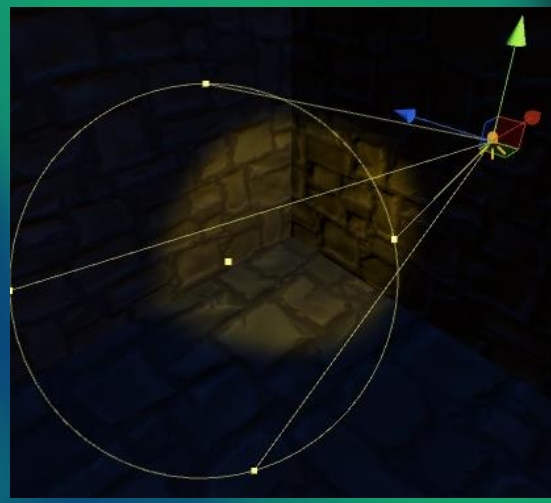
Light types

- SpotLight / PointLight
- Directional light
  - Can be linked to procedural Skybox (default scene behavior)
- Area light
  - Only in Baked
  - Light emitted in all directions uniformly across their surface area, from one side

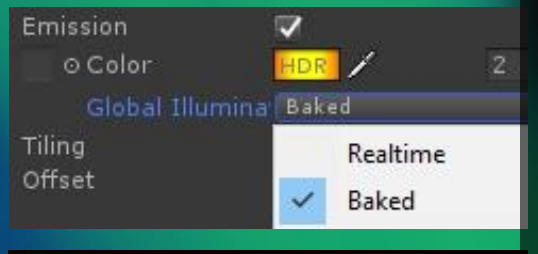


http://bit.ly/LIGHTING_END_WO_FINAL_EX

Light types

- Emissive
  - Emission is a StandardShader property
  - Only in PrecomputedGI/Baked (StandardShader flag)
  - Emission will be received only by lightmap static objects
- Ambient
  - Lighting window settings

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



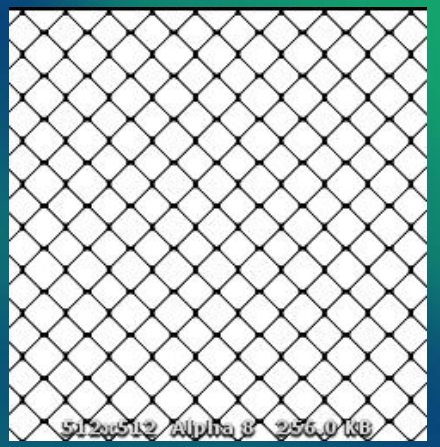
Cookies

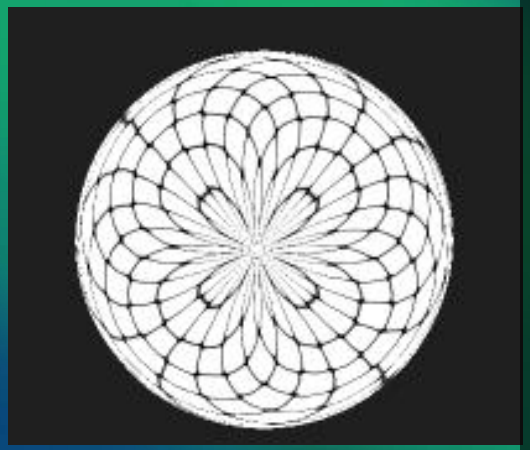
- Environment effects (Cinema, Theatre)

1. Set texture as Cookie Asset
2. Specify for what kind of light and what kind of mapping do you prefer
    1. MirroredBall / 6 Frames / LatLong / Auto
3. Set Appropriate WrapMode

Used for

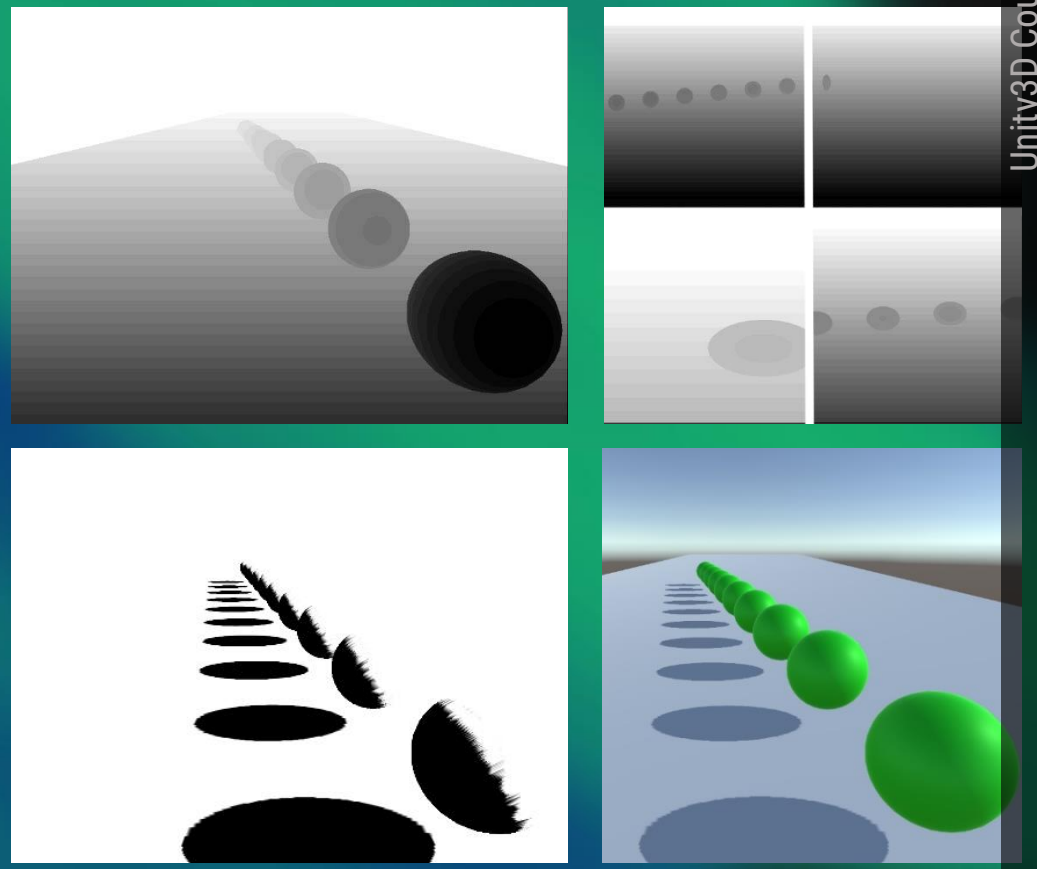
- Change the shape of a light.
    - A dark tunnel with striplights along the ceiling
    - Monitor screen glow should be restricted to a small box shape
- Can also incorporate grayscale levels. Useful for simulating dust in the path of the light
    - Torch light glass usually contains ridges that create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





Shadow mapping

1. Render the entire scene, but only the depth information of each fragment > screen resolution texture output
2. Values in the 0–1 range (clip space, MVP Matrix)
3. For each light, render the scene from the light source POV (again only the depth information of each fragment) - Directional light is orthographic
4. If we are using CSM, the scene is rendered N times per light
5. Make the comparison between camera depth buffer and light depth buffer
    1. set Lit texels are set 1, and shadowed texels to 0



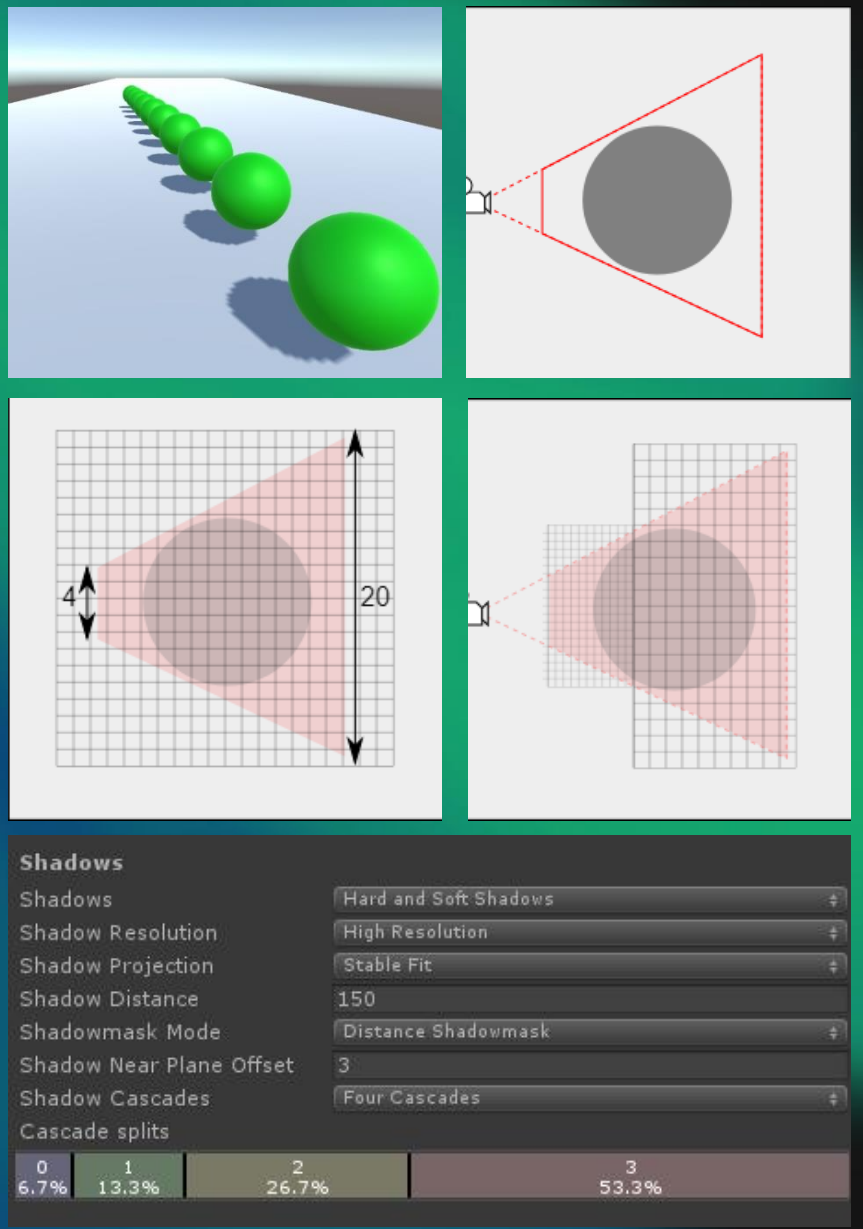
Shadow Quality

Perspective aliasing SM pixels seen close to the camera look enlarged compared to those farther away

- 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 the same 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

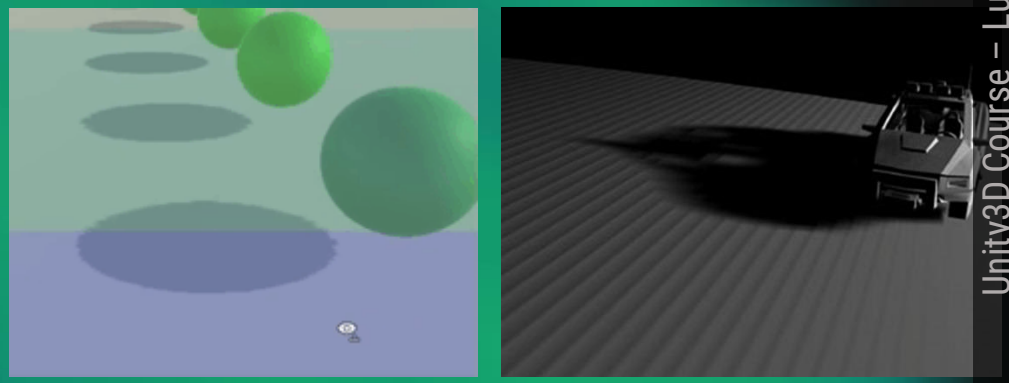
- Keep ShadowDistance as low as possible
  - Shadows that are far away often don't increase image quality
  - We can add Fog to the scene

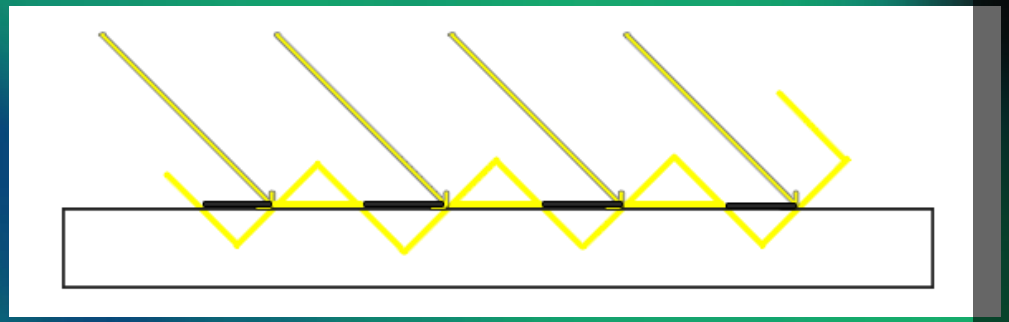


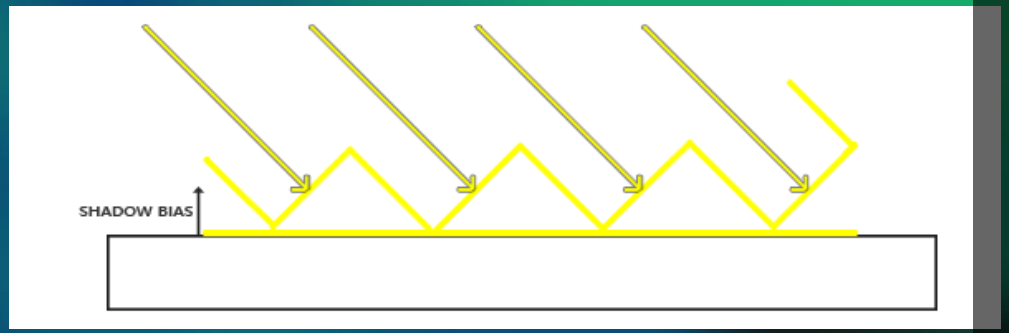
Shadow Quality

- ShadowProjection (SceneView/ShadowCascades)
  - StableFit Coose CSM band based on Cam depth buffer
  - CloseFit Choose CSM band based on Cam pos distance

- Bias Reduces the Shadow Acne Problem
- NormalBias Reduces Self shadowing Problem



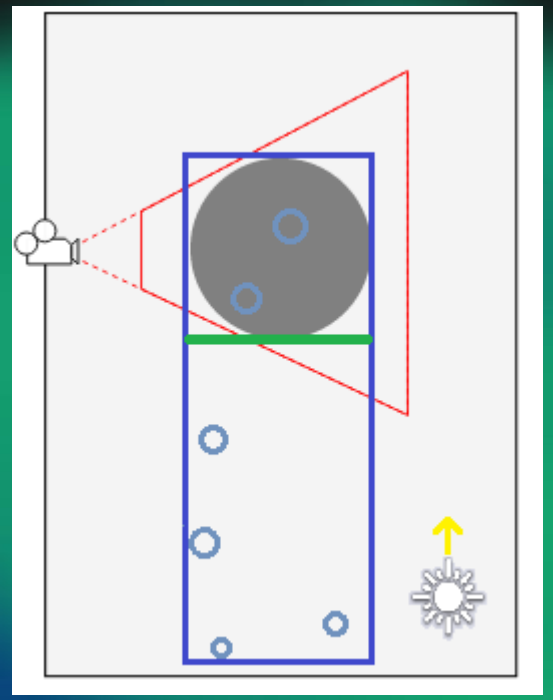


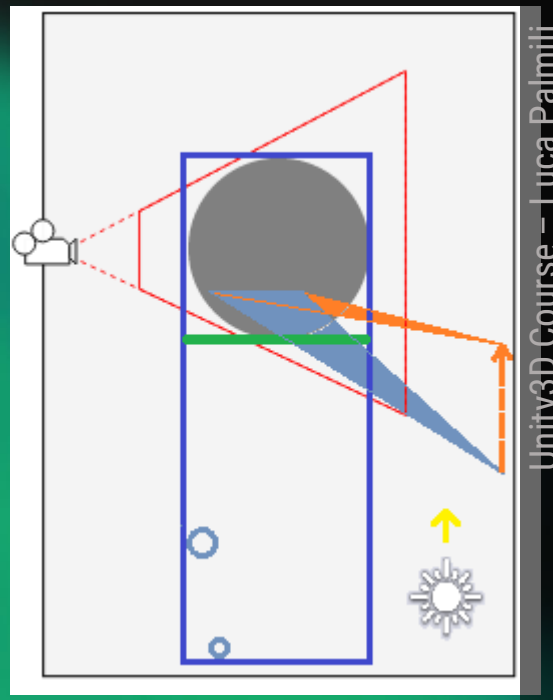


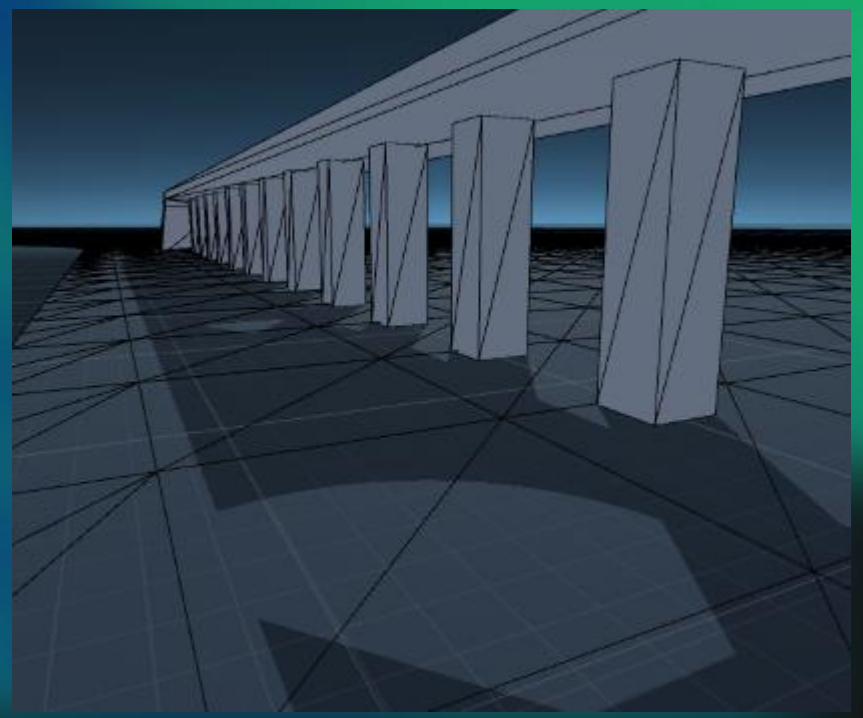
Shadow Pancaking

Instead of render the entire scene in lightSpace

1. Calculate a near plane *Np* excluding any shadow caster not visible in the camera view frustum
2. Clamp the shadow casters in the camera VF to plane *Np* in the Vertex Shader.
    - This can create artifacts for very large triangles crossing *Np*
3. Use the QualitySettings/ShadowNearPlaneOffset to pull back Np and avoid this problem

