CSE 167: Introduction to Computer Graphics Lecture #17: Shadow Mapping

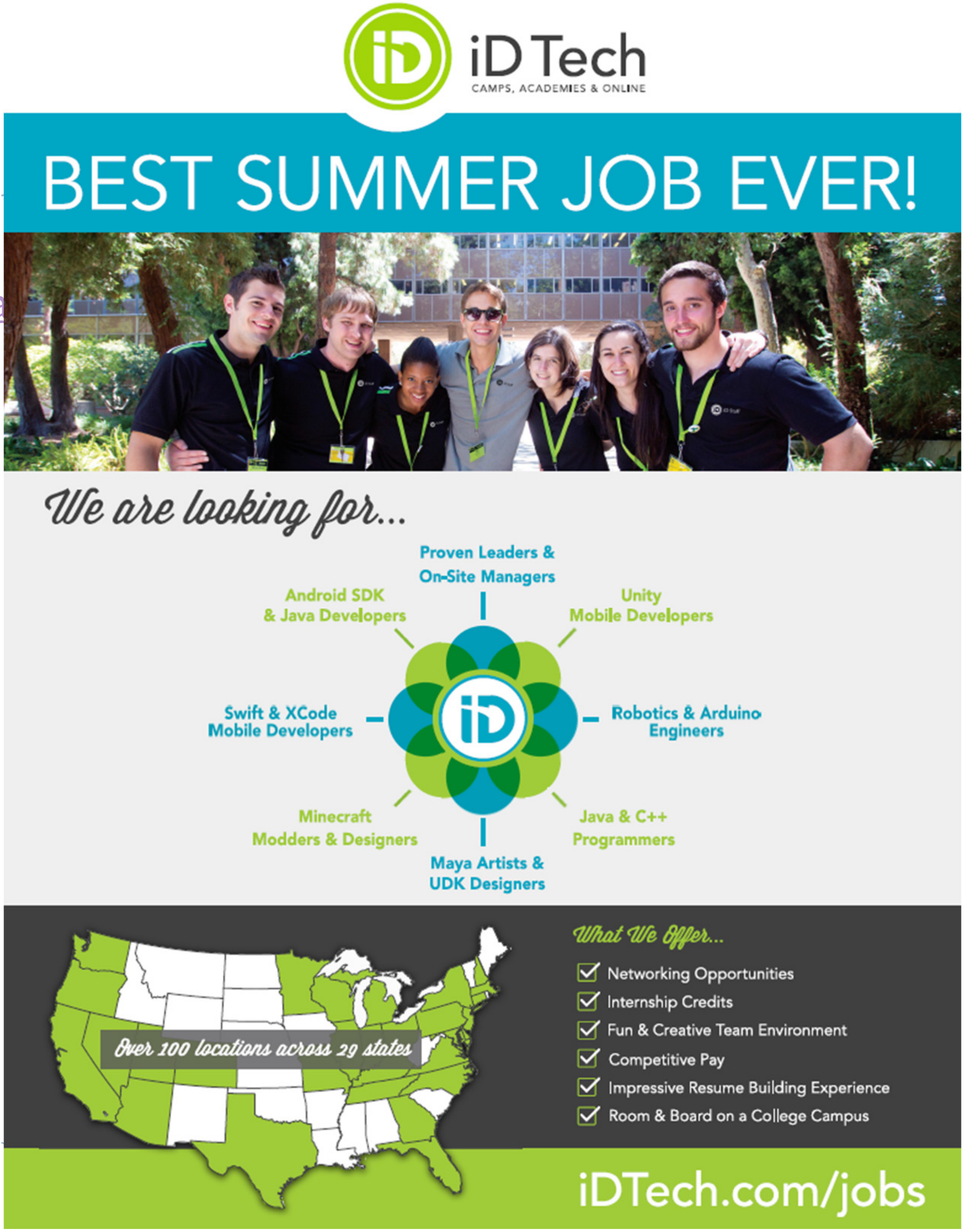
Jürgen P. Schulze, Ph.D.
University of California, San Diego

Fall Quarter 2015

Announcements

- TA evaluations
- CAPE
- 2nd blog entry due tonight at midnight
- 3rd blog entry due next Tuesday evening
- Final project presentations next Thursday 8am-11am in CSE 1202
- Winter:
  - CSE 190 Advanced Computer Graphics with Prof. Ramamoorthi
  - CSE 165 3D User Interfaces
- Independent research (CSE 199) projects in my lab: apply now

- ID Tech camp summer jobs: https://www.youtube.com/wa



iD Tech
CAMPS, ACADEMIES & ONLINE

BEST SUMMER JOB EVER!

We are looking for...

Proven Leaders & On-Site Managers

Android SDK & Java Developers

Unity Mobile Developers

Swift & XCode Mobile Developers

Robotics & Arduino Engineers

Minecraft Modders & Designers

Java & C++ Programmers

Maya Artists & UDK Designers

Over 100 locations across 29 states

What We Offer...

- ☑ Networking Opportunities
- ☑ Internship Credits
- ☑ Fun & Creative Team Environment
- ☑ Competitive Pay
- ☑ Impressive Resume Building Experience
- ☑ Room & Board on a College Campus

iDTech.com/jobs

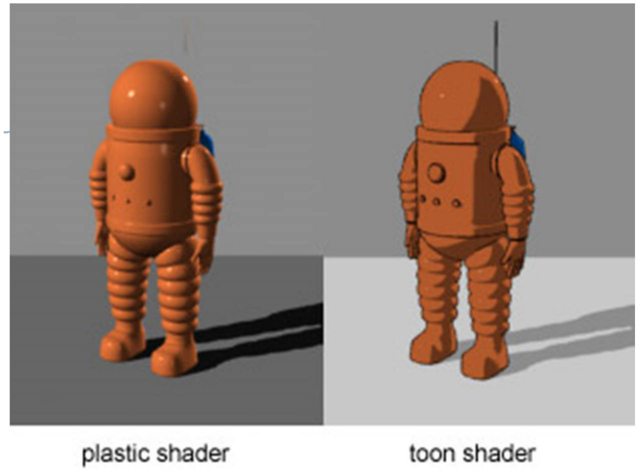
Lecture Overview

Advanced Shader Effects

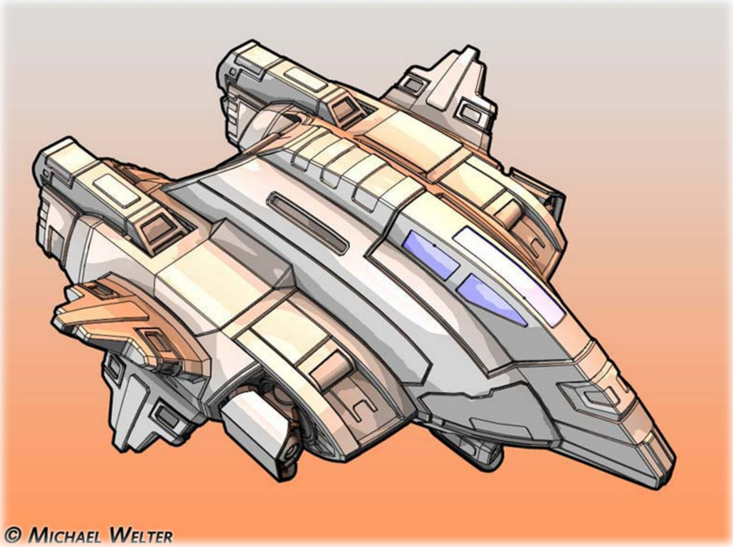
- Toon shading
- Shadow Mapping

Toon Shading

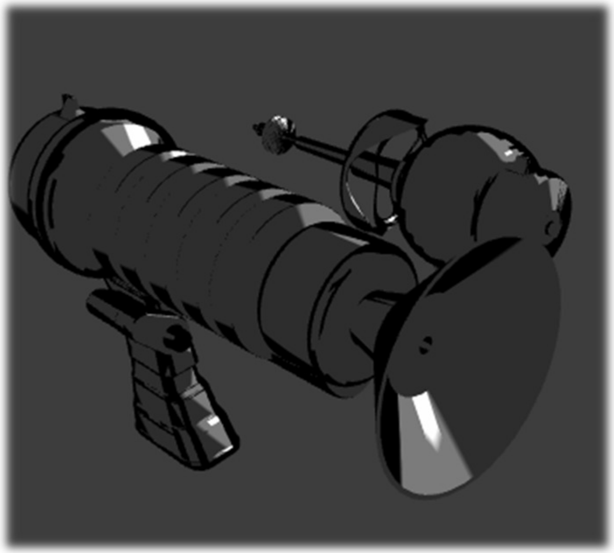
- A.k.a. Cel Shading ("Cel" is short for "celluloid" sheets, on which animation was hand-drawn)
- Gives any 3D model a cartoon-style look
- Emphasizes silhouettes
- Discrete steps for diffuse shading, highlights
- Non-photorealistic rendering method (NPR)
- Programmable shaders allow real-time performance



Source: Wikipedia



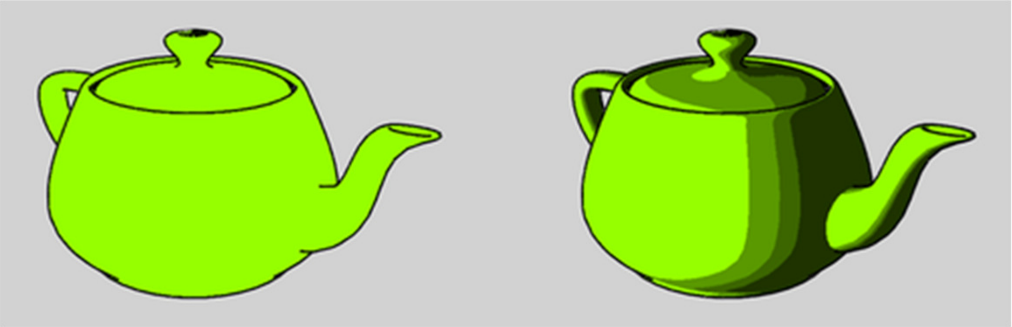
Off-line toon shader



GLSL toon shader

Approach

- Start with regular 3D model
- Apply two rendering tricks:
  - Silhouette edges
    - Emphasize pixels with normals perpendicular to viewing direction.
  - Discretized shading
    - Conventional (smooth) lighting values calculated for each pixel, then mapped to a small number of discrete shades.

