CSE 167:
Introduction to Computer Graphics
Lecture #14: Shadow Mapping

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2010

Announcements

- Final project outline due November 19
- Second midterm exam: Tuesday November 23rd during lecture hours (2-3:20pm)
- Midterm review session:
  - Wed Nov 17th, 1-2:30pm
  - Atkinson hall, room 4004
  - Will have index cards
- Final project presentations: Friday December 3rd, 2-4pm

- **TA Evaluation for Han Suk Kim and Iman Sadeghi**

Final Project

- Problem description is on-line
- Must be done in teams of two or three
- Application design description (min. 300 words) due Friday, November 19 at 4pm.
  Email to me: `jschulze@ucsd.edu`
- Project due to be presented on
  Friday, Dec 3rd, between 2 and 4pm
- No late submissions accepted

Final Project: Overview

- Score:
  - Proposal: 10 points
  - Presentation: 90 points
  - Web page: 10 points

- Implement at least 10 skill points of advanced rendering features per person
- Score will consist of two parts:
  - 70% for technical quality
  - 30% are for artistic quality
  - Grading will be based on presentation: your responsibility to show off technical features
- 60 seconds per team member for presentation

Final Project: Technical Features

- Per-pixel illumination with multiple light sources and different light source types such as spot lights; combination with texture mapping (7 skill points): enable/disable shader
- Bump mapping or displacement mapping (not both) (7 skill points): enable/disable
- Piecewise Bezier surfaces (7 skill points)
- Move the camera or objects in the scene along a path defined as a piecewise Bezier curve, for example to render a ride on a roller coaster (7 skill points)
- Any visual effect using CUDA code, e.g., effects based on physics simulation (10 skill points)
- Toon shading (7 skill points): enable/disable toon shader
- Environment mapping (10 skill points): enable/disable environment map
- Shadow mapping (10 skill points): toggle shadows on/off
- Shadow volumes (10 skill points): toggle shadows on/off
- Fractal terrain (10 skill points)
- Fractal plants or trees (10 skill points)
- Procedurally modeled city or other environment (10 skill points)
- Ambient occlusion (10 skill points): enable/disable the shader
- L-Systems (10 skill points)
- Collision detection (bounding boxes/spheres: 7 skill points; polygon level: 10 skill points): enable/disable mode in which colliding objects are highlighted
- Particle effect (7 skill points)

Final Project: Themes

- A museum room. Include artwork as texture maps. Include lights, benches, wood floors, rugs, doors, etc. if you wish. The user should be able to navigate the scene and change view direction with keyboard or mouse controls.
- Build a robot. Animate it. Make it walk or dance or respond to keyboard controls.
- A space ship flying over a planet modeled using fractal terrain.
- Build a simple car and a region (or a track) where it can be driven under user control.
- Build a virtual roller coaster. Let the user's viewpoint follow along (in or behind) the roller coaster car. Include some interesting scenery.
- Take photographs of a room or two from around UCSD or from your residence with a digital camera. Also take photographs of furniture. Make a three-dimensional model of this area and its furnishings, using your photographs as texture maps. Allow the user to navigate with arrows or other keyboard controls around the scene. (Sort of like Quake without the shooting.)
- Render a number of marbles bouncing around and bumping into each other, following the laws of physics, and casting shadows on each other and the surface they are on.

Final Project: Extra Credit

- In order to receive extra credit, you need to create a web page about your project.
- At a minimum it needs to include a description of your project (min. 300 words), and at least one screen shot.
- Explain plot, game play, supported keyboard and mouse commands, artistic efforts
- Web page has to be emailed to the instructor by **Thursday, December 2nd at 11:59pm**.
- Send content of your web page in one of two ways:
  - Description as ASCII text in email and attach your screen shot(s) as image files (JPEG, TIFF, GIF, PNG)
  - ZIP/tar up your web page directory tree including your HTML file(s) and your images

StarCAVE Tour

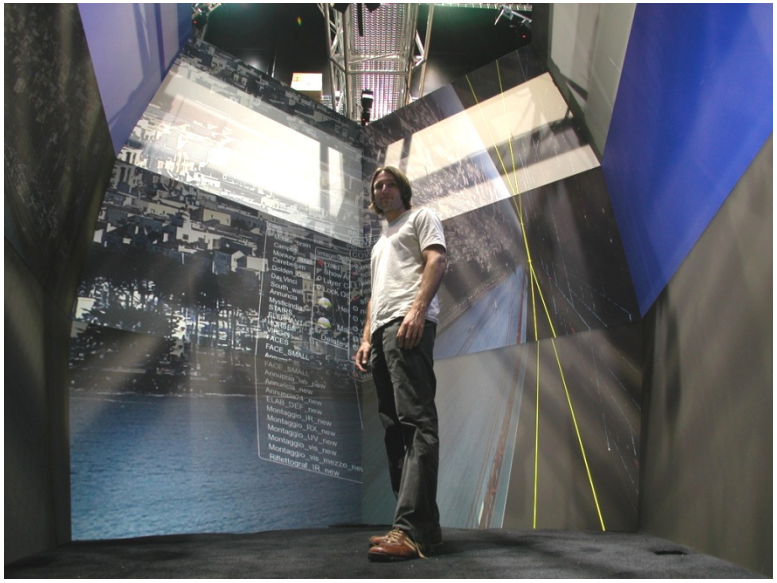
- Location: Atkinson Hall, 1st floor
- 18 Dell XPS PCs with Quad Core Intel CPUs
- CentOS 5.3 Linux
- Dual Nvidia Quadro 5600 graphics cards per node
- 34 JVC HD2k projectors (1920x1080 pixels): ~34 megapixels per eye
- Passive stereo with circular polarization filters
- 15 screens, ~8 x 4 feet each
- Floor projection
- Optical, wireless tracking system
- Software: COVISE
- Programming Language: C++

Tour Date:

- Friday, Nov 19, 4:00-5:00pm

Location:

Immersive Visualization Laboratory
1st floor Atkinson Hall
Turn right at main entrance

