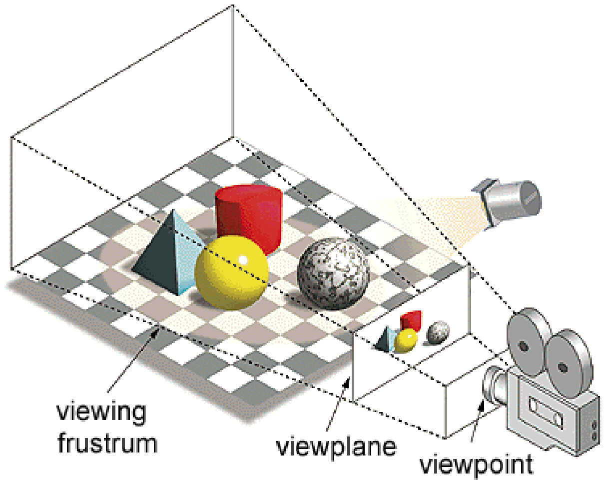
Visual Rendering: Rasterization, Lighting and Shading, Fragment Processing

CS 6384 Computer Vision

Professor Yu Xiang

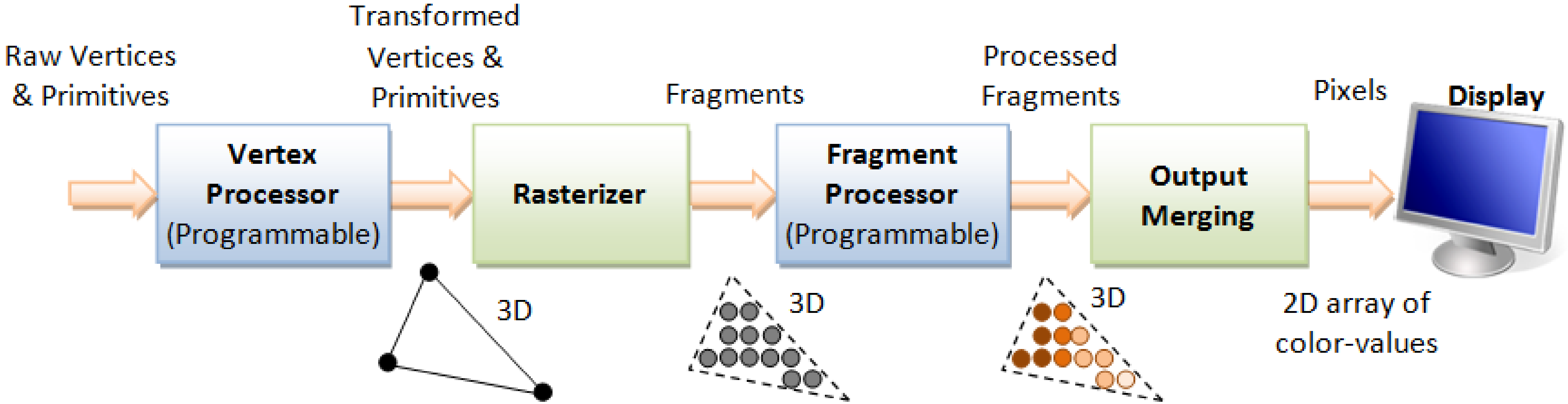
