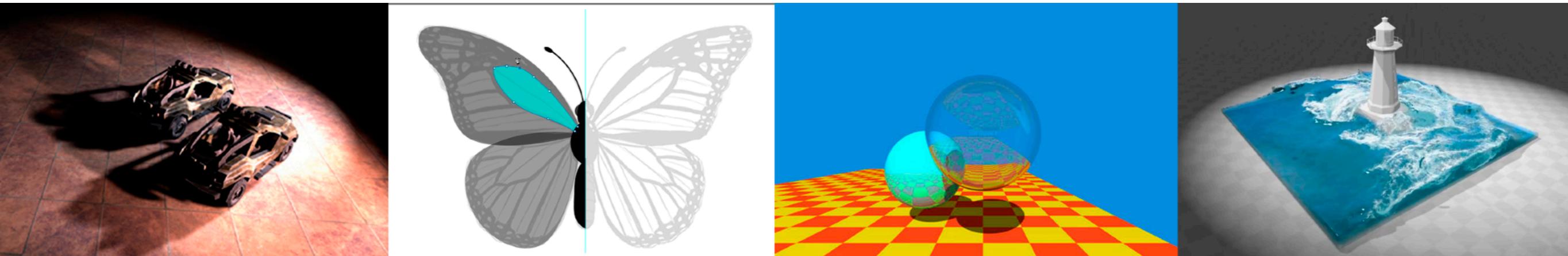


# Introduction to Computer Graphics

GAMES101, Lingqi Yan, UC Santa Barbara

## Lecture 7: Shading 1 (Illumination, Shading and Graphics Pipeline)



# Announcements

- Homework 1
  - 300+ submissions
  - Will start TA recruiting (from existing applications) soon
- Homework 2 will be out today
  - About Z-buffering
  - Much easier than HW1
- May need an additional lecture for shading

# Last Lectures

- Rasterization
  - Rasterizing **one triangle**
  - Sampling theory
  - Antialiasing

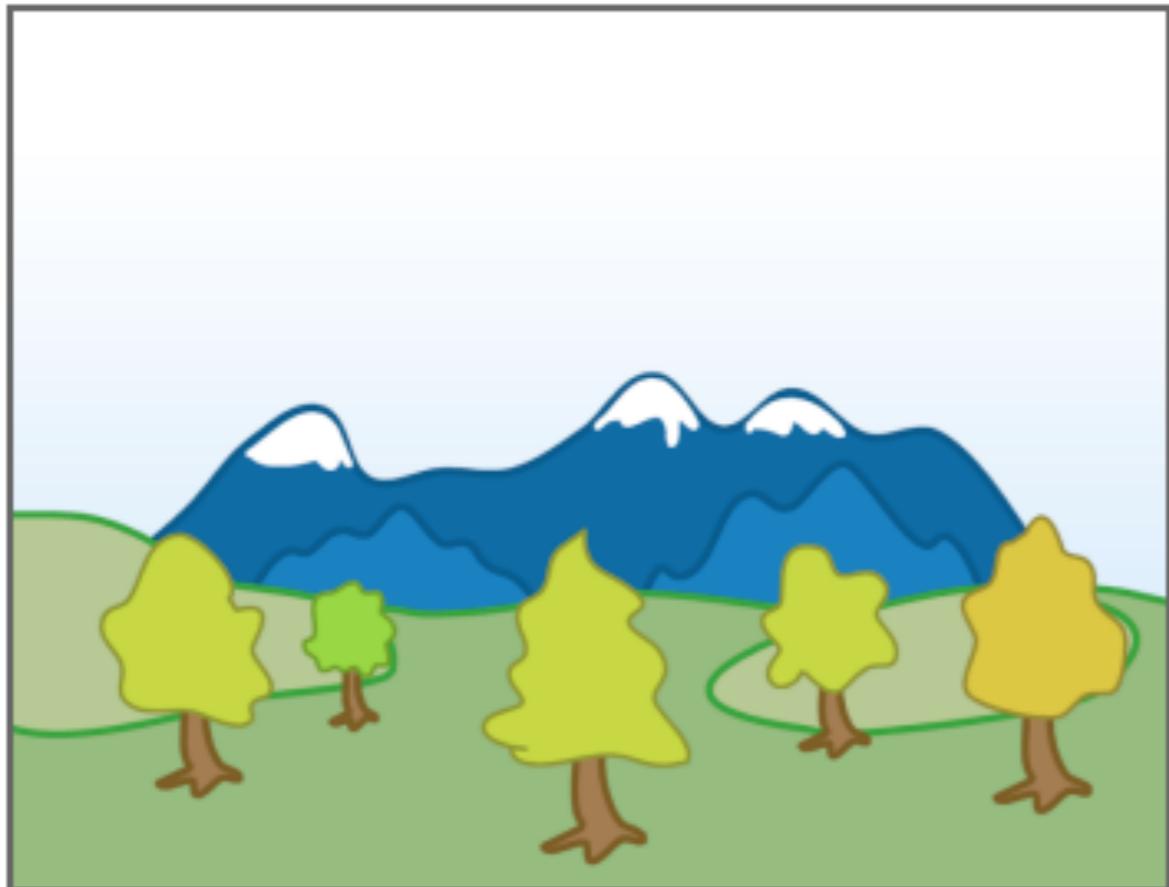
# Today

- **Visibility / occlusion**
  - Z-buffering
- Shading
  - Illumination & Shading
  - Graphics Pipeline

