

Generalized Bump-mapping with Surface Local Coordinates



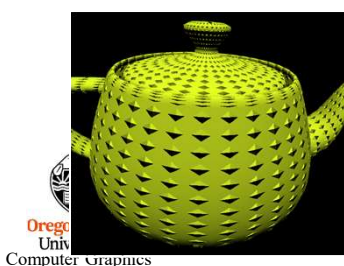
This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/)



Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu



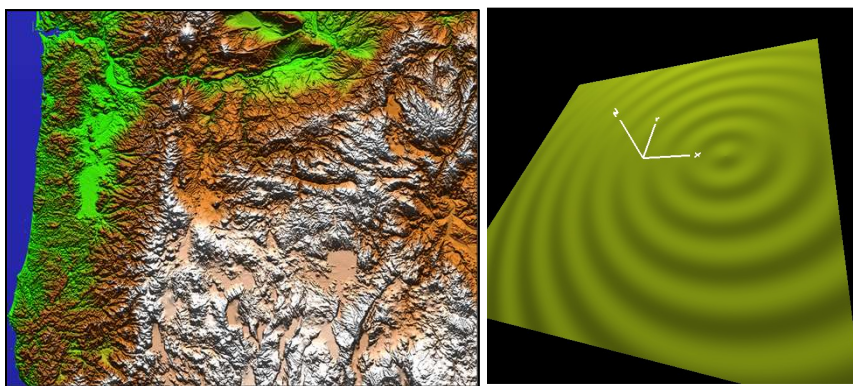
Oregon State University
Computer Graphics



SurfaceLocalCoordinates.pptx

mjb - December 24, 2023

The Most Straightforward Types of Bump-Mapping are Height Fields *Why?*



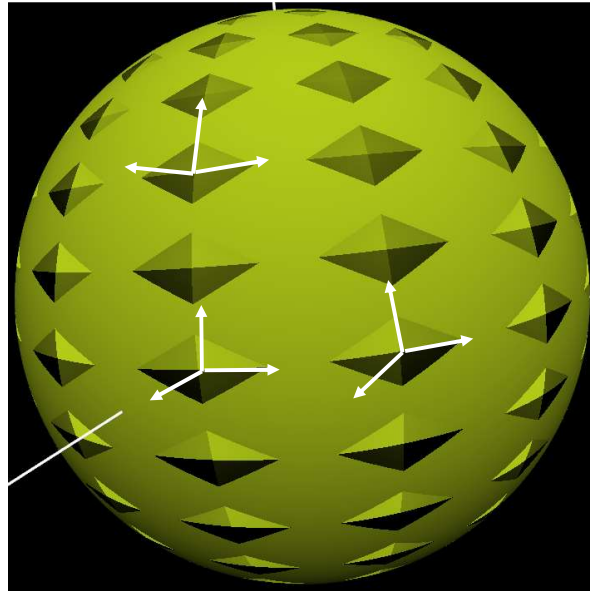
Height Field bump-mapping is straightforward because the underlying coordinate system is constant. Each fragment's Z points up, each fragment's X points right, etc. Thus, the tangent vectors always involve $\frac{dz}{dx}$ and $\frac{dz}{dy}$.

Oregon State University
Computer Graphics

mjb - December 24, 2023

What if that is not the case? Here, the coordinate system is constantly changing, depending on where you are on the sphere

3



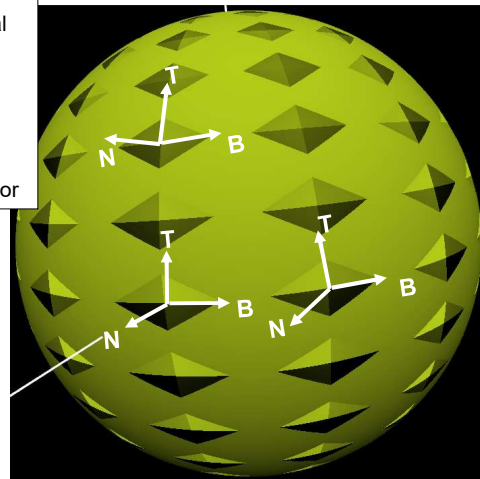
mjb - December 24, 2023

This is referred to as *Surface Local Coordinates*

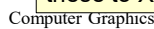
4

To call these moving axes X-Y-Z would be confusing. Rather than X-Y-Z, Surface Local Coordinates are **B-T-N**:

