Unity 4.x

游戏开发技巧集锦

（内部资料）

大学霸

www.daxueba.net


大学霸

前　言

Unity 是一款世界知名的游戏开发工具，也是一款全面整合的专业游戏引擎。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，如 Windows、Linux、Mac OS X、iOS、Android、Xbox 360、PS3、WiiU 和 Wed 等。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。

强大的工具还需要灵活的应用。现在的游戏种类众多，其中的声光效果更是精彩炫目。作为游戏开发初学者，往往被别人的游戏效果和功能所惊叹，但往往又为自己的游戏所汗颜。

本书分析世界各类知名游戏，如《仙剑奇侠传》、《红警》、《使命召唤》、《穿越火线》、《劲舞团》、《极品飞车》、《斗地主》、《植物大战僵尸》、《天天跑酷》等。从这些游戏中选择大量经典应用功能和特效进行讲解，如：

《极品飞车》的后视镜功能
《红警警戒》的士兵巡逻功能
《荣誉勋章》的罗盘功能
《拳皇》的倒计时功能
《超级玛丽》的消失文字效果
《星际争霸》的士兵响应效果
.....

相信读者从中学到的将不只是各种特效的实现方法，还会从中感受到无限的成就感和欢乐。

1.学习所需的系统和软件

- ❑ 安装 Windows 7 操作系统
- ❑ 安装 Unity 4.5.2

2.学习建议

大家学习之前，可以致信到 XXXXXXXXX，获取相关的资料和软件。如果大家在学习过程遇到问题，也可以将问题发送到该邮箱。我们尽可能给大家解决。

目录

第 3 章　材质的应用

游戏中，大多数材质的应用都离不开纹理，而纹理本身是图片。所以，在学习本章时，最好在电脑上安装一个可以编辑图片的功能强大的软件，如 Photoshop 和 GIMP。本章使用的是前者。另外，有些纹理需要包含 Transparents 通道，所以图片最好保存成可以存储相应信息的格式，例如，PSD 和 TGA。

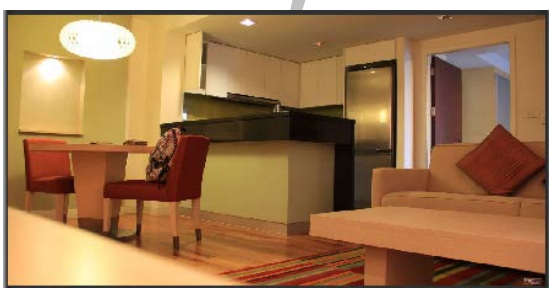
3.1　创建反射材质

生活中，反射材质（reflective material）的实例有很多。例如，水面、光滑的金属、镜子等物体都可以反射其它的物体，如水中的倒影、镜子里的人物，如图 3-1 所示。

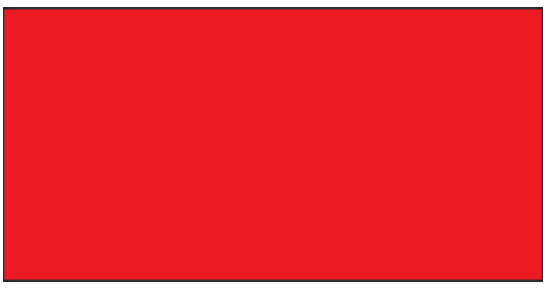
图 3-1　图中的金属球，以及反射出的周围环境

在 Unity 里，可以模仿出这种反射效果。只要使用具有反射特性的着色器，然后再稍做配置即可。具体的操作过程如下：

（1）准备两张纹理图，其中一张纹理图需要包含透明信息，也就是有 Alpha 通道，然后导入到项目中。本节使用的两张纹理，如图 3-2 所示。



普通纹理图



包含透明信息的纹理

图 3-2　导入两张纹理图

（2）选中不包含透明信息的纹理，然后在 Inspector 视图里，做如下修改：

- 设置 Texture Type 属性为 Reflection；
- 设置 Mapping 属性为 Cylindrical；

然后单击 Apply 按钮。如此设置，说明了此纹理将被用做圆柱体的反射图。对纹理的设置以及设置后的纹理如图 3-3 所示。

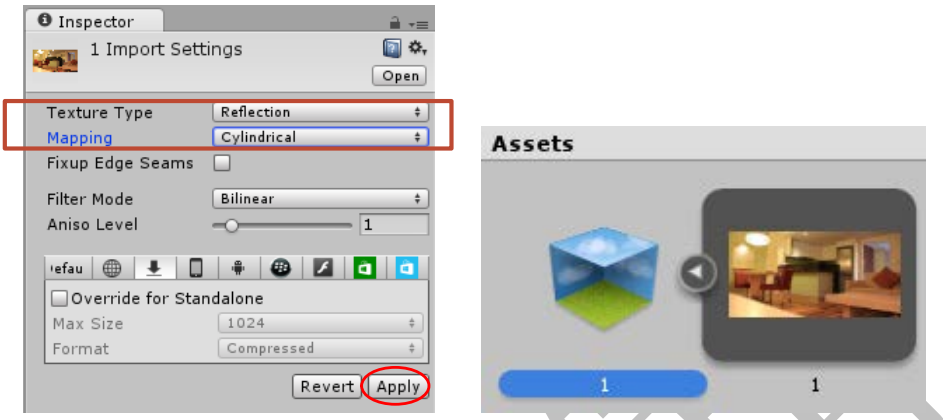


图 3-3　在 Inspector 视图对纹理的设置以及设置后的纹理

（3）在 Project 视图里，创建一个材质，并命名为 Mat_reflect，选中它然后在 Inspector 视图做如下设置：

- 设置 Shader（着色器）属性为 Reflective/Specular；
- 设置 Base(RGB) Gloss(A)为包含透明信息的纹理；
- 设置 Reflection Cubemap 为普通纹理；

如图 3-4 所示。

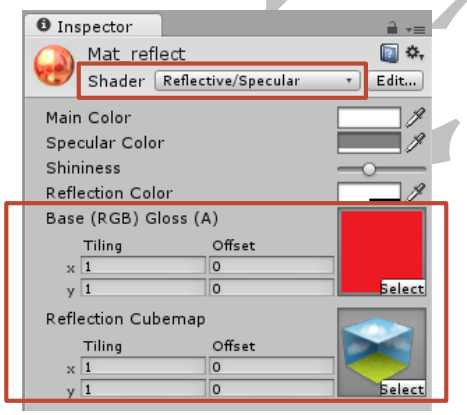


图 3-4　在 Inspector 视图里

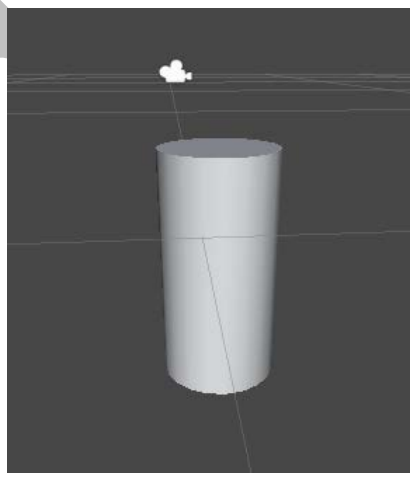


图 3-5　Scene 视图里的 Cylinder 对象

（4）在 Hierarchy 视图里，添加 Cylinder 对象。在 Scene 视图里看到的 Cylinder 如图 3-5 所示。它此时没有反射出周围的任何物体。

（5）拖动 Mat_reflect 材质到 Cylinder 对象上，如图 3-6 所示，效果就大不相同，它即显示出了本身的颜色，又反射出了周围的物体。

提示：纹理含有的透明信息，越透明所反射的物体所成的像越清晰。图 3-7，调低透明度以后，反射的图像暗了，清晰度也下降了。

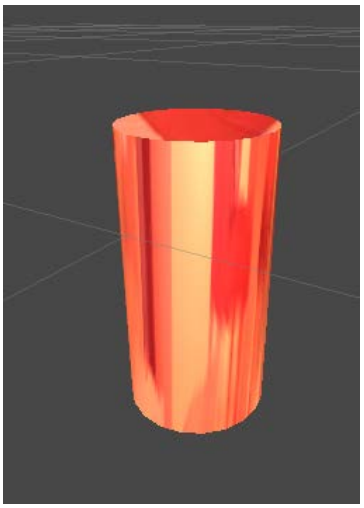


图 3-6　正在反射出周围物体的 Cylinder 对象

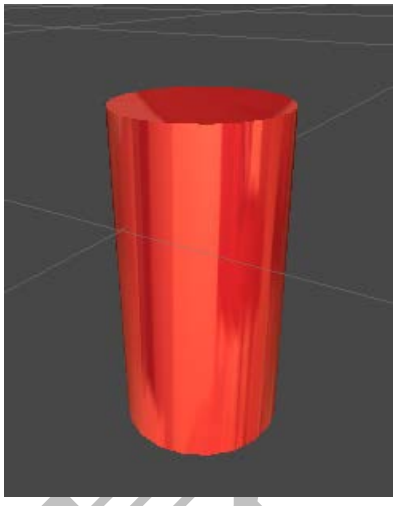


图 3-7　调低透明度以后，反射成像的效果

3.2　创建自发光材质

自发光材质（self-illuminated material）是指自己会发光的材质。生活中与之相似的例子，就是液晶显示屏上显示的信息，文字信息本身是发光的，如图 3-8 所示。

