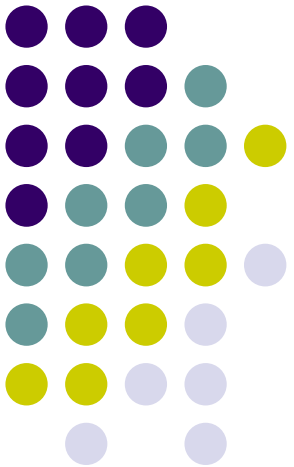
Computer Graphics (CS 563)
Lecture 5: Advanced Computer Graphics
Global Illumination

Prof Emmanuel Agu

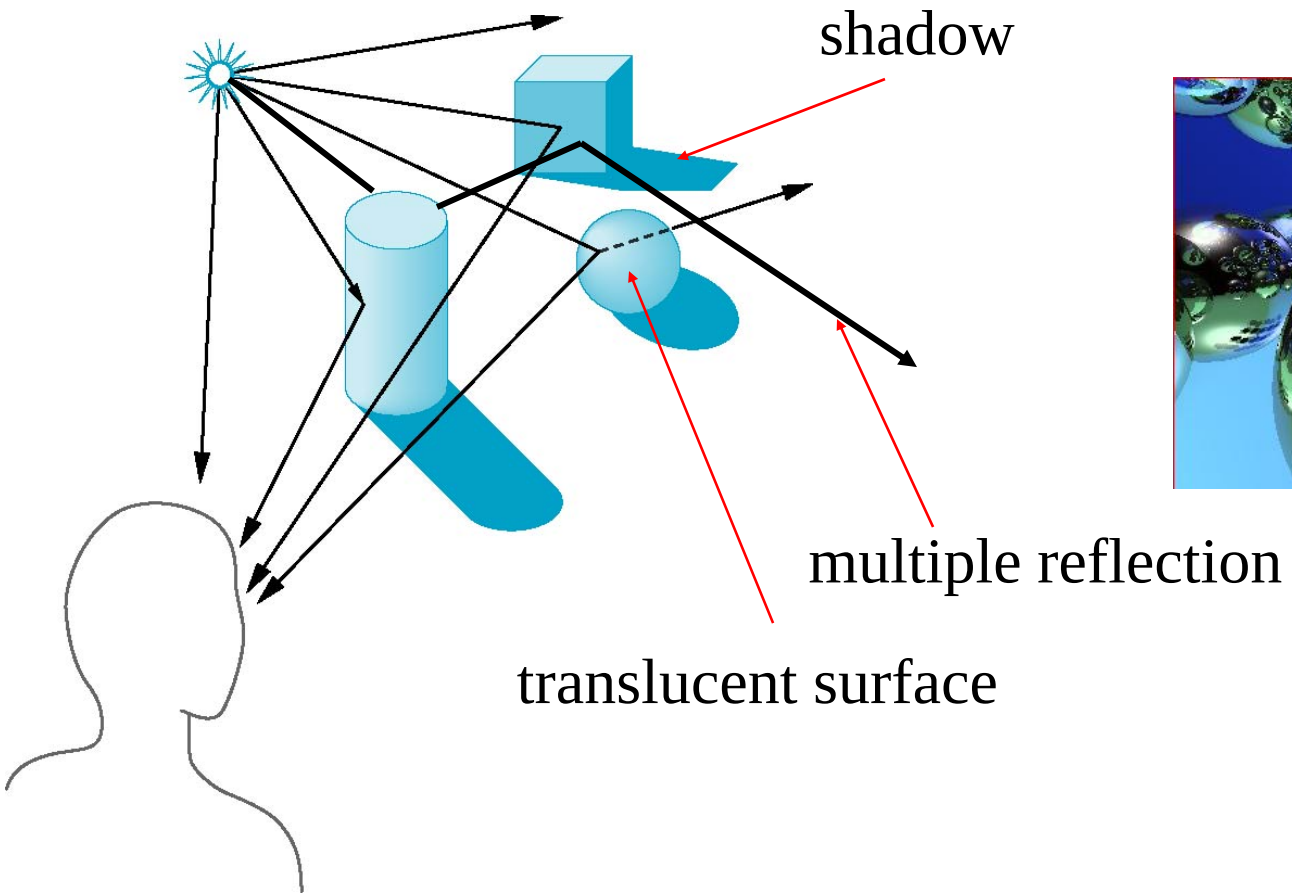
Computer Science Dept.

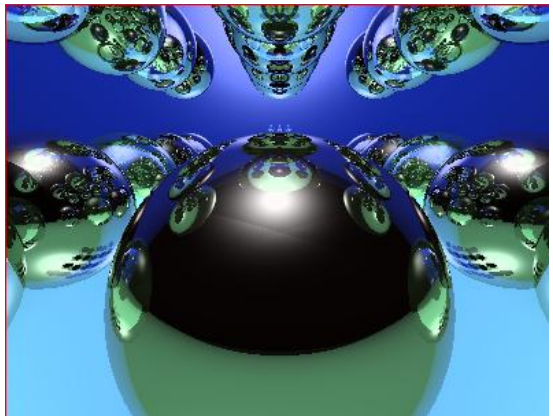
Worcester Polytechnic Institute (WPI)



Global Illumination

- Global illumination: model interaction of light from all surfaces in scene (track multiple bounces)





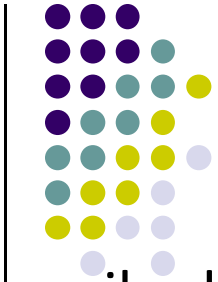
Effects of GI on Game Engine Images

- Images without GI look fake (e.g. Torque 3D)
- Images with GI look more real (e.g. Crytek's Crysis)

Torque 3D

Crysis Engine

Rendering Equation



- The infinite reflection, scattering and absorption of light is described by the ***rendering equation***
- Introduced by James Kajiya in 86 Siggraph paper.
- Mathematical basis for all global illumination algorithms

$$L_o = L_e(x,\vec{\omega}) + \int_{\Omega} fr(x,\vec{\omega}',\vec{\omega}) Li(x,\vec{\omega}')(\vec{\omega}' \cdot \vec{n}) d\vec{\omega}'$$

- Lo is outgoing radiance
- Le emitted radiance,
- Lr reflected radiance
- ***fr*** is bidirectional reflectance distribution function (BRDF)
  - Describes how a surface reflects light energy
  - Reflectance is fraction of incident light reflected

Rendering Equation

Rendering Computation

- Rendering Equation:

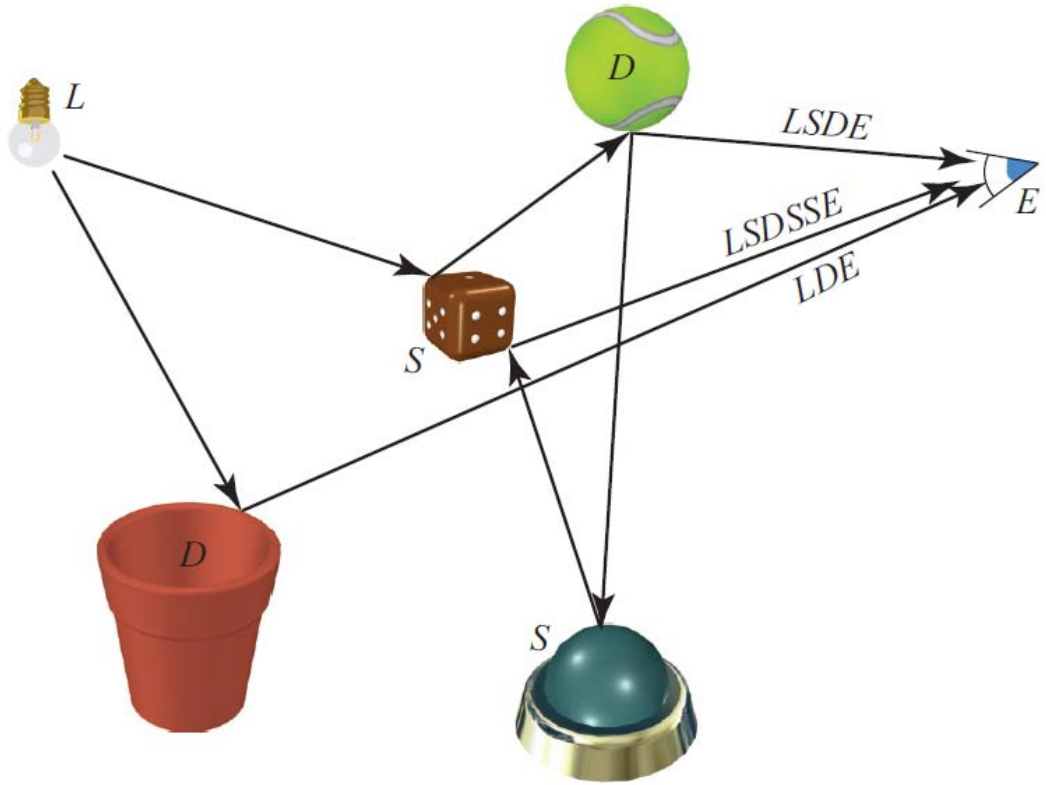
$$L_x(\omega_r) = \text{Emitted Light} + \int \text{Incident Light} * \text{BRDF} \, d\omega_i$$

- Emitted light - Only if point on light	- Represents light sources - Light from all directions	- Bidirec. Reflectance Distrib. - Material properties - Gloss / color / ...

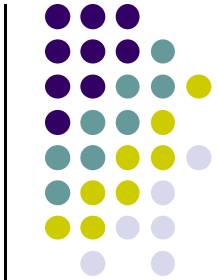
Heckbert Notation

- Formal way of representing possible light paths
- L: from light, E: to eye
- Intermediate bounce

  D (diffuse) or S (specular)



Neumann Expansion



- Using Neumann expansion can separate rendered image into **direct** and **indirect** components

- Direct Illumination: $L_0(\mathbf{x}, \omega_r)$

- Direct + Indirect:

