Unity杂项

（1）Unity5.x天空盒子

在现实生活中，我们可以看到各种各样的天空，在Unity3D游戏中也能看到各种各样的天空，那么这些天空是如何添加的呢，本文就以图文的形式详细的向大家介绍如何在Unity3D中添加天空盒子。

（1）导入天空盒子资源包

如果已经导入天空盒子资源包了，则可以省略此步骤。

鼠标右击Project视图的Assets文件夹，在弹出来的列表中点击“Import Package”—>"Skyboxes"

（2）弹出一个“Import packages”窗口，选择导入的天空盒子资源，这里一般默认就可以了，点击“Import”按钮。

（3）等待天空盒子导入完成，点击Project视图下的”Assets"-->“Skyboxes”-->"Textures"，可以看到系统提供了9款天空贴图资源，点击其中的一个文件夹，可以看到里面，放置了6个面的贴图材质，刚好对应前、后、左、右、上、下6个面。

（4）设置天空盒子有两种方法，一种是将天空盒子绑定到摄像机上，这样在摄像机的视野里看到的天空都是设置了我们的天空贴图的，但是如果切换摄像机就无法显示同一个天空了。第二种是在场景中添加天空盒子，这样避免了在多摄像机中切换摄像机所带来的天空显示不一致的问题。