- N is the surface Normal vector, which we usually know already
- T is a Tangent vector
- B is the Bitangent, the other tangent vector



We will assume that we know the Normal everywhere because of how the shape was modeled. Now, how do we find T and B? And, how do we convert these to X-Y-Z?



mjb - December 24, 2023

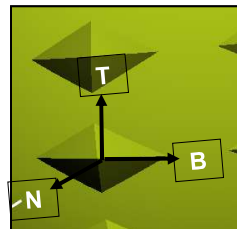
Generalized Bump Mapping: A Problem

5

The problem is that we need to do lighting, but the lighting needs to be done in X-Y-Z, *but* the bump information is in B-T-N!

We need to:

1. Figure out how to determine T and B, and,
2. Figure out how to convert B-T-N coordinates to X-Y-Z for lighting



We will refer to the coordinates in the B-T-N system as (b,t,n) .



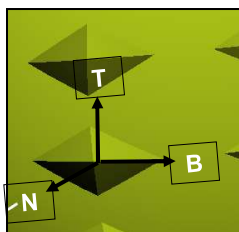
mjb - December 24, 2023

Bump Mapping: Establishing the Surface Local Coordinate System

6

We need a second piece of information: Pick a general rule, e.g., “Tangent \approx up $(0.,1.,0.)$ ”
We then have two choices:

- a. Use two cross-products to correctly orthogonalize it wrt the Normal
- b. Use the Gram-Schmidt rule to correctly orthogonalize it wrt the Normal



// the vectors B-T-N form an X-Y-Z-looking
// right handed coordinate system:

```
vec3 N = normalize( gl_NormalMatrix * gl_Normal );
vec3 Tg, T;      // Tguess and corrected T
vec3 B;
```

#define CROSS_PRODUCT_METHOD

```
#ifdef CROSS_PRODUCT_METHOD
Tg = vec3( 0.,1.,0.);      // guess at T
B = normalize( cross(Tg,N) ); // correct B
T = normalize( cross(N,B) ); // corrected T
#endif
```

```
#ifdef GRAM_SCHMIDT_METHOD
Tg = vec3( 0.,1.,0.);      // guess at T
float d = dot( Tg, N );
T = normalize( Tg - d*N );  // corrected T
B = normalize( cross(T,N) ); // correct B
#endif
```



mjb - December 24, 2023


Cross Product Orthogonalization 7

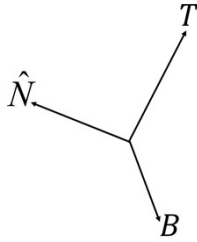
```
vec3 Tg = vec3( 0.,1.,0.); // initial guess
vec3 B = normalize(cross(Tg,N) );
vec3 T = normalize(cross(N,B) );
```


1 Given that **N** is correct, how do we change **T_g** to be exactly perpendicular to **N** ?

3 Take the cross product of **N** and **B** to get a **T** vector that is perpendicular to both

2 Take the cross product of **T_g** and **N** to get a **B** vector that is perpendicular to both







mjb - December 24, 2023

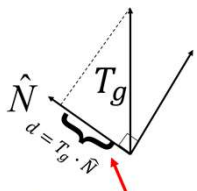
Gram-Schmidt Orthogonalization 8

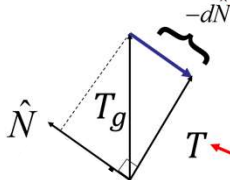
```
vec3 Tg = vec3( 0.,1.,0.); // initial guess
float d = dot( Tg, N );
vec3 T = normalize( Tg - d*N );
vec3 B = normalize(cross(T,N) );
```

1 Given that **N** is correct, how do we change **T_g** to be exactly perpendicular to **N**?

3 How much of **T_g** do we need to get rid of so that *none* of it is in the same direction as **N**?

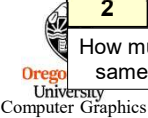
2 How much of **T_g** is in the same direction as **N**?





