418382 สภาพแวดล้อมการทำงานคอมพิวเตอร์กราฟิกส์
การบรรยายครั้งที่ 4

ประมุข ขันเงิน

pramook@gmail.com

RASTERIZATION

Projection Transformation

- เปลี่ยน eye space เป็น clip space
- พิกัดใน clip space จะใช้เป็นตัวบอกว่าเราจะเห็น vertex ใด หรือไม่เห็น vertex ใด
- กระบวนการตัดสินใจ: vertex ที่เห็นจะต้องมี
  - $-1 \le x \le 1$
  - $-1 \le y \le 1$
  - $-1 \le z \le 1$
- Projection transform ยังมีผลต่อลักษณะภาพที่เราเห็นอีกด้วย

Projection Transform ใน OpenGL

- OpenGL จะจำ matrix ของ projection transform เอาไว้
- เวลาต้องการเปลี่ยนแปลง projection matrix ให้เปลี่ยน mode ของ matrix เป็น GL_PROJECTION ด้วยคำสั่ง glMatrixMode(GL_PROJECTION);
- หลังจากนั้นใช้คำสั่งในการเปลี่ยนแปลง matrix อื่นแบบเดิม เช่น glLoadIdentity(), glMultMatrix(…), ฯลฯ
- ส่วนมากเราจะสั่ง glLoadIdentity() ทันทีหลังจากสั่ง glMatrixMode(GL_PROJECTION) เสร็จแล้ว เพื่อเคลียร์ค่า projection matrix ก่อนใส่ค่าใหม่

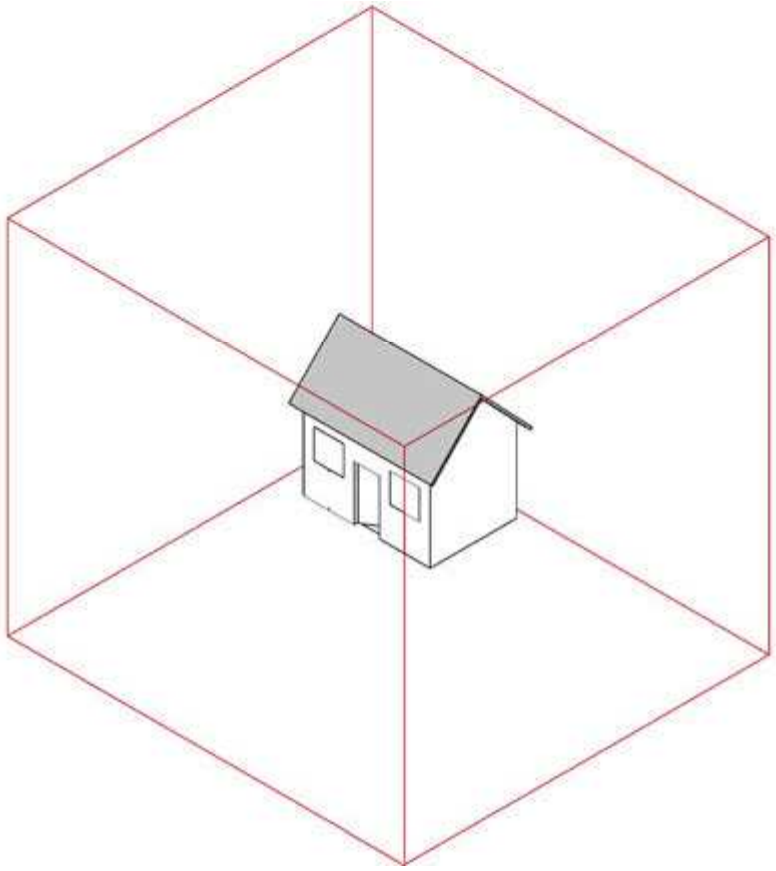
Projection Transformation ที่สำคัญ 2 แบบ

- Orthographic Projection
- Perspective Projection

Orthographic Projection

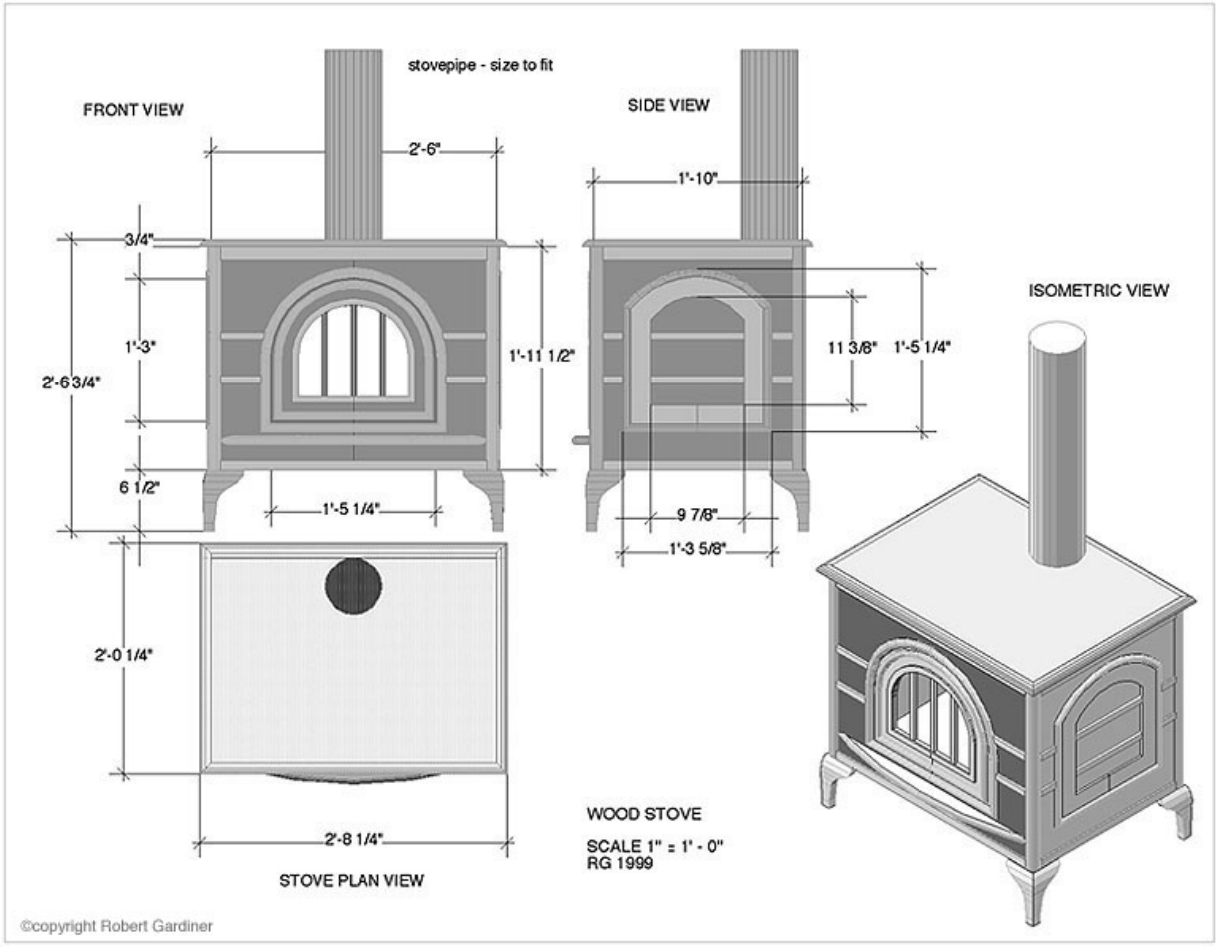
- ปริมาตรของบริเวณที่เห็นเป็นปริซึม
- ไม่มี foreshortening กล่าวคือ ไม่ว่าวัตถุจะอยู่ใกล้ไกลก็เห็นขนาดเท่ากันหมด
- หลังจากฉาก เส้นขนานยังเป็นเส้นขนานอยู่
- ใช้ในโปรแกรมช่วยเขียนแบบ/CAD เนื่องจากขนาดของวัตถุเป็นเรื่องสำคัญ

Orthographic Projection (ต่อ)



http://www2.arts.ubc.ca/TheatreDesign/crslib/drft_1/orthint.htm

Orthographic Projection (ต่อ)



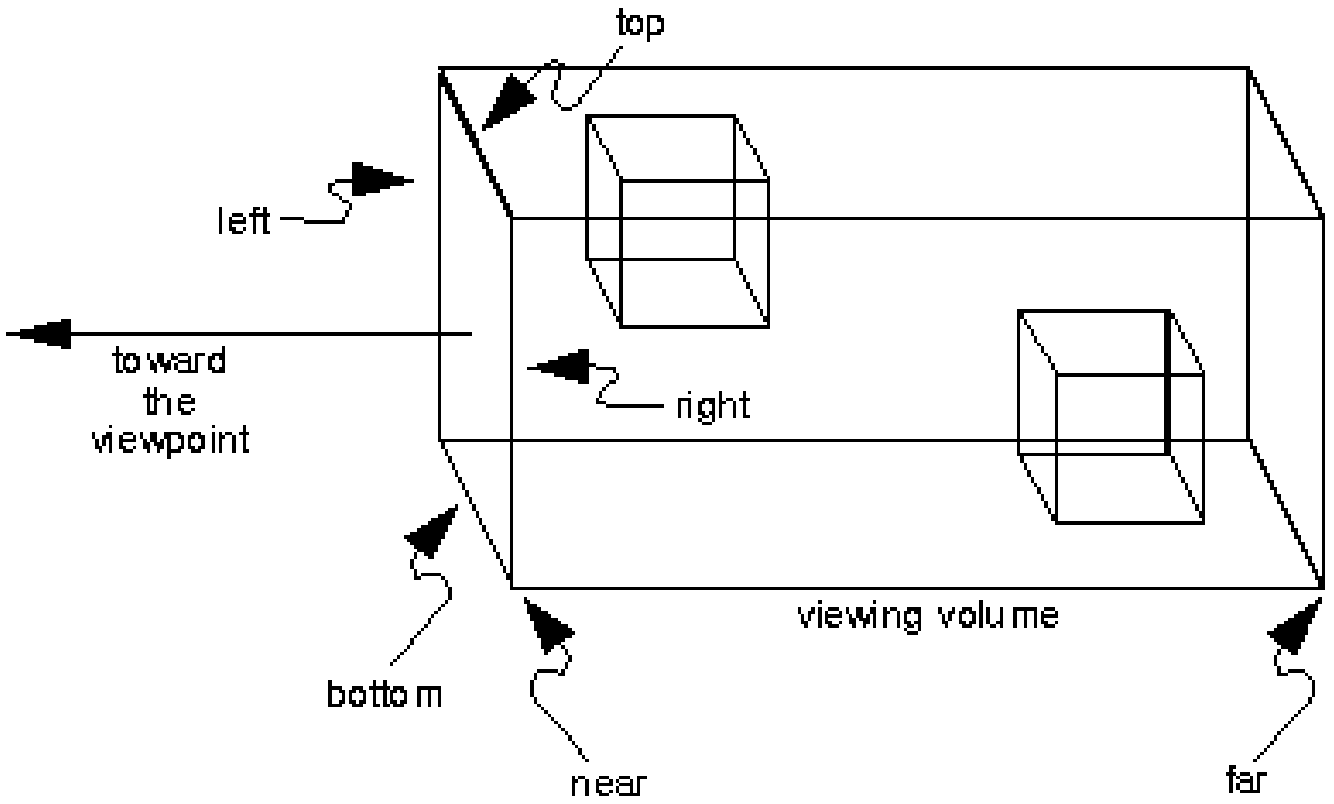
http://www2.arts.ubc.ca/TheatreDesign/crslib/drft_1/cad/wdstv.htm

การนิยาม Orthographic Projection

- นิยามได้โดยการนิยามปริซึมของปริมาตรที่เราต้องการมองเห็น
- ปริซึมนี้สามารถนิยามได้ด้วยตัวเลข 3 คู่
  - left และ right --- ขอบเขตในแนวแกน x
  - top และ bottom --- ขอบเขตในแนวแกน y
  - near และ far --- ขอบเขตในแนวแกน -z (เพราะเรามองในแนว -z)
- ค่าทั้งหกเป็นพิกัดใน eye space
- ปริซึมที่นิยามคือ

$$\{(x,y,z) : left \le x \le right,\ top \le y \le bottom,$$
$$near \le -z \le far\}$$

ปริซึมปริมาตรที่มองเห็น



การนิยาม Orthographic Projection (ต่อ)

- Matrix ของ orthographic projection ต้องทำอะไรบ้า
  - ส่ง x = left ไป x = -1
  - ส่ง x = right ไป x = 1
  - ส่ง y = bottom ไป y = -1
  - ส่ง y = top ไป y = 1
  - ส่ง z = -far ไป z = 1
  - ส่ง z = -near ไป z = -1

Matrix ของ Orthographic Projection

$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

คำสั่ง OpenGL เกี่ยวกับ Orthographic Projection

- glOrtho(left, right, bottom, top, near, far)
  - คูณ matrix ปัจจุบันด้วย matrix ของ orthographic projection ในหน้าก่อน
  - ก่อนใช้ควรเรียก

    glMatrixMode(GL_PROJECTION)

    glLoadIdentity()

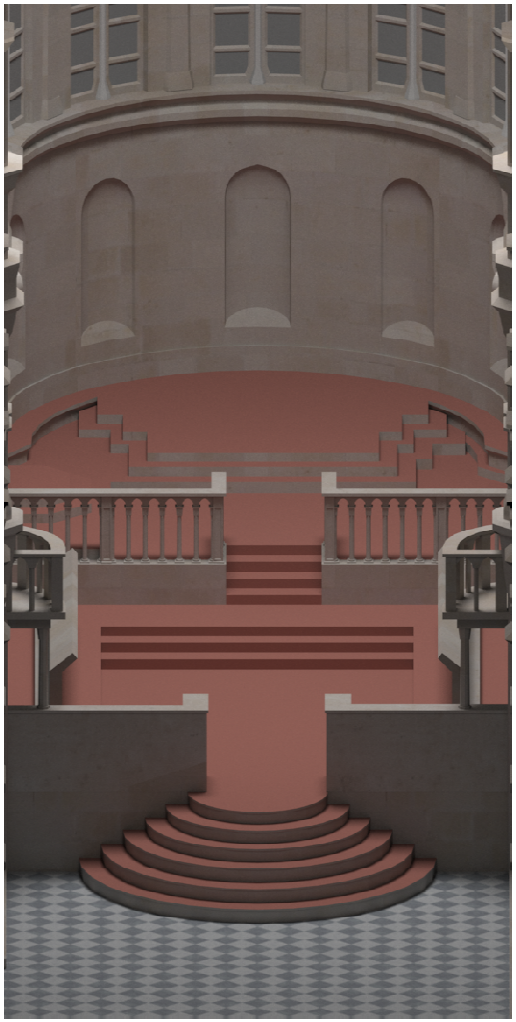
    ก่อนเพื่อเปลี่ยน mode และเคลียร์ค่า projection matrix เดิม
- glOrtho2D(left, right, bottom, top)
  - เหมือนกับ glOrtho แต่ให้ค่า near เป็น 0 และ ค่า far เป็น 1

Perspective Projection

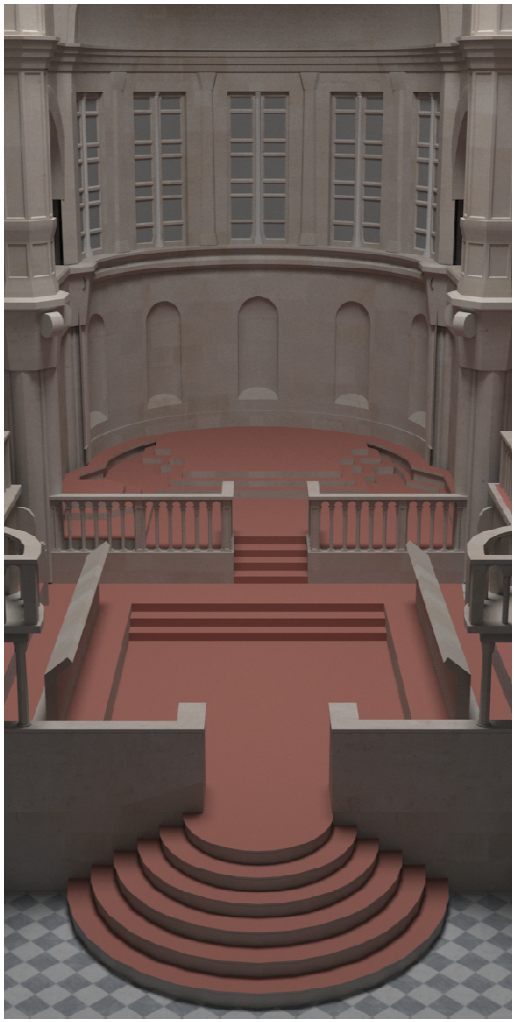
- ปริมาตรของบริเวณที่เห็นเป็น frustum (ปีระมิดยอดตัด)
- มี foreshortening กล่าวคือ อะไรที่อยู่ใกล้จะเห็นใหญ่กว่า
- หลังจากฉายแล้ว เส้นขนานอาจจะไม่ขนานกันเหมือนเดิม
- ให้ความเป็นสามมิติ เพราะเหมือนกับที่ตาคนทำงาน ทำให้เหมือนเข้าไปอยู่ในฉากจริงๆ
- ใช้กับโปรแกรมทางความบันเทิง

Perspective Projection (ต่อ)

Perspective Projection (cont.)



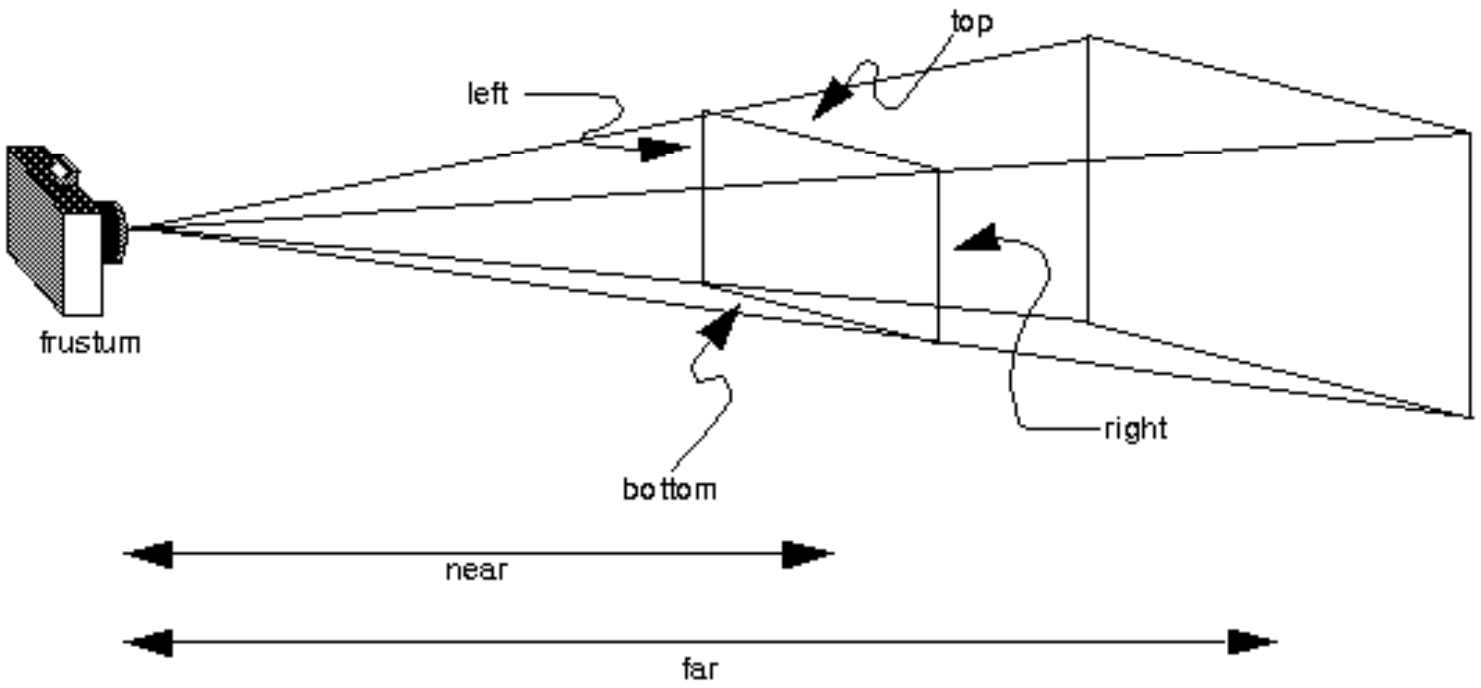
orthographic



perspective

การนิยาม Perspective Projection

- นิยามด้วยเลข 6 ตัวเหมือนกับ orthographic projection



การนิยาม Perspective Projection (ต่อ)

- ปริมาตรที่มองเห็นคือปีระมิดยอดตัดที่มียอดเป็นสี่เหลี่ยม

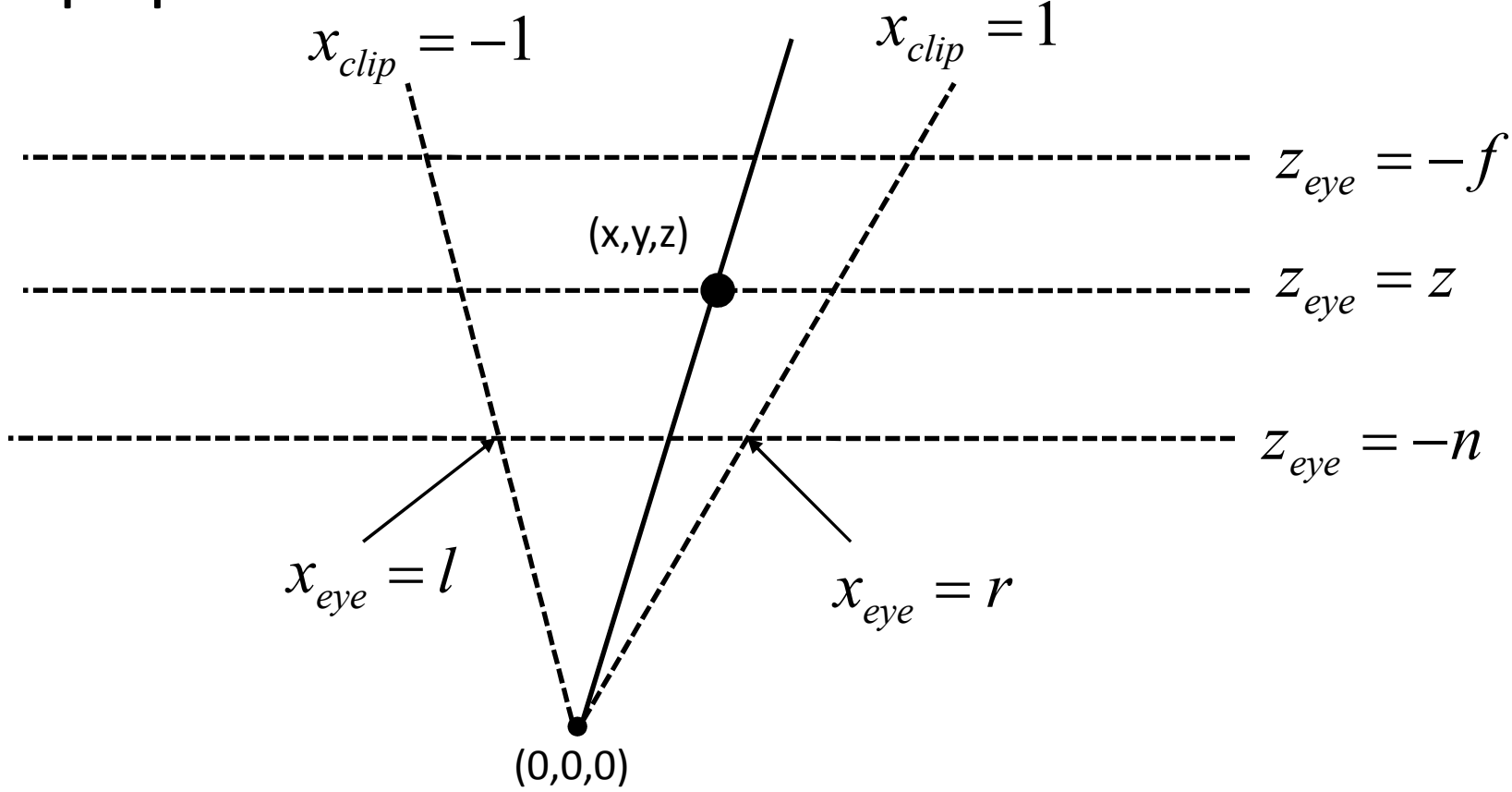
  $\{(x,y,z) : left \le x \le right, bottom \le y \le top,$