方法一：将天空盒子绑定到摄像机上

首先，在Hierarchy视图中选中“Main Camera”。

然后，点击菜单栏上的“Component”按钮，在弹出来的列表中点击“Rendering”—>”Skybox“，如下图所示。

接着在Inspector视图中可以看到添加了Skybox组件，点击组件最右侧的带有点的小圆圈，如下图所示。

当点击带有点的小圆圈后会弹出一个“Select Material”窗口，可以看到里面有很多天空贴图，我们选择其中一个。

最后，我们点击运行按钮，在Game视图中我们可以看到游戏场景中有背景天空了，如图所示。

方法二：在场景中添加天空盒子

Unity3D 5 将场景中天空盒的设置移到了Lighting中

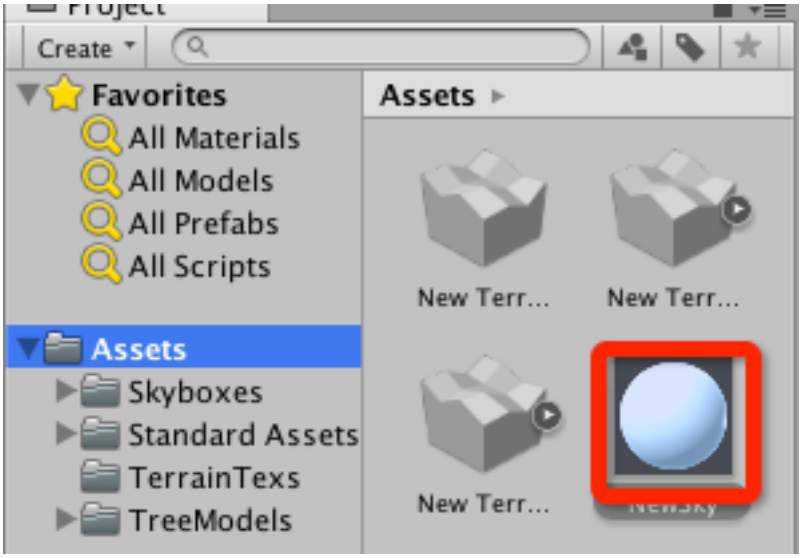
打开Unity3D，选择菜单栏的Window中的Lighting

在Lighting面板中点击Scene，就可以看到Skybox的选项，将自己的天空盒材质赋予即可。

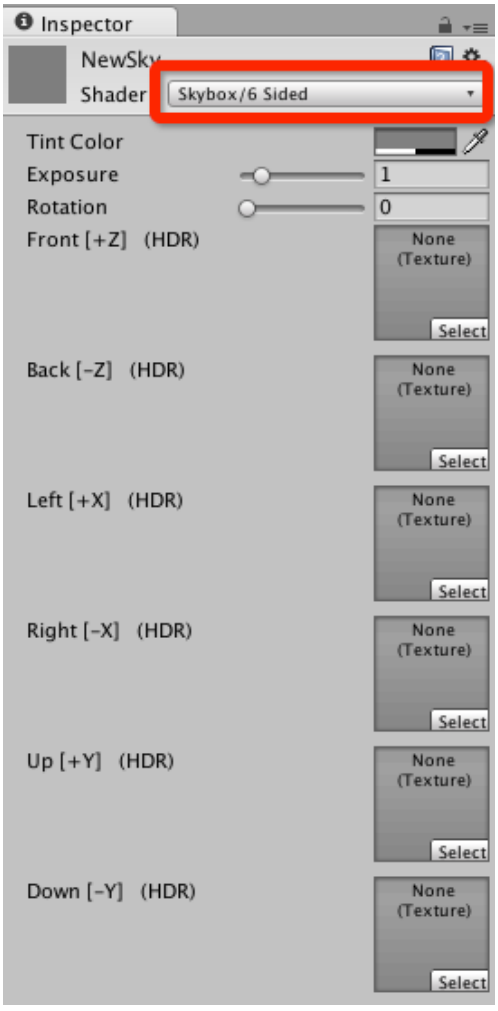
（2）如何制作自己的天空盒子

上面我们使用的天空盒子是直接从package中导入的，下面我们演示一下如何创建自己的天空盒子。

（1）新建一个材质，命名为SKY

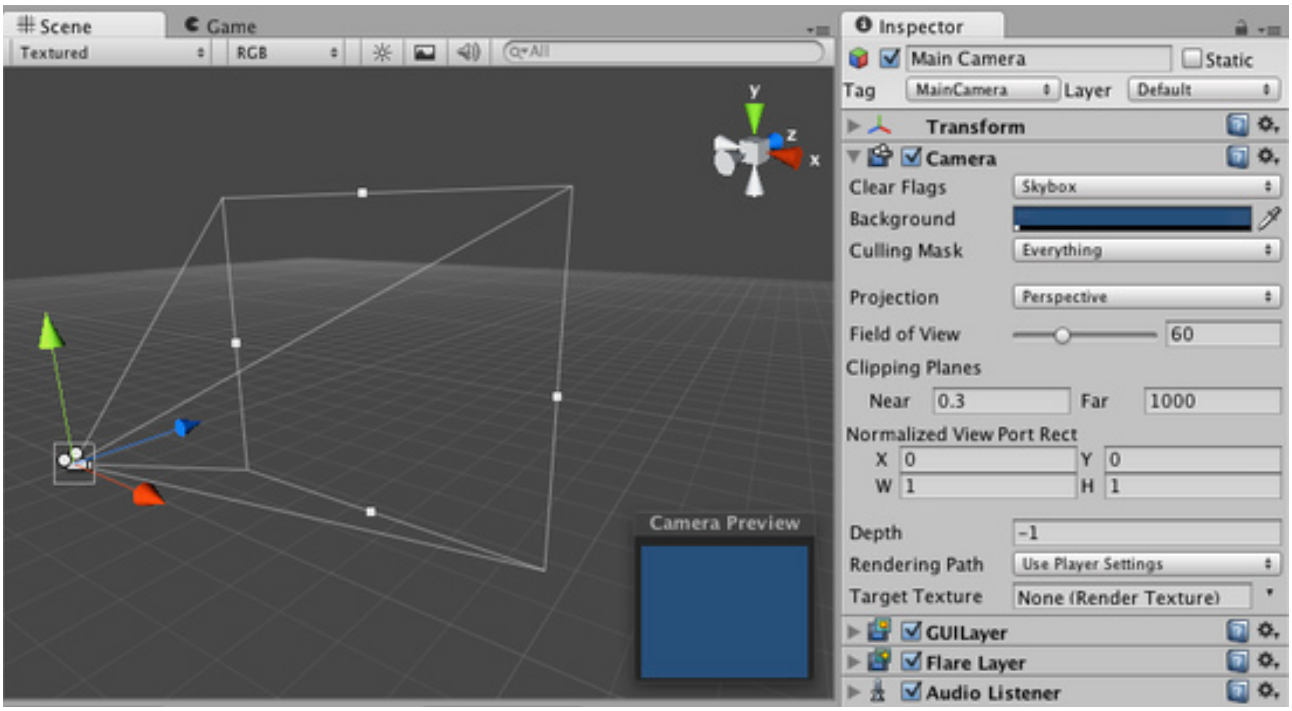


（2）选择材质，点击右侧的Shader选项，依次添加六张图片即可，也就是四方形的六个面。



（二）Unity Camera详解

正如电影中的镜头用来将故事呈现给观众一样，Unity的相机用来将游戏世界呈现给玩家。你始终至少有一个相机在场景中，你也可以有多个。多相机可以给你一个双人分屏效果或创建高级的自定义效果。你可以让相机动起来，或用物理（组件）控制它们。几乎你能想到的任何事，都可以用相机变成可能，而且为了适合你的游戏风格，你可以用典型的或特殊的相机类型。



相机是为玩家捕捉和显示世界的一种装置。通过定制和操作相机，可以让你的游戏外观与众不同。在一个场景中你可以有数量不限的相机。它们可以被设置为以任何顺序来渲染，在屏幕上的任何地方来渲染，或仅仅渲染屏幕的一部分。

相机参数详解

1. Clear Flags:清除标记。决定屏幕的哪部分将被清除。一般用户使用对台摄像机来描绘不同游戏对象的情况，有3中模式选择：

Skybox：天空盒。默认模式。在屏幕中的空白部分将显示当前摄像机的天空盒。如果当前摄像机没有设置天空盒，会默认用Background色。

Solid Color：纯色。选择该模式屏幕上的空白部分将显示当前摄像机的background色。

Depth only：仅深度。该模式用于游戏对象不希望被裁剪的情况。

Dont Clear：不清除。该模式不清除任何颜色或深度缓存。其结果是，每一帧渲染的结果叠加在下一帧之上。一般与自定义的shader配合使用。

2. Background: 背景。设置背景颜色。在镜头中的所有元素渲染完成且没有指定skybox的情况下，将设置的颜色应用到屏幕的空白处。

3. Culling Mask: 剔除遮罩，选择所要显示的layer。

4. Projection: 投射方式。
Perspective: 透视。摄像机将用透视的方式来渲染游戏对象。
Field of view: 视野范围。用于控制摄像机的视角宽度以及纵向的角度尺寸。
Orthographic: 正交。摄像机将用无透视的方式来渲染游戏对象。
Size: 大小。用于控制正交模式摄像机的视口大小。

5. Clipping Planes: 剪裁平面。摄像机开始渲染与停止渲染之间的距离。
Near: 近点。摄像机开始渲染的最近的点。
Far: 远点。摄像机开始渲染的最远的点。修改此数值，可以节省渲染资源。

6. Viewport Rect: 标准视图矩形。用四个数值来控制摄像机的视图将绘制在屏幕的位置和大小，使用的是屏幕坐标系，数值在0~1之间。坐标系原点在左下角。

7. Depth: 深度。用于控制摄像机的渲染顺序，较大值的摄像机将被渲染在较小值的摄像机之上。

8. Rendering Path: 渲染路径。用于指定摄像机的渲染方法。
Use Player Settings: 使用Project Settings-->Player中的设置。
Vertex Lit: 顶点光照。摄像机将对所有的游戏对象座位顶点光照对象来渲染。
Forward: 快速渲染。摄像机将所有游戏对象将按每种材质一个通道的方式来渲染。
Deferred Lighting: 延迟光照。摄像机先对所有游戏对象进行一次无光照渲染，用屏幕空间大小的Buffer保存几何体的深度、法线已经高光强度，生成的Buffer将用于计算光照，同时生成一张新的光照信息Buffer。最后所有的游戏对象会被再次渲染，渲染时叠加光照信息Buffer的内容。

