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 2 章　摄像机的应用

作为游戏开发者，千万不要忽略了摄像机（Camera）的重要性。毕竟玩家是通过摄像机，才看到了游戏的视图。本章将介绍一些摄像机的常见应用。

2.1　设置双游戏视图

很多游戏里，一个游戏的视图中，经常会包含另一个游戏视图。而两个视图所呈现的，是由两个摄像机在场景的不同位置所拍摄的画面。例如，《QQ 飞车》中，除了第三人称视图以外，游戏视图的右侧还有一个跑道位置预览视图，如图 2-1 所示。本节将模拟这种情况，然后说明生成这种视图效果的方法。

图 2-1　《QQ 飞车》中的双游戏视图

2.1.1　环境准备

首先，除了场景中默认生成的 Main Camera 对象外，还需要为场景添加 4 个游戏对象：Directional light、Camera、Cube 和 Sphere。改变两个摄像机的位置，使得它们看到场景中不同的游戏对象，如图 2-2 所示。

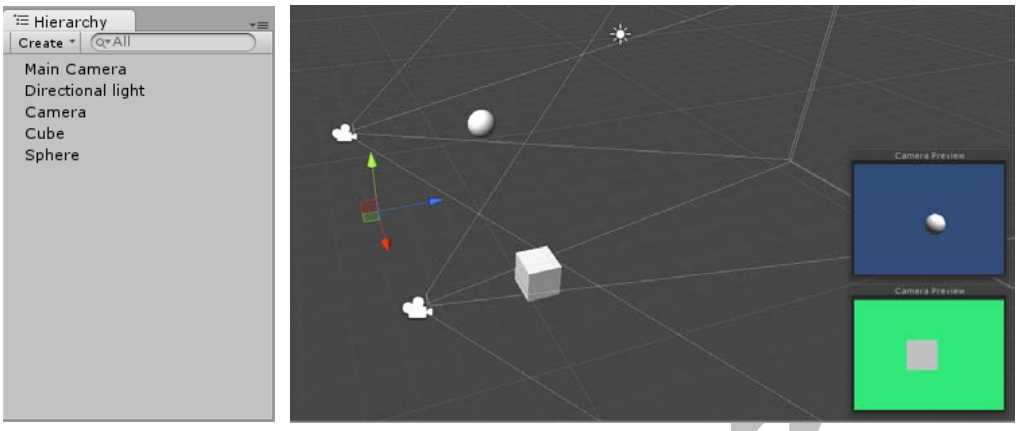


图 2-2　为场景添加游戏对象，并调整它们的位置

Main Camera 所拍摄到的视图里有球体，Camera 所拍摄到的视图里有立方体，修改 Camera 拍摄视图的背景颜色，具体操作是：选中 Camera，在查看器中修改名为 Camera 的组件的 Background 属性为其它的颜色，如图 2-3 所示。最终两个游戏视图合并时，背景颜色可以区分两个视图的边界。

图 2-3　修改摄像机所拍摄视图的背景颜色

2.1.2　编写脚本

然后，在 Project 视图里创建一个 C#脚本文件，并命名为 PictureInPicture。打开脚本文件，并向其中添加如下代码：

```
01  using UnityEngine;
02
03  public class PictureInPicture : MonoBehaviour
04  {
05      //定义枚举类型
06      public enum HorizontalAlignment{left, center, right};
07      public enum VerticalAlignment{top, middle, bottom};
08      public enum ScreenDimensions{pixels, screen_percentage};
09      //定义枚举类型的变量
10      public HorizontalAlignment horizontalAlignment = HorizontalAlignment.left;
11      public VerticalAlignment verticalAlignment = VerticalAlignment.top;
12      public ScreenDimensions dimensionsIn = ScreenDimensions.pixels;
13
14      public int width = 50;
15      public int height= 50;
16      public float xOffset = 0f;
17      public float yOffset = 0f;
```

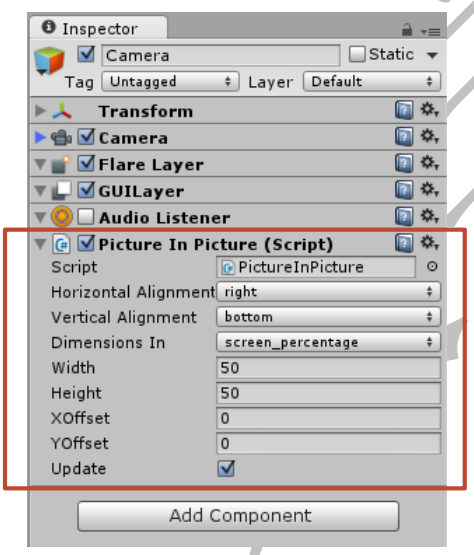
```
18          public bool update = true;
19
20          private int hsize, vsize, hloc, vloc;
21          //游戏对象初始化时，调用此函数
22          void Start ()
23          {
24                  AdjustCamera ();
25          }
26          // 游戏运行时，每一帧都调用此函数
27          void Update ()
28          {
29                  if(update)
30                          AdjustCamera ();
31          }
32          void AdjustCamera()
33          {
34                  if(dimensionsIn == ScreenDimensions.screen_percentage)      //调节视图为指定百
分比大小
35                  {
36                          hsize = Mathf.RoundToInt(width * 0.01f * Screen.width);
37                          vsize = Mathf.RoundToInt(height * 0.01f * Screen.height);
38                  }
39                  else                                                        //调节视图为指定像
素大小
40                  {
41                          hsize = width;
42                          vsize = height;
43                  }
44                  if(horizontalAlignment == HorizontalAlignment.left)         //水平方向上是左对
齐
45                  {
46                          hloc = Mathf.RoundToInt(xOffset * 0.01f * Screen.width);
47                  }
48                  else if(horizontalAlignment == HorizontalAlignment.right)   //水平方向上是右对
齐
49                  {
50                          hloc = Mathf.RoundToInt((Screen.width - hsize) - (xOffset * 0.01f *
Screen.width));
51                  }
52                  else                                                        //水平方向上是居中
对齐
53                  {
54                          hloc = Mathf.RoundToInt(((Screen.width * 0.5f) –
55                                                  (hsize * 0.5f)) - (xOffset * 0.01f *
Screen.height));
56                  }
57                  if(verticalAlignment == VerticalAlignment.top)              //垂直方向上是顶端
对齐
58                  {
59                          vloc = Mathf.RoundToInt((Screen.height - vsize) - (yOffset * 0.01f *
Screen.height));
60                  }
61                  else if(verticalAlignment == VerticalAlignment.bottom)      //垂直方向上是底端
```

```
对齐
62              {
63                  vloc = Mathf.RoundToInt(yOffset * 0.01f * Screen.height);
64              }
65              else                                                    //垂直方向上是居中
对齐
66              {
67                  vloc = Mathf.RoundToInt(((Screen.height * 0.5f) –
68                                                  (vsize  *  0.5f))  -  (yOffset  *  0.01f  *
Screen.height));
69              }
70              camera.pixelRect = new Rect(hloc,vloc,hsize,vsize);
71          }
72  }
```

脚本将依据用户自定义的设置，决定该如何摆放，以及以多大的尺寸，显示 Camera 所拍摄的视图。

2.1.3 实现效果

将此脚本加到 Camera 对象上，选中此对象，即可查看对象上此脚本组件中的各项属性，如图 2-4 所示。



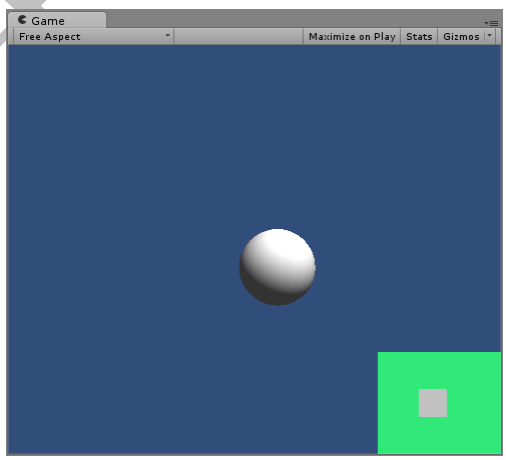


图 2-4　对象脚本组件里的各项属性　　图 2-5　Game 视图

各属性将设置由 Camera 所拍摄的视图，将放置在由 Main Camera 所拍摄视图中的位置和大小。更形象的说法是，小屏幕将放置于大屏幕的哪个位置，是左上角、右下角，还是中间，且小屏幕的大小是多少，占大屏幕的 50%，还是有固定的大小。运行这个游戏项目，得到的 Game 视图，如图 2-5 所示。一个摄像机所拍摄的视图，放置在了另一个摄像机所拍摄视图的右下角。

提示：运行游戏时，Console 视图会显示多条同样的输出信息。如果不想让这个信息出现的话，可以取消 Camera 对象上 Audio Listener 组件的复选即可，如图 2-6 所示。