图 3-8　自己发光来显示文字信息的液晶显示屏

3.2.1　创建并配置材质

在 Project 视图里，创建一个材质，并命名为 LCDMaterial，选中它然后在 Inspector 视图里修改 Shader 属性为 Self-Illumin/Diffuse，如图 3-9 所示。

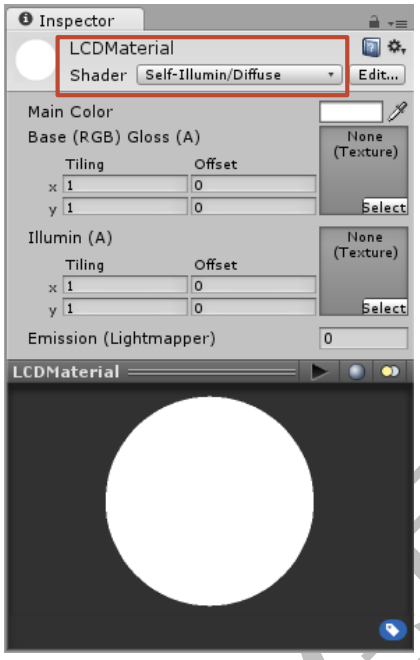


图 3-9　创建材质，并修改材质的 Shader 属性

3.2.2　制作应用于发光材质的纹理

材质配置好了，接下来就该制作纹理了，因为材质需要依靠纹理才能显示信息。本小节就来说明，使用 PhotoShop 制作应用于发光材质的纹理的方法。具体的步骤是：

（1）在 PhotoShop 内新建一个纹理，要求是：

- 命名为 LCDText；
- 宽度、高度都设置为 256，单位是像素；
- 分辨率设置为 72，单位是像素/英寸；
- 颜色模式设置为 RGB；

如图 3-10 所示。

图 3-10　新建一个纹理时，初始属性值的设置

（2）设置此纹理的背景为黑蓝色（R：8，G：16，B：99），并命名此图层为 background。接着在此纹理上，使用蓝色（R：8，G：90，B：231）写入文字信息，本示例写下的是 LCDText，如图 3-11 所示。

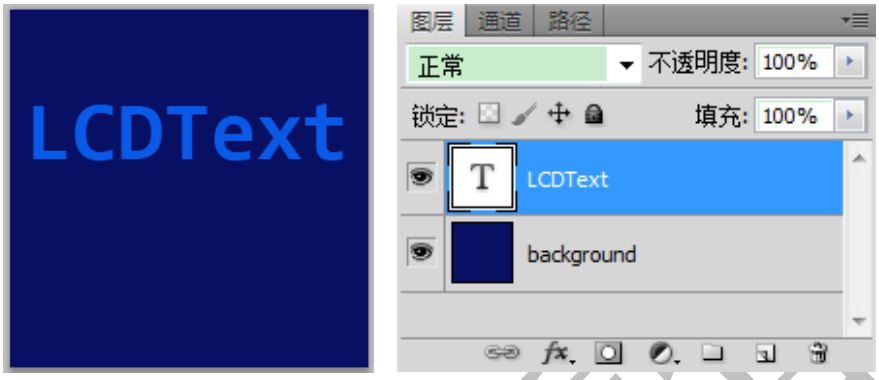


图 3-11　黑蓝色背景上写下文字信息的纹理

（3）复制 LCDText 和 background 图层，然后分别命名为 AlphaLCDText 和 Alphabackground。修改前者的字体颜色为白色，后者的背景色为黑色，如图 3-12 所示。

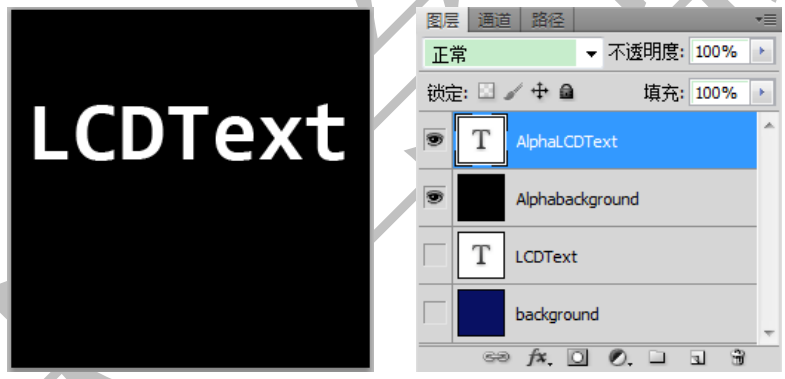


图 3-12　复制图层，并修改各自的颜色

（4）合并 LCDText 和 background 图层，以及 AlphaLCDText 和 Alphabackground 图层，如图 3-13 所示。

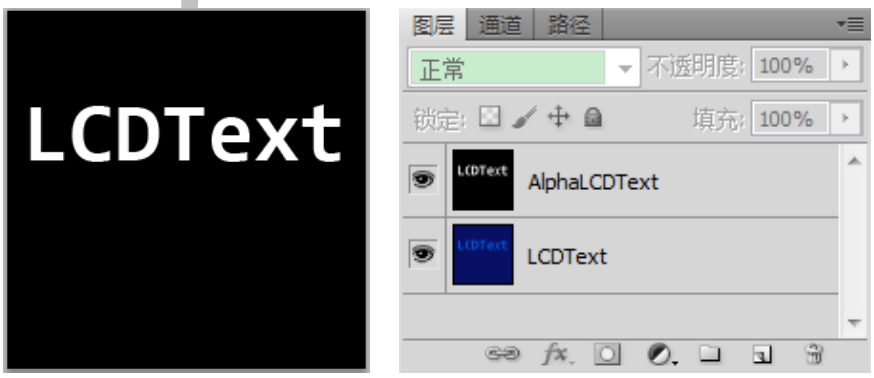


图 3-13　合并图层

（5）选中 AlphaLCDText 图层，然后依次按下 Ctrl+A（全选快捷键）和 Ctrl+C（复制快捷键）。从图层窗口切换到通道窗口，并新建一个通道，默认的名称是 Alpha 1。选中新

建的通道后，按下 Ctrl+V（粘贴快捷键），如图 3-14 所示，此过程为纹理添加了透明信息。透明信息的含义是，文本位置处完全透明，背景完全不透明。

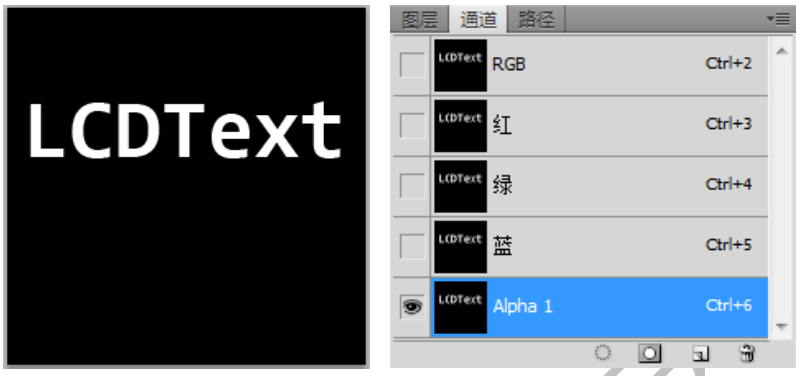


图 3-14　Alpha 1 通道

（6）切换到图层窗口，删除名为 AlphaLCDText 的图层后，含有透明信息的纹理就制作好了。以 PSD 格式保存，并命名为 LCDTexture。

3.2.3　效果展示

将上一小节制作的纹理导入到项目中，然后在 Project 视图里选中 LCDMaterial 材质，在 Inspector 视图里设置材质的 Base(RGB)Gloss(A)和 Illumin(A)属性为导入的纹理，如图 3-15 所示。

