
Unity 4.x 2D

游戏开发基础教程

（内部资料）

大学霸



大学霸

www.daxueba.net

前 言

Unity 是一款综合性的游戏开发工具，也是一款全面整合的专业游戏引擎。它可以运行在 Windows 和 Mac OS X 下，并提供交互的图形化开发环境为首要操作方式。使用 Unity 开发的游戏，可以部署到所有的主流游戏平台，而无需任何修改。这些平台包括 Windows、Linux、Mac OS X、iOS、Android、Xbox 360、PS3、WiiU 和 Web 等。开发者无需过多考虑平台之间的差异，只需把精力集中到制作高质量的游戏即可，真正做到“一次开发，到处部署”。

据权威机构统计，国内 53.1% 的人使用 Unity 进行游戏开发；有 80% 的手机游戏是使用 Unity 开发的；苹果应用商店中，有超过 1500 款游戏使用 Unity 开发。

网上有为数众多的 2D 和 3D 游戏。稍微关注一下，就会发现 2D 游戏才是主流，如植物大战僵尸、愤怒的小鸟、打飞机、2048 等。而且，问问身边的人让他们印象深刻的游戏是什么，你会惊讶的发现，大部分游戏同样是 2D 的。

基于以上不可忽略的事实，本书决定着眼于讲解使用 Unity 开发 2D 游戏的基础知识，且书中包含了两个生动的 2D 游戏示例，相信读者会喜欢它们的。

学习所需的系统和软件

- ☐ 安装 Windows 7 操作系统
- ☐ 安装 Unity 4.5.1

www.daxueba.net

目 录

第 1 章	Unity 及其组成的介绍.....	错误!未定义书签。
1.1	Unity 概述.....	错误!未定义书签。
1.2	项目、资源和场景.....	错误!未定义书签。
1.2.1	项目.....	错误!未定义书签。
1.2.2	资源.....	错误!未定义书签。
1.2.3	场景.....	错误!未定义书签。
1.3	场景视图的操作.....	错误!未定义书签。
1.3.1	使用快捷键操作场景视图.....	错误!未定义书签。
1.3.2	使用 Gizmo 操作场景视图.....	错误!未定义书签。
1.4	游戏对象和组件.....	错误!未定义书签。
1.5	脚本与脚本编辑器.....	错误!未定义书签。
1.5.1	创建脚本.....	错误!未定义书签。
1.5.2	脚本编辑器.....	错误!未定义书签。
1.6	脚本的调试.....	错误!未定义书签。
1.6.1	调试方法一.....	错误!未定义书签。
1.6.2	调试方法二.....	错误!未定义书签。
第 2 章	材质和纹理.....	8
2.1	材质和纹理的使用.....	8
2.1.1	使用材质.....	8
2.1.2	不同的材料类型——着色器.....	11
2.1.3	使用纹理.....	12
2.2	应用于 2D 游戏的材质.....	14
2.2.1	缘由.....	14
2.2.2	技巧一：使用白色的环境光.....	14
2.2.3	技巧二：使用光不敏感着色器.....	15
2.3	纹理使用规则.....	18
2.3.1	规则 1：分辨率是 2 的次方.....	18
2.3.2	规则 2：保证“质量”.....	19
2.3.3	规则 3：增加阿尔法通道（Alpha Channel）.....	20
2.4	导入纹理.....	21
2.4.1	导入纹理时默认设置介绍.....	21
2.4.2	含有透明信息的纹理.....	23
第 3 章	着手开发一个简单的 2D 游戏.....	错误!未定义书签。
3.1	开始开发 2D 游戏.....	错误!未定义书签。
3.1.1	导入纹理资源.....	错误!未定义书签。
3.1.2	新建材质资源.....	错误!未定义书签。
3.1.3	修改场景的环境光以及游戏时的屏幕尺寸.....	错误!未定义书签。
3.2	为场景添加游戏对象.....	错误!未定义书签。

3.2.1	调整游戏对象的角度.....	错误!未定义书签。
3.2.2	改变游戏对象的位置.....	错误!未定义书签。
3.2.3	游戏对象的“碰撞”组件.....	错误!未定义书签。
3.3	让飞船动起来.....	错误!未定义书签。
3.4	让飞船发射子弹.....	错误!未定义书签。
3.4.1	在场景中添加子弹.....	错误!未定义书签。
3.4.2	游戏时，让子弹在场景中移动.....	错误!未定义书签。
3.4.3	生成子弹的预设.....	错误!未定义书签。
3.4.4	设置子弹的发射位置.....	错误!未定义书签。
3.4.5	在恰当的时机发射子弹.....	错误!未定义书签。
3.5	让外星飞船动起来.....	错误!未定义书签。
3.5.1	编写脚本.....	错误!未定义书签。
3.5.2	设置外星飞船的触发器.....	错误!未定义书签。
3.5.3	为子弹预设添加刚体组件.....	错误!未定义书签。
3.6	为游戏添加背景.....	错误!未定义书签。
第 4 章	使用编辑器类自定义编辑器.....	错误!未定义书签。
4.1	编辑器类.....	错误!未定义书签。
4.2	开始使用编辑器类编写工具.....	错误!未定义书签。
4.2.1	为项目添加脚本.....	错误!未定义书签。
4.2.2	创建指定名称的文件夹.....	错误!未定义书签。
4.3	把工具添加到菜单.....	错误!未定义书签。
4.3.1	CreateWizard函数.....	错误!未定义书签。
4.3.2	测试脚本的实现效果.....	错误!未定义书签。
4.4	读取场景中选择对象.....	错误!未定义书签。
4.4.1	在脚本中使用Selection类.....	错误!未定义书签。
4.4.2	测试脚本的实现效果.....	错误!未定义书签。
4.5	为工具窗口添加用户输入框.....	错误!未定义书签。
4.6	完成工具的所有功能.....	错误!未定义书签。
第 5 章	图片与几何图形对象.....	错误!未定义书签。
5.1	2D游戏常用的图片.....	错误!未定义书签。
5.1.1	精灵.....	错误!未定义书签。
5.1.2	图块集.....	错误!未定义书签。
5.1.3	图形绘制中的问题.....	错误!未定义书签。
5.1.4	设想.....	错误!未定义书签。
5.2	开始编写编辑器工具.....	错误!未定义书签。
5.3	设置四边形的轴点.....	错误!未定义书签。
5.4	指定四边形资源的存放路径.....	错误!未定义书签。
5.5	生成四边形.....	错误!未定义书签。
5.5.1	阶段一：创建构成四边形的顶点、UV和三角形.....	错误!未定义书签。
5.5.2	阶段二：在资源面板中生成四边形.....	错误!未定义书签。
5.5.3	阶段三：在场景中实例化一个四边形.....	错误!未定义书签。
5.6	使用四边形生成工具.....	错误!未定义书签。
第 6 章	生成纹理图集.....	错误!未定义书签。