9. Target Texture: 目标纹理。用于将摄像机视图输出并渲染到屏幕。一般用于制作导航图或者画中画等效果。

10:Occlusion Culling:遮挡剔除。
对于GPU来说，它负责整个渲染流水线。它会从处理CPU传递过来的模型数据开始，进行Vertex Shader、Fragment Shader等一系列工作，最后输出屏幕上的每个像素。因此它的性能瓶颈可能和需要处理的顶点数目的、屏幕分辨率、显存等因素有关。总体包含了顶点和像素两方面的性能瓶颈。在像素处理

中，最常见的性能瓶颈之一是overdraw。Overdraw指的是，我们可能对屏幕上的像素绘制了多次。

11. HDR：高动态光照渲染。用于启用摄像机的高动态范围渲染功能。

实战演示：

在当前场景添加一个新的相机，将其放置在右上角，大小为屏幕的三分之一，相当于小地图。

（三）Unity 光源

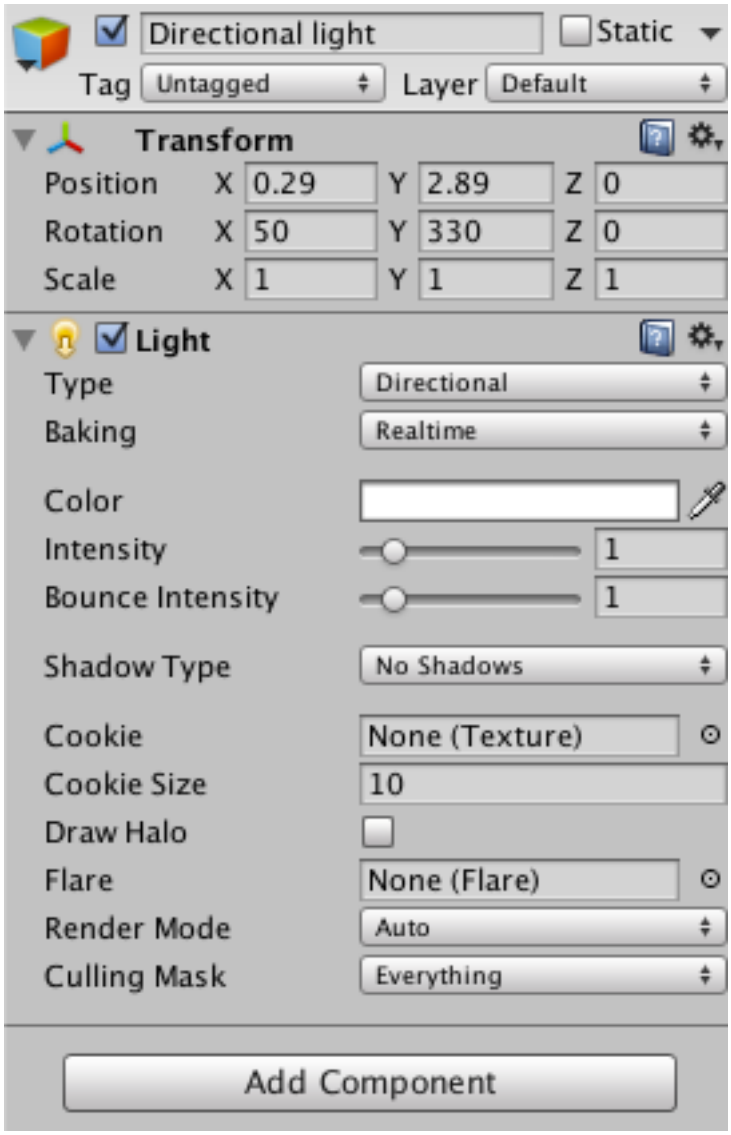
Unity中提供了四种光源：

Directional light：方向光，类似太阳的日照效果。直线光可以照射无限空间范围

Point light：点光源，类似蜡烛。可以向四面八方照射，通常用于爆炸，弥红灯

Spotlight：聚光灯，类似手电筒。在一个圆锥体范围内发射光线

Area Light：区域光，无法用作实时光照，一般用于光照贴图烘培。



Type：可以选择以上介绍的四种光源类型

Range：设置光源范围的大小，从光源对象的中心发射的距离。只有Point和Spotlight有该参数。

Color：光源的颜色

Instensity：光源的强度，聚光灯以及点光源默认为1，方向光默认值是0.5

Cookie：用于为光源设置拥有alpha通道的纹理，时光线在不同地方有不同的亮度，如果是聚光灯(Spotlight)和方向光(DirectionalLight)，可以指定一个2D纹理。如果是一个点光源(Point light)，必须指定一个Cubemap(立方体纹理)。

Cookie Size：用于控制缩放Cookie投影，只有方向光(DirectionalLight)有该参数。

Shadow Type：阴影类型

No Shadows 关闭阴影

Hard Shadows 硬阴影

Soft Shadows 软阴影

个人理解：与现实世界对比，硬阴影就好比太阳光特别的强烈，照出来的影子有棱有角；软阴影就好比阴天的时候，但是有有那么一丝丝阳光，影子相比没那么明显，需要注意的是软阴影会消耗系统更多的资源。注意：默认设置下，只有Directional light光源才可以开启阴影，Pointlight、Spotlight光源开启阴影的话会弹出提示(Only directionallight have shadows in forward redering)，意思就是说只有Directionallight光源在Forward模式下才可以开启阴影(只有发布web版或单机版才支持)

Strength：强度，就是晴天跟阴天的效果吧

Resolution：控制阴影分辨率的质量

Bias：设置灯光控件的像素位置与阴影贴图值比较的偏移量，取值范围0~0.5，当值过小，对象表面会产生self-shadow，就是物体的表面会有来自于自身阴影的错误显示；当值过大，阴影就会较大程度的偏离投影的对象。

Softness：控制阴影模糊采样去的偏移量，只有方向光设置为软阴影的情况才会启用。

Softness Fade：控制阴影模糊采样区的偏移量，有有方向光设置为软投影的情况下才会启用。

Draw Halo：勾选此项，光源会开启光晕效果。

Flare：耀斑/炫光，镜头光晕效果。

RenderModel 渲染模式

Auto：自动，根据光源的亮度以及运行时Quality Settings的设置来确定光源的渲染模式。

Important：重要，逐像素进行渲染，一般用于非常重要的光源渲染

Not Important：光源总是以最快的速度进行渲染。

Culling Mask ：剔除遮蔽图，选中层所关联的对象将收到光源照射的影响。

Lightmapping：光照贴图，用于控制光源对光照贴图的影响模式

需要注意的是，如果场景中对光源要求不高的话，就尽量选用Directional Light光源，另外两个光源(PointLight,Spotlight)会比较消耗内存资源。