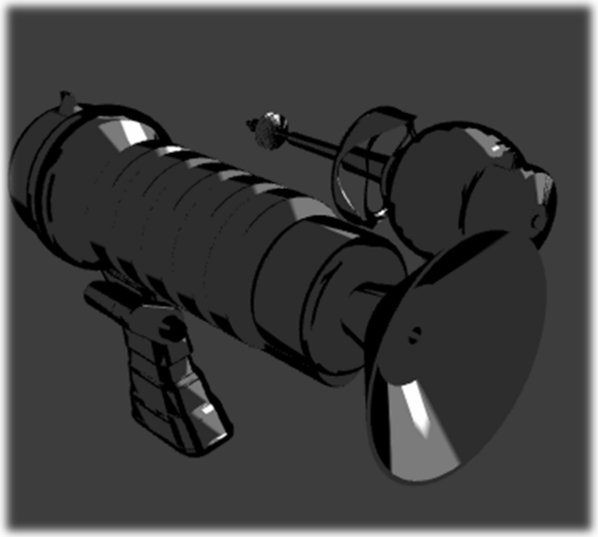


Source: Wikipedia

Silhouette Edges

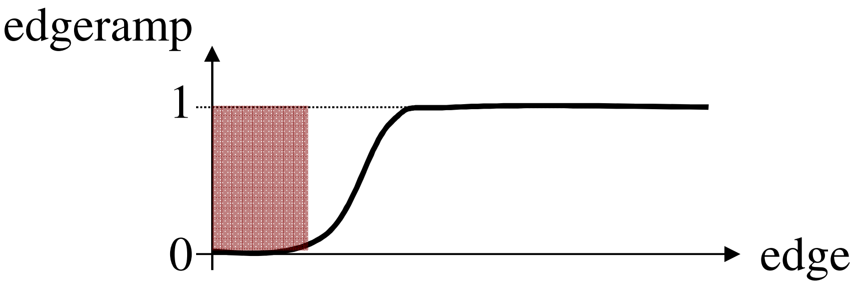
- Silhouette edge detection
  - Compute dot product of viewing direction $\mathbf{v}$ and normal $\mathbf{n}$

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



  - Use 1D texture to define edge ramp

    uniform sample1D edgeramp; e=texture1D(edgeramp,edge);



Discretized Shading

- Compute diffuse and specular shading

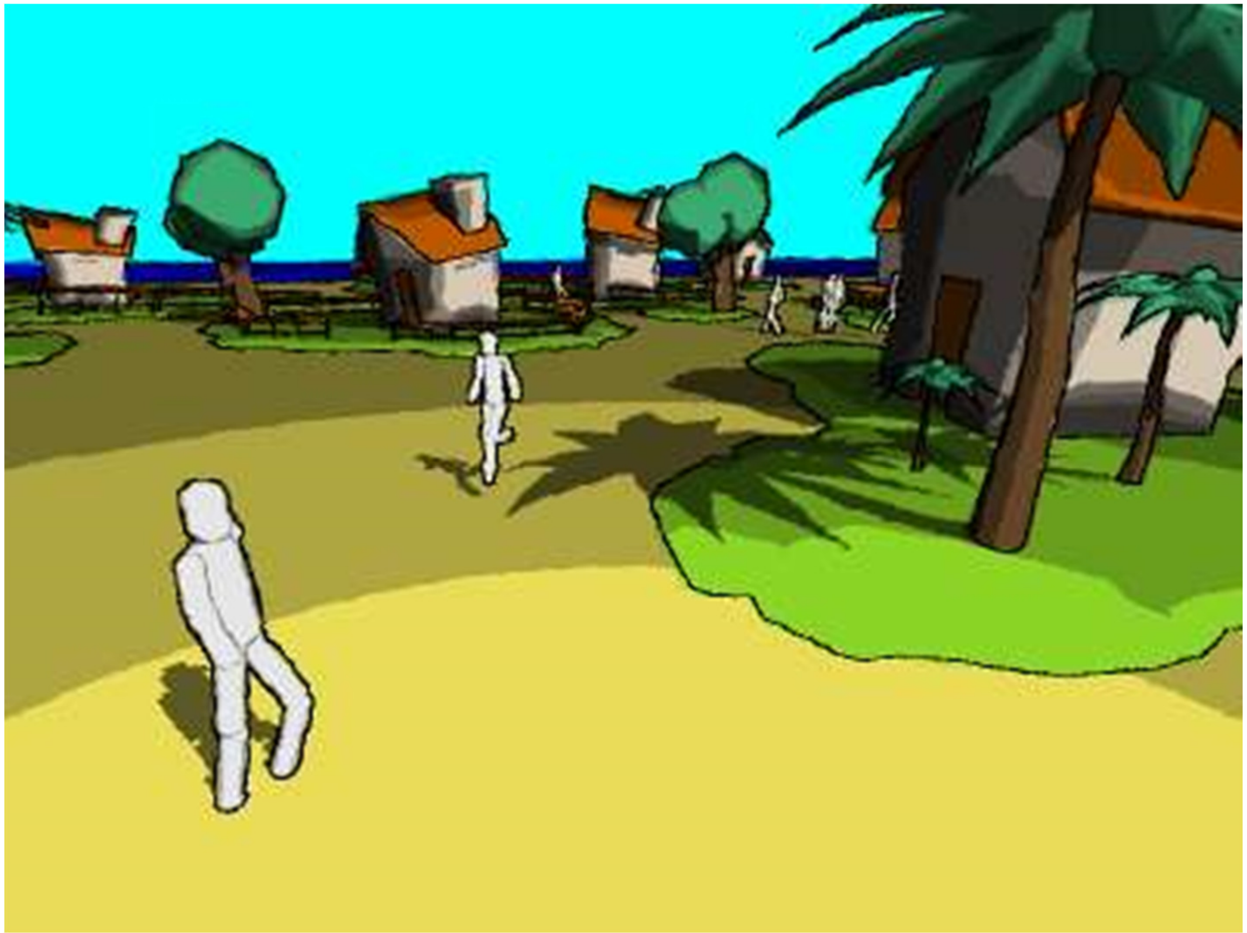
$$\text{diffuse} = \mathbf{n} \cdot \mathbf{L} \qquad \text{specular} = (\mathbf{n} \cdot \mathbf{h})^s$$

- Use 1D textures diffuseramp, specularramp to map diffuse and specular shading to colors
- Final color:

```
uniform sampler1D diffuseramp;
uniform sampler1D specularramp;
c = e * (texture1D(diffuse,diffuseramp) +

texture1D(specular,specularramp));
```