6.1	为什么要使用纹理图集.....	错误!未定义书签。
6.1.1	降低绘制调用的次数.....	错误!未定义书签。
6.1.2	便于灵活的使用纹理.....	错误!未定义书签。
6.1.3	便于管理纹理.....	错误!未定义书签。
6.2	开始编写生成纹理图集的工具.....	错误!未定义书签。
6.3	添加组成纹理图集的纹理.....	错误!未定义书签。
6.4	UV对纹理图集的重要性.....	错误!未定义书签。
6.5	生成纹理图集.....	错误!未定义书签。
6.5.1	步骤一：优化输入的纹理.....	错误!未定义书签。
6.5.2	步骤二：构建纹理图集.....	错误!未定义书签。
6.5.3	步骤三：保存图集的预置.....	错误!未定义书签。
6.6	脚本文件TexturePacker代码汇总.....	错误!未定义书签。
6.7	测试工具的使用效果.....	错误!未定义书签。
第 7 章	UV和动画.....	错误!未定义书签。
7.1	生成一个可停靠的编辑器.....	错误!未定义书签。
7.2	编辑工具窗口的界面.....	错误!未定义书签。
7.2.1	添加预置资源选择区域.....	错误!未定义书签。
7.2.2	添加纹理选择区域.....	错误!未定义书签。
7.2.3	添加纹理选择的两种方式.....	错误!未定义书签。
7.2.4	编写用于修改网格对象UV坐标的函数.....	错误!未定义书签。
7.2.5	添加应用所有设置的按钮.....	错误!未定义书签。
7.3	工具脚本代码的汇总与使用.....	错误!未定义书签。
7.4	一个播放动画的平面对象.....	错误!未定义书签。
第 8 章	益于 2D 游戏的摄像机与场景设置.....	错误!未定义书签。
8.1	摄像机类型：透视与正交.....	错误!未定义书签。
8.2	世界单元与像素.....	错误!未定义书签。
8.3	世界单元与像素的转换.....	错误!未定义书签。
8.3.1	添加纹理和四边形对象.....	错误!未定义书签。
8.3.2	调整四边形与摄像机的位置.....	错误!未定义书签。
8.3.3	世界单元：像素 = 1: 1.....	错误!未定义书签。
8.3.4	对齐屏幕和场景坐标的原点.....	错误!未定义书签。
8.4	纹理图片的完美显示.....	错误!未定义书签。
8.5	其它有用的设置技巧.....	错误!未定义书签。
8.5.1	调节深度.....	错误!未定义书签。
8.5.2	合成视图.....	错误!未定义书签。
第 9 章	获取玩家对 2D 游戏的输入.....	错误!未定义书签。
9.1	自动检测鼠标单击事件.....	错误!未定义书签。
9.2	手动检测鼠标单击事件.....	错误!未定义书签。
9.2.1	鼠标按下的键及其位置.....	错误!未定义书签。
9.2.2	鼠标点击的第一个对象.....	错误!未定义书签。
9.2.3	鼠标点击的所有对象.....	错误!未定义书签。
9.3	修改游戏中的鼠标图标.....	错误!未定义书签。
9.3.1	准备所需的资源，并做适当设置.....	错误!未定义书签。

9.3.2	编写脚本.....	错误!未定义书签。
9.3.3	两个坐标系导致的问题.....	错误!未定义书签。
9.3.4	查看游戏视图中的效果.....	错误!未定义书签。
9.4	使用键盘控制鼠标移动.....	错误!未定义书签。
9.5	对输入的抽象——输入轴.....	错误!未定义书签。
9.5.1	了解输入轴.....	错误!未定义书签。
9.5.2	输入轴在输入过程中的应用.....	错误!未定义书签。
9.6	来自移动设备的输入.....	错误!未定义书签。
9.6.1	检测移动设备上的触摸操作.....	错误!未定义书签。
9.6.2	把触摸操作当作鼠标操作.....	错误!未定义书签。
9.6.3	有选择的编译代码.....	错误!未定义书签。
第 10 章	2D 卡片游戏——记忆大作战.....	错误!未定义书签。
10.1	游戏设计文档.....	错误!未定义书签。
10.2	开始着手创建游戏.....	错误!未定义书签。
10.2.1	在资源面板创建文件夹.....	错误!未定义书签。
10.2.2	创建一个纹理图集.....	错误!未定义书签。
10.2.3	创建四边形对象.....	错误!未定义书签。
10.2.4	修改四边形的材质和UV.....	错误!未定义书签。
10.2.5	设置摄像机和游戏视图的分辨率.....	错误!未定义书签。
10.3	设置场景中的卡片.....	错误!未定义书签。
10.3.1	设置卡片的属性.....	错误!未定义书签。
10.3.2	定位卡片的位置.....	错误!未定义书签。
10.3.3	编写控制卡片行为的脚本.....	错误!未定义书签。
10.3.4	补全场景中其余的卡片.....	错误!未定义书签。
10.4	游戏管理类.....	错误!未定义书签。
10.4.1	重置卡片.....	错误!未定义书签。
10.4.2	处理玩家输入.....	错误!未定义书签。
10.4.3	响应玩家输入.....	错误!未定义书签。
10.4.4	游戏管理类代码汇总.....	错误!未定义书签。
10.5	完善并运行游戏.....	错误!未定义书签。
10.5.1	替换系统鼠标图标.....	错误!未定义书签。
10.5.2	游戏运行效果展示.....	错误!未定义书签。
第 11 章	可联机玩的游戏——记忆大作战.....	错误!未定义书签。
11.1	网络连接.....	错误!未定义书签。
11.2	建立服务器端.....	错误!未定义书签。
11.3	建立客户端.....	错误!未定义书签。
11.4	测试网络连接的功能.....	错误!未定义书签。
11.5	网络视图组件.....	错误!未定义书签。
11.6	构建授权服务器.....	错误!未定义书签。
11.7	建立游戏输入操作的秩序.....	错误!未定义书签。
11.7.1	游戏启动时，禁止输入操作.....	错误!未定义书签。
11.7.2	连接建立后，允许服务器端的输入操作.....	错误!未定义书签。
11.7.3	服务器端远程调用客户端上的函数.....	错误!未定义书签。

11.7.4 客户端远程调用服务器端上的函数.....	错误!未定义书签。
11.8 修改游戏管理类脚本.....	错误!未定义书签。
11.9 游戏运行效果展示.....	错误!未定义书签。
11.10 为游戏添加分数记录.....	错误!未定义书签。
第 12 章 优化游戏的方法.....	错误!未定义书签。
12.1 最优化，如你所想吗？.....	错误!未定义书签。
12.2 减少顶点的数目.....	错误!未定义书签。
12.3 减少材质.....	错误!未定义书签。
12.4 减少UV接缝.....	错误!未定义书签。
12.5 不同平台下，纹理的不同设置.....	错误!未定义书签。
12.6 对象缓存组件.....	错误!未定义书签。
12.7 避免频繁使用Update()函数.....	错误!未定义书签。
12.8 合理使用Collider组件.....	错误!未定义书签。
12.9 避免使用OnGUI()和GUI类.....	错误!未定义书签。
12.10 使用静态批处理.....	错误!未定义书签。
12.11 使用天空盒子.....	错误!未定义书签。

大学霸

www.daxueba.net

第 2 章 材质和纹理

材质（materials）和纹理（textures）是构成 2D 游戏的原始材料。一个游戏中你所看到的，基本都是由它们实现的，例如人物角色、游戏场景，甚至是游戏菜单。本章将详细深入的讲解它们，包括它们的创建和本质上的区别。

2.1 材质和纹理的使用

在游戏中，一定会有游戏对象。如果想要改变游戏对象的外观，就需要使用材质和纹理。

2.1.1 使用材质

为了可以更加直观的体会材质的作用，可以实际的使用一下材质。

1. 应用于一个对象的材质

使用 Unity 创建一个新的项目。为这个项目添加 2 个游戏对象：球体（Sphere）和直射光（Directional light），如图 2-1 所示。再加上游戏本身的游戏对象：主摄像机（Main Camera），游戏中一共有 3 个游戏对象，如图 2-2 所示。

