让人头疼的纹理（下）：几何纹理

华中科技大学软件学院　万琳

提纲

1. 几何纹理的概念及算法
2. 法线贴图
3. 基于OpenGL的法线贴图

1 几何纹理的定义和算法

◆几何纹理：

现实中遇到的问题：粗糙的外观

1 几何纹理的定义和算法

◆几何纹理：

现实中遇到的问题：表面的凹凸不平

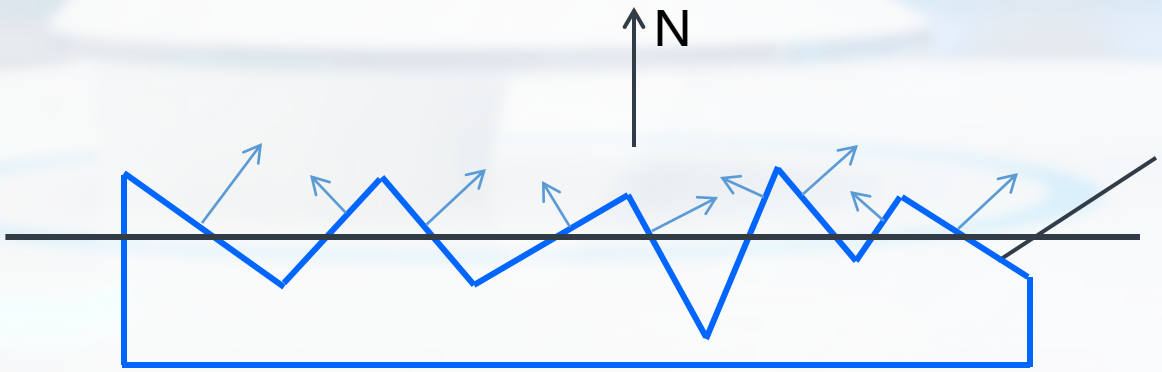
1 几何纹理的概念及算法

◆几何纹理：

•几何纹理方法 - 对物体表面几何性质作微小扰动，产生凹凸不平的细节效果，给物体表面图象加上一个粗糙的外观

•物体表面上的每一个点P(u,v)，都沿该点处的法向量方向位移F(u,v)个单位长度，

新表面位置：$\tilde{P}(u,v) = P(u,v) + F(u,v) * N(u,v)$



最终还是利用光照产生立体感

1 几何纹理的定义及算法

◆几何纹理算法：

年代	算法	思想
1978	Bump Mapping 凹凸贴图	计算顶点光强时，不是直接使用原始法向量，而是加上一个扰动
1984	Displacement Mapping移位贴图	直接作用于顶点，根据Displacement Mapping中相对应的像素值，使顶点沿法向移动，产生**真正的凹凸表面**
1996	Normal Mapping 法线贴图	通过height map获得法向量信息，而且对应的RGB值表示法向量的XYZ，利用这个信息计算光强，产生凹凸阴影的效果
2001	Parallax Mapping 视差贴图	通过视线和height map的计算，陡峭的视角就给顶点更多的位移，平缓的就给较小的位移，通过视差获得更强的立体感
2005	Relief Mapping 浮雕贴图	更精确地找出观察者视线与高度的交点，实现更精确的位移

