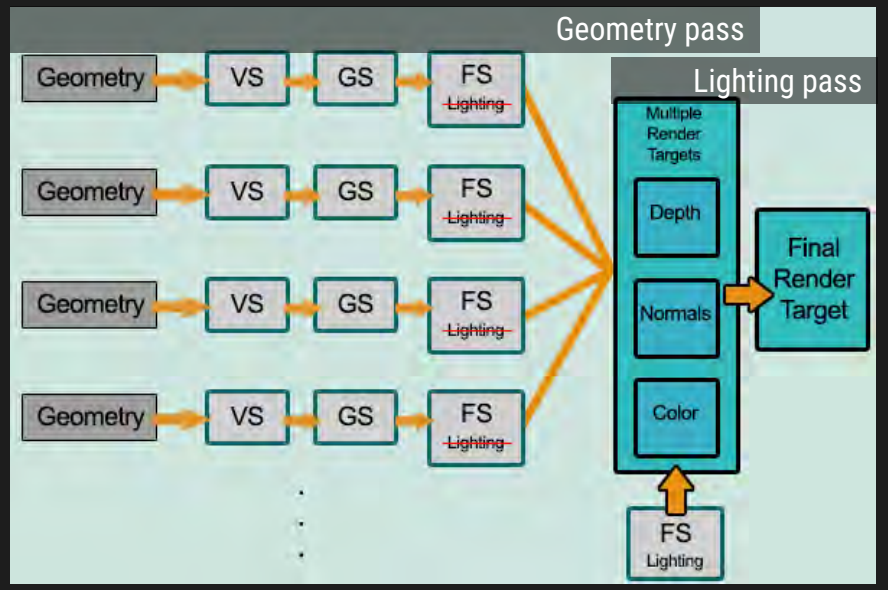
Complexity

- O(num_geometry_fragments * num_lights)
- Directly related to the number of geometries and number of lights

- Each light
  - New drawcall [BIRP]
  - New iteration on Pixel shader [URP]



Deferred Rendering Path

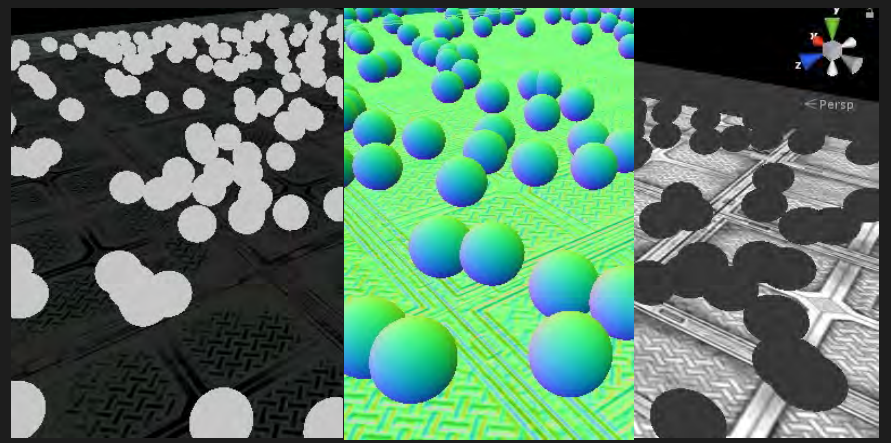


Geometry Pass

- Render the entire scene like in FRendering, but the PShader output is not the screen, instead the PShader output are multiple RederTargets (normal, specular, albedo)
  - No lighting calculations

In the Pixel shader

- Lighting calculations # = 0
- Gbuffer calculations # = [geometry_fragments #] x #Rtarget
  - In each PS we create 4 RT at the same time



Lighting Pass

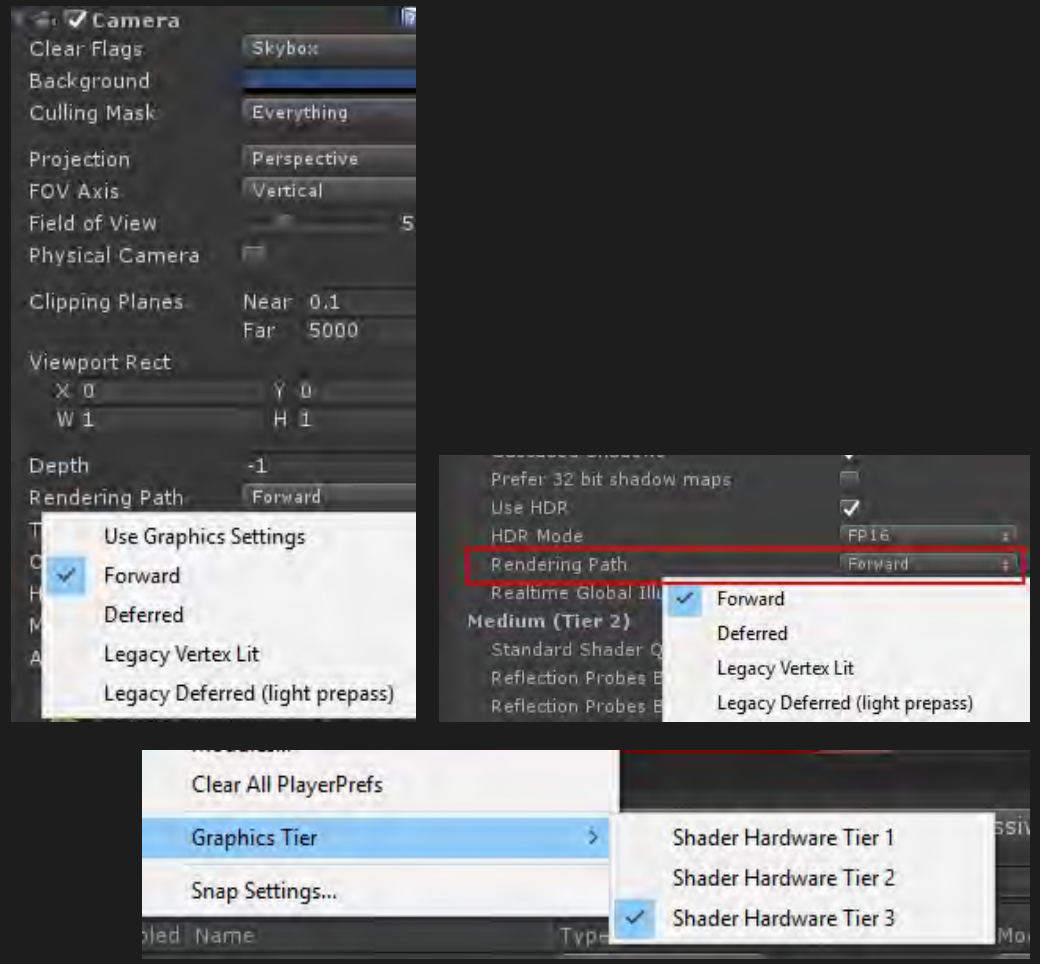
1. Pass to the VSHADER a FullScreenQuad
2. Rasterizer will let all the pixel to pass through the Pshader
3. Use the Multiple render targets in the GBuffer as Textures
4. PSHADER

In the Pixel shader

- Lighting calculations # = [screen_resolution] x [num_lights]
- We'll see later that this Lighting calculations # can be highly optimized in deferred rendering

[BIRP] Forward / Deferred switch

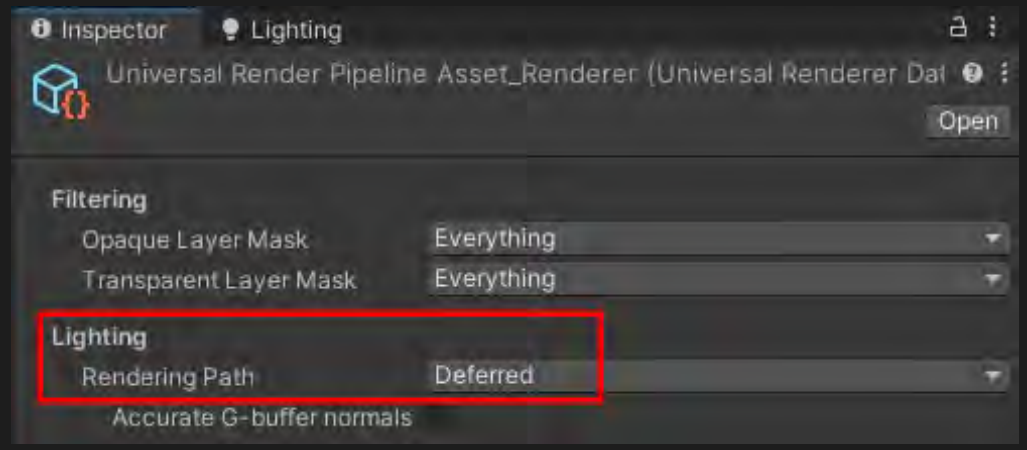
- Camera/Rendering Path
- ProjectSettings/Graphics
- Edit/GraphicsTier

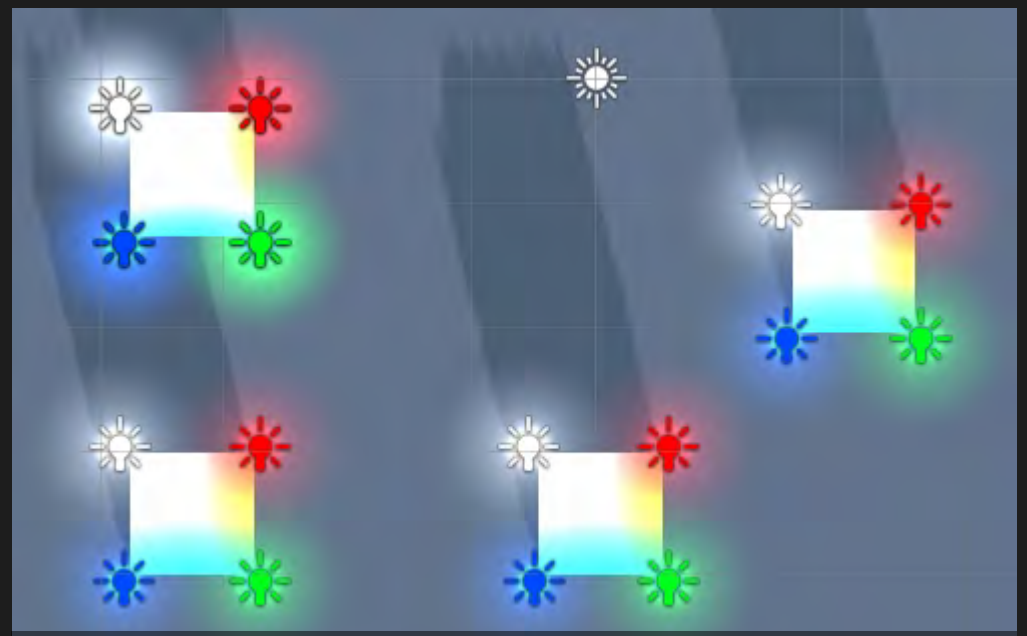


[URP] Forward / Deferred switch

- Select your UniversalRenderPipelineAsset_Renderer Asset
- Lighting/RenderingPath/Deferred
- Accurate G-buffer normal
  - OFF Quantization of normal, better performances, especially on mobile GPU, could lead to graphic artifacts
  - ON Octahedron encoding, extra load on GPU

Minimum requirements
- Shader Model 4.5
- Deferred Rendering Path does not support the OpenGL-based graphics API: Desktop OpenGL, OpenGL ES 2.0, OpenGL ES 3.0, WebGL 1.0, WebGL 2.0



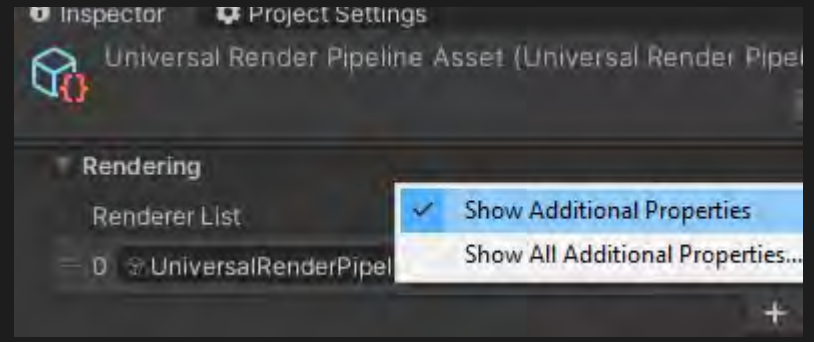


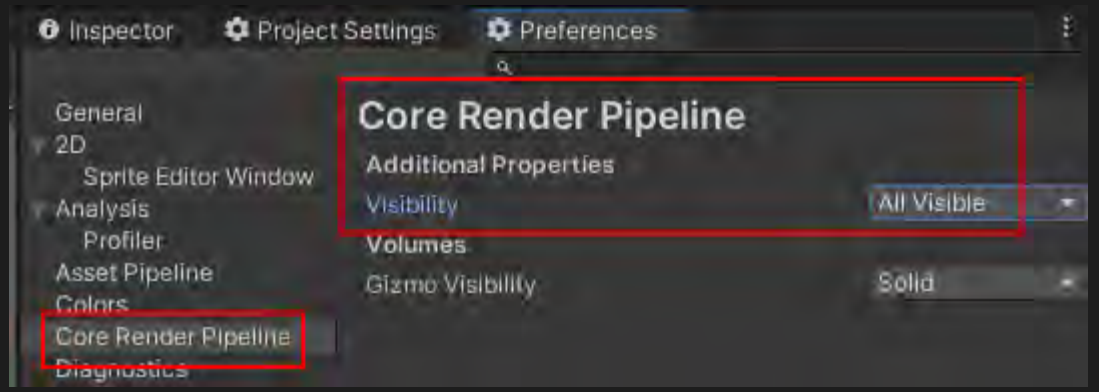
After Deferred switch, you can enable all the spotlights in the scene

Render pipeline options

New Unity UI hides some options. To see them all:

- Use the vertical 3 dots in each section and select “ShowAdditionalProperties”
- Preferences/CoreRenderPipeline/AddProperties/Visibility/AllVisible





Rendering Path comparison

Feature	Forward	Deferred
Maximum number of real-time lights per object.	9 Lights per object.	Unlimited number of real-time lights.
Per-pixel normal encoding	No encoding (accurate normal values).	Two options: • Quantization of normals in G-buffer (loss of accuracy, better performance). • Octahedron encoding (accurate normals, might have significant performance impact on mobile GPUs). For more information, see the section Encoding of normals in G-buffer.
MSAA	Yes	No
Vertex lighting	Yes	No
Camera stacking	Yes	Supported with a limitation: Unity renders only the base Camera using the Deferred Rendering Path. Unity renders all overlay Cameras using the Forward Rendering Path.

GBuffer

data structure for each pixel of the render targets
Albedo (sRGB)

This field contains the albedo color in sRGB format, 24 bits.

MaterialFlags

- Bit 1, ReceiveShadowsOff: if set, the pixel does not receive dynamic shadows.
- Bit 2, SpecularHighlightsOff: if set, the pixel does not receive specular highlights.
- Bit 4, SubtractiveMixedLighting: if set, the pixel uses subtractive mixed lighting.
- Bit 8, SpecularSetup: if set, the Material uses the specular workflow.

Specular

- SimpleLit Material: RGB specular color stored in 24 bits.
- Lit Material with metallic workflow: reflectivity stored in 8 bits, 16 bits are not used.
- Lit Material with specular workflow: RGB specular color stored in 24 bits.