4 The resulting **T** is perpendicular to **N**

$$T = T_g - d\hat{N} = T_g - (T_g \cdot \hat{N})\hat{N}$$



mjb - December 24, 2023

Bump Mapping: Converting Between Coordinate Systems

9

Converting from X-Y-Z to b-t-n:

$$\begin{Bmatrix} b \\ t \\ n \end{Bmatrix} = \begin{bmatrix} B_x & B_y & B_z \\ T_x & T_y & T_z \\ N_x & N_y & N_z \end{bmatrix} \begin{Bmatrix} x \\ y \\ z \end{Bmatrix}$$

Converting from b-t-n to X-Y-Z:

$$\begin{Bmatrix} x \\ y \\ z \end{Bmatrix} = \begin{bmatrix} B_x & T_x & N_x \\ B_y & T_y & N_y \\ B_z & T_z & N_z \end{bmatrix} \begin{Bmatrix} b \\ t \\ n \end{Bmatrix}$$



I prefer to use the second one so we can do lighting in X-Y-Z like we are used to doing.

mjb - December 24, 2023

Generalized Bump Mapping: Establishing the Surface Local Coordinate System

10

Vertex shader:

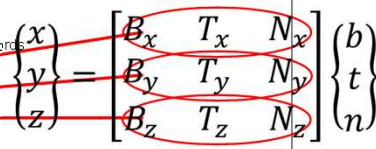
```
#version 330 compatibility
uniform vec3 uLightPosition;

out vec2  vST;           // texture coords
out vec3  vN;           // normal vector
out vec3  vL;           // vector from point to light
out vec3  vE;           // vector from point to eye
out vec3  vBTNx, vBTNy, vBTNz;

void
main()
{
    vN = normalize( gl_NormalMatrix * gl_Normal );           // normal vector
    vec3 Tg = vec3( 0., 1., 0.);           // guess
    vec3 B = normalize( cross(Tg,vN) );
    vec3 T = normalize( cross(vN,B) );

    // produce the transformation from Surface coords to Eye coords
    vBTNx = vec3( B.x, T.x, vN.x );
    vBTNy = vec3( B.y, T.y, vN.y );
    vBTNz = vec3( B.z, T.z, vN.z );
    vST = gl_MultiTexCoord0.st;

    vec4 ECposition = gl_ModelViewMatrix * gl_Vertex;           // eye coordinate position
    vL = uLightPosition - ECposition.xyz;           // vector from the point to the light position
    vE = vec3( 0., 0., 0. ) - ECposition.xyz;           // vector from the point to the eye position
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```



mjb - December 24, 2023

Generalized Bump Mapping: Using the s-t-h to X-Y-Z Transform

11

Fragment shader:

```

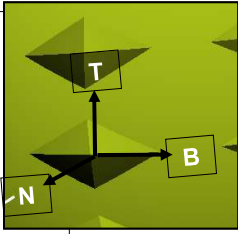
#version 330 compatibility
uniform vec3 uColor;
uniform vec3 uSpecularColor;
uniform float uKa, uKd, uKs; // coefficients of each type of lighting
uniform float uShininess; // specular exponent
uniform float uBumpDensity; // density of bumps

in vec2 vST; // texture cords
in vec3 vN; // normal vector
in vec3 vL; // vector from point to light
in vec3 vE; // vector from point to eye
in vec3 vBTNx, vBTNy, vBTNz;

vec3
ToXYZ( vec3 btn )
{
    btn = normalize( btn );

    vec3 xyz;
    xyz.x = dot( vBTNx, btn );
    xyz.y = dot( vBTNy, btn );
    xyz.z = dot( vBTNz, btn );

    return normalize( xyz );
}
    
```



$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} B_x & T_x & N_x \\ B_y & T_y & N_y \\ B_z & T_z & N_z \end{bmatrix} \begin{pmatrix} b \\ t \\ n \end{pmatrix}$$

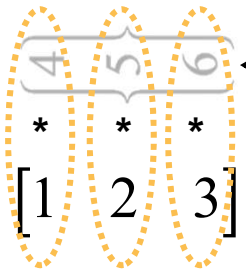
Look at this closely. It is actually a matrix-multiply!

University Computer Graphics mjb - December 24, 2023

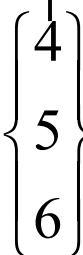
Matrix Multiplication is Really Row-by-Row Dot Products

12

The basic operation of matrix multiplication is to pair-wise multiply a single row by a single column



A
1x3




B
3x1

→

$4*1 + 5*2 + 6*3 \rightarrow$

32

C
1x1



Oregon State
University
Computer Graphics

mjb - December 24, 2023

Generalized Bump Mapping: Using the Surface Local Transform, I

13

