CSE167: Introduction to Computer Graphics

Lecture #16

Jürgen Schulze
University of California, San Diego
Fall 2008
Announcements

- Regular lab hours with Robert on Wed Nov 26th, 2-4pm
- Late homework submission deadline Mon Dec 1st
- Deadline for project description submission (email): Mon Dec 1st
- Guest lecturer Raj Singh from NCMIR to talk about CUDA programming on Tue Dec 2nd
Hard and soft shadows

Hard shadow from point light source

Soft shadow from area light source
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
  - Soft shadows often too hard to compute in interactive graphics

• Two main techniques
  - Shadow mapping
  - Shadow volumes
Today

- Shadow mapping
- Shadow volumes
- Procedural modeling
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*)

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
Questions?
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
Bias

Not enough

Correct

Too much
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
LiSPSM

Basic shadow map

Light space perspective shadow map
Demo

• LiSPSM from

http://www.cg.tuwien.ac.at/research/vr/lispsm/videos/shadows_egsr2004.mpg
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
Percentage closer filtering

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

\[ P_{\text{light}} V_{\text{light}} M \]

- Object-to-world (modeling) matrix \( M \)
- World-to-light space matrix \( V_{\text{light}} \)
- Light frustum (projection) matrix \( P_{\text{light}} \)

- 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:
  \[ T = P_{light} V_{light} M \]
- $T$ is called texture matrix.
- After perspective projection we have shadow map coordinates.

![Diagram showing transformation from object space to shadow map coordinates](image)
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
Questions?
Today

- Shadow mapping
- Shadow volumes
- Procedural modeling
Shadow volumes

Light source

Eye position
*(note that shadows are independent of the eye position)*

Shadowing object

Partially shadowed object

Surface outside shadow volume *(illuminated)*

Surface inside shadow volume *(shadowed)*

Shadow volume *(infinite extent)*
Shading with shadow volumes

- Many variations
- Stencil shadow volumes
  - Classic algorithm
  - Hard shadows
- Here, two-pass algorithm for approximate soft shadows
  - Very simple and inaccurate, but often plausible enough
- Many more complicated and more accurate variations exist
Shadow volume construction

- Need to generate shadow polygons to bound shadow volume
- Extrude silhouette edges from light source

Extruded shadow volumes
Shadow volume construction

- Needs to be done on the CPU
- Silhouette edge detection
  - An edge is a silhouette if one adjacent triangle is front facing, the other back facing with respect to the light
- Extrude polygons from silhouette edges
Two-pass algorithm

First pass

• Render scene without shading, store depth buffer

Second pass

• Use depth image from first pass for z-buffering

• Render polygons of shadow volume only

• For each rasterized pixel, compute a fractional shadow value in a fragment shader
**Two-pass algorithm**

Approximation

- Shadow value depends on distance to center of occluder, distance to center of shadow volume

Second pass

- Construct a local coordinate system for each shadow volume
  - Origin is center of occluder
  - z-axis is center of shadow volume
- Transform location of each pixel to shadow volume coordinate system
- Use z-coordinate and distance to z-axis to look up shadow texture
Demo

- Shadow mapping demo from
  
  http://www.paulsprojects.net/opengl/shadowmap/shadowmap.html
Resources

- Overview, lots of links

- Basic shadow maps

- Avoiding sampling problems in shadow maps
  [http://www.cg.tuwien.ac.at/research/vr/lispsm/](http://www.cg.tuwien.ac.at/research/vr/lispsm/)

- Faking soft shadows with shadow maps

- Alternative: shadow volumes
Questions?
Today

- Shadow mapping
- Shadow volumes
- Procedural modeling
Modeling

- Creating 3D objects/scenes and defining their appearance (texture, etc.)
- So far we saw
  - Triangle meshes
  - Bezier patches
- Interactive modeling
  - Place vertices, control points manually
- For realistic scenes, need extremely complex models containing millions or billions of primitives
- Modeling everything manually extremely tedious
Alternatives

• Data-driven modeling
  - Scan model geometry from real world examples
  - Use laser scanners or similar devices
  - Photographs as textures

