CSE 167: Introduction to Computer Graphics Lecture #15: Shadow Volumes

Jürgen P. Schulze, Ph.D.
University of California, San Diego

Fall Quarter 2014

Announcements

- Project 5 late grading and Project 6 due tomorrow
- Final Project on-line now, due December 18th at 3pm
- TA Evaluations Dec 1-Dec 15
- CAPE

Shadow Mapping With GLSL

First Pass

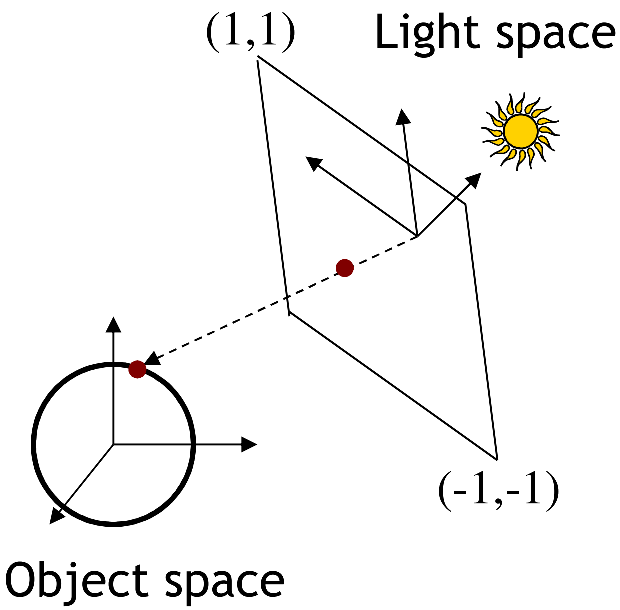
- Render scene by placing camera at light source position
- Compute light view (look at) matrix
  - Similar to computing camera matrix from look-at, up vector
  - Compute its inverse to get world-to-light transform
- Determine view frustum such that scene is completely enclosed
  - Use several view frusta/shadow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

First Pass

- Each vertex point is transformed by

$$\mathbf{P}_{light}\mathbf{V}_{light}\mathbf{M}$$

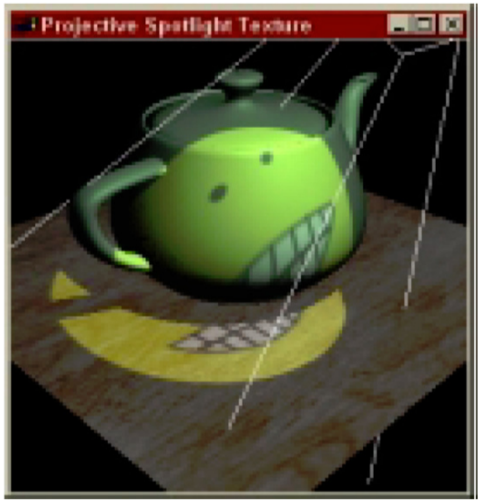
  - Object-to-world (modeling) matrix $\mathbf{M}$
  - World-to-light space matrix $\mathbf{V}_{light}$
  - Light frustum (projection) matrix $\mathbf{P}_{light}$
- Remember: points within frustum are transformed to unit cube $[-1,1]^3$



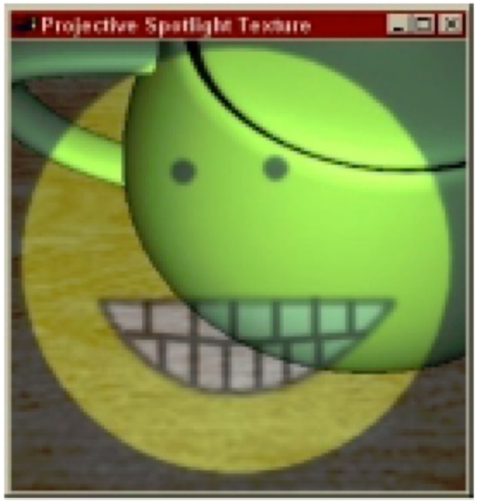
UCSD

First Pass

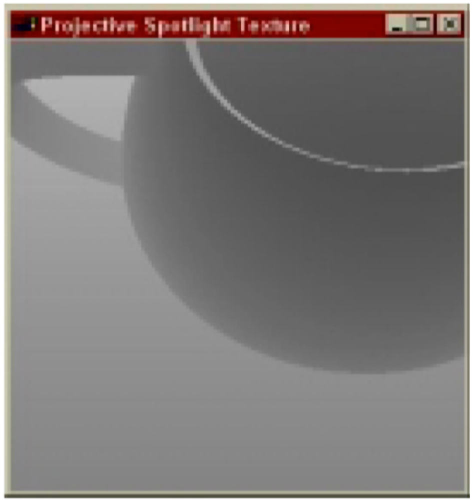
- Use glPolygonOffset to apply depth bias
- Store depth image in a texture
  - Use glCopyTexImage with internal format GL_DEPTH_COMPONENT



