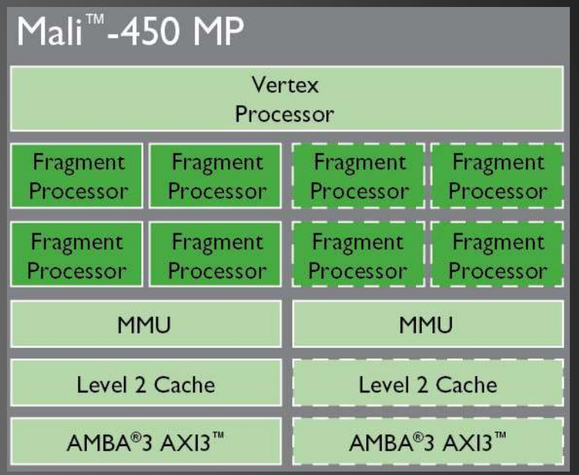
如何优化基于Unity开发的3D移动游戏

Unity Technologies China

当今的移动硬件

- ImgTech PowerVR SGX
- NVIDIA Tegra
- ARM Mali
- Qualcomm Adreno



unity

3D游戏从梦想逐渐变成了现实...

Shadow Gun (暗影之枪)

unity

3D游戏从梦想逐渐变成了现实...

Dead Trigger (死亡扳机)

unity

3D游戏从梦想逐渐变成了现实...

Dead Trigger 2 (死亡扳机2)

unity

3D游戏从梦想逐渐变成了现实...

Deus Ex:The Fall (杀出重围：陨落)

unity

3D游戏从梦想逐渐变成了现实...

Call of Duty: Strike Team (使命召唤：突击队)

unity

3D游戏从梦想逐渐变成了现实...

永恒战士3

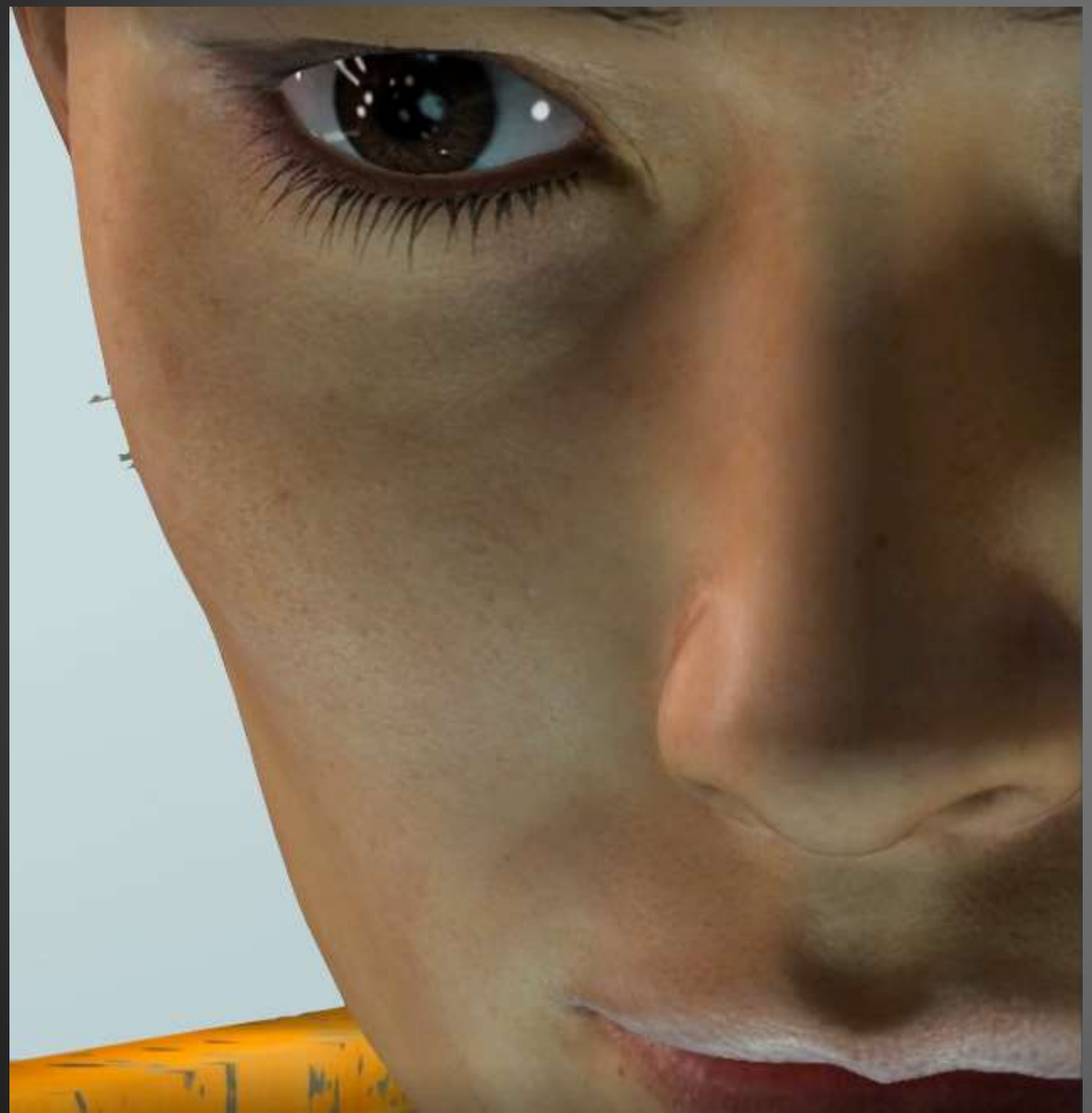
unity

3D游戏从梦想逐渐变成了现实...

The Chase

unity

3D游戏从梦想逐渐变成了现实...



unity

3D游戏从梦想逐渐变成了现实...