$$L_0(\mathbf{x}, \omega_r) + L_1(\mathbf{x}, \omega_r)$$

Neumann Expansion

Direct + Indirect

$$L_0(\mathbf{x}, \omega_r) + L_1(\mathbf{x}, \omega_r)$$

Direct only

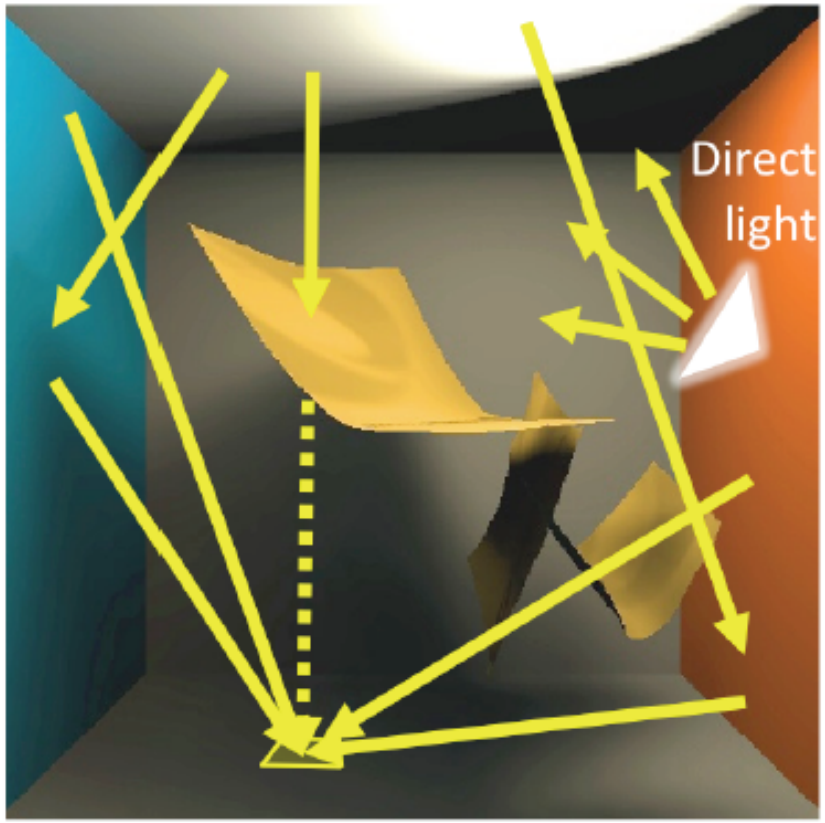
$$L_0(\mathbf{x}, \omega_r)$$

Indirect only

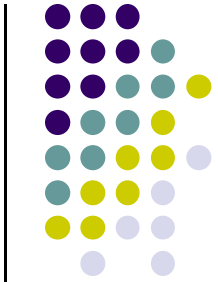
$$L_1(\mathbf{x}, \omega_r)$$

Challenge in Real-Time Global Illumination

- Global illumination is **global**



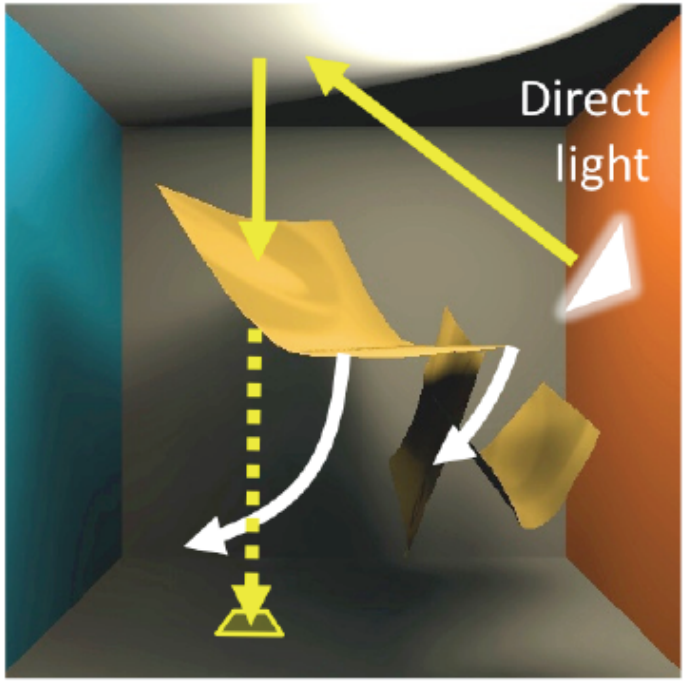
Light from ***all*** directions



Challenge in Real-Time Global Illumination

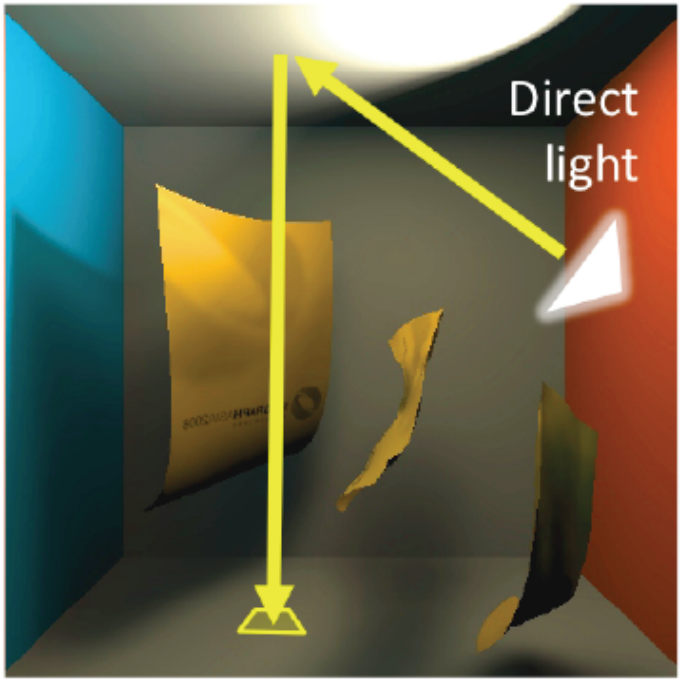
- Dynamic indirect visibility

Frame *t*

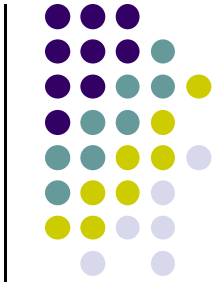


Indirect shadow

Frame *t*+1



No indirect shadow



Common Shading Algorithms

- Real-time Rendering: Approx. Rendering Equation
  - Direct lighting: many solutions (assume given)
  - Indirect lighting: ambient term

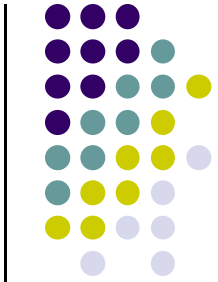
  - Indirect lighting: no visibility

  - Indirect lighting: semi-static scenes

  - Indirect lighting: 1-bounce only

  - ...

GI Tricks



- Full GI is tough to render
- Start with tricks to approximate/fake GI
  - Shadows
  - Reflection
  - Transmittance
  - Caustics, etc

Why shadows?

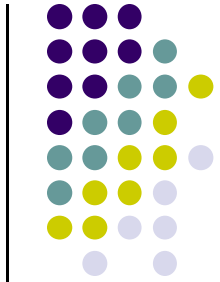
- More realism and atmosphere

Image courtesy of BioWare

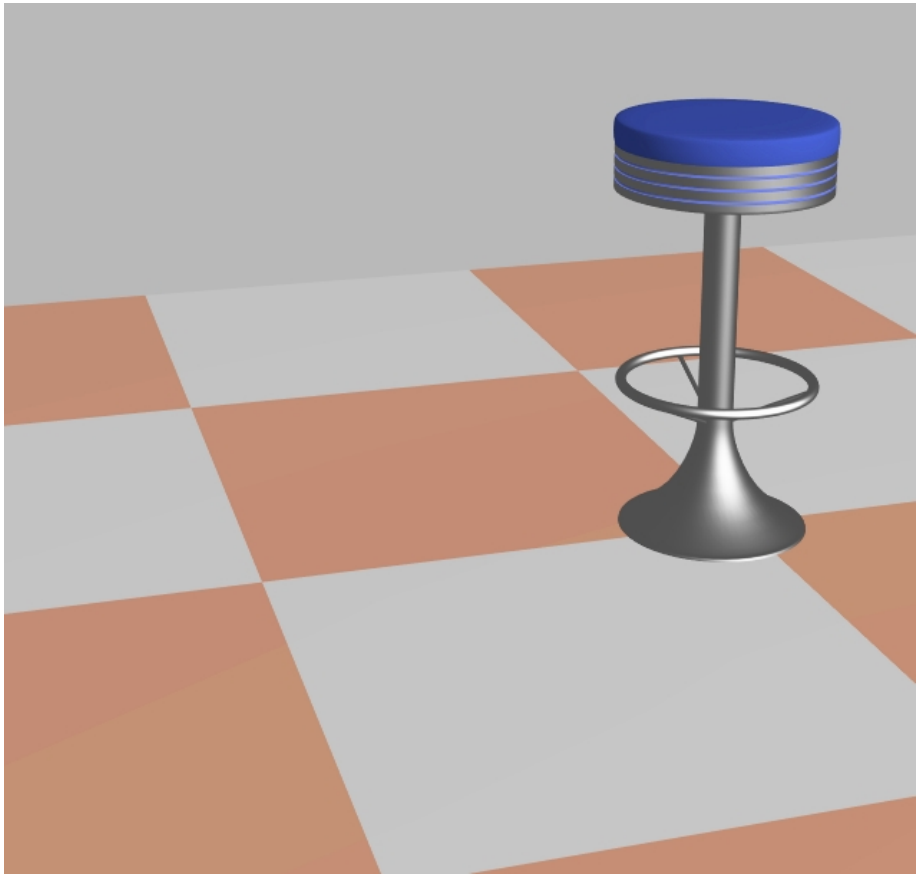
Another example

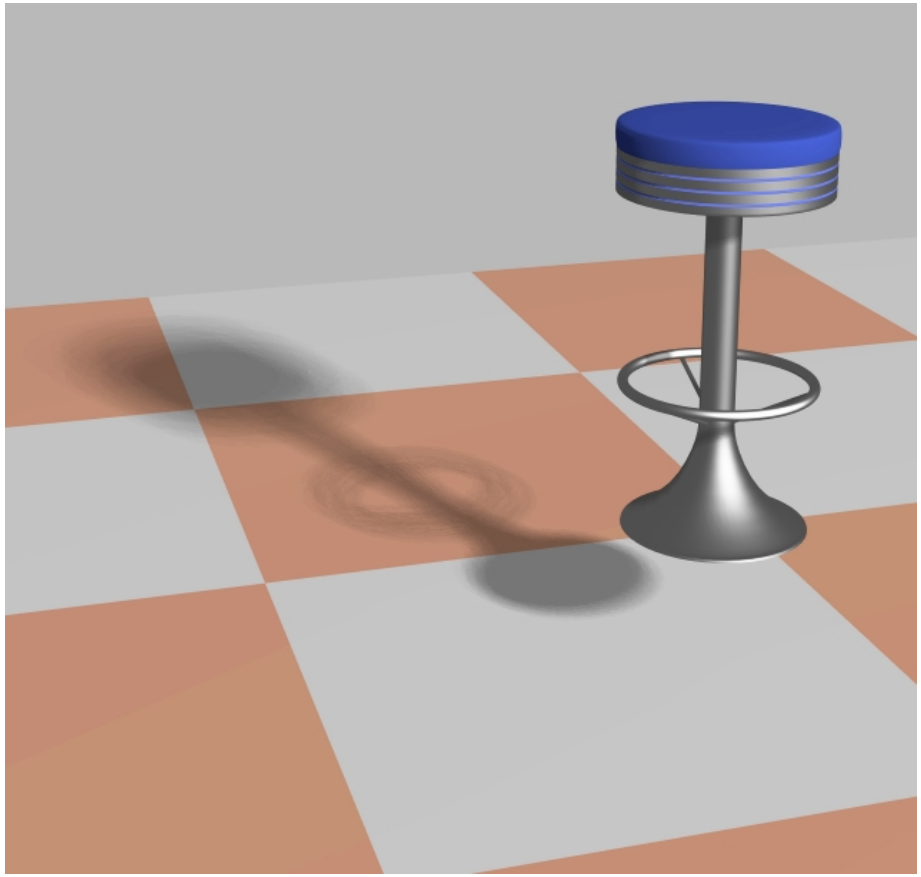
Image courtesy of Codemasters & Rebel Act

Why shadows?

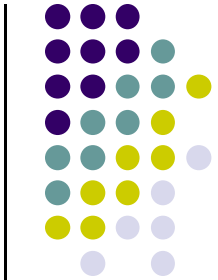


- More clues about spatial relationships
- Orientation & gameplay

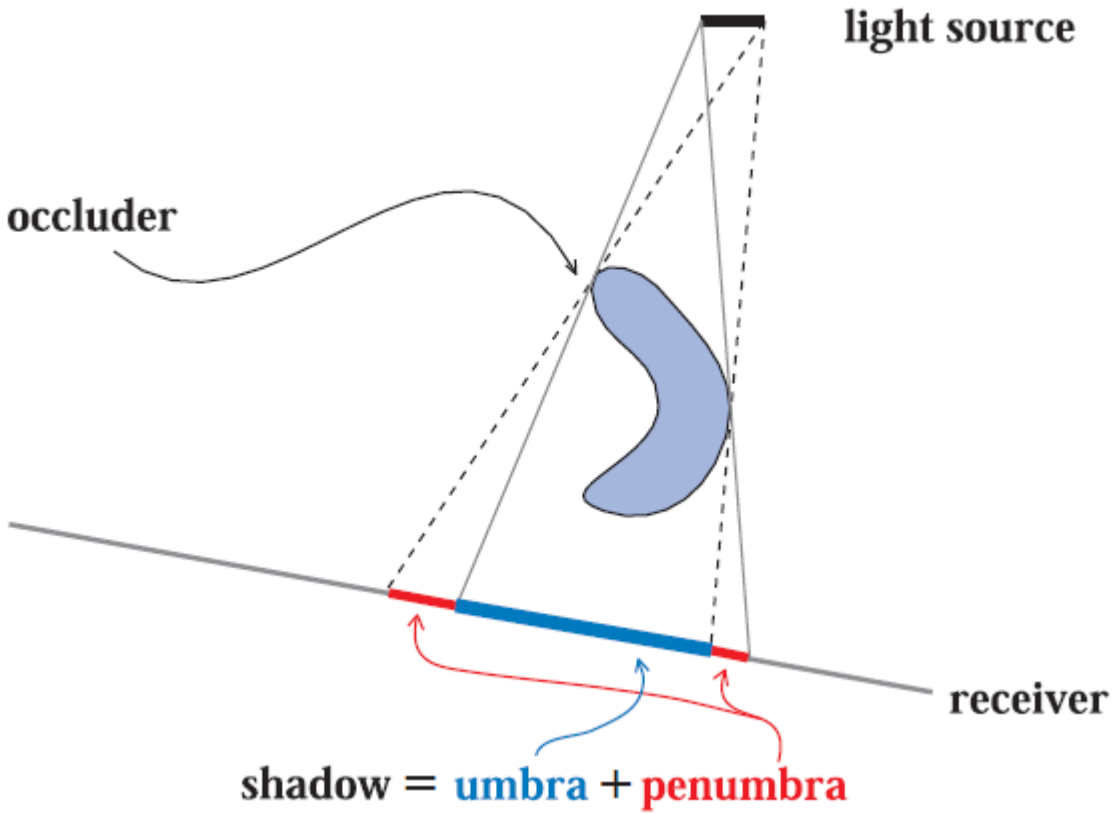




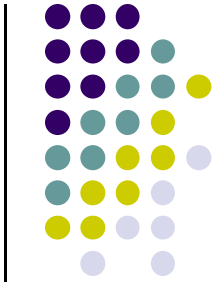
Shadow Terminology



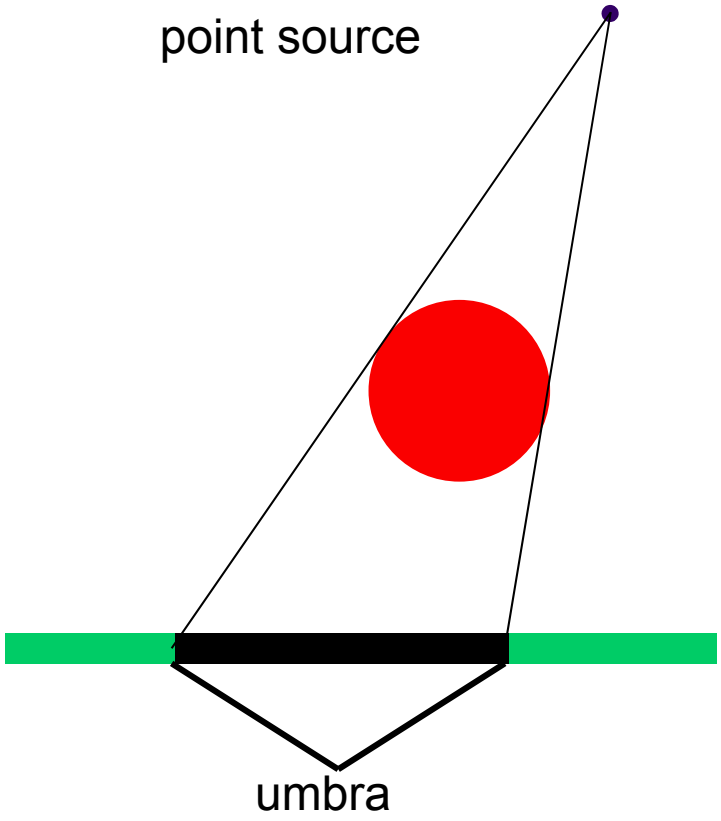
- Occluders block light going from light source to receiver