Final result
with shadows



Scene rendered
from light source



Depth map
from light source

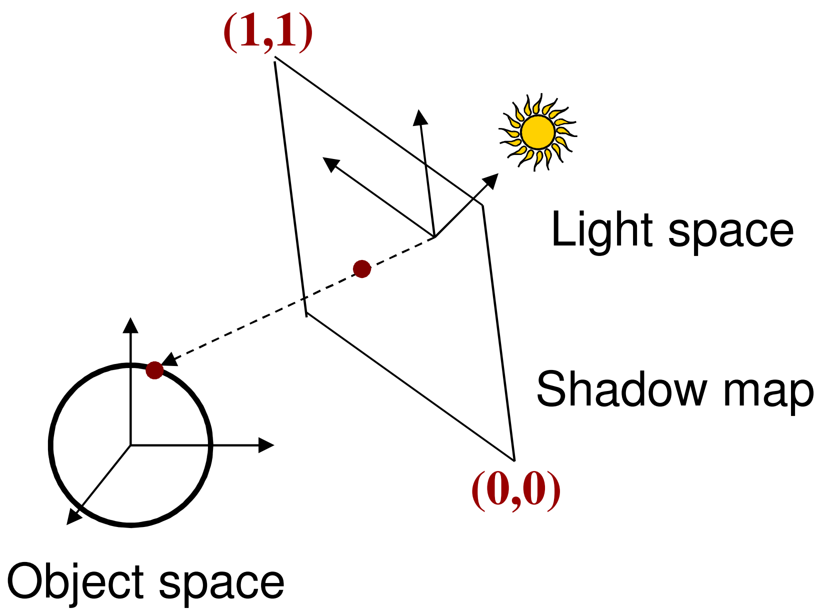
Second Pass

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

Shadow Map Look-Up

- 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

$$\mathbf{T} = \begin{bmatrix} 1/2 & 0 & 0 & 1/2 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{P}_{light} \mathbf{V}_{light} \mathbf{M}$$



- $\mathbf{T}$ is called texture matrix
- After perspective projection we have shadow map coordinates

Shadow Map Look-Up

- Transform each vertex to normalized frustum of light

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

- 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

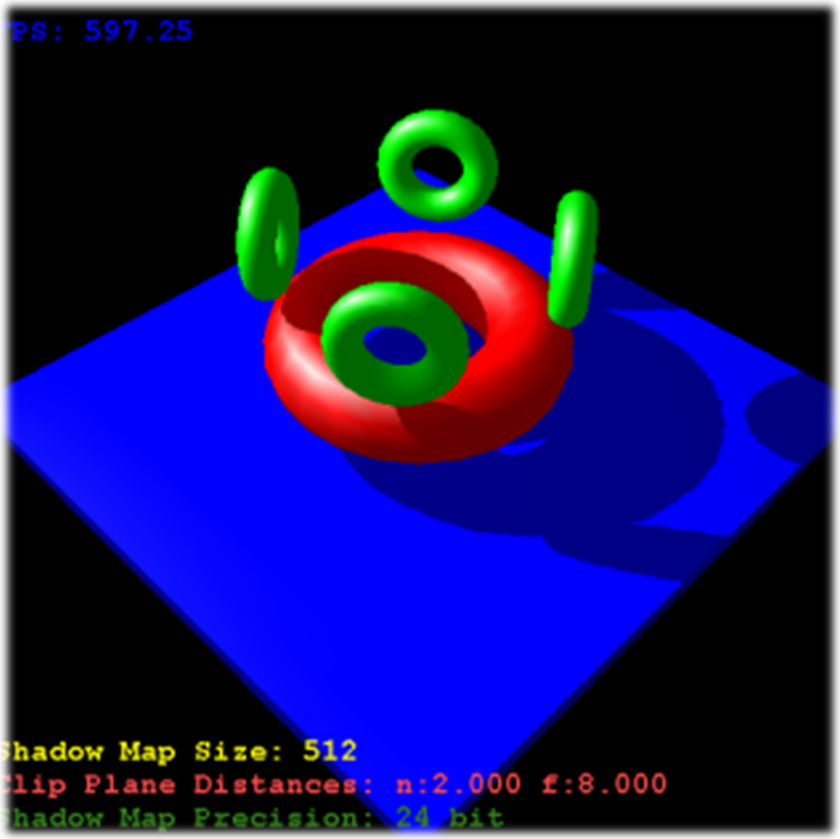
- Declare shadow map as sampler2DShadow
- Look up shadow map using projective texturing with `vec4 texture2DProj(sampler2D, vec4)`

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

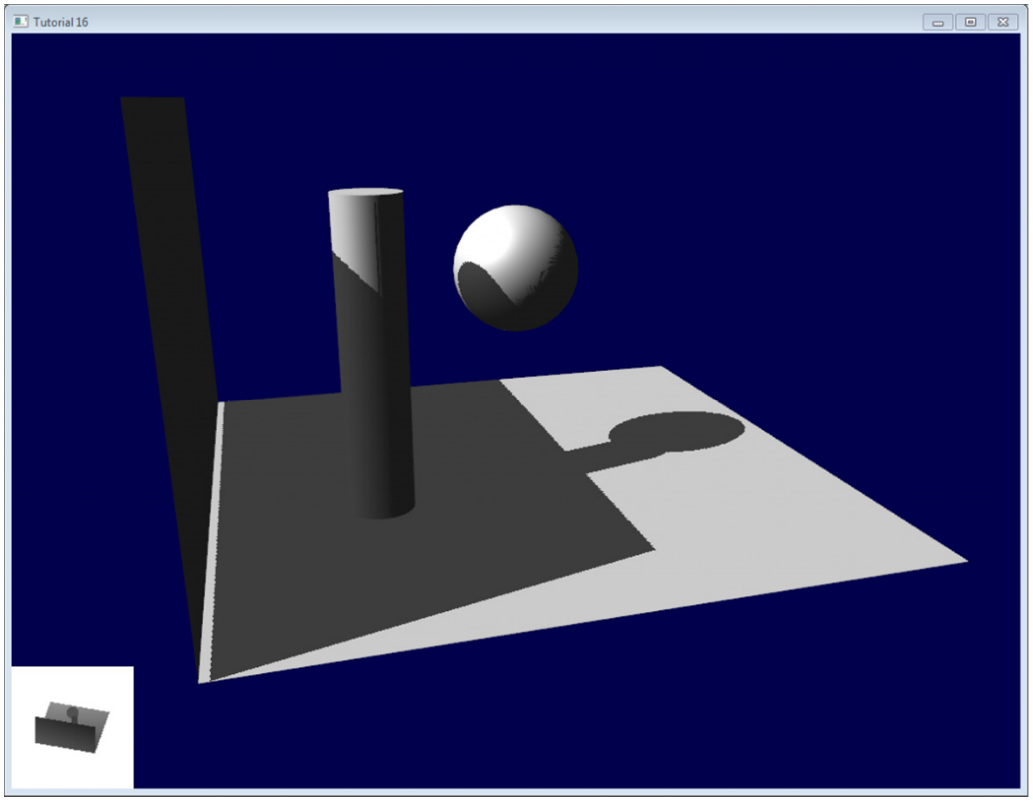
- Shadow mapping demo from http://www.paulsprojects.net/opengl/shadowmap/shadowmap.html



```
PS: 597.25
hadow Map Size: 512
lip Plane Distances: n:2.000 f:8.000
hadow Map Precision: 24 bit
```