The University of Texas at Dallas

Visual Rendering

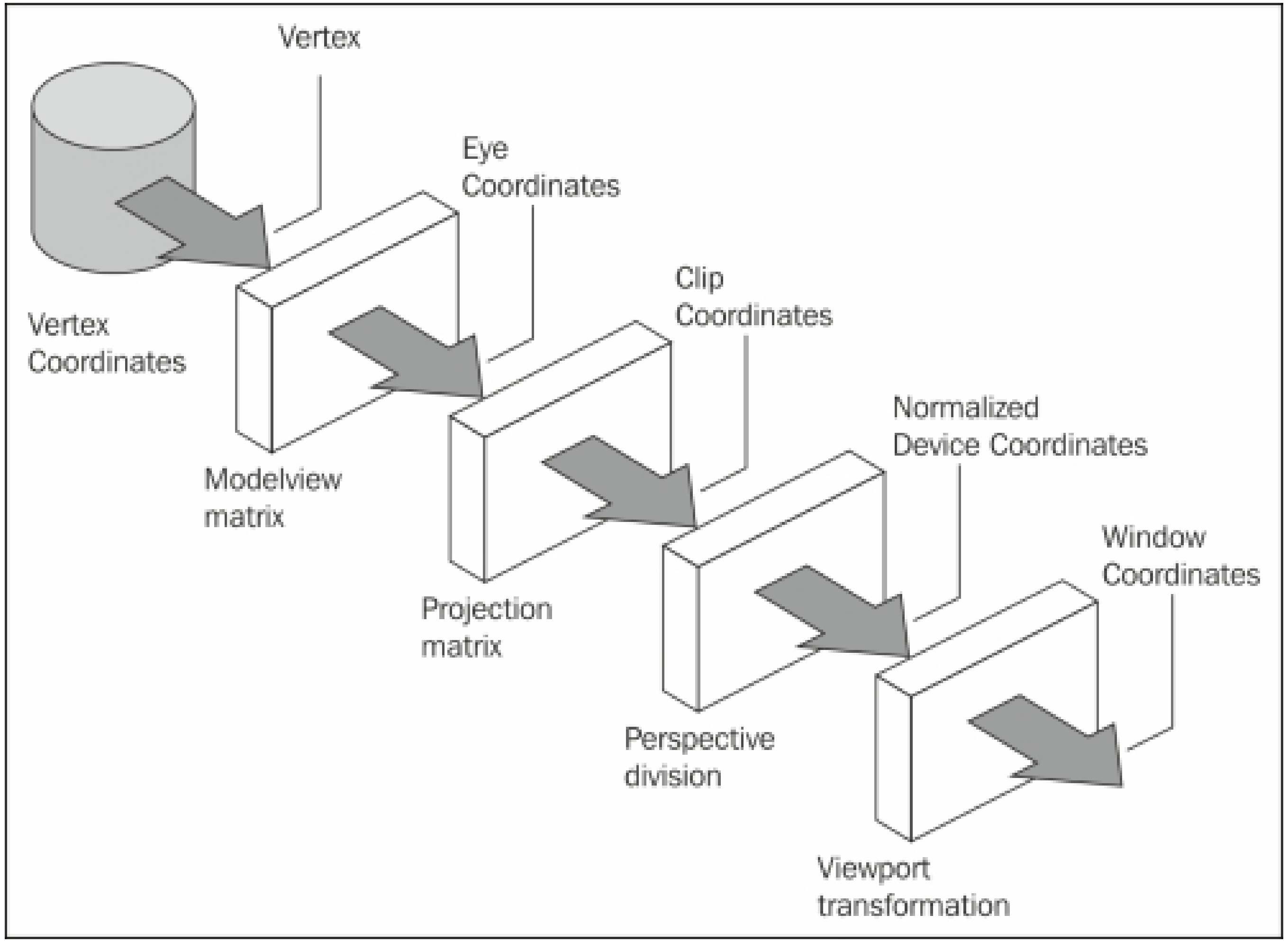


- Converting 3D scene descriptions into 2D images

- The graphics pipeline



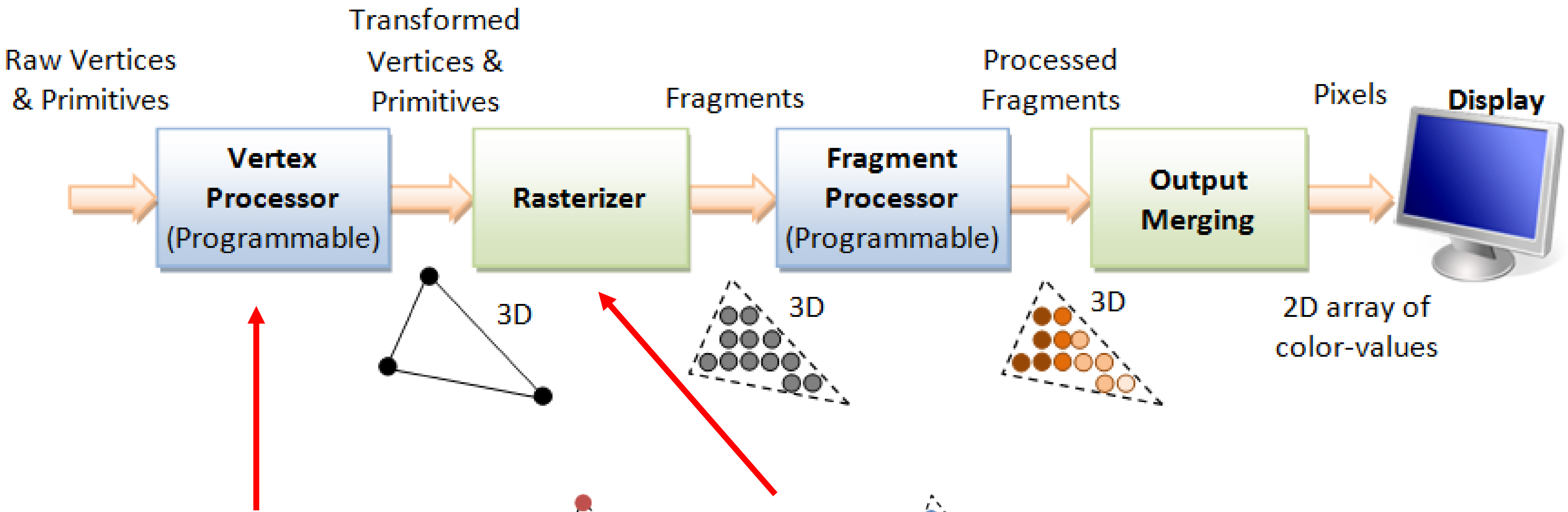
Vertex Transform