The Chase

unity

PC平台 vs. 移动平台

- 体积小
- 计算能力弱
- 带宽小

移动平台上实现效果出色的3D游戏，需要对流程进行严格的优化，以及对资源进行合理的使用！

unity

3D移动游戏的优化方向

- 资源相关优化
- 引擎相关优化
- 代码相关优化
- 着色器相关优化

资源相关优化

- 动态物体
- 静态物体
- 纹理数据
- 音频数据
- 程序包数据

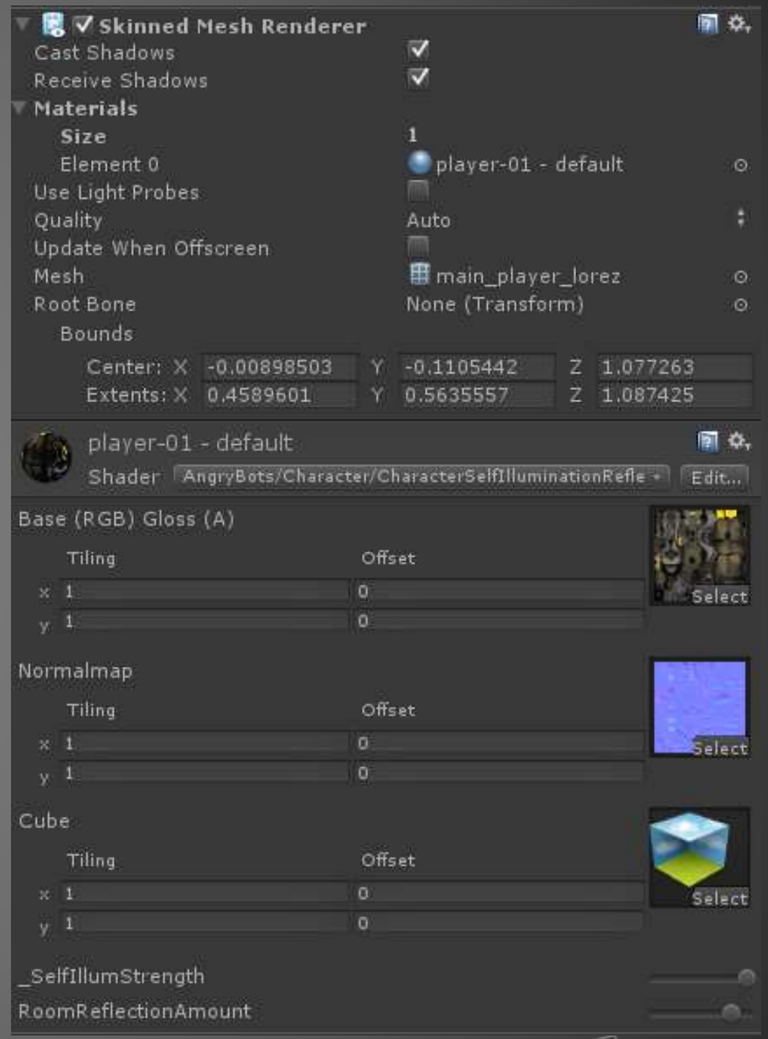
unity

资源相关优化

- 动态物体
  - 主角、NPC、怪物等
    - 控制面片数量
      - 300-2000面片

资源相关优化

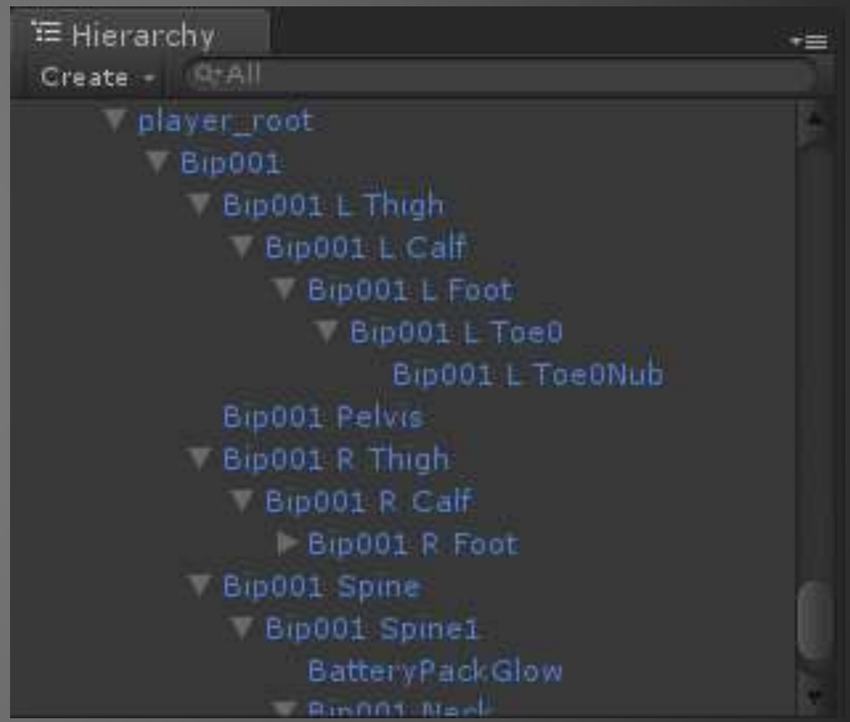
- 动态物体
  - 主角、NPC、怪物等
    - 控制面片数量
      - 300-2000面片
    - 控制Skinned Mesh Renderer数量
      - 1个
    - 控制材质数量
      - 1-3种



unity

资源相关优化

- 动态物体
  - 主角、NPC、怪物等
    - 控制面片数量
      - 300-2000面片
    - 控制Skinned Mesh Renderer数量
      - 1个
    - 控制材质数量
      - 1-3种
    - 控制骨骼数量
      - 小于30根

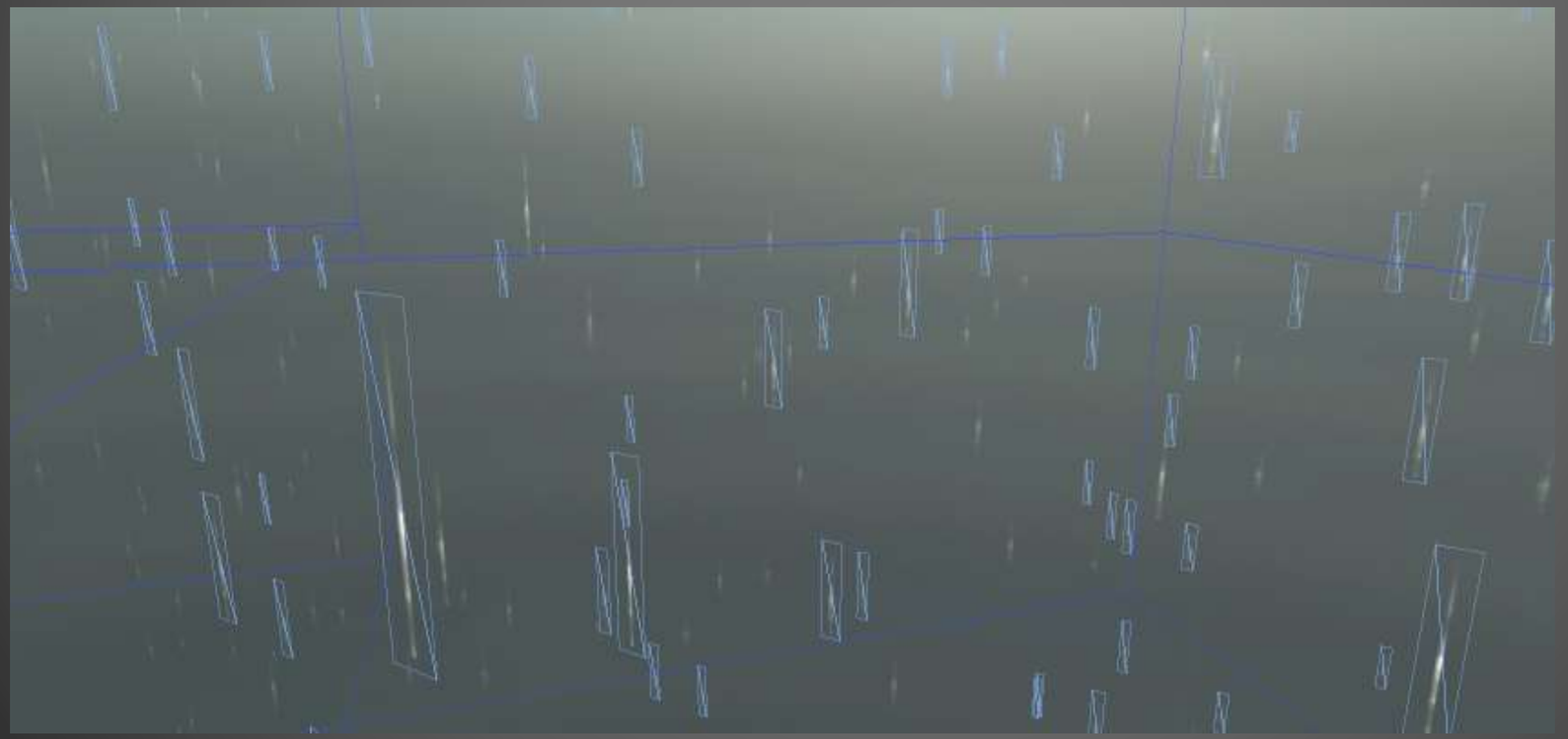


unity

资源相关优化

- 动态物体
  - 其他动态物体
    - 控制整体数据大小

      利用Dynamic Batching进行合批



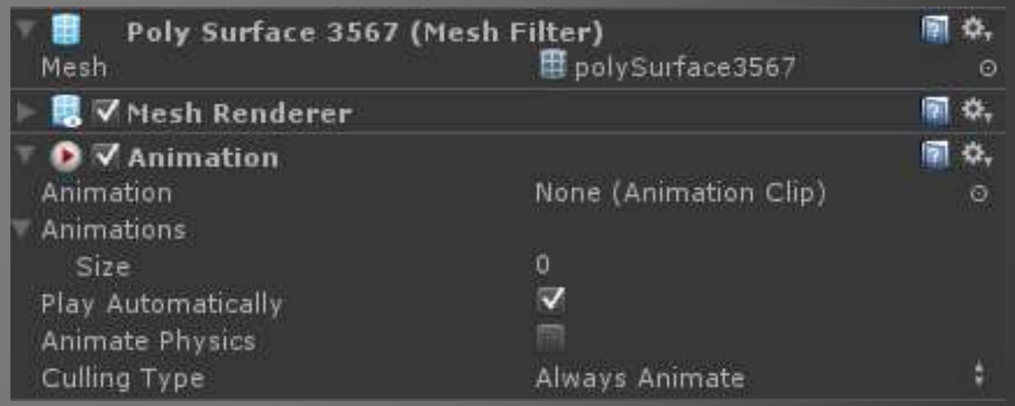
资源相关优化

- 静态物体
  - 一般静态物体
    - 控制网格顶点数
      - 少于500个
    - 标记为“Static”
      - Static Batching

unity

资源相关优化

- 静态物体
  - 一般静态物体
    - 控制网格顶点数
      - 少于500个
    - 标记为“Static”
      - Static Batching
    - Animation组件
      - 不要附加Animation 组件



