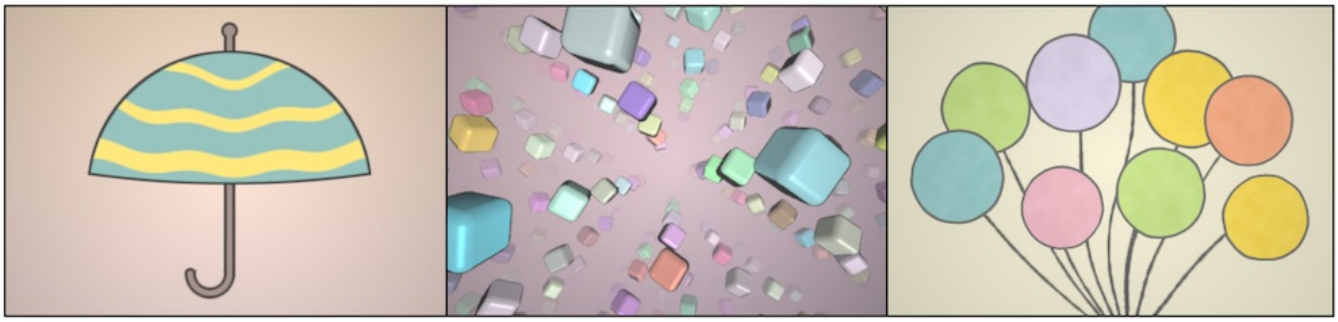
《Unity Shader入门精要》随书彩色插图

说明：本页面是书籍《Unity Shader入门精要》的随书彩图集锦，包含了书中所有的插图，使用时可通过图片编号进行搜索。

作者：冯乐乐
邮箱：lelefeng1992@gmail.com

前言



第2章 渲染流水线

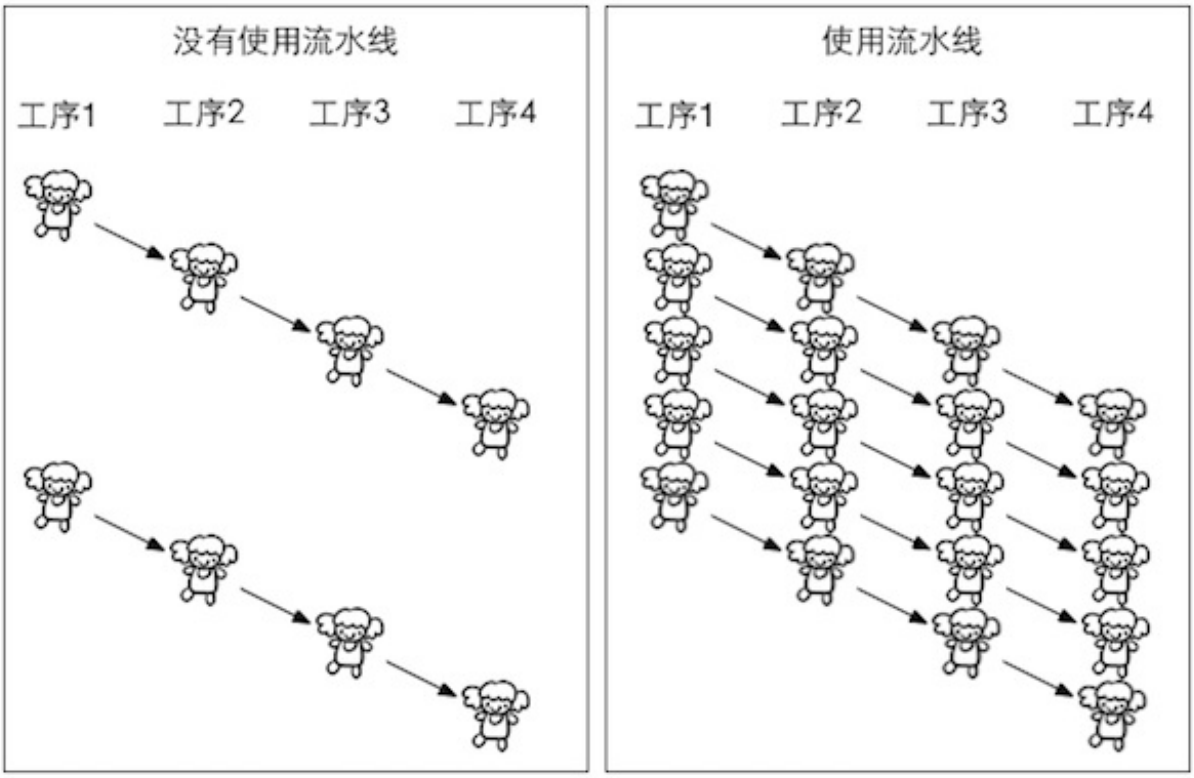


图2.1　真实生活中的流水线

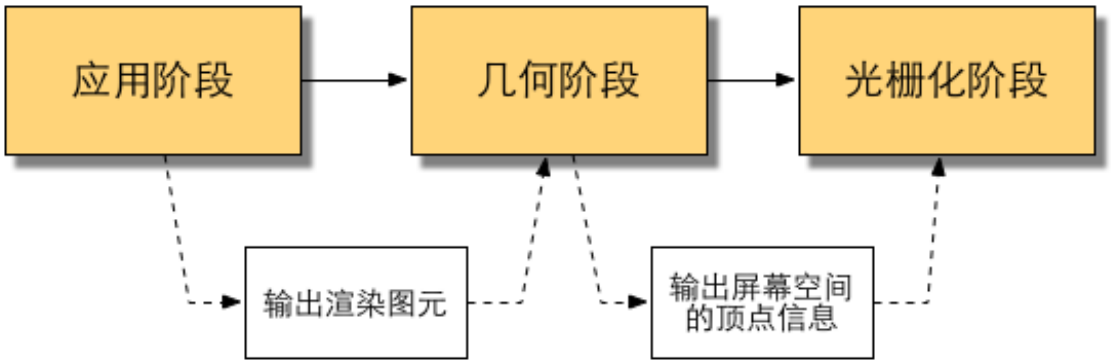


图2.2 渲染流水线中的三个概念阶段

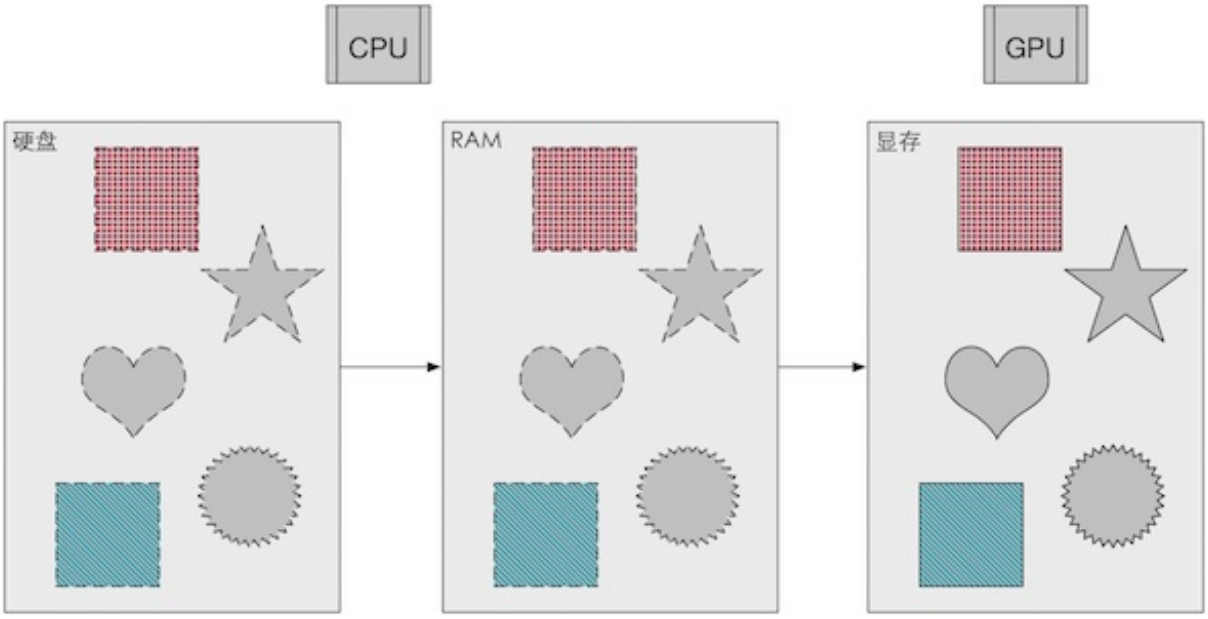


图2.3 渲染所需的数据（两张纹理以及3个网格）从硬盘最终加载到显存中。在渲染时，GPU可以快速访问这些数据

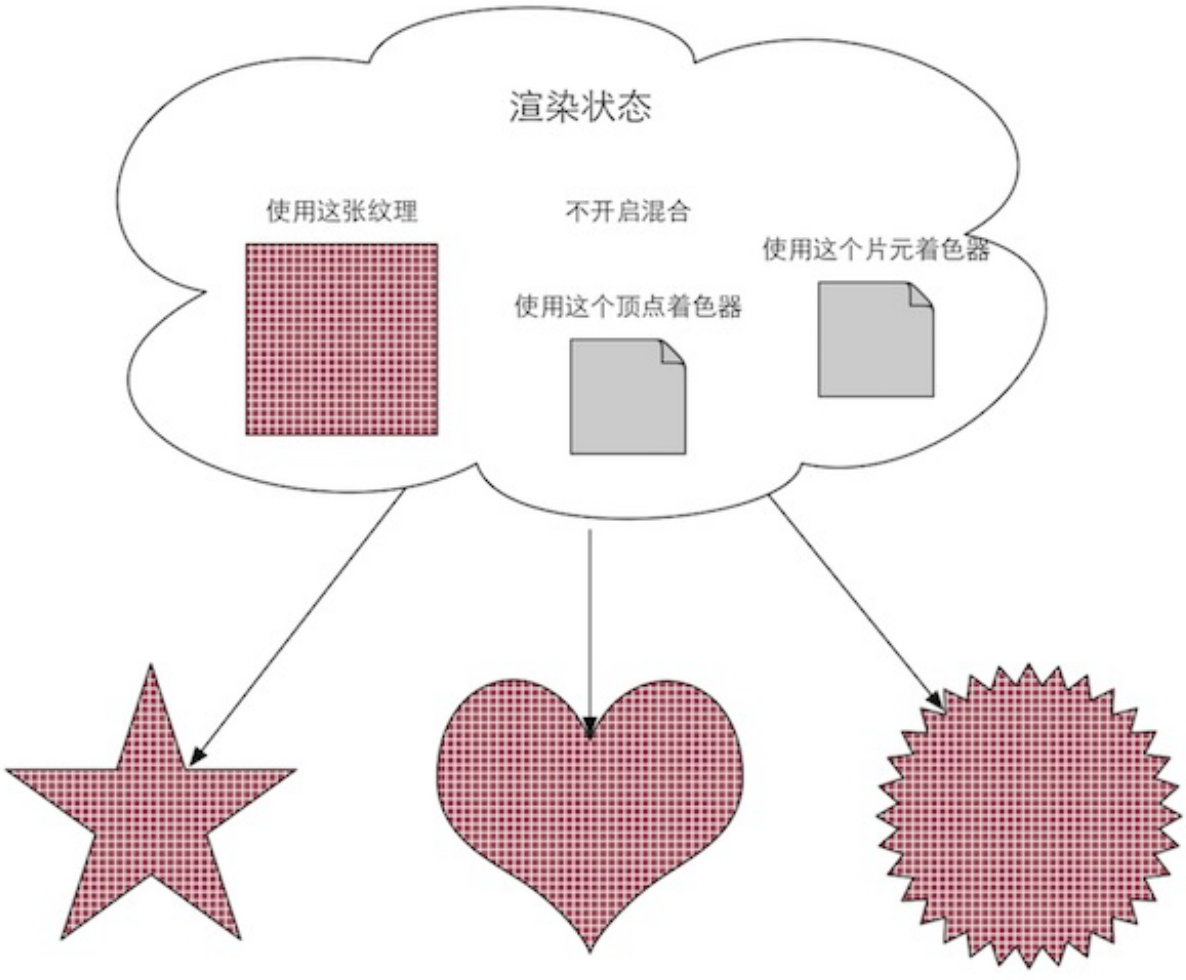


图2.4 在同一状态下渲染三个网格。由于没有更改渲染状态，因此三个网格的外观看起来像是同一种材质的物体。

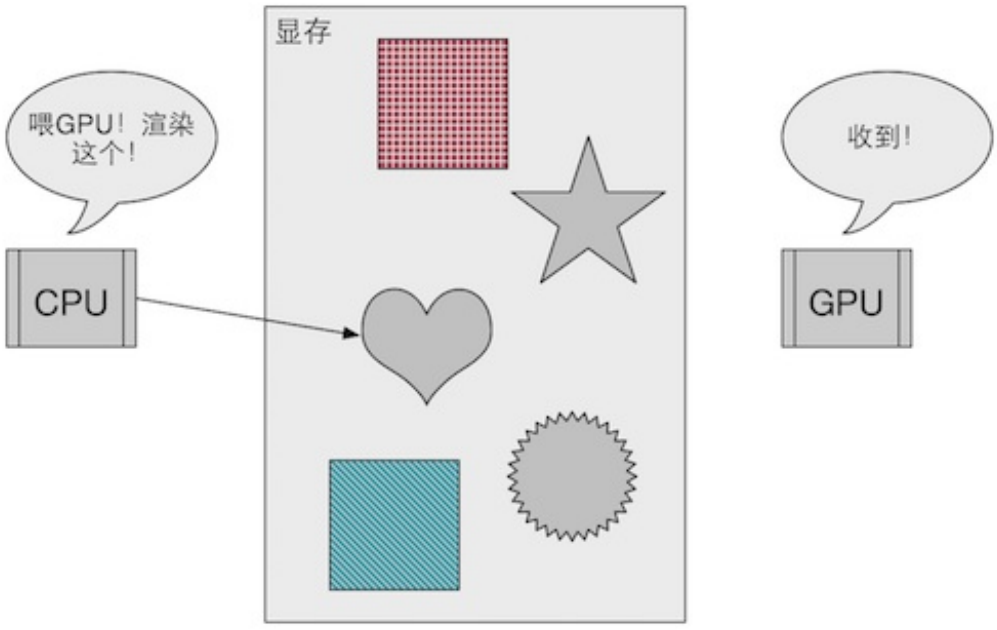


图2.5 CPU通过调用Draw Call来告诉GPU开始进行一个渲染过程。一个Draw Call会指向本次调用需要渲染的图元列表

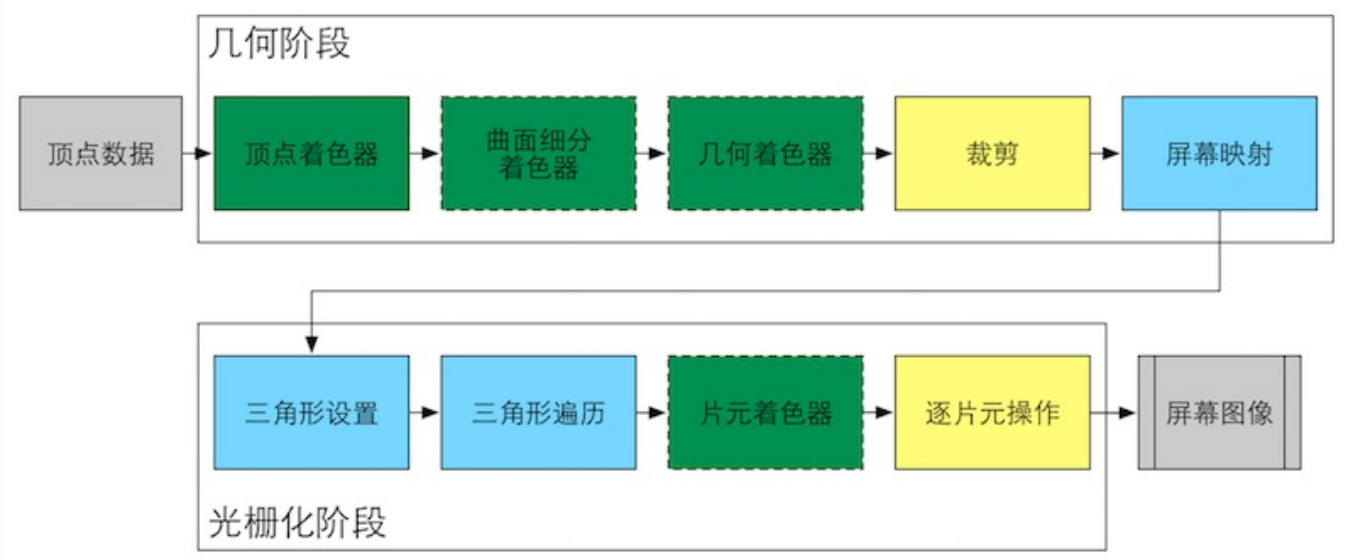


图2.6 GPU的渲染流水线实现。颜色表示了不同阶段的可配置性或可编程性：绿色表示该流水线阶段是完全可编程控制的，黄色表示该流水线阶段可以配置但不是可编程的，蓝色表示该流水线阶段是由GPU固定实现的，开发者没有任何控制权。实线表示该shader必须由开发者编程实现，虚线表示该Shader是可选的

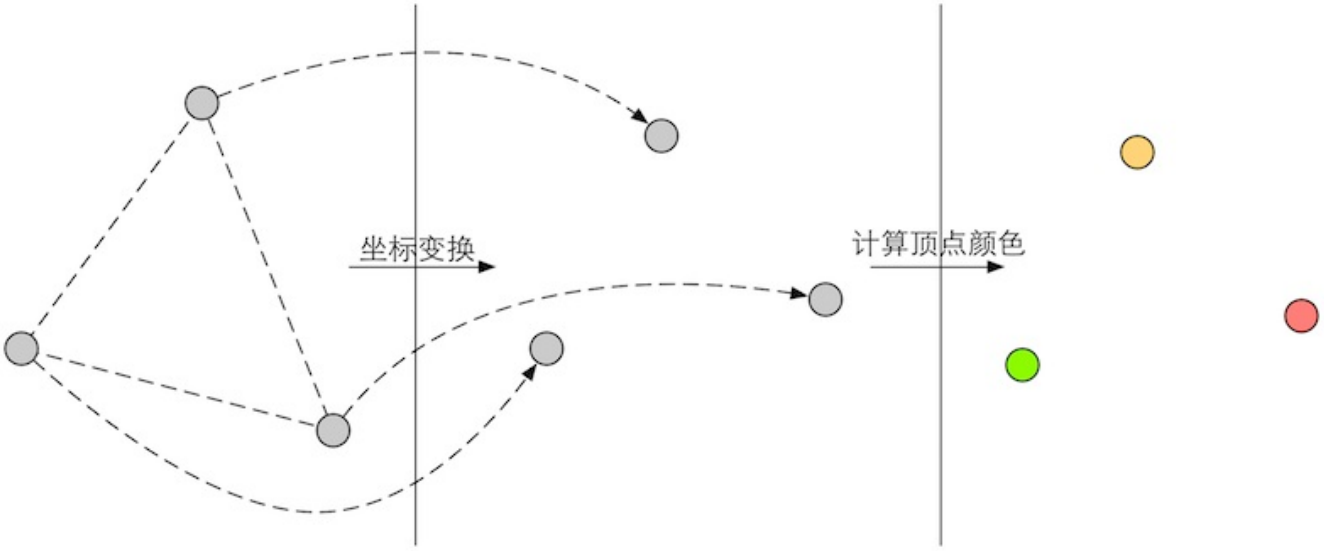


图2.7 GPU在每个输入的网格顶点上都会调用顶点着色器。顶点着色器必须进行顶点的坐标变换，需要时还可以计算和输出顶点的颜色。例如，我们可能需要进行逐顶点的光照

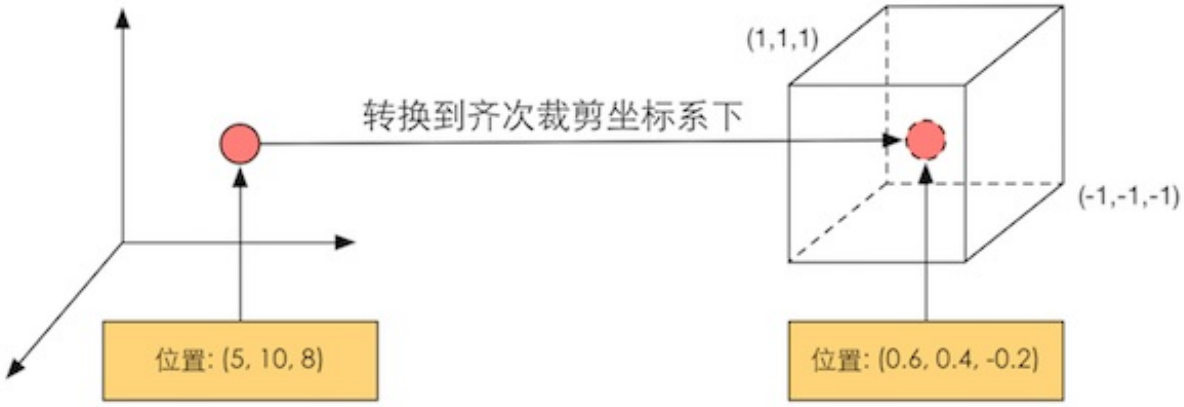


图2.8 顶点着色器会将模型顶点的位置变换到齐次裁剪坐标空间下，进行输出后再由硬件做透视除法得到NDC下的坐标

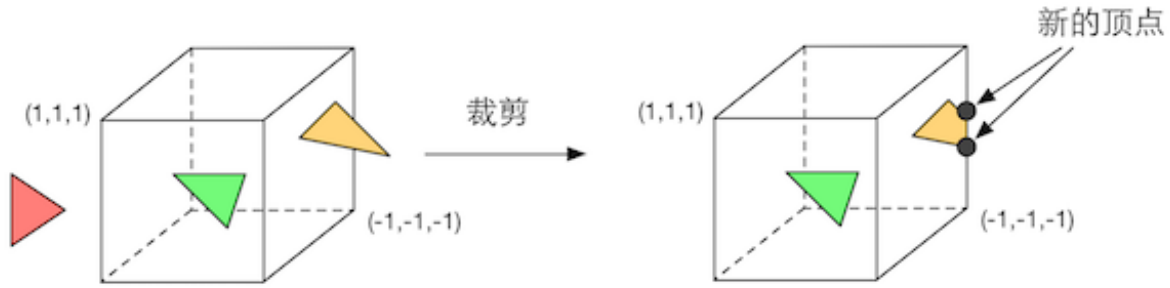


图2.9 只有在单位立方体的图元才需要被继续处理。因此，完全在单位立方体外部的图元（红色三角形）被舍弃，完全在单位立方体内部的图元（绿色三角形）将被保留。和单位立方体相交的图元（黄色三角形）会被裁剪，新的顶点会被生成，原来在外部的顶点会被舍弃

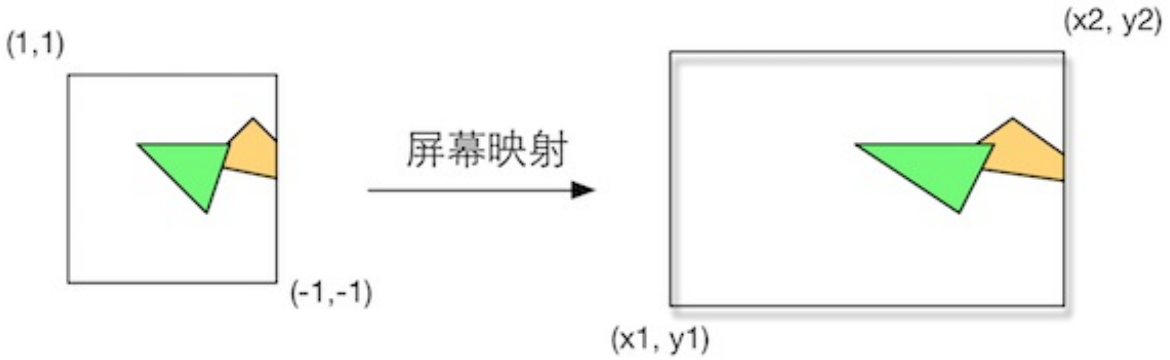


图2.10 屏幕映射将x、y坐标从（-1, 1）范围转换到屏幕坐标系中

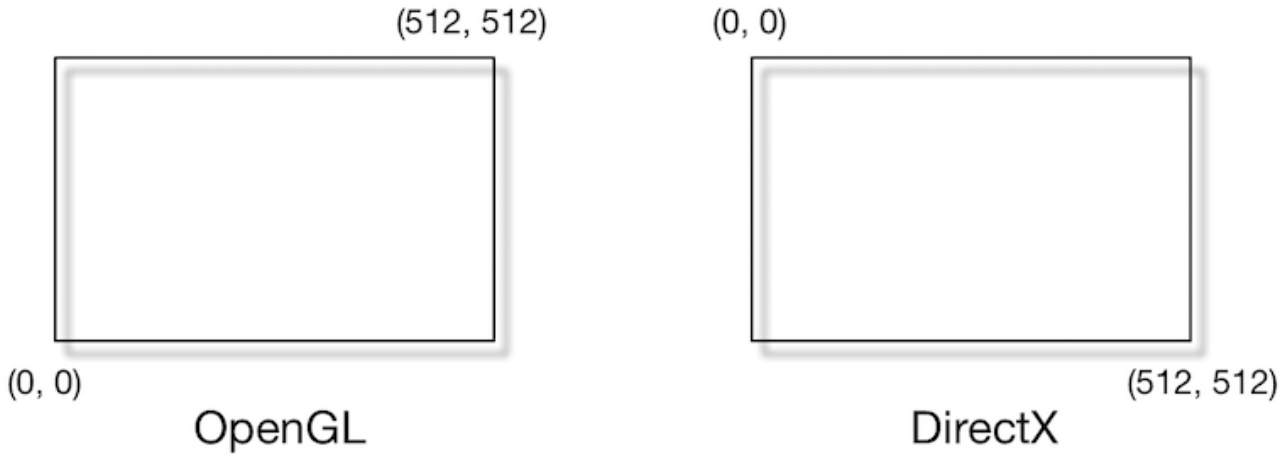


图2.11 OpenGL和DirectX的屏幕坐标系差异。对于一张512*512大小的图像，在OpenGL中其（0, 0）点在左下角，而在DirectX中其(0, 0)点在左上角

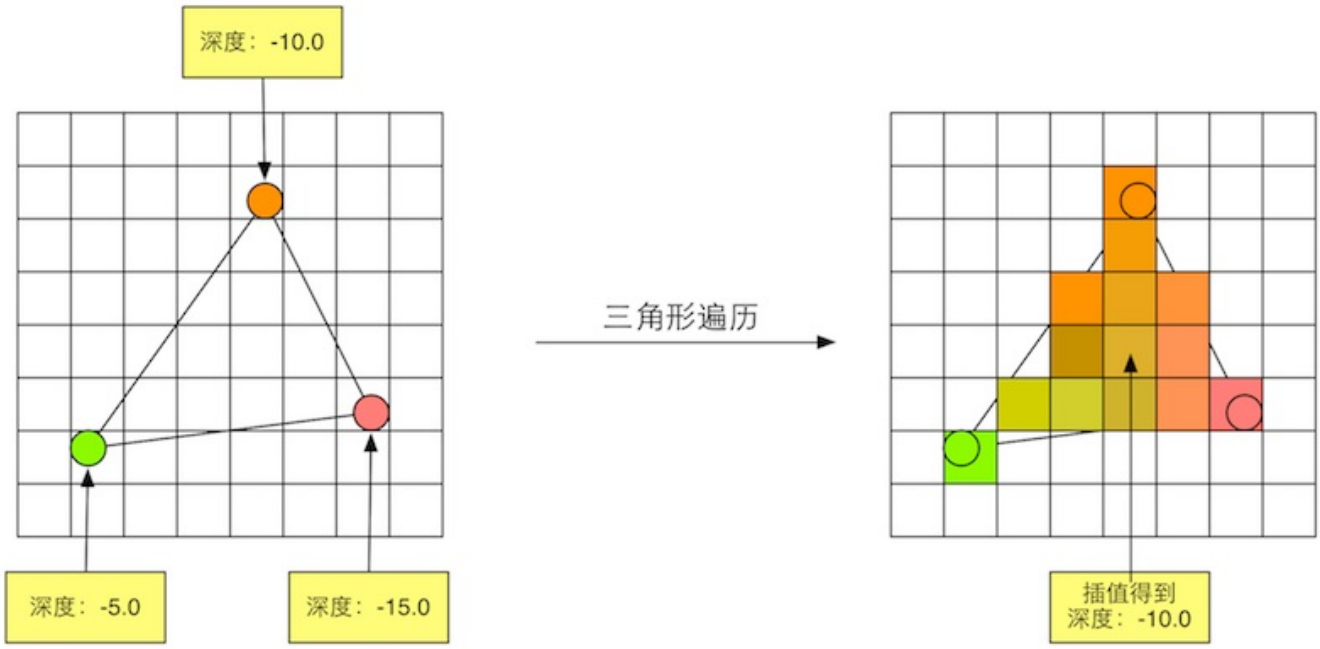


图2.12 三角形遍历的过程。根据几何阶段输出的顶点信息，最终得到该三角网格覆盖的像素位置。对应像素会生成一个片元，而片元中的状态是对三个顶点的信息进行插值得到的。例如，对图2.12中三个顶点的深度进行插值得到其重心位置对应的片元的深度值为-10.0

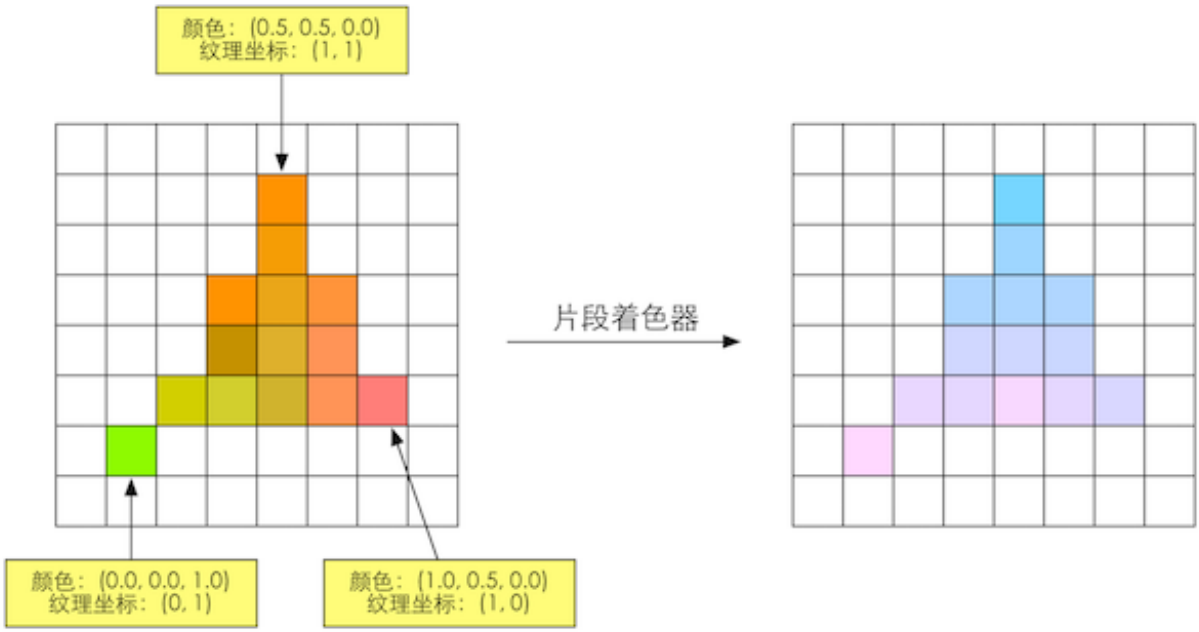


图2.13 根据上一步插值后的片元信息，片元着色器计算该片元的输出颜色

图2.14 逐片元操作阶段所做的操作。只有通过了所有的测试后，新生成的片元才能和颜色缓冲区中已经存在的像素颜色进行混合，最后再写入颜色缓冲区中

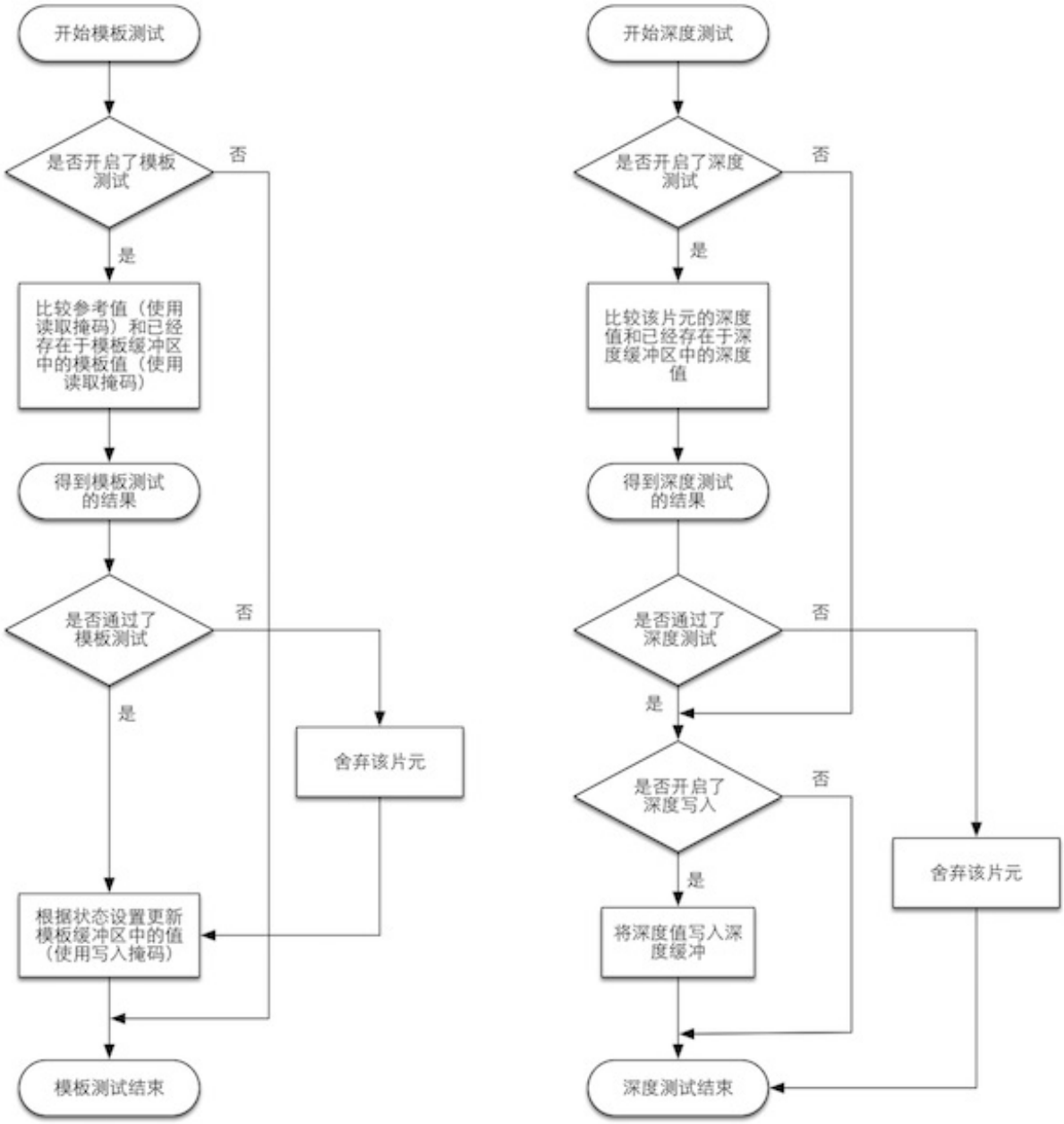


图2.15 模板测试和深度测试的简化流程图。

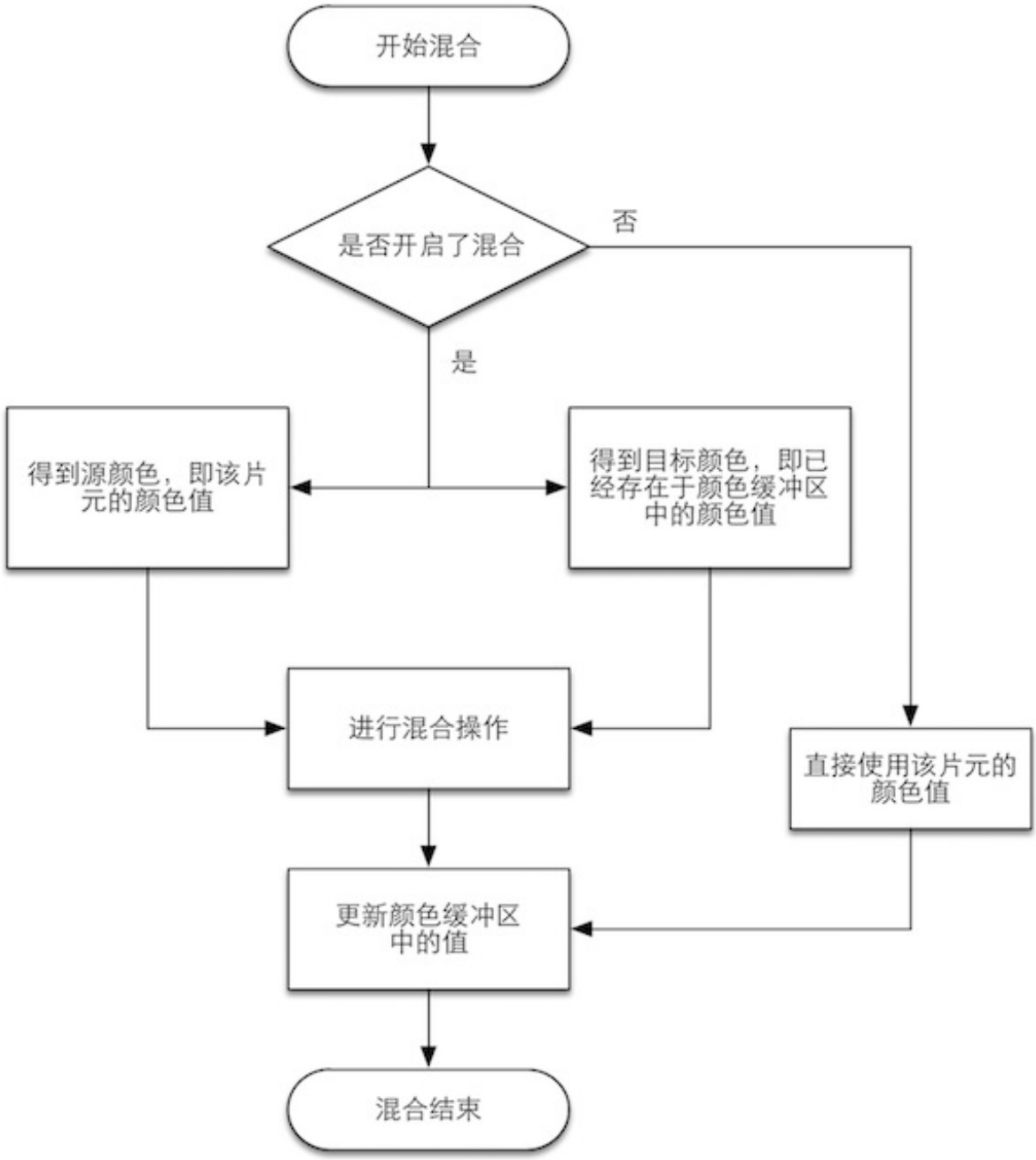


图2.16 混合操作的简化流程图

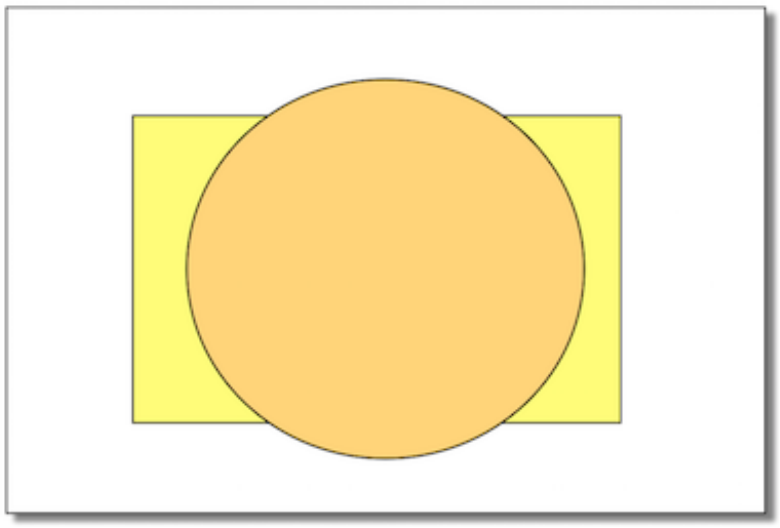


图2.17 图示场景中包含了两个对象：球和长方体，绘制顺序是先绘制球（在屏幕上显示为圆），再绘制长方体（在屏幕上显示为长方形）。如果深度测试在片元着色器之后执行，那么在渲染长方体时，虽然它的大部分区域都被遮挡在球的后面，即它所覆盖的绝大部分片元根本无法通过深度测试，但是我们仍然需要对这些片元执行片元着色器，造成了很大的性能浪费

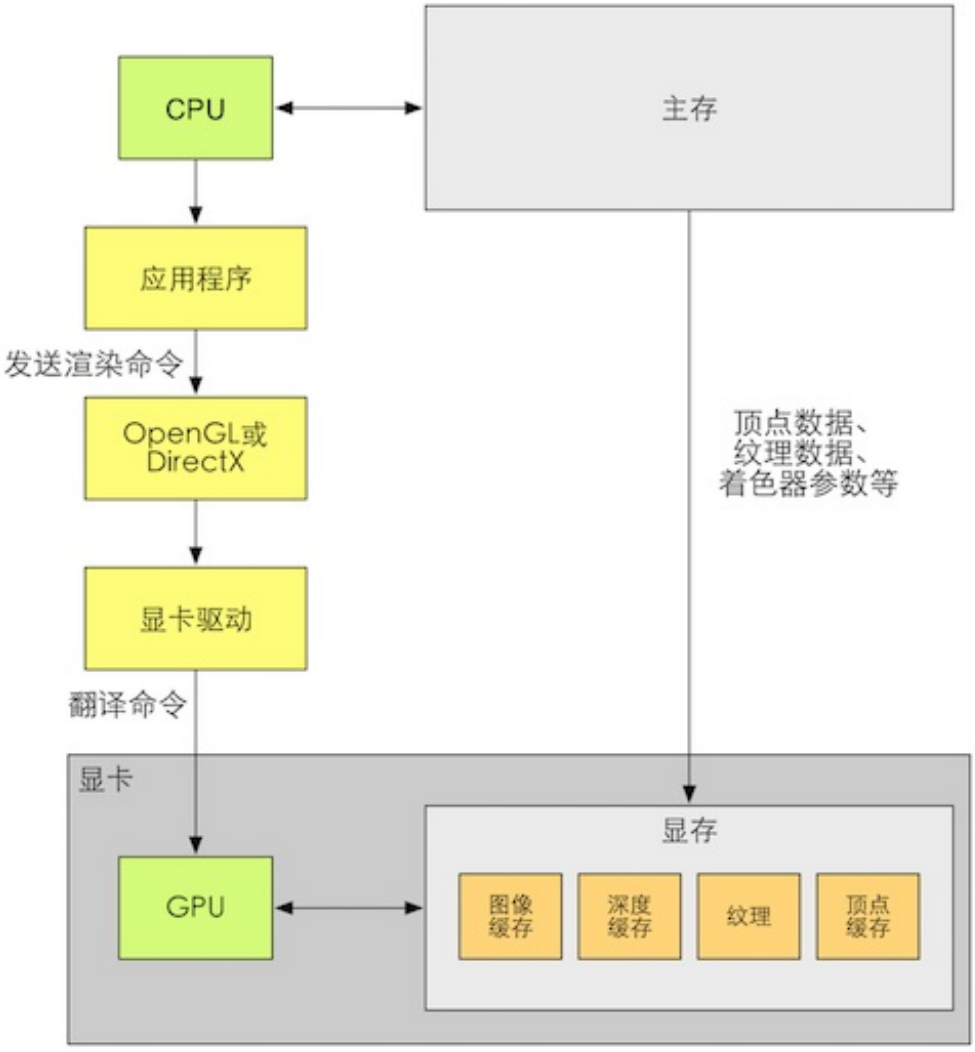


图2.18 CPU、OpenGL/DirectX、显卡驱动和GPU之间的关系

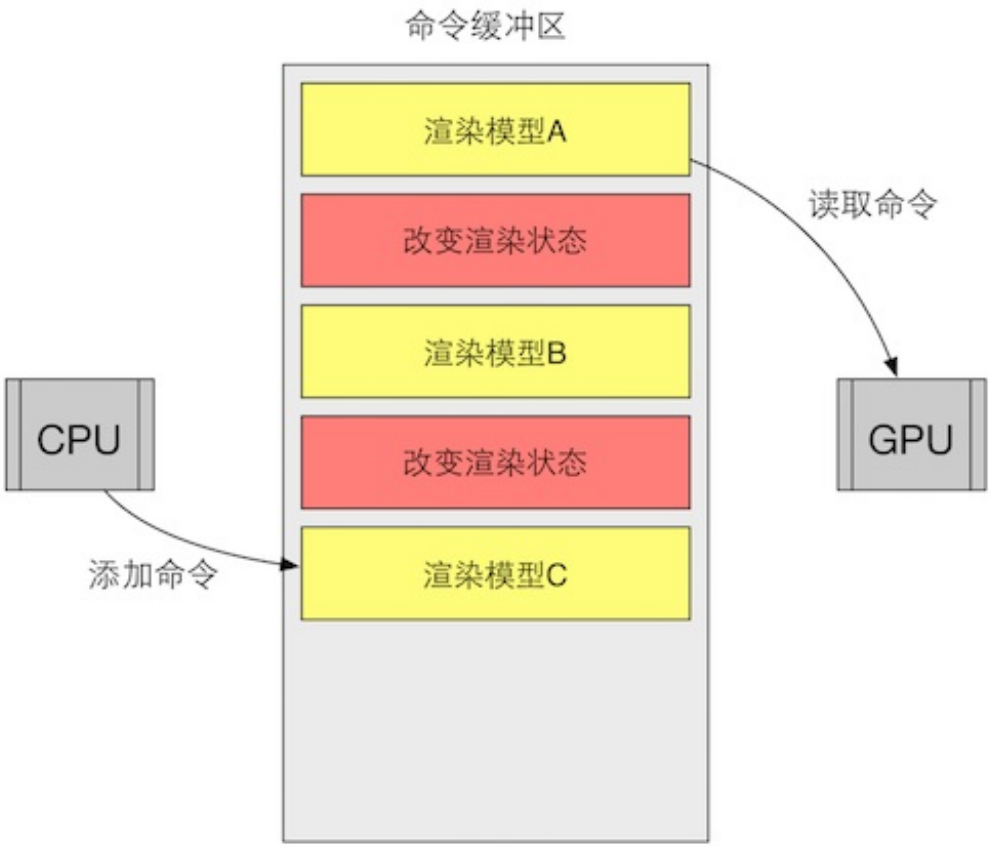


图2.19 命令缓冲区。CPU通过图像编程接口向命令缓冲区中添加命令，而GPU从中读取命令并执行。黄色方框内的命令就是Draw Call，而红色方框内的命令用于改变渲染状态。我们使用红色方框来表示改变渲染状态的命令，□是因为这些命令往往更加耗时

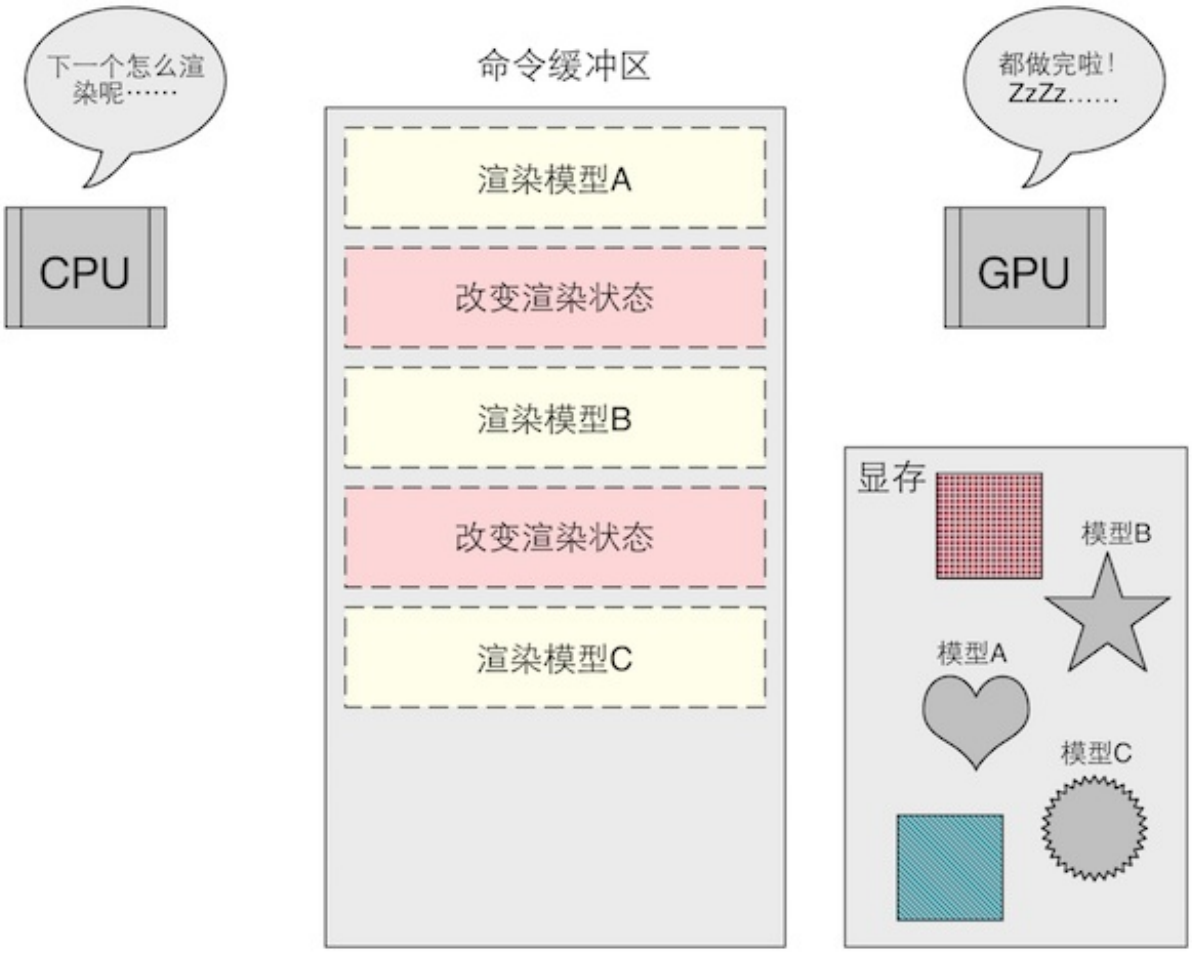


图2.20 命令缓冲区中的虚线方框表示GPU已经完成的命令。此时，命令缓冲区中没有可以执行的命令了，GPU处于空闲状态，而CPU还没有准备好下一个渲染命令。

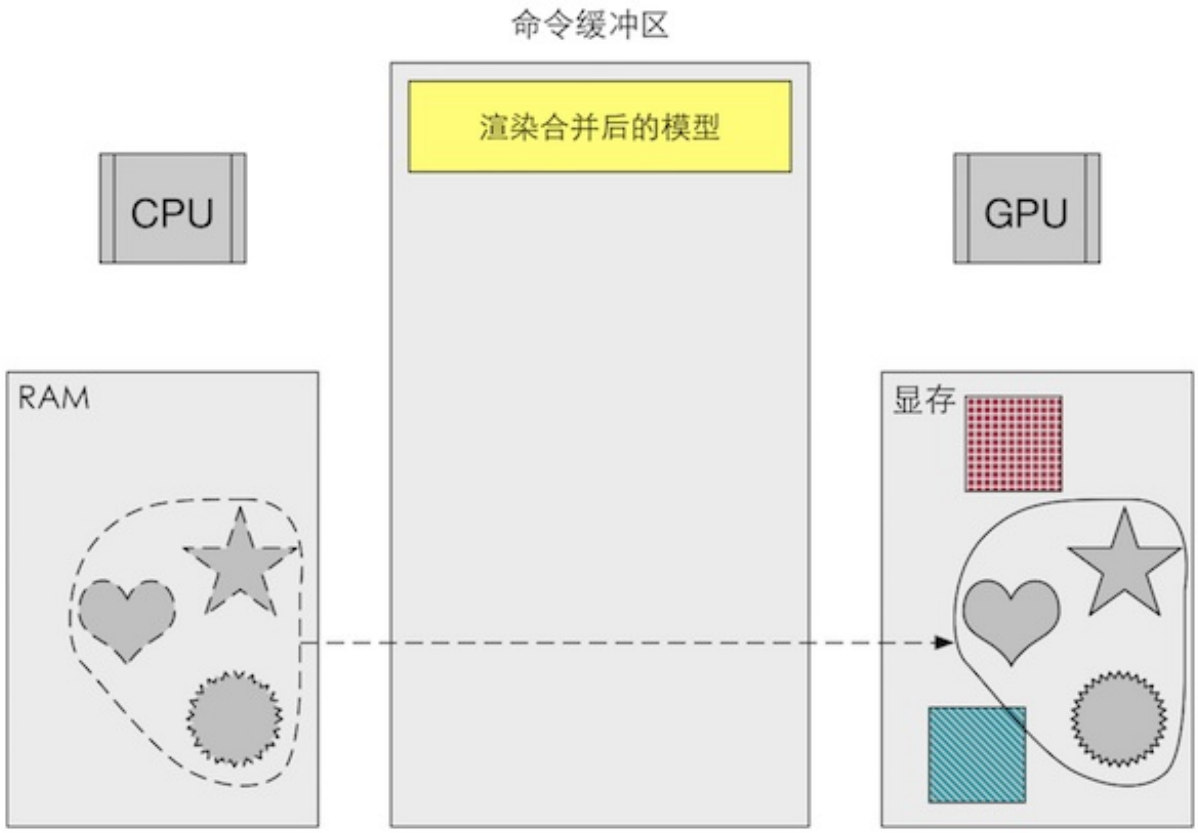


图2.21 利用批处理，CPU在RAM把多个网格合并成一个更大的网格，再发送给GPU，然后在一个Draw Call中渲染它们。但要注意的是，使用批处理合并的网格将会使用同一种渲染状态。也就是说，如果网格之间需要使用不同的渲染状态，那么就无法使用批处理技术

第3章 Unity Shader基础

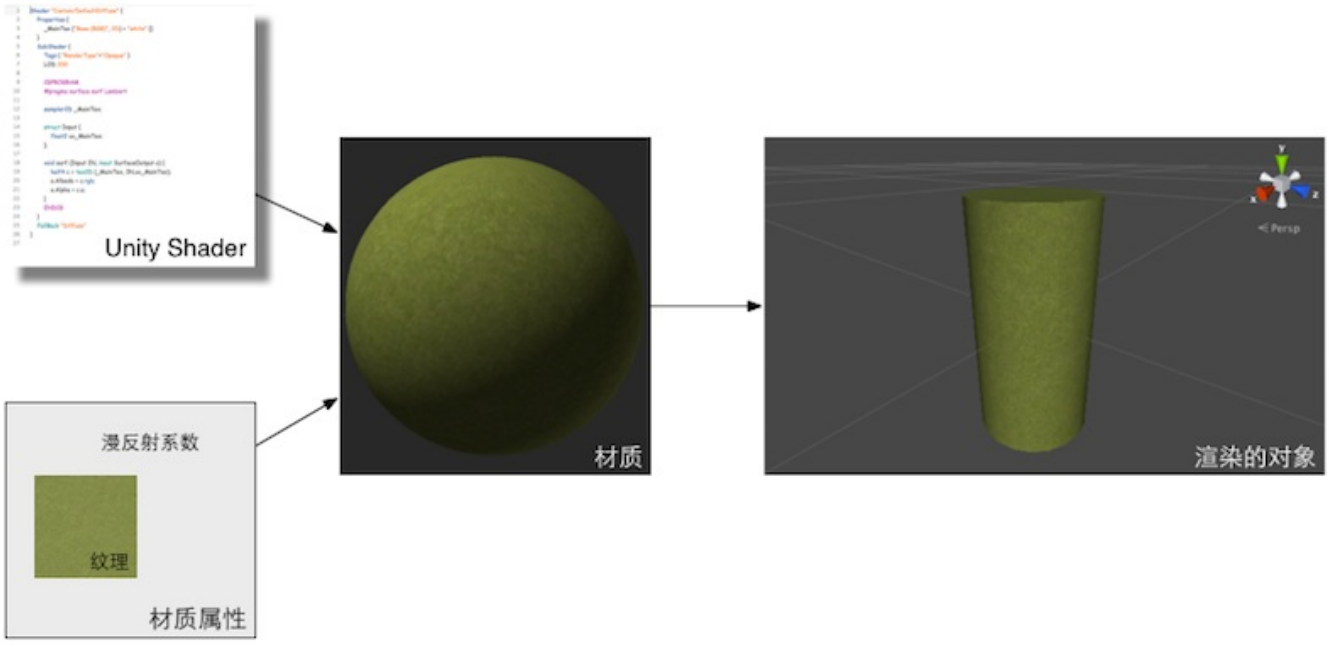


图3.1 Unity Shader和材质。首先创建需要的Unity Shader和材质，然后把Unity Shader赋给材质，并在材质面板上调整属性（如使用的纹理、漫反射系数等）。最后，将材质赋给相应的模型来查看最终的渲染效果

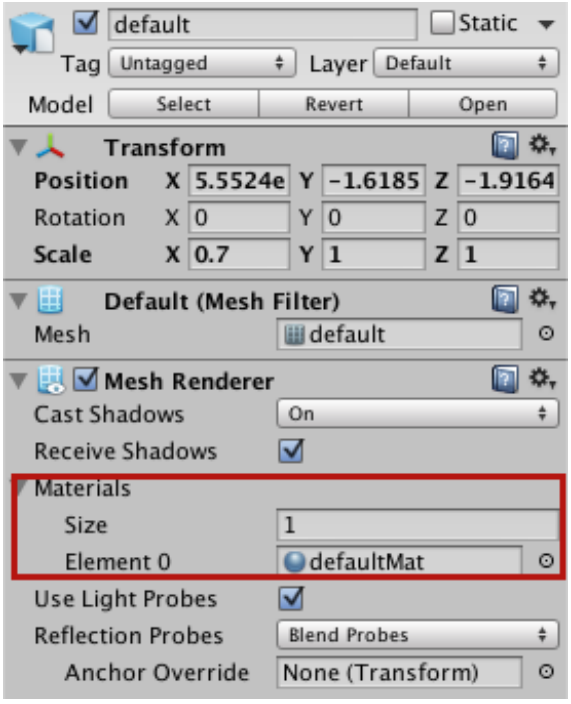


图3.2 将材质直接拖曳到模型的Mesh Renderer组件中

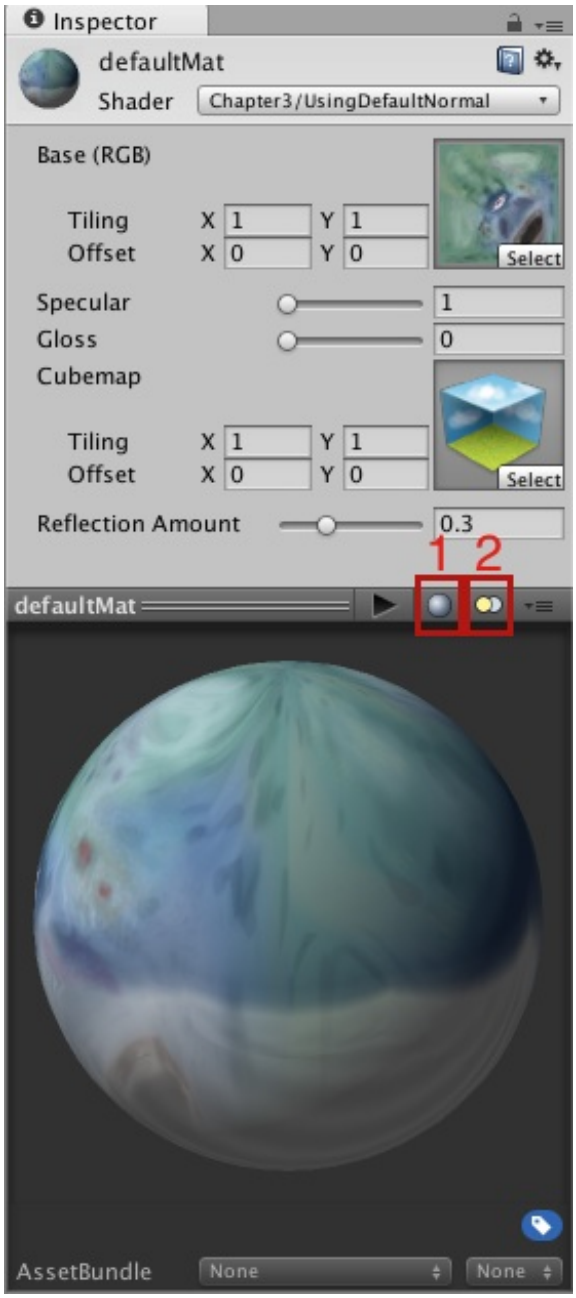


图3.3 材质提供了一种可视化的方式来调整着色器中使用的参数

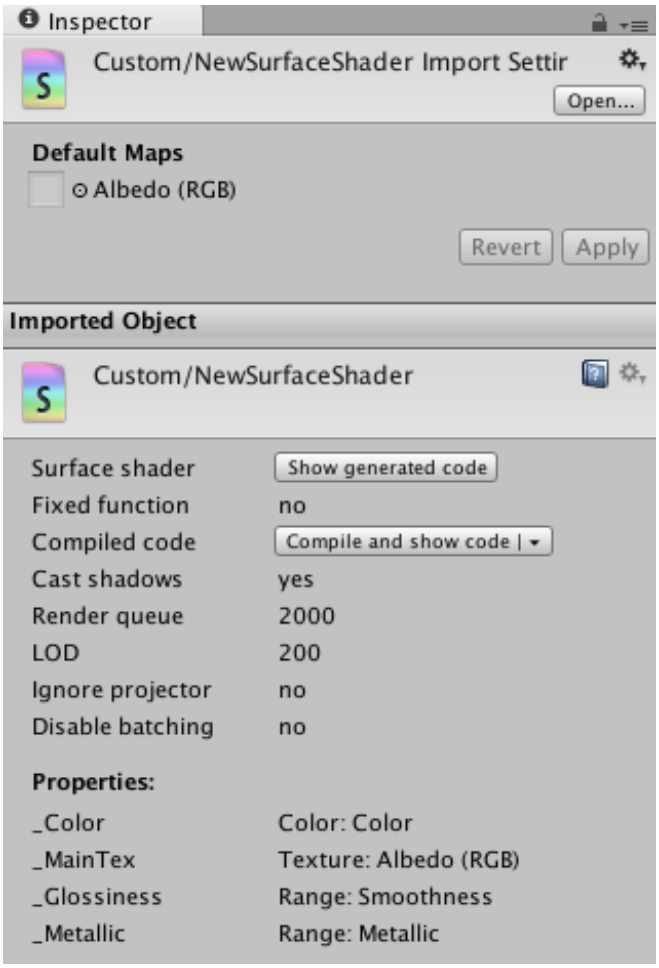


图3.4 Unity Shader的导入设置面板

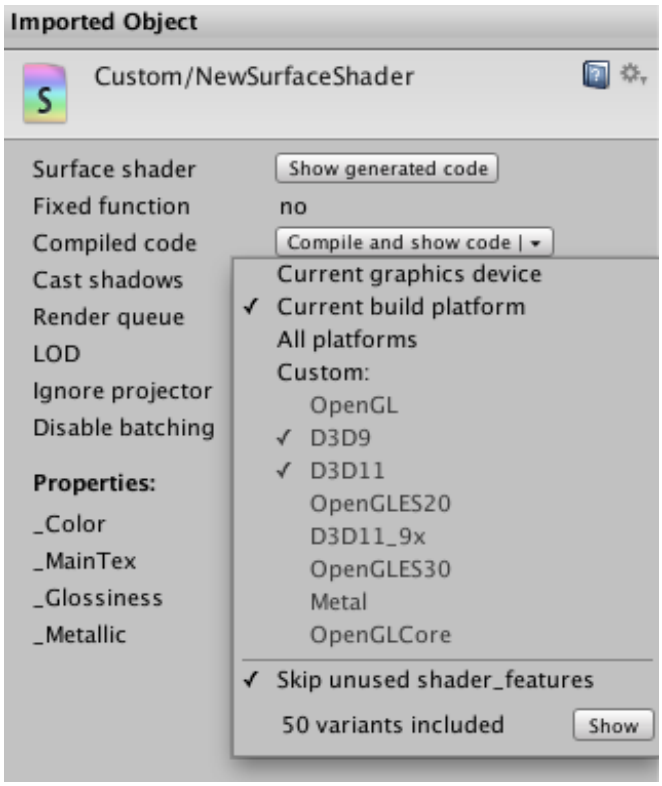


图3.5 Gompile and show code下拉列表

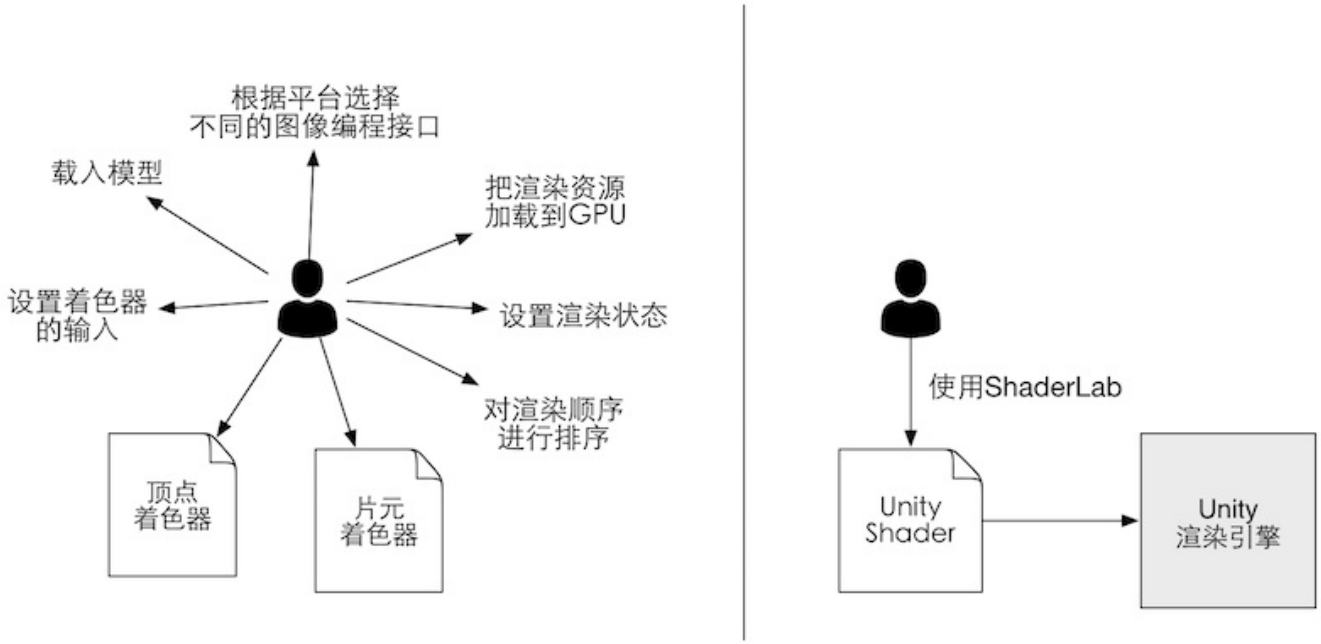


图3.6 Unity Shader为控制渲染过程提供了一层抽象。如果没有使用Unity Shader（左图），开发者需要和很多文件和设置打交道，才能让画面呈现出想要的效果；而在Unity Shader的帮助下（右图），开发者只需要使用ShaderLab来编写Unity Shader文件就可以完成所有的工作

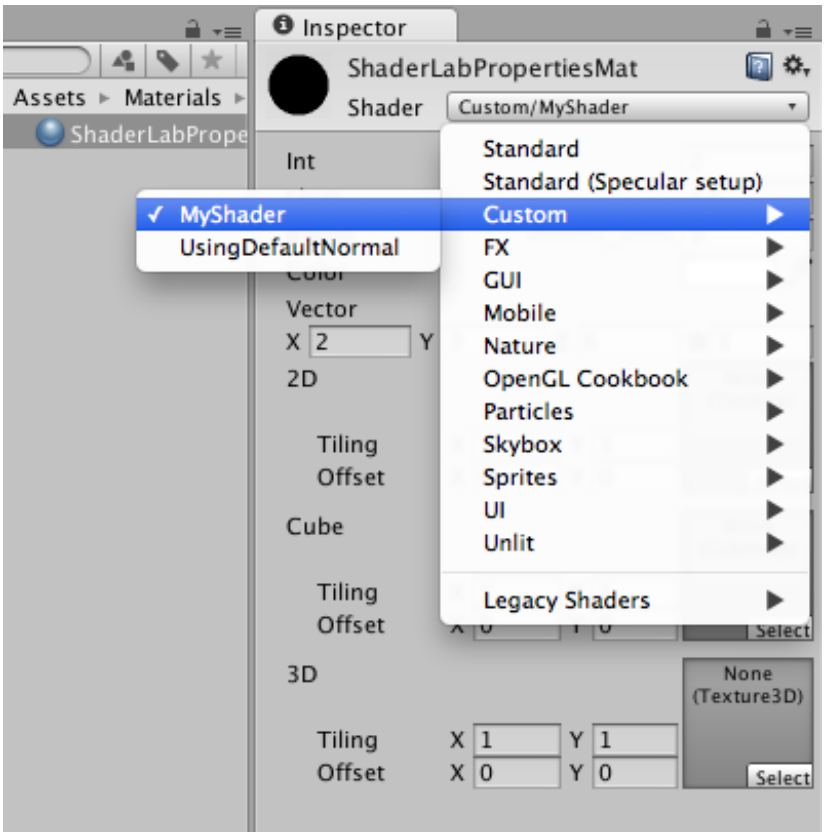


图3.7 在Unity Shader的名称定义中利用□斜杠来组织在材质面板中的位置

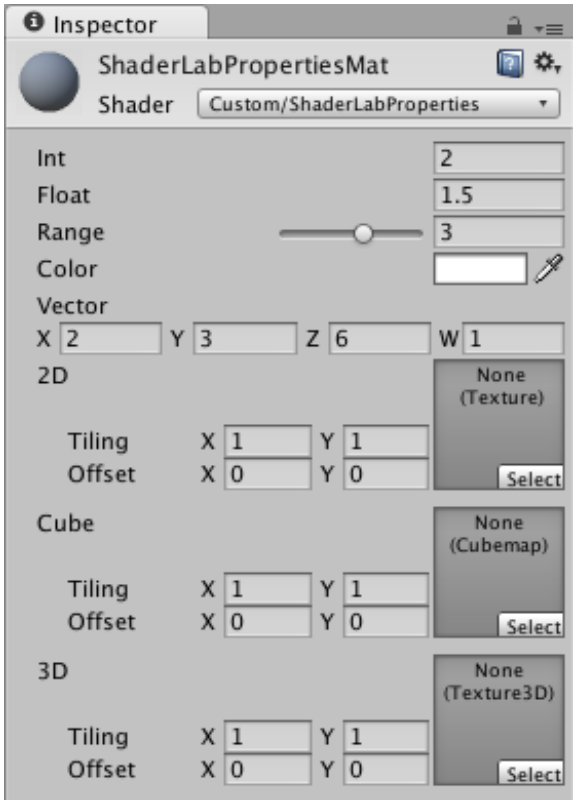


图3.8 不同属性类型在材质□面板中的显示结果

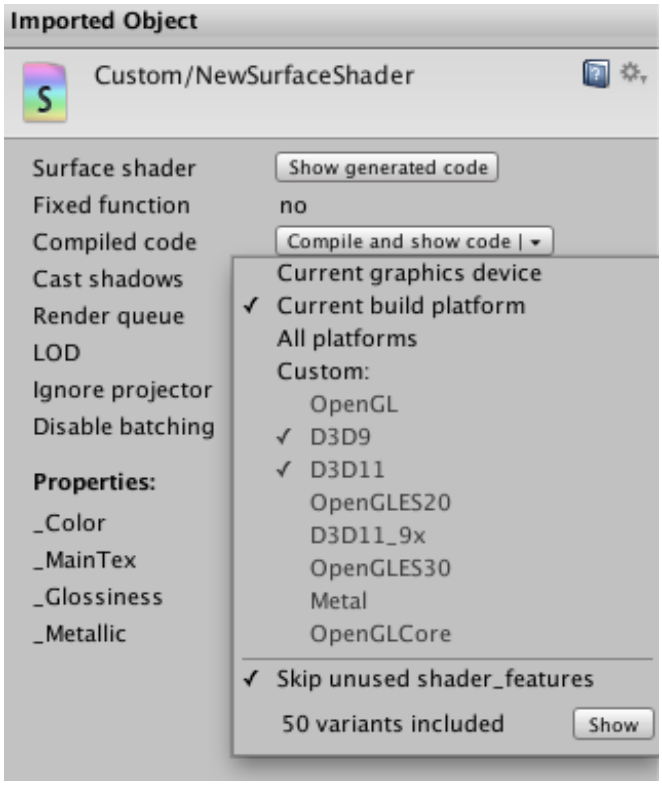


图3.9 在Unity Shader的导入设置面板中可以通过Compile and show code按钮来查看Unity

对CG片段编译后的代码。通过单击Compile and show code按钮右端的倒三角可以打开下拉菜单，在这个下拉菜单中可以选择编译的平台种类，如只为当前的显卡设备编译特定的汇编代码，或为所有的平台编译汇编代码，我们也可以自定义选择编译到哪些平台上

