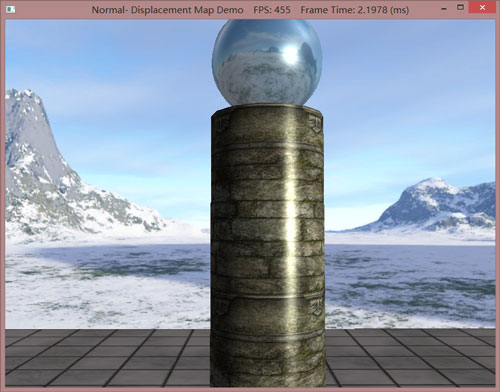
# 法线贴图(Normal Map)

# 概述

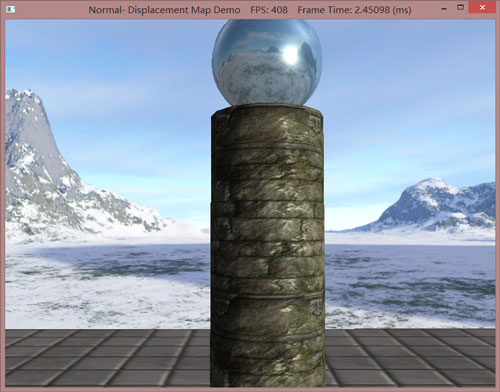
之前我们使用过纹理贴图映射，它可以将纹理细节映射到三角形表面上。然而，我们的法线向量仍然定义在比较粗糙的顶点级别上，使用的是三角形表面插值。这样会产生一些问题，如下图中的圆柱体，其高光看上去有些问题——光滑的柱体表面与凹凸不平的砖块纹理放在一起显得极不谐调。这是因为网格几何体是平滑的，我们只是在平滑的柱体表面上使用了一个凹凸不平的砖块纹理而已。而光照计算是根据网格几何体（尤其是顶点法线插值）来实现的，没有考虑到纹理图像的内容。



在理想状态下，我们可以对网格几何体进行细化，通过网格来模拟砖块实际具有的凹凸裂痕，使光照和纹理表现出一致的质感。硬件曲面细分可以实现这一目的，但是，我们仍需要一种指定由细分器产生的顶点法线（通过插值得到的法线并不能增加法线的分辨率）的方法。

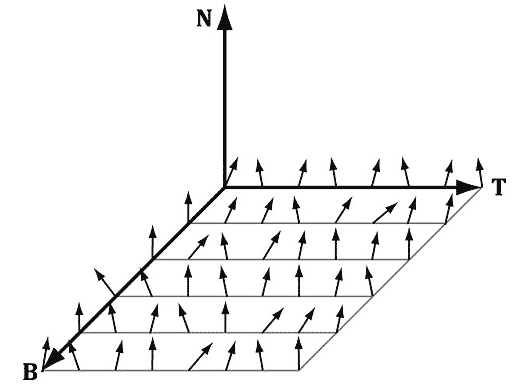
另一种可能的解决办法是将光照细节直接烘焙(bake)到纹理上。不过，当灯光在场景中移动时，这种方法会产生错误的光照结果。因为固定在纹理元素中的光照颜色不会随着灯光的移动而变化。

法线贴图则可用于解决该问题。下图是使用法线贴图后的情况。



# 法线贴图

法线贴图是一种特殊的纹理，它的每个纹理元素存储的不是RGB数据，而是压缩后的x、y、z坐标（分别存储在R、G、B分量中）。这些坐标定义了一个法线向量；法线贴图的每个像素存储的都是这种法线向量。下图说明了法线贴图的实现方式。



法线向量存储在一个由向量T（x轴）、B（y轴）、N（z轴）构成的纹理空间坐标系中。向量T水平向右遍历法线贴图(u方向)，向量B垂直向下遍历法线贴图(v方向)，向量N垂直于法线贴图平面。

如图所示，向量一般与z轴对齐。也就是，z坐标的值最大。所以，当把法线贴图视为一个彩色图像时（例如在Photoshop中查看法线贴图），它一般呈现为偏蓝的颜色。这是因为z坐标存储在蓝色通道中，该通道的数值最大，蓝色占据首要地位。

