【Unity基础教程】重点知识汇总

（十七）

Unity性能优化常用方法

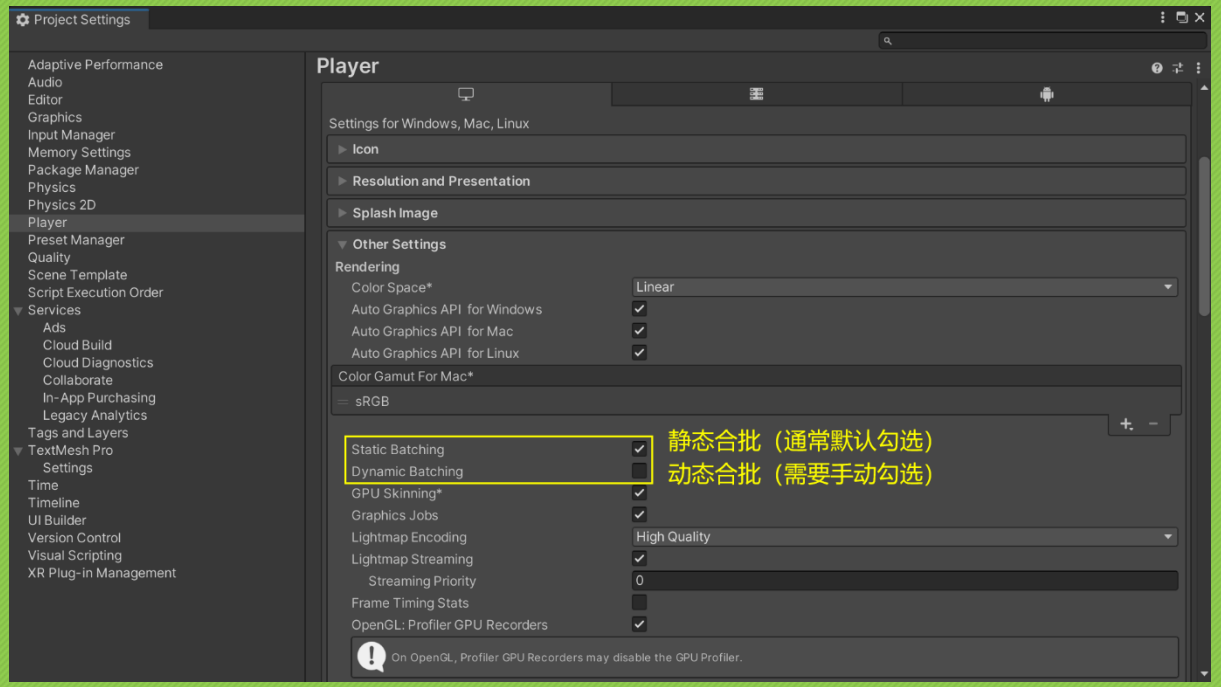
1.减少Draw Call（1）

概念：Draw Call是指**CPU给GPU**发出的**绘制命令**。每渲染一个材质、网格或对象都会产生一个Draw Call。而过多的Draw Call就会影响性能（**性能瓶颈**）。

优化方法：

- 使用**静态合批**（Static Batching）：适用于**静态**对象（不会移动、旋转或缩放的对象）。将对象设置为静态后，Unity会在渲染时将它们合并为一个（合并网格）。
- 使用**动态合批**（Dynamic Batching）：适用于**移动**的、**共享相同材质**的小型对象（顶点数需少于300）。需要在编辑器Player Settings中启用Dynamic Batching。
- *使用**GPU Instancing**：适用于**大量重复**对象（例如树、草等）。

1.减少Draw Call（2）




```
1 public MeshRenderer myRenderer;
2 void Start()
3 {
4     // 启用 GPU Instancing 的材质
5     Material material = myRenderer.material;
6     material.enableInstancing = true;
7 }
```