Occlusion

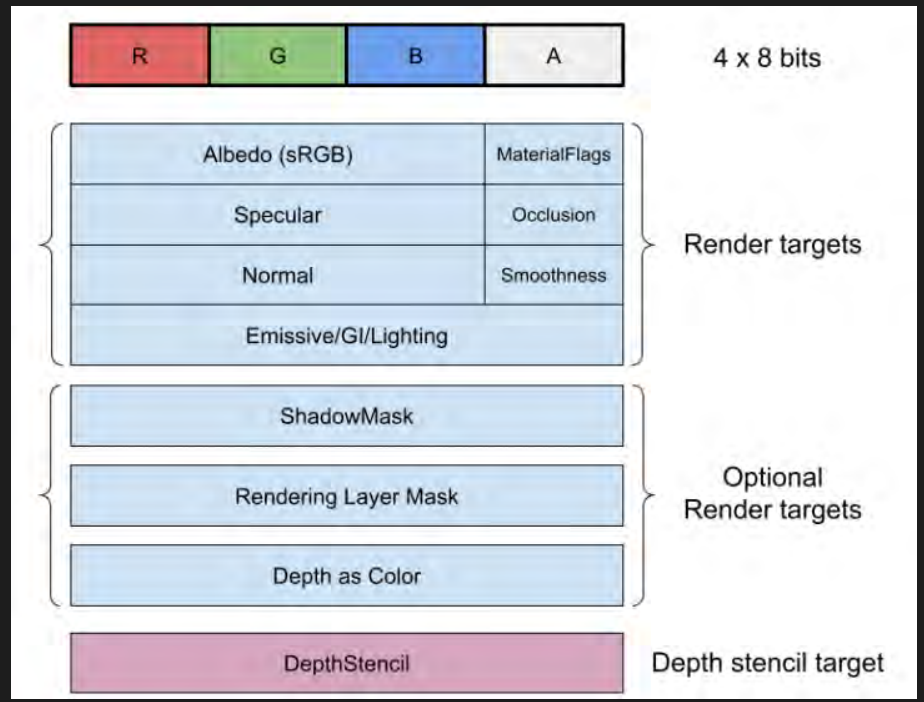
- This field contains the baked occlusion value from the baked lighting. For real-time lighting, Unity calculates the ambient occlusion value by combining the baked occlusion value with the SSAO value.

Normal

- This field contains the world space normals encoded in 24 bits

Smoothness

- This field stores the smoothness value for the SimpleLit and Lit materials.



GBuffer

Emissive/GI/Lighting

- This render target contains the Material emissive output and baked lighting. Unity fills this field during the G-buffer Pass. During the deferred shading pass, Unity stores lighting results in this render target.

ShadowMask

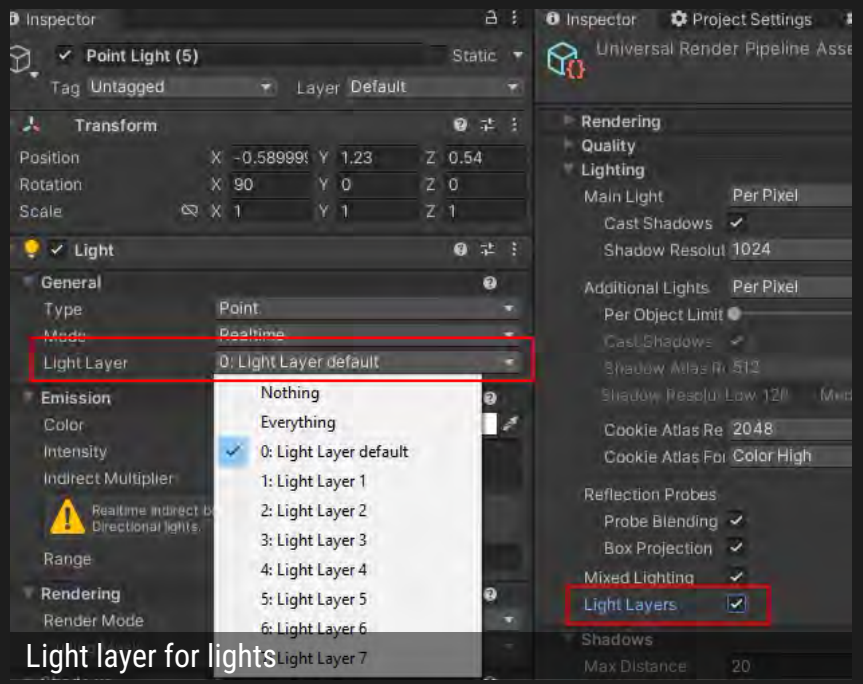
- Unity adds this render target to the G-buffer layout when Lighting Mode is set to Subtractive or Shadow mask
- The Subtractive and the Shadow mask modes are optimized for the Forward Rendering Path, and are less efficient in the Deferred Rendering Path. In the Deferred Rendering Path, avoid using these modes and use the Baked Indirect mode instead to improve GPU performance

Rendering Layer Mask

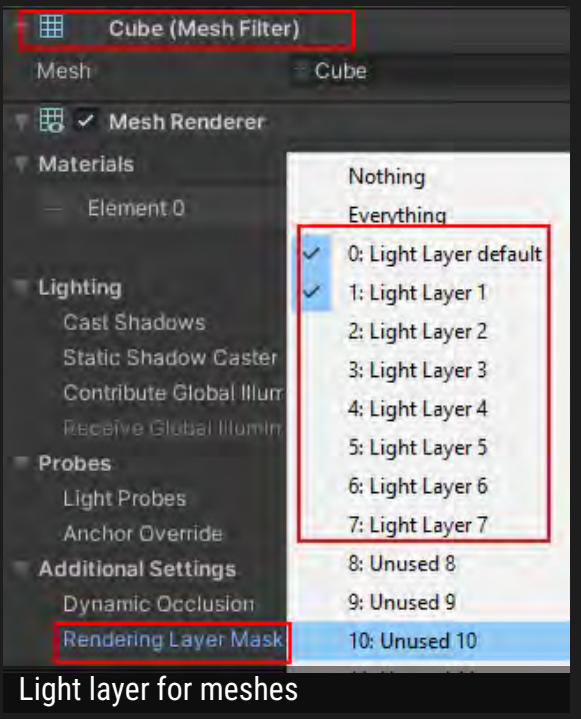
- Unity adds this render target to the G-buffer layout when the Light Layers feature is enabled (URP Asset, Lighting/LightLayers). The Light Layers feature might have a significant impact on the GPU performance

Depth as Color

- Unity adds this render target to the G-buffer layout when Native Render Pass is enabled on platforms that support it. Unity renders depth as a color into this render target. This render target has the following purpose:
  - Improves performance on Vulkan devices
  - Lets Unity get the depth buffer on Metal API, which does not allow fetching the depth from the DepthStencil buffer within the same render pass



Light layer for lights



Light layer for meshes

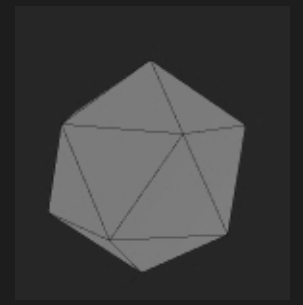
Forward/Deferred Rendering comparison

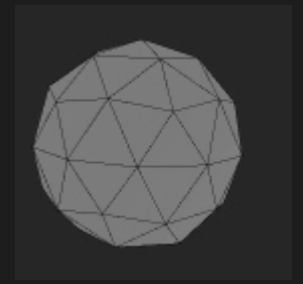
Example
- Screen Res 1024x768 = about 800K pixels = about 1M fragments (taking into account overdraw)
- # of lights: 100

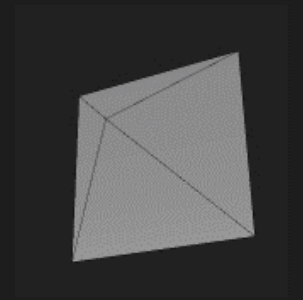
FRendering
- O(num_geometry_fragments * num_lights) = 1M*100 = 100M PixelShader with lighting calculations
- PSHADER EXECUTIONS = 100M

DRendering
- Geometry pass = 1M PixelShader (4 RT Normal, Specular, Albedo, Lightmap+Emission+RP are rendered at the same time) = 1M
- O(screen_resolution * num_lights) = 800K*100 = 80M PixelShader with lighting calculations (There is no waste of lighting calculations!)
- PSHADER EXECUTIONS: 80M + 1M = 81M

What if 99 lights are not Directional but Point lights that lit 1/10 of the screen fragments?
DRendering
- Geometry pass = 1M PixelShader (see previous example) = 1M
- Directional light: O(screen_resolution * 1) = 800K*1 = 800K PixelShader with lighting calculations
- Point lights: O(1/10 * screen_resolution * 99) = 80K*99 = about 8M PixelShader with lighting calculations
- PSHADER EXECUTIONS : 1M + 800K + 8M = 9.8M







Deferred Rendering Path

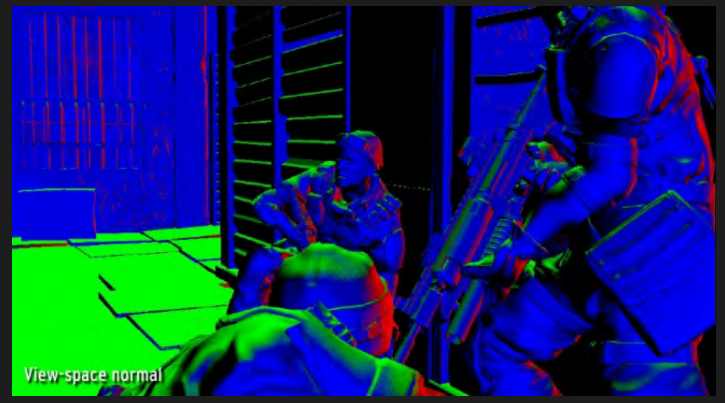
PRO

- Scene geometry decoupled from lighting
- Shading/lighting only applied to visible fragments
- Lot of Post-processing camera Fx rely on GBuffers data, already calculated during Geometry Pass
  - SSAO
  - MotionBlur

PROBLEMS

- No Transparent objects (Why?) / Solution: Alpha to coverage, Forward path
  - If something semi-transparent exists on the scene, there's no way to write down depth and normals for objects that is visible through semi-transparent objects and for current object itself. Unity handles this limitation rendering semi-transparent objects using forward rendering path at the end of the whole process. These objects can cast a shadow, but unfortunately they are unable to receive shadows from other objects
- No Orthographic cam: the cam will always use Forward rendering in orthographic mode (performance issues)
- No MSAA (MultiSamplingAA, use Post-processing AA instead)
  - Relies on data being stored per fragment, which is done via textures. This is not compatible with MSAA, because that anti-aliasing technique relies on sub-pixel data. You'll have to rely on a post-processing filter for anti-aliasing
- No MeshRenderer's ReceiveShadows flag: a mesh must always receive shadows
- Shadows are still dependent on the number of lights
- Old hardware/consoles/GPUs don't support MRT

Deferred Rendering Path / KillZone



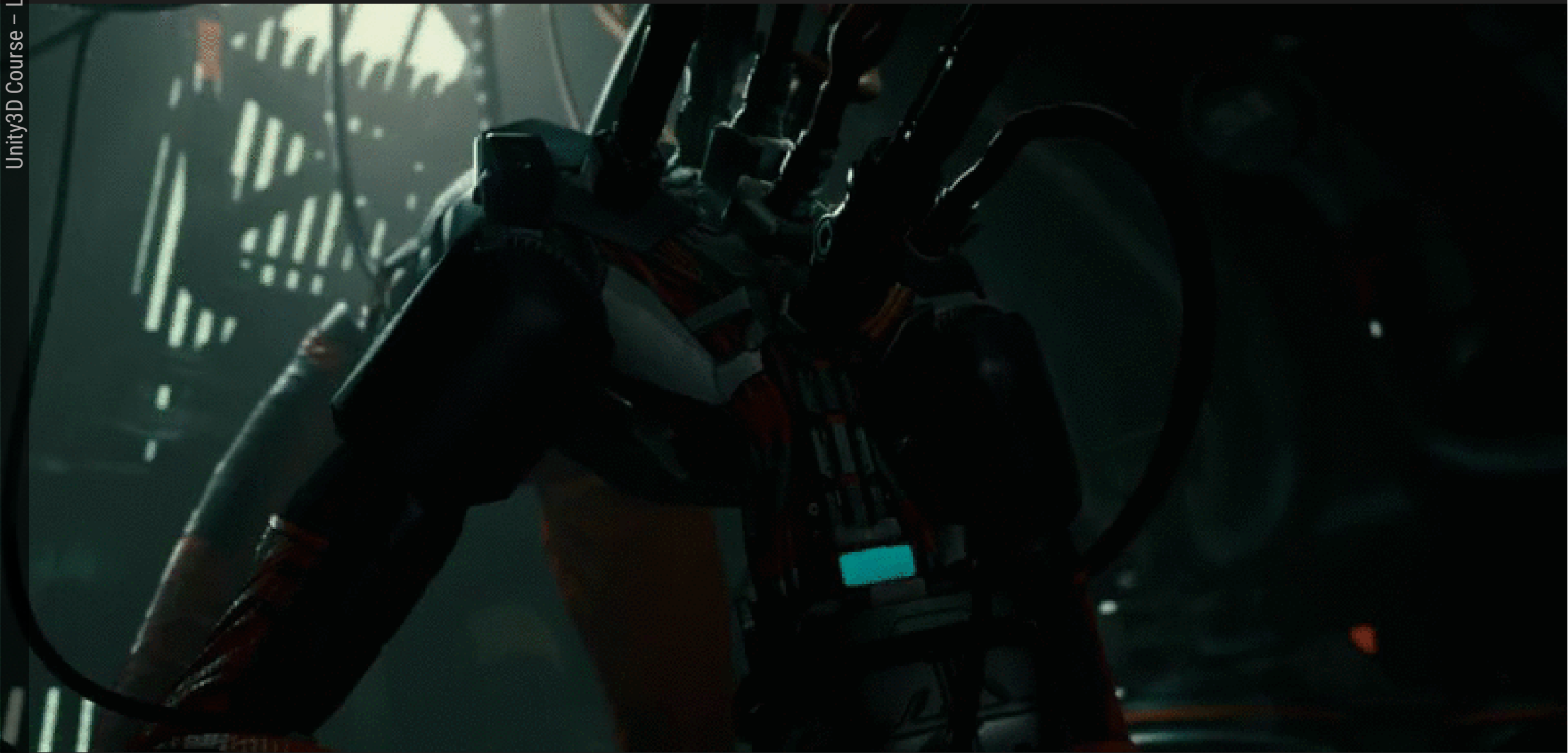
G-Buffer : Our approach

R8	G8	B8	A8	
Depth 24bpp			Stencil	DS
Lighting Accumulation RGB			Intensity	RT0
Normal X (FP16)		Normal Y (FP16)		RT1
Motion Vectors XY		Spec-Power	Spec-Intensity	RT2
Diffuse Albedo RGB			Sun-Occlusion	RT3

- Lighting accumulation – output buffer
- Intensity – luminance of Lighting accumulation
  - Scaled to range [0...2]
- Normal.z = sqrt(1.0f - Normal.x^2 - Normal.y^2)