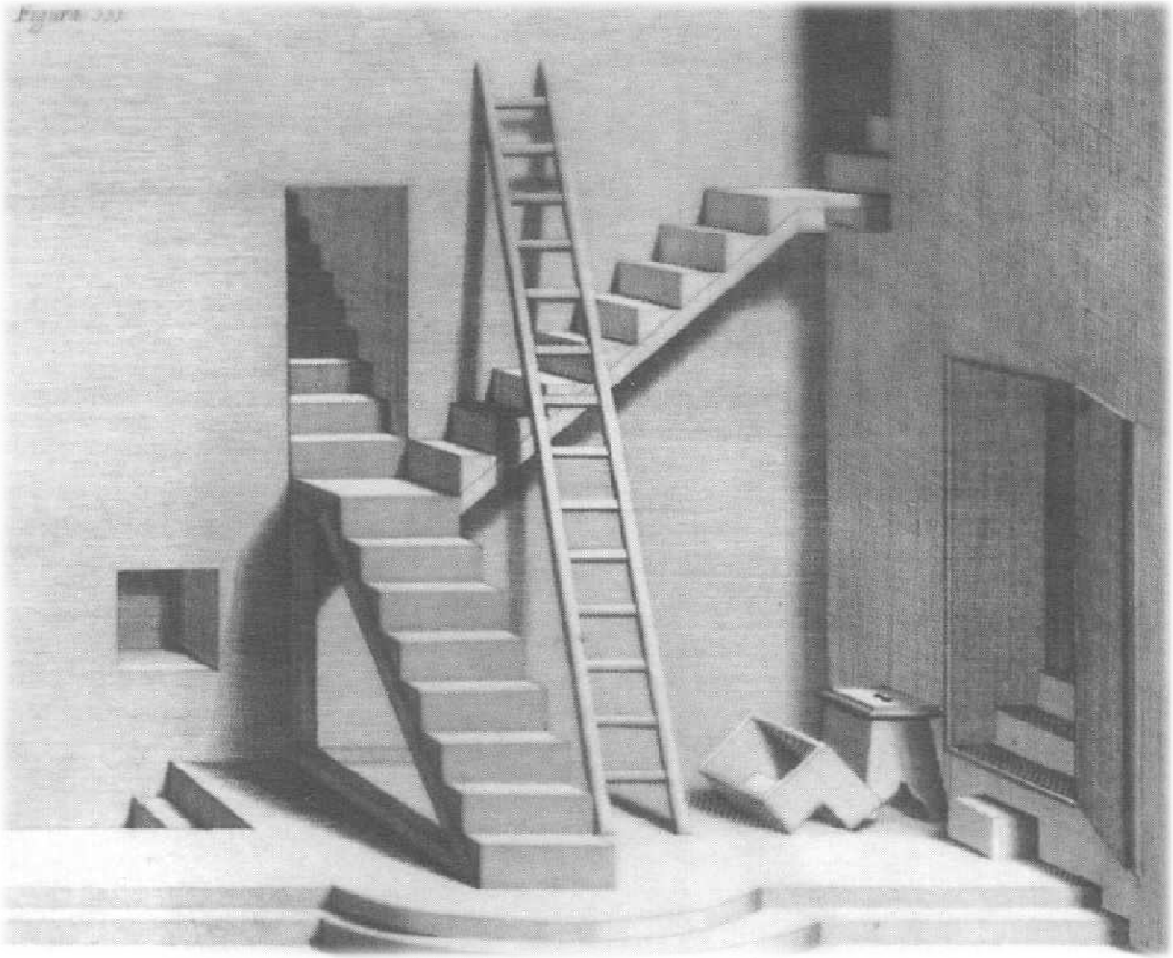


Lecture Overview

- Shadows
- Shadow mapping

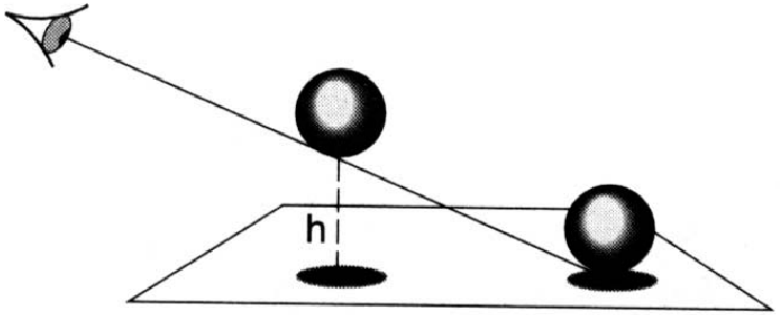
Why Are Shadows Important?

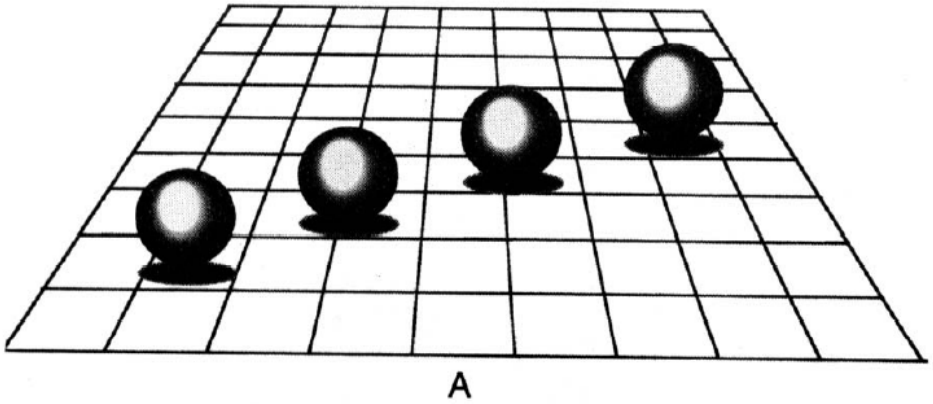
- Give additional cues on scene lighting

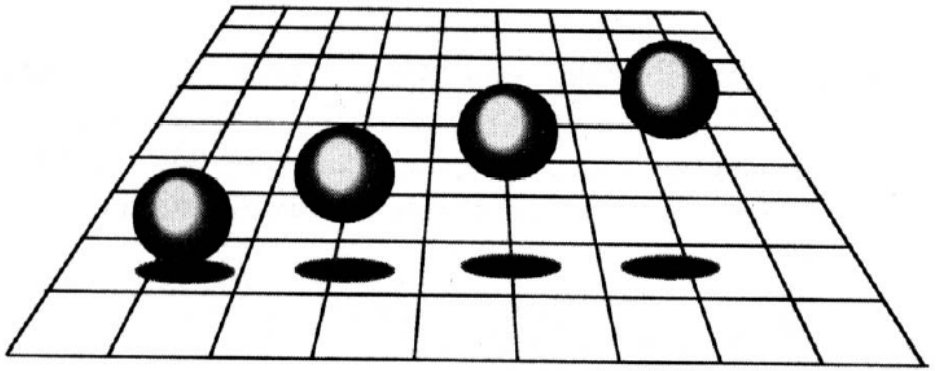


Why Are Shadows Important?

- Contact points
- Depth cues

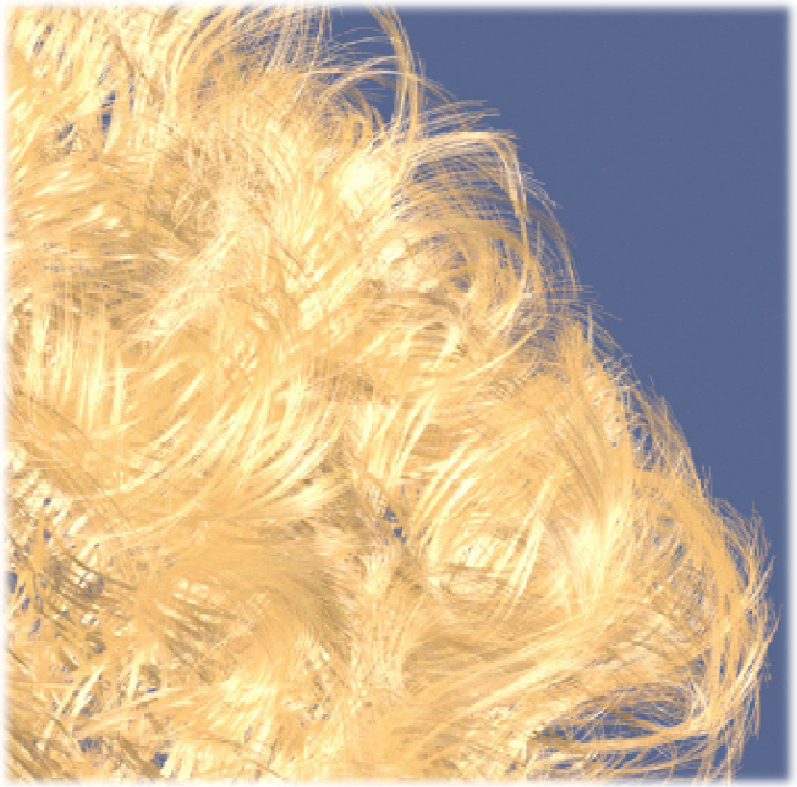






Why Are Shadows Important?

