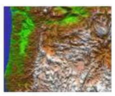
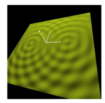
Bump Mapping



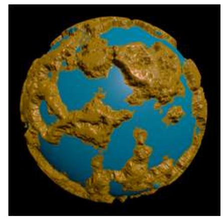
Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu



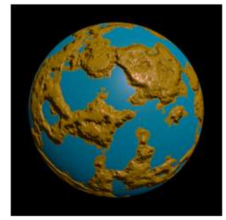
What is Bump-Mapping?

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.

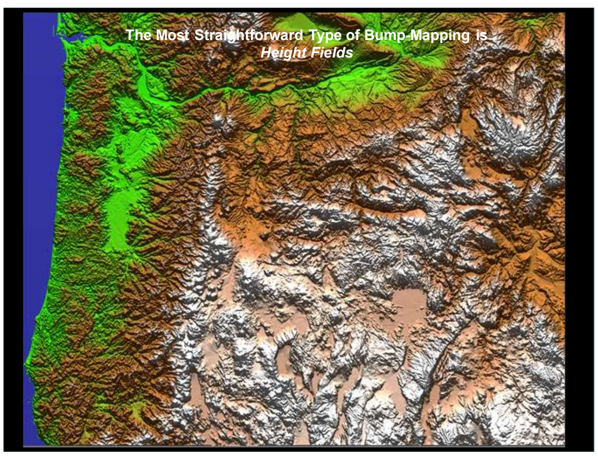
Displacement-mapped



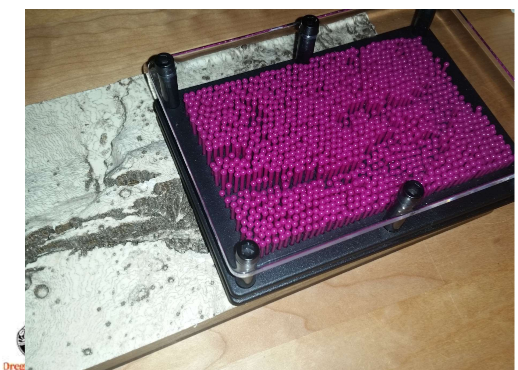
Bump-mapped



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.



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

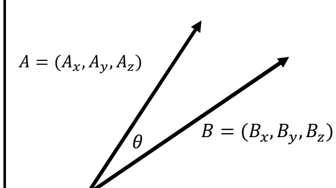


terrain.vert

```
#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;
}
```


The Vector Cross Product



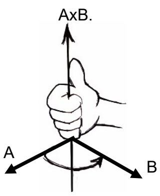
$$A \times B = (A_y B_z - A_z B_y, A_z B_x - A_x B_z, A_x B_y - A_y B_x)$$

$$\|A \times B\| = \|A\| \|B\| \sin\theta$$

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 AxB.

terrain.frag, I


```
#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** component actually contains the heights (in meters)

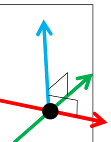
It turns out that textures are a great place to "hide" data. They are allowed to be very large and they are fast to lookup values in.

terrain.frag, II