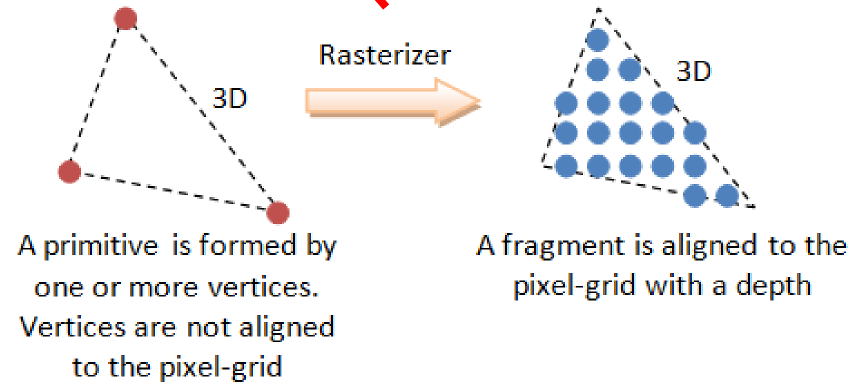
$$v_{window} = \begin{pmatrix} x_{window} \\ y_{window} \\ z_{window} \\ 1 \end{pmatrix} \begin{matrix} \in (0, width) \\ \in (0, height) \\ \in (0,1) \\ \end{matrix}$$

