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:

\[ P = P + \text{normalize}(N) \times \text{disp}; \]

\[ N = \text{calculateNormal}(P); \]

The Most Straightforward Type of Bump-Mapping is Height Fields

Terrain fragment shader:

```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 vertex shader:

```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;
}
```
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(normal, vec3(uLightX, uLightY, uLightZ) - vMCposition);
    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.);
    }
}
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.

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 \( z = -Amp \cdot \cos(\frac{2\pi x}{Pd} - 2\pi Time) \), then the slope \( dz/dx \) is:
\[ \frac{dz}{dx} = Amp \cdot \frac{2\pi}{Pd} \cdot \sin(\frac{2\pi x}{Pd} - 2\pi Time) \]

And the vector slope is:
\[ \text{Slope} = [\ 1., 0., Amp \cdot \frac{2\pi}{Pd} \cdot \sin(\frac{2\pi x}{Pd} - 2\pi Time) \ ] \]

Bump-mapping to Create Polar Ripples

Following the pattern from before, the normal vector is:
\[ \text{Normal} = [-Amp \cdot \frac{2\pi}{Pd} \cdot \sin(\frac{2\pi x}{Pd} - 2\pi Time) , 0., 1.] \]

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.

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

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

The problem is that lighting information is in Eye Coordinates, but the bump information is in Surface Local Coordinates!

We need to:
1. Figure out how to convert from one to the other, and,
2. Decide which of light information or bump information gets converted to the other’s coordinate system.

While we are at it, let’s also rename the Surface Local coordinate to \((s,t,h)\) for \((\text{texture}_s, \text{texture}_t, \text{bump}_h)\). This is the same as \((R,T,H)\), but uses terminology that is more bump-specific.
Bump Mapping:

Converting Between Coordinate Systems

Converting from Eye Coordinates to Surface Local Coordinates:

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
\end{bmatrix} =
\begin{bmatrix}
  B_x & B_y & B_z \\
  T_x & T_y & T_z \\
  N_x & N_y & N_z \\
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
\end{bmatrix}
\]

( The "Orange Book" uses this to convert the light vector to Surface Local Coordinates.)

Converting from Surface Local Coordinates to Eye Coordinates:

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
\end{bmatrix} =
\begin{bmatrix}
  B_x & T_x & N_x \\
  B_y & T_y & N_y \\
  B_z & T_z & N_z \\
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
\end{bmatrix}
\]

( I prefer to use this one to convert the bump normal to Eye Coordinates.)

Converting from Eye Coordinates to Surface Local Coordinates:

\[
\begin{bmatrix}
  x \\
  y \\
  z \\
\end{bmatrix} =
\begin{bmatrix}
  T_x & N_x & B_x \\
  T_y & N_y & B_y \\
  T_z & N_z & B_z \\
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
\end{bmatrix}
\]

There are 2 good ways to get the tangent and binormal vectors:

1. Have the Tangent already defined (glman’s Sphere does this)
2. Pick a general rule, e.g., "Tangent ≈ up"
   2a. Use Gram-Schmidt to correctly orthogonalize it wrt the Normal
   2b. Use two cross-products to correctly orthogonalize it wrt the Normal

Note: 2a and 2b give the same result, but some people find 2b easier to understand.

Bump Mapping:

Establishing the Surface Local Coordinate System

// Produce the transformation from Surface coords to Eye coords:

\[
BTNx = \begin{bmatrix}
  B_x \\
  T_x \\
  N_x \\
\end{bmatrix};
BNTy = \begin{bmatrix}
  B_y \\
  T_y \\
  N_y \\
\end{bmatrix};
BTNz = \begin{bmatrix}
  B_z \\
  T_z \\
  N_z \\
\end{bmatrix};
\]

// where the light is coming from:

\[
\text{vec3 LightPosition} = \begin{bmatrix}
  \text{LightX} \\
  \text{LightY} \\
  \text{LightZ} \\
\end{bmatrix};
\]

\[
\text{vec3 ECposition} = \left( \text{gl_ModelViewMatrix} \times \text{gl_Vertex} \right).xyz;
\]

\[
\text{DirToLight} = \text{normalize}(\text{LightPosition} - \text{ECposition});
\]

\[
\text{gl_Position} = \text{gl_ModelViewProjectionMatrix} \times \text{gl_Vertex};
\]

Bump Mapping:

Using the Surface-Local-to-Eye-Coordinate Transform

\[
\text{vec3 ToXyz( vec3 sth )}
\]

\[
\text{sth = normalize(sth) ;}
\]

\[
\text{vec3 xyz ;}
\]

\[
\text{xyz.x = dot(BTNx, sth) ;}
\]

\[
\text{xyz.y = dot(BNTy, sth) ;}
\]

\[
\text{xyz.z = dot(BTNz, sth) ;}
\]

\[
\text{return normalize(xyz) ;}
\]

Bump Mapping:

Using the Surface Local Transform

\[
\text{void main( )}
\]

\[
\text{const float PI = 3.14159265;}
\]

\[
\text{float stx = vST.x ;}
\]

\[
\text{float Swidth = 1. / BumpDensity ;}
\]

\[
\text{float Theight = 1. / BumpDensity ;}
\]

\[
\text{float numInS = floor( st.s / Swidth ) ;}
\]

\[
\text{float numInT = floor( st.t / Theight ) ;}
\]

\[
\text{vec2 center ;}
\]

\[
\text{center.s = numInS * Swidth + Swidth/2. ;}
\]

\[
\text{center.t = numInT * Theight + Theight/2. ;}
\]

\[
\text{vec2 stp = st - center ;}
\]

\[
\text{float \theta = atan( stp.t, stp.s ) ;}
\]

\[
\text{float Swidth = 1. / BumpDensity ;}
\]

\[
\text{float Theight = 1. / BumpDensity ;}
\]

\[
\text{float numInS = floor( st.s / Swidth ) ;}
\]

\[
\text{float numInT = floor( st.t / Theight ) ;}
\]

\[
\text{vec2 center ;}
\]

\[
\text{center.s = numInS * Swidth + Swidth/2. ;}
\]

\[
\text{center.t = numInT * Theight + Theight/2. ;}
\]

\[
\text{vec2 stp = st - center ;}
\]

\[
\text{float \theta = atan( stp.t, stp.s ) ;}
\]

\[
\text{float Swidth = 1. / BumpDensity ;}
\]

\[
\text{float Theight = 1. / BumpDensity ;}
\]

\[
\text{float numInS = floor( st.s / Swidth ) ;}
\]

\[
\text{float numInT = floor( st.t / Theight ) ;}
\]

\[
\text{vec2 center ;}
\]

\[
\text{center.s = numInS * Swidth + Swidth/2. ;}
\]

\[
\text{center.t = numInT * Theight + Theight/2. ;}
\]

\[
\text{vec2 stp = st - center ;}
\]

\[
\text{float \theta = atan( stp.t, stp.s ) ;}
\]
Bump Mapping: Using the Surface Local Transform