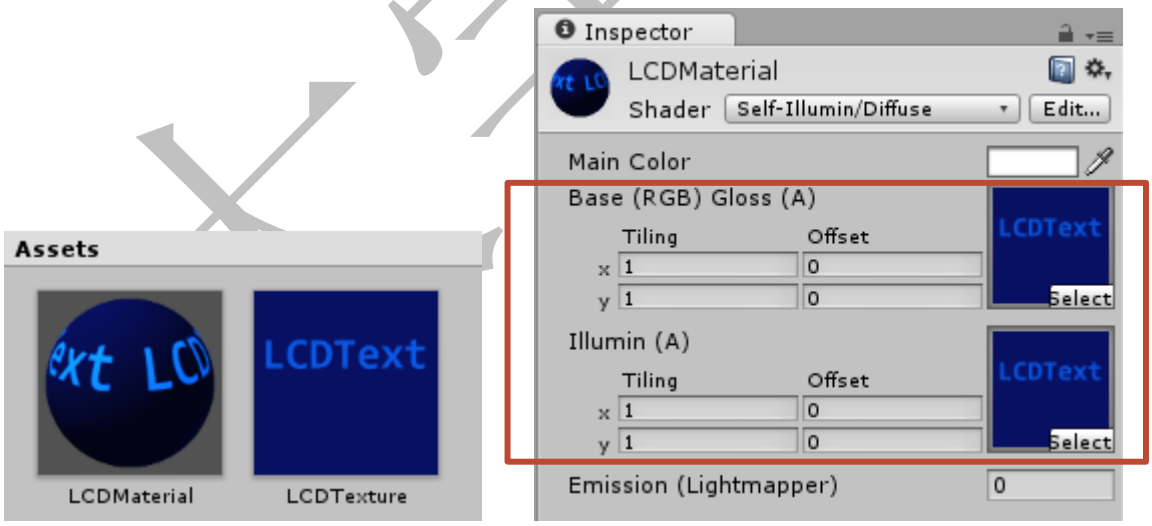


图 3-15　设置材质的纹理属性

此时材质的预览窗口，展示的就是发光字体的显示效果。纹理指定位置越透明，字体越亮；反之，字体会变暗。如图 3-16，对比了字体透明度不同时，发光的效果。

高透明度的字体　　低透明度的字体

图 3-16　不同透明度字体的发光效果比较

3.3　创建部分光滑部分粗糙的材质

生活中，有类物体的表面既有光滑的部分，又有粗糙的部分，例如丽江的石板路，如图 3-17 所示，石板的表面本来是粗糙的，但是在石板上面走的人多了，石板的一部分就变得光滑了。有时，游戏为了显得更加逼真，就需要模拟这样一种材质。

图 3-17　兼具光滑和粗糙表面的丽江石板路

要制作部分光滑部分粗糙的材质，需要用到两种资源：拥有镜面着色器的材质和模拟了现实状况的纹理。

3.3.1　创建并配置材质

在 Project 视图里，创建一个材质，并命名为 rustyMetalMaterial，选中它然后在 Inspector 视图里修改 Shader 属性为 Specular，如图 3-18 所示。

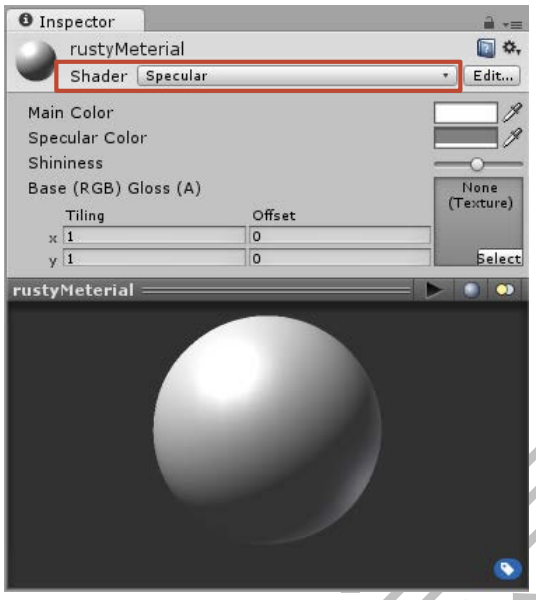


图 3-18　设置材质的 Shader 属性

在此种 Shader 属性下，会出现两个重要的属性：

- Specular Color：可以调节镜面所反射的光的颜色。
- Shininess：可以调节镜面所反射的光的强度。

3.3.2　制作兼具光滑和粗糙效果的纹理

本小节使用的纹理，模拟的是生了锈的金属表面，部分光滑部分因为有铁锈而显得粗糙，如图 3-19 所示。本小节打算为这个纹理做些简单的修饰：添加透明度的信息；以及在纹理上写些文字，用来衬托出光滑和粗糙位置处的不同效果。

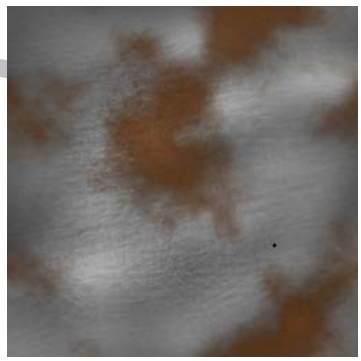


图 3-19　拥有生锈的金属表面效果的纹理

使用 PhotoShop 打开此纹理，然后依次做下面的操作：

（1）在通道窗口下，新建一个通道，使用默认的名称 Alpha 1。拷贝蓝色通道上的图像到 Alpha 1 通道，并设置 Alpha 1 通道上图像的亮度和对比度，如图 3-20 所示。通过此种方式，设置了这样的透明信息：铁锈处是完全不透明的，而光滑处是完全透明的。

图 3-20　设置 Alpha 1 通道上图像的亮度和对比度

（2）切换到图层视图，然后为此图像添加黄色的文字，文本内容随意。然后设置文字图层为叠加效果，不透明度为 80%，如图 3-21 所示，对比了文字设置前后，效果的差异。这样的话，文字看起来就是写在金属表面上，而非铁锈上了。

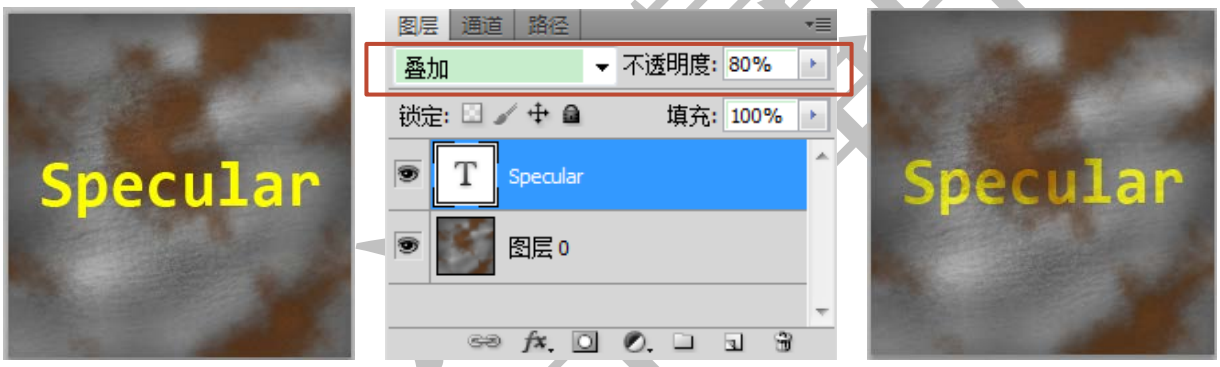


图 3-21　设置文字图层为叠加效果，不透明度为 80%

（3）合并文字图层和图像图层后，存储此文件，接下来会导入到 Unity 中使用。

3.3.3　效果展示

将上一小节制作的纹理导入到 Unity 中，然后选中 Project 视图里的 rustyMeterial，在 Inspector 视图里设置它的 Base(RGB)Gloss(A)属性为导入的纹理，最后就可以在 Inspector 视图的预览窗口中看到效果了，如图 3-22 所示。

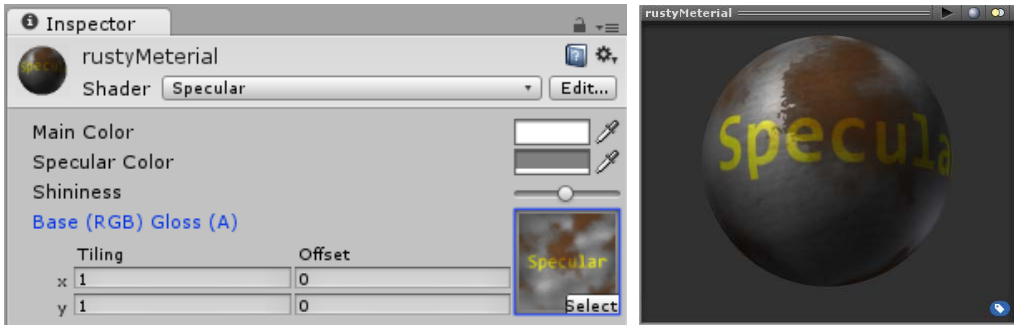


图 3-22　材质效果展示

3.4 创建透明的材质

生活中不乏透明或者半透明的事物。例如，擦的十分干净的玻璃，看起来就是透明的；一些塑料卡片，看起来就是半透明的，如图 3-23 所示。在 Unity 中，可以创建模拟了透明效果的材质，这也是本节主要讲解的内容。

