XIAMEN UNIVERSITY

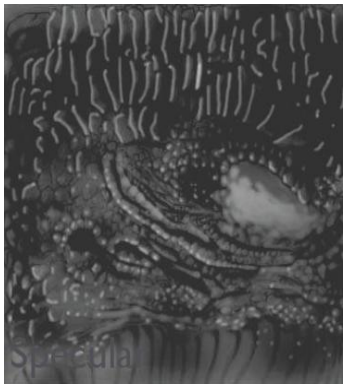
COMPUTER GRAPHICS

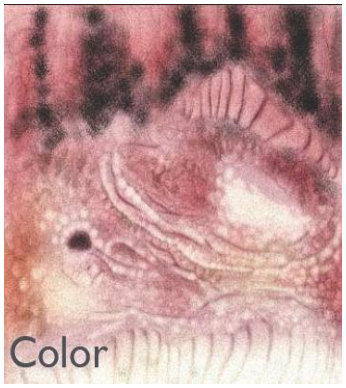
第十章 增强表面细节

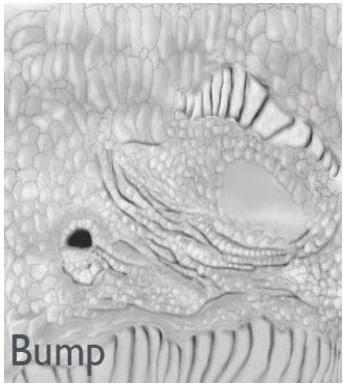
陈中贵
厦门大学信息学院
http://graphics.xmu.edu.cn

如何处理凹凸的表面

- 整个不规则表面进行几何建模 （需要大量的顶点）
- 将纹理图片应用于平滑的对象 （光源移动时效果差）
- 使用凹凸贴图或者法线贴图

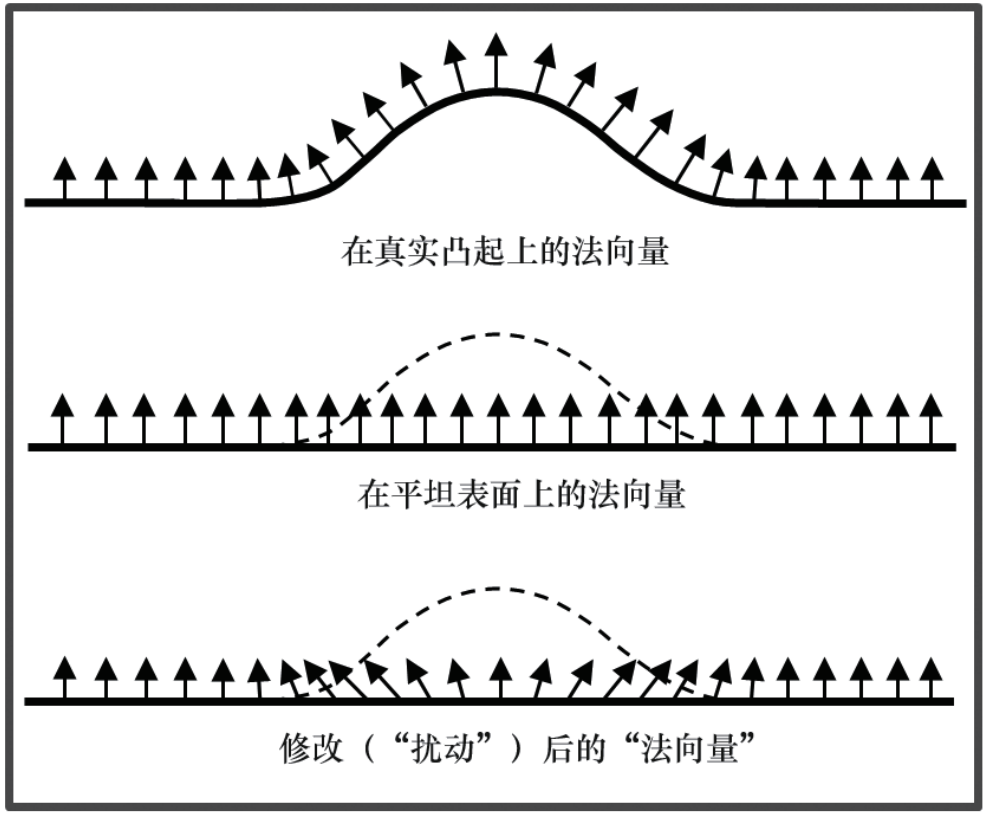






凹凸贴图 Bump Mapping

- 表面法向量在创建逼真的光照效果中是至关重要的
- 直接生成相应法向量，避免生成与凹凸不平或褶皱表面相对应的顶点

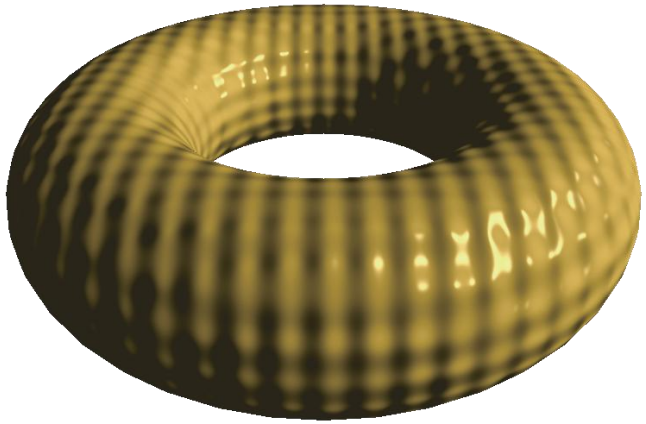


过程式凹凸贴图

- 在运行时使用数学函数对法向量进行改变

Fragment Shader

```
. . .
in vec3 originalVertex;     // passed in from vertex shader
. . .
void main(void)
{  . . .
   float a = 0.25;       // a controls height of bumps
   float b = 100.0;      // b controls width of bumps
   float x = originalVertex.x;
   float y = originalVertex.y;
   float z = originalVertex.z;
   N.x = varyingNormal.x + a*sin(b*x);     // perturb normal using sine
   N.y = varyingNormal.y + a*sin(b*y);
   N.z = varyingNormal.z + a*sin(b*z);
   N = normalize(N);
   // lighting now utilizes the perturbed normal N
   . . .
}
```