unity

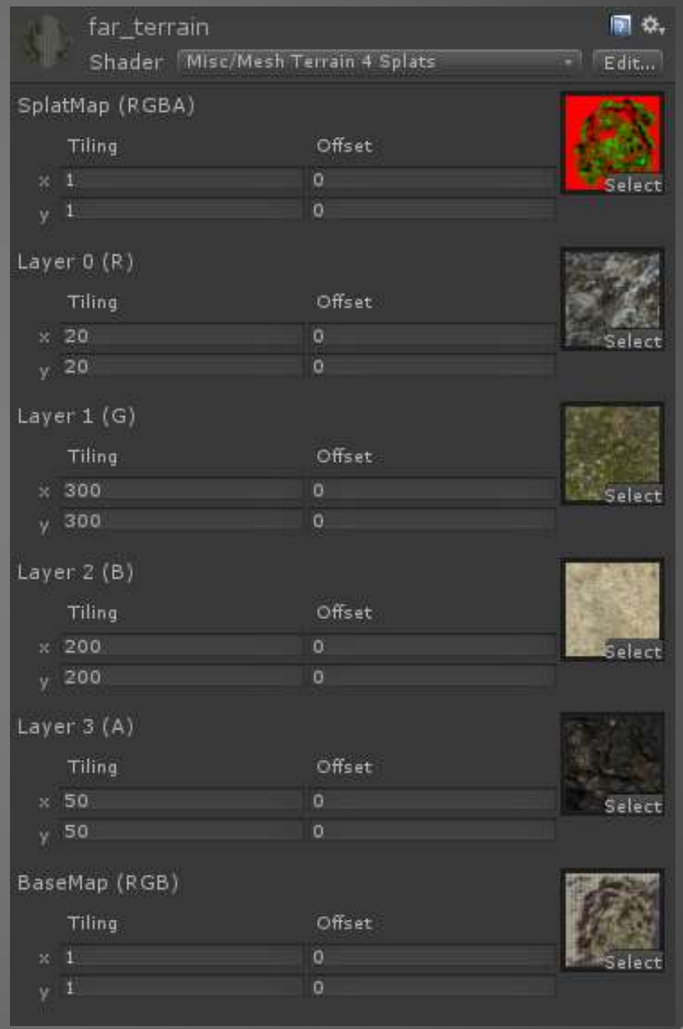
资源相关优化

- 静态物体
  - 地形
    - 控制地形的分辨率
      - 长宽均尽量小于257

unity

资源相关优化

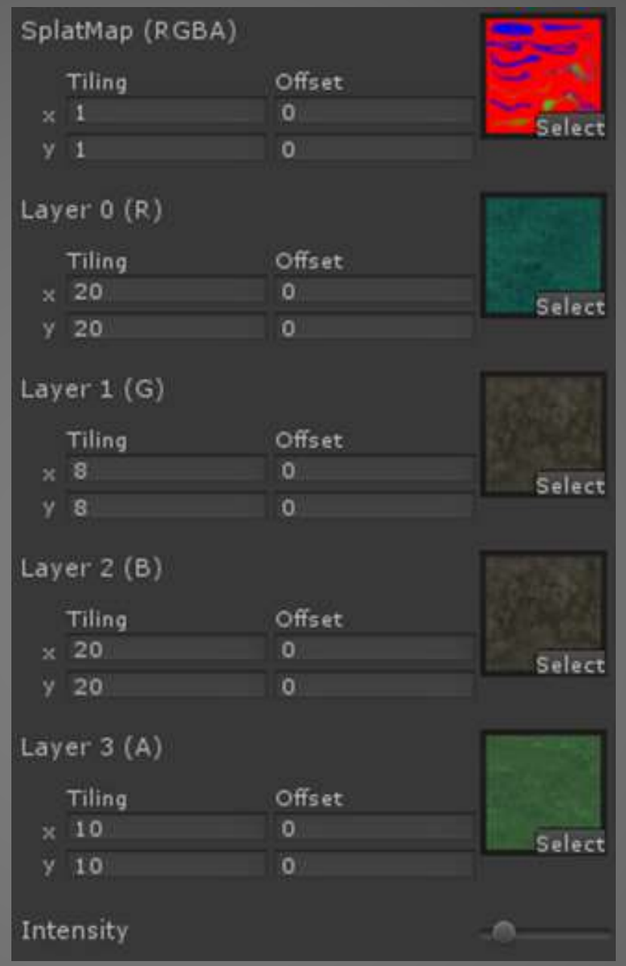
- 静态物体
  - 地形
    - 控制地形的分辨率
      - 长宽均尽量小于257
    - 混合纹理数量
      - 不要超过4



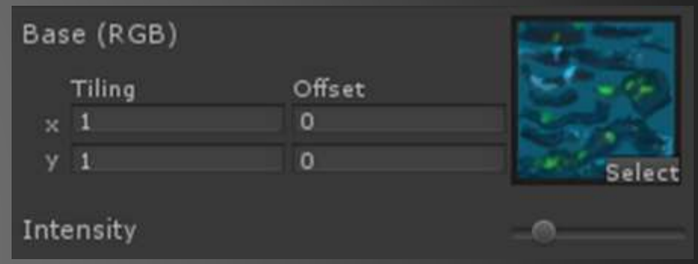
unity

资源相关优化

- 静态物体
  - 地形
    - 控制地形的分辨率
      - 长宽均尽量小于257
    - 混合纹理数量
      - 不要超过4



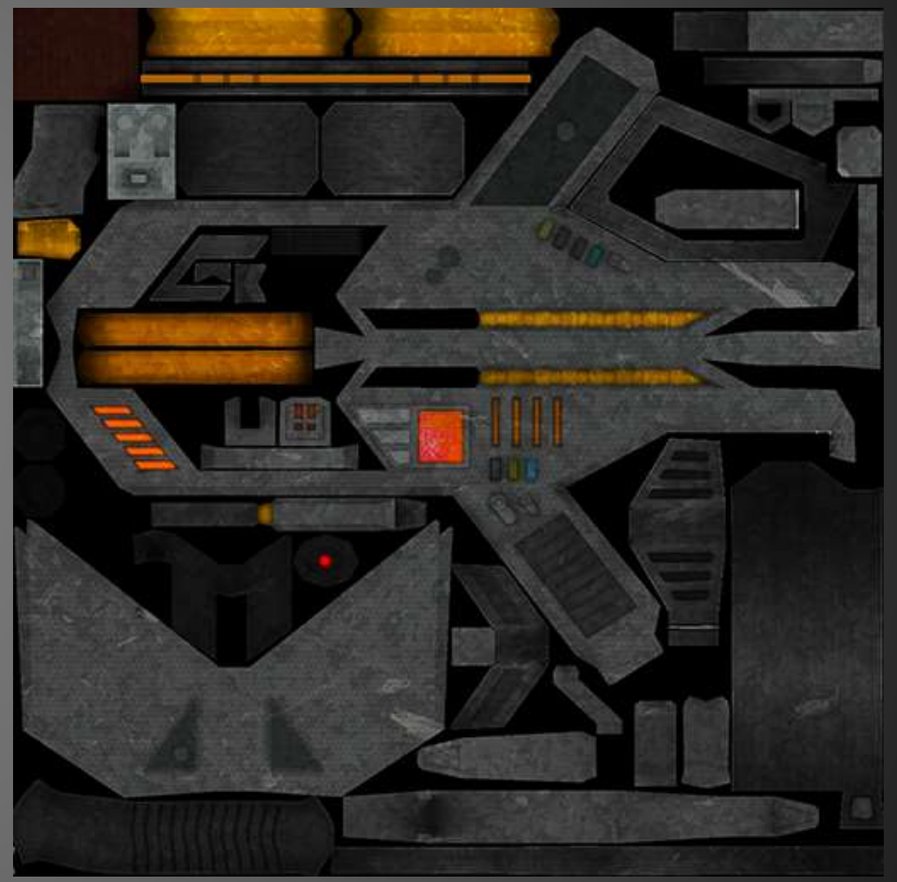
纹理融合前



纹理融合后

资源相关优化

- 纹理数据
  - 纹理格式
    - 建议png或tga
  - 纹理尺寸
    - 长宽小于1024。
      同时应该尽可能地小，够用就好

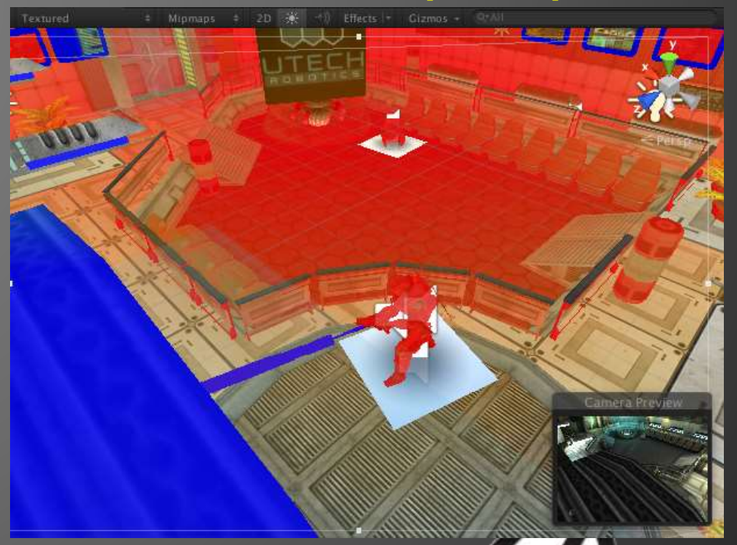


资源相关优化

- 纹理数据
  - 纹理格式
    - 建议png或tga
  - 纹理尺寸
    - 长宽小于1024。
      同时应该尽可能地小，够用就好，可使用Scene View中的MipMap模式

256 x 256

128 x 128



unity

资源相关优化

- 纹理数据
  - 纹理格式
    - 建议png或tga
  - 纹理尺寸
    - 长宽小于1024。
      同时应该尽可能地小，够用就好，可使用Scene View中的MipMap模式
  - 支持Mipmap
    - 建议生成Mipmap。
      虽然这种做法会增加一些应用程序的大小，
      但会提高一些渲染效率
  - 控制UV值范围
    - UV值范围尽量不要超过（0, 1）区间

资源相关优化

- 音频数据
  - 播放时间较长的音乐（如背景音乐）
    - 建议使用.ogg或.mp3的压缩格式
  - 播放时间较短的音乐（如枪声）
    - 使用.wav的未压缩音频格式

