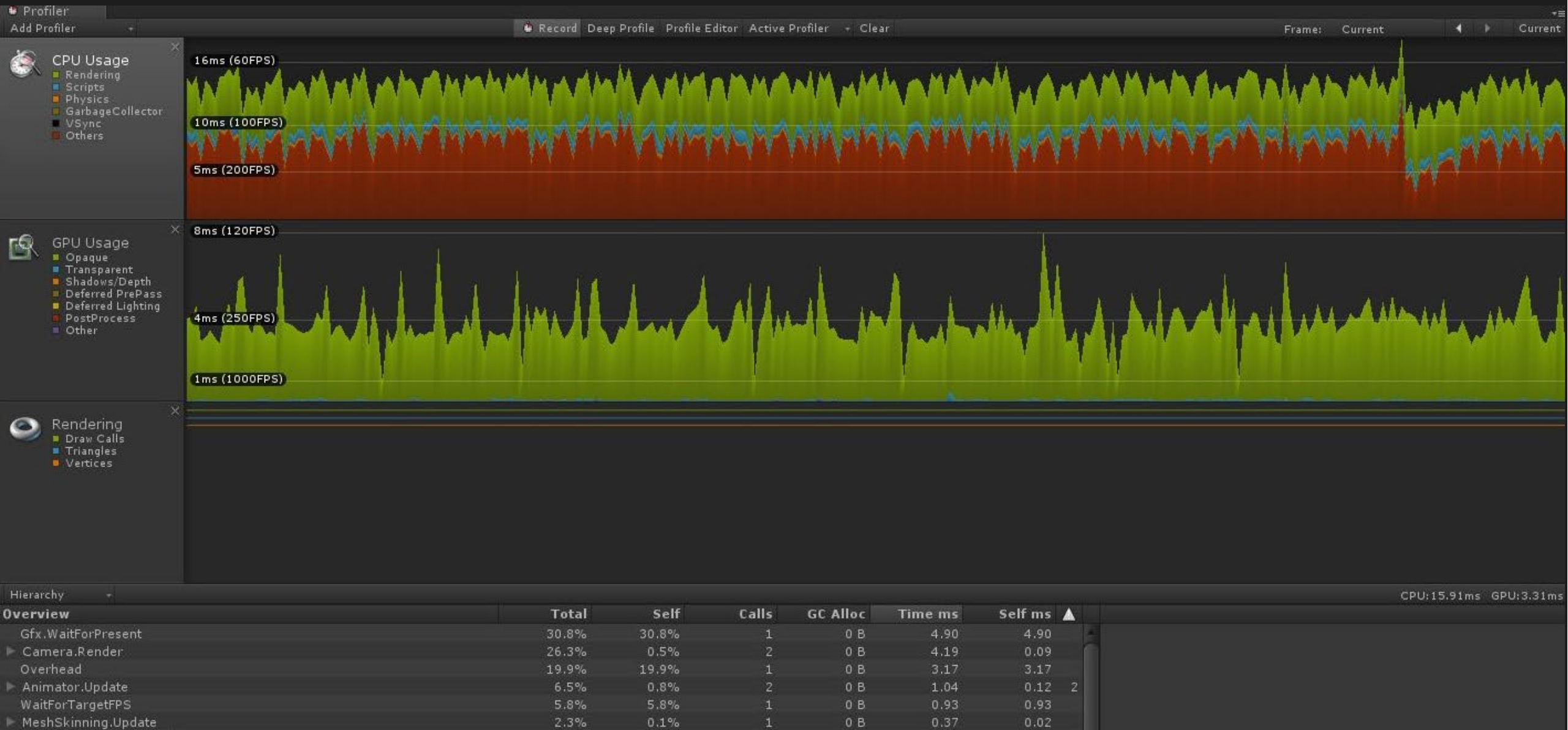
Optimization



Performance, why?

- Positive reviews => Improve sales
  - Required for release
    - Oculus Quest (>70 FPS)
- Increase pro skills => easier to get a job

- Hard to achieve
- The effort to increase performance is underestimate
- Often comes before release or after user complains: and often it is too late
- Vast topic
- You are never finished: android/Oculus ships a new patch and it could screw your performance

Tools

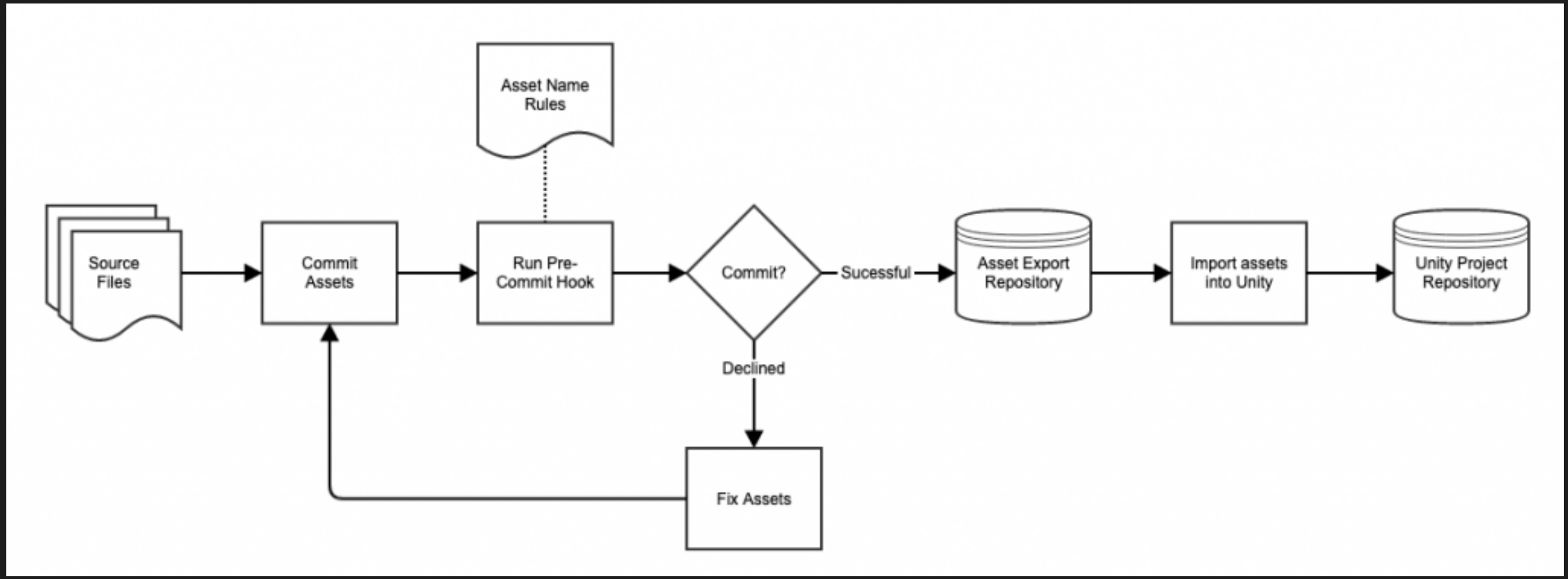
1. **General**: Unity Profiler, Unity Debug Visualization, PIX, OVR Metrics, NG tools, command line, ...
2. **Draw Calls**: Unity Frame Debugger, RenderDoc, Mesh Baker, SRP visualizers...
3. **Shaders**: Mali OC, RenderDoc, VS Code, ...
4. **Geometry**: Simplygon, AutoLOD, ...
5. **Code**: VSCode/VS/Rider, Dnspy, Simpleperf, Perfetto ...
6. **Professional**: asset/shader cache servers, CD/CI, architecture, ...
7. **Memory**: dumpsys, addressables event viewer, unity memory analyze

The Asset DB

- Unity needs to convert the data from the asset's source file into a format that it can use in a game or real-time application. It stores these converted files, and the data associated with them, in the Asset Database. The conversion process is required because most file formats are optimized to save storage space, whereas in a game or a real-time application, the asset data needs to be in a format that is ready for CPU, GPU, audio hardware, to use immediately
- If you subsequently modify an asset's source file that you have already imported (or if you change any of its dependencies) Unity reimports the file and updates the imported version of the data
- The Asset Cache is where Unity stores the imported versions of assets. Because Unity can always recreate these imported versions from the source asset file and its dependencies, you should exclude the files in the Asset Cache from version control. However, if you work as part of a team and use a version control system, it might be beneficial to also use Unity Accelerator, which shares the Asset Cache across your LAN
- Unity maintains two database files in the Library folder, which together are called the Asset Database
    - Source Asset Database: Library\SourceAssetDB Contains meta-information about your source asset files which Unity uses to determine whether the file has been modified, and therefore whether it should reimport the files. This includes information such as last modified date, a hash of the file's contents, GUIDs and other meta-information
    - Artifact Database: Library\ArtifactDB Artifacts are the results of the import process. The Artifact database contains information about the import results of each source asset. Each Artifact contains the import dependency information, Artifact meta-information and a list of Artifact files

Asset Name Validation

- Unity's asset pipeline starts the moment you add a new file into the project. For the bigger picture, however, let's assume the asset pipeline begins at the point where the artist's creative work ends: This could indeed be the moment someone imports the asset files into Unity, or it could be the moment an artist runs an export script from a 3D graphics application, such as Autodesk Maya

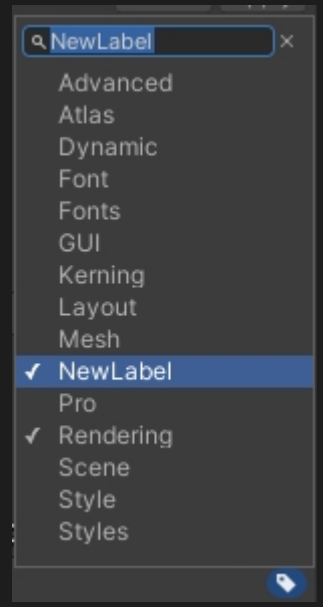


Asset DB API

- The Asset Database also provides an AssetDatabase API that you can use to access Assets, and control or customize the import process

- Importing Unity normally imports assets automatically when they are dragged into the project but it is also possible to import them under script control
  AssetDatabase.ImportAsset("Assets/TexturesToImport/texture00.jpg"); //the file is in [projectfolder]/Assets/TexturesToImport
- Other Operations import and inspect the AssetDatabaseIOExample.cs script: AssetDatabase menu item will be created
  Try it
  - Create /Asset/Tmp folder to test it. Put inside a couple of textures
  - FindAssets() will use label and name search
    - you'll need the NewLabel label in your project. To add a Label, use the inspector. To create a new label, just write in the search field the new labelName and then press Enter. One texture in /Tmp folder should have this label
    - one texture in /Tmp folder should have 'test' in its name

- GUID is the Globally Unique Identifier. Scenes, Prefabs, and other Unity files (assets) has a GUIDs to uniquely identify files within a single project and between different projects. GUID is originally written in .meta file of targeted file, and is generated by Unity app automatically when you
  - import a new file into Unity Project
  - duplicate the file (new GUID for the duplicated file)



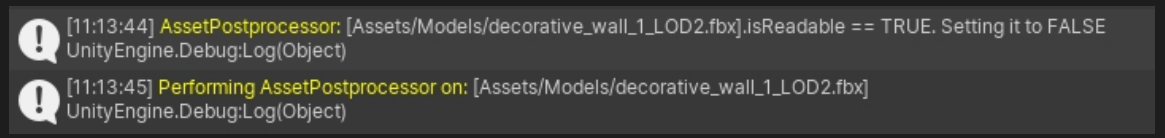
[AssetDatabaseIOExample.cs]

Import Asset pipeline 2.0

- Up until 2019.2 the Library folder was comprised of the GUIDs of Assets being their filename. Thus, switching from a platform to another platform would invalidate the Import Result in the Library folder, causing it to be re-imported every time you switch platforms
- With the new Asset Import Pipeline, Unity removed the GUID to File Name mapping. This allows us to have multiple revisions per Asset, which allows us to have Import Results which work across different configurations. For Fast Platform Switching, we could have an Import Result per platform, so that when you switch platforms back and forth the Import Result is already there

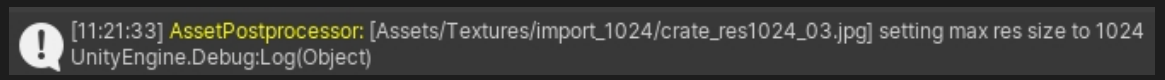
Scripted Importers

- You can define your own importers to add import functionality for new file types, or to override the importer for an existing file type. These importers are called Scripted Importers
- Many of the problems found in real projects occur because of human mistakes, for example change the import settings of existing Assets
- For any project of significant scale, it is best to have a first line of defence against human errors
- The AssetPostprocessor class in the Unity Editor can be used to enforce certain minimum standards on a Unity project. This class receives callbacks when Assets are imported. To use it
  - inherit from AssetPostprocessor and implement one or more OnPreprocess methods, like:
    - OnPreprocessTexture()
    - OnPreprocessModel()
    - OnPreprocessAudio()
  - Use assetImporter field to get the right Importer
    - ModelImporter modelImporter = (ModelImporter)assetImporter;
- There are also PostProcess version of that methods
- Keep these files in a folder called /Editor (otherwise they will generate errors during compiling)



Try it

- Create TextureResolutionPostprocessor.cs
- Use the OnPreprocessTexture() method
  - If the texture name contains “res1024”, change its maxTextureSize to 1024
  - Use TextureImporter.assetPath, string.Contains(), TextureImporter.maxTextureSize
  - To test it, change one name inside /Textures/Import_1024 files, set it to 2048, and change name again: it should automatically set its resolution to 1024



[ReadOnlyModelPostprocessor.cs, TextureResolutionPostprocessor.cs]

Quality switcher

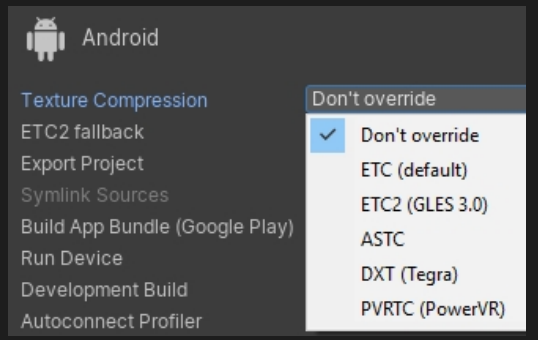
- Import [QualitySwitcher_00]
- QualitySwitcher.cs shows you how to switch quality settings
- QualitySettings.SetQualityLevel(int index, bool applyExpensiveChanges = true);
- Changing the quality level can be an expensive operation if the new level has different anti-aliasing setting. It's fine to change 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 always applied

[QualitySwitcher_00, QualitySwitcher.cs]

Common asset rules

Textures

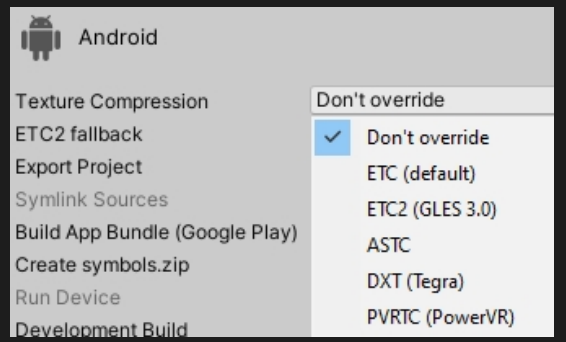
- read/write flag
  - causes a Texture to be kept twice in memory: once on the GPU and once in CPU-addressable memory. This is because, on most platforms, readback from GPU memory is extremely slow. Reading a Texture from GPU memory into a temporary buffer for use by CPU code (e.g. Texture.GetPixel) would be very nonperformant). Set this to ON only when manipulating Texture data outside of Shaders (such as with the Texture.GetPixel and Texture.SetPixel APIs)
- Disable Mipmaps for objects that have a relatively invariant Z-depth relative to the Camera
  - This would to save about a third of the memory required to load the Texture. In general, this is useful for ScreenSpace UI and Textures and other Textures that are displayed at a constant size on screen
- Use the correct texture compression format
  - If it is unsuited to the target platform, Unity decompresses the Texture when it is loaded, consuming both CPU time and an excessive amount of memory
  - On newer mobile devices, you should favour ASTC compressed Texture formats
    - More flexible than previous formats due to support for various block sizes.
      Using this format is a good way to optimize the size of your game
    - If ASTC is not available on your target devices, use
      - ETC2 on Android
      - PVRTC on iOS

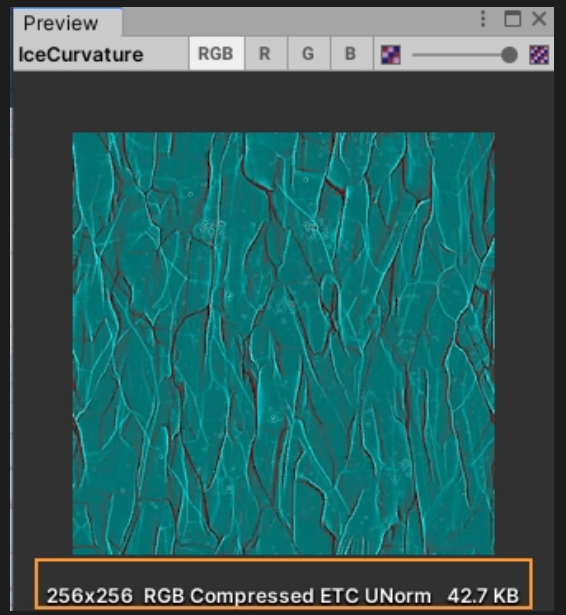


- Enforce sensible Texture size limits
  - For many mobile applications, 2048x2048 or 1024x1024 is sufficient for Texture atlases, and 512x512 is sufficient for Textures applied to 3D models
  - You can change global texture quality settings from Psettings/Quality/TextureQuality

TextureCompression

- Textures are worst offenders. Can very well be 70-90% of the final build
  - Less download
  - If I have to remove something from my device > Let's sort by size my apps and… remove the bigger ones
- RAM/VRAM space
- Memory bandwidth (RAM > VRAM, VRAM)
- Crashes, loading times, install rate, retention rate, lower FPS
- GPU doesn't like JPG/PNG
  - Compress textures globally, while GPU access textures locally. We need block-based compression formats.
- Texture format
  - Choose RGB format for uncompressed
- Compression quality
  - Fast/Normal/Best: same amount of space in RAM/VRAM, but worst or better quality. Bigger textures can take hours to compress if you choose Best
- You can override the texture compression in BuildPanel/Android/TextureCompression





Albedo

Platform	No alpha Old/new formats	Alpha Old/new formats
PC	DXT1 / BC7	DXT5 / BC7
Android	ETC2 4 bits / ASTC	ETC2 8 bits / ASTC
iOS	PVRTC RGB 2-4 bits / ASTC	PVRTC RGBA 2-4 bits / ASTC

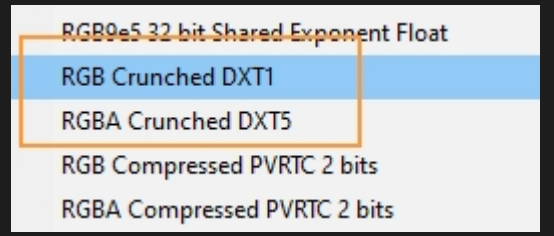
Normal maps (Underline on preferred)

Platform	Format
PC	DXT5 / BC5 / BC7
Android	ETC2 / ASTC
iOS	PVRTC 2-4 bits / ASTC

TextureCompression

- Hardware support (see table on the right)
  - If not supported @runtime, Unity decompresses your texture to RBGA32 @ runtime
  - Works, but loading times & RAM problems
  - Alternative on Android: split APKs, depending on the hardware (difficult to manage)
- ASTC let you control the pixel block size: if you pack 12x12 pixels or 4x4 pixels, the bytes-per-block will be the same, so 12x12 will have a worst quality
- Crunch compression: Once you compress them, there is another step to compress further. Worst quality, but less size in the final package. Once decompressed, it takes the same amount of space in VRAM, like it is uncompressed
- UI have alpha component, use
  - ETC2 / ASTC
- A/B Testing
  - Duplicate textures & Tweak parameters
  - Bind key to swap textures on same renderer
  - Build, compare, decide (blind testing)

Compression format	% Google play devices with support
ETC1	99
ETC2	87
ASTC	77
PVRTC	11
DXT1	0.7



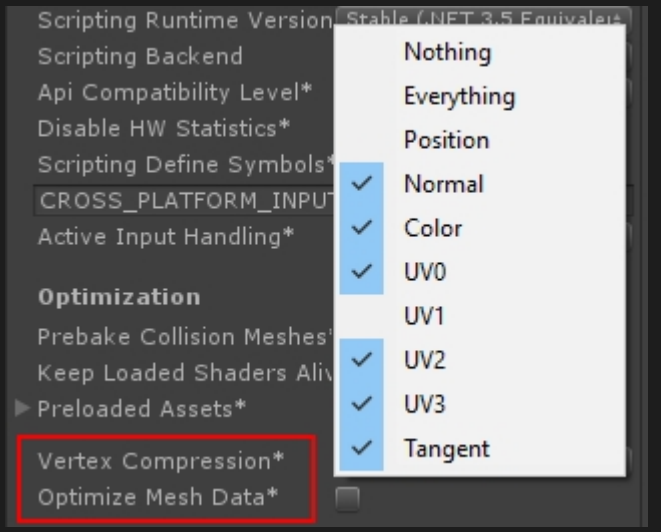
Common asset rules

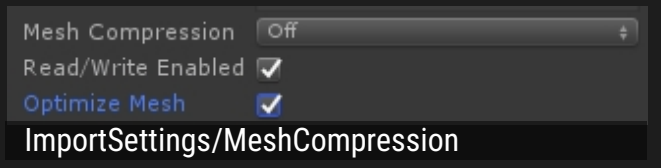
Settings per model:

- read/Write flag
  - Unity requires this flag to be enabled if a project is modifying a Mesh at runtime via script
  - ON The original mesh data will be stored in memory
  - OFF Discard the original mesh data from memory once it has determined the final mesh to use, since it knows it will never change
  - Thumb rule
    - If we use only a uniformly scaled version of a mesh throughout the entire game, leave it OFF
    - If mesh often reappears at runtime with different scales, leave it ON
      - If this data is in memory, can recalculate a new mesh more quickly
- MeshCompression
  - Reduces the number of bits used to represent the floating-point numbers for different channels of a model's data. This can lead to a minor loss of precision, and the effects of this imprecision should be checked by artists before use in a final project

Settings for ALL the meshes in the project, and shared between all platforms:

- PlayerSettings/OtherSettings/Optimization/OptimizeMeshData
  - Strip away any data from the mesh that isn't required by the Material assigned to it
    - if the mesh contains tangent information but the Shader never requires it, then Unity will ignore it during build time
    - However, be aware of this flag if you're switching shaders on some objects at runtime. For example, if a mesh uses a simple Diffuse shader when building the game, Unity will remove tangent vectors since they are not needed. If you'd want to switch to a bumpmapped shader on this mesh at runtime, you will not get proper tangent data since it was removed!
- PlayerSettings/OtherSettings/VertexCompression
  - Reduce the accuracy in vertex position/Normal direction, simplify vertex color information, etc



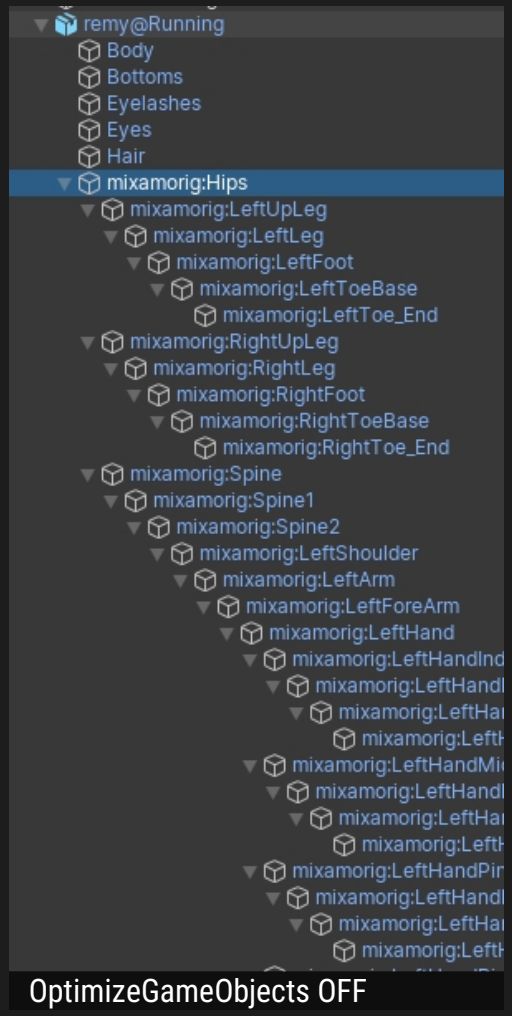


ImportSettings/MeshCompression

Common asset rules

Models

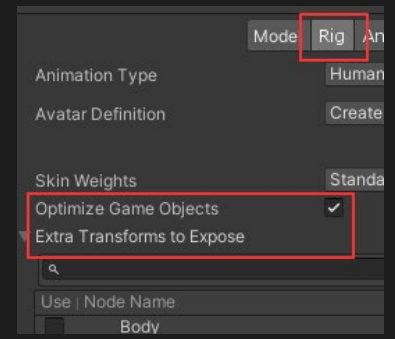
- Disable rigs on non-character models
  - By default, Unity imports a generic rig for non-character models. This causes an added Animator component if Rig/AvatarDefinition is different from NoAvatar. If the model is not animated via the Animation system, this adds unnecessary overhead to the animation system, because all active Animators must be ticked once per frame
- Enable the OptimizeGameObjects option on animated models
  - When OFF Unity creates a large transform hierarchy mirroring the model's bone structure whenever the model is instantiated. This transform hierarchy
    - is expensive to update, especially if other components (such as Particle Systems or Colliders) are attached to it
    - limits Unity's ability to multithread Mesh skinning and bone animation calculations
  - If specific locations on a model's bone structure need to be exposed (such as exposing a model's hands in order to dynamically attach weapon models) then these locations can be specifically allowed in the Extra Transforms list
  - Try it
    - Import [MariaPropWithAnimatorIDLE_00]
    - New scene
    - Drag Maria in Tpose in your hierarchy, assign the animator controller MariaIDLE
    - Add a Cube inside SwordJoint In Humanoid Maria hierarchy
    - Play: you should see IDLE animation, and cube moving according to hierarchy pos rot
    - Change Maria Mesh Rig importer settings, leaving Sword_joint exposed
    - Play: you should see the same prev result, but this time we only have Sword_joint exposed



OptimizeGameObjects OFF

OptimizeGameObjects ON

OptimizeGameObjects ON
+
RightHand in
ExtraTransformToExpose



Common asset rules

Mesh Renderer

- By default, Unity enables Shadow casting/receiving, Light Probe sampling, Reflection Probe sampling. Don't use them if you don't need

Audio

- Platform-appropriate compression settings
  - Import uncompressed audio files into Unity. Unity always recompresses audio when building a project. There is no need to import compressed audio and then recompress it; this only serves to degrade the quality of the final audio clips
- Use audio clips to mono when possible
  - Not all mobile devices have stereo speakers. On a mobile project, forcing the imported audio clips to be mono-channel halves their memory consumption. This setting is also applicable to any audio without stereo effects, such as most UI sound effects
- Reduce audio bitrate
  - While this requires consultation with an audio designer, attempt to minimize the bitrate of audio files in order to further save on memory consumption and built project size

Rendering pipeline

- Poor rendering performance: the device is limited by CPU activity or by GPU activity?
- CPU-bound: is simpler to investigate
- GPU-bound: could be difficult to investigate the Rendering Pipeline

Rendering Pipeline

1. CPU > rendering instructions > Graphics API > hardware driver

--- CPU/GPU Boundary ---

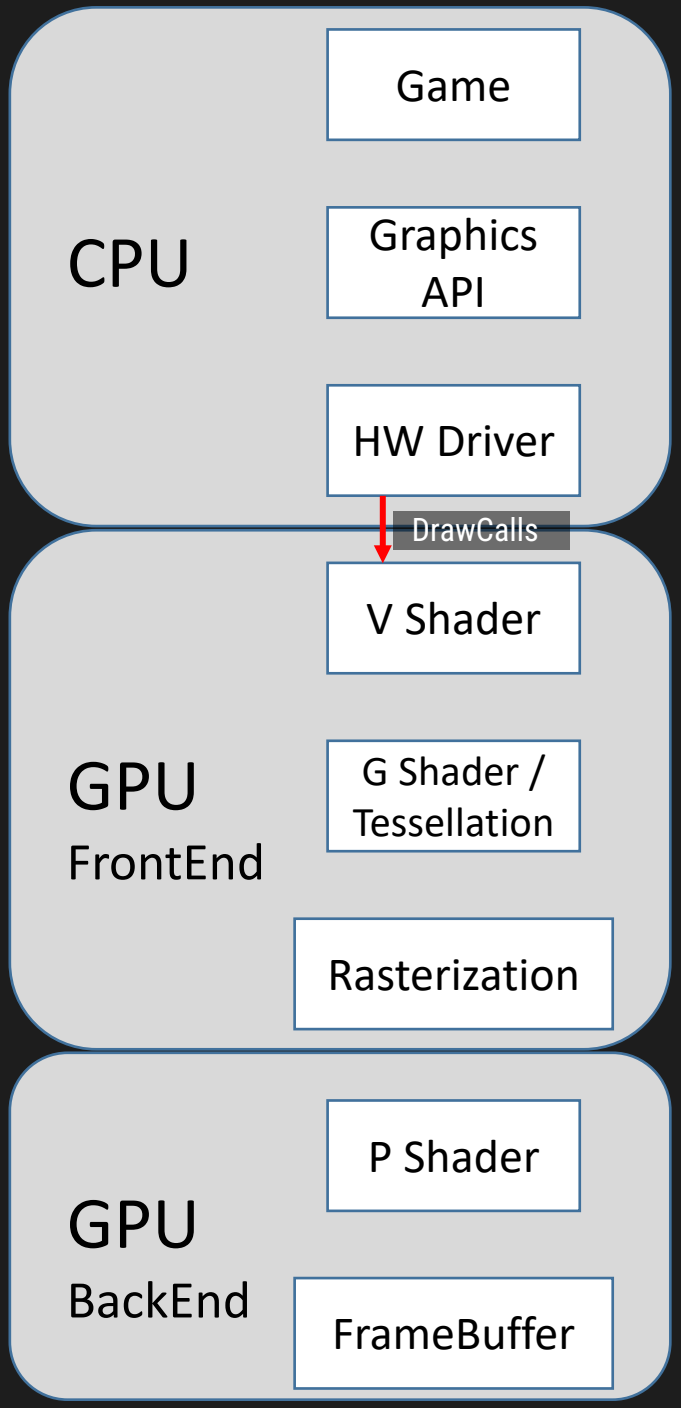
2. > list of rendering instructions in a queue (Command Buffer) > processed one by one until the Command Buffer is empty
    - If the GPU falls behind (GPU Bound), or the CPU spends too much time generating commands (CPU Bound). In both cases, the frame rate will start to drop

GPU Front End Handles vertex data

- mesh data from the CPU + Draw Call > Vertex Shaders > modify vertex data (1-to-1 relation) >
- > Rasterizer, Geometry shader (1-to-many vertices relation) > fragments

GPU Back End Handles fragments

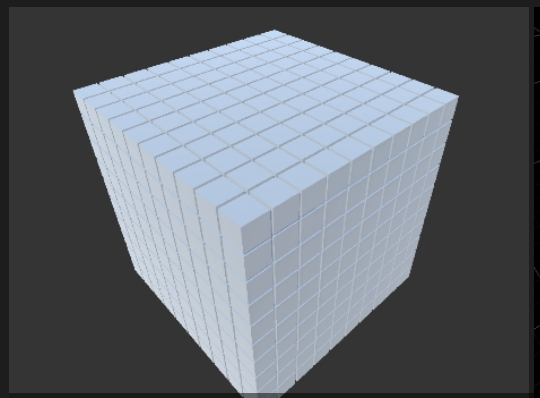
- Pixel (Fragment) Shader > Discard fragment (ZTesting) > Pixels
- Pixels data is drawn onto the FrameBuffer. There are at least 2 frame buffers
    - one contains the data from the frame we just rendered to, and is being presented to the screen; the other is actively being drawn to by the GPU
    - Once the GPU reaches a swap buffers command, the Frame Buffers are flipped around



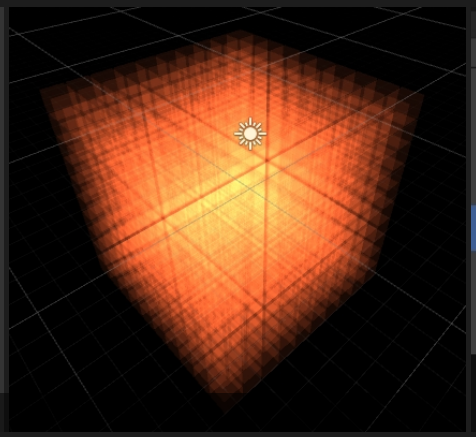
GPU BackEnd Bottlenecks

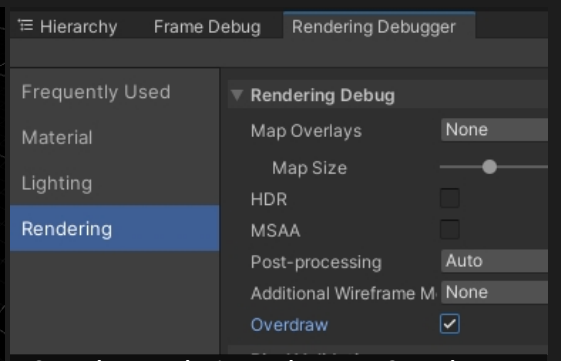
- Fillrate Speed at which the GPU can draw fragments that have survived all of the Fragment Shader tests
  - 30GPixels/second, monitor target 60Hz, resolution 2560x1440.
  - 30.000.000.000 / 60 = 500M pixel per image refresh on the monitor > if there is no overdraw, we can paint the entire screen (which has 2560x1440 = 3.7M pixel) about 135 times
  - There is always Overdraw, which could transform the Fillrate into a bottleneck
    - Objects in the Unity opaque queue are rendered in front-to-back order using a bounding box (AABB center coordinates) and depth testing to minimize overdraw. However, Unity renders objects in the transparent queue in a back-to-front order (based on the center position of their bounding boxes), and does not perform depth testing, making objects in the transparent queue subject to overdraw

- MemoryBandwidth
  - VRAM contains Rendering State info (also textures)
  - Once the texture is being loaded, it is copied onto a super-fast memory called cache. If the texture used doesn't change, texture fetching will be very fast. If texture changes, we have to perform a PULL instruction, which costs a texture-size memory bandwidth amount
  - 96GBs/second, target 60Hz > GPU can PULL 96/60 = 1.6GB data every frame before trigger a BandWidth bottleneck (Maximum texture swapping for every frame)
  - TitanX has 336GB/second = 5.6GB/frame



Overdraw = Fillrate Wasting





OverdraRenderingDebugger Overdraw view is still a bit buggy

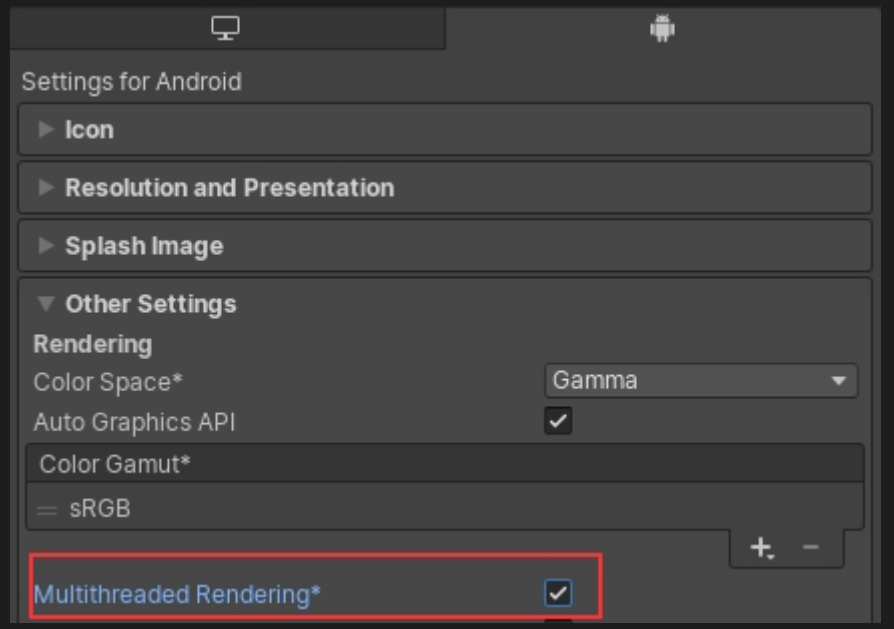
Multithreaded rendering

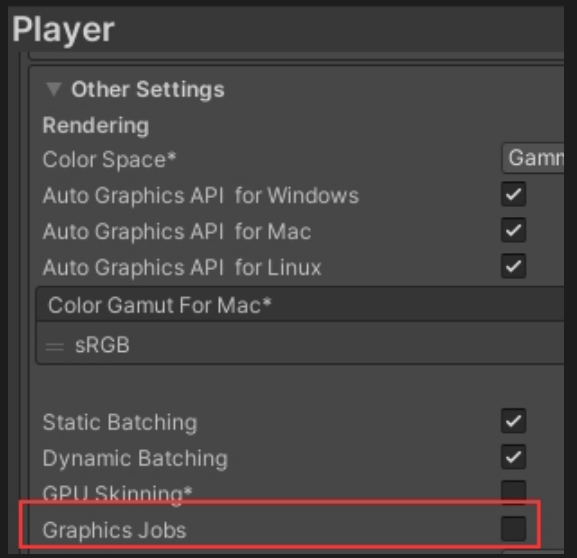
Single thread
- Determine whether the object needs to be rendered
- Generate commands to render the object
- Send the command to the GPU using the relevant Graphics API
- Physics and script code

If we choose a Multithreaded rendering (Enabled by default on Desktop):
- Main thread
  - Physics and script code
- Render thread
  - Pushing commands into the GPU
- Other worker threads
  - Culling, mesh skinning, etc

- Multithreaded is
  - Enabled by default on Desktop
  - Android PlayerSettings/OtherSettings/MultithreadedRendering
  - iOS PlayerSettings/OtherSettings/GraphicsAPI/Metal
  - **Helps CPU-bounded scenarios (GPU is multicore, SIMD)**

  - PlayerSettings/Graphics Jobs Try to take some Main/Rendering thread tasks and distribute them on multiple cores
    - Supported on Win/Mac/PS4/XB1





CPU Bound optimization

- Reducing the number of objects to be rendered > Reduce Batches / SetPass calls
  - Camera FarClipPlane + Fog
  - Camera LayerCullDistance
  - OcclusionCulling

- Reducing the # of times each obj must be rendered > Reduce SetPass calls
  - Forward/Deferred rendering
    - [BIRP] In forward mode, we have to render one object N times, if it is lit by N lights

- Combining the data from objects that must be rendered > Reduce Batches / SetPass calls
  - Static Batching
  - SRP Batching (URP/HDRP)
  - Dynamic Batching
  - GPU Instancing
  - Texture atlasing
  - Combine mesh manually
  - Batching UI

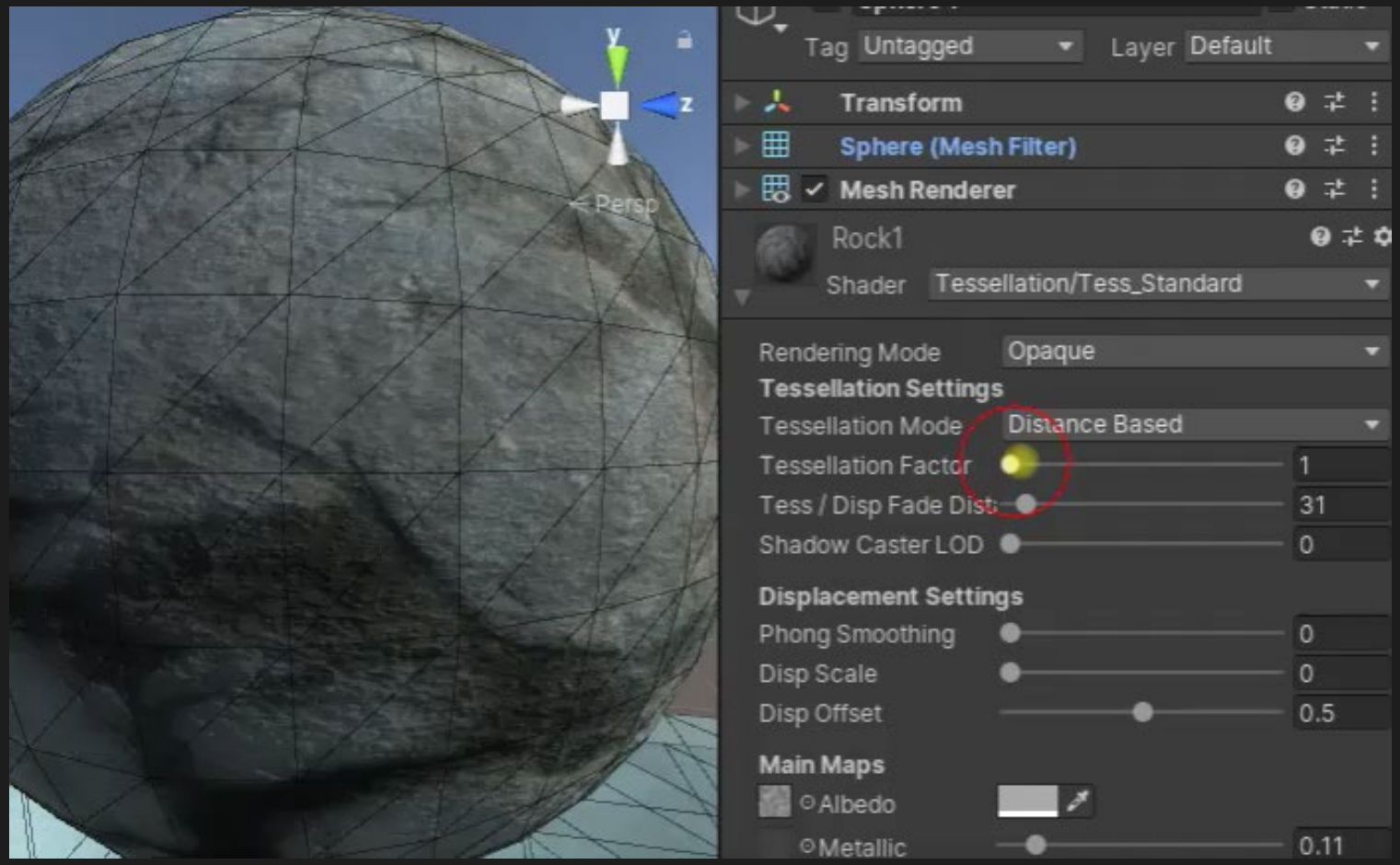
GPU Bound optimization

- FrontEnd Bottleneck
  - Not so common: Vshaders trivial VS Pshaders
  - Complex Geometry shader
  - Improve Tessellation shader
  - Normal Mapping
  - LOD

- BackEnd BottleNeck
  - If it is a Fillrate problem…
    - Reducing screen resolution should improve FPS
      - Simpler shaders (E.g. Mobile shaders)
      - Use fewer Standard Shader options (it is an Uber-shader)
      - Overdraw – Transparent materials / UI / ParticleSystems
      - PProcessing
  - If it is a MemoryBandwidth problem…
    - Reducing texture quality Edit/ProjectSettings/Quality/TextureQuality should improve FPS
      - Texture compression
      - Mipmaps

[BIRP] Tessellation Demo

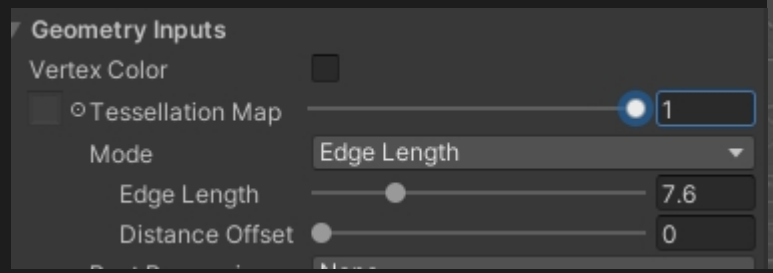
- **Use 3D Pipeline**
- Try to change Tessellation Factor and see Stats/Verts count. The value doesn't change!