Toon Shading Demo



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

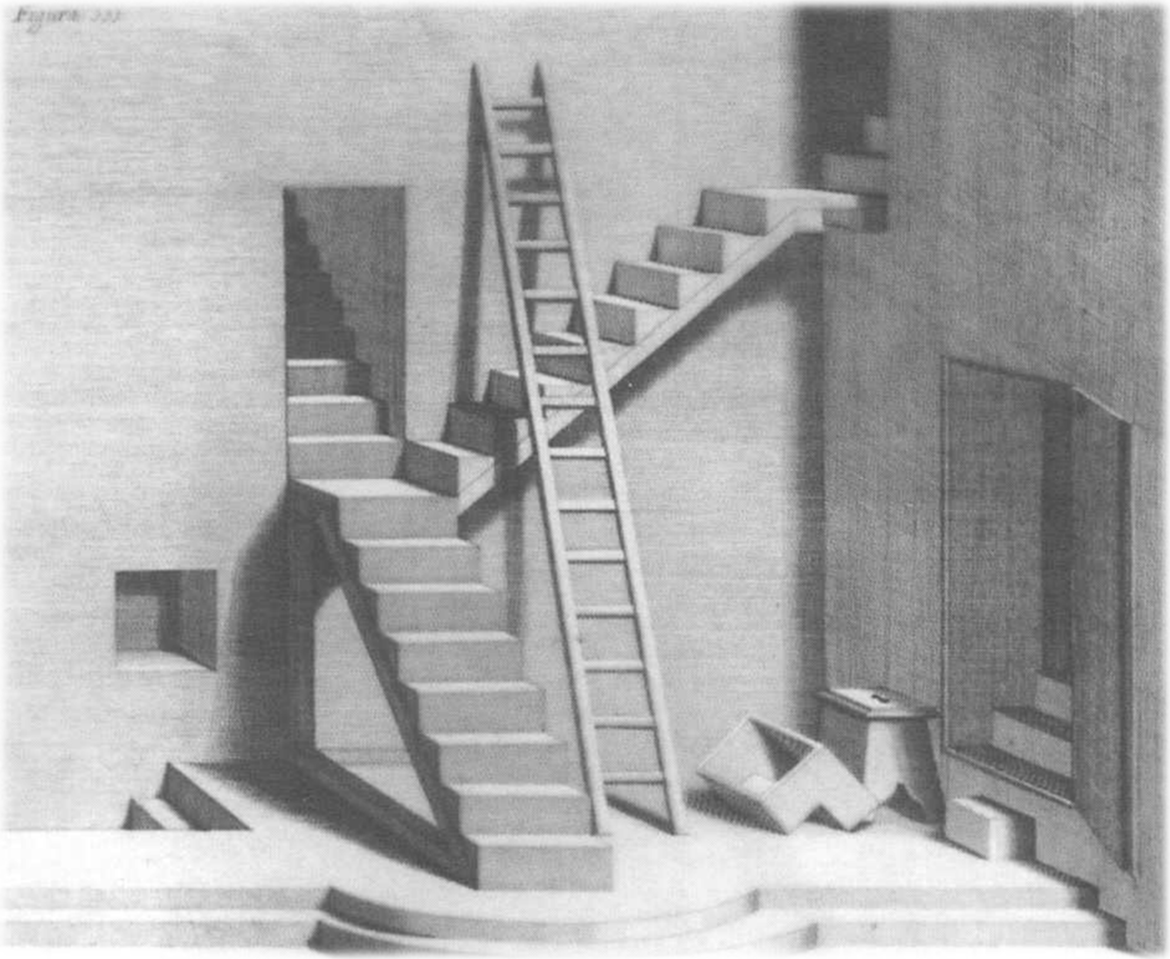
Lecture Overview

Advanced Shader Effects

- Toon shading
- Shadow Mapping

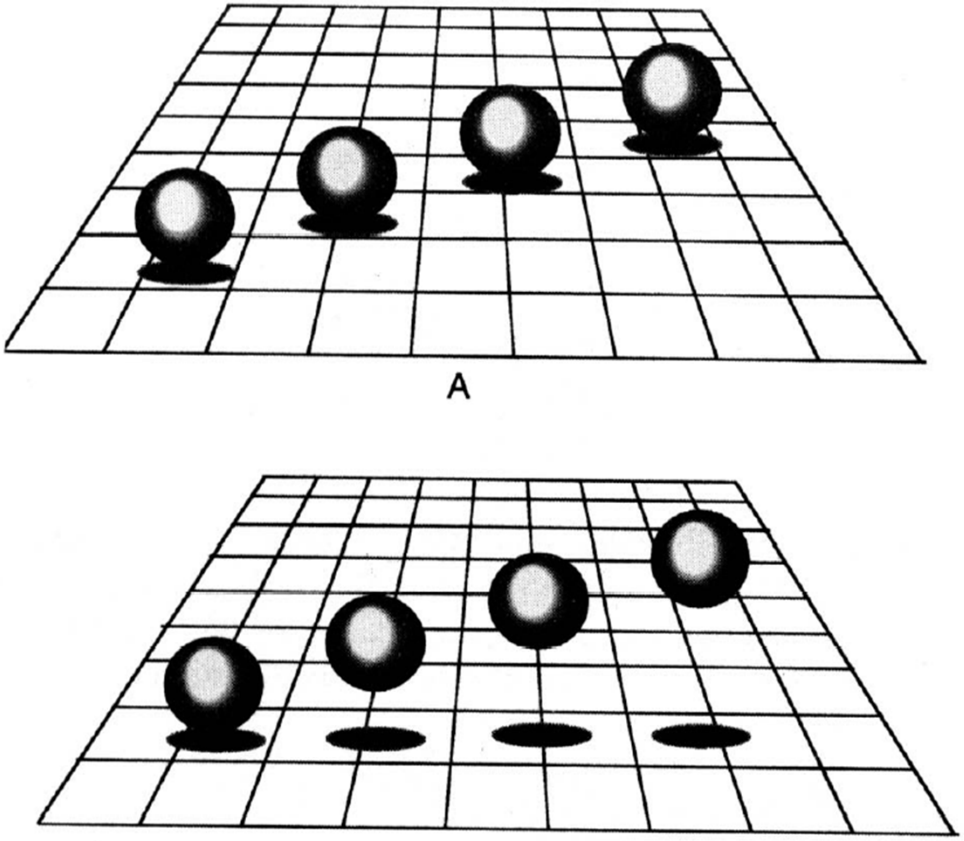
Why Are Shadows Important?

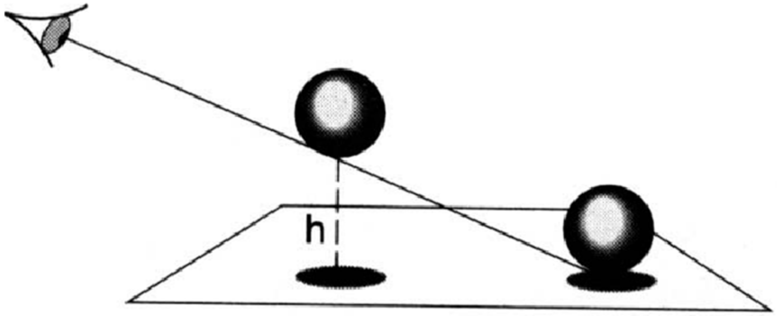
- Give additional cues on scene lighting



Why Are Shadows Important?

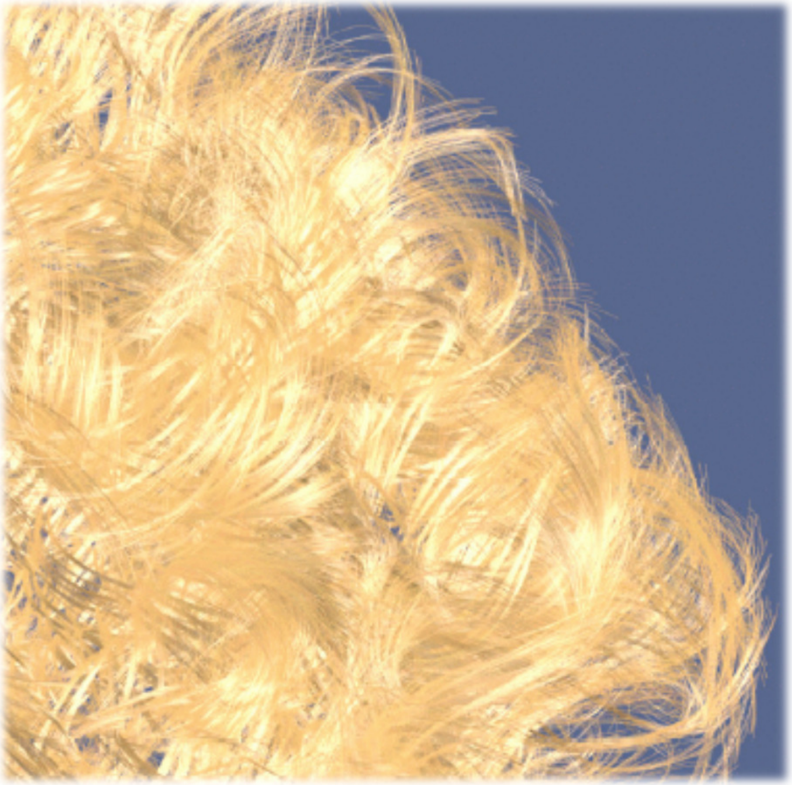
- Contact points
- Depth cues



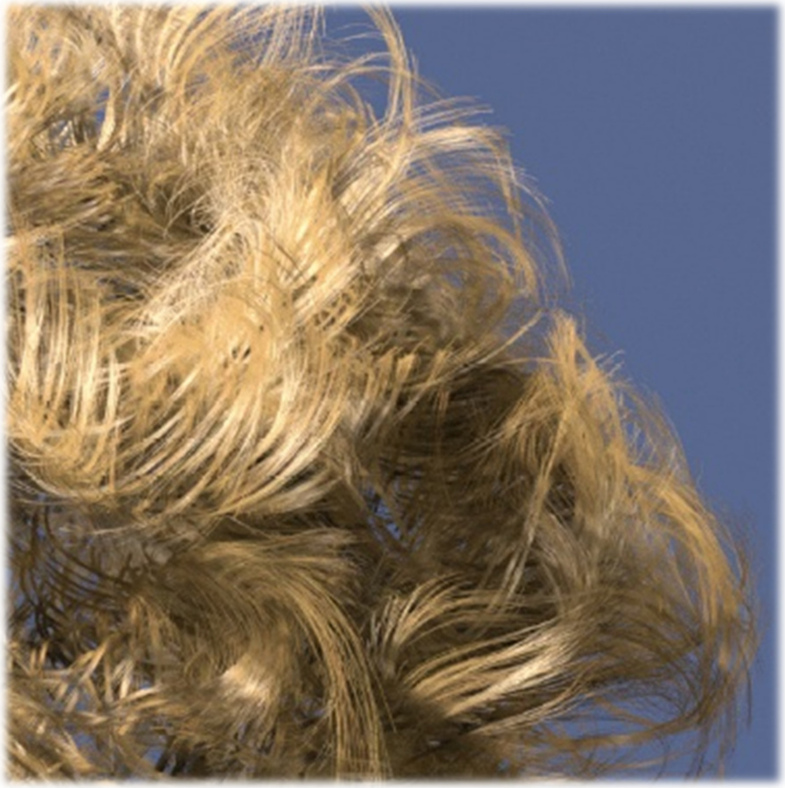


Why Are Shadows Important?

