Unity小游戏开发简介

2023

内容

→ 小游戏平台

→ 资源流式加载

→ Native Instant Game

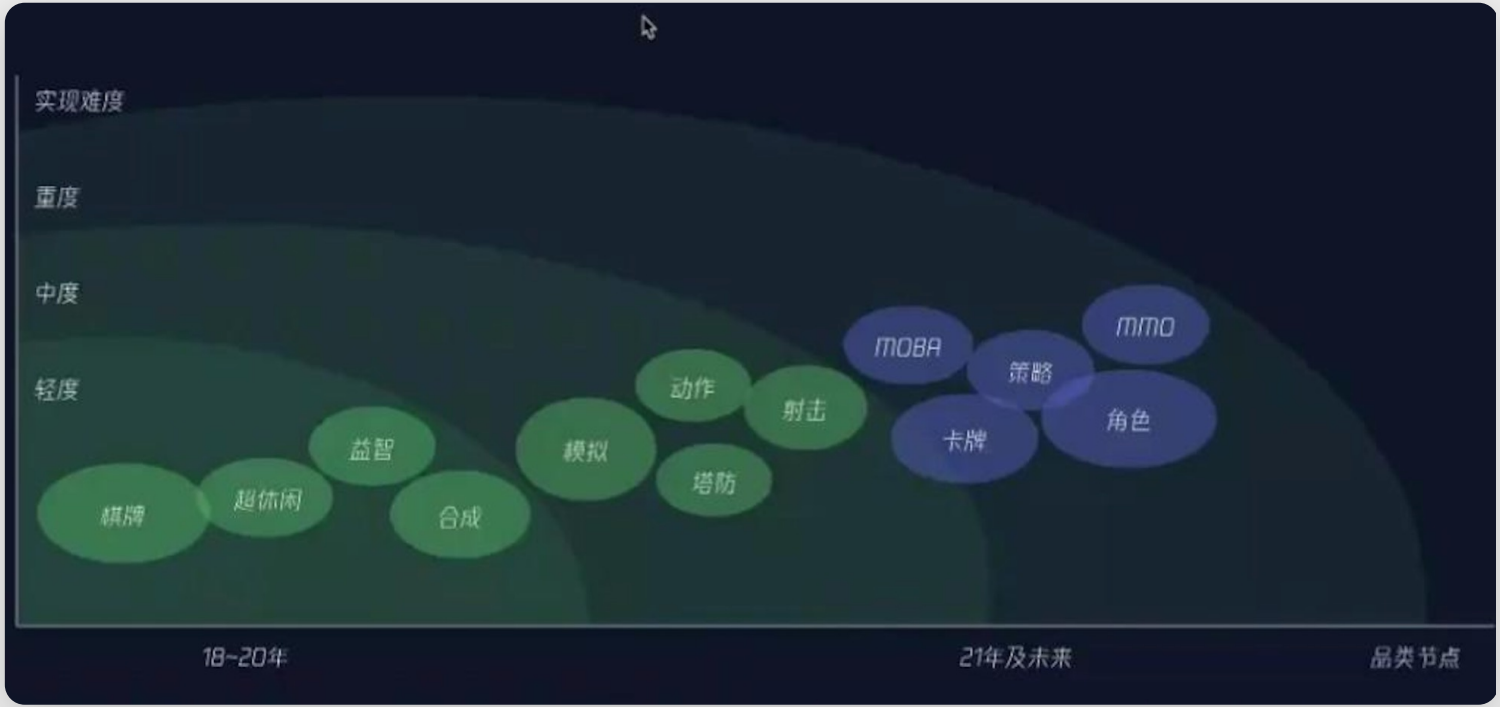
→ WebGL

→ 未来工作

小游戏平台

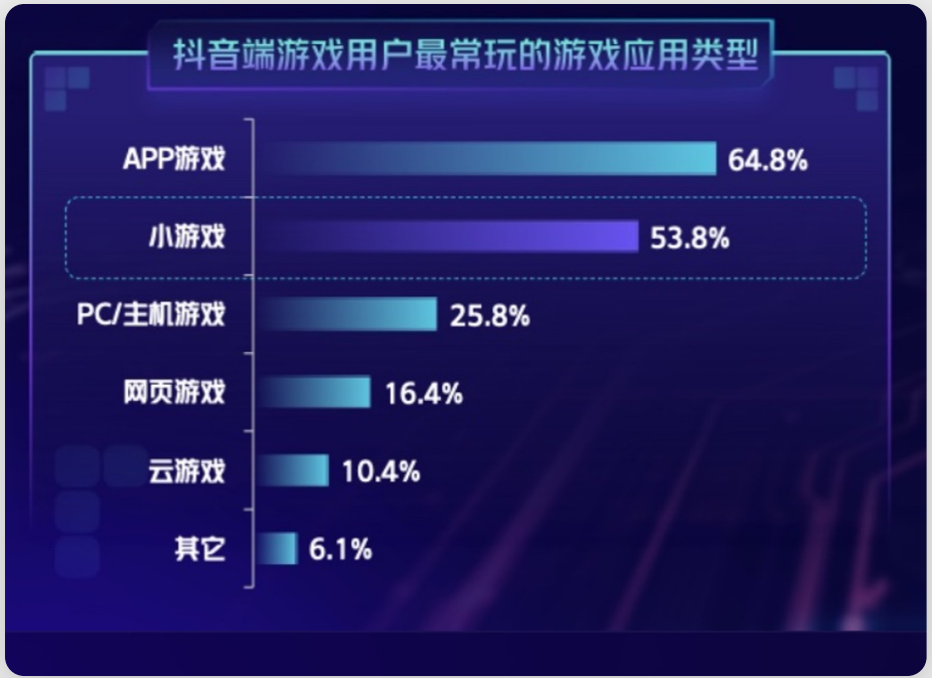
微信

微信



微信

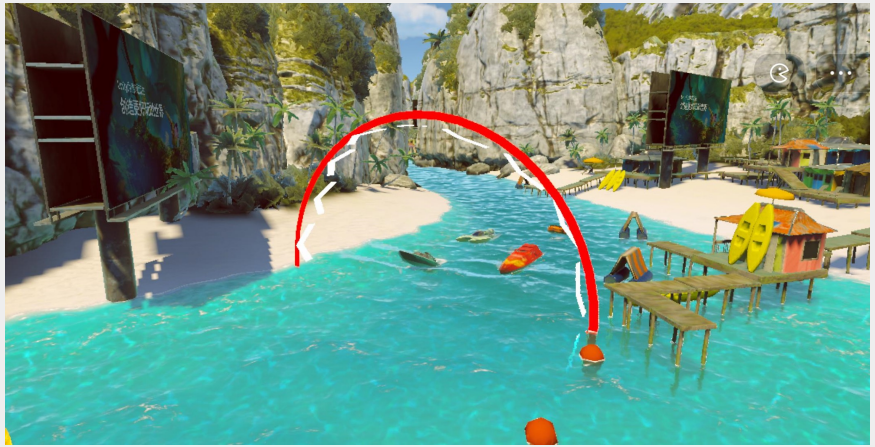
抖音



两种技术方案

WebGL

- 支持iOS、Android
- 各大平台均在使用

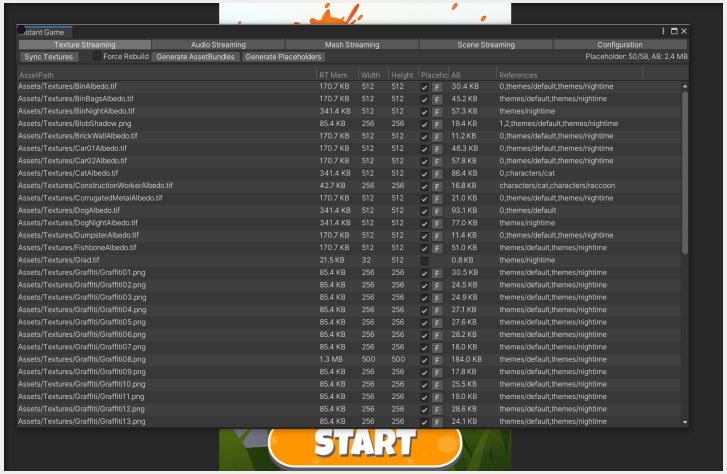