- Realism

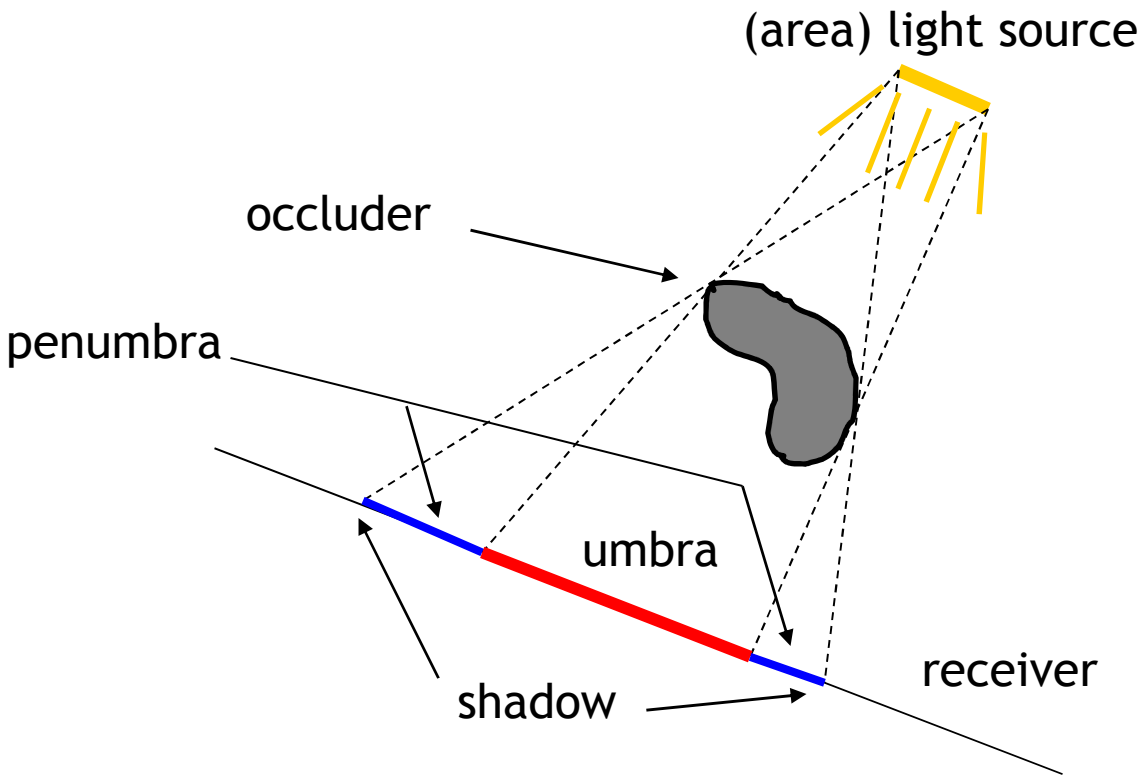


Without self-shadowing

With self-shadowing

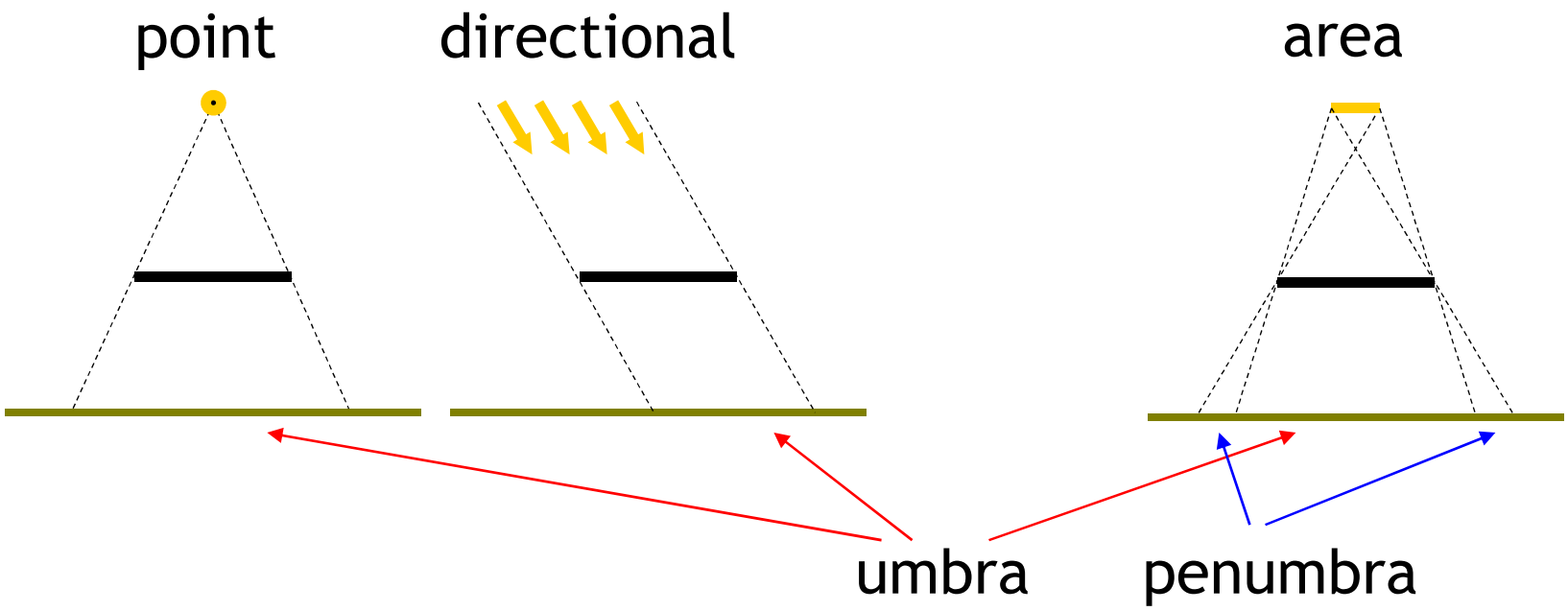
Terminology

- 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

- Here: focus on hard shadows
  - Soft shadows often too hard to compute in interactive graphics
- Two most popular techniques:
  - Shadow mapping
  - Shadow volumes
- Many variations, subtleties
- Active research area