图 3-23 半透明的卡片

3.4.1 创建并配置材质

在 Project 视图里，创建一个材质，并命名为 TransMaterial，选中它然后在 Inspector 视图里修改 Shader 属性为 Transparent/Diffuse，如图 3-24 所示。

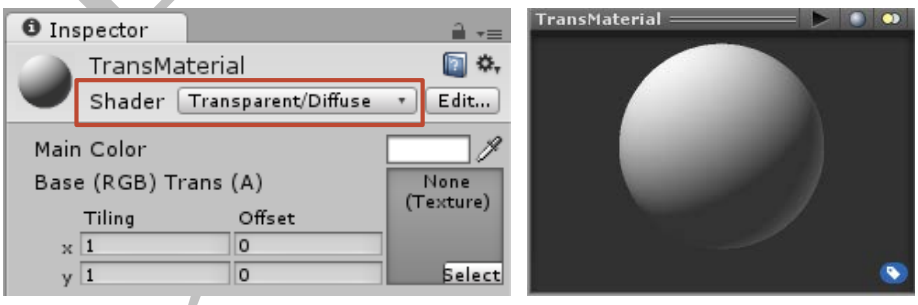


图 3-24 设置材质的 Shader 属性

3.4.2 制作有透明效果的纹理

选择一张有趣的图片，本小节将在 PhotoShop 里处理这张图片，简单来说就是为此图片添加透明度的信息。示例选择的图片如图 3-25 所示。

图 3-25　示例图片

（1）在通道窗口中，新建一个通道，使用默认的名称 Alpha 1。

（2）选中 RGB 通道，然后使用 PhotoShop 中的魔棒工具，圈选出图片的背景区域。再选中 Alpha 1 通道，此时背景的轮廓线在此通道下依然可见。使用油漆桶工具为背景轮廓填充黑色，如图 3-26 所示。记录的透明度信息是：动画角色完全不透明、其余完全透明。

图 3-26　为图像添加透明信息

（3）存储此文件，并导入 Unity 以供接下来的使用。

3.4.3　效果展示

选中 Project 视图里的 TransMaterial，然后在 Inspector 视图里修改 Base(RGB)Trans(A) 属性为上一小节导入的纹理。接着在游戏场景中新建一个立方体，将 TransMaterial 材质拖动到此立方体对象上，在场景中就可以看出效果，如图 3-27 所示，与一个没有使用透明材质的球体做了对比。

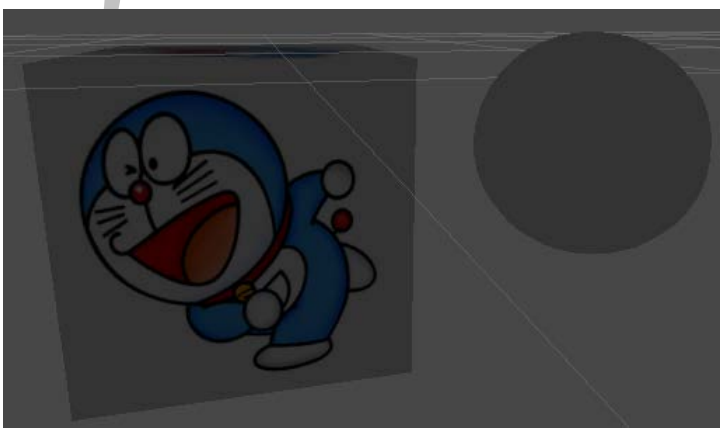


图 3-27　透明材质效果对比

3.5 使用 cookie 类型的纹理模拟云层的移动

现实生活中，当阳光直射大地，而天空中又有很多云时，云层的影子总是会投射在大地上，风吹着云层移动，影子也跟着运动，如图 3-28 所示。

图 3-28 天空中的云朵与大地上的影子

要在游戏中，模拟与之类似的大气现象时，就需要使用 cookie 类型的纹理。

3.5.1 制作云层效果的纹理

本小节将使用 PhotoShop 绘制有云层效果的纹理图，然后为其添加透明度信息。具体操作过程如下：

（1）使用 PhotoShop 创建 512×512 像素大小的图。

（2）在 PhotoShop 内，单击【滤镜】|【渲染】|【云彩】命令，即可在瞬间完成云层的绘制，如图 3-29 所示。

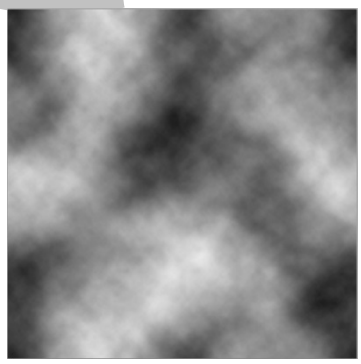


图 3-29 使用 PhotoShop 绘制云层

（3）全选（Ctrl+A）并复制（Ctrl+C）此图像，切换到通道窗口下，新建通道，使用默认的名称 Alpha 1，选中新建的通道，然后粘贴（Ctrl+V）。这样透明度的信息也就添加完了。

（4）存储此云层纹理，然后导入到 Unity。

3.5.2 在 Unity 中完成的准备工作

为了模拟云层的移动效果，需要想游戏场景中添加相应对象，并做些简单设置，具体步骤如下：

（1）在 Project 视图里，选中导入的云层纹理，然后在 Inspector 视图里设置下列属性值，如图 3-30 所示。

- Texture Type 为 Cookie；
- Light Type 为 Directional；

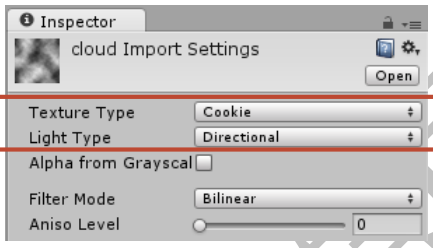


图 3-30 设置云层纹理的属性

（2）单击 GameObject|Create Other|Terrain 和 Directional Light 命令，为游戏添加地形（Terrain）和方向光源（Directional Light）对象。

（3）选中 Directional Light，然后在 Inspector 视图里，修改光源下列属性，如图 3-31 所示。

- Position 的 X、Y、Z 均设置为 0；
- Rotation 的 X、Y、Z 设置为 90、0、0；此时光线会与地形平面垂直，如图 3-32 所示，这是为了避免阴影扭曲。
- Cookie 设置为云层纹理；
- Cookie Size 设置为 200；此属性控制照射在地形上的云层的密度，值越大密度越小。
- Shadow Type 设置为 No Shadows；

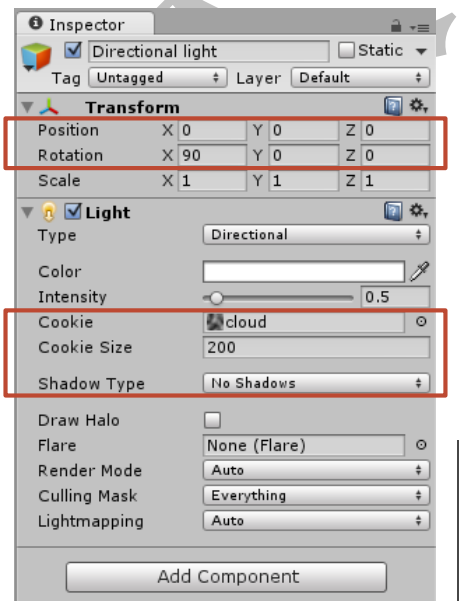


图 3-31 修改光源的属性　　图 3-32 光线与地形平面垂直

3.5.3 编写控制云层移动的脚本

光是将云层的阴影投射在地形平面上，还看不出什么效果，所以本小节打算编写一个脚本，用于控制云层的移动，这样的话效果会更好些。在 Project 视图下，创建一个 C#脚本，并命名为 MovingShadows。打开此脚本，写入下面的代码：