我们采用24位图像格式，为每个颜色分量分配一个字节，每个颜色分量的取值范围在0到255之间。对于单位向量来说，每个坐标都在[−1,1]区间内。如果我们将该区间调整为[0,1]、乘以255并舍去小数，那结果就是一个在[0,255]区间内的整数。这样我们就将一个单位向量压缩到了这种格式里。

当我们在着色器中对一个法线贴图进行采样时，可使用如下代码：

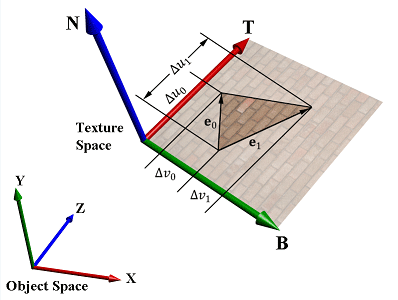
float3 normalT = gNormalMap.Sample(gTriLinearSam,pIn.texC);

颜色向量normalT将会包含规范化分量(r,g,b)，其中0≤r，g，b≤1。这样，该方法已经替我们完成了一部分解压缩工作（也就是除以255、将[0,255]区间内的整数变换为[0,1]区间内的浮点数）。我们只需要使用函数将每个分量从[0,1]区间调整到[−1,1]区间，即可完成转换工作：

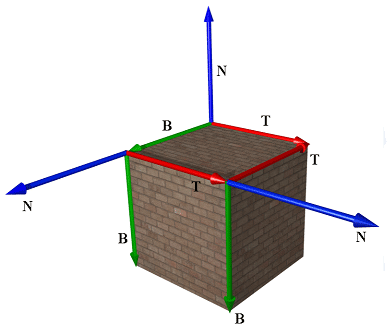
normalT = 2.0f\*normalT - 1.0f;

# 纹理/正切空间

考虑一个3D纹理映射三角形。假设纹理是一张彩色帖纸。我们对它进行平移和旋转，把它贴到3D三角形上。下图说明了如何在纹理坐标轴与3D三角形之间建立联系：它们正切于三角形，并位于三角形平面上。当然，三角形的纹理坐标是相对于纹理坐标系的。现在把三角形平面法线N合并进来，我们便得到了一个在三角形平面上的3D TBN基（3D TBN-basis），我们将它称为纹理空间（texture space）或正切空间（tangent space）。T、B、N向量分别称为切线向量、副法线向量（或副切线向量）、法线向量。



注意，正切空间通常会随着三角形而变化。



# 纹理空间的变换

我们存储在法线贴图中的法线向量使用的是纹理空间坐标系。而场景中的光照使用的是世界空间坐标系。为了进行光照，我们必须把法线向量和灯光变换到同一个坐标系中。只要进入物体空间，我们就可以通过世界矩阵实现从物体空间到世界空间的坐标变换。故我们需要将法线变换到物体空间。

正切空间在三角形平面上是一个常量。所以，我们在每个顶点上指定切线向量，之后我们通常需要对TBN基进行正交化处理，使这3个向量彼此垂直并为单位向量。这里，我们不直接在内存中存储副切线向量B。而是，在需要B时计算B=N×T，其中N是常规的经过平均后的顶点法线。

此时，我们的顶点结构体为：

|  |
| --- |
| struct NormalMap  {  XMFLOAT3 Pos;  XMFLOAT3 Normal;  XMFLOAT2 Tex;  XMFLOAT3 TangentU;  } ; |

进行变换的代码如下

|  |
| --- |
| float3 NormalSampleToWorldSpace(float3 normalMapSample, float3 unitNormalW, float3 tangentW)  {  // Uncompress each component from [0,1] to [-1,1].  float3 normalT = 2.0f\*normalMapSample - 1.0f;      // Build orthonormal basis.  float3 N = unitNormalW;  float3 T = normalize(tangentW - dot(tangentW, N)\*N);  float3 B = cross(N, T);      float3x3 TBN = float3x3(T, B, N);      // Transform from tangent space to world space.  float3 bumpedNormalW = mul(normalT, TBN);      **return** bumpedNormalW;  } |