Announcements

• No lab hours Wed Nov 25

• Late homework submission deadline for project 6: Fri Nov 27, 5pm; must be submitted to instructor and TAs by email, then demonstrated to a TA next week

• Points for final project will go online by 6pm

• Deadline for project description submission (email): Mon Nov 30th

• Update: e-mail description to instructor at jschulze AT ucsd.edu, not to TAs
Lecture Overview

Advanced shader effects

- Toon shading
- Shadow mapping
Toon shading

- A.k.a. cel shading
- Simple cartoon style shader
- Emphasize silhouettes
- Discrete steps for diffuse shading, highlights
- Non-photorealistic rendering method (NPR)
Toon Shading Demo

http://www.bonzaisoftware.com/npr.html
Toon shading

- **Silhouette edge detection**
  - Compute dot product of viewing direction $v$ and normal $n$

  $$\text{edge} = \max(0, n \cdot v)$$

- **Use 1D texture to define edge ramp**
  uniform sample1D edgeramp; e=textured1D(edgeramp,edge);
Toon shading

- Compute diffuse and specular shading
  \[
  \text{diffuse} = \mathbf{n} \cdot \mathbf{L} \quad \text{specular} = (\mathbf{n} \cdot \mathbf{h})^s
  \]

- Use 1D textures `diffuseramp`, `specularramp` to map diffuse and specular shading to colors

- Final color
  ```
  \text{uniform sampler1D diffuseramp;
  uniform sampler1D specularramp;
  c = e \times (\text{texture1D(diffuse,diffuseramp)} + \text{texture1D(specular,specularramp)});
  ```
More shaders

- OpenGL shading language book
- NVidia shader library
  - Most shaders are in **HLSL** (DirectX’s shader
- NVidia **Cg** toolkit
  - Current version: Cg 2.1
  - Predecessor of GLSL
  - Lots of example shaders
Lecture Overview

Advanced shader effects

• Toon shading

• Shadow mapping
Why are shadows important?

- Give additional cues on scene lighting
Why are shadows important?

- Contact points
- Depth cues
Why are shadows important?

- Realism

Without self-shadowing

With self-shadowing
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
Hard and soft shadows

Hard shadow from point light source

Soft shadow from area light source
Shadows for interactive rendering

- Focus on hard shadows
  - Soft shadows often too hard to compute in interactive graphics

- 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

Scene points are lit if visible from light source

Determine visibility from light source by placing camera at light source position
Two pass algorithm

First pass

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

![Diagram showing depth image seen from light source]
Two pass algorithm

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

Final image with shadows
Issues with shadow maps

- 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

Correct bias  Not enough bias  Too much 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/
  - With GLSL source code!
- Combination of splitting and LiSPSM
  - Basis for most serious implementations
Basic shadow map

Light space perspective shadow map
Demo

- LiSPSM from

http://www.cg.tuwien.ac.at/research/vr/lispsm/videos/shadows_egsr2004.mpg
Shadow mapping with GLSL

First pass

- Render scene by placing camera at light source position
- Compute light view (look at) matrix
  - Similar to computing camera matrix from look-at, up vector
  - Compute its inverse to get world-to-light transform
- Determine view frustum such that scene is completely enclosed
  - Use several view frusta/shadow maps if necessary
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 \([-1,1]^3\)
First pass

- Use `glPolygonOffset` to apply depth bias
- Store depth image in a texture
  - Use `glCopyTexImage` with internal format `GL_DEPTH_COMPONENT`

Final result with shadows

Scene rendered from light source

Depth map from light source
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

\[
T = \begin{pmatrix}
1/2 & 0 & 0 & 1/2 \\
0 & 1/2 & 0 & 1/2 \\
0 & 0 & 1/2 & 1/2 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

- $T$ is called texture matrix
- After perspective projection we have shadow map coordinates
Looking up shadow map

- Transform each vertex to normalized frustum of light

\[
\begin{bmatrix}
  s \\
  t \\
  r \\
  q \\
\end{bmatrix} = T \begin{bmatrix}
  x \\
  y \\
  z \\
  1 \\
\end{bmatrix}
\]

- Pass \(s, t, r, q\) as texture coordinates to rasterizer
- Rasterizer interpolates \(s, t, r, q\) to each pixel
- Use **projective texturing** to look up shadow map
  - This means, the texturing unit automatically computes \(s/q, t/q, r/q, 1\)
  - \(s/q, t/q\) are shadow map coordinates in \([0,1]^2\)
  - \(r/q\) is depth in light space
- Shadow depth test: compare shadow map at \((s/q, t/q)\) to \(r/q\)
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

- Shadow mapping demo from

http://www.paulsprojects.net/opengl/shadowmap/shadowmap.html
Next Lecture

- Date: Tuesday Dec 1\textsuperscript{st}

Happy Thanksgiving!