项目实例：地形之上，添加光源，分别将场景设置为白昼和夜晚效果。

(四)Unity音频

Unity中目前支持的音频剪辑有4种音频格式

MP3:适合较长音频，作为背景音乐

Ogg:适合较长音频，作为背景音乐

Wav:适合较短音频，作为环境音效

Aiff:适合较短音频，作为环境音效

音频侦听器（Audio Listener）

音频侦听器（Audio Listener）充当类似于麦克风的设备。它从场景中的任何给定音频源（Audio Source）接收输入，并通过计算机扬声器播放声音。对于大多数应用程序，最有意义的是将侦听器附加到相机（Camera）。如果音频侦听器（Audio Listener）处于混响区域（Reverb Zone）边界内，则会对场景中的所有可听见声音应用混响。（仅限专业版）而且，音频效果（Audio Effect）可以应用于侦听器，并且将应用于场景中的所有可听见声音。

音频侦听器（Audio Listener）没有属性。它只是必须添加才能正常工作。它在默认情况下始终添加到主相机（Main Camera）。

音频侦听器（Audio Listener）与音频源（Audio Source)协同工作，使您可以为游戏创造听觉体验。将音频侦听器（Audio Listener）附加到场景中的游戏对象（GameObject）时，会选取足够接近侦听器的任何源并将其输出到计算机的扬声器。每个场景都只能有 1 个音频侦听器（Audio Listener）正常工作。

实例分析：

①创建一个Empty对象 点击菜单栏[GameObject]->Create Empty 命名为Audio

②选取这个空对象 点击菜单栏[Component]->Audio->Audio Source 这样就在场景里创建了如图所示的一个喇叭

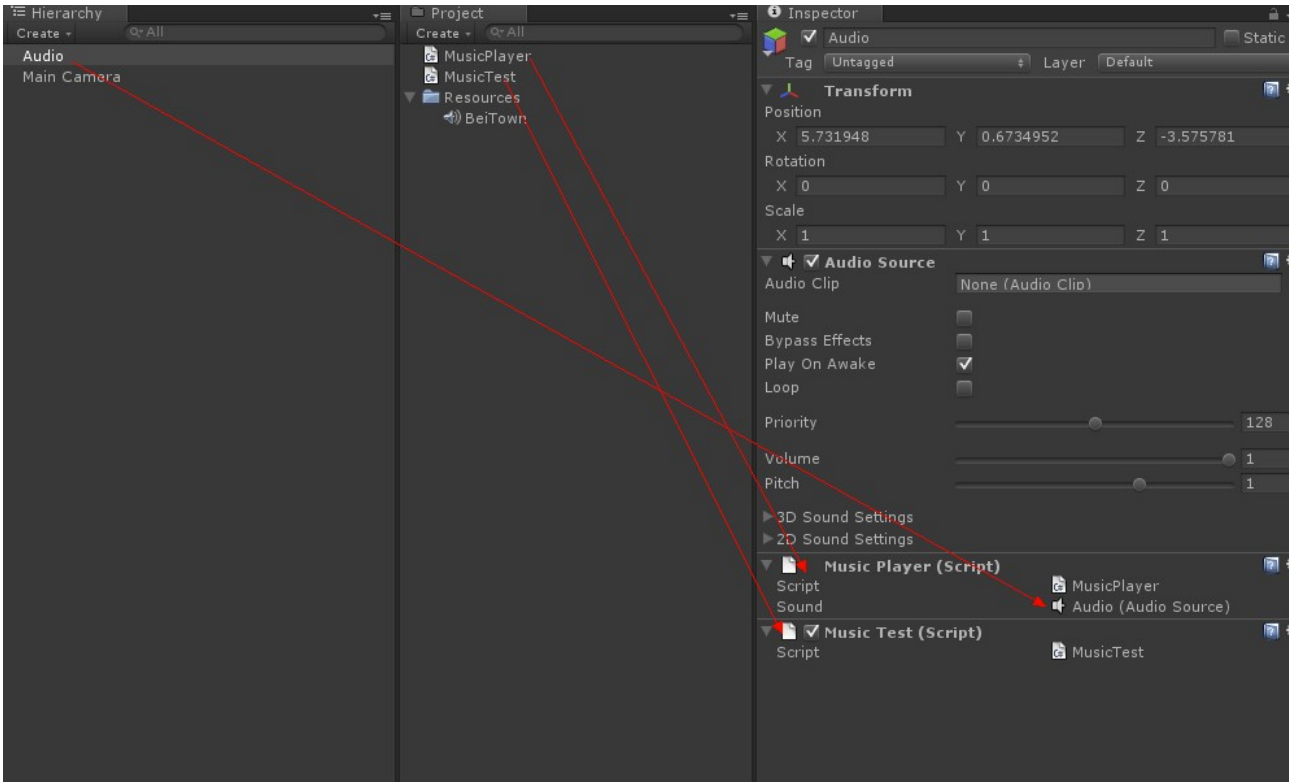
③创建资源文件夹命名为Resources，并把BeiTown.mp3文件拖入其中，注意一定要将资源文件夹名设为Resources，否则Resources.Load方法将无法找到资源文件

④创建脚本

接下来开始创建脚本

```
using UnityEngine;
using System.Collections;

public class MusicPlayer: MonoBehaviour {

     public AudioSource sound;
     private bool isPlaying;

     public void Play(string name)
     {
          sound.clip = (AudioClip)Resources.Load (name,
typeof(AudioClip));
          isPlaying = !isPlaying;
          if (isPlaying) {
               sound.Play ();
          }  else {
               sound.Pause();
          }
     }
     // Use this for initialization
     void Start () {

     }
```

```
    // Update is called once per frame
    void Update () {}
}


public class MusicTest : MonoBehaviour {
    //need assign to musictest
    private MusicPlayer music;
    // Use this for initialization
    void Start () {

        music = (GetComponent("MusicPlayer") as MusicPlayer);//获取播放器对象

    }

    void OnGUI()
    {
        if(GUI.Button(new Rect(10, 10, 100, 50), "PLAY")){

            music.Play("GameGUI_BackgroundAudio");//调用播放器Play方法

        }

    }

    // Update is called once per frame
    void Update () {

    }
}
```



