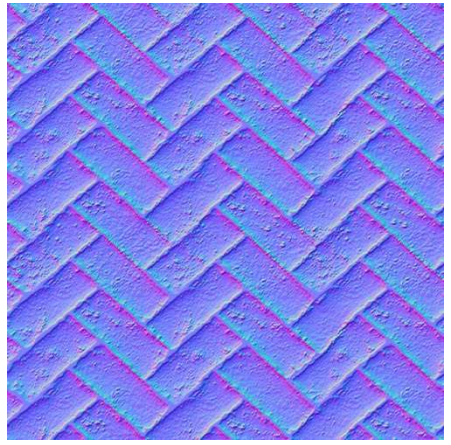
Normal-Mapping

Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu



The Next Step in Bump-Mapping

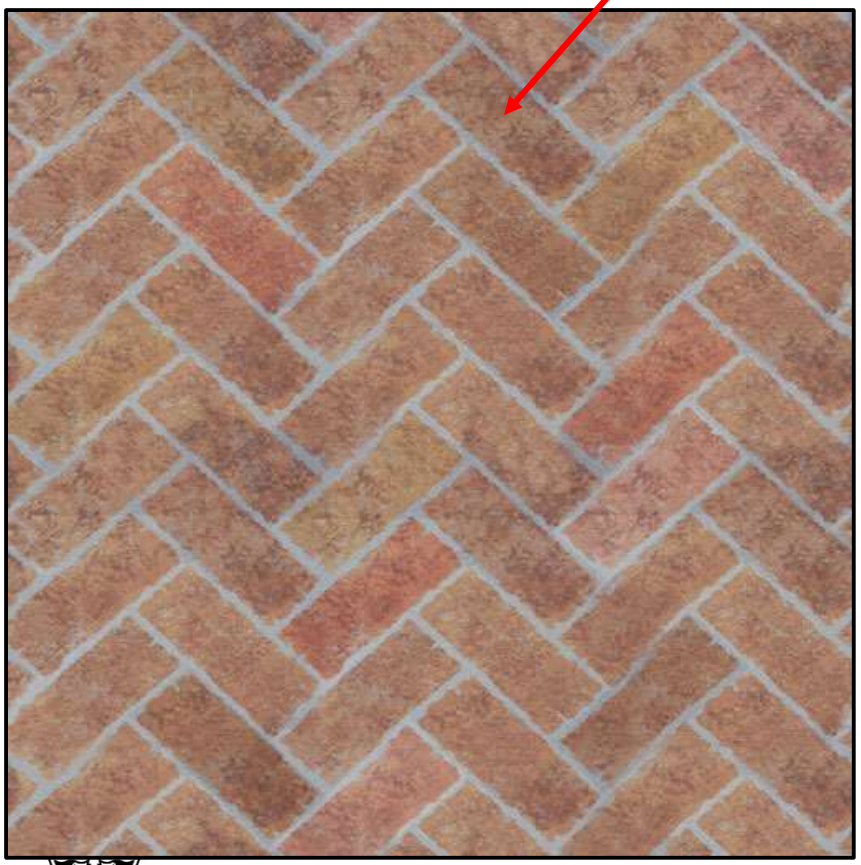
The Scenario:

You want to do bump-mapping. You have a very specific and detailed set of surface normal vectors but don't have an equation that describes them. Yet you would still like to somehow "wrap" the normal vector field around a lower-resolution object so that you can perform good lighting everywhere.

