Lecture 5:

Shading

Computer Graphics 2025

Fuzhou University - Computer Science

Today's Topics

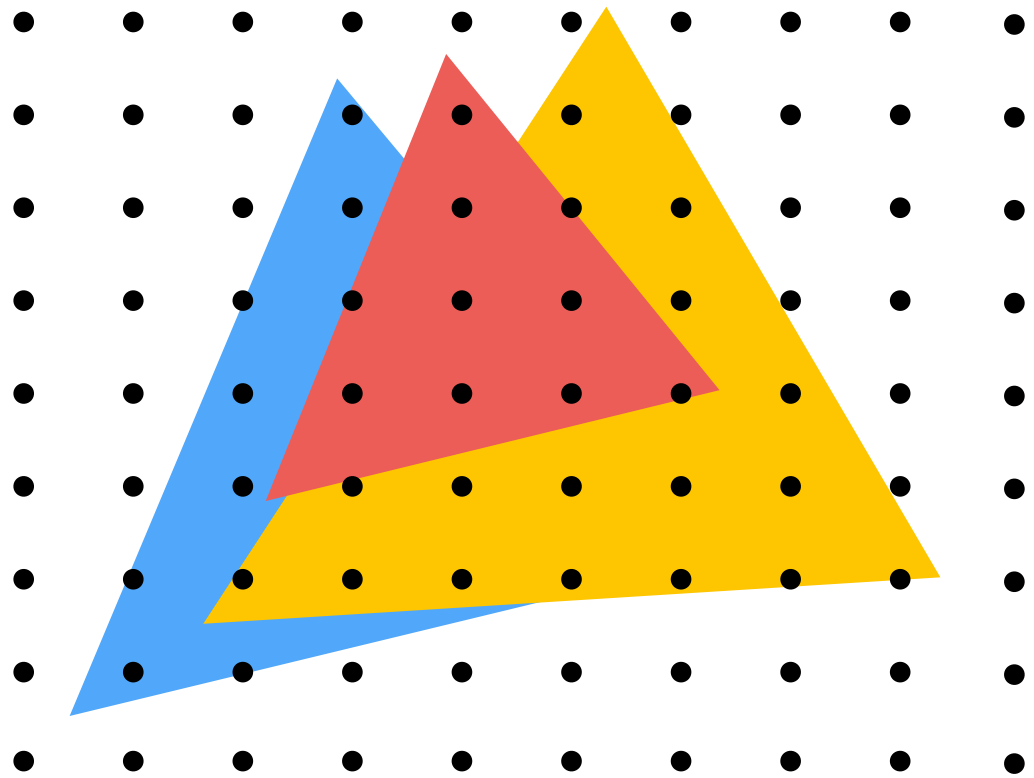
Visibility/Occlusion (Z-Buffering)

Barycentric Coordinates

Shading

Visibility

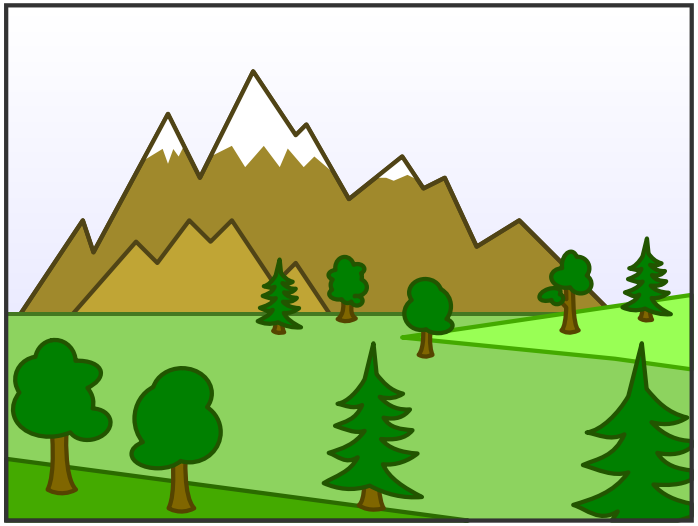
Which triangle is visible at each covered sample point?



Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer

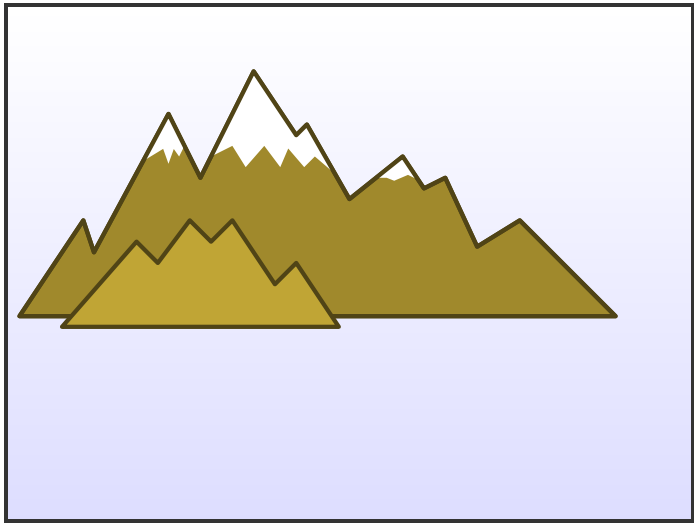


[Wikipedia]

Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer

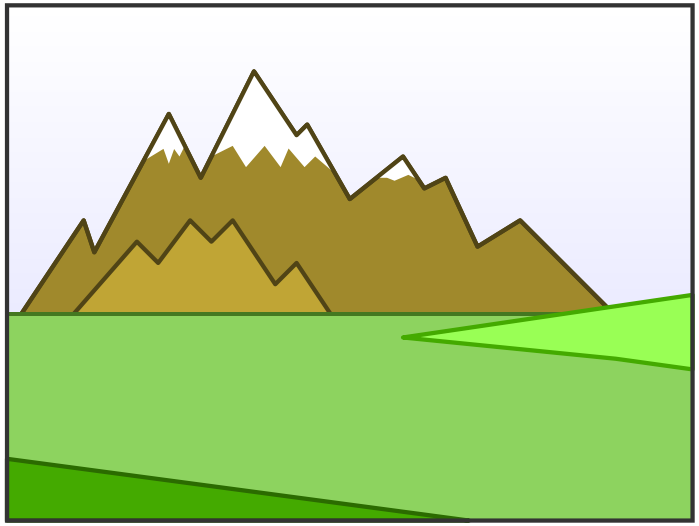


[Wikipedia]

Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer

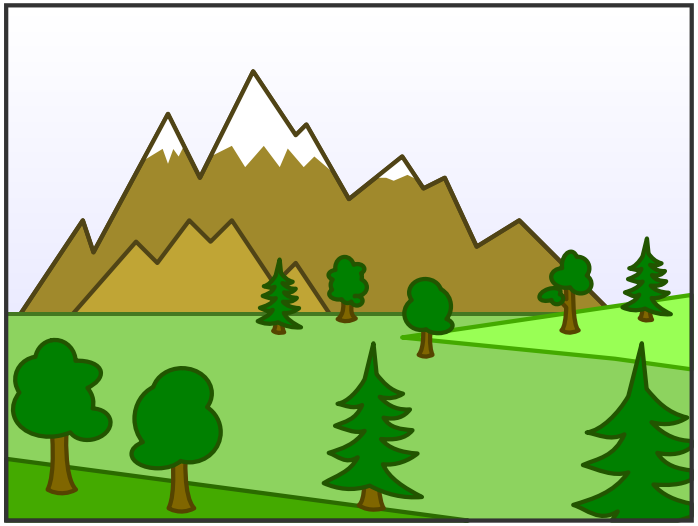


[Wikipedia]

Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer

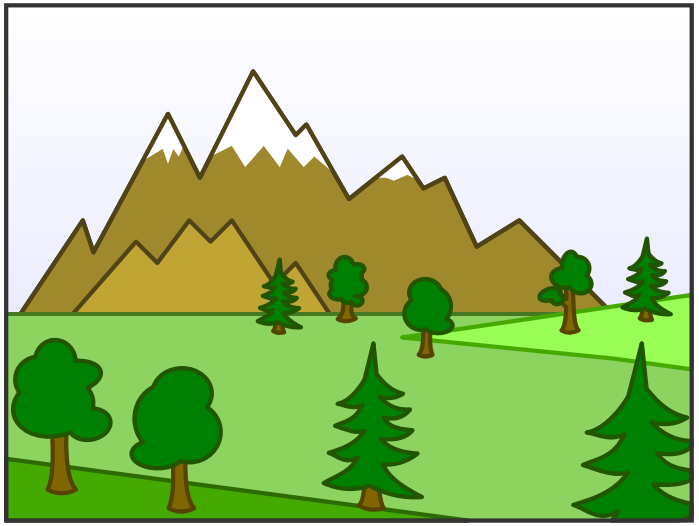


[Wikipedia]

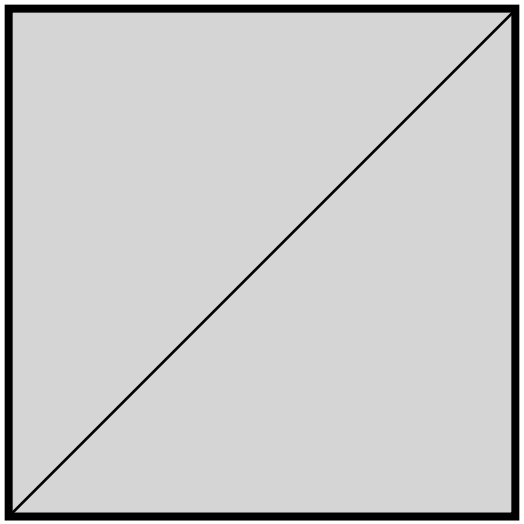
Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer



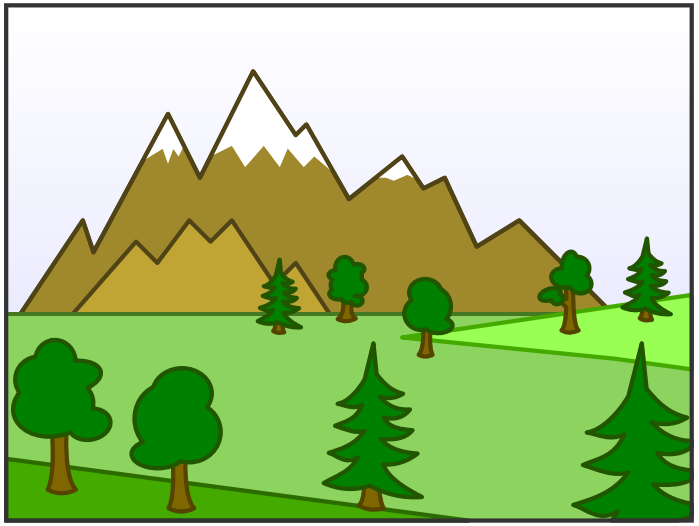
[Wikipedia]



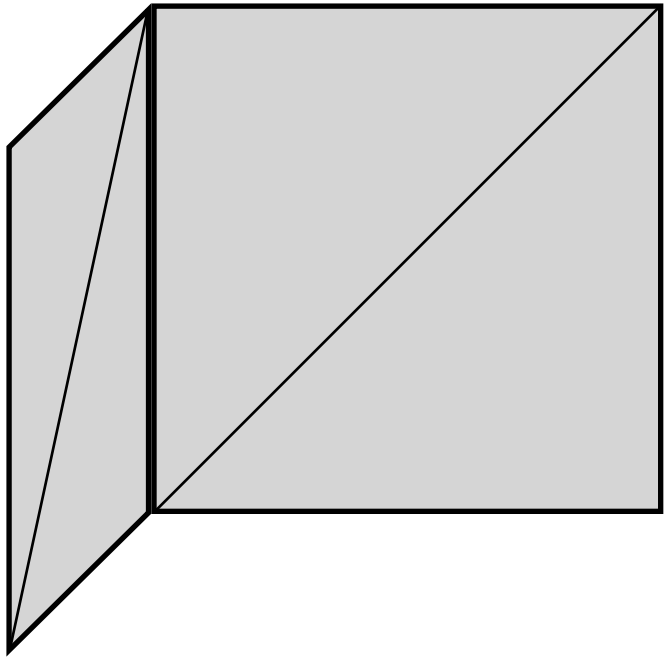
Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer



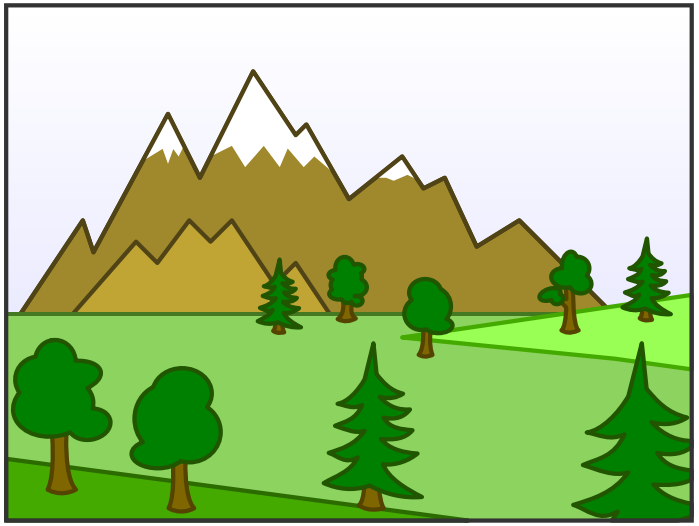
[Wikipedia]



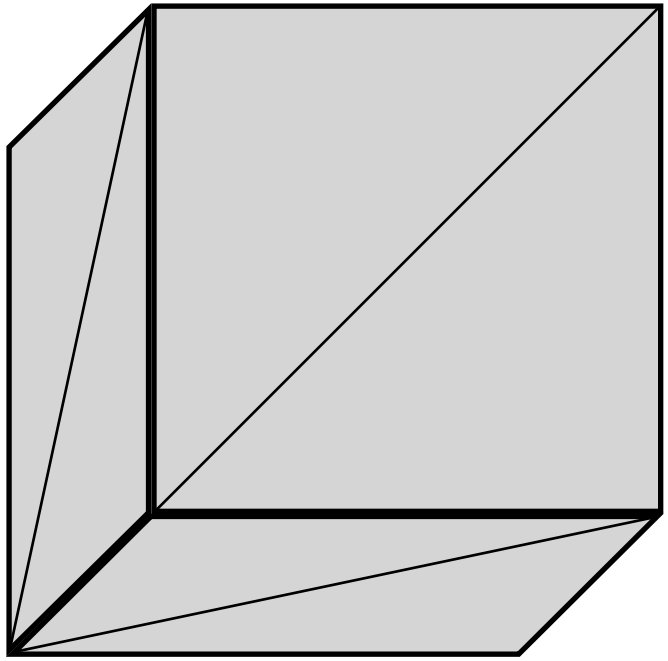
Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer



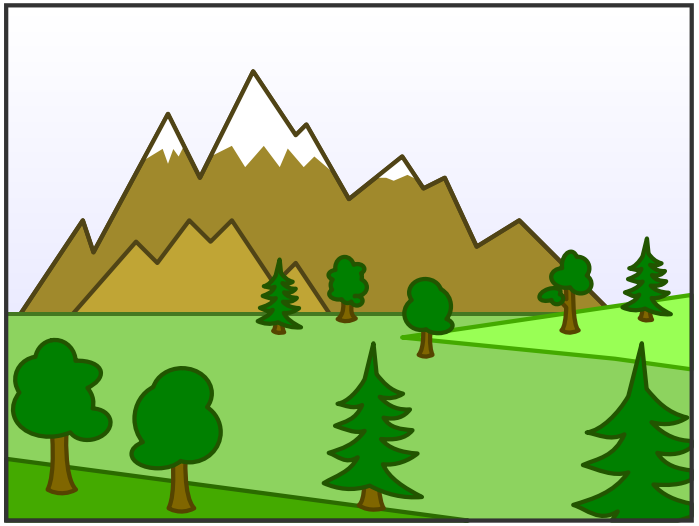
[Wikipedia]



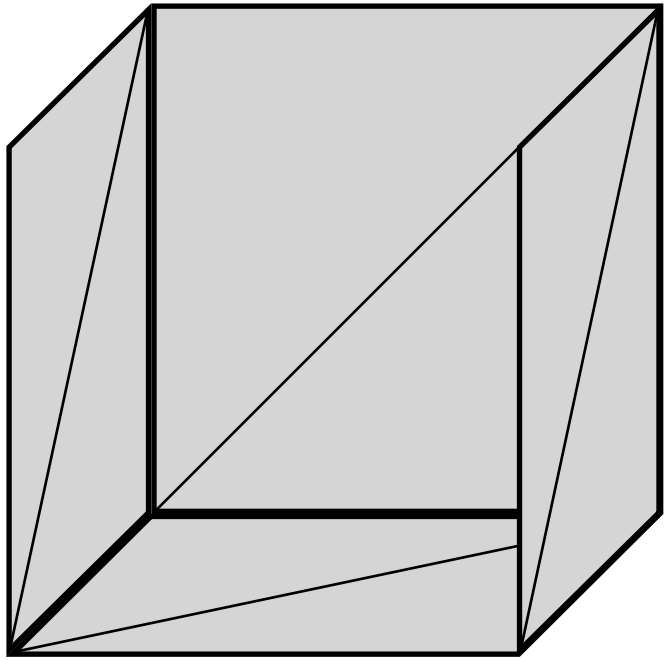
Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer



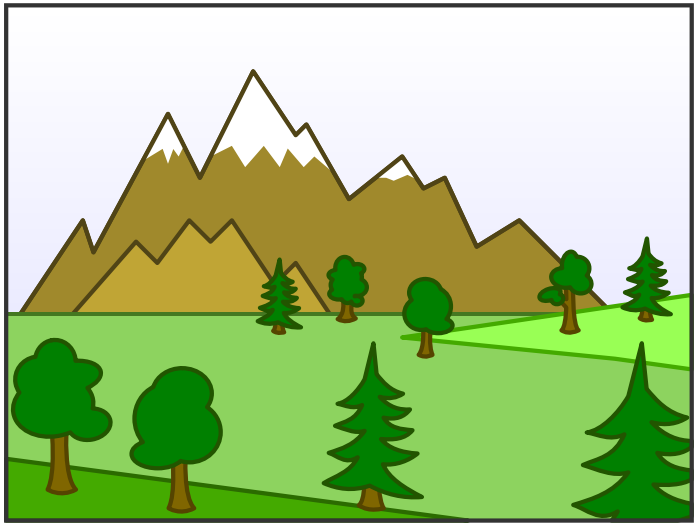
[Wikipedia]



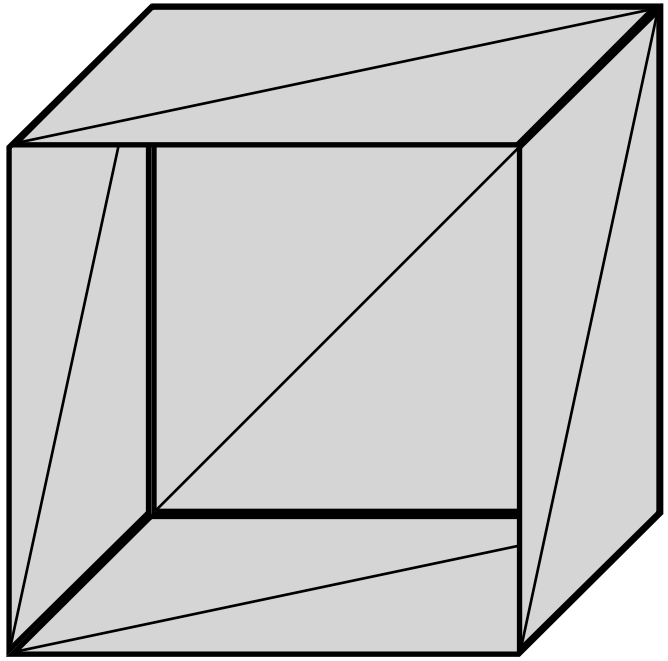
Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer



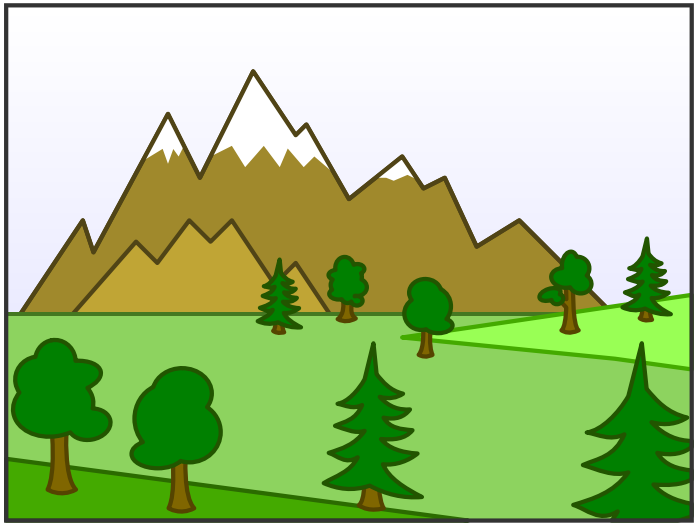
[Wikipedia]



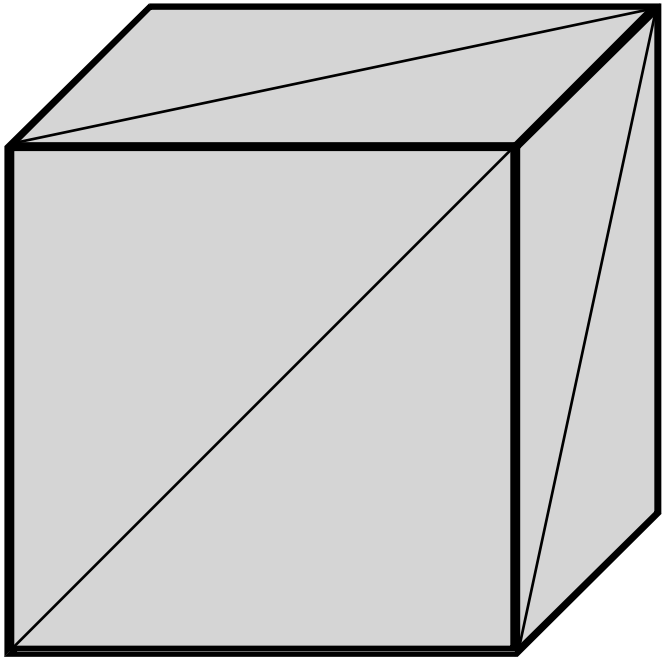
Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer



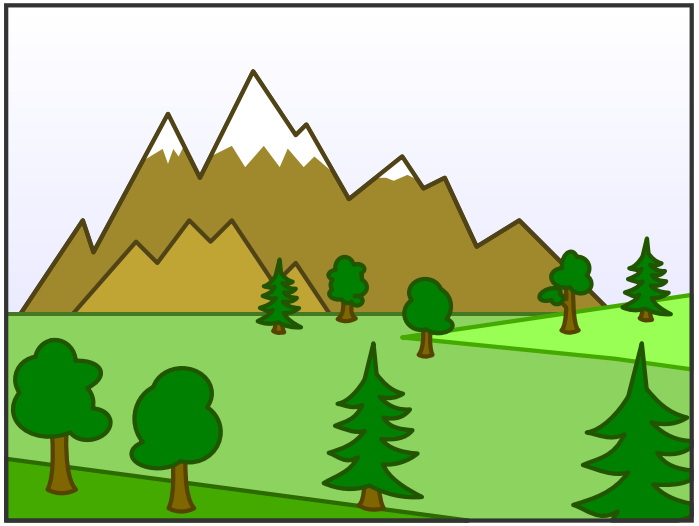
[Wikipedia]



