BUMP MAPPING

Programação 3D Simulação e Jogos
Prof. João A. Madeiras Pereira
MEIC-A/IST
Examples
## Shading

<table>
<thead>
<tr>
<th>Flat shading</th>
<th>Gouraud shading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only the first normal of the triangle is used to compute lighting in the entire triangle.</td>
<td>The light intensity is computed at each vertex and interpolated across the surface.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phong shading</th>
<th>Bump mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normals are interpolated across the surface, and the light is computed at each fragment.</td>
<td>Normals are stored in a bumpmap texture, and used instead of Phong normals.</td>
</tr>
</tbody>
</table>
Generating Normal Map

Base texture (RGB)

Height map (Grey scale)

Normal map (normal encoded RGB)
Displacement Mapping

- **Bump mapping**
  - can be at pixel level
  - has no geometry/shape change

- **Displacement Mapping**
  - Actually modify the surface geometry (vertices)
  - re-calculate the normals
  - Can include bump mapping
Displacement Mapping

- Bump mapped normals are **inconsistent** with actual geometry. No shadow.
- Displacement mapping affects the surface geometry.
Bump Mapping

Base texture (RGB)

Normal map (normal encoded RGB)
Normal Map

- Normal vector encoded as rgb
  - $[-1,1]^3 \rightarrow [0,1]^3$: rgb = $n \times 0.5 + 0.5$
- RGB decoding in fragment shaders
  - vec3 $n = \text{texture2D}(\text{NormalMap}, \text{texcoord.st}).rgb \times 2.0 - 1.0$
- In tangent space, the default (unit) normal points in the +z direction.
  - Hence the RGB color for the straight up normal is (0.5, 0.5, 1.0). This is why normal maps are a blueish color
- Normals are then used for shading computation
  - Diffuse: $n \cdot l$
  - Specular: $(n \cdot h)^{\text{shininess}}$
  - Computations done in tangent space
• The big problem is how to convert our normal, which is expressed **in the space of each individual triangle** (tangent space), in model space (since this is what is used in our shading equation).

• we need 3 vectors. We already have our UP vector: it’s the normal, computed from the triangle by a simple cross product. It’s represented in blue, just like the overall color of the normal map.
Next we need a tangent, $T$: a vector perpendicular to the surface. But there are many such vectors:
Which one should we choose? In theory, any, but we have to be consistent with the neighbors to avoid introducing ugly edges. The standard method is to orient the plane defined by the tangent (red color) and the other orthogonal vector (called bitangent in green color) in the same direction that our 2D texture space referential:
Tangent Space concept
In order to build this Tangent Space, which is a 3D space, we need to define an orthonormal basis.

Tangent space is composed of 3 orthogonal vectors (T, B, N).

- **Tangent (T Tangent)**
- **Bitangent (B Tangent) – wrongly also called Binormal**
- **Normal (N) - perpendicular to (T,B) plane**

To align the tangent space (3D) with texture space (2D), means:

- (T, B) vectors correspond to (u, v) vectors. T is the vector (1, 0) and B is the vector (0, 1).
- Note that a 2D point in texture space (ui, vi) corresponds to a 3D point (ui, vi, 0) in Tangent space.
TBN at each Vertex

- Lighting calculations: light vector in World/Eye space; normal vector at Tangent space -> spaces transformation matrix;
- First, calculate the TBN vectors in World coordinates for each triangle of the mesh: the Triangle-based TBN
- Then, calculate a tangent space matrix for every single vertex by averaging the triangle-based TBNs which share that vertex: Vertex-based TBN.
Vertex-based TBN (per vertex)

The values between brackets represent: (U, V) or also (S, T)

- Normal Vector
- Tangent Vector (G Tangent)
- Bitangent Vector (T Tangent)

Texcoord (0, 0)
Texcoord (1, 0)
Texcoord (0, 1)
Triangle-based Tangent Space

- Suppose a point $p_i$ in **world coordinate system** for whose texture coordinates are $(u_i, v_i)$
- Writing this equation for the points $p_1$, $p_2$ and $p_3$, defining the triangle:

  $p_1 = u_1.T + v_1.B$
  $p_2 = u_2.T + v_2.B$
  $p_3 = u_3.T + v_3.B$
Triangle-based Tangent Space

- \( \mathbf{p}_2 - \mathbf{p}_1 = (\mathbf{u}_2 - \mathbf{u}_1) \cdot \mathbf{T} + (\mathbf{v}_2 - \mathbf{v}_1) \cdot \mathbf{B} \)
- \( \mathbf{p}_3 - \mathbf{p}_1 = (\mathbf{u}_3 - \mathbf{u}_1) \cdot \mathbf{T} + (\mathbf{v}_3 - \mathbf{v}_1) \cdot \mathbf{B} \)  
  6 eqns, 6 unknowns