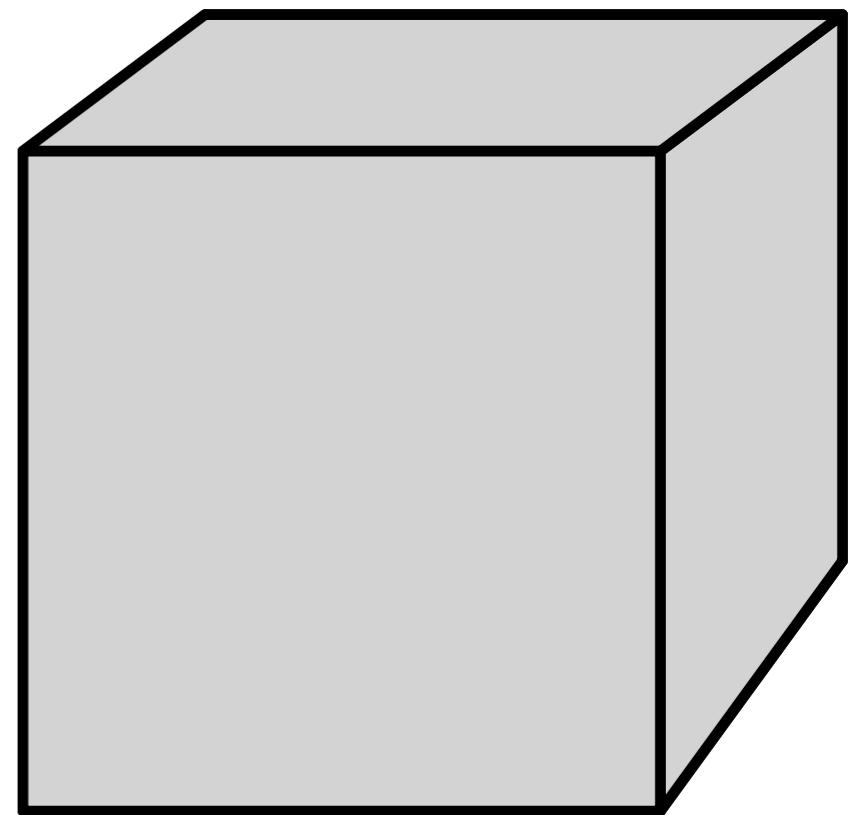
# Painter's Algorithm

Inspired by how painters paint

Paint from back to front, **overwrite** in the framebuffer



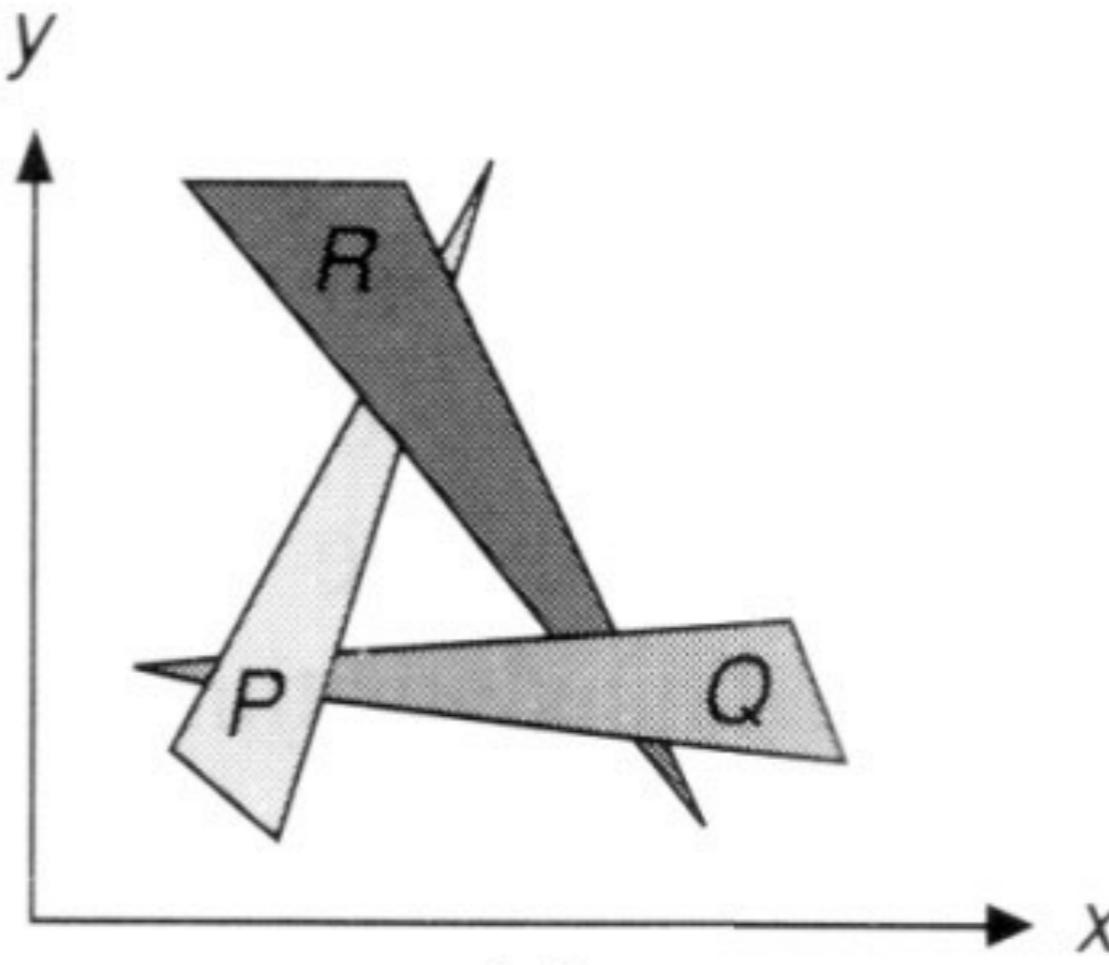
[Wikipedia]



# Painter's Algorithm

Requires sorting in depth ( $O(n \log n)$  for  $n$  triangles)

Can have unresolvable depth order



# Z-Buffer

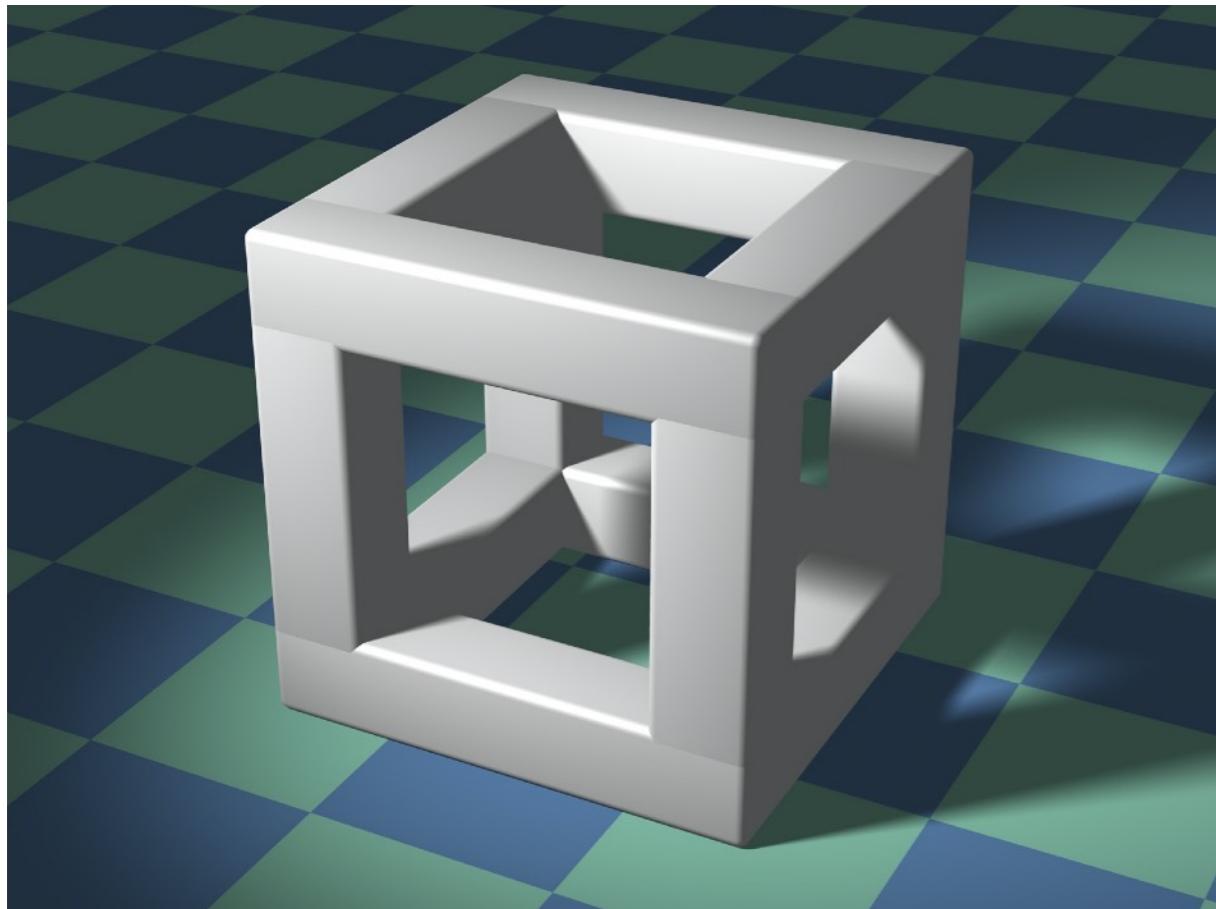
This is the algorithm that eventually won.