除了移位贴图以外，表面并没有变得真正的凹凸不平

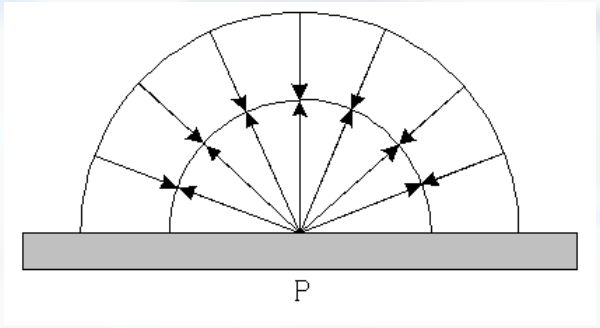
来自《Real-Time Rendering 3rd》提炼总结

1 几何纹理的定义及算法

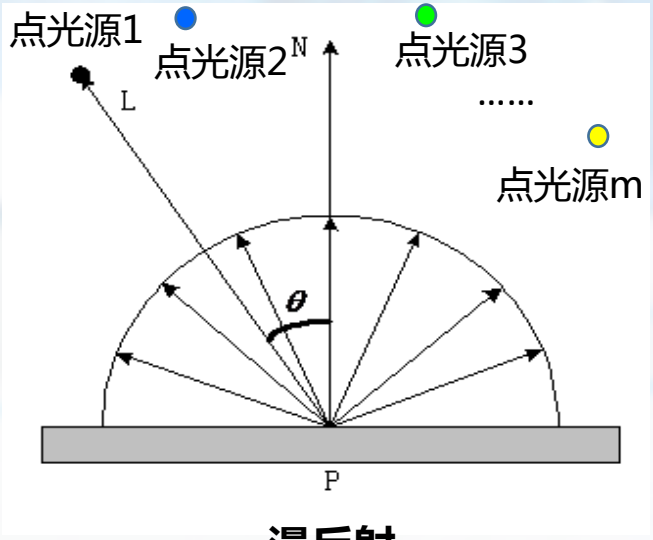
◆从Phong模型可见，法向量N不同，光照计算结果就不同

光强计算假定只有一个点光源，若在场景中有m个点光源，则可以在任一P点上叠加各个光源所产生的光照效果：

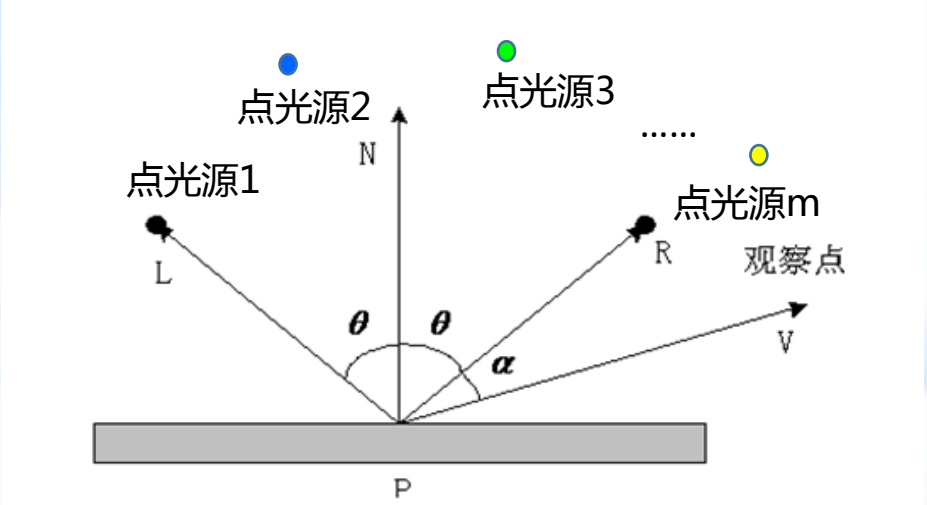
$$I=I_aK_a+\sum_{i=1}^{m}I_{p,i}K_d(L_i\cdot N)+\sum_{i=1}^{m}I_{p,i}K_s(H_i\cdot N)^n$$