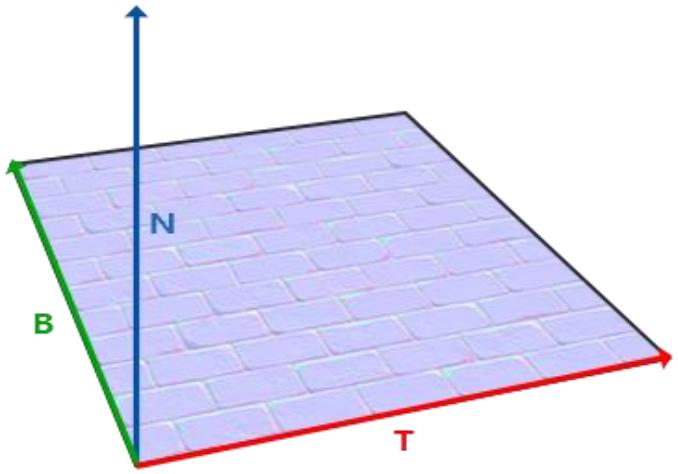
法线贴图

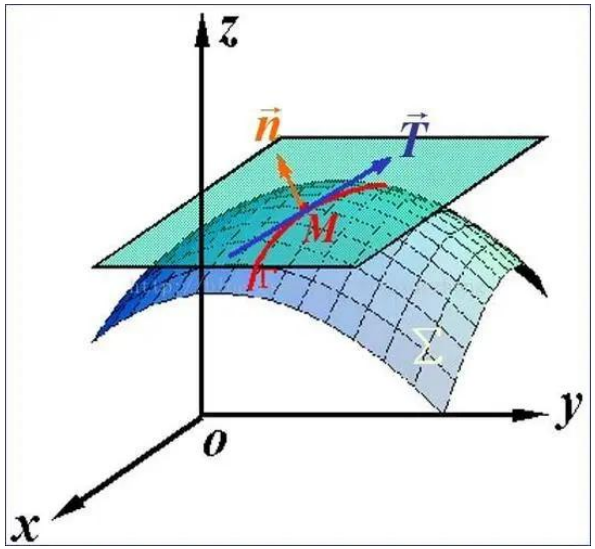
- 使用查找表来替换法向量
- 法向量存储在彩色图像文件中，其中 *R*、 *G*和 *B* 通道分别对应于 *x*、 *y* 和 *z*。
- 将法向量分量限制在[-1,+1]区间，转换到[0,1]：

$$R = (N_x + 1)/2$$
$$G = (N_y + 1)/2$$
$$B = (N_z + 1)/2$$

切线空间

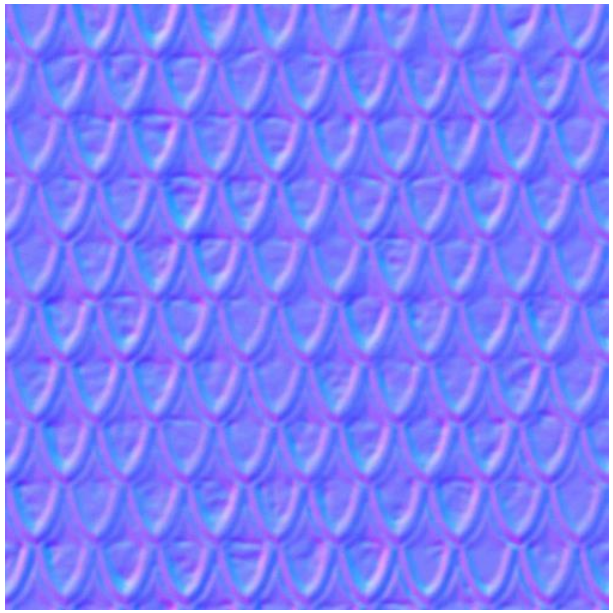
- 从法线图查找到的法向量不能直接使用
- 在法线图中，法向量相对于**切平面**表示，其中，法向量的 *x* 和 *y* 分量表示其被扰动后与“垂直”方向的偏差，*z* 分量设置为 1
- 切线空间：
  - T:切向量 Tangent
  - B:副切向量 Bitangent
  - N:法向量 Normal

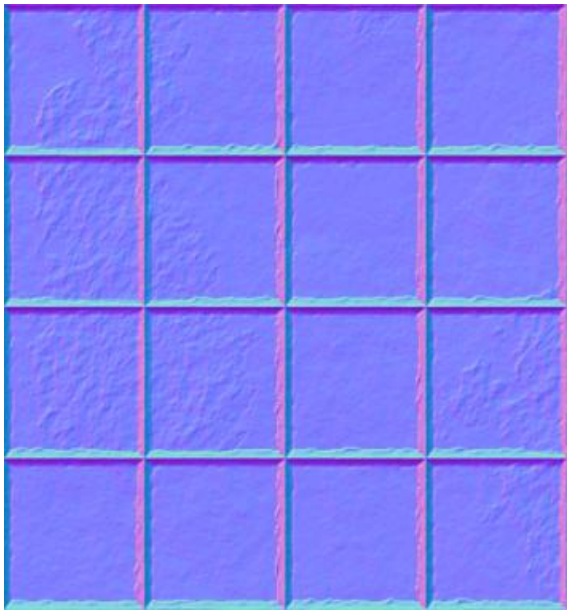




法线贴图

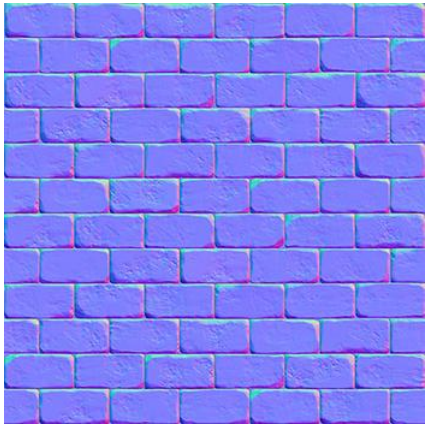
- 从法线图查找到的法向量不能直接使用
- 在法线图中，法向量相对于**切平面**表示，其中，法向量的 *x* 和 *y* 分量表示其被扰动后与“垂直”方向的偏差， *z* 分量设置为 1
- 严格垂直的向量（即没有偏差）将表示为(0, 0, 1)，而不垂直的向量将具有非零的 *x*、 *y* 分量
  - 例如， (0, 0,1)将存储为(0.5, 0.5, 1)，因为实际偏移的区间均为[-1,+1]