Idea:

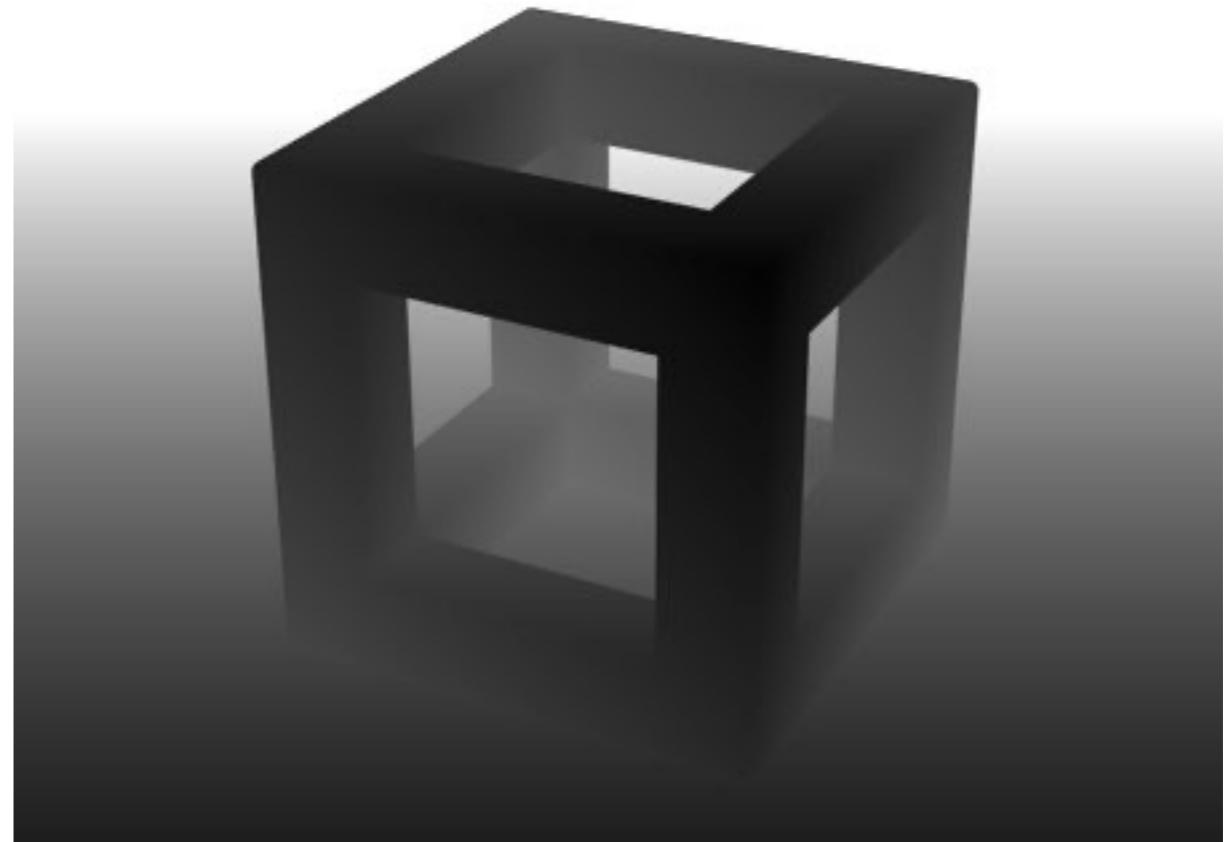
- Store current min. z-value **for each sample (pixel)**
- Needs an additional buffer for depth values
  - frame buffer stores color values
  - depth buffer (z-buffer) stores depth

IMPORTANT: For simplicity we suppose  
***z is always positive***  
(smaller  $z \rightarrow$  closer, larger  $z \rightarrow$  further)

# Z-Buffer Example



Rendering



Depth / Z buffer

Image source: Dominic Alves, flickr.

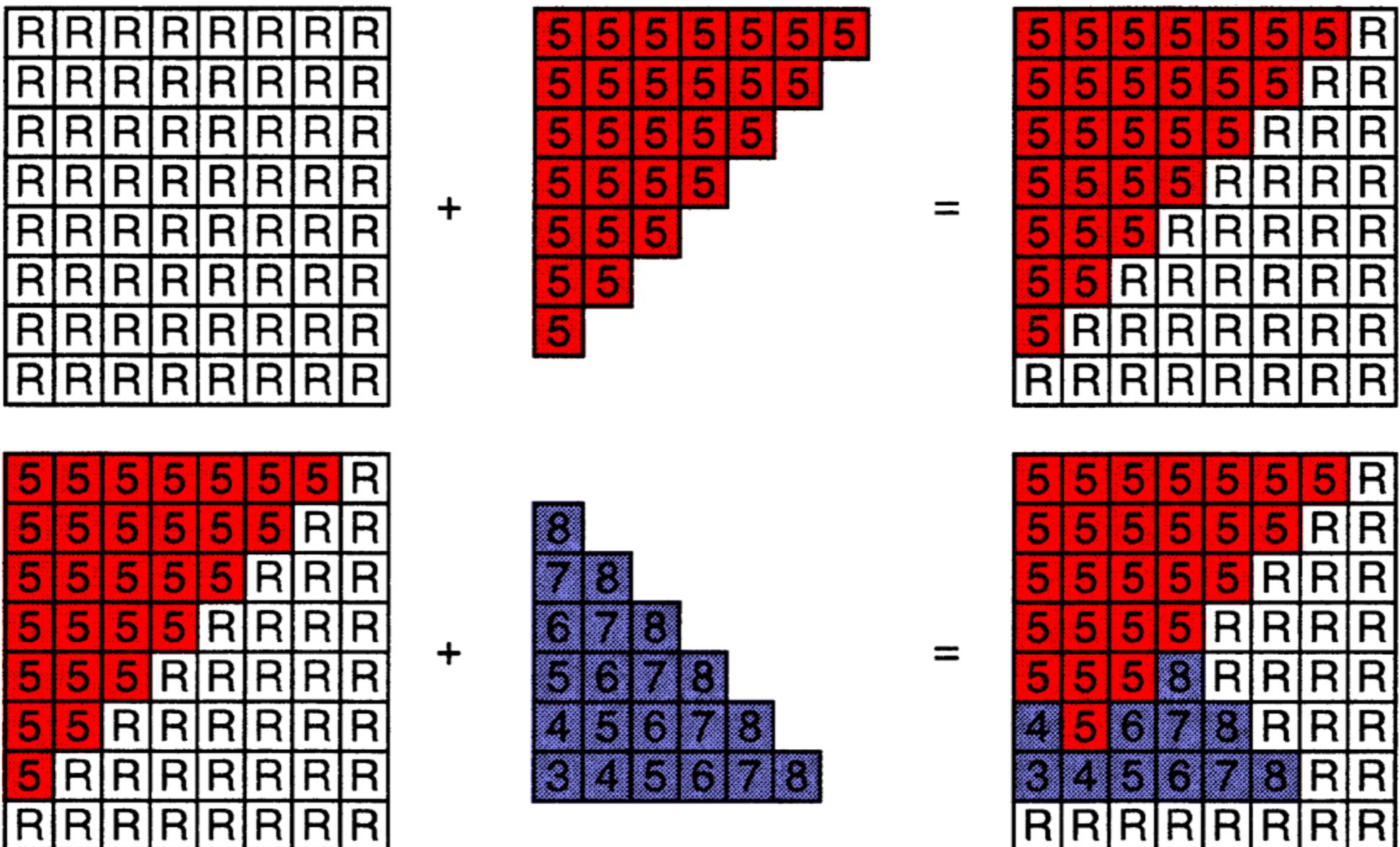
# Z-Buffer Algorithm

Initialize depth buffer to  $\infty$

During rasterization:

```
for (each triangle T)
    for (each sample (x,y,z) in T)
        if (z < zbuffer[x,y])                // closest sample so far
            framebuffer[x,y] = rgb;           // update color
            zbuffer[x,y] = z;                 // update depth
        else
            ;                                // do nothing, this sample is occluded
```

# Z-Buffer Algorithm



# Z-Buffer Complexity

## Complexity

- $O(n)$  for  $n$  triangles (assuming constant coverage)
- How is it possible to sort  $n$  triangles in linear time?

