团结1.1版本小游戏新功能详解与性能优化

Unity

2024 3

Unity 赵亮

团结1.1 新增

.Net 8

Shader内存优化

Extreme Level

Remapper优化

内存分配器

IL2CPP 元数据

降低内存占用

GPU Skinning

wasm simd

降低CPU占用

Texture Mgr

Shader Warmup

优化AB打包耗时

优化加载/打包

Dev Host

Frame Debugger

C# Debug

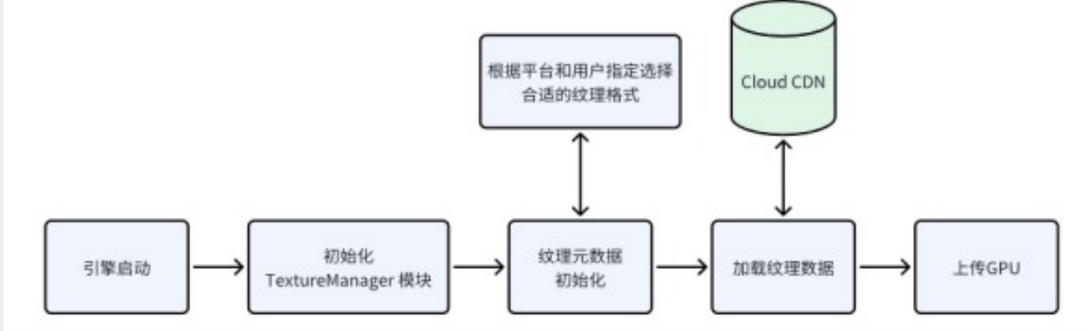
深度集成WXSDK

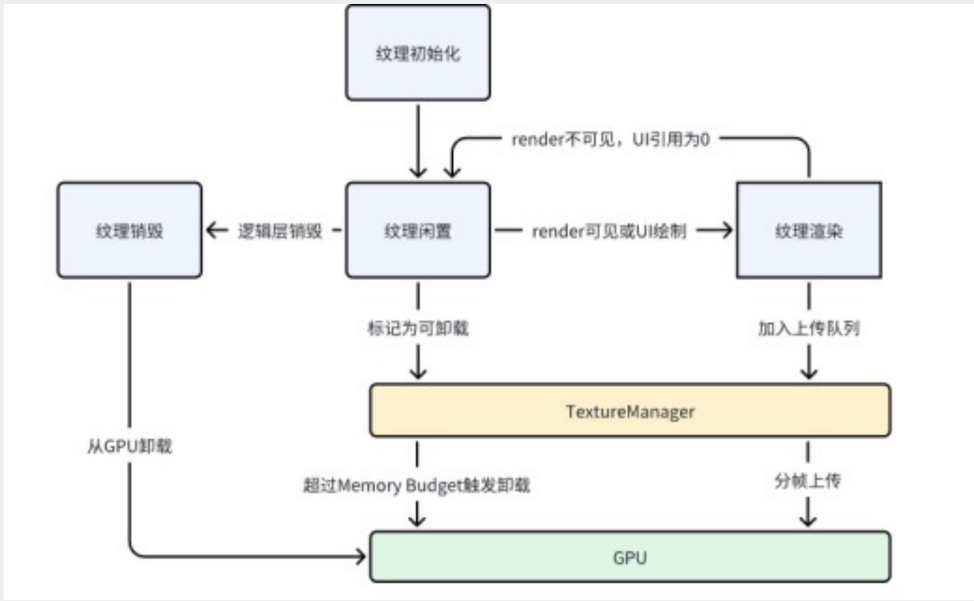
完善开发调试工具

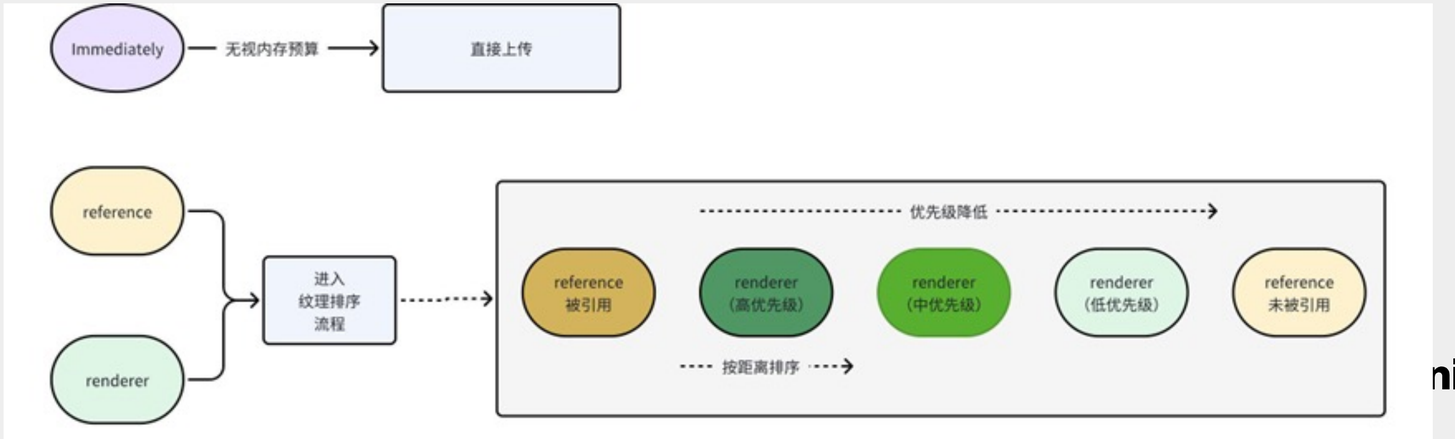
- .NET 8
- GPU Skinning
- Dev Host（Android）
- TextureManager
- Math库支持Wasm SIMD
- Shader Warmup
- Shader 内存优化
- 内存分配器优化
- Remapper运行时内存优化
- Managed Code Stripping - Extreme Level
- IL2CPP元数据精简
- 优化AssetBundle打包耗时
- 深度集成微信小游戏SDK

TextureManager

- 多纹理压缩格式支持
- 生命周期管理
- 显存预算控制
- 纹理重映射







Math库支持Wasm SIMD

- **Wasm SIMD**
  - Emscripten支持 WebAssembly SIMD功能，SSE2 和 ARM_NEON SIMD 指令集的指令可以通过编译转化成在 wasm 虚拟机下的指令进行模拟，从而获得比普通标量化运算更好的性能
  - 团结早些版本已经在微信小游戏平台的Skinning模块接入了WebAssembly SIMD
- **Math库**
  - 为引擎各个模块提供基础的数学运算
  - 其中的向量和矩阵等相关的运算天生适合使用SIMD进行优化
  - 使用WebAssembly SIMD intrinsics重写了Math库中的实现
  - 由此引擎整体都获得了来自SIMD的性能提升
- **微信小游戏分包即将支持simd**

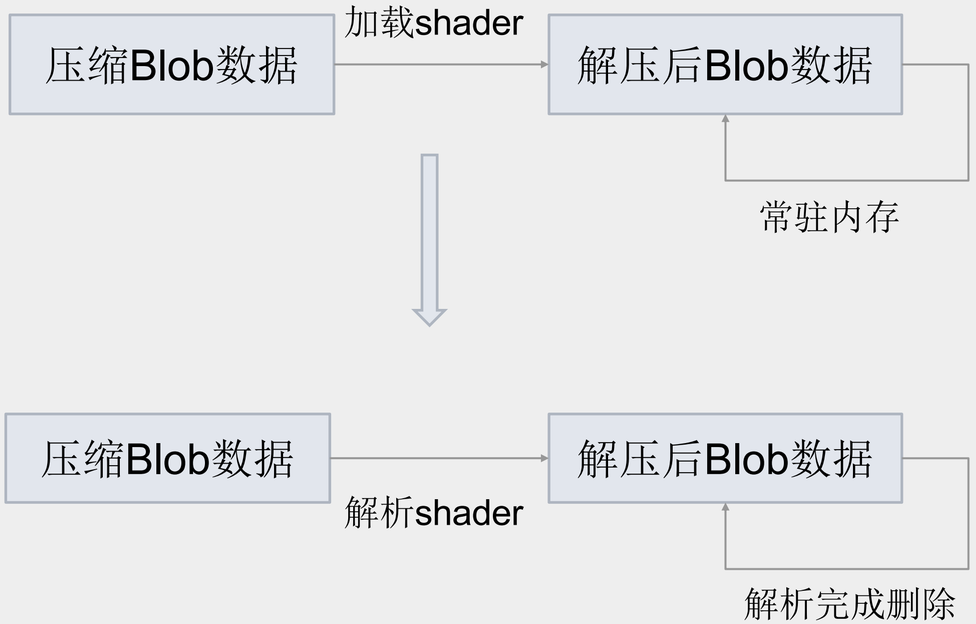
Shader Warmup

- **Shader编译比较耗时**
  - WebGL不支持Binary Shader, 单个Shader编译时间通常在几十毫秒
  - 现有WarmUp
    - ShaderVariantCollection.WarmUp()：可能会让主线程卡住长达数秒的时间
    - ShaderVariantCollection.WarmupShadersProgressively(1)：分散逐帧编译，但仍可能严重影响帧率
- **异步Shader Warmup**
  - WebGL的扩展 KHR_parallel_shader_compile 支持将Shader编译异步化
- **微信小游戏在安卓平台上暂时不支持**

Shader内存优化

优化Shader Blob数据内存

- 延迟解压Blob数据
- 卸载解压后的Blob数据



Shader	0	-34.6 MB	42.8 MB	8.2 MB	103	103
Universal Render Pipeline/Simple Lit	0	-8.9 MB	10.4 MB	1.6 MB	2	2
Hidden/Universal Render Pipeline/Terrain/Lit (A	0	-6.3 MB	7.2 MB	0.9 MB	1	1
Hidden/Universal Render Pipeline/Terrain/Lit (B	0	-4.6 MB	5.3 MB	0.7 MB	1	1
Universal Render Pipeline/Terrain/Lit	0	-3.1 MB	3.6 MB	0.5 MB	1	1
BoatAttack/Vegetation_Mobile	0	-2.3 MB	2.8 MB	434.0 KB	1	1
Hidden/Universal Render Pipeline/UberPost	0	-2.0 MB	2.3 MB	266.0 KB	1	1
Shader Graphs/gasPBR	0	-1.3 MB	1.6 MB	279.6 KB	1	1
Hidden/TerrainEngine/Details/UniversalPipeline,	0	-0.8 MB	0.9 MB	125.4 KB	1	1
Hidden/TerrainEngine/Details/UniversalPipeline,	0	-0.5 MB	0.6 MB	103.2 KB	1	1
BoatAttack/CliffShader_simple	0	-0.5 MB	0.7 MB	147.8 KB	1	1
BoatAttack/PBR_HUE_simple	0	-0.5 MB	0.6 MB	134.2 KB	1	1
Shader Graphs/Houses_Jetty_simple	0	-505.4 KB	0.6 MB	138.1 KB	1	1
BoatAttack/Water	0	-414.3 KB	502.7 KB	88.5 KB	1	1
BoatAttack/RaceBoats	0	-359.3 KB	479.5 KB	120.3 KB	1	1
Hidden/Universal Render Pipeline/FinalPost	0	-287.0 KB	364.0 KB	77.0 KB	1	1
Universal Render Pipeline/Particles/Lit	0	-272.1 KB	395.5 KB	123.4 KB	1	1
Hidden/TerrainEngine/Details/UniversalPipeline,	0	-266.4 KB	342.1 KB	75.8 KB	1	1
TextMeshPro/Mobile/Distance Field	0	-240.8 KB	386.5 KB	145.7 KB	4	4
Universal Render Pipeline/Particles/Simple Lit	0	-224.9 KB	328.8 KB	103.9 KB	1	1
TextMeshPro/Distance Field Overlay	0	-216.2 KB	361.7 KB	145.5 KB	3	3

BoatAttack Shader内存优化前后对比

内存分配器优化

引擎分配器内存Overhead

- 出于Profile、平台兼容等原因，会记录每次分配的信息，并且由此会在内存分配器层面多引发一次内存对齐，Release版本也是如此
- 微信小游戏内存比较紧张，去除了这部分开销

内存Alignment

- 引擎默认内存Aligment是16，在微信小游戏平台上我们优化为8字节

实测案例

- 某款中重度游戏，上述优化减少内存占用约 **10~12MB**

Remapper运行时内存优化

- **Remapper**
  - **序列化位置**和**InstanceID**之间双向映射关系的数据结构。
- 优化前
  - InstanceID每次加二，不复用
  - 哈希Map，稀疏，内存翻倍增长
- 优化后
  - InstanceID每次加一，复用
  - 数组，紧凑，内存线性增长

Remapper目前仅在微信小游戏平台开启，后续会逐步向其他平台开放

Managed code stripping – Extreme level

Managed code stripping

- 通过UnityLinker
  - 从DLL中剔除没有使用到的代码
  - 原有最高级别 – High
- 剔除策略仍然偏保守
  - 保留了 MonoBehaviour 和 ScriptableObject 的全部成员
- 可能剔除必要代码
  - 需要用户自行发现，添加到link.xml
- 新增Dryrun 模式
- .Net 8 backend没有必要使用

IL2CPP元数据精简

IL2CPP元数据

- IL2CPP运行时依赖元数据来获知C#类型、方法等信息
- 默认的元数据结构可以支持超过21亿个类型或方法
- 但在小游戏来说，方法的数量通常在万这个级别

元数据精简

- 打包小游戏时，根据方法的数量自动选择满足当前要求的、精简的元数据结构
- global-metadata.dat体积缩减约 **15%**

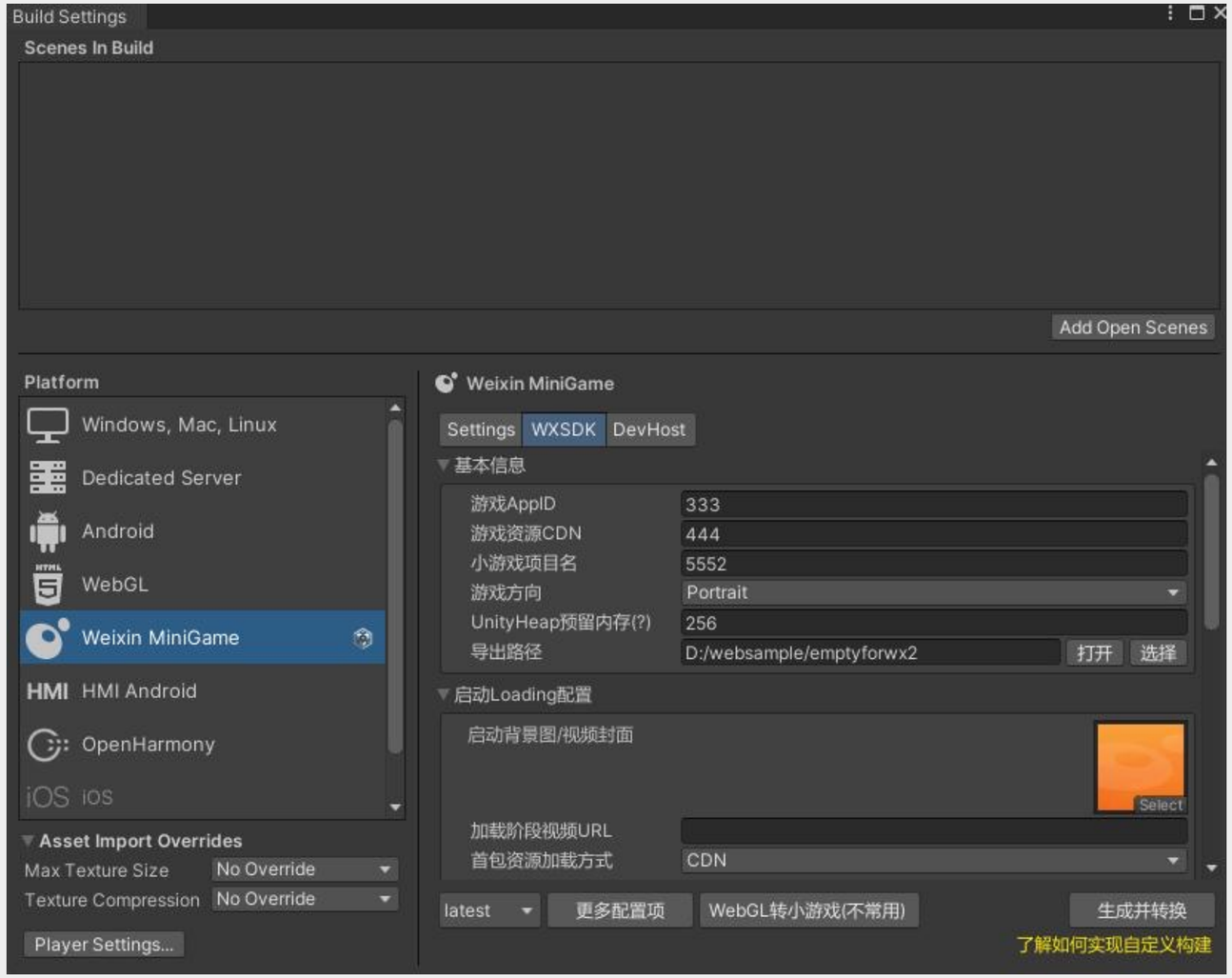
AssetBundle打包优化

优化BuildPipeline.BuildAssetBundles()接口

- 改进了AB打包流程中图集相关的逻辑
- 该优化目前仅在微信小游戏平台开启，后续会逐步向其他平台开放
- 实测案例
  - 2.5万个AB，打包AB时间从优化前的**160分钟减少为70分钟**