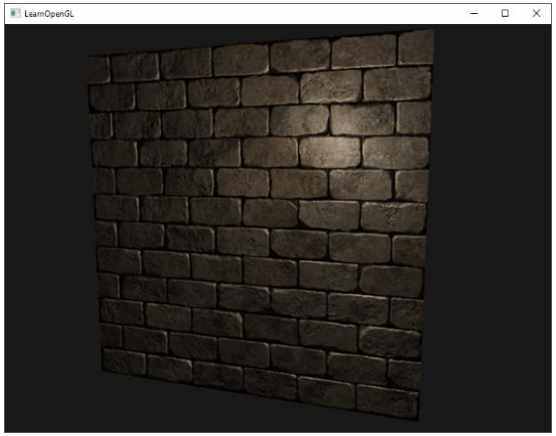




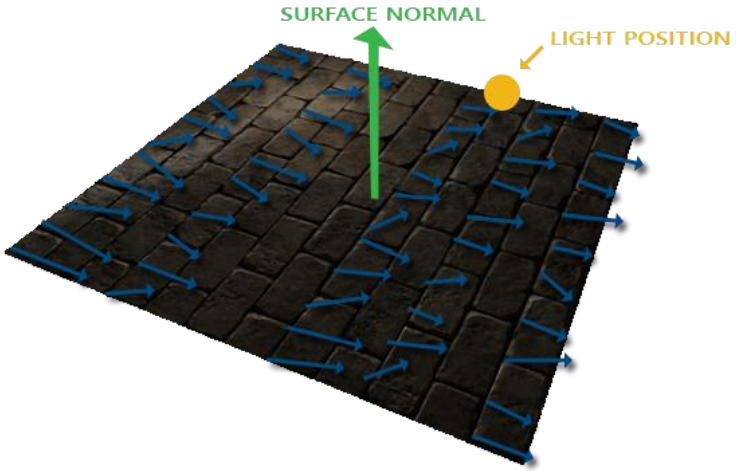
为什么使用切线空间?

- 假设直接使用法线贴图中的法向
  - 法向大体朝向Z轴正方向，能正确渲染和Z轴垂直的如下墙面

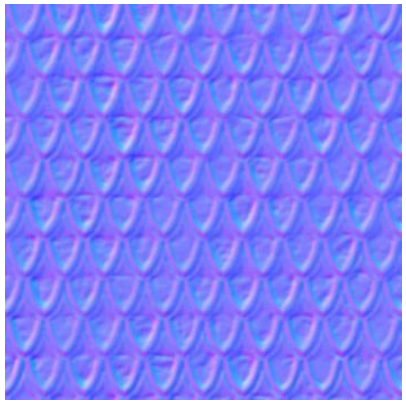


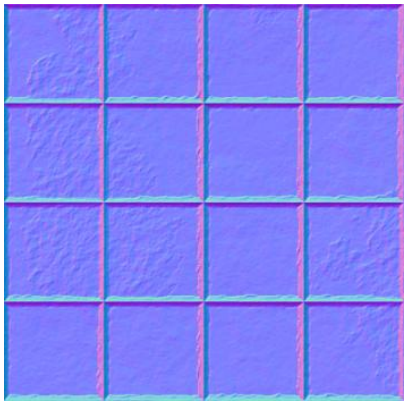


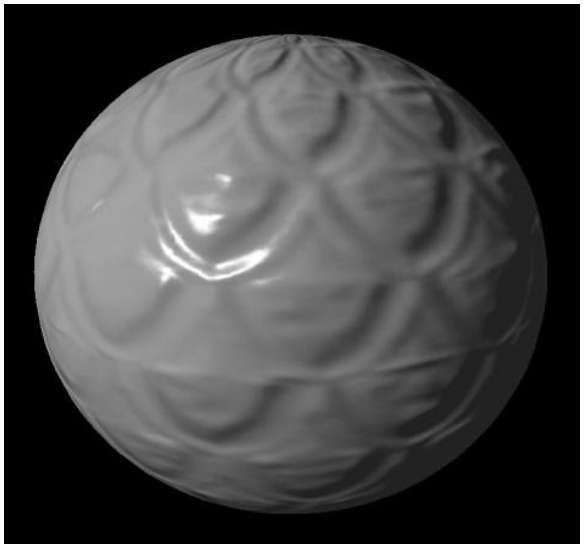
  - 对于铺设在地面上的表面，由于这个表面朝向Y轴正向，不能使用该法向贴图，将导致一张法向贴图只能用在一个平面上。