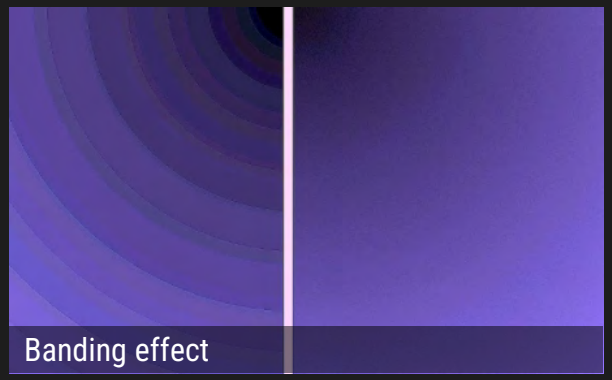
KILLZONE

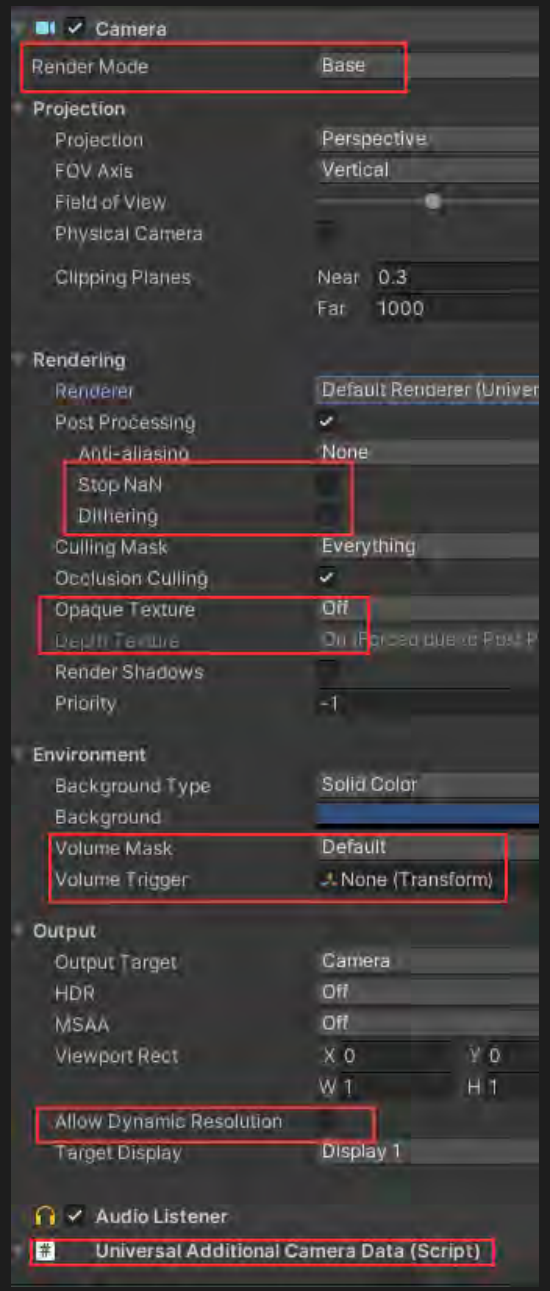
Post Processing



URP Cam component

- Open [Room_7] and add SimpleCameraController
- StopNan Enable the checkbox to make this Camera replace values that are Not a Number (NaN) with a black pixel. This stops certain effects from breaking, but is a resource-intensive process. Only enable this feature if you experience NaN issues that you can not fix
- Dithering Enable the checkbox to apply 8-bit dithering to the final render. This can help reduce banding on wide gradients and low light areas
- VolumeMask Use the drop-down to set the Layer Mask that defines which Volumes affect this Camera
- VolumeTrigger Assign a Transform that the Volume system uses to handle the position of this Camera. For example, if your application uses a third person view of a character, set this property to the character's Transform. The Camera then uses the post-processing and Scene settings for Volumes that the character enters. If you do not assign a Transform, the Camera uses its own Transform instead


Banding effect



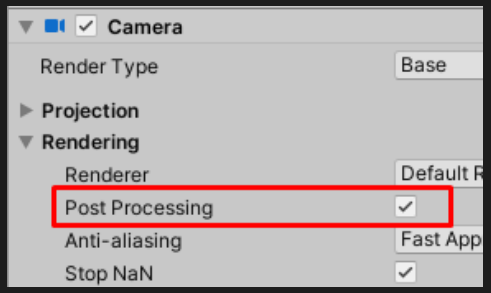
URP PProcessing limits

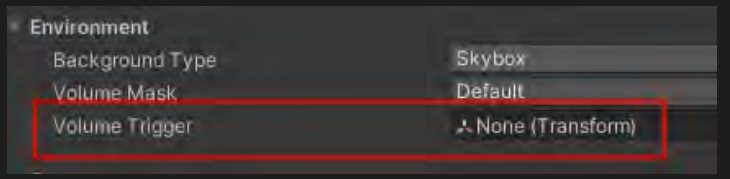
Post-processing	Uses Post-Processing Version 2 package	Uses integrated post-processing solution
Ambient Occlusion (MSVO)	Yes	In research
Auto Exposure	Yes	Not supported
Bloom	Yes	Yes
Chromatic Aberration	Yes	Yes
Color Grading	Yes	Yes
Depth of Field	Yes	Yes
Grain	Yes	Yes
Lens Distortion	Yes	Yes
Motion Blur Camera Object	 Yes Not supported	 Yes In research
Screen Space Reflections	Yes	Not supported
Vignette	Yes	Yes

URP PProcessing setup

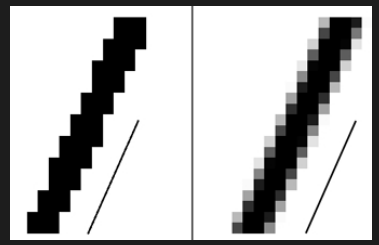
- Enable Camera component Rendering/Pprocessing flag
- Add a GObj with a Volume component
  - Global Volume works everywhere, the other Local volumes must be triggered by Camera/Environment/VolumeTrigger (Camera.transform if not specified)
  - Create a new Profile
  - Ensure that the Volume component is on Default layer
- Weight Influence the Volume has on the Scene. It is a multiplier to the value it calculates using the Camera position and Blend Distance
- Priority URP uses this value to determine which Volume it uses when Volumes have an equal amount of influence on the Scene. URP uses Volumes with higher priorities first
- Performances
  - For DOF, Unity recommends that you use Gaussian DOF for lower-end devices. For console and desktop platforms, use Bokeh DOF
  - For anti-aliasing on mobile platforms, Unity recommends that you use FXAA
  - VR
    - To reduce motion sickness in fast-paced or high-speed apps, use the Vignette effect for VR, and avoid the effects Lens Distortion, Chromatic Aberration, and Motion Blur for VR

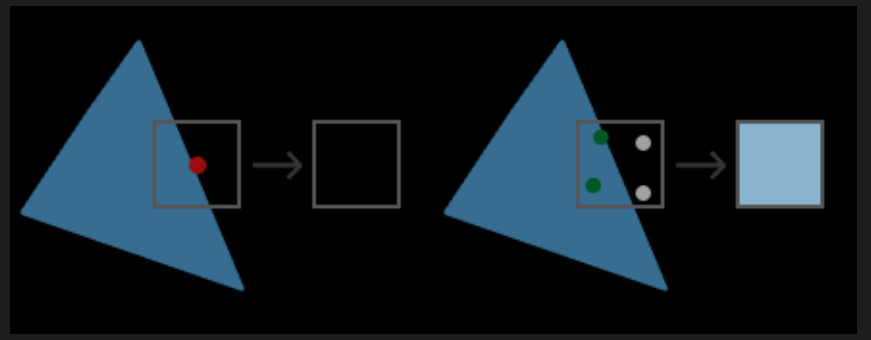
- If you use Local volumes, you can choose a BlendDistance to blend between different PProcessProfiles





[BIRP] AntiAliasing

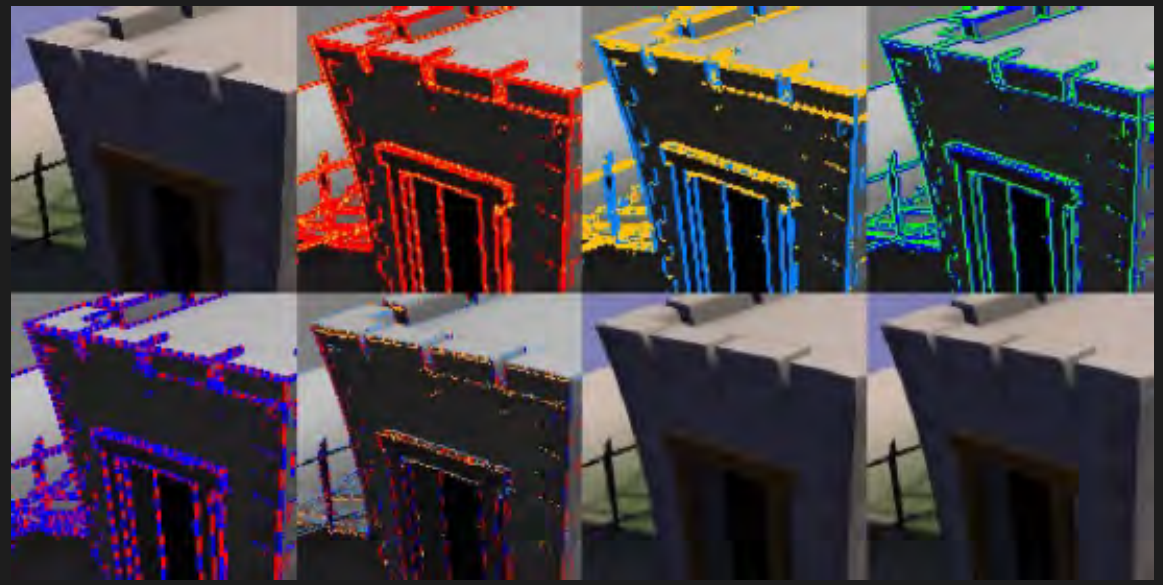




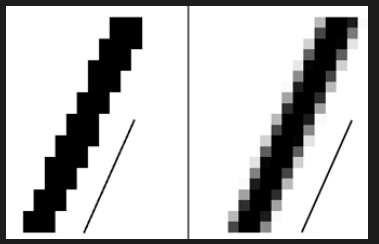
- Aliasing: problem in the conversion from analog to digital signal, when we use an insufficient sampling frequency (for images, insufficient # of pixels)
- Supersampling (SSAA) Very expensive and old approach. Render the scene with a higher resolution, uses extra pixel to calculate an average color, use this color to downsample the image to the final resolution
- Multisampling (MSAA) Uses a sub-pixel approach, doesn't render a higher resolution image. Sample that subpixels, and uses this extra info to calculate an average color. Enable it in QualitySettings and Camera component
    - Doesn't work in Deferred mode

Post Processing (from cheap to expensive)

- Enable it in PostProcessingLayer/AAMode
- Fast Approximate AA (FXAA)
    - Only edges
    - Loss of visual clarity especially on objs in motion
    - Cheapest technique > recommended for platforms that don't support motion vectors
- SubPixel Morphological AA (SMAA)
    - Only edges
    - Use pixel color and contrast to sharpen the entire scene
    - Use prev frame data
- Temporal Anti-Aliasing (TXAA)
    - Use prev frame data, analyzing still and motion fragments
    - Use motion vectors

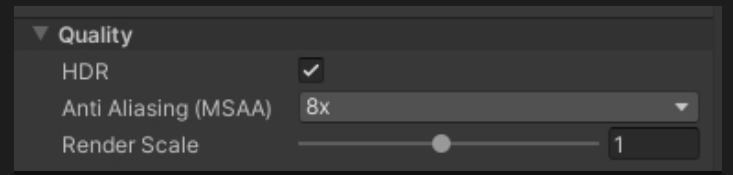


[URP] AntiAliasing

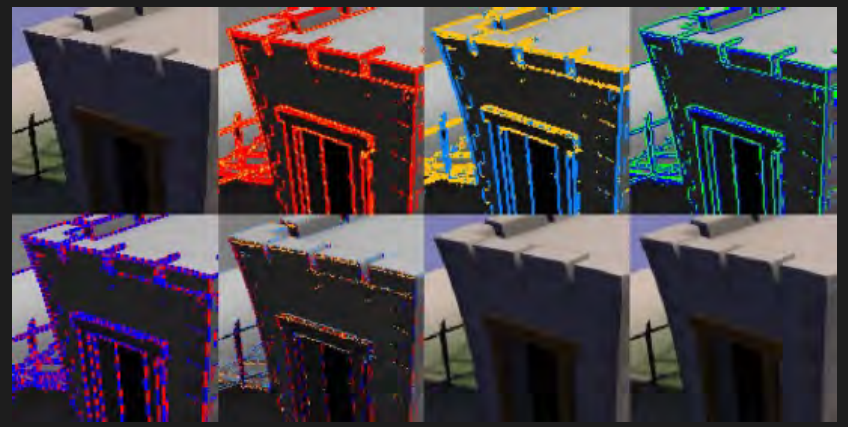


- Aliasing: problem in the conversion from analog to digital signal, when we use an insufficient sampling frequency (for images, insufficient # of pixels)

- Supersampling (SSAA) Very expensive and old approach. Render the scene with a higher resolution, uses extra pixel to calculate an average color, use this color to downsample the image to the final resolution
  - An app runs at a fixed resolution. Some apps allow the resolution to be changed via a settings menu, but this requires a complete reinitialization of the graphics. A more flexible approach is to keep the app's resolution fixed but change the size of the buffers that the camera uses for rendering. This affects the entire rendering process except the final draw to the frame buffer, at which point the result gets rescaled to match the app's resolution
  - Scaling the buffer size can be used to improve performance, by reducing the amount of fragments that have to be processed. This could for example be done for all 3D rendering, while keeping the UI crisp at full resolution. The scale could also be adjusted dynamically, to keep the frame rate acceptable. Finally, we could also increase the buffer size to supersample, which lessens aliasing artifacts caused by a finite resolution. This last approach is also known as SSAA, which stands for supersampling antialiasing



URP Asset/Quality/Camera Render scale option (if>1 => SSAA)

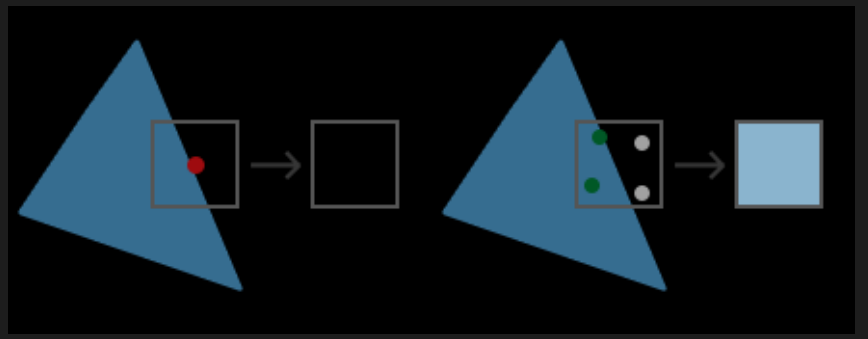


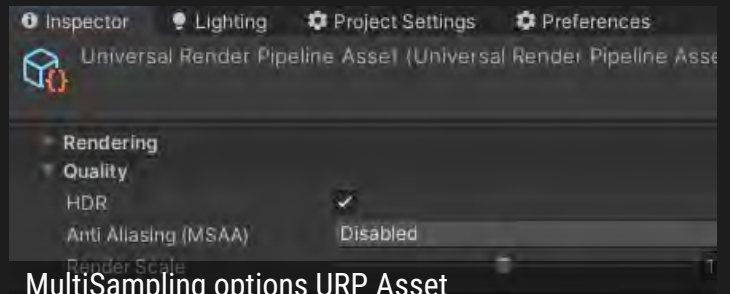
SuperSampling

[URP] AntiAliasing

Multisampling (MSAA) Uses a sub-pixel approach, doesn't render a higher resolution image. Sample that subpixels, and uses this extra info to calculate an average color.