资源相关优化

- 程序包数据
  - 减小程序包的大小
    - 使用压缩格式的纹理
    - 使用压缩的网格和动画数据
    - 尽量不要使用System.Xml而使用较小的Mono.Xml
    - 启用Stripping来减小库的大小

Unity引擎相关优化

- 光源设置
- 相机设置
- 粒子特效
- 物理系统
- 渲染设置
- UI设置

Unity引擎相关优化

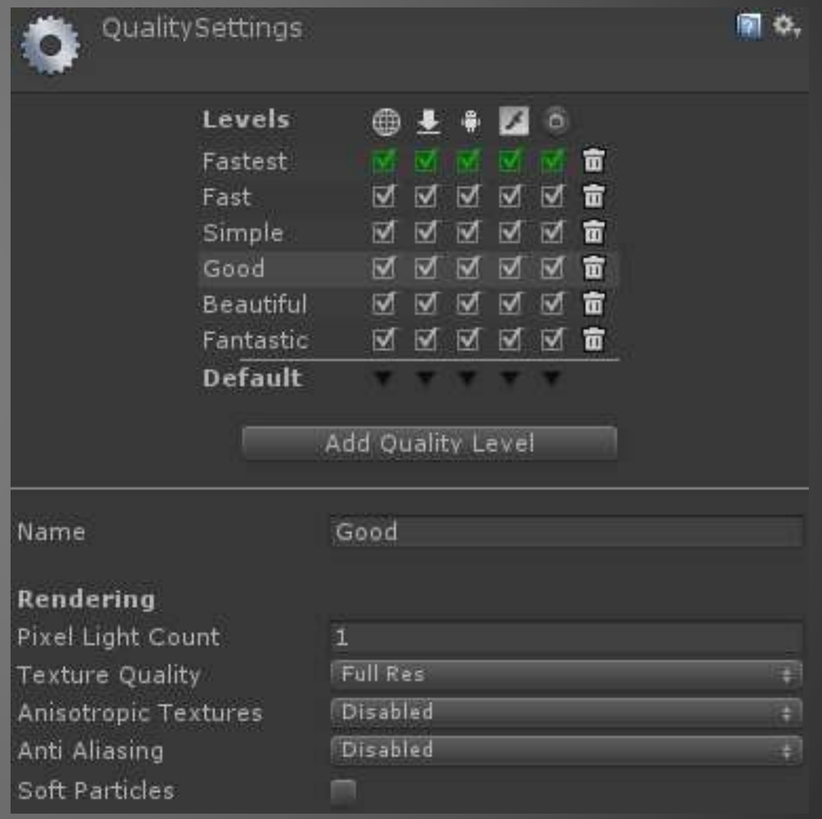
- 光源设置

Unity引擎相关优化

- 光源设置
  - 控制 “Important”光源个数
    - 建议1个或0个，一般为方向光
    - 个数越多，drawcall越多

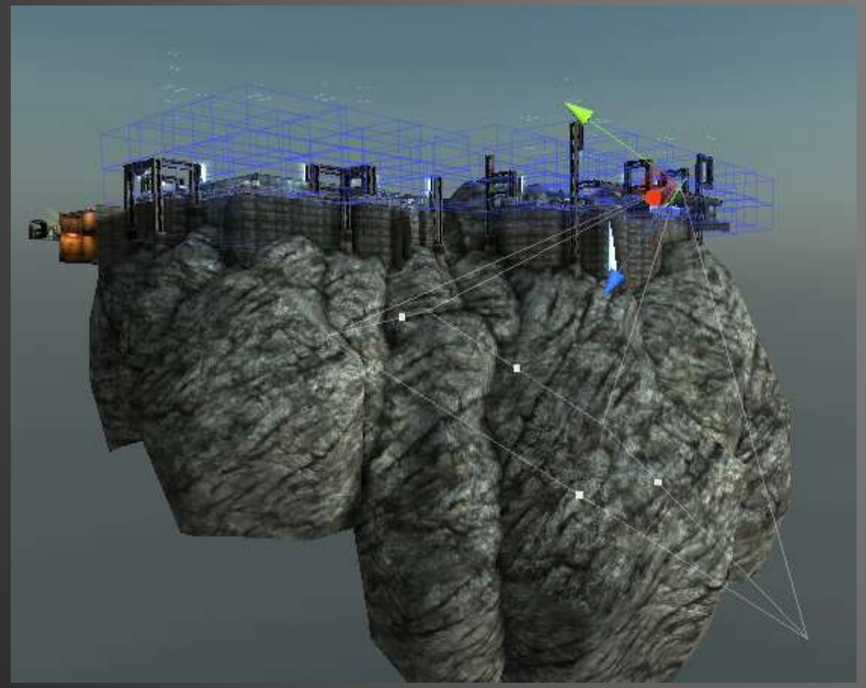
Unity引擎相关优化

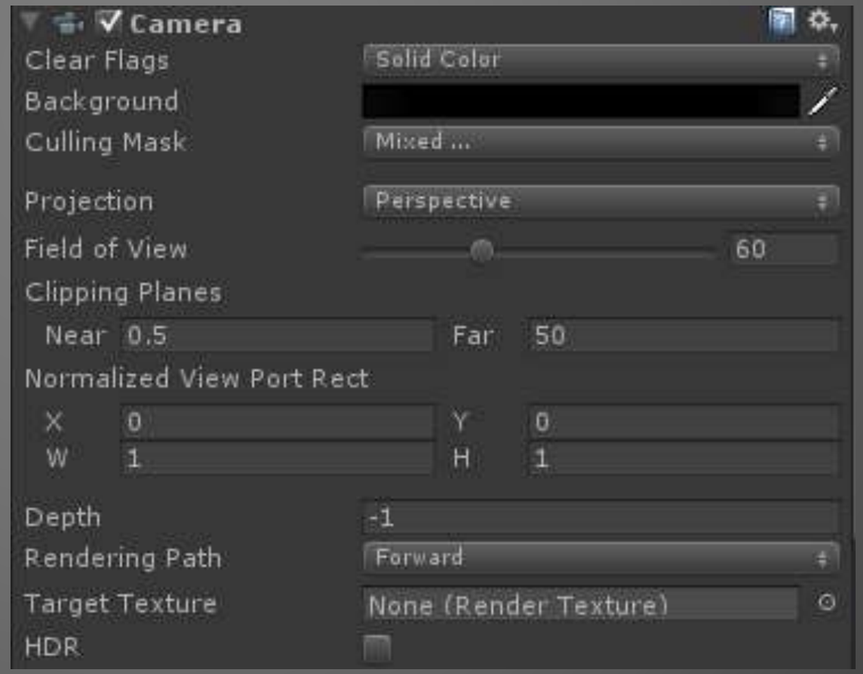
- 光源设置
  - 控制 “Important”光源个数
    - 建议1个或0个，一般为方向光
    - 个数越多，drawcall越多
  - Pixel Light数目
    - 1-2个



Unity引擎相关优化

- 相机设置
  - 设置合理的远裁剪平面
    - 默认为1000，建议根据各自不同的游戏场景进行修改





unity

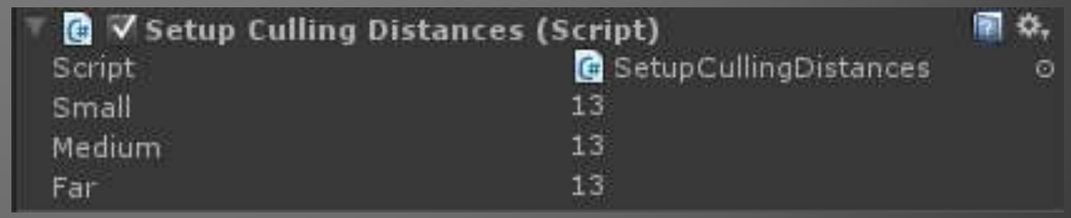
Unity引擎相关优化

- 相机设置
  - 设置合理的远裁剪平面
    - 默认为1000，建议根据各自不同的游戏场景进行修改
  - 设置分层的远裁剪平面
    - 根据物体的体积大小、渲染成本来进行分层(camera.layerCullSpherical)

```
using UnityEngine;
using System.Collections;

public class SetupCullingDistances : MonoBehaviour
{
    public float small = 13;
    public float medium = 13;
    public float far = 13;

    void Start ()
    {
        float[] distances = new float[32];
        distances[0] = small;
        distances[1] = medium;
        distances[2] = far;
        camera.layerCullDistances = distances;
    }
}
```

