



The Gamedev Guru 
Game Performance Expertise

PERFORMANCE OPTIMIZATION CHECKLIST

UNITY 2021+



Index

🔧 Date:

🔧 Project:

Introduction	3
Unity CPU Optimization	4
▽ Rendering	5
▽ User Interfaces	5
▽ Scripting	6
▽ Animation	6
▽ General	7
▽ Physics	7
Unity GPU Optimization	8
▽ General	9
▽ Overdraw	9
▽ Vertex Processing	10
▽ Shaders	10
▽ VR	10
Unity Memory Optimization	11
▽ Loading Times and Memory Usage	12
▽ Build Size	12
Bonus Section	13
Performance Trade-Offs	14
How to Get 140 More Performance Tips	15



Introduction

Hey there,

I'm Ruben from The Gamedev Guru.

This is my opportunity to *thank you* for downloading this Performance Checklist for Unity 2021+

This document is the result of *years of experience* shipping games to tenths of millions of players all throughout the world.

As you know, game development is a complicated business.

That means: take the recommendations that help you and leave those that don't.

No amount of expertise will ever replace *profiling your own game*.

I hope this checklist serves you as well as it did serve me.

Ruben Torres Bonet (The Gamedev Guru)

Questions?

Send me an e-mail at ruben@thegamedev.guru

* Some of the assets I recommend here contain affiliate links. Rest assured, I recommend them because they're great and I use them continuously.





The **Gamedev Guru** 
Game Performance Expertise

UNITY CPU OPTIMIZATION



1. CPU Optimization

▼ Rendering

- ☐ Use [GPU instancing](#) only for dynamic objects that aren't transformed often
- ☐ Aim for < 100 draw calls on mobile, < 1000 draw calls on desktop
- 🔗 ☐ Disable *depth prepass* on mobile
- ☐ Restrict cameras' culling mask to the strictly required layers
- ☐ Use [occlusion culling](#) in interiors
- 👍 ☐ Oculus VR: Set [CPU hardware level](#) to 4
- ☐ All your particle systems are [procedural](#)
- ☐ Use [MaterialPropertyBlock](#) instead of instancing materials
- ☐ Disable precomputed real-time GI
- 🔗 ☐ Don't use (realtime) reflection probes
- 👍 ☐ Set light modes to [baked](#)

▼ User Interfaces

- 👍 ☐ Change materials' [color property](#) instead of multiple sprites with color variations
- 👍 ☐ Do not use [auto-layout components](#) on dynamic UI: content size fitter, layout element, horizontal/vertical/grid layout group
- ☐ If you use auto-layout components, [disable](#) them once they've done their work
- ☐ For tables, consider [TSTableView](#)
- 📱 ☐ Avoid per-frame changes in UI components to reduce *canvas rebuild* events: RectTransform, colors, sprites, text and other properties
- 🐛 ☐ Learn *UIBuilder* and [UIElements Runtime](#) for upcoming Unity releases
- ☐ Use TheGamedevGuru *canvas [rebuild detector](#)* to find and resolve canvas rebuilds
- 👍🔗 ☐ Use the new [SpriteAtlas](#) to reduce UI draw calls. Alternative: ShoeBox



1. CPU Optimization

▼ Scripting

- ☐ Do not target Mono but *il2cpp* in [master](#) mode
- 👍 ☐ Avoid using the Update function; prefer using TheGamedevGuru's [BatchUpdate](#)
- ☐ Use [structs](#) instead of classes for short-time data storage
- ☐ Disable *script debugging*
- ☐ Use *String.Empty* instead of ""
- ☐ Write C# *jobs+burst* for slow operations (>0.2ms) on multiple elements (>4)
- 👍🔧 ☐ Implement [DOTS](#) for massive amount of homogeneous elements
- ☐ < 32 bytes of per-frame *allocations*
- 👍🔧 ☐ Consider [CPU Slicing](#) to reduce per-frame CPU cost
- ☐ Don't use *Instantiate* during gameplay; prefer during loading screens for [pooling](#)