Tutorial URL

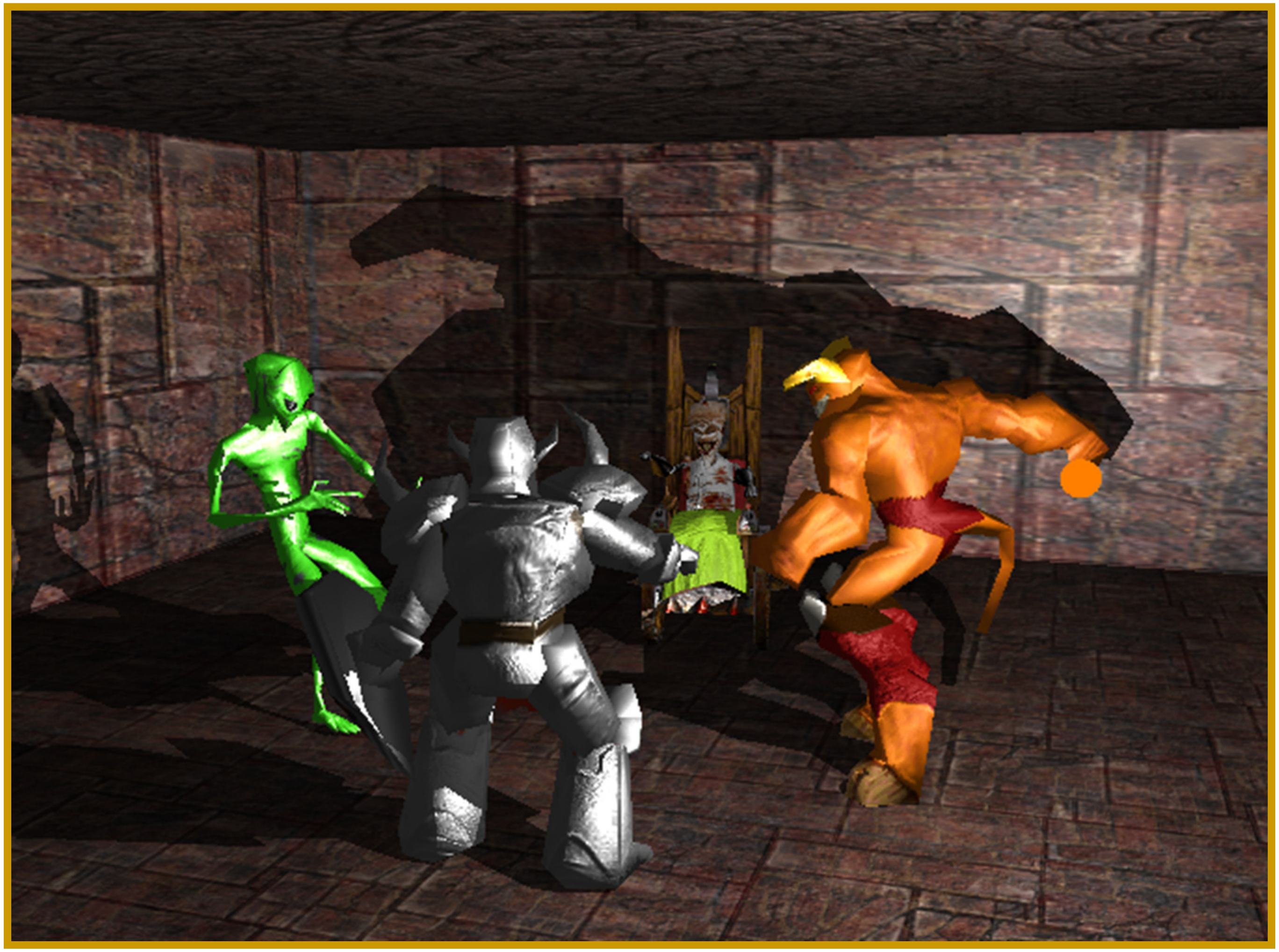
- http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/