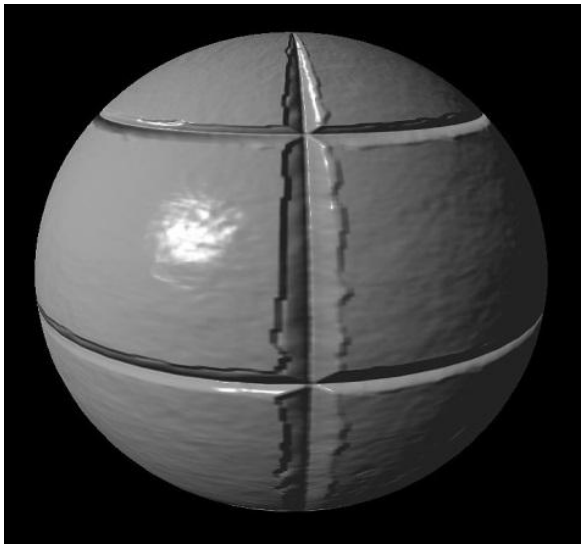


法线图示例









为球面计算切向量

```
. . .
for (int i=0; i<=prec; i++) {
    for (int j=0; j<=prec; j++) {
        float y = (float)cos(toRadians(180-i*180/(prec)));
        float x = -(float)cos(toRadians(j*360/(prec))) * (float)abs(cos(asin(y)));
        float z = (float)sin(toRadians(j*360/(prec))) * (float)abs(cos(asin(y)));
        vertices[i*(prec+1)+j] = glm::vec3(x,y,z);
        . . .
        // calculate tangent vector
        if (((x==0) && (y==1) && (z==0)) || ((x==0) && (y==-1) && (z==0)))
        {   tangent[i*(prec+1)+j] = glm::vec3(0,0,-1);      // if N/S pole, set tangent to -Z axis
        }
        else                    // otherwise, calculate tangent
        {   tangent[i*(prec+1)+j] = glm::cross(glm::vec3(0,1,0), glm::vec3(x,y,z));
        }
        . . . // remaining computations are unchanged
} }
```

其他表面切向的计算：https://blog.csdn.net/Motarookie/article/details/123507610

法向计算

- 一旦在相机空间中得到法向量、切向量和副切向量，就可以使用它们来构造矩阵（称为TBN 矩阵）

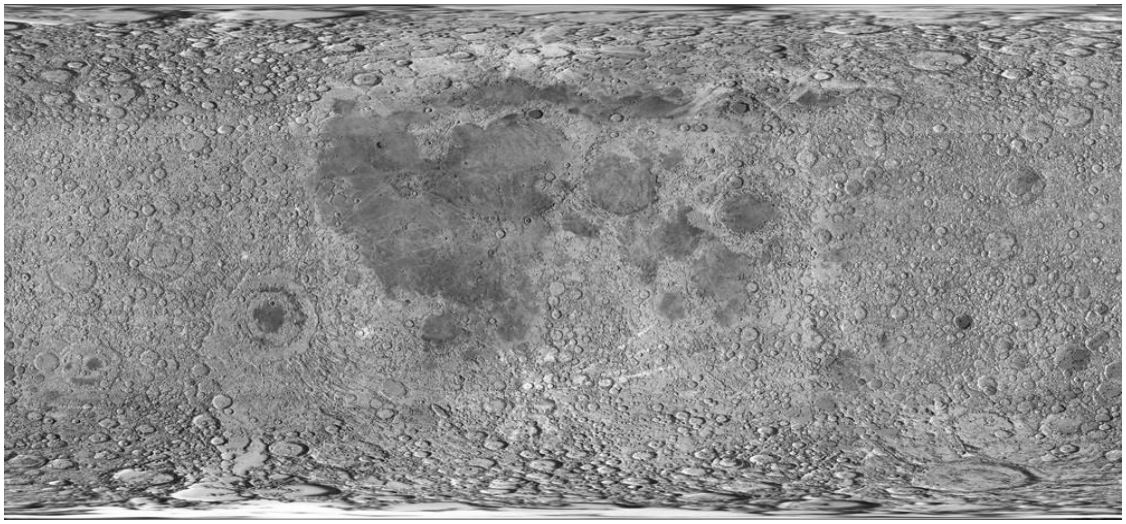
$$TBN = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix}$$

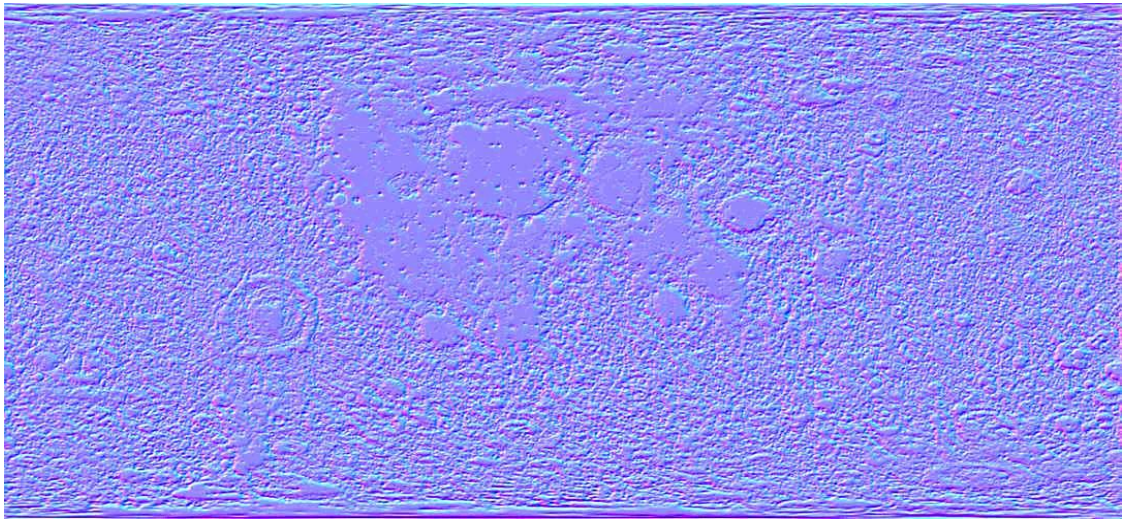
- 该矩阵用于将从法线贴图中检索到的法向量转换为在相机空间中相对于物体表面的法向量。