- Enable it in URPAsset/Quality
- To use it: BaseCamera/Output/MSAA
- Use Forward rendering





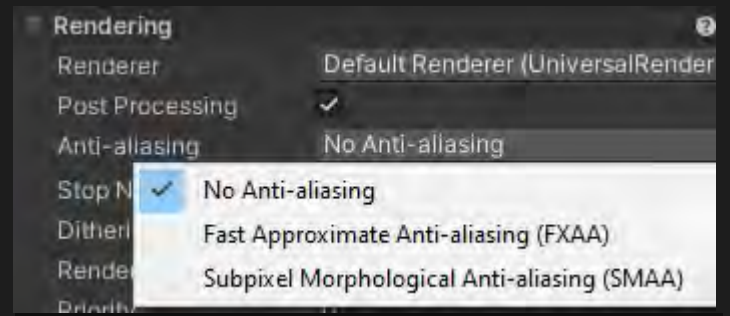
MultiSampling options URP Asset

MultiSampling options Camera

[URP] AntiAliasing

BaseCamera/Rendering/Anti-Aliasing (from cheap to expensive)

- CameraBase/Rendering/PostProcessing ON (works even if there is no Volume component in the scene)
- Fast Approximate AA (FXAA)
  - Only edges
  - Loss of visual clarity especially on objs in motion
  - Cheapest technique > recommended for platforms that don't support motion vectors
- SubPixel Morphological AA (SMAA)
  - Only edges
  - Use pixel color and contrast to sharpen the entire scene
  - Use prev frame data



FXAA SMAA options Camera

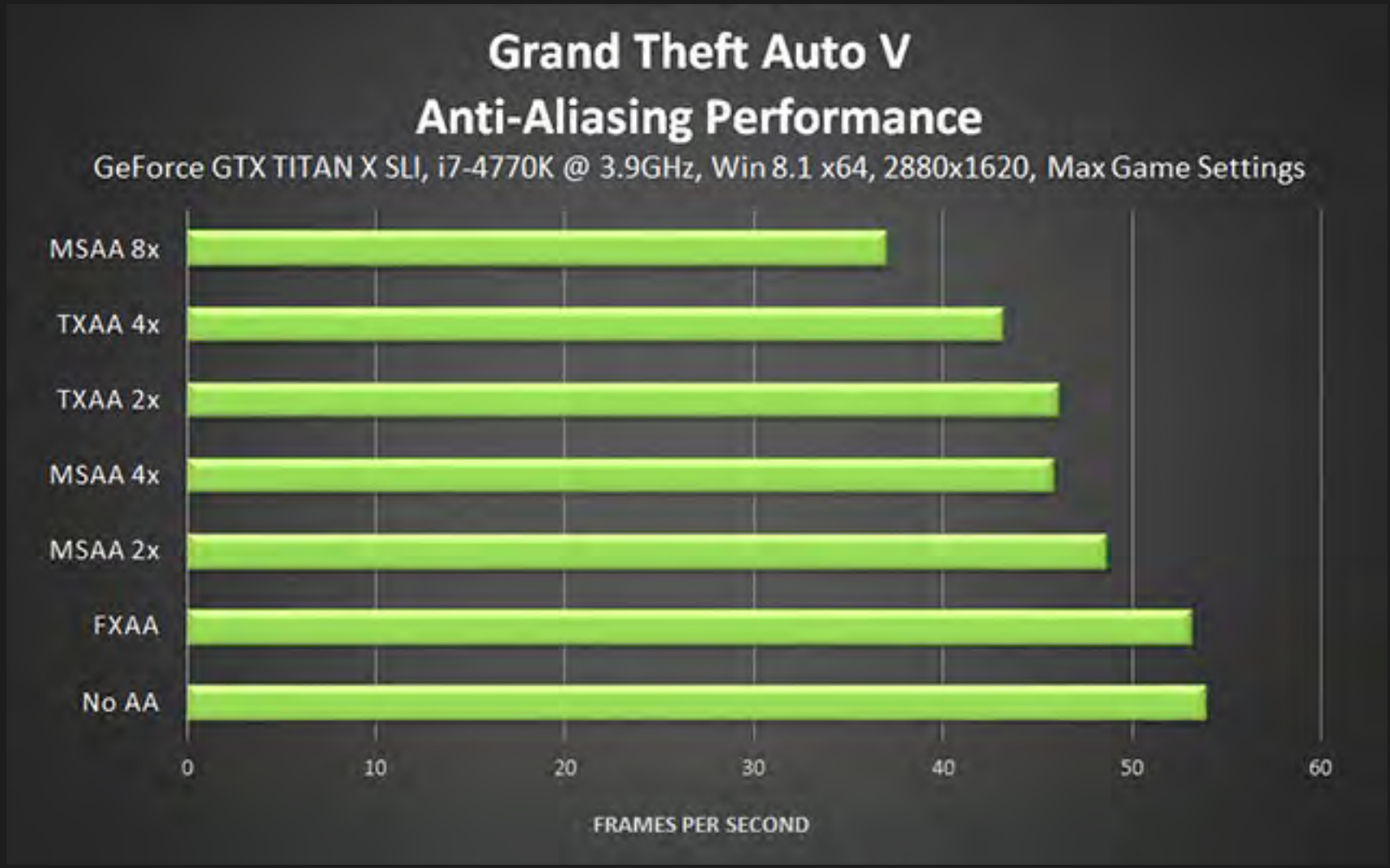
[URP] AntiAliasing

- MSAA and Pprocessing AA do work together and many modern videogames, especially PC, have them both as options to enable at the same time.
  - MSAA can be cheap on some hardware (some mobile GPUs can give up to 4x for free pretty much), it will only solve geometry aliasing (so where triangles start/end) and requires more gpu memory
  - Pprocessing AA, being FXAA or SMAA work on every pixel after it is rendered by 'guessing' how much the pixel should share colour with a neighbour, so this helps all over the screen generally, meaning things like specular aliasing (when small specular highlights from light sources are flickering) and alpha aliasing (vegetation materials with alpha clip textures) can be fixed

- Which one to use comes down to the look of what you are making I think, if it is a modern PBR style with detailed textures and lighting, then use FXAA or SMAA (and eventually TXAA when URP add support) but if you have more simple textures and lots of geometry to add details then MSAA will work better

AA Performances Comparison

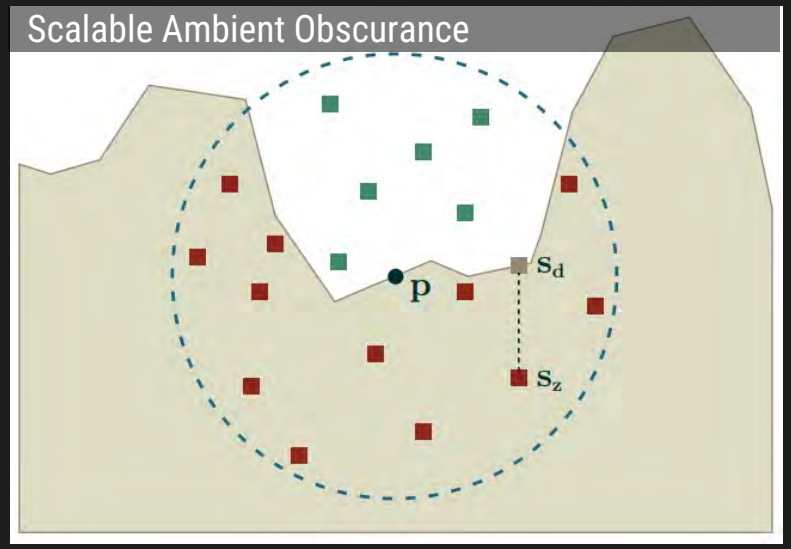
- Multisampling (MSAA)
- Fast Approximate AA (FXAA)
- Temporal Anti-Aliasing (TXAA)



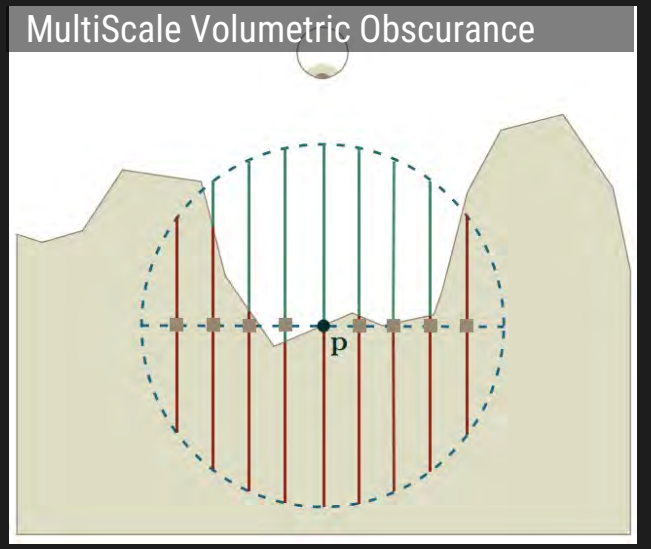
[BIRP] Screen Space Ambient Occlusion

- No CPU Usage
- Scalable Ambient Obscurance
  1. For each screen pixel p, we know p_{xy}
  2. Read p_z from depth map
  3. Choose n random points s around p inside a sphere of radius r
  4. For each s, calculate if it is visible or not
- MultiScale Volumetric Obscurance
- AmbientOnly While using Deferred and HDR Camera, it allows to render AO directly in the Gbuffer.
  - Take into account AO during lighting calculations

