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

#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.vert

#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.;

terrain.frag, I

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. );
    }
    }

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.

Several textures are being mixed onto the surface of the globe.
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.

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

Radial-ripple equation with height decay

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

\[
\begin{align*}
\frac{\partial n}{\partial x} &= \sin(2\pi Br + C)(2\pi B)e^{-Dr} + \cos(2\pi Br + C)(-D)e^{-Dr} \\
\frac{\partial n}{\partial y} &= \sin(2\pi Br + C)(2\pi B)e^{-Dr} + \cos(2\pi Br + C)(-D)e^{-Dr}
\end{align*}
\]

Cross product of the tangent vectors gives the surface normal.

\[
\frac{\partial x}{\partial r} = 2x, \quad \frac{\partial y}{\partial r} = 2y, \quad \frac{\partial z}{\partial r} = z
\]

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

Combining Bump and Cube Mapping