Overview

• Bump–mapping principle
• How to Bump–map?
• What the exercise is about: How to approximately Bump–map using Textures …
**Bump–mapping principle**

- Add more realism to synthetic images without adding a lot of geometry

- Image texture / Bump–map / Displacement–map / Environment–map
How to Bump–map

Definitions

• Flat object surface: \( P(u,v) \)
• Bump–map (2D height field): \( F(u,v) \)

\[ P'(u,v) = P(u,v) + \frac{F(u,v)N}{|N|} \]

• Bumpy surface:

\[ \begin{align*}
N' &= P'_u \times P'_v \\
&= N + F'_u \frac{N \times P'_v}{|N|} + F'_v \frac{P'_u \times N}{|N|}
\end{align*} \]

• Normal of bumped surface? \( N' = P'_u \times P'_v = ? \)

How to Bump–map

Perturbed normal

• Normal of bumped surface, so-called perturbed normal:

\[ N' = N + F'_u \frac{N \times P'_v}{|N|} + F'_v \frac{P'_u \times N}{|N|} \]

• Derivation can be found in “Simulation of Wrinkled Surfaces”
  James F. Blinn
  SIGGRAPH '78 Proceedings, pp. 286-292, 1978
  (Pioneering paper...)
• Partial derivatives thru forward differencing / finite differences in bump–map & surface parameterization
How to Bump–map

Implementation

• Needs Phong–shading, i.e. evaluation of the lighting equation @ every pixel position

• Doable using:
  – Software renderer
  – Today’s graphix hardware (pixel-/vertex-shaders)
  – Approximative solution, using textures & accumulation buffering …

Bump–maps with textures

History

• This is what the exercise is about

• Original idea:
  “Efficient bump mapping hardware”
  Mark Peercy et al. (sgi)

• (Mis-)use texture to store either:
  – perturbed normal map
  – bump–map itself
**Bump–maps with textures**

**Assumption**

- We need: \( N' = P'_u \times P'_v \)
- Assume a tangent surface coincident with the xy–plane @ some point on the surface \( P \)

Then: \( P'(u, v) \equiv F(u, v) \)

\[
N' = \begin{pmatrix} \frac{\partial F}{\partial u}, \frac{\partial F}{\partial v}, 1 \end{pmatrix}^T
\]

**Evaluation (1)**

- Diffuse lighting component: \( N'.L \)

\[
N'.L = \frac{\partial}{\partial u} F \cdot L_x + \frac{\partial}{\partial v} F \cdot L_y + L_z
\]

- This requires the surface to lie in the xy–plane!

- Instead of transforming surface, transform light source direction vector to local *tangent space* (local coordinate system)
**Bump–maps with textures**

**Tangent Space**

- Tangent space defined @ polygon vertices

- Basis vectors: \( T, N, B = (T \times N) \)

**Bump–maps with textures**

**Tangent Space Transformation**

- Coordinate system transform:
  \[
  x = M \cdot x_{new} \quad x_{new} = M^{-1} \cdot x
  \]

- Transformation matrix (homogeneous, orthonormal):
  \[
  M = \begin{bmatrix}
    T_x & B_x & N_x & 0 \\
    T_y & B_y & N_y & 0 \\
    T_z & B_z & N_z & 0 \\
    0 & 0 & 0 & 1
  \end{bmatrix}
  \quad
  M^{-1} = M^T = \begin{bmatrix}
    T_x & T_y & T_z & 0 \\
    B_x & B_y & B_z & 0 \\
    N_x & N_y & N_z & 0 \\
    0 & 0 & 0 & 1
  \end{bmatrix}
  \]

- Transform light vector, using transformation matrix
Bump–maps with textures
Evaluation (2)

• How do we evaluate

\[
N' \cdot L = \frac{\partial}{\partial u} F \cdot L_x + \frac{\partial}{\partial v} F \cdot L_y + L_z
\]

→ Separation: use parallel and perpendicular components of transformed light vector

Bump–maps with textures
Evaluation (3)

• Multipass method with accumulation buffer:

\[
N' \cdot L = \frac{\partial F \cdot L_y}{\partial u} + \frac{\partial F \cdot L_z}{\partial v} + L_x
\]

Directional derivatives in 1-D thru forward differencing:

\[
F'(s) \cdot \Delta s \equiv F(s) - F(s + \Delta s)
\]
\[
F'(t) \cdot \Delta t \equiv F(t) - F(t + \Delta t)
\]

Diffuse lighting term

• „Render – Shift – Subtract“ (1st part, 2 passes)

• Add diffuse lighting term (2nd part, 1 pass)
Bump–maps with textures

Render – Shift – Subtract

• Render the bump–map texture (1st pass)
• Shift the texture coordinates (s,t) (towards light!)
• Re-render, subtracting from 1st image (2nd pass)
• this corresponds to evaluating the directional derivatives using forward differencing!

Bump–maps with textures

Recipe

1. Render polygon with bump–map on it (1st pass)
2. Find vectors T,N,B @ each vertex
3. Transform light vector L into tangent space
4. Shift texture coordinates s,t @ each vertex in direction of the light (using x,y coords of transformed light vector)
5. Re-render polygon with shifted map (2nd pass)
6. Subtract 2nd image from 1st
7. Render polygon, smooth shaded, with lighting enabled, texturing disabled, add it! (3rd pass)
Bump–maps with textures
What do you have to do?

• Implement the following 2 functions:
  – shiftcoords(…)
    • Determine light vector L
    • Transform into tangent space
    • Shift texture coordinates s,t
  – redrawbump(void)
    • Render cylinder using accumulation buffering

• Read paper by Peercy ...
• http://www.opengl.org/resources/tutorials/advanced/advanced98
  /notes/node107.html ...
• Submit to deniss@inf.ethz.ch until January 28th