⑤将脚步绑定到AudioSource音源对象上，并将Audio对象赋给MusicPlayer脚本里的Sound对象，以便MusicPlayer对其进行操作。

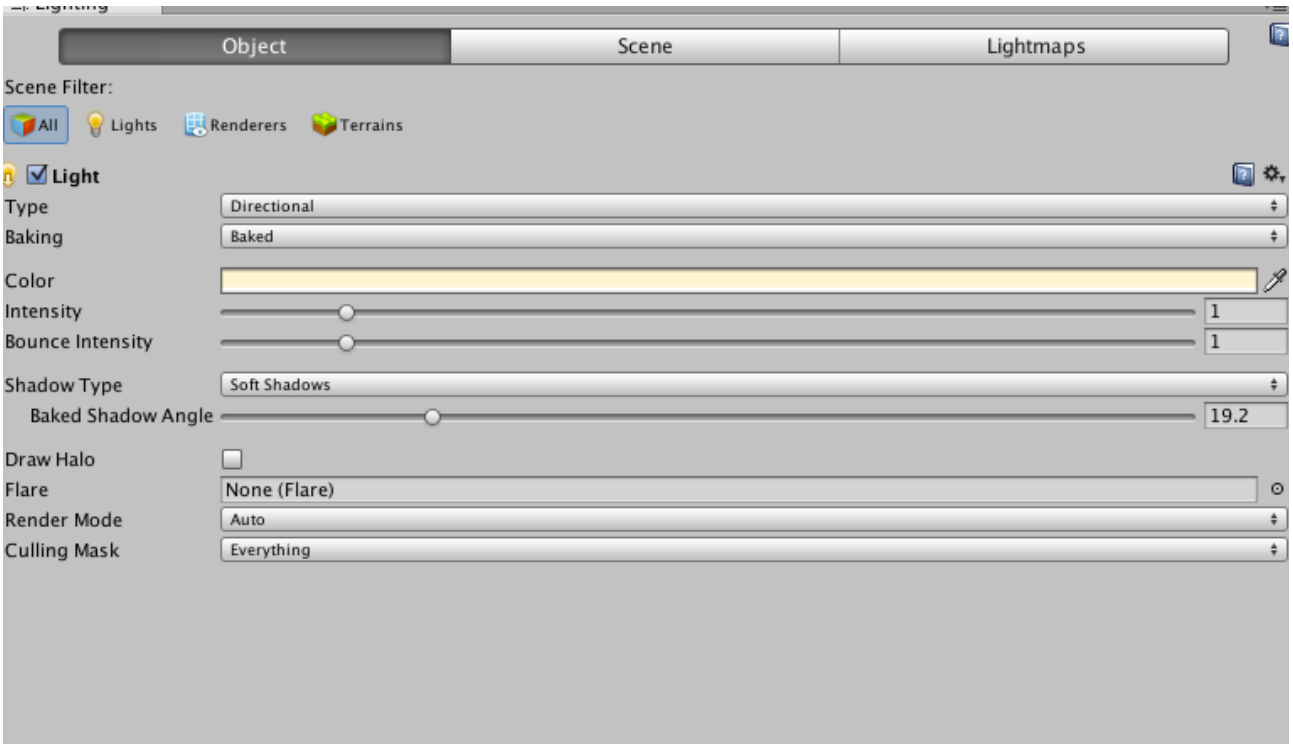
五：光照烘焙技术

光照烘焙技术是Unity提供的静态场景效果增强技术，其目的是在消耗很少系统性能的前提下制作出效果丰富，逼真，画面绚丽的游戏场景。其中光照烘焙分为静态光照烘焙以及灯光探测器技术。我们首先讲解静态光照烘焙。

（1）新建一个项目，使用Unity内置的对象，搭建一个基本的场景。

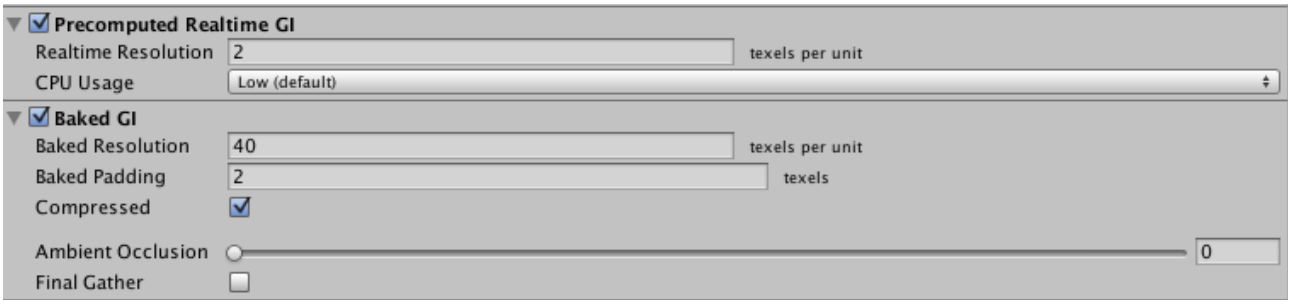
（2）把项目层级的所有游戏对象设置为静态的，为进行光照烘焙做准备

（3）选中直线光，设置Baking为Baked，ShadowType为SoftShadows,ShadowAngle为光线衍射范围。



（4）选中Scene选项，选择Baked GI

参数名	功能
Type	设置灯光类型
Baking	光源的烘焙模式
RealTime	对场景中的所有物体使用实时光照



参数名	功能
Baked	对静态物体使用烘焙光照，非静态无效
Mixed	对静态物体使用烘焙光照，非静态物体使用实时光照
Bounce Intensity	间接光（一个物体反射到另外一个物体的光）强度