Fragment Shader

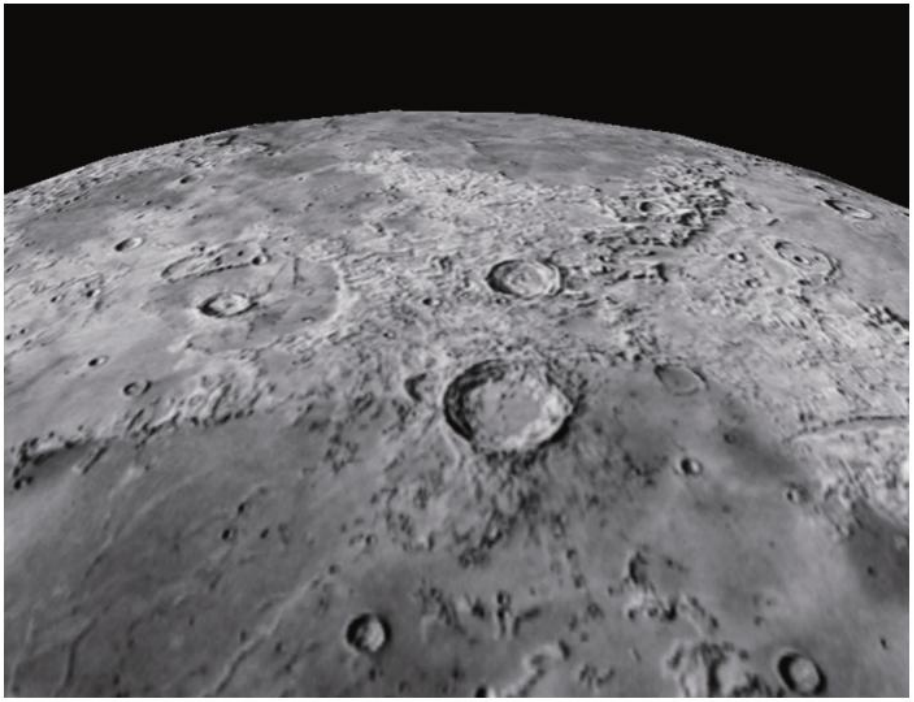
```
. . .
in vec3 varyingTangent;
. . .
layout (binding=0) uniform sampler2D normMap;
. . .
vec3 calcNewNormal()
{   vec3 normal = normalize(varyingNormal);
    vec3 tangent = normalize(varyingTangent);
    tangent = normalize(tangent - dot(tangent, normal) * normal);
    vec3 bitangent = cross(tangent, normal);
    mat3 tbn = mat3(tangent, bitangent, normal);     // bulid TBN matrix
    vec3 retrievedNormal = texture(normMap,tc).xyz;
    retrievedNormal = retrievedNormal * 2.0 - 1.0;    // convert from RGB space
    vec3 newNormal = tbn * retrievedNormal;
    newNormal = normalize(newNormal);
    return newNormal;
}
void main(void)
{   vec3 N = calcNewNormal();
    . . .  // the rest of the main() is the same as before
}
```

