**过程式网格程序设计表单**

创建一个点的网格。

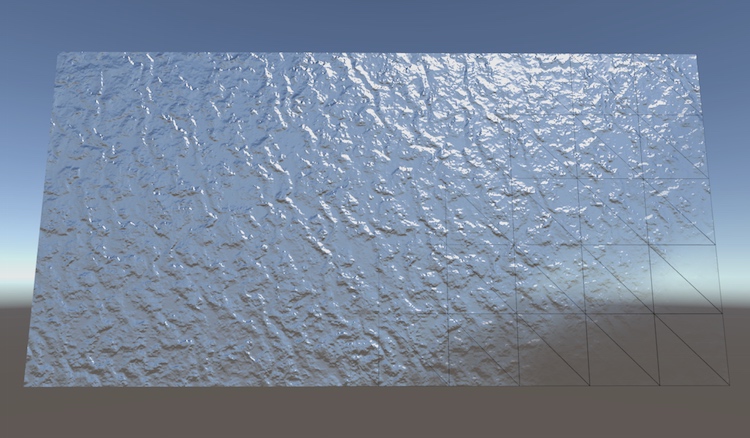
使用协同程序来分析它们的位置。

用三角形定义一个曲面。

自动生成法线。

添加纹理坐标和切线。

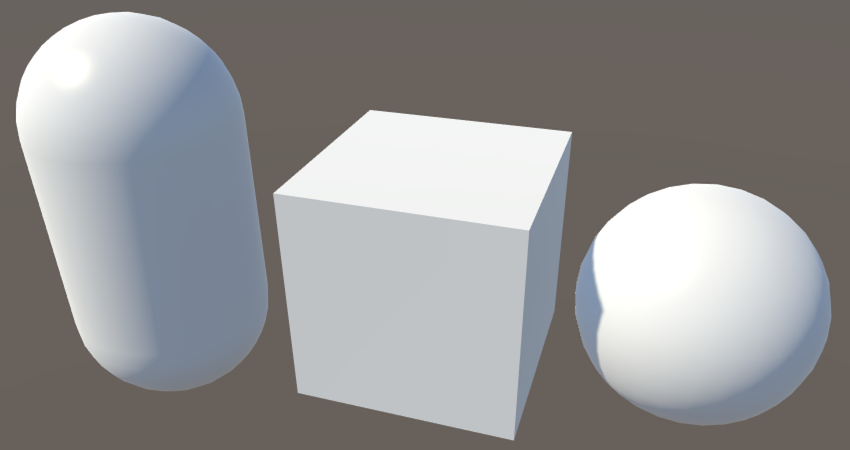
在本教程中，我们将创建一个简单的顶点和三角形网格。本教程假设您熟悉Unity脚本的基础知识。这些基本知识见时钟。构造分形提供了协同程序的介绍。本教程是为Unity 5.0.1及以上版本编写的。

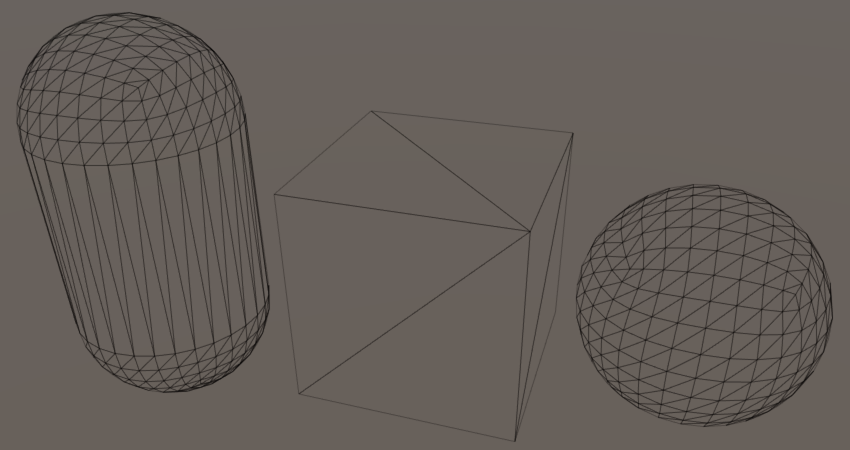


复杂的外表下隐藏着简单的几何图形。

**1渲染的东西**

如果你想在Unity中可视化一些东西，你可以使用网格。它可以是从另一个程序导出的3D模型。它可以是一个程序生成的网格。它可以是精灵，Ul元素，或者粒子系统，Unity也可以使用网格。甚至屏幕效果都是用网格渲染的什么是网格?从概念上讲，网格是图形硬件用来绘制复杂图形的一种构造。它至少包含一个定义3D空间中的点的顶点集合，以及一组连接这些点的三角形(最基本的2D形状)。三角形构成了网格所代表的任何表面。因为三角形是平面的，而且有直边，所以它们可以用来完美地呈现平面和直的事物，比如立方体的面。曲面或圆形表面只能用许多小三角形来近似。如果三角形看起来足够小——不超过一个像素——那么你就不会注意到这个近似值。通常这对于实时性能来说是不可行的，因此表面总是在某种程度上呈现锯齿状。