Unity > Edit > Project Settings > Player > Other Settings勾选静态及动态合批

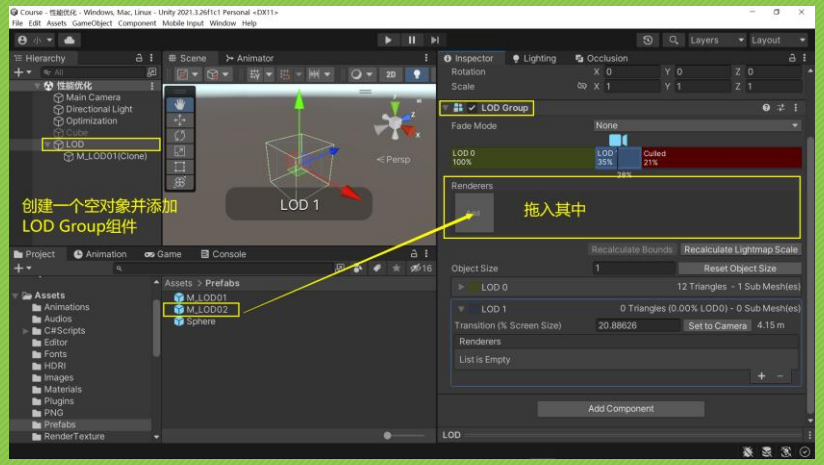
2.使用对象池

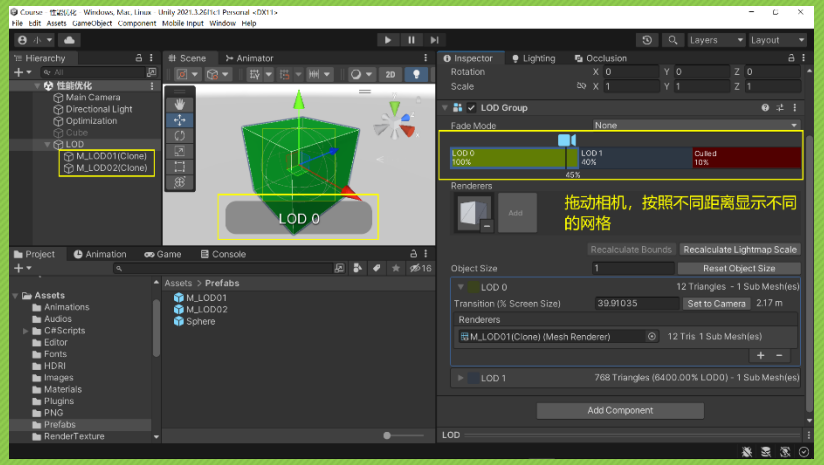
概念：频繁创建和销毁对象会导致垃圾回收GC（Garbage Collection）开销，降低性能。对象池通过**复用对象**减少GC开销，即减少内存分配和垃圾回收的频率（尤其是在大量重复对象（如子弹、敌人）中使用）。

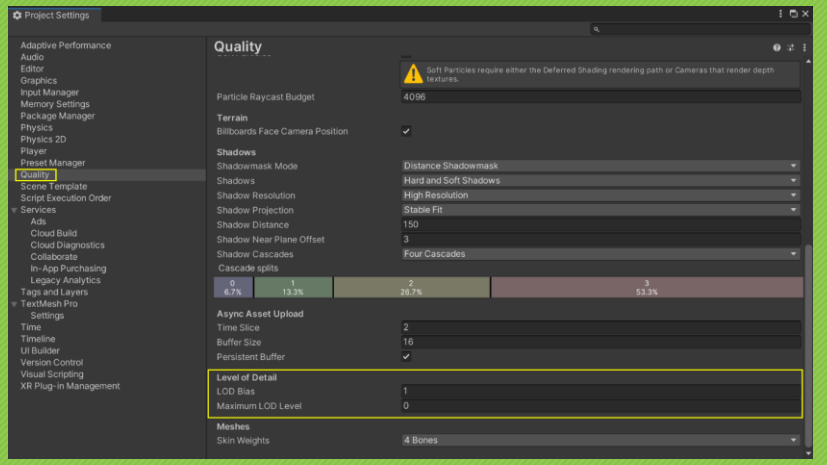
具体实现：参考【Unity基础教程】重点知识汇总（十三）——Unity C#设计模式之对象池模式。