Lecture Overview

- Shadow Volumes

Shadow Volumes



NVIDIA md2shader demo

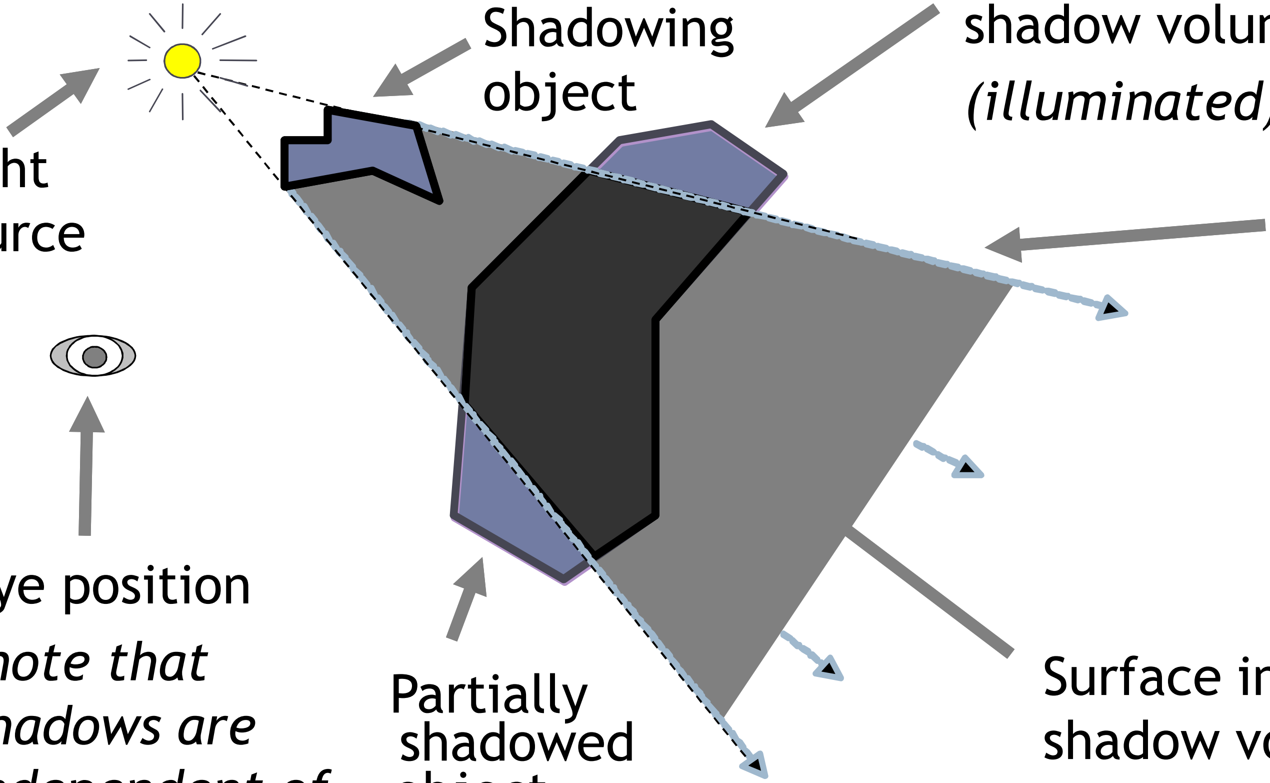
Shadow Volumes

- 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

Shadowing object

Surface outside shadow volume

(illuminated)

Shadow volume

(*infinite extent*)

Eye position

(note that shadows are independent of the eye position)

Partially shadowed object

Surface inside shadow volume

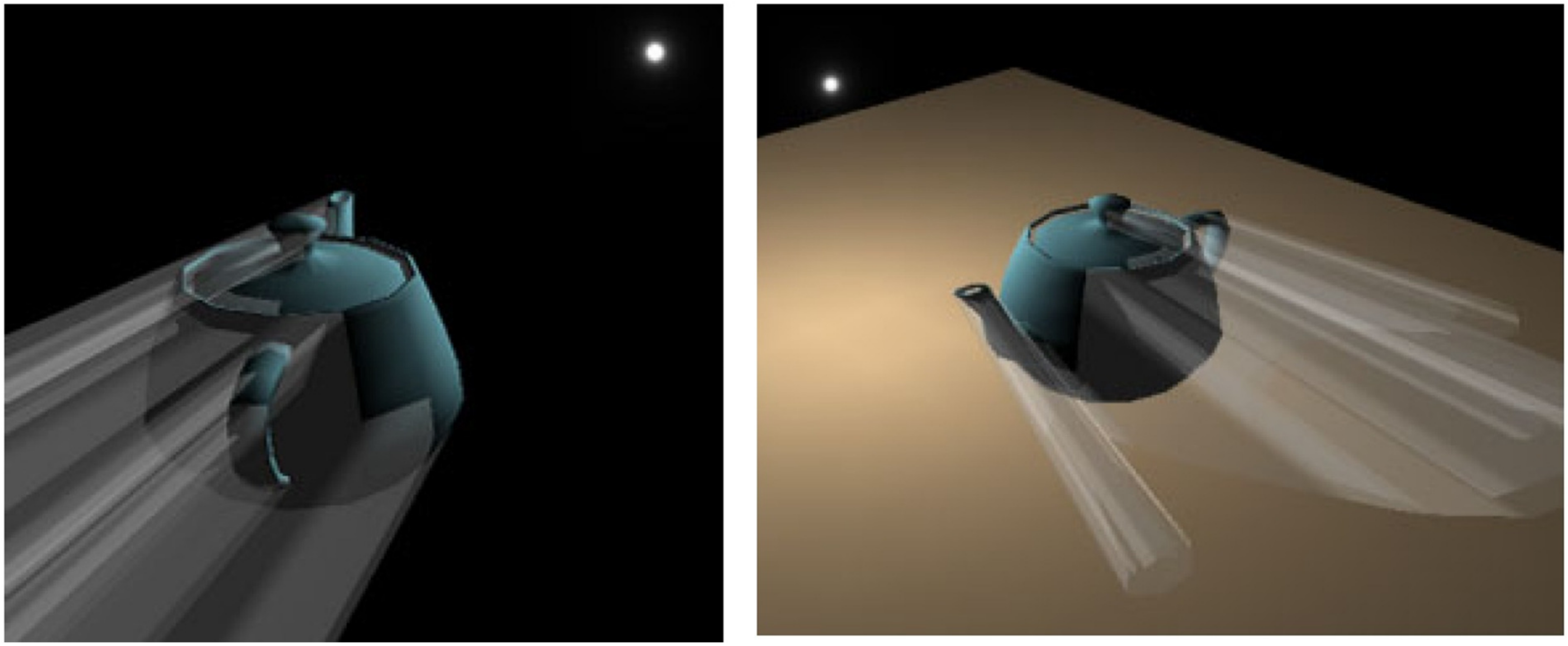
(shadowed)

Shadow Volume Algorithm

- High-level view of the algorithm
  - Given the scene and a light source position, determine the geometry of the shadow volume
  - 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