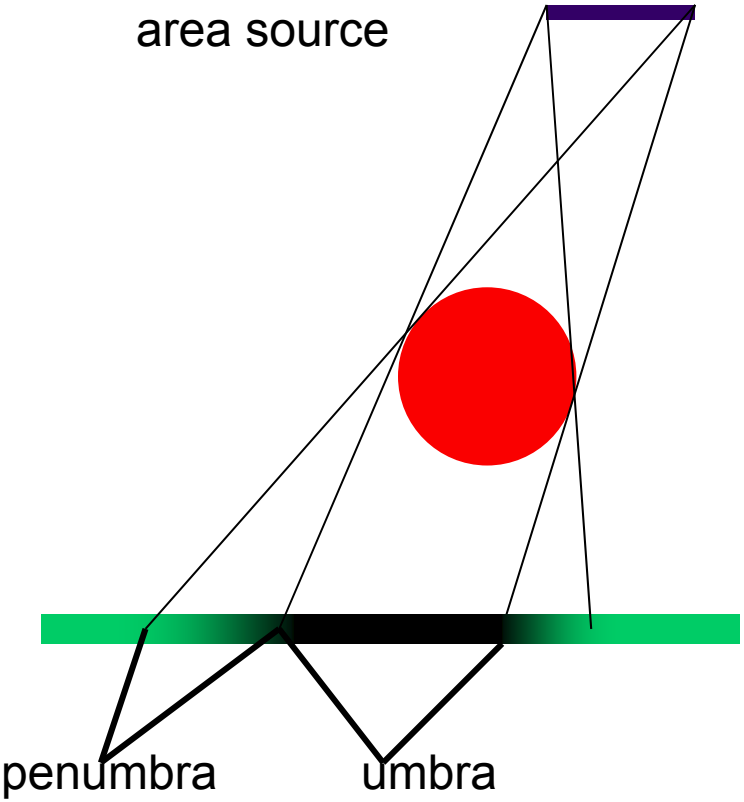


Definitions

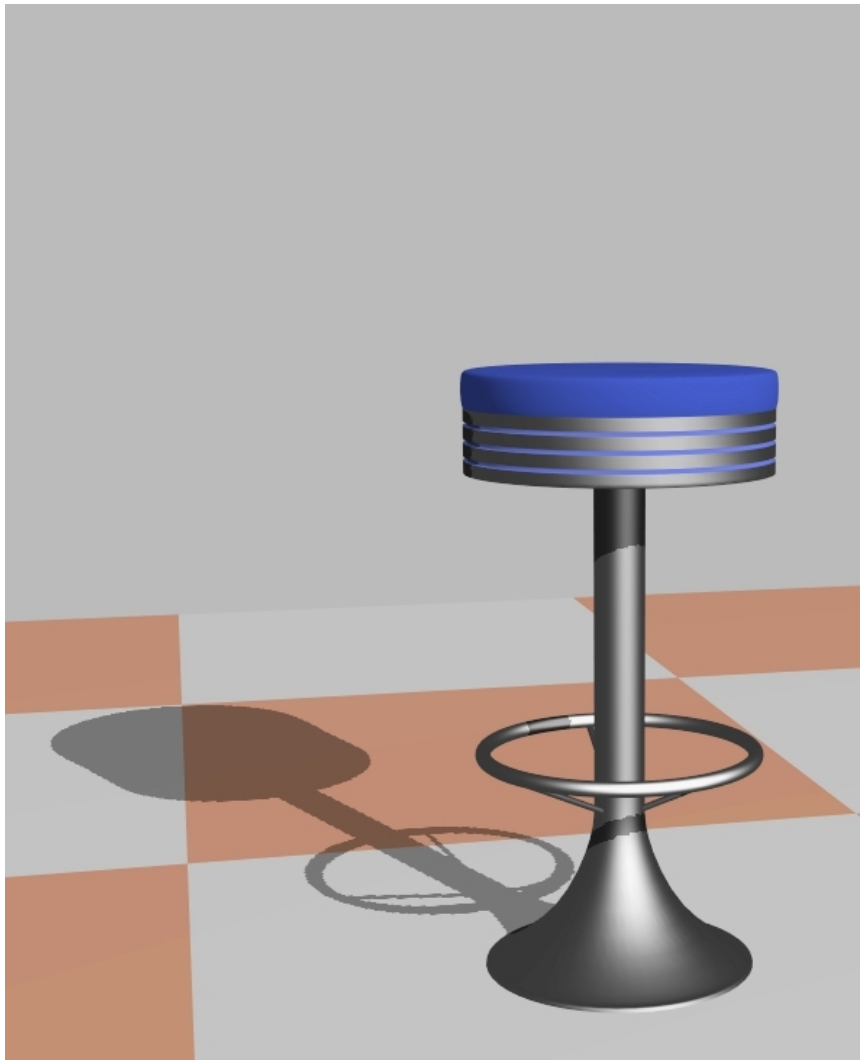


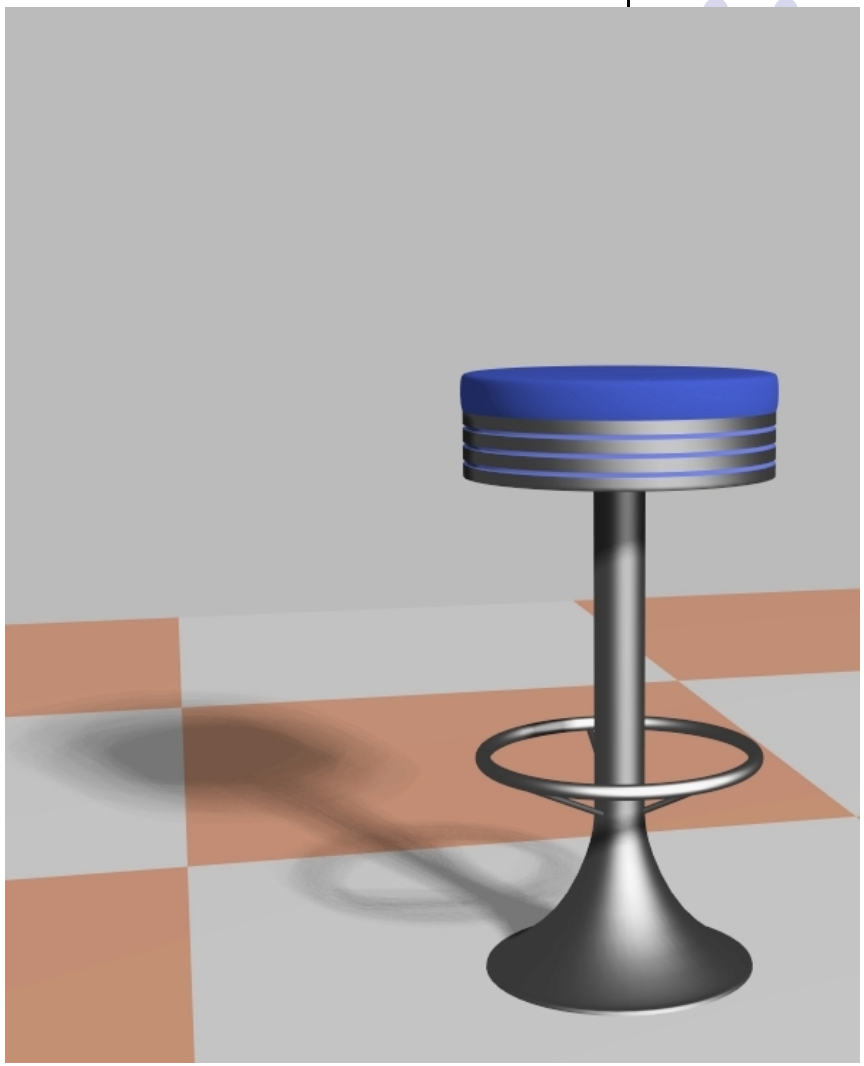
- Point light sources create hard shadows
- Area light sources create soft shadows



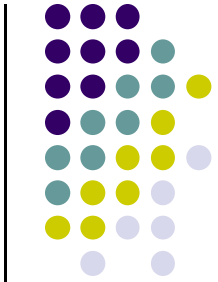


Example: Hard vs Soft Shadows



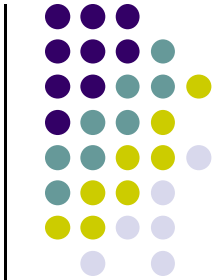


Ways of thinking about Shadows

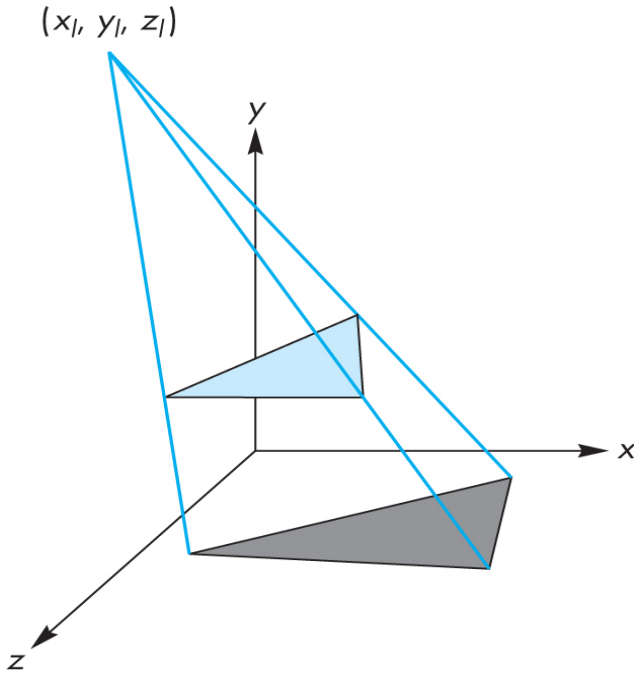


- As separate objects (like Peter Pan's shadow)
  - **Projective shadows**
- As volumes of space that are dark
  - **Shadow volumes** [Franklin Crow 77]
- As places not seen from a light source looking at the scene
  - **Shadow maps** [Lance Williams 78]

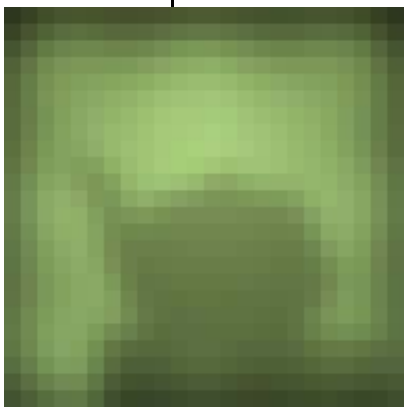
Projective Shadows



- Paint shadows as a texture
- Works for flat surfaces with point light source
- Shadow outline found by calculating projections of object vertices onto plane

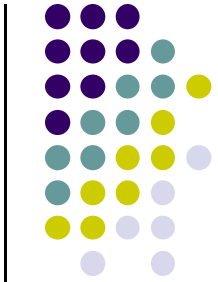


Store precomputed shadows in textures

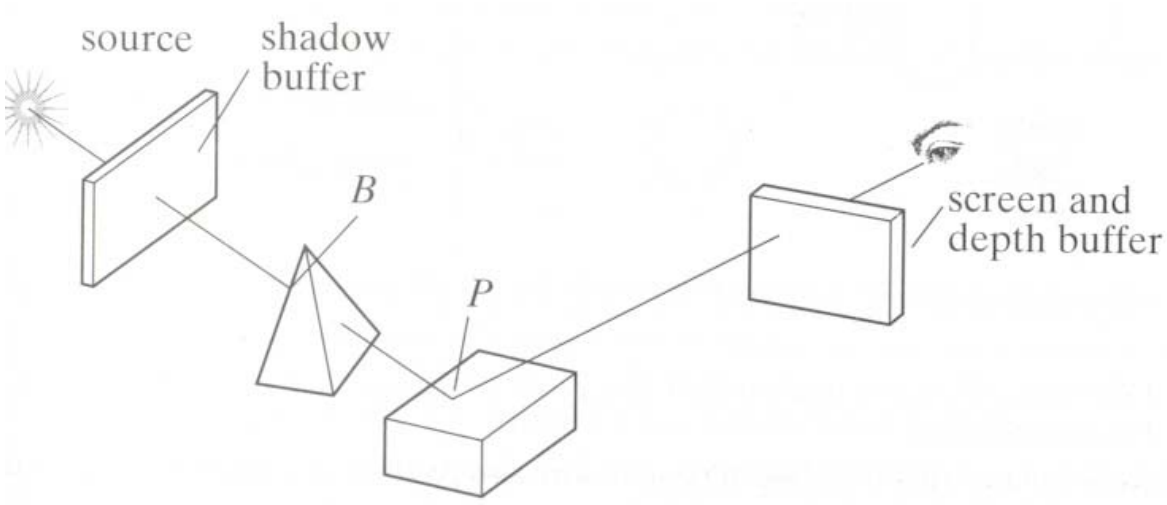


Images courtesy of Kasper Høy Nielsen.