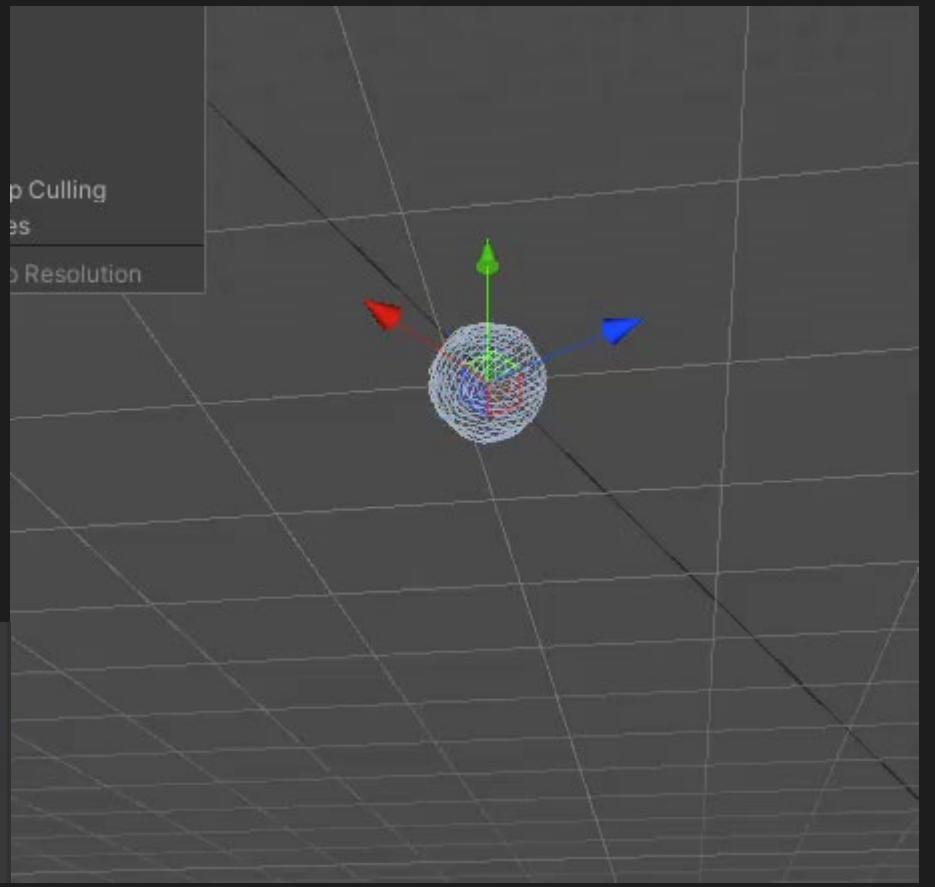


[TessellationDemo_00]

[URP] Tessellation Demo

- Unzip the zip file into /Assets folder
- Delete the original zip file
- Open examples into /Assets/Tessellation & Displacement/Example/Scenes
- Try to change TessellationMap, Mode and EdgeLen values and see Stats/Verts count
- Verts count doesn't change!





[Other/Tessellation&Displacement.zip]

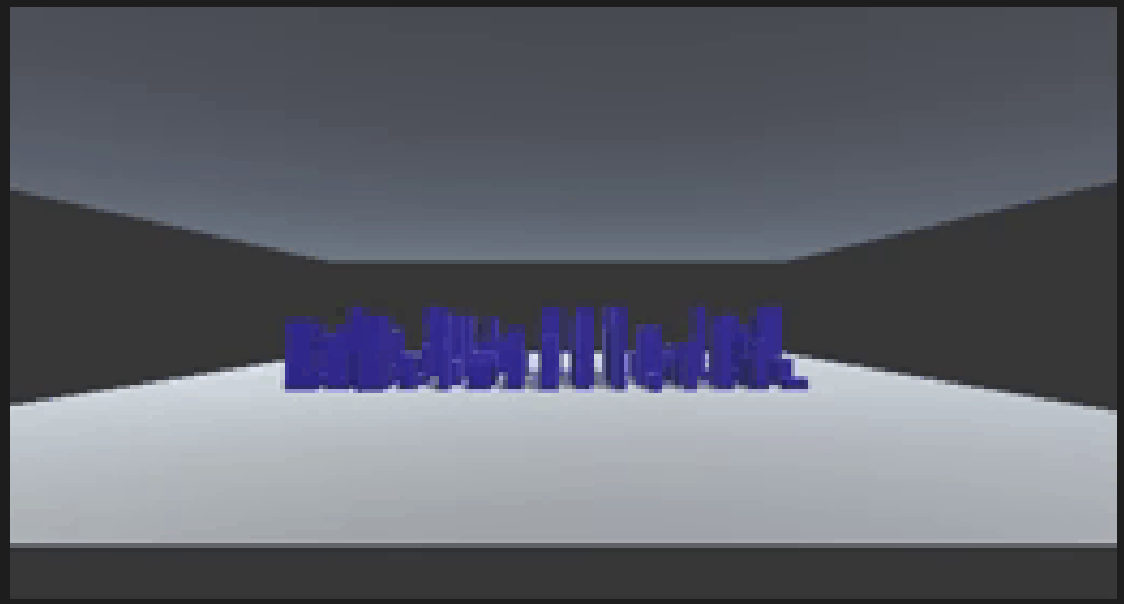
LayerCullDistance

- Normally Camera skips rendering of objects that are further away than farClipPlane
- You can set up some Layers to use smaller culling distances using layerCullDistances
- Very useful to cull small objects early on, if you put them into appropriate layers

- Cull CPU-Bound - Reduce Batches / SetPass calls
- CullDistanceSetter.Reset() sets all layer distance culling to 0 (0 is equivalent to Cam.FarPlane distance)

- Camera.main.layerCullDistances = new float{…32 values…}

Try to
- Spawn 3 city blocks and 1 city cars
- Assign CityNear/CityMid/CityFar/CityCars layers to each group
- Assign different distances in CullDistanceSetter.cs
    - Layer CityNear:50
    - Layer CityMid:60
    - Layer CityFar:70
    - Layer Cars:20 – This means that if distance(camera,car) > 20, car is culled



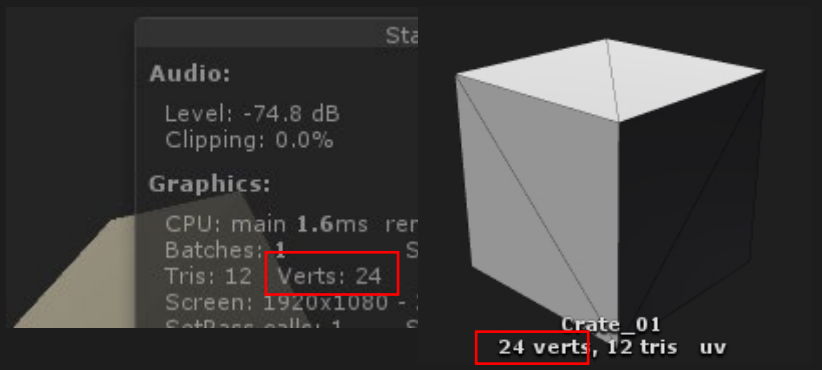
[CullingLayer_01, CullDistanceSetter.cs]

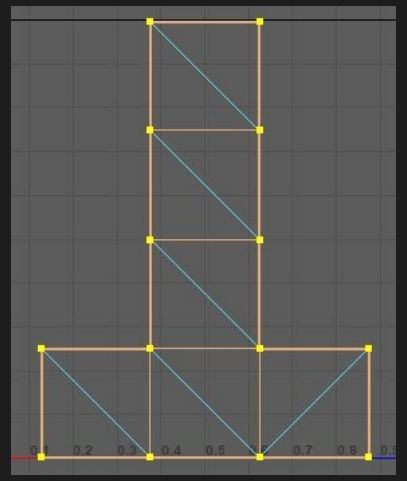
Stats panel

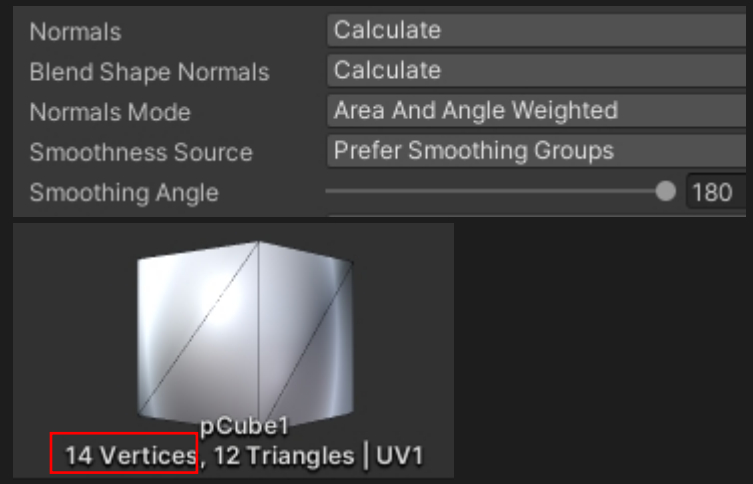
- Not a PRO debug tool

- FPS Editor/Scene view, inspector and other editor-only processing takes additional FPS: final build will be faster
- Batches How you are able to combine SetPass and DrawCalls
- Saved by batching # of batches that was combined
- SetPass # of rendering passes. Each pass requires Unity runtime to bind a new shader
- VisibleSkinnedMesh
- Tris/Verts If there are no elements (no skybox), these values could show the last valid value (not the real value)

- Try to count Tris/Verts of a Cube primitive
- Vertices count in 3D software VS Unity may differ:
  - UV splits – When 2 vertices are on an edge that is an UV seam (Maya UVs in the image is counting 14 vertices)
  - Smoothing groups splits (we need different normals)
  - Geometry vertices (Real 8 vertices of the cubes)
- Why in Unity we have 24 vertices for a cube?
  - Because vertices cannot be shared: they have different normal
  - Even if we set Cube_Tfolded import settings with a SmoothingAngle of 180, we got 14 vertices, because there are UVSplits!

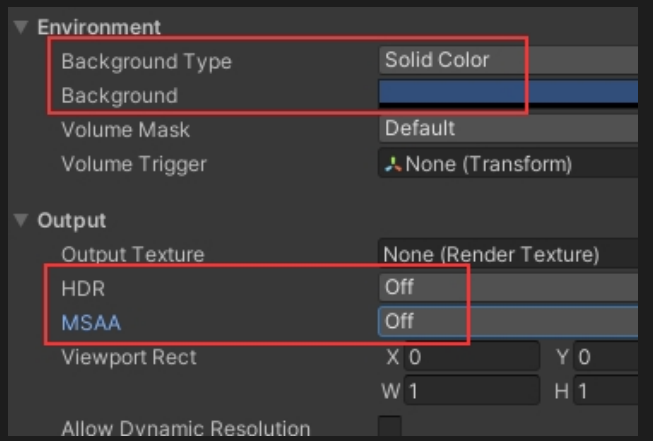


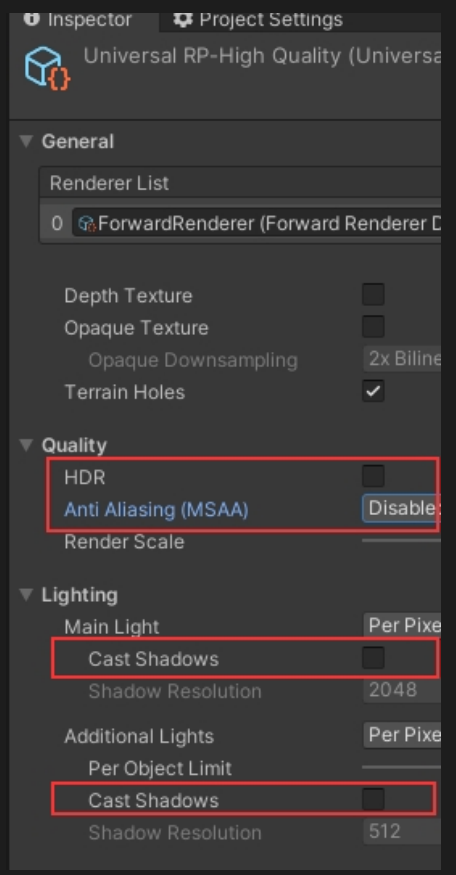


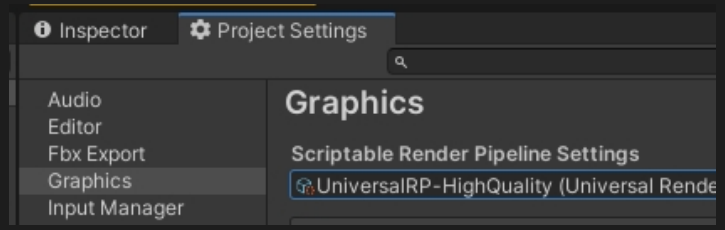


SceneSetup

- In order to examine the number of Batch/SetPassCall, configure your scene in this way:
  - Camera/Environment/BackgroundType SolidColor
  - Camera.HDR OFF
  - Camera.MSAA OFF
  - Select your URPAsset
    - Lighting/Main-AdditionalLight/CastShadows OFF
    - CascadeCount 0
    - Rendering/SRPBatcher OFF







DrawCall

Why minimizing drawCalls is so important?

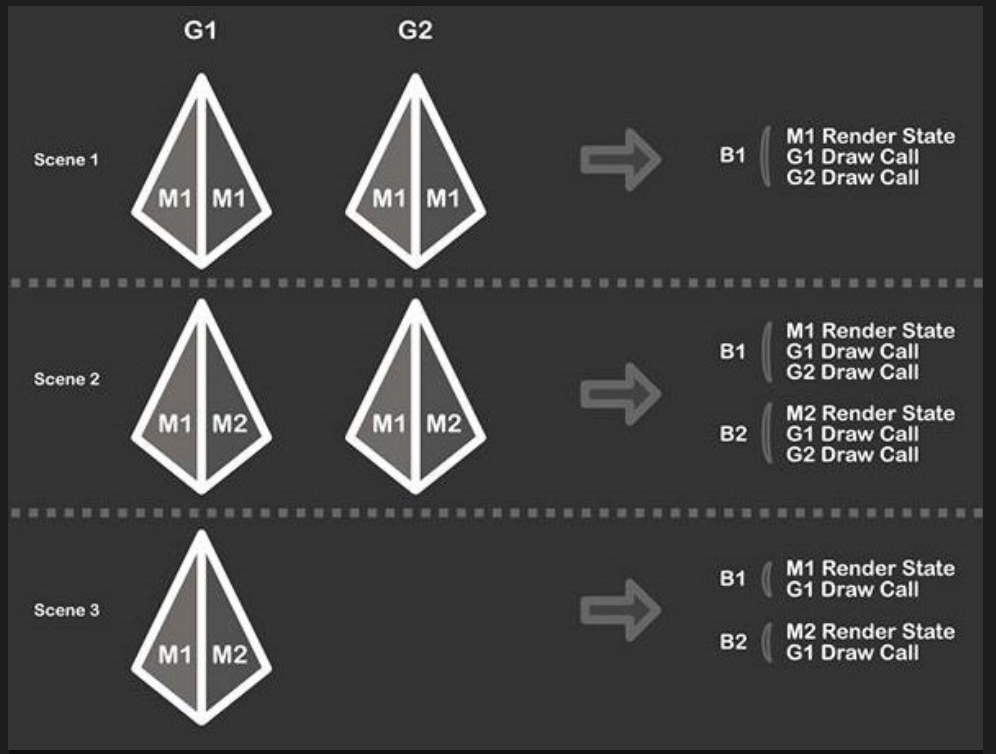
- To draw something, we need to set the RenderState
- If these RenderState options are unchanged, the GPU maintains the same RenderState for all incoming objects and render them in a similar fashion
- RenderState is equivalent to set rendering parameters. Doing this manually is tedious, that's why game developers introduced the concept of Materials, that are containers around Shaders

- A single Material can only support a single Shader. The use of multiple Shaders on the same mesh requires separate Materials to be assigned to different parts of the same mesh, and the use of SubMeshes
- A single shader can be used by multiple Materials

- Every material you add to your scene increases the complexity of your rendering pipeline. Each material adds at least one SetPass (this sets rendering parameters)
- We use batching to group the rendering of similar objects into the same draw call. In batching, it all comes down to using the same material across different objects

- E.g. Think about the texture to use as a global variable on the GPU
  - Change a global variable in a SIMD architecture is not simple
- If next obj has the same material, GPU can avoid switching the RenderState
- Reducing materials leds to
  - Less CPU time to send RenderState switch instructions to the GPU
  - GPU won't need to stop and re-synch state changes as often

- Try it
  - Create a cube in Pbuilder and assign a different material to a face => 2 materials, 2 SetPassCalls, 2 Batches!



NB: DynamicBatching is active (see later)
Scene1 – 1 Batch, 2 DrawCalls
Scene2 – 2 Batches, 4 DrawCalls
Scene3 – 2 Batches, 2 DrawCall

Batches, SetPass

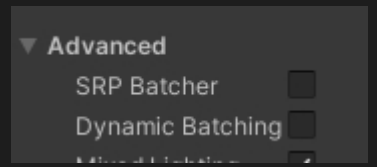
Setup
- URPSettings/Advanced/SRPBatcher OFF
- URPSettings/Advanced/DynamicBatching OFF

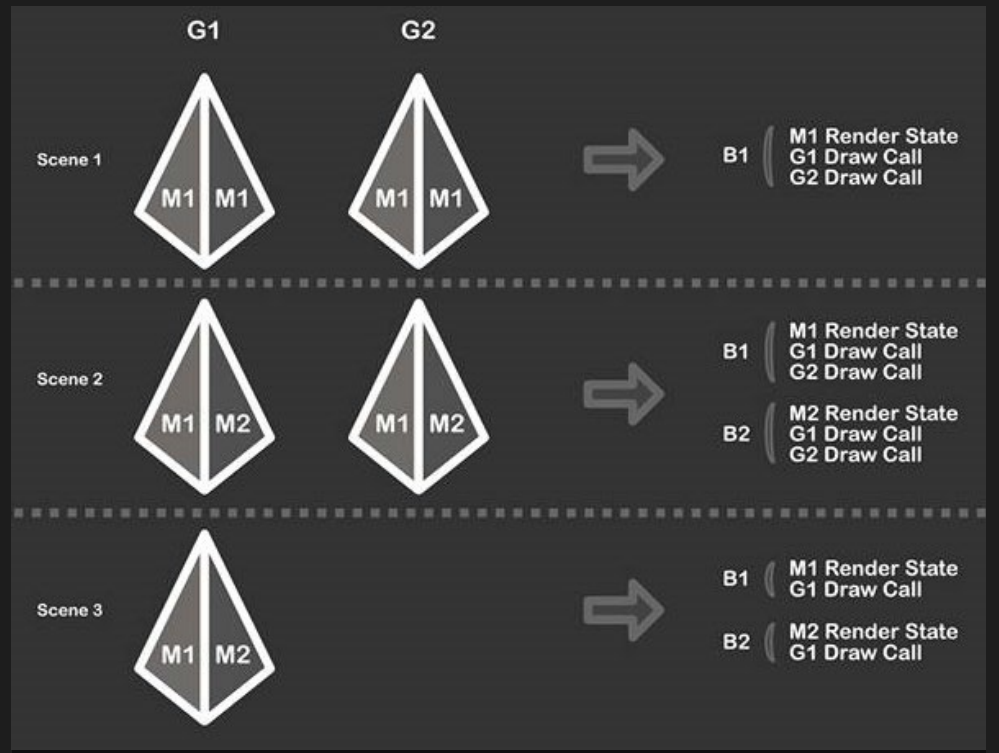
Drawcalls
- Draw Calls are issued every time Unity renders a mesh, but it's not always one draw call per mesh. Materials split meshes into sub-meshes which are drawn separately. So, for a single Object, if it has sub-meshes and different materials, we'll perform more than a drawcall
- When CPU tells the GPU to render a mesh, that mesh is on the VRAM
  - Mesh data can be uploaded on the VRAM every frame (DynamicBatching) or at startup (static batching)

SetPass
- A SetPass call is issued every time Unity needs to change the shader pass that is being used. It's important to note that this is not the same as the number of unique shaders in view as most shaders have a few passes for handling different parts of the rendering

Batch
- Combination of SetPass and DrawCalls

Why to keep Drawcalls count low, even if we got good performances?
- Porting to other platforms. The more optimized your game is, the less effort you must put in the porting process
- Time saving. Soon or later, when your project will grow, you'll probably face the problem of optimization
- Improves the efficiency. Less CPU use & better performances = less battery drain



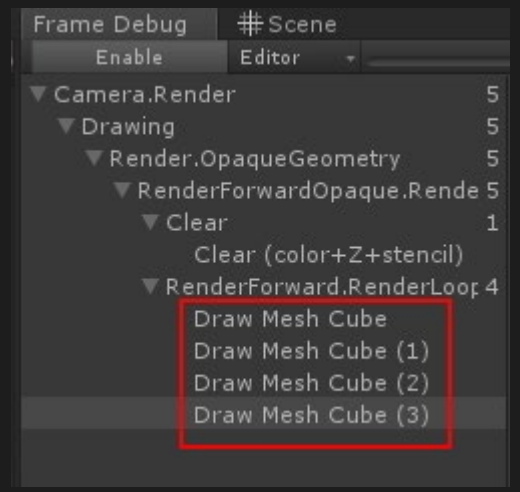


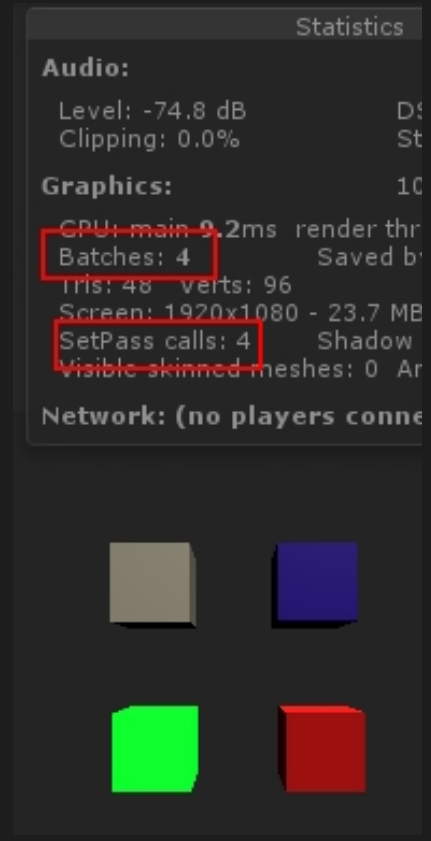
NB: DynamicBatching is active (see later)
Scene1 – 1 Batch, 2 DrawCalls
Scene2 – 2 Batches, 4 DrawCalls
Scene3 – 2 Batches, 2 DrawCall

FrameDebugger

- Open a new scene
- Add 4 cubes
- Open FrameDebugger windows and see what happens if you add materials

- One single material
  - 4 DrawCall (Batches) (Dynamic Batching is OFF, see later)
  - 1 SetPass Call – 1 RenderState is needed
- 4 Materials
  - 4 DrawCall (Batches)
  - 4 SetPass Call – 4 RenderStates are needed





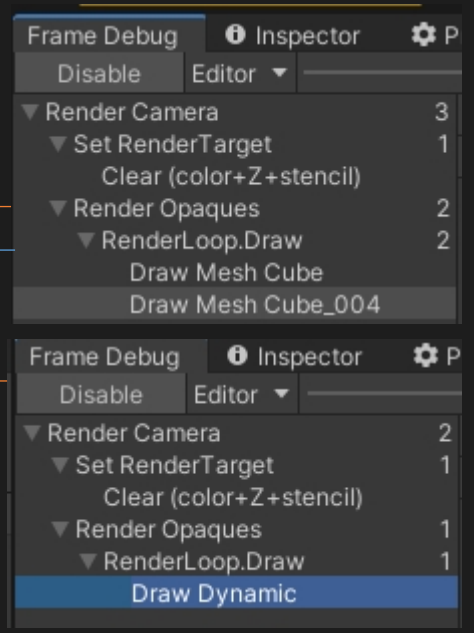
DynamicBatching

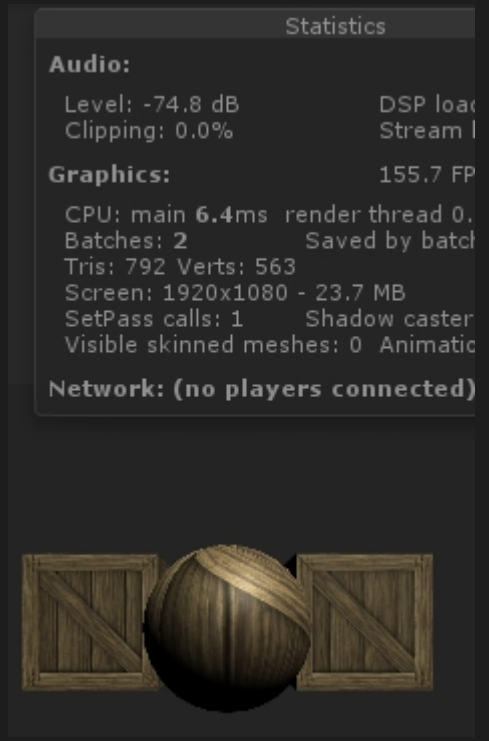
Setup
- SRPSettings/Advanced/DynamicBatching ON

Try
- 1 cube 1 material
- 2 cubes 1 material
- 2 cubes 1 sphere 1 material
- 2 cubes, 1 with negative scale
- 2 spheres, 1 material: can't batch, because 1 sphere has more than 300 vertices

DynamicBatching OFF

DynamicBatching ON





Limits
- Meshes must share the same material
- Max 300 vertices per mesh. The real vertices limitation is linked to the total # of vertex attributes used by the Shader: it must be < 900
  - Hence, to have a 300 vertices max limit, we have max 3 attributes per-vertex: Eg. position, normal, single set of UV
    - E.g. Complex Shader - 5 attributes per-vertex (Vertex Position, Normal, UV0, UV1 and Tangent) = no more than 180 vertices
- The material have to use one single-pass shader
- Highly unpredictable. You cannot really tell how your objects will batch: the result often changes from frame to frame. Try to move objects in your scene and see in Unity Frame Debugger how results dramatically change across frames

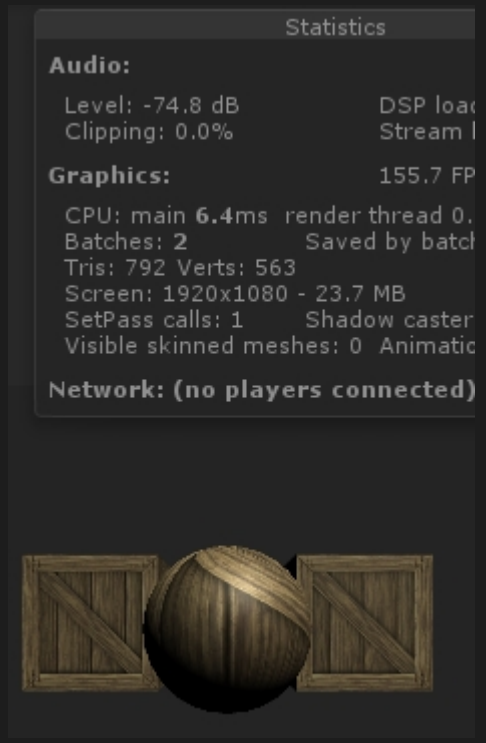
DynamicBatching

How it works
- Transforms all GObjs vertices into world space on the CPU
  - It is only an advantage if that work is smaller than doing a draw call. The resource requirements of a draw call depends on many factors, primarily the graphics API used. For example, on consoles or modern APIs like Apple Metal, the draw call overhead is generally much lower, and often dynamic batching cannot be an advantage at all
- It is possible to Dynamic batch objects that use different meshes together

Samples
- Large forest filled with rocks, trees, and bushes
- Building, factory, space station with many simple, common elements (corridor pieces, pipes, etc)
- Scene with objects with simple geometry

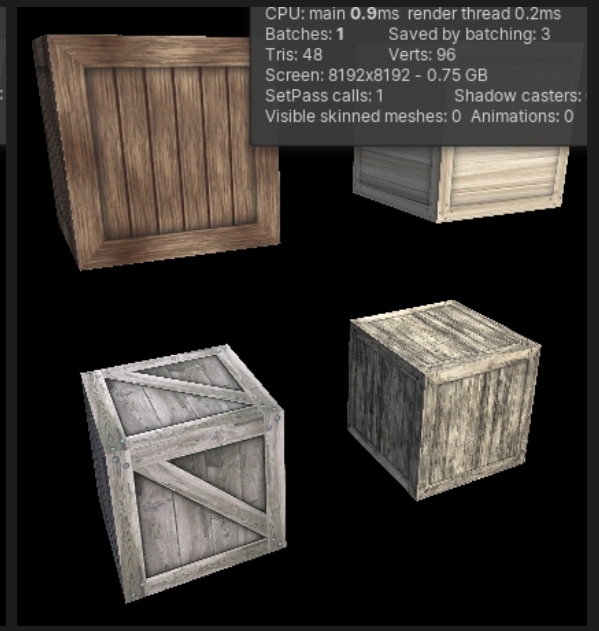
Hint
- If two objs that use the same shader aren't batched because of different textures > Use the same material with a Texture atlas (see next slide)

Memory cost
- Two dynamically batched cubes (1 mesh composed by 2 cubes batched + 1 mesh of the original cube) will use more memory that two statically batched cubes (1 mesh composed by 2 cubes batched)



Use texture atlas

- Try to create 4 cubes with materials Crate0,1,2,3
  - 4 Batches, 4 SetPass calls
- Instead of using Unity cube primitive, use CubeUV_TL,TR,BL,BR and the same material WoodBoxes_Atlas: they have different UV cords
  - 1 Batch, 1 SetPass call



[AtlasTexturesTest_00]

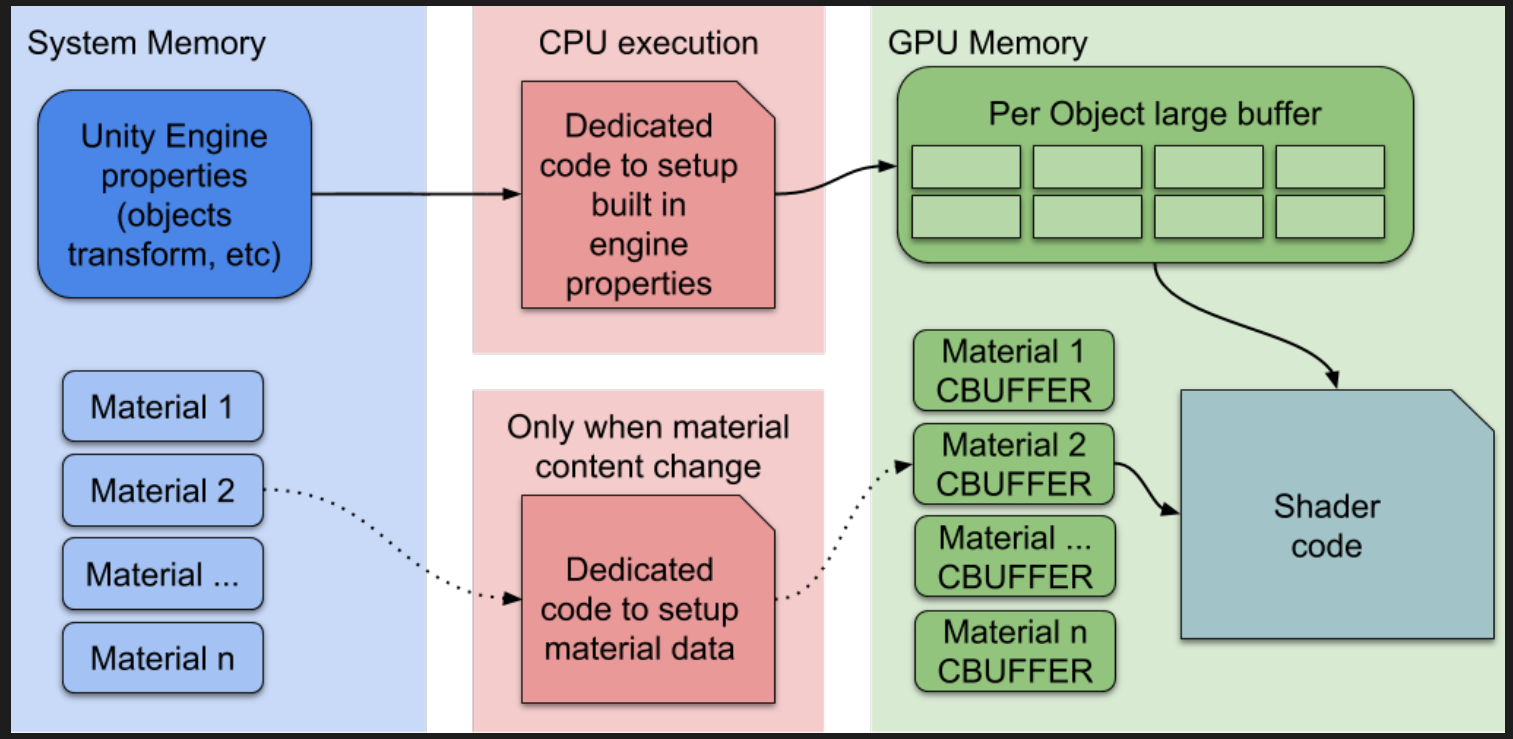
SRP Batcher

- URPAsset/Advanced/SRPBatcher ON

- You can use as many different Materials with the same Shader variant
- During the inner render loop, when Unity detects a new Material, the CPU collects all properties and sets up different constant buffers in GPU memory

- The SRP Batcher is a low-level render loop that makes Material data persist in GPU memory. If the Material content does not change, the SRP Batcher does not need to set up and upload the buffer to the GPU. Instead, the SRP Batcher uses a dedicated code path to quickly update the Unity Engine properties in a large GPU buffer
- This speeds up rendering because:
  All Material content now persists in GPU memory.
  Dedicated code manages a large per-object
  GPU CBUFFER for all per-object properties

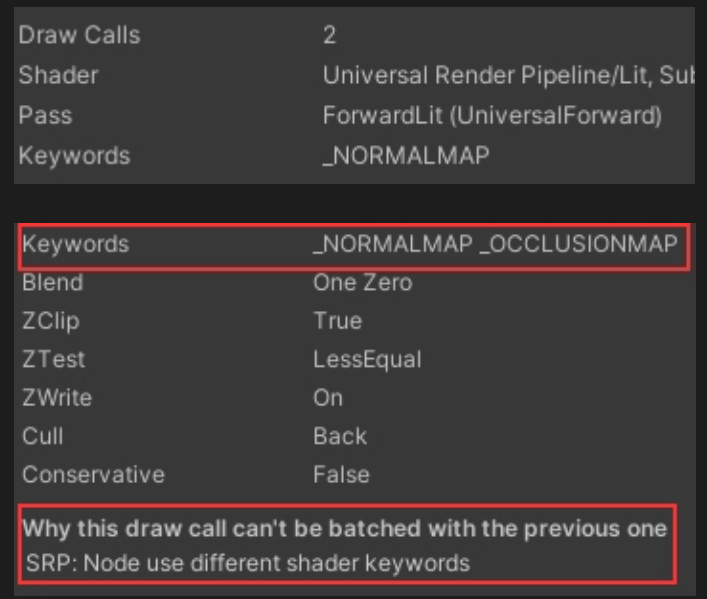
Limitations
- The rendered object cannot be a particle
- The Shader must be compatible with the SRP Batcher. All Lit and Unlit Shaders in HDRP and URP fit this requirement (except for the Particles versions of these Shaders)



SRP Batcher

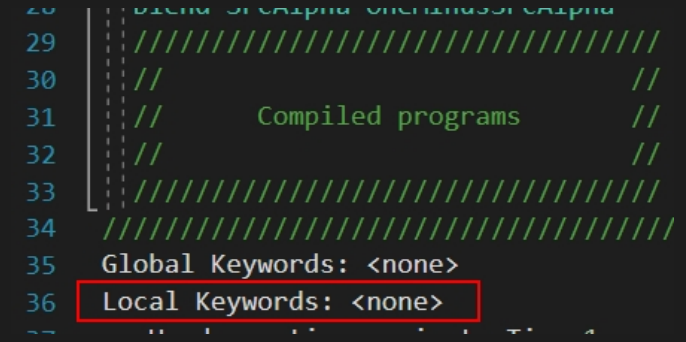
- Frame Debug shows you
  - Real number of batches (it works differently than the other batching technique, Statistics panel doesn't show this batch optimization)
  - # of draw calls
  - which keywords were attached to the Shader
  - the reason why that specific draw call wasn't batched with the previous one

- Try it
  - Create a new URP/Lit material A using BaseColor texture and NormalMap
  - Create a new URP/Lit material B using BaseColor texture and NormalMap and OcclusionMapping
  - Create 2 Spheres with material A: SRPBatch shows that you are using shader variant compiled with the keyword _NORMALMAP, and you are collecting 2 draw calls in one SRP Batch
  - Create Another Spheres with material B: SRPBatch shows that you are using shader variant compiled with the keywords _NORMALMAP and _OCCLUSIONMAP => you are using a different Shader variant from the previous => can't batch in the previous SRP Batch



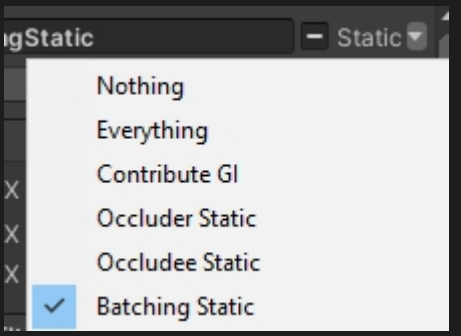
SRP Batcher

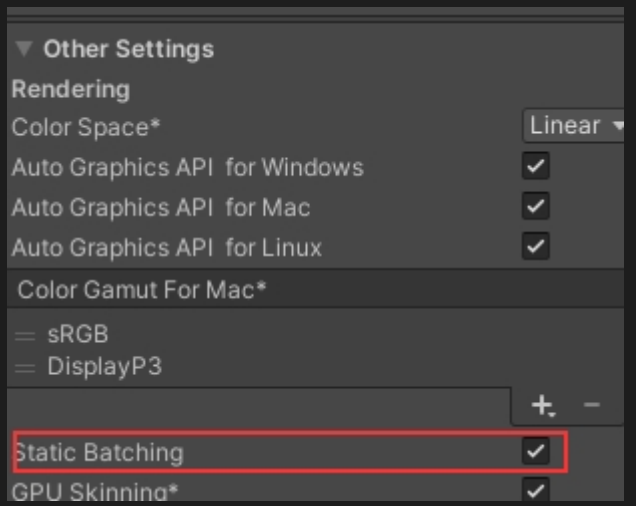
- Click on Compile&ShowCode of your shader with variants and look for “Local Keywords”: these are “mini shaders”, or “shader variants” compared with the “Uber shader”
- It could be better to use always the same variant: using normal map only for half objects inside the scene could not save shader time rendering on the other half objects of the scene, because it could introduce a batch breaking cause. Using the same shader variant – i.e. use always a normal map even if it is a flat one, could be more efficient!