• Procedural modeling
  - Construct 3D models and textures using algorithms

• Examples
  - http://www-graphics.stanford.edu/data/3Dscanrep/
  - http://www.tsi.enst.fr/3dmodels/
  - .ply file format reader
    http://www.tecgraf.puc-rio.br/~diego/professional/rply/
Procedural modeling

- Wide variety of techniques for algorithmic model creation
- Used to create models that are too complex (or tedious) for a person to build
  - Trees, landscapes (mountains), clouds, etc.
  - Plants, ecosystems
  - Buildings, cities
- Usually defined by a small set of data, or rules, that describes the overall properties of the model
  - Tree defined by branching properties and leaf shapes
- Model is constructed by an algorithm (procedure)
  - Often including randomness to add variety
  - E.g., a single tree pattern can be used to model an entire forest

[Deussen et al.]
Randomness

• Use some sort of randomness to make models more interesting, natural, less uniform, clean

• *Pseudorandom* number generation algorithms
  - Produce a sequence of (apparently) random numbers based on some initial seed value

• Pseudorandom sequences are repeatable, as one can always reset the sequence
  - E.g., if tree is built using several random numbers, then the entire tree can be rebuilt by just resetting the seed to its initial value
  - If the seed is set to a different value, a different sequence of numbers will be generated, resulting in a slightly different tree
Recursion

- Repeatedly apply the same operation (set of operations) to an object

- Generate objects that are self-similar, fractals
  - Object that look the same when viewed at different scales

- For example, the shape of a coastline may appear as a jagged line when we view a map of California
  - As we zoom in closer and closer, we see that there is more and more detail at finer scales
  - We always see a jagged line no matter how close we look at the coastline
Height fields

- Landscapes are often constructed as *height fields*
- Regular grid in the ground plane (assume xz plane here)
- Store a height \((y)\) value at each point
- Can store large terrain in memory
  - No need to store xz coordinates, connectivity
- Shaped terrain by operations that modify the \(y\) coordinates
- Can interpret height values as gray scale values
  - Apply image processing tools
Fractal landscapes

- Random midpoint displacement algorithm
  - Recursively subdivide triangles
  - Randomly displace edge midpoints
  - Reduce size of displacement as triangles get smaller
  - Similar for quadrilaterals
Fractal landscapes

- Add textures, nice rendering...
- Terragen, free software
  http://www.planetside.co.uk/terragen/

[http://www.planetside.co.uk/gallery/f/tg09]
L-systems

- Developed by a biologist (Lindenmayer) in 1968 to study growth patterns of algae
- Defined by a grammar

\[ G = \{ V, S, \omega, P \} \]

- \( V \) alphabet, set of symbols that can be replaced (variables)
- \( S \) set of symbols that remain fixed (constants)
- \( \omega \) string of symbols defining initial state
- \( P \) production rules
Sierpinski triangle

- Variables: A, B
  - Draw forward
- Constants: +, -
  - Turn left, right by 60 degrees
- Start: A
- Rules: (A→B-A-B), (B→A+B+A)

2 iterations

4 iterations

6 iterations

9 iterations
Fractal fern

- **Variables:** X, F
  - X: no drawing operation
  - F: move forward

- **Constants:** +, −
  - Turn left, right

- **Start:** X

- **Rules:**
  \[(X \rightarrow F-[[X]+X]+F[+FX]-X), (F \rightarrow FF)\]

- **Stochastic L-system**
  - If there is more than one production rule for a symbol, randomly choose one
Fractal trees

- Recursive generation of trees in 3D
  http://web.comhem.se/solgrop/3dtree.htm
- Model trunk, branches as cylinders
- Change color from brown to green at certain level of recursion

Fractal tree

Sierpinski tree
Algorithmic beauty of plants

- Online book on algorithmic beauty of plants by Prusinkiewicz
  http://algorithmicbotany.org/papers/#abop

Buildings, cities

Pascal Mueller
[http://www.vision.ee.ethz.ch/~pmueller/publications.html]
Next time

• Next lecture is on Tuesday Dec 2\textsuperscript{nd}
• Guest lecture on CUDA

Happy Thanksgiving!