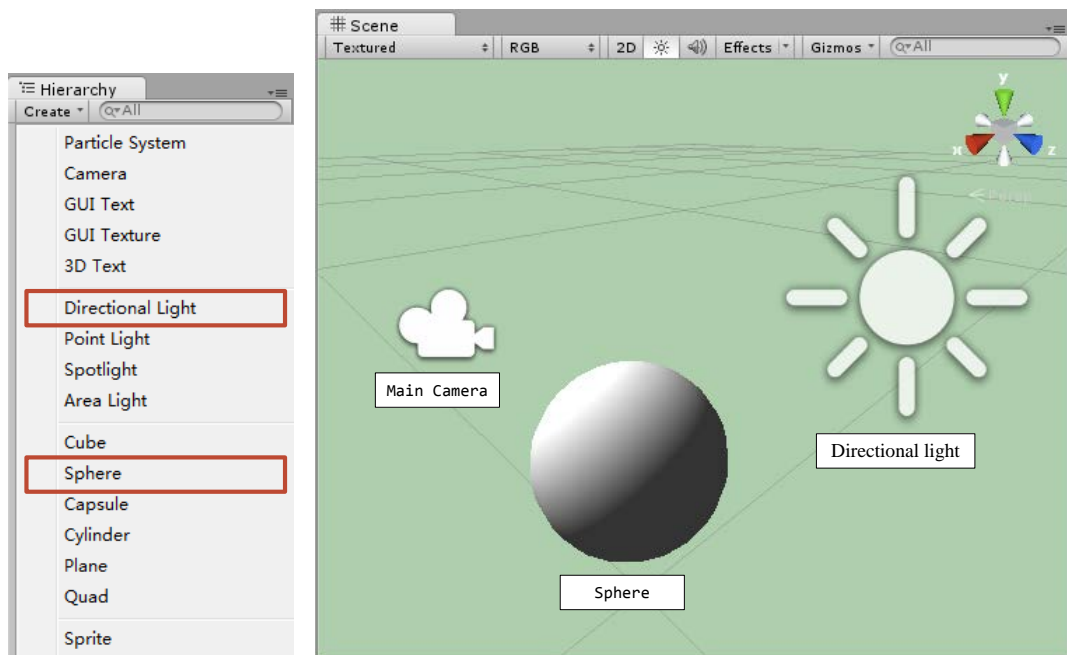


图 2-1 为项目添加的 2 个游戏对象

图 2-2 场景中的 3 个游戏对象

现在为这个游戏添加“材质”这个资源。右击资源面板，在弹出的快捷菜单中选择 Create|Material 命令，然后在资源面板中就会生成一个名为 New Material 的材质，如图 2-3 所示。

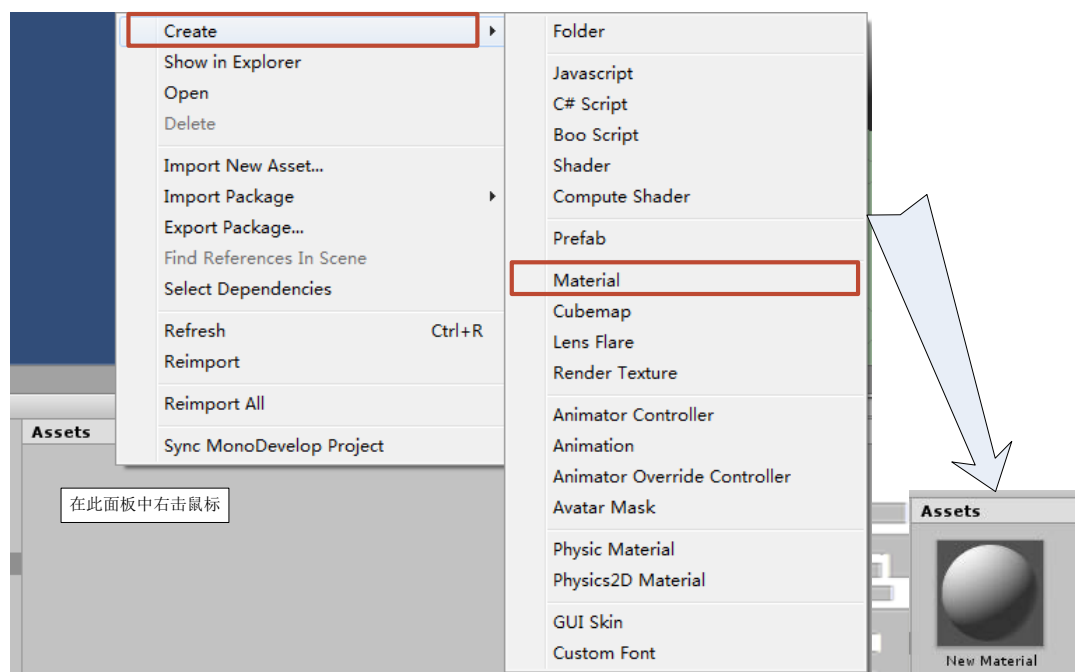


图 2-3 在 Assets 面板中创建一个材质

用鼠标单击这个新生成的材质，可以在对象查看器中看到这个材质的各种属性。新生成的材质默认的主色调是白色，很显然这个颜色是可以修改的，如图 2-4 所示的操作，可将这个材质修改为黄色。

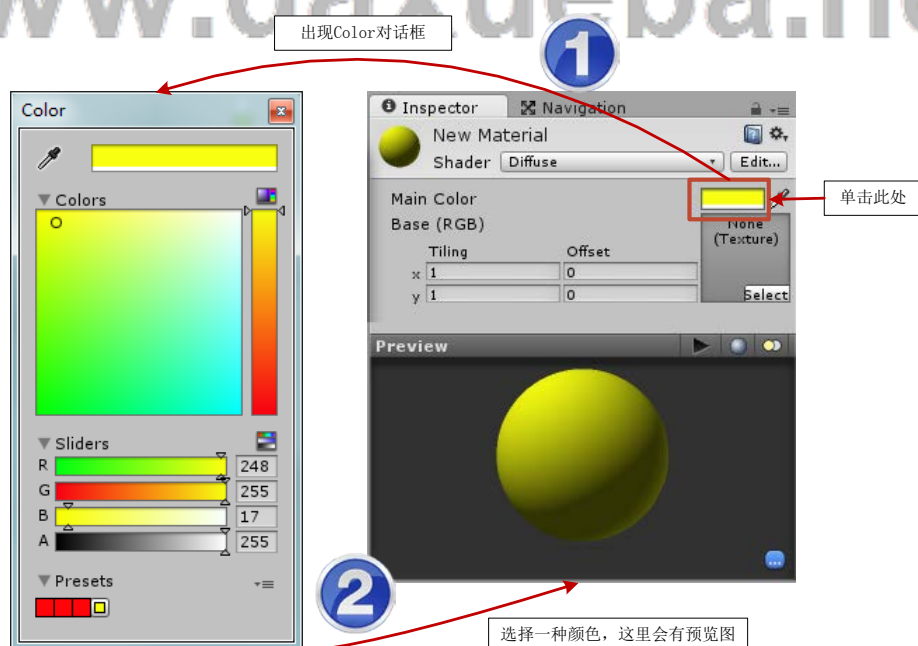


图 2-4 修改材质的 Main Color 属性

将这个材质资源拖动到球体对象上，球体表面的颜色会发生变化，由原来的灰色变成了黄色。拖动过程，使得球体对象引用了刚才我们生成的材质。由于材质可以决定游戏对象的外观，所以被赋予黄色材质的球体的外观就发生了变化。单击 Scene 视图中的球体对象，可以在球体查看器的 Mesh Renderer 组件下的 Materials 中找到我们刚才赋予的材质资源，如图 2-5 所示。