Static Batching

Setup
- Objs must have Static flag BatchingStatic
- URP/Advanced/DynamicBatching OFF
- URP/Advanced/SRPBatching OFF
- ProjSettings/Player/StaticBatching ON (we must enable StaticBatching BEFORE entering PlayMode)
    - This setting is under Player setting because Static Batching is something that is performed DURING the building process





Try
- Enable StaticBatch_Offline
- Enter PlayMode
- Even if we have 5 spheres… we have only ONE batch!
- Enable StaticBatch_RunTime
- Spawn Spheres and flag RuntimeStaticBatcher.PerformStaticBatching

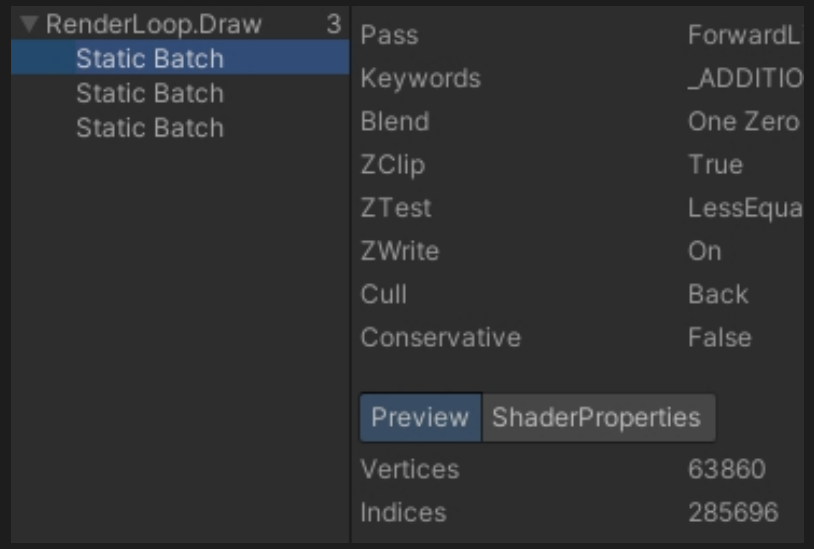
How it works
- All visible meshes data is copied into a single, large mesh data buffer, and passed it to the GPU, ignoring the original mesh
- Performed during enter in Play-mode / during build: final build will have staticBatching meshes grouped in a single mesh, not the single staticBatching meshes
- Can be used on meshes of big sizes, which Dynamic Batching cannot provide
- Not designed for a lot of the same mesh (use GPU instancing for that)

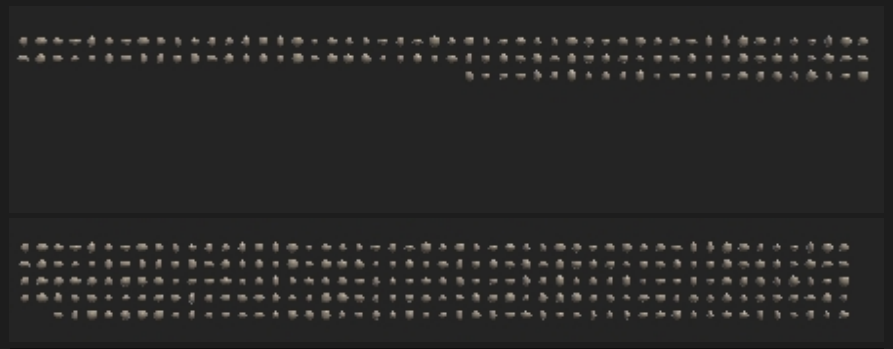
[StaticBatching_End_02]

Static Batching

Limits

- The vertices upper limit that can be combined in a static batch varies per Graphics API and platform (around 32k-64k vertices)
- Meshes must share the same material
- Objects marked Batching Static introduced in the Scene at runtime will not be automatically included in Static Batching (would cause runtime overhead). To force it, use
  - StaticBatchingUtility.Combine(this.gameObject);
    - Requires that the import settings of all sub-meshes to batch must allow CPU read/write





Runtime Batching: there are 250 spheres. Each sphere has 515 vertices
If static batch limit is 64K, then we have 64K / 515 =>
- 2 batches of 124 spheres
- 1 batch of 2 spheres

Static Batching

Memory cost

- If N batched meshes are unique > this costs no additional memory
- If N batched meshes are the same (like the example before [A]) > this costs N times more memory
    - E.g marking trees as static in a dense forest level can have serious memory impact

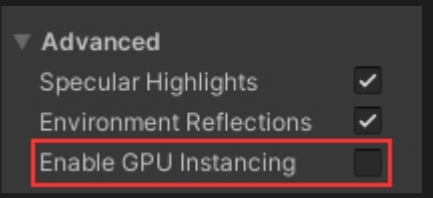
NB

- Draw Call savings are not immediately visible from the Stats window until runtime. When exported, the executable will contain only the static mesh, not the source meshes (no need to recalculate static batching during scene load)
    - start working on Static Batching optimization early in the process of building a new Scene
- At Start, Unity MeshMemory already contains 3D primitives (Cube, Sphere, Capsule, etc), hence if you allocate a cube or a sphere, you won't see any difference in MeshMemory allocation

[A] During StaticBatchingUtility.Combine() Mesh Memory grows and CPU Rendering has a peak

GPU Instancing

Setup
- URP/Advanced/DynamicBatching OFF
- URP/Advanced/SRPBatching OFF
- ProjSettings/Player/StaticBatching OFF



Try it
- NxNxN spheres
- Activate EnableGPUInstancing flag on the material

Memory cost
- Profiler/Memory/MeshMemoryCost is unchanged

Limits
- Objs must have same Materials and same Mesh

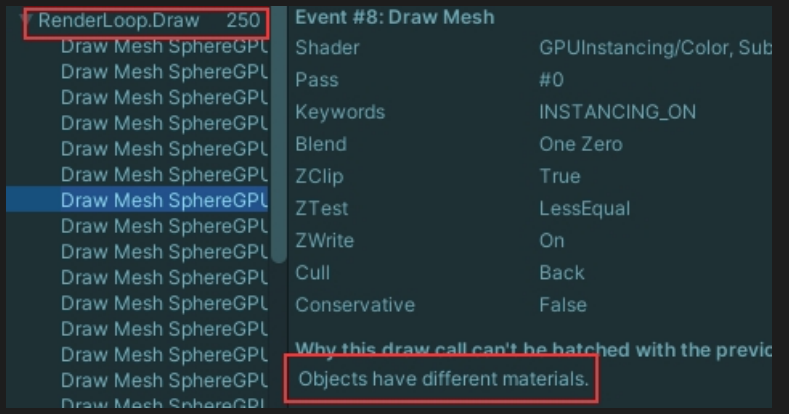
How it works
- The GPU is told to render the same mesh multiple times in one go
- Doesn't combine all the meshes into one, but pass only one mesh description to the GPU
- Mesh variations are introduced via Shader code: the instanced property is passed to the shader via an array, and objects have different instanceIDs: Object with i-th instanceID will be rendered using InstancedProperty[i] value

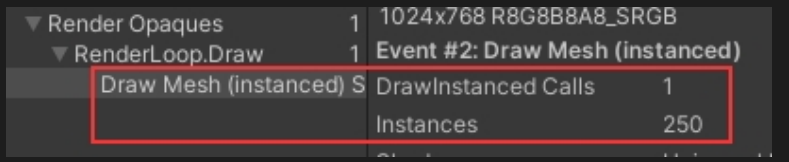
[GPUInstancing_02]

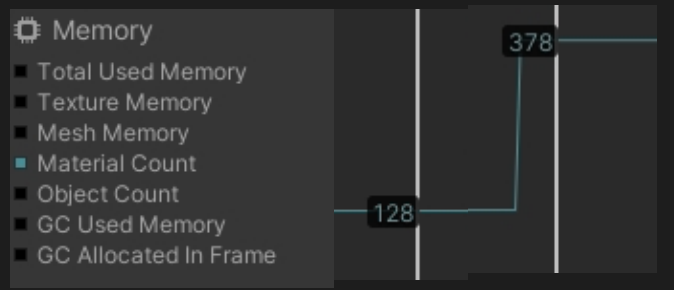
GPU Instancing

Try it

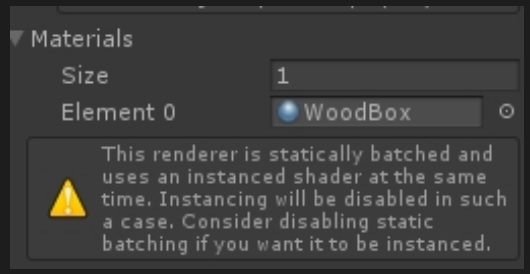
- Enable GPUInstancing_Transform
- Spheres use URP/Lit shader material, with EnableGPUInstancing: even if they have different Transform values, they are all rendered in one single DrawCall! Different Transform matrices are sent to the shader linked to different instanceIDs
- Enable GPUInstancing_Color
- They use GPUInstancing/Color shader material. Because it supports GPUInstancing, it allows to render the same mesh with different Transforms in one single drawcall, like before. We also have an additional Instance property in this shader, that let us to update the "_Color" property per instance
    - Press Play: 1 Draw call / 250 instances
    - Click on ChangeMaterialCol_PropBlock.ChangeCol: 250 DrawCall, because each sphere now has a different Material instance
        - in order to change the color in the shader, Unity needs to tell the GPU that this object is going to be rendered differently, and the only way it can do so is by changing the material instance. Because of this, Unity creates a copy of a material when you access renderer.material for the first time. This means that there are potentially a lot of material copies out there, all eating memory
        - Close and open again Unity. Check Material numbers in the profiler: from N to N+250!
    - Press Play: 1 Draw call / 250 instances
    - Click on ChangeMaterialCol_PropBlock.ChangeColUsingPropertyBlock: 1 DrawCall, because each sphere now takes its color using its instanceID!
        - Check Material numbers in the profiler: from low number to about 240 materials!







GPU Instancing



- By default, Unity only batches instances of GameObjects with different Transforms in each instanced draw call (we can give different instances different translation, rotations, scales)
  - To add more variance to your instanced GameObjects, modify your Shader to add per-instance properties such as Material color

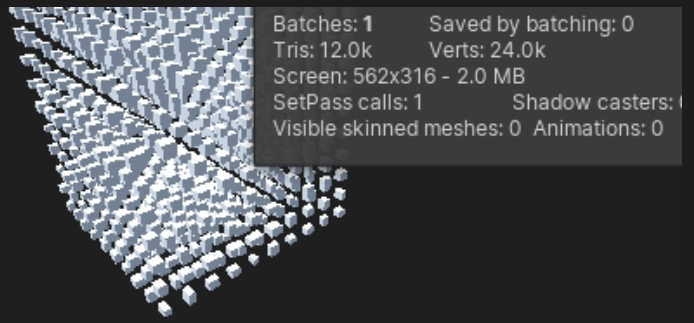
NB
- Static batching has priority over GPU instancing (A Warning message will appear)
- GPUInstancing has priority over Dynamic batching
  - Using GPUInstancing will log DrawMesh (Instance) in Frame Debugger
  - Using DynamicBatching, FrameDebugger will log DrawDynamic
- If SRP batcher is ON, GPUInstancing will be ignored

- Let's say that you have to draw a forest. You provide two streams of data
  - common data that will be rendered multiple timesthe mesh and textures
  - the list of instances and their parameters that will be used to vary that first chunk of data each time it's drawn
  - With a single draw call, an entire forest grows. The fact that this API is implemented directly by the graphics card means that Flyweight pattern is one design pattern that has actual hardware support

CombineMeshes

Setup
- PlayerSettings/OtherSettings/DynamicBatching OFF

Let's say the aim of the game is to collect items, which will end up in a cart. The cart is a dynamic object that can move, so we cannot use Static batching. But these elements become kind of static within your cart
We can use APIs to create these batches manually with
- StaticBatchingUtility.Combine (see StaticBatch slide), but in this case I won't be able to move those objects!
- Mesh.CombineMeshes() takes a list of meshes and creates a combined mesh
  - See CombineMeshes_AtStartup attached script CombineMeshes.cs
  - We need MeshFilter and MeshRenderer Components in the root that will merge all the submeshes
  - CombineInstance is a Struct used to describe meshes to be combined using Mesh.CombineMeshes()
  - NB: CombineMeshes() uses the transform in CombineInstance structs. In our case we are using localToWorldMatrix, hence the MeshFilter root transform must be the identity, otherwise the combined mesh will be changed by the root Transform (see CombineMeshes.cs comments //1 and //2)



Try it
- spawn 10x10x10 cubes under StandardMeshes
- spawn 10x10x10 cubes under CombineMeshes_1Material

[CombineMeshes_00, CombineMeshes_1Material.cs]

CombineMeshes

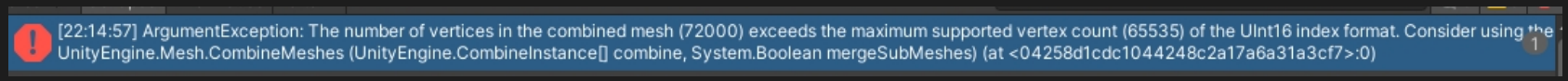
What if you want to combine Meshes that uses N different Materials?

- Merge them in one root Mesh that has
  - N Materials
  - N Submeshes
- See CombineMeshes_NMaterials attached script CombineMeshes_NMaterials.cs to know how

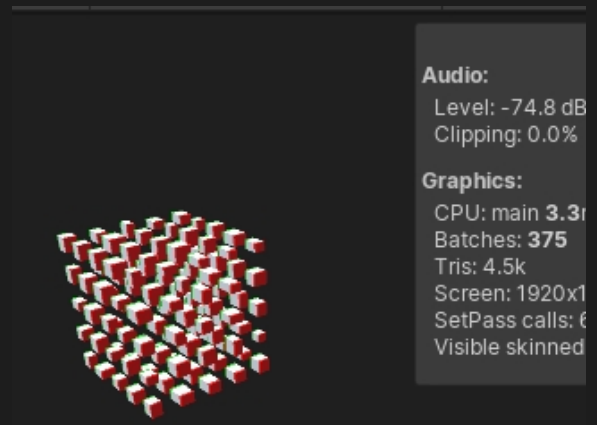
Try it

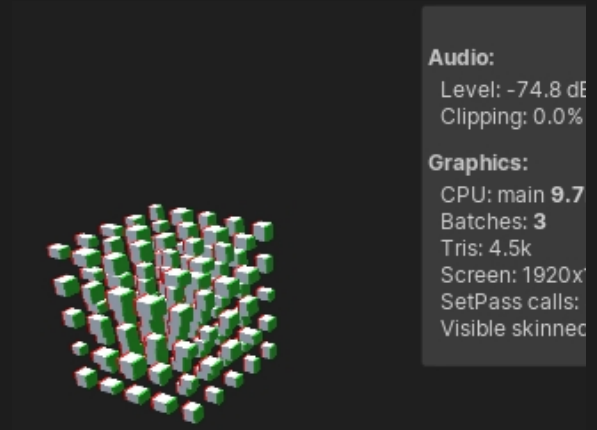
- If you disable CombineMeshes_NMaterials.cs and enable DynamicBatching, the # of Batches vary depending on the Transform rotation of CombineMeshes_Nmaterials
  - Unpredictable results!

- NB
  - Be careful not to accidentally instance Materials (line 31). Accessing Renderer.material automatically creates an instance and opts that object out of batching (during play mode, you'll see an empty result, and each disabled cube will have an instance of its original material [A]). Use Renderer.sharedMaterial whenever possible
  - We have to use indexFormat = UnityEngine.Rendering.IndexFormat.UInt32 to avoid vertices number overflow



[CombineMeshes_00, CombineMeshes_NMaterials.cs]





[A]

Review

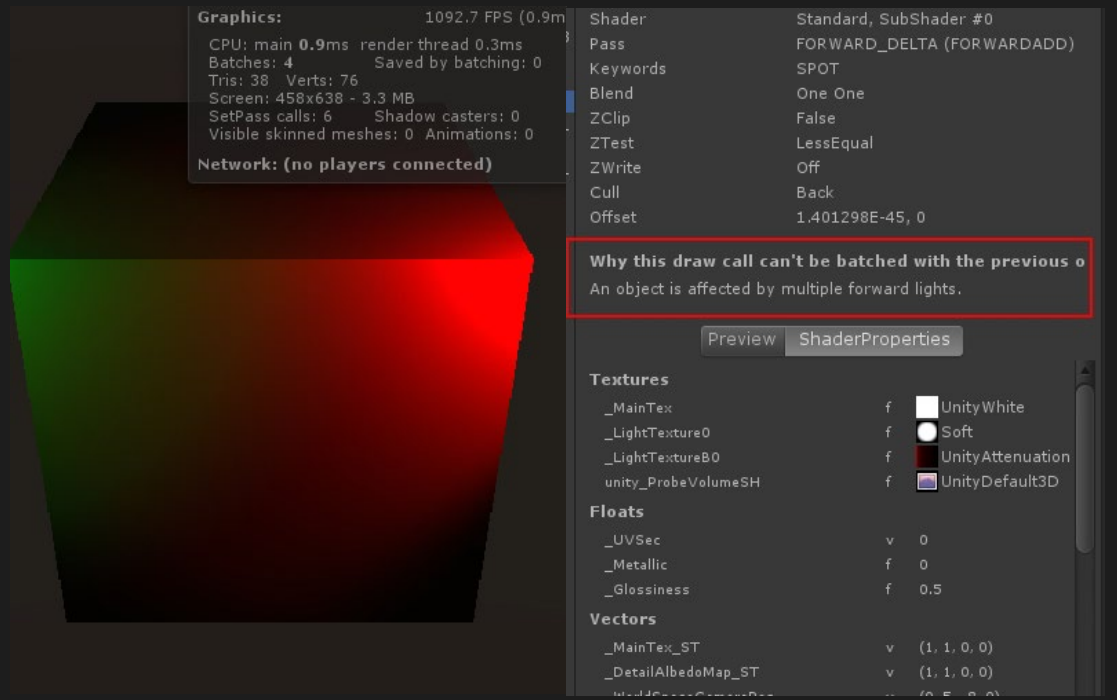
	Static Batching	Dynamic Batching	GPU Instancing	SRP Batcher
Built-in RP support	Yes	Yes	Yes	No
Universal RP (URP) support	Yes	Yes	Yes (Need to disable SRP batcher	Yes
Works on static objects? (the "**Batching Static**" flag on MeshRenderer component)	Yes	No 1	No 2 (if Static batching is used, instancing will be turned off)	Yes
Works on **dynamic** objects?	No	Yes	Yes	Yes
Objects can use **different meshes**?	Yes (Different meshes are combined into a big mesh)	Yes	No	Yes

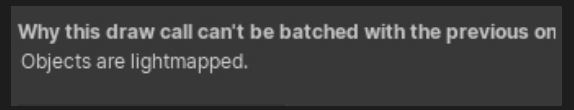
	Static Batching	Dynamic Batching	GPU Instancing	SRP Batcher
Objects can use **different materials**?	No	No	No	Yes (Be-aware of shader variant)
Objects can use **different shaders**?	No	No	No	No
Objects can have **different property values** set by script? (Per-object / Per-instance properties e.g. color)	No	No	Yes (use MaterialPropertyBlock)	Yes (setup different materials)
Works on **SkinnedMeshRenderer**	No	No	No	Yes
Typical use case	Useful for larger unique meshes that will share the same material. Reduce draw calls (increase CPU performance).	For very low-poly dynamic objects that share the same material, e.g. a quad mesh. Reduce draw calls. Useful for platforms and graphics APIs that do not support GPU instancing, e.g. very low-end devices	Useful for drawing objects that appear repeatedly (same mesh). Reduce draw calls (increase CPU performance).	Useful if you have many different Materials that use the same Shader. Speeds up CPU rendering without affecting GPU performance.

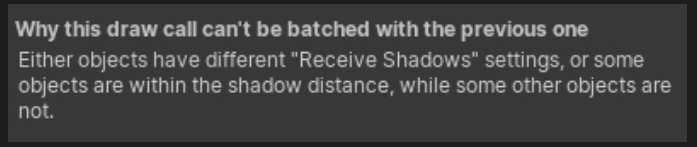
1. If objs have BatchingStatic flag & StaticBatching == DynamicBatching == ON, StaticBatching is used; but if StaticBatching == OFF, then DynamicBatching is used
2. If objs have BatchingStatic flag & StaticBatching == OFF GPUInstancing is not used (Frame debugger tells you that a DrawMesh(instanced) is performed, but every obj will result at the same place)

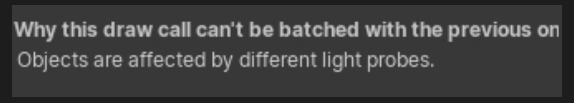
[BIRP] Batch-Breaking Causes

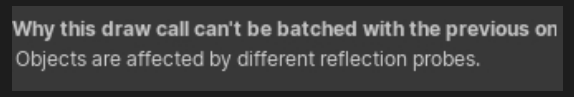
- Open Scenes inside /Scenes/BatchBreakingCause folder
- MultipleForwardLights
  - Try to move one Spotlight away from the cube: there will be one less drawCall
- ObjectsOnDifferentLightingLayers
- ObjectsSplitByShadowDistance
  - Set ProjSettings/QualityShadowDistance to 150
  - Increase ProjSettings/QualityShadowDistance to restore DynamicBatching
- MixedSidedModeShadowCasters
  - Disable DirectionalLight and enable OppositeDirLight to show that TwoSidedQuad casts shadows also on the opposite side
  - The doubled batch is inside Shadows.RenderJob
- LightprobeAffectedObjects
- LightMappedObjects
  - Even if DirectionLight is disabled: there is the baked light
- DifferentShadowReceivingSettings

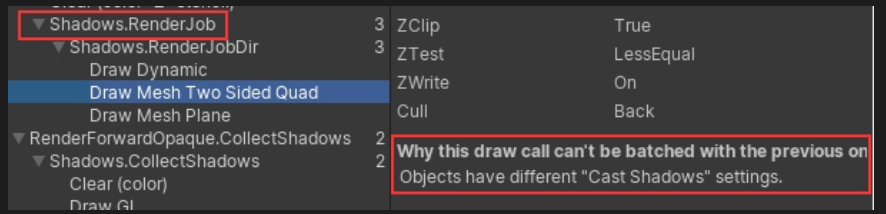












Level Of Detail (LOD)

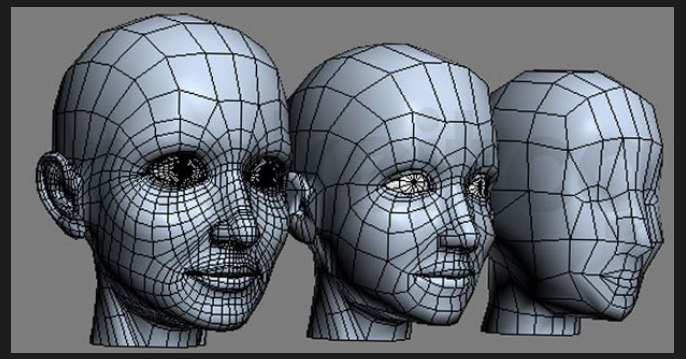
- Reduce (vertex) shading cost
- Reduce drawcalls (if LODs are correctly authored)
  - Simpler materials for LODs
- Reduce geometry aliasing (lot of informations - vertices - in a small place)
- Use different meshes depending on the view size of the object
  - May cost a large amount of development time
  - Third-party tools for automated LOD mesh generation

Try it

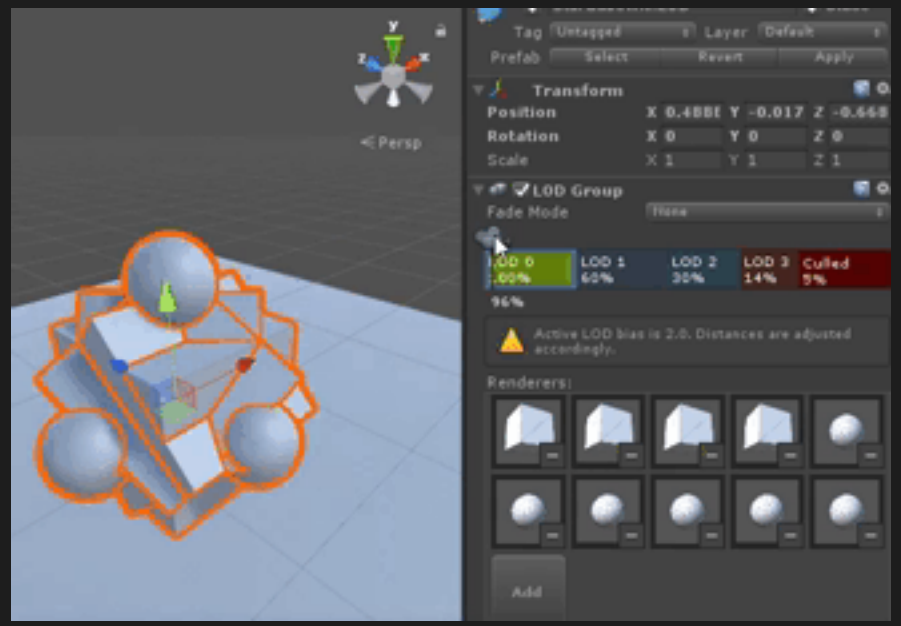
- Create an EmptyObj
- Create N Children, Child_N will have a lower LOD than Child_N-1
- Add LODGroup component to the root
- Drag Each LOD Children to the linked LOD
- To add a LOD Layer, rightClick on LOD row in the inspector
- Check the difference between creating a city using Home2_LODGroup prefab, and Home2
- Results
  - Less vertices in the scene
  - Drawcalls are not lower: Home2_LODs LOD1/2 could be improved using a single material (they are not optimize for this at the moment)!

- The percentage value is the obj cover of vertical viewport. If you want, you can use a multiplier for this: ObectSize
- RecalculateBounds recalculate the bounding volume of the object after a new LOD level is added
- LightmapScale updates the Scale in Lightmap property in the lightmaps
- In [1], LOD1/60% means that when the object pixels will occupy between 60% and 30% of the screen, we'll use LOD1

[LOD_Start]



[1]



LOD

Use a common renderer

- You can assign by hand the same Meshfilter to more than one LOD (and remove the MeshFilters/MeshRenderer components on the other gameObjects)

Occlusion culling consistency

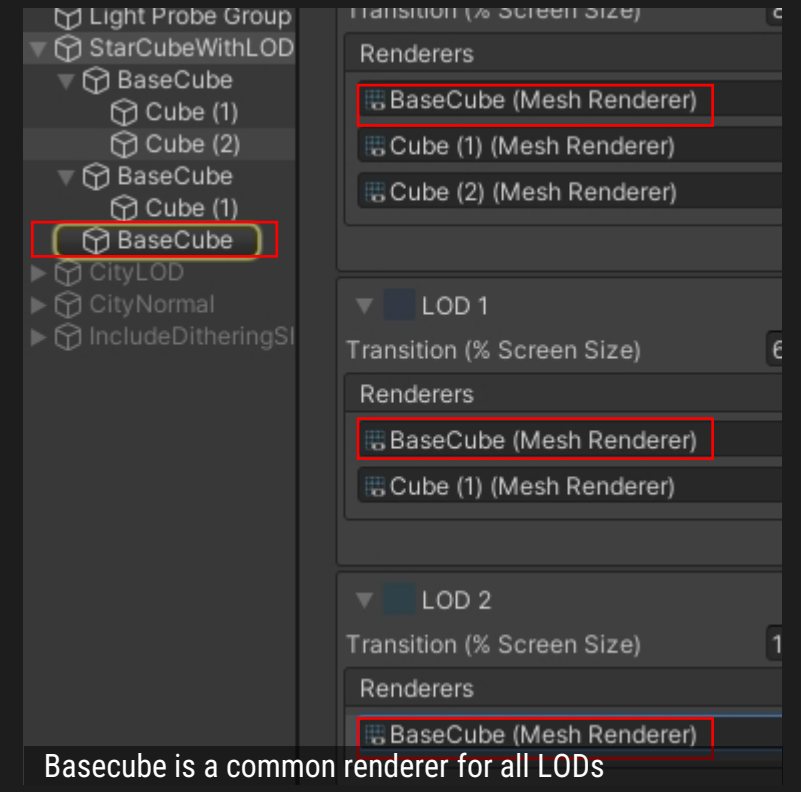
- Avoid significant volume changes across LOD levels

- ProjSettings/Quality
  - LODBias a value [0-1] prefer the less detailed model [>1] prefer the more detailed model
  - MaximumLODLevel If you are exporting your app to a mobile device, you may want to have a max LOD level of 1, never reach LED0. Use MaximumLODLevel 1 for this case

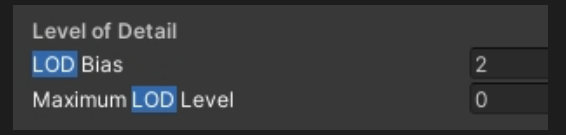
FadeMode

- Unity doesn't provide a ready-to-use solution for fade fx between LOD Objs
- FadeTransitionWidth 0.5 means that half the LOD's range is used to fade to the next LOD level
- [BIRP] Try to use CrossFadingLodDither Material. Its shader supports LOD (But not shadowing)
  - LOD_FADE_CROSSFADE directive
  - Use unity_LODFade.x to know the fade amount for the object

- LOD is not free: LOD meshes need to be loaded into RAM, the LODGroup Component must check Camera distance
- Scenes with large, expansive views of the world and have lots of Camera zoom movements, you might want to consider implementing this technique very early
- Indoor scenes or with a Camera looking down at the isometric world (see img on the right) will have little benefit in this technique since objects will tend to be at a similar distance from the Camera



Basecube is a common renderer for all LODs



[LOD_Start]

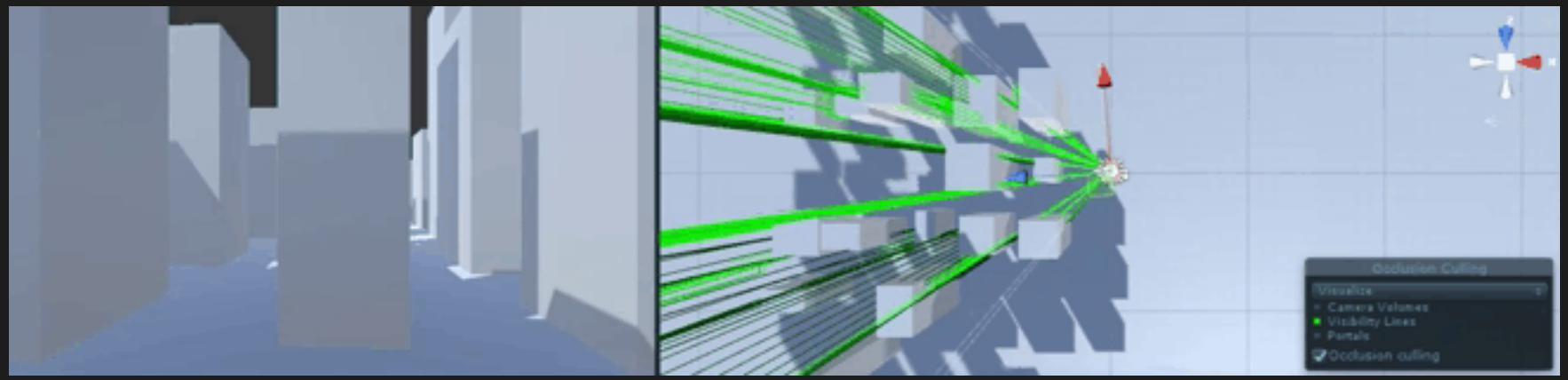
Occlusion culling

- Z-buffering with a front-to-back primitive rendering order gets pretty close to the ideal of only calculating a single value for each output pixel
- Unfortunately, the culling takes place very late in the rendering pipeline of a 3D game application
- Methods exist for early Z-culling in the rendering pipeline: occlusion culling refers to this type of rendering optimization
- This addresses Fillrate (Overdraw)

- Frustum Culling culls objects outside the current Camera view. It is always active and automatic
- Occlusion Culling works by partitioning the world into a series of small cells, calculating which cells are invisible from other cells

- Cost
  - additional disk space - store the occlusion data
  - RAM - keep the data structure in memory
  - CPU time - determine which objects are being occluded in each frame

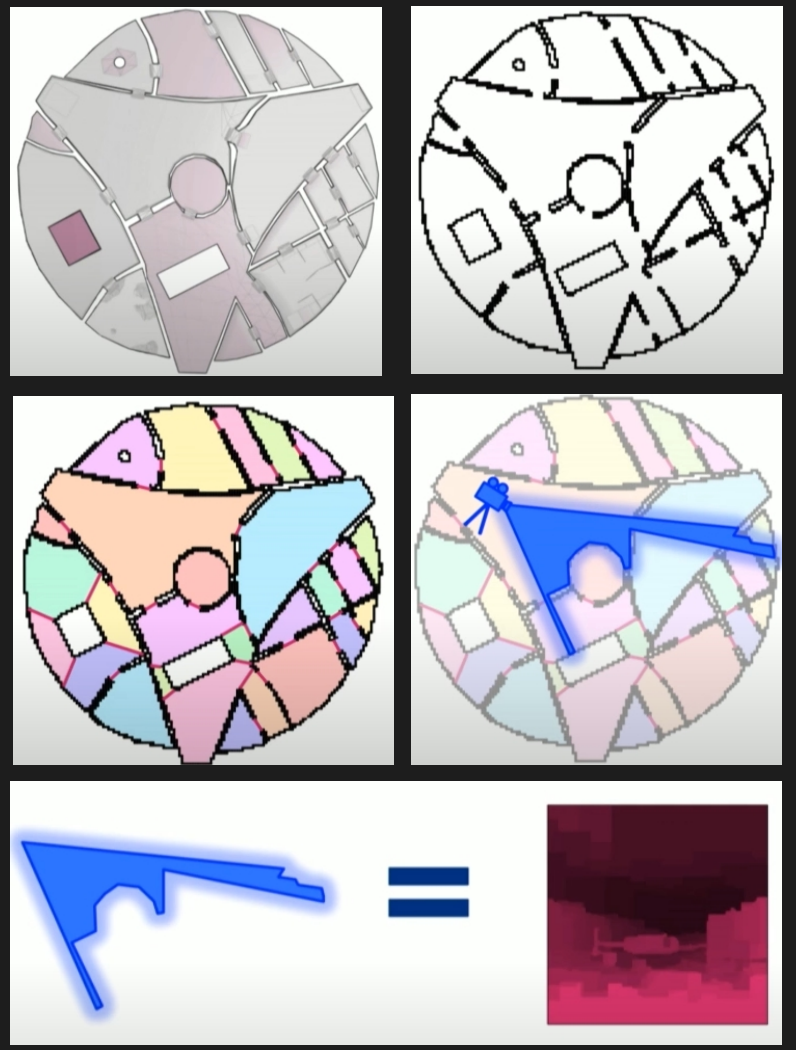
- Even though an obj may be culled by occlusion, its shadows must still be calculated



Occlusion culling

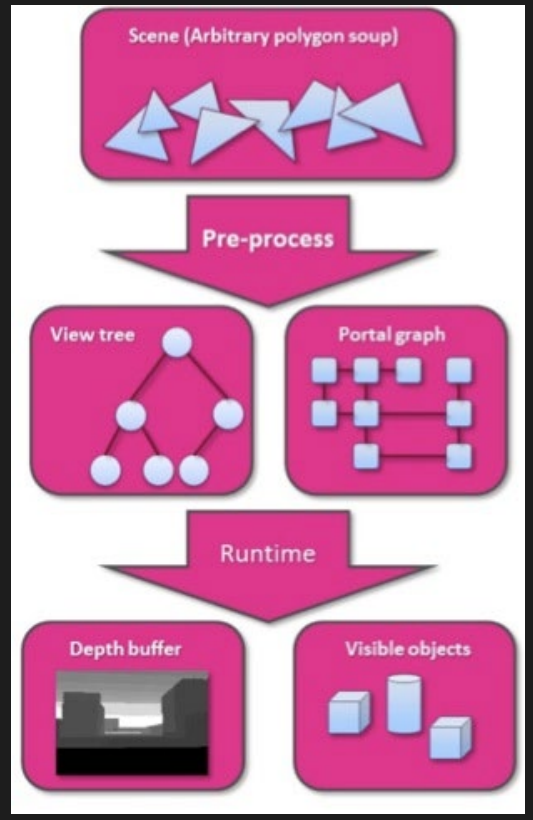
Umbra middleware consists of 2 components:

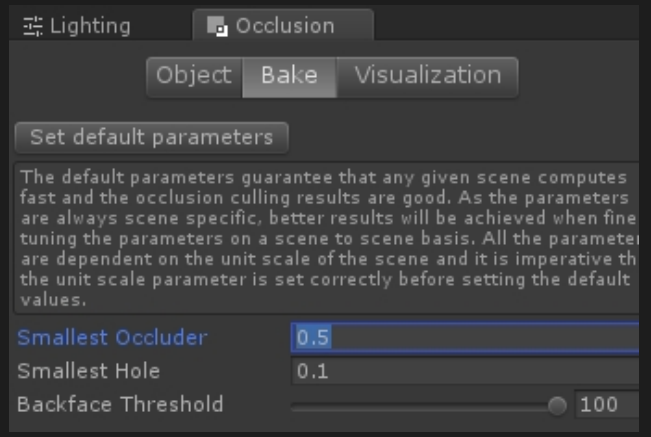
- Offline optimizer
  - Scene voxelization: the computation becomes easier because a lot of details are discarded. The voxelization takes a lot of memory, so we need to optimize further: we need to translate voxels into
  - Cells, by flood filling the empty boxes. We identify cells which are connected regions, and the
  - Portals between cells. Finally, we can compute the
  - Occlusion data
- Runtime component
  - Portal meshes rasterization into depth buffer
  - Cam pos/orientation > Umbra > Visible objects list (conservative)
- Tradeoffs
  - Least conservative > hi-res data > slow run-time traverse
  - If OC takes more frames than it saves > no sense

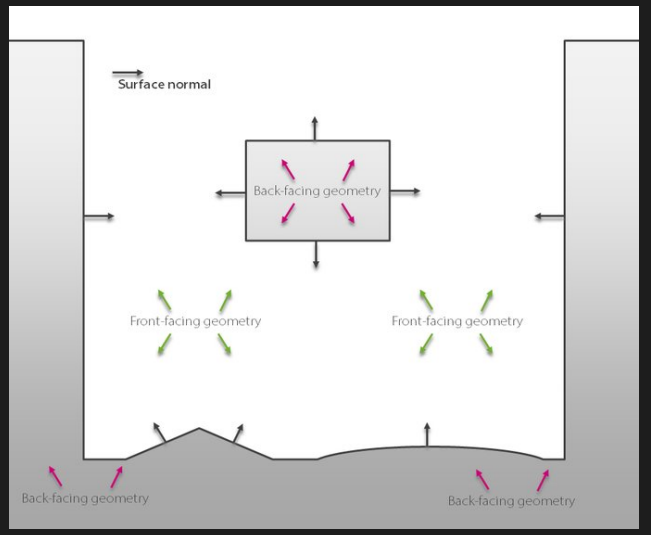


Occlusion culling

- SmallestHole is like Umbra input resolution of the bake [0.05, 0.5]
- SmallestOccluder is like Umbra output resolution. Larger values = faster run-time OC performances = increased conservativity (false positives) [2, 5]
- If you need to reduce the size of the baked data, Unity can sample the Scene as it bakes, and exclude parts of the Scene where the visible occluder geometry consists of more than a given percentage of Backface threshold. An area with a high percentage of backfaces is likely to be underneath or inside geometry, and therefore not likely somewhere the Camera is at runtime. The default value of 100 never removes areas from the data. Lower values result in a smaller file size, but can lead to visual artifacts.
    - Try reducing the value to 90, which should drop a lot of data underneath terrains for example. If you start popping into rendering artifacts, increase the value back to 100 and see if it fixes the problems







Occlusion culling

Umbra middleware was used in:

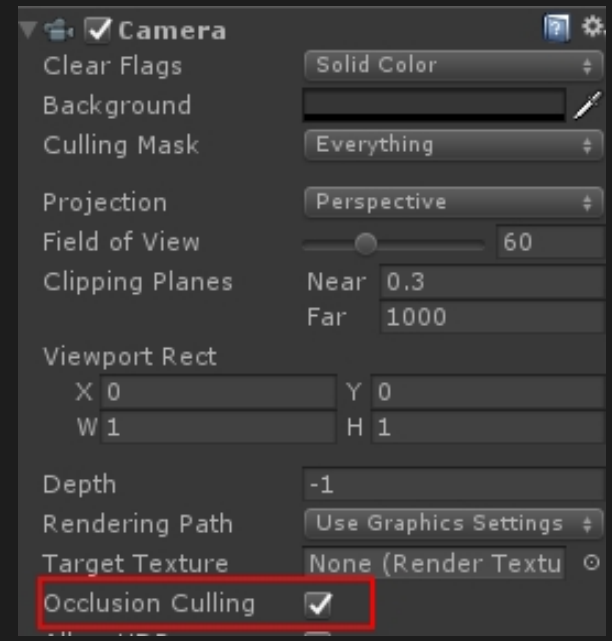
- Remedy & 505 Games - Control
- iD Software / Bethesda Softworks - DOOM Eternal
- Capcom - Monster Hunter World: Iceborne
- Arkane Studios / Bethesda Softworks - Dishonored 2
- CD Projekt Red - The Witcher 3: Wild Hunt
- Infinity Ward / Activision - Call of Duty: Modern Warfare
- Bungie - Destiny 2
- Square Enix - Final Fantasy XV

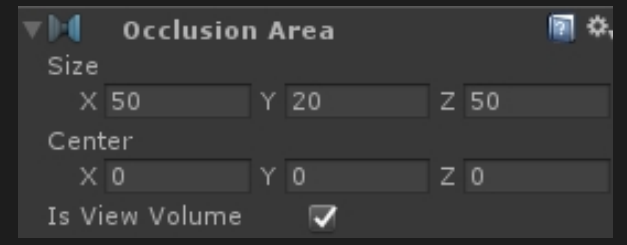
Occlusion culling setup

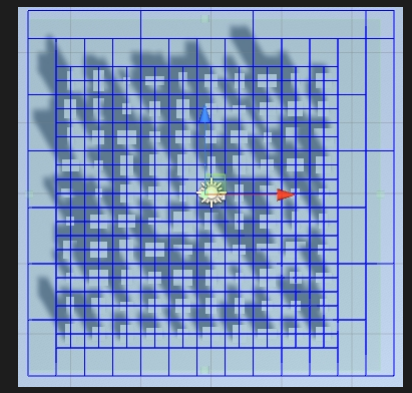
- Rendering Camera must have OcclusionCulling flag ON

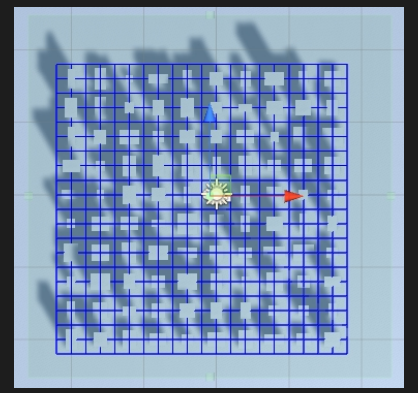
- Occluder Static Static objects which can hide other objects behind them, as well as be hidden behind each other
- Occludee Static is a special case: transparent or small static objs, that always require other objs behind them to be rendered, but they themselves need to be hidden if something large blocks their visibility
- OcclusionAreas
  - If not present, occlusion culling will be applied to the whole scene
  - It is the only way to occlude moving objects
  - isViewVolume
    - ON Camera can be inside this Area
    - OFF Camera can only look at this area

[OcclusionCulling_Start_02]