BakedGI参数介绍

参数名	功能
BakedResolution	烘焙分辨率，若单位为10表示每个单位分布10个纹理元素
Baked padding	不同物体烘焙图的间距
Compressed	压缩光照图，移动设备必须勾选
AmbientOcullsion	环境阻光
FinalGather	发射的光线数量

（5）点击烘焙按钮Build,即可，看到游戏效果。

（六）遮挡剔除(OcclusionCulling)

遮挡剔除（Occlusion Culling）功能可在对象因被其他物体遮挡，当前在相机中无法看到时，禁用对象渲染。该功能不会在三维计算机图形中自动开启，因为在大部分情况下，离相机最远的对象最先渲染，离相机近的对象覆盖先前的物体（该步骤称之为“重复渲染（overdraw)”）。遮挡剔除（Occlusion Culling）与视锥体剔除（Frustum Culling）不同。视锥体剔除（Frustum Culling）只禁用相机视野外的对象渲染，不禁用视野中被遮挡的任何物体的渲染。注意，使用遮挡剔除（Occlusion Culling）功能时，仍将受益于视锥体剔除（Frustum Culling)。

当场景中包含大量模型时，势必会造成渲染效率的降低。如果使用遮挡剔除技术，可以使那些被阻挡的物体不被渲染，从而达到提高渲染效率的目的。

遮挡剔除的基本原理是在场景中创建一个遮挡区域，该遮挡区域由单元格组成；每个单元格构成整个场景遮挡区域的一部分，这些单元格会把整个场景拆分为多个部分。当摄像机能够看到该单元格时，单元格中的物体就会被渲染出来，而被其他单元格挡住的不被摄像机看到的物体不会被渲染。

下面，我们来做遮挡剔除的案例

（1）首先构建一个高楼林立的虚拟世界。

（2）除了主角，摄像机，直线光，地面，把层级视图中所有对象标记为遮挡静态。

特别提示：当我们选择一个包含多个字对象的父对象，会出现是否修改子对象的提示，根据自己的需求修改即可。

（3）执行菜单命令window->OculusionCulling，弹出遮挡剔除面板，单机Bake按钮，开始烘焙。

（4）单击遮挡剔除窗口的Visualizatior运行程序，进行测试。修改第一人称的摄像机位置。

当摄像机低头看地的时候，整个游戏场景的所有游戏对象都无需渲染。向左看，右边的无需渲染。虽然场景中的大部分对象都在摄像机范围之内，但是由于面前对象的遮挡，后面的高楼是不显示的。

(7)层级消隐

如果场景中存在大量的小物体，则可以使用层消隐优化场景。层消隐就是在比较远的距离将小物体剔除，减少绘制调用的次数。例如，在比较远的距离，大型建筑物依然可见，但是小型的石块和碎片隐藏掉。可以将小物件单独的放入一个层，并且使用Camera.main.layerCullDistance函数设置层的消隐距离。调整摄像机位置进行测试即可。只有在摄像机距离这些物体小于10M的时候，地面上的这些物体才能显示出来。

```
public class SeperateControl : MonoBehaviour {
    // Use this for initialization
    void Start () {
        float[] distances = new float[32];
        distances [8] = 10;
        Camera.main.layerCullDistances = distances;
```

```
    }

    // Update is called once per frame
    void Update () {

    }
}
```

(8)层级细节

层级细节LOD全称为LevelOfDetail,它是根据物体在游戏画面中所占据的百分比来diao用不同复杂度的模型的。简单理解就是当一个物体距离摄像机比较远的时候，使用复杂度低的模型，比较近的时候，使用复杂度高的模型。

在建模软件中，制作好各个层级的模型，并且根据复杂程度自高向低命名为:模型名称_LOD0,模型名称_LOD1,模型名称_LOD2,数字越低，复杂程度越高。

我们新建一个场景，构造最简单的LOD模型示例。

（1）准备3个Unity基本游戏对象，添加必要的材质。

(2)定义一个空对象，命名为_LOD,添加LODGroup组件

（3）分别将以上三个基本对象拖拽到LODGroup的各个级别上

（4）首先添加LOD0的对象，当然中间需要修改父节点，点击确定即可

（5）在Scene视图中，拖动摄像机分别近距离与远距离观察模型的变化。

(9)Profile工具使用

(10)项目优化策略

项目优化技能是优秀研发人员的基本素质，除了以上我们介绍的两种主要性能优化方式与性能检测工具。下面还有丰富的经验与优化建议与大家共享。下面从六个方面进行归纳与总结

- ✺DrawCall
- ✺模型 / 图像方面
- ✺光照与摄像机处理
- ✺程序优化方面
- ✺Unity系统设置

✺开发与使用习惯

一：DrawCall优化

对于Unity研发人员来说，系统的性能优化几乎等同于DrawCall优化，重要性可见一斑。

一个模型的数据经过CPU传递到GPU,并且命令GPU进行绘制，称为一个DrawCall。

Unity引擎准备数据并且渲染对象的过程是逐个对象进行的，所以对于每个游戏对象，不仅GPU的渲染很耗时，引擎重设材质与Shader也是一项非常耗时的操作。因此，每一帧的drawcall是一个非常重要的性能指标

降低DrawCall的基本原理基于DrawCall是CPU调用底层图形接口，对于GPU来说，一个对象与大量游戏对象，其图形处理的工作量是一样的。所以对于DrawCall的优化，主要工作量就是为尽量减少CPU在调用图形接口上的开销而努力。针对drawCall,我们的主要思路是，每个游戏对象尽量减少渲染次数，多个游戏对象尽量一起渲染。

降低DrawCall 的主要途径

一般项目中，角色和场景是消耗资源最多的两个方面，其中角色是CPU的瓶颈，场景是GPU的瓶颈。所以项目的优化就是降低DrawCall。总体思路就是对美术资源进行合并，大量合并drawcall、人物角色减少材质与纹理的依赖，简化多余特效等。当然也可以允许玩家在低端设备上关闭一些特效换取更佳流畅的性能。

下面通过10个途径来进行详细讨论

1:DrawCall批处理技术(DrawCall Batching)