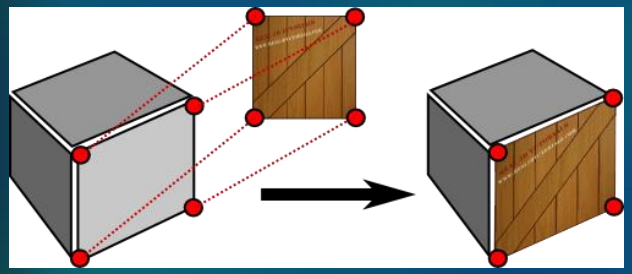
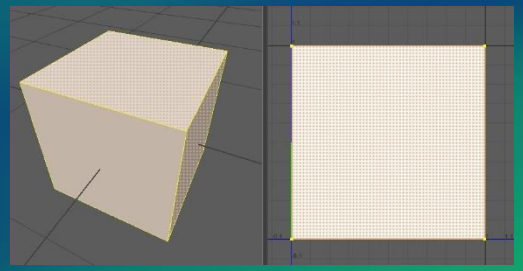
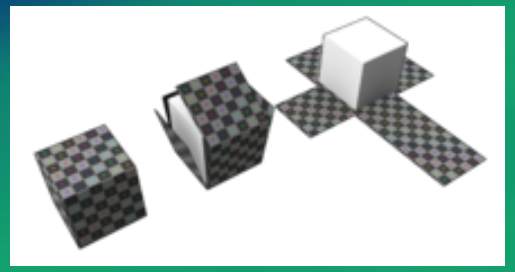
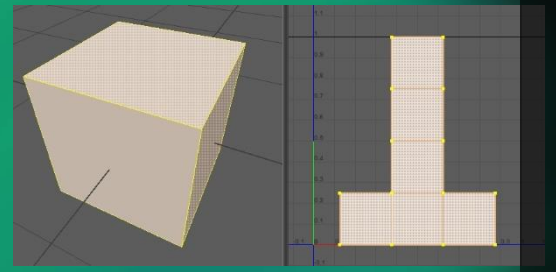
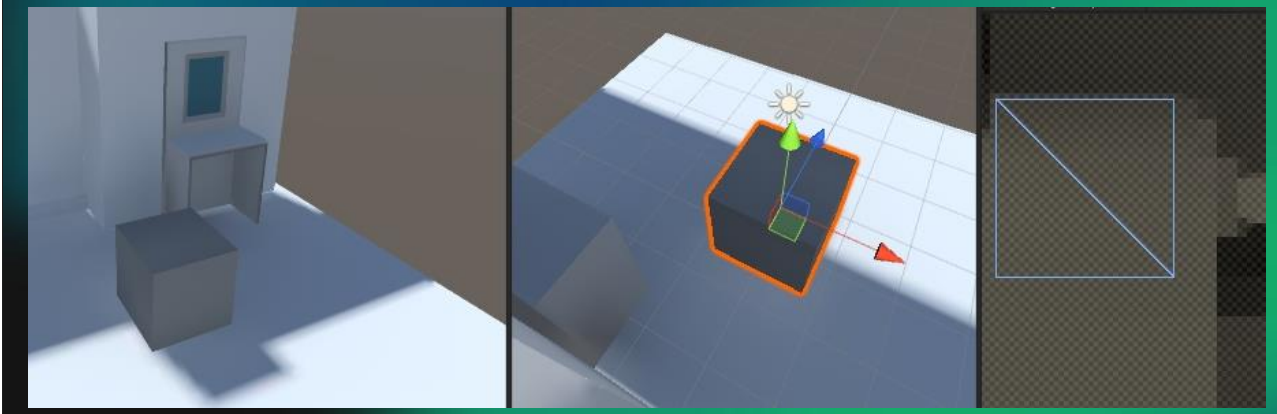
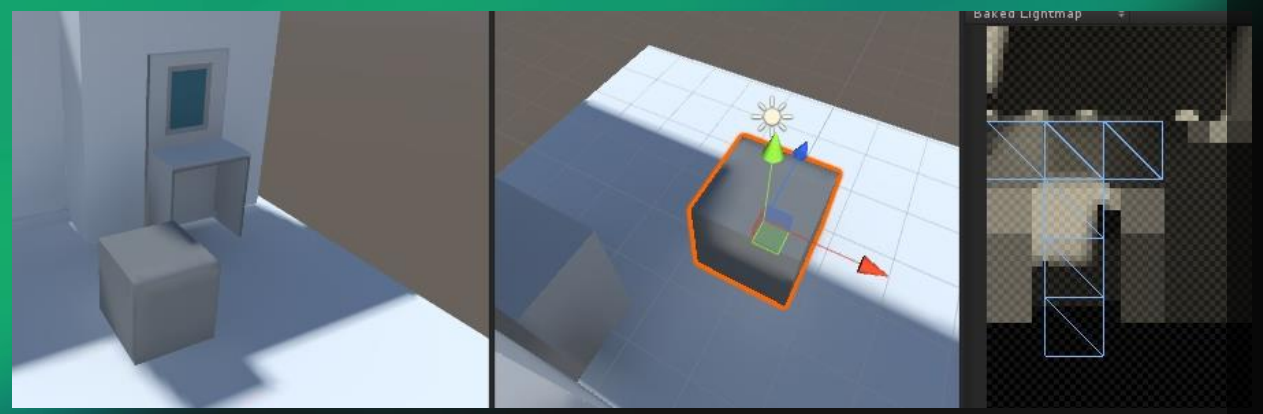






Lightmapping UVs

- 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

UVs channels

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

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

Enlighten components

Precompute

- Precalculates light transport in a scene
- Only depends on the static geometry and not the materials or light
  - Packing
  - Clustering
  - Compositing the light transport

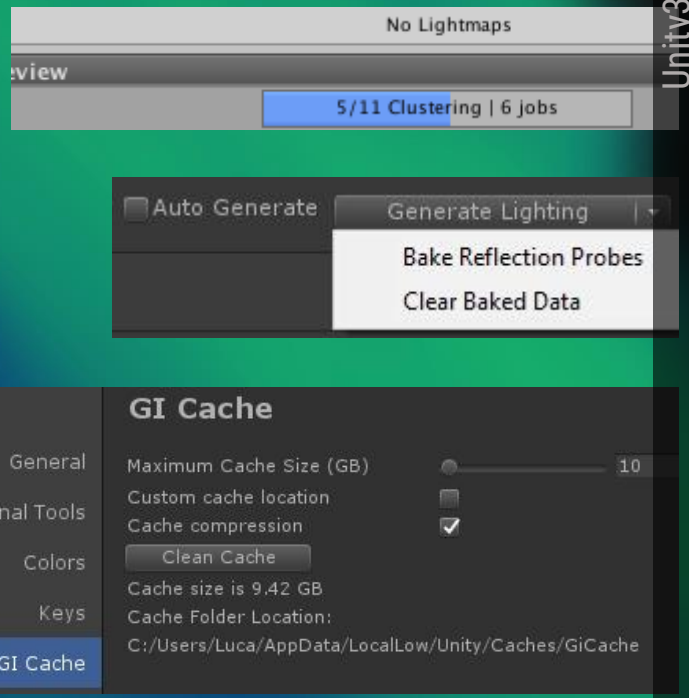
Real-time solver

- Combines the precalculated data with the material and light information to produce light maps and probes in real time

Light map baker

- Produces baked light maps for direct and indirect light and also AO
- Relies on the precompute data

- Autogenerate 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



Precompute/Packing

The resolution of Realtime GI lightmap MUST be low

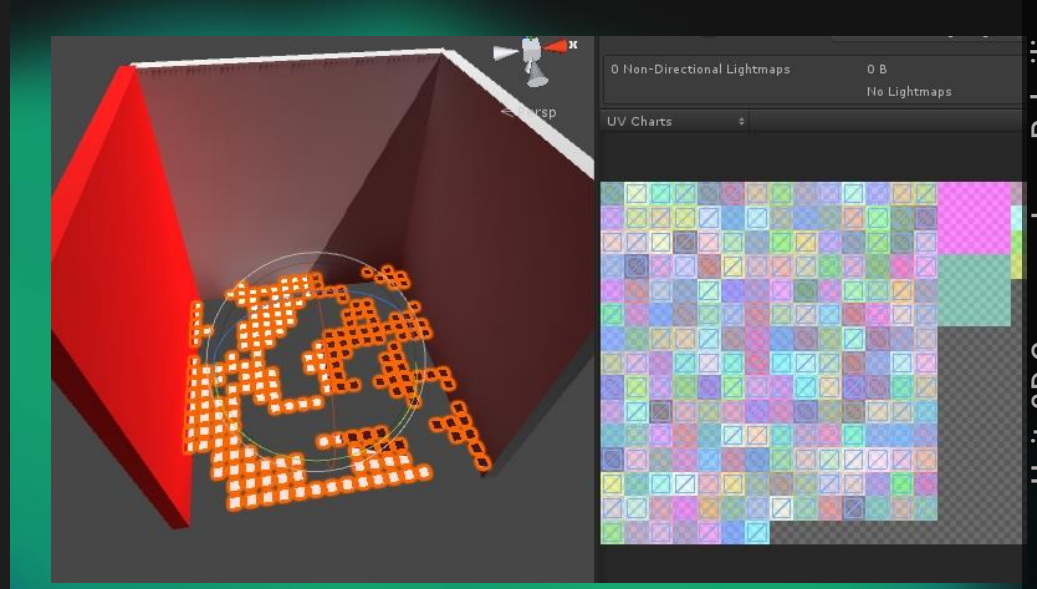
- We need another lightmap texture

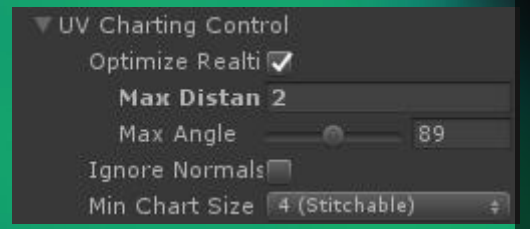
Packing main tasks

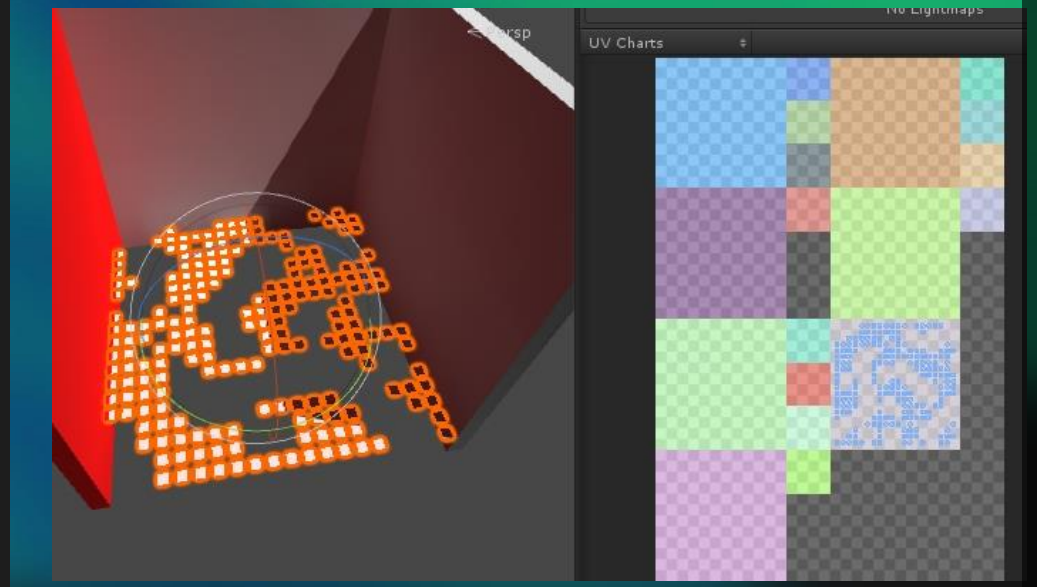
1. Identifies Charts in BGI UVs
   - Charts Groups of triangles in UVs that share vertices
2. Pack the Charts into lightmap used for real-time GI, ensuring that there is no light leaking between charts and that UVs are packed as tightly as possible

Uses

- Existing BakedGI UVs (Copy them into RealtimeGI UVs)
  - Usually produce artefacts, because of the low resolution lightmap
  - Not optimized
- Auto generated RealtimeGI UVs (from StaticGI UVs)
  - MaxDistance/MaxAngle
  - Ignore Normals
  - Min ChartSize





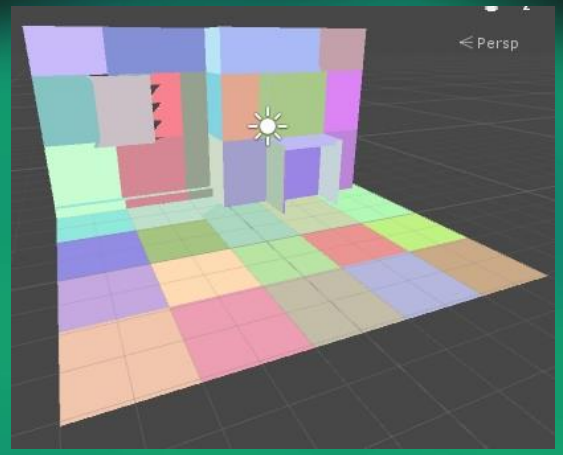


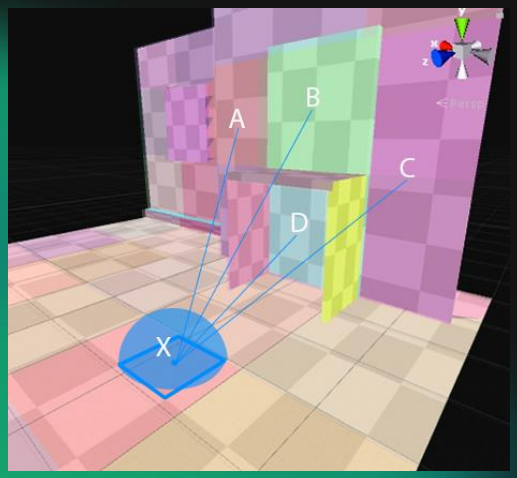
Precompute/Clustering

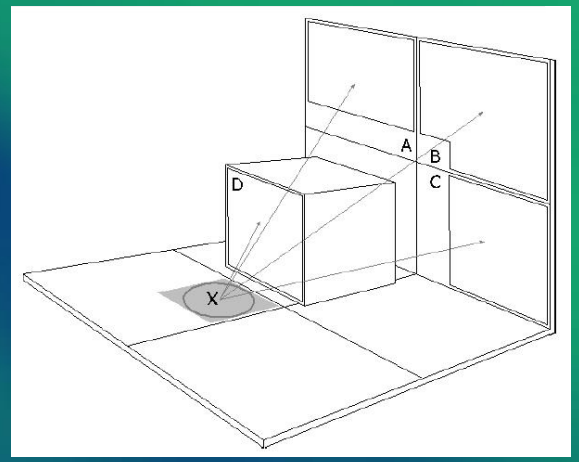
- Splits scene Geometry into clusters
- Use only triangles pos & orientation NOT UVs or Materials

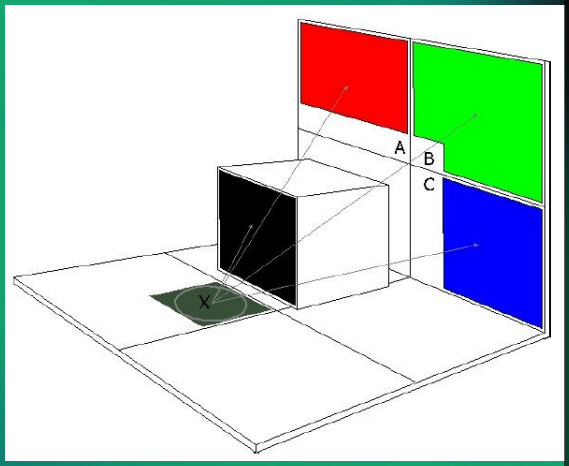
1. ∀ texel in a real-time lightmap / ∀ lightprobe
    1. Cast rays in all directions of its hemisphere / all directions
    2. Calculates the visibility of the cluster (form factor)

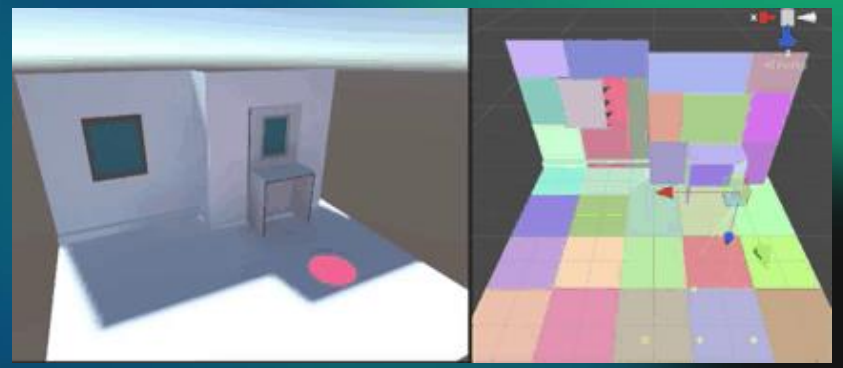
Lighting received from X = 0.05*A + 0.1*B + 0.05*C + 0.2*D









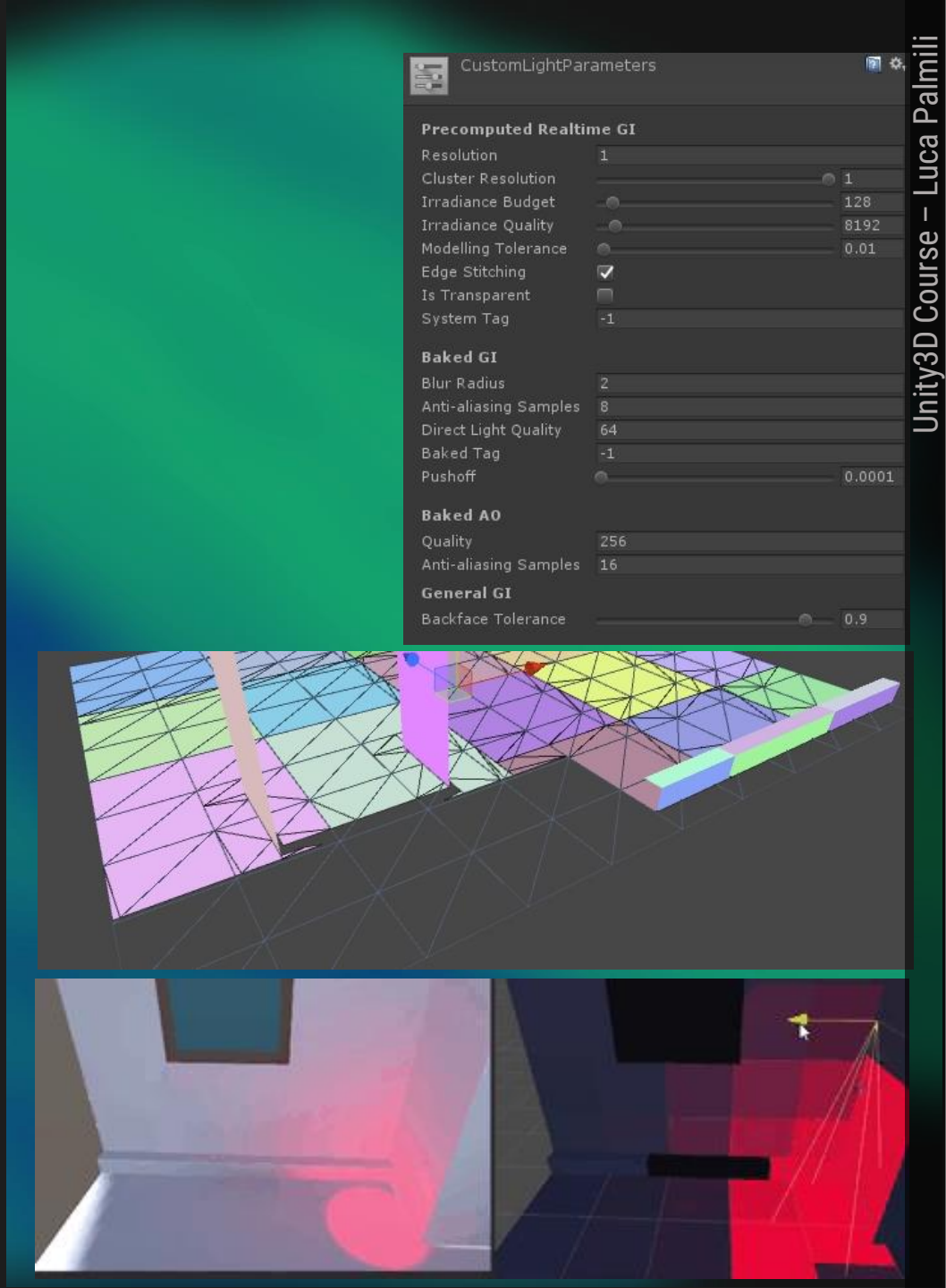


Precompute/Clustering

Every obj can have its own lightmap parameters
- Create>LightMap Parameters

Light Transport quality based on
- Cluster resolution Default is half the real-time lightmap
- Irradiance Budget Form-factors # that Enlighten stores
- Irradiance Quality Rays # to cast per pixel
  - Multiple rays can contribute to build better form values

- BackfaceTolerance Invalidate Texel if too many of rays cast from it hit back faces



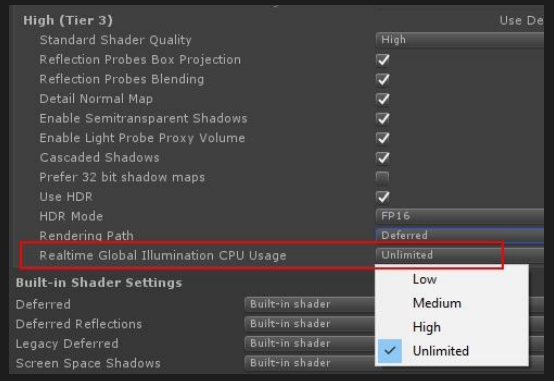
Real-time Solver

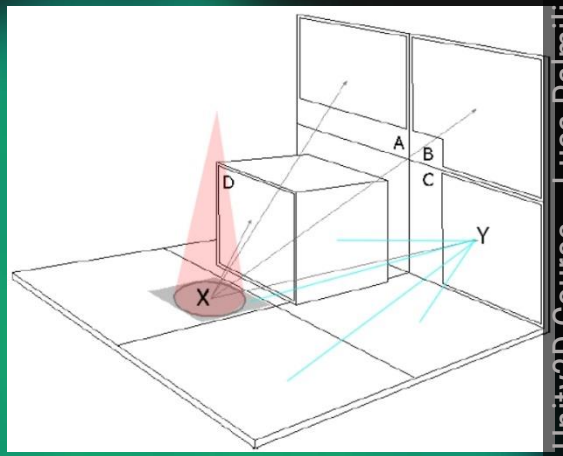
- Present in the Editor and in the Game

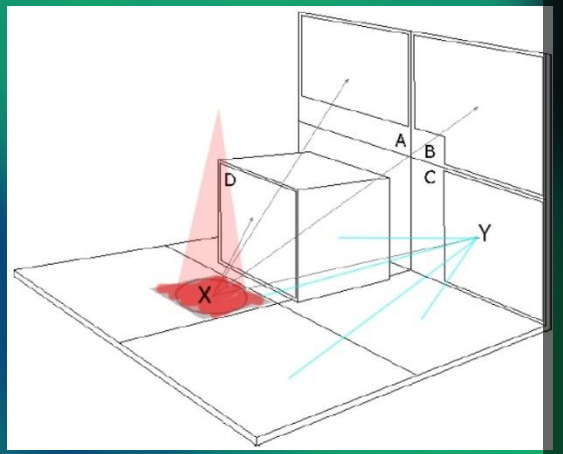
- Input lighting stage Direct lights contribute on the clusters, including dynamic lights
- Solve stage The solve stage sums up the cluster value multiplied by the stored form-factors, and stores the results in the light map