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class MovingShadows : MonoBehaviour
05  {
06      public float windSpeedX;                    //在 X 轴方向上的速度
07      public float windSpeedZ;                    //在 z 轴方向上的速度
08      public float lightCookieSize;               //直线光源 Cookie Size 属性的值
09      private Vector3 initPos;
10      //脚本被初始化时，调用此函数
11      void Start ()
12      {
13          initPos = transform.position;
14      }
15      //运行游戏时，每帧都调用此函数
16      void Update ()
17      {
18          //在 X 轴方向，移动云层
19          if (Mathf.Abs(transform.position.x) >= Mathf.Abs(initPos.x) + lightCookieSize)
20          {
21              Vector3 pos = transform.position;
22              pos.x = initPos.x;
23              transform.position = pos;
24          }
25          else
26          {
27              transform.Translate(Time.deltaTime * windSpeedX, 0, 0, Space.World);
28          }
29          //在 Z 轴方向，移动云层
30          if (Mathf.Abs(transform.position.z) >= Mathf.Abs(initPos.z) + lightCookieSize)
31          {
32              Vector3 pos = transform.position;
33              pos.z = initPos.z;
34              transform.position = pos;
35          }
36          else
37          {
38              transform.Translate(0, 0, Time.deltaTime * windSpeedZ,Space.World);
39          }
40      }
41  }
```

此脚本将被赋予方向光源。游戏开始运行时，脚本 11 行的 Start()函数被执行，获知当前方向光源的位置；游戏运行时，16 行的 Update()在每帧都被执行的，作用是控制方向光源在 XZ 平面上的移动。

选中被赋予 MovingShadows 脚本的方向光源，在 Inspector 视图里可以看到脚本组件上

的 3 个属性：Wind Speed X、Wind Speed Z 和 Light Cookie Size，如图 3-33 所示。

提示：属性的含义在脚本的注释中有说明。

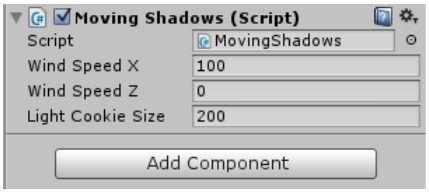


图 3-33　脚本组件上的 3 个属性

3.5.4　效果展示

设置好云层的移动速度，调整好摄像机的视图效果后，就可以运行游戏了。当然了，看到的就是云层在地形表面上移动的场景了，如图 3-34 所示。

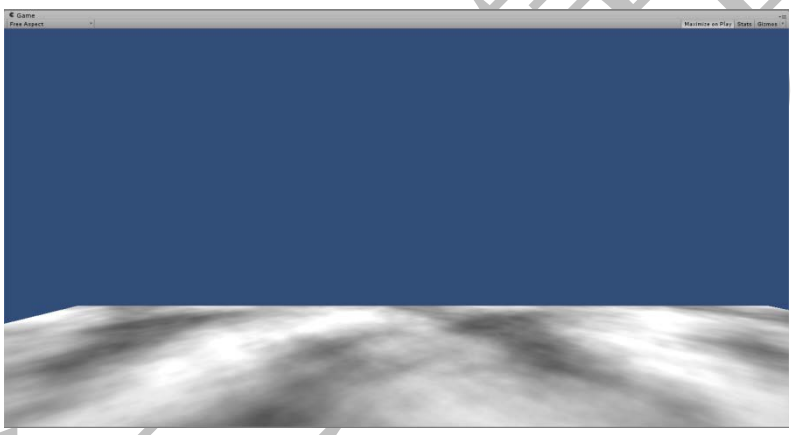


图 3-34　模拟云层移动的效果

3.6　制作一个颜色选择对话框

有些游戏允许玩家自定义角色的一些外观属性，例如颜色，如图 3-35 所示，《赛尔号》这款游戏要求玩家在进入游戏前选择机器人的涂装颜色。

图 3-35　游戏中角色外观颜色的选择

这个功能特性很常见，所以本章就来介绍下，在游戏时，可以弹出颜色选择对话框，并允许玩家自定义游戏对象颜色属性的方法。

（1）在 Project 视图里，创建一个 C#脚本文件，并命名为 ColorSelector。打开此脚本，然后在里面编写下面的代码：

```
using UnityEngine;
using System.Collections;

public class ColorSelector : MonoBehaviour
{
    //公有成员
    //颜色属性中，红色、绿色和蓝色的值
    public float redValue = 1.0f;
    public float greenValue = 1.0f;
    public float blueValue = 1.0f;
    //私有成员
    private bool selectorOn = false;
    //预置的红色、绿色和蓝色的值
    private float redReset = 1.0f;
    private float greenReset = 1.0f;
    private float blueReset = 1.0f;
    //鼠标选中脚本所在的游戏对象，当鼠标弹起时调用此函数
    void OnMouseUp()
    {
        selectorOn = true;
    }
    //绘制颜色选择对话框
    void OnGUI()
    {
        if (selectorOn)
        {
            //绘制颜色条
            GUI.Label(new Rect(10, 30, 90, 20), "Red: " + Mathf.RoundToInt(redValue * 255));
            redValue = GUI.HorizontalSlider(new Rect(80, 30, 256,20), redValue, 0.0f, 1.0f);
            GUI.Label(new Rect(10, 50, 90, 20), "Green: " + Mathf.RoundToInt(greenValue * 255));
            greenValue = GUI.HorizontalSlider(new Rect(80, 50,256, 20), greenValue, 0.0f, 1.0f);
            GUI.Label(new Rect(10, 70, 90, 20), "Blue: " + Mathf.RoundToInt(blueValue * 255));
            blueValue = GUI.HorizontalSlider(new Rect(80, 70, 256,20), blueValue, 0.0f, 1.0f);
            //绘制按钮 Ok
            if (GUI.Button(new Rect(10, 110, 50, 20), "Ok"))
            {
                selectorOn = false;
                redReset = redValue;
                greenReset = greenValue;
                blueReset = blueValue;
            }
            //绘制按钮 Reset
```

```
43                  if (GUI.Button(new Rect(70, 110, 80, 20), "Reset"))
44                  {
45                      redValue = redReset;
46                      greenValue = greenReset;
47                      blueValue = blueReset;
48                  }
49                  //设置对象的外观颜色属性
50                  renderer.material.SetColor("_Color", new Color(redValue, greenValue,
blueValue, 1));
51              }
52              else
53              {
54                  GUI.Label(new Rect(10, 10, 500, 20), "Click the spaceship to change color");
55              }
56          }
57      }
```

使用此脚本，在游戏视图上绘制的颜色对话框如图 3-36 所示。

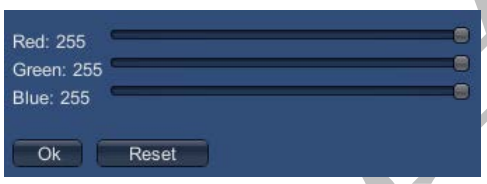


图 3-36　脚本代码在游戏视图上绘制的颜色选择对话框

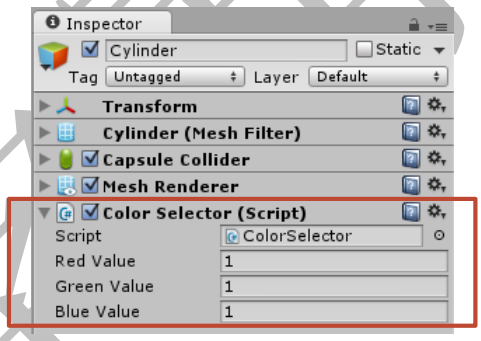


图 3-37　圆柱对象上，脚本组件的各属性

（2）此时可以在游戏场景中，创建任意的游戏对象。既可以是简单的几何体对象，如立方体、圆柱体，也可以是任何导入的复杂游戏对象。只要将前面编写脚本附加到游戏对象上，运行游戏时，在玩家选中游戏对象以后，就可以设置对象的外观颜色属性。

本示例以圆柱体为例演示，赋予它脚本后，在 Inspector 视图上，看到的对应脚本属性如图 3-37 所示。然后运行游戏，用鼠标点击圆柱体后，颜色选择对话框才会出现，如图 3-38 所示，设置圆柱体的外观颜色为浅绿色。

图 3-38　设置圆柱体的外观颜色为浅绿色

3.7 实时合并纹理——拼脸小示例

除了有改变角色外观颜色属性的功能外，很多时候还允许改变其它属性，比如外貌。图 3-39，《脸萌》可以改变角色的发型，及其它外观属性。

图 3-39 《脸萌》，改变角色外观属性

本节将借助于 GUI，实现纹理的合并，并以一个生动的拼脸小示例，说明实时合并纹理的操作方法：

（1）准备 4 张大小一致的纹理，本示例选择的大小是 512×512 像素，其中两张的背景色是透明的，如图 3-40 所示。

图 3-40 准备的 4 张纹理图

为两张背景不透明的纹理命名为 Base_1 和 Base_2，两张的背景色透明的纹理命名为 Decal_1 和 Decal_2。然后将它们导入到 Unity 中

（2）在 Project 视图里，新建一个材质，并命名为 CombinMaterial，选中它然后在 Inspector 视图里修改下列属性，如图 3-41 所示。

- Shader 为 Decal；
- Base(RGB)为 Base_1；
- Decal(RGBA)为 Decal_1；

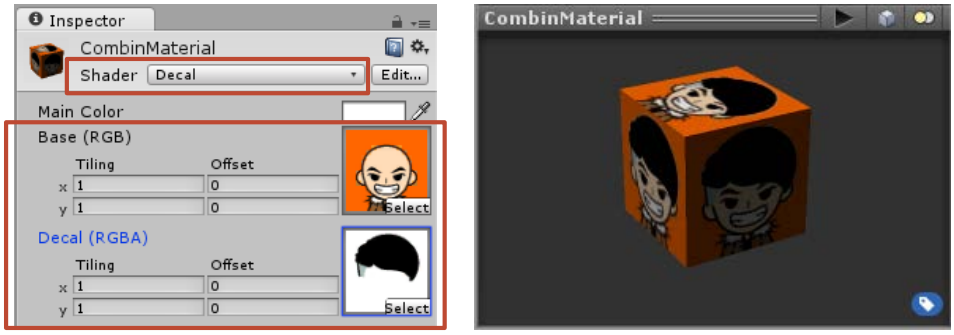


图 3-41　设置材质的各属性，以及预览窗口的效果

（3）在 Project 视图，新建一个 C#脚本，并命名为 SelectTexture，打开它并编写下面的代码：

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class SelectTexture : MonoBehaviour
05  {
06      public Texture2D[] faces;
07      public Texture2D[] props;
08      //绘制界面上的按钮
09      void OnGUI()
10      {
11          //绘制按钮，并在按钮上贴图
12          for (int i = 0; i < faces.Length; i++)
13          {
14              if (GUI.Button(new Rect(0, i * 64, 128, 64),faces[i]))
15                  ChangeMaterial("faces", i);
16          }
17          for (int j = 0; j < props.Length; j++)
18          {
19              if (GUI.Button(new Rect(128, j * 64, 128, 64),props[j]))
20                  ChangeMaterial("props", j);
21          }
22      }
23      void ChangeMaterial(string category, int index)
24      {
25          //改变 Base(RGB)属性上的纹理
26          if (category == "faces")
27              renderer.material.mainTexture = faces[index];
28          //改变 Decal(RGBA)属性上的纹理
29          if (category == "props")
30              renderer.material.SetTexture("_DecalTex", props[index]);
31      }
32  }
```