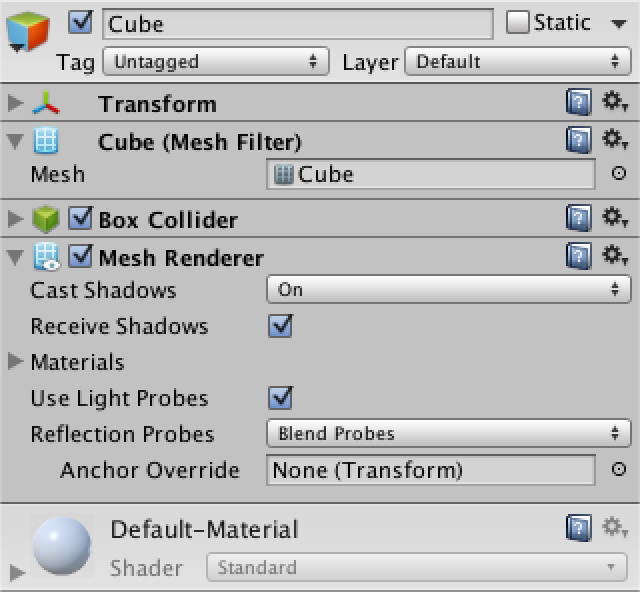




unity的默认胶囊，立方体，球体，阴影与线框。

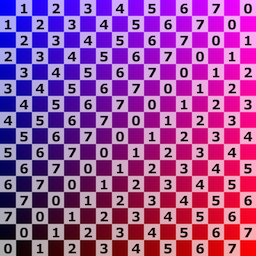
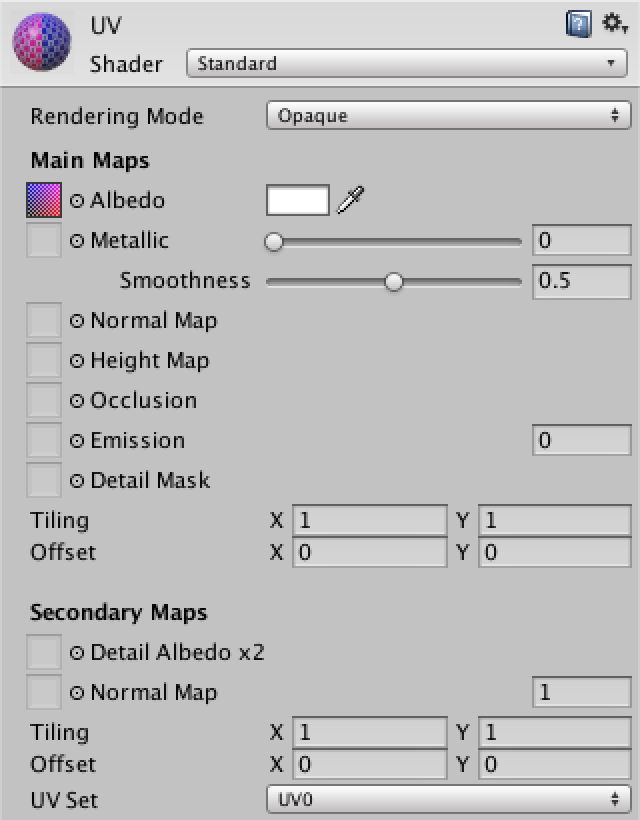
如何显示线框图?您可以选择场景视图在其工具栏左侧的显示模式。前三个选项是阴影线框和阴影线框。

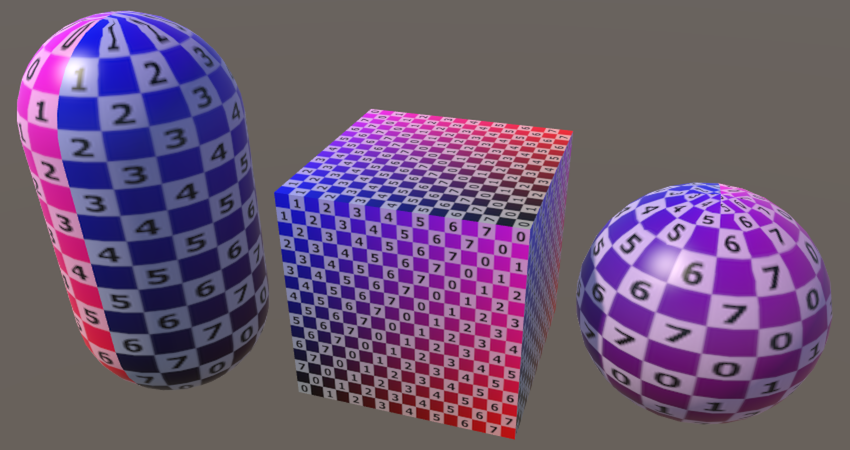
如果你想让一个游戏对象显示一个3D模型，它需要有两个组件。第一个是mesh filter（滤网）。此组件包含对要显示的网格的引用。第二个是mesh renderer（网格渲染器）。您可以使用它来配置网格的呈现方式。应该使用哪种材质，它是否应该投射或接收阴影，等等。



