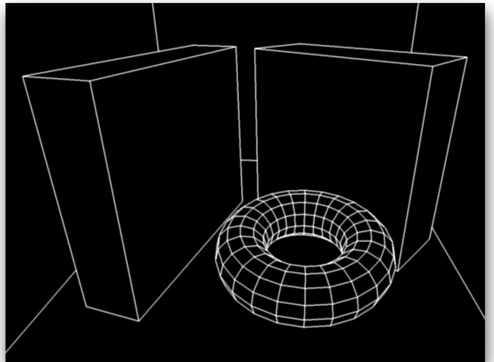
Introduction to Visualization and Computer Graphics

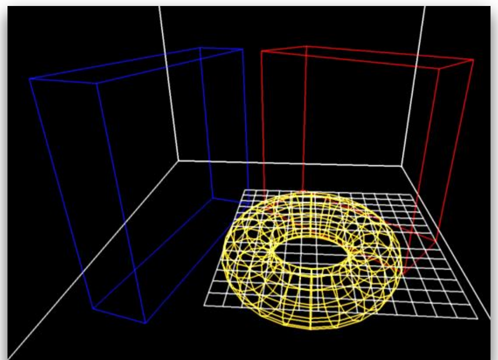
Visibility
Shading

3D Rendering

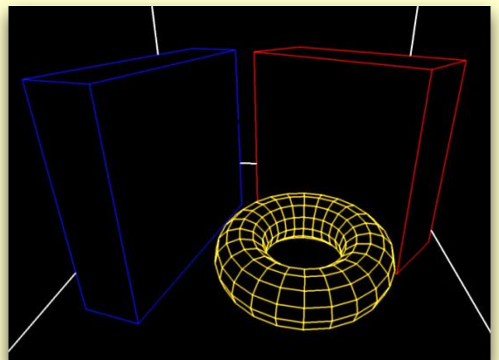


Geometric Model

Color



Perspective



Visibility

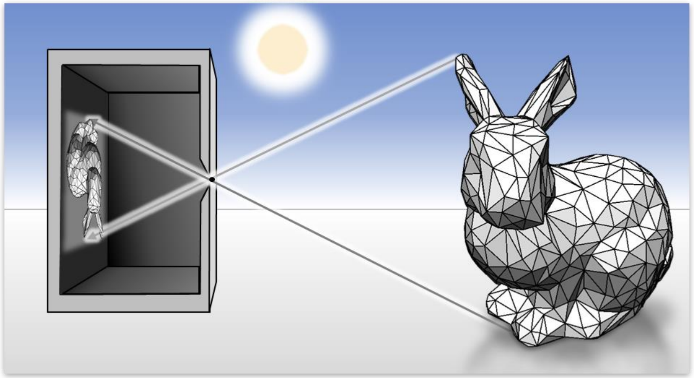
Local Illumination

Smooth Shading

Simple Shadows

Global Illumination

Visibility Algorithms



Two Rendering Pipelines

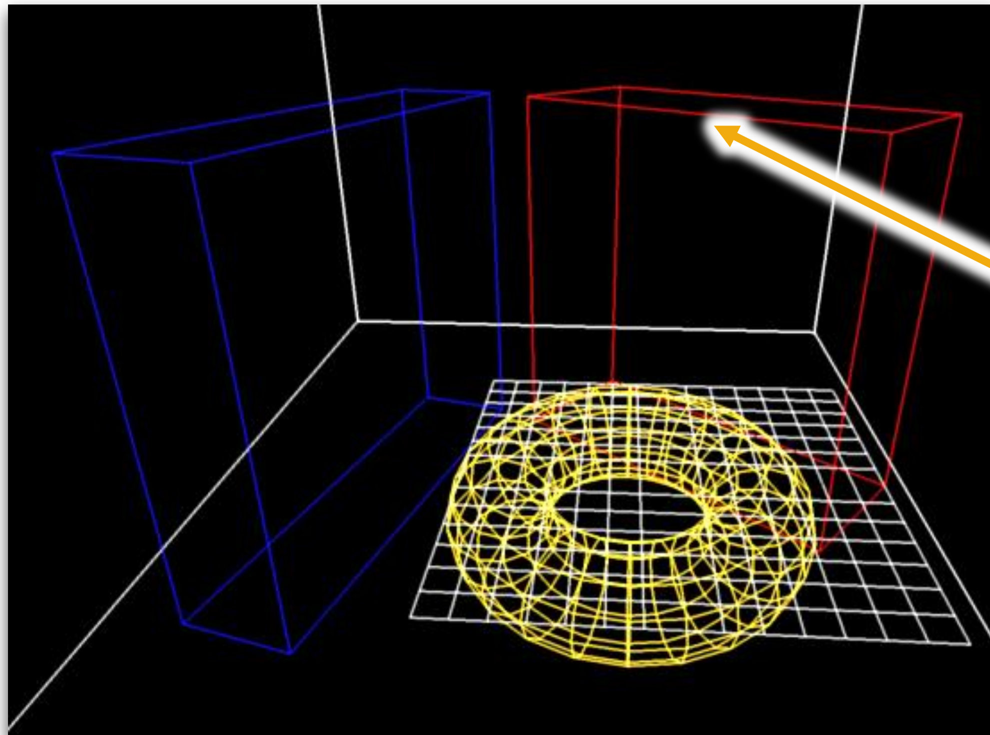
Rasterization

- 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
- ➔ next lecture

Triangle / Polygon Rasterization



After Perspective Projection

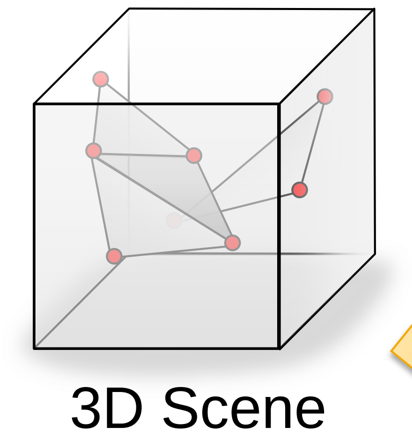
Observations

Straight lines remain *straight*!

Triangles mapped to *triangles*

Polygons to *polyogns*

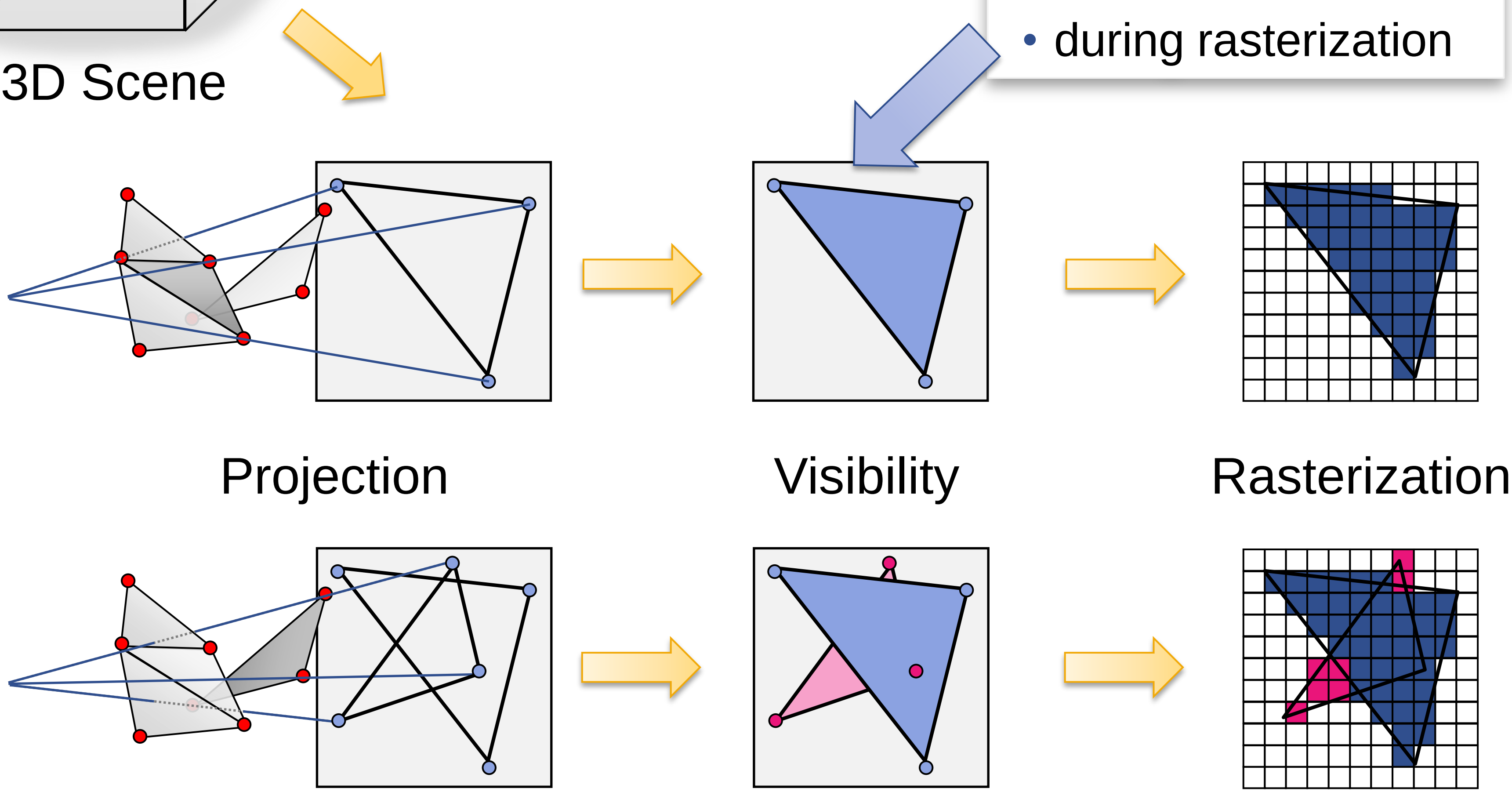
Rasterization



Visiblity

- preprocessing

  or
- during rasterization



Rasterization

Two main algorithms

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

Painter's Algorithm

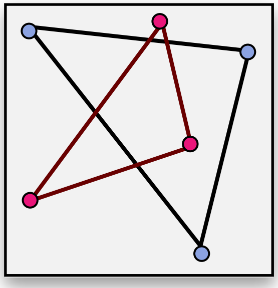
Painter's Algorithm

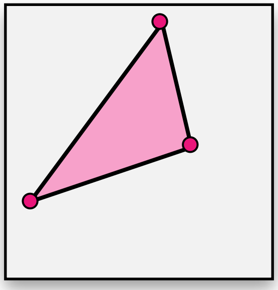
Painters Algorithm

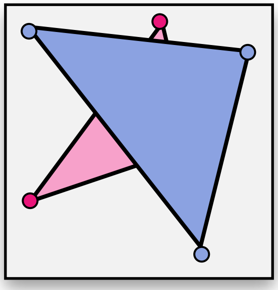
- Sort primitives back-to-front
- Draw with overwrite

Drawbacks

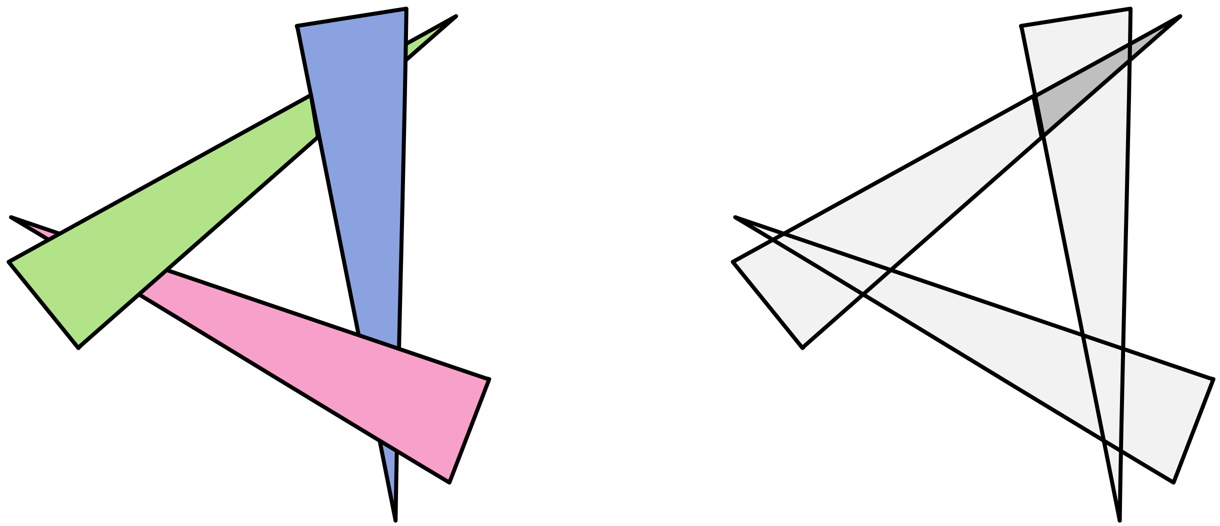
- Slowish
  - $\mathcal{O}(n \cdot \log n)$ for n primitives
  - "Millions per second"
- Wrong
  - Not guaranteed to always work







