Advanced Texture and Lighting

• Bump Mapping
• Displacement Mapping
• Environment Mapping
• Light Mapping
• Shadow Mapping
Bump Mapping

- Specify vertex/pixel normals of polygon

You have to calculate each vertex or pixel normal according to the specification.
Normal Map

- Use texel values to modify vertex/pixel normals of polygon
- Texel values correspond to normals (or heights) modifying the current normals
- \( \text{RGB} = \frac{(n+1)}{2} \)
- \( n = 2 \times \text{RGB} - 1 \)
Bump Mapping

• The light source direction \( L \) and pixel normal \( N \) are represented in the global coord \( x, y, z \)

• the bump map normal \( n \) is in its local coordinates, which is called tangent space or texture space

- We create a local coordinates \( TNB \)
  - \( T, B \) are in the texture
  - \( N \) is perpendicular to the texture
  - \( T, N, B \) are three perpendicular vectors in \( x,y,z \) coordinates

- The bump map normal \( n \) is represented in the \( TNB \) coordinates.

- Therefore, each pixel has a \( TNB \)
Bump Mapping (cont.)

• The tangent space can be aligned with any orientation as long as they are perpendicular to the normal $\mathbf{N}$, and they do not rotate around $\mathbf{N}$ from point to point.

• The tangent space coordinates TNB: $T = \mathbf{N} \times \mathbf{S}$; $B = \mathbf{T} \times \mathbf{N}$

• $L' = (T \cdot \mathbf{L}, N \cdot \mathbf{L}, B \cdot \mathbf{L})$

• $T, N, B$ as vectors should be the same across the polygon. So they are calculated at vertex level.
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
Environment Mapping
Environment Mapping

- Used to show the reflected colors in shiny objects
- A ray $A$ and a normalized normal $n$, the reflected ray $B = 2n(n \cdot A) - A$. 

![Diagram of Environment Mapping](image)
Environment Mapping

- a simplified method uses the **surface normal** as an index for the texel on the cube surface
Cube Environment Mapping
Ent mapping with GL_REFLECTION_MAP & GL_TEXTURE_CUBE_MAP

```c
// For texture mapping
gl.glBindTexture(GL.GL_TEXTURE_CUBE_MAP_POSITIVE_X, cubeMap[0]);
gl.glBindTexture(GL.GL_TEXTURE_CUBE_MAP_NEGATIVE_X, cubeMap[1]);
gl.glBindTexture(GL.GL_TEXTURE_CUBE_MAP_POSITIVE_Y, cubeMap[2]);
gl.glBindTexture(GL.GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, cubeMap[3]);
gl.glBindTexture(GL.GL_TEXTURE_CUBE_MAP_POSITIVE_Z, cubeMap[4]);
gl.glBindTexture(GL.GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, cubeMap[5]);
gl.glEnable(GL.GL_TEXTURE_CUBE_MAP);

// GL_REFLECTION_MAP for s, t, r texture coordinates
gl.glTexGeni(GL.GL_S, GL.GL_TEXTURE_GEN_MODE, GL.GL_REFLECTION_MAP);
gl.glTexGeni(GL.GL_T, GL.GL_TEXTURE_GEN_MODE, GL.GL_REFLECTION_MAP);
gl.glTexGeni(GL.GL_R, GL.GL_TEXTURE_GEN_MODE, GL.GL_REFLECTION_MAP);

gl.glEnable(GL.GL_TEXTURE_GEN_S);
gl.glEnable(GL.GL_TEXTURE_GEN_T);
gl.glEnable(GL.GL_TEXTURE_GEN_R);

// Draw shape to apply reflection on it
...
```
Spherical Environ. Mapping

- More Efficient than cube maps
- Image of perfect sphere reflector seen from infinity (orthographic)
Indexing Sphere Maps

- Implemented in hardware
  - //Texture use for automatic coordinate generation
    ```
    gl.glBindTexture(GL.GL_TEXTURE_2D, texture);
    gl.glEnable(GL.GL_TEXTURE_2D);
    
    //Sphere mapping and enable s & t texture generation
    gl.glTexGeni(GL.GL_S, GL.GL_TEXTURE_GEN_MODE,
                 GL.GL_SPHERE_MAP);
    gl.glTexGeni(GL.GL_T, GL.GL_TEXTURE_GEN_MODE,
                 GL.GL_SPHERE_MAP);
    gl.glEnable(GL.GL_TEXTURE_GEN_S);
    gl.glEnable(GL.GL_TEXTURE_GEN_T);
    
    //Draw the shapes to apply the texture
    ...
    ```

- Problems:
  - Highly non-uniform sampling
  - Highly non-linear mapping
Non-uniform Sampling
Non-linear Mapping

- Linear interpolation of texture coordinates picks up the wrong texture pixels
  - Use small polygons!
Example
Light Mapping

- Pre-render special lighting effects
- ‘Multi-texturing’ idea: arbitrary texel-by-texel shading calc’d from multiple texture maps

$$\text{Reflectance Texture} \times \text{Light Map (Illumination Texture)} = \text{Display texture}$$
Shadow Mapping

- Render scene from light’s point of view
  - Store depth of each pixel
Shadow Mapping

• Render scene from light’s point of view
  – Store depth of each pixel
  – From light’s point of view, any pixel blocked is in the shadow.

• When shading a surface:
  – Transform surface pixel into light coordinates
  – Compare current surface depth to stored depth. If depth > stored depth, the pixel is in shadow; otherwise the pixel is lit
  – Very expensive
Aliasing

- Williams, Siggraph ‘78
  - render scene from light source
  - shadowing by depth comparison
    - prone to aliasing when zooming into shadow boundaries

single shadow map pixel