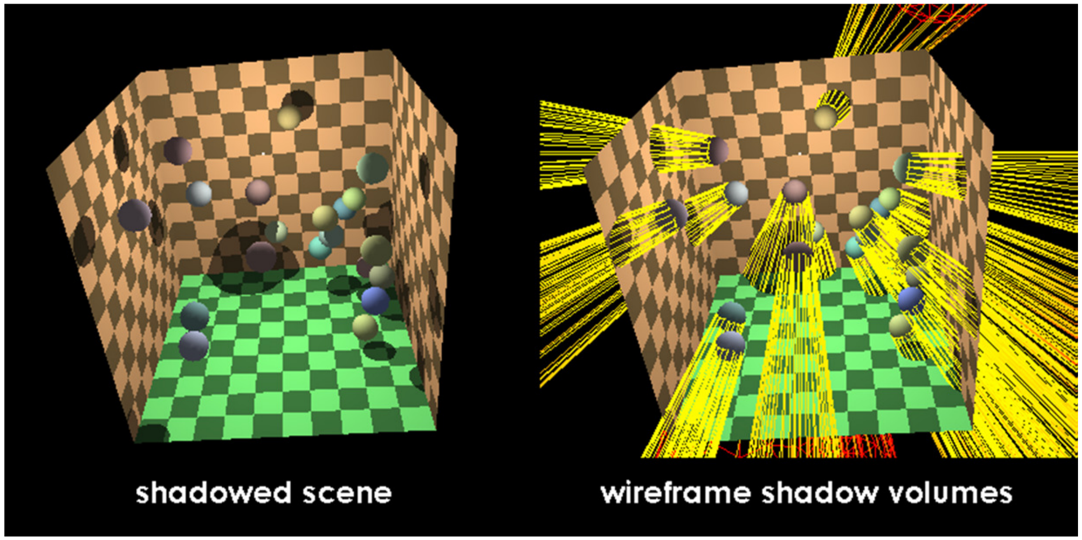
- Need to generate shadow polygons to bound shadow volume
- Extrude silhouette edges from light source



Extruded shadow volumes

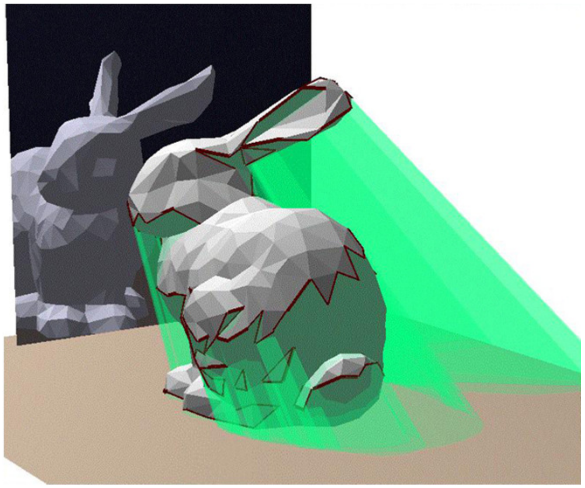
Shadow Volume Construction

- 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

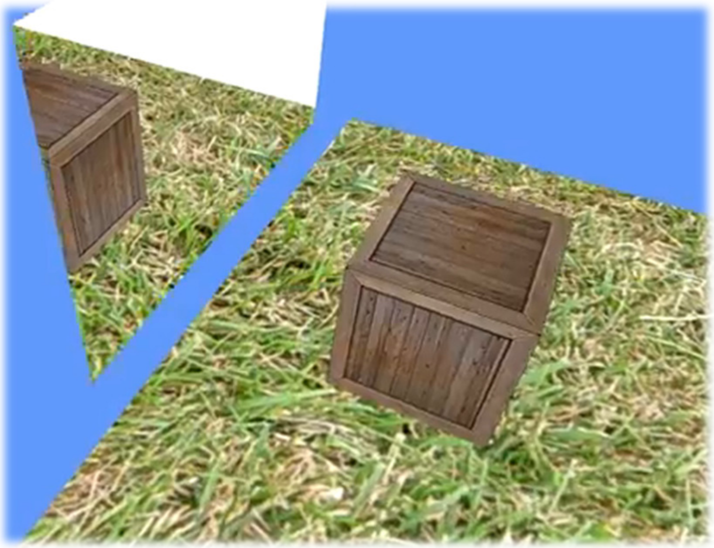


Source: Zach Lynn

UCSD

The Stencil Buffer

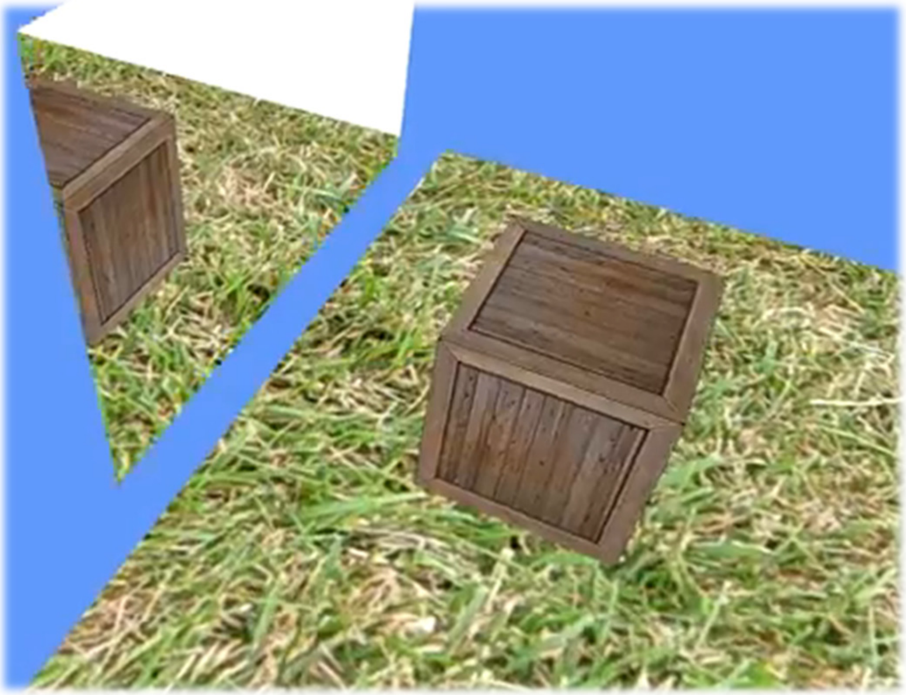
- Per-pixel 2D buffer on the GPU
- Similarities to depth buffer in way it is stored and accessed
- Stores an integer value per pixel, typically 8 bits
- Like a stencil, allows to block pixels from being drawn
- Typical uses:
  - shadow mapping
  - planar reflections
  - portal rendering