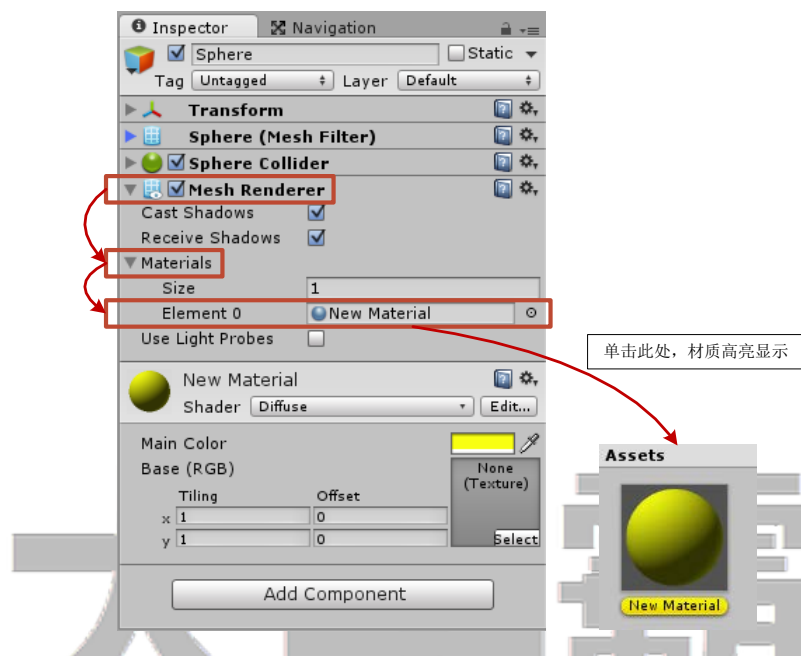


图 2-5 查看球体所赋予的材质

如果要找这个球体引用的材质，可以单击球体查看器面板中的材质名，然后资源面板中的相应材质会高亮显示。

2. 应用于多个对象的材质

材质可以同时被多个对象引用，于是当你改变这个材质的时候，这个改变会马上应用到所有引用这个材质的游戏对象上。如图 2-6，3 个游戏对象：球体、立方体、胶囊，都引用了同一个材质。当材质的主色调由黄色变为红色时，这 3 个游戏对象的颜色也会立即改变。

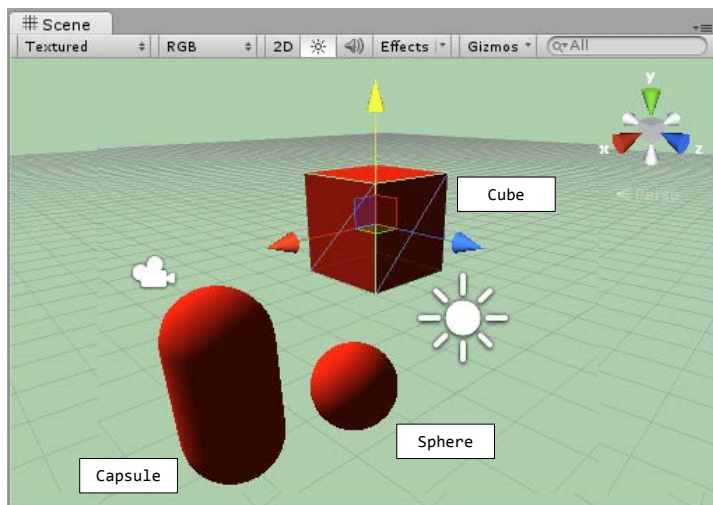


图 2-6 引用同一材质的 3 个游戏对象

2.1.2 不同的材料类型——着色器

材质不仅仅能指定对象的颜色，它还可以指定对象的其它外观属性。而每一种不同的外观属性有时会要求我们使用不同的材料类型，而不同的材料类型就是本小节要讲解的着色器（Shader）。改变着色器就可以改变材质在光照下的显示效果，即改变了对象的外观属性。

注意：材质和材料是不同的，材质的一个属性是材料，而多种材料是我们说的着色器。简单来说，材质包括材料，但材质又不仅仅是材料。

默认的着色器是漫反射（Diffuse），这种材质在光照下显的很粗糙，如图 2-7 所示。

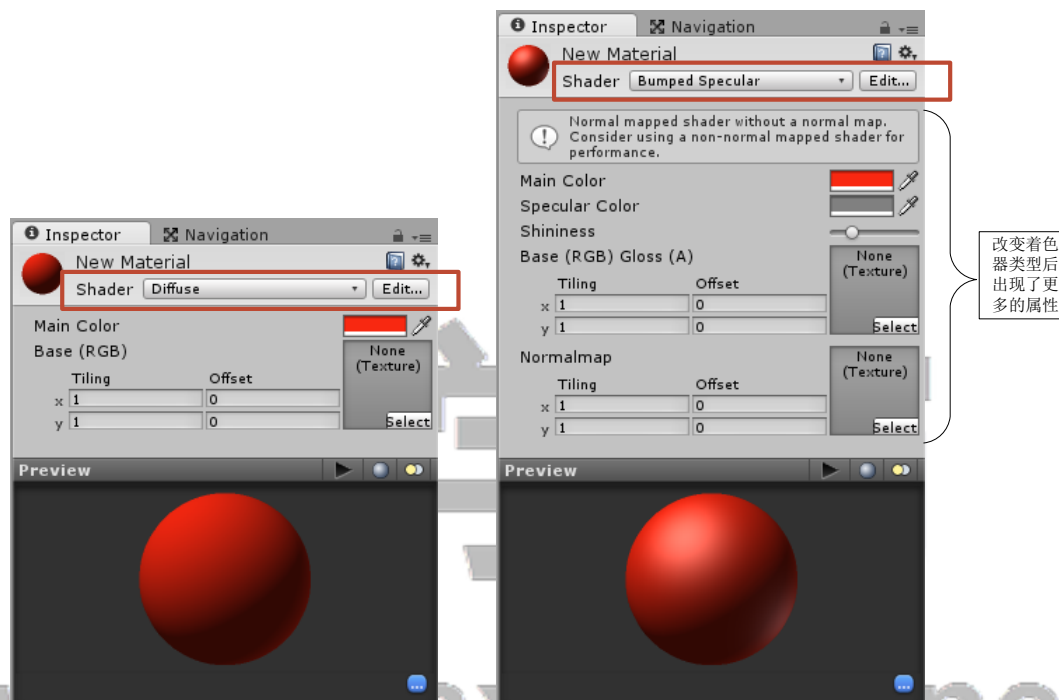


图 2-7 光照下很粗糙的着色效果

图 2-8 光照下显得很光滑

可以在材质的查看器中修改着色器的类型。例如，在着色器下拉列表中选中凸起的镜面反射（Bumped Specular），材质在光照下的显示效果就会发生很大的改变——变的光滑，看起来更有光泽了。如图 2-8 所示。改变材质的着色器后，又出现了其它的属性，如反射光的颜色（Specular Color）和反射光的强度（Shininess）。修改这 2 个属性后的材质显示效果如图 2-9 所示。