深度集成微信小游戏SDK

- 切换至Weixin MiniGame平台，引擎自动安装WXSDK package；
- 原微信小游戏转换工具面板内嵌至BuildSettings界面中；
- 后续将支持引擎内选择微信SDK版本。



详解

- .NET 8
- GPU Skinning
- Dev Host（Android）

.NET 8

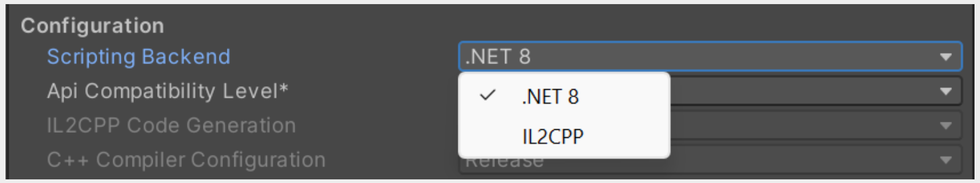
IL2CPP方案痛点

- Wasm
  - IL转换为Cpp，编译链接进一个All-in-one的Wasm
  - 浏览器加载执行Wasm时，代码编译、运行时指令优化、JIT优化等等，会消耗Wasm体积数倍的内存
  - 例如，Wasm体积**50MB**，仅Wasm相关的内存开销就可能多达**500MB**

.NET 8

.NET 8

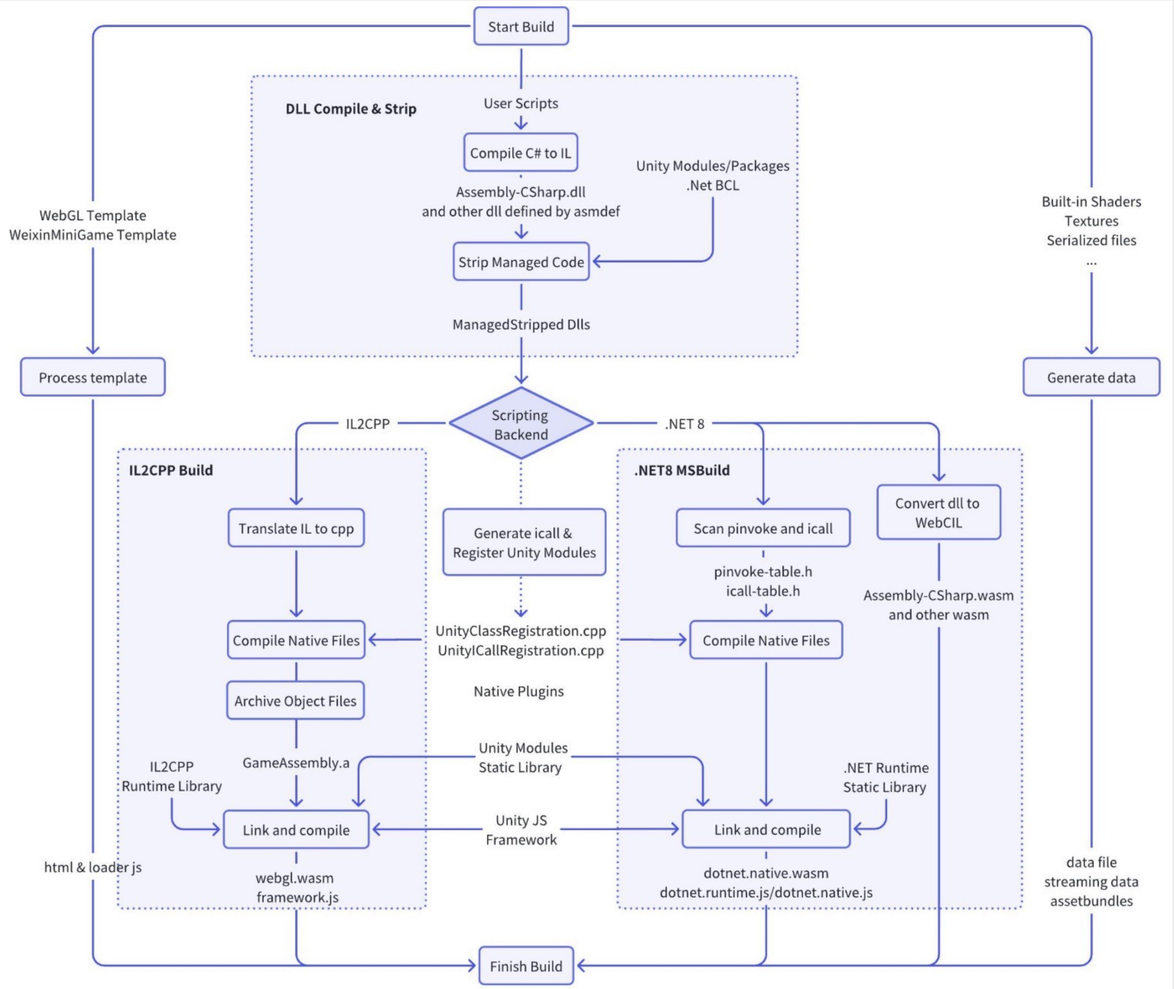
- 2023微软最新发布
  - 稳定支持WebAssembly
- 解释执行IL
  - 将Dll部分从Wasm中剥离
  - 减小WASM体积，显著降低运行内存
- .NET生态
  - AOT，JITerpreter，SGen等新技术
  - 性能提升



.NET 8

构建环节

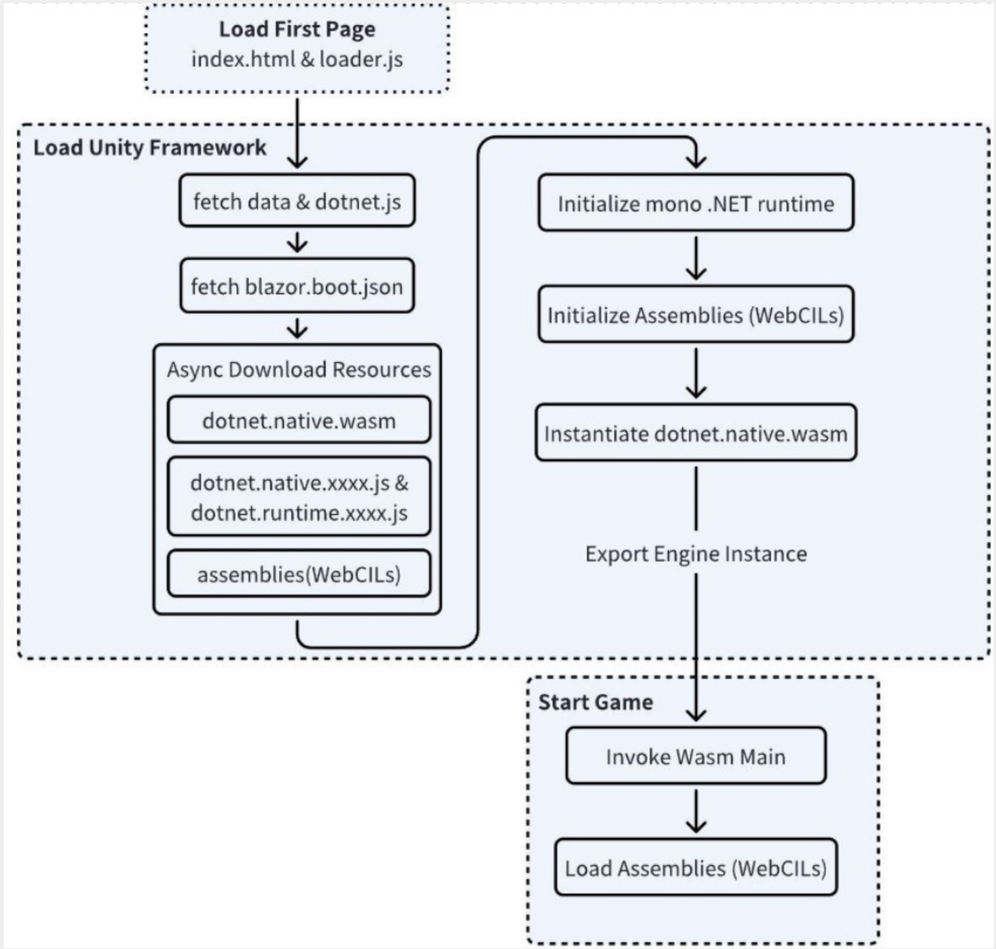
- DLL Compile & Strip
  - 与IL2CPP一致
- Convert DLL
  - In：DLL
  - Out：WebCIL格式的Wasm文件
- Native code Compile & Link
  - In：Native代码 + 引擎 + .NET Runtime
  - Out：dotnet.native.wasm + js胶水代码



.NET 8

启动流程

- 加载引导文件：
  - .Net的data和js + blazor.boot.json
- 下载代码资源：
  - dotnet.native.wasm + WebCIL文件（从CDN）
- 初始化
  - 初始化.NET Runtime
  - 以Buffer形式加载WebCIL文件
  - 加载dotnet.native.wasm
- 进入游戏
  - CallMain
  - 按需加载WebCIL，执行其中方法



.NET 8

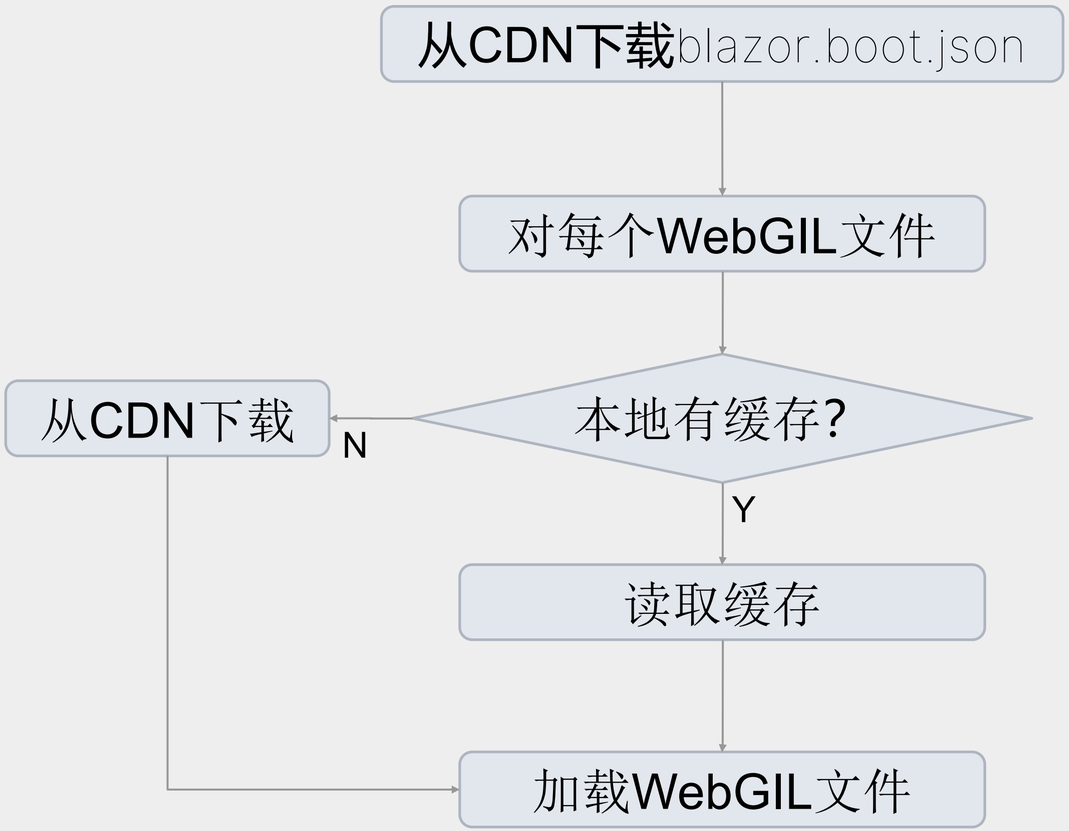
SGen GC

- 专为Mono设计
  - 精确扫描Mono Stack
  - 运行时搬移整理内存碎片
  - 分代回收，回收行为可以分散在每一帧，减少卡顿
- 有一些内存Overhead
  - 预分配40~50MB内存用于维护GC相关的数据结构，同时可能按需上浮
  - 内存比较紧张的项目仍然可以使用Boehm GC

.NET 8

Dotnet Wasm方案天然支持热更新

- blazor.boot.json
  - WebCIL文件列表：文件名 & Hash
- WebCIL本地缓存
  - 基于微信本地文件API
  - 路径中包含Hash
- 天然支持热更新
  - 都是从CDN下载
  - WebCIL本地缓存可以通过Hash区分



.NET 8

热更新方案POC

- 下载blazor.boot.json时禁用缓存
  - Dotnet Wasm方案本身不对blazor.boot.json做缓存
  - blazor.boot.json下载URL是固定的，需要禁用WebRequest缓存
  - 修改微信SDK，设置header
- 游戏代码准备
  - 建议将需要热更的部分放入单独的DLL
  - 热更部分用到的外部接口需要在主版本中就Preserve，以防被Strip
- 修改热更部分的代码，并重新Build
- 修改主版本的blazor.boot.json中热更文件的信息
  - 更新Hash
- 部署热更的WebCIL文件到CDN
  - 新版本文件的修改时间要晚于旧版本
  - 否则下载请求会返回304（Not Modified）

```
// minigame-tuanjie-transform-sdk-main\Editor\WXConvertCore.cs

...
case 'manifest':
    return new Promise((resolve, reject) => {
        wx.request({
            url: remoteUrl,
            header: {
              'Cache-Control' : 'no-cache'
            },
            success: (res) => {
                resolve({
```

```
// blazor.boot.json
  ...
"resources": {
   ...
  "assembly": {
    "Assembly-CSharp.wasm": "sha256-xxxx",
   ...
   }
}
```

.NET 8

- **JITerpreter （微信小游戏暂不支持）**
  - 类似JIT（Just-in-Time）编译优化的特性
    - 将部分热点代码转换为细密的Wasm代码，从而提高运行效率
      - 一段热点代码可能生成多份Wasm文件，每份Wasm不超过4KB
  - 性能提升
    - 对解释执行的程序可以带来相当可观的性能提升
    - 也可以用来优化Wasm和JS代码间的相互调用
  - .Net8 默认开启

.NET 8

优势总结

- 更小的运行内存
- 更平滑的帧率波动
- 更快的出包时间
- 热更新的天然支持

.NET 8

FAQ

- Nightly版本微信开发者工具
- 开启wasm experimental feature
- 开启wasm exception handling
- .Net sdk路径过长问题
- iOS需要开启高性能模式
- Managed Stripping Level推荐使用
  - Low / Minimal
- System.Cryptography 不完全支持（例如MD5）
- BinaryFormatter未实现
- 暂不支持VS/Rider调试C#

FAQ

.NET 8

GPU Skinning

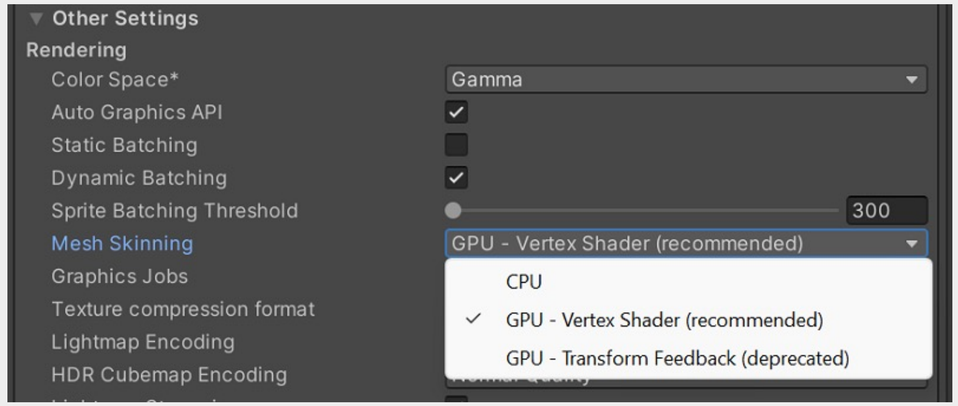
几种原有的Skinning方案

- CPU
  - 最慢
- CPU - SIMD
  - 微信小游戏分包尚不支持
- GPU - Compute Shader
  - WebGL不支持
- GPU - Tranform Feedback
  - 增加一个pass，利用Vertex shader计算，输出到GL_TRANSFORM_FEEDBACK_BUFFER
  - 对某些场景效果不佳甚至是负优化（角色多，每个角色顶点数少）