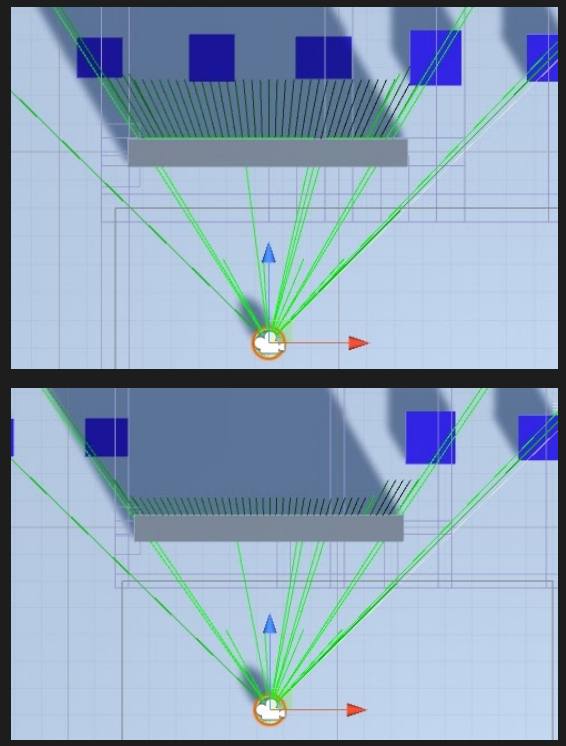






Occlusion culling setup

- Try it
  - Under CitySpawner spaw a city, with OccluderStatic flag active
  - Add OcclusionArea component to OcclusionArea Gobj and resize it
  - Open Window/Rendering/Occlusion panel
    - Smallest occlude: 1 Smallest Hole: 0.05
    - Bake
  - Under DynamicObjs_End
    - We have blue cubes that are dynamic: they will be occluded by the BigWall Cube, they can't occlude themselves
    - BigWall is OccluderStatic
    - Open Window/Rendering/Occlusion panel
      - Smallest occlude: 1 Smallest Hole: 1
      - Bake
    - Values are too high: BigWall can't occlude blue cubes
      - Try with these values:
      - Smallest occlude: 1 Smallest Hole: 0.05
      - Bake
    - Now it works
  - Try to configure Occlusion culling for StaticObks/CityHomeSpawner GObj



[OcclusionCulling_Start_02]

Occlusion culling setup

- Try it
  - Under CityHomeSpawner spaw a city, with OccluderStatic flag active
  - Add OcclusionArea component to OcclusionArea Gobj and resize it
  - Open Window/Rendering/Occlusion panel
    - Smallest occlude: 3 Smallest Hole: 1
    - Bake

[OcclusionCulling_Start_02]

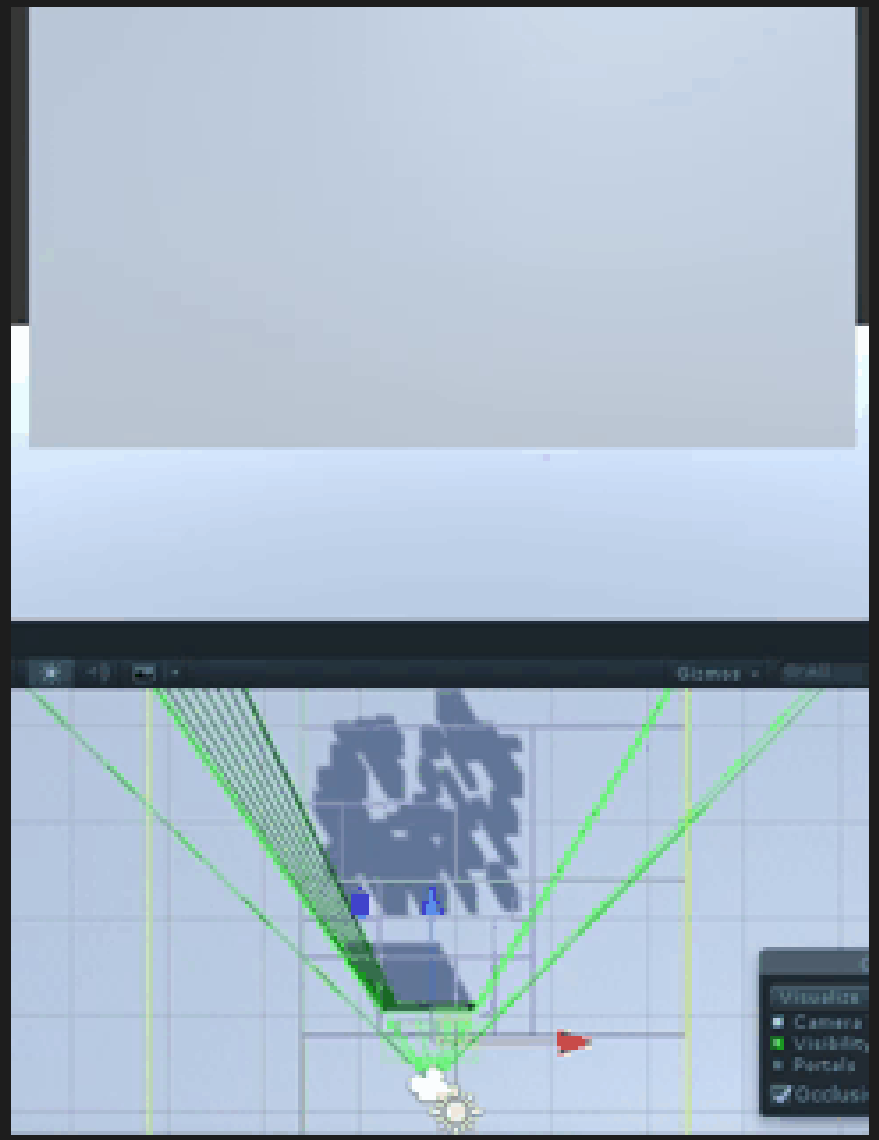
Occlusion Portal

- OcclusionPortal
  - DoorOccPortal Gobject has OcclusionPortal component
  - Use them to add dynamic Occluders (they can't animate, but can be enabled or disabled) into your Occlusion Area

- Try it
  - Enable OcclusionPortals
  - Bake Occlusion Area - Smallest occlude: 1 Smallest Hole: 0.05

  - Enable/Disable the Door using DoorOccPortal/PortalSwitcher OpenDoor/CloseDoor
    - It uses OcclusionPortal.open = true/false

[OcclusionCulling_Start_02]

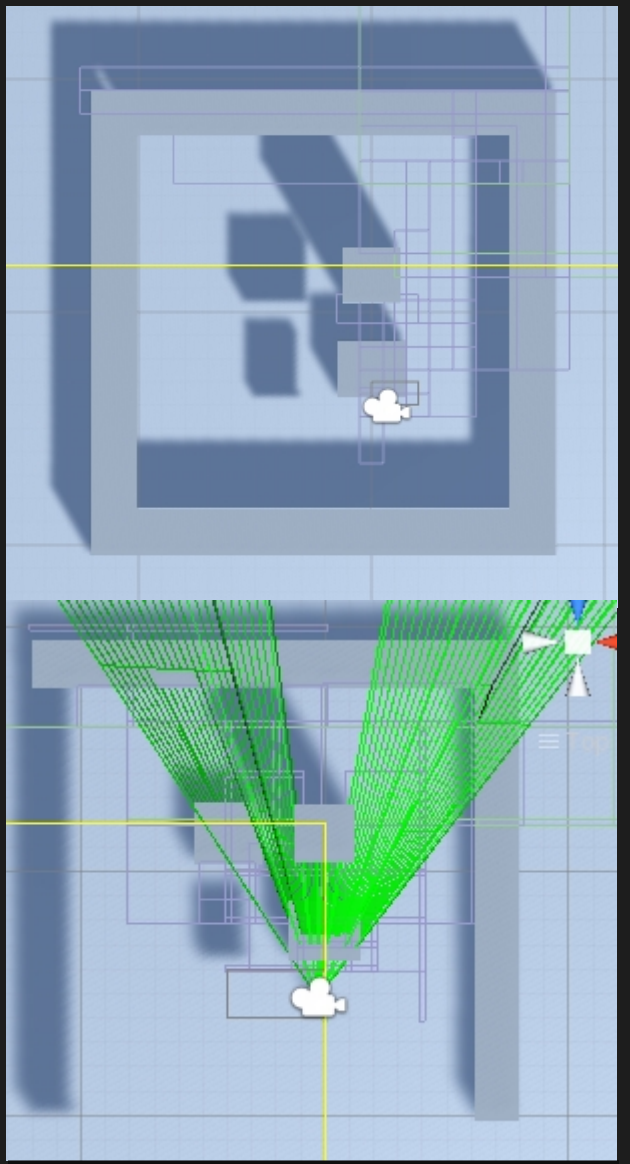


Occlusion culling Best practices

- Occlusion quality
  - Large occluders are better
  - Umbra is not able to perform occluder fusion > trees and forests are bad occluders
  - Avoid cells that are too small in comparison with your objects (objects that cover many cells)
- Objs flags
  - Non-opaque obj > Occludee, NOT Occluder
  - Unique scene GameObject with small holes that you wish to see through > remove them from OC (you would use low SmallestHole value only for this GObj)
  - If the camera can be inside an occluder > remove the occluder flag, otherwise from inside the occluder you'll see nothing outside (everything should be culled)
- Objs granularity
  - You can use giant occluders, but obj subdivision of occludees matters
- Use Occlusion area to apply OC also to DynamicObjs

- Pay attention to mesh subdivision if you want to use occlusion culling (see Occlusion_MeshSubdivision WallsEntireObj/Walls_Divided) [1]
- BatchingStatic objects are still able to be optimized by Occlusion Culling (try to set City cubes also BatchingStatic)

- Debug
  - CameraVolumes if it looks like the cell bounds don't make sense, e.g. when the cell incorrectly extends to the other side of what should be an occluding wall, something is wrong
  - Visibility lines helps to figure out which holes cause occlusion artifacts



[1] If walls are one single mesh, they can't be culled

[OcclusionCulling_Start_02]

Culling groups

- Offers a way to integrate your own systems into Unity's culling and LOD pipeline
  - You know that Unity is able to not render hidden objs, but hidden objs don't cost anything: do you need to run all your scripts when your object is not visible?
  - Skipping rendering particle systems that are behind a wall
  - Tracking which spawn points are hidden from the camera in order to spawn enemies without the player seeing them 'pop' into view (spawn points are empty objs)
  - Switching characters from full-quality animation and AI calculations when close, to lower-quality cheaper behavior at a distance (avoid to use Unity LOD system and activate Animation layers by yourself)
  - Having 10,000 marker points in your scene and efficiently finding out when the player gets within 1m of any of them
  - No 1:1 relashionship necessary: one cullingGroup can suffice for multiple objects!

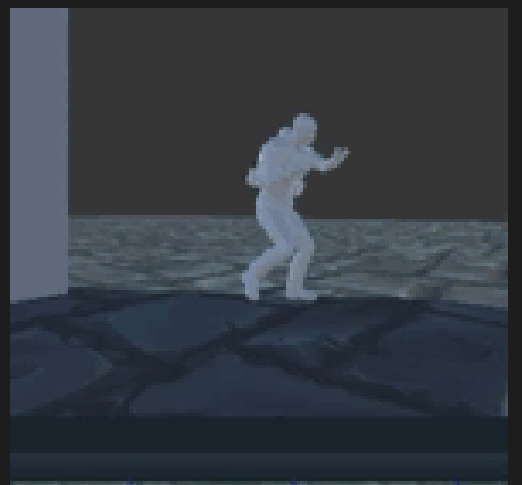
- The CullingGroup will calculate visibility based on
  - frustum culling
  - static occlusion culling if baked
    - try to add an OcclusionArea

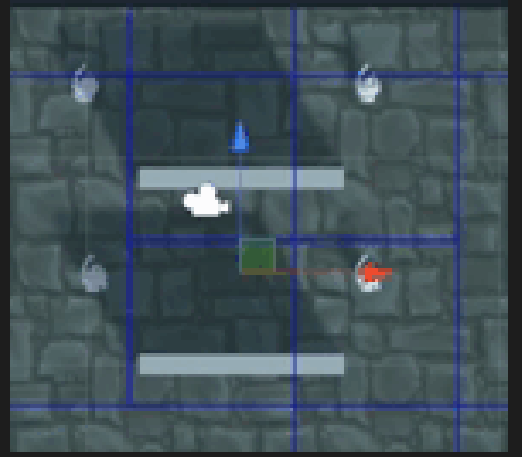
Culling groups

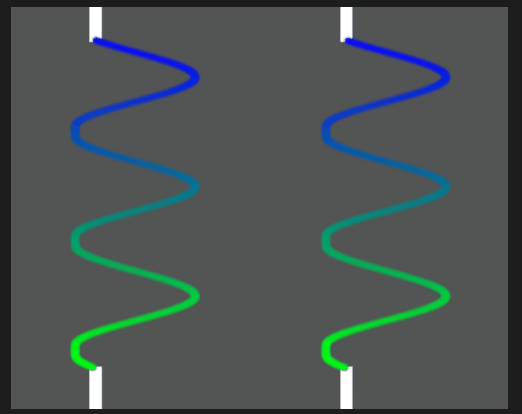
How to use it:

- CullingGroup group = new CullingGroup(); //Create a culling group
- group.targetCamera = Camera.main; //Assign the target camera
- group.SetBoundingDistances(new float[] { 1, 5, 10, 30, 100 }); //Distance bands
- group.SetDistanceReferencePoint(Camera.main.transform); //Starting point to measure distants bands
- bounds = new BoundingSphere[100]; //Prepare more space than you need at start
    - Create and populate an array of BoundingSphere structures with the positions and radii of your spheres, and pass it to SetBoundingSpheres along with the number of spheres that are actually in the array. The number of spheres does not need to be the same as the length of the array; it is a good rule create an array that is big enough to hold the most spheres you will ever have at one time, even if initially the number of spheres you actually have in the array is very low. This way you avoid having to resize the array as spheres are added or removed, which is an expensive operation
- bounds[i].radius = r; bounds[i].position = newPos; //Set pos and radius of i-th BoundingSphere
    - you are observing n spheres, each sphere has its radius and its position
- group.SetBoundingSpheres(bounds); //This cullingGroup is able to track up to 100 BoundingSpheres
    - provided array is simply referenced, not copied; therefore you can simply modify the data in the array on successive frames without calling SetBoundingSpheres again each time
- group.SetBoundingSphereCount(1); //Tell the cullingGroup to track only the first sphere
- group.onStateChanged = OnChange; //Register our cullingGroup listener
- group.Dispose() //OnDestroy

[CullingGroup_02, CullingGroupOps.cs]







Culling groups

- Currently a single CullingGroup only supports a single camera. If you need to evaluate visibility to multiple cameras, you should use one CullingGroup per camera and combine the results
- The CullingGroup will calculate visibility based on frustum culling and static occlusion culling only. It will not take dynamic objects into account as potential occluders
- Any spheres that are beyond the last distance band will be considered to be invisible, allowing you to easily construct a culling implementation which completely deactivates objects that are very far away. If you do not want this behaviour, simply set your final threshold value to be at an infinite distance away

[CullingGroup_02, CullingGroupOps.cs]

Culling groups

Try it

- Activate Particles
- Particle systems are paused when not in the frustum camera
  - Try to move away GameView camera and see the particle systems using SceneView: they will be paused/hided, depending on CullingGroupOps settings
- Activate ActiveAnimations
- Particle systems are paused when not in the frustum camera
  - Try to move away GameView camera and see the particle systems using SceneView: they will be paused/hided, depending on CullingGroupOps settings

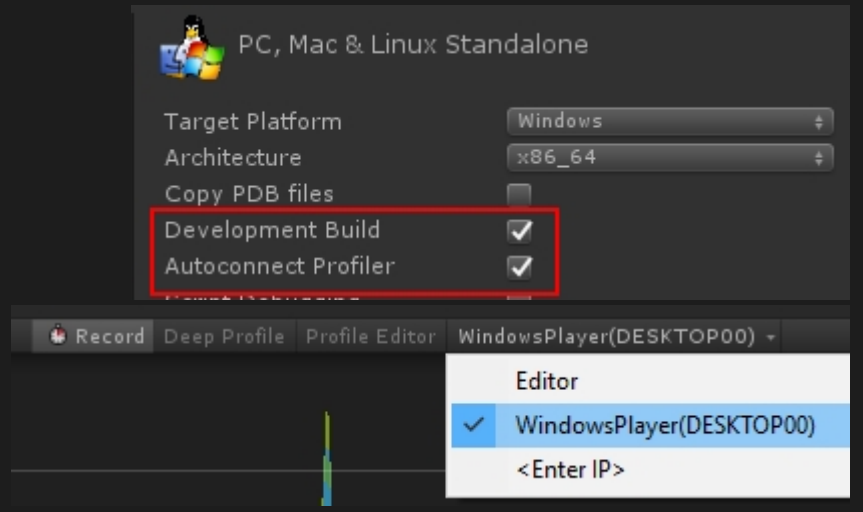
- Similar effect can be achieved using the MonoBehaviour.OnBecameVisible() method if the object has a MeshRenderer component
  - Use CullingGroups when you need to cull
    - Empty GameObjects
    - When you want a centralised method of tracking object visibilities
    - Keep track of distance of objects w/o use Physics collisions

[CullingGroup_02, CullingGroupOps.cs]

If occlusion culling is used, even if soldiers are inside frustum culling, they are disabled if we choose CugglingGroupOps.CullType == ActivateGO

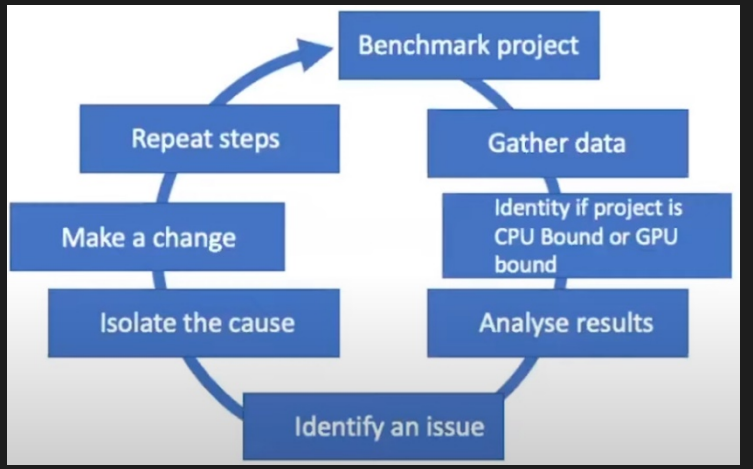
Profiler

- CPU consumption
- Rendering and GPU information
- Runtime memory allocations
- Audio source/ data usage
- Physics Engine (2D and 3D) usage
- Video playback usage
- Basic and detailed user interface performance
- Global Illumination statistics



Main uses

- General overview of what's going on in your game
- Spikes in CPU/GPU activities
- How performant are your scripts in your project

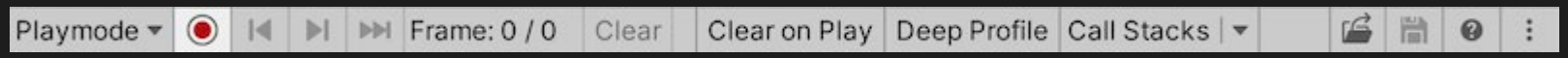


- Profiling has its own performance cost: The performance of the editor affect profiling data, and the Profiler affect the performance of the Editor. You can start also the Profiler as a separate Standalone Process [Windows/Analysis/Profiler (StandaloneProcess)]: ensures cleaner Profiler data when you target the Editor or Play mode. It also reduces overhead, because the Profiler isn't profiling itself or sharing a process with your application or the Editor
- Good profiling is done on exported build (StandaloneProfiler + DevelopmentBuild/AutoconnectProfiler)
- Let's say you are aiming for 60 FPS. Then 1/60 = 0.016: this means that we have a budget of about 16ms per frame
- Look at your profile: Is your frame taking longer than 16ms? Then, it is a bottleneck

Try it

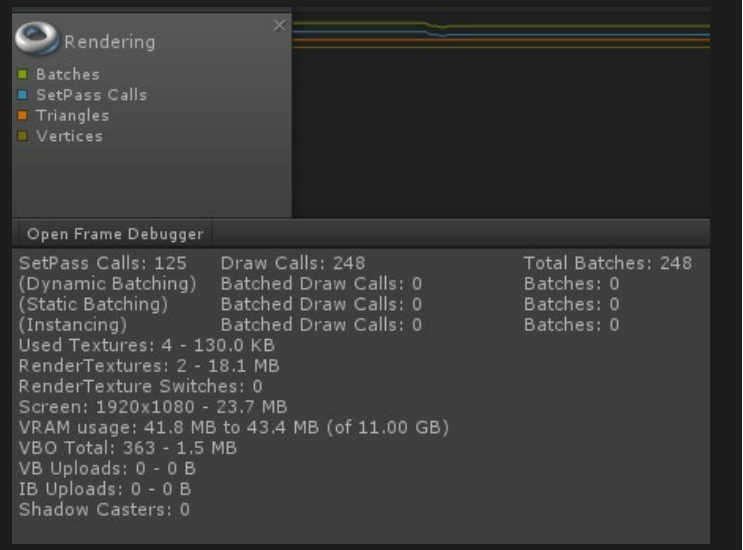
- Build the scene with DevelopmentBuild and AutoconnectProfiler to TRUE
- Execute the exported .exe,
- In the Profiler it should appear a WindowsPlayer(DESKTOPXX) item in the DropDown menu

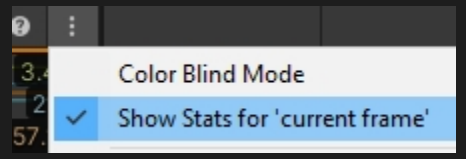
Profiler

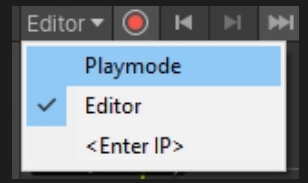


ProfilerControls

- ProfilerModules
- Playmode/Editor Profile custom Editor Scripts
- Record must be active to record data
- CurrentFrameMode in this mode profiler stays on the current frame and displays the data it collects in real-time. Click the button again to exit Current Frame mode. If ShowStatsForCurrentFrame is active, if will display value labels also during this mode
- DeepProfile Without adding more explicit ProfilerMarker instrumentation to your own code, the samples you can see as child samples of your scripting code are those that call back into Unity's API, if that API has been instrumented. When you enable the Deep Profile setting, the Profiler profiles every part of your script code and records all function calls, including at least the first call stack depth into any Unity API. This is useful information to help you work out where your code impacts on your application's performance, but it comes with a large overhead
- Load/Save – Up to 300/2000 frames of data (set it into Preferences/Analysis/Profiler): you can share your profiler recordings

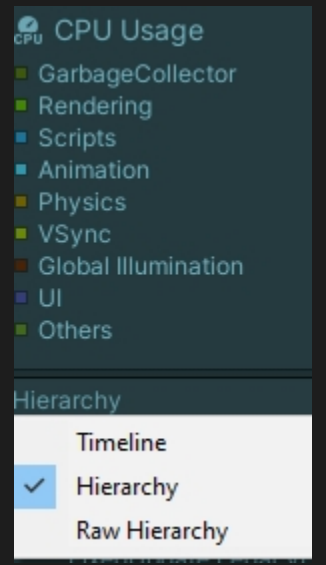






Profiler

- Timeline
  - Shows CPU tasks order. You can focus on each task by pressing ‘F’
  - If focusing on script tasks, you can highlight the associated GameObject by clicking on the task
  - Which thread is responsible for which tasks
    - Main
    - Render
      - Sends commands to the GPU
    - Working thread / Jobs
      - Perform a single task (culling/skinning, etc.)
      - The more CPU cores our target hardware has, the more worker threads can be spawned. Hence, it is very important to profile our game on target hardware; our game may perform very differently on different devices

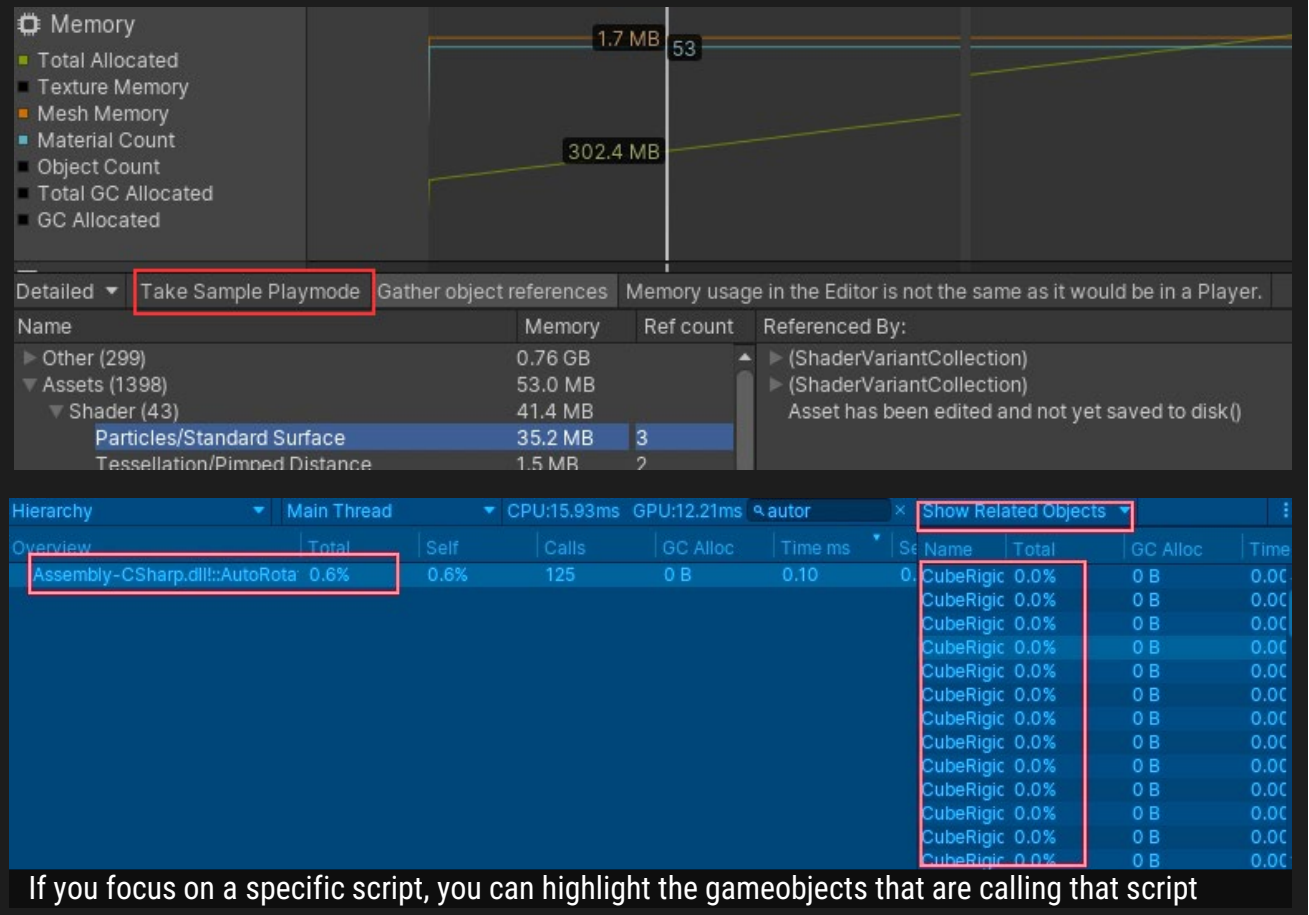


- Hierarchy
  - Groups the timing data by its internal hierarchical structure. This option displays the elements that your application called in a descending list format, ordered by the time spent by default. You can also order the information by the amount of scripting memory allocated (GC Alloc), or the number of calls

- RawHierarchy
  - Raw Hierarchy view does not group samples together, which makes it ideal for looking into samples on a granular level

Profiler

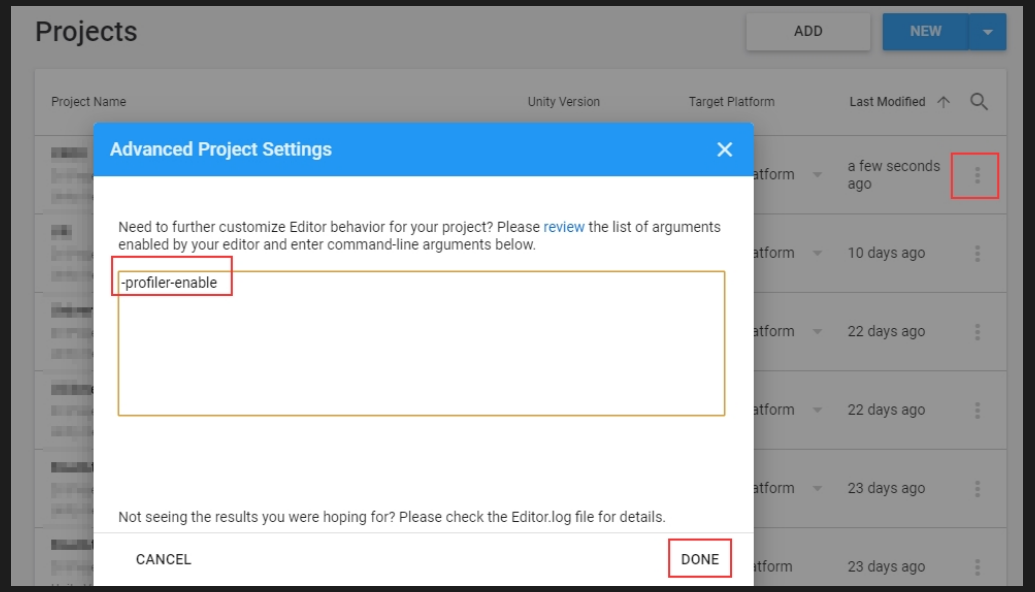
- Hierarchy view
  - You can sort by percentage, total time, search for a specific task
  - You can choose to have No details/Show related objects/Show calls about the task you are focusing on (on the right)
- CPU Module
  - Rendering data are the instructions that are sent from CPU to GPU
- Rendering Module
  - Draw/SetPass calls
  - You can access FrameDebugger from the Rendering detail area
  - The Screen resolution memory grows as soon as we extend the FreeAspect Area of our game
- Memory Module
  - You can get an in-depth view taking a Sample during playmode
  - It is not a look inside the Heap memory: for that, there is the MemoryProfiler. This is a look to the resources you are using
- Physics
  - General Rule of thumb: you may want to have less active dynamic objects. A typical mistake is not to set objects static for those that ARE static

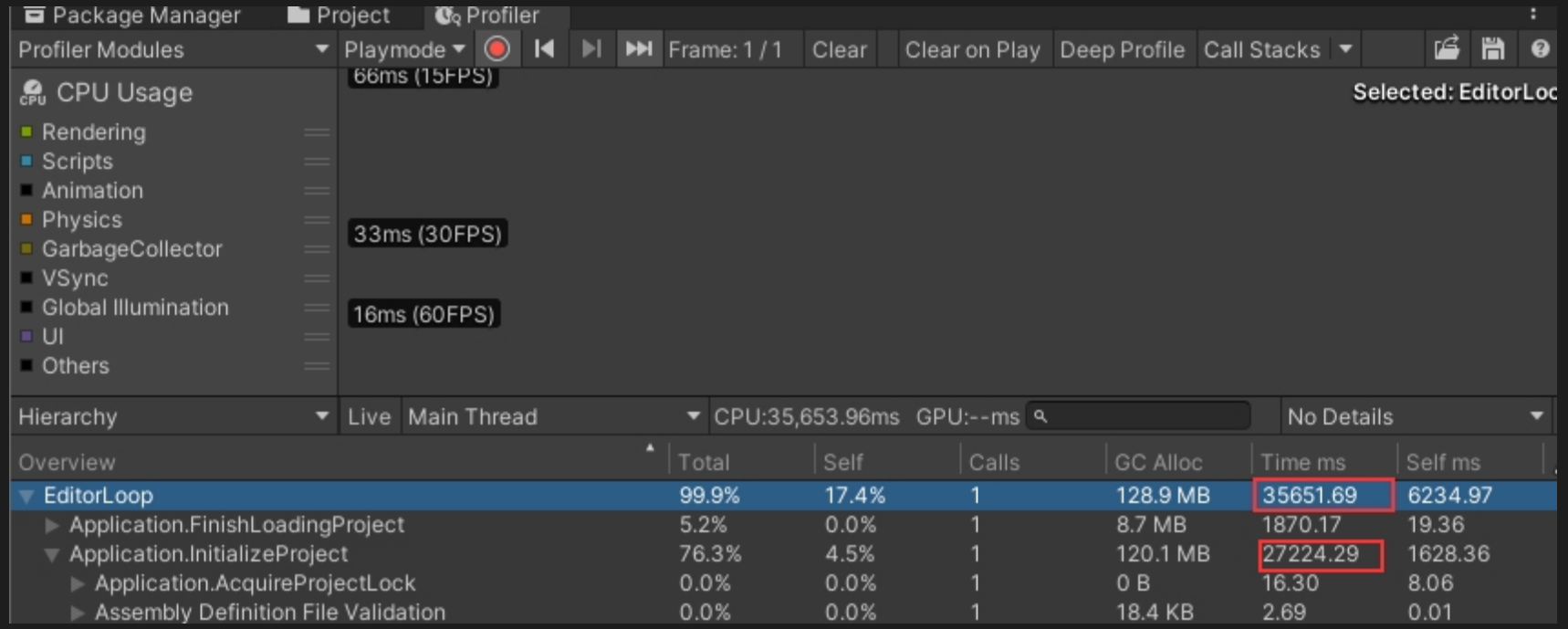


If you focus on a specific script, you can highlight the gameobjects that are calling that script

Profile startup times

- As a project becomes more complex and uses more features, it takes longer to open. Being able to open your projects quickly is important.
- you can supply upon opening Unity, the -profiler-enable command line argument allows you to profile the Editor during launch (use AdvancedProjectSettings option at startup)
- If you open Profiler window and add the CPU Module, you can see that opening this particular project takes ~35 s. It appears to take 27s to load the AssetDatabase



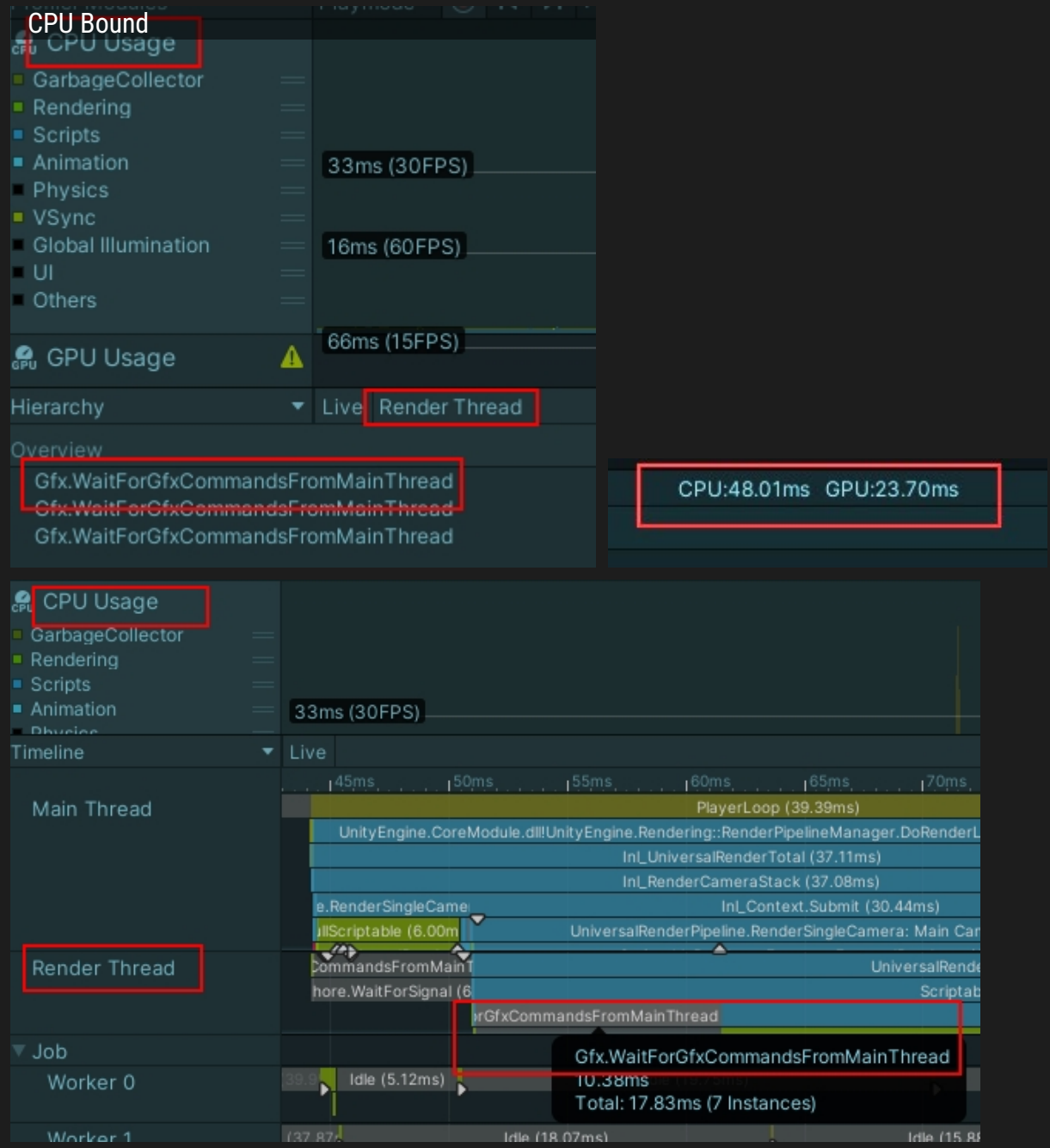


CPU / GPU Bound

CPU Bound
- Open [Profiler_CPUBound]
- Disable SRPSettings/Advanced/DynamicBatching
- Try to move the camera inside SceneView: if you still can move smoothly, try to increase Spawner x/y/zLen
- Lot of drawcalls w/o optimization
- If a large part of the frame time on a slow frame is taken up by rendering, this indicates that rendering could be the cause of our problem
- Profiler/CPU Timeline // Hierarchy/RenderThread should display Gfx.WaitForGfxCommandsFromTheMainThread

GPU Bound
- Few drawcalls for the CPU
- Extremely heavy Fragment shaders
- High resolutions

[Profiler_CPUBound]

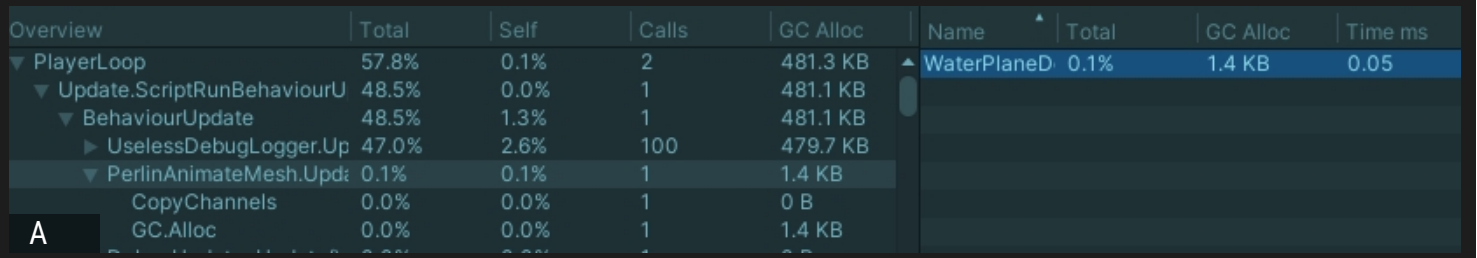


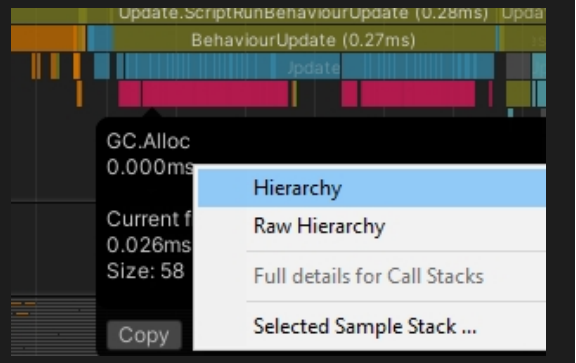
Profiler Example

- Start from [Profiler_Samples_03]

Memory alloc

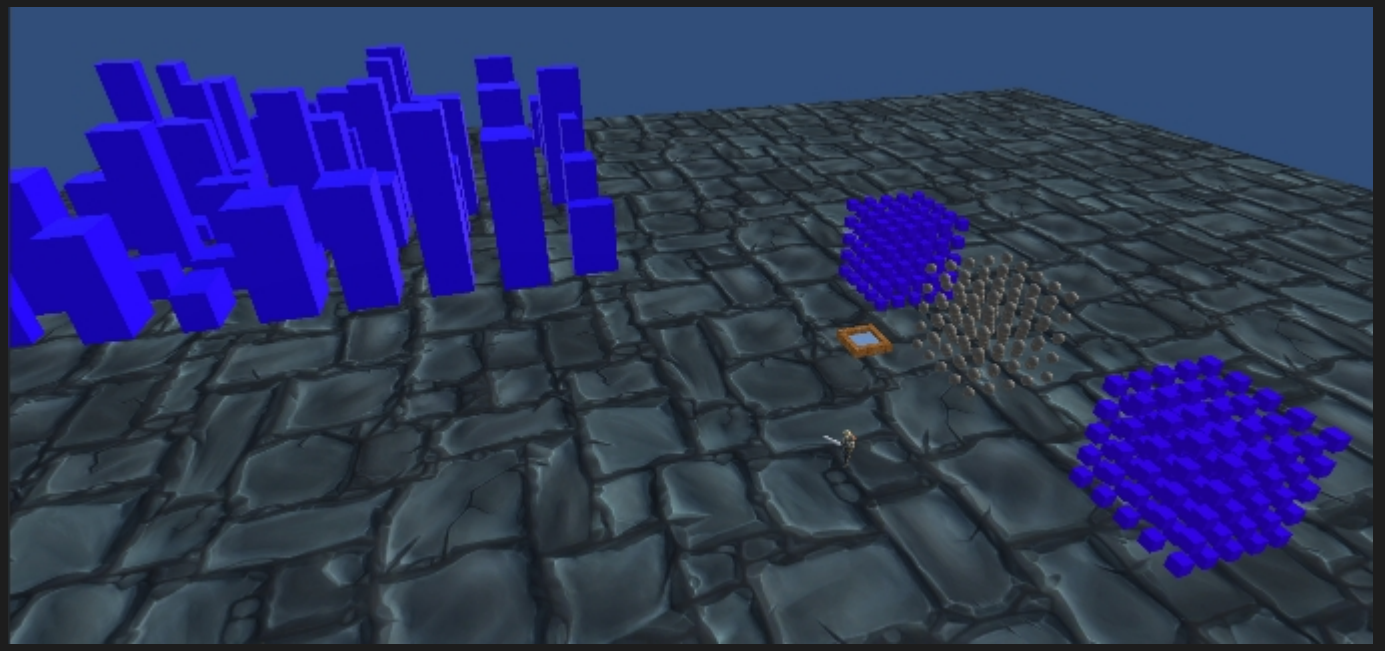
- PerlinAnimateMesh.cs has a CG alloc of 1.4KB every frame. This is because every frame it requests vertices[] to the Mesh filter [A]
  - PerlinAnimateMeshNoAlloc.cs solves the problem (see the differences in between the scripts) [B]
- UslessDebugLogger.cs alloc X KB every frame. This is because we have 100 calls with a è= string concatenation operator [C]
  - Check Optimize StringAlloc: now we use StringBuilder [D], and we alloc 0KB!
  - In TimelineView, CGAlloc is Pink