第4章 学习Shader所需的数学基础

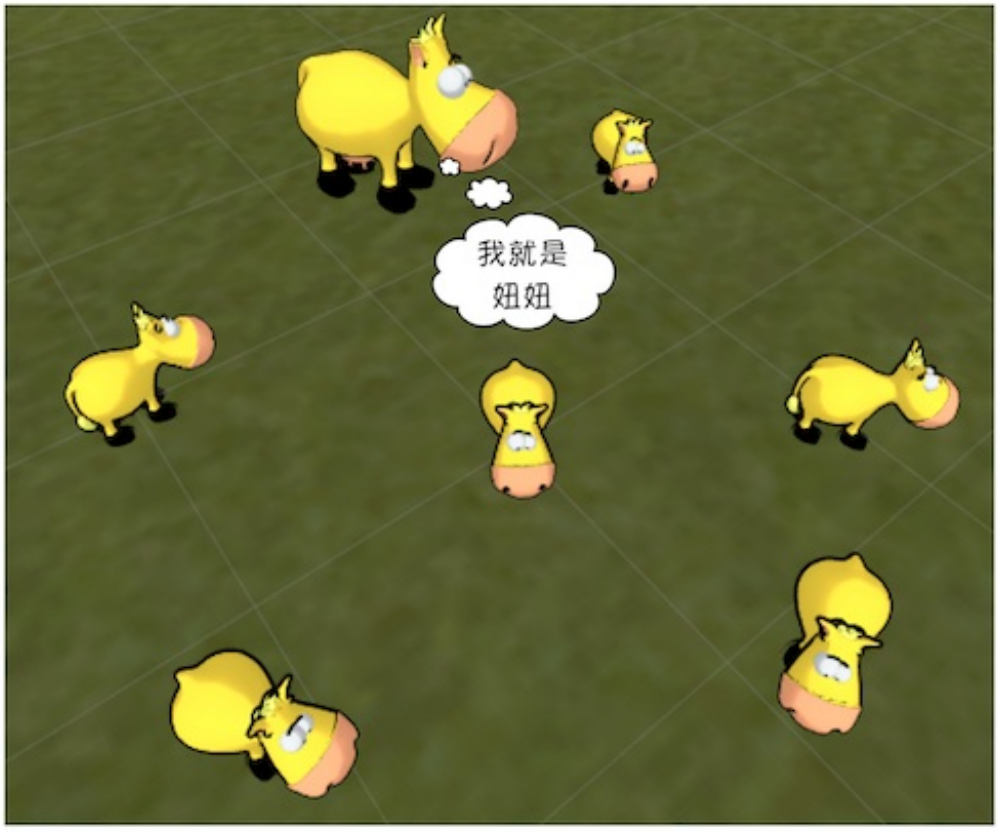


图4.1 我们的农场游戏。我们的主角妞妞是一头长得最壮、好奇心很强的奶牛

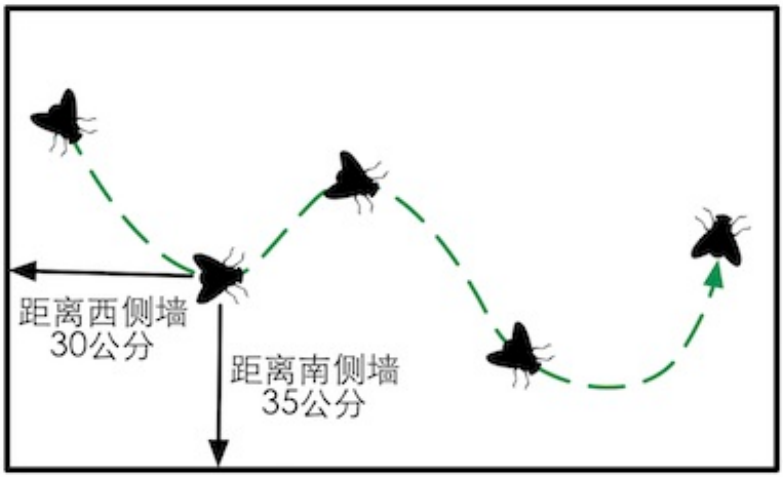


图4.2 传说，笛卡尔坐标系来源于笛卡尔对天花板上一只苍蝇的运动轨迹的观察。笛卡尔发

现，可以使用苍蝇距不同墙面的距离来描述它的当前位置

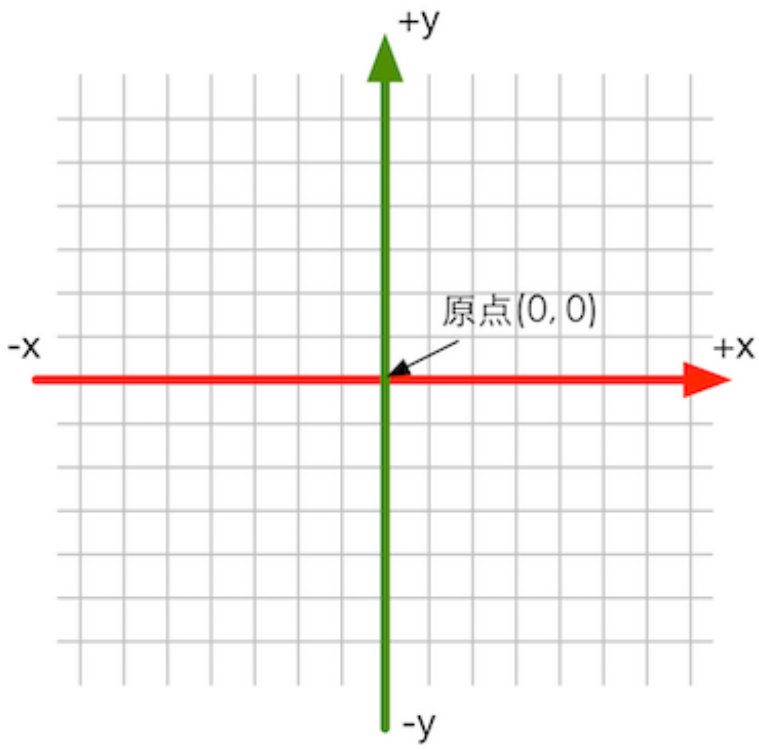


图4.3 一个二维笛卡尔坐标系

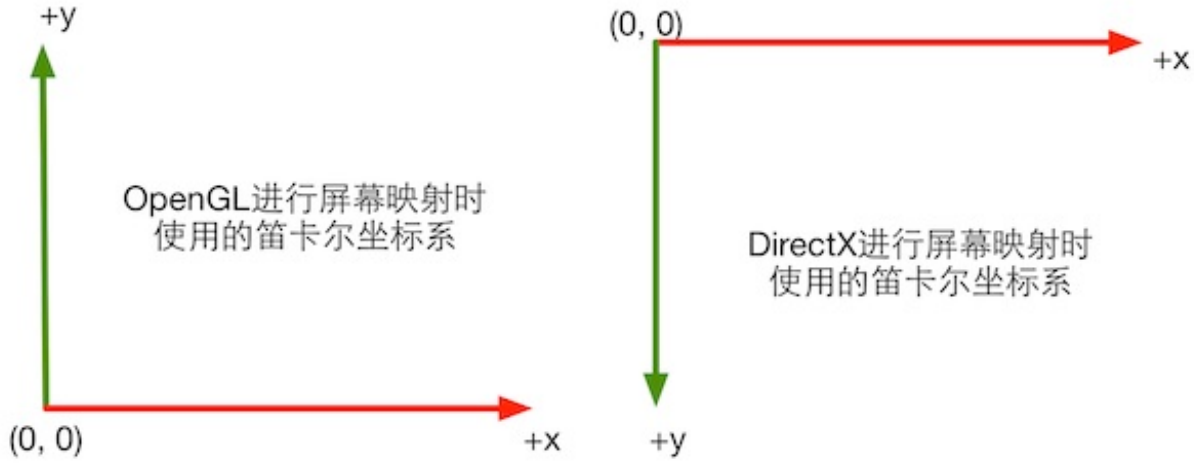


图4.4 在屏幕映射时，OpenGL和DirectX使用了不同方向的二维笛卡尔坐标系

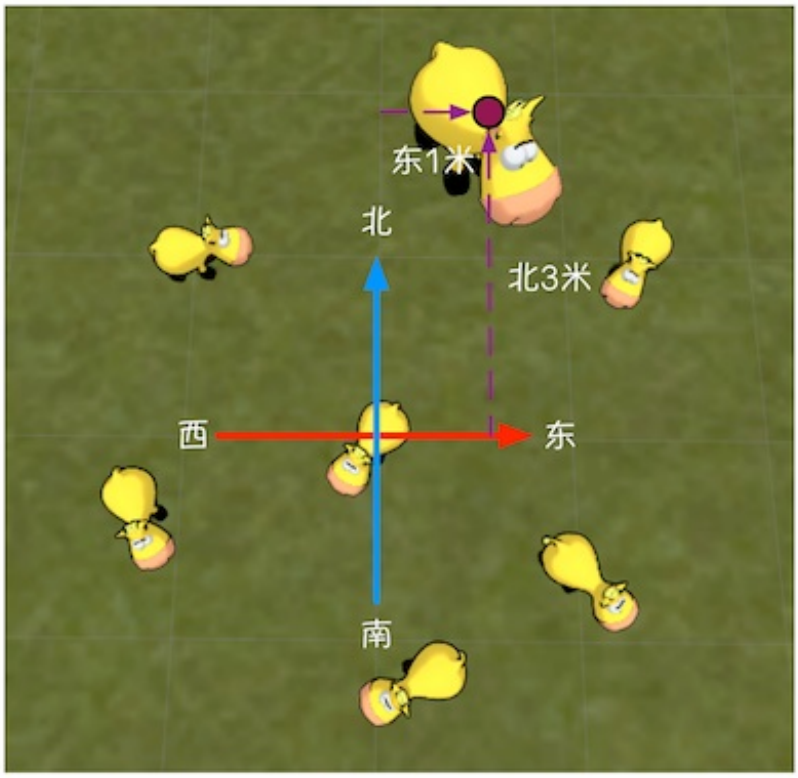


图4.5 笛卡尔坐标系可以让妞妞精确表述自己的位置

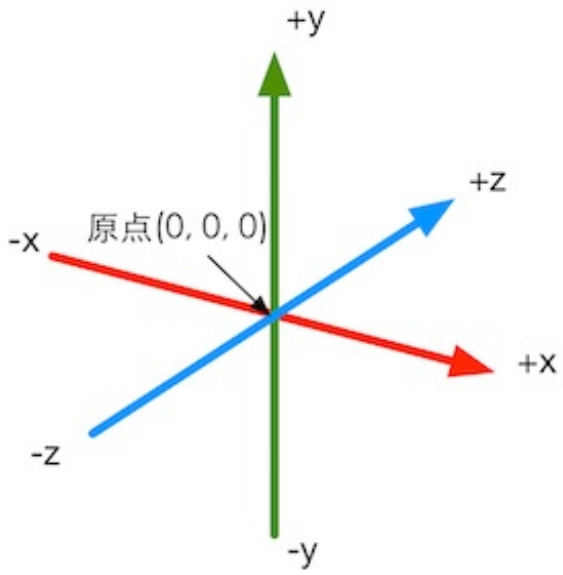


图4.6 一个三维笛卡尔坐标系

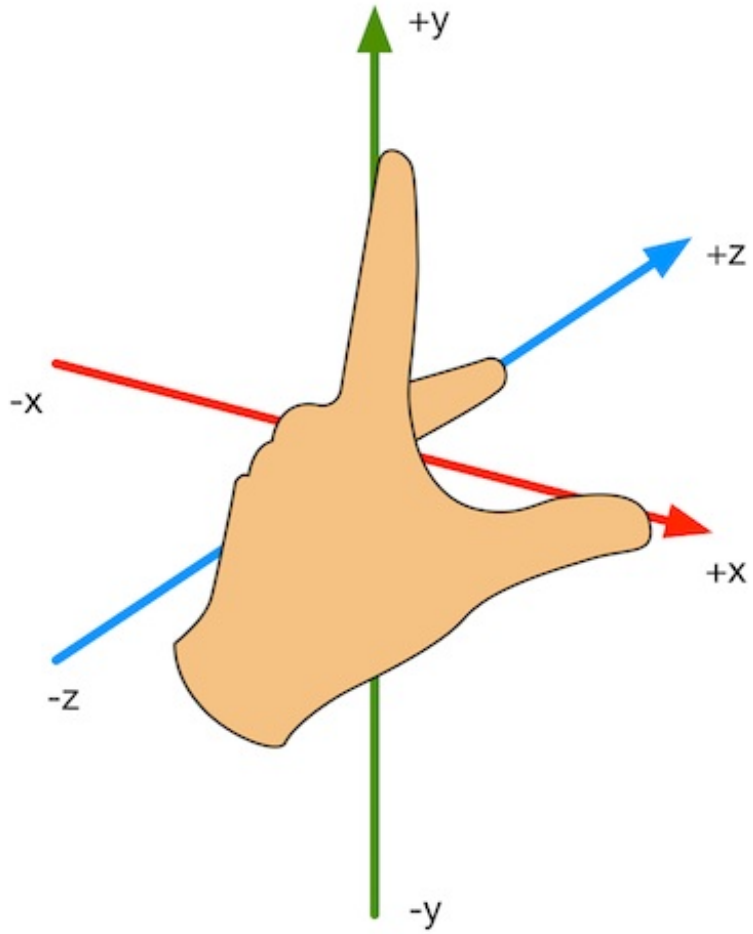


图4.7 左手坐标系

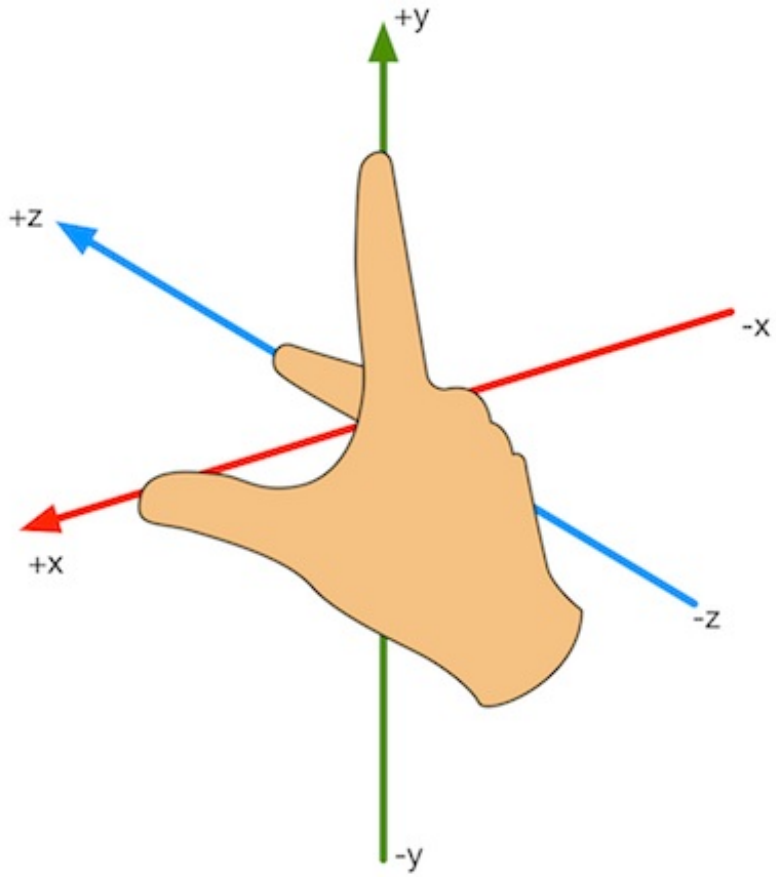


图4.8 右手坐标系

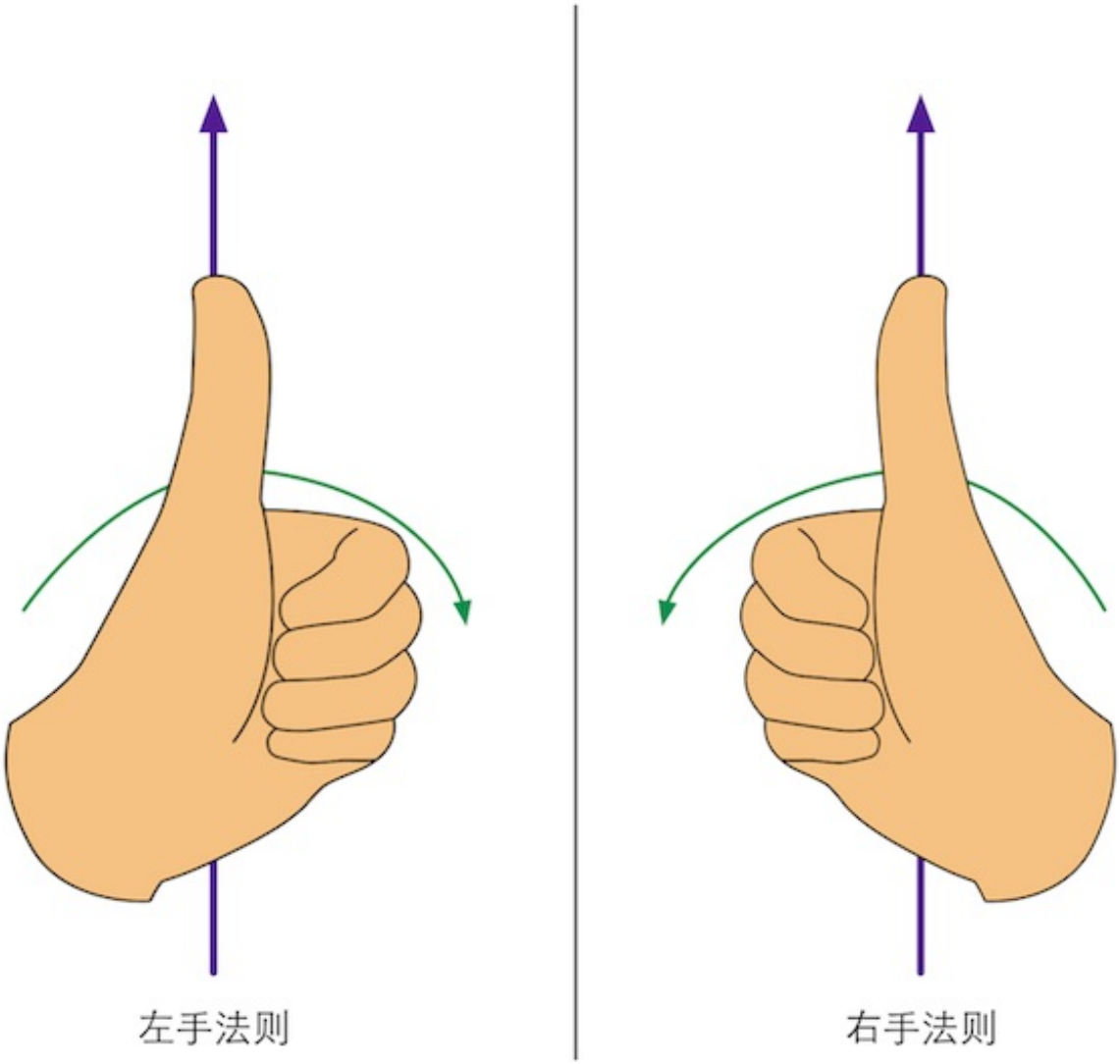


图4.9 用左手法则和右手法则来判断旋转正方向

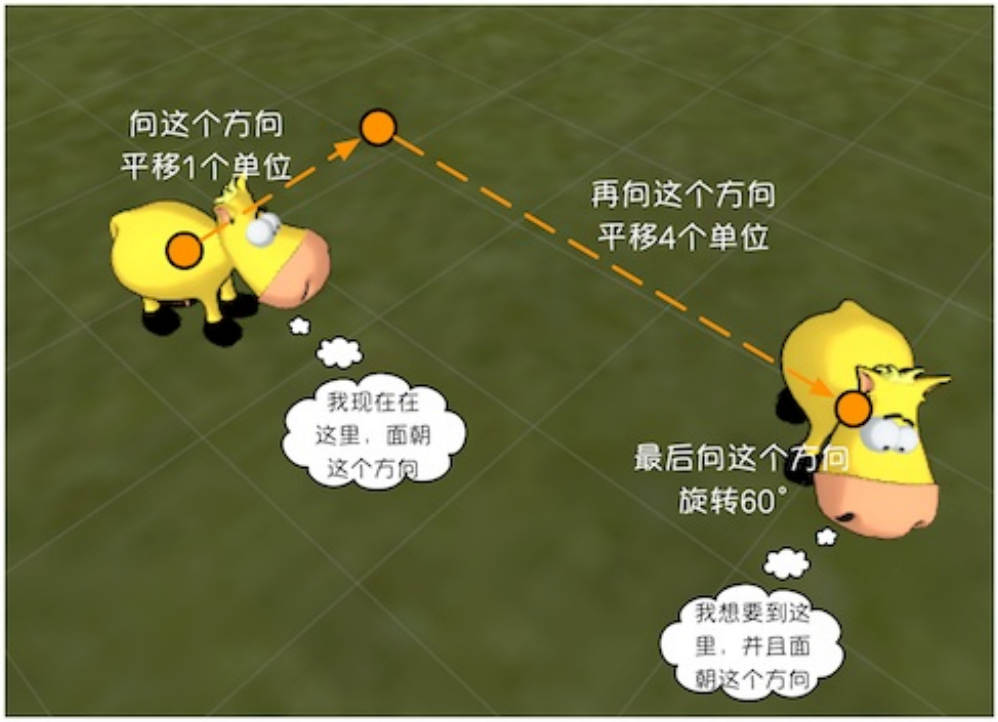


图4.10 为了移动到新的位置，妞妞需要首先向某个方向平移1个单位，再向另一个方向平移4个单位，最后再向一个方向旋转60°

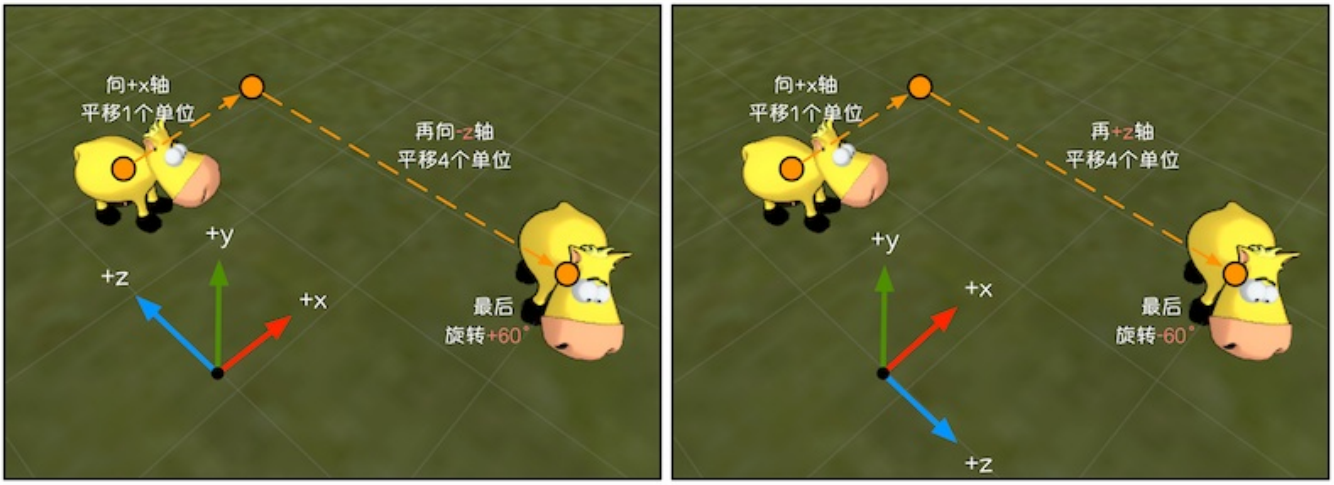


图4.11 左图和右图分别表示了在左手坐标系和右手坐标系中描述妞妞这次运动的结果，得到的数学描述是不同的

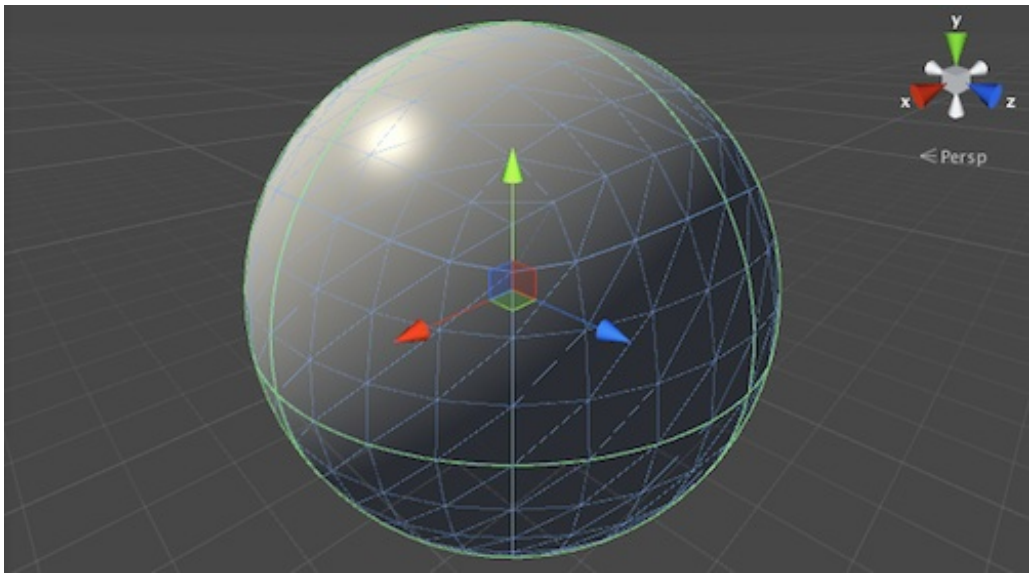


图4.12 在模型空间和世界空间中，Unity使用的是左手坐标系。图中，球的坐标轴显示了它在模型空间中的3个坐标轴（红色为x轴，绿色是y轴，蓝色是z轴）

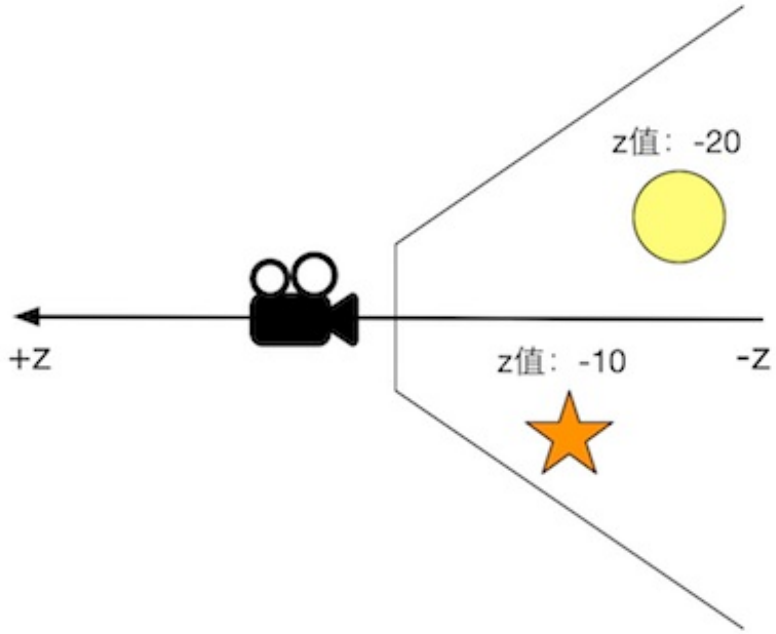


图4.13 在Unity中，观察空间使用的是右手坐标系，摄像机的前向是z轴的负方向，□z轴越小，物体的深度越大，离摄像机越远

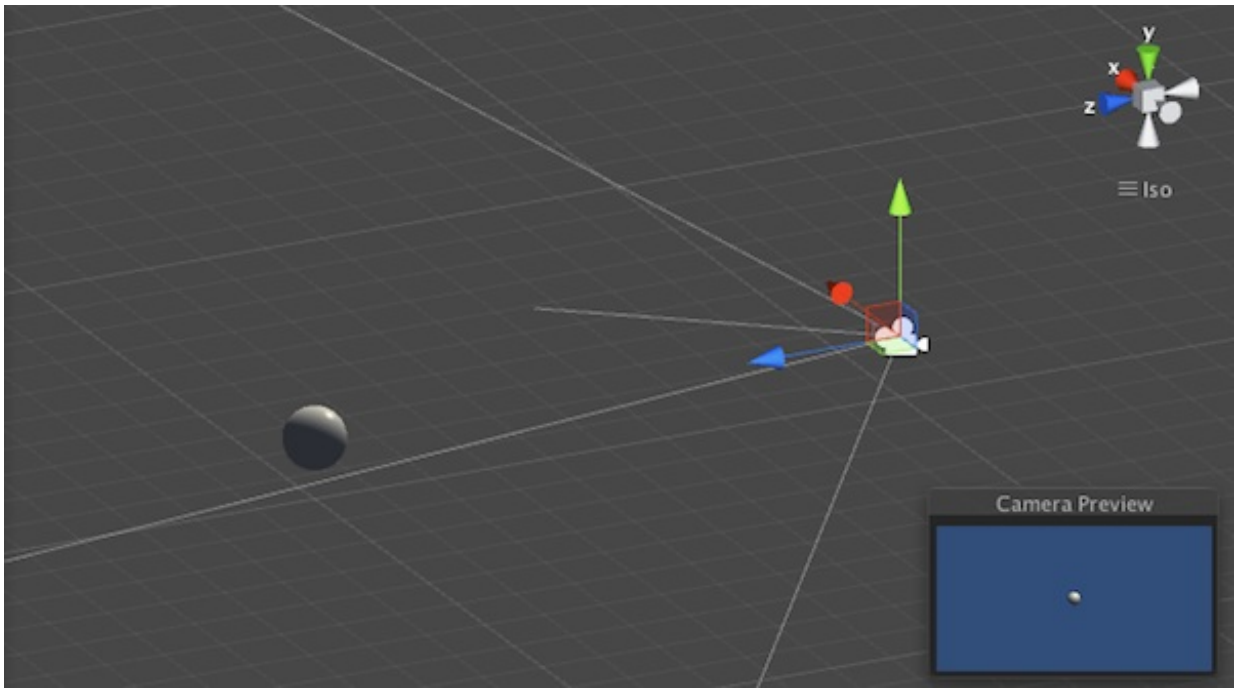


图4.14 摄像机的位置是（0, 1, -10），球体的位置是（0, 1, 0）

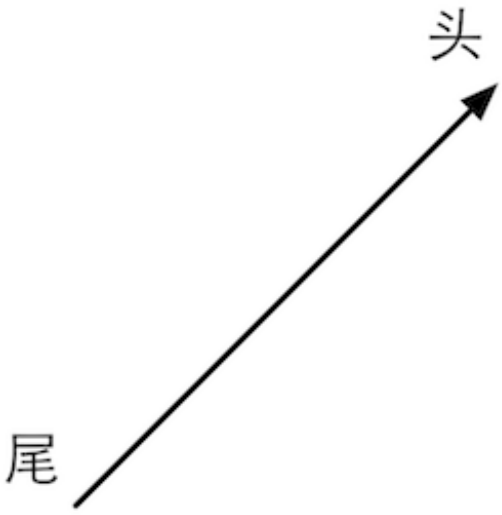


图4.15 一个二维向量以及它的头和尾

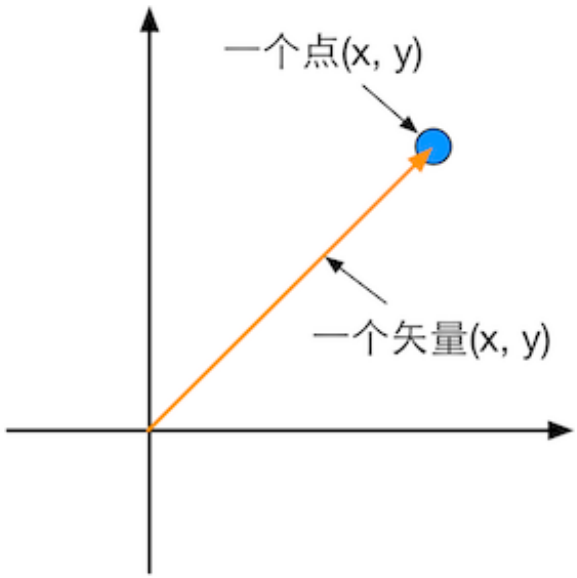


图4.16 点和矢量之间的关系

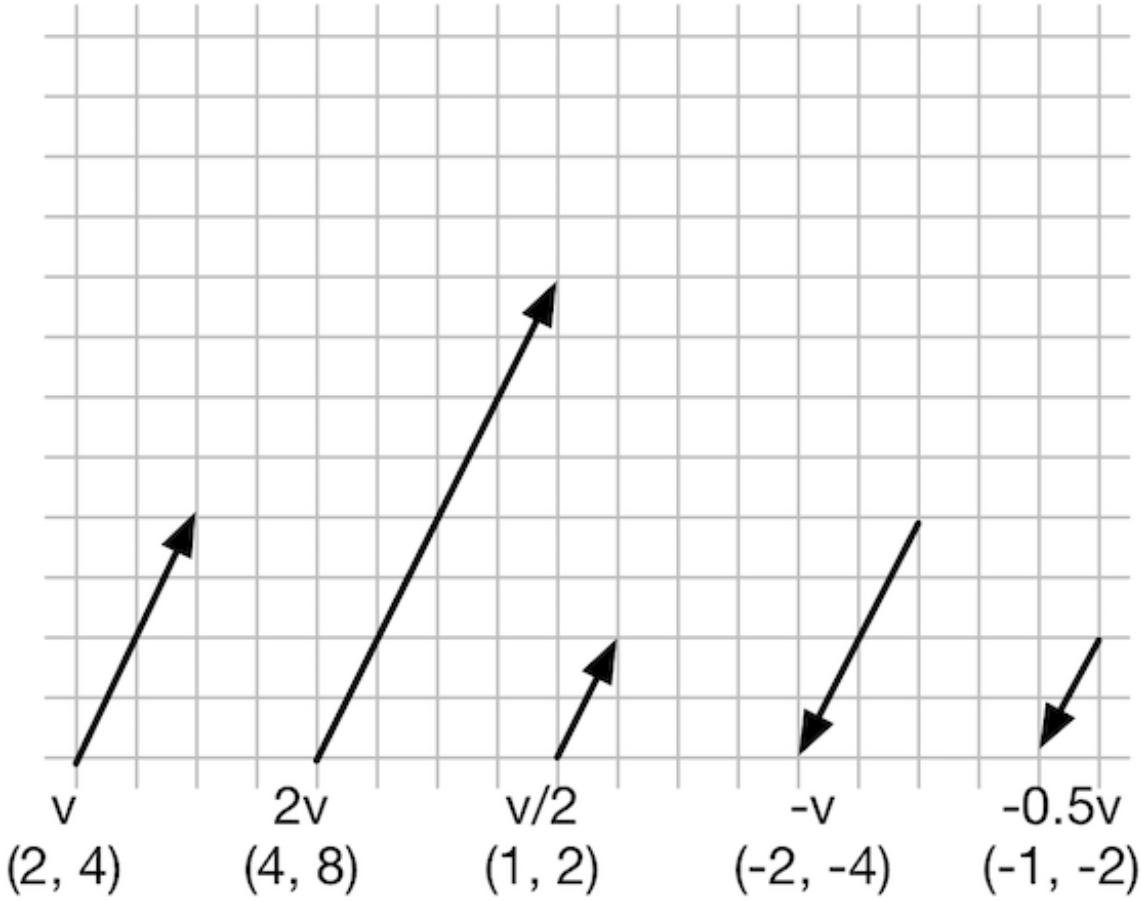


图4.17 二维矢量和一些标量的乘法和除法

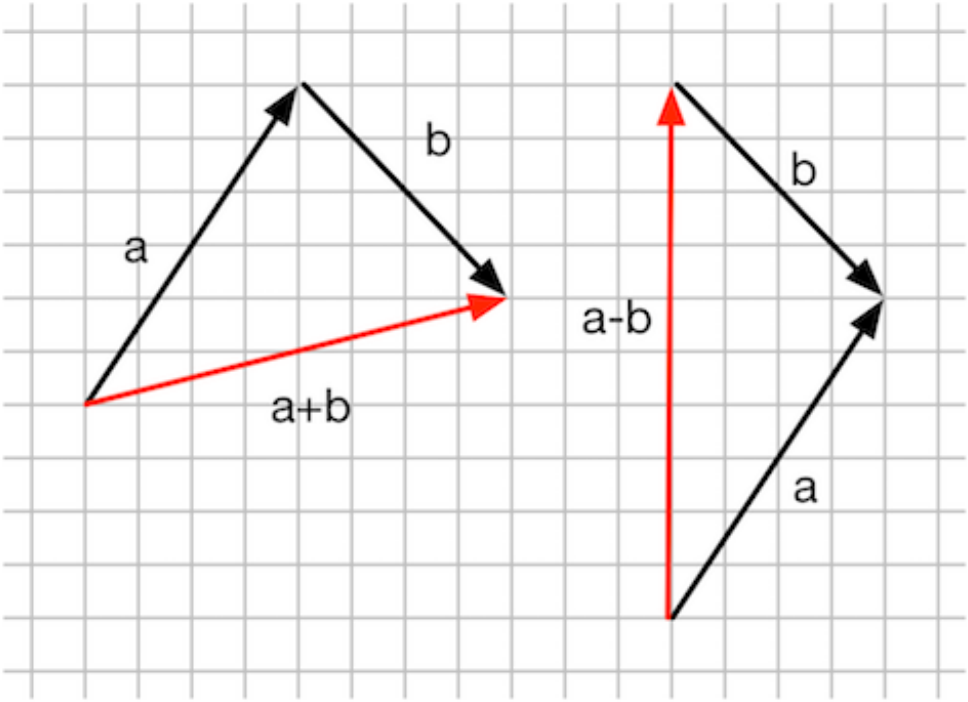


图4.18 二维矢量的加法和减法

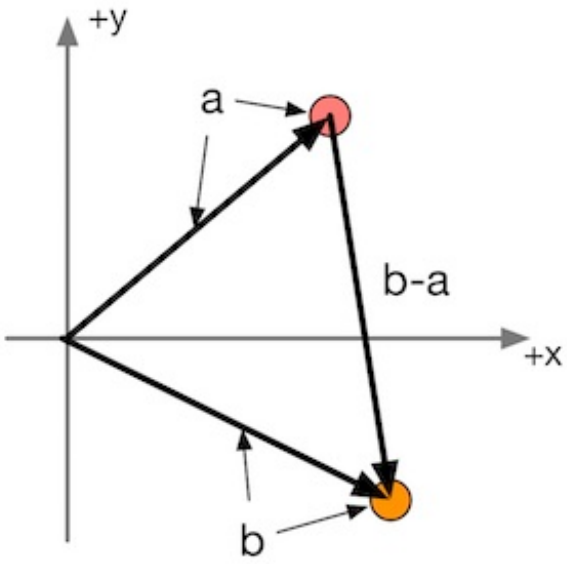


图4.19 使用矢量减法来计算从点a到点b的位移

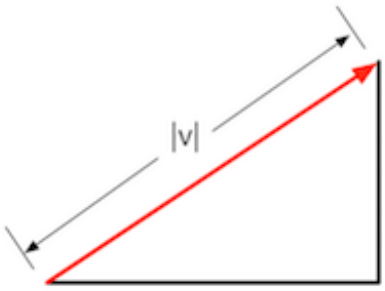


图4.20 矢量的模

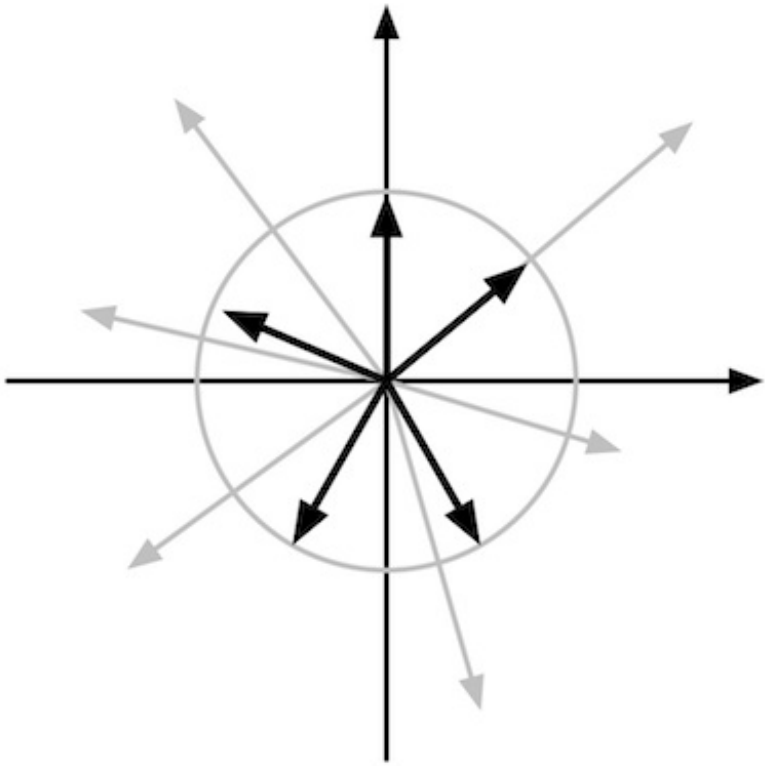


图4.21 二维空间的单位矢量都会落在单位圆上

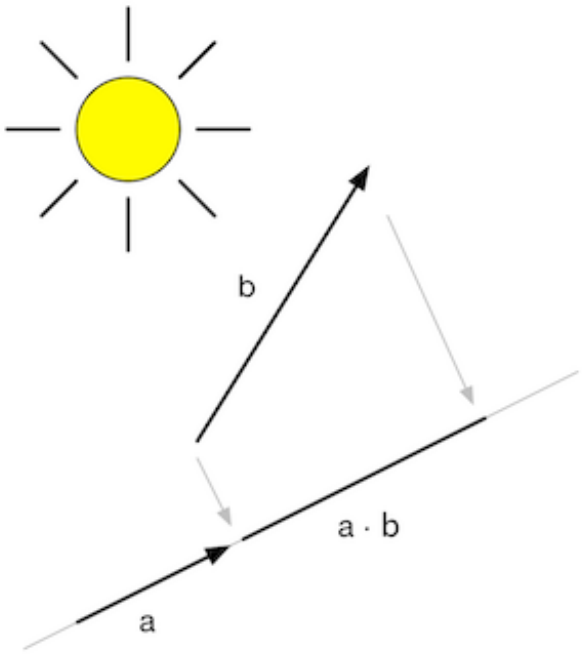


图4.22 矢量b在单位矢量a方向上的投影

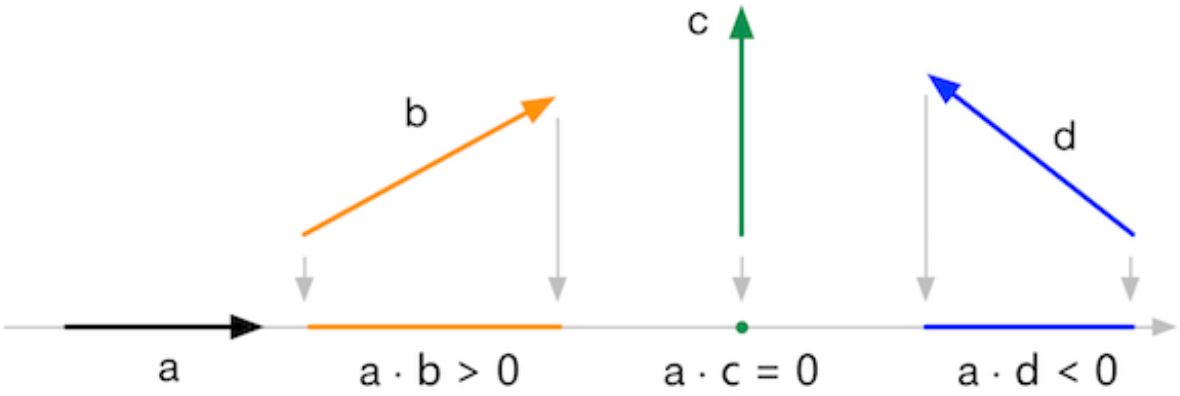


图4.23 点积的符号

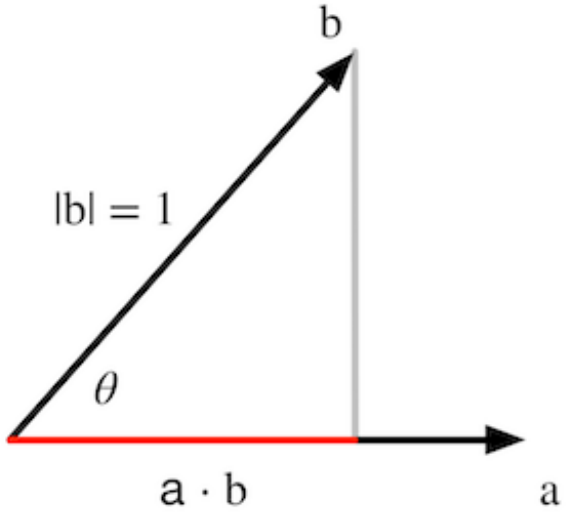


图4.24 两个单位矢量进行点积

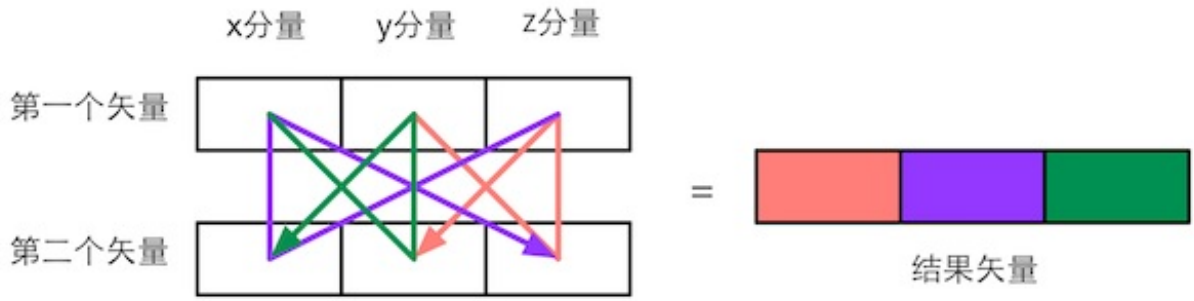


图4.25 三维矢量叉积的计算规律。不同颜色的线表示了计算结果矢量中对应颜色的分量的计算路径。以红色为例，即结果矢量的第一个分量，它是从第一个矢量的y分量出发乘以第二个矢量的z分量，再减去第一个矢量的z分量和第二矢量的y分量的乘积

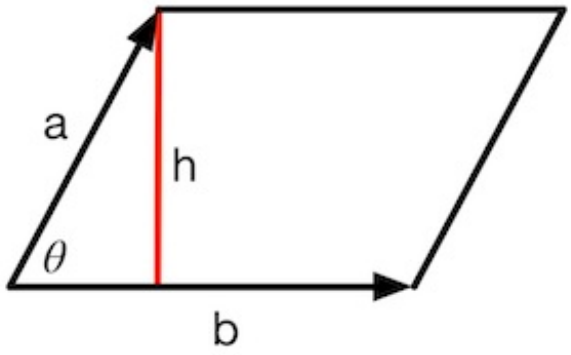


图4.26 使用矢量a和矢量b构建一个平行四边形

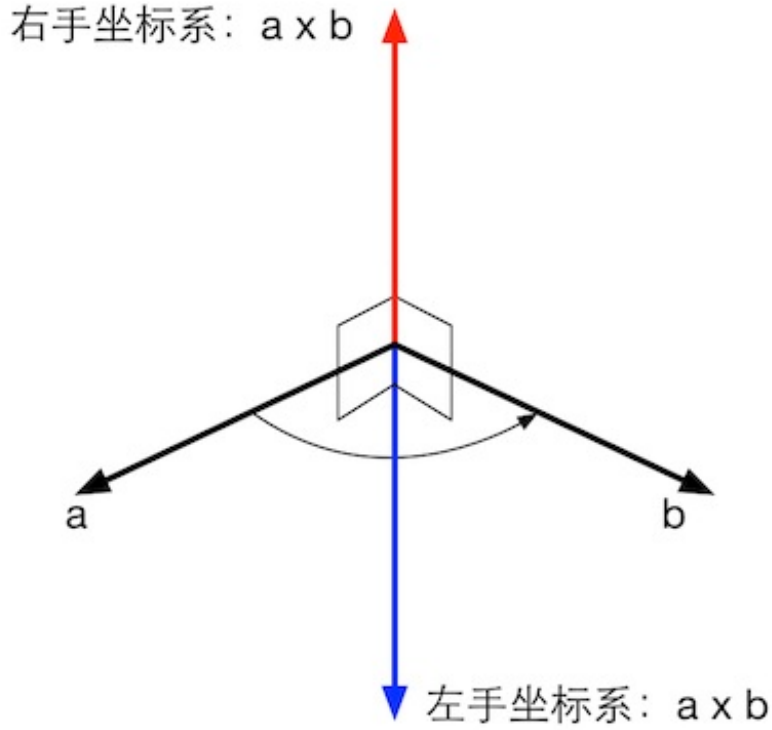


图4.27 分别使用左手坐标系和右手坐标系得到的叉积结果

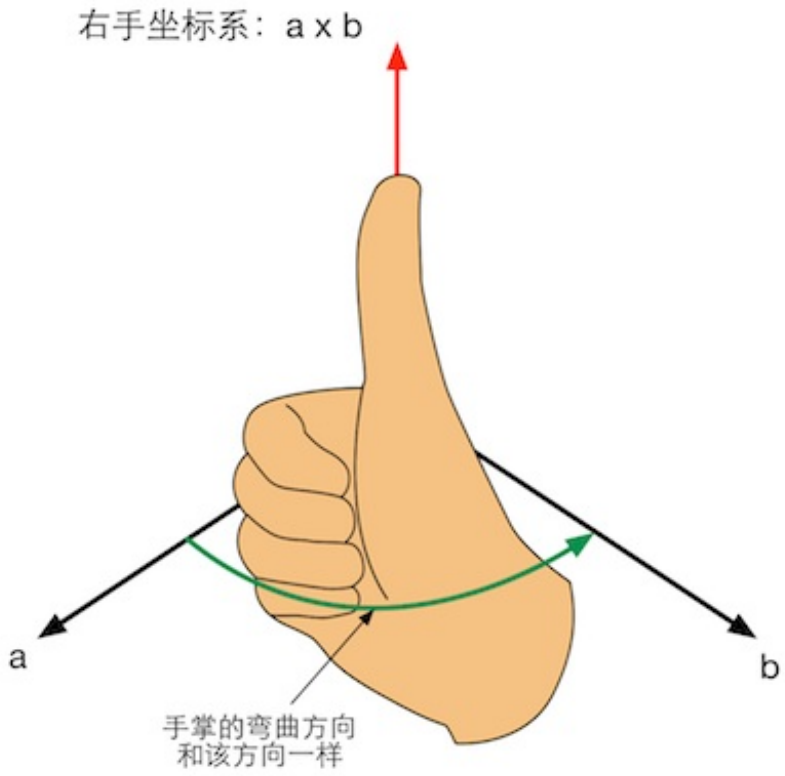


图4.28 使用右手法则判断右手坐标系中a×b的方向

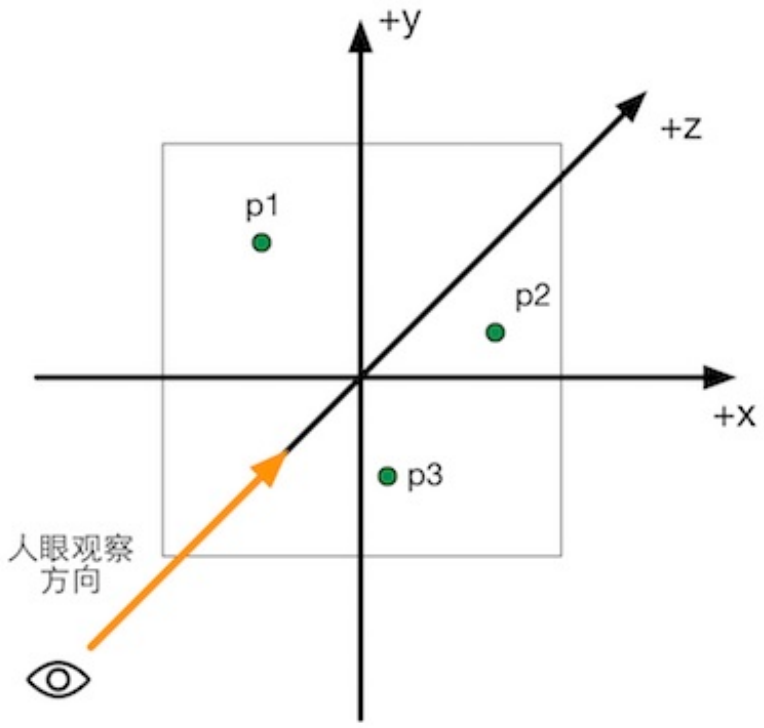


图4.29 三角形的三个顶点位于xy平面上，人眼位于z轴负方向，向z轴正方向观察

$$\begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{41} & a_{42} \end{bmatrix} \quad \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

图4.30 计算c23的过程

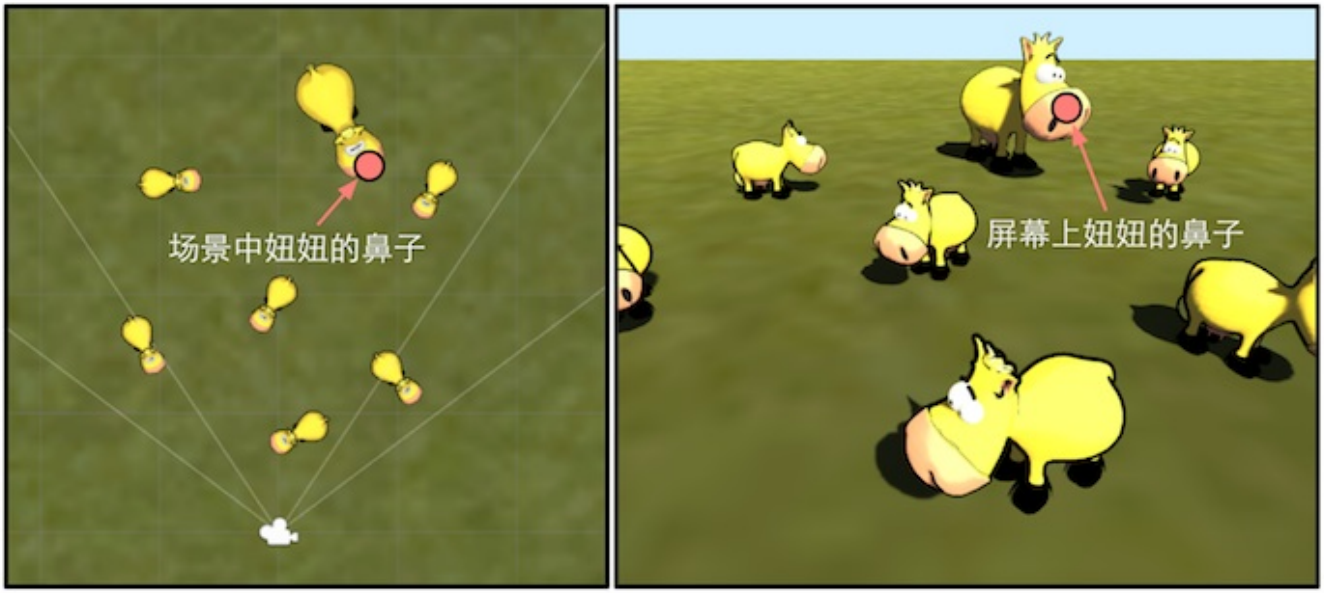


图4.31 场景中的妞妞（左图）和屏幕上的妞妞（右图）。妞妞想知道，自己的鼻子是如何被画到屏幕上的

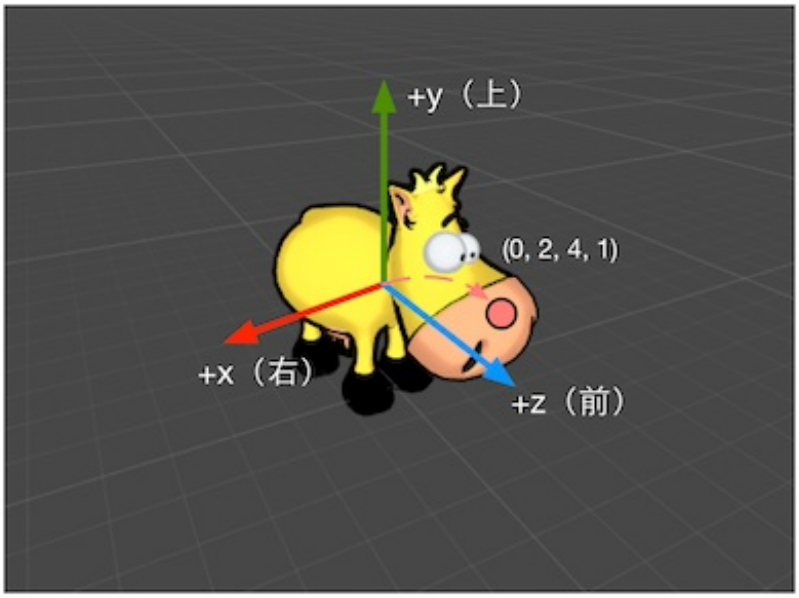


图4.32 在我们的农场游戏中，每个奶牛都有自己的模型坐标系。在模型坐标系中妞妞鼻子的位置是(0, 2, 4, 1)

图4.33 Unity的Transform组件可以调节模型的位置.如果Transform有父节点，如图中的“Mesh”，那么Position将是在其父节点（这里是“Cow”）的模型空间中的位置；如果没有父节点，Position就是在世界空间中的位置

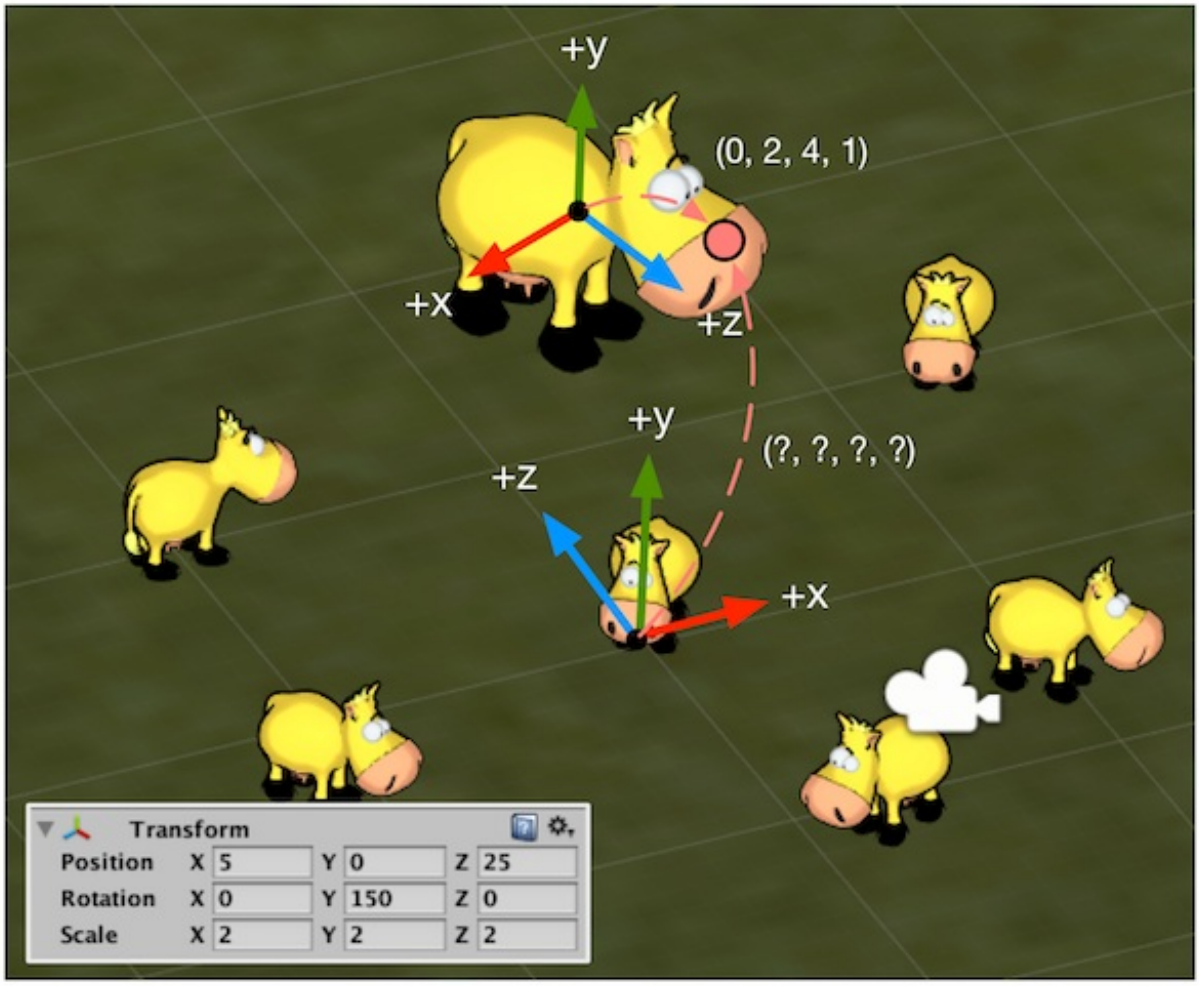


图4.34 农场游戏中的世界空间。世界空间的原点被放置在农场的中心。左下角显示了妞妞在世界空间中所做的变换。我们想要把妞妞的鼻子从模型空间变换到世界空间中

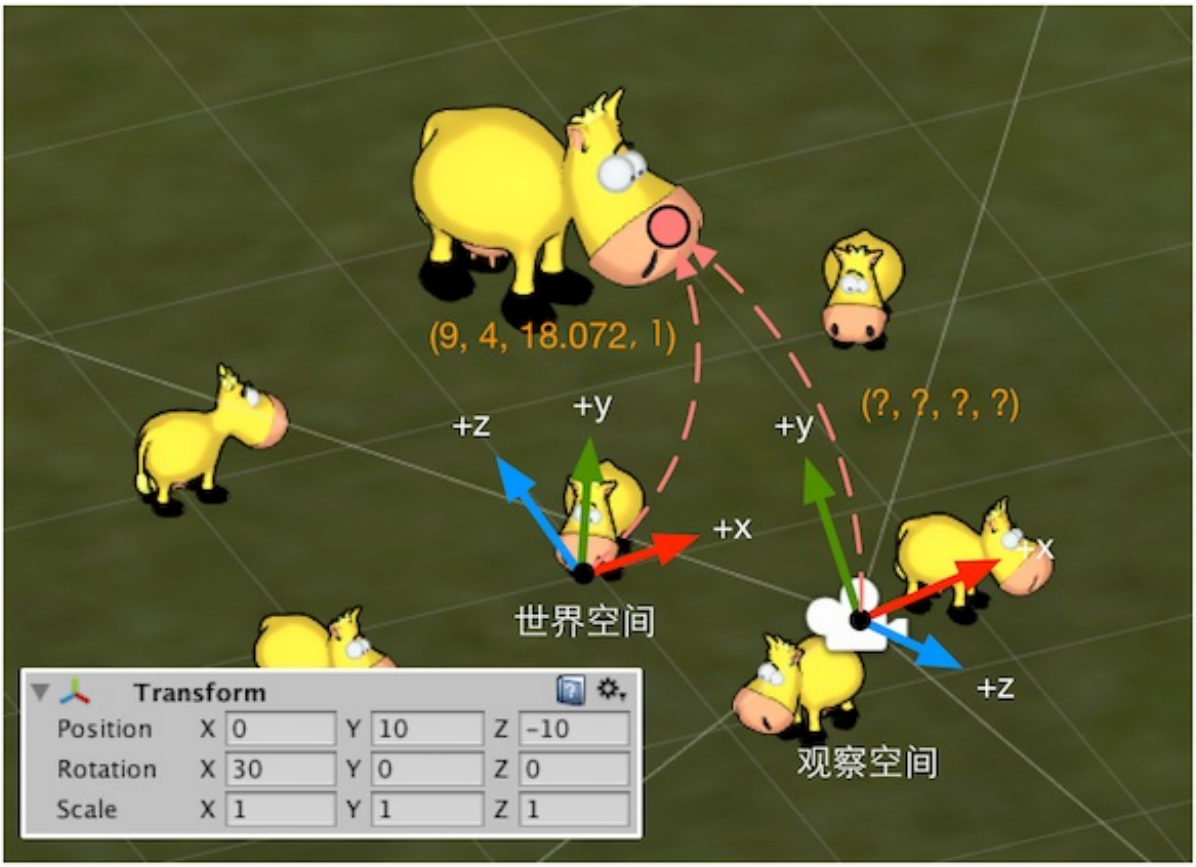


图4.35 农场游戏中摄像机的观察空间。观察空间的原点位于摄像机处。注意在观察空间中，摄像机的前向是z轴的负方向（图中只画出了z轴正方向），这是因为Unity在观察空间中使用了右手坐标系。左下角显示了摄像机在世界空间中所做的变换。我们想要把妞妞的鼻子从世界空间变换到观察空间中