- Realism



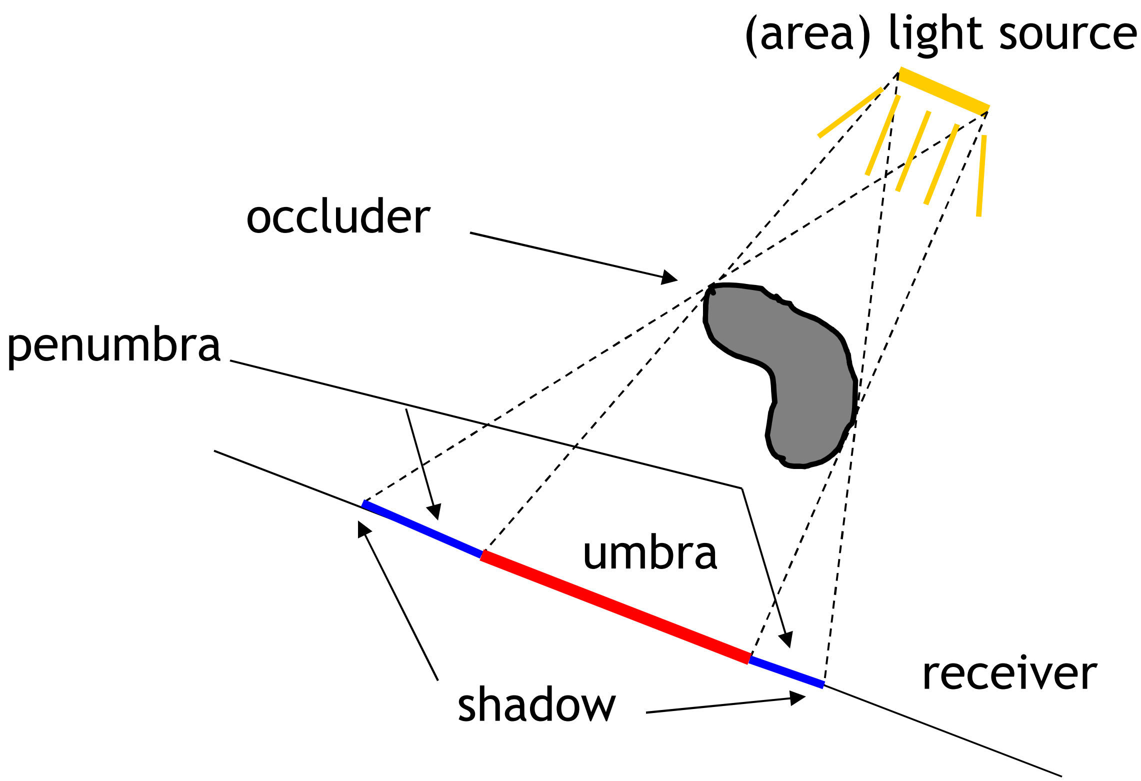
Without self-shadowing



With self-shadowing

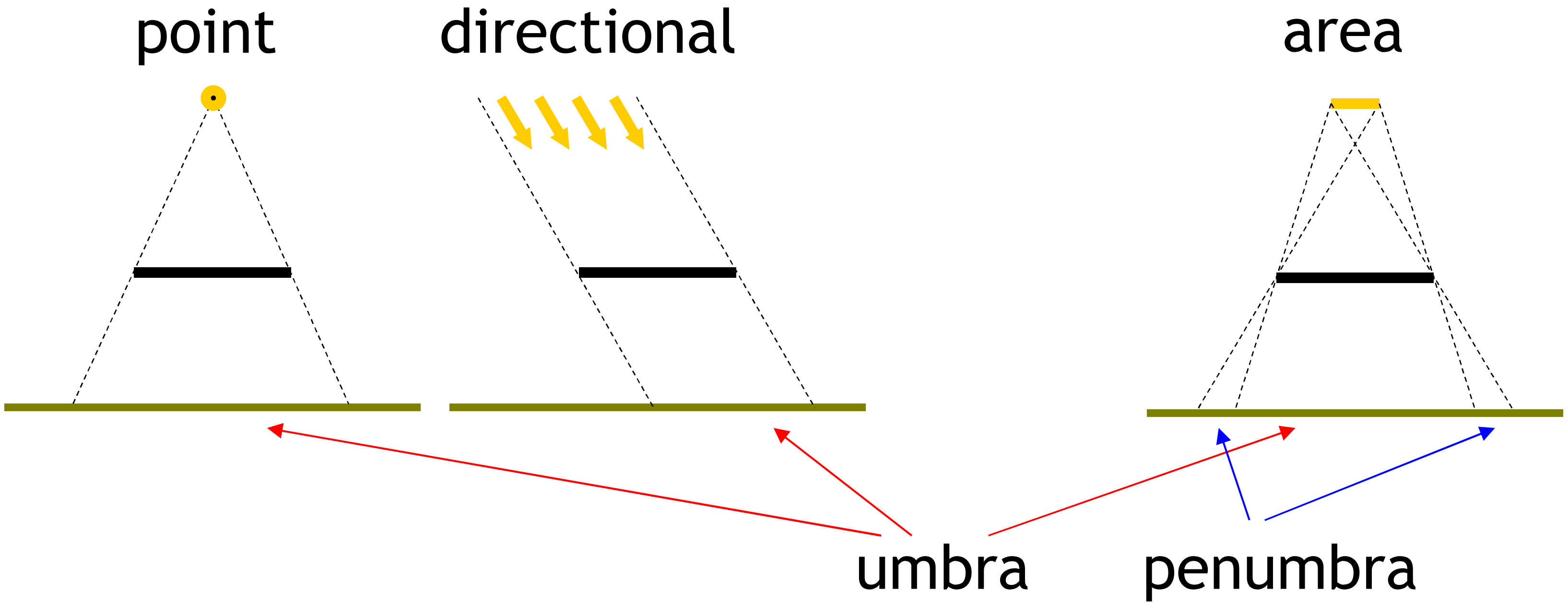
Terminology

- Umbra: fully shadowed region
- Penumbra: partially shadowed region

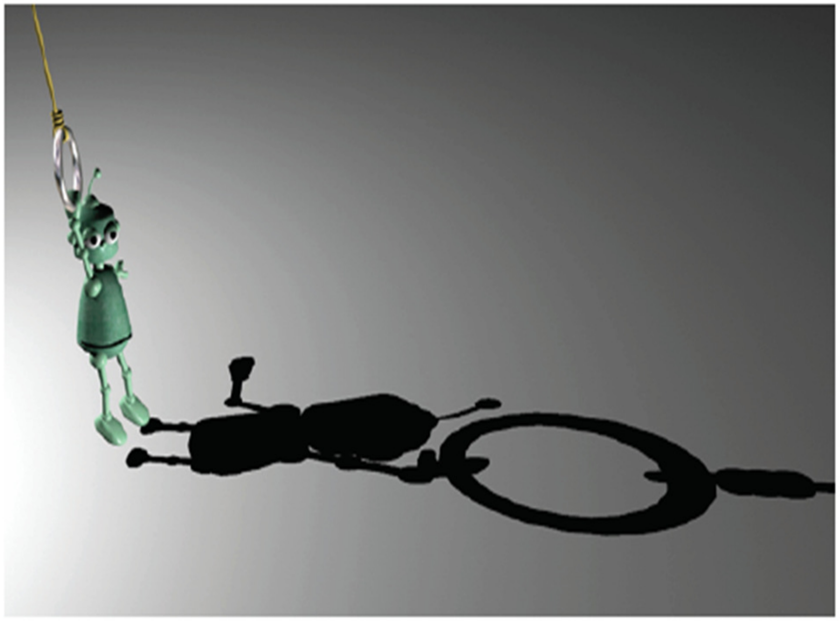


Hard and Soft Shadows

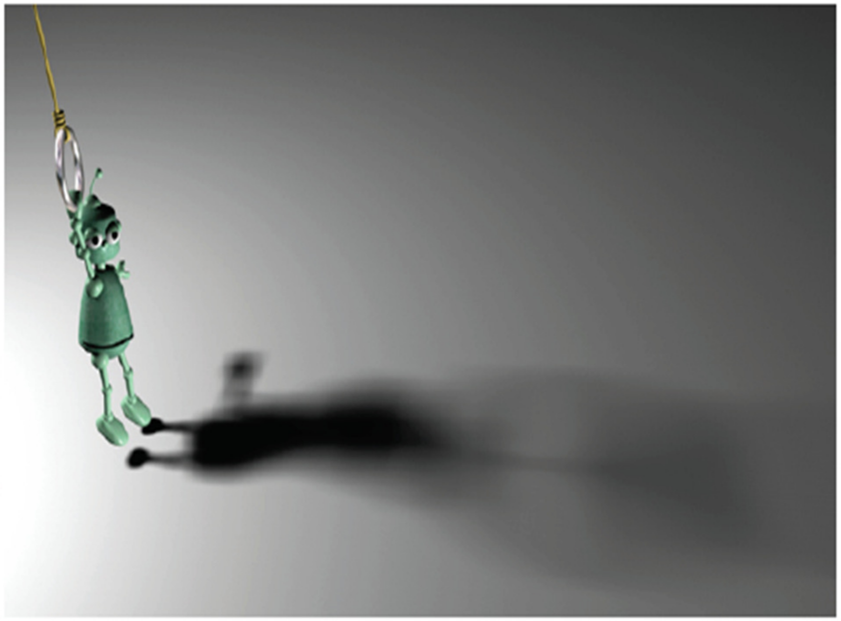
- Point and directional lights lead to hard shadows, no penumbra
- Area light sources lead to soft shadows, with penumbra