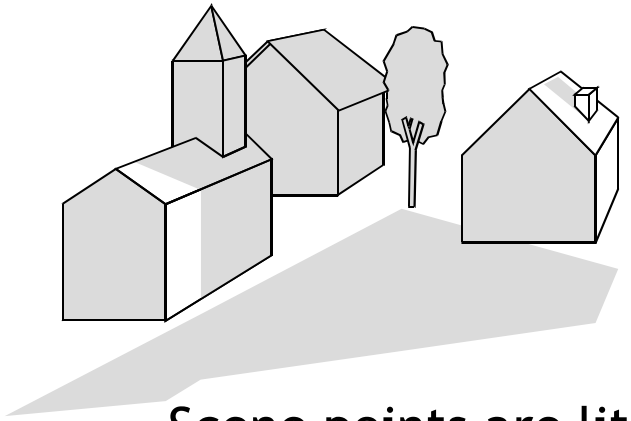
Lecture Overview

- Shadows
- Shadow mapping

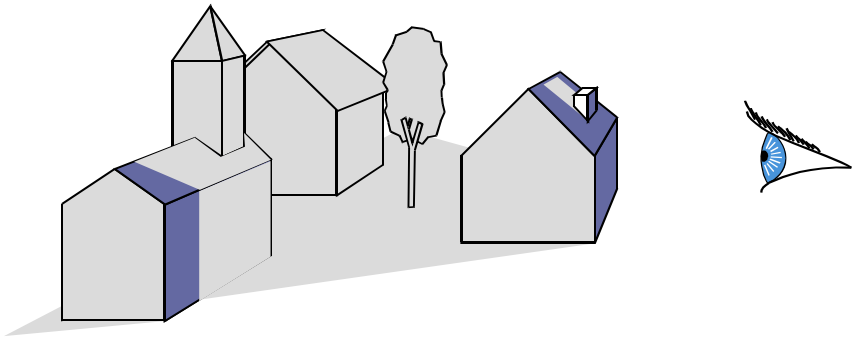
Shadow Mapping

Main Idea

- A scene point is lit by the light source if visible from the light source
- Determine visibility from light source by placing a camera at the light source position and rendering the scene from there



Scene points are lit if visible from light source

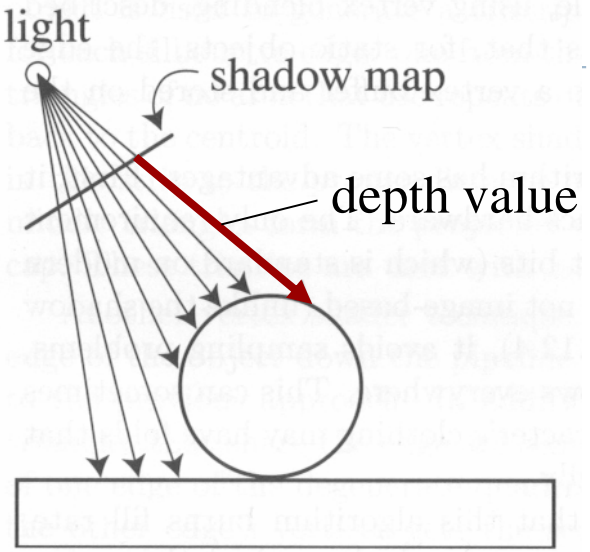


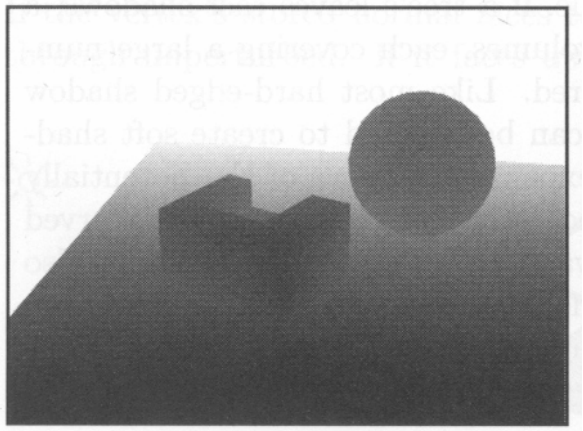
Determine visibility from light source by placing camera at light source position

Two Pass Algorithm

First Pass

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



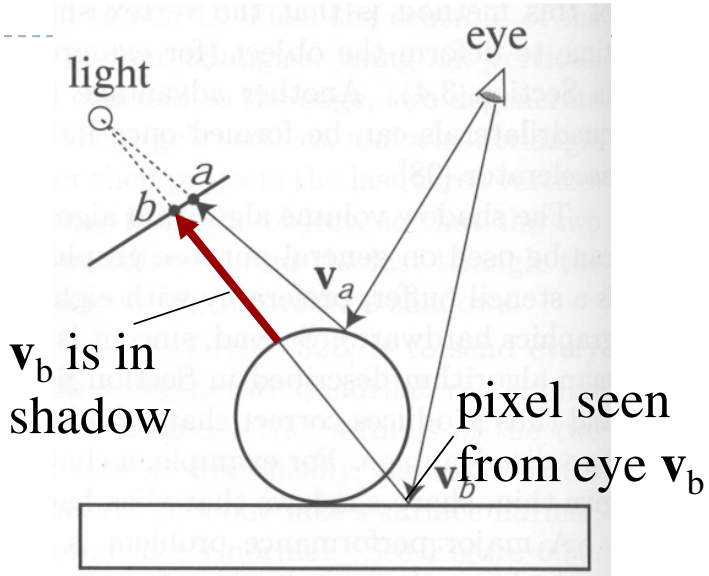


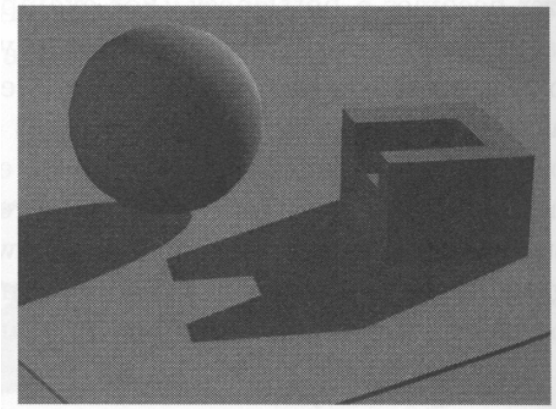
Depth image as seen from light source

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, pixel is in shadow
  - If distance is smaller or equal, pixel is lit





