Surface Local Coordinates for Advanced Bump-Mapping

The Most Generalized Type of Bump Mapping, but the Trickiest, is Surface Local Coordinate Systems

- N is the surface Normal and is always perpendicular to the surface
- T is the Tangent, which is tangent to the surface and must be consistently re-oriented from vertex to vertex
- B is the Bi-Tangent, which is perpendicular to T, but also tangent to the surface and must be consistently re-oriented from vertex to vertex

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License
Bump Mapping: A Problem

The problem is that lighting information is in Eye Coordinates, but the bump information is in Surface Local Coordinates!

We need to figure out how to convert from Surface Local Coordinates to Eye Coordinates so we can light the bumps.

While we are at it, let’s also rename the Surface Local coordinates to (s,t,h) for (texture_s, texture_t, bump_height). This is the same as (B,T,N), but uses terminology that is more bump-specific.

Bump Mapping: Converting Between Coordinate Systems

Converting from Surface Local Coordinates to Eye Coordinates:

\[
\begin{bmatrix}
 x \\
 y \\
 z
\end{bmatrix} =
\begin{bmatrix}
 B_x & T_x & N_x \\
 B_y & T_y & N_y \\
 B_z & T_z & N_z
\end{bmatrix}
\begin{bmatrix}
 s \\
 t \\
 h
\end{bmatrix}
\]

We will now figure out how to derive the B’s, T’s, and N’s.
Bump Mapping: Establishing the Surface Local Coordinate System

Assuming that we have the Normal vector from the model, then:

Pick a general rule, e.g., "Tangent ≈ up", then either:

a. Use two cross-products to correctly orthogonalize it wrt the Normal, or,

b. Use Gram-Schmidt to correctly orthogonalize it wrt the Normal

Note: a and b give the same result, but some people find b easier to understand

Cross Product Orthogonalization

\[ T = \text{vec3}(0.,1.,0.); \]
1. \[ B = \text{normalize}(\text{cross}(T,N)); \]
2. \[ T' = \text{normalize}(\text{cross}(N,B)); \]
Gram-Schmidt Orthogonalization

Given that \( \mathbf{N} \) is correct, how do we change \( \mathbf{T} \) to be exactly perpendicular to \( \mathbf{N} \)?

1. How much of \( \mathbf{T} \) is in the \( \mathbf{N} \) direction

2a. How much of \( \mathbf{T} \) to get rid of so that none of it is in the same direction as \( \mathbf{N} \)

2b. The resulting \( \mathbf{T}' \) is exactly perpendicular to \( \mathbf{N} \)

3. Use the cross product to get \( \mathbf{B} \)

Bump Mapping:
Establishing the Surface Local Coordinate System

// Produce the transformation from Surface coords to Eye coords:

\[
\mathbf{BTNx} = \begin{pmatrix} B.x & T.x & N.x \end{pmatrix}; \\
\mathbf{BTNy} = \begin{pmatrix} B.y & T.y & N.y \end{pmatrix}; \\
\mathbf{BTNz} = \begin{pmatrix} B.z & T.z & N.z \end{pmatrix};
\]

// where the light is coming from:

\[
\mathbf{vec3 \ LightPosition} = \begin{pmatrix} \text{LightX} & \text{LightY} & \text{LightZ} \end{pmatrix}; \\
\mathbf{vec3 \ ECposition} = (\begin{pmatrix} \text{gl_ModelViewMatrix} & \text{gl_Vertex} \end{pmatrix} \cdot \text{xyz}); \\
\mathbf{DirToLight} = \text{normalize}(\begin{pmatrix} \text{LightPosition} - \text{ECposition} \end{pmatrix}); \\
\mathbf{gl\_Position} = \begin{pmatrix} \text{gl_ModelViewProjectionMatrix} & \text{gl_Vertex} \end{pmatrix};
\]
**Bump Mapping:**
Using the Surface-Local-to-Eye-Coordinate Transform

```cpp
vec3 ToXyz( vec3 sth )
{
    sth = normalize( sth );
    vec3 xyz;
    xyz.x = dot( BTNx, sth );
    xyz.y = dot( BTNy, sth );
    xyz.z = dot( BTNz, sth );
    return normalize( xyz );
}
```

\[
\begin{bmatrix}
    x \\
    y \\
    z
\end{bmatrix} =
\begin{bmatrix}
    B_x & T_x & N_x \\
    B_y & T_y & N_y \\
    B_z & T_z & N_z
\end{bmatrix}
\begin{bmatrix}
    s \\
    t \\
    h
\end{bmatrix}
\]

---

**Bump Mapping:**
Using the Surface Local Transform

```cpp
void main()
{
    const float PI = 3.14159265;
    vec2 st = vST; // locate the bumps based on (s,t)

    float swidth = 1. / uBumpDensity;
    float theight = 1. / uBumpDensity;
    float numInS = floor( st.s / swidth );
    float numInT = floor( st.t / theight );
    vec2 center;
    center.s = numInS * swidth + swidth/2.;
    center.t = numInT * theight + theight/2.;
    vec3 stp = st - center; // st' is now wrt the center of the bump

    float theta = atan( stp.t, stp.s );
    // ... more code...
}
```
Bump Mapping:
Using the Surface Local Transform

```
vec3 normal = ToXyz( vec3( 0., 0., 1. ) ); // un-pyramided 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., uHeight, theight/4. )  );
    }
    else if( -PI/4. <= theta  &&  theta <= PI/4. )
    {
        normal = ToXyz(  vec3( 0., 0., swidth/4. )  );
    }
    else if( -3.*PI/4. <= theta  &&  theta <= -PI/4. )
    {
        normal = ToXyz(  vec3( 0., -uHeight, theight/4. )  );
    }
    else if( theta >= 3.*PI/4.  ||  theta <= -3.*PI/4. )
    {
        normal = ToXyz(  vec3( -uHeight, 0., swidth/4. )  );
    }
}

float intensity = max( 0., dot(normal, vDirToLight) );
vec3 litColor = uSurfaceColor.rgb * intensity;
gl_FragColor = vec4( litColor, 1. );
```
Changing the Bump Density

Other Objects

Cow Pox? :-}