▼ Animation

- ☐ Enable [Optimize Game Objects](#) in the rigging import settings of your characters
- ☐ Reduce [blend tree](#) complexity in animations; aim for < 6 blend nodes
- ☐ Use animators exclusively for characters; avoid them at all costs in UI
- ☐ Prefer [tweening](#) and custom scripts over animators
- 🔧 ☐ Reduce *bone count* for skinning to maximum 2 on mobile
- ☐ Aim to reduce *triangle count* on animated characters to < 3k on mobile
- 🔧 ☐ Consider [impostor rendering](#) and *spritesheet animations* for distant characters




1. CPU Optimization

▼ General

- ☐ Keep [scene hierarchies](#) shallow (< 5)
- ☐ Keep [scene hierarchies](#) narrow (< 50)
- ☐ Set *static flags* on static elements
- ☐ Use [CullingGroups](#) to pause out-of-screen subsystems
- ☐ Use the [P3 Optimization Framework](#)
- ☐ Keep *packages* up to date, e.g. addressables, TextMesh Pro
- ☐ Strip [unused shaders](#)
- ☐ Automate measuring your performance continuously
- ☐ Short *SFX*: enable [decompress on load](#)

▼ Physics

- ☐ Disable [auto-sync transforms](#)
- ☐ Enable re-use [collision callbacks](#)
-  ☐ Avoid [mesh colliders](#), use instead compound colliders of simple shapes
- ☐ If using mesh colliders: enable [read/write flag](#) on its mesh importer
- ☐ Adapt the budget allocated to physics on [Time settings](#)
- ☐ Try [multibox pruning broadphase](#)
- ☐ Add more layers and minimize the [layer collision matrix](#) enabled pairs







The Gamedev Guru 
Game Performance Expertise

UNITY GPU OPTIMIZATION






2. GPU Optimization

▼ General

- ☐ Use 0 (or max. 1) real-time [per-pixel lights](#) on mobile
- ☐ Use 4x *MSAA* on (most) mobile platforms
- ☐ Full-screen *post-processing* forbidden on mobile
- ☐ [Clear](#) only with pure black color
-  ☐ Avoid *real-time shadows* on mobile
-  ☐ Avoid *multi-camera* setups
- ☐ Avoid real-time generated *render textures*
- ☐ Optimize texture space usage with color palettes and efficient UVs
- ☐ Improve GPU caching behaviour by using unique big *texture atlases* with tools such as [Mesh Baker](#)

▼ Overdraw

- ☐ Avoid *stacking* over 2 layers of UI on top of each other
-  ☐ Use the [frame debugger](#) to check if you truly render opaque geometry in *front-to-back order*
-  ☐ Be willing to disable [batching](#) if it breaks optimal object sorting
- ☐ Remove geometry [intersections](#) and *overlaps* in your 3d meshes
- ☐ Create *tighter meshes* for sprite renderers with the [sprite editor](#)
-  ☐ Favor [Sprite Renderers over UI Images](#)
- ☐ Make overdraw less expensive with [additive blending](#) instead of alpha blend
- ☐ Reduce *particle system* particles count
- ☐ Reduce *particle system* particle sizes



2. GPU Optimization

▼ Vertex Processing

- 👍 ☐ Use [LOD Groups](#)
- ☐ Use [CullingGroup](#) to disable renderers
- 📖 ☐ Use [Occlusion Culling](#) in interiors to avoid rendering invisible elements
- 👍 ☐ If programmer: use tools such as [simplygon](#) to automatically reduce vertex count
- ☐ Avoid *flat-shading* (it multiplies vertices)
- ☐ Use [GPU instancing](#) on dynamic elements to make vertex processing cheaper
- 🔗 ☐ Reduce vertices while maintaining quality by using *normal* and *displacement maps*
- ☐ <200k vertex/polygons on mobile