Hard and Soft Shadows



Hard shadow from point light source



Soft shadow from area light source

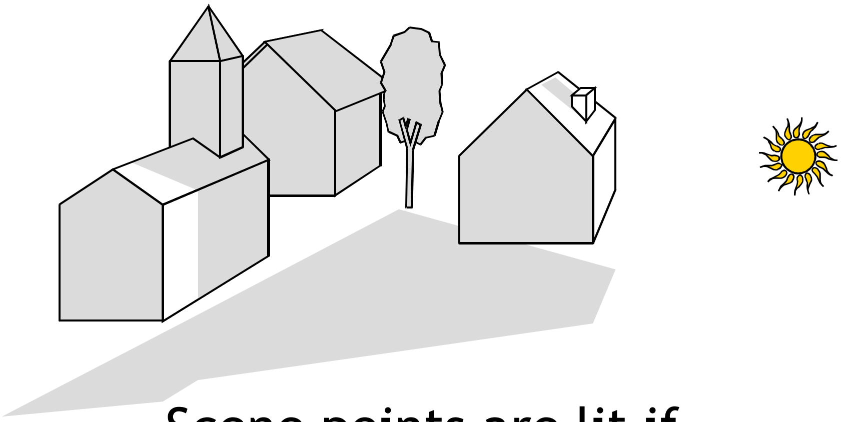
Shadows for Interactive Rendering

- In this course: hard shadows only
  - Soft shadows hard to compute in interactive graphics
- Two most popular techniques:
  - Shadow mapping
  - Shadow volumes
- Many variations, subtleties
- Active research area

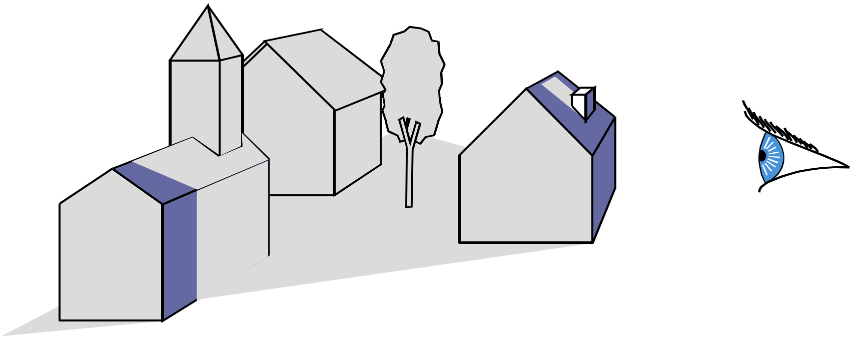
Shadow Mapping

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

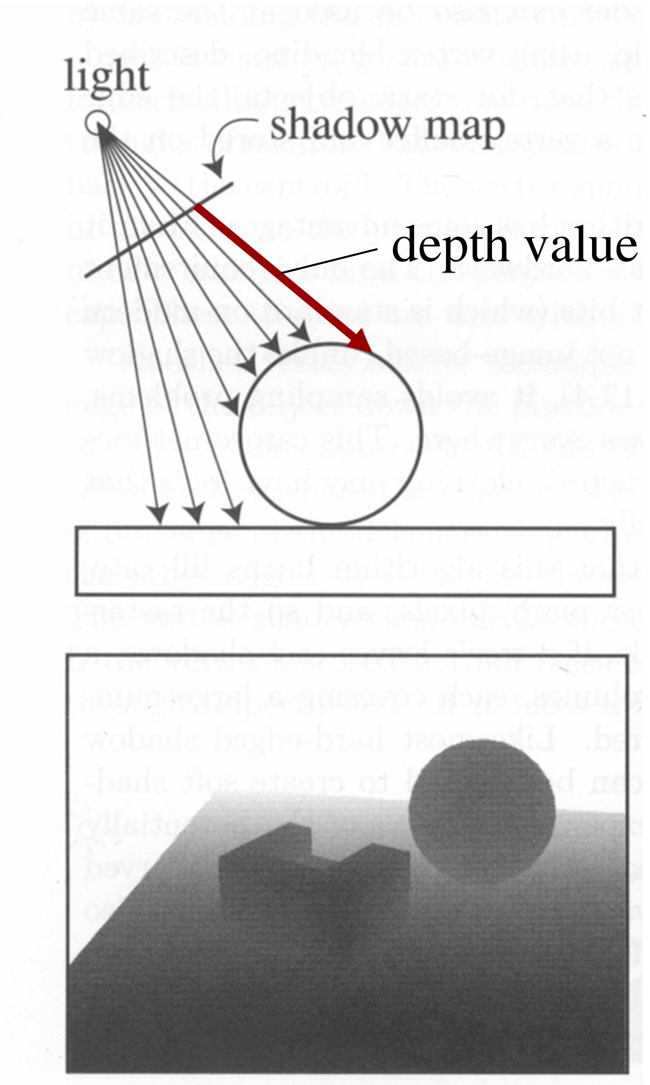


Determine visibility from light source by placing camera at light source position

Two Pass Algorithm

First Pass

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

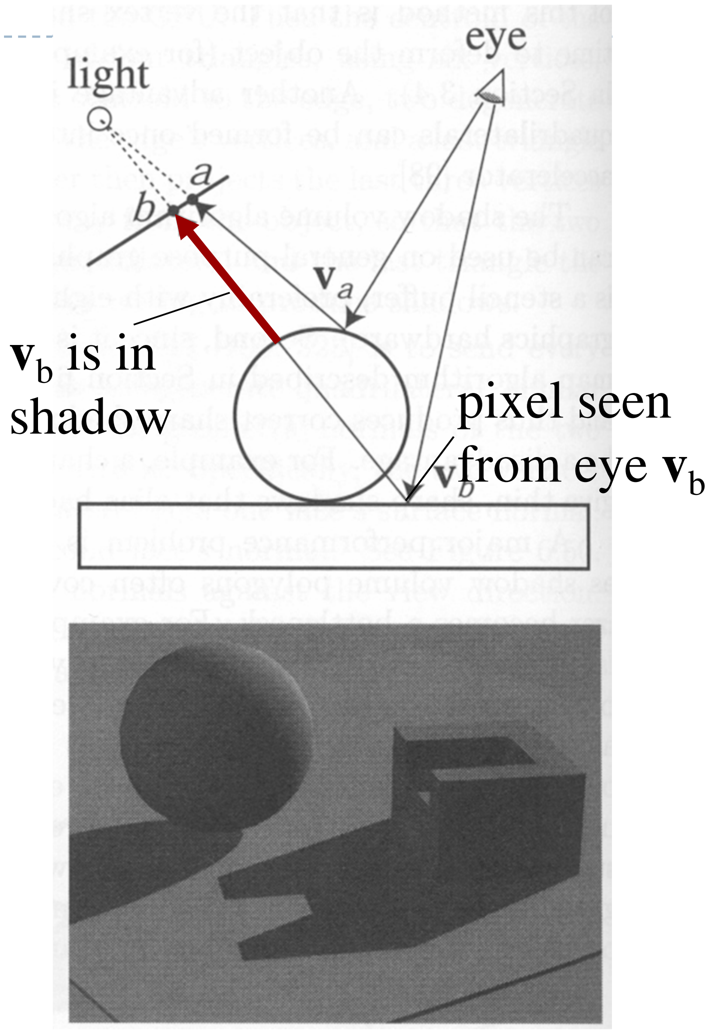


Depth image as seen from light source

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

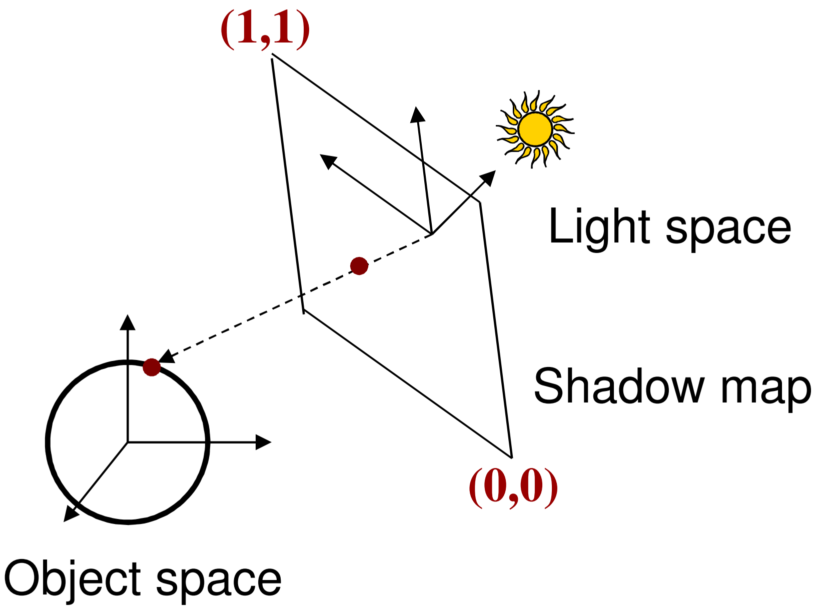

light
eye
a
b
$\mathbf{v}_a$
$\mathbf{v}_b$ is in shadow
pixel seen from eye $\mathbf{v}_b$
$\mathbf{v}_b$