此脚本将控制所赋予对象上纹理图像的合并，以及其他行为。

（4）在场景中，添加一个 Plane 对象，赋予此对象 CombinMaterial 材质和 SelectTexture 脚本。调节 Plane 对象和 Main Camera 对象的位置，得到的 Scene 和 Game 视图，如图 3-42 所示。

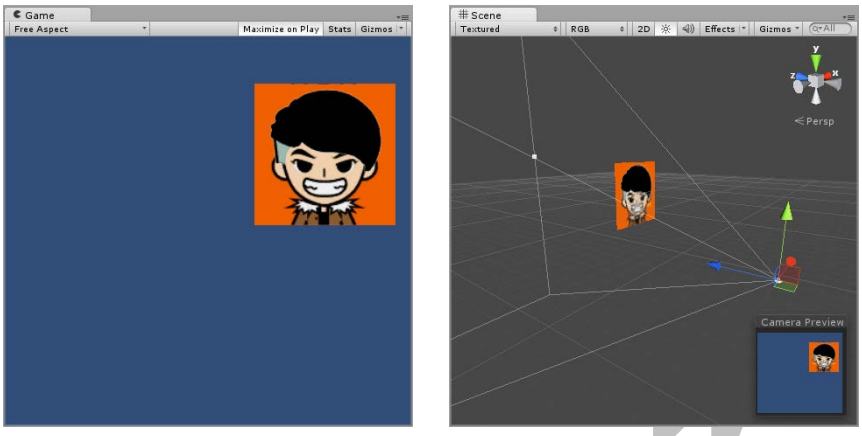


图 3-42　调节 Plane 对象和 Main Camera 对象的位置

（5）在 Hierarchy 视图里选中 plane 对象，然后在 Inspector 视图，查看并修改脚本 SelectTexture 组件的各属性，如图 3-43 所示。

- Faces 中 Size 为 2，Element 0 为 Base_1，Element 1 为 Base_2；
- Props 中 Size 为 2，Element 0 为 Decal_1，Element 1 为 Decal_2；

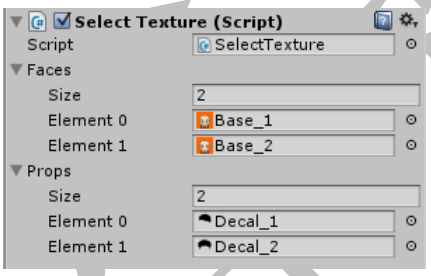


图 3-43　修改 Plane 对象上脚本 SelectTexture 组件的各属性

（6）运行游戏，在 Game 视图的左上角会出现 4 个按钮，右下角会显示合并成的纹理图。单击按钮即可改变合成纹理的子纹理。得到的两个合并纹理如图 3-44 所示。

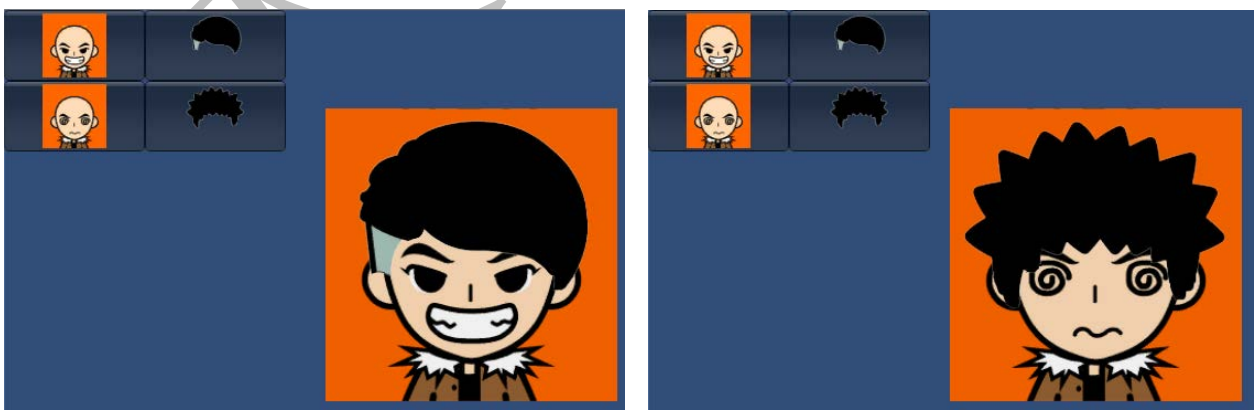


图 3-44　拼脸小示例的效果展示

3.8 创建高亮材质

高亮（Highlighting）这种效果被广泛的使用在各种软件上，它使软件使用者意识到，软件的这部分是可以与自己交互的。游戏中也经常使用这种效果，例如，《红色警戒 2》中，在鼠标触动了菜单项以后，后者会高亮显示，如图 3-45 所示。

图 2-45 高亮显示的游戏菜单

要做到高亮这种效果，可以通过创建高亮材质来实现，实现的具体步骤如下：

（1）在 Project 视图里，新建一个材质，并命名为 HighLight_Mat，选中它然后在 Inspector 视图里修改它的下列属性，如图 3-46 所示。

- Shader 为 VertexLit（顶点光照）
- Bass(RGB)为一张任意的纹理图。尽管纹理图不是必须的，但这会使得高亮的效果更加明显。

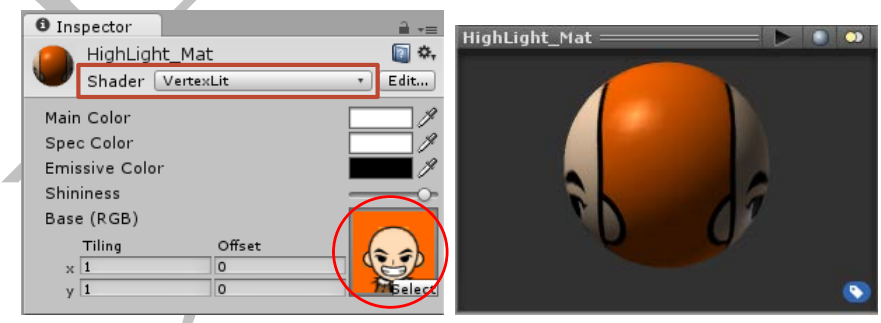


图 3-46 对材质属性的设置

提示：读者可以尝试修改材质的 Emissive Color（发射的颜色）属性，看看效果，因为后面编写的脚本就是通过修改这个属性，进而实现高亮效果的。

（2）在 Project 视图里，新建一个 C#脚本文件，并命名为 HighlightObject，打开它，在里面编写下面的代码：

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class HighlightObject : MonoBehaviour
05  {
06      public Color initialColor;
```