3. LOD（Level of Detail）

概念：LOD（细节层级）是通过显示不同的网格模型或材质，在**远距离降低模型复杂度**（近距离用高模，中距离用中模，远距离用低模）。



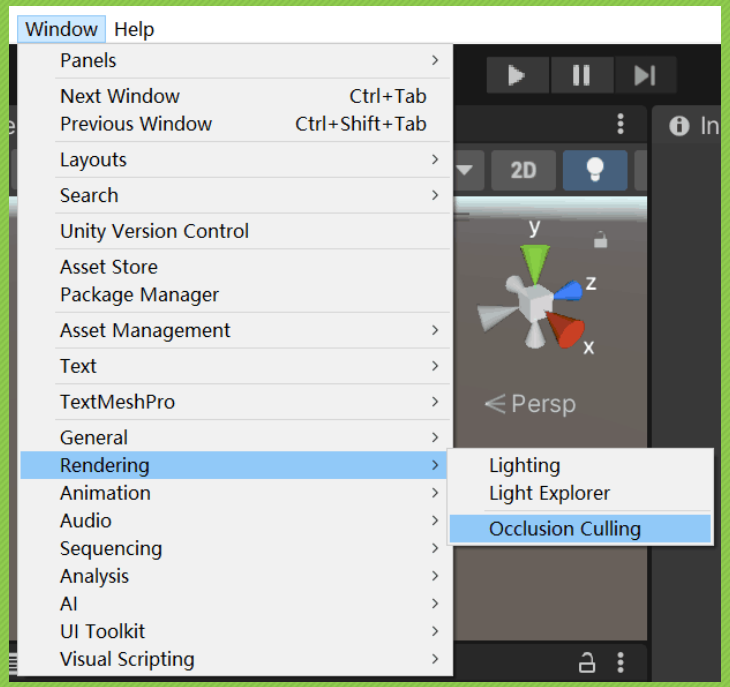


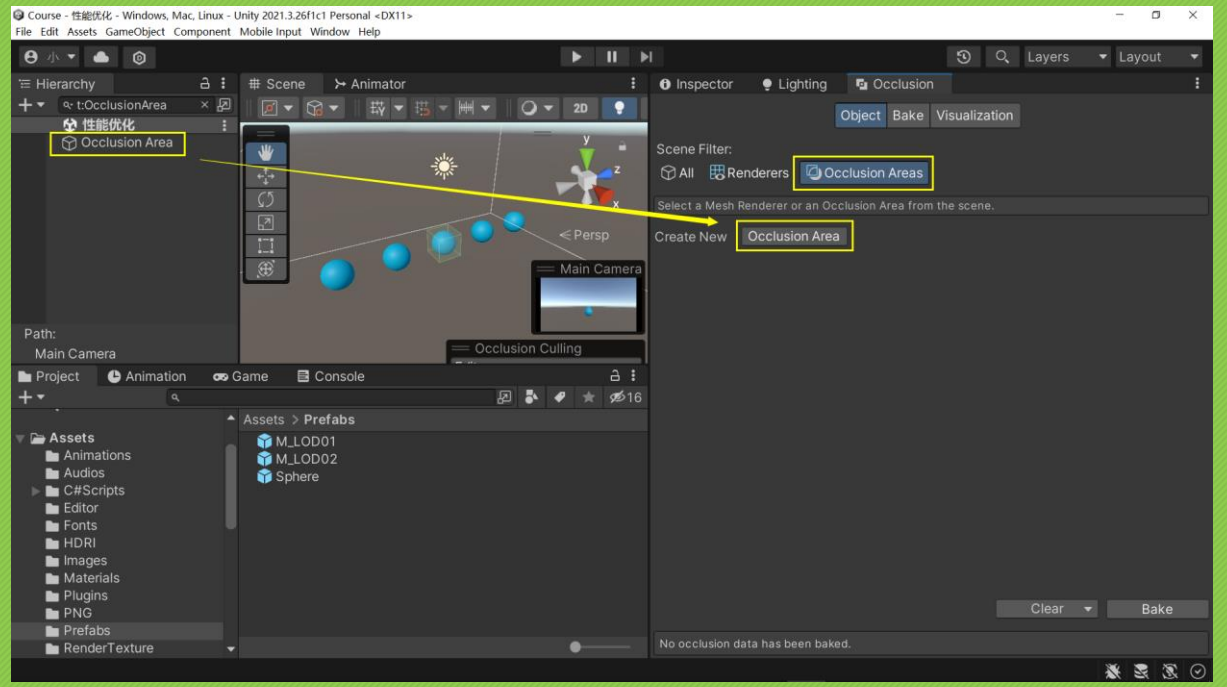