Final image with shadows

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

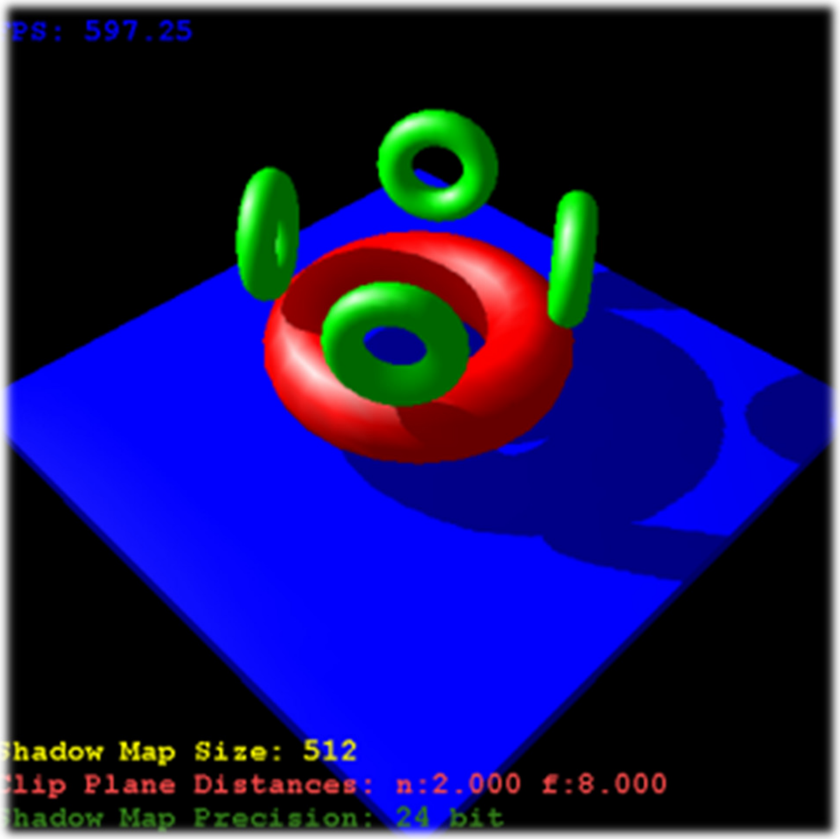
- Declare shadow map as sampler2DShadow
- Look up shadow map using projective texturing with `vec4 texture2DProj(sampler2D, vec4)`