```c
vec3 normal = ToXyz( vec3( 0., 0., 1. ) ); // un-bumped normal
if( abs(stp.s) > Swidth/4.  ||  abs(stp.t) > Theight/4. )
{
  normal = ToXyz( vec3( 0., 0., 1. ) );
}
else
{
  if( PI/4. <= theta  &&  theta <= 3.*PI/4. )
  {
    normal = ToXyz( vec3( 0., Height, Theight/4. ) );
  }
  else if( -PI/4. <= theta  &&  theta <= PI/4. )
  {
    normal = ToXyz( vec3( Height, 0., Swidth/4. ) );
  }
  else if( -3.*PI/4. <= theta  &&  theta <= -PI/4. )
  {
    normal = ToXyz( vec3( 0., -Height, Theight/4. ) );
  }
  else if( theta >= 3.*PI/4.  ||  theta <= -3.*PI/4. )
  {
    normal = ToXyz( vec3( -Height, 0., Swidth/4. ) );
  }
}
float intensity = Ambient + (1.-Ambient)*dot(normal, DirToLight);
vec3 litColor = SurfaceColor.rgb * intensity;
gl_FragColor = vec4( litColor, SurfaceColor.a );
```

Changing the Bump Height

Changing the Bump Density

It's Handy to not need a Program-supplied Tangent Vector

Combining Bump and Cube Mapping: A Good Reason to Work in Eye Coordinates instead of Surface Local Coordinates