注意：不需要的LOD可以右键删除，层级越高，距离越近，模型精度越高。要注意合理使用。

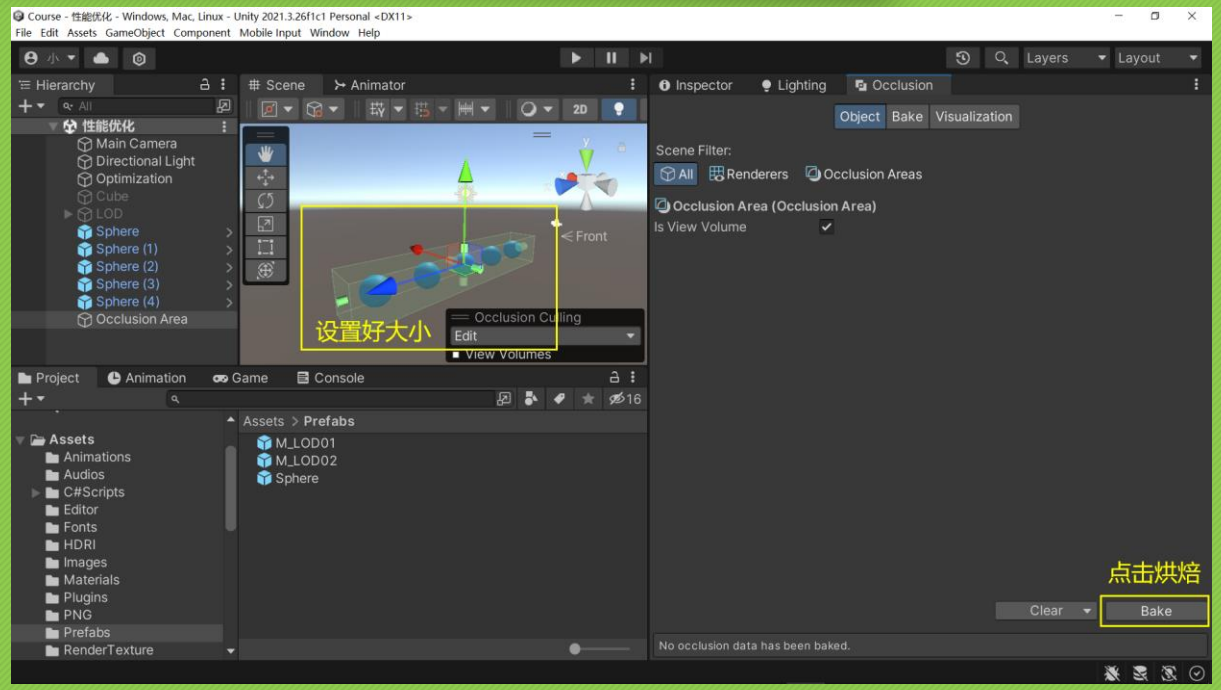
4. Culling（1）

概念：Culling（剔除）： 通过**剔除不可见对象**（如视锥剔除和遮挡剔除）减少渲染工作。通常会使用Unity中的**Occlusion Culling（遮挡剔除）**。