Native Instant Game

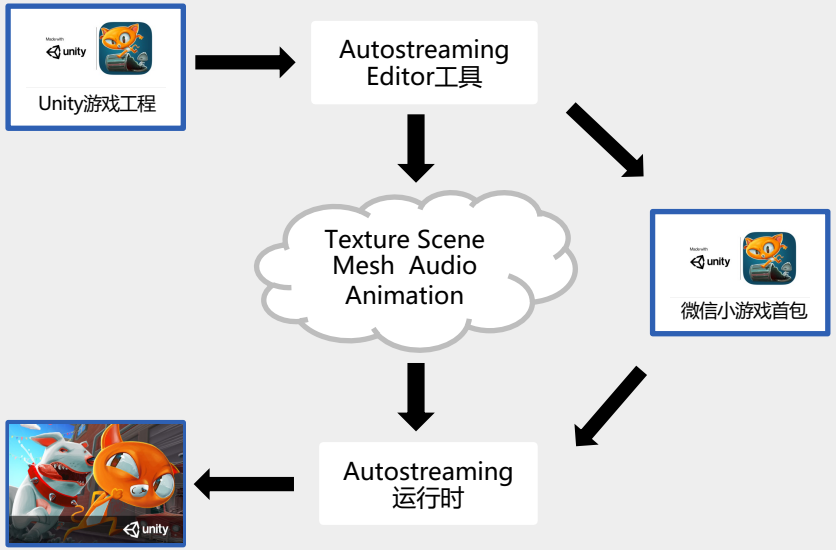
- 仅支持Android
- 原生App品质
- 已支持抖音、头条、快手
- 支付宝集成中

资源流式加载

1. 打包时自动分离重度资源



2. 运行时按需加载云端资源



优化效果

下载资源量对比

	未开启AutoStreaming	开启AutoStreaming
webgl.data压缩后	42.0MB	6.8MB
AssetBundles	50.6MB	32.5MB
启动耗时	40.0秒	7.88秒
内存占用	1265MB	1190MB