Drawing triangles in different orders?

Most important visibility algorithm

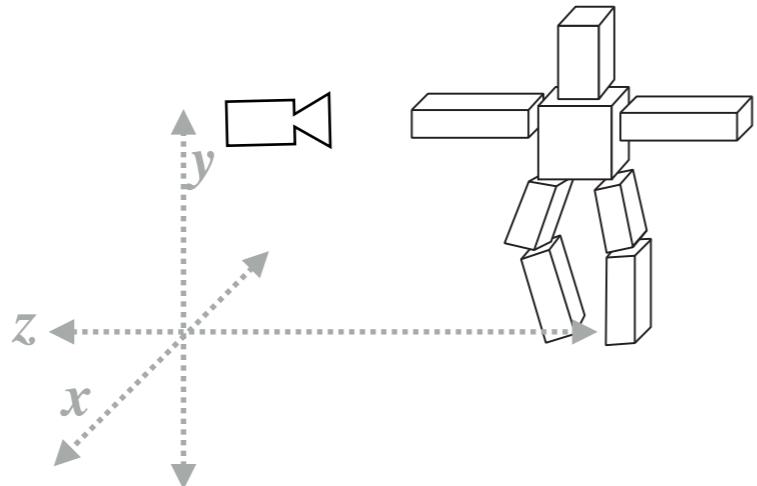
- Implemented in hardware for all GPUs

# Questions?

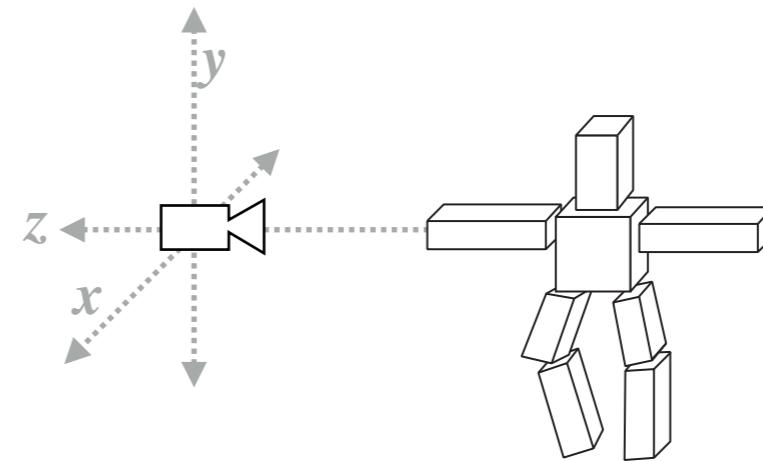
# Today

- Visibility / occlusion
  - Z-buffering
- Shading
  - Illumination & Shading
  - Graphics Pipeline

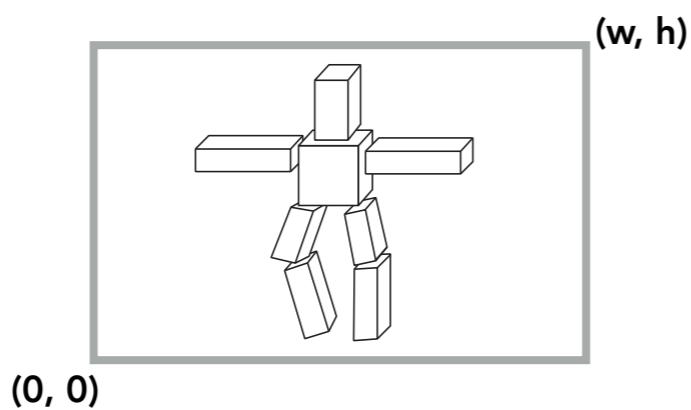
# What We've Covered So Far



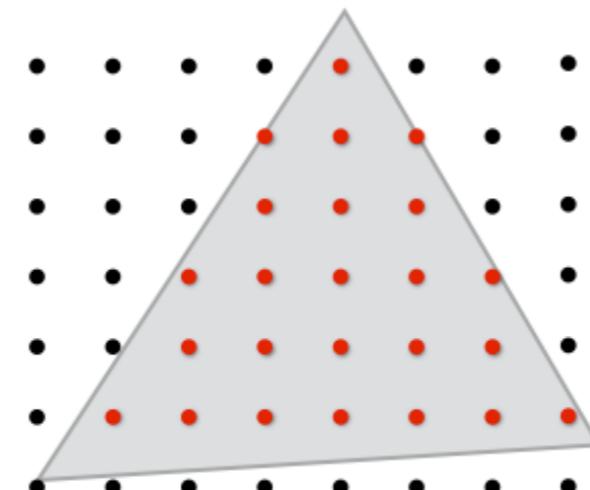
**Position objects and the camera in the world**