GPU Skinning

Vertex Shader Skinning

- 实现原理
  - Skinning在Vertex shader里计算
  - 不增加绘制pass
  - 使用Uniform更新骨骼矩阵
  - 未来考虑在WebGL2上提供cbuffer方案
    - 好处是支持更大数量的骨骼
    - 缺点是更新速度不及uniform快



GPU Skinning

Vertex Shader Skinning

- 用户自定义shader
  - 只需添加如下一行

```
CGPROGRAM
    ...
    #pragma vertex vert
    #pragma fragment frag
    #include "UnityCG.cginc"
    #pragma multi_compile _ ENABLE_VS_SKINNING
    ...
    struct appdata_t {
        float4 vertex : POSITION;
        float2 texcoord : TEXCOORD0;
        ...
    };

    struct v2f {
        float4 pos : SV_POSITION;
        float2 uv : TEXCOORD0;
        ...
    };

    v2f vert (appdata_t v)
    {
        v2f o;
        ...
        o.vertex = UnityObjectToClipPos(v.vertex);
        ...
    }
ENDCG
```

GPU Skinning

```
1  static const int max_bone_count = 64;
2  uniform int BonesPerVertex;
3  uniform float4 Bones [max_bone_count * 3];
4
5  void vs_skinning (inout float3 vertex, in float4 boneWeights, in float4 boneIndices)
6  {
7      ...
8  }
9
10 v2f vert (appdata_t v, float4 boneIndices : BLENDINDICES, float4 boneWeights : BLENDWEIGHTS)
11 {
12     vs_skinning(v.vertex, boneWeights, boneIndices);
13
14     v2f o;
15     o = (v2f) 0;
16     o.vertex = UnityObjectToClipPos (v.vertex);
17     ...
18 }
```

GPU Skinning

- **Vertex Shader Skinning**
  - 部分内置shader已添加支持

Built-in RP:

- Mobile/Bumped Diffuse
- Mobile/Bumped Specular (1 Directional Realtime Light)
- Mobile/Bumped Specular
- Mobile/Diffuse
- Mobile/VertexLit (Only Directional Lights)
- Unlit/Transparent
- Unlit/Transparent Cutout
- Unlit/Color
- Unlit/Texture

URP:

- Universal Render Pipeline/Simple Lit

GPU Skinning

Vertex Shader Skinning使用时注意事项

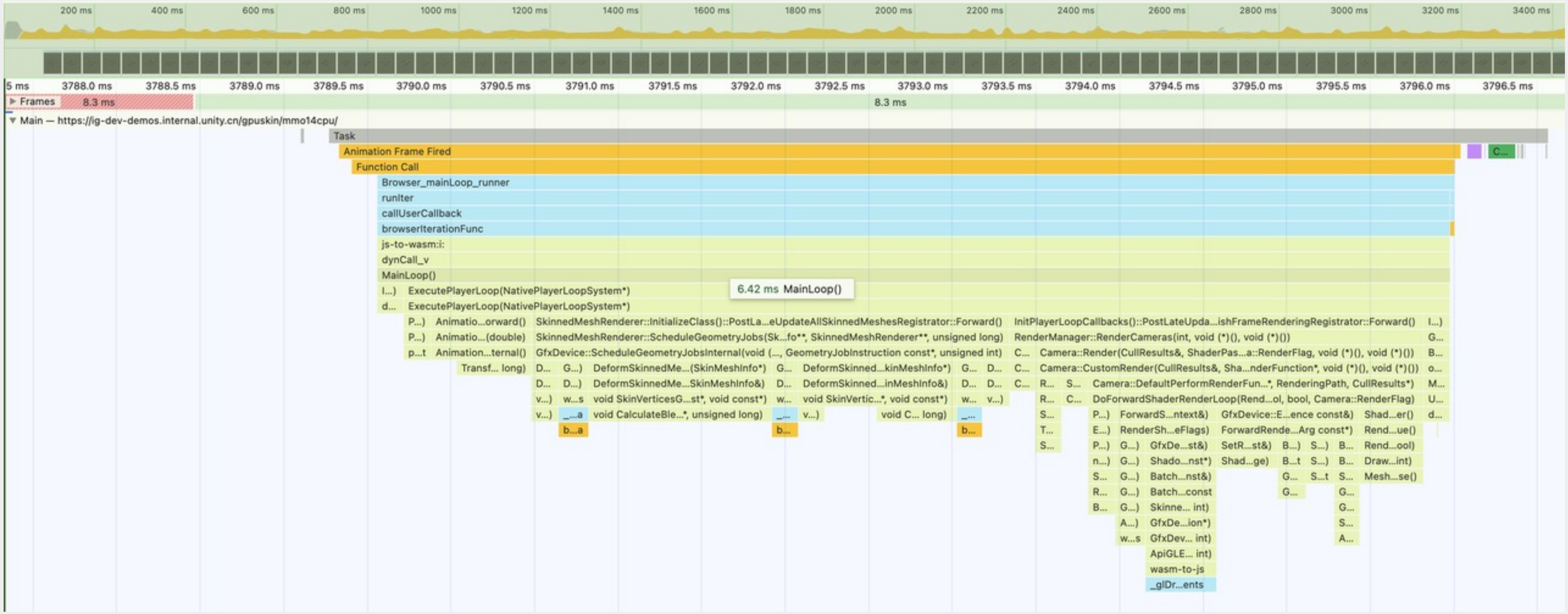
- 目前的数量限制为64根，超出时自动fallback到CPU skinning
- 当PlayerSetting中选择了VS Skinning，没有添加keyword的shader会Fallback到CPU Skinning
- 暂不支持BlendShape和MotionVector
- Renderer使用多个Material时，每个材质都需要ENABLE_VS_SKINNING
- 在WebGL1上有个bug，团结1.1.4版本将修复

GPU Skinning

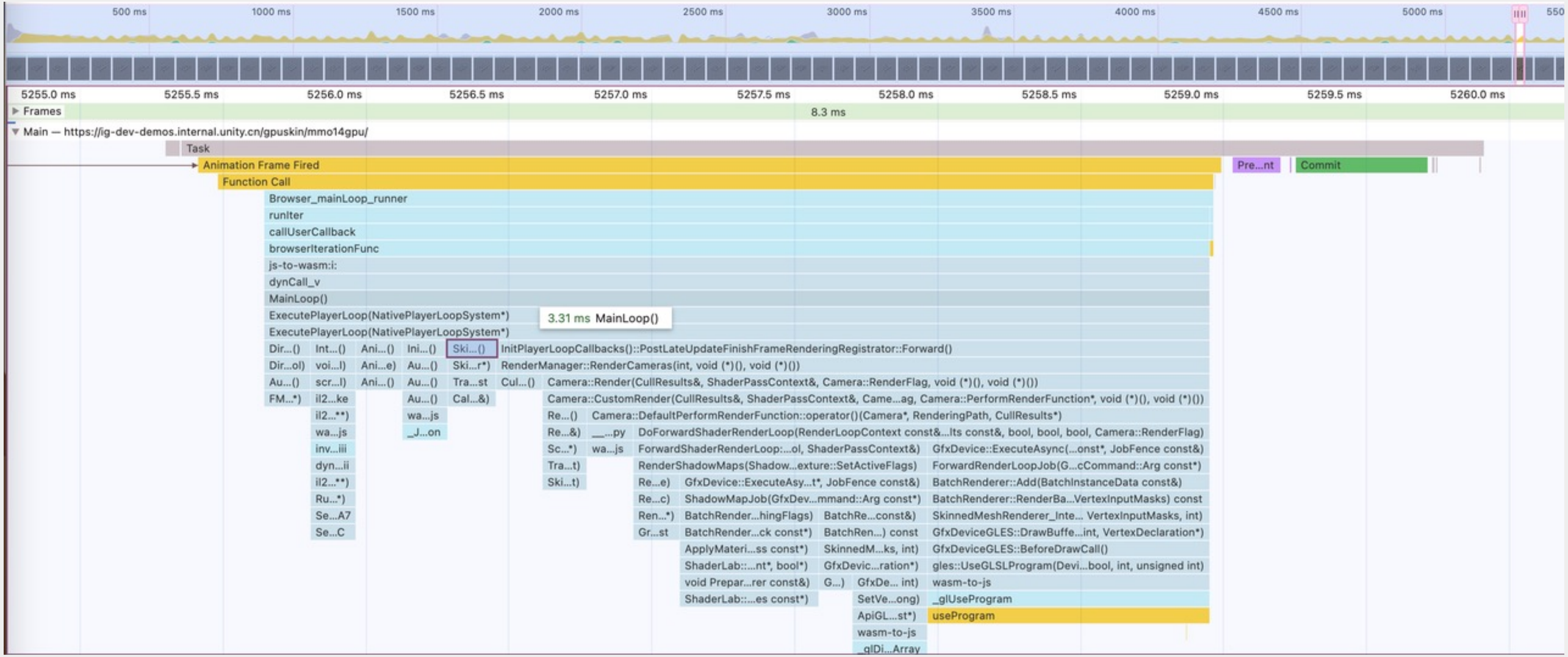
Vertex Shader Skinning，动画测试工程

	MainLoop CPU耗时	Skinning CPU耗时	RenderCam耗时	CPU占用率
CPU Skinning	6.42ms	2.86ms	2.48ms	37～40%
GPU Skinning	3.32ms	0.17ms	2.50ms	53～65%

GPU Skinning



GPU Skinning

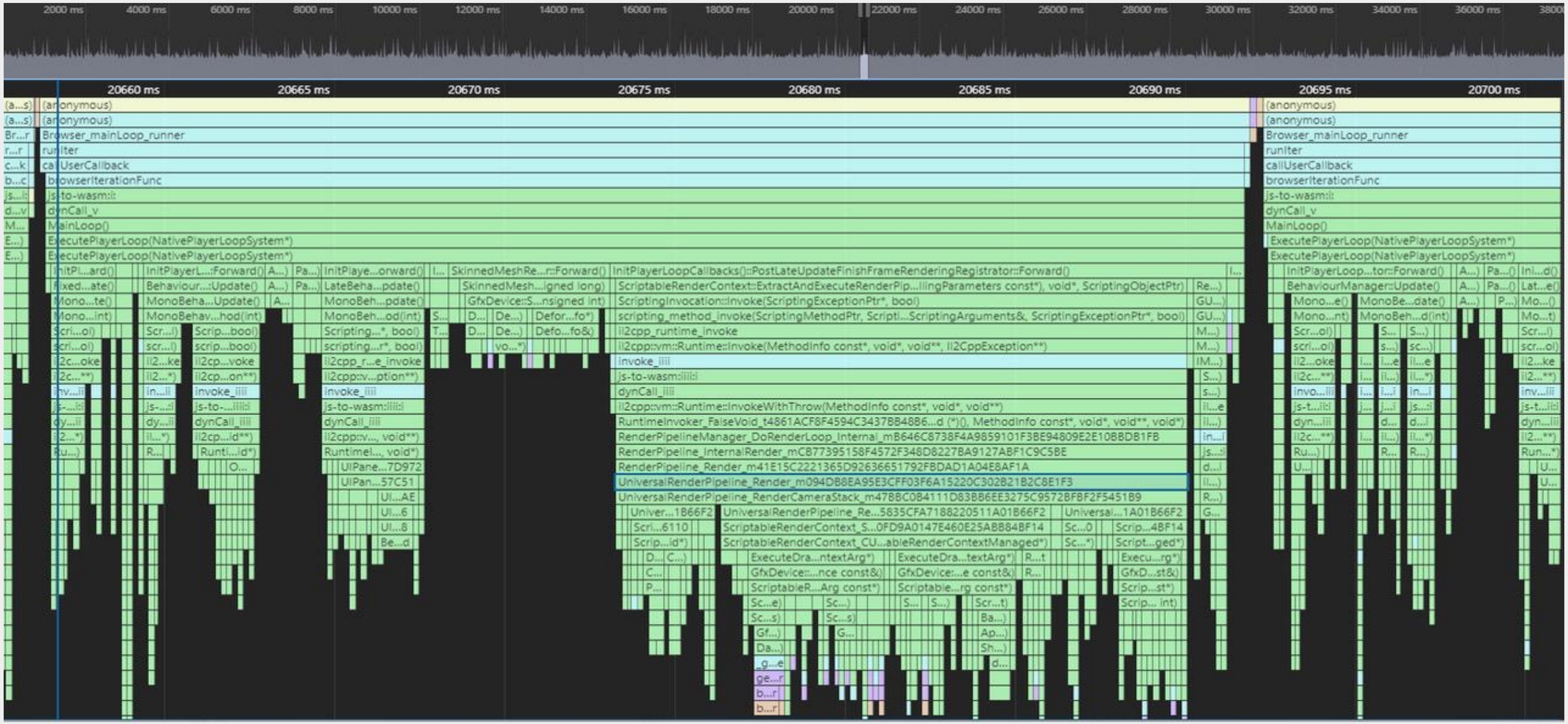


GPU Skinning

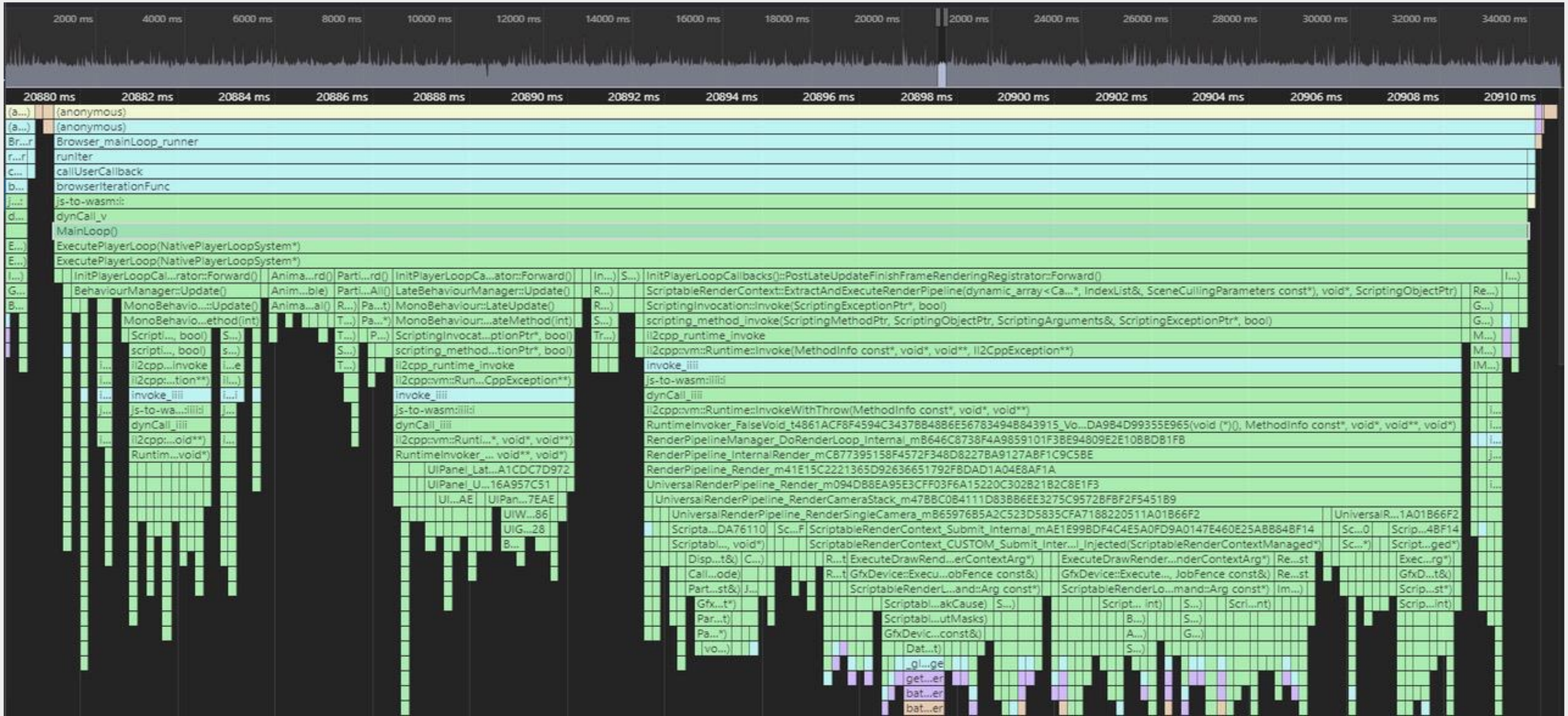
Vertex Shader Skinning

	MainLoop CPU耗时	Skinning CPU耗时	RenderCam耗时	CPU占用率
CPU Skinning	37.3ms	6.052ms	17.667ms	37～40%
GPU Skinning	34.548ms	0.438ms	19.74ms	53～65%