AutoStreaming云资源

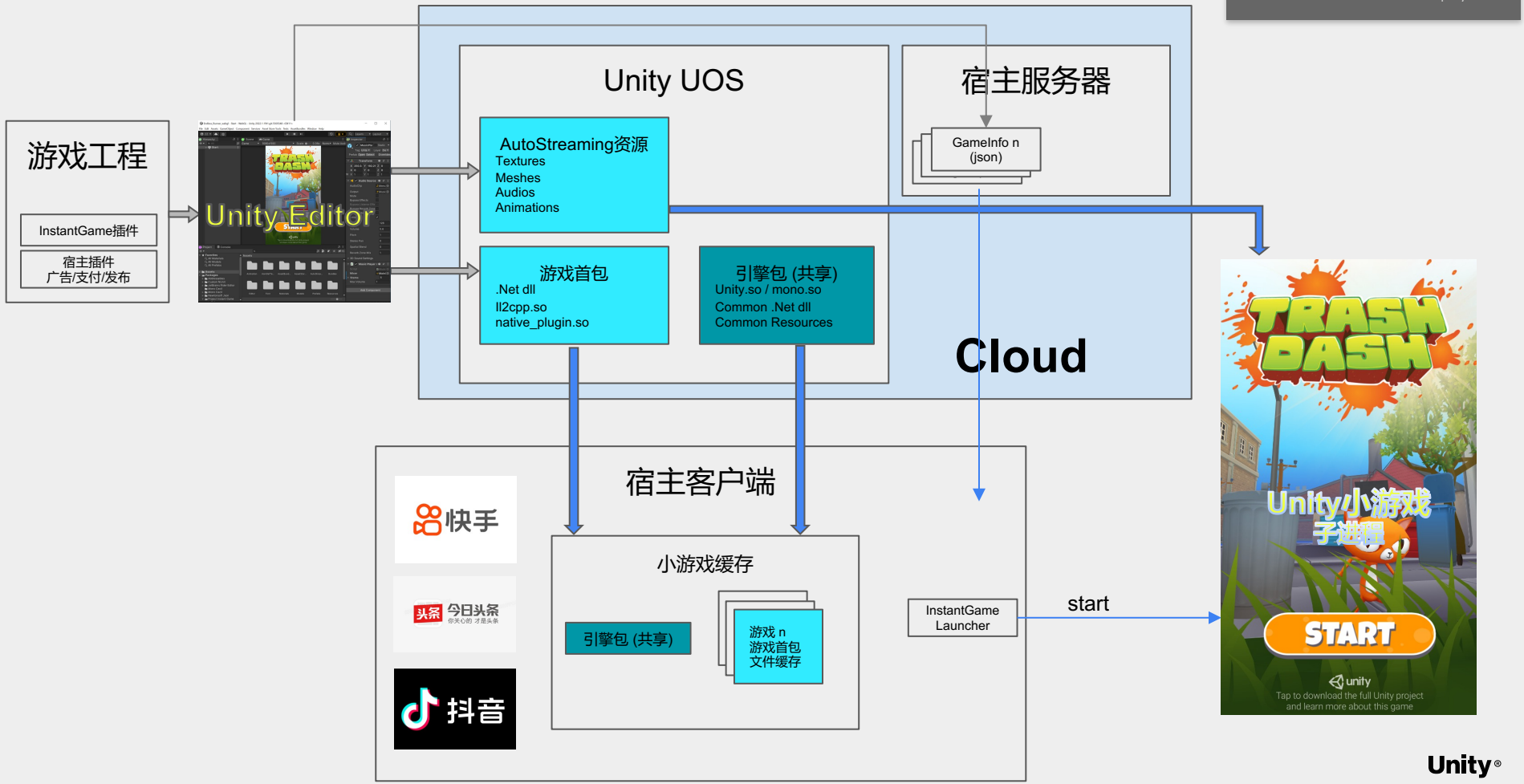
	个数	总大小
Texture	131	21.0MB
Audio	239	30.6MB
Scene	6	8MB
Animation	10	0.1MB


Native Instant Game

Native Instant Game

- 与原生App性能、体验一致
  - 支持多线程
  - 支持Gles3、Vulkan
- 兼容各种原生app能用的插件
- 可以访问沙盒中的文件
- 独立子进程运行在沙盒中
- 包含安全方案，提审、发布、更新方案
- 适配成本低
  - 只需要做好资源流式加载
  - 无需额外的适配与优化

https://unity.cn/instantgame/doc/2019/2019.4.29f1c110



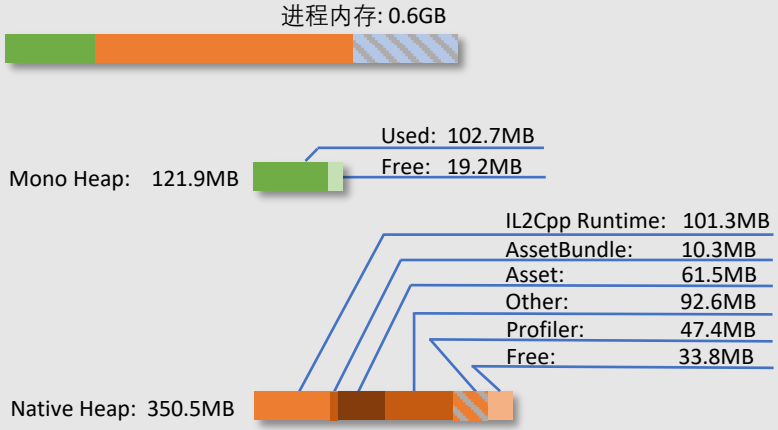
WebGL

平台特点

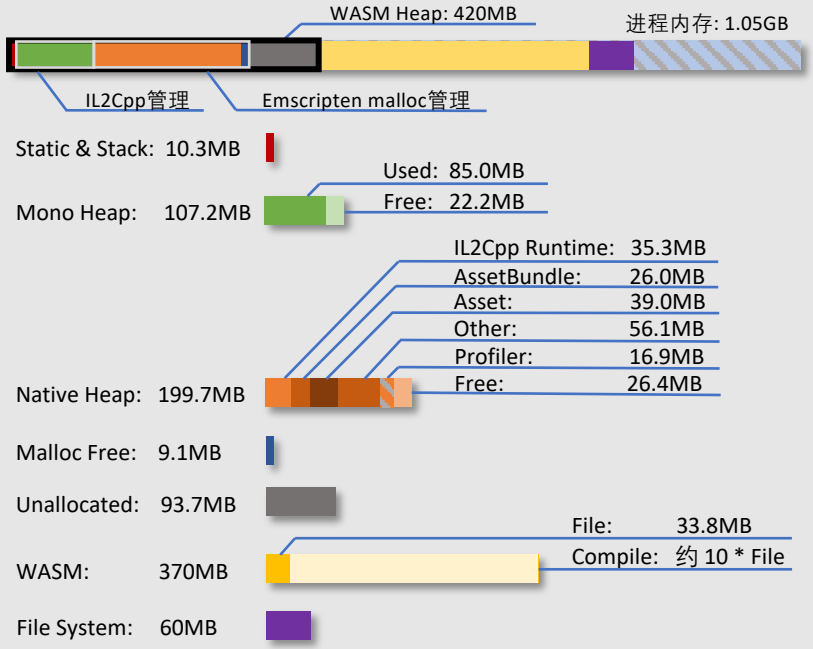
- → 内存
  - 在iOS上受限
- → CPU
  - Wasm, 单线程
- → GPU
  - WebGL1，WebGL2
- → 文件系统
  - 异步访问、费内存
- 发热

内存

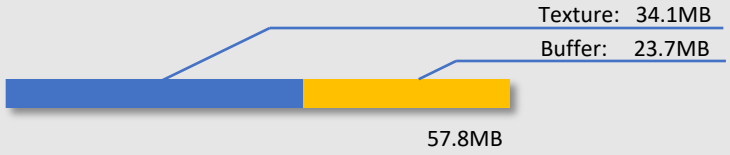
iOS 原生App

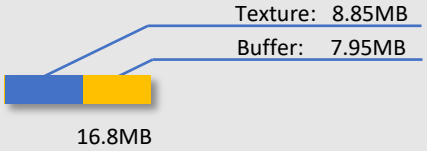


iOS WebGL (开启AutoStreaming)