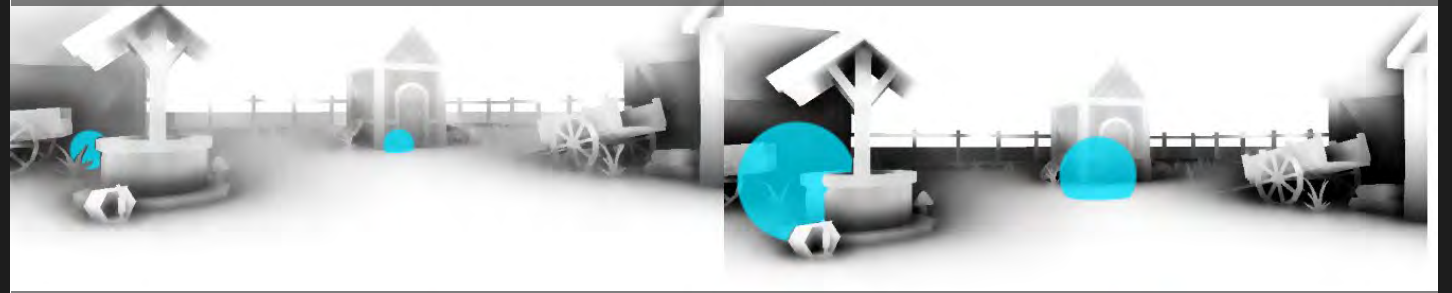
Scalable Ambient Obscurance



MultiScale Volumetric Obscurance



MultiScale Volumetric Obscurance

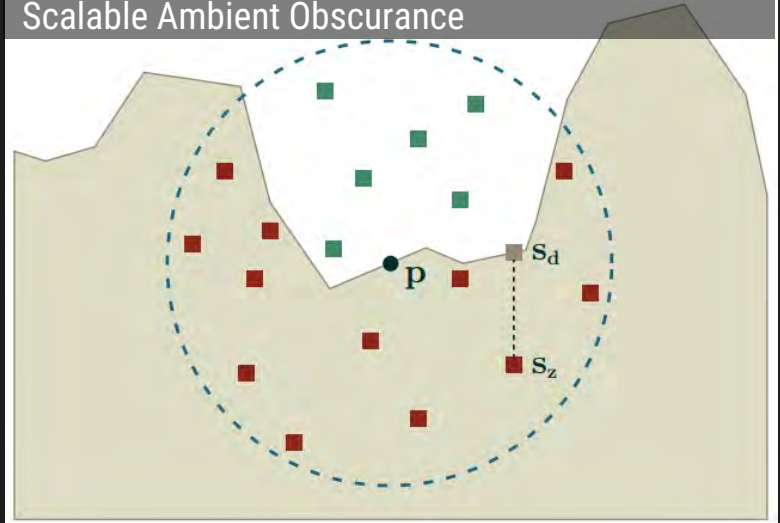


Scalable Ambient Obscurance

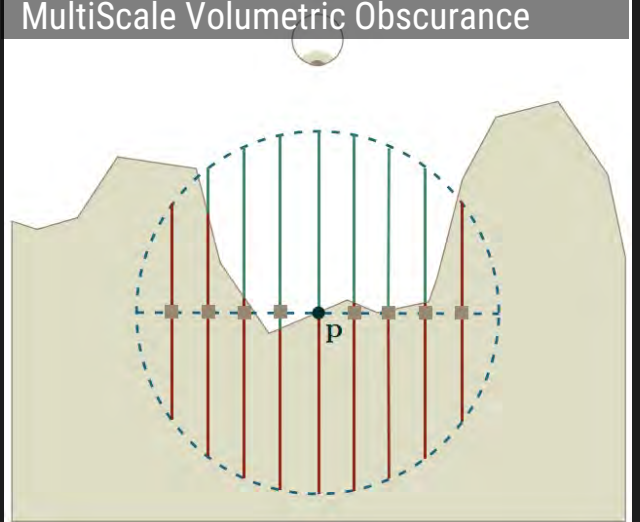
[URP] Screen Space Ambient Occlusion

- No CPU Usage
- Assets/Settings/ForwardRenderer
- AddRenderFeature
- SSAO

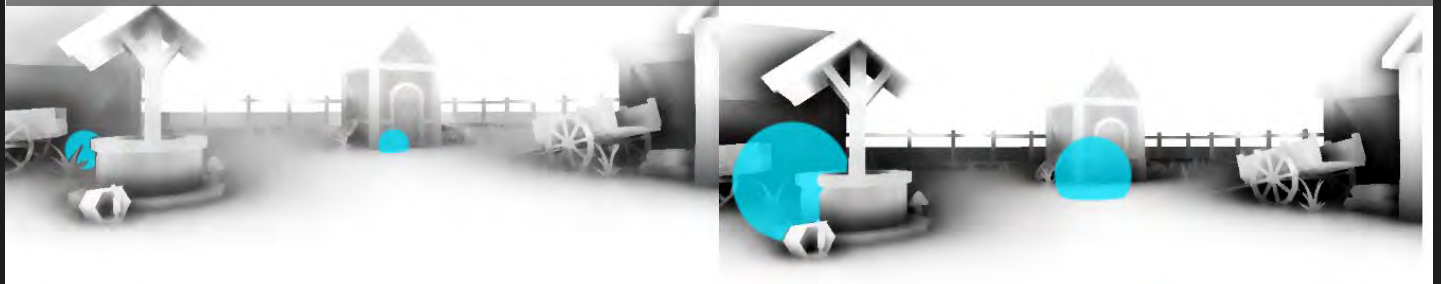
Scalable Ambient Obscurance



MultiScale Volumetric Obscurance



MultiScale Volumetric Obscurance

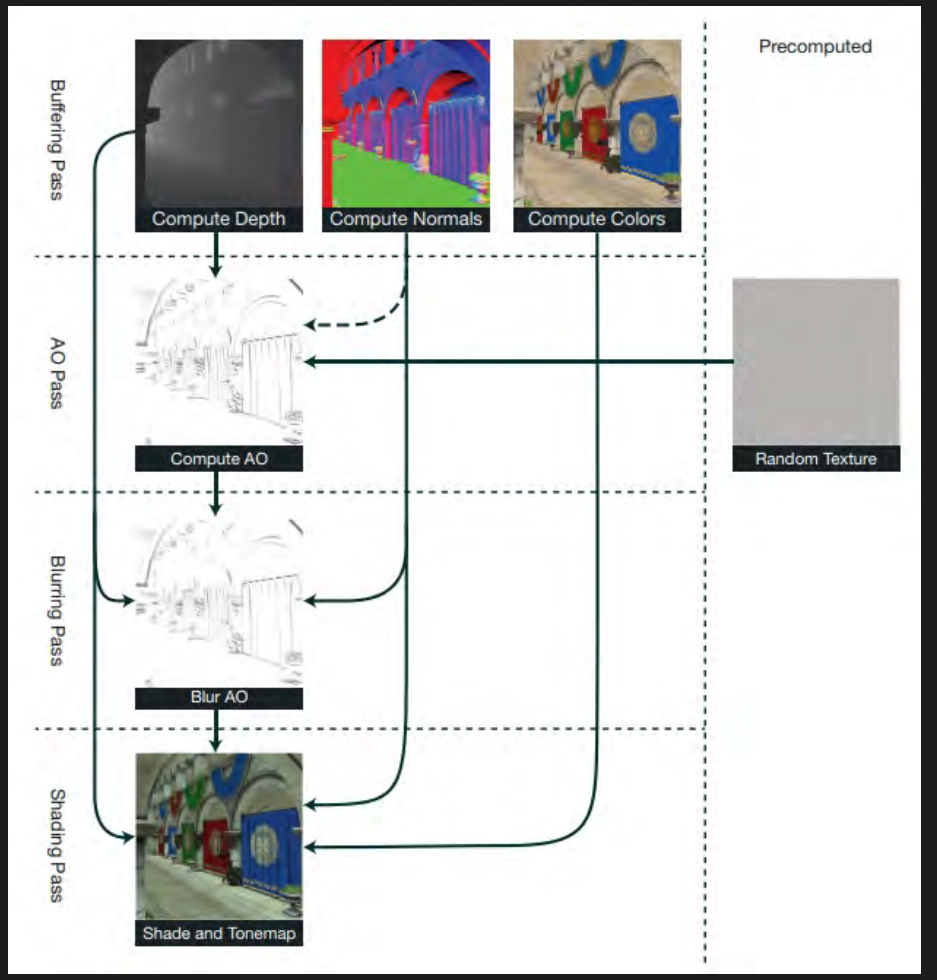


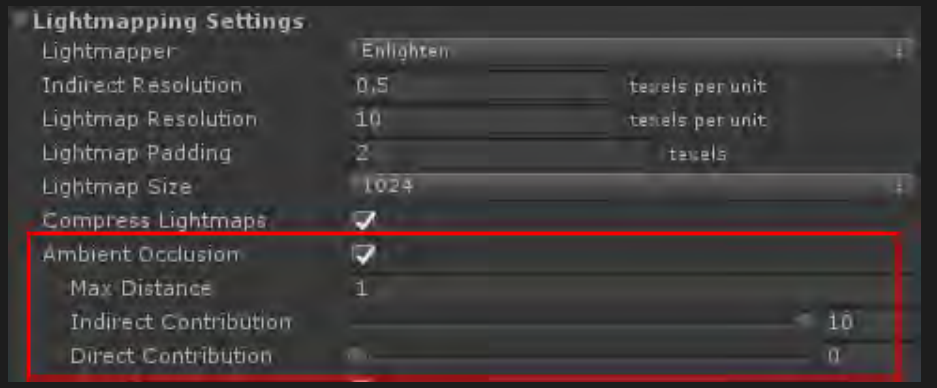
Scalable Ambient Obscurance

Ambient Occlusion

Baked AO

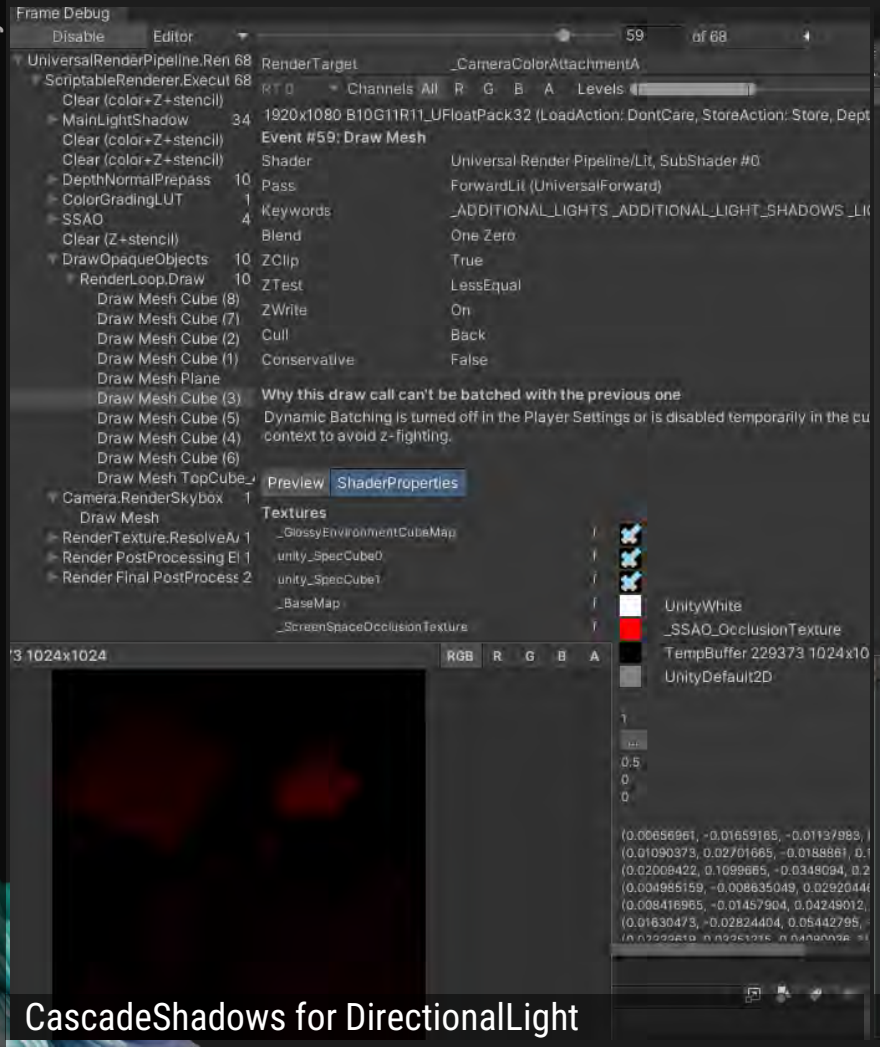
- MaxDistance How far the rays are traced before they are terminated
- Indirect/Direct how much the AO affects indirect/direct light



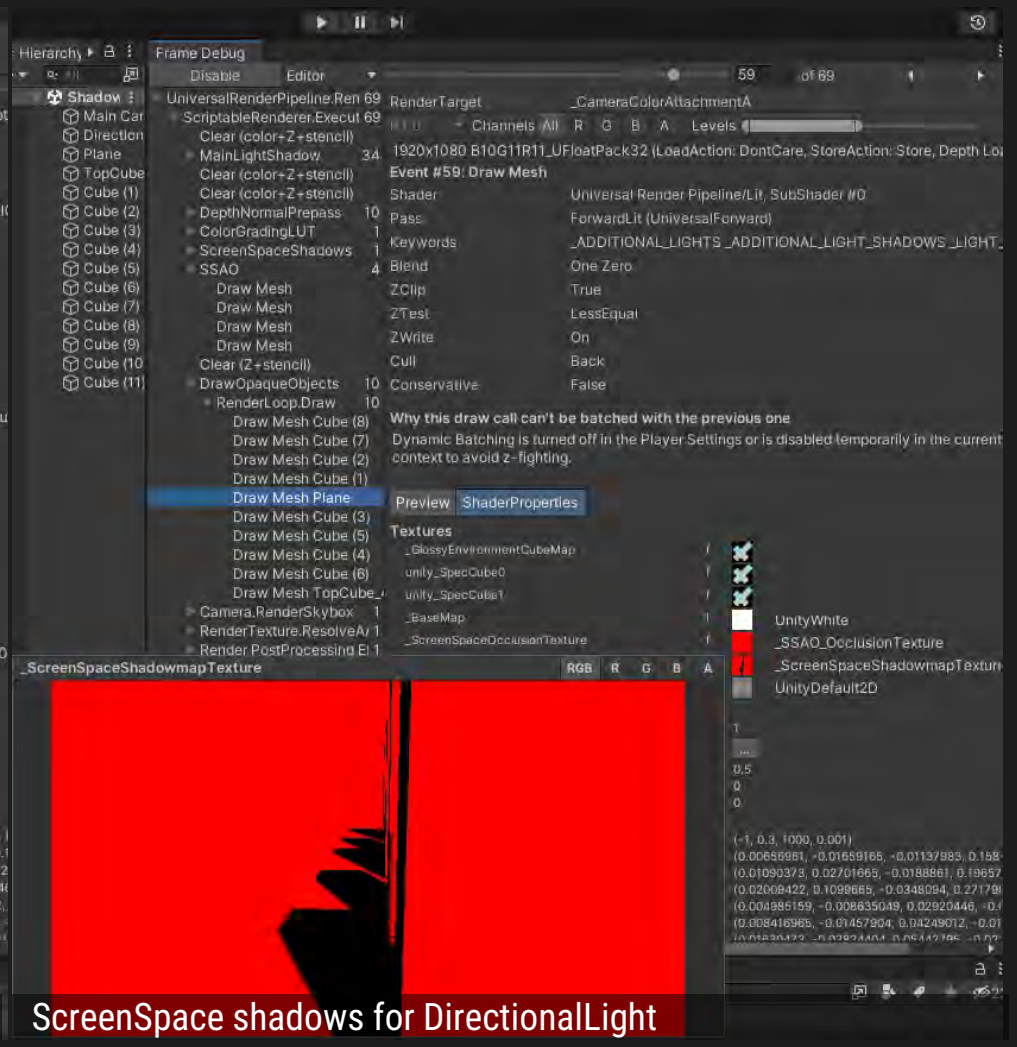


[URP] Screen Space Shadows

- The Screen Space Shadows Renderer Feature calculates screen-space shadows for opaque objects affected by the main directional light and draws them in the scene. To render screen-space shadows, URP requires an additional render target. This increases the amount of memory your application requires, but if your project uses forward rendering, screen-space shadows can benefit the runtime resource intensity. This is because if you use screen-space shadows, URP doesn't need to access the cascade shadow maps multiple times
- Try it using [ShadowPancaking] scene



CascadeShadows for DirectionalLight



ScreenSpace shadows for DirectionalLight

[URP] MotionBlur

- Only blurs camera motions (Not fast moving objects)
  - Try it
  - MBlurTest - CoolingFan Gobj: It spins at high rate, but it doesn't have Mblur
  - Move camera quickly: you should see MBlur

- Clamp Set the maximum length that the velocity resulting from Camera rotation can have. This limits the blur at high velocity, to avoid excessive performance costs. The value is measured as a fraction of the screen's full resolution. The value range is 0 to 0.2. The default value is 0.05

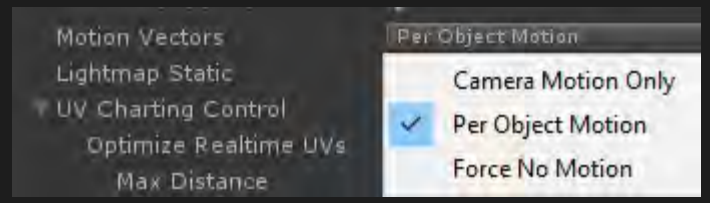
[Built-in RP] MotionBlur

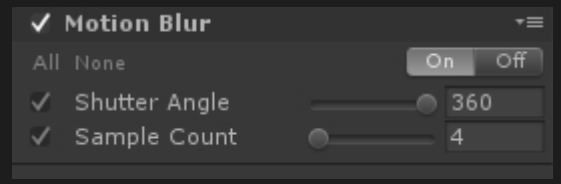
- Occurs when objs are moving faster than the camera's exposure time
- Depends on relative motion

How to achieve MB?
- Adding geometry
- Accumulation buffer
- Velocity buffer / Motion Vectors

- ShutterAngle (from Cinema)
  - 24 FPS
  - 180° shutter = half a circle occluded
  - 270° = a quarter of a circle occluded
  - Degrees are referred to empty space, hence: higher values = more time exposed to light (more blur)
- SampleCount

Example of 180° shutter





Bloom

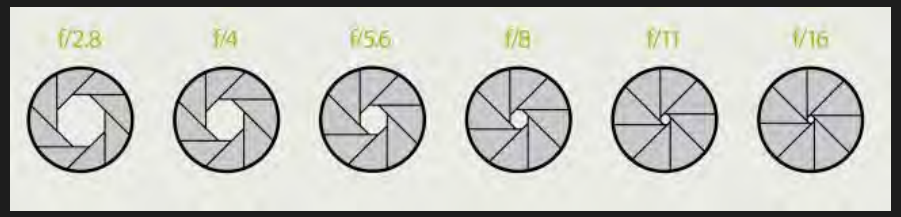
- Occurs when an extremely bright area spills over into near pixels

1. Create an image consisting only of the overexposed objects
   - Can be rendered at lower res
2. Blur this image
3. Compose back to the original image

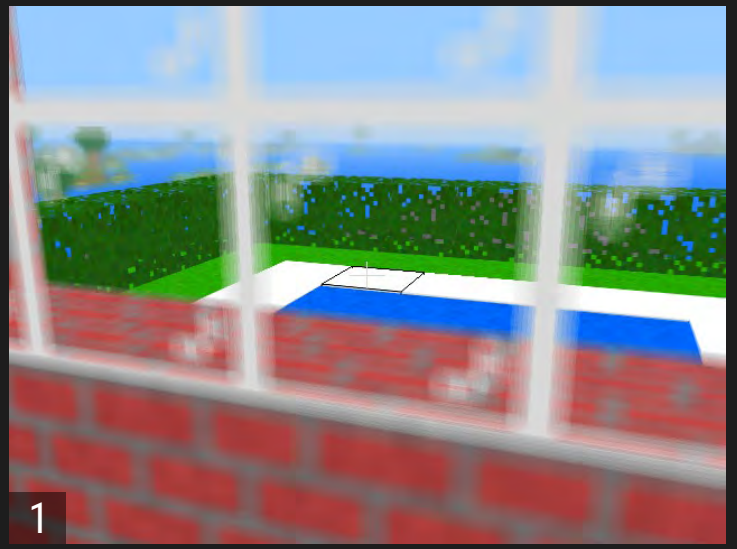
- Objs with greater HDR brightness will receive more bloom

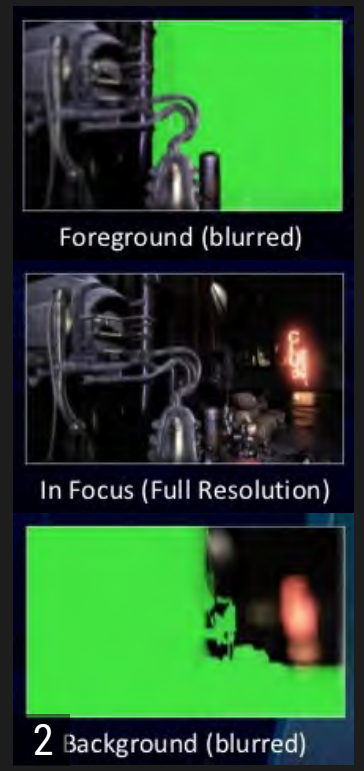
Depth of Field

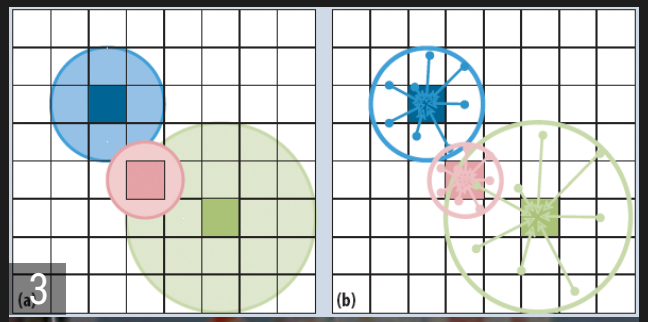
- FocusDistance Distance from the camera where we put our focus point
- FocalLenght Distance between the film and the lens
  - The greater the value, the blurrier the image (i.e. Zoom in = more blur)
- Aperture (f-stop) how much we open our lens in order to let the light in
  - The lower the value, the wider we open the lens, the blurrier the image (because we let more light to pass). Aperture value is the denominator:



- Use accumulation buffer by varying the view and keeping the point of focus fixed > Multiple rendering per image [1]
- Create separate image layers moving near/far clipping plane locations > problems when objs span multiple imgs / uniform blurriness [2]
- Circle of confusion (Bokeh) [3]
- Render the image 3 times: sharp, blurry, blurrier [4]
  1. Use the depth buffer to know the distance from the focal plane
  2. Pixel shader will interpolate among the 3 textures
     - Problems when a sharp silhouette edge from an object in focus is next to a distant, blurred object, or viceversa