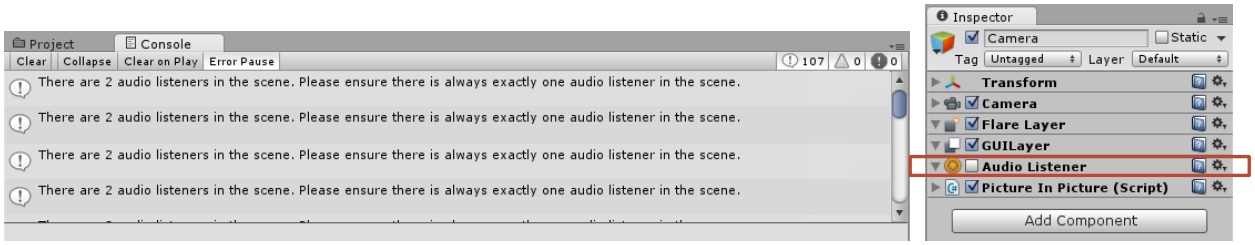


图 2-6　Console 视图上的输出信息，以及取消 Audio Listener 组件的复选

2.2　在多个游戏视图间切换

很多游戏支持玩家切换视角，例如，可以在第一人称视角和第三人称视角间来回切换的《穿越火线》，如图 2-7 所示。本节就来学习，通过键盘按键，切换游戏视图的方法。在上一节使用的项目的基础上，完成本节的示例演示。

图 2-7　第一人称视角与第三人称视角

2.2.1　环境准备

继续为项目添加一个摄像机，项目中拥有的 3 个摄像机的名字分别是：Main Camera、Camera1 和 Camera2。禁用后 2 个摄像机的 Camera 和 Audio Listener 组件。在 Unity 里，单击 GameObject|Create Empty 命令，创建一个空游戏对象，并修改它的名字为 Switchboard。此时场景中的所有游戏对象，以及 3 个摄像机的视图如图 2-8 所示。

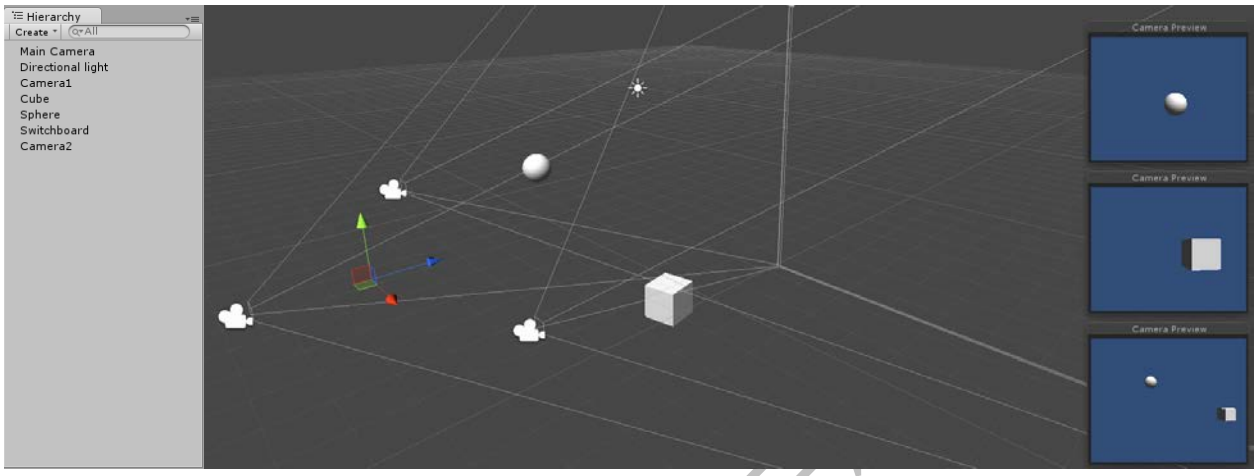


图 2-8　场景中的游戏对象，以及 3 个摄像机的视图

2.2.2　编写脚本

在 Project 视图中，创建一个 C#脚本文件，并命名为 CameraSwitch，打开此脚本文件，并填写如下代码：

```
01  using UnityEngine;
02
03  public class CameraSwitch : MonoBehaviour
04  {
05      public GameObject[] cameras;
06      public string[] shortcuts;
07      public bool changeAudioListener = true;
08      //运行游戏时，每一帧都调用此函数
09      void Update ()
10      {
11          int i = 0;
12          for(i=0; i<cameras.Length; i++)
13          {
14              if (Input.GetKeyUp(shortcuts[i]))
15                  SwitchCamera(i);
16          }
17      }
18      void SwitchCamera ( int index )
19      {
20          int i = 0;
21          for(i=0; i<cameras.Length; i++)
22          {
23              if(i != index)
24              {
25                  if(changeAudioListener)
26                  {
27                      cameras[i].GetComponent<AudioListener>().enabled = false;
28                  }
29                  cameras[i].camera.enabled = false;
```

```
30                  }
31                  else
32                  {
33                      if(changeAudioListener)
34                      {
35                          cameras[i].GetComponent<AudioListener>().enabled = true;
36                      }
37                      cameras[i].camera.enabled = true;
38                  }
39              }
40          }
41      }
```

脚本代码，将依据玩家按下的按键（键盘上的 1、2 和 3），决定启用对应摄像机的 AudioListener 和 camera 组件，禁用其余两部摄像机的 AudioListener 和 camera 组件。玩家将因此看到不同的游戏视图。

2.2.3 实现效果

将脚本加到 CameraSwitch 上，并选中此游戏对象，在 Inspector 视图里查看脚本组件的属性。设置 Cameras 和 Shortcuts 中的 Size 属性为 3，前者依次指定 Main Camera、Camera1 和 Camera2，后者依次指定 1、2 和 3，为 Element0~3 的属性值。如图 2-9 所示。

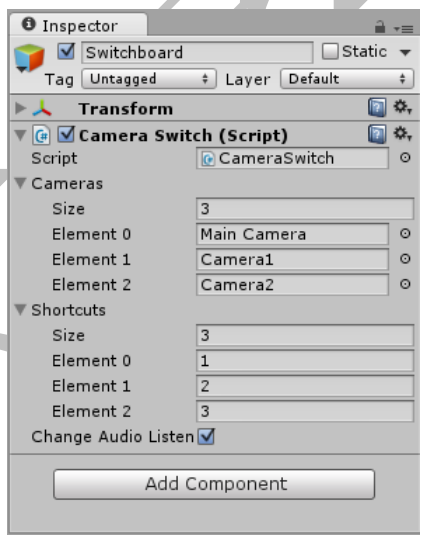
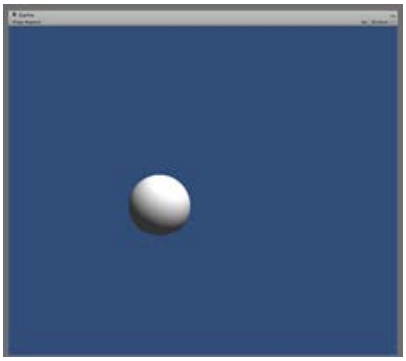
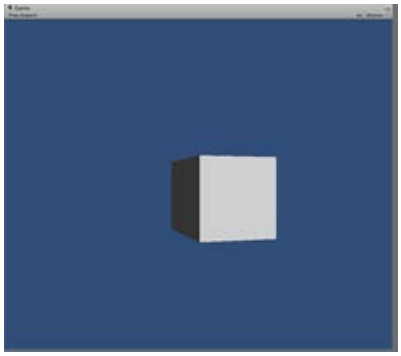


图 2-9　指定脚本组件中各属性的值

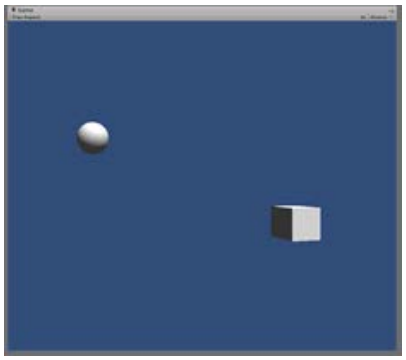
在 Unity 里，运行这个游戏，当分别按下键盘上的 1、2 和 3 按键时，游戏中的视图将发生与按键对应的切换，如图 2-10 所示。



按下1时



按下2时



按下3时

图 2-10　按下键盘上的对应按键，所看到的不同游戏视图

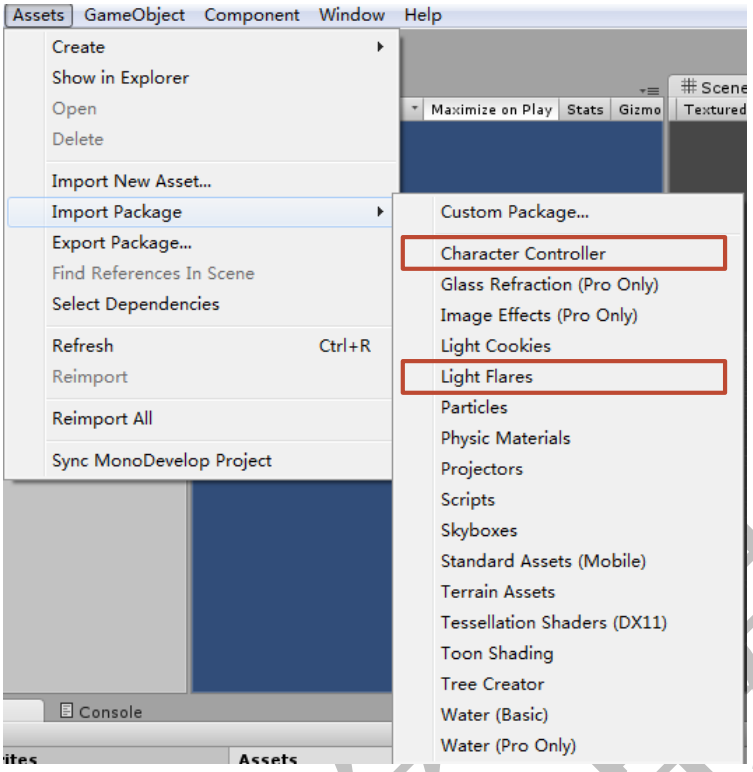
2.3　制作镜头光晕效果

镜头光晕（lens flare）这种效果经常被用在游戏中，尤其是在室外看强光的时候。如图 2-11 所示，网游中的人物得到宝贝时，所产生的镜头光晕效果。

图 2-11　游戏中的镜头光晕效果

Unity 内置的资源包提供了制作这种效果的资源，本节会直接拿来使用。制作步骤如下：

（1）在 Unity 里，单击 Assets|Import Package|Character Controller 命令，导入 Character Controller（角色控制器）资源包。同样的方法，导入 Light Flares（光晕）资源包，如图 2-12 所示。