GPU Skinning

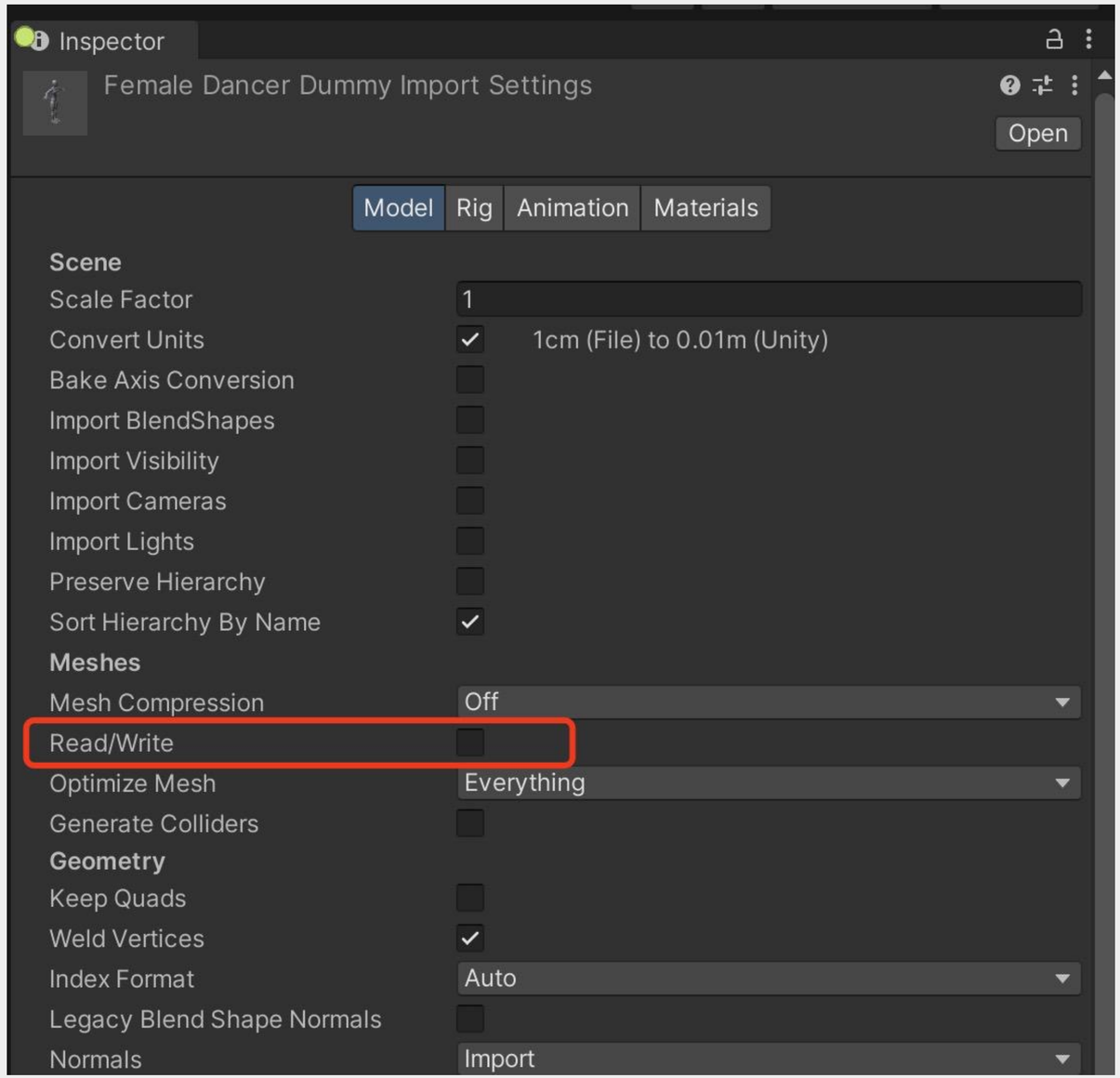


GPU Skinning



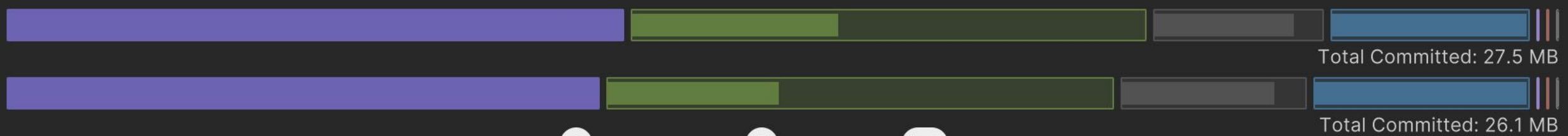
GPU Skinning

- **Vertex Shader Skinning**
  - CPU侧顶点数据可以丢弃
    - 不用开启Read/Write
  - 共用GfxBuffer
    - 不用每个SMR都分配一个GfxBuffer
      （用于保存变换后的顶点）



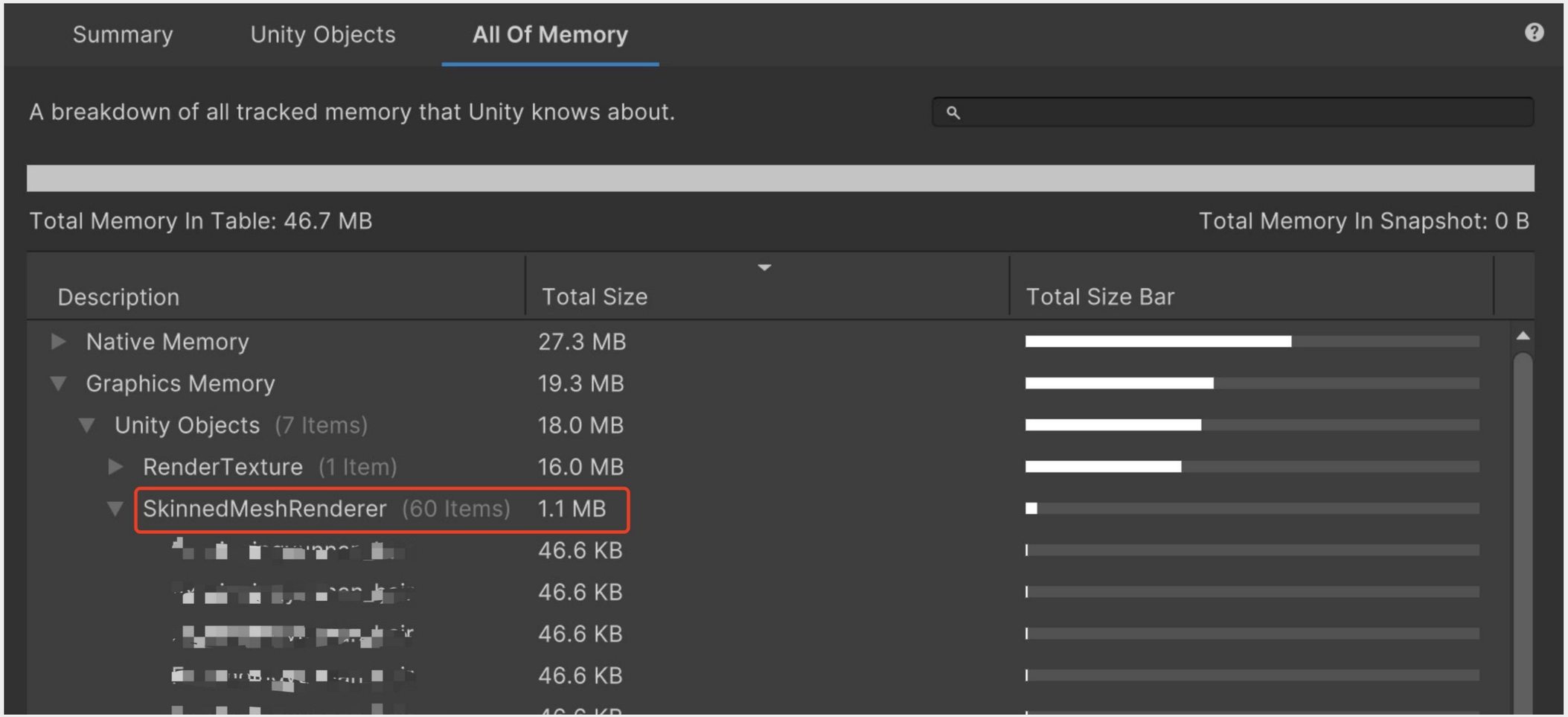
GPU Skinning

Total Committed Memory



	A	B	A-B
Graphics	19.3 MB	17.1 MB	2.2 MB
Native *(In use / Reserved)*	6.5 MB / 16.1 MB	5.0 MB / 14.6 MB	1.5 MB / 1.5 MB
Profiler *(In use / Reserved)*	4.4 MB / 5.3 MB	4.4 MB / 5.3 MB	4.0 KB / 4.0 KB
Managed *(In use / Reserved)*	6.1 MB / 6.2 MB	6.1 MB / 6.2 MB	0 B
Audio	510 B	510 B	0 B
Executables & Mapped	0 B	0 B	0 B
Unknown	0 B	0 B	0 B

GPU Skinning



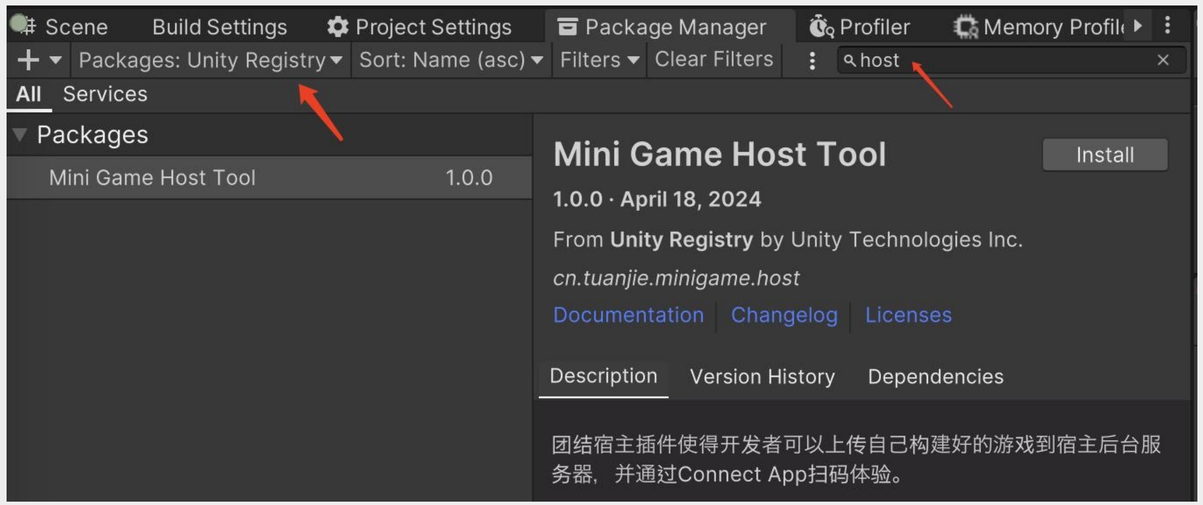
Dev Host（Android）

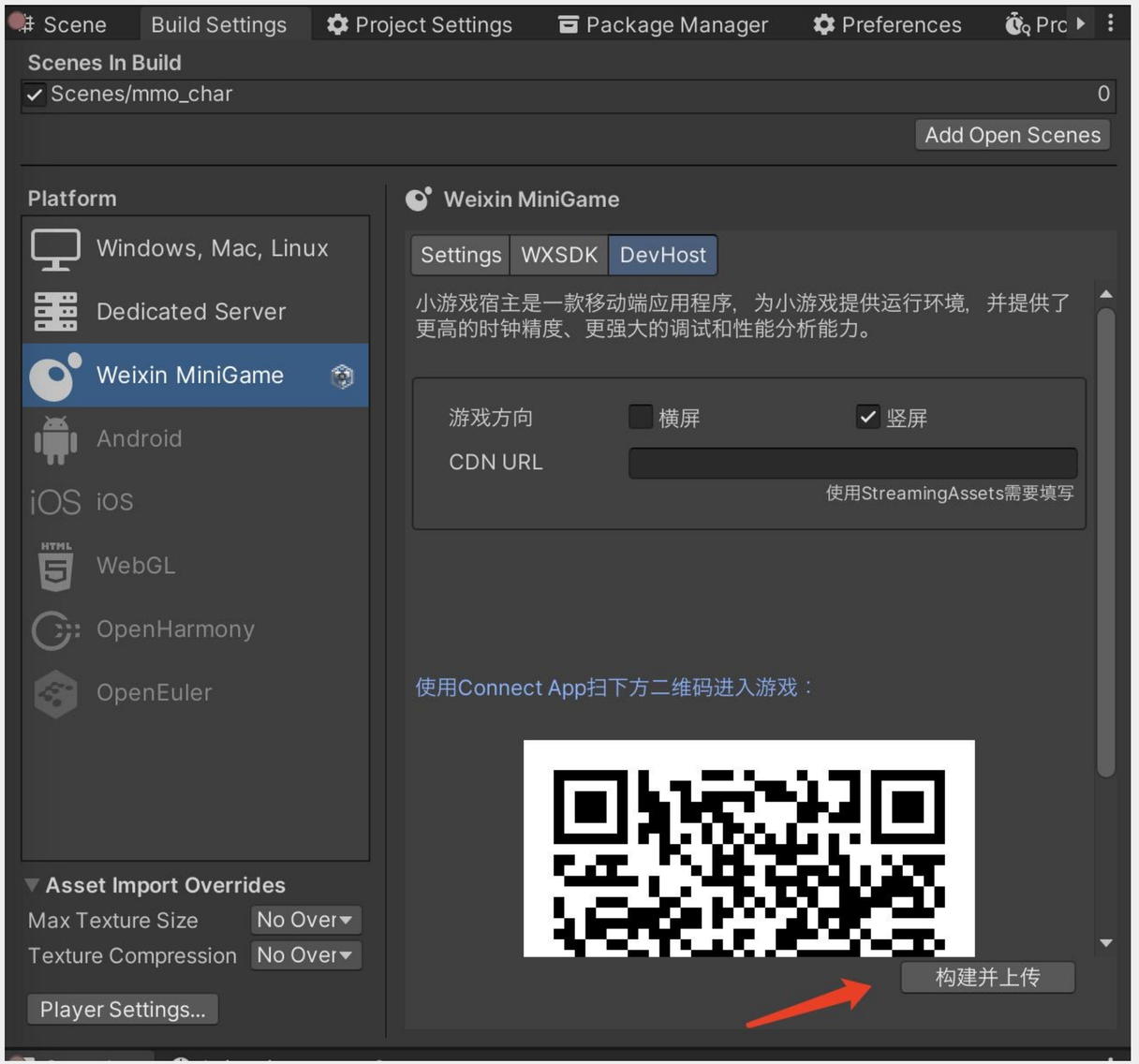
- 基于v8开发；
- 提供CPU使用率、帧率、内存、VConsole、启动时间等信息
- 引擎内一键打包上传，扫码即可运行
- 高时钟精度Timeline Profile
- 支持Frame Debugger
- 提供C# Debugging的能力