Counter Example



Correct Algorithm

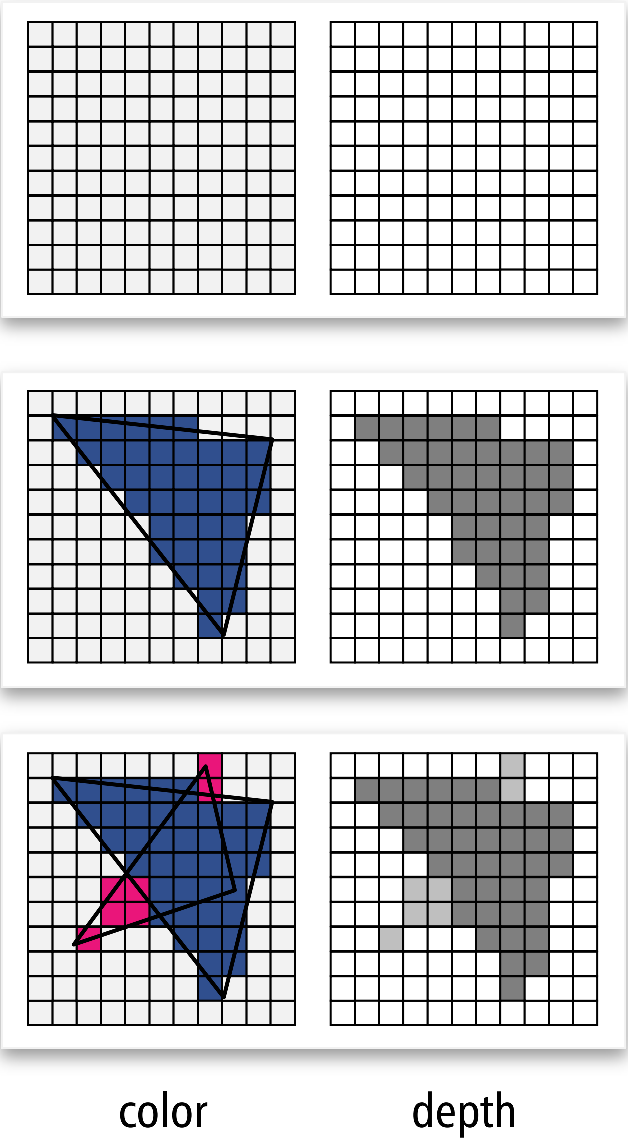
- Need to cut primitives
- Several strategies
  - Notable: BSP Algorithm in Quake
  - Old graphics textbooks list many variants
  - No need for us to go deeper

z-Buffer Algorithm

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 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 in 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

stream surfaces around a delta wing rendered using A-buffers

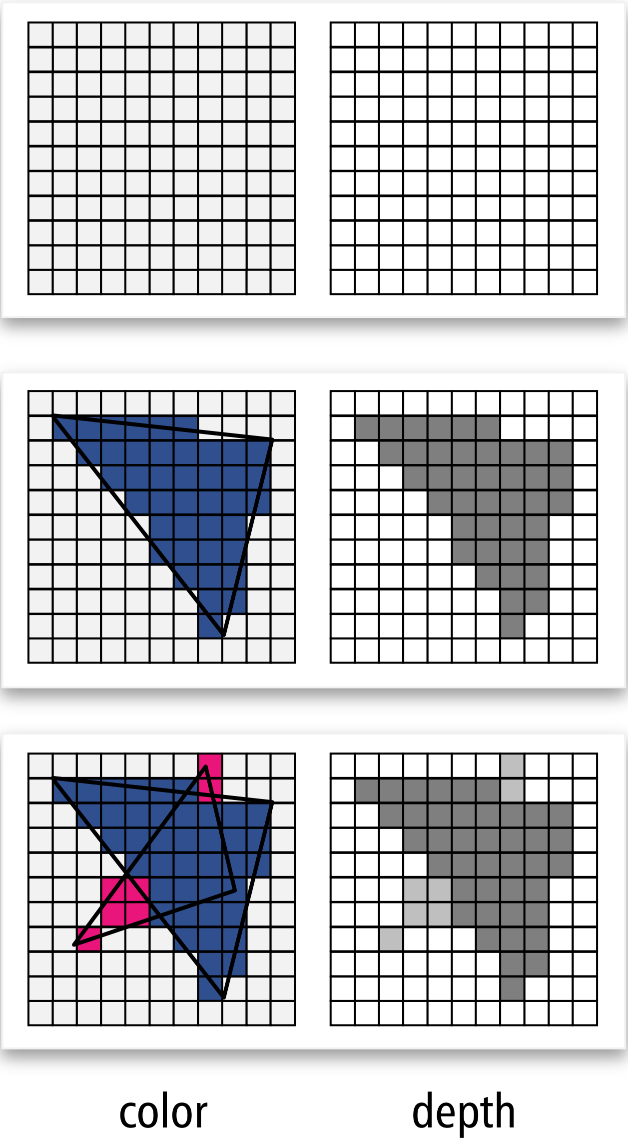
Rasterization and Clipping

Rasterization

How to rasterize Primitives?

Two problems

- Rasterization
- Clipping

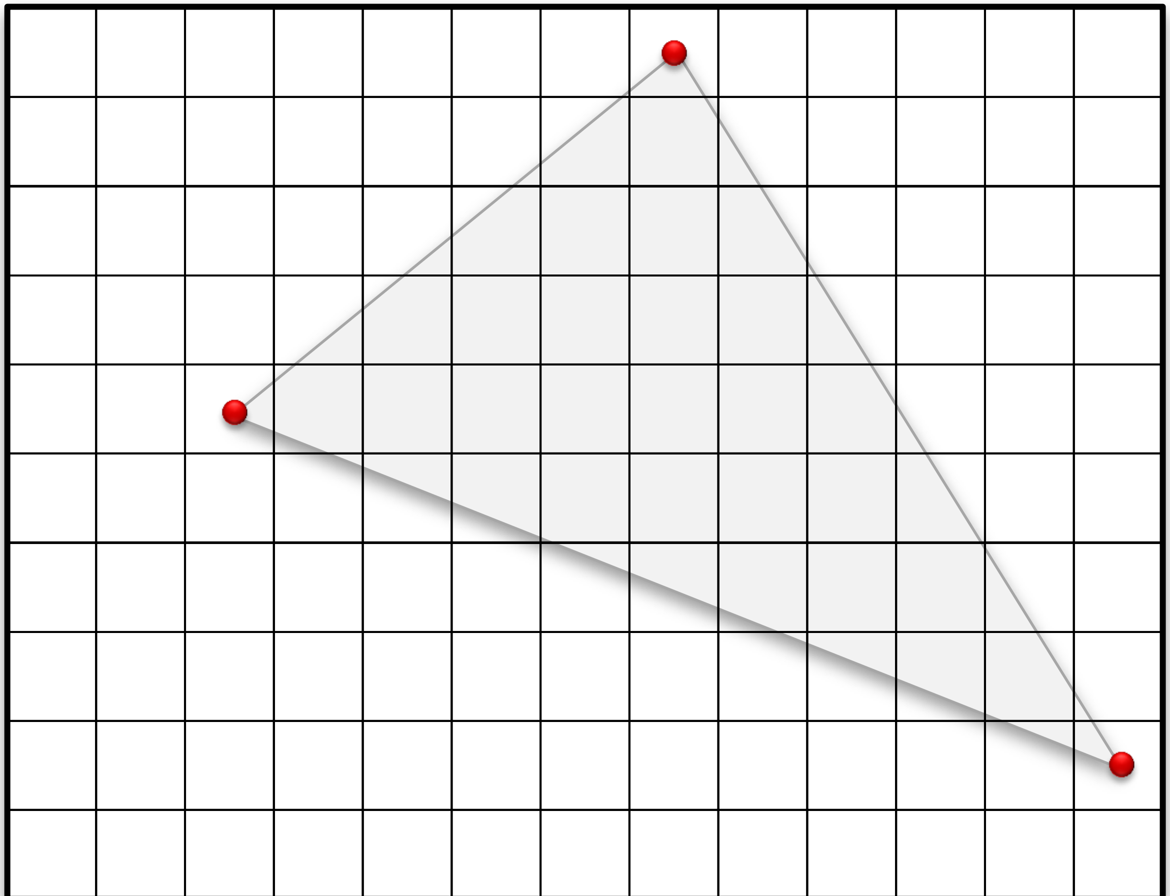


Rasterization

Assumption

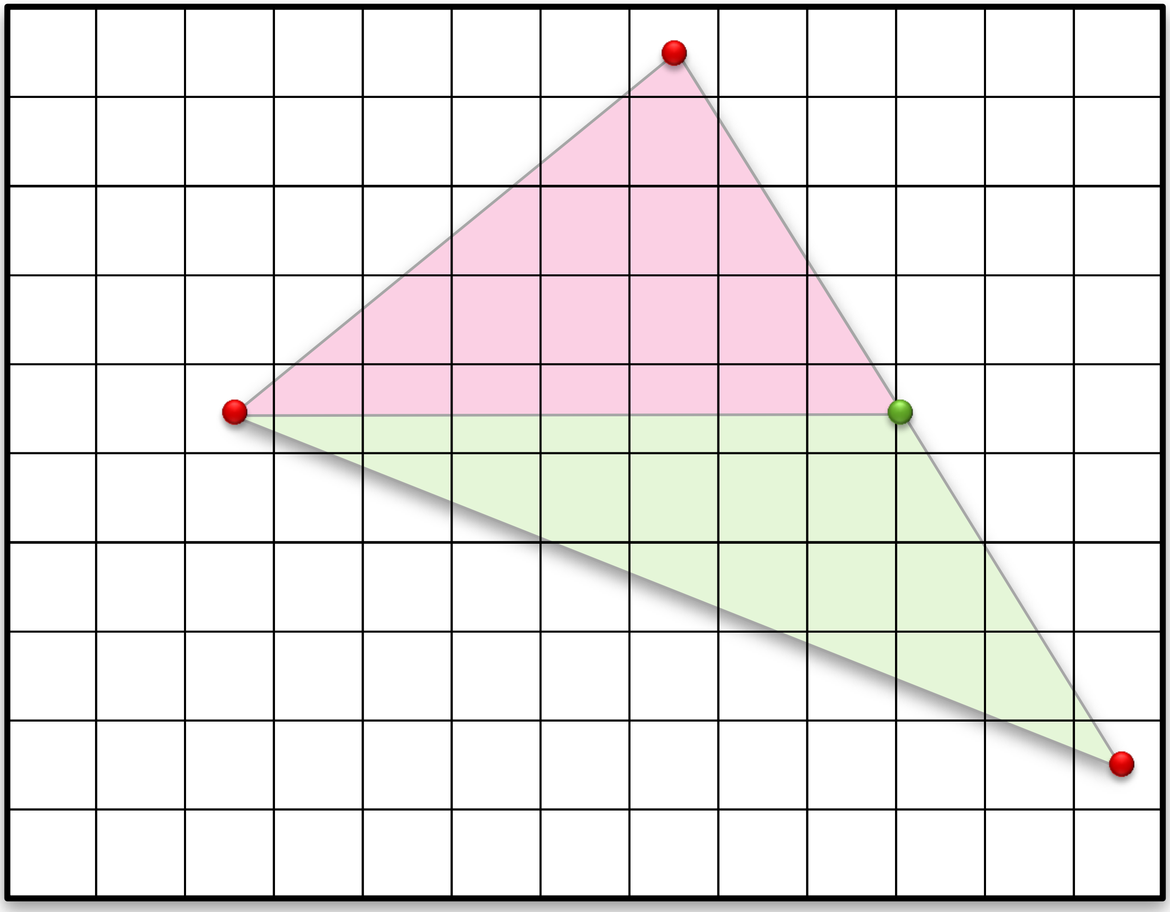
- Triangles only
- Triangle not outside screen
- No clipping required

Triangle Rasterization



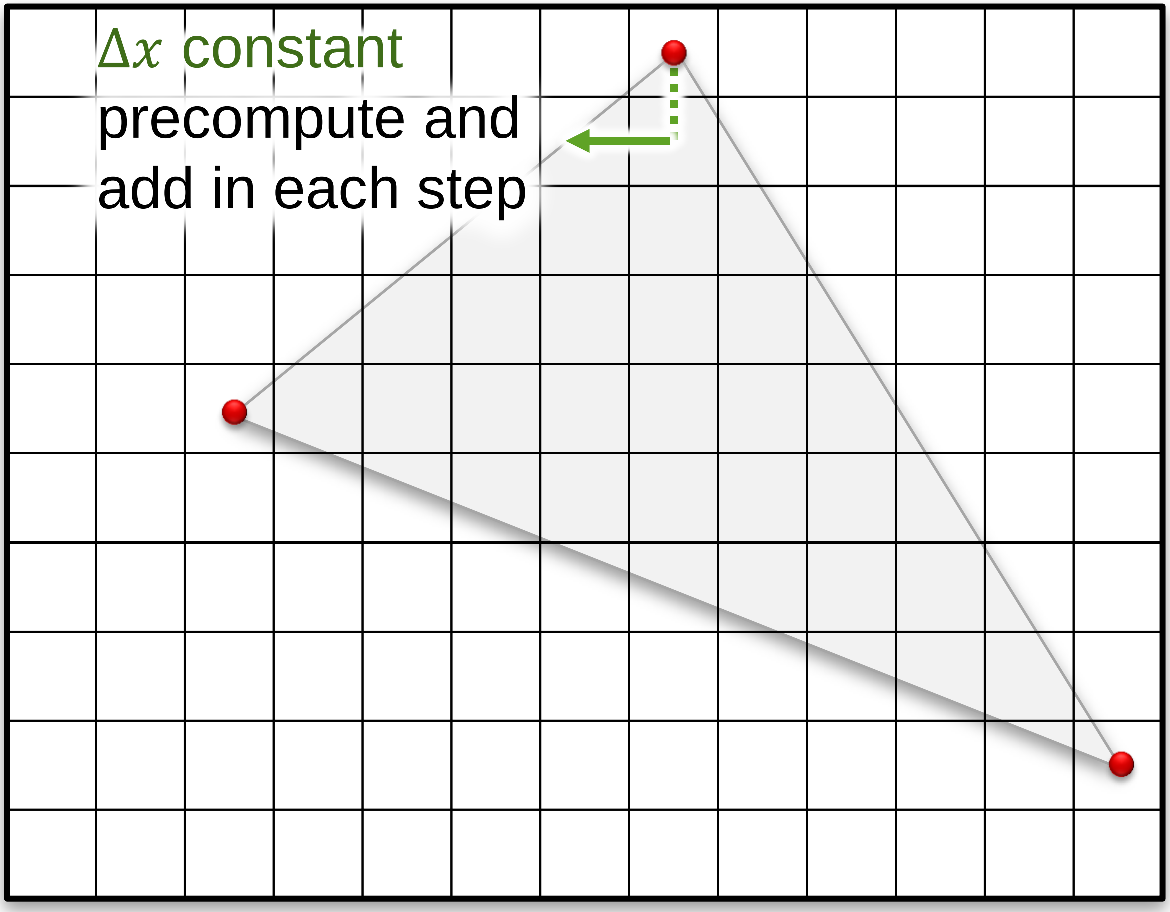
Several Algorithms...