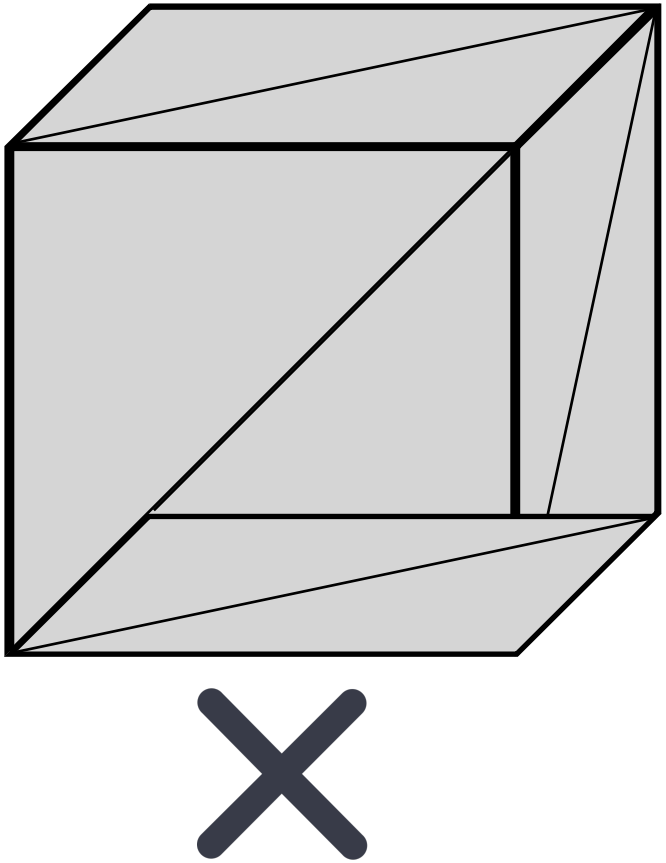
Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer



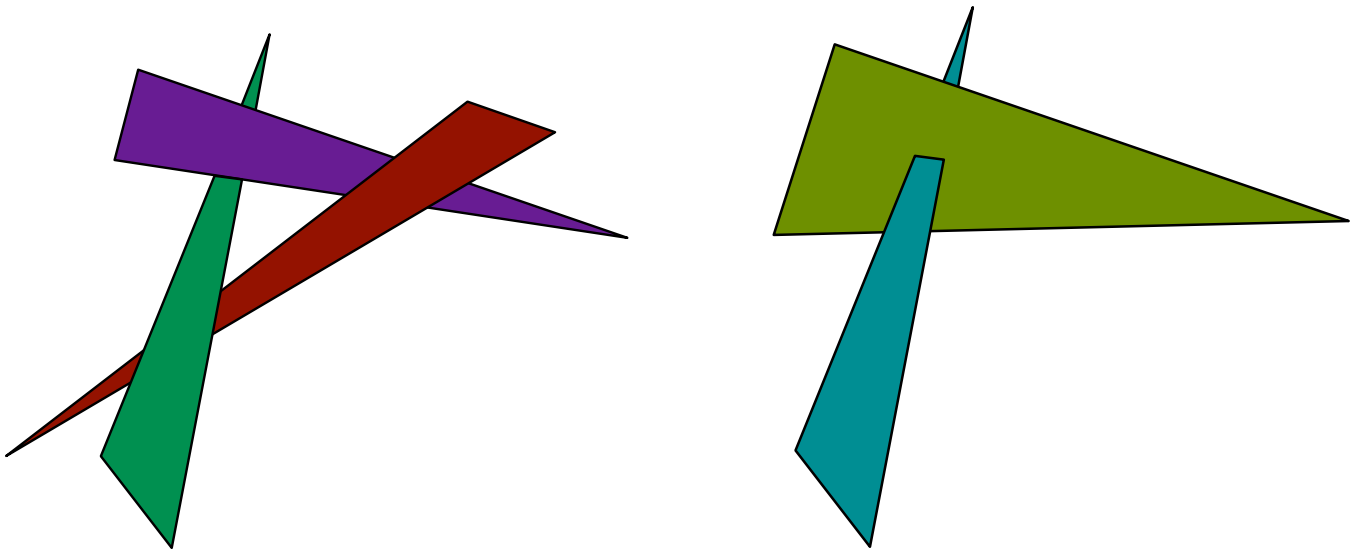
[Wikipedia]



Painter's Algorithm

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

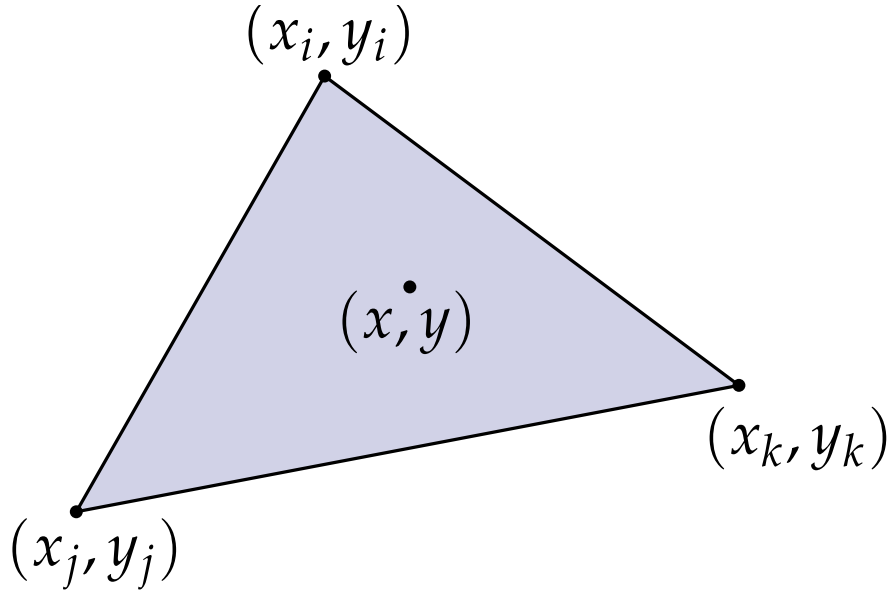
Can have unresolvable depth order

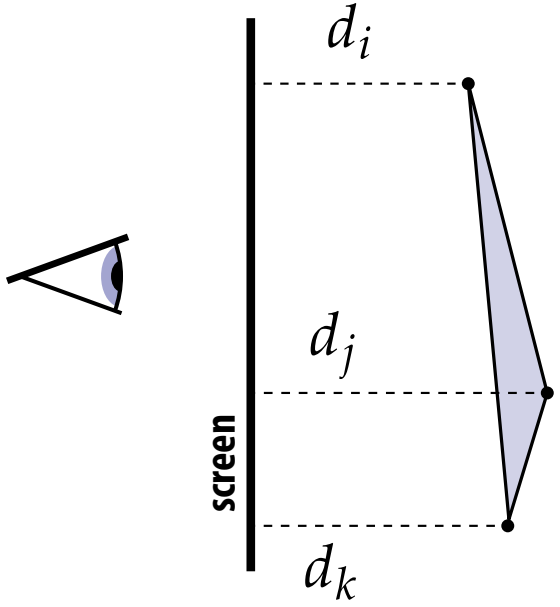


Triangle Coordinates with Depths

Assume we have a triangle given by:

- The projected 2D coordinates (x_i, y_i) of each vertex
- The "depth" d_i of each vertex (distance from the viewer)





Q: How do we compute the depth d at a given sample point (x, y)?

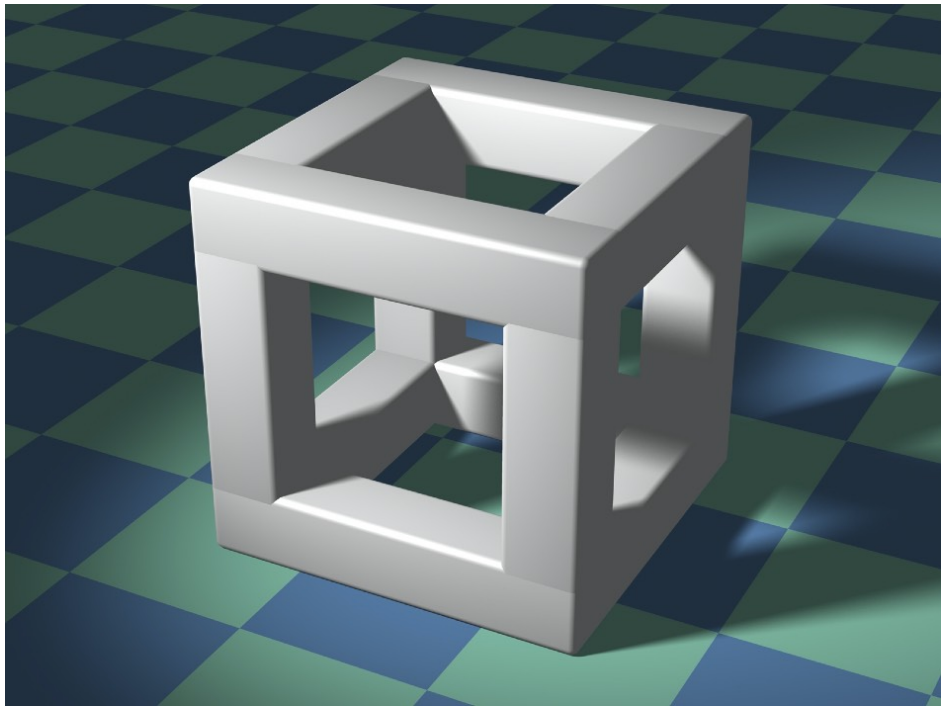
Z-buffer 深度缓冲

计算并记录每个采样点的深度值

Idea:

- Store current min. z-value for each sample position
- Needs an additional buffer for depth values
  - framebuffer stores RBG color values
  - depth buffer (z-buffer) stores depth (16 to 32 bits)

Z-Buffer Example



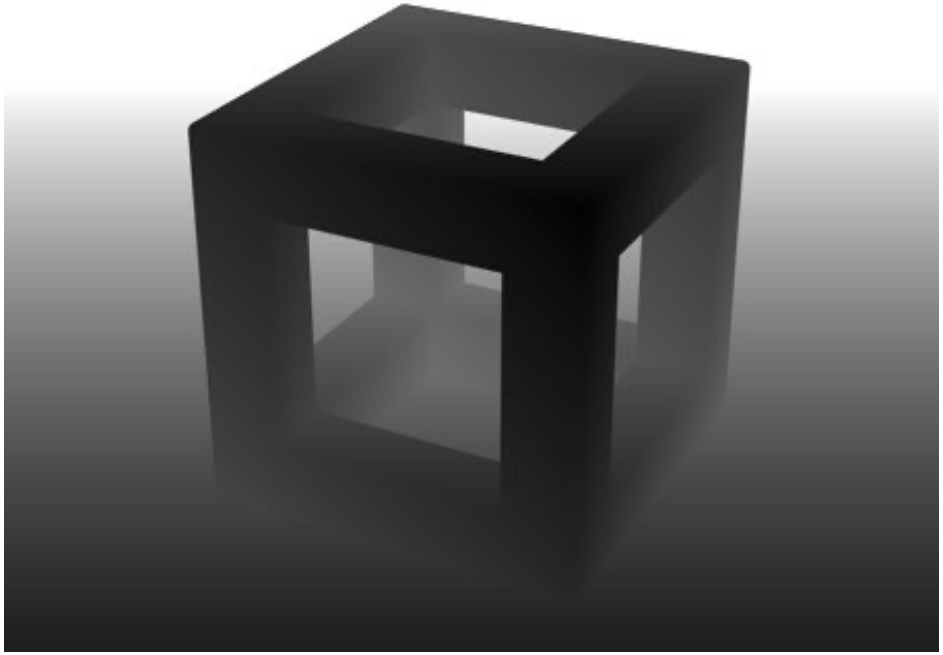
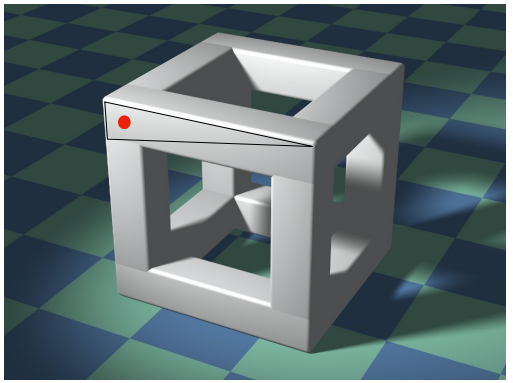


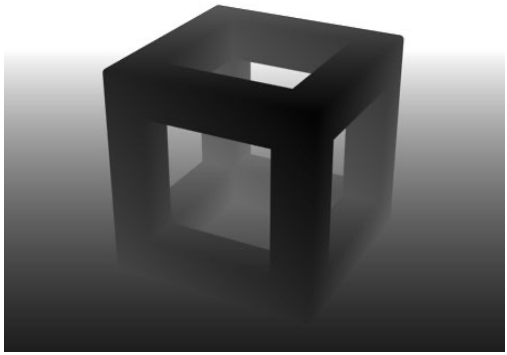
Image source: Dominic Alves, flickr.

Z-Buffer Algorithm

Initialize depth buffer to ∞

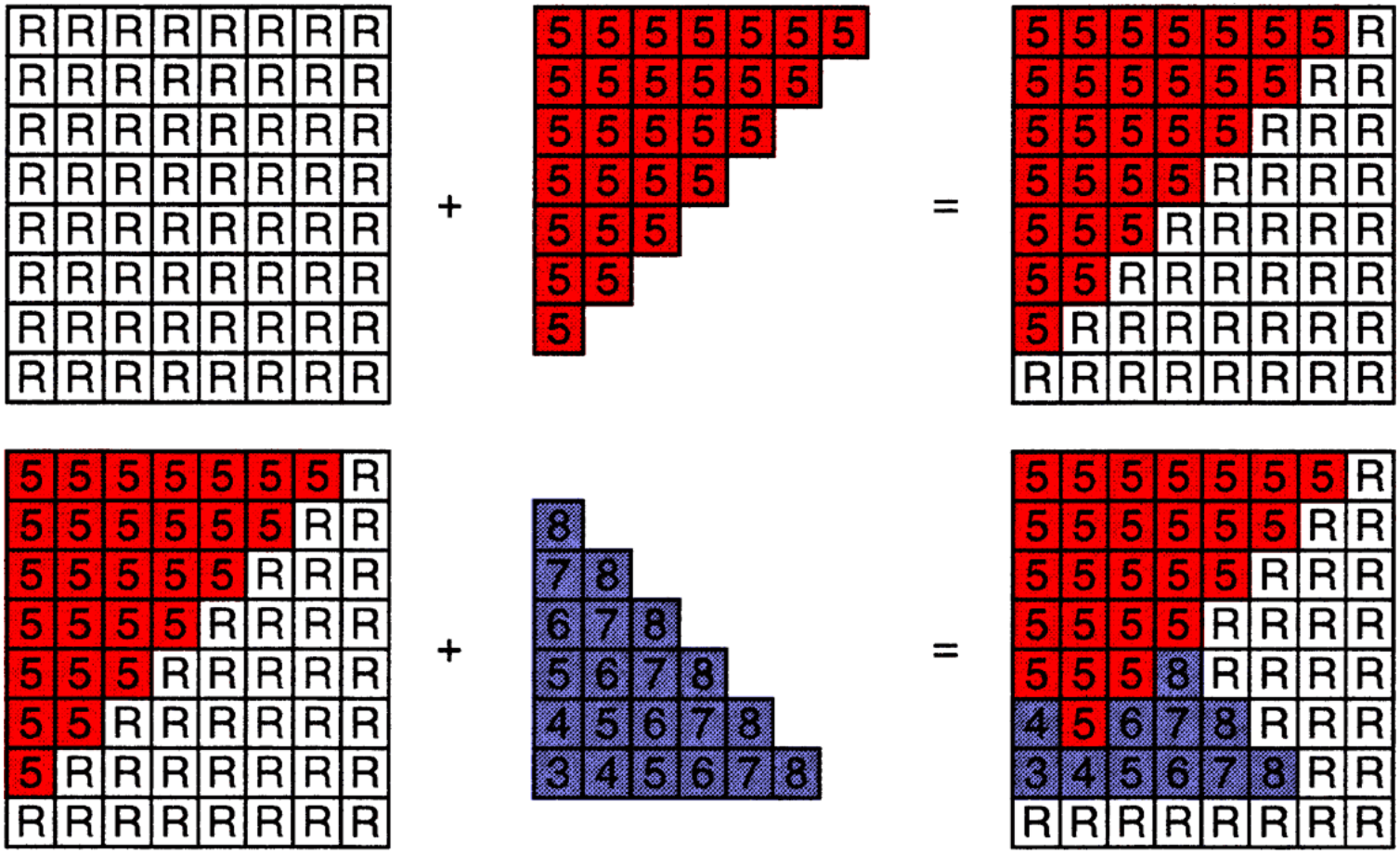
During rasterization:





```
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 z
    else
      ;     // do nothing, this sample is not closest
```

Z-Buffer Algorithm



Z-Buffer Complexity

Complexity

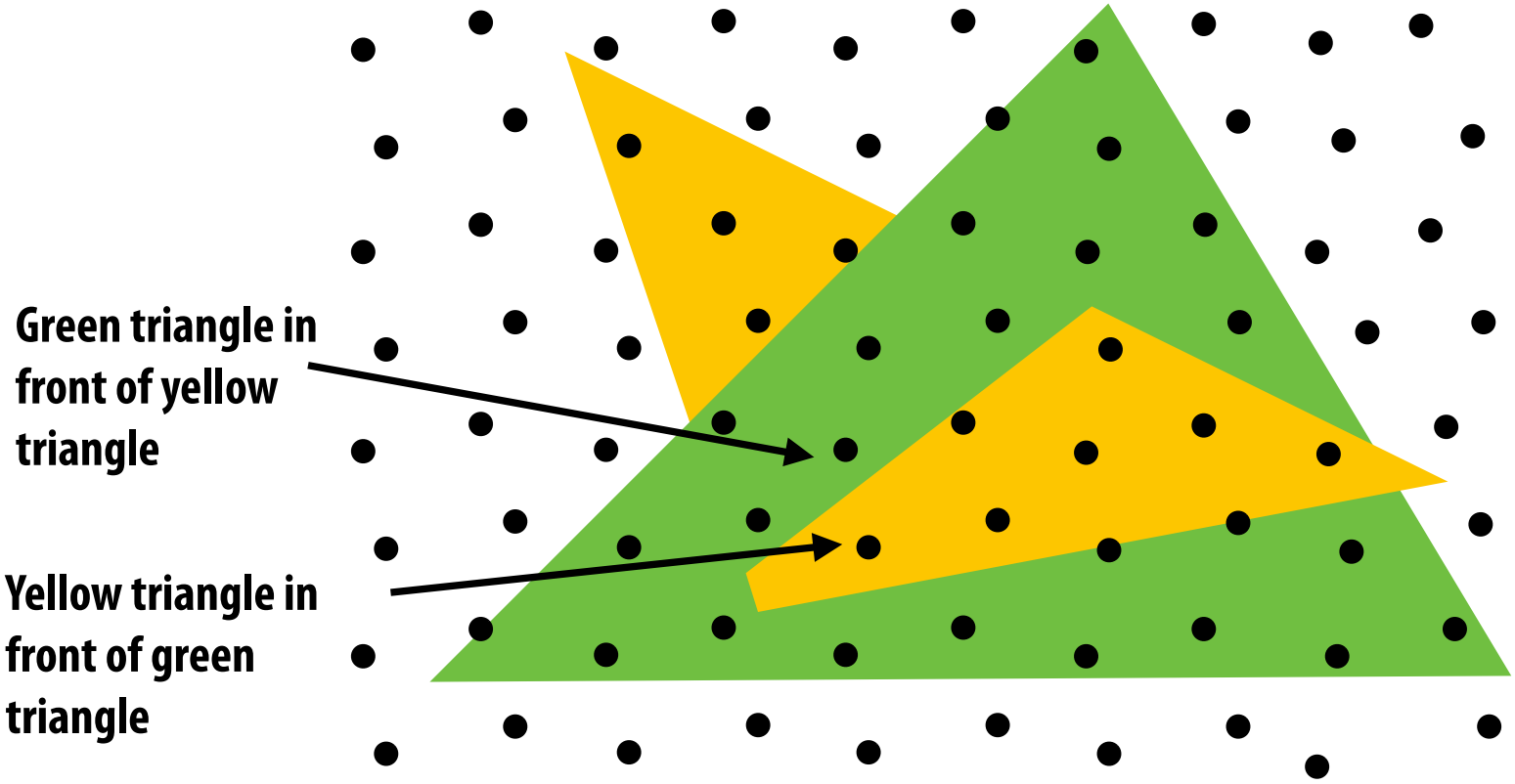
- O(*n*) for *n* triangles
- How can we sort *n* triangles in linear time? (Counting sort)

Drawing triangles in different orders?