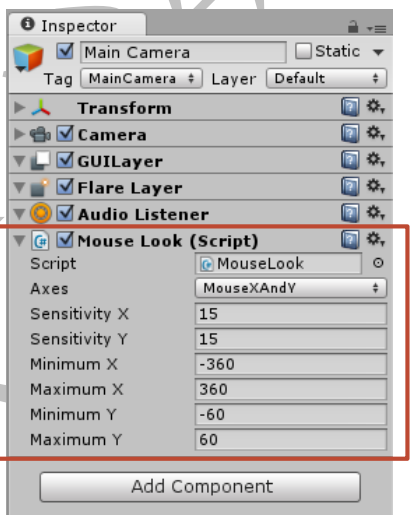


图 2-12　导入 Character Controller 和 Light Flares 资源包

图 2-13　为 Main Camera 添加 Mouse Look 组件

（2）选中 Main Camera，单击 Component|Camera Control|Mouse Look 命令，为其添加 Mouse Look 组件，如图 2-13 所示。

（3）在 Project 视图里，Standard Assets|Light Flares 路径下，有个名为 Sun 的文件。选中它，在 Inspector 视图里可以看到它引用了名为 50mmflare 的光晕纹理，用鼠标点击这个名称就可以定位到这个纹理的所在位置，如图 2-14 所示。

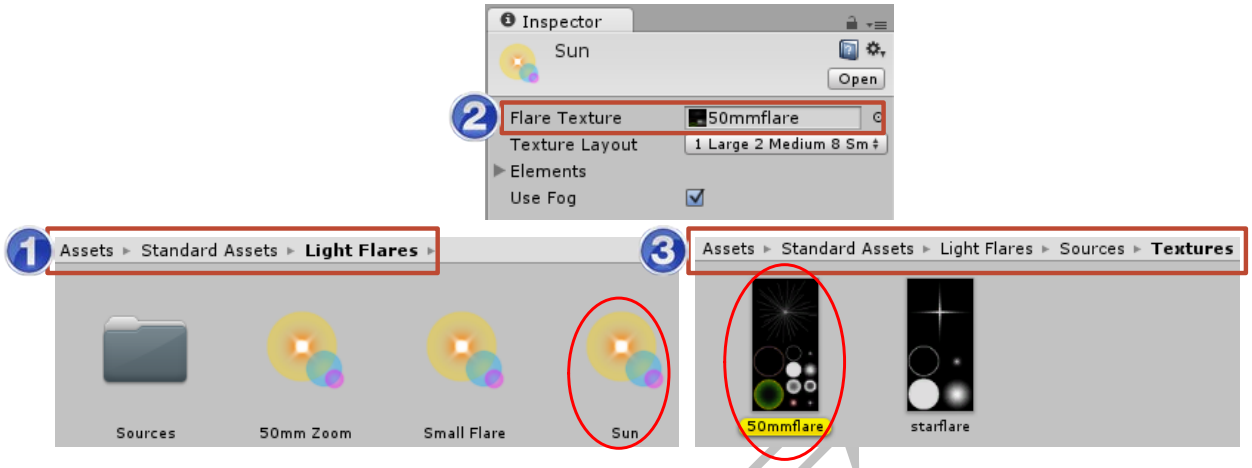
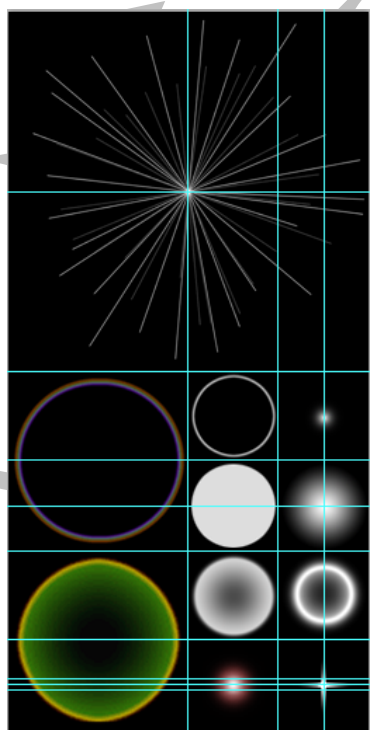


图 2-14　Sun 文件，及其引用的纹理

（4）鼠标双击，打开这个名为 50mmflare 的文件，可以看到这个文件包含了光晕效果的各组成部分，如图 2-15 所示。



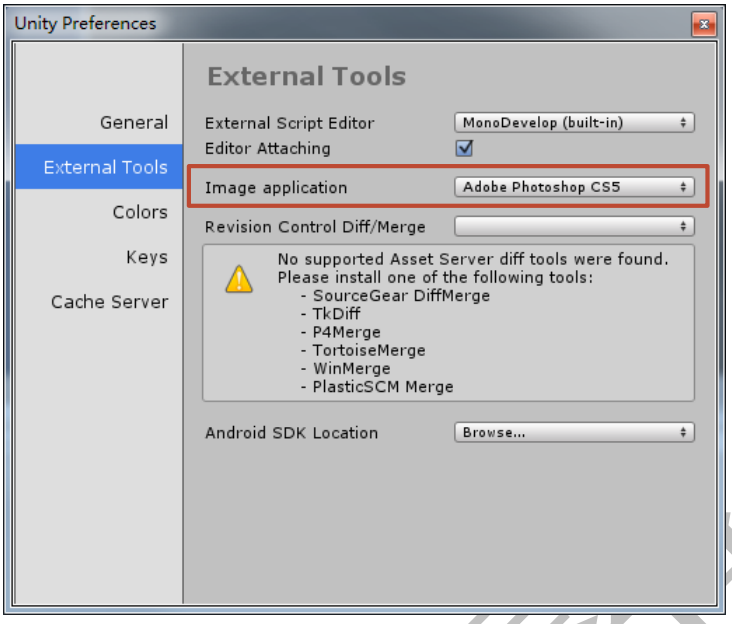


图 2-15　50mmflare 文件中的纹理　　图 2-16　设置打开图像文件的指定软件为 PhotoShop

注意：此文件是.psd 为后缀的文件，默认要由 PhotoShop 打开。如果此时电脑中已经安装了 PhotoShop，最好在 Unity 里，单击 Edit|Preferences…命令，在弹出的 Unity Preferences 对话框中，设置 External Tools 标签下的 Image application 属性为对应的 PhotoShop 软件，如图 2-16 所示。

（5）为游戏场景添加 Directional Light 对象，选中它，然后在 Inspector 视图下，修改 Light 组件的 Flare 属性为 Sun，如图 2-17 所示。

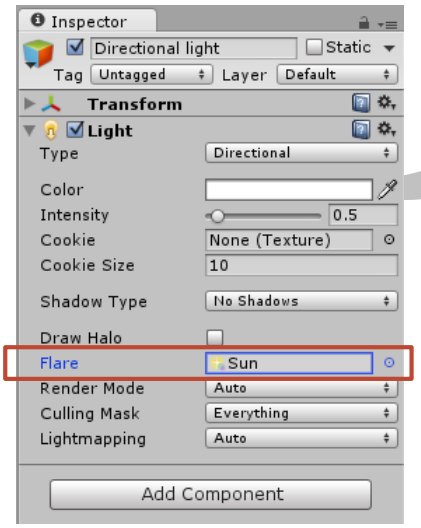


图 2-17　设置 Directional Light 的 Light 组件中的 Flare 属性　　图 2-18　镜头光晕效果

6.运行这个游戏，在 Game 视图中移动鼠标，当摄像机的朝向 Directional Light 时，就会看到镜头光晕的效果了，如图 2-18 所示。

2.4 制作游戏的快照

有些游戏支持玩家“拍快照”，也就是将游戏的精彩瞬间以图片的形式记录下来的功能。这个功能比较有趣，而且以后的用途也会很广，为此本节打算介绍：截取矩形区域内游戏视图，并将其显示在视图其它区域的方法。具体的操作步骤如下：