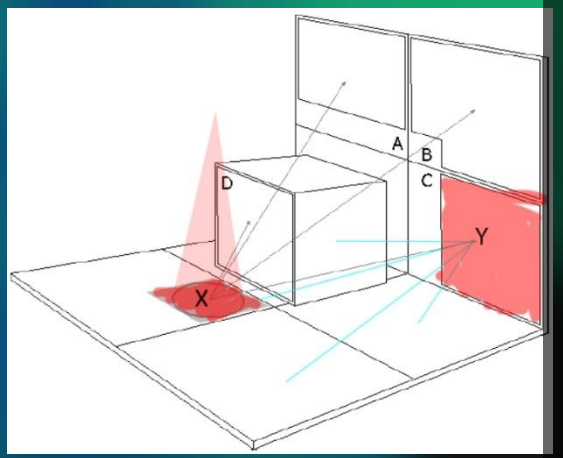
$$B_i = L_e + p_i \sum_{j=1}^{n} F_{ij} L_j$$

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









Setup a scene

Turn off MixedLighting

- When happy with the lighting setup, turn on MixedLighting
  - baked light maps match the real-time rendered results (except fo soft shadows and area lights)

Setting the indirect resolution

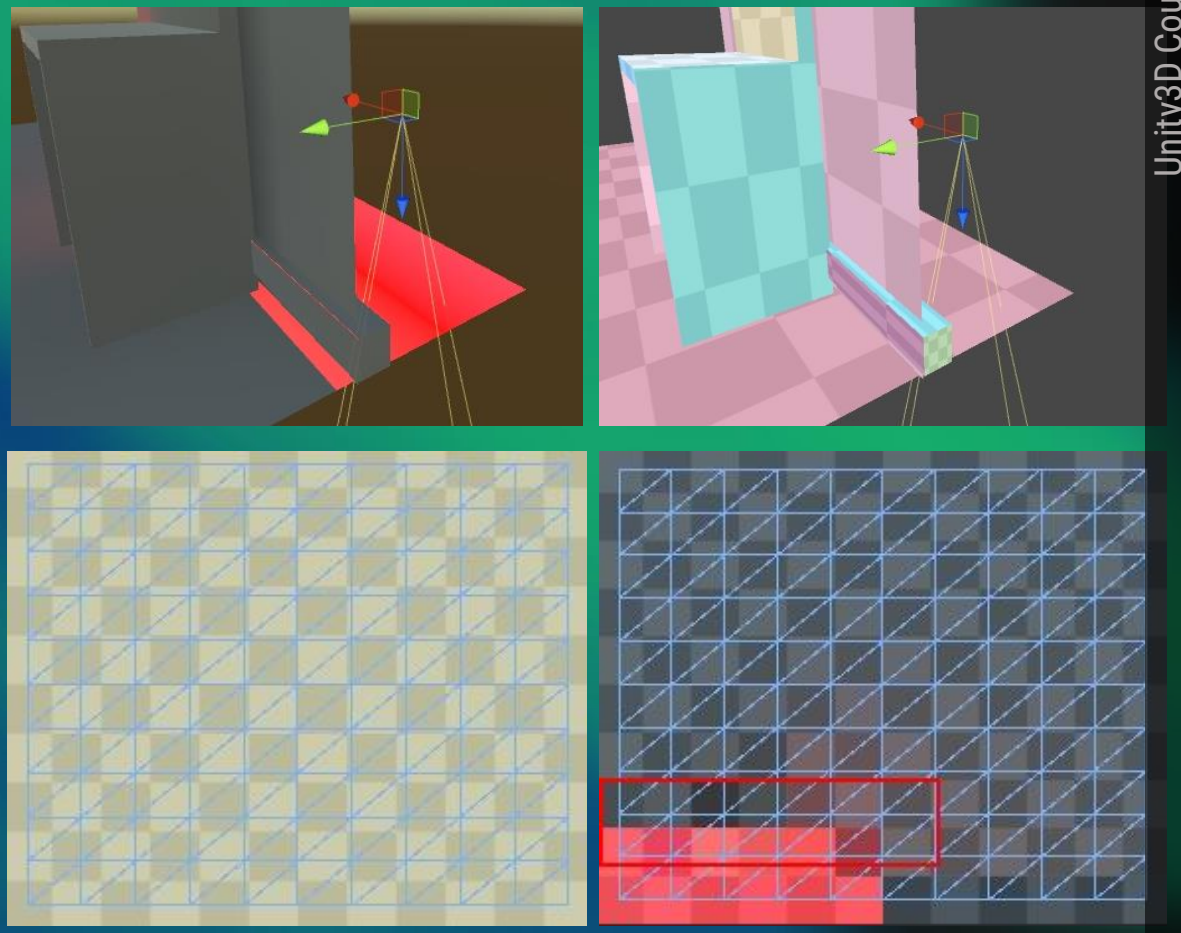
- Indoor 2-3
- Outdoor 0.5-1
- Terrains 0.1-0.5

Pay attention to Charts disposition

- Charts that spans through a wall can produce light leaking
  - Split input UVs by hand

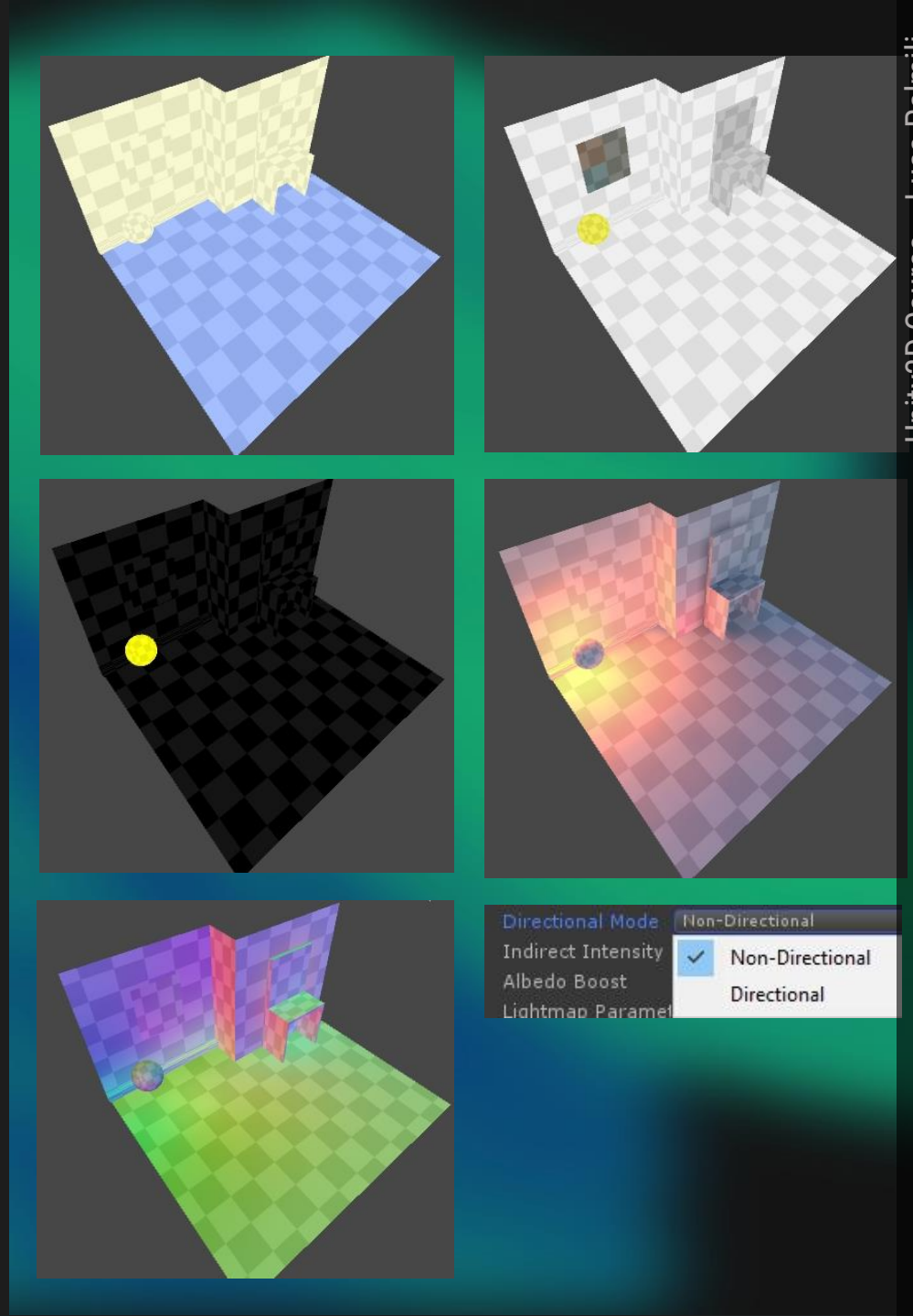
Use emissive lighting for fake areaLights (but in realtime GI)

Use lightprobes for small objs



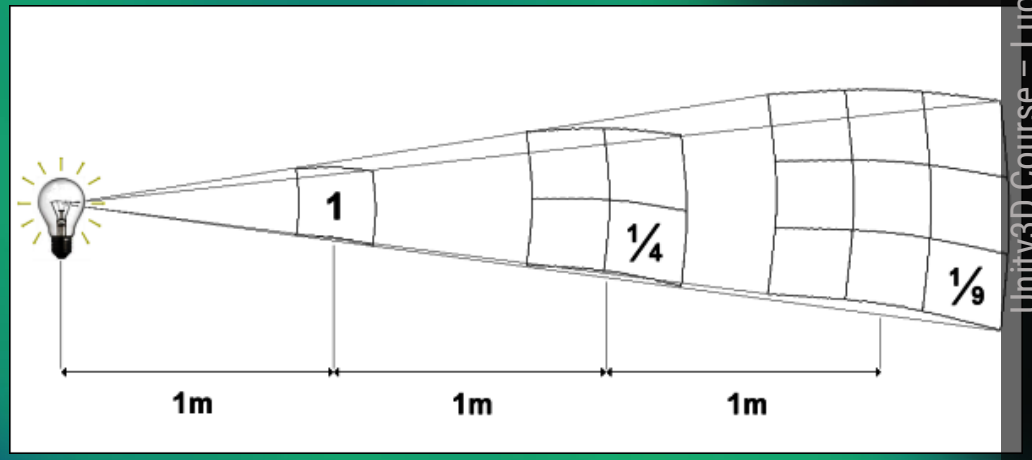
SceneView GI Draw Modes

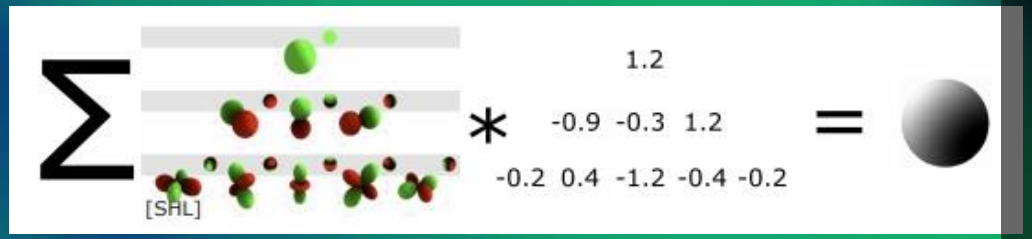
- Systems Enlighten is able to generate multiple Systems: each one is able to perform the precompute and the runtime in parallel
- Albedo Shows at what detail Enlighten uses the color information from your scene
- Emissive Color and intensity of the emitted light from emissive objects
  - Tip: we can use emissive objects with Enlighten to create area lights without any rendering cost
- Indirect Irradiance received by surfaces using light maps
- 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

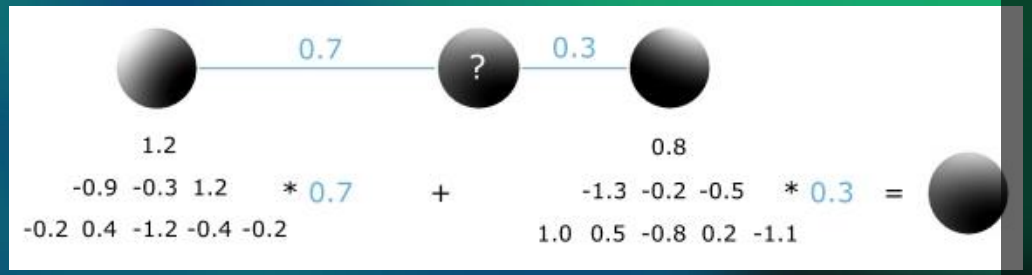


Light Probes

- Lightmaps store info about light hitting the surfaces, Light Probes store info about light passing through empty space
  - provide high quality lighting (including indirect bounced light) on dynamic objs
- Irradiance
- A LightProbe use L2 SH to store directionality of RGB ray values (27 floats)
- Interpolating two probes can be done just by interpolating their coefficients.







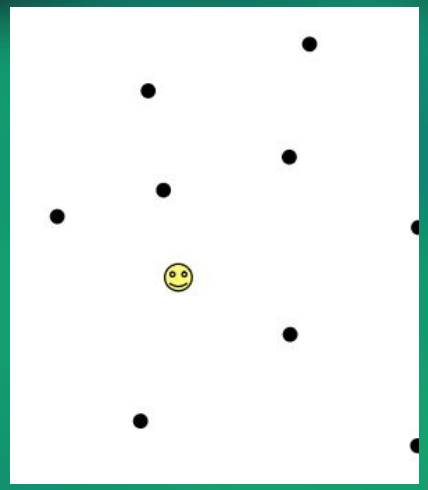
Light Probes

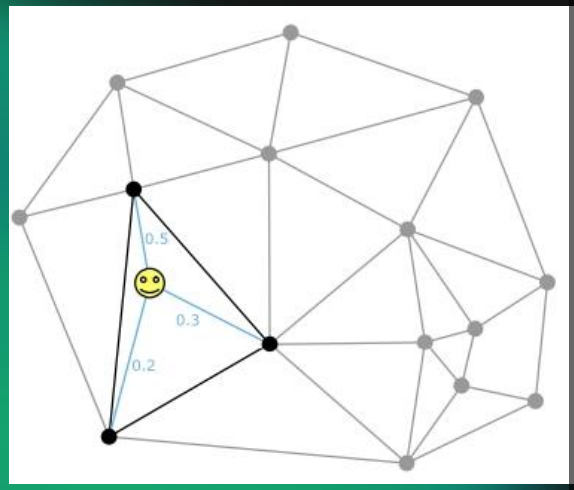
- Which probes & what wheights?
- Wow many pts do I need to define a circle?
- Delaunay Triangulation
- 2D > 3D Delaunay Tetrahedralisation

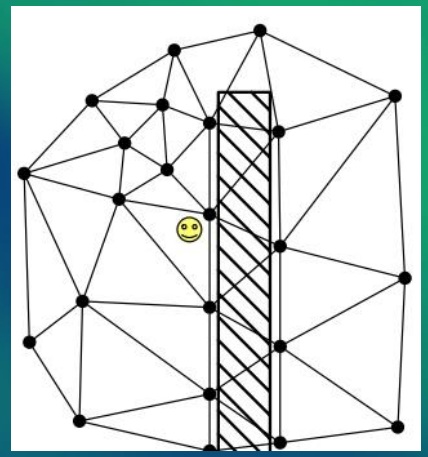
1. Cache the tetrahedron index from the previous frame
2. Calculate barycentric coordinates *Bc*
    1. *Bc > 0* => We are inside
    2. *Bc < 0* => The obj moved the most negative coordinate
3. Each tetrahedron has exactly 4 neighbours. Find the right one

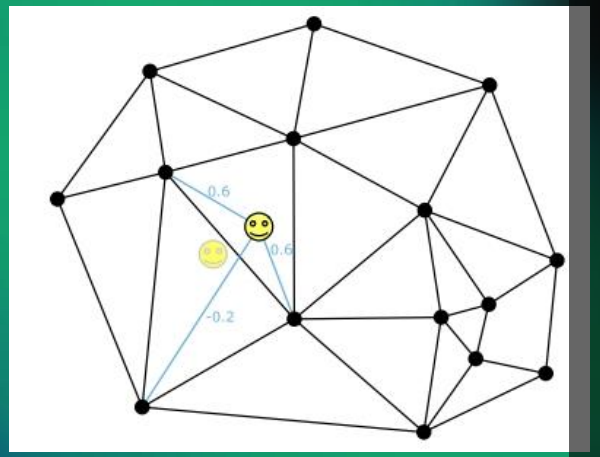
Data needed

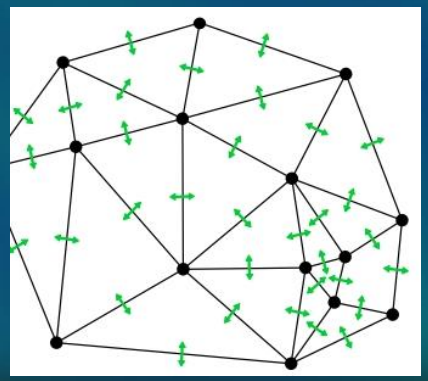
- probe positions probe_count * 3 floats
- SH coefficients probe_count * 27 floats
- hull rays hull_probe_count * 3 floats
- Tetrahedron indices 4 vertices + indices 4 neighbours

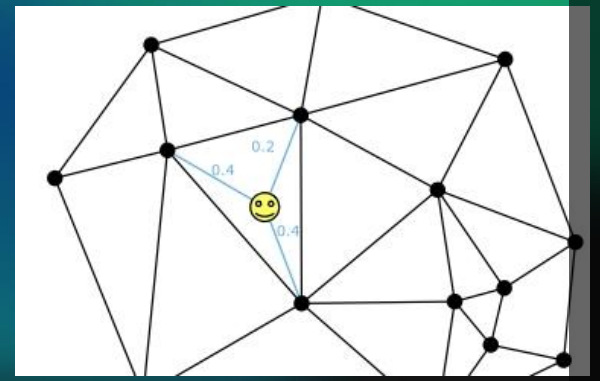












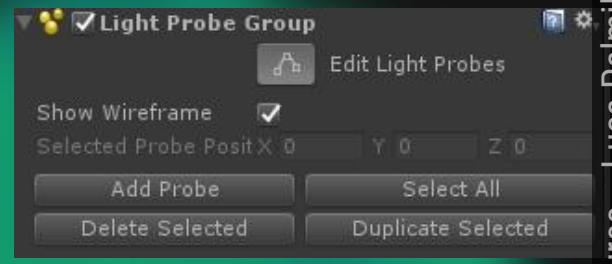
Images from GDC2012

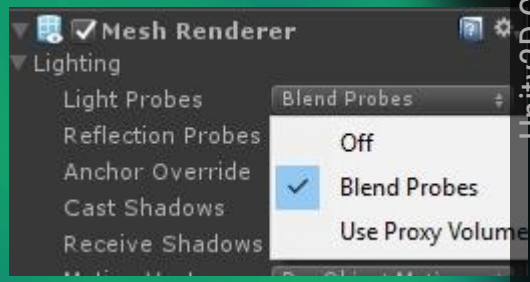
Unity3D Course – Luca Palmili

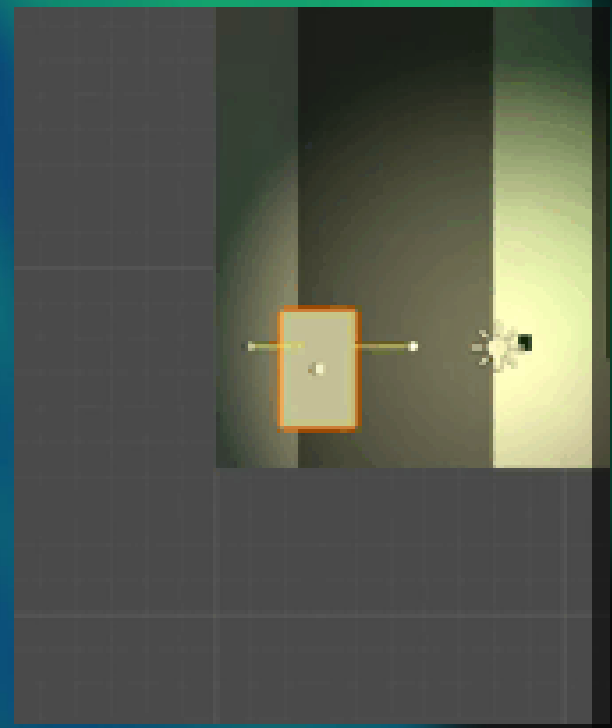
Light Probes - Placement

- 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). Aat 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
  - UseProxyVolumes + lightProbeProxyVolume large moving objs
  - OverridePoint
- LightProbeProxyVolume

[LightProbesTetrahedralGrid.cs]