为什么会有一系列的材料?一个网格渲染器可以有多个材质。这主要用于渲染具有多个独立三角形集的网格，称为子网格。这些主要用于导入的3D模型，本教程中不会介绍。

你可以通过调整网格的材质来完全改变它的外观。Unity的默认材质是纯白色。你可以通过创建一个新的material来替换它（*Assets / Create / Material* ），并拖动到你的游戏对象。新材质默认使用Unity的标准着色器，这给了你一组控制来调整你的表面在视觉上的表现。一个快速的方法来添加大量的细节到您的网格是通过提供一个反照率地图。这是一种纹理，代表了一种材料的基本颜色。当然，我们需要知道如何把这个纹理投射到网格的三角形上。这是通过向顶点添加2D纹理坐标来实现的。纹理空间的两个维度被称为U和V，这就是为什么它们被称为UV坐标这些坐标通常位于(0,0)和(1,1)之间，它们覆盖了整个纹理。超出这个范围的坐标要么被夹住，要么导致平铺，这取决于纹理设置



**2创建一个顶点网格**

那么如何制作自己的网格呢?我们来看看，通过生成一个简单的矩形网格。网格将由单位长度的正方形块(四等分)组成。创建一个新的c#脚本，并将其转换为一个具有水平和垂直大小的网格组件。

**using** UnityEngine;

**using** System.Collections;

**public** **class** **Grid** : [**MonoBehaviour**](http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.html) {

**public** **int** xSize, ySize;

}

当我们把这个组件添加到一个游戏对象，我们需要给它一个网格过滤器和网格渲染器。我们可以添加一个属性到类中，让Unity自动为我们添加它们。

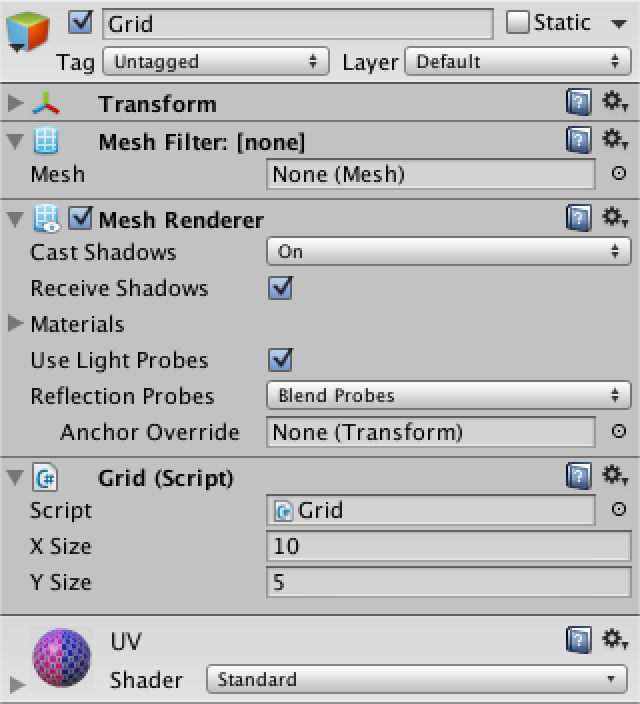
[**[RequireComponent](http://docs.unity3d.com/Documentation/ScriptReference/RequireComponent.html)**(**typeof**(**[MeshFilter](http://docs.unity3d.com/Documentation/ScriptReference/MeshFilter.html)**), **typeof**(**[MeshRenderer](http://docs.unity3d.com/Documentation/ScriptReference/MeshRenderer.html)**))]

**public** **class** **Grid** : [**MonoBehaviour**](http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.html) {

**public** **int** xSize, ySize;

}

现在您可以创建一个新的空游戏对象，并将网格组件添加到其中，它还将拥有其他两个组件。设置渲染器的材质，保持滤镜的网格未定义。我将网格的大小设置为10×5。



当对象被显示，会调用

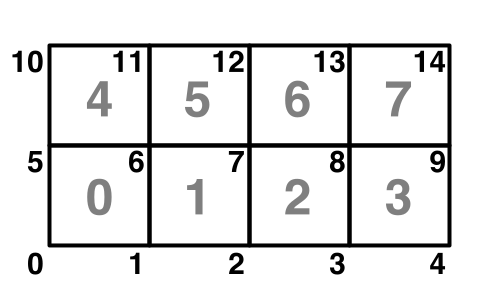
**private** **void** [**Awake**](http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.Awake.html) () {

Generate();

}

让我们先关注顶点位置，稍后再讨论三角形。我们需要一个三维向量数组来存储这些点。顶点的数量取决于网格的大小。我们需要在每个四边形的角上都有一个顶点，但是相邻的四边形可以共享同一个顶点。所以我们需要一个比每个维中有瓦片更多的顶点

(#x+1)(#y+1)



顶点和四分指标为4×2的网格。

**private** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)[] vertices;

**private** **void** Generate () {

vertices = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)[(xSize + 1) \* (ySize + 1)];

}

让我们把这些顶点形象化，这样我们就可以检查它们的位置是否正确。我们可以通过添加onprawGizmos方法并在场景视图中为每个顶点绘制一个小的黑色球体来做到这一点。

**private** **void** [**OnDrawGizmos**](http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.OnDrawGizmos.html) () {

[**Gizmos**](http://docs.unity3d.com/Documentation/ScriptReference/Gizmos.html).color = [**Color**](http://docs.unity3d.com/Documentation/ScriptReference/Color.html).black;

**for** (**int** i = 0; i < vertices.Length; i++) {

[**Gizmos**](http://docs.unity3d.com/Documentation/ScriptReference/Gizmos.html).DrawSphere(vertices[i], 0.1f);