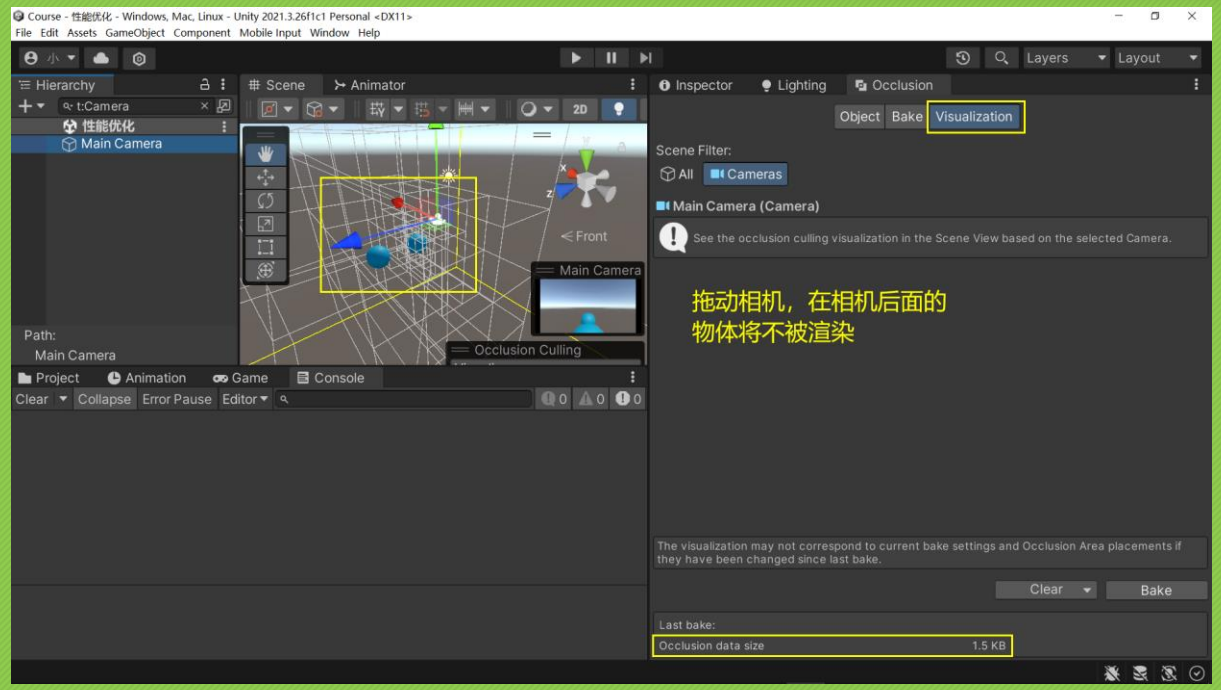


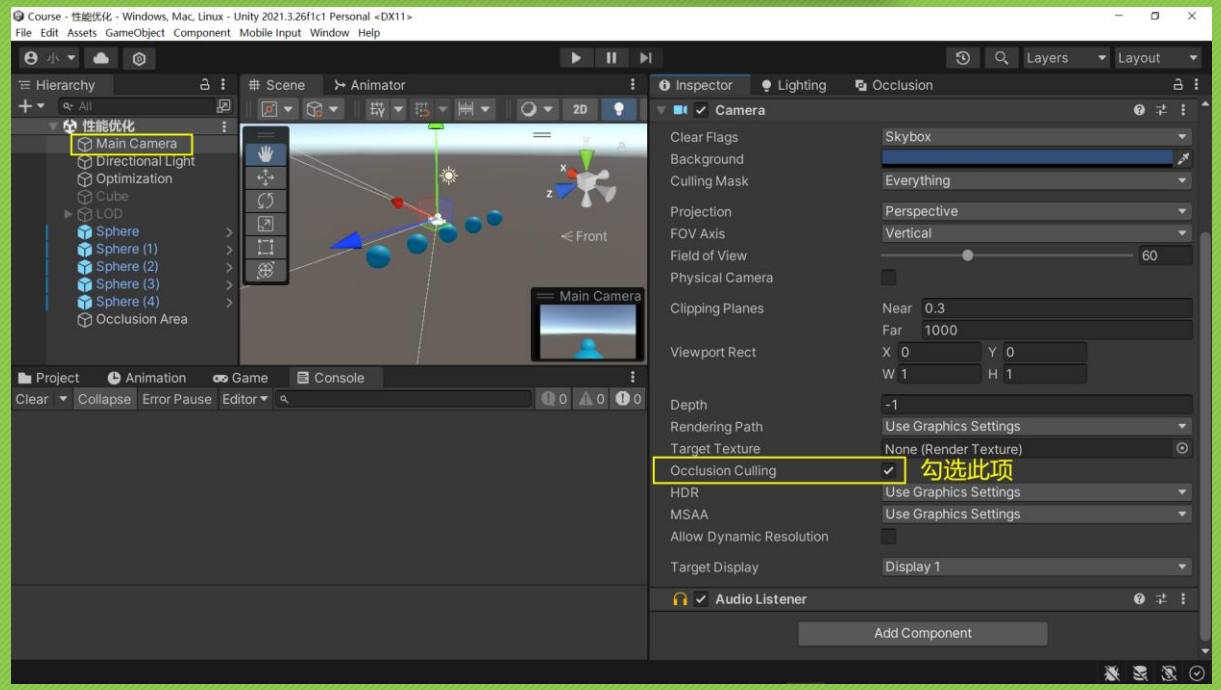


4. Culling（2）



4. Culling（3）


Course - 性能优化 - Windows, Mac, Linux - Unity 2021.3.26f1c1 Personal <DX11>
File Edit Assets GameObject Component Mobile Input Window Help
Hierarchy
t:Camera
性能优化
Main Camera
Scene
Animator
Inspector
Lighting
Occlusion
Object
Bake
Visualization
Scene Filter:
All
Cameras
Main Camera (Camera)
See the occlusion culling visualization in the Scene View based on the selected Camera.
拖动相机，在相机后面的
物体将不被渲染
Front
Main Camera
Occlusion Culling
Path:
Main Camera
Project
Animation
Game
Console
Clear
Collapse
Error Pause
Editor
The visualization may not correspond to current bake settings and Occlusion Area placements if they have been changed since last bake.
Clear
Bake
Last bake:
Occlusion data size
1.5 KB


Course - 性能优化 - Windows, Mac, Linux - Unity 2021.3.26f1c1 Personal <DX11>
File Edit Assets GameObject Component Mobile Input Window Help
Hierarchy
性能优化
Main Camera
Directional Light
Optimization
Cube
LOD
Sphere
Sphere (1)
Sphere (2)
Sphere (3)
Sphere (4)
Occlusion Area
Scene
Animator
Inspector
Lighting
Occlusion
Camera
Clear Flags
Skybox
Background
Culling Mask
Everything
Projection
Perspective
FOV Axis
Vertical
Field of View
60
Physical Camera
Clipping Planes
Near 0.3
Far 1000
Viewport Rect
X 0
Y 0
W 1
H 1
Depth
-1
Rendering Path
Use Graphics Settings
Target Texture
None (Render Texture)
Occlusion Culling
勾选此项
HDR
Use Graphics Settings
MSAA
Use Graphics Settings