（1）在 Project 视图里，创建一个 C#脚本文件，并命名为 ScreenTexture。在此脚本中编写如下的代码：

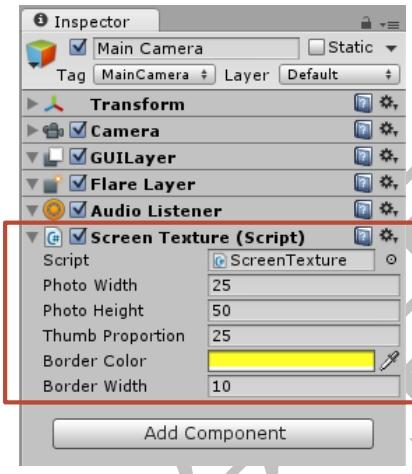
```
01  using UnityEngine;
02  using System.Collections;
03
04  public class ScreenTexture : MonoBehaviour
05  {
06      //公有成员
07      public int photoWidth = 50;                    //矩形的宽度
08      public int photoHeight = 50;                   //矩形的高度
09      public int thumbProportion = 25;               //截图的显示比例
10      public Color borderColor = Color.white;        //矩形框架的颜色
11      public int borderWidth = 2;                    //矩形框的宽度
12      //私有成员
13      private Texture2D texture;
14      private Texture2D border;
15      private int screenWidth;
16      private int screenHeight;
17      private int frameWidth;
18      private int frameHeight;
19      private bool shoot = false;
20      // 脚本初始化时，调用此函数
21      void Start ()
22      {
23          screenWidth = Screen.width;
24          screenHeight = Screen.height;
25          frameWidth = Mathf.RoundToInt(screenWidth * photoWidth * 0.01f);
26          frameHeight = Mathf.RoundToInt(screenHeight * photoHeight * 0.01f);
27          texture = new Texture2D (frameWidth,frameHeight,TextureFormat.RGB24,false);
28          border = new Texture2D (1,1,TextureFormat.ARGB32, false);
29          border.SetPixel(0,0,borderColor);
30          border.Apply();
31      }
32      // 运行游戏时，每帧都调用此函数
33      void Update ()
34      {
35          //鼠标左键按下的时候
36          if (Input.GetKeyUp(KeyCode.Mouse0))
37              StartCoroutine(CaptureScreen());
38      }
39      //在 Game 视图上，绘制纹理
40      void OnGUI ()
41      {
42          //绘制矩形框的四个边
43          GUI.DrawTexture(
```

```
                                new Rect(
                                    (screenWidth*0.5f)-(frameWidth*0.5f) - borderWidth*2,
                                    ((screenHeight*0.5f)-(frameHeight*0.5f)) - borderWidth,
                                    frameWidth + borderWidth*2,
                                    borderWidth),
                                border,ScaleMode.StretchToFill);
        GUI.DrawTexture(
                                new Rect(
                                    (screenWidth*0.5f)-(frameWidth*0.5f) - borderWidth*2,
                                    (screenHeight*0.5f)+(frameHeight*0.5f),
                                    frameWidth + borderWidth*2,
                                    borderWidth),
                                border,ScaleMode.StretchToFill);
        GUI.DrawTexture(
                                new Rect(
                                    (screenWidth*0.5f)-(frameWidth*0.5f)- borderWidth*2,
                                    (screenHeight*0.5f)-(frameHeight*0.5f),
                                    borderWidth,
                                    frameHeight),
                                border,ScaleMode.StretchToFill);
        GUI.DrawTexture(
                                new Rect(
                                    (screenWidth*0.5f)+(frameWidth*0.5f),
                                    (screenHeight*0.5f)-(frameHeight*0.5f),
                                    borderWidth,
                                    frameHeight),
                                border,ScaleMode.StretchToFill);
        //绘制矩形框中截取到的 Game 视图
        if(shoot)
        {
            GUI.DrawTexture(
                                new Rect (
                                    10,
                                    10,
                                    frameWidth*thumbProportion*0.01f,
                                    frameHeight*thumbProportion* 0.01f),
                                texture,ScaleMode.StretchToFill);
        }
    }
    //截取矩形框里的 Game 视图
    IEnumerator CaptureScreen ()
    {
        yield return new WaitForEndOfFrame();
        texture.ReadPixels(
            new Rect(
                (screenWidth*0.5f)-(frameWidth*0.5f),
                (screenHeight*0.5f)-(frameHeight*0.5f),
                frameWidth,
                frameHeight),
            0,0);
        texture.Apply();
        shoot = true;
    }
```

```
97  }
```

脚本代码中，40 行的 OnGUI()函数，使用 GUI.DrawTexture()绘制了矩形框，以及矩形框中截取到的游戏视图；84 行的 CaptureScreen()函数，使用 texture.ReadPixels()读取矩形框中的所有像素点，然后存储到 texture 中。触发“拍照”功能的操作是，在 Game 视图中的任意位置，单击鼠标左键，即 36 行代码实现的功能。

（2）将脚本 ScreenTexture 赋予 Main Camera，然后选中 Main Camera，在 Inspector 视图里可以设置脚本组件的一些属性，如图 2-19 所示。



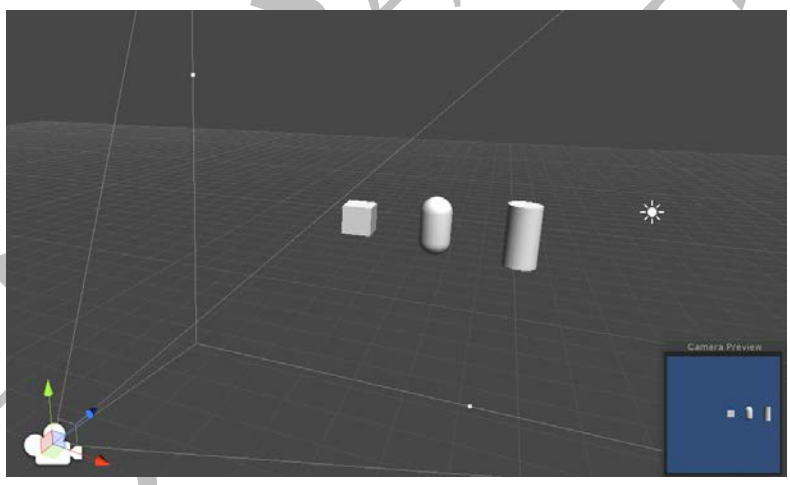


图 2-19　Main Camera 上 ScreenTexture 脚本组件的各属性　　图 2-20　当前的 Scene 视图

（3）为游戏的场景添加一些几何体，并调整它们各自的位置，如图 2-20 所示。

（4）运行游戏，当在 Game 视图中的任意位置，单击鼠标左键的时候，可在视图的左上角看到矩形框中截取到的游戏视图，如图 2-21 所示。

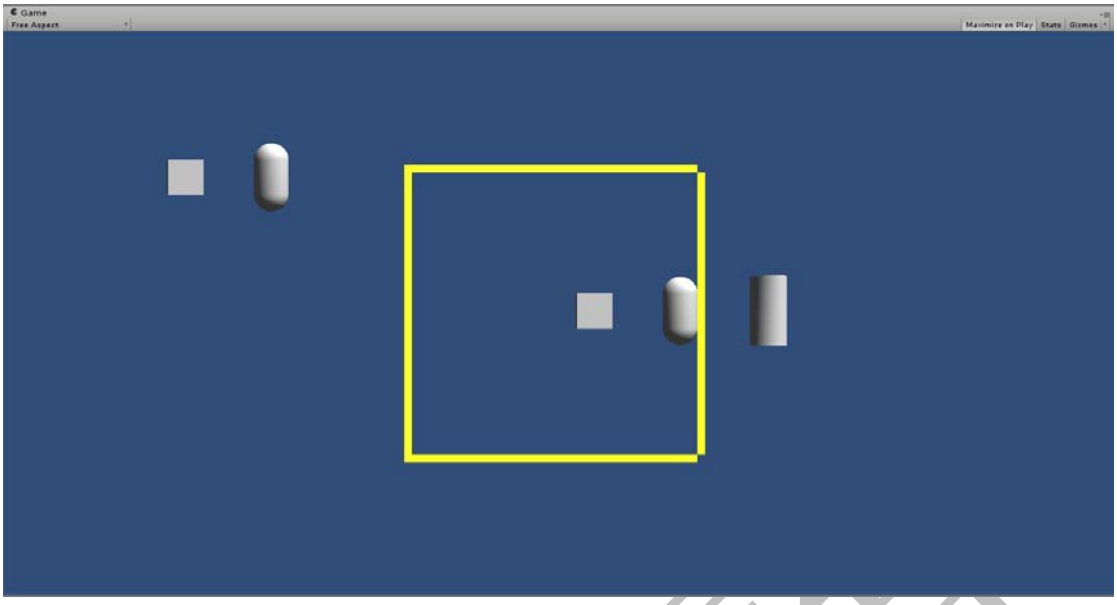


图 2-21　运行游戏，查看截图的效果

2.5　制作一个望远镜

本节制作的望远镜，在鼠标左键按下时，看到的视图会变大；当不再按下的时候，会慢慢缩小成原来的视图。游戏中时常出现的狙击手就是使用望远镜的一个例子，如图 2-22 所示。

