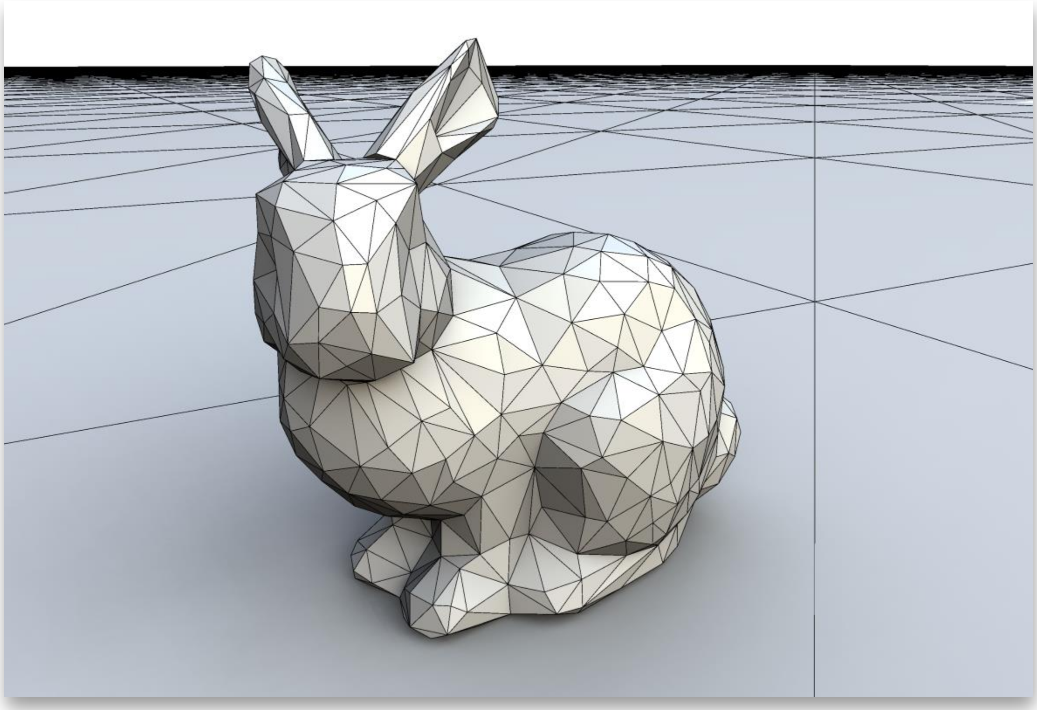
Graphics 2014



The Rasterization Pipeline

Projection, Visibility,
& Shading

[Faculty of Science]
Information and
Computing Sciences

Universiteit Utrecht

Announcements

Practicals this week

Tuesday (today)

- Tue 9-11 (was held)
- Tue 13-15 canceled (programming contest)

Wednesday (tomorrow)

- Wed 15-17: additional practical slot
- We 17-19: additional practical slot

Thursday: no practicals

Addendum:

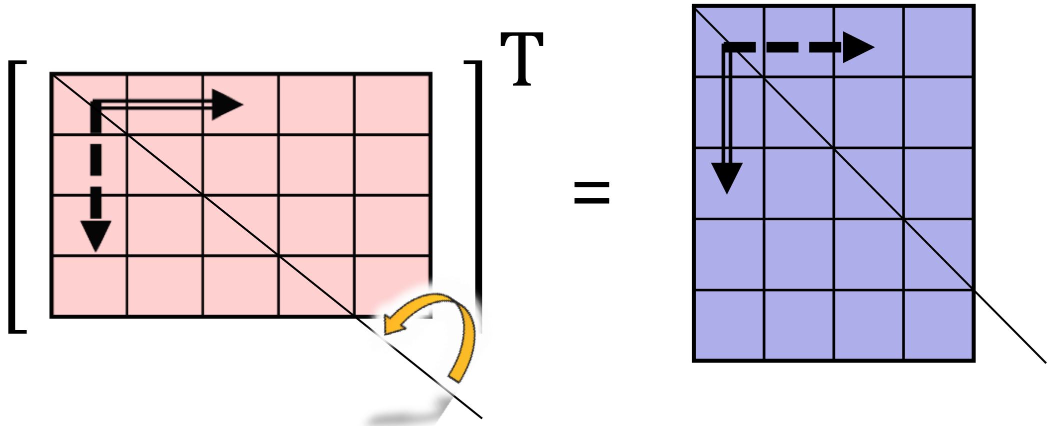
Matrix Algebra

CORE

core topics

important

Transposition



Matrix Transposition

- Swap rows and columns
  - In other words: Flip around diagonal
- Formally:

$$\begin{bmatrix} \ddots & \cdot & \iddots \\ \cdot & \cdot & \cdot \\ \cdot & a_{i,j} & \cdot \\ \cdot & \cdot & \cdot \\ \iddots & \cdot & \ddots \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \ddots & \cdot & \cdot & \cdot & \iddots \\ \cdot & \cdot & a_{j,i} & \cdot & \cdot \\ \iddots & \cdot & \cdot & \cdot & \ddots \end{bmatrix}$$

Orthogonal Matrices

Othogonal Matrices

- (i.e., column vectors ortho*normal*)

$$\mathbf{M}^T = \mathbf{M}^{-1}$$

- Proof: next three slides

Matrix Multiplication

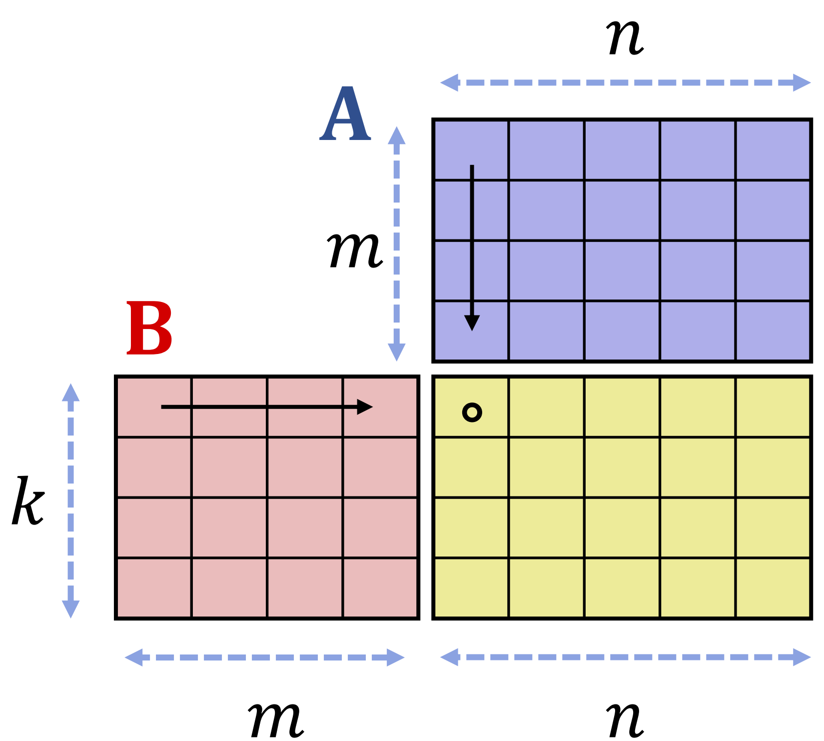
General matrix products:

- $\mathbf{B} \cdot \mathbf{A}$: possible if #Row($\mathbf{A}$) = #Columns($\mathbf{B}$)

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} b_{1,1} & \cdots & b_{1,m} \\ \vdots & & \vdots \\ b_{k,1} & \cdots & b_{k,m} \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} r_{1,1} & \cdots & r_{1,n} \\ \vdots & & \vdots \\ r_{k,1} & \cdots & r_{k,n} \end{bmatrix}$$



$$\mathbf{R} = \mathbf{B} \cdot \mathbf{A}$$

$$r_{i,j} = \sum_{q=1}^{m} a_{q,j} \cdot b_{i,q}$$

Matrix Multiplication

Matrix Multiplication

$$\mathbf{A} \cdot \mathbf{B}$$

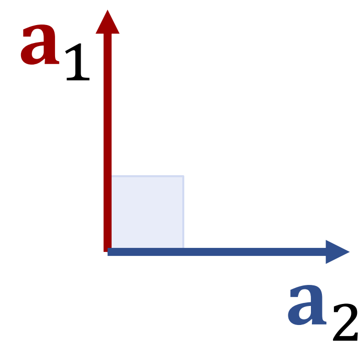
$$= \begin{pmatrix} - & \mathbf{a}_1 & - \\ & \vdots & \\ - & \mathbf{a}_d & - \end{pmatrix} \cdot \begin{pmatrix} | & & | \\ \mathbf{b}_1 & \cdots & \mathbf{b}_d \\ | & & | \end{pmatrix}$$

$$= \begin{pmatrix} \ddots & & ⋰ \\ & \langle \mathbf{a}_i, \mathbf{b}_j \rangle & \\ ⋰ & & \ddots \end{pmatrix}$$

- Scalar products of rows and columns

Matrix Multiplication

Othogonal matrices:



$$\mathbf{A}^{\mathrm{T}} \cdot \mathbf{A}$$

$$= \begin{pmatrix} - & \mathbf{a}_1 & - \\ & \vdots & \\ - & \mathbf{a}_d & - \end{pmatrix} \cdot \begin{pmatrix} | & & | \\ \mathbf{a}_1 & \cdots & \mathbf{a}_d \\ | & & | \end{pmatrix}$$

$$= \begin{pmatrix} \ddots & & ⋰ \\ & \langle \mathbf{a}_i, \mathbf{a}_j \rangle & \\ ⋰ & & \ddots \end{pmatrix} = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix} = \mathbf{I}$$

Transposition Rules

Transposition

- Multiplication: $(\mathbf{A} \cdot \mathbf{B})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}} \cdot \mathbf{A}^{\mathrm{T}}$
- Inversion: $(\mathbf{A} \cdot \mathbf{B})^{-1} = \mathbf{B}^{-1} \cdot \mathbf{A}^{-1}$
- Inverse-transp.: $\left(\mathbf{A}^{\mathrm{T}}\right)^{-1} = (\mathbf{A}^{-1})^{\mathrm{T}}$
- Othogonality: $[\mathbf{A}^{\mathrm{T}} = \mathbf{A}^{-1}] \Leftrightarrow [\mathbf{A}\ \textbf{is orthogonal}]$

Homogeneous Coordinates

(short version)

CORE

core topics
important

Problem

Translations are not linear

- $\mathbf{x} \rightarrow \mathbf{M}\mathbf{x}$ cannot encode translations
- **Proof:** Origin cannot be moved:

$$\mathbf{M} \cdot \mathbf{0} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Homogeneous Coordinates

Solution: Just add a constant one

- Increase dimension $\mathbb{R}^d \rightarrow \mathbb{R}^{d+1}$
- Last entry = 1 in vectors
  - "Cheap Trick", "Evil Hack"

$$\mathbf{M}' \cdot \mathbf{x} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & t_1 \\ m_{21} & m_{22} & m_{23} & t_2 \\ m_{31} & m_{32} & m_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} \ddots & & \iddots & | \\ & \mathbf{M} & & \mathbf{t} \\ \iddots & & \ddots & | \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} | \\ \mathbf{x} \\ | \\ 1 \end{pmatrix} = \begin{pmatrix} | \\ \mathbf{M}\mathbf{x} + \mathbf{t} \\ | \\ 1 \end{pmatrix}$$

Homogeneous Coordinates

General case

$$\mathbf{M} \cdot \mathbf{x} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix}$$

- w' might be different from 1
- Convention: Divide by w-coord. before using

Result: $\begin{pmatrix} x'/w' \\ y'/w' \\ z'/w' \\ 1 \end{pmatrix}$

Homogeneous Coordinates

General case

$$\mathbf{M} \cdot \mathbf{x} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} \equiv \begin{pmatrix} y_1/y_4 \\ y_2/y_4 \\ y_3/y_4 \\ 1 \end{pmatrix}$$

- Can express *divisions by common denominator*

$$y_4 = m_{41}x_1 + m_{42}x_2 + m_{43}x_3 + m_{44}x_4$$

- Rules:
  - Before using as 3D point, divide by last (4th) entry
  - No normalization required during subsequent transformations (matrix-mult.)

The Full Story?

Projective Geometry

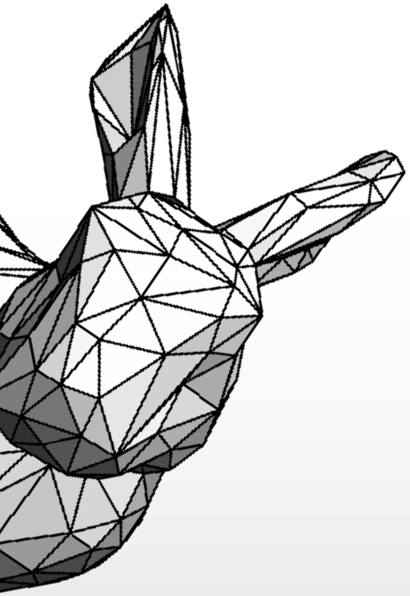
- Not just an evil hack
- Deep & interesting theoretical background
- More on this later

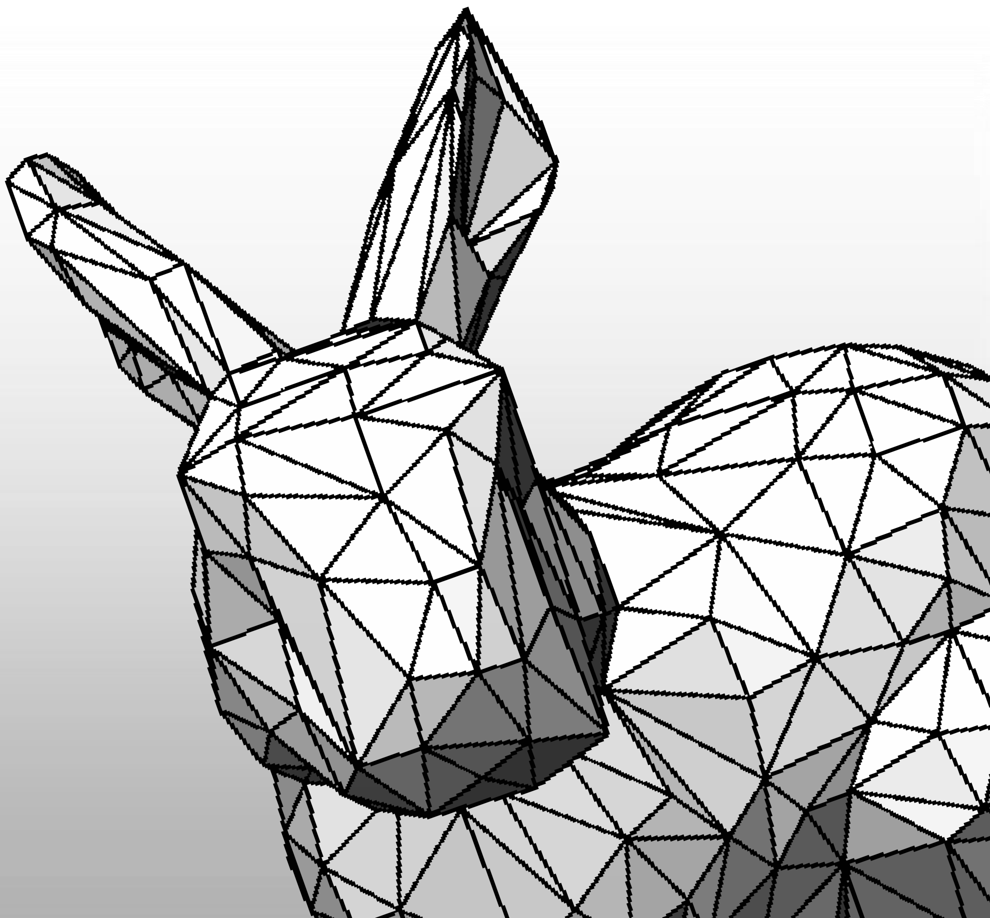
For simplicity

- We'll treat it as a computational trick for now
  - Focus on the graphics application
- Remember for now:
  - We can build "4D Translation matrices" for 3D+1 points
  - We can "divide" by a common linear factor

Now for

3D Rendering





3D Rendering Overview

BASIC

basic topics

study completely

3D Computer Graphics

Three main aspects

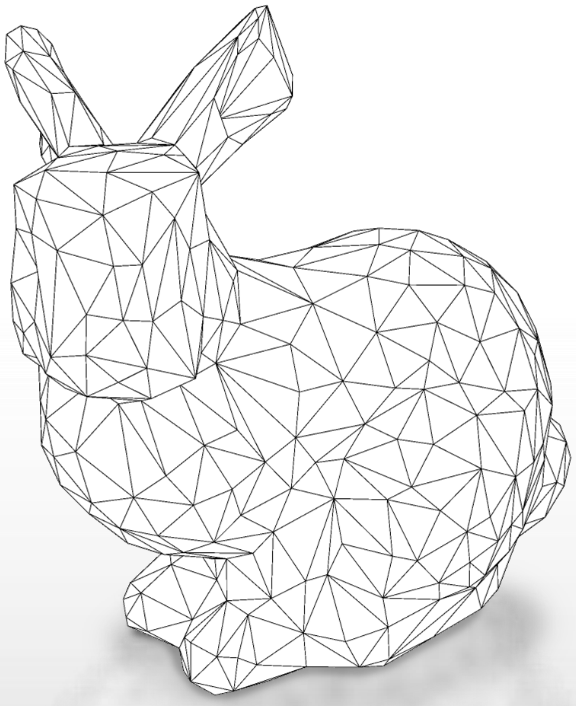
- Modeling
  - Describe 3D geometry mathematically
    - From machine parts (e.g., CAD)
    - To natural phenomena (e.g., fractals)
- Animation
  - Set scenes into motion
    - Simple: Camera fly-through
    - Complex: Fluid simulation, human motion
- Rendering
  - Convert geometry into images
  - Our Focus right now

INFODDM
Driedimensionaal modelleren

INFOMCANIM
Computer Animation
INFOMGP
Game Physics

Our Main Topic
(continued: **INFMAGR**, Advanced Graphics)

3D Rendering



Assumption

- 3D Model is given
- Triangle mesh
  (for simplicity)

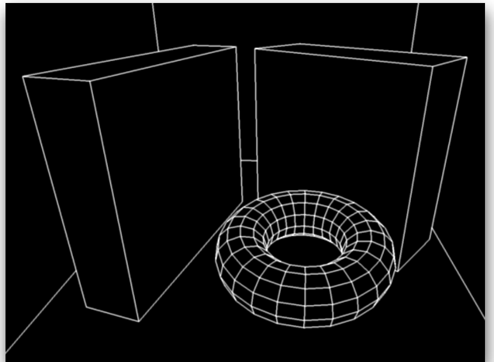
How do we get it to the screen?

Agenda

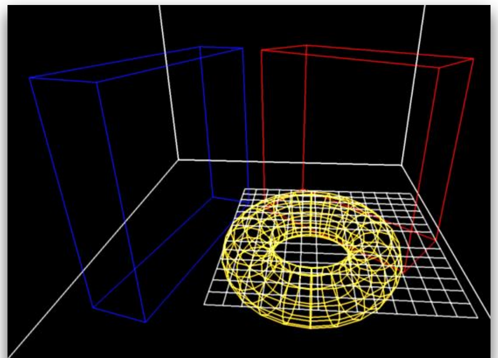
Upcoming Topics

- Modeling: mesh representation
- Physics: Perspective projection
- Rendering: Two main rendering methods
  - Rasterization
    - Perspective projection
    - Rasterization
    - Visibility
    - Shading
    - Programmable shaders / GPUs
  - Raytracing

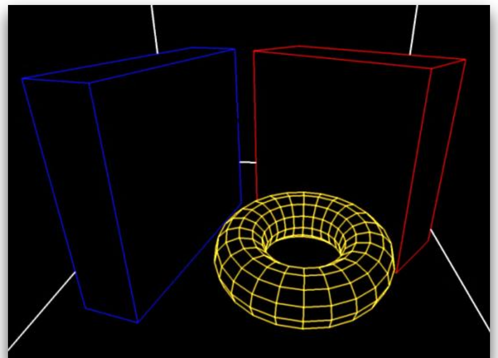
3D Rendering Steps



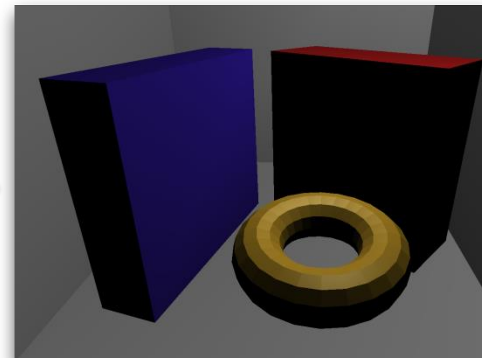
Geometric Model



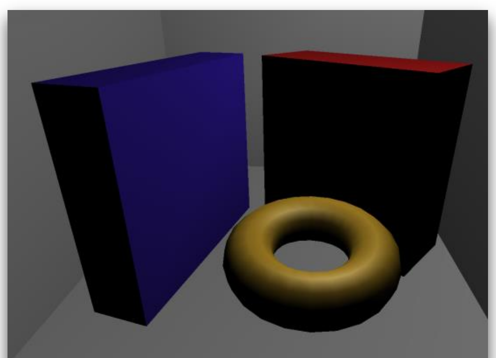
Perspective



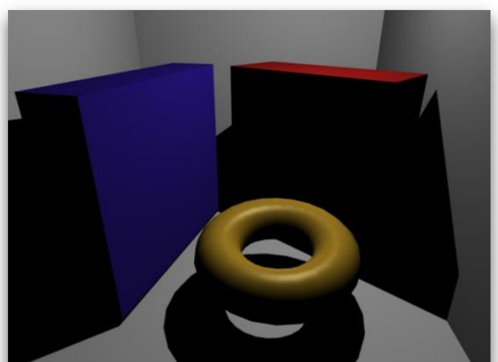
Visibility



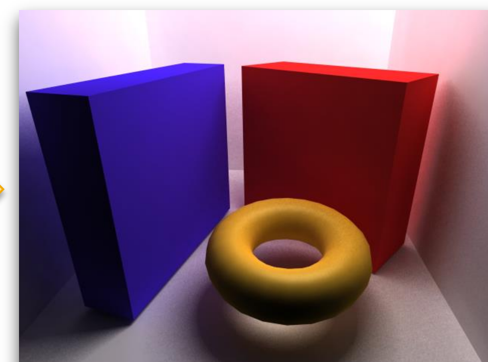
Local Illumination



Smooth Shading

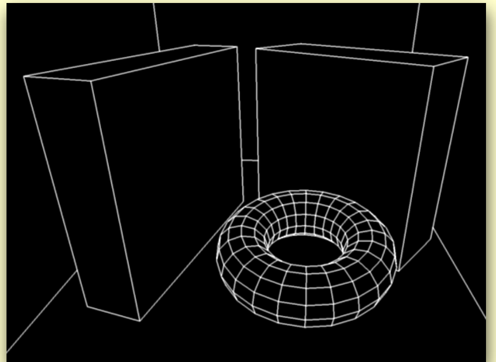


Simple Shadows



Global Illumination

3D Rendering Steps



Geometric Model

Perspective

Visibility

Local Illumination

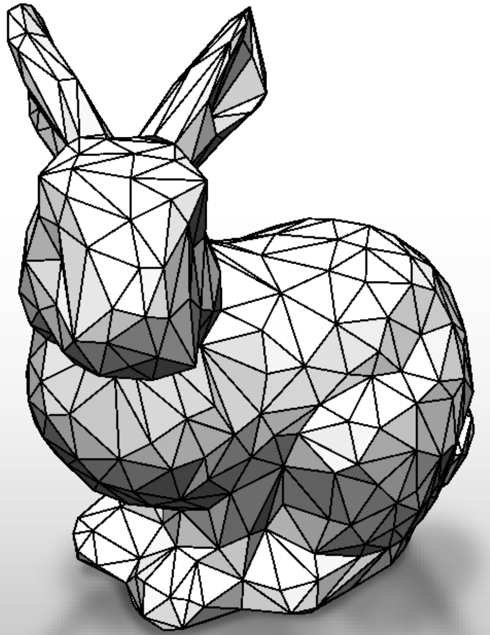
Smooth Shading

Simple Shadows

Global Illumination

Modeling

Mesh Representation



BASIC

basic topics

study completely

Modeling Shapes

Primitives

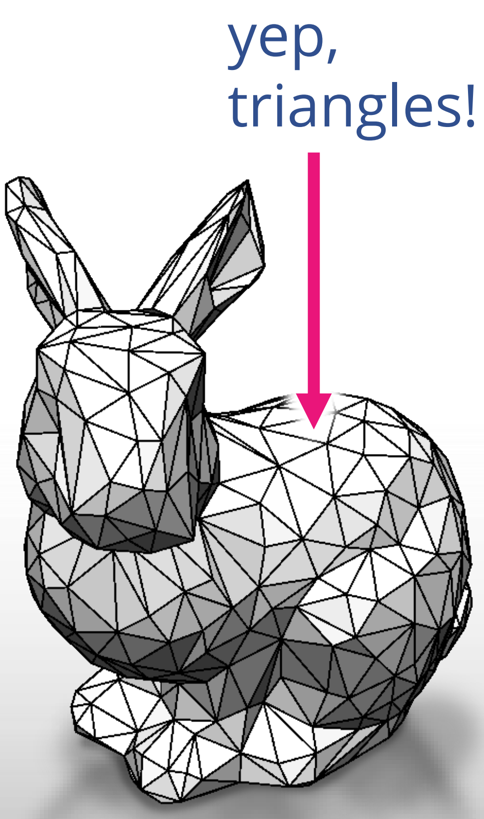
- Elementary geometric building blocks
- Easy to handle directly

Complex models

- Sets of primitives
- Approximate shapes with primitives

Most-frequently-used

- Triangles!



Simple Triangle List

Vertex list

Vector3D vertices[n];

(1) $\mathbf{p}_1 = (x_1, y_1, z_1)$

(2) $\mathbf{p}_2 = (x_2, y_2, z_2)$

(3) $\mathbf{p}_3 = (x_3, y_3, z_3)$

(4) $\mathbf{p}_4 = (x_4, y_4, z_4)$

$\vdots$

(*n*) $\mathbf{p}_\mathrm{n} = (x_n, y_n, z_n)$

Triangle list

(int[3]) triangles[m];

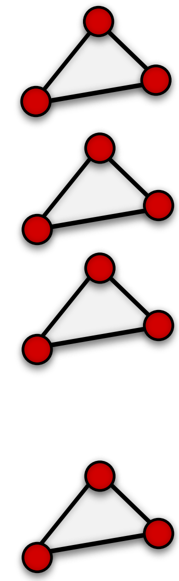
(1) $\mathbf{t}_1 = (i_1, j_1, k_1)$

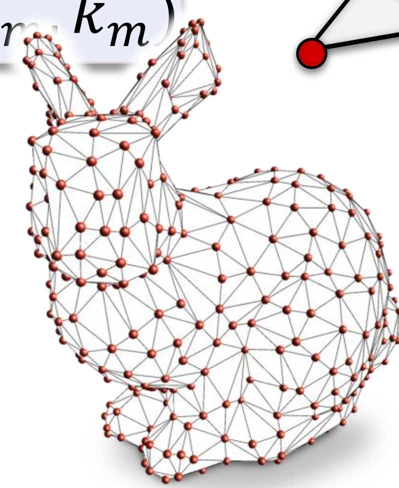
(2) $\mathbf{t}_2 = (i_2, j_2, k_2)$

(3) $\mathbf{t}_3 = (i_3, j_3, k_3)$

$\vdots$

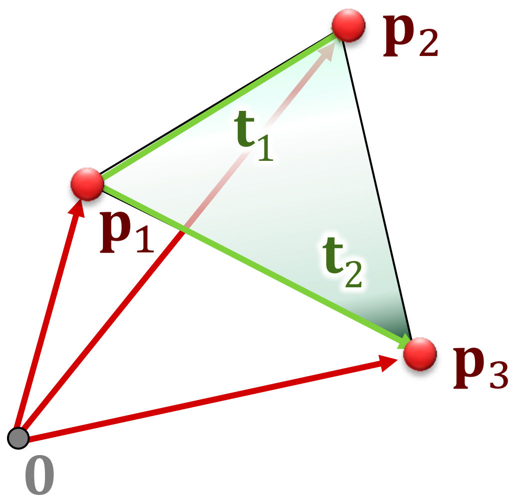
(*m*) $\mathbf{t}_\mathrm{m} = (i_m, j_m, k_m)$





Modeling a Triangle

Triangle



Parametric Plane Equation

(with constraints)

Triangles:

$$\mathbf{x}(\lambda, \mu) = \mathbf{p}_1 + \lambda \mathbf{t}_1 + \mu \mathbf{t}_2$$
$$= \mathbf{p}_1 + \lambda(\mathbf{p}_2 - \mathbf{p}_1) + \mu(\mathbf{p}_3 - \mathbf{p}_1)$$

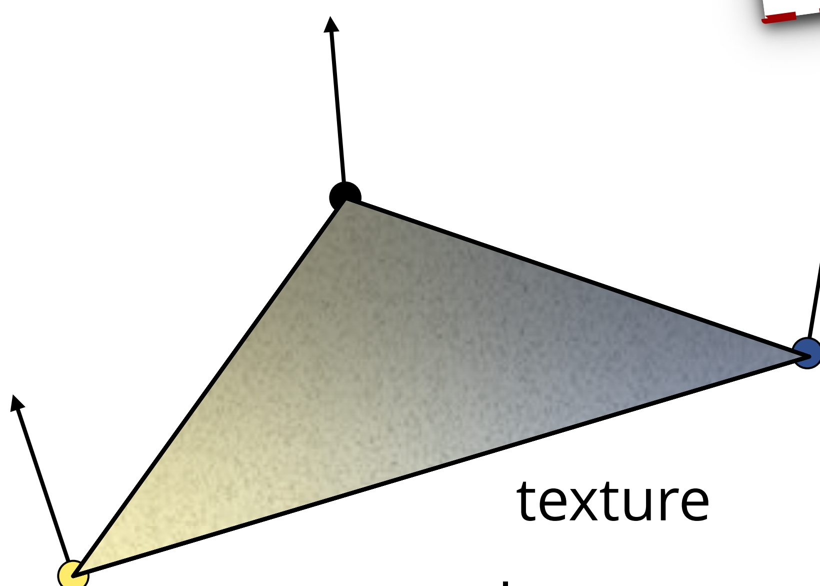
$$0 \leq \lambda \leq 1,$$
$$0 \leq \mu \leq 1,$$
$$\mu + \lambda \leq 1$$

Attributes

How to define a triangle?

- We need three points in $\mathbb{R}^3$
- But we can have more:

More on this:
Later

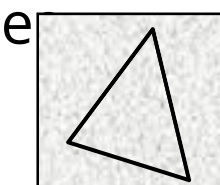


per-vertex normal

texture

per-vertex color

per-vertex texture coordinate



(etc...)

Complete Data Structures

Multiple Arrays: Vertices, Triangles, Edges

$\mathtt{v}_1$: (posx posy posz), $\mathtt{attrib}_1$, ..., $\mathtt{attrib}_n$

...

$\mathtt{v}_N$: (posx posy posz), $\mathtt{attrib}_1$, ..., $\mathtt{attrib}_n$

$\mathtt{e}_1$: ($\mathtt{index}_1$ $\mathtt{index}_2$), $\mathtt{attrib}_1$, ..., $\mathtt{attrib}_k$

...

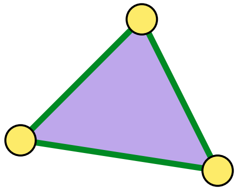
$\mathtt{e}_K$: ($\mathtt{index}_1$ $\mathtt{index}_2$), $\mathtt{attrib}_1$, ..., $\mathtt{attrib}_k$

edges: optional

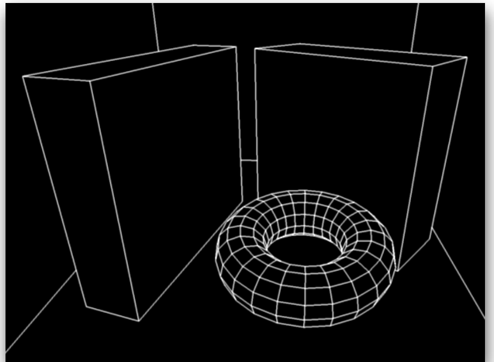
$\mathtt{t}_1$: ($\mathtt{idx}_1$ $\mathtt{idx}_2$ $\mathtt{idx}_3$), $\mathtt{attrib}_1$, ..., $\mathtt{attrib}_m$

...

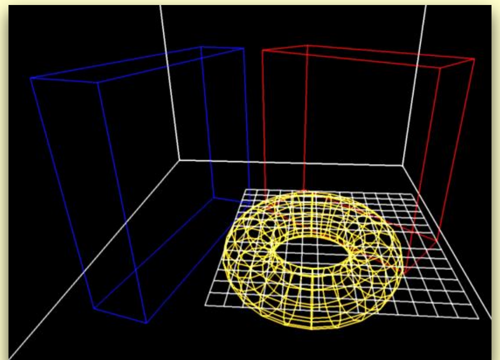
$\mathtt{t}_M$: ($\mathtt{idx}_1$ $\mathtt{idx}_2$ $\mathtt{idx}_3$), $\mathtt{attrib}_1$, ..., $\mathtt{attrib}_m$



3D Rendering Steps



Geometric Model



Perspective

Visibility

Local Illumination

Smooth Shading

Simple Shadows

Global Illumination

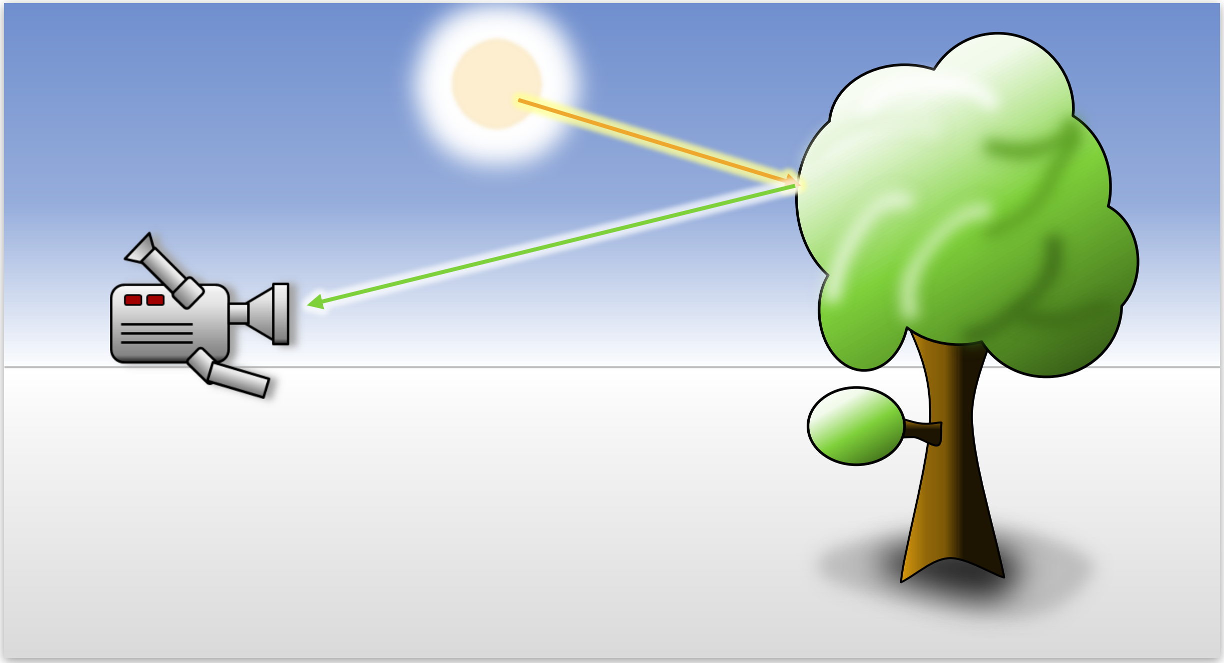
Physics

Ray Optics & Color

CORE

core topics
important

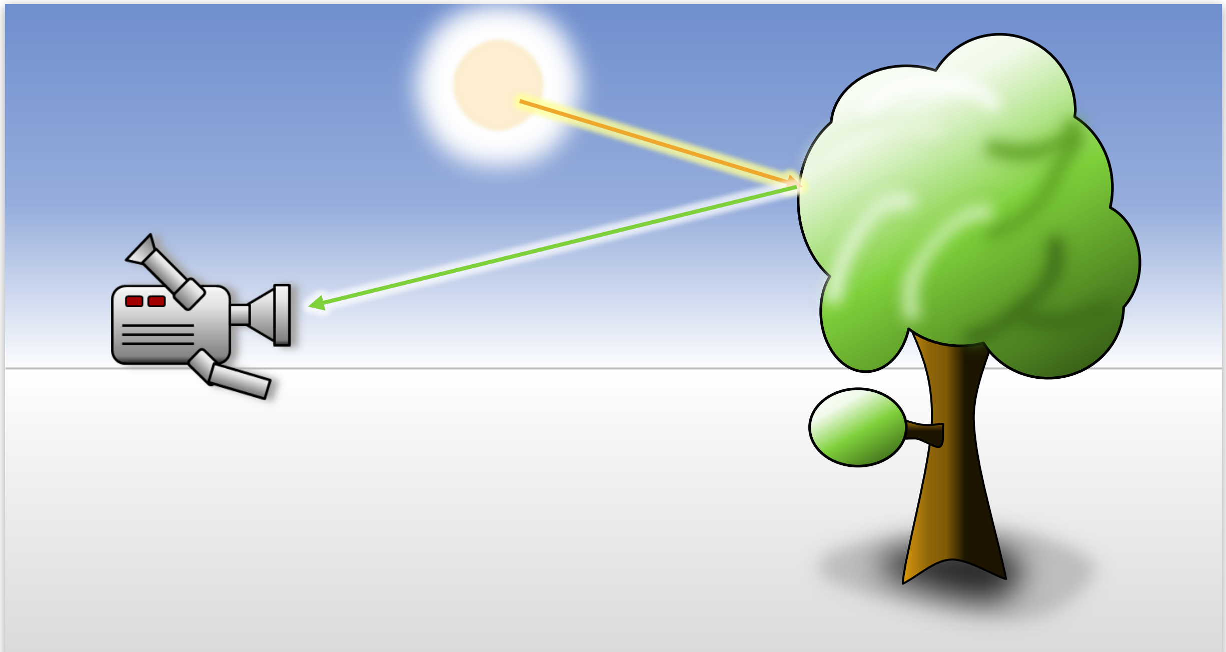
Ray Optics



Geometric ray model

- Light travels along rays

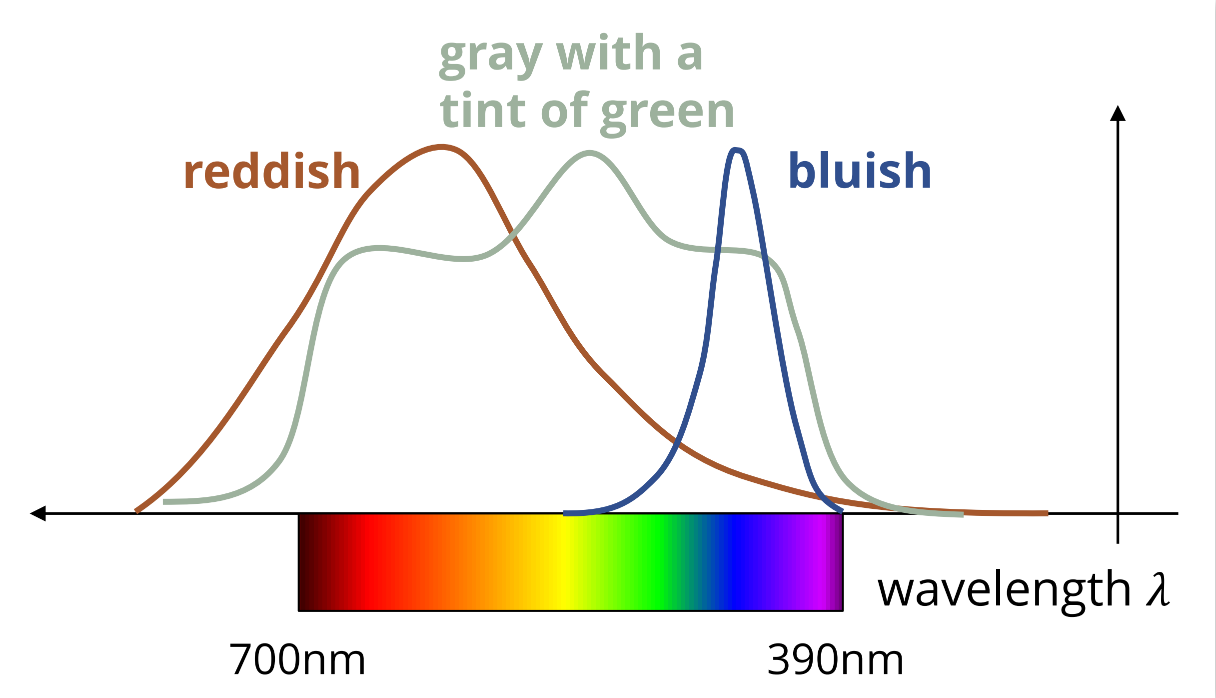
Ray Optics



Geometric ray model

- Rays have “intensity” and “color”

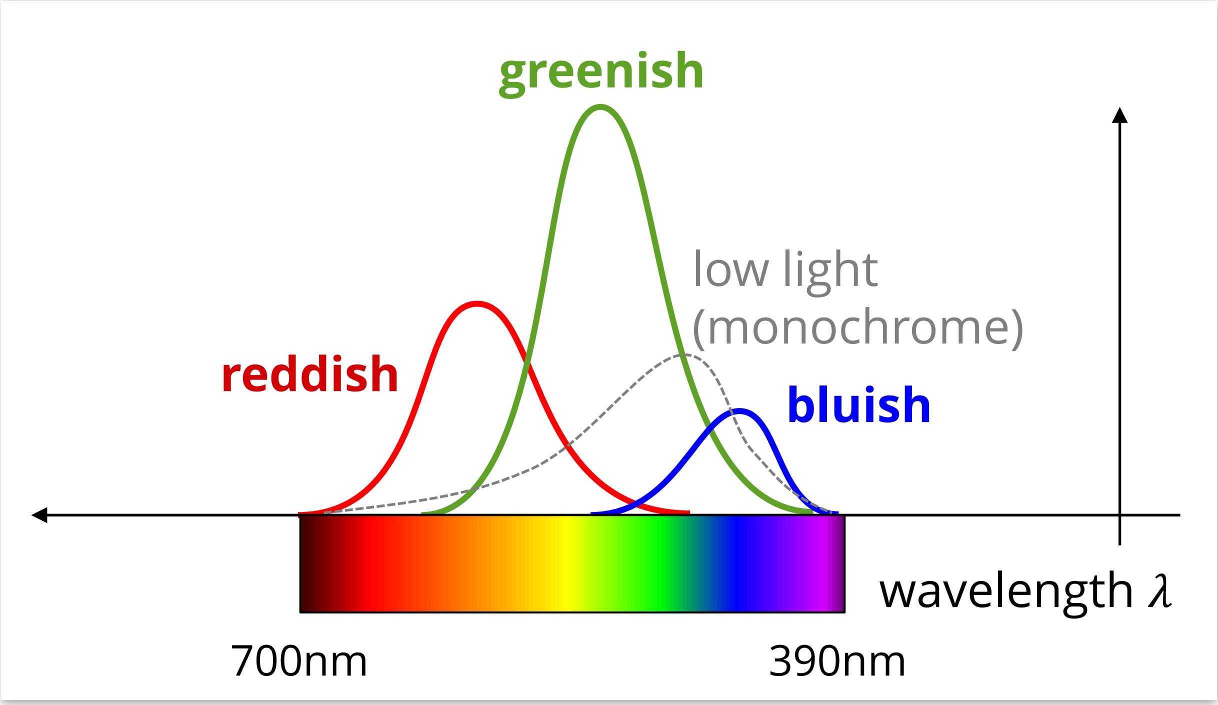
Ray Optics



Color spectrum

- Continuous spectrum
- Intensity for each wavelength

Human Vision



(curves: schematic, not accurate)

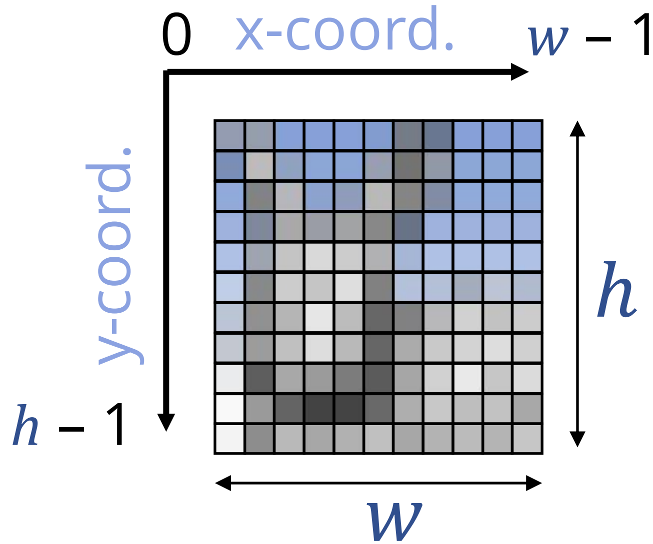
Color spectrum

- Two types of receptive cells (color/low-light)
- Three types of color cells

RGB Model

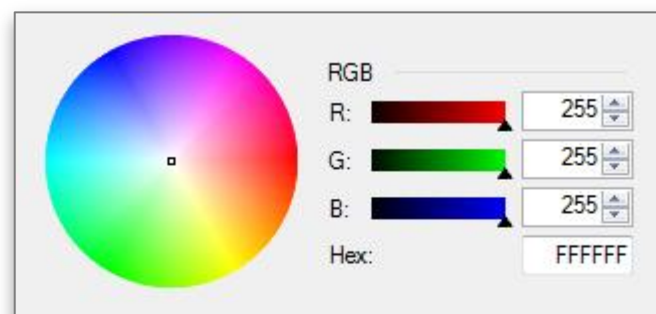
Bitmap (Pixel Display)

- Screen: $w \cdot h$ discrete pixels
  - Origin: usually upper left
- Varying color per pixel

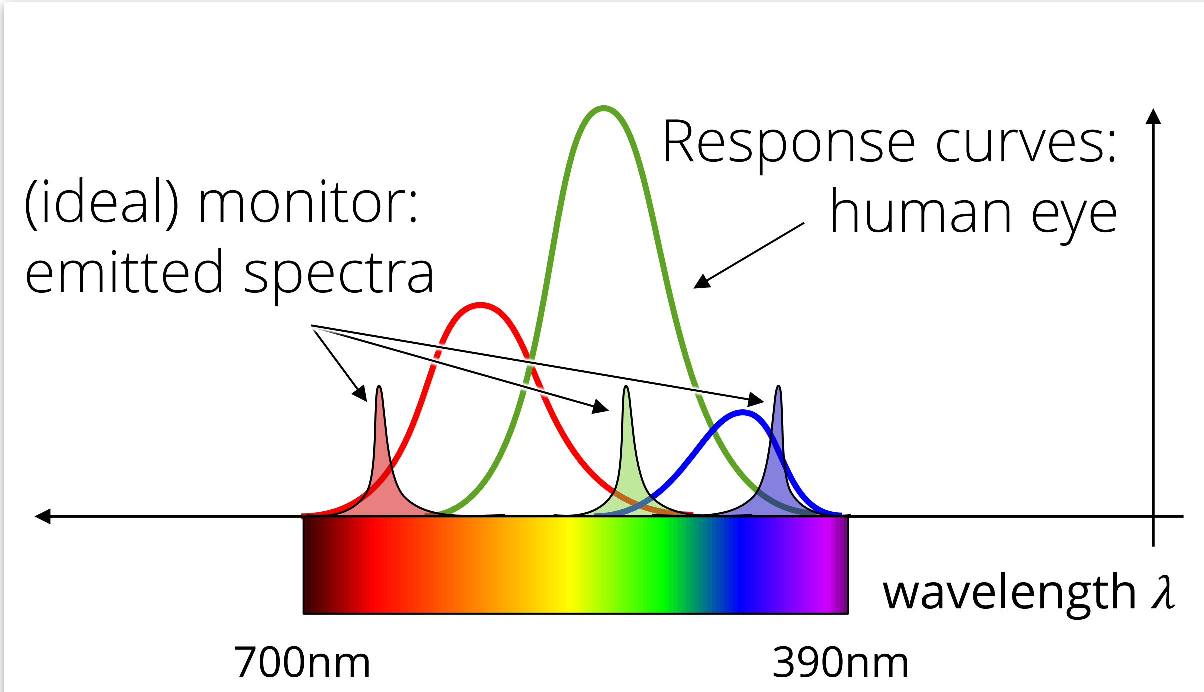


RGB Model

- Every pixel can emit *red*, *green*, *blue* light
- Intensity range:
  - Usually: bytes 0...255
  - 0 = dark
  - 255 = maximum brightness



Human Vision



(curves: schematic, not accurate)

Create color impressions

- Basis for three-dimensional color space
- Wide spacing, narrow bands: purer colors
  - Otherwise: washed out colors

Physics

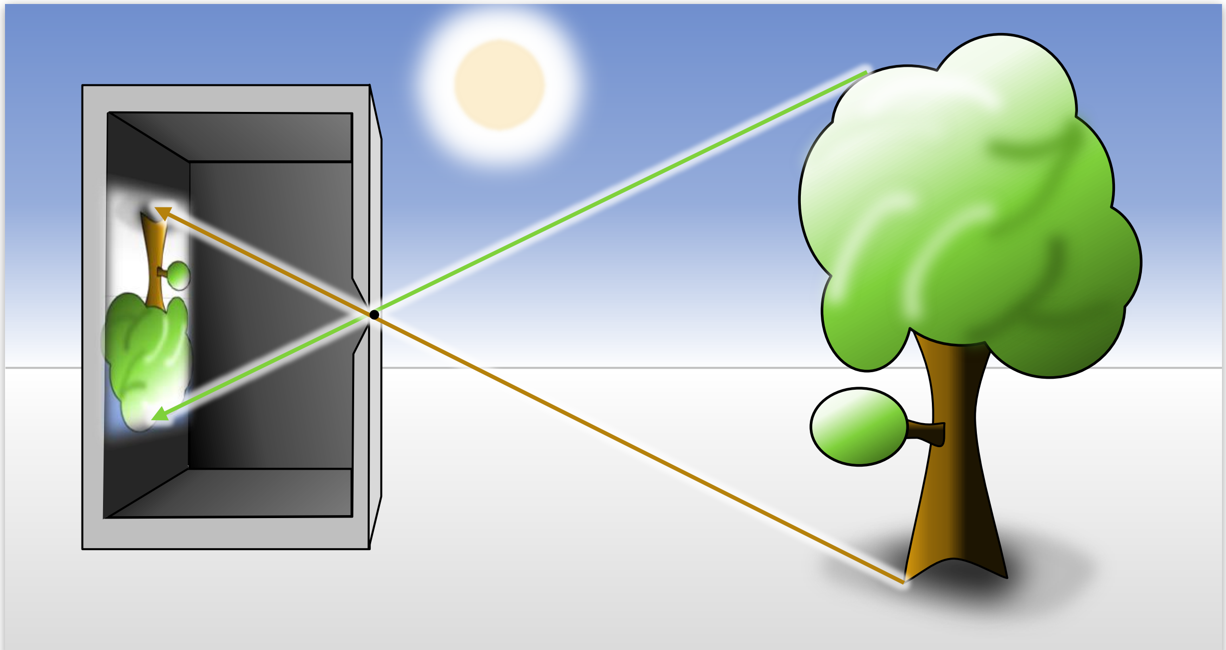
Perspective Projection

BASIC

basic topics

study completely

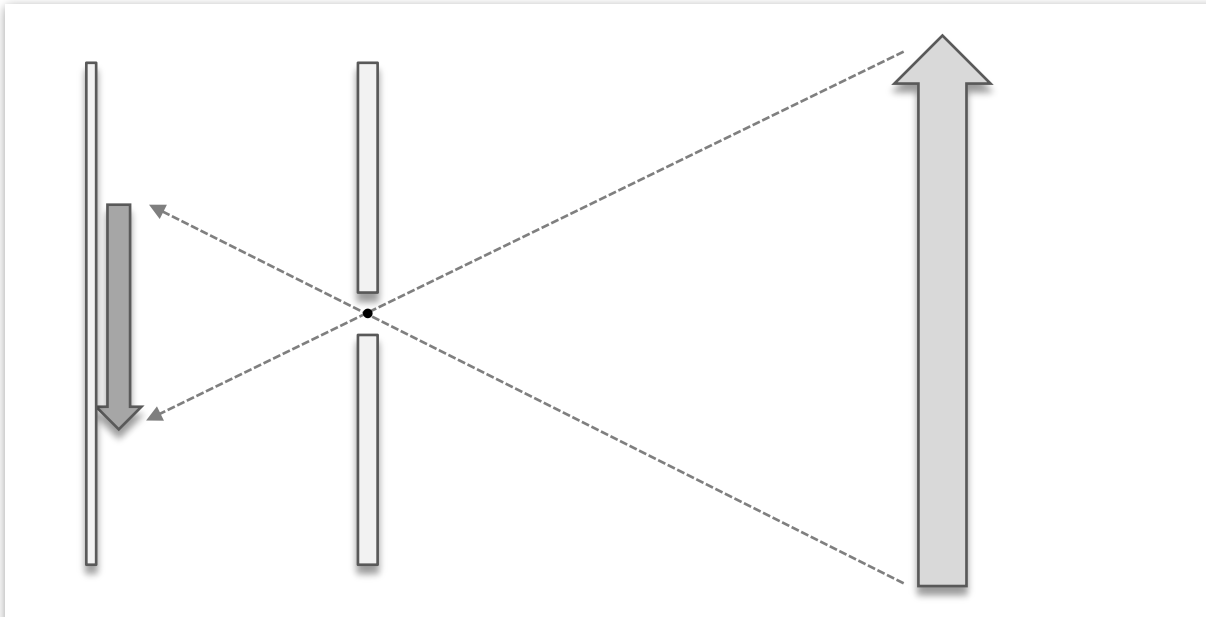
Pinhole Camera



Pinhole camera

- Create image by selecting rays of specific angles
- Low efficiency (small holes for sharp images)

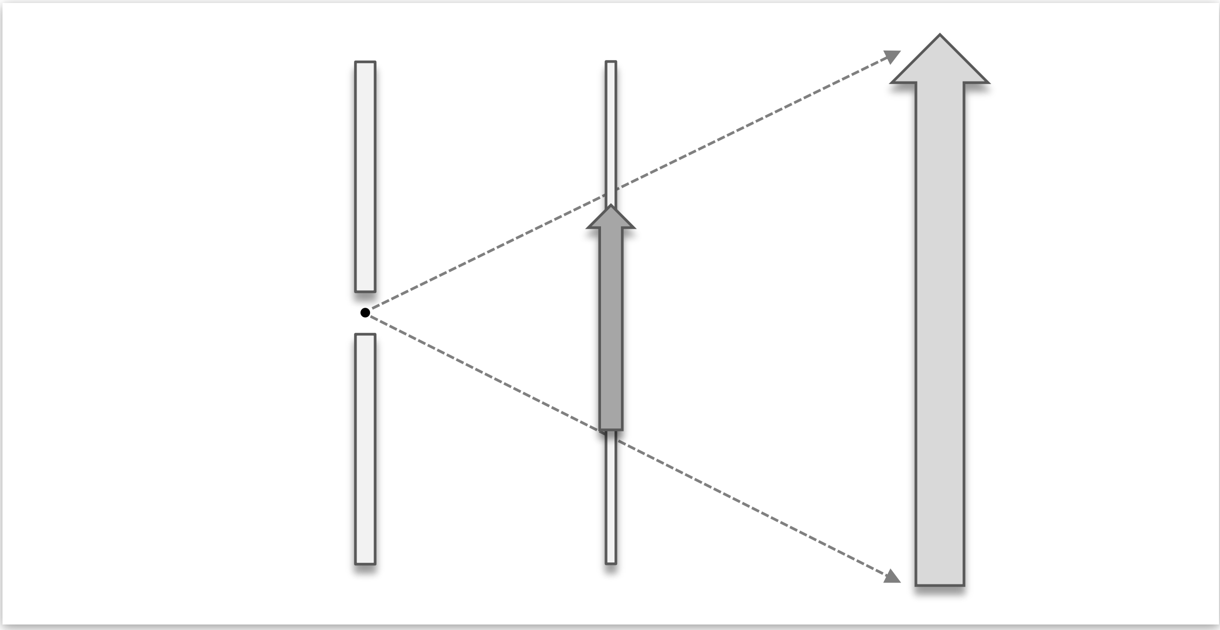
Pinhole Camera



Pinhole camera

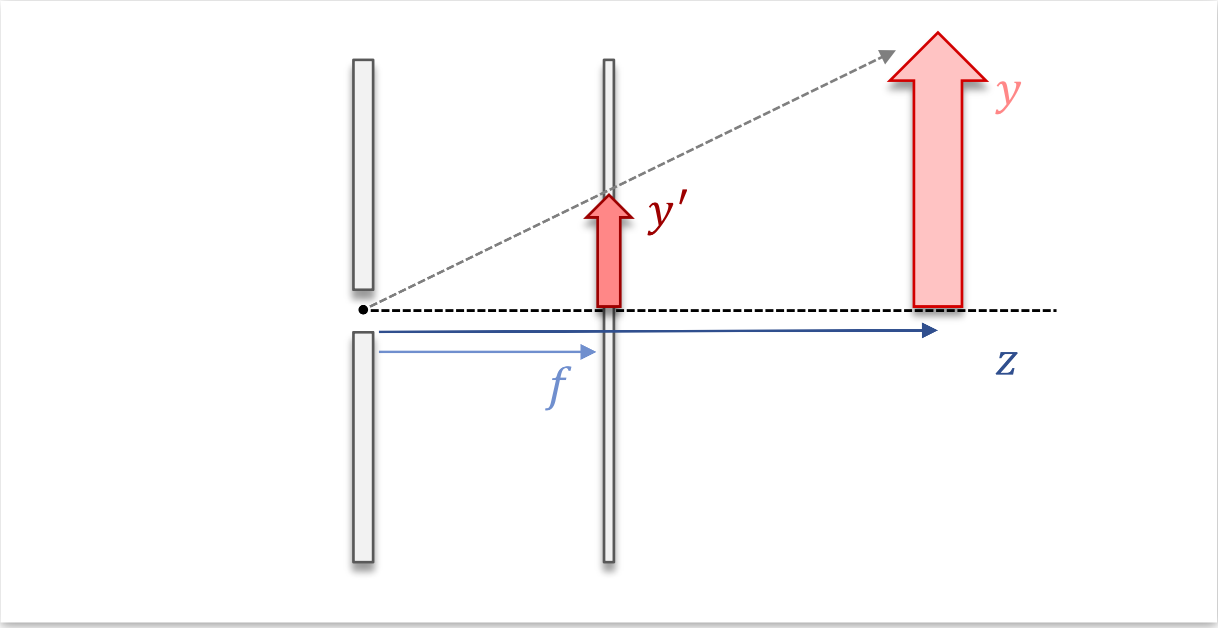
- Create image by selecting rays of specific angles
- Low efficiency (small holes for sharp images)

Pinhole Camera



Central Projection

Pinhole Camera

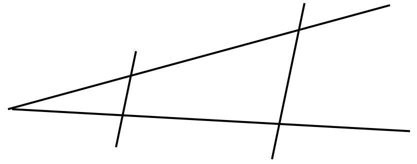


Central projection

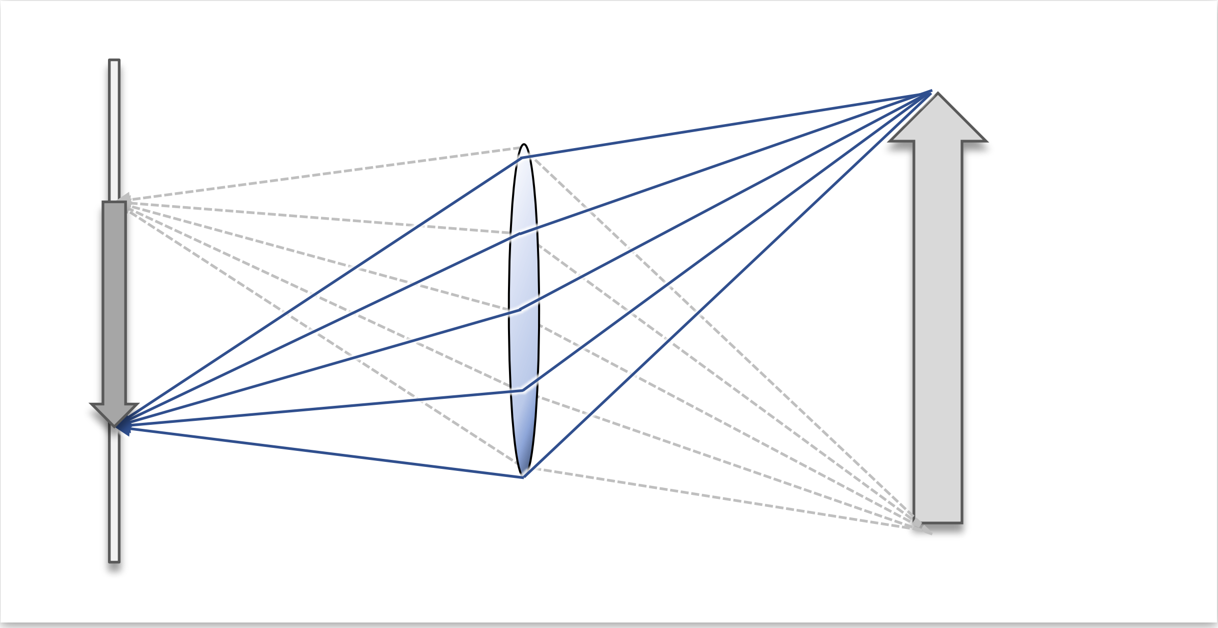
$$x' = f\frac{x}{z}$$

$$y' = f\frac{y}{z}$$

Proof:
Intercept theorem!



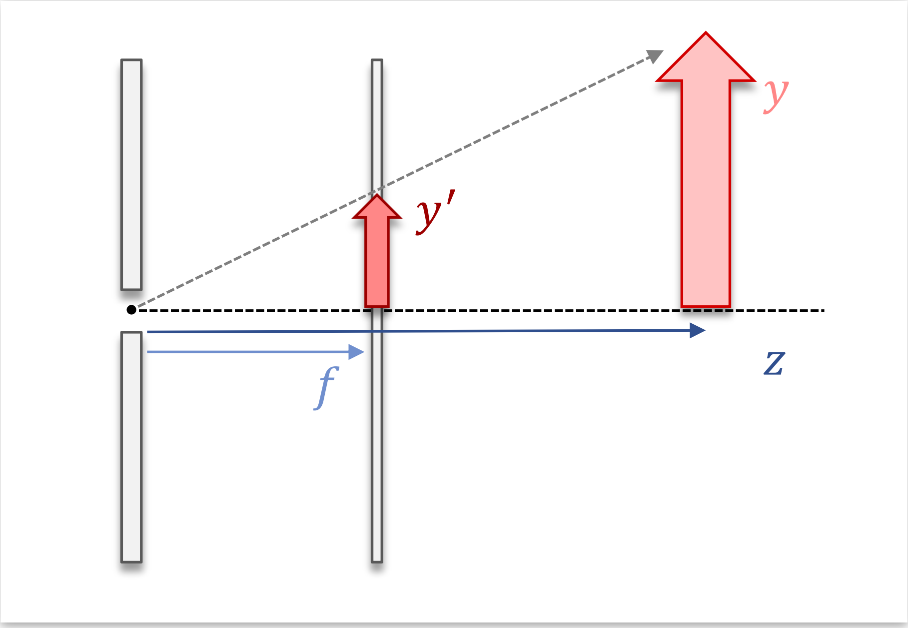
(Actual Camera)



Camera with Lens

- Higher efficiency (bundles many rays)
- Finite Depth of field
- We will consider pinhole cameras only.

Pinhole Camera



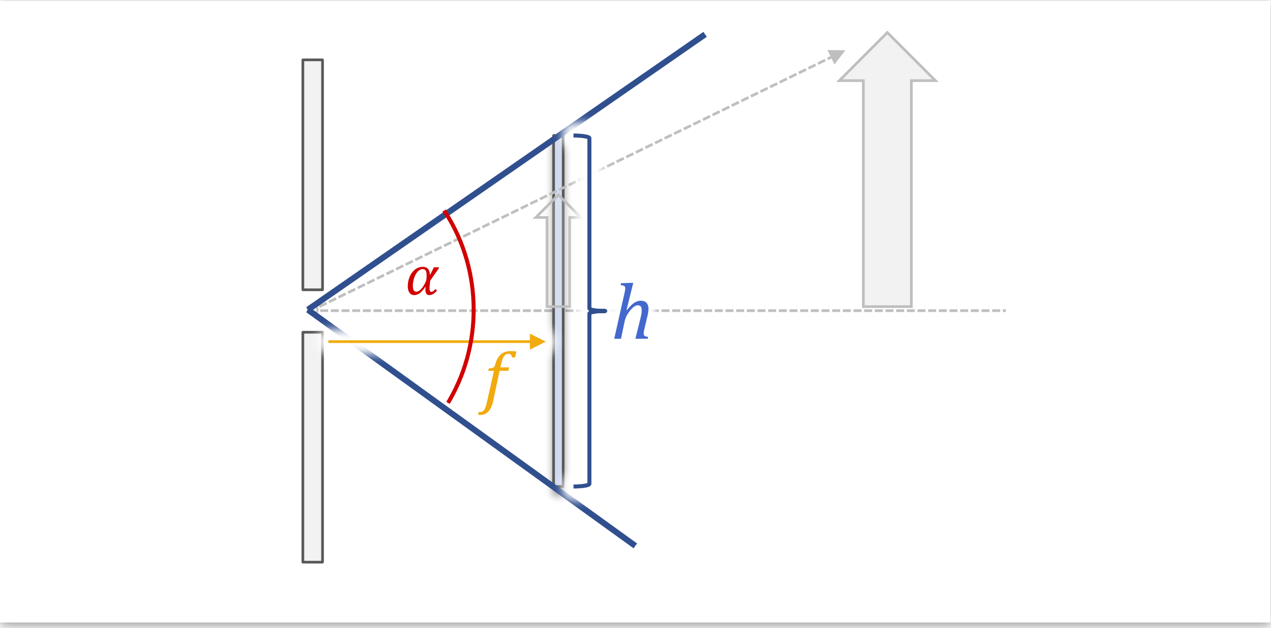
$$x' = f\,\frac{x}{z}$$

$$y' = f\,\frac{y}{z}$$

Undetermined degree of freedom

- Focal length vs. image size
- Source of a lot of confusion!

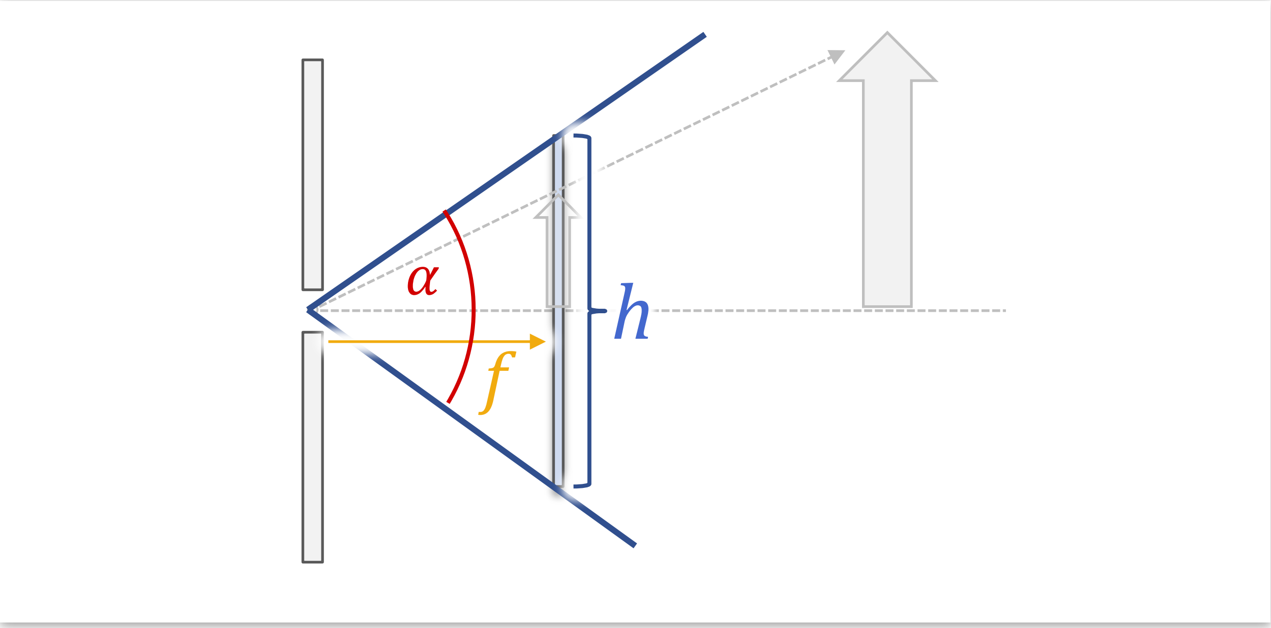
Pinhole Camera



Parameters

- h - size of the screen (pixels, cm, ± 1.0,...)
- f – focal length (classical photography)
- Meaningful parameter: α – viewing angle

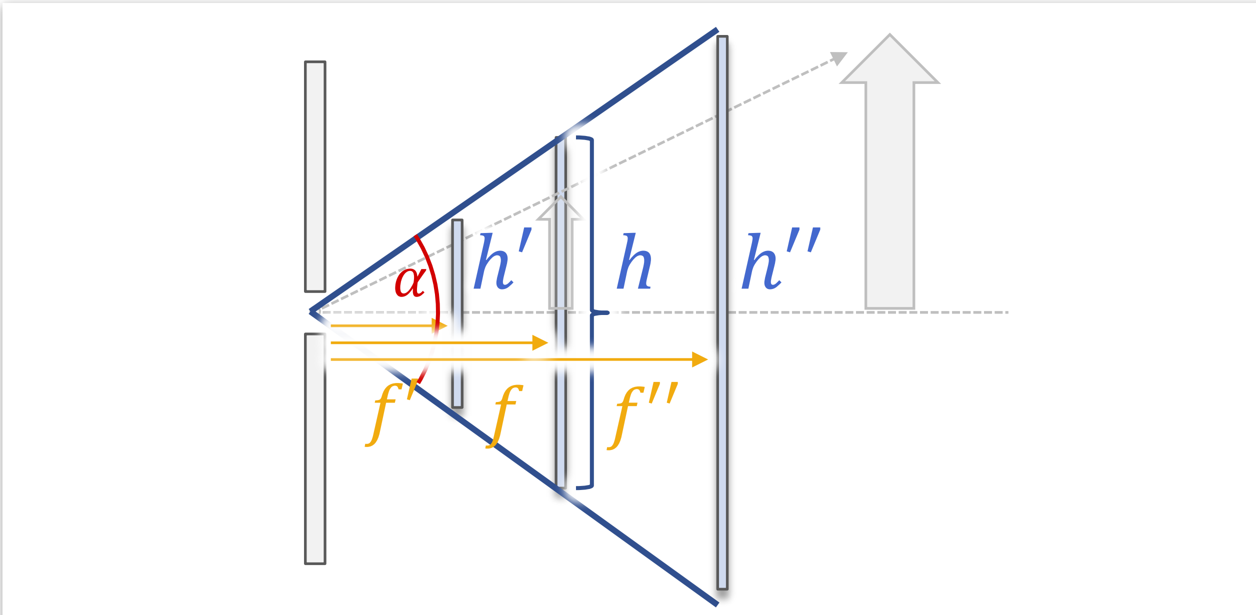
Pinhole Camera



Relation:

$$\tan\frac{\alpha}{2} = \frac{h}{2f}$$

Pinhole Camera

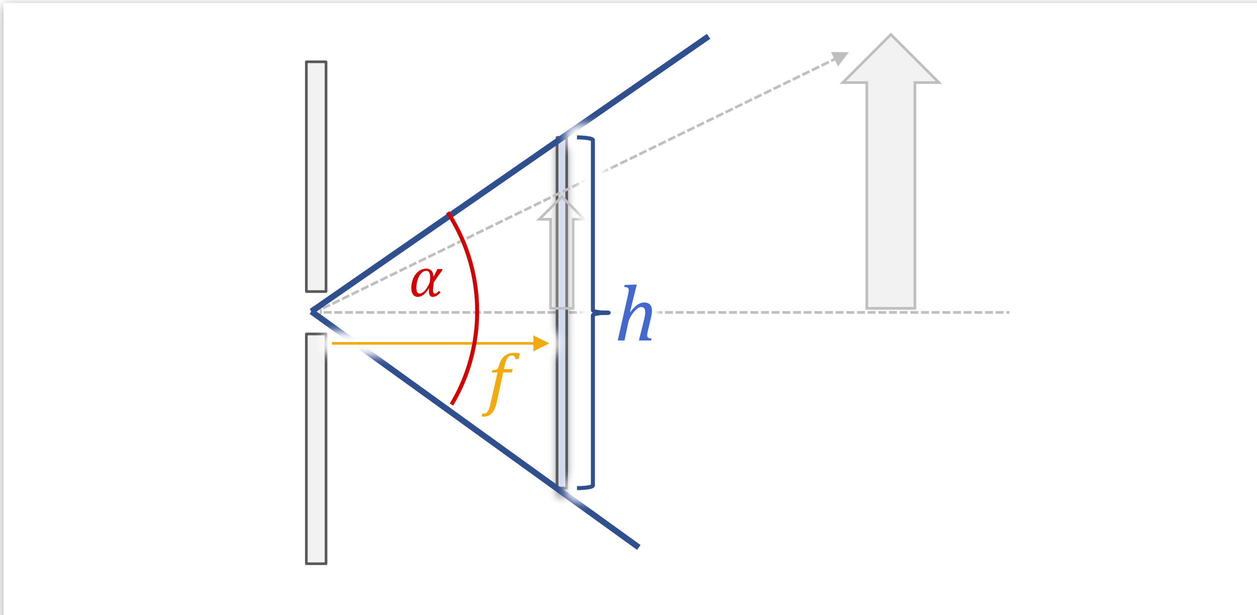


Invariance

$$\tan\frac{\alpha}{2} = \frac{h}{2f} = \frac{h'}{2f'} = \frac{h''}{2f''}$$

- Scaling *h* and *f* by a common factor: *no change*

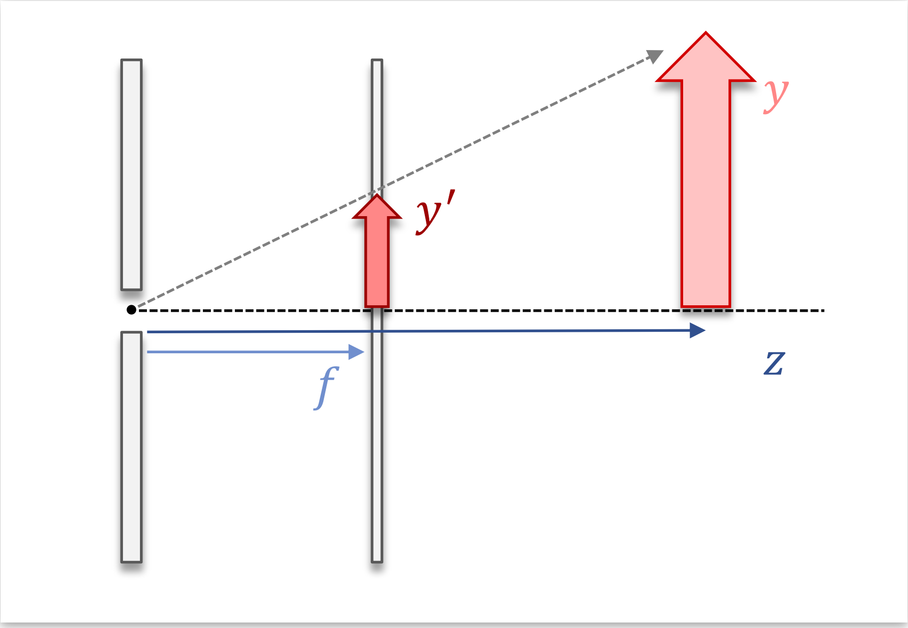
Pinhole Camera



Typical choices (vertical angles)

- "Normal" perspective: $\alpha \approx 30°$ ("50mm" lens: 27°)
- Tele photography: $\alpha \approx 5° - 20°$ (275–70mm)
- Wide angle lens: $\alpha \approx 45° - 90°$ (28–12mm)

General Camera



$$x' = f\,\frac{x}{z}$$

$$y' = f\,\frac{y}{z}$$

Our camera so far:

- Focus point: origin
- View direction: z-axis
- General position/orientation?

Homogeneous Coordinates

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \underbrace{\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{Projection Matrix } \mathbf{P}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$x' = f\frac{x}{z}$$

$$y' = f\frac{y}{z}$$

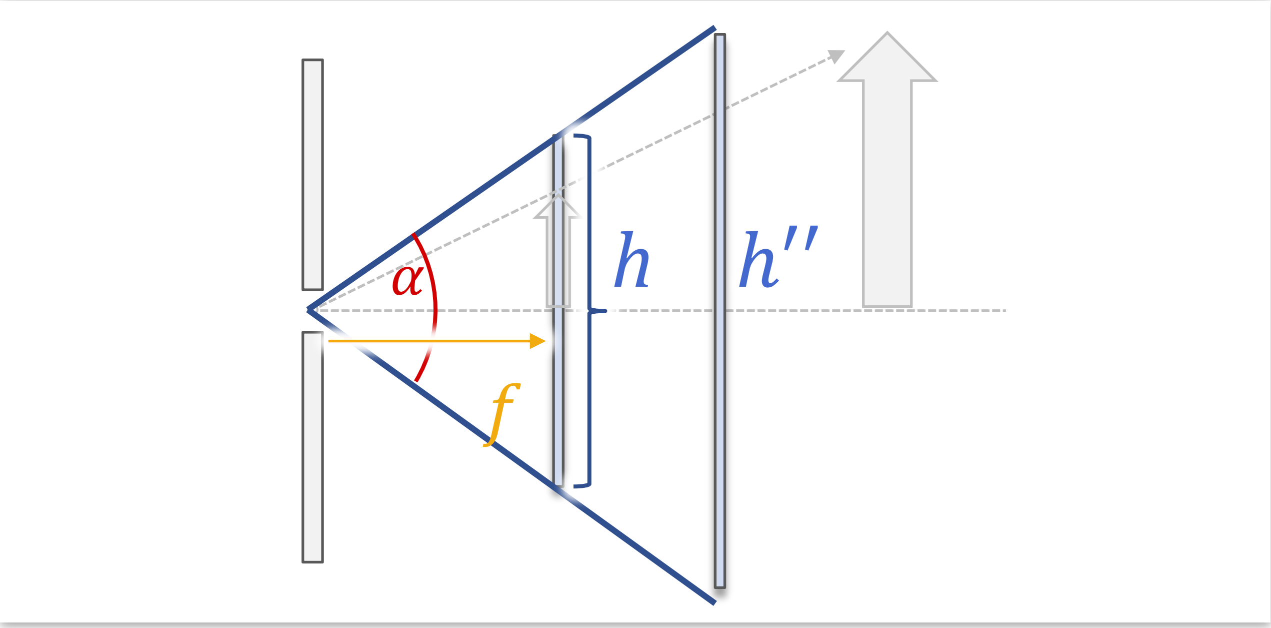
$$z' = \frac{z-1}{z}$$

$$w' = z$$

Write in homogeneous coordinates

- Third row is arbitrary (for now), not used.

View transform

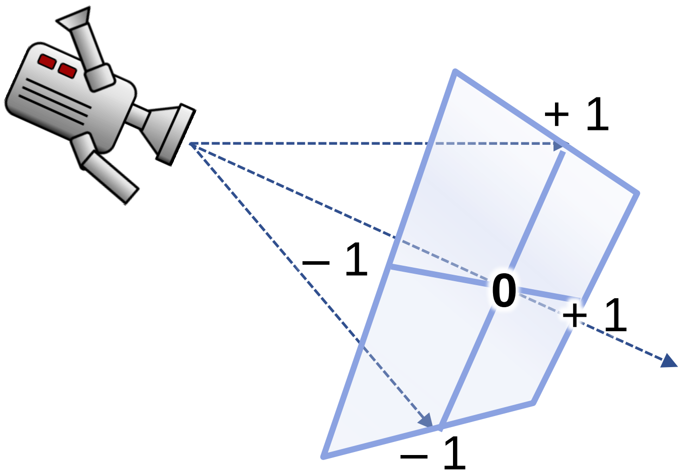


Reminder:

$$\tan\frac{\alpha}{2} = \frac{h}{2f}$$

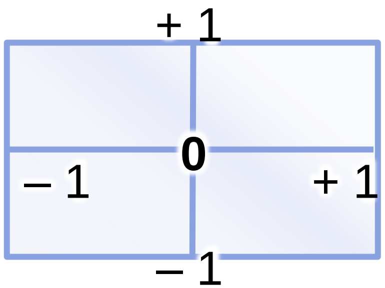
To Screen Coordinates

$$\begin{pmatrix} \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Scale to unit screen coordinates

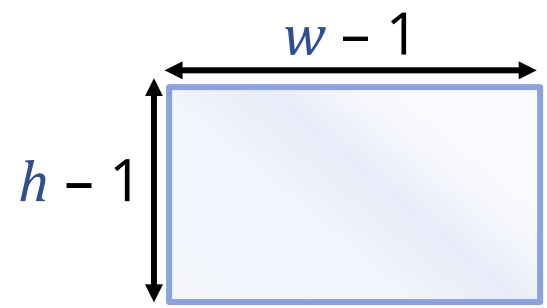
- We set f to 1 in previous matrix
- Third row is arbitrary (for now), not used.



normalized screen coordinates

Aspect Ratio

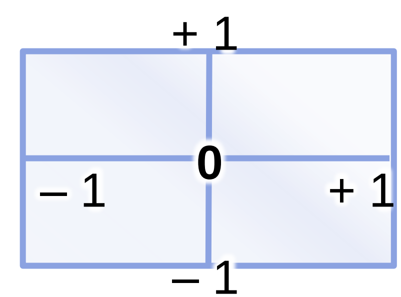
$$\begin{pmatrix} \dfrac{1}{\frac{w}{h}\cdot\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \dfrac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



non-square screen

Non-square screens?

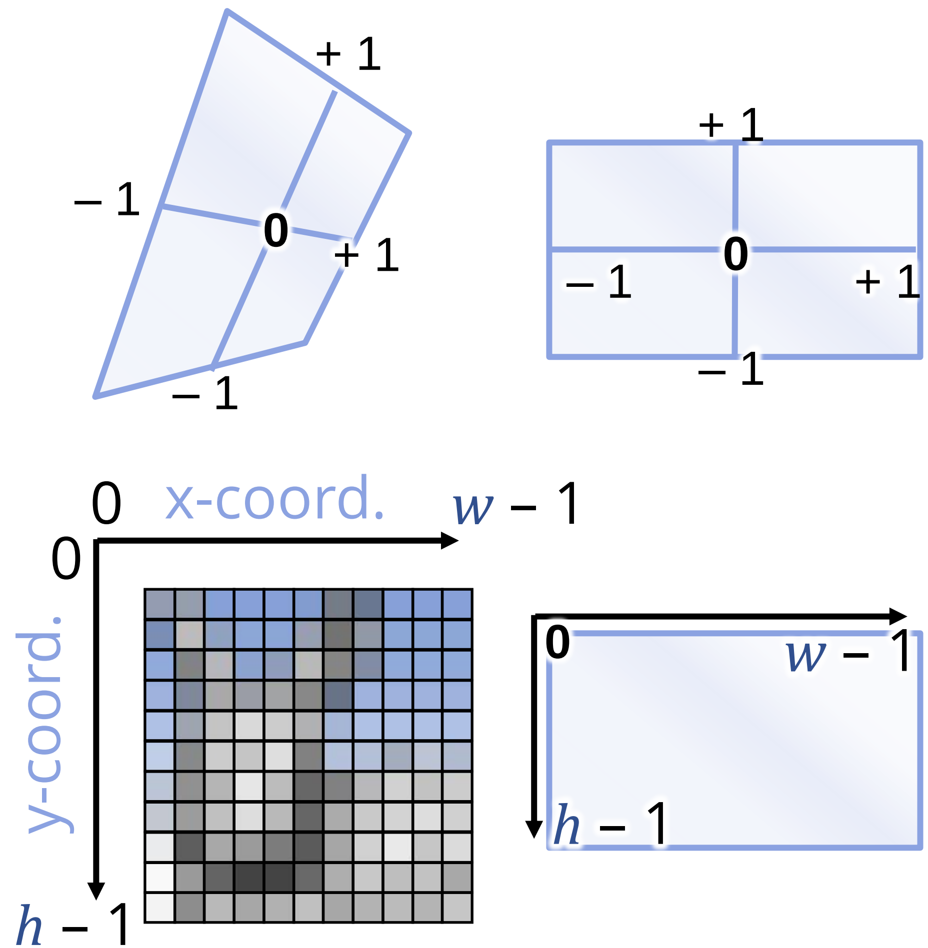
- Screen: $w \times h$ pixels
- Aspect ratio $\frac{w}{h}$
- Different horizontal angle!



normalized screen coordinates

To Screen Coordinates

$$\begin{pmatrix} w/2 & 0 & 0 & w/2 \\ 0 & -h/2 & 0 & h/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Scale to pixels

- Third row is arbitrary (for now), not used.

To Screen Coordinates

$$\begin{pmatrix} \frac{h/2}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & \frac{w/2}{\tan\left(\frac{\alpha}{2}\right)} \\ 0 & -\frac{h/2}{\tan\left(\frac{\alpha}{2}\right)} & 0 & \frac{h/2}{\tan\left(\frac{\alpha}{2}\right)} \\ 0 & 0 & a & b \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Overall

- Multiple both

$$a = \frac{z_{far} + z_{near}}{z_{near} - z_{far}}$$

$$b = \frac{2 \cdot z_{near} \cdot z_{far}}{z_{near} - z_{far}}$$

Additionally:
Also scale + shift such that

$$z' = \frac{z - 1}{z}$$

are in value [0..1] for inputs
$z \in [z_{near}, z_{far}]$

Summary

Projection matrix

$$\mathbf{P} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Projection & conversion to screen coords

$$\mathbf{P}_s = \underbrace{\begin{pmatrix} w/2 & 0 & 0 & w/2 \\ 0 & -h/2 & 0 & h/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{scaling to pixels, upper left origin}} \cdot \underbrace{\begin{pmatrix} \frac{1}{\frac{w}{h}\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{normalized screen coord's}} \cdot \underbrace{\overset{(f\,=\,1)}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}}}_{\text{projection matrix}}$$

Alternative (1)

Alternative formulation: Only two steps

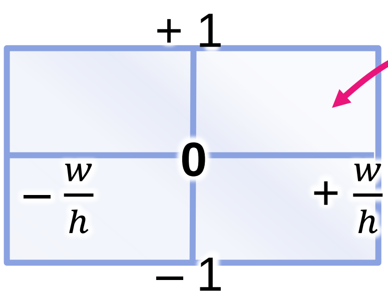
$$\mathbf{P}_s = \underbrace{\begin{pmatrix} w/2 & 0 & 0 & w/2 \\ 0 & -h/2 & 0 & h/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{scaling to pixels, upper left origin}} \cdot \underbrace{\begin{pmatrix} \frac{1}{\frac{w}{h}\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{normalized screen coord's \& projection matrix}}$$

- Different scale factors (not a focal length)
- Use two different scale factors $f_x = \frac{1}{\frac{w}{h}\tan\left(\frac{\alpha}{2}\right)}$, $f_y = \frac{1}{\tan\left(\frac{\alpha}{2}\right)}$

Alternative (2)

Another Alternative Formulation

$$\mathbf{P}_s = \underbrace{\begin{pmatrix} h/2 & 0 & 0 & w/2 \\ 0 & -h/2 & 0 & h/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{scaling to pixels, upper left origin}} \cdot \underbrace{\begin{pmatrix} \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{vertically normalized screen coord's}} \text{ \& projection matrix}$$



- Constant focal length $f = \frac{1}{\tan\left(\frac{\alpha}{2}\right)}$
- Intermediate result not normalized to $[-1,1]^2$

Alternatives

All three derivations lead to the *same result*

- Intermediate results not used ⇒ all equivalent
- Product of the 2/3 matrices is the same

Intermediate results being used:

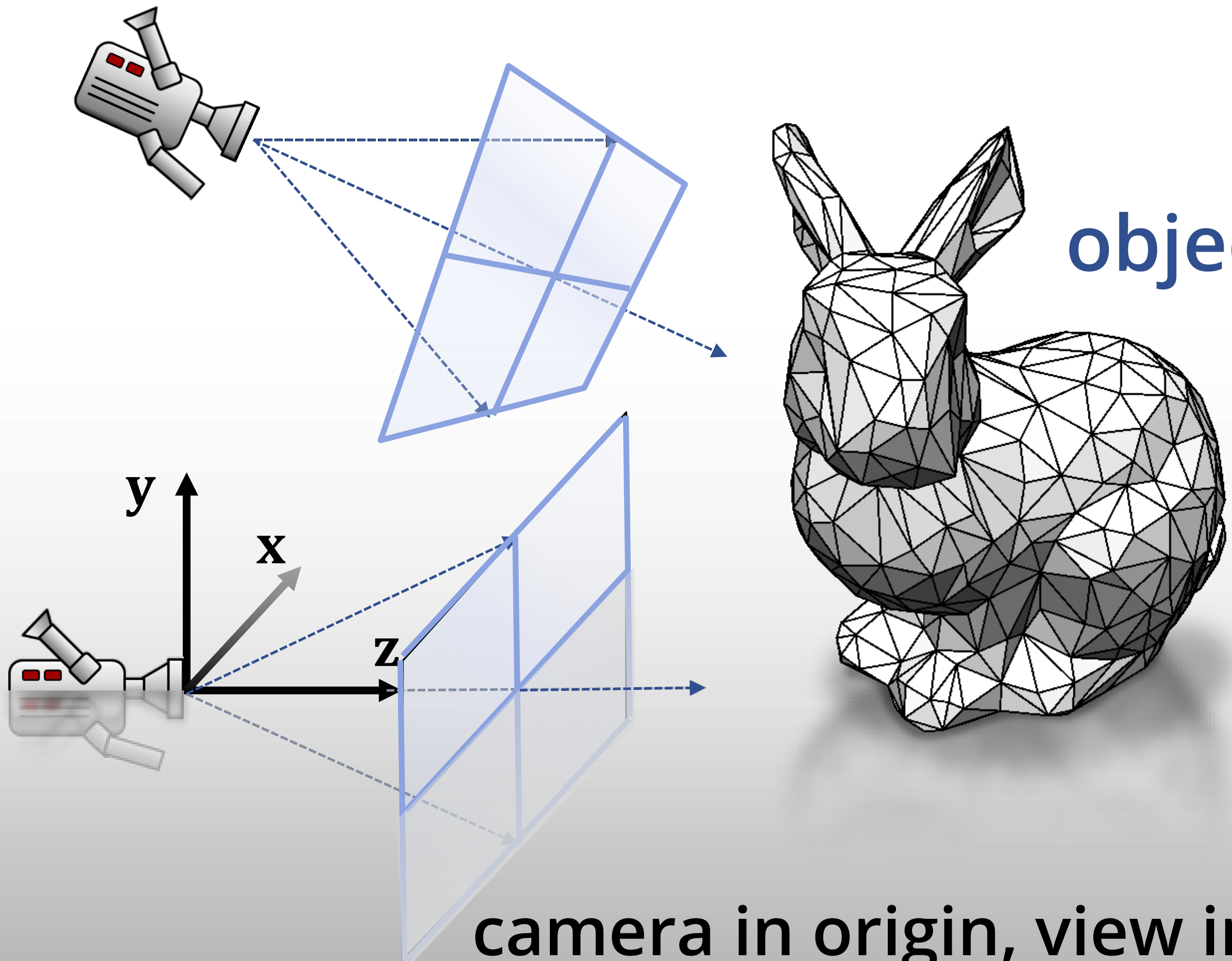
- Some graphics APIs (e.g., OpenGL) do use normalized device coordinates as *intermediate*
  - OpenGL – for pixels to appear on screen:

$$x' \in [-1,1]$$
$$y' \in [-1,1]$$
$$z' \in [0,1)$$
$$w \in \left[z_{near}, z_{far}\right)$$

coupled, so this
is the same criterion

General Camera

general camera

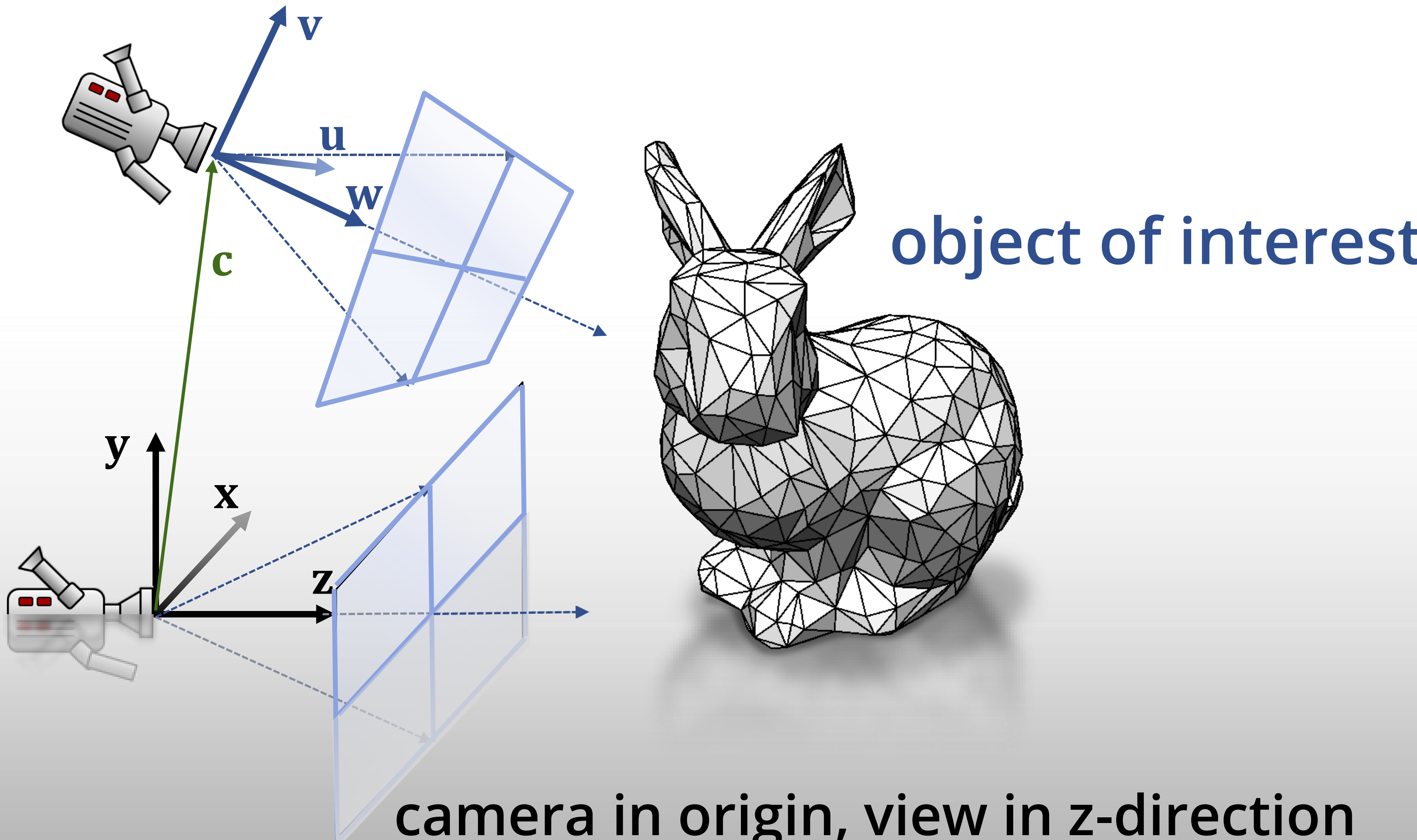


object of interest

camera in origin, view in z-direction

General Camera

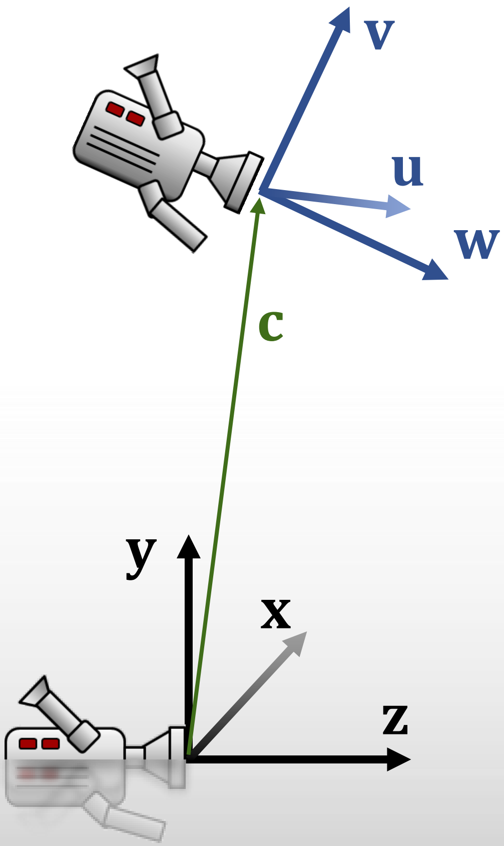
general camera



camera in origin, view in z-direction

General Camera

general camera



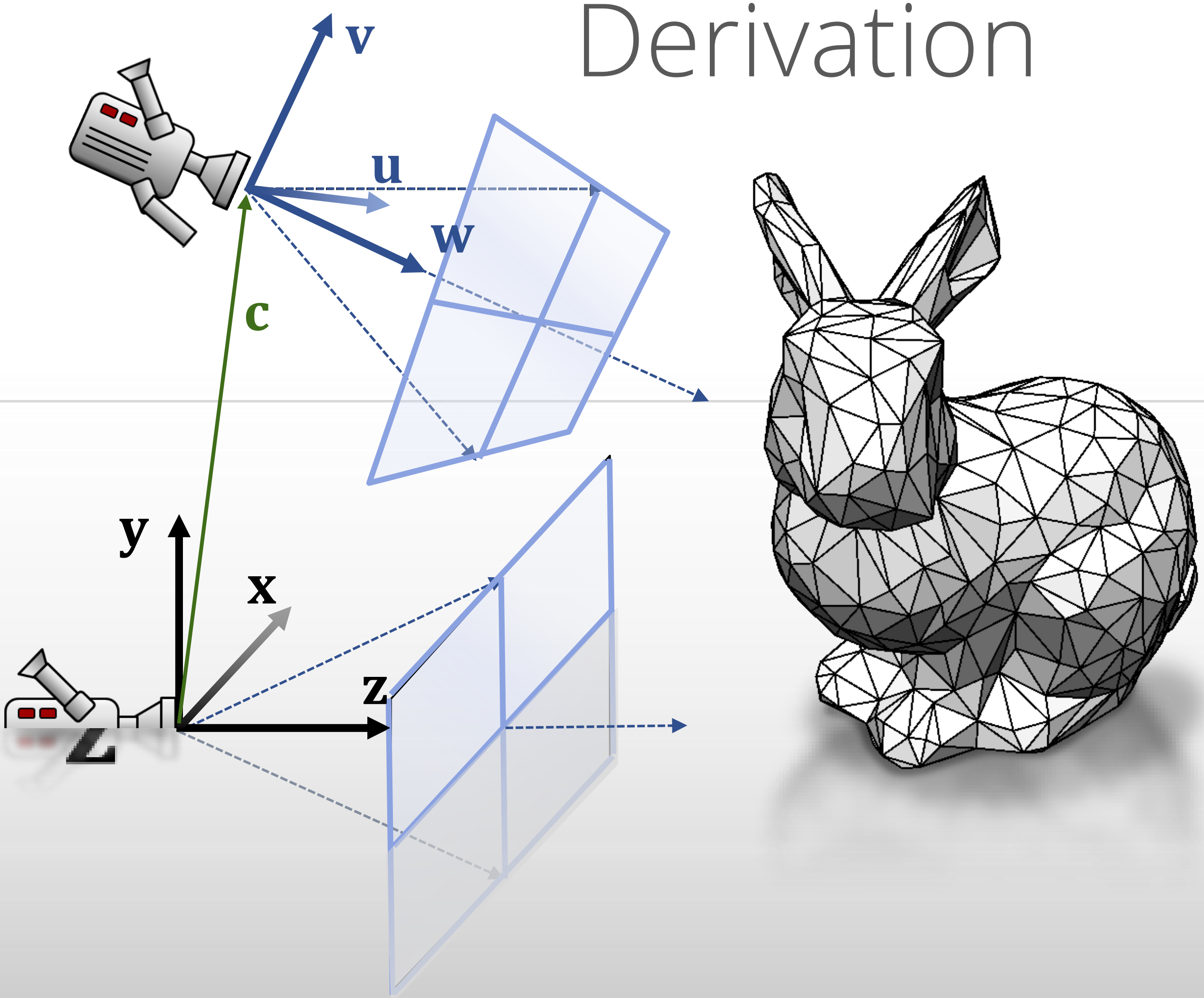
Camera coordinate system $(\mathbf{u}, \mathbf{v}, \mathbf{w})$

Origin: $\mathbf{c}$

Standard coordinates $(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right)$

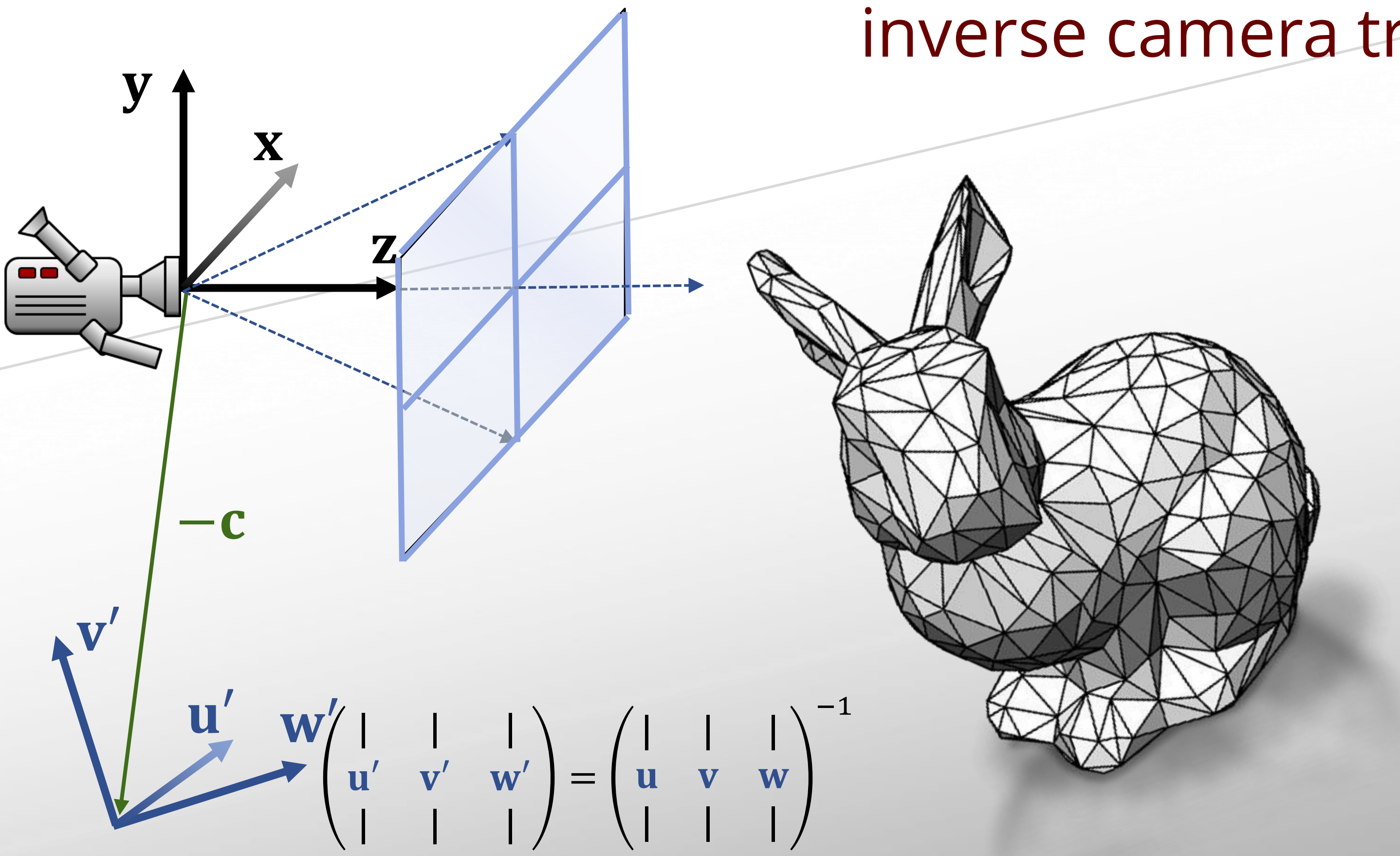
camera in origin, view: z-direction

Derivation



Derivation

Same effect:
Transform the world with inverse camera transform

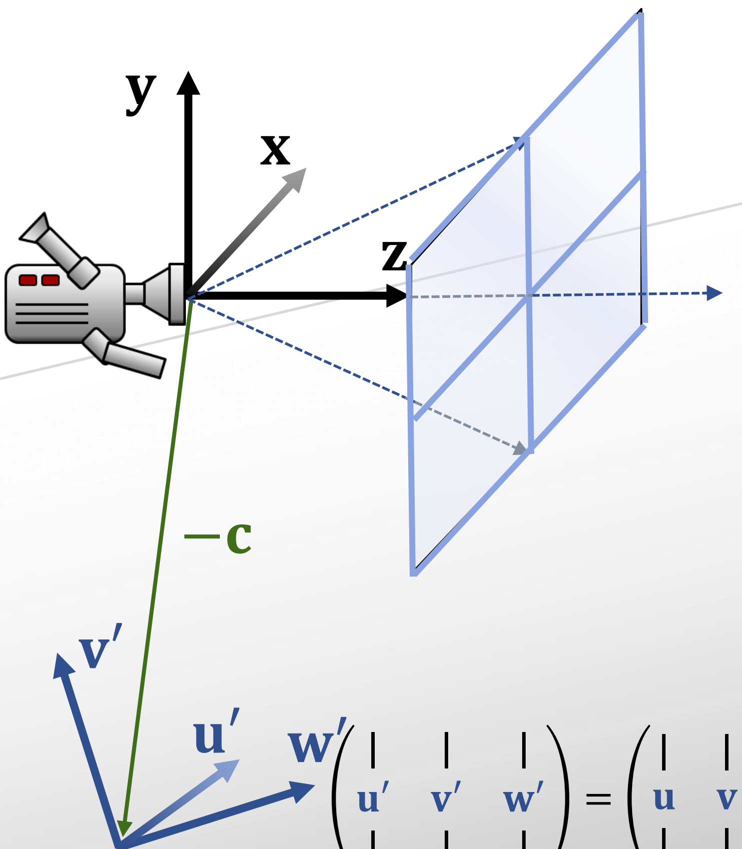


$$\begin{pmatrix} | & | & | \\ \mathbf{u}' & \mathbf{v}' & \mathbf{w}' \\ | & | & | \end{pmatrix} = \begin{pmatrix} | & | & | \\ \mathbf{u} & \mathbf{v} & \mathbf{w} \\ | & | & | \end{pmatrix}^{-1}$$

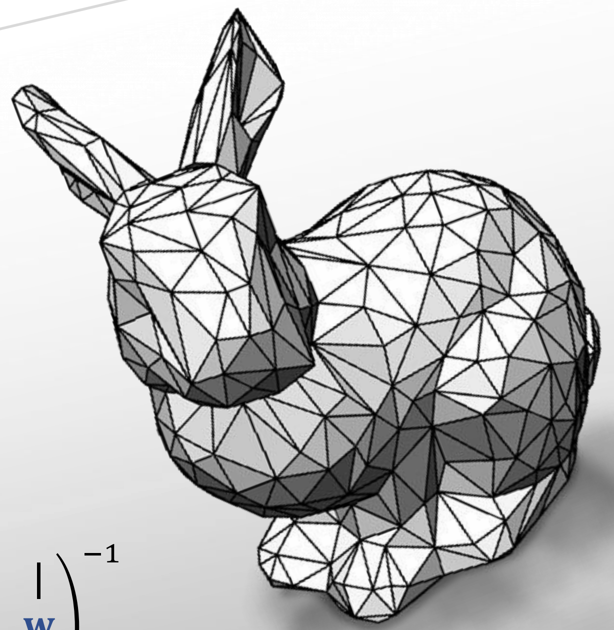
Derivation

Transform:

$$\mathbf{p} \to \begin{pmatrix} | & | & | \\ \mathbf{u} & \mathbf{v} & \mathbf{w} \\ | & | & | \end{pmatrix}^{-1} (\mathbf{p} - \mathbf{c})$$



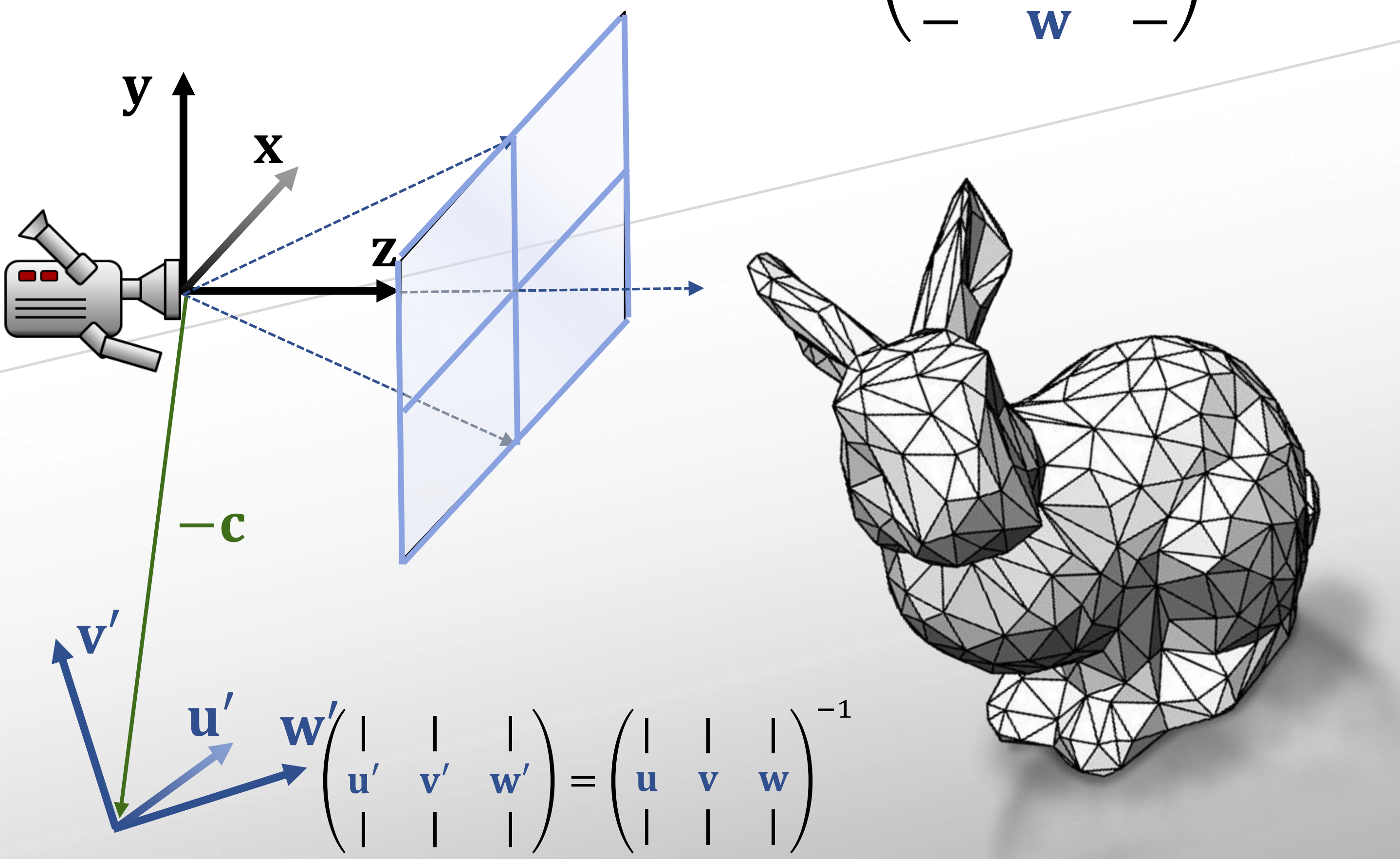
$$\begin{pmatrix} | & | & | \\ \mathbf{u}' & \mathbf{v}' & \mathbf{w}' \\ | & | & | \end{pmatrix} = \begin{pmatrix} | & | & | \\ \mathbf{u} & \mathbf{v} & \mathbf{w} \\ | & | & | \end{pmatrix}^{-1}$$



Derivation

Transform: $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ orthogonal!

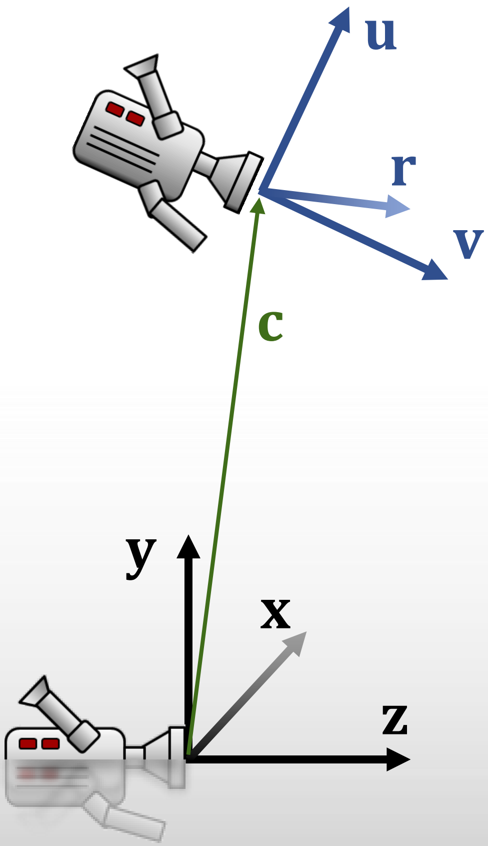
$$\mathbf{p} \rightarrow \begin{pmatrix} - & \mathbf{u} & - \\ - & \mathbf{v} & - \\ - & \mathbf{w} & - \end{pmatrix} (\mathbf{p} - \mathbf{c})$$



$$\begin{pmatrix} | & | & | \\ \mathbf{u}' & \mathbf{v}' & \mathbf{w}' \\ | & | & | \end{pmatrix} = \begin{pmatrix} | & | & | \\ \mathbf{u} & \mathbf{v} & \mathbf{w} \\ | & | & | \end{pmatrix}^{-1}$$

General Camera

general camera



Camera coordinate system $(\mathbf{u}, \mathbf{r}, \mathbf{v})$
Origin: $\mathbf{c}$

Transform:

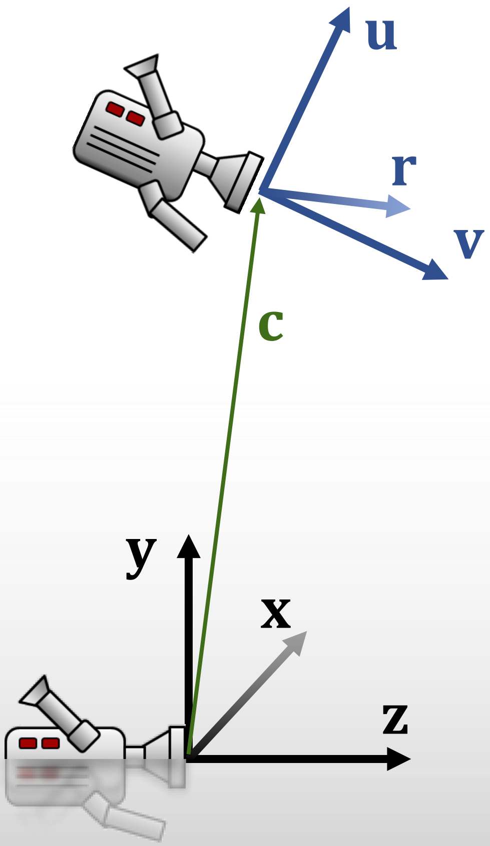
$$\mathbf{p} \rightarrow \begin{pmatrix} - & \mathbf{u} & - \\ - & \mathbf{v} & - \\ - & \mathbf{w} & - \end{pmatrix} (\mathbf{p} - \mathbf{c})$$

Standard coordinates $(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right)$

camera in origin,
view: z-direction

General Camera

general camera



Camera coordinate system $(\mathbf{u}, \mathbf{r}, \mathbf{v})$
Origin: $\mathbf{c}$

Homogeneous:

$$\mathbf{p} \to \begin{pmatrix} - & \mathbf{u} & - & | \\ - & \mathbf{v} & - & -\mathbf{c}' \\ - & \mathbf{w} & - & | \\ 0 & 0 & 0 & 1 \end{pmatrix} (\mathbf{p})$$

$$\mathbf{c}' = \begin{pmatrix} - & \mathbf{u} & - \\ - & \mathbf{v} & - \\ - & \mathbf{w} & - \end{pmatrix} \mathbf{c}$$

Summary

Projection (screen coord's)

$$\mathbf{P}_s = \begin{pmatrix} h/2 & 0 & 0 & w/2 \\ 0 & -h/2 & 0 & h/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\alpha}{2}\right)} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{array}{c} (f = 1) \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

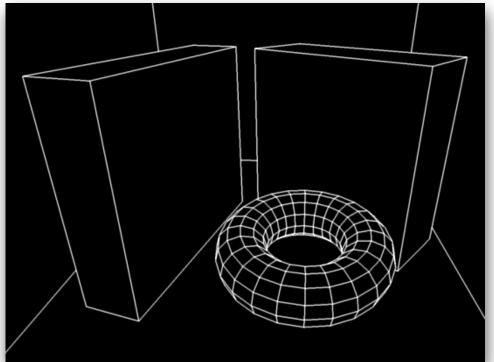
Add View Matrix

Benefit:

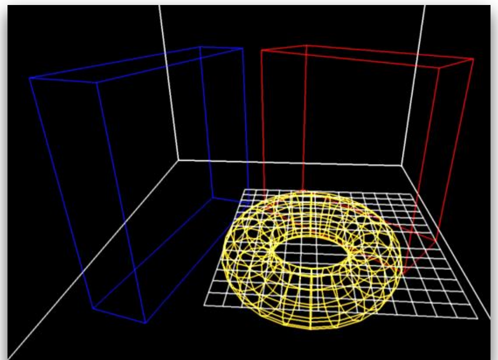
Still only one overall
4×4 matrix
to multiply with!

$$\mathbf{P}_s \cdot \overbrace{\begin{pmatrix} - & \mathbf{u} & - & | \\ - & \mathbf{v} & - & -\mathbf{c}' \\ - & \mathbf{w} & - & | \\ 0 & 0 & 0 & 1 \end{pmatrix}}$$

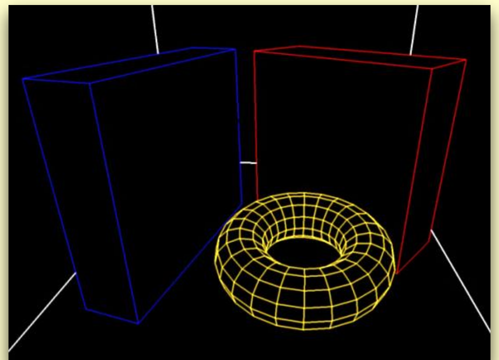
3D Rendering Steps



Geometric Model



Perspective



Visibility

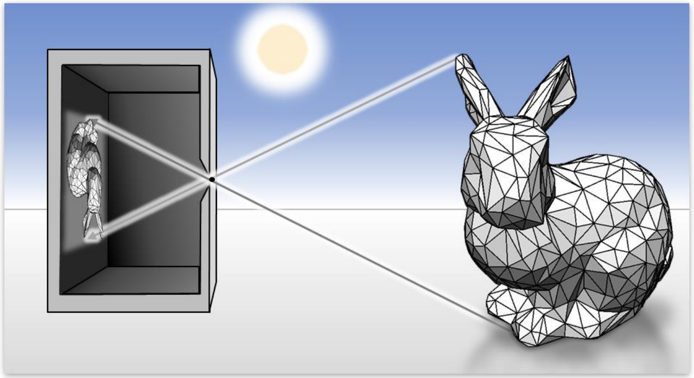
Local Illumination

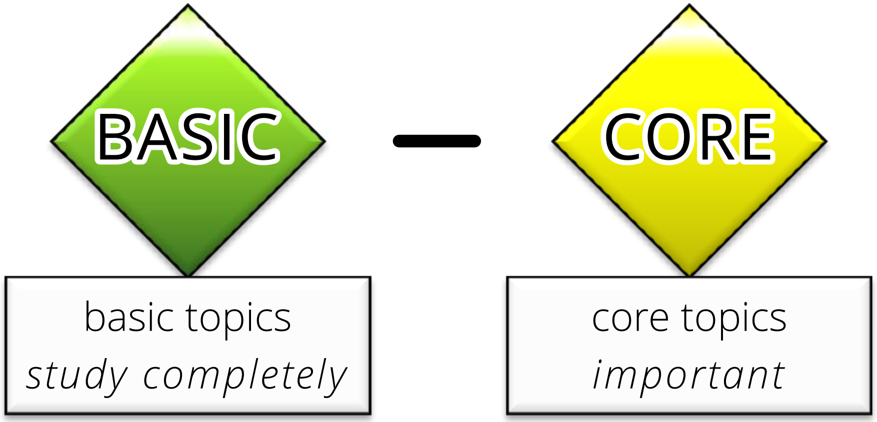
Smooth Shading

Simple Shadows

Global Illumination

Visibility Algorithms





Two Rendering Pipelines

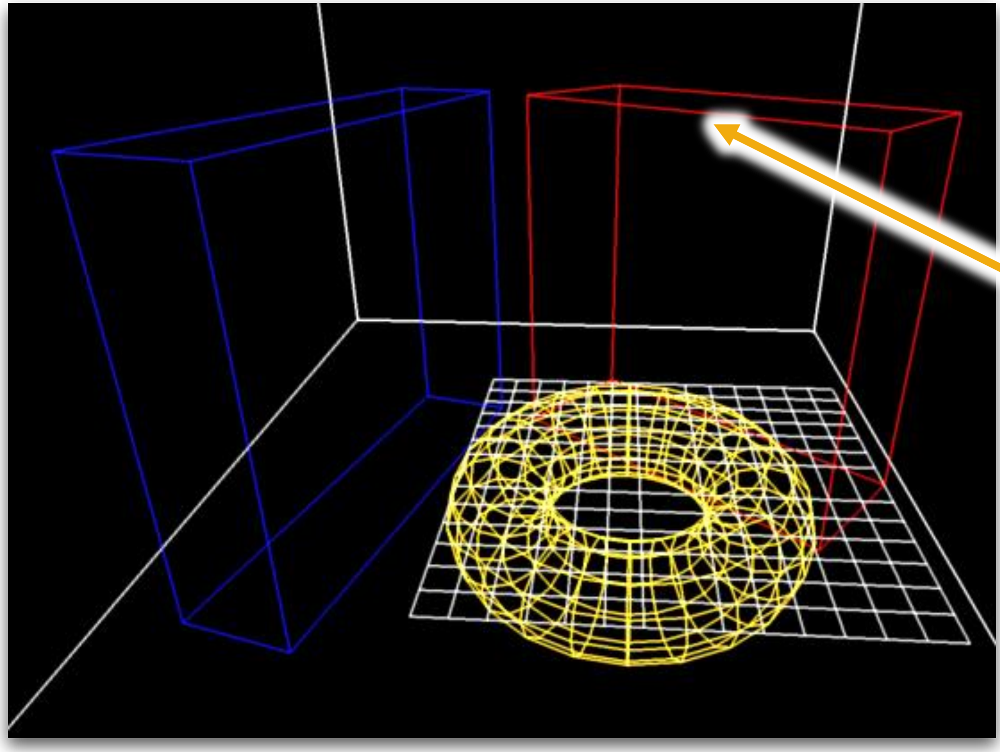
Rasterization

- Project all triangles to the screen
- Rasterize them (convert to pixels)
- Determine visibility
- Apply shading (compute color)

Raytracing

- Iterate over all pixels
- Determine visible triangle
- Compute shading, color pixel

Triangle / Polygon Rasterization



After Perspective Projection

Observations

Straight lines remain *straight*!

Triangles mapped to *triangles*

Polygons to *polyogns*

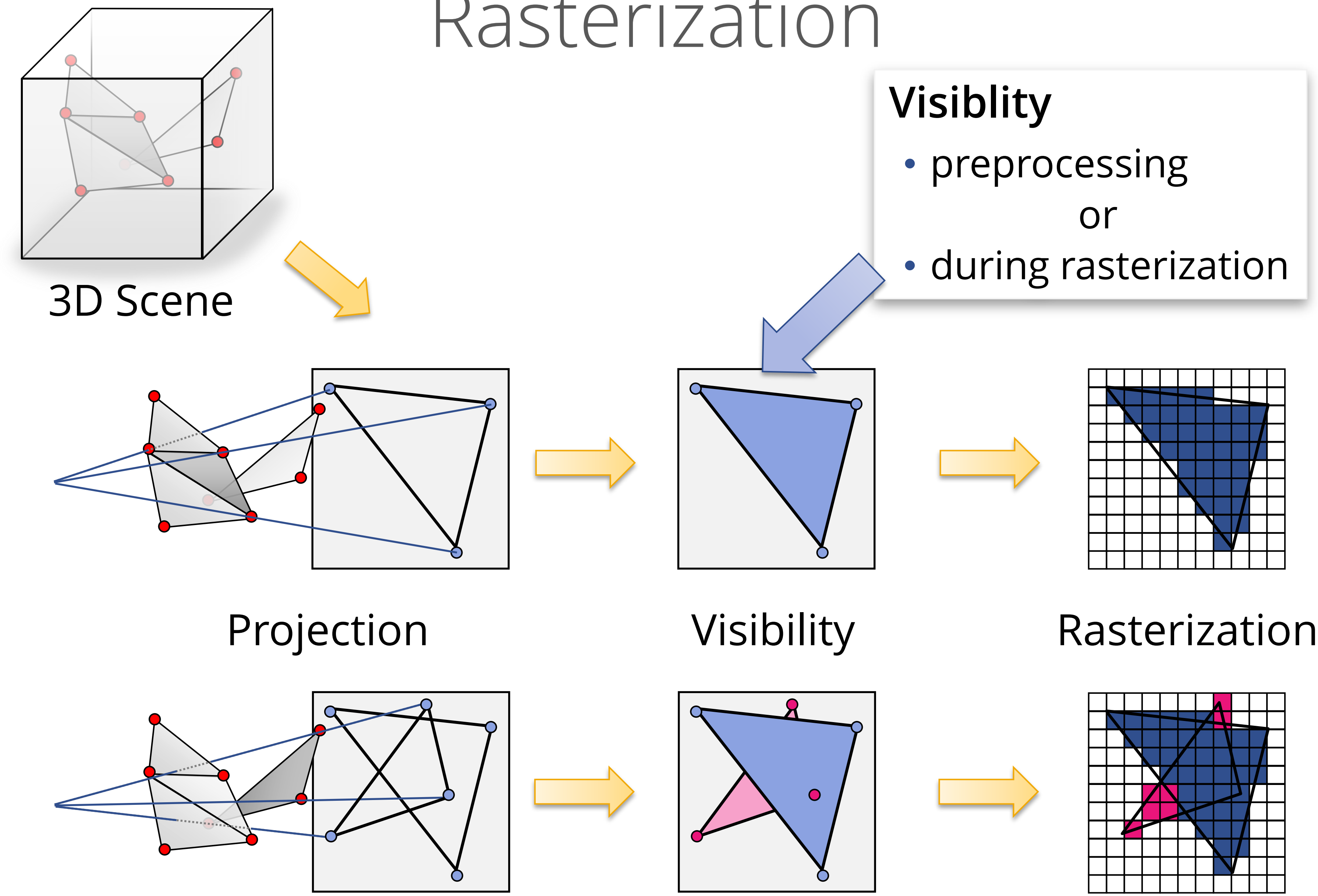
Rasterization

Visiblity

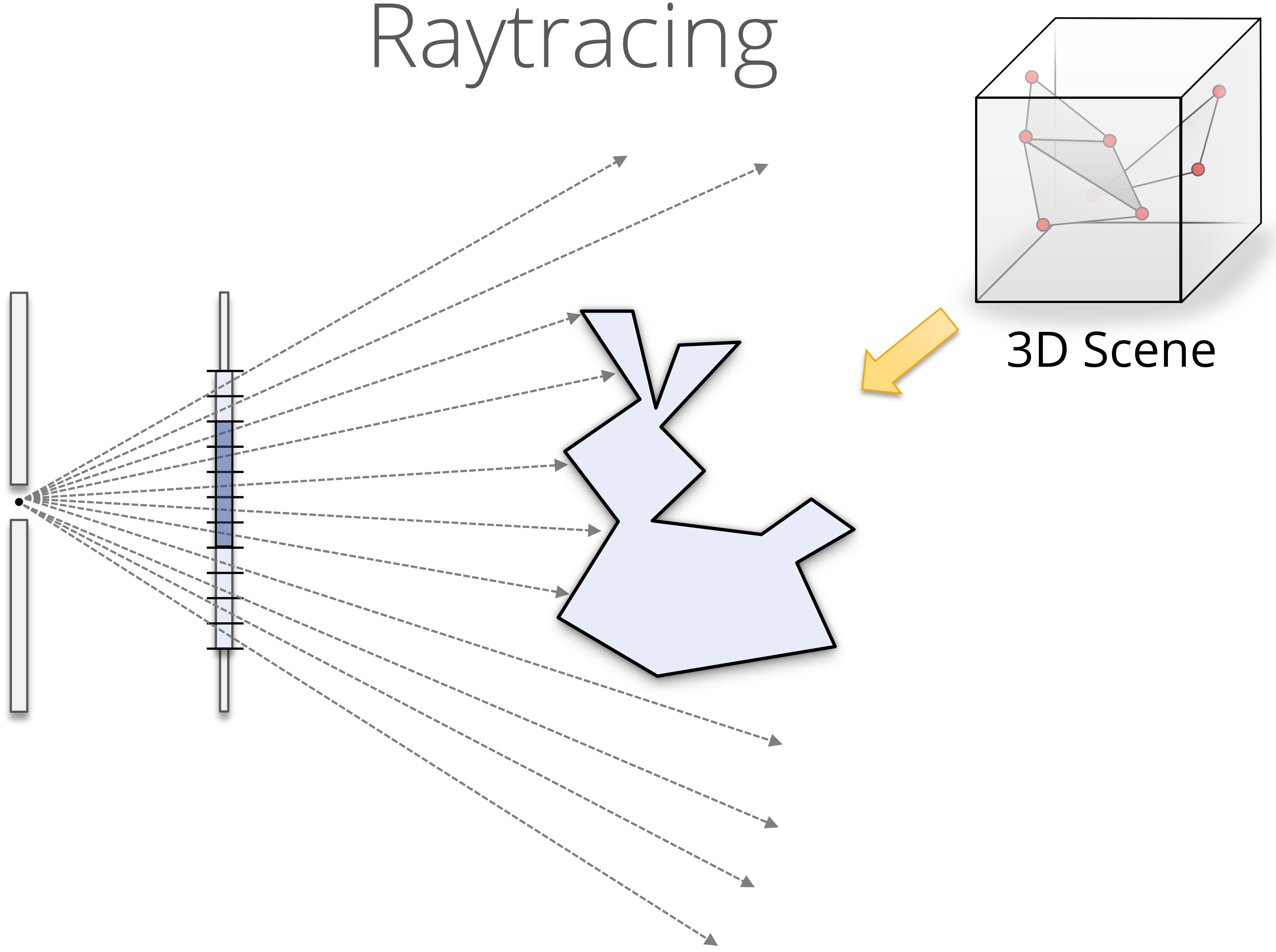
- preprocessing

  or

- during rasterization



Raytracing



Comparison

Rasterization

```
FOR (each triangle) {
    compute pixels covered
    ("fragments")
    FOR (all fragments) {
        fragment visible?
        IF (visible) {
            shade fragment
            write color
        }
    }
}
```

Raytracing

```
FOR (each pixel) {
    compute visible triangle
    IF (found) {
        shade fragment
        write color
    }
}
```

Rasterization

Focus for now:

- Rasterization (Raytracing covered later)

Two main algorithms

- Painter's algorithm (old)
  - Simple version
  - Correct version
- z-Buffer algorithm
  - Dominant real-time method today

Painter's Algorithm

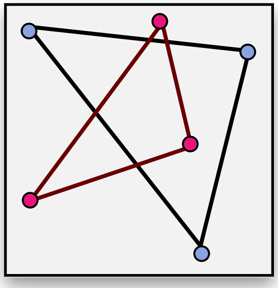
CORE

core topics
important

Painter's Algorithm

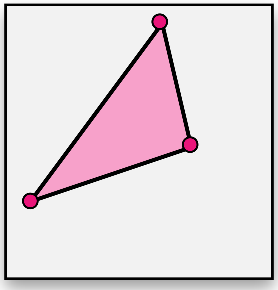
Painters Algorithm

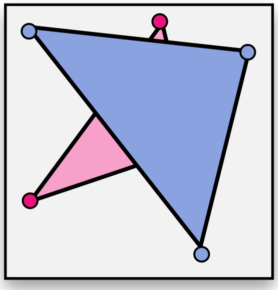
- Sort primitives back-to-front
- Draw with overwrite



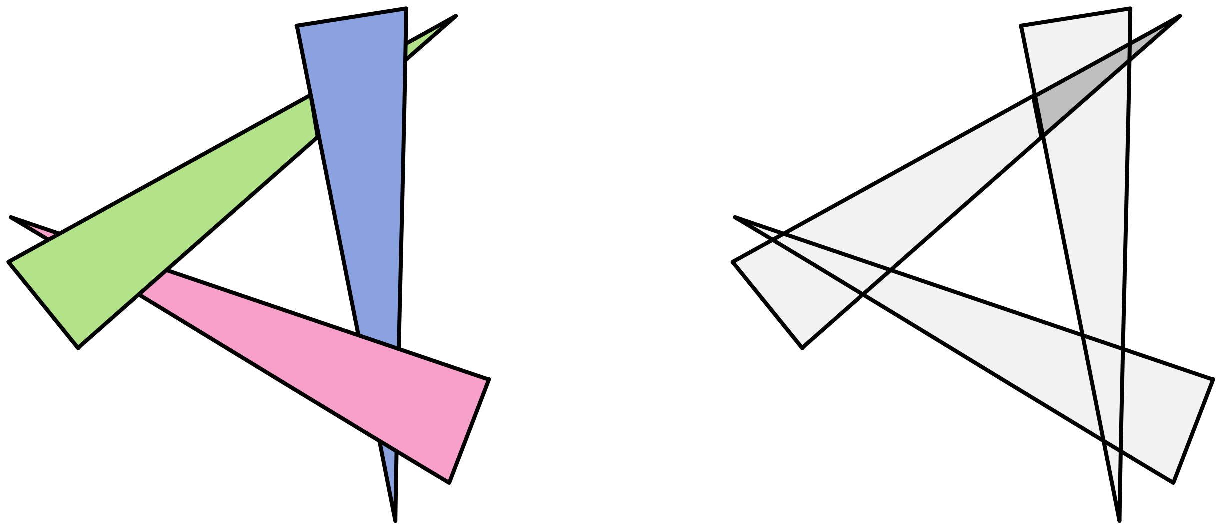
Drawbacks

- Slowish
  - $\mathcal{O}(n \cdot \log n)$ for n primitives
  - "Millions per second"
- Wrong
  - Not guaranteed to always work





Counter Example



Correct Algorithm

- Need to cut primitives
- Several strategies
  - Notable: BSP Algorithm in Quake
  - Old graphics textbooks list many variants
  - No need for us to go deeper

z-Buffer Algorithm

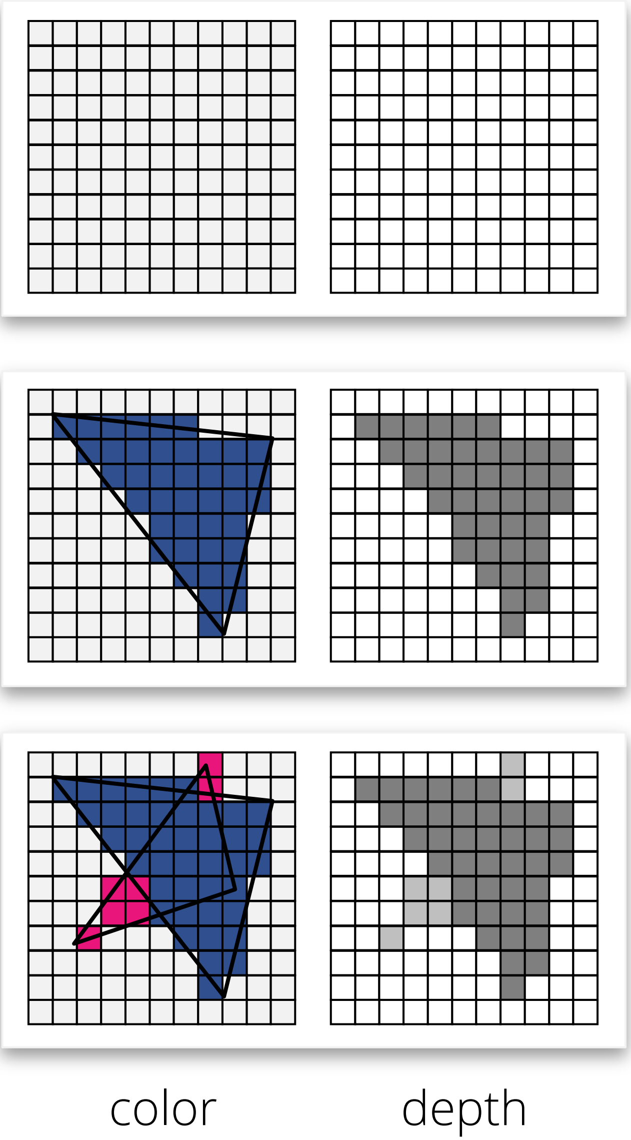
BASIC

basic topics
study completely

z-Buffer Algorithm

Algorithm

- Store depth value for each pixel
- Initialize to MAX_FLOAT
- Rasterize all primitives
  - Compute fragment depth & color
  - Do not overwrite if fragment is farer away than the one stored the one in the buffer



Discussion: z-Buffer

Advantages

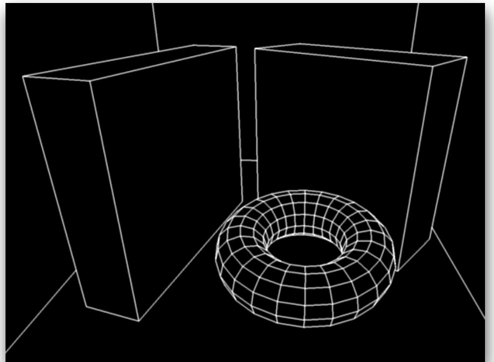
- Extremely simple
- Versatile – only primitive rasterization required
- Very fast
  - GeForce 2 Ultra: 2GPixel /sec
    (release year: 2000)
  - GeForce 700 GTX Titan: 35 GPixel / sec
    (release year: 2013)

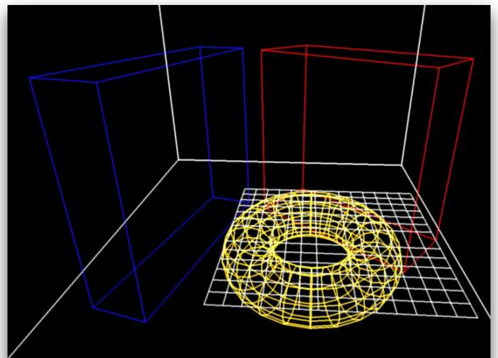
Discussion: z-Buffer

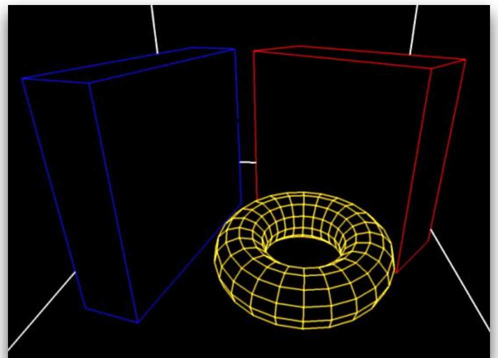
Disadvantages

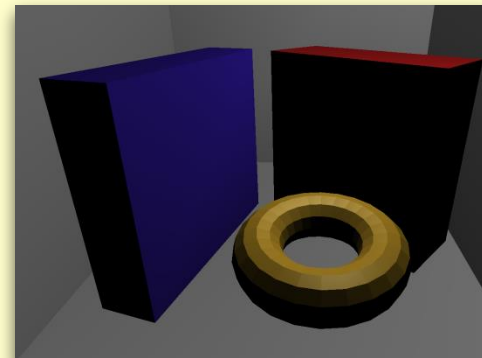
- Extra memory required
  - This was a serious in obstacle back then...
  - Invented 39 years ago (1974; Catmull / Straßer)
- Only pixel resolution
  - Need painter's algorithm for certain vector graphics computations
- No transparency
  - This is a real problem for 3D games / interactive media
  - Often fall-back to sorting
  - Solution: A-Buffer, but no hardware support

3D Rendering Steps


Geometric Model


Perspective


Visibility

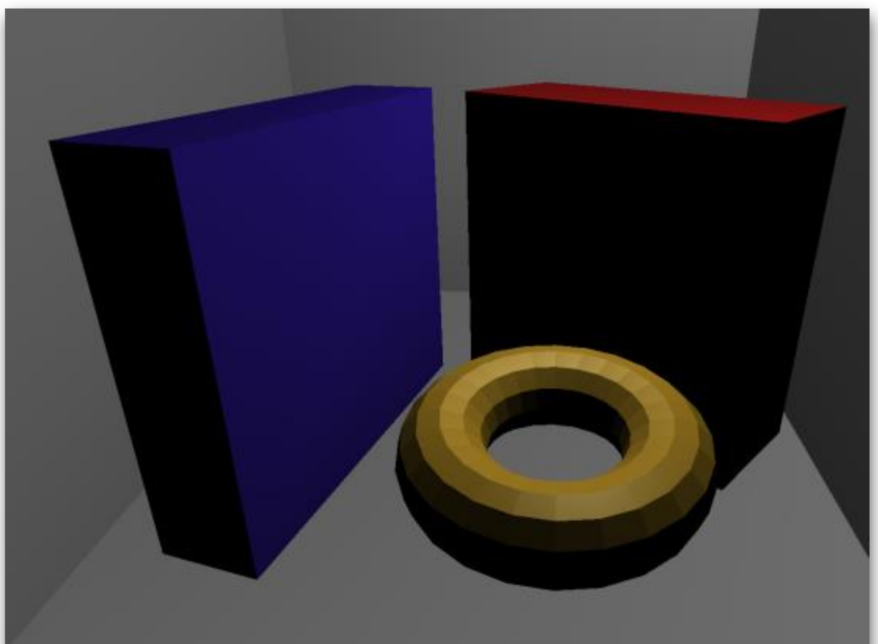

Local Illumination

Smooth Shading

Simple Shadows

Global Illumination

Shading Models

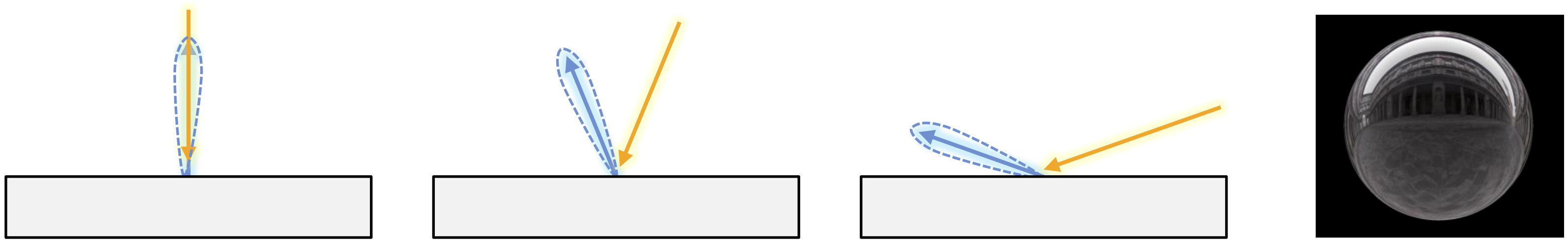


CORE

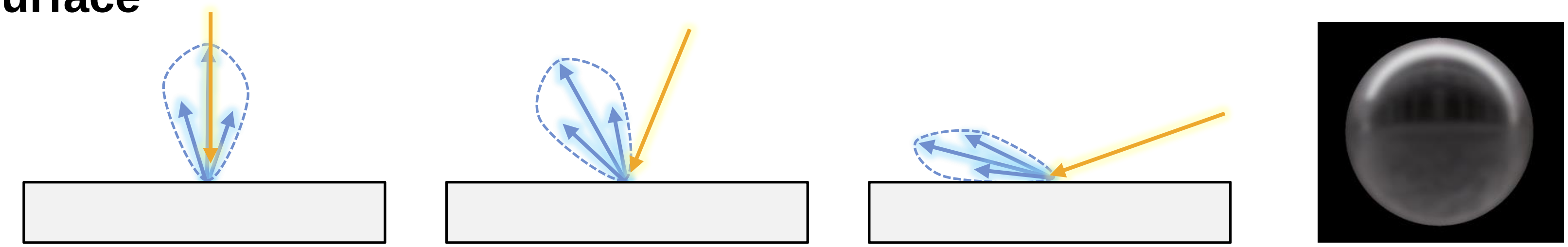
core topics
important

Reflectance Models

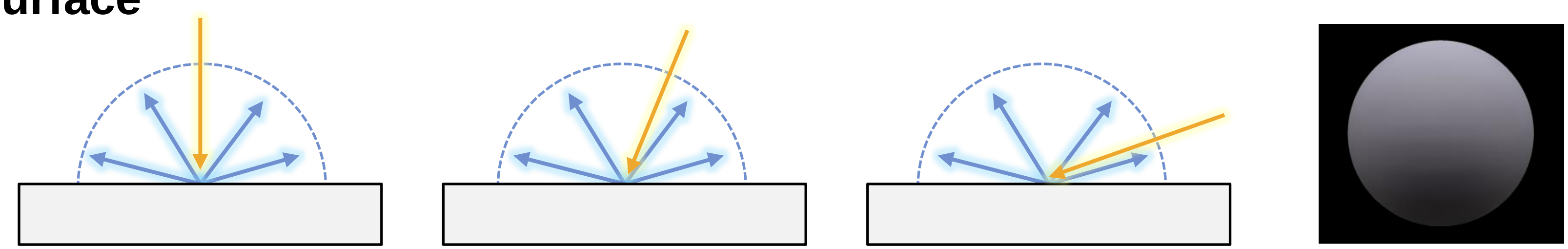
mirror



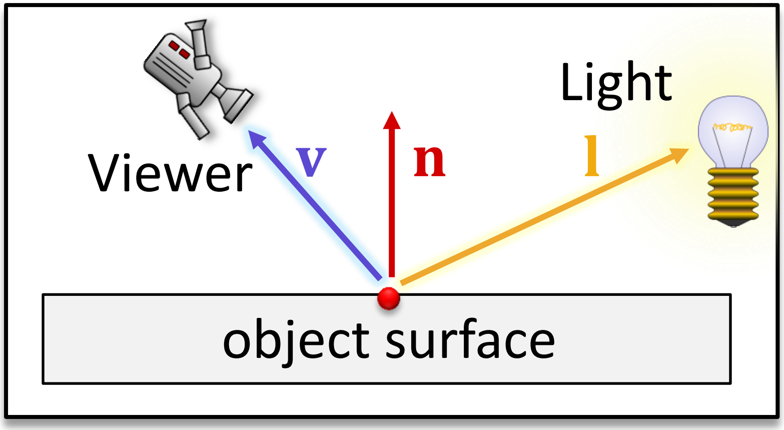
glossy surface

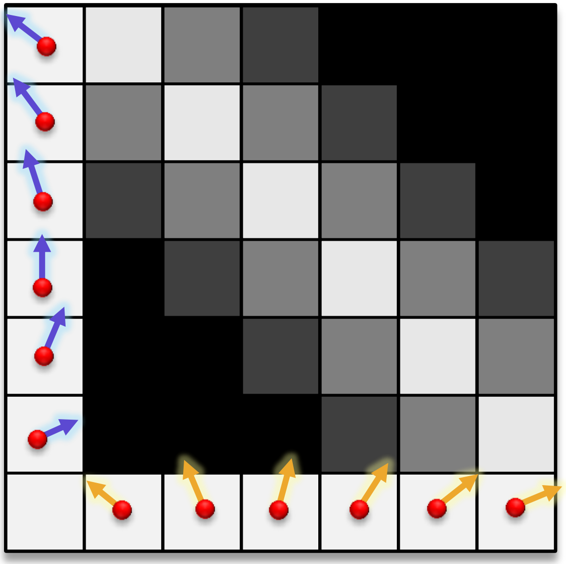


diffuse surface



Interaction with Surfaces



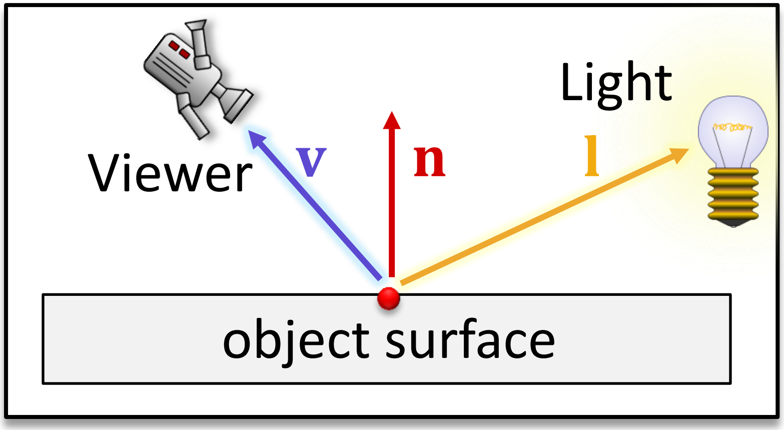


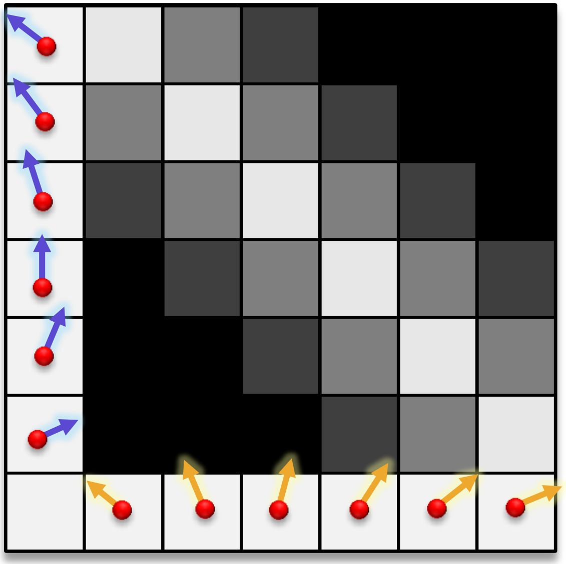
Formalization: BRDF

Local Shading Model

- Single point light source
- Shading model / material model
  - **Input:** light vector $\mathbf{l} = (\mathbf{pos}_{light} - \mathbf{pos}_{object})$
  - **Input:** view vector $\mathbf{v} = (\mathbf{pos}_{camera} - \mathbf{pos}_{object})$
  - **Input:** surface normal $\mathbf{n}$ (orthogonal to surface)
  - **Output:** *color* (RGB)

Interaction with Surfaces





Formalization: BRDF

General scenario

- Multiple light sources?
  - Light is linear
  - Multiple light sources: add up contributions
  - Double light strength ⇒ double light output

Remark

Simplify notation

- Define component-wise vector product

$$\mathbf{x} \circ \mathbf{y} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \circ \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} := \begin{pmatrix} x_1 \cdot y_1 \\ x_2 \cdot y_2 \\ x_3 \cdot y_3 \end{pmatrix}$$

- No fixed convention in literature
- The symbol “∘” only used in these lecture slides!

Remark

Lighting Calculations

- Need to perform calculations for r, g, b-channels
- Often:

$$output_r = light_r \cdot material_r \cdot function(\mathbf{v}, \mathbf{l}, \mathbf{n})$$
$$output_g = light_g \cdot material_g \cdot function(\mathbf{v}, \mathbf{l}, \mathbf{n})$$
$$output_b = light_b \cdot material_b \cdot function(\mathbf{v}, \mathbf{l}, \mathbf{n})$$

- Shorter

$$\mathbf{output} = \mathbf{light\_strength} \circ \mathbf{material} \cdot function(\mathbf{v}, \mathbf{l}, \mathbf{n})$$

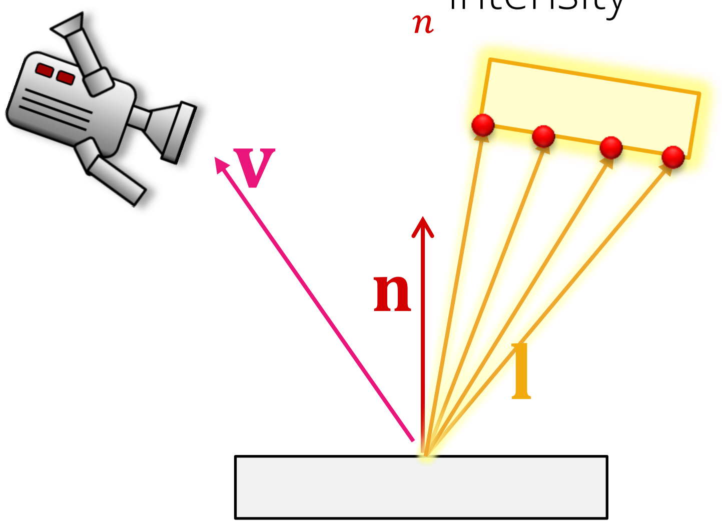
Area Light Sources

Area Light Sources

- Integrate over area
- In practice often:
  - Sample with many point-light sources
  - Add-up contributions

n light sources

$\frac{1}{n}$ intensity



Shading Effects

Shading effects

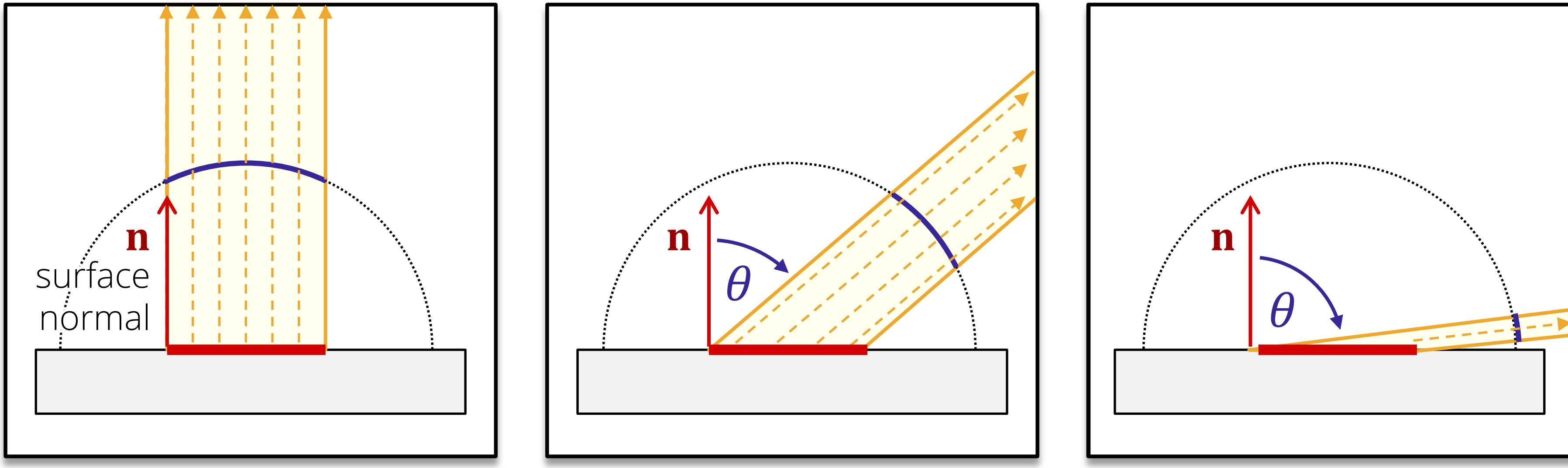
- Diffuse reflection
- “Ambient reflection”
- Perfect mirrors
- Glossy reflection
  - Phong / Blinn-Phong
  - (Cook Torrance)
- Transparency & refraction

Shading Effects

Shading effects

- Diffuse reflection
- “Ambient reflection”
- Perfect mirrors
- Glossy reflection
    - Phong / Blinn-Phong
    - (Cook Torrance)
- Transparency & refraction

Diffuse ("Lambertian") Surfaces



Equation

(set to zero if negative)

$$c \sim \cos\theta$$

- Less light received at flat angles

$$\mathbf{c} = \mathbf{c}_r \circ \mathbf{c}_l \cdot \cos\theta$$

light color

surface color

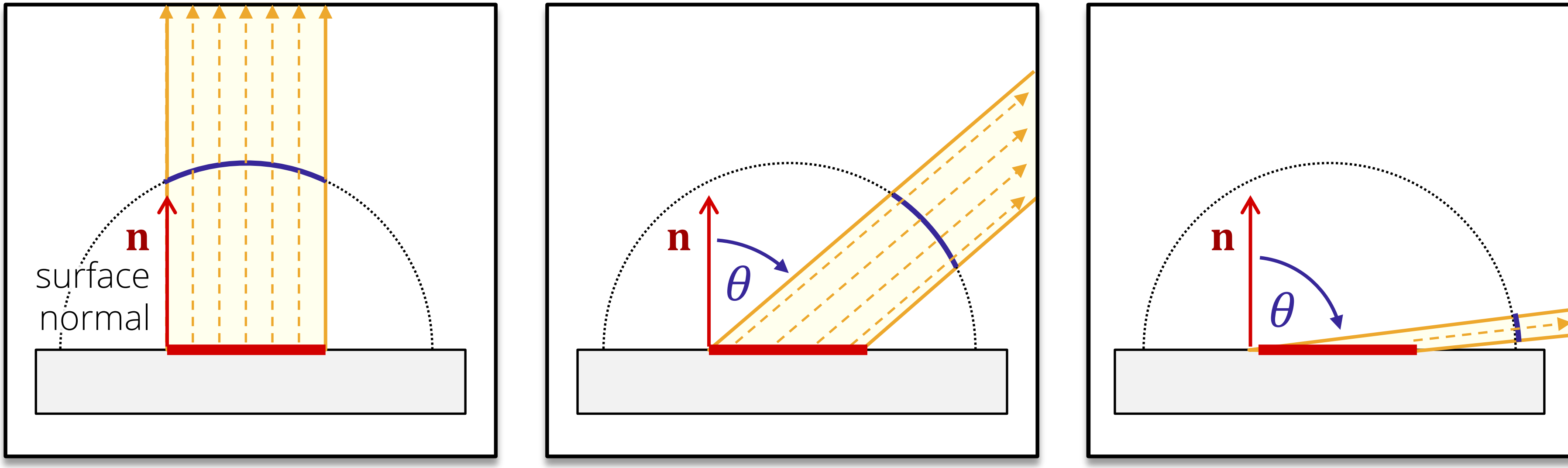
c – intensity (scalar)

$\mathbf{c}$ – color (RGB, $\mathbb{R}^3$)

$\mathbf{c}_r$ – surface color (RGB)

$\mathbf{c}_l$ – light color (RGB)

Diffuse ("Lambertian") Surfaces



Equation

$$c \sim \cos\theta \, \frac{1}{dist^2}$$

- Less light received at flat angles

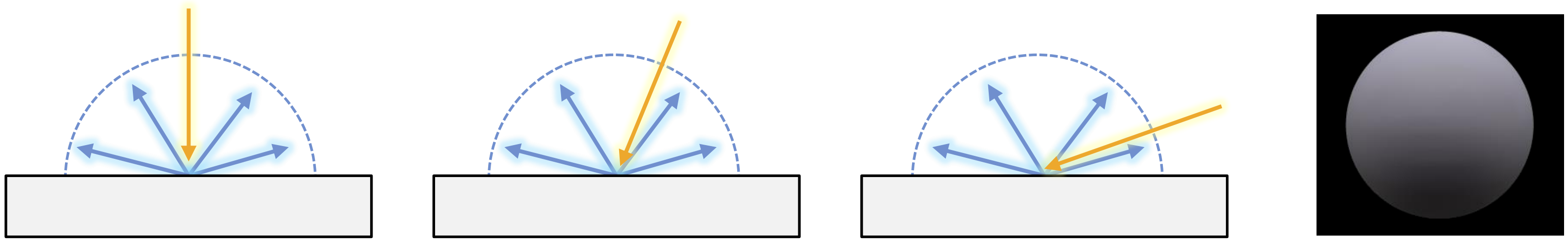
c – intensity (scalar)
$\mathbf{c}$ – color (RGB, $\mathbb{R}^3$)
$\mathbf{c}_r$ – surface color (RGB)
$\mathbf{c}_l$ – light color (RGB)

$$\mathbf{c} = \mathbf{c}_r \circ \mathbf{c}_l \cdot \cos(\theta) \cdot \frac{1}{dist^2}$$

light color
surface color

Attenuation: $\frac{1}{dist^2}$
(point lights)

Diffuse Reflection



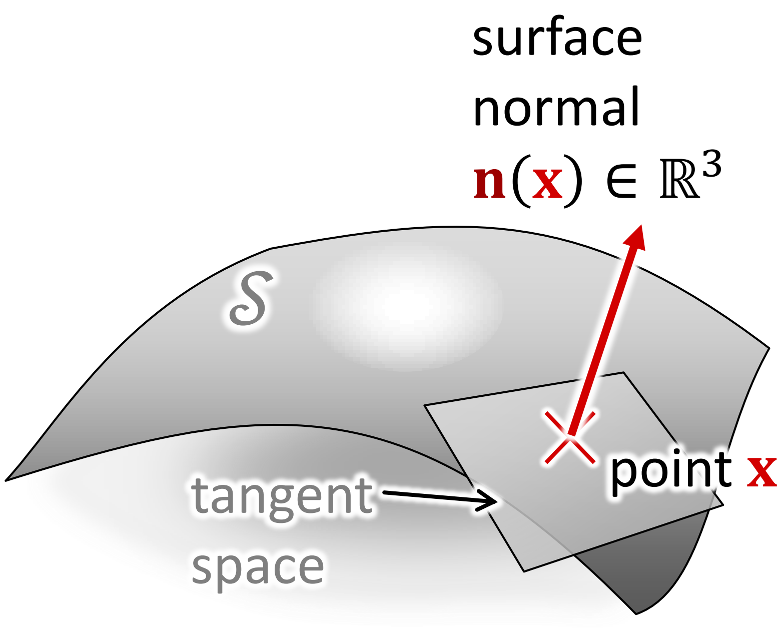
Diffuse Reflection

- Very rough surface microstructure
- Incoming light is scattered in all directions uniformly
- “Diffuse” surface (material)
- “Lambertian” surface (material)

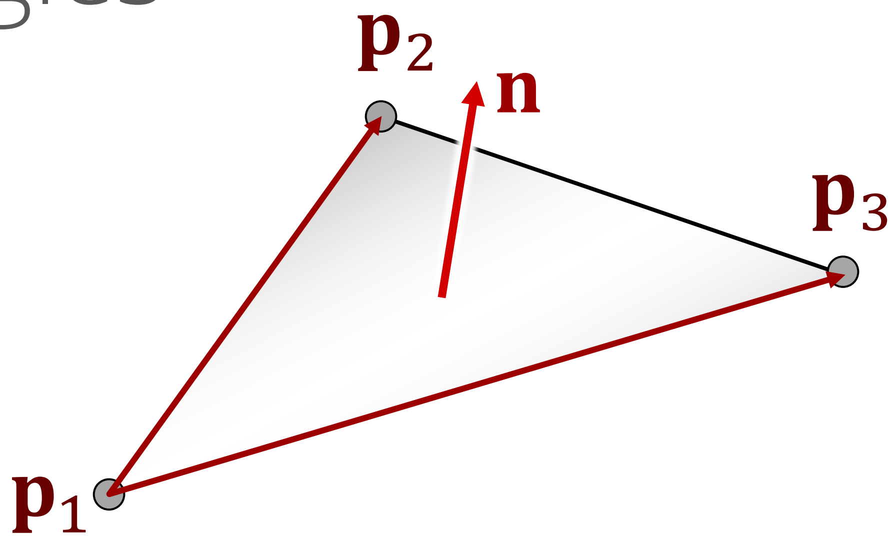
Surface Normal?

What is a surface normal?

- Tangent space:
  - Plane approximation at a point $\mathbf{x} \in \mathcal{S}$
- Normal vector:
  - Perpendicular to that plane
- Oriented surfaces:
  - Pointing outwards (by convention)
  - Orientation defined only for closed solids



Triangles



Single Triangle

- Parametric equation

$$\{\mathbf{p}_1 + \lambda(\mathbf{p}_2 - \mathbf{p}_1) + \mu(\mathbf{p}_3 - \mathbf{p}_1) | \lambda, \mu \in \mathbb{R}\}$$

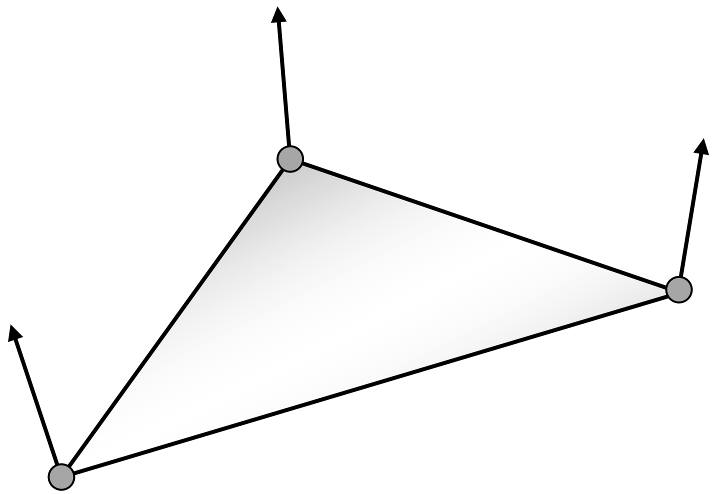
  - Tangent space: the plane itself

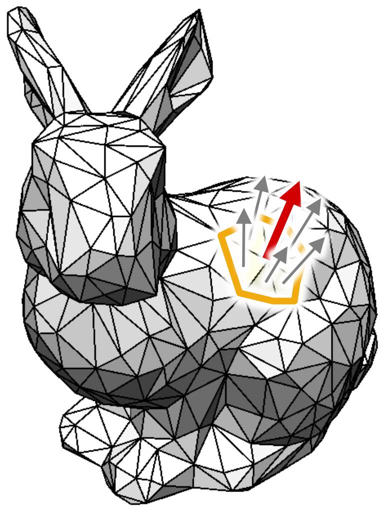
- Normal vector

$$(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)$$

  - Orientation convention:
    $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ oriented counter-clockwise
  - Length: Any positive multiple works (often $\|\mathbf{n}\| = 1$)

Triangle Meshes

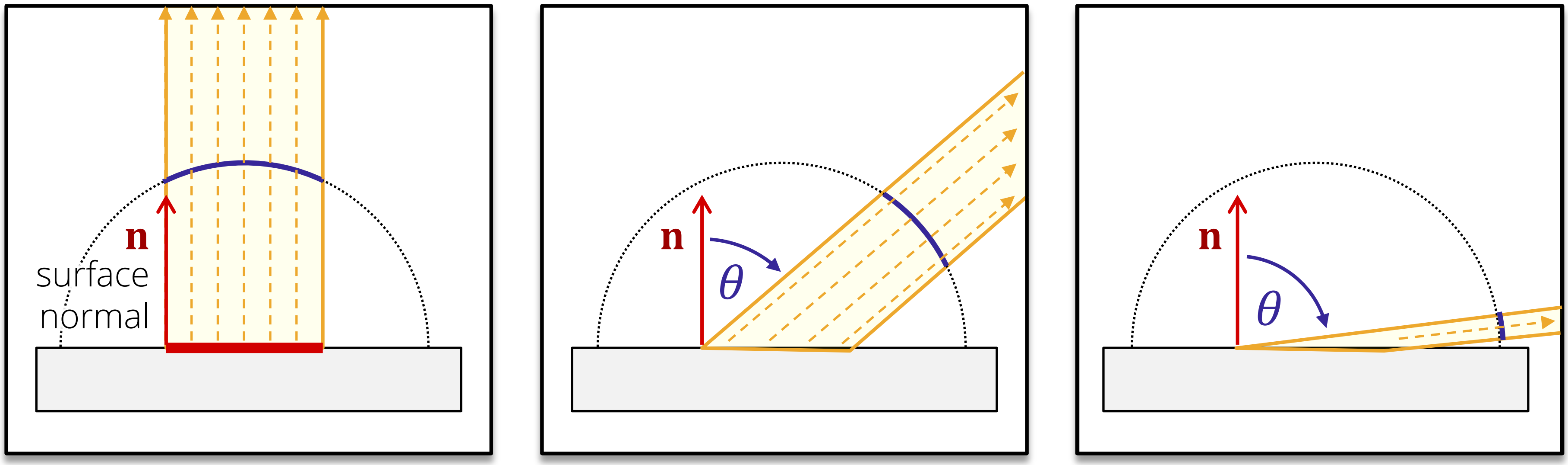




Smooth Triangle Meshes

- Store three different “vertex normals”
  - E.g., from original surface (if known)
- Heuristic:
  Average neighboring triangle normals

Lambertian Surfaces



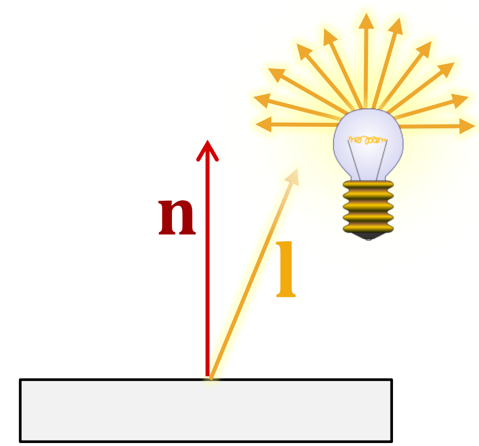
Equation

$$\begin{aligned}\mathbf{c} &= \mathbf{c}_r \circ \mathbf{c}_l \cdot \cos\theta \\ &= \mathbf{c}_r \circ \mathbf{c}_l \cdot \langle \mathbf{n}, \mathbf{l} \rangle\end{aligned}$$

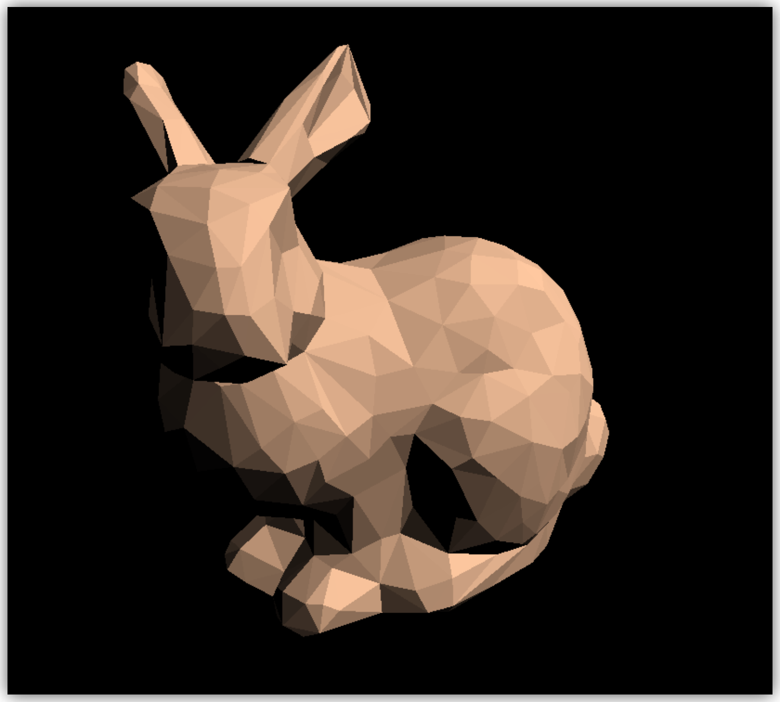
light direction

normal vector

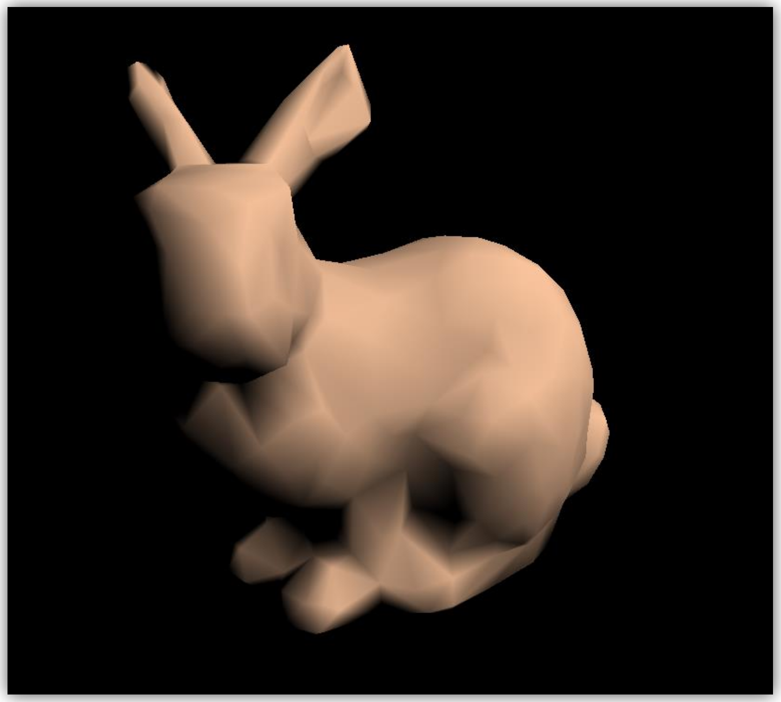
(assuming: $\|\mathbf{n}\| = \|\mathbf{l}\| = 1$)



Lambertian Bunny



Face Normals



Interpolated
Normals

Shading Effects

Shading effects

- Diffuse reflection
- “Ambient reflection”
- Perfect mirrors
- Glossy reflection
  - Phong / Blinn-Phong
  - (Cook Torrance)
- Transparency & refraction

“Ambient Reflection”

Problem

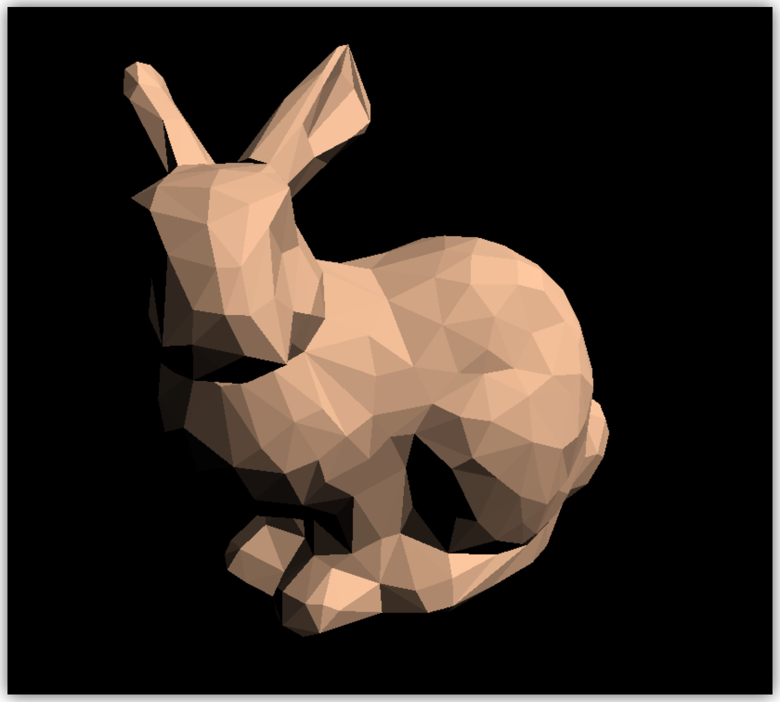
- Shadows are pure black
- Realistically, they should be gray
  - Some light should bounce around…
- Solution: Add constant

$$\mathbf{c} = \mathbf{c}_a \circ \mathbf{c}_a$$

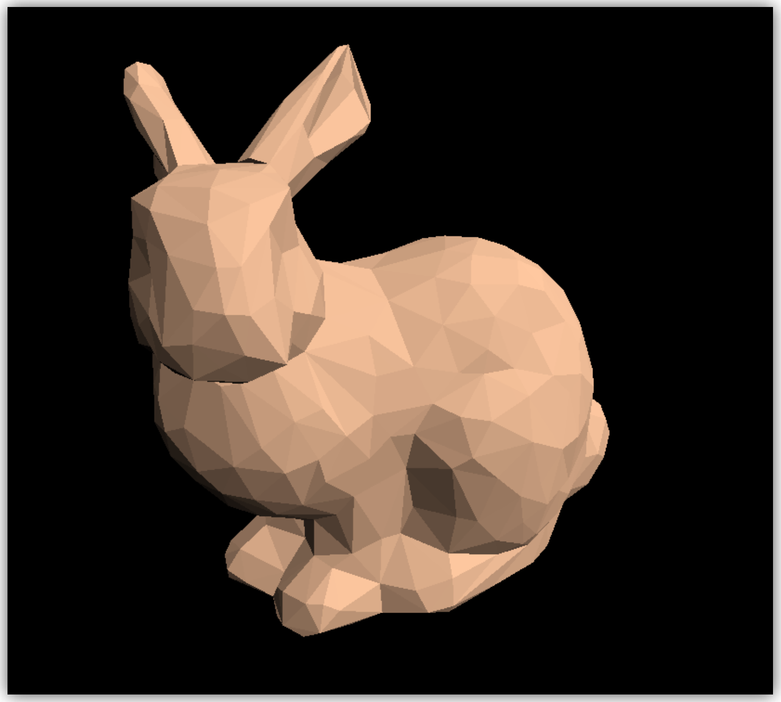
ambient light color
surface color

- Not very realistic
  - Need global light transport simulation for realistic results

Ambient Bunny



Pure Lambertian



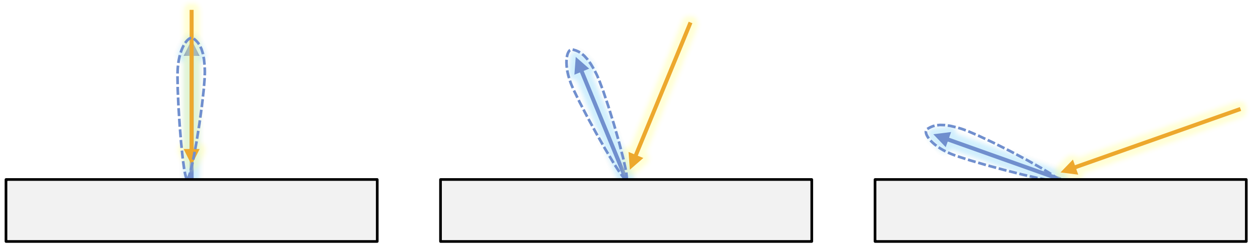
Mixed with Ambient Light

Shading Effects

Shading effects

- Diffuse reflection
- "Ambient reflection"
- Perfect mirrors
- Glossy reflection
  - Phong / Blinn-Phong
  - (Cook Torrance)
- Transparency & refraction

Perfect Reflection

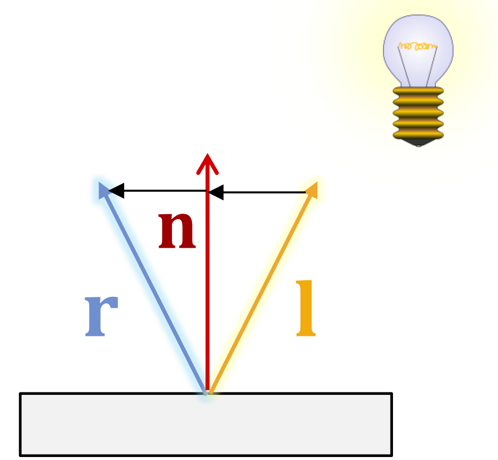


Perfect Reflection

- Rays are perfectly reflected on surface
- Reflection about surface normal

$$\mathbf{r} = 2(\langle \mathbf{n}, \mathbf{l} \rangle \cdot \mathbf{n} - \mathbf{l}) + \mathbf{l},$$
$$\|\mathbf{n}\| = 1$$

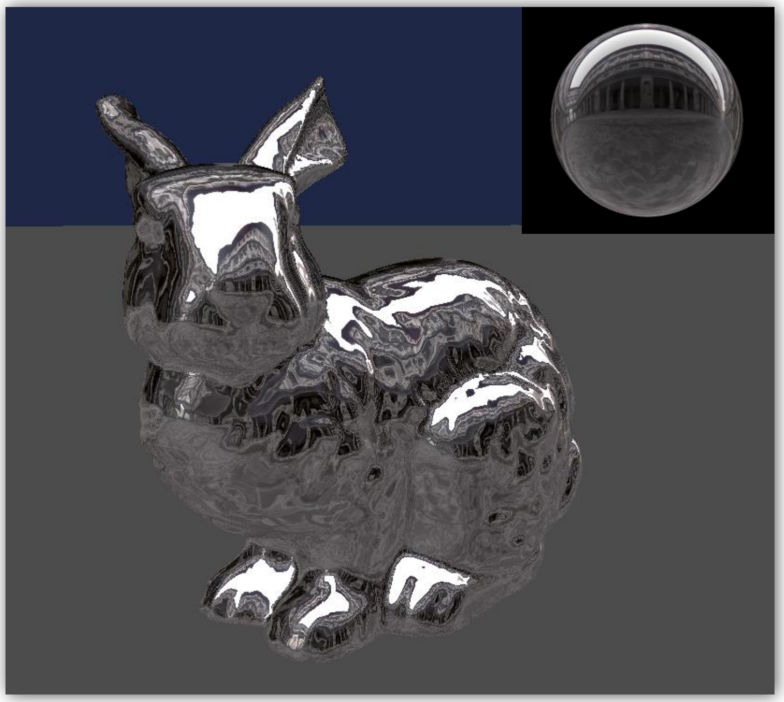
$\mathbf{l}$ arbitrary



Silver Bunny

Perfect Reflection

- Difficult to compute
  - Need to match camera and light emitter
- More later:
  - Recursive raytracing
  - Right image: Environment mapping



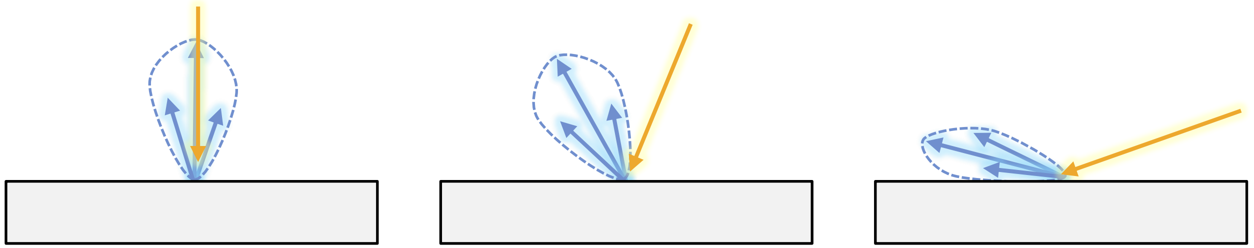
Reflective Bunny
(Interpolated Normals)

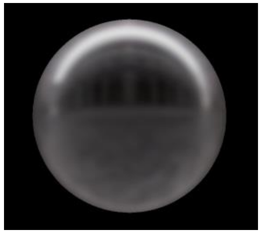
Shading Effects

Shading effects

- Diffuse reflection
- “Ambient reflection”
- Perfect mirrors
- Glossy reflection
  - Phong / Blinn-Phong
  - (Cook Torrance)
- Transparency & refraction

Glossy Reflection





Glossy Reflection

- Imperfect mirror
- Semi-rough surface
- Various models

Phong Illumination Model

Traditional Model: Phong Model

- Physically incorrect
  (e.g.: energy conservation not guaranteed)
- But “looks ok”
  - Always looks like plastic
  - On the other hand, our world is full of plastic…

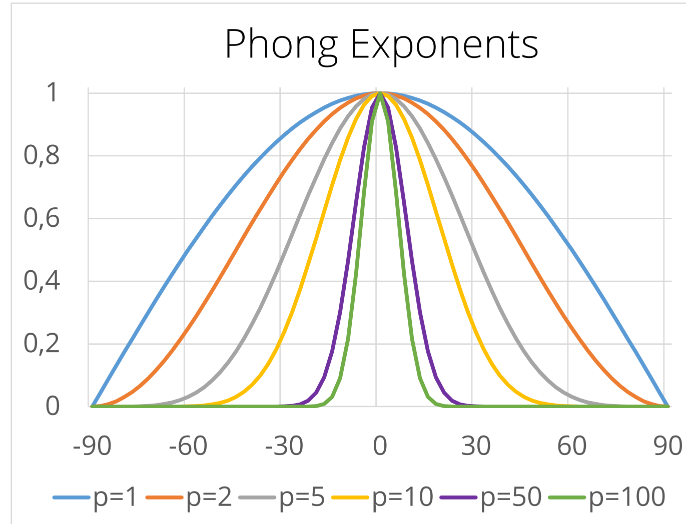
How does it work?

Phong Model:

- "Specular" (glossy) part:

$$\mathbf{c} = \mathbf{c}_p \circ \mathbf{c}_l \cdot \underbrace{\left\langle \frac{\mathbf{r}}{\|\mathbf{r}\|}, \frac{\mathbf{v}}{\|\mathbf{v}\|} \right\rangle}_{\cos \angle \mathbf{r},\mathbf{v}}^{p}$$

(high-) light color



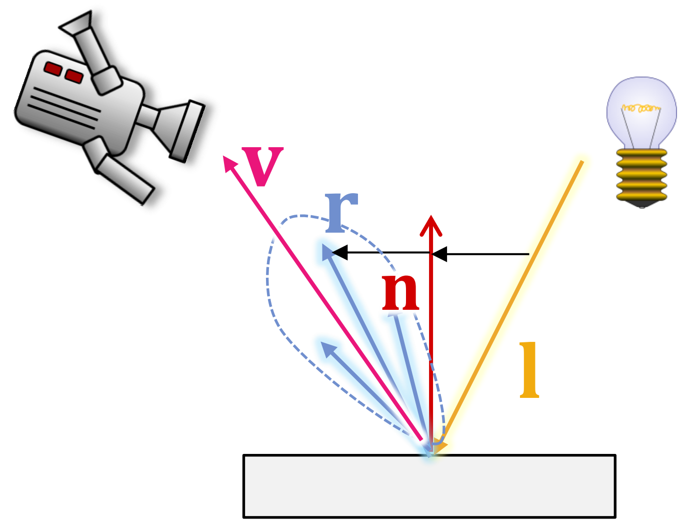
- Ambient part:

$$\mathbf{c} = \mathbf{c}_r \circ \mathbf{c}_a$$

- Diffuse part:

$$\mathbf{c} = \mathbf{c}_r \circ \mathbf{c}_l \cdot \langle \mathbf{n}, \mathbf{l} \rangle$$

- Add all terms together

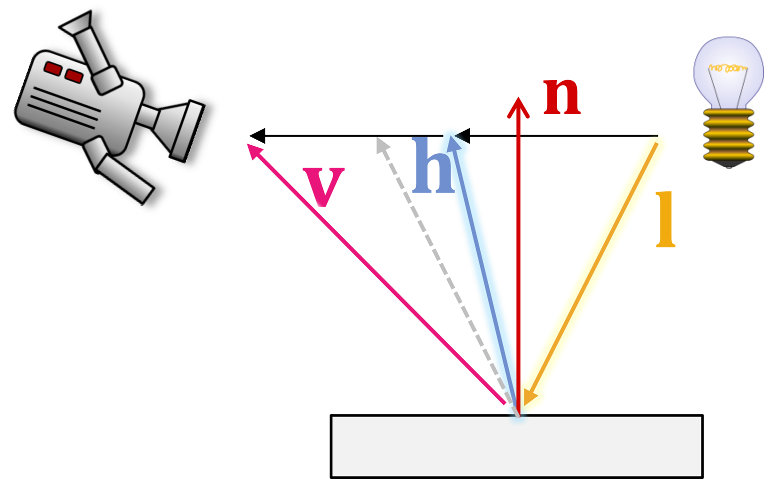


Blinn-Phong

Blinn-Phong Model:

- "Specular" (glossy) part:

$$\mathbf{c} = \mathbf{c}_p \circ \mathbf{c}_l \cdot \underbrace{\left\langle \frac{\mathbf{h}}{\|\mathbf{h}\|}, \frac{\mathbf{n}}{\|\mathbf{n}\|} \right\rangle}_{\cos \angle \mathbf{h},\mathbf{n}}^{p}$$



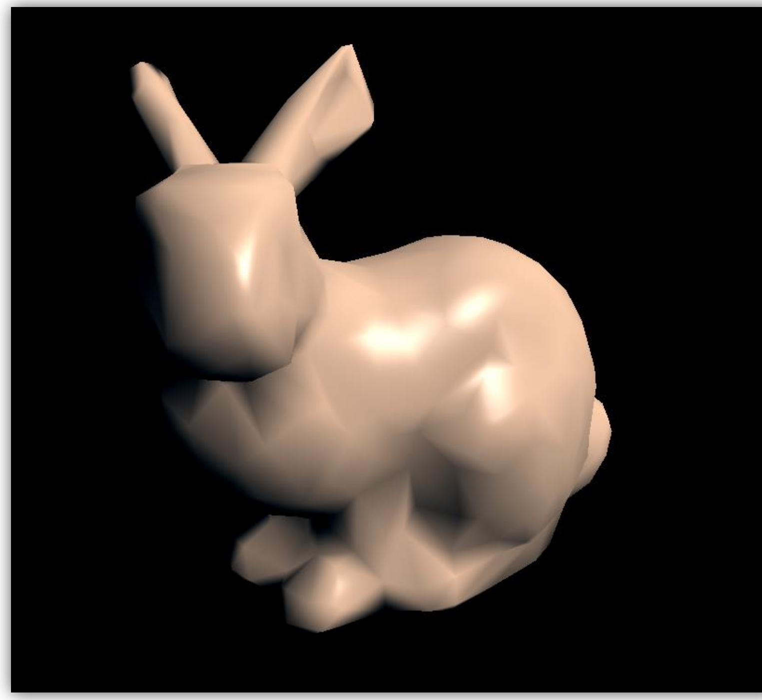
- Half-angle direction

$$\mathbf{h} = \frac{\mathbf{1}}{\mathbf{2}}\left(\frac{\mathbf{l}}{\|\mathbf{l}\|} + \frac{\mathbf{v}}{\|\mathbf{v}\|}\right)$$

- In the plane: $\angle\left(\frac{\mathbf{h}}{\|\mathbf{h}\|}, \frac{\mathbf{n}}{\|\mathbf{n}\|}\right) = \frac{1}{2}\angle\left(\frac{\mathbf{r}}{\|\mathbf{r}\|}, \frac{\mathbf{v}}{\|\mathbf{v}\|}\right)$
  - Approximation in 3D

Phong+Diffuse+Ambient Bunny

Blinn-Phong Bunny



Interpolated Normals

Phong+Diffuse+Ambient Bunny

Blinn-Phong Bunny

Interpolated Normals

Cook-Torrance Model

advanced topics
main ideas

Physically-Motivated Model

- D – Infinitesimal micro-facets
  - Characterize by distribution
  - Expected reflection (density)
  - Gaussian, Beckmann,...
  - Approximate occlusion term (G)
- F – Fresnel term
  - Model: wave-optics
  - Interaction of wave with surface under different angles
  - Percentage reflection/refraction

$$c_{spec} = \frac{D \cdot G \cdot F}{4\langle \mathbf{v}, \mathbf{n} \rangle \langle \mathbf{n}, \mathbf{l} \rangle}$$

$$F(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5$$

$\cos\theta = \langle \mathbf{h}, \mathbf{v} \rangle$ $\quad$ $R_0 =$"ratio of refractive indices"

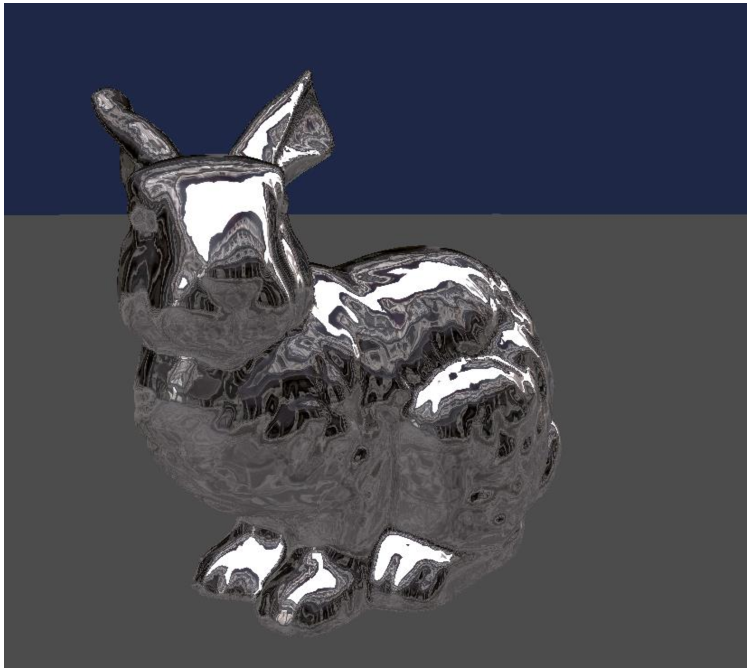
Artistic “Fresnel” Reflection

Exponent 4

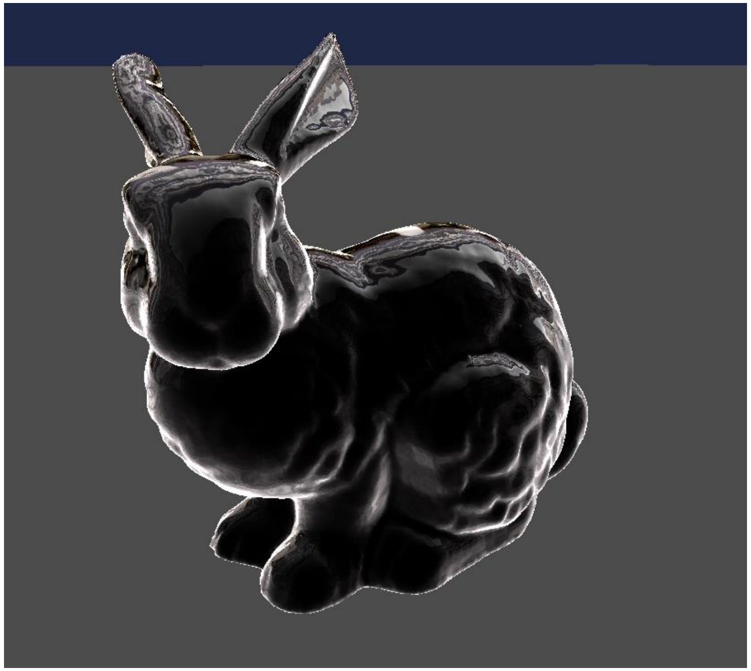
Approx. Fresnel-Reflection

$$F(\theta) \sim (1 - \cos\theta)^p$$

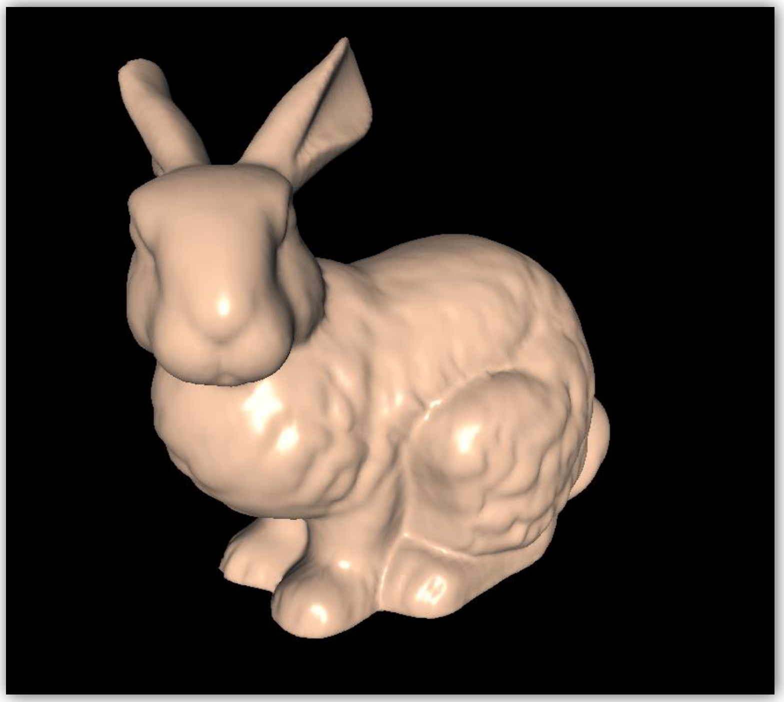
unweighted reflection



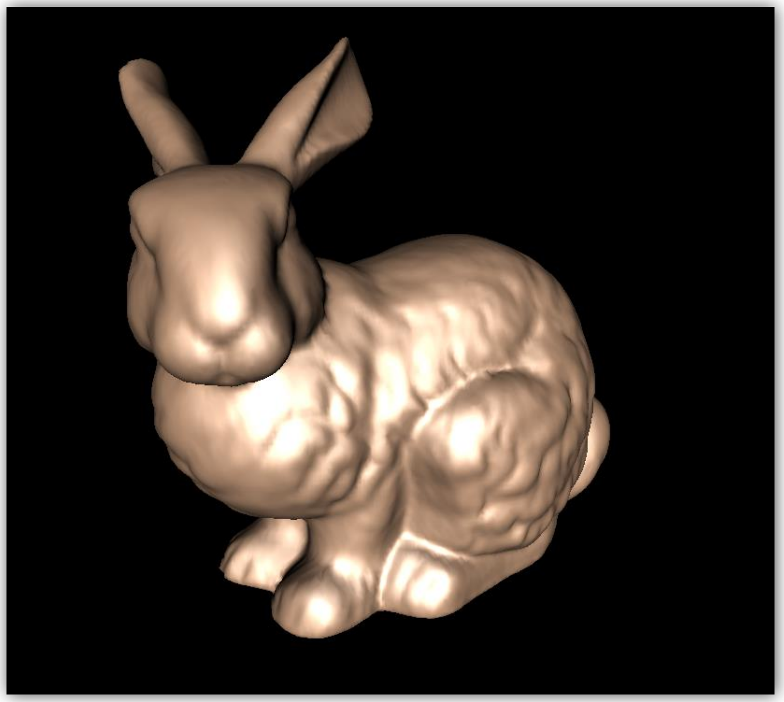
Exponent 5



Better Models



Phong Bunny



Cook-Torrance Model

Shading Effects

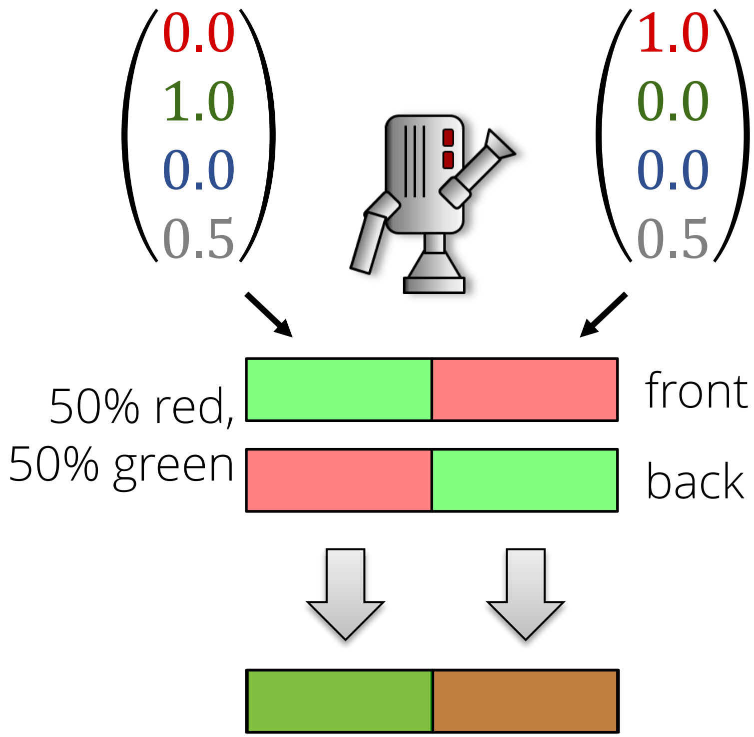
Shading effects

- Diffuse reflection
- “Ambient reflection”
- Perfect mirrors
- Glossy reflection
  - Phong / Blinn-Phong
  - (Cook Torrance)
- Transparency & refraction

Transparency

Transparency

- "Alpha-blending"
- α = "opacity"
- Color + opacity: RGBα



Blending

- Mix in α of front color, keep $1-\alpha$ of back color

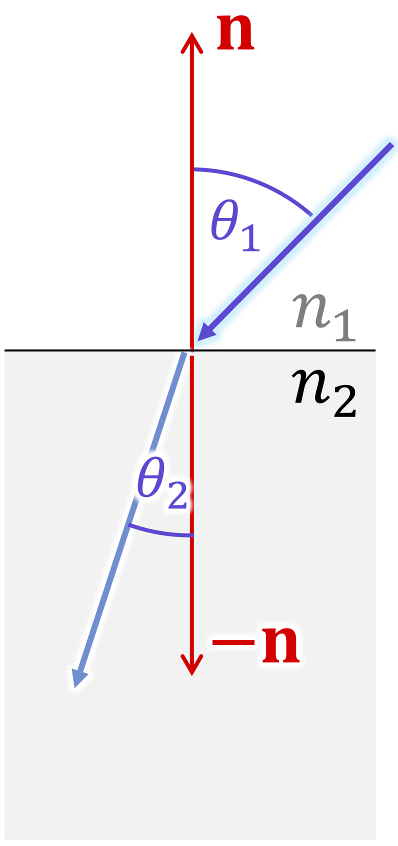
$$\mathbf{c} = \alpha \cdot \mathbf{c}_{front} + (1-\alpha) \cdot \mathbf{c}_{back}$$

- Not commutative! (order matters)
  - unless monochrome

Refraction: Snell's Law

Refraction

- Materials of different "*index of refraction*"
- Light rays change direction at interfaces

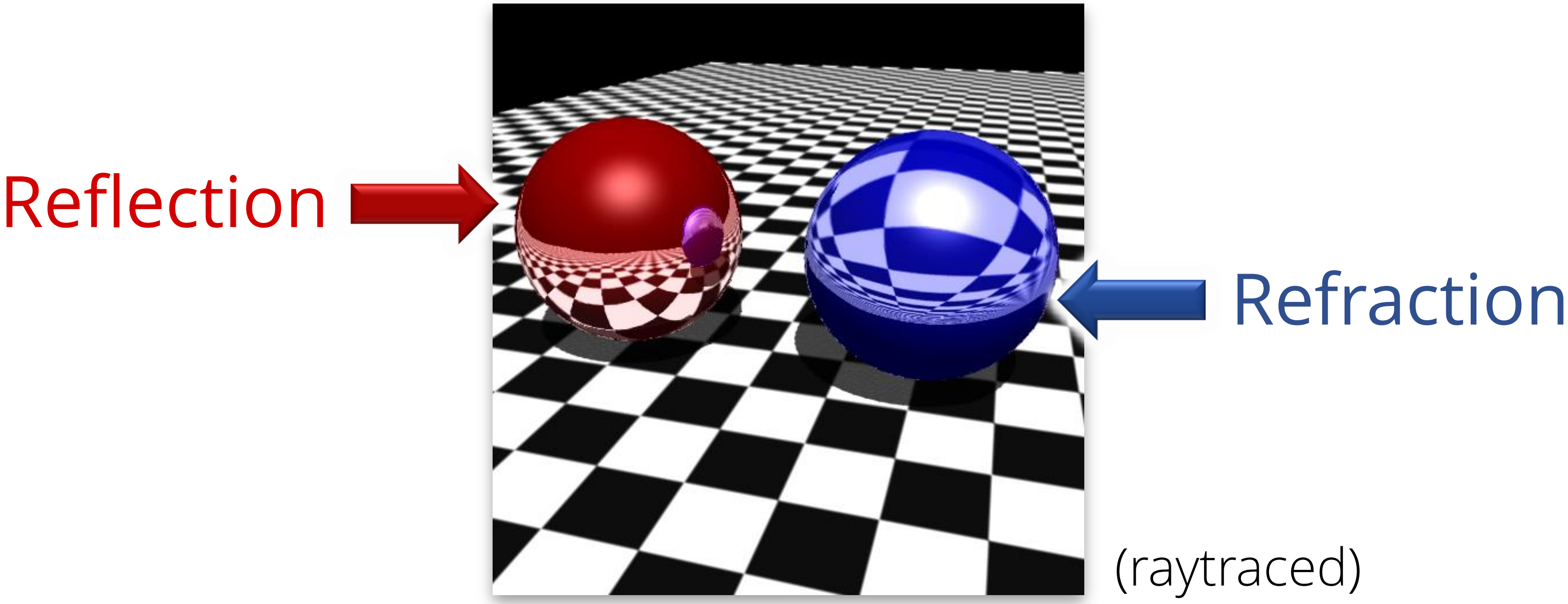


Snell's Law

$$\frac{\sin\theta_1}{\sin\theta_2} = \frac{n_2}{n_1}$$

- n_1, n_2: indices of refraction
  - vacuum: 1.0, air: 1.000293
  - water: 1.33, glass: 1.45-1.6

Refraction

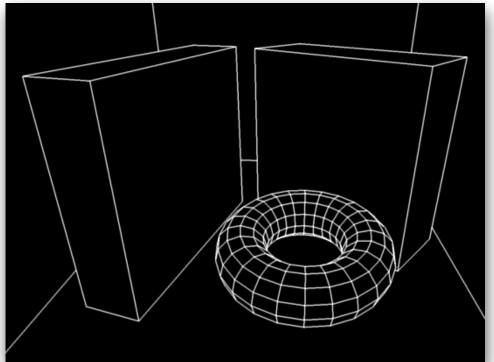


(raytraced)

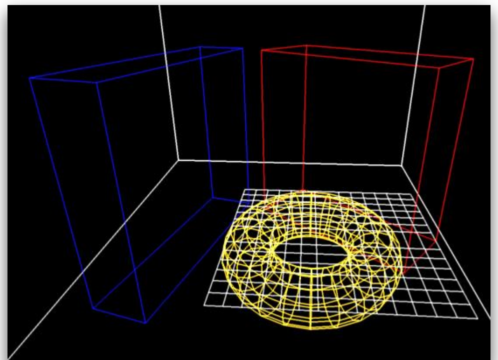
Implementation

- Not a local shading model
- Global algorithms: mostly raytracing
- Various “fake” approximations for local shading

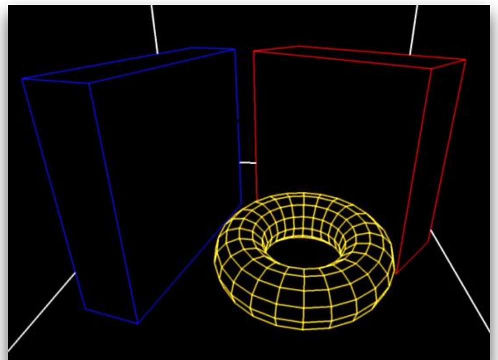
3D Rendering Steps



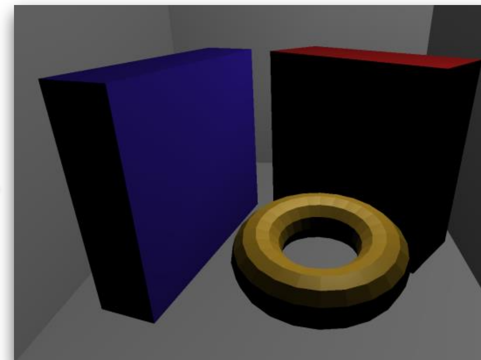
Geometric Model



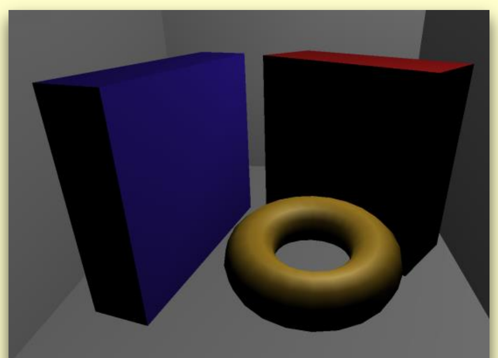
Perspective



Visibility



Local Illumination

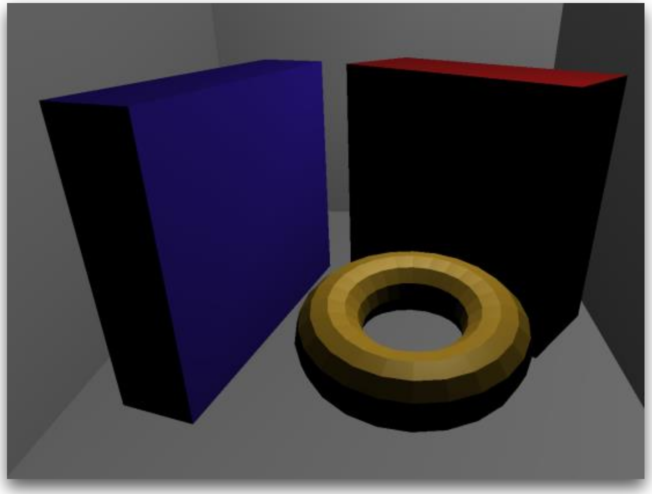


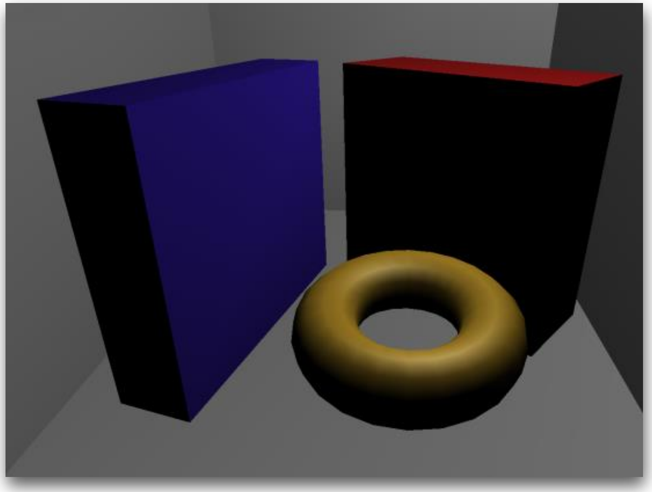
Smooth Shading

Simple Shadows

Global Illumination

Shading Algorithms





CORE

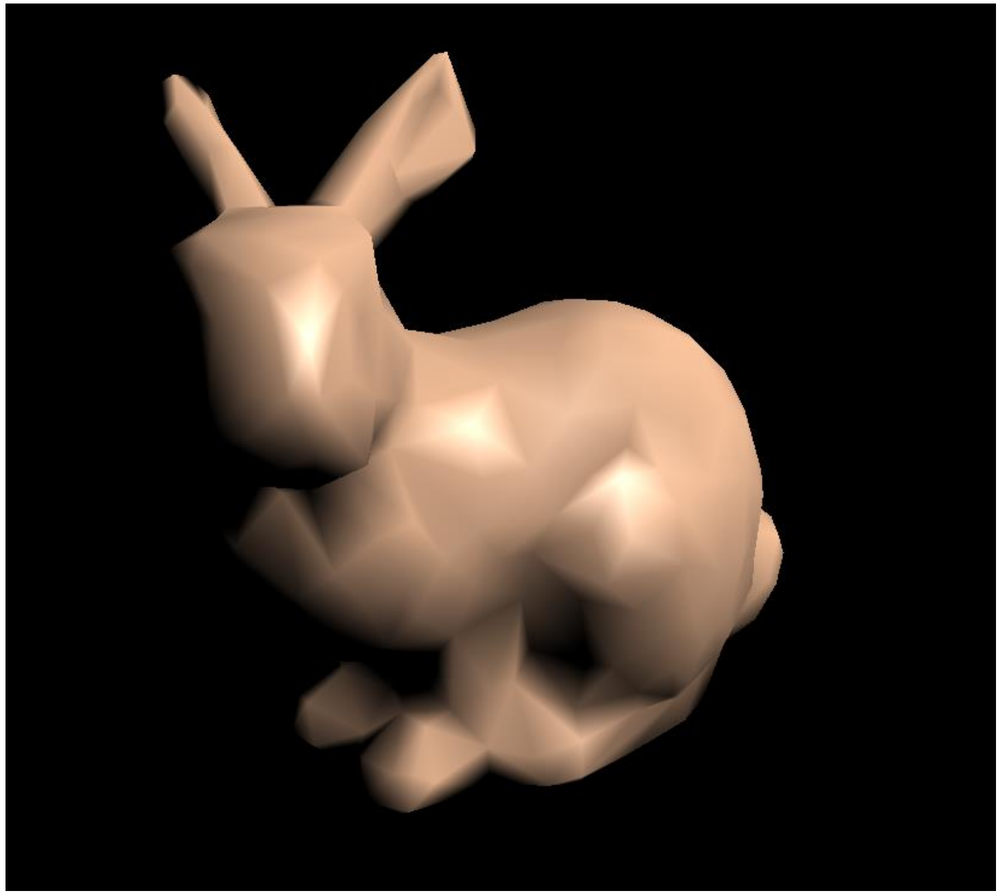
core topics

important

Flat Shading

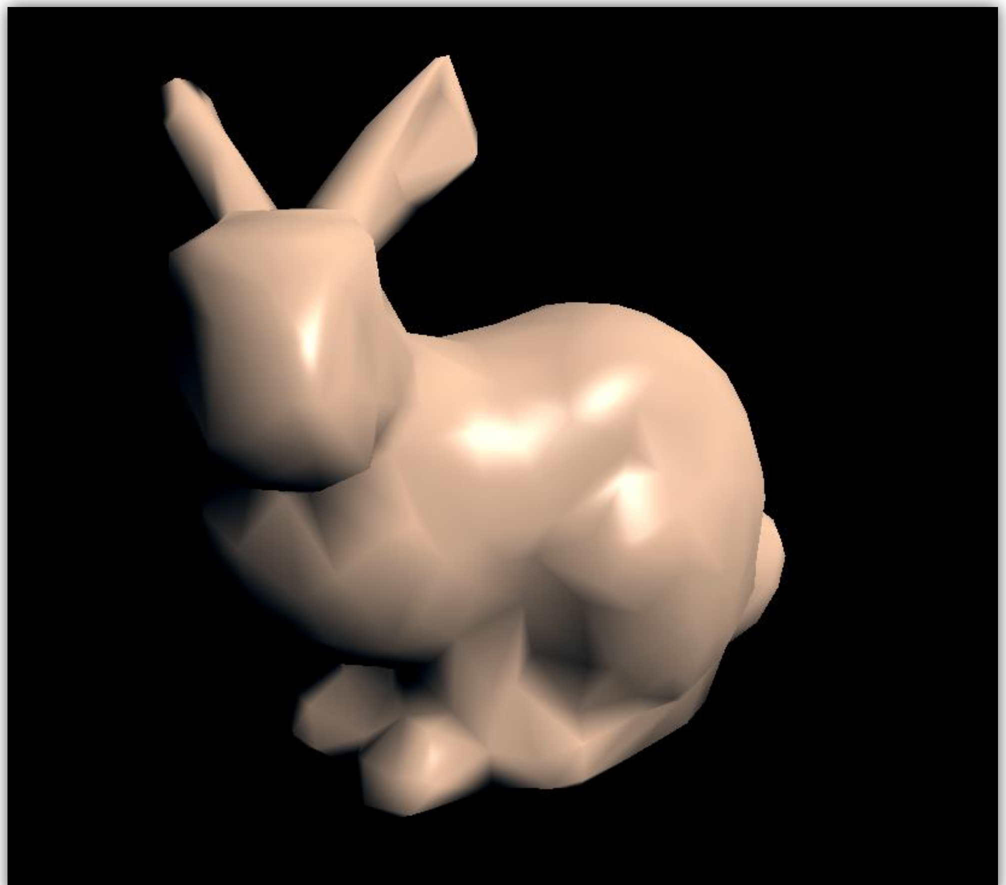
Flat Shading
constant color per triangle

Flat Shading



“Gouraud Shading” Algorithm
compute color at vertices, interpolate color for pixels

Flat Shading



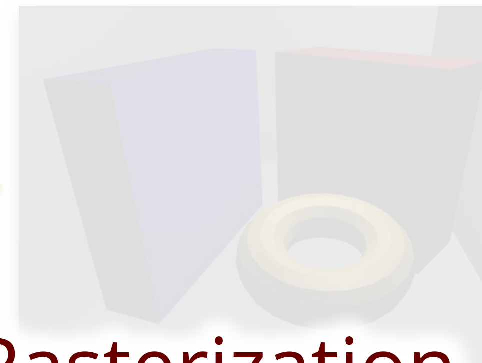
"Phong Shading" Algorithm
interpolate normals for each pixel

3D Rendering Steps

Geometric Model

Perspective

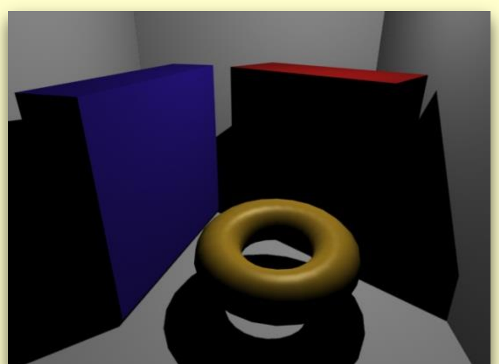
Visibility



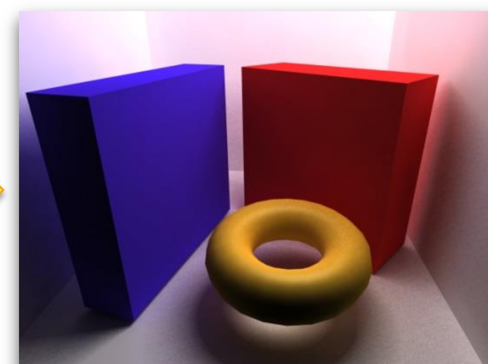
Local Illumination

Next: Advanced Rasterization

Smooth Shading



Simple Shadows



Global Illumination

3D Rendering Steps

Geometric Model

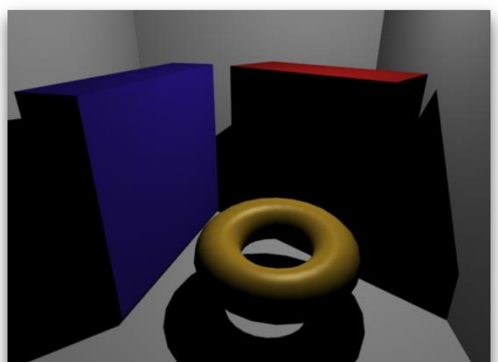
Perspective

Visibility

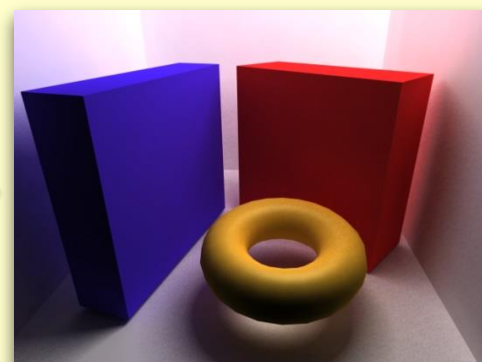
Local Illumination

Global Illum: Keep this for later

Smooth Shading



Simple Shadows



Global Illumination