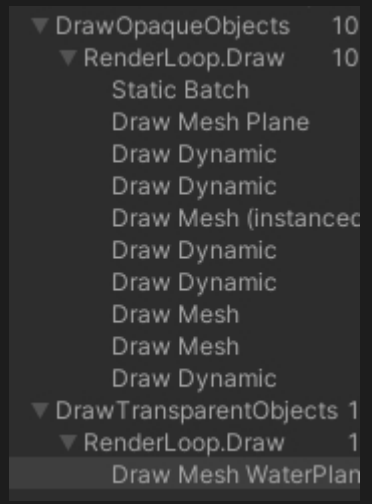




Profiler Example

- Place the camera in order to see the entire scene
- Enable GPUInstancing, Static, DynamicBatching
- FrameDebug shows us that we are using
  - StaticBatching, DynamicBatching, GPUInstancing
- Enable also SRPBatching
  - Now everything is SRPBatched

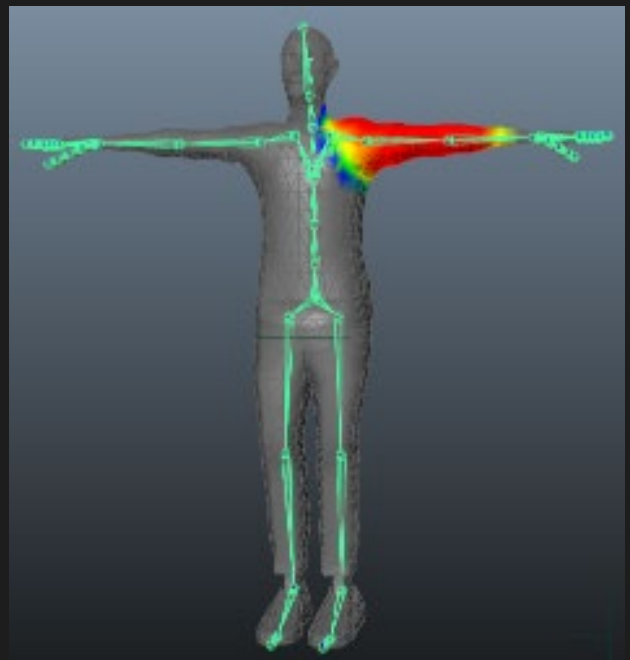




GPU Skinning

- CPU/GPU FrontEnd
- To test it, hide the scene view

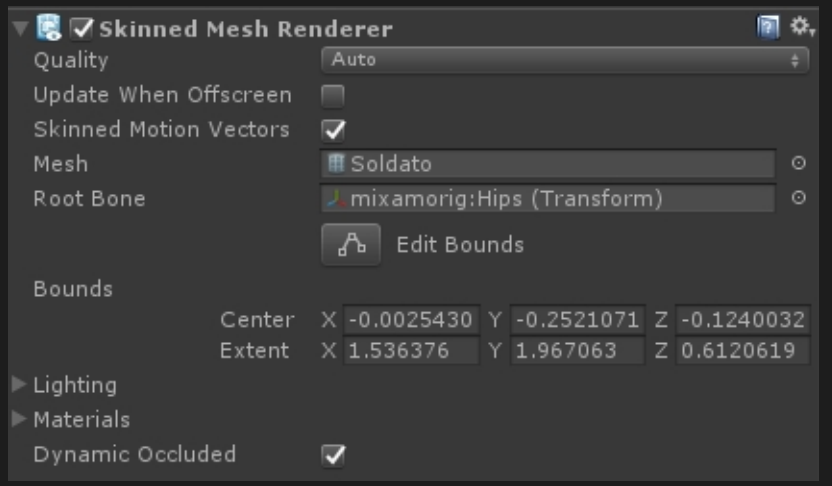
- If we don't import animations in the model's Import Settings, the model will have a MeshRenderer instead of a SkinnedMeshRenderer
- Skinned mesh cannot be batched
- Skinning is the process where mesh vertices are transformed based on the current location of their animated bones
  - CPU performs animation system: transforms the object's bones to apply its current pose
  - CPU/GPU performs skinning: wrapping the mesh vertices around those bones to place the mesh in the final pose
- If PlayerSettings/GPUSkinning is ON
  - MeshSkinning.SkinOnGPU task appears on GPU
  - PostLateUpdate.UpdateAllSkinnedMeshes task is splitted between from CPU to GPU
  - Both CPU and GPU overhead should decrease
    - CPU performs less calculation (no skinning)
    - GPU is better in skinning calculation
  - DX11, DX12, OpenGL ES 3.0, Xbox One, PS4, Nintendo Switch and Vulkan
    - With GPU Skinning enabled, CPU must still transfer the data to the GPU and will generate instructions on the Command Buffer for the task > it doesn't remove the CPU's workload entirely, but helps



[GPUSkinning]

Skinned Mesh

- Quality Define the maximum number of bones used per vertex while skinning. The higher the number of bones, the higher the quality of the Renderer. Set the Quality to Auto to use the Meshes/SkinWeights value from the QualitySettings
- UpdateWhenOffscreen If enabled, the Skinned Mesh will be updated even when it can't be seen by any Camera. If disabled, the animations themselves will also stop running when the GameObject is off-screen

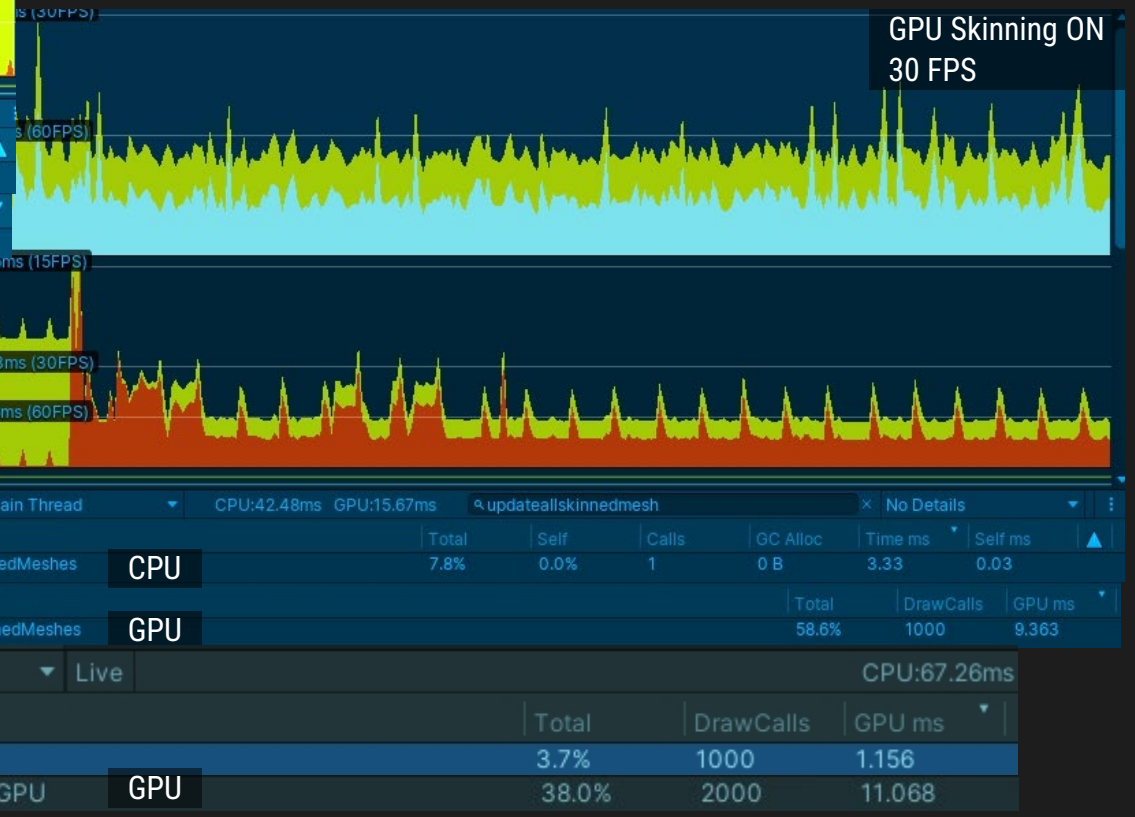


Try it

- Rotate the camera in order to look away from the skinned soldiers
- Search for meshSkinning.Update in CPU or GPU (if GPUSkinning is enabled) Profiler area.
  - NB: Scene view must be closed in order to test UpdateWhenOffscreen flag!

[GPUSkinning]

Skinned Mesh



Modeling Characters best practices

- Use a single SkinnedMeshRenderer for each character
- Use as few materials as possible
  - More than one material only if you DO need different shaders (E.g. Eyes)
    - Reduces RenderState switch calls on the GPU
- Use as few bones as possible
  - About 15 bones, no more than 30
  - Less bones reduces mesh deforming to calculate
- Don’t export IK nodes
  - IK nodes are baked into animations (FK)
  - Unity doesn’t need IK Nodes

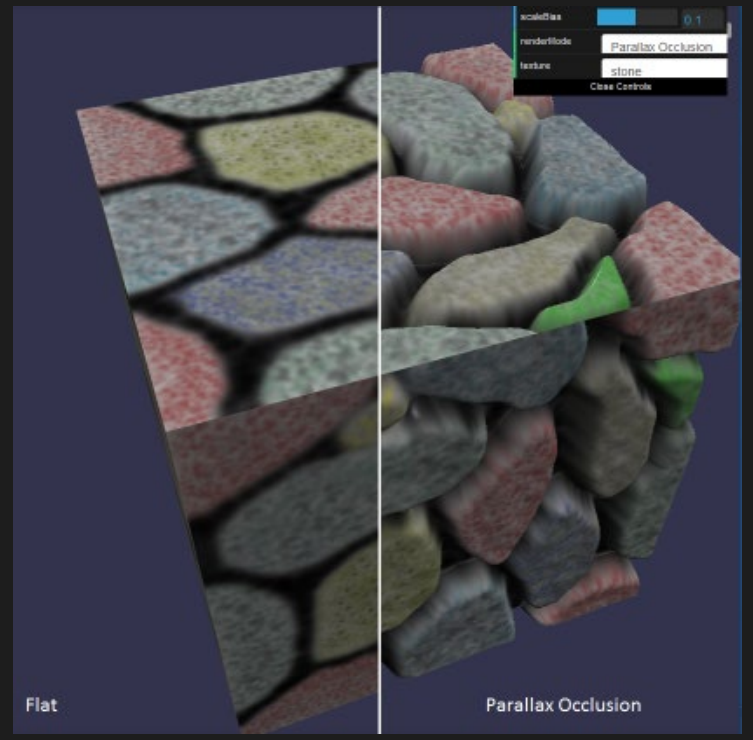
SkinnedMesh.Bake

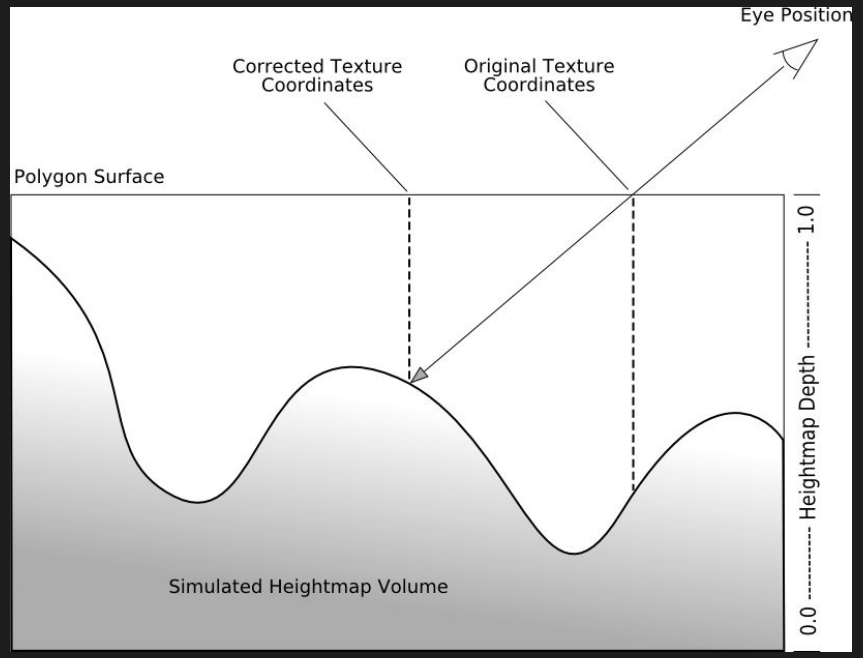
- If we are animating our object only some of the time (e.g., only on start up or only when it is within a certain distance of the cam)
  - Switch its mesh for a less detailed version
  - Take a static snapshot of the SkinnedMeshRender component, bake it into a MeshRenderer component
    - SkinnedMeshRenderer.BakeMesh(Mesh m) creates a mesh in a matching pose

[BakeMesh_SkinnedMesh_01]

Geometry

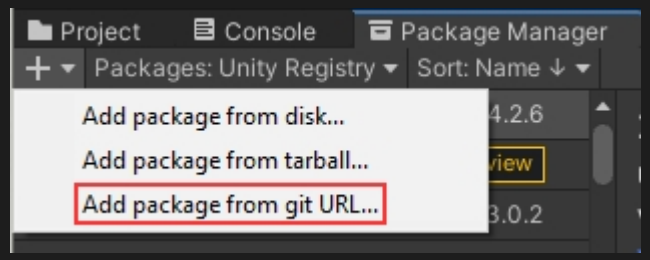
- Keep the geometric complexity of GameObjects in your Scenes to a minimum
- Remove faces from geometry that you cannot see, and don't render things the player never sees
- Simplify Meshes as much as possible
- Add details via high-resolution Textures to compensate for low poly geometry, potentially parallax mapping and tessellation
- Reduce pixel complexity (per-pixel calculations) by baking as much detail into the Textures as possible

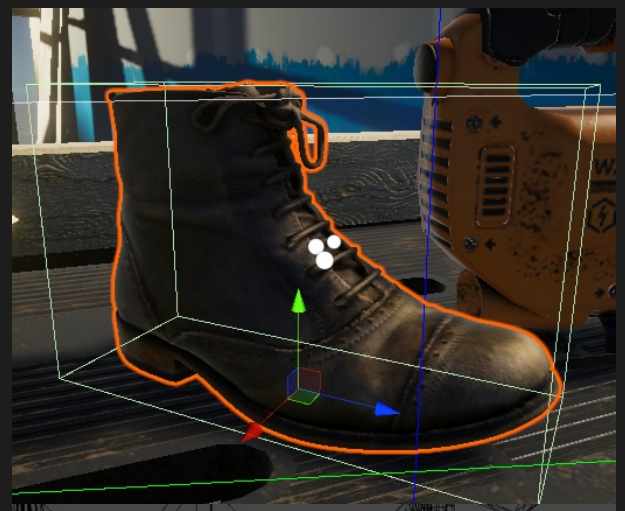


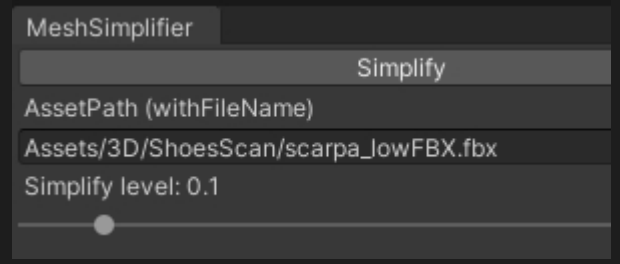


UnityMeshSimplifier

- Free tool that allows to reduce Mesh polygon count
- You can add it using a direct GitHub URL from PackageManager: [https://github.com/Whinarn/UnityMeshSimplifier.git]
- NB: You need Git installed in your system. If you don't have it, download and install from [https://git-scm.com/download/win]
- Try to simplify the [ShoesScan_00] using

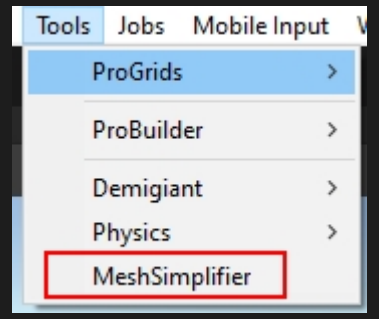


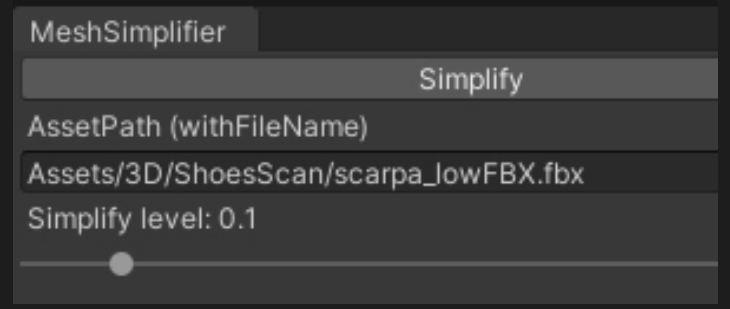




UnityMeshSimplifier

- Import [ShoesScan_00] and [MeshSimplifier.cs]
  - NB: If you are using AssemblyDefinitions, add UnityMeshSimplifier.Runtime Assembly dependency
- Try to simplify the imported shoe model, using Tools/MeshSimplifier



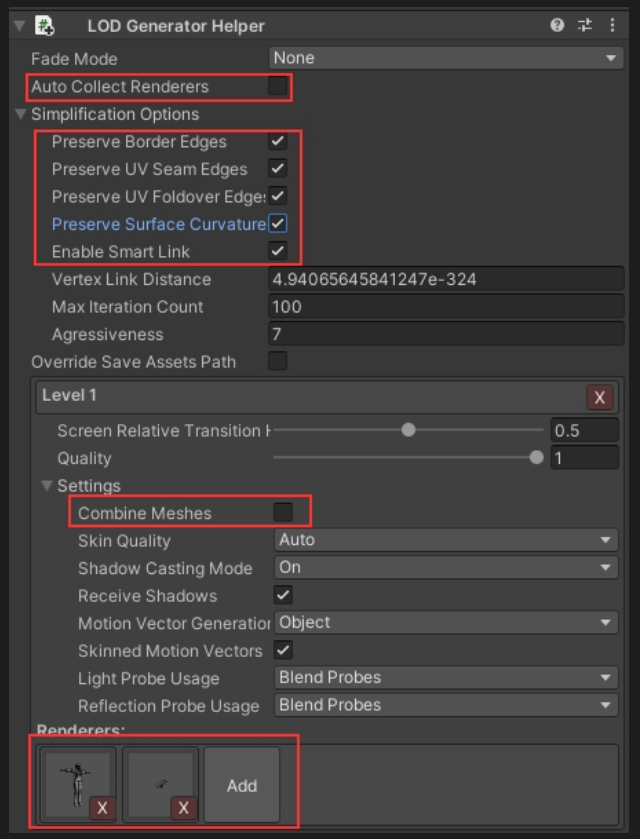


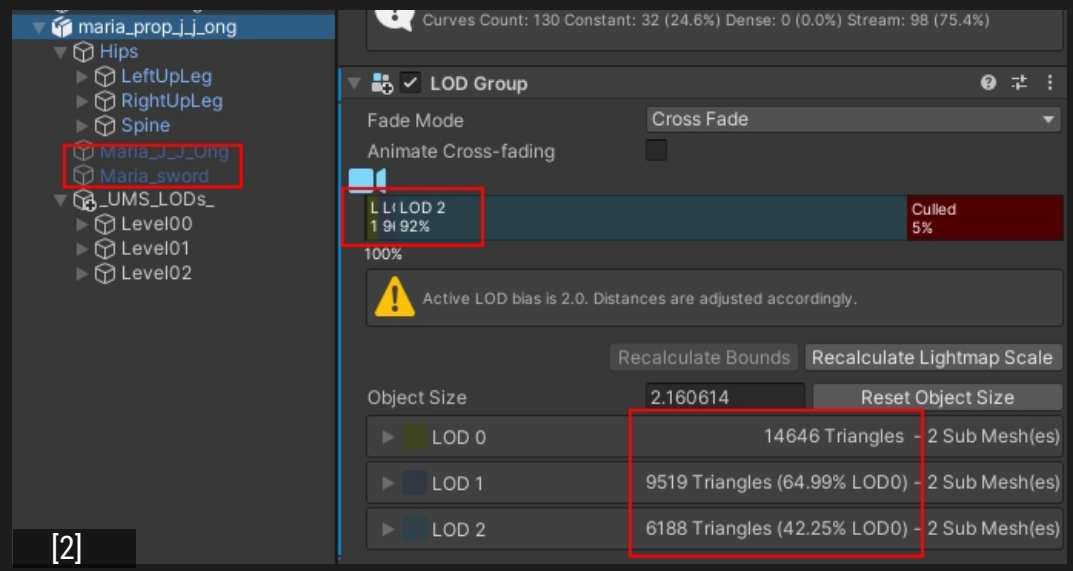
UnityMeshSimplifier

Generate LODs

- Drag the original fbx shoe in the hierarchy
- Set AssetPath to Assets/3D/ShoesScan/scarpa_lowFBX.fbx
- Click Simplify: you'll find simplified mesh in Assets/MeshSimplifierResults
- Update the shoe mesh field with the simplified one

- Drag Maria avatar (with the sword) in the hierarchy, and add the Combo Animator
- Don't import the FBX with OptimizeGameObjects flag
- Add LodGeneratorHelper component to the avatar root
- This is a complex setup: Skinned renderer with additional props
  - Set AutoCollectRenderers to OFF
  - For each level
    - add Maria_J_J_Ong and Maria_Sword Renderers to each layer
    - Turn OFF Combine Meshes
  - Set Quality levels 1, 0.3, 0.1
  - With this setup, the LOD generator could generate holes [1]: the used algorithm don't support situations where multiple vertices shared the same position, and this would then end up creating visible holes in the mesh
    - Try to enable all Preserve features in the script. This will strongly limiting the decimation algorithm, but should prevent holes in most situations
- Click on Generate LODs
- Disable the 2 FBX skinned mesh renderers: now the skinned Mesh renderers will be under _UMS_LODs_, and you can check the Triangle count resoult in the LOD Group component [2]
- Press Play: Skinning Animations should be preserved, and now we have a LOD group!

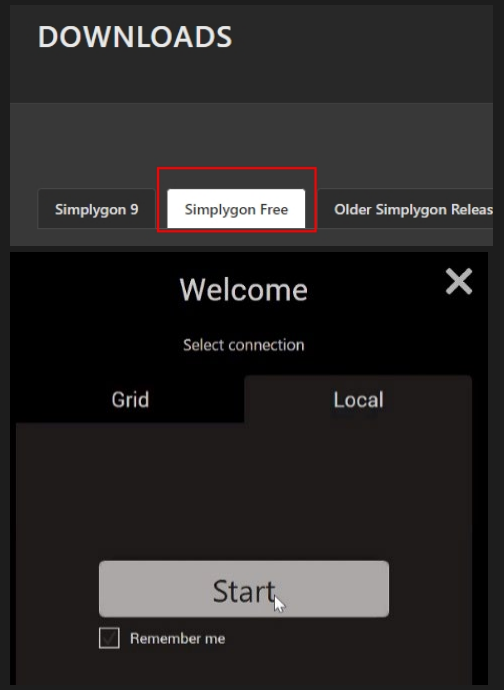


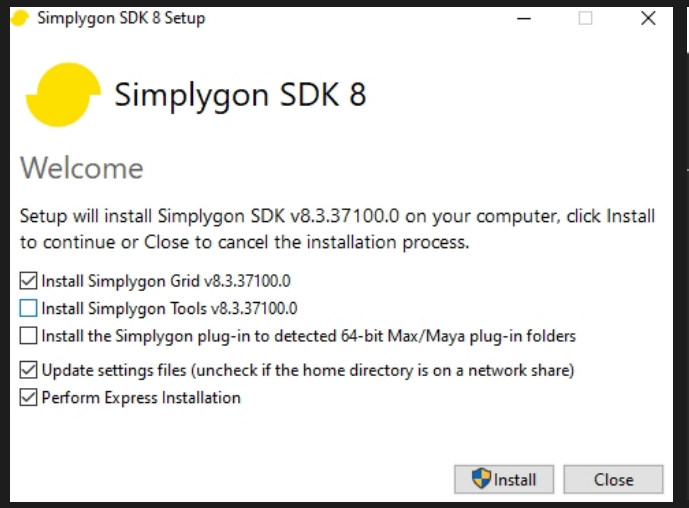


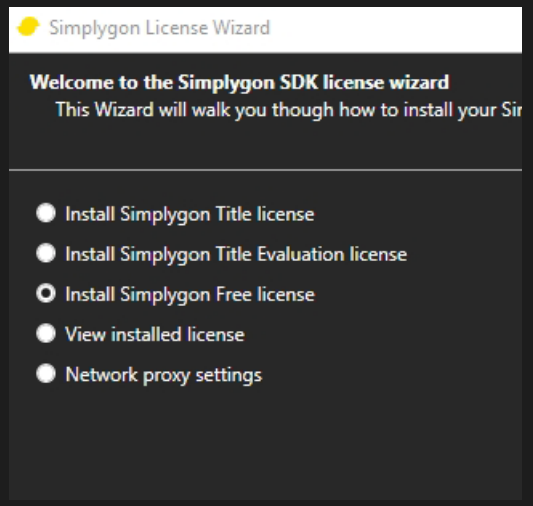
Simplygon

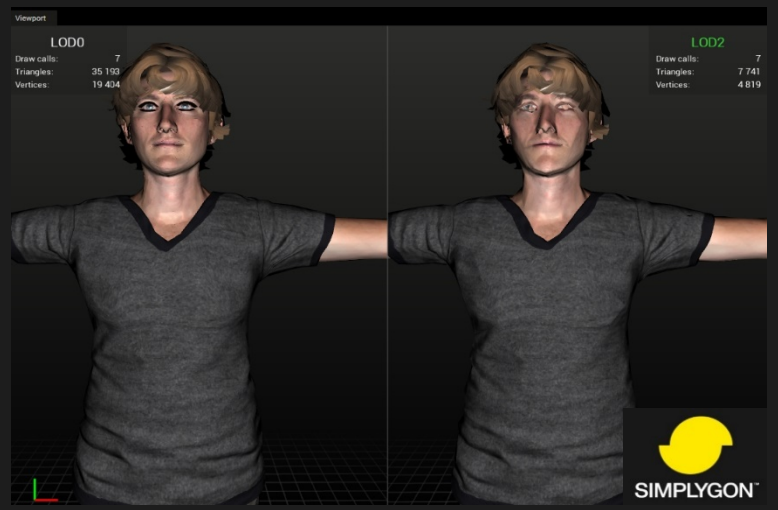
- https://www.simplygon.com/
  - Downloads/SimplygonFree/Download SDK
- Install & licensing: https://youtu.be/GHnDi6H6244
  - Install SDK tools
  - Custom-install: disable all the plugins you don't want to install, install Simplygon Grid service and tools
  - Once installed, activate free licence from Simplygon service in the taskbar (you'll need a Microsoft Account)
  - Lunch Simplygon UI and select "connection Local"

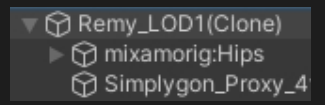
- Remeshing
  - LOD based on pixels on screen
  - 1 single mesh
- Reduction
  - LOD based on
    - Pixels on screen
    - Triangles
    - Vertices





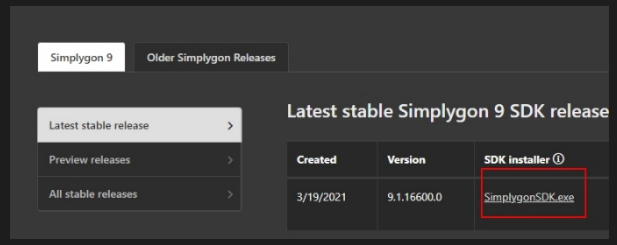


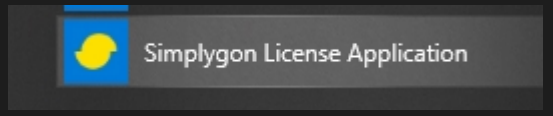


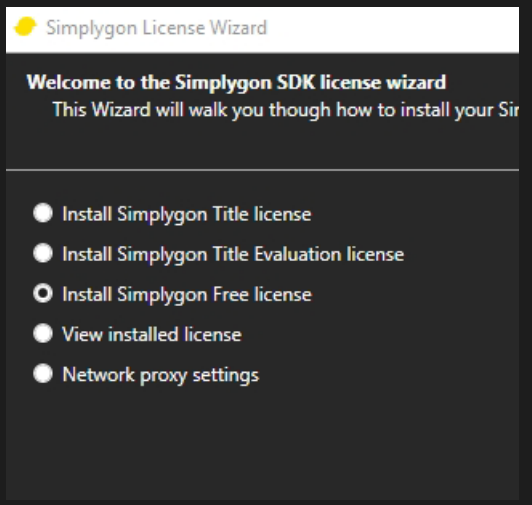


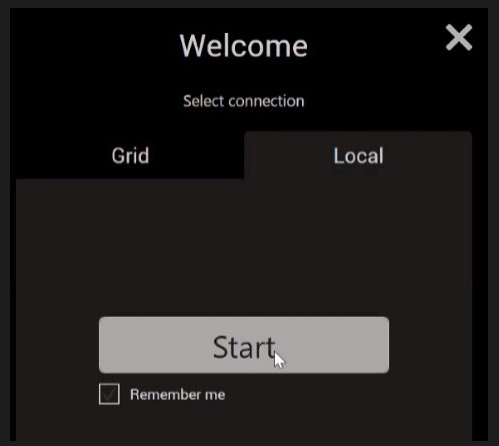
Simplygon

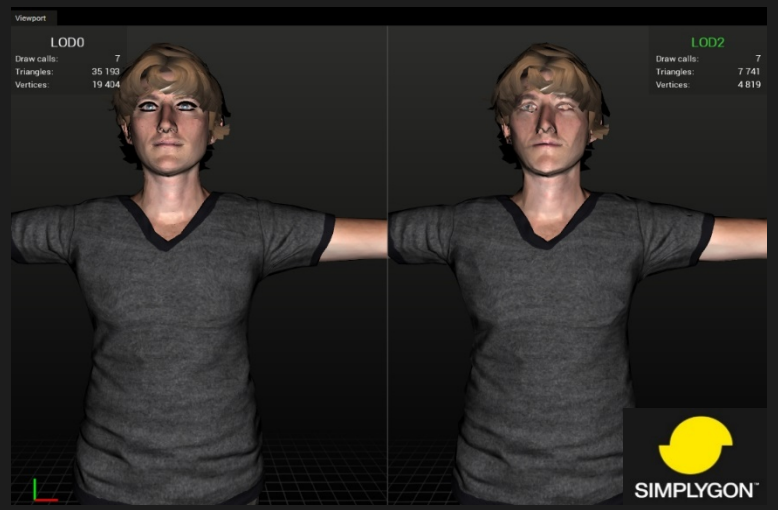
- https://www.simplygon.com/
- Downloads/SimplygonFree/Download SDK
- Install SDK tools
- Custom-install: disable all the plugins you don't want to install, install Simplygon Grid service and tools
- Open Simplygon License Application and choose Free
- Install Unity plugin copying the dll in the asset folder: https://documentation.simplygon.com/SimplygonSDK_9.1.16600.0/unity/gettingstarted/installation.html#simplygon-unity-plugin

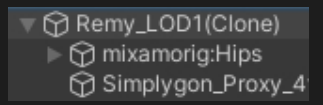






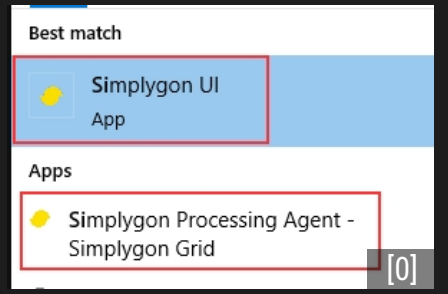


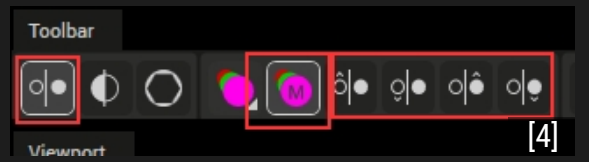


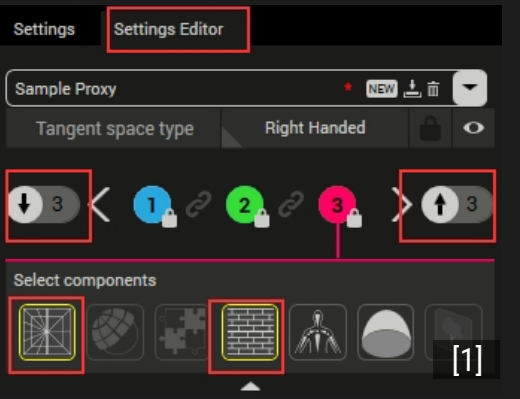


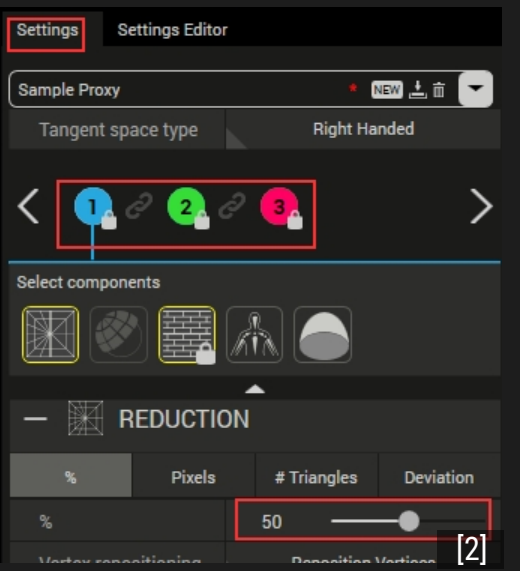
Simplygon

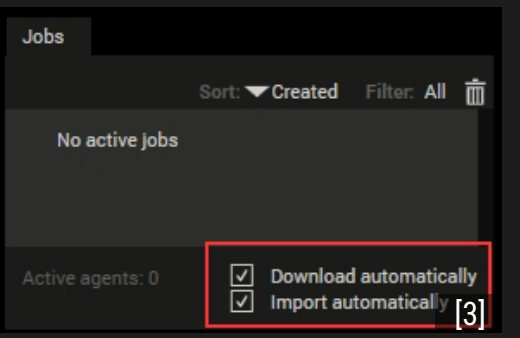
- [0] Start Simplygon processing Agent
  - Check if you have the Simplygon icon on the system tray tasks
- [0] Start Simplygon UI
- [1] Under Settings Editor
  - choose min/max-number of LOD-settings
    - ShoesScan
      - Remove SettingsEditor LOD1 to disable MaterialBaking component
      - Add the 2nd LOD Level
  - Foreach LOD-settings, choose Reduction/Remeshing and if you want to bake the material
    - ShoesScan
      - Reduction (No MaterialBaking)
- [2] Under Settings Tab
  - Foreach LOD level, choose how much triangles you want to keep (e.g. 50 / 25 / 10)
- [3] Under Jobs Tab
  - Enable Download automatically and Import automatically
- Click on ClickToProcess
- [4] Once the Job is completed, you should be able to see LOD comparison using
  - SideBySide LOD comparison
  - Manual LOD Switching
  - Increase/Decrease LODs
- File/Export/Scene
- Choose FBX file format











Animations LOD

- When you want LODs for animations, you must manually set them up via masking
- E.g. you have a human character model which does not animate fingers in lower LODs

Try it

- Create one Avatar Mask Mask_OnlyFingers without the rest of the hand or body
- Create another Mask_WithoutFingers and add the rest of the body
- Create an animator with
  - Base layer: no animations
  - WOFinger layer, Mask_WithoutFingers, Typing animation, Override
  - Finger layer, Mask_OnlyFingers , Typing animation, Additive
  - WOFinger layer will have weight 1 when this animation must be executed
  - Finger layer will have weight 1 when we want a higher animation LOD, with finger detail

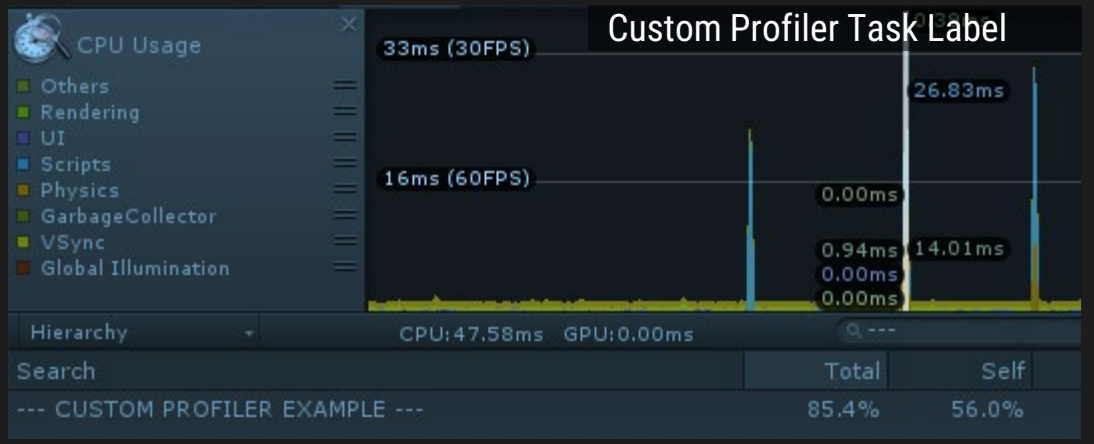
- This setup doesn't read all the animation's curves, but it makes sure Unity only loads the masks which are needed. Using LODs in Animation layers also saves CPU time, because animations do not evaluate with zero weights on the animation clip

[AnimationsLOD_01]

Scripting

- Profiling scripting [ProfilerBeginEnd]
  - Profiler.BeginSample("--- CUSTOM PROFILER EXAMPLE ---");
  - Profiler.EndSample();
- Custom Timer [CustomTimer]
  - Use a Mono framework System.Diagnostics.Stopwatch class
    - If testing memory access, keep in mind that repeatedly requesting the same blocks in a single test will likely use fast cache memory
    - Avoid tests in Awake() or Start() methods
    - Better to test your task N times, and calculate an average

NB: In C#, the using keyword has two purposes: The first is the using directive, which is used to import namespaces at the top of a code file. The second is the using statement. using statements ensure that classes that implement the IDisposable interface call their Dispose() method. It guarantees that the dispose method will be called, even if the code throws an exception. This guarantee is essential when the disposable object references lockable or finite resources like files and network connections. You don't want your code locking resources up indefinitely because it blew up while trying to use that resource

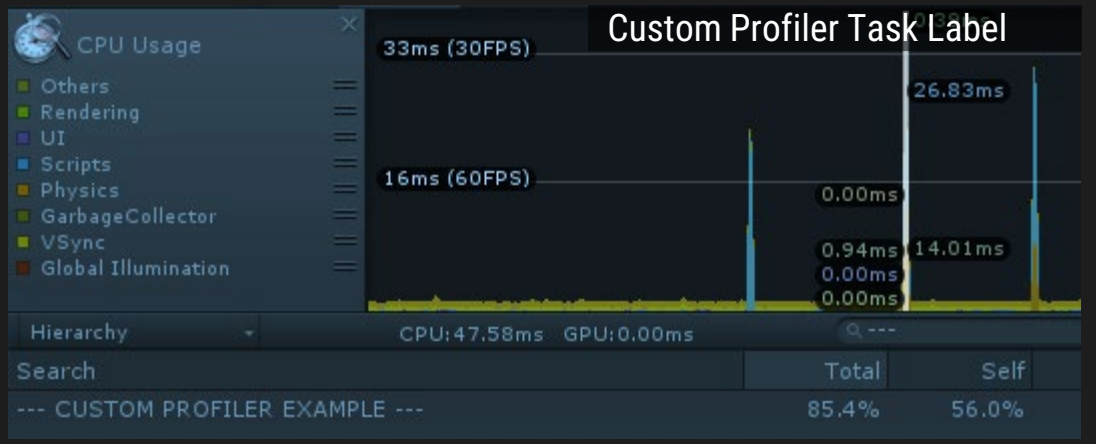


[Profiler_Scripting_03]

Scripting

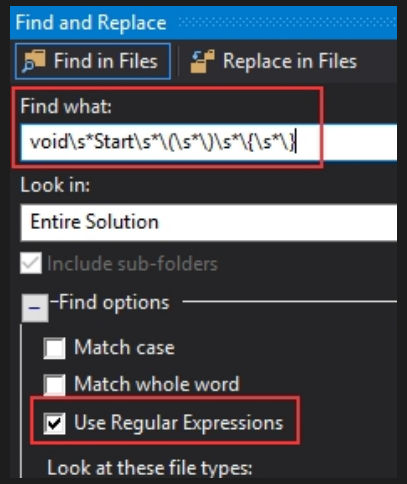
- Obtain components [ObtainComponents]
  - GetComponent(“componentName”)
  - GetComponent<componentName>()
  - GetComponent(typeof(componentName))
  - Different Unity versions have different optimizations
    - Don’t use in production-level application
  - Camera.main calls Object.FindObjectWithTag(“MainCamera”) every single time you access it
    - Optimized in Unity 2020.x

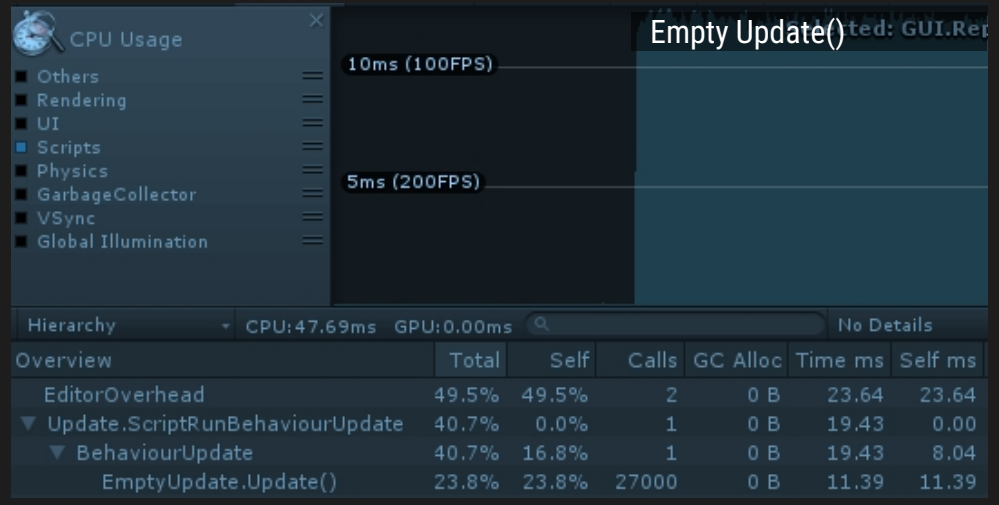
[Profiler_Scripting_03]