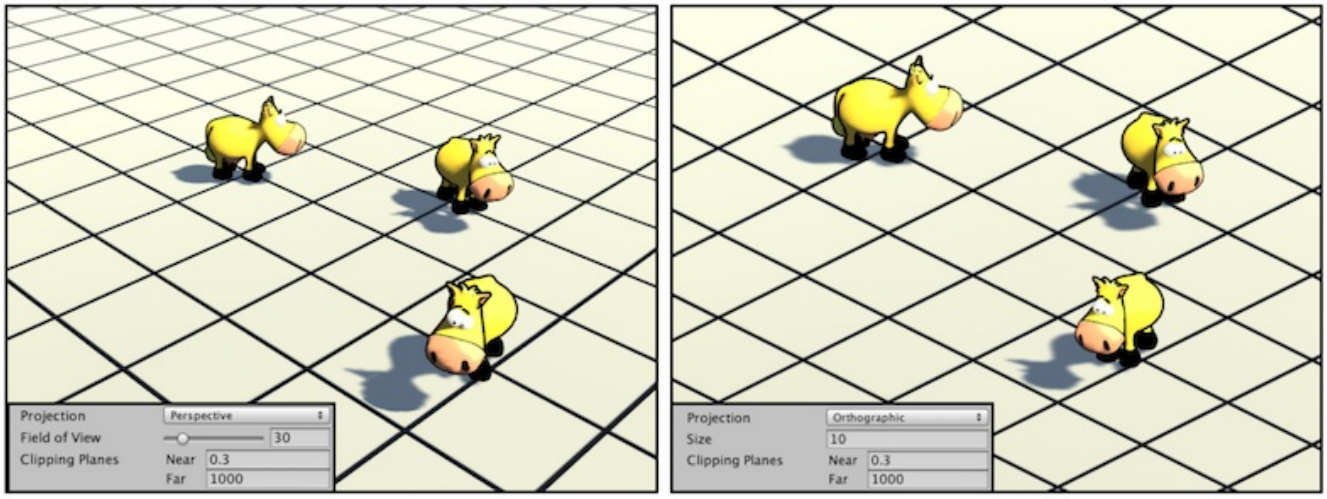


图4.36 透视投影（左图）和正交投影（右图）。左下角分别显示了当前摄像机的投影模式和相关属性

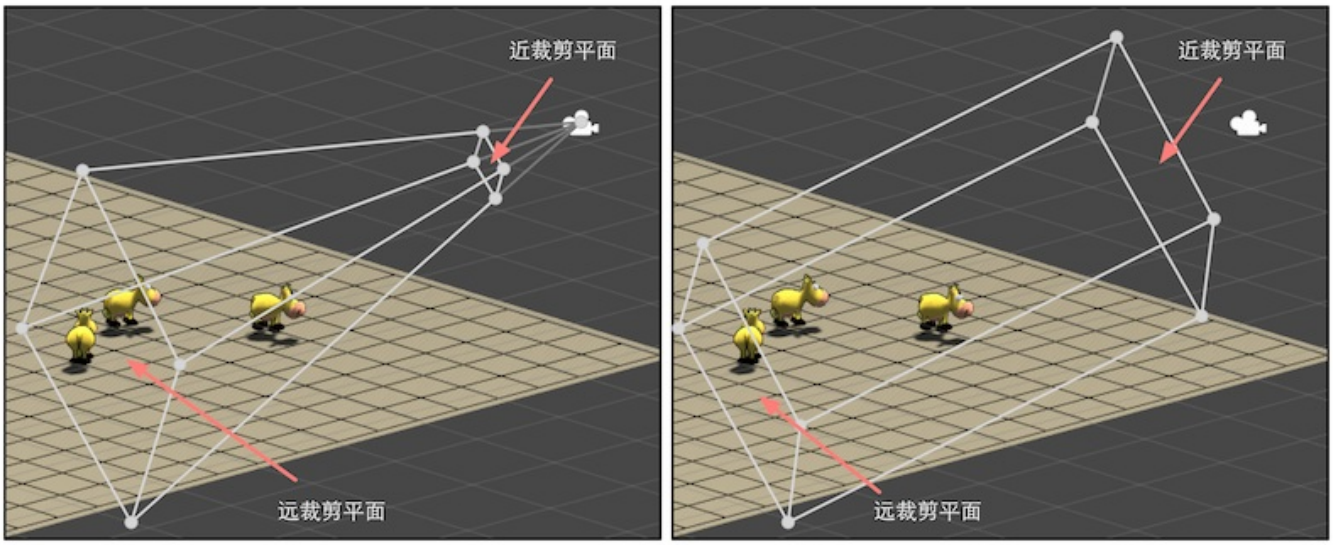


图4.37 视锥体和裁剪平面。左图显示了透视投影的视锥体，右图显示了正交投影的视锥体

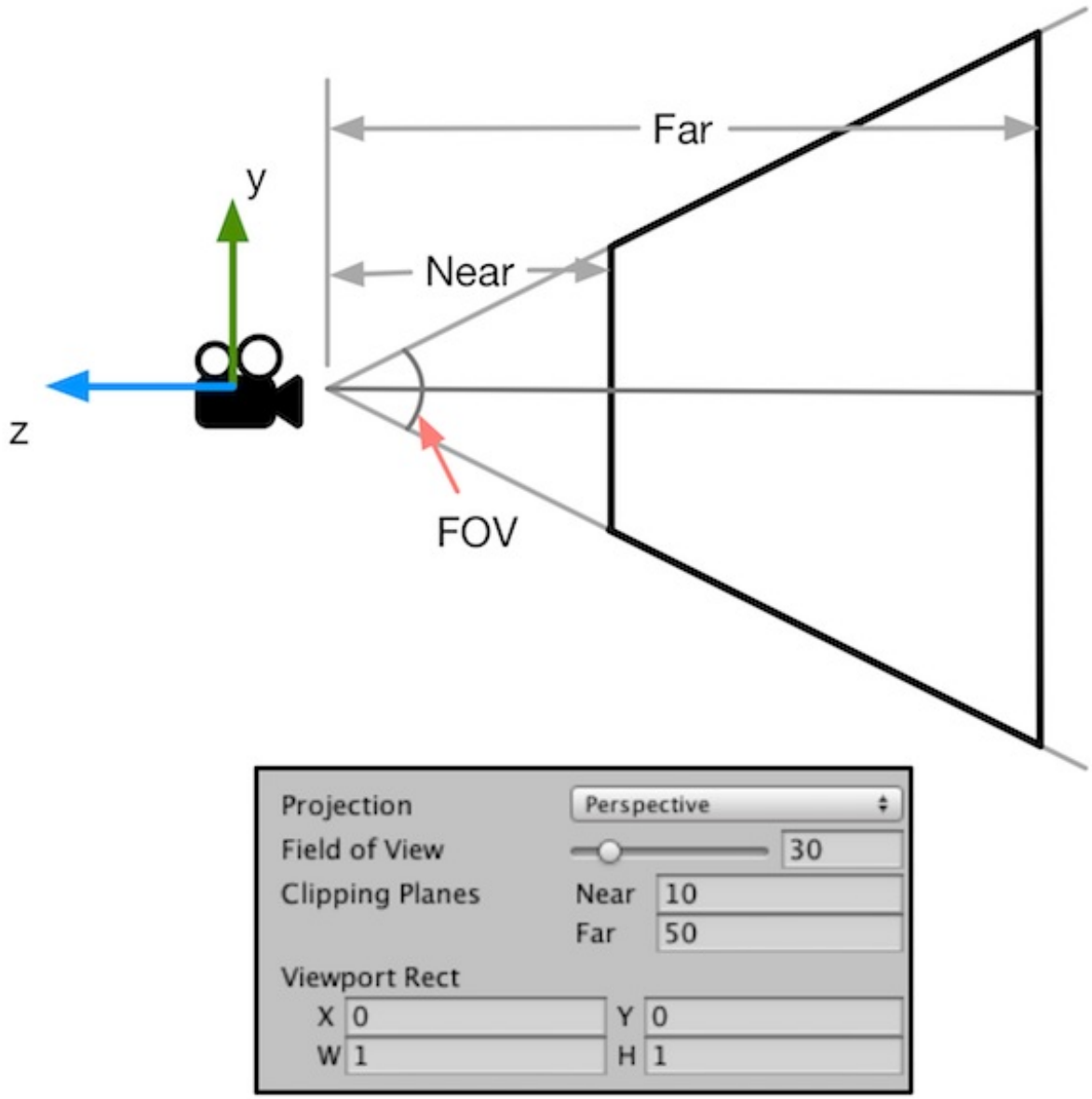


图4.38 透视摄像机的参数对透视投影视锥体的影响

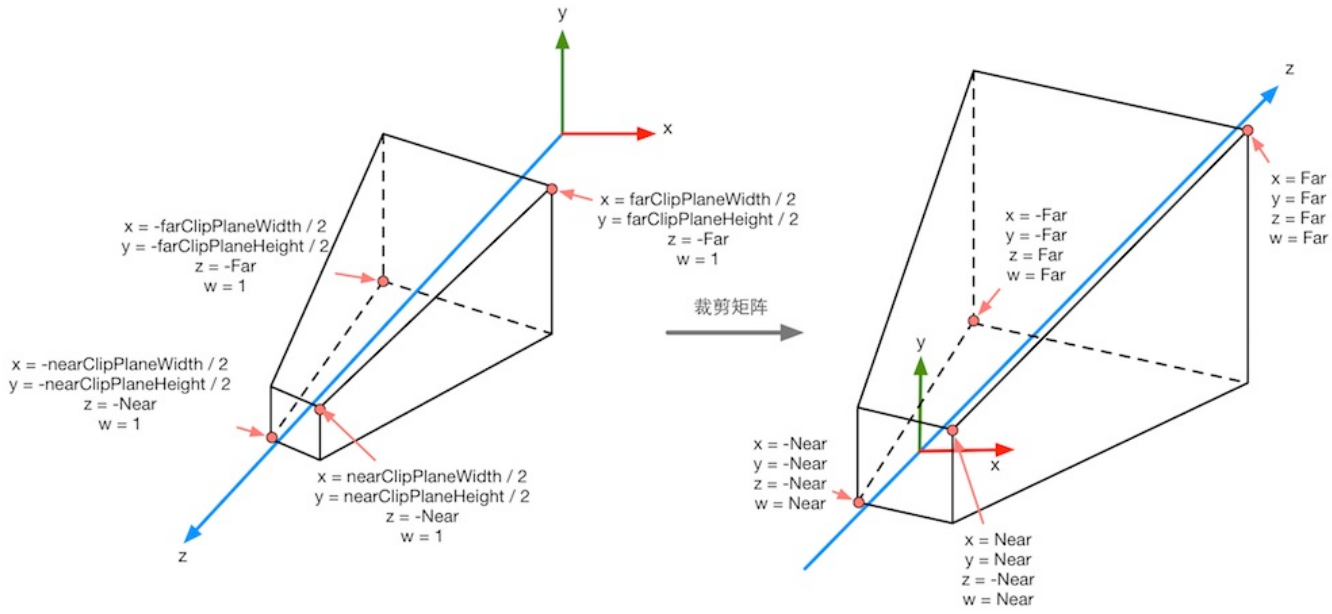


图4.39 在透视投影中，投影矩阵对顶点进行了缩放。图3.38中标注了4个关键点经过投影矩阵变换后的结果。从这些结果可以看出x、y、z和w分量的范围发生的变化

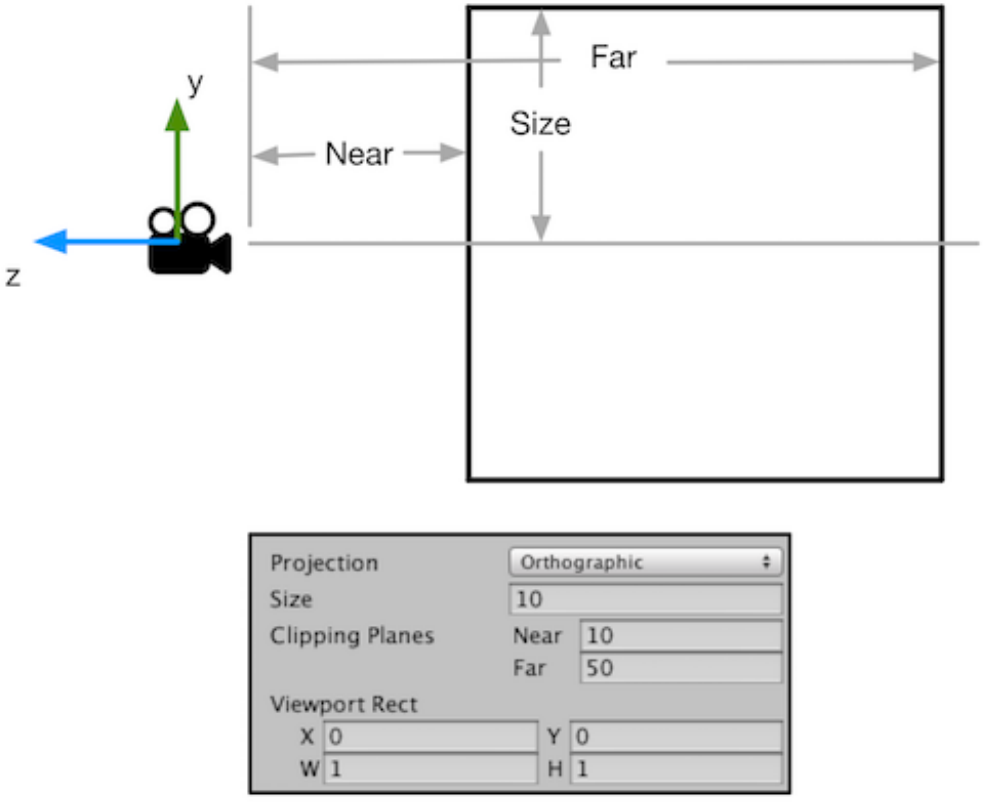


图4.40 正交摄像机的参数对正交投影视锥体的影响

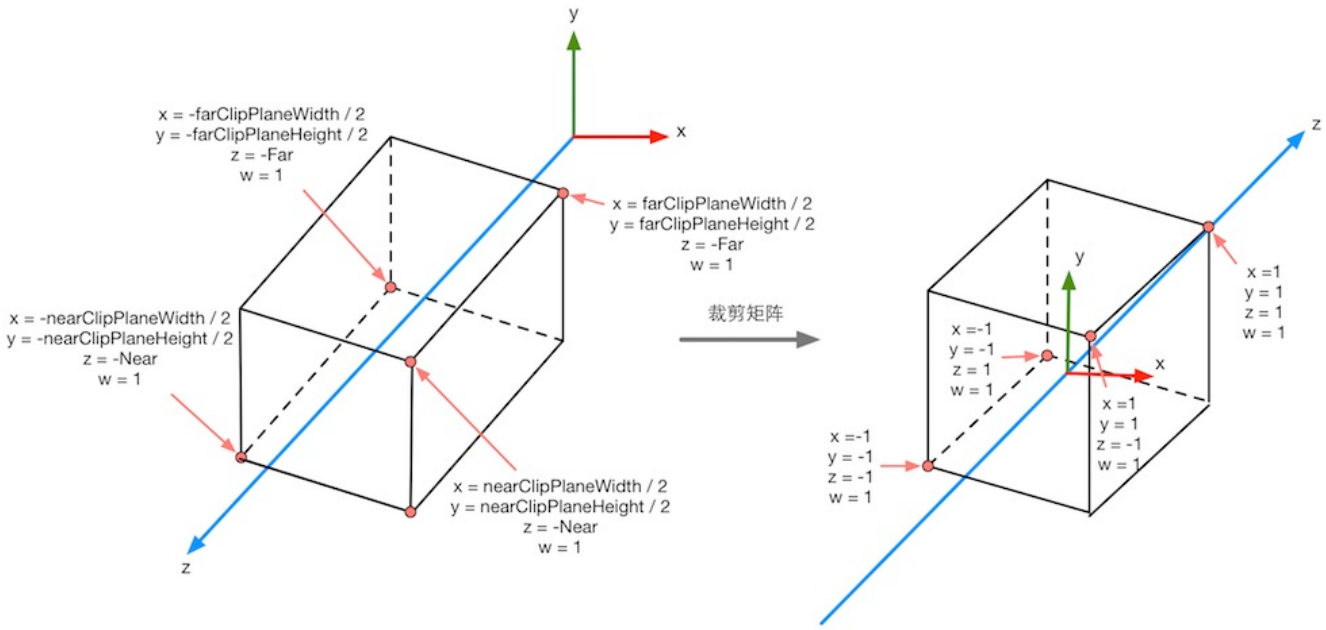


图4.41 在正交投影中，投影矩阵对顶点进行了缩放。图中标注了4个关键点经过投影矩阵变换后的结果。从这些结果可以看出x、y、z和w分量范围发生的变化

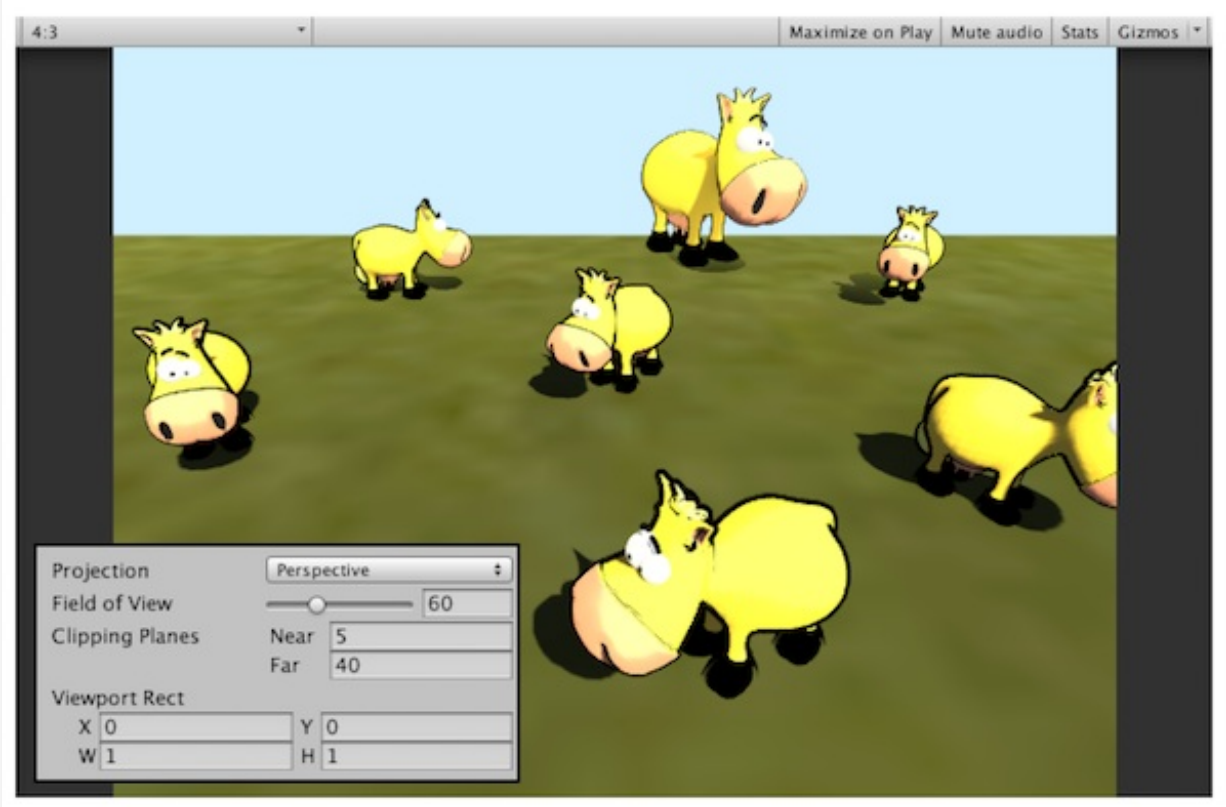


图4.42 农场游戏使用的摄像机参数和游戏画面的横纵比

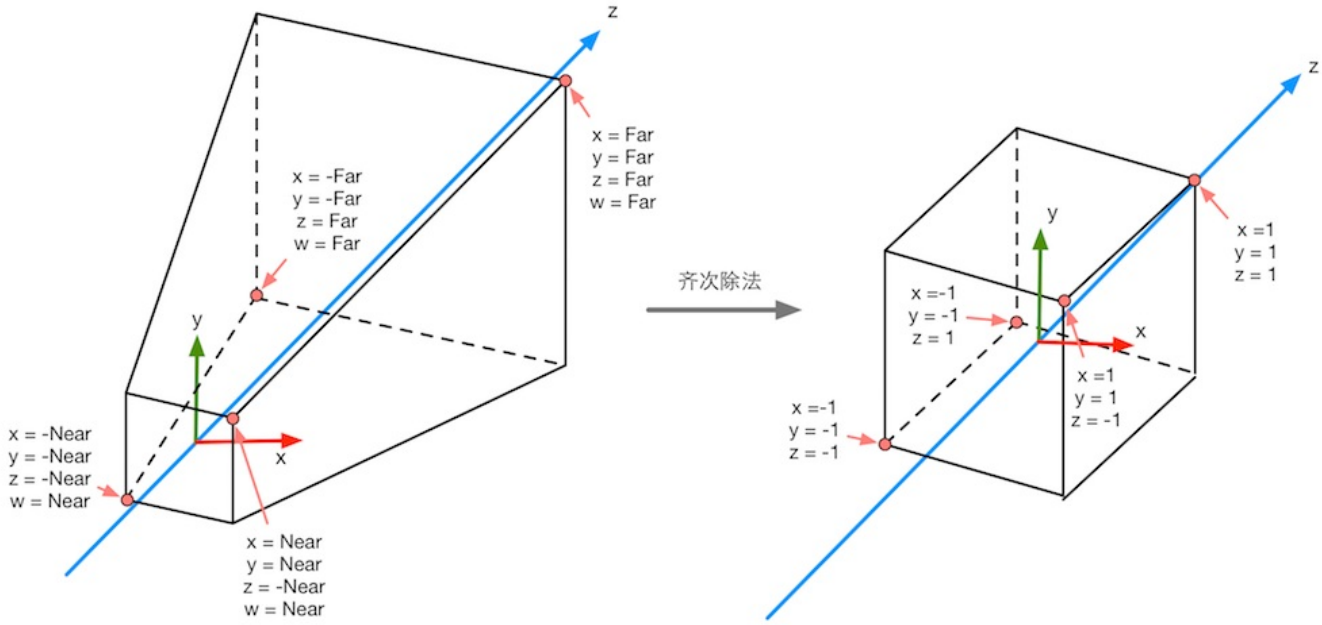


图4.43 经过齐次除法后，透视投影的裁剪空间会变换到一个立方体

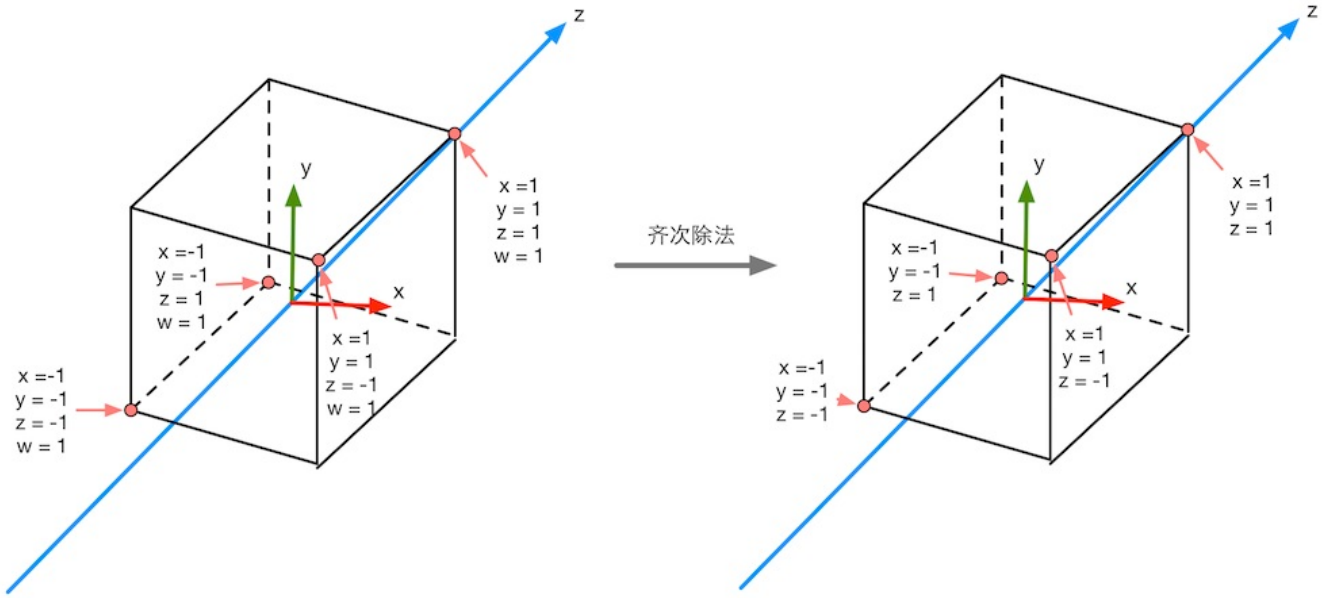


图4.44 经过齐次除法后，正交投影的裁剪空间会变换到一个立方体

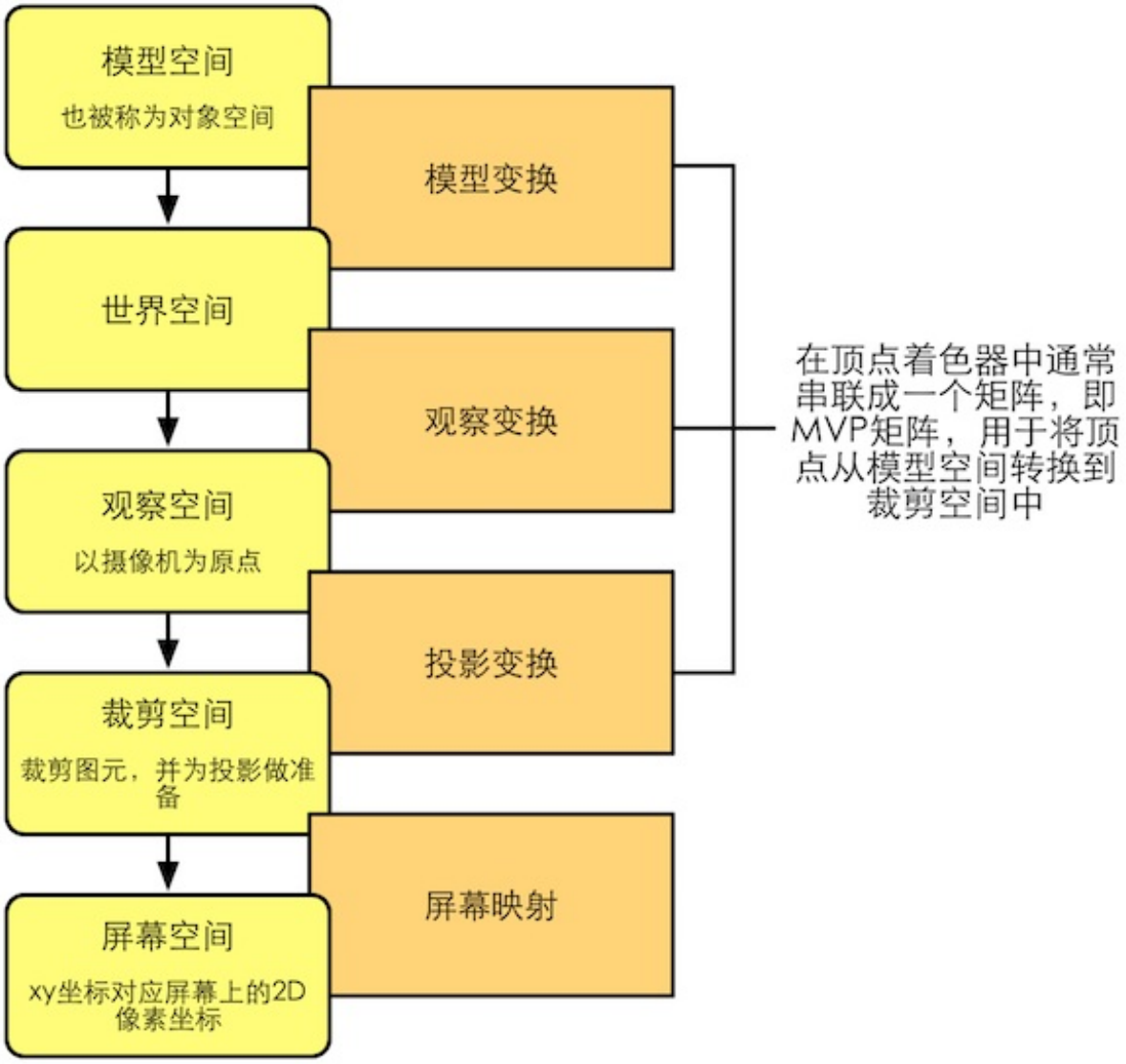


图4.45 渲染流水线中顶点的空间变换过程

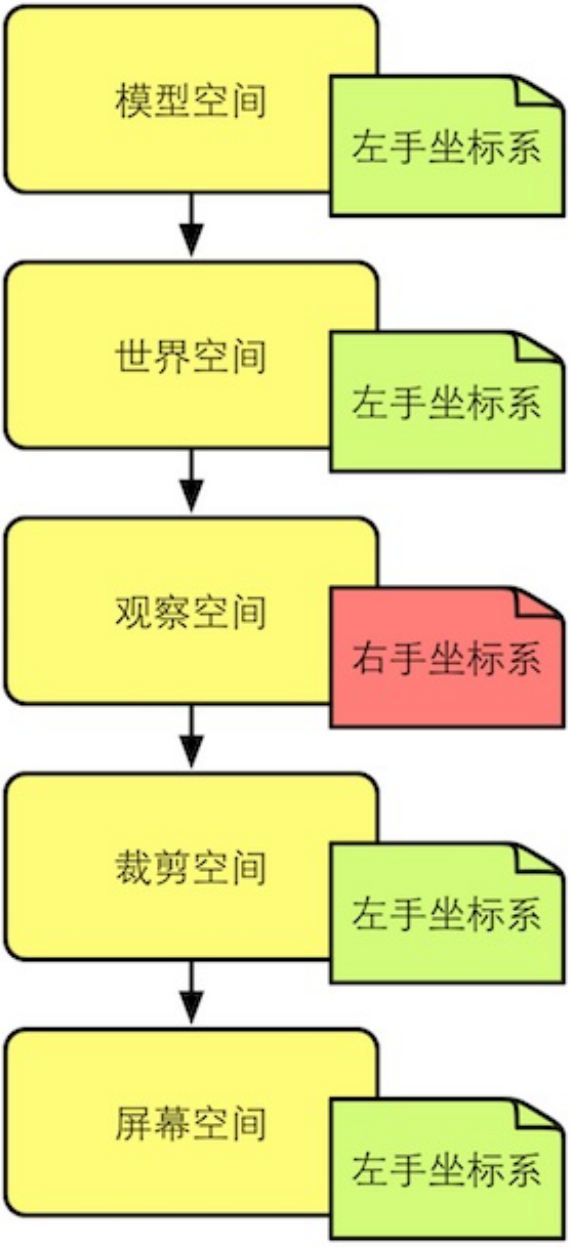


图4.46 Unity中各个坐标空间的旋向性

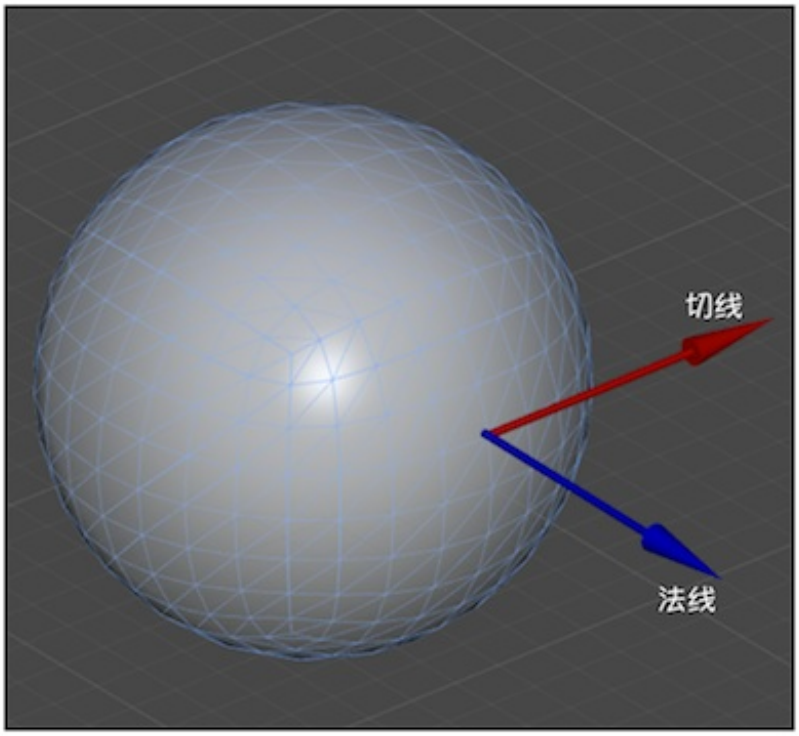


图4.47 顶点的切线和法线。切线和法线互相垂直

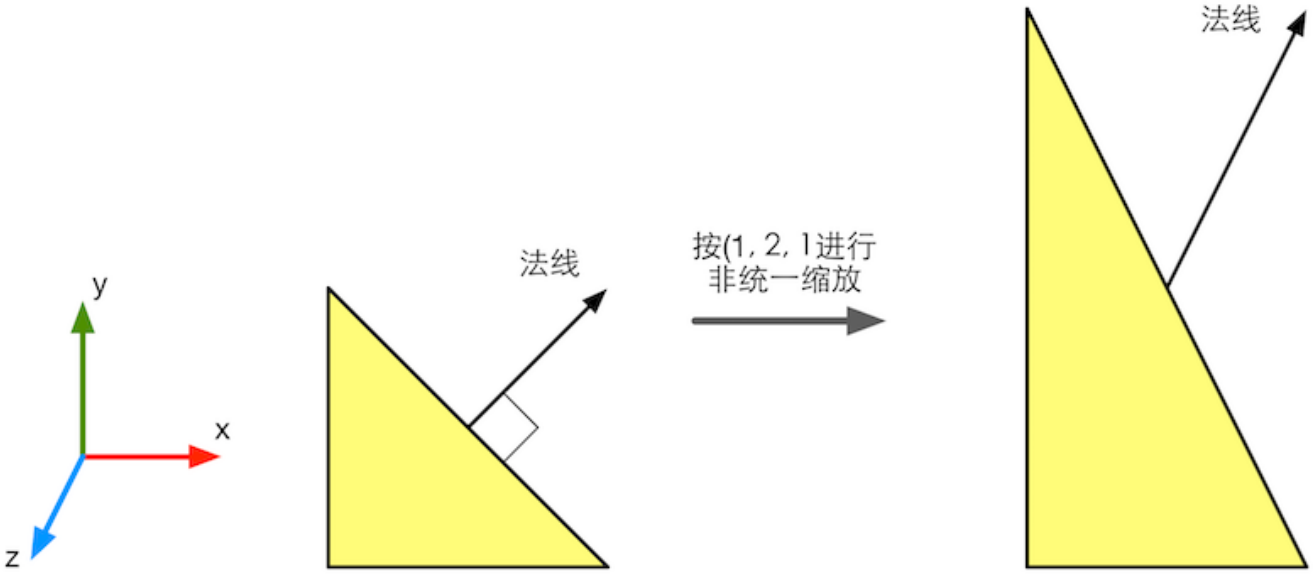


图4.48 进行非统一缩放时，如果使用和变换顶点相同的变换矩阵来变换法线，就会得到错误的结果，即变换后的法线方向与平面不再垂直

图4.49 由片元的像素位置得到的图像

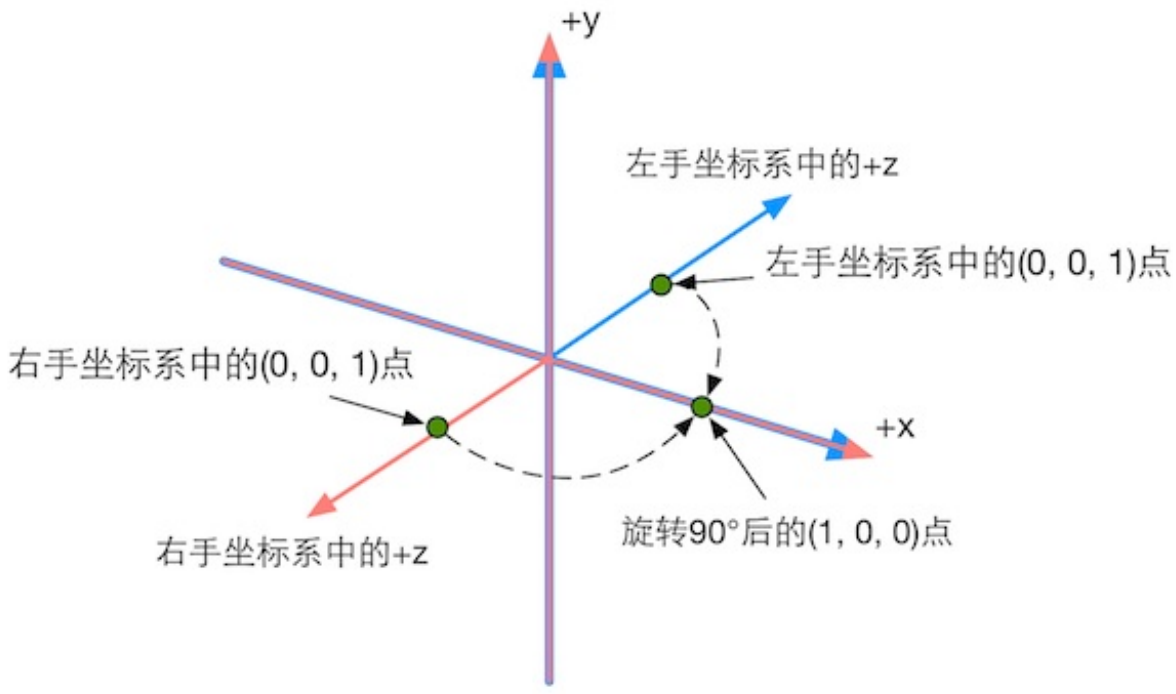


图4.50 图中两个坐标系的x轴和y轴是重合的，区别仅在于z轴的方向。左手坐标系的（0, 0, 1）点和右手坐标系中的（0, 0, 1）点是不同的，但它们旋转后的点却对应到了同一点

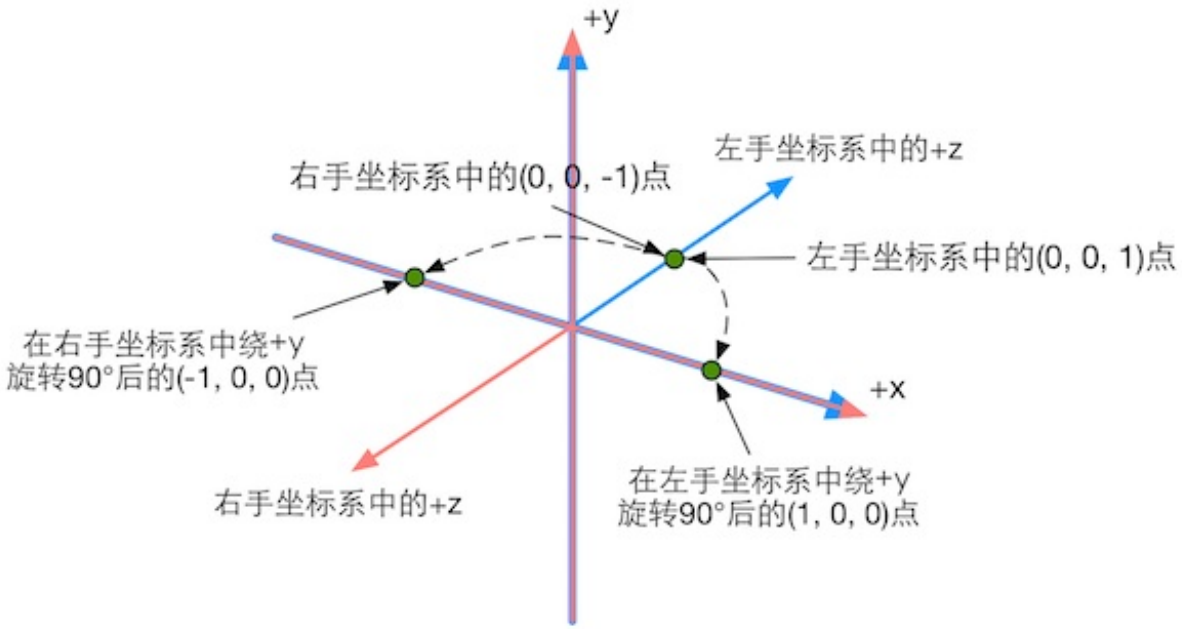


图4.51 绝对空间中的同一点，在左手和右手坐标系中进行同样角度的旋转，其旋转方向是不一样的。在左手坐标系中将按顺时针方向旋转，在右手坐标系中将按逆时针方向旋转

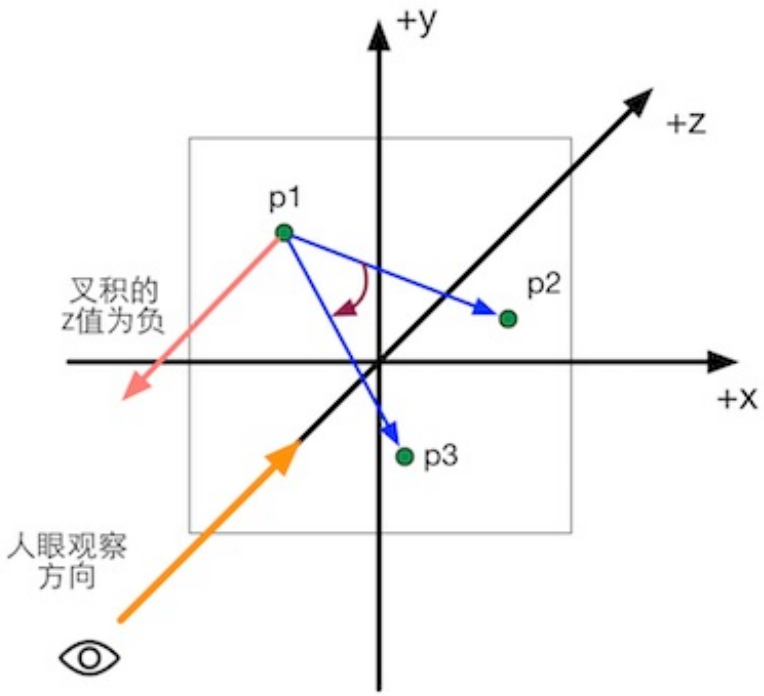


图4.52 在左手坐标系中，如果叉积结果为负，那么3点的顺序是顺时针方向

第5章 开始Unity Shader学习之旅

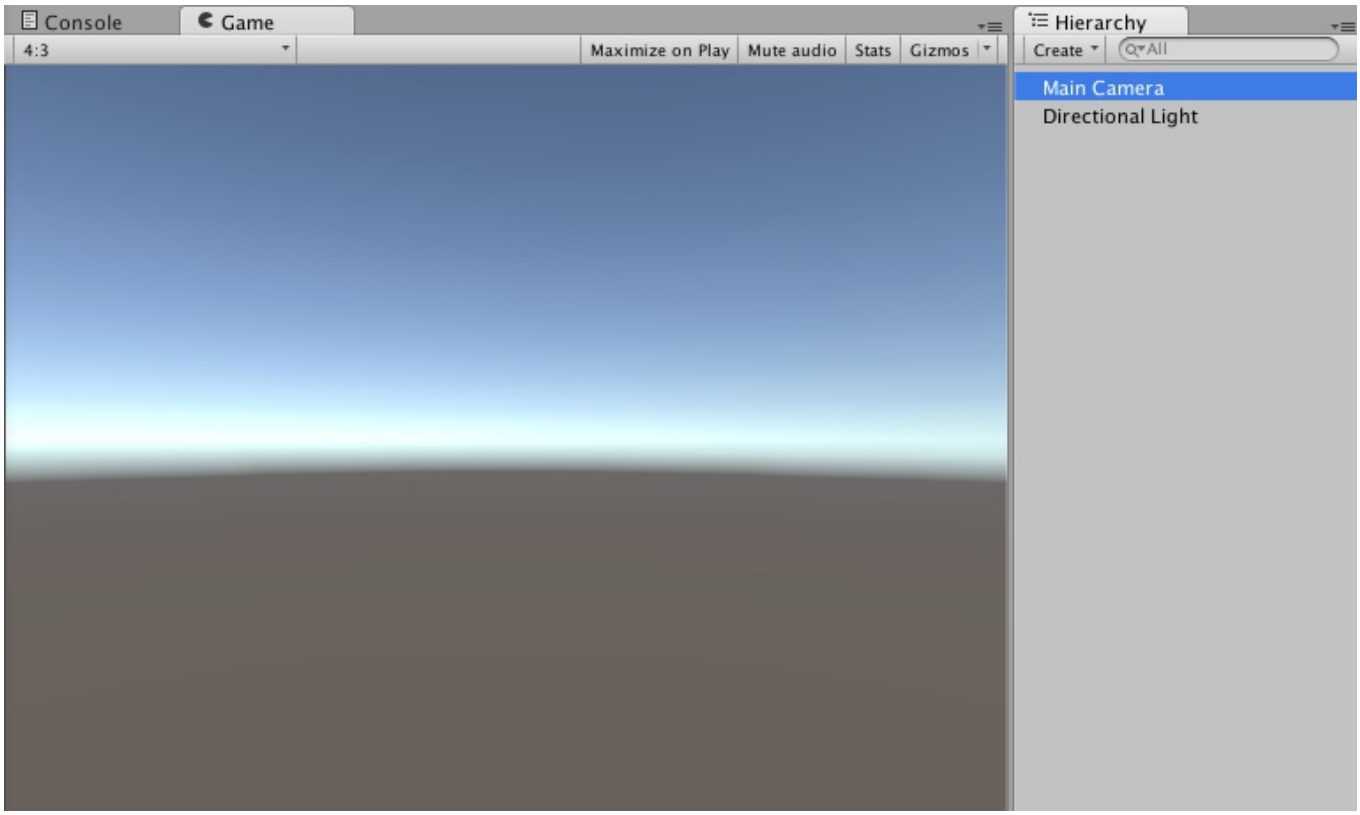


图5.1 在Unity 5中新建一个场景得到的效果

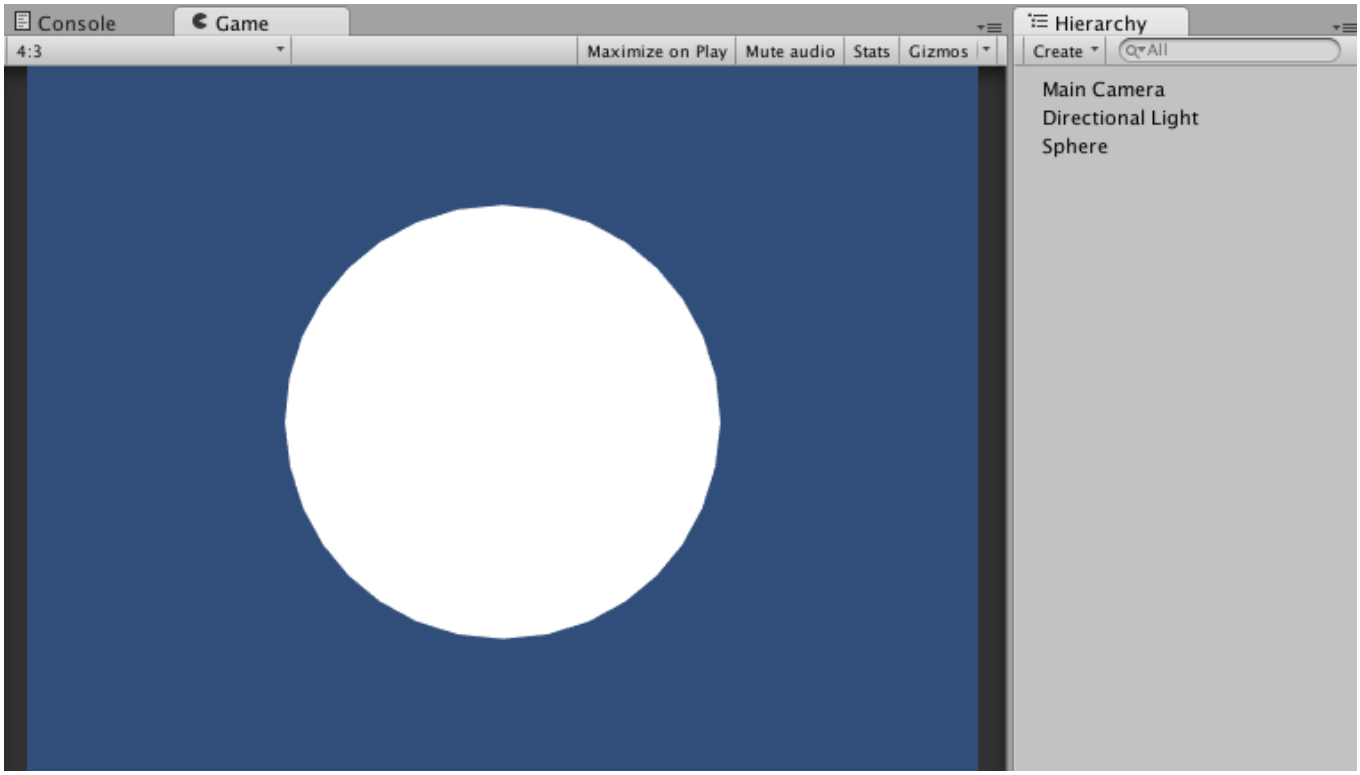


图5.2 用一个最简单的顶点/片元着色器得到一个白色的球

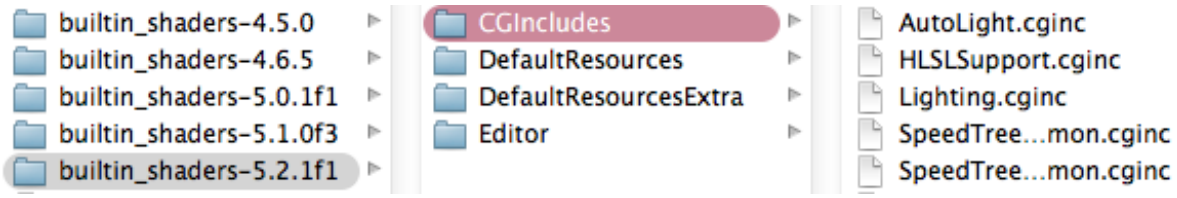


图5.3 Unity的内置着色器

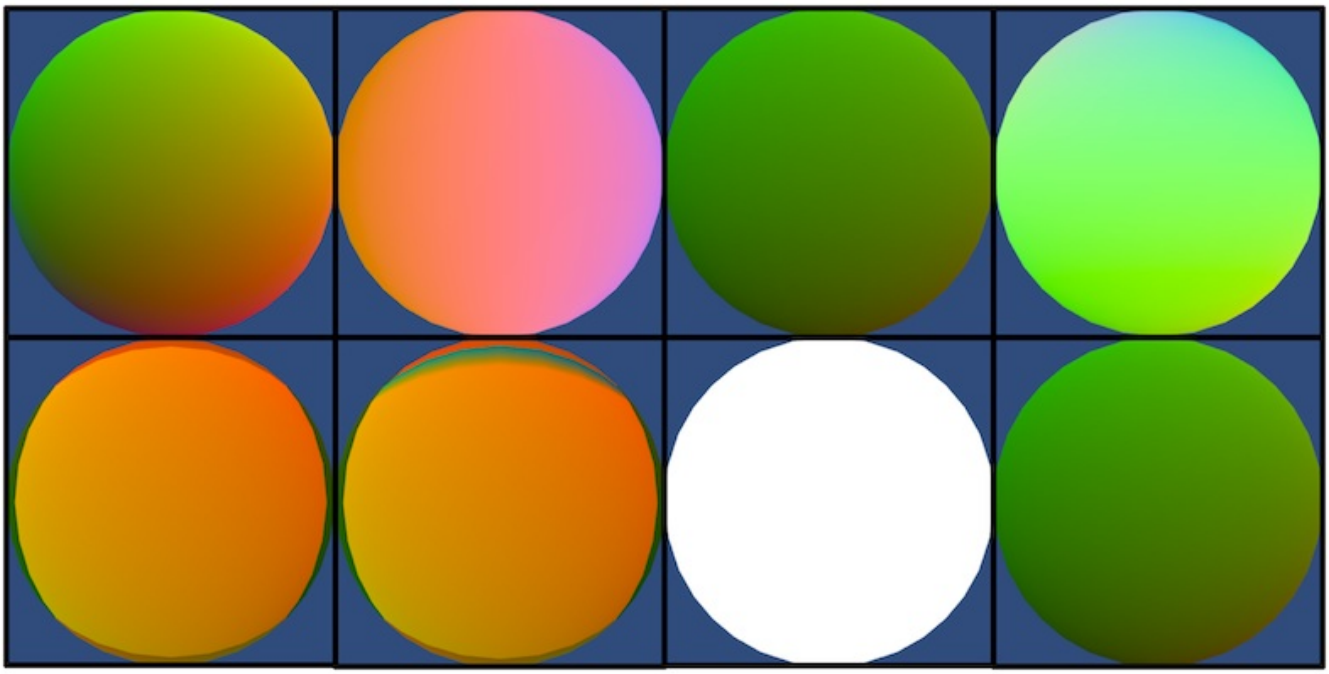


图5.4 用假彩色对Unity Shader进行调试

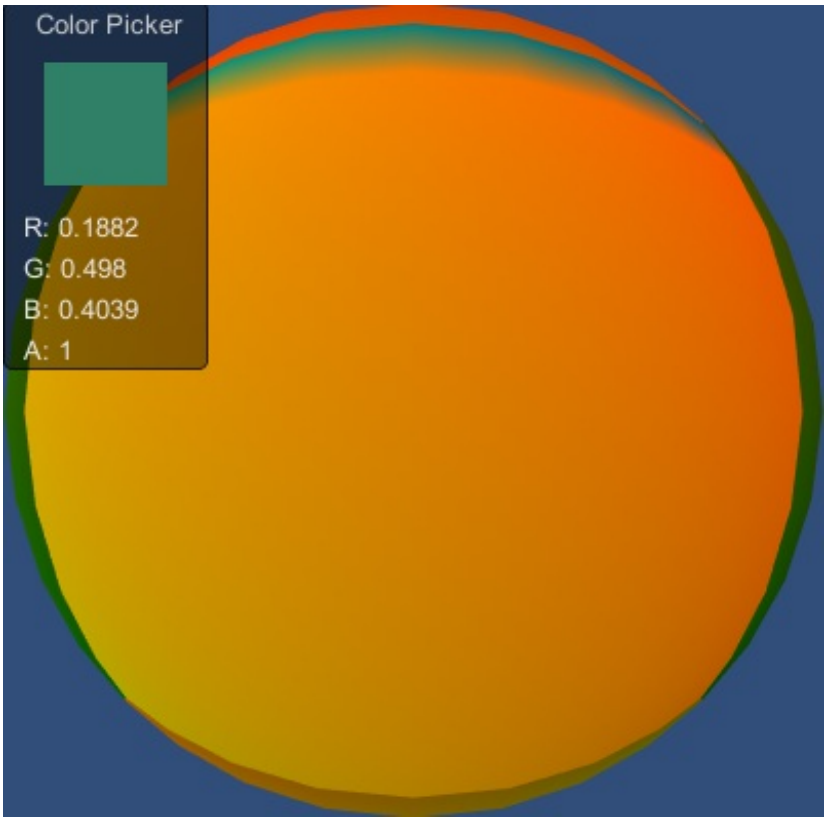


图5.5 使用颜色拾取器来查看调试信息

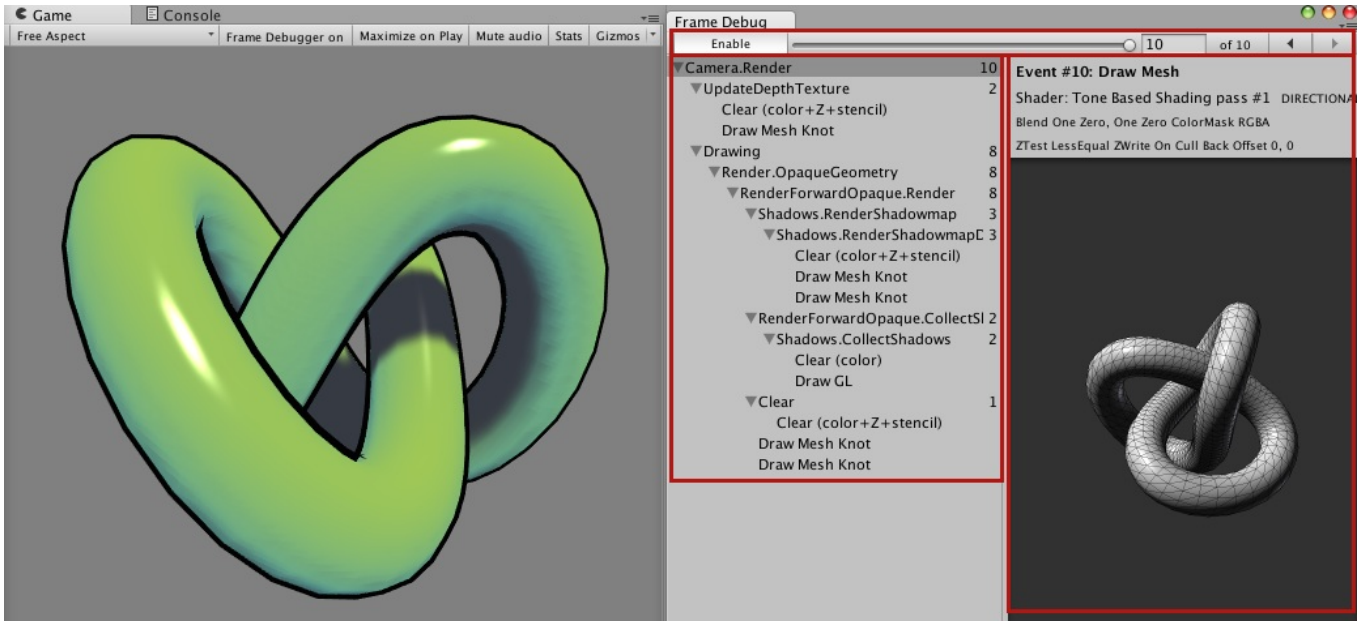


图5.6 帧调试器

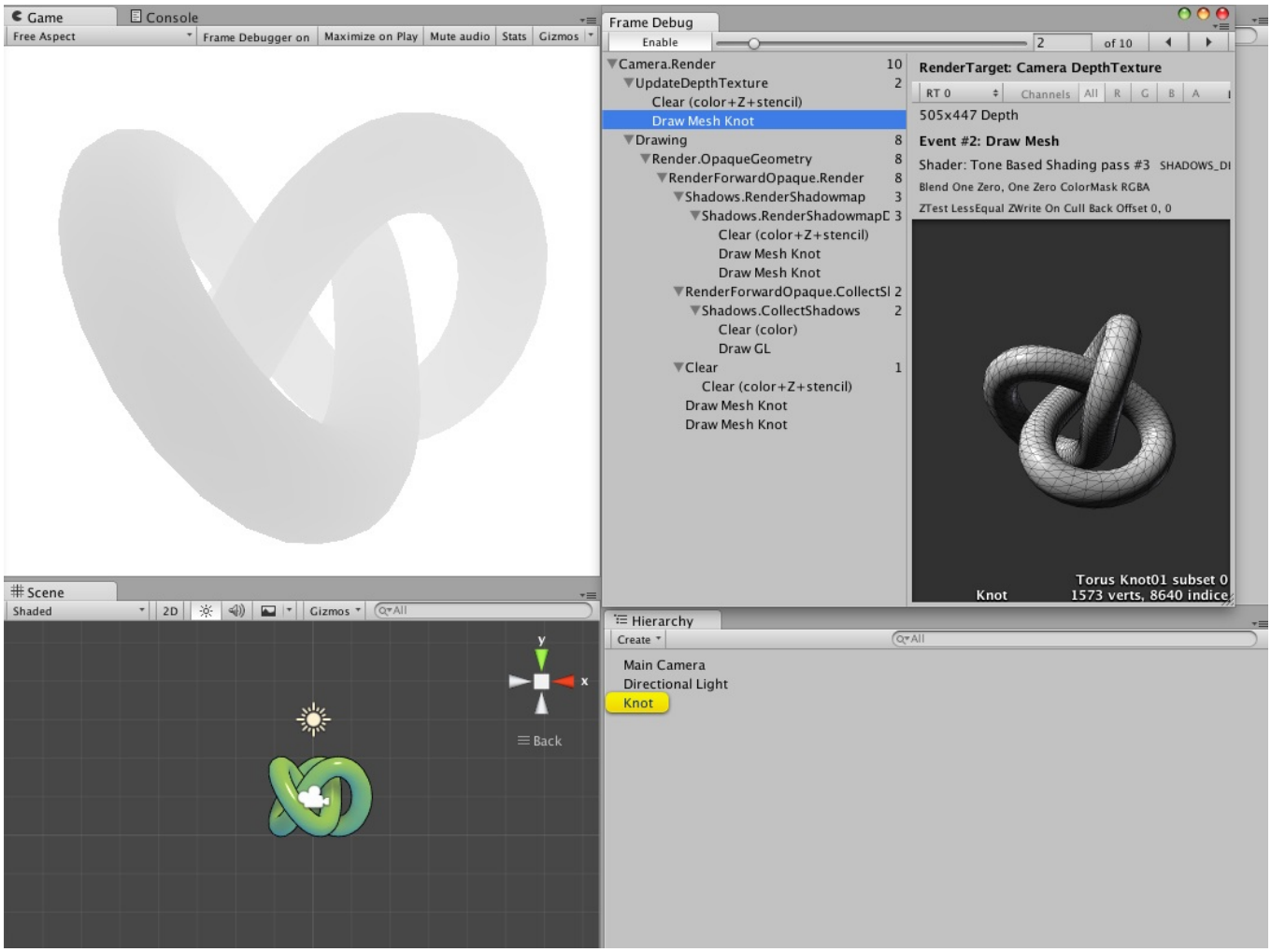


图5.7 单击Knot的深度图渲染事件，在Game视图会显示该事件的效果，在Hierarchy视图中会高亮显示Knot对象，在帧调试器的右侧窗口会显示出该事件的细节

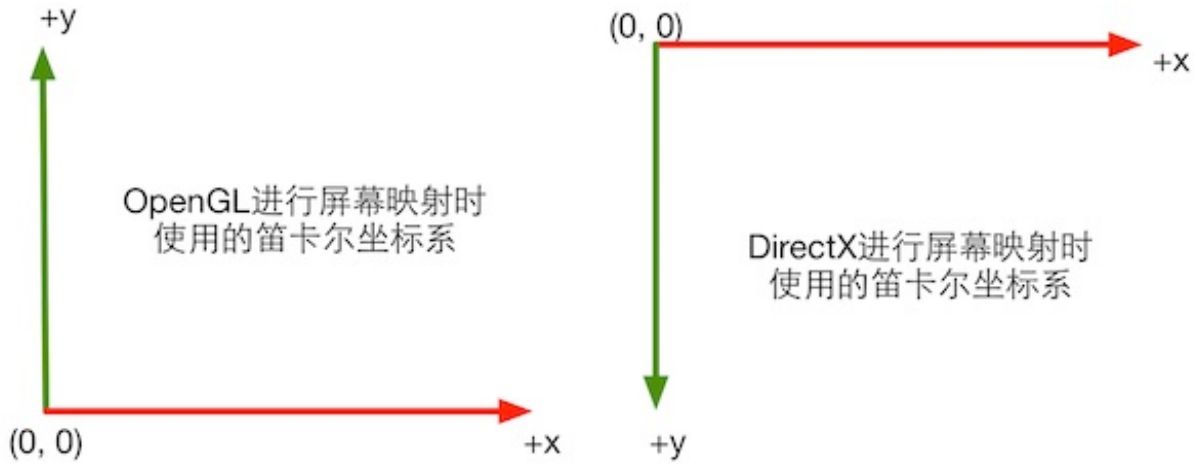


图5.8 OpenGL和DirectX使用了不同的屏幕空间坐标

第6章 Unity中的基础光照

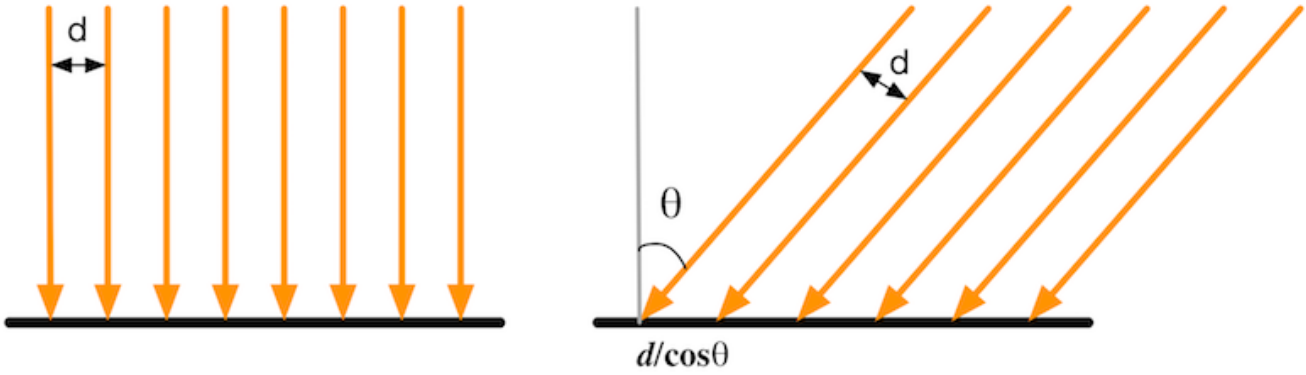


图6.1 在左图中，光是垂直照射到物体表面，因此光线之间的垂直距离保持不变；而在右图中，光是斜着照射到物体表面，在物体表面光线之间的距离是d/cosθ，因此单位面积上接收到的光线数目要少于左图

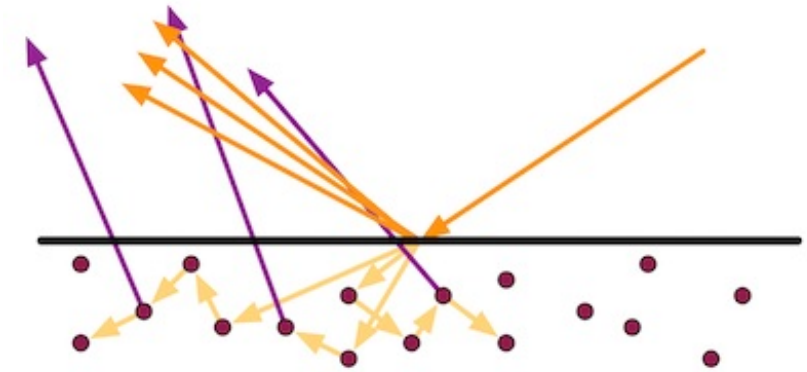


图6.2 散射时，光线会发生折射和反射现象。对于不透明物体，折射的光线会在物体内部继续传播，最终有一部分光线会重新从物体表面被发射出去

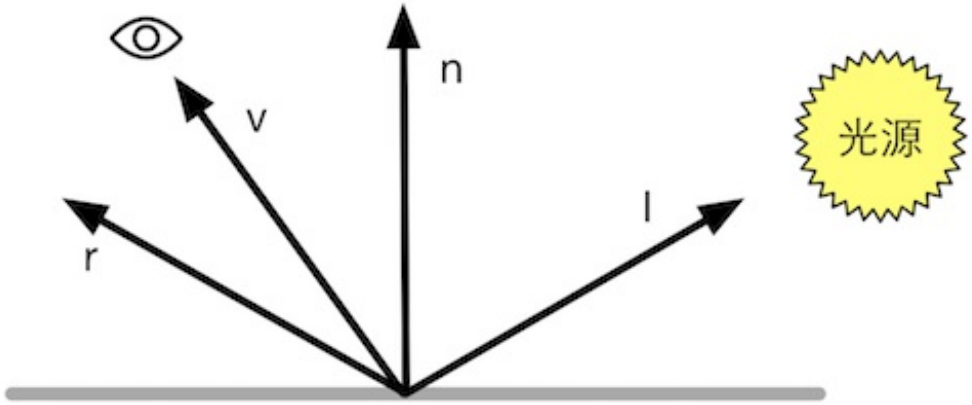


图6.3 使用Phong模型计算高光反射

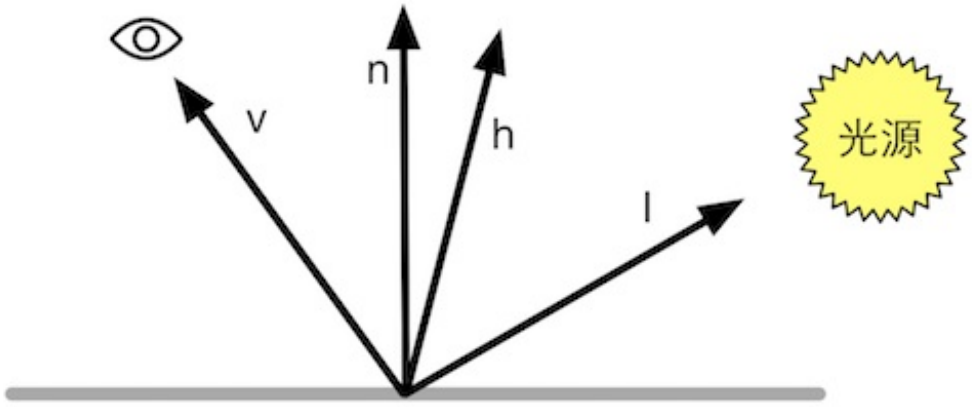


图6.4 Blinn模型

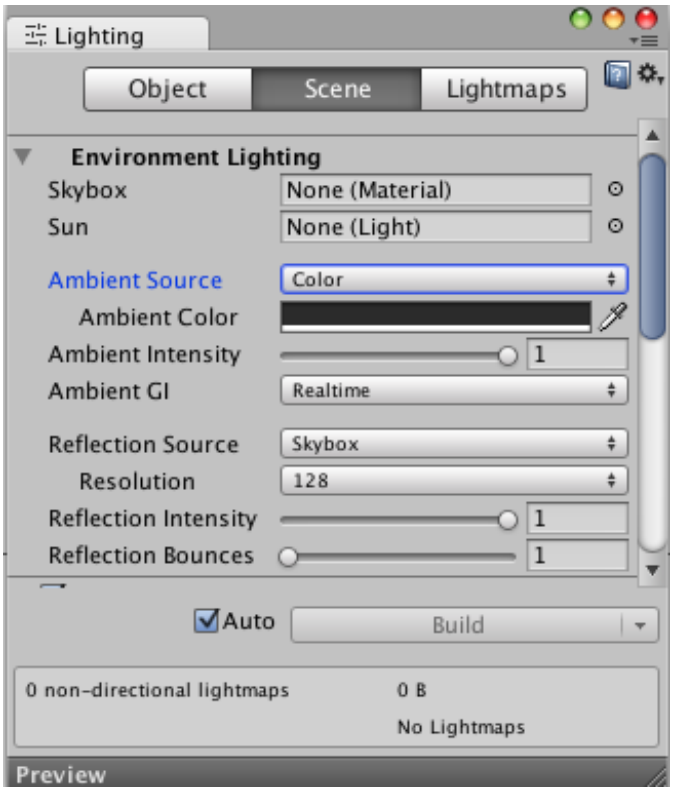


图6.5 在Unity的Window -> Lighting面板中，我们可以通过Ambient Source/Ambient Color/Ambient Intensity来控制场景中的环境光的颜色和强度