环境光
Ambient



漫反射
Diffuse



镜面反射
Specular

1 几何纹理的定义及算法

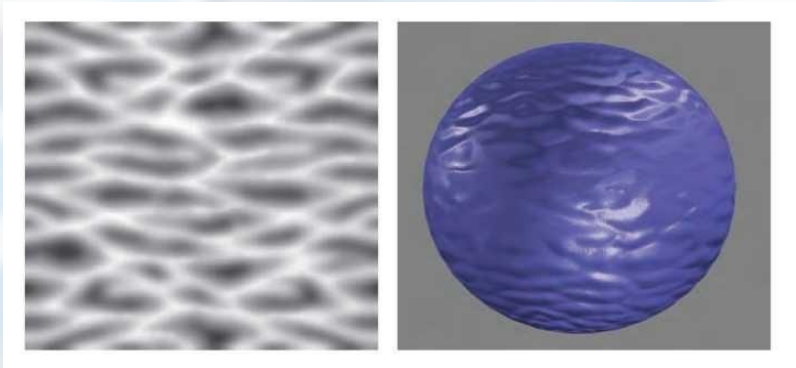
◆几何纹理算法：

Displacement Mapping移位贴图

其余的纹理算法

1 几何纹理的定义和算法

◆几何纹理算法：

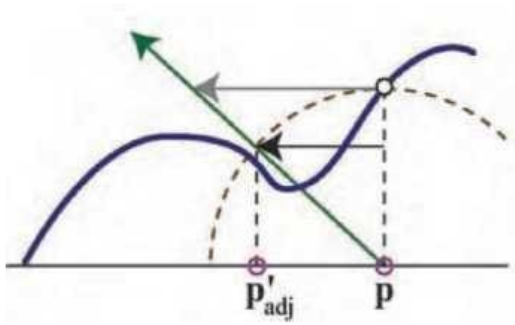


Bump Mapping
凹凸贴图

Normal Mapping
法线贴图

1 几何纹理的定义和算法

◆几何纹理算法：



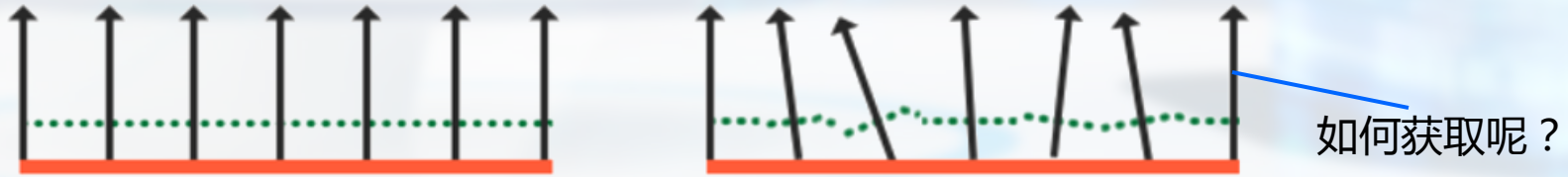
Parallax Mapping
视差贴图

Relief Mapping
浮雕贴图

2 法线贴图

法线贴图

现实中的物体表面并非是平坦的，而是表现出无数（凹凸不平的）细节。
每个fragment使用了自己的法线，我们就可以让光照相信一个表面由很多微小的（垂直于法线向量的）平面所组成，物体表面的细节将会得到极大提升。这种每个fragment使用各自的法线，替代一个面上所有fragment使用同一个法线的技术叫做法线贴图。