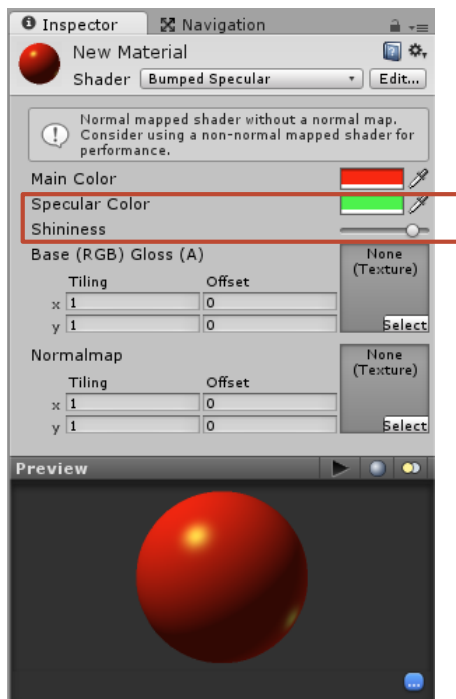


图 2-9 修改材质的反射光的颜色和反射光的强度

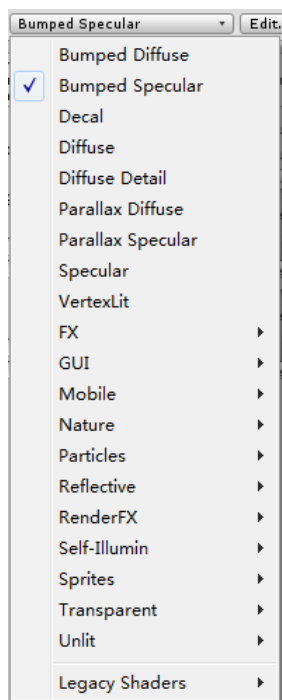


图 2-10 部分着色器的类型

如图 2-10 显示了部分的着色器类型，可以看出种类还是相当多的。

注意：大多数的着色类型并不适用于多数的 2D 游戏。

2.1.3 使用纹理

我们最常听说的纹理就是鹅卵石上的纹理，即花纹。说的简单点，纹理就是对象上的“涂鸦”。在 Unity 中，纹理的本质是图像，格式可以是 BMP、JPEG、PNG 等。而纹理是材质的一个属性，对于不使用纹理的材质，引用这个材质的对象就只能显示出单一的颜色，即要么是黄色，要么是绿色。要让对象有更多的颜色，那么对象引用的材质必须使用纹理这个属性。

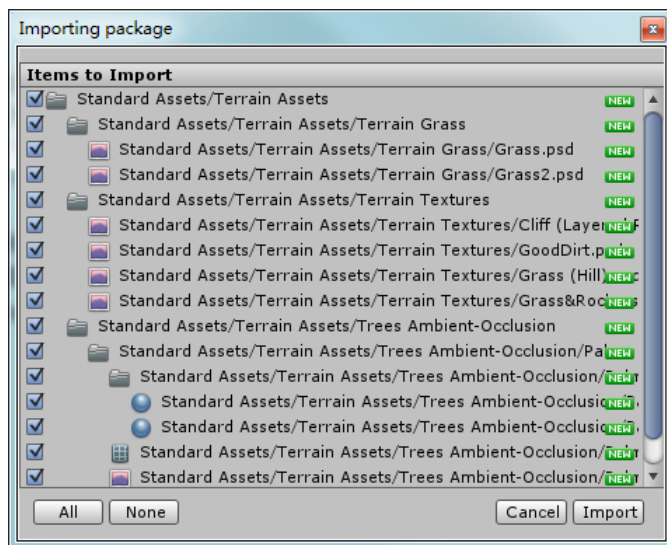


图 2-11 Import package 对话框

要使用纹理，就必须要在项目的资源中添加纹理，我们可以导入 Unity 自带的地形资源（terrain assets）。只需要选择菜单 **Assets|Import Package|Terrain Assets** 命令，然后会出现如图 2-11 所示的导入包（Importing package）对话框，如图 2-12 所示。默认全选，直接点击 **Import** 按钮，导入所有地形资源。



图 2-12 依次打开文件夹



图 2-13 导入项目的地形纹理

在资源面板中会出现一个名为 **Standard Assets** 的文件夹，依次打开 **Terrain Assets**、**Terrain Textures**，如图 2-13 所示。就可以在资源面板中看到被导入到项目中的地形纹理（**Terrain Textures**），如图 2-20 所示。选中资源面板中的材质，在查看器中单击 **Select** 按钮，然后在弹出的 **Select Texture** 对话框中选择悬崖峭壁（**Cliff**）这个纹理，然后材质就引用了这个纹理图像，整个过程如图 2-14 所示。

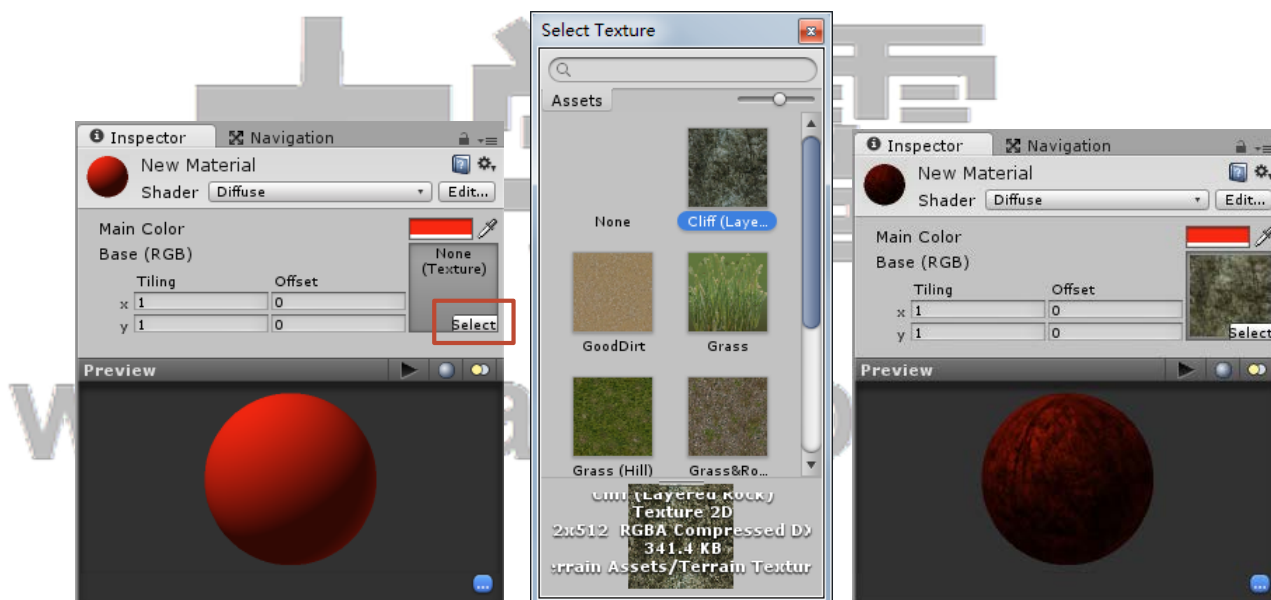


图 2-14 材质引用纹理图像

引用这个材质的 3 个游戏对象的表面也会随之发生变化，感觉就像是对象的表面被纹理绘制了，如图 2-15 所示。

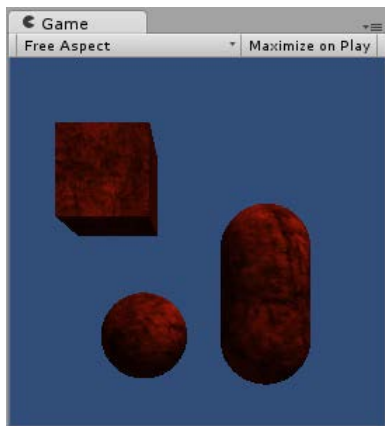


图 2-15 引用同一材质的 3 个游戏对象

注意：一个材质可以使用零个或多个纹理。

2.2 应用于 2D 游戏的材质

不论是 2D 游戏，还是 3D 游戏，都需要使用材质。然而要在 Unity 中使用适合于 2D 游戏的材质却并不那么容易，但好消息是本章介绍了 2 种技巧，可以帮助我们解决这一问题。

2.2.1 缘由

实际上，Unity 提供的材质的着色器类型，都是被设计用于 3D 游戏对象的。例如，在 3D 的场景中，各个游戏对象会在光照下出现面向光的一面被照亮，而背向光的一面变暗的情况，而在 2D 游戏中，并不是十分需要这种效果。