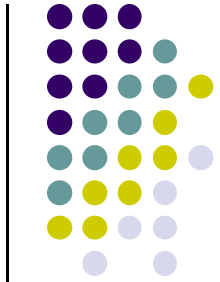
Shadow Map Algorithm



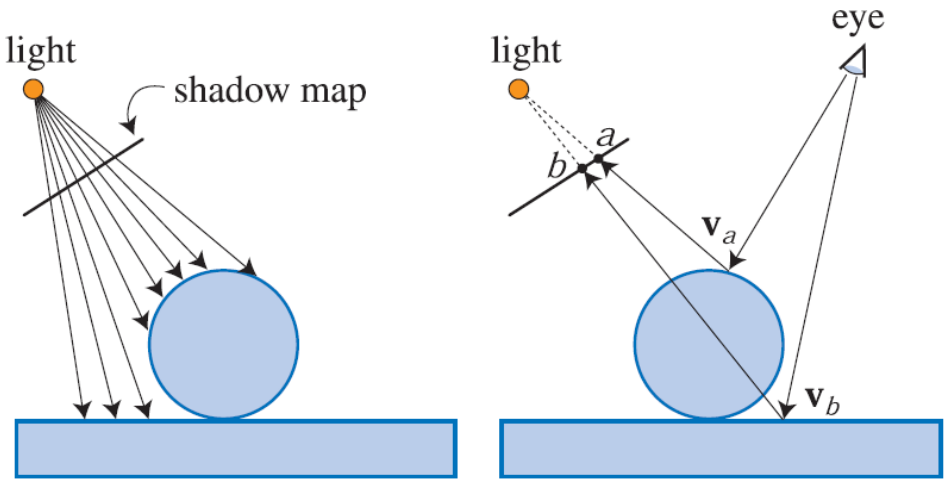
- Render scene from light source
  - Store closest object in each direction in **Shadow Map**
- Render scene from Camera
  - Test if rendered point is closest to light source (in shadow map)
    - If so point is in light
    - Else, point is in shadow

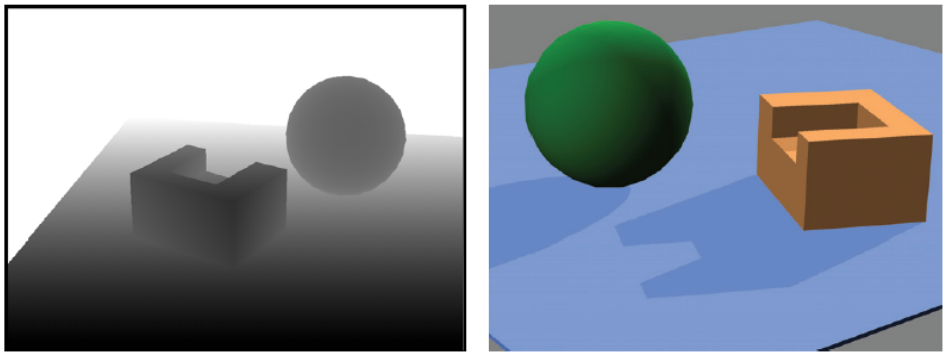


Shadow Map Illustrated

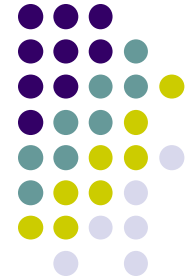


- Point v_a stored in element a of shadow map: lit!
- Point v_b **NOT** in element b of shadow map: In shadow

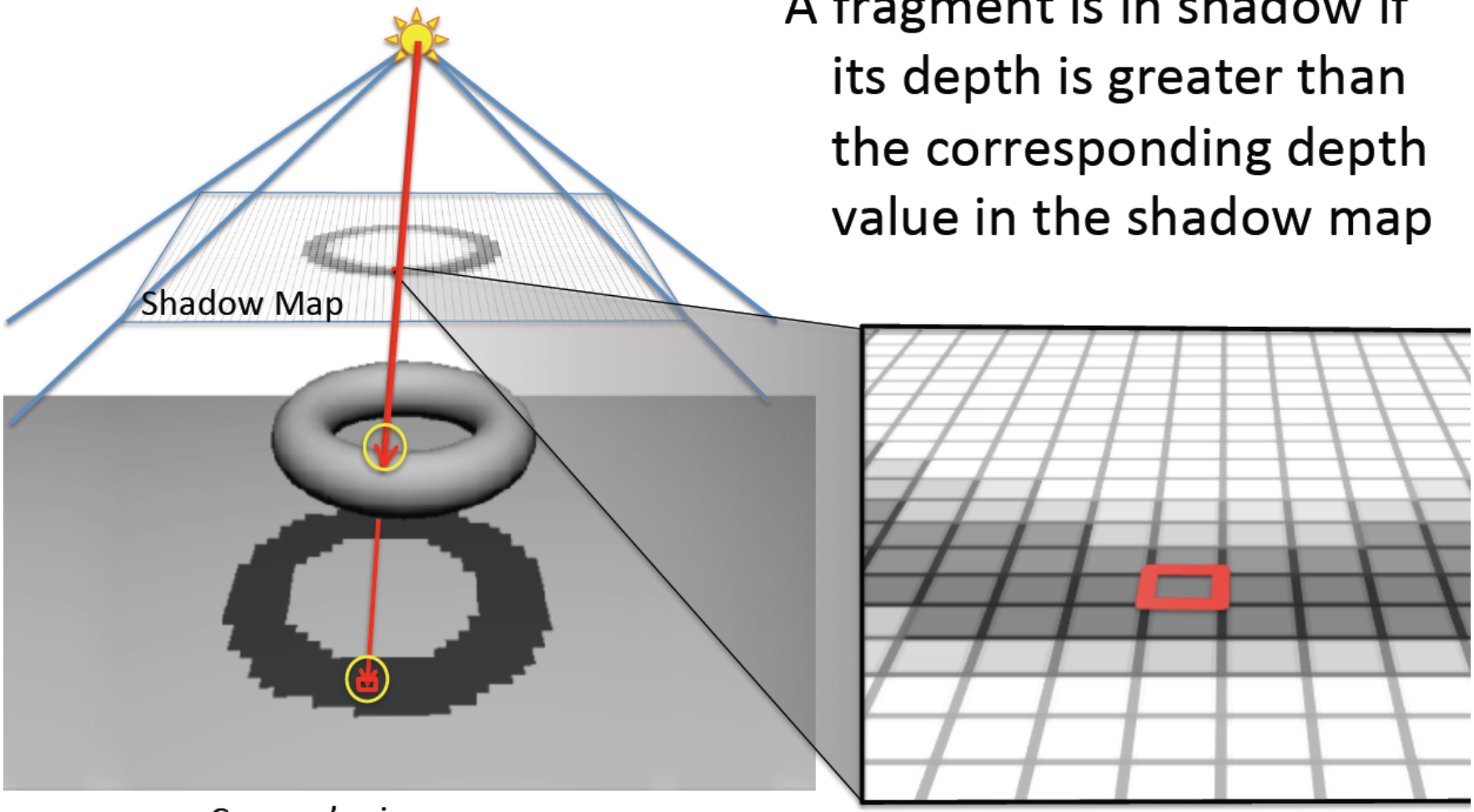




Shadow Map: Depth Comparison

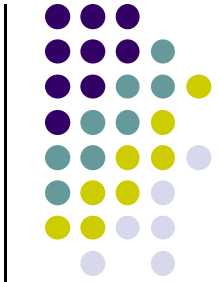


Render depth image from light

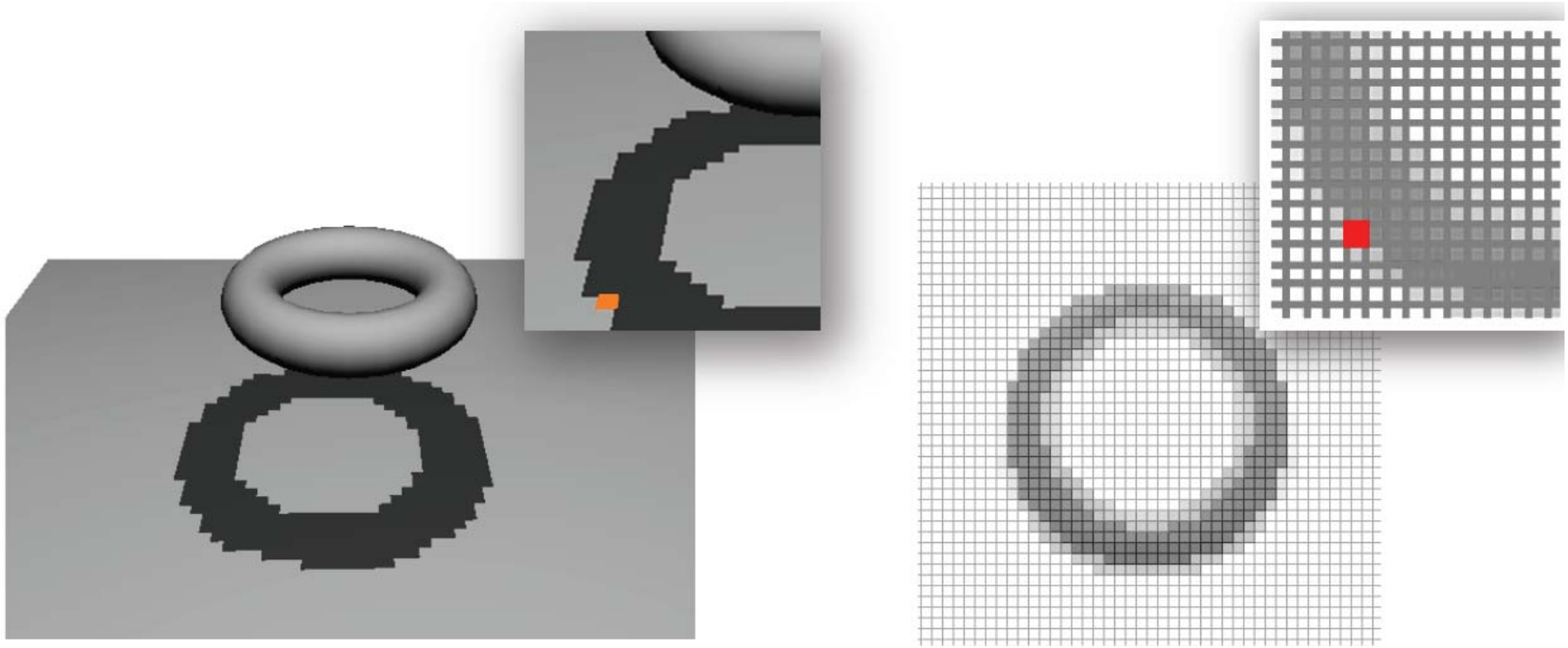


A fragment is in shadow if its depth is greater than the corresponding depth value in the shadow map

Shadow Map Problems



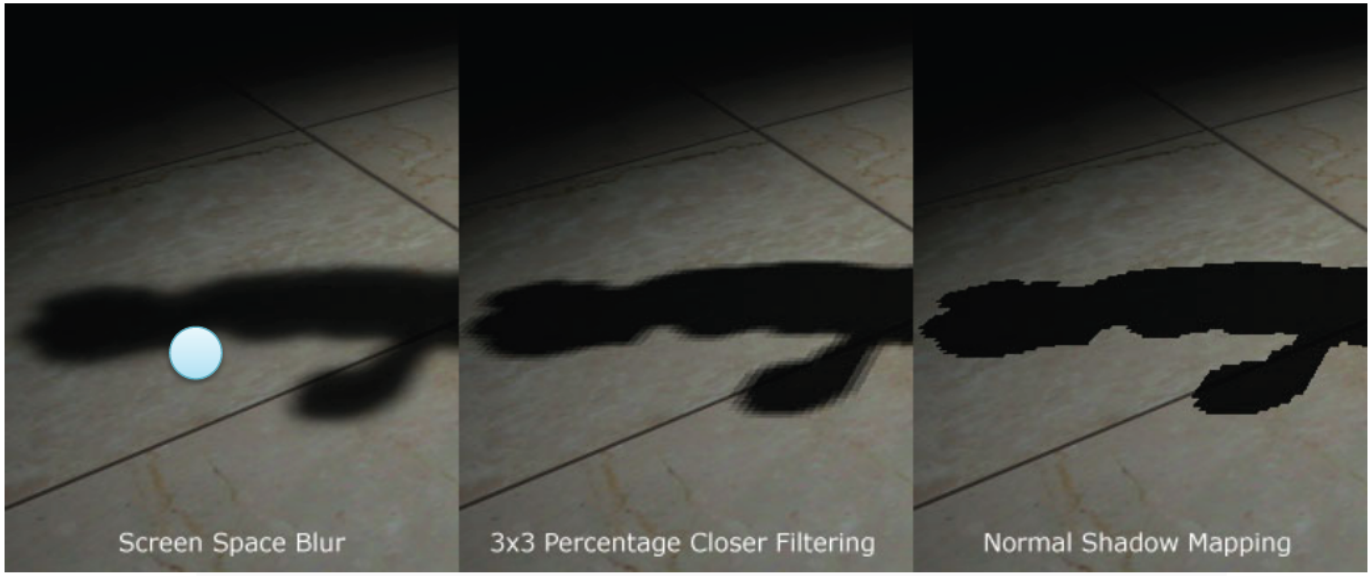
- Low shadow map resolution results in jagged shadows