2 法线贴图

◆问题一：height map的使用

高度图

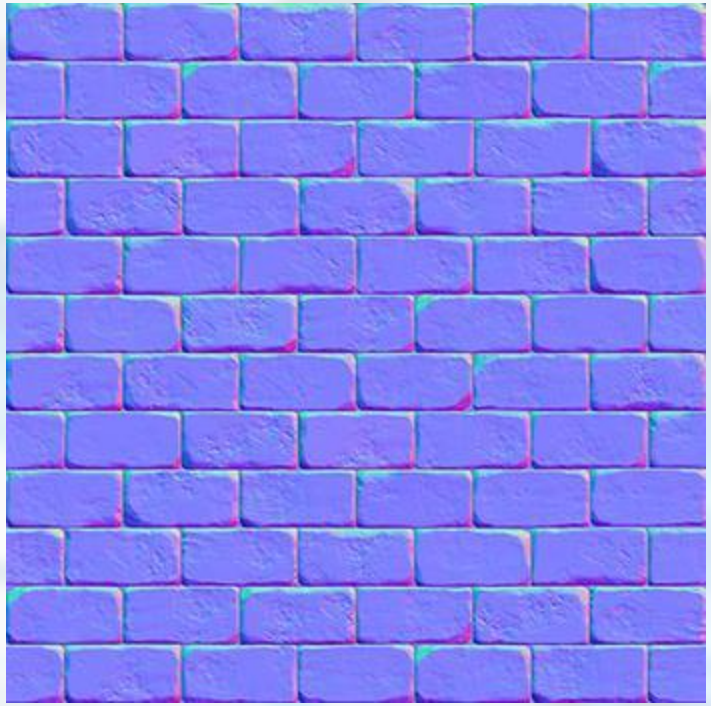
高度图上存储的是RGB值

但每个颜色通道实际上是表面的法线坐标：

◆R红色通道对应x方向

◆G绿色通道对应y方向

◆B蓝色通道对应z方向



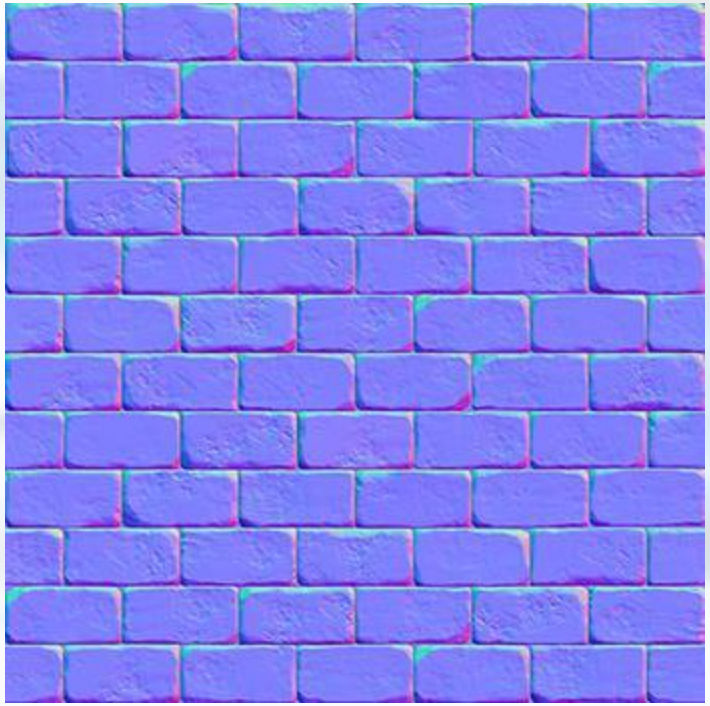
2 法线贴图

◆问题一：height map的使用

通过height map获得法向量信息，即对应的RGB值表示法向量的XYZ，利用这个信息计算光强，产生凹凸阴影的效果

普通贴图+高度图

普通贴图