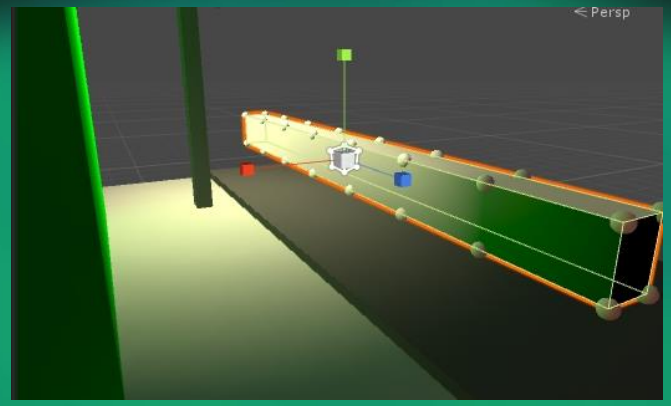


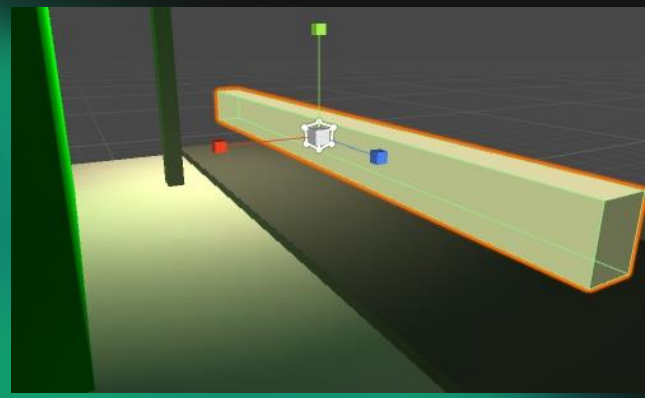


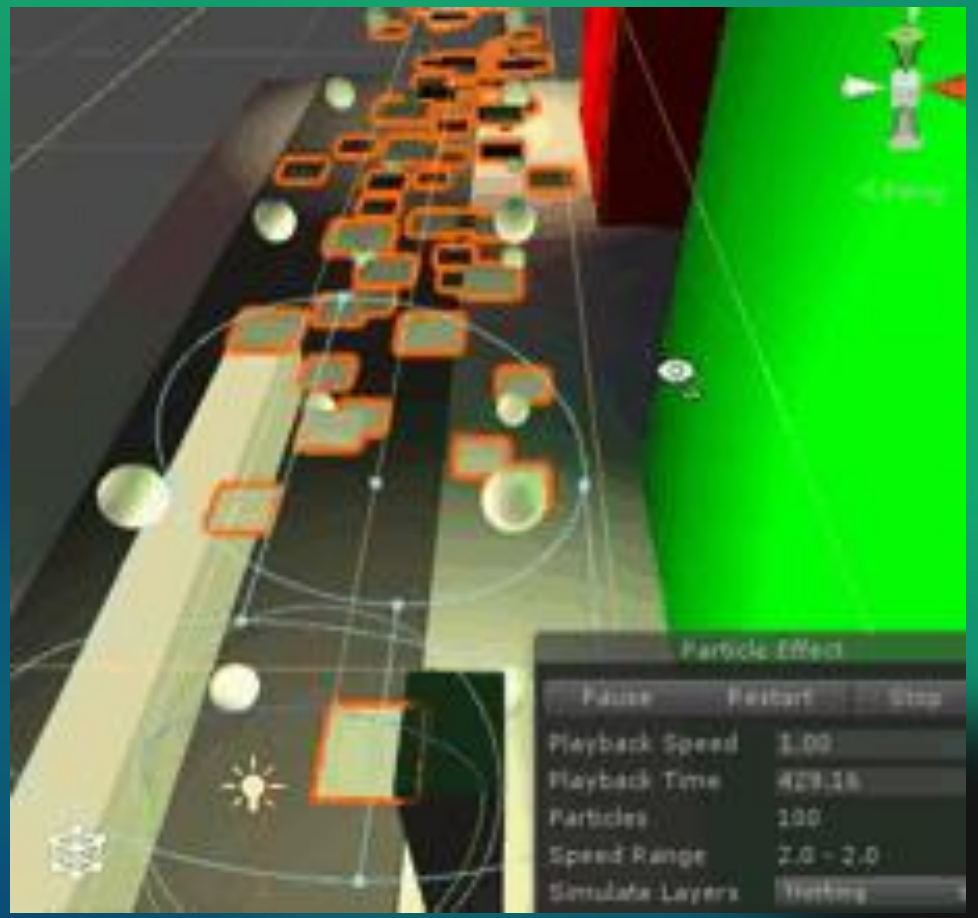


LP Proxy Volume

- UseProxyVolumes + lightProbeProxyVolume
  - Large moving objs
  - Particles
  - ProxyVolumeResolution
  - ProbePositionMode

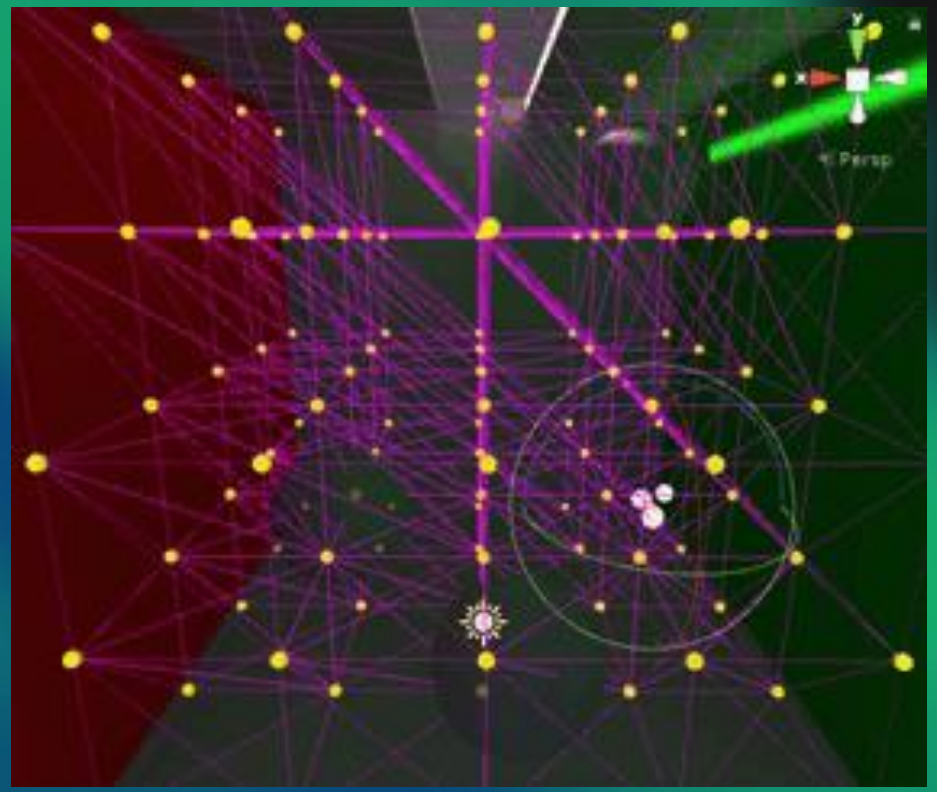






Ex 39 – Italian Box

- Use Realtime GI
- Add LightProbeCreator.cs to LightProbeGroup
  - In OnValidate() it will reset LightProbeGroup LProbes, creating a cube grid of Xp x Yp x Zp Probes, which will span across a volume of Xsize x Ysize x Zsize
  - LightProbeGroup LPG = GetComponent<LightProbeGroup>();
  - List<Vector3> probePos = new List<Vector3>();
    - //Add Vector3 positions to probePos
    - LPG.probePositions = probePos.ToArray();

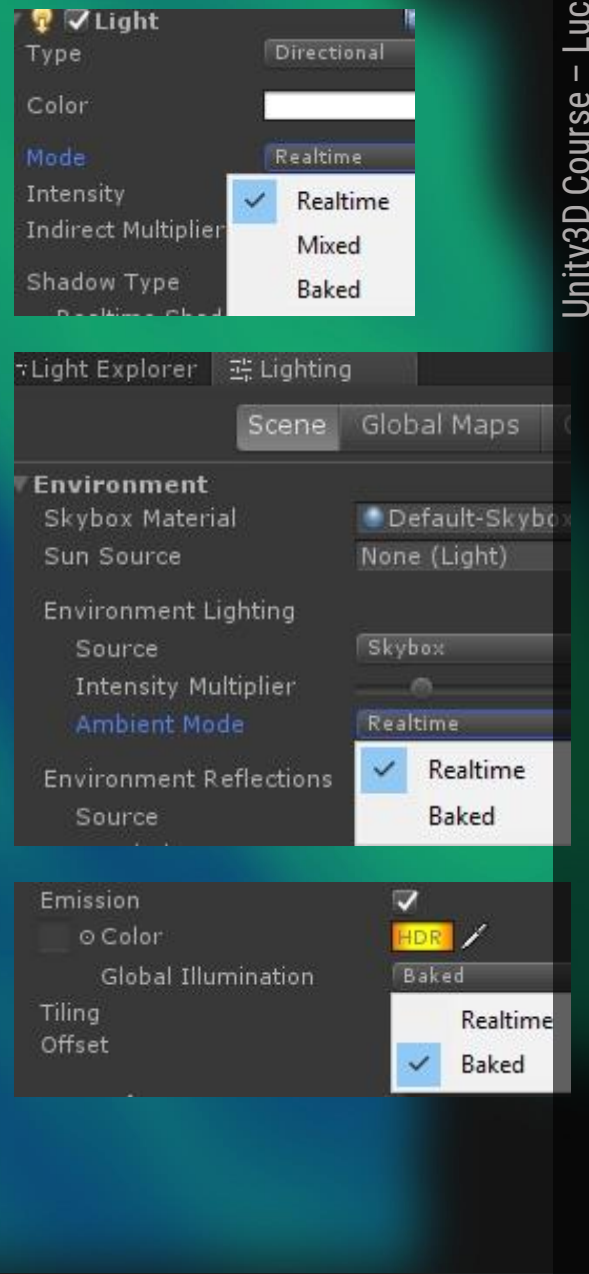


Baked/Realtime lights

- Baked GI (Mixed lighting)
- Real-time GI

Either one or both of these lightmap sets can be used to light your scenes

- Light sources and material emission can be marked as realtime or baked and that determines to which lightmaps the light contributions and resulting GI is added
  - Lights have a realtime/baked/mixed control
  - Environment lighting has a realtime/baked control
  - Materials have realtime/baked emission control



Baked Lighting

Useful for

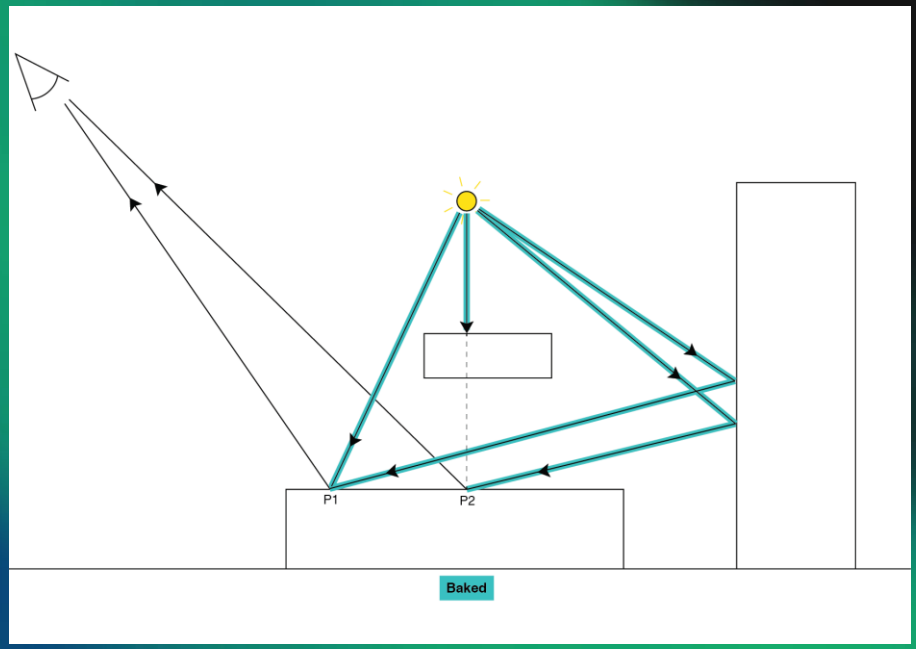
- local ambience: these lights are pre-calculated before run time. They are not included in any run-time lighting calculations

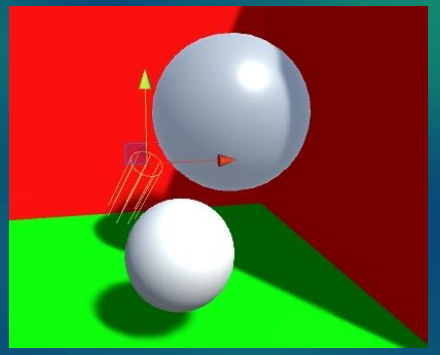
PROs

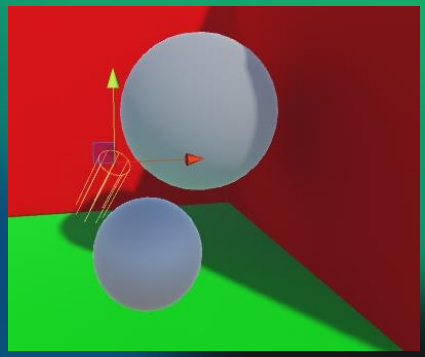
- High-quality shadows from statics GObjs on static GObjs
- Lighting for static GObjs is one Texture fetched from the light map in the Shader

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







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

Problems

- All baked objs will receive dynamic light as well > double exposure, overbrightness
- Mask real-time lights for baked objects > discrepancies in terms of brightness values

Solution Mixed lighting

- Illuminates dynamic and static objects within the same balanced range > retains visual fidelity and consistency

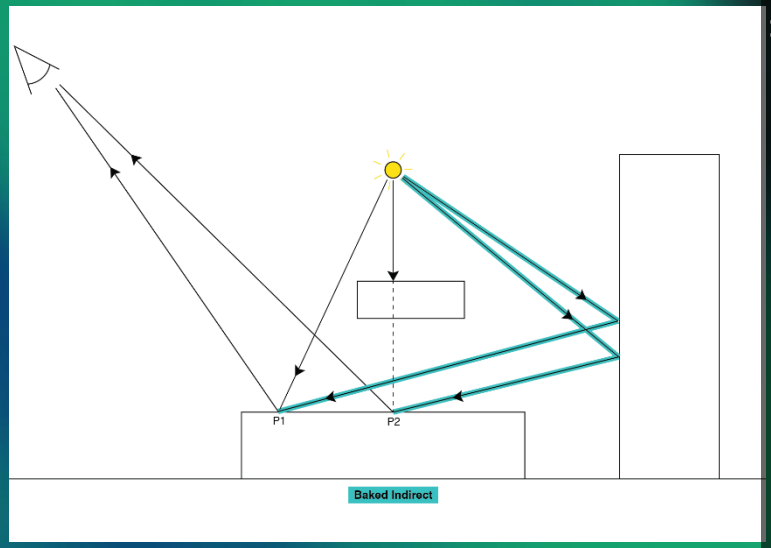
Mixed Lighting / BakedIndirect

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



ML / Shadowmask

- Set Edit/ProjectSettings/Quality/ShadowMaskMode/Shadowmask
- ShadowMasks (and Light Probes) stores information on up to 4 overlapping light sources: this enables shadow merging between dynamic and static objs. Above 4 overlapping lights, they will be fully baked

Performance requirements

- Mid to low

PROs

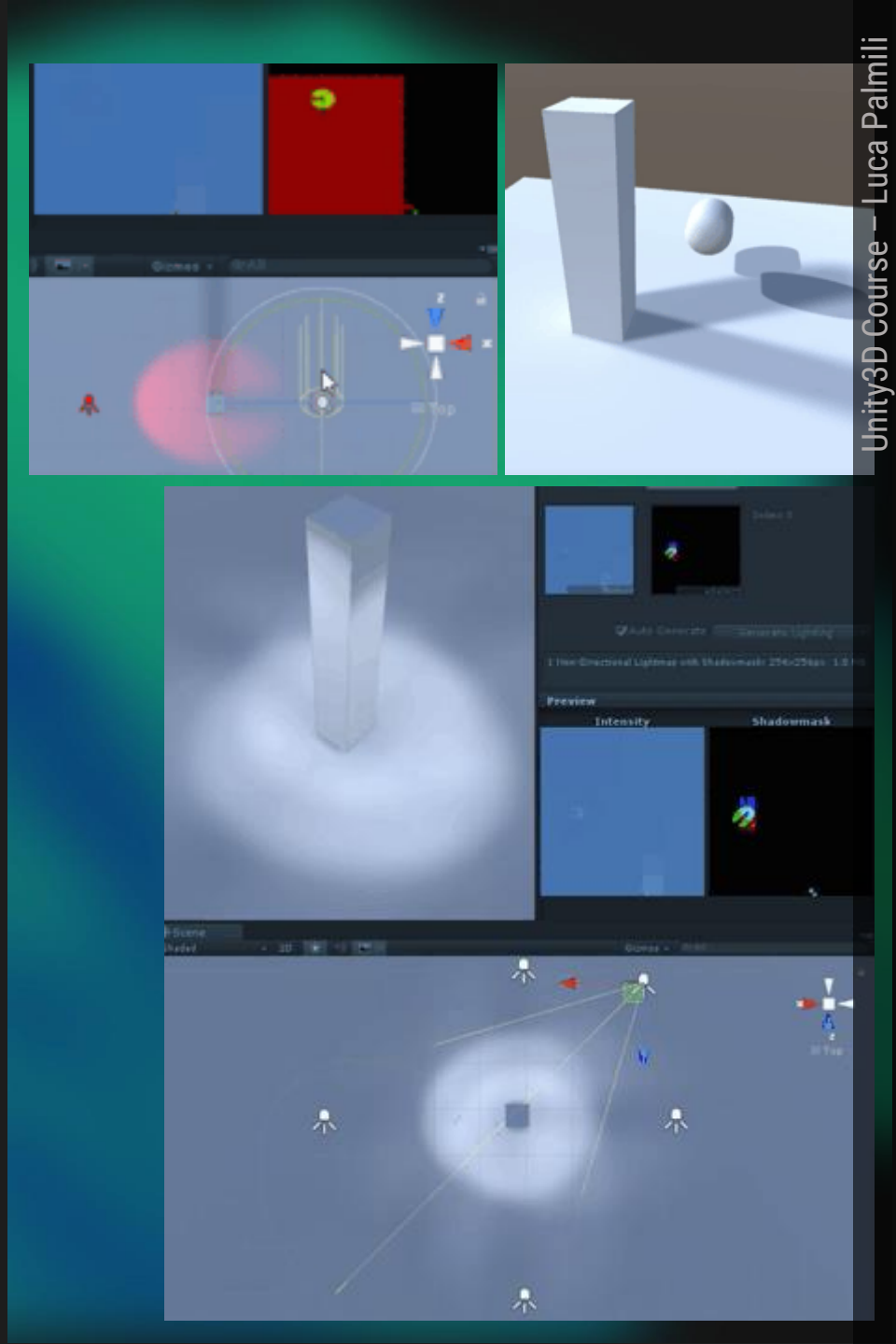
- Automatically composites overlapping shadows from static and dynamic Gobjs
- Offers less draw calls than distance shadowmask mode (no shadowmaps for static objs)

CONs

- Low-resolution shadows from static GObjs onto dynamic GObjs, via Light Probes (because of the shadowmask)
- Allows up to 4 overlapping light volumes
- 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
- Open-world Scene with baked shadows up to the horizon, but without dynamic lighting such as a day/night cycle



ML / Distance Shadowmask

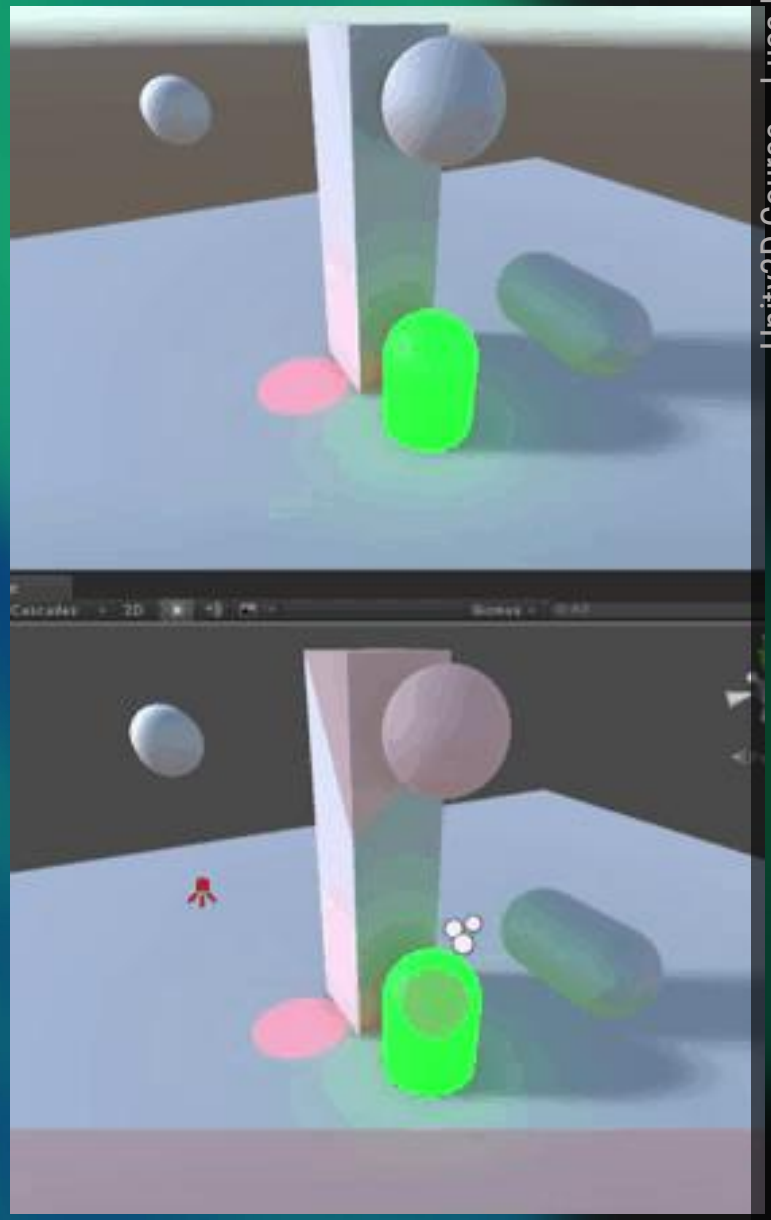
- Set Edit/ProjectSettings/Quality/ShadowMaskMode/DistanceShadowmask
- The Distance Shadowmask mode is a version of the Shadowmask lighting mode that includes high quality shadows cast from static GObjs onto dynamic Gobjs
- It is possible to switch between then @ run-time
    - QualitySettings.shadowmaskMode = ShadowmaskMode.Shadowmask