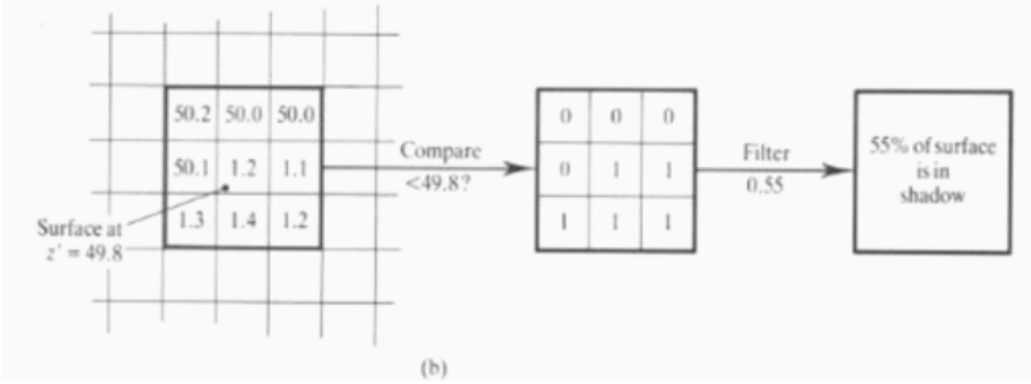
from viewpoint

from light

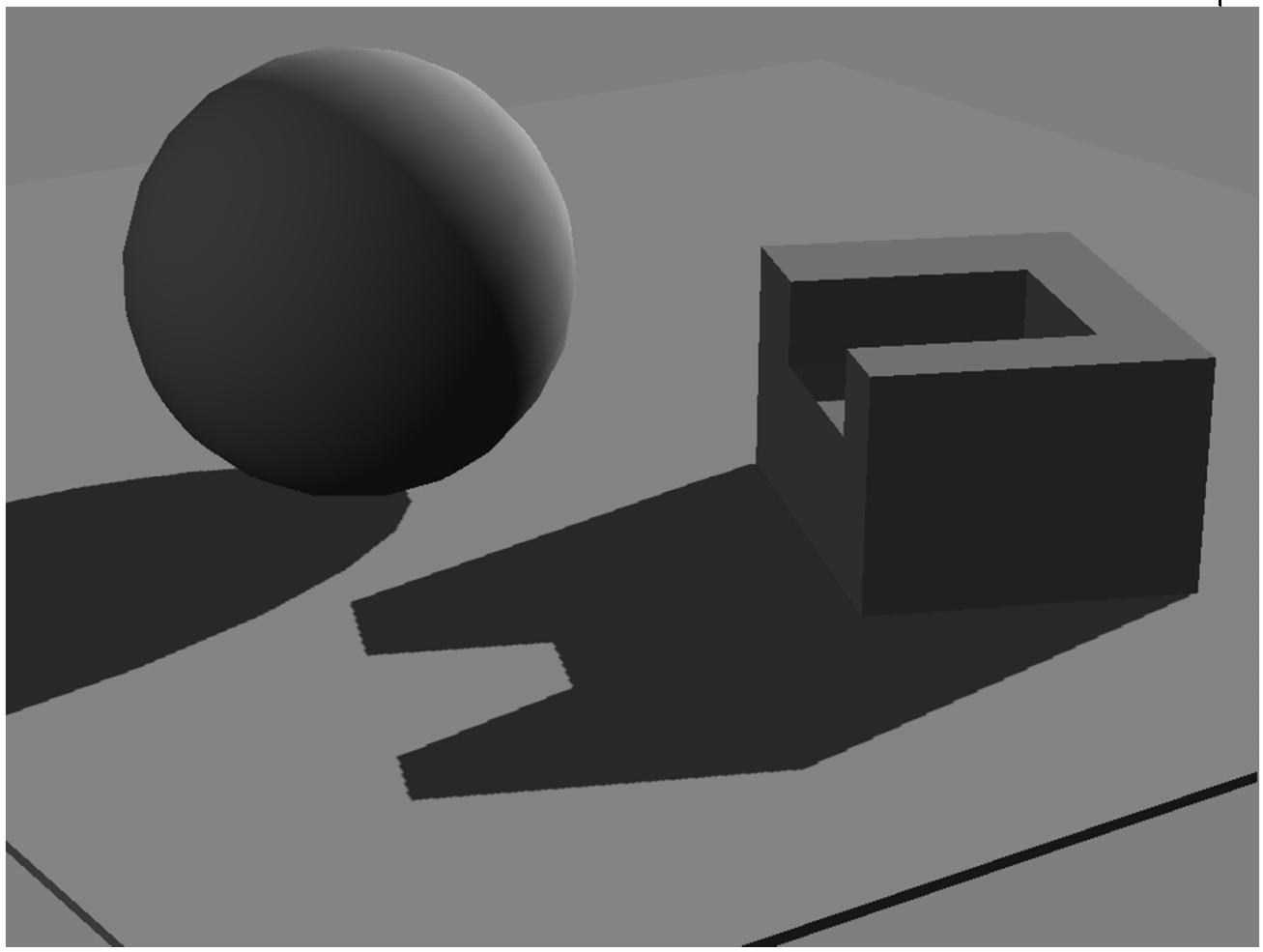
Percentage Closer Filtering



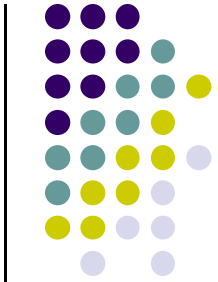
- Blend multiple shadow map samples to reduce jaggies



Shadow Map Result

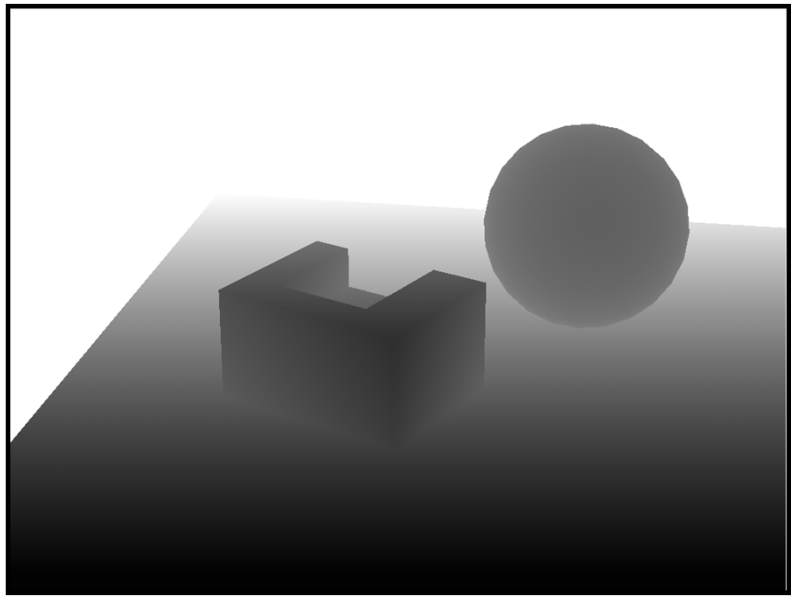


Arbitrary geometry

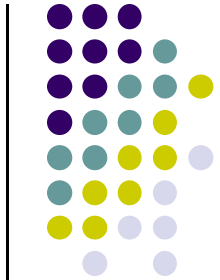


- Shadow mapping and shadow volumes can render shadows onto arbitrary geometry
  - Recent focus on shadow volumes, because currently most popular, and works on most hardware
- Works in real time…
- Shadow mapping is used