Most important visibility algorithm

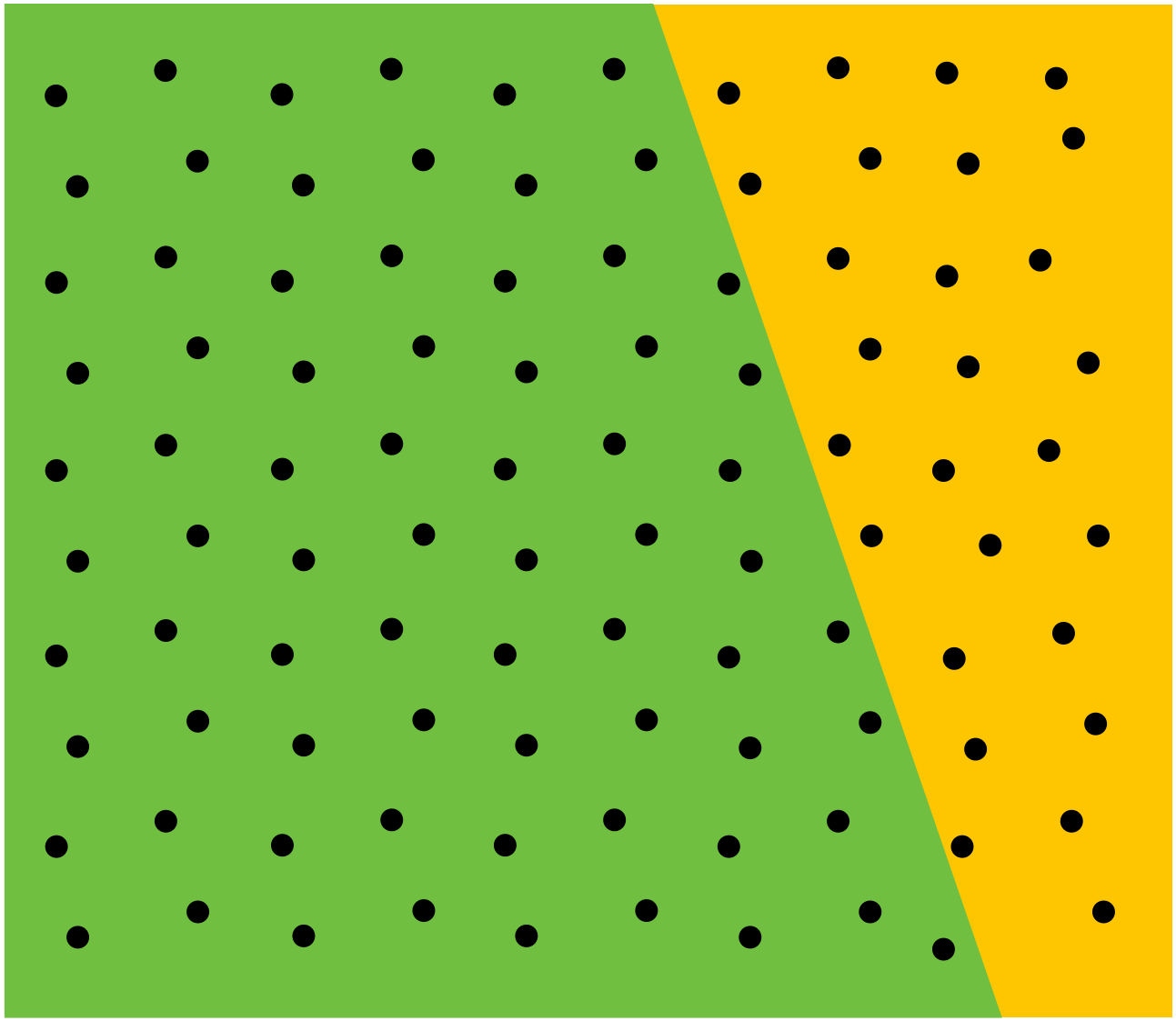
- Implemented in hardware for all GPUs
- Used by OpenGL

Z-buffer can handle interpenetrating triangles

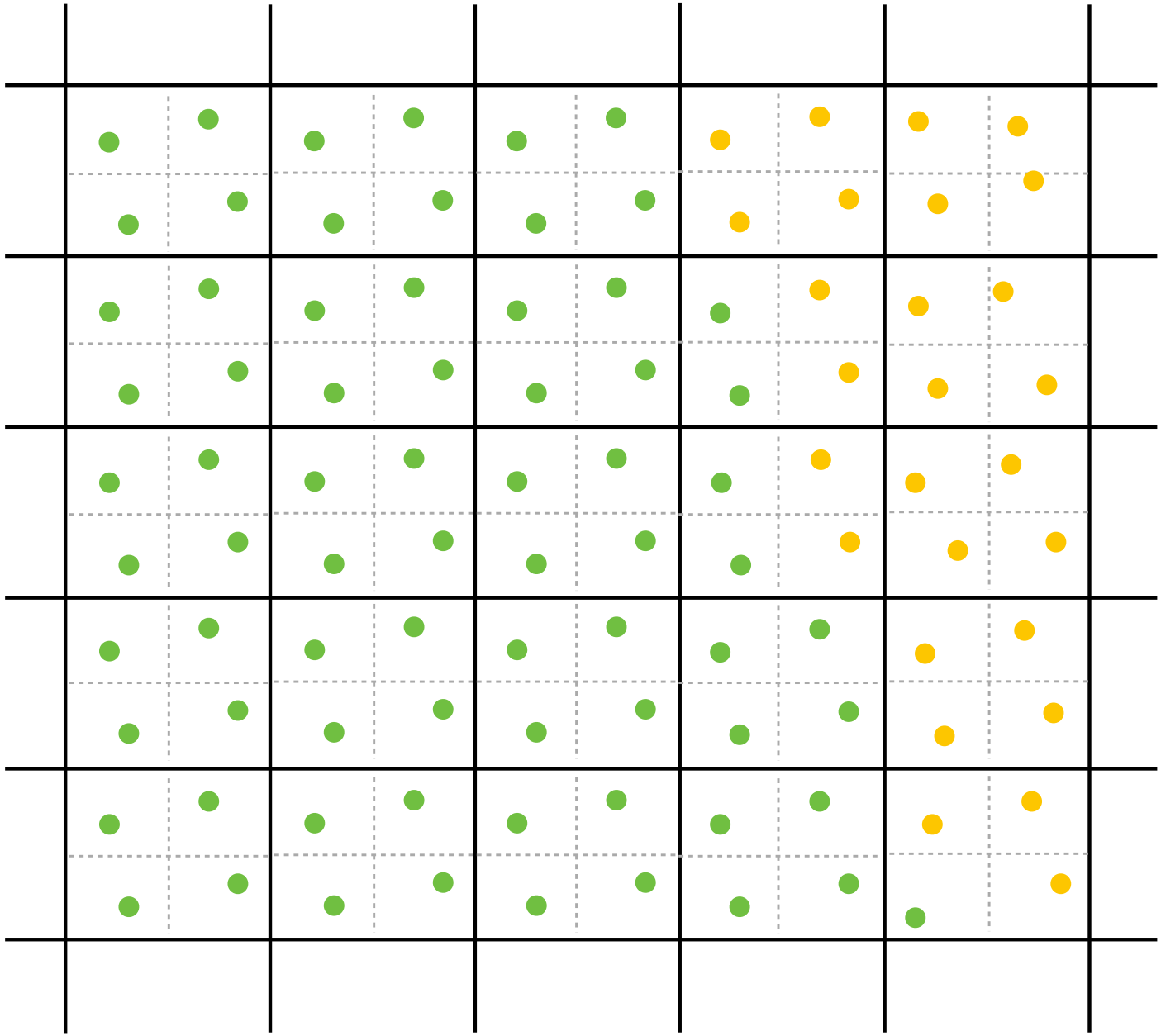


Does Z-buffer work with super sampling?

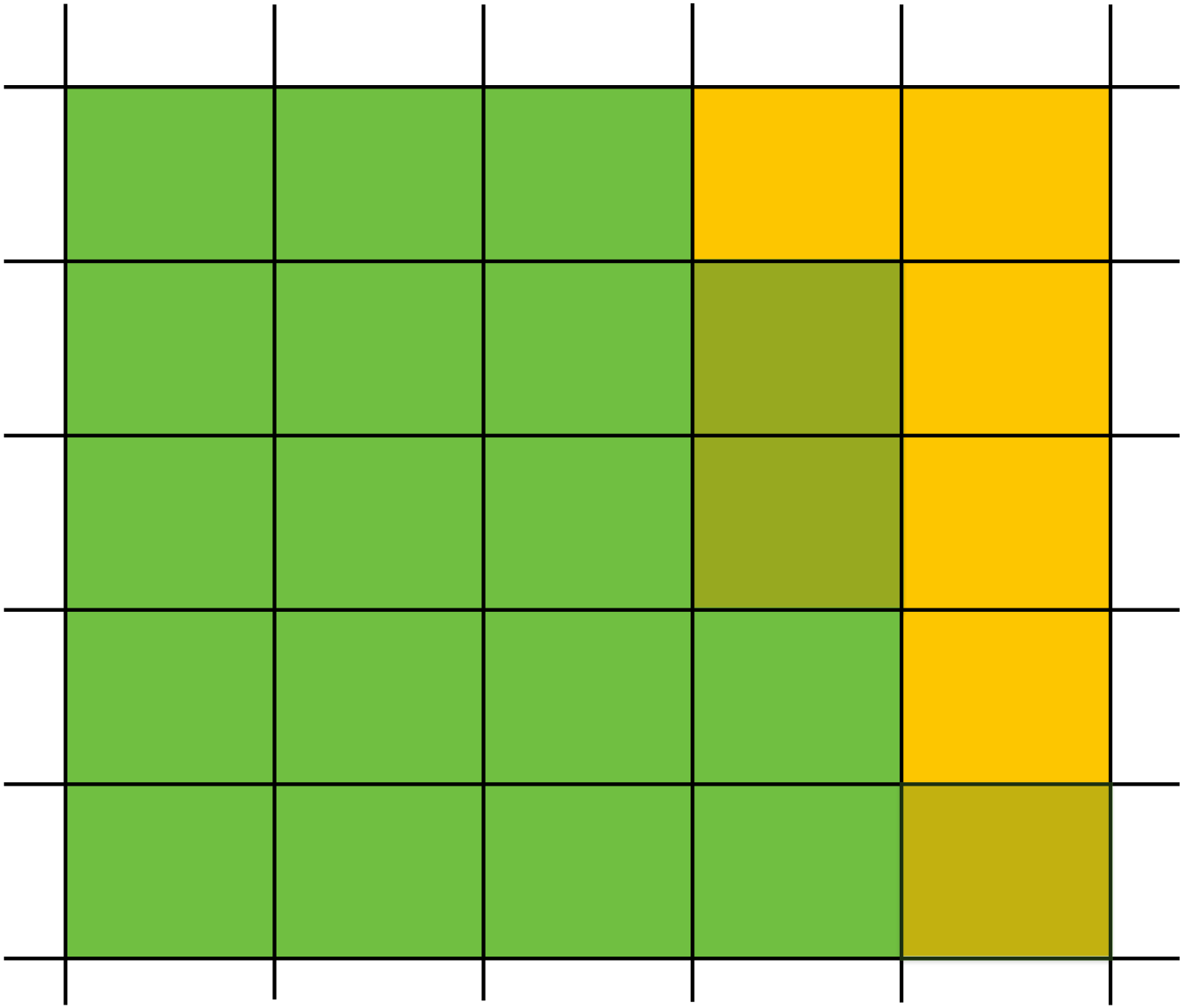
Yes! Occlusion test is per sample, not per pixel.



Z-buffer And Super Sampling



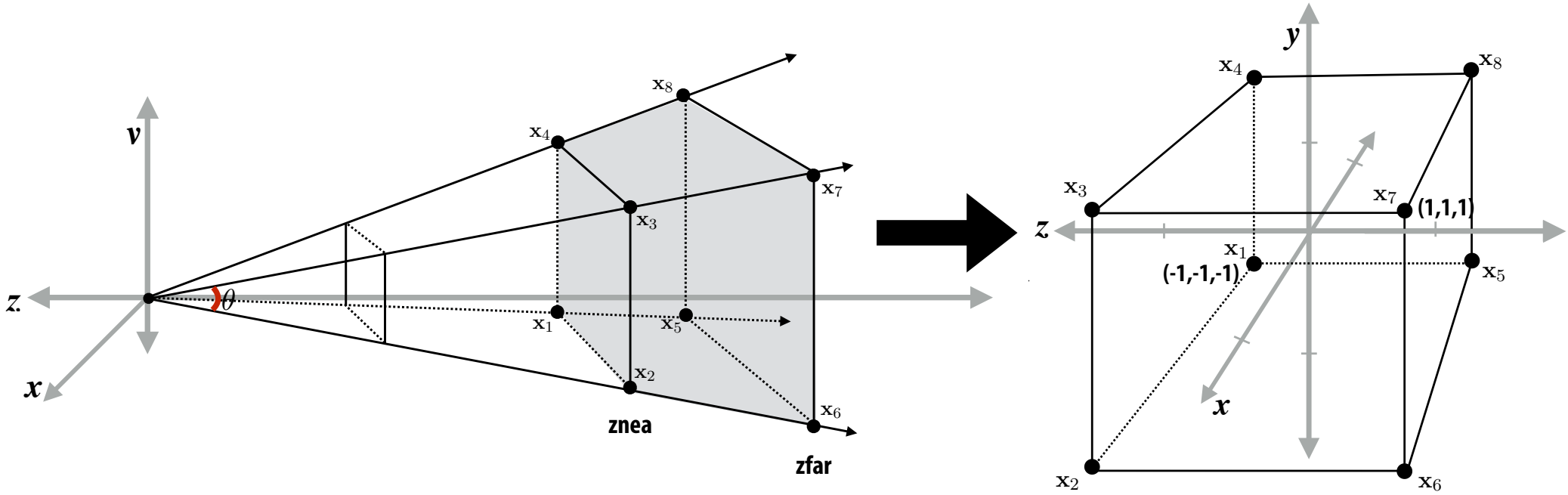
Z-buffer And Super Sampling



Note anti-aliasing of edge due to filtering of green and yellow samples.

How to Compute Depth?

Before mapping to 2D, map corners of frustum to corners of cube:



Camera Coordinates

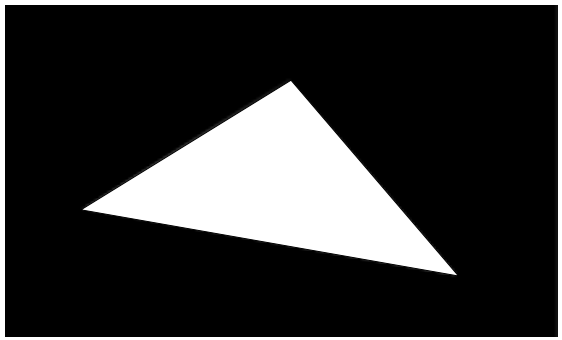
Normalized Device Coords "NDC"

Z-value is preserved after projection

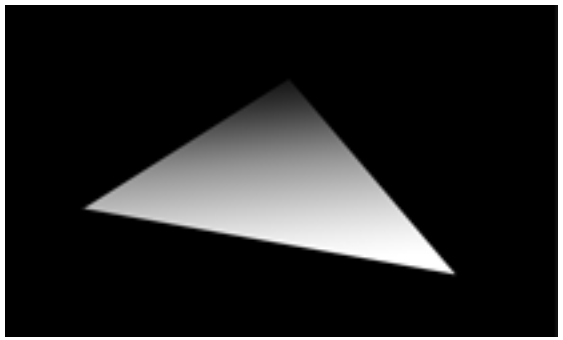
Recall Z-value Computation

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$z \mapsto \frac{f+n}{f-n} + \frac{1}{z}\frac{2fn}{f-n}$$



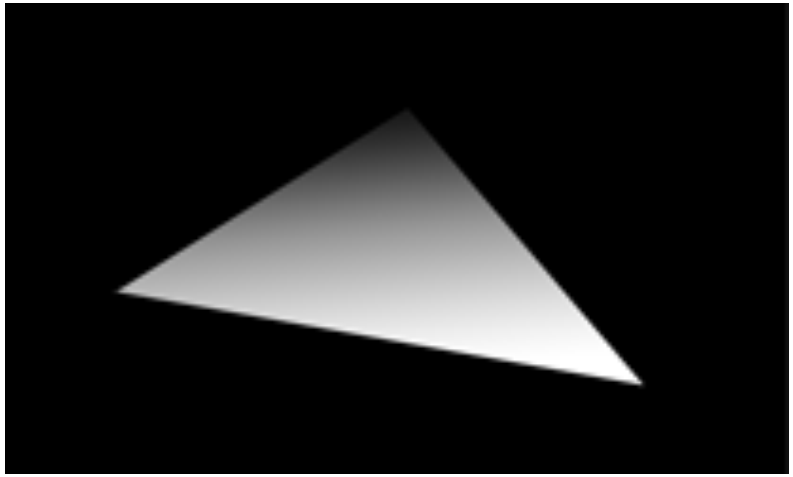
z-value are the same



z-value are different

Z-value Computation

1. Compute for all samples $z \mapsto \frac{f+n}{f-n} + \frac{1}{z}\frac{2fn}{f-n}$



2. or interpolation between triangle vertices

Today's Topics

Visibility/Occlusion (Z-Buffering)

Barycentric Coordinates

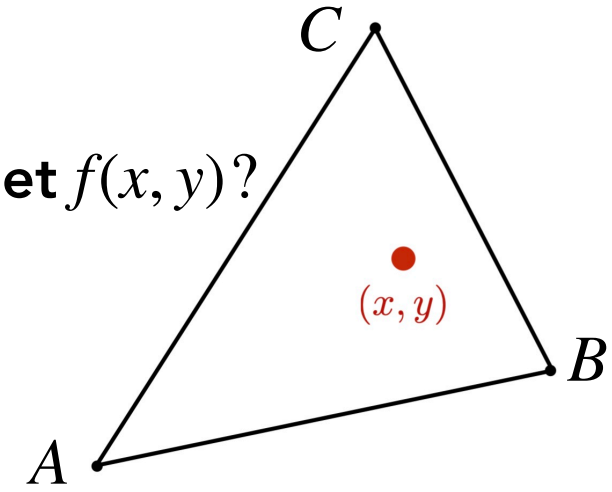
Shading

Interpolation Across Triangles

Why do we want to interpolate?

- Specify values (e.g. texture coordinates) at vertices, and obtain smoothly varying values across surface

Interpolate from $f(A), f(B), f(C)$ to get $f(x, y)$?



What do we want to interpolate?

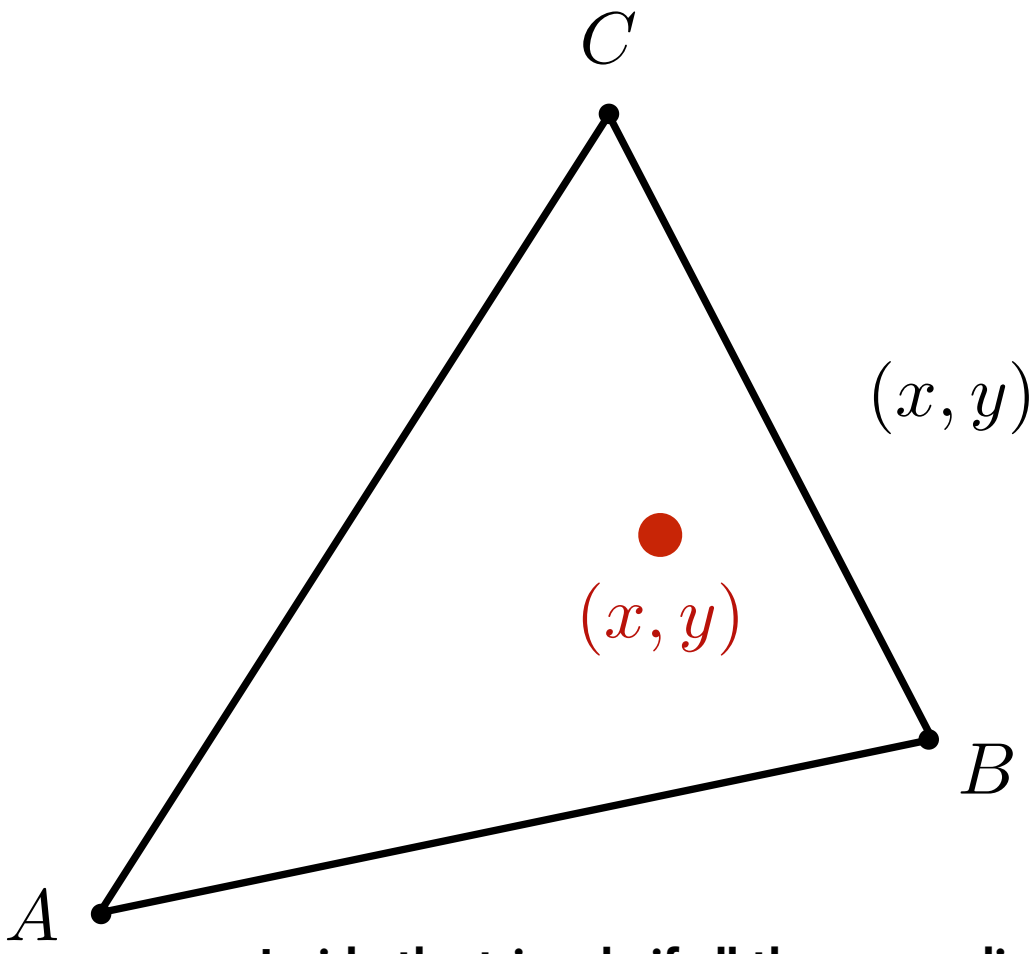
- Texture coordinates, colors, normal vectors, …

How do we interpolate?

- Barycentric coordinates

Barycentric Coordinates

A coordinate system for triangles (α, β, γ)

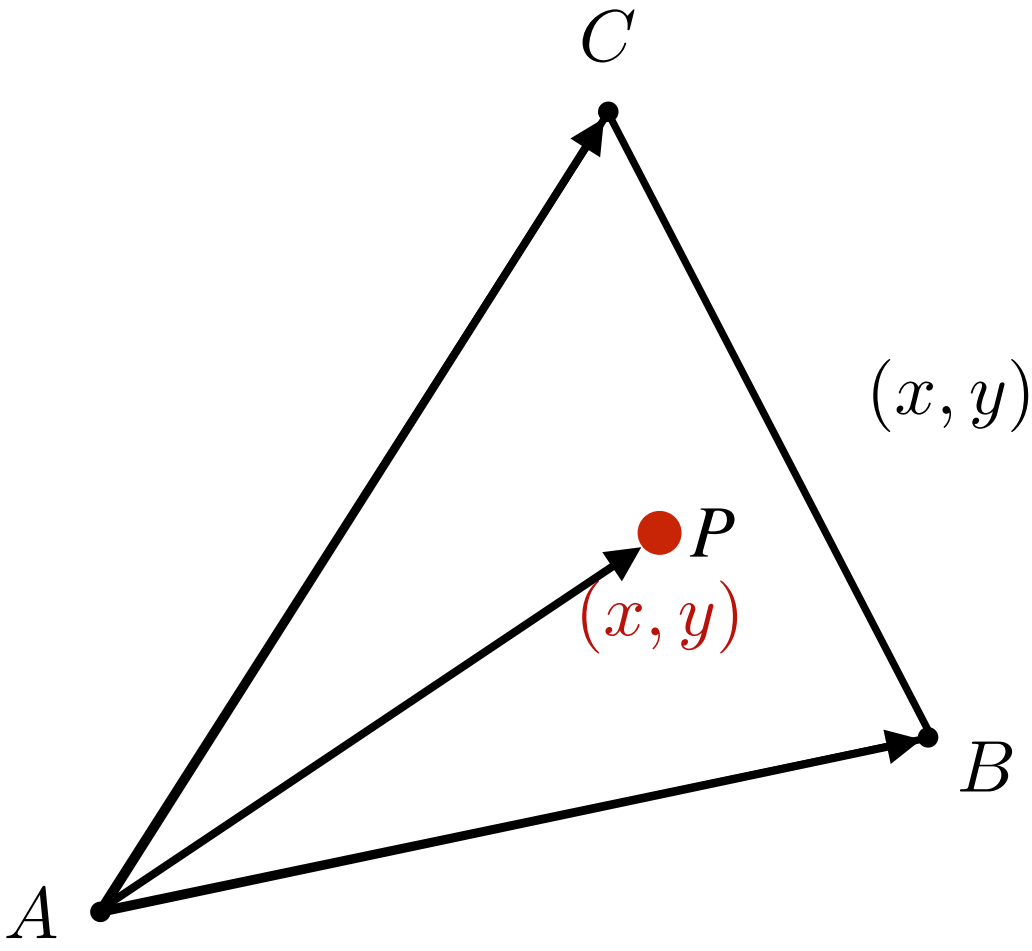


$$(x, y) = \alpha\, A + \beta\, B + \gamma\, C$$

$$\alpha + \beta + \gamma = 1$$

Inside the triangle if all three coordinates are non-negative

Barycentric Coordinates Derivation



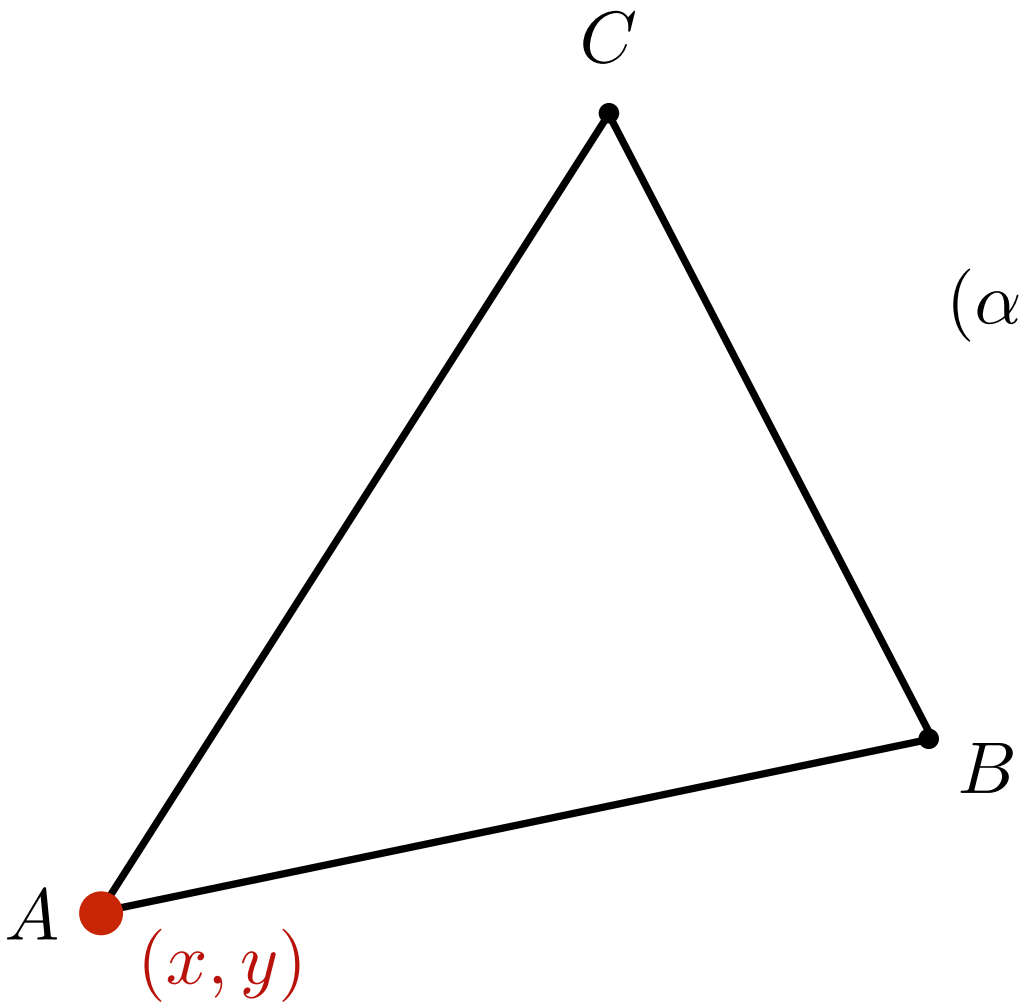
$$(x, y) = \alpha A + \beta B + \gamma C$$

$$\alpha + \beta + \gamma = 1$$

$$\begin{aligned}
P &= A + \overrightarrow{AP} \\
&= A + (\beta\overrightarrow{AB} + \gamma\overrightarrow{AC}) \\
&= A + \beta(B - A) + \gamma(C - A) \\
&= A - \beta A - \gamma A + \beta B + \gamma C \\
&= (1 - \beta - \gamma)A + \beta B + \gamma C \\
&= \alpha A + \beta B + \gamma C \quad ,(\alpha + \beta + \gamma = 1)
\end{aligned}$$