Source: Adrian-Florin Visan

Video

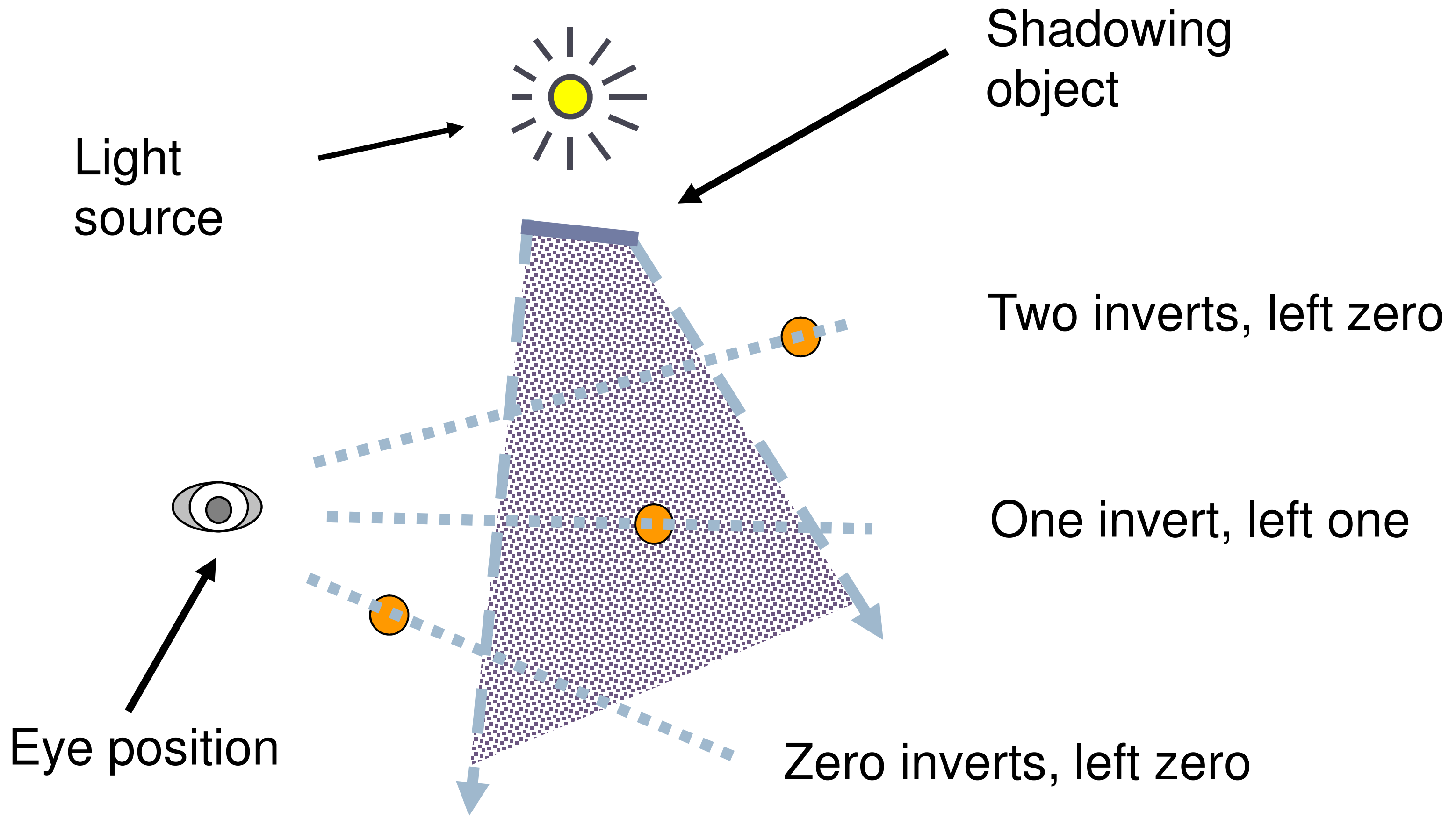
- Using the stencil buffer, rendering a stencil mirror tutorial
  - http://www.youtube.com/watch?v=3xzq-YEOlsk