Connect App下载

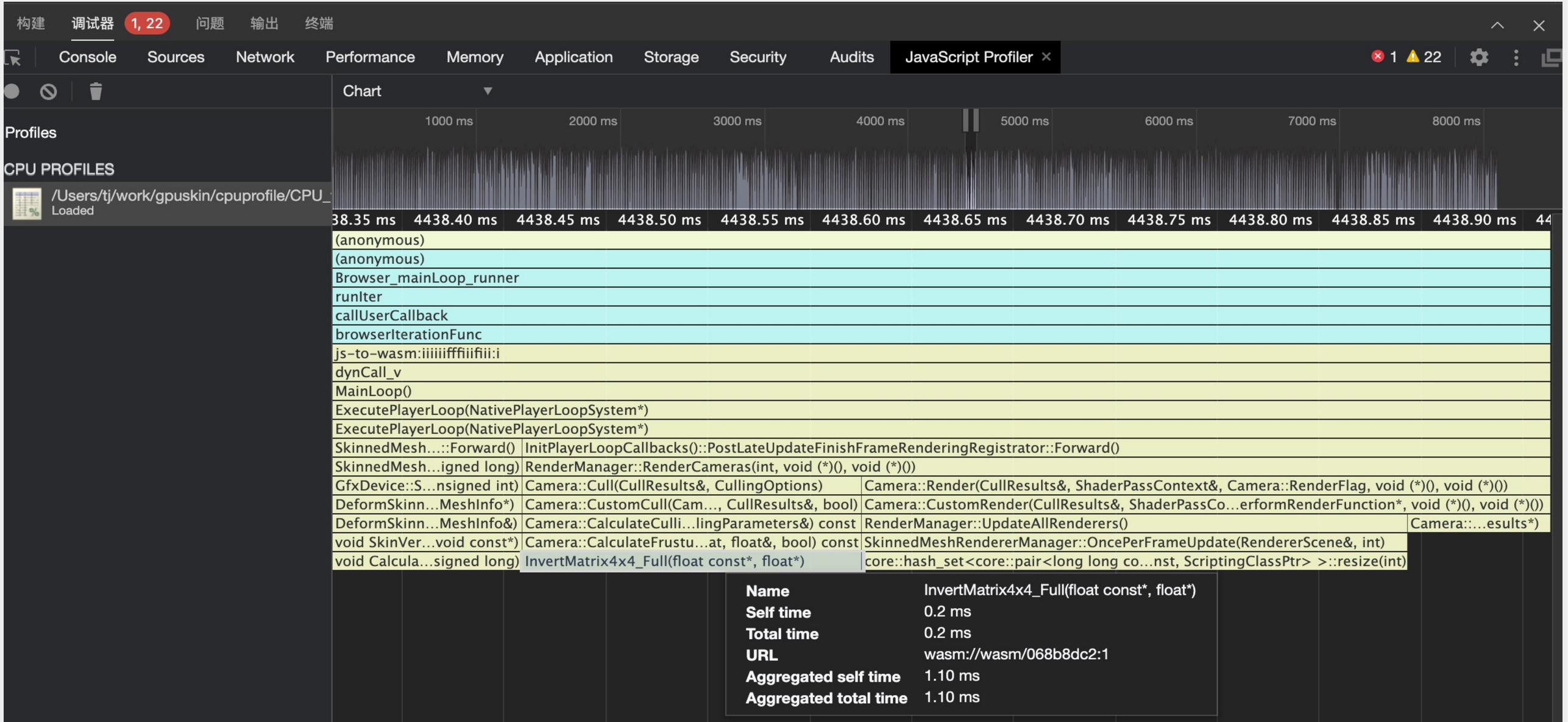
Dev Host （Android）

- 从Package Manager 安装cn.tuanjie.minigame.host package
- DevHost -> 构建并上传
- 使用Connect App扫码运行