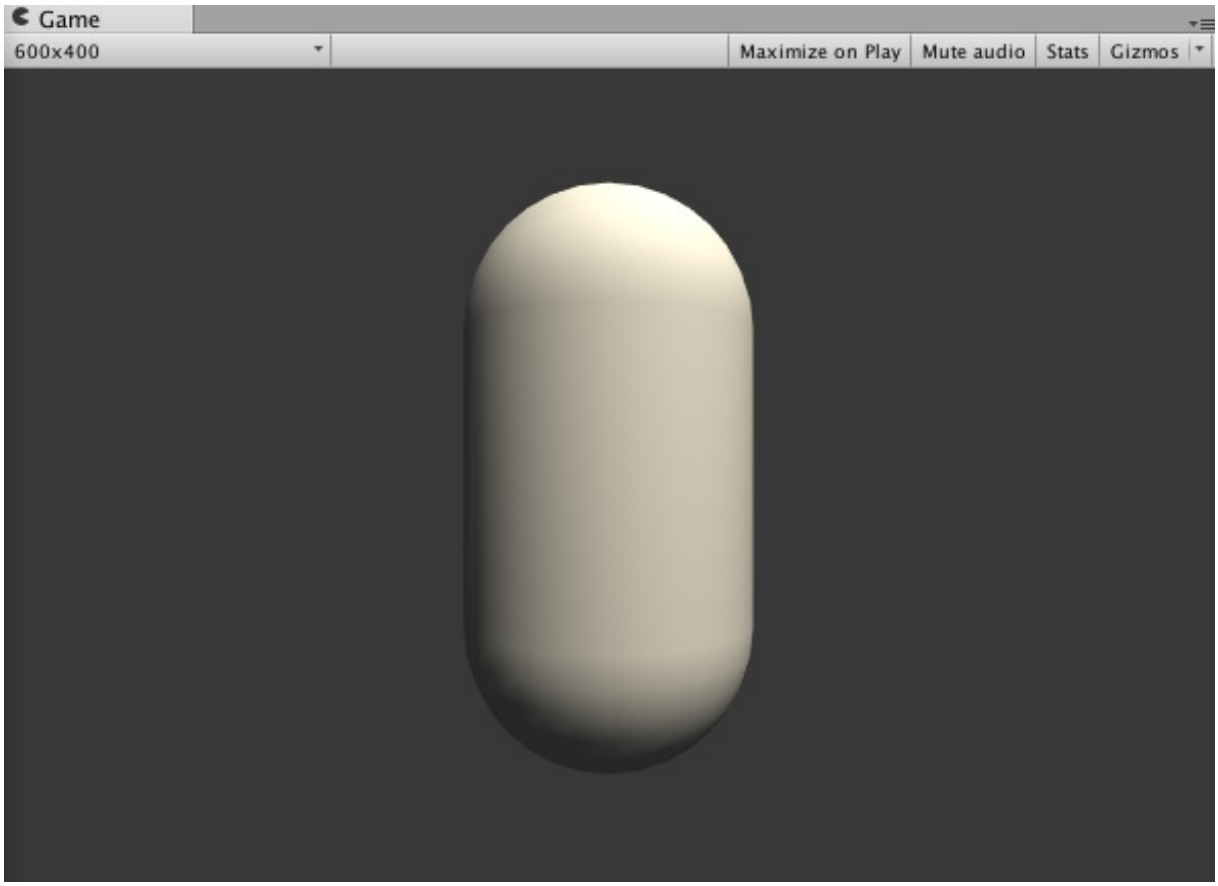


图6.6 逐顶点的漫反射光照效果

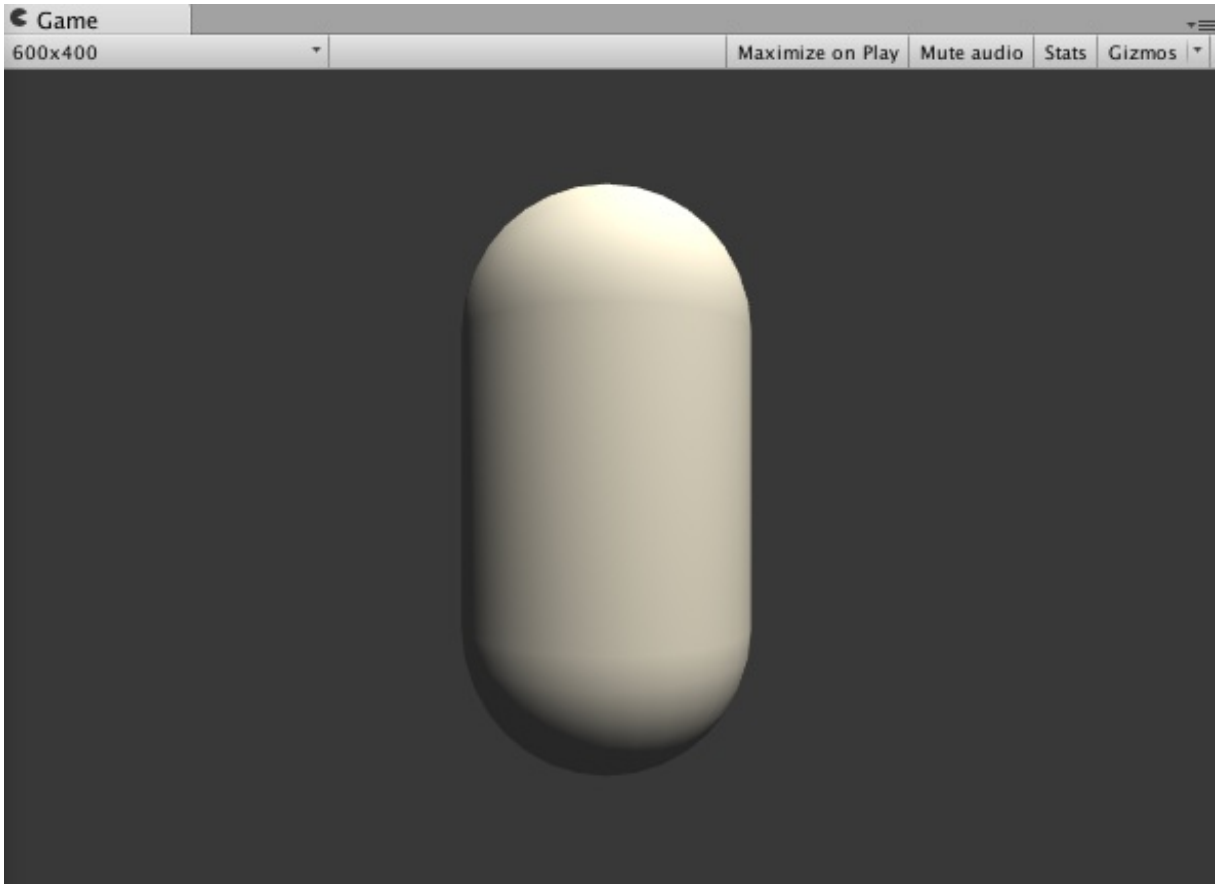


图6.7 逐像素的漫反射光照效果

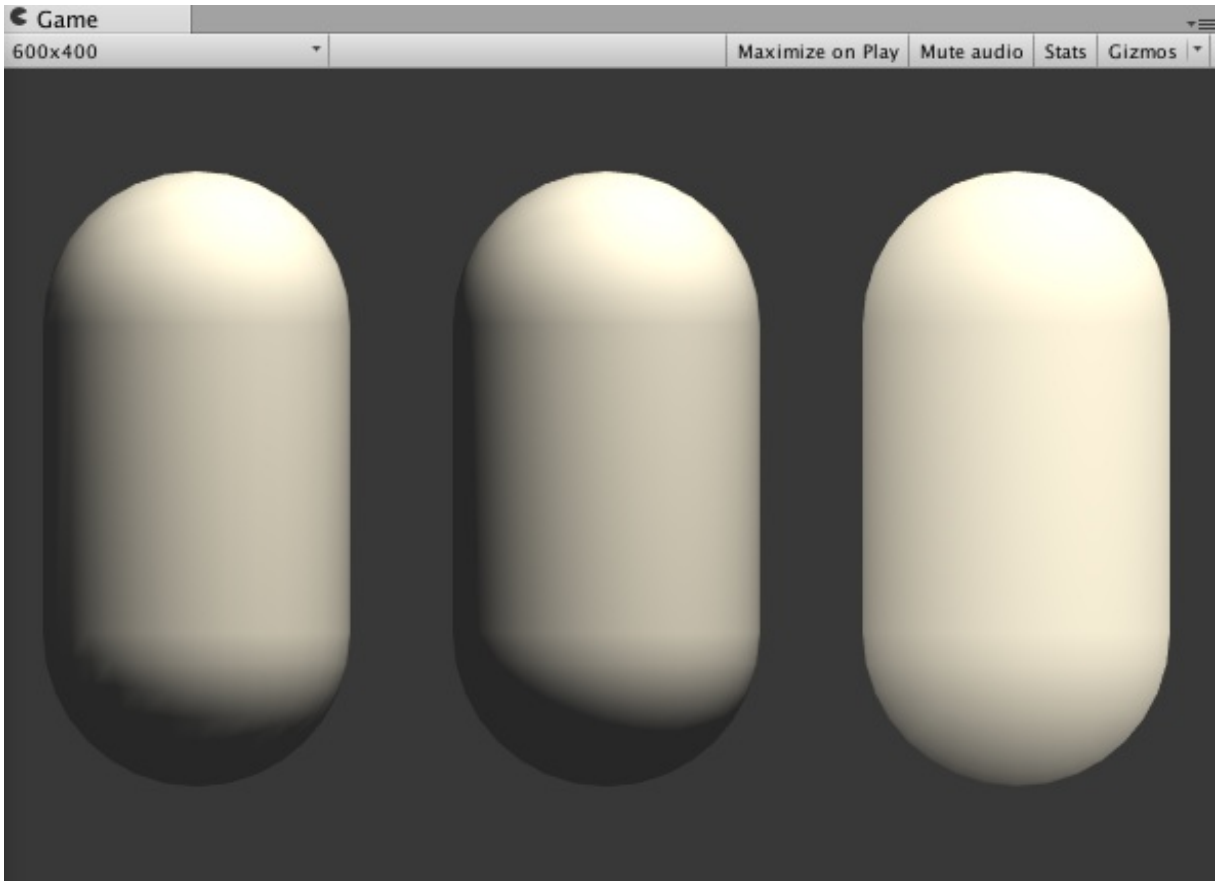


图6.8 逐顶点漫反射光照、逐像素漫反射光照、半兰伯特光照的对比效果

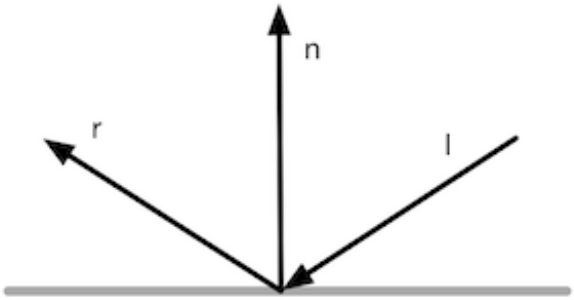


图6.9 CG的reflect函数

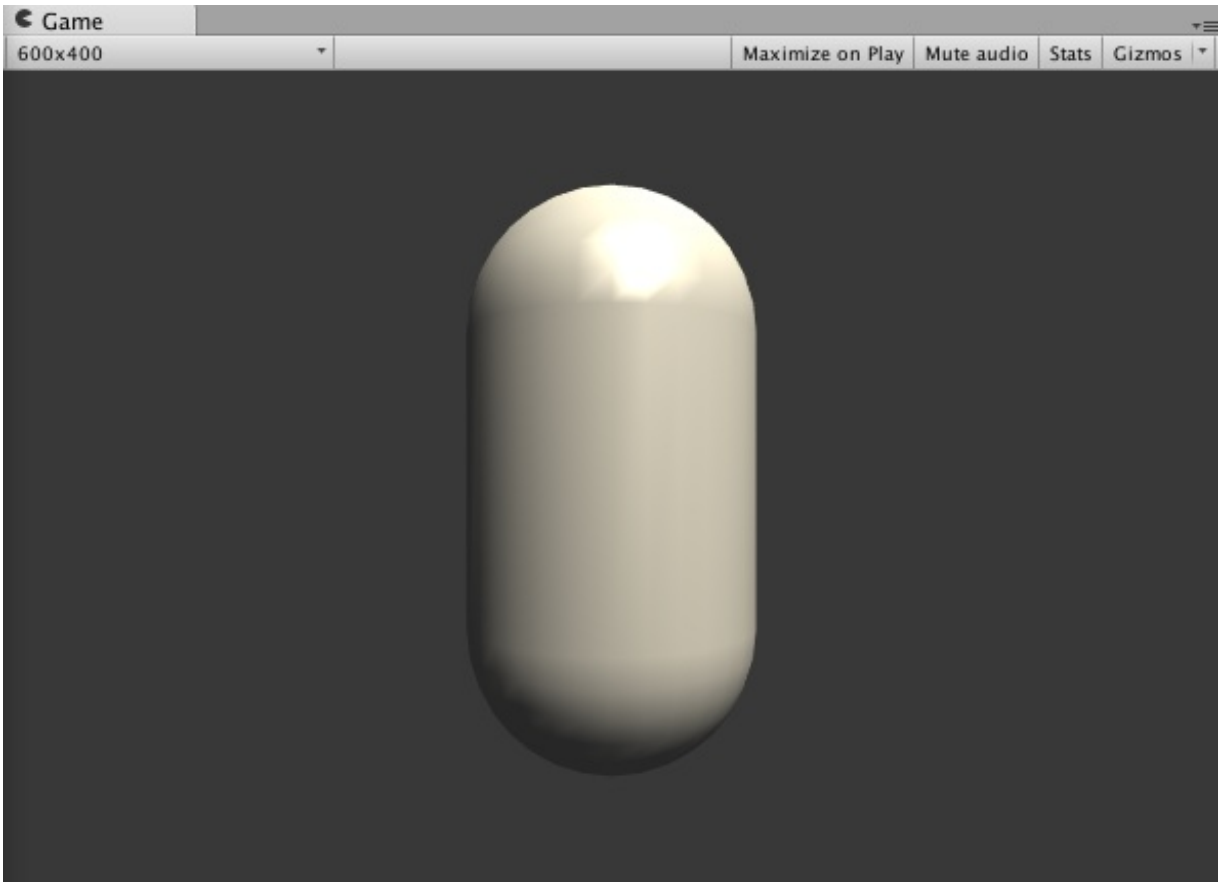


图6.10 逐顶点的高光反射光照效果

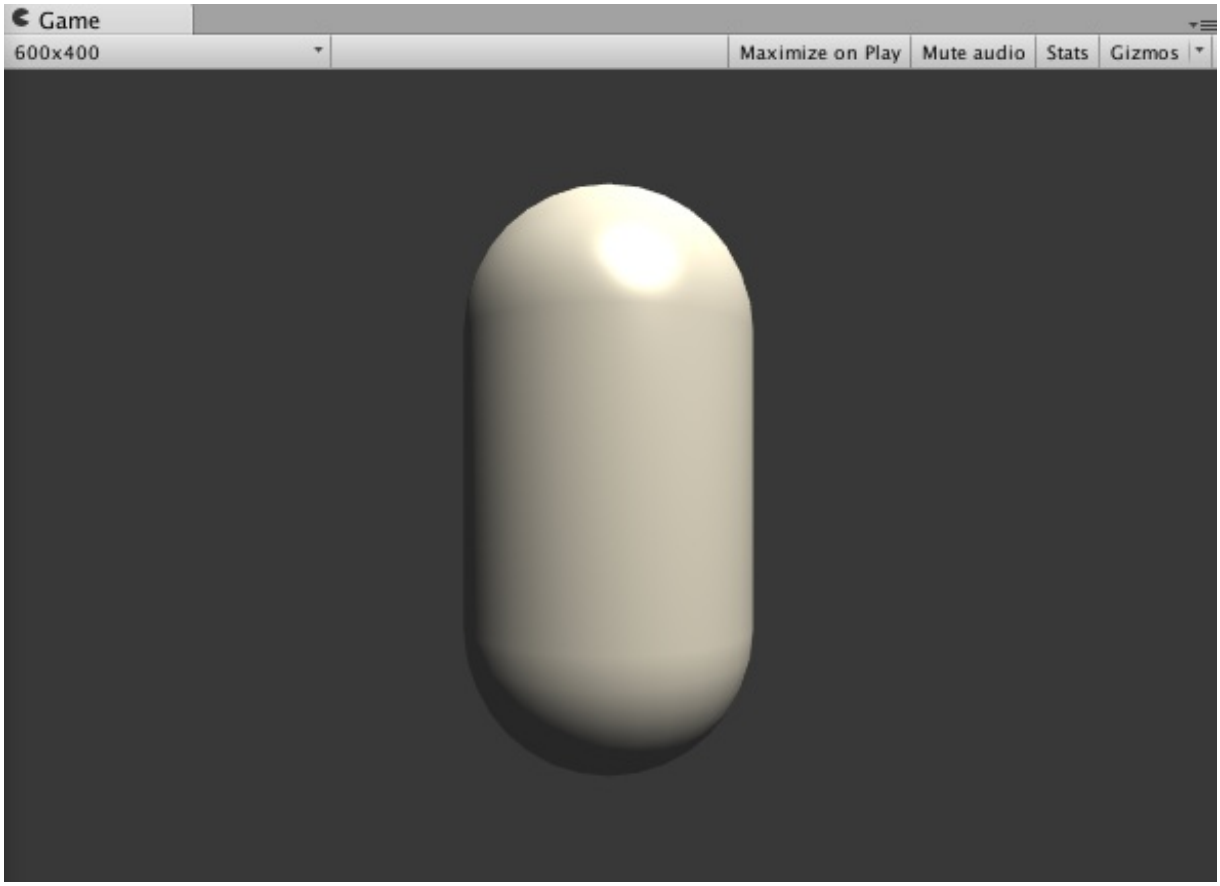


图6.11 逐像素的高光反射光照效果

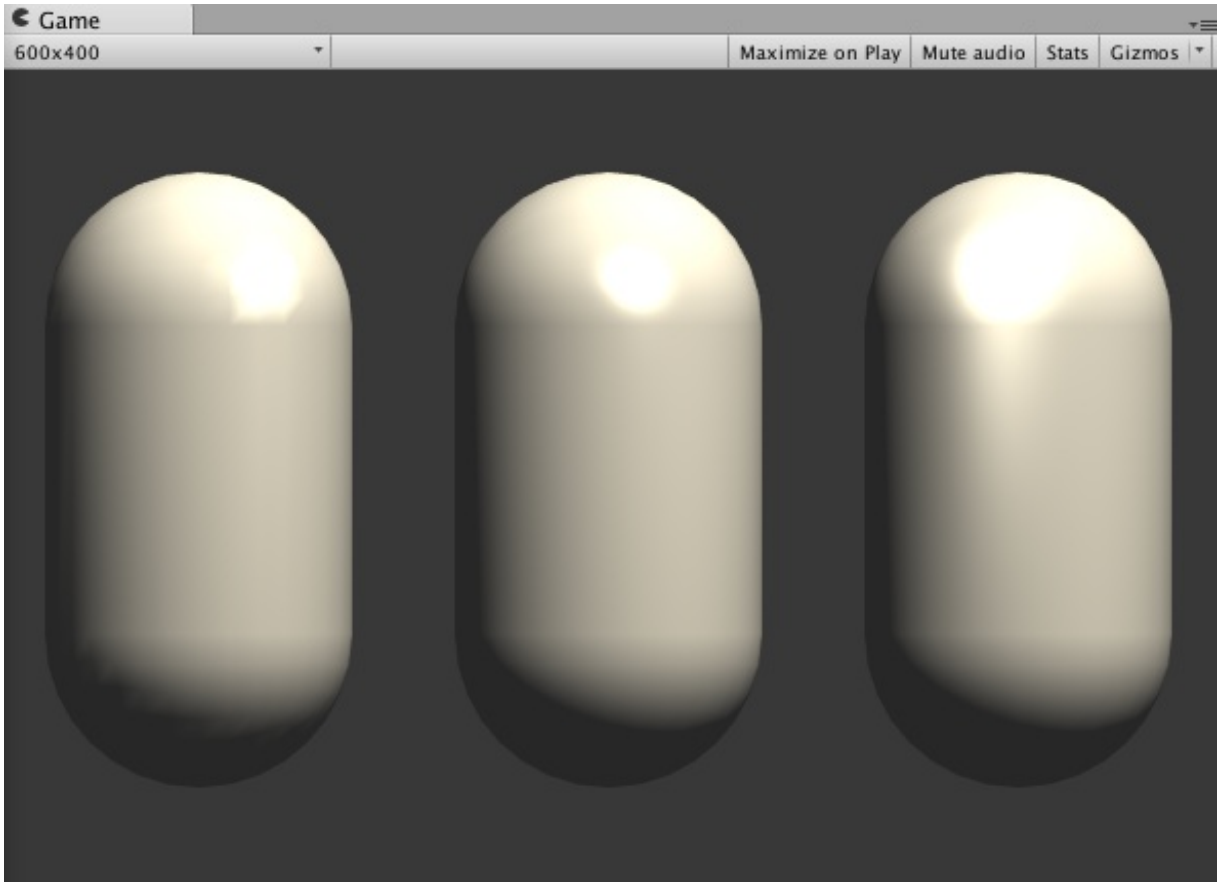


图6.12 逐顶点的高光反射光照、逐像素的高光反射光照（Phong光照模型）和Blinn-Phong高光反射光照的对比结果

第7章 基础纹理

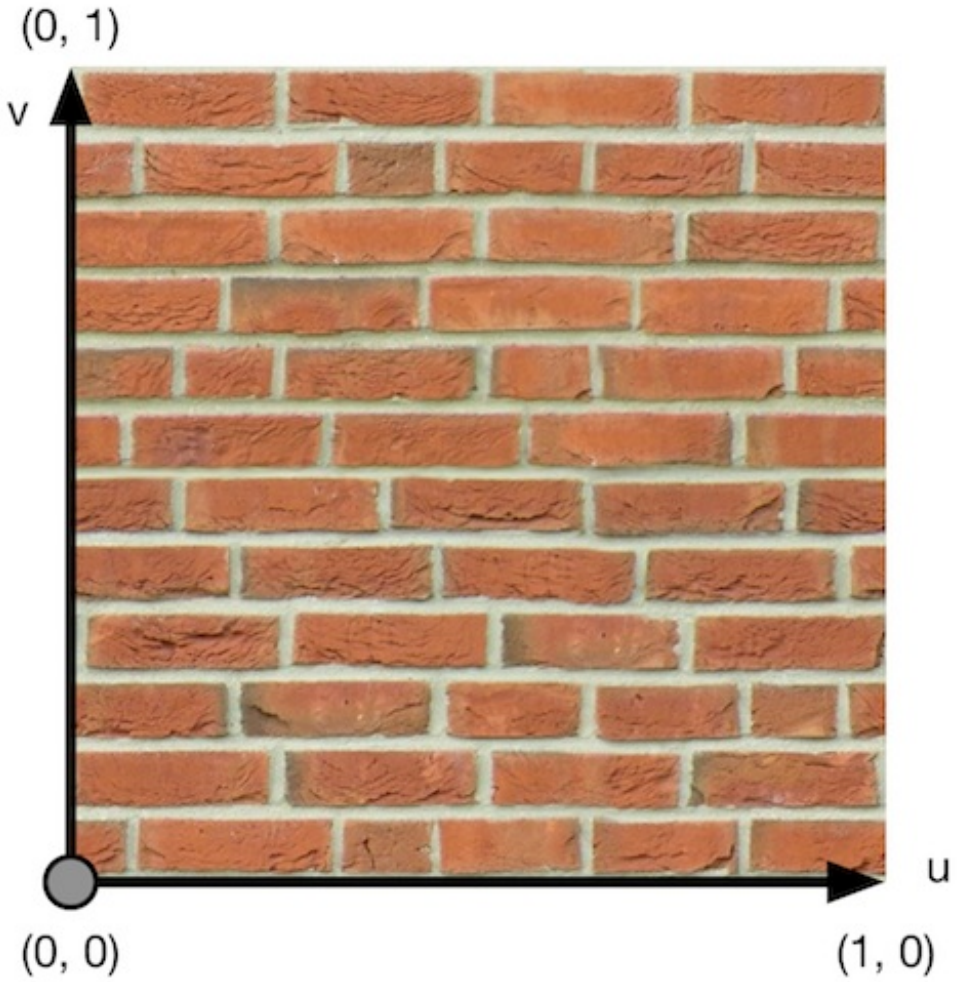


图7.1 Unity中的纹理坐标

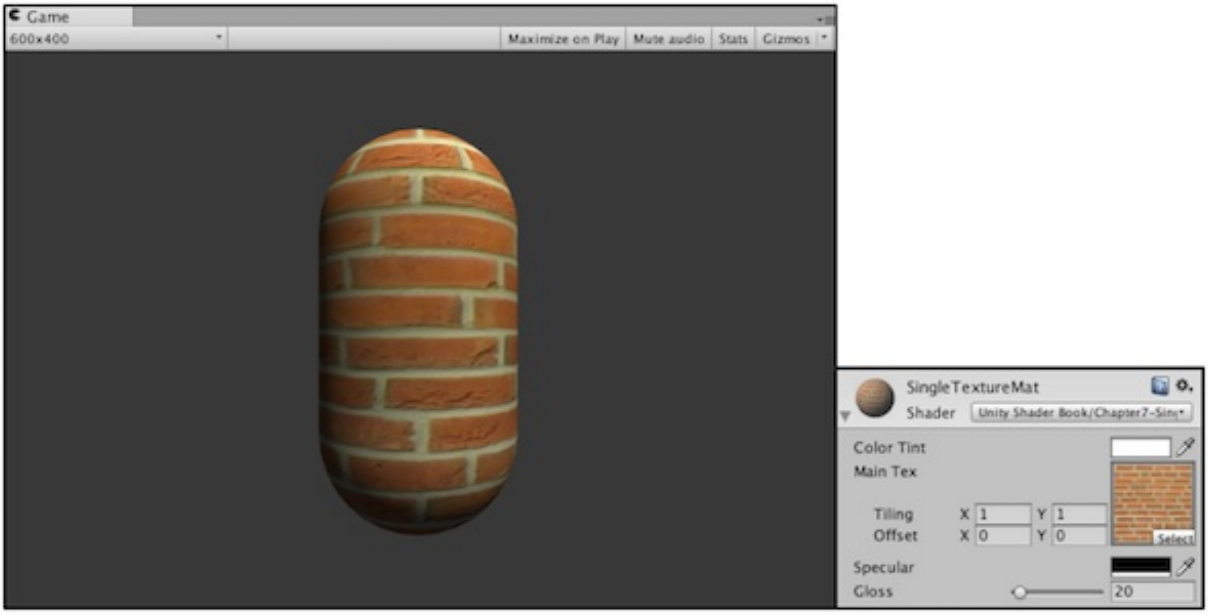


图7.2 使用单张纹理

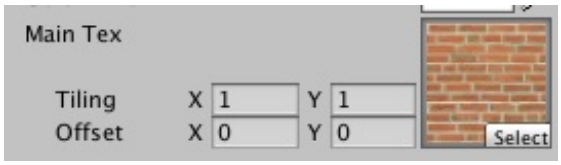


图7.3 调节纹理的平铺（缩放）和偏移（平移）属性

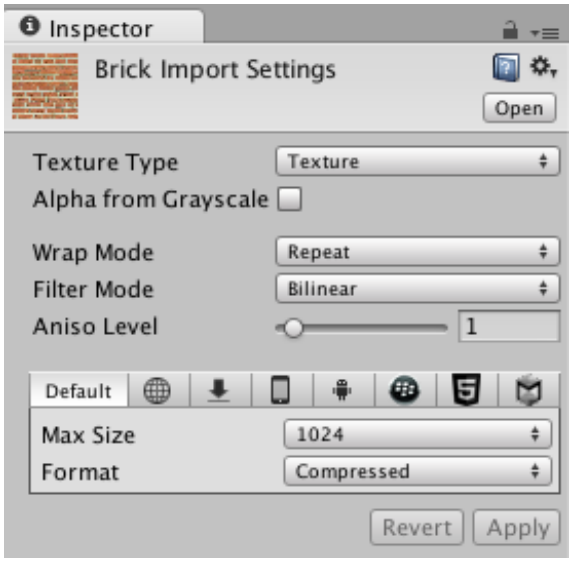


图7.4 纹理的属性

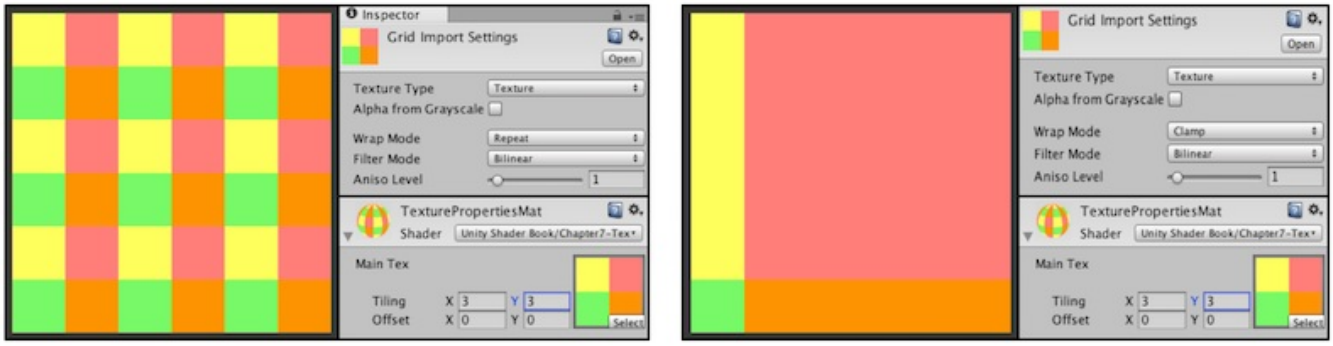


图7.5 Wrap Mode决定了当纹理坐标超过[0, 1]范围后将会如何被平铺

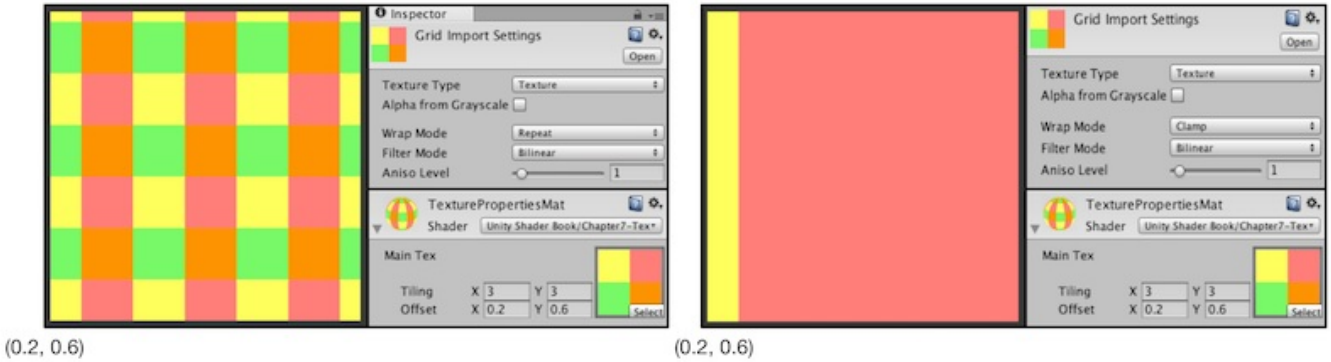


图7.6 偏移（Offset）属性决定了纹理坐标的偏移量

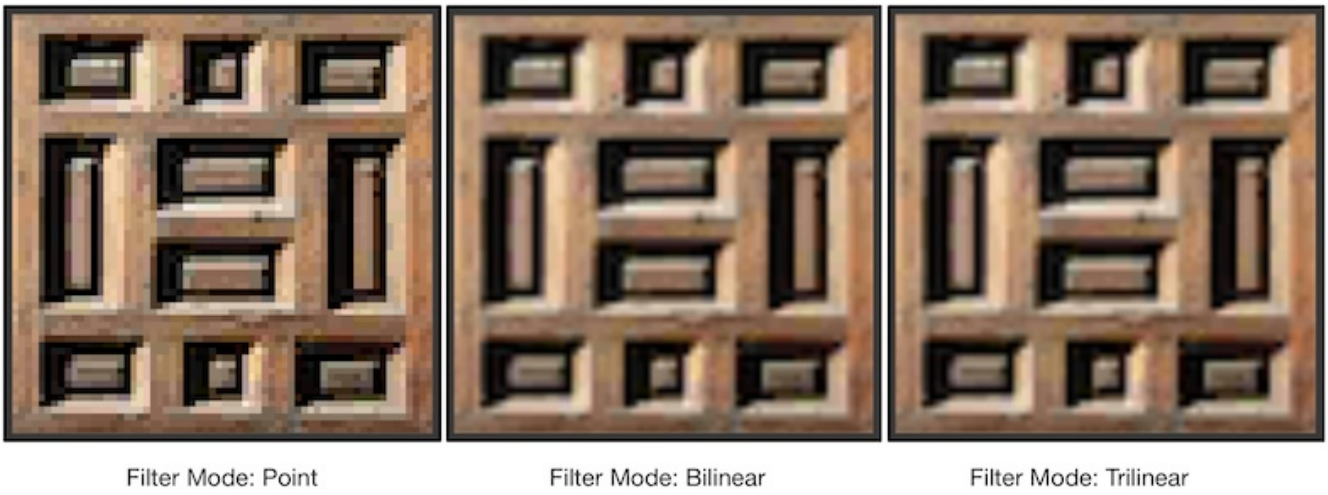


图7.7 在放大纹理时，分别使用三种Filter Mode得到的结果

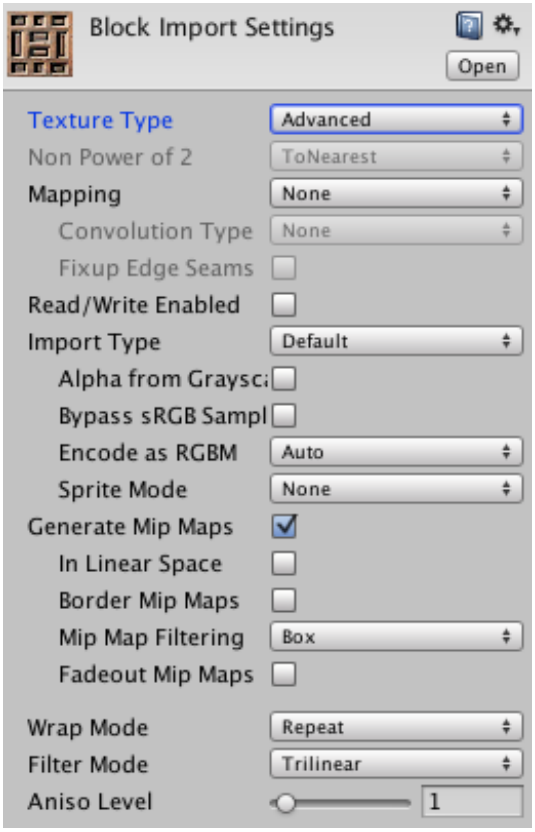


图7.8 在Advanced模式下可以设置多级渐远纹理的相关属性

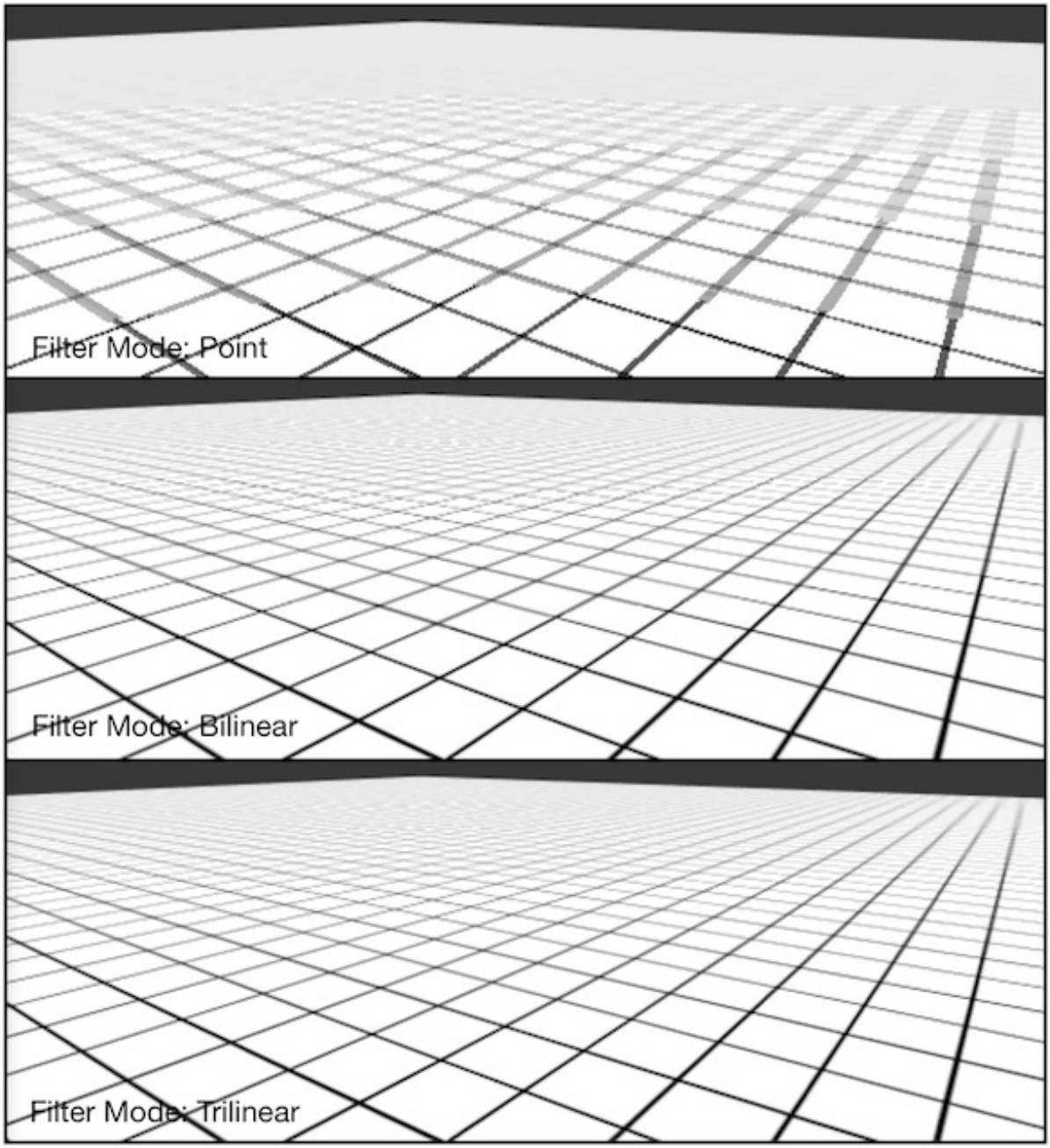


图7.9 从上到下： Point滤波 + 多级渐远纹理技术，Bilinear滤波 + 多级渐远纹理技术，Trilinear滤波 + 多级渐远纹理技术

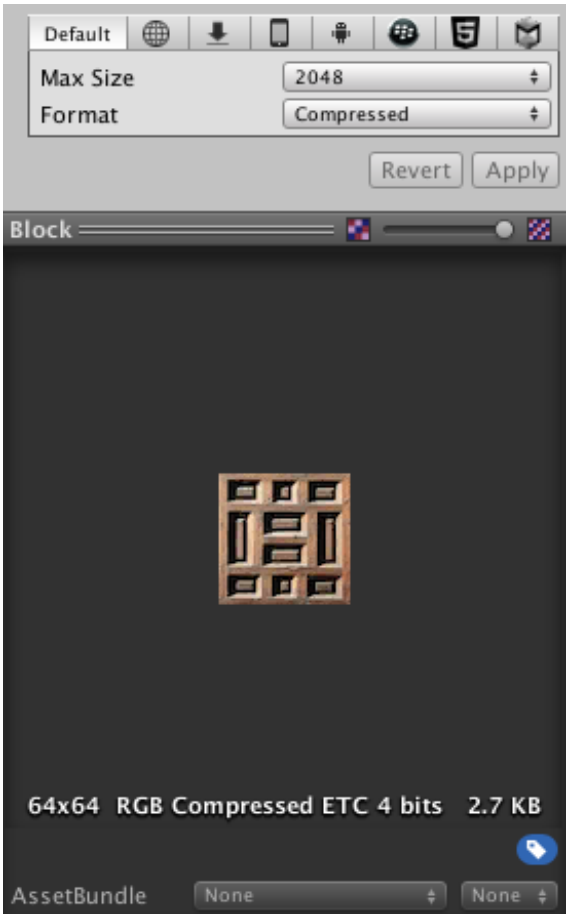


图7.10 选择纹理的最大尺寸和纹理模式

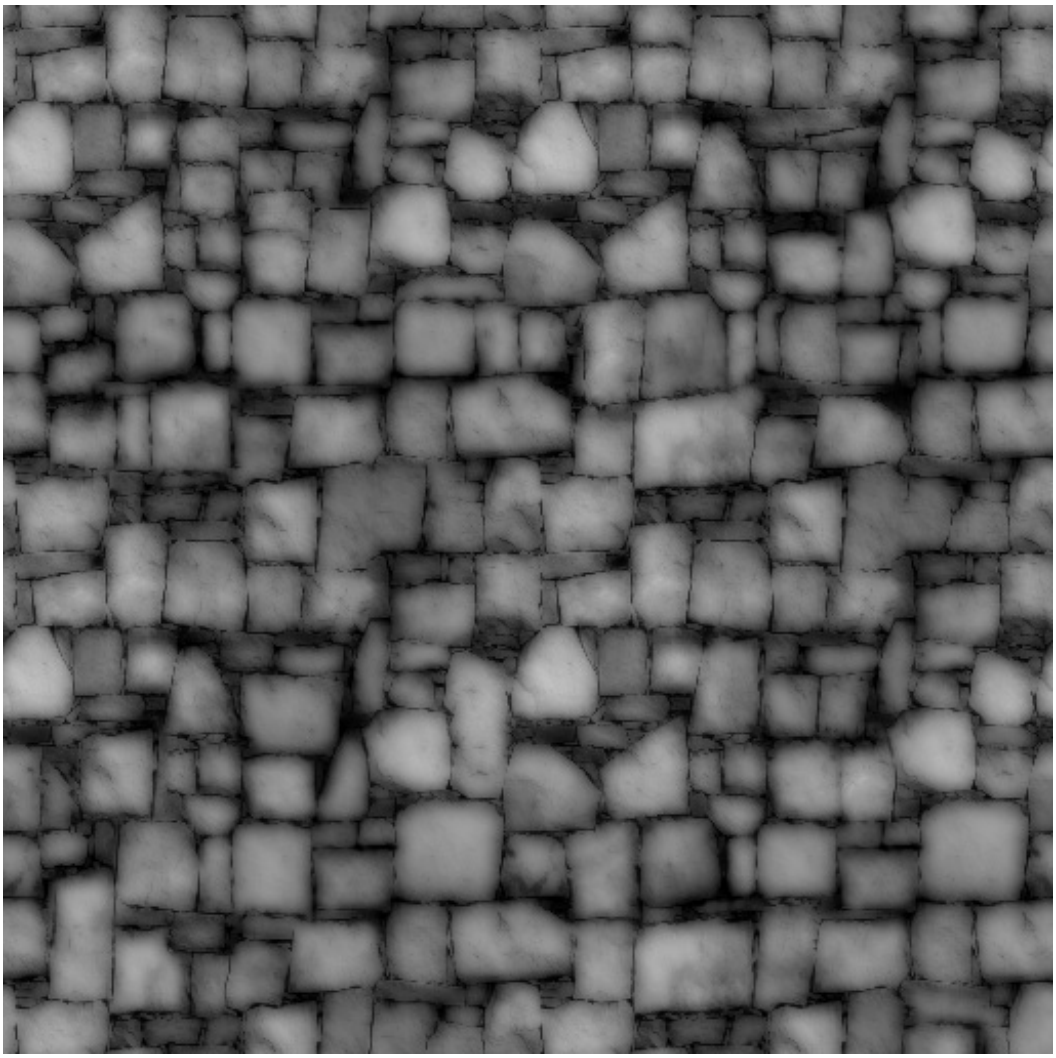


图7.11 高度图

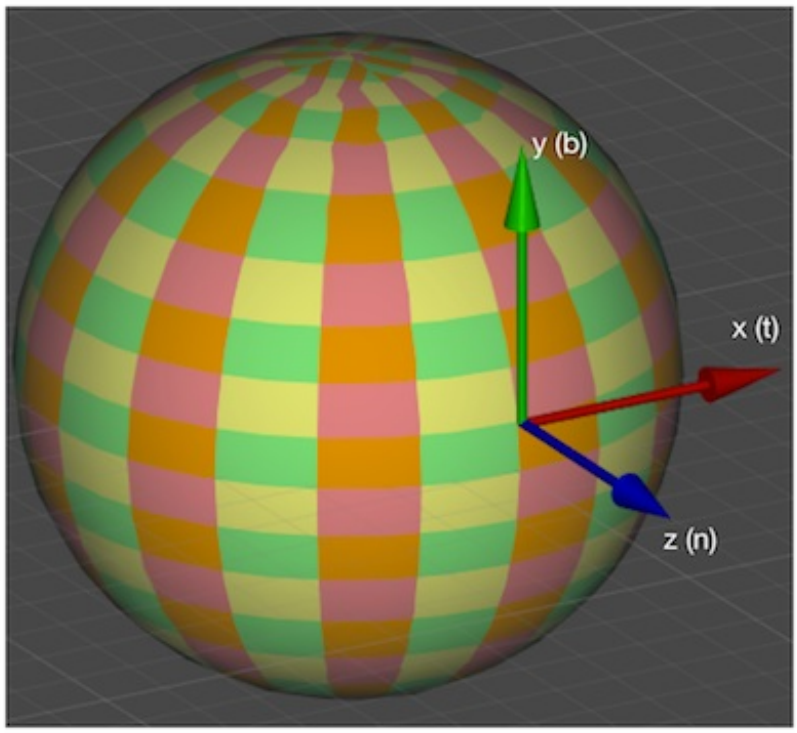


图7.12 模型顶点的切线空间。其中，原点对应了顶点坐标，x轴是切线方向（t），y轴是副切线方向（b），z轴是法线方向（n）

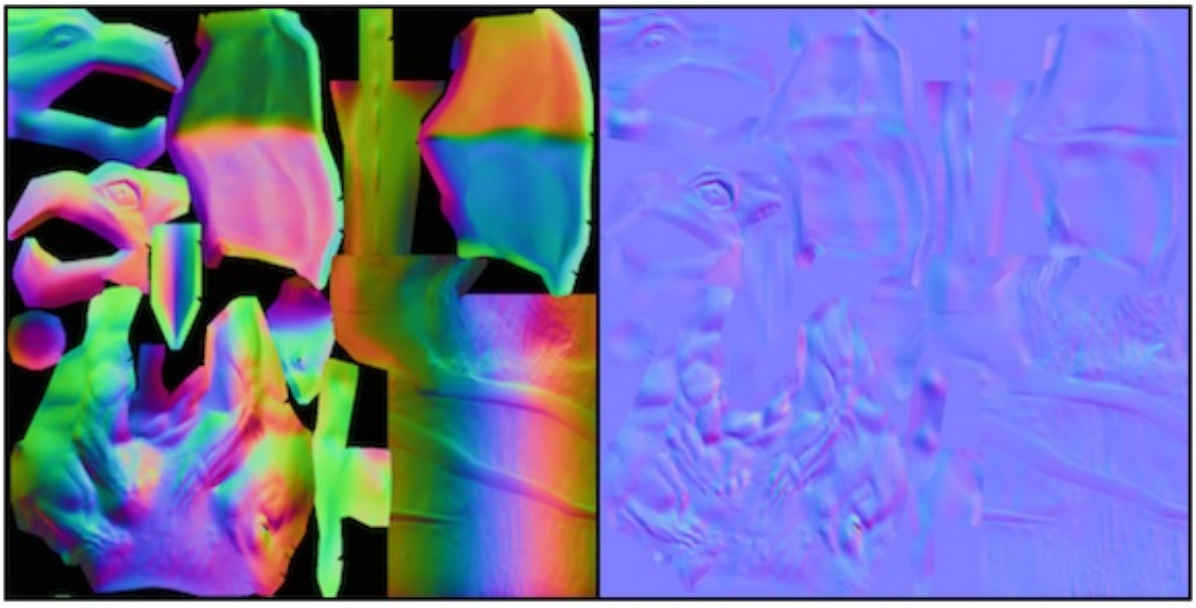


图7.13 左图：模型空间下的法线纹理。右图：切线空间下的法线纹理

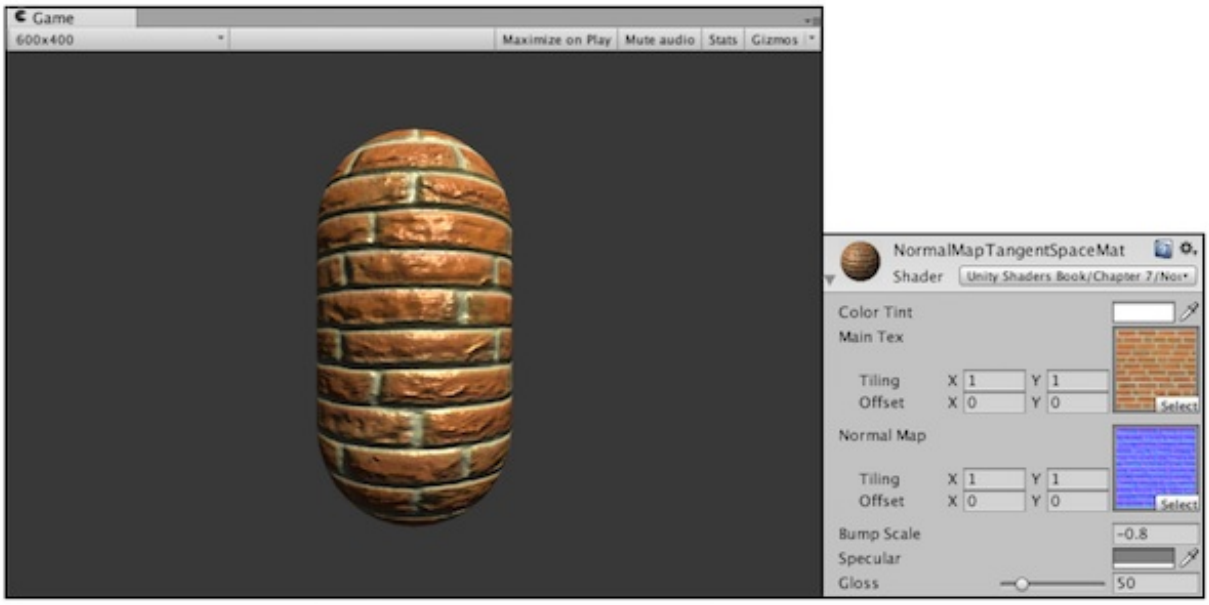


图7.14 使用法线纹理

图7.15 使用Bump Scale属性来调整模型的凹凸程度

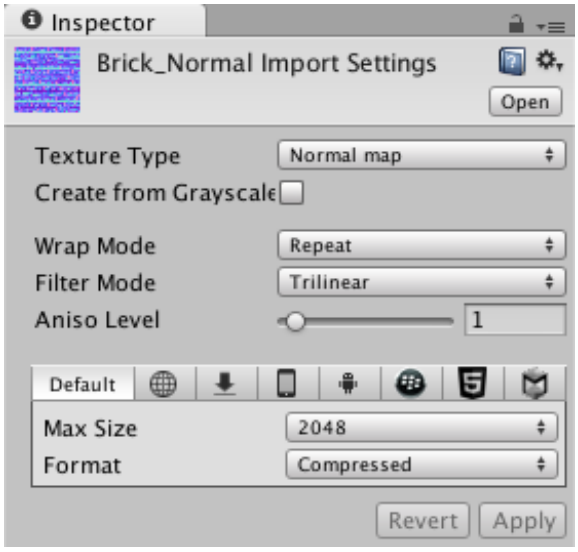


图7.16 当使用UnpackNormal函数计算法线纹理中的法线方向时，需要把纹理类型标识为Normal map

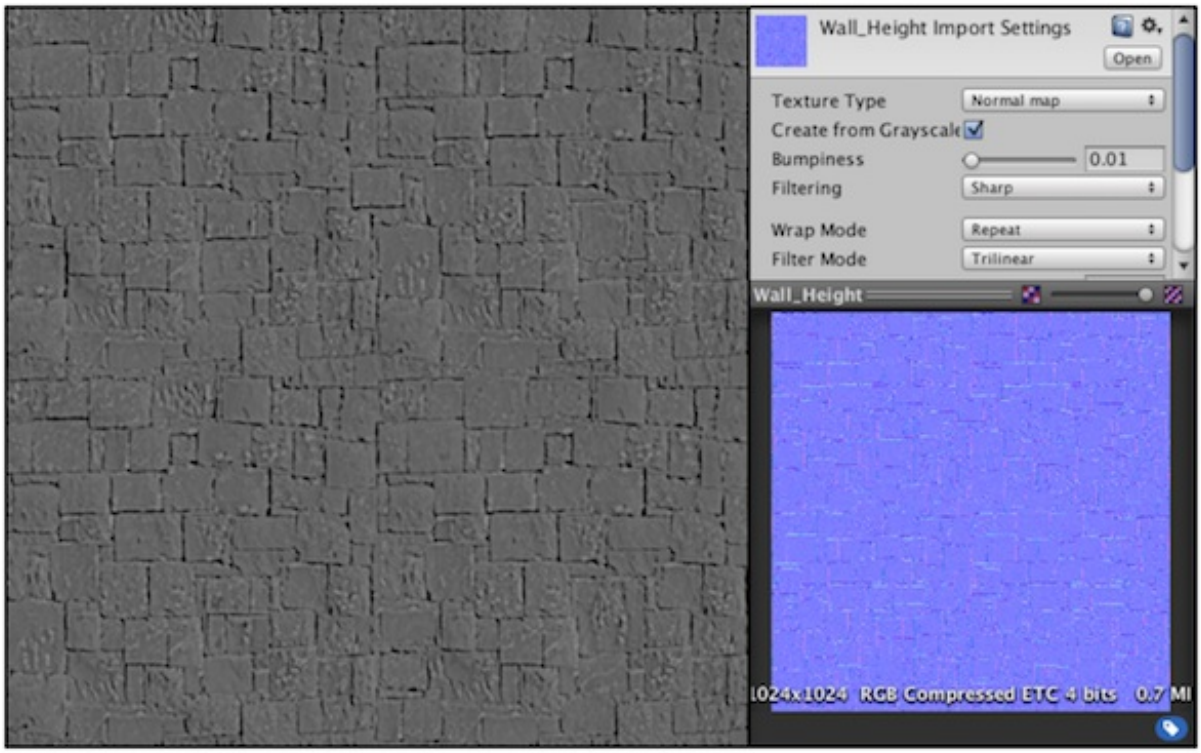


图7.17 当勾选了Create from Grayscale后，Unity会根据高度图来生成一张切线空间下的法线纹理

图7.18 使用不同的渐变纹理控制漫反射光照，左下角给出了每张图使用的渐变纹理

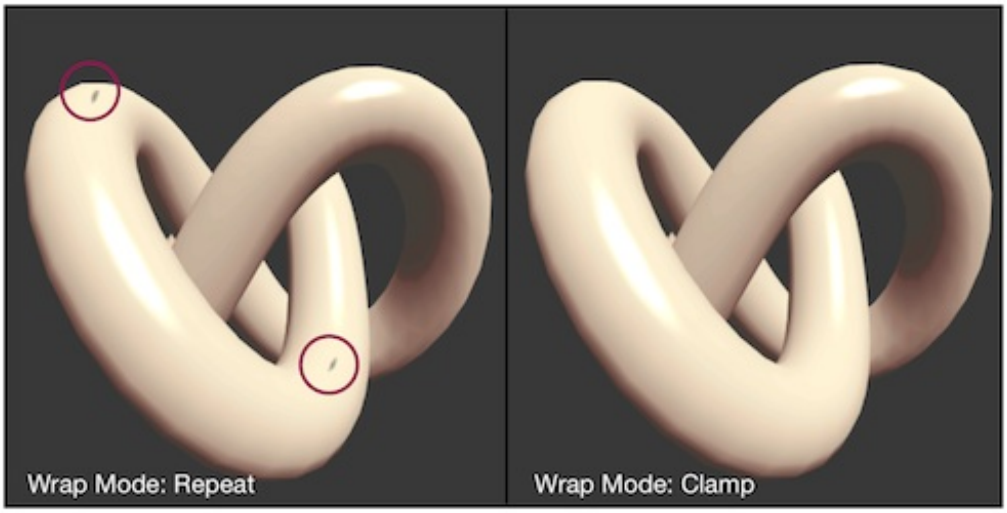


图7.19 Wrap Mode分别为Repeat和Clamp模式的效果对比

图7.20 使用高光遮罩纹理。从左到右：只包含漫反射，未使用遮罩的高光反射，使用遮罩的高光反射

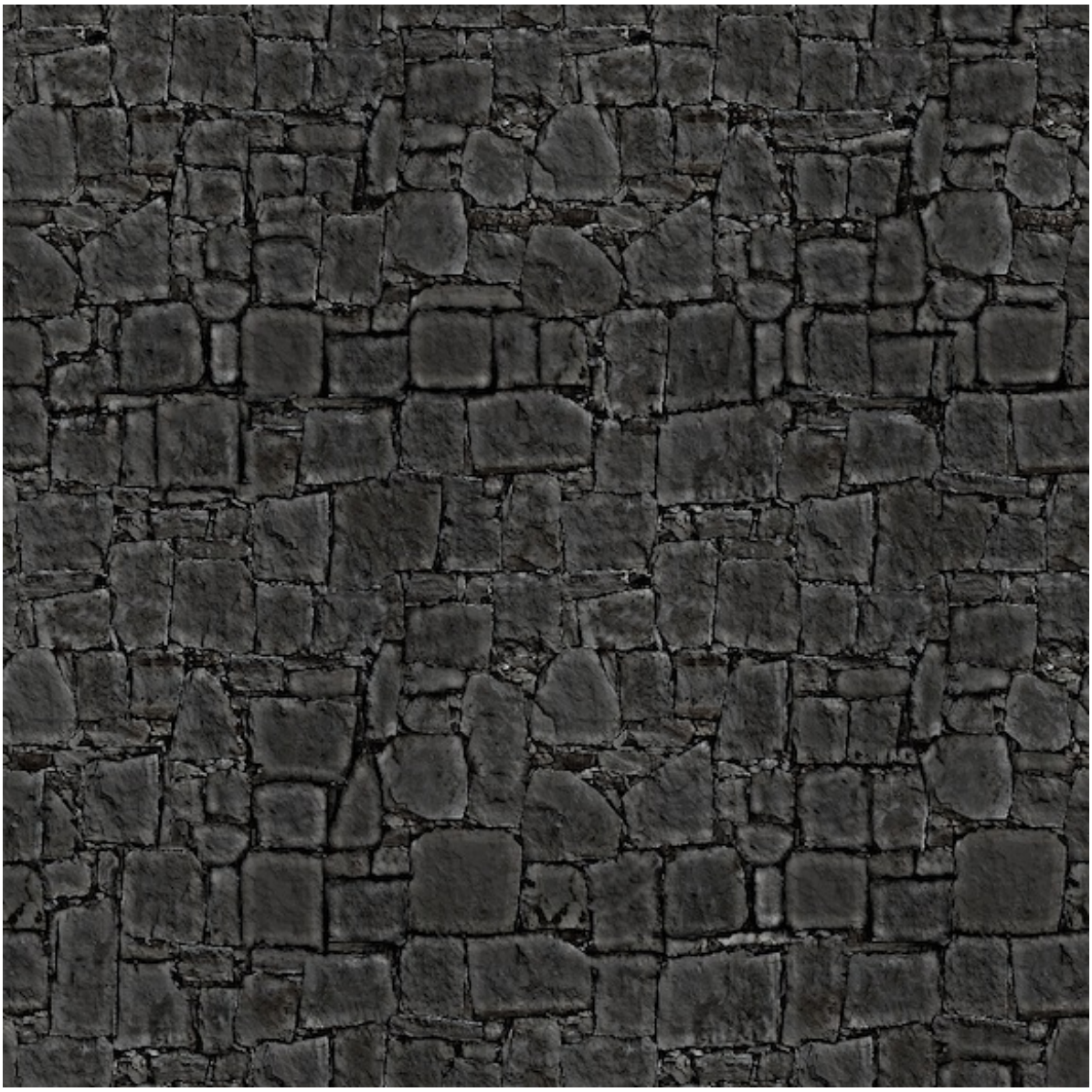