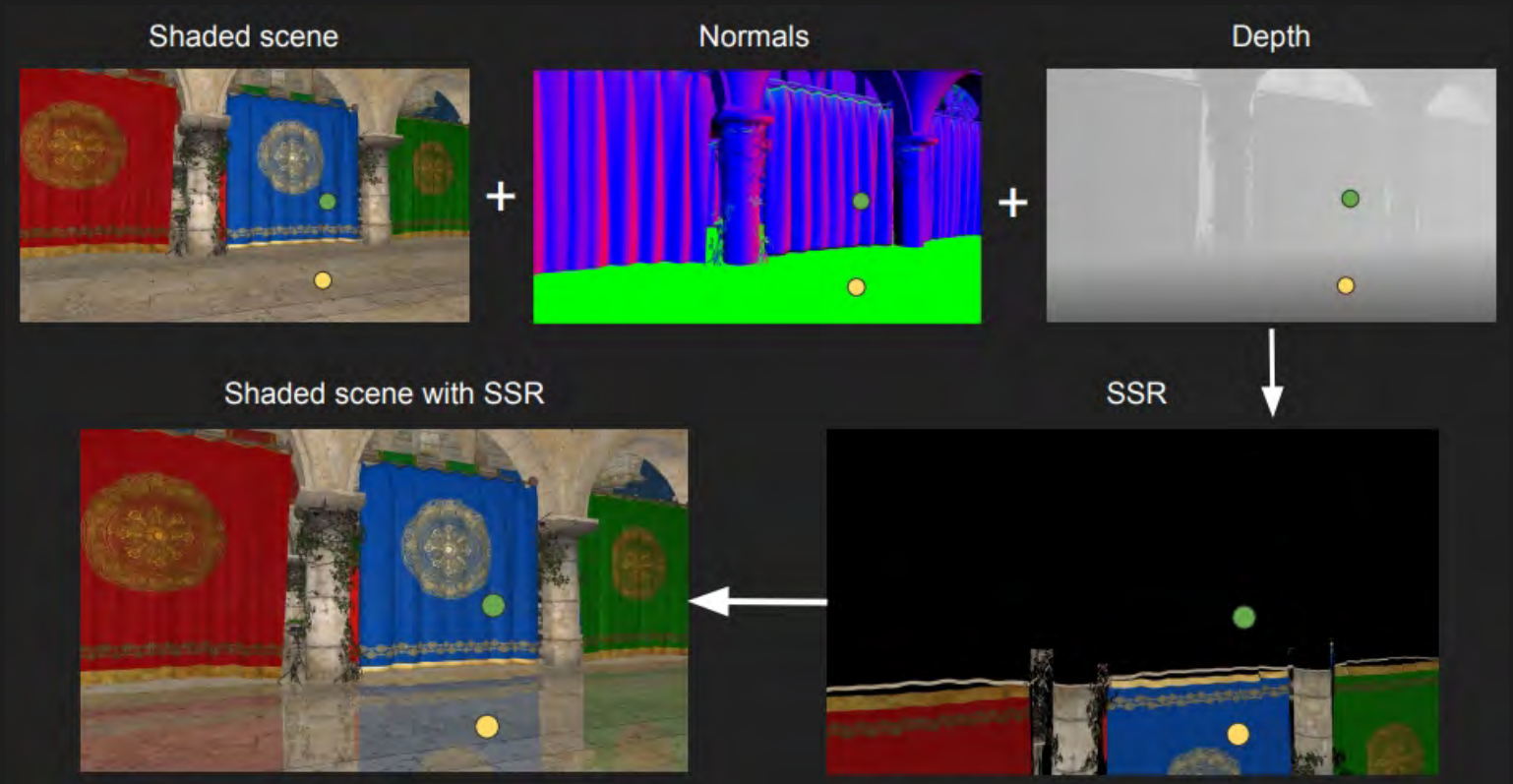






[Built-in] ScreenSpaceReflections

- **Works only in deferred mode**
- Reflection probes with BoxProjections work well, but offer no self reflection [1]
- We can then Reuse Screen-space data [2]
- Basic SSR Algorithm
  - Compute reflection ray
  - Trace along ray direction (using depth buffer)
  - Use color of intersection point as reflection color



[Built-in] ScreenSpaceReflections

- Hidder Geometry problem [1]
- Edge Cutoff problem [2]
  - Edge fading [3]

Panini Projection

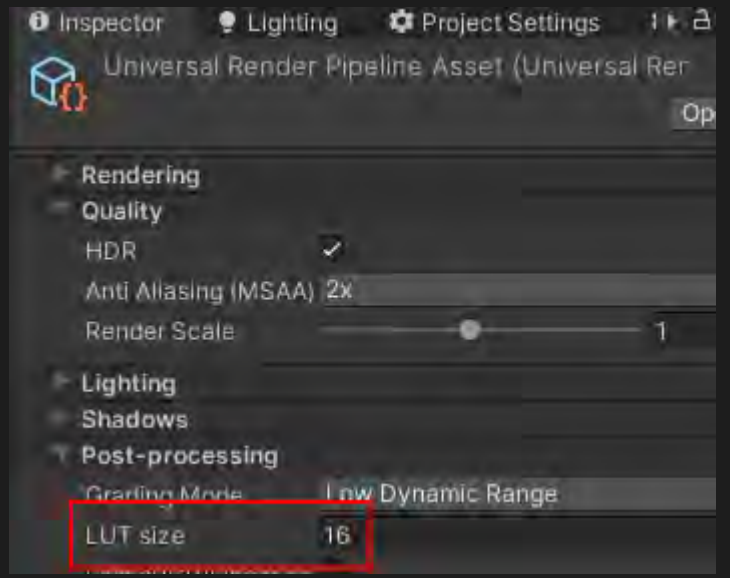
- This effect helps you to render perspective views in Scenes with a very large field of view. Panini projection is a cylindrical projection, which means that it keeps vertical straight lines straight and vertical. Unlike other cylindrical projections, panini projection keeps radial lines through the center of the image straight too
- It is named in honor of Gian Paolo Pannini, an 18th Century Roman painter and professor of perspective, who may very well have used it to draw spectacular views

Color LookUp

- Color Correction Lut (Lut stands for lookup texture) is an optimized way of performing color grading in a post effect. Instead of tweaking the individual color channels via curves as in Color Correction Curves, only a single texture is used to produce the corrected image. The lookup will be performed by using the original image color as a vector that is used to address the lookup texture.
- Advantages include better performance and more professional workflow opportunities, where all color transforms can be defined in professional image manipulation software (such as Photoshop or Gimp) and thus a more precise result can be achieved.

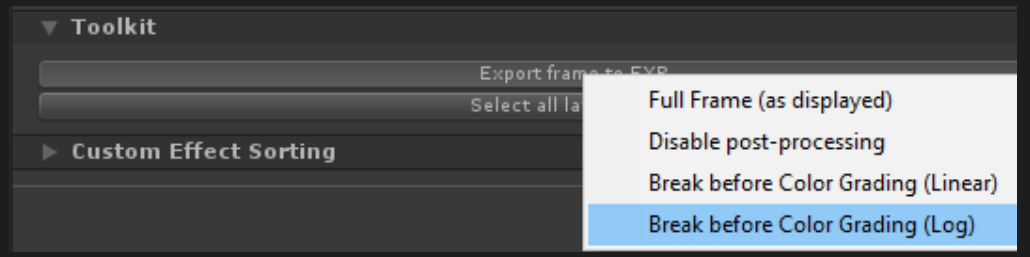
- Your LUT texture should:
  - 2D Texture
  - Have no sRGB flag in import texture
  - Same height of In URP Asset/PProcessing/LutSize

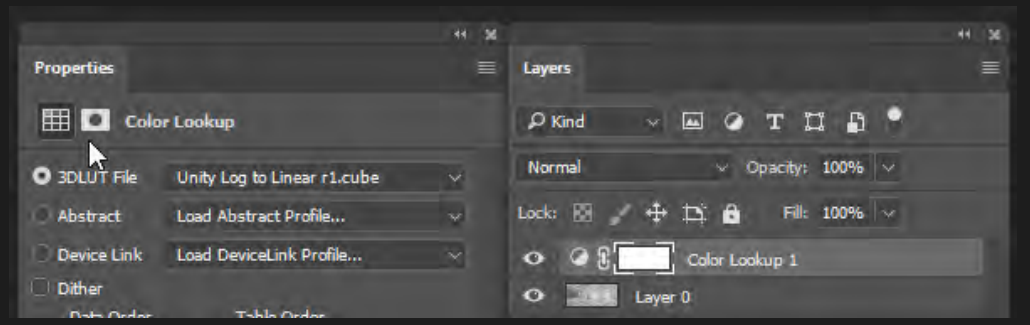
- Try with [Other/3DUnwrappedContrastEnhance.png], and a LUT size of 16

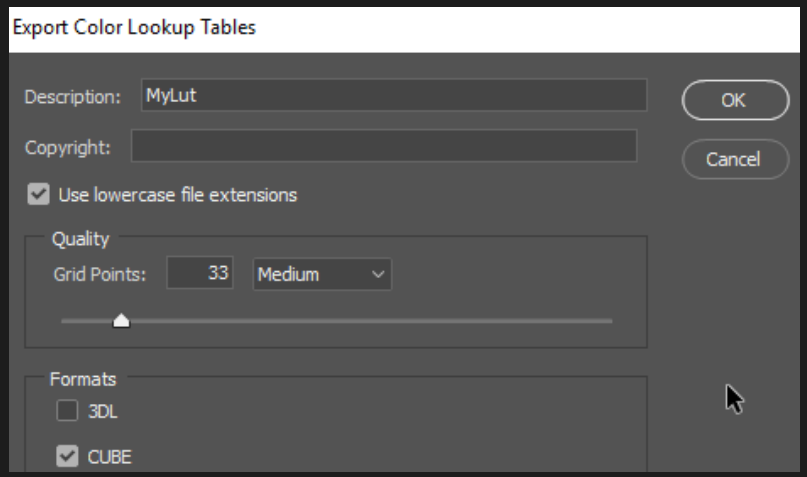


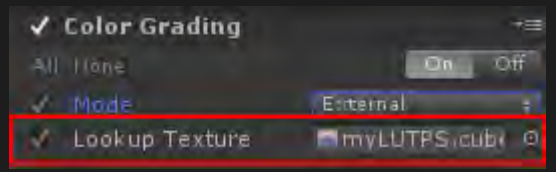
LUT Authoring in PS

1. Export the current frame in Log as EXR without color grading applied (remember the 0.45 gamma?)
2. Open it in PS. It should look like a Log footage (0.45 gamma)
3. Add a non-destructive color LUT adjustment layer. Use the Unity LogToLinearLUT that you'll find in /PostProcessing/Textures/Cubes/
4. Start grading your image.
   - Use only non-destructive adjustment layers. filters that has an effect on neighbor pixels (blur) can't be stored in LUTs
5. Transform the original frame layer into background Layer>New>BackgroundFromLayer
6. Once you're done, esport the LUT File>Export>ColorLookupTables using the settings in the image
7. Back in Unity, it will detect the CUBE and automatically convert it to a Texture3D asset. Grab this texture and populate the Lookup Texture field in the color grading effect.







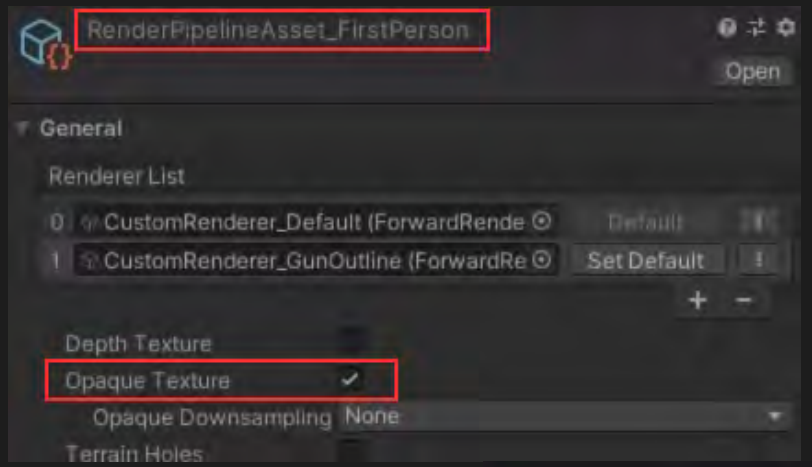


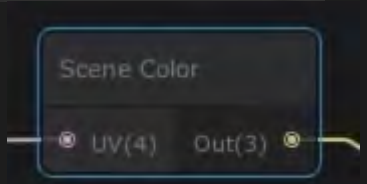
URP Camera stack VFXGraph

- Different cameras could render different object layers and have personalized PProcessing effects! So now you can finally easily do selective bloom or similar things

Examples

- UsePass&GrabPass: if we have a cockpit with transparent panels, it could be useful to use a shadergraph VFX only on that panels
  - Turn on Opaque texture on your RenderPipelineAsset used by the cockpit camera
  - In ShaderGraph you'll have the opaque pass of your camera rendered as a texture in the SceneColor node (SceneDepth if you want to use depth image)
- You can make an Xray effect to see enemies behind walls (See URP Examples Git)





Fog

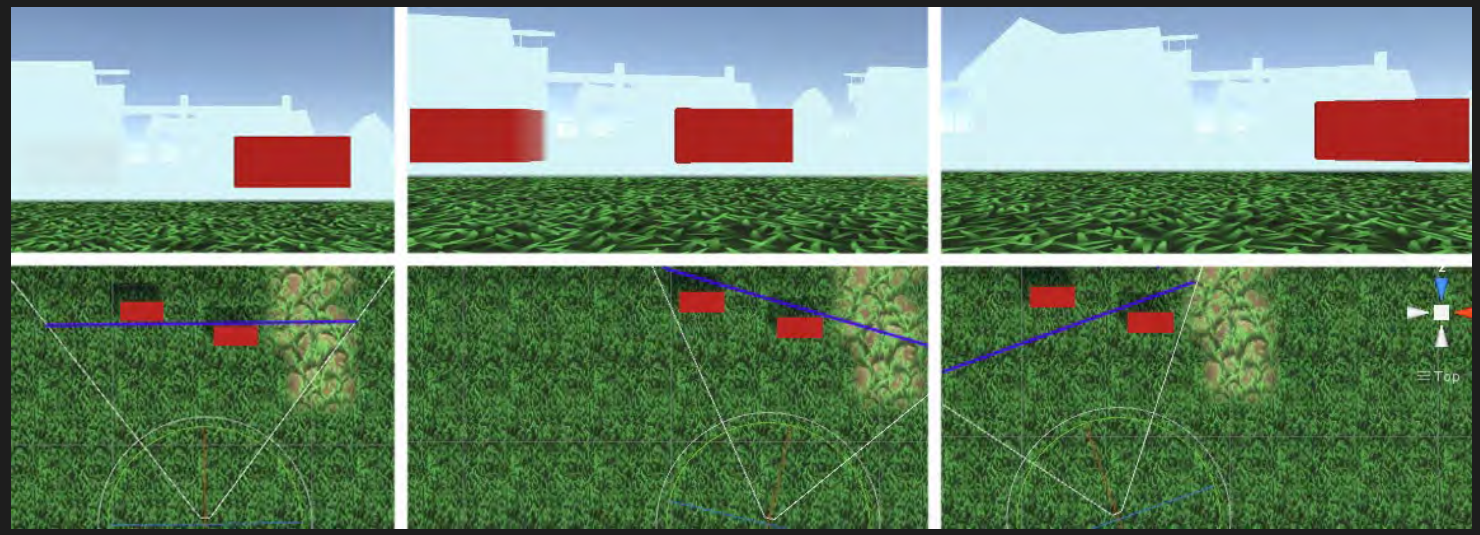
- Simple atmospheric effect performed at the end of the rendering pipeline
- Can be Linear/Exponential/ExpSquared
- Its distance is calculated parallel to near clipping plane (simple effect)