  $z = -near\}$

  ซึ่งยอดของมันถูกฉายต่อไปจนถึง $z = -far$

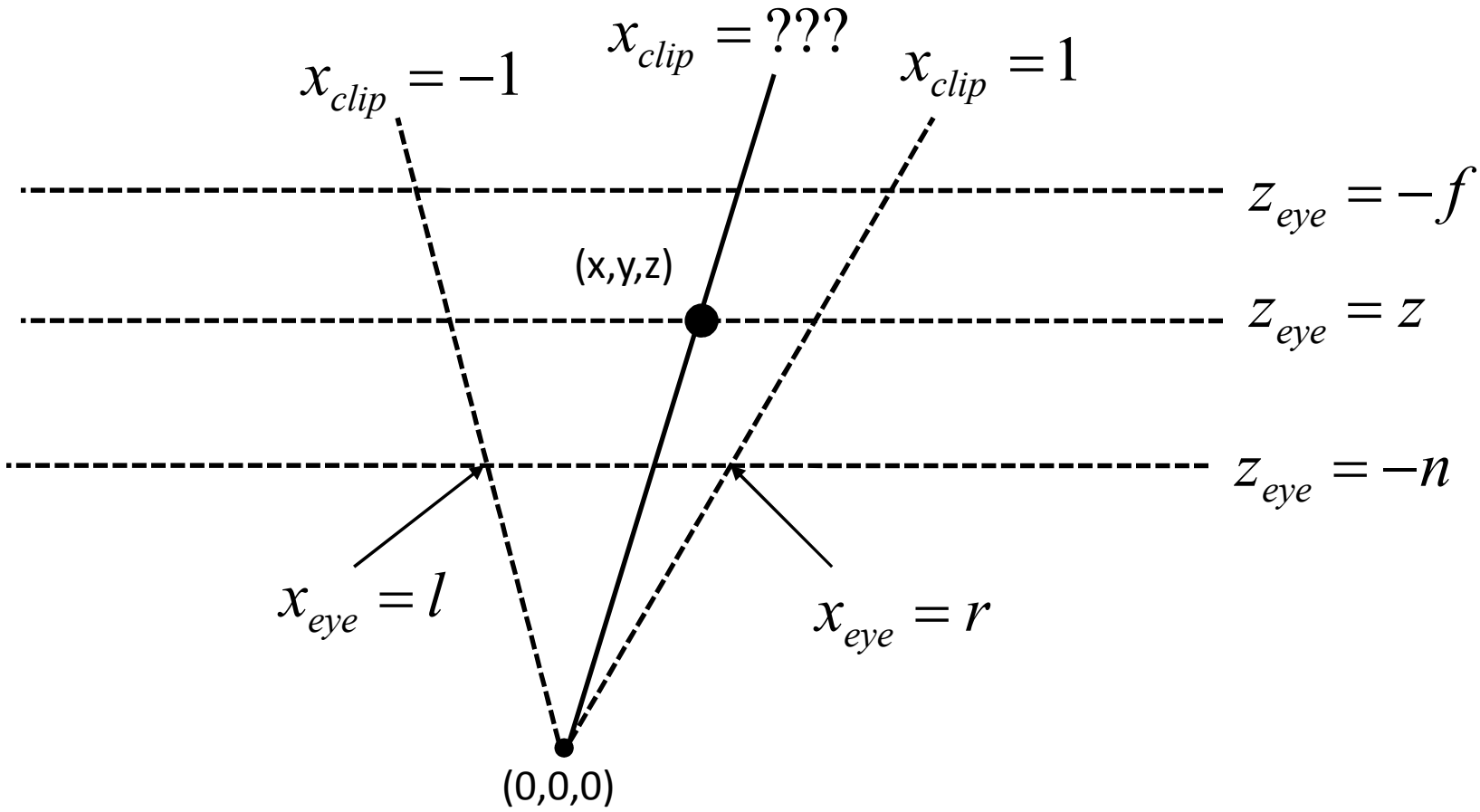
การนิยาม Perspective Projection (ต่อ)

- ให้จุด (x,y,z) มาใน eye space แล้วมันจะถูกแปลงเป็นอะไรใน clip space?



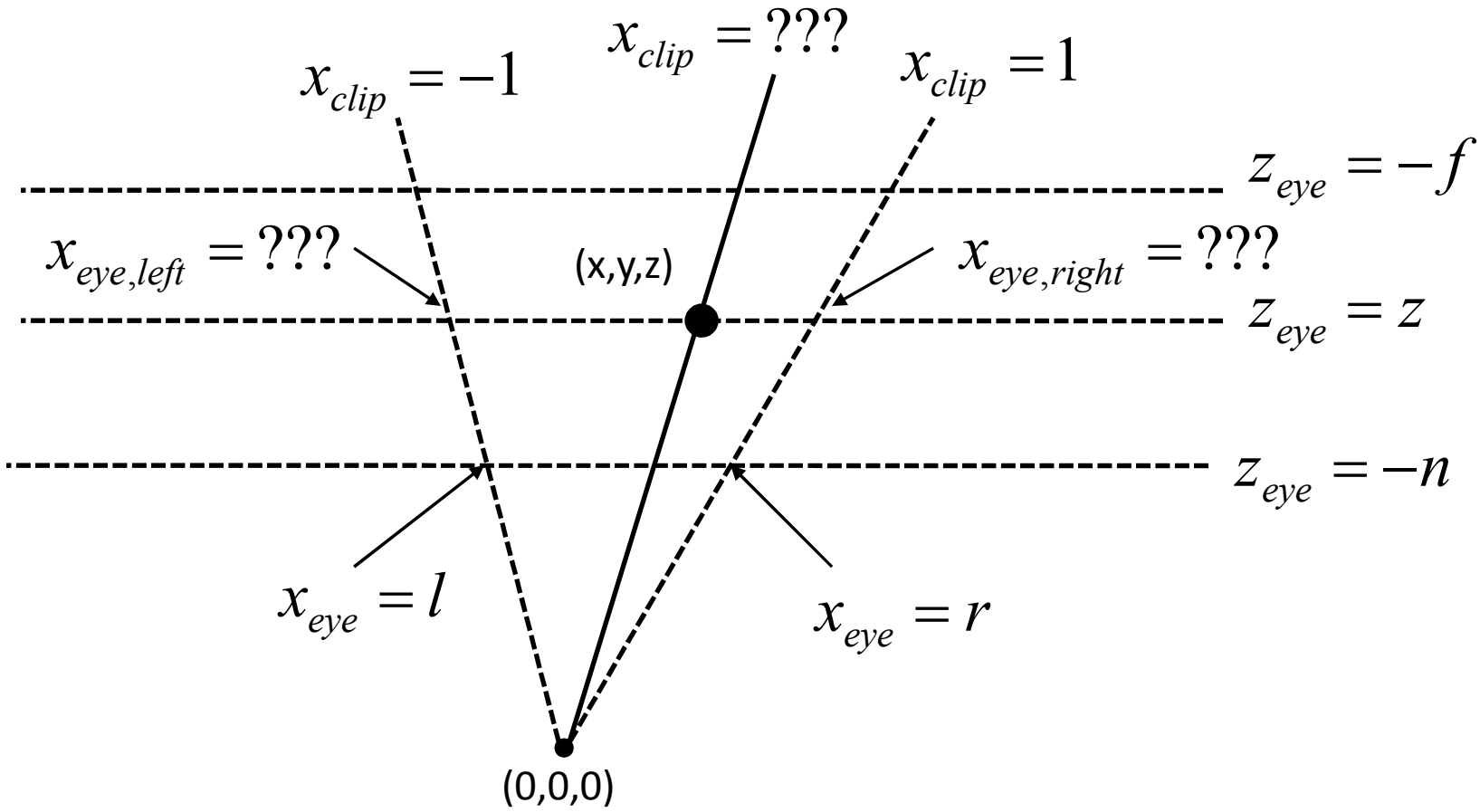
การนิยาม Perspective Projection (ต่อ)

- หา x ใน clip space



การนิยาม Perspective Projection (ต่อ)

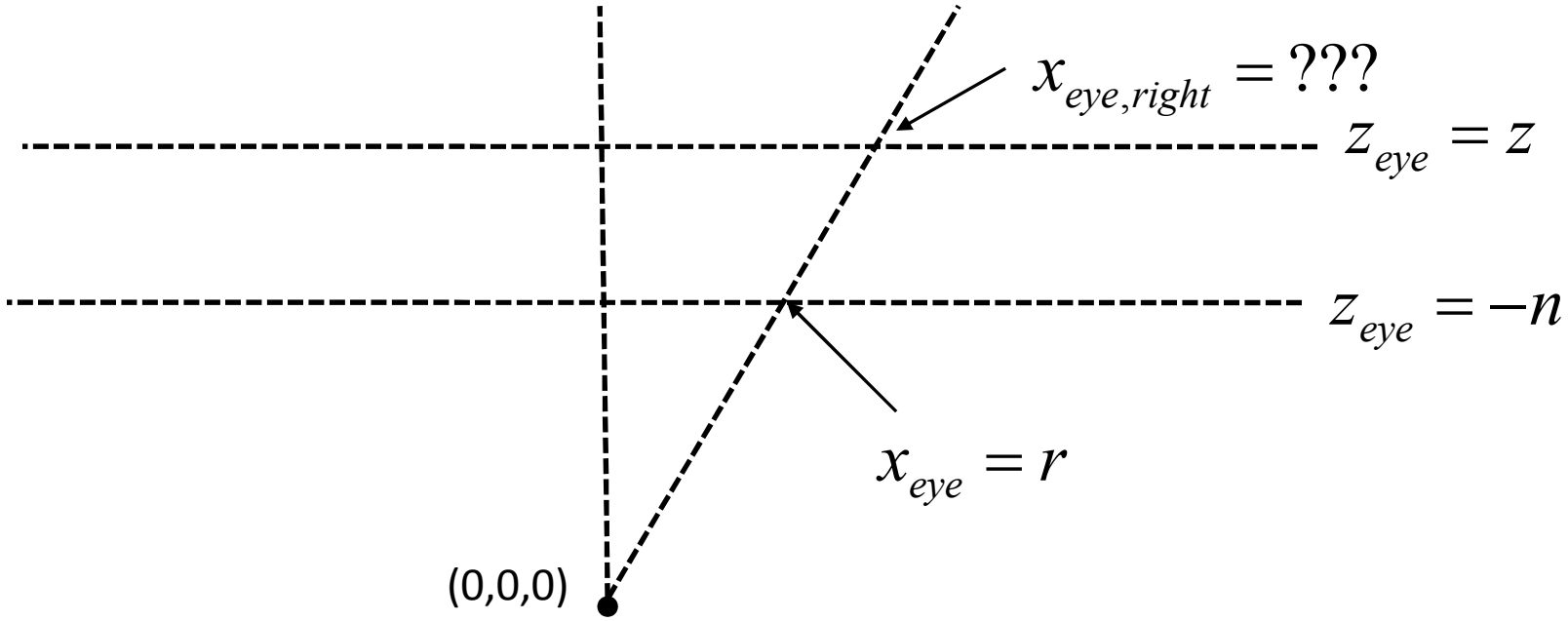
- เริ่มจากการหา x ใน eye space ของจุดปลายสองจุด



การนิยาม Perspective Projection (ต่อ)

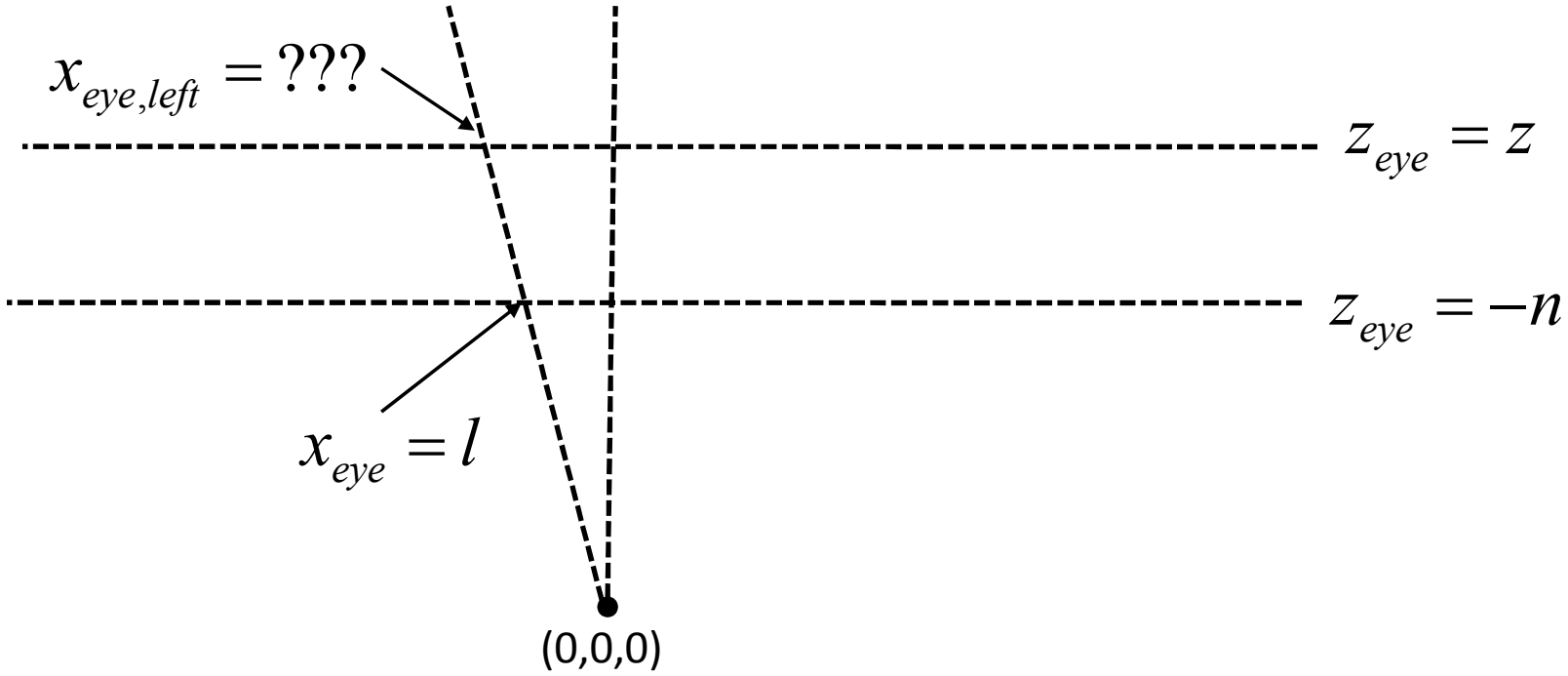
- อาศัยความรู้เรื่องสามเหลี่ยมคล้าย ได้ว่า $\frac{x_{eye,right}}{r} = \frac{z}{-n}$

  ดังนั้น $x_{eye,right} = -\frac{zr}{n}$



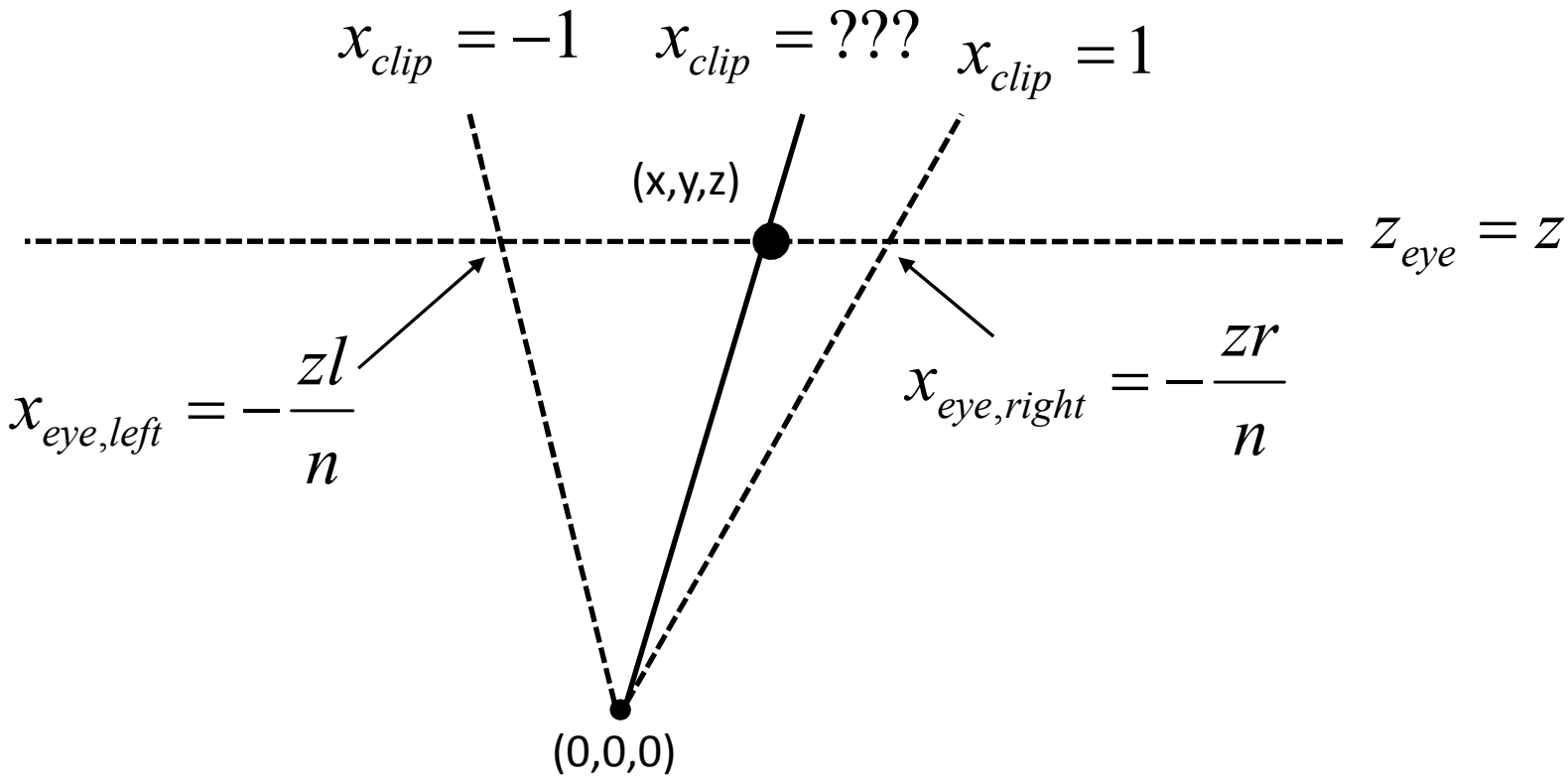
การนิยาม Perspective Projection (ต่อ)

- ทำนองเดียวกัน $\frac{x_{eye,left}}{l} = \frac{z}{-n}$ ดังนั้น $x_{eye,left} = -\frac{zl}{n}$



การนิยาม Perspective Projection (ต่อ)

- เรารู้ว่า $x_{clip} = ax + b$ สำหรับค่าคงที่ a และ b บางตัว



การนิยาม Perspective Projection (ต่อ)

- เนื่องจากถ้า $x = -zl/n$ แล้ว $x_{clip} = -1$

  และถ้า $x = -zr/n$ แล้ว $x_{clip} = 1$
- ได้ว่า

$$-1 = -a\frac{zl}{n} + b$$

$$1 = -a\frac{zr}{n} + b$$

- เมื่อแก้สมการออกมาจะได้ว่า

$$a = -\frac{2n}{(r-l)z}$$

$$b = -\frac{r+l}{r-l}$$

การนิยาม Perspective Projection (ต่อ)

- กล่าวคือ

$$x_{clip} = -\frac{2n}{(r-l)z}x - \frac{r+l}{r-l}$$

- ในทำนองเดียวกันเราก็จะได้ว่า

$$y_{clip} = -\frac{2n}{(t-b)z}y - \frac{t+b}{t-b}$$

การนิยาม Perspective Projection (ต่อ)

- แล้ว z_{clip} ควรจะมีค่าเท่าไหร่?
- ค่า z_{clip} จะถูกใช้เป็น “ความลึก” ของ fragment
- z_{clip} จะต้องมีคุณสมบัติสองประการ
  - ถ้า z น้อย z_{clip} ก็ต้องน้อยตามไปด้วย
  - perspective matrix จะต้องส่งเส้นตรงไปยังเส้นตรง
- ตัวอย่าง z_{clip} ที่ใช้ไม่ได้
  - $z_{clip}=z$
  - $z_{clip} = \sqrt{x^2 + y^2 + z^2}$

การนิยาม Perspective Projection (ต่อ)

- z_{clip} ที่ OpenGL ใช้มีรูป

$$z_{clip} = A + \frac{B}{z}$$

- เนื่องจาก $z_{clip} = -1$ ถ้า $z = -n$

  และ $z_{clip} = 1$ ถ้า $z = -f$

  จะได้ว่า

$$-1 = A - \frac{B}{n}$$

$$1 = A - \frac{B}{f}$$

การนิยาม Perspective Projection (ต่อ)

- เมื่อแก้สมการออกมาแล้วจะได้ว่า

$$B = \frac{2fn}{f-n}$$

$$A = \frac{f+n}{f-n}$$

- กล่าวคือ

$$z_{clip} = \frac{f+n}{f-n} + \frac{2fn}{(f-n)z}$$

การนิยาม Perspective Projection (ต่อ)

- กล่าวคือ perspective projection matrix จะต้องส่ง

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

Matrix ของ Perspective Projection

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

คำสั่ง OpenGL เกี่ยวกับ Perspective Projection

- glFrustum(left, right, bottom, top, near, far)
  - คูณ matrix ปัจจุบันด้วย matrix ของ perspective projection ในหน้าก่อน
  - ก่อนใช้ควรเรียก

    glMatrixMode(GL_PROJECTION)

    glLoadIdentity()

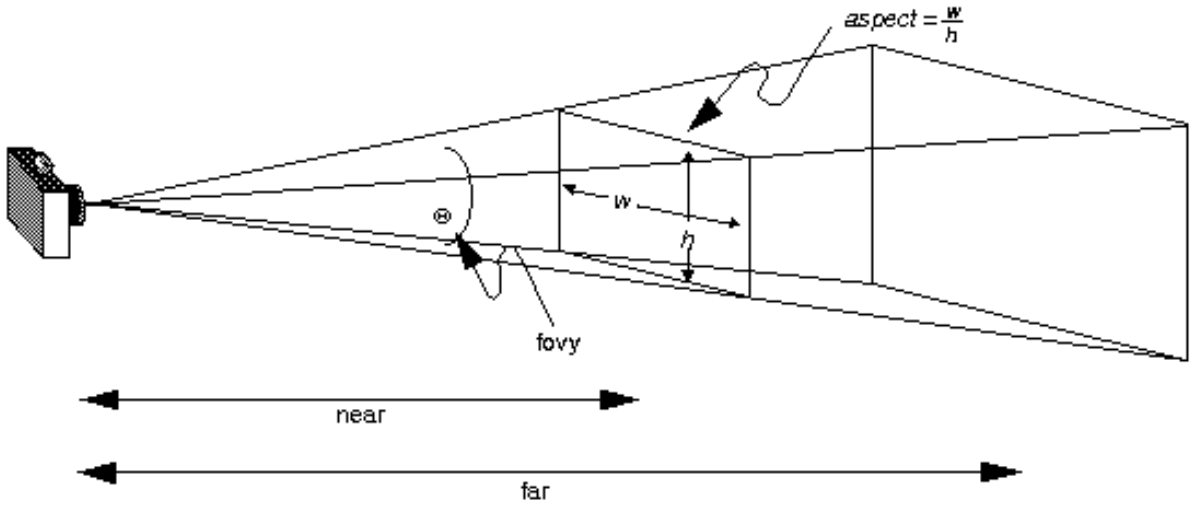
    ก่อนเพื่อเปลี่ยน mode และเคลียร์ค่า projection matrix เดิม

คำสั่ง OpenGL เกี่ยวกับ Perspective Projection (ต่อ)

- gluPerspective(fovy, aspect, near, far)
  - คูณ matrix ปัจจุบันด้วย perspective projection matrix เช่นเดียวกับ glFrustrum
  - มีผลเหมือนกับสั่ง glFrustum โดยได้
    - top = near * tan(fovy / 2)
    - bottom = -top
    - right = aspect * top
    - left = -right

คำสั่ง OpenGL เกี่ยวกับ Perspective Projection (ต่อ)

- fovy ย่อมาจาก field of view Y หมายถึงความกว้างของมุมมองตามแนวแกน y (มีหน่วยเป็นองศา)
- aspect คือ aspect ratio ของหน้าตัดของปีระมิด
- ปีระมิดที่ gluPerspective สร้างมีหน้าตาเป็นดังข้างล่าง



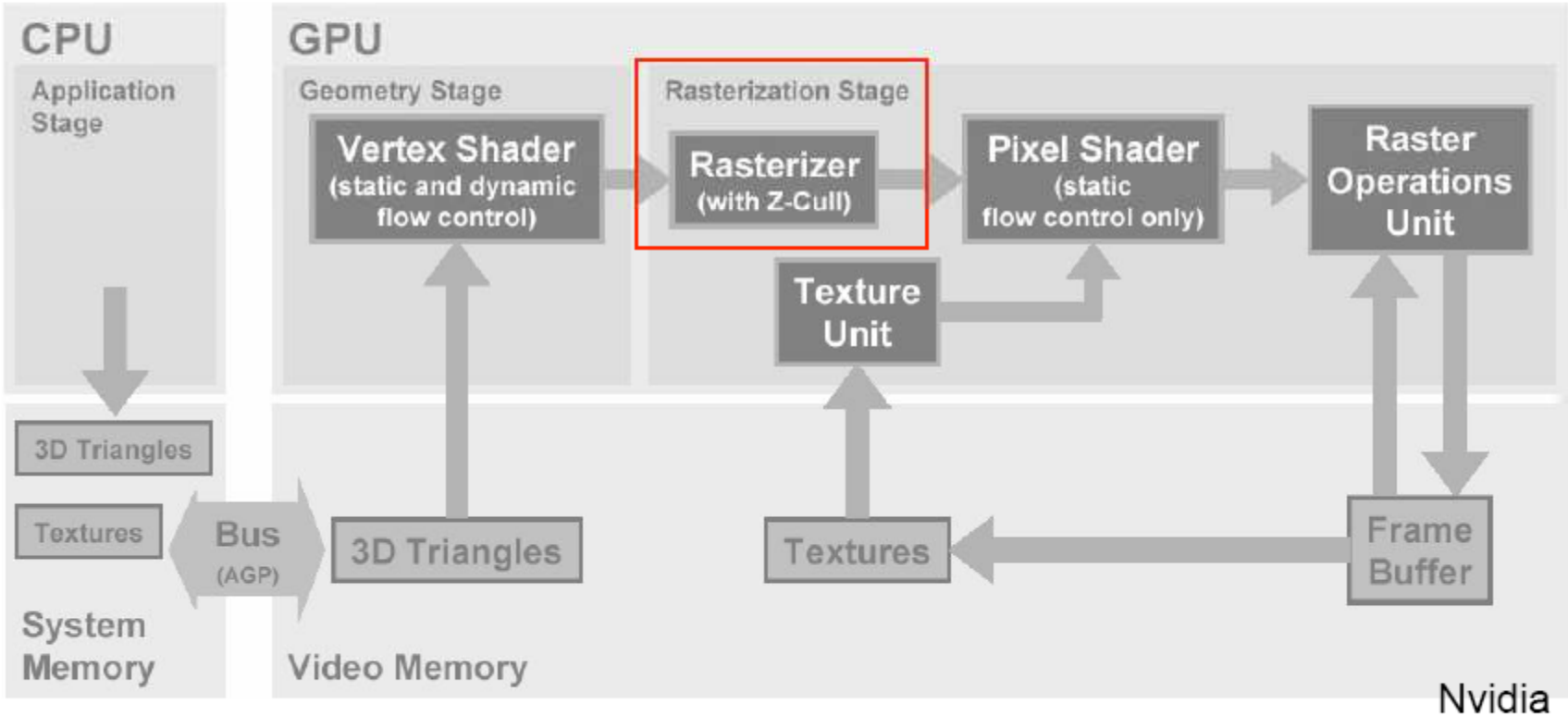
คำสั่ง OpenGL เกี่ยวกับ Perspective Projection (ต่อ)

- สังเกตว่าคำสั่ง glFrustum สามารถสร้างปีระมิดที่ไม่สมมาตรรอบแกน z ได้
- แต่ปีระมิดที่สร้างด้วย gluPerspective จะเป็นปีระมิดที่สมมาตรรอบแกน z เสมอ

RASTERIZATION

3D Graphics Pipeline

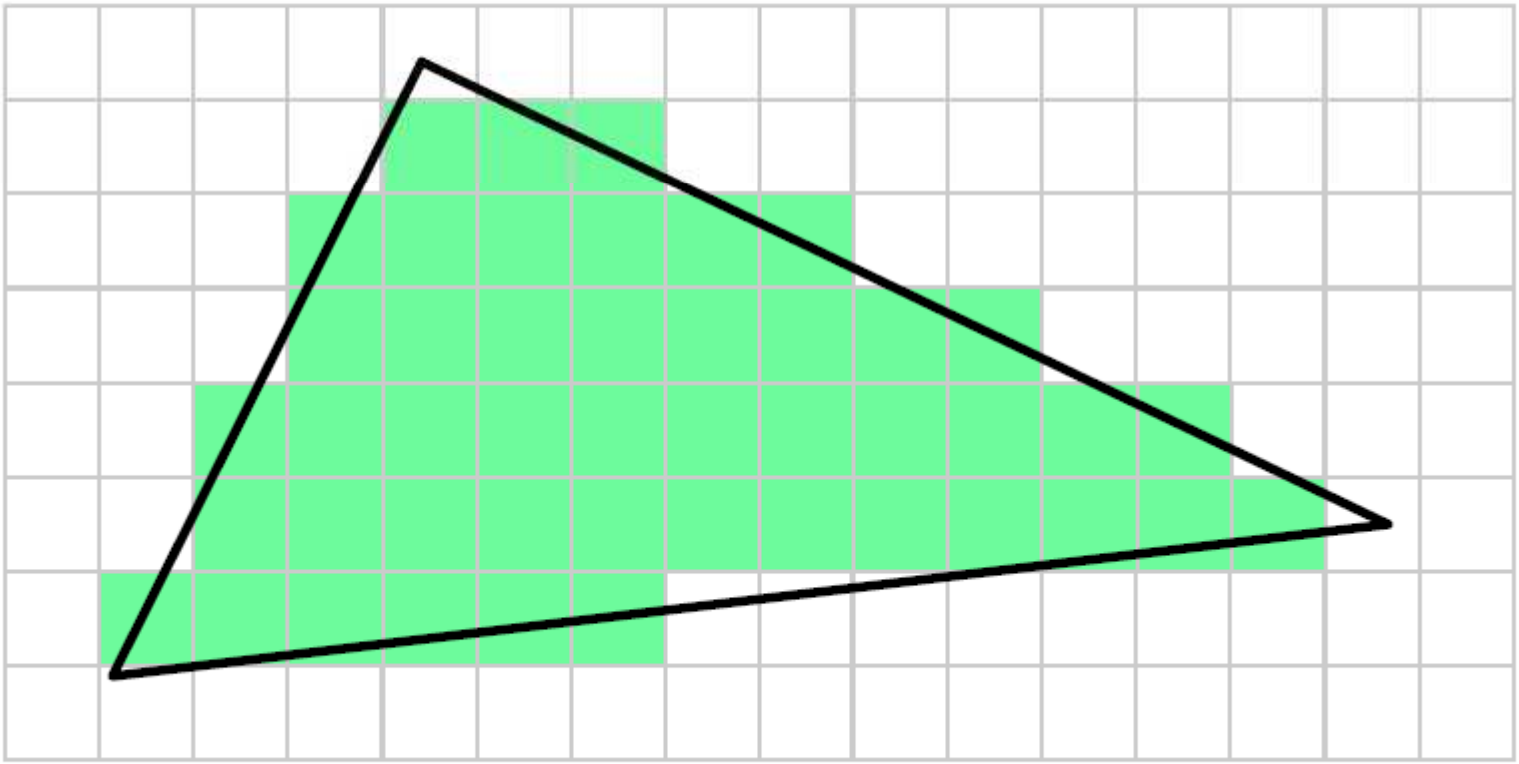
- The rasterization step scan converts the object into pixels



Nvidia

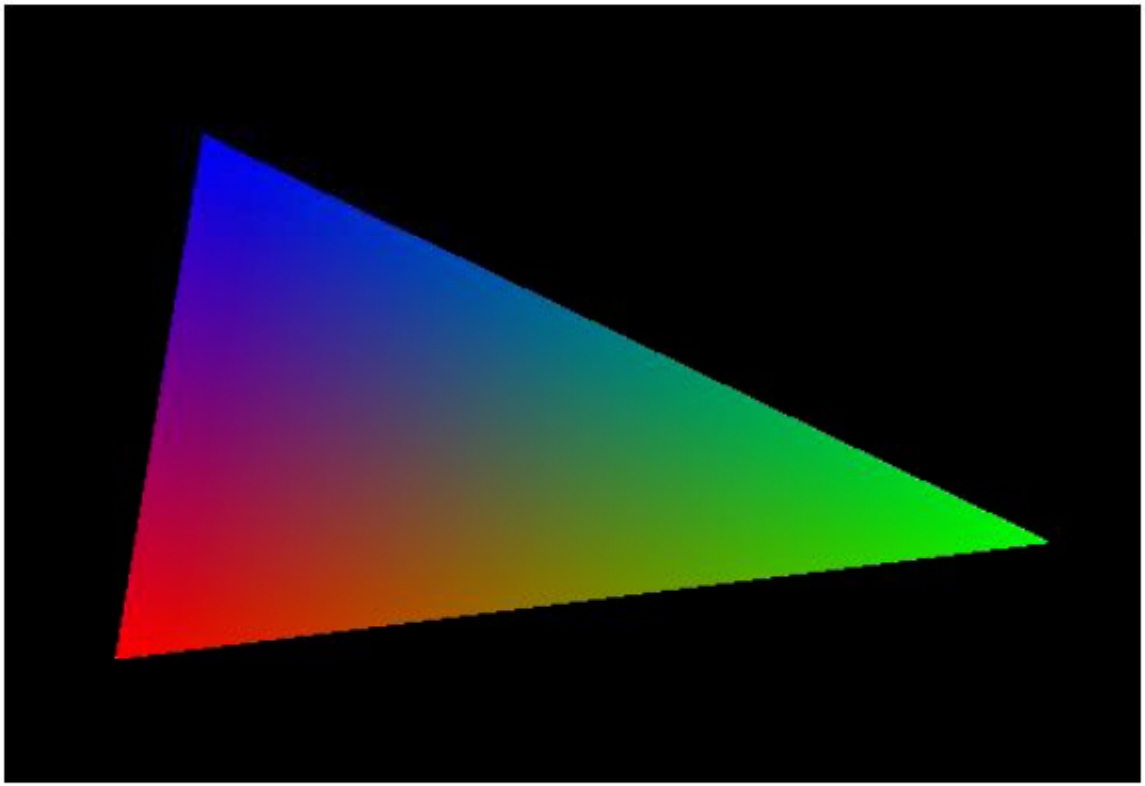
Rasterization (scan conversion)

- Determine which fragments get generated
- Interpolate parameters (colors, texture coordinates, etc.)



Parameter interpolation

- What does “interpolation” mean?
- Example: colors

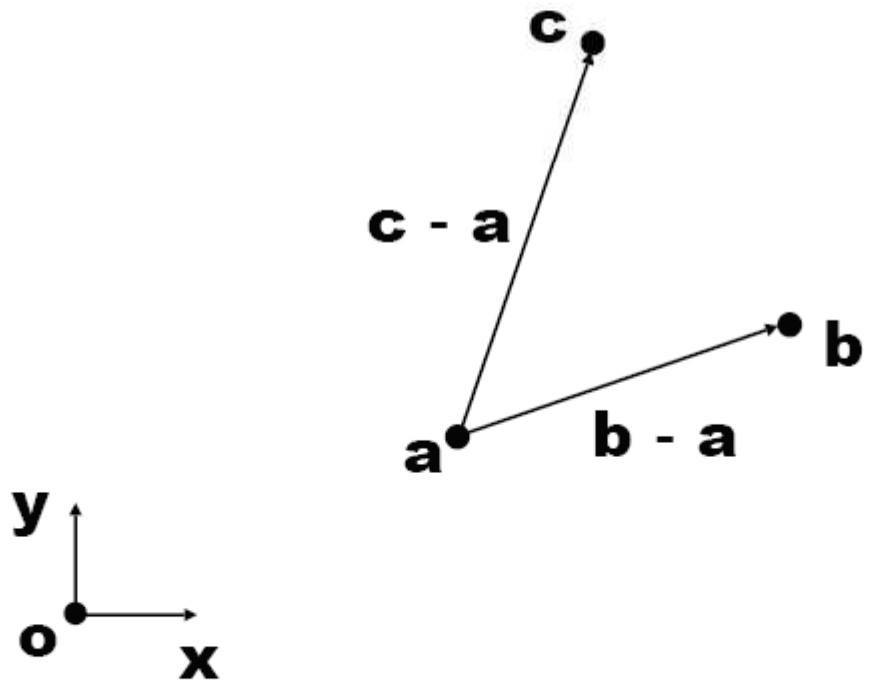


Game Plan

- Theory (using triangles):
  - vector representation of a triangle
  - introduction to barycentric coordinates
  - implicit lines
- Application (still using triangles):
  - rasterization and interpolation using implicit lines and barycentric coordinates

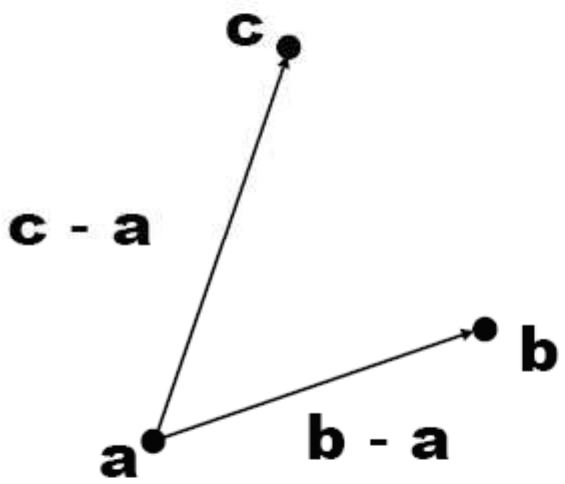
A triangle in terms of vectors

- We can use vertices **a**, **b**, and **c** to specify the three points of a triangle.
- We can also compute the edge vectors.



Points and planes

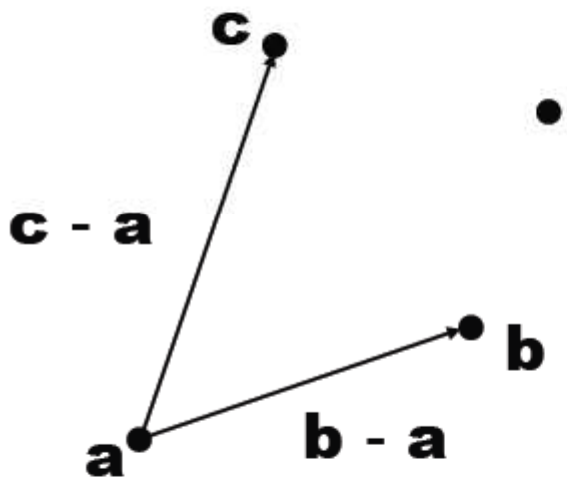
- Three non-collinear points determine a plane



- Example: the vertices **a**, **b**, and **c** determine a plane
- The vectors **b-a** and **c-a** form a basis for this plane

Basis vectors

- This (non-orthogonal) basis can be used to specify the location of any point **p** in the plane



- $\mathbf{p} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$

Barycentric Coordinates

- We can reorder the terms of the equation:

$$\begin{aligned}\mathbf{p} &= \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \\ &= (1 - \beta - \gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\ &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}\end{aligned}$$

- This yields the equation:

$$\mathbf{p}(\alpha, \beta, \gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

- with:

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

- This coordinate system is called <u>barycentric coordinates</u>
  - α, β, γ are the "coordinates"

Barycentric Coordinates

- Barycentric coordinates describe a point **p** as an affine combination of the triangle vertices

$$\mathbf{p}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \qquad \alpha + \beta + \gamma = 1$$

- For any point **P** inside the triangle (**a**, **b**, **c**):

$$0 < \alpha < 1,$$
$$0 < \beta < 1,$$
$$0 < \gamma < 1.$$

- Point on an edge: one coefficient is 0
- Vertex: two coefficients are 0, remaining one is 1

Recap so far

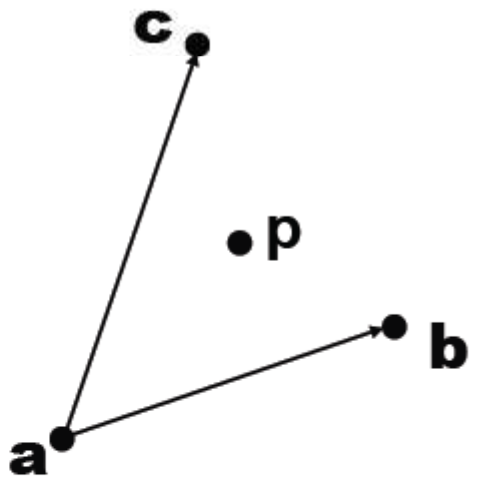
- We need to interpolate parameters during rasterization
- If a triangle is defined by (**a**, **b**, **c**) then any point p inside can be written as

$$\mathbf{p}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

with these properties:

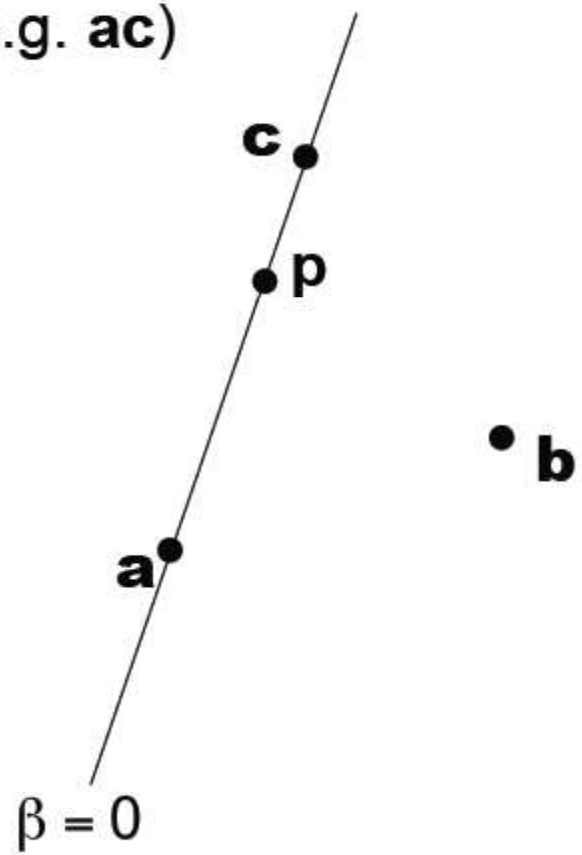
$$0 < \alpha < 1,$$
$$0 < \beta < 1,$$
$$0 < \gamma < 1.$$

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



Signed distances

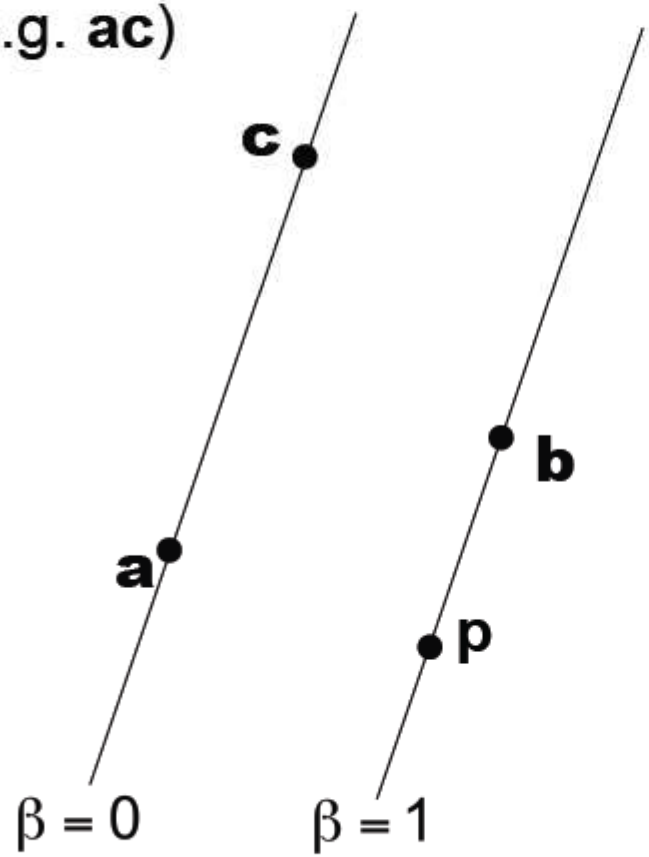
- Let $\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$. Each coordinate (e.g. β) is the signed distance from **p** to the line through a triangle edge (e.g. **ac**)



$$\mathbf{p}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

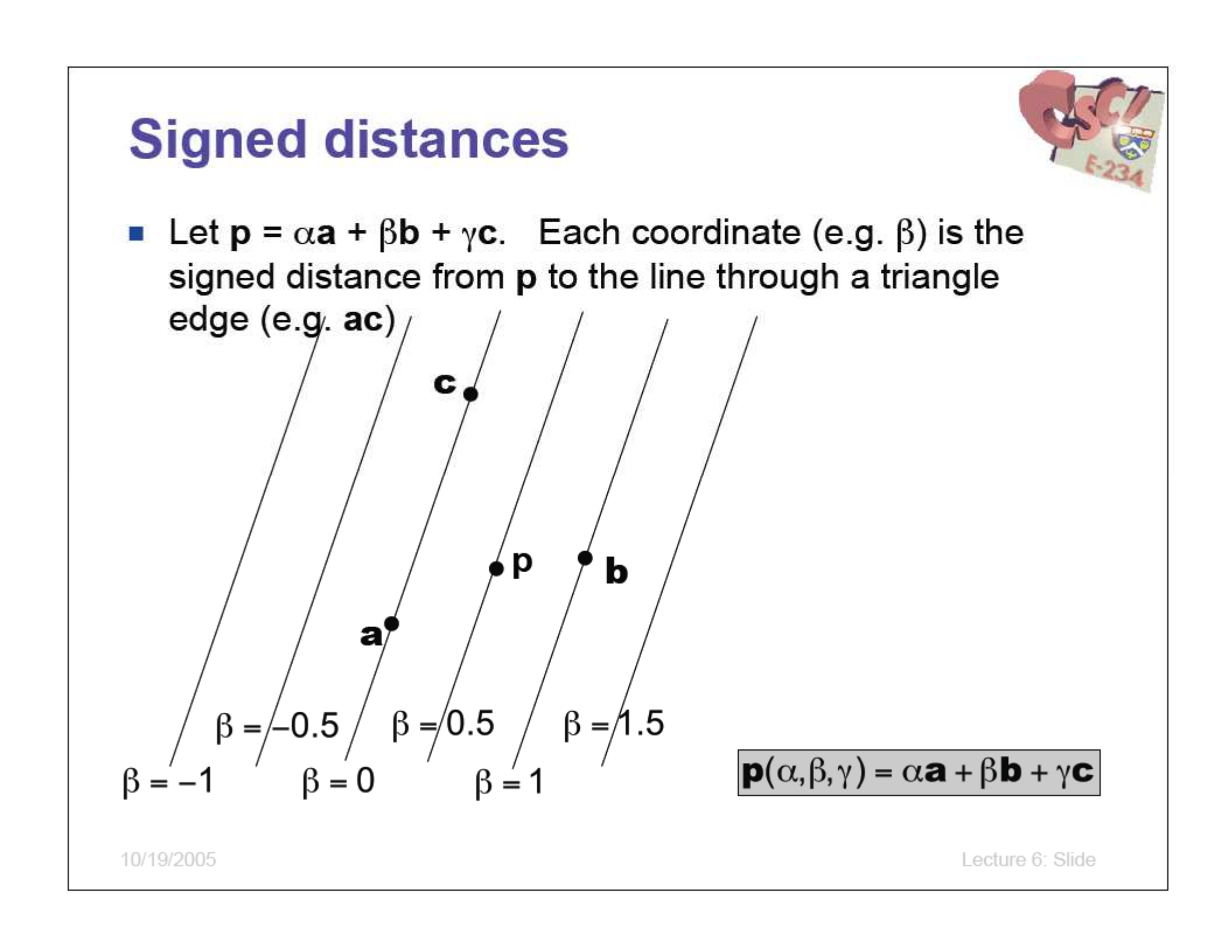
Signed distances

- Let $\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$. Each coordinate (e.g. β) is the signed distance from **p** to the line through a triangle edge (e.g. **ac**)



$$\mathbf{p}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

Signed distances

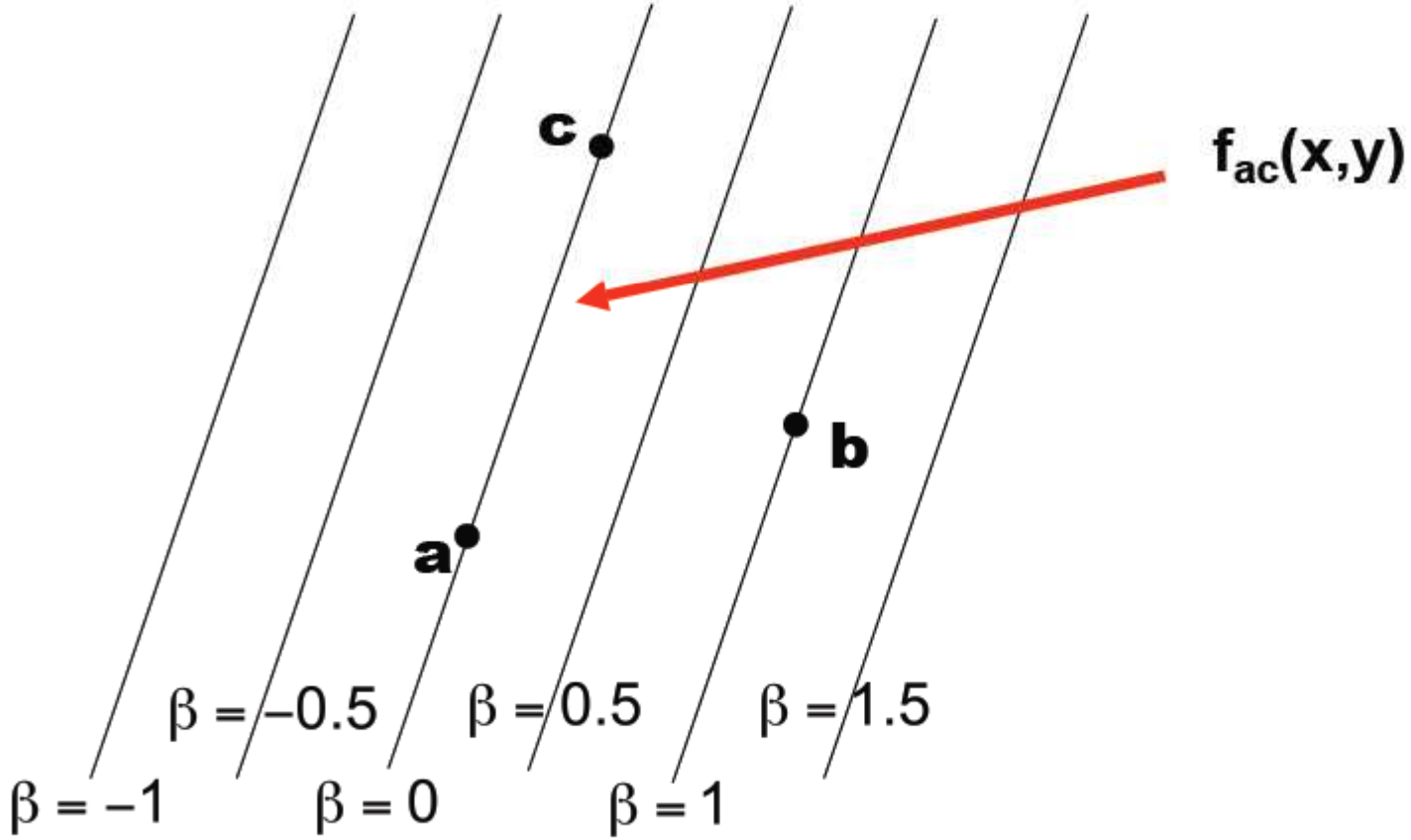


- Let $\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$. Each coordinate (e.g. β) is the signed distance from **p** to the line through a triangle edge (e.g. **ac**)

$$\mathbf{p}(\alpha,\beta,\gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

Signed distances

- The signed distance can be computed by evaluating implicit line equations, e.g., $f_{ac}(x,y)$ of edge **ac**



Implicit Lines

- Implicit equation in two dimensions:

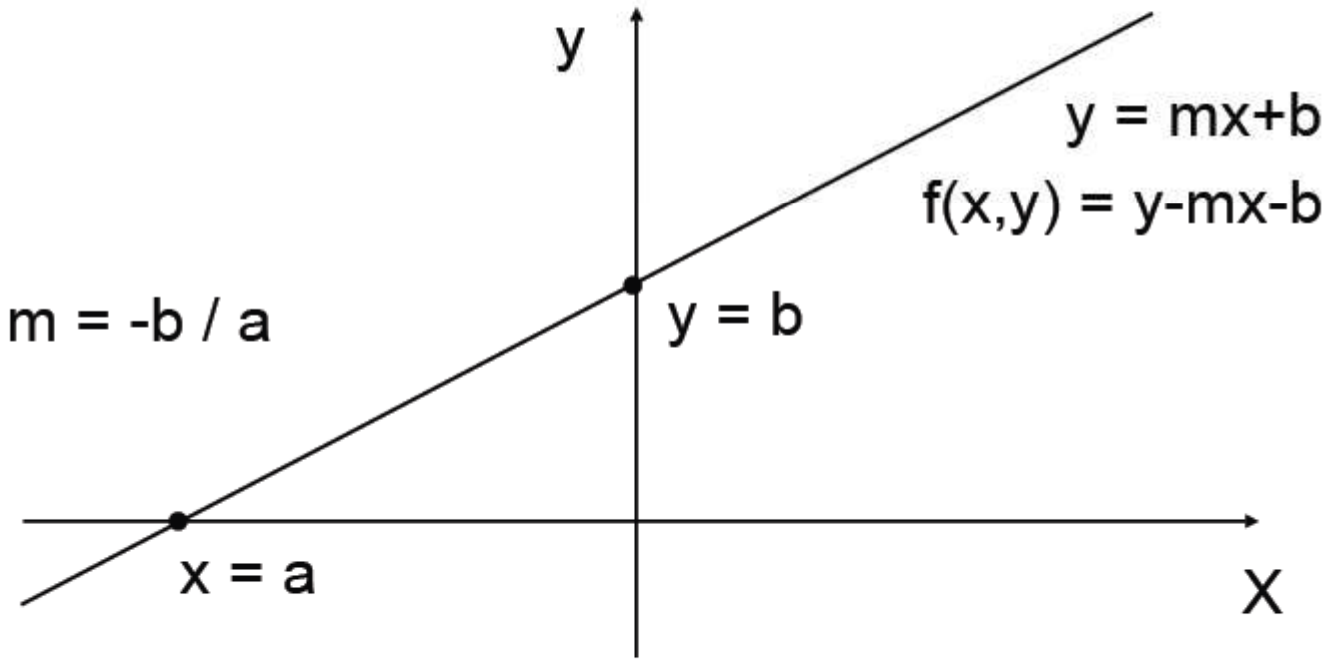
$$f(x, y) = 0$$

  - Points with f(x,y) = 0 are on the line
  - Points with f(x,y) ≠ 0 are not on the line

Implicit Lines

- The implicit form of the slope-intercept equation:

$$y - mx - b = 0$$



Implicit Lines

- The slope-intercept form can not represent some lines, such as $x = 0$.
- A more general implicit form is more useful:

$$Ax + By + C = 0$$

- The implicit line through two points (x_0,y_0) and (x_1,y_1):

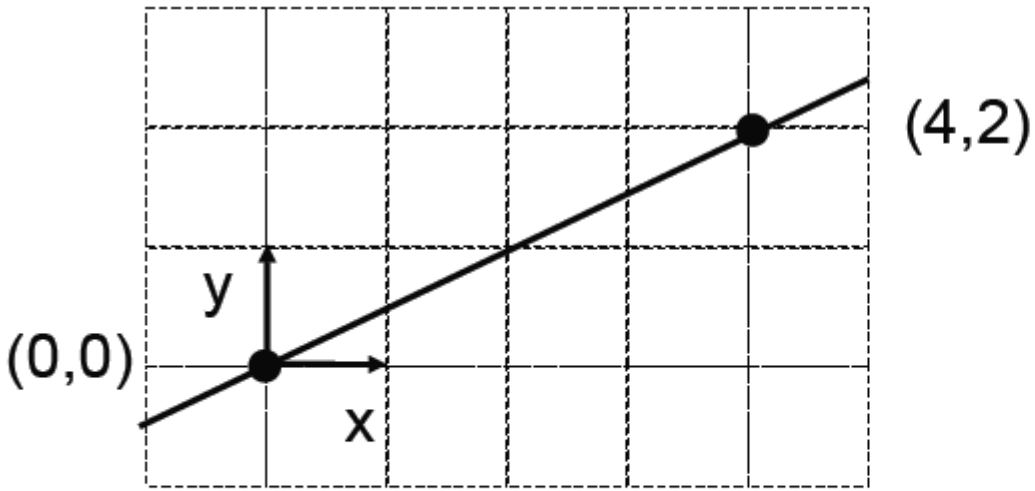
$$(y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0$$

Example

- What is the implicit equation of this line?

$$(y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 = 0$$

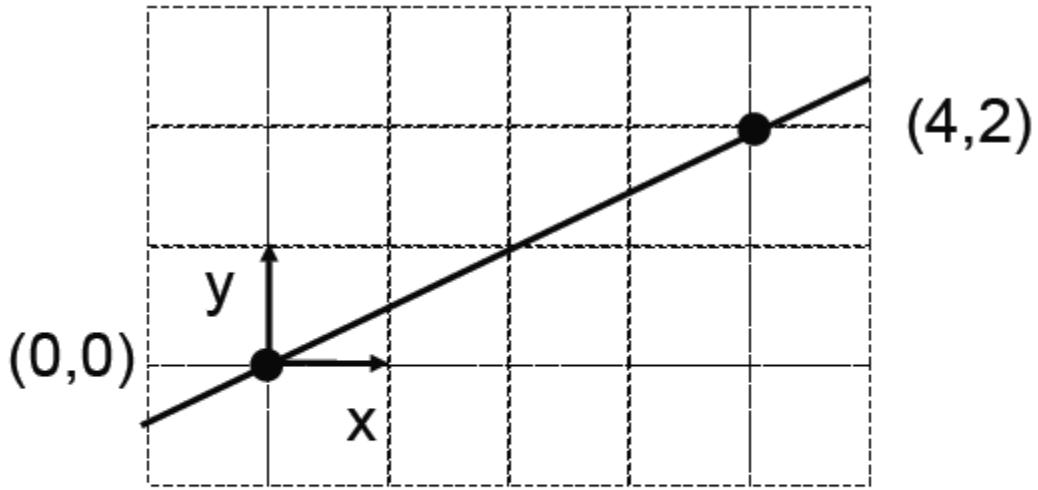
$$\_x + \_y + \_ = 0$$



Example

- Solution 1: $-2x + 4y = 0$
- Solution 2: $2x - 4y = 0$
- What's the lesson here?

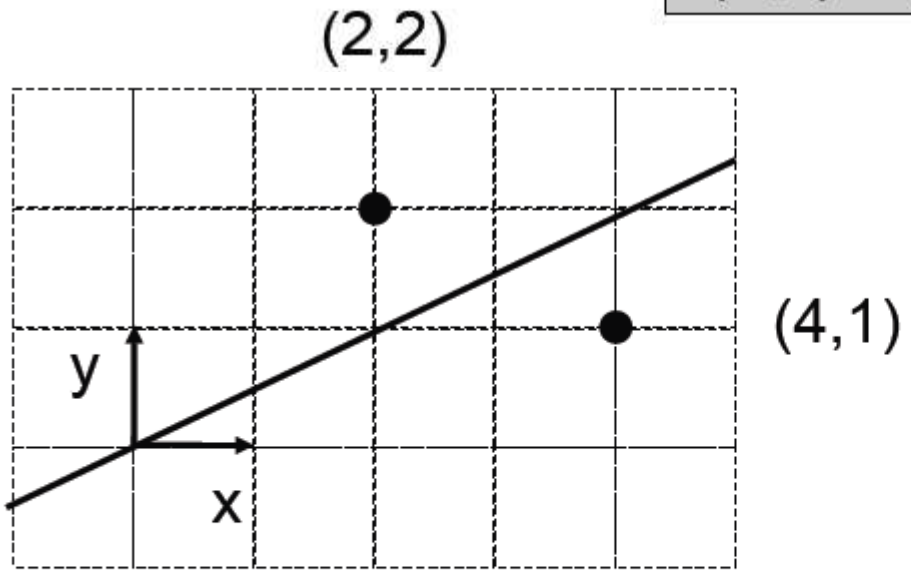
k f(x,y) = 0 is the same line, for any value of k



Example

- The value of f(x,y) = -2x +4y tells us which side of the line a point (x,y) is on

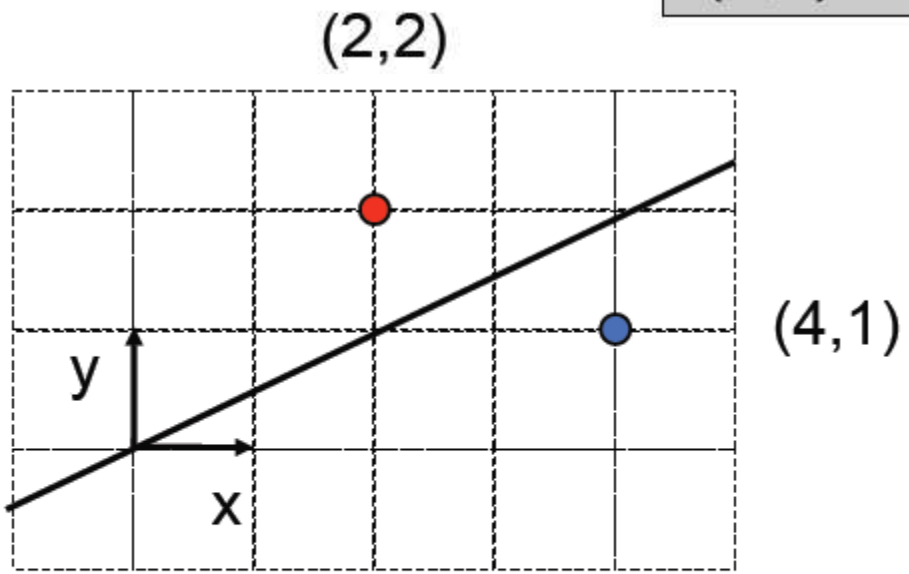
f(2,2) = _
f(4,1) = _



Example

- The value of f(x,y) = -2x +4y tells us which side of the line a point (x,y) is on

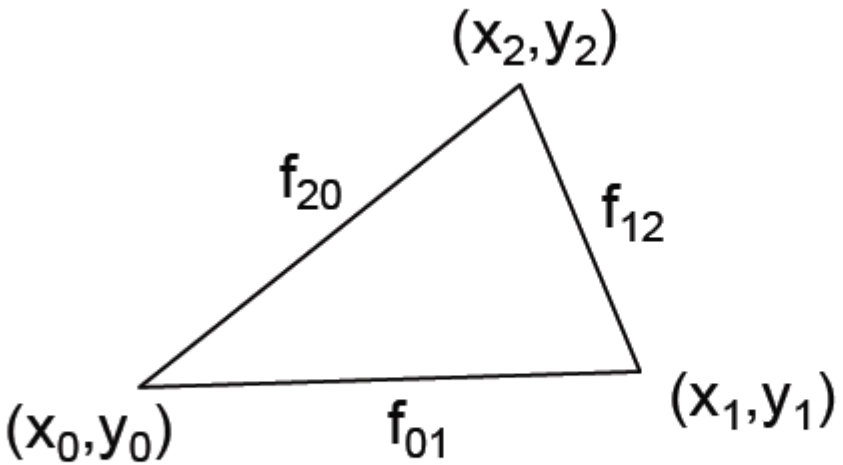
$$f(2,2) = +4 \quad (+ = \text{above})$$
$$f(4,1) = -4 \quad (- = \text{below})$$



Edge Equations

- Given a triangle with vertices (x_0,y_0), (x_1,y_1), and (x_2,y_2).
- The line equations of the edges of the triangle are:

$$f_{01}(x,y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0$$
$$f_{12}(x,y) = (y_1 - y_2)x + (x_2 - x_1)y + x_1y_2 - x_2y_1$$
$$f_{20}(x,y) = (y_2 - y_0)x + (x_0 - x_2)y + x_2y_0 - x_0y_2$$



Barycentric Coordinates

- Remember that: $f(x,y) = 0 \Leftrightarrow kf(x,y) = 0$
- A barycentric coordinate (e.g. β) is a signed distance from a line (e.g. the line that goes through **ac**)
- For a given point **p**, we would like to compute its barycentric coordinate β using an implicit edge equation.
- We need to choose **k** such that $kf_{ac}(x,y) = \beta$

Barycentric Coordinates

- We would like to choose k such that: $kf_{ac}(x,y) = \beta$
- We know that $\beta = 1$ at point **b**:

$$kf_{ac}(x_b, y_b) = 1 \Leftrightarrow k = \frac{1}{f_{ac}(x_b, y_b)}$$

- The barycentric coordinate β for point **p** is:

$$\beta = \frac{f_{ac}(x,y)}{f_{ac}(x_b, y_b)}$$

cartesian coordinates of **p**

cartesian coordinates of **b**

Barycentric Coordinates

- In general, the barycentric coordinates for point **p** are:

cartesian coordinates of **p**

$$\alpha = \frac{f_{bc}(x,y)}{f_{bc}(x_c,y_c)} \quad \beta = \frac{f_{ac}(x,y)}{f_{ac}(x_b,y_b)} \quad \gamma = 1-\alpha-\beta$$

cartesian coordinates of **a**

cartesian coordinates of **b**

- Given a point **p** with cartesian coordinates (**x**,**y**), we can compute its barycentric coordinates (α,β,γ) as above.

GPU Triangle Rasterization

- Many different ways to generate fragments for a triangle
- Checking ($0<\alpha<1$ && $0<\beta<1$ && $0<\gamma<1$) is one method
- In practice, GPUs use optimized methods:
  - fixed point precision (not floating-point)
  - incremental (use results from previous pixel)

Triangle Rasterization

- We can use barycentric coordinates to rasterize and color triangles

```
for all x do
   for all y do
          compute (alpha, beta, gamma) for (x,y)
          if ( 0 < alpha < 1 and
               0 < beta < 1 and
               0 < gamma < 1 ) then
               c = alpha c0 + beta c1 + gamma c2
               drawpixel(x,y) with color c
```

- The color **c** varies smoothly within the triangle
- This is called Gouraud interpolation after its inventor Henri Gouraud (French, born 1944)

Triangle Rasterization

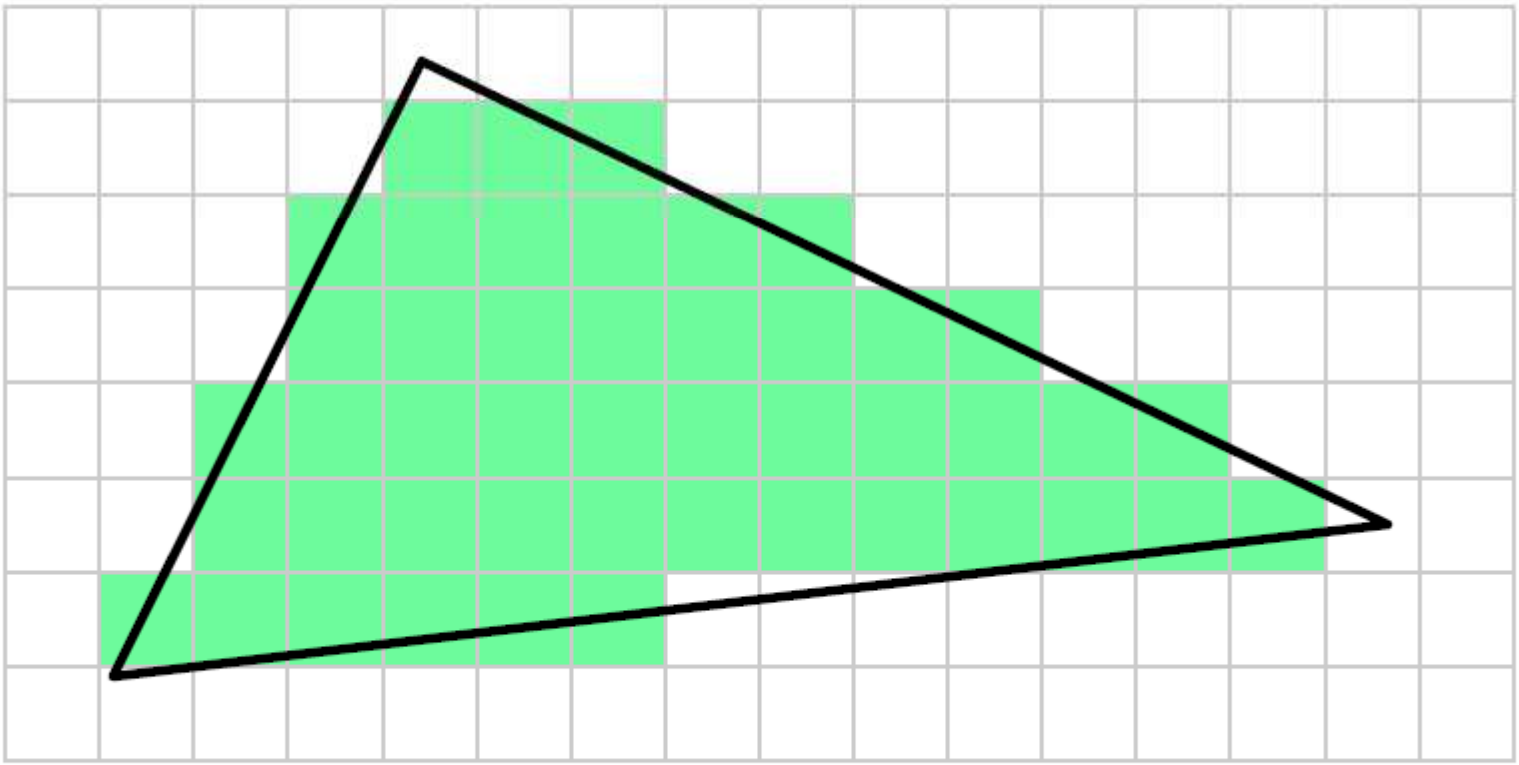
- Instead of looping over the whole image, we can loop over pixels inside the bounding rectangle of the triangle

```
xmin = floor(xi)
xmax = ceiling(xi)
ymin = floor(yi)
ymax = ceiling(yi)
for all y = ymin to ymax do
   for all x = xmin to xmax do
          alpha = f12(x,y) / f12(x0,y0)
          beta = f20(x,y) / f20(x1,y1)
          gamma = f01(x,y) / f01(x2,y2)
          if ( alpha > 0 and beta > 0 and gamma > 0 ) then
               c = alpha c0 + beta c1 + gamma c2
               drawpixel(x,y) with color c
```

HIDDEN SURFACE REMOVAL

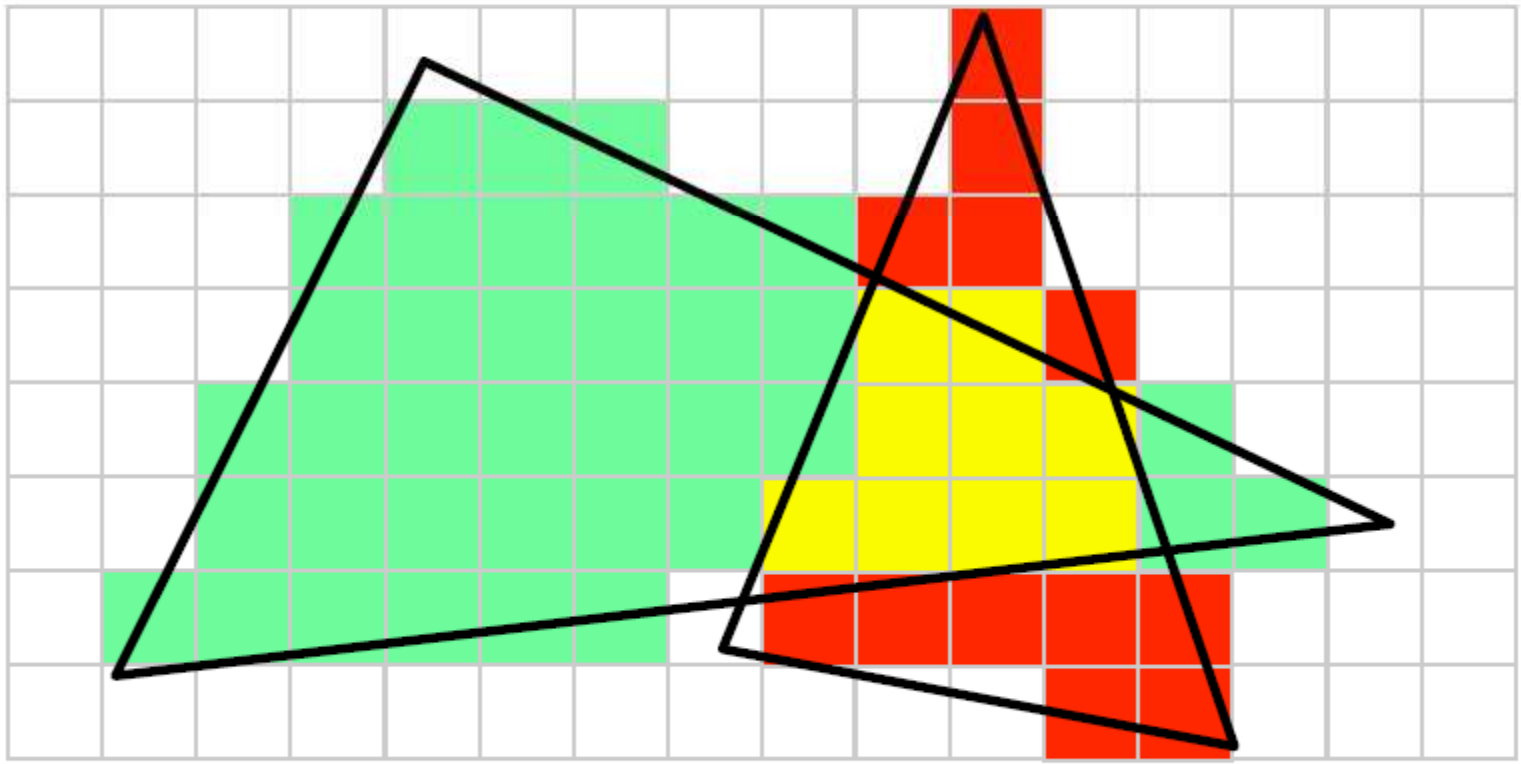
One Triangle

- With one triangle, things are simple
- Fragments never overlap!



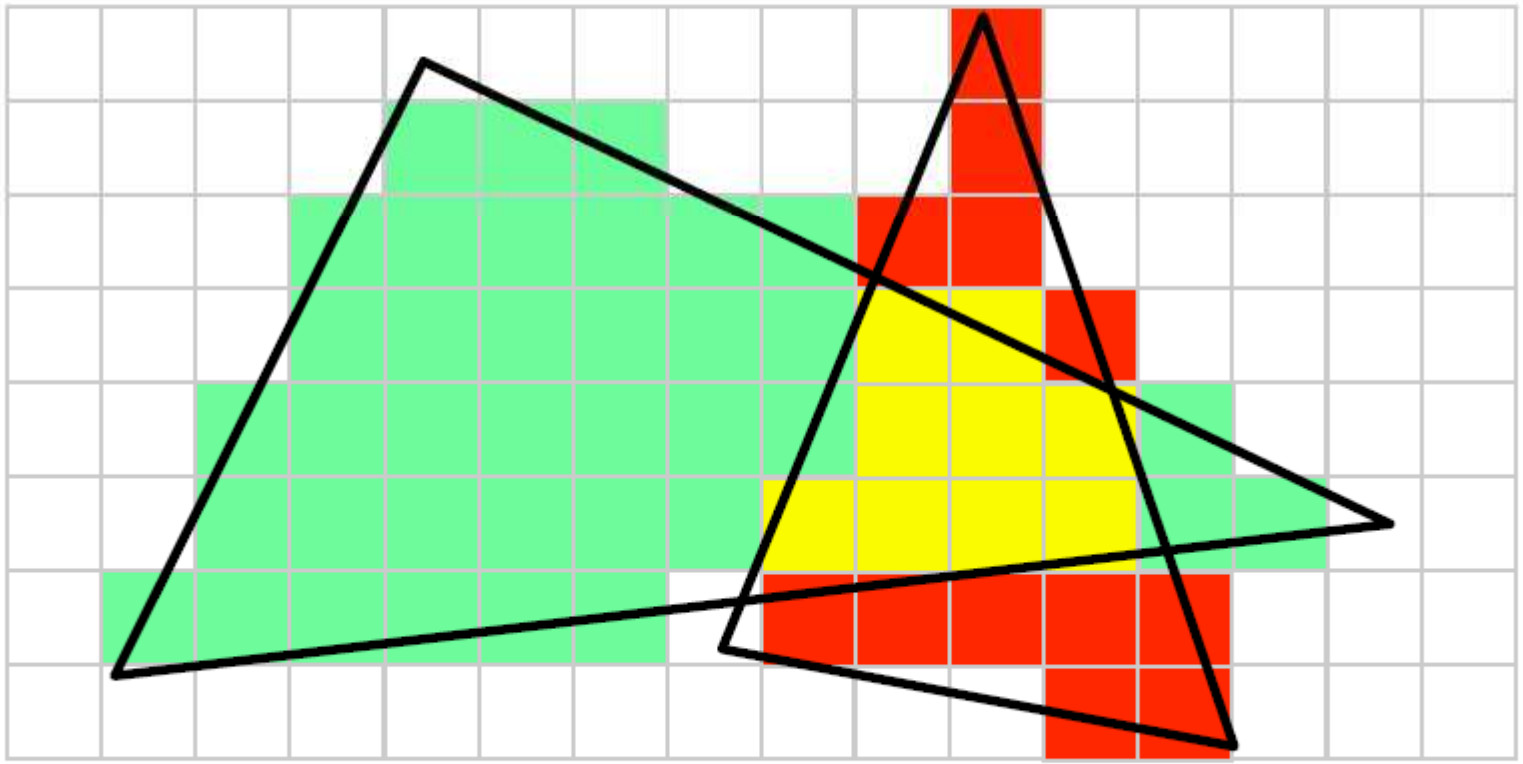
Two Triangles

- Things get more complicated with multiple triangles
- Fragments might overlap in screen space!



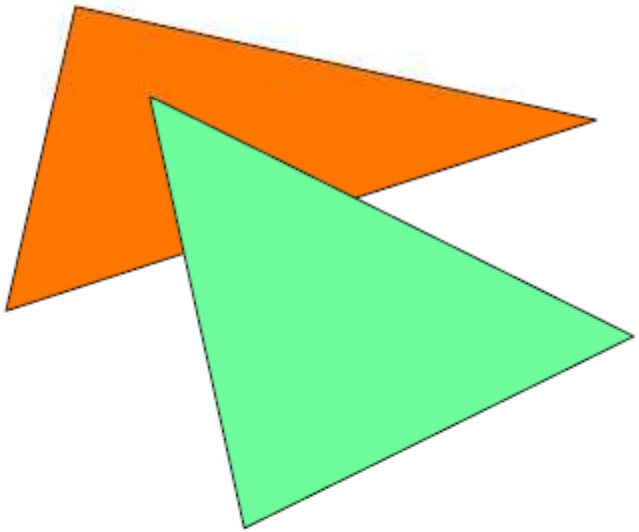
Fragments vs. Pixels

- Each pixel has a unique framebuffer (image) location
- But multiple fragments may end up at same address

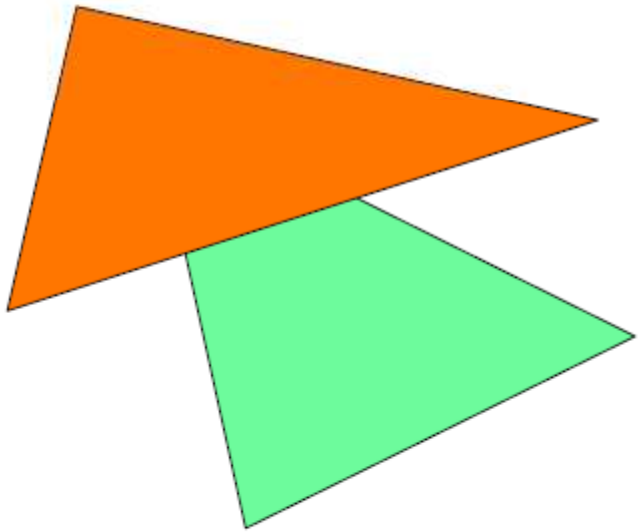


Which triangle wins?

- Two possible cases:



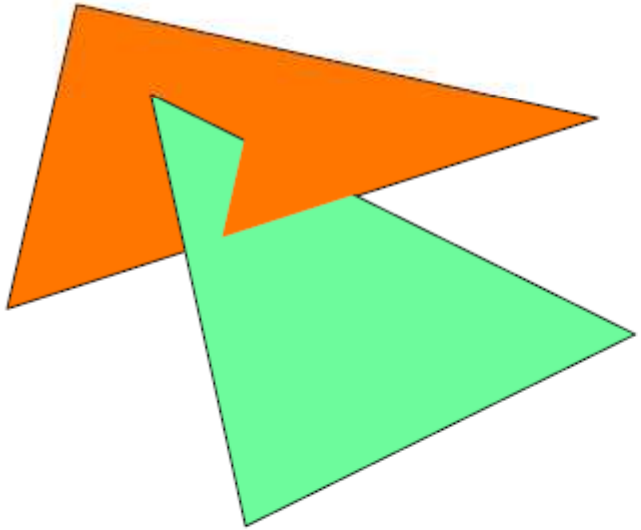
green triangle on top



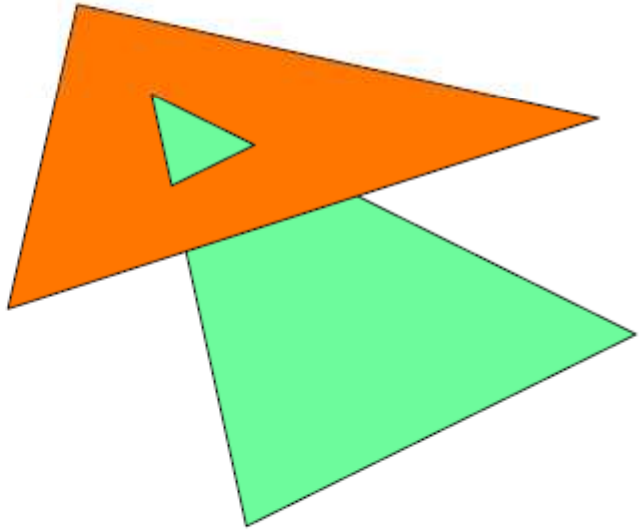
orange triangle on top

Which (partial) triangle wins?

- Many other cases possible!



intersection #1



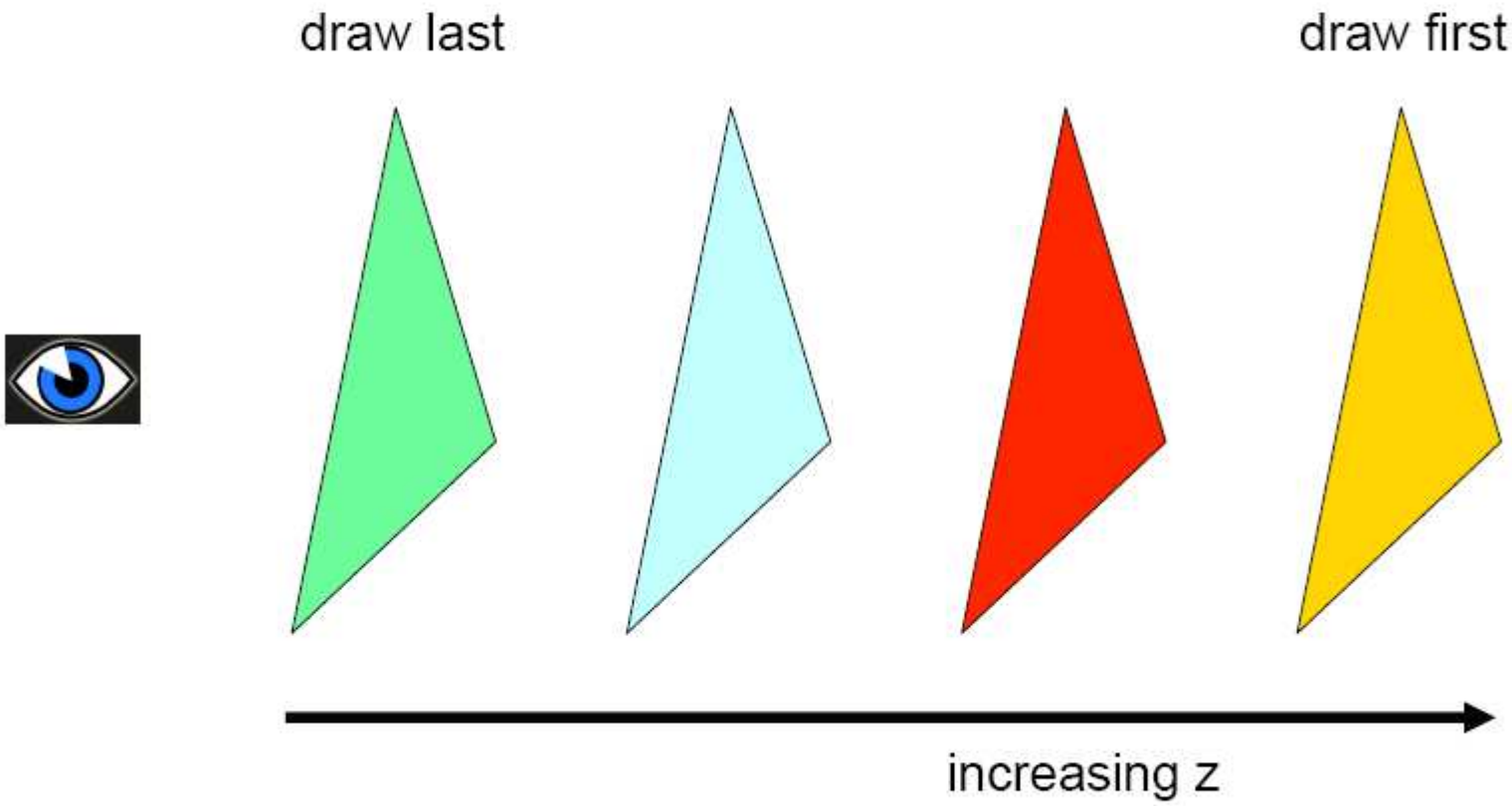
intersection #2

Hidden Surface Removal

- Idea: keep track of visible surfaces
- Typically, we see only the front-most surface
- Exception: transparency

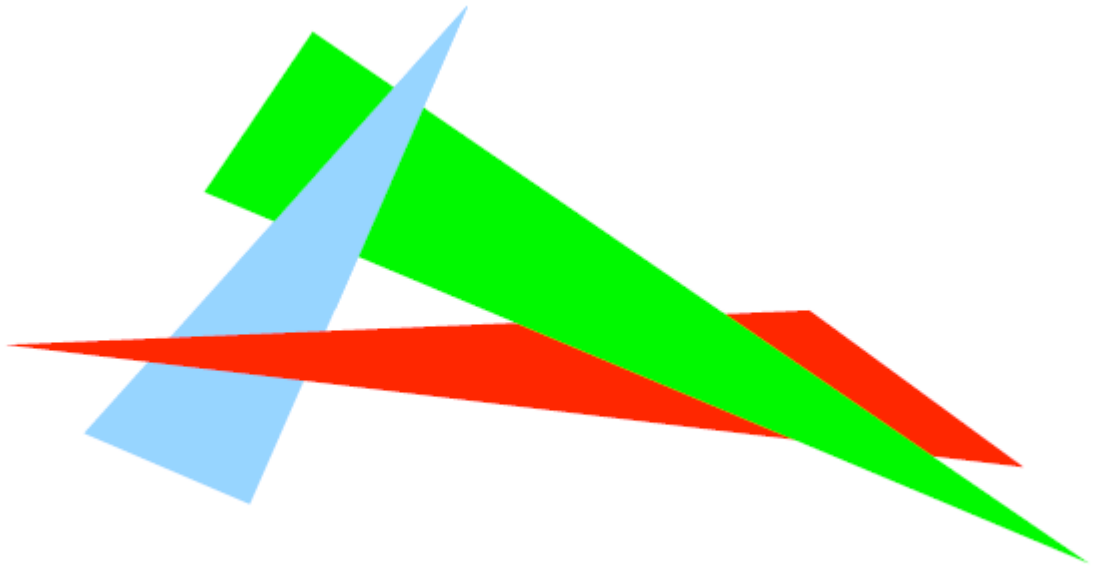
First Attempt: Painter's Algorithm

- Sort triangles (using **z** values in eye space)
- Draw triangles from back to front



Problems?

- Correctness issues:
  - Intersections
  - Cycles
  - Solve by splitting triangles, but ugly and expensive
- Efficiency (sorting)

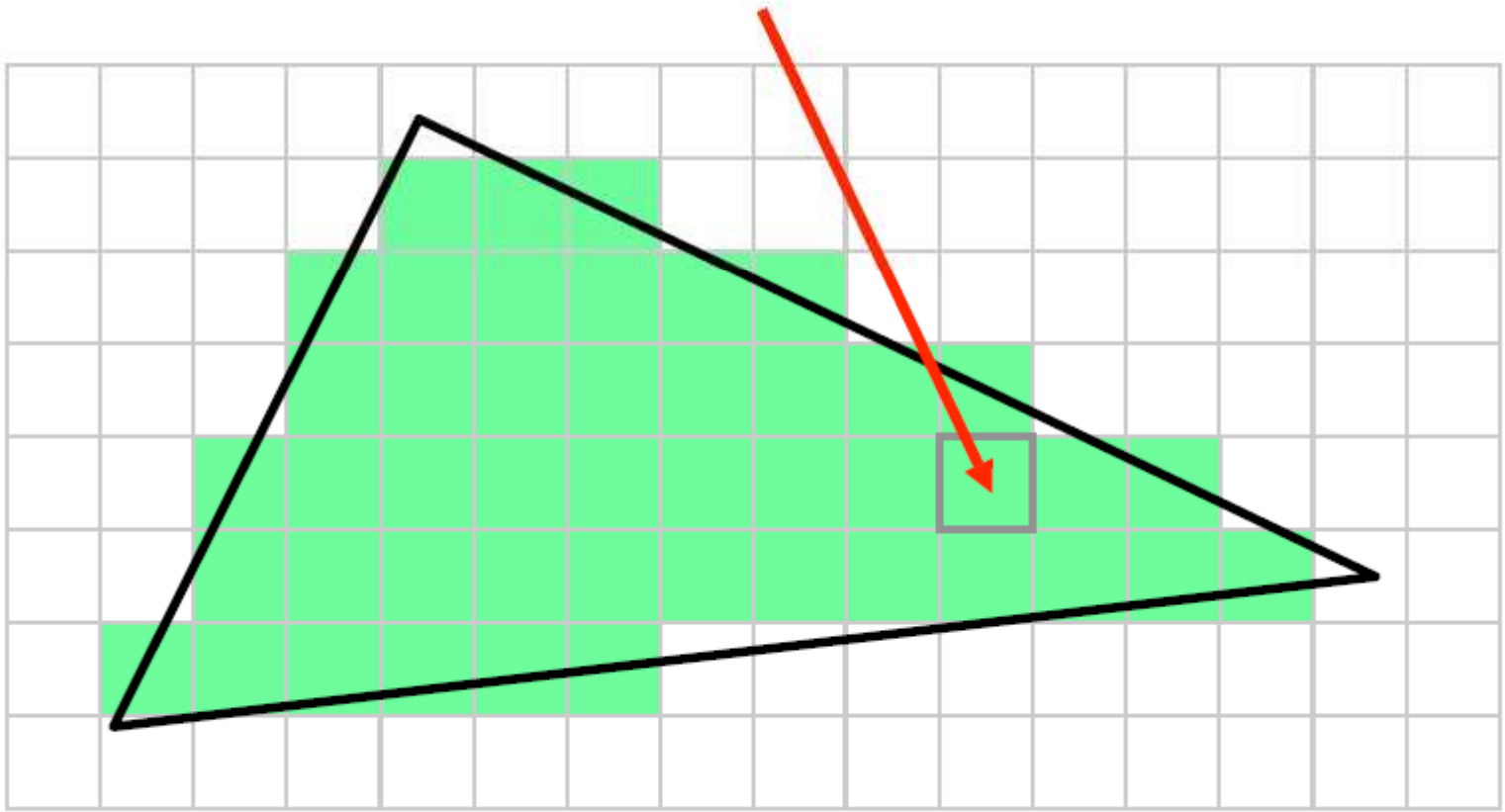


The Depth Buffer (Z-buffer)

- Perform hidden surface removal per-fragment
- Idea:
  - Each fragment gets a **z** value in screen space
  - Keep only the fragment with the smallest **z** value

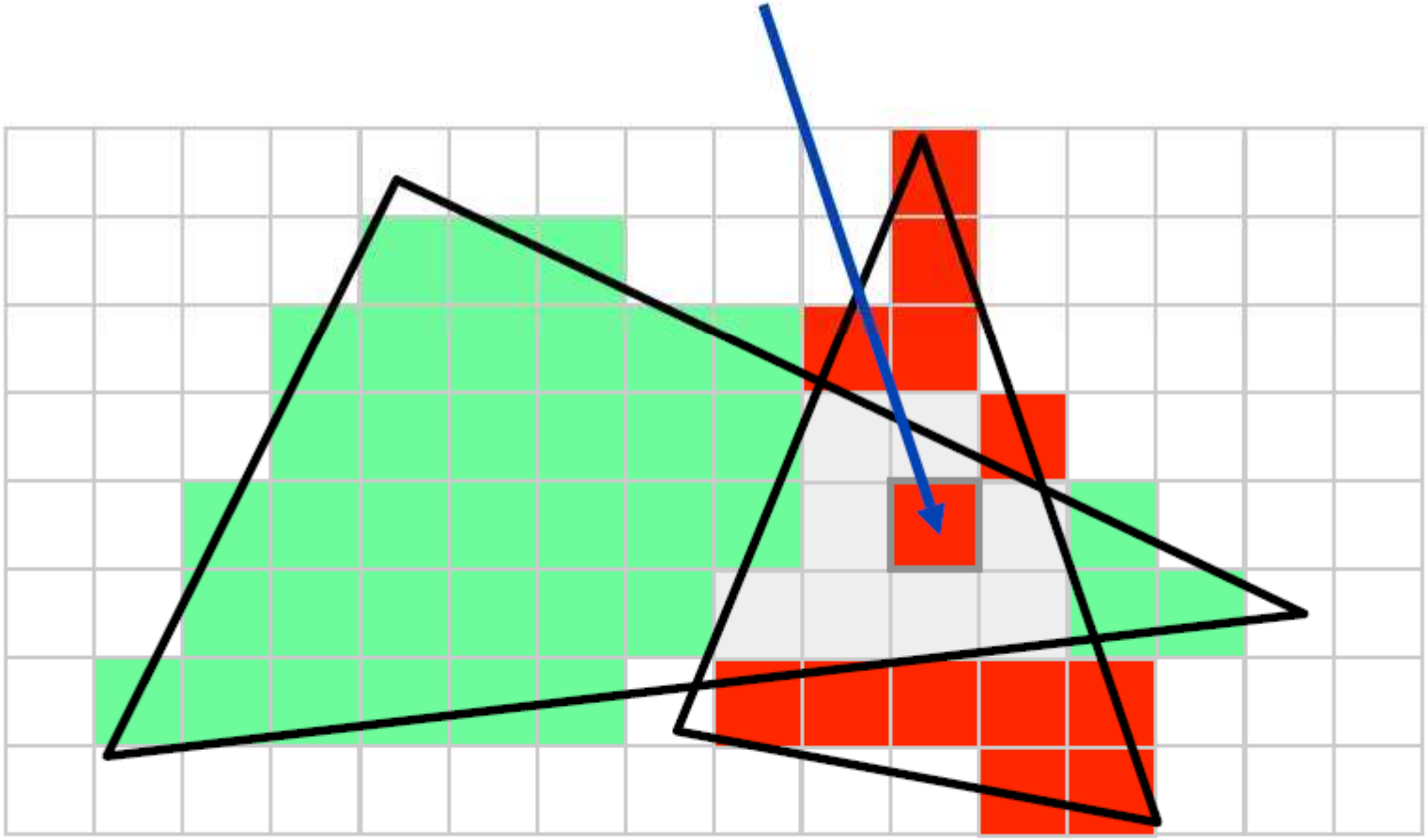
The Depth Buffer (Z-buffer)

- Example:
  - fragment from green triangle has z value of 0.7



The Depth Buffer (Z-buffer)

- Since 0.3 < 0.7, the red fragment wins



The Z-buffer

- Lots of fragments might map to the same pixel location
- How to track their z-values?
- Solution: z-buffer (2D buffer, same size as image)

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.1	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.1	0.1	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.2	0.2	0.3	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3	0.3	0.4	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.3	0.4	0.4	0.5	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.4	0.5	0.5	0.5	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.5	1.0	1.0	1.0

Z-buffer Algorithm

- Let CB be color buffer, ZB be z-buffer
- Initialize z-buffer contents to 1.0 (far away)

Z-buffer Algorithm

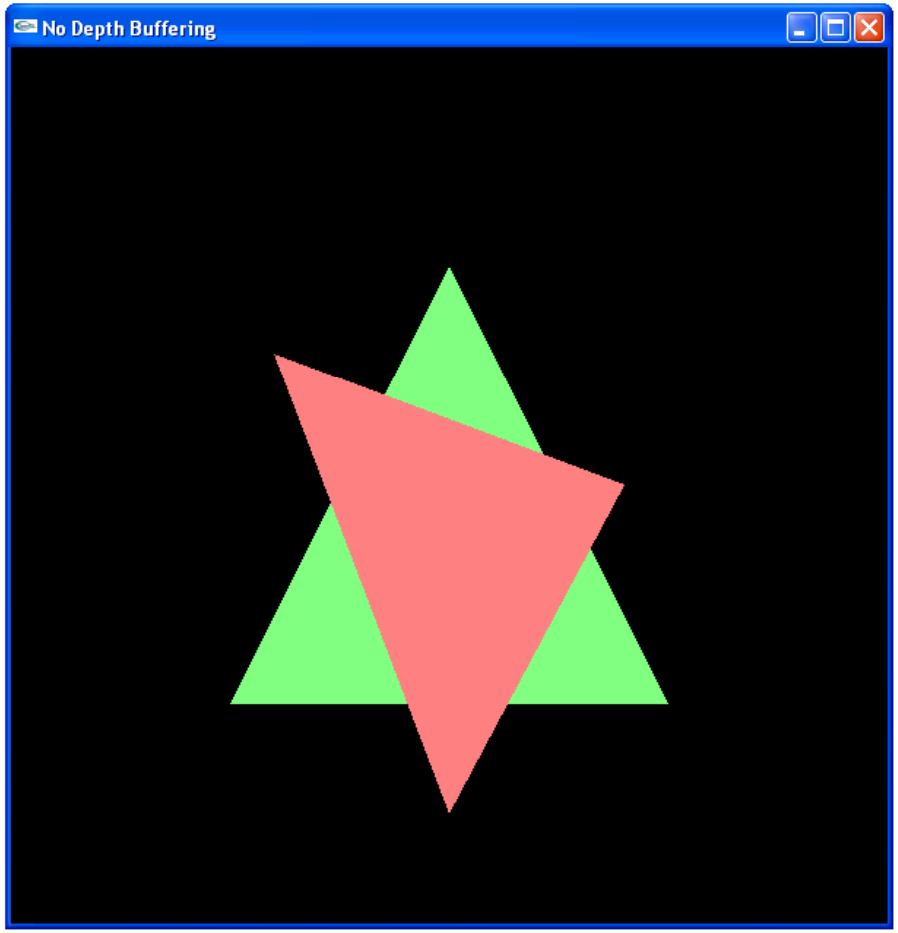
- Let CB be color buffer, ZB be z-buffer
- Initialize z-buffer contents to 1.0 (far away)
- For each triangle T
  - Rasterize T to generate fragments
    - For each fragment F with screen position (x,y,z) and color value C
      - If (z < ZB[x,y]) then
        - Update color: CB[x,y] = C
        - Update depth: ZB[x,y] = z

Z-buffer Algorithm Properties

- What makes this method nice?
  - simple (faciliates hardware implementation)
  - handles intersections
  - handles cycles
  - draw opaque polygons in any order

DEPTH BUFFER ใน OPENGL

เมื่อ Render โดยไม่มี Depth Buffer

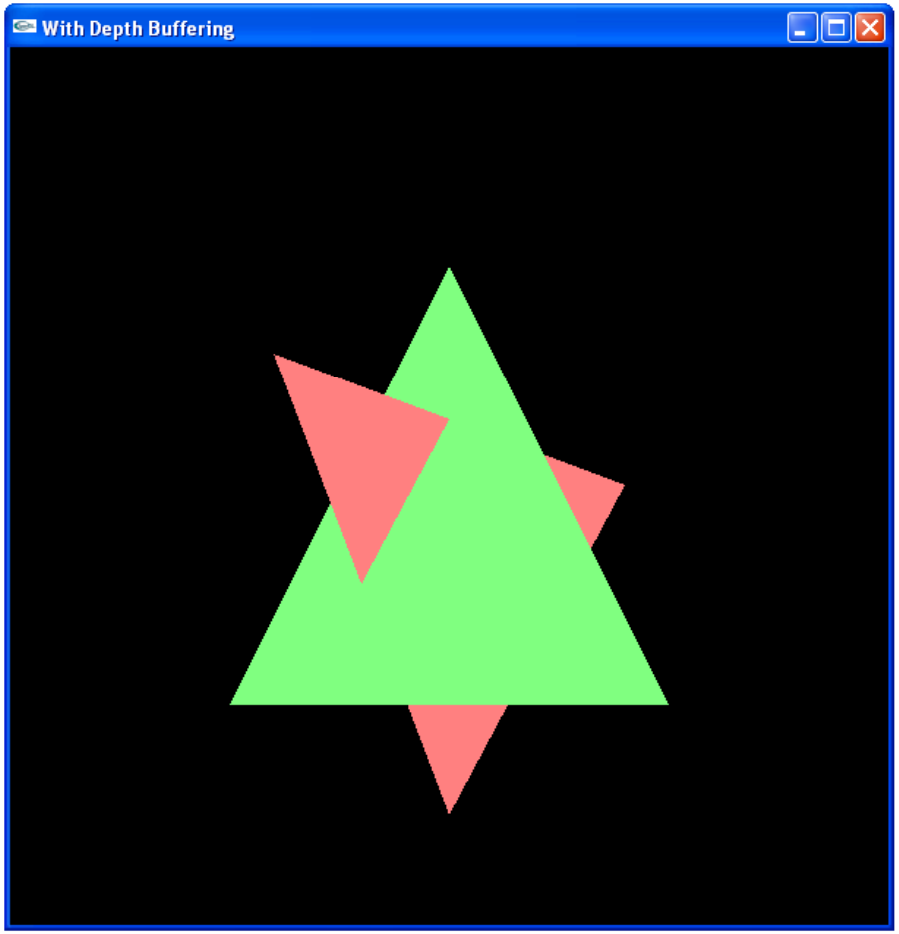


จริงๆ แล้ว code เป็นอย่างนี้

```
glColor3d(0.5, 1, 0.5);
glBegin(GL_TRIANGLES);
glVertex3d(0, 0.5, 0);
glVertex3d(-0.5, -0.5, 0);
glVertex3d(0.5, -0.5, 0);
glEnd();

glColor3d(1, 0.5, 0.5);
glBegin(GL_TRIANGLES);
glVertex3d(0, -0.75, 1);
glVertex3d(0.40, 0, 1);
glVertex3d(-0.40, 0.30, -1);
glEnd();
```

จริงๆ แล้วภาพควรจะเป็นแบบนี้



การใช้ Depth Buffer ใน GLUT

- เวลาสั่ง glutInitDisplayMode ให้เพิ่ม GLUT_DEPTH ด้วย

```
glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH)
```

- เรียก glEnable(GL_DEPTH_TEST)

- เวลาเรียก glClear ให้เพิ่ม GL_DEPTH_BUFFER_BIT

```
glClear(GL_COLOR_BUFFER_BIT | GLUT_DEPTH_BUFFER_BIT)
```

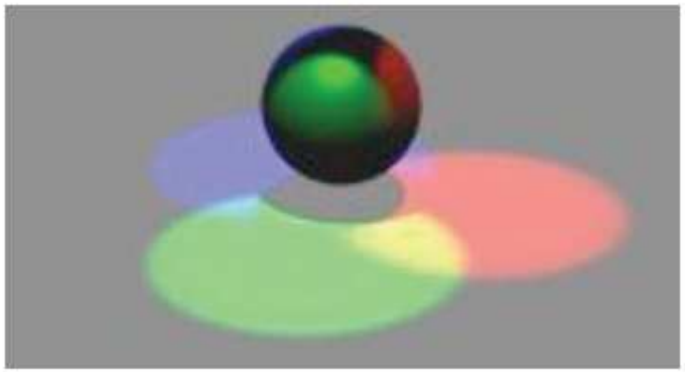
โค้ดตัวอย่าง

```
int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA |
     GLUT_DOUBLE);
  glutInitWindowSize(600, 600);
  glutCreateWindow("No Depth Buffering");
  glutDisplayFunc(display);
  glEnable(GL_DEPTH_TEST);
  glutMainLoop();
  return 0;
}
```

ILLUMINATION

Two Components of Illumination

- Light sources with:
  - Emittance spectrum (color)
  - Geometry (position and direction)
  - Directional attenuation (falloff)
- Surface properties with:
  - Reflectance spectrum (color)
  - Geometry (position, orientation, and micro-structure)
  - Absorption

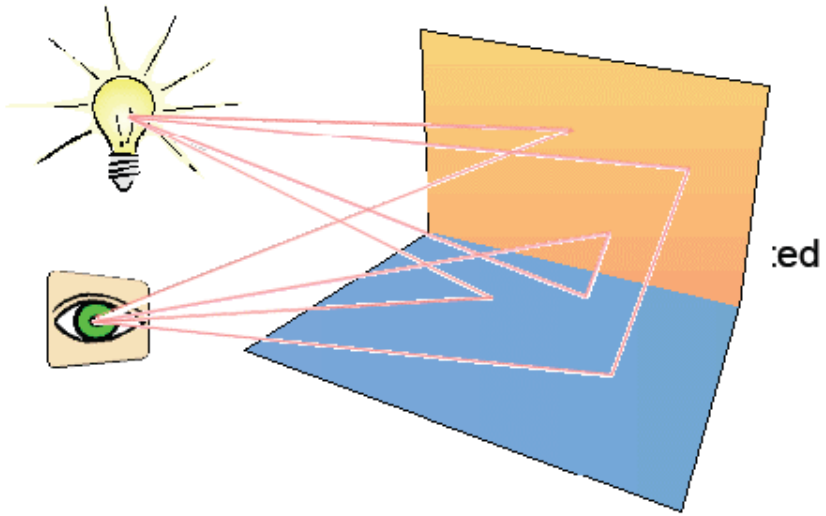


Computer Graphics Jargon

- **Illumination:** the transport of energy from light sources between points via direct and indirect paths
- **Lighting**: the process of computing the light intensity reflected from a specific 3-D point
- **Shading**: the process of assigning a color to a pixel based on the illumination in the scene

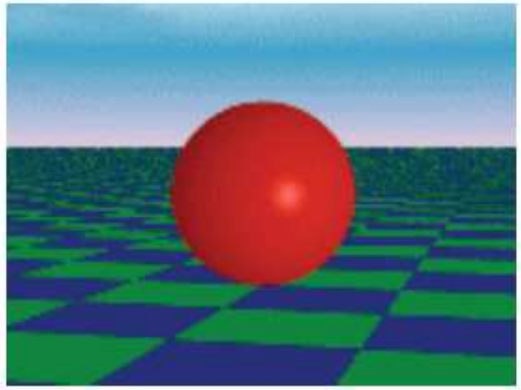
Direct and Global Illumination

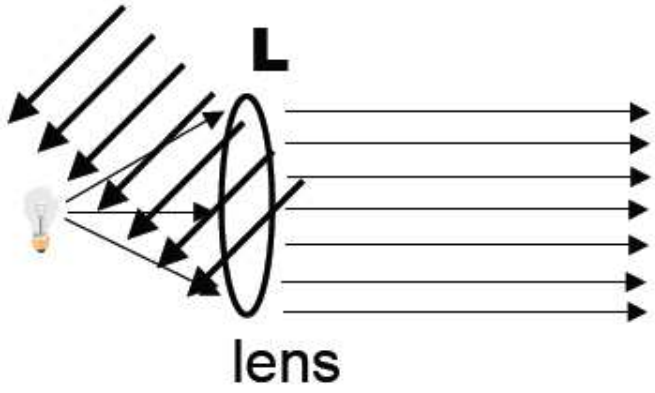
- Direct illumination: A surface point receives light directly from all light sources in the scene
  - Computed by the local illumination model
  - Determine which light sources are visible
- Global illumination: A surface point receives light after the light rays interact with other objects in the scene



Directional Light Sources

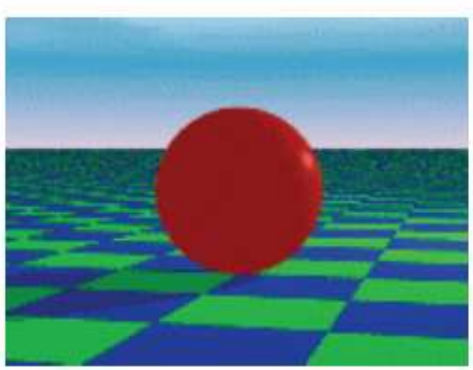
- All of the rays from a directional light source have a common direction
- The direction is a constant at every point in the scene
- It is as if the light source was infinitely far away from the surface that it is illuminating
- Examples?

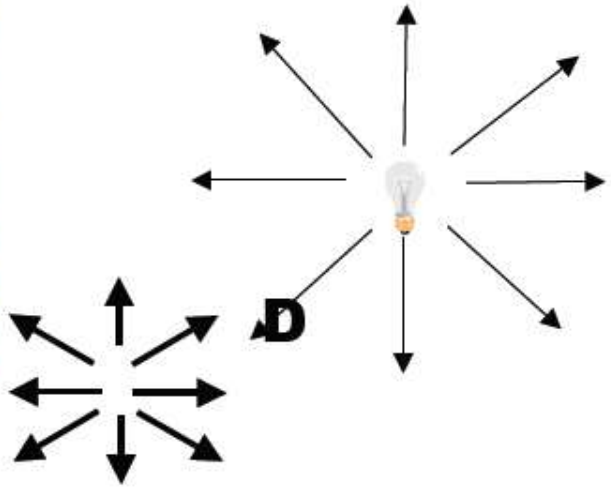




Point Light Sources

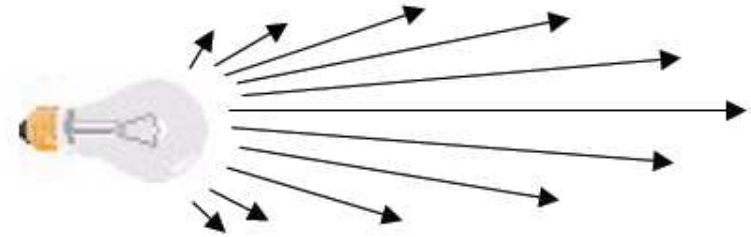
- The rays emitted from a point light radially diverge from the source
- Direction to the light changes at each point
- Examples?



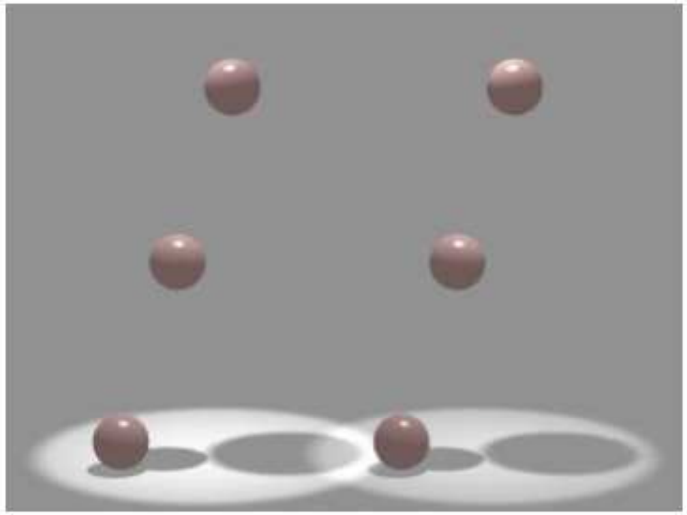


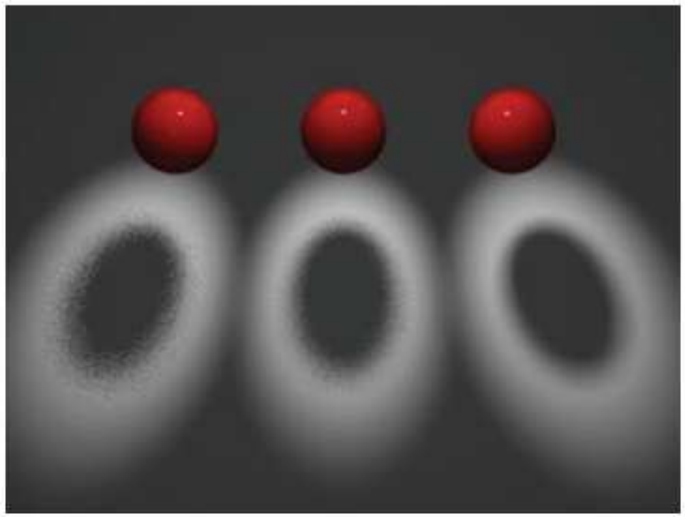
Other Light Sources

- Spotlights



- Area light sources
  - Light source occupies a 2D area (polygon)
  - Generates *soft* shadows.
- Extended light sources
  - Spherical light source
  - Generates *soft* shadows





Linearity of Light

=

+

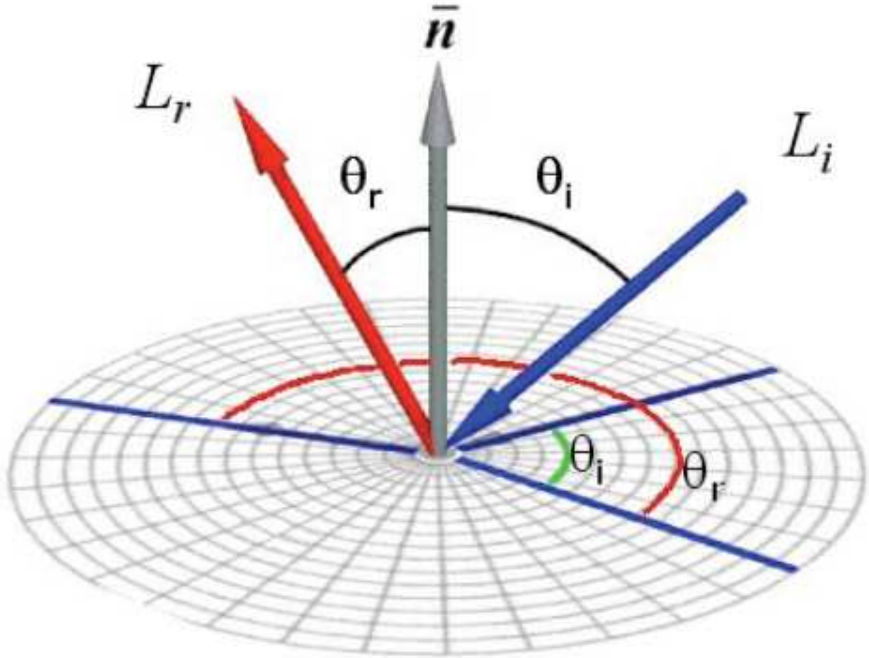
+

Paul Haeberli, Grafica Obscura

Linearity of Light

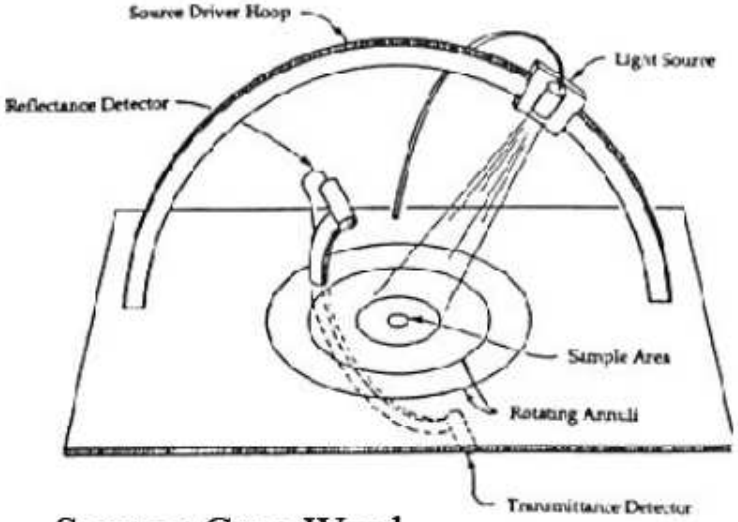
Reflectance Models

- Bi-Directional Reflection Distribution Function (BRDF)
- Difficult to measure in practice
- Often modeled analytically



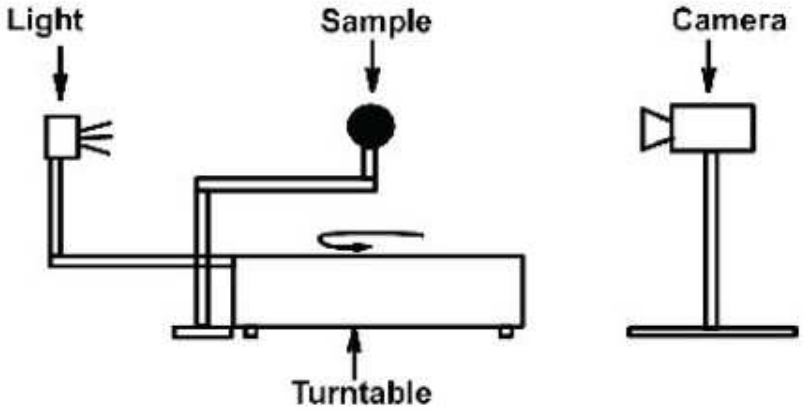
How do we obtain BRDFs?

- Gonio-reflectometer



Source: Greg Ward

- Steve Marschner / MERL

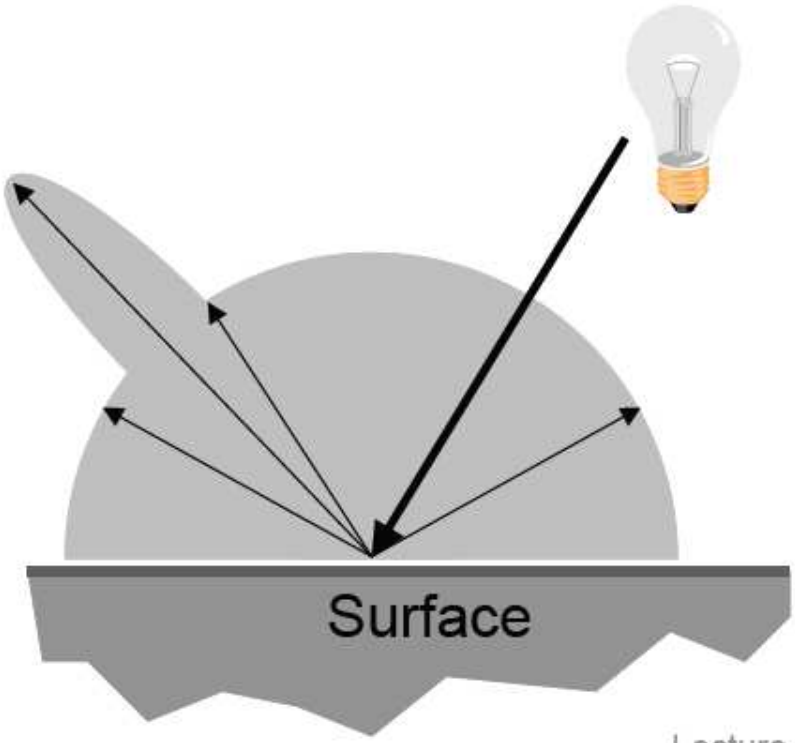


BRDF Models

- Physically-based models
  - Based on laws on physics
  - Derived from measurements
- Phenomenological (empirical) models
  - “Ad hoc” formulas that work
  - Easier to control by artist
- Data-driven models [Matusik et al. 2003]
  - Lot of data – require data reduction techniques
  - Very realistic, but harder to edit & manipulate

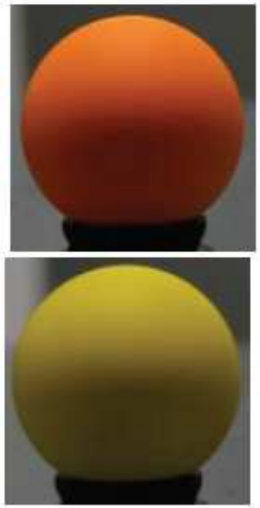
OpenGL Reflectance Model

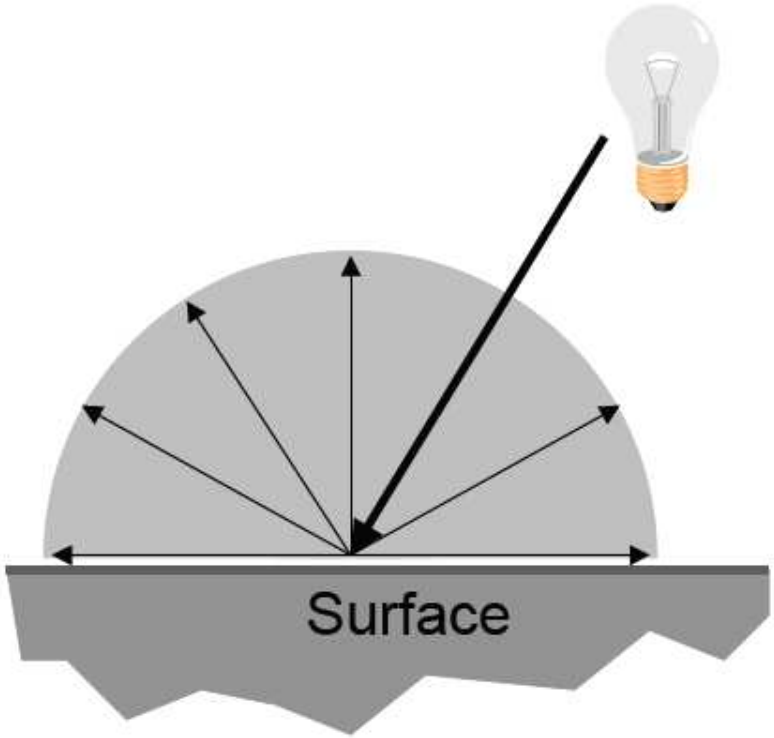
- Simple analytic model proposed by T. Phong:
  - diffuse reflection +
  - specular reflection +
  - “ambient”



Ideal Diffuse Reflectance

- Surface reflects light equally in all directions
- Why? Examples?





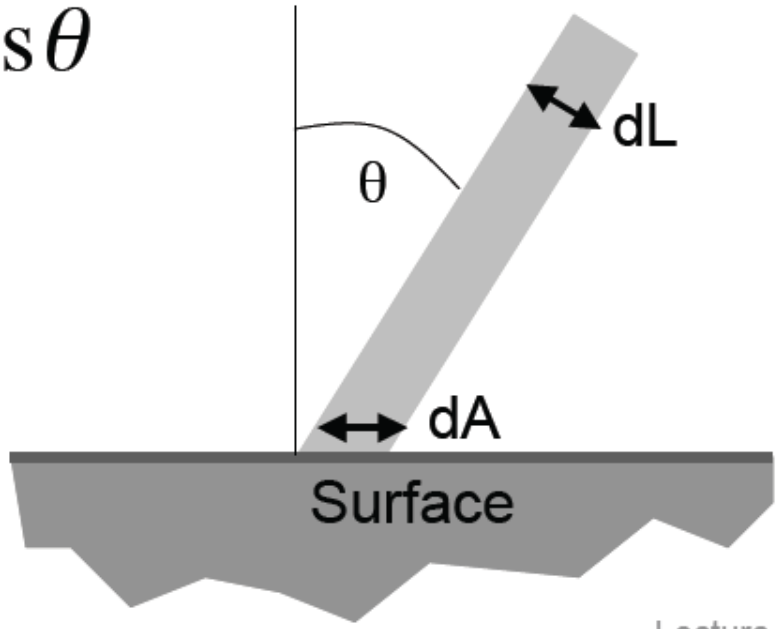
Lambert's Cosine Law

- Diffuse reflectance scales with cosine of angle

  cos θ is maximum when θ = 0°

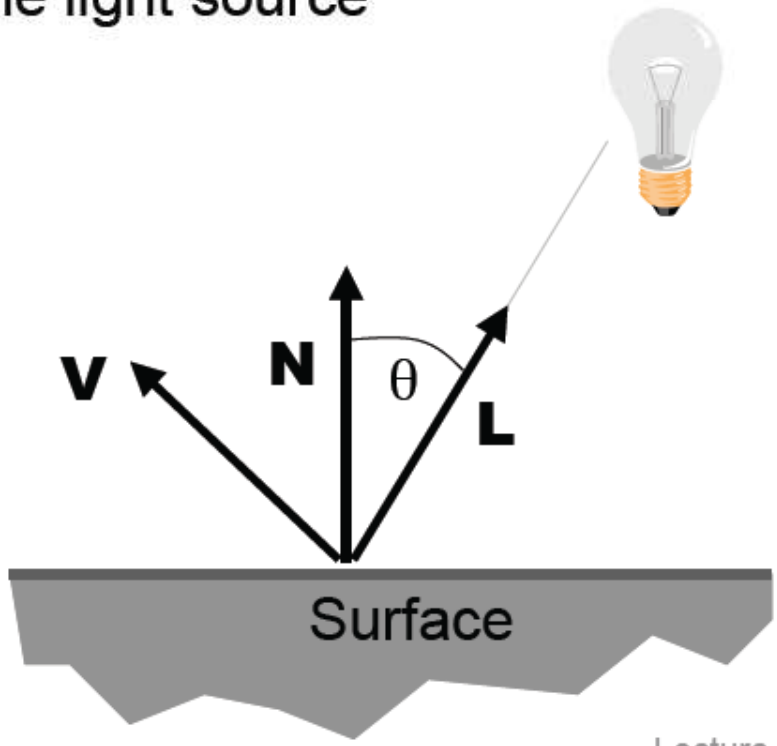
  cos θ is zero when θ = 90°

$$dL = dA\cos\theta$$



Geometric Ingredients

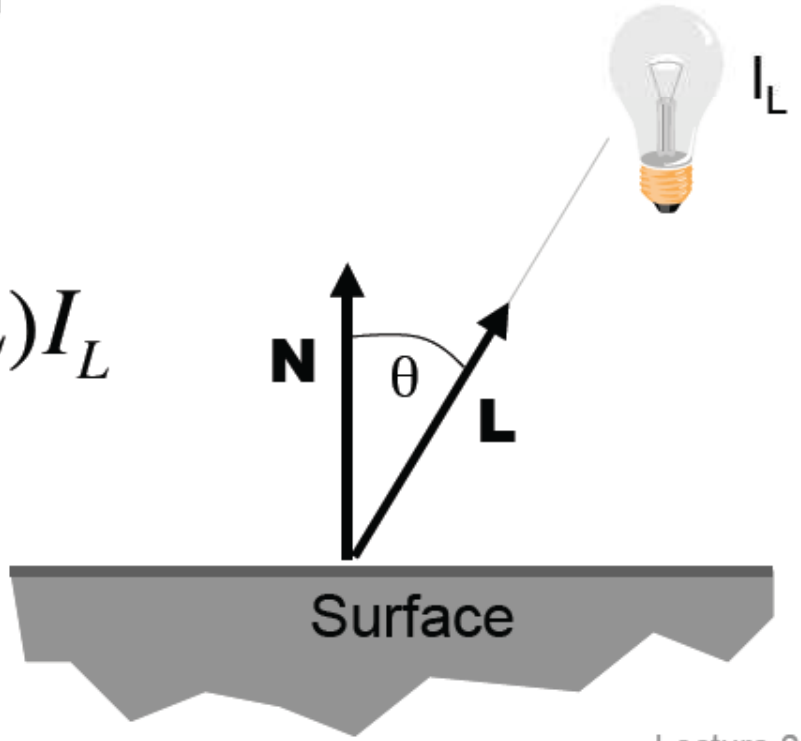
- P: Point on the surface
- **N**: Normal vector at the surface point
- **V**: Vector from P to the camera / eye
- **L**: Vector from P to the light source



Ideal Diffuse Reflectance

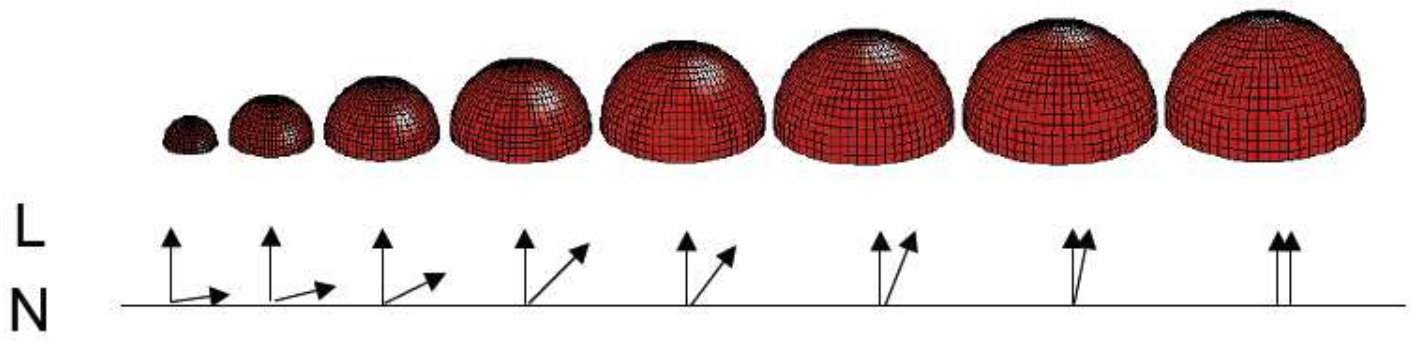
- Lambertian reflection model
  - I_L: The incoming light intensity
  - k_d: The diffuse reflection coefficient
  - N: Surface normal
  - L: Light direction

$$I_d = k_d (N \bullet L) I_L$$

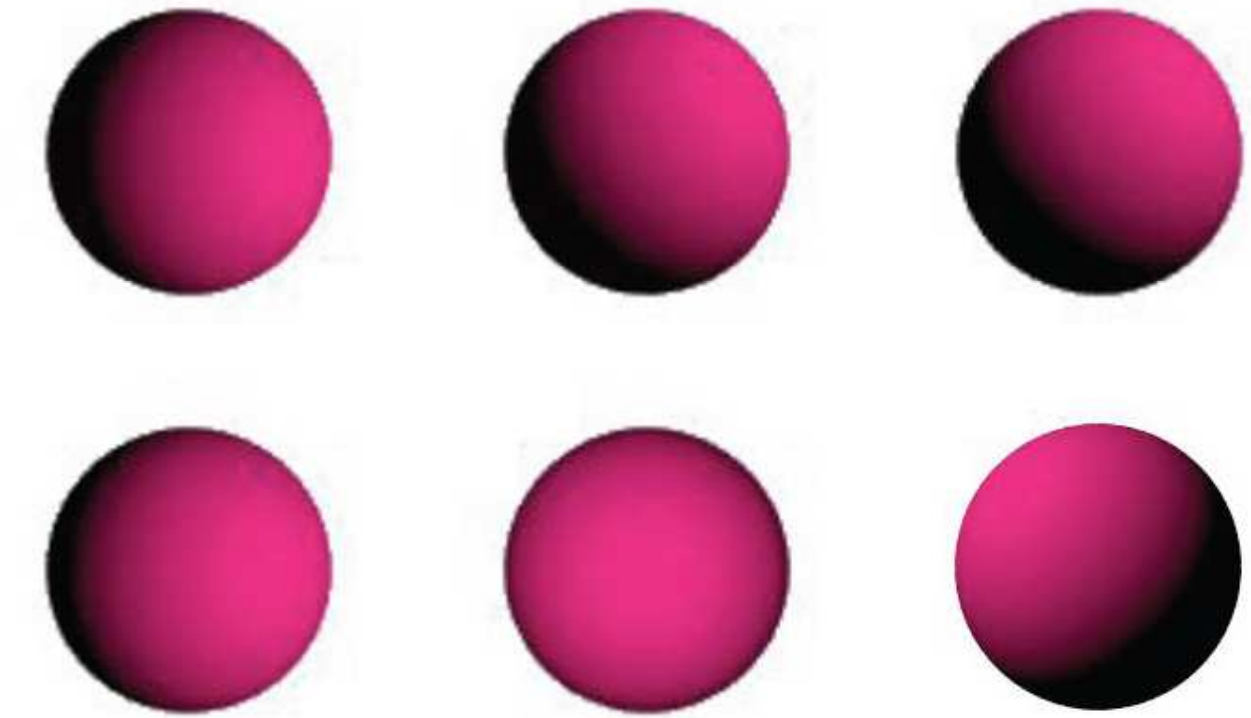


Ideal Diffuse Reflectance

- If **N** and **L** are facing away from each other, **N** • **L** becomes negative.
- Using max((**N** • **L**),0) makes sure that the result is zero.
  - From now on, we mean max() when we write •
- Do not forget to normalize your vectors for the dot product!

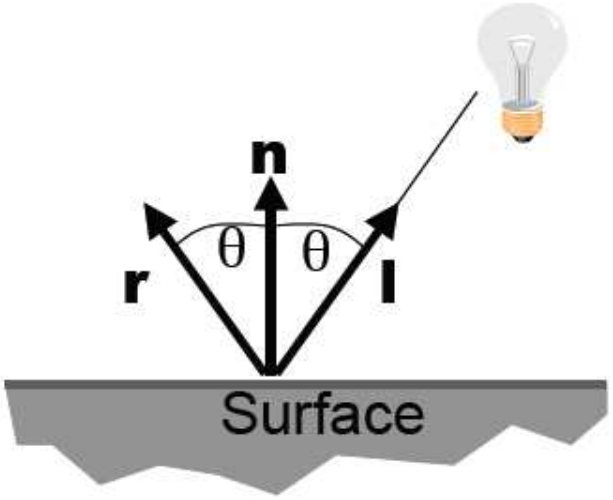


Diffuse Reflectance Example



Ideal Specular Reflectance

- Surface reflects light only at mirror angle
- Reflection is strongest near mirror angle
- Why? Examples?



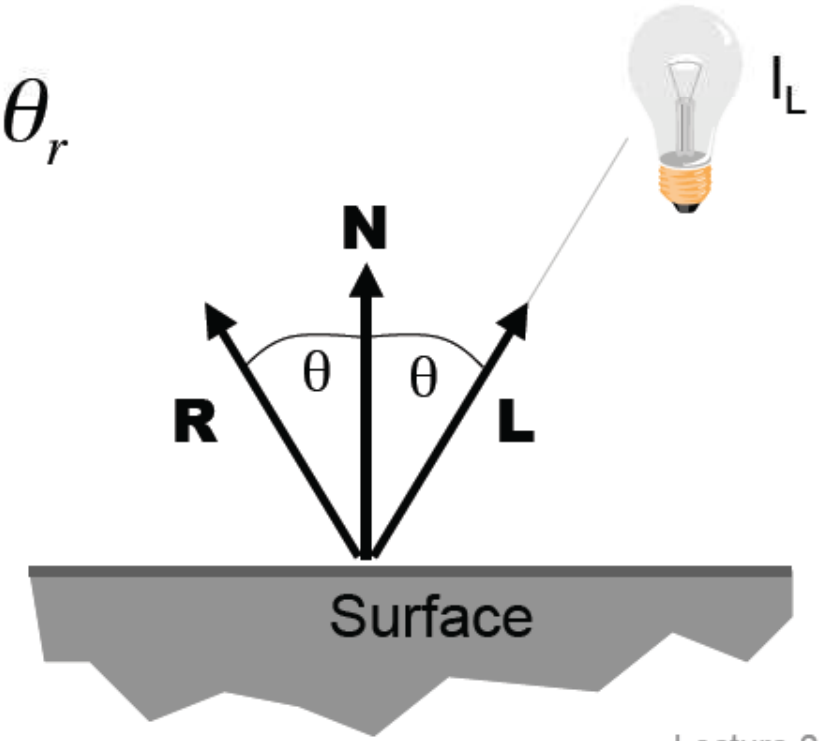
Ideal Specular Reflectance

- Special case of Snell's Law
  - The incoming ray, the surface normal, and the reflected ray all lie in a common plane

$$n_l \sin\theta_l = n_r \sin\theta_r$$

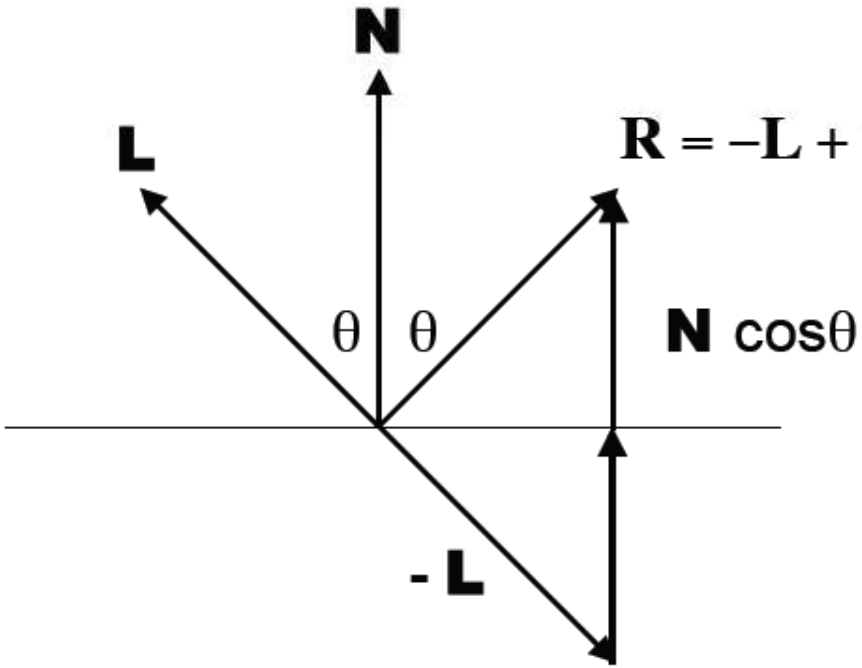
$$n_l = n_r$$

$$\theta_l = \theta_r$$



Reflection Vector R

- The vector R can be computed from the incident ray direction L and the surface normal N
- Note that all vectors have unit length



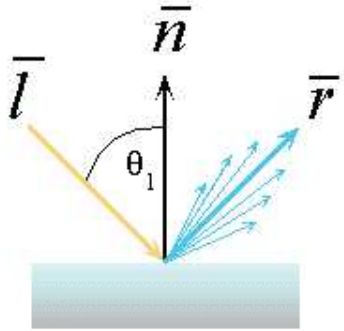
$$\mathbf{R} = -\mathbf{L} + 2\mathbf{N}cos\theta = -\mathbf{L} + 2(\mathbf{L} \bullet \mathbf{N})\mathbf{N}$$

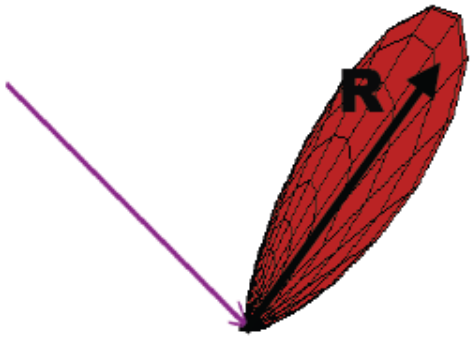
Non-ideal Reflectors

- Real materials tend to deviate significantly from ideal mirror reflectors

Non-ideal Reflectors

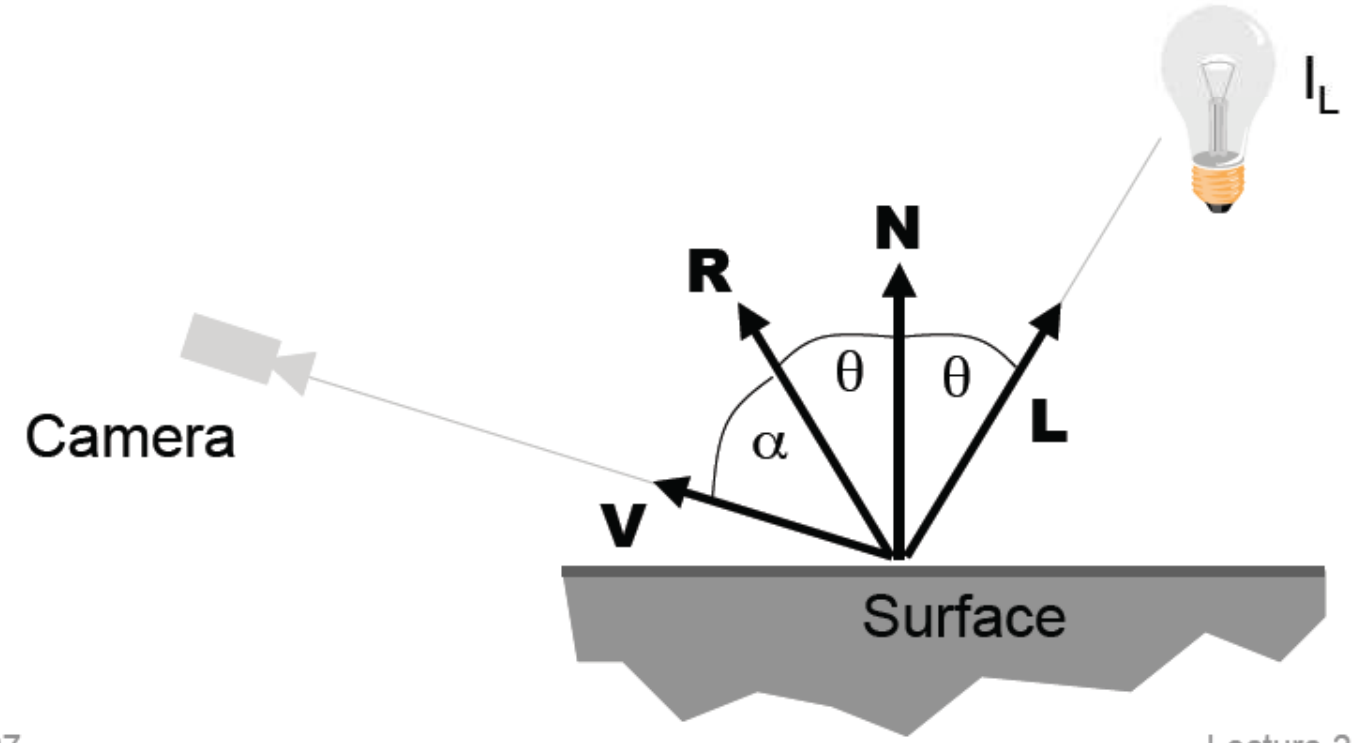
- Real materials deviate significantly from ideal reflectors
- Introduce an empirical model that is consistent with our experience
- The amount of reflected light is greatest in the direction of the perfect mirror reflection **R**
- The reflected light forms a “beam” pattern around this mirror direction





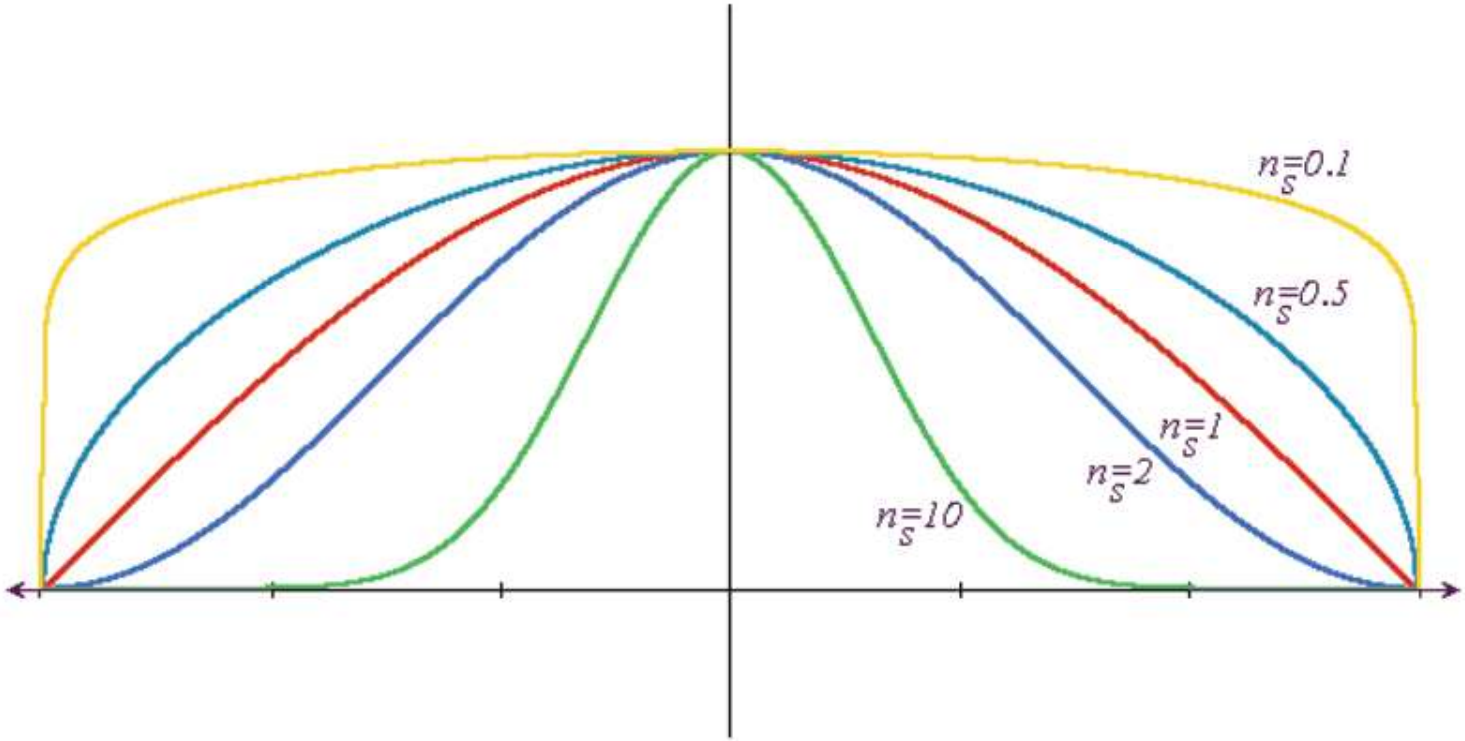
Phong Specular Reflection

$$I_s = k_s (V \bullet R)^n I_L$$



Effect of n

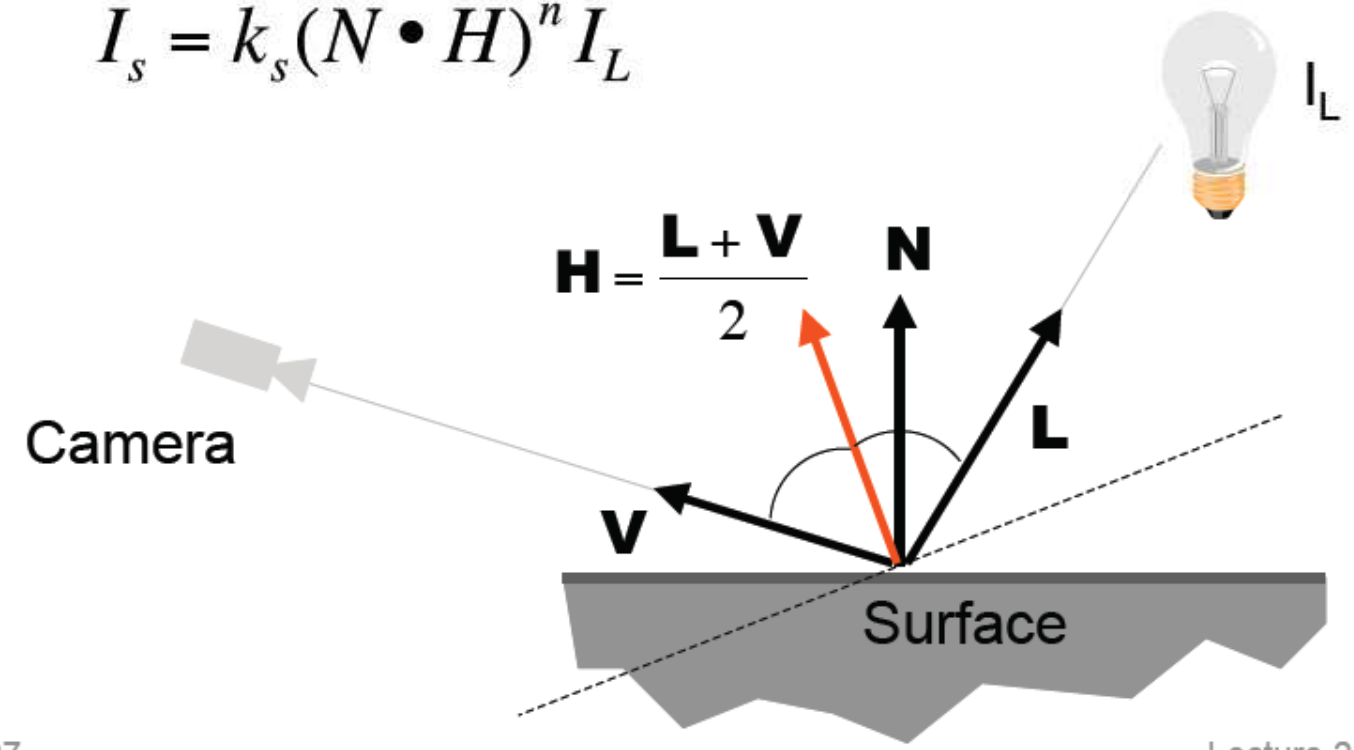
- The cosine lobe gets more narrow with increasing n.



Blinn & Torrance Variation

- Uses the halfway vector H between L and V
- H is the normal to the (imaginary) surface that maximally reflects light in the V direction

$$I_s = k_s(N \bullet H)^n I_L$$



Blinn & Torrance Variation

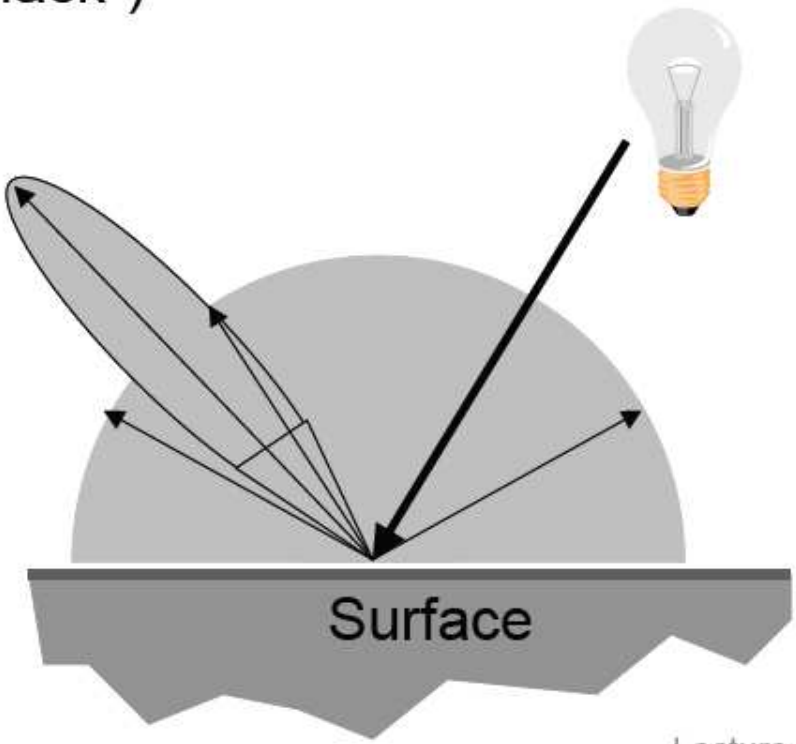
- No need to compute reflection vector **R** at every point
- I_s is a function only of **N**, if:
  - the viewer is very far away and **V** does not change for all points on the object (e.g., orthographic projection)
  - **L** does not change for all points on the object (e.g., directional lights)

The Phong Illumination Model

- Sum of three components:

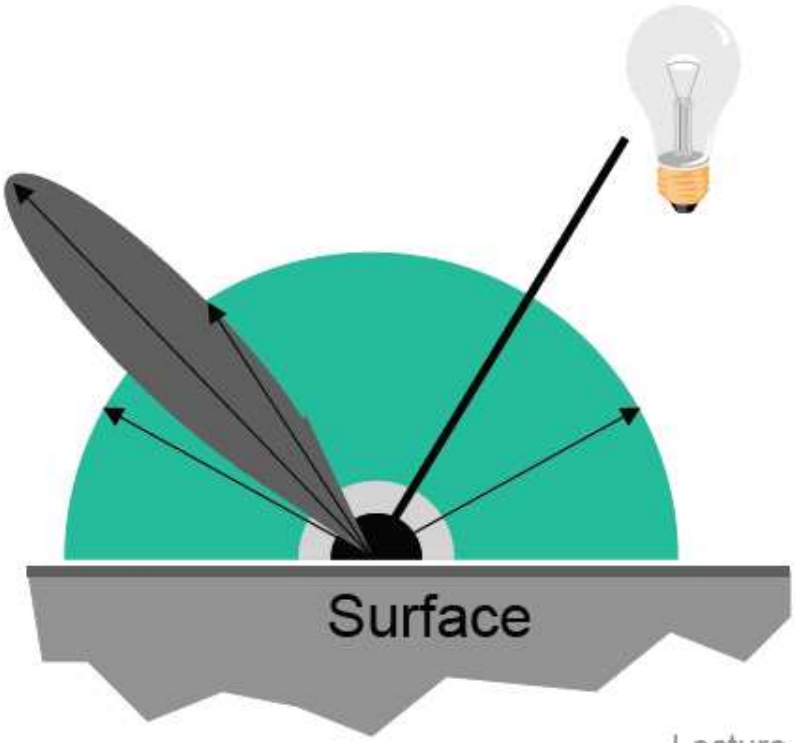
  diffuse reflection + specular reflection + "ambient"

- Ambient represents the reflection of all indirect illumination (aka a "hack")



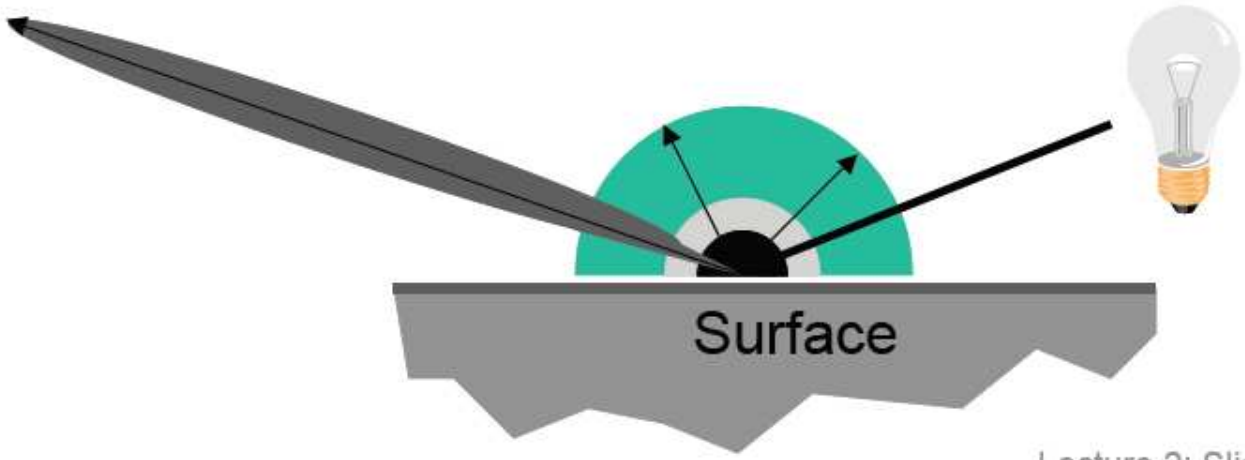
Phong Reflectance Model

- Simple analytical model:
  - diffuse reflection +
  - specular reflection +
  - "ambient"



Phong Reflectance Model

- Simple analytical model:
  - diffuse reflection +
  - specular reflection +
  - "ambient"

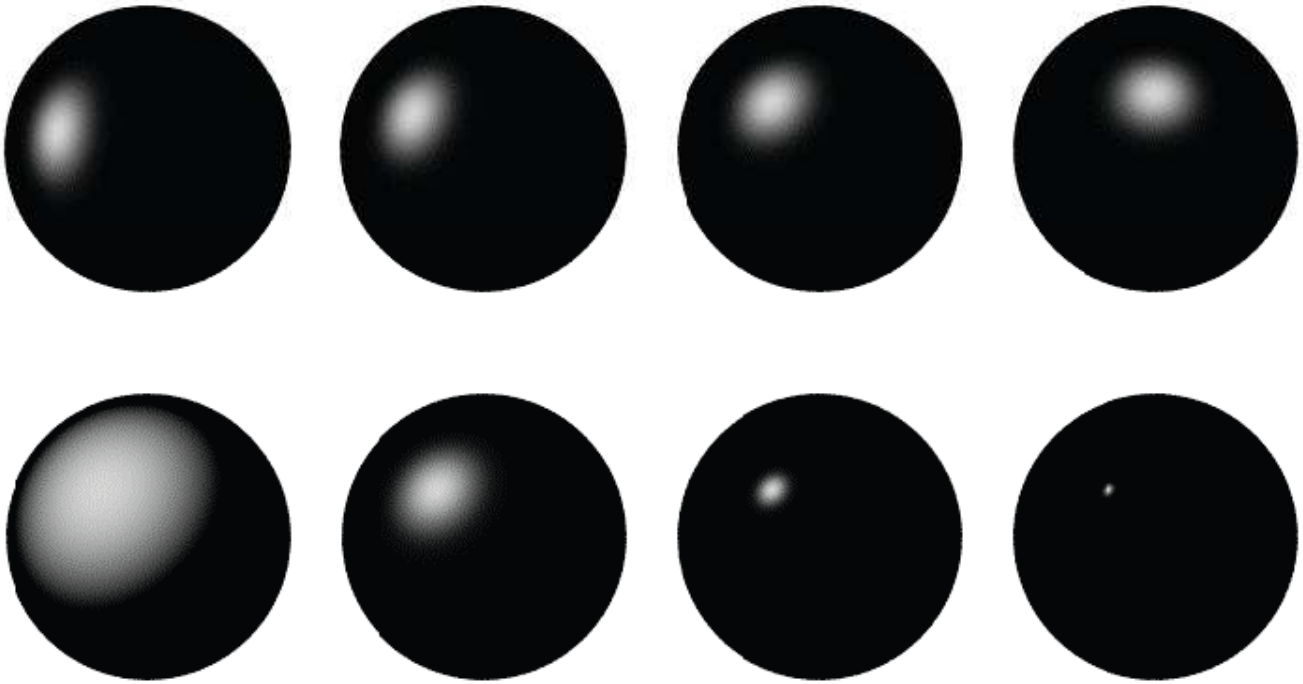


Phong Reflectance Model

Phong	$\rho_{ambient}$	$\rho_{diffuse}$	$\rho_{specular}$	ρ_{total}
$\phi_i = 60^\circ$				
$\phi_i = 25^\circ$				
$\phi_i = 0^\circ$				

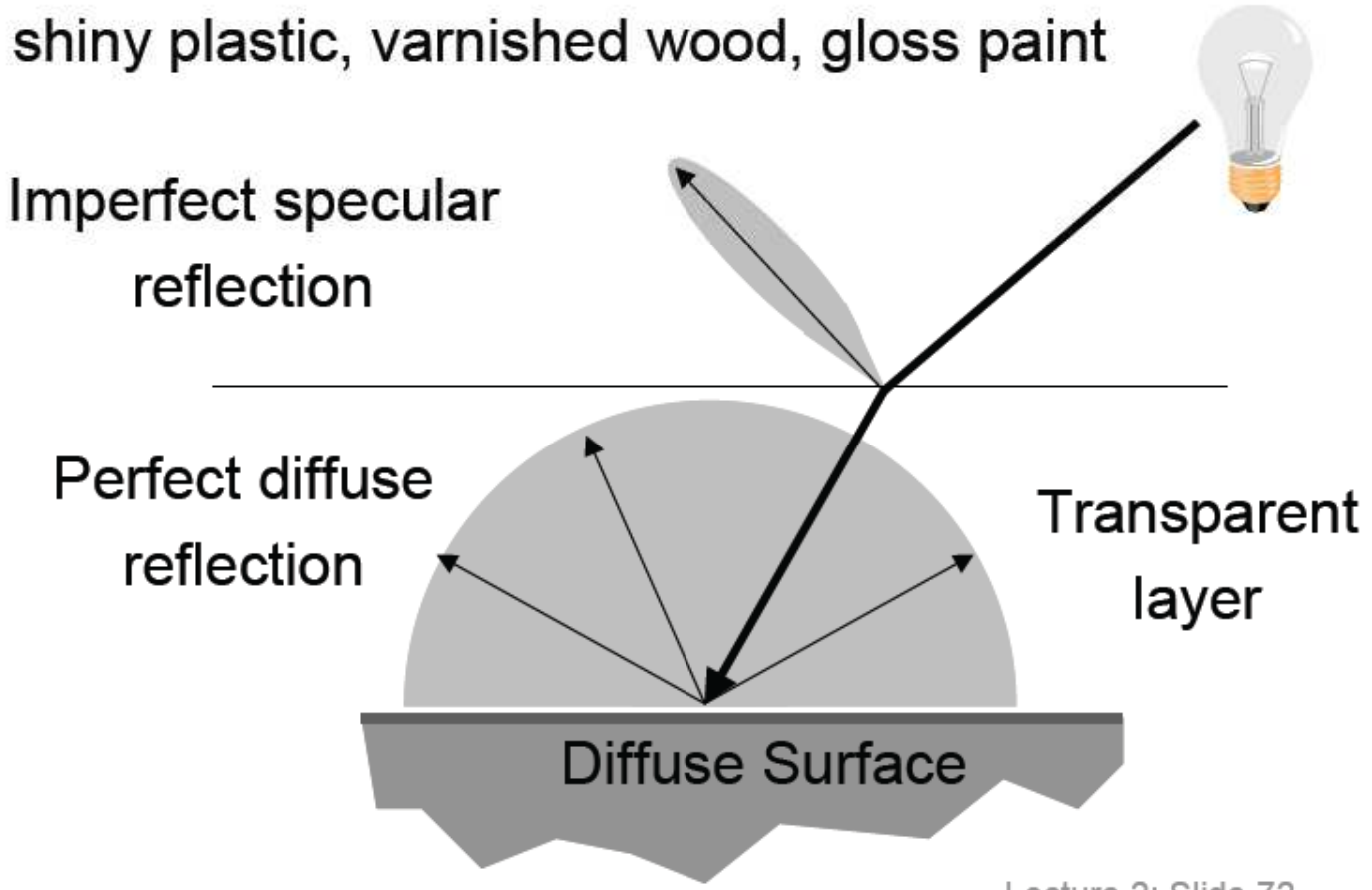
Phong Examples

- The direction of the light source and the n are varied



The Plastic Look

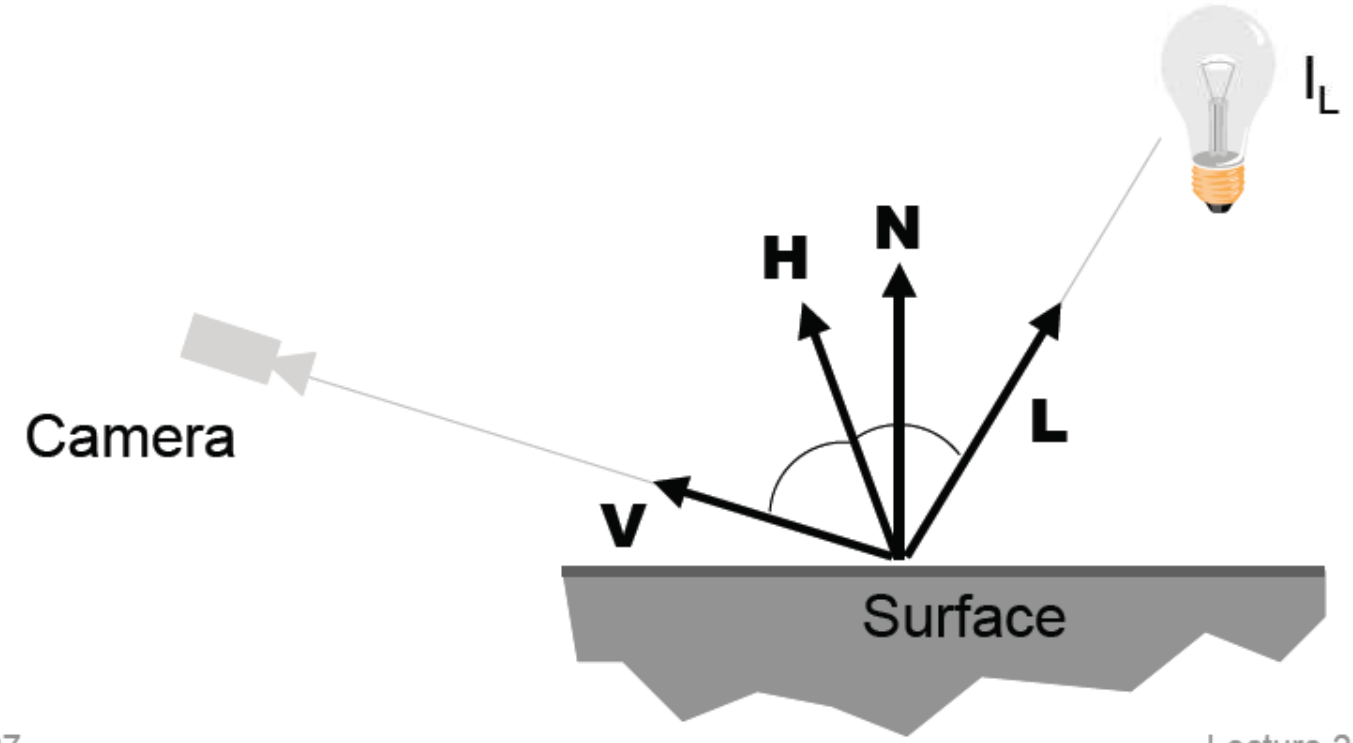
- The Phong illumination model is an approximation of a surface with a specular and a diffuse layer
  - E.g., shiny plastic, varnished wood, gloss paint



Phong Reflectance Model

- Single light source:

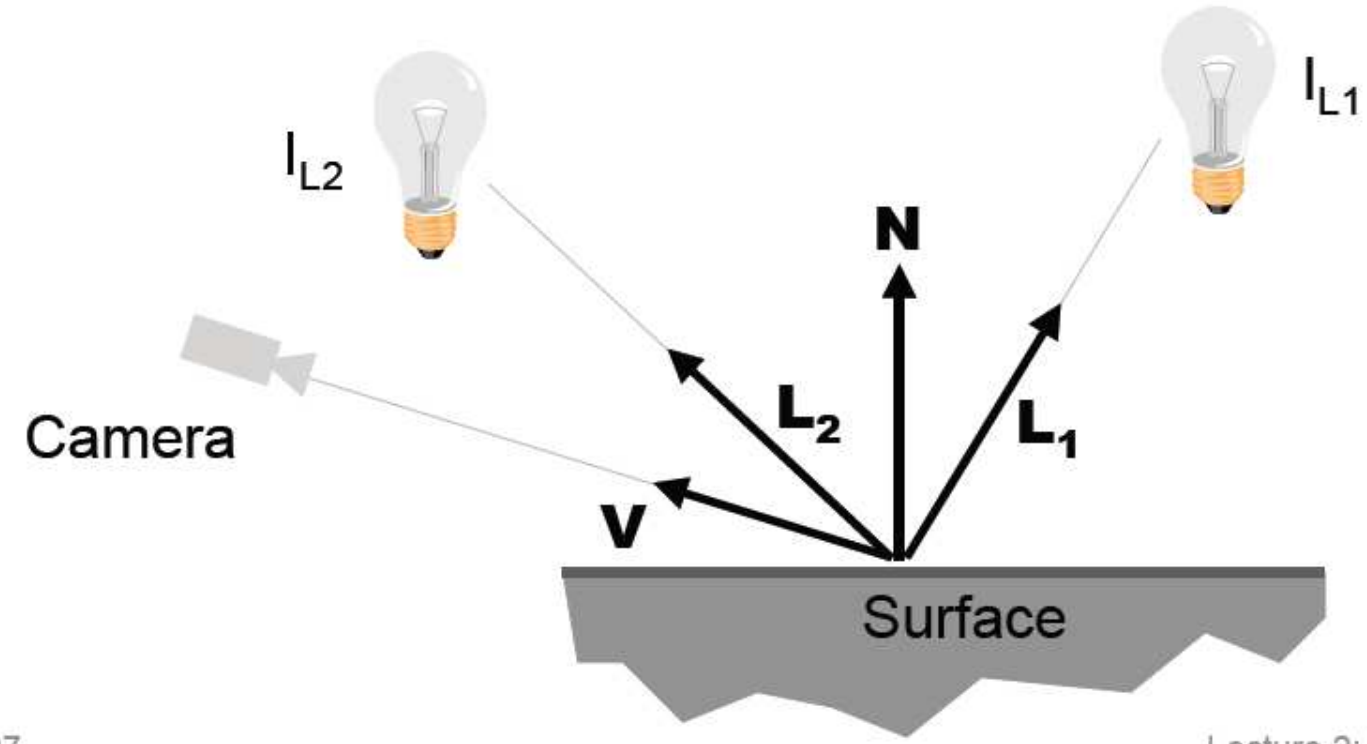
$$I = k_a I_a + k_d (N \bullet L) I_L + k_s (N \bullet H)^n I_L$$



Phong Reflectance Model

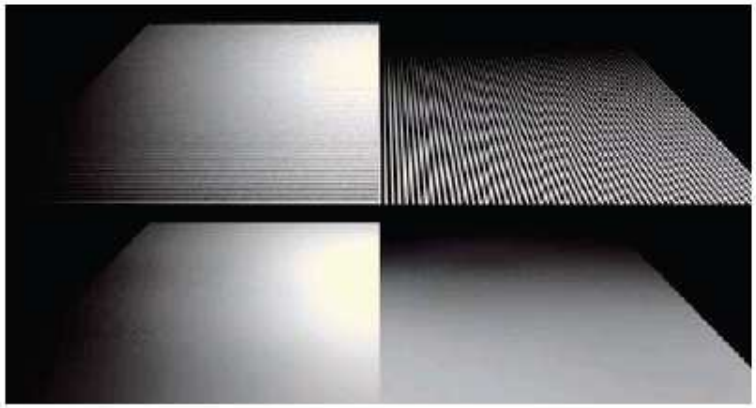
- Multiple light sources:

$$I = k_a I_a + \sum_i \left(k_d (N \bullet L_i) I_i + k_s (N \bullet H_i)^n I_i \right)$$



Anisotropic BRDFs

- Surfaces with strongly oriented micro-elements
- Examples:
  - Brushed metals,
  - Hair, fur, cloth, velvet



Source: Westin et.al 92