图7.21 本节使用的高光遮罩纹理

第8章 透明效果

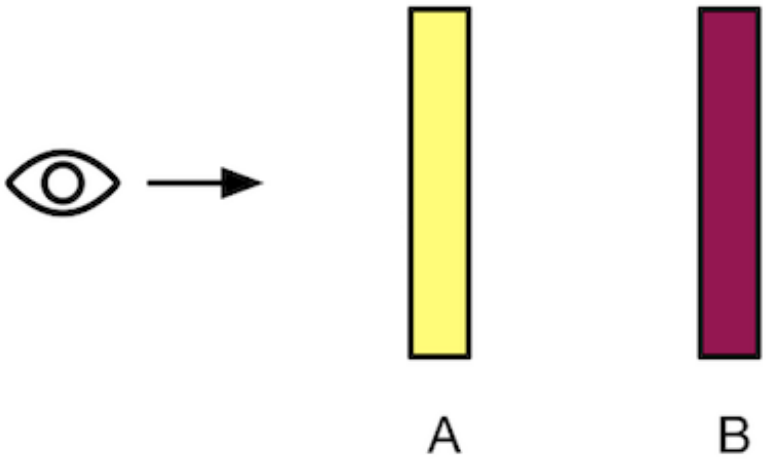


图8.1 场景中有两个物体，其中A（黄色）是半透明物体，B（紫色）是不透明物体

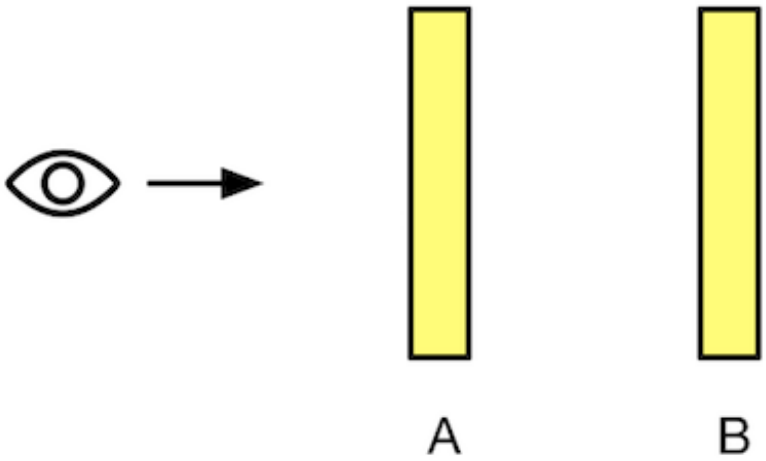


图8.2 场景中有两个物体，其中A和B都是半透明物体

图8.3 循环重叠的半透明物体总是无法得到正确的半透明效果

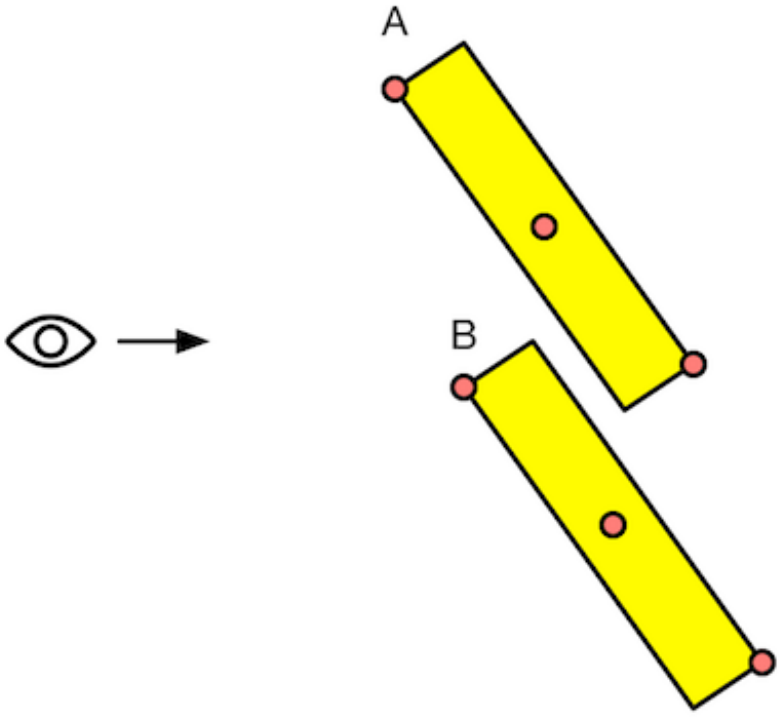


图8.4 使用哪个深度对物体进行排序。红色点分别标明了网格上距离摄像机最近的点、最远的点以及网格中点

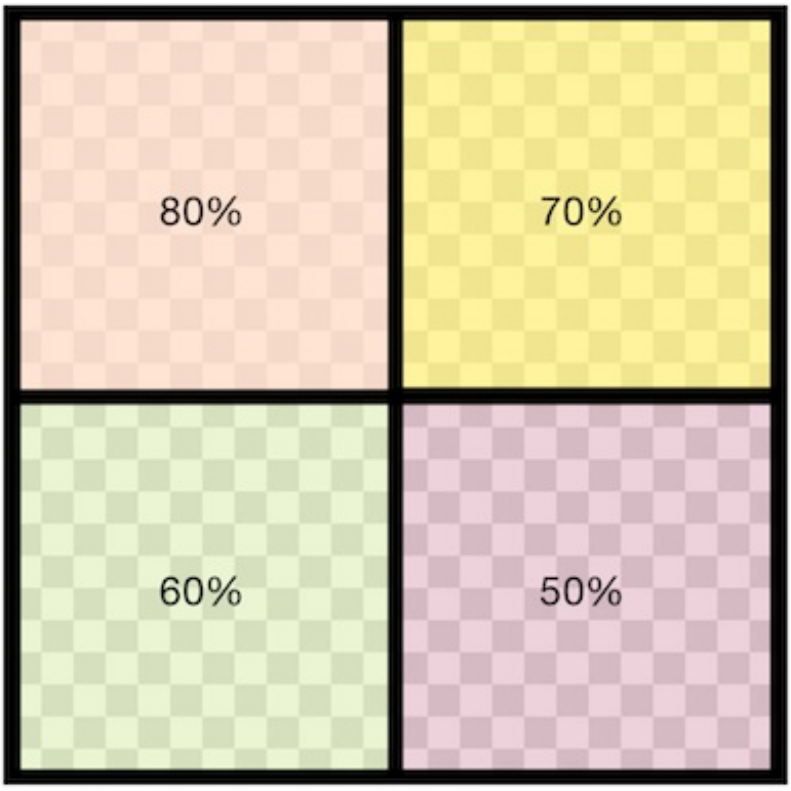


图8.5 一张透明纹理，其中每个方格的透明度都不同

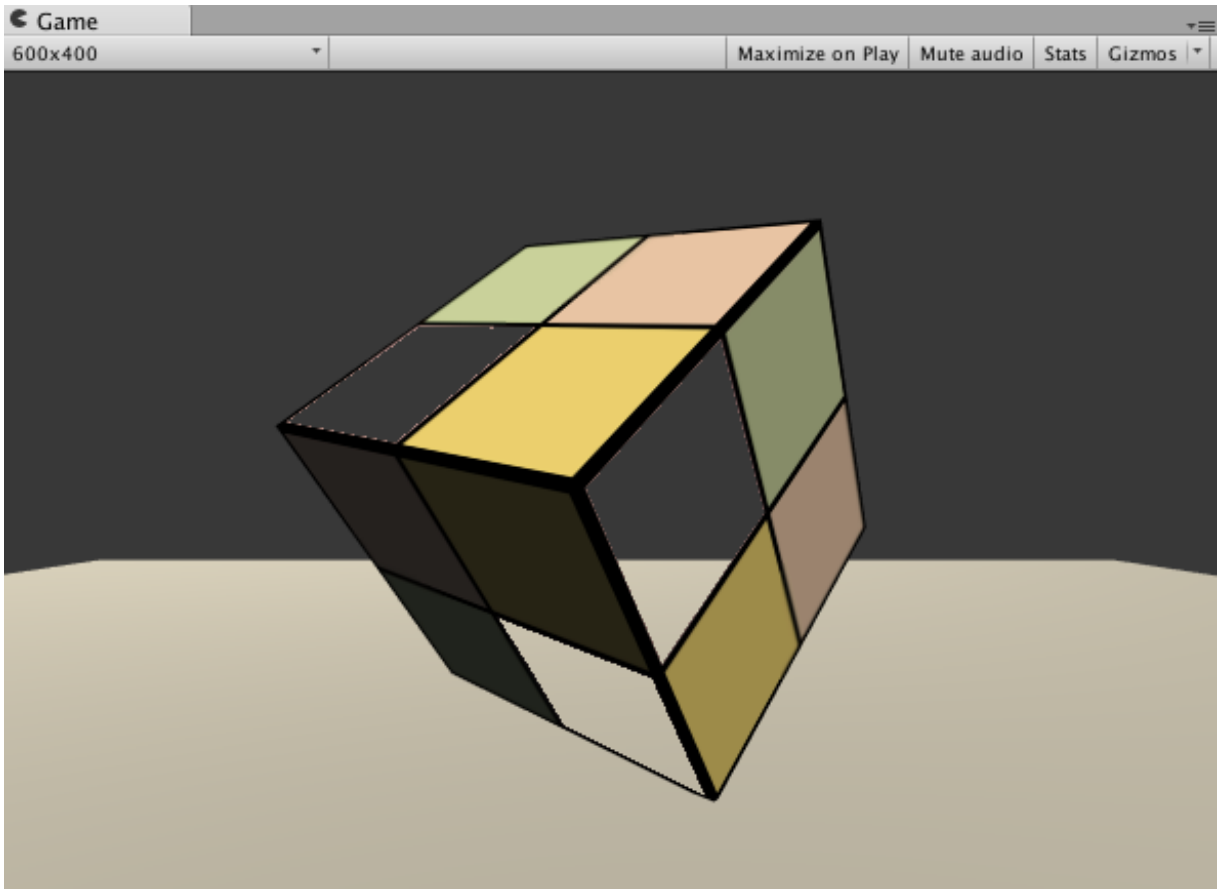


图8.6 透明度测试

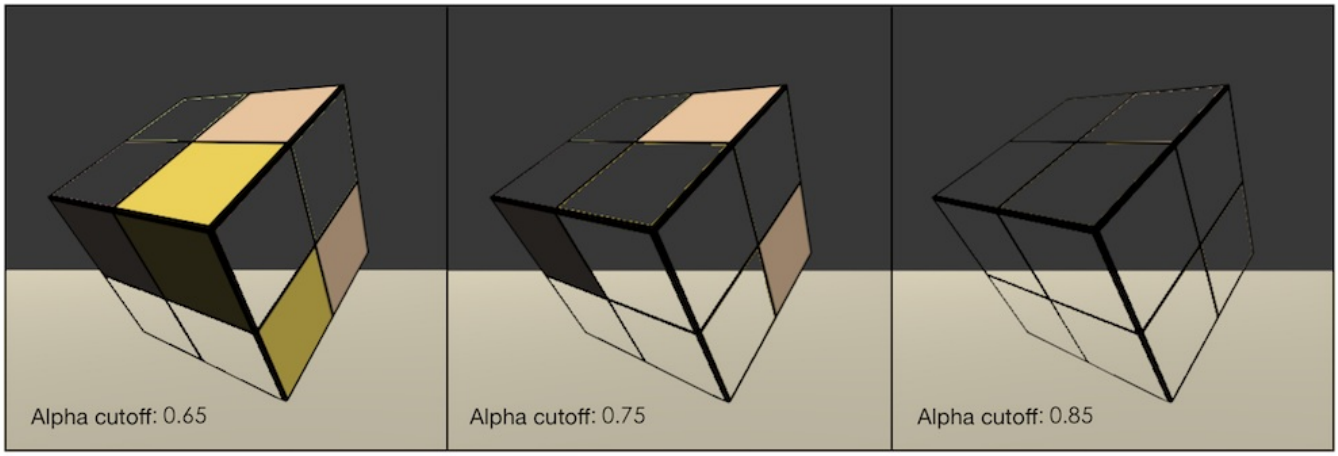


图8.7 随着Alpha cutoff参数的增大，更多的像素由于不满足透明度测试条件而被剔除

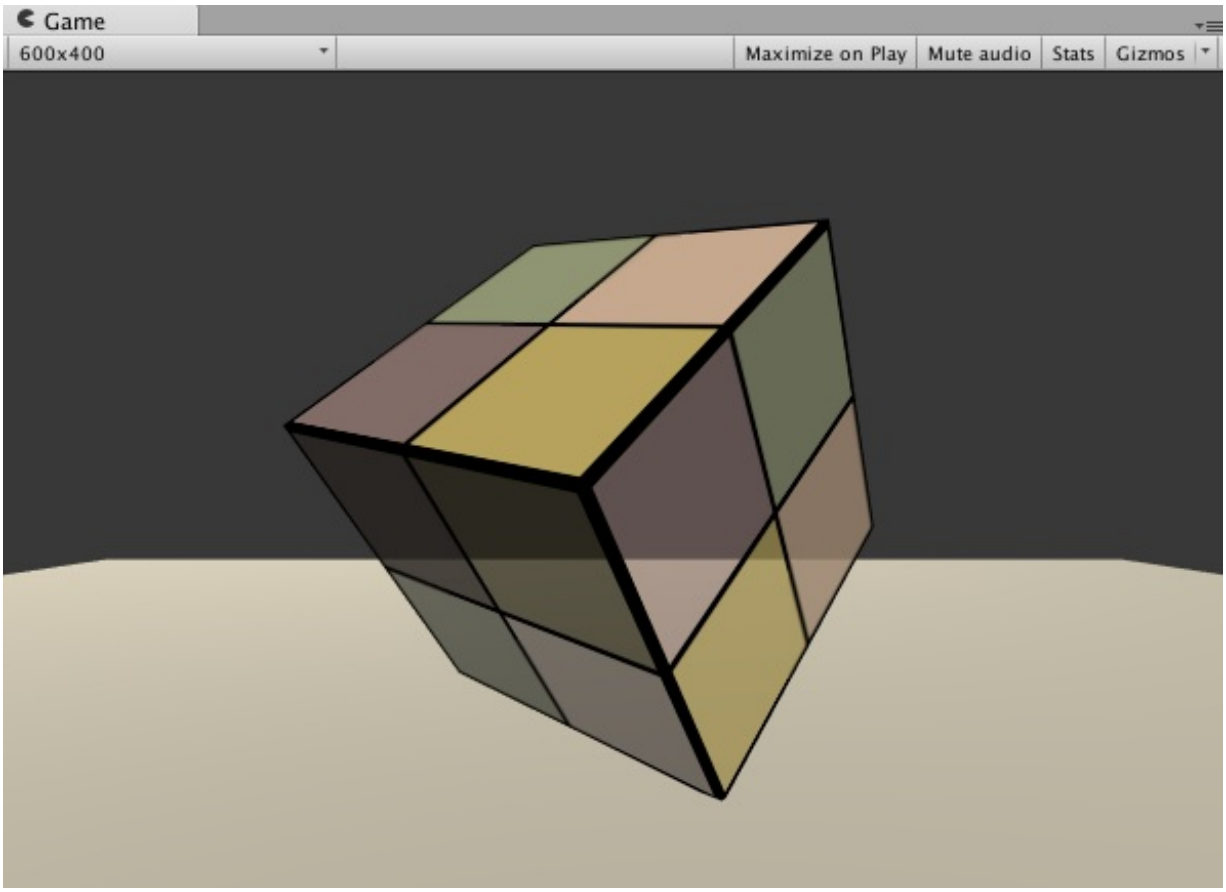


图8.8 透明度混合

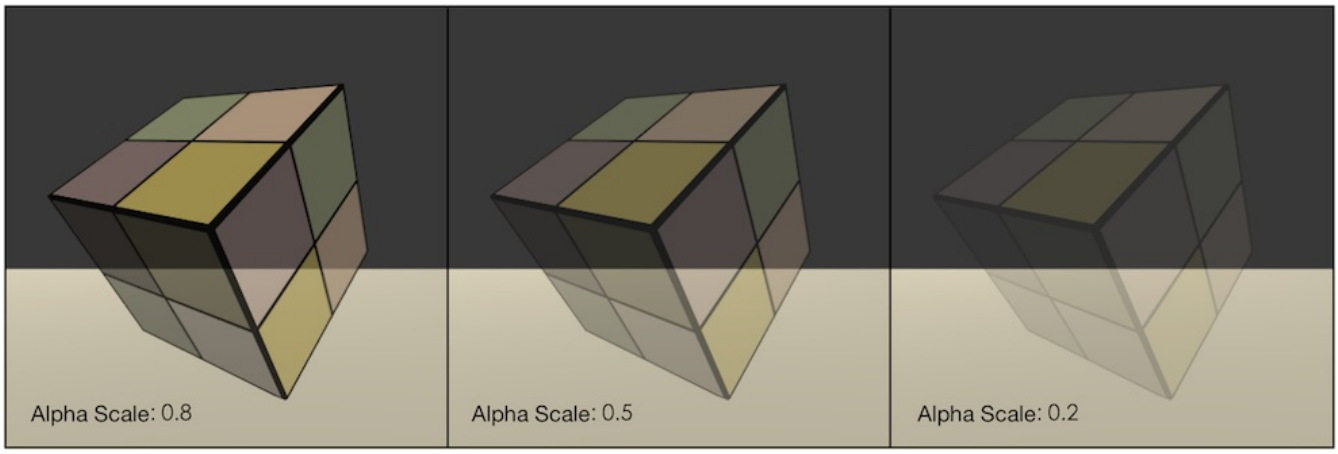


图8.9 随着Alpha Scale参数的增大，模型变得越来越透明

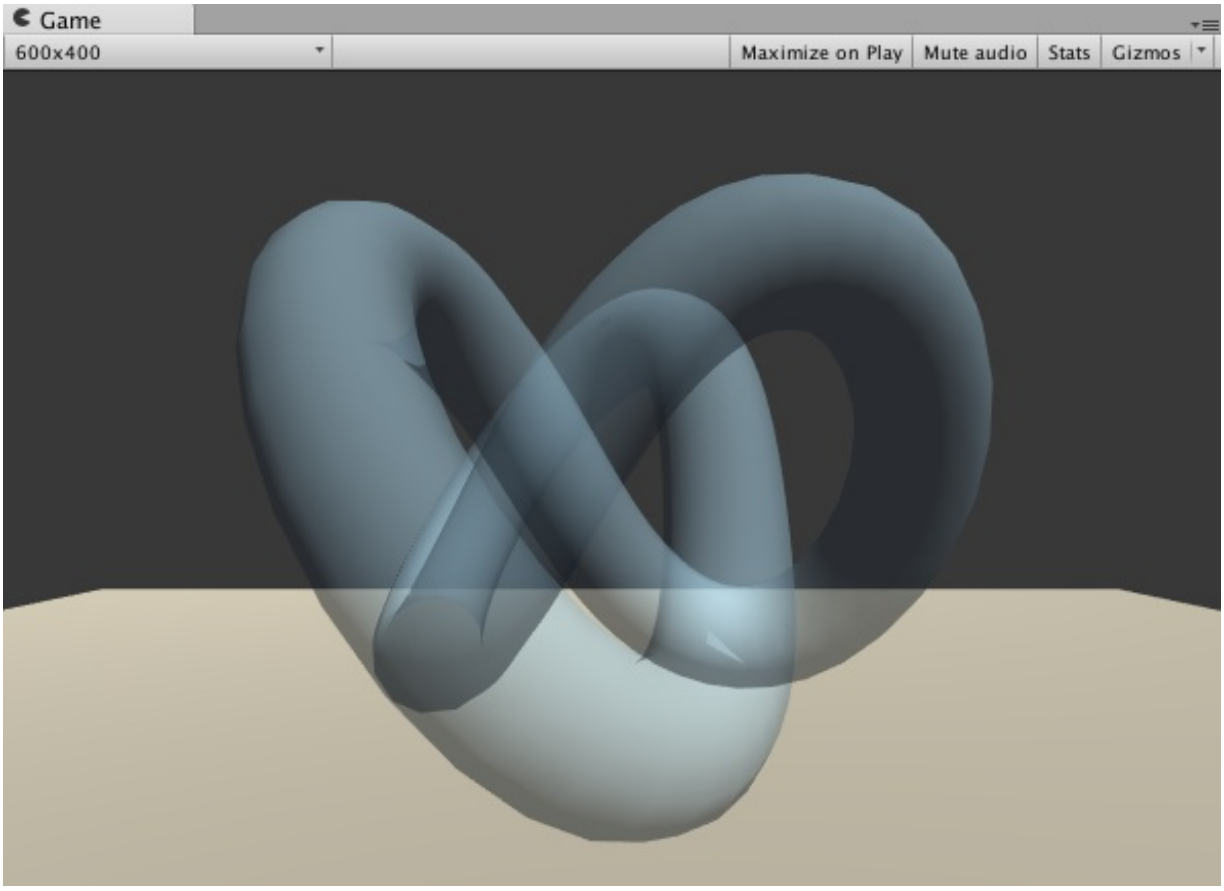


图8.10 当模型网格之间有互相交叉的结构时，往往会得到错误的半透明效果

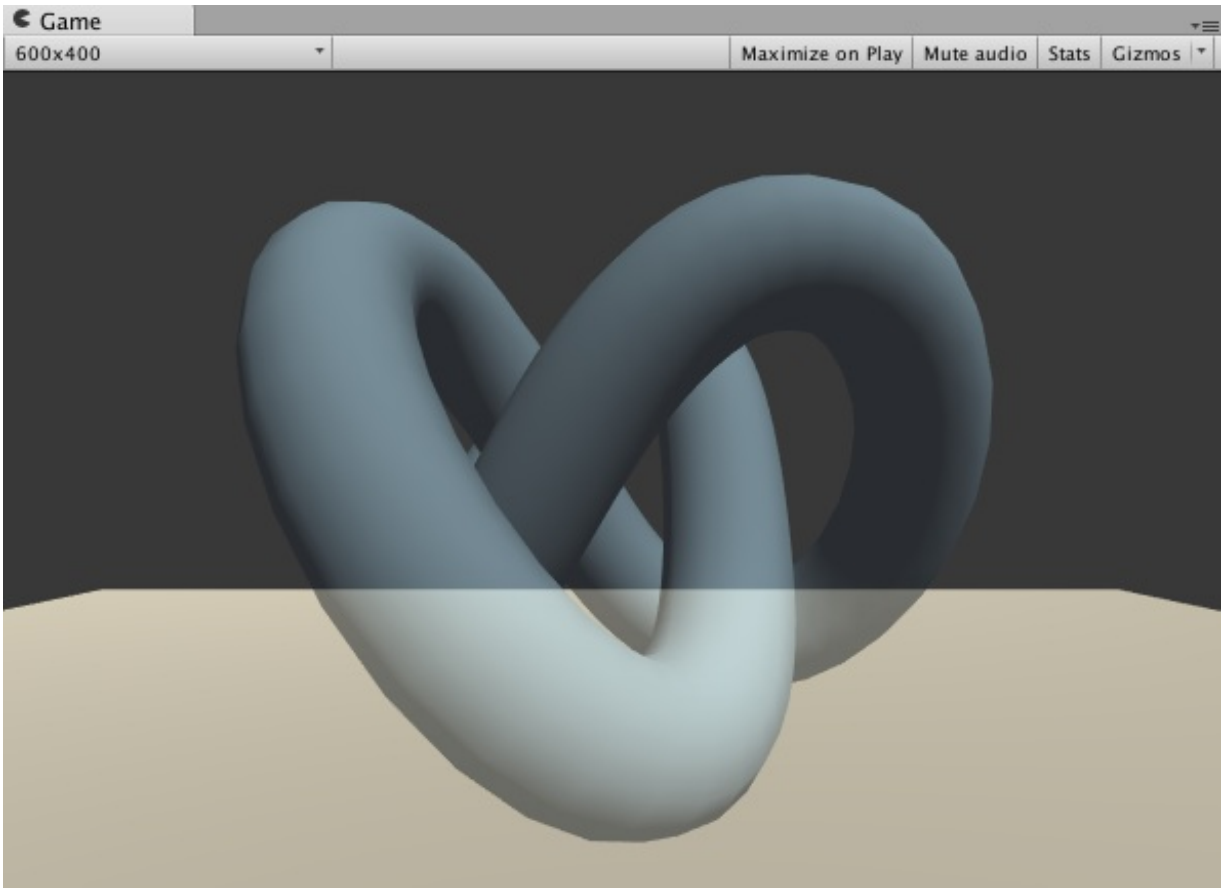


图8.11 开启了深度写入的半透明效果

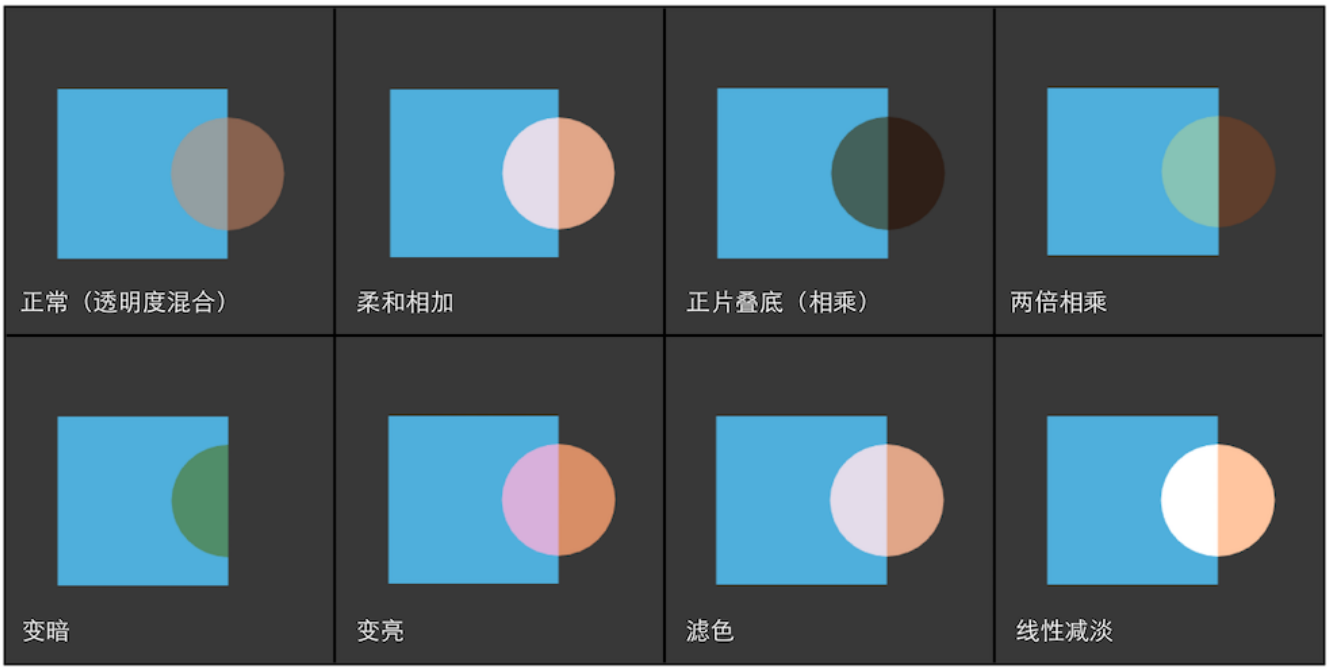


图8.12 不同混合状态设置得到的效果

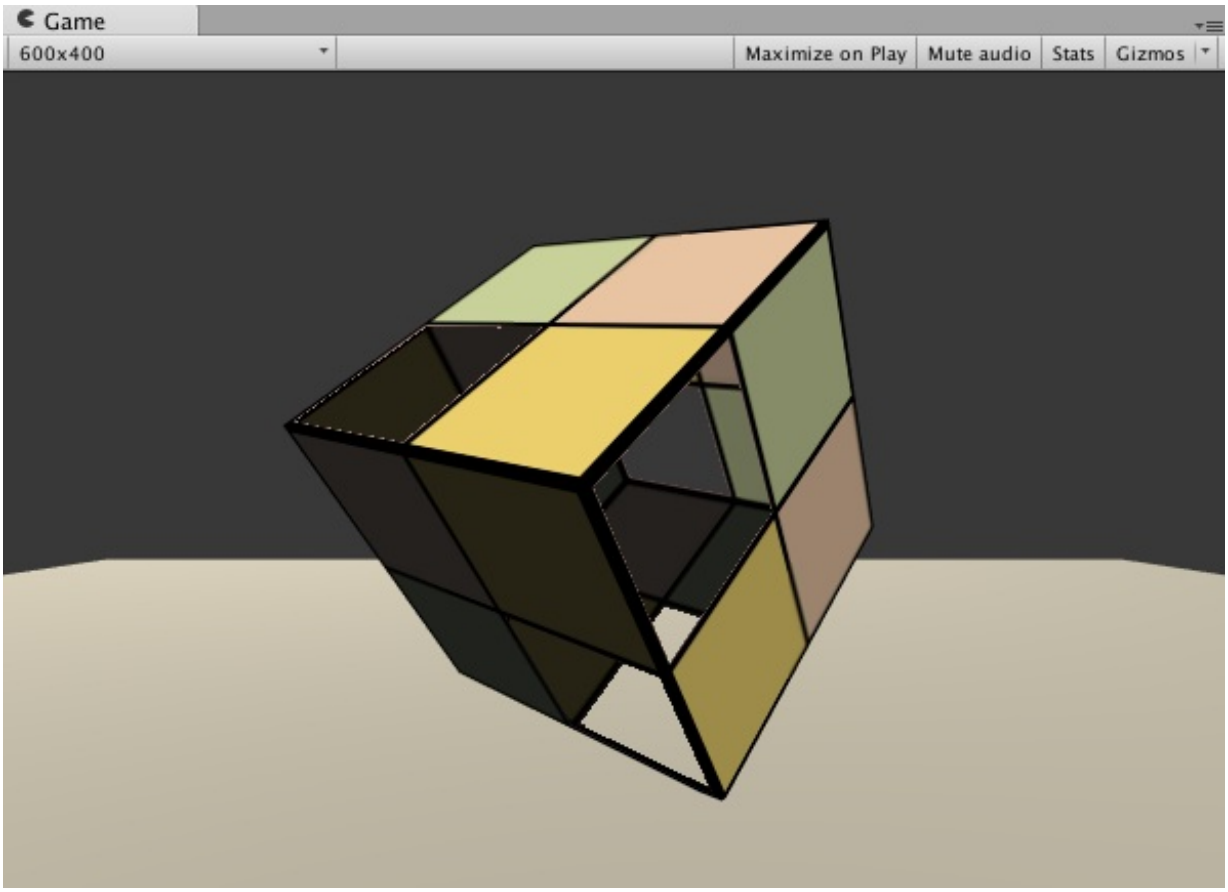


图8.13 双面渲染的透明度测试的物体

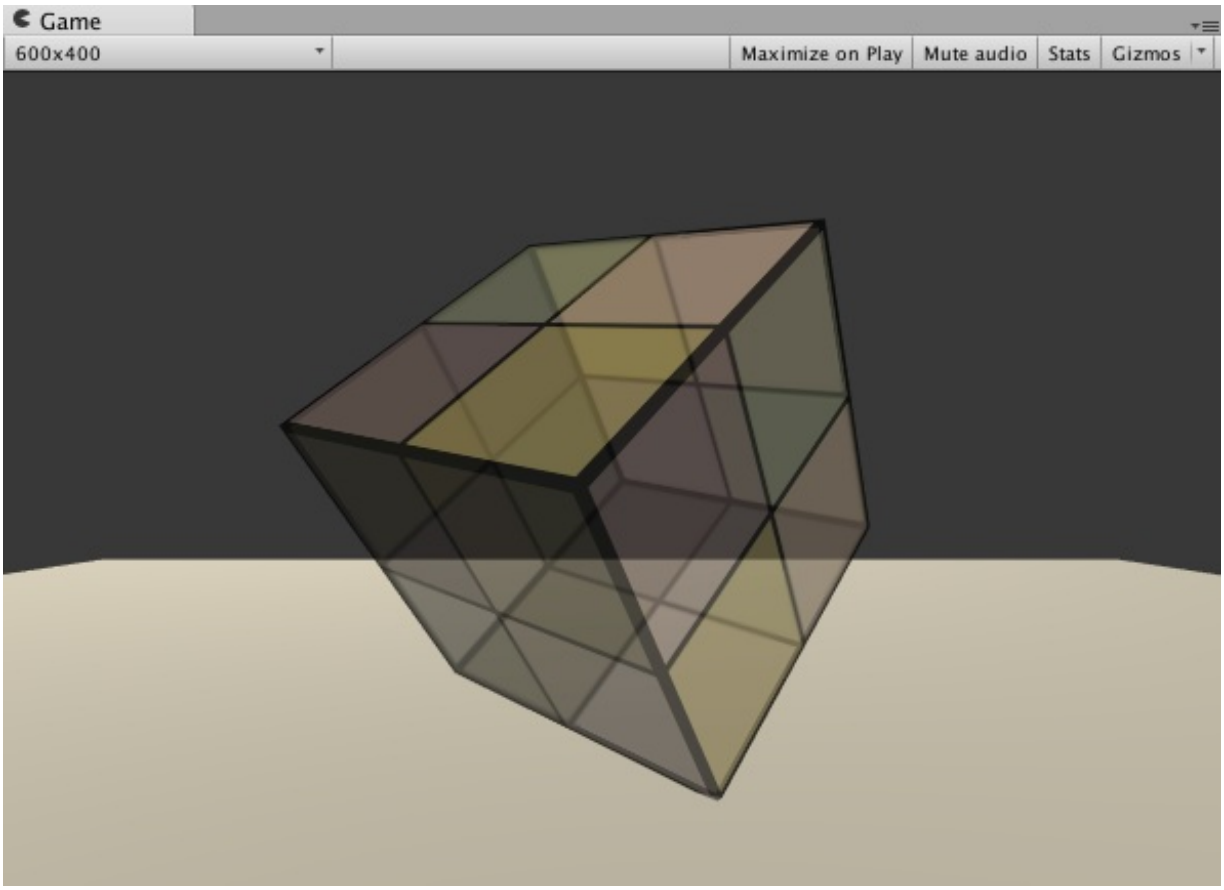


图8.14 双面渲染的透明度混合的物体

第9章 更复杂的光照

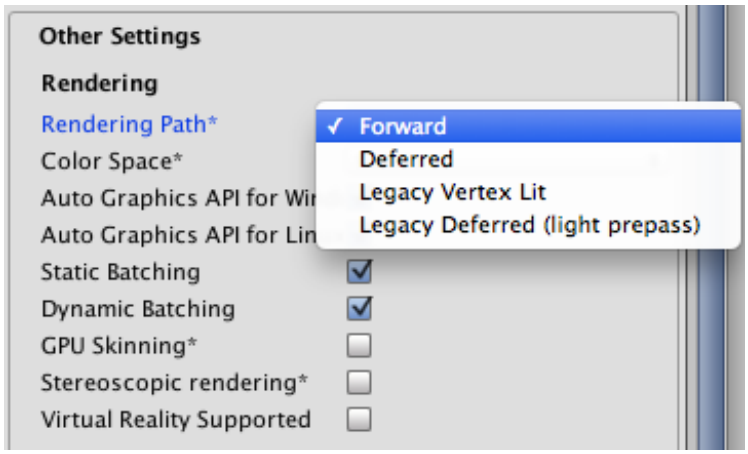


图9.1 设置Unity项目的渲染路径

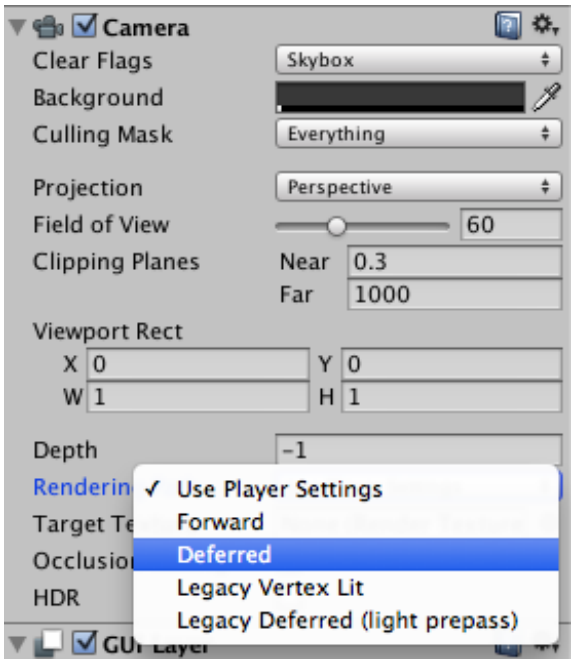


图9.2 摄像机组件的Rendering Path中的设置可以覆盖Project Settings中的设置

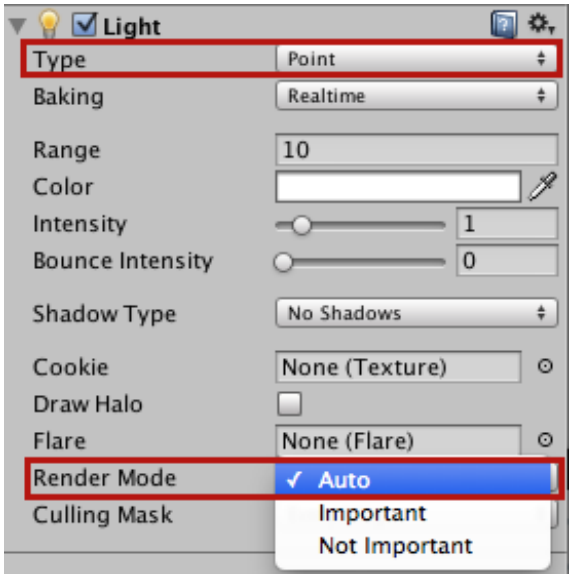


图9.3 设置光源的类型和渲染模式

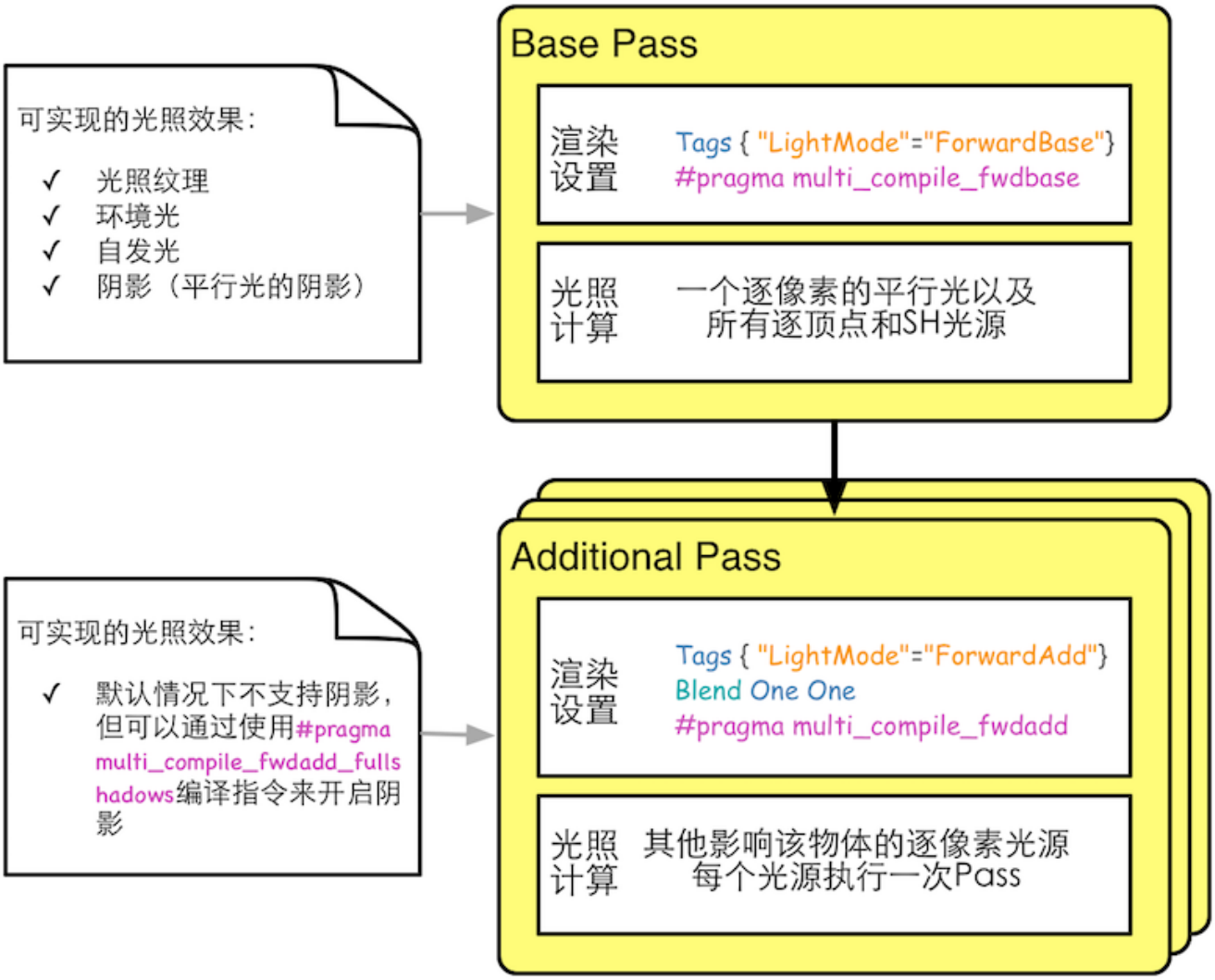


图9.4 前向渲染的两种Pass

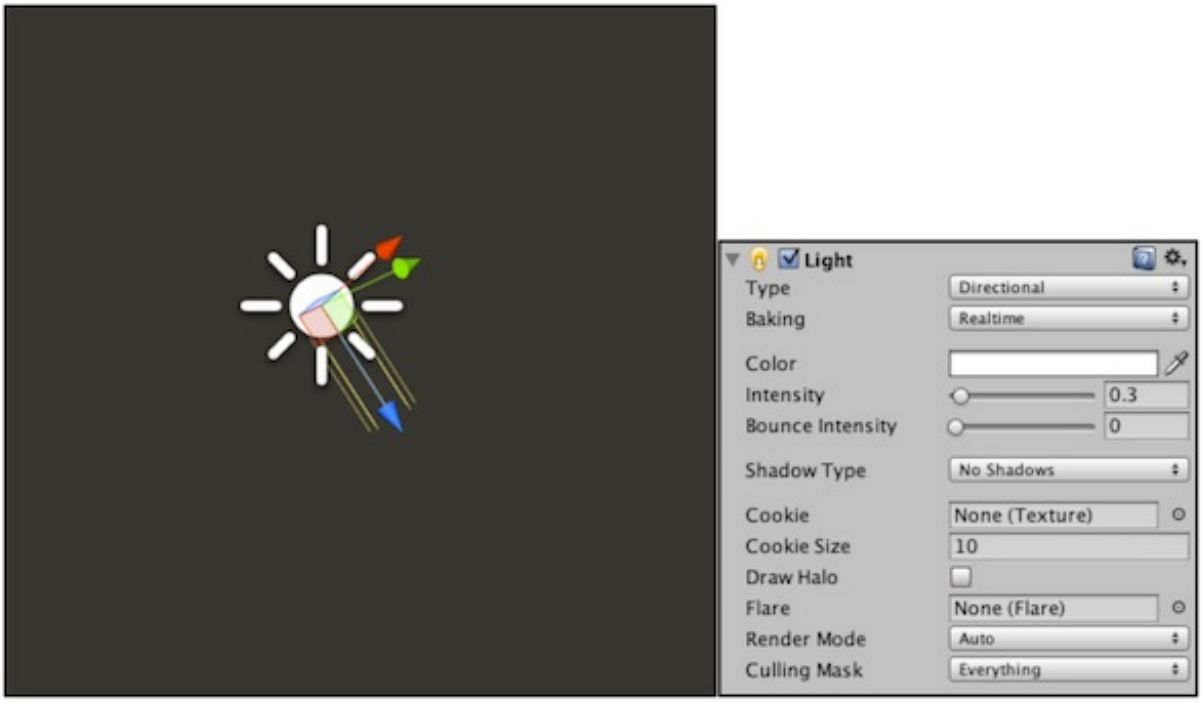


图9.5 平行光

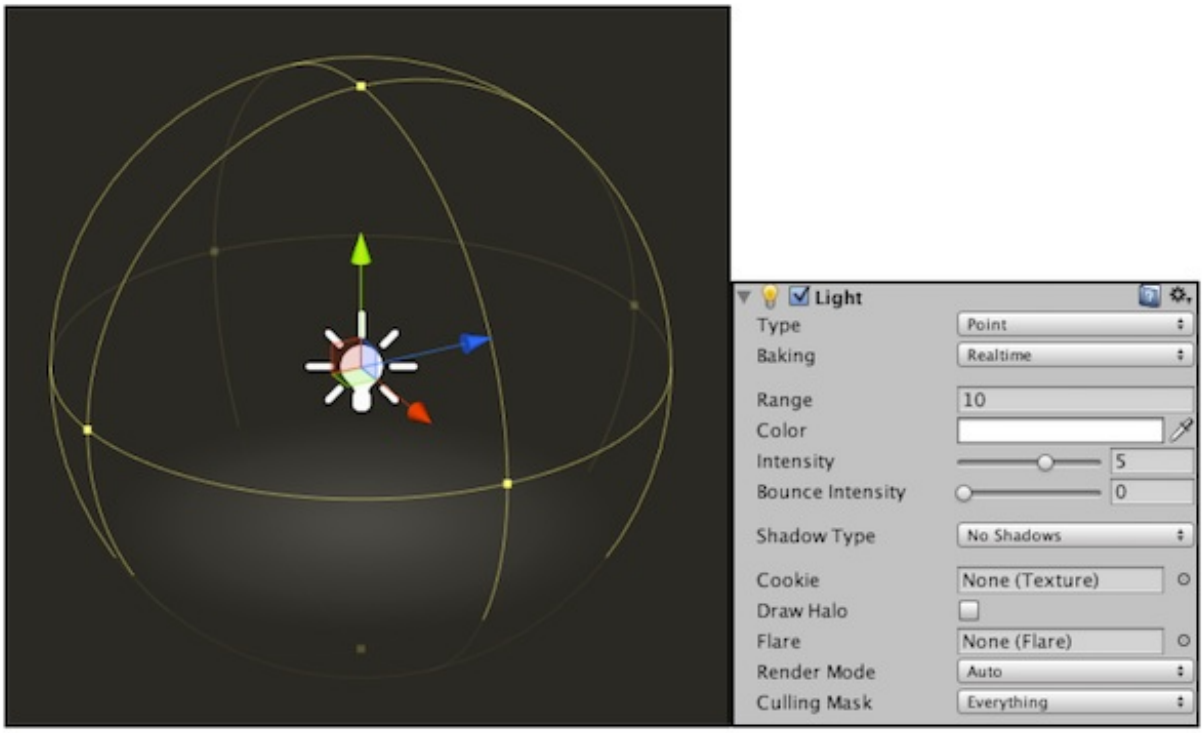


图9.6 点光源

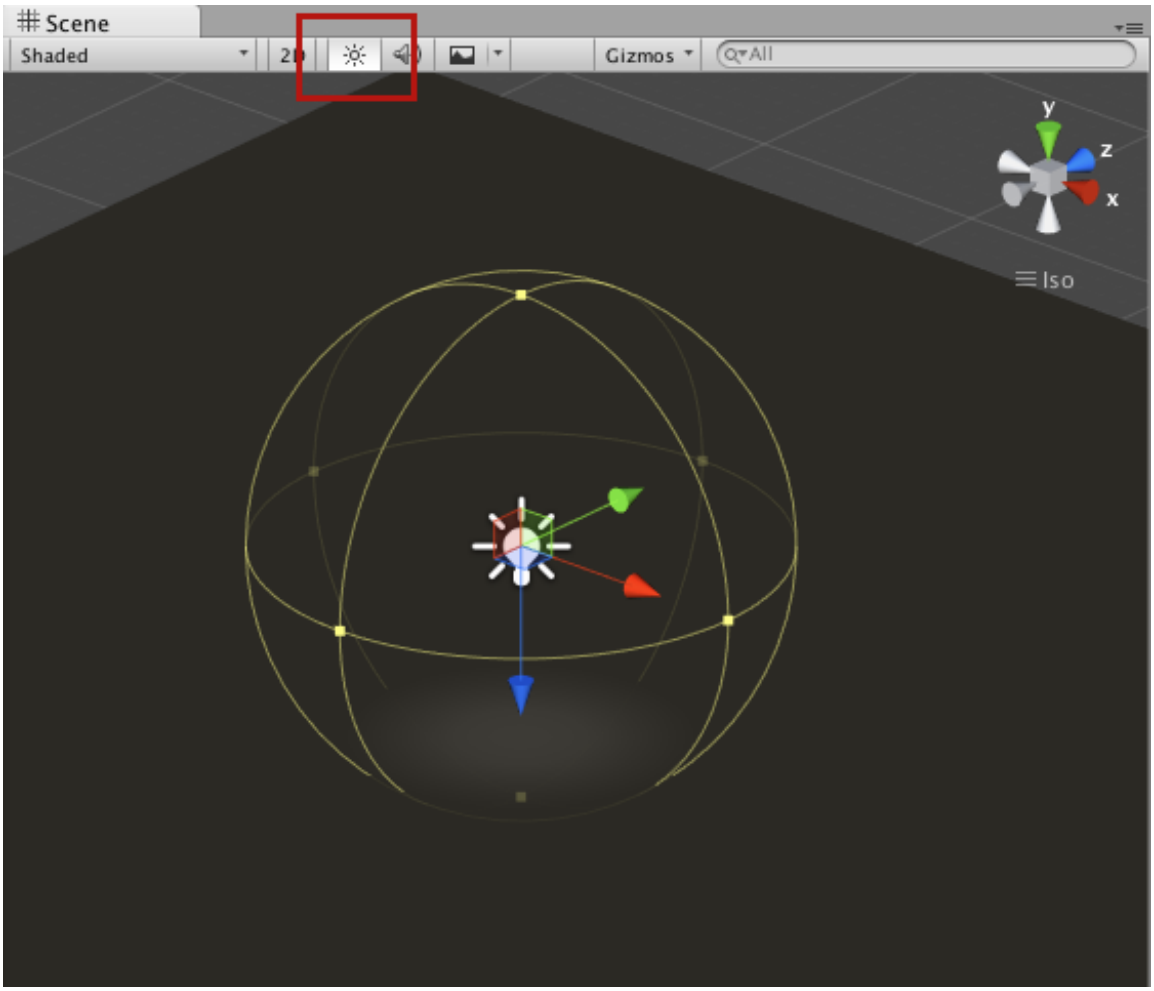


图9.7 开启Scene视图中的光照

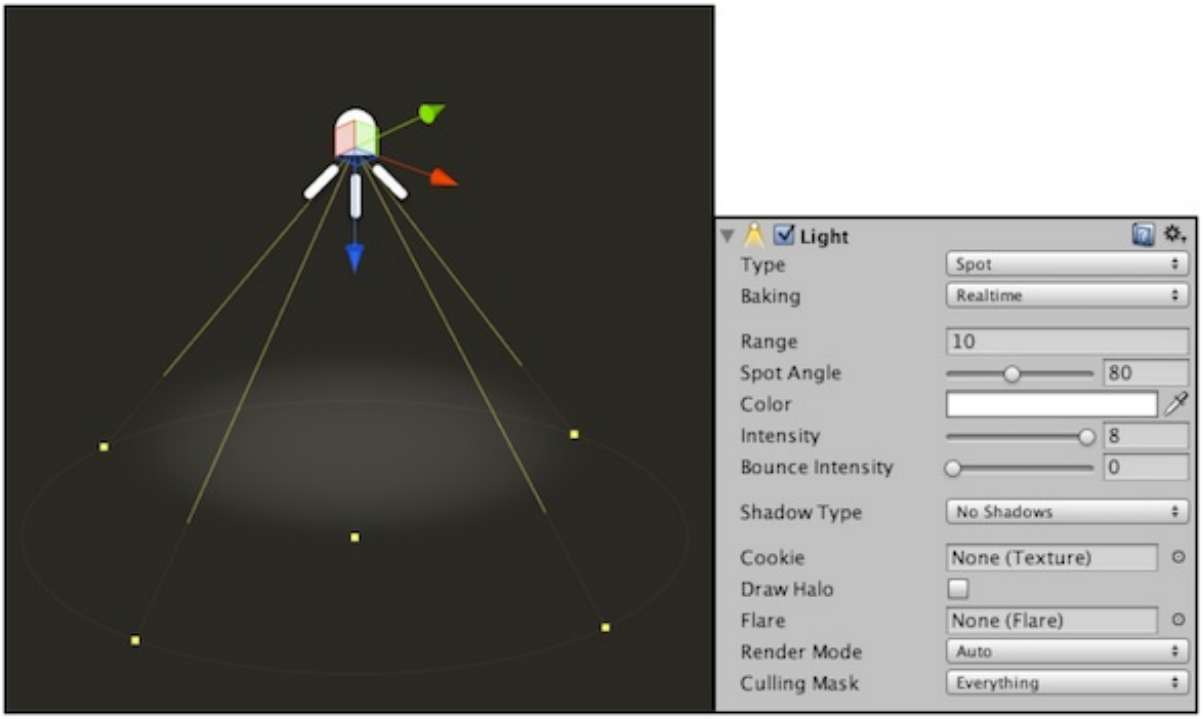


图9.8 聚光灯

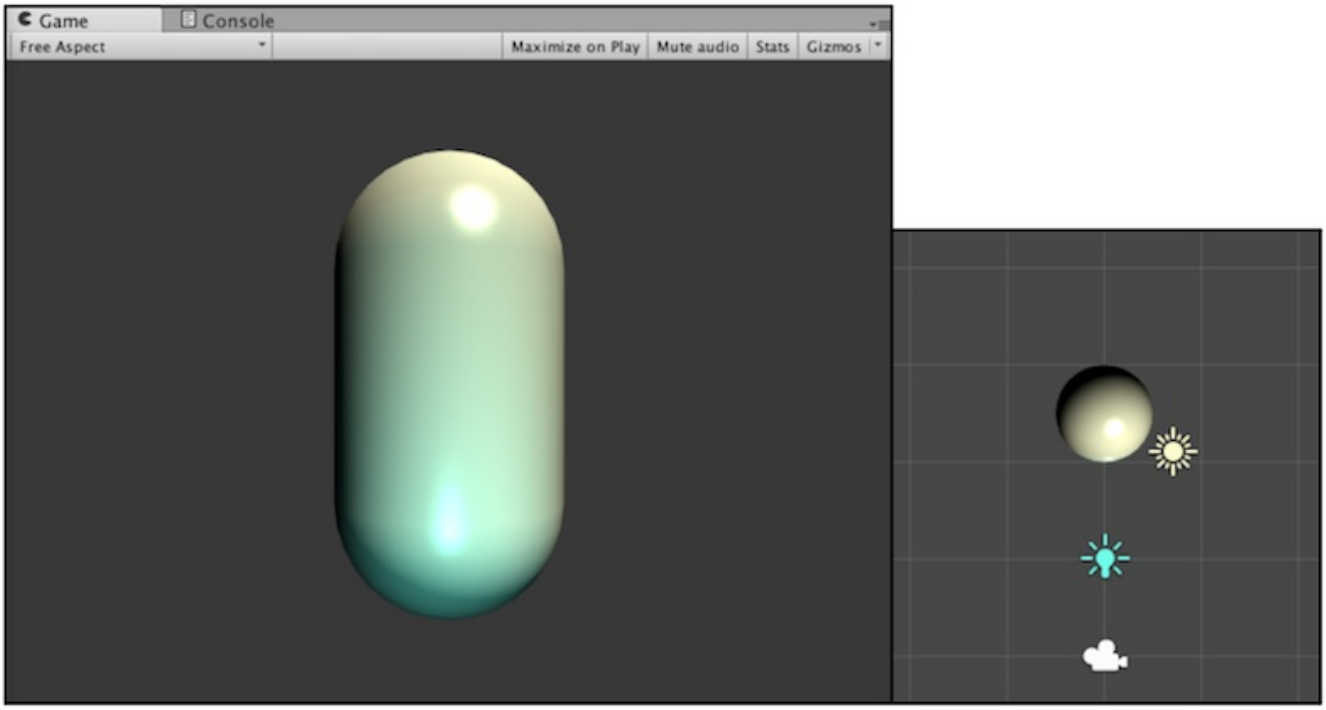


图9.9 使用一个平行光和一个点光源共同照亮物体。右图显示了胶囊体、平行光和点光源在场景中的相对位置

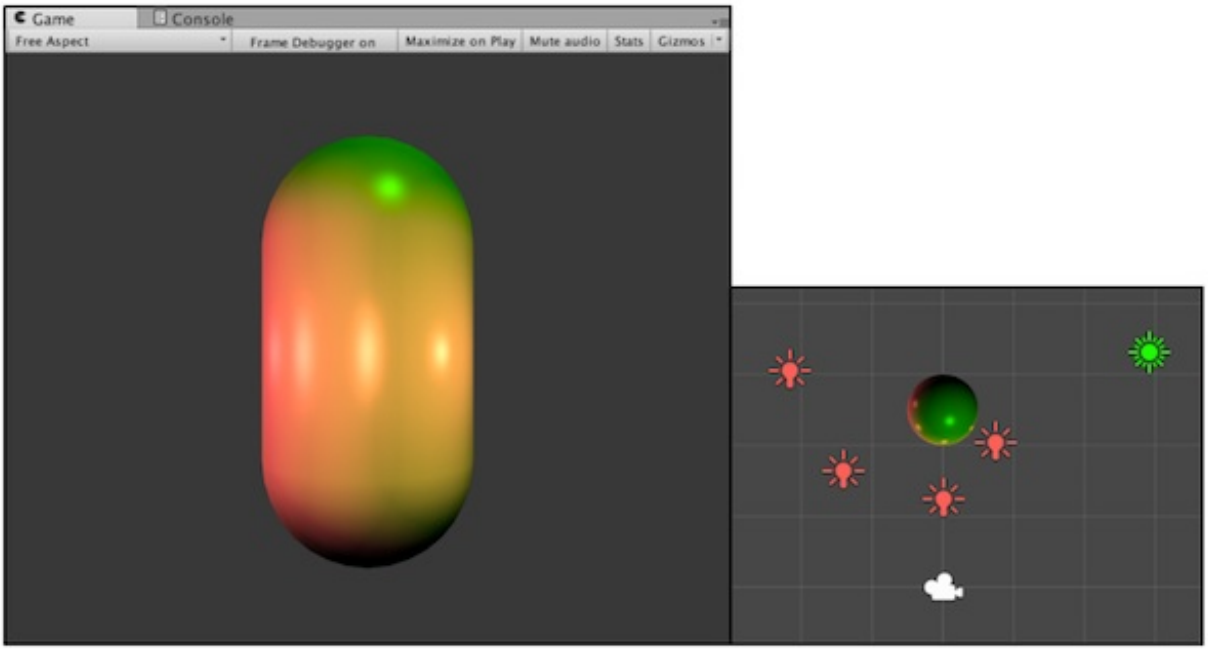


图9.10 使用1个平行光 + 4个点光源照亮一个物体

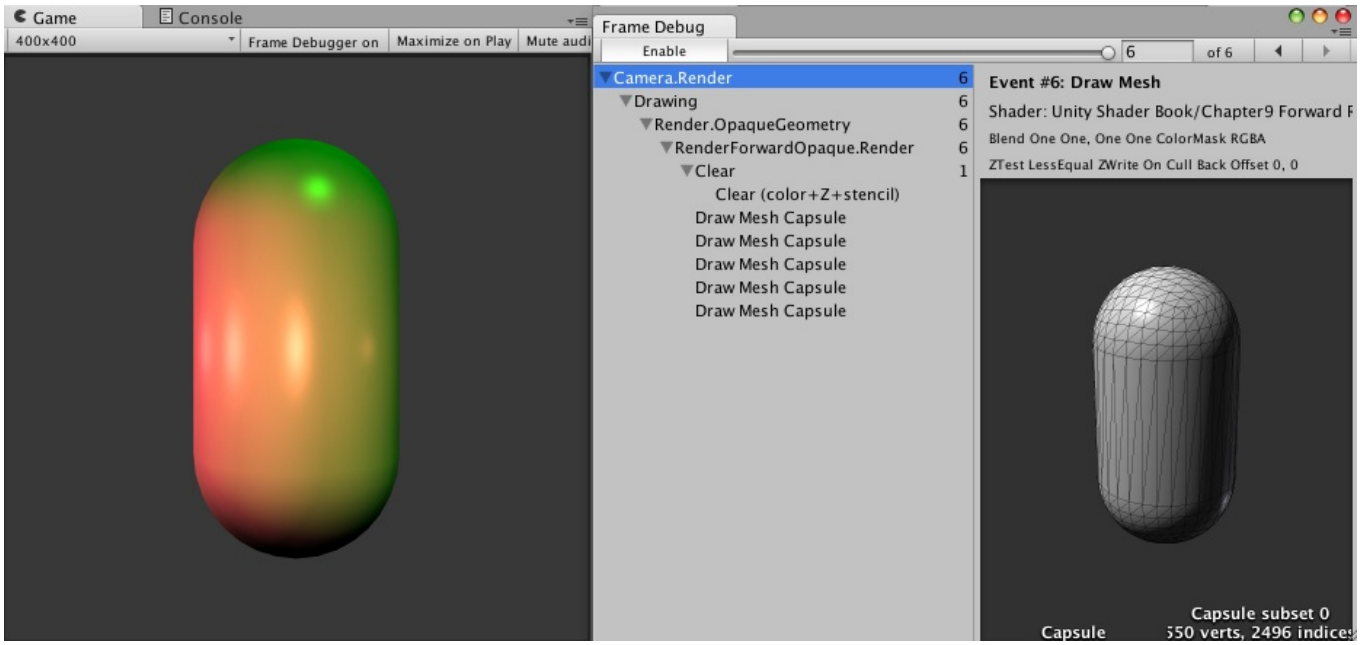


图9.11 打开帧调试器查看场景的绘制事件

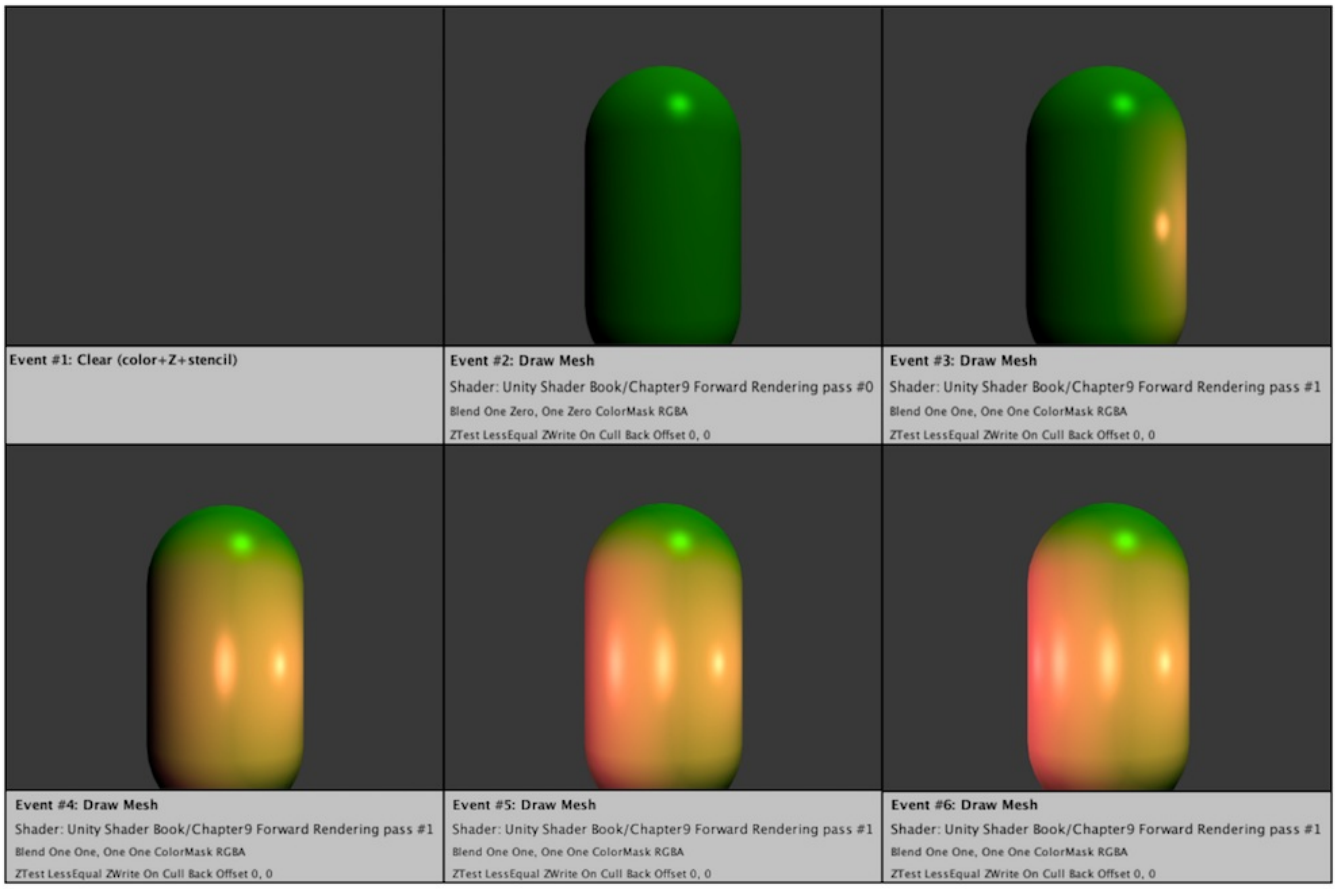


图9.12 本例中的6个渲染事件，绘制顺序是从左到右、从上到下进行的

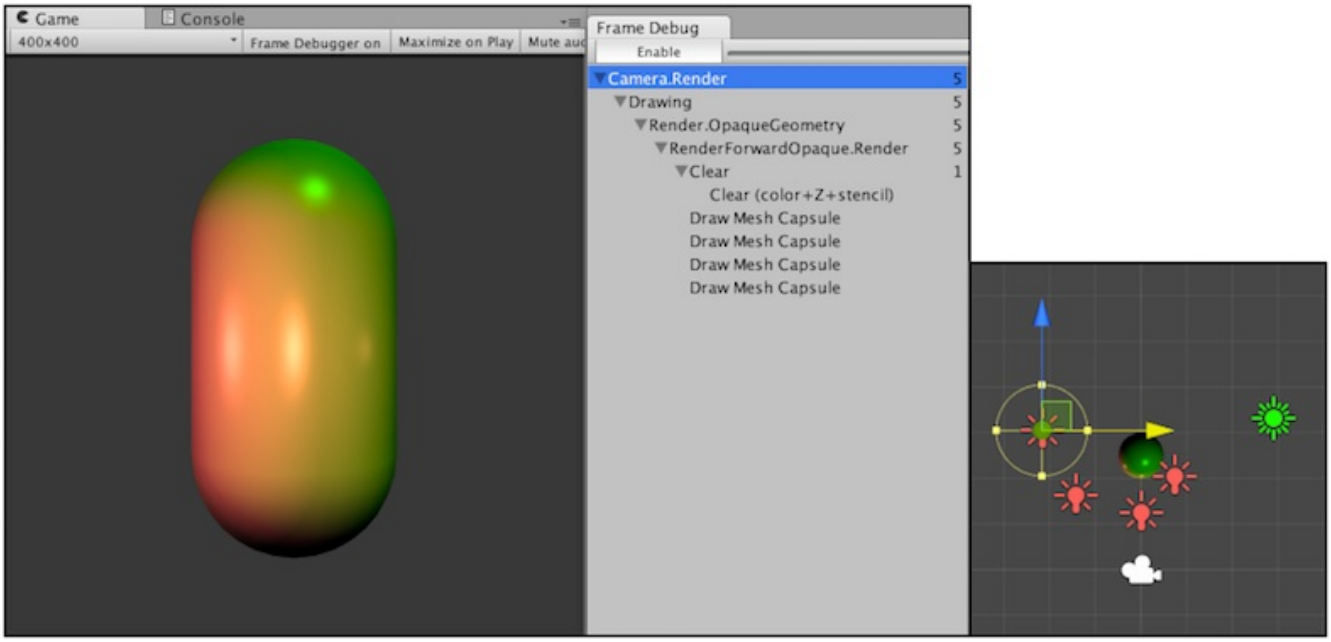


图9.13 如果物体不在一个光源的光照范围内（从右图可以看出，胶囊体不在最左方的点光源的照明范围内），Unity是不会调用Additional Pass来为该物体处理该光源的