**Compute position of objects relative to the camera**

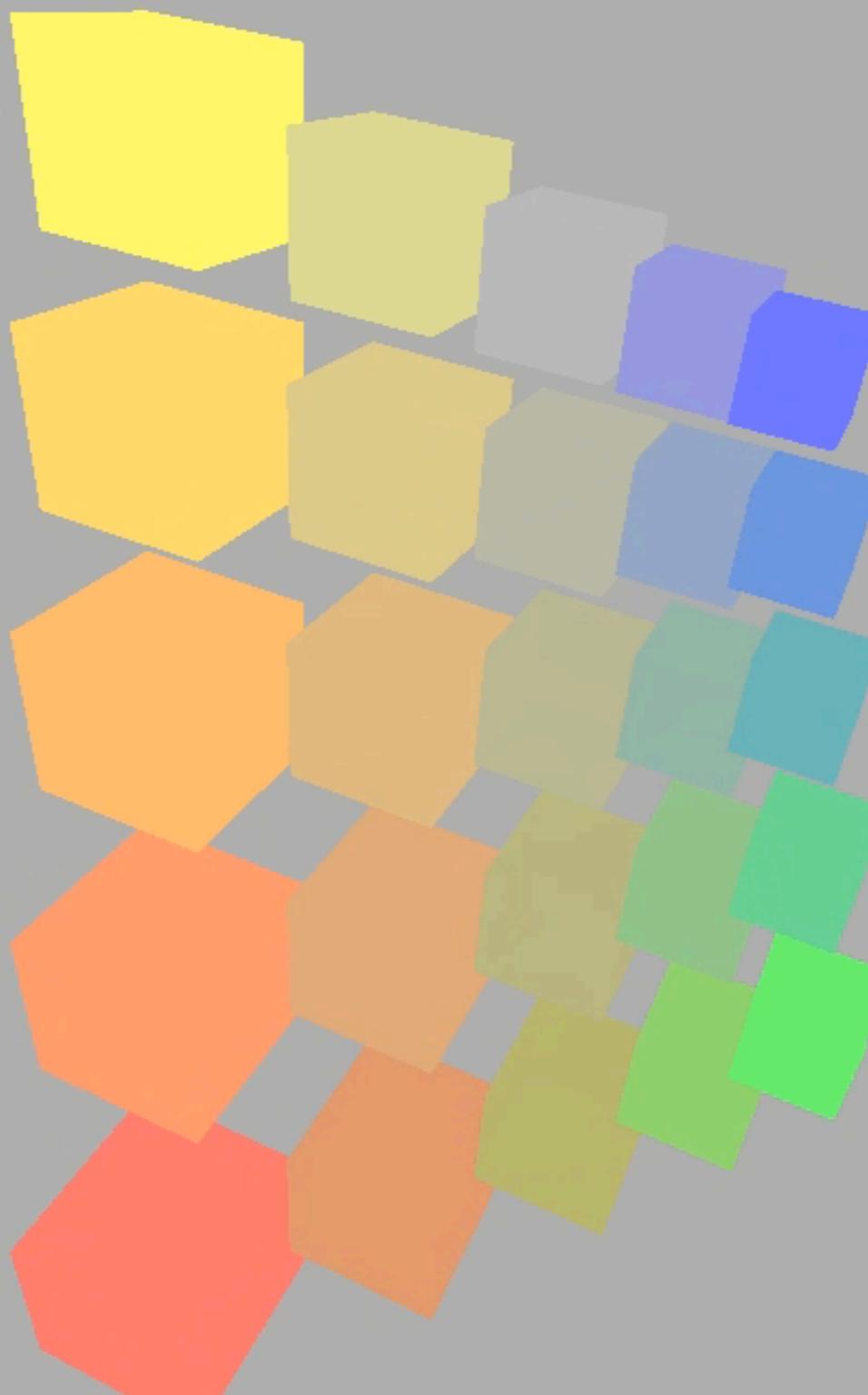


**Project objects onto the screen**

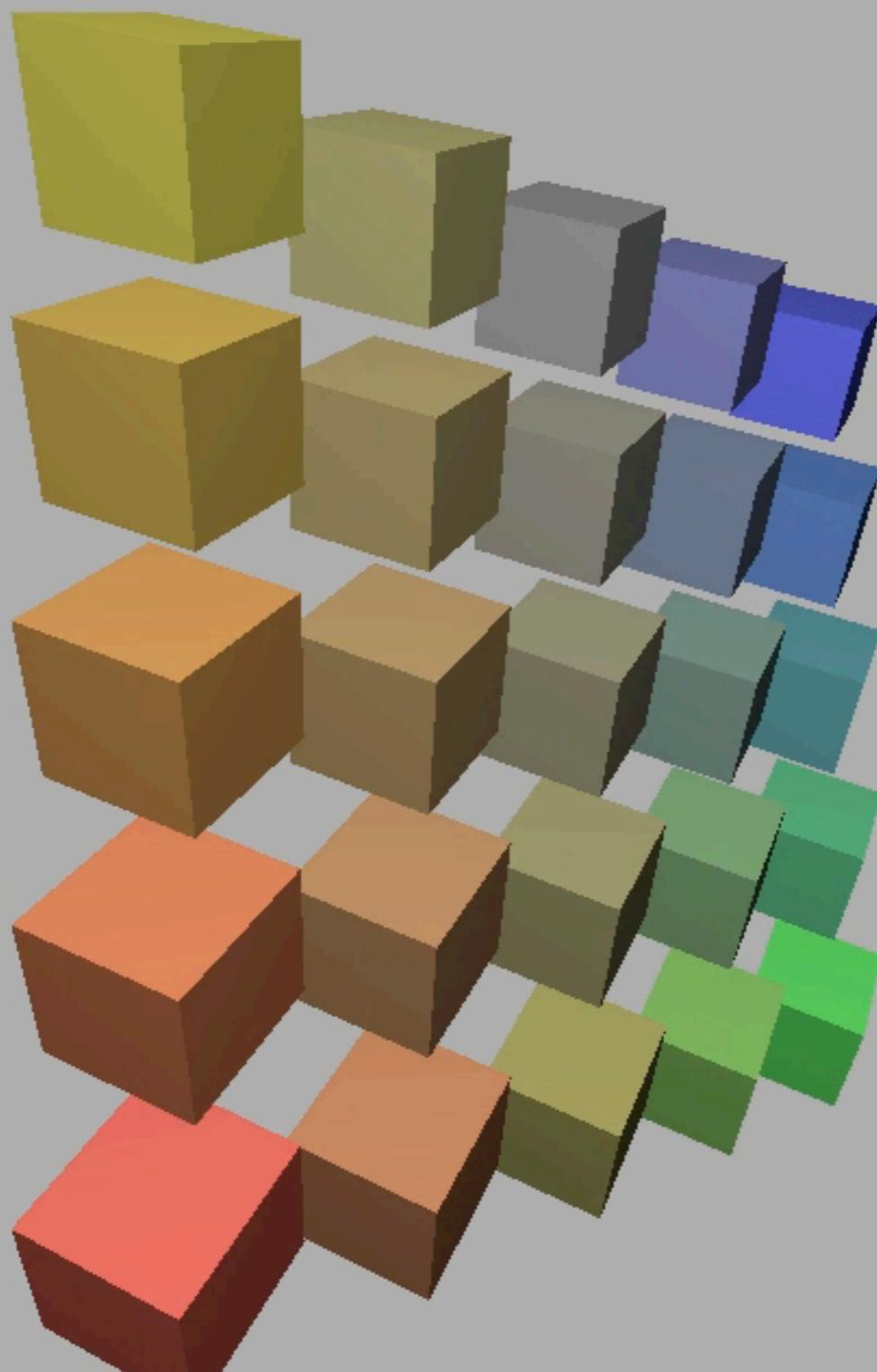


**Sample triangle coverage**

# Rotating Cubes (Now You Can Do)



# Rotating Cubes (Expected)



# What Else Are We Missing?



Credit: Bertrand Benoit. "Sweet Feast," 2009. [Blender /VRay]

# Shading

# Shading: Definition

- \* In Merriam-Webster Dictionary

**shad·ing**, ['ʃeɪdɪŋ], noun

The darkening or coloring of an illustration or diagram with parallel lines or a block of color.

- \* In this course

The process of **applying a material** to an object.