图 2-22　游戏中狙击手所看到的视图

制作望远镜的过程如下：

（1）在 Project 视图里，创建一个 C#脚本文件，命名为 TelescopicView。打开这个脚本文件，并在里面添加下面的代码：

```
01    using UnityEngine;
02    using System.Collections;
03
```

```
04  public class TelescopicView : MonoBehaviour
05  {
06      //公有成员
07      public float ZoomLevel = 2.0f;
08      public float ZoomInSpeed = 100.0f;
09      public float ZoomOutSpeed = 100.0f;
10      //私有成员
11      private float initFOV;
12      //脚本初始化时，调用此函数
13      void Start ()
14      {
15          //获取当前摄像机的视野范围
16          initFOV = Camera.main.fieldOfView;
17      }
18      //运行游戏时的每一帧，都调用此函数
19      void Update ()
20      {
21          //当鼠标左键按下时
22          if (Input.GetKey(KeyCode.Mouse0))
23          {
24              ZoomView();
25          }
26          else
27          {
28              ZoomOut();
29          }
30      }
31      //放大摄像机的视野区域
32      void ZoomView()
33      {
34          if (Mathf.Abs(Camera.main.fieldOfView - (initFOV / ZoomLevel)) < 0.5f)
35          {
36              Camera.main.fieldOfView = initFOV / ZoomLevel;
37          }
38          else if (Camera.main.fieldOfView - (Time.deltaTime * ZoomInSpeed) >= (initFOV /
ZoomLevel))
39          {
40              Camera.main.fieldOfView -= (Time.deltaTime * ZoomInSpeed);
41          }
42      }
43      //缩小摄像机的视野区域
44      void ZoomOut()
45      {
46          if (Mathf.Abs(Camera.main.fieldOfView - initFOV) < 0.5f)
47          {
48              Camera.main.fieldOfView = initFOV;
49          }
50          else if (Camera.main.fieldOfView + (Time.deltaTime * ZoomOutSpeed) <= initFOV)
51          {
52              Camera.main.fieldOfView += (Time.deltaTime * ZoomOutSpeed);
53          }
54      }
55  }
```

放大和缩小摄像机的视野区域，依靠的变量是 Camera.main.fieldOfView，所以代码 32 行的 ZoomView()和 44 行的 ZoomOut()就是通过改变 Camera.main.fieldOfView 的大小，进而调节视野区域的大小的。

（2）将脚本 TelescopicView 添加到 Main Camera 上，选中后者后，在 Inspector 视图上查看脚本组件中的各属性，如图 2-23 所示。Zoom Level 可以调节视图能放大到什么程度；Zoom In Speed 调节视图的放大速度；Zoom Out Speed 调节视图缩小的速度。

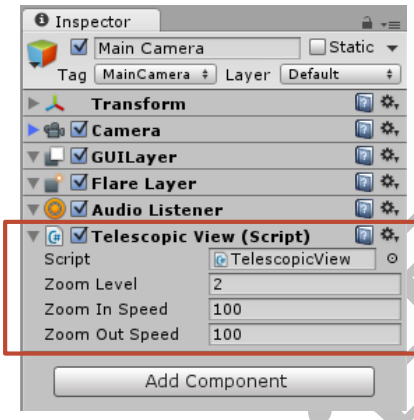


图 2-23　TelescopicView 组件上的各属性

（3）运行游戏，当鼠标左键按下时，视图会放大，松开时，视图会慢慢恢复成原来的样子。如图 2-24 所示。

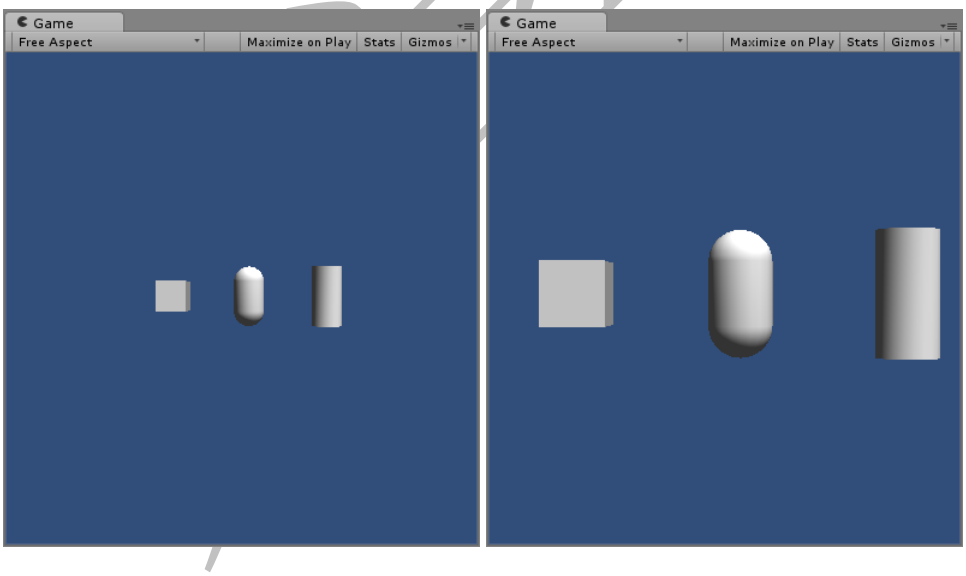


图 2-24　视图的放大与缩小

（4）如果读者此时使用的 Unity 是付费版本的话，还可以把这个望远镜的视图效果做的更形象一些。在 Unity 里，单击 Assets|Import Package|Image Effects(Pro only)命令，导入 Image Effects（图像效果）资源包，如图 2-25 所示。

（5）选中 Main Camera，单击 Component|Image Effects|Vignette 命令，为它添加 Vignette（光晕、光损失）组件，如图 2-26 所示。

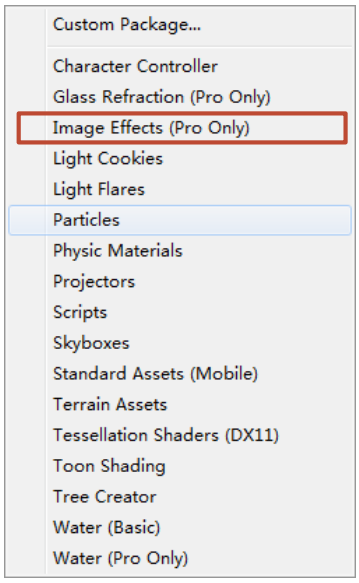


图 2-25　导入 Image Effects 资源包

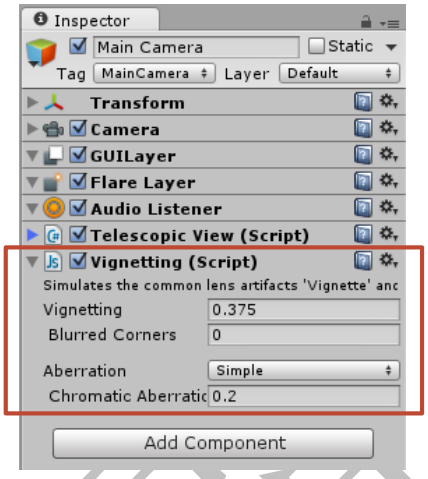


图 2-26　添加 Vignette 组件

（6）再次运行游戏。当鼠标左键按下时，望远镜的视图效果更逼真了（添加了光晕的效果），如图 2-27 所示。

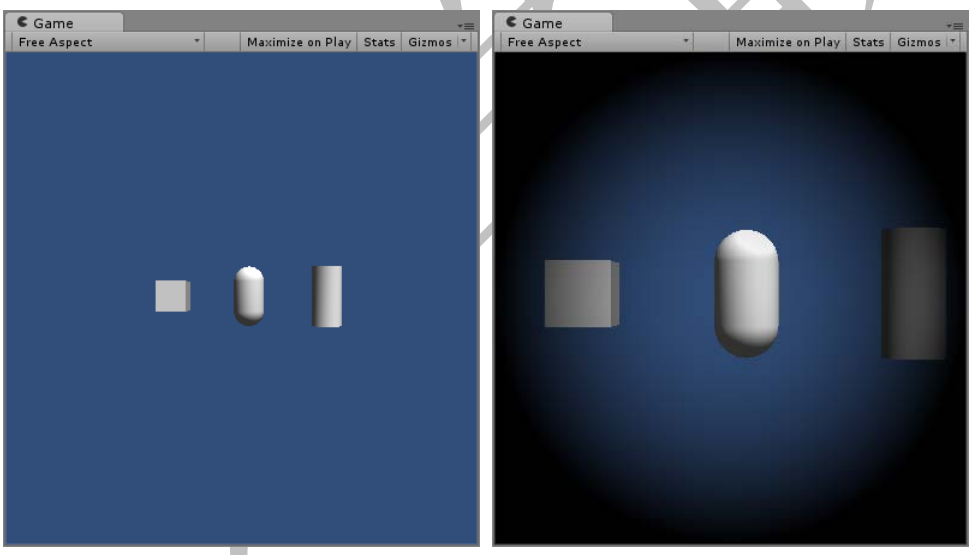


图 2-27　望远镜视图的放大与缩小

2.6　制作一个查看器摄像机

如果是一个侦探类的游戏，在侦探找到一个可疑物品时，总会四下打量这个物品，有时还会拿着放大镜放大一些细节的地方。而本节要制作的查看器摄像机，就模拟了这一过程，它的操作类似于在 Scene 视图上对游戏对象所做的旋转和缩放操作。制作步骤如下：