Tagging Pixels as Shadowed or Unshadowed

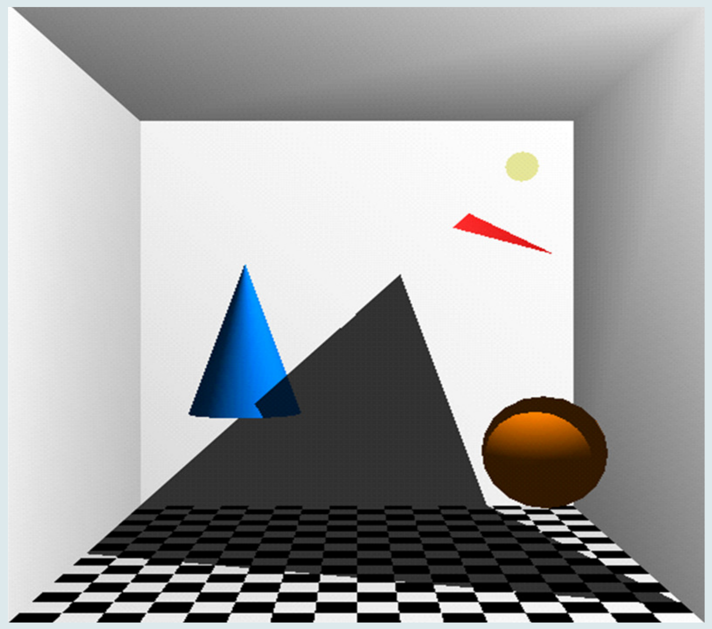
- The stenciling approach
  - Clear stencil buffer to zero and depth buffer to 1.0
  - Render scene to leave depth buffer with closest Z values
  - Render shadow volume into frame buffer with depth testing but *without* updating color and depth, but *inverting* a stencil bit (Exclusive-Or method)
  -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

Stencil Inverting of Shadow Volume

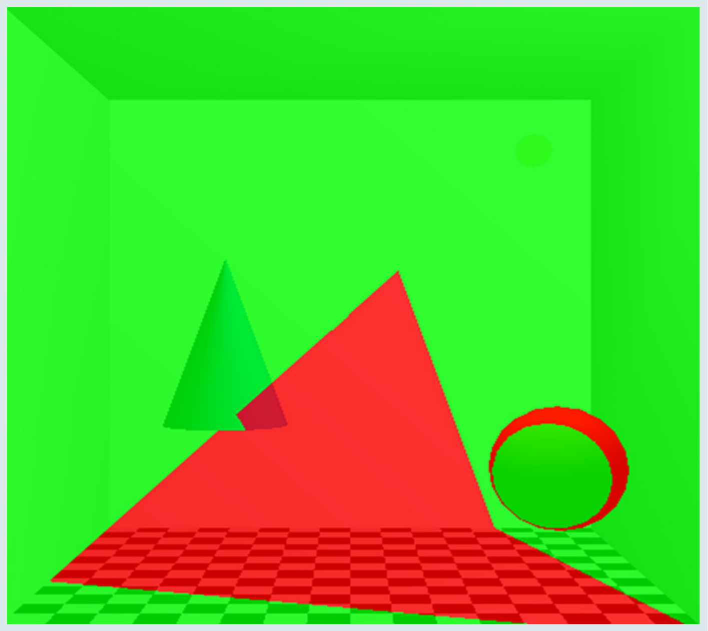


Visualizing Stenciled Shadow Volume Tagging

Shadowed scene



Stencil buffer contents



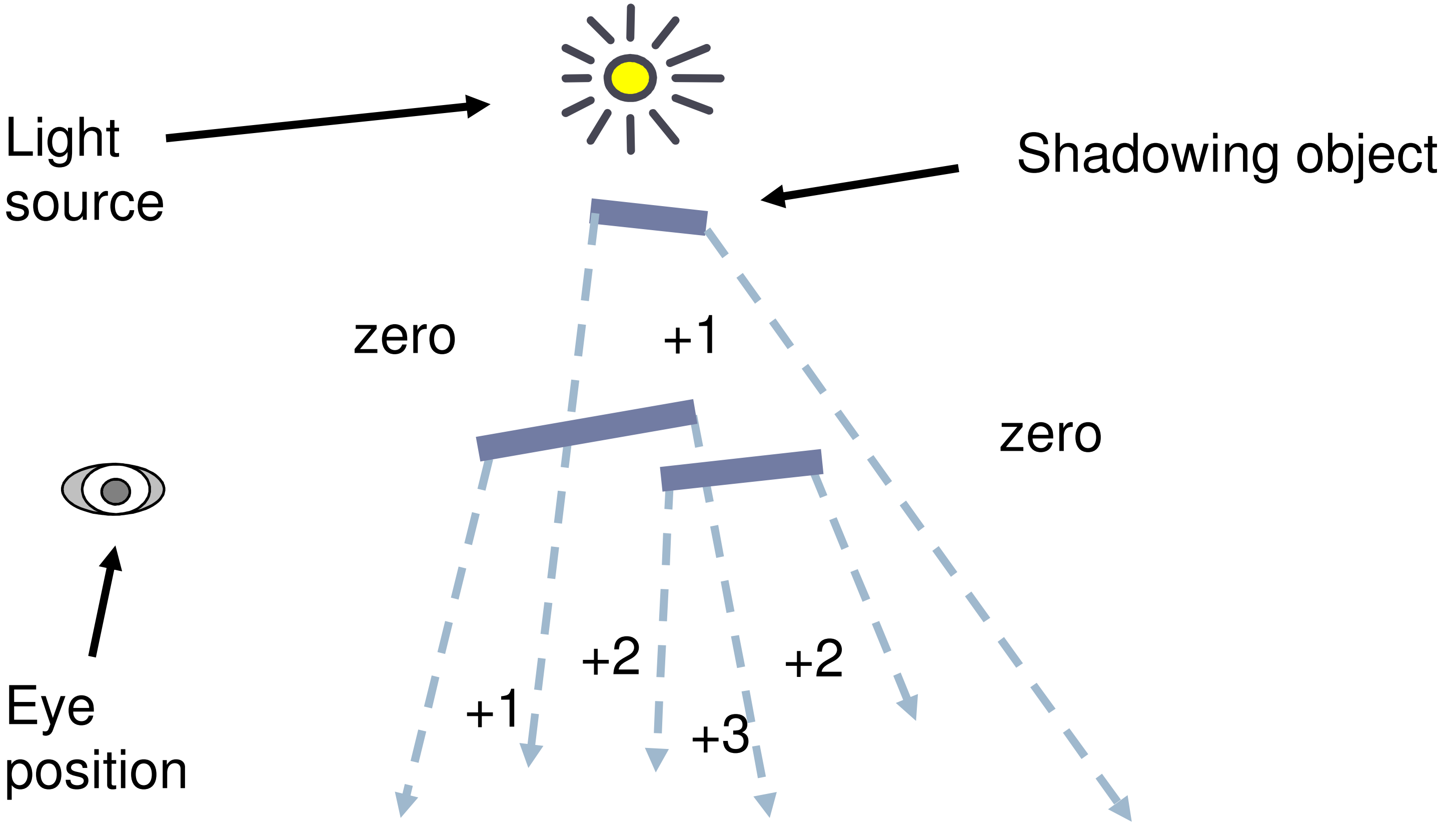
red = stencil value of 1
green = stencil value of 0