- \( (\mathbf{v}_3 - \mathbf{v}_1) \cdot (\mathbf{p}_2 - \mathbf{p}_1) = (\mathbf{v}_3 - \mathbf{v}_1) \cdot (\mathbf{u}_2 - \mathbf{u}_1) \cdot \mathbf{T} + (\mathbf{v}_3 - \mathbf{v}_1) \cdot (\mathbf{v}_2 - \mathbf{v}_1) \cdot \mathbf{B} \)
- \( - (\mathbf{v}_2 - \mathbf{v}_1) \cdot (\mathbf{p}_3 - \mathbf{p}_1) - (\mathbf{v}_2 - \mathbf{v}_1) \cdot (\mathbf{u}_3 - \mathbf{u}_1) \cdot \mathbf{T} - (\mathbf{v}_2 - \mathbf{v}_1) \cdot (\mathbf{v}_3 - \mathbf{v}_1) \cdot \mathbf{B} \)

- \( (\mathbf{u}_3 - \mathbf{u}_1) \cdot (\mathbf{p}_2 - \mathbf{p}_1) = (\mathbf{u}_3 - \mathbf{u}_1) \cdot (\mathbf{u}_2 - \mathbf{u}_1) \cdot \mathbf{T} + (\mathbf{u}_3 - \mathbf{u}_1) \cdot (\mathbf{v}_2 - \mathbf{v}_1) \cdot \mathbf{B} \)
- \( - (\mathbf{u}_2 - \mathbf{u}_1) \cdot (\mathbf{p}_3 - \mathbf{p}_1) - (\mathbf{u}_2 - \mathbf{u}_1) \cdot (\mathbf{u}_3 - \mathbf{u}_1) \cdot \mathbf{T} - (\mathbf{u}_2 - \mathbf{u}_1) \cdot (\mathbf{v}_3 - \mathbf{v}_1) \cdot \mathbf{B} \)

\[
T = \frac{(\mathbf{v}_3 - \mathbf{v}_1) \cdot (\mathbf{p}_2 - \mathbf{p}_1) - (\mathbf{v}_2 - \mathbf{v}_1) \cdot (\mathbf{p}_3 - \mathbf{p}_1)}{(\mathbf{u}_2 - \mathbf{u}_1) \cdot (\mathbf{v}_3 - \mathbf{v}_1) - (\mathbf{v}_2 - \mathbf{v}_1) \cdot (\mathbf{u}_3 - \mathbf{u}_1)}
\]

\[
B = \frac{(\mathbf{u}_3 - \mathbf{u}_1) \cdot (\mathbf{p}_2 - \mathbf{p}_1) - (\mathbf{u}_2 - \mathbf{u}_1) \cdot (\mathbf{p}_3 - \mathbf{p}_1)}{(\mathbf{v}_2 - \mathbf{v}_1) \cdot (\mathbf{u}_3 - \mathbf{u}_1) - (\mathbf{u}_2 - \mathbf{u}_1) \cdot (\mathbf{v}_3 - \mathbf{v}_1)}
\]

\( \mathbf{N} = \text{cross}(\mathbf{T}, \mathbf{B}) \)  // no need to do it since \( \mathbf{N} \) is the triangle normal
Use the averaged triangle normal as the vertex normal
Do the same for tangent and bitangent vectors
But, as we are going to see, it’s not necessary to store an extra array containing the per-vertex bitangent; why?
Spaces Transformation

- Forget the Translation. Only the rotation is important. Why?

Tangent space to World space:
\[
\begin{bmatrix}
  w v_x \\
  w v_y \\
  w v_z 
\end{bmatrix} = \begin{bmatrix}
  T_x & B_x & N_x \\
  T_y & B_y & N_y \\
  T_z & B_z & N_z 
\end{bmatrix} \begin{bmatrix}
  T v_x \\
  T v_y \\
  T v_z 
\end{bmatrix}
\]

World space to Tangent space:
\[
\begin{bmatrix}
  T v_x \\
  T v_y \\
  T v_z 
\end{bmatrix} = \begin{bmatrix}
  T_x & B_x & N_x \\
  T_y & B_y & N_y \\
  T_z & B_z & N_z 
\end{bmatrix}^{-1} \begin{bmatrix}
  w v_x \\
  w v_y \\
  w v_z 
\end{bmatrix} = \begin{bmatrix}
  T_x & T_y & T_z \\
  B_x & B_y & B_z \\
  N_x & N_y & N_z 
\end{bmatrix} \begin{bmatrix}
  w v_x \\
  w v_y \\
  w v_z 
\end{bmatrix}
\]

Two sources of non-orthogonality in World space:
1) Triangle-based TBN
2) Vertex-based TBN

Only if TBN is also orthogonal in World space.
Non-orthogonality in World space

- Triangle-based TBN
  - The transformation from the texture space into world space may not be distance/angle conservative.
  - So, generally, (T, B, N) is not orthonormal in world space: T and B are not necessarily perpendicular, but they are perpendicular with N. As well, possibly not unit vectors.