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

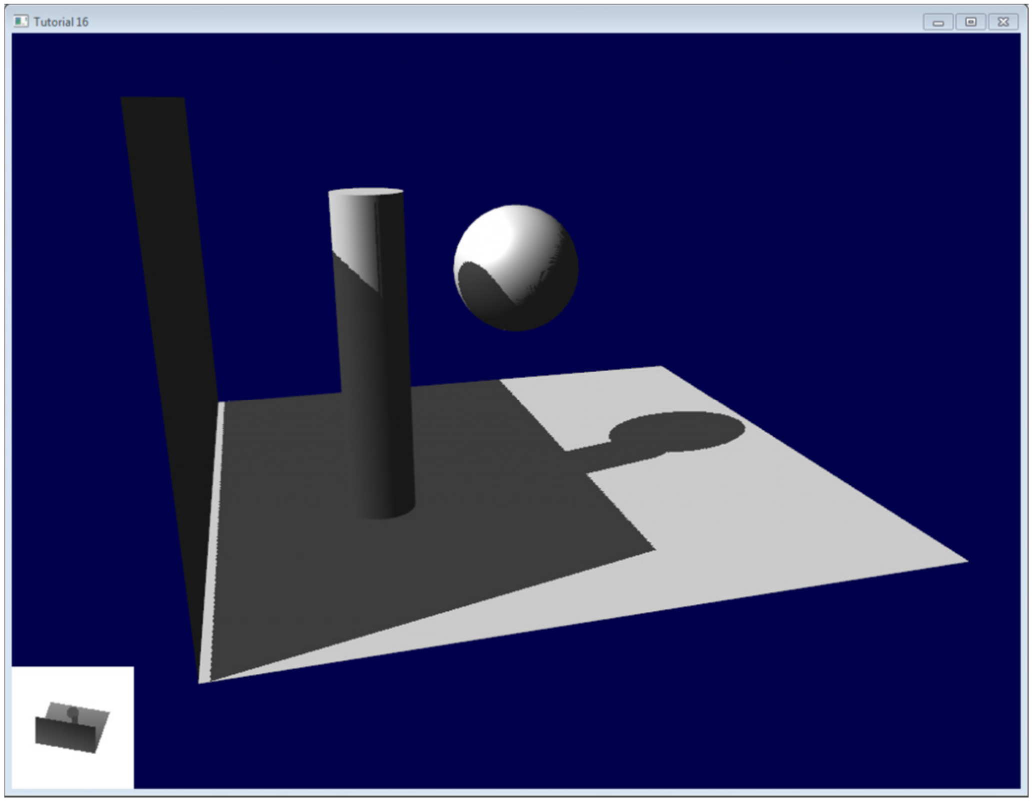
- Shadow mapping demo from http://www.paulsprojects.net/opengl/shadowmap/shadowmap.html



```
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/

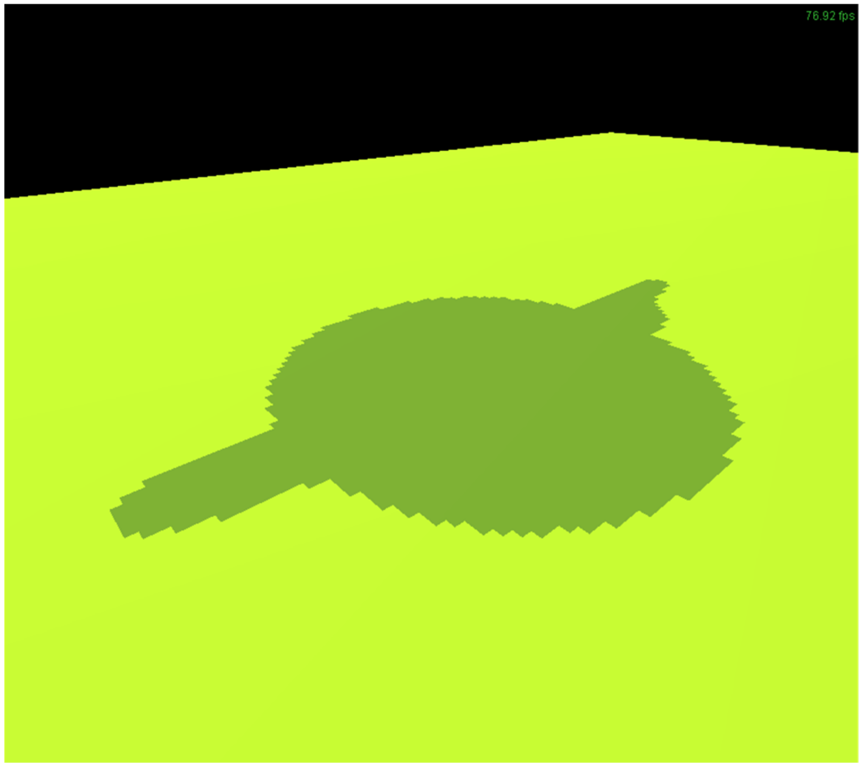


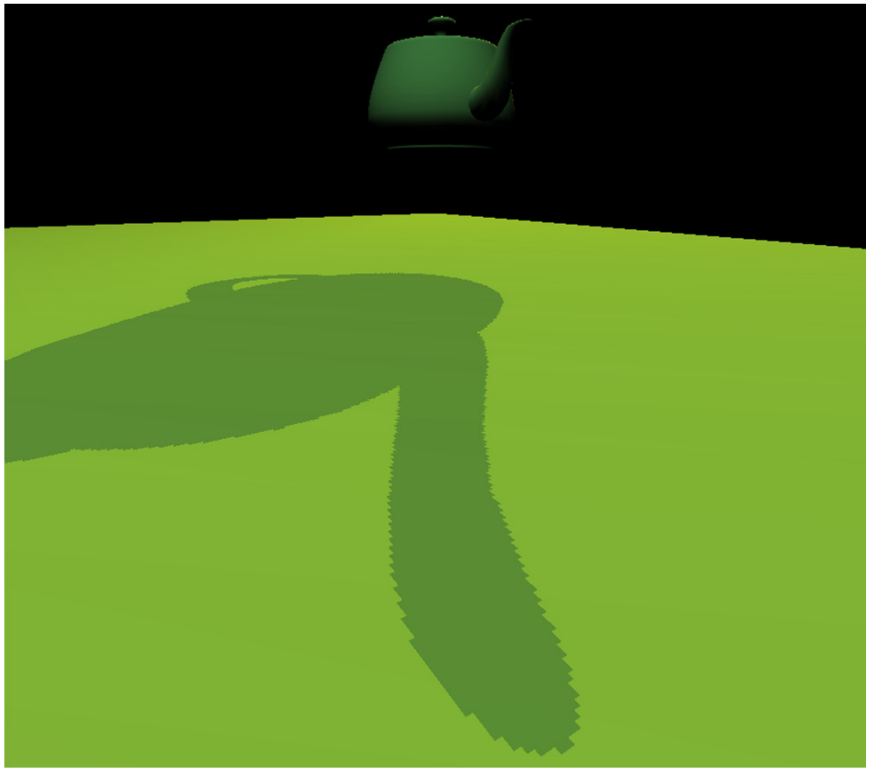
Issues With Shadow Maps

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

Sampling Problems

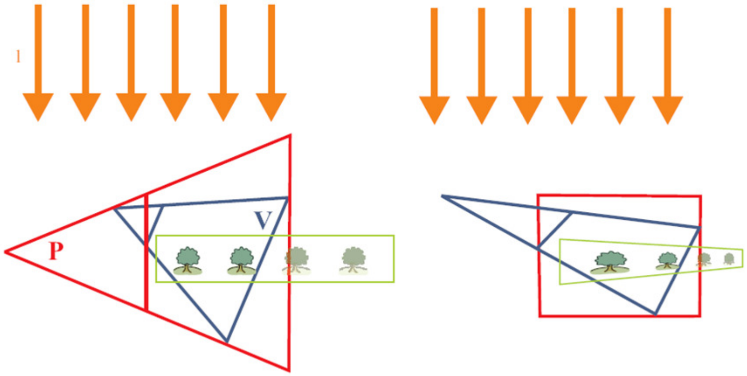
- Shadow map pixel may project to many image pixels
  → Stair-stepping artifacts





Solutions

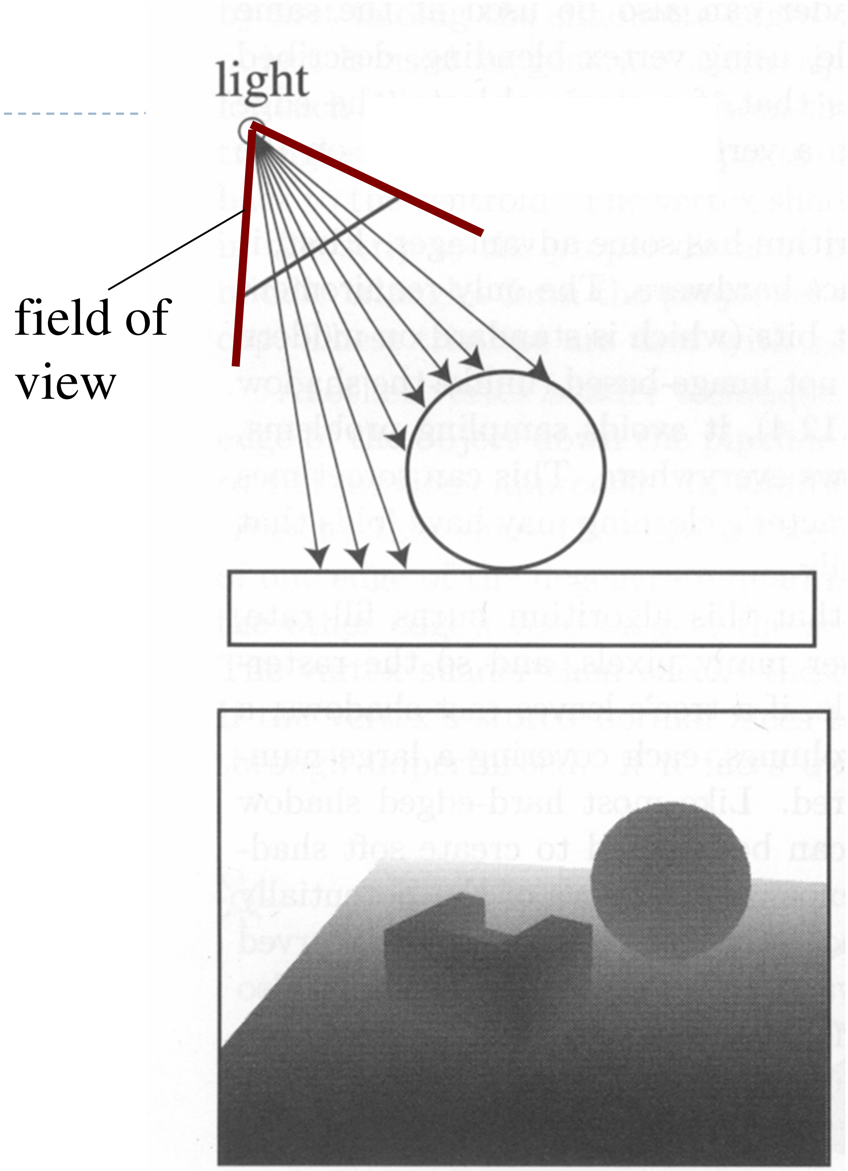
- Increase resolution of shadow map
  - Not always sufficient
- Split shadow map into several tiles
- Tweak projection for shadow map rendering
  - Light space perspective shadow maps (LiSPSM) http://www.cg.tuwien.ac.at/research/vr/lispsm/