显存

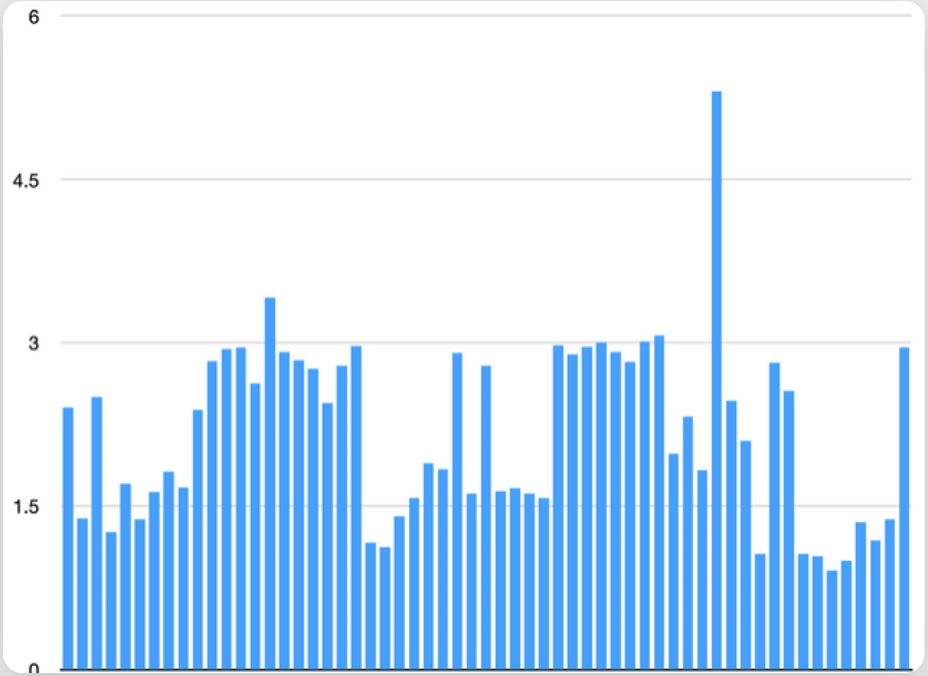




CPU

→ 3倍native耗时

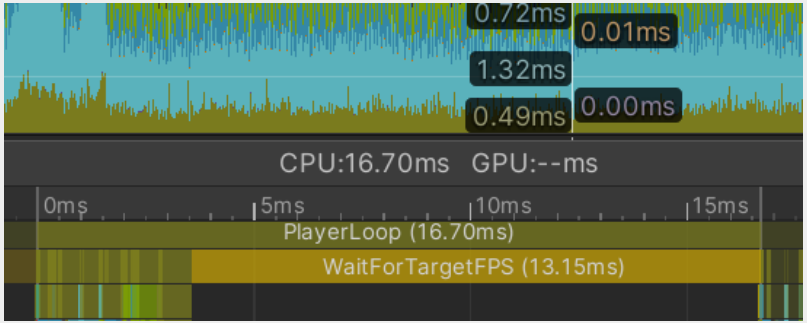
https://00f.net/2023/01/04/webassembly-benchmark-2023/

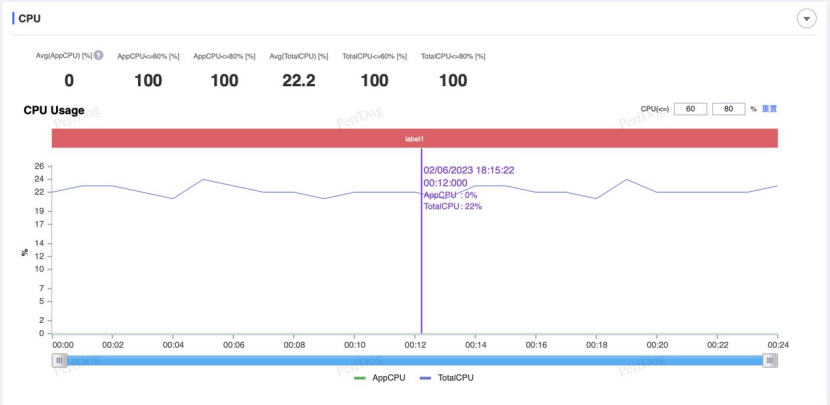


Wasm vs Native, 2023

CPU

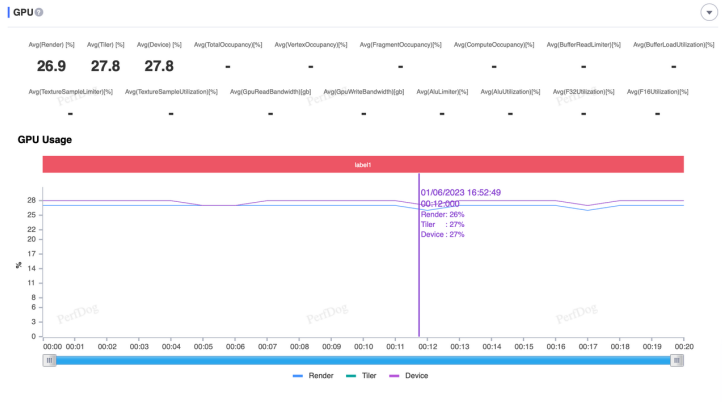
	iOS App	WebGL
单帧耗时(ms)	3.55ms	10.0ms
CPU 使用率	13.6%	22.2%





GPU

	iOS App (60fps)	WebGL (50fps)	空白网页 (30fps)
GPU 使用率	26.9%	43.2%	9.5%

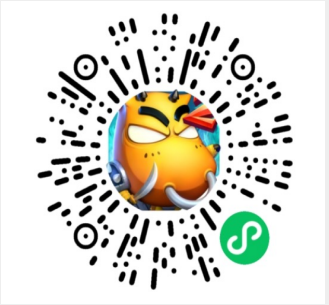


移植简介

- → 已经有大量的成功案例
- → 官方QQ群： 628540768
- → 微信WebGL适配教程

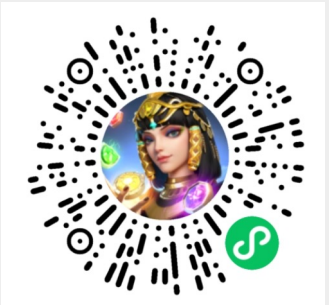
https://github.com/wechat-miniprogram/minigame-unity-webgl-transform