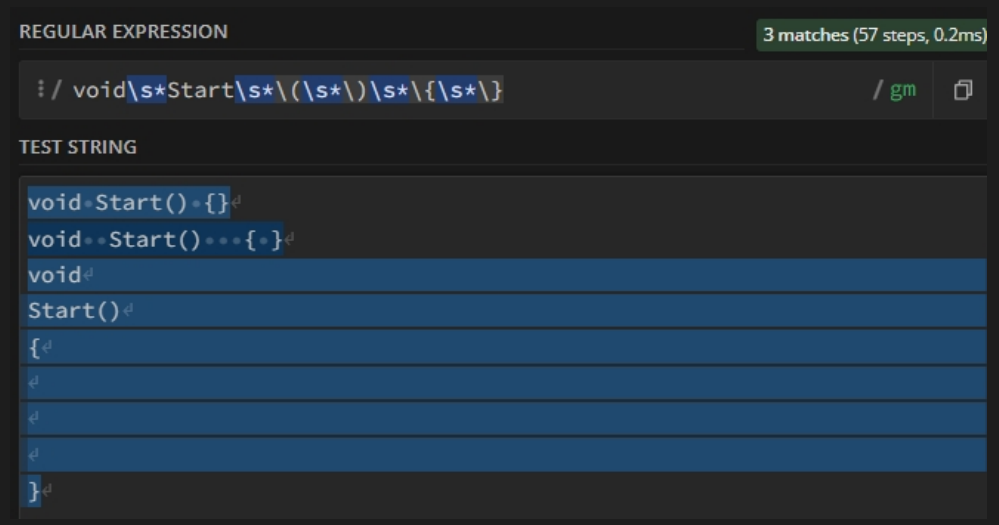
Scripting

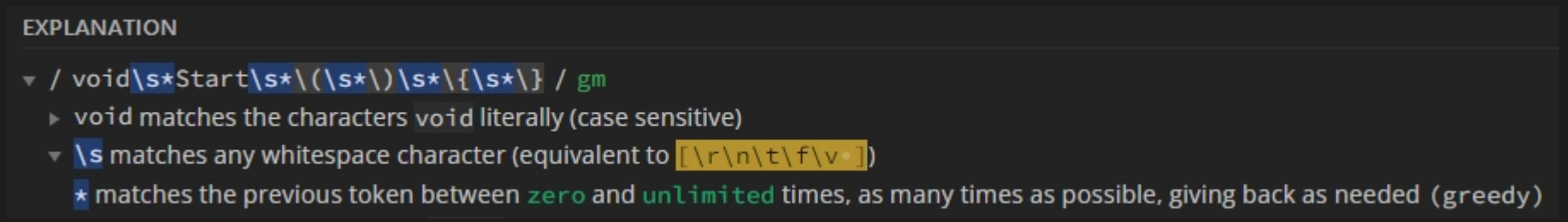
- Avoid empty callback [EmptyCallback]
  - Activate EmpyCallback and see the difference in CPU Profiler
  - Regex to search for empty Start/Update callbacks:
    - void\s*Start\s*\(\s*\)\s*\{\s*\}
    - void\s*Update\s*\(\s*\)\s*\{\s*\}
  - Test it on [regex101.com]





[Profiler_Scripting_03]



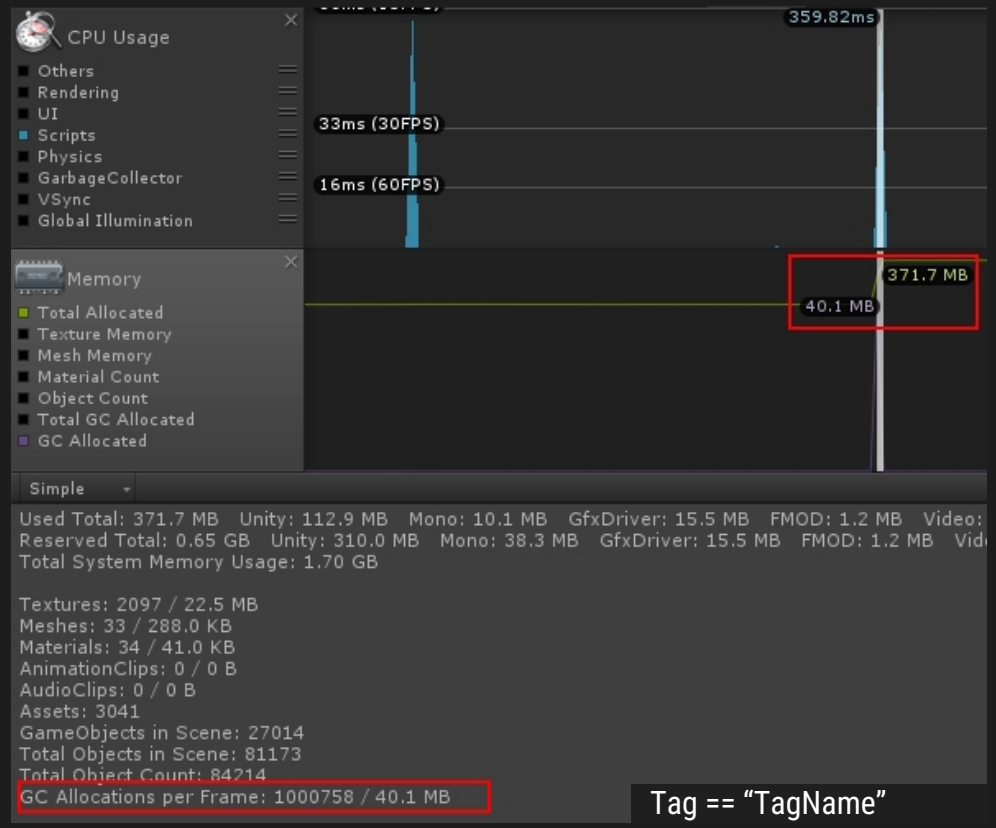


Scripting

- Share calculation output, don't waste time to recalculate the same result
- GameObject, Monobehaviour are not typical C# Objects, they have 2 representations in memory: C#side and C++side
  - Each time data moves between these parts GarbageCollector performs some additional operations
  - Faster GameObject null reference checks [NullCheck]
    - Instead of if(gameObject != null) use
    - If(!System.Object.ReferenceEquals(goToTest, null))
  - Retrieve string property from GameObject [CompareTag]
    - Instead of GO2Test.tag == "MainCamera"
      - 1M tag comparison = about 40MB CG Allocated memory: search for CPU/GetTag.OnGUI() task
    - use GO2Test.CompareTag("MainCamera")
      - No new Allocated Memory

[Profiler_Scripting_03]

Empty Update()



Tag == "TagName"

Reference-checking

- Null objects are not a real null
- Unity objects live in 2 places at once
  - We write in C#
  - But the references are pointer to native C/C++ side
- Unity overloads the == operator
  - A C# object is null if its underlying object is dead… even if the C# object isn't really null yet
  - == does a C++ check (overloaded operation)
- How and when to avoid ==
  - If you don't Destroy an object
    - Obj = newGameObject()
    - Obj = GetComponent<>()
  - Use (they check only the C# side)
    - Obj is null
    - ReferenceEquals(Obj, null)
- If == is expensive… can I avoid it? Yes, but we need to know that if we test null with is or ReferenceEquals against Destroyed objects… they will return NOT null!

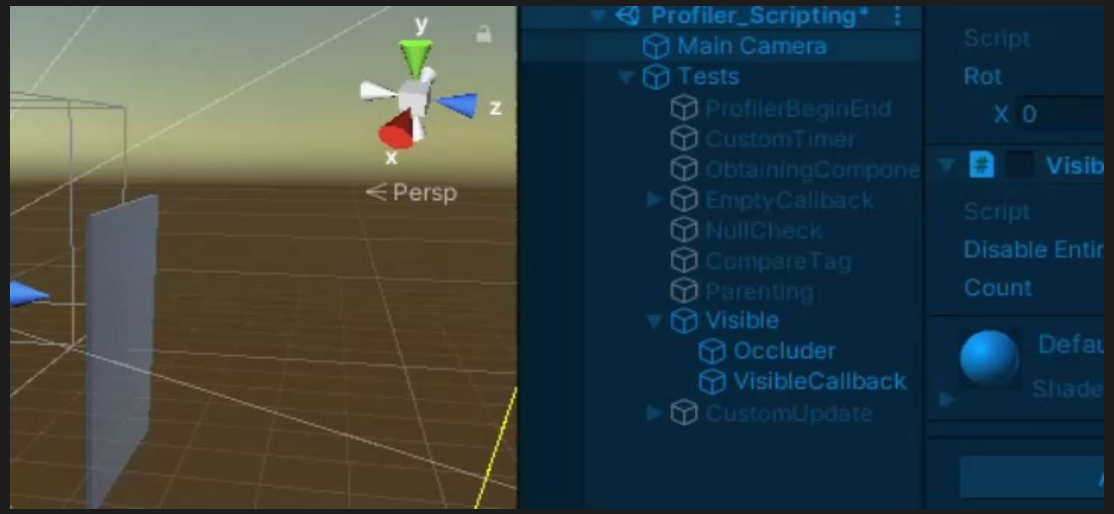
[ReferenceTest.cs]

Scripting

- Avoid re-parenting Transforms at runtime, it will trigger these operations [Parenting]
  1. Fit the new child within its pre-allocated memory buffer
     - If there isn't enough pre-allocated space to fit the new child, then it must expand its buffer
  2. Sort all of these Transforms based on the new depth
- Use Instantiate() parent parameter if needed

[Profiler_Scripting_03]

Scripting



- Open Occlusion culling window if you want to test this!

- Use OnBecameVisible/Invisible() to disable AI/logic/UI scripting [Visible]
  - Needs [Skinned]MeshRenderer component attached
  - Called when the Obj is inside/outside the camera Frustum
    - If there is more than one camera
      - if the GObj is outside all cameras VFrustum > OnBecameInvisible()
      - if at least one camera can see it > OnBecameVisible()
  - it works also with occlusion culling (keep occlusion culling window open to test it during playmode)

  - NB
    - If you disable a script, you are disabling Unity Update/FixedUpdate/LateUpdate/etc functions, not the other Unity Events function!
    - Disabling the GameObject will disable everything

[Profiler_Scripting_03]

Scripting

Use a custom Update layer

- 1K MonoBehaviours having the same behavior
- Coroutines every X seconds may trigger CPU spikes
  - Spread out coroutines call randomizing waiting time
  - Implement a custom Update layer: call the update on each obj from the outside
    - 1 Update() that calls 1K external functions is better than 1K Update()

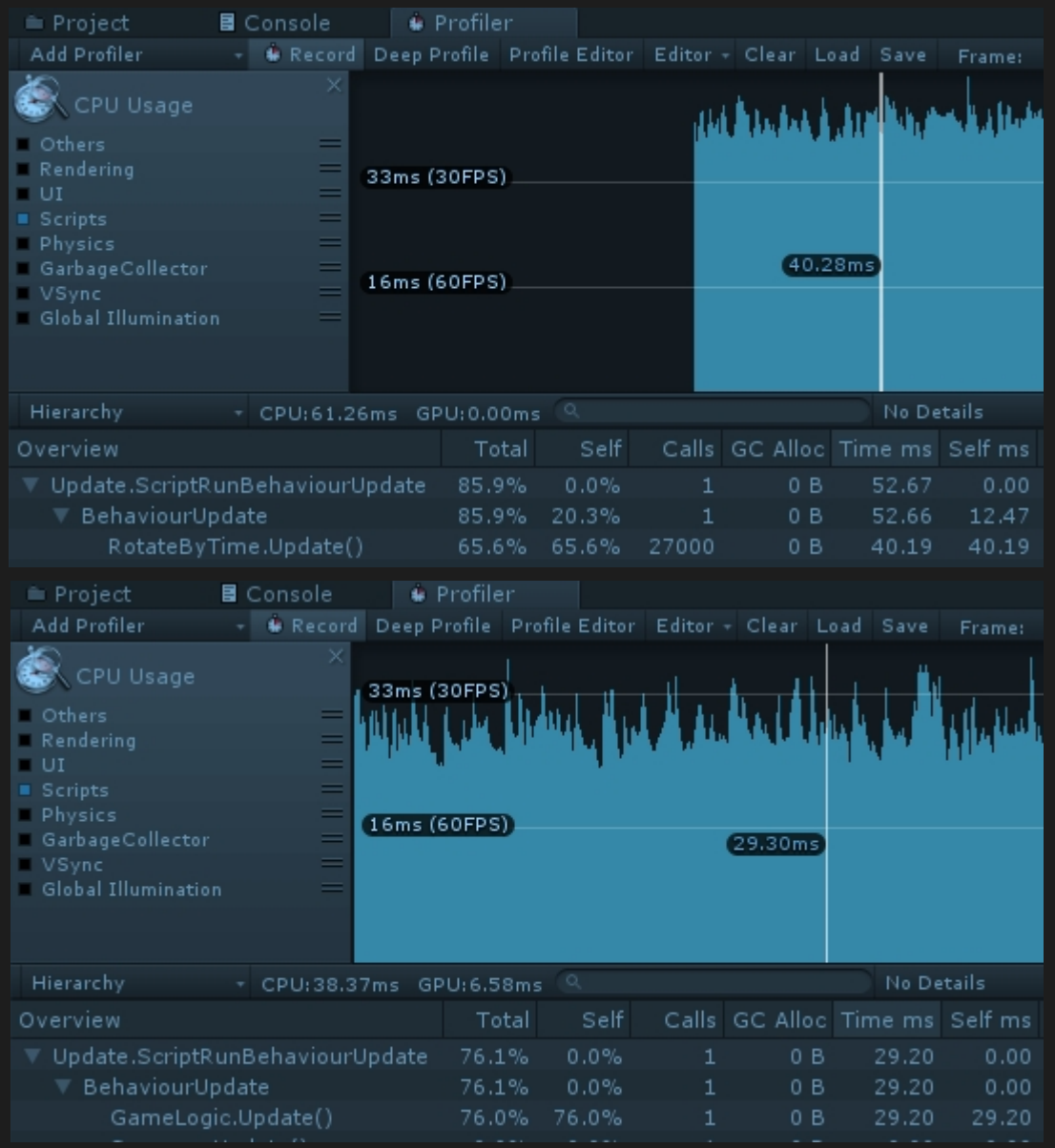
[ClassicUpdate]

- Search for RotateByTime.Update() task

[CustomUpdate]

- Search for GameLogic.Update() task

[IUpdateable.cs, GameLogic.cs, RotateByTime.cs, UpdateableComponent.cs]



Use the correct Data structures

[DataStructures]

- Mostly iterating
  - Array, List
- Add members
  - List, Dictionary, Hashset (like Dictionaries, but only with values, no keys, and no duplicates)
- Indexing/Search by key
  - Dictionary O(1)
- DuplicateChecks
  - Hashset, Dictionary (contains key = O(1))
  - ~~Don't use Array: O(n)~~

[ArrayListDictionary.cs]

i.e. Previous Custom update layer example, we'd like to have:

- Fast iteration (Array or List)
- Constant-time insertion (List, Dictionary, HashSet)
- Constant-time duplicate checks, to avoid register 2 times the same Iupdatable obj (Dictionary, HashSet)
- Solution? Use two data structures
  - List for iteration
  - Before changing the List, check on a HashSet
- Downside: Higher memory cost

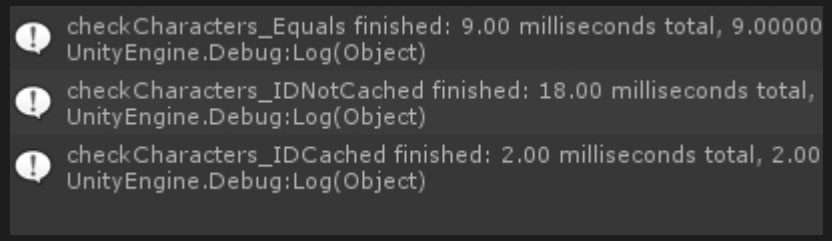
```
for_Array finished: 64.00 milliseconds total, 0.000 milliseconds per-test for 10000000 tests
UnityEngine.Debug:Log(Object)
for_List finished: 253.00 milliseconds total, 0.000 milliseconds per-test for 10000000 tests
UnityEngine.Debug:Log(Object)
foreach_List finished: 422.00 milliseconds total, 422.000 milliseconds per-test for 1 tests
UnityEngine.Debug:Log(Object)
for_Dictionary finished: 745.00 milliseconds total, 745.000 milliseconds per-test for 1 tests
UnityEngine.Debug:Log(Object)
foreach_Dictionary finished: 910.00 milliseconds total, 910.000 milliseconds per-test for 1 tests
UnityEngine.Debug:Log(Object)
contains_List finished: 218.00 milliseconds total, 0.000 milliseconds per-test for 50000000 tests
UnityEngine.Debug:Log(Object)
containsKey_Dictionary finished: 0.00 milliseconds total, 0.000 milliseconds per-test for 50000000 tests
UnityEngine.Debug:Log(Object)
containsValue_Dictionary finished: 328.00 milliseconds total, 0.000 milliseconds per-test for 50000000 tests
UnityEngine.Debug:Log(Object)
```

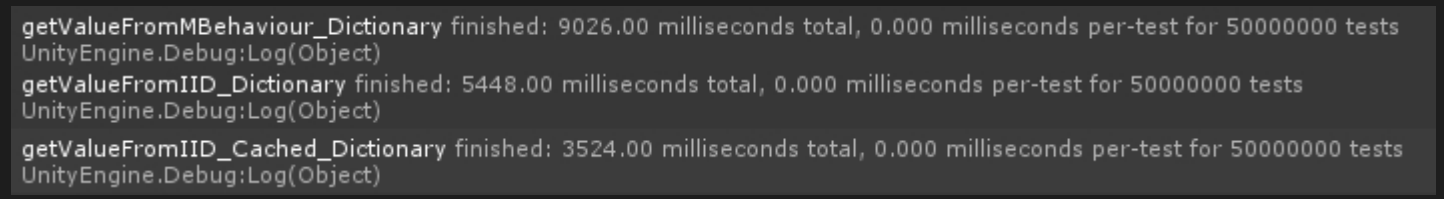
Use of InstanceIDs for Object comparison

- Objects comparisons can be slow
- We could use Object.GetInstanceID() to know if we are looking at the same object [ObjectComparison, InstanceIDs.cs]
  - The instance id of an object is always guaranteed to be unique
  - If not cached, call GetInstanceID() each time could be slower than the direct object comparison

- If you have to use MonoBehaviour or ScriptableObject as Dictionary Keys => Use their instanceIDs as Key [ArrayListDictionary , ArrayListDictionary.cs]
  - Dictionaries that are indexed thousands of times per frame

Try it
- getValueFromMBehaviour_Dictionary takes more time than getValueFromIID_Dictionary
  - Calculate hash value from a MonoBehaviour is more expensive than Calculate hash value from an int
- getValueFromIID_Dictionary takes more time than getValueFromIID_Cached_Dictionary
  - No need to recalculate InstanceID every time





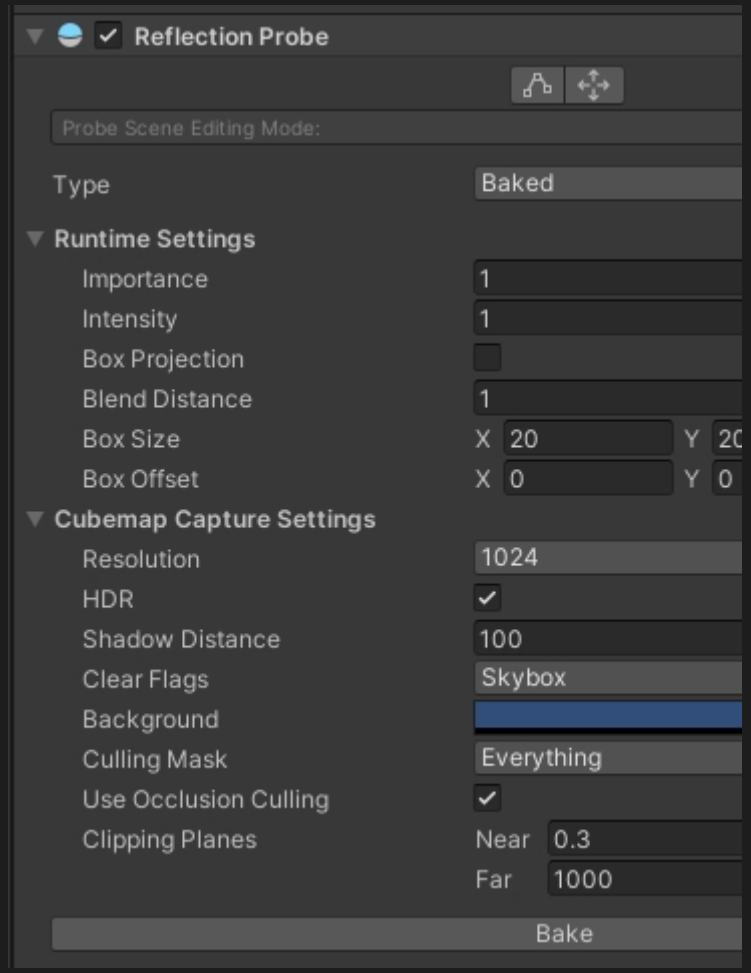
Reflection probes

Use shaders that don’t support Reflection probes, if it isn’t needed

- URP/SimpleLit (No RP)
- URP/Lit (RP)

If RunTime RP, treat it like a camera

- Clear flag: if possible, solid color
- Shadow distance
- Clipping planes



Shader Stripping

When should you use it?

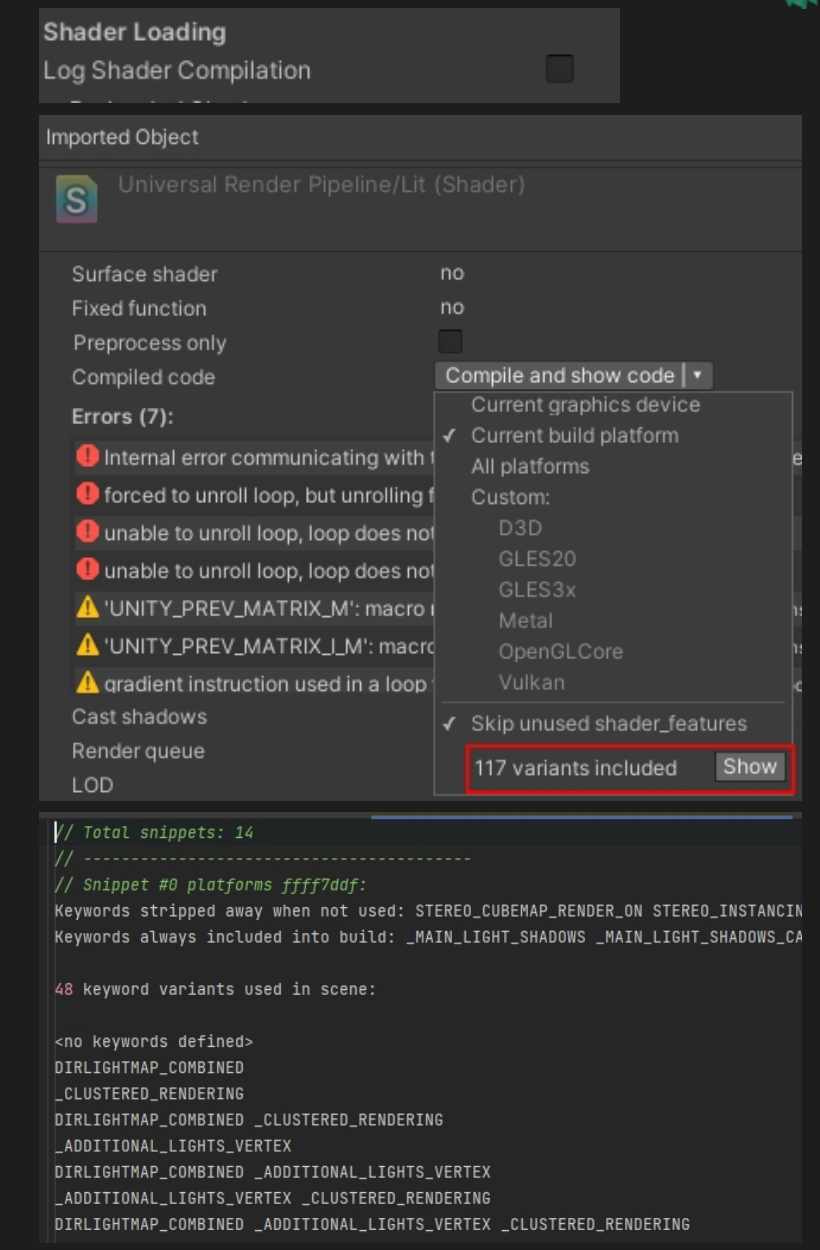
- Compile times
- Loading times
  - Even if a shader variant is built by Unity, it is done in an intermediate language: then the final hardware during loading needs to perform a final shader variant build
- Too much RAM spent on shaders
- Too many expensive draw calls due to shader variants

Why do this manually?

- Automatically stripping does not always work (Unity doesn't know if your' going to change a material at runtime)
- If you are going to use one stripped away variant, Unity is going to build that variant at runtime

How to see shader variants that Unity is preparing for you

- Select the shader and then ShowVariants
- Settings/Graphics/LogShaderCompilation



Shader Stripping

How

- Find the shader variants you are not using
- Remove them and test visuals on a build
- Using Script API, implementing IPreprocesShaders
  - OnProcessShader() callback is able to remove the shader with the keyword(s) that we want to remove!

```
using System.Collections.Generic;
using UnityEditor.Build;
using UnityEditor.Rendering;
using UnityEngine;
using UnityEngine.Rendering;

class MyCustomBuildProcessor : IPreprocessShaders
{
    ShaderKeyword m_GlobalKeywordBlue;

    public MyCustomBuildProcessor()
    {
        m_GlobalKeywordBlue = new ShaderKeyword("_BLUE");
    }

    public int callbackOrder { get { return 0; } }

    public void OnProcessShader(Shader shader, ShaderSnippetData snippet, IList<ShaderCompilerData> data)
    {
        ShaderKeyword localKeywordRed = new ShaderKeyword(shader, "_RED");
        for (int i = data.Count - 1; i >= 0; --i)
        {
            if (!data[i].shaderKeywordSet.IsEnabled(m_GlobalKeywordBlue))
                continue;
            if (!data[i].shaderKeywordSet.IsEnabled(localKeywordRed))
                continue;

            data.RemoveAt(i);
        }
    }
}
```

Memory domains

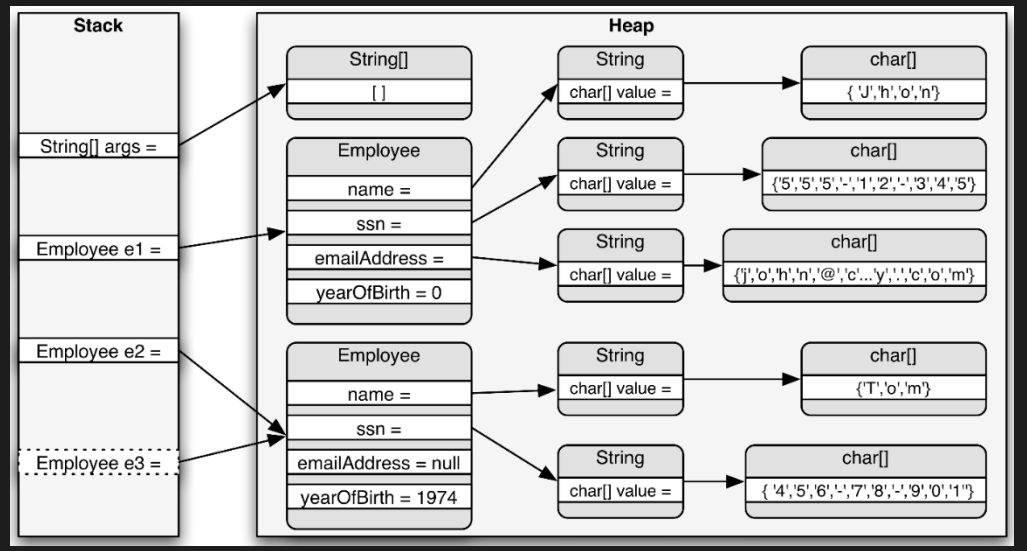
- Managed Domain
  - Where the Mono platform does its work
  - C# Scripts
  - Memory is automatically managed by GC
  - Includes wrappers for Native Domain GO Components
- Native Domain
  - It is in Unity Native code side (written in C++)
  - Allocates
    - Asset data (texture, audio, mesh)
    - Subsystems memory (Physics, Input, Rendering pipeline)
    - GameObjects Components (Transform, RigidBody)
      - Managed Domain includes wrappers for the same GO Components
      - Cross the Native-Managed bridge: Interaction with Transform in M.D. = Access N.D. representation > perform calculations > copy it back to the M.D.
        - when we interact with Components such as Transform, most instructions will ask Unity to dive into its Native Code, generate the result there, and then copy it back to the Managed Domain for us
- External libraries Domain (OpenGL, DirectX, plugins)
  - Referencing these libraries from our C# code will cause a similar memory context switch and subsequent cost

Stack/Heap

Stack
- Order of MBytes
- Small data values
- Local vars, load/unload function calls

Heap
- Every data too big to fit in the Stack or must persist outside the function call

- Native Code
  - Memory allocations handled manually > Memory leaks
- Managed Languages
  - Memory allocations handled by GC
  - Mono requests at start a few MBytes to allocate Heap, which will grow and shrink if GC determines that the data is no longer needed
- Profiler
  - Unity Memory tracked by allocations in native Unity code
  - Mono heap size used by managed code and GC
  - GfxDriver memory the driver is using on Textures, Shaders and Mesh
  - Audio Audio driver's estimated memory usage
  - Profiler Memory used for the Profiler data



Value/Reference types

- Value types
  - Primitives
  - Structures
  - Allocated either on the Stack or the Heap
  - Contains all bits of data stored

- Reference types
  - Classes
  - Arrays
  - Strings
  - Always allocated on the Heap
  - Contains a pointer (4 or 8 Bytes) to the data

- Temporary Value type within a class method => Stack [1]
- If a Value type is a Class member, it will be stored within the Reference type of the Class => Will be allocated in the Heap [2]
- if we put the integer into a normal C# class, then the rules for Reference types still apply and the object is allocated on the heap [3]
  - DoSomething() creates a new reference to dataObj, hence dataObj will not be removed once TestFunction() finishes
- We should try to replace Heap allocations with stack allocations where possible

```
public class TestComponent {
  void TestFunction() {
    int data = 5; // allocated on the stack
    DoSomething(data);
  } // integer is deallocated from the stack here
}
```
[1]

```
public class TestComponent : MonoBehaviour {
  private int _data = 5;
  void TestFunction() {
    DoSomething(_data);
  }
}
```
[2]

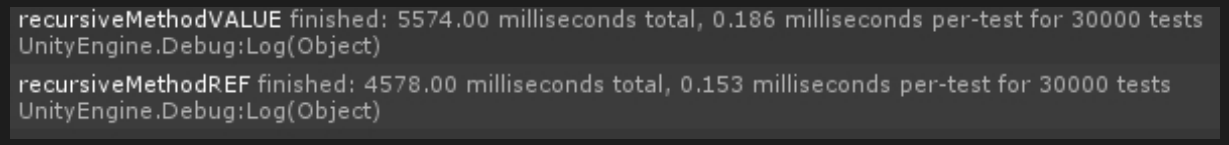
```
public class TestData {
  public int data = 5;
}
public class TestComponent {
  private TestData _testDataObj;

  void TestFunction() {
    TestData dataObj = new TestData(); // allocated on the heap
    DoSomething(dataObj     );
  }
  void DoSomething (TestData dataObj) {
    _testDataObj = dataObj; // a new reference created! The referenced
    // object will now be marked during Mark-and-Sweep
  }
}
```
[3]

Pass by Value/Reference

- passing by value
  - We're passing the object's data
- passing by reference
  - When we're copying a reference to something else
  - Any changes to the data will change the original
  - Can be forced using ref keyword

- 4 data passing situations
  - Value type by value
  - Value type by reference
  - Reference type by value
  - Reference type by reference (e.g. if you pass a string with ref keyword)

- A Value type contains the full bits of data stored => Pass it by Value means all of the data will be copied => Could be more costly than just using a Reference type and letting the GC take care of it



Struct = Value types

- Struct are ValueTypes
- If we are using a class only to pass data, use a struct => Avoid heap allocation
  - Class DamageResult in [1] should be changed with a struct [2]
- [3] _memberStruct is a Value Type, hence is allocated in the Stack or in the Heap. In this case it is allocated in the Heap (because it is a global variable)

```
public class DamageResult {
  public Character attacker;
  public Character defender;
  public int totalDamageDealt;
  public DamageType damageType;
  public int damageBlocked;
  // etc.
}
```
[1]

```
public struct DamageResult {
  // ...
}
```
[2]

```
public struct DataStruct {
  public int val;
}

public class StructHolder {
  public DataStruct _memberStruct;
  public void StoreStruct(DataStruct ds) {
      _memberStruct = ds;
  }
}
```
[3]

Arrays = Reference types

- Arrays are ReferenceTypes [ArraysReference]

- [1] TestStruct[1000] is allocated in the Heap
- [2] There is only 1 TestStruct at time allocated in the Stack

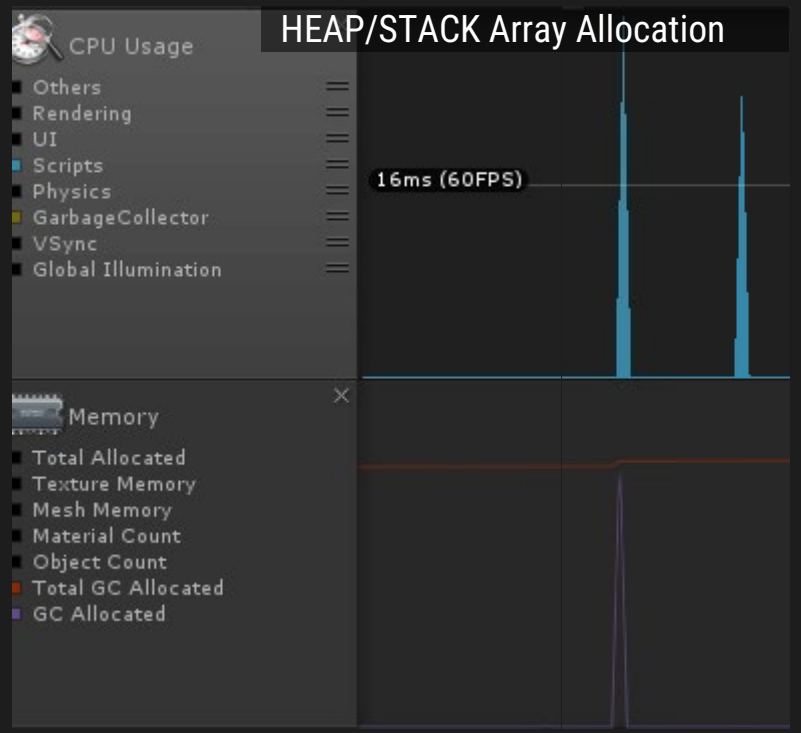
- Try it
  - Click on IMGUI buttons manageCharacters_HEAP and manageCharacters_STACK and see CGAllocated memory

```
TestStruct[] dataObj = new TestStruct[1000];

for(int i = 0; i < 1000; ++i) {
  dataObj[i].data = i;
  DoSomething(dataObj[i]);
}
```

[1]

```
for(int i = 0; i < 1000; ++i) {
  TestStruct dataObj = new TestStruct();
  dataObj.data = i;
  DoSomething(dataObj);
}
```

[2]



Strings = Reference types

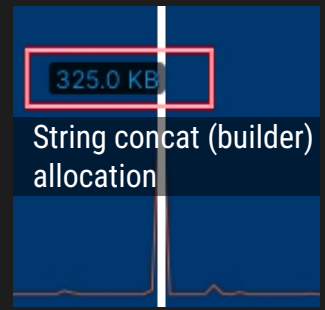
[StringsRef]

- Strings are immutable ReferenceTypes
  - Different results if we pass them by value or reference

[StringsConcatenation, StringConcat_End.cs]

- Concatenation
  - +, or += will result in a new Heap allocation; only a single pair of strings will be merged at a time, and it allocates a new string object each time
- StringBuilder
  - Allocates an appropriate buffer ahead of time, saving undue allocations
  - Works like a Char dynamic array
  - StringBuilder sb = new StringBuilder(n); //Prepare a dynamic n-char array
  - sb.Append("string to append");
  - Debug.Log(sb.toString());
- The difference is huge: 286MB vs 325KB

- Try it
  - Each character is 2 Bytes
  - Starting from StringConcat.cs, try to calculate the amount of Bytes allocated in bytesAllocated
  - You should obtain 286MB



Use the correct Data structures

- Mostly iterating
  - Array, List
- Add members
  - List, Dictionary, Hashset (like Dictionaries, but only with values, no keys)
- Indexing/Search by key
  - Dictionary O(1)
- DuplicateChecks
  - Hashset, Dictionary (contains key = O(1))
  - ~~Don't use Array: O(n)!~~

[ArrayListDictionary.cs]

i.e. Previous Custom update layer example, we'd like to have:

- Fast iteration (Array or List)
- Constant-time insertion (List, Dictionary, HashSet)
- Constant-time duplicate checks, to avoid register 2 times the same Iupdatable obj (Dictionary, HashSet)
- Solution? Use two data structures
  - List for iteration
  - Before changing the List, check on a HashSet
- Downside: Higher memory cost

```
for_Array finished: 64.00 milliseconds total, 0.000 milliseconds per-test for 10000000 tests
UnityEngine.Debug:Log(Object)
for_List finished: 253.00 milliseconds total, 0.000 milliseconds per-test for 10000000 tests
UnityEngine.Debug:Log(Object)
foreach_List finished: 422.00 milliseconds total, 422.000 milliseconds per-test for 1 tests
UnityEngine.Debug:Log(Object)
for_Dictionary finished: 745.00 milliseconds total, 745.000 milliseconds per-test for 1 tests
UnityEngine.Debug:Log(Object)
foreach_Dictionary finished: 910.00 milliseconds total, 910.000 milliseconds per-test for 1 tests
UnityEngine.Debug:Log(Object)
contains_List finished: 218.00 milliseconds total, 0.000 milliseconds per-test for 50000000 tests
UnityEngine.Debug:Log(Object)
containsKey_Dictionary finished: 0.00 milliseconds total, 0.000 milliseconds per-test for 50000000 tests
UnityEngine.Debug:Log(Object)
containsValue_Dictionary finished: 328.00 milliseconds total, 0.000 milliseconds per-test for 50000000 tests
UnityEngine.Debug:Log(Object)
```

Unity API Array return type

- GetComponents<T>(); //T[]
- Mesh.vertices; //Vector3[]
- Camera.allCameras //Camera[]

- If a Unity method returns an array, it will be a new Heap allocation
  - Try to cache this kind of results as much as possible

- ParticleSystem.GetParticles(Particle[] particles); //Avoid new Heap allocation each time

RaycastAll vs RaycastNotAlloc

- RaycastNotAlloc is like Physics.RaycastAll, but generates no garbage
- Physics.RaycastAll creates and returns an array of RaycastHit structs. This creates garbage on the heap
- RaycastNonAlloc does the same but instead of returning a new array each time it's called, you have pass a preallocated array to the method and it just fills in the elements. The method just returns an integer that tells you how many hits has been filled in. The array should be large enough to hold all possible hits. The great thing is you can reuse the same array each time you call the method

- RaycastNonAlloc can't be compared to Raycast as it does something different. They should have called it “RaycastAllNonAlloc“

Garbage collector

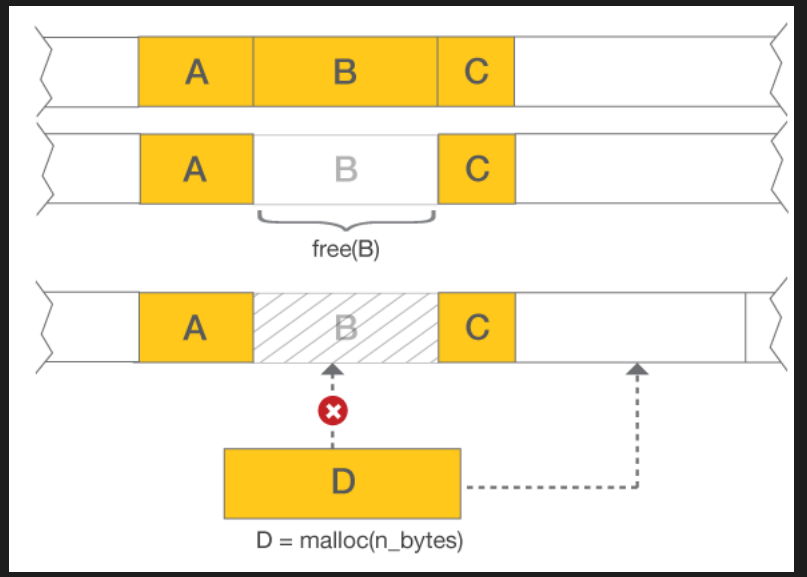
If a large part of the frame time on a slow frame is taken up by garbage collection, this could indicate that we have a problem with excessive garbage collection. We can dig into the profiling data to confirm this

- Profiler/CPU Timeline should display GC.Collect

- https://docs.unity3d.com/Manual/BestPracticeUnderstandingPerformanceInUnity4-1.html

Garbage Collection

- Ensures that
  - We don’t use more Managed Heap mem than we need
  - Mem that is no longer needed will be auto deallocated
- Objs are rarely deallocated in the same order they were allocated, and they don’t have the same size in memory
  - Memory Fragmentation



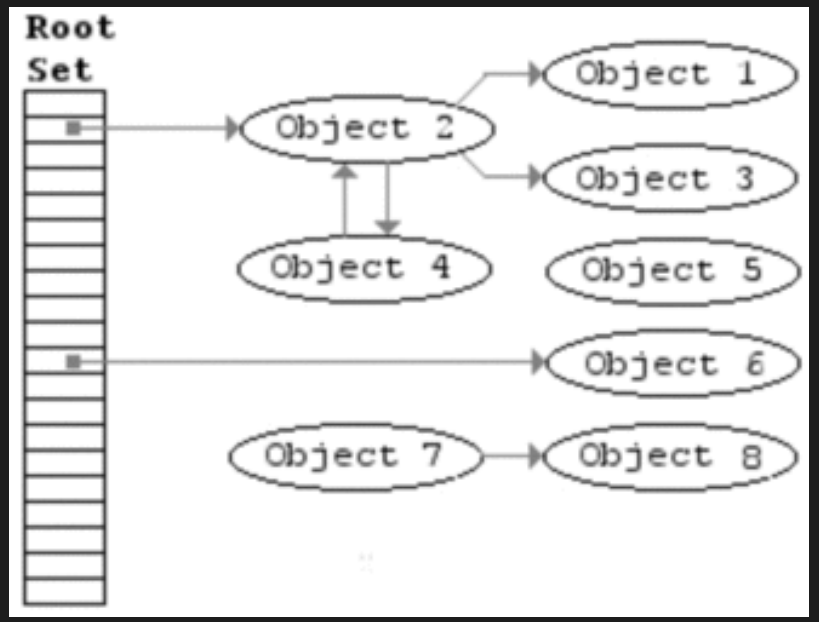
Garbage Collection

- Unity uses a type of Tracing Garbage Collector
  - Mark and Sweep strategy

New mem allocation

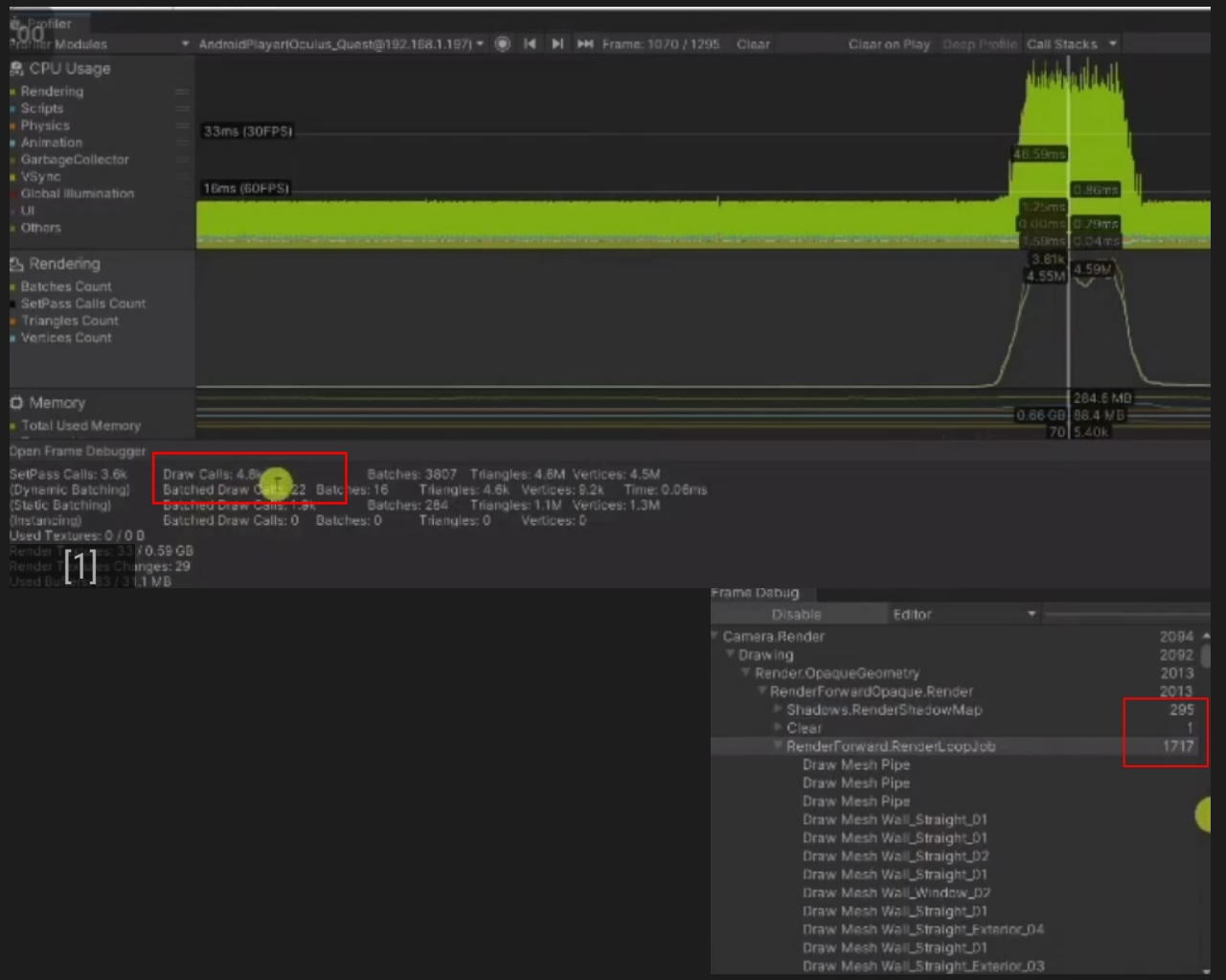
1. Is there enough contiguous space?
2. If not, iterate through all known direct and indirect references, marking everything they connect to as reachable
3. Iterate through all of these references again, flagging unmarked objects for deallocation
4. Iterate through all flagged objects to check whether deallocating some of them would create enough contiguous space for the new object
5. If not, request a new mem block to expand the heap
6. Allocate the new object at the front of the newly allocated block and return it to the caller.

- GC workload scales poorly as the allocated heap space grows
- GC runs on 2 separate threads
  - Main thread flags MH mem blocks for deallocation
  - Finalize thread perform a lazy deallocation



Practical example

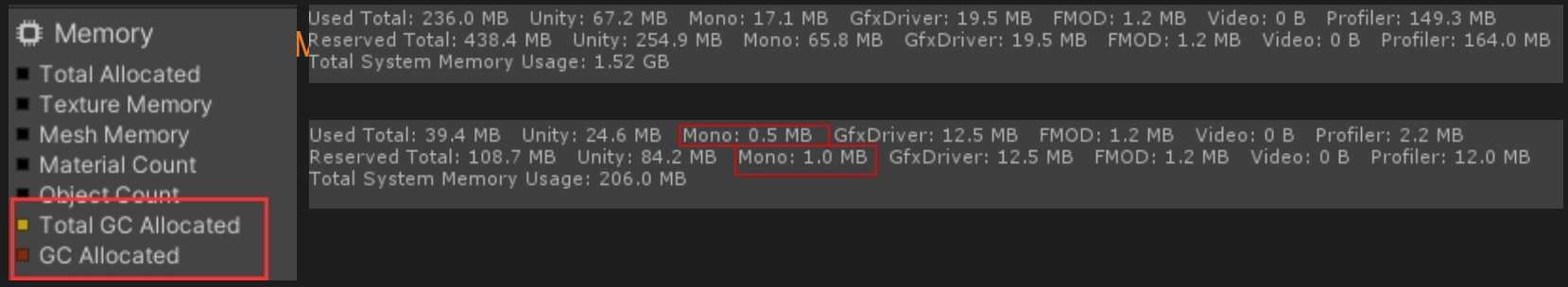
- Play the profiler with Android
- The project for this is: 3D Beginner Complete Project, PackageManager
- Batches for Quest 2 < 200. To see DrawCalls number (batches) go to profiler /Rendering section [1]
- Profiler doesn't say much information about where drawcalls are spent, use Frame debugger. You'llsee that >200 are spent on shadows
- > 10K are spent in opaque section, and you can check why are not batched together: different forward lights for example



Memory Profiling

- Memory consumption in Editor Mode is very different from Stand-alone version [MemAllocation]
  - Profiler.GetRuntimeMemorySizeLong(Object o); //Bytes allocated for Object o
  - Profiler.GetMonoUsedSizeLong(); //HeapMemory used
  - Profiler.GetMonoHeapSizeLong(); //HeapMemory reserved
- Profiler filters
  - CPU Area – Search for GC*
  - Memory Area – Total GC allocated, (instant) GC allocated
- CG Spike is not always related to current frame operations
- Manually invoke GC
  - System.GC.Collect(); //ManagedDomain (Mono in Profiler)
  - Resources.UnloadUnusedAssets(); //NativeDomain (Unity in Profiler)
  - Resources.UnloadAsset(Object assetToUnload); //NativeDomain (Unity in Profiler). Unload an asset on disk
  - Perform during
    - Loading between levels
    - Gameplay is paused
- Click SPAWN to instantiate MemAllocator prefabs
- Remove MemoryAllocatorSpawner GameObject manually
- CG acts after a few seconds, not immediatly

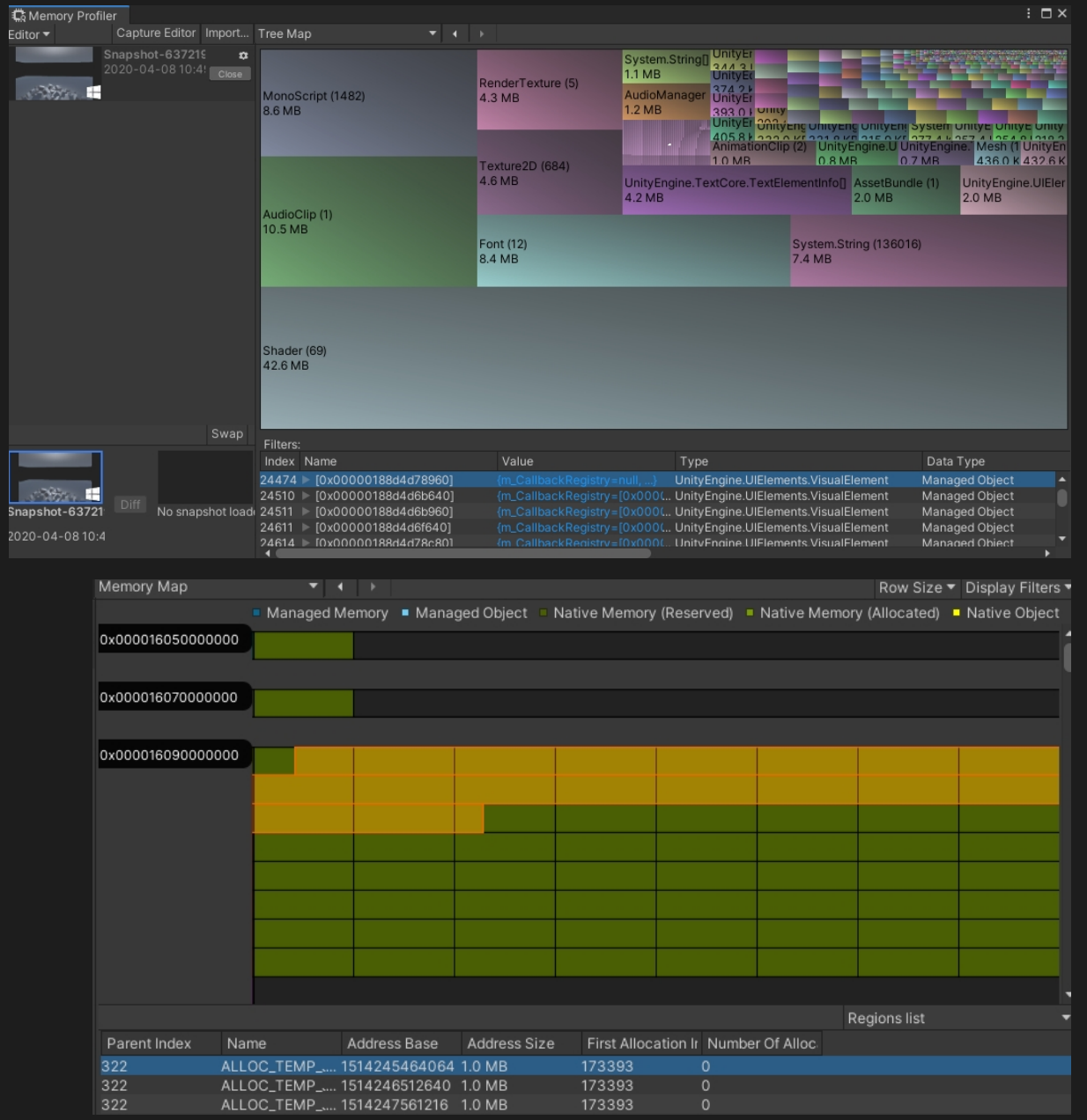
[Memory_00.scene, MemoryProfiler]



LEFT arrow: manually remove MemoryAllocatorSpawner GameObject
RIGHT arrow: CG free memory

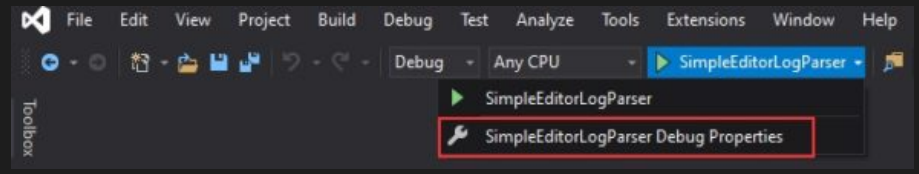
MemoryProfiler

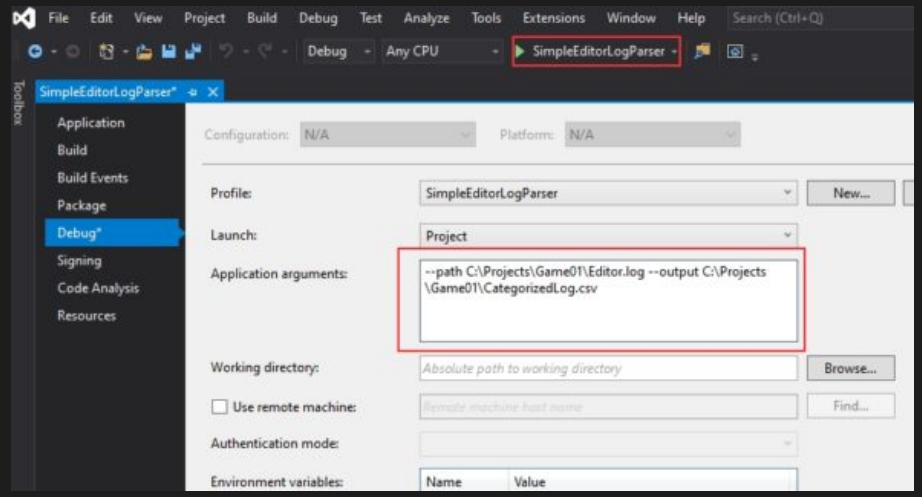
- Your memory structure is something that Unity hides from you for an excellent reason: Unity game developers rarely need to be concerned about this
- Install it from the PackageManager
- Your memory map not only includes assets content, but also: Variables in the heap, Networking data structures, Memory pools, etc..
- May change a lot if you profile on your desktop instead that on your mobile device
- During play, take a snapshot of the memory
- TreeMap is useful for quickly analyzing which objects take up the most memory. You can navigate through elements pressing F/A
- MemoryMap
  - VirtualAddress current location of memory area
  - MemoryBlocks
  - Details
    - The ObjectsList view is the most useful for us
- Very advanced. Could be useful to:
  - Analyze memory fragmentation (memory-intensive games that need further optimization)
  - Analyze differences in memory layout between different game sessions
    - You need to take 2 memory snapshots and click Diff

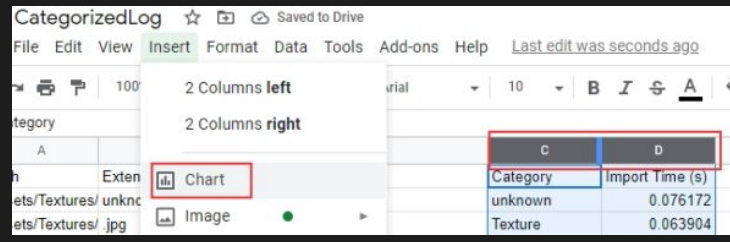


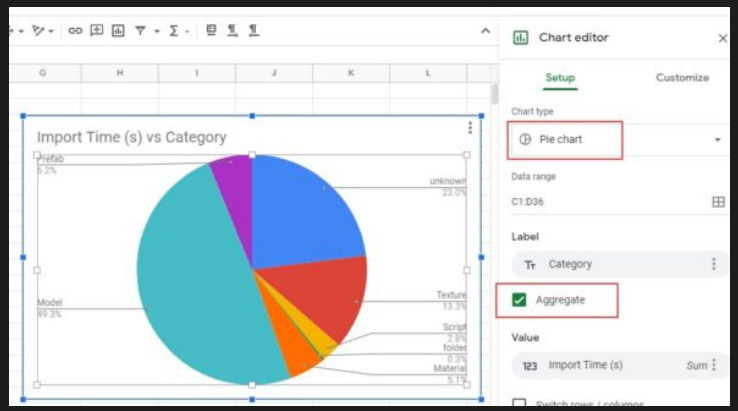
Simple Editor Log Parser

- What happens when you're creating a new project or opening a project where your asset Library isn't already present? Unity starts importing your assets. And it takes time. If we'd like to measure this import time, we need to analyse Editor.log file
- Download from https://github.com/Unity-Javier/SimpleEditorLogParser
- Open C:\Users\[username]\AppData\Local\Unity\Editor\Editor.log with a text editor. You can access it also via Console/UpRightcornerDotsIcon/OpenEditorLog)
- Open your Unity project /Asset folder
- Clear Editor.log content and in Explorer create a new /Tmp folder inside your /Asset folder. Copy inside it some assets (textures, meshes, materials, scripts)
- Make a copy of Editor.log file. Now you should have a bunch of Start importing log
- Open the SimpleEditorLogParser .sln project with VS2019
- If you're prompted with a warning saying that VS needs some extra module to be installed in order to open this project, do it (this will open VSInstaller automatically)
- In Debug command line settings add these arguments:
  --path C:\[pathToYourCopyOfTheEditorLog]\Editor.log --output \CategorizedLog.csv
- Save and Run the project: in the same folder, a cvs file should appear
- Upload this cvs into your google drive and open it as Google sheet file
- Select both Category and Import columns, and then insert a Chart
- Click on the created graph, and inside ChartEditor options select Aggregate and PieChart type. We now have our ImportTime vs Category Pie chart, ready to give use at a glance what we can do to improve import asset times









Performance analysis

- Verifying that the target script is present in the Scene / appears the correct number of times
  - Editor helpers
- Verifying the correct order of events
  - Edit/ProjectSettings/ScriptExecutionOrder

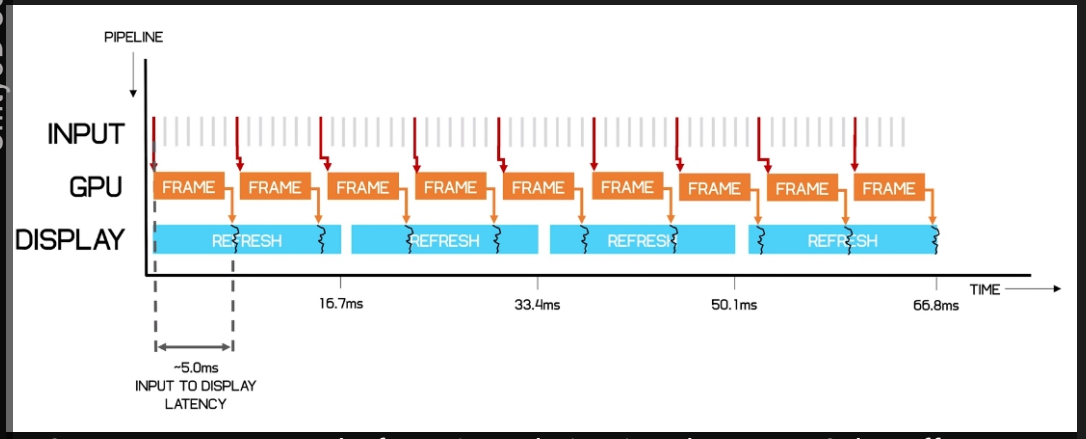
- Internal overhead
  - Vsynch used to match the application's frame rate to the frame rate of the monitor (e.g. 60 Hertz)
    - If a rendering loop is running faster, then it will switch to idle state
    - Reduces screen-tearing
    - Generates noise Spikes – WaitForTargetFPS task (CPU side) / Gfx.WaitForPresentOnGfxThread task (GPU side)
    - Edit/ProjectSettings/Quality to enable/disable
  - Logging

- External overhead
  - Double check for background processes eating CPU cycles

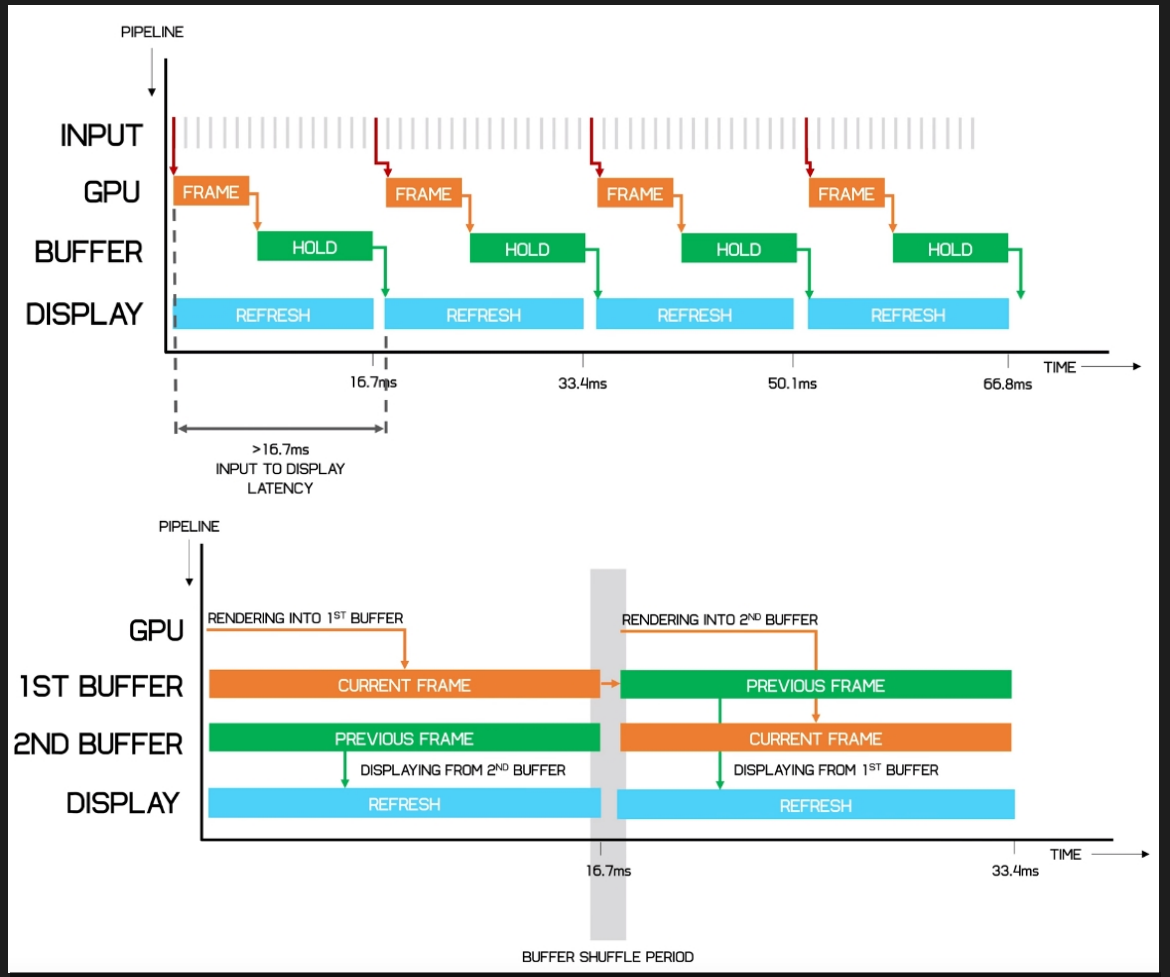
- To help you analyze your profiler thread tasks, use https://docs.unity3d.com/Manual/ProfilerCPU.html

VSync

- Running a game above your monitor maximum refresh rate does deliver a more responsive game experience



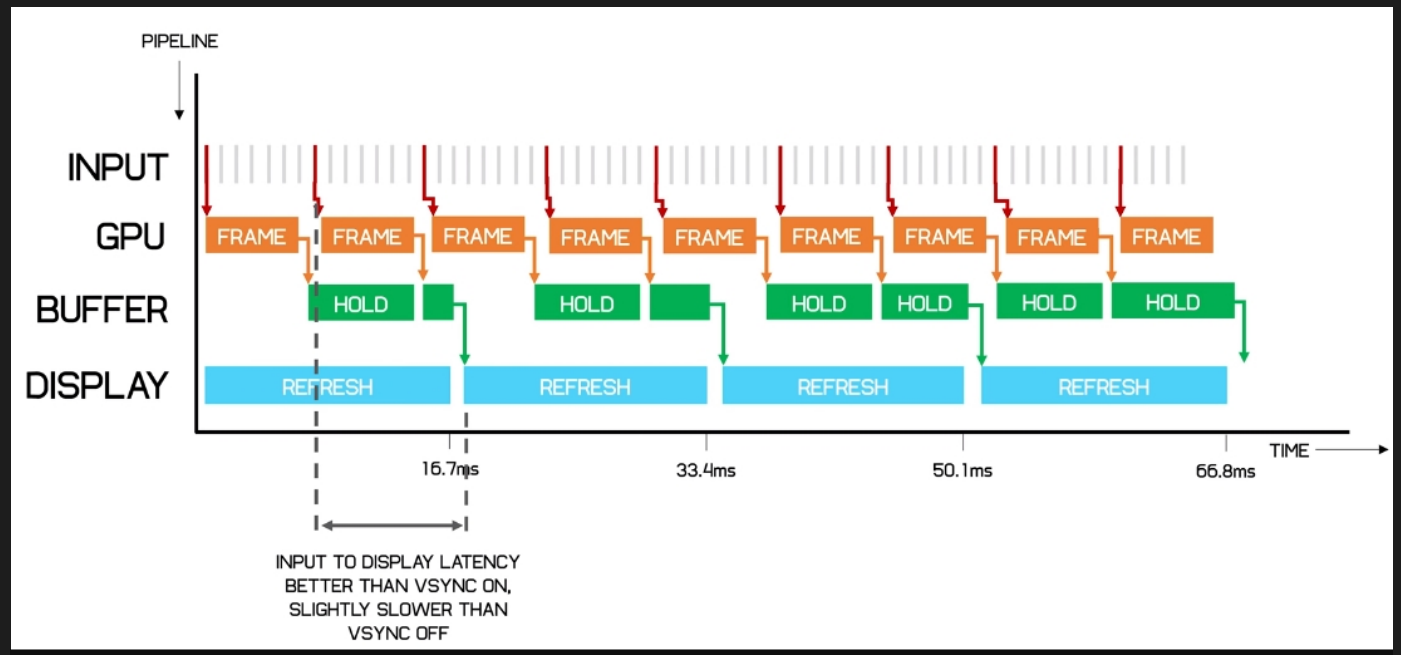
NO Vsync – As soon as the frame is ready, it writes the current ColorBuffer



Vsync – instead of the GPU sending its new frame immediately, it shuffles its frames into a second Color buffer (hold) – when Refresh happens, the previous Hold ColorBuffer is being shown on the monitor

VSync

- NVIDIA provide Fast sync: a combination of Vsync ON and OFF, producing Low Input latency and no Tearing
- The GPU keeps rendering frames so that when GPU access the ColorBuffer at the beginning of the refresh period, that frame has been rendered more closely to the refresh window
- Now we have 3 Color buffer: Front/Back/Last – the LastColorBuffer is shown on the next refresh



NVIDIA FastSync

UI

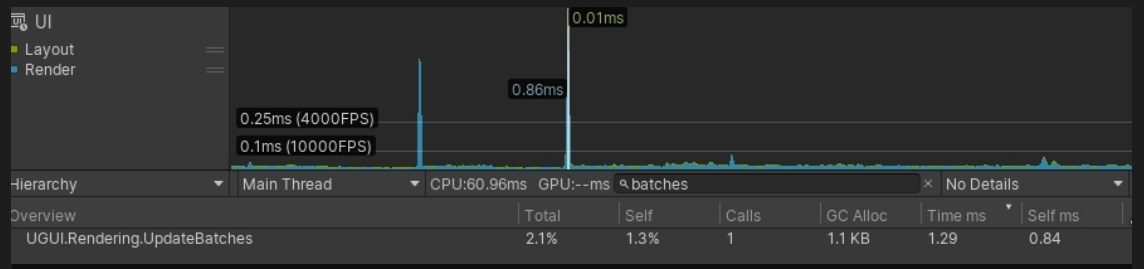
- UI geometry will always be drawn in the transparent queue > each pixel, even if it is fully covered by other polygons, will be drawn
  - High overdraw > fill-rate GPU problems on mobile
- Canvas primary task
  - Collect UI children elements, batch them together to reduce DrawCalls, generate the appropriate render commands to send to Unity's Graphics system
  - MultiThreaded operation: very different performances between Desktop/Mobile builds
- Unity tries to cache the canvas so the expensive layout process do not happen every frame, but.. As soon as a UI element is changed, the entire canvas is marked as Dirty, and requires Rebatching
  1. Recalculate Layout (top-bottom order)
  2. Mask clipping
  3. Rebuild graphical elements
- Causes: scaling, text change, positioning, color, image change, rotation, animations

UI

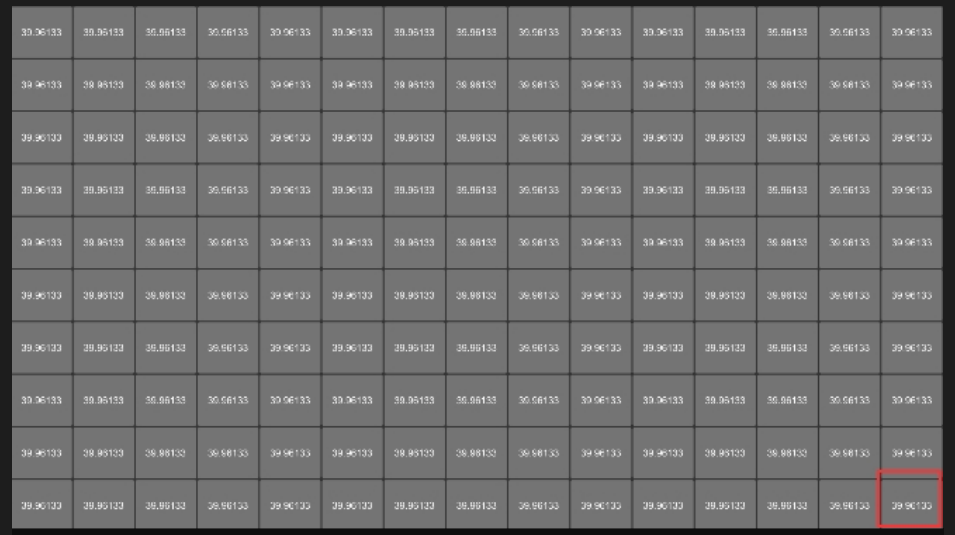
Try it

- Activate Canvas_single
- During Playmode, profiler: Search for Canvas.*
  - You'll find Canvas.RenderOverlays and Canvas.SendWillRenderCanvases (Invoke C# scripts that are subscribed to the Canvas component's willRenderCanvases)
    - willRenderCanvases is an Event that is called just before Canvas rendering happens. This allows you to delay processing / updating of canvas based elements until just before they are rendered
  - You'll see also CanvasUpdate.Layout task , taking almost zero-time. If you activate Gridlayout/GridLayoutGroup component, this will generate a spike: CanvasUpdate.Layout is calculated only once
- Activate the GridLayout children UI_Panel_UPDATER
  - You'll find Canvas.BuildBatch
  - See UI graph: there there is a spike in UI visualization each time you activate UI_Panel_UPDATER (both in CPU and in GPU panel)
  - now, if you activate Gridlayout/GridLayoutGroup component, you'll see also CanvasUpdate.Layout task taking a lot of time (up to 2ms!)
  - now Canvas.SendWillRenderCanvases is taking too much time
    - Dynamic elements have been grouped together with static elements and are forcing the entire Canvas to rebuild too frequently
- Canvases (or sub-canvases) are independent
  - Activate CanvasMulti
  - Activate CanvasMulti/Canvas_dynamic
    - See UI graph: there there is no spike in UI visualization
  - Now, if you activate Gridlayout/GridLayoutGroup component, you'll see also CanvasUpdate.Layout task taking zero time!

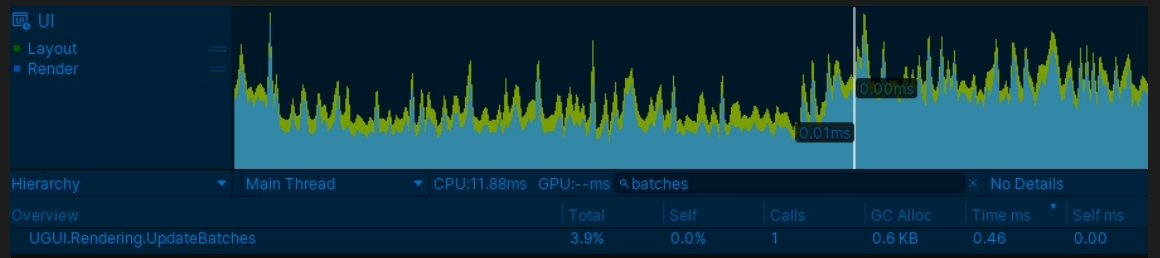
[UI.scene]



Every time you activate UI_Panel_UPDATER, there is a spike in UI visualization, caused by (CPU Task) UGUI.Rendering.UpdateBatches



In CanvasMulti, only the bottom right panel is under the Dynamic Canvas

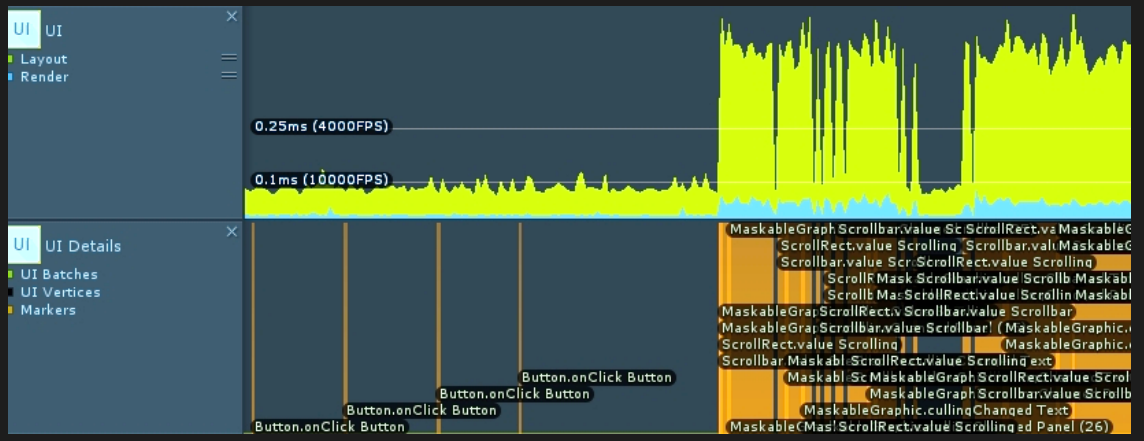


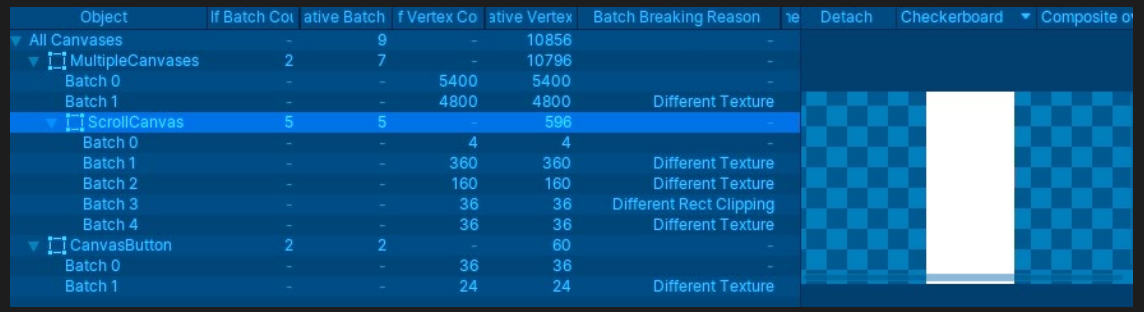
In CanvasMulti, UGUI.Rendering.UpdateBatches now causes no spike

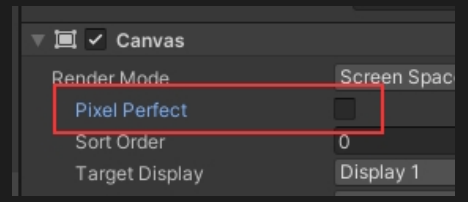
UI

Profiler [ScrollRect]

- UI shows time passed in UI layout calculation and render time
- UIDetails shows event markers, to help you determine what caused a CPU spike (scrolling culling, ButtonClicks, etc)
- BatchViewer
  - Batch Breaking Reason
  - Each batch has its associated objs visible by double-clicking it. Try to double-click on BatchN to highlight the objects in hierarchy







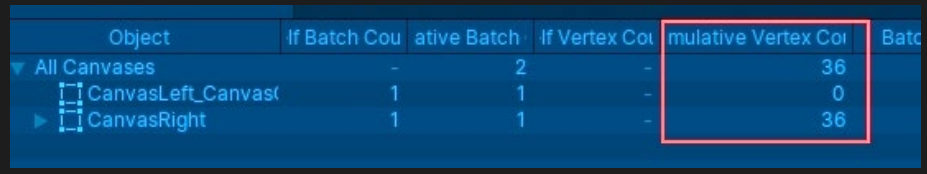
Try it

- Enable ScrollRect
- Disable UIDetails Profiler view (too much data to collect for the profiler)
- ScrollRect/SingleCanvas
  - Each time you scroll, CanvasUpdate.Layout takes a lot of time to recalculate the Grid layout
- ScrollRect/MultipleCanvases
  - uses one canvas (same size of the scrollRect)
  - Disables Canvas.PixelPerfect
    - Force elements in the canvas to be aligned with pixels. Only applies with renderMode is Screen Space. Can make elements appear sharper. On animated UI elements it may be advantageous to disable pixelPerfect, since the movement will be smoother without it
  - Each time you scroll, CanvasUpdate.Layout takes zero-time

UI

- If there are Canvas.BuildBatch and Canvas.UpdateBatches spikes
  - Excessive number of Canvas Renderer components on a single Canvas
  - Splitting Canvases
    - Split UI into 3 groups: Static, Discrete Dynamic (buttons), Continuous Dynamic (Animator component, text that is always changing, color pulse, etc)

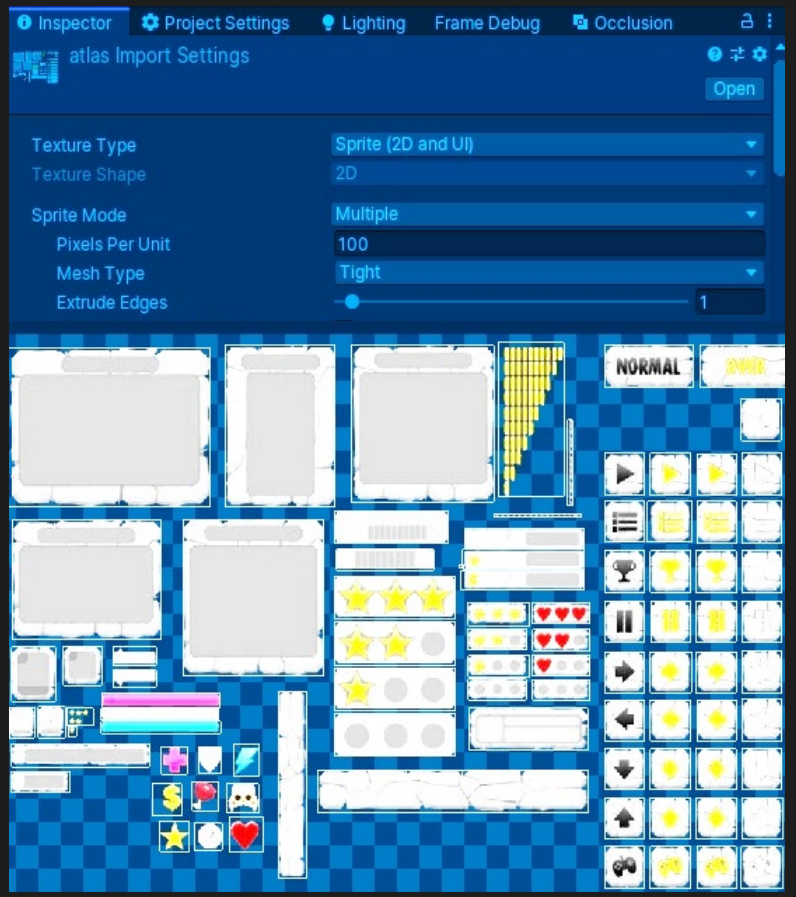
- If there are GPU Spikes > Fill-rate problem
  - How to eliminate invisible UI (from most to least efficient way) [Disable_UI_components]
    - Disabling the parent Canvas (or Sub-Canvas) Component
    - Use CanvasGroup.alpha property (alpha 0 will cull children objs)
      - UIElement.Alpha = 0 Still sends data to GPU!
    - UIElement.IsActive = False (Require a Canvas.BuildBatch)
    - Try it
      - Set CanvasLeft_CanvasGroup CanvasGroup.alpha to 0
        - CumulativeVertexCount: 0
      - Set CanvasRight/Panel.alpha to 0
        - CumulativeVertexCount: 36 (no changes)

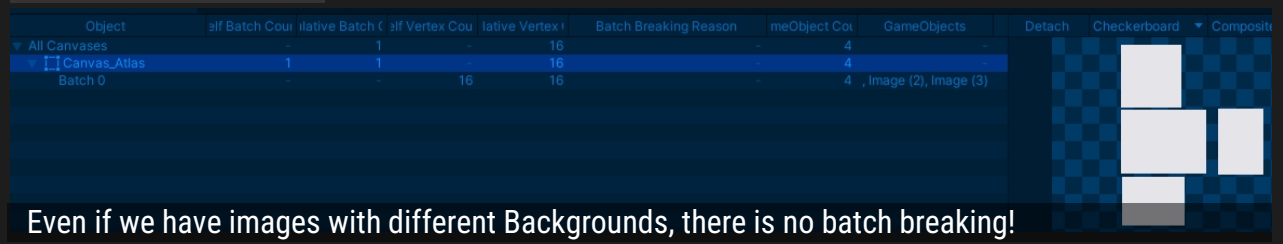


UI

- Simplify UI Structure
  - Batch as much as you can (use Texture atlas) [Canvas_Atlas]
  - Don't create game objects acting like folders and having no other purpose than organizing your Scenes
  - To create an UI Atlas
    - TextureType: Sprite
    - SpriteMode: Multiple

- Disabling invisible camera output [InvisibleCameraOut]
  - In case of full-screen UI with opaque background, the world-space camera will still render the standard 3D scene behind the UI
    - Disable all 3D world behind the UI
    - Use RenderToTexture once and use it as background



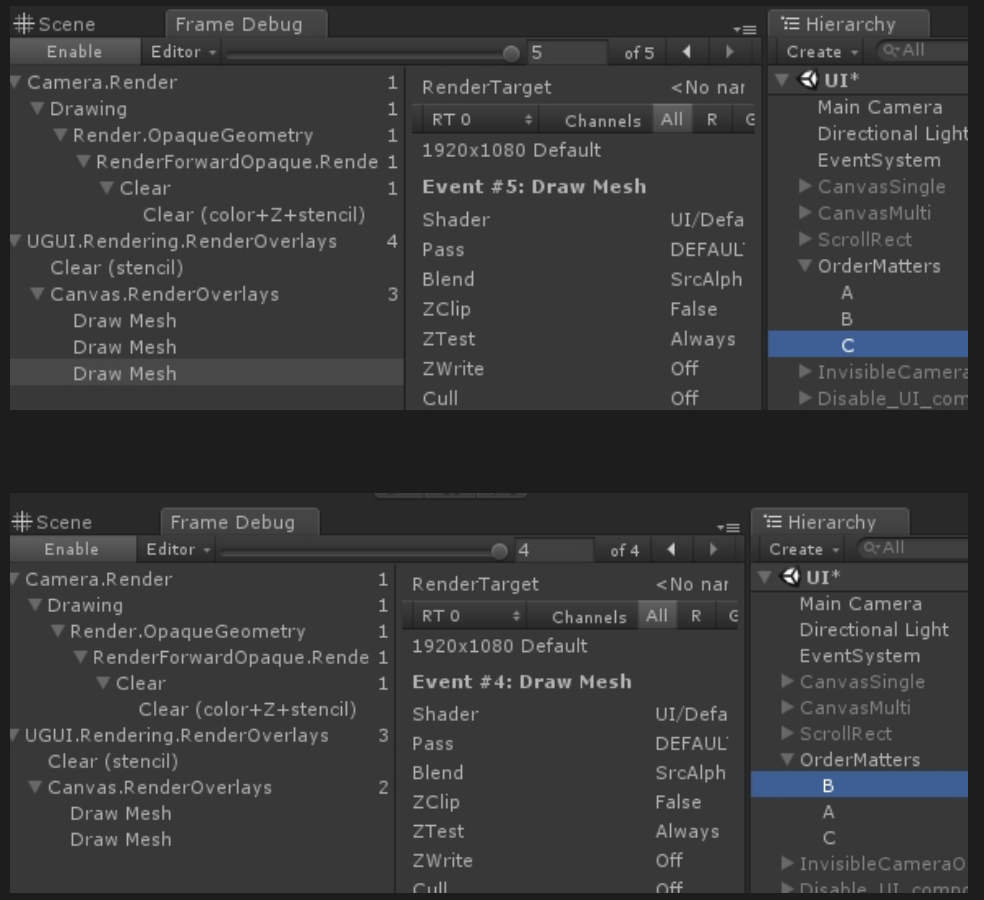


Even if we have images with different Backgrounds, there is no batch breaking!

UI

- Since UI elements are rendered in the transparent queue, UI Element order matters in terms of batching [OrderMatters]
  - A,C same material; B different material
  - Any quads that have unbatchable quads overlaid atop them must be drawn before the unbatchable quads (they cannot be batched with other quads placed atop the unbatchable quads)
    - A,C,B and B,A,C order results in 2 batches, A,B,C in 3 batches
    - Use FrameDebugger to see it in action (NB: Panels MUST overlay!)

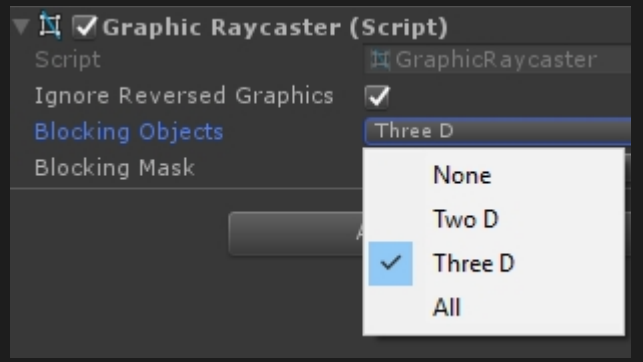
- Animators will dirty their elements every frame
  - Even if values in animation doesn't change
  - Animators have no no-op checks
  - Use them only on UI elements that Always change. Otherwise, use your own tweening system



UI Raycast

- Canvas.GraphicRaycaster is not a 'real raycaster': iterates over all Graphic components that have the RaycastTarget ON
- Need one on every Canvas that requires input
  - If we have a sub-canvas (a Canvas as child of another root Canvas), we need GraphicRaycaster component also on the sub-canvas, even if the root canvas has it

- For WorldSpaceCamera or ScreenSpaceCamera Canvases, the GraphicRaycaster can cast a ray into the 3D or 2D physics system
- Always disable Raycast Target for noninteractive elements (Text on a button, etc, to avoid useless work:
  - RaycastTargets list is sorted by
    1. Depth
    2. Filtered for reversed targets
    3. Filtered to ensure that elements not visible are removed



Try it

- Activate Raycast_Blocking/WorldSpace, or Raycast_Blocking/ScreenSpace
  - If click reaches Panel, the upper right image is shown
  - FacingCameraPanel prevent the click to reach Panel
  - The 3D Cube prevent the click to reach Panel
  - ReversedPanel allows the click to reach Panel
- Activate Raycast_Blocking/Hierarchy
  - Every Panel has RaycastTarget active. If you disable one, the raycast is forwarded to the parent (causing the activation of the parent top-right image)

UI Controls Optimization

- Always explicitly set ScreenSpaceCanvas/WorldSpaceCanvas.EventCamera
  - SSOverlay Canvas will set this property to the active camera
  - SSCamera or 3D Canvas leaves it to null: each time the is needed, the Main Camera is still used, but do so by calling FindObjectWithTag()

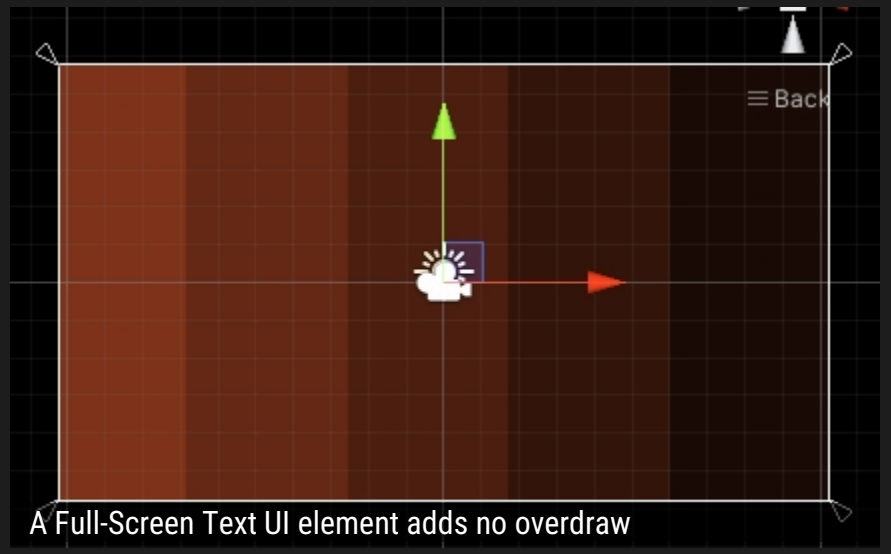
- When possible, disable Canvas.PixelPerfect flag
  - UI animated components, e.g. ScrollRect

- Use RectTransform-based Layouts instead of Layout Components [CanvasSingle]

Full-screen interaction

[Hierarchy/Canvas/FullScreenInteraction_Text_NoOverdraw]

- A common implementation in most UIs is to activate a large, transparent interactable element that covers the entire screen, forcing the player to handle a popup before proceeding, while still allowing the player to see what's going on behind it
- You can do this using a UIImage element. Unfortunately, this can break batching operations, and transparency can be a problem on mobile devices
- A hacky way around this problem is to use a UIText element with no Font or Text defined. This creates an element that doesn't need to generate any renderable information and only handles bounding box checks for interaction

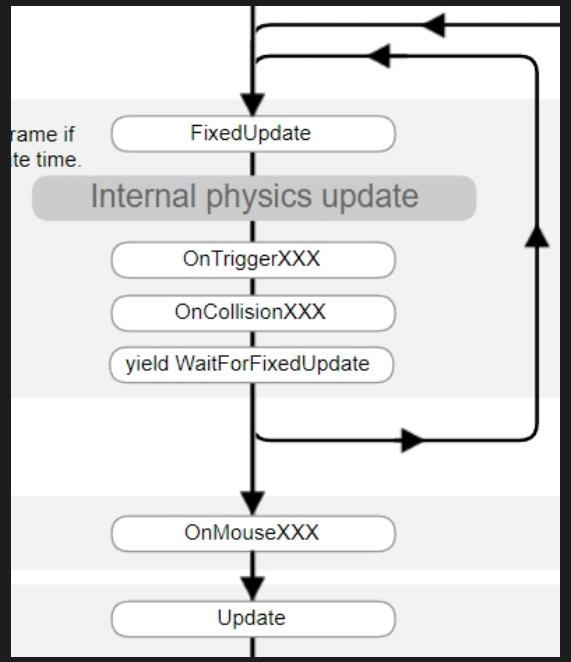


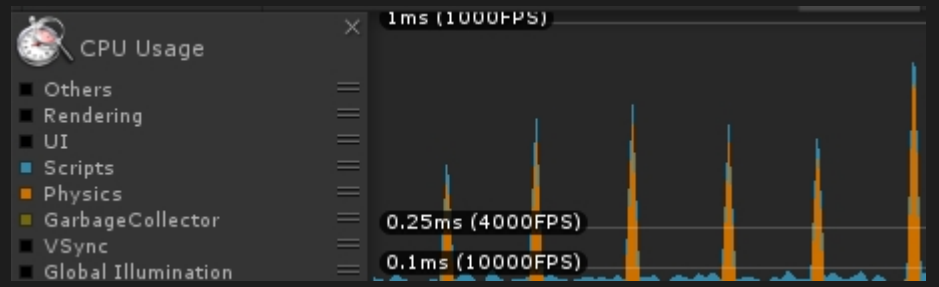
A Full-Screen Text UI element adds no overdraw

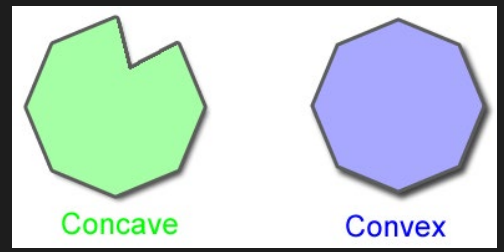
Physics

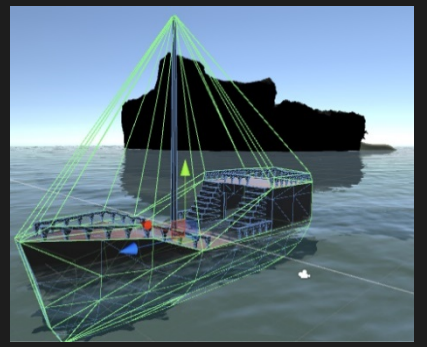
- FixedUpdates are processed just Before the Physics Engine perform its own update
  - ProjectSettings/Time/FixedTimestep
- Keep your physics cost < 1.5ms per Fixed Frame

Important factors
- Number of rigid bodies
- Number of colliders
- Complexity of colliders
- Project settings:
  - Collision matrix
  - ProjSettings/Physics/Auto sync transforms
    - 1 transform update = 1 physics update
    - Expensive! Unless you do it MAX once per frame -> AutoSync Transforms OFF
  - Broadphase Type
    - How to find objs to do the physics simulation for?
    - Mostly static: Sweep & Prune
    - Mostly dynamic: Multibox Pruning / Auto Box pruning
  - Fixed Timestamp
    - For profiling: set to your target FPS in ms, e.g. 0.00833 for 120 FPS
    - For production; anything between 2 to 3 times your target FPS in ms
  - ProjSettings/Physics/ReuseCollisionCallbacks
    - Leave it ON: fewer allocations for every collision







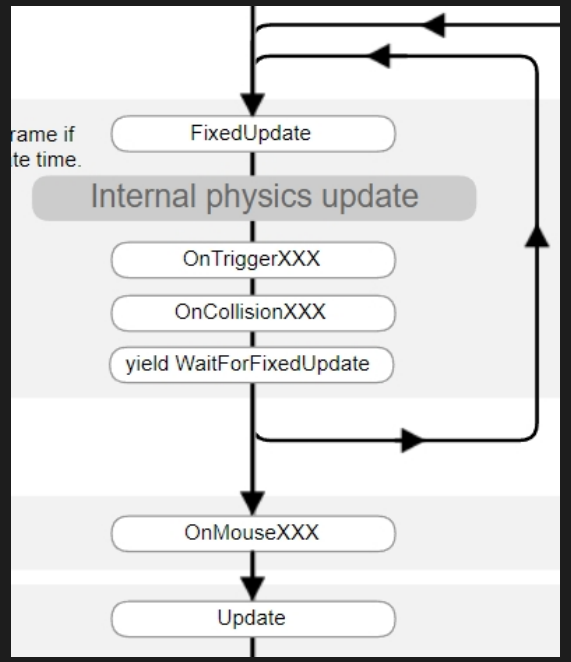


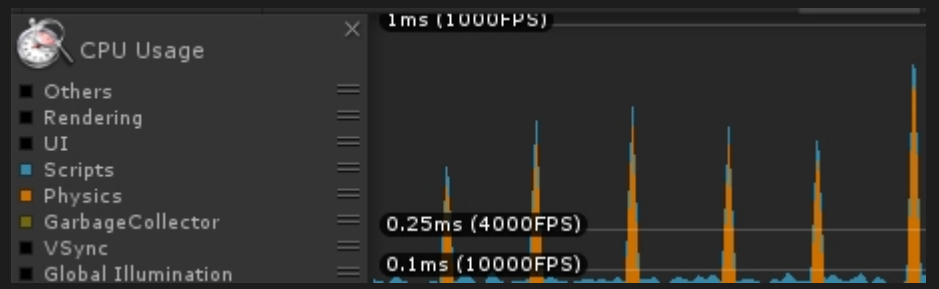
Physics

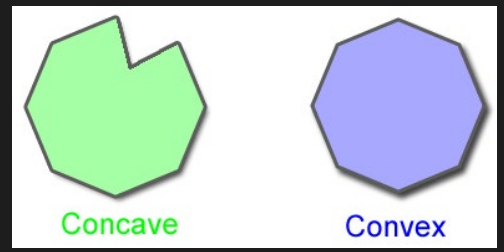
- ProjectSettings/Time/MaximumAllowedTimestep
  - Avoid “Spiral of Death” problem (when Physic engine cannot escape the FixedUpdate() loop)

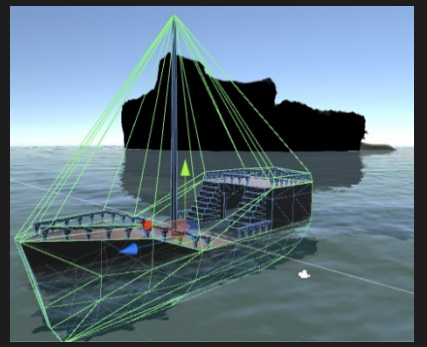
- FixedUpdate() Useful place to perform frame-rate independent calculations and those changes that must be synched with PhysicsEngine
  - AI
  - RigidBody changes (Apply Forces/Impulses)
    - applying a constant force each Update() while the player holds down a key would result in completely different resultant velocity between two different devices than if we did the same thing in FixedUpdate() since we can't rely on the number of Update() calls being consistent

- Try this
  - Set ProjSettings Time/FixedTimeStep TO 0.1 OR 0.5. You can see in CPU profiler that Profiler Spikes = FixedUpdate()
- Convex/Concave colliders
  - ConcaveColliders are too expensive => Cannot be dynamic





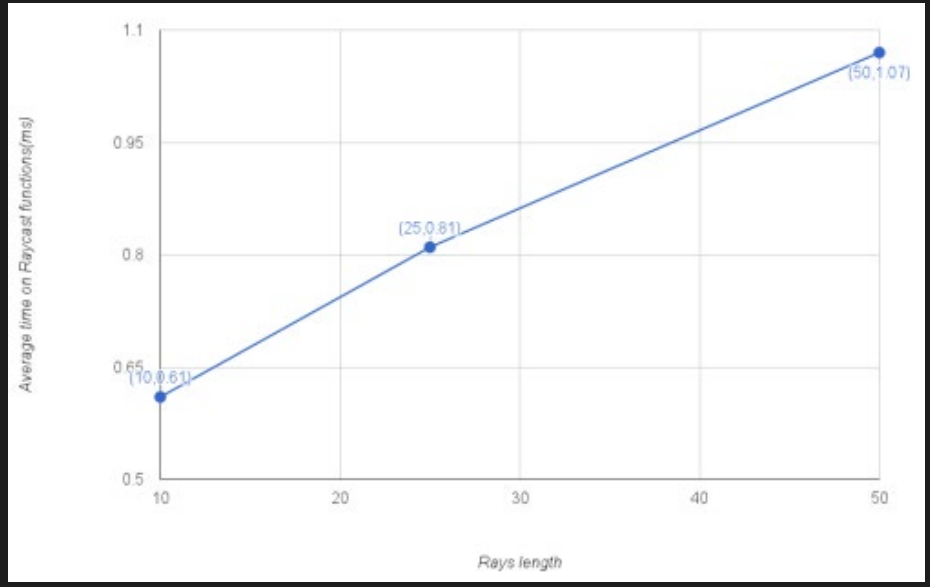


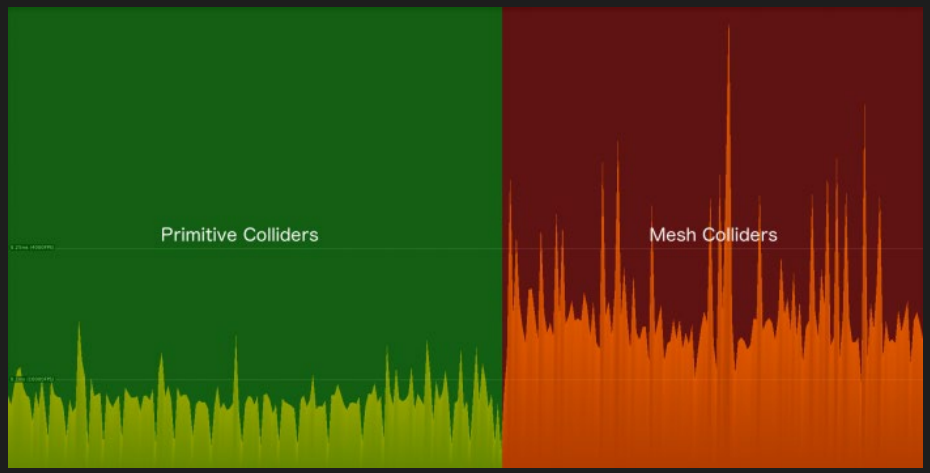


Raycast

Types of Casts

- Physics.CheckBox/Capsule/sphere Check if a shape overlaps with any collider in its area
- Physics.OverlapBox/Capsule/sphere Find all colliders touching or inside the shape
- Casts
  - RayCast
  - BoxCast
  - SphereCast
  - CapsuleCast
- A few casts per frame are ok, more than a few… maybe not
- Don't extend the ray's length more than you need to
- Raycasting against a mesh collider is really expensive
  - Create children with primitive colliders and try to approximate the meshes shape
  - All the children colliders under a parent Rigidbody behave as a compound collider
  - If you do need mesh colliders, make them convex
- Use a layer mask





Unity3D Course – Luca Palmili

Images from Unity3D Official Guide

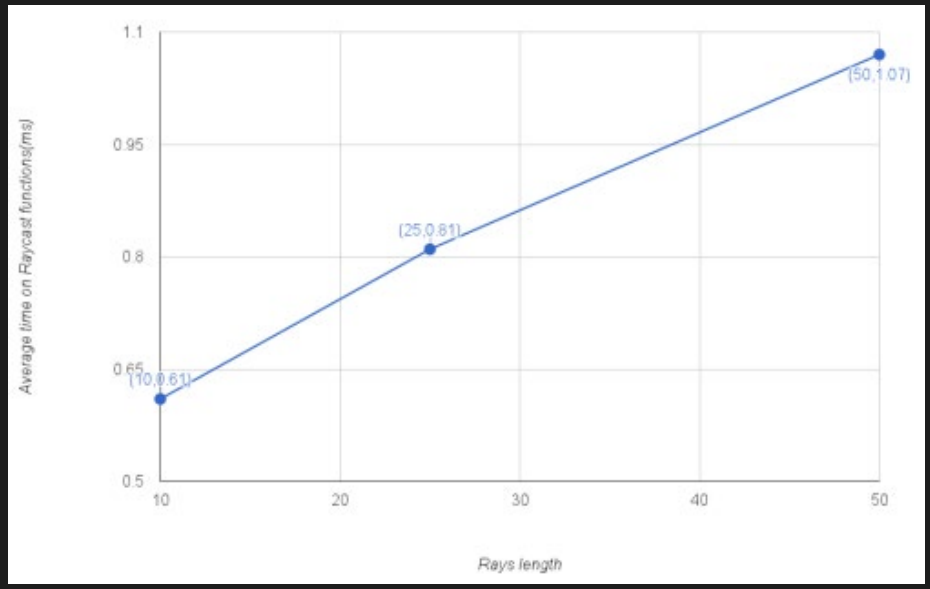
Raycast

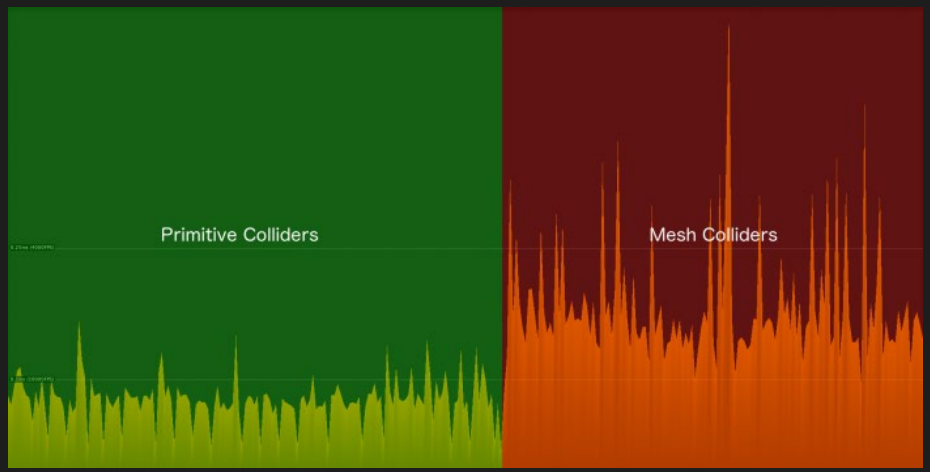
Use non-alloc version: RaycastAll vs RaycastNotAlloc (there is a NotAlloc version for all casts)

- RaycastNotAlloc is like Physics.RaycastAll, but generates no garbage
- Physics.RaycastAll creates and returns an array of RaycastHit structs. This creates garbage on the heap
- RaycastNonAlloc does the same but instead of returning a new array each time it's called, you have pass a preallocated array to the method and it just fills in the elements. The method just returns an integer that tells you how many hits has been filled in. The array should be large enough to hold all possible hits. The great thing is you can reuse the same array each time you call the method

- RaycastNonAlloc can't be compared to Raycast as it does something different. They should have called it "RaycastAllNonAlloc"

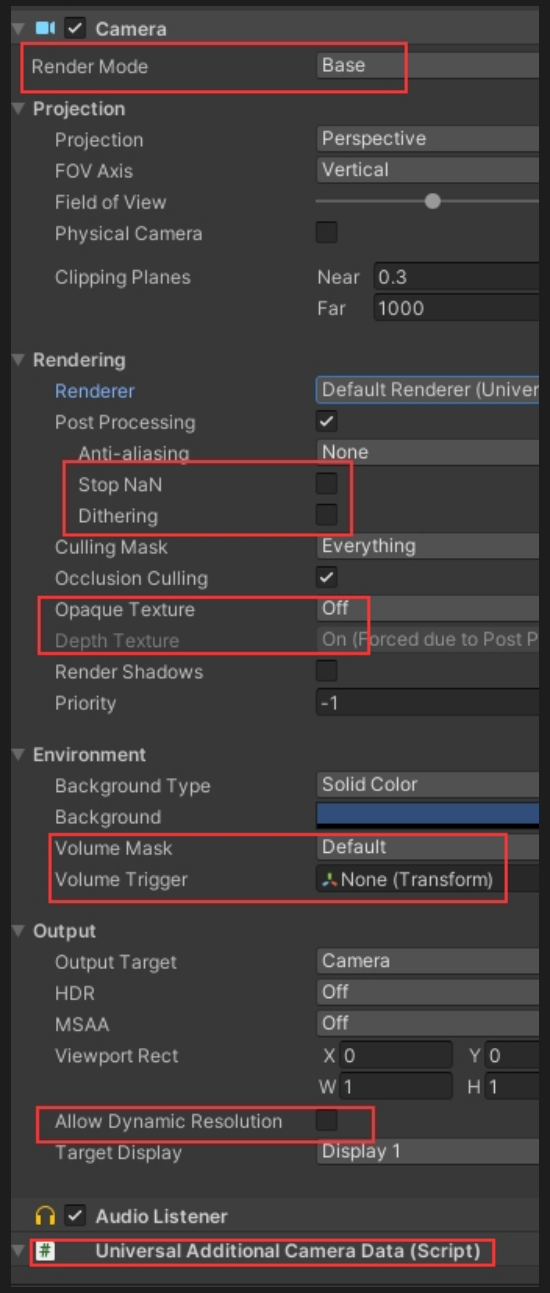
- [Advanced] Do them asynchronously (Jobs)
  - In this way they not block the main thread
  - Instead of using Physics.CapsuleCast, use CapsulecastCommand





Cam component

- DynamicResolution Dynamic resolution reduces the workload on the GPU, which helps maintain a stable target frame rate
  - https://docs.unity3d.com/Manual/DynamicResolution.html?_ga=2.183544806.1129183846.1588411358-1656116434.1556010765



URP Material conversion tool

- UpdateDefaultMaterial.cs
  - We can now use AssetPathToGUID() and LoadAssetAtPath() to better perform DefaultMaterial assignment in case of a wrong material in the scene

Update Materials to URP

- Some materials can’t be converted into URP materials, simply because there is not a provided shader converter from the start shader to a compatible URP shader
- You can try to hack this conversion when you got an error like this:
  - “xxx material was not upgraded. There's no upgrader to convert xxx shader to selected pipeline UnityEditor.Rendering.Universal.UniversalRenderPipelineMaterialUpgrader:UpgradeSelectedMaterials() (at Library/PackageCache/com.unity.render-pipelines.universal@x.y.z/Editor/UniversalRenderPipelineMaterialUpgrader.cs)”
  - If the error was a Custom/ToonShader material, try to add something like upgraders.Add(new StandardUpgrader("Custom/ToonShader"));
    In UniversalRenderPipelineMaterialUpgrader.cs GetUpgraders() method

Scripting 2/4

To change position value of a GameObject, we do

- Vector3 oldPos = GObj.transform.position;
- Vector3 newPos = oldPos*2;
- GObj.transform.position = newPos;

To change PS properties, we need to access its modules

```
// Get the emission module
ParticleSystem.EmissionModule emissionModule = GetComponent<ParticleSystem>().emission;
// Enable it and set a value
emissionModule.enabled = true;
emissionModule.rateOverTime = 15;
```

- Does it seem weird? Why?
  - We grab the struct and set its value, but never assign it back to the PS. How can the PS ever know about this change?

Scripting 3/4

```
var emissionModule = GetComponent<ParticleSystem>().emission; [1]

public sealed class ParticleSystem : Component
{…
    public EmissionModule emission { get {
        return new EmissionModule(this); } } [2]
…}

public partial struct EmissionModule
{
    // Direct access to the PS that owns this module
    private ParticleSystem m_ParticleSystem;
    EmissionModule(ParticleSystem particleSystem) {
        m_ParticleSystem = particleSystem;
    }
    public MinMaxCurve rateOverTime { set {
        // Here we call down to the C++ side to perform the rate variation
        m_ParticleSystem->GetEmissionModule()->SetRate(value); [3]
    }}
}
```

- PS are in the Unity C++ side
- PS Modules are properties of a PS, they are never shared between different PS: a module will always belong to the same PS

1. PS receives a request for the Emission module
2. The engine create a new EmissionModule struct, passing the owning PS as its only parameter
3. To set the rate, the var m_ParticleSystem is used to access the moduel and set it directly

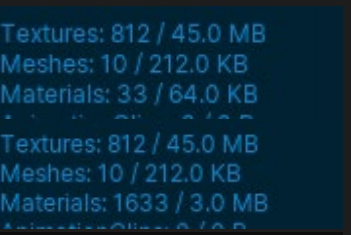
- This is why we don’t need to reassign the module to the PS: it is always part of it
- EmissionModule struct is just an interface into the PS internals
- It is not possible for modules to be shared or assigned to different PS

Material Property blocks

[Needs shader programming]

Try to

- Change Sphere materials with Renderer.material.color = newColDraw. Set MPropertyBlocks.ChangeMaterial to TRUE
  - If we have N Spheres, we'll create N new Materials
  - To change the color in the shader Unity needs to tell the GPU that this object is going to be rendered differently > changes the material instance > creates a copy of a material when you access renderer.material for the first time
  - Be aware that this material copy will not be destroyed automatically! You will have to do so yourself, lest your create a memory leak
    - In your OnDestroy() method, if you accessed the Renderer.material property, call Destroy(Renderer.material) to clean up



1600 spheres
+ 1600 new materials
+ 3MB of memory size

[MaterialPropertyBlocks.scene]

Code Compilation

Unity is written in C++. Why use C# for scripting?

- C++ is frustrating
- In C# is more difficult to introduce Memory bugs
  - GarbageCollection
- C# is reasonable fast (Runtime performance cost are >= NativeCode)
- C# has a base class library and resources that just works well together
- C# programmer are easy to find

1. When we make changes to our C# code, it is automatically compiled when we switch back from our IDE to Unity. The C# code is not converted directly into Machine Code. Instead, the code is converted into an intermediate stage called Common Intermediate Language (CIL) - This is how .NET can support multiple languages
2. CIL run through MonoVirtualMachine, which is an infrastructure element that allows the same code to run against multiple platforms, without the need to change the code itself. This is an implementation of the .NET Common Language Runtime (CLR)
3. CLR compiles CIL into Native Code, using AheadOfTime or JustInTime compilation (depend on the platform that is being targeted)

- AOT
  - Happens during build process or at start
  - No runtime cost
  - Can optimize code (better mem sharing)
  - Reduced startup time
- JIT
  - Happens dynamically at runtime in a separate thread
  - First invocation of a piece of code is slower than AOT
    1. Allocates memory for NativeCode instructions
    2. Generates NativeCode instructions
    3. Mark memory as executable
  - Can’t use AOT code optimization
- 90% of work is being done by 10% of code > JIT compilation could be a good choice

AOT Limitation

- Generic Virtual Methods
  - ExecutionEngineException: Attempting to call method 'AOTProblemExample::OnMessage<AOTProblemExample+AnyEnum>' for which no ahead of time (AOT) code was generated
  - The AOT compiler does not realize that it should generate code for the generic method OnMessage with a T of AnyEnum, so it continues, skipping this method. When that method is called, and the runtime can't find the proper code to execute

```
public class AOTProblemExample : MonoBehaviour, IReceiver  {
        public enum AnyEnum { Zero, One, }
        void Start() {
                // Subtle trigger: The type of manager *must* be
                // IManager, not Manager, to trigger the AOT problem.
                IManager manager = new Manager();
                manager.SendMessage(this, AnyEnum.Zero);
            }
        }
        public void OnMessage<T>(T value) {
                Debug.LogFormat("Message value: {0}", value);
            }
        }
}
```

```
public class Manager : IManager {
    public void SendMessage<T>(IReceiver target, T value) {
        target.OnMessage(value);
    }
}

public interface IReceiver {
    void OnMessage<T>(T value);
}

public interface IManager {
    void SendMessage<T>(IReceiver target, T value);
}
```

AOT Limitation

- How to solve that?

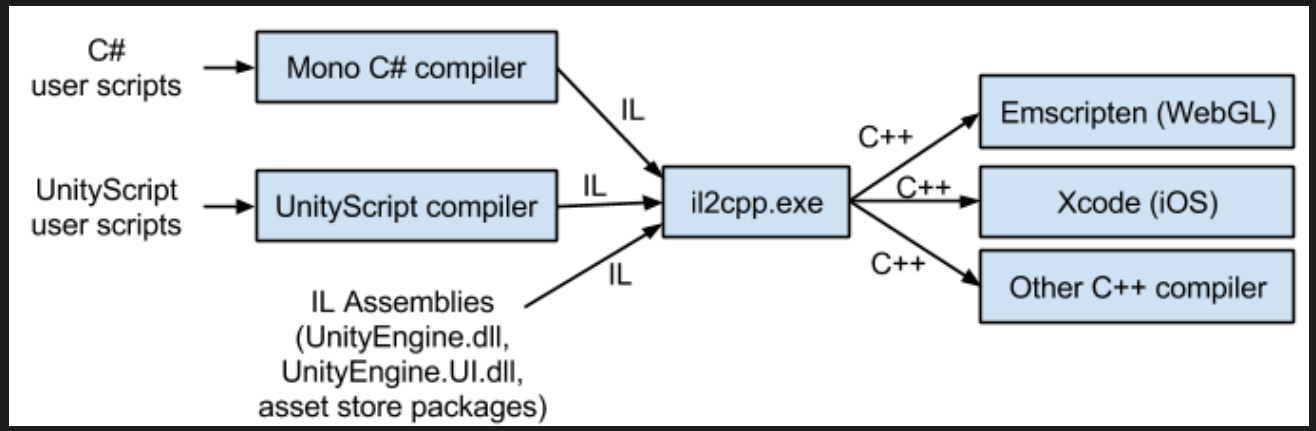
```
public class AOTProblemExample : MonoBehaviour, IReceiver  {
        public enum AnyEnum { Zero, One, }
        void Start() {
            // Subtle trigger: The type of manager *must* be
            // IManager, not Manager, to trigger the AOT problem.
            IManager manager = new Manager();
            manager.SendMessage(this, AnyEnum.Zero);
        }
    }
    public void OnMessage<T>(T value) {
            Debug.LogFormat("Message value: {0}", value);
        }
    }
}
```

```
public void UsedOnlyForAOTCodeGeneration() {
        OnMessage(AnyEnum.Zero);

        // Include an exception so we can be sure to know if
        this method is ever called.
        throw new InvalidOperationException("This method is
        used for AOT code generation only. Do not call it at
        runtime.");
}
```

IL2CPP

- IntermediateLanguageToC++
- ProjectSettings/Player/Configure/ScriptingBackend
- Mono
  - C# > Mono C# Compiler > CIL > MonoVM AOT/JIT compilation > Native code
- IL2CPP
  - C# > Mono C# Compiler > CIL > IL2CPP > C++ > Architecture specific compiler > Native code

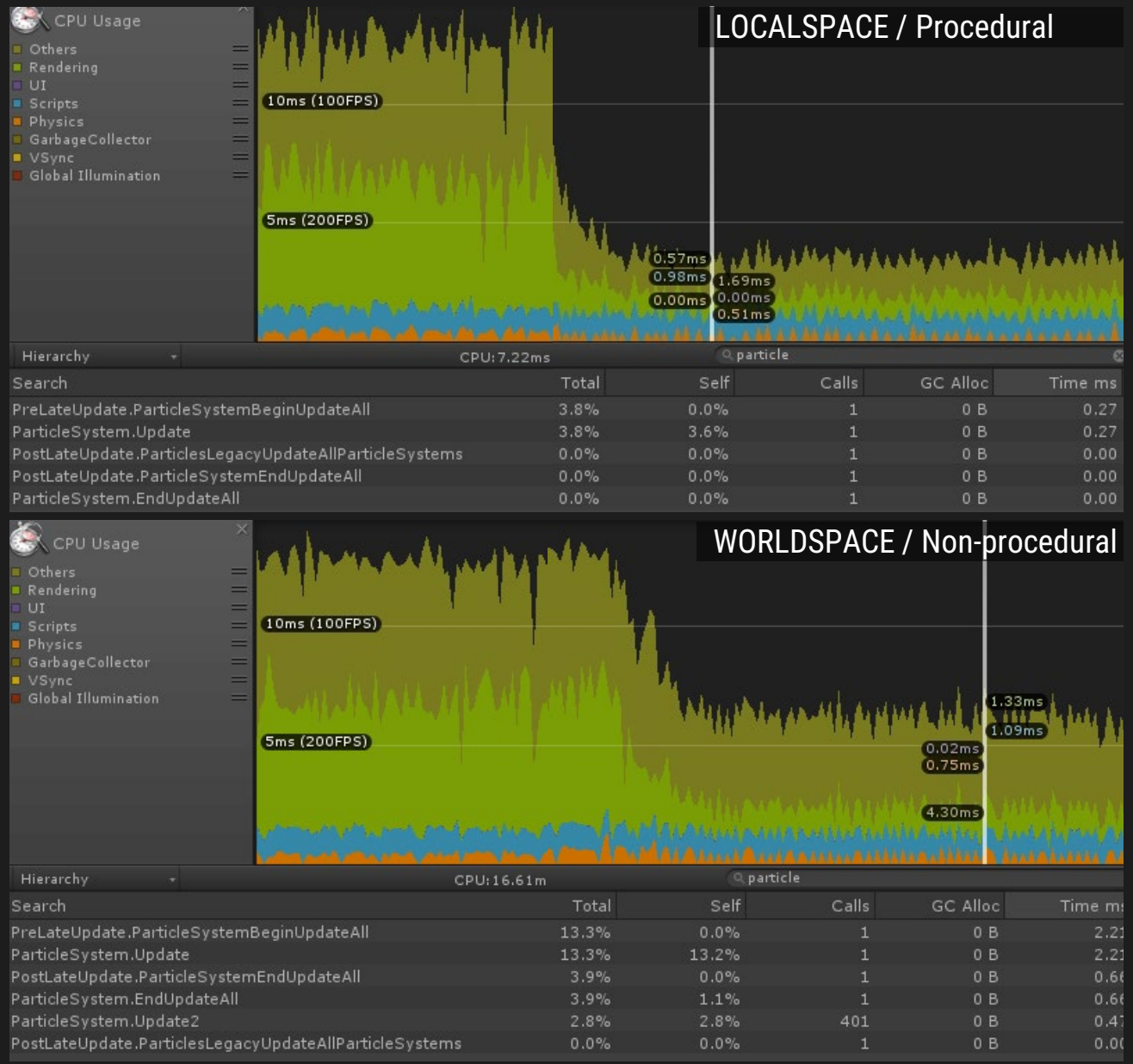
- IL2CPP PROs
  - Lower cost of porting and maintenance of architecture specific code generation
  - Features and bug fixes are immediately available for all platforms
  - Uses architecture specific compilers rather than MonoVM: better optimization
  - GC is not specific, but a pluggable API



Particle Systems

- Test Profiler stats with procedural and non-procedural ParticleSystems
  - LocalSpace: CPU Spikes with ParticleSystem.Prewarm Task
- Avoid recursive ParticleSystem calls: Start/Stop/Pause/Clear/Simulate() call GetComponent <ParticleSystem>() on each child
  - Pass false as to withChildren param, e.g. ps.Clear(false) will avoid recursive calls
  - Cache Particle System Components in a PSManager and manually iterate through them

[ParticleSystem.scene]

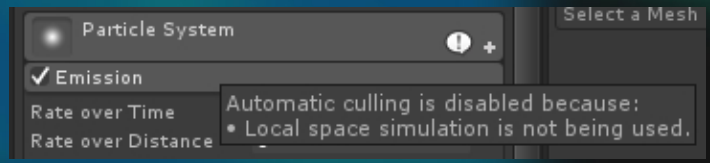


Particle Systems

- Culling is only possible when a system has predictable behaviour
  - What breaks ProceduralMode?

- Changing values via script
- Change values in the editor during play mode
- Calling Play on a system that has been stopped will reset the system and re-validate procedural mode

Module	Property	What breaks it?
	Simulation Space	World space
Main	Gravity modifier	Using curves
Emission	Rate over distance	Any non zero value
External forces	enabled	true
Clamp velocity	enabled	true
Rotation by speed	enabled	true
Collision	enabled	true
Trigger	enabled	true
Sub Emitters	enabled	true
Noise	enabled	true
Trails	enabled	true
Rotation by lifetime	Angular Velocity	if using a curve and t procedural*
Velocity over lifetime	X, Y, Z	If using a curve and t procedural*
Force over lifetime	X, Y, Z	If using a curve and t procedural*
Force over lifetime	Randomise	enabled



Projectors

Used to create fake shadows, Simulate real projectors, Bullet marks, Special Fx

Quick Setup

- Import projector Shaders
  - Import [ProjectorAssets.Upackage]
- As a good start reference, use StandardAssets/Effects/Projector/Prefab/BlobShadowProjector
- Create your own projector
  - Add Projector component
  - Create a new Material with Projector shader
  - Remember to set IgnoreLayers layermask on the projector
- Cookie texture:
  1. Make sure texture wrap mode is set to "Clamp“
  2. Turn on "Border Mipmaps" option in import settings
  3. Use uncompressed texture format
  4. Projector/Shadow also requires alpha channel to be present (typically Alpha from Grayscale option is ok)
- Falloff texture (if present), use the one provided with BlobShadowProjector
  1. Data needs to be in alpha channel, so typically Alpha8 texture format
  2. Make sure texture wrap mode is set to "Clamp“
  3. Make sure leftmost pixel column is black; and "Border mipmaps" import setting is on

[Materials_00/Projector]