# A Simple Shading Model (Blinn-Phong Reflectance Model)

# Perceptual Observations

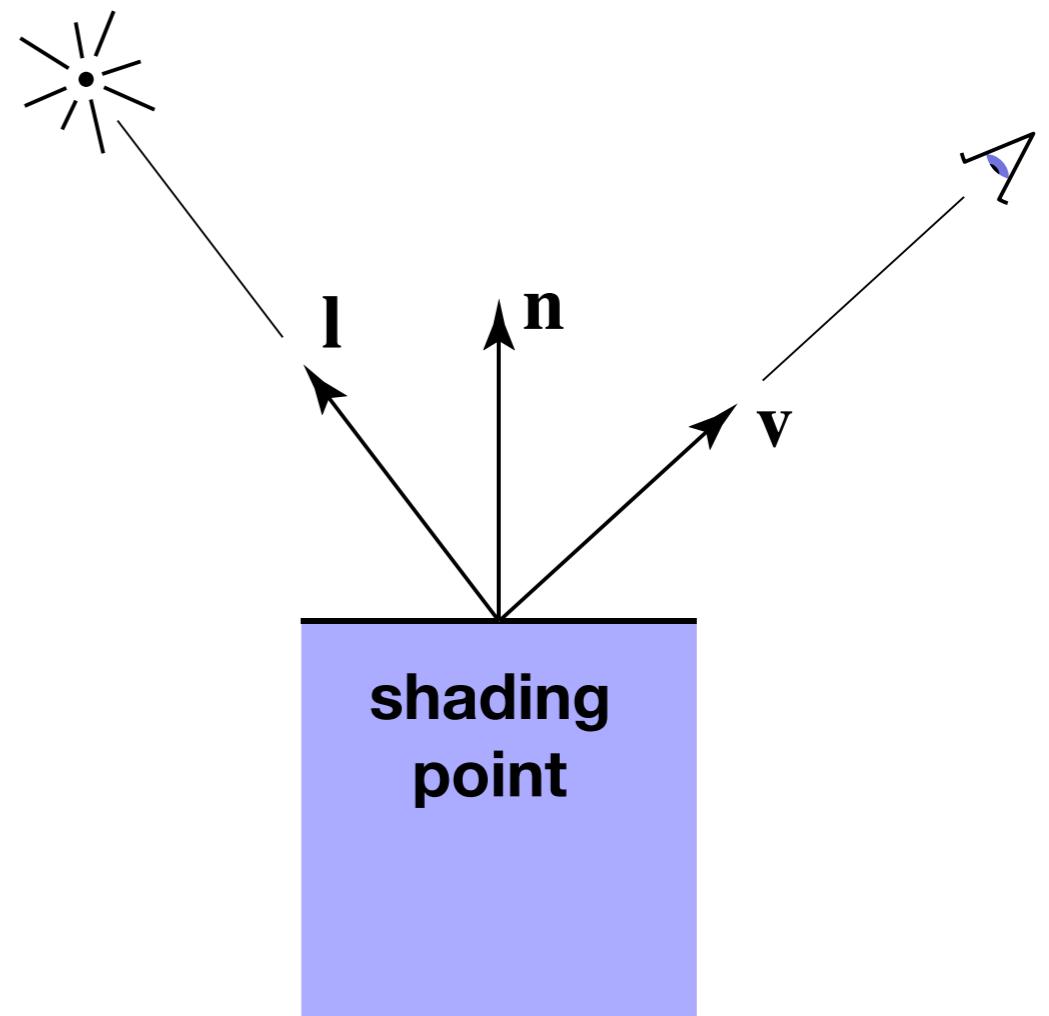


# Shading is Local

Compute light reflected toward camera  
at a specific **shading point**

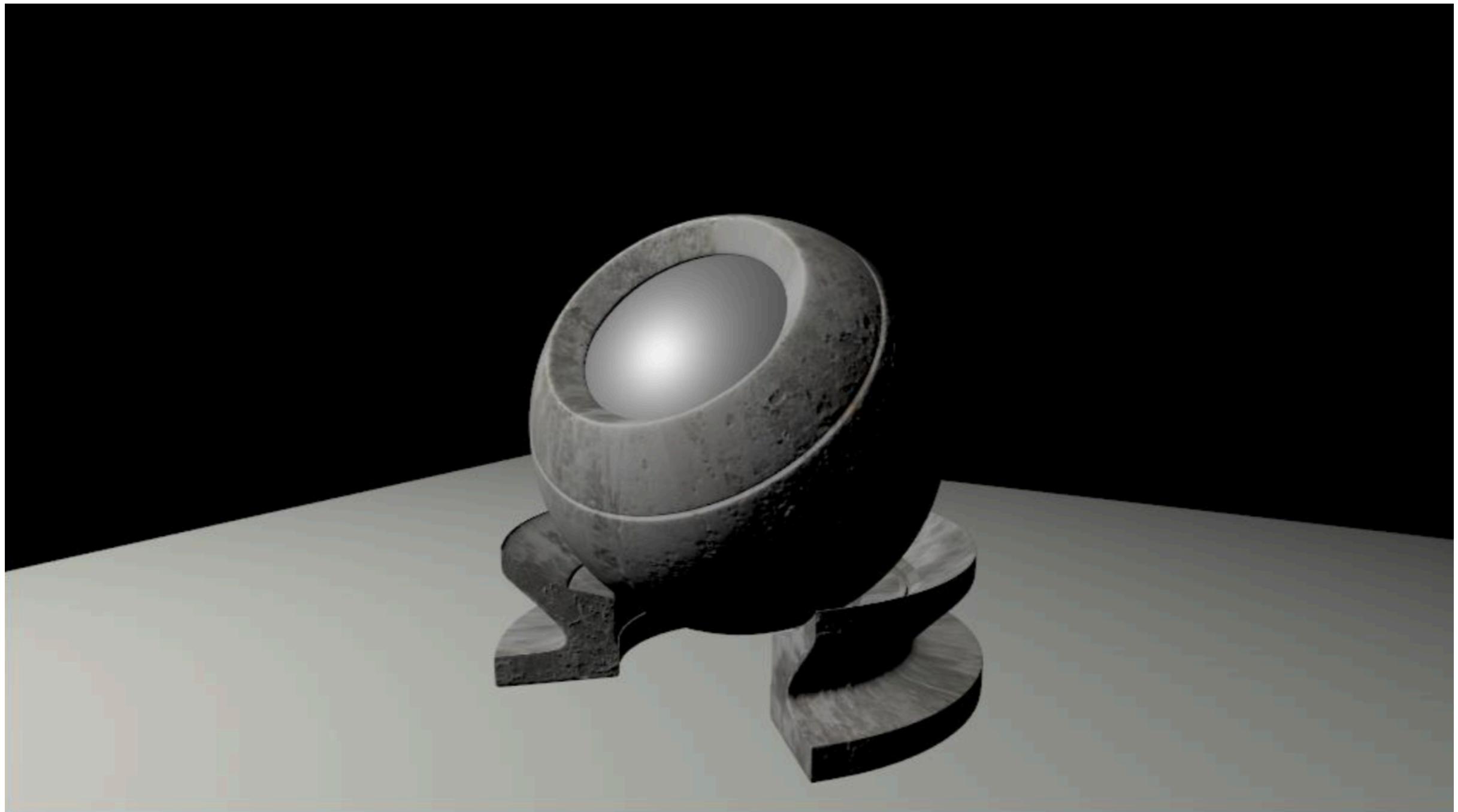
Inputs:

- Viewer direction,  $v$
- Surface normal,  $n$
- Light direction,  $l$   
(for each of many lights)
- Surface parameters  
(color, shininess, ...)