（1）在 Project 视图里，创建一个 C#脚本文件，命名为 InspectCamera。打开这个脚本文件，并在里面添加下面的代码：

```
01  using UnityEngine;
02  using System.Collections;
```

```
03  //添加一个菜单项
04  [AddComponentMenu("Camera-Control/Inspect Camera")]
05  public class InspectCamera: MonoBehaviour
06  {
07      //公有成员
08      public Transform target;                    //查看器摄像机查看的目标对象
09      public float distance= 10.0f;               //摄像机与目标对象间的距离
10      public float xSpeed= 250.0f;                //横向旋转目标对象时的速度
11      public float ySpeed= 120.0f;                //纵向旋转目标对象时的速度
12      public float yMinLimit= -20.0f;             //纵向旋转目标对象时的范围（最小值）
13      public float yMaxLimit= 80.0f;              //纵向旋转目标对象时的范围（最大值）
14      public float zoomInLimit= 2.0f;             //最大的方法倍数
15      public float zoomOutLimit= 1.0f;            //最小的缩小倍数
16      //私有成员
17      private float x= 0.0f;
18      private float y= 0.0f;
19      private float initialFOV;
20      //脚本初始化时调用
21      void Start ()
22      {
23          initialFOV = camera.fieldOfView;
24          transform.position = new Vector3(0.0f, 0.0f, -distance) + target.position;
25          Vector3 angles= transform.eulerAngles;
26          x = angles.y;
27          y = angles.x;
28          if (rigidbody)
29              rigidbody.freezeRotation = true;
30      }
31      //在每帧即将结束时调用
32      void   LateUpdate ()
33      {
34          //按下鼠标左键时
35          if (target && Input.GetMouseButton(0))
36          {
37              //按下键盘左边的 Shift，或者右边的 Shift
38              if(Input.GetKey(KeyCode.RightShift) || Input.GetKey(KeyCode.LeftShift))
39              {
40                  float zoom= camera.fieldOfView - Input.GetAxis ("Mouse Y");
41                  if(zoom >= initialFOV / zoomInLimit && zoom <= initialFOV /
zoomOutLimit)
42                  {
43                      //改变摄像机视图的大小
44                      camera.fieldOfView -= Input.GetAxis ("Mouse Y");
45                  }
46              }
47              else
48              {
49                  x += Input.GetAxis("Mouse X") * xSpeed * 0.02f;
50                  y -= Input.GetAxis("Mouse Y") * ySpeed * 0.02f;
51              }
52              y = ClampAngle(y, yMinLimit, yMaxLimit);
53              Quaternion rotation= Quaternion.Euler(y, x, 0);
54              Vector3 position= rotation * new Vector3(0.0f, 0.0f, -distance) +
```

```
target.position;
55                      //修改旋转度和位置
56                      transform.rotation = rotation;
57                      transform.position = position;
58                  }
59              }
60              //旋转的角度范围
61              static float    ClampAngle ( float angle ,     float min ,     float max    )
62              {
63                  if (angle < -360.0f)
64                      angle += 360.0f;
65                  if (angle > 360.0f)
66                      angle -= 360.0f;
67                  return Mathf.Clamp (angle, min, max);
68              }
69      }
```

（2）将脚本 InspectCamera 添加到 Main Camera 上，选中后者，在查看器中查看脚本组件中的各属性，如图 2-28 所示。各属性的意义在脚本的 08~15 行有注释说明，这里不再重复。

注意：需要给 Target 属性赋予一个明确的游戏对象，本示例赋予的是 Capsule（胶囊），如图 2-29 所示。

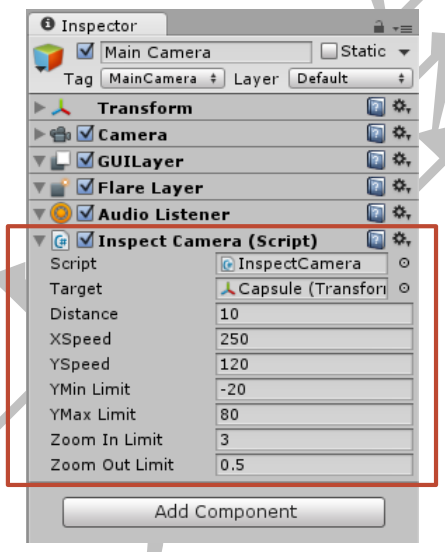


图 2-28　添加到 Main Camera 上的脚本组件中的各属性

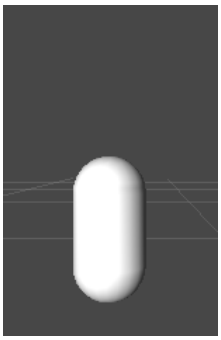
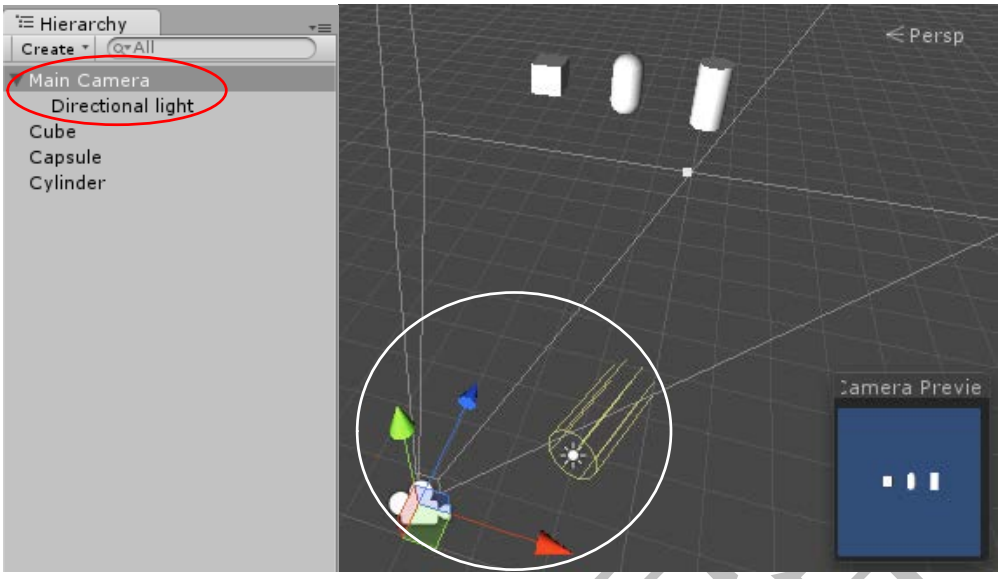


图 2-29　场景中的 Capsule 对象

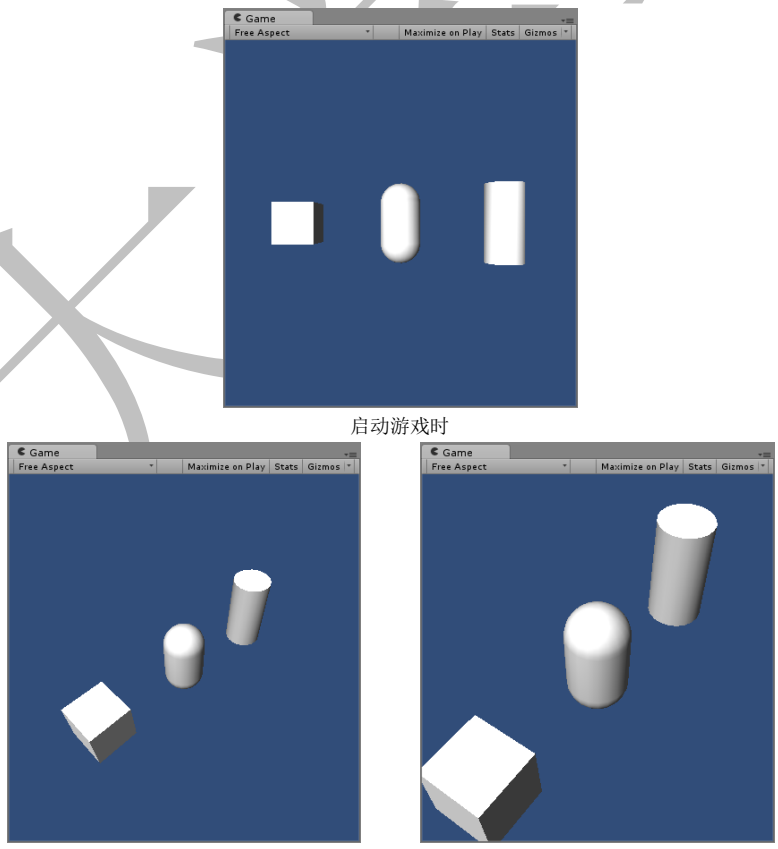
（3）为游戏场景添加 Directional light，并保证此对象上光的照射方向与摄像机的视图方向一致，这会使得摄像机的视图，所呈现的对象的正面是被照亮的。然后在 Hierarchy 视图里拖动 Directional light 到 Main Camera，使前者称为后者的子对象，如图 2-30 所示。如此一来，Directional light 就会随着 Main Camera 的移动而移动。最后达到的效果，就像是人物角色的帽子上有手电筒，人往哪里看，光就往哪里照；人走到哪里，光就跟到哪里。



Directional light是Main Camera的子对象　　摄像机视图朝向与光照方向一致

图 2-30　Directional light 与 Main Camera 的设置

（4）运行游戏，在 Game 视图里，按下鼠标左键并移动，视图会绕着目标对象移动；按下鼠标的同时按下键盘上的 Shift 键，上下移动鼠标可以放大和缩小视图所看到的目标对象，如图 2-31 所示。