Performance
- higher performance requirements than Shadowmask mode: both static & dynamic GObjs are rendered into the shadow map (demo video), every frame

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

Useful for
- Open world Scene with shadows up to the horizon, and complex static Meshes casting real-time shadows on moving characters



ML – Subtractive

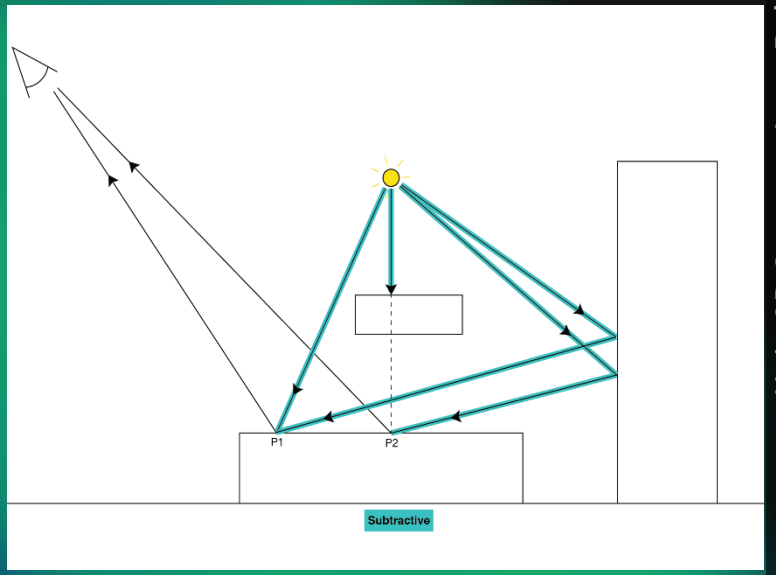
- The only Mixed lighting mode that bakes direct lighting into the light map: Unity cannot perform any direct lighting calculations at run time.
  - try to disable one light @ RunTime – it'll remain baked in Intensity map
- Subtractive mode has a Realtime Shadow Color field. Unity uses this color in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

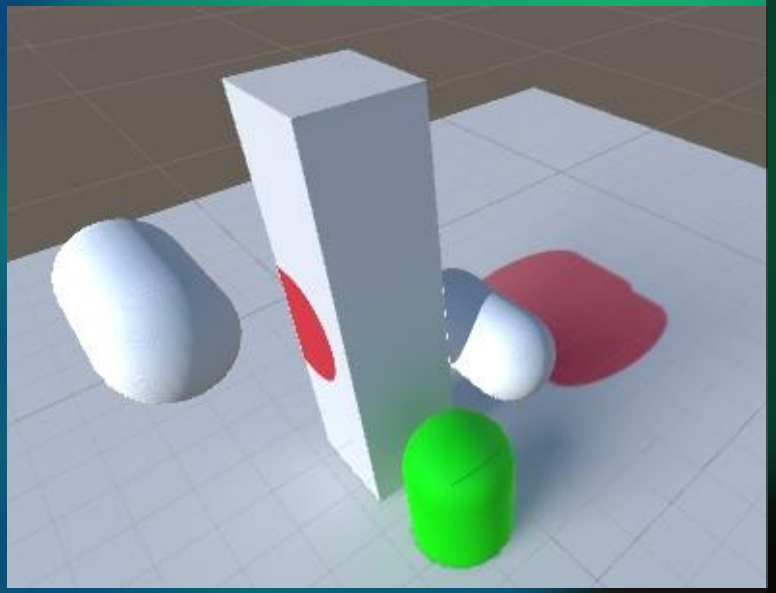
PROs

- Provides high-quality shadows between static GObjs in lightmaps at no additional performance requirement

CONs

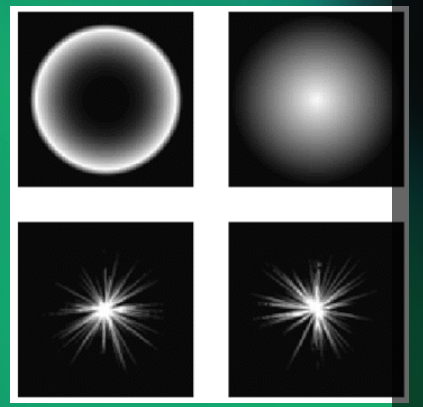
- No real-time direct lighting > No specular lighting
- No dynamic shadows on static GObjs, except for one Directional Light (the main Light)
- Provides low-resolution shadows from static GObjs onto dynamic GObjs, via Light Probes (use LPVolumes for better results)
- Provides inaccurate composition of dynamic and static shadows
- Designed for forward rendering and gamma color space: because it is relatively cheap, is targeted for low end devices such as mobile platform

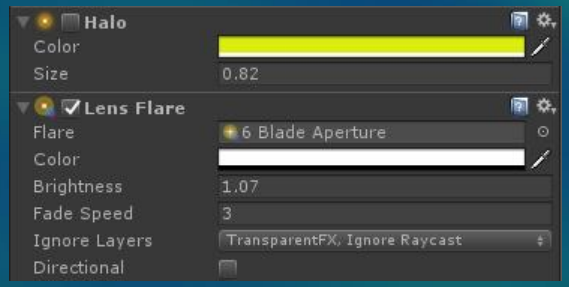


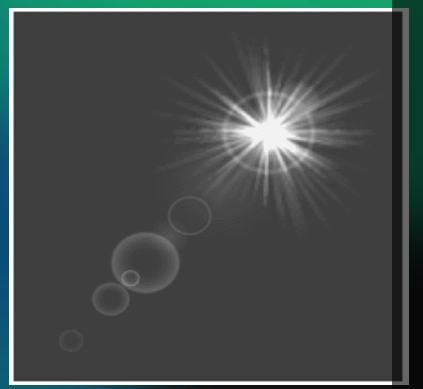


Halo / LensFlare

- Halo effect size and color is linked to light paramenters
  - To override this use Halo component
- Flare
  - Similar to Halo, but imitates a bright light source seen through optical glass
  - Depends on the intensity of the light
  - The camera must have a Flare layer attached
  - 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 Physics.queriesHitTriggers is true







Forward Rendering Path

For each drawcall

1. VERTEX SHADER
2. RASTERIZER
3. PIXEL SHADER

In the Pixel shader

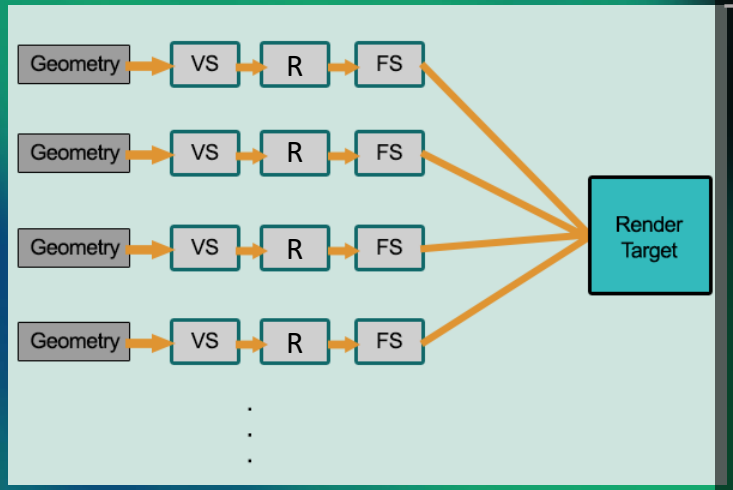
- Light calculations # = [geometry_fragments #] x [num_lights]

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
  - 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.



Deferred Rendering Path

For each drawcall - Geometry Pass

1. Render the entire scene like in FRendering, but the PShader output is not the screen, instead multiple RederTargets will be built (normal, specular, albedo)

In the Pixel shader

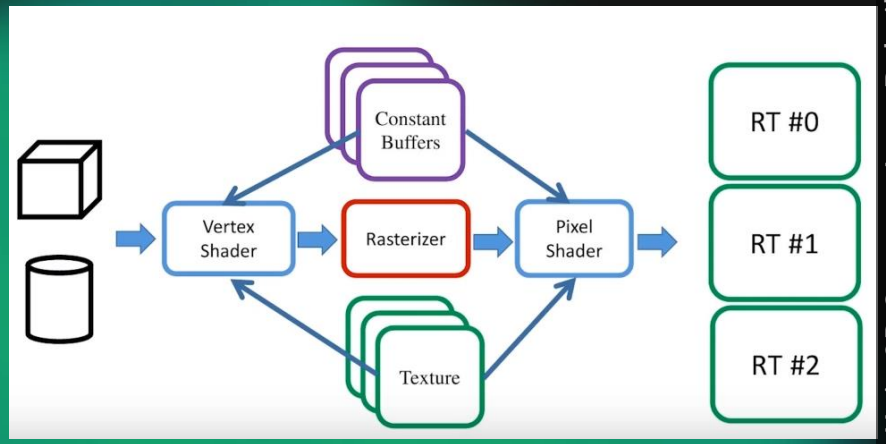
- Lighting calculations # = 0
- Gbuffer calculations # = [geometry_fragments #]

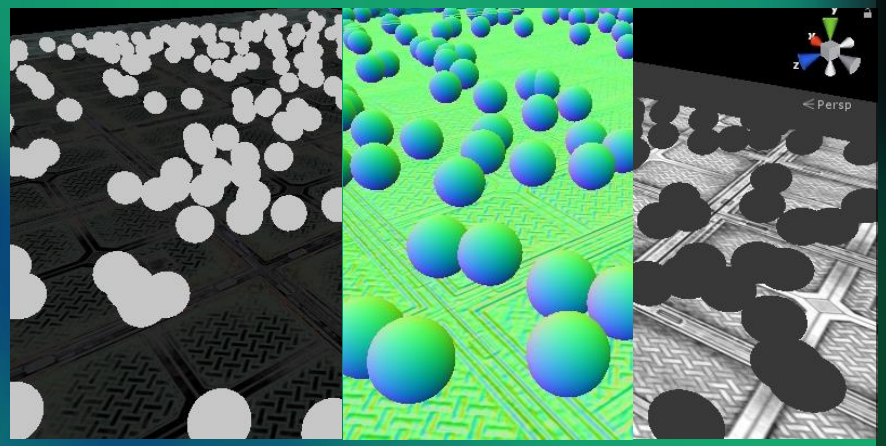
For each light - Lighting Pass

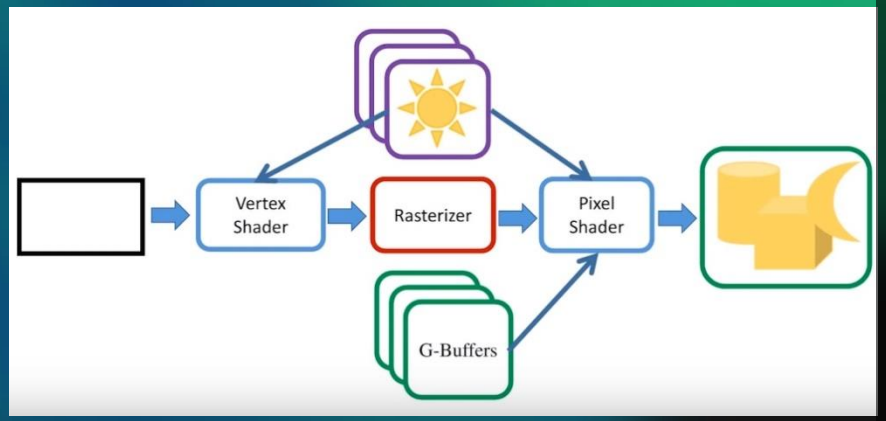
1. Pas to the VSHADER a FullScreenQuad (*)
2. Rasterizer will let all the pixel to pass through the PShader
3. PSHADER

In the Pixel shader

- Lighting calculations # = [screen_resolution]







Deferred Rendering Path

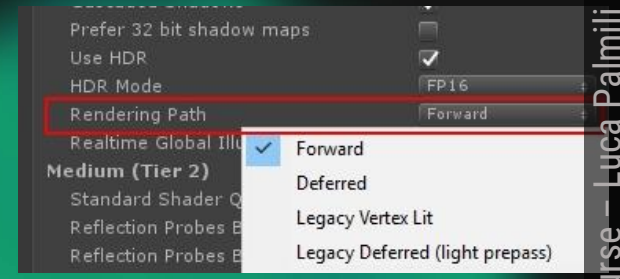
Example
- Screen Res 1024x768 = about 800K pixels = about 1M fragments
- # of lights: 100

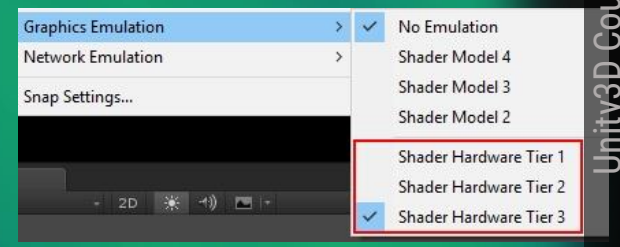
FRendering
- O(num_geometry_fragments * num_lights) = 1M*100 = 100M PixelShader with lighting calculations
- PSHADER EXECUTIONS = 100M

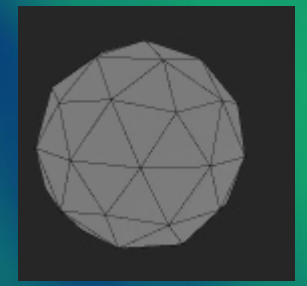
DRendering
- Geometry pass = 1M PixelShader with GBuffer calculations
- O(screen_resolution * num_lights) = 800K*100 = 80M PixelShader with lighting calculations
- PSHADER EXECUTIONS: 80M + 1M = 81M

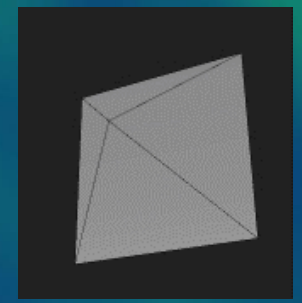
What if 99 lights are not Directional but Point lights that lit 1/10 of the screen fragments?
DRendering
- Geometry pass = 1M PixelShader with GBuffer calculations
- Directional light: O(screen_resolution * 1) = 800K*1 = 800K PixelShader with lighting calculations
- Directional light: O(1/10 * screen_resolution * 99) = 80K*100 = 8M PixelShader with lighting calculations
- PSHADER EXECUTIONS : 1M + 800K + 8M = 9.8M

- We can change Rendering path in Edit/ProjSettings/Graphics, changing the correct Tier we selected in Edit/GraphicsEmulation









Deferred Rendering Path

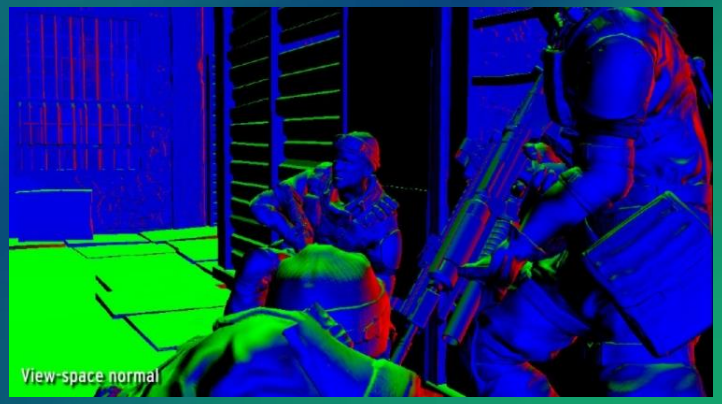
PRO

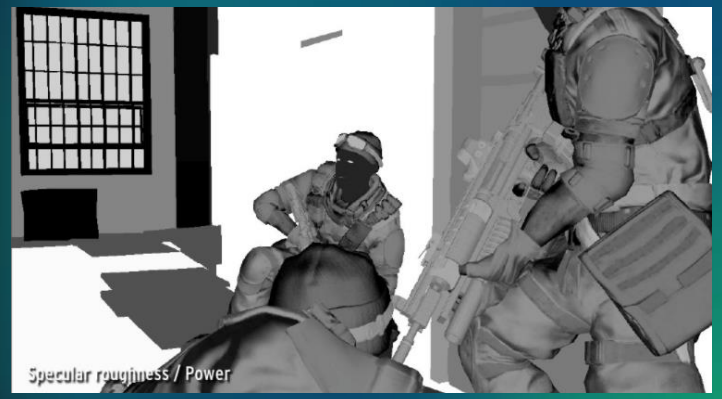
- Scene geometry decoupled from lighting
- Shading/lighting only applied to visible fragments
- Lot of Post-processing camera Fx rely on GBuffers data, altready calculated during Geometry Pass
    - SSAO
    - MotionBlur

PROBLEMS

- No Transparent objects (Why?) / Solution: Alpha to coverage, Forward path
- No Orthographic cam: the cam will always use Forward rendering (performance issues)
- No MSAA (Post-processing AA)
- No MeshRenderer's ReceiveShadows flag
- No ReflectionProbe AnchorOverride
- Camera culling layer mask must at least contain all layers minus 4 arbitrary layers, so 28 of the 32 layers must be set. Otherwise you will get graphical artifacts
- Shadows are still dependent on the number of lights
- Old hardware/consoles/GPUs don't support MRT
- Deferred render requires a high bandwidth which can be a problem even on current-gen consoles with 128 bit GPU bus

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)