Allow Dynamic Resolution
Target Display
Display 1
Audio Listener
Add Component
Front
Main Camera
Project
Animation
Game
Console
Clear
Collapse
Error Pause
Editor

5.减少物理运算的开销

概念：过多的物理计算（碰撞检测、刚体模拟等）会增加CPU负担。

优化方法：

- **优化碰撞器：** 使用简单的碰撞器（如Box Collider、Sphere Collider），避免使用复杂的Mesh Collider。
- **降低物理更新频率：** Unity的物理更新频率可通过脚本修改**Time.fixedDeltaTime**（默认是0.02，可以适当增加），也可通过调整Fixed Timestep（Edit > Project Settings > Time）。
- **禁用不必要的物理计算：**例如，使用Rigidbody.isKinematic（设置为true）避免模拟不需要的刚体。
- **调整物理层：**通过设置物理层来控制哪些物体之间可以发生碰撞。例如，射线检测中设置的LayerMask。

6.减少光照计算

概念：实时光照计算会显著增加性能开销，尤其是在移动设备上。

优化方法：

- 使用**烘焙**光照（Baked Lighting）。
- 将实时光源设置为**混合光源**（Mixed Lighting）。
- **减少实时阴影**数量（Unity支持的实时光源数量有限）。
- 避免使用**过多的Shader**特效，尽量选择简单的Shader。

具体实现：参考【Unity基础教程】重点知识汇总（十六）——Unity场景打光。

7.优化脚本和逻辑

概念：复杂的脚本逻辑和频繁的Update调用会导致CPU性能下降。

优化方法：

- 避免在**Update中**执行重复性计算。
- 使用**Coroutine**替代频繁调用的逻辑。
- **缓存**经常使用的组件和变量。

```
// 每秒更新一次
InvokeRepeating("UpdateScore", 0, 1.0f);
```

```
public class Optimization : MonoBehaviour
{
    public MeshRenderer myRenderer;
    // 缓存组件（避免每次都调用GetComponent）
    private Transform cachedTransform;
    void Start()
    {
        // 降低物理更新频率
        Time.fixedDeltaTime = 0.02f; // 默认是0.02，可以适当增加

        // 启用 GPU Instancing 的材质
        Material material = myRenderer.material;
        material.enableInstancing = true;

        cachedTransform = transform;
    }
    void Update()
    {
        cachedTransform.position += Vector3.forward * Time.deltaTime;
    }
}
```