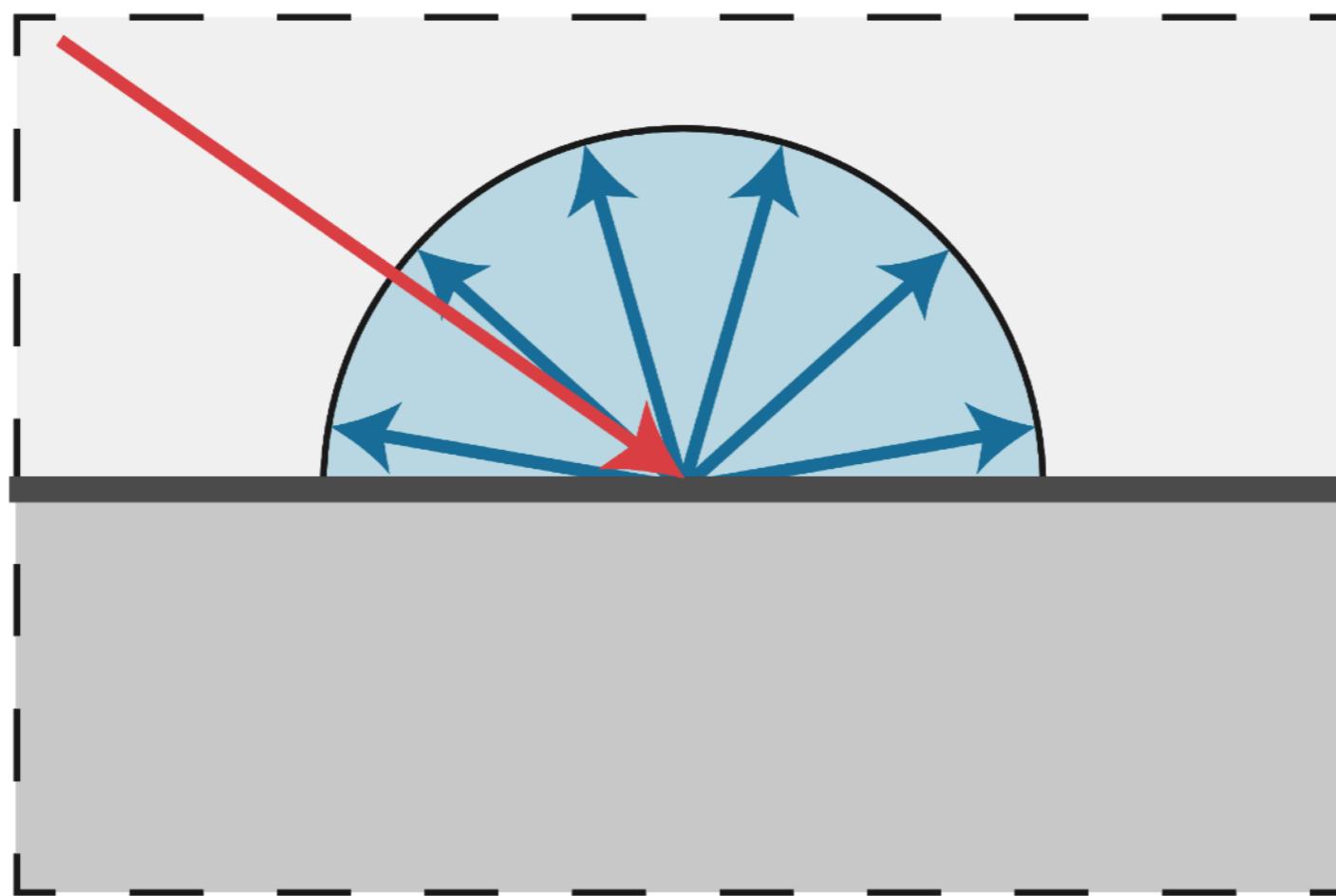
# Shading is Local

No shadows will be generated! (**shading ≠ shadow**)



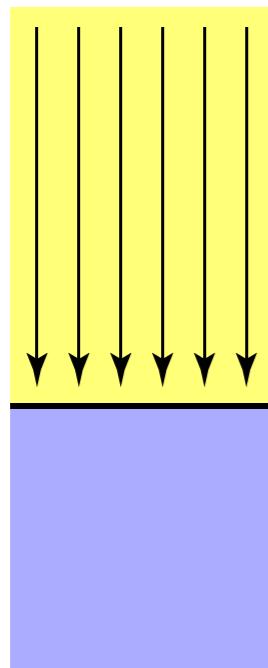
# Diffuse Reflection

- Light is scattered uniformly in all directions
  - Surface color is the same for all viewing directions

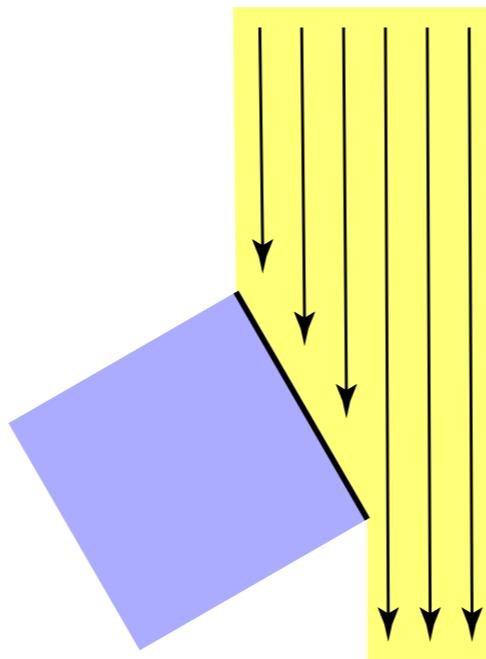


# Diffuse Reflection

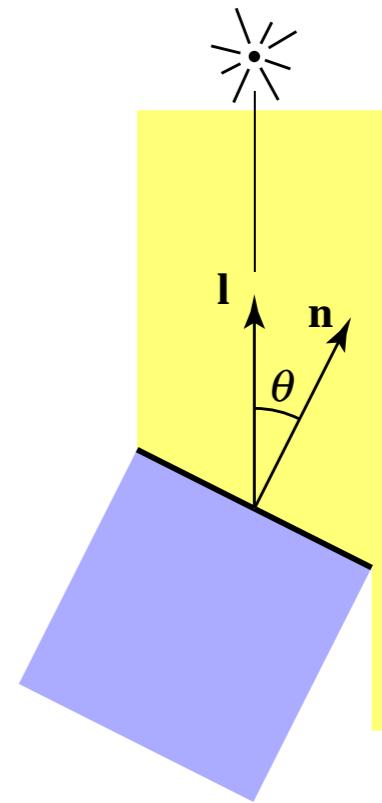
- But how much light (energy) is received?
  - Lambert's cosine law



