CS 112 - Bump Mapping and Environment Mapping
Bump Mapping

- Simulate the effects of details in geometry without adding geometry
Difference from Texture Mapping

- Texture mapping cannot simulate rough surface details
- Rough surfaces show illumination changes with the movement of the light or object
  - Textured objects cannot simulate that
  - Since independent of illumination parameters

Basic Idea
- Perturb the normal and use the perturbed normal for illumination computation
Normal Perturbation Theory

Modify the surface position by adding a small perturbation (called bump function) in the direction of the normal:

\[ P'(u,v) = P(u,v) + B(u,v) n \]

- \( P \) – Point on the surface
- \( P_u \) – Tangent at \( P \) in \( u \) direction
- \( P_v \) – Tangent at \( P \) in \( v \) direction
- \( N \) – Normal at \( P \)

\( B(u,v) \) is a scalar function

This is a vector addition
Normal Perturbation Theory

- \( P'(u,v) = P(u,v) + B(u,v) N \)
- \( P'_u = P_u + B_u N + B N_u \)
- \( P'_u \approx P_u + B_u N \)
- \( P'_v \approx P_v + B_v N \)
- \( N' = P'_u \times P'_v \)
  
  \[
  P'_u \times P'_v = P_u \times P_v + B_v (P_u \times N) + B_u (P_v \times N) + B_u B_v (N \times N)
  \]
  
  \[
  = N + B_v (P_u \times N) + B_u (P_v \times N)
  \]
Implementation

- Start with a gray scale texture – $B(u,v)$
Implementation

- Find the image $B_{u}(u,v)$ by subtracting every pixel from its *right* neighbor
Implementation

- Find the image $B_v(u,v)$ by subtracting every pixel from its *bottom* neighbor
Implementation

\[ N' = N + B_v (P_u \times N) + B_u (P_v \times N) \]
Implementation

- \[ \mathbf{N}' = \mathbf{N} + B_v (\mathbf{P}_u \times \mathbf{N}) + B_u (\mathbf{P}_v \times \mathbf{N}) \]

\[(0, 0', 1) \quad (0, 1', 0) \quad (1', 0, 0)\]

- These vectors are orthogonal to each other

- Define a *local* coordinate system at a vertex by normal and the tangent plane at this vertex

- \[ \mathbf{N}' = (B_u, B_v, 1) - \text{Perturbed normal in the local coordinate system of each triangle} \]
Normal Map

- $N' = (B_u, B_v, 1)$ - Stored as RGB value
- Since $B$ is always 1, it is bluish in appearance
Light and View Vector

- Light and View vectors
  - Defined in the *global coordinate system*
- Convert these to the local coordinate system
  - Standard coordinate transformation
Rasterization

- Using pixel shaders
  - Interpolate light vector
  - Interpolate normal
  - Find the perturbed normal from the normal map
  - Do lighting computation in the local coordinate system
Examples
Interpolating Vectors?

- Yes, using pixel shaders you can do it
- So, you can do Phong shading using pixel shaders
- Standard openGL lighting does NOT do phong shading
Environment Map

- Simulating the effect of reflection of environment on a shiny object
Environment Map

- Generate the map of the environment
  - On a sphere, cube or paraboloid
- Use a view-dependent mapping on the geometry
Generating the Map (Cube)

- Render the scene from one point, six times with six different view frustums
- On six planes of a cube
Cubic Environment Map
Generating the Map (Sphere)

- Use a angle parameterization
- Use ray tracing to sample the angles at uniform intervals
- OpenGL provides a spherical mapping for this
Mapping the Environment

- Enclose the arbitrary geometry in the sphere or the cube
- Reflect the view vector about the normal
- The environment in the direction of $R$ is getting reflected about $N$ and reaching the viewer at $V$

What does this remind you of?

Difference: For texture mapping, the mapping dependent on geometry (vertex and normals), here it depends on the view.
How to find $R$?

- $S = V \cos \theta - V$
- $-V + R = 2S$
- $R = 2S + V$
  
  
  $= 2V \cos \theta - V$
  
  $= 2V (N \cdot V) - V$
Results
It is an approximation

- Ideally, we should get self reflections
- In this case, we would not get
- This is just an approximation
- To get accurate environment map, we have to use ray tracing