我叫MT2(回合战斗)

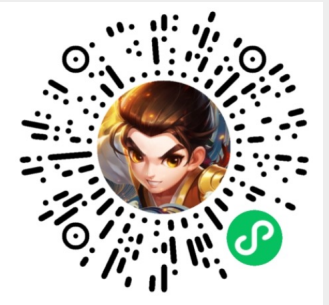


旅行串串(休闲)

谜题大陆(SLG)



热血神剑(MMO)



引擎改进

- → 优化内存占用
- → 优化绘制效率
- → 引擎代码轻量化
- → 加快启动速度

优化内存占用

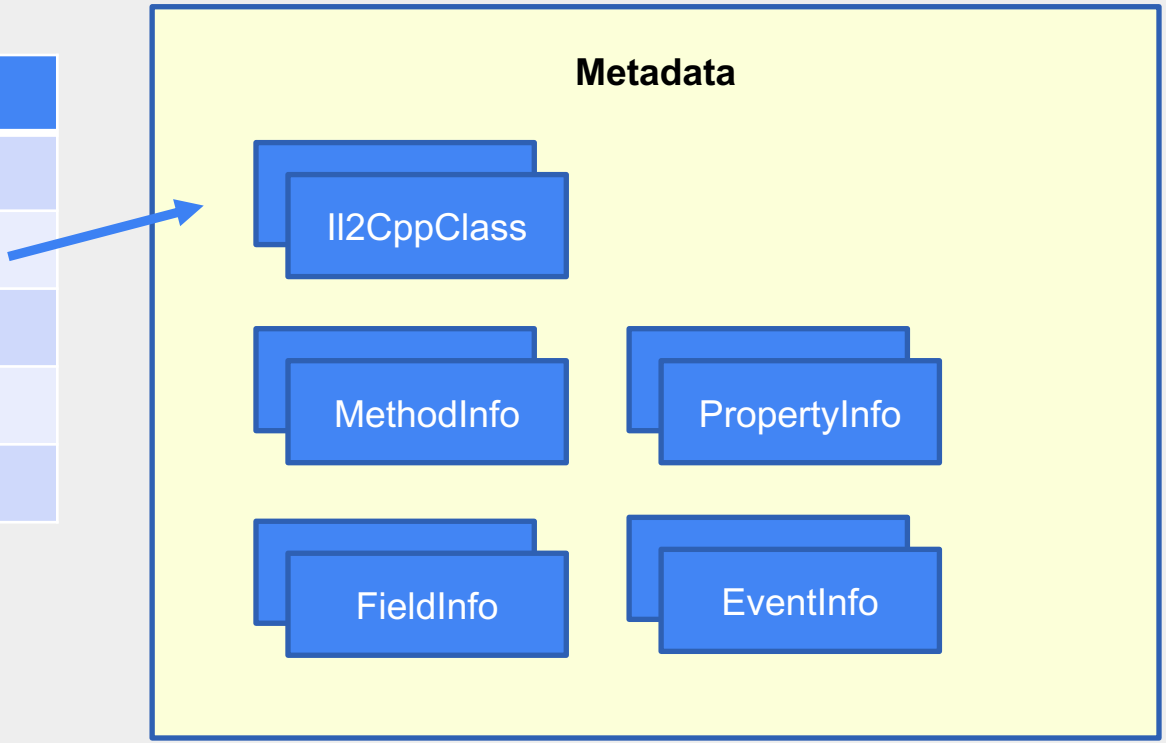
→ 优化IL2CPP运行时内存占用 (64MB -> 33MB)

→ 优化DynamicVBO pool的复用机制，减少粒子系统等显存占有 (59MB -> 38MB)

→ 后面提到的代码轻量化、资源裁剪也会帮助减小运行时内存

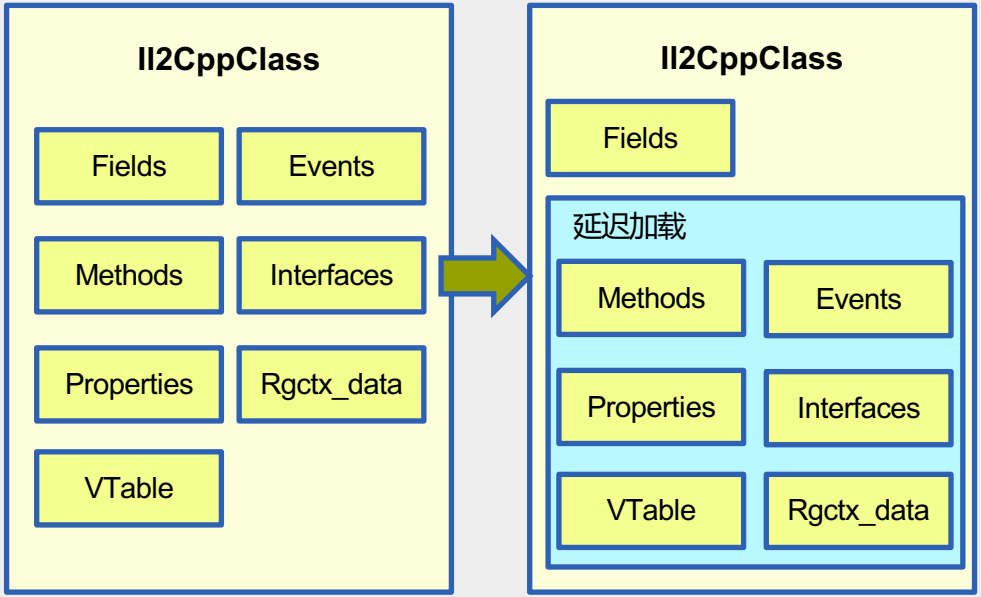
优化IL2CPP运行时内存占用

	案例1(MB)
总和	63.89
Metadata	37.03
global-metadata.dat	14.00
HashTable	7.96
Others	4.90