Top face of cube receives a certain amount of light

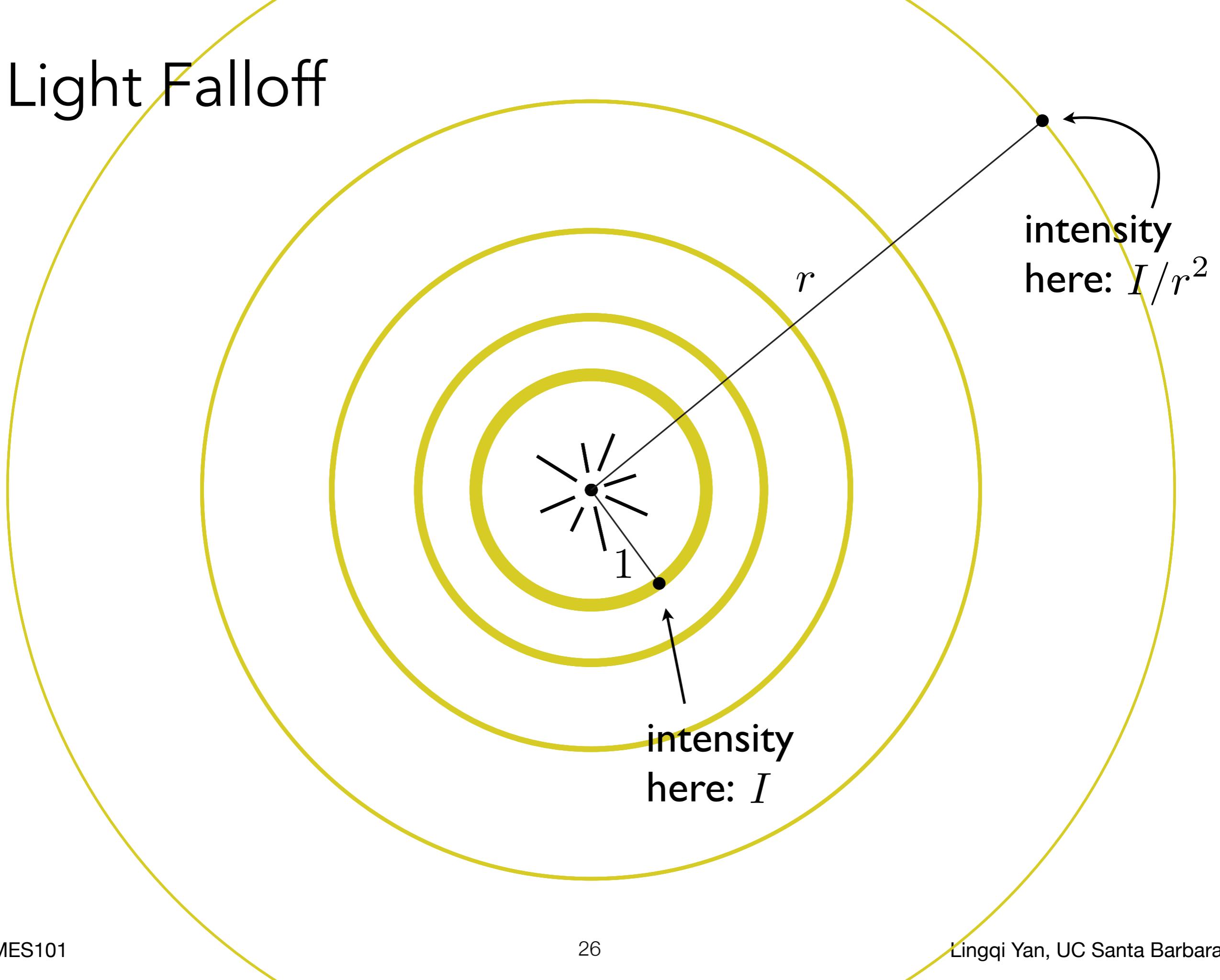


Top face of 60° rotated cube intercepts half the light



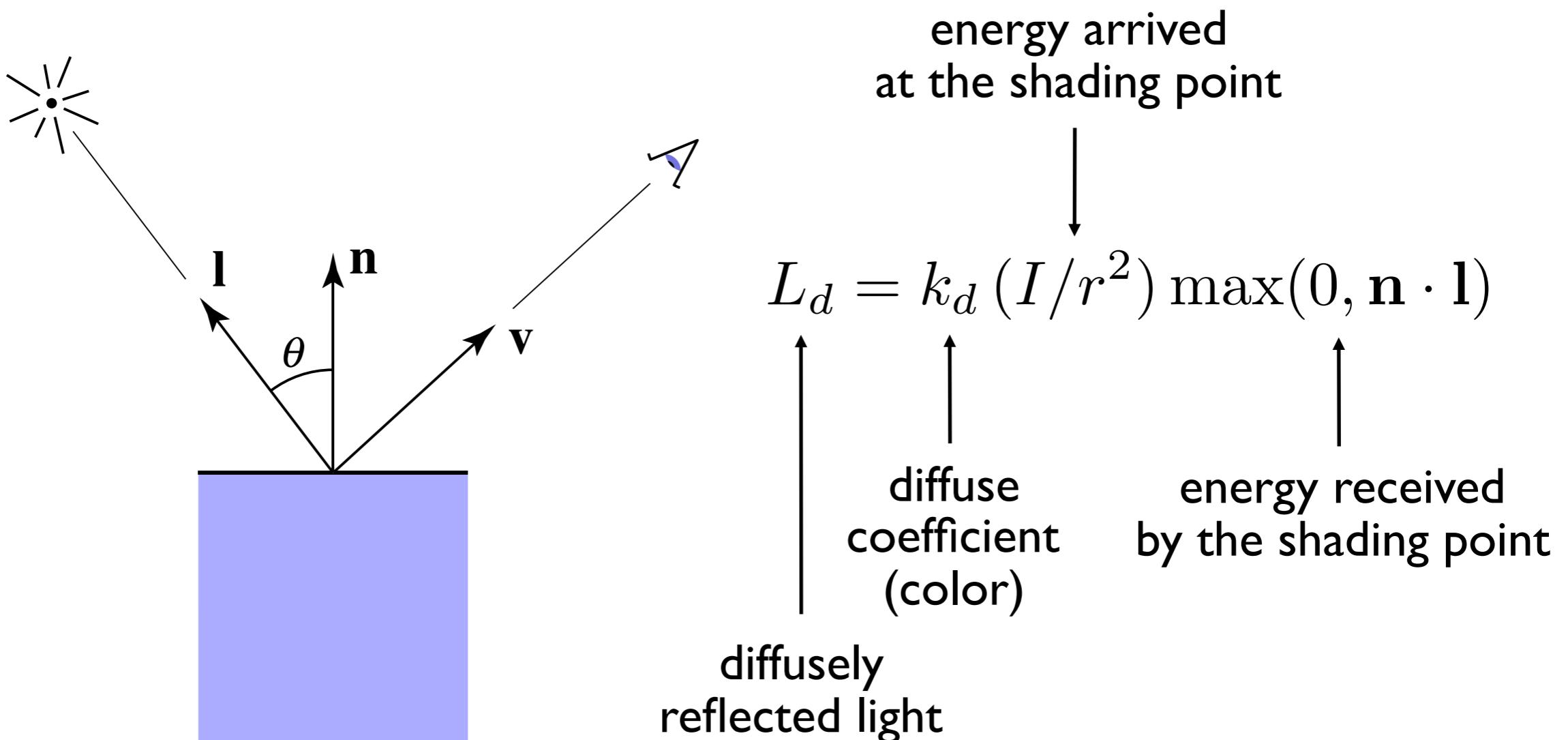
In general, light per unit area is proportional to  $\cos \theta = I \cdot n$

# Light Falloff



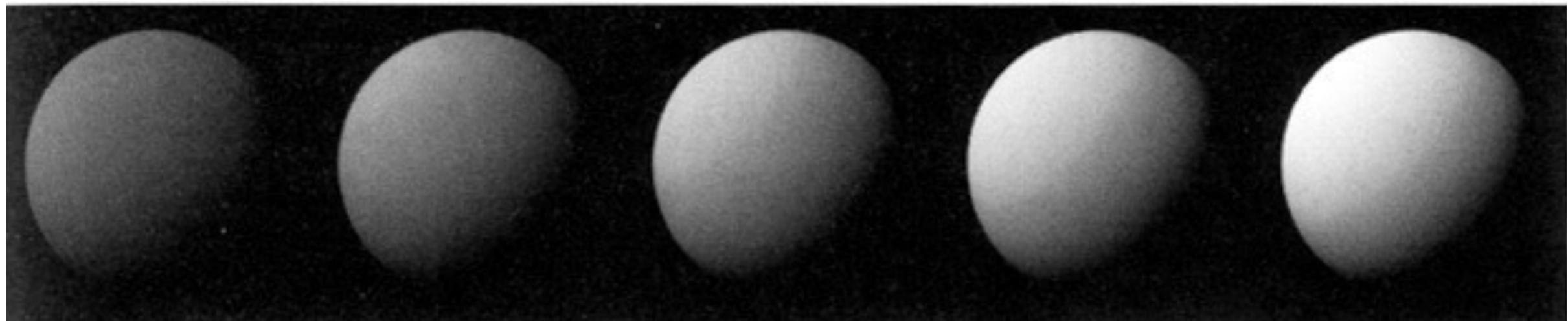
# Lambertian (Diffuse) Shading

Shading **independent** of view direction



# Lambertian (Diffuse) Shading

Produces diffuse appearance



$$k_d \longrightarrow$$

[Foley et al.]

# Thank you!

(And thank Prof. Ravi Ramamoorthi and Prof. Ren Ng for many of the slides!)