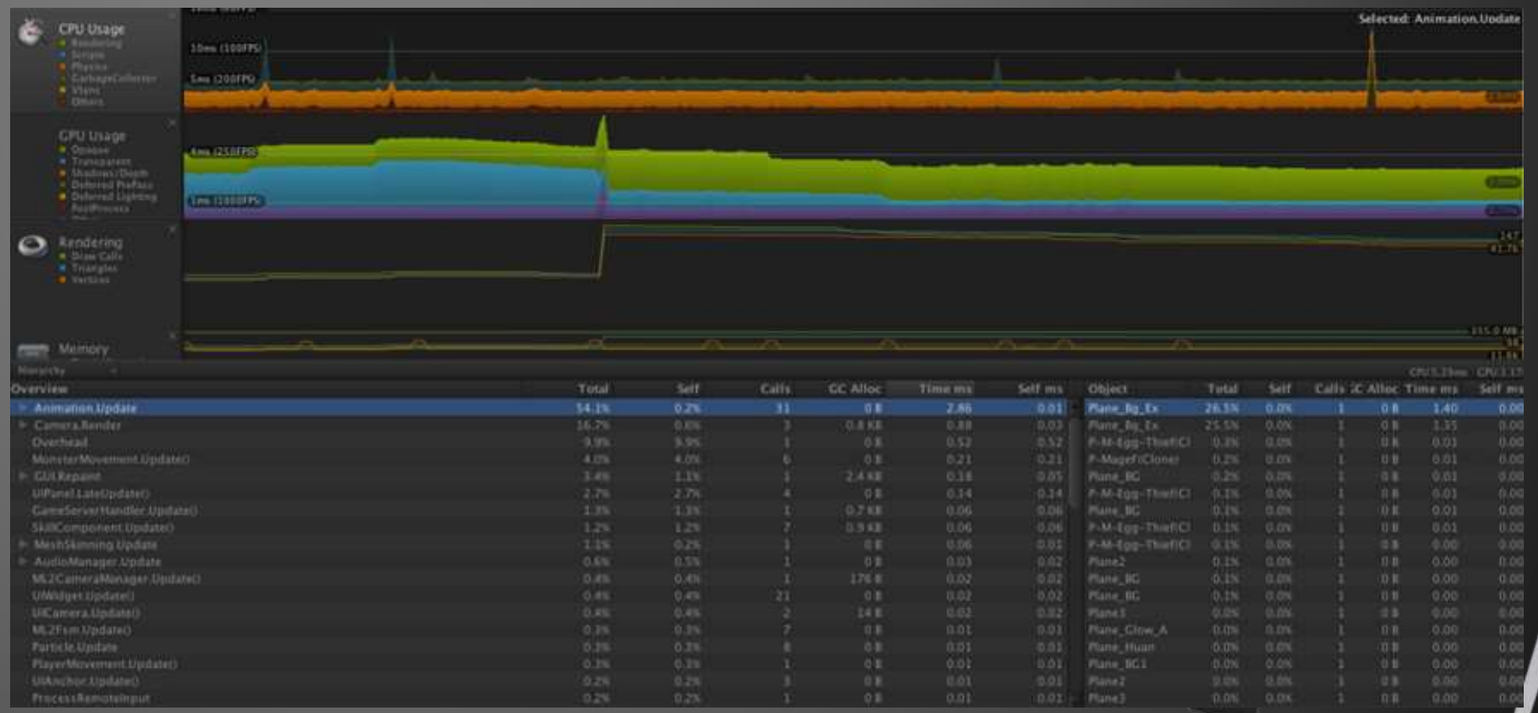


Unity引擎相关优化

- 粒子特效
  - 屏幕上的最大粒子数
    - 建议小于200个粒子
  - 每个粒子发射器发射的最大粒子数
    - 建议不超过50个
  - 粒子大小
    - 如果可以的话，粒子的size应该尽可能地小
    - 对于非常小的粒子，建议粒子纹理去掉alpha通道
  - 尽量不要开启粒子的碰撞功能

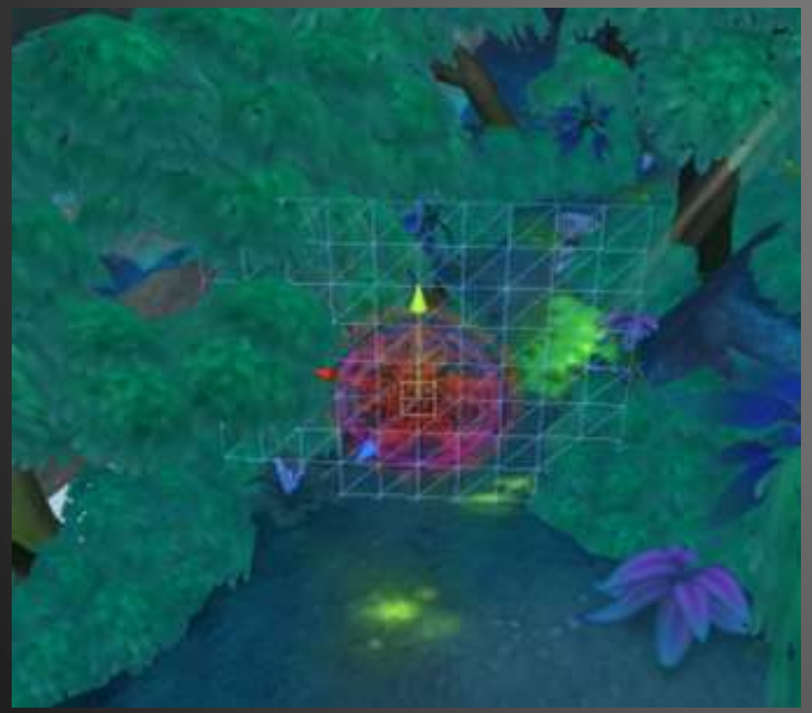
Unity引擎相关优化

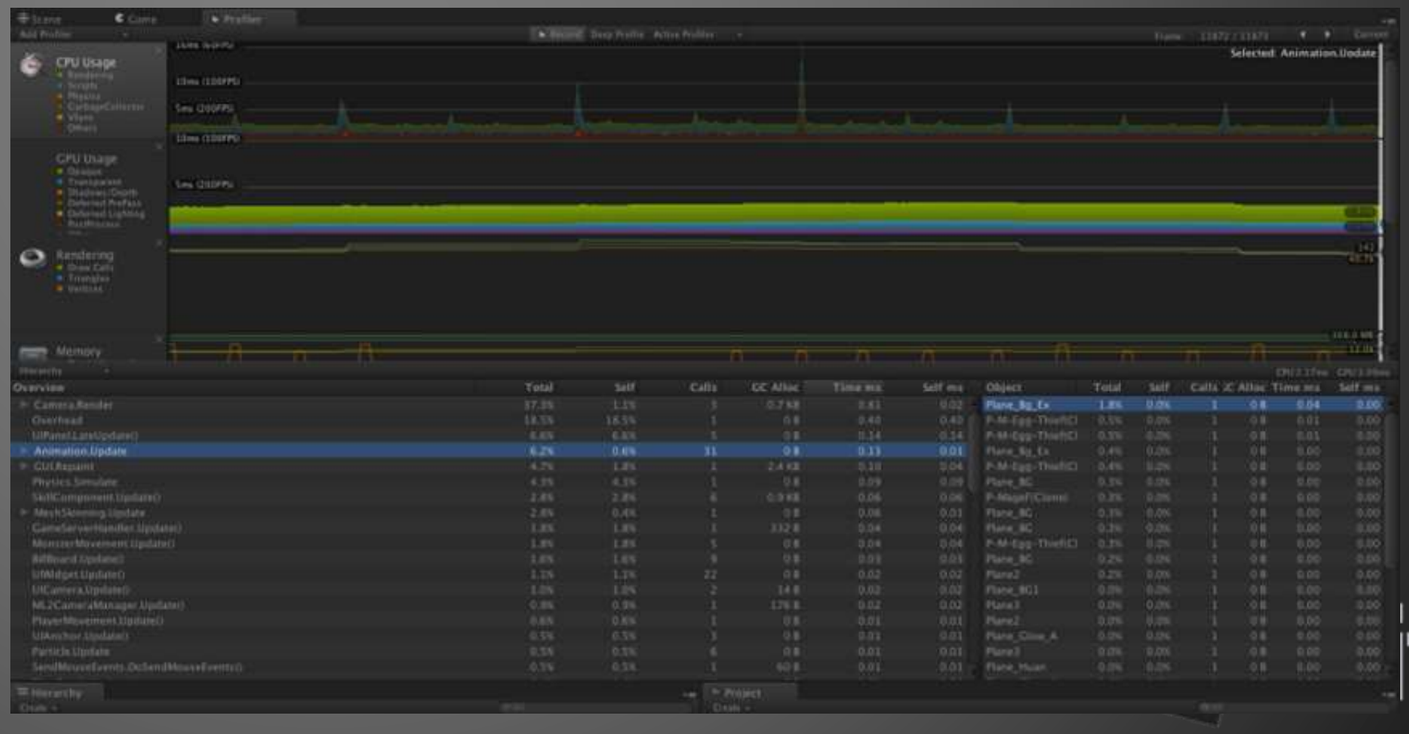
- 物理系统
  - 碰撞体控制
    - 如果可以的话，尽可能地使用Sphere Collider、Box Collider
    - 尽量避免使用Mesh Collider等



Unity引擎相关优化

- 物理系统
  - 碰撞体控制
    - 如果可以的话，尽可能地使用Sphere Collider、Box Collider
    - 尽量避免使用Mesh Collider等





Unity引擎相关优化

- 渲染设置
  - 尽可能地避免Alpha Test和Alpha Blend
    - 非常耗时，性价比很低！
    - 手工变为较复杂的多边形，以减小透明区域(甚至转为不透明)。
    - 尽可能用 Alpha blend 替换 Alpha Test。
    - 可将需要 Alpha Test 的面片手工分为两部分，内部作为不透明绘制（diffuse），边缘部分才用Alpha Test 或 alpha Blend。
    - 如果必须使用 Alpha Test，尽可能使得这部分面片始终只占屏幕上较小的区域(像素)。