优化IL2CPP运行时内存占用

	优化前(MB)	优化后(MB)
Metadata总和	37.03	13.22
Class	9.63	3.43
Method	16.9	4.29
Property	0.85	0.001
GenericClass	0.48	0.10
GenericMethod	2.26	0.93



优化IL2CPP运行时内存占用

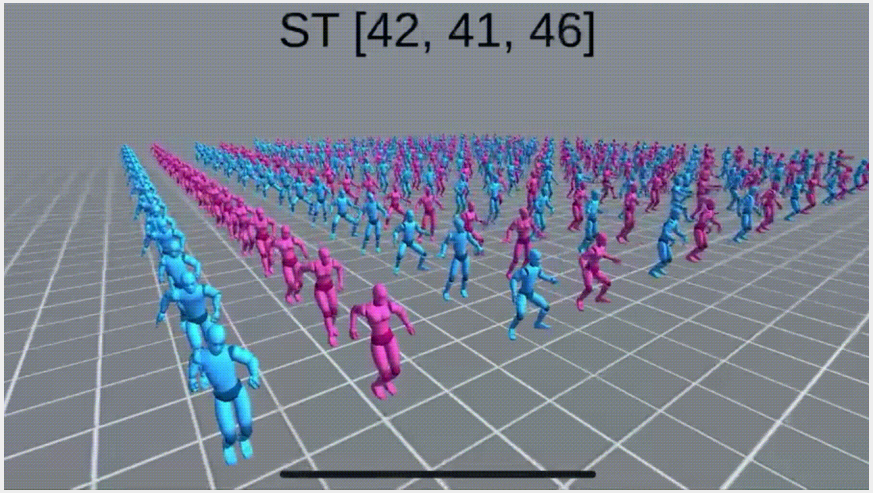
	案例1 优化前 (MB)	案例1 优化后 (MB)	案例2 优化前 (MB)	案例2 优化后 (MB)
总和	63.89	33.11	11.07	6.56
Metadata	37.03	13.22	5.02	1.99
global-metadata.dat	14.00	14.00	3.04	3.04
HashTable	7.96	1.90	2.18	0.72
Others	4.90	4.87	0.83	0.82

优化绘制效率

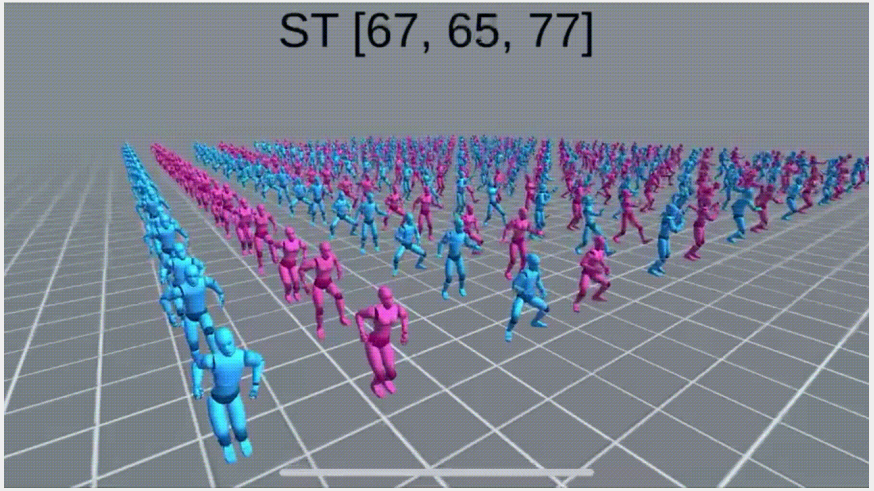
- → 在WebGL2上支持GPU skinning (15fps -> 24fps)
- → 优化Shader Compiler，将non-const global变量移到main函数中 (23fps -> 55fps)
- → 修改Immediate Const Buffer转换过程，声明成const并直接附初始值 (32fps -> 37fps)
- → 提供可配置的max visible lights值，32可降为16 (11fps -> 28fps)
- → iOS实现相关
  - 优化urpbatch在iOS上的行为，避免使用过多uniform变量，优化性能。
  - 在iOS14.x-15.4的WebGL上，同一个Canvas不共享IB和VB，改善UI渲染性能。