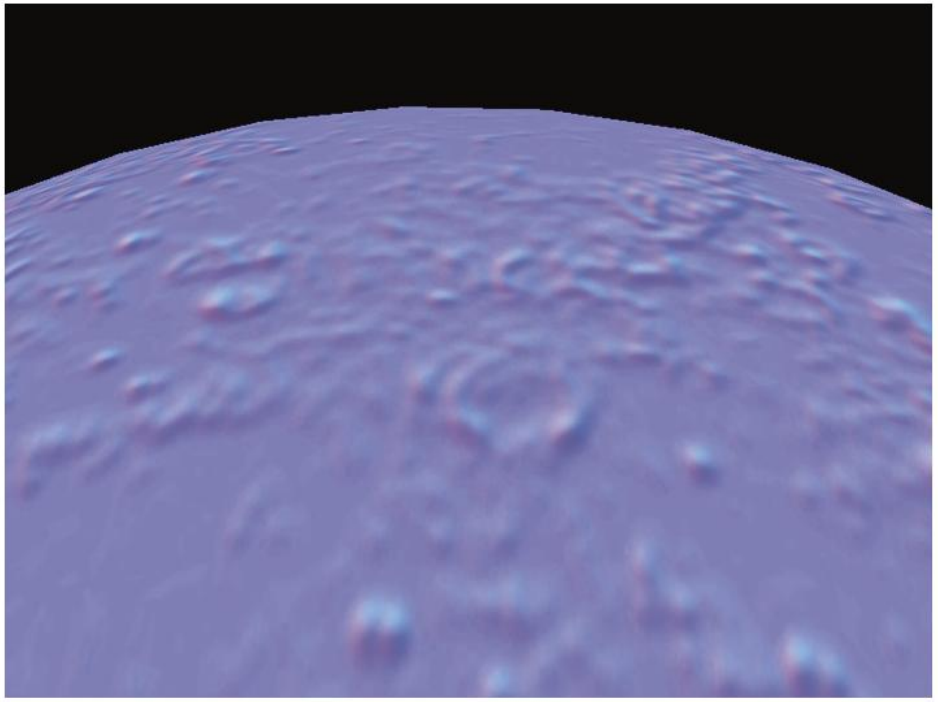
月球例子





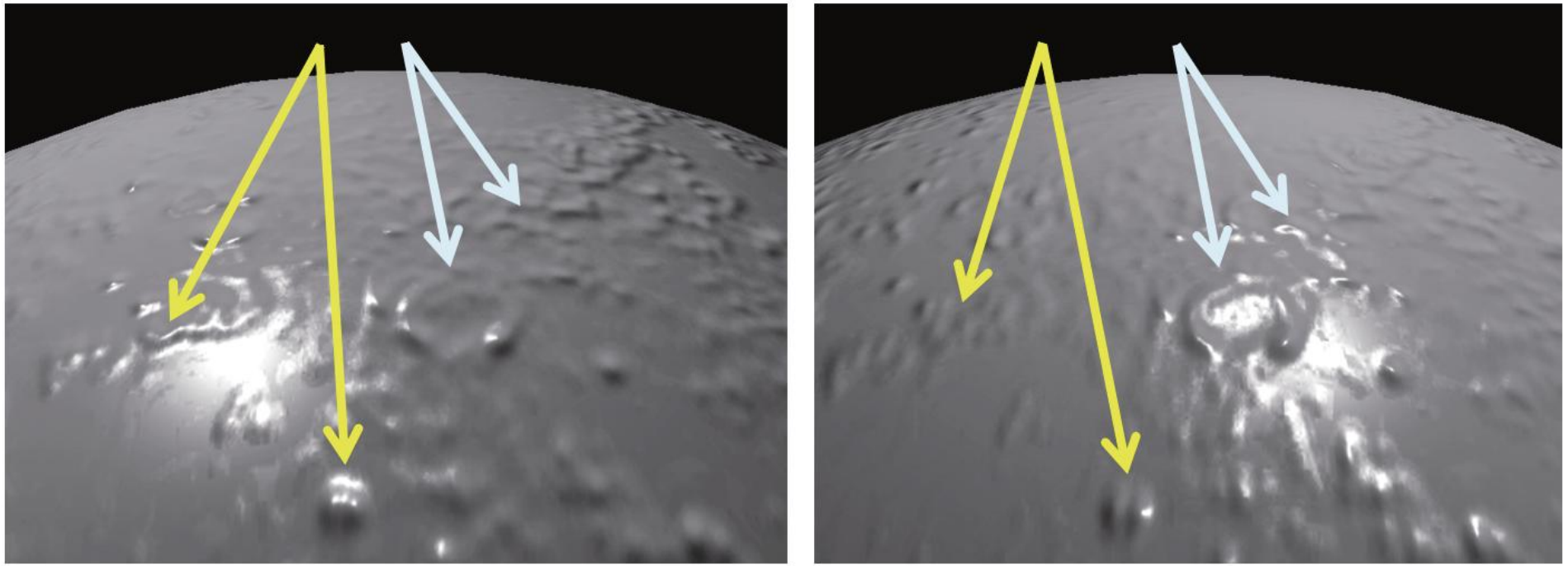
月球例子





使用月面纹理的球体（左）和使用法线图的球体（右）

月球例子



法线贴图对月球的影响

月球例子

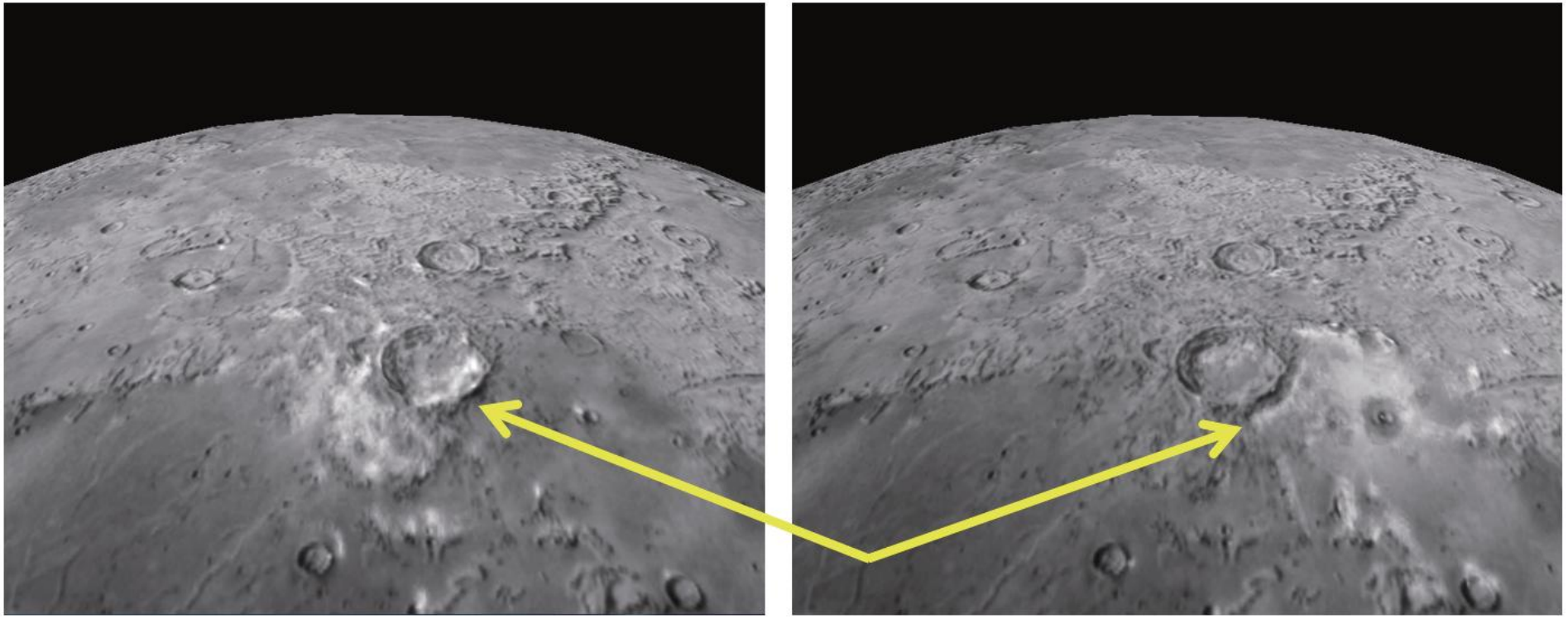
- 结合纹理和法线贴图

Fragment Shader

```
. . .
// variables and structs same as in previous fragment shader, plus:
layout (binding=0) uniform sampler2D s0;   // normal map
layout (binding=1) uniform sampler2D s1;   // texture

void main(void)
{   // computations same as before, until:
    vec3 N = CalcNewNormal();
    vec4 texel = texture(s1,tc);    // standard texture
    . . .
    // reflection computations as before, then blend results:
    fragColor = global.ambient
        +  texel  *  ( light.ambient + light.diffuse * max(cosTheta,0.0)
        + light.specular * pow(max(cosPhi,0.0), material.shininess ) )
}
```

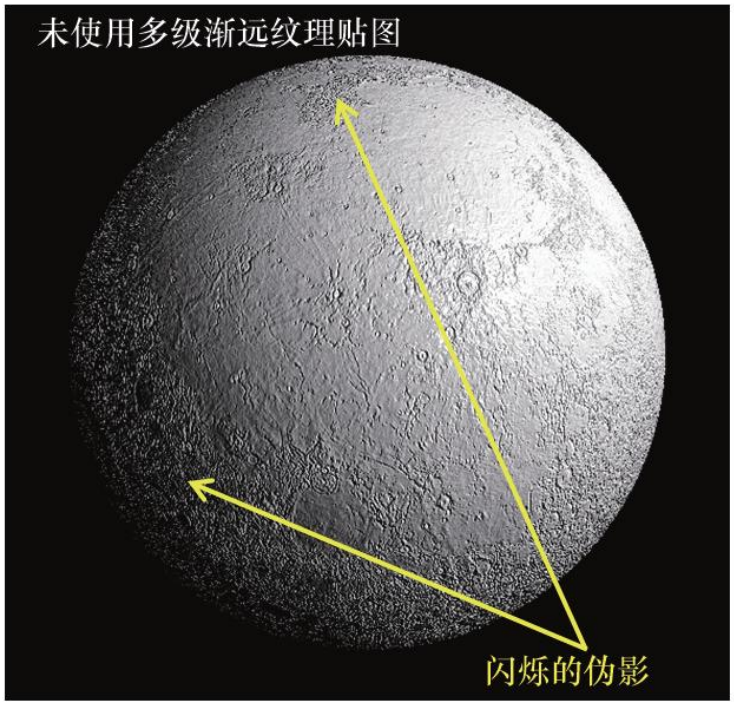
月球例子

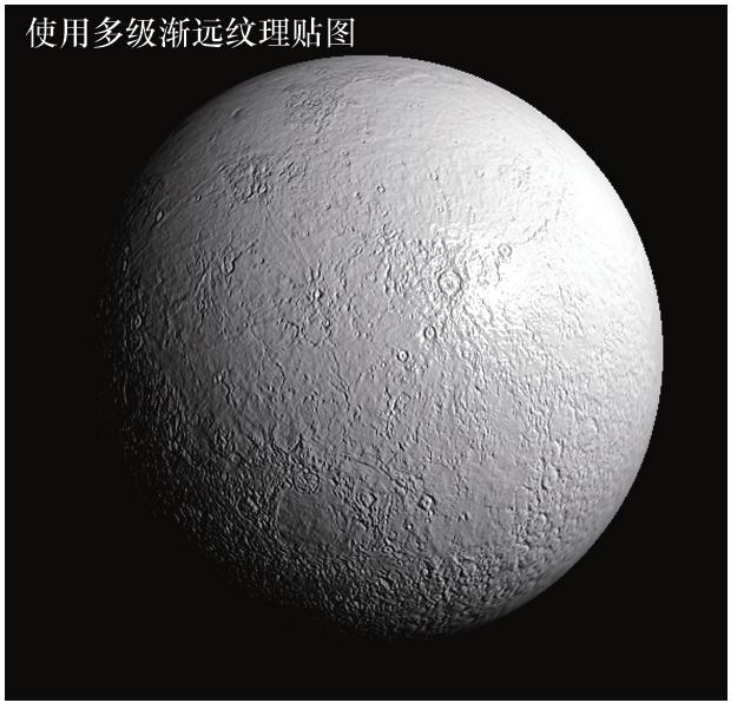
- 结合纹理和法线贴图



纹理加法线贴图，分别从左侧和右侧进行光照

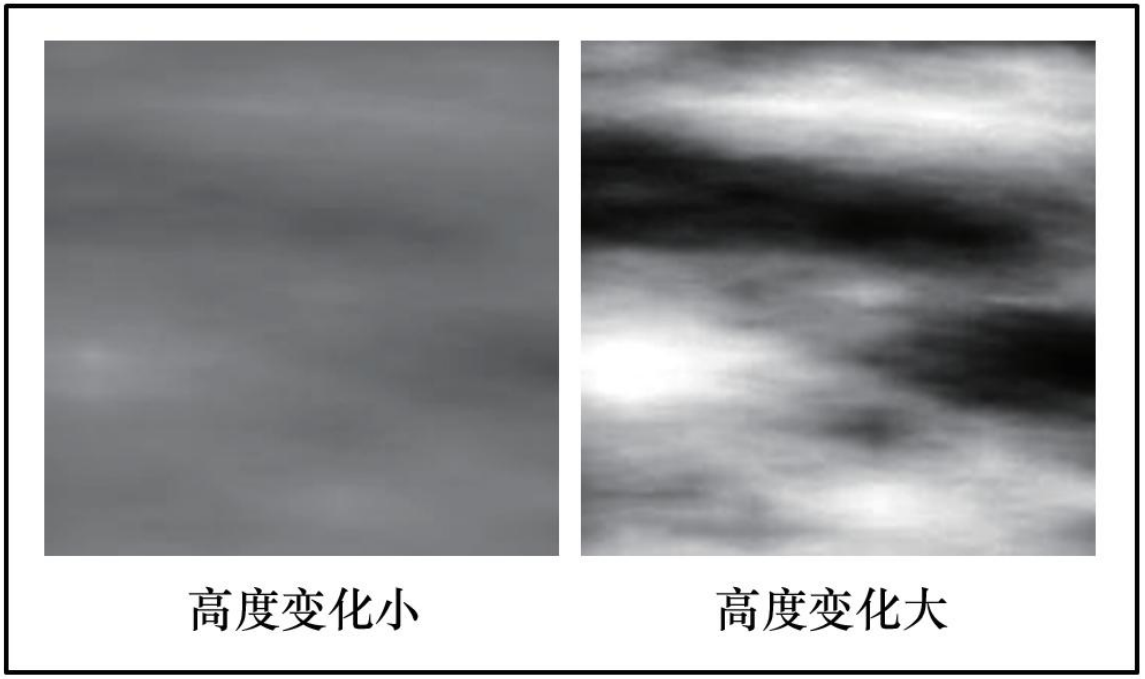
- 法线贴图也可以使用多级渐远纹理贴图改善效果
- 在第 5 章中看到的纹理化产生的“锯齿”伪影，在使用纹理图像进行法线贴图时也会产生
- 尽管在静止的图像中不容易观察到，但是左边的球体（未使用多级渐远纹理贴图）周边有闪烁的伪影





高度贴图

- 使用纹理图像来存储高度值，然后使用该高度值来提升（或降低）顶点位置，也称为**位移贴图**或**置换贴图**
- 含有高度信息的图像称为高度图，使用高度图更改对象的顶点的方法称为高度贴图
- 高度图通常将高度信息编码为灰度颜色：(0,0,0)=黑色=高度低，(1,1,1)=白色=高度高。

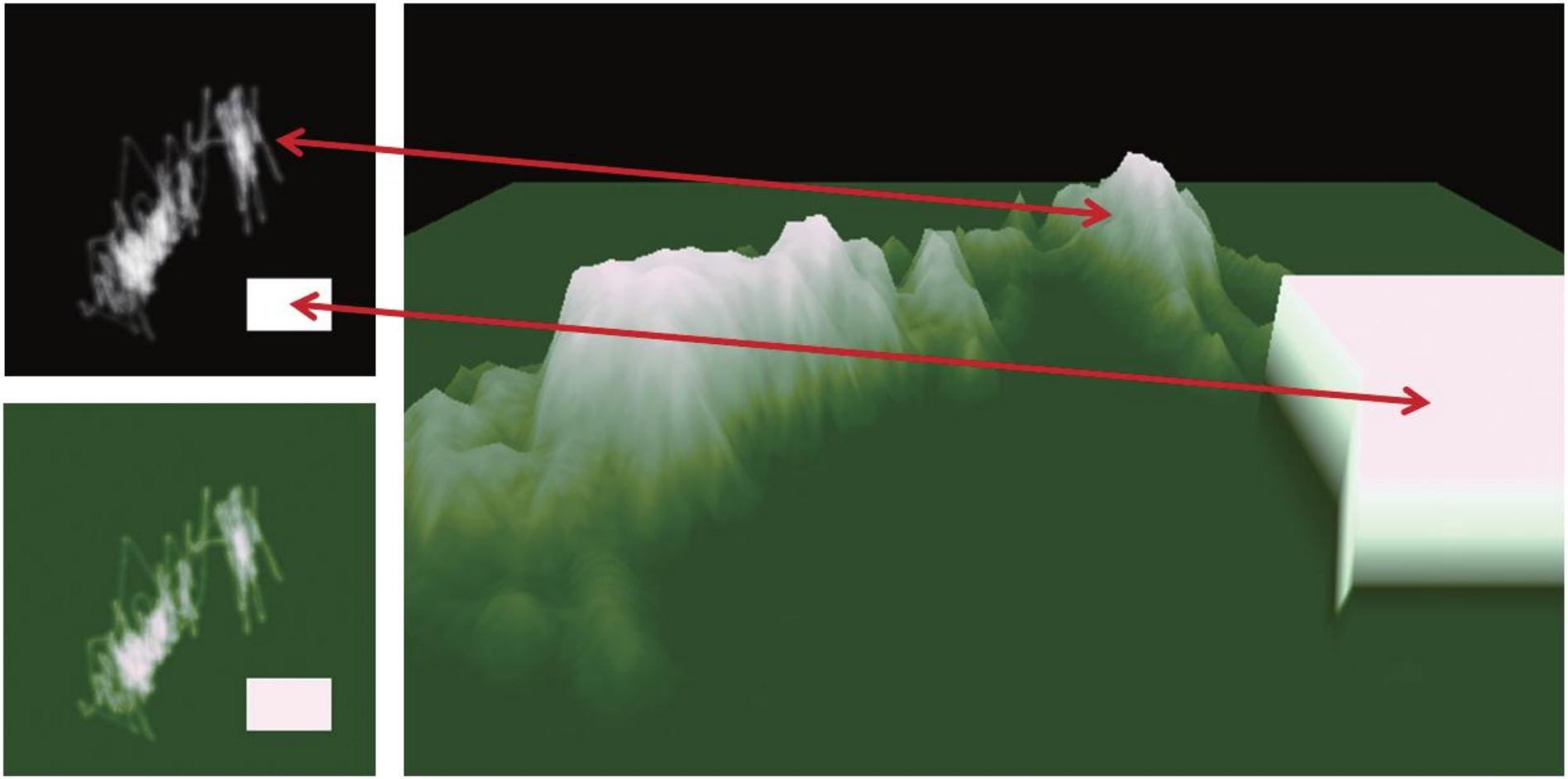


顶点着色器中的高度贴图

```
#version 430
layout (location=0) in vec3 vertPos;
layout (location=1) in vec3 vertNormal;
layout (location=2) in vec2 texCoord;

out vec2 tc;

uniform mat4 mv_matrix;
uniform mat4 proj_matrix;

layout (binding=0) uniform sampler2D t;  // for texture
layout (binding=1) uniform sampler2D h; // for heightmap

void main(void)
{  // "p" is the vertex position altered by the height map.
   // Since the height map is grayscale, any of the color components can be
   // used (we use "r"). Dividing by 5.0 is to adjust the height.
   vec4 p = vec4(vertPos,1.0) + vec4((vertNormal * ((texture(h,texCoord).r)/5.0f)),1.0f);
   tc = tex_coord;
   gl_Position = proj_matrix * mv_matrix * p;
}
```

高度贴图例子

- 地形，在顶点着色器中进行高度贴图



高度贴图的局限性

- 当模型顶点细节级别够高（例如在足够高精度的球体中） 时，改变顶点高度的方法效果很好。
- 当顶点着色器中可用于改变高度的顶点数量很少时，许多像素的高度将无法从高度图中检索，而需要由插值生成，从而将导致表面细节较差。
- 例如，球面的顶点不够多，将月球纹理作为高度贴图时，将不能捕获纹理细节，使用曲面细分着色器 （第12章）将解决这个问题