Shadow volumes and deferred rendering

Computer Graphics
CSE 167
Lecture 16
CSE 167: Computer graphics

- Shadow volumes
- Deferred rendering
  - Deferred shading
  - Bloom effect
  - Glow effect
  - Screen space ambient occlusion

Based on slides courtesy of Jurgen Schulze
Shadow volumes

NVIDIA md2shader demo
CSE 167, Winter 2020
Shadow volumes

• A single point light source splits the world in two
  – Shadowed regions
  – Unshadowed regions
  – Volumetric shadow technique

• A shadow volume is the boundary between these shadowed and unshadowed regions
  – Determine if an object is inside the boundary of the shadowed region and know the object is shadowed
Shadow volumes

• Many variations of the algorithm exist
• Most popular ones use the stencil buffer
  – Depth Pass
  – Depth Fail (a.k.a. Carmack’s Reverse, developed for Doom 3)
  – Exclusive-Or (limited to non-overlapping shadows)
• Most algorithms designed for hard shadows
• Algorithms for soft shadows exist
Shadow volumes

- Light source
- Eye position
- Partially shadowed object
- Shadowing object
- Surface inside shadow volume (shadowed)
- Shadow volume (infinite extent)
- Surface outside shadow volume (illuminated)

(note that shadows are independent of the eye position)
Shadow volumes

• Given the scene and a light source position, determine the geometry of the shadow volume

• High-level algorithm
  – Render the scene in two passes
    • Draw scene with the light enabled, updating only fragments in unshadowed region
    • Draw scene with the light disabled, updated only fragments in shadowed region
Shadow volume construction

• Need to generate shadow polygons to bound shadow volume
• Extrude silhouette edges from light source
Shadow volume construction

• 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
Stenciled shadow volumes

• Advantages
  – Support omnidirectional lights
  – Exact shadow boundaries

• Disadvantages
  – Fill-rate intensive
  – Expensive to compute shadow volume geometry
  – Hard shadow boundaries, not soft shadows
  – Difficult to implement robustly
The stencil buffer

- Per-pixel 2D buffer on the GPU
- Similarities to depth buffer in way it is stored and accessed
- Stores an integer value per pixel, typically 8 bits
- Like a stencil, allows to block pixels from being drawn
- Typical uses:
  - Shadow mapping
  - Planar reflections
  - Portal rendering
The stencil buffer

• Using the stencil buffer, rendering a stencil mirror tutorial

https://www.youtube.com/watch?v=3xzq-YEOIsk
Tagging pixels as shadowed or unshadowed

• The stenciling approach
  – Clear stencil buffer to zero and depth buffer to 1.0
  – Render scene to leave depth buffer with closest Z values
  – Render shadow volume into frame buffer with depth testing but without updating color and depth, but inverting a stencil bit (Exclusive-Or method)
  – This leaves stencil bit set within shadow
Stencil inverting of shadow volume

Light source

Eye position

Shadowing object

Two inverts, left zero

One invert, left one

Zero inverts, left zero
Visualizing stenciled shadow volume tagging

$\text{red} = \text{stencil value of 1}$

$\text{green} = \text{stencil value of 0}$

GLUT $\text{shadowvol}$ example credit: Tom McReynolds
Shadow volumes with intersecting polygons

• Use a stencil enter/leave counting approach
  – Draw shadow volume twice using face culling
    • 1st pass: render front faces and increment when depth test passes
    • 2nd pass: render back faces and decrement when depth test passes
  – This two-pass way is more expensive than inverting
    • Use inverting if all shadow volumes have no polygon intersections
Increment/Decrement stencil volumes

Light source → Shadowing object

Eye position

Zero → +1 → +2 → +3

(CSE 167, Winter 2020)
Shadow volume demo

http://www.paulsprojects.net/opengl/shadvol/shadvol.html
Shadow volumes

• Resources
  – https://www.gamedev.net/articles/programming/graphics/the-theory-of-stencil-shadow-volumes-r1873/
Deferred rendering
Deferred rendering

- Different than forward rendering (traditional rendering, what we have been covering)
- Deferred rendering defers the final computation until a second pass
  - First pass
    - Data required for the final computation is computed and stored in the geometry buffer (G-buffer)
      - Examples: depth (or position, though position can be calculated from depth), surface normal vector, surface color, surface texture, additional surface properties
  - Second pass
    - An algorithm, typically implemented as a shader, generates the final image using the G-buffer and possibly other data
Deferred shading