▼ Shaders

- ☐ Consider *baking* lighting information on the diffuse texture itself for static elements without using normal maps
- ☐ Avoid *conditional branches*
- ☐ Use the smallest *variable precision* you need, e.g. half over float
- ☐ Avoid *multi-pass* shaders
- ☐ Avoid sampling from *reflection probes*
- ☐ *Standard shaders* only on high-end HW
- ☐ Use *deferred-rendering* only on high-end desktop hardware
- ☐ Measure your *shader complexity* with tools like [Mali offline shader compiler](#)
- ☐ Don't use [GrabPass](#) on mobile
- ☐ [Pre-compile](#) shaders to avoid hiccups

▼ VR

- 👍 ☐ Use [foveated rendering](#) to reduce fragment shading and ROP complexity
- 👍 ☐ Set Oculus [GPU hardware levels](#) to 4
- ☐ Reduce [eye/render resolution](#) on emergencies
- ☐ Avoid *standard shaders* on mobile VR
- ☐ Consider *baking* lighting information on the diffuse texture itself for static elements without using normal maps
- ☐ Use [RenderDoc](#) extensively





The Gamedev Guru 
Game Performance Expertise

UNITY MEMORY OPTIMIZATION



3. Memory Optimization

▼ Loading Times and Memory Usage

- 👍 ☐ Do NOT use the *Resources* directory; Migrate to Addressables if you do
- ☐ Use an *almost-empty animated initial scene* for loading screens while you load the next scene asynchronously
- ☐ Make an exhaustive inventory of *shaders* and use a small set of unique shaders, stripping out the rest. Use tools such as [A+ Asset Explorer](#) to confirm.
- ☐ *Pre-compile* the few shaders you use
- ☐ Confirm all your assets are using the

intended optimized formats with tools such as [A+ Asset Explorer](#)

- 👍📁 ☐ Atlas your *textures* and reduce their sizes. Tools like [Mesh Baker](#) will help you
- ☐ Be selective about AudioClip import settings: do not overuse *Decompress on Load*
- ☐ Try *Load in background* and *streaming* for long AudioClips
- ☐ Learn the [Addressables system](#)

▼ Build Size

- ☐ Measure the output build size components with *build analyzers* such as [Build Report Tool](#)
- 👍 ☐ Delegate content to [CDNs](#) and download in run-time using Addressables
- ☐ Use [LZ4](#) for general fast compression with addressable asset bundles
- ☐ Use [LZMA](#) for maximum (but slow) compression with addressable asset bundles

☐ Detect and remove *duplicated content* with tools such as [A+ Asset Explorer](#)

- ☐ Disable *mipmaps* where possible to gain 33%
- 🚫 ☐ Consider *Unity Tiny* for ultra-small builds
- 👍📁 ☐ Merge materials and textures with tools like [Mesh Baker](#)





The Gamedev Guru 
Game Performance Expertise

BONUS SECTION



Performance Trade-Offs

Static Batching

-		+
↓	CPU Performance	↑
~	GPU Performance	~
↓	GPU Overdraw	↑
↓	Memory Usage	↑

SpriteRenderer vs Image

-		+
↑	CPU Performance	↓
↑	GPU Performance	↑
✗	Ease of Use for UI	✓
✓	Supports Tight Meshes	✗

CPU Slicing

-		+
↓	CPU Performance	↑
↓	Development Overhead	↑
↑	System Accuracy	↓

Addressables Usage

-		+
↑	Content Update Overhead	↓
↓	Latency	↑
↑	Memory Usage	↓
↑	Loading Times	↓



Next Steps

Dear Game Developer,

If you liked my *Unity Performance Checklist - Lite Edition*, you can find out more about my other **game performance level-up programs**:

- [Game Performance Taskforce](#): community-driven weekly lessons on game performance, including live q&a and live lessons
- [Unity Performance Checklist PRO](#) (2x the size & features)
- [The P3 Optimization Framework](#): know your next steps in optimization
- [Addressables for the Busy Developer](#): nullify loading times and excessive memory usage. Deliver your content over CDNs for 10x faster iteration times.

Ruben (The Gamedev Guru)