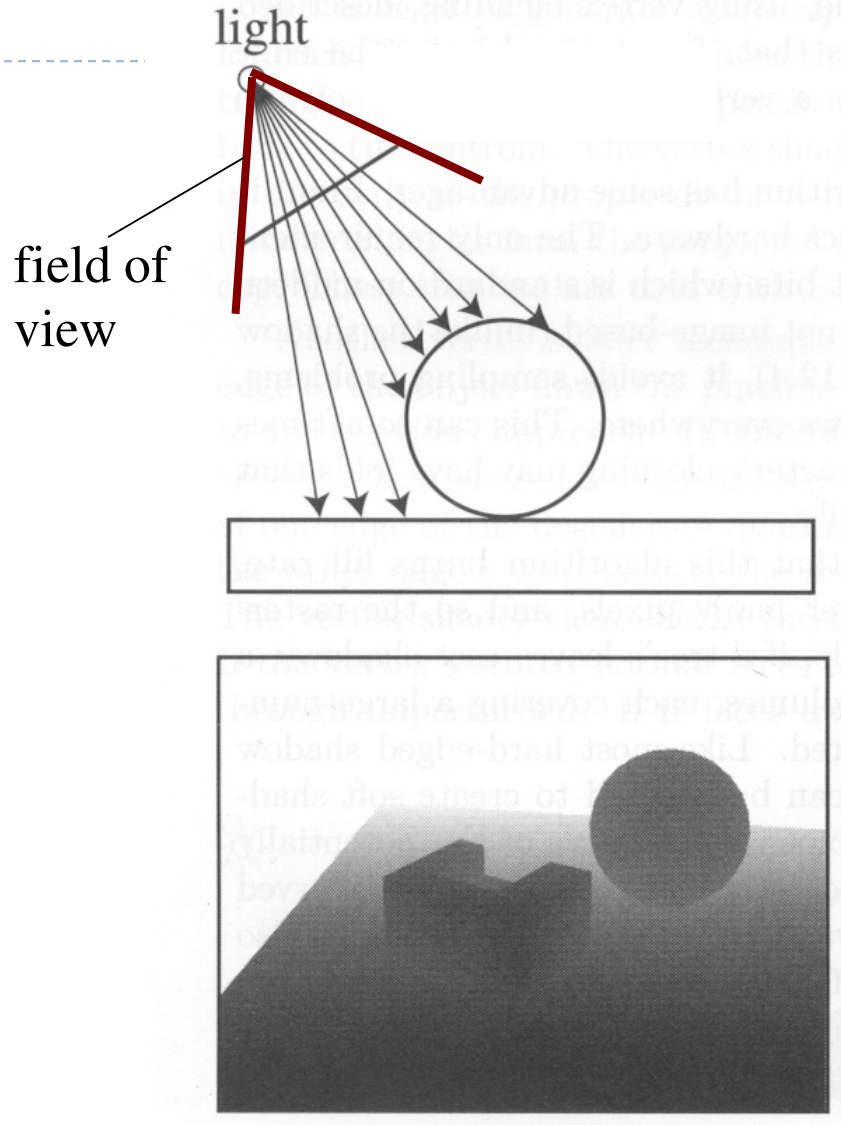
Final image with shadows

Issues With Shadow Maps

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

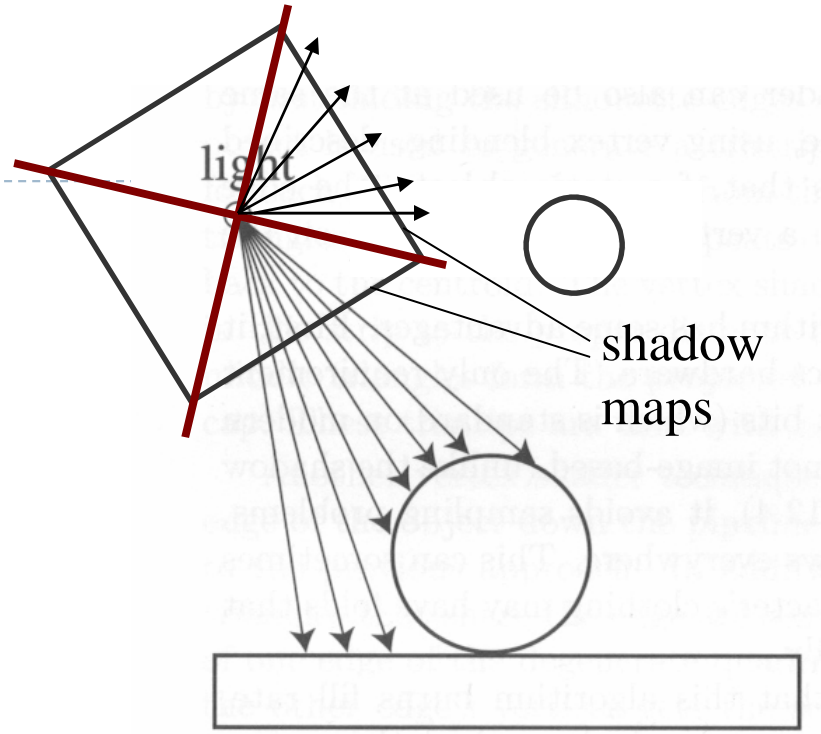
Limited Field of View

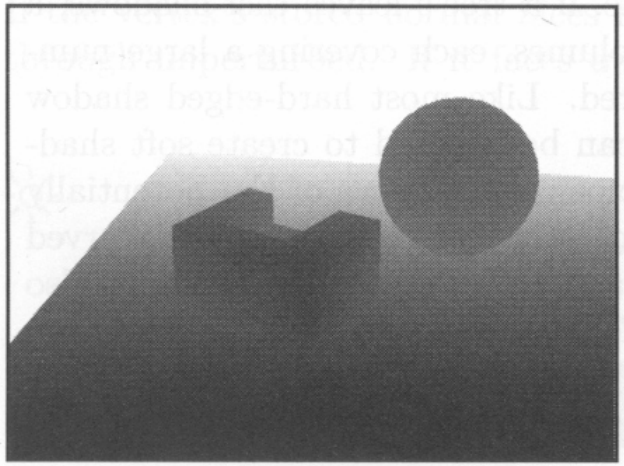
- What if a scene point is outside the field of view of the shadow map?



Limited Field of View

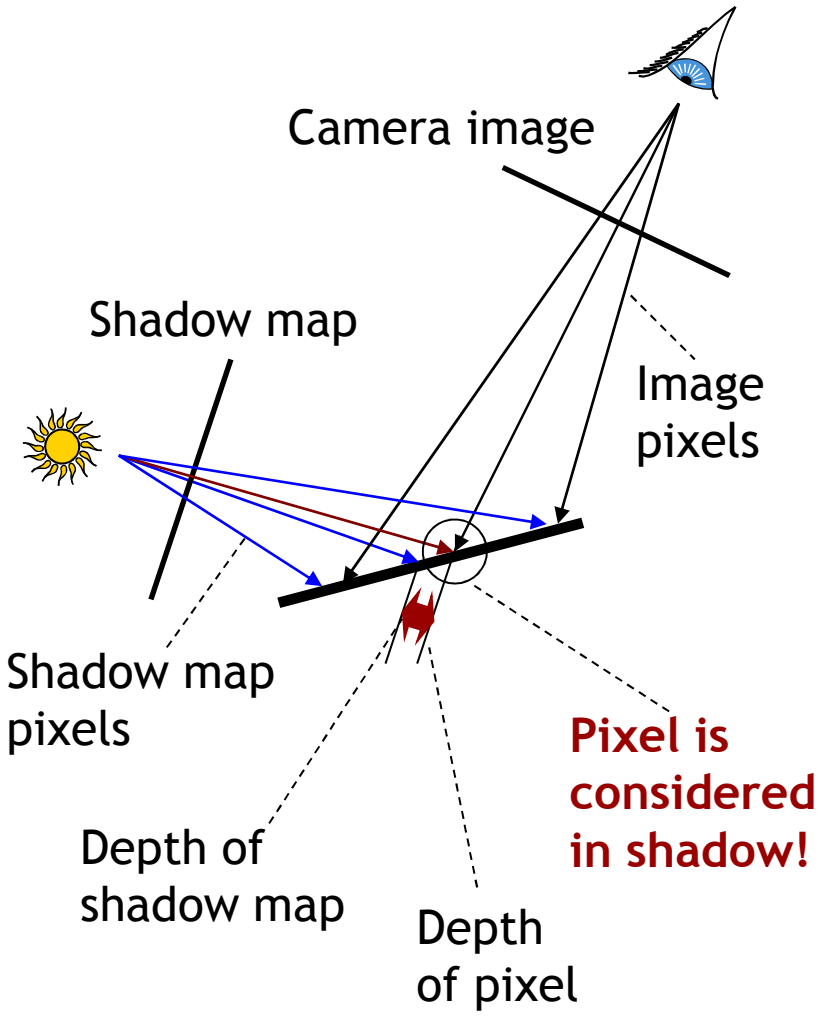
- What if a scene point is outside the field of view of the shadow map?
  → Use six shadow maps, arranged in a cube
- Requires a rendering pass for each shadow map!