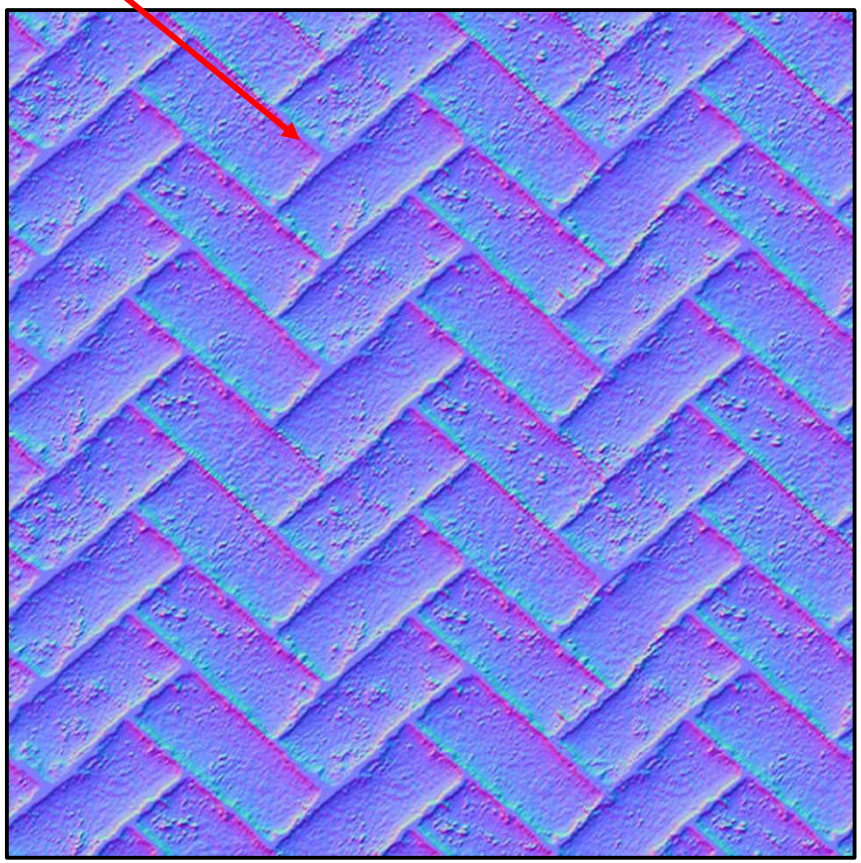
This is a job for ***Normal-Maps!***

What is Normal-Mapping?

Normal-Mapping is a modeling technique where, in addition to you specifying the color texture, you also create a texture image that contains all of the normal vectors on the object



Color Texture



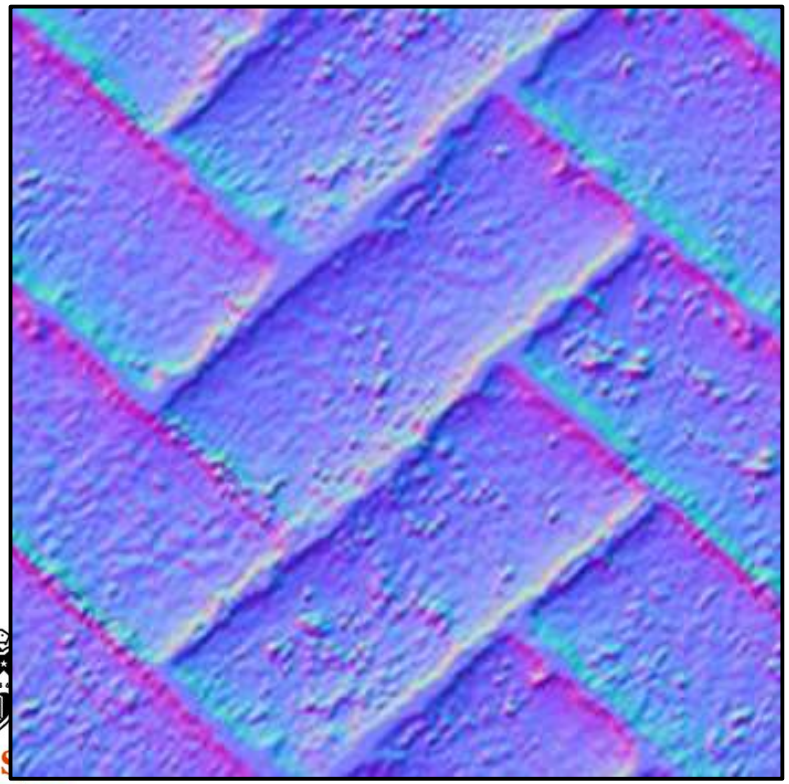
Normal-Map Texture

Color map and normal map provided by Michael Tichenor

How Do You Store a Surface Normal Field in a Texture?