```

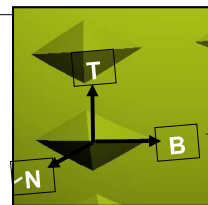
...
void
main( )
{
    vec3 Normal = normalize(vN);
    vec3 Light  = normalize(vL);
    vec3 Eye   = normalize(vE);
    vec3 myColor = uColor;           // default color

    // locate the bumps based on (s,t):
    float Swidth = (1.-0.) / uBumpDensity; // s distance between bumps
    float Theight = (1.-0.) / uBumpDensity; // t distance between bumps
    float numInS = int( vST.s / Swidth ); // which "checker" square we are in
    float numInT = int( vST.t / Theight ); // which "checker" square we are in

    vec2 center;
    center.s = numInS * Swidth + Swidth/2.; // center of that bump checker
    center.t = numInT * Theight + Theight/2.; // center of that bump checker
    vec2 st = vST - center; // st is now wrt the center of the bump

    float theta = atan( st.t, st.s );
    ...

```



Generalized Bump Mapping: Using the Surface Local Transform, II

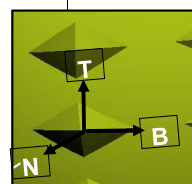
14

```

...
vec3 normal = ToXyz( Normal ); // 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. ) );
    }
}
...

```



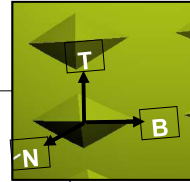
Generalized Bump Mapping: Using the Surface Local Transform, III

15