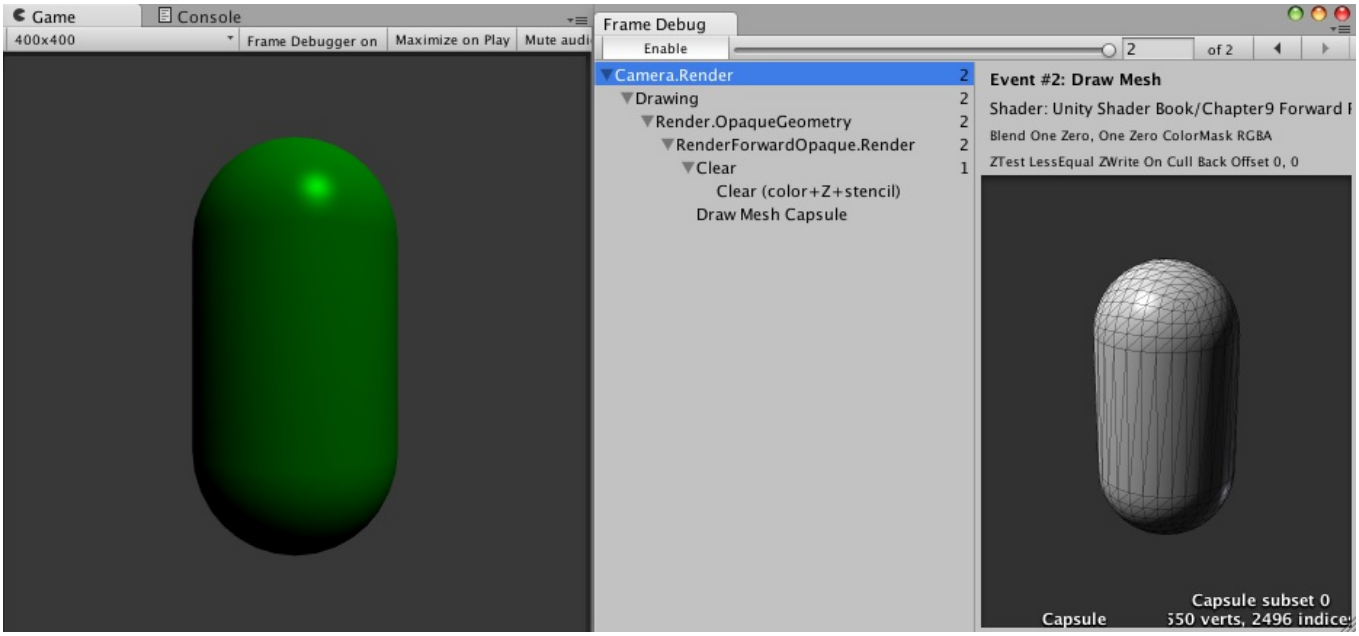


图9.14 当把光源的Render Mode设为Not Important时，这些光源就不会按逐像素光来处理

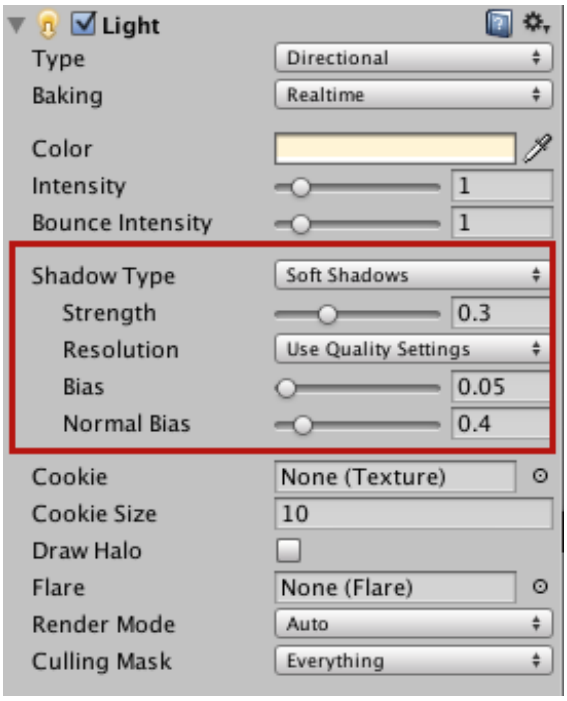


图9.15 开启光源的阴影效果

图9.16 Mesh Renderer组件的Cast Shadows和Receive Shadows属性可以控制该物体是否投射/接收阴影

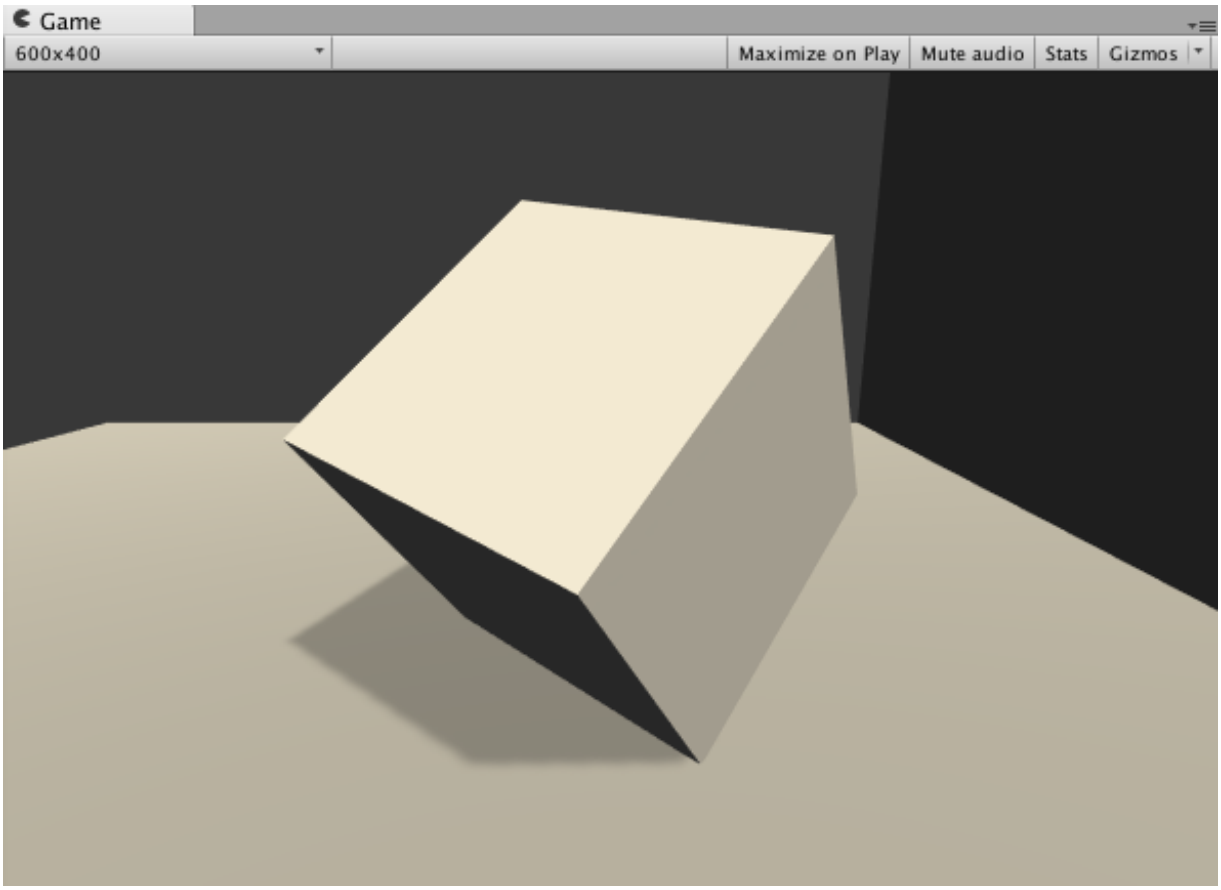


图9.17 开启Cast Shadows和Receive Shadows，从而让正方体可以投射和接收阴影

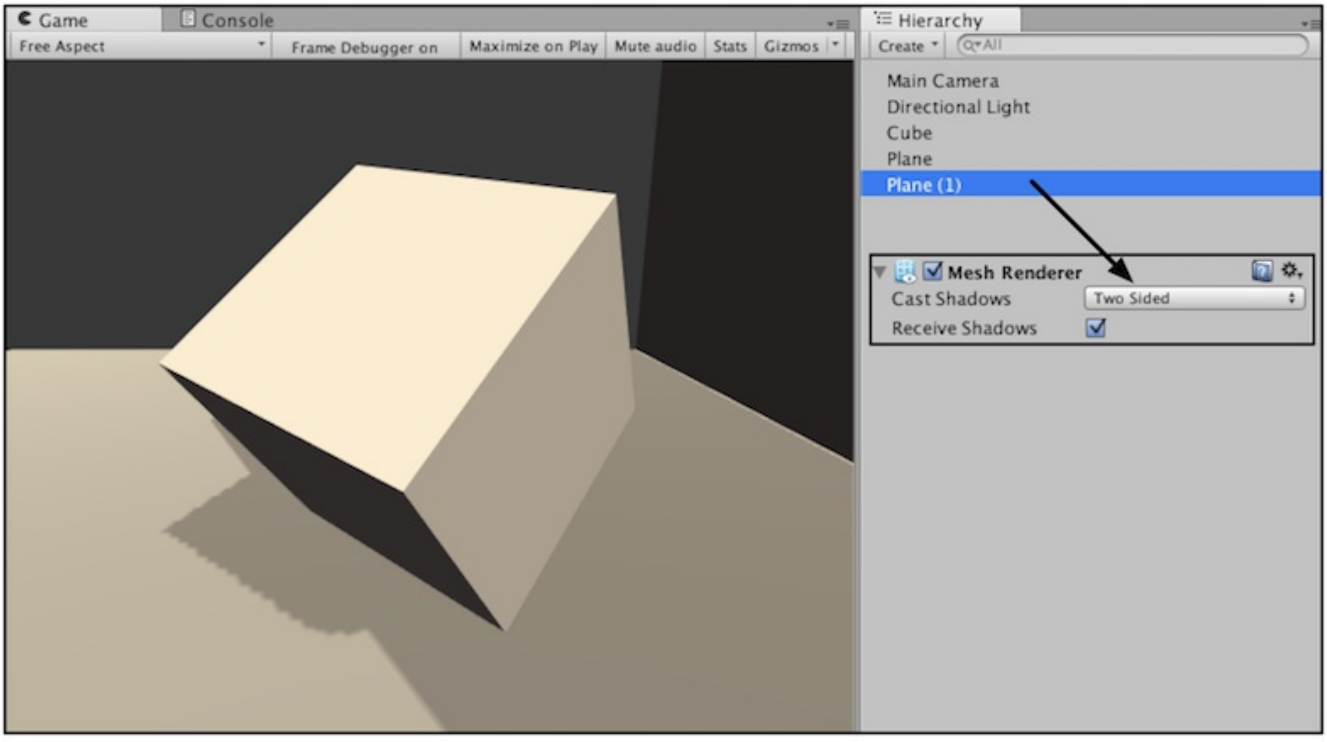


图9.18 把Cast Shadows设置为Two Sided可以让右侧平面的背光面也产生阴影

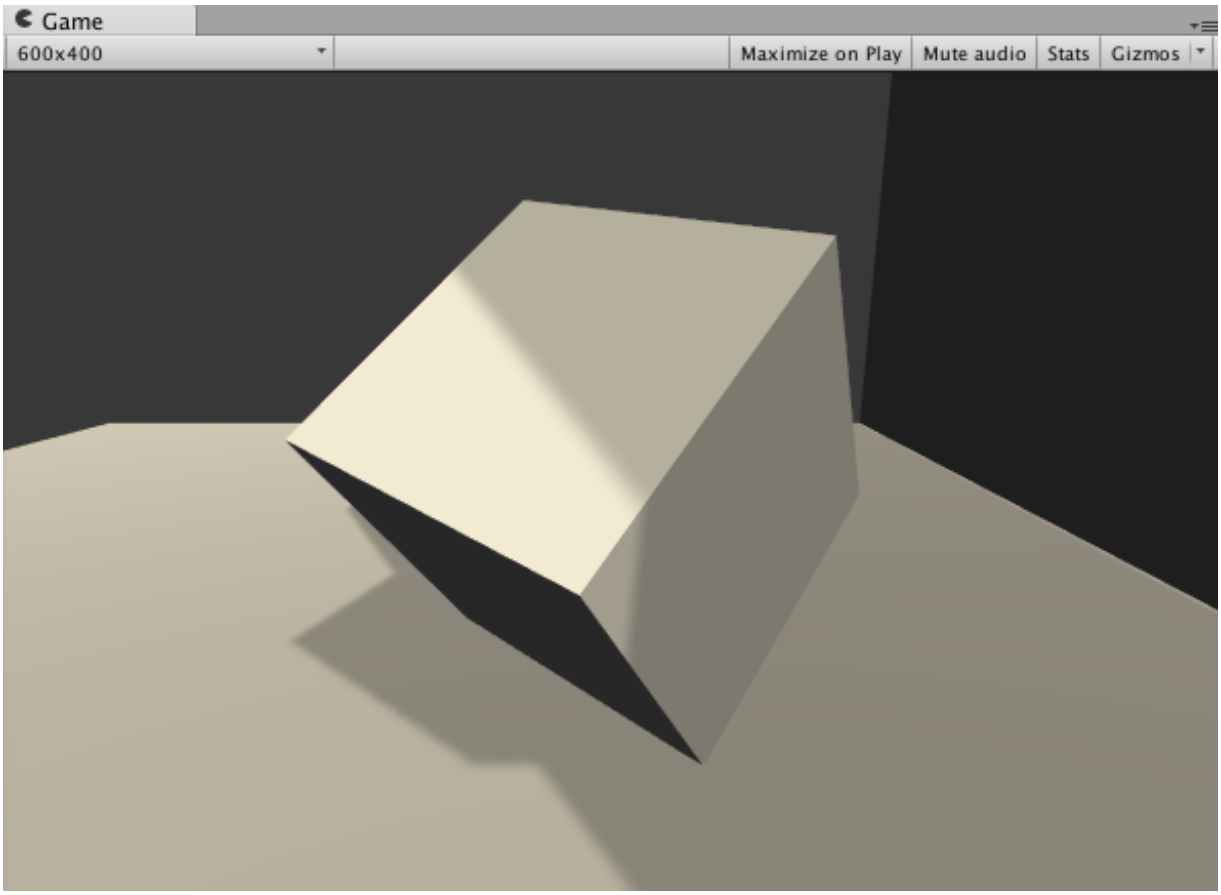


图9.19 正方体可以接收来自右侧平面的阴影

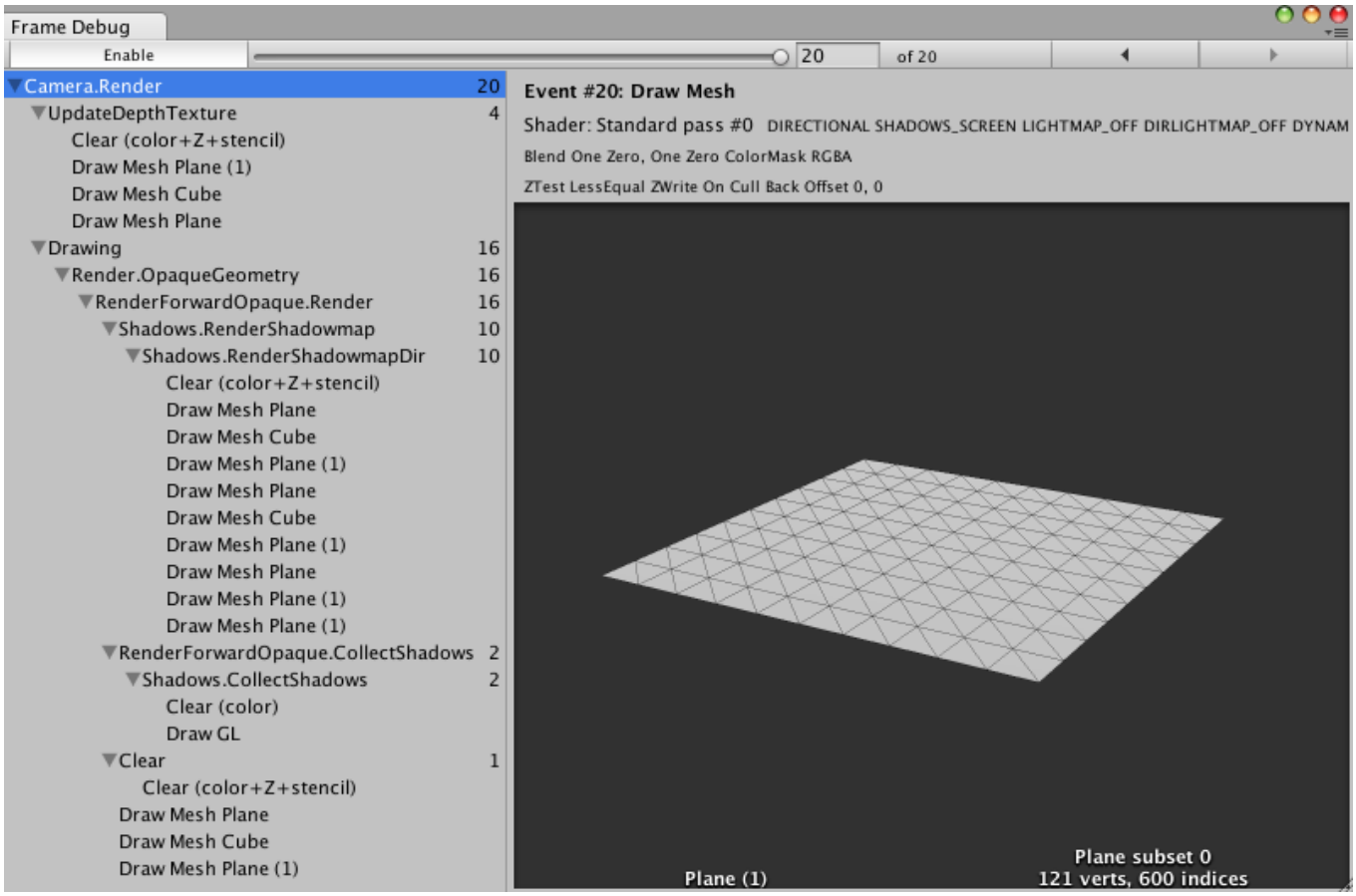


图9.20 使用帧调试器查看阴影绘制过程

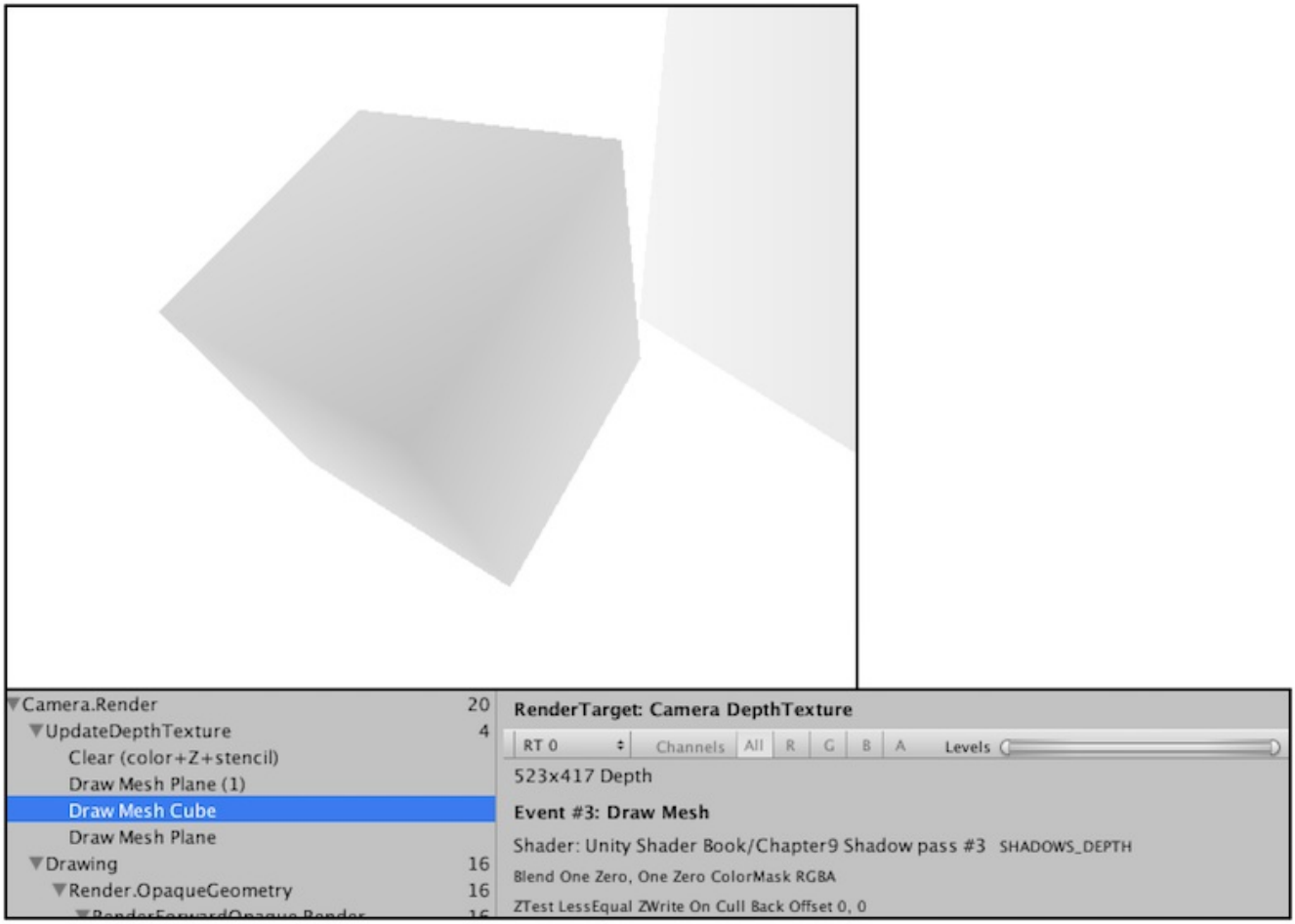


图9.21 正方体对深度纹理的更新结果

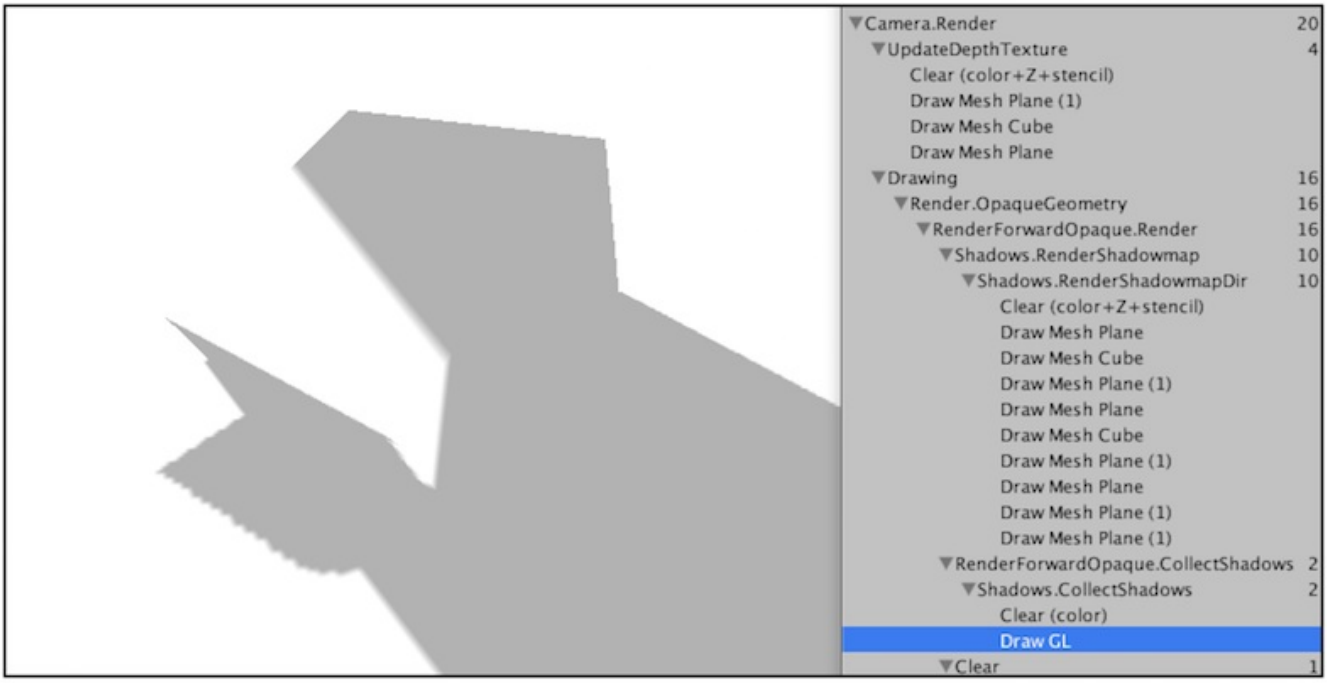


图9.22 屏幕空间的阴影图

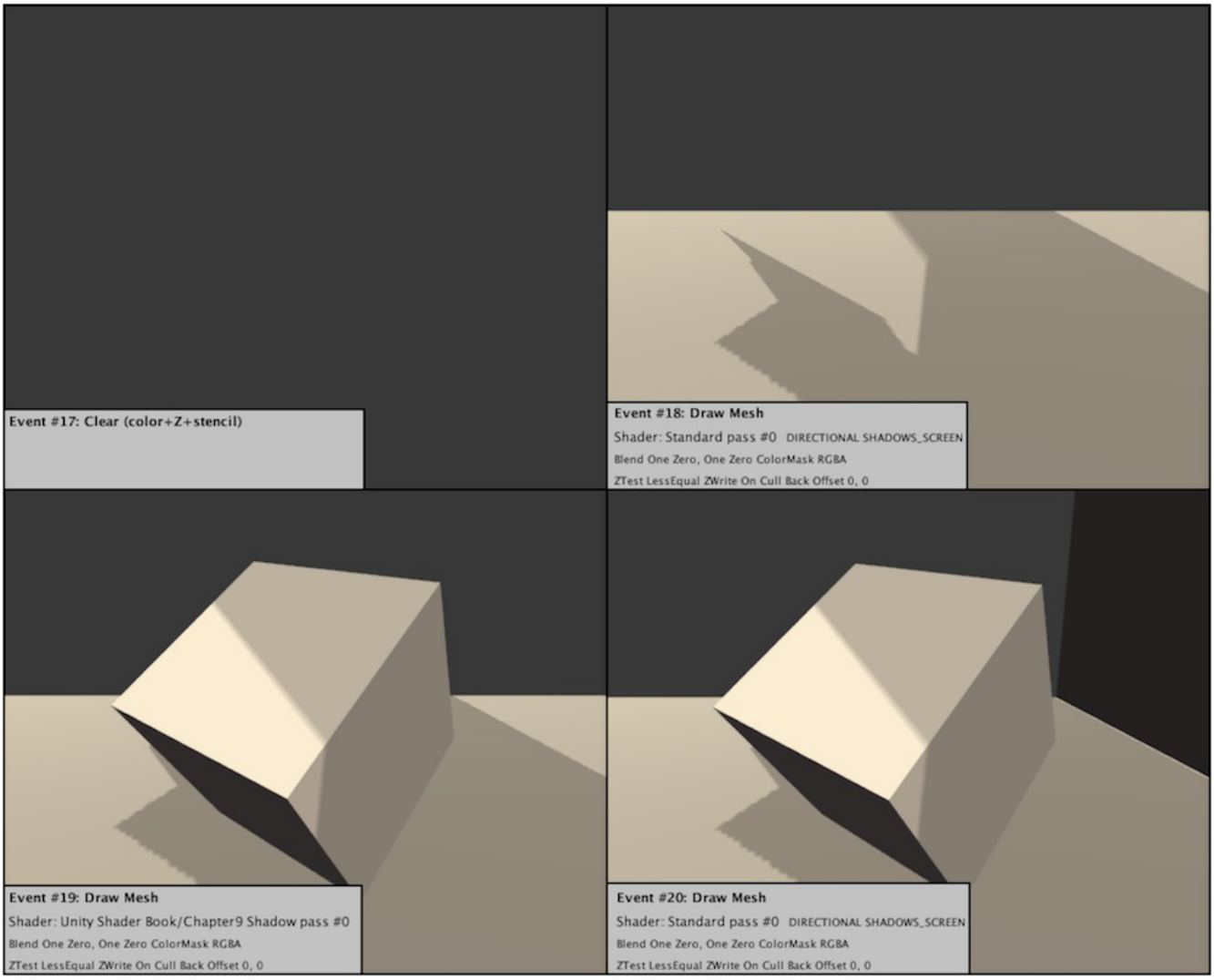


图9.23 Unity绘制屏幕阴影的过程

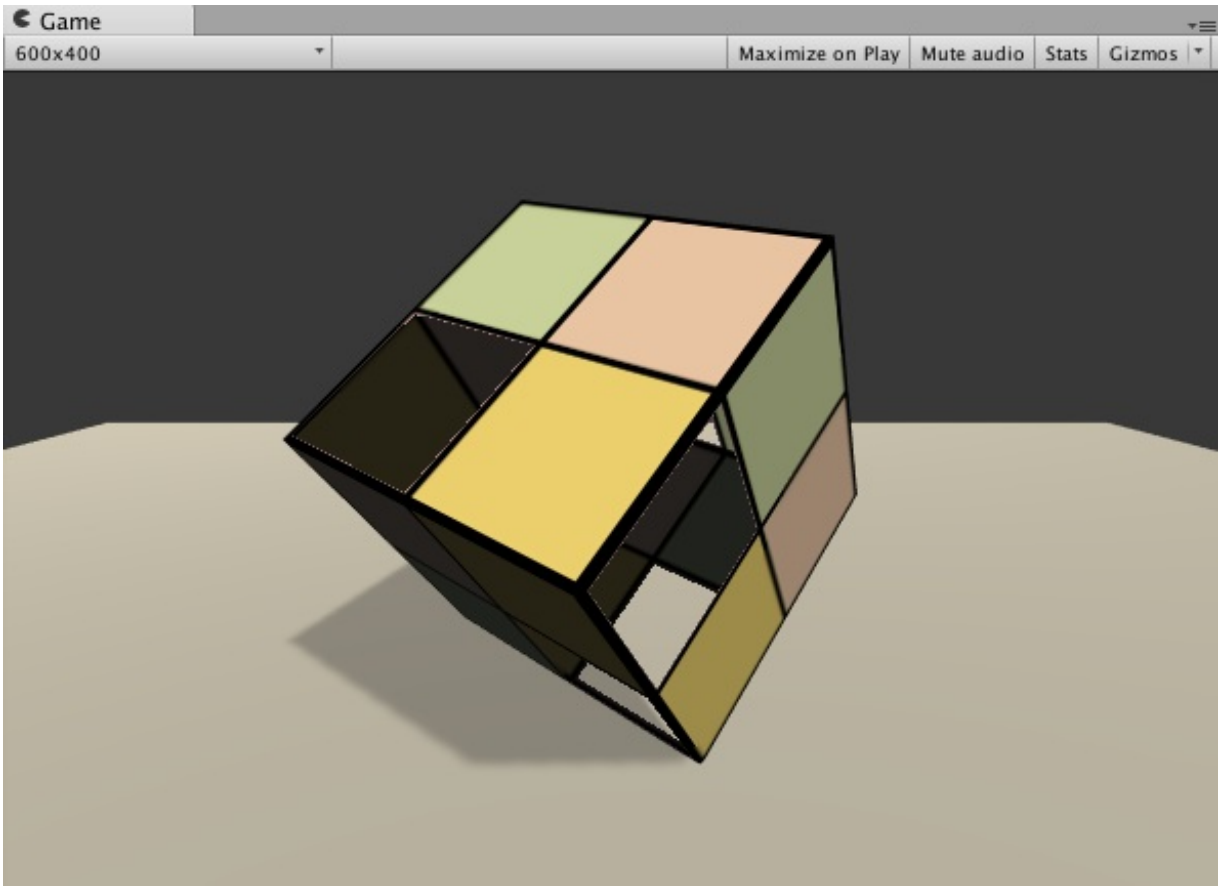


图9.24 可以投射阴影的使用透明度测试的物体

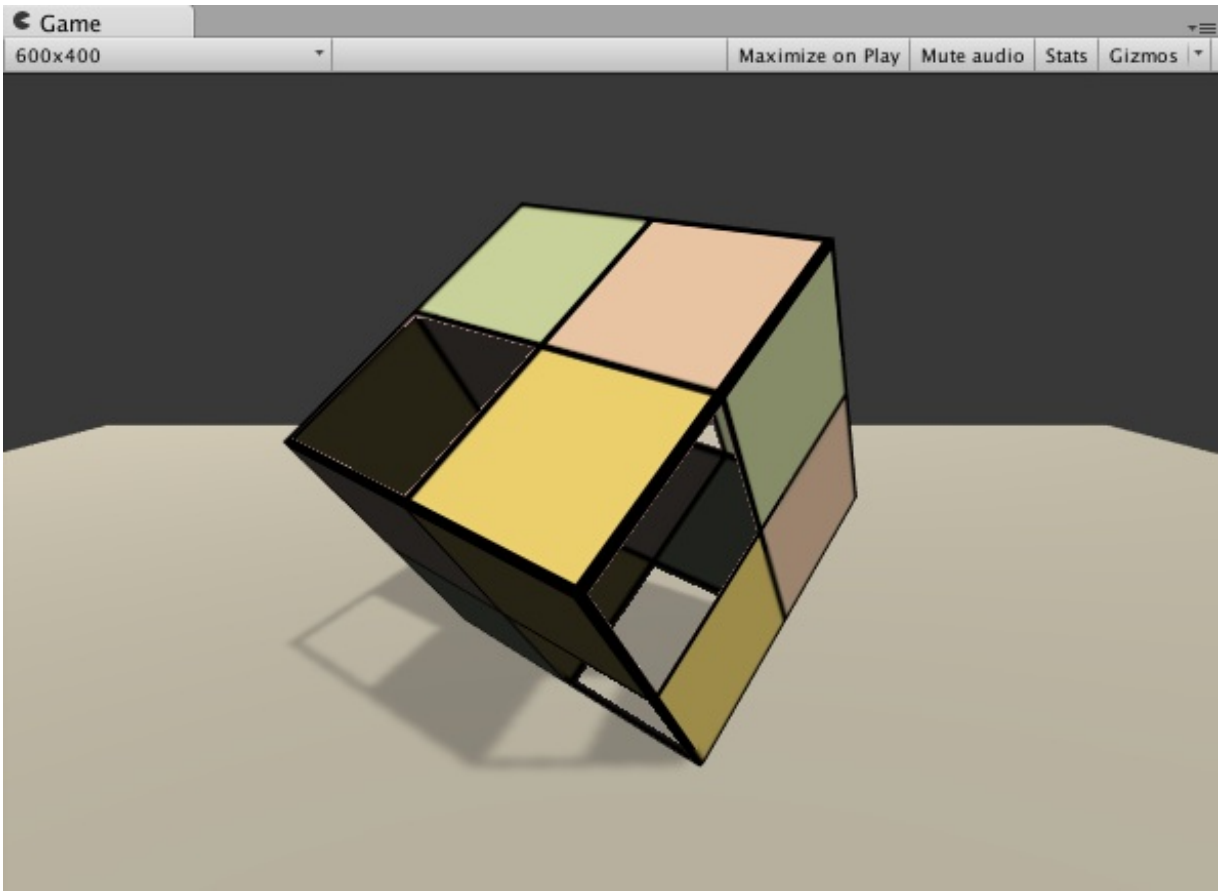


图9.25 正确设置了Fallback的使用透明度测试的物体

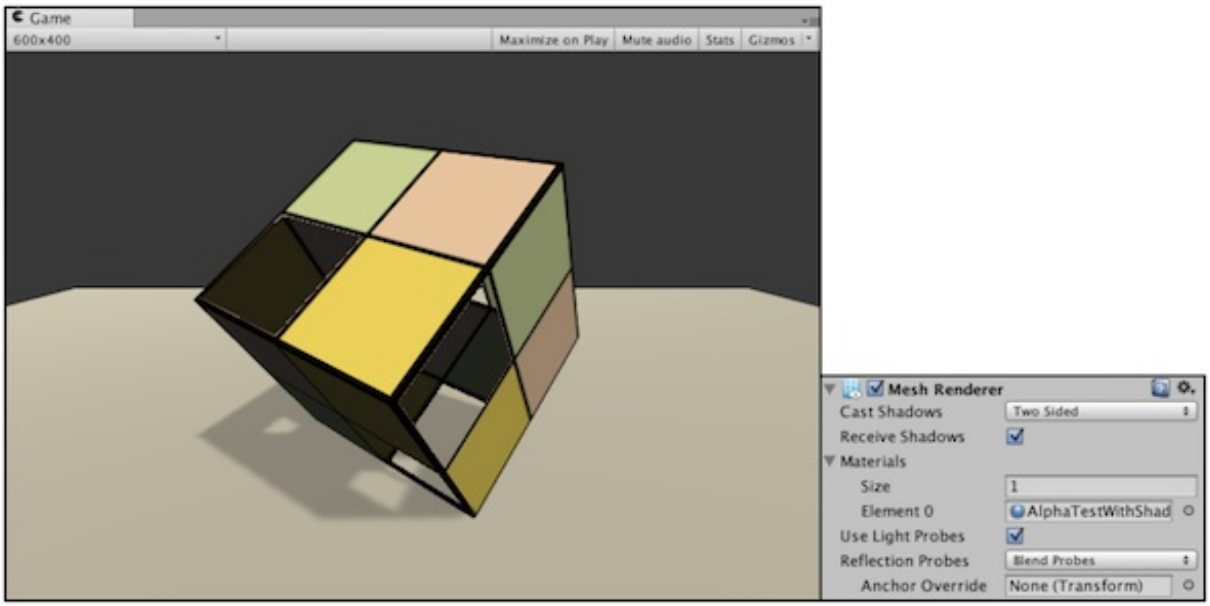


图9.26 正确设置了Cast Shadow属性的使用透明度测试的物体

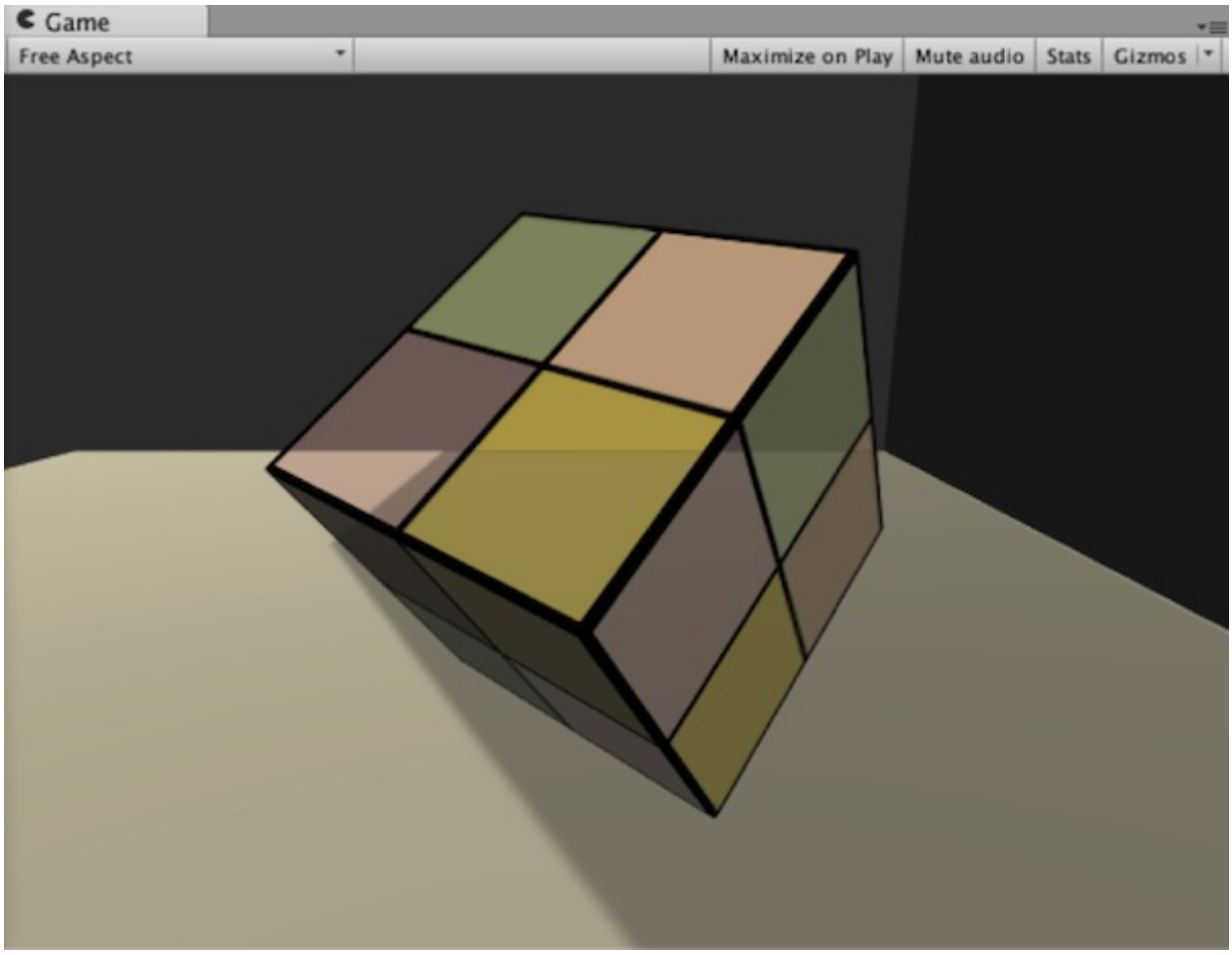


图9.27 把使用了透明度混合的Unity Shader的Fallback设置为内置的Transparent/VertexLit。半透明物体不会向下方的平面投射阴影，也不会接收来自右侧平面的阴影，它看起来就像是完全透明一样

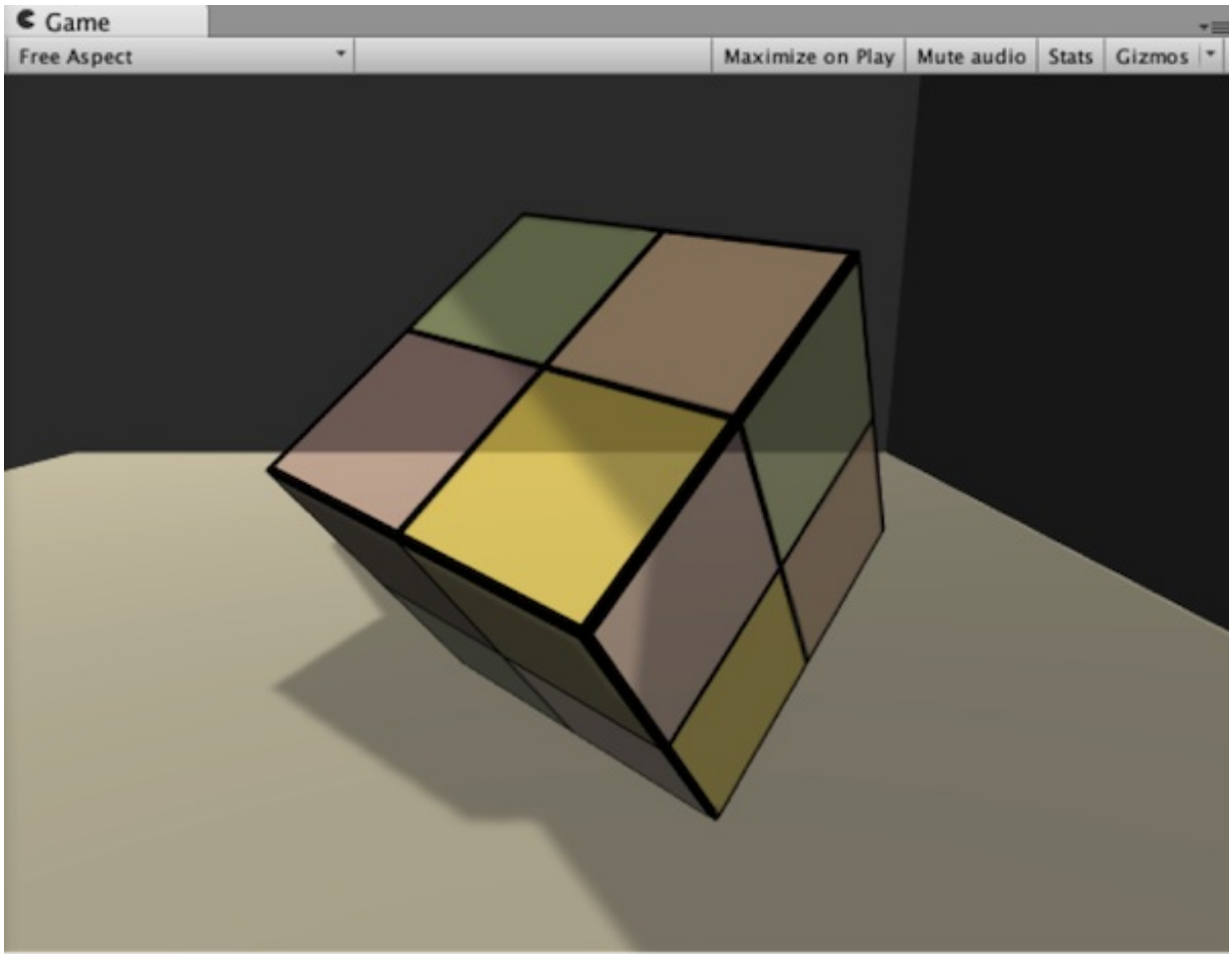


图9.28 把Fallback设为VertexLit来强制为半透明物体生成阴影

第10章 高级纹理

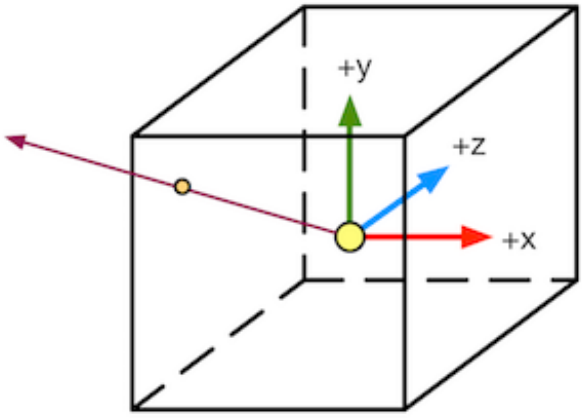


图10.1 对立方体纹理的采样


SkyboxMat
Shader
Skybox/6 Sided
Tint Color
Exposure
1
Rotation
0
Front [+Z] (HDR)
Select
Back [-Z] (HDR)
Select
Left [+X] (HDR)
Select
Right [-X] (HDR)
Select
Up [+Y] (HDR)
Select
Down [-Y] (HDR)
Select
SkyboxMat

图10.2 天空盒子材质

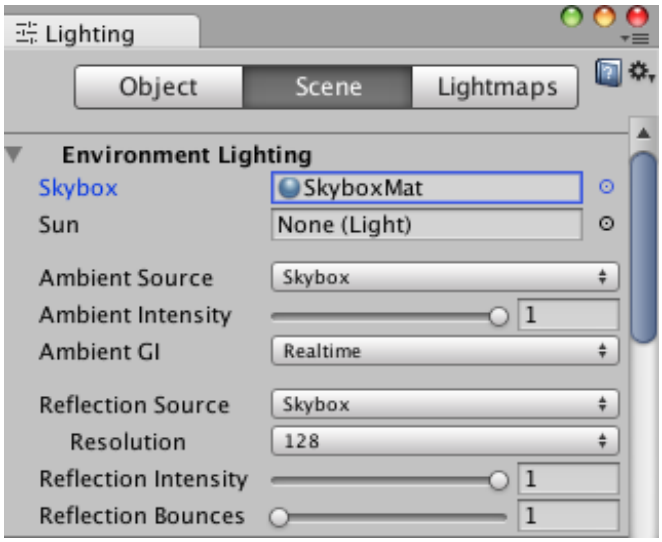


图10.3 为场景使用自定义的天空盒子

图10.4 使用了天空盒子的场景

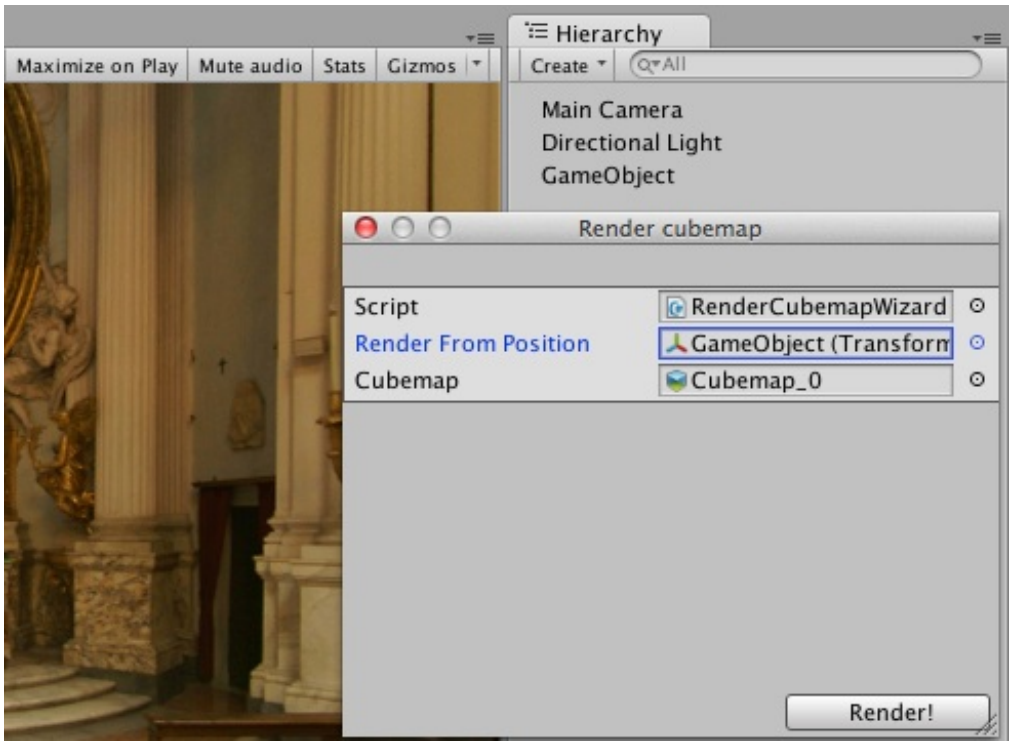


图10.5 使用脚本创建立方体纹理

图10.6 使用脚本渲染立方体纹理

图10.7 使用了反射效果的Teapot模型

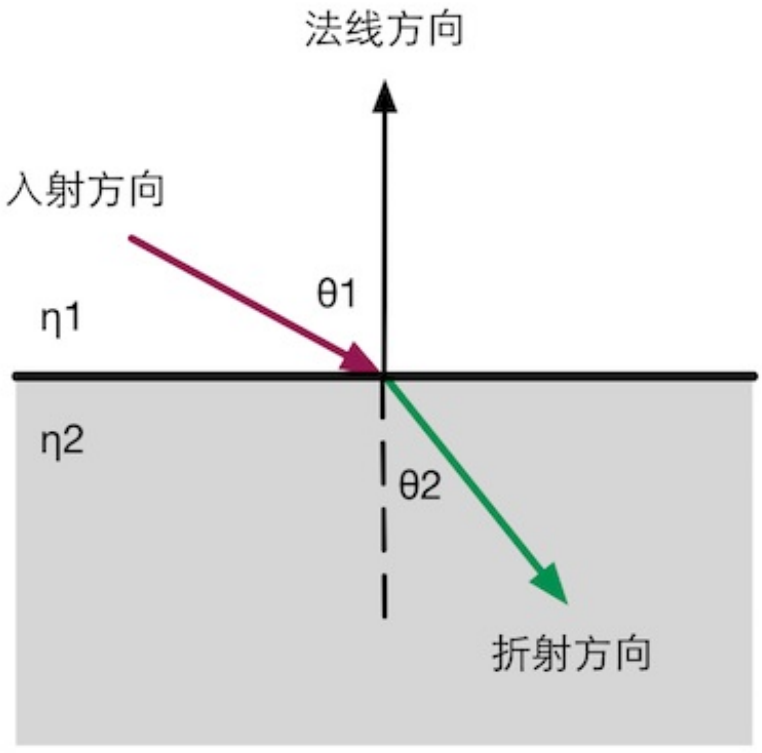


图10.8 斯涅尔定律

图10.9 使用了折射效果的Teapot模型

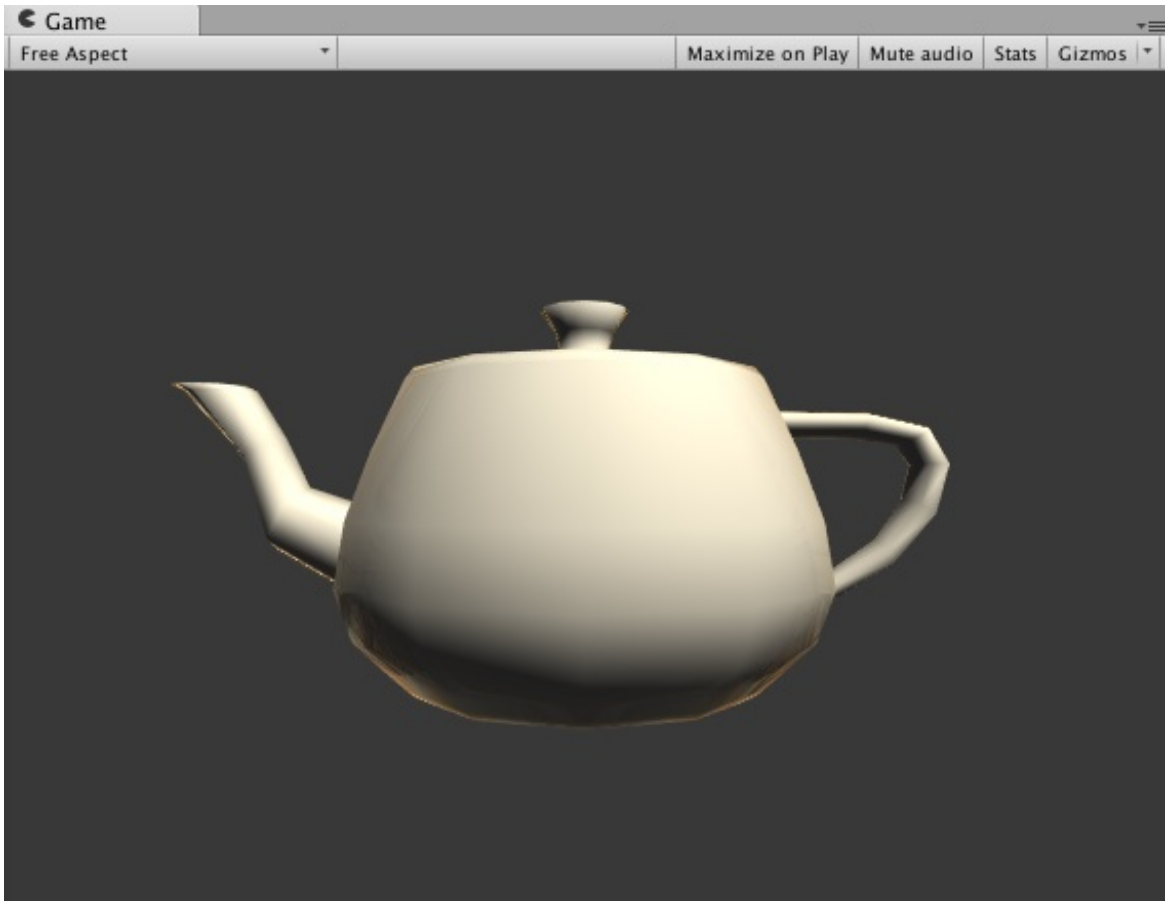


图10.10 使用了菲涅耳反射的Teapot模型

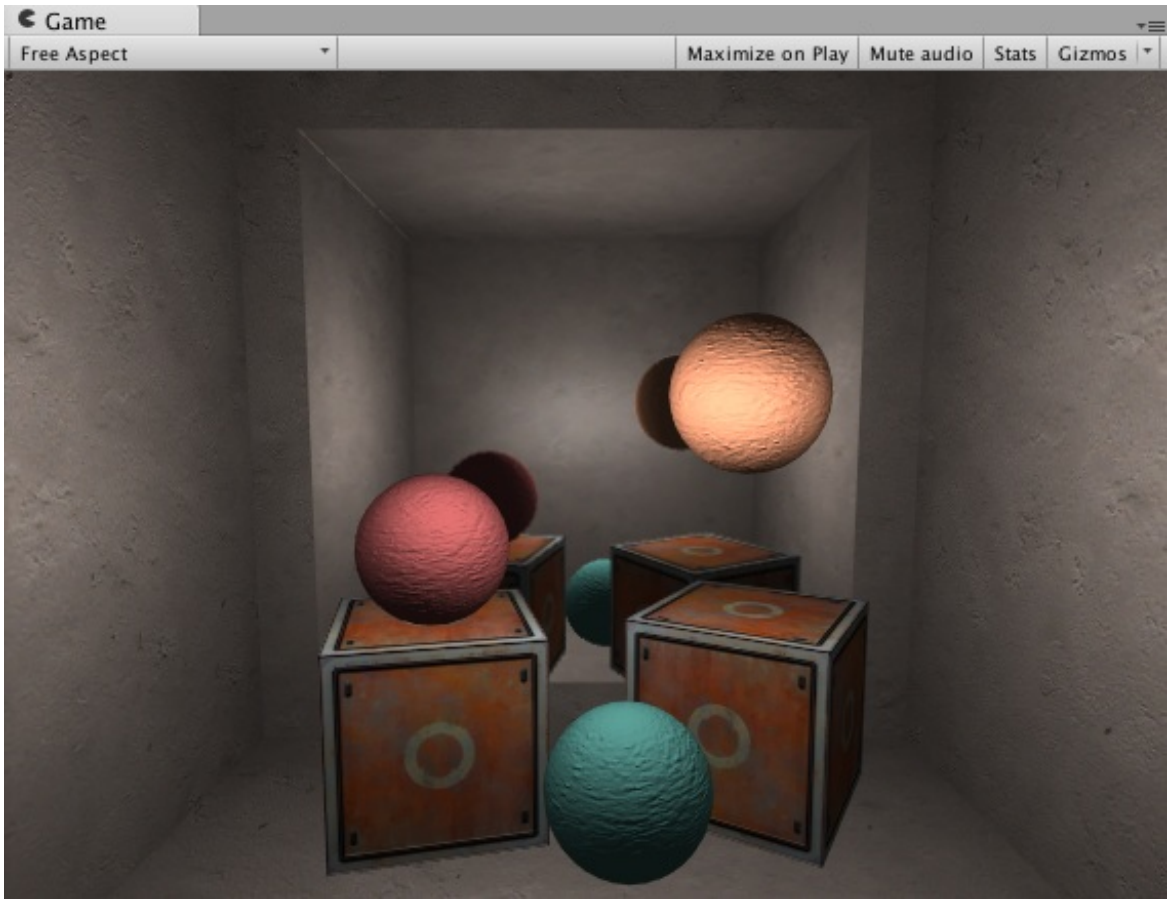


图10.11 镜子效果

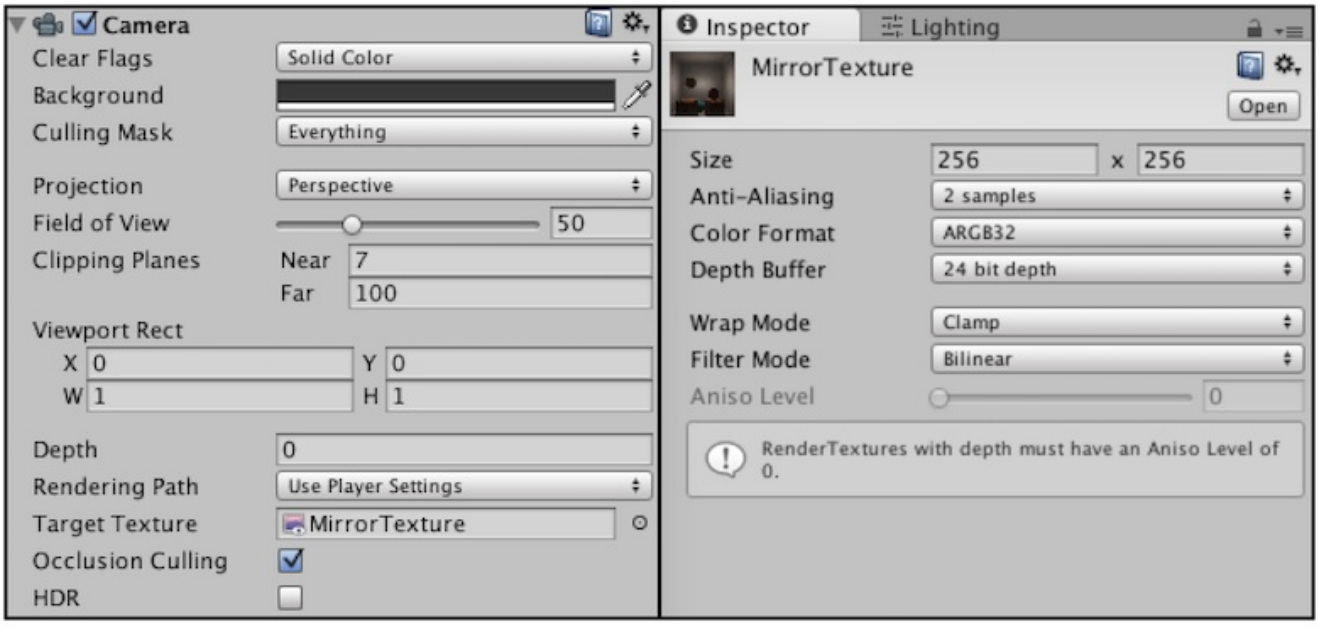


图10.12 左图：把摄像机的Target Texture设置成自定义的渲染纹理。右图：渲染纹理使用的纹理设置

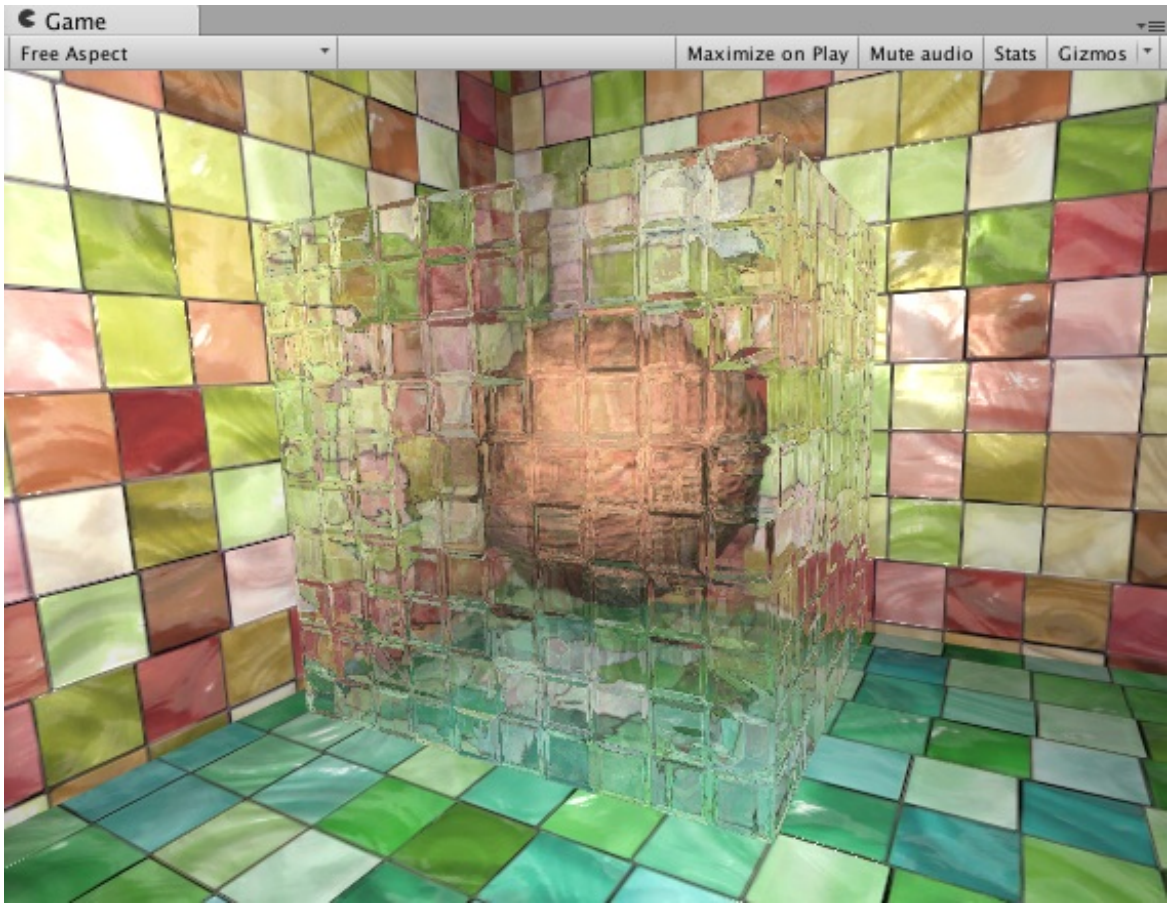


图10.13 玻璃效果

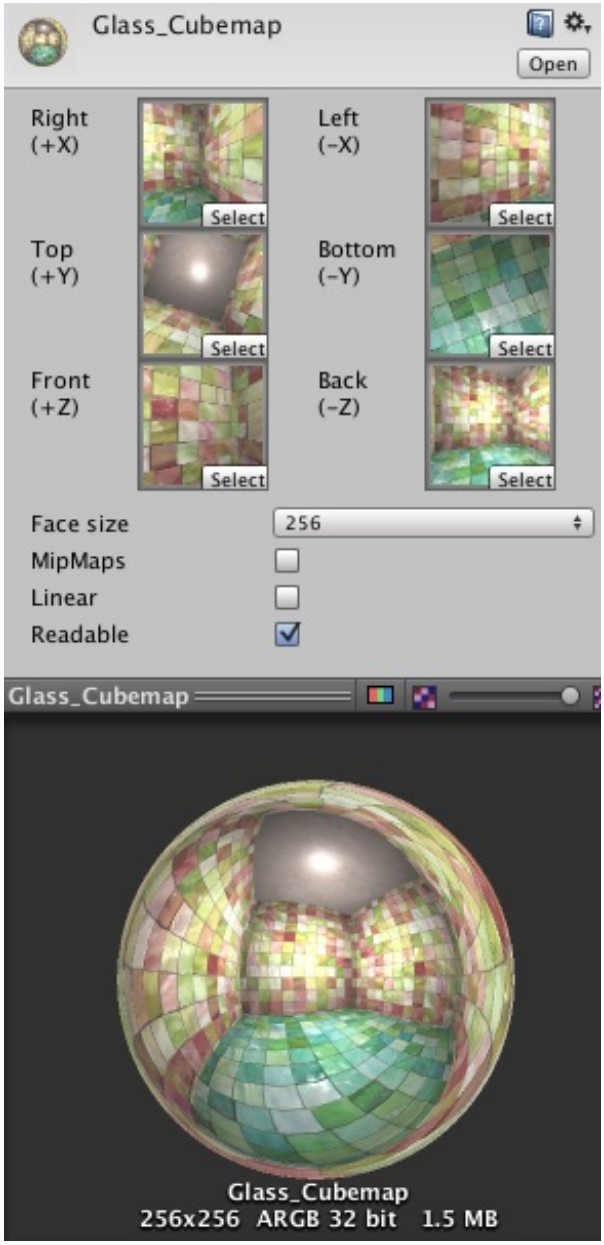


图10.14 本例使用的立方体纹理

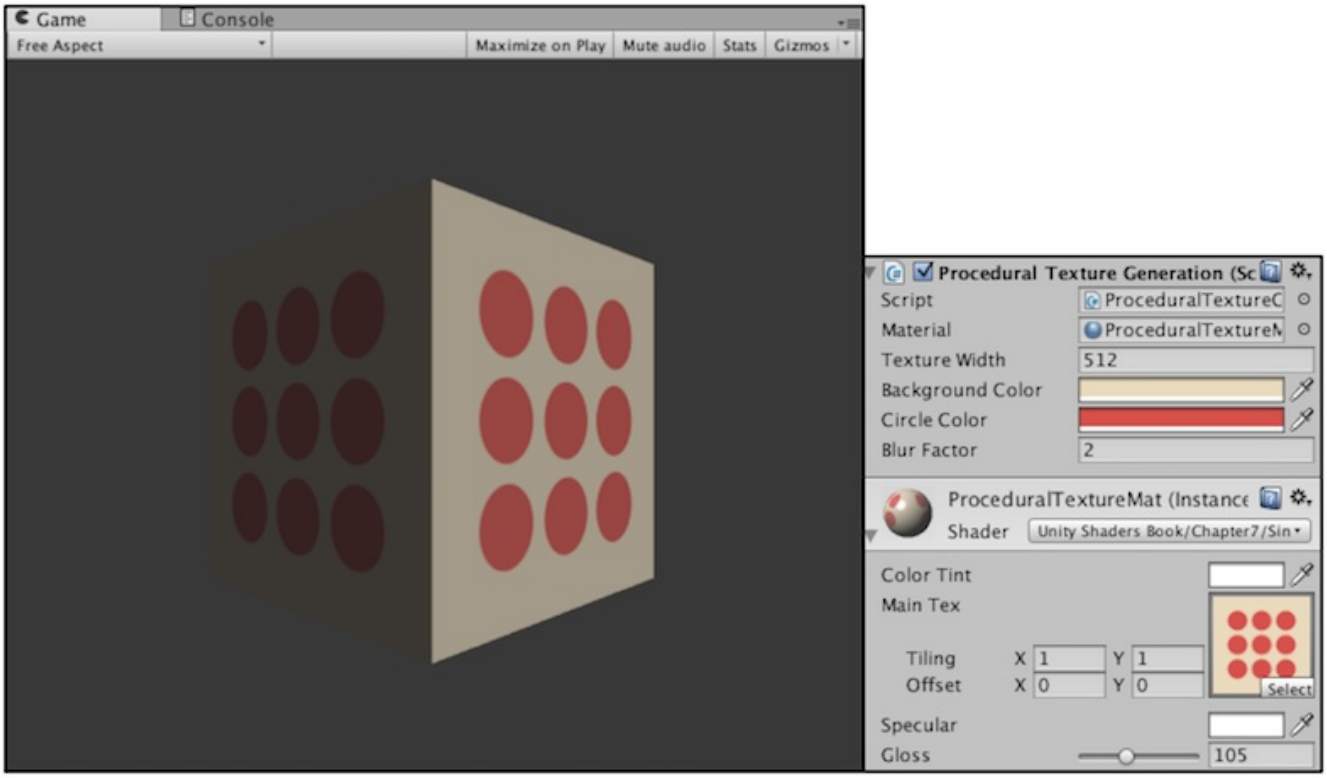


图10.15 脚本生成的程序纹理

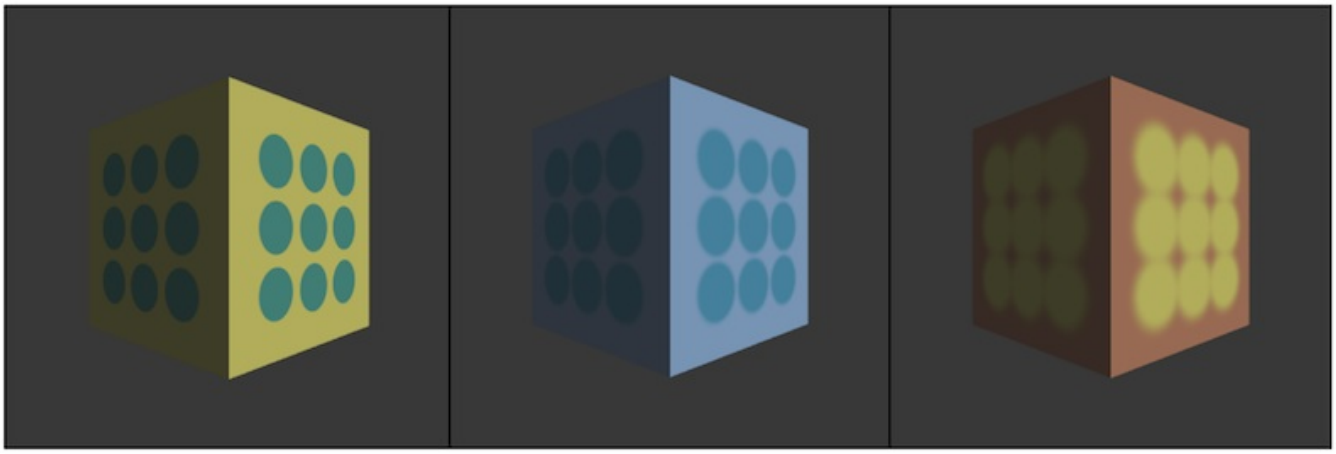


图10.16　调整程序纹理的参数来得到不同的程序纹理

图10.17　后缀为.sbsar的Substance材质

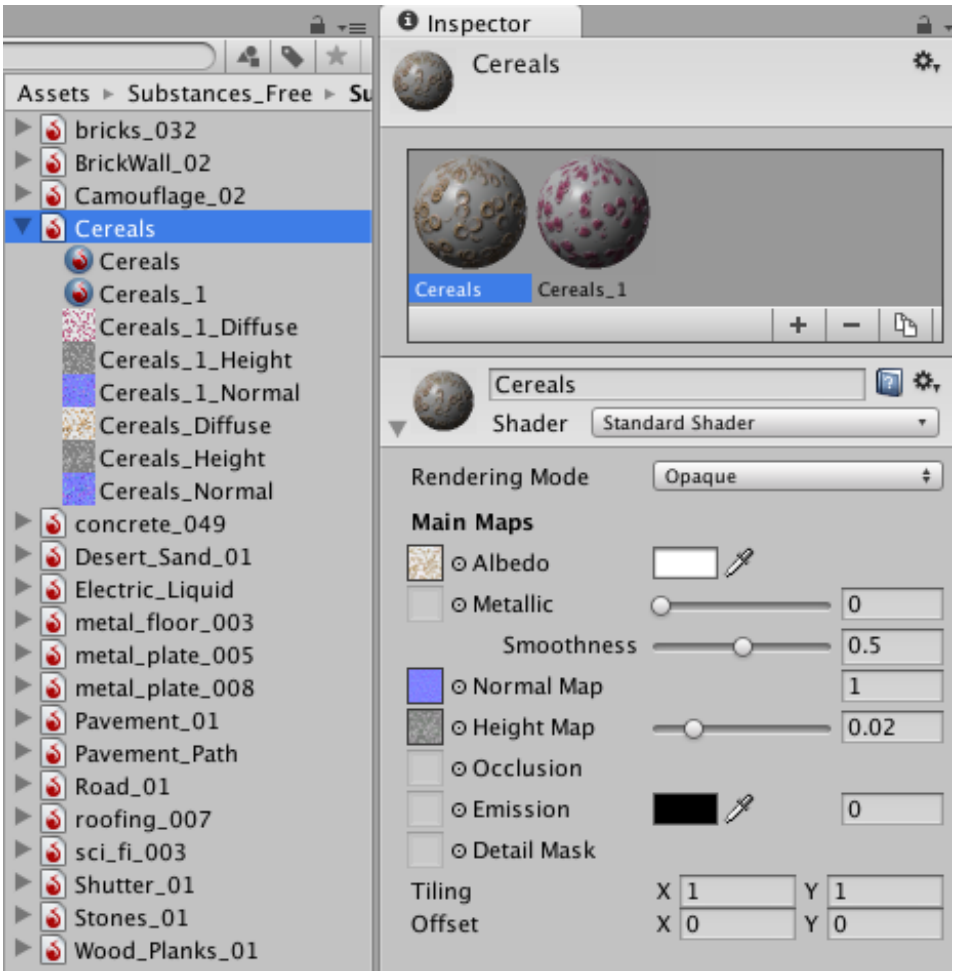


图10.18　程序纹理资源

图10.19　调整程序纹理属性可以得到看似完全不同的程序材质效果

第11章 让画面动起来

图11.1 本节使用的序列帧图像

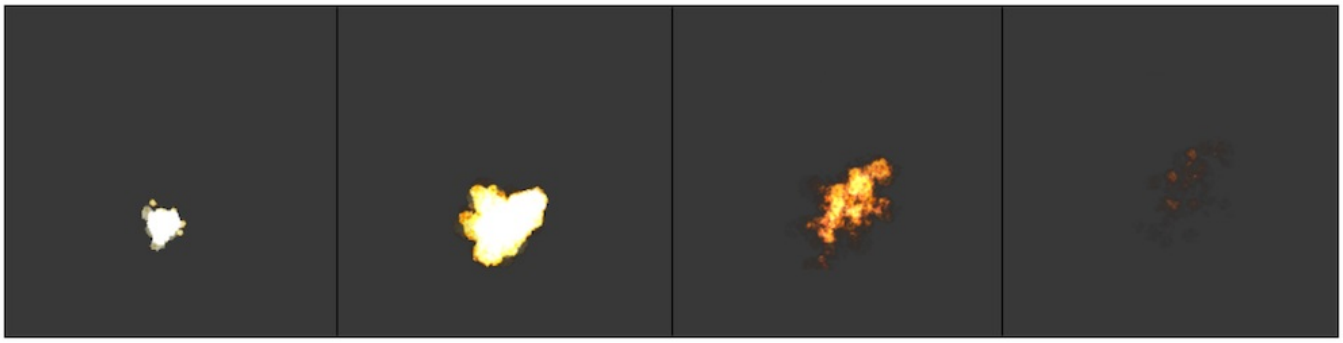


图11.2 使用序列帧动画来实现爆炸效果

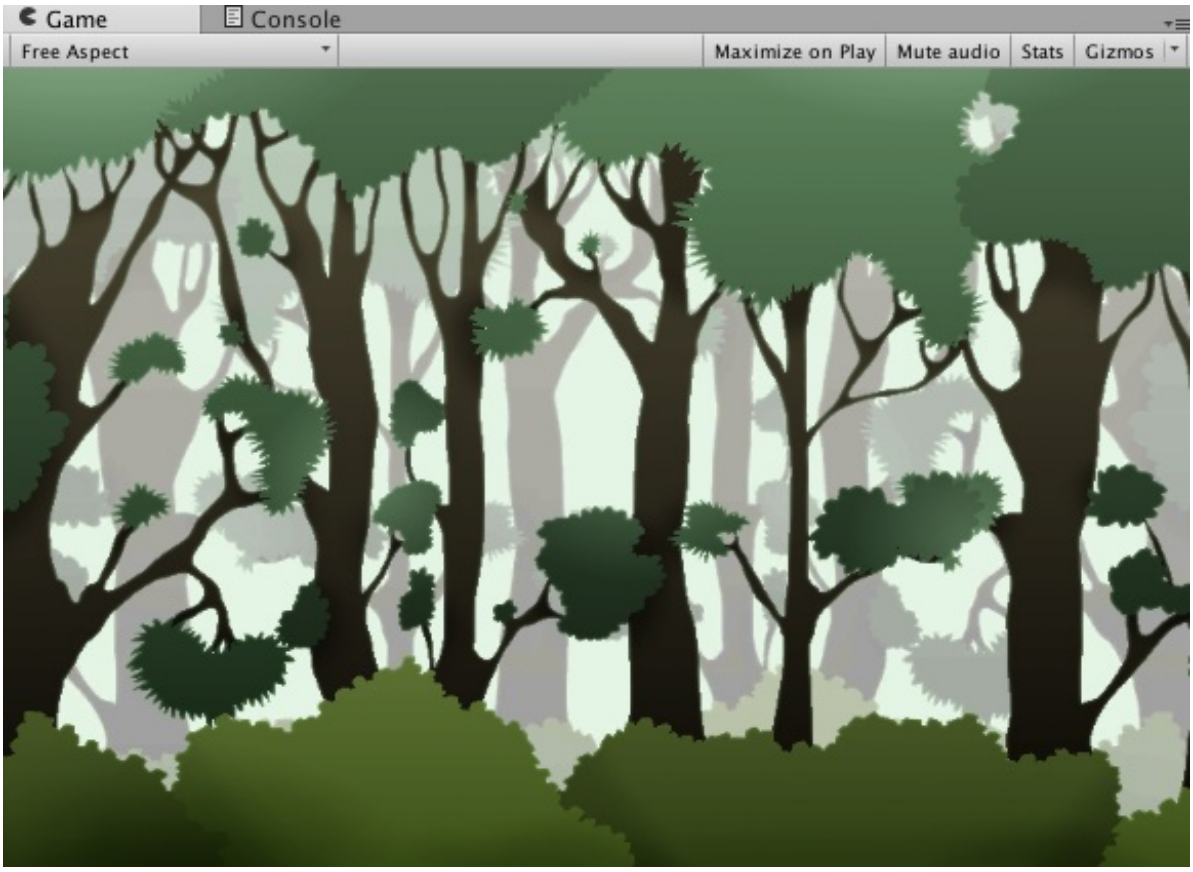


图11.3 无限滚动的背景（纹理来源：forest-background © 2012-2013 Julien Jorge julien.jorge@stuff-o-matic.com）

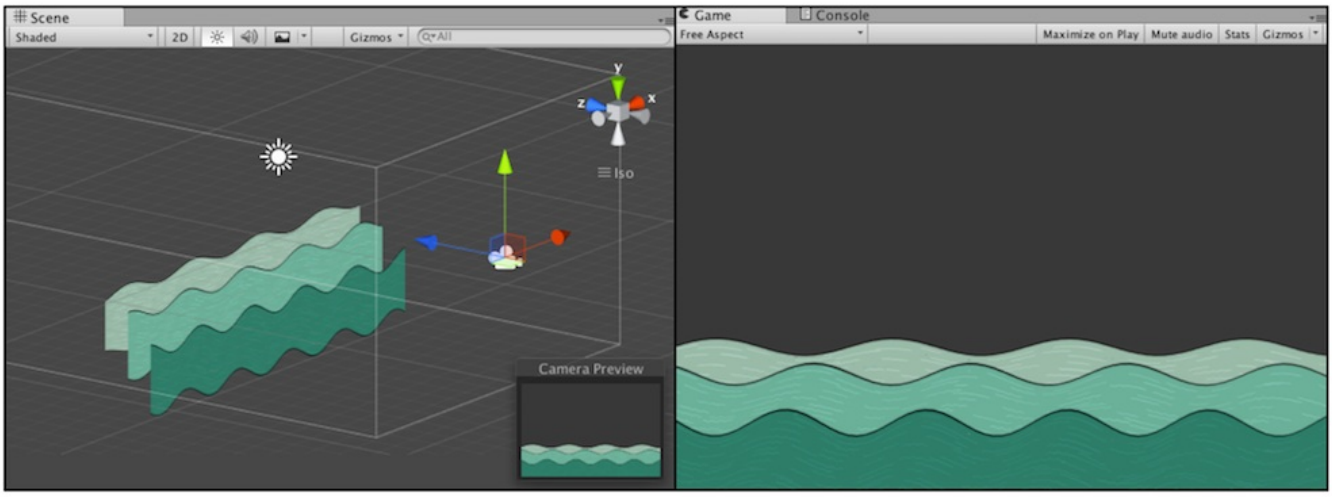


图11.4 使用顶点动画来模拟2D的河流

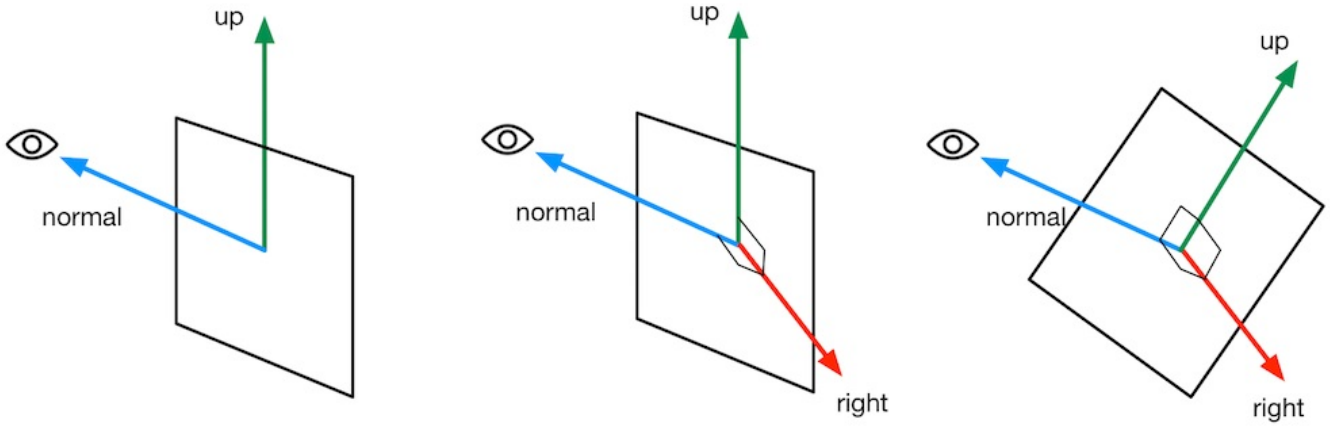


图11.5 法线固定（总是指向视角方向）时，计算广告牌技术中的三个正交基的过程

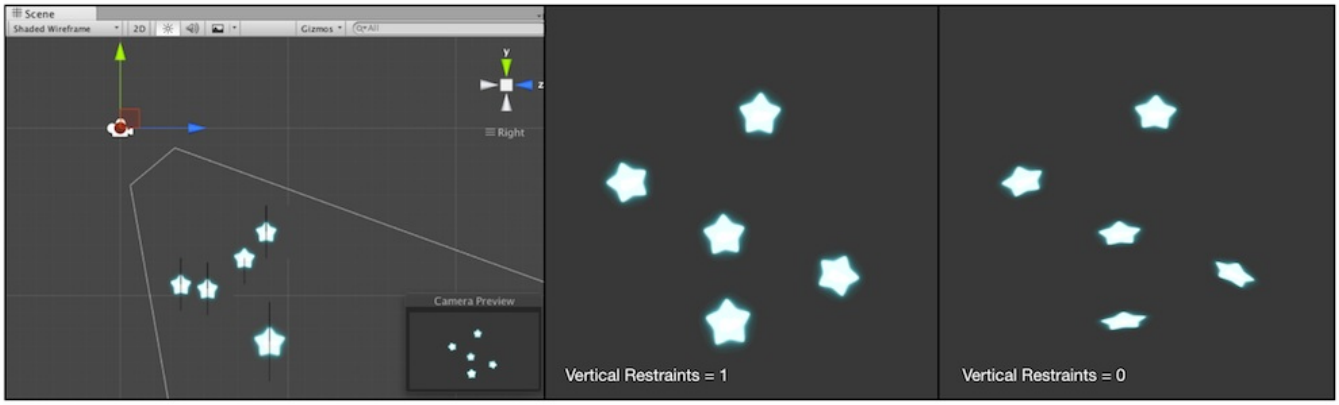


图11.6 广告牌效果。左图显示了摄像机和5个广告牌之间的位置关系，摄像机是从斜上方向下观察它们的。中间的图显示了当Vertical Restraints属性为1，即固定法线方向为观察视角时所得到的效果，可以看出，所有的广告牌都完全面朝摄像机。右图显示了当Vertical Restraints属性为0，即固定指向上的方向为(0, 1, 0)时所得到的效果，可以看出，广告牌虽然最大限度地

面朝摄像机，但其指向上的方向并未发生改变

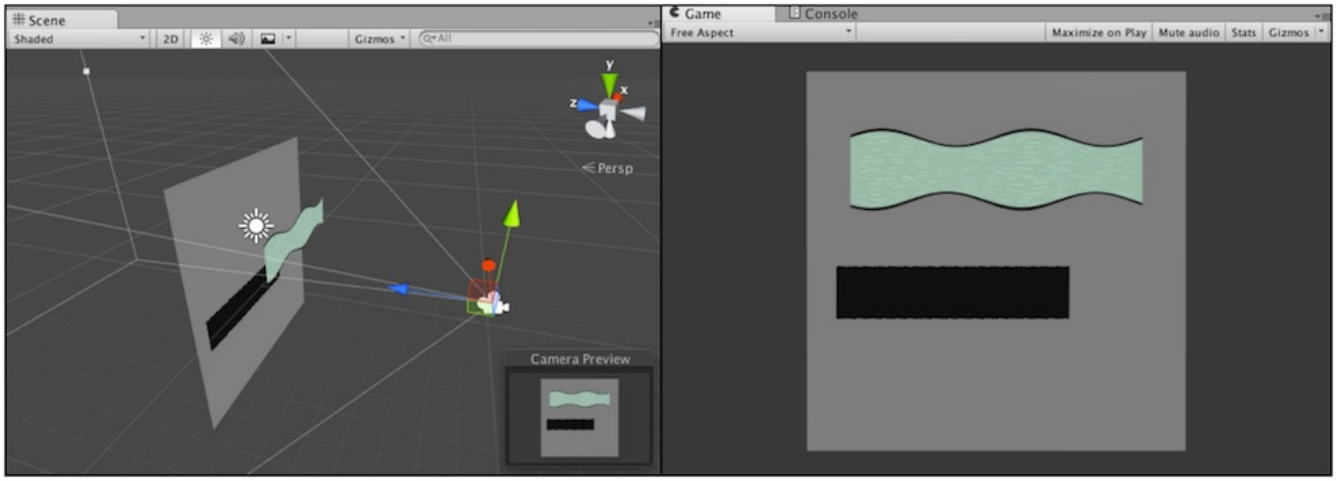


图11.7 当进行顶点动画时，如果仍然使用内置的ShadowCaster Pass来渲染阴影，可能会得到错误的阴影效果

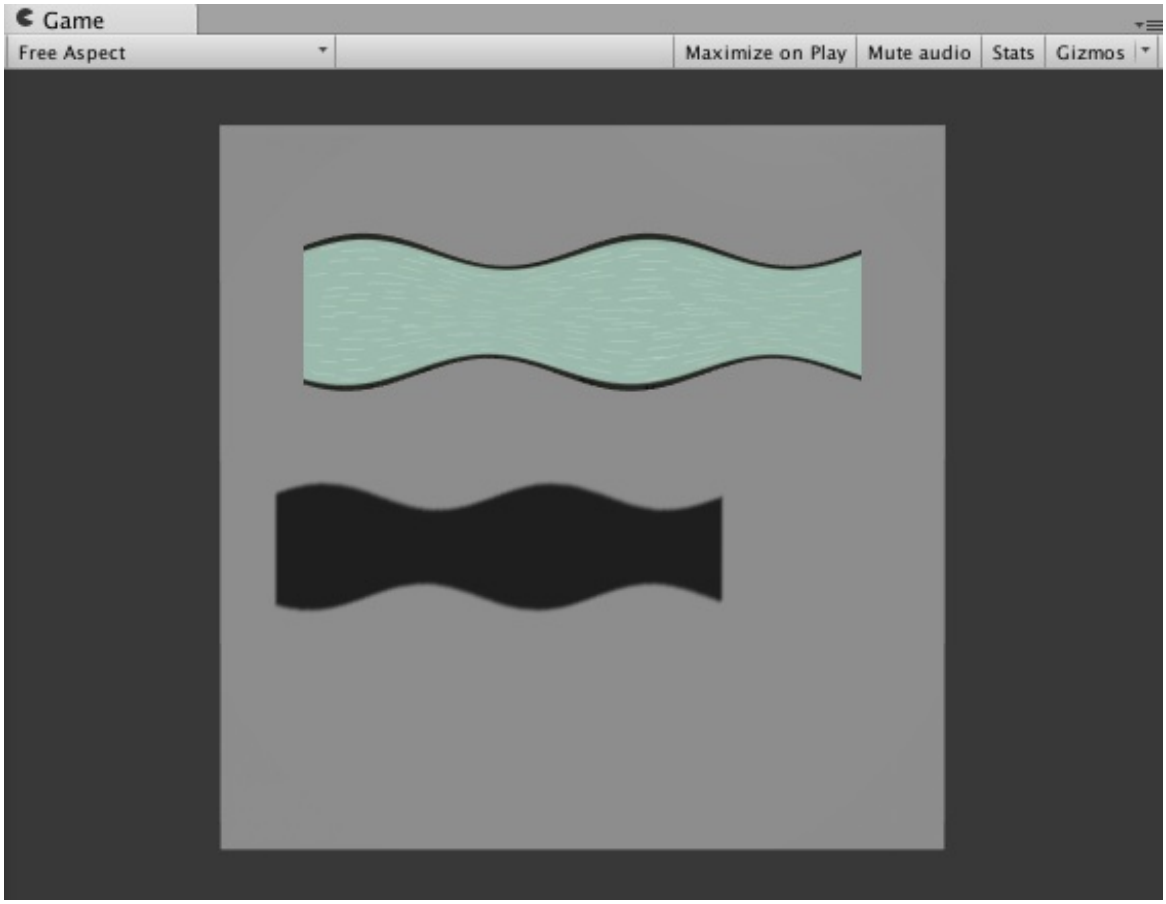


图11.8 使用自定义的ShadowCaster Pass□为变形物体绘制正确的阴影

第12章 屏幕后处理效果

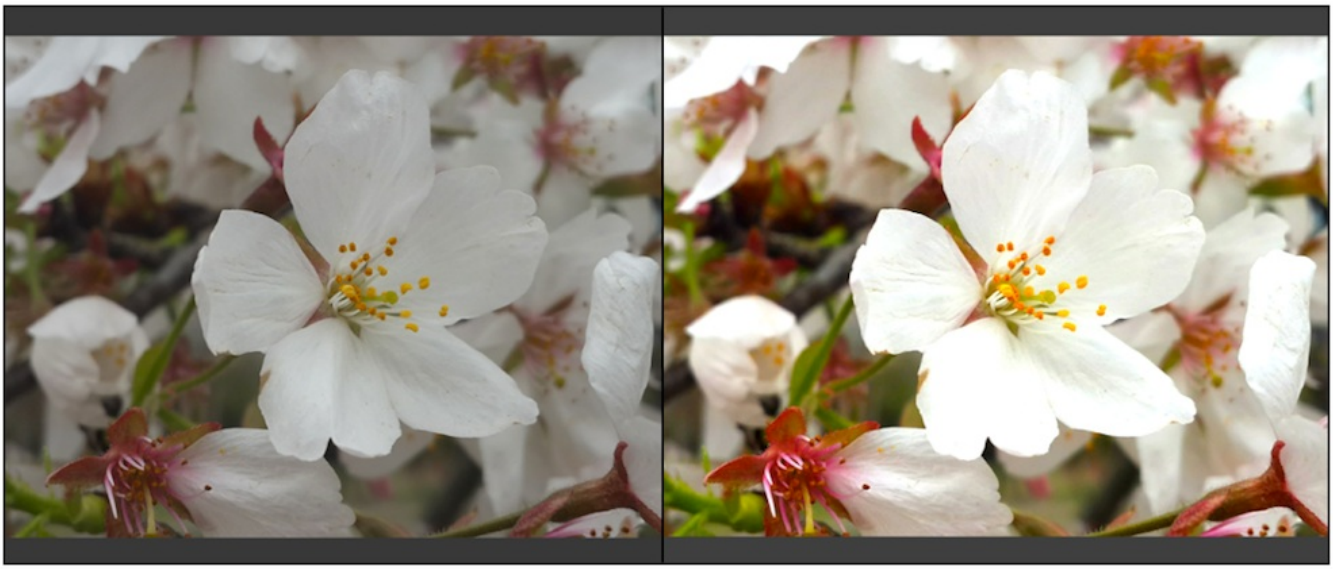


图12.1 左图：原效果。右图：调整了亮度（值为1.2）、饱和度（值为1.6）和对比度（值为1.2）后的效果

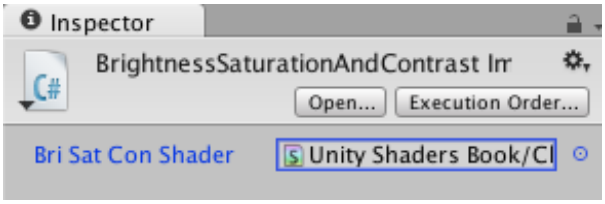


图12.2 为脚本设置Shader的默认值

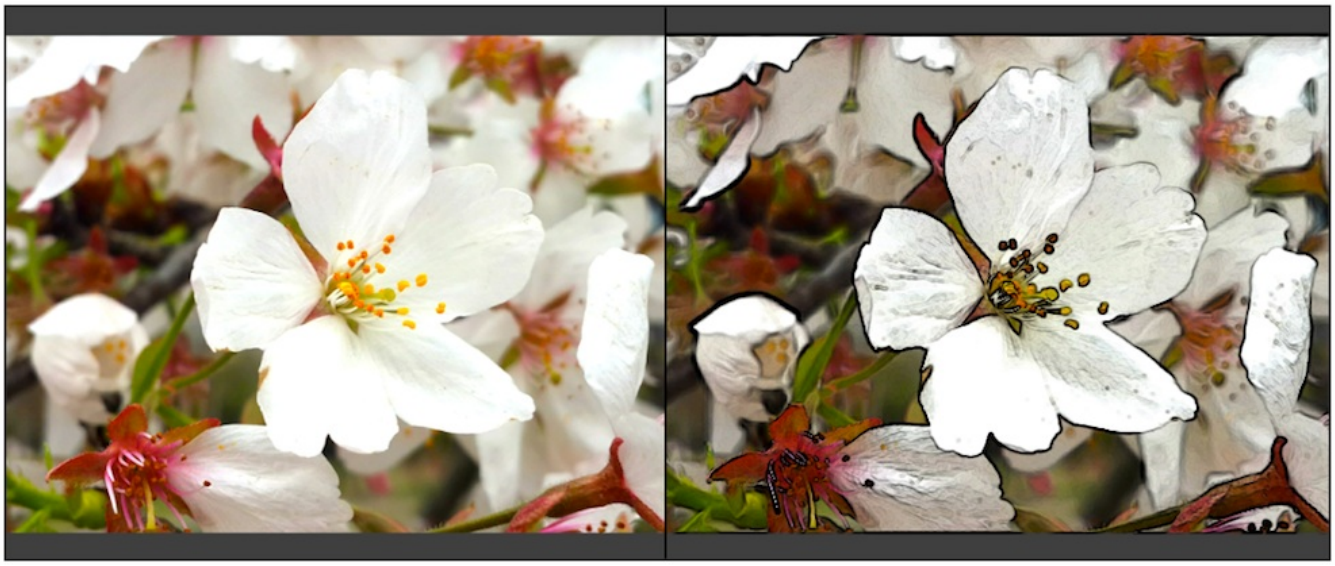


图12.3 左图：12.2节得到的结果。□右图：进行边缘检测后的效果

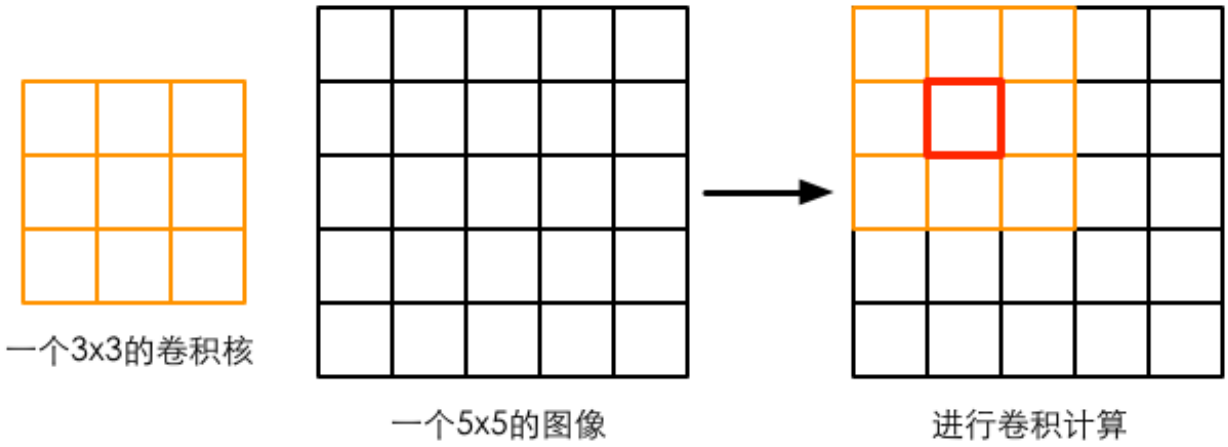


图12.4 卷积核与卷积。使用一个3×3大小的卷积核对一张5×5大小的图像进行卷积操作，当计算图中红色方块对应的像素的卷积结果时，我们首先把卷积核的中心放置在该像素位置，翻转核之后再依次计算核中每个元素和其覆盖的图像像素值的乘积并求和，得到新的像素值