}

}

这将产生错误，当我们不在播放模式，因为onDrawGizmos方法也被调用时，Unity在编辑模式，当我们没有任何顶点。要防止此错误，请检查数组是否存在，如果不存在则跳出方法。

**private** **void** [**OnDrawGizmos**](http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.OnDrawGizmos.html) () {

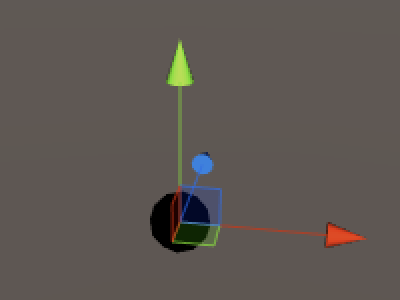
**if** (vertices == **null**) {

**return**;

}

…

}



一个小物件

在游戏模式中，我们在原点只能看到一个球体。这是因为我们还没有给顶点定位，所以它们都在那个位置重叠。我们必须使用双循环遍历所有位置

**private** **void** Generate () {

vertices = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)[(xSize + 1) \* (ySize + 1)];

**for** (**int** i = 0, y = 0; y <= ySize; y++) {

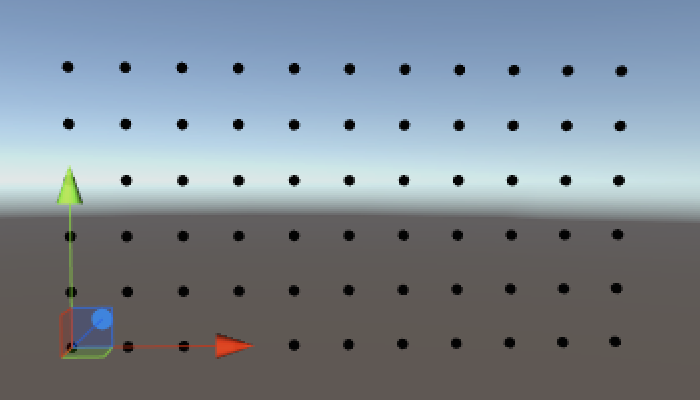
**for** (**int** x = 0; x <= xSize; x++, i++) {

vertices[i] = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)(x, y);

}

}

}



一个顶点网格。

我们现在看到了顶点，但是它们的排列顺序是不可见的。我们可以用颜色来表示，但是我们也可以通过使用协同程序来减慢这个过程。这就是为什么我包括使用系统

**private** **void** [**Awake**](http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.Awake.html) () {

StartCoroutine(Generate());

}

**private** [**IEnumerator**](http://social.msdn.microsoft.com/search/en-us?query=IEnumerator) Generate () {

[**WaitForSeconds**](http://docs.unity3d.com/Documentation/ScriptReference/WaitForSeconds.html) wait = **new** [**WaitForSeconds**](http://docs.unity3d.com/Documentation/ScriptReference/WaitForSeconds.html)(0.05f);

vertices = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)[(xSize + 1) \* (ySize + 1)];

**for** (**int** i = 0, y = 0; y <= ySize; y++) {

**for** (**int** x = 0; x <= xSize; x++, i++) {

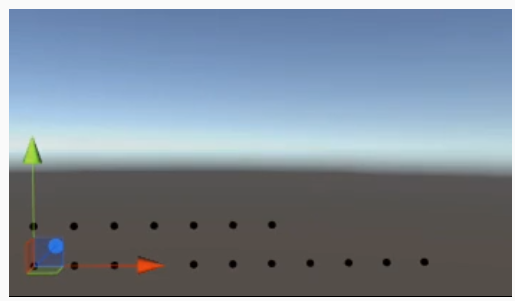
vertices[i] = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)(x, y);

**yield** **return** wait;

}

}

}



**3创建网格**

现在我们知道顶点的位置是正确的，我们可以处理实际的网格。除了在我们自己的组件中保存对它的引用外，我们还必须将它分配给mesh过滤器。一旦我们处理了顶点，我们就可以把它们放到网格里

**private** [**Mesh**](http://docs.unity3d.com/Documentation/ScriptReference/Mesh.html) mesh;

**private** [**IEnumerator**](http://social.msdn.microsoft.com/search/en-us?query=IEnumerator) Generate () {

[**WaitForSeconds**](http://docs.unity3d.com/Documentation/ScriptReference/WaitForSeconds.html) wait = **new** [**WaitForSeconds**](http://docs.unity3d.com/Documentation/ScriptReference/WaitForSeconds.html)(0.05f);

[**GetComponent**](http://docs.unity3d.com/Documentation/ScriptReference/Component.GetComponent.html)<[**MeshFilter**](http://docs.unity3d.com/Documentation/ScriptReference/MeshFilter.html)>().mesh = mesh = **new** [**Mesh**](http://docs.unity3d.com/Documentation/ScriptReference/Mesh.html)();

mesh.name = "Procedural Grid";

vertices = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)[(xSize + 1) \* (ySize + 1)];

…

mesh.vertices = vertices;

}



运行 网格出现。

我们现在有一个网格在播放模式，但它还没有出现，因为我们没有给它任何三角形。三角形是通过顶点索引数组定义的。由于每个三角形有三个点，三个连续的指标描述了一个三角形。我们从一个三角形开始