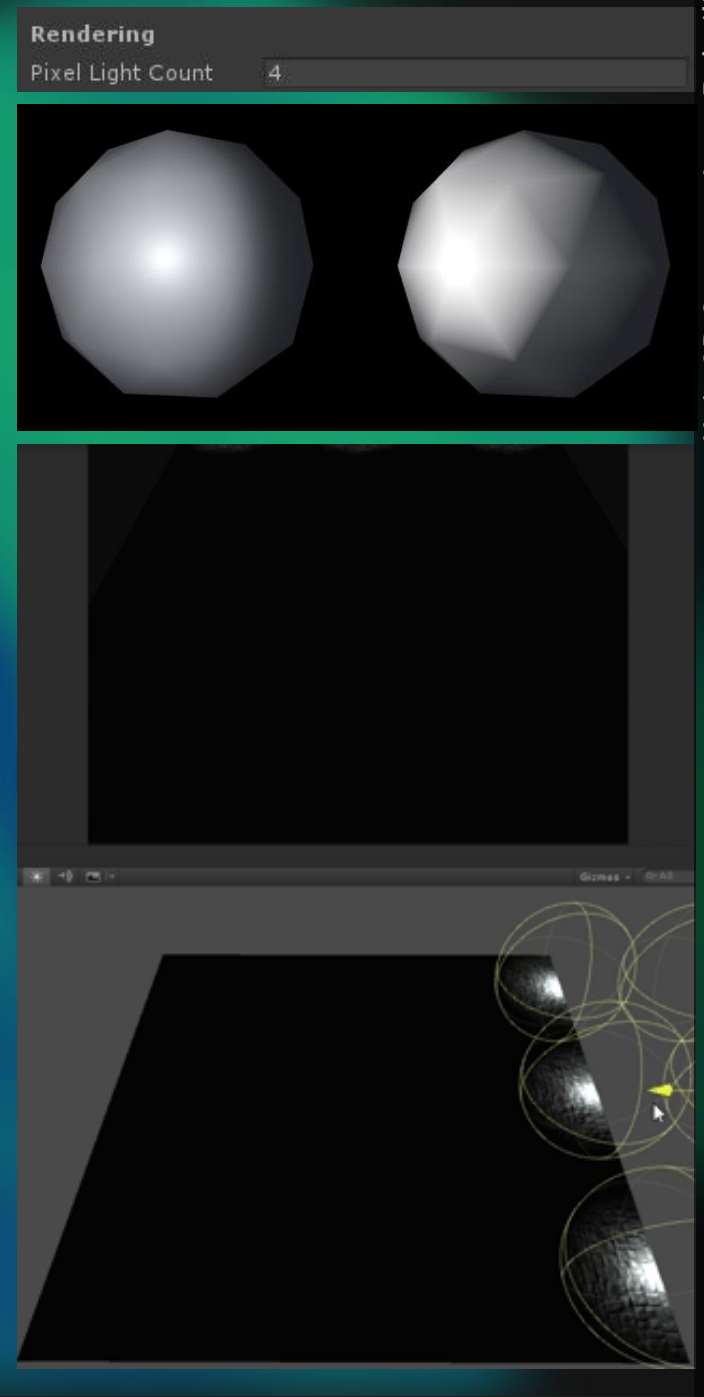
Pixel light / Vertex light

- 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



HDR

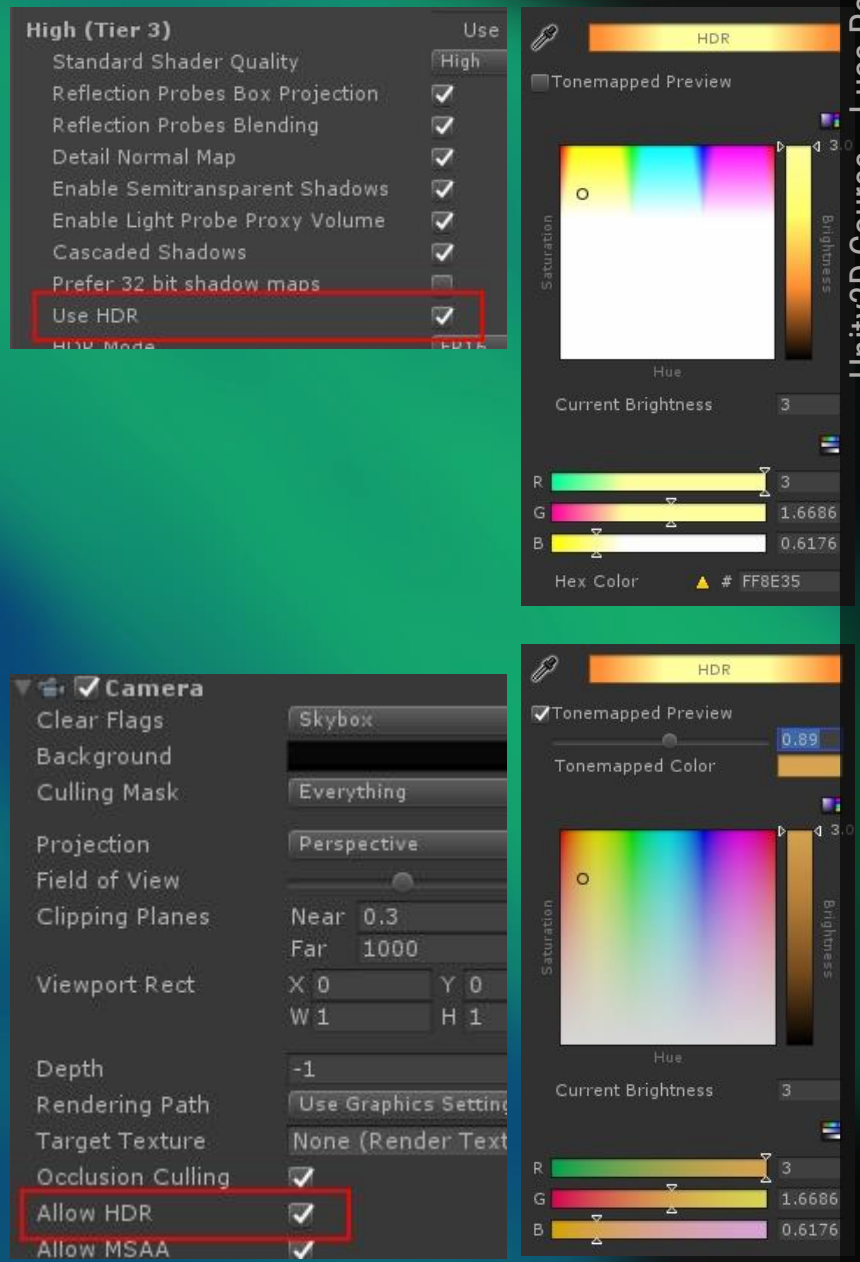
- In Standard rendering RGB values are in [0,1] range (LDR), where 1 is the max intensity for the display device
- Human eyes works differently
  - Adjust to local lighting conditions: the same obj could look white or grey, in a dimly litroom or in full daylight
  - More sensitive to difference between darker shades than lighter shades
- HDR allow to store color values outside the [0,1] range
- Post-processing effects (Bloom, Glow, etc) 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
- Graphics/UseHDR

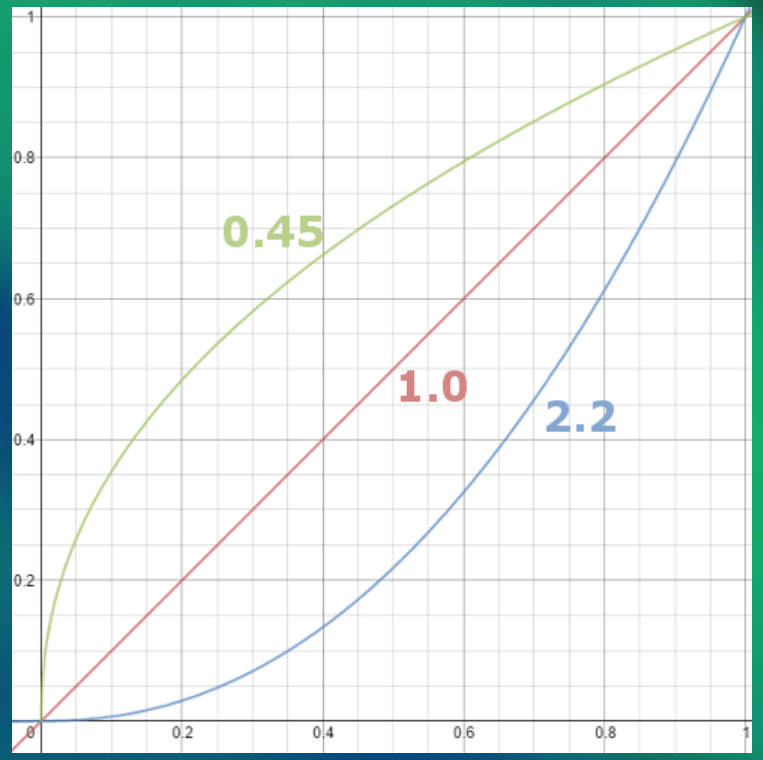
ColorPicker

- 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

- 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 are stored with gamma 0.45 (images in gamma space), so when they are displayed on screen they appear like the original (0.45 x 2.2 = 1.0)

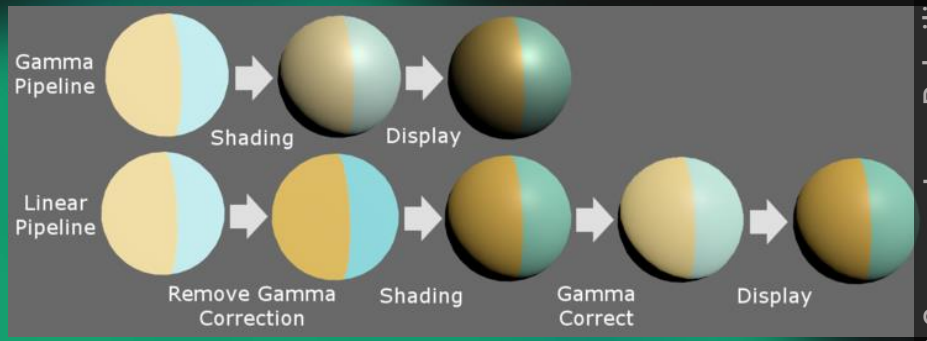


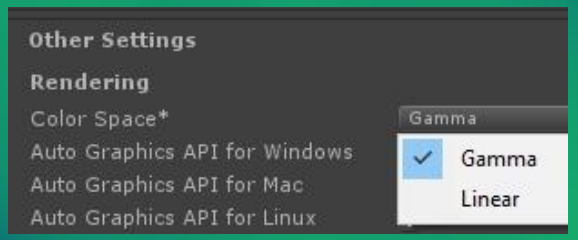
Camera Img > * .45 > Hard drive > * 2.2 > Monitor Out

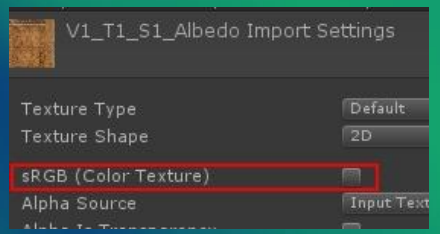
Gamma & Linear space

Unity Pipeline

- Gamma Shaders receive gamma corrected textures
  - 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 in Gamma space) > Apply shading > Gamma correct > Monitor
  - sRGB flag ON if texture content is stored in Gamma Space > allows to bypass sRGB sampling
- sRGB flag OFF 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







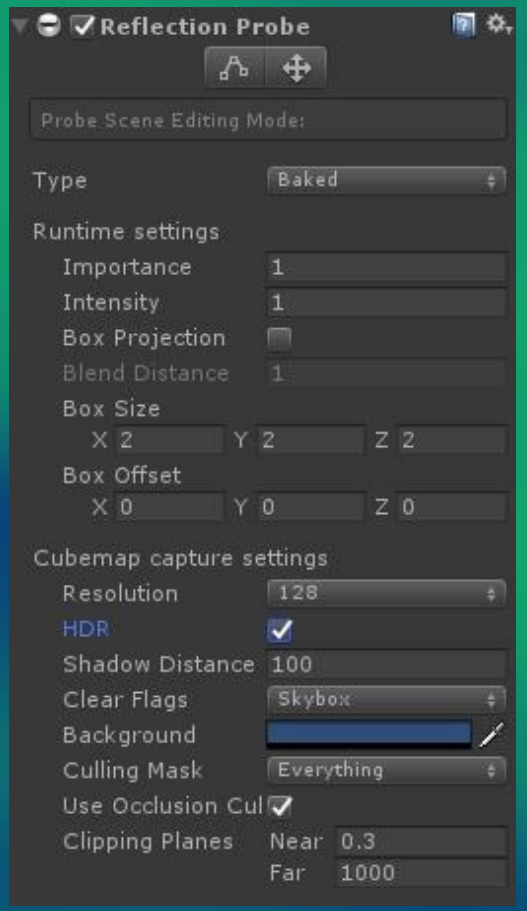
Reflection Probes

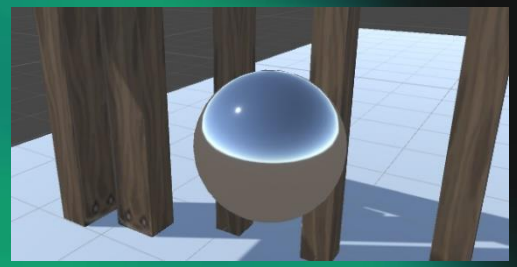
- StandardShader with Smoothness == 1 => Environment reflections
  - Set them in Lighting/Environment/SkyboxMaterial
    - Deferred rendering: works
    - Forward rendering: enable Reflections flag on Standard shader material

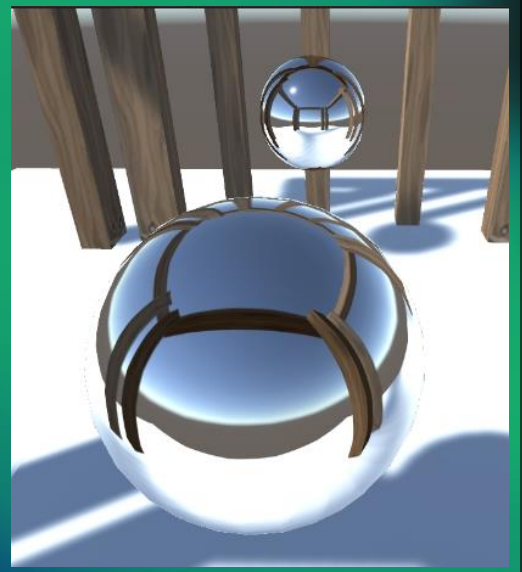
What if we want to reflect also objs around us? We need Reflection probes

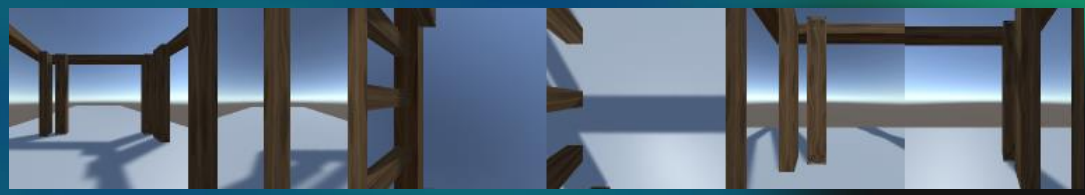
- Baked Reflected objs must be ReflectionProbeStatic
- Custom Provide your own Cubemap
- Realtime
- HDR Should High Dynamic Range rendering be enabled for the cubemap (save in PNG, EXR)
- CullingMask refinement

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









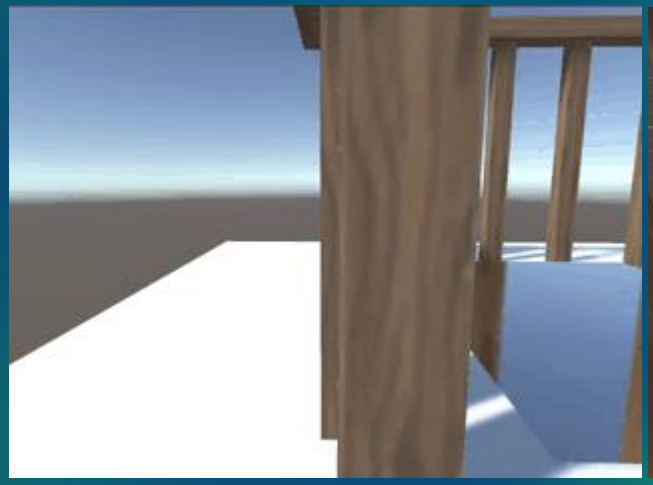
RP / Realtime

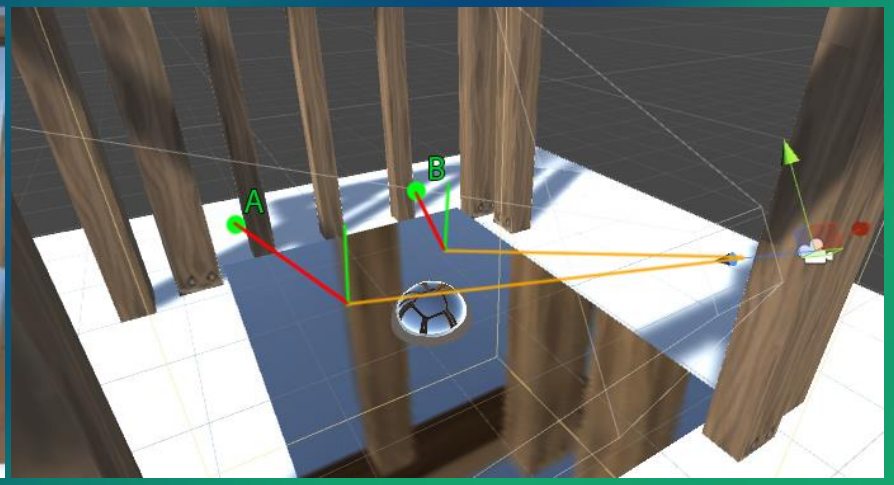
Realtime RefreshMode

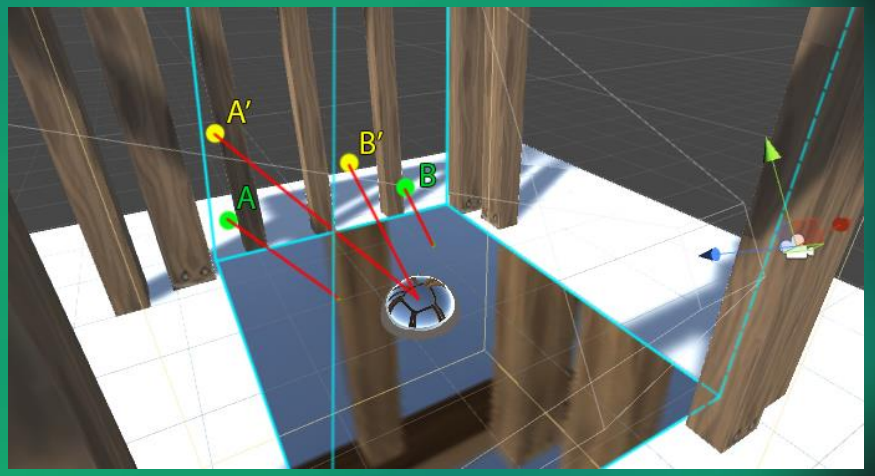
- OnAwake() Useful if the scene is set up at run-time but does not change during its lifetime
- EveryFrame
- ViaScripting
  - ReflectionProbe.RenderProbe()
  - Update only if large enough passing objs
- Timeslicing
  - AllFacesAtOnce 6 cubemap faces rendered on the same frame, then blurring operation using mipmaps will follow (full updt in 9 frames)
  - IndividualFaces 1 cubemap face per frame (full updt in 14 frames)
  - NoTimeSlicing full updt in 1 frame. Very expensive!

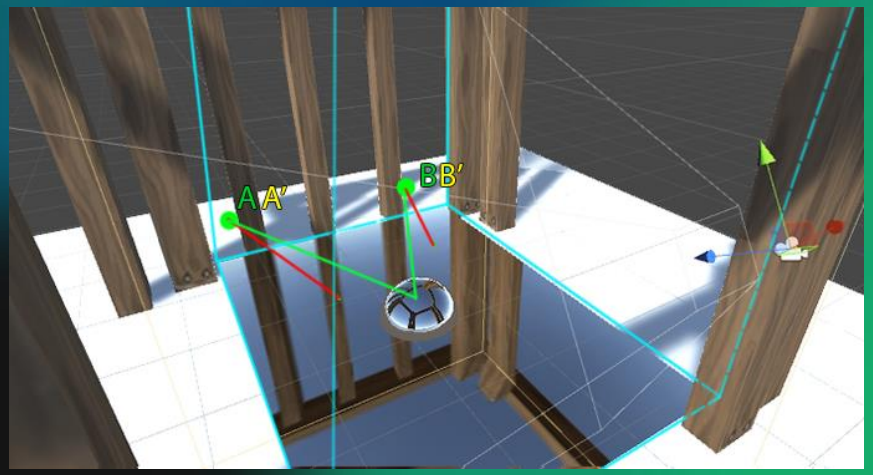
RP / Box Projections

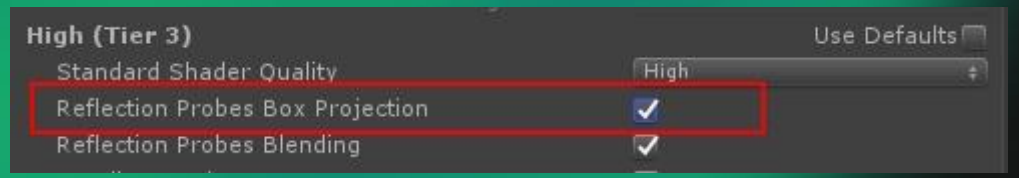
- How to correctly setup Baked Planar reflections?
- Box Projection flag is also very useful for multiple objects in the scene (let's try on our spheres)











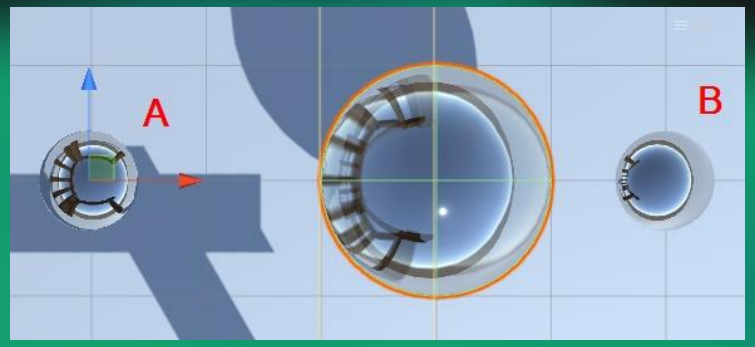
RP / Interpolation / Bounces

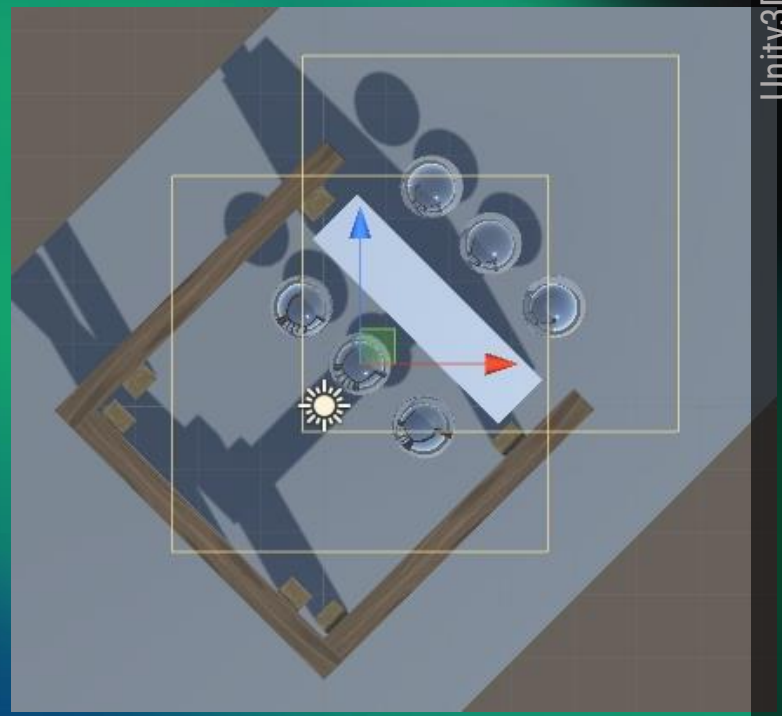
Interpolation

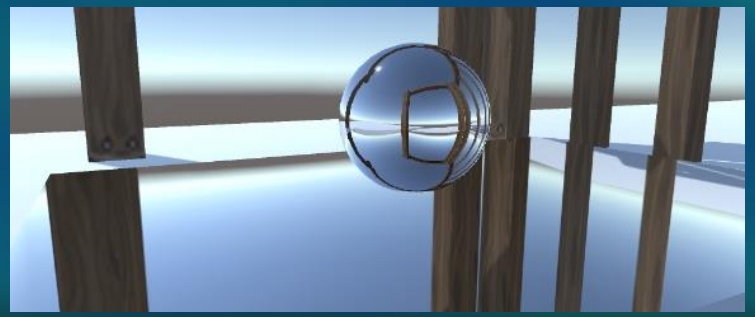
- Based on Importance value
- BlendProbes
  - The Sphere overlaps A by 1 unit, B by 2 units
  - A & B have same importance
  - ReflectionProbeContribution on Sphere = overlap / (Importance + # of probes)
    - Probe A 1.0 / (1.0 + 2.0) = 0.33
    - Probe B 2.0 / (1.0 + 2.0) = 0.67
- BlendProbes&Skybox
- Simple
- AnchorOverride Doesn't work in deferred mode
  - Useful for non diagonal room spaces

How to reflect the reflections?