Triangle Rasterization



Example: two slabs

Triangle Rasterization



Incremental rasterization

Incremental Rasterization

Precompute steps in x, y-direction

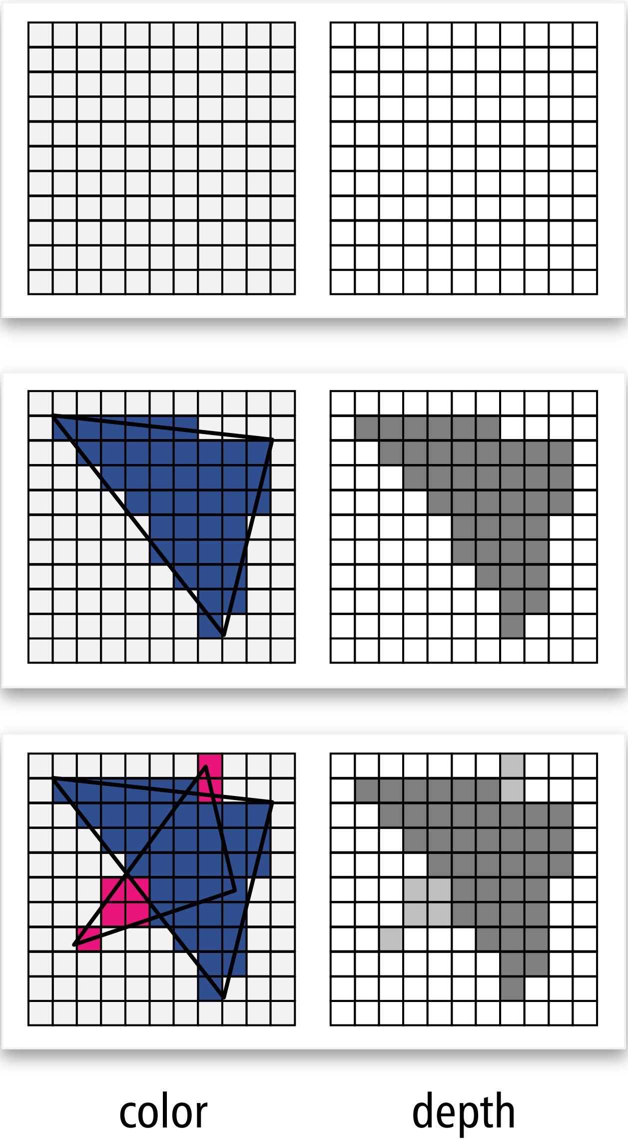
- For boundary lines
- For linear interpolation within triangle
  - Colors
  - Texture coordinates (more later)
- Inner loop
  - Only one addition (“DDA” algorithm)
  - Floating point value
  - Strategies
    - Fixed-point arithmetics
    - Bresenham / midpoint algorithm
      (requires if; problematic on modern GPUs)

Rasterization

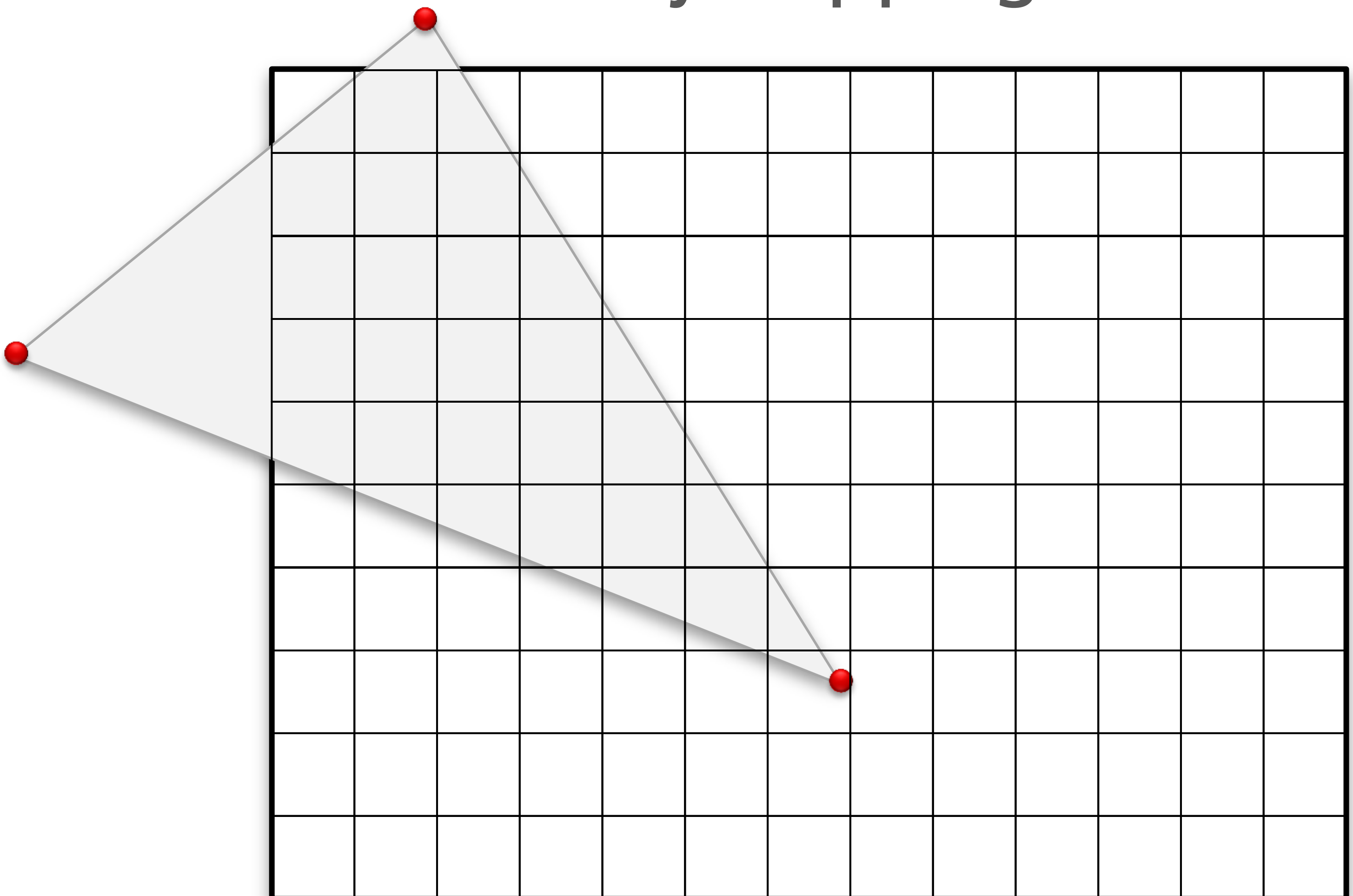
How to rasterize Primitives?

Two problems

- Rasterization
- Clipping



Why Clipping?



Crashes – write to off-screen memory!

Clipping Strategies

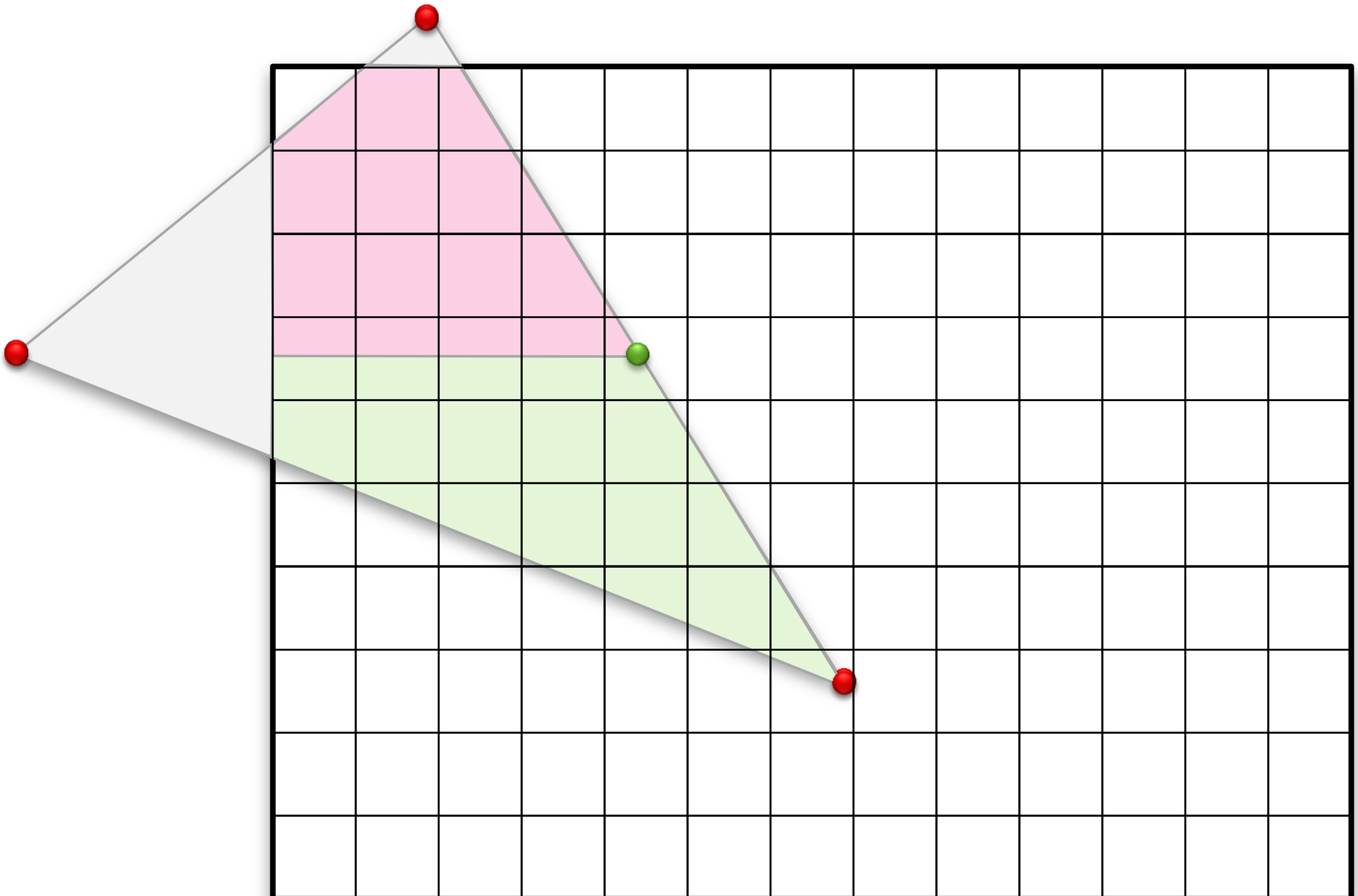
Pixel Rejection

- "if (x,y $\notin$ screen) continue;"
  - Can be arbitrarily slow (large triangles)
  - Nope. Not a good idea.

Screen space clipping

- Modify rasterizer to jump to visible pixels
- Efficient
- Still problems with when crossing camera plane ($w = 0$) $\Rightarrow$ a semi-good idea

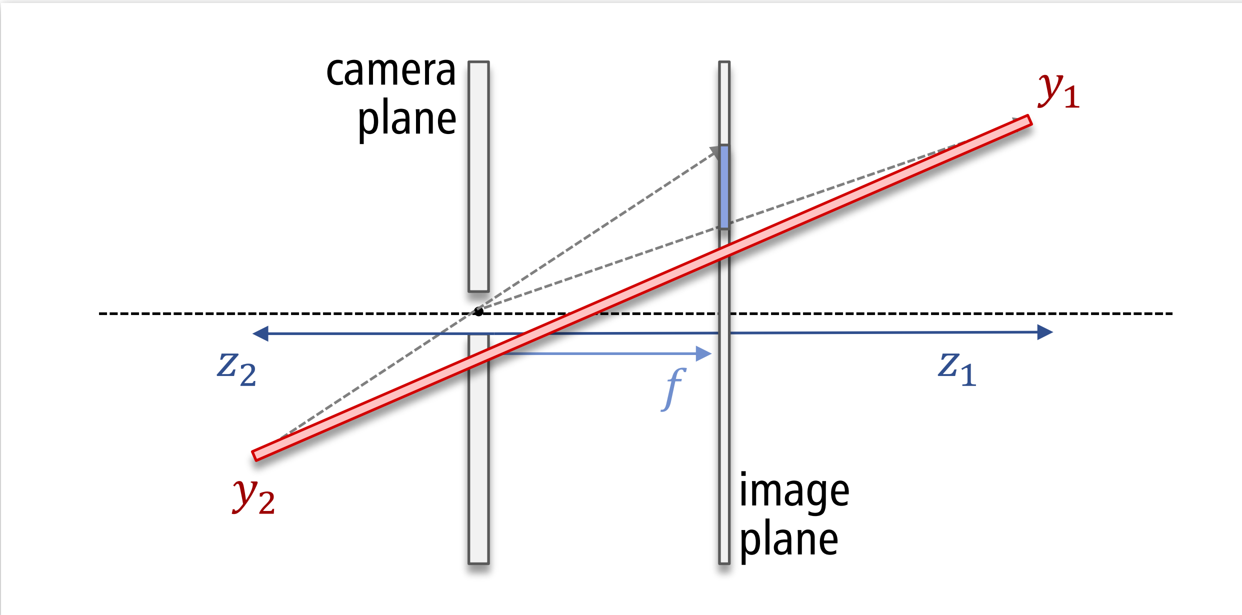
Smart Slab Renderer



Does not crash, optimal complexity

- $O(k)$ for k output fragments

Problem

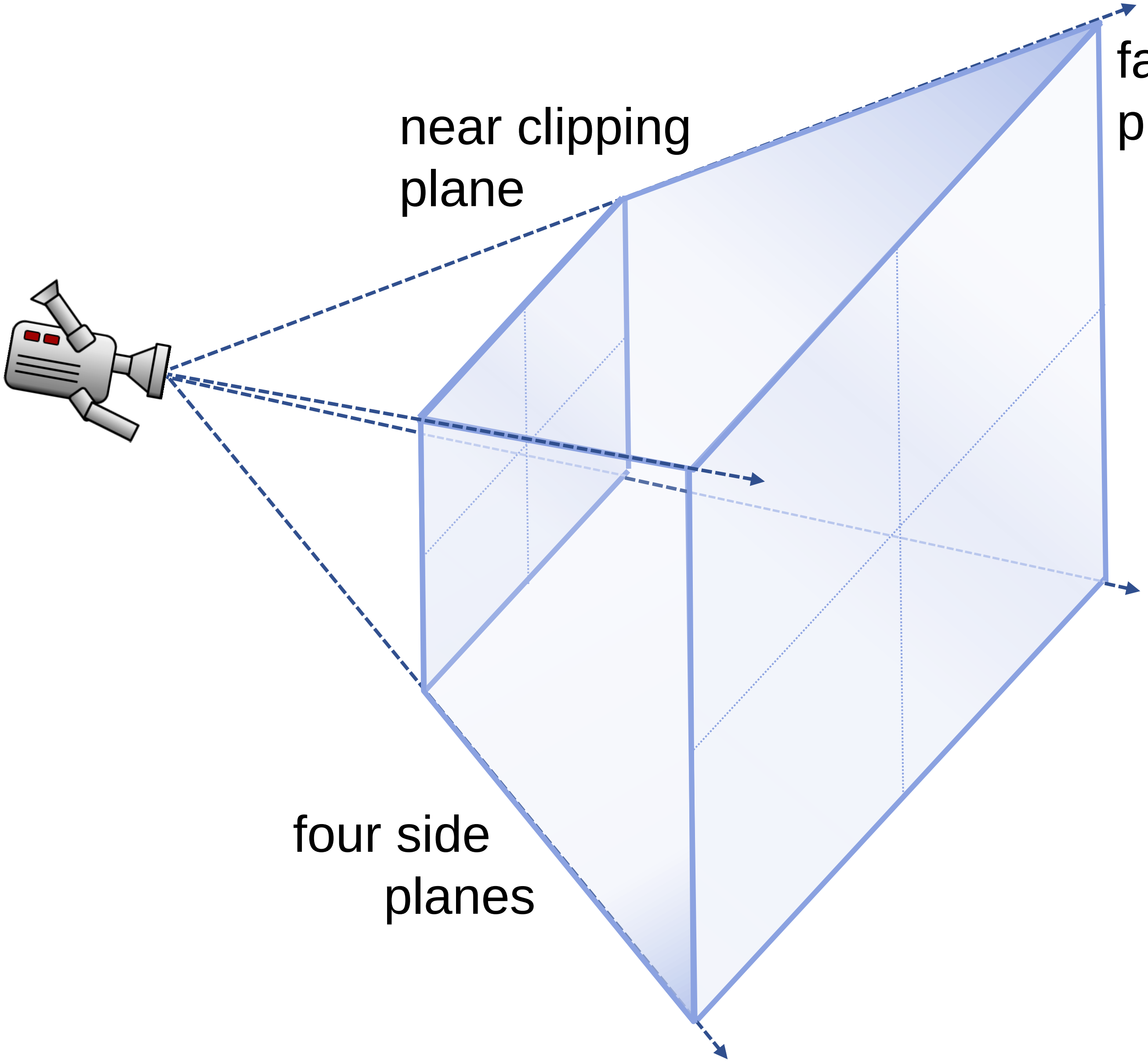


$$y' = f\frac{y}{z}$$

Problem:

