Bump Mapping

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License
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. You saw this before in RenderMan like this:
The Most Straightforward Type of Bump-Mapping is *Height Fields*
Definition of Height Fields -- Think of the Pin Box!
terrain.vert

```glsl
#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

#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)
void main( )
{
  vec2 stp0 = vec2( DELTA, 0. );
  vec2 st0p = vec2( 0., DELTA );
  float west  = texture2D( uHgtUnit, vST-stp0 ).r;
  float east  = texture2D( uHgtUnit, vST+stp0 ).r;
  float south = texture2D( uHgtUnit, vST-st0p ).r;
  float north = texture2D( uHgtUnit, vST+st0p ).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
Terrain Height Bump-mapping: Coloring by Height

No Exaggeration

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

\[ [-m, 1] \]

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

So, if \( z = -Amp \cdot \cos(\frac{2\pi x}{Pd} - 2\pi \text{Time}) \), then the slope \( \frac{dz}{dx} \) is:

\[ \frac{dz}{dx} = Amp \cdot \frac{2\pi}{Pd} \cdot \sin(\frac{2\pi x}{Pd} - 2\pi \text{Time}) \]

and the vector slope is:

\[ \text{Slope} = [1., 0., Amp \cdot \frac{2\pi}{Pd} \cdot \sin(\frac{2\pi x}{Pd} - 2\pi \text{Time})] \]
Bump-mapping to Create Polar Ripples

Following the pattern from before, the normal vector is:

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

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.

\[
\begin{align*}
\text{Nx'} &= \text{Nx} \times \cos\Theta - \text{Ny} \times \sin\Theta = \text{Nx} \times \cos\Theta \\
\text{Ny'} &= \text{Nx} \times \sin\Theta + \text{Ny} \times \cos\Theta = \text{Nx} \times \sin\Theta \\
\text{Nz'} &= \text{Nz} = 1.
\end{align*}
\]

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