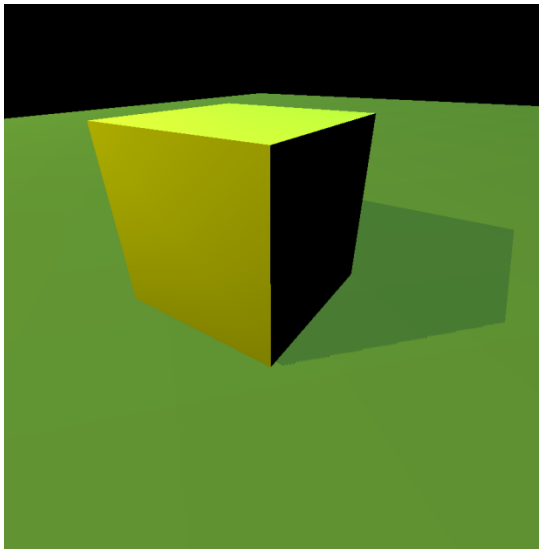
Z-Fighting

- 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

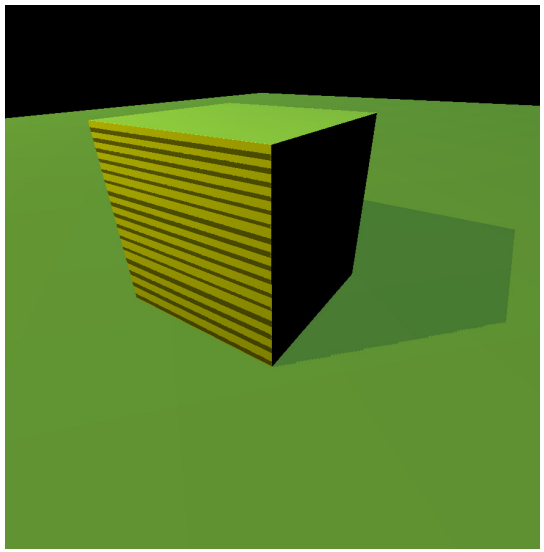


Solution: Bias

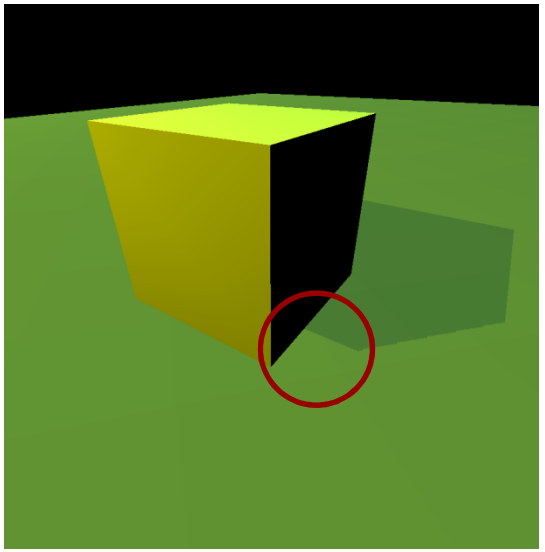
- Add bias when rendering shadow map
  - Move geometry away from light by small amount
- Finding correct amount of bias is tricky