(x, y)可看成由$\overrightarrow{AB}, \overrightarrow{AC}$两个向量组成的坐标系下的点$P$的坐标，其仅有两个维度（自由度）

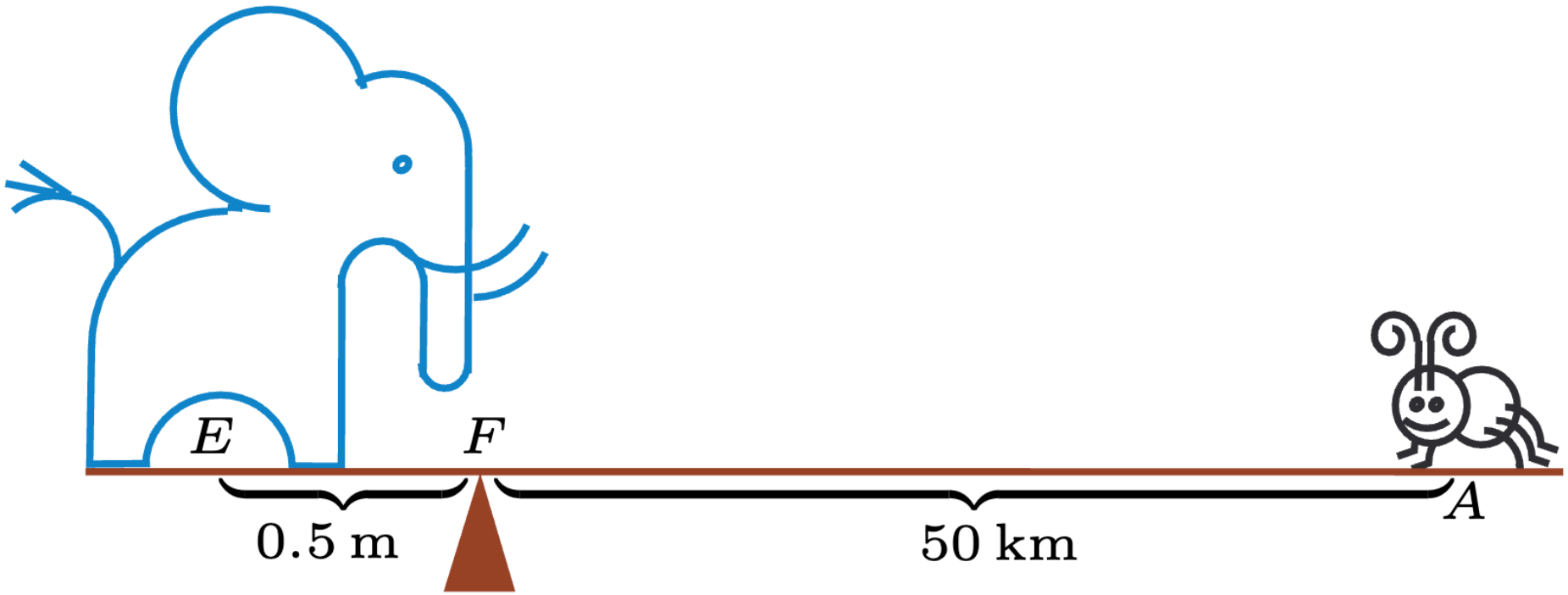
Barycentric Coordinates - Examples



$$(\alpha, \beta, \gamma) = (1, 0, 0)$$
$$(x, y) = \alpha\, A + \beta\, B + \gamma\, C$$
$$= A$$

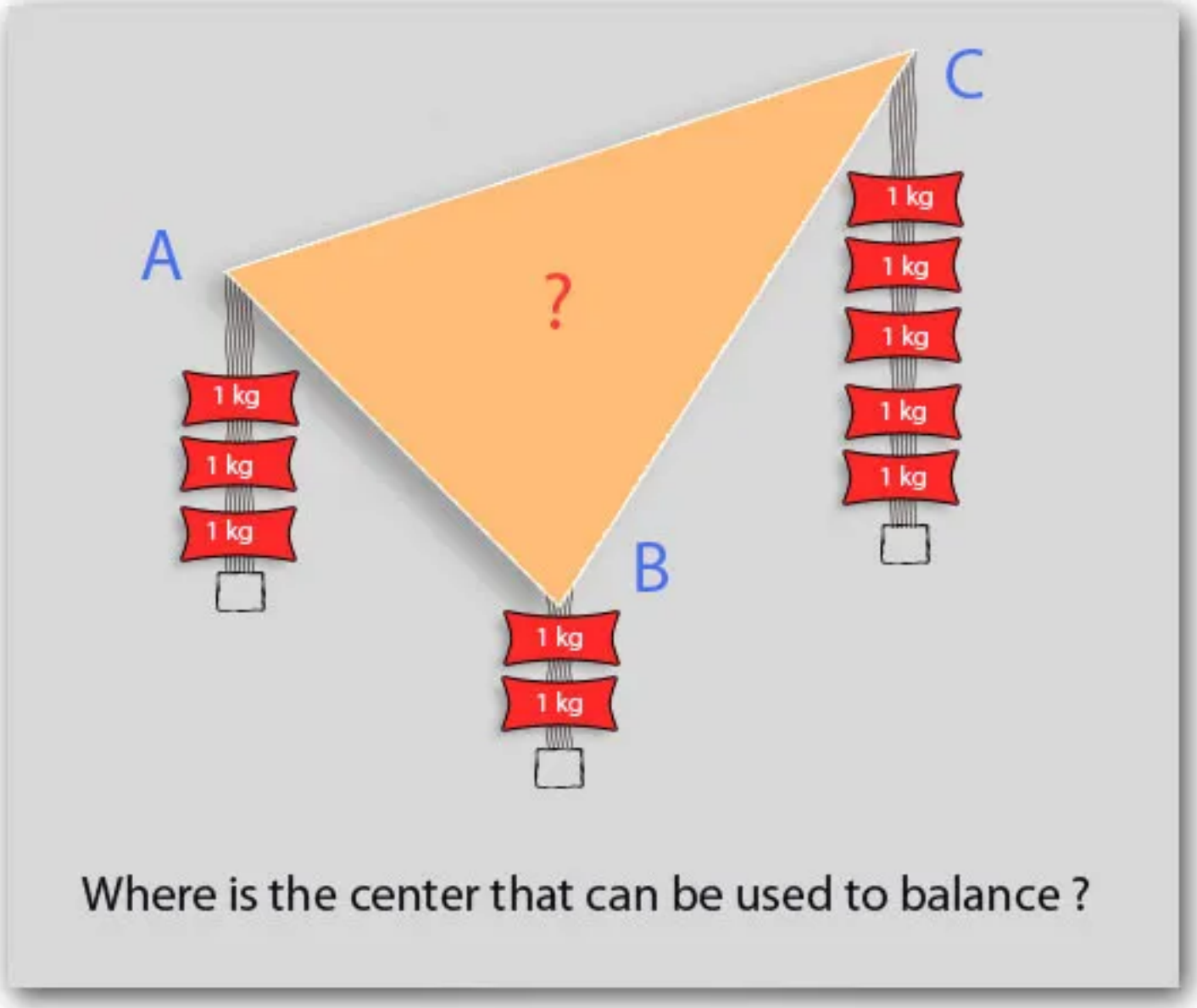
Understanding Barycentric Coordinates

Center of mass!

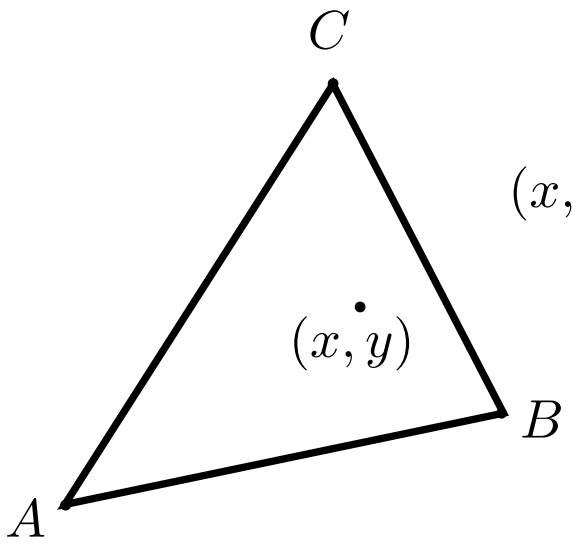


Understanding Barycentric Coordinates

Center of mass!



Barycentric Coordinate Formulas



$$(x, y) = \alpha\, A + \beta\, B + \gamma\, C$$
$$\alpha + \beta + \gamma = 1$$

$$\alpha = \frac{-(x - x_B)(y_C - y_B) + (y - y_B)(x_C - x_B)}{-(x_A - x_B)(y_C - y_B) + (y_A - y_B)(x_C - x_B)}$$

$$\beta = \frac{-(x - x_C)(y_A - y_C) + (y - y_C)(x_A - x_C)}{-(x_B - x_C)(y_A - y_C) + (y_B - y_C)(x_A - x_C)}$$

$$\gamma = 1 - \alpha - \beta$$

Barycentric Coordinate Formulas

Derivation:

$$P = \alpha A + \beta B + \gamma C, \qquad \alpha + \beta + \gamma = 1.$$

1. Eliminate γ.

Use $\gamma = 1 - \alpha - \beta$ and plug in:

$$P = \alpha A + \beta B + (1 - \alpha - \beta)C = C + \alpha(A - C) + \beta(B - C)$$

So

$$P - C = \alpha(A - C) + \beta(B - C).$$

2. Write component equations.

Let $v_0 = A - C,\ v_1 = B - C,\ v_2 = P - C$. In components:

$$\begin{cases} x - x_C = \alpha(x_A - x_C) + \beta(x_B - x_C) \\ y - y_C = \alpha(y_A - y_C) + \beta(y_B - y_C) \end{cases}$$

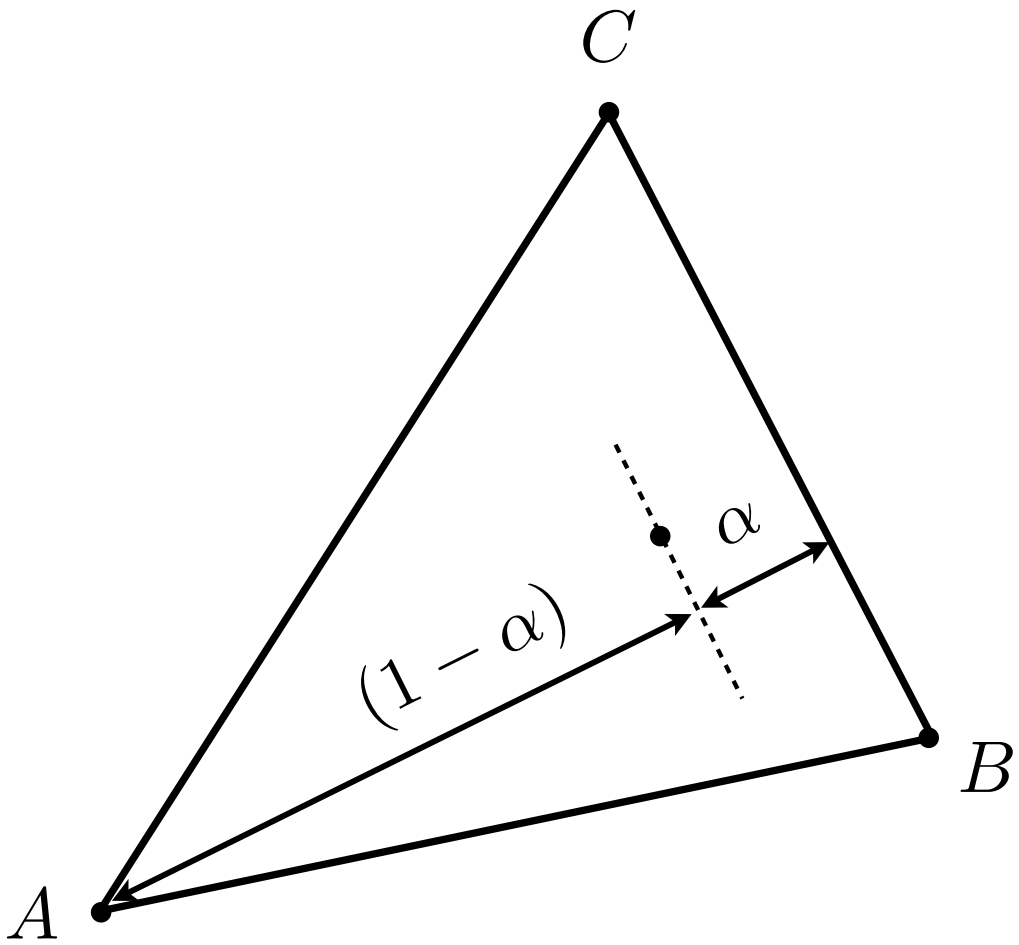
3. Matrix form.

$$\begin{bmatrix} x_A - x_C & x_B - x_C \\ y_A - y_C & y_B - y_C \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} x - x_C \\ y - y_C \end{bmatrix}.$$

Barycentric Coordinates

How to Compute α, β, γ?

Geometric viewpoint — proportional distances

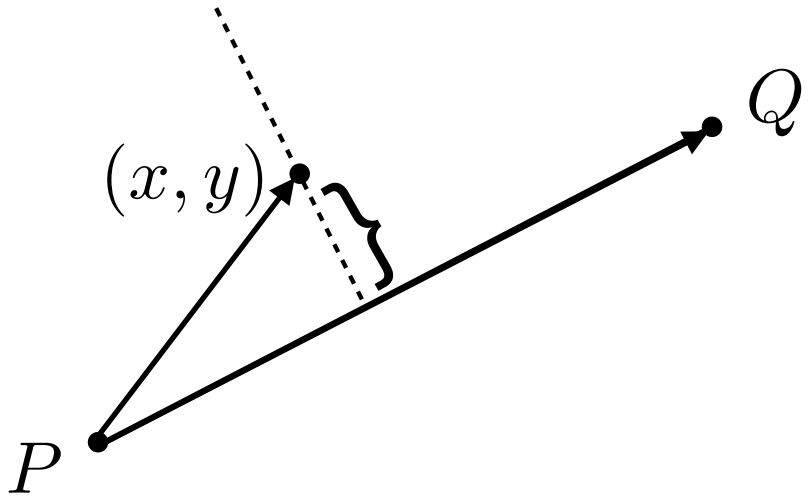


Similar construction
for other coordinates

Computing Barycentric Coordinates

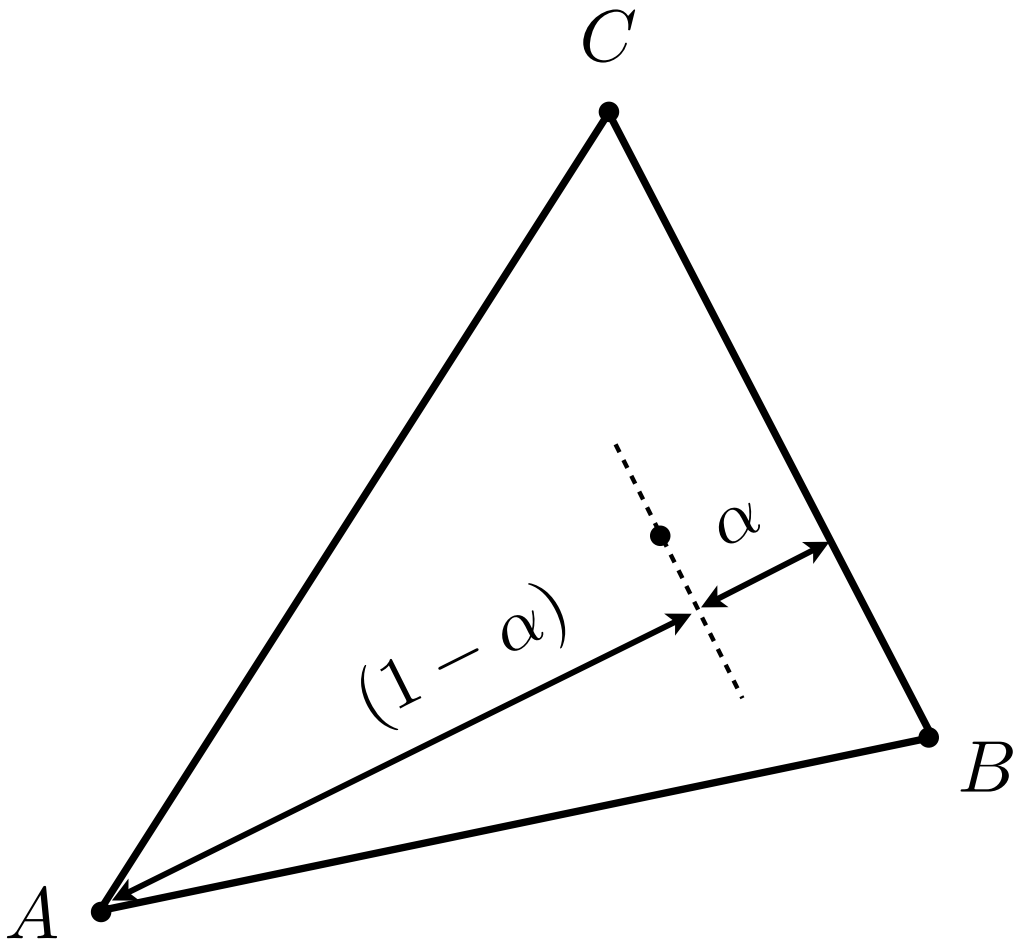
Recall the line equation we derived in Lecture 2. $L_{PQ}(x,y)$ is proportional to the distance from line PQ.

$$L_{PQ}(x,y) = -(x - x_P)(y_Q - y_P) + (y - y_P)(x_Q - x_P)$$



Computing Barycentric Coordinates

Geometric viewpoint — proportional distances

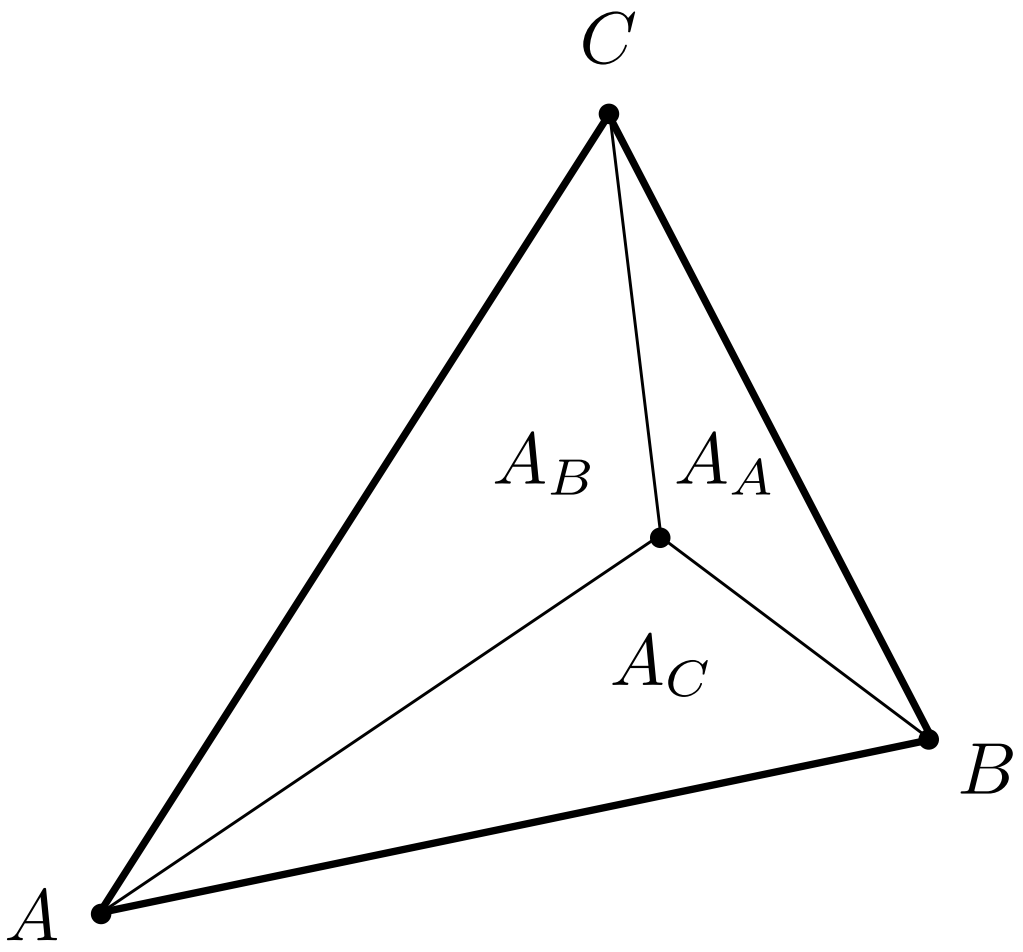


$$\alpha = \frac{L_{BC}(x, y)}{L_{BC}(x_A, y_A)}$$

Similar construction for other coordinates

Barycentric Coordinates

How to Compute α, β, γ? Proportional Areas!