Unity引擎相关优化

- 渲染设置
  - DrawCall Batching

    Unity在运行时可以将一些物体进行合并，从而用一个Draw Call来渲染它们。我们将这一操作称之为“DrawCall Batching”。

    一般来说，Unity batch的物体越多，你就会得到越好的渲染性能。

Unity引擎相关优化

- 渲染设置
  - DrawCall Batching
    - Static Batching
      - 针对材质相同的静态物体进行batch
      - 对几何数据的大小没有限制
    - 原理
      - 静态VertexBuffer + 动态IndexBuffer
      - 在Build过程中，同种材质的物体被合并在一个大的VertexBuffer中
      - 在Run-time时，通过视域体裁剪来动态改变IndexBuffer
    - 注意
      - 使用Static Batching需要额外的内存开销来储存batch后的几何数据

Unity引擎相关优化

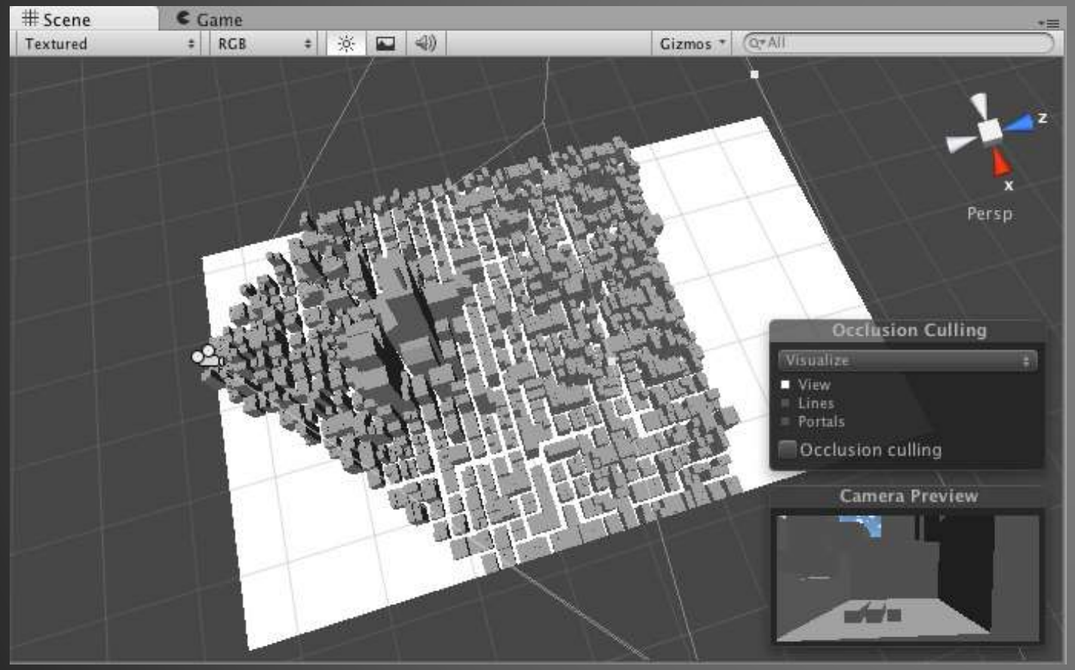
- 渲染设置
  - DrawCall Batching
    - Dynamic Batching
      - 相同材质的动态物体，Unity会自动对其进行batch
      - 原理
        - 动态VertexBuffer + 动态IndexBuffer
      - 注意
        - 目前仅支持小于900顶点的网格物体
        - 如果shader里使用Position，Normal和UV三种属性，那么你只能batch 300顶点以下的物体；如果使用Position，Normal，UV0，UV1和Tangent，那你只能batch 180顶点以下的物体
        - 进行缩放物体无法与非缩放物体进行batch
        - 合并总数有上限

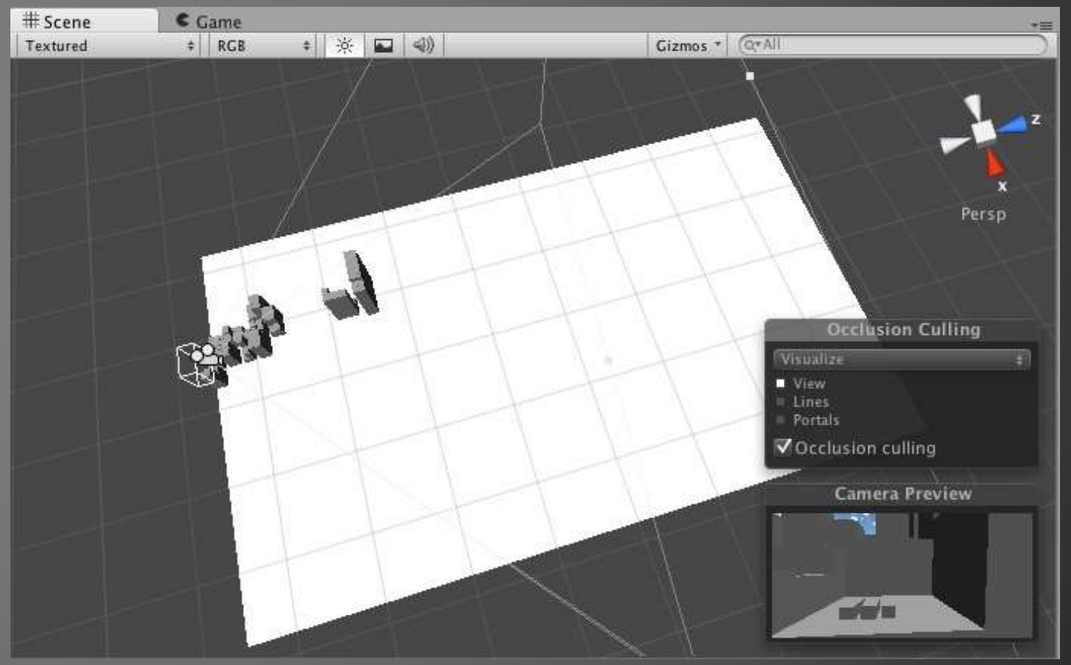
Unity引擎相关优化

- 渲染设置
  - 纹理拼合（Texture Packing）

Unity引擎相关优化

- 渲染设置
  - 遮挡剔除（Occlusion Culling）





Unity引擎相关优化

- 渲染设置
  - 遮挡剔除（Occlusion Culling）
    - 区分Occluder（遮挡者）和Occludee（被遮挡者）
    - 半透明物体一般不设定为Occluder
  - 选择合适的遮挡技术（Legacy）
    - PVS Only

      动态物体仅通过视域体进行裁剪，适合动态物体很少的情况
    - PVS and dynamic objects

      动态物体通过portal culling进行裁剪
    - Automatic Portal Generation

      动态物体通过portal culling进行裁剪，同时场景支持动态portal

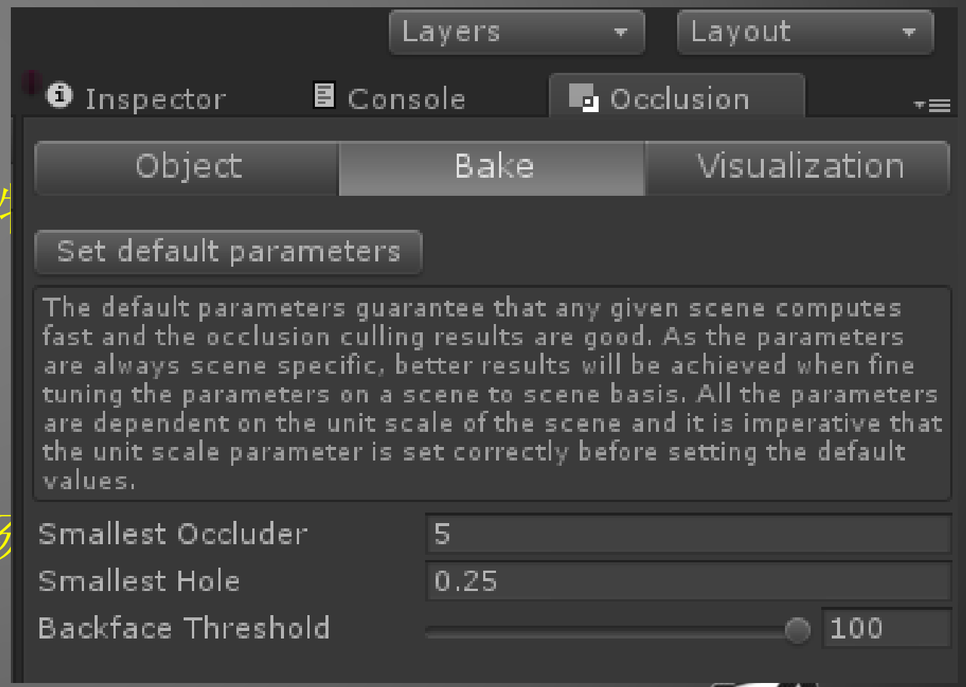
Unity引擎相关优化

- 渲染设置
  - 遮挡剔除（Occlusion Culling）
    - 区分Occluder（遮挡者）和Occludee（被遮挡者）
    - 半透明物体一般不设定为Occluder
  - 选择合适的遮挡技术（Legacy）
    - PVS Only