Correct bias



Not enough bias

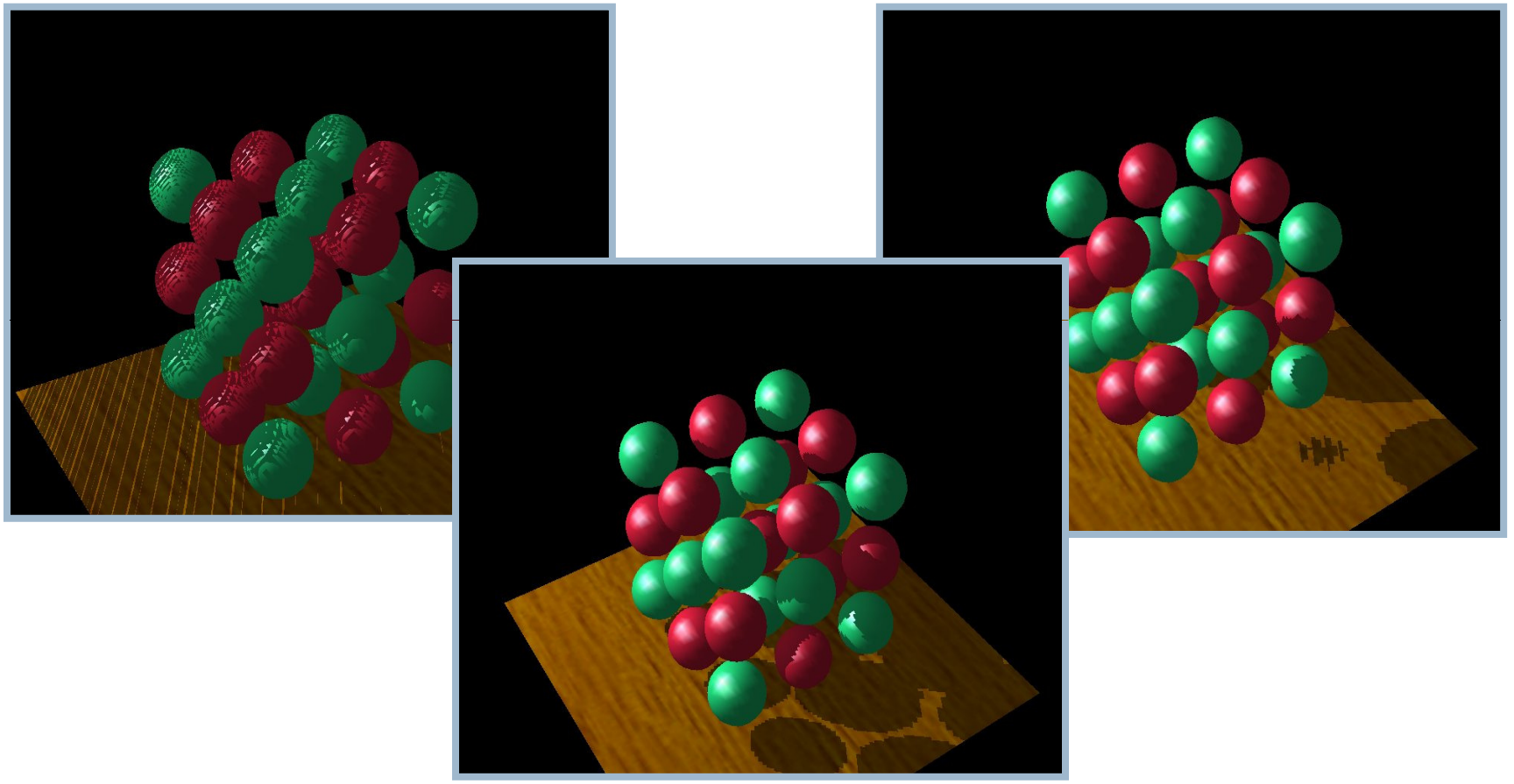


Too much bias

Bias Adjustment

Not enough

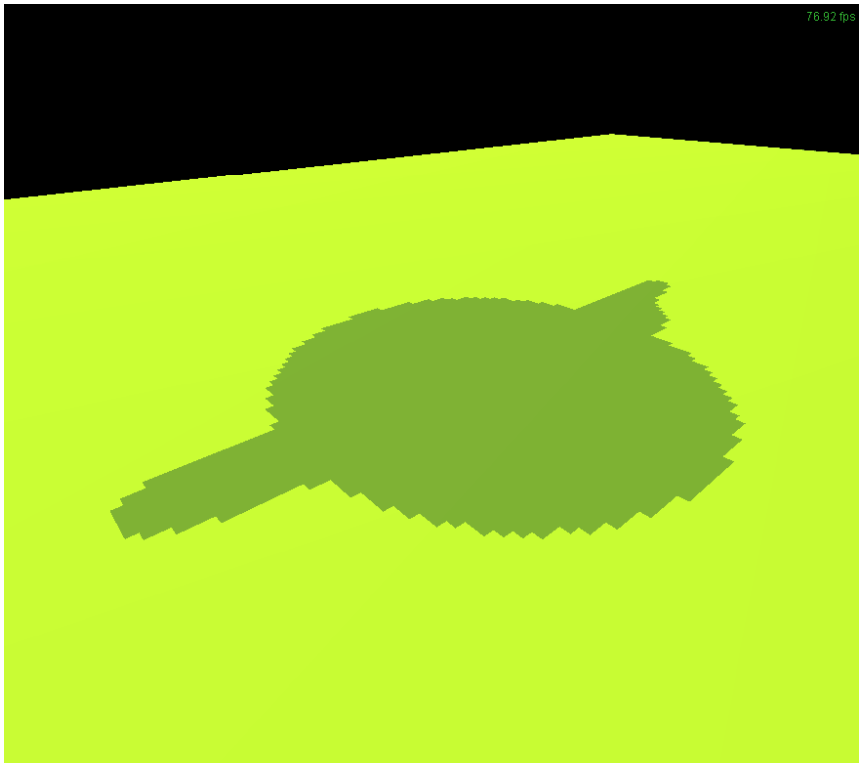
Too much



Just right

Sampling Problems

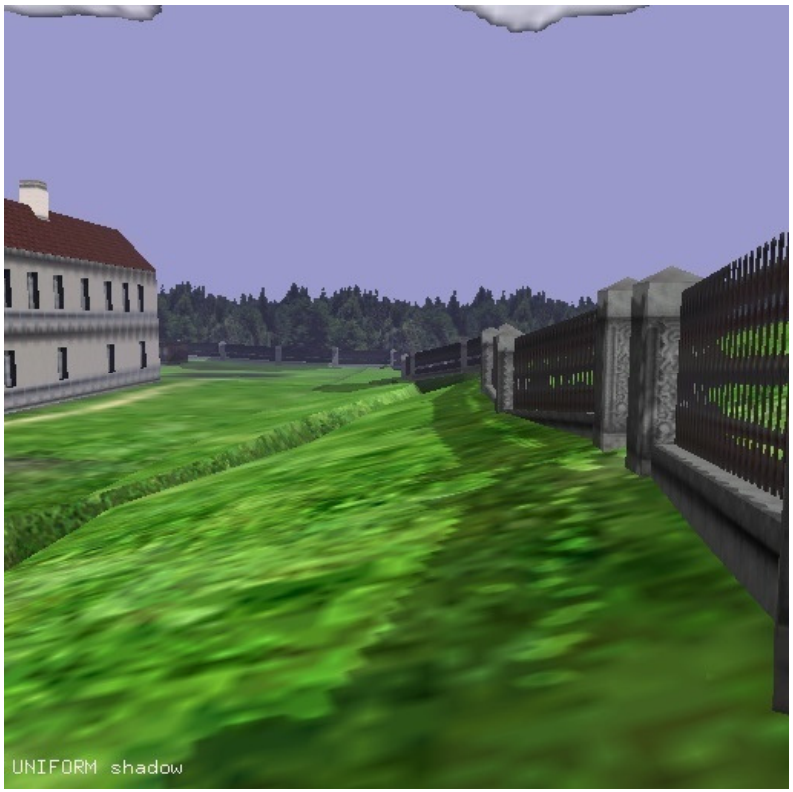
- Shadow map pixel may project to many image pixels
  → Stair-stepping artifacts



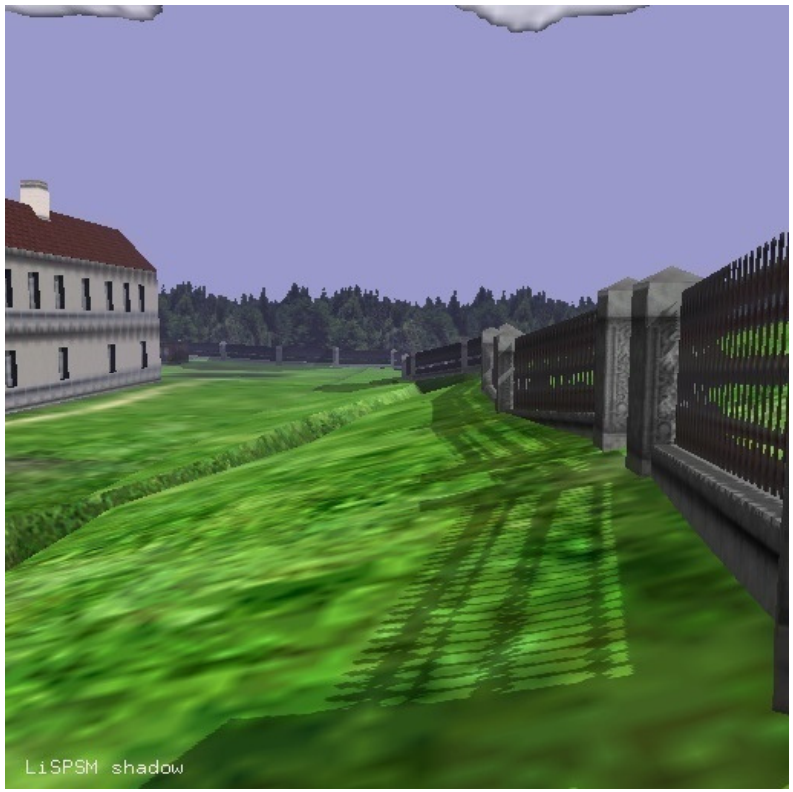
Solutions

- 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/
  - Includes GLSL source code
- Combination of splitting and LiSPSM
  - Basis for most serious implementations