$$\alpha = \frac{A_A}{A_A + A_B + A_C}$$

$$\beta = \frac{A_B}{A_A + A_B + A_C}$$

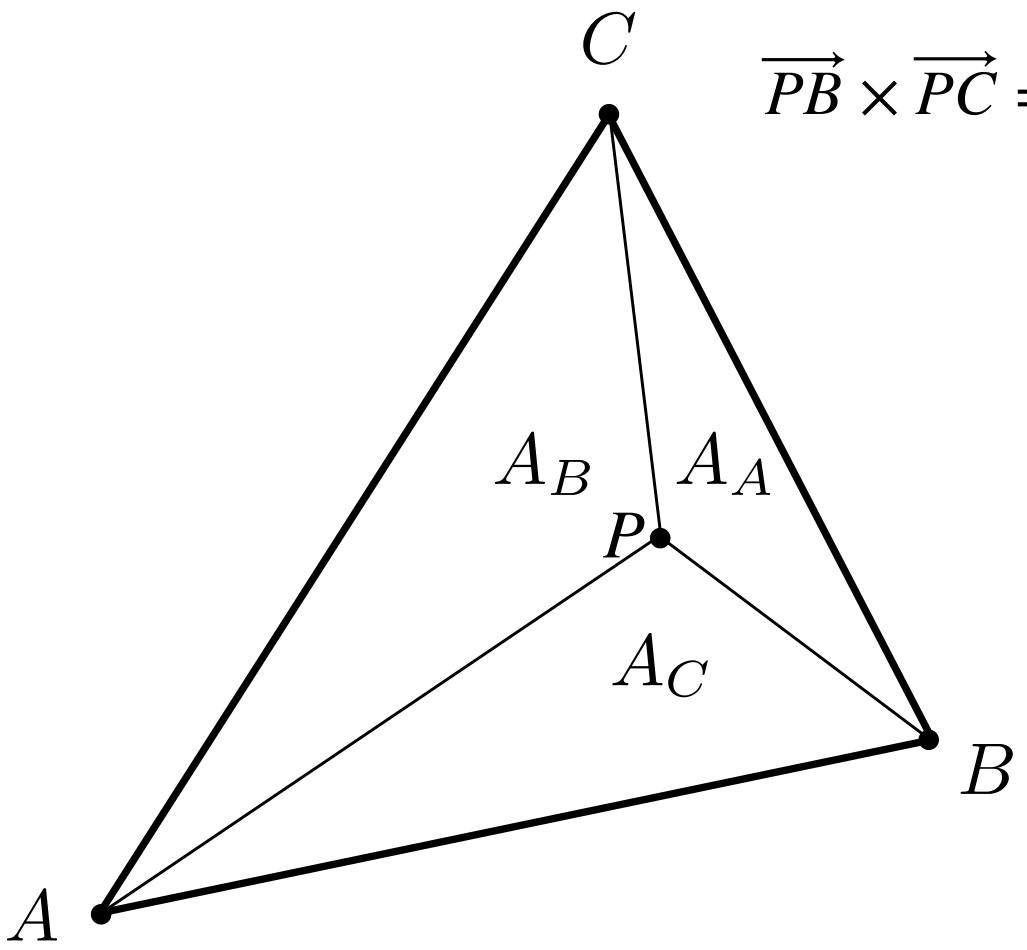
$$\gamma = \frac{A_C}{A_A + A_B + A_C}$$

Barycentric Coordinates

How to compute triangle areas? Cross Product!

$$e.g., A_A = \frac{1}{2}\|\overrightarrow{PB} \times \overrightarrow{PC}\|$$

$$\overrightarrow{PB} \times \overrightarrow{PC} = \|\overrightarrow{PB}\|\|\overrightarrow{PC}\| sin(\theta), \theta = \angle \overrightarrow{PB}, \overrightarrow{PC}$$



$$\alpha = \frac{A_A}{A_A + A_B + A_C}$$

$$\beta = \frac{A_B}{A_A + A_B + A_C}$$

$$\gamma = \frac{A_C}{A_A + A_B + A_C}$$

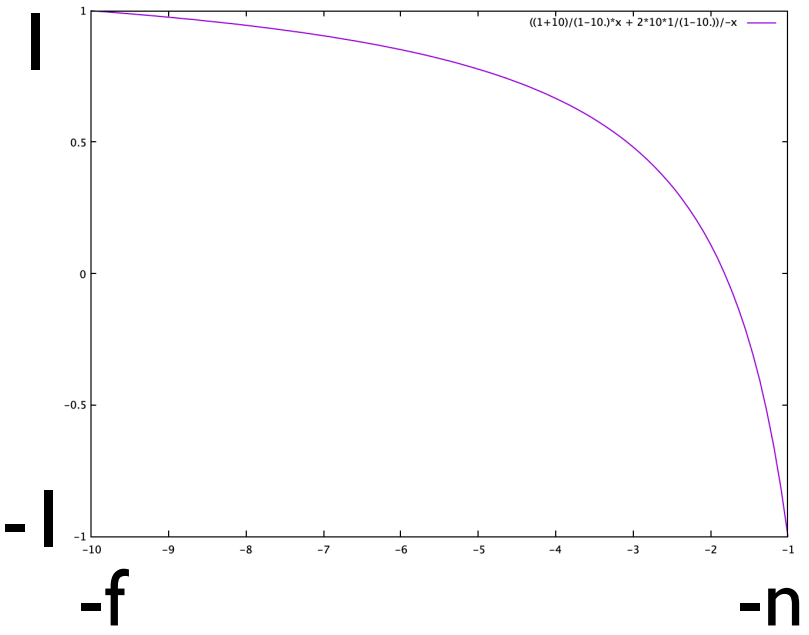
Perspective Projection and Interpolation

Recall Perspective Transform

透视变换不是线性变换

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Graphically:



In equation form:

$$z \mapsto \frac{f+n}{f-n} + \frac{1}{z}\frac{2fn}{f-n}$$

Perspective Projection and Interpolation

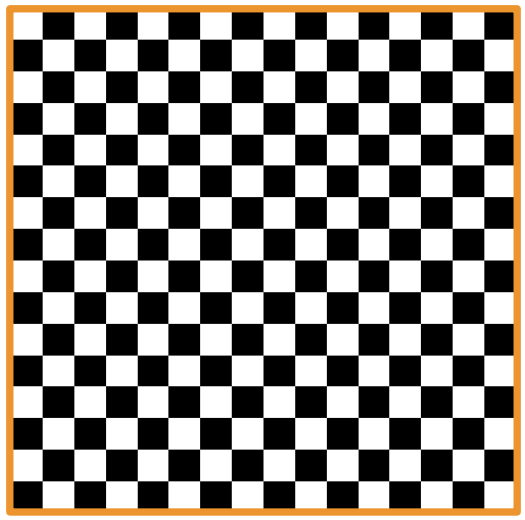
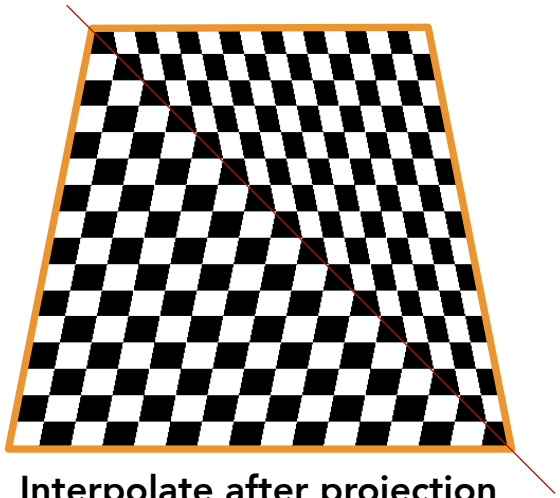


Image Function

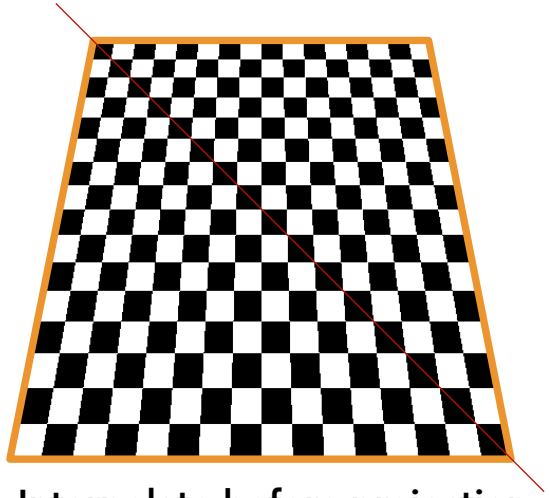
$$C(x,y) = \begin{cases} c_1 & \text{if } \lfloor x/s \rfloor + \lfloor y/s \rfloor \text{ is even} \\ c_2 & \text{if } \lfloor x/s \rfloor + \lfloor y/s \rfloor \text{ is odd} \end{cases}$$

What's wrong?



Interpolate after projection

Correct

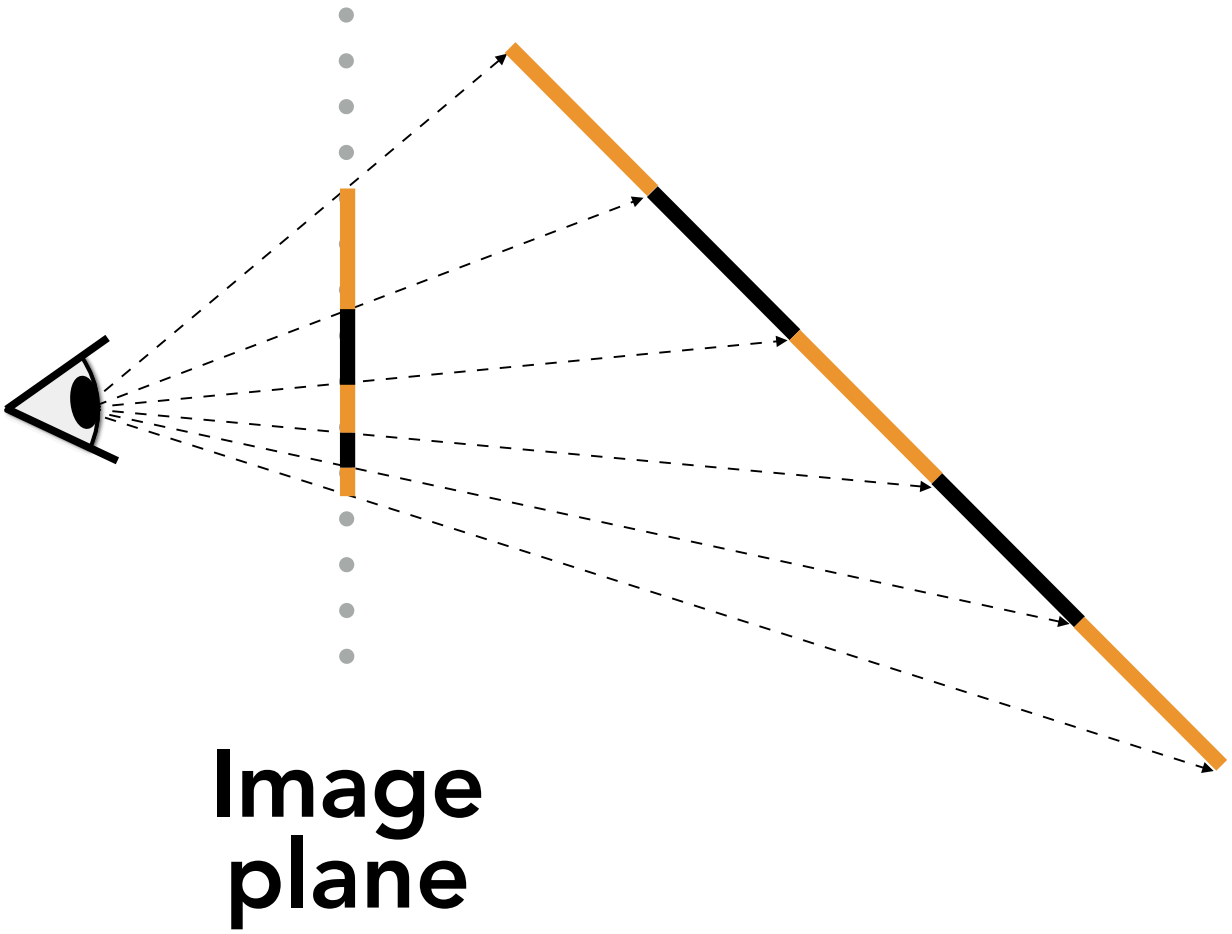


Interpolate before projection

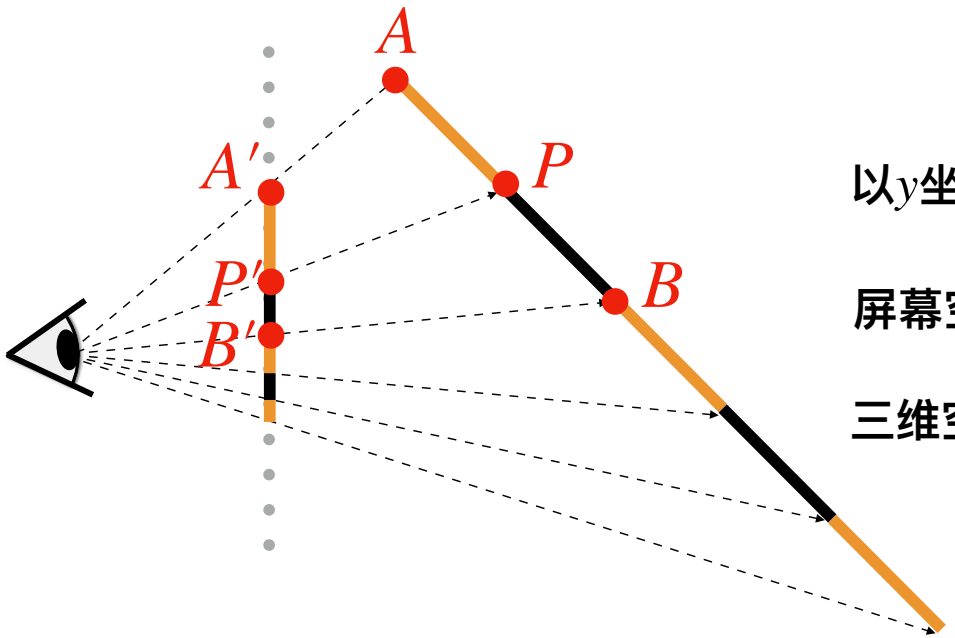
Plane tilted down with perspective projection

Perspective Projection Creates Non Linearity

Linear interpolation in world coordinates yields nonlinear interpolation in screen coordinates!



Perspective-Correct Interpolation



以y坐标为例，由透视变换有 $y' = \frac{n}{z} y$

屏幕空间线性插值有 $y'_P = \alpha y'_A + (1-\alpha) y'_B$

三维空间线性插值有 $y_P = \beta y_A + (1-\beta) y_B$

接下来计算α, β之间的关系：（屏幕空间的插值转换成原三维空间的插值）

展开 $y'_P = \frac{n}{z_P} y_P \longrightarrow \alpha y'_A + (1-\alpha) y'_B = \frac{n}{z_P}[\beta y_A + (1-\beta) y_B]$

同样代入y'_A, y'_B $\longrightarrow$ $\frac{\alpha}{z_A} y_A + \frac{1-\alpha}{z_B} y_B = \frac{1}{z_P}[\beta y_A + (1-\beta) y_B]$

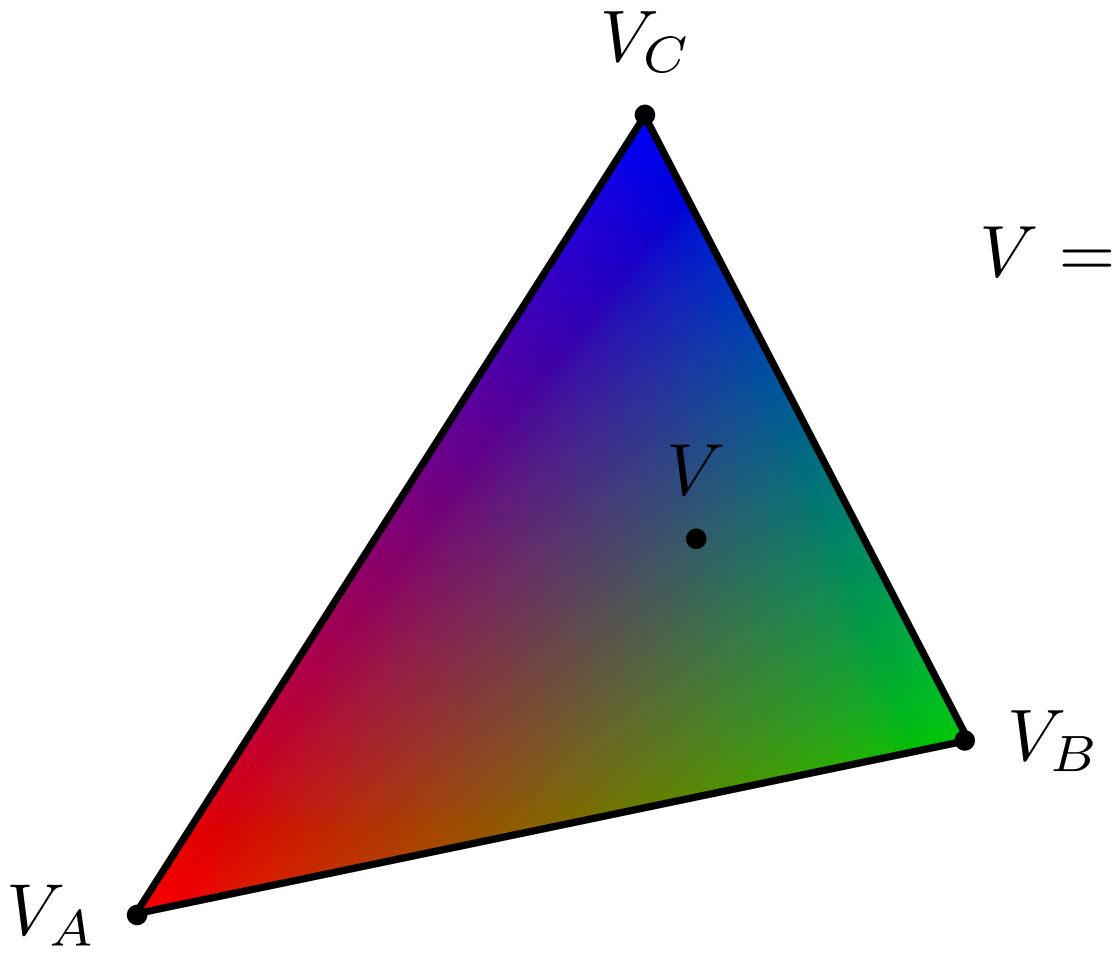
代入z_P(线性插值) $\longrightarrow$ $\frac{\alpha}{z_A} y_A + \frac{1-\alpha}{z_B} y_B = \frac{\beta y_A + (1-\beta) y_B}{\beta z_A + (1-\beta) z_B}$

$$\beta = \frac{(1-\alpha) z_A}{z_B + (1-\alpha)(z_A - z_B)}$$

得到关于α, β的等式，简化后得到

Linear Interpolation Across Triangle

Barycentric coords linearly interpolate values at vertices



$$V = \alpha V_A + \beta V_B + \gamma V_C$$

V_A, V_B, V_C can be positions, texture coordinates, color, normal vectors, material attributes…

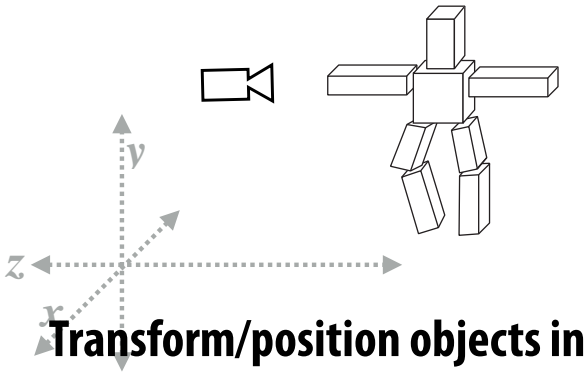
Today's Topics

Visibility/Occlusion (Z-Buffering)