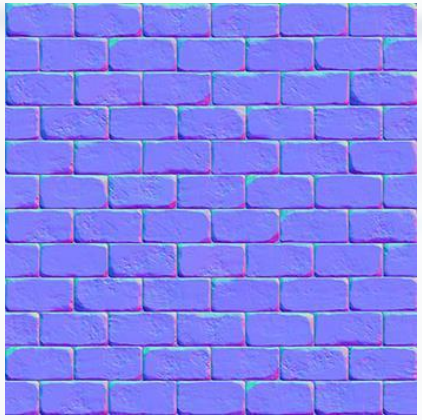


高度图height map

2 法线贴图

◆问题一：height map的使用

光照后颜色值发生变换产生凹凸不平的视觉效果

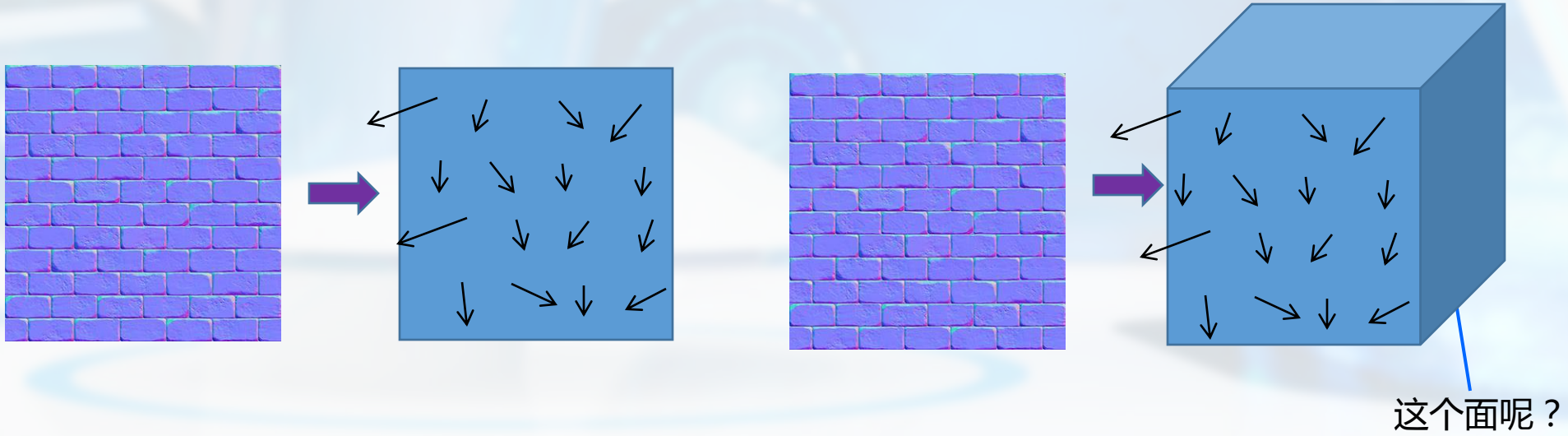


2 法线贴图

◆问题二：切线空间的引入

一个平面上顶点的法向量方向会根据height map取值

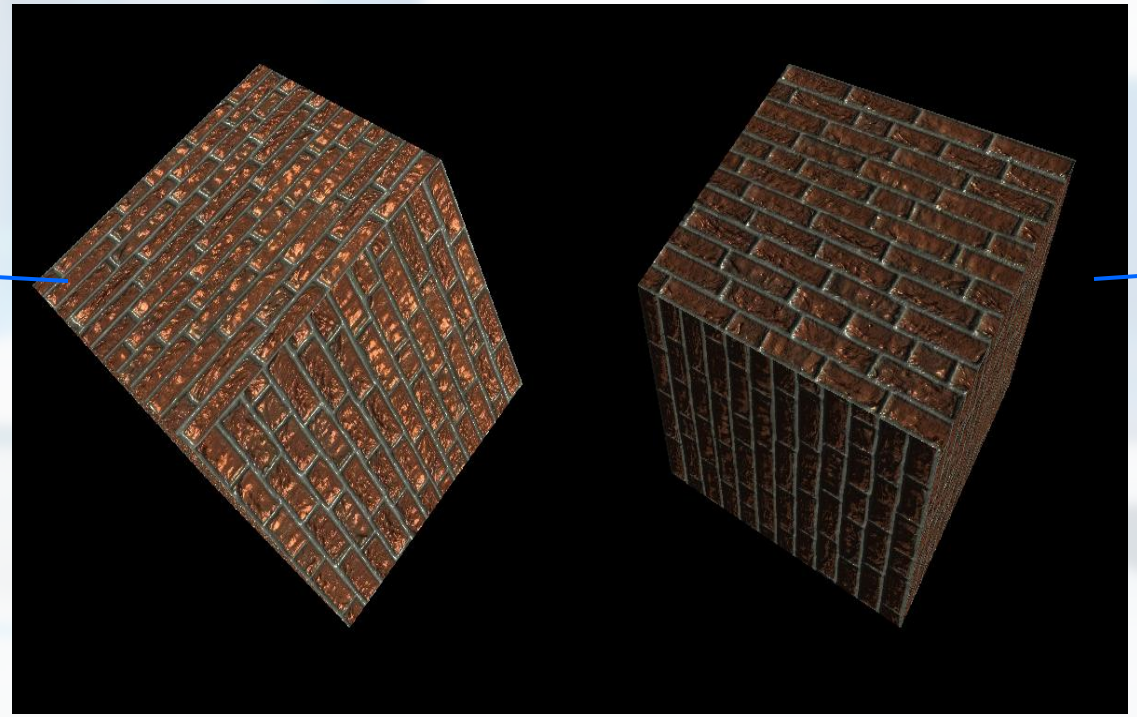
多个不同朝向的平面呢？需要将相对于某个平面的法向量变换到全局中！



2 法线贴图

◆问题二：切线空间的引入

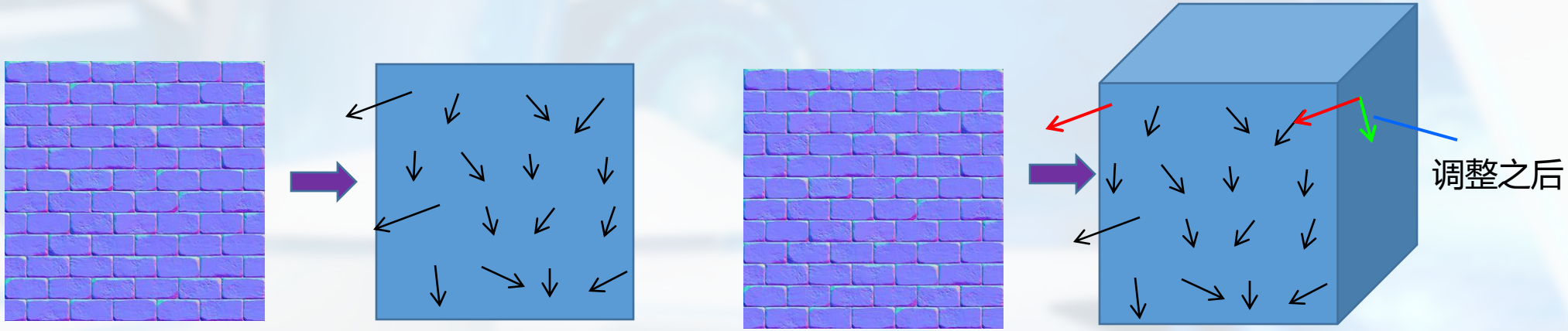