      动态物体仅通过视域体进行裁剪，适合动态
    - PVS and dynamic objects

      动态物体通过portal culling进行裁剪
    - Automatic Portal Generation

      动态物体通过portal culling进行裁剪，同时场



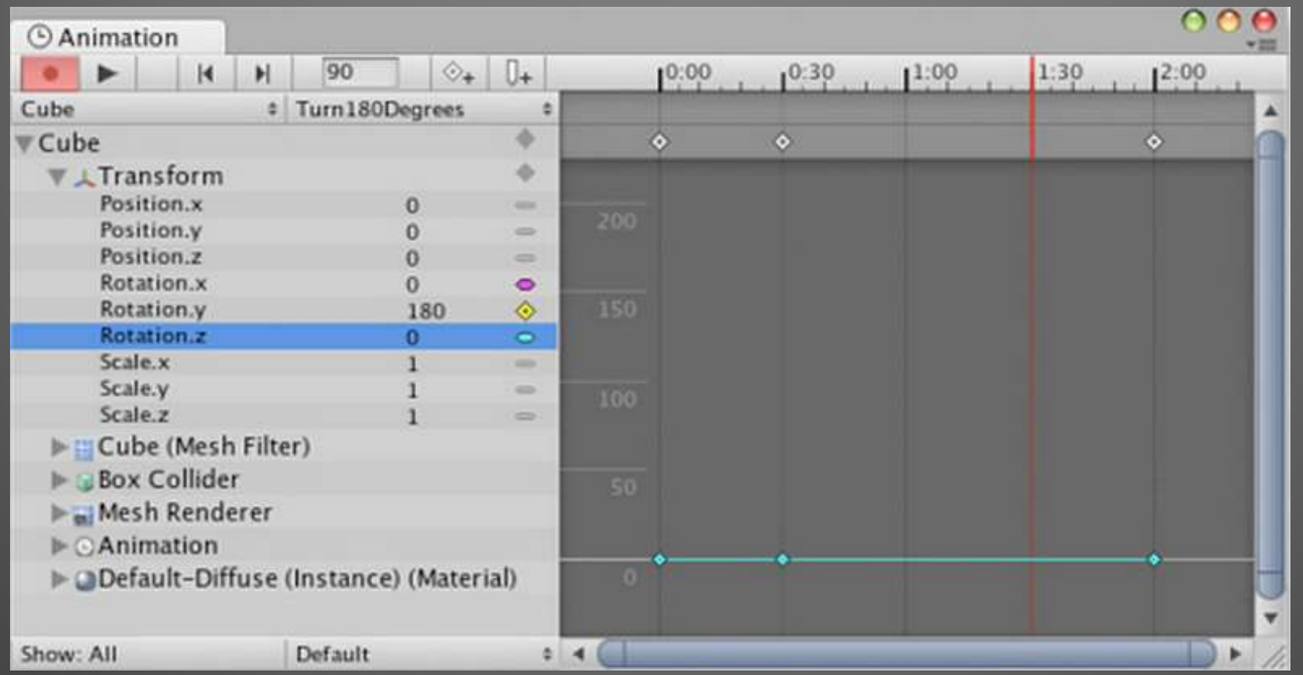
unity

Unity引擎相关优化

- 动画系统
  - 没有骨骼动画的模型要除去Animation组件

Unity引擎相关优化

- 动画系统
  - 没有骨骼动画的模型要除去Animation组件
  - 如果Animation不进行缩放，去除Scale Curves
    - 减少33%的Blending时间



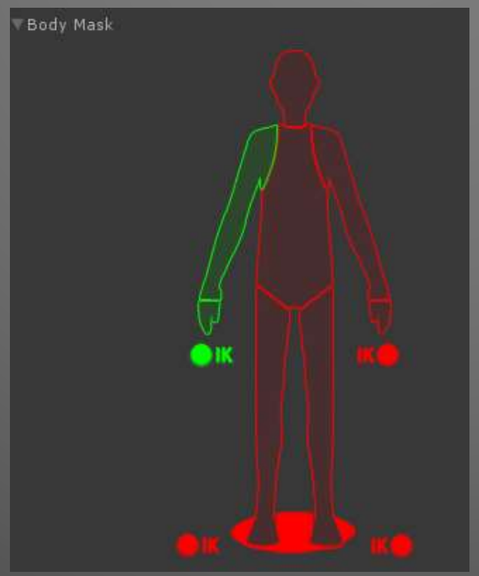
unity

Unity引擎相关优化

- 动画系统
  - 没有骨骼动画的模型要除去Animation组件
  - 如果Animation不进行缩放，去除Scale Curves
    - 减少33%的Blending时间
  - 只有一个AnimationClip，使用老版本的Animation系统(Legacy)

Unity引擎相关优化

- 动画系统
  - 没有骨骼动画的模型要除去Animation组件
  - 如果Animation不进行缩放，去除Scale Curves
    - 减少33%的Blending时间
  - 只有一个AnimationClip，使用老版本的Animation系统
  - 使用Body Mask来减少不必要的计算量



unity

Unity引擎相关优化

- 动画系统
  - 没有骨骼动画的模型要除去Animation组件
  - 如果Animation不进行缩放，去除Scale Curves
    - 减少33%的Blending时间
  - 只有一个AnimationClip，使用老版本的Animation系统
  - 使用Body Mask来减少不必要的计算量
  - 注意Mecanim系统在Android上运行效率
    - 需要支持NEON的Android

代码相关优化

- 使用静态类型

```
function Start () {                          // 声明Start方法
    var test = GetComponent(Test);           // 调用Test脚本
    test.DoSomething();                      // 执行Test脚本中的DoSomething方法
}
```

```
function Start () {                           // 声明Start方法
    var test: Test = GetComponent(Test);      // 调用Test脚本
    test.DoSomething();                       // 执行Test脚本中的DoSomething方法
}
```

- 使用#pragma strict

  禁用JavaScript的动态类型

代码相关优化

- 缓存组件查询

```
private var myTransform : Transform;

function Awake () {
	myTransform = transform;
}

function Update () {
	myTransform.Translate (0, 0, 2);
}
```

代码相关优化

- 减少Update函数中所做的事情
- 减少Fixed Delta Time
  - 将Fixed Delta Time设定在0.04-0.067区间（即每秒15-25帧）
  - 降低FixedUpdate函数的调用频率
  - 降低物理引擎执行碰撞检测与刚体更新的频率
- 减少不必要的内存分配
- 使用iOS脚本调用优化功能
  - Slow and Safe
  - Fast and Exceptions Unsupported

代码相关优化

- 减少代码中的string连接

```
int score;
void Update()
{
    string scoreText = "Score: " + score.ToString();   // bad
}
```

使用StringBuilder来进行代替

- 尽可能地使用 “for”循环来代替“foreach”循环

代码相关优化

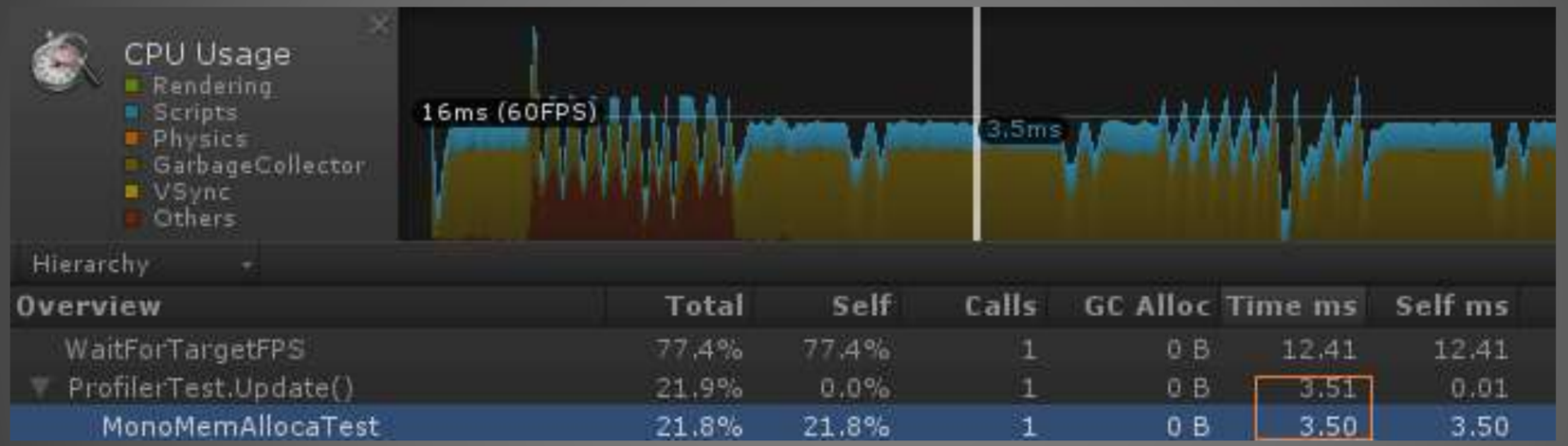
- 使用 struct 替换 class
  - Struct分配的内存在栈上，Class分配的内存在堆上。
  - 栈上的内存空间是连续的，且由操作系统负责分配，执行较快。
  - 堆上的内存由GC负责回收

```
public class ProfilerTest : MonoBehaviour {
    struct TestClass
    {
        public int dataValue;
    }
    void Update ()
    {
        MonoMemAllocaTest();
    }
    void MonoMemAllocaTest()
    {
        Profiler.BeginSample("MonoMemAllocaTest", gameObject);
            for (int i = 0; i < 1024 * 1024; i++)
            {
                TestClass test = new TestClass();
                test.dataValue = 10;
            }
        Profiler.EndSample();
    }
}
```

```
public class ProfilerTest : MonoBehaviour {
    class TestClass
    {
        public int dataValue;
    }
    void Update ()
    {
        MonoMemAllocaTest();
    }
    void MonoMemAllocaTest()
    {
        Profiler.BeginSample("MonoMemAllocaTest", gameObject);
            for (int i = 0; i < 1024 * 1024; i++)
            {
                TestClass test = new TestClass();
                test.dataValue = 10;
            }
        Profiler.EndSample();
    }
}
```