Unity 也没有搭载专用于 2D 游戏的材质，所以要制作 2D 游戏，就需要采用一些技巧和技术实现我们需要的 2D 效果。这个效果是，我们的多个游戏对象在光源的照耀下，不会出现一面亮一面暗的效果。为此本书将使用两种技巧。

2.2.2 技巧一：使用白色的环境光

设想这样一种情况：去掉游戏场景中的所有光源。会发生什么情况？理论上来说，都没有光了，应该是什么都看不到了才对，那么实际情况是什么呢？如图 2-16 所示。

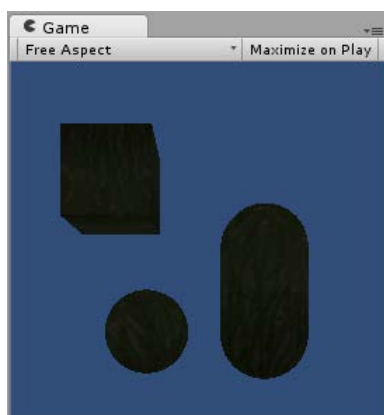


图 2-16 没有光源存在的场景

虽说看的不是很清楚，但 3 个游戏对象的轮廓还是十分清晰的，难道不应该是全黑吗？怎么还能看到轮廓呢？实际上，场景中还存在着一处无处的光，只不过这种光的颜色趋近于黑色，正是因为有这种光源的存在，所以我们才在场景中看到了那 3 个对象。

这种光就是本小节要重点讲解的环境光（Ambient Light）。认识到存在这样一种光源对制作 2D 游戏很有帮助。它给我们提供了这样一个思路：去掉 3D 场景中的所有光源，然后设置环境光的颜色为白色，那么游戏中的对象就会以本来的颜色显示，且不会出现阴影，这样就实现了 2D 的效果。要实现这一思路，可以选择菜单 `Edit>Render Settings` 命令来设置，在 `RenderSettings` 的查看器中修改环境光为白色，如图 2-17 所示。然后 3 个游戏对象的显示效果如图 2-18 所示。

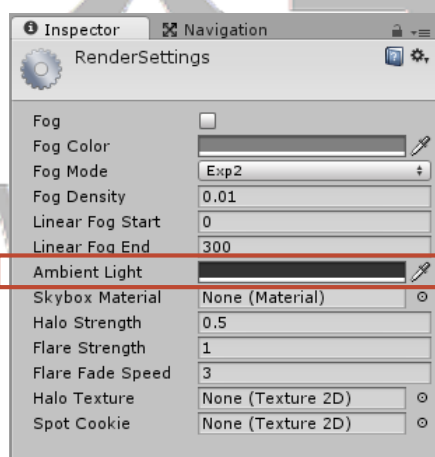


图 2-17 修改环境光的颜色



图 2-18 场景中 3 个游戏对象以本来的颜色显示

这种方法的缺点是环境光在游戏场景中是无处不在的，所以使用此种技巧的 2D 游戏，里面是不应该有光源的。但是，有些 2D 游戏也许需要光源，有了光源然后要有选择的使得某些游戏对象可见，有些不可见。对于此种情况的 2D 游戏，我们就不得不使用第 2 种技巧了。

2.2.3 技巧二：使用光不敏感着色器

如果游戏场景中必须要使用光源，但是又希望场景中的部分对象可以完全不受光源的影

响，那么你就需要一种特定的着色器以及工作流程。

要做到这点需要两步操作，首先选中你希望可以不受光源影响的材质，并且设置它们的着色器类型为自发光（Self-Illumination）中的漫反射（Diffuse）类型，如图 2-19 所示。

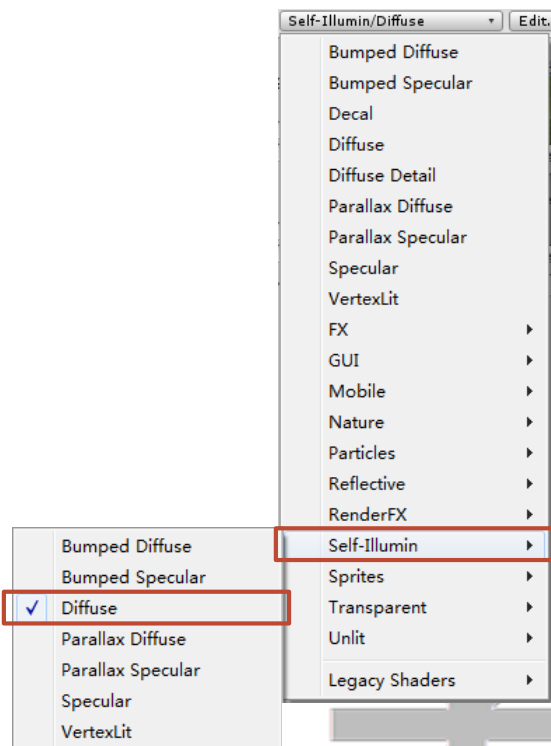


图 2-19 改变材质的着色器类型

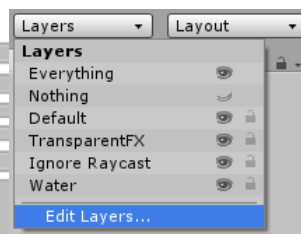


图 2-20 新建一个场景图层

第一步的设置，只是保证了对对象可以不受场景中黑暗的影响，即材质可以自己发光。但是此时材质仍然会受到光照的影响，当光源有颜色时，这样的影响会更加明显。所以有必要完成第二步的操作：给那些希望不受光照影响的游戏对象新建一个场景图层（Scene Layer），然后设置这个图层可以不受场景光源的影响。

具体的操作过程是：选中 Unity 软件右上角的 Layers 选项，然后在下拉列表中选择 Edit Layers...选项，如图 2-20 所示。然后在出现的 Tags & Layers 查看器中给你的图层命名为 Light-Immune，如图 2-21 所示。

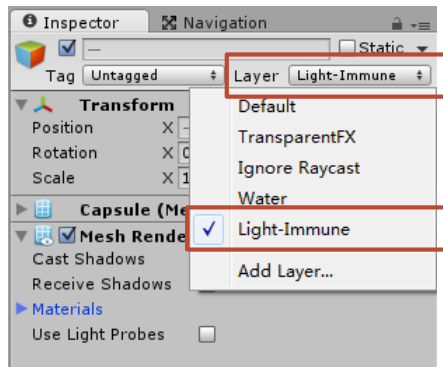
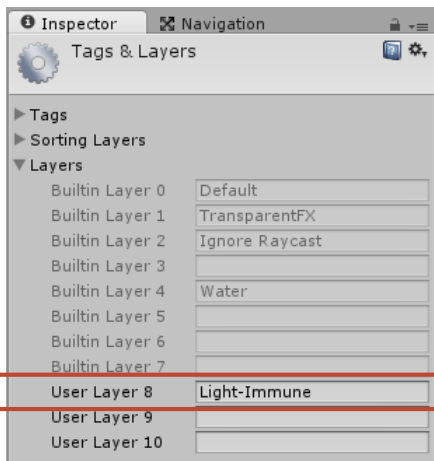


图 2-21 为新的图层命名

图 2-22 改变游戏对象所在的图层

创建了新的图层以后,就可以把希望不受光照影响的游戏对象放到这个图层里。方法是:选中所有希望不受光照影响的游戏对象,然后在查看器的 Layer 下列列表框中选择 Light-Immune (即我们创建的图层),如图 2-22 所示。然后选中场景中的所有光源,在查看器中移除 Culling Mask 属性下拉列表中对 Light-Immune 选项的选择,如图 2-23 所示。