GPU Skinning

测试模型：4000顶点，53骨骼，每个顶点由4根骨骼控制
测试场景：320个模型，一共有8种动画
测试机型：iphone12+chrome



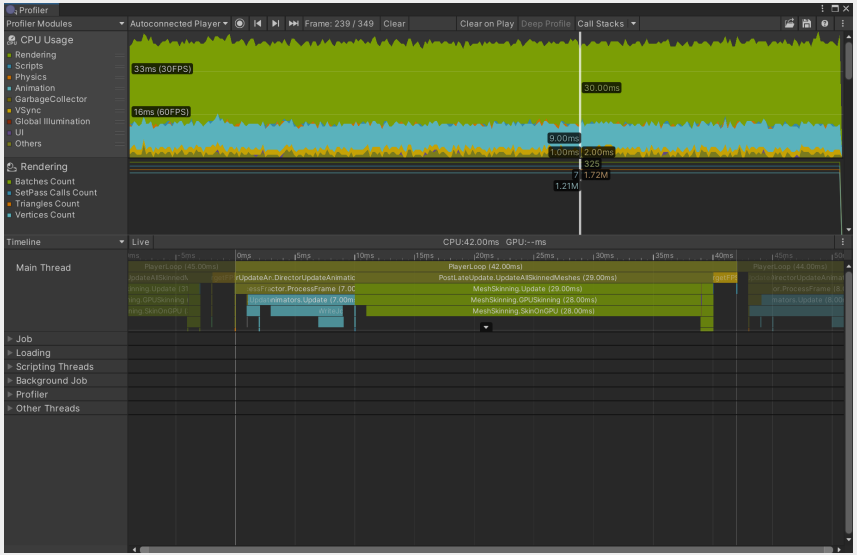
开启TF Skinning



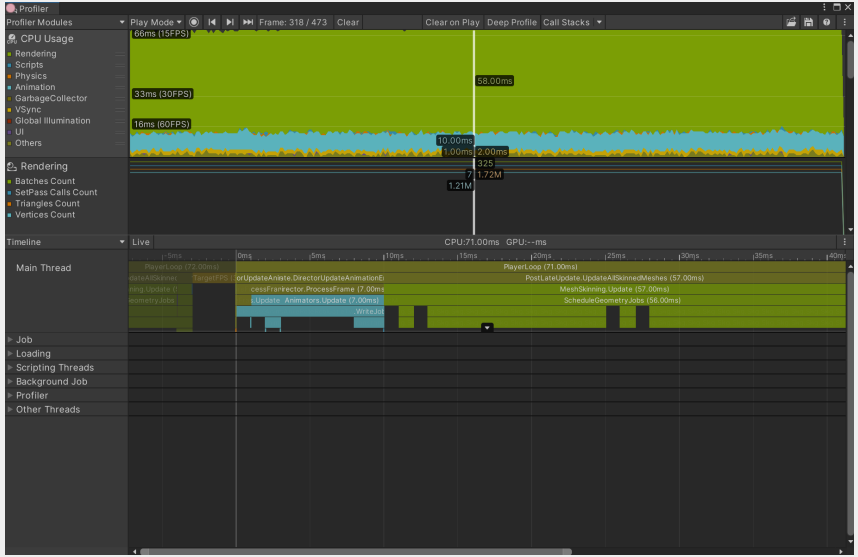
未开启TF Skinning

GPU Skinning

测试模型:4000顶点，53骨骼，每个顶点由4根骨骼控制
测试场景：320个模型，一共有8种动画
测试机型：iphone12+chrome



开启TF Skinning

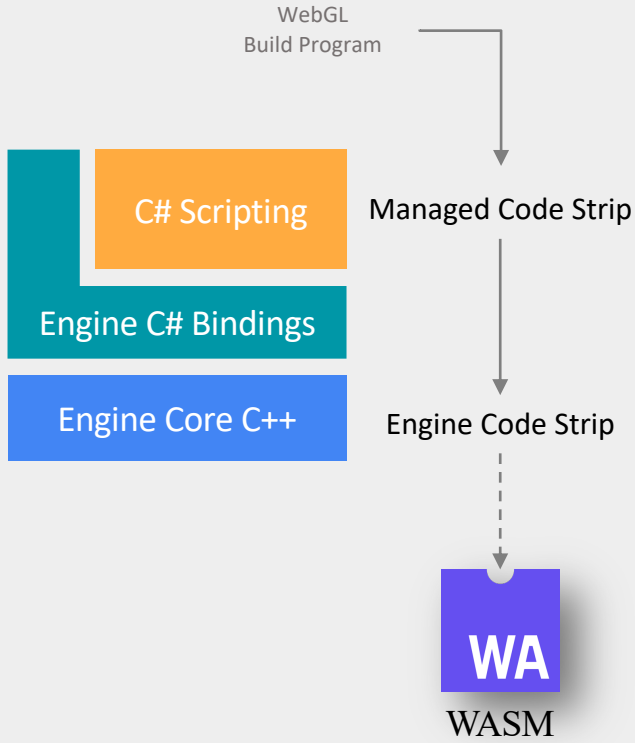


未开启TF Skinning

引擎代码轻量化

→ 当前方法

- Managed Code Strip
- Engine Code Strip
- 它们是通过静态分析依赖的方式做的strip
- 以函数为粒度

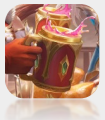


引擎代码轻量化

Wasm.br →

WASM →

Brotli



案例1

案例2

	案例1		案例2	
压缩大小 / br.Size	7.3 MB		7.0 MB	
解压后大小 / Size	35.1 MB		32.6 MB	
解析指令数 / Instructions	12.8 M		12.0 M	
▪ IL2CPP-gen	7.7 M	~60 %	7.5 M	~60 %
▪ Engine C++		~40 %		~40 %
Physx	1.0 M	8 %	0.99 M	8 %
ParticleSystem	370 K	3 %	370 K	3 %
Sqlite	130 K	1 %	133 K	1 %
Mecanim	68 K	.5%	67 K	.5%
…				
Non-Modular	1.3 M	10 %	1.5 M	13 %