**private** [**IEnumerator**](http://social.msdn.microsoft.com/search/en-us?query=IEnumerator) Generate () {

…

**int**[] triangles = **new** **int**[3];

triangles[0] = 0;

triangles[1] = 1;

triangles[2] = 2;

mesh.triangles = triangles;

}

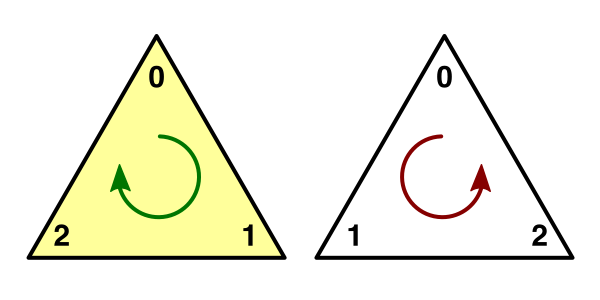
现在我们有一个三角形，但是我们用的三个点都在一条直线上。这就产生了一个不可见的退化三角形。前两个顶点是好的，但然后我们应该跳到下一行的第一个顶点

triangles[0] = 0;

triangles[1] = 1;

triangles[2] = xSize + 1;

这确实给了我们一个三角形，但它只能从一个方向上看。在这种情况下，它只在观察Z轴的相反方向时可见。你可能需要旋转视图才能看到它三角形的哪边是可见的由顶点指数的方向决定。默认情况下，如果它们按顺时针方向排列，则三角形被认为是正对的，而可视的逆时针三角形则被丢弃，因此我们不需要花费时间来绘制对象的内部，因为通常情况下，无论如何都不希望看到对象的内部。



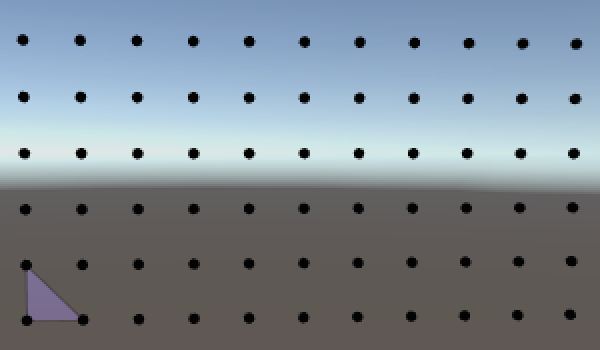
三角形的两条边。

所以当我们向下看Z轴时，为了让三角形出现，我们必须改变它的顶点被遍历的顺序。我们可以通过交换最后两个指标来实现。

triangles[0] = 0;

triangles[1] = xSize + 1;

triangles[2] = 1;



第一个三角形。

现在我们有一个三角形，它覆盖了第一个网格的一半。为了覆盖整个瓦片，我们需要的是第二个三角形。

**int**[] triangles = **new** **int**[6];

triangles[0] = 0;

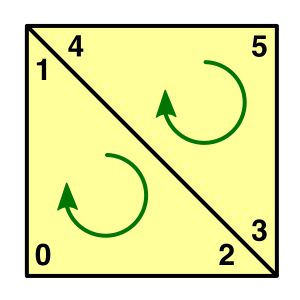
triangles[1] = xSize + 1;

triangles[2] = 1;

triangles[3] = 1;

triangles[4] = xSize + 1;

triangles[5] = xSize + 2;



由两个三角形组成的四边形。

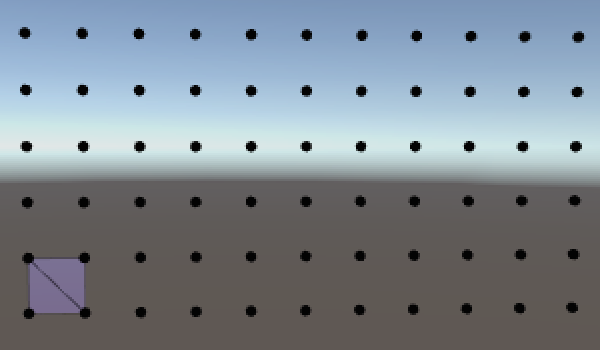
由于这些三角形共享两个顶点，我们可以将其缩减为四行代码，只显式地提及每个顶点索引一次

triangles[0] = 0;

triangles[3] = triangles[2] = 1;

triangles[4] = triangles[1] = xSize + 1;

triangles[5] = xSize + 2;



第一个四边形

我们可以通过将它转换成一个循环来创建整个第一行的tiles。当我们遍历顶点和三角形指标时，我们必须同时跟踪这两个指标。让我们也把yield语句移动到这个循环中，这样我们就不必再等待顶点出现了。

**int**[] triangles = **new** **int**[xSize \* 6];

**for** (**int** ti = 0, vi = 0, x = 0; x < xSize; x++, ti += 6, vi++) {

triangles[ti] = vi;

triangles[ti + 3] = triangles[ti + 2] = vi + 1;

triangles[ti + 4] = triangles[ti + 1] = vi + xSize + 1;

triangles[ti + 5] = vi + xSize + 2;