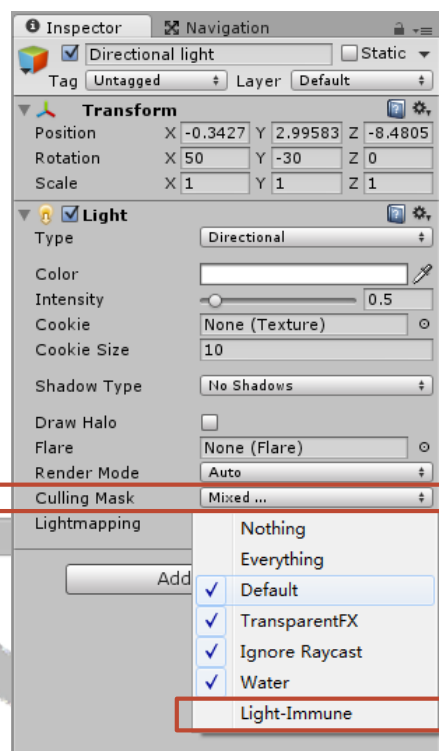


图 2-23 移除图层 Light-Immune 在光源 Culling Mask 属性中的复选

这使得 Light-Immune 图层屏蔽了场景中的光照,即在这个图层里是没有光的,所有的对象都是自己来发光的。应用这种技巧以后,即使场景中有光源,也不再会对游戏对象产生影响了,如图 2-24 所示。

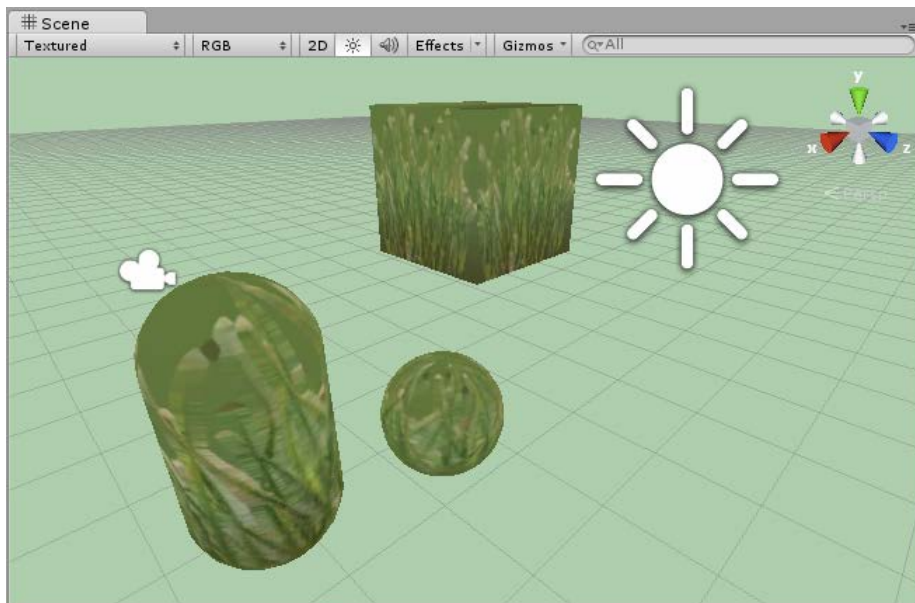


图 2-24 场景中的对象不受光照的影响

2.3 纹理使用规则

Unity 本身无法创建纹理，所以必须导入由其它程序创建的纹理。但是对于导入到 Unity 的纹理，要想被很好的使用，创建和导入的时候最好遵循本节讲解的 3 个规则。

2.3.1 规则 1：分辨率是 2 的次方

每个基于像素点的图像，都可以使用像素点来计量图像的长和宽。像素点就是一个颜色块，多个像素点按照行和列排在一起就构成了图像。用像素点计量图像的宽和高，这也被称为图像的像素点规模（pixel dimensions），通常被用于指代一个图像的分辨率（resolution）。

虽然 Unity 可以导入任何 4096×4096 以内像素点规模的图像，但这并不意味着纹理图像可以和你导入前的一模一样。这是由于分辨率导致的。由于图形硬件（graphics hardware）和实时渲染（real-time rendering）的限制，纹理的分辨率只能是 2 的次方。这就意味着图像的长宽应该是： $2^2=4$ ， $2^3=8$ ， $2^4=16$ ， $2^5=32$... $2^{12}=4096$ 中的任意组合，如 32×256 、 1024×2048 。如果你导入的纹理的分辨率并不是 2 的次方，那么 Unity 将会把纹理缩放或者拉伸到最接近的 2 次方数的大小，而这会影响纹理在场景中的显示效果。

注意：也有这个规则的例外情况，这就涉及到图形用户界面纹理（GUI Textures）。因为有些纹理可以只用在 2D 屏幕上，为一些接口着色，这些接口包括按钮、菜单等。如图 2-25，是《穿越火线》游戏时的一个截图，左上角的圆形就是图形用户界面纹理的一个应用。



图 2-25 游戏中的图形用户界面纹理

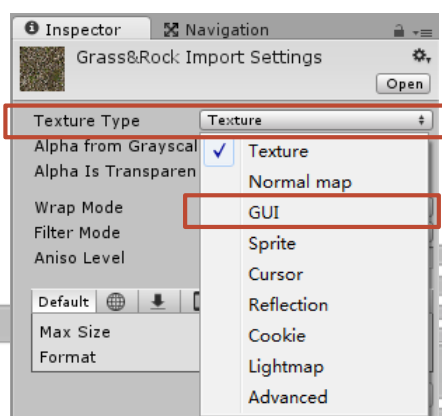


图 2-26 设置纹理的 Texture Type 属性为 GUI

对于导入项目的纹理，选中纹理可以看到查看器中有一个名为 **Texture Type** 的属性，在这个属性的下拉列表选中 **GUI** 时（如图 2-26 所示），这个纹理就被指定用作图形界面了。此时纹理可以是任意的大小，当然，最大不能超过 4096×4096 。

2.3.2 规则 2：保证“质量”

对于一个基于像素点的图像，如 PNG 和 TGA，术语“质量”指的是“数据的完整性”。如果你将一个图像作为纹理导入到项目中，那么你当然希望这个图像在被游戏使用以后，还可以像导入前那样完整显示，至少不会出现清晰度下降、图像显示不完整这样的问题。其实要真正做到“保证质量”，你将不得不面对 2 个敌人：图像重采样（Image Resampling）和有损压缩（Lossy Compression）。幸好我们可以采取措施来迎击这 2 个敌人。

首先，在你创建了图像以后，可以把图像保存为无损（lossy）的格式。例如，你可以保存为 PNG、TGA、TIFF 和 PSD 格式，但是一定不要是 JPG 格式。

其次，这还涉及到重采样（resampling），或者说图像过滤（Image Filtering）的问题。重采样操作决定如何将纹理的像素映射到游戏对象上，然后以怎样的视角显示在屏幕上。而这个重采样操作的设置是在 Unity 里完成的。先选中导入到项目中的纹理，然后在查看器中找到名为 **Filter Mode** 的属性，如图 2-27 所示。