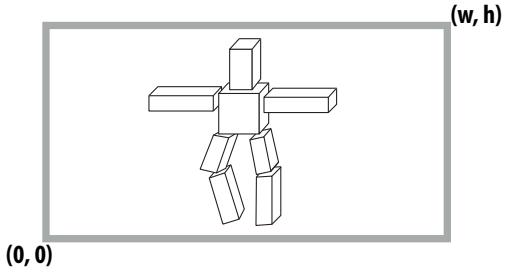
Barycentric Coordinates

Shading

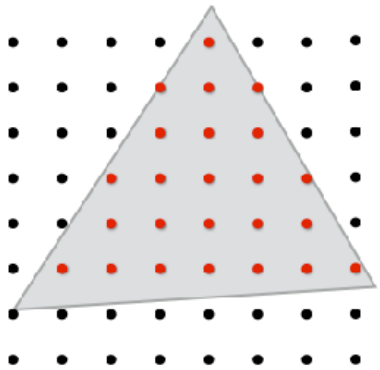
What We've Covered So Far



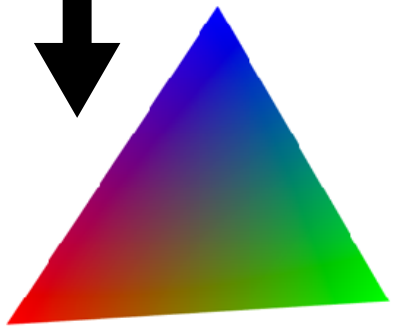
Transform/position objects in the world



Project objects onto the screen



Sample triangle coverage

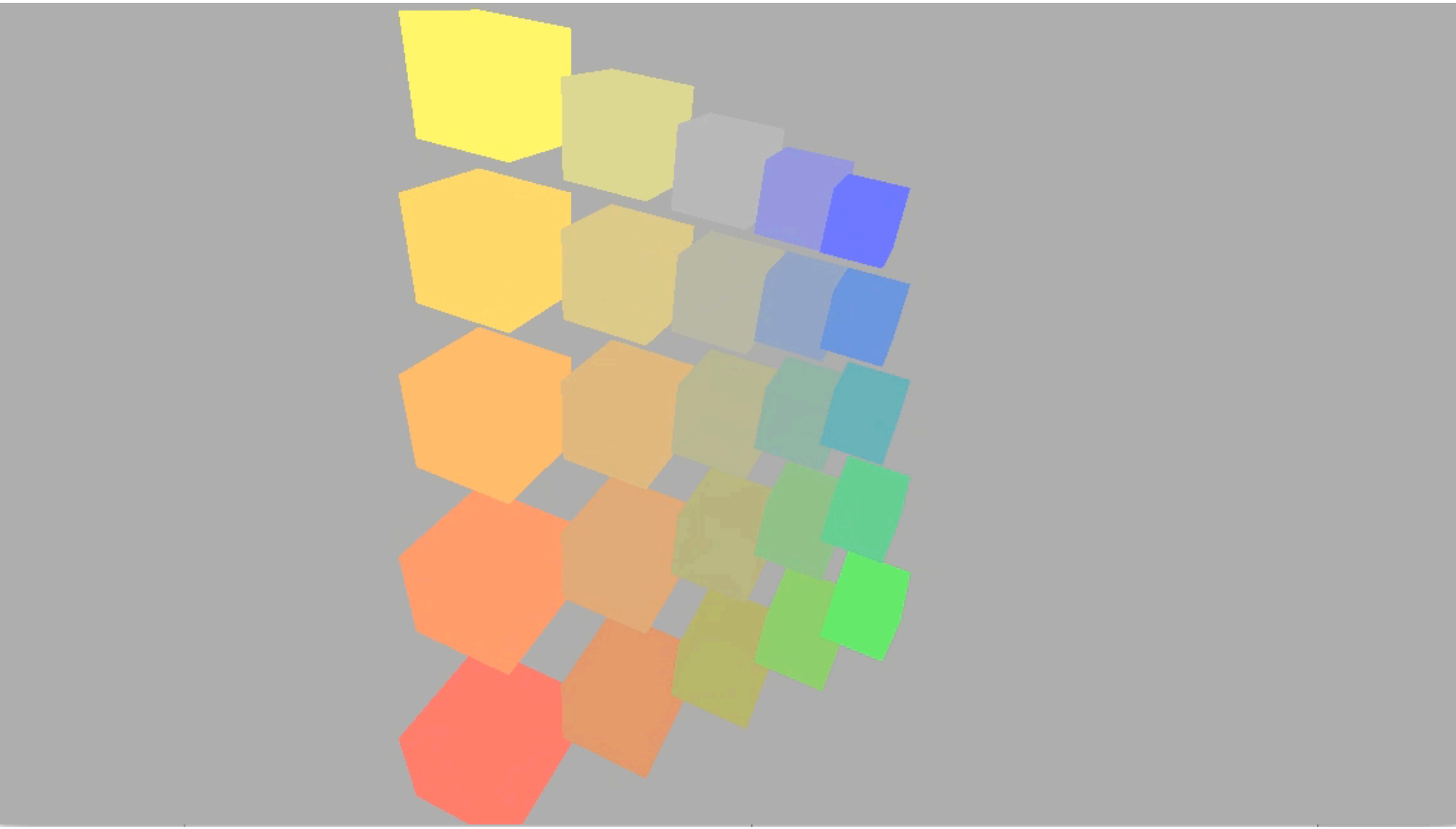


Interpolate triangle attributes at covered samples

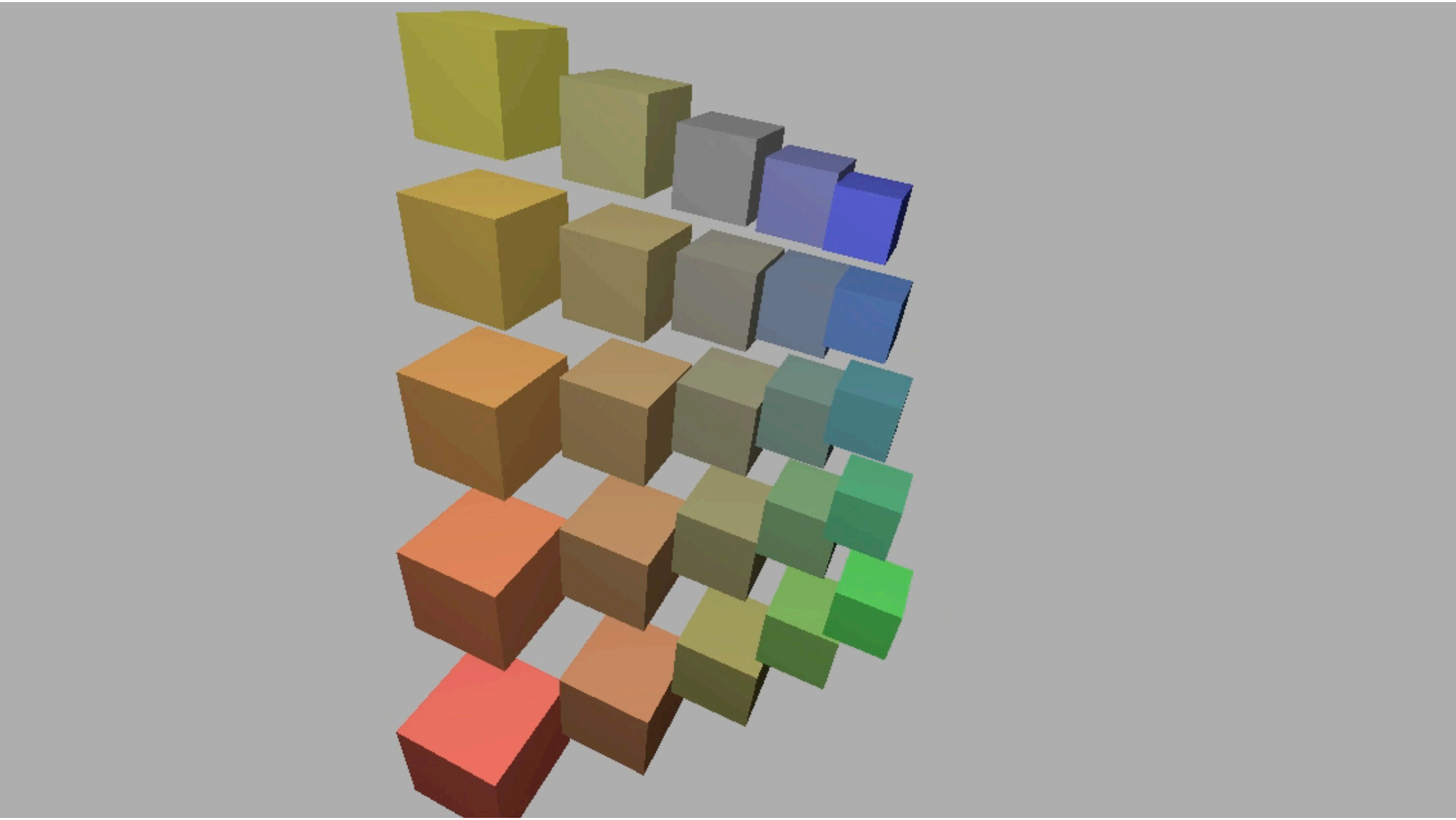
Sample texture maps / evaluate shaders

Combine samples into final image (depth, alpha, …)

Rotating Cubes in Perspective



Rotating Cubes in Perspective



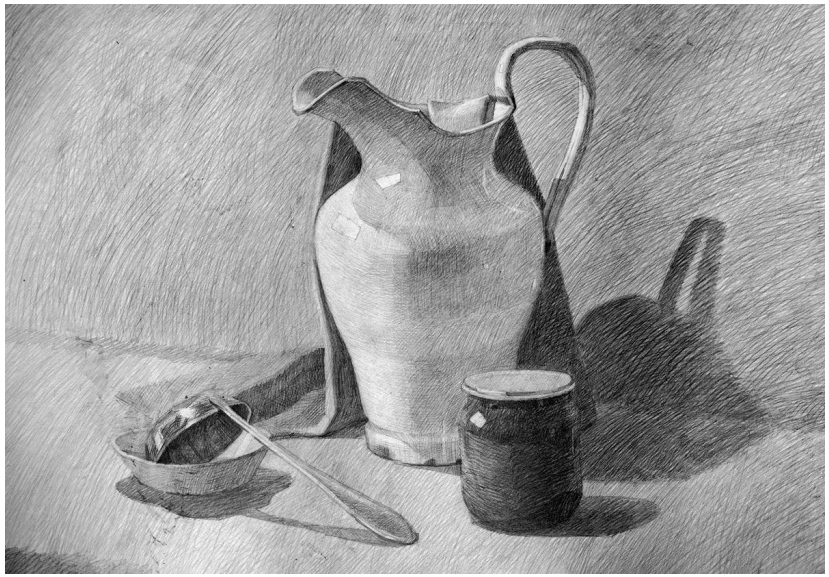
What Else Are We Missing?

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

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.

Simple Shading vs Realistic Lighting & Materials

What we will cover today

- A local shading model: simple, per-pixel, fast
- Based on perceptual observations, not physics

What we will cover later in the course

- Physics-based lighting and material representations
- Global light transport simulation

Simple Shading
(Blinn-Phong Reflection Model)

Perceptual Observations

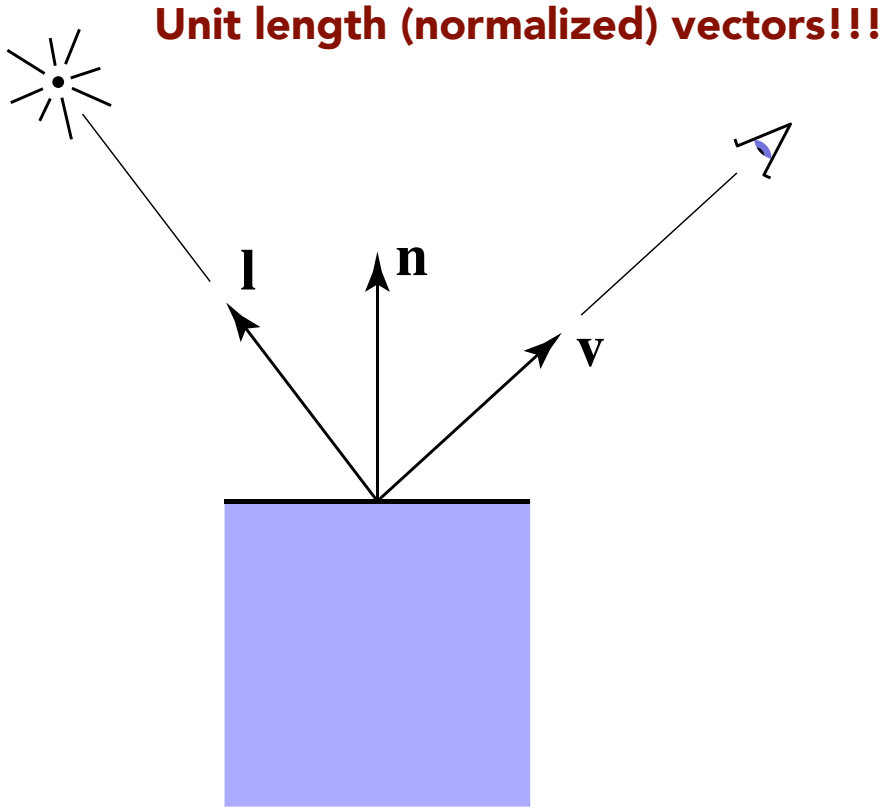
Photo credit: Jessica Andrews, flickr

Local Shading

Compute light reflected toward camera

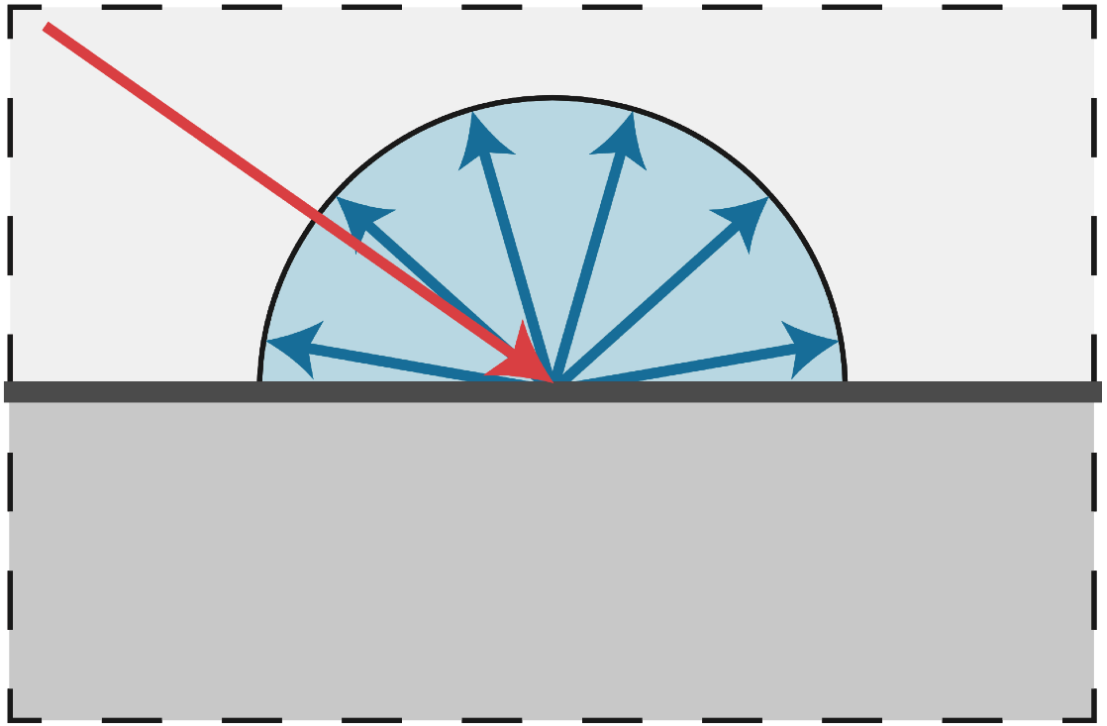
Inputs:

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

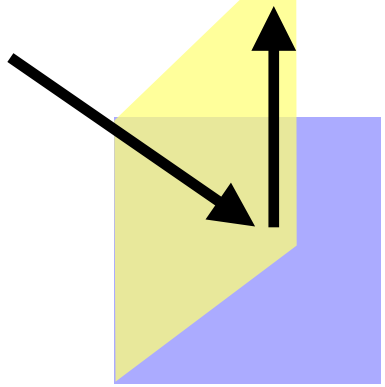


Diffuse Reflection

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



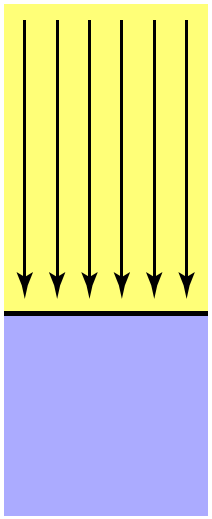
Diffuse Reflection



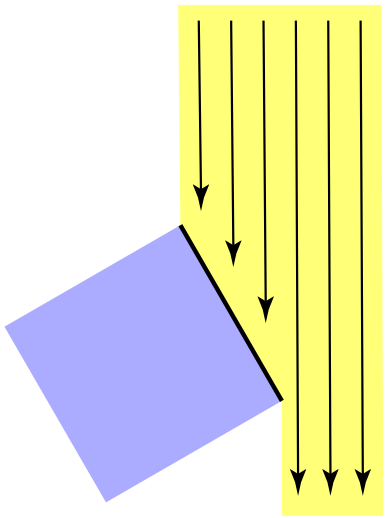
Light is scattered uniformly in all directions

- Surface color is the same for all viewing directions

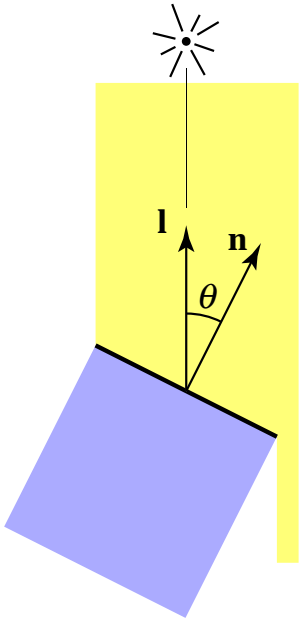
Lambert's cosine law



Top face of cube receives a certain amount of light



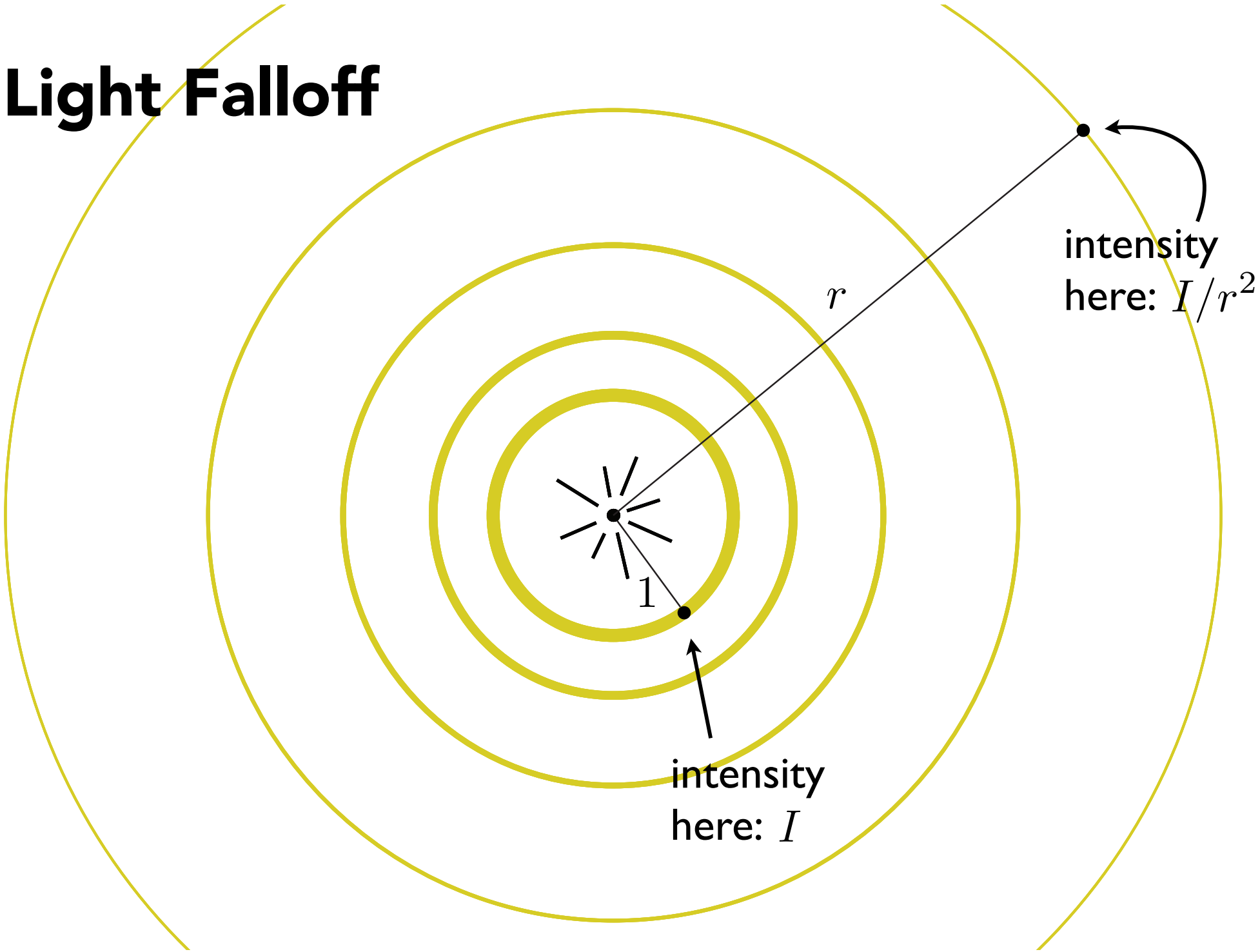
Top face of 60° rotated cube intercepts half the light



In general, light per unit area is proportional to $\cos\theta = \mathbf{l} \cdot \mathbf{n}$

How much light (energy) is received at a shading point?

Light Falloff



Falloff

Physically correct: $1/r^2$ light intensify falloff

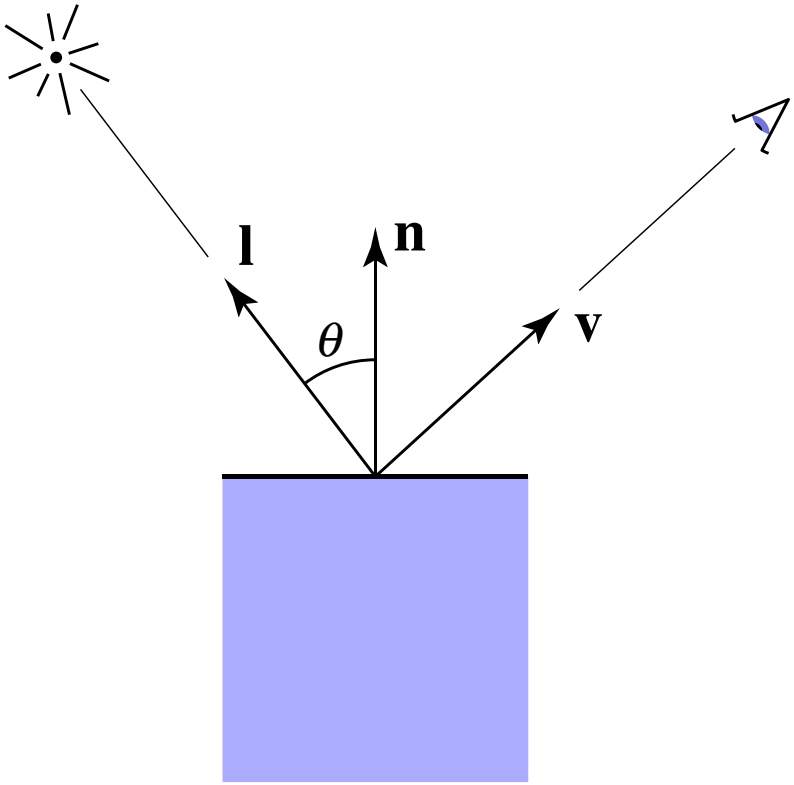
- Tends to look bad with local shading (why?)

Sometimes compromise of $1/r$ used.

Very important to use $1/r^2$ for correct global illumination methods.

Lambertian (Diffuse) Shading

Shading independent of view direction



illumination from source

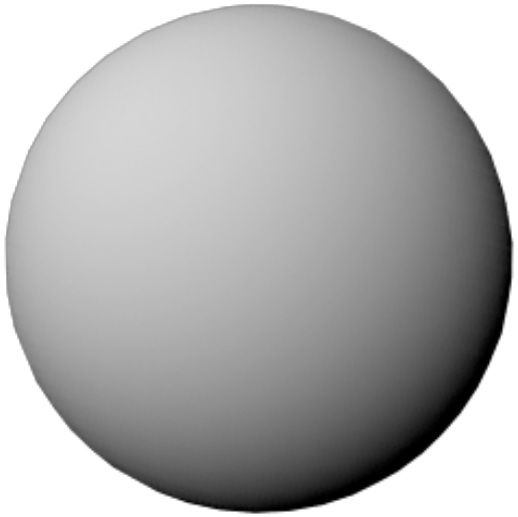
$$L_d = k_d \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l})$$

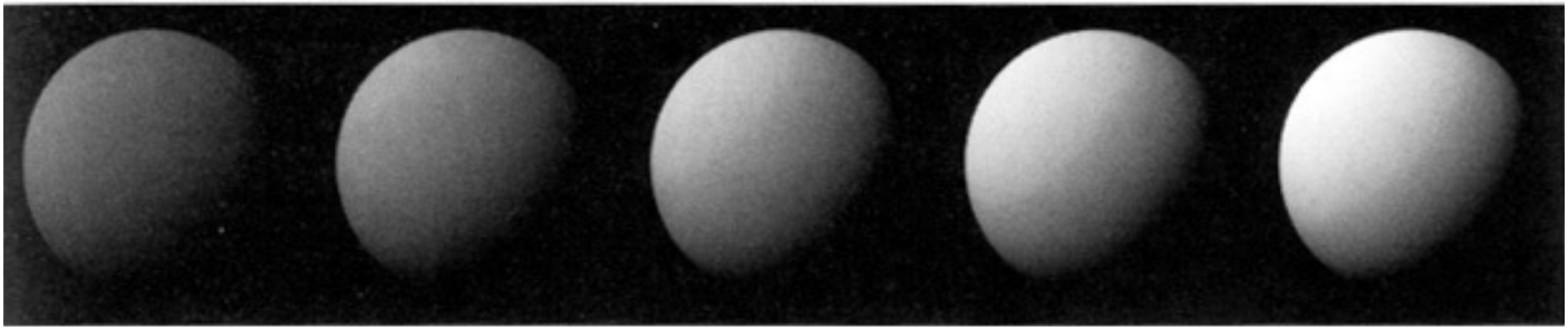
diffuse coefficient

diffusely reflected light

Lambertian (Diffuse) Shading

Produces matte appearance





k_d ⟶

[Foley et al.]