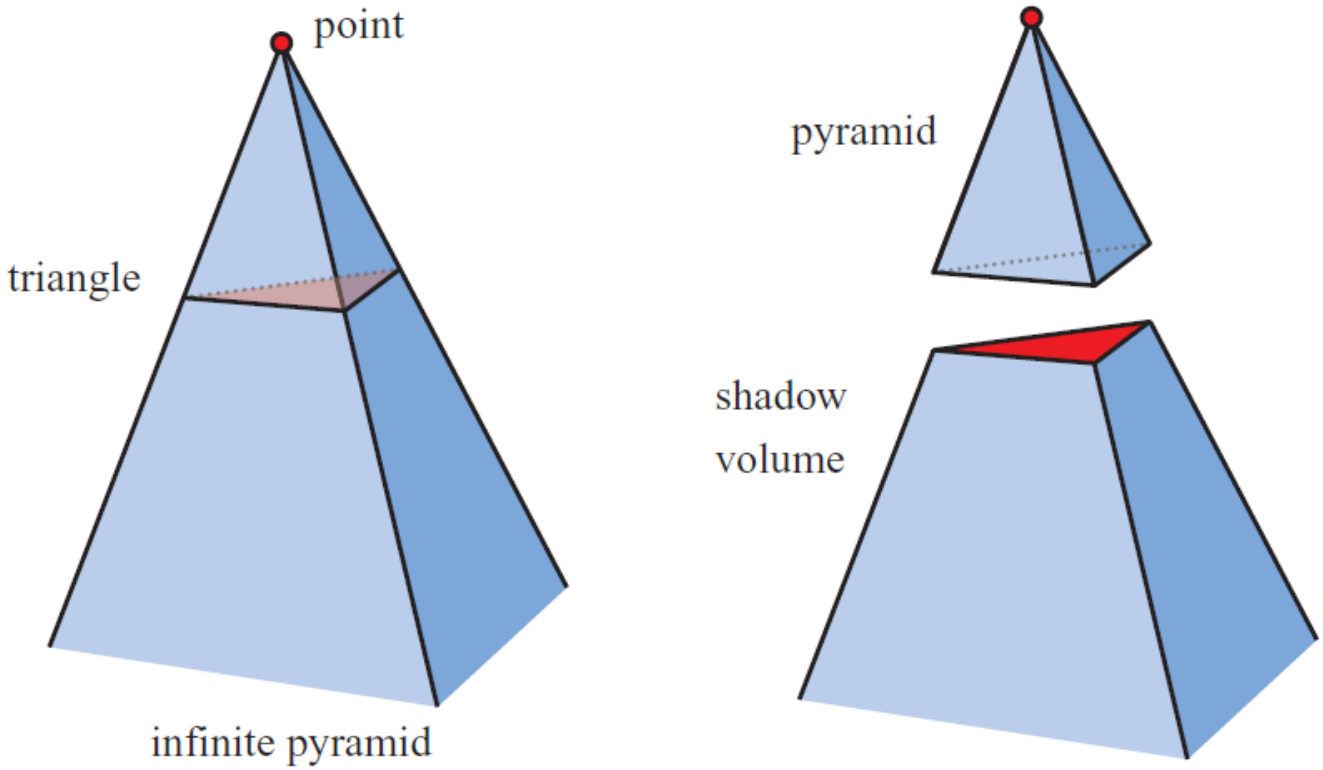
  in Pixar's rendering software



Shadow volumes

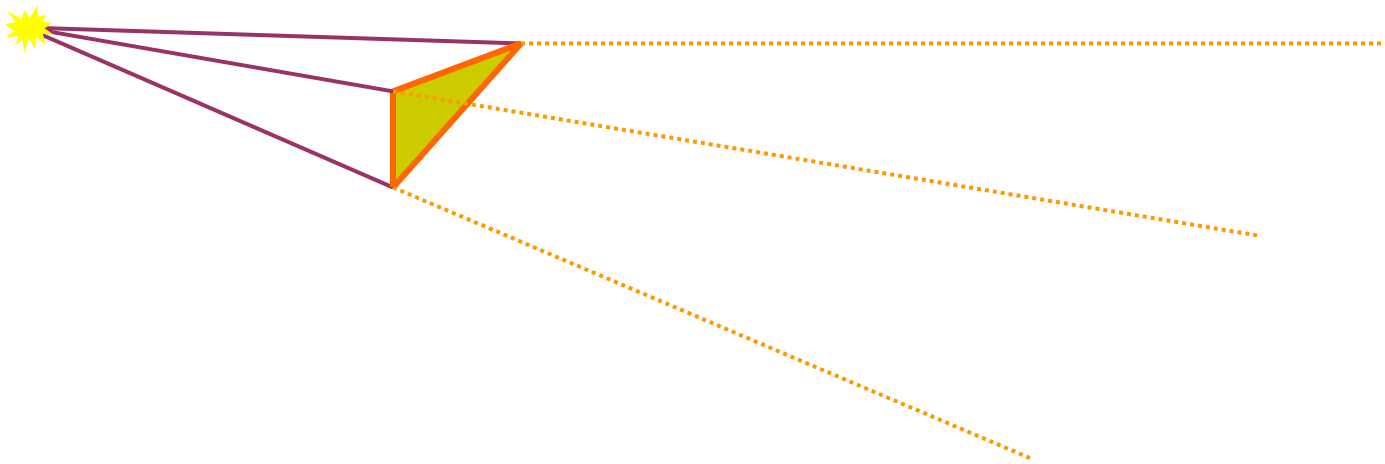


- Most popular method for real time
- Shadow volume concept

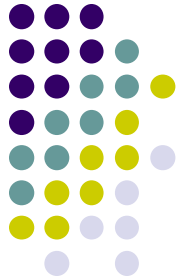


Shadow volumes

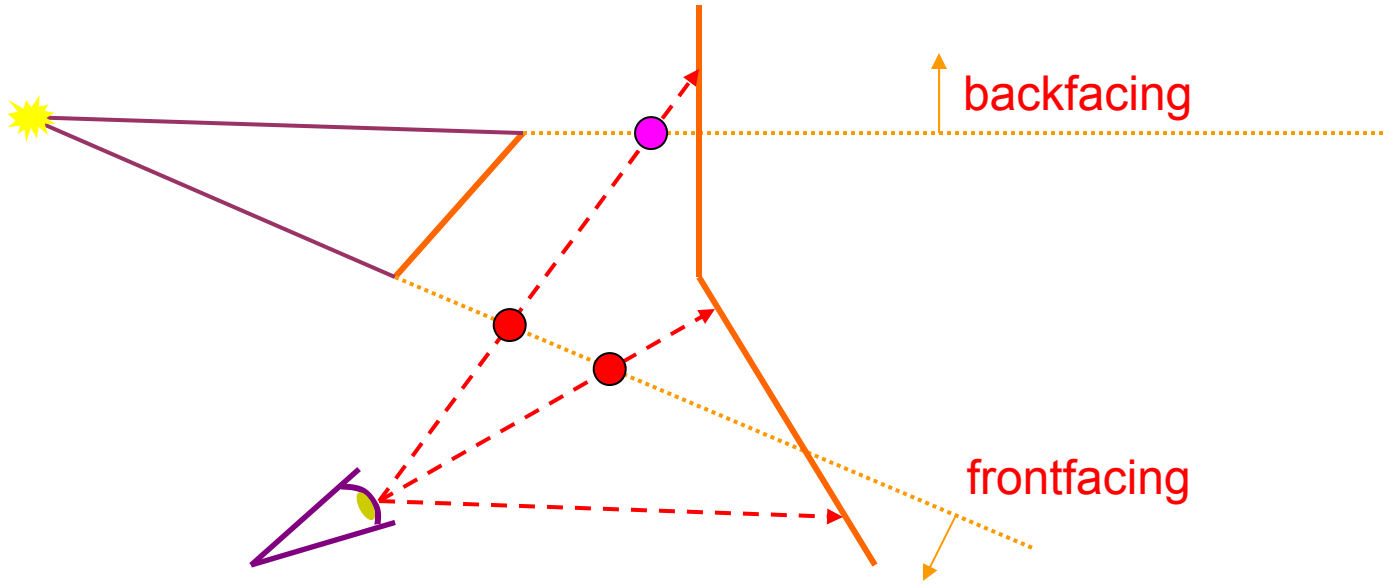
- Create volumes of space in shadow from each polygon in light
- Each triangle creates 3 projecting quads



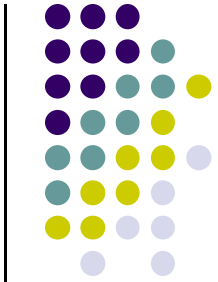
Using Shadow Volume



- To test a point, count the number of polygons between it and the eye.
- If we look through more frontfacing than backfacing polygons, then in shadow.

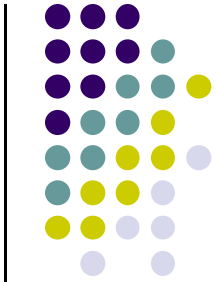


Shadow volume algorithm uses stencil buffer



- Stencil what?
- Is just another buffer (often 8 bits per pixel)
- When rendering to it, we can add, subtract, etc
- Then, the resulting image can be used to mask off subsequent rendering

How to implement Shadow volumes using Stencil Buffer

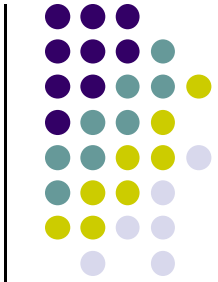


- A four pass process [Heidmann91]:
  - **1st Pass**: render the scene with just ambient lighting.
  - Turn off updating Z-buffer and writing to color buffer (i.e. Z-compare, draw to stencil only).
  - **2nd pass**: render front facing shadow volume polygons to stencil buffer, incrementing count.
  - **3rd pass**: render backfacing shadow volume polygons to stencil, decrementing.
  - **4th pass**: render diffuse and specular where stencil buffer is 0.

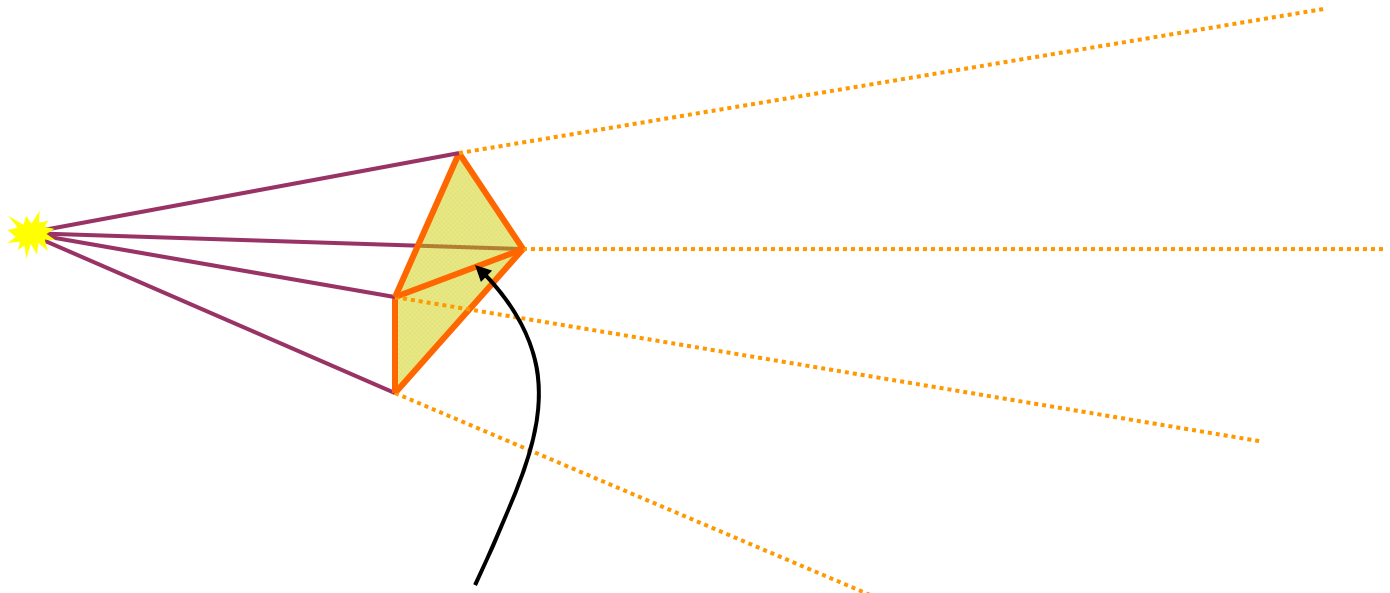
Shadow Volume Example

Image courtesy of NVIDIA Inc.

Merging Volumes

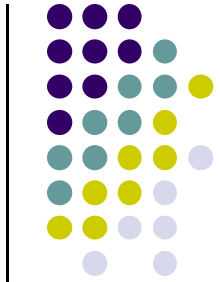


- Edge shared by two polygons facing the light creates front and backfacing quad.

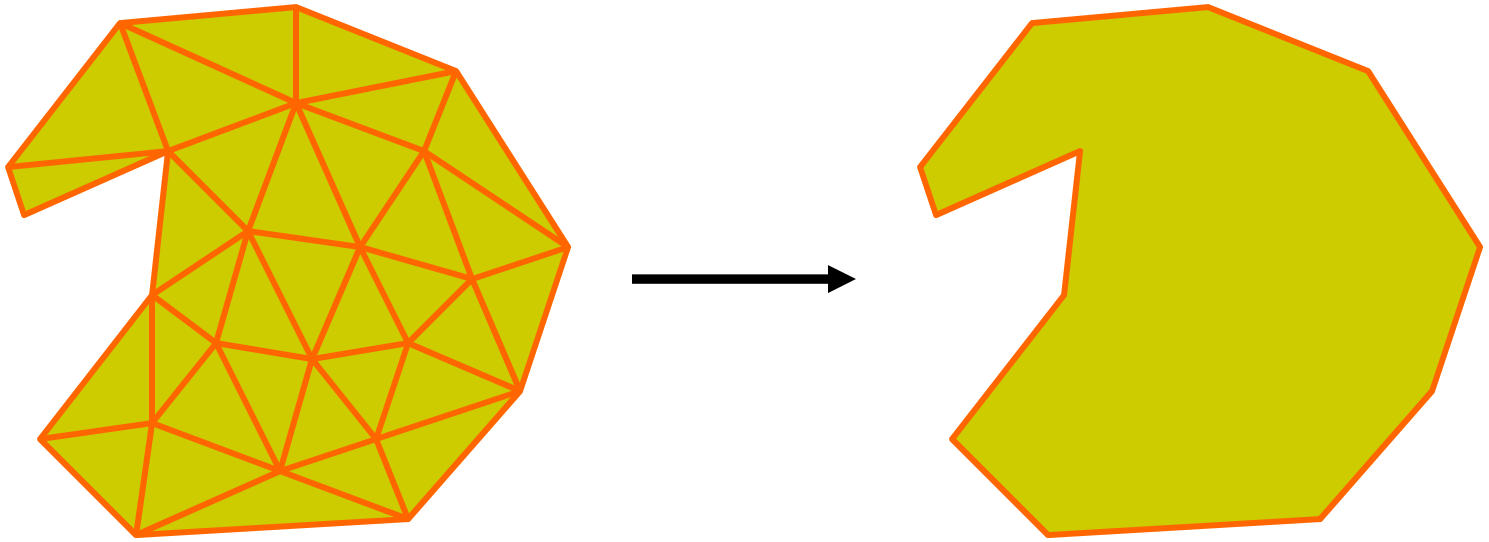


This interior edge makes
two quads, which cancel out

Silhouette Edges

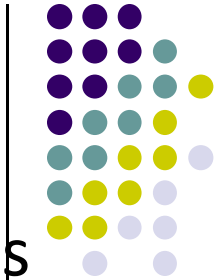


From the light's view, caster interior edges do not contribute to the shadow volume.



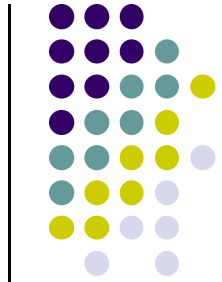
Finding the silhouette edge gets rid of many useless shadow volume polygons.

Ambient Occlusion

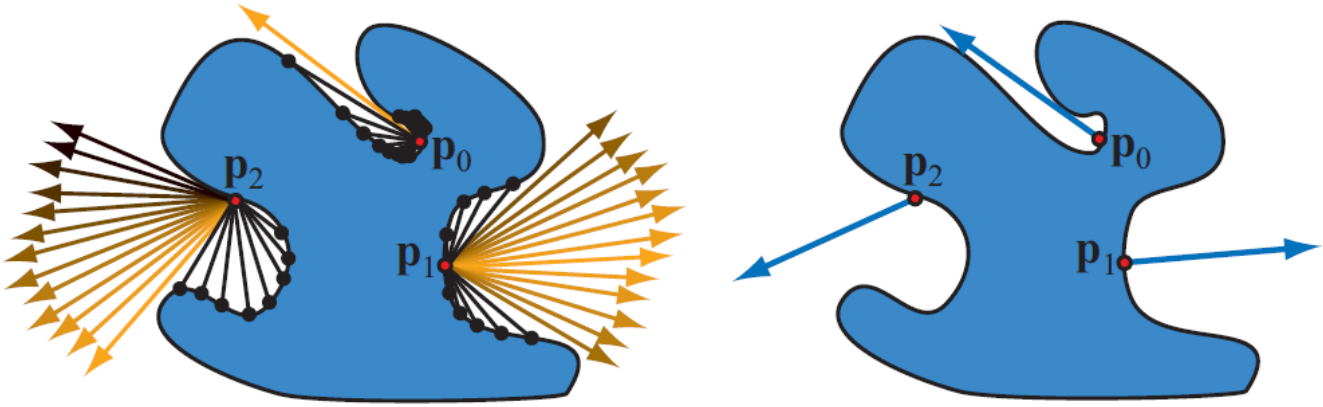


- Estimate what fraction of hemisphere above point is occluded (blocked)
- Render ambient term proportional to fraction of hemisphere occluded
- Result: areas deep in creases get darker ambient term

Ambient Occlusion Theory

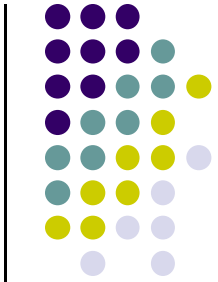


$$k_A(\mathbf{p}) = \frac{1}{\pi}\int_{\Omega} v(\mathbf{p},\mathbf{l})\cos\theta_i \, d\varpi_i$$



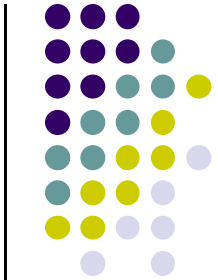
- $\mathbf{p_0}$ is less visible than $\mathbf{p_1}$ so has less ambient occlusion
- **Orientation matters: $\mathbf{p_1}$** is less occluded around its normal so higher ambient occlusion term

