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.

Displacement-mapped
Bump-mapped

This is a good trick! Displacement-mapping is per-vertex and requires a lot of triangles. Bump-mapping is per-fragment and since you needed to process all those fragments anyway, you might as well do slightly more.

The Most Straightforward Type of Bump-Mapping is Height Fields

Definition of Height Fields -- Think of the Pin Box!

The Vector Cross Product

\[ A \times B = (A_yB_z - A_zB_y, A_zB_x - A_xB_z, A_xB_y - A_yB_x) \]

\[ ||A \times B|| = ||A|| ||B|| \sin \theta \]

Because it produces a vector result (i.e., three numbers), this is also called the Vector Product.
The Perpendicular Property of the Vector Cross Product

The vector \( A \times B \) is both perpendicular to \( A \) and perpendicular to \( B \).

The Right-Hand-Rule Property of the Cross Product

Curl the fingers of your right hand in the direction that starts at \( A \) and heads towards \( B \). Your thumb points in the direction of \( A \times B \).

\[
\begin{align*}
\text{terrain.frag} 1 & \text{ - Terrain Height Bump-mapping: Exaggerating the Height} \\
\text{terrain.frag} 2 & \text{ - Terrain Height Bump-mapping: Coloring by Height}
\end{align*}
\]

Remember that the cross product of two vectors gives you a vector that is perpendicular to both. So, the cross product of two tangent vectors gives you a good approximation to the surface normal.

\[
\begin{align*}
\text{terrain.frag} 1 & \text{ - Terrain Height Bump-mapping: Exaggerating the Height} \\
\text{terrain.frag} 2 & \text{ - Terrain Height Bump-mapping: Coloring by Height}
\end{align*}
\]

It turns out that textures are a great place to “hide” data. They are allowed to be very large and they are fast to lookup values in.
### Terrain Height Bump-mapping: Even Zooming-in Looks Good

![Terrain Height Bump-mapping](image)

**Crater Lake**
- Corvallis
- Salem
- Portland
- Eugene

**Visualization by Nick Gebbie**

### Terrain Height Bump-Mapping on a Globe

Several textures are being mixed onto the surface of the 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.

![Height Field Equations](image)

The plane is X-Y with Z pointing up.

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.

### The Second Most Straightforward Type of Bump-Mapping is Height Field Equations

**Rock Dropped**

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

**Radial-ripple height equation with decay**

If we get the two tangent vectors, then their cross product will give us the surface normal.

$x_tangent = vec3(1.0, 0.0, \frac{\partial z}{\partial x})$

$y_tangent = vec3(0.0, 1.0, \frac{\partial z}{\partial y})$

#### Radial-ripple height equation with decay

$z = A \cos(2\pi Br + C)e^{-Dr}$

(normal = $x_tangent \times y_tangent$)

If we can get the two tangent vectors, then their cross product will give us the surface normal.

$x_tangent = vec3(1.0, 0.0, \frac{\partial z}{\partial x})$

$y_tangent = vec3(0.0, 1.0, \frac{\partial z}{\partial y})$

#### Radial-ripple height equation with decay

$\frac{\partial z}{\partial r} = -A \sin(2\pi Br + C)(2\pi B)e^{-Dr} + A \cos(2\pi Br + C)(-D)e^{-Dr}$

#### Radial-ripple height equation with decay

$2r \frac{\partial r}{\partial x} = 2x$

$2r \frac{\partial r}{\partial y} = 2y$

$\frac{\partial r}{\partial x} = \frac{x}{r}$

$\frac{\partial r}{\partial y} = \frac{y}{r}$

#### Radial-ripple height equation with decay

$\theta$, $\phi$, and $\psi$ are actually the cosine and one of the polar angle.

### The Second Most Straightforward Type of Bump-Mapping is Height Field Equations

$z = A \cos(2\pi Br + C)e^{-Dr}$

(normal = $x_tangent \times y_tangent$)

If we can get the two tangent vectors, then their cross product will give us the surface normal.