• Defers shading until the second pass
  – Only fragments determined to be visible are shaded

• First pass
  – Fill the G-buffer with data required to compute shading
    • Position, surface normal vector, surface properties

• Second pass
  – Compute shading for each pixel given light sources and the G-buffer
Deferred shading

• Advantages
  – Decouples lighting from scene geometry
  – Reduces per pixel shading complexity
    • Forward rendering $O(\text{num}_\text{fragments} \times \text{num}_\text{lights})$
      – Fragments from all objects
    • Deferred rendering $O(\text{num}_\text{lights})$
      – Only the fragment that is visible
    • For example, greater than 1000 lights can be rendered at 60 fps

• Disadvantages
  – More expensive (memory, bandwidth, shader instructions)

• Tutorial:
Deferred shading

• Example G-buffer

![Normal vector (color encoded)](image1)

![Specular color](image2)

![Color](image3)

![Depth (color encoded)](image4)
Deferred shading

• Example result
Bloom effect

• Computer displays have limited dynamic range
• Bloom gives a scene a look of bright lighting and overexposure
• Provides visual cues about brightness and atmosphere
  – Caused by light scattering in atmosphere, or within our eyes

Left: no bloom, right: bloom. Source: https://jmonkeyengine.org
Bloom effect

- Step 1: Extract all highlights of the rendered scene, superimpose them and make them more intense
  - Operates on G-buffer
  - Often done with G-buffer smaller (lower resolution) than frame buffer
  - Highlights found by thresholding luminance
- Step 2: Blur G-buffer (e.g., using a Gaussian low pass filter)
- Step 3: Composite off-screen buffer with back buffer
Bloom effect

• Example result

Without bloom  

With bloom
Glow effect

• Use a G-buffer for “glow sources”
  – Render entire scene to back buffer
  – Render only glowing objects to a smaller (lower resolution) off-screen G-buffer (can be combined with bloom G-buffer)
  – Blur G-buffer (e.g., using a Gaussian low pass filter)
  – Composite off-screen buffer with back buffer
Example: glowing lava

https://www.youtube.com/watch?v=hmsMk-skquI
Bloom and glow references

• Bloom Tutorial
  https://prideout.net/blog/old/archive/bloom/index.php.html

• GPU Gems Chapter on Glow

• GLSL Shader for Gaussian Blur
  https://www.ozone3d.net/tutorials/image_filtering_p2.php
Ambient occlusion

• Coarse approximation of global illumination
• Often referred to as "sky light"
• Global method
  – Illumination at each point is a function of other geometry in the scene
• Appearance is similar to what objects appear as on an overcast day
  – Assumption: concave objects are hit by less light than convex ones
Screen space ambient occlusion (SSAO)

- Screen space = deferred rendering
- Approximates ambient occlusion in real time
- Developed by Vladimir Kajalin (Crytek)
- First use in PC game Crysis (2007)
Basic SSAO algorithm

• First pass
  – Render scene normally and copy Z values to G-buffer alpha channel

• Second pass
  – Pixel shader samples depth values around the processed fragment and computes amount of occlusion, stores result in G-buffer red channel
  – Occlusion depends on depth difference between sampled fragment and currently processed fragment
SSAO algorithm with surface normals

• First pass
  – Render scene normally and copy Z values to G-buffer alpha channel and surface normal vectors to G-buffer RGB channels

• Second pass
  – Use Z values and surface normal vectors to compute occlusion between current pixel and several samples around that pixel

Without SSAO  CSE 167, Winter 2020  With SSAO
Screen space ambient occlusion (SSAO)

• Advantages
  – Deferred rendering algorithm: independent of scene complexity
  – No pre-processing, no memory allocation in RAM
  – Works with dynamic scenes
  – Works in the same way for every pixel
  – No CPU usage: executed completely on GPU

• Disadvantages
  – Local and view-dependent (dependent on adjacent texel depths)
  – Hard to correctly smooth/blur out noise without interfering with depth discontinuities, such as object edges, which should not be smoothed out
SSAO References

• Nvidia’s documentation