8.压缩和优化资源（1）

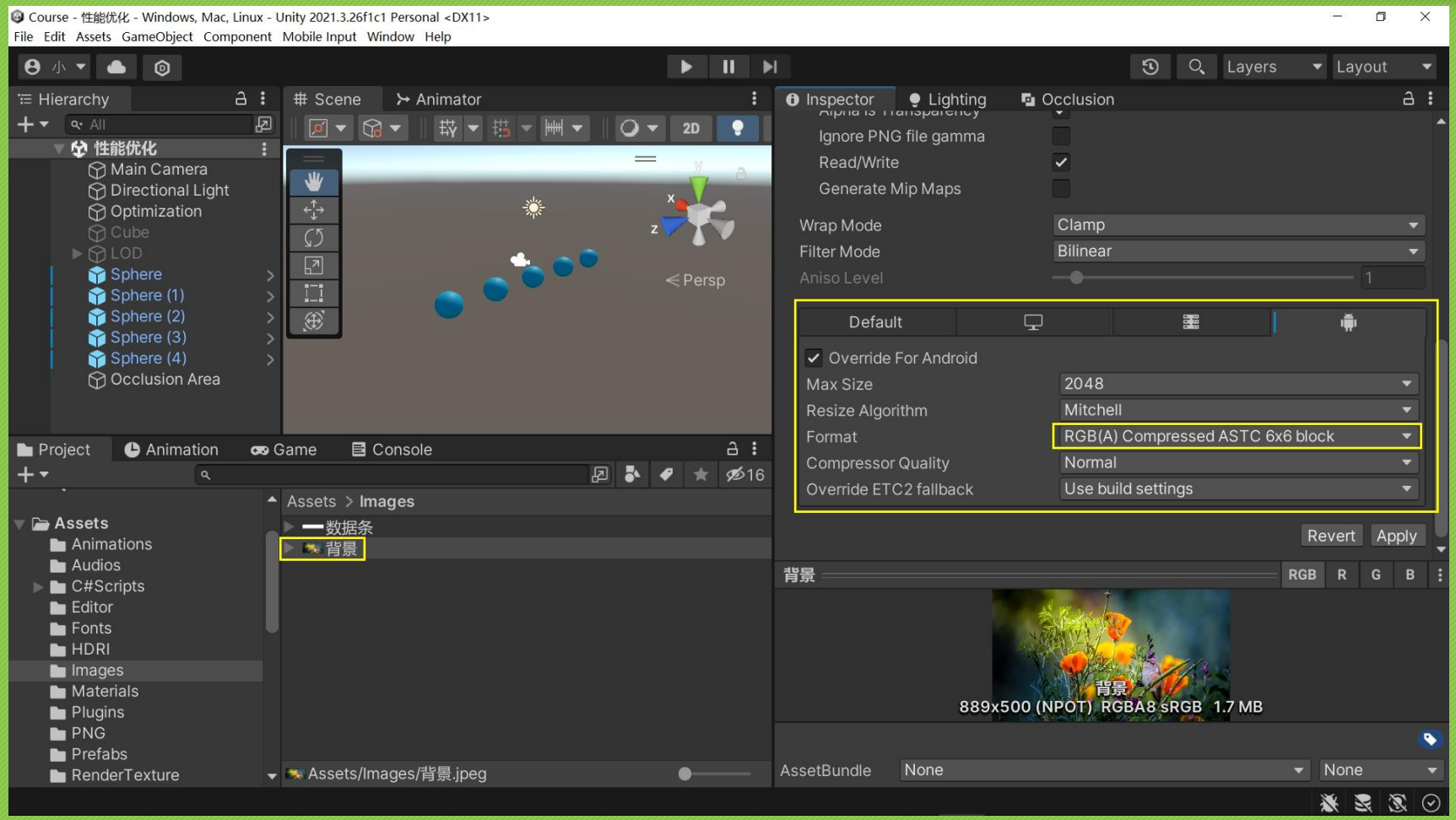
概念：高分辨率纹理和复杂材质会占用大量内存和GPU资源。

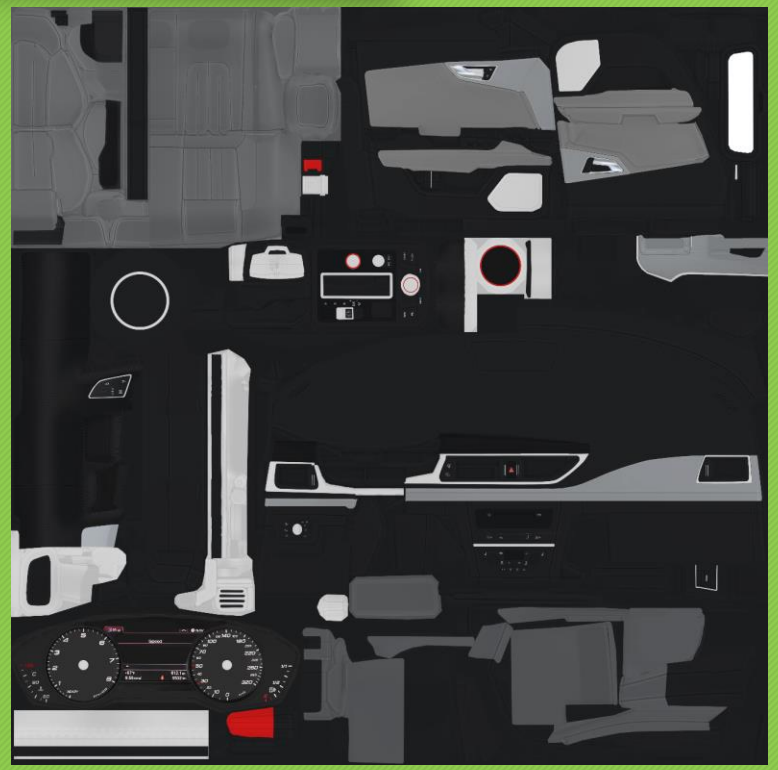
优化方法：

- 使用**压缩**格式（如ASTC、ETC2等，具体取决于目标平台）。
- **合并**材质（通过**纹理图集**减少材质数量）。
- 避免使用高开销的**着色器**（如实时反射）。
- 减少模型的**多边形数量**，删除无用的顶点和UV。
- 减少过度使用**粒子特效**（占用大量CPU和GPU资源）。
- 减少**动画的开销**。复杂的骨骼动画和Animator状态机会占用大量性能。

补充：纹理图集（Texture Atlas） 是将**多个小的纹理资源（如图片、图标、UI元素等）合并到一张大纹理中**的技术。这张大纹理称为纹理图集，通过一次加载大纹理，渲染多个图形元素，从而**减少材质切换和Draw Call**，提升渲染性能。

8.压缩和优化资源（2）





纹理图集（将多个小元素纹理合并在一张纹理上）

优化方法（总结）

- **静态合批（Static Batching）：**将静态物体标记为静态（Static），Unity会在构建时将这些对象合并为一个网格。
- **动态合批（Dynamic Batching）：**对顶点数较少（小于300个顶点）的动态物体进行合批，前提是它们使用同一个材质。