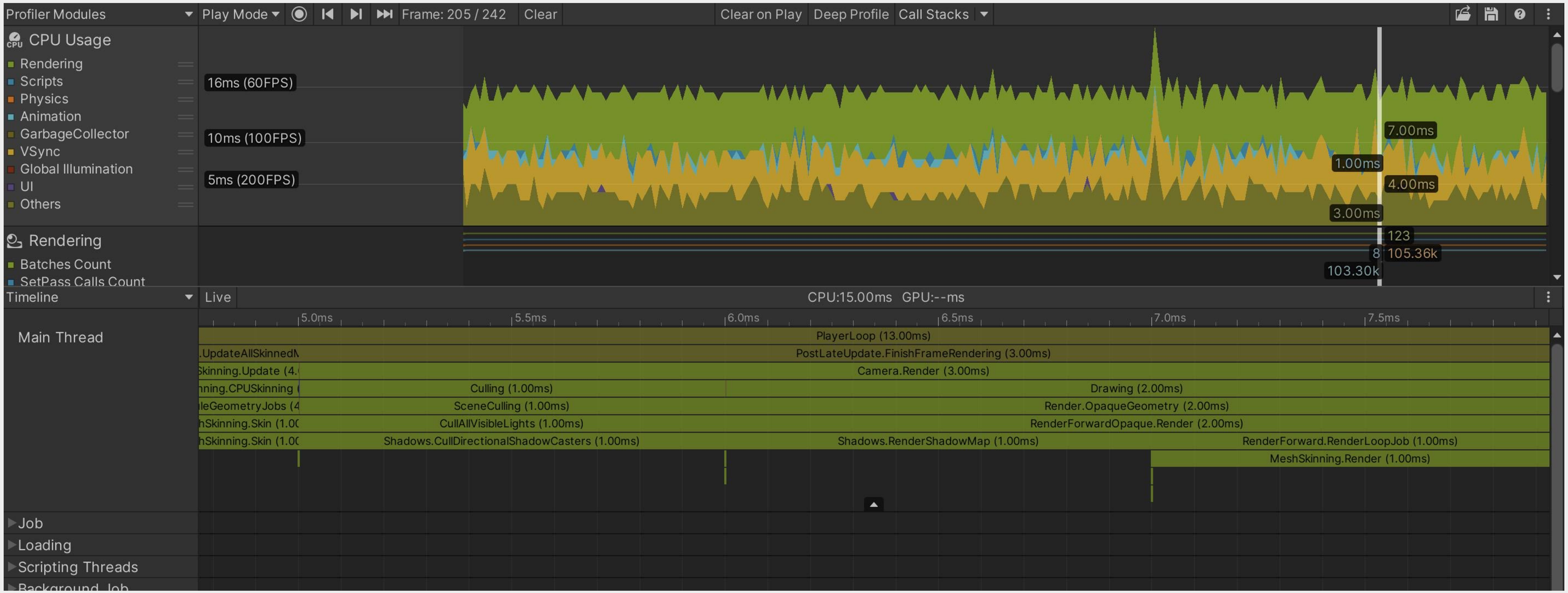




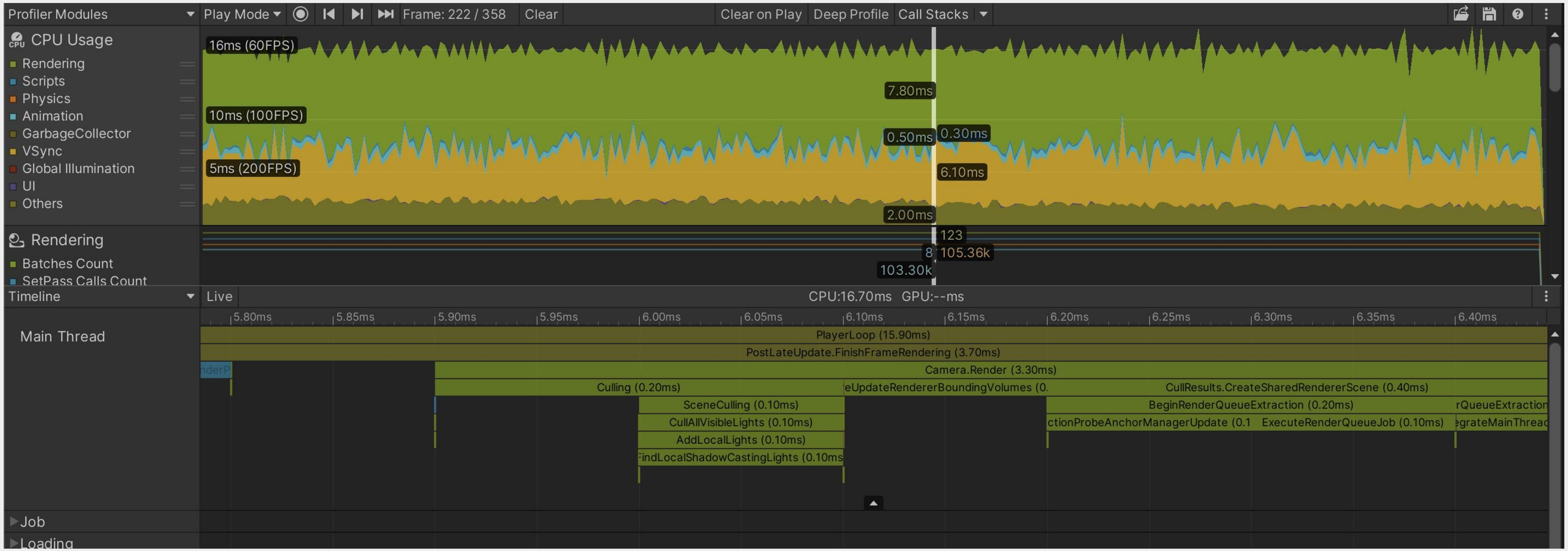
Dev Host开发工具



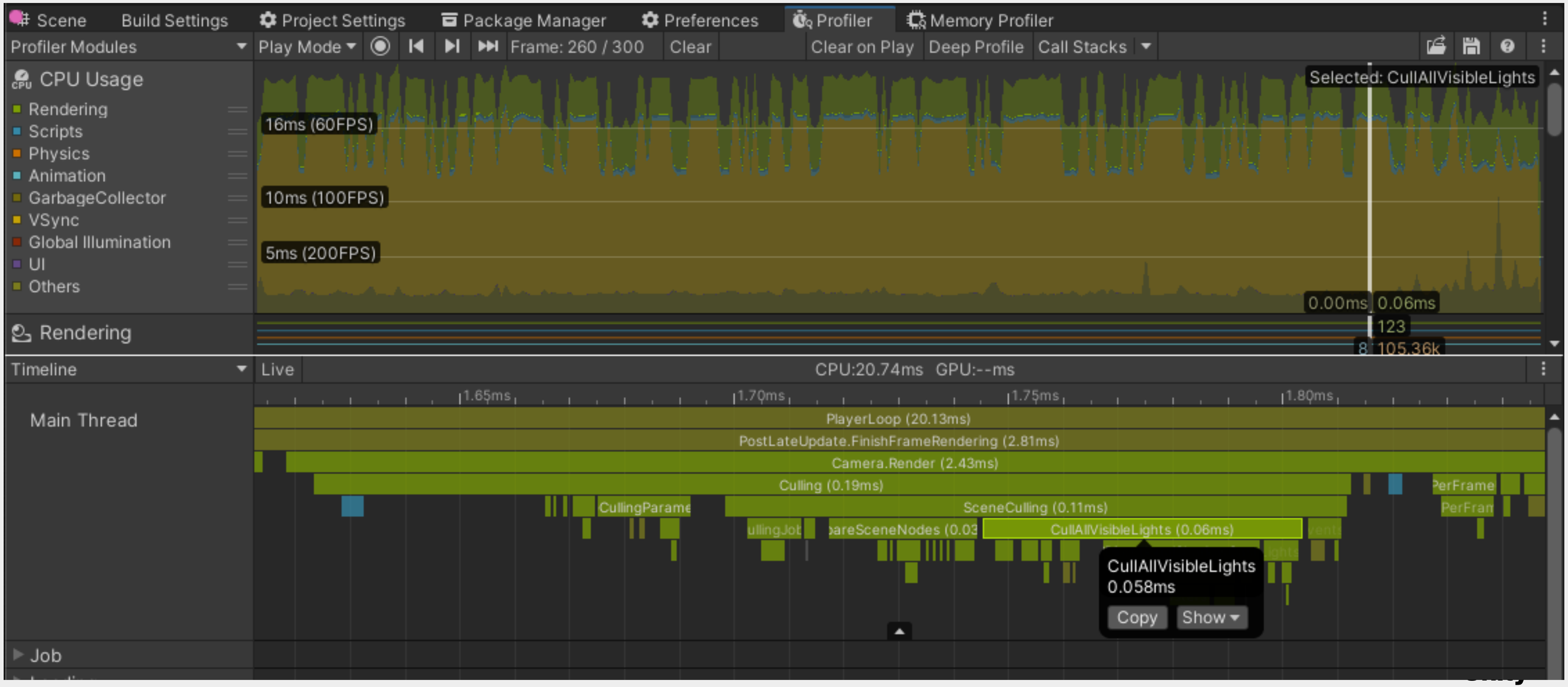
Dev Host开发工具



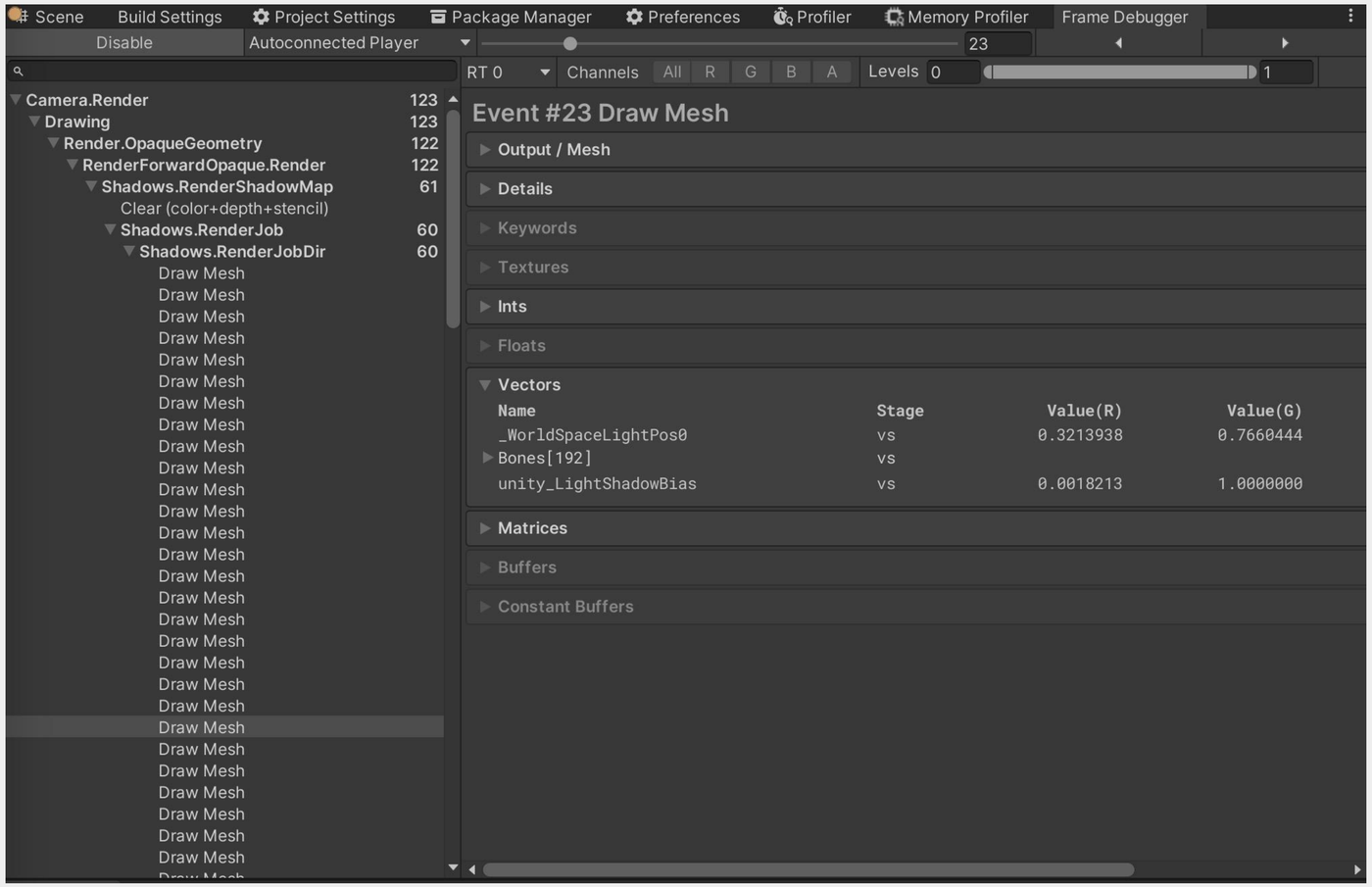
Dev Host开发工具



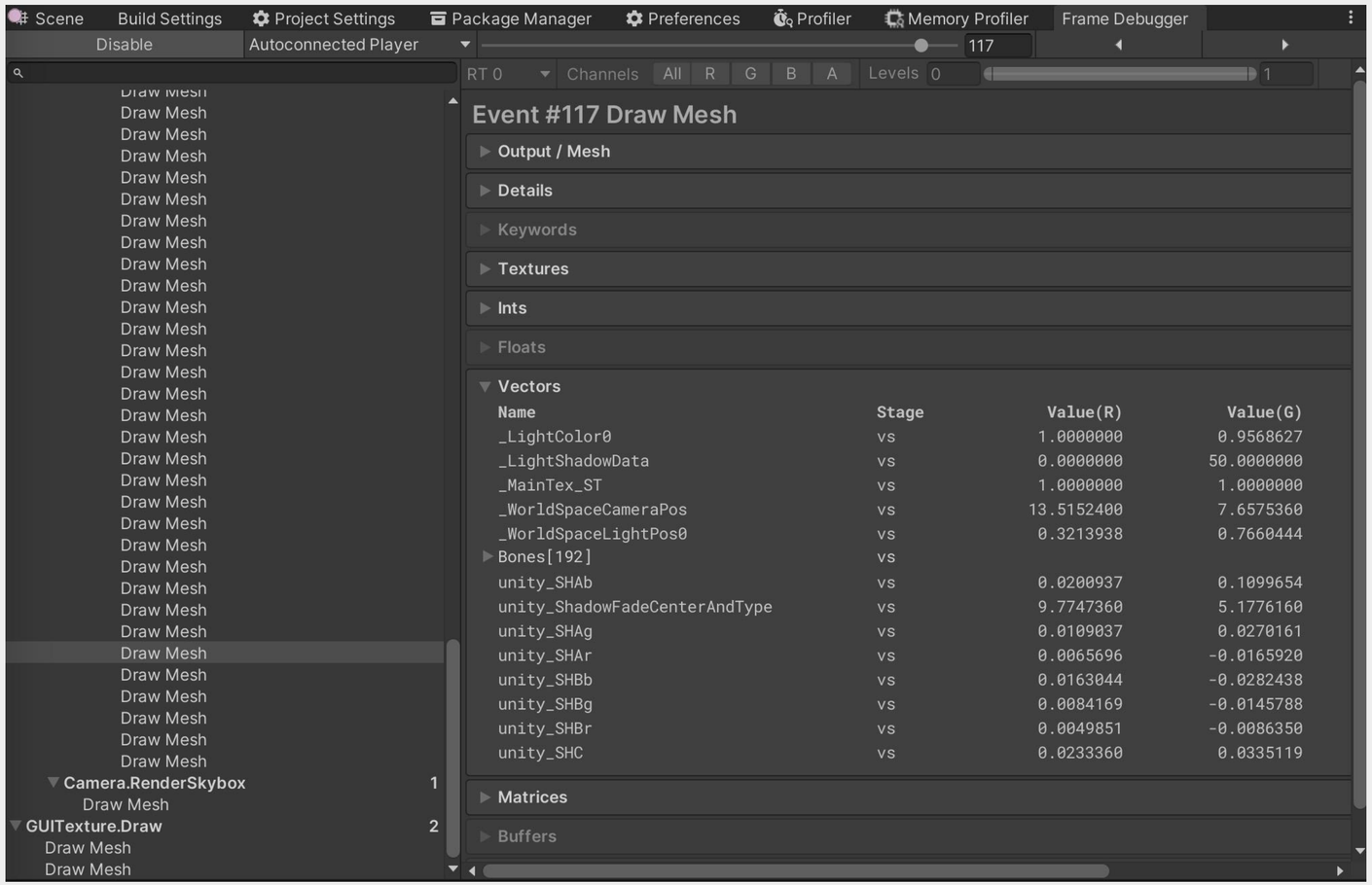
Dev Host开发工具



Frame Debugger



Frame Debugger



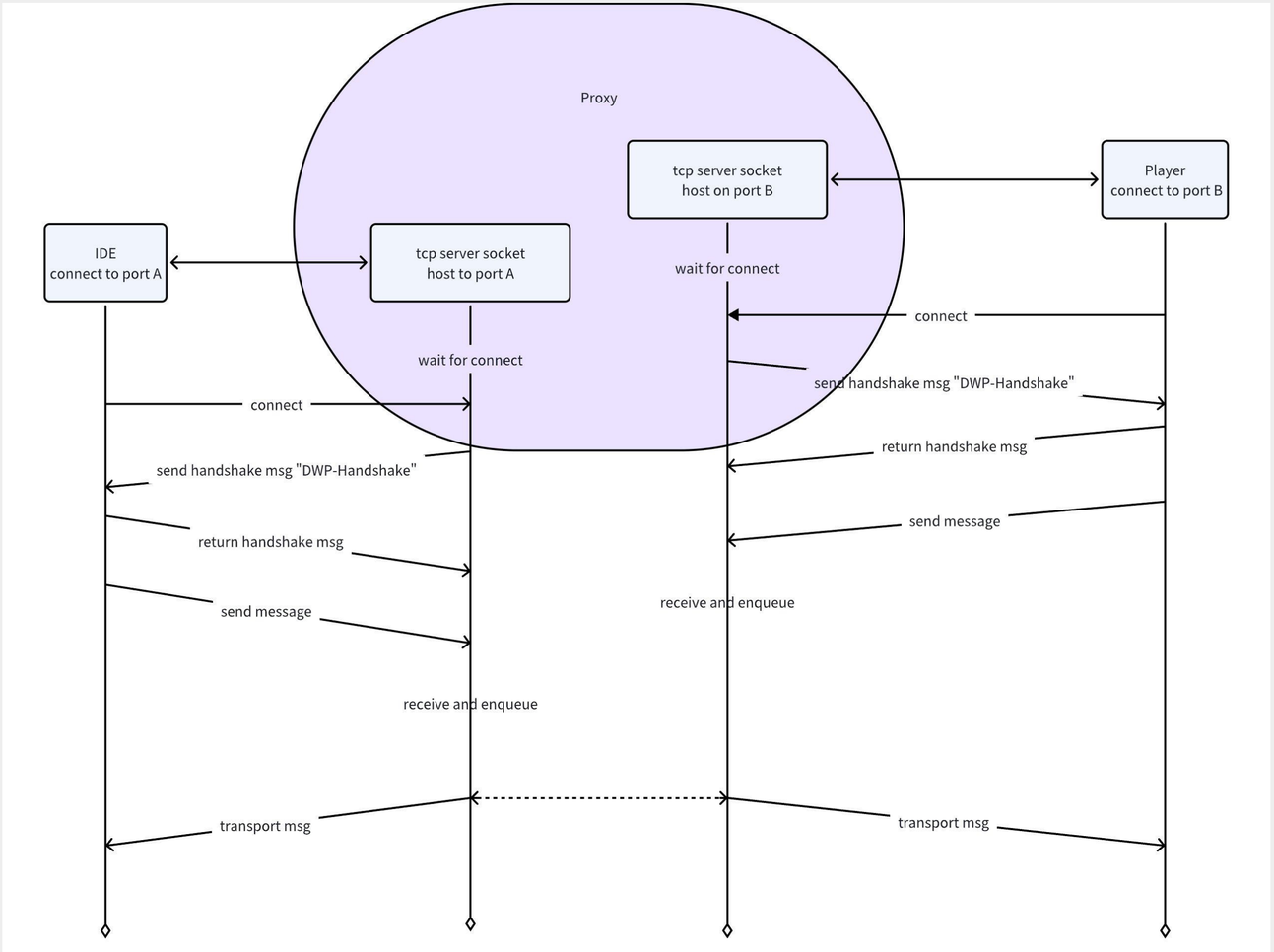
C# Debugging

Unity WebGL未支持代码调试的原因

- 多线程受限
- 不支持Socket
- 无法发送广播
- 无法监听端口等原因

微信小游戏平台的解决方案：

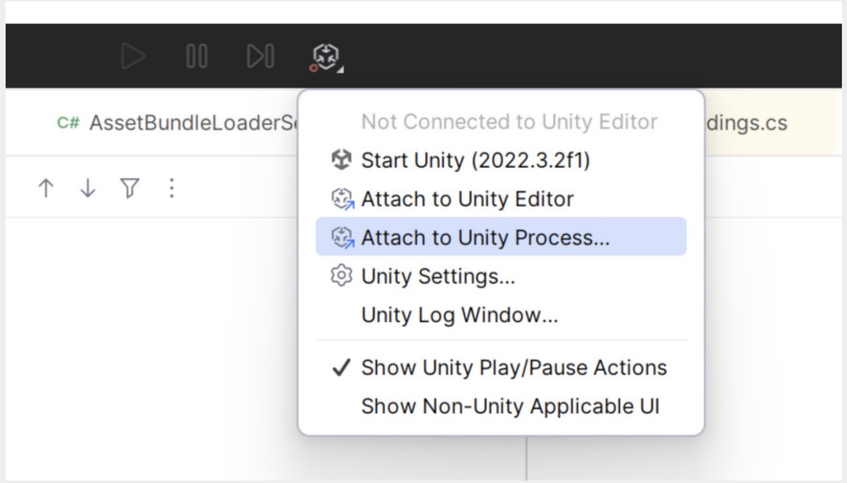
- 微信小游戏平台支持多线程；
- 小游戏宿主实现了广播、监听能力；
- 小游戏宿主增加中间代理，
  桥接WebSocket和Socket。

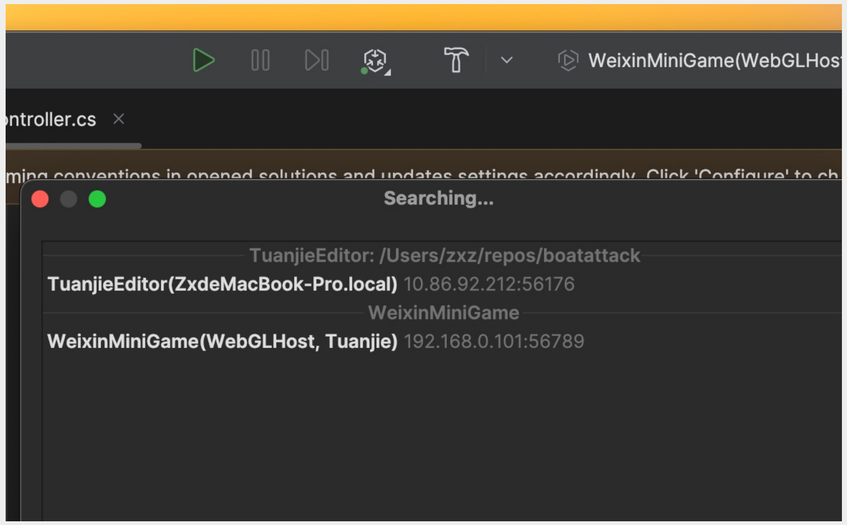


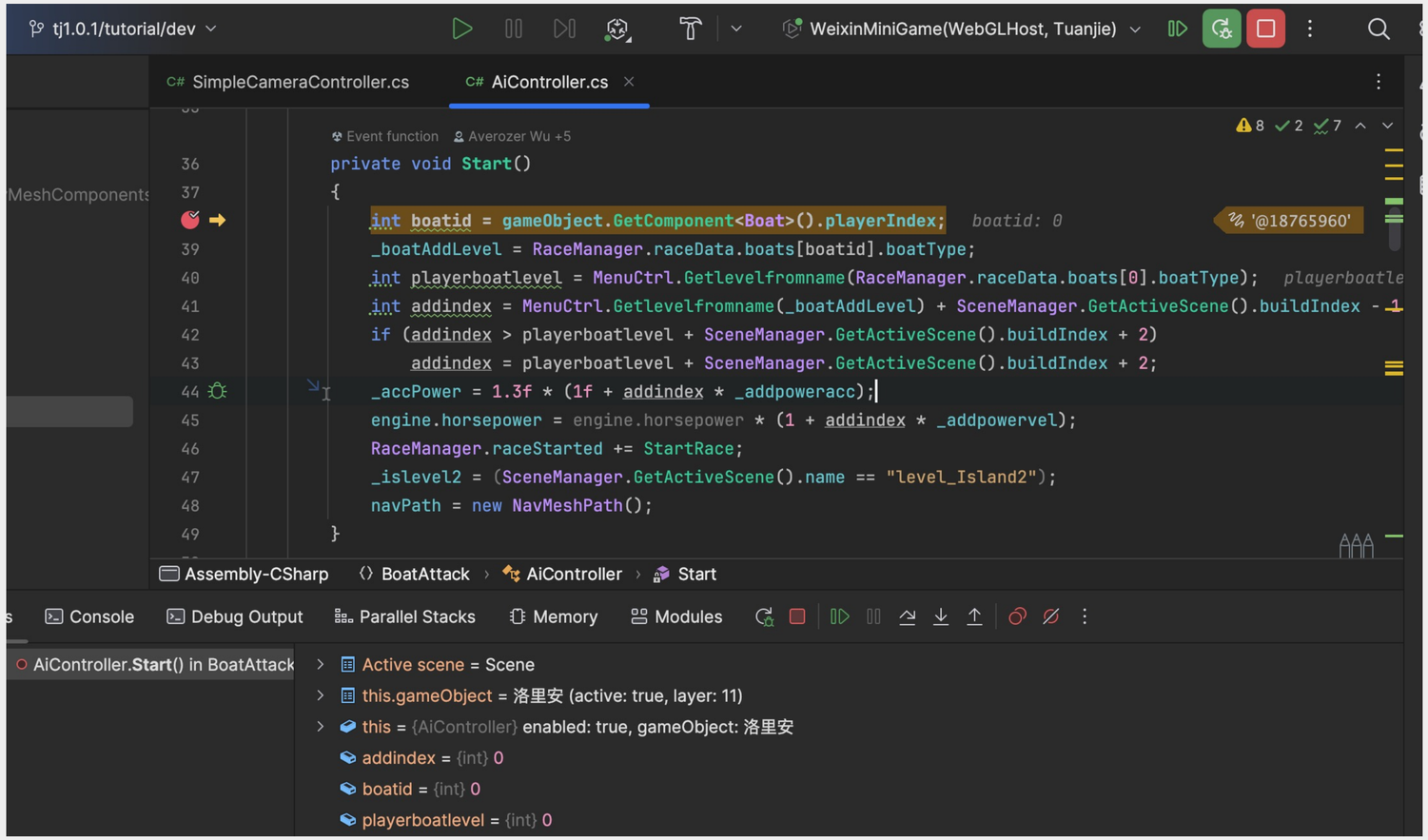
C# Debugging

- **使用Connnect小游戏宿主调试代码**
  - 打开Development Build和Script Debugging
  - 宿主打开C#代码调试
  - IDE和宿主运行在同一局域网内
  - 支持多个 IDE
    - Visual studio
    - Rider
  - 暂不支持.Net 8 scripting backend

C# Debugging


C# AssetBundleLoaderS
dings.cs
Not Connected to Unity Editor
Start Unity (2022.3.2f1)
Attach to Unity Editor
Attach to Unity Process...
Unity Settings...
Unity Log Window...
Show Unity Play/Pause Actions
Show Non-Unity Applicable UI


WeixinMiniGame(WebGLHost
ntroller.cs
Searching...
TuanjieEditor: /Users/zxz/repos/boatattack
TuanjieEditor(ZxdeMacBook-Pro.local) 10.86.92.212:56176
WeixinMiniGame
WeixinMiniGame(WebGLHost, Tuanjie) 192.168.0.101:56789


tj1.0.1/tutorial/dev
WeixinMiniGame(WebGLHost, Tuanjie)
C# SimpleCameraController.cs
C# AiController.cs
Event function Averozer Wu +5
private void Start()
{
int boatid = gameObject.GetComponent<Boat>().playerIndex; boatid: 0
'@18765960'
_boatAddLevel = RaceManager.raceData.boats[boatid].boatType;
int playerboatlevel = MenuCtrl.Getlevelfromname(RaceManager.raceData.boats[0].boatType);
int addindex = MenuCtrl.Getlevelfromname(_boatAddLevel) + SceneManager.GetActiveScene().buildIndex - 1
if (addindex > playerboatlevel + SceneManager.GetActiveScene().buildIndex + 2)
addindex = playerboatlevel + SceneManager.GetActiveScene().buildIndex + 2;
_accPower = 1.3f * (1f + addindex * _addpoweracc);
engine.horsepower = engine.horsepower * (1 + addindex * _addpowervel);
RaceManager.raceStarted += StartRace;
_islevel2 = (SceneManager.GetActiveScene().name == "level_Island2");
navPath = new NavMeshPath();
}
Assembly-CSharp BoatAttack AiController Start
Console
Debug Output
Parallel Stacks
Memory
Modules
AiController.Start() in BoatAttack
Active scene = Scene
this.gameObject = 洛里安 (active: true, layer: 11)
this = {AiController} enabled: true, gameObject: 洛里安
addindex = {int} 0
boatid = {int} 0
playerboatlevel = {int} 0

优化

- 游戏数据分析
- IL2CPP Code Generation
- Code Optimization

内存优化（Unity 2022.3.13f1 vs 团结 1.1.3）

- 总内存减少95.8MB，排除Reserved 部分的37.5MB，剩余58.3MB为团结引擎优化效果

A comparison of all tracked memory in each capture.

