Today

• Bump mapping
• Shadows

Bump mapping

• Surface detail is often the result of small perturbations in the surface geometry
• Modeling detailed surfaces would lead to impractical number of triangles
• Bump mapping alters the surface normal to provide the illusion of small scale surface detail
• Normals are encoded in texture maps
• Requires per-pixel shading using a fragment program

Generating bump maps

• Usually done in a pre-process
• Input
  - Texture map that encodes small surface displacements
  - Height field
• Output
  - Texture map that encodes normals of displaced surface
  - This texture will be stored as an image, read by the application
Generating bump maps
- Displacement map
- Normal
  \[ n(u, v) = \begin{bmatrix} \frac{d(x(u, v), x)}{\partial u} \\ \frac{d(x(u, v), x)}{\partial v} \end{bmatrix} \]
- Discrete case using central differencing
  \[ n(u, v) = \frac{1}{2\Delta u} \begin{bmatrix} d(u+\Delta u, v) - d(u-\Delta u, v) \\ d(u, v+\Delta v) - d(u, v-\Delta v) \end{bmatrix} \]
- Normalize!

Storing bump maps
- Encode normal direction in RGB color channels
  - Coordinates of unit normal are in \([-1..1]^3\)
  - Need to map range \([-1..1]\) to \([0..255]\) for all channels

Rendering with bump maps
- When applying a bump map to a curved surface, how are the normals specified in the bump map related to the surface?
- Normals are defined relative to local tangent/normal vectors

Tangent space
- Triangle with texture coordinates can be expressed as parametric surface \(x(u, v)\)
- Triangle vertices in object space \(v_0, v_1, v_2\)
- We know \(x(u_0, v_0) = v_0, x(u_1, v_1) = v_1, x(u_2, v_2) = v_2\)

Rendering with bump maps
- Bump map normals are defined in tangent space
  - Defined by two tangent vectors and normal
  - Will define tangent space for each triangle
    - Texture coordinates provide parameterization of each triangle, i.e., function \(x(u, v)\)
    - Compute tangent space using partial derivatives of parameterization
  - Will need to transform normals from tangent space to camera space

Tangent space
- Solve for affine function
  \[ x(u, v) = \begin{bmatrix} m_{0,0} & m_{0,1} & m_{0,2} \\ m_{1,0} & m_{1,1} & m_{1,2} \\ m_{2,0} & m_{2,1} & m_{2,2} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \]
- Using correspondences at vertices
  \[ \begin{bmatrix} m_{0,0} & m_{0,1} & m_{0,2} \\ m_{1,0} & m_{1,1} & m_{1,2} \\ m_{2,0} & m_{2,1} & m_{2,2} \end{bmatrix} \begin{bmatrix} u_0 & u_1 & u_2 \\ v_0 & v_1 & v_2 \end{bmatrix} = \begin{bmatrix} v_0 & v_1 & v_2 \end{bmatrix} \begin{bmatrix} u_0 & u_1 & u_2 \\ v_0 & v_1 & v_2 \end{bmatrix}^{-1} \]
Tangent space

- Tangent space defined by tangent, bi-tangent, and normal vectors
- \( t = \frac{\partial x(u, v)}{\partial u} \) (\( m_{00,0} \ m_{00,1} \ m_{00,2} \))
- \( b = \frac{\partial x(u, v)}{\partial v} \) (\( m_{10,0} \ m_{10,1} \ m_{10,2} \))
- \( n = t \times b \)
- \( t, b, n \) defined in object space coordinates
- Tangent, bi-tangent not orthogonal in general
- No normalization necessary

Normal in object space

- Normal map stores normals in tangent coordinates
  - Basis vectors \( t, b, n \)
- Can transform normal from tangent to object space
  - Values \([bm_0, bm_1, bm_2]\) from bump map
  - Unpacked from \([0..1]\) to range \([-1..1]\)
  - Normalization

\[
\mathbf{n}_{\text{object space}} = \begin{bmatrix} t & b & n \end{bmatrix} \begin{bmatrix} bm_0 \\ bm_1 \\ bm_2 \end{bmatrix}
\]

Storing tangent vectors

Before rendering

- For each triangle, compute tangent, bi-tangent vector
- At each vertex, average tangent, bi-tangent vectors over adjacent triangles
- Store tangent, bi-tangent vectors as additional vertex attributes

Rendering

Vertex shader

- Per-vertex input
  - Vertex position, tangent, bi-tangent vector in object space
  - Bump map texture coordinates
- Transform everything to camera space using modelview matrix
- Output to fragment shader
  - Vertex position, texture coordinates, tangent, bi-tangent, normal vector in camera space
  - Bump map texture coordinates

Rendering with bump maps

Fragment shader

- Transform normal \([bm_0, bm_1, bm_2]\) stored in bump map to camera coordinates
  - Use \( t, b, n \) basis to transform to object space
  - Use modelview matrix to transform from object space to camera space
  - Normalize \( \mathbf{n}_{\text{camera space}} \)
  - Perform lighting in camera coordinates

Variations

- Perform lighting in different coordinate system than camera space
  - Object space
  - Tangent space
- Tangent space is more efficient
  - Transform light direction to tangent space in vertex shader
  - Interpolate across triangle
  - No need to transform bump mapped normal at each pixel
### Caveats
- Avoid triangles with zero area in texture space
  - Cannot compute valid tangent space
- Avoid triangles with negative area in texture space
  - May happen when texture is mirrored
- Avoid non-uniform stretching of bump map

### Combination with env. map
- “Environment mapped bump mapping” (EMBM)
- Use bump mapped normal to compute reflection vector, look up cube map

### Demo
- AMD RenderMonkey
  - Shader development IDE
  - Supports GLSL, HLSL, etc.
- Illumination_Advanced.rfx
- Reflection_Refractions.rfx

### Env. mapped bump mapping
- Use additional ‘dirt’ texture to modulate strength of reflection from environment map

### Tutorials
- Caution, slightly different derivation
- OpenGL shading language book
  - Bump mapping uses shading in tangent space

### Today
- Bump mapping
- Shadows
Why are shadows important?

- Cues on scene lighting

Why are shadows important?