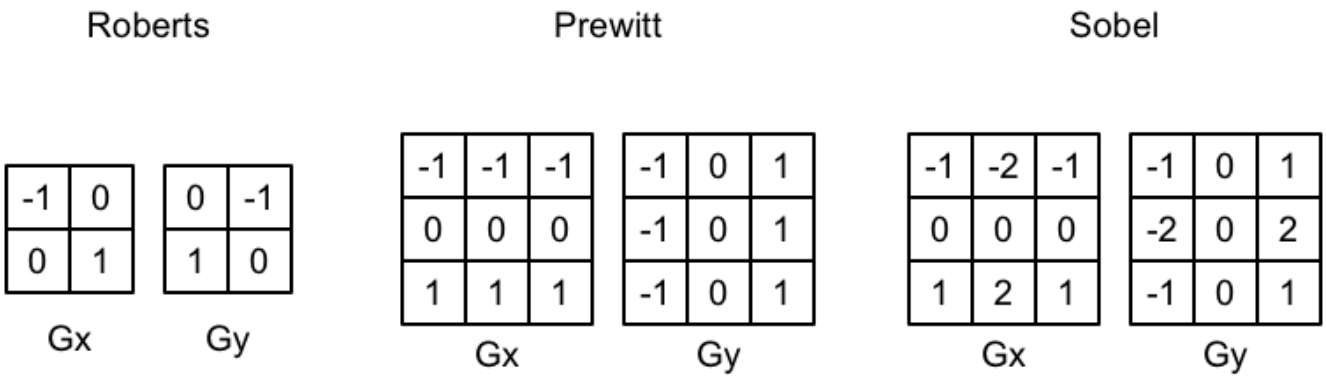


图12.5 三种常见的边缘检测算子

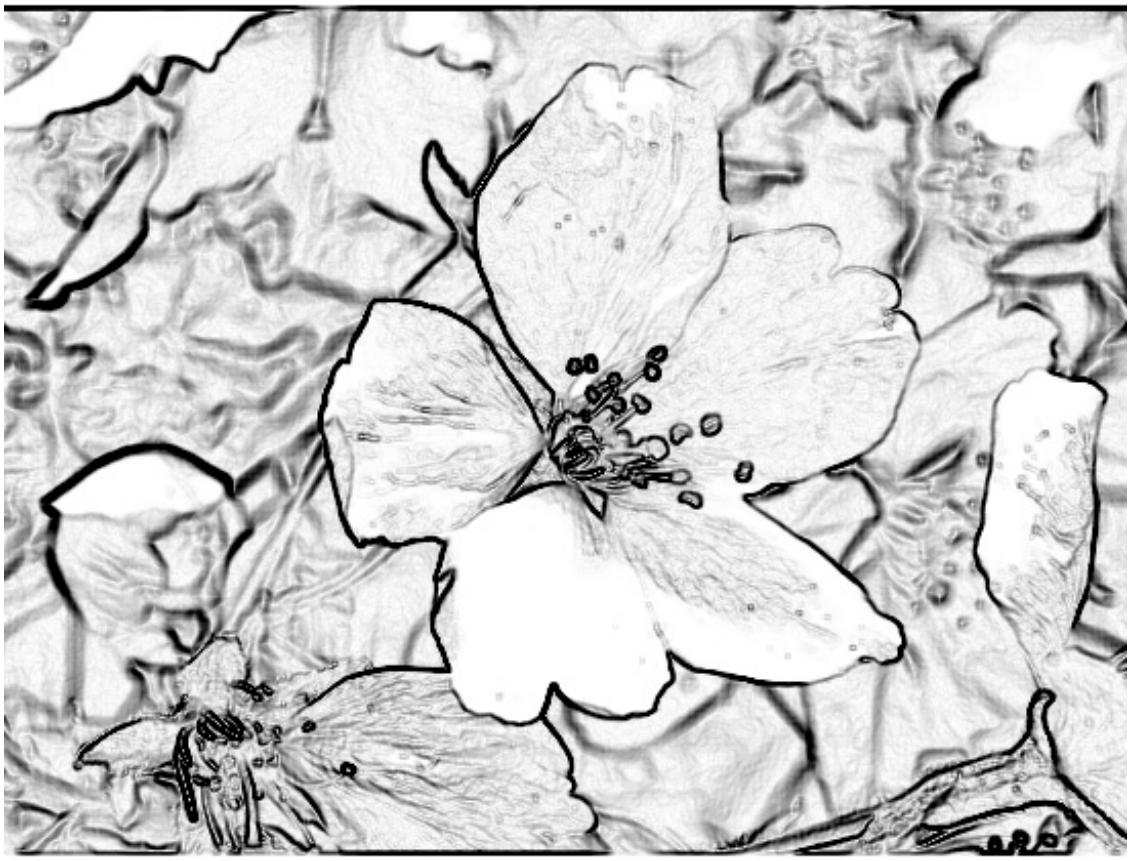


图12.6 只显示边缘的屏幕效果

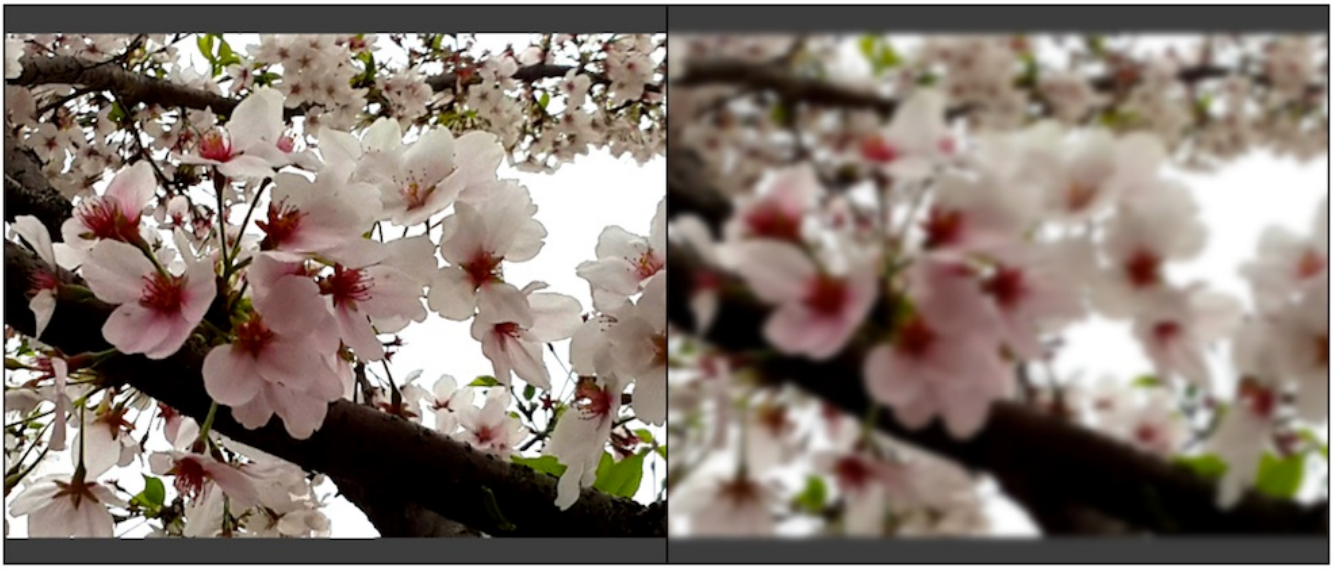


图12.7 左图：原效果。右图：高斯模糊后的效果

0.0030	0.0133	0.0219	0.0133	0.0030
0.0133	0.0596	0.0983	0.0596	0.0133
0.0219	0.0983	0.1621	0.0983	0.0219
0.0133	0.0596	0.0983	0.0596	0.0133
0.0030	0.0133	0.0219	0.0133	0.0030

0.0545	0.2442	0.4026	0.2442	0.0545

0.0545
0.2442
0.4026
0.2442
0.0545

图12.8 一个5×5大小的高斯核。左图显示了标准方差为1的高斯核的权重分布。我们可以把这个二维高斯核拆分成两个一维的高斯核（右图）

图12.9 动画短片《大象之梦》中的Bloom效果。光线透过门扩散到了周围较暗的区域中

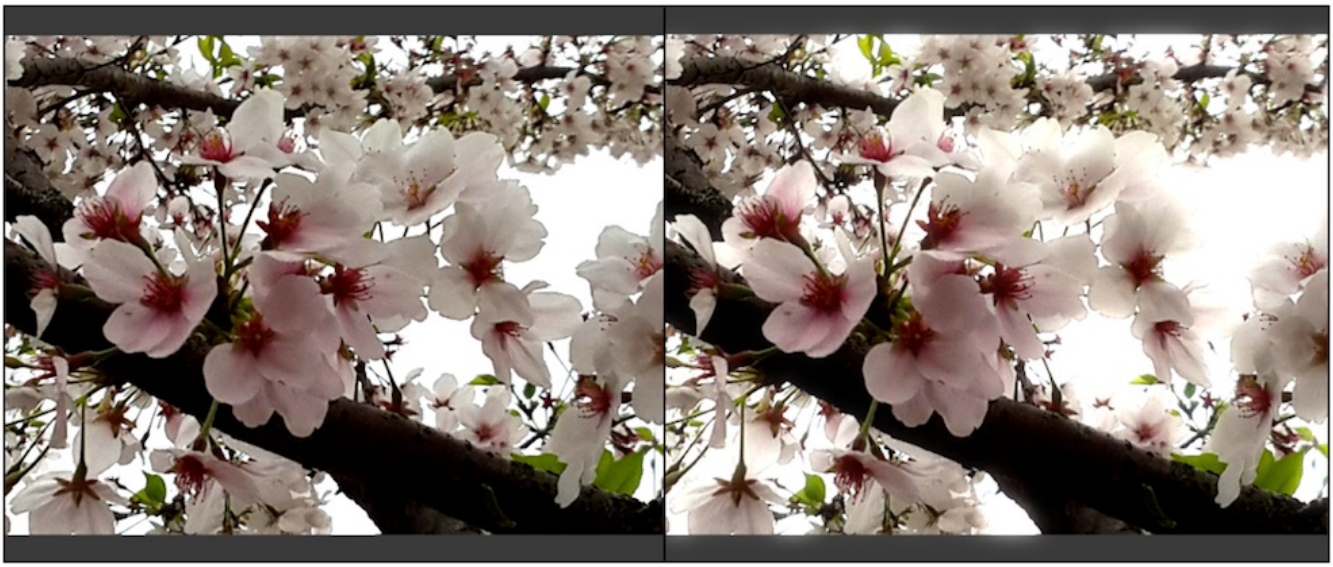


图12.10 左图：原效果。右图：Bloom处理后的效果

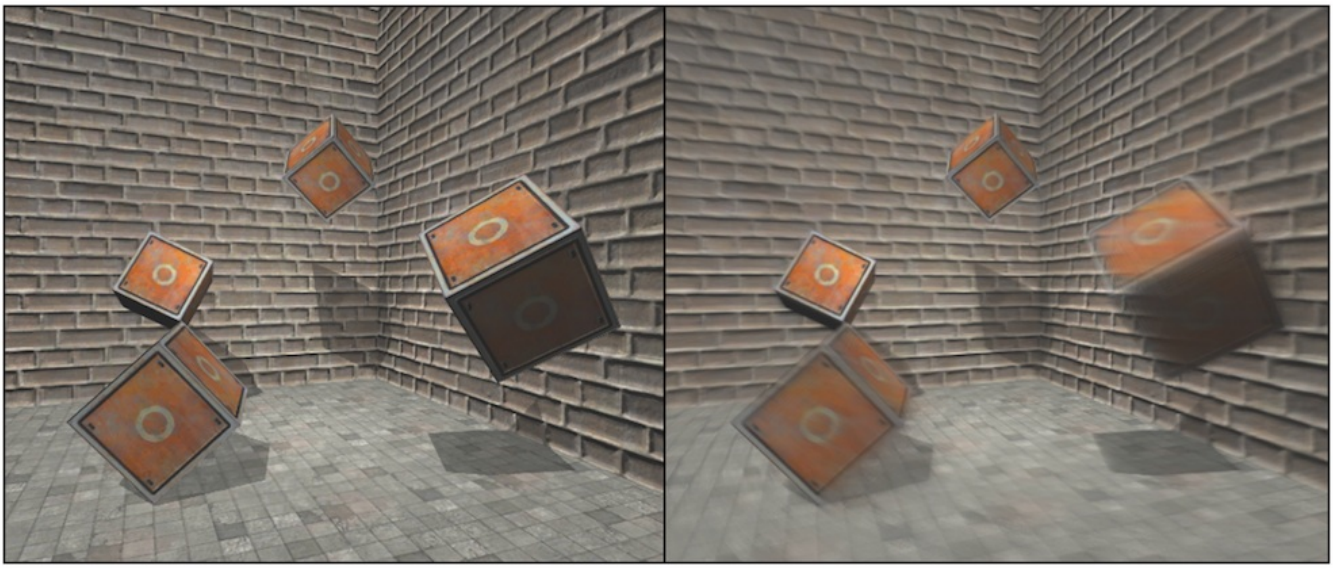


图12.11 左图：原效果。右图：应用运动模糊后的效果

第13章 使用深度和法线纹理

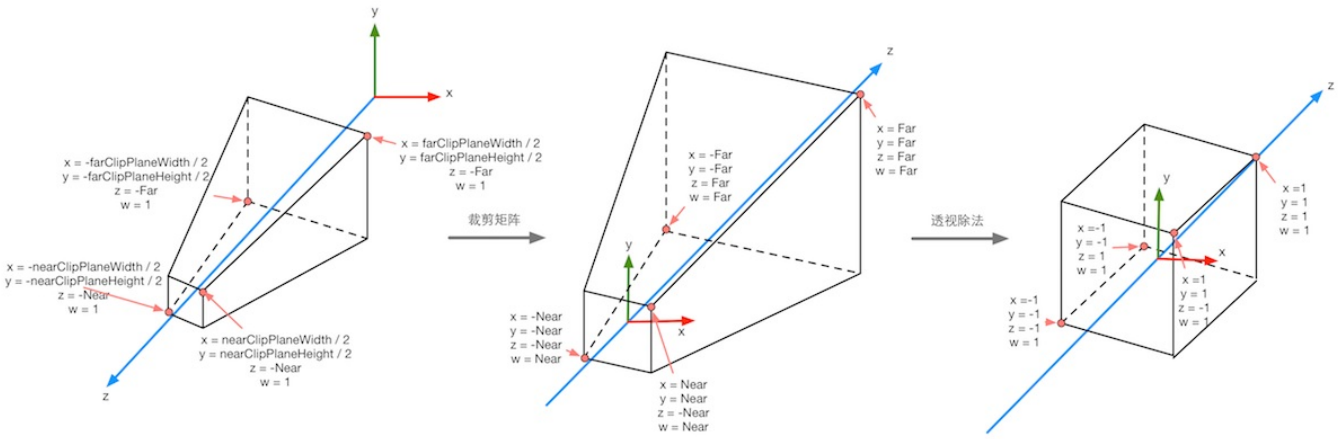


图13.1 在透视投影中，投影矩阵首先对顶点进行了缩放。在经过齐次除法后，透视投影的裁剪空间会变换到一个立方体。图中标注了4个关键点经过投影矩阵变换后的结果

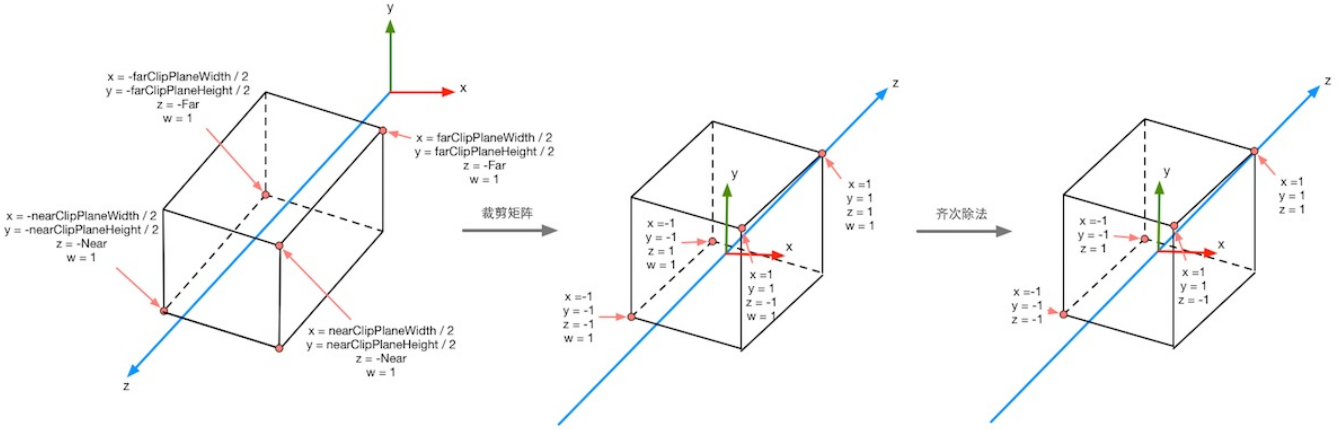


图13.2 在正交投影中，投影矩阵对顶点进行了缩放。在经过齐次除法后，正交投影的裁剪空间会变换到一个立方体。图中标注了4个关键点经过投影矩阵变换后的结果

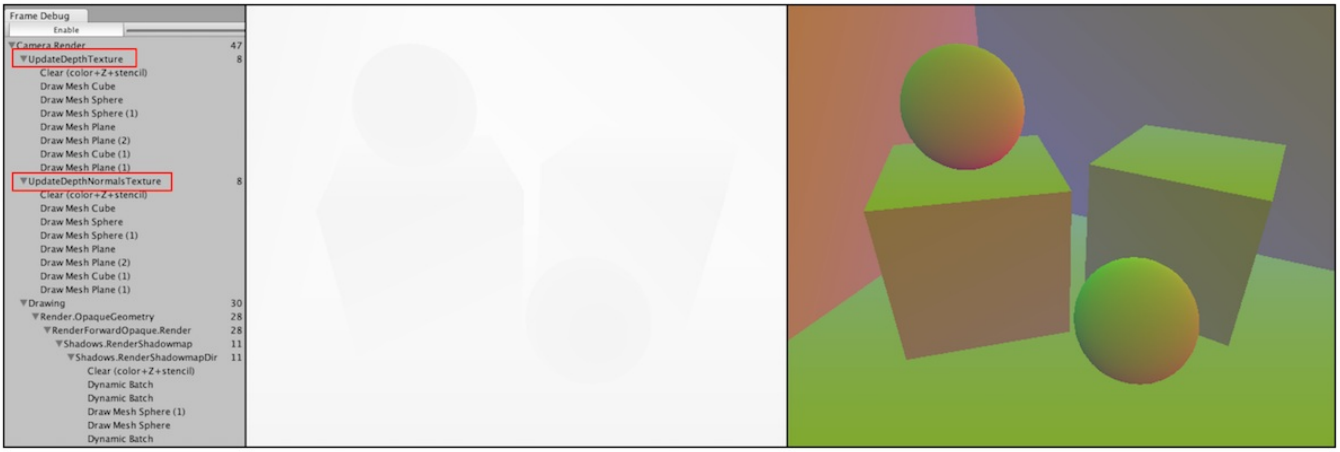


图13.3 使用Frame Debugger查看深度纹理（左）和深度+法线纹理（右）。如果当前摄像机需要生成深度和法线纹理，帧调试器的面板中就会出现相应的渲染事件。只要单击对应的事件就可以查看得到的深度和法线纹理