- Lighitng/EnvironmentReflections/Bounces Very expensive, not the same frame





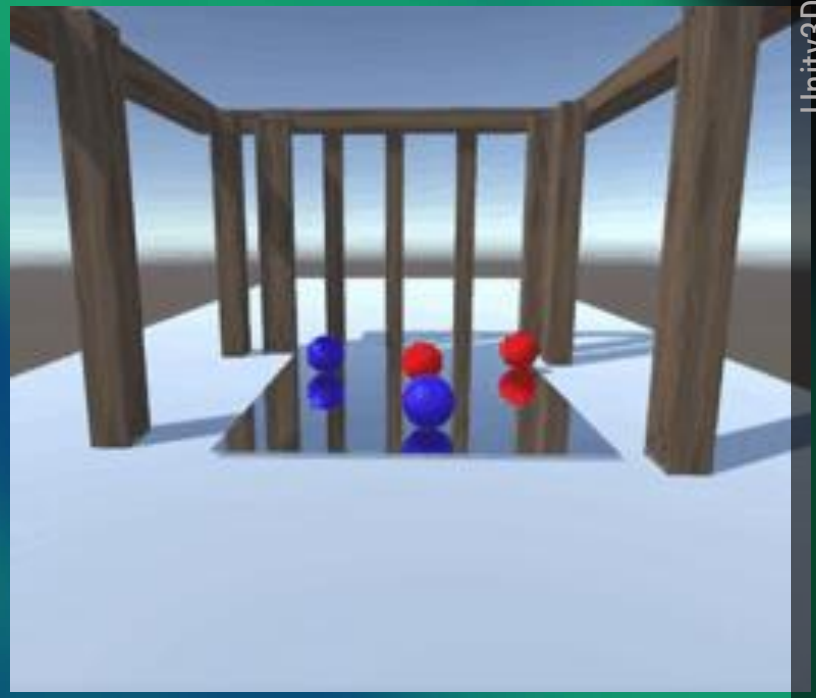


Ex 40 – Magic Mirror

- Open Mirror_Ex scene
- Transform Mirror_RProbe into a mirror, using Reflection probes

EXTRA
- Make Mirror_RProbe work also on Vertical planes

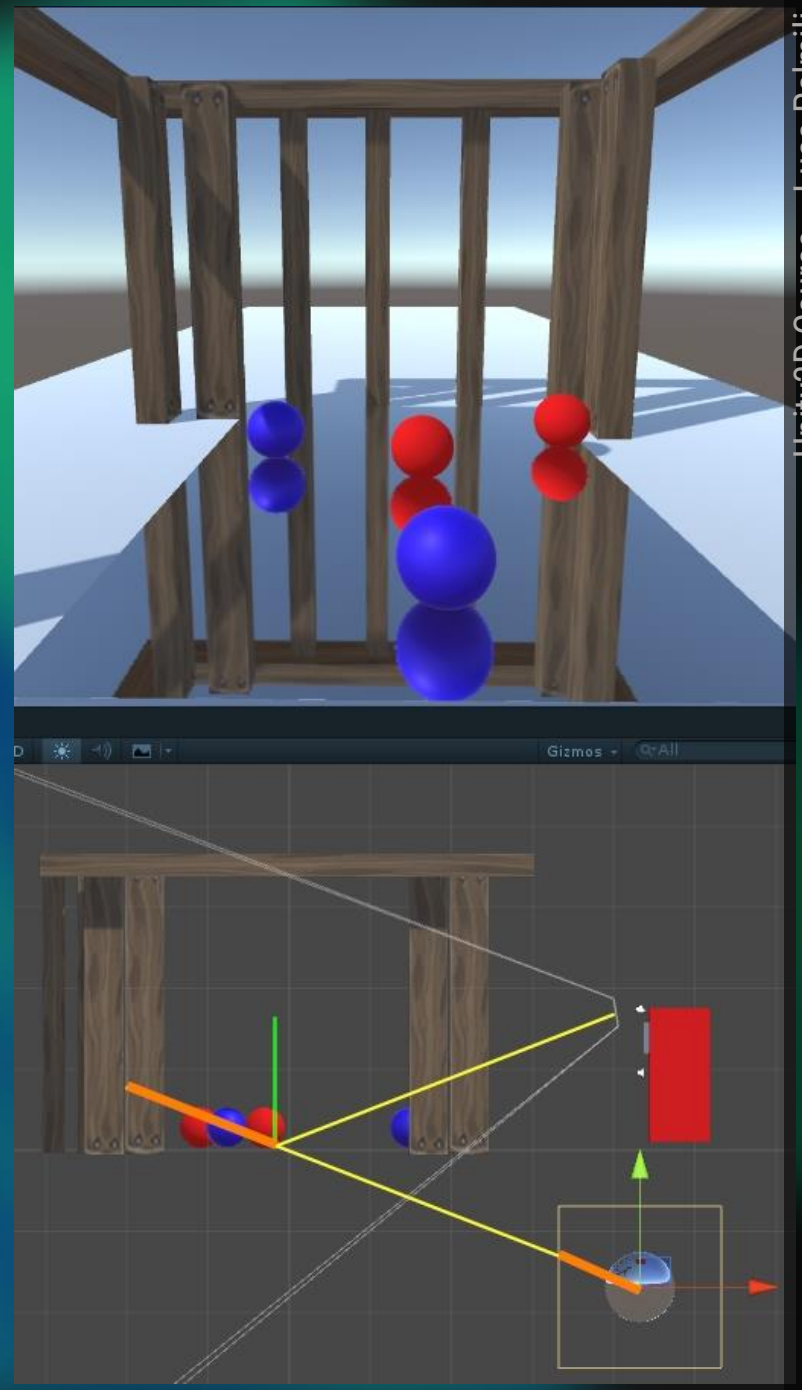
HINTS
- Think about how mirrors work, then
- Think about how reflection probes work



Ex 40 – Magic Mirror

HINTS

- Think about how mirrors work, then
- Think about how reflection probes work
- Use MeshRenderer/Lighting/AnchorOverride stuff (hence Forward rendering)
- Use Realtime lightprobes

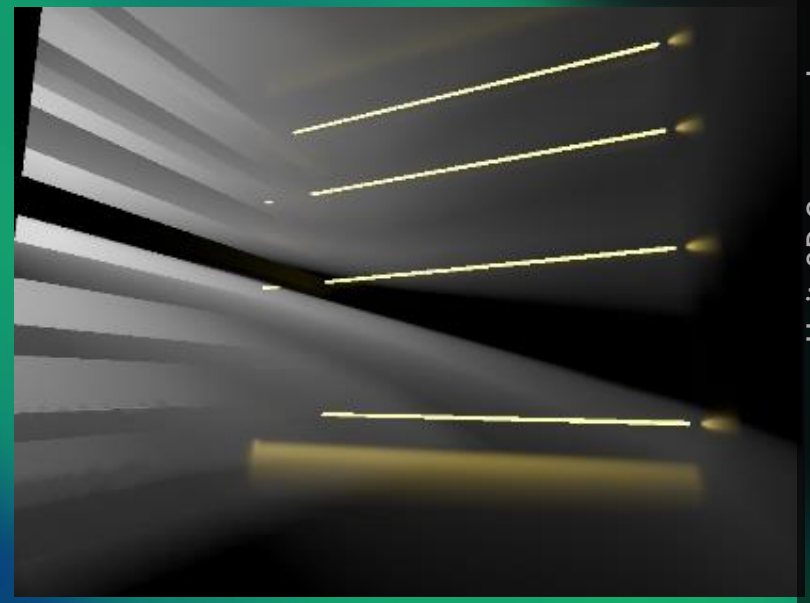


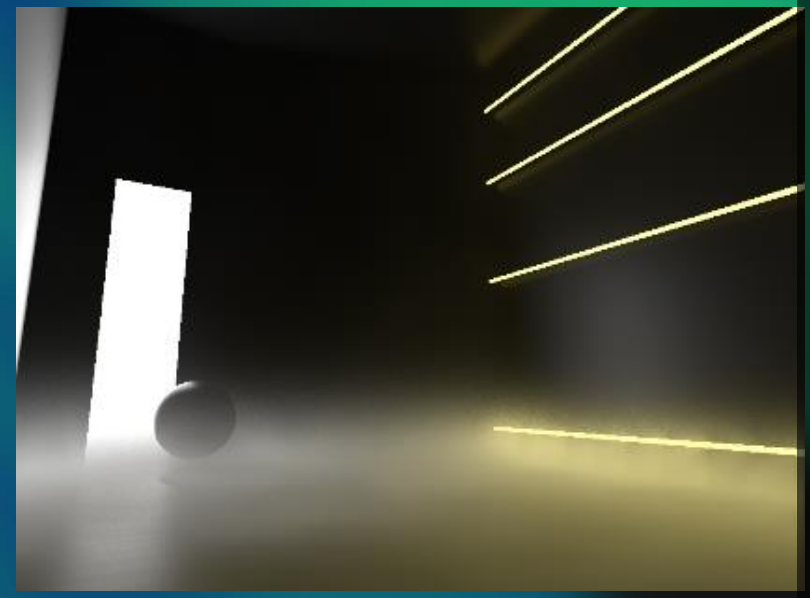
Light Shafts, Volumetric Fog / Area / Tube lights

- Does not ship with Unity
- Volumetric light Shafts [https://github.com/robertcupisz/LightShafts]
- Volumetric Fog, Area/Tube lights in realtime (used in Adam) [https://github.com/Unity-Technologies/VolumetricLighting]

- Here you can find a Zip with both Repositories [http://bit.ly/VOLUMETRIC_LIGHTS]
    1. Download the Zip
    2. Move the unzipped folders into your /Asset folder

[VolumetricLights.scene]





Scripting

Update GI

- We can animate Emission property even if the component is static, and therefore change the realtime GI
- EmMat = GetComponent<MeshRenderer>().material;
  EmMat.SetColor("_Emission", newColor);
  //Update the realtime GI system if we have a texture..
  GetComponent<MeshRenderer>().UpdateGIMaterials();
  //Or not
  DynamicGI.SetEmissive(emissiveRenderer, c);
- Use [ColorUsageAttribute] to allow HDRColor public inspector input

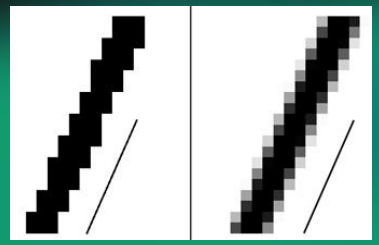
QualitySettings

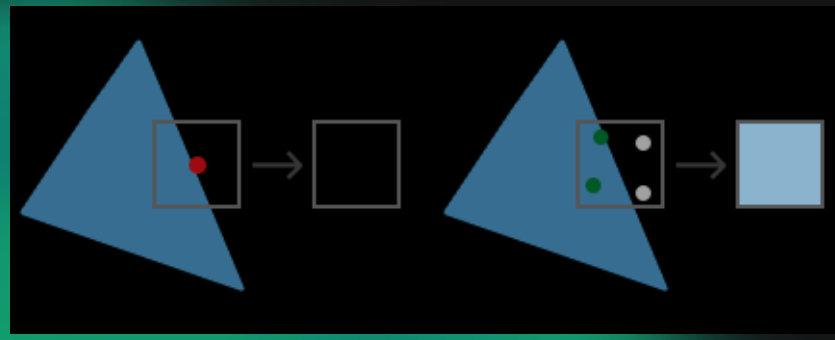
[Realtime_ex, EmissiveAnimation.cs, Qsettings.cs]

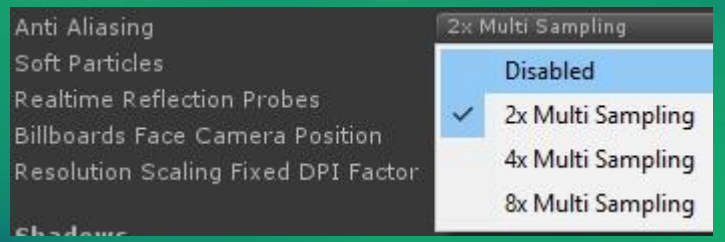
AntiAliasing

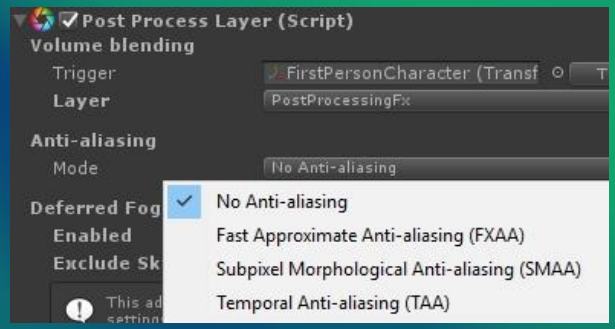
- Aliasing
- Supersampling (SSAA)
- Multisampling (MSAA) Enable it in QualitySettings and Camera component
  - Doesn't work in Deferred mode

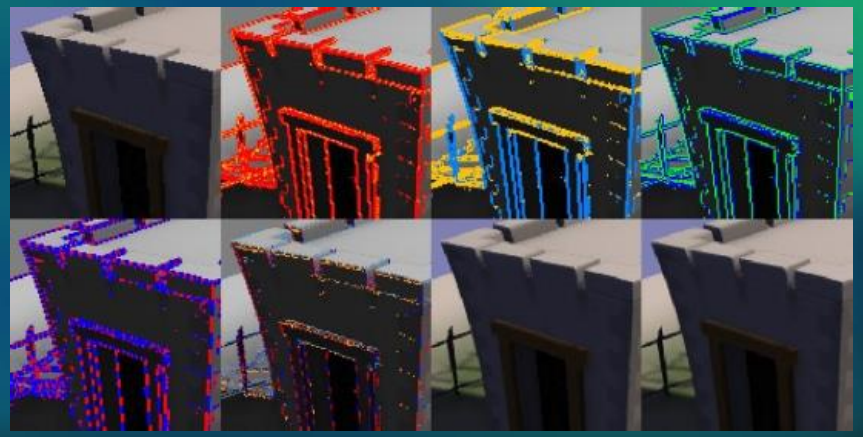
Post Processing
- Enable it in PostProcessingLayer/AAMode
- Fast Approximate AA (FXAA)
  - Only edges
  - Loss of visual clarity especially on objs in motion
  - Cheapest technique > recommended for platforms that don't support motion vectors
- SubPixel Morphological AA (SMAA)
  - Only edges
  - Use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







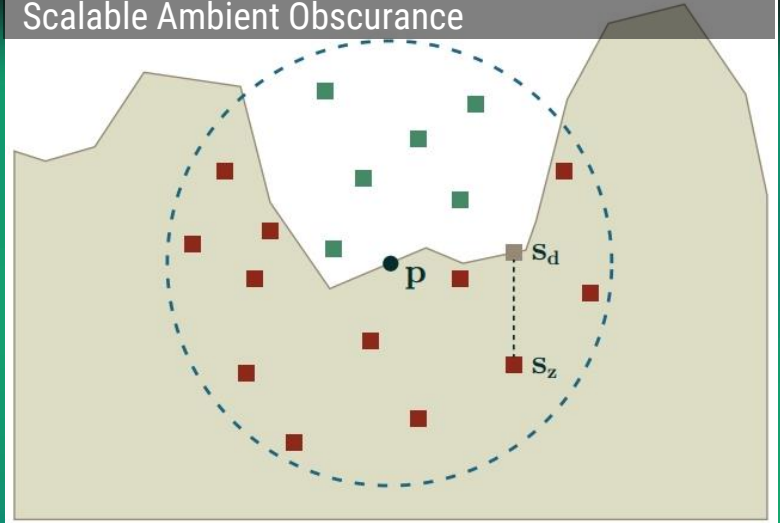




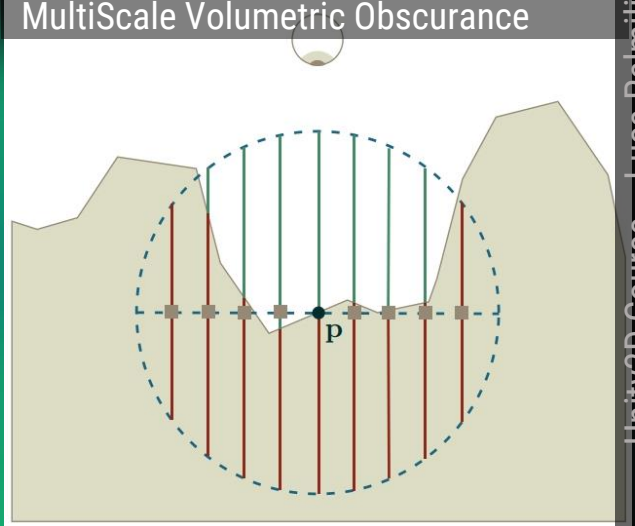
Screen Space Ambient Occlusion

- 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

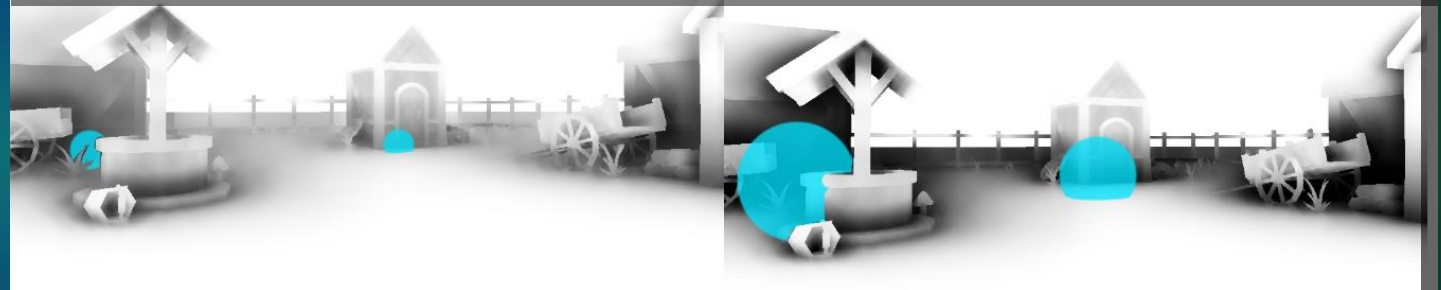
Scalable Ambient Obscurance



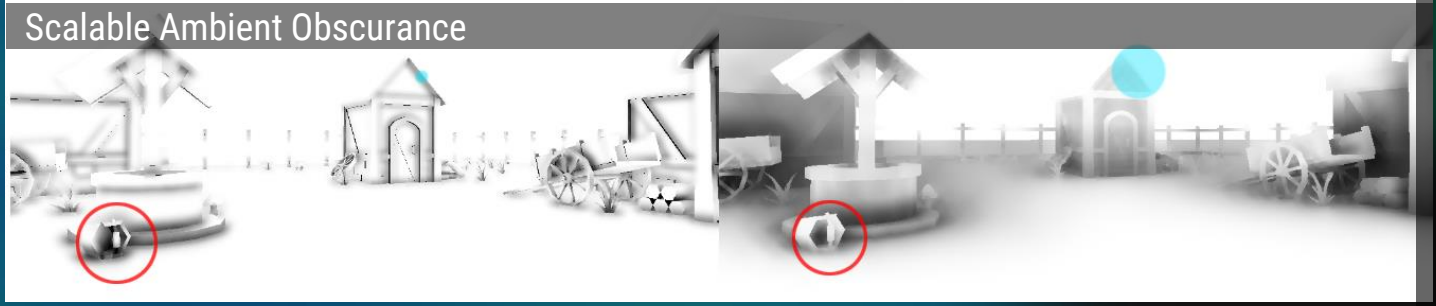
MultiScale Volumetric Obscurance



MultiScale Volumetric Obscurance



Scalable Ambient Obscurance



MotionBlur

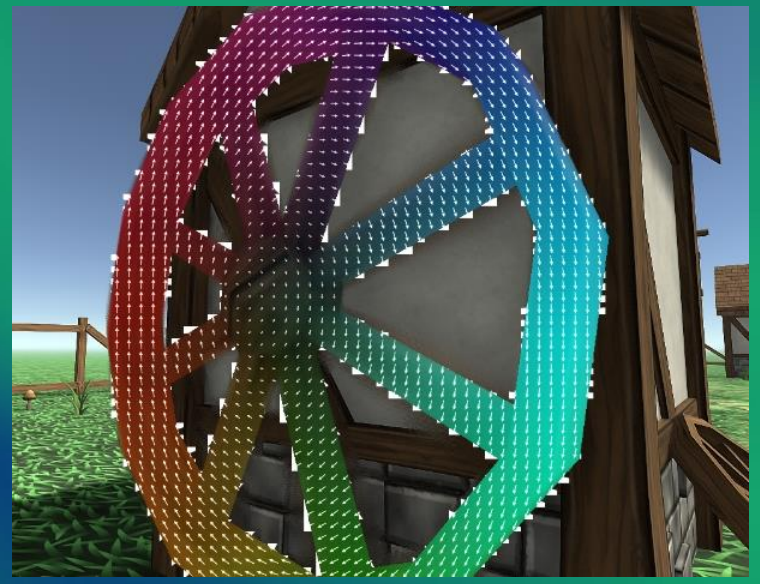
- Occours when objs are moving faster than the camera's exposure time
- Depends on relative motion