```
07          public Color highlightColor;
08          public Color mousedownColor;
09          private bool mouseon = false;
10          //鼠标进入高亮区域
11          void OnMouseEnter()
12          {
13                mouseon = true;
14                renderer.material.SetColor("_Emission", highlightColor);
15          }
16          //鼠标离开高亮区域
17          void OnMouseExit()
18          {
19                mouseon = false;
20                renderer.material.SetColor("_Emission", initialColor);
21          }
22          //鼠标按下
23          void OnMouseDown()
24          {
25                renderer.material.SetColor("_Emission", mousedownColor);
26          }
27          //鼠标弹起
28          void OnMouseUp()
29          {
30                if (mouseon)
31                      renderer.material.SetColor("_Emission", highlightColor);
32                else
33                      renderer.material.SetColor("_Emission", initialColor);
34          }
35    }
```

脚本选取了 4 个事件，作为触发高亮的条件，分别位于脚本代码的 10、16、22 和 27 行。

（3）为游戏场景添加任意一个对象，这里以立方体对象为例，将新建的材质和脚本赋予此游戏对象。在 Inspector 视图里，设置对象的脚本组件属性，如图 3-47 所示。

- Initial Color：设置对象的初始颜色；
- HighlightColor：对象高亮时的颜色；
- Mousedown Color：鼠标在对象上按下时，对象的颜色；

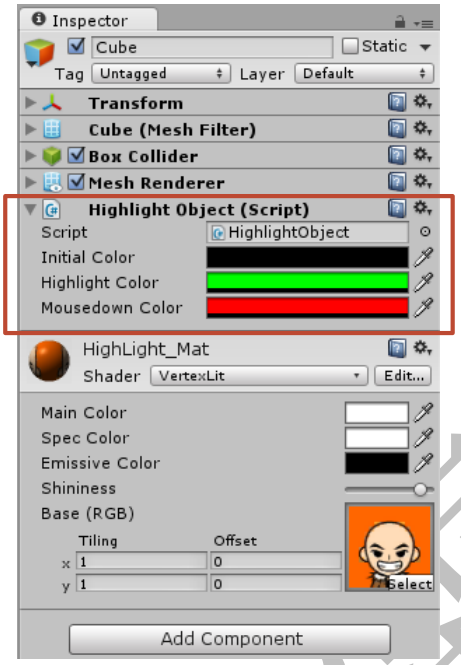


图 3-47　设置对象上脚本组件的属性

（4）运行游戏，鼠标在对象上划过，或者在对象上按下时，对象的颜色会有所不同。也就是说，鼠标划过和在对象上按下时，对象会产生高亮的效果，如图 3-48 所示。

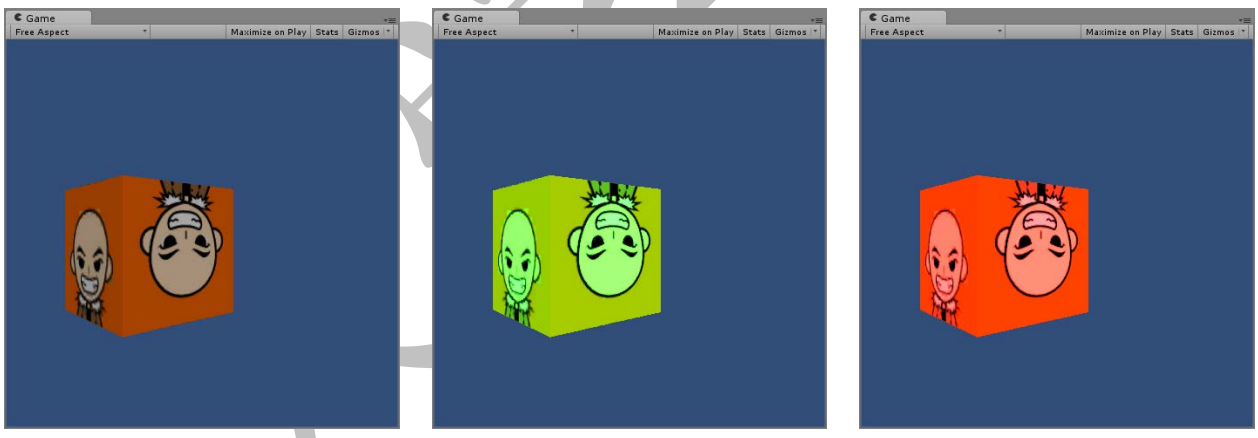


图 3-48　对象默认时、鼠标划过时和鼠标按下时的颜色

3.9　使用纹理数组实现动画效果

有时游戏中会出现类似电视的东西，而它总是在重复着播放一些画面。例如，在《星际争霸 2》中，在游戏视图的右下角，就有这样一个播放动画的小屏幕，如图 3-49 所示。要实现这种效果，可以使用纹理数组。

图 3-49　《星际争霸 2》中播放动画的小屏幕

本节要实现的效果是：在游戏场景的 Plane 对象上，一次播放一组图片，调节播放的速度后就可以模拟成动画的效果了。具体的实现步骤是：

（1）从网络上找一组图片资源，这些图片连着播放最好能显示出动画的效果。本节选取的一组图片如图 3-50 所示。

图 3-50　用于实现动画的一组图片（奔跑的小鹿）

提示：找这样一组图片资源的一种方法可以是，找到一张 GIF 动态图，然后使用 PhotoShop 打开这个图。在图层窗口中，每个图层都会显示这个 GIF 图的一张子图，依次保存每一个图层，即可得到本节所需的一组图片资源，如图 3-51 所示。

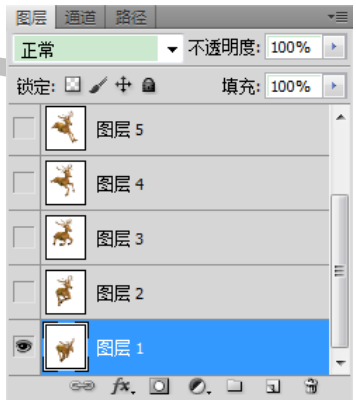


图 3-51　图层窗口中显示出 GIF 图中的所有子图

（2）为游戏场景添加 Plane 对象，调整 Main Camera 与 Plane 的位置，使得前者的视图方向与后者垂直，如图 3-52 所示。

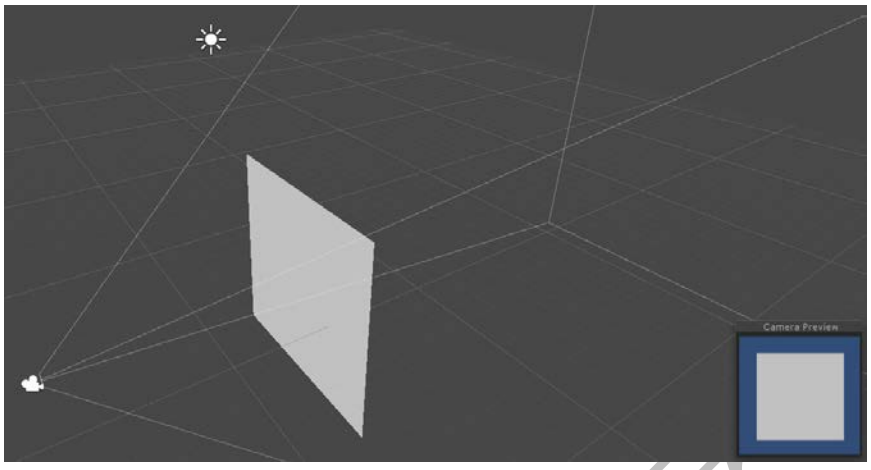


图 3-52　调节 Main Camera 与 Plane 的位置

（3）在 Project 视图里，新建 C#脚本文件，并命名为 AnimatedTexture，打开它并在里面编写下面的代码：

```
using UnityEngine;
using System.Collections;

