Bump Mapping

What is Bump-Mapping?

Bump-mapping is the process of creating the illusion of 3D depth by using a manipulated surface normal in the lighting, rather than actually creating the extra surface detail.
The Most Straightforward Type of Bump-Mapping is Height Fields

Definition of Height Fields -- Think of the Pin Box!
terrain.vert

```cpp
#version 330 compatibility
out vec3 vMCposition;
out vec3 vECposition;
out vec2 vST;

void main() {
    vST = gl_MultiTexCoord0.st;
    vMCposition = gl_Vertex .xyz;
    vECposition = ( gl_ModelViewMatrix * gl_Vertex ).xyz;
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

terrain.frag

```cpp
#version 330 compatibility
uniform float uLightX, uLightY, uLightZ;
uniform float uExag;
uniform vec4 uColor;
uniform sampler2D uHgtUnit;
uniform bool uUseColor;
uniform float uLevel1;
uniform float uLevel2;
uniform float uTol;
uniform float uDelta;

in vec3  vMCposition;
in vec3  vECposition;
in vec2 vST;

const float DELTA = 0.001;
const vec3 BLUE = vec3( 0.1, 0.1, 0.5 );
const vec3 GREEN = vec3( 0.0, 0.8, 0.0 );
const vec3 BROWN = vec3( 0.6, 0.3, 0.1 );
const vec3 WHITE = vec3( 1.0, 1.0, 1.0 );

const float LNGMIN = -579240./2.; // in meters, same as heights
const float LNGMAX =  579240./2.;
const float LATMIN = -419949./2.;
const float LATMAX =  419949./2.;
```

Floating-point texture whose .r components contain the heights (in meters)
terrain.frag, II

```glsl
void main() {
    vec2 stp0 = vec2( DELTA, 0.);
    vec2 stp1 = vec2( 0., -DELTA);
    float west = texture2D( uHgtUnit, vST-stp0 ).r;
    float east = texture2D( uHgtUnit, vST+stp0 ).r;
    float south = texture2D( uHgtUnit, vST-stp1 ).r;
    float north = texture2D( uHgtUnit, vST+stp1 ).r;
    vec3 stangent = vec3( 2.*DELTA*(LNGMAX-LNGMIN), 0., uExag*(east - west) );
    vec3 ttangent = vec3( 0., 2.*DELTA*(LATMAX-LATMIN), uExag*(north - south) );
    vec3 normal = normalize( cross(stangent, ttangent) );
    float LightIntensity = dot(normalize(vec3(uLightX,uLightY,uLightZ) - vMCposition), normal);
    if( LightIntensity < 0.1 )
        LightIntensity = 0.1;
    if( uUseColor )
        {
            float here = texture2D( uHgtUnit, vST ).r;
            vec3 color = BLUE;
            if( here > 0. )
                {
                    float t = smoothstep( uLevel1-uTol, uLevel1+uTol, here );
                    color = mix( GREEN, BROWN, t );
                }
            if( here > uLevel1+uTol )
                {
                    float t = smoothstep( uLevel2-uTol, uLevel2+uTol, here );
                    color = mix( BROWN, WHITE, t );
                }
            gl_FragColor = vec4( LightIntensity*color, 1. );
        }
    else
        {
            gl_FragColor = vec4( LightIntensity*uColor.rgb, 1. );
        }
}
```

Terrain Height Bump-mapping: Exaggerating the Height

No Exaggeration

Exaggerated
Terrain Height Bump-mapping: Coloring by Height

No Exaggeration

Exaggerated
Terrain Height Bump-mapping: Even Zooming-in Looks Good

- Portland
- Salem
- Corvallis
- Eugene
- Crater Lake

Terrain Height Bump-Mapping on a Globe

Visualization by Nick Gebbie
The Second Most Straightforward Type of Bump-Mapping is Height Field Equations

This is the coordinate system we will be using. The plane is X-Y with Z pointing up.

Bump-mapping to Create Polar Ripples

In 2D, a slope \( m = \frac{dy}{dx} \). It can be expressed as the vector \([1, m]\).

The normal to the shape is the vector perpendicular to the vector slope:

Note that \([1,m] \cdot [-m,1] = 0\), as it must be.

So, if \( z = -Amp \ast \cos(2\pi x/Pd - 2\pi Time) \), then the scalar slope \( dz/dx \) is:

\[
\frac{dz}{dx} = Amp \ast \frac{2\pi}{Pd} \ast \sin(2\pi x/Pd - 2\pi Time),
\]

and the vector slope is:

Tangent Vector (i.e., slope) = \([1,,0,,Amp \ast \frac{2\pi}{Pd} \ast \sin(2\pi x/Pd - 2\pi Time)]\)
Bump-mapping to Create Polar Ripples

Following the pattern from before, the normal vector is:

\[
\text{Normal Vector} = \begin{bmatrix} -\text{Amp} \times \frac{2\pi}{\text{Pd}} \times \sin\left(\frac{2\pi x}{\text{Pd}} - 2\pi \text{Time}\right), & 0, & 1 \end{bmatrix}
\]

This is true along just the X axis. The trick now is to rotate the normal vector into where we really are. Because we are just talking about a rotation, the transformation is the same as if we were rotating a vertex.

\[
\begin{align*}
N_x' &= N_x \cos\Theta - N_y \sin\Theta = N_x \cos\Theta \\
N_y' &= N_x \sin\Theta + N_y \cos\Theta = N_x \sin\Theta \\
N_z' &= N_z = 1.
\end{align*}
\]

In the final code, you would substitute \( R \) for \( x \) in the slope and normal equations.

(Also note that you could include some exponential decay to make this behave more like real ripples.)

Combining Bump and Cube Mapping

In the image, you can see a comparison between bump mapping and cube mapping on a curved surface. The bump mapping creates more localized ripples, while the cube mapping results in smoother, more uniform ripples across the entire surface.