- Enable it under LightingSettings/OtherSettings/Fog

$$f = e^{-(density \cdot z)} \quad (GL\_EXP)$$

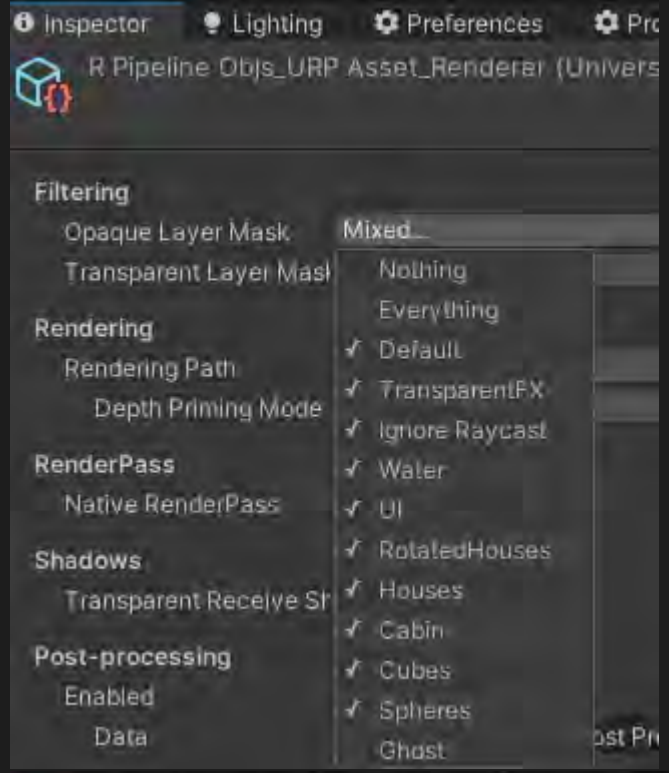
$$f = e^{-(density \cdot z)^2} \quad (GL\_EXP2)$$

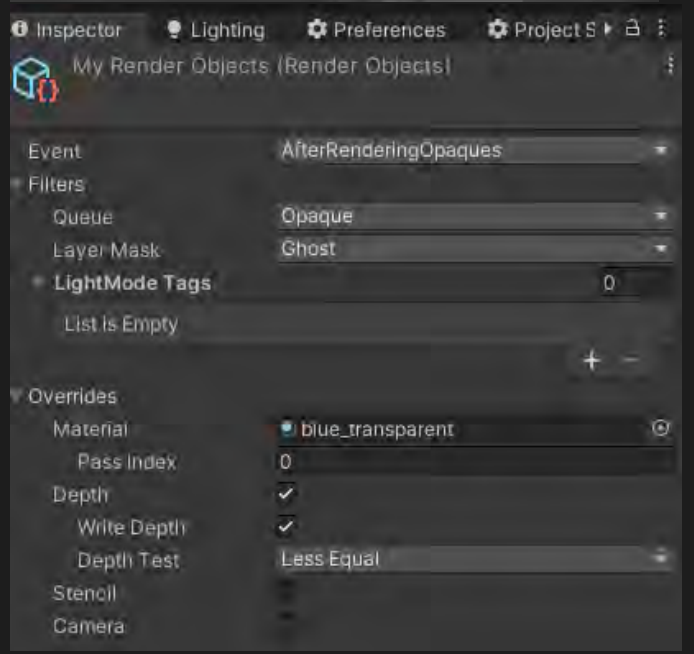
$$f = \frac{end - z}{end - start} \quad (GL\_LINEAR)$$

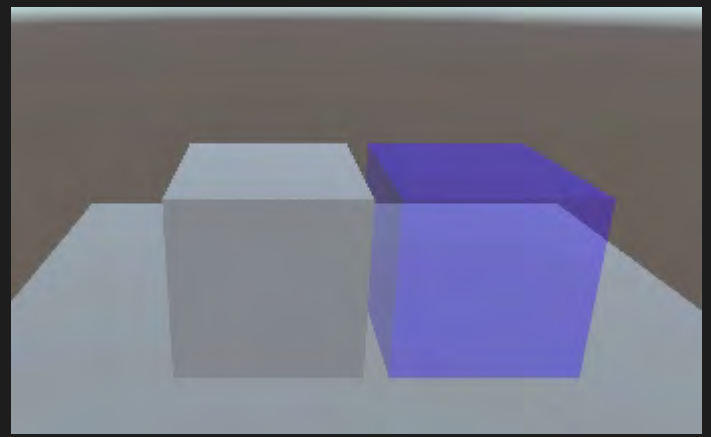


RenderObjects

- [RPipelineFeatures_00]
- Ghosting
  - I want to render GhostCube with a transparent material while it belongs to Ghost layer
  - Set GhostCube layer to GhostLayer
  - Add RPipelineObjs_URPAsset_Renderer a GhostRObjs feature
    - Renders objs with Ghost layer with a different material: blueTransparent (mat with SurfaceType: Transparent)
      - NB: SurfaceType: Transparent doesn't mean that belongs to Transparent Queue!
    - Now FrameDebug shows that GhostCube renders 2 times: first time using an opaque mat, scond time with blueTransparent mat
    - Set RPipelineObjs_URPAsset_Renderer filtering/OpaqueLayerMask w/o Ghost
    - Now FrameDebug shows that GhostCube renders 1 time, using with blueTransparent mat, but...
    - ...Skybox is rendered after the Ghostcube, and since Ghostcube is a transparent obj, doesn't write Depth buffer
      - First solution: Add Depth ON, Write Depth ON
      - Second solution: Event/AfterRenderingSkybox
        - Notice that blending result is different!

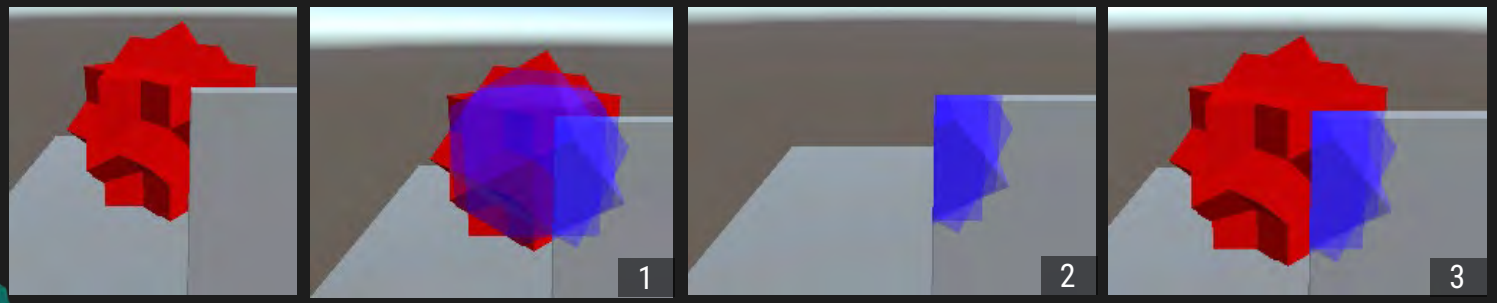


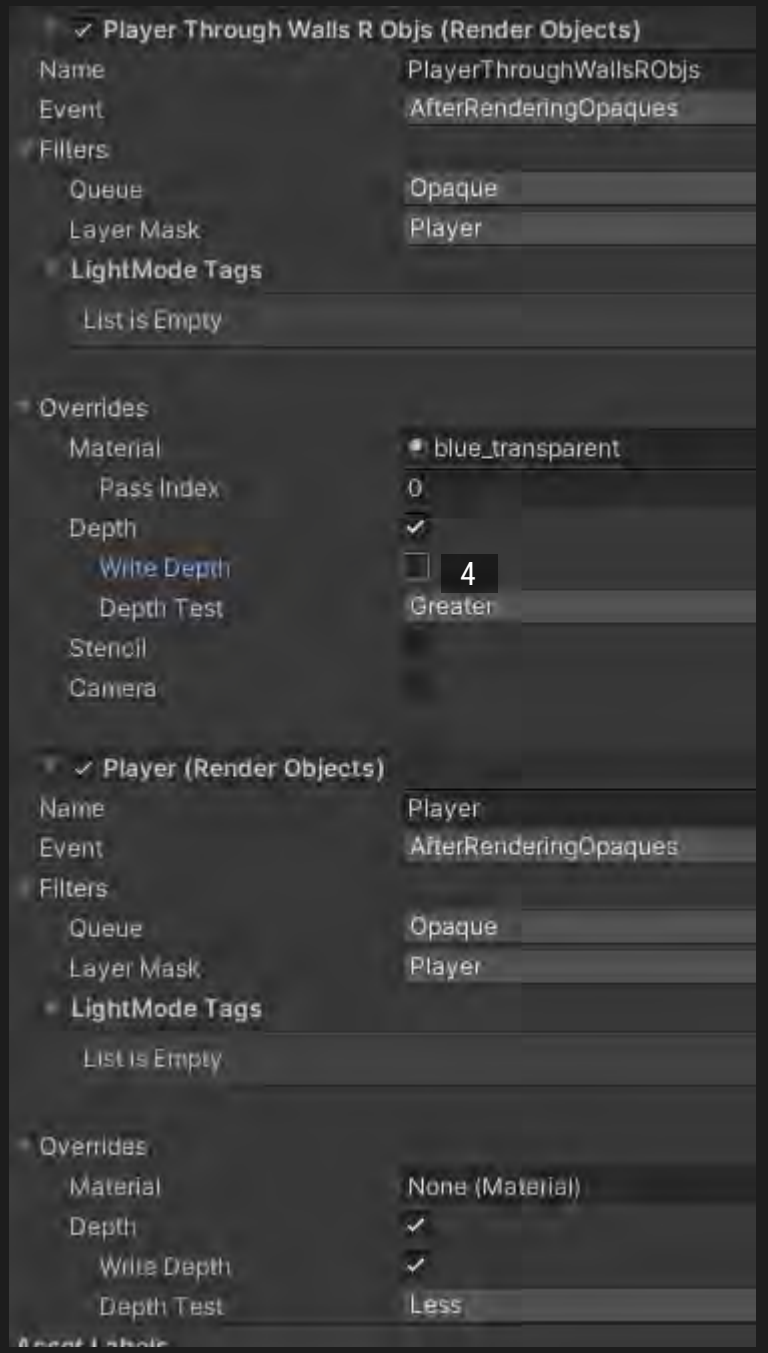




RenderObjects

- [RPipelineFeatures_00]
- Through Walls
  - I want to render PlayerCube with a transparent material while it is occluded
  - Set PlayerCube layer to Player
  - Add RPipelineObjs_URPAsset_Renderer a PlayerThroughWallsRObjs feature
    - Renders objs with Player layer with a different material: blueTransparent (mat with SurfaceType: Transparent)
    - Add Player layer to RPipelineObjs_URPAsset_Renderer filtering/OpaqueLayerMask
    - Now PlayerCube is rendered with blueTransparent mat
    - Add Depth ON, DepthTest Greater
    - Now PlayerCube is rendered with blueTransparent only if Depth test is Greater, but there is some Zfighting [1]
    - Remove Player from Filtering/OpaqueLayerMask [2]
    - Add RPipelineObjs_URPAsset_Renderer a Player feature
    - Overrides/Depth ON, WriteDepth ON, DepthTest Less [3]
      - PlayerThroughWallsRObjs Change WriteDepth OFF [4]



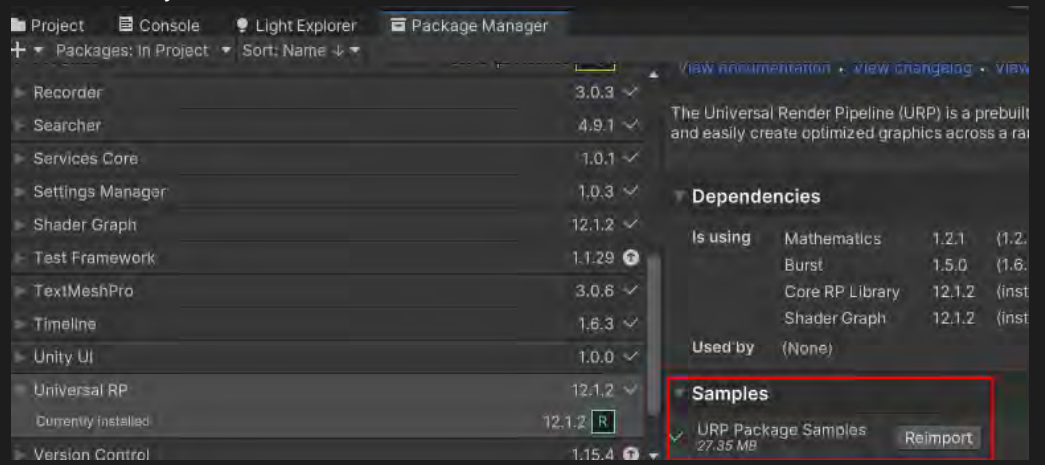


URP Samples / Cam stacking

- Once imported, they are in Samples/Universal RP

3DSkybox

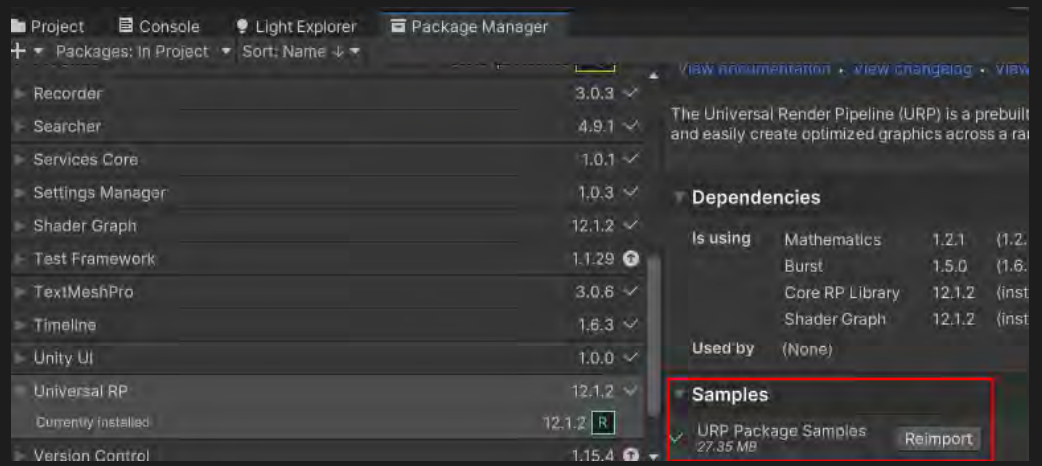
- MainCamera has AutoLoadPipelineAsset
  - Has [ExecuteAlways], and Update the pipeline if the ProjSettings/Quality/Rendering/RPAsset is not set correctly
  - Use this kind of script everytime you know that the scene use a different RPAsset!
- PlanetSkyboxCamera
  - culling: transparentFX
  - Spheres under PlanetSkybox layer: TransparentFX
- CitySkyboxCamera
  - Culling: houses
  - Floor and other building layer: houses
- MainCamera (base)
  - Culling: cabin
  - Porch layer: cabin



- how an overlay camera can be used to draw parts of the 3D world that are inaccessible from a balcony, such as distant buildings and planets
- Intuitively, you might want to have the base camera draw the cityscape and then have an overlay camera draw the balcony on top, but this could cause potential overdraw. Instead, you'll see that the balcony camera is used as the base camera, and the stencil settings in the Pipeline Asset are set so that the overlay camera only draws the cityscape in pixels that haven't already been written by the base camera.
- More specifically, the balcony camera specifies the value of one to the stencil buffer, and the overlay camera only draws in areas where the stencil buffer's value differs. In order for this to work, it is imperative to uncheck the Don't Clear option on the overlay camera, as the stencil buffer is embedded in the depth buffer and would otherwise be cleared with it.