**yield** **return** wait;

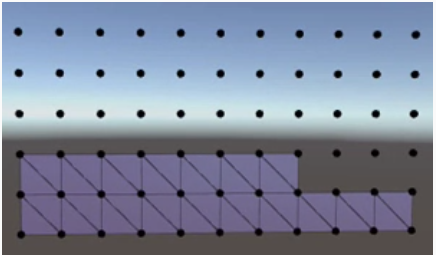
}

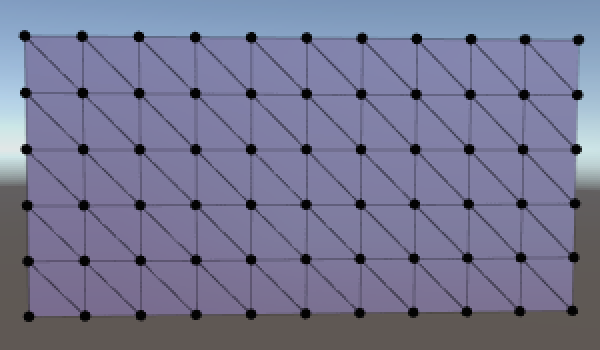
顶点gizmos现在立即出现，和三角形都出现在一次短暂的等待。为了看到一个接一个的块出现，我们必须在每次迭代中更新网格，而不是仅仅在循环之后

mesh.triangles = triangles;

**yield** **return** wait;

现在通过把单环变成双环来填充整个网格。注意，移动到下一行需要将顶点索引增加1，因为每一行比平铺多一个顶点。





填充整个网格。

如您所见，整个网格现在充满了三角形，一次一行。一旦你满意了，你可以删除所有的协同程序代码，这样网格将创建没有延迟。

**private** **void** [**Awake**](http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.Awake.html) () {

Generate();

}

**private** **void** Generate () {

[**GetComponent**](http://docs.unity3d.com/Documentation/ScriptReference/Component.GetComponent.html)<[**MeshFilter**](http://docs.unity3d.com/Documentation/ScriptReference/MeshFilter.html)>().mesh = mesh = **new** [**Mesh**](http://docs.unity3d.com/Documentation/ScriptReference/Mesh.html)();

mesh.name = "Procedural Grid";

vertices = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)[(xSize + 1) \* (ySize + 1)];

**for** (**int** i = 0, y = 0; y <= ySize; y++) {

**for** (**int** x = 0; x <= xSize; x++, i++) {

vertices[i] = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)(x, y);

}

}

mesh.vertices = vertices;

**int**[] triangles = **new** **int**[xSize \* ySize \* 6];

**for** (**int** ti = 0, vi = 0, y = 0; y < ySize; y++, vi++) {

**for** (**int** x = 0; x < xSize; x++, ti += 6, vi++) {

triangles[ti] = vi;

triangles[ti + 3] = triangles[ti + 2] = vi + 1;

triangles[ti + 4] = triangles[ti + 1] = vi + xSize + 1;

triangles[ti + 5] = vi + xSize + 2;

}

}

mesh.triangles = triangles;

}

**4生成额外的顶点数据**

我们的网格目前以一种特殊的方式发光。那是因为我们还没有给网格任何法线。默认的法线方向是(0,0,1)，与我们需要的方向完全相反。

法线是如何工作的?

法向量是垂直于曲面的向量。我们总是使用单位长度的法线它们指向表面的外面，而不是里面。法线可以用来确定光线射入表面的角度，如果有的话。如何使用它的细节取决于着色器。因为三角形总是平面的，所以没有必要提供关于法线的单独信息，但是，这样做我们可以作弊。实际上顶点没有法线，三角形有。通过将自定义法线附加到顶点，并在顶点之间插入三角形，我们可以假装我们有一个平滑的曲面，而不是一堆平面三角形。这种错觉是令人信服的，只要你不注意尖锐的轮廓的网格

法线是每个顶点定义的，所以我们必须填充另一个向量数组。或者，我们可以要求网格根据它的三角形计算出它自己的法线。这次让我们偷懒去做吧。

**private** **void** Generate () {

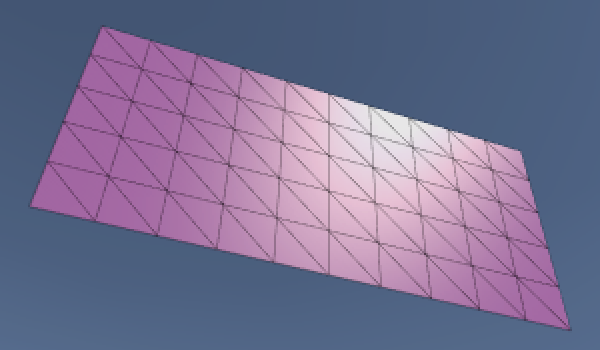
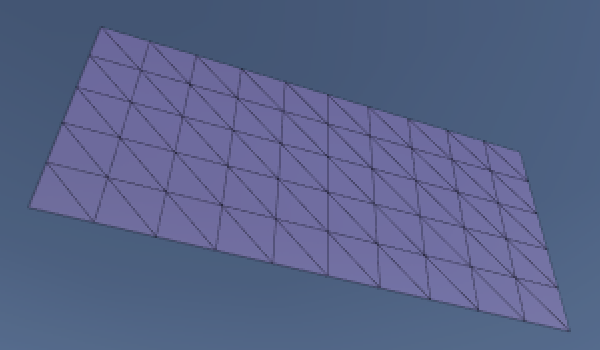
…

mesh.triangles = triangles;

mesh.RecalculateNormals();

