# **游戏中的3D数学知识之法线的应用**

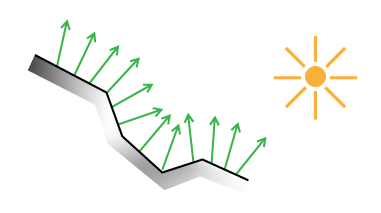
作者：日音

链接：<http://gad.qq.com/article/detail/286121>

      很多时候做游戏都想实现一些炫酷的效果，包括水，风，火，土，雷等。了解游戏开发的人都知道这些东西往往都是由程序自己实现的，美术更多提供的是辅助。而平时自己想要实现的时候总觉得这些会涉及一些极为麻烦的数学知识，这里我就想从法线作为一个切入角度，并且把对应的知识点都列出来，因为涉及比较多的内容，具体细节还希望大家多多去查找资料。一旦掌握这些全部的知识点之后，相信以后这些效果都会信手拈来（当然如果要做次时代效果还需要进一步研究，如果只是手机上的话，基本是够用的）。

**知识点：理解法线是什么**

       我们知道，游戏中的模型面数越高，细节越丰富，那么就可以出越好看的效果。但出于性能的限制，我们往往无法无限制的增加细节。于是人们就想了一个办法，就是将一个面数很高的模型的法线信息存入到一张图片里面，然后低面数的模型通过对这个贴图的采样去制造自己有很多细节的效果。在这种场景下，法线的作用可以理解为反映出模型表面的凹凸信息。

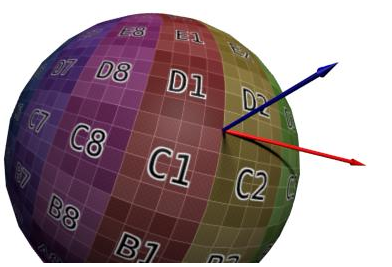
      **知识点：模型法线怎么算**

      首先我们需要知道如何表示一个法线。正常情况下我们都是用一个三维向量表示法线的方向。这个向量垂直于当前顶点所在的这个面，如果有多个面共用这个顶点，那么就可以通过取所有面的平均值来计算出这个法线。再得知如何计算法线之后，我们就可以开始考虑如何存储法线。

**知识点：模型法线的坐标系**

      对于静止的物体，我们可以轻易的把它在当前状态下的全部法线信息保存下来，但是如果物体发生了移动或者旋转，我们就会发现物体相对于世界坐标发生了改变，这是不可接受的。因为我们的游戏中的物体肯定会经常移动变换，如果要针对每一个状态都存储对应的法线信息，显然是不可取的。于是人们想到了可以用模型坐标系或者切线空间坐标系来表示法线的坐标。目前绝大多数都是采用切线空间坐标系来表示法线坐标的。

**知识点：切线空间坐标系**

****

参考这个球，蓝色的线垂直于球面，就是法线坐标系的Z轴，而红色的线和贴图的方向一致，我们可以认为是X轴，然后Y轴可以根据这两个算出来，图中没有画，但我们可以知道他是垂直于这两个轴的。那么这个坐标的含义是什么呢？在切线空间坐标中(0, 0, 1)代表当前顶点法线没有任何偏移，而一旦x,y不为0，那么就代表着法线分别向x，y进行一定的偏移。这样做的好处十分明显，对于不停移动变换的物体，法线信息存储一份就可以了，因为相对当前顶点而言，这个顶点的法线和法线贴图之间的偏移总是保持不变。