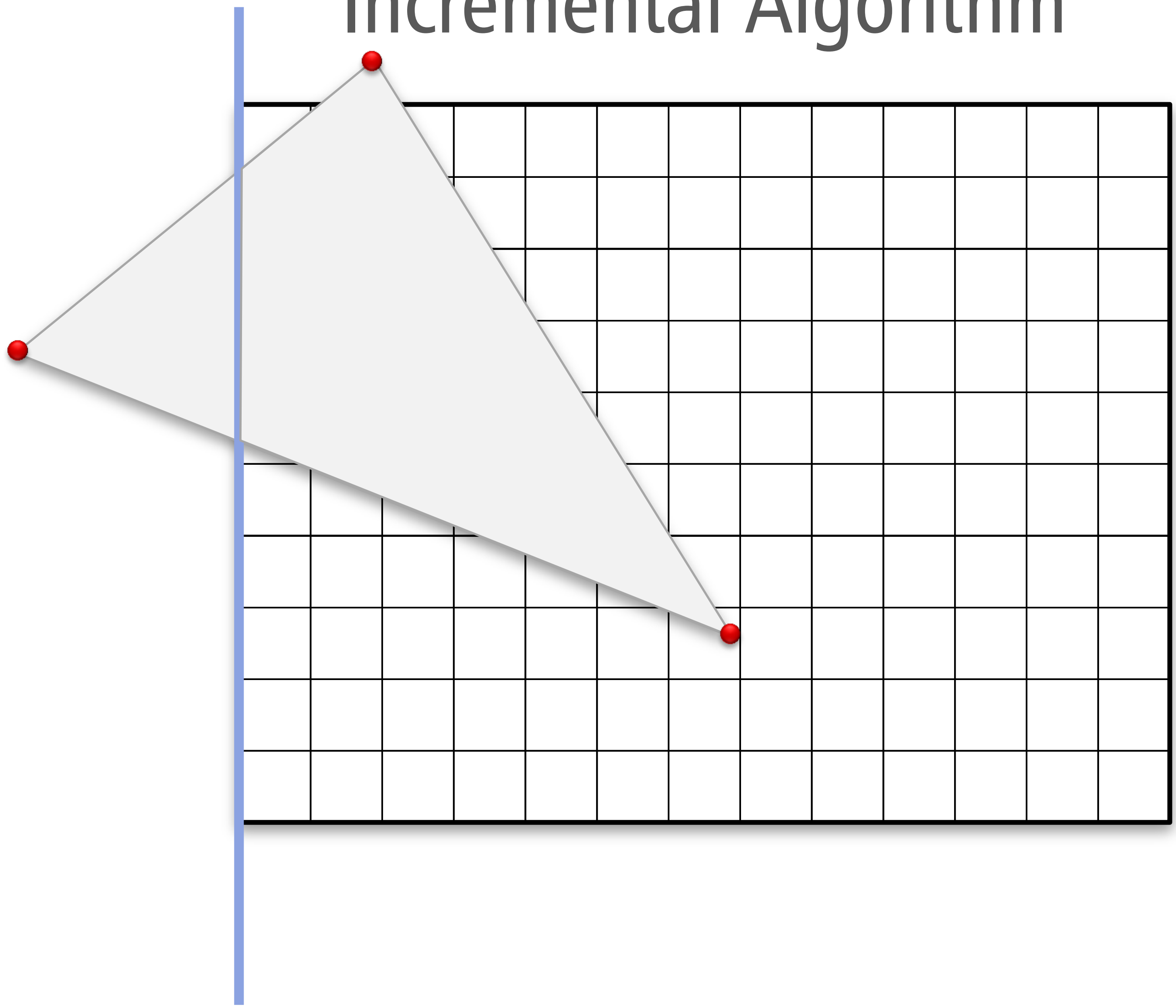
- Triangles crossing camera plane!
  - Wrong results
- Need object space clipping

View Frustum Clipping

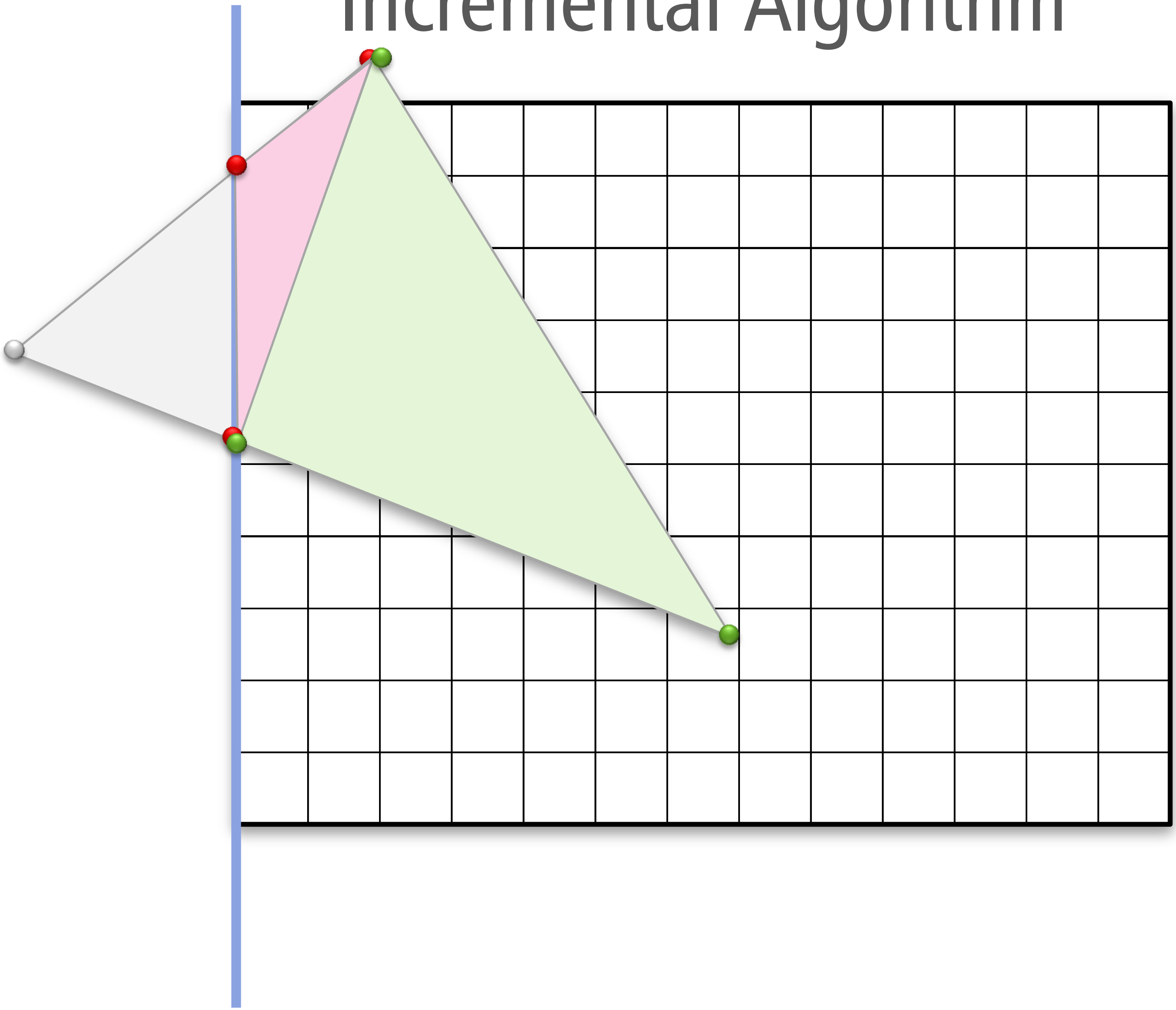


six planes
clip triangles
against all
six planes

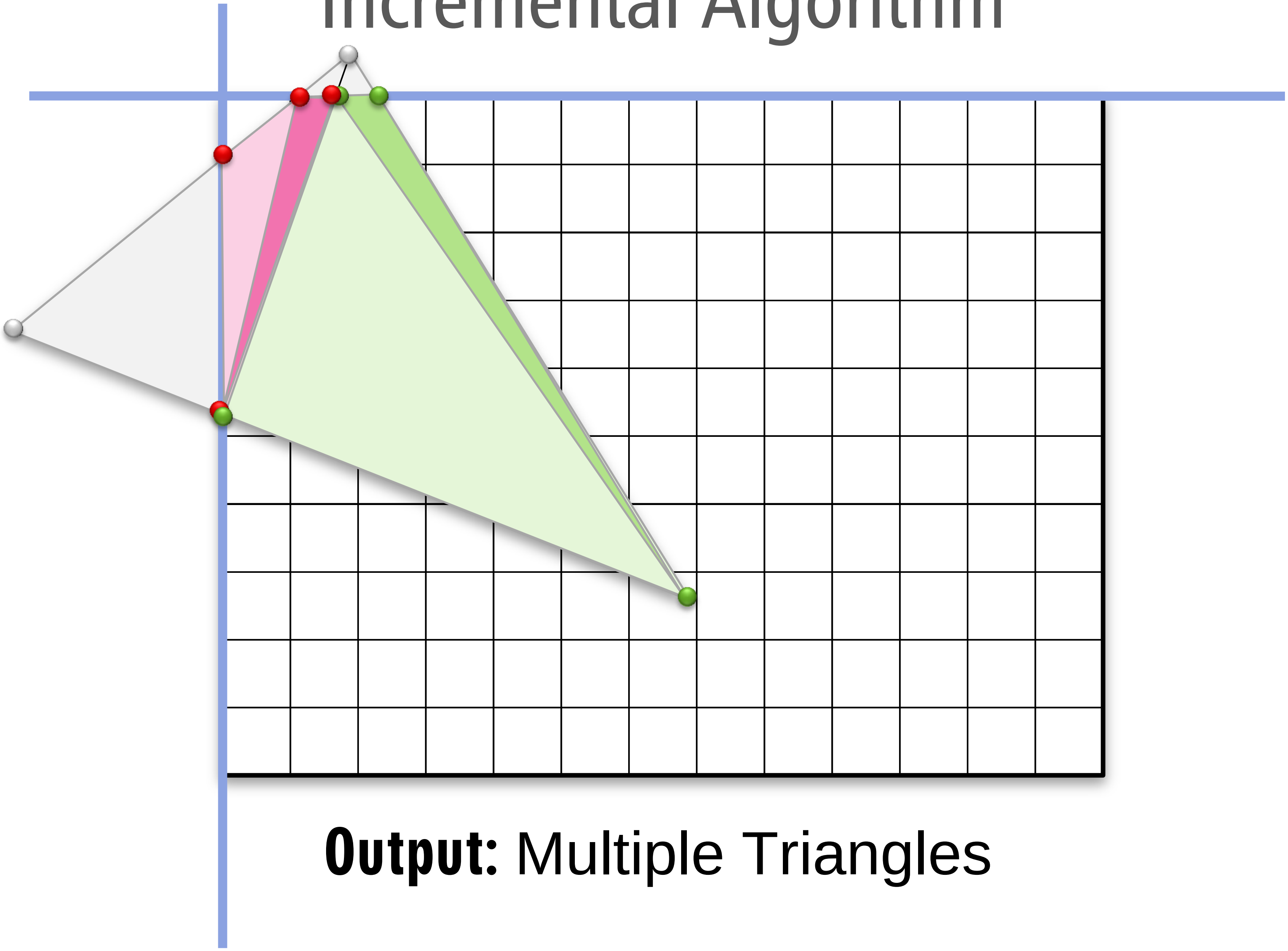
Incremental Algorithm



Incremental Algorithm



Incremental Algorithm



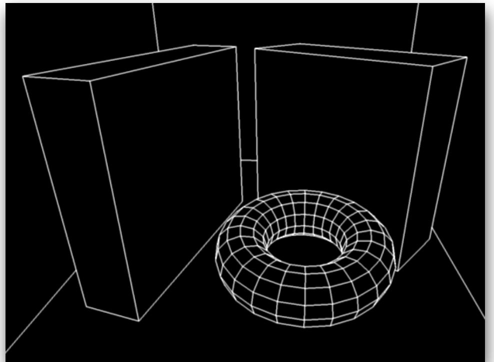
Output: Multiple Triangles

Further Optimization

View Frustum Culling

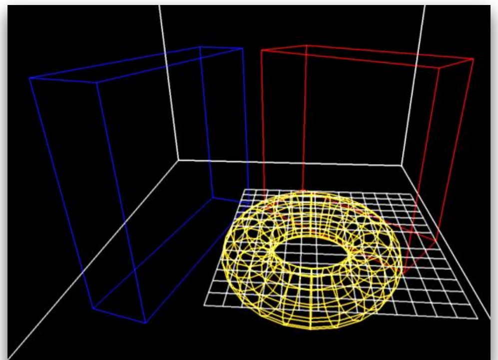
- Complex shapes (whole bunnies)
- Coarse bounding volume (superset)
  - Cube, Sphere
  - Often: Axis-aligned bounding box
- Reject all triangles inside if bounding volume outside view frustrum

3D Rendering

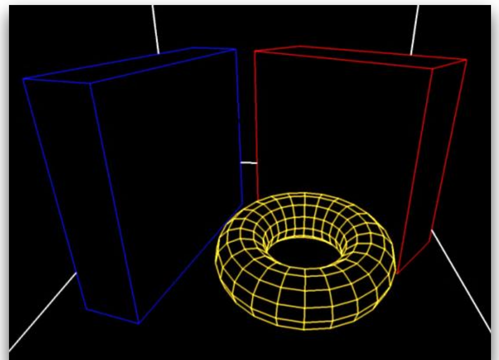


Geometric Model

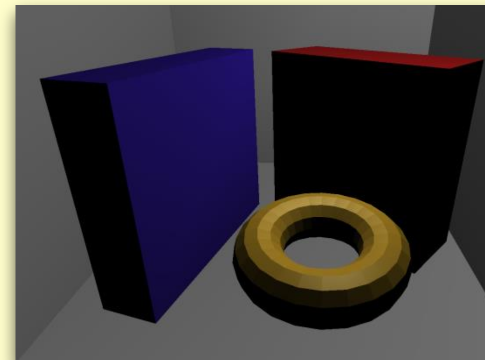
Color



Perspective



Visibility



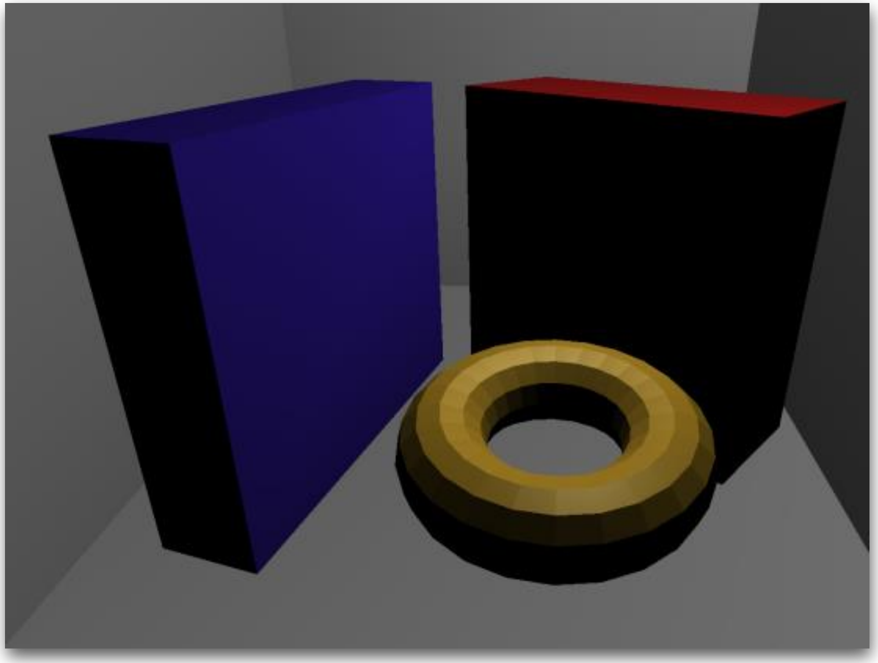
Local Illumination

Smooth Shading

Simple Shadows

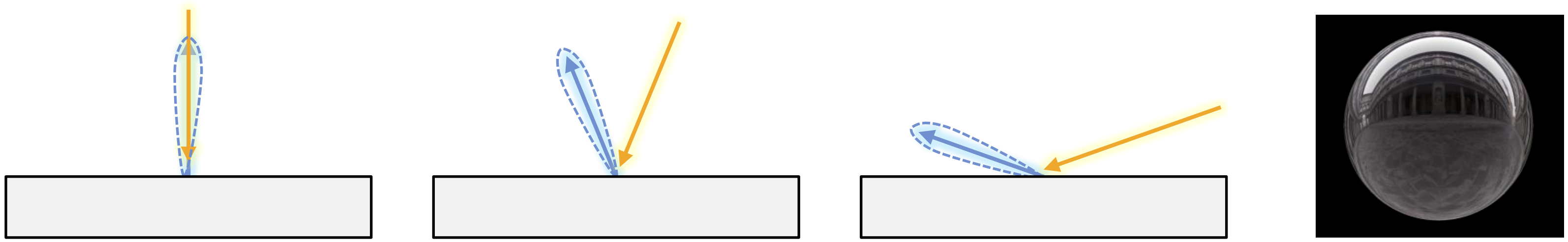
Global Illumination

Shading Models

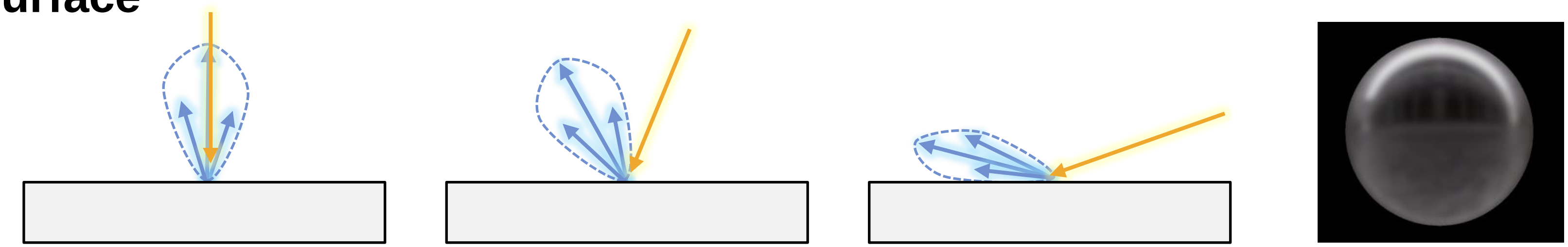


Reflectance Models

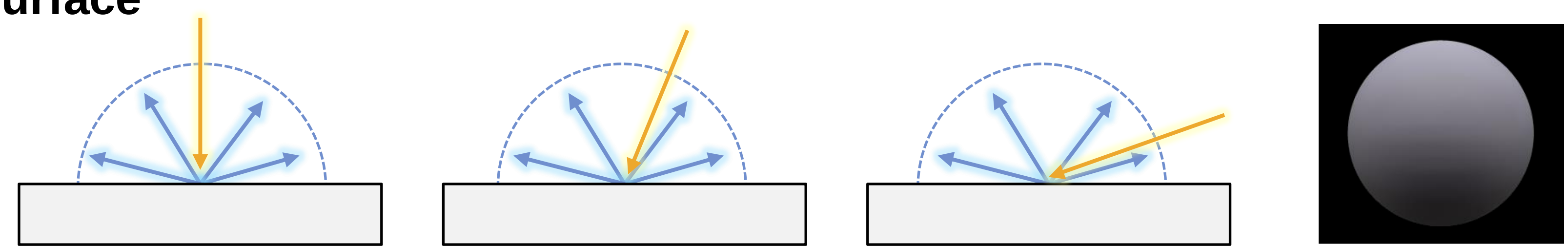
mirror



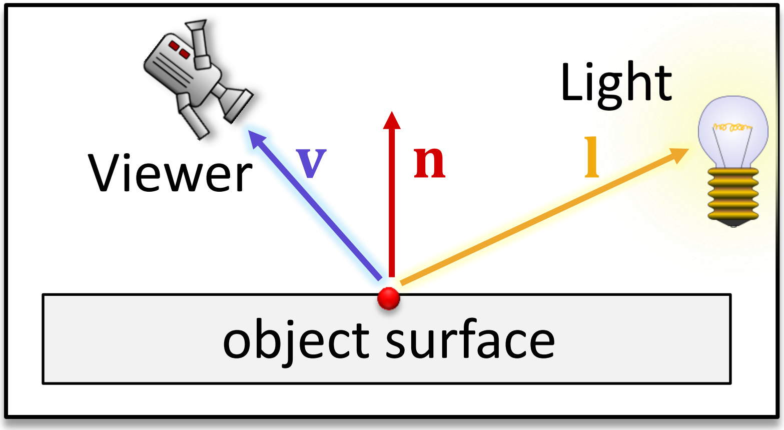
glossy surface



diffuse surface



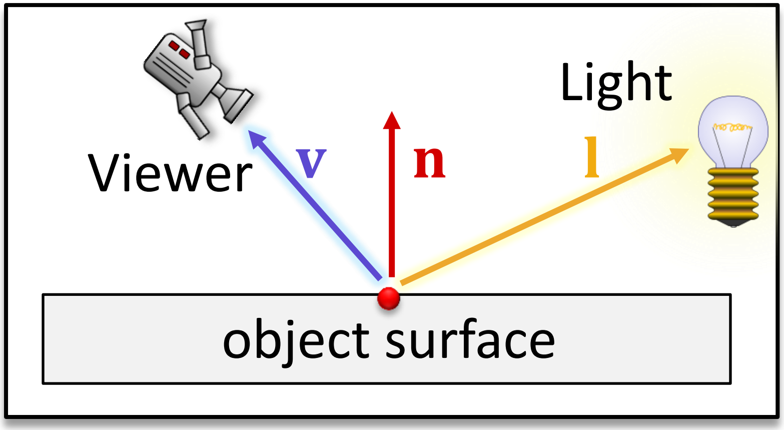
Interaction with Surfaces



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



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 "$\circ$" 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

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

Shading Effects

Shading effects

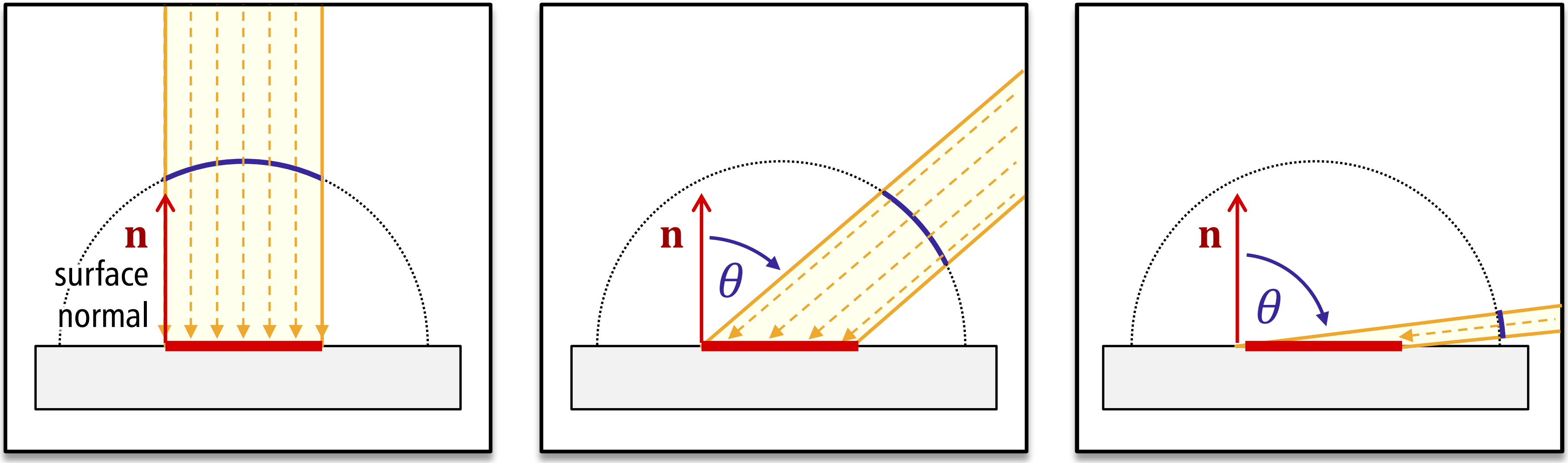
- Diffuse reflection
- “Ambient reflection”
- Perfect mirrors
- Glossy reflection
  - Phong / Blinn-Phong
  - (Cook Torrance)
- Transparency & refraction

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

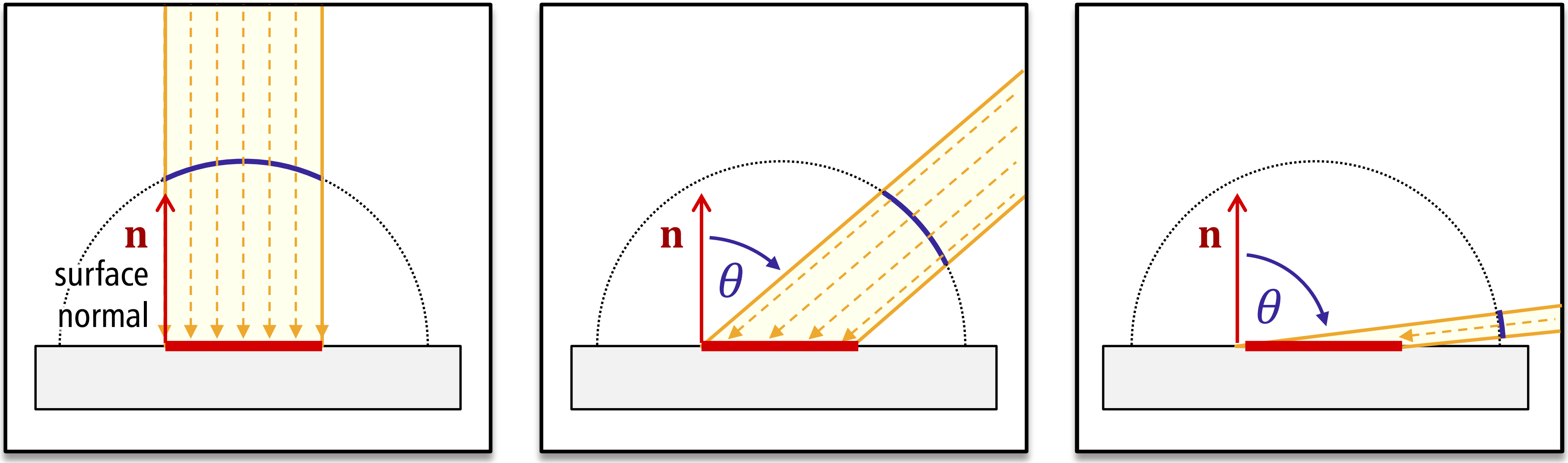
c – intensity (scalar)

$\mathbf{c}$ – 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

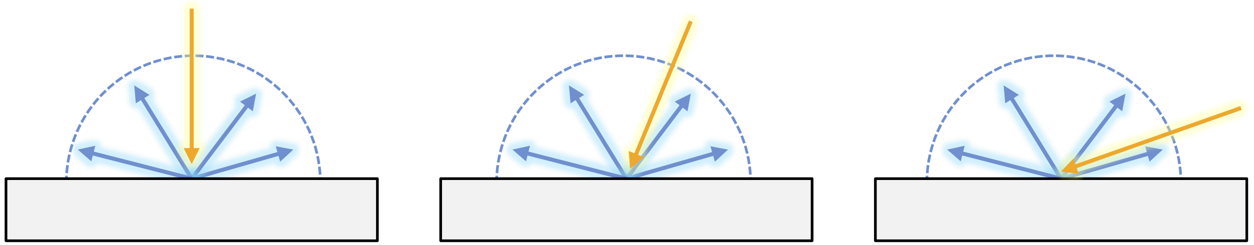
c – intensity (scalar)
$\mathbf{c}$ – color (RGB, $\mathbb{R}^3$)
$\mathbf{c}_r$ – surface color (RGB)
$\mathbf{c}_l$ – light color (RGB)

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

light color
surface color

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

Diffuse Reflection

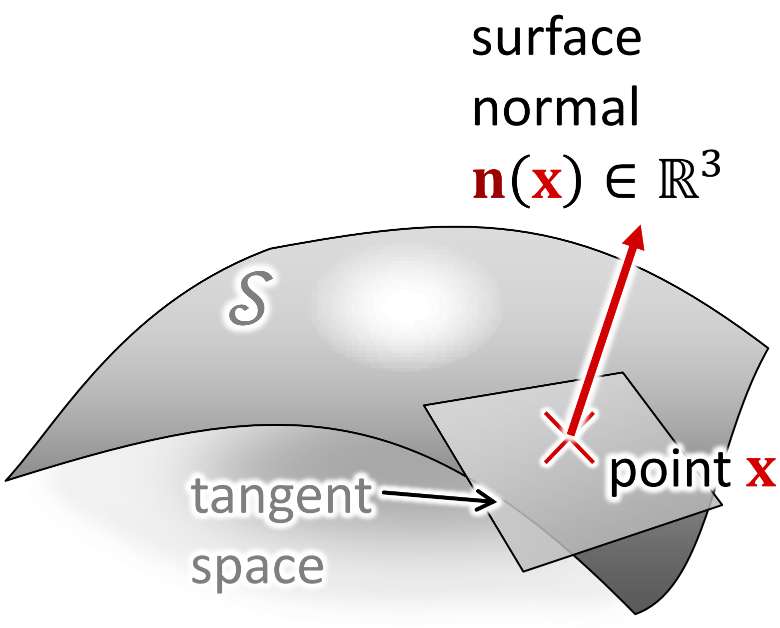
Diffuse Reflection

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

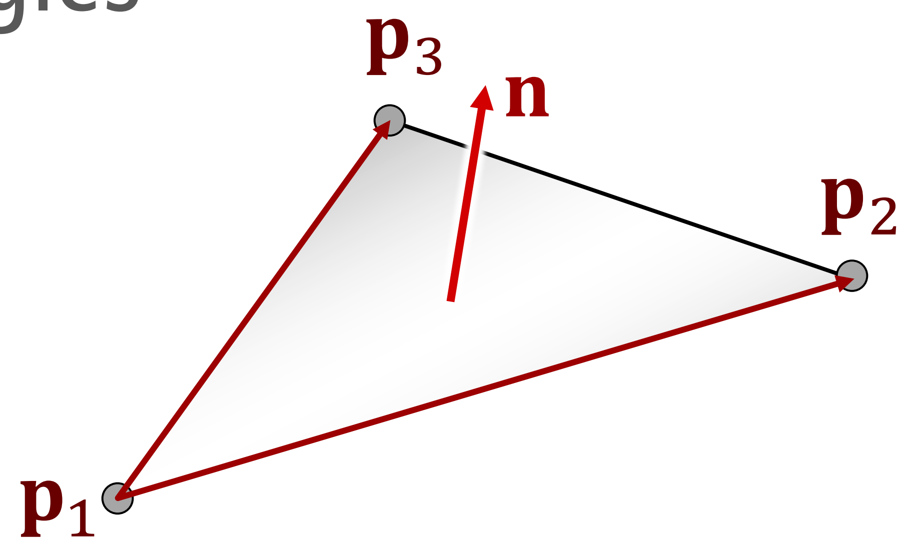
Surface Normal?

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

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

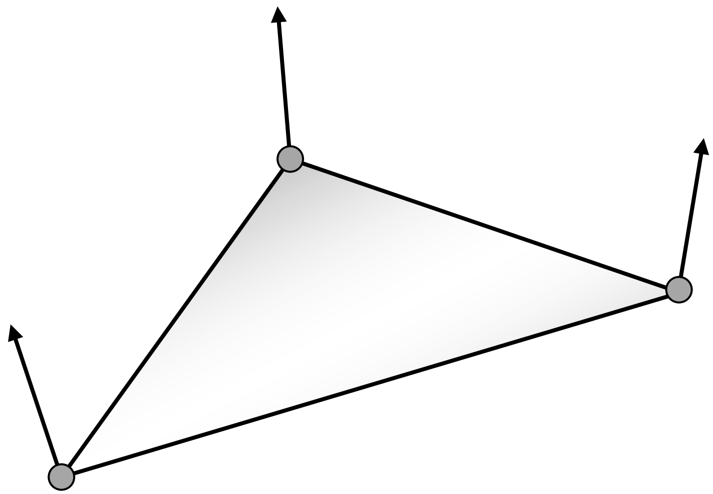
  - Tangent space: the plane itself

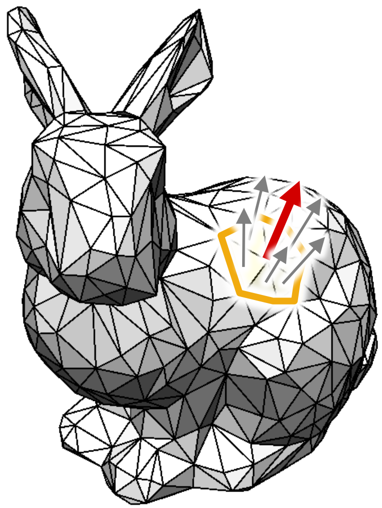
- Normal vector

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

  - 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

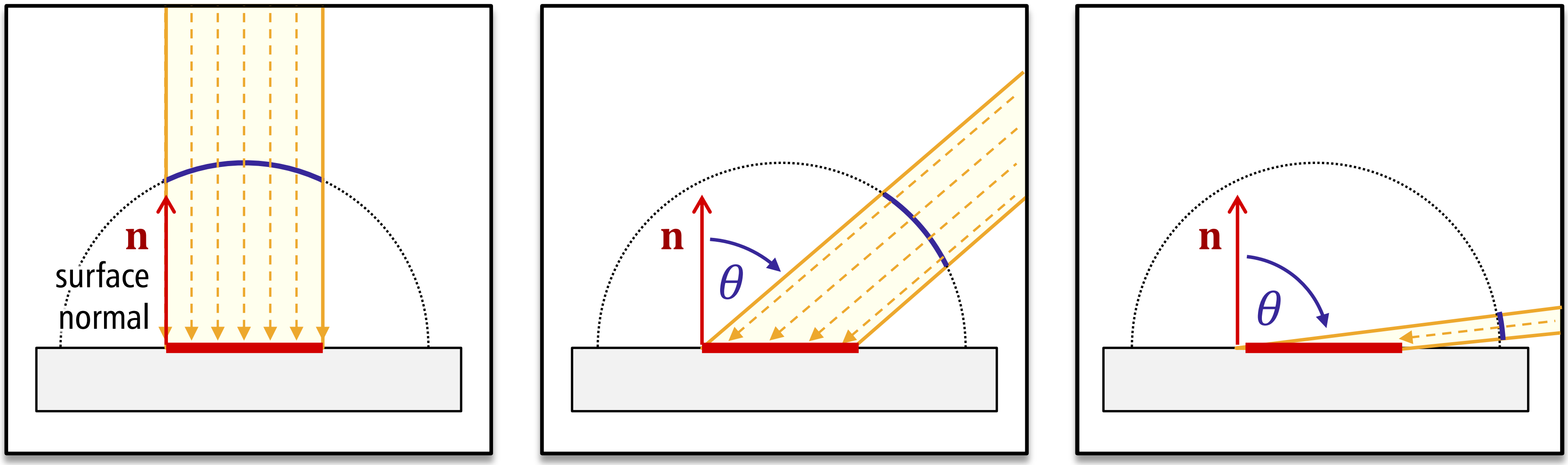




Smooth Triangle Meshes

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

Lambertian Surfaces



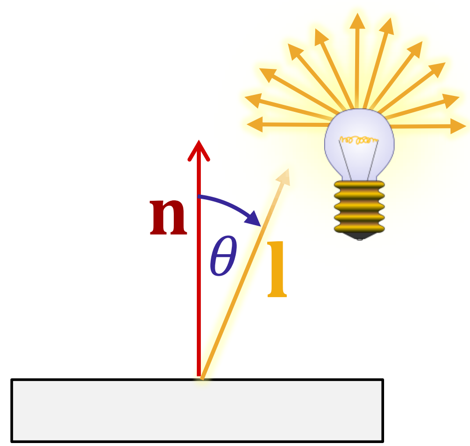
Equation

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

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”

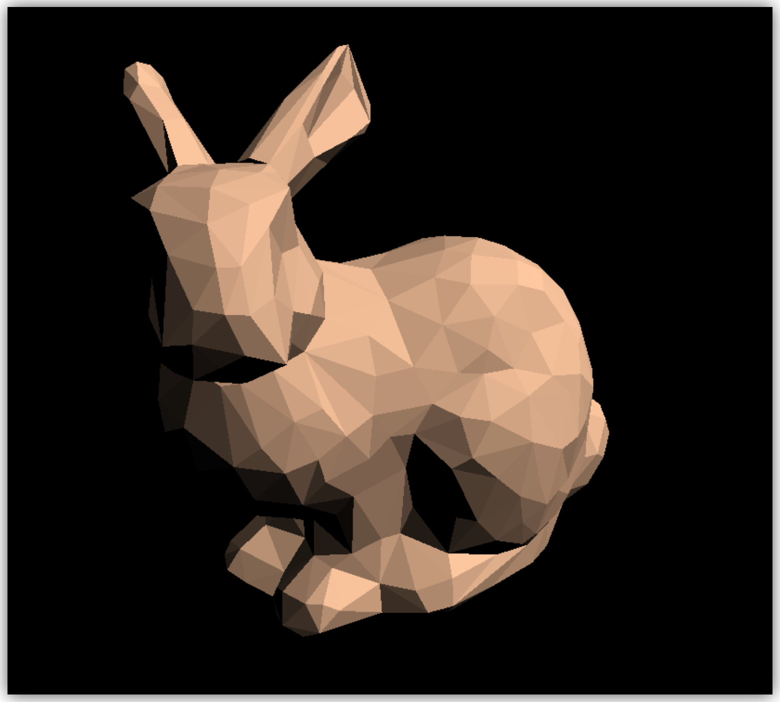
Problem

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

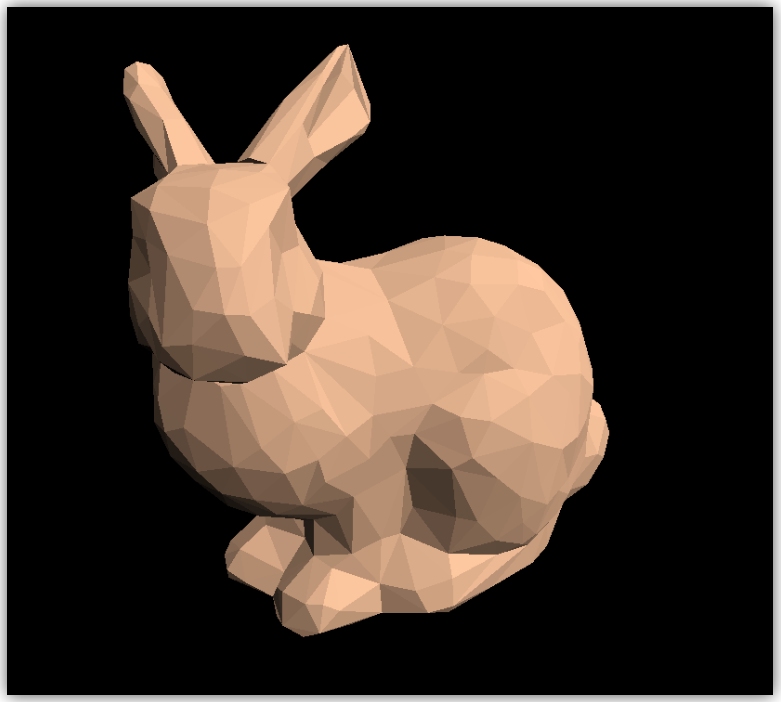
$$\mathbf{c} = \underbrace{\mathbf{c}_a}_{\text{surface color}} \circ \underbrace{\mathbf{c}_a}_{\text{ambient light color}}$$

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

Ambient Bunny



Pure Lambertian



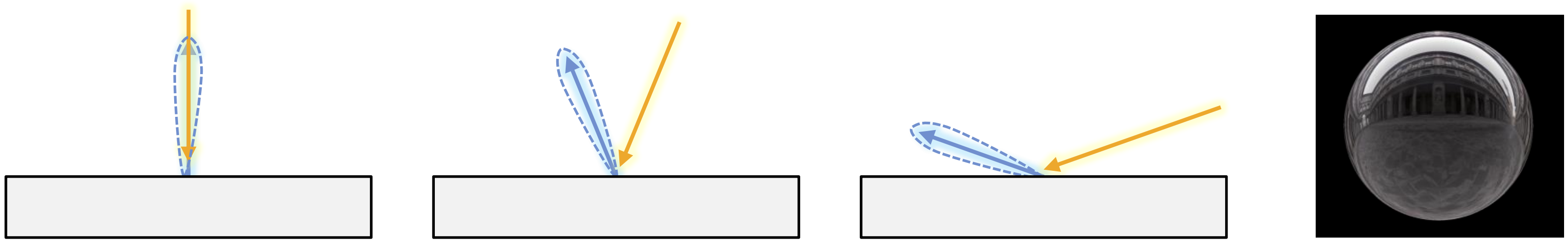
Mixed with Ambient Light

Shading Effects

Shading effects

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

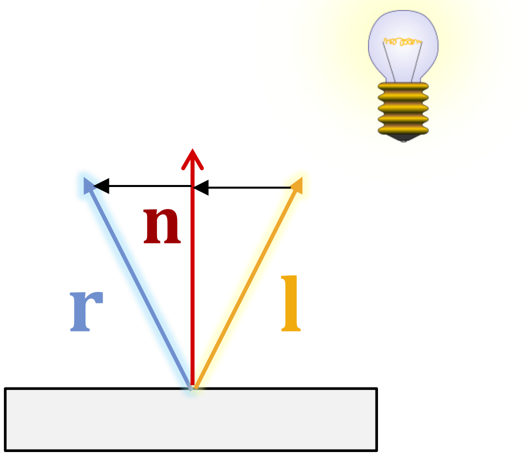
Perfect Reflection



Perfect Reflection

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

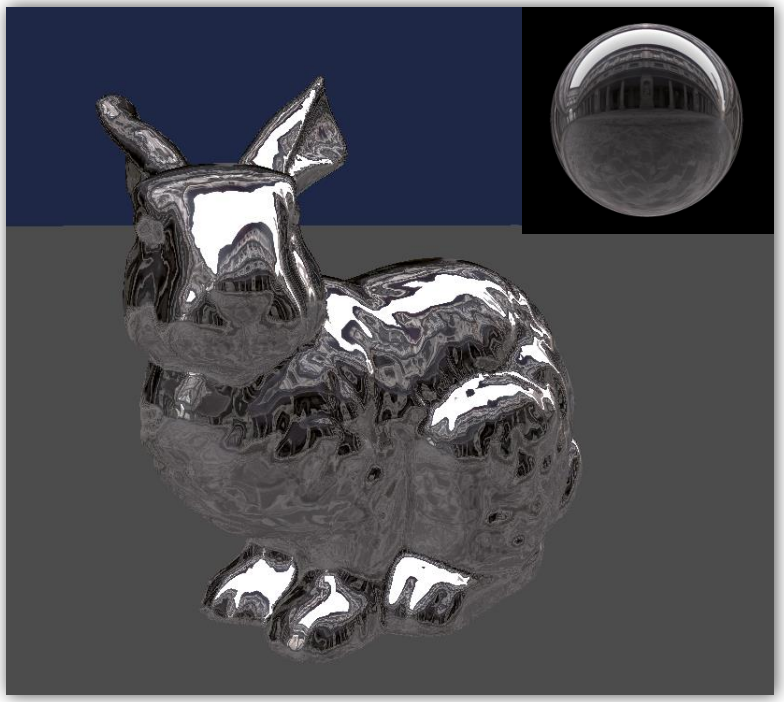
$$\mathbf{r} = 2\langle \mathbf{n}, \mathbf{l} \rangle \mathbf{n} - \mathbf{l}$$



Silver Bunny

Perfect Reflection

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



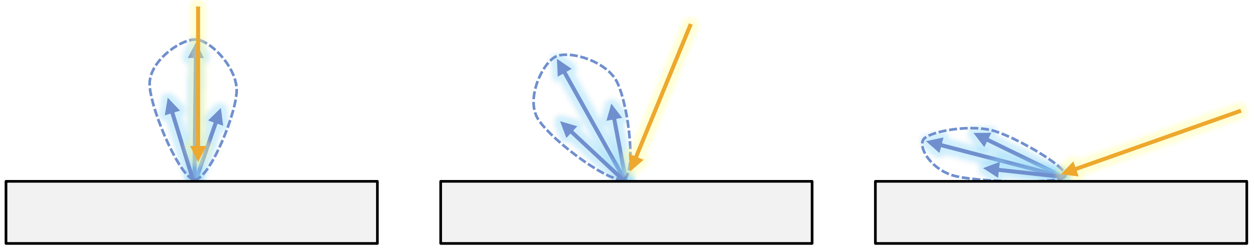
Reflective Bunny
(Interpolated Normals)

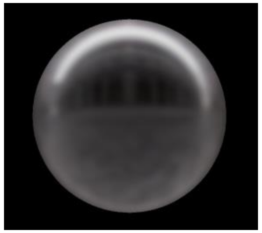
Shading Effects

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

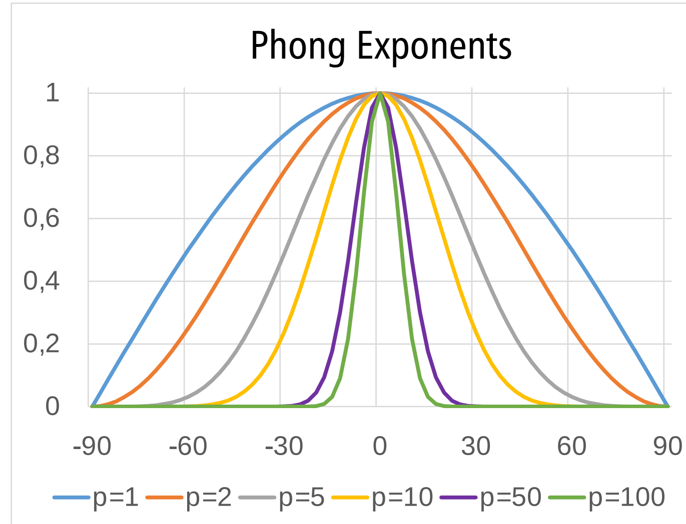
- Ambient part:

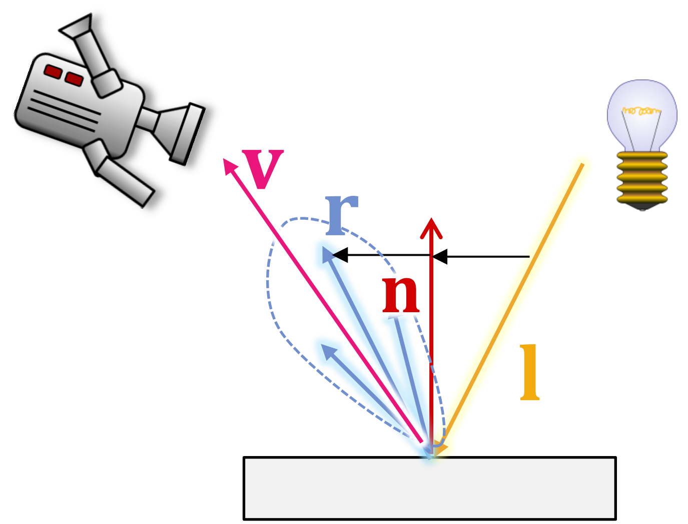
$$\mathbf{c} = \mathbf{c}_r \circ \mathbf{c}_a$$

- Diffuse part:

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

- Add all terms together



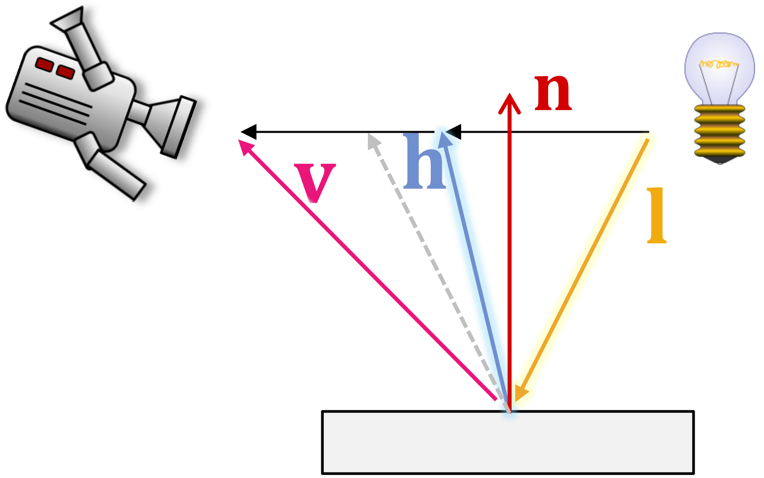


Blinn-Phong

Blinn-Phong Model:

- "Specular" (glossy) part:

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



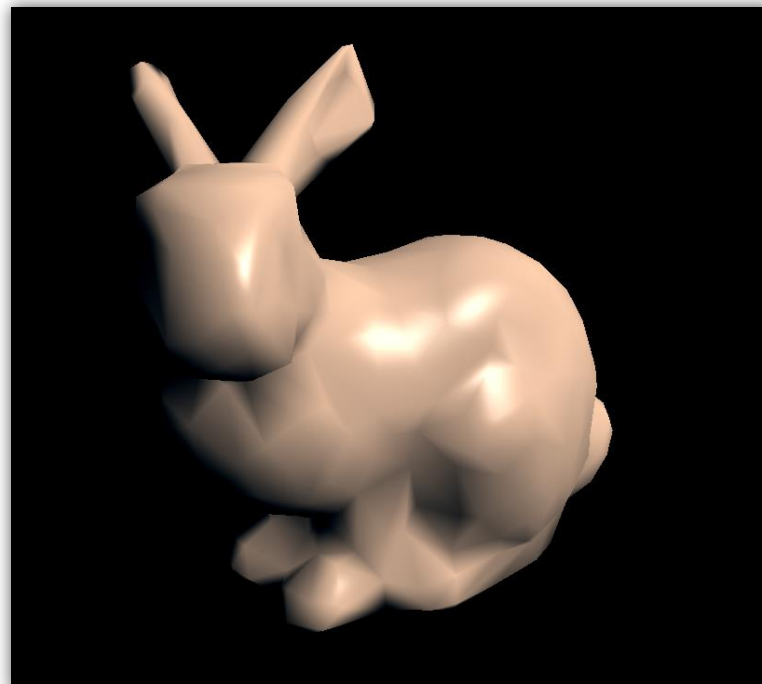
- Half-angle direction

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

- 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



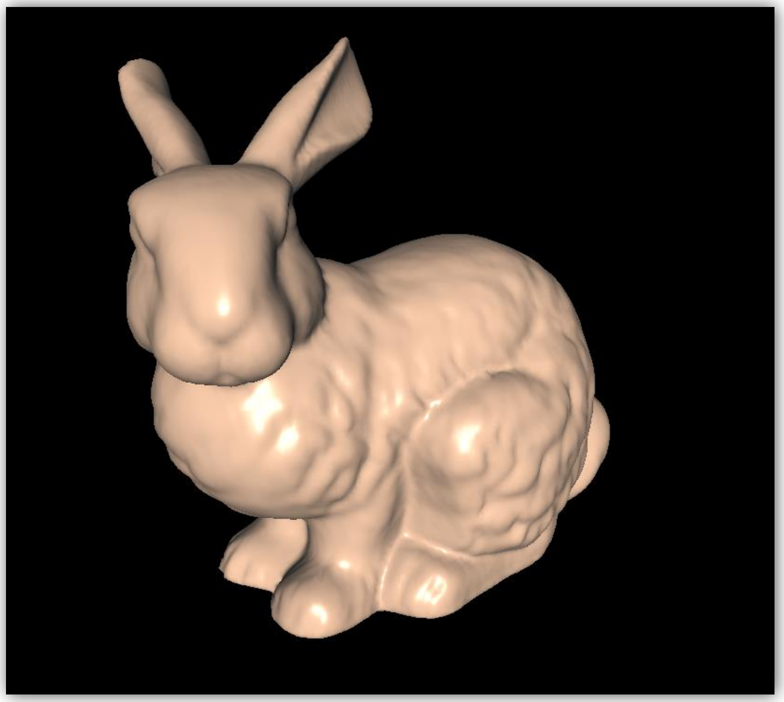
Interpolated Normals

Phong + Diffuse + Ambient Bunny

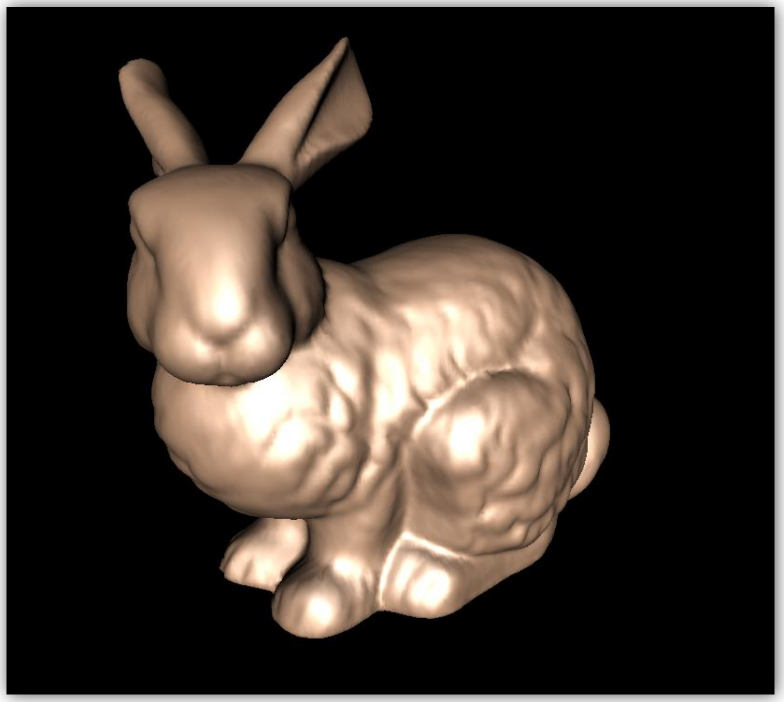
Blinn-Phong Bunny

Interpolated Normals

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

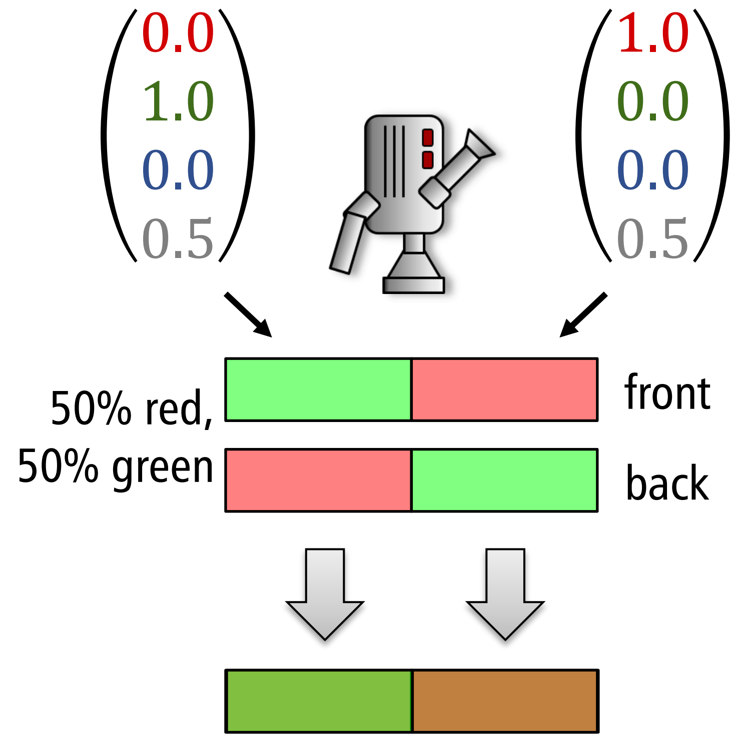
- "Alpha-blending"
- α = "opacity"
- Color + opacity: RGBα

Blending

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

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

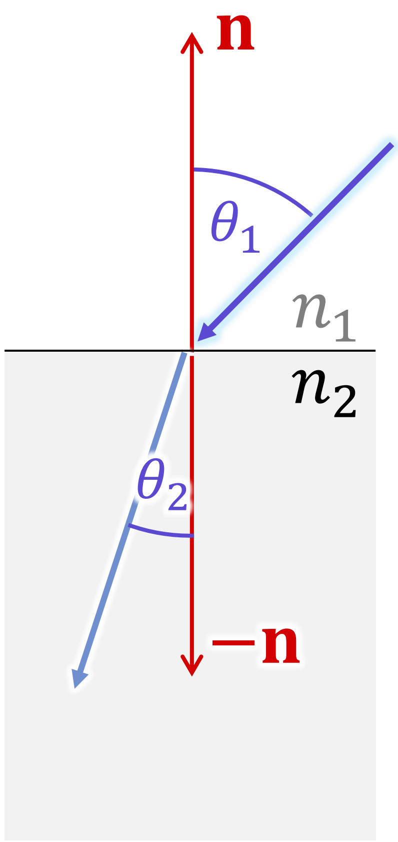
- Not commutative! (order matters)
  - unless monochrome



Refraction: Snell's Law

Refraction

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



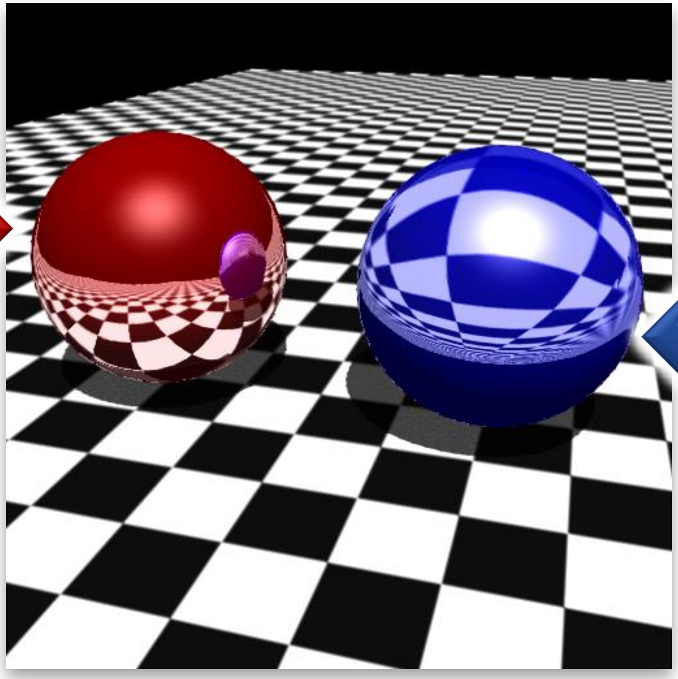
Snell's Law

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

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

Refraction

Reflection



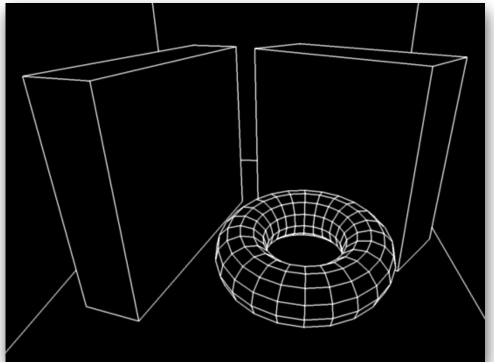
Refraction

(raytraced)

Implementation

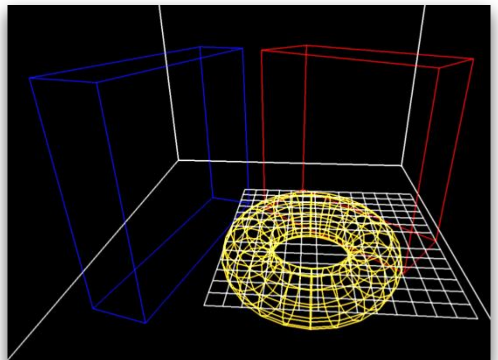
- Not a local shading model
- Global algorithms: mostly raytracing
- Various “fake” approximations for local shading

3D Rendering

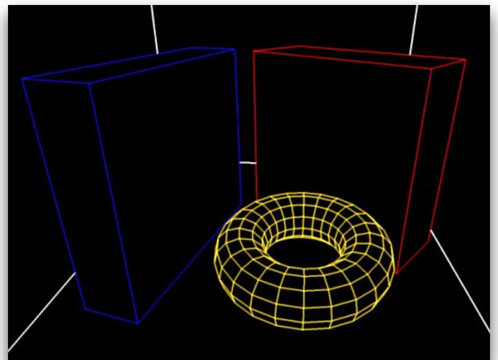


Geometric Model

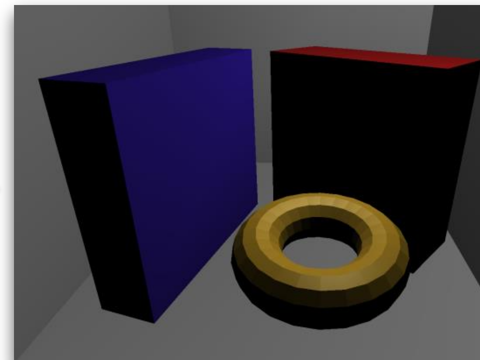
Color



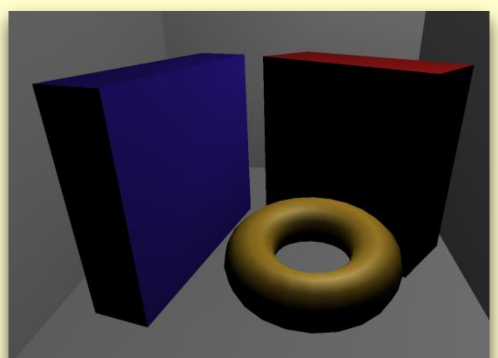
Perspective



Visibility



Local Illumination

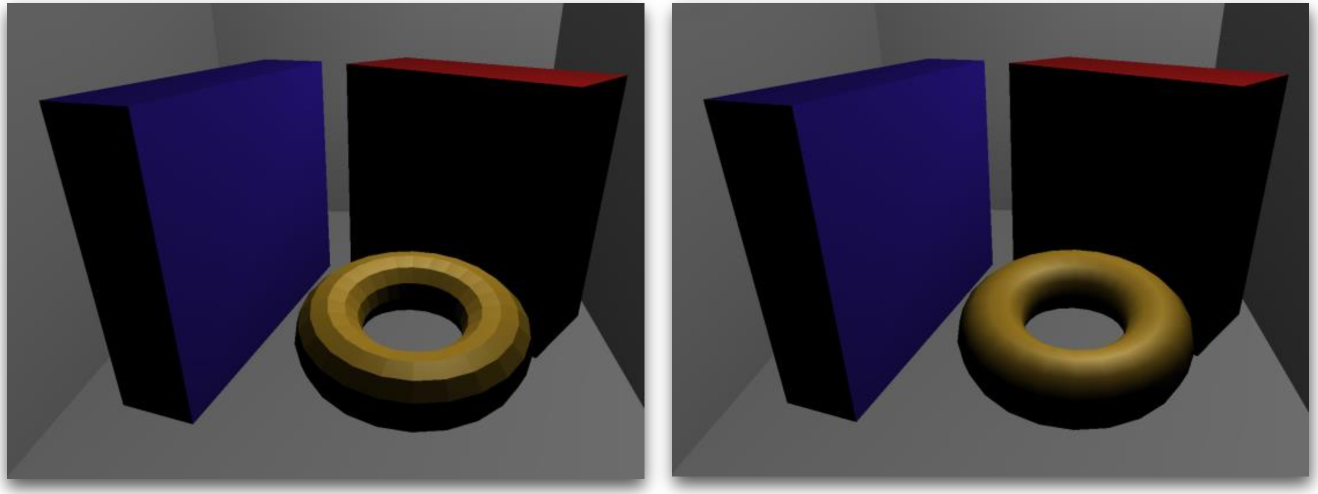


Smooth Shading

Simple Shadows

Global Illumination

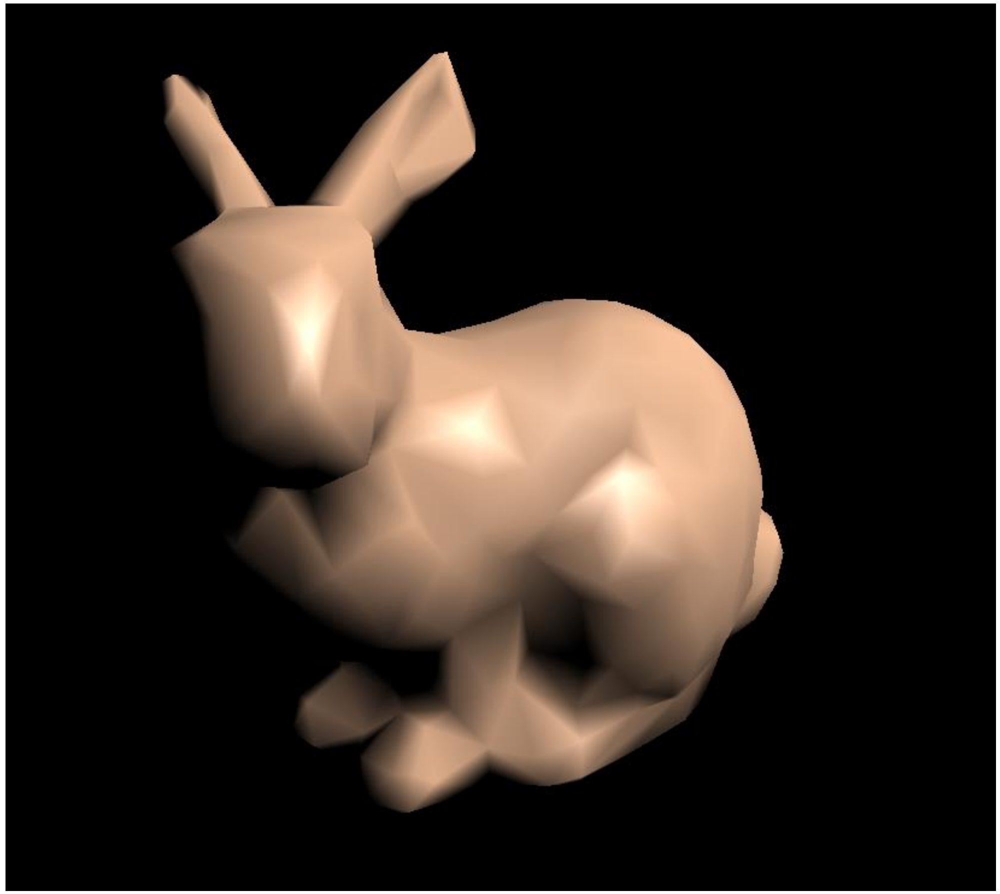
Shading Algorithms



Flat Shading

Flat Shading
constant color per triangle

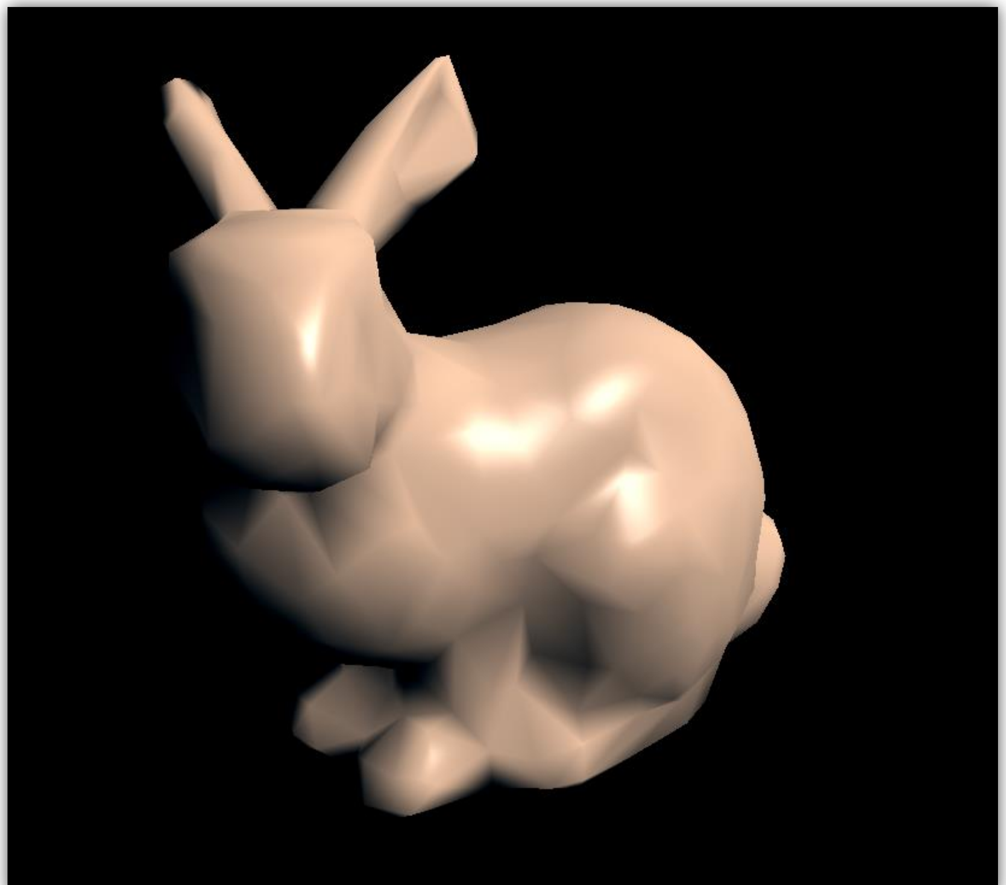
Flat Shading



"Gouraud Shading" Algorithm

compute color at vertices, interpolate color for pixels

Flat Shading



"Phong Shading" Algorithm
interpolate normals for each pixel

3D Rendering

Geometric Model

Color

Perspective

Visibility

Local Illumination

Global Illumination: next lecture

Smooth Shading

Simple Shadows

Global Illumination