}

如何重新计算法线?网格recculatenormals method计算每个顶点的法线，方法是找出哪些三角形与该顶点相连，确定这些平面三角形的法线，对它们求平均值，然后将结果归一化。



接下来是UV坐标。你可能已经注意到网格当前有一个统一的颜色，即使它使用的是一种具有反照率纹理的材质。这是有道理的，因为如果我们不提供UV坐标，它们都是零要使纹理适合我们的整个网格，只需将顶点的位置除以网格的维数。

vertices = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)[(xSize + 1) \* (ySize + 1)];

[**Vector2**](http://docs.unity3d.com/Documentation/ScriptReference/Vector2.html)[] uv = **new** [**Vector2**](http://docs.unity3d.com/Documentation/ScriptReference/Vector2.html)[vertices.Length];

**for** (**int** i = 0, y = 0; y <= ySize; y++) {

**for** (**int** x = 0; x <= xSize; x++, i++) {

vertices[i] = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)(x, y);

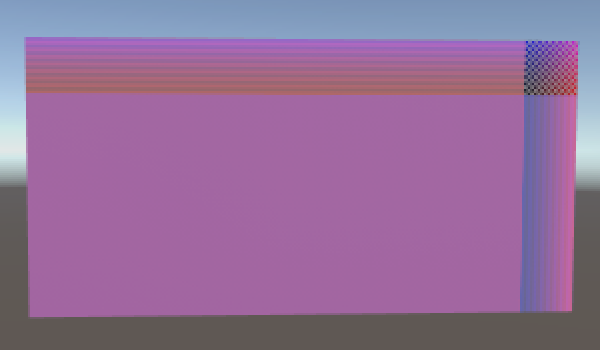
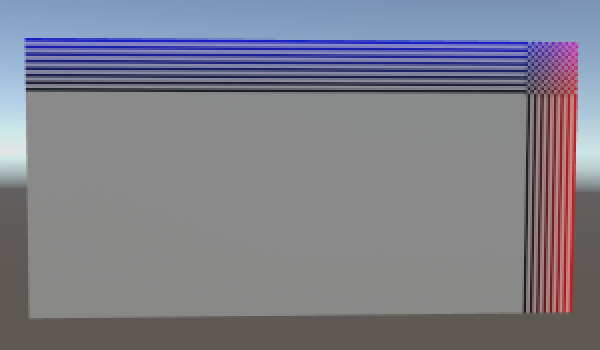
uv[i] = **new** [**Vector2**](http://docs.unity3d.com/Documentation/ScriptReference/Vector2.html)(x / xSize, y / ySize);

}

}

mesh.vertices = vertices;

mesh.uv = uv;

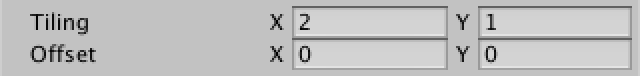


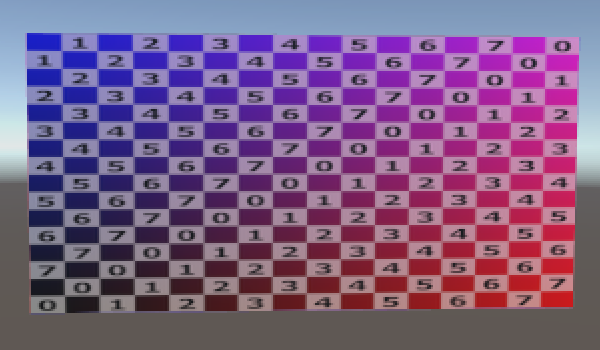
不正确的UV坐标，夹紧与包装纹理。

纹理现在显示出来，但它没有覆盖整个网格。它的准确外观取决于纹理的缠绕模式是设置为钳形还是重复。这是因为我们现在用整数除整数，结果是另一个整数。为了在整个网格中得到0和1之间的正确坐标，我们必须确保使用浮点数

uv[i] = **new** [**Vector2**](http://docs.unity3d.com/Documentation/ScriptReference/Vector2.html)((**float**)x / xSize, (**float**)y / ySize);

纹理现在被投射到整个网格上。当我将网格的大小设置为10 \* 5时，纹理会显示为水平拉伸。这可以通过调整材质的平铺设置来解决。通过设置它为(2,1)U坐标将加倍。如果纹理被设置为重复，那么我们将看到它的两个正方形块