**知识点：法线的存储方式**

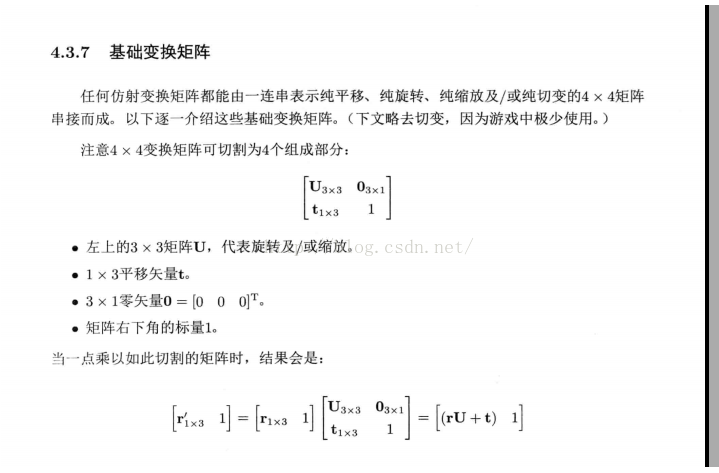
知道了法线的坐标系，那么我们就可以通过模型去算出法线，同时转化成切线空间下的三维向量。那么要怎么保存到一张贴图中呢？首先我们可以把法线坐标限制在长度为1内，那么可以通过范围[-1,1]来表示法线在360度内的偏移。但我们知道贴图正常情况下是无法存储负值的，于是我们进行这样的变换，f(x) = (x + 1) / 2。 这样子我们就发现范围被限制到了[0,1]中，就可以存储在图片里面了。那么使用的时候，我们需要反向操作，而在Unity中也是这么做的：

inline fixed3 UnpackNormal(fixed4 packednormal)  
｛  
    return packednormal.xyz \* 2 - 1;  
｝

这个函数就是将从图片读出的值经过变换，重新回到[-1,1]的范围。

**知识点：法线坐标的变换，矩阵的使用**

我们已经知道怎么存储法线，也知道了怎么读取法线，可是这样读取的法线是在切线空间坐标下的，我们需要将他们转到世界坐标下，这样才可以和光线等进行计算，获得光影效果。那么怎么去进行坐标变换呢？这就需要用矩阵的知识，我简要说下矩阵的小基础：

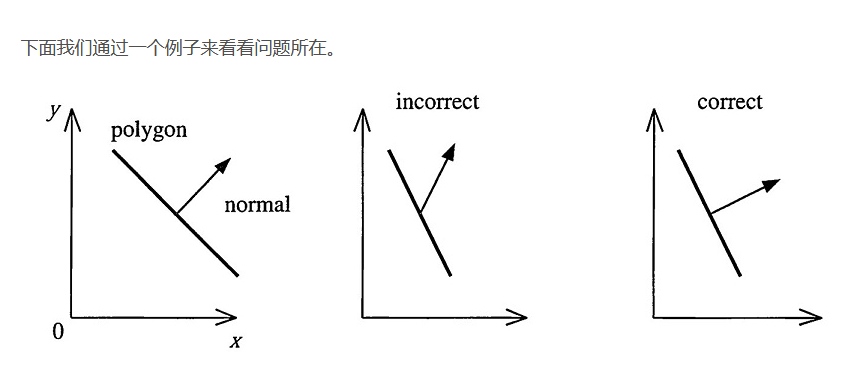


     极为简要的说，矩阵可以实现坐标的平移变换和选择变换。两个坐标系的变换，无非就是将坐标平移一下，旋转一下。而数学家们早已经将这些常用变换的证明以及应用搞得明明白白。我们只需要知道原理并且去使用即可。

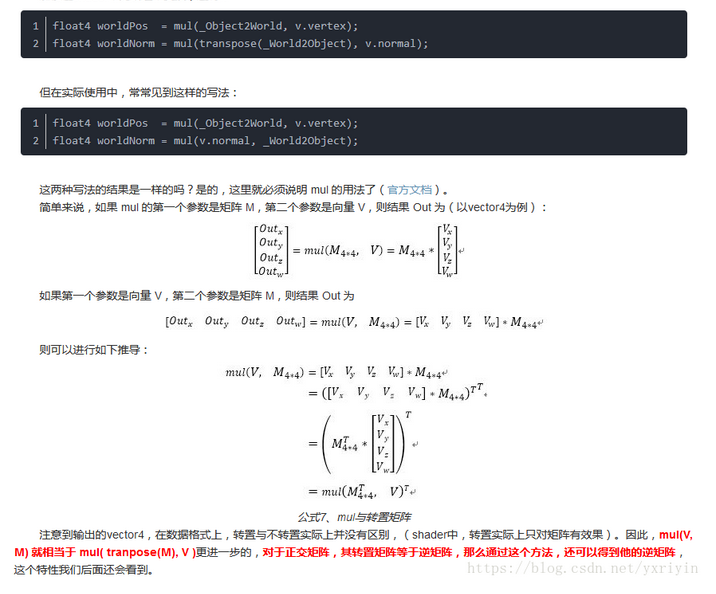
     对于从切线空间到世界空间的坐标变换，其实就是将切线空间的各个坐标列成一个矩阵，然后乘以切线空间中的坐标，就会变成世界空间的坐标。