A
Total Memory In Table: 310.3 MB　　Total Memory In Snapshot: 0 B

B
Total Memory In Table: 214.5 MB　　Total Memory In Snapshot: 0 B

Description	Count Differ...	Size Differe...	Size Differe...	Size In A	Size In B	Count In A	Count In B
Graphics Memory	+14 ▲		-8.9 MB	53.3 MB	44.4 MB	214	228
Unity Subsystems	0		-52.4 KB	310.1 KB	257.7 KB	1	1
Reserved	0		-4.3 MB	43.3 MB	39.0 MB	1	1
Unity Objects	+14 ▲		-4.5 MB	9.7 MB	5.2 MB	212	226
Scripting Memory	+4,314 ▲		-31.5 MB	67.7 MB	36.3 MB	495,907	500,221
Managed Objects	+4,314 ▲		306.2 KB	33.3 MB	33.6 MB	495,906	500,220
Reserved (Unused)	0		-31.8 MB	34.4 MB	2.7 MB	1	1
Native Memory	+742 ▲		-55.5 MB	189.3 MB	133.8 MB	20,859	21,601
Reserved	0		-1.4 MB	31.0 MB	29.6 MB	1	1
Unity Subsystems	+389 ▲		-24.0 MB	86.8 MB	62.8 MB	744	1,133
Unity Objects	+353 ▲		-30.0 MB	71.5 MB	41.4 MB	20,114	20,467

内存优化（Unity 2022.3.13f1 vs 团结 1.1.3）

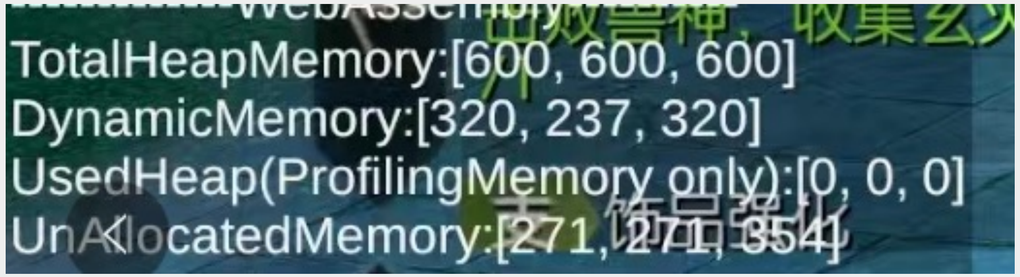
- il2cpp内存优化， 减少内存 22.6MB

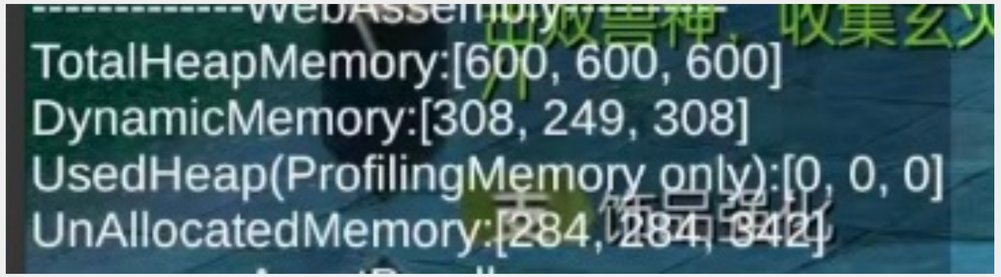
Native Memory	+742 ▲	-55.5 MB	189.3 MB	133.8 MB	20,859	21,601
Unity Objects	+353 ▲	-30.0 MB	71.5 MB	41.4 MB	20,114	20,467
Unity Subsystems	+389 ▲	-24.0 MB	86.8 MB	62.8 MB	744	1,133
Managers	+1 ▲	-23.7 MB	71.5 MB	47.7 MB	10	11
IL2CPPMemoryAllocator	0	-22.6 MB	66.3 MB	43.8 MB	1	1
BaseObjectManager	0	-1.2 MB	4.1 MB	2.8 MB	1	1

- Shader内存优化， 减少内存24.7MB

Native Memory	+742 ▲	-55.5 MB	189.3 MB	133.8 MB	20,859	21,601
Unity Objects	+353 ▲	-30.0 MB	71.5 MB	41.4 MB	20,114	20,467
Shader	0	-24.7 MB	32.1 MB	7.4 MB	76	76
AnimationClip	-15	-1.9 MB	9.1 MB	7.2 MB	102	87

- 内存分配器优化（仅release版本），减少约12MB





内存优化（Unity 2022.3.13f1 vs 团结 1.1.3）

- Remapper内存优化, 减少内存 1.1MB

Native Memory	+742 ▲	-55.5 MB	189.3 MB	133.8 MB	20,859	21,601
Unity Objects	+353 ▲	-30.0 MB	71.5 MB	41.4 MB	20,114	20,467
Unity Subsystems	+389 ▲	-24.0 MB	86.8 MB	62.8 MB	744	1,133
Managers	+1 ▲	-23.7 MB	71.5 MB	47.7 MB	10	11
PersistentManager.Remapper	0	-1.1 MB	2.0 MB	0.9 MB	1	1

- Vertex Shader GPU Skinninig, 减少显存 1.7MB

SkinnedMeshRenderer	-46	-1.7 MB	1.7 MB	0 B	46	0
Qc_shenmuzhi_1	-5	-463.5 KB	463.5 KB	0 B	5	0
Npc_xiaoheiwuchang_2	-5	-295.7 KB	295.7 KB	0 B	5	0
Fx_qingyunxinshounan_1_hair	-1	-190.9 KB	190.9 KB	0 B	1	0
Fx_xinfenxiangnnv_1_hair	-1	-111.2 KB	111.2 KB	0 B	1	0
Sz_xinqingyunnan_1_bt	-3	-106.6 KB	106.6 KB	0 B	3	0

- WebGL 2.0 Alpha8纹理支持, 减少显存 4.3MB

Graphics Memory	+14 ▲	-8.9 MB	53.3 MB	44.4 MB	214	228
Unity Objects	+14 ▲	-4.5 MB	9.7 MB	5.2 MB	212	226
Texture2D	-4	-4.3 MB	6.1 MB	1.8 MB	87	83
Font Texture	0	-3.2 MB	4.2 MB	1.1 MB	2	2
Medium04	0	-127.0 KB	128.0 KB	1.0 KB	1	1
Thin02	0	-127.0 KB	128.0 KB	1.0 KB	1	1
Thin01	0	-127.0 KB	128.0 KB	1.0 KB	1	1

内存优化（Unity 2022.3.13f1 vs 团结 1.1.3）

il2cpp优化	22.0MB(内存)
Shader优化	13.5MB(内存)
内存分配器优化	12MB(内存)
Remapper优化	1.1MB(内存)
GPU Skinninig	1.7MB(显存)
Alpha8 纹理支持	4.3MB(显存)
总计	54.6MB

切换团结

Process...	Process Name	Responsible...	User...	% CPU	CPU Time	# Th...	Memory
7471	com.apple.WebKit.WebContent (7471)	launchd (1)	mob...	27.6%	39.50 s	26	806.13 MiB
7461	WeChat (7461)	launchd (1)	mob...	13.8%	26.39 s	103	650.85 MiB

unity 2022.3.13f1

Process...	Process Name	Responsible...	User...	% CPU	CPU Time	# Th...	Memory
7501	com.apple.WebKit.WebContent (7501)	launchd (1)	mob...	28.7%	49.54 s	23	755.18 MiB
7493	WeChat (7493)	launchd (1)	mob...	12.7%	28.12 s	95	624.61 MiB

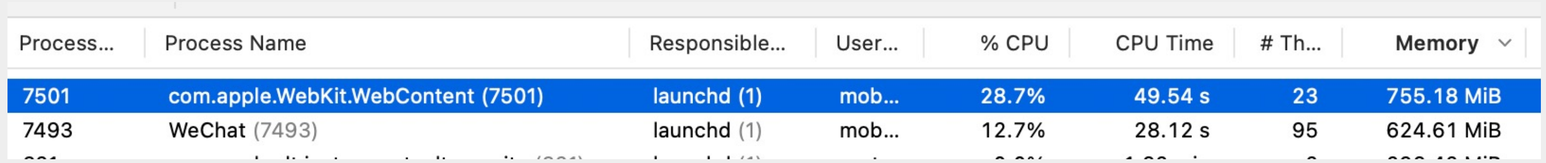
团结 1.1.3 il2cpp

内存优化（Unity 2022.3.13f1 vs 团结 1.1.3）

il2cpp VM	43.5MB(内存)
globalmetadata	13.3MB(内存)
wasm编译内存	73MB(预估内存)
总计	129.8MB

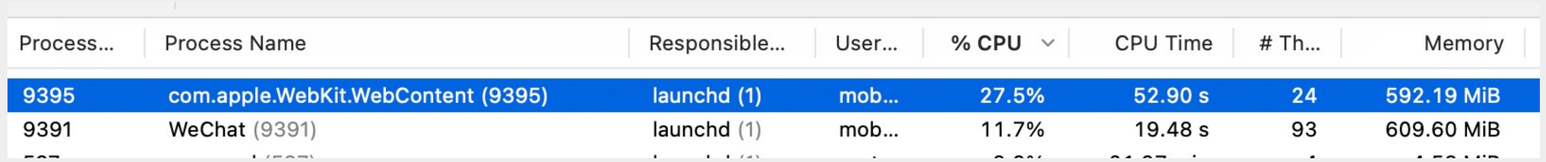
il2cpp wasm (分包)	21.9MB， br: 4.15MB
.Net 8 wasm	14.6MB， br: 3.57MB

使用 .Net 8



Process...	Process Name	Responsible...	User...	% CPU	CPU Time	# Th...	Memory
7501	com.apple.WebKit.WebContent (7501)	launchd (1)	mob...	28.7%	49.54 s	23	755.18 MiB
7493	WeChat (7493)	launchd (1)	mob...	12.7%	28.12 s	95	624.61 MiB

团结 1.1.3 il2cpp

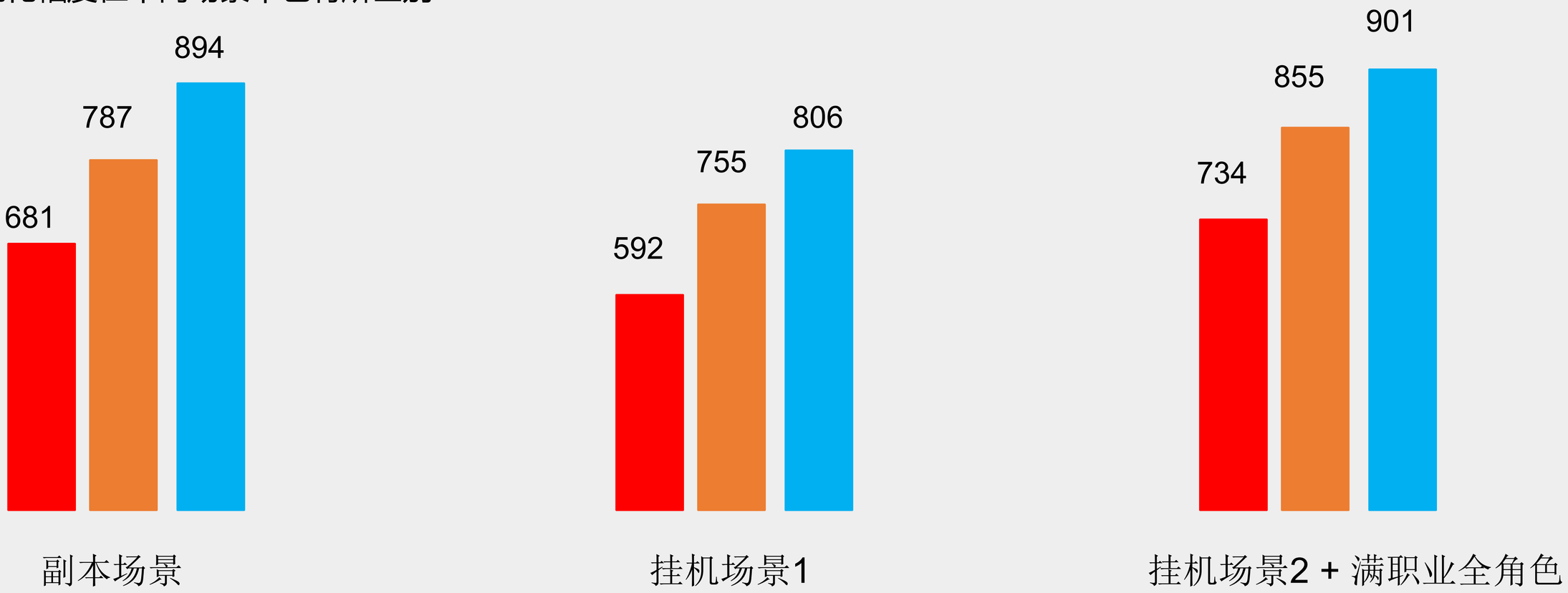


Process...	Process Name	Responsible...	User...	% CPU	CPU Time	# Th...	Memory
9395	com.apple.WebKit.WebContent (9395)	launchd (1)	mob...	27.5%	52.90 s	24	592.19 MiB
9391	WeChat (9391)	launchd (1)	mob...	11.7%	19.48 s	93	609.60 MiB

团结 1.1.3 .Net 8

内存优化（Unity 2022.3.13f1 vs 团结 1.1.3）

- 不同游戏场景，游戏内存差异较大
- 内存优化幅度在不同场景中也有所区别



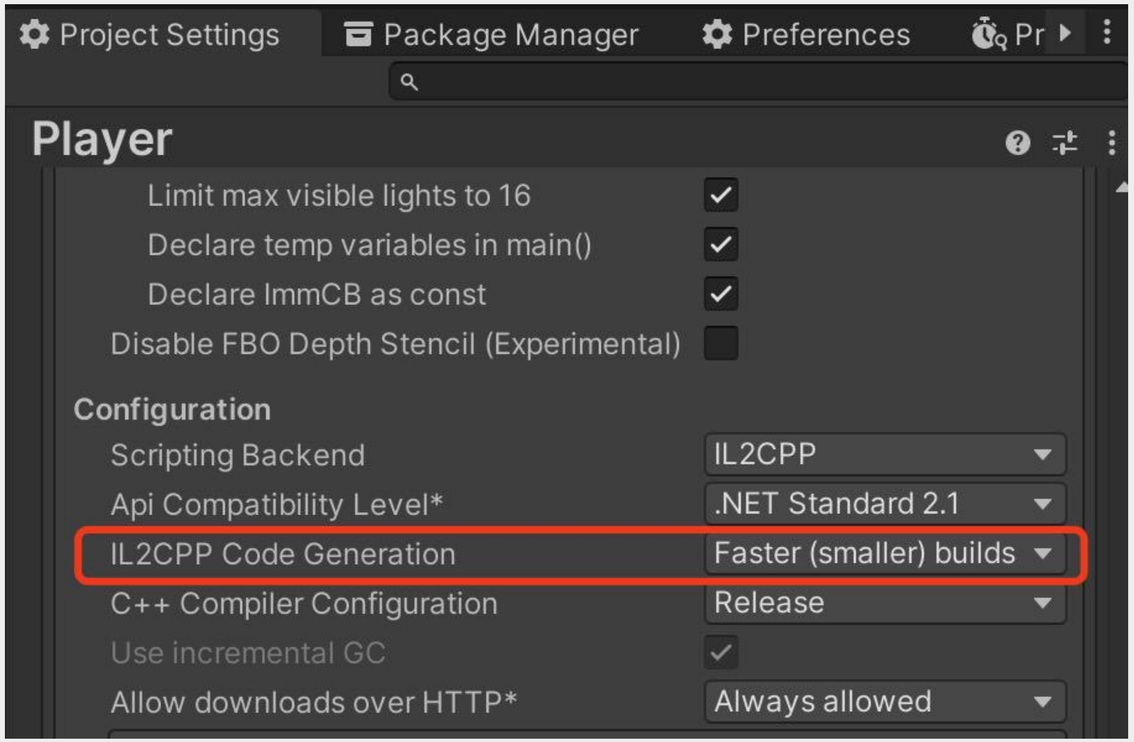
IL2CPP vs .Net 8

- 以C#为主，wasm代码较大的游戏，.Net 8方案有明显内存优势；
- Lua较重，wasm代码小的游戏，il2cpp方案内存可能更优；

		Tuanjie IL2CPP	Tuanjie .NET 8
MMO 1, C#逻辑 **Performance +**	分包前wasm	未压缩：48.7MB， br压缩:9.24MB	未压缩：14.6MB， br压缩:3.57MB
	分包后wasm	未压缩：21.9MB， br压缩:4.15MB	未使用分包
	内存均值	855 MB	734 MB
	CPU均值	41.7%	33.8%
MMO 2，Lua逻辑	分包前wasm	未压缩：17.4MB	未压缩：13.2MB
	分包后wasm	未知	未使用分包
	内存均值	1053 MB	1147 MB
	CPU均值	12.2%	11.6%