Unity运行的时候可以将一些游戏对象进行合并，也就是把多个游戏对象打包，然后再使用一个DrawCall来渲染他们，这个操作称为批处理。

批处理的核心在于可见性测试之后，检查所有要绘制的对象材质，把相同材质的合为一个对象，这样就可以在一个call里面处理多个对象了。

Unity提供了静态批处理和动态批处理两种方式，动态批处理是完全自动进行的，不需要也无法进行任何干预，对于顶点数在900以内的可移

动物体，只要使用相同的材质，就会组成批处理。静态批处理则需要把静态的物体标记为静态，然后无论大小，都会进行批处理。

为了更好的使用静态批处理，需要明确指出哪些物体是静止的，并且在游戏中永远不会移动、旋转与缩放。在属性窗口将static勾选即可。所以应该尽量将非运动对象设置为static。

2:使用图集减少材质的使用

Unity判断哪些对象进行批处理时候，一般根据这些对象是否具有共同的材质和贴图，也就是说拥有相同材质的对象才可以进行批处理。因此，尽量复用材质到不同的对象上。

3:尽量少使用反光阴影，因为会使物体多次渲染

4:视锥体合理裁切

视锥体合理裁切是Unity自带的功能，我们需要做的就是寻求一个合理的裁剪平面。一般来说，对于大型场景中的大量游戏对象进行合理分层，对于大型建筑物使用大型裁剪距离，对于小游戏对象使用小裁剪距离，场景中的粒子对象使用更小的裁剪距离。

5:遮挡剔除

6:网格渲染器的控制

可以将暂时不进行显示的对象render设置为unenable，需要的时候在显示

7:减少游戏对象的缩放

分别拥有大小（1，1，1）和（2，2，2）的两个对象将不会进行批处理。

8:减少多通道Shader的使用

多通道的Shader会防碍批处理操作

9:尽量多使用预设体

使用预设体的对象会自动使用相同的网格模型和材质，因此会自动被批处理。

二：项目优化之模型与图像方面

模型与图像方面的优化分为以下6个途径进行讨论

途径1:模型优化

（1）模型几何体的优化

模型的优化主要是模型的顶点，三角面数量。如果可能，把相近的合并为一个。比如森林与树木，完全可以坐在一起

（2）压缩面片：3D模型导入Unity之后，不影响显示效果的前提下，最好打开MeshCompression,Off,Medium,Low,High几个选项，根据需要进行设置。

（3）避免大量使用Unity自带的Sphere对象

内建的部分游戏对象，多边形数量比较大，如果物体不要求特别圆滑，导入其他的模型替换。

途径2:纹理优化

（1）使用贴图压缩优比：尺寸越小，压缩比率越高的贴图，占用的内存空间越低，因此可以降低对他的渲染处理时间，同时减少游戏文件的体积。

（2）贴图压缩格式选择：纹理建议使用压缩纹理。不透明贴图的压缩格式为ETC。安卓的GPU有多种，但是都支持ETC格式。透明贴图，我们选择RGBA 16或者32

（3）选择支持MipMap

Mipmap技术有点类似于LOD技术，但是不同的是，LOD针对的是模型资源，而Mipmap针对的纹理贴图资源

使用Mipmap后，贴图会根据摄像机距离的远近，选择使用不同精度的贴图。

缺点：会占用内存，因为mipmap会根据摄像机远近不同而生成对应的八个贴图，所以必然占内存！

优点：会优化显存带宽，用来减少渲染，因为可以根据实际情况，会选择适合的贴图来渲染，距离摄像机越远，显示的贴图像素越低，反之，像素越高！

MipMap可以用于跑酷类游戏，当角色靠近时，贴图清晰显示，否则模糊显示

如果我们使用的贴图不需要这样效果的话，就一定要把Generate Mip Maps选项和Read/Write Enabled选项取消勾选！因为Mipmap会十分占内存！

mipMap会让你的包占更大的容量！

下面来看下怎么设置贴图的mipmap:

设置贴图的Texture Type为Advanced类型 → 勾选Generate Mip Maps → Apply应用

途径3:材质

尽量合并使用相同材质球的对象，尽可能减少网络材质的使用数量。尽量使用合图工具。

途径4:碰撞体

如果可以，尽量不使用MeshCollider，如果不能优化，则尽量减少面数。车轮碰撞和布料也是尽量少用

途径5:粒子系统

屏幕上的最大粒子数建议小于200，每个粒子发射器的最大发射数量建议不超过50.如果可以，粒子的size尽量小点。非常小的粒子，去掉alpha通道。不要开启粒子的碰撞功能。

途径6:其他

尽量使用预设体

三：光照与摄像机处理

分为三个途径进行探讨

途径1:渲染路径（RenderingPath）

Unity提供了不同的渲染路径，这些渲染路径用于决定灯光和阴影的计算方式，不同的渲染路径具有不同的性能特性和渲染效果。Unity提供

了几种方法，分别是顶点光照（Vertex Lit），前向渲染(Forward Rending)，延时光照(Deferred Lighting)

Vertex Lit：顶点光照。摄像机将对所有的游戏对象座位顶点光照对象来渲染。

Forward：快速渲染。摄像机将所有游戏对象将按每种材质一个通道的方式来渲染。

Deferred Lighting：延迟光照。摄像机先对所有游戏对象进行一次无光照渲染，用屏幕空间大小的Buffer保存几何体的深度、法线已经高光强度，生成的Buffer将用于计算光照，同时生成一张新的光照信息Buffer。最后所有的游戏对象会被再次渲染，渲染时叠加光照信息Buffer的内容。

途径2:光照与阴影

使用光照烘焙技术。

光线性能消耗占用顺序为：聚光灯＞点光源＞平行光。尽量使用小的点光源，影响的物体少，不在范围的不受影响。一个网格在有8个以上光源影响的时候，只响应前8个最亮的光源。

如果硬阴影可以解决问题，不要使用软阴影，并且使用不影响效果的低分辨率阴影。实时阴影很消耗性能，允许的话在大场景中使用线性雾，这样可以使的远距离对象或者阴影不易察觉，可以减少摄像机的远裁切距离和阴影距离提高性能。