vertex in window coords

Rasterization



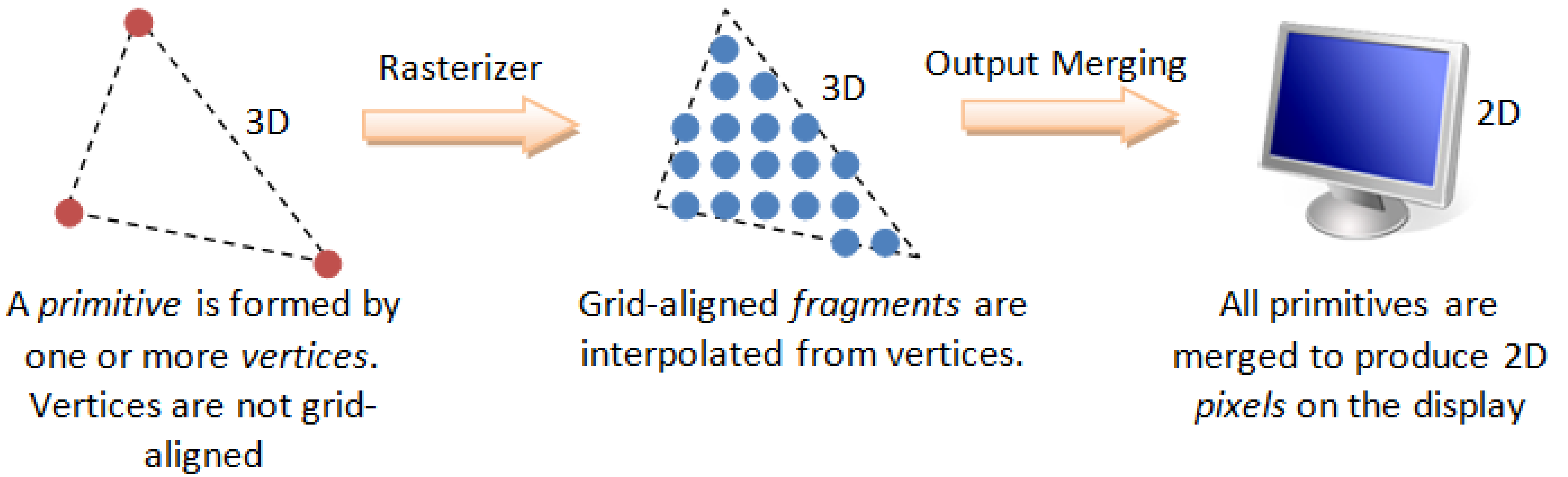
Vertex transforms



- Determine which pixels are inside the triangles
- Interpolate vertex attributes (e.g., color)

Pixels vs. Fragments

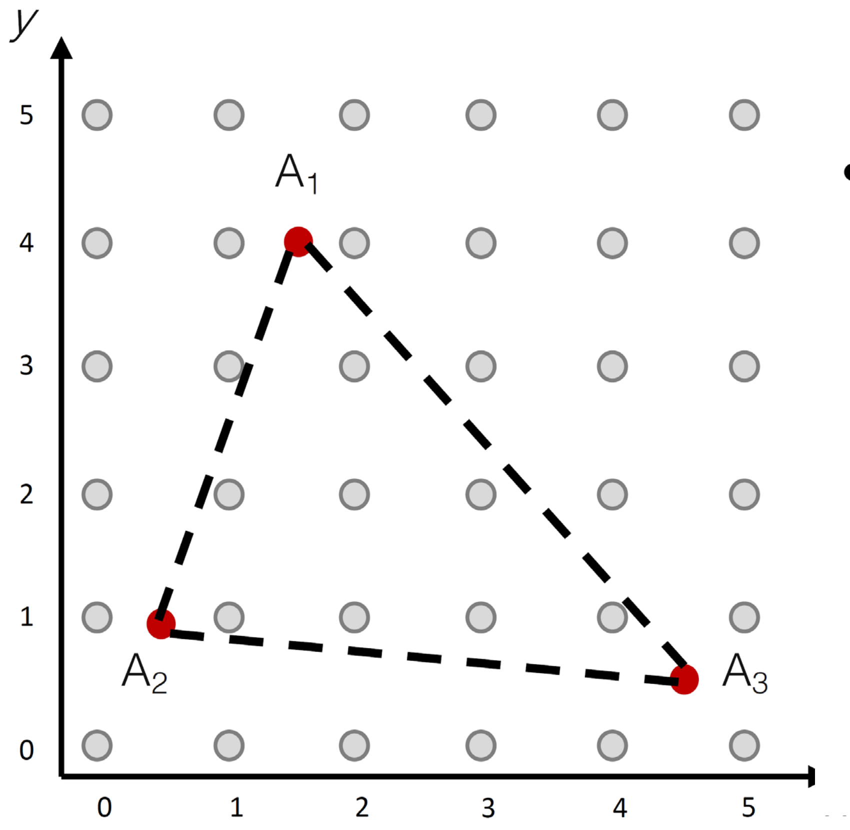
- Pixels are dots on the screen: (x, y) and RGB color
- Fragments: (x, y, z), z is the depth and other attributes (color, normal, texture coordinates, alpha value, etc.)