- **GPU Instancing（GPU实例化）：**使用相同材质和网格的重复对象，启用材质的Enable GPU Instancing选项。
- **对象池（Object Pooling）：**重复使用已实例化的对象而不是频繁创建和销毁，减少GC（垃圾回收）开销和内存碎片化问题。
- **LOD（细节层级，Level of Detail）：**根据对象与相机的距离动态切换不同精度的模型，降低远距离对象的渲染负担。
- **Occlusion Culling（遮挡剔除）：**利用Unity的遮挡剔除功能（如Occlusion Culling窗口），避免渲染被其他物体遮挡的对象。
- **减少物理运算开销：**优化物理碰撞的复杂度（如简化碰撞体形状），降低FixedUpdate的执行频率，限制Rigidbody的使用范围。
- **减少光照计算：**使用烘焙光照（Baked Lighting）代替实时光照。减少实时光源数量，利用探针优化动态物体的光照效果。
- **优化脚本逻辑：**避免频繁调用性能昂贵的函数（如Find、GetComponent等），用对象缓存代替。减少每帧的Update逻辑。
- **压缩优化资源：**减少贴图尺寸，使用压缩格式。优化模型的多边形数，去除不必要的顶点和UV通道。
- **粒子系统优化：**限制粒子数量，使用预渲染的粒子贴图。使用子发射器减少单个粒子系统的复杂度。
- **减少Draw Call（绘制调用）：**合并材质（如使用Texture Atlas纹理图集）以减少材质切换。

【Unity基础教程】重点知识汇总

（十七）

Unity性能优化常用方法