```
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. );
        }
}
```

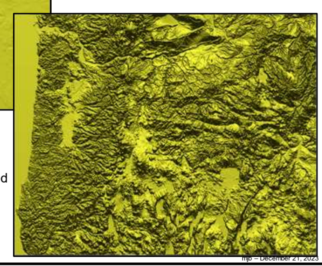


Remember that the cross product of two vectors gives you a vector that is perpendicular to both. So, the cross product of two tangent vectors gives you a good approximation to the surface normal.

Terrain Height Bump-mapping: Exaggerating the Height

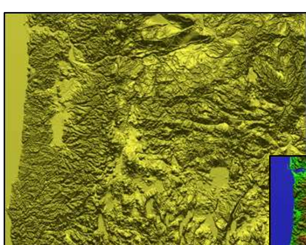
No Exaggeration

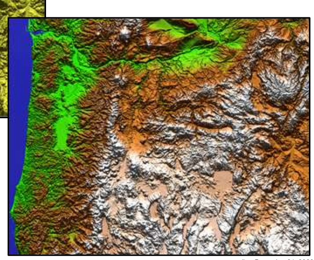
This entire geometry consists of just a single quadrilateral!



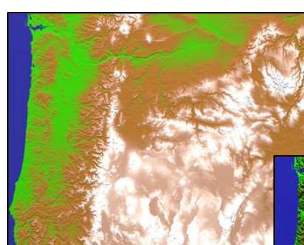
Exaggerated

Terrain Height Bump-mapping: Coloring by Height

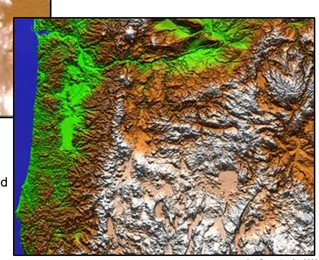




Terrain Height Bump-mapping: Coloring by Height



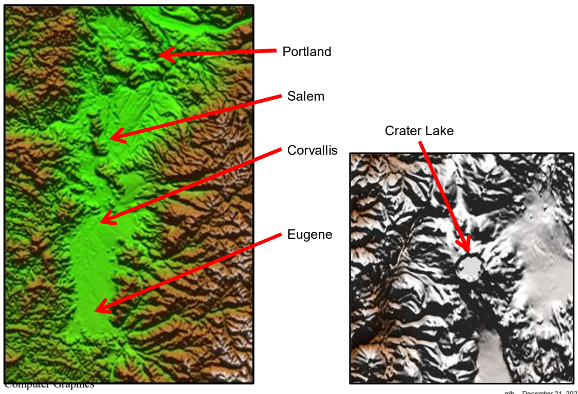
No Exaggeration



Exaggerated

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

13

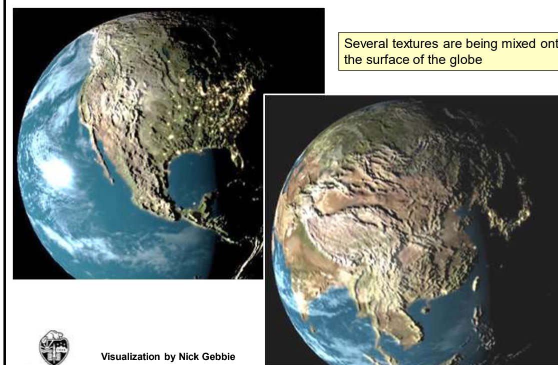


Computer Graphics

mjb – December 21, 2023

Terrain Height Bump-Mapping on a Globe

14



Several textures are being mixed onto the surface of the globe

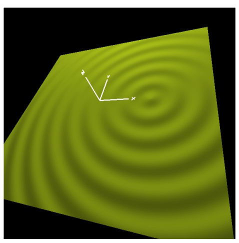
Visualization by Nick Gebbie

Oregon State University
Computer Graphics

mjb – December 21, 2023

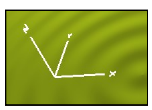
The Second Most Straightforward Type of Bump-Mapping is *Height Field Equations*

15



Rock Dropped

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



Oregon State University
Computer Graphics

mjb – December 21, 2023

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

16

$$z = A\cos(2\pi Br + C)e^{-Dr}$$ Radial-ripple height equation with decay

$$normal = xtangent \;\; \text{x} \;\; ytangent$$ If we can get the two tangent vectors, then their **cross product** will give us the surface normal

$$xtangent = vec3(1., 0., \frac{\partial z}{\partial x}) \qquad ytangent = vec3(0., 1., \frac{\partial z}{\partial y})$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial r}\frac{\partial r}{\partial x} \qquad \frac{\partial z}{\partial y} = \frac{\partial z}{\partial r}\frac{\partial r}{\partial y}$$

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

$$r^2 = x^2 + y^2 \qquad 2r\frac{\partial r}{\partial x} = 2x \qquad \frac{\partial r}{\partial x} = \frac{x}{r}$$

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

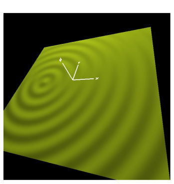
(Note: x/r and y/r are actually the cosine and sine of the polar angle.)

Oregon State University
Computer Graphics

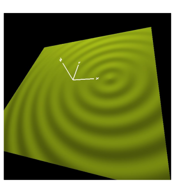
mjb – December 21, 2023

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

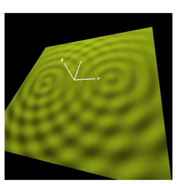
17



Rock A Dropped



Rock B Dropped



Both Rocks Dropped

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

Oregon State University
Computer Graphics

mjb – December 21, 2023

The *ripples* Bump-Map Shader

18

ripples.glib

```
##OpenGL GLIB

Perspective 70
LookAt 0 0 8  0 0 0  0 1 0


Vertex   ripples.vert
Fragment ripples.frag
Program  Ripples                        \
        uLightX <-10.  0. 10.0>         \
        uLightY <-10. 10. 10.0>         \
        uLightZ <-10. 10. 10.0>         \
        uColor {0.7 0.8 0.1 1.}         \
        uTime <0. 0. 10.>               \
        uPd <.2 1. 1.5>                 \
        uAmp0 <0. .05 .05>              \
        uAmp1 <0. 0. .05>               \
        uPhaseShift <0. 0. 6.28>

QuadXY  -.1  5.
```

Oregon State University
Computer Graphics

mjb – December 21, 2023

The *ripples* Bump-Map Shader

ripples.vert

```
#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;
}
```


The *ripples* Bump-Map Shader

ripples.frag, I

```
#version 330 compatibility

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 );

        float u = -uAmp0 * (TWOPI/uPd) * sin( TWOPI*rad0/uPd - TWOPI*uTime );
        float v = 0.;
        float w = 1.;
```


The *ripples* Bump-Map Shader

ripples.frag, II

```
        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 );
}
```


Combining Bump and Cube Mapping