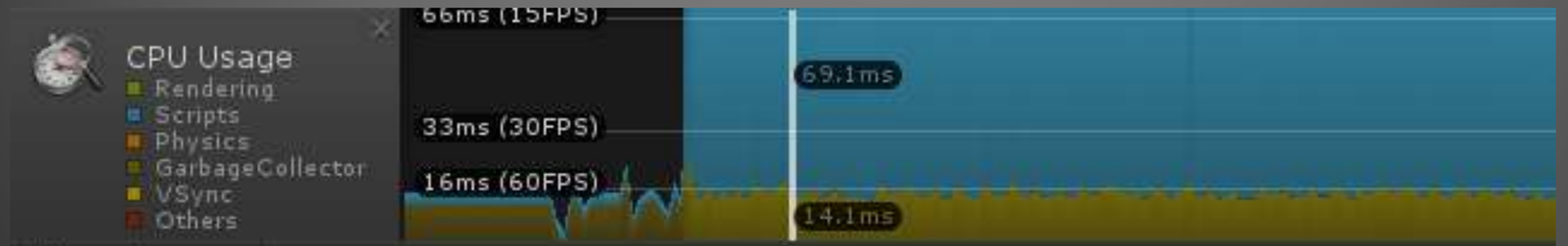
代码相关优化

- 使用 struct 替换 class
  - Struct



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
WaitForTargetFPS	77.4%	77.4%	1	0 B	12.41	12.41
ProfilerTest.Update()	21.9%	0.0%	1	0 B	3.51	0.01
MonoMemAllocaTest	21.8%	21.8%	1	0 B	3.50	3.50

  - Class



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
ProfilerTest.Update()	99.5%	0.0%	1	12.0 MB	83.23	0.00
MonoMemAllocaTest	99.5%	82.6%	1	12.0 MB	83.23	69.11
GC.Collect	16.8%	16.8%	1	0 B	14.11	14.11

代码相关优化

- 使用资源缓存池（Object Pool）
  - 大量减少Instantiate和Destroy的频率
  - 减少GC的频率
  - 游戏运行更加流畅

着色器相关优化

- 减少复杂的数学计算
  - 一些数学函数（如pow、exp、log、cos、sin、tan等）将增大GPU计算的负担，减少其使用次数，可用查找纹理来代替
- 不要使用discard操作
  - 效率低下
- 考虑浮点数计算
  - float/highp – 全32位浮点型，适合顶点变换，效率最低
  - half/mediump – 缩减的16位浮点型，适合于纹理UV坐标
  - fixed/lowp – 10位定点型，适合于颜色，光照计算
- PowerVR特点
  - 使用两个vec2变量，而不是将其拼合成一个vec4

如何能做出良好的优化？

如何能做出良好的优化？

- 多用Profiler

如何能做出良好的优化？

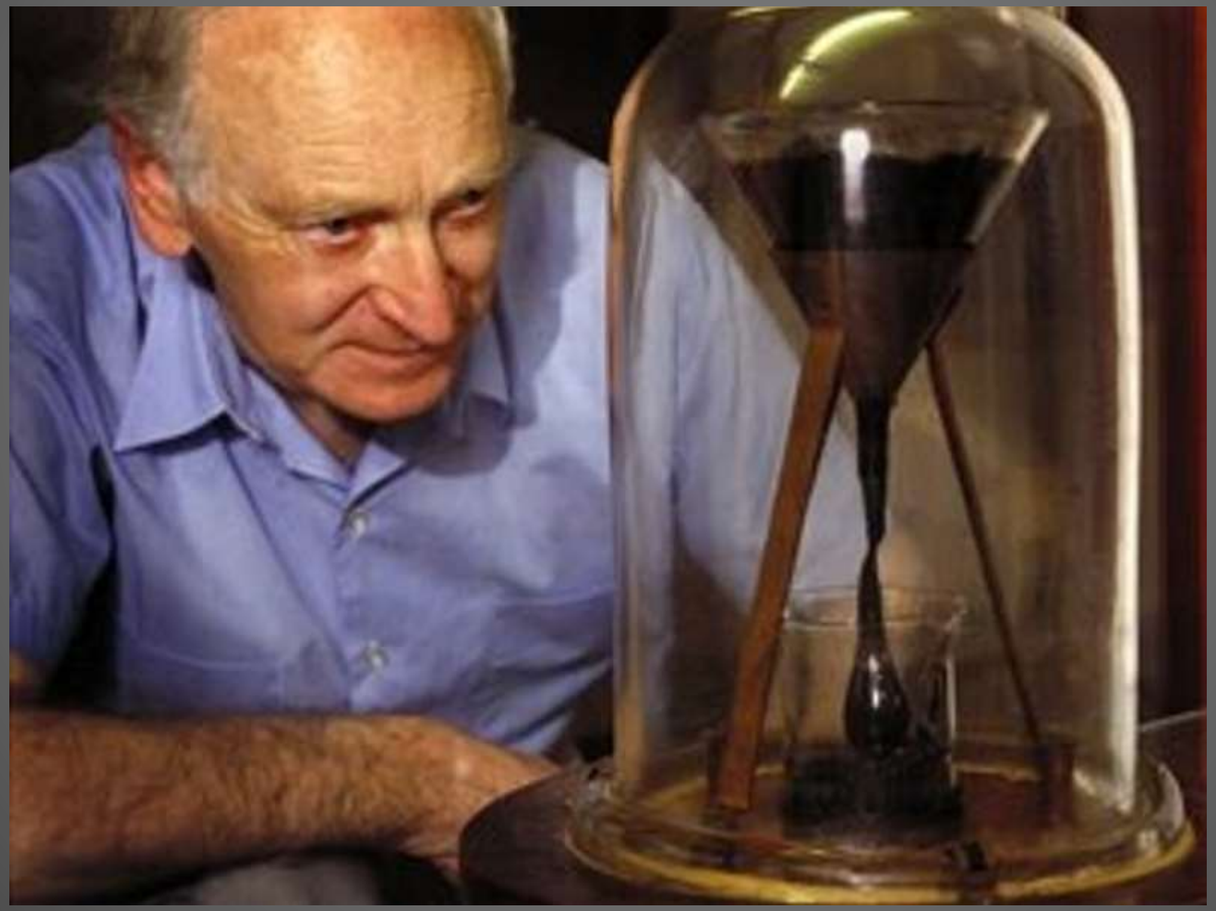
- 良好的规划
  优化做的好，不如规划做的好！！

GameResourceSetting.xlsx - Microsoft Excel

	A	B	C	D
1	个体角度			
2	检查项目	编号	检查参数	建议系数
3	单位	A0	模型单位是否规范(scale factor)	1
4	物理	B0	骨骼命名是否规范	0
5		B1	如果是MeshColider，检测是否有代理体（面片数应尽量少）	100
6	角色	C0	每个角色Skinned Mesh Renderer数量	1
7		C1	蒙皮网格（Skinned Mesh）总数量	7
8		C2	Material数量	3
9		C3	骨骼数量	30
10		C4	面片数量	1500
11		C5	是否含有IK结点	0
12		C6	Animation数量	5
13	静态实体	D0	检测Animation Component	0
14		D1	单个Mesh的顶点数	500
15		D2	UV值是否超出范围	0
16		D3	列出所有非静态物体	0
17	地形	E0	地形的分辨率大小	257
18		E1	地形中混合纹理的数量	4
19	纹理	F0	纹理格式	png tga
20		F1	纹理尺寸	1024
21		F2	是否含有Mipmap	1
22		F3	纹理的名字是否有重名	0
23		F4	检测alpha值是否有用	0
24	光源	G0	查看光源“Important”个数。	1
25		G1	Pixel Light数目	1
26	粒子特效	H0	粒子发射器数量	10
27		H1	每个粒子发射器发射的最大粒子数	20
28		H2	所使用的贴图大小	64
29	模型特效	I0	绘制Mesh（Batch）的数量	5
30		I1	每个Mesh的三角面片数	30
31		I2	使用的材质种类数目	3
32		I3	所使用的贴图大小	64
33	音频	J0	使用.ogg格式的压缩音频格式	0
34		J1	对于比较短的声音特效（爆炸、开枪）时，尽量使用无损音频格式	.wav和.aif格式

如何能做出良好的优化？

- 不断地试验、不断地总结



unity

谢谢

http://china.unity3d.com