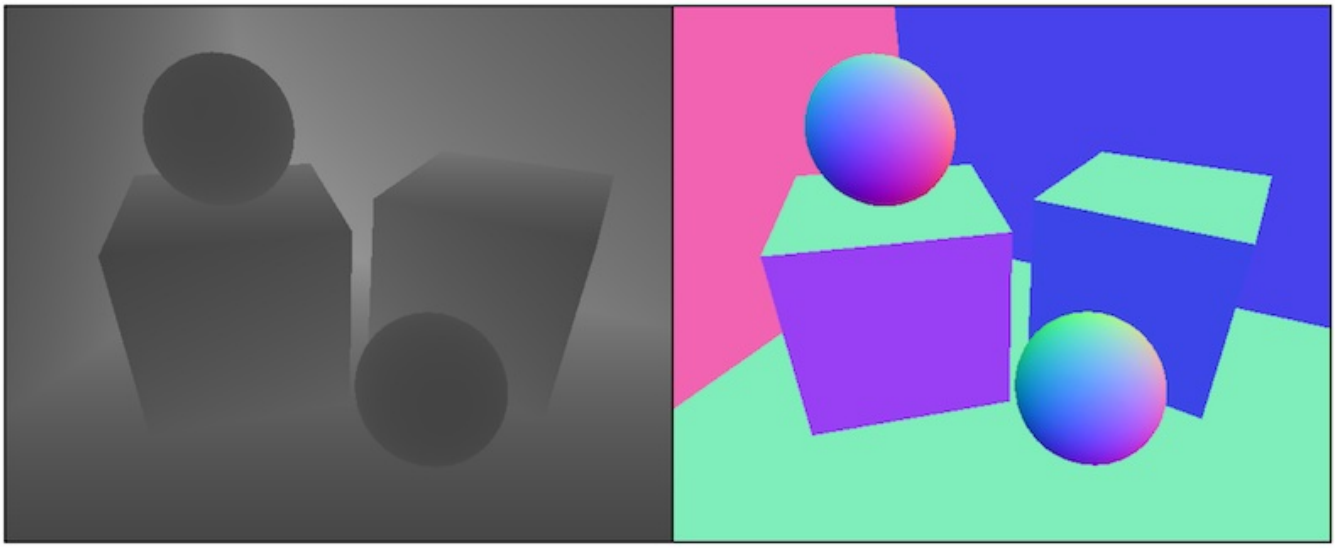


图13.4 左图：线性空间下的深度纹理。右图：解码后并且被映射到[0, 1]范围内的视角空间下的法线纹理

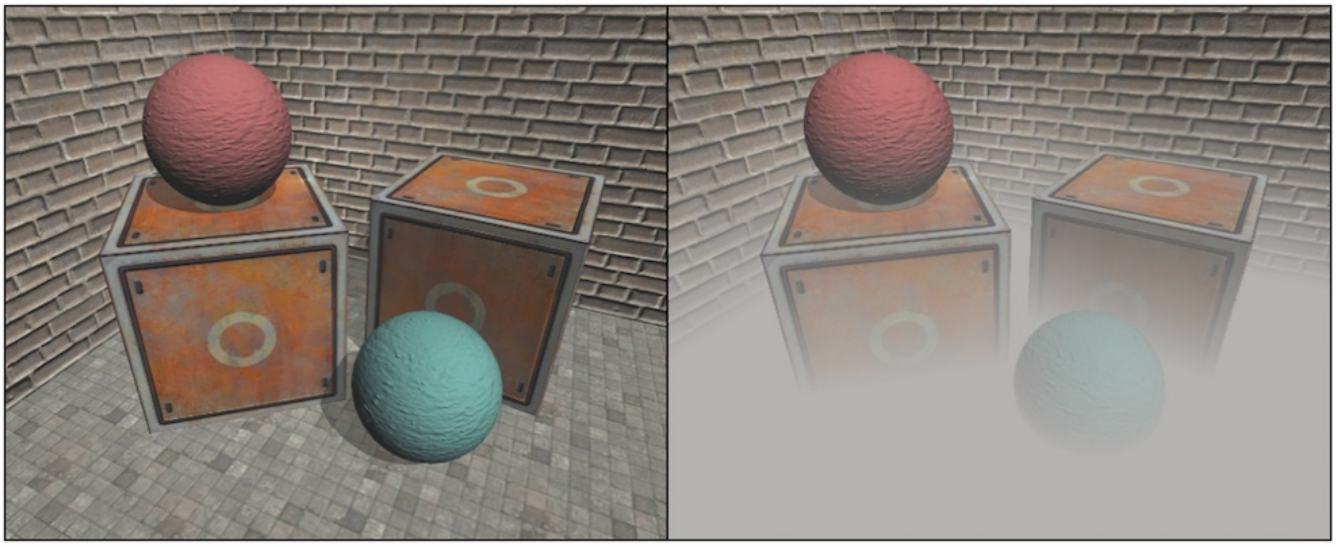


图13.5 左图：原效果。右图：添加全局雾效后的效果

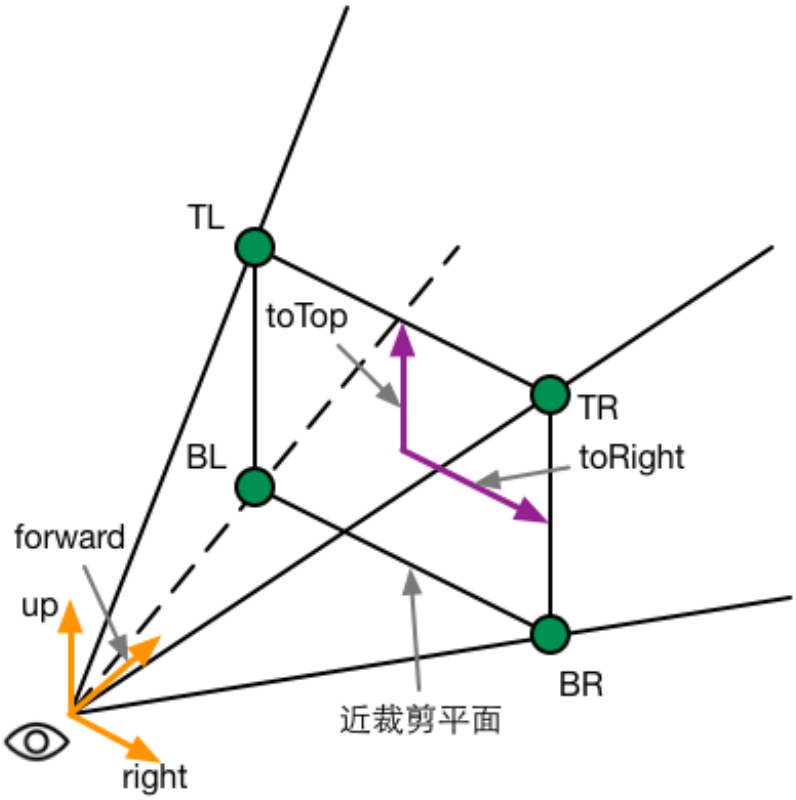


图13.6 计算interpolatedRay

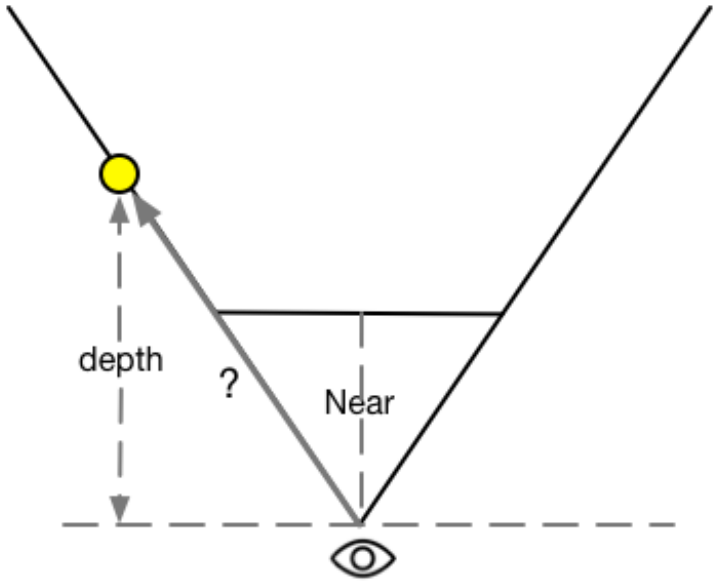


图13.7 采样得到的深度值并非是点到摄像机的欧式距离

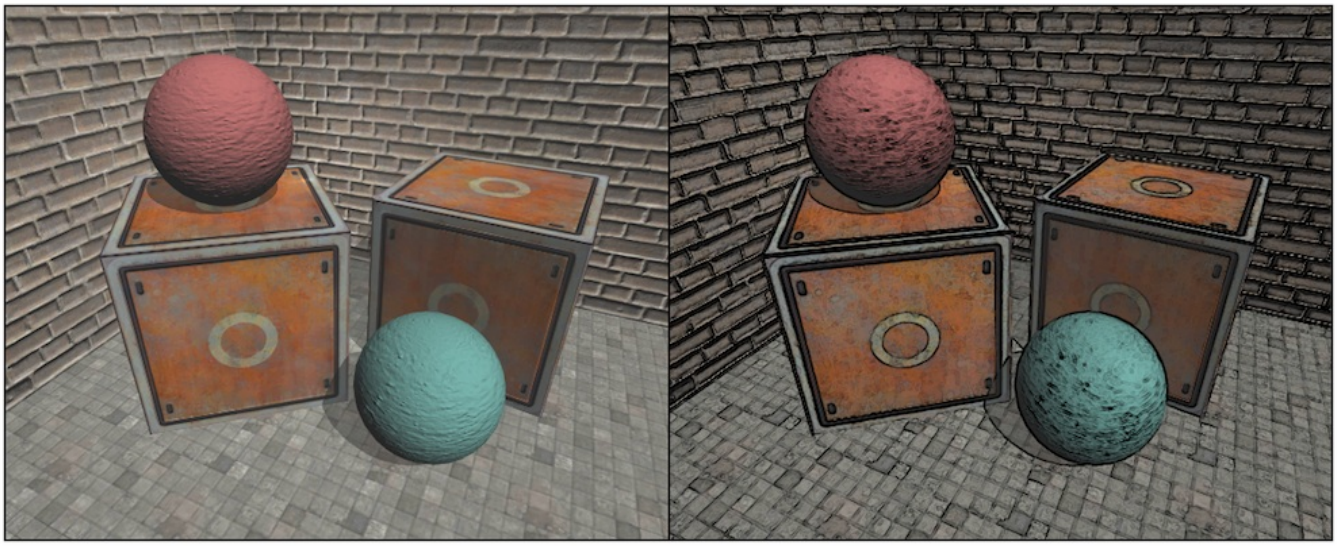


图13.8 左图：原效果。右图：直接对颜色图像进行边缘检测的结果

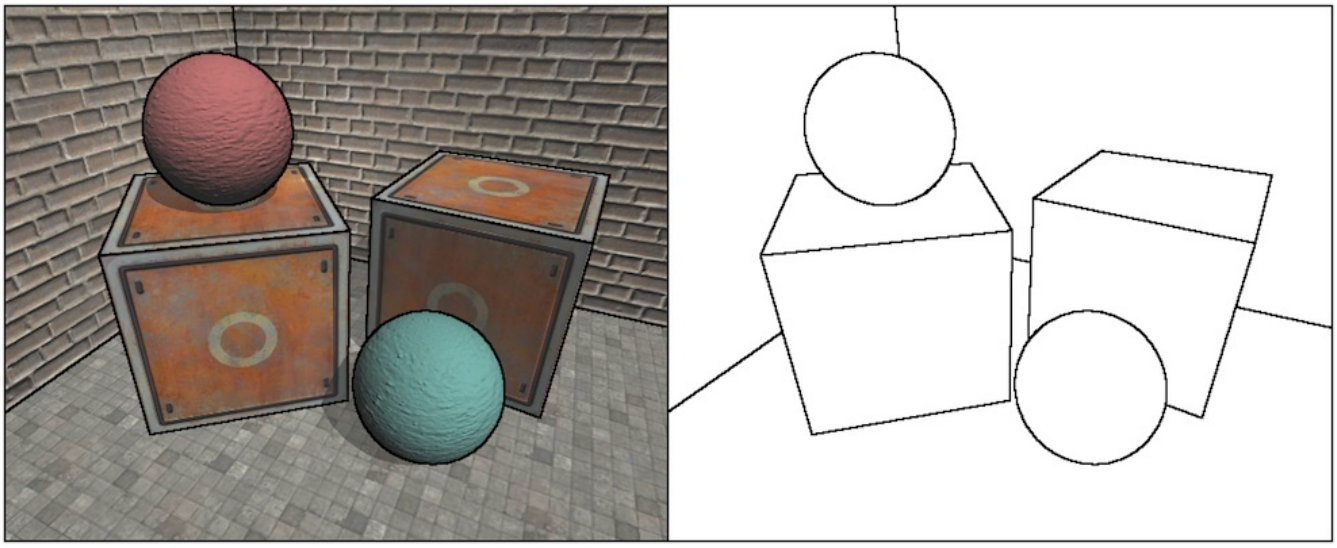


图13.9 在深度和法线纹理上进行更健壮的边缘检测。左图：在原图上描边的效果。右图：只显示描边的效果

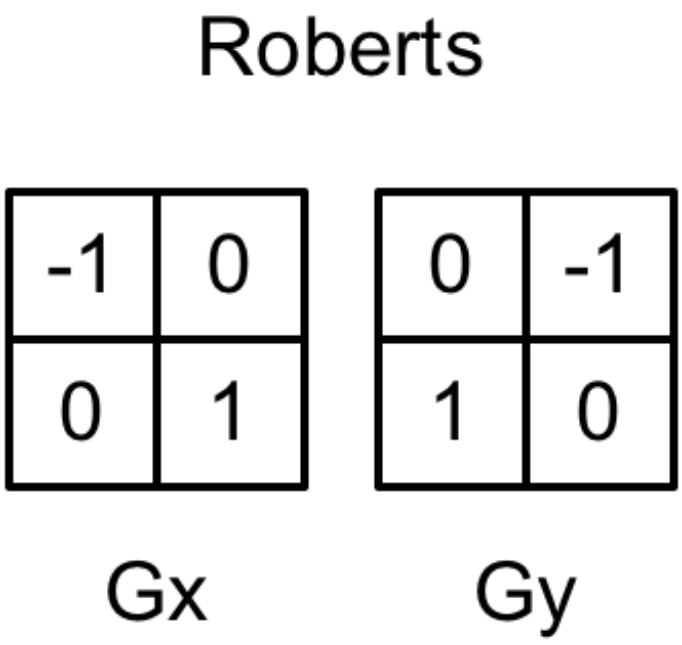


图13.10 Roberts算子

第14章 非真实感渲染

图14.1 游戏《大神》（英文名：Okami）的游戏截图

图14.2 卡通风格的渲染效果

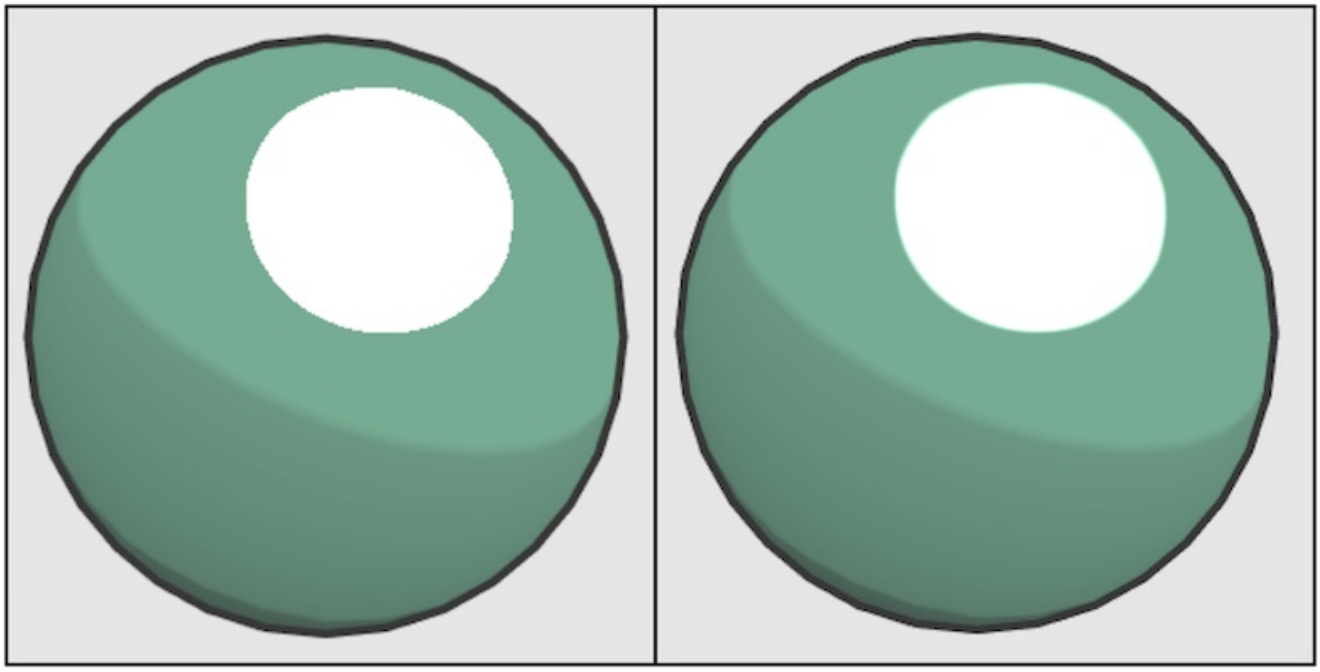


图14.3 左图：未对高光区域进行抗锯齿处理。右图：使用fwidth函数对高光区域进行抗锯齿处理

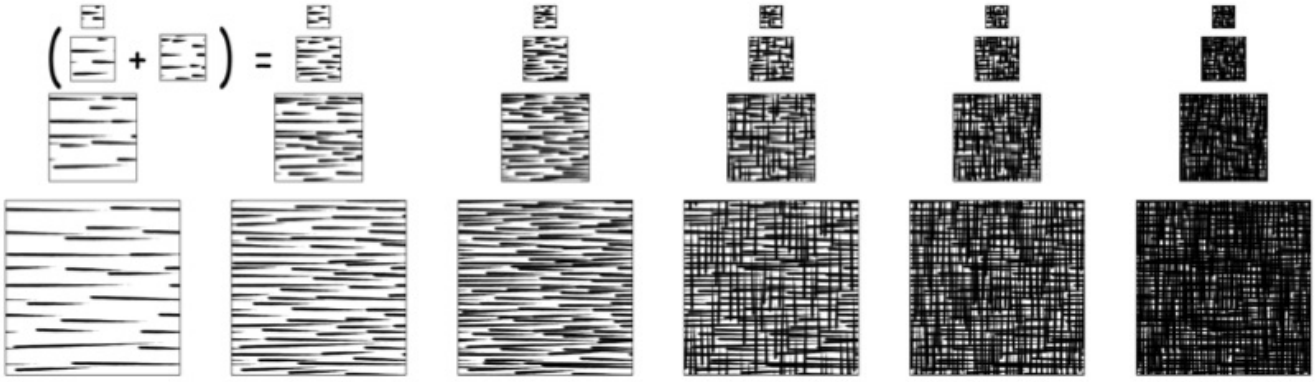


图14.4 一个TAM的例子（来源：Praun E, et al. Real-time hatching4）

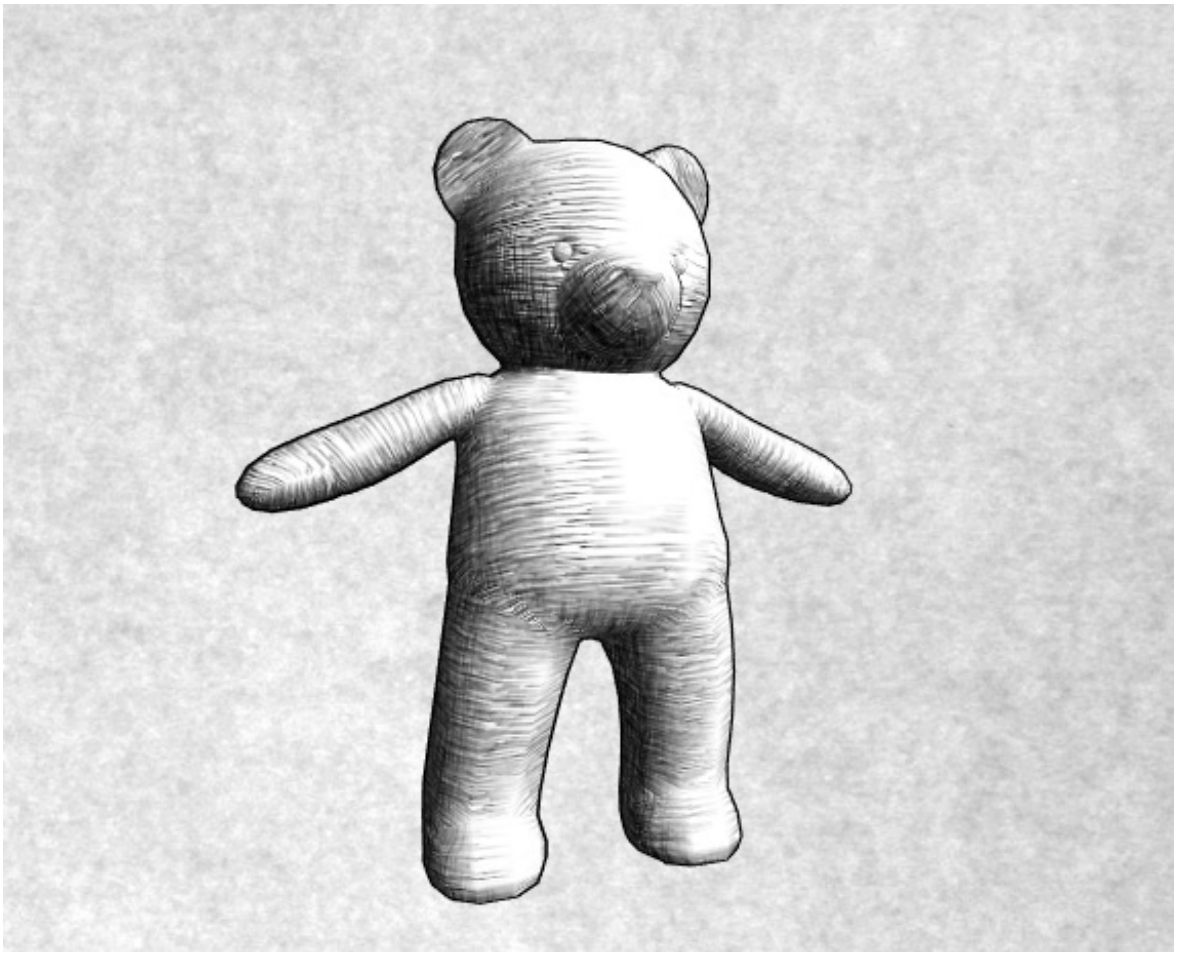


图14.5 素描风格的渲染效果

第15章 使用噪声

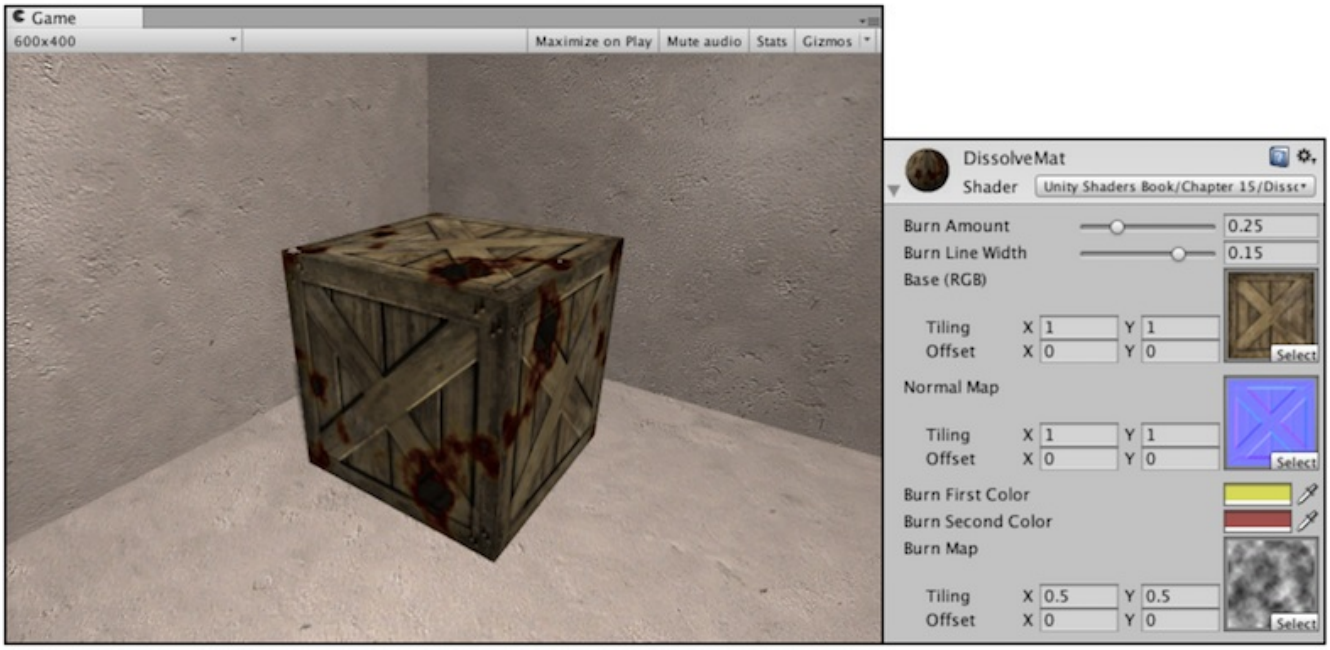


图15.1 箱子的消融效果

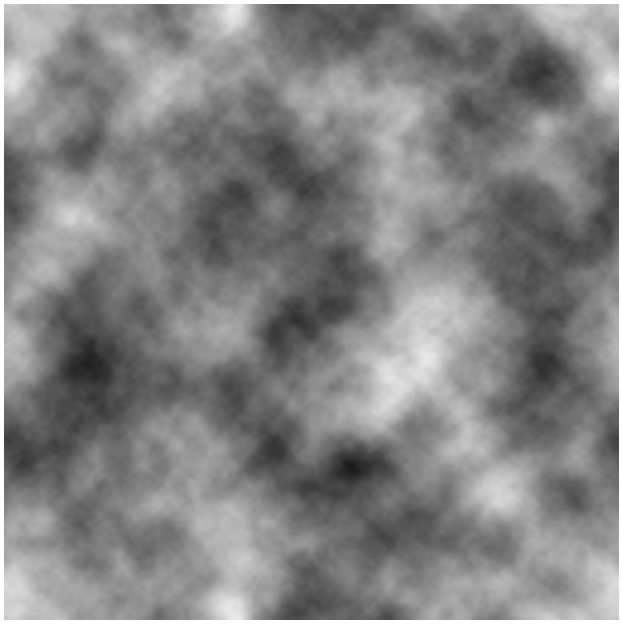


图15.2 消融效果使用的噪声纹理

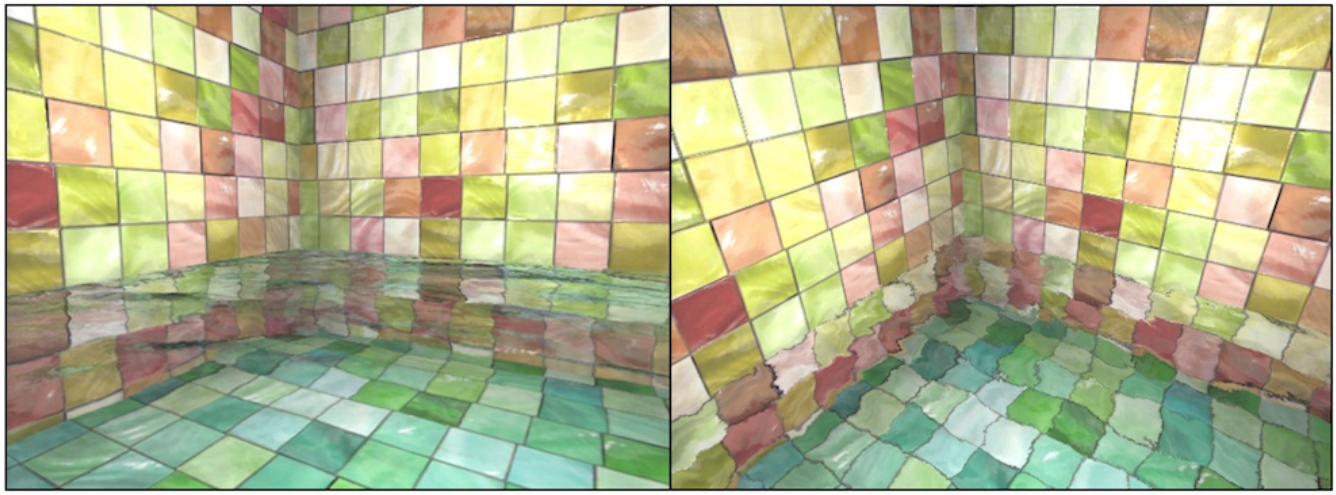


图15.3 包含菲涅耳反射的水面波动效果。在左图中，视角方向和水面法线的夹角越大，反射效果越强。在右图中，视角方向和水面法线的夹角越大，折射效果越强

图15.4 本例使用的立方体纹理

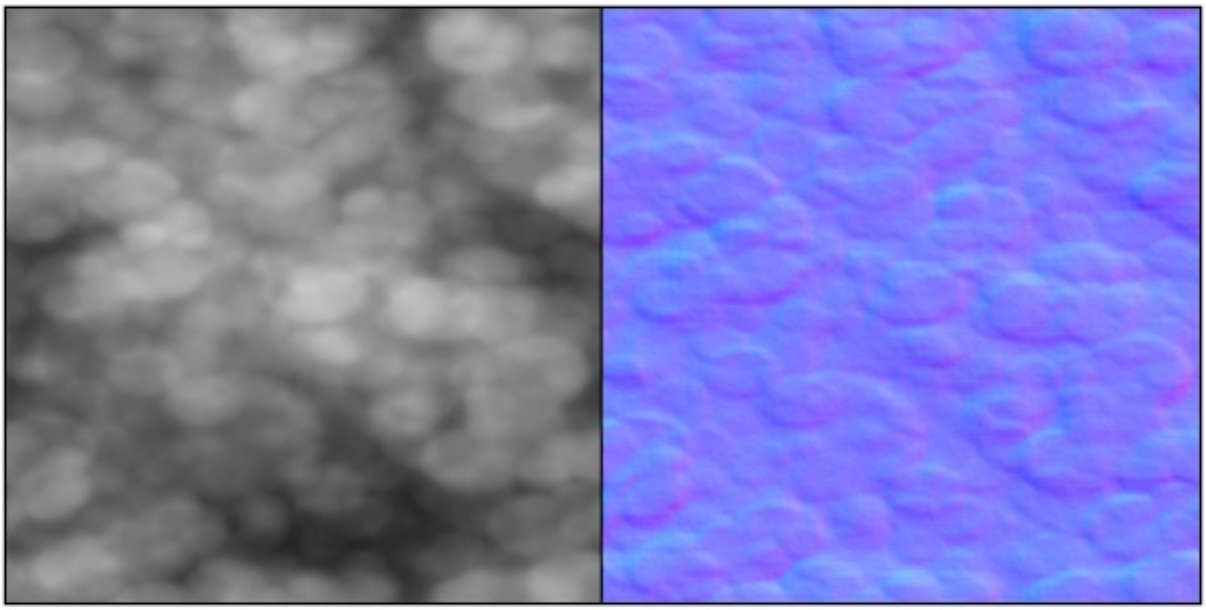


图15.5 水波效果使用的噪声纹理。左图：噪声纹理的灰度图。右图：由左图生成的法线纹理

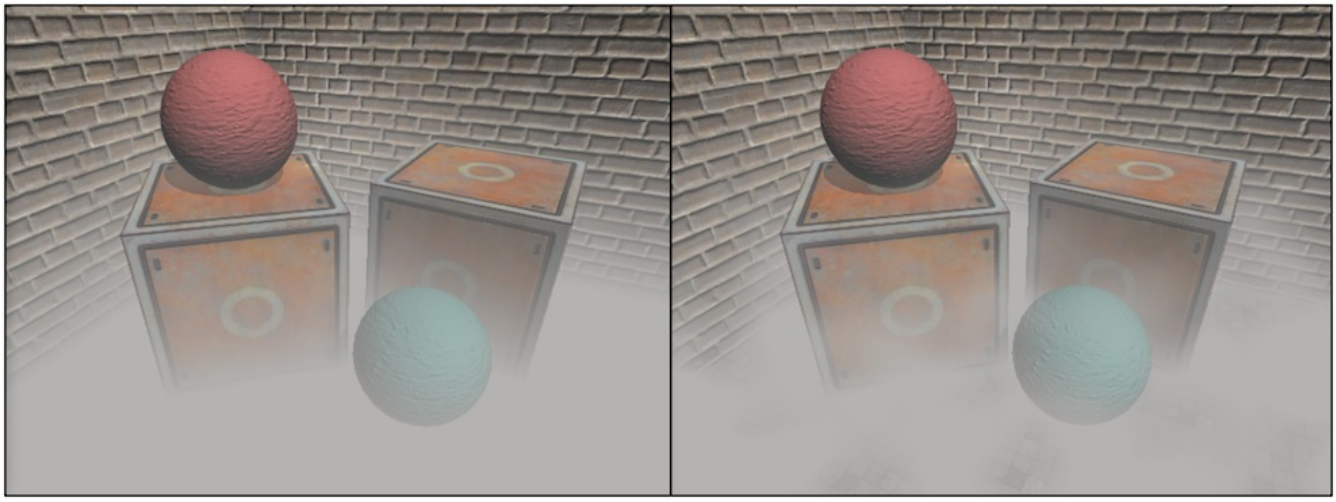


图15.6 左图：均匀雾效。右图：使用噪声纹理后的非均匀雾效

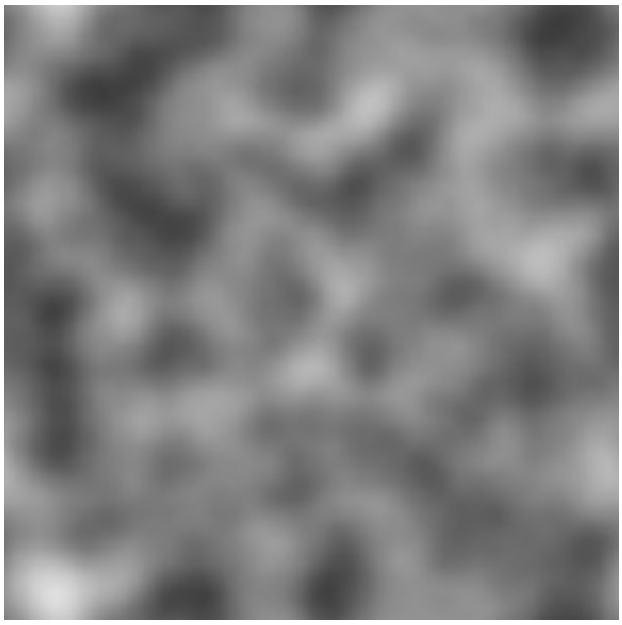


图15.7 本节使用的噪声纹理

第16章 Unity中的渲染优化技术

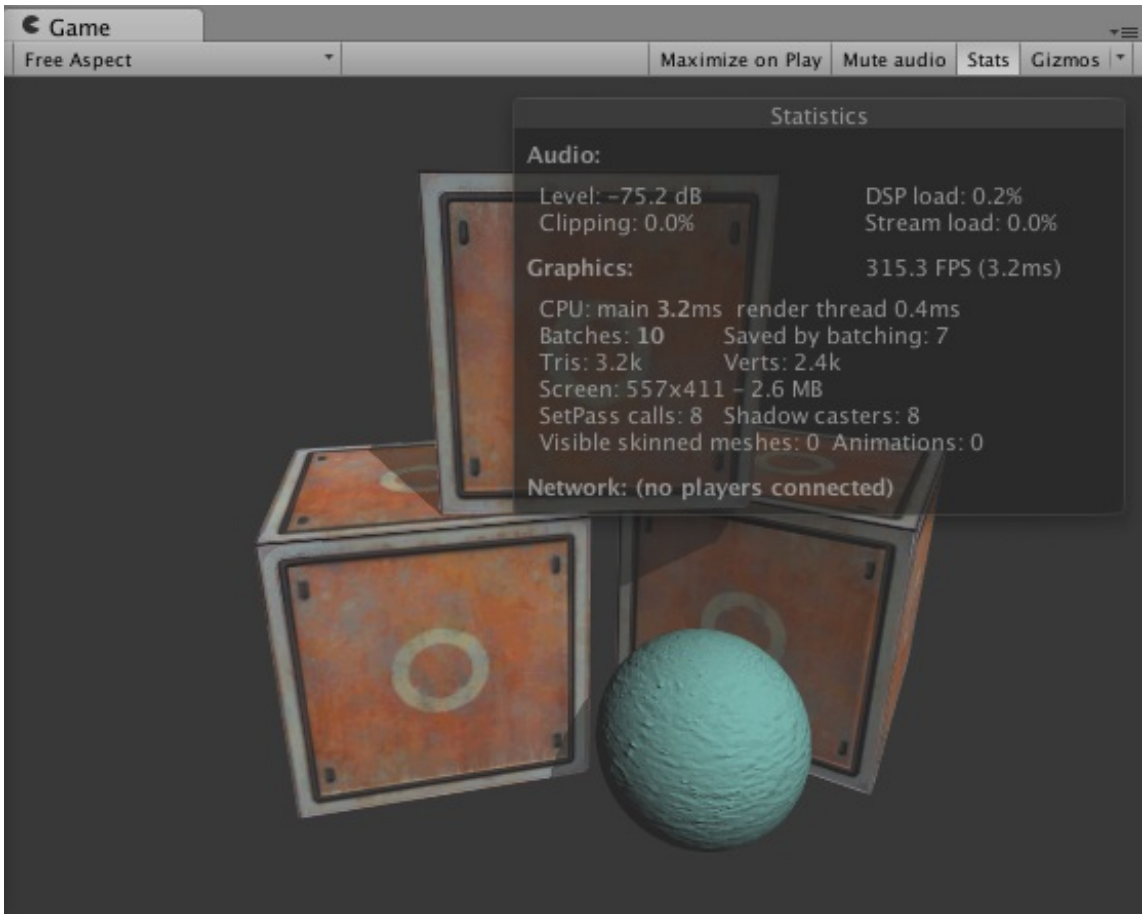


图16.1 Unity 5的渲染统计窗口

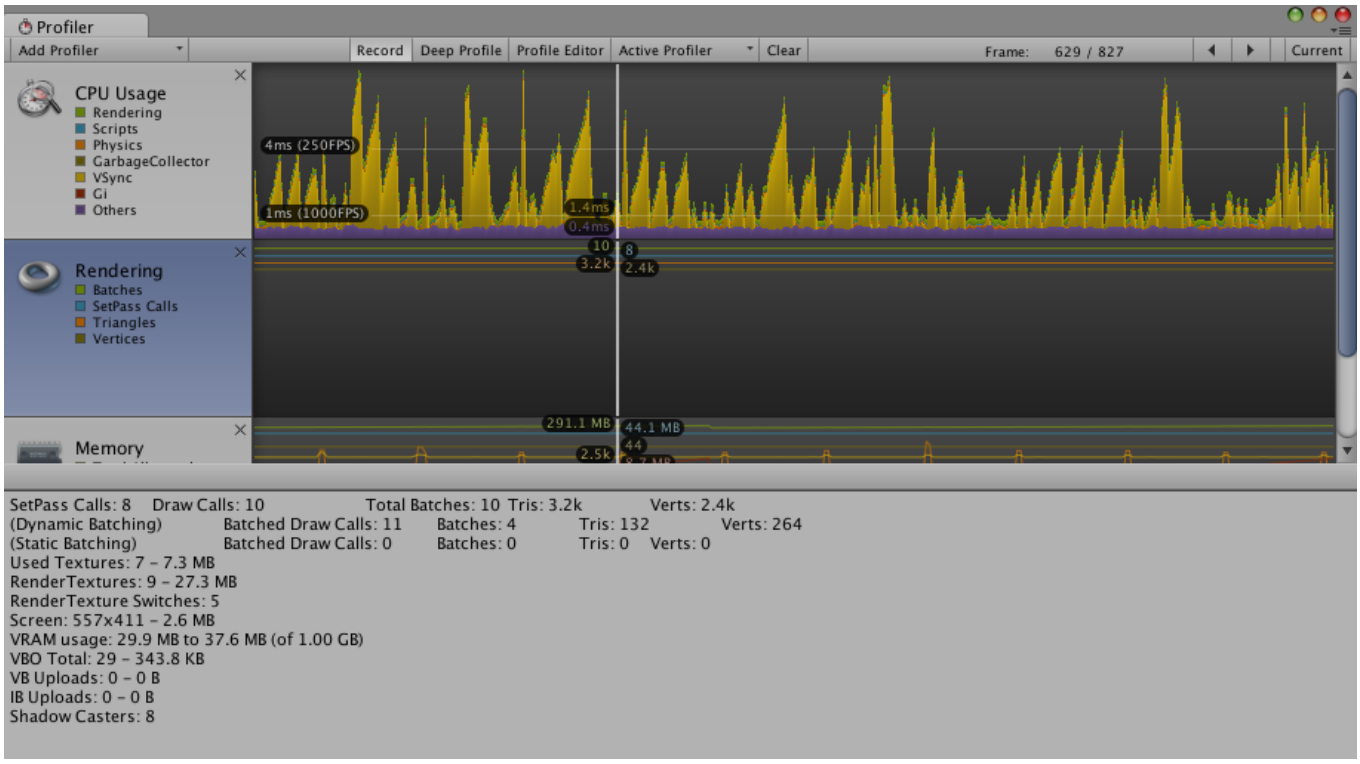


图16.2 使用Unity的性能分析器中的渲染区域来查看更多关于渲染的统计信息

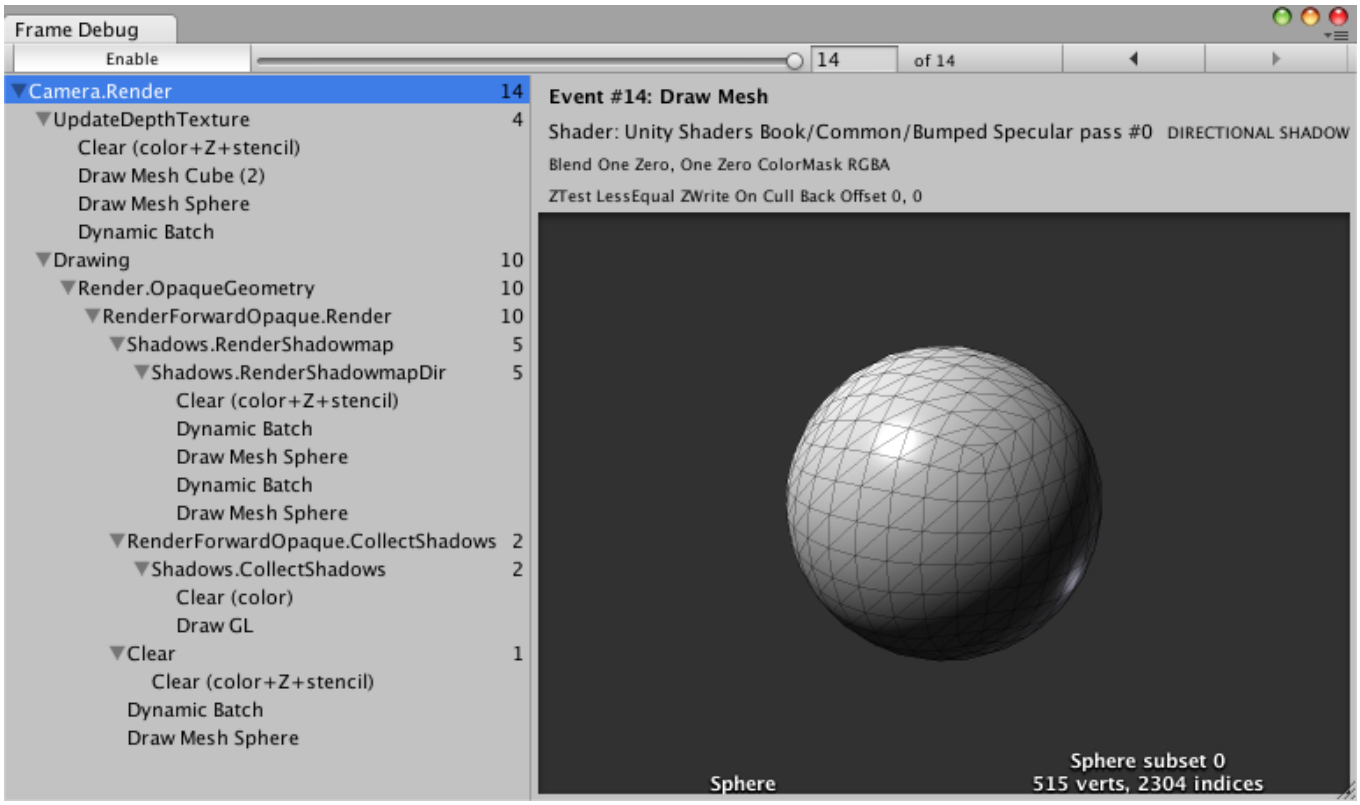


图16.3 使用帧调试器来查看单独的draw call的绘制结果

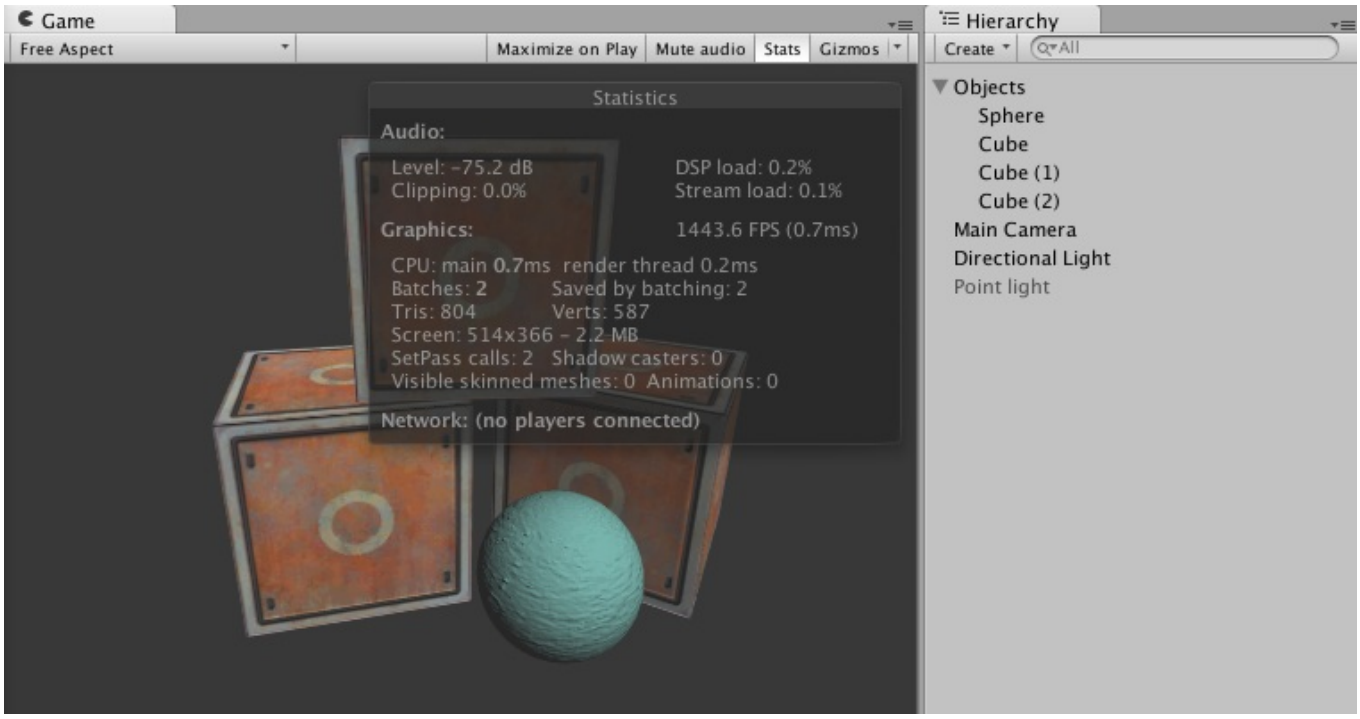


图16.4 动态批处理

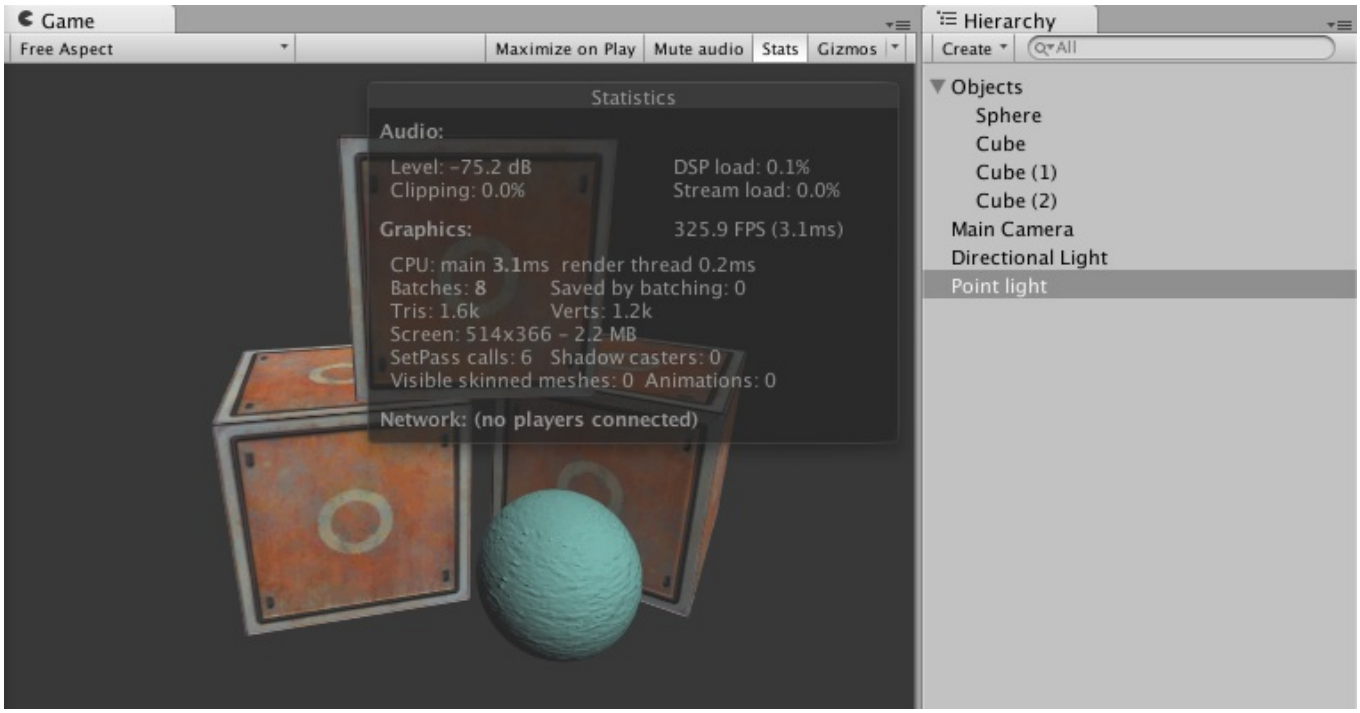


图16.5 多光源对动态批处理的影响结果

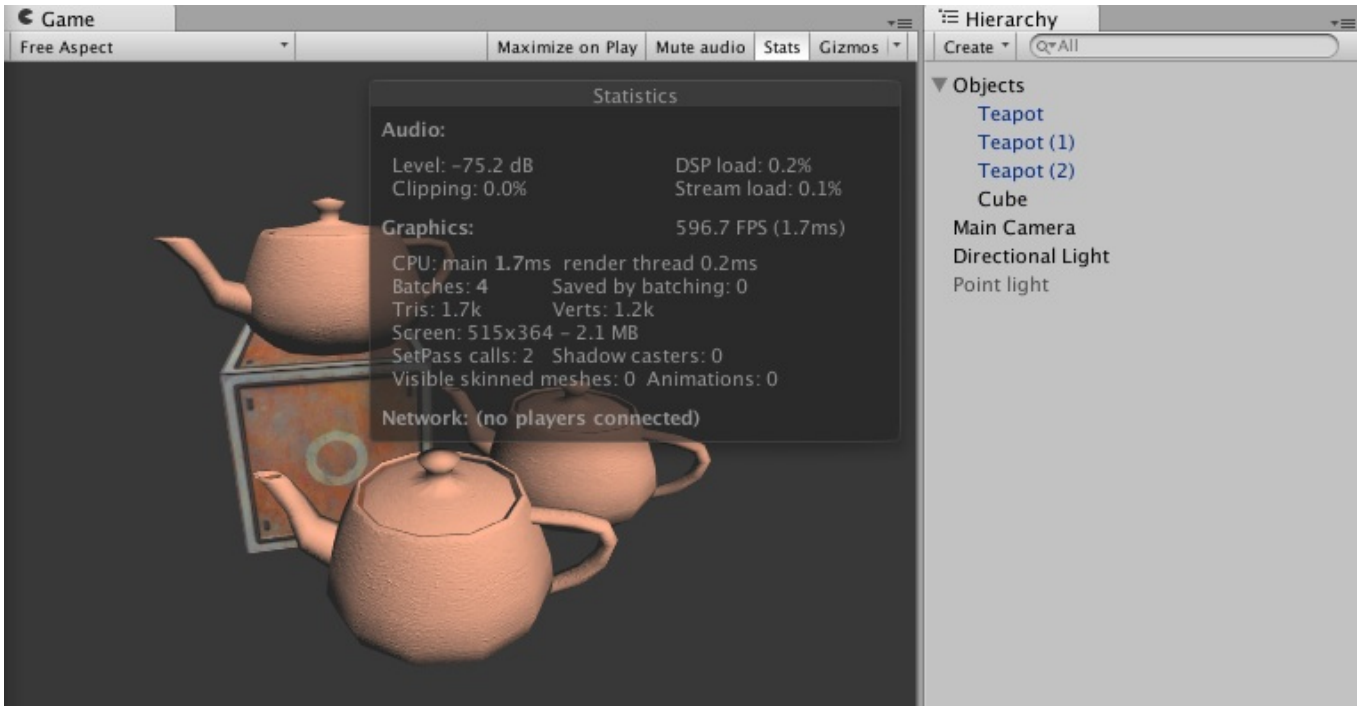


图16.6 静态批处理前的渲染统计数据

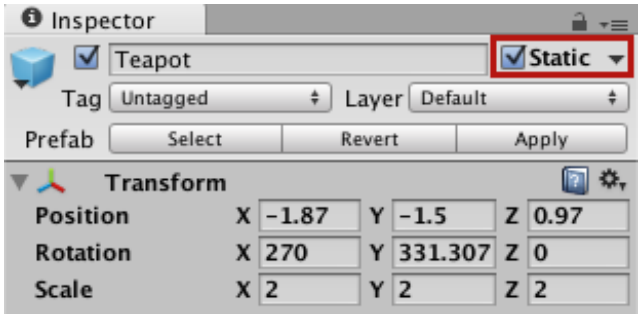


图16.7 把物体标志为Static

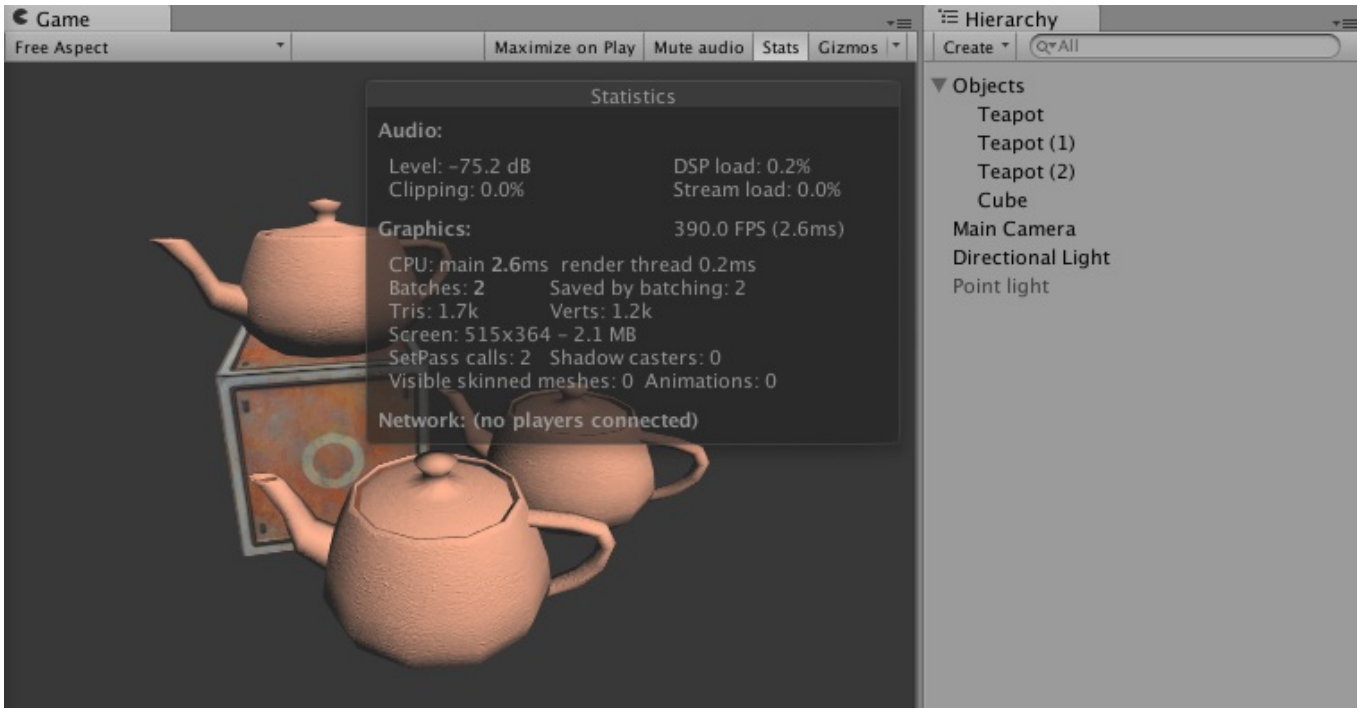


图16.8 静态批处理

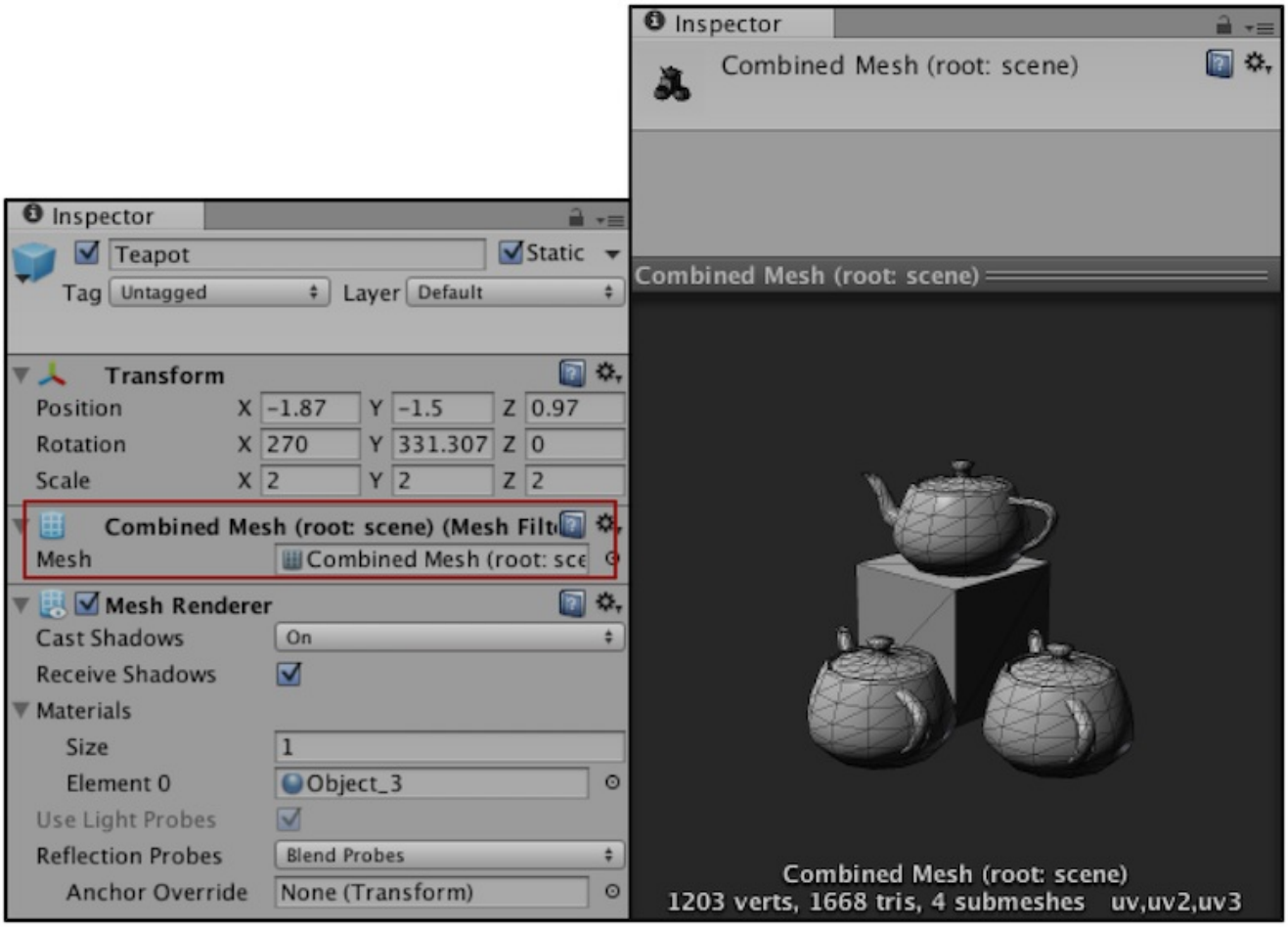


图16.9 静态批处理中Unity会合并所有被标识为“Static”的物体

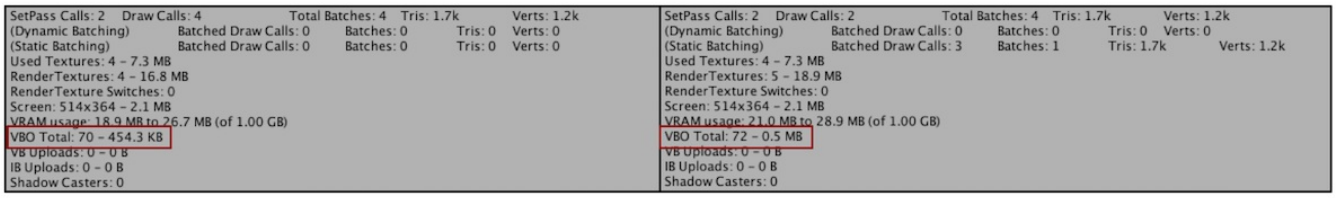


图16.10 静态批处理会占用更多的内存。左图：静态批处理前的渲染统计数据。右图：静态批处理后的渲染统计数据

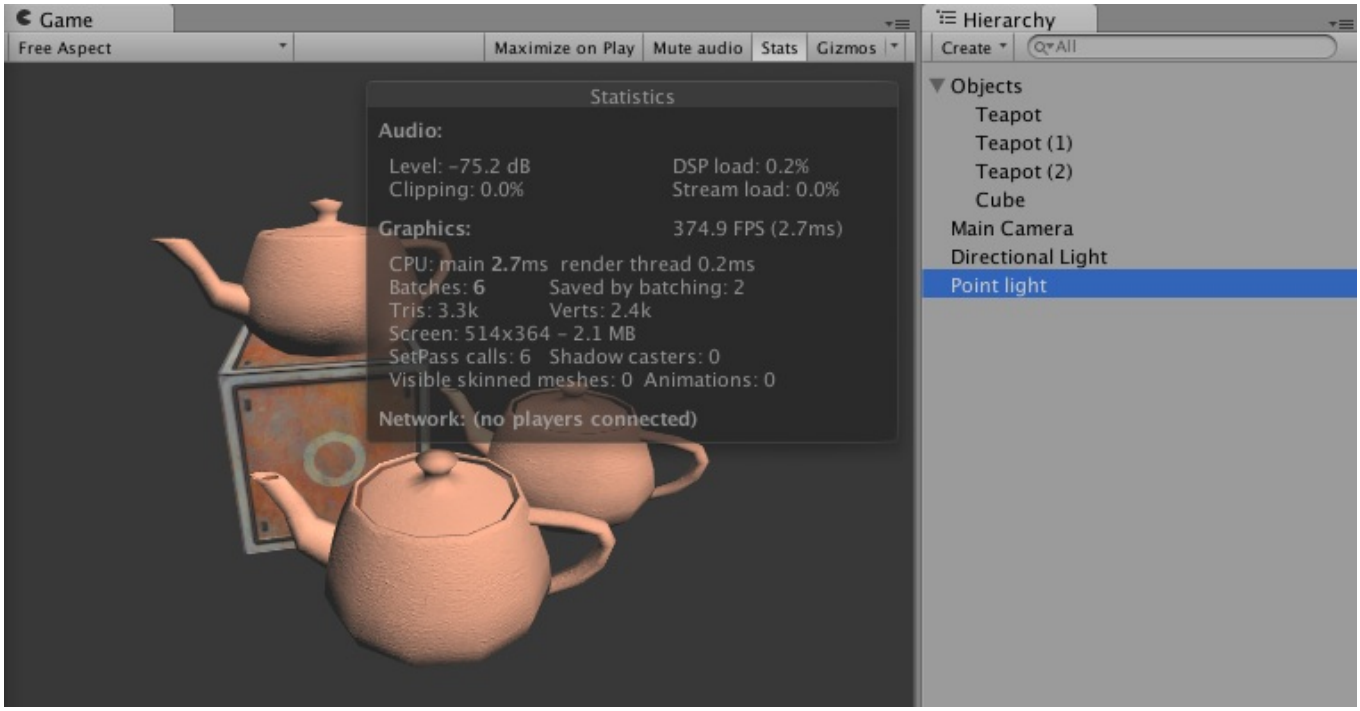


图16.11 处理其他逐像素光的Pass不会被静态批处理

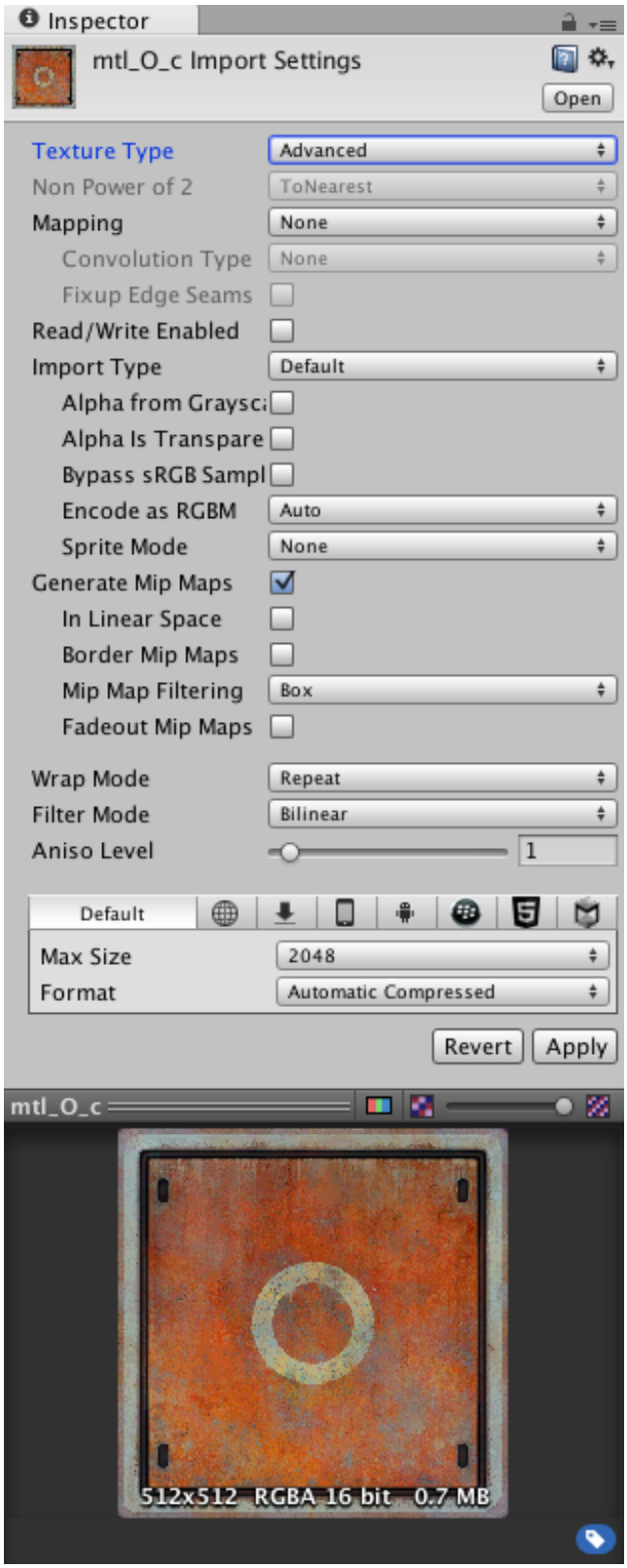


图16.12 Unity的高级纹理设置面板

第17章 Surface Shader探秘

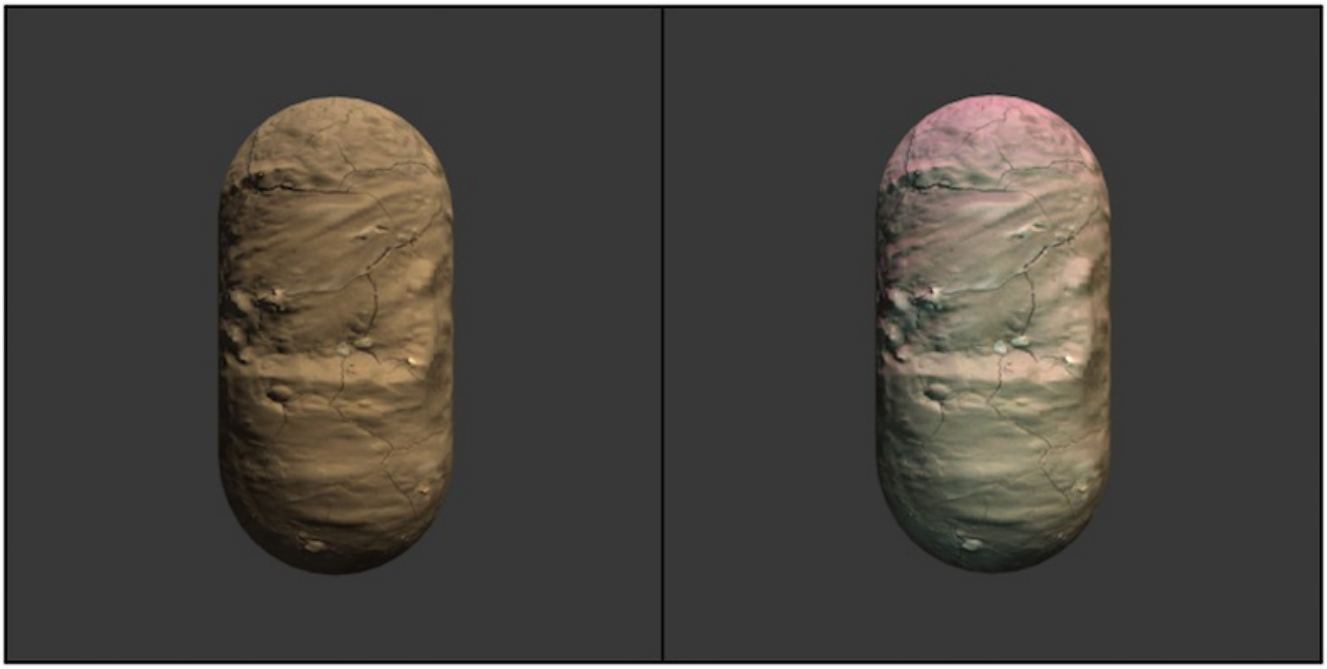


图17.1 表面着色器的例子。左图：在一个平行光下的效果。右图：添加了一个点光源（蓝色）和一个聚光灯（紫色）后的效果

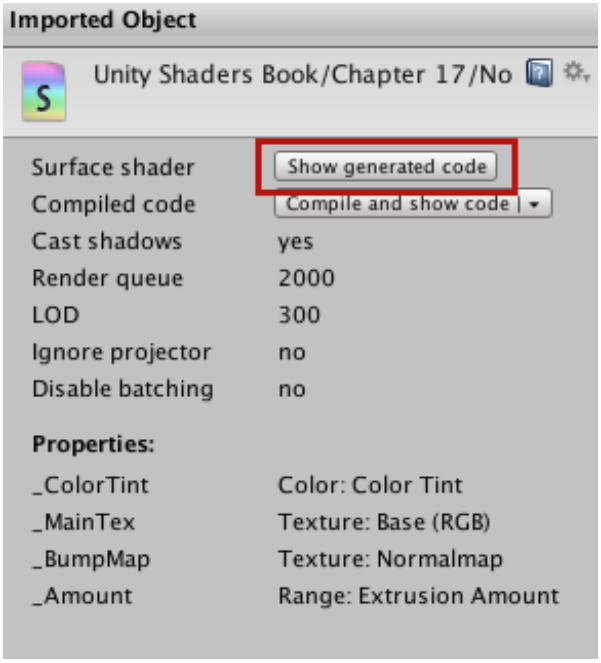


图17.2 查看表面着色器生成的代码

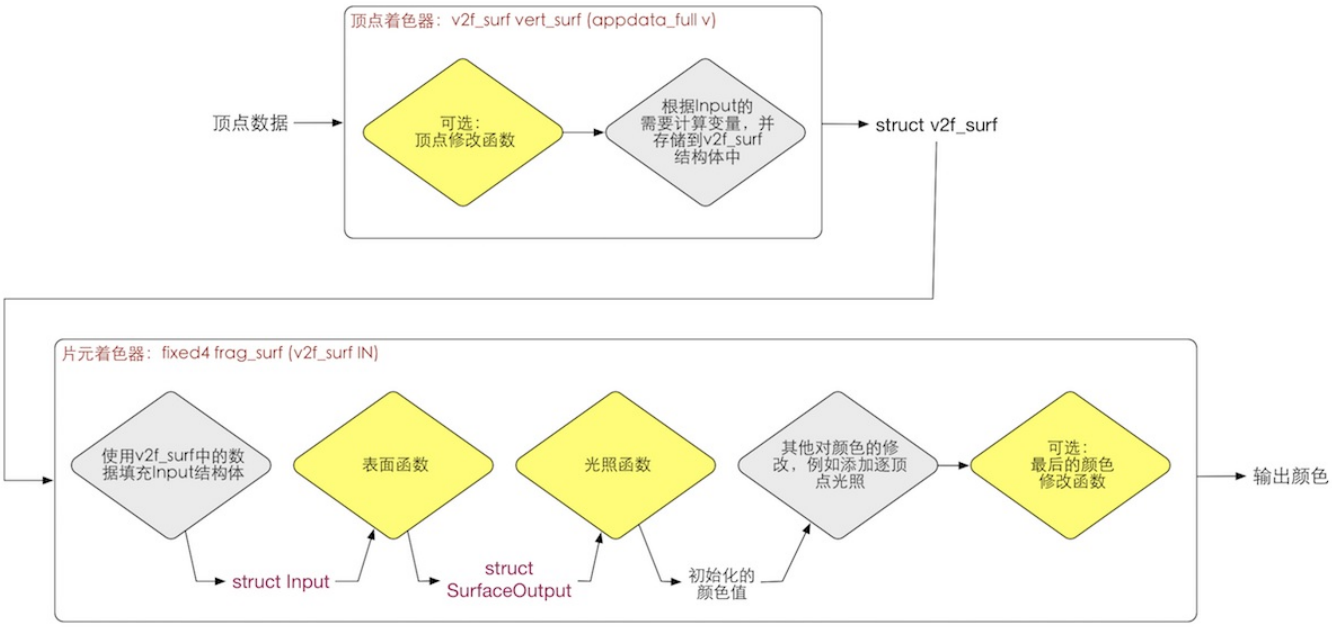


图17.3 表面着色器的渲染计算流水线。黄色：可以自定义的函数。灰色：Unity自动生成的计算步骤

图17.4 沿顶点法线对模型进行膨胀。左图：膨胀前。右图：膨胀后

第18章 基于物理的渲染

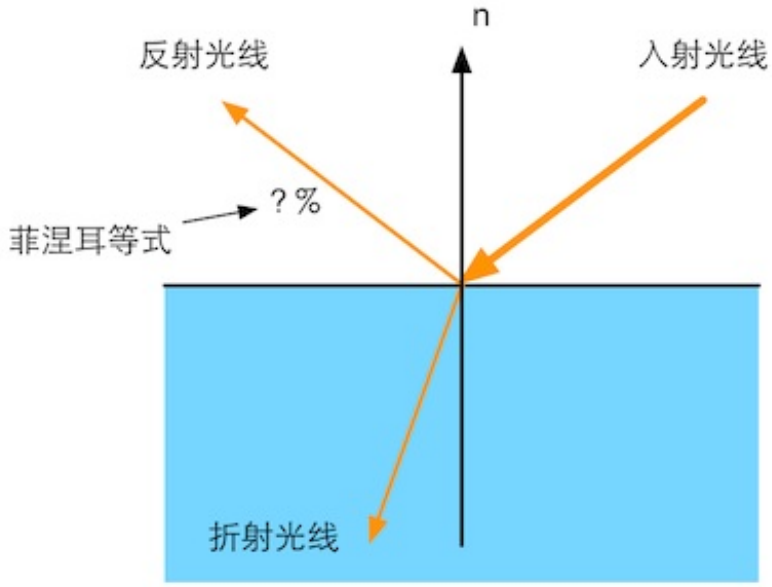


图18.1 在理想的边界处，折射率的突变会把光线分成两个方向

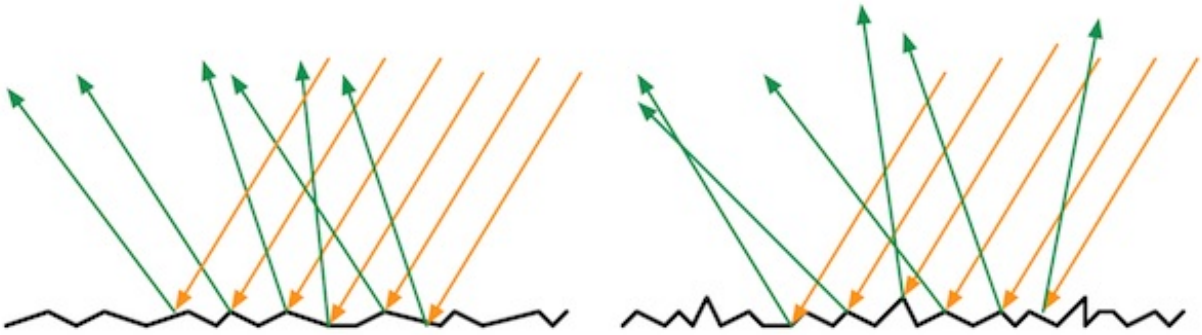


图18.2 左图：光滑表面的微平面的法线变化较小，反射光线的方向变化也更小。□右图：粗糙表面的微平面的法线变化较大，反射光线的方向变化也更大

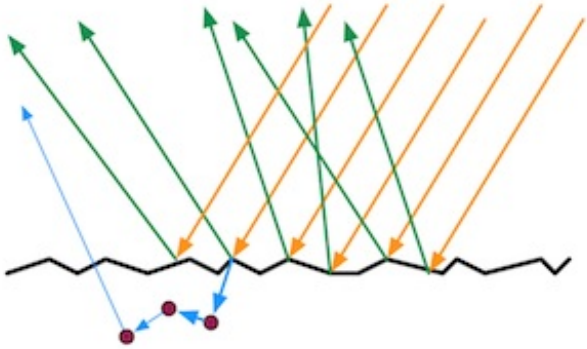


图18.3 微表面对光的折射。这些被折射的光中一部分被吸收，一部分又被散射到外部

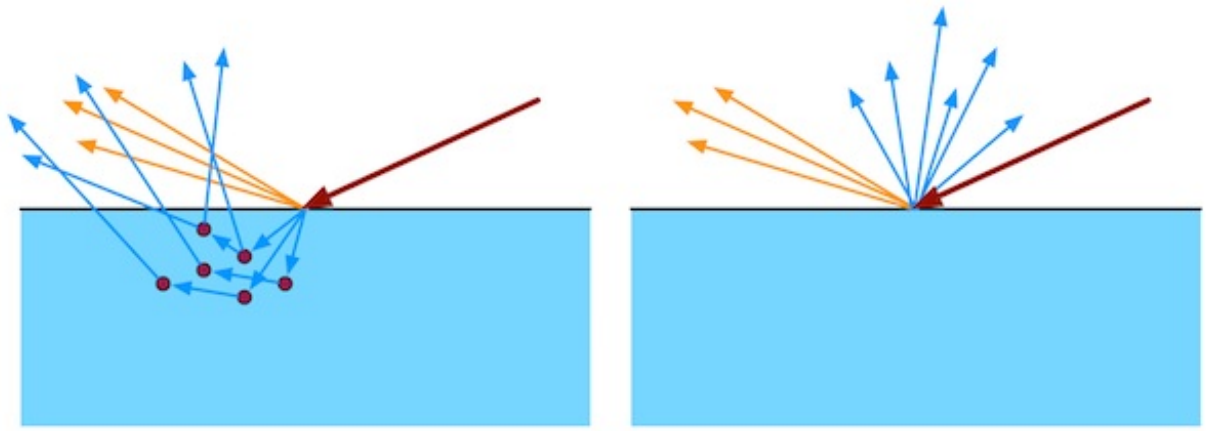


图18.4 次表面散射。左图：次表面散射的光线会从不同于入射点的位置射出。如果这些距离值小于需要被着色的像素大小，那么渲染就可以完全在局部完成（右图）。否则，就需要使用次表面散射渲染技术

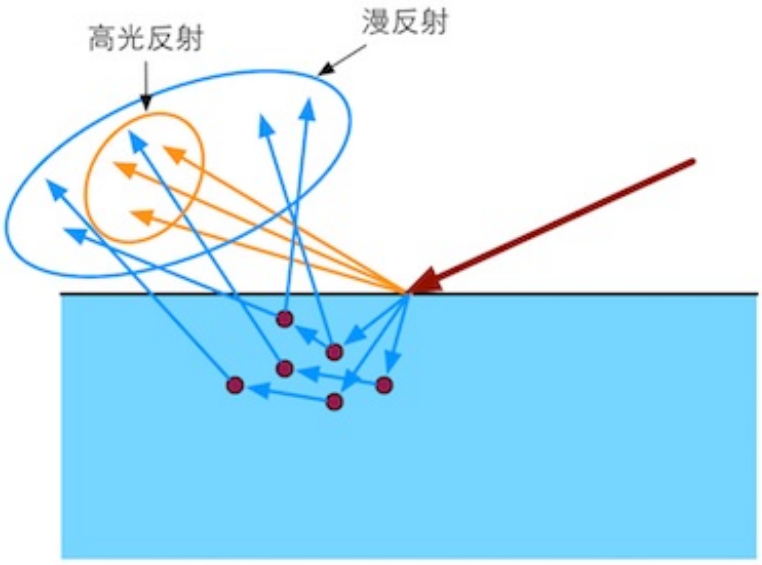


图18.5 BRDF描述的两种现象。高光反射部分用于描述反射，漫反射部分用于描述次表面散射

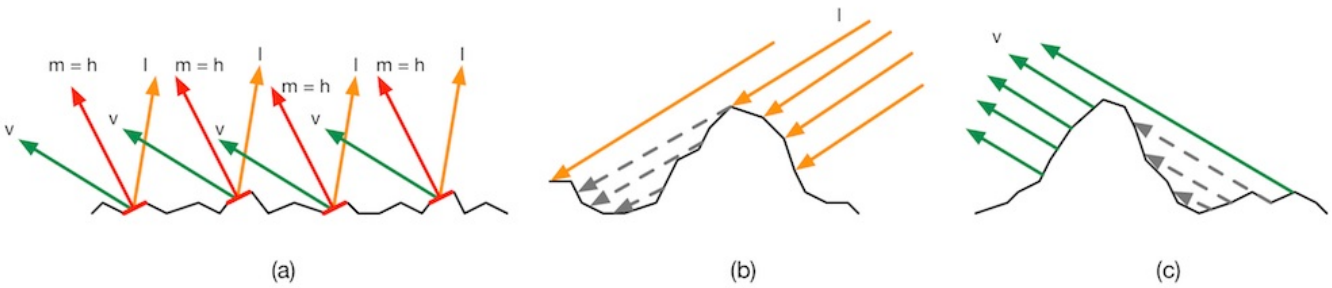


图18.6 （a）那些m=h的微面元会恰好把入射光从I反射到v上，只有这部分微面元才可以添加到BRDF的计算中。（b）一部分满足（a）的微面元会在I方向上被其他微面元遮挡住，它们不会接受到光照，因此会形成阴影。（c）还有一部分满足（a）的微面元会在反射方向v上被其

他微面元挡住，因此，这部分反射光也不会被看到

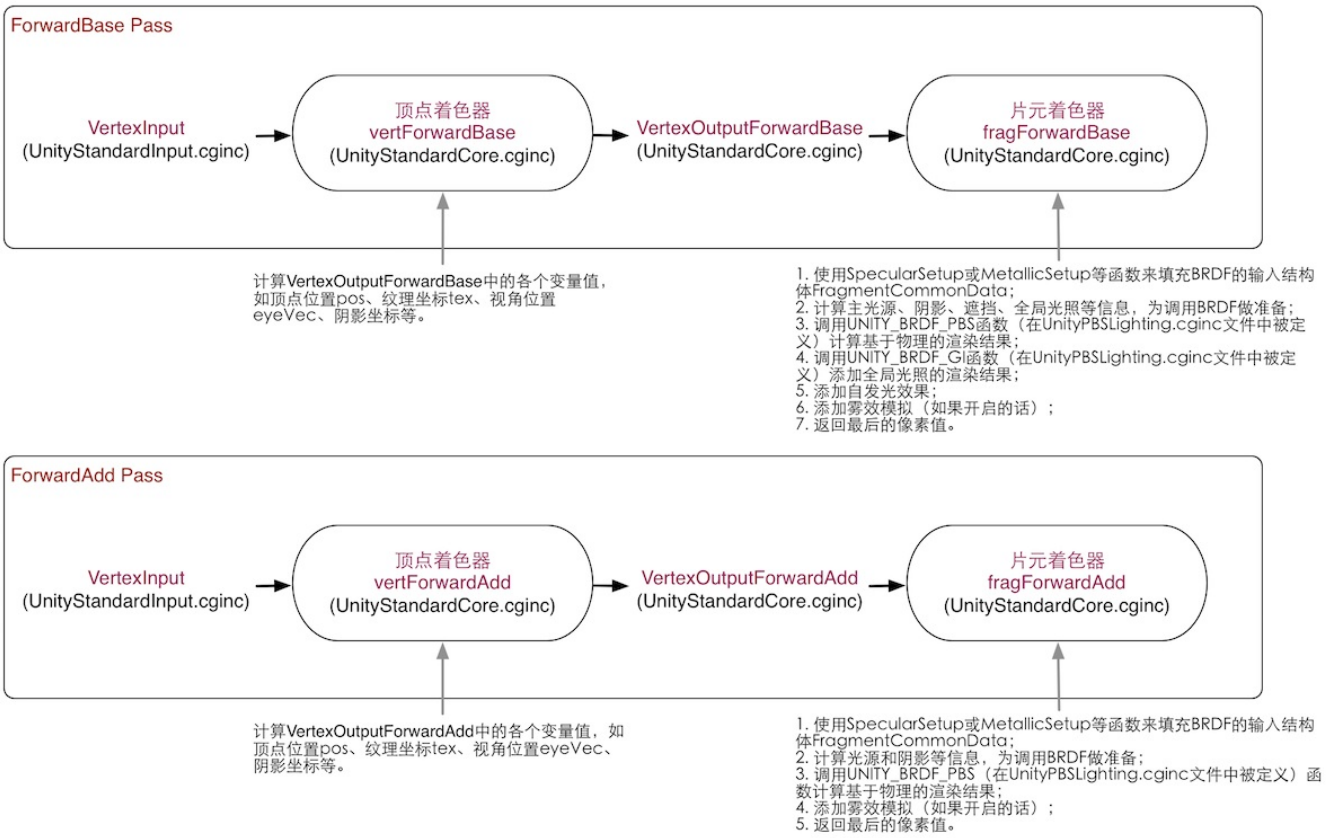


图18.7 Standard Shader中前向渲染路径使用的Pass（简化版本的PBS使用了VertexOutputBaseSimple等结构体来代替相应的结构体）

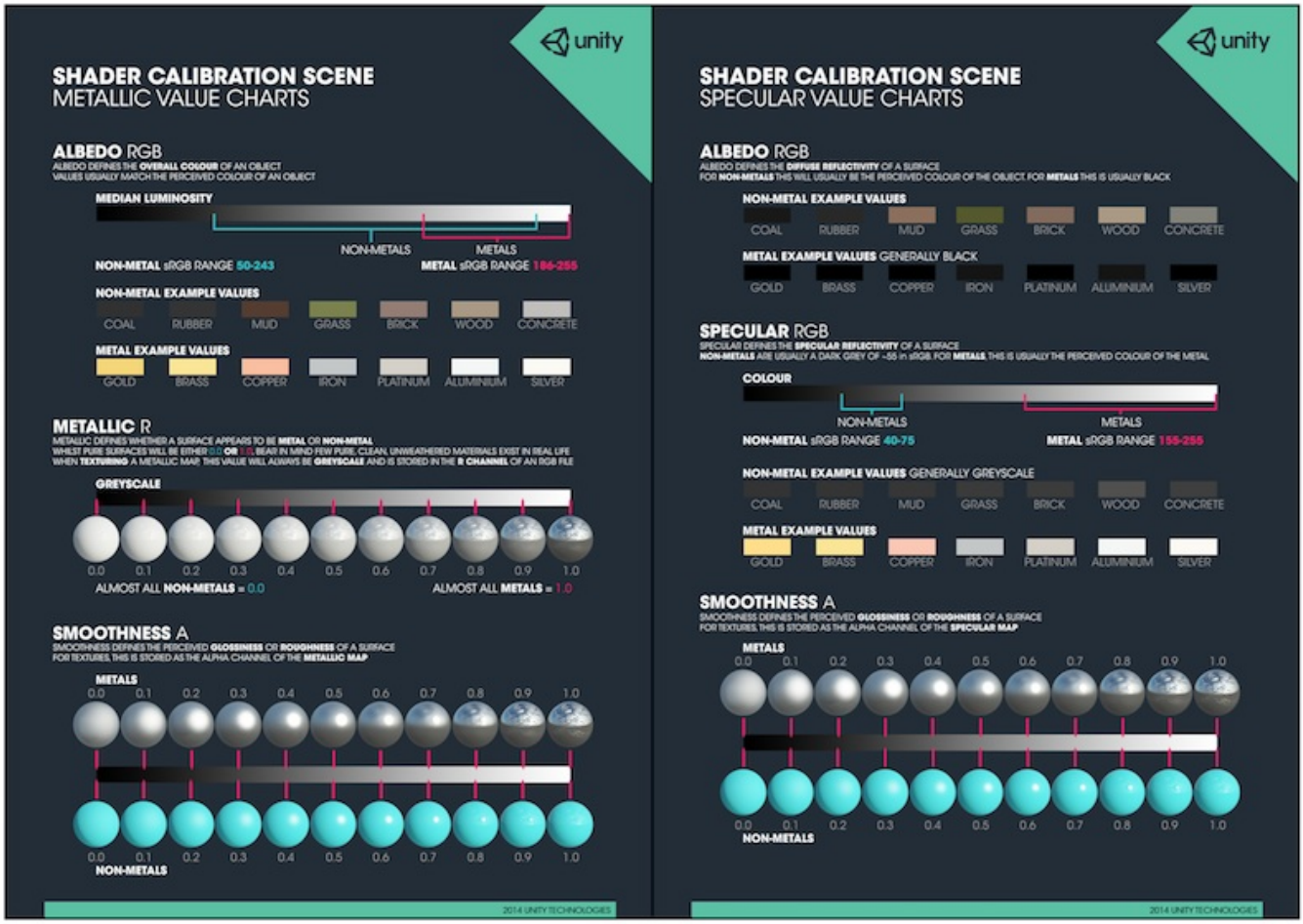


图18.8 Unity提供的校准表格。左图：金属工作流□使用的校准表格。右图：高光反射工作流使用的校准表格

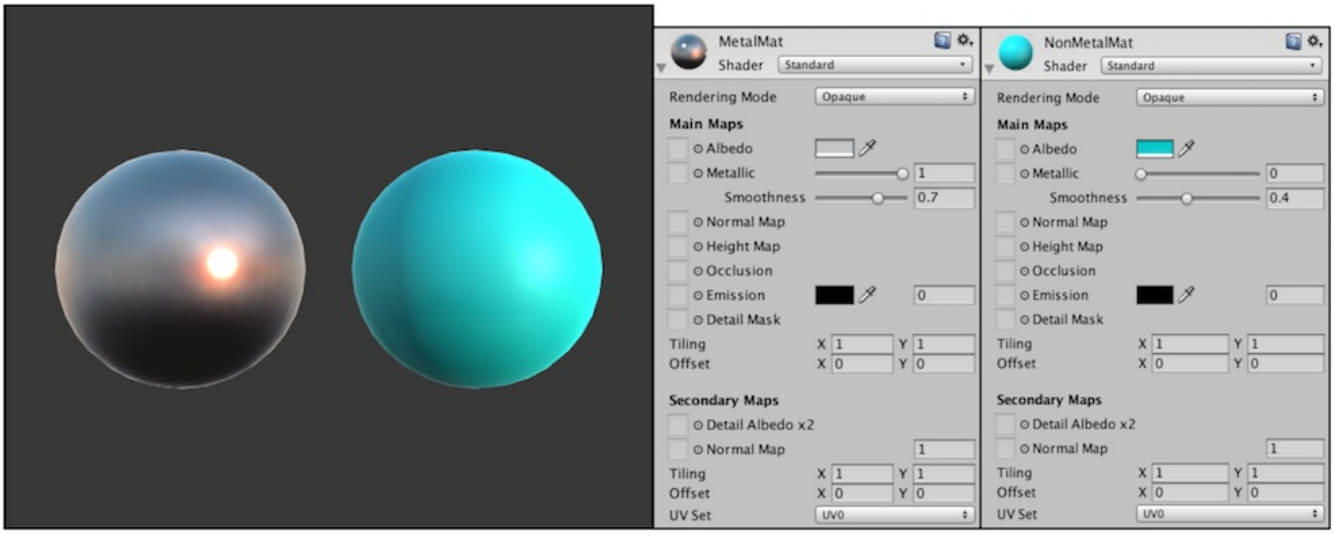


图18.9 使用金属工作流来实现不同类型的材质。左边的球体：金属材质。右边的球体：塑料材质

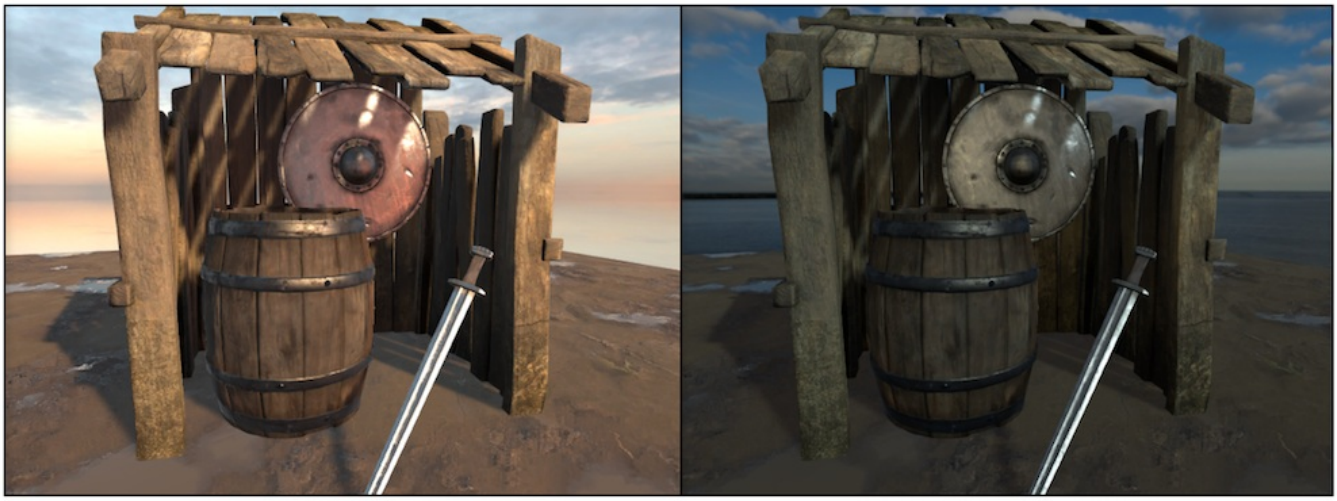


图18.10 在Unity 5中使用基于物理的渲染技术，场景在不同光照下的渲染结果

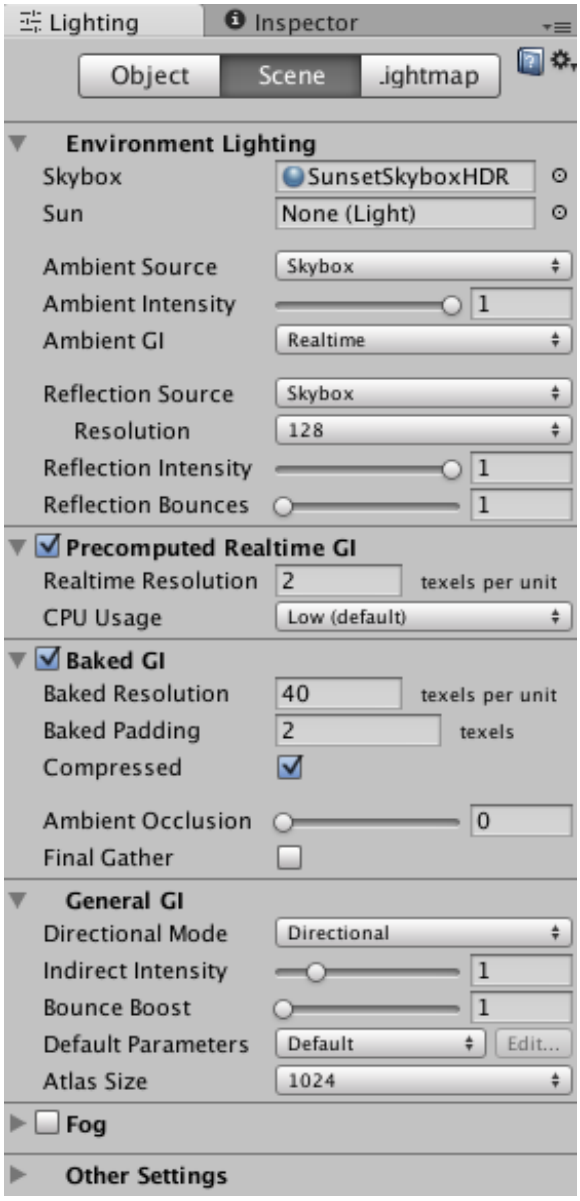


图18.11 光照面板下的Scene标签页

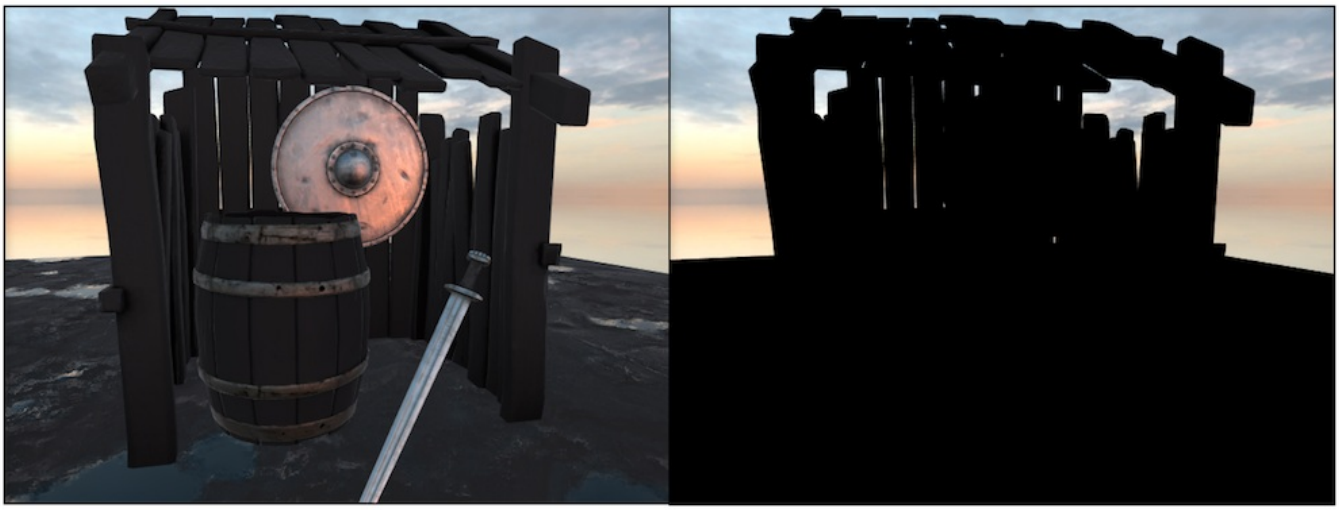


图18.12 左图：当关闭场景中的所有光源并把环境光照强度设为0后，使用了Standard Shader的物体仍然具有光照效果。右图：在左图的基础上，把反射源设置为空，使得物体不接受任何默认的反射信息

图18.13 使用的平行光

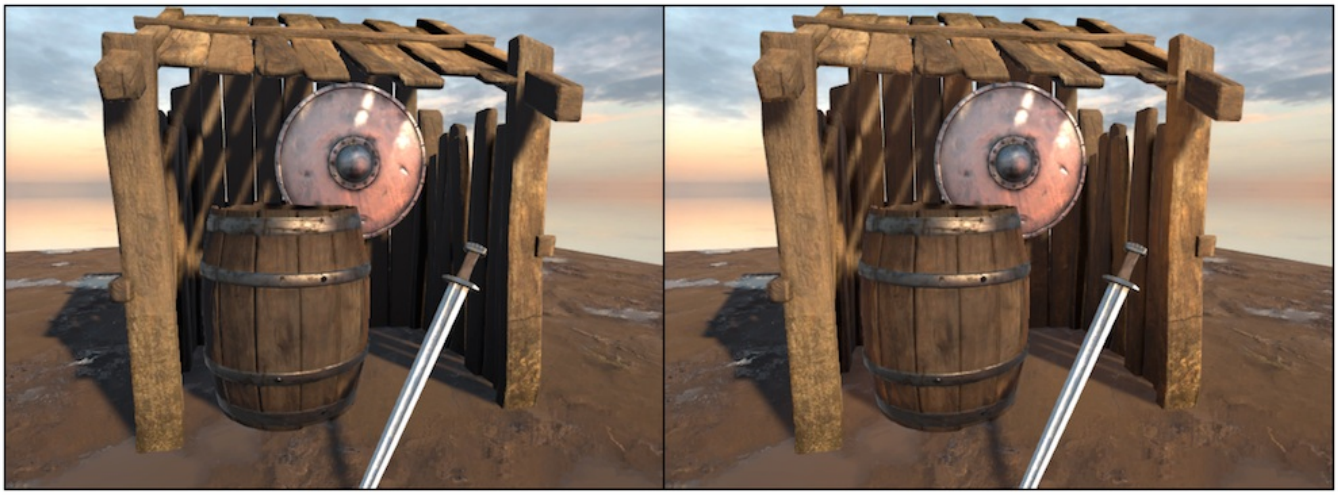


图18.14 左图：将Bounce Intensity设置为0，物体不再受到间接光照的影响，木屋内阴影部分的可见细节很少。右图：将Bounce Intensity设为8，阴影部分的细节更加清楚

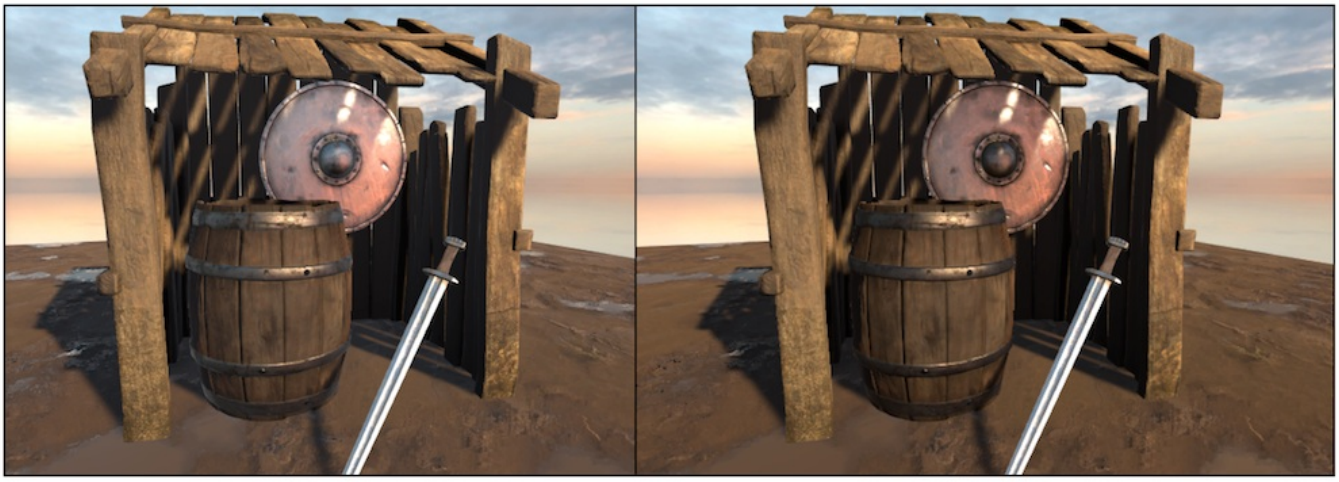


图18.15 左图：未使用反射探针。右图：在场景中放置了两个反射探针，注意墙上的盾牌与左图的差别

图18.16 使用反射探针实现相互反射的效果

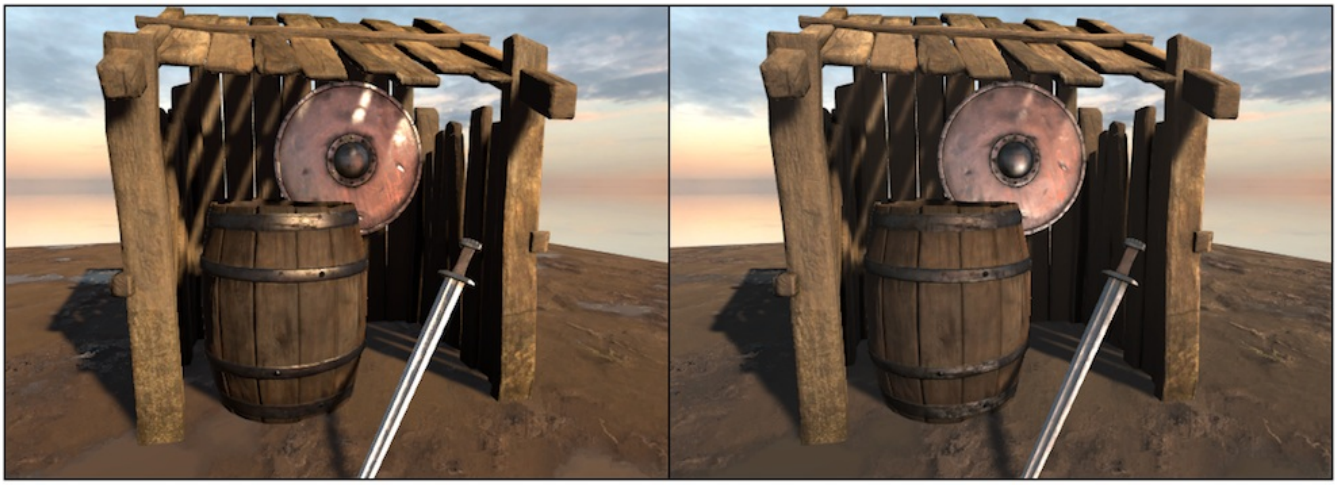


图18.17 左图：在线性空间下的渲染结果。右图：在伽马空间下的渲染结果

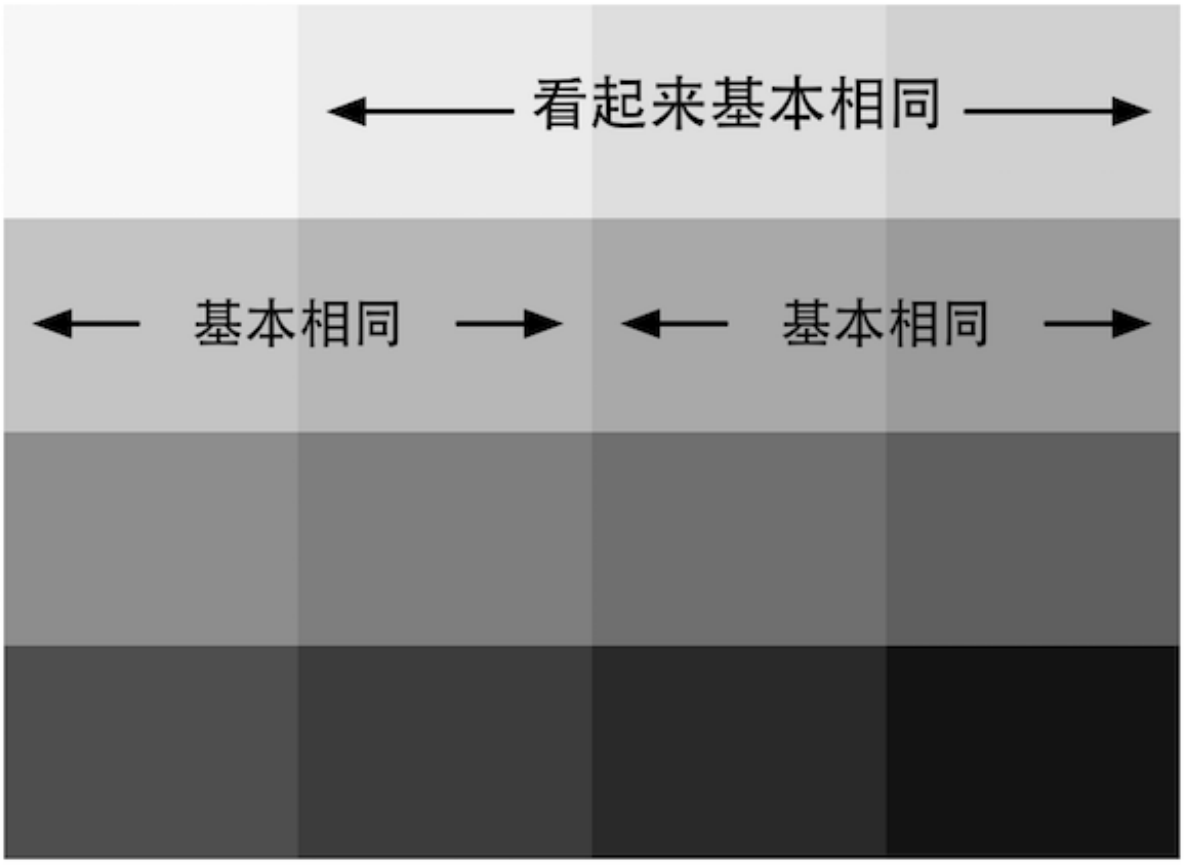


图18.18 人眼更容易感知暗部区域的变换，而对较亮区域的变化比较不敏感

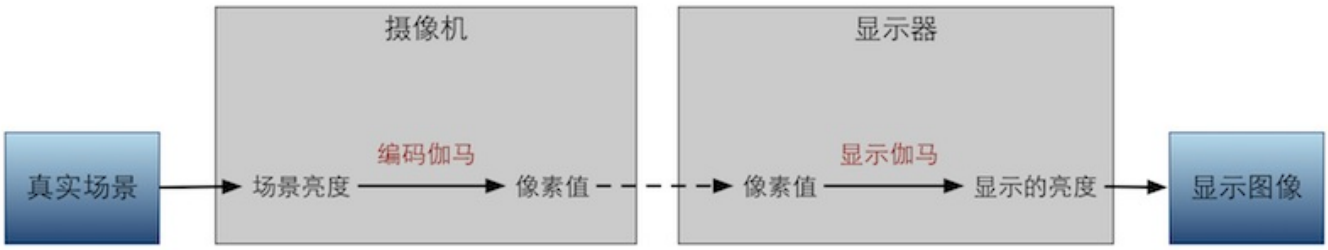


图18.19 编码伽马和显示伽马

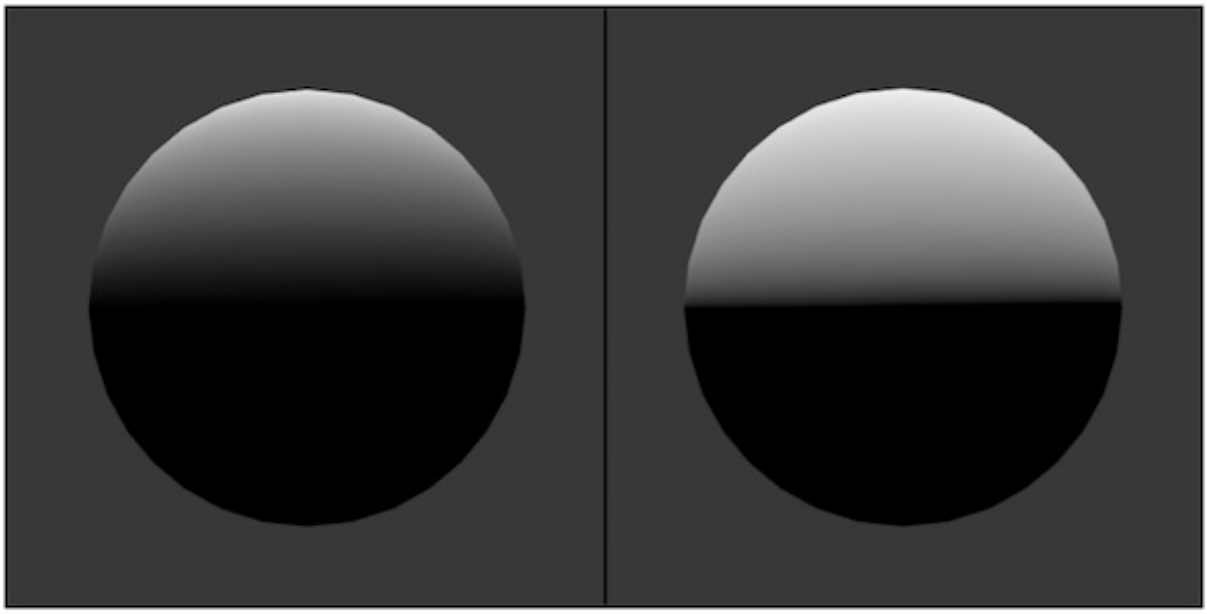


图18.20 左图：伽马空间下的渲染结果。右图：线性空间下的渲染结果

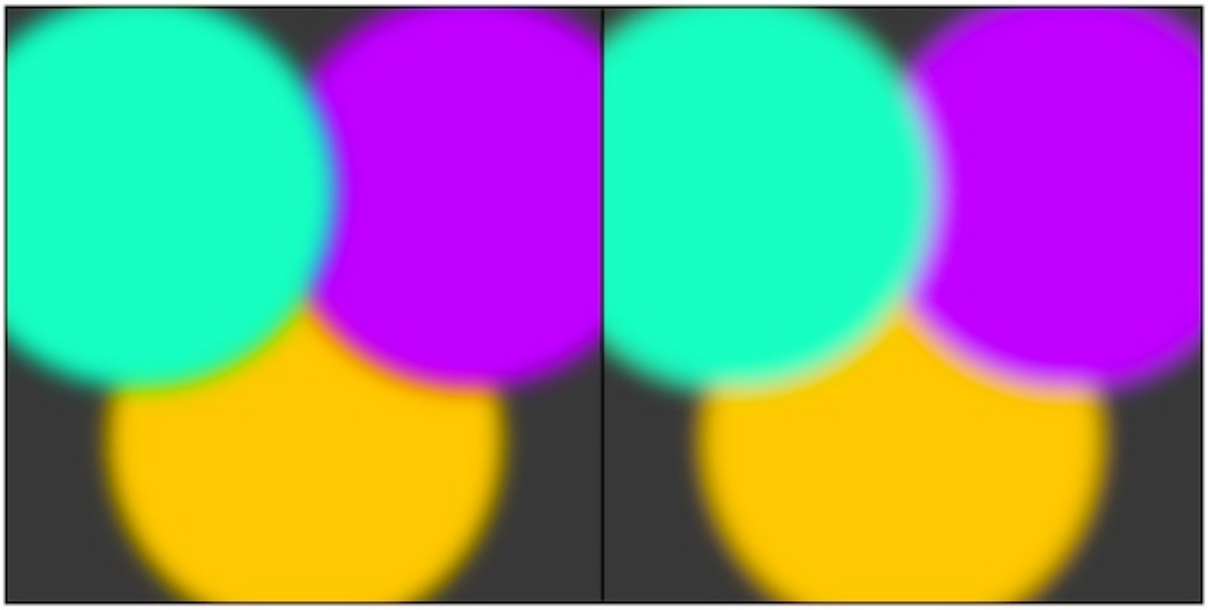


图18.21 左图：伽马空间下的混合结果。右图：线性空间下的混合结果

第19章 Unity 5更新了什么

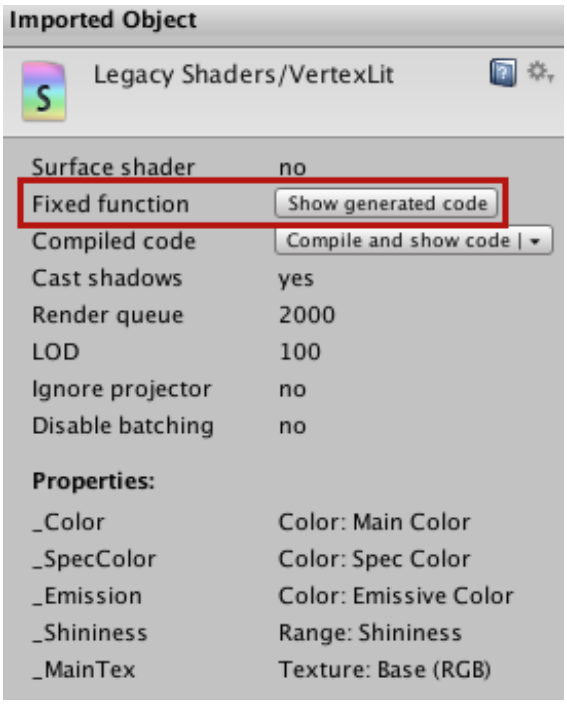


图19.1 在shader的导入面板中，单击图中按钮可查看Unity为该固定管线着色器生成的顶点/片元着色器代码