**知识点：法线坐标变换的不同点**

对于不是垂直于模型面的轴，我们可以用上述的方法变换，但是对于垂直于面的轴，用上面的变换会发现一些错误的地方：

     第二个图就是我们算出的结果，没错，法线不垂直面了，而图三才是我们希望的结果。那么问题出在哪呢？经过数学家的证明（有兴趣的可以搜索法线坐标变换，这个证明并不难，可以自己看看），垂直面的法线坐标变换是需要用原矩阵的逆转置矩阵进行变换。

      于是这样才是正确的：



这里是不是看不太懂，如果没有任何矩阵基础的话看上去是有点费力，可以先去了解下矩阵基础，然后就会明白Unity里面采用了等价的写法，原因是等价的写法性能更好，少了一步运算。

      掌握以上全部知识，就可以自己写一个使用法线贴图的着色器了，为了更加深入的了解，我们可以一步步验证下Unity内置的着色器是如何去做的：

v2f o;  
 o.vertex = mul(UNITY\_MATRIX\_MVP, v.vertex);

o.uv.xy = TRANSFORM\_TEX(v.uv.xy, \_MainTex);

o.uv.zw = TRANSFORM\_TEX(v.uv.xy, \_BumpTex);

//从这里开始就是法线相关内容

//这是把顶点坐标转成世界坐标

float3 worldPos = mul(unity\_ObjectToWorld, v.vertex).xyz;

//这里是把顶点法线转到世界法线

fixed3 worldNormal = UnityObjectToWorldNormal(v.normal);

//这里是吧顶点切线转到世界切线，注意v.tangent和v.normal是Unity算好的内置变量

fixed3 worldTangent = UnityObjectToWorldDir(v.tangent);

//根据上述两个法线，我们就可以得到第三个轴的坐标，我们称他为副法线

fixed3 binormal = cross(worldNormal, worldTangent) \* v.tangent.w;

//前面说过，将上面三个轴的坐标列成一个矩阵，就可以将法线从切线空间转化到世界空间

//第四个参数只是为了传递坐标，和法线其实无关

o.TtoW0 = float4(worldTangent.x, binormal.x, worldNormal.x, worldPos.x);

o.TtoW1 = float4(worldTangent.y, binormal.y, worldNormal.y, worldPos.y);

o.TtoW2 = float4(worldTangent.z, binormal.z, worldNormal.z, worldPos.z);

//取出法线颜色

fixed4 bumpColor = tex2D(\_BumpTex, i.uv.zw);

fixed3 tangentNormal;

//将法线颜色转成法线数值，即从[0,1]变成[-1,-1]范围

tangentNormal = UnpackNormal(bumpColor);

//这里是前面没有提到的，这么做主要是对法线进行归一化，主要是为了后面光照计算进

// 行处理，由于光照部分是另一个大课题，这里就不展开了

tangentNormal.z = sqrt(1 - saturate(dot(tangentNormal.xy, tangentNormal.xy)));

//最后就是我们说的矩阵变换，至此法线已经被得到了

float3x3 t2wMatrix = float3x3(i.TtoW0.xyz, i.TtoW1.xyz, i.TtoW2.xyz);

tangentNormal = normalize(half3(mul(t2wMatrix, tangentNormal)));  
  
当然，仅仅这样一片文章是无法涵盖所以的细节的，只是作为一篇引子，希望大家有兴趣可以自己去学习更多的内容。