The three components of the normal vector (nx, ny, nz) are mapped into the three color components (red, green, blue) of the texture:

$\begin{Bmatrix} nx \\ ny \\ nz \end{Bmatrix}$ in the range -1. → 1. are placed into the texture's $\begin{Bmatrix} red \\ green \\ blue \end{Bmatrix}$ in the range 0. → 1.



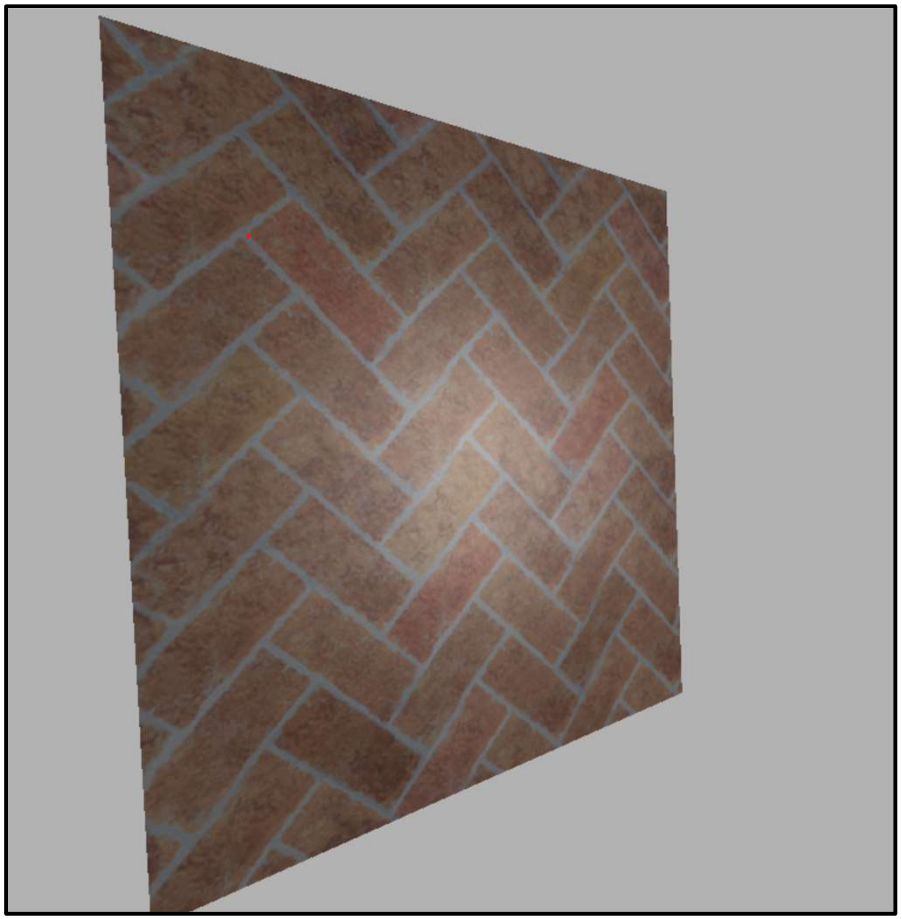
To convert the normal to a color:

$$\begin{Bmatrix} red \\ green \\ blue \end{Bmatrix} = \frac{\begin{Bmatrix} nx \\ ny \\ nz \end{Bmatrix} + \begin{Bmatrix} 1. \\ 1. \\ 1. \end{Bmatrix}}{2.}$$

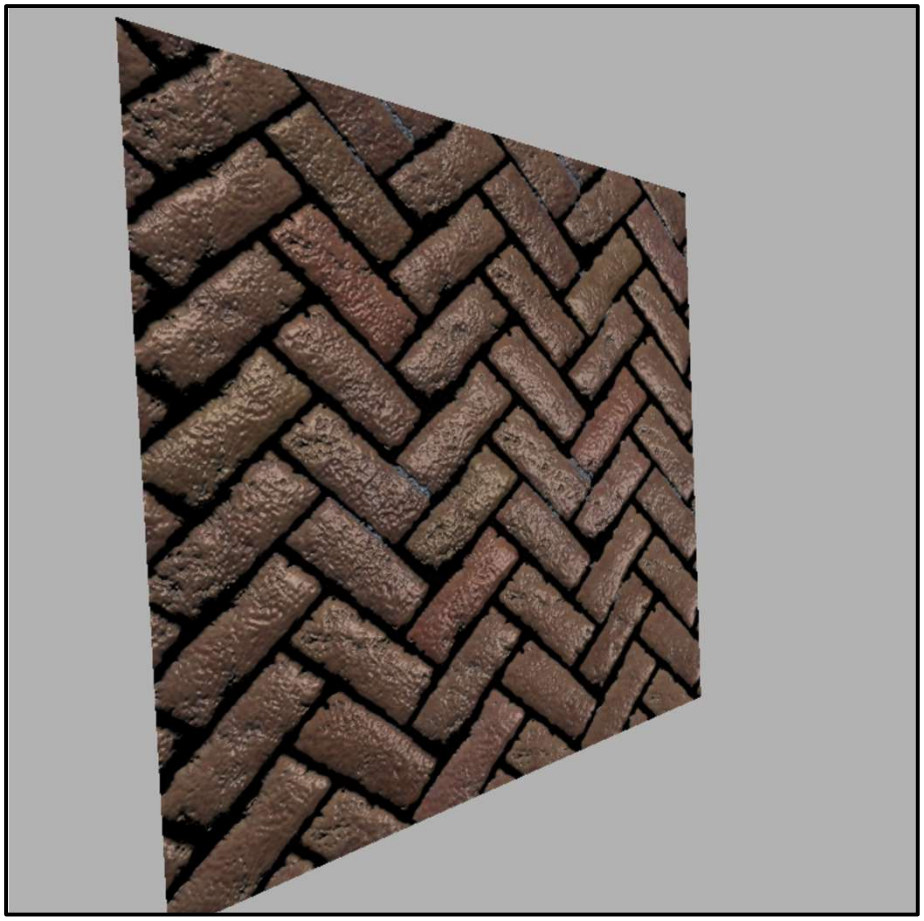
To convert the color back to a normal:

$$\begin{Bmatrix} nx \\ ny \\ nz \end{Bmatrix} = 2. * \begin{Bmatrix} red \\ green \\ blue \end{Bmatrix} - \begin{Bmatrix} 1. \\ 1. \\ 1. \end{Bmatrix}$$

This Gets Us Better Lighting Behavior, While Still Maintaining the Advantages of Bump-Mapping



Ordinary Texture



Normal-Mapping

This Gets Us Better Lighting Behavior, While Still Maintaining the Advantages of Bump-Mapping

Small Specular Shininess

Large Specular Shininess

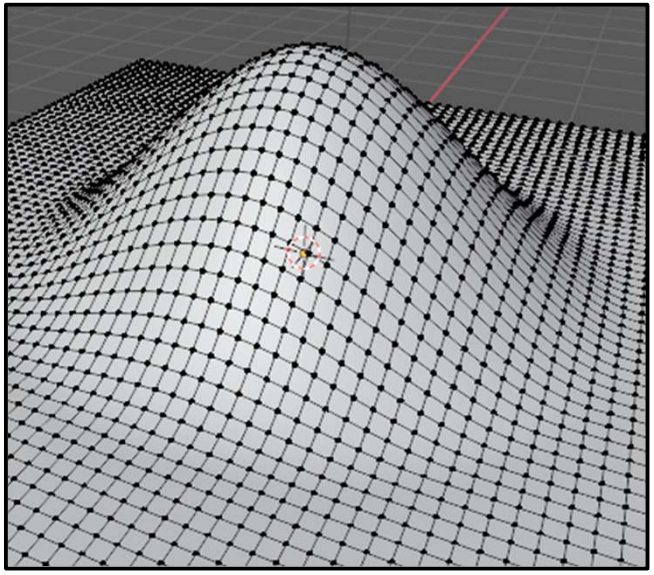
Vertex shader

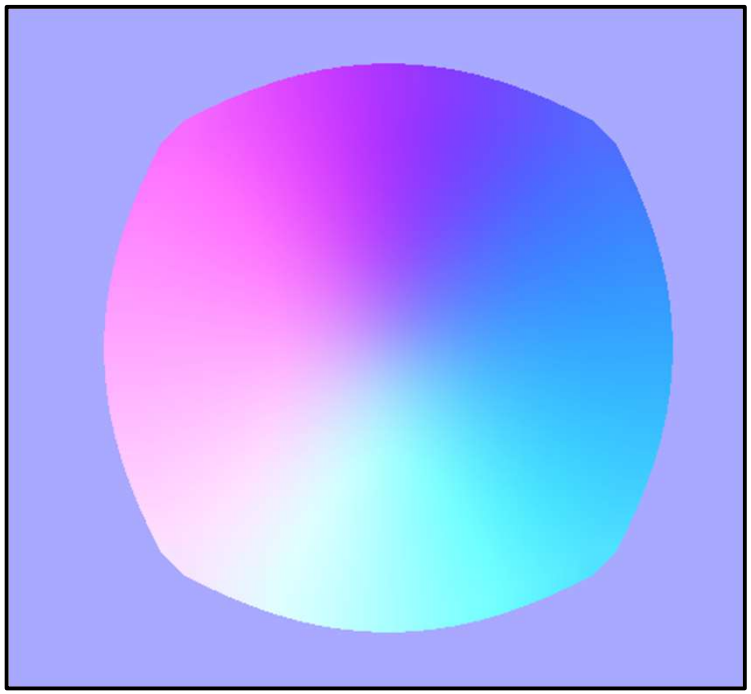
```
#version 330 compatibility

out vec3 vSurfacePosition;
out vec3 vSurfaceNormal;
out vec3 vEyeVector;
out vec2 vST;

void
main( )
{
	vSurfacePosition = (gl_ModelViewMatrix * gl_Vertex).xyz;
	vSurfaceNormal = normalize( gl_NormalMatrix * gl_Normal );
	vEyeVector = vec3( 0., 0., 0. ) – vSurfacePosition;

	vST = gl_MultiTexCoord0.st;

	gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

Fragment shader

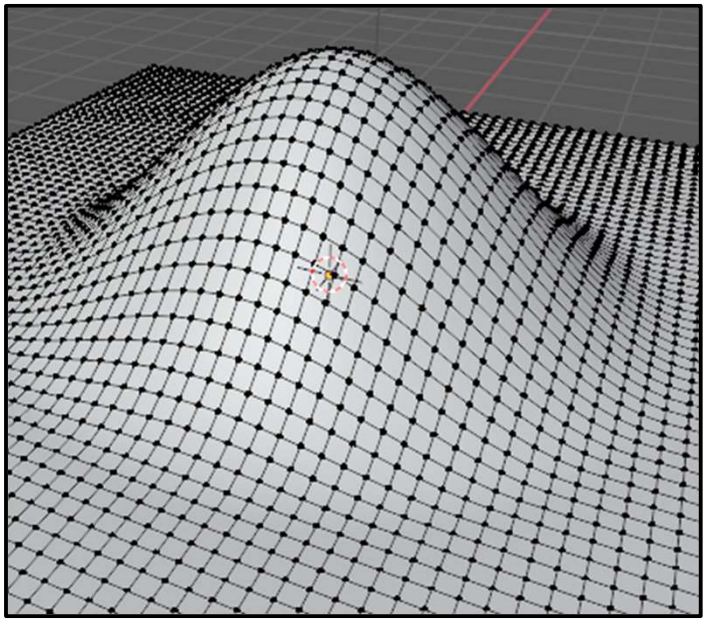
```
#version 330 compatibility

uniform float uKa;
uniform float uKd;
uniform float uKs;
uniform float uShininess;
uniform float uFreq;
uniform sampler2D Color_Map;
uniform sampler2D Normal_Map;

in vec3 vSurfacePosition;
in vec3 vSurfaceNormal;    // not actually using this – just here if we need it
in vec3 vEyeVector;
in vec2 vST;

const vec3 LIGHTPOSITION = vec3( 0., 10., 0. );
const vec3 WHITE = vec3( 1., 1., 1. );


void
main( )
{
        vec3 P = vSurfacePosition;
        vec3 E = normalize( vEyeVector );
        vec3 N = normalize( gl_NormalMatrix * (2.*texture( Normal_Map, uFreq*vST ).xyz - vec3(1.,1.,1.) ) );
        vec3 L = normalize( LIGHTPOSITION – P );

        vec3 Ambient = uKa * texture( Color_Map, uFreq * vST ).rgb;
        float Diffuse_Intensity = dot( N, L );
        vec3 Diffuse = uKd * Diffuse_Intensity * texture( Color_Map, uFreq * vST ).rgb;
        float Specular_Intensity = pow( max( dot( reflect( -L, N ), E ), 0. ), uShininess );
        vec3 Specular = uKs * Specular_Intensity * WHITE;
        gl_FragColor = vec4(Ambient+ Diffuse + Specular, 1. );
}
```

Generating a Normal Map: A Blender Example

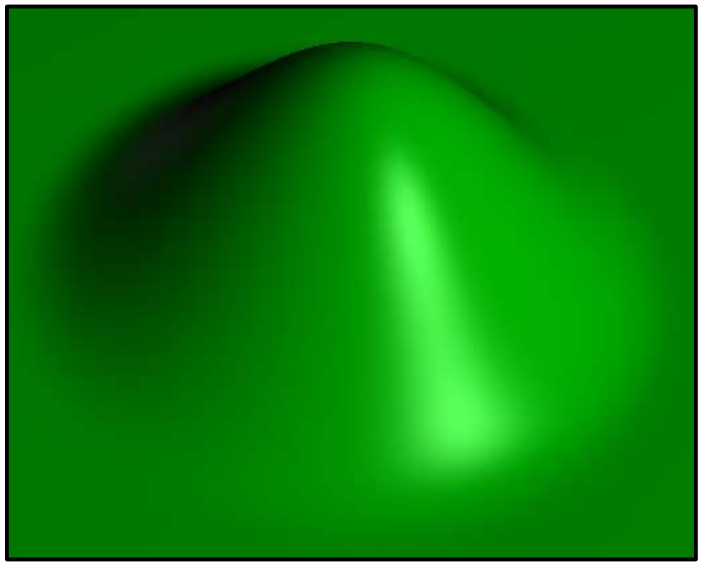


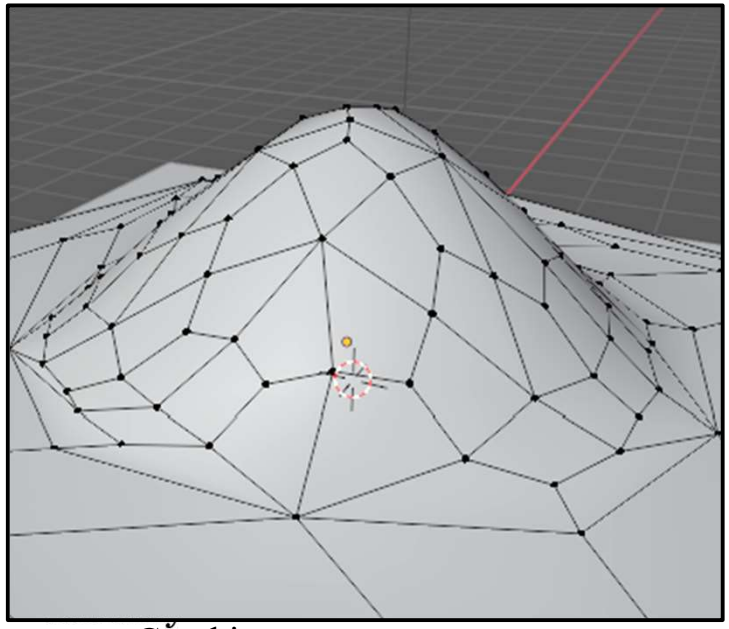


Generating a Normal Map: A Blender Example



8,192 triangles





240 triangles