Ambient Occlusion

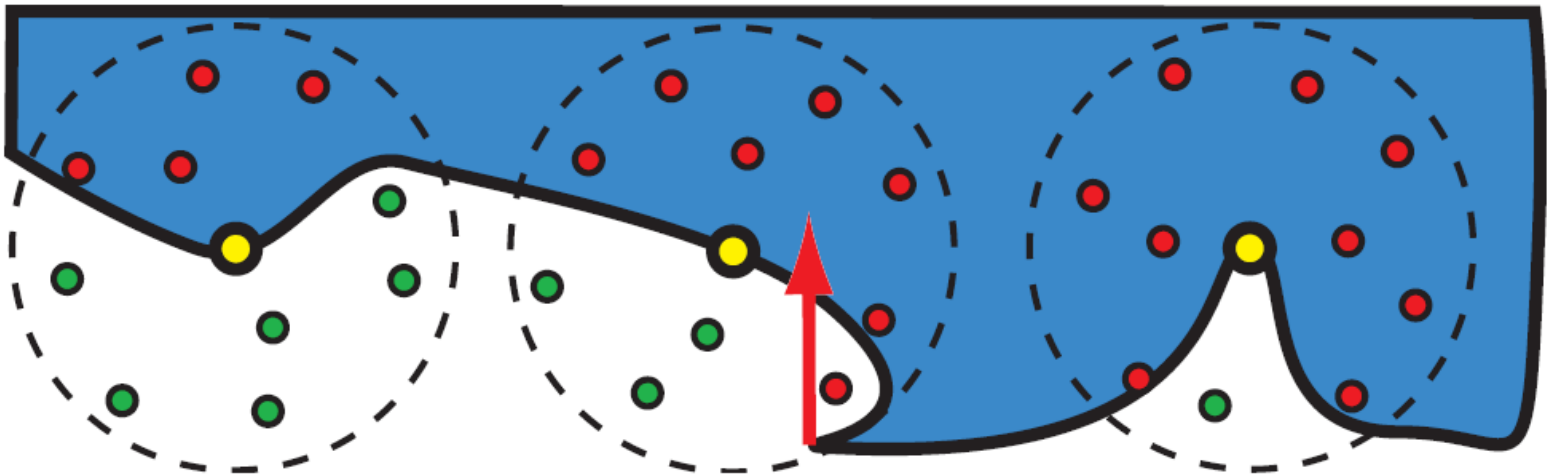


- To render, during lighting calculation simply multiply ambient term with ambient occlusion term
- Calculating ambient occlusion during vertex lighting has complexity proportional to scene complexity
- Crytek developed **Screen Space Ambient Occlusion (SSAO)** independent of screen complexity
- Screen space? Do in fragment shader not vertex shader

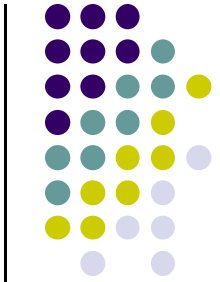
Screen Space Ambient Occlusion



- Take sample in circle around pixel center (yellow dot)
- **Red dot:** sample's z value behind pixel center's z
- **Green dot:** sample's z value in front of pixel center's z
- Ambient Occlusion term: No. green dots/ total samples
- Example: circle on left has ambient occlusion of 6/10



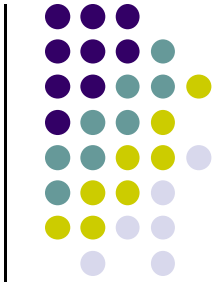
Planar reflections



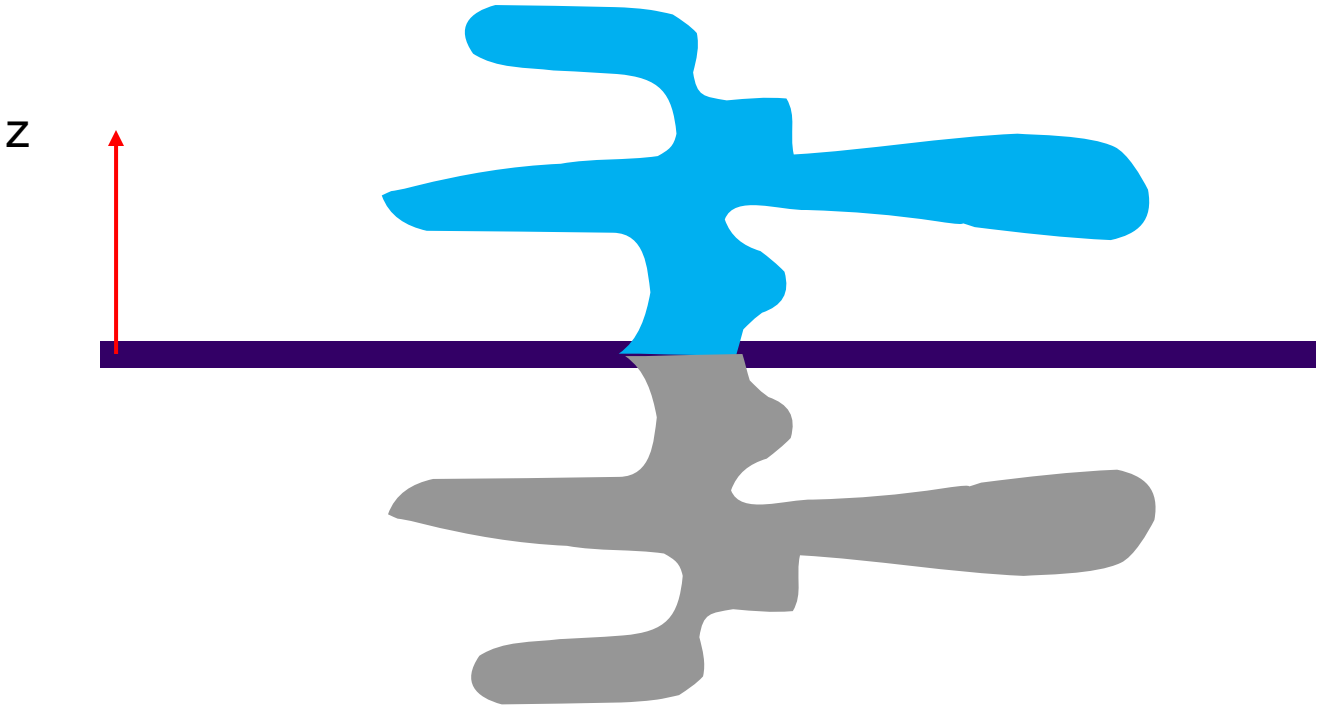
- We've already done reflections on far objects using environment mapping
- Does not work for nearby planar surfaces
- Planar reflections are important, because they too give clues about spatial relationships and increases realism

- Based on law of reflection:
  - Incoming angle is equal to outgoing angle

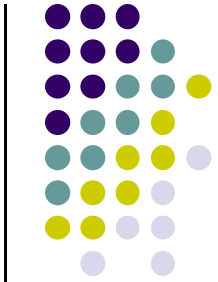
Planar reflections



- To reflect about plane z=0
- Then apply scaling matrix s**`cale(1,1,-1);`**
- Effect:



Planar reflections



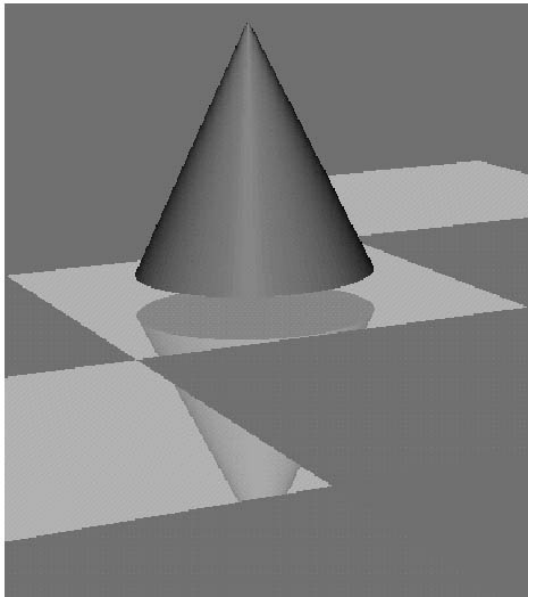
- How should you render?

1) the ground plan polygon into the stencil buffer

2) then render scaled (1,1,-1) model, but mask with stencil buffer

3) the ground plane (semi-transparent)

4) the unscaled model



Another example

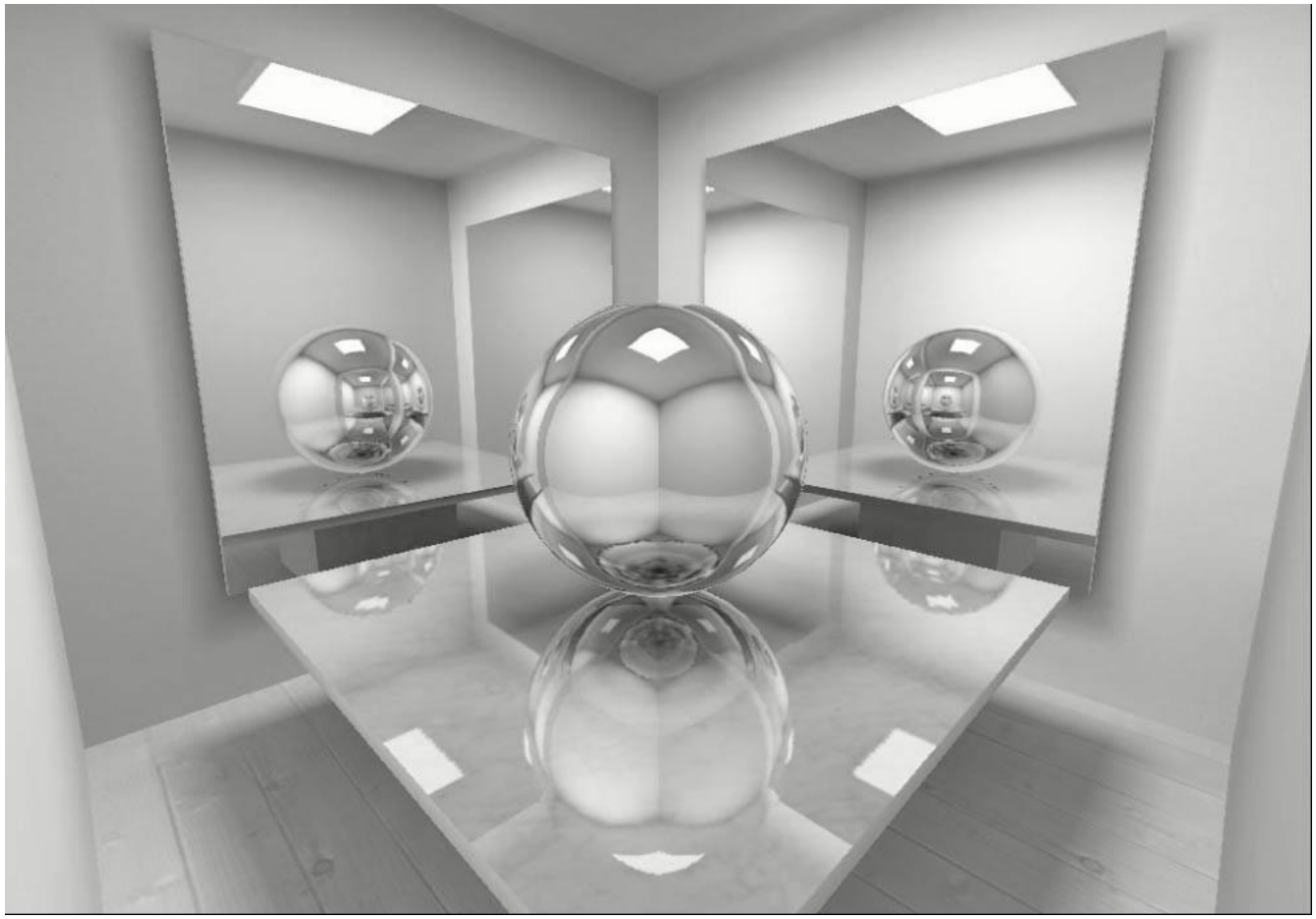
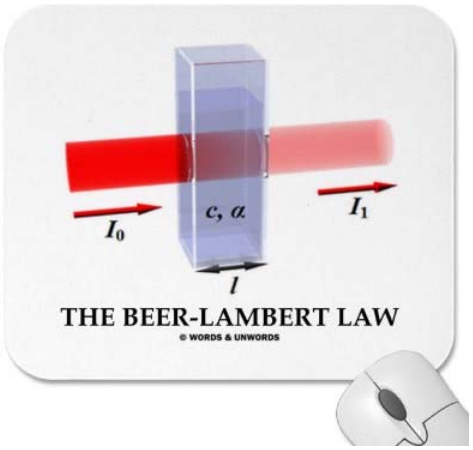


Image courtesy of Kasper Hoey Nielsen

Transmittance

- Transparent objects transmit light
- To render transparent objects compute
  - α x object color + (1 - α) x (color of objects behind)
- For objects of varying thickness use Beer's Law