- Contact points
- Depth cues

Terminology

- Umbra: fully shadowed region
- Penumbra: partially shadowed region

Hard and soft shadows

- Point and directional lights lead to hard shadows, no penumbra
- Area light sources lead to soft shadows, with penumbra
**Shadows for interactive rendering**
- Focus on hard shadows
- Two main techniques
  - Shadow mapping
  - Shadow volumes
- Many variations, subtleties
- Active research area

**Shadow mapping**

**Main idea**
- Scene point is lit by light source if it is visible from light source
- Determine visibility from light source by placing camera at light source position and rendering scene

**Two pass algorithm**

**First pass**
- Render scene by placing camera at light source position
- Store depth image (shadow map)

**Second pass**
- Render scene from camera position
- At each pixel, compare distance to light source with value in shadow map
  - If distance is larger, we are in shadow
  - If distance is smaller or equal, pixel is lit

**Issues**
- Limited field of view of shadow map
- Z-fighting
- Sampling problems

**Limited field of view**
- What if a scene point is outside the field of view of the shadow map?
**Limited field of view**

- What if a scene point is outside the field of view of the shadow map?
- Use six shadow maps, arranged in a cube
- Requires rendering pass for each shadow map!

**z-fighting**

- Depth values for points visible from light source are equal in both rendering passes
- Because of limited resolution, depth of pixel visible from light could be larger than shadow map value
- Need to add bias in first pass to make sure pixels are lit

**Solution**

- Add bias when rendering shadow map
  - Move geometry away from light by small amount
- Finding correct amount of bias is tricky

**Bias**

- Not enough bias
- Too much bias
- Correct bias

**Sampling problems**

- Shadow map pixel may project to many image pixels
- Ugly stair-stepping artifacts

**Solutions**

- Increase resolution of shadow map
  - Not always sufficient
- Split shadow map into several slices
- Tweak projection for shadow map rendering
  - Light space perspective shadow maps (LiSPSM)
    - [http://www.cg.tuwien.ac.at/research/vr/lispsm/](http://www.cg.tuwien.ac.at/research/vr/lispsm/)
    - With GLSL source code!
- Combination of splitting and LiSPSM
  - Basis for most serious implementations
Percentage closer filtering

- Instead of looking up one shadow map pixel, look up several pixels
- Perform depth test for each shadow map pixel
- Compute percentage of pixels that are lit

[Diagram showing depth testing and percentage closer filtering]

Percentage closer filtering

- Supported in hardware for small filters (2x2 shadow map pixels)
- Can use larger filters with additional rendering passes
- Fake soft shadows

First pass

- Each vertex point is transformed by
  \[ P_{light} v_{light} M \]
  - Object-to-world (modeling) matrix \( M \)
  - World-to-light space matrix \( v_{light} \)
  - Light frustum (projection) matrix \( P_{light} \)
- Remember: points within frustum are transformed to unit cube \([-1,1]^3\)
**Second pass**

- Render scene from camera
- At each pixel, look up corresponding location in shadow map
- Compare depths with respect to light source

**Looking up shadow map**

- Need to transform each point from object space to shadow map
- Shadow map texture coordinates are in \([0,1]^2\)
- Transformation from object to shadow map coordinates

\[
\begin{bmatrix}
 s \\
 t \\
 r \\
 q \\
\end{bmatrix} = T
\]

- \(T\) is called texture matrix
- After perspective projection we have shadow map coordinates

**GLSL specifics**

**In application**

- Store matrix \(T\) in OpenGL texture matrix
- Set using glMatrixMode(GL_TEXTURE)

**In vertex shader**

- Access texture matrix through predefined uniform gl_TextureMatrix

**In fragment shader**

- Declare shadow map as sampler2DShadow
- Look up shadow map using projective texturing with vec4 texture2DProj(sampler2D, vec4)

**Implementation specifics**

- When you do a projective texture look up on a sampler2DShadow, the depth test is performed automatically
  - Return value is \((1,1,1,1)\) if lit
  - Return value is \((0,0,0,1)\) if shadowed
- Simply multiply result of shading with current light source with this value

**Demo**

- Cg tutorial examples shadowMapping
Resources

- Overview, lots of links
  http://www.realtimerendering.com/
- Basic shadow maps
- Avoiding sampling problems in shadow maps
  http://www.cg.tuwien.ac.at/research/vw/lspsm/
- Faking soft shadows with shadow maps
  http://people.csail.mit.edu/ericchan/papers/smoothsh/
- Alternative: shadow volumes
  http://www.gamedev.net/reference/articles/article1873.asp

Next time

- Procedural modeling