引擎代码轻量化

→ 进一步轻量化

- 将C++模板参数改为函数参数，减少生成的代码量
- 通过宏隔离无法执行到的逻辑，避免引入不必要的依赖

加快启动速度

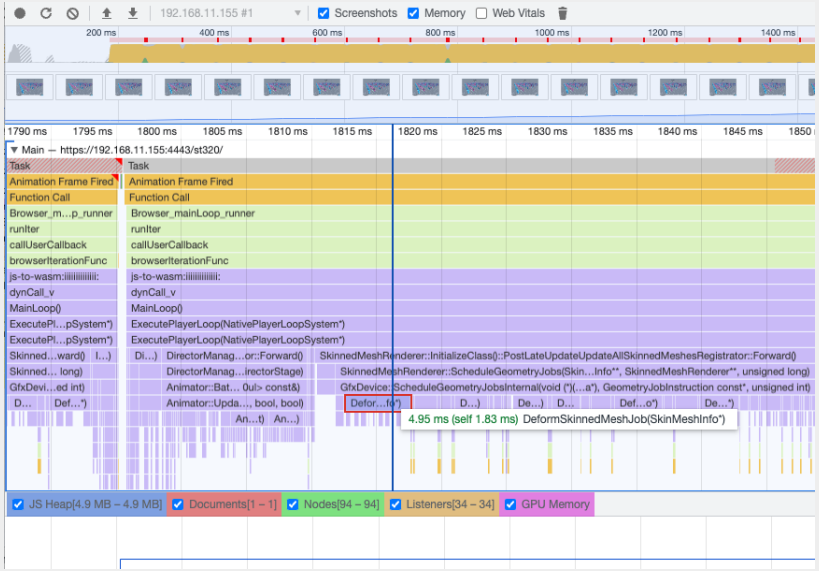
→ 与平台合作，使用宿主提供的中文字体

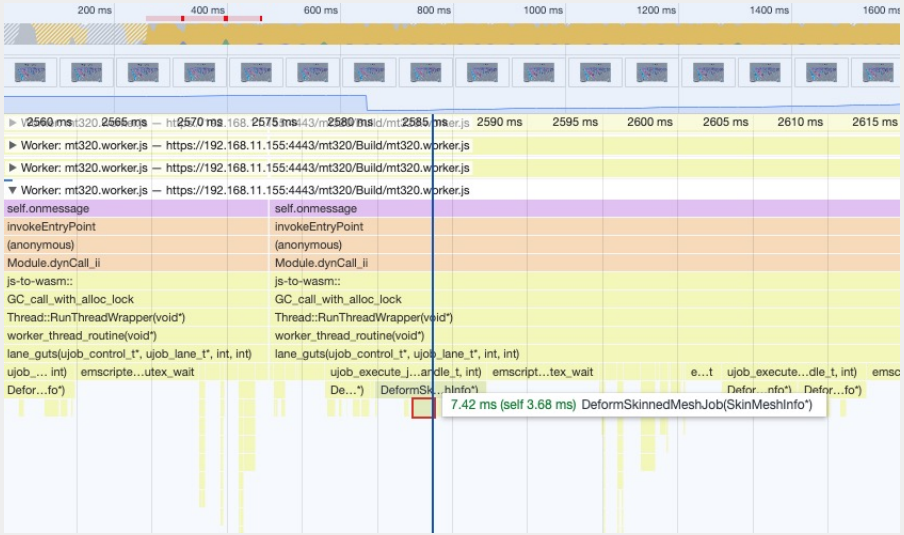
→ 动态裁减unity default resource

→ 尝试wasm snapshot方案

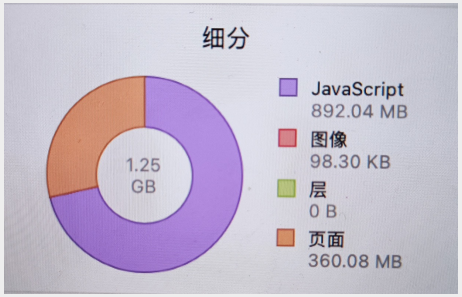
未来工作

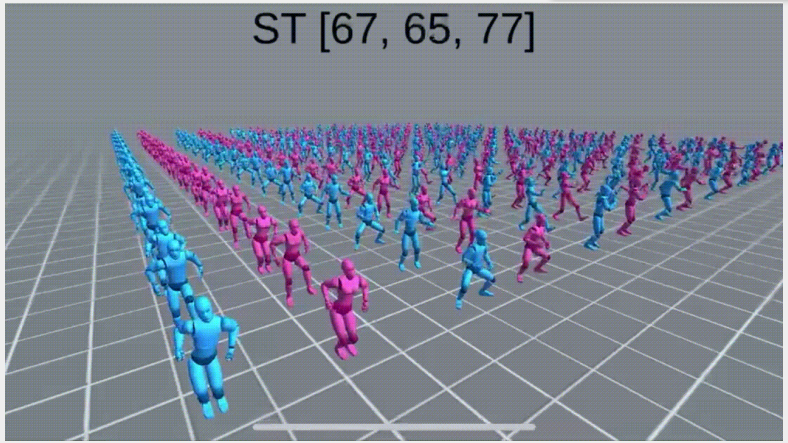
WebGL多线程

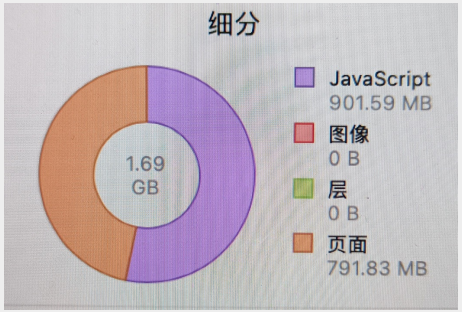


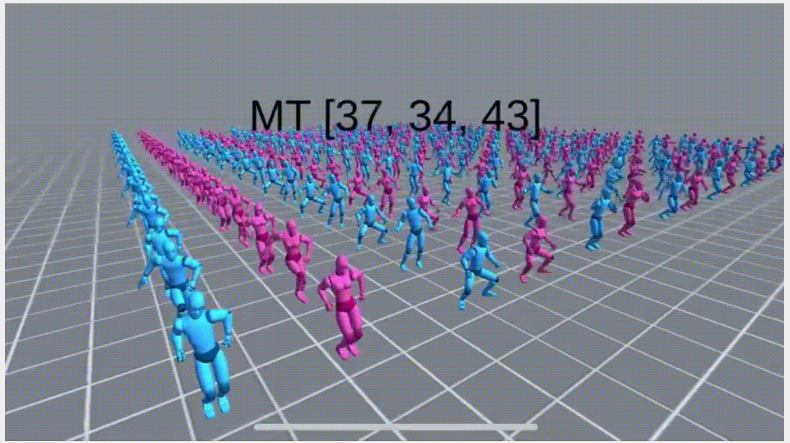


WebGL多线程









WebGL多线程

→ 目前还不够稳定，例如：切换场景会crash

→ 还不支持RenderThread（因为web worker无法访问DOM ）

→ 内存占用过大

→ 仅支持Native代码，不支持C#代码

WebGPU

→ 支持更多功能

- Compute Pipeline
  - GPU skinning
  - GPU particles
  - GPU culling
  - Post Processing
- Indirect draw
- Debug easily
  - Label
  - Debug group
- ...

References:
https://www.khronos.org/assets/uploads/developers/presentations/WebGPU_Best_Practices_Google.pdf

WebGPU

→ 更高效率

- CommandBuffer + Queue
- RenderBundle(预录制，可复用)
- RenderPipeline(支持异步创建)
- ShaderModule(允许多程序入口)
- 更好地支持多线程

References:
https://gpuweb.github.io/gpuweb/explainer
https://github.com/gpuweb/gpuweb/wiki/The-Multi-Explainer

Thank you!

Let’s keep in touch - liang.zhao@unity.cn