public class AnimatedTexture : MonoBehaviour
{
    public float frameInterval = 0.9f;                    //控制图片的播放速度
    public Texture2D[] imageArray;                        //存储多个纹理的数组
    private int imageIndex = 0;
    //脚本初始化时，调用此函数，比 Start()更早被调用
    private void Awake()
    {
        //检查数组中是否有图片
        if (imageArray.Length < 1)
            Debug.LogError("no images in array!");
        else
            StartCoroutine( PlayAnimation() );
    }
    //播放动画
    private IEnumerator PlayAnimation()
    {
        while( true )
        {
            ChangeImage();
            //等待一定的时间
            yield return new WaitForSeconds(frameInterval);
        }
    }
    //改变图片
    private void ChangeImage()
    {
        imageIndex++;
        imageIndex = imageIndex % imageArray.Length;
        Texture2D nextImage = imageArray[ imageIndex ];
        renderer.material.SetTexture("_MainTex", nextImage);
    }
}
```

（4）将此脚本赋予 Plane 对象，修改对象上脚本组件的各属性，如图 3-53 所示。

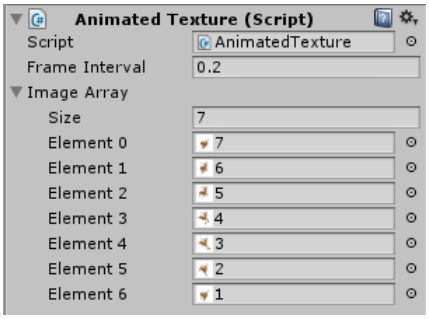


图 3-53　脚本组件中的各属性

图片最好依次赋值到 ImageArray 数组里，因为脚本代码所实现的功能是依次播放它们，如果顺序有误的话，播放出来的动画就会显得十分怪异了。设置好以后，运行游戏，即可在 Plane 对象上看到动画的播放过程，如图 3-54 所示。如果播放的太慢，可以适当调小脚本组件中，Frame Interval 属性的值。

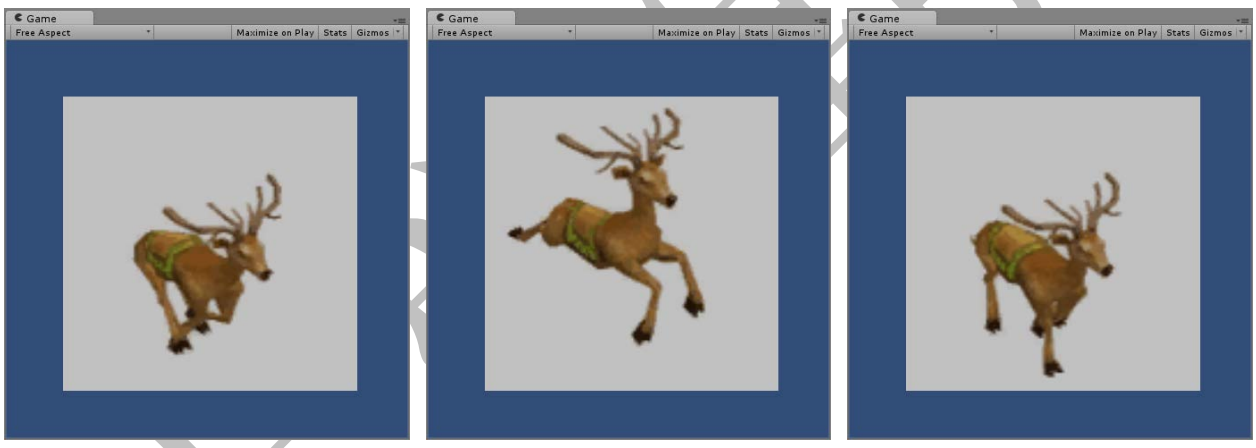


图 3-54　纹理数组实现动画播放的效果

3.10　创建一个镂空的材质

有时游戏中会出现一些镂空的游戏对象，也就是说玩家不光可以看到对象的表面，还还可以看到对象的内部。例如，角色在外面看到一栋大房子，而房子里面具体有什么，则可以通过窗口来查看，这就是镂空的一种效果，如图 3-55 所示。本节就来说明这种效果的制作方式。

图 3-55　游戏中的镂空效果

具体的操作步骤是：

（1）从网址：http://unity3d.com/unity/download/archive，下载 Unity 的内置着色器的代码，如图 3-56 所示。

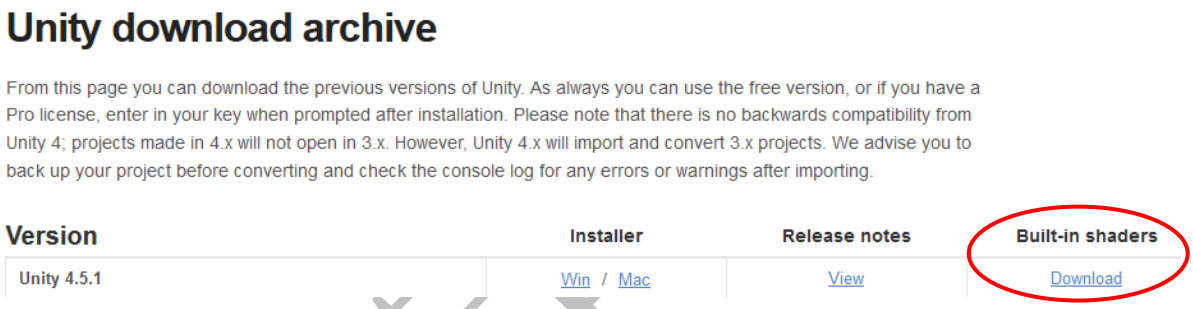


图 3-56　下载 Unity 内置着色器的代码

（2）因为 Unity 的着色器没有提供镂空这种效果，所以这种效果是在修改其它着色器代码的前提下实现的。在下载的资源里，DefaultResourcesExtra 文件夹下找到 Alpha-BumpSpec.shader 文件，复制这个文件，将得到的副本命名为 AlphaTest-DoubleSided.shader，然后使用 MonoDevelop 打开这个文件，并使用下面的代码修改里面与之类似的代码：

```
01  Shader "Transparent/Cutout/DoubleSided" {
02  Properties {
03          _Color ("Main Color", Color) = (1,1,1,1)
04          _SpecColor ("Specular Color", Color) = (0.5, 0.5, 0.5, 0)
05          _Shininess ("Shininess", Range (0.01, 1)) = 0.078125
06          _MainTex ("Base (RGB) TransGloss (A)", 2D) = "white" {}
07          _BumpMap ("Normalmap", 2D) = "bump" {}
08  }
09
10  SubShader {
11          Tags                    {"Queue"="AlphaTest"                    "IgnoreProjector"="True"
"RenderType"="TransparentCutout"}
12          LOD 400
13          Cull Off
```

保存这个文件以后，再将它导入到 Unity 的 Project 视图里。

（3）自己制作一个纹理，要求此纹理具有 transparency 通道，本节使用的纹理如图 3-57

所示。最后将纹理导入到 Unity 的 Project 视图里。

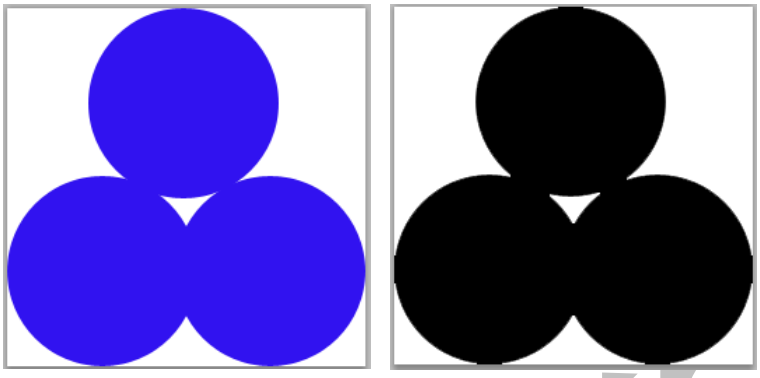


图 3-57　具有 transparency 通道的纹理

（4）在 Project 视图里，新建一个材质，并命名为 cull_mat，至此 Project 视图的三个资源资源如图 3-58 所示。

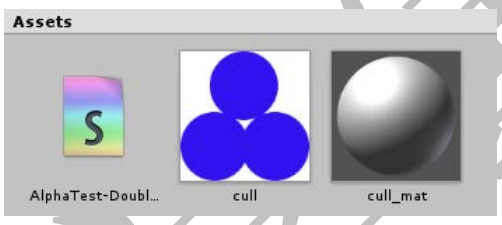


图 3-58　用于实现镂空效果的资源

（5）选中 cull_mat 材质，在 Inspector 视图里，修改它的属性：

- Shader 为 Transparent/Cutout/DoubleSided；
- Base(RGB) TransGloss(A)为导入的纹理；

如图 3-59 所示，即可在预览视图中看到具有镂空效果的立方体图。

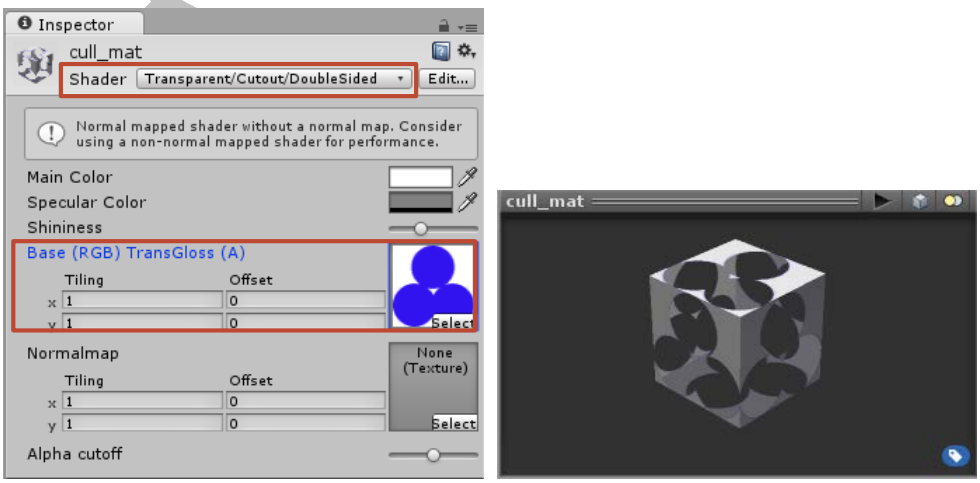


图 3-59　镂空效果的实现