启动游戏时

绕着目标对象移动　　放大查看目标对象

图 2-31　查看器摄像机运行效果

2.7 使用忍者飞镖创建粒子效果

游戏中，诸如烟、火、水滴、落叶等粒子效果，都可以使用粒子系统（particle system）来实现。例如，《明朝传奇》中的篝火，如图 2-32 所示。粒子系统的最新版本也被称做忍者飞镖（Shuriken），因为场景中添加的粒子系统酷似忍者飞镖，如图 2-33 所示。

图 2-32 游戏中的篝火

图 2-33 粒子系统，也被称为忍者飞镖

2.7.1 粒子基本属性

在使用粒子系统前，先了解一下它的基本属性：

- Energy：表示粒子的生命周期，也就是从生成到销毁的时间。
- Looping：决定是否在所有粒子的生命周期结束以后，重新生成这些粒子。
- Speed，direction，and rotation：每个粒子都有 transform 组件，因此它们的移动方向、朝向，甚至是大小都可以是不同的。

2.7.2 粒子的值

给每个粒子赋予的值可以分为以下四类：

- Constant（常量）：表示给所有粒子赋予的值都是一样的，如图 2-34 所示。

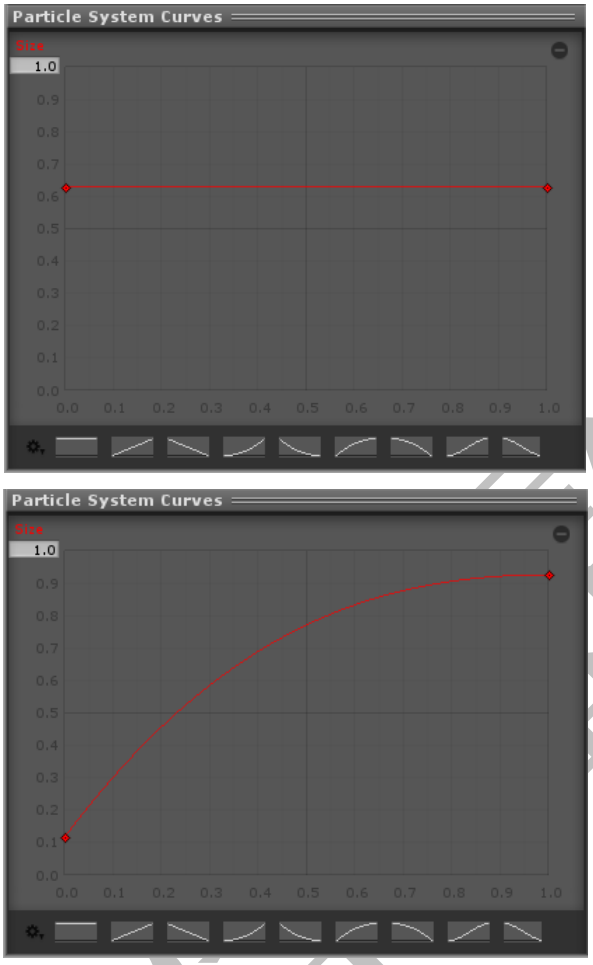


图 2-34　给粒子赋予 Constant 类型的值　　图 2-35　给粒子赋予 Curve 类型的值

- Curve（曲线）：给粒子赋予的值随时间变化而变化，具体的值将依据时间而赋予曲线上的值，如图 2-35 所示。
- Random Between Two Constants（两个常量范围内的随机值）：给粒子赋予的值随时间变化而变化，但赋予的值仅限于两个常量的范围内。如图 2-36 所示。

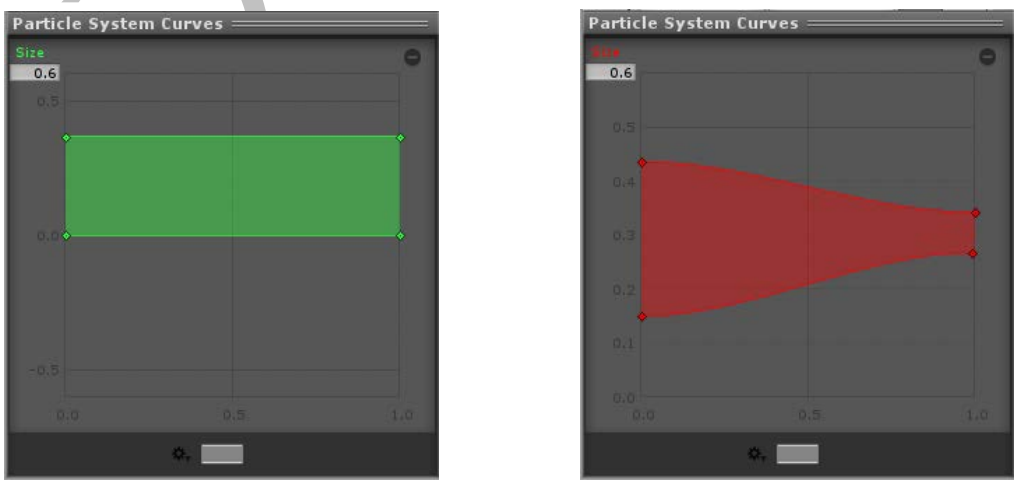


图 2-36　给粒子赋予 Random Between Two Constants 类型的值图 2-37 给粒子赋予 Random Between Two Curves 类型的值

- Random Between Two Curves（两个曲线范围内的随机值）：给粒子赋予的值随时间变化而变化，但赋予的值仅限于两个曲线的范围内。如图 2-37 所示。

2.7.3 创建粒子效果

创建粒子效果的方式有两种：

- 在 Unity 里，单击 GameObject|Create Other|Particle System 命令，在游戏场景中添加 Particle System（粒子系统）这个游戏对象。
- 在 Unity 里，单击 GameObject|Create Empty 命令，在游戏场景中添加空游戏对象。选中它，再单击 Component|Effects|Particle System 命令，为空游戏对象添加 Particle System（粒子系统）组件。

Scene 视图和 Game 视图里所看到的粒子效果如图 2-38 所示。

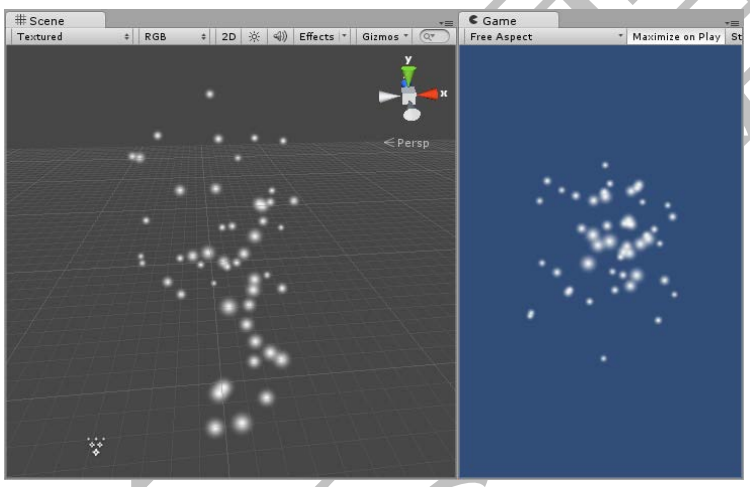


图 2-38　Scene 视图和 Game 视图里的粒子效果

2.7.4 了解粒子系统的初始化模块

Shuriken（忍者飞镖）粒子系统，是采用模块化来管理的，由于有个性化的粒子模块，再配合上粒子曲线编辑器，会使得开发者更容易创作出缤纷复杂的粒子效果。Shuriken 粒子系统有多达 17 个模块，选中 Scene 视图里的粒子系统，然后在 Inspector 视图里就可以查看这 17 个模块，如图 2-39 所示。

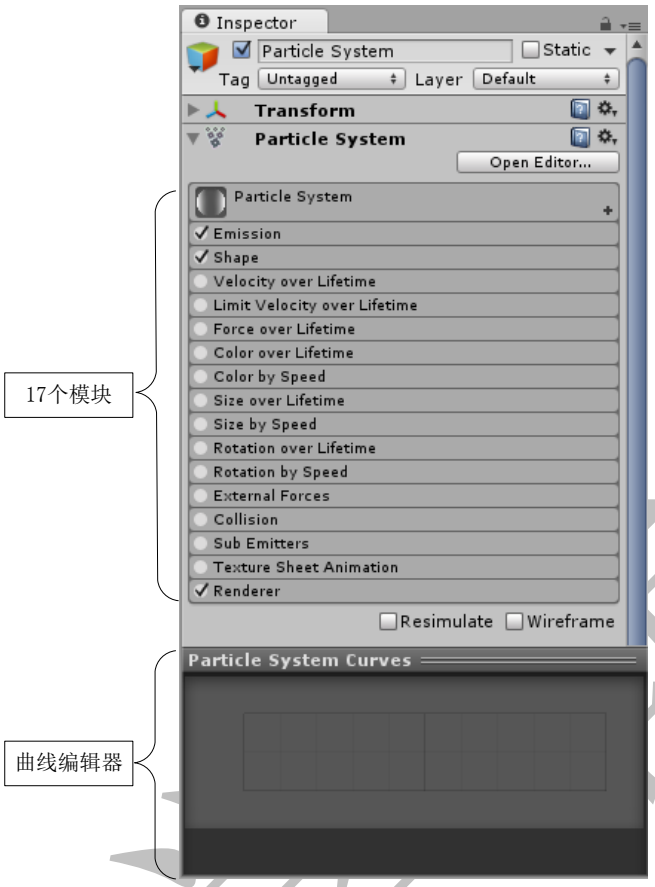


图 2-39　Shuriken 粒子系统的 17 个模块

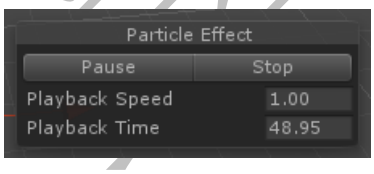


图 2-40　粒子效果面板

除此以外，选中 Scene 视图里的粒子系统时，Scene 视图里还会出现 Particle Effect（粒子效果）面板，如图 2-40 所示。面板中各控件的功能描述如下：