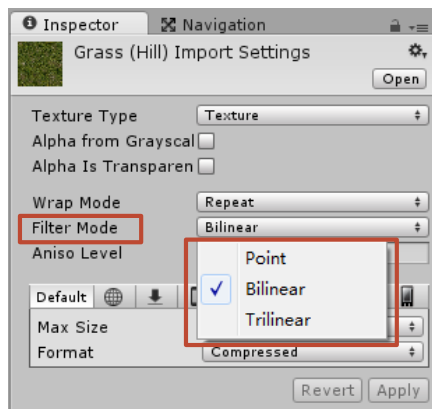


图 2-27 纹理查看器中的 Filter Mode 属性

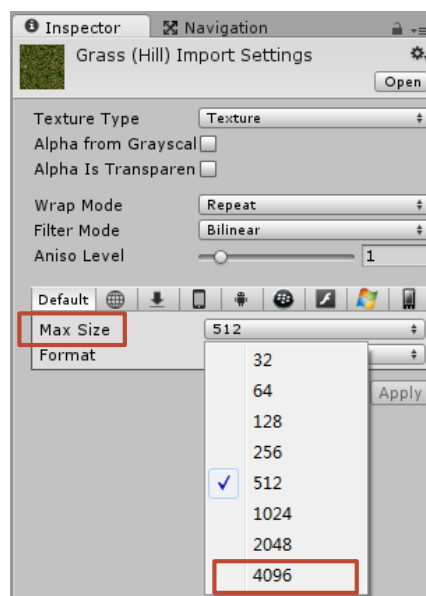


图 2-28 纹理查看器中的 Max Size 属性

过滤模式（Filter Mode）属性有 3 个可选项：点（Point）、双线性（Bilinear）和三线性（Trilinear）。这 3 种过滤模式处理过的纹理图像，图像数据的完整性是依次增加的。Unity 的默认设置是双线性。

最后，涉及到图像的大小。通常情况下，大家习惯设置导入图像的大小为图像的实际大小，或者图像用在游戏中的大小。但是却忽略了 Unity 处理图像的特性：它可以缩小图像的尺寸同时保证数据的完整性，但无法保证放大图像的时候还能保证数据的完整性。因此，鼓励大家以 Unity 可以接受的最大尺寸 4096 来导入图像，然后使用的时候可以在保证数据完整性的情况下任意缩小。大小的设置如图 2-28 所示。

2.3.3 规则 3：增加阿尔法通道（Alpha Channel）

在 Unity 中，阿尔法纹理（alpha texture）用来指代任何包含透明信息的纹理。游戏里应用这种纹理时，部分图像是透明的。要为纹理添加透明信息可以使用阿尔法通道（alpha channel），它通常也被称为一个图像的灰度（grayscale）。如图 2-29 所示，说明了阿尔法通道是如何实现透明效果的。



图 2-29 阿尔法通道实现透明效果的过程

对于一个图像的灰度，黑色区域表示透明，白色区域表示不透明。如果把这个图像作为纹理应用到游戏中时，就只会显示不透明的部分了。

2.4 导入纹理

材质要引用纹理前，需要导入纹理。导入纹理的方法和导入其它资源的方法一样，直接拖拽纹理文件到 Unity 的资源面板即可。

2.4.1 导入纹理时默认设置介绍

导入纹理时，Unity 会应用一系列默认的设置到纹理上，通常这些设置是可以被接受的，但有时还是需要做些调整，为此本小节打算详细介绍每一个默认的设置。选中一个纹理，在查看器中可以看到图 2-30 所示的属性。

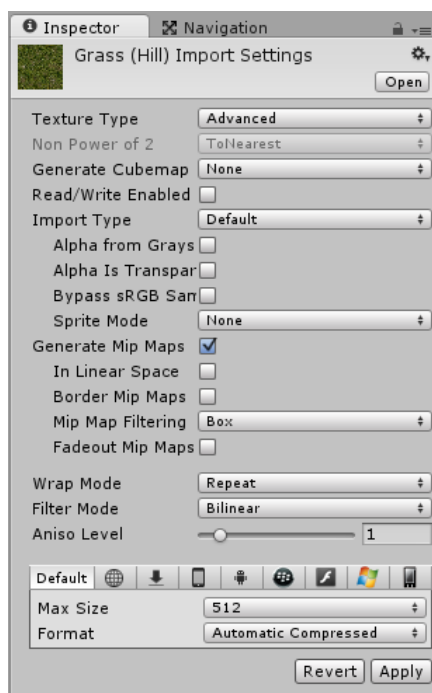


图 2-30 导入纹理时，Unity 应用到纹理上的默认设置

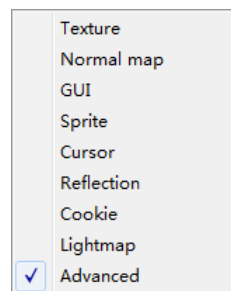


图 2-31 纹理类型

各属性的作用如下：

- ❑ 纹理类型（Texture Type）：这个属性用于指定被选中的纹理的类型。每种类型都决定着 Unity 将在游戏中如何处理这种纹理。全部的纹理类型如图 2-31 所示。对于 2D 游戏，类型通常设置为高级的（Advanced），默认的纹理类型纹理（Texture）也可以，只不过这种类型会要求导入的纹理的尺寸必须是 2 的次方，否则 Unity 会自动调整纹理尺寸为最接近的 2 的次方的大小。“高级”纹理类型会对导入的纹理设置更多的控制属性。
- ❑ 生成立方体贴图（Generate Cubemap）：保留默认设置，即 None。
- ❑ 导入类型（Import Type）：保留默认设置，即 Default。
- ❑ 生成 Mip Maps（Generate Mip Maps）：这个属性的复选应该取消，因为这个属性会依据纹理与摄像机在 Z 轴上的距离修改纹理数据的完整性，但对于 2D 游戏，这个距离通常是不会改变的，所以这个属性就没有必要设置。
- ❑ 循环模式（Wrap Mode）：这个属性控制纹理如何“着色”对象，可选项有 2 个：重复（Repeat）和截断（Clamp）。当纹理“着色”的对象大于纹理时，前者会像瓷砖一样重复“铺满”整个对象；后者则会被拉伸，直到“盖满”整个对象。
- ❑ 各向异性级别（Aniso Level）：当视线与纹理呈斜角时，纹理在游戏中的清晰程度。属性值的范围是 0~9，数字越大越清晰。例如，我们站在房间里看天花板，或者地板的效果。对于 2D 游戏而言，如果这个斜角是 0 度，即平时时，就不再显示纹理了。这个属性的默认值为 1。
- ❑ 最大尺寸（Max Size）：指定纹理显示时的最大尺寸。
- ❑ 格式（Format）：这个属性进一步设置了纹理的“质量”和性能。对于包含透明信息的纹理，这个属性应该被设置为 ARGB 32 bit，或者 RGBA 32 bit。

2.4.2 含有透明信息的纹理

默认情况下，材质的着色器类型是漫反射（Diffuse）。如果这种材质引用一个有透明信息的纹理，纹理将全部显示，就像是没有透明信息那样。所以，要想纹理的透明信息起作用，必须让材质使用特定的着色器类型。下面使用一个示例演示如何在游戏中使用这种纹理。

第一步：导入阿尔法纹理（Alpha Texture）

将阿尔法纹理导入到项目中，可以直接将纹理文件拖拽到项目的资源面板，不修改 Unity 对纹理的默认设置，如图 2-32 所示。



图 2-32 将纹理导入到项目中，保留默认设置



图 2-33 使用特定着色器类型的材质引用纹

理

第二步：为材质设置合适的着色器

创建一个新的材质，设置它的着色器为 Unlit 组中的 Transparent CutOut。这个着色器是轻量级的，适合用于透明信息分别是“完全透明”和“完全不透明”的情况。如果透明信息介于这两者之间，可以使用着色器：Unlit 组中的 Transparent。让新的材质引用前面导入的纹理，此时材质的属性信息以及预览图如 2-33 所示。

第三步：创建对象，引用材质

在场景中创建一个平面对象，让这个对象引用前面创建的材质，然后材质引用的纹理就会显示在平面的表面，如图 2-34 所示。



图 2-34 3D 场景中被材质着色的平面

大学霸

www.daxueba.net