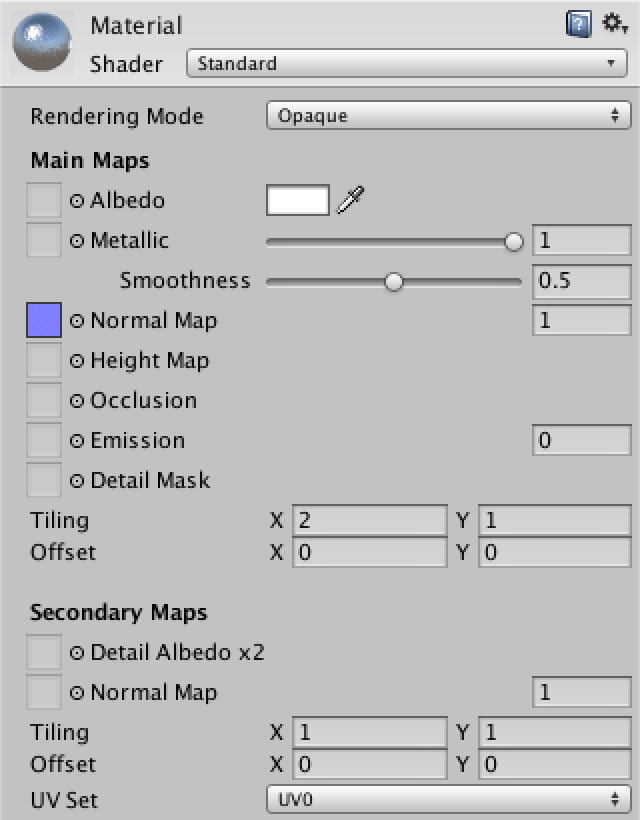
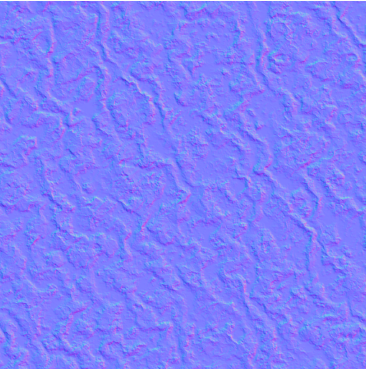






正确的UV坐标，平铺1 ,1对2, 1。

另一种增加表面细节的方法是使用法线贴图。这些地图包含用颜色表示的法向量。将它们应用到一个表面上将会产生比仅使用顶点法线创建更详细的光效果。



凹凸不平的表面，为达到戏剧效果而金属化。

将这种材料应用于我们的网格会产生凸点，但它们是不正确的。我们需要添加切向量到我们的网格，以正确地定向它们。

切线是如何工作的?法向映射是在切线空间中定义的。这是一个围绕物体表面流动的三维空间。这种方法允许我们在不同的位置和方向应用相同的法线贴图。表面法线在这个空间中是向上的，但是哪个方向是对的呢?它由tan定义。理想情况下，这两个向量的夹角是90度。它们的外积产生定义三维空间所需的第三个方向。在现实中，角度往往不是90°，但结果仍然足够好所以正切是一个三维向量，而单位是一个四维向量。它的第四个分量总是-1或1，用来控制第三个切线空间维度的方向——向前或向后。这有助于法向地图的镜像，法向地图常用于具有双边对称性的物体(如人)的三维模型中。Unity的着色器执行这个计算的方式要求我们使用-1。

因为我们有一个平面，所有的切线都指向同一个方向，也就是右边。

vertices = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)[(xSize + 1) \* (ySize + 1)];

[**Vector2**](http://docs.unity3d.com/Documentation/ScriptReference/Vector2.html)[] uv = **new** [**Vector2**](http://docs.unity3d.com/Documentation/ScriptReference/Vector2.html)[vertices.Length];

[**Vector4**](http://docs.unity3d.com/Documentation/ScriptReference/Vector4.html)[] tangents = **new** [**Vector4**](http://docs.unity3d.com/Documentation/ScriptReference/Vector4.html)[vertices.Length];

[**Vector4**](http://docs.unity3d.com/Documentation/ScriptReference/Vector4.html) tangent = **new** [**Vector4**](http://docs.unity3d.com/Documentation/ScriptReference/Vector4.html)(1f, 0f, 0f, -1f);

**for** (**int** i = 0, y = 0; y <= ySize; y++) {

**for** (**int** x = 0; x <= xSize; x++, i++) {

vertices[i] = **new** [**Vector3**](http://docs.unity3d.com/Documentation/ScriptReference/Vector3.html)(x, y);

uv[i] = **new** [**Vector2**](http://docs.unity3d.com/Documentation/ScriptReference/Vector2.html)((**float**)x / xSize, (**float**)y / ySize);

tangents[i] = tangent;

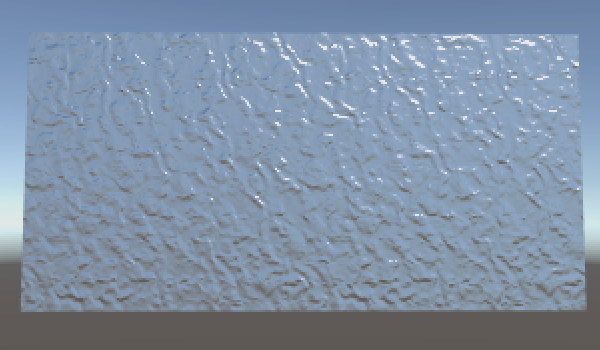
}

}

mesh.vertices = vertices;

mesh.uv = uv;

mesh.tangents = tangents;



表面平坦，表面凹凸不平。

现在你知道如何创建一个简单的网格，使它看起来更复杂的材料。网格需要顶点位置和三角形，通常也需要UV坐标——最多四组——通常还需要切线。你也可以添加顶点颜色，尽管Unity的标准着色器不使用这些。你可以创建自己的着色器来使用这些颜色，但那是另一个教程的内容。