$x_tangent = vec3(1.0, 0.0, \frac{\partial z}{\partial x})$

$y_tangent = vec3(0.0, 1.0, \frac{\partial z}{\partial y})$

#### Radial-ripple height equation with decay

$z = A \cos(2\pi Br + C)e^{-Dr}$

(normal = $x_tangent \times y_tangent$)

If we can get the two tangent vectors, then their cross product will give us the surface normal.

$x_tangent = vec3(1.0, 0.0, \frac{\partial z}{\partial x})$

$y_tangent = vec3(0.0, 1.0, \frac{\partial z}{\partial y})$

#### Radial-ripple height equation with decay

$\frac{\partial z}{\partial r} = -A \sin(2\pi Br + C)(2\pi B)e^{-Dr} + A \cos(2\pi Br + C)(-D)e^{-Dr}$

#### Radial-ripple height equation with decay

$2r \frac{\partial r}{\partial x} = 2x$

$2r \frac{\partial r}{\partial y} = 2y$

$\frac{\partial r}{\partial x} = \frac{x}{r}$

$\frac{\partial r}{\partial y} = \frac{y}{r}$

#### Radial-ripple height equation with decay

$\theta$, $\phi$, and $\psi$ are actually the cosine and one of the polar angle.

### The ripples Bump-Map Shader

**ripples.glib**

```glib
#OpenGL GLIB
Perspective 70
LookAt 0 0 0 0 1 0

Vertex ripples.vert
Fragment ripples.frag
Program: Ripples

uTime <0, 0, 10>
uPd <0.2, 1, 1.5>

QuadXY -.1 5.
```

### The ripples Bump-Map Shader

**ripples.glib**

```glib
#OpenGL GLIB
Perspective 70
LookAt 0 0 0 0 1 0

Vertex ripples.vert
Fragment ripples.frag
Program: Ripples

uTime <0, 0, 10>
uPd <0.2, 1, 1.5>

QuadXY -.1 5.
```
#version 330 compatibility
out vec3 vMCposition;
out vec3 vECposition;

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

uniform float uTime;
uniform float uAmp0, uAmp1;
uniform float uPhaseShift;
uniform float uPd;
uniform float uLightX, uLightY, uLightZ;
uniform vec4 uColor;
in vec3 vMCposition;
in vec3 vECposition;

const float TWOPI = 2.*3.14159265;
const vec3 C0 = vec3( -2.5, 0., 0. );
const vec3 C1 = vec3(  2.5, 0., 0. );

void main()
{
    float rad0 = length( vMCposition - C0 );
    float H0   = -uAmp0 * cos( TWOPI*rad0/uPd - TWOPI*uTime );
    float rad1 = length( vMCposition - C1 );
    float H1   = -uAmp1 * cos( TWOPI*rad1/uPd - TWOPI*uTime - uPhaseShift );
    float u = -uAmp0 * (TWOPI/uPd) * sin( TWOPI*rad0/uPd - TWOPI*uTime );
    float v = 0.;
    float w = 1.;
    
    float ang = atan( vMCposition.y - C0.y, vMCposition.x - C0.x );
    float up = dot( vec2(u,v), vec2(cos(ang), -sin(ang)) );
    float vp = dot( vec2(u,v), vec2(sin(ang),  cos(ang)) );
    float wp = 1.;
    u = -uAmp1 * (TWOPI/uPd) * sin( TWOPI*rad1/uPd - TWOPI*uTime - uPhaseShift );
    v = 0.;
    ang = atan( vMCposition.y - C1.y, vMCposition.x - C1.x );
    up += dot( vec2(u,v), vec2(cos(ang), -sin(ang)) );
    vp += dot( vec2(u,v), vec2(sin(ang),  cos(ang)) );
    wp += 1.;
    
    vec3 normal = normalize( vec3( up, vp, wp ) );
    float LightIntensity = abs( dot( normalize(vec3(uLightX,uLightY,uLightZ) - vECposition), normal ) );
    if( LightIntensity < 0.1 )
        LightIntensity = 0.1;
    gl_FragColor = vec4( LightIntensity*uColor.rgb, uColor.a );
}