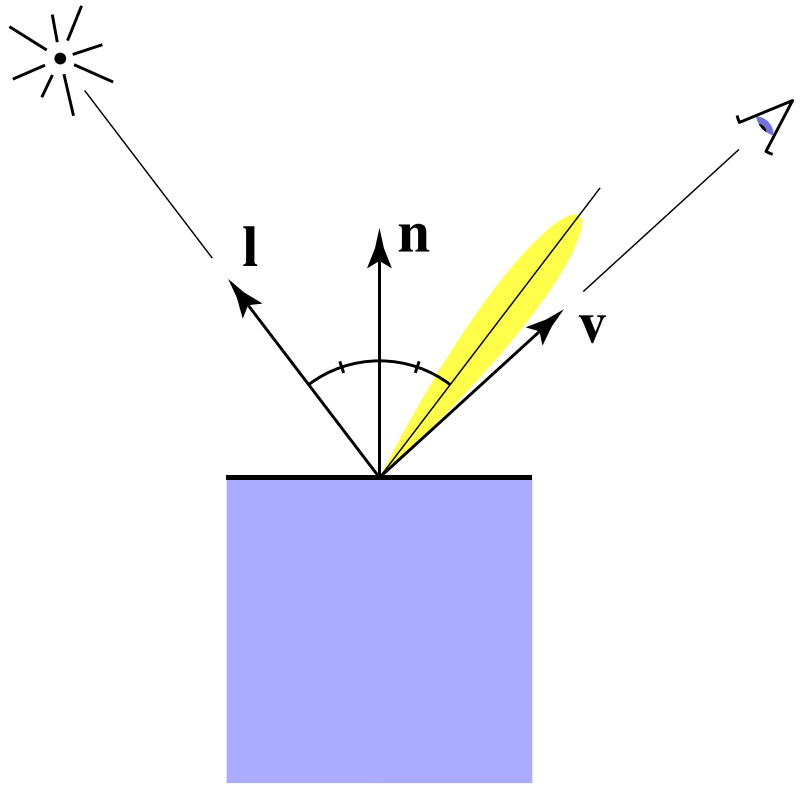
Perceptual Observations

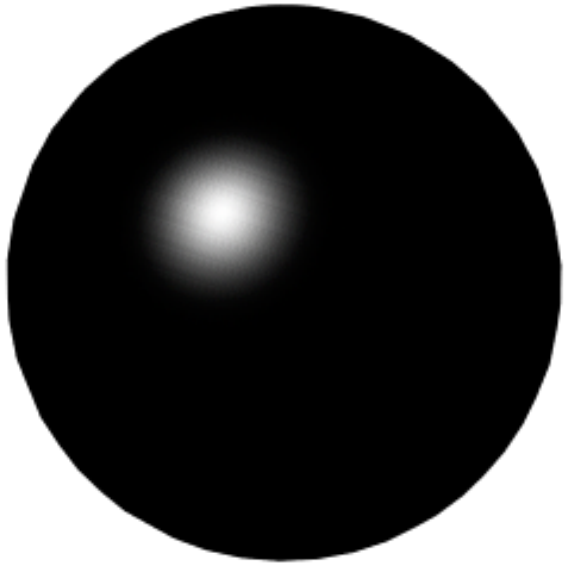
Photo credit: Jessica Andrews, flickr

Specular Shading (Blinn-Phong)

Intensity depends on view direction

- Bright near mirror reflection direction

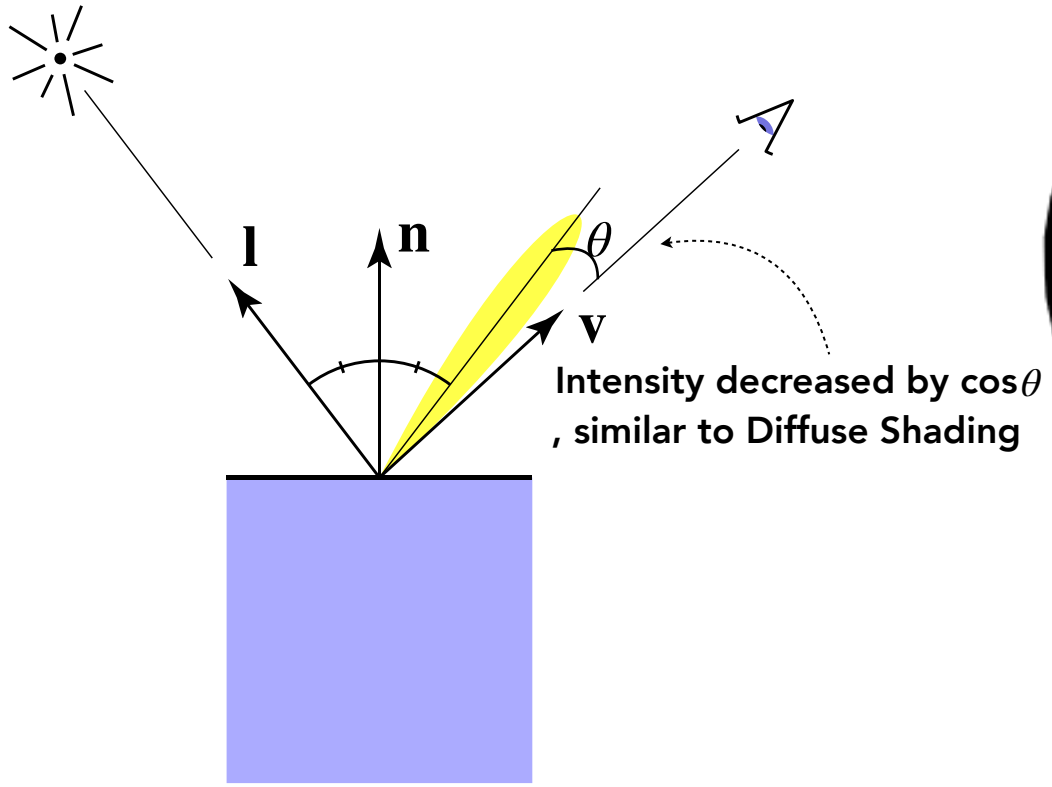


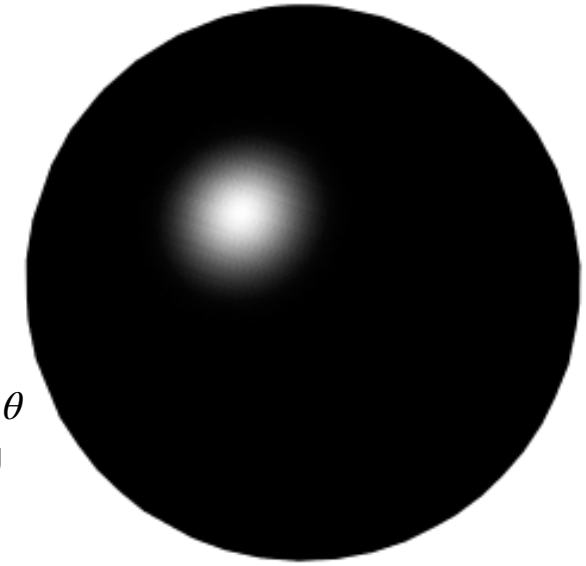


Specular Shading (Blinn-Phong)

Intensity depends on view direction

- Bright near mirror reflection direction

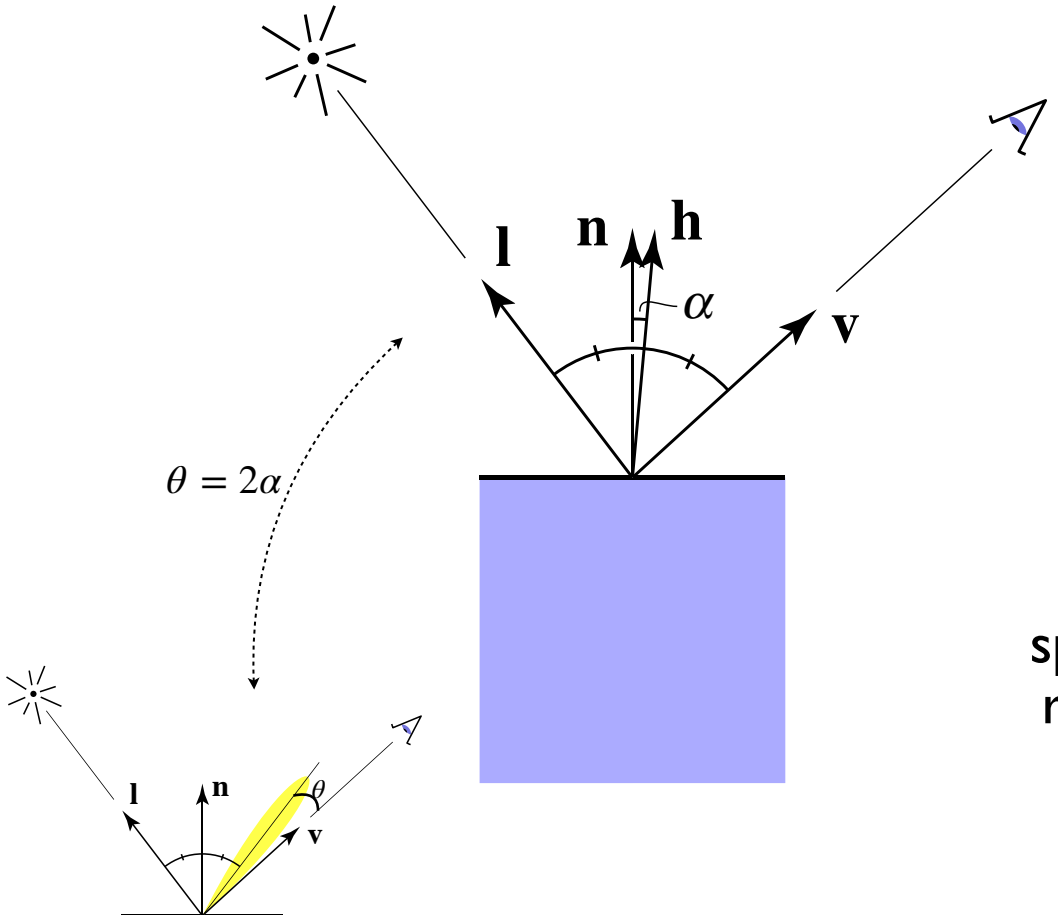




Specular Shading (Blinn-Phong)

Close to mirror direction ⇔ half vector near normal

- Measure "near" by dot product of unit vectors



$$\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l})$$
$$= \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

$$L_s = k_s \left(I/r^2\right) \max(0, \cos\alpha)^p$$
$$= k_s \left(I/r^2\right) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

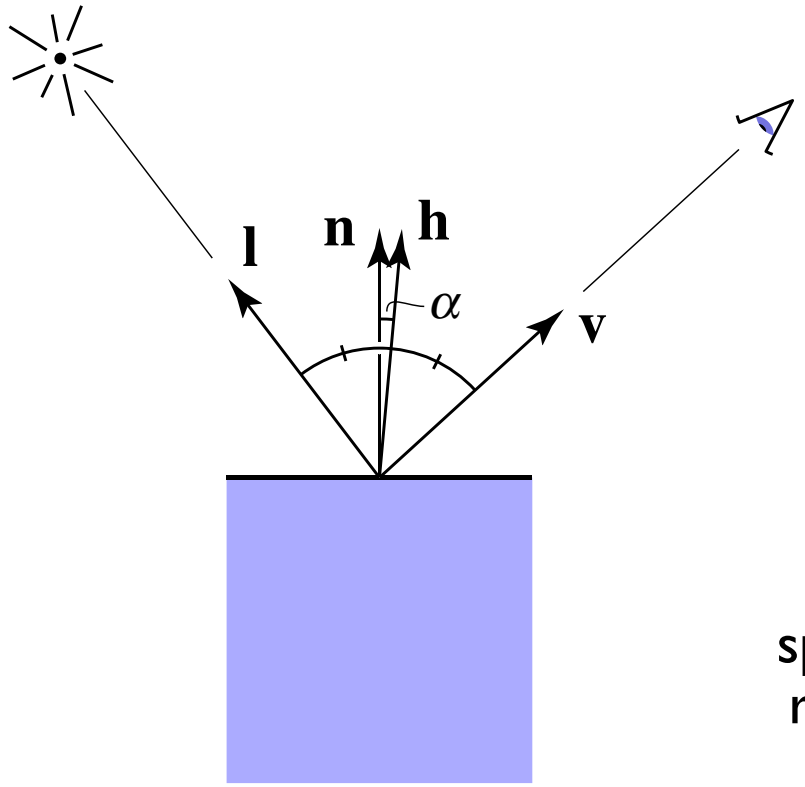
specularly reflected light

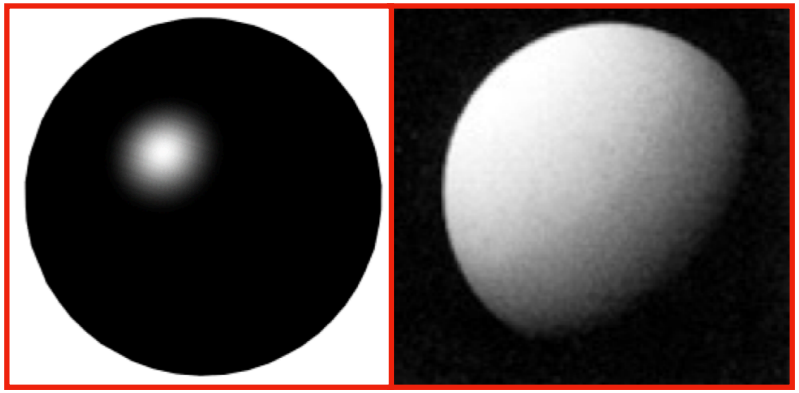
specular coefficient

Specular Shading (Blinn-Phong)

Close to mirror direction ⇔ half vector near normal

- Measure "near" by dot product of unit vectors





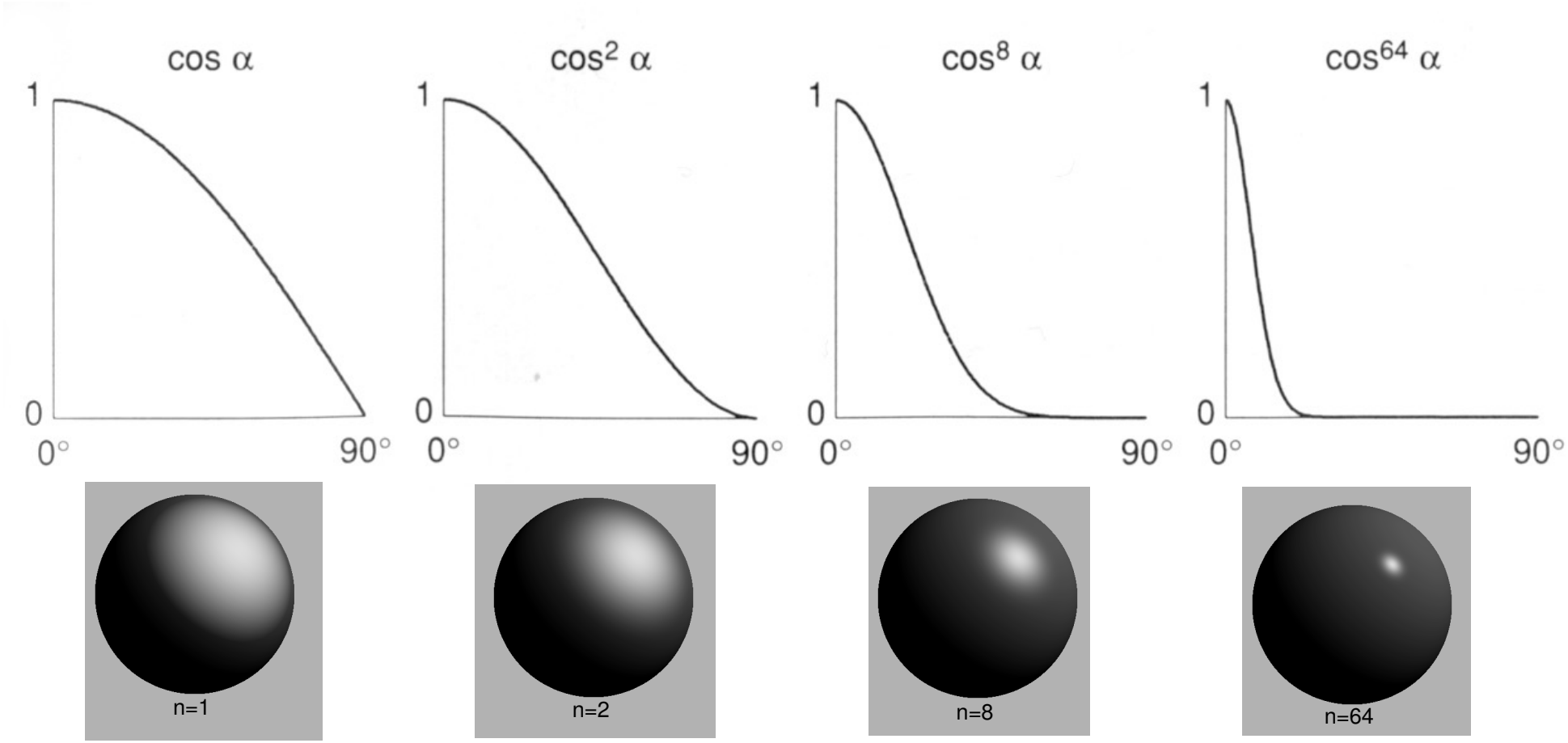
$$L_s = k_s\,(I/r^2)\max(0, \cos\alpha)^p$$
$$= k_s\,(I/r^2)\max(0, \mathbf{n}\cdot\mathbf{h})^p$$

specularly reflected light

specular coefficient

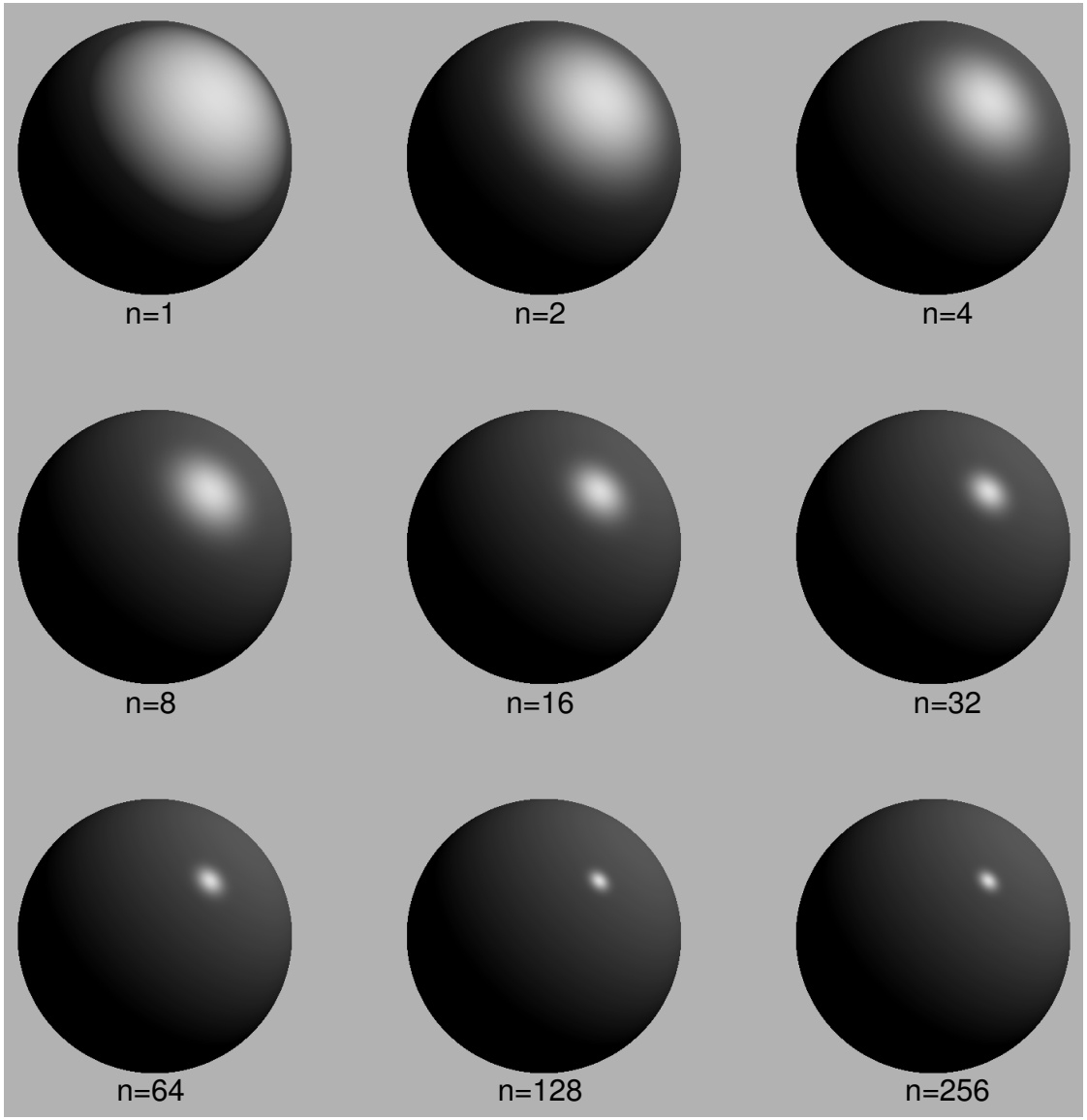
Cosine Power Plots

Increasing p narrows the reflection lobe



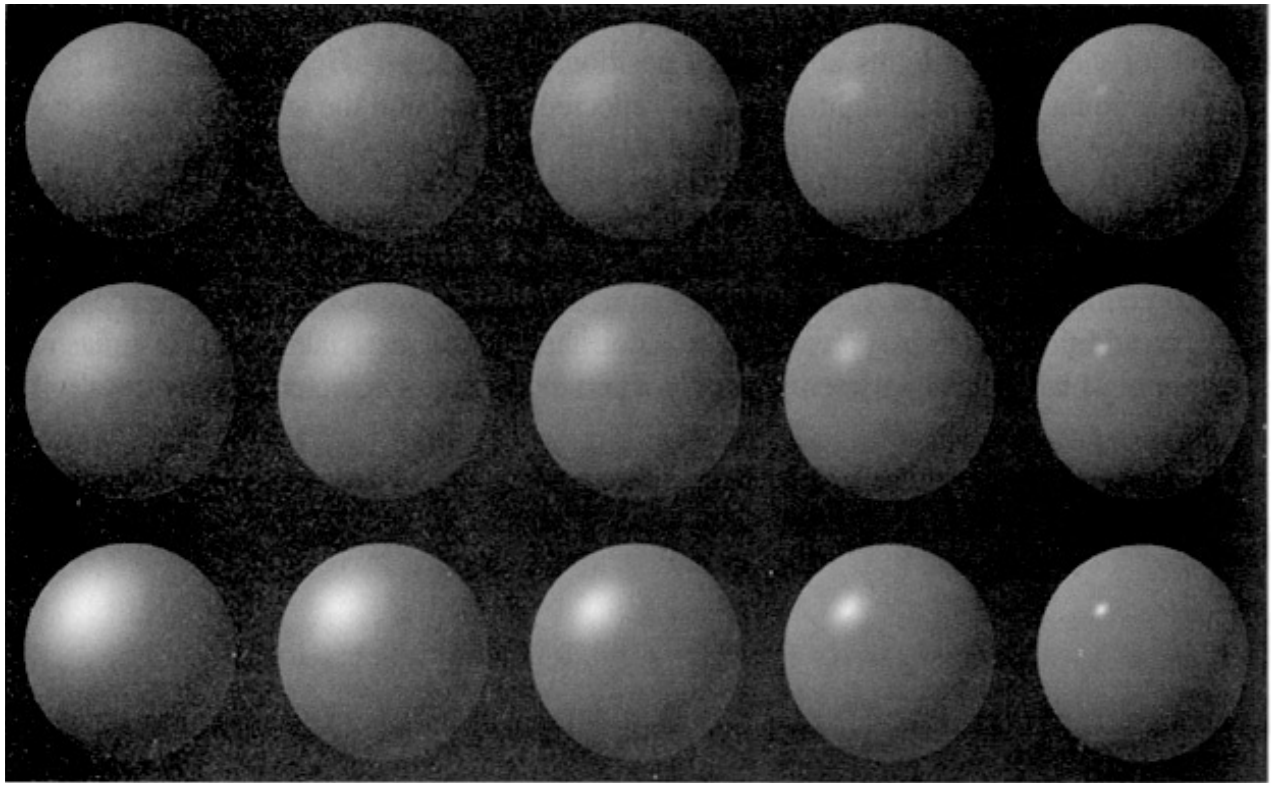
[Foley et al.]