- Pause（暂停）：单击此按钮，可暂停播放当前的粒子。再次单击此按钮，则继续播放。
- Stop（停止）：单击此按钮，可停止粒子的播放。
- Playback Speed（回放速度）：可改变粒子的播放速度，数值越大，播放速度越快。
- Playback Time（回放时间）：设置播放哪一时刻的粒子效果。

点击 Inspector 视图里的 Particle System 模块，即可展开此模块，如图 2-41 所示，同理于其它模块，此模块可以设置粒子系统初始化时的状态。

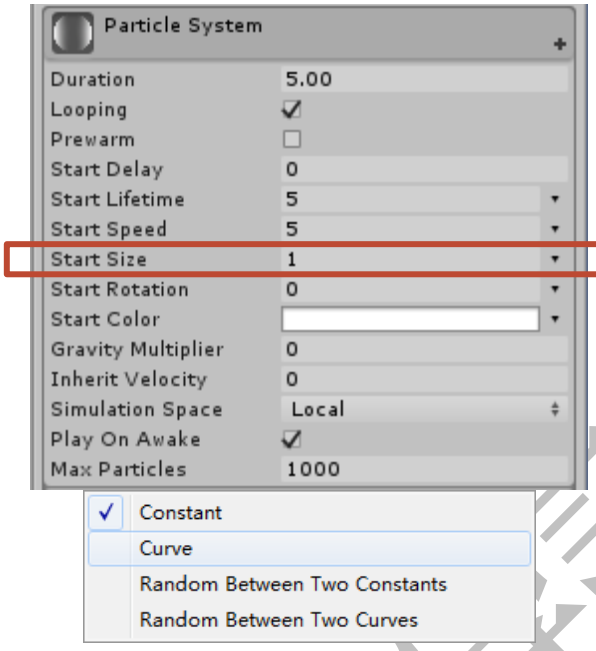


图 2-41　设置粒子系统初始化状态的模块　　图 2-42　Start Size 值的 4 个类型

接下来将以初始化时，设置 Start Size 的值为例，说明设置方法。单击此属性最右边的，会弹出 4 个类型值的选择项，如图 2-42 所示。

- 默认选择的是 Constant，设置 Start Size 的值为常量，即粒子播放的过程中，所有粒子的大小都是一样的，如图 2-43 所示。

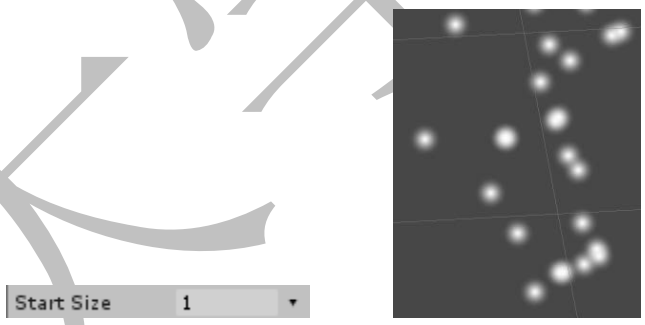


图 2-43　Start Size 为 Constant 类型的值时，粒子播放的效果

- 选择 Curve 时，即可在曲线编辑器中，编辑曲线的形状，设置 Start Size 的值为曲线上的值，粒子播放的效果如图 2-44 所示。随着时间的变化，粒子越来越大。

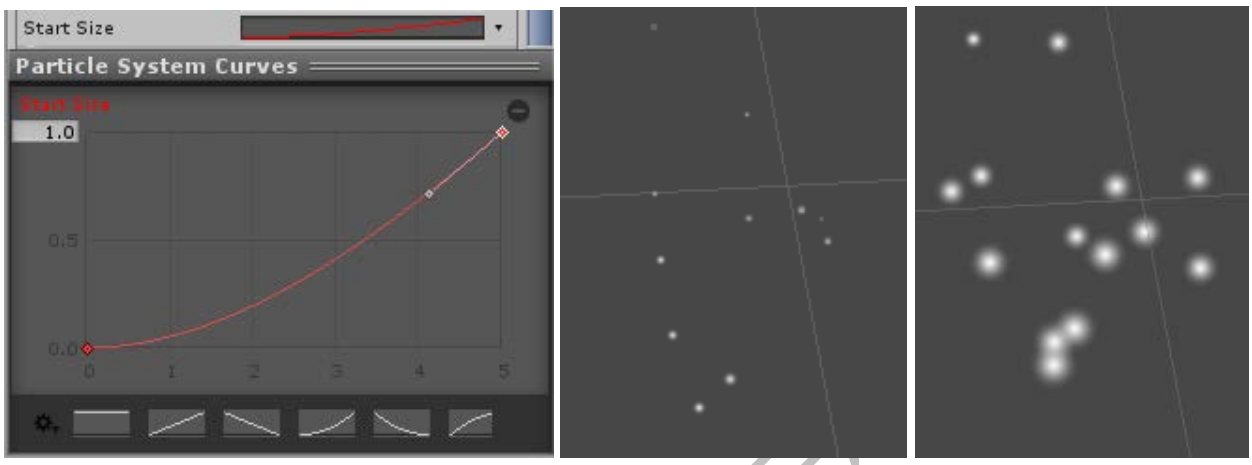


图 2-44　Start Size 为 Curve 类型的值时，粒子播放的效果

- 选择 Random Between Two Constants 时，设置 Start Size 的值为两个常量间的任意值，粒子播放的效果如图 2-45 所示。在任意时刻，粒子总是有大有小。

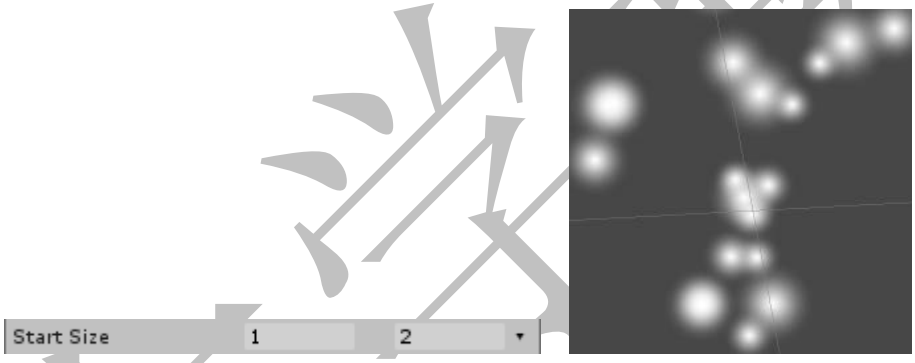


图 2-45　Start Size 为 Random Between Two Constants 类型的值时，粒子播放的效果

- 选择 Random Between Two Curves 时，即可在曲线编辑器中，编辑两条曲线的形状，设置 Start Size 的值为曲线范围内的任意值，粒子播放的效果如图 2-46 所示。任意时刻粒子都是有大有小的，但是随着时间的变化，所有粒子都会越来越大。

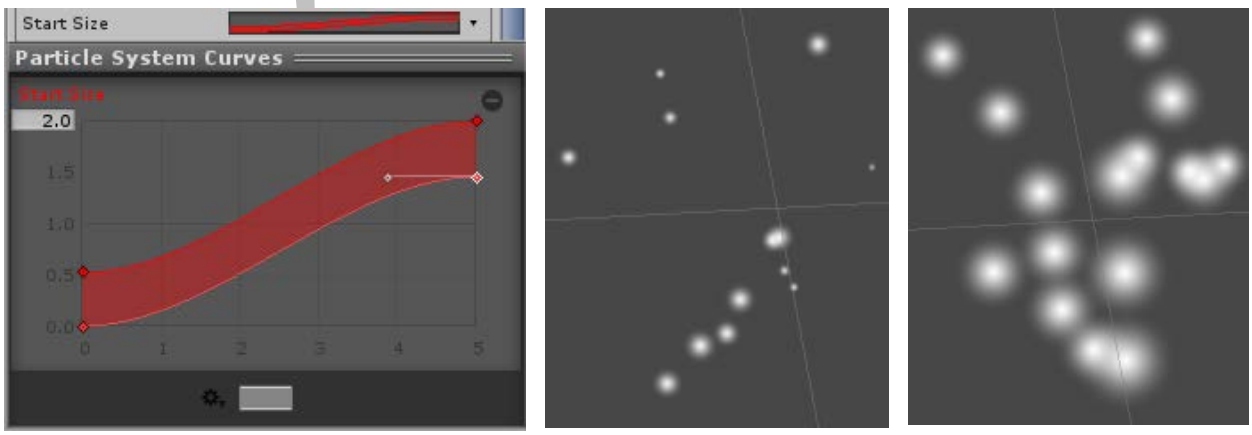


图 2-46　Start Size 为 Random Between Two Curves 类型的值时，粒子播放的效果