Bump Mapping

CSE 781 Winter 2010

Han-Wei Shen
Bump Mapping

- Many textures are the result of small perturbations in the surface geometry
- Modeling these perturbations would result in an explosion in the number of geometric primitives
- Bump mapping is a cheap way to alter the lighting across a polygon to provide the illusion of surface displacement
- The algorithm is applied at the fragment level
Bump Mapping

- Example
Bump Mapping
Basic Idea

- Consider the lighting for a bumpy surface.
Basic Idea

- We can model a bumpy surface as deviations from a base surface.
- The question is then how these deviations change the lighting.
- Remember we do not want to create the actual geometry.
Surface Displacement

- Bumps: small deviations along the normal direction from the surface.

\[ X' = X + B \hat{N} \]

Where B is the amount of surface displacement, defined as a 2D function parameterized over the surface:
Surface Parameterization

- $B(u,v)$ is the displacement in a 2D parameterized surface at the point $(u,v)$ along the normal direction.

- Surface parameterization: $(x,y,z) \leftrightarrow (u,v)$
  
  - We can get the surface point $(x,y,z)$ from the parameterization: $x(u,v)$, $y(u,v)$, $z(u,v)$, or $O(u,v)$ ($O = (x,y,z)$).
  
  - The transformation between $(x,y,z)$ and $(u,v)$ is given in parametric surfaces, but can be also derived for other analytical surfaces.

- $(u,v)$ can be seen as the texture coordinates.

- Assuming such a parameterization already exists, let’s see how to calculate the new normal given $B(u,v)$. 
The Original Normal

- Define the tangent plane to the surface at a point \((u,v)\) by using the two vectors \(O_u\) and \(O_v\), resulting from the partial derivatives.
- Analytic derivatives or, you can compute them using central difference:
  - \(O_u = (O(u+1,v) - O(u-1,v)) / 2\)
  - \(O_u = (O(u,v+1) - O(u,v-1)) / 2\)
- The normal at \(O\) is then given by:
  - \(N = O_u \times O_v\)
The New Normal

- The new surface positions are given by:
  - $O'(u,v) = O(u,v) + B(u,v)\ N$
  - Where, $N = N / |N|$

- We need to get the new normal after the displacement. Differentiating of $O'(u,v)$ leads to:
  - $O'_u = O_u + B_u\ N + B\ N_u \approx O'_u = O_u + B_u\ N$
  - $O'_v = O_v + B_v\ N + B\ N_v \approx O'_v = O_v + B_v\ N$

If $B$ is small.
Bump Mapping

- This leads to a new normal:
  \( N'(u,v) = O_u \times O_v + B_u(N \times O_v) - B_v(N \times O_u) \]
  \[ + B_u B_v(N \times N) \]

\[ = N + B_u(N \times O_v) - B_v(N \times O_u) \]
\[ = N + D \]
Bump Mapping Representation

- For efficiency, we can store \( B_u \) and \( B_v \) in a 2-component texture map.
  - This is commonly called a *offset vector* map.
  - Note: \( B_u \) and \( B_v \) are oriented in tangent-space.
- \( B_u \) and \( B_v \) are used to modify the normal \( N \).
- Another way is to represent the bump as a high field.
  - The high field can be used to derive \( B_u \) and \( B_v \) (using central difference).
- See Figure in the next page.
Bump Map Representation
Bump Mapping Representation

- An alternative representation of bump maps can be viewed as a rotation of the normal.
- The rotation axis is the cross-product of \( N \) and \( N' \).

\[
\vec{A} = \vec{N} \times \vec{N}' = \vec{N} \times (\vec{N} + \vec{D}) = \vec{N} \times \vec{D}
\]
Bump Mapping Representation

- In sum, we can store:
  - The height displacement
  - The offset vectors in tangent space
  - The rotations in tangent space
    - Matrices
    - Quaternions
    - Euler angles
Bump Mapping in Graphics Hardware

- The primary method for modern graphics hardware (also called dot product bump mapping)
- The bump map texture stores the actual normals to be used for bump mapping for the surface (defined in the fragment’s tangent space)
- Lighting for every fragment is calculated by evaluating $N \cdot L$ (for diffuse) and $(N \cdot H)^8$ (for specular)
Dot3 Bump Mapping

- The lighting is calculated in the tangent space – we need to transform the light vector of very vertex to the vertex’s tangent space
- The light vector for each fragment is obtained by interpolation using graphics hardware (pass the light vector as the vertex color)

\[ L' = \text{lerp} \ (L \text{ at } V1, \ L \text{ at } V2, \ L \text{ at } V3) \]

** Remember \( L' \) has to be normalized for lighting calculation – how?
Light Vector in Tangent Space

- Transform the light vector to the tangent plane of the surface point – remember different surface point has different tangent plane.
- To do so, we first construct a tangent space.

![Diagram of a point P with vectors t, n, and b]

- $t$ is the tangent vector follows the texture parameter $u$ direction.
- $n$ is the surface normal.
- $b$ is another tangent vector perpendicular to $t$ and $n$ ($b = n \times t$).

Then $t$, $n$, and $b$ form a coordinate frame for the point $P$ – called tangent space.
Shifting \((u,v)\) in the light direction? (II)

- After we know \(t, b, n\) (the basis vector of the tangent space), we can transform the light vector to the tangent space using the transformation matrix \( \mathbf{M} \):

\[
\begin{pmatrix}
(lx', ly', lz', 0) = \mathbf{M} \times (lx, ly, lz, 0)
\end{pmatrix}
\]

\[
\mathbf{M} = \begin{bmatrix}
\mathbf{t}_x & \mathbf{t}_y & \mathbf{t}_z & 0 \\
\mathbf{b}_x & \mathbf{b}_y & \mathbf{b}_z & 0 \\
\mathbf{n}_x & \mathbf{n}_y & \mathbf{n}_z & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Emboss Bump Mapping

- A cheap bump mapping implementation (but not really bump mapping)
- Image embossing – shift to the light and then subtract
Emboss Bump Mapping

- The same idea can be applied to 3D surfaces
- Basic idea: Apply the input texture to the surface twice
  - First pass – use the input texture as before
  - Second pass – shift the texture coordinates of each vertex in the direction of light, render it, and then subtracting it from the result of the first pass
  - Then you can render the surface using regular lighting and add it to the result
Shifting \((u,v)\) in the light direction?

- Project the light to the tangent plane of the surface point – remember different surface point has different tangent plane
- To do so, we first construct a tangent space

- \(t\) is the tangent vector follows the texture parameter \(u\) direction
- \(n\) is the surface normal
- \(b\) is another tangent vector perpendicular to \(t\) and \(n\) \((b = n \times t)\)

Then \(t, n, b\) form a coordinate frame for the point \(P\) – called tangent space
After we know \( t, b, n \) (the basis axes of the tangent space), we can transform light to the tangent space using the transformation matrix \( M \): 

\[
M : (l_x', l_y', l_z', 0) = M \ast (l_x, l_y, l_z, 0)
\]

\[
M = \begin{bmatrix}
    tx & ty & tz & 0 \\
    bx & by & bz & 0 \\
    nx & ny & nz & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]

Normalize \( (l_x', l_y') \) and then multiple both by \( 1/r \) (texture resolution). Add these two numbers to the texture coordinate \((u, v)\) – done.