- Combination of splitting and LiSPSM
  - Basis for most commercial implementations

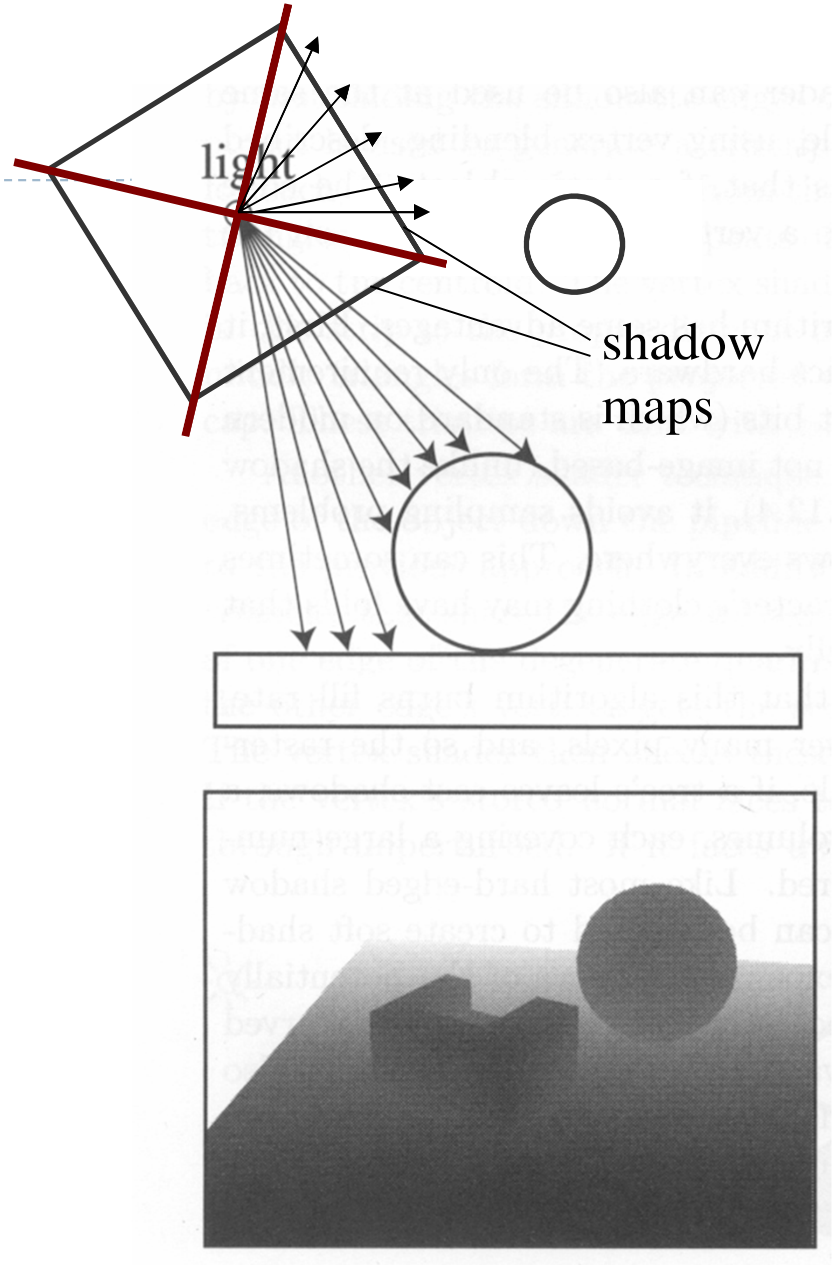
Limited Field of View

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



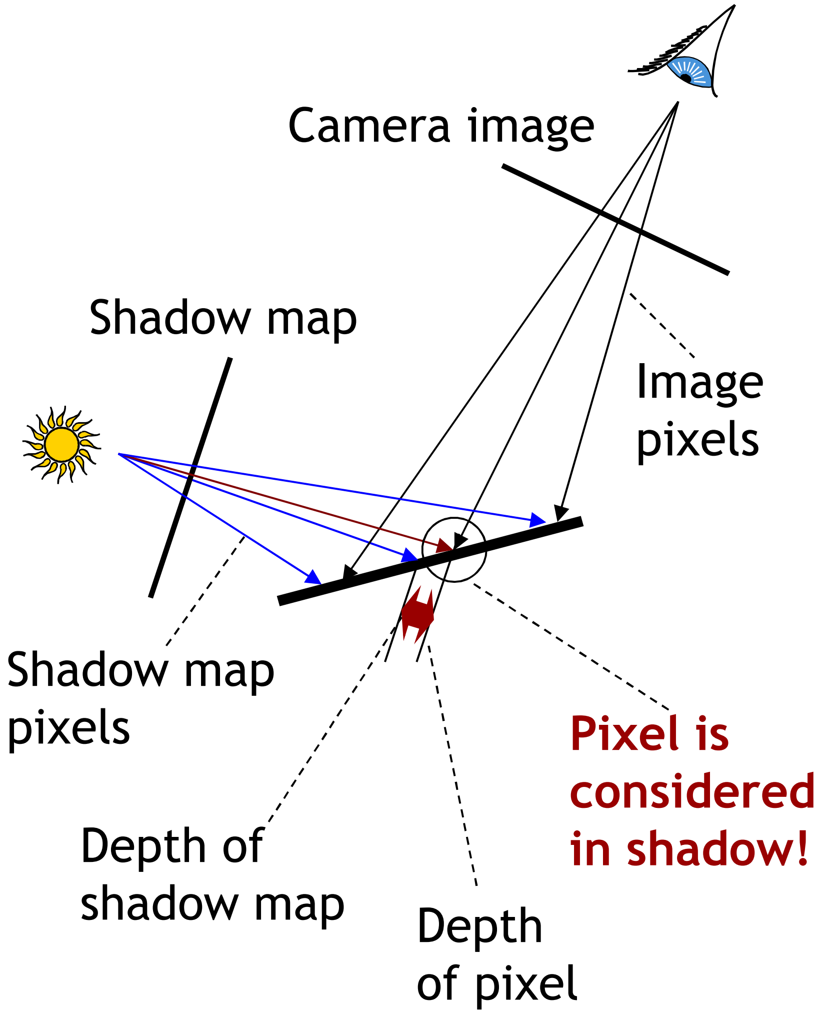
Limited Field of View

- 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



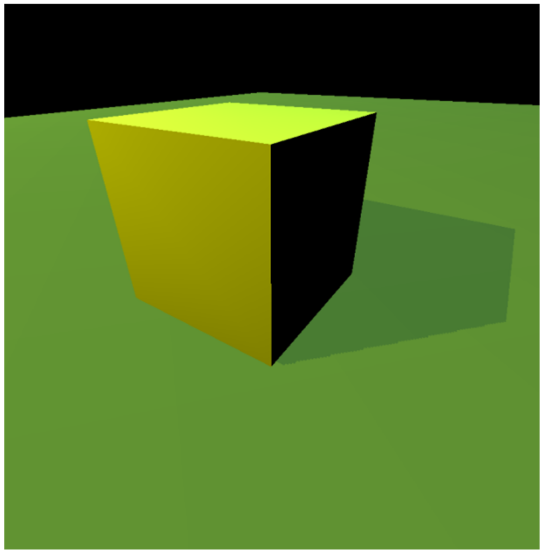
Z-Fighting

- 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

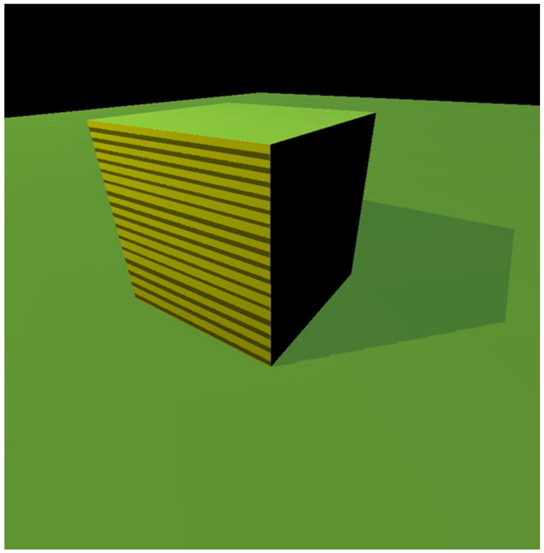


Solution: Bias

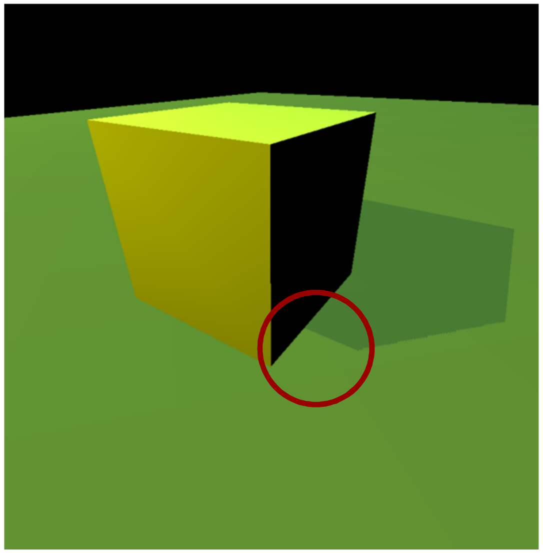
- Add bias when rendering shadow map
  - Move geometry away from light by small amount
- Finding correct amount of bias is tricky



Correct bias

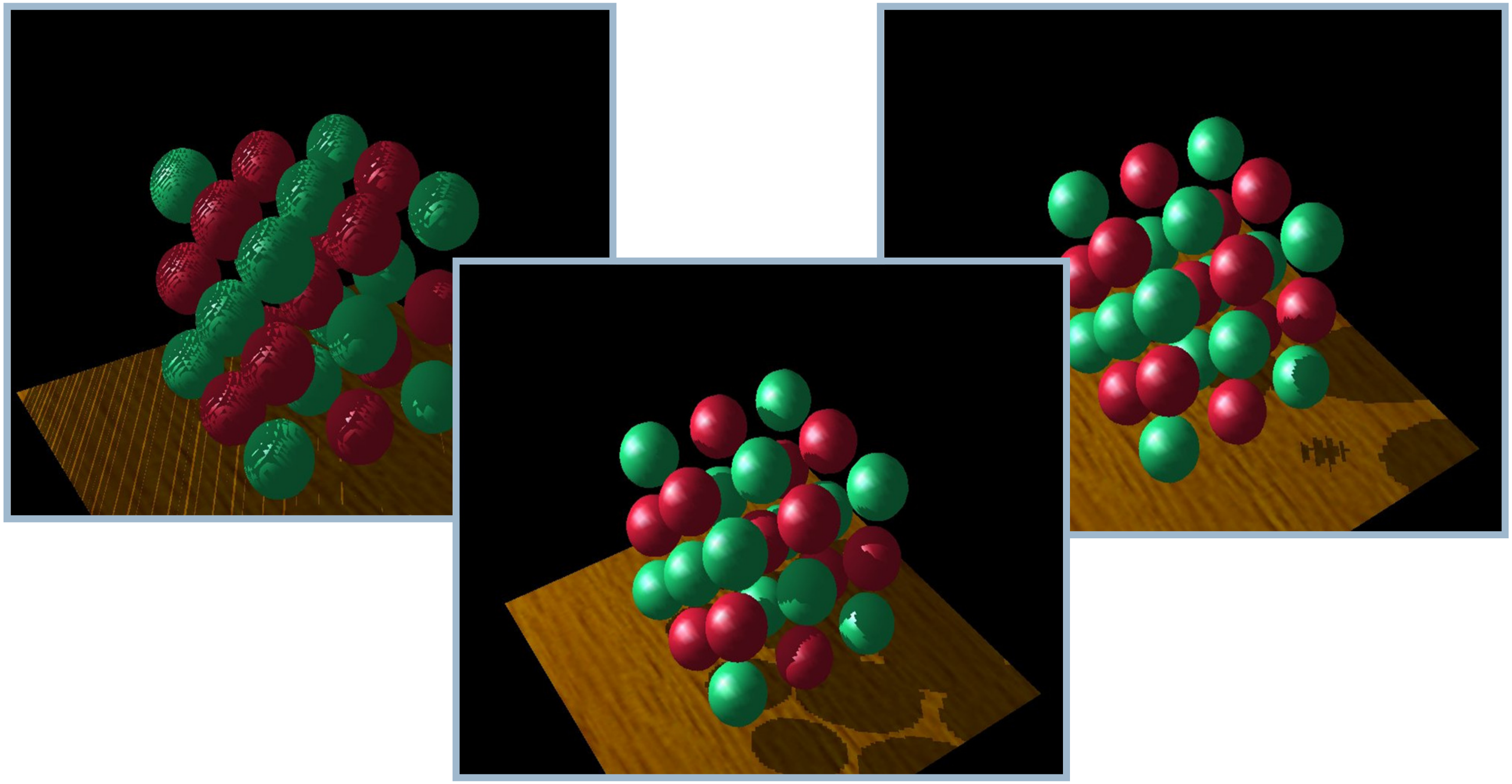


Not enough bias



Too much bias

Bias Adjustment



Not enough

Too much

Just right