IL2CPP Code Generation

- WXSDK中的IL2Cpp Optimize Size(?)选项会覆盖PlayerSettings中的IL2CPP Code Generation



IL2CPP Code Generation

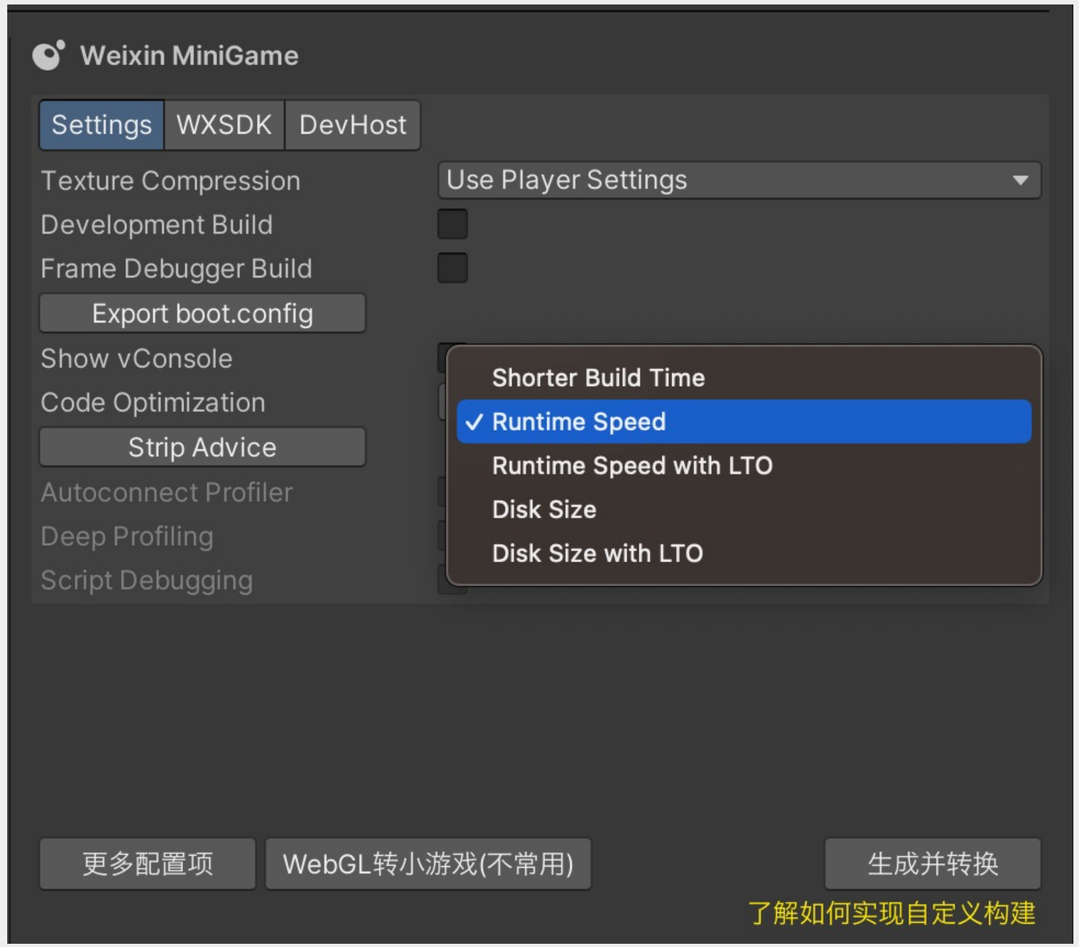
- Faster (smaller) builds
  - 缩短打包时间
  - 减小生成的wasm体积
  - 降低运行时性能
    - 对泛型容器（如List<T>）性能影响较为明显
- 使用Full Generic Sharing技术
  - https://blog.unity.com/engine-platform/il2cpp-full-generic-sharing-in-unity-2022-1-beta
- 推荐选项
  - 使用泛型容器较多的项目可以根据实际情况选用Faster runtime
  - 否则建议选用Faster (smaller) builds，在诛仙等游戏上没有明显区别

	游戏1 wasm	诛仙 wasm
Faster Runtime	28.1 MB	54.1 MB
Faster (smaller) builds	17.8 MB (-18%)	48.7 MB (-10%)

	游戏2 MainLoop耗时
Faster Runtime	20ms
Faster (smaller) builds	26ms (+30%)

Code Optimization

- Short Build Time
- Runtime Speed
- Disk Size
- with LTO
  - 启用LLVM的LTO功能，链接时进行跨模块优化
  - 更多inline，剔除dead code更彻底
  - 偶现Wasm代码生成异常，运行时抛出unreachable
- 推荐选项
  - 优先使用Runtime Speed（卡顿少、帧率高）
  - 对Wasm体积比较敏感时可选用Disk Size
    - 客户反馈，有的项目中，wasm减小后内存占用下降并不明显
  - 项目实际验证有收益且没有异常后可选用with LTO



Code Optimization

- 不同选项对应的编译、链接参数

	编译参数	链接参数	测试工程 wasm (KB)
Disk Size	O3	Os	15,714
Disk Size with LTO	Os flto	Os	15,217
Runtime Speed	O3	O3	16,889
Runtime Speed with LTO	O3 flto	O3	17,077
Shorter Build Time	O3	O2	17,041

Code Optimization

- 实测差异

	诛仙 wasm	诛仙 FPS (Android 865)
Runtime Speed	48.7 MB	27.8
Disk Size	45.3 MB (-7%)	15.5 (-55.8%)

开发中

- 启动优化
- 场景加载优化
- 多线程渲染

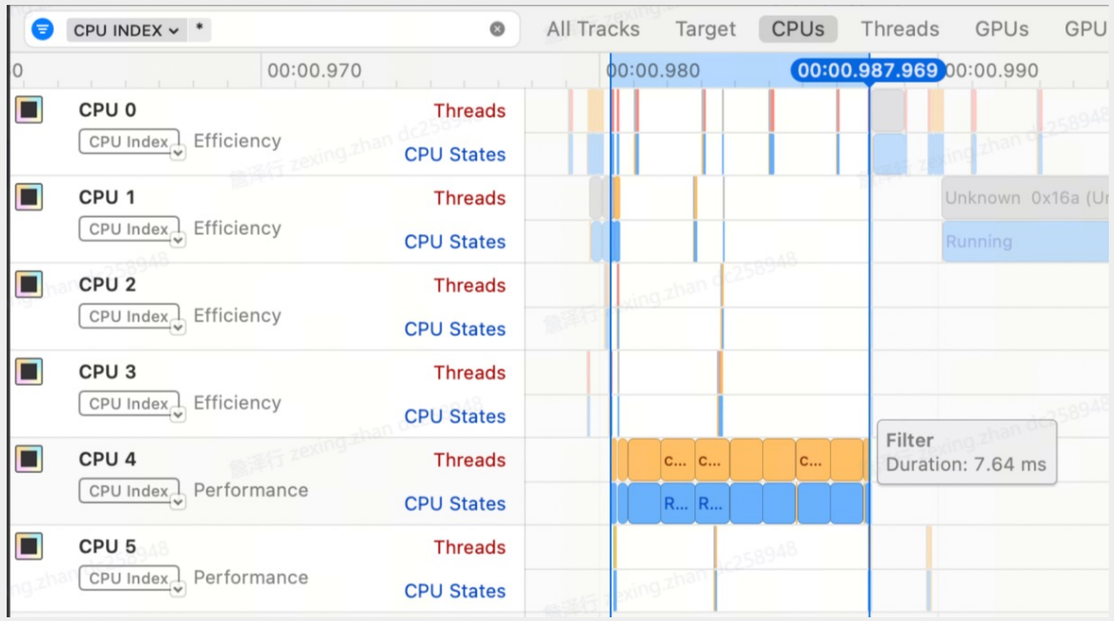
多线程功耗调研

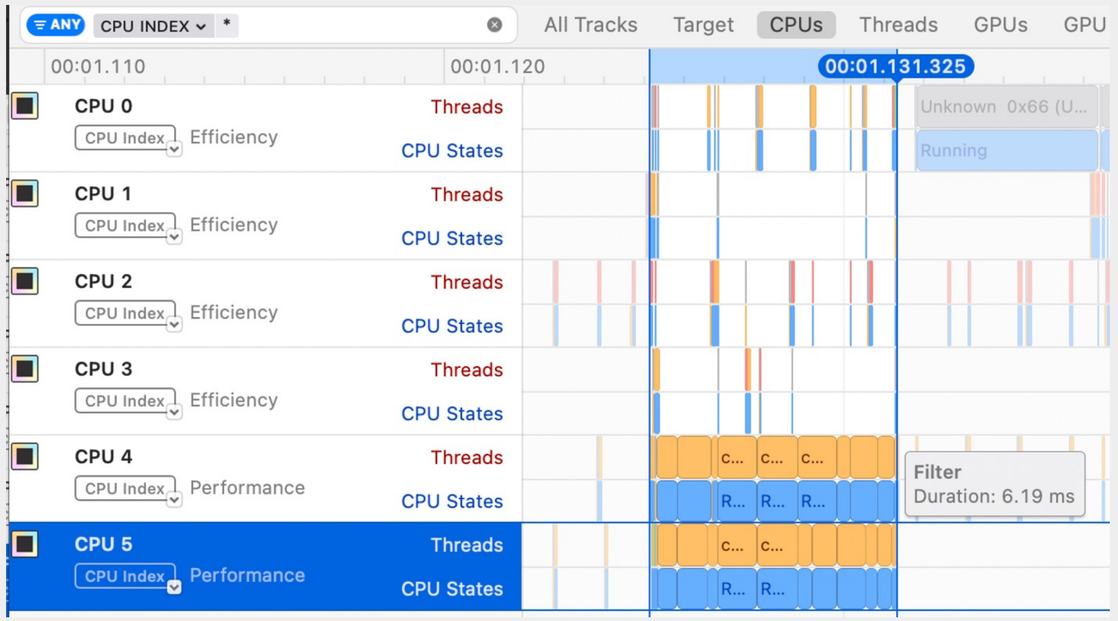
- **多线程**
  - 主线程 1倍工作负载
  - 一个Emscripten pthread worker 1倍工作负载
  - 60fps
  - 每帧主线程和worker同时开始工作

- **单线程**
  - 主线程 2倍工作负载
  - 60fps

测试过程中，屏幕亮度降到最低，退出其他程序，减少干扰

多线程功耗调研 (iOS Safari)





	内存（MB）	cpu使用率	cpu单帧耗时	每分钟耗电量	电池温度	电压	电池输出电流
单线程	21.31	45%	7.64ms	0.3359%	37.5	3100	701mA
多线程	39.52（+18.21）	71.3% (+26.3%)	6.19ms (-1.45ms)	0.1923%（-42.75% ）	33.89 (-3.61)	3100	411mA (-41.37%)

相同工作量，未扣除系统基础消耗情况下，多线程方案的功耗是单线程的 51.2% - 58.6%左右。

多线程功耗调研 (Android Chrome)

- **Android Chrome功耗**
  - 从满电开始运行，对比每分钟耗电
  - 不同设备多线程功耗差别较大
  - 和系统CPU调度策略有关

	cpu单帧耗时	每分钟耗电量
单线程	10.53ms	0.236%
多线程	6ms - 14ms （-4.53 - +3.47ms）	0.219% （-7.2%）

小米 MI Note3 （骁龙660， 电池 3500mAh)

	cpu单帧耗时	每分钟耗电量
单线程	11.26ms	0.275%
多线程	7.49ms （-3.77ms）	0.1933% (-29.8%)

Nokia 7 （骁龙630， 电池 3000mAh)

多线程渲染

背景

- 游戏一帧可以分为：游戏逻辑，渲染逻辑，Gfx指令提交
- Unity/团结多线程渲染仅将Gfx指令提交部分剥离进Worker线程
- 微信小游戏平台目前没有开启多线程渲染

预期收益

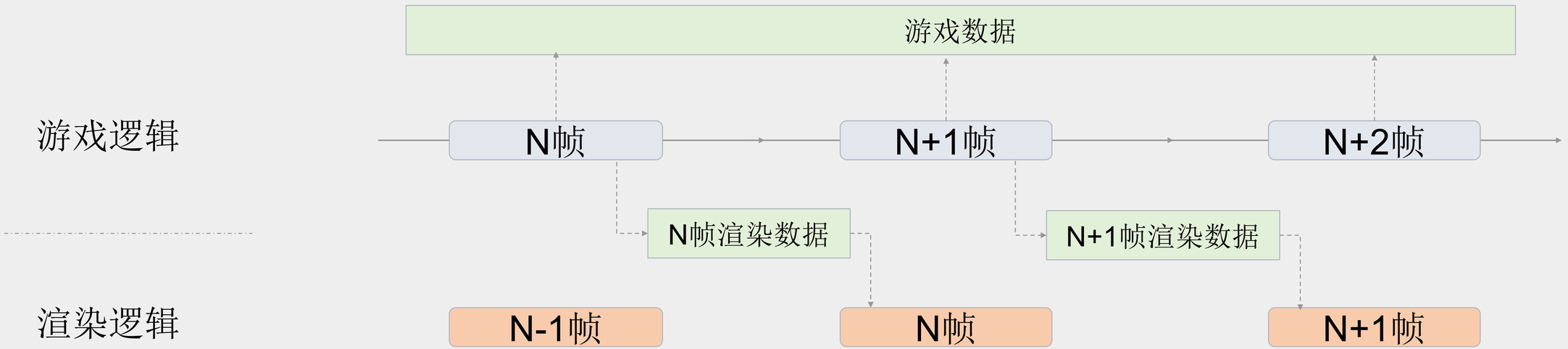
- 提升FPS
  - 每帧耗时更短
- 降低功耗和发热
  - 任务在更短时间内在多个线程上更密集地完成

多线程渲染

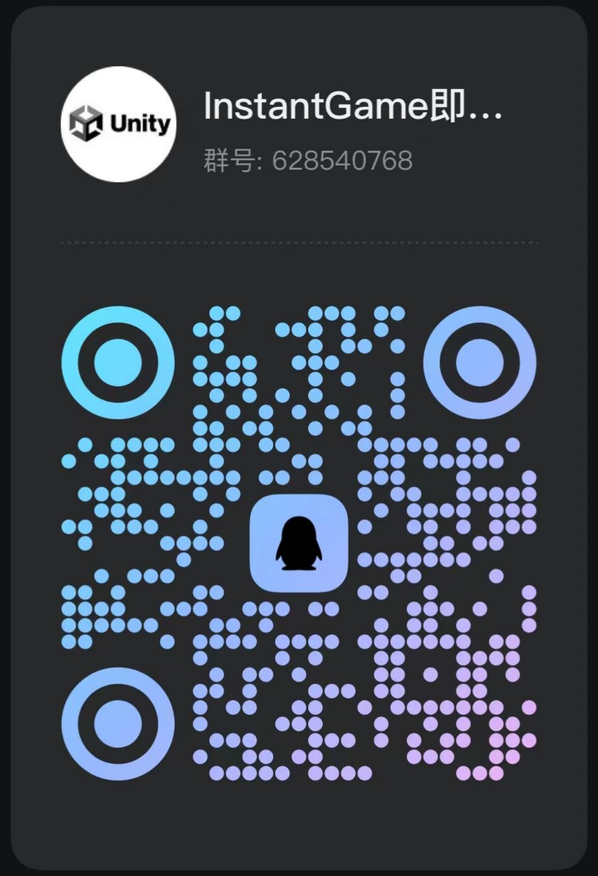
整体思路

- 逻辑拆分
  - 游戏逻辑 + 渲染逻辑，运行在各自的线程上
- 数据隔离
  - 游戏数据：由游戏逻辑持续维护和使用
  - 渲染数据：
    - 生产者-消费者模式：游戏逻辑产生，渲染逻辑使用
    - 全新设计数据结构，更加紧凑和高效
- 时序调整
  - 并行 + 错帧：游戏逻辑领先渲染一帧
  - 精密控制时序，最大程度并发，发挥多线程优势

多线程渲染



FAQ



扫一扫二维码，加入群聊

QQ

Thank you!

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