变换前后的对比



2 法线贴图

◆问题二：切线空间的引入

将相对于某个平面的法向量变换到全局中！

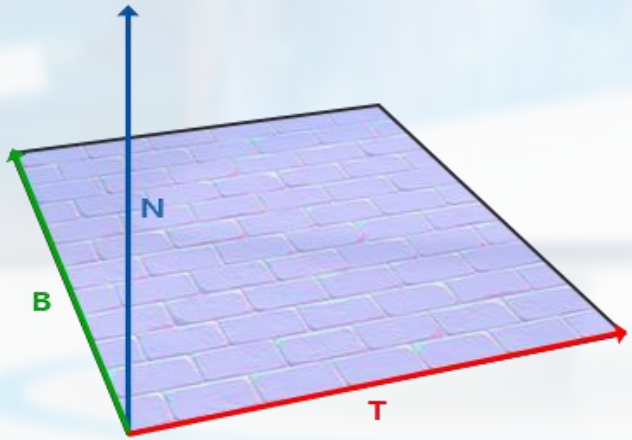


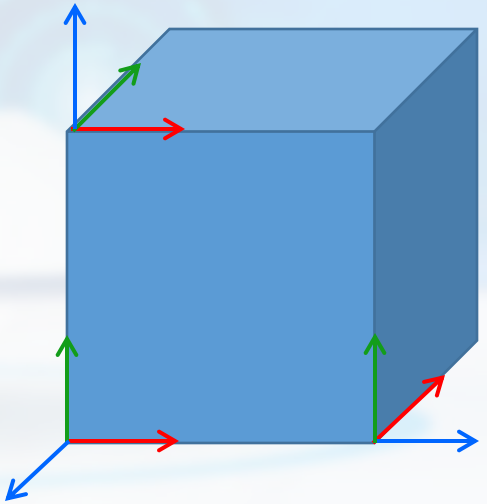
2 法线贴图

◆问题二：切线空间的引入

每个平面都有一个自己的切线空间

T：tangent 切线 B：bitangent 副切线 N：normal法线





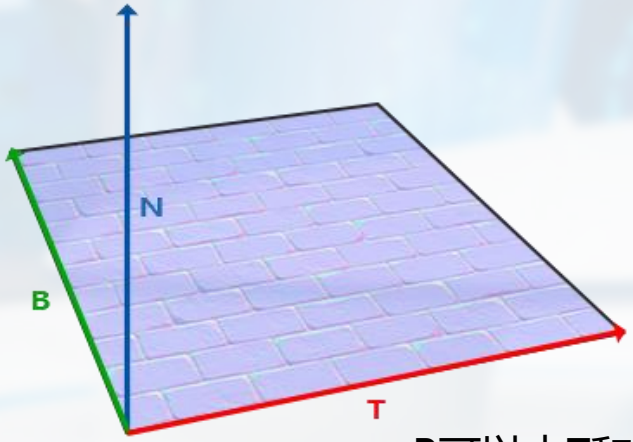
2 法线贴图