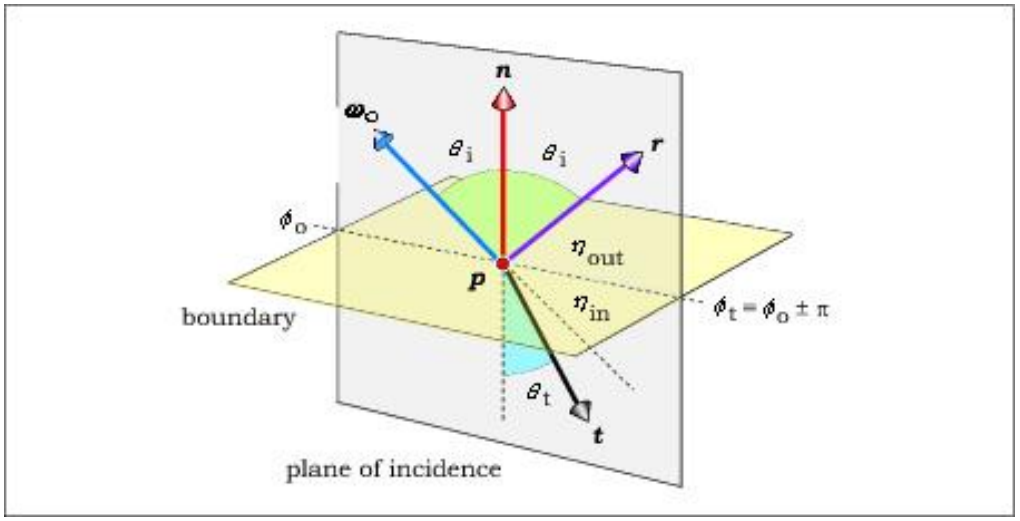
$$T = e^{-\alpha' cd}$$



- α' is absorption coefficient of material
- *c* is concentration of absorbing material
- *d* is distance travelled through the medium

Glass Rendered using Beer's Law

Refraction



ω_o: incident ray
r: reflected ray
t: transmitted ray
Θ_i : angle of incidence
Θ_t : angle of refraction

Classic approach to refraction uses Snell's law:

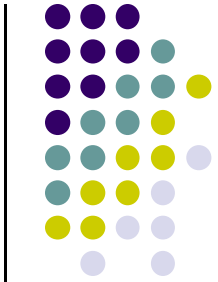
$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_{in}}{\eta_{out}} = \eta$$

$$\boldsymbol{t} = \frac{1}{\eta}\omega_o - \left(\cos\theta_t - \frac{1}{\eta}\cos\theta_i\right)\boldsymbol{n}$$

$$\cos\theta_t = \left[1 - \frac{1}{\eta^2}(1 - \cos^2\theta_i)\right]^{1/2}$$

$$\cos\theta_i = \boldsymbol{n}\cdot\boldsymbol{\omega_o}$$

Refraction



- Bec presents a more efficient method for refraction

$$\mathbf{t} = (\omega - k)\mathbf{N} - n\mathbf{L}$$

- Where $n = n_1 / n_2$ is the relative index of refraction and

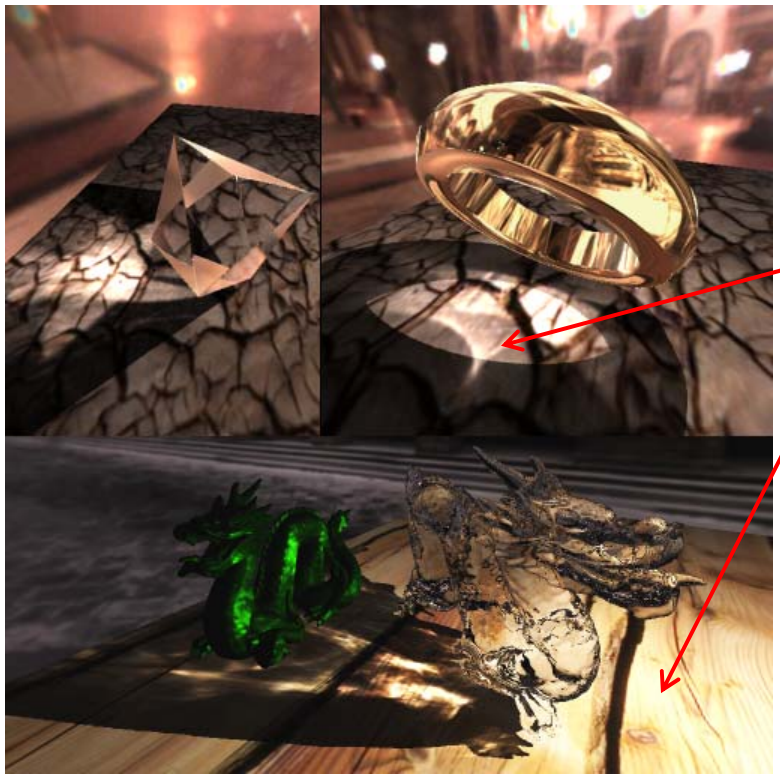
$$\omega = n(\mathbf{L} \bullet \mathbf{N})$$

$$k = \sqrt{1 + (\omega - n)(\omega + n)}$$

Refraction

Courtesy Chris Wyman, Univ Iowa

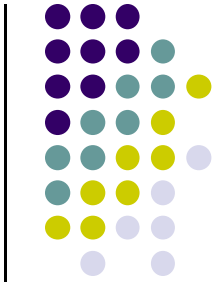
Caustics



Caustics occur when light is focussed on diffuse surface

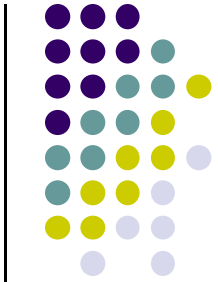
Courtesy Chris Wyman, Univ Iowa

Rendering Caustics



- Several object-space and image-space techniques proposed
- Popular image-space techniques by Wyman
- **Render pass 1:** Render photons from light source, store where they land in **photon buffer**
- **Render pass 2:** Transform points in photon buffer to eye's viewpoint and render to **caustic map**
- **Render pass 3:** project caustic map onto scene and use to illuminate pixels

Sub-Surface Scattering



Crysis skin demo

Marble

Human Skin

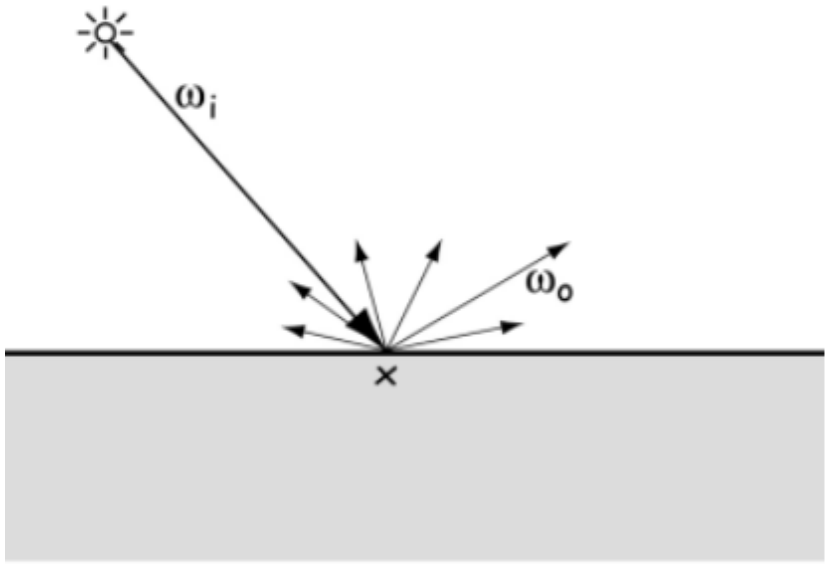
More Examples…

Leaves

Hair

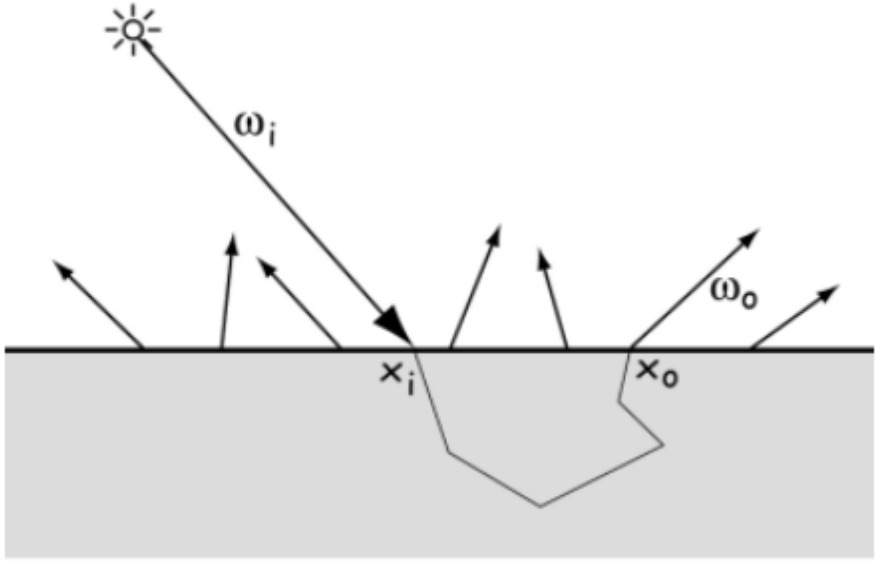
Milk

Subsurface Scattering



$$f_r(x,\omega_i,\omega_o) \equiv \frac{dL_r(x,\omega_o)}{dE_i(x,\omega_i)}$$

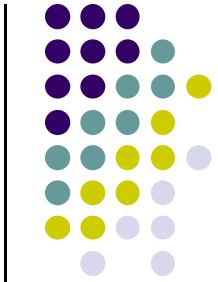
Reflection



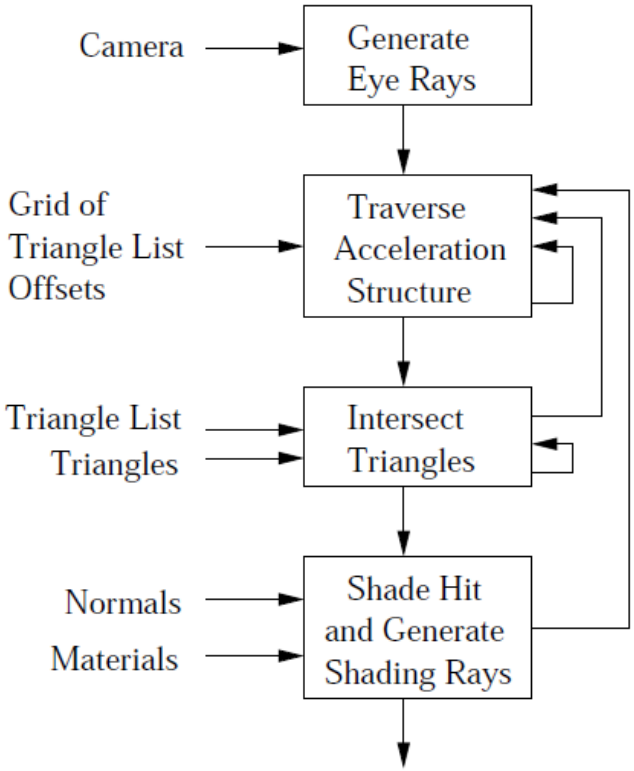
$$S(x_i,\omega_i;x_o,\omega_o) \equiv \frac{dL_r(x_o,\omega_o)}{d\Phi_i(x_i,\omega_i)}$$

Subsurface Scattering

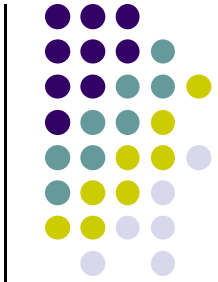
Real-Time Global Illumination



- For real-time GI on the fly, need real time GI algorithm
- One option: Make classic GI techniques real time (already discussed)
  - Real-time **ray tracing**
  - Real-time **photon mapping**
  - Real-time **radiosity**
- **General idea:** Use multiple shaders

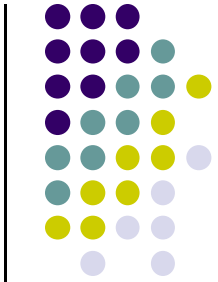


Pre-Computed Global Illumination



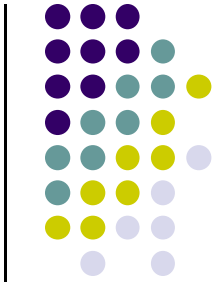
- Real-time RT/photon mapping still not 30 FPS
- Can pre-compute parts of rendering, faster rendering
- Hence, **pre-computed global illumination** (also already discussed)
  - Pre-compute **Global Illumination**
    - Lights objects mostly static
    - Use GPU to pre-compute approximate lighting solutions
    - Speeds up run-time
  - Pre-computed **Occlusion** (ambient occlusion)
  - Pre-computed **Radiance Transfer** (reflections)
    - Use spherical harmonics

New Real-Time GI Algorithms



- Emerging brand new real time GI approaches
- Designed to run fast on graphics hardware/shaders
- Two examples:
  - Light Propagation Volumes (LPV)
  - Screen Space Directional Occlusion (SSDO)
- Will be studied in next lectures

References



- Kutulakos K, CSC 2530H: Visual Modeling, course slides
- UIUC CS 319, Advanced Computer Graphics Course
- David Luebke, CS 446, U. of Virginia, slides
- Chapter 1-4 of RT Rendering
- CS 543/4731 course slides