```

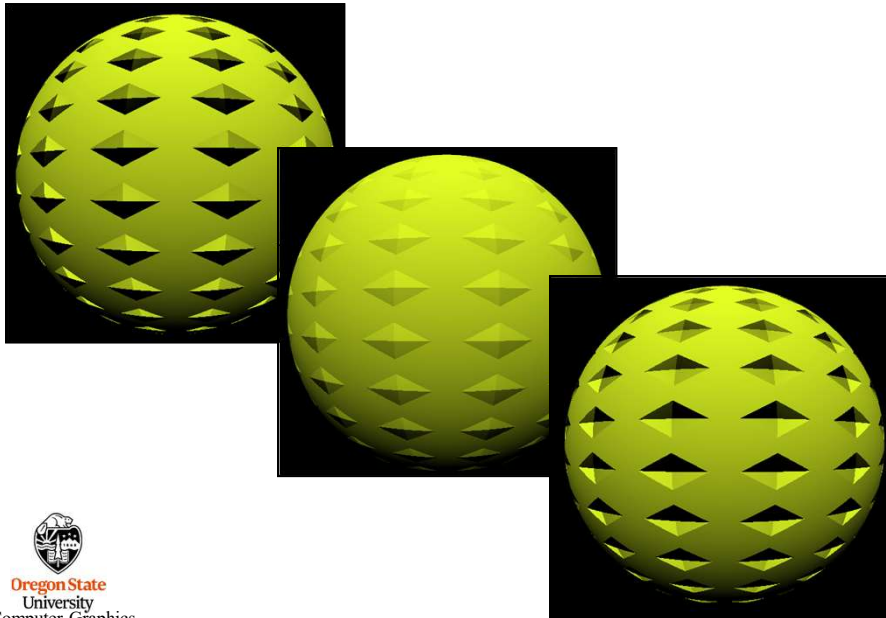
...
vec3 ambient = uKa * myColor;
float d = 0.;
float s = 0.
if( dot(normal,Light) > 0. // only do specular if the light can see the point
{
    d = dot(normal,Light);
    vec3 R = normalize( reflect( -Light, normal ) ); // reflection vector
    s = pow( max( dot(Eye,R), 0. ), uShininess );
}
vec3 diffuse = uKd * d * myColor;
vec3 specular = uKs * s * uSpecularColor;
gl_FragColor = vec4( ambient + diffuse + specular, 1. );
}

```



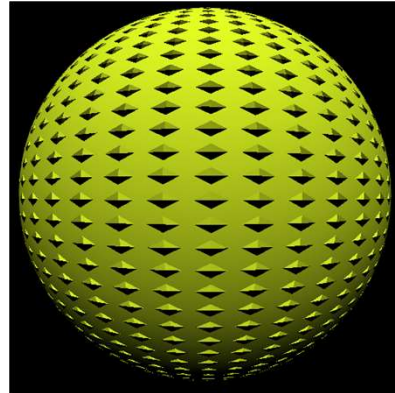
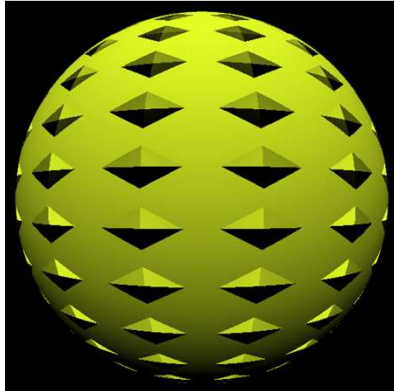
Changing the Bump Height

16



Changing the Bump Density

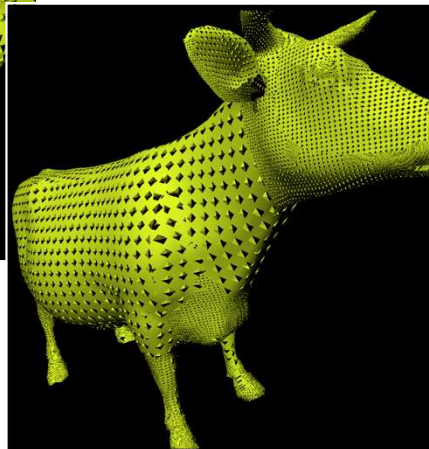
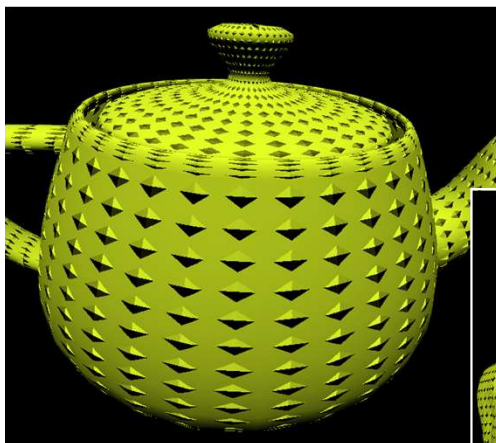
17



mjb - December 24, 2023

Different Objects

18



Cow Pox? :-)

mjb - December 24, 2023

Combining Bump and Cube Mapping:
A Good Reason to Work in X-Y-Z instead of B-T-N

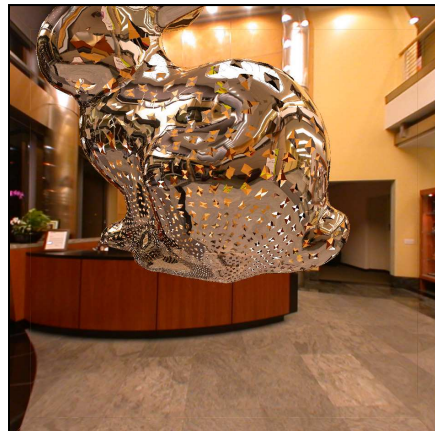
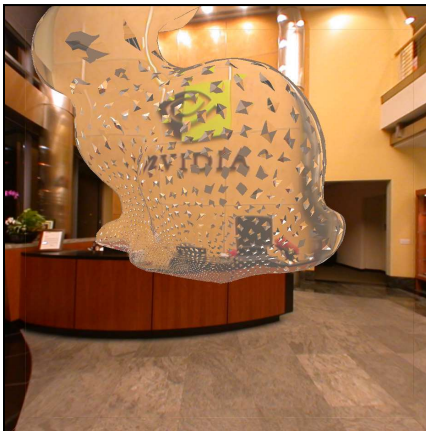
19



mjb - December 24, 2023

Combining Bump and Cube Mapping:
A Good Reason to Work in X-Y-Z instead of B-T-N

20



mjb - December 24, 2023