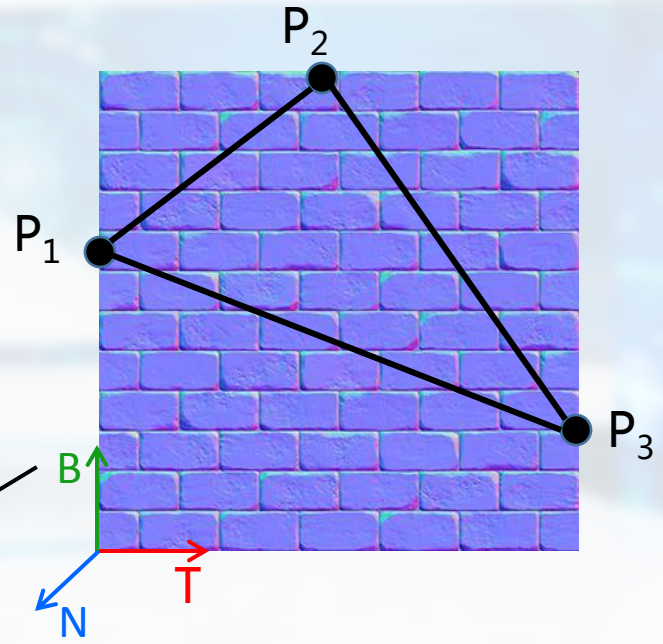
◆问题二：切线空间的引入

TBN矩阵的计算

通过三个共面且不是共线的点P_1、P_2和P_3分别计算TBN的方向量

TBN的方向量构成TBN矩阵





B可以由T和N的方向量计算

3 基于OpenGL的法线贴图

◆在OpenGL中的实现　TBN矩阵

获取TBN矩阵

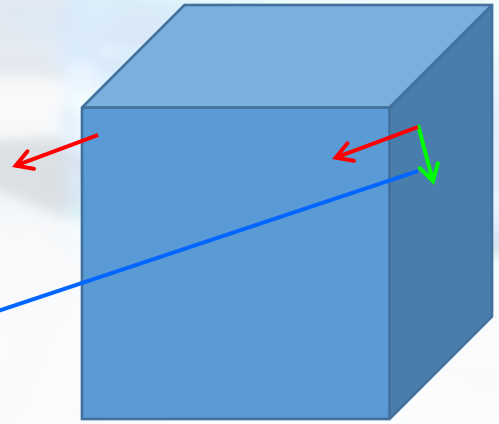
```
layout (location = 1) in vec3 aNormal;      //N
layout (location = 3) in vec3 aTangent;     //T
```