Specular Shading (Blinn-Phong)



Specular Shading (Blinn-Phong)

$$L_s = k_s \left(I/r^2\right) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$



k_s

p

[Foley et al.]

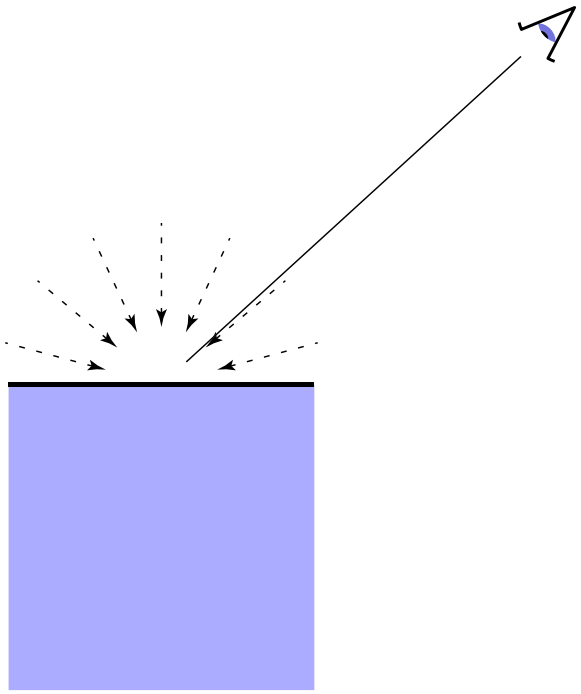
Perceptual Observations

Photo credit: Jessica Andrews, flickr

Ambient Shading

Shading that does not depend on anything

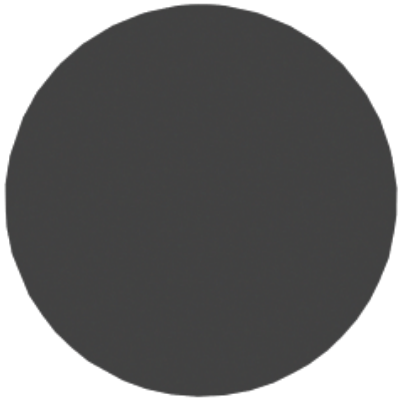
- Add constant color to account for disregarded illumination and fill in black shadows



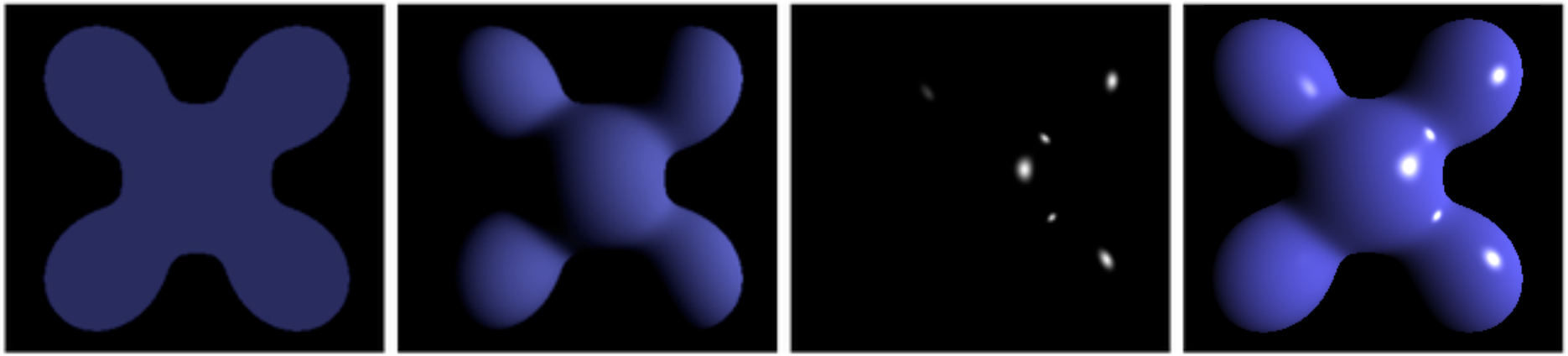
$$L_a = k_a \, I_a$$

ambient coefficient

reflected ambient light



Blinn-Phong Reflection Model



Ambient + Diffuse + Specular = Phong Reflection

$$L = L_a + L_d + L_s$$
$$= k_a\, I_a + k_d\, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s\, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

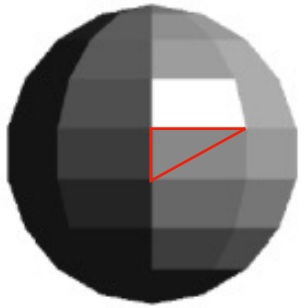
Shading Triangle Meshes

Shading Frequency: Triangle, Vertex or Pixel

Shading Frequency: Triangle, Vertex or Pixel

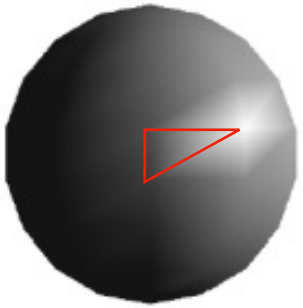
Shade each triangle (flat shading)

- Triangle face is flat — one normal vector
- Not good for smooth surfaces



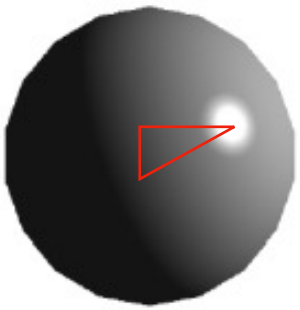
Shade each vertex ("Gouraud" shading)

- Interpolate colors from vertices across triangle
- Each vertex has a normal vector



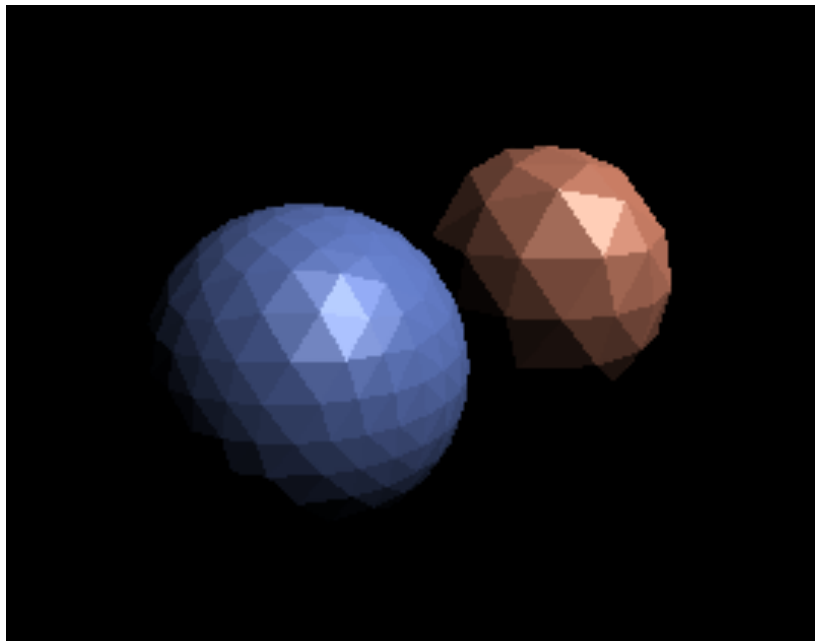
Shade each pixel ("Phong" shading)

- Interpolate normal vectors across each triangle
- Compute full shading model at each pixel



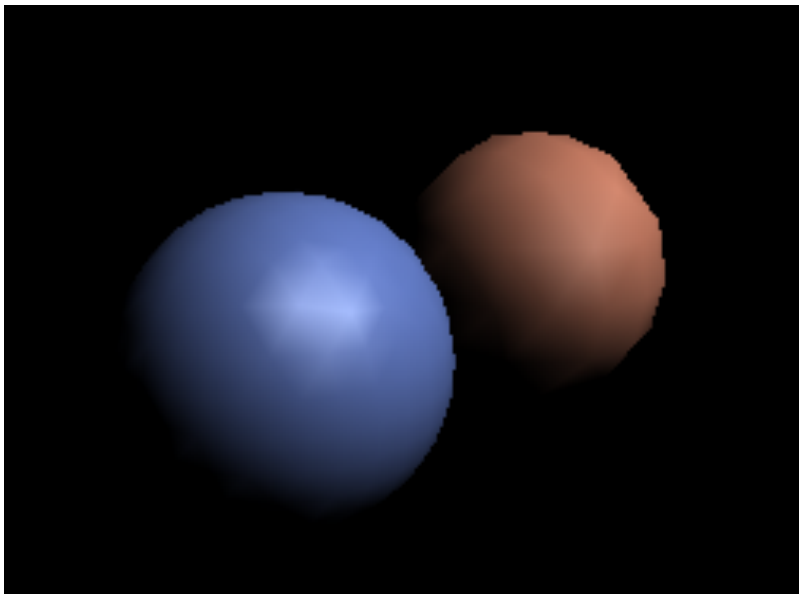
Shade each triangle (Flat shading)

- Flat shading
  - Triangle face is flat — one normal vector
  - Not good for smooth surfaces



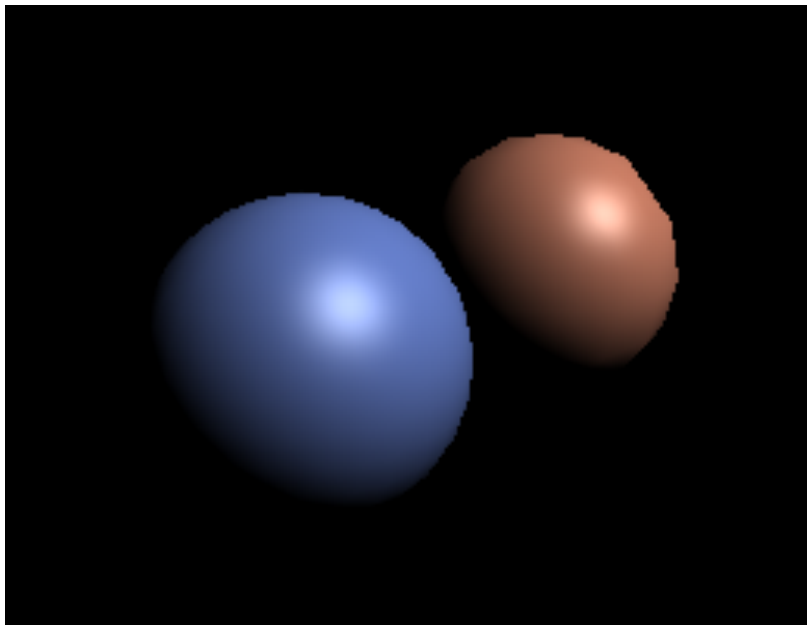
Shade each vertex (Gouraud shading)

- Gouraud shading
  - Interpolate colors from vertices across triangle
  - Each vertex has a normal vector (how?)

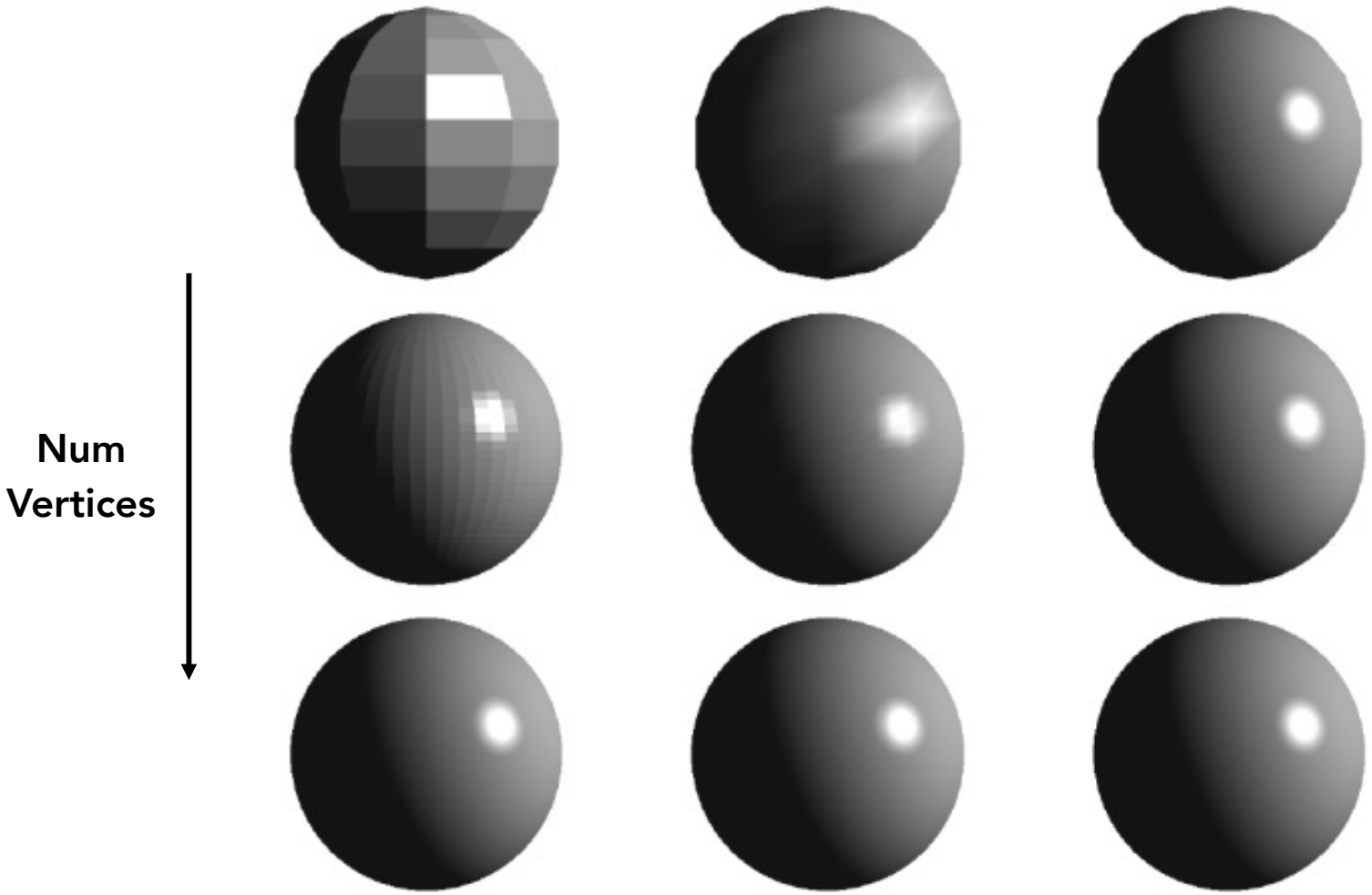


Shade each pixel (Phong shading)

- Phong shading
  - Interpolate normal vectors across each triangle
  - Compute full shading model at each pixel
  - Not the Blinn-Phong Reflectance Model



Shading Frequency: Face, Vertex or Pixel

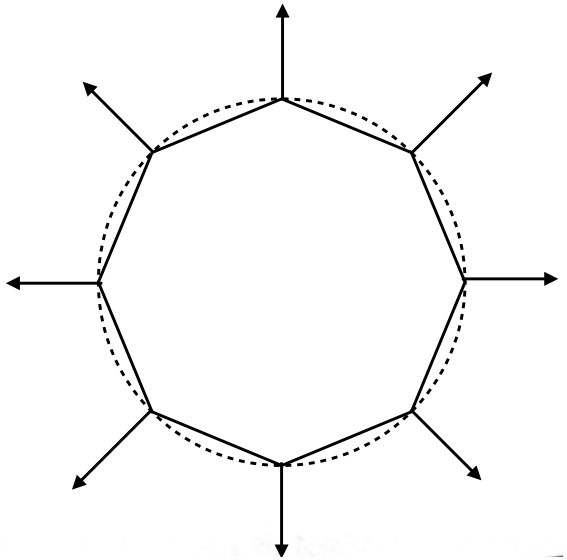


Shading freq. :	Face	Vertex	Pixel
Shading type :	Flat	Gouraud	Phong (*)

Defining Per-Vertex Normal Vectors

Best to get vertex normals from the underlying geometry

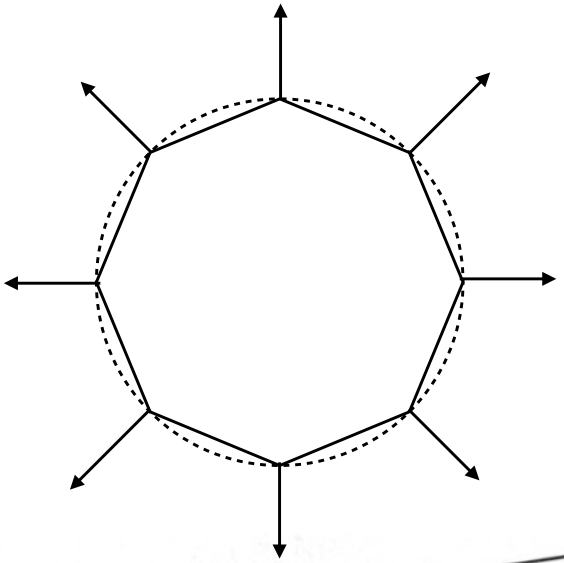
- e.g. consider a sphere



Defining Per-Vertex Normal Vectors

Best to get vertex normals from the underlying geometry

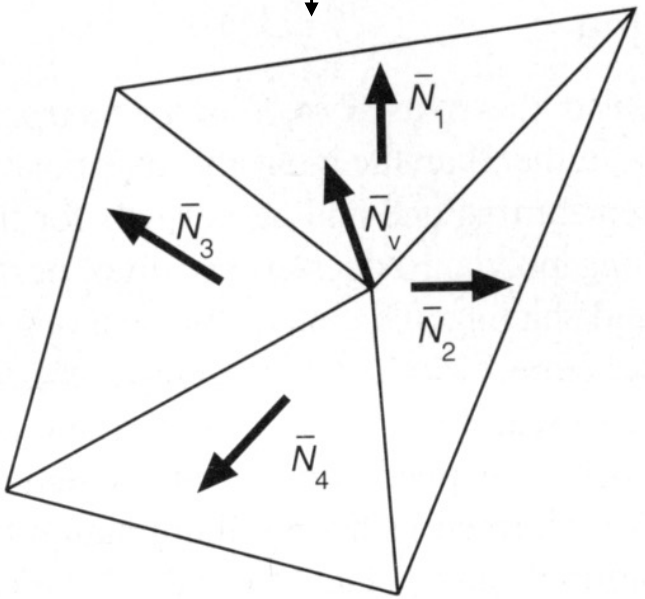
- e.g. consider a sphere



Otherwise have to infer vertex normals from triangle faces

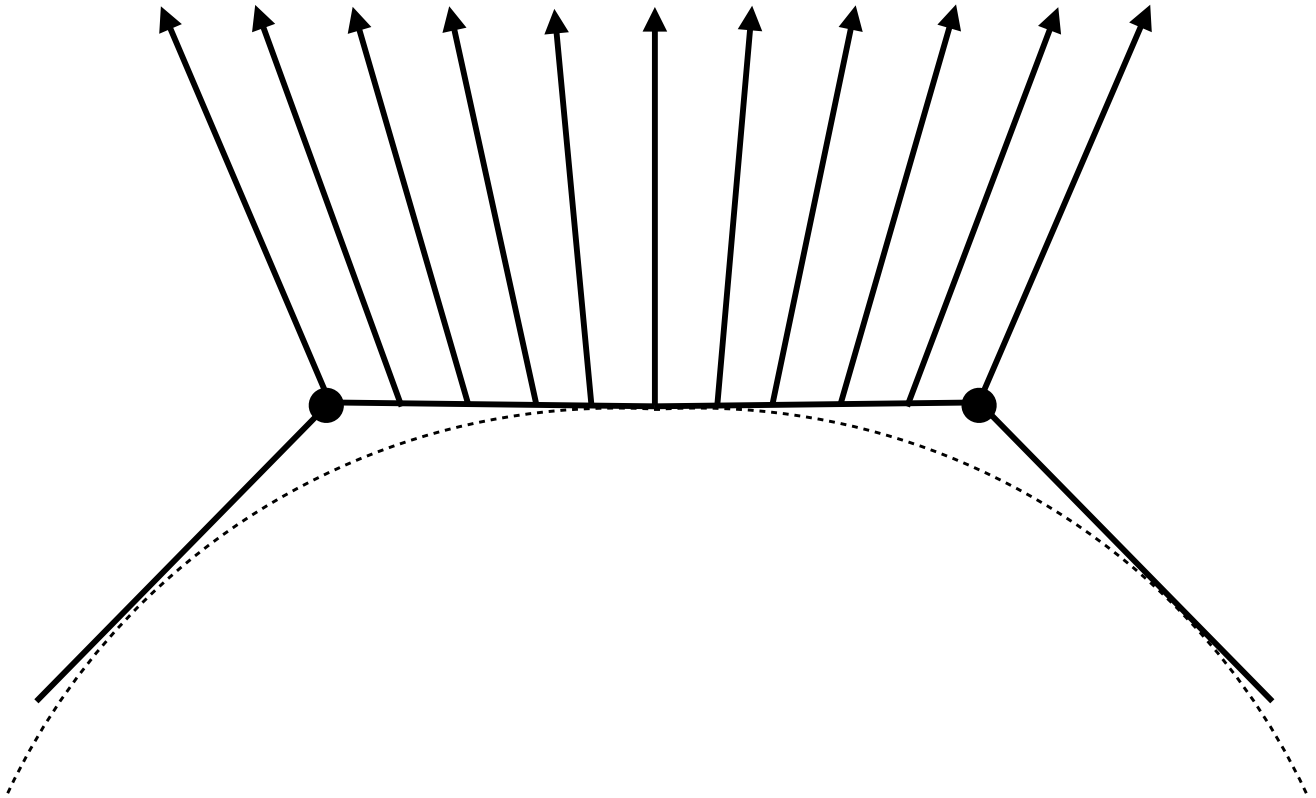
- Simple scheme: average surrounding face normals

$$N_v = \frac{\sum_i N_i}{\| \sum_i N_i \|}$$



Defining Per-Pixel Normal Vectors

Barycentric interpolation of vertex normals



Problem: length of vectors?

Things to Remember

Visibility

- Painter's algorithm and Z-Buffer algorithm

Simple Shading Model

- Key geometry: lighting, viewing & normal vectors
- Ambient, diffuse & specular reflection functions
- Shading frequency: triangle, vertex or fragment

Barycentric coordinates

- Interpolation across triangles

Acknowledgments

Thanks to the CMU 15-462, UC Berkeley CS184 and GAMES 101 courses for the slides.