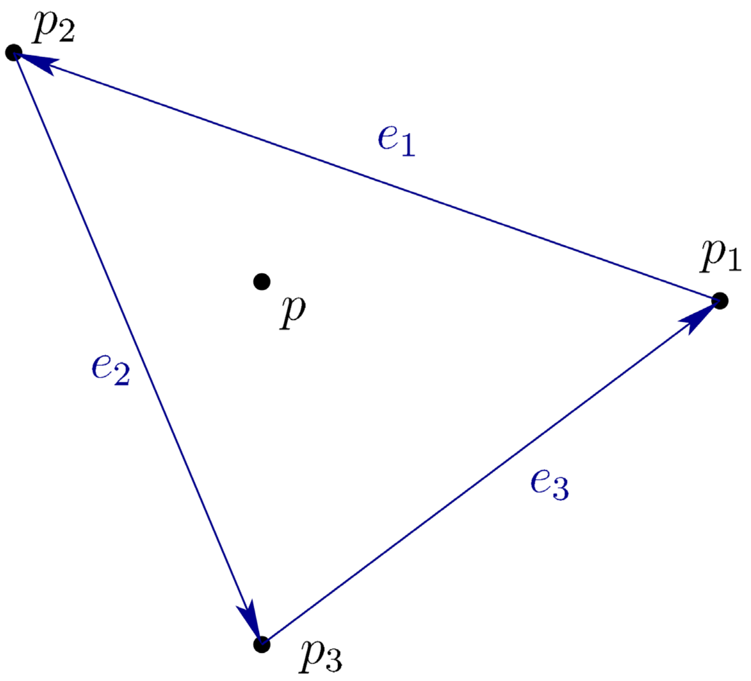


Vertex, Primitives, Fragment and Pixel

Rasterization

- Determine which fragments are inside the triangle





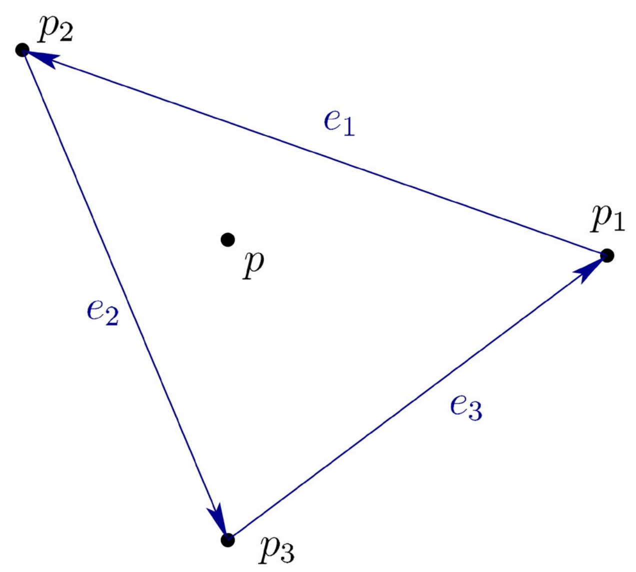
$$e_1 = p_2 - p_1$$
$$e_2 = p_3 - p_2$$
$$e_3 = p_1 - p_3$$

p is inside if and only if

$$(p - p_1) \times e_1 < 0$$
$$(p - p_2) \times e_2 < 0$$
$$(p - p_3) \times e_3 < 0$$

magnitude of the cross products

Barycentric Coordinates



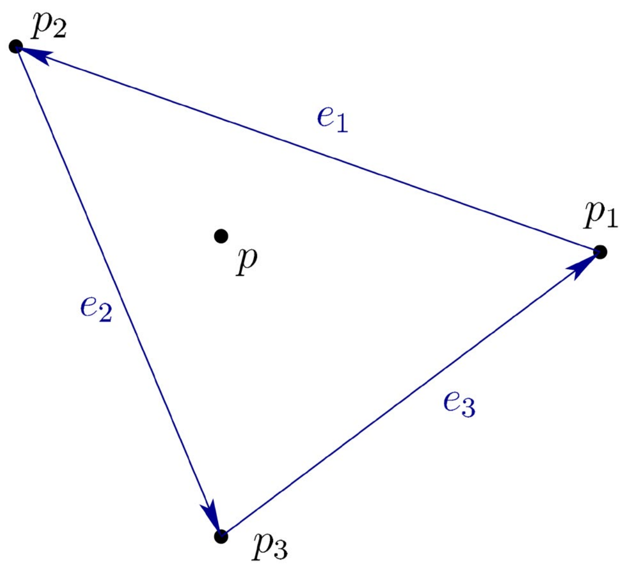
- Interpolate attributