Stats panel

- Not a debug tool

- FPS does not include the time taken for Edotor/Scene view, inspector and other editor-only processing
- Batches DrawCalls
- Saved by batching # of batches that was combined
- SetPass # of rendering passes. Each pass requires Unity runtime to bind a new shader
- VisibleSkinnedMesh

- Try to count Tris/Verts of a Cube primitive

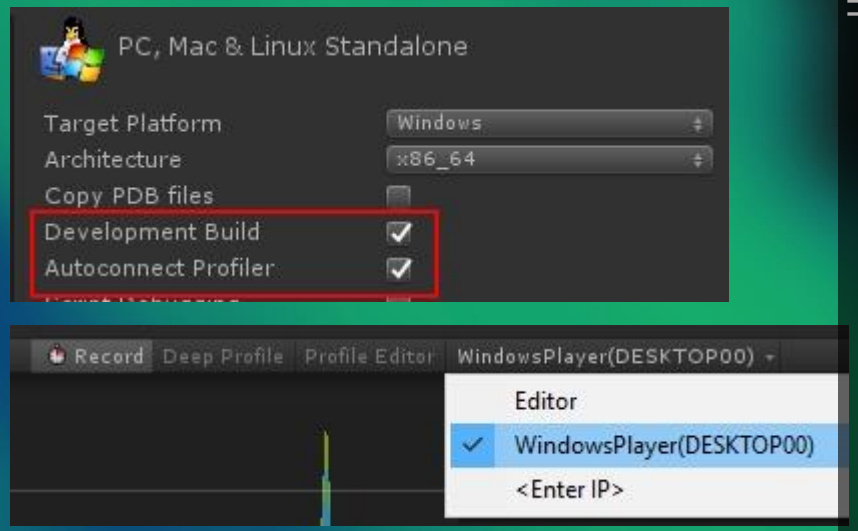
Profiler

- CPU consumption
- Rendering and GPU information
- Runtime memory allocations
- Audio source/ data usage
- Physics Engine (2D and 3D) usage
- Network messaging and operation usage
- Video playback usage
- Basic and detailed user interface performance (2017+)
- Global Illumination statistics (2017+)

- Instrumentation
  - Detail info
  - Profiling has its own performance cost
- Benchmarking
  - FPS
  - Spikes in CPU/GPU activities

- Good profiling is done on exported build
  - Development Build
  - Autoconnect Profiler

[Profiler.scene]

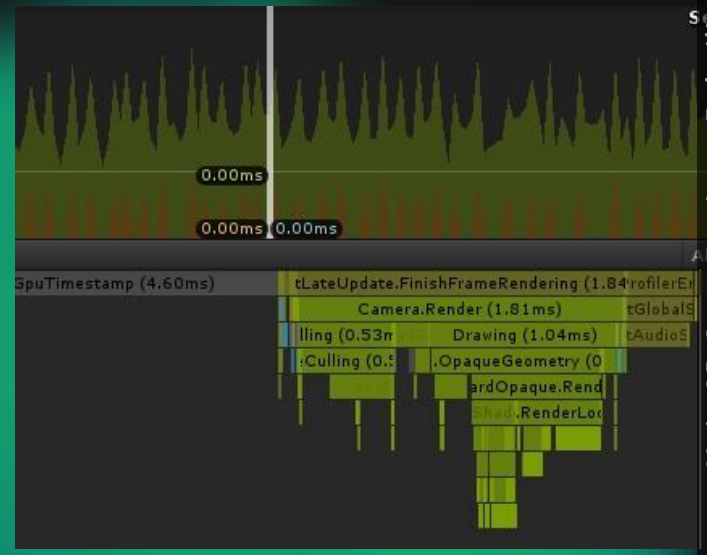


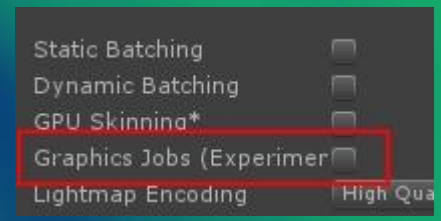
Profiler

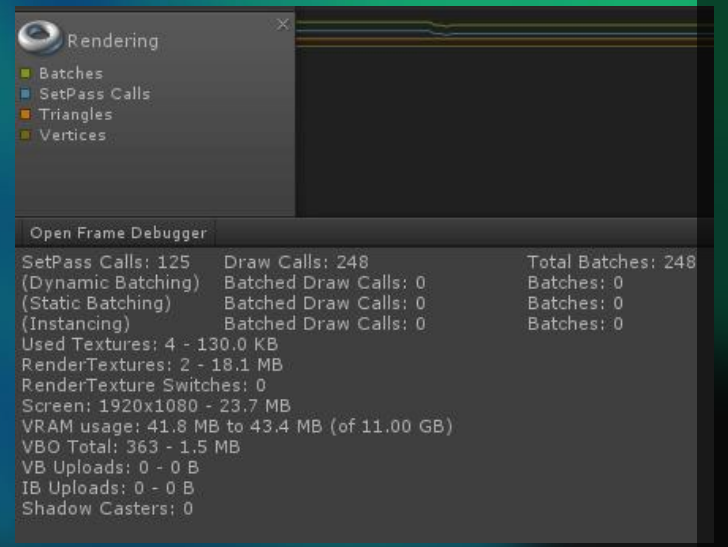
- ProfilerControls
  - AddProfiler
  - Deep Profile – Only if we don't have enough detail or for small test scene
  - Profile editor – Custom Editor Scripts
  - Load/Save – Up to 300 frames of data

- CPU Area / Timeline
  - CPU tasks order
  - Which thread is responsible for which tasks (Main/Render/Working thread)

- Rendering Area
  - Draw/SetPass calls
  - FrameDebugger link





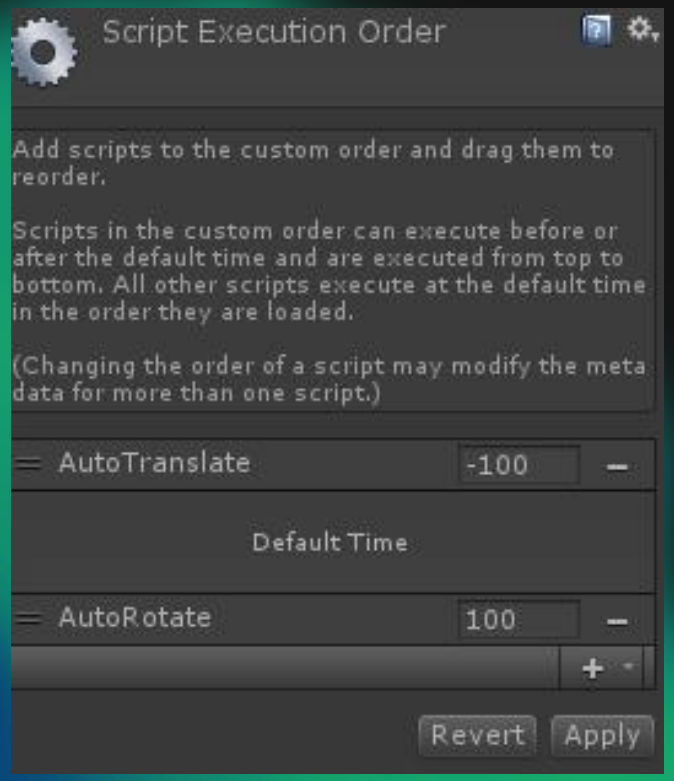


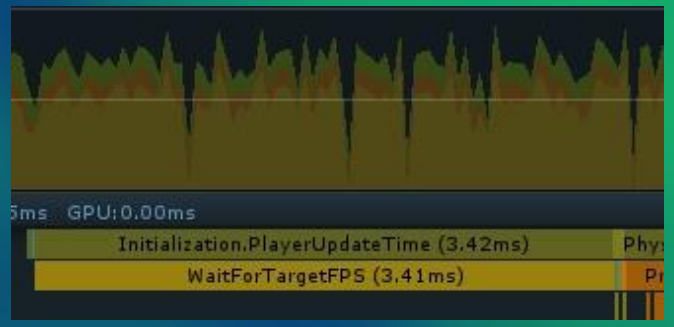
Performance analysis

- 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
    - Edit/ProjectSettings/Quality to enable/disable
  - Logging

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





Rendering pipeline (again)

- 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

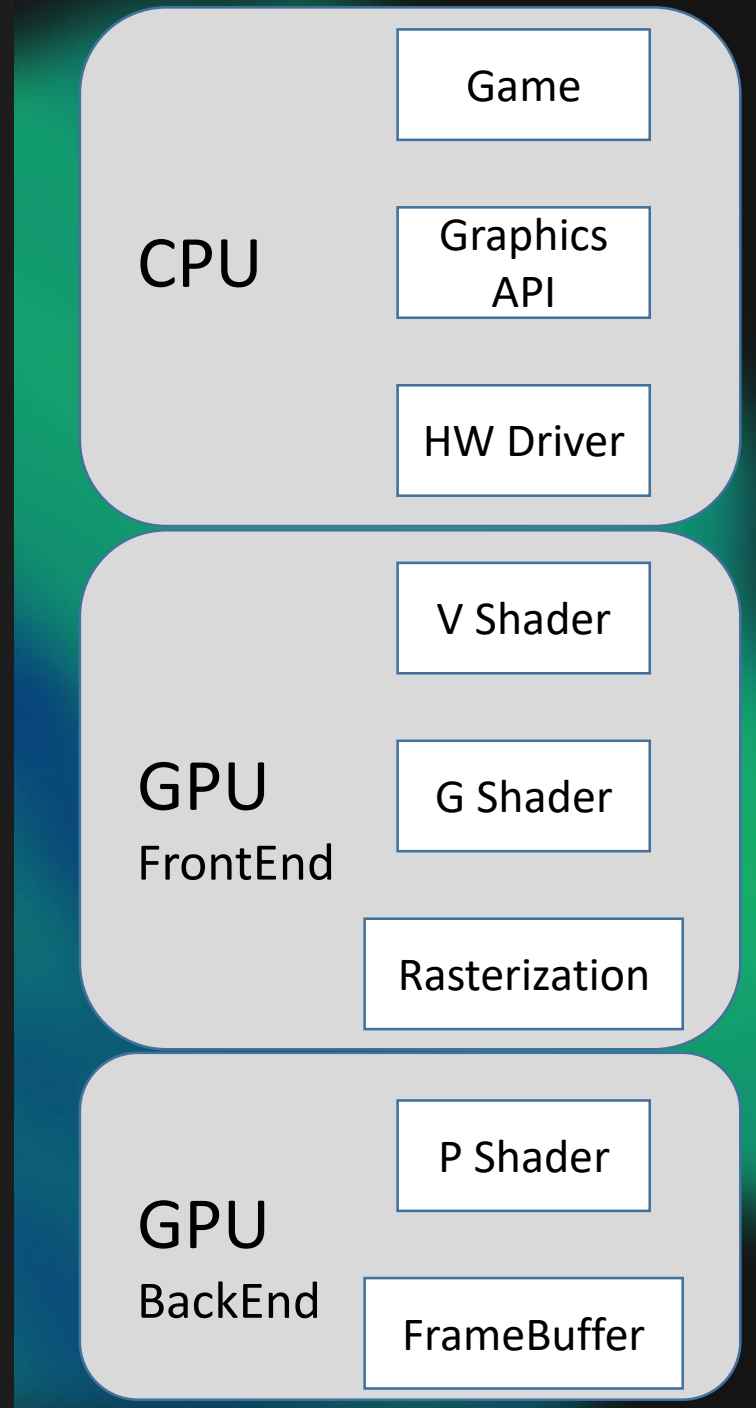
- If the GPU falls behind (GPU Bound), or the CPU spends too much time generating commands (CPU Bound), the frame rate will start to drop

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



Rendering pipeline (again)

BackEnd Bottlenecks

- Fillrate Speed at which the GPU can draw fragments that have survived all of the Fragment Shader tests
  - 30GPixels/second, target 60Hz, resolution 2560x1440 > 30.000.000.000 / 60 = 500M fragment/frame > if there is no overdraw, we can paint the entire screen 125 times
  - There is always Overdraw, which could transform the Fillrate into a bottleneck
- MemoryBandwidth
  - VRAM contains Rendering State info (also textures)
  - Uses a cache to perform texture fetching
  - 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 336GB/second = 5.6GB/frame

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

Multithread
- Main thread
  - Physics and script code
- Render thread
  - Pushing commands into the GPU
- Other worker threads
  - Culling, mesh skinning, etc

- Enabled by default on Desktop
- Android PlayerSettings/OtherSettings/MultithreadedRendering
- iOS PlayerSettings/OtherSettings/GraphicsAPI/Metal

Helps CPU-bounded scenarios (GPU is multicore, SIMD)

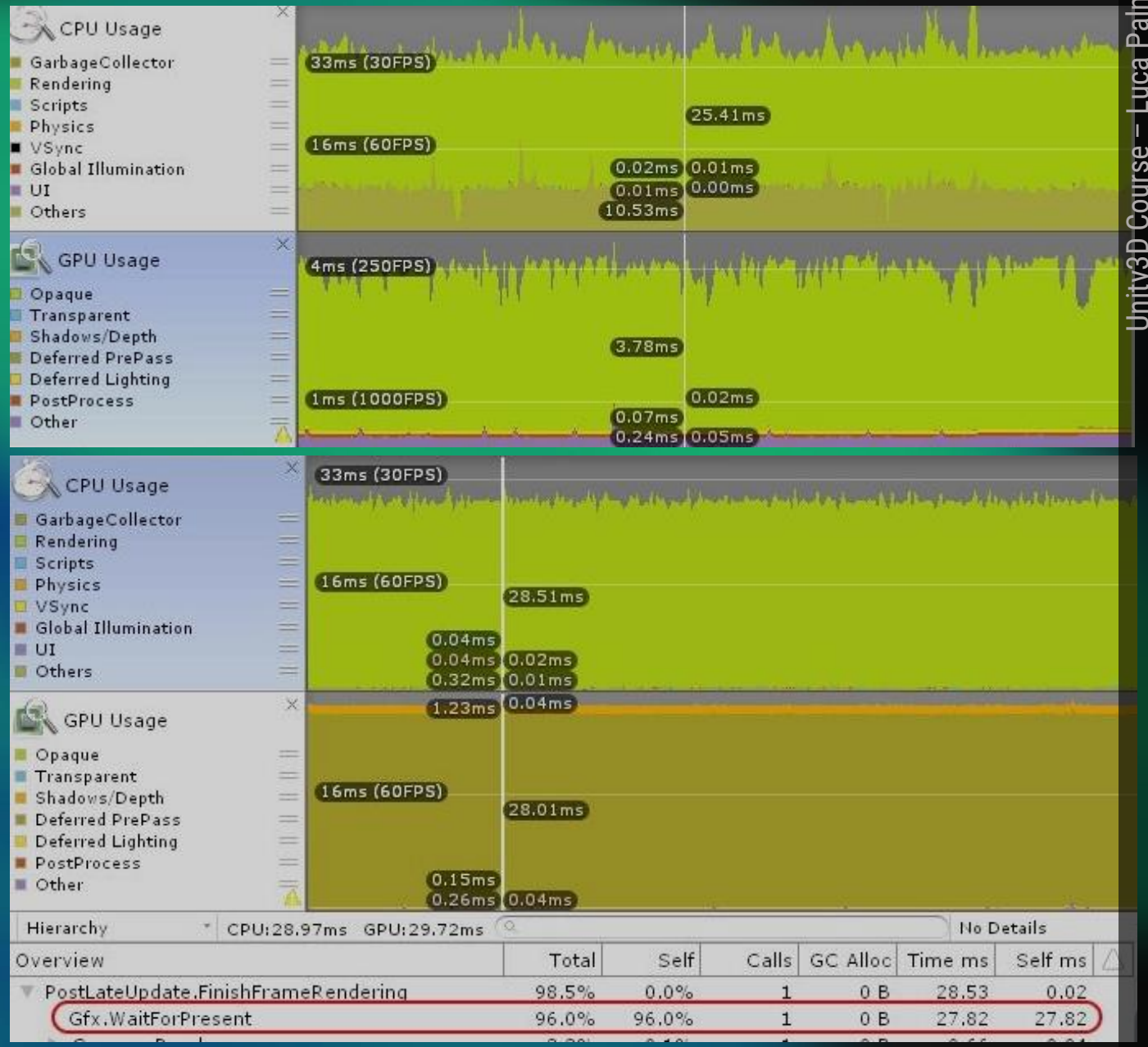
- PlayerSettings/Graphics Jobs Try to take some Main/Rendering thread tasks and distribute them on multiple cores
- Win/Mac/PS4/XB1

CPU / GPU Bound

CPU Bound
- Lot of drawcalls w/o optimization

GPU Bound
- Few drawcalls for the CPU
- Extremely heavy Fragment shaders
- Profiler/CPU Timeline should display Gfx.WaitForPresent task

[Profiler_CPUBound.scene]



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

- Combining the data from objects that must be rendered
  - Reduce Batches / SetPass calls
  - Static Batching
  - Dynaminc Batching
  - Batching UI
  - GPU Instancing
  - Texture atlasing
  - Combine mesh manually

GPU Bound optimization

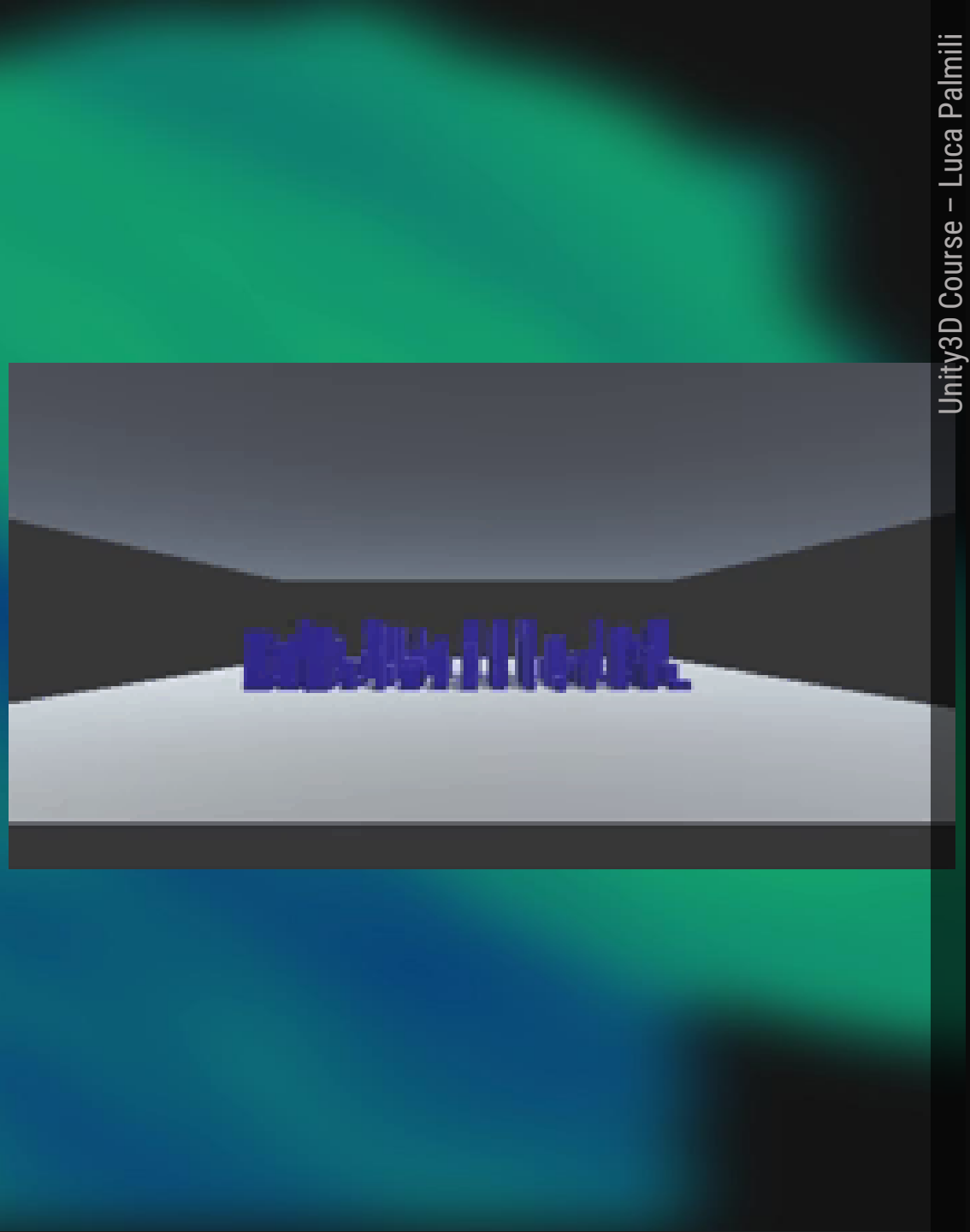
- FrontEnd Bottleneck
  - Not so common: Vshaders trivial VS Pshaders
  - Complex Geometry shader
    - Improve Tessellation
  - Normal Mapping
  - LOD

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

LayerCullDistance

- CPU-Bound - Reduce Batches / SetPass calls
- Camera.main.layerCullDistances = new float{…32 values…}

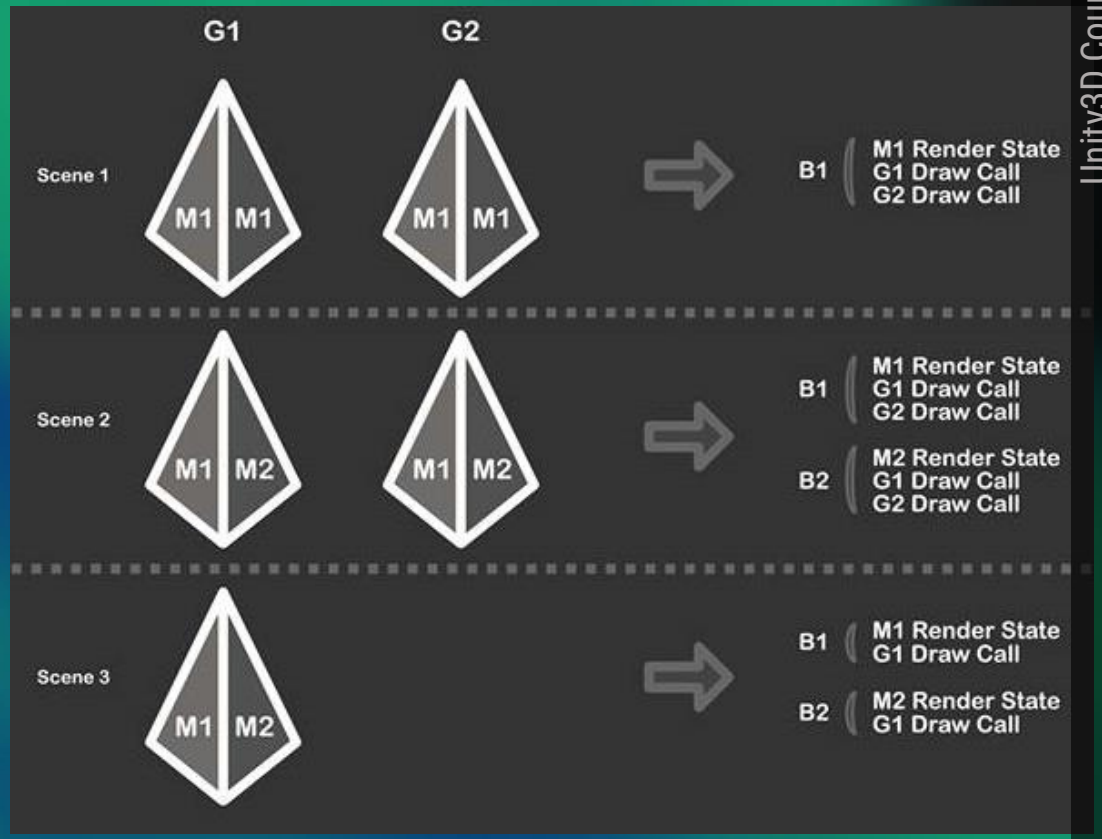
[CullingLayer.scene, CullDistanceSetter.cs]



DrawCall

Why minimizing drawCalls is so important?

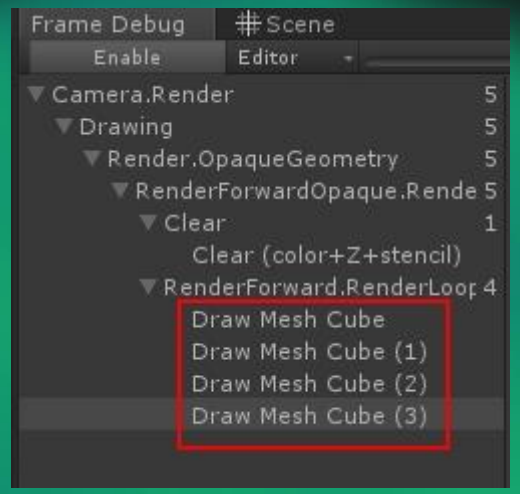
- To draw something, we need to set the RenderState
- E.g. Think about the texture to use as a global variable on the GPU
  - Change a global variable in a SIMD architecture is not simple
- Each obj has a Material, linked to one Shader
- If next obj has the same material, GPU can avoid switching the 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

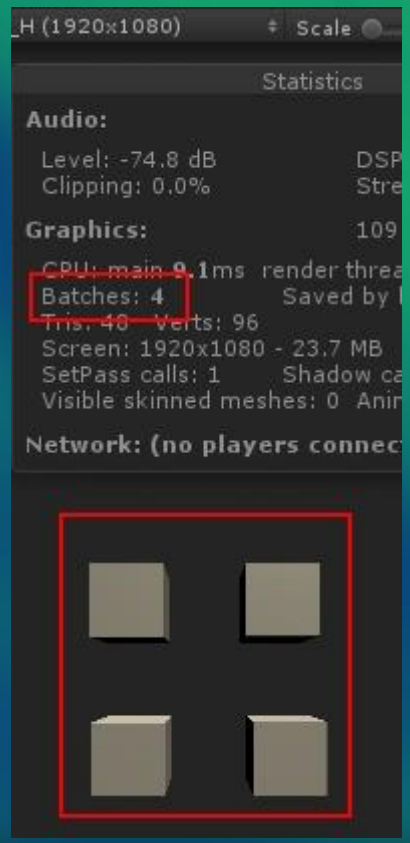


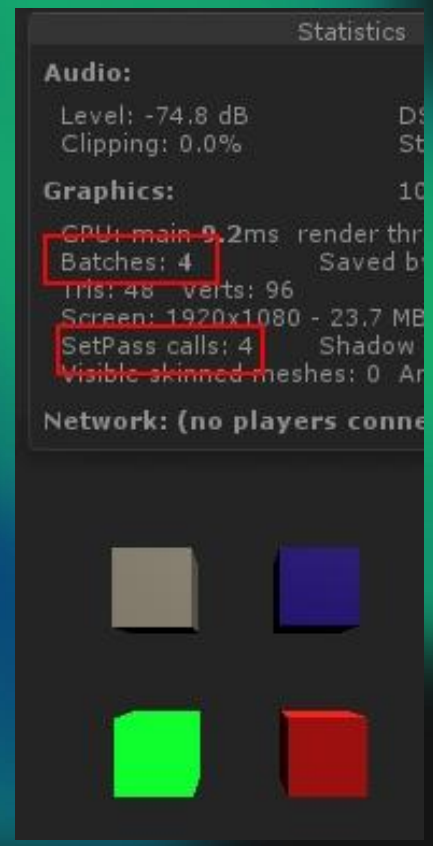
FrameDebugger

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

- One single material
  - 4 DrawCall
  - 1 SetPass Call – 1 RenderState is needed
- 4 Materials
  - 4 DrawCall
  - 4 SetPass Call – 4 RenderStates are needed







Batching

- Dynamic batching
  - Takes several small meshes each frame, transforms their vertices on the CPU, groups many similar vertices together, and draws them all in one go

- Static Batching
  - Combines static meshes in one or more large meshes at build time and at run time renders them as one batch per mesh

- GPU instancing
  - Draws many identical objects with different positions, rotations, and other shader properties in fewer draw calls

DynamicBatching

Setup

- PlayerSettings/PtherSettings/DynamicBatching ON

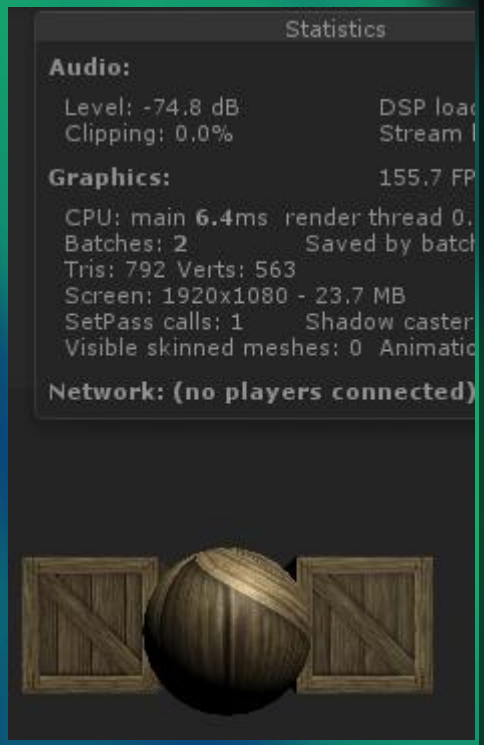
Try

- 1 cube 1 material
- 2 cubes 1 material
- 2 cubes 1 sphere 1 material
- 2 cubes, 1 with negative scale

Limits

- Meshes must share the same material
- Max 300 vertices per mesh
- total # of vertex attributes used by the Shader < 900
- Max 3 attributes per-vertex: Eg. position, normal, single set of UV
    - E.g. Complex Shader - 5 attributes per-vertex = no more than 180 vertices

[DynamicBatching.scene]



DynamicBatching

How it works

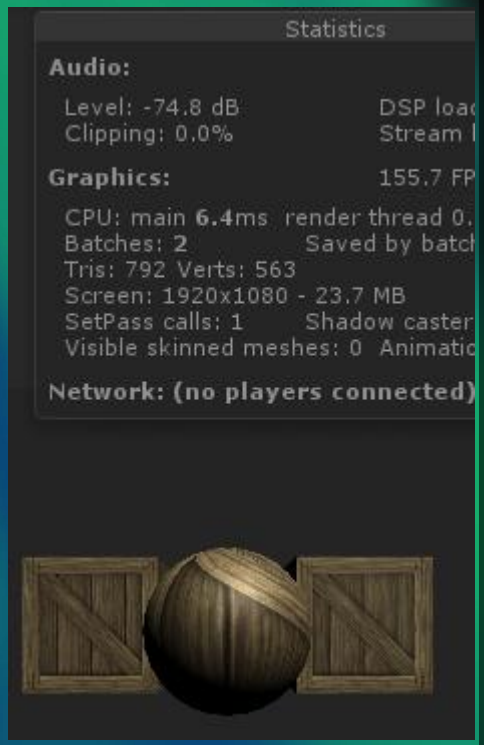
- Transforms all GObjs vertices into world space on the CPU
- Only an advantage if that work is smaller than doing a draw call

Samples

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

Hint

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



Static Batching

Setup
- Objs must have Static flag BatchingStatic
- PlayerSettings/PtherSettings/DynamicBatching OFF

Try
- 1 sphere 1 material
- 2 spheres 1 material (Offline and in PlayMode)

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

- Can be used on meshes of big sizes, which Dynamic Batching cannot provide

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 (wit would cause runtime overhead). To force it, use
  - StaticBatchingUtility.Combine(this.gameObject);
- Memory cost
  - If N batched meshes are unique > this costs no additional memory
  - If N batched meshes are the same > this costs N times more memory
    - E.g marking trees as static in a dense forest level can have serious memory impact

Hints

- Draw Call savings are not immediately visible from the Stats window until runtime
  - start working on Static Batching optimization early in the process of building a new Scene

GPU Instancing

Setup
- Objs must have Static flag BatchingStatic

Try
- NxNxN spheres w Dynamic Batching, NO Static Batching
- Activate UseGPUInstancing flag on the material

How it works
- The GPU is told to render the same mesh multiple times in one go. So it cannot combine different meshes or materials, but it's not restricted to small meshes
- the matrices of all spheres in a batch are now send to the GPU as an array
- We need 1 array for each batch (GPU shader uniform variables are limited to 64K)
  - 1 matrix = 16 floats = 64 Byte/Matrix.
  - We need Obj2World & World2Obj Matrices for normal = 128 Byte/Matrix (Matrices different for each obj)
  - 64 KB / 128 B = 500 objs / batch
  - Try to confirm this with your Stats Batch # before/After GPUInstancing!

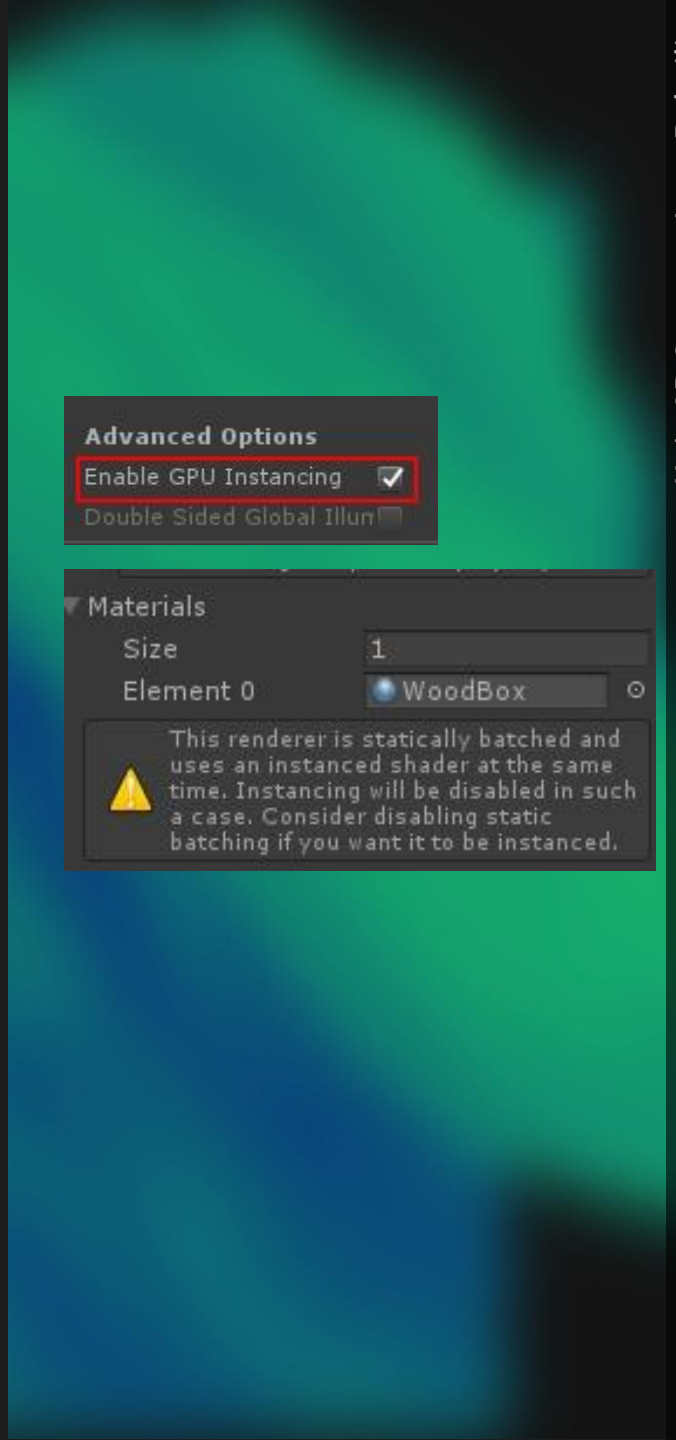
GPU Instancing

Limits

- Objs must have same Materials and same Mesh

Hints

- Static batching has priority over GPU instancing (A Warning message will appear)
- GPU instancing has priority over Dynamic batching
- Variations can be introduced via Shader code
  - E.g. we can give different instances different rotations, scales, colors, etc



LOD

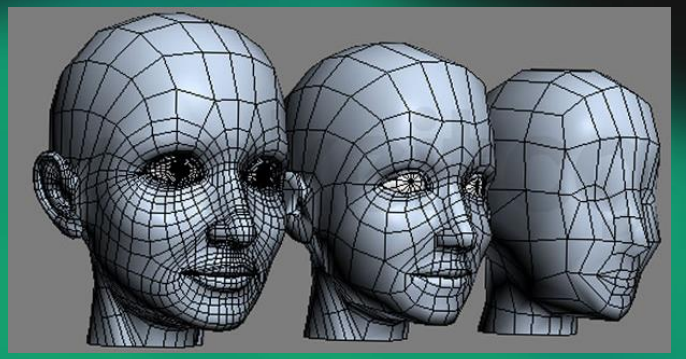
- Use different meshes depending on the view size of the object
  - May cost a large amount of development time
  - Third-party tools for automated LOD mesh generation

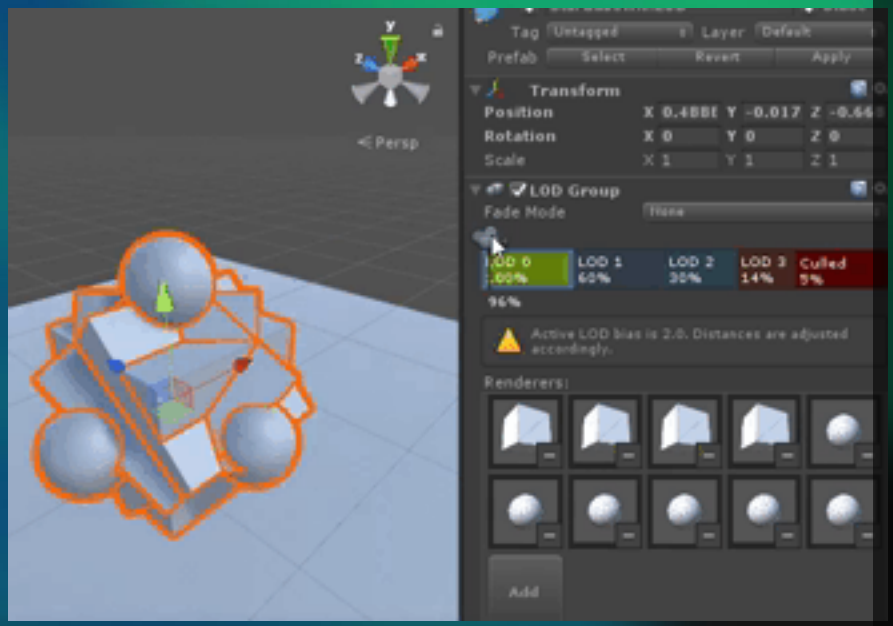
Setup

- 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

- 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

[LOD_Start.scene]





Level Of Detail (LOD)

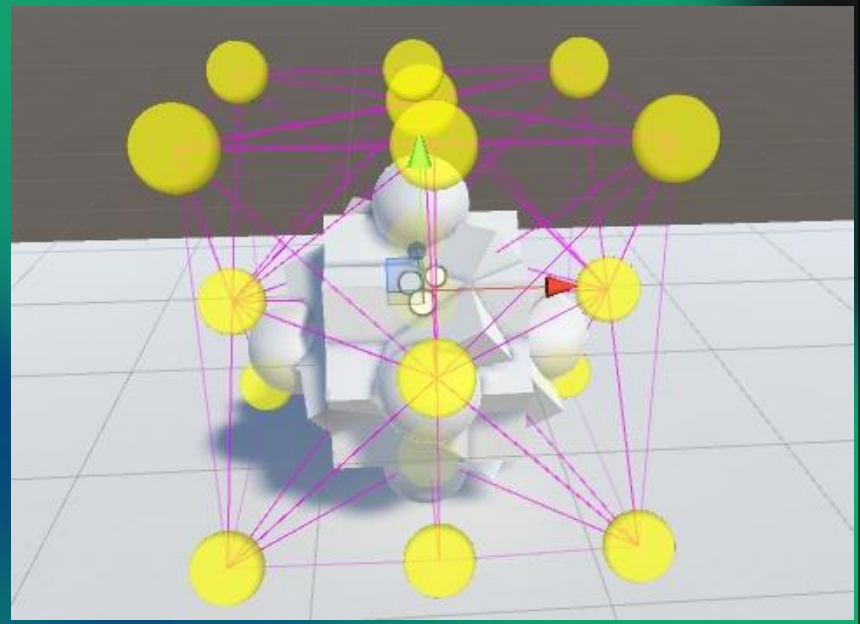
FadeMode

- 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
- Try to use CrossFadingLod Material. Its shader supports LOD (But not shadowing)
  - LOD_FADE_CROSSFADE directive
  - Use unity_LODFade.x to know the fade amount for the object

Hints

- We need LightProbeGroup for LOD objs indirect lighting
- LOD is not free: LOD meshes need to be loaded into RAM, the LODGroup Component must check Camera distance
- The benefits on the Rendering Pipeline could be impressive
- Scenes with large, expansive views of the world and have lots of Camera movement, might want to consider implementing this technique very early
- Indoor scenes or with a Camera looking down at the world will have little benefit in this technique since objects will tend to be at a similar distance from the Camera at all times

[LOD_Start.scene]

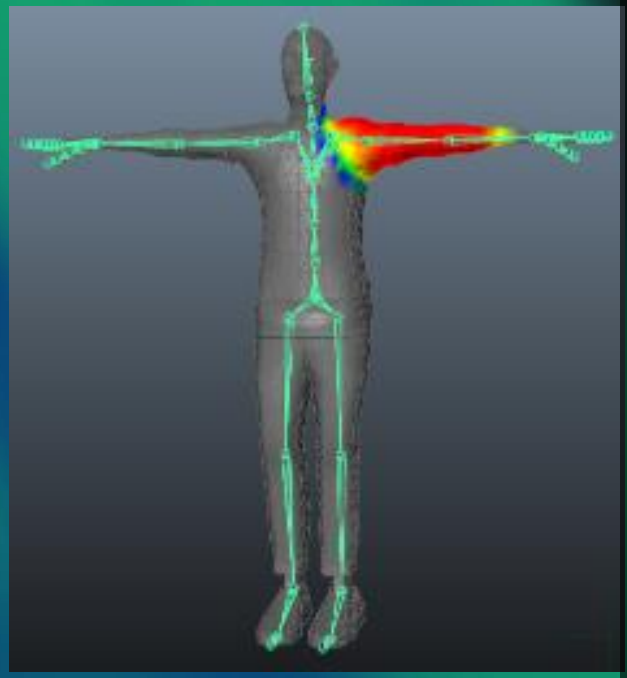


GPU Skinning

- CPU/GPU FrontEnd

- 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
- PlayerSettings/GPUSkinning
    - Check MeshSkinning.SkinOnGPU task
    - Both CPU and GPU overhead should descrease
        - CPU performs less calculation (no skinning)
        - GPU is better in skinning calculation
        - CPU<->GPU data passing is lower
    - 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

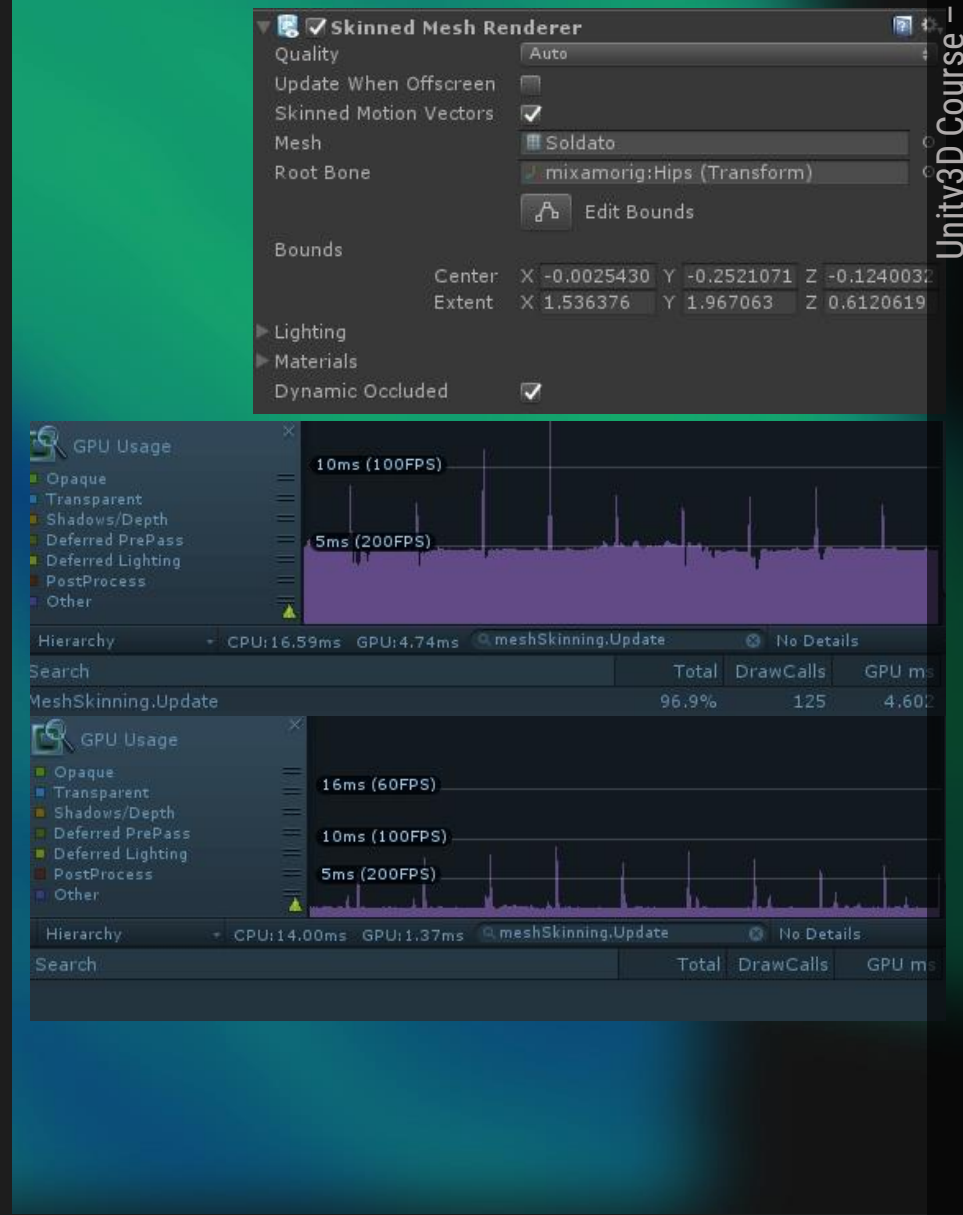
[GPUSkinning_UpdateWhenVisible.scene]



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 Blend Weights value from the Quality Settings.
- 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.
    - Search for meshSkinning.Update in CPU or GPU (if GPUSkinning is enabled) Profiler area. NB: Both Scene and Game view must be closed in order to test UpdateWhenOffscreen flag!

[GPUSkinning_UpdateWhenVisible.scene]



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

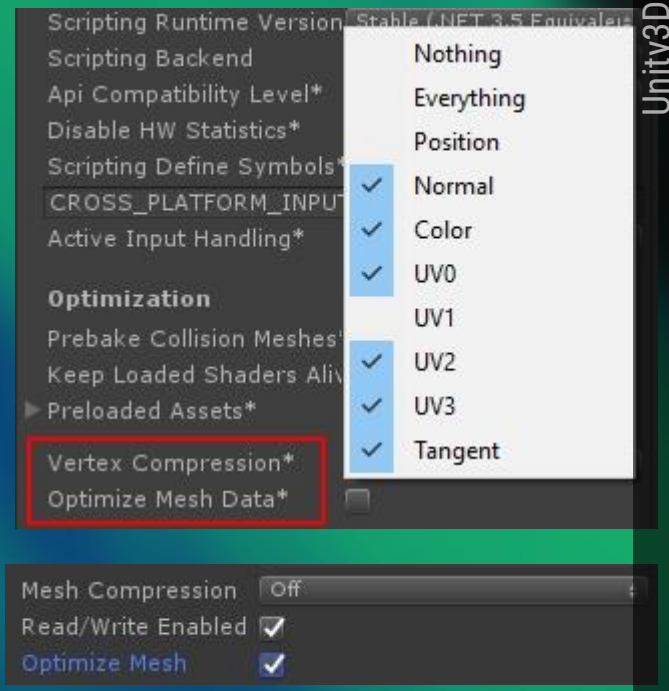
[BakeMesh_SkinnedMesh.scene]

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
- Polygon count
  - Mobile 300/1500
  - Desktop 1500/4000
  - Reduces GPU work to render the scene
- Don’t export IK nodes
  - IK nodes are baked into animations (FK)
  - Unity doesn’t need IK Nodes

Mesh Compression

- PlayerSettings/OtherSettings/[VertexCompression,OptimizeMeshData]
  - Reduce the accuracy in vertex position/Normal direction, simplify vertex color information, etc
  - Strip away any data from the mesh that isn't required by the Material assigned to it
    - E.g. if the mesh contains tangent information but the Shader never requires it, then Unity will ignore it during build time
- ImportSettings/ReadWrite Allows changes to be made to the mesh at runtime either via Scripting or automatically by Unity during runtime
  - 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

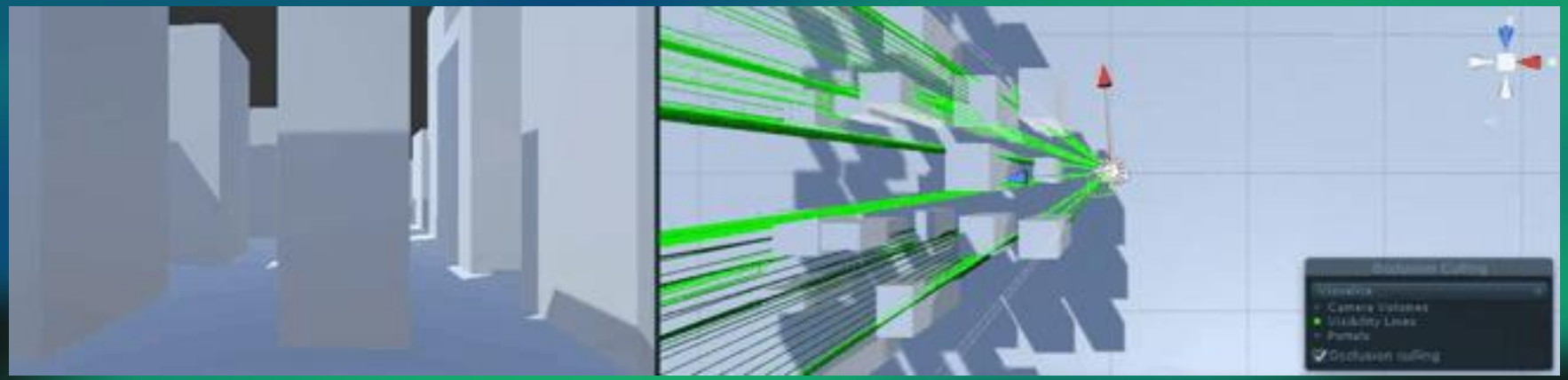


Occlusion culling

- 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

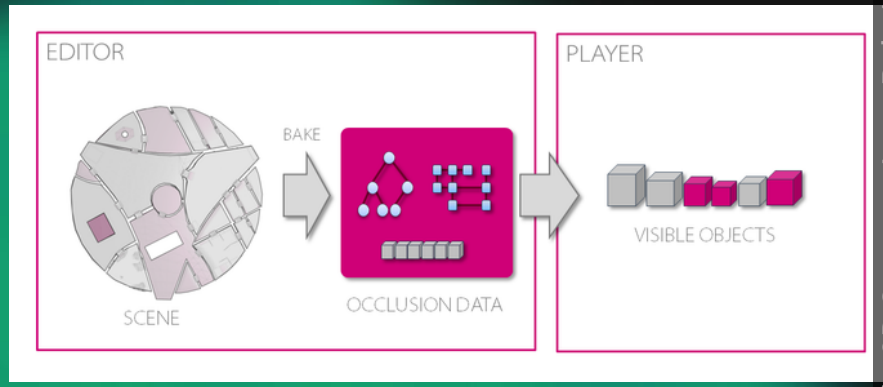
- Even though an obj may be culled by occlusion, its shadows must still be calculated

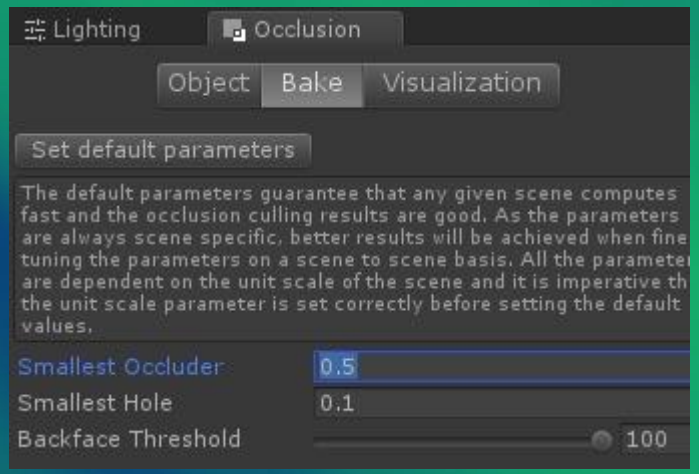


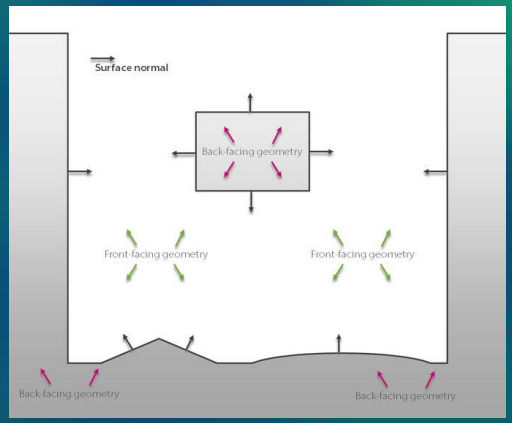
Occlusion culling

- Umbra middleware
- Scene voxelization > Voxels to cells > portals between cells > Occlusion data
- Runtime
  - Portal 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

- SmallestHole is like Umbra input resolution. [0.05, 0.5]
- SmallestOccluder is like Umbra output resolution. Larger values = faster run-time OC performances = increased conservativity (false positives) [2, 5]
- BackfaceTreshold A value of X means that all scene locations, from which over X% of the visible occluder geometry doesn't face the camera, can be stripped away from the occlusion data
  - Start with 90 and increase if you notice artifacts







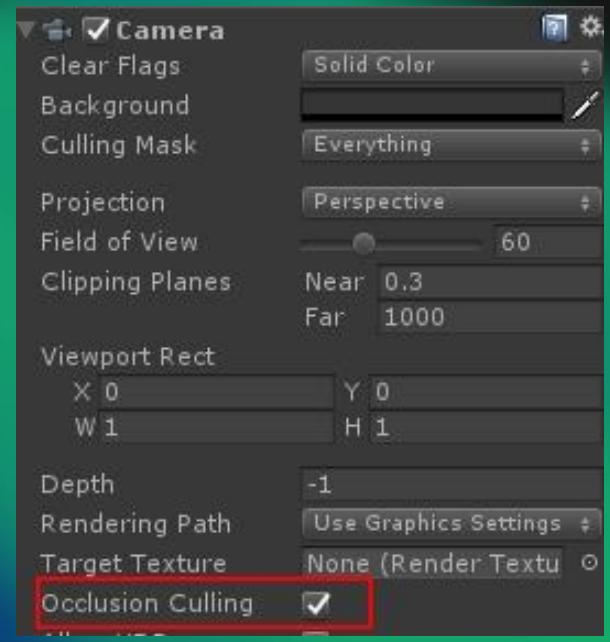
Occlusion culling setup

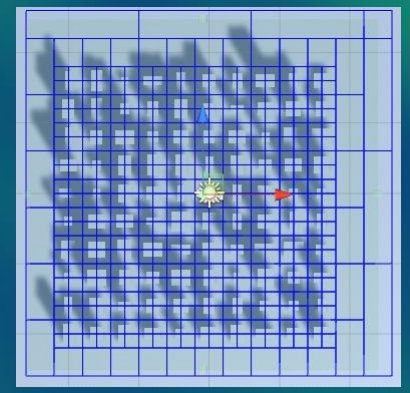
- Rendering Camera must have OcclusionCulling flag ON

- 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
  - Only way to occlude moving objects
  - isViewVolume
    - ON Camera can be inside this Area
    - OFF Camera can only look at this area

- Try Static city spawner
- Try Dynamic city spawner (w & w/o Occlusion Area)

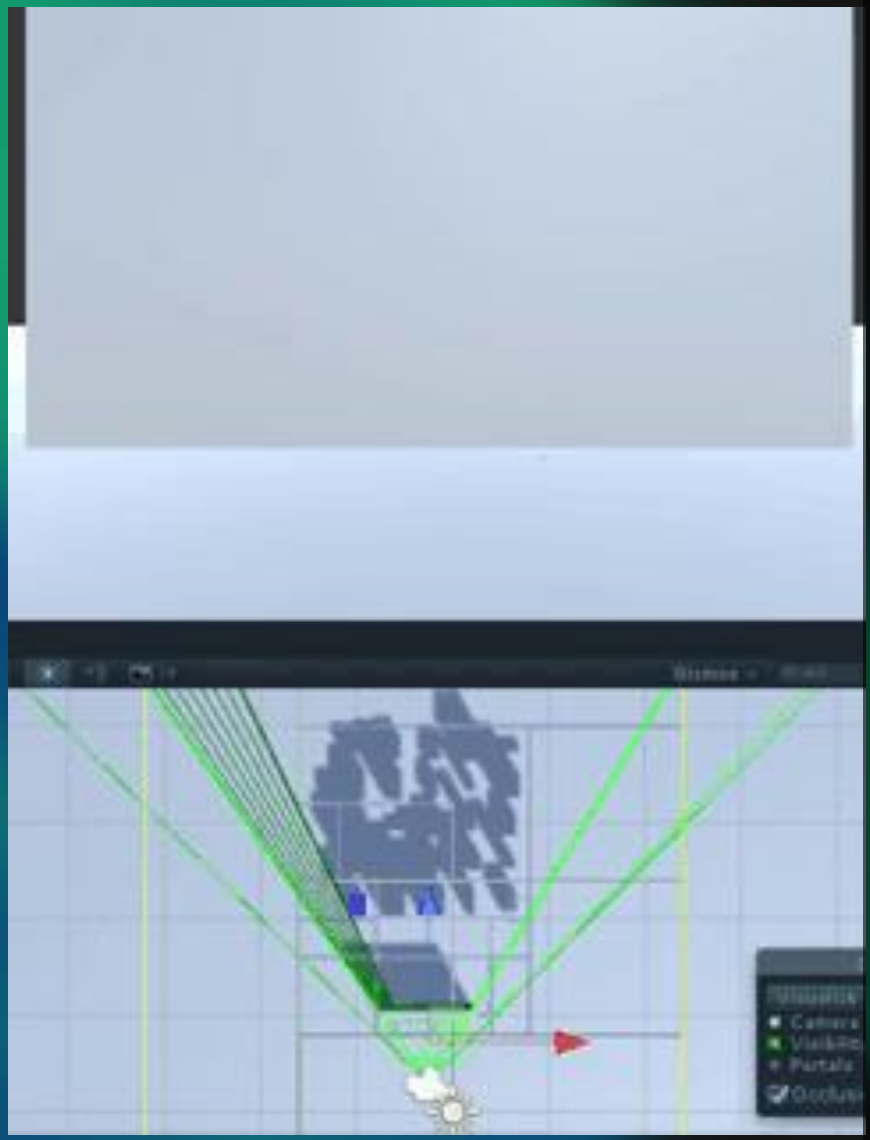
[OcclusionCulling_Start.scene]





Occlusion Portal

- OcclusionPortal
  - Use them to add dynamic Occluders into your Occlusion Area

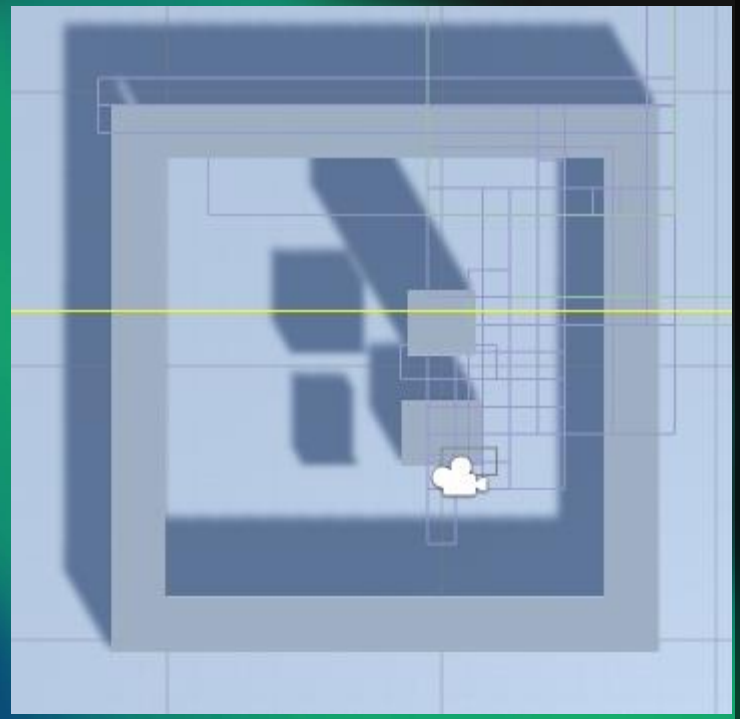


[OcclusionCulling_Start.scene]

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)
  - Camera can be inside an occluder > remove the occluder flag
- Objs granularity
  - You can use giant occluders, but obj subdivision of occludees matters
- Use Occlusion area to cull DynamicObjs
- 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

[OcclusionCulling_Start.scene]

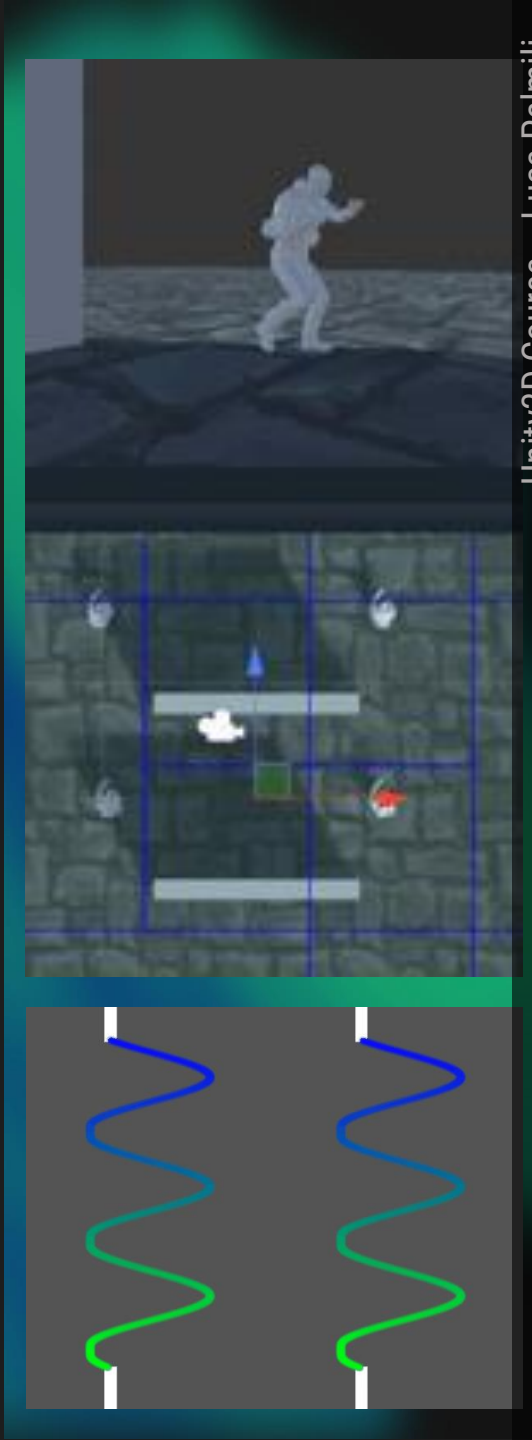


Culling groups

- Offers a way to integrate your own systems into Unity's culling and LOD pipeline
  - 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
  - Switching characters from full-quality animation and AI calculations when close, to lower-quality cheaper behaviour at a distance
  - Having 10,000 marker points in your scene and efficiently finding out when the player gets within 1m of any of them
- The CullingGroup will calculate visibility based on frustum culling and static occlusion culling only
  - Try to add an OcclusionArea

- 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
- group.SetBoundingSpheres(bounds); //This cullingGroup is able to track up to 100 BoundingSpheres
- group.SetBoundingSphereCount(1); //Tell the cullingGroup to track only the first sphere
- group.onStateChanged = OnChange; //Register our cullingGroup listener

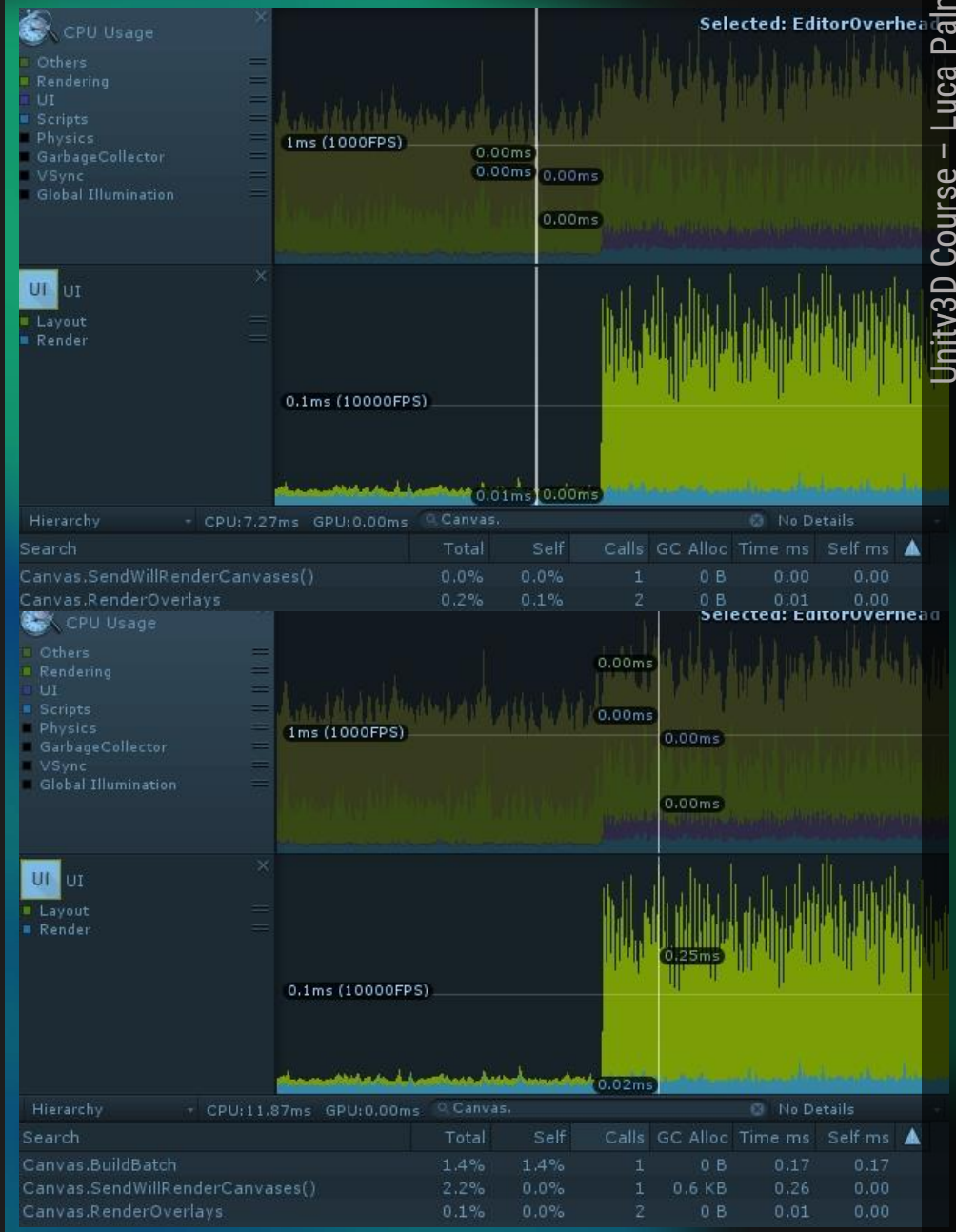
[CullingGroup.scene, CullingGroupOps.cs]



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
- When 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
- Search for Canvas.BuildBatch and Canvas.UpdateBatches
- Canvases (or sub-canvases) are independent
  - See CanvasSingle vs CanvasMulti in [UI.scene]

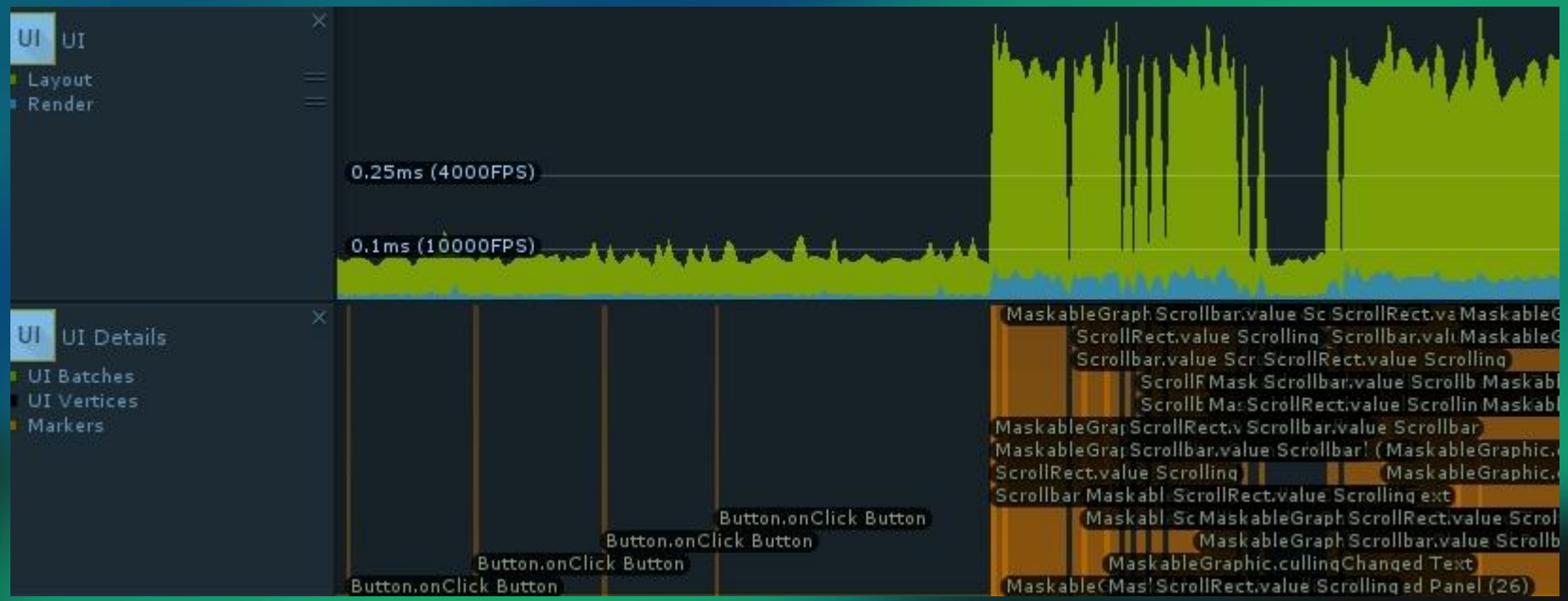
[UI.scene]



UI

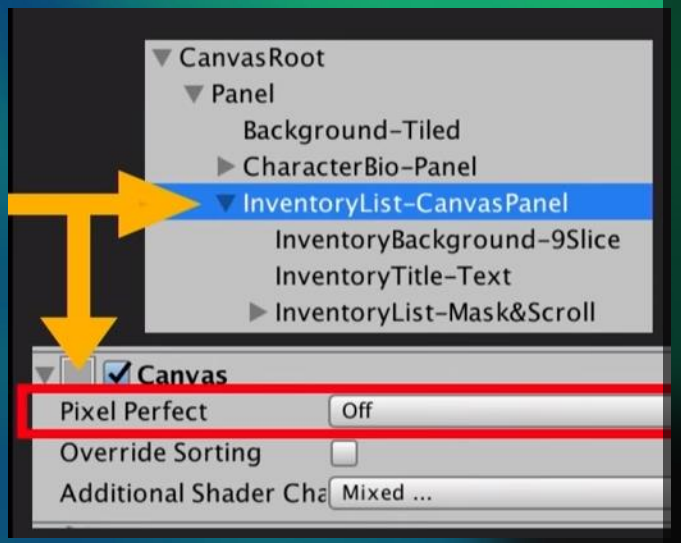
- UI shows time passed in UI layout calculation and render time
- UIDetails shows event markers, to help you determine what caused a CPU spike (scrolling culling, ButtonClicks, etc)
- BatchViewer
  - Batch Breaking Reason
  - Each batch has its associated objs visible by doubleclick

[ScrollRect]



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, Continuous Dynamic
- If there are GPU Spikes > Fill-rate problem
  - Eliminating 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)
  - 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
  - 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

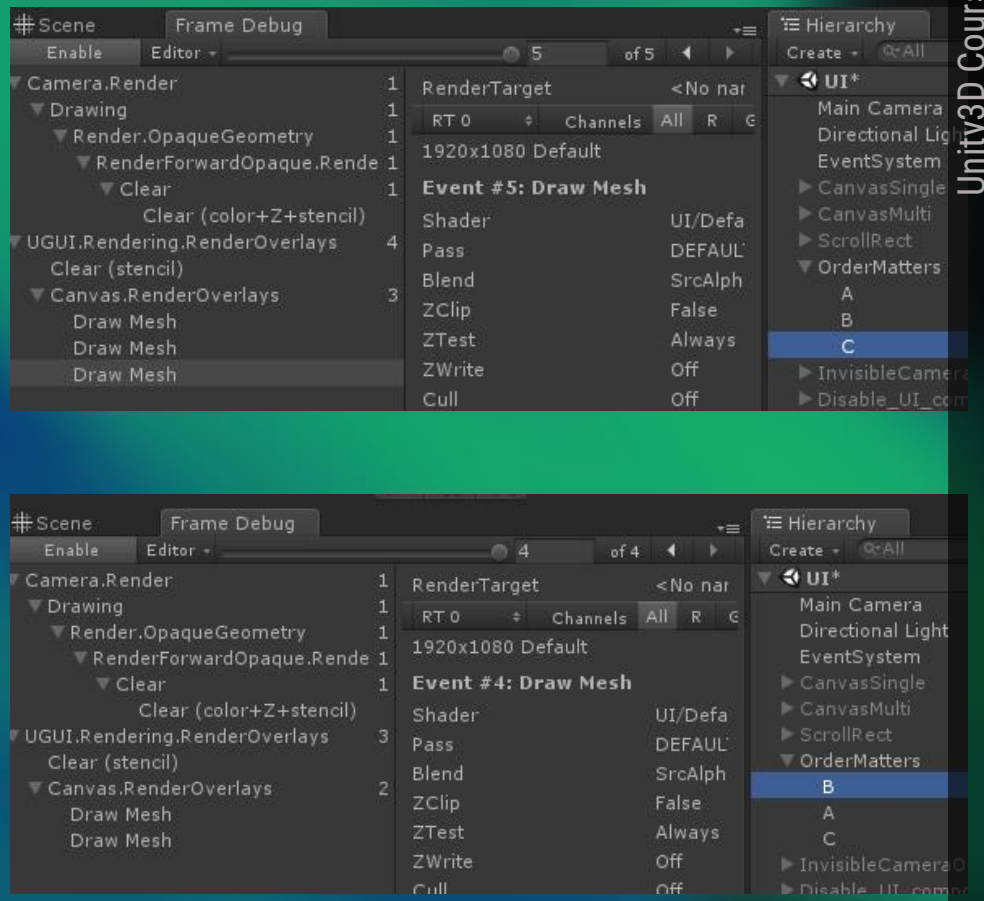


UI

- 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

- If Canvas.SendWillRenderCanvases is running every frame
  - Dynamic elements have been grouped together with static elements and are forcing the entire Canvas to rebuild too frequently

- 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

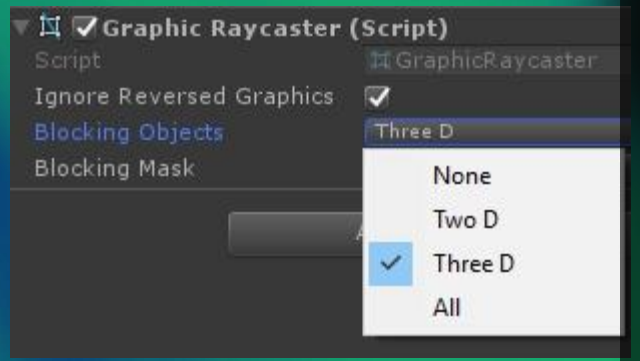


UI Raycast

- 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
- RaycastTargets list is then sorted by
  1. Depth
  2. Filtered for reversed targets
  3. Filtered to ensure that elements not visible are removed

- ignoreReversedGraphic Is used for the actual Graphic object, not for the blocking one
- blockingObjects [Raycast_Blocking]
  - 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)



UI Controls Optimization

- Always explicitly set WorldSpaceCanvas.EventCamera
  - 2D Canvas will set this property to the MainCamera
  - 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

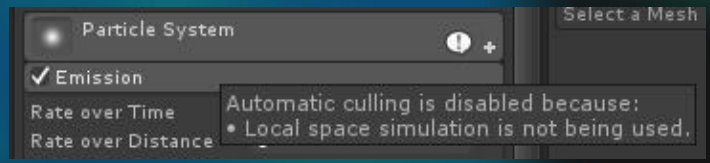
- Use RectTransform-based Layouts instead of Layout Components [CanvasSingle]

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

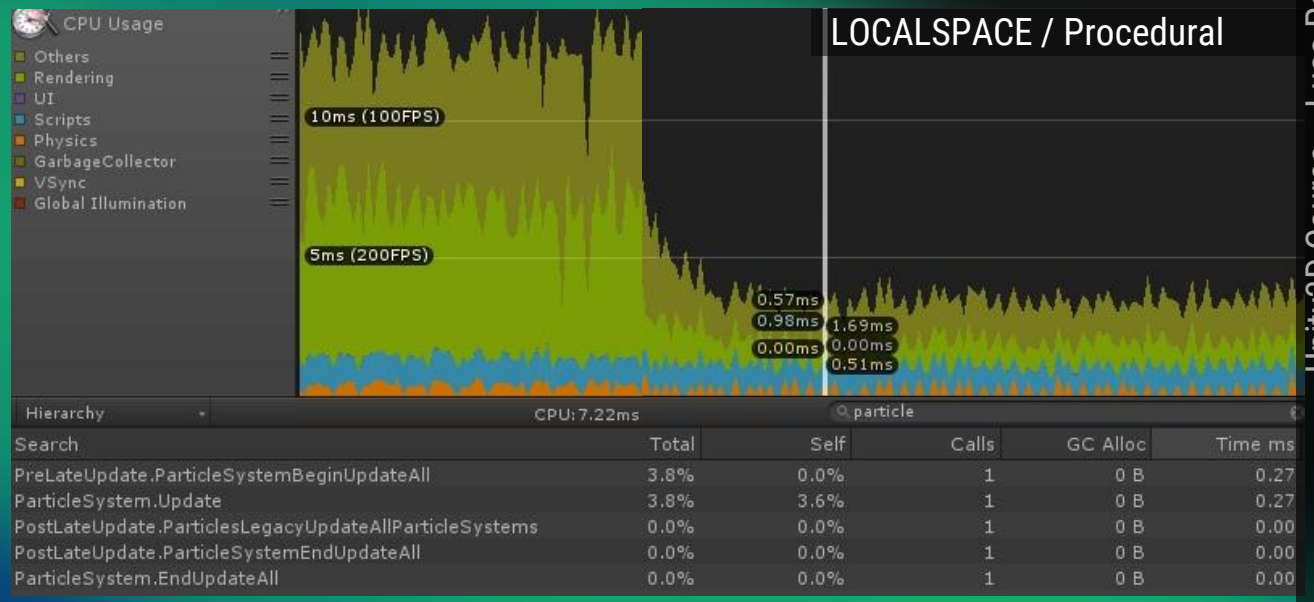


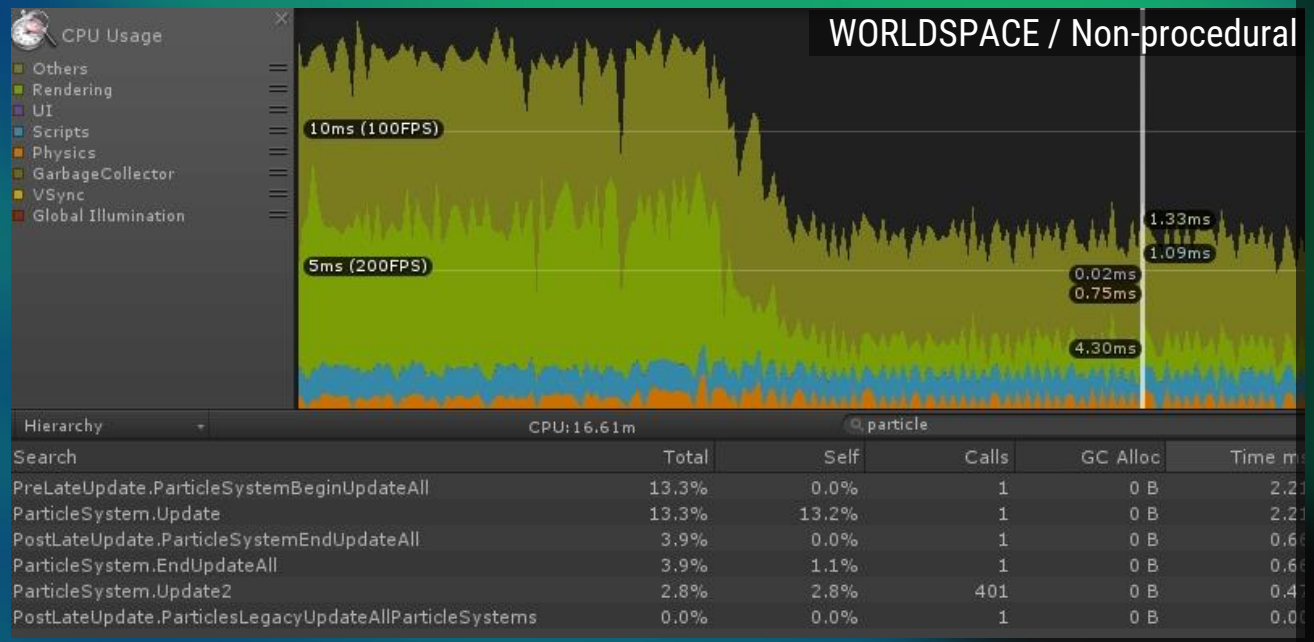
Particle Systems

- 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]



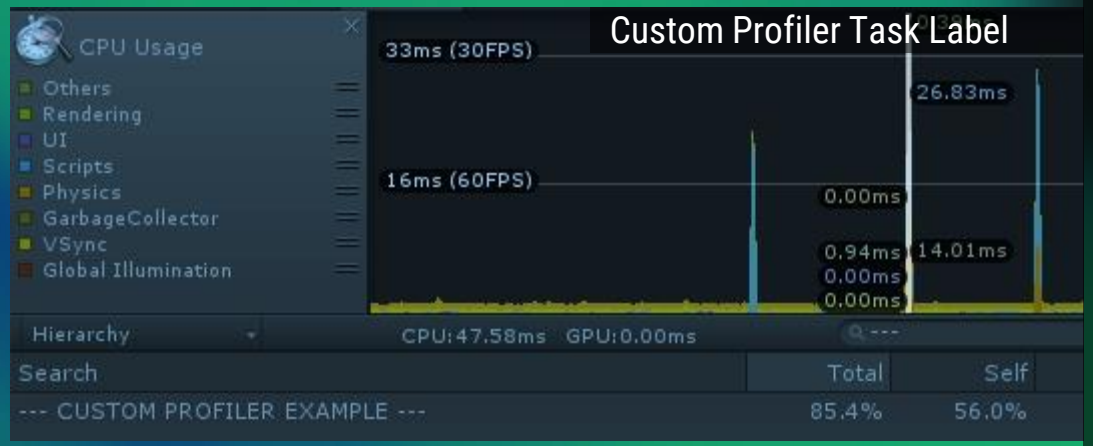


Scripting

- 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

- 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

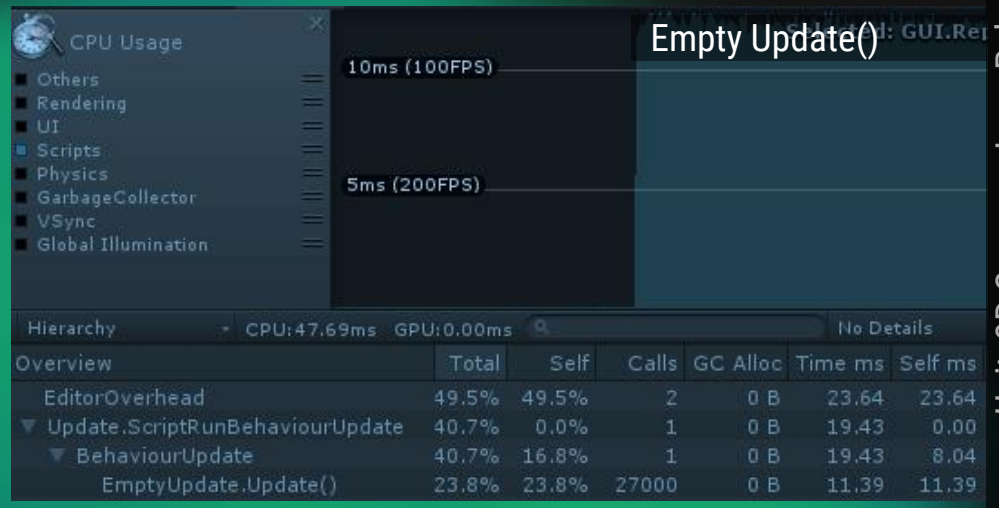
[Profiler_Scripting.scene]

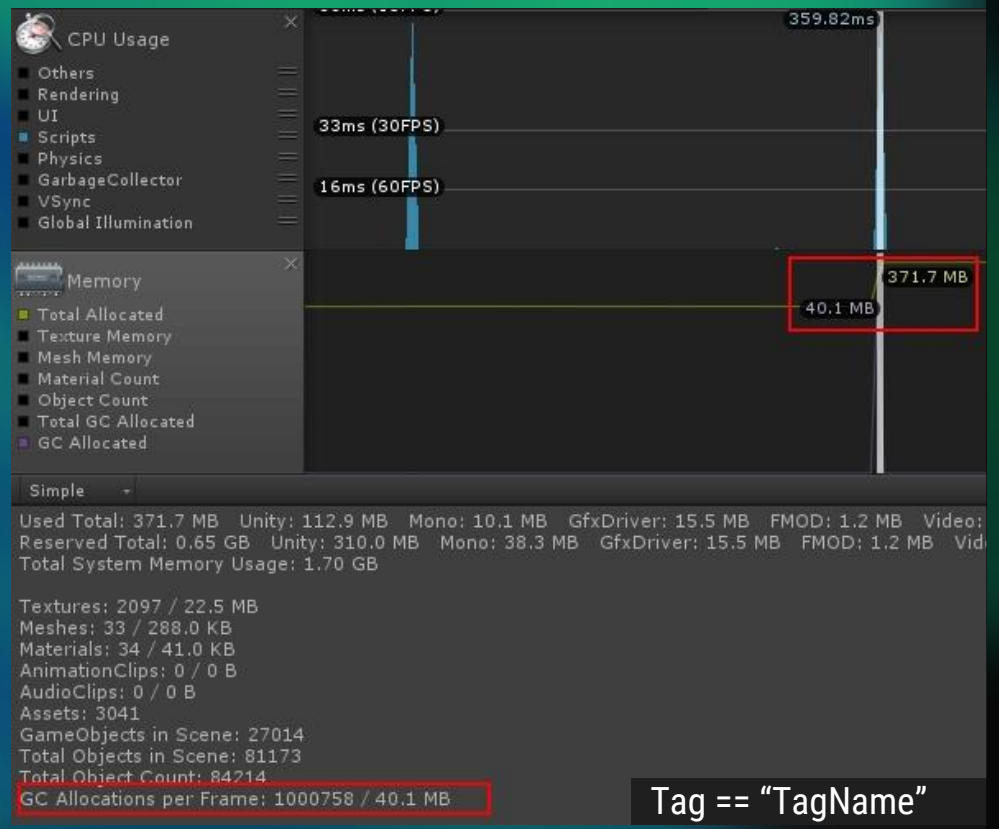


Scripting

- Avoid empty callback [EmptyCallback]
  - Regex to search for empty Start/Update callbacks:
    - void\s*Start\s*\(\s*\)\s*\{\s*\}
    - void\s*Update\s*\(\s*\)\s*\{\s*\}
- Share calculation output
- 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)
  - Retrive string property from GameObject [CompareTag]
    - Instead of goToTest.tag == "MainCamera“ use
    - goToTest.CompareTag("MainCamera")
    - 1M tag comparison = about 40MB CG Allocated memory

[Profiler_Scripting.scene]





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
- Preallocate large Transform buffer for the GObj children, using Transform.hierarchyCapacity

[Profiler_Scripting.scene]

Scripting

- 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()
  - NB
    - Disable a script disable Unity Update/FixedUpdate/LateUpdate/etc functions, not the other Unity Events function!
    - Disable the GameObject will disable everything

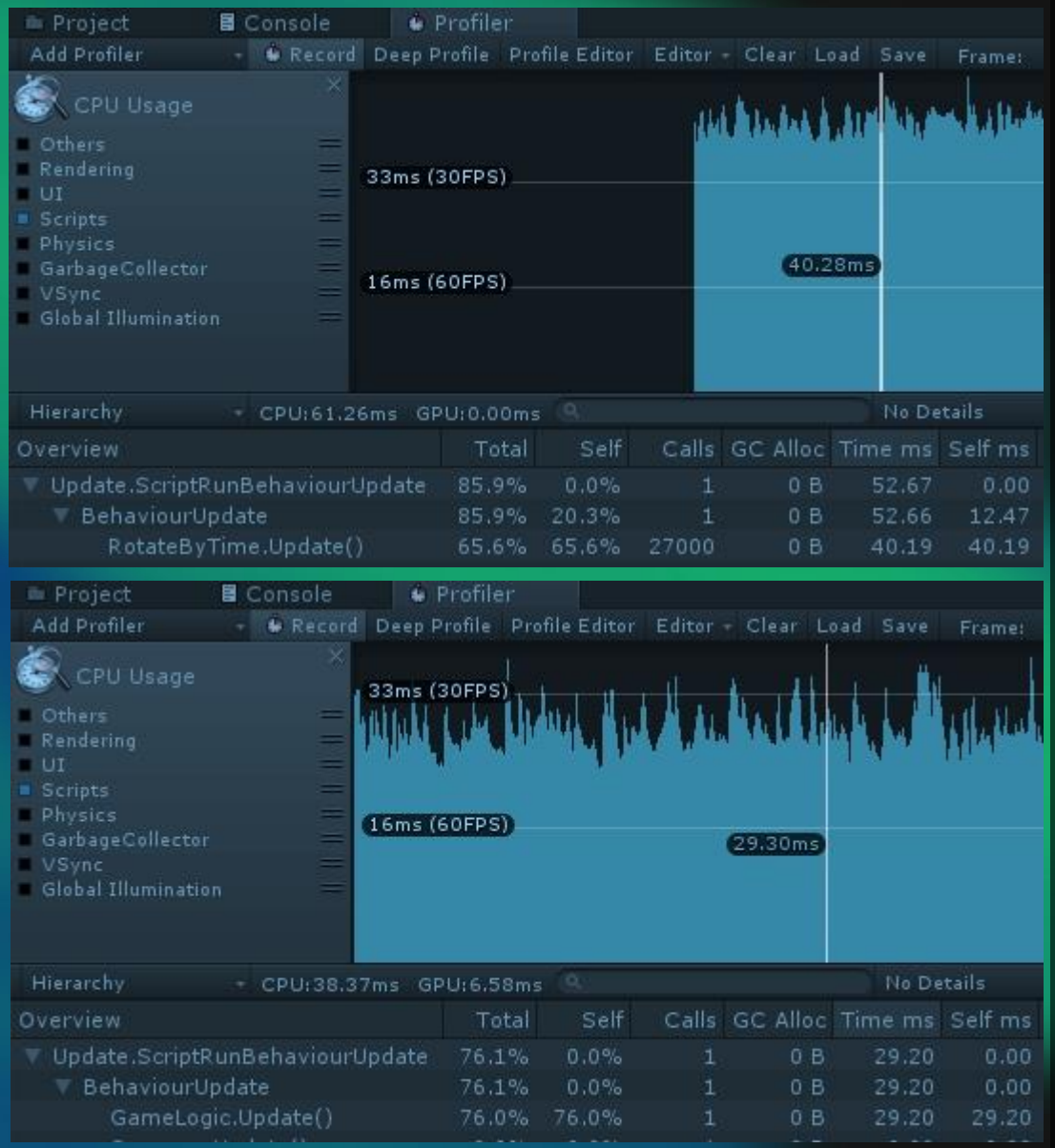
[Profiler_Scripting.scene]

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()

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



Mono

- OpenSource .Net Framework-compatible set of tools
  - Libraries
  - C# compiler
  - Common Language Runtime
- Provides cross-platform development on different hw platforms (Linux, MacOS, Windows, etc)
- Supports more languages (that can be compiled into .NET's Common Intermediate Language (CIL))

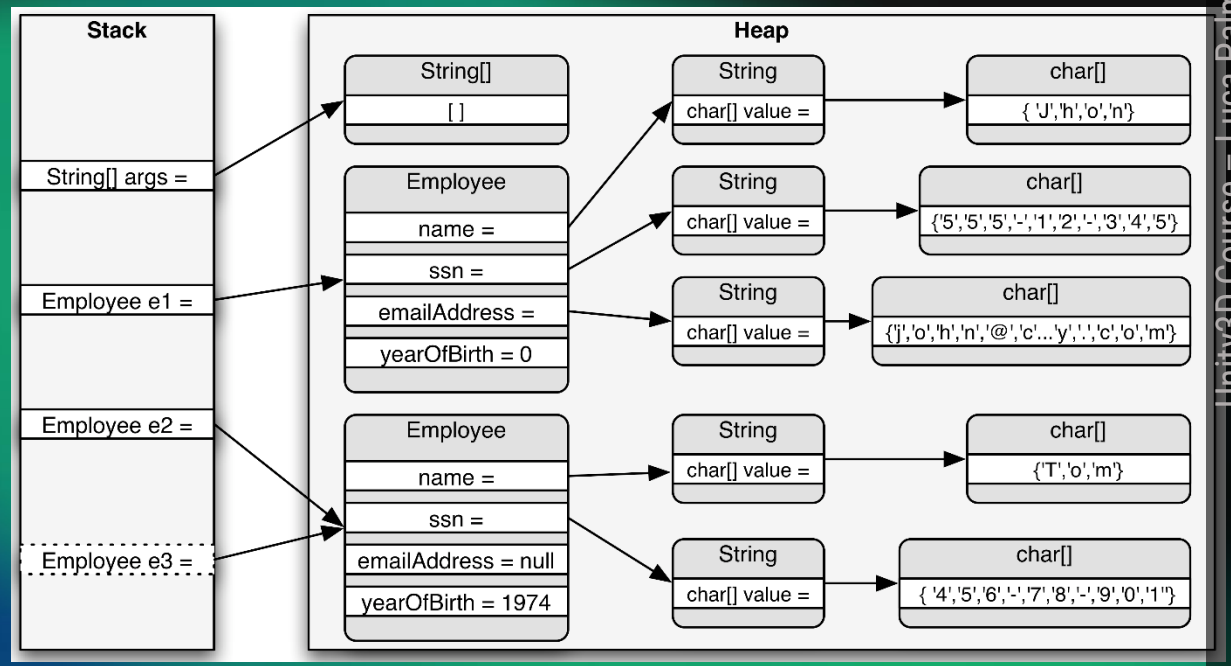
Memory domains

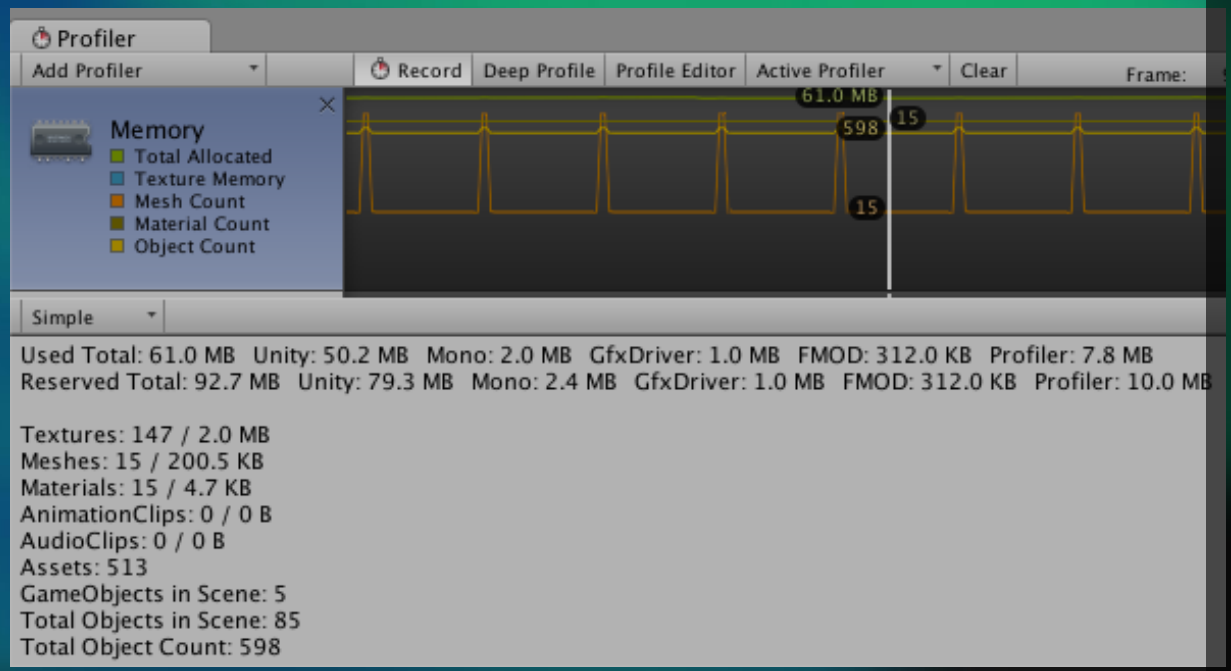
- Managed Domain
  - Mono platform
  - C# Scripts
  - Memory is automatically managed by GC
  - Includes wrappers for Native Domain GO Components
- Native Domain
  - It is in Unity Native code side (written in C++)
  - Allocates
    - Asset data (texture, audio, mesh)
    - Subsystems memory (Physics, 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.
- External libraries Domain (OpenGL, DirectX, plugins)
  - Require a memory context switch

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

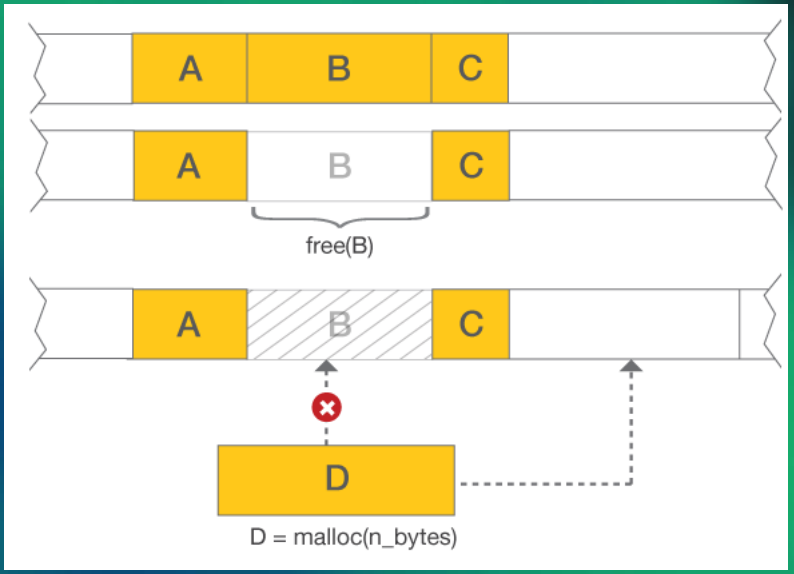
- Native Code
  - Memory allocations handled manually > Memory leaks
- Managed Languages
  - Memory allocations handled by GC
  - Mono requests at start a few MBytes to allocate Heap, which will grow and shrink if GC determines that the data is no longer needed
- Profiler
  - Unity Memory tracked by 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
  - FMOD Audio driver's estimated memory usage
  - Profiler Memory used for the Profiler data





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



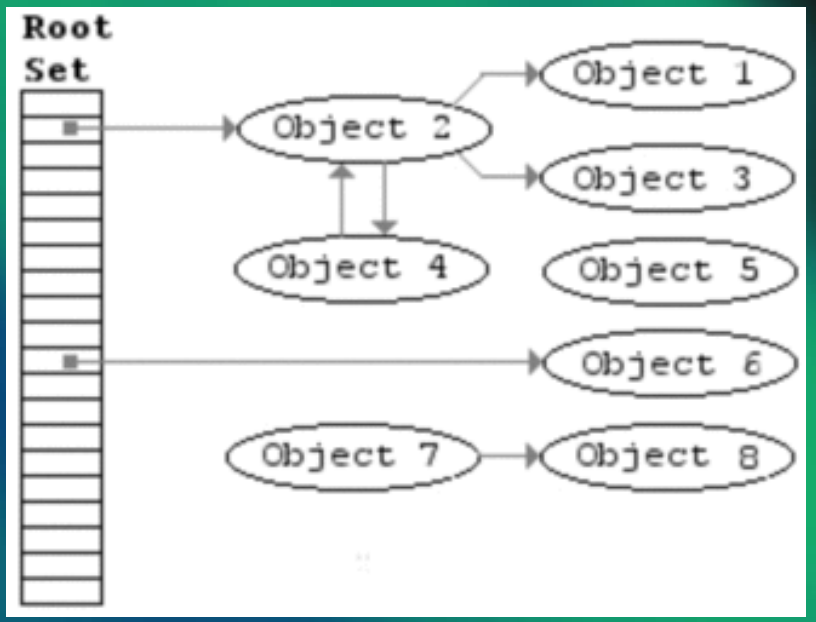
Garbage Collection

- 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
- Main thread flags MH mem blocks for deallocation
- Finalize thread perform a lazy deallocation



Code Compilation

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

- C++ is frustrating
- In C# is more difficult to introduce Memory bugs
  - GC
- 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. C# Code change > Switch from IDE to Unity > CIL
2. CIL run through MonoVM (.NET CLR implementation)
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.

```
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);
        }
    }
```

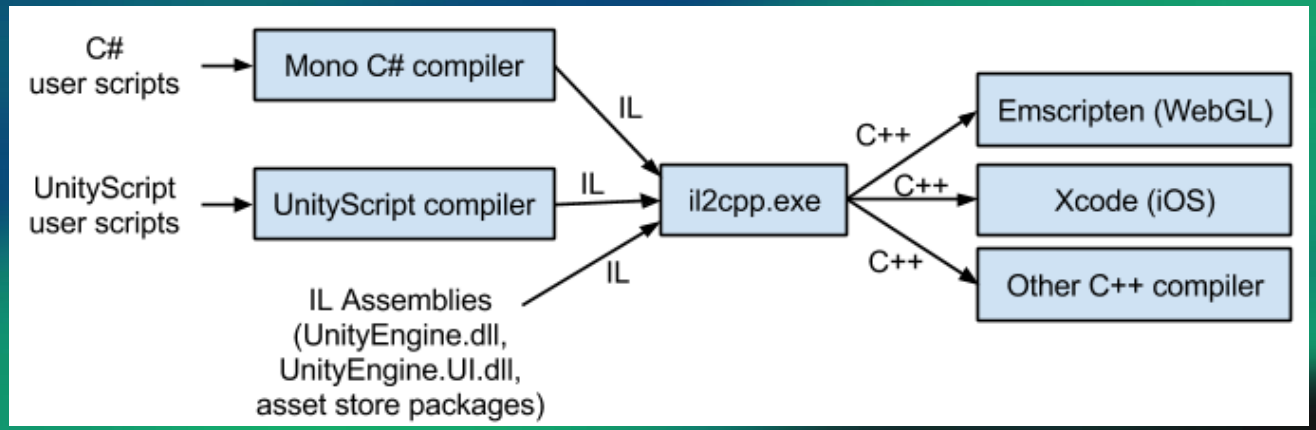
```
public void UsedOnlyForAOTCodeGeneration() {
        OnMessage(AnyEnum.Zero);

        // 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



Memory Profiling

- Memory consumption in Editor Mode is very different from Stand-alone version
  - Profiler.GetRuntimeMemorySizeLong(Object o); //Bytes allocated for Object o
  - Profiler.GetMonoUsedSizeLong(); //HeapMemory used
  - Profiler.GetMonoHeapSizeLong(); //HeapMemory reserved
- Profiler filters
  - CPU Area – 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
  - Resources.Unload[Unused]Assets(); //NativeDomain
  - Loading between levels
  - Gameplay is paused

[Memory_00.scene, MemoryProfiler]

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 [img1]
- 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 [img 2]
- We should try to replace Heap allocations with stack allocations where possible

[Memory_00.scene, MemoryProfiler]

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

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

```
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
  }
}
```

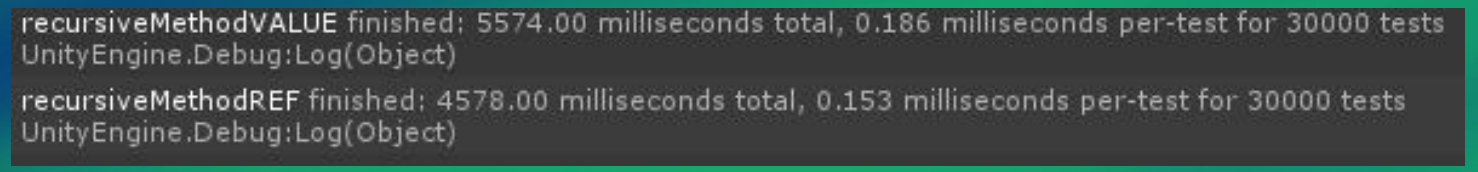
Pass by Value/Reference

- 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

- 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_val_ref]



Struct = Value types

- Struct are ValueTypes
- If we are using a class only to pass data, use a struct => Avoid heap allocation
- [img3] _memberStruct is a Value Type, even if is allocated in the Heap along with the Reference type StructHolder.

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

```
public struct DamageResult {
  // ...
}
```

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

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

Arrays = Reference types

- Arrays are ReferenceTypes

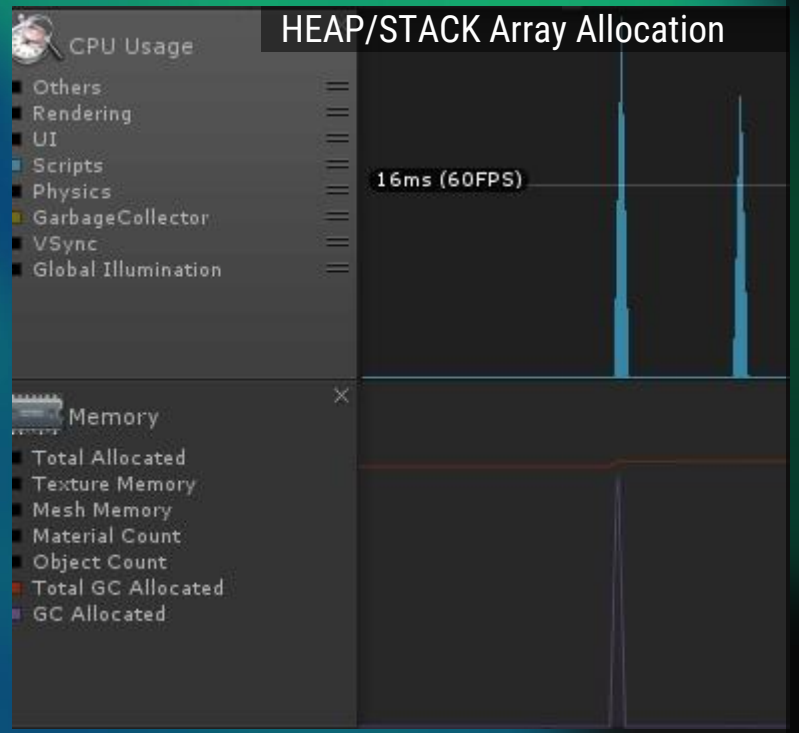
- [img1/img3 left] TestStruct[1000] is allocated in the Heap
- [img2/img3 right] There is only 1 TestStruct at time allocated in the Stack

[ArraysReference]

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

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

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



Strings = Reference types

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

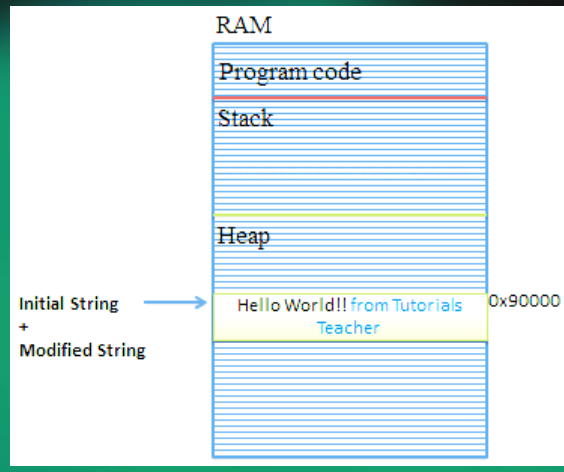
[Memory_00.scene, StringsRef.cs]

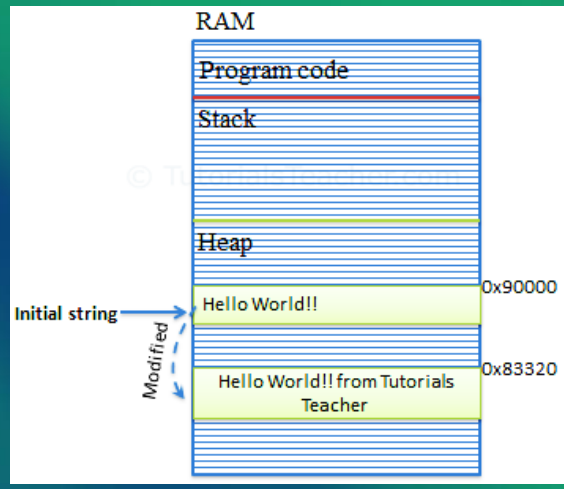
- Concatenation
  - +, or += will result in a new Heap allocation; only a single pair of strings will be merged at a time, and it allocates a new string object each time

[StringConcat.cs]

- 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());

[StringConcat.cs]





Unity API Array return type

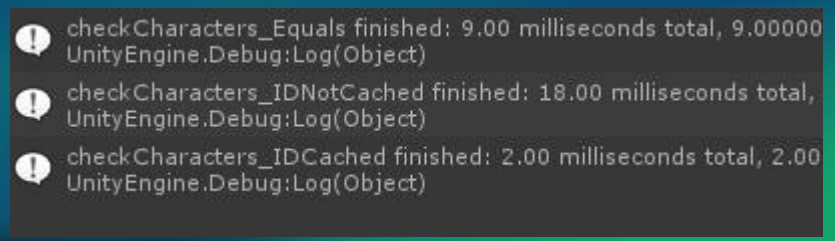
- GetComponents<T>(); //T[]
- Mesh.vertices; //Vector3[]
- Camera.allCameras //Camera[]

- 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

Use of InstanceIDs for Object comparison

- Objects comparisons can be slow (Monobehaviour, ScriptableObjects)
- 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.cs]
  - Dictionaries that ate indexed thousands of times per frame



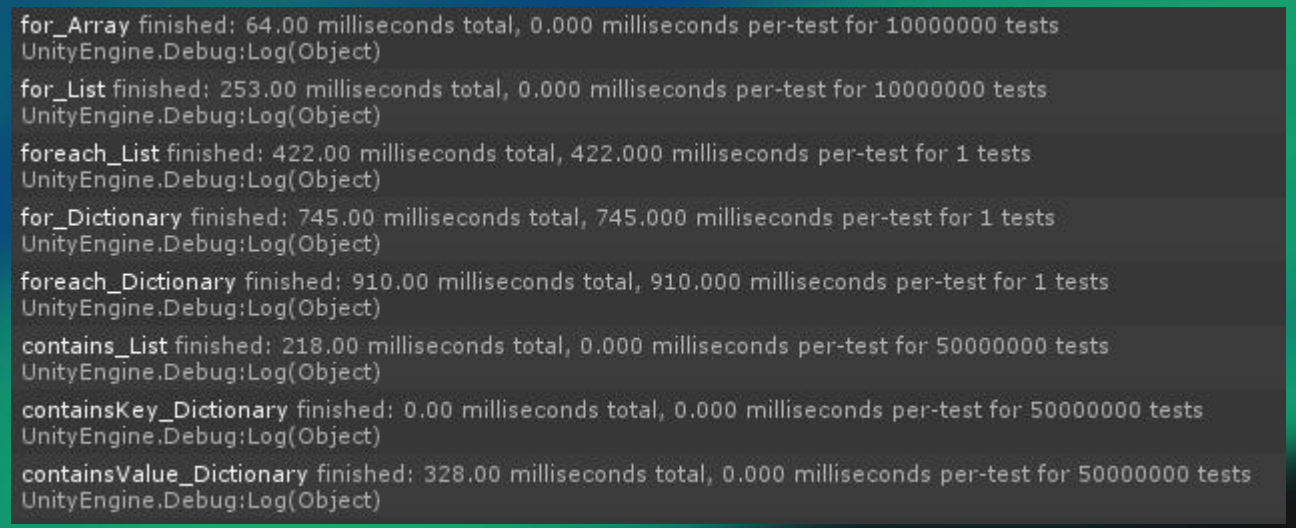
Use the correct Data structures

- Mostly iterating
  - Array, List
- Add members
  - List, Dictionary, Hashset
- Indexing/Search by key
  - Dictionary O(1)
- DuplicateChecks
  - Hashset, Dictionary (contains key = O(1))
  - ~~Array O(n)~~

[ArrayListDictionary.cs]

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

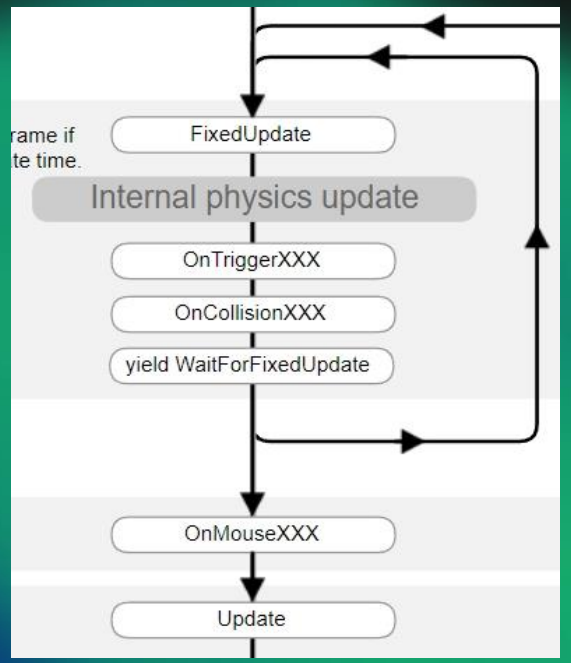
- Fast iteration (Array or List)
- Constant-time insertion (List, Dictionary, HashSet)
- Constant-time duplicate checks, to avoid register 2 times the same Iupdatable obj (Dictionary, HashSet)
- Solution? Use two data structures
  - List for iteration
  - Before changing the List, check on a HashSet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

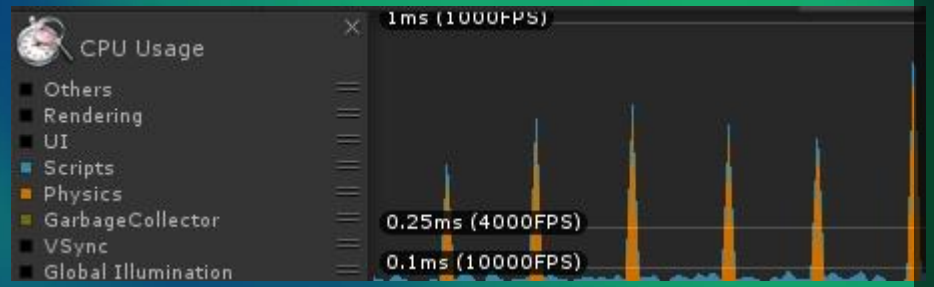


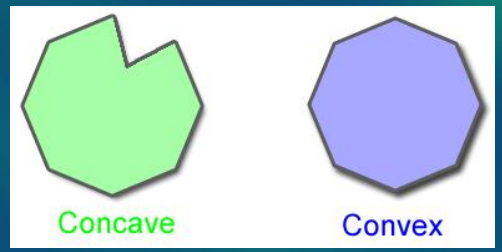
Physics

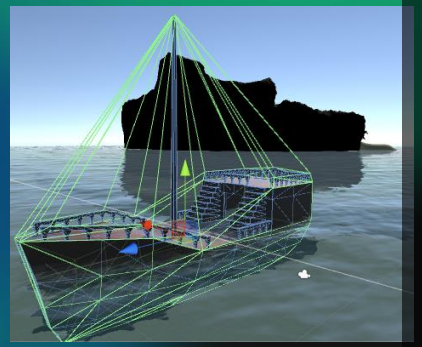
- FixedUpdates are processed just Before the Physics Engine perform its own update
  - ProjectSettings/Time/FixedTimestep
- ProjectSettings/Time/MaximumAllowedTimestep
  - Avoid “Spiral of Death” problem: Physic engine cannot escape the FixedUpdate() loop

- FixedUpdate() Useful place to perform frame-rate independent calculations and those changes that must be synched with PhysicsEngine
  - AI
  - RigidBody changes (Apply Forces/Impulses)
- Try this: Profiler Spikes = FixedUpdate()
- Dynamic vs StaticColliders
- CollisionDetection
  - Discrete, Continuous, ContinuousDynamic
  - Try this: FixedTymestep 2, Cube (dynamic collider) doesn’t collide with Plane (static collider) if Collision Detection is Discrete
- Convex/Concave colliders
  - ConcaveColliders are too expensive => Cannot be dynamic









Raycast

- 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