URP Samples / Decals

- Blob shadow
  - Decals are used under the capsule to create the impression of a shadow
- Proxylighting
  - In the Proxy Lighting scene, decals are also used to modify the emissive color of surfaces in a cone shape. This makes surfaces appear as though they're being lit by a spotlight, even though there are no real-time lights involved in this example
- Noise

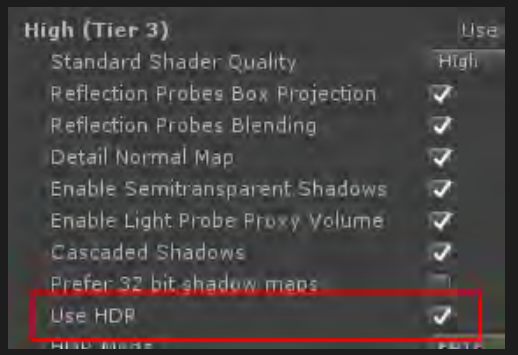


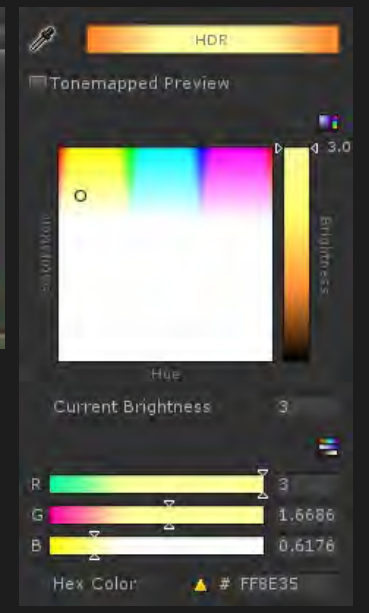
HDR

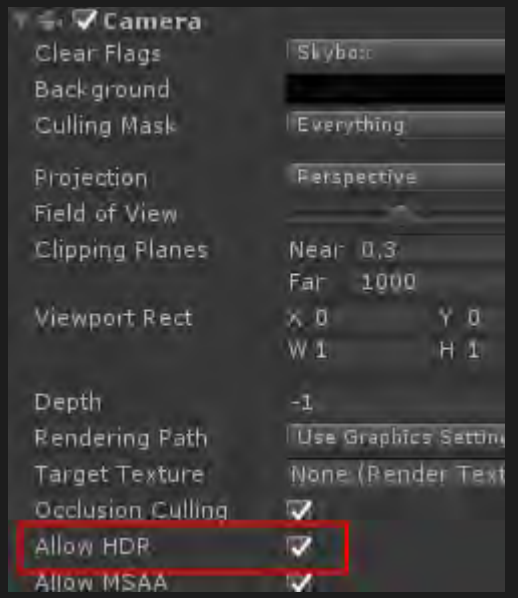
- In Standard rendering RGB values are in [0,1] range (LDR), where 1 is the max intensity for the display device
- Human eyes works differently
  - Adjust to local lighting conditions: the same obj could look white or grey, in a dimly lit room or in full daylight
  - More sensitive to difference between darker shades than lighter shades
- HDR allow to store color values outside the [0,1] range
- When HDR is active, the scene is rendered into an HDR image buffer which can accommodate pixel values outside the [0,1] range. This buffer is then used by post-processing effects such as the Bloom
- Post-processing effects (like Bloom) working in HDR will improve realism of the entire scene, even if at the end the output buffer must be converted again in LDR (Because our monitor is not HDR capable) via Tonemapping

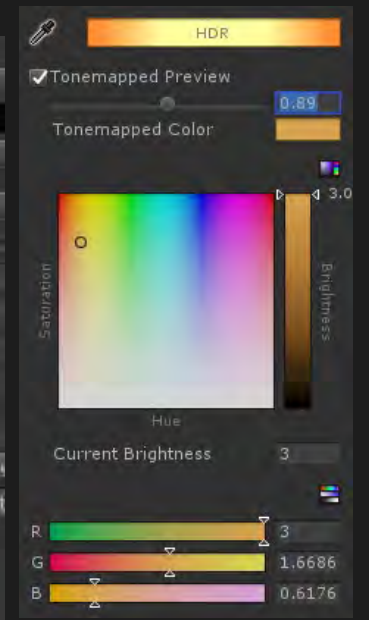
To enable HDR
- Camera/AllowHDR
- ProjectSettings/Graphics/UseHDR

ColorPicker
- Can set the brightness to a value > 1
- TonemappingPreview allows to preview the final LDR color after Tonemapping (use the same Exposure value in the Tonemapping Script in the PostProcessing Stack)



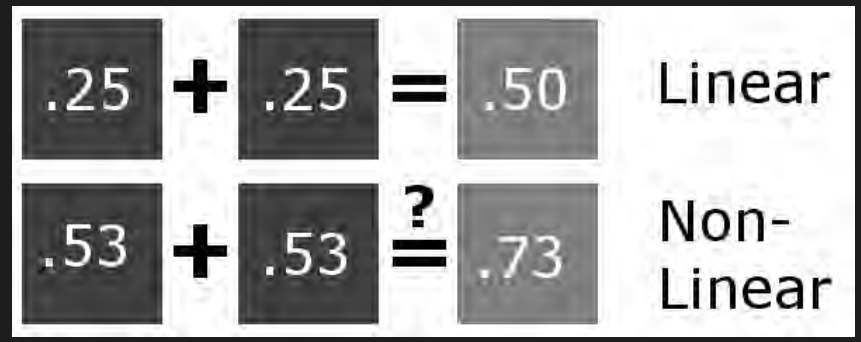


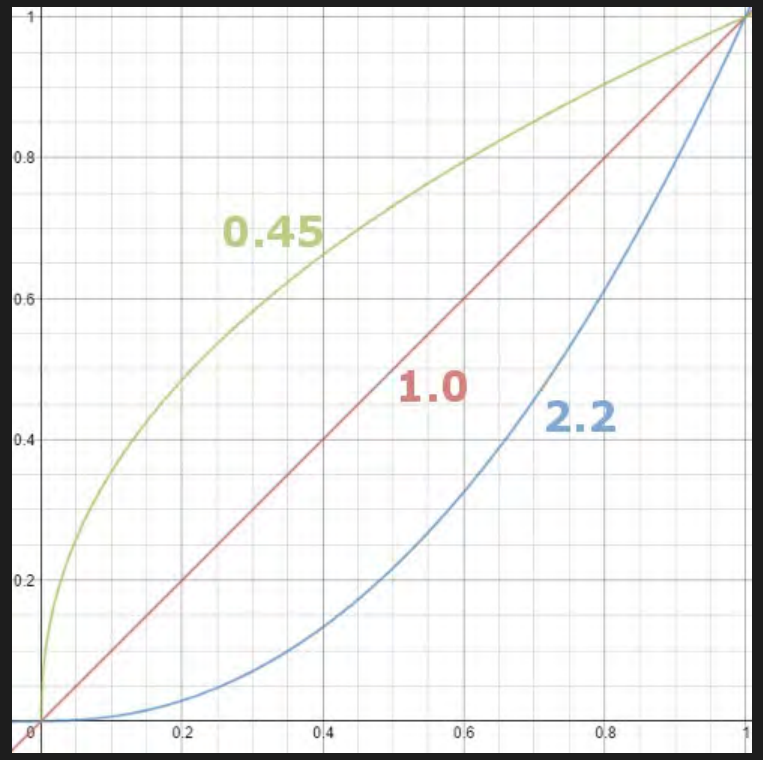


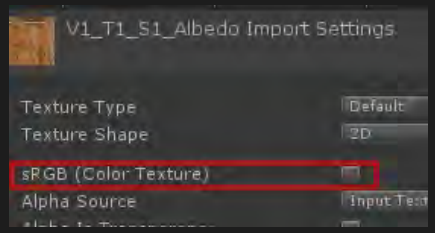


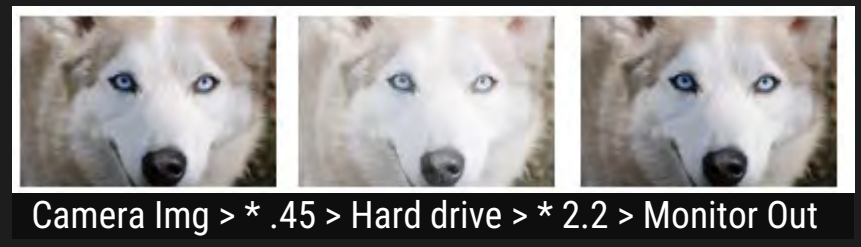
Gamma & Linear space

- [URP] By default, URP uses a linear color space while rendering. You can also use a gamma color space, which is non-linear. To do so, toggle it in the Player Settings









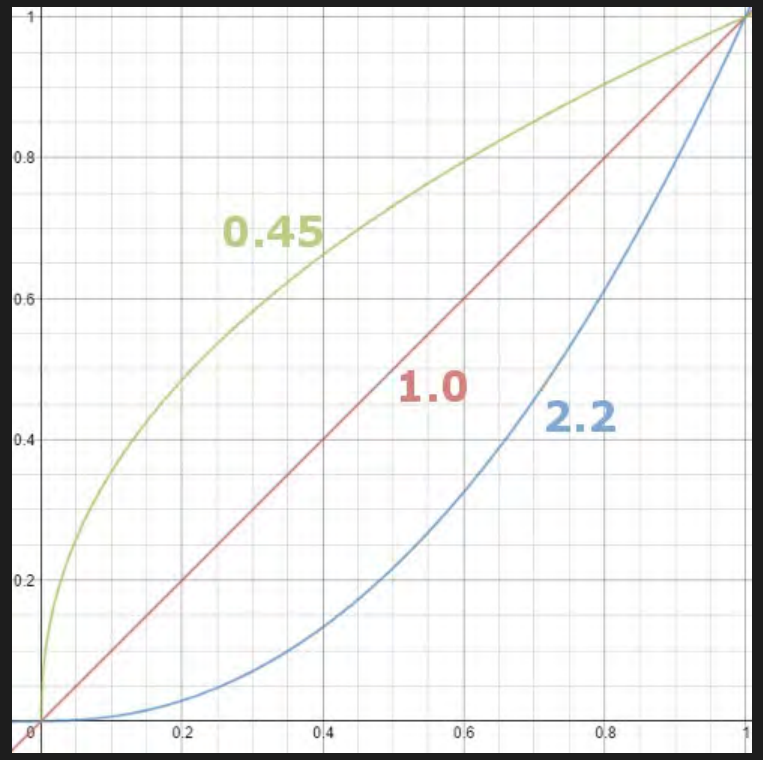
Camera Img > * .45 > Hard drive > * 2.2 > Monitor Out

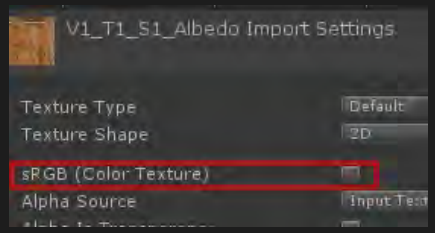
Gamma & Linear space

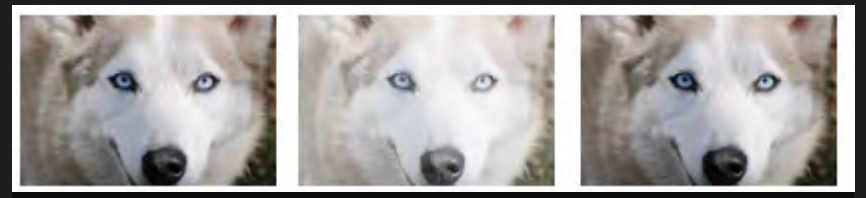
- Linear colors can be added and multiplied correctly: 0.2 + 0.2 = 0.4
- If we perform a pow() on every pixel color, we're applying a GammaCorrection
- Why we need gamma correction?
  - Screens have no linear response to intensity > Our monitors have a gamma of about 2.2 (sRGB)
  - Eyes are more sensitive to difference between darker shades than lighter shades

- Imgs JPEG, PNG, BMP are stored with gamma 0.45 (images in gamma space)
- Imgs EXR TIFF HDRI could be encoded with gamma 1.0
- Our monitors apply a gamma corrections of 2.2

- If sRGB flag is turned on in Texture import inspector (the default value), the texture is assumed to be stored with gamma 0.45
- This means that
  - If the texture is actually encoded with a gamma 0.45 texture, everything is ok
  - If the texture is an EXR with a linear space encoding, it will be displayed darker







Camera Img > * .45 > Hard drive > * 2.2 > Monitor Out

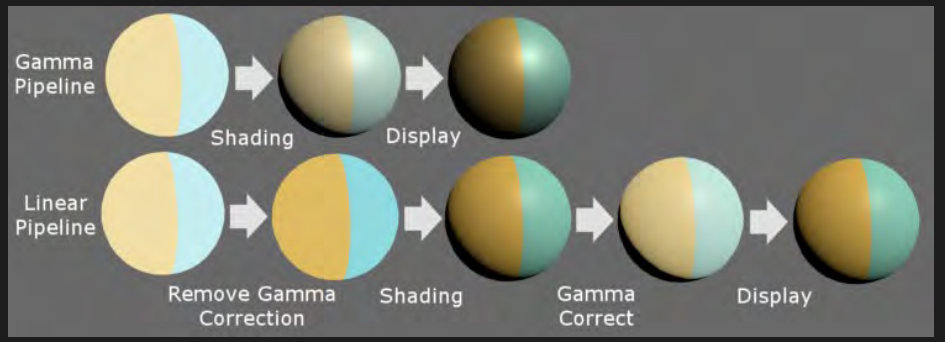
Gamma & Linear space

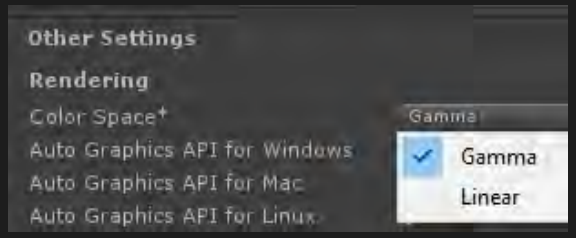
Unity Pipeline

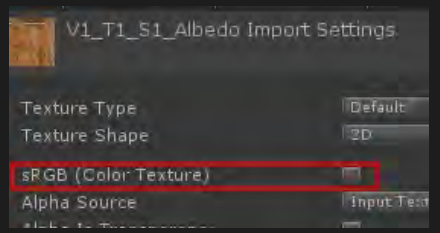
- Gamma The texture is sampled without corrections, then we apply shading in gamma space > Monitor
  - sRGB flag won't have any effect. The texture is read as it is
- Linear Shaders receive non-gamma corrected textures
  - Remove gamma (if texture is imported with sRGB flag ON) > Apply shading in linear space > Gamma correct > Monitor

- In Linear space, sRGB flag must be OFF for
  - Lookup Textures, masks, and other textures with RGB values that mean something specific and have no gamma correction applied to them

- Edit/ProjectSettings/Player/OtherSettings
- Default is Gamma (linear rendering is not supported on all platforms)
  - No fallback to gamma when linear rendering is not supported > Player quits
- You can support linear with your own shader, but it is computationally expensive!
  - Apply the pow() function to gamma corrected input textures > transform the inputs to linear space > Perform shading calculations > Apply pow() again before returning the result to put it back in gamma space







Color Grading

- Allows to correct the color and luminance of the final image (Instagram)

LDR

- White, Tone, Channel, Trackballs

HDR

- Tonemapping
  - HDR values > range suitable for the screen
  - If not applied, values color intensities above 1 will be clamped at 1

External

- ColorCorrectionLUT (LookUp Texture) Cheap way to use complex Color Correction curves: a single texture is used to produce the corrected image, using original image color as sampling cords
  - 2D Represents an unwrapped volume texture (imagine an image sequence of depth slices)
  - 3D Textures
    - Enable Read/Write support
    - Disable texture compression

Fog

- Simple atmospheric effect performed at the end of the rendering pipeline

1. Can be Linear/Exponential/ExpSquared
2. Once f is calculated, assuming that c_p is the final color of the pixel, c_f is the color of the fog, c_s is the original color of the pixel, then

$$c_p = f\,c_s + (1 - f)\,c_f$$

- Enable it under LightingSettings/OtherSettings/Fog

$$f = e^{-(density \cdot z)} \quad (GL\_EXP)$$

$$f = e^{-(density \cdot z)^2} \quad (GL\_EXP2)$$

$$f = \frac{end - z}{end - start} \quad (GL\_LINEAR)$$