GLUT *shadowvol* example credit: Tom McReynolds

For 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 invert
  - Inverting is better if all shadow volumes have no polygon intersections

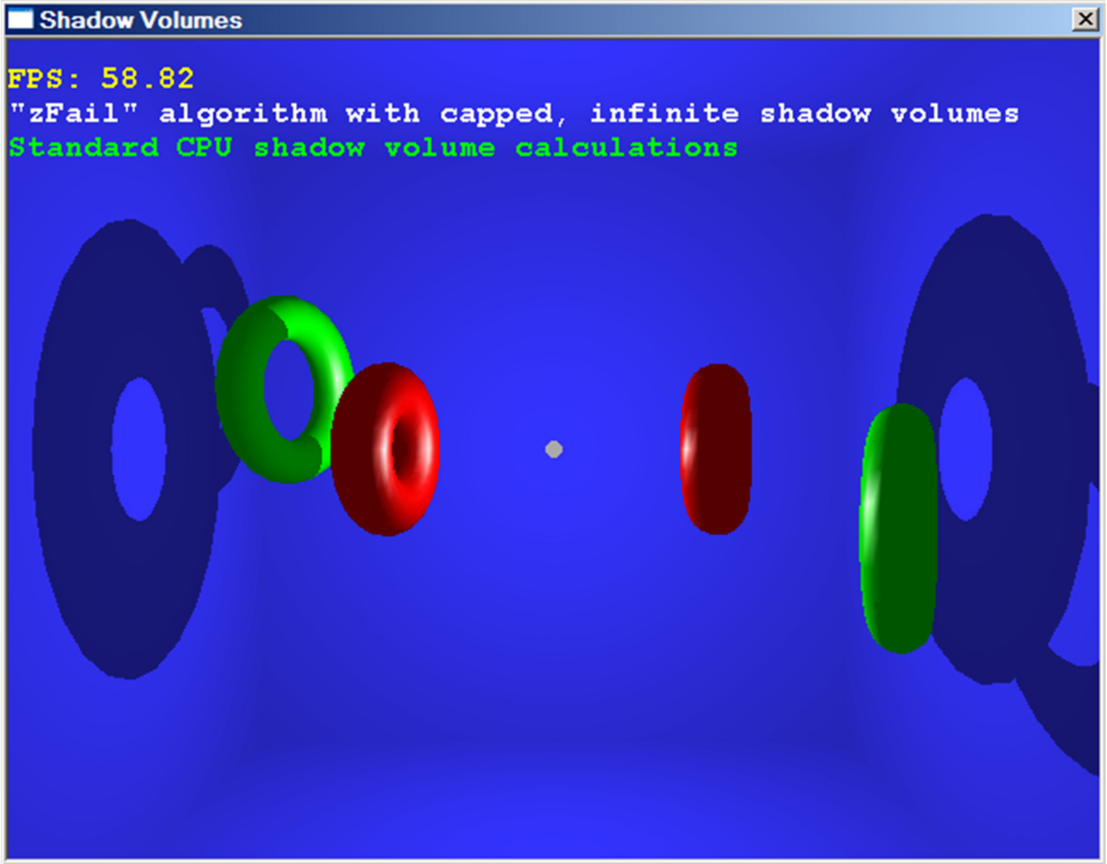
Increment/Decrement Stencil Volumes



Shadow Volume Demo

- URL:

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



FPS: 58.82
"zFail" algorithm with capped, infinite shadow volumes
Standard CPU shadow volume calculations

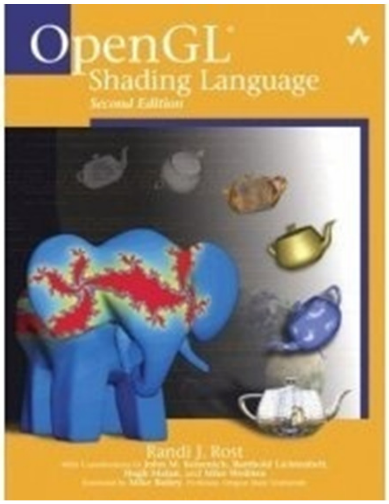
UCSD

Resources for Shadow Rendering

- Overview, lots of links
  http://www.realtimerendering.com/
- Basic shadow maps
  http://en.wikipedia.org/wiki/Shadow_mapping
- Avoiding sampling problems in shadow maps
  http://www.comp.nus.edu.sg/~tants/tsm/tsm.pdf
  http://www.cg.tuwien.ac.at/research/vr/lispsm/
- Faking soft shadows with shadow maps
  http://people.csail.mit.edu/ericchan/papers/smoothie/
- Alternative: shadow volumes
  http://en.wikipedia.org/wiki/Shadow_volume
  http://www.gamedev.net/reference/articles/article1873.asp

More on Shaders

- OpenGL shading language book
  - "Orange Book"



- Shader Libraries
  - GLSL:
    - http://www.geeks3d.com/geexlab/shader_library.php
  - HLSL:
    - NVidia shader library
    - http://developer.download.nvidia.com/shaderlibrary/webpages/shader_library.html