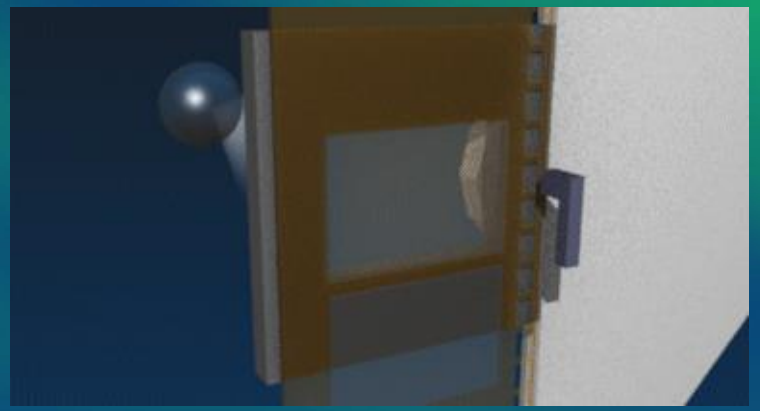
How to achieve MB?
- Adding geometry
- Accumulation buffer
- Velocity buffer / Motion Vectors

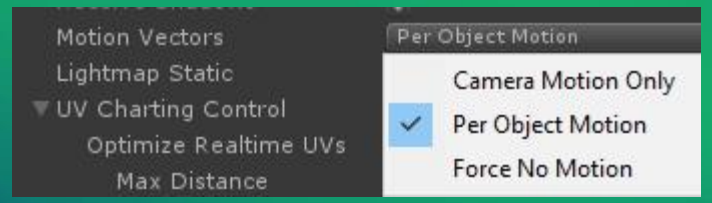
- ShutterAngle (from Cinema)
  - 24 FPS
  - Use 1/48 as normal shutter speed
  - 1/48 = 180°, 1/24 = 360°
- SampeCount

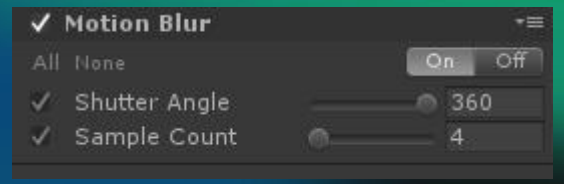
- Unity doesn't support per-particle MB

[MotionBlur_Test, Village_00]







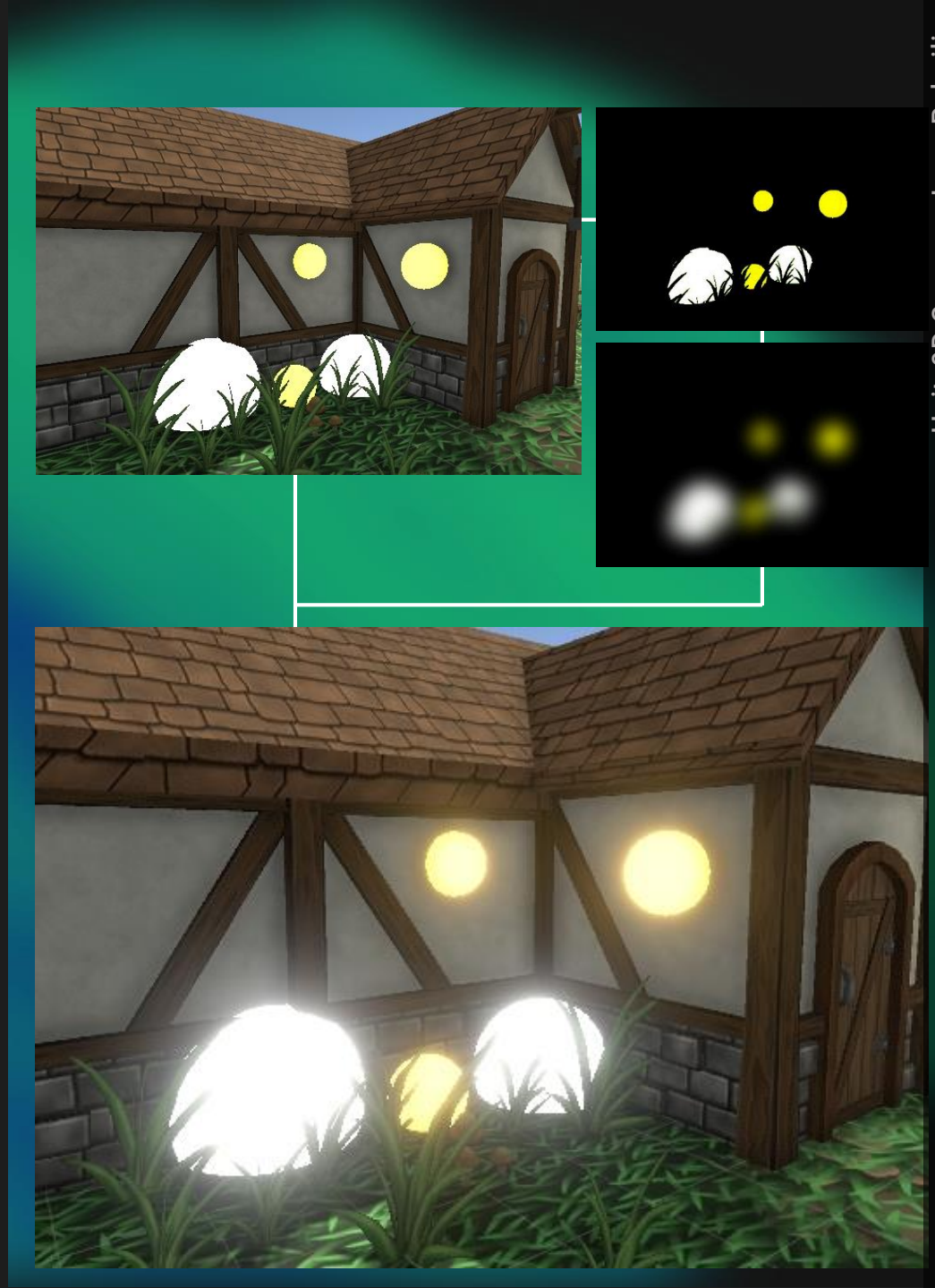


Bloom

- 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

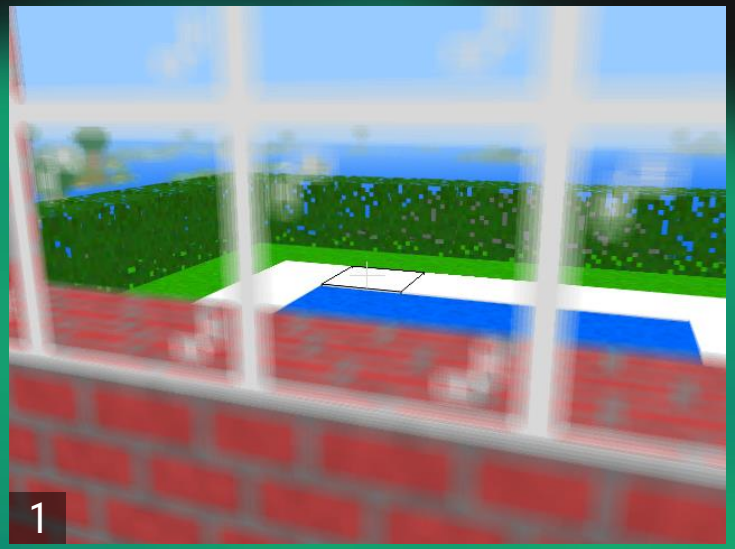
- Objs with greater HDR brightness will receive more bloom

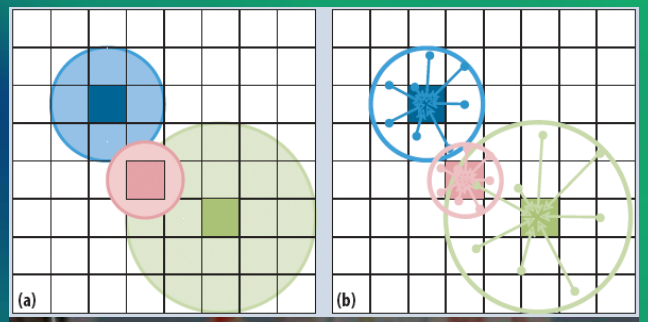


Depth of Field

- 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

- 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]
  1. Render the image 3 times: sharp, blurry, blurrier
  2. Use the depth buffer to know the distance from the focal plane
  3. 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





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

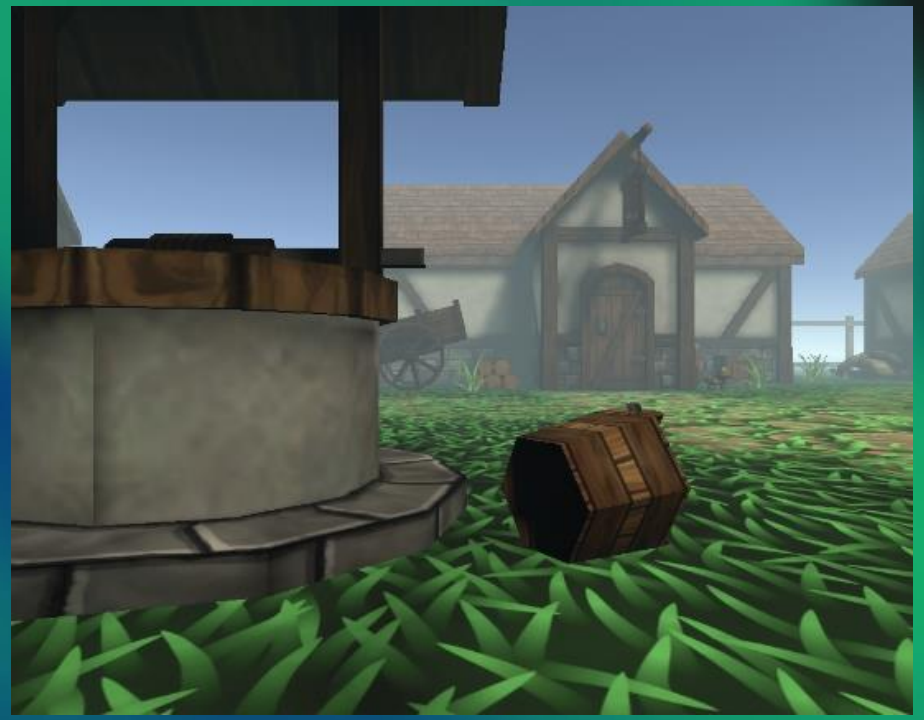
$$c_p = f\,c_s + (1 - f)\,c_f$$

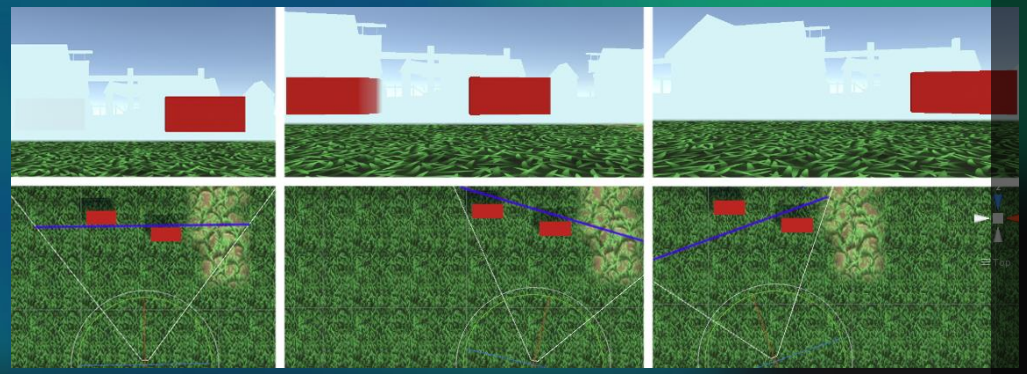
- Enable it under LightingSettings/OtherSettings/Fog
- PPStack v2 enables fog also in deferred mode
  - PostProcessingLayer/DeferredFog/Enable

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

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

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





PostProcessing Stack v2

- Does not ship with Unity [https://github.com/Unity-Technologies/PostProcessing]
  1. Download the Zip
  2. Move the unzipped folders into your /Asset folder
- Algorithms that operates on every pixel of a render texture to apply specific effect: O(screen_size)
- Uber Effect

Scene setup

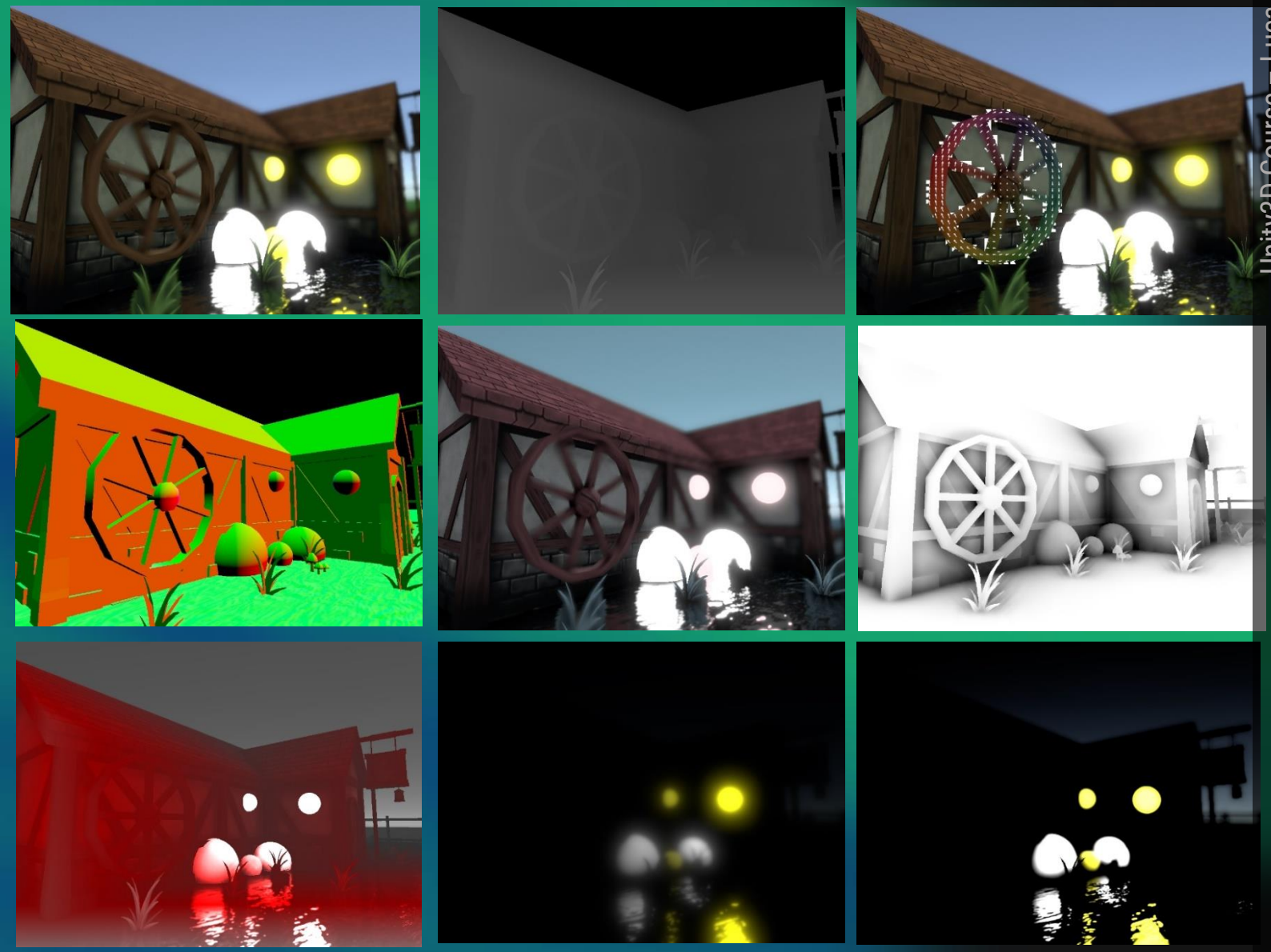
1. Create>PostProcesingProfile The PostProcessProfile is an asset, so we can:
   - Save play mode changes
   - Swap profile during run-time (e.g. Gameplay / Flashback / Gamplay)
2. Camera>Add PostProcessingLayer
   - VolumeBlending If this tansform will fall into a Local PostProcessingVolume, will trigger that volume
   - Layer Every PostProcessingVolume applied to this camera must belong to this layer
3. Create>EmptyObject and add a PostProcessingVolume to it. Mark it as isGlobal
4. Set the EmptyObject layer according to the camera PostProcessingLayer setup

- Local PostProcessingVolume
  - Add a Collider with a trigger

[Village_start_00.scene]

PPFx Debug

- Add PostProcessDebug component to the Camera with PostProcessLayer attached



Scripting

Dynamically create a global volume on the scene

- PostProcessVolume newVol = PostProcessManager.instance.QuickVolume(LayerID, priority, PostProcessEffectSettings)

Override a profile

- Similar to how material scripting works in Unity
  - Modify the shared profile
    - Changes will be applied to all volumes using the same profile
    - Modifies the actual asset and won't be reset when you exit play mode
  - Request a clone of the shared profile that will only be used for this volume
    - Changes will only be applied to the specified volume
    - Resets when you exit play mode
    - NB: Destroy the profile when you don't need it anymore

Adds and returns an effect you created

- PostProcessEffectSettings AddSettings(PostProcessEffectSettings effect)

Removes an effect from the profile

- void RemoveSettings<T>():. Will throw an exception if it doesn't exist

Check the existence of this setting on the profile

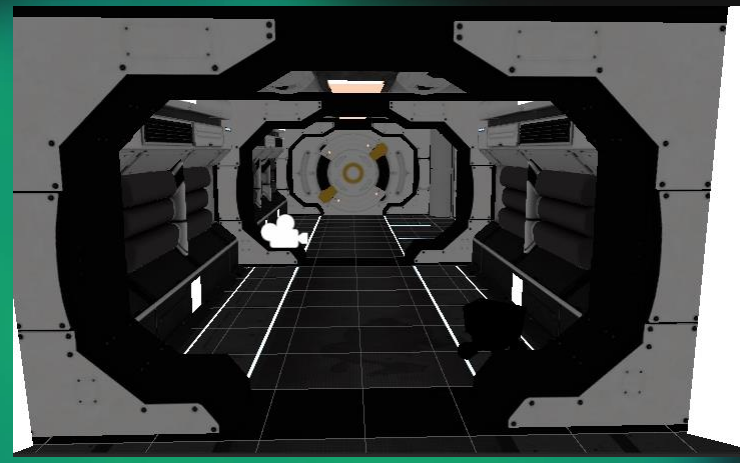
- bool TryGetSettings<T>(out T outSetting)

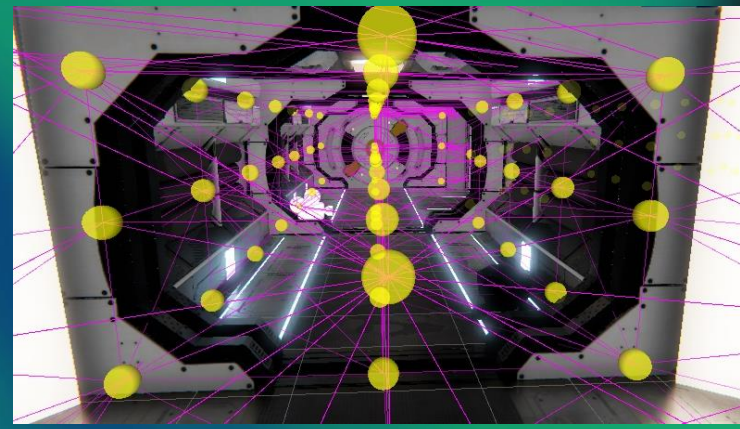
[ChangeProfile.cs]

Ex 41 – Wrap it up

- Open TheCorridor project (see the download link below) and customize it
- Choose your style
  - Dark, White & polish, Old style, SciFi Horror

Write a short description of your scene
- Lights, Shadows
- Texturing
  - Find & replace textures using StandardShader
- Particles
- Realtime GI / Baked GI (Choose the correct mode)
- ReflectionProbe
- LightProbe, LightProbeProxyVolume
  - Create script to distribute LP inside an array of BBoxes
- Gamma/Linear, HDR
- PostProcessingStack
  - AO, Aberration, interactive FXs, LUTs
- Fog, Volumetric Fog on the floor, tubelights, area lights
- Animations
  - Animate doors, camera, particles





http://bit.ly/CORRIDOR_EX