LiSPSM



Basic shadow map



Light space perspective shadow map

Video

- Video URL:
  http://www.cg.tuwien.ac.at/research/vr/lispsm/videos/shadows_egsr2004.mpg

Shadow Mapping With GLSL

First Pass

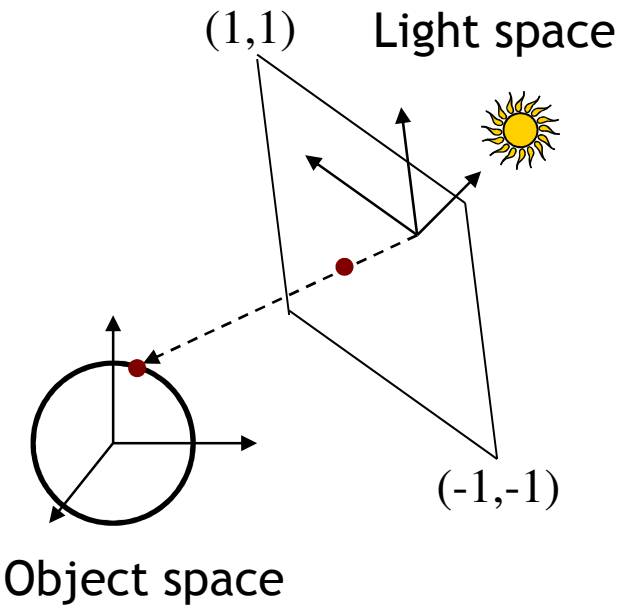
- Render scene by placing camera at light source position
- Compute light view (look at) matrix
  - Similar to computing camera matrix from look-at, up vector
  - Compute its inverse to get world-to-light transform
- Determine view frustum such that scene is completely enclosed
  - Use several view frusta/shadow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

First Pass

- Each vertex point is transformed by

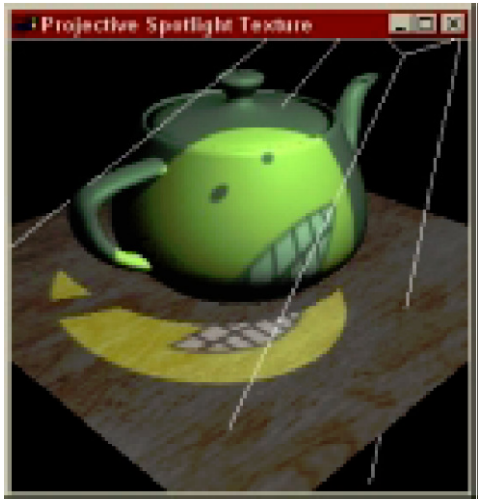
$$\mathbf{P}_{light}\mathbf{V}_{light}\mathbf{M}$$

  - 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 $[-1,1]^3$



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

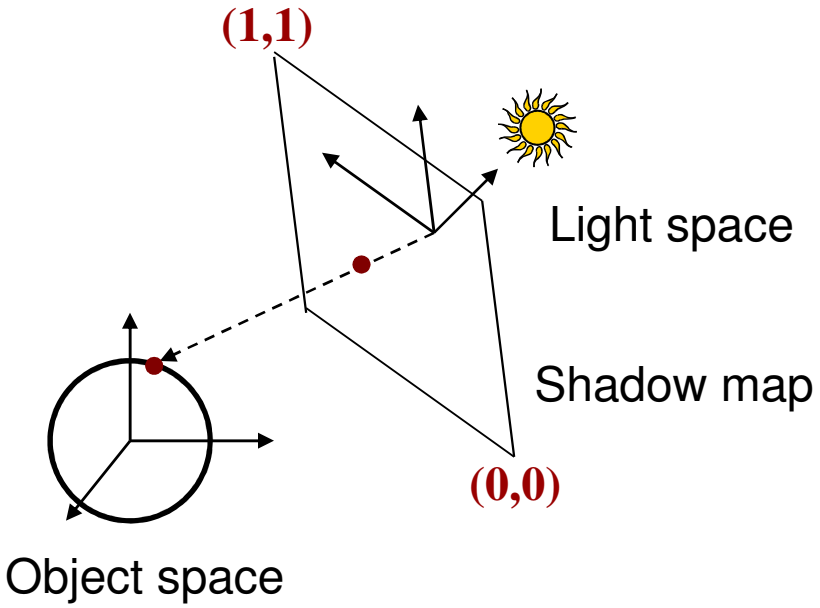
- Render scene from camera
- At each pixel, look up corresponding location in shadow map
- Compare depths with respect to light source

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

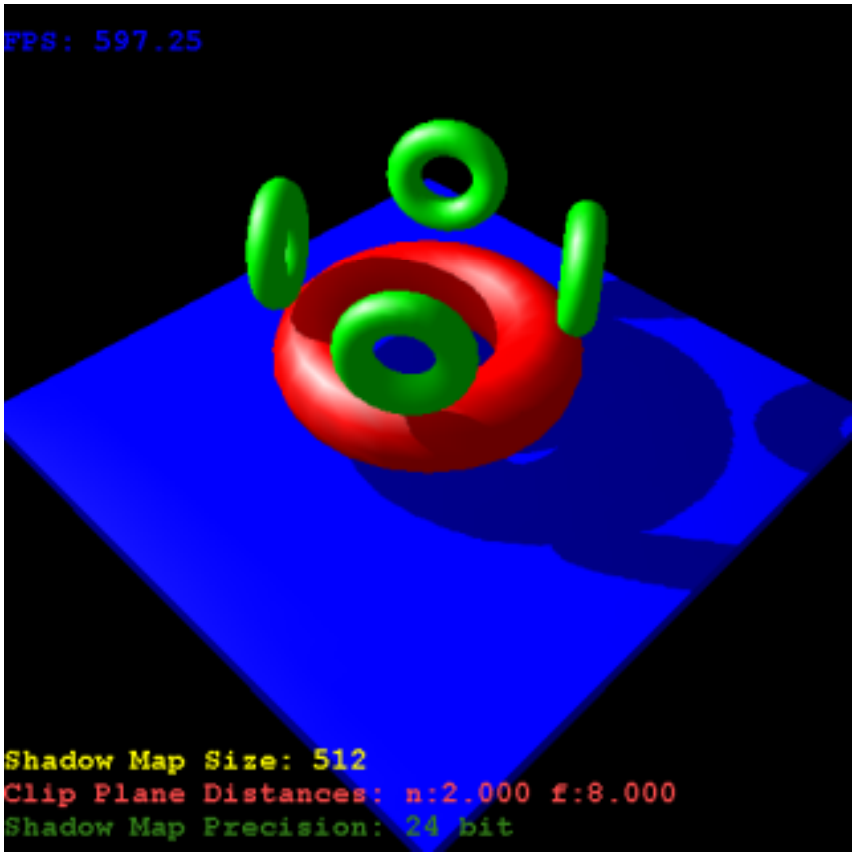
- Declare shadow map as sampler2DShadow
- Look up shadow map using projective texturing with `vec4 texture2DProj(sampler2D, vec4)`

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



Next Lecture

- Procedural Modeling