质量设置面板的ShadowDistance属性设置阴影的显示距离，该距离是根据当前摄像机参考的。当可以生成阴影的物体与当前摄像机距离超过该值时候，就不产生阴影。

途径3：摄像机技巧

层级消隐

四：程序优化方面

从四个方面讨论这个问题

1:程序整体优化方面

（1）如果项目的瓶颈不在渲染，那么肯定就在脚本了。我们要删除脚本中不用或者为空的默认方法，尽量少在Update里面做事情，脚本不用时候禁止。

（2）尽量不使用原生GUI，使用NGUI或者UGUI

（3）需要隐藏或者显示来回切换的对象，少用active，可以将其移出摄像机的范围。也可以使用脚本方式开启或者关闭游戏对象的MeshRender组件

（4）不要频繁获取组件，声明为成员或者属性

（5）脚本在不使用时候禁用，使用时候启用

（6）尽量获取一次脚本组件之后，使用变量保存

（7）尽量少使用update等每一帧调用的函数，节省电量

（8）使用C#的委托与事件机制，比使用SendMessage机制效率高多了。

2:事件函数方面

（1）按照脚本生命周期的原理，对于协程，调用函数，一般脚本禁用的时候，需要显示禁用

（2）同一脚本频繁使用的变量，声明为全局；脚本之间频繁使用的，声明为静态

（3）尽量避免每一帧处理，可以每隔几帧处理一次

```
function Update(){if(Time.frameCount%100==0){  DoSomething()}}
```

(4)使用协同或者InvokeRepeating代替不必每帧都执行的函数

（5）避免在Update等函数中使用搜索方法，比如GameObject.Find()。良好的代码是把搜索放在单次执行的函数里面，比如 Start（）；

3:数学计算方面

尽量使用int代替float，少用正弦等三角函数，尽量少用除法运算和取模运算

4:垃圾回收方面

（1）尽量主动回收垃圾

（2）回收的时机很重要。尽量放在游戏场景的加载与场景结束的时候，主动卸载资源。

（3）避免频繁分配内存：使用对象池

（4）在较大的场景中，距离摄像机远的对象可以将上面的脚本禁用。需要启用再次setActive即可。这样也可以配合事件函数中的OnBecameInvisible和OnBecameVisible，使的可见时候启用，不可见禁用。

（5）善于使用OnBecameInvisible和OnBecameVisible控制物体Update的执行

（6）资源预先加载技术

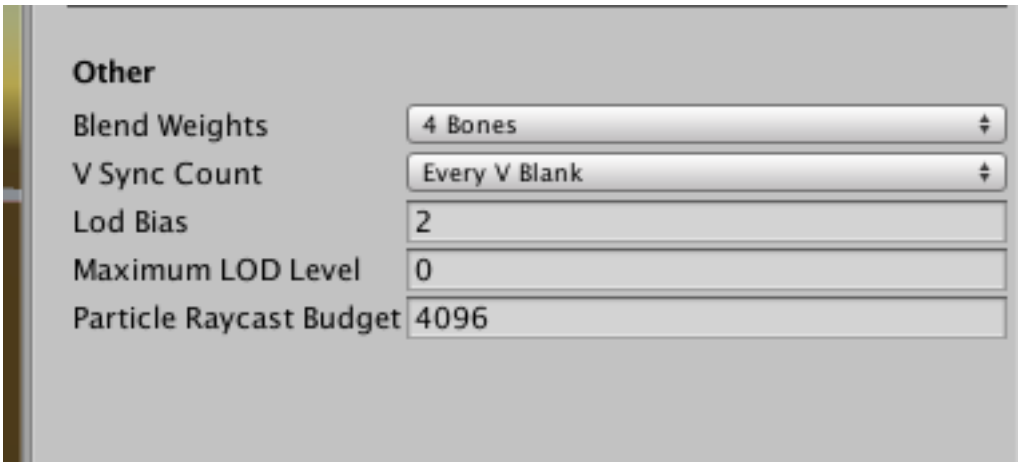
五：项目优化之Unity系统设置

1:限帧措施

手机上主动减少FPS，能够显著的减少发热和耗电，稳定游戏FPS。

具体方法：

Sync Count会影响你的FPS，EveryVBlink相当于FPS＝60；EverySecond VBlink相当于30。这两种情况如果都不符合我们的FPS，那么我们需要手动设置FPS。首先关闭垂直同步功能，然后在代码的Awake函数里面手动设置FPS(Application.targetFrameRate=45).



2:物理性能优化

1）增加固定时间步长

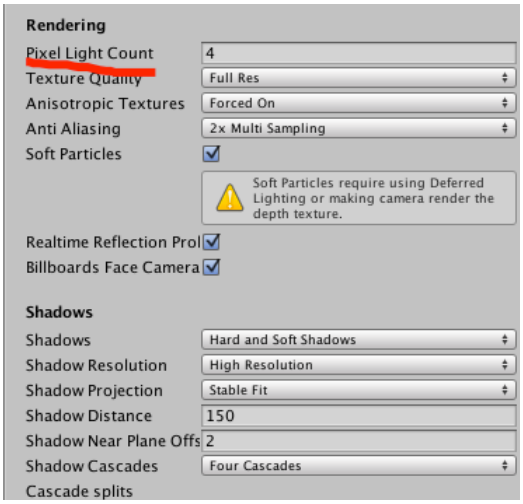
设置为0.04-0.06之间即可。降低了物理引擎刚体更新的频率

2）设置最大允许时钟回调

物理计算和FixedUpdate执行不会超过该值制定的时间。使的在最坏的情况下封顶物理计算时间

3）设置时间缩放因子

3:调整像素光数量：使游戏看起来更多彩



六：项目优化之良好开发习惯与使用习惯

（1）养成良好的标签，层次，Layer条理化习惯，将不同的对象置于不同的标签或者图层，三者的有效结合将按照名称，类别和属性进行查找

（2）研发过程中经常通过Status或者Profile查看对效率影响最大的方面或对象，而不是等到项目发布再去解决。

（3）采取合适的项目框架，便于维护与协作开发