计算TBN矩阵

```
//计算切线空间所需的TBN矩阵
vec3 T = normalize(vec3(model * vec4(aTangent, 0.0f)));
vec3 N = normalize(vec3(model * vec4(aNormal, 0.0f)));
 vec3 B = normalize(cross(T, N));
vs_out.TBN = mat3(T, B, N); //传出TBN矩阵
```

引入切线到世界空间变换

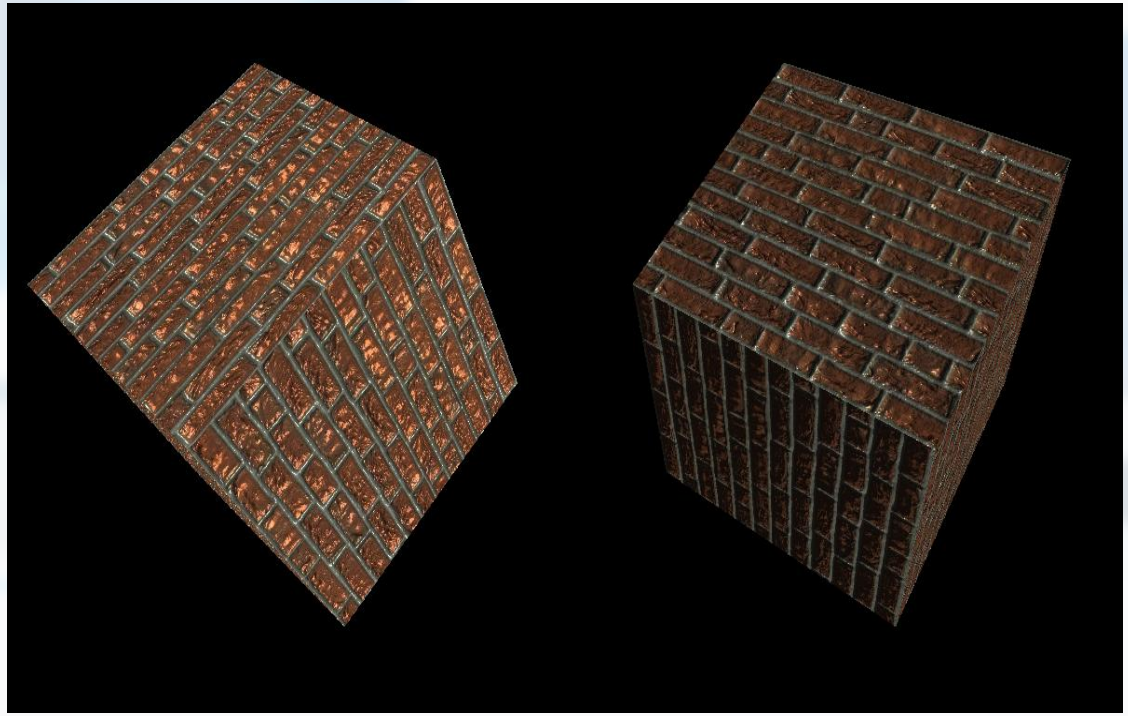
```
normal = texture(normalMap, fs_in.TexCoords).rgb;
normal = normalize(normal * 2.0 - 1.0);
normal = normalize(fs_in.TBN * normal);
```



3 基于OpenGL的法线贴图

◆实验

要求：基于切线空间实现法线贴图（达到右边图的效果）




谢谢
软件学院　万琳