- Vertex-based TBN
  - Use the averaged triangle normal as the vertex normal
  - Do the same for tangent and bitangent vectors
  - T, B vectors might not be orthogonal to the normal vector

- Use Gram-Schmidt to make sure they are orthogonal
Orthogonal Vertex-based T’B’N

- N is the **normal vertex**
- By using the Gram-Schmidt technique:
  - \( T' = T - (N \cdot T)N \)
  - \( B' = B - (N \cdot B)N - (T' \cdot B)T'/T'^2 \)
- After normalizing, the matrix to convert from World to Tangent is simply the **transposed** matrix:

\[
\begin{bmatrix}
T'_x & T'_y & T'_z \\
B'_x & B'_y & B'_z \\
N_x & N_y & N_z
\end{bmatrix}
\]

- Send both \( T' \) and \( B' \) to the Vertex Shader, or
- not necessary to store an extra array containing the per-vertex bitangent since the cross product \( N \times T' \) can be used to obtain \( mB' \)
- \( m = \pm 1 \) represents the handedness of the tangent space.
- Implementation: \( T' \) as a four-dimensional entity whose \( w \) coordinate holds the value of \( m \); so
  \( B' = T'_w(N \times T') \)

- Code for \( T'_w \), where \( t, b, n \) are vec3D and \( t' \) is vec4D

\[
t' = \text{normalize} \left[ (t - n \times (\text{Dot}(n, t))) \right]
\]
\[
t'.w = (\text{Dot}(\text{Cross}(n, t'), b) < 0.0f? -1.0f : 1.0f);
\]
GLSL Matrix operations

GLSL makes no distinction between column and row vectors. If a vector multiplied from the left to the matrix is row vector; if multiplied from the right is column vector

\[
\begin{bmatrix}
T_{L_x}^T \\
T_{L_y}^T \\
T_{L_z}^T
\end{bmatrix} =
\begin{bmatrix}
T_x' & B_x' & N_x \\
T_y' & B_y' & N_y \\
T_z' & B_z' & N_z
\end{bmatrix}^{-1}\begin{bmatrix}
L_x \\
L_y \\
L_z
\end{bmatrix} =\begin{bmatrix}
T_x' & T_y' & T_z' \\
B_x' & B_y' & B_z'
\end{bmatrix}\begin{bmatrix}
N_x \\
N_y \\
N_z
\end{bmatrix}\begin{bmatrix}
L_x \\
L_y \\
L_z
\end{bmatrix}
\]

matrix - vector product

\[
\begin{bmatrix}
T_{L_x}^T & T_{L_y}^T & T_{L_z}^T
\end{bmatrix} =\begin{bmatrix}
L_x \\
L_y \\
L_z
\end{bmatrix}\begin{bmatrix}
T_x' & B_x' & N_x \\
T_y' & B_y' & N_y \\
T_z' & B_z' & N_z
\end{bmatrix}
\]

vector - matrix product

\[
^T L_x = \text{dot}(L, T');
\]

\[
^T L_y = \text{dot}(L, B');
\]

\[
^T L_z = \text{dot}(L, N);
\]

LightDir = normalize \((^T L)\)
GLSL Vertex Shader

in vec4 vPos, vNormal, vTexCoord, vTangent;

out vec3 lightVec, halfVec;
out vec2 tex_coord;

uniform mat4 ModelViewMatrix, pvmMatrix;
uniform light LightSource;
uniform mat3 NormalMatrix;

void main()

} }
GLSL Fragment Shader

uniform sampler2D baseTexture;
uniform sampler2D normalTexture;
uniform light LightSource;
uniform material Material;
in vec3 lightVec, halfVec;
in vec2 tex_coord;

void main()
{
    // lookup normal from normal map, move from [0,1] to [-1, 1] range, normalize
    vec3 normal = 2.0 * texture(normalTexture, tex_coord).rgb - 1.0;
    normal = normalize(normal);

    // compute diffuse lighting
    float lambertFactor = max(dot(lightVec, normal), 0.0);
    vec4 diffuseMaterial = vec4(0.0);
    vec4 diffuseLight = vec4(0.0);

    vec4 ambientLight = LightSource.ambient;

    if (lambertFactor > 0.0)
    {
        diffuseMaterial = texture(baseTexture, tex_coord) * Material.diffuse;
        shininess = pow(max(dot(halfVec, normal), 0.0), Material.shininess);

        gl_FragColor = diffuseMaterial * LightSource.diffuse * lambertFactor;
        gl_FragColor += Material.specular * LightSource.specular * shininess;
    }
    gl_FragColor += ambientLight;
}
References

- http://jerome.jouvie.free.fr/opengl-tutorials/Lesson8.php
- http://www.ozone3d.net/tutorials/bump_mapping.php
original mesh
4M triangles

simplified mesh
500 triangles

simplified mesh and normal mapping
500 triangles