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 optimization! Displacement-mapping requires a lot of triangles, bump-mapping doesn’t.

The Most Straightforward Type of Bump-Mapping is Height Fields

terrain.vert

#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 = 4199949./2.;

texture coordinate that has its "red" channel contain the heights (in meters)

Floating-point texture whose .r components contain the heights (in meters)

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.
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. );
}
}
The Second Most Straightforward Type of Bump-Mapping is Height Field Equations

[Diagram of Rock Dropped]

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

The Radial-Ripple height equation with decay is:

\[ z = A \cos(2\pi B r + C) e^{-D} \]

The normal of the surface can be calculated as the cross product of the tangent vectors:

\[ \text{normal} = \text{x tangent} \times \text{y tangent} \]

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

\[ \text{x tangent} = \text{vec}(1.0, 0.0, \frac{\partial z}{\partial x}) \]
\[ \text{y tangent} = \text{vec}(0.0, 1.0, \frac{\partial z}{\partial y}) \]

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

The cross product of the tangent vectors gives the surface normal:

\[ \text{normal} = x_tangent \times y_tangent \]

The surface normal is calculated as the cross product of the tangent vectors.

You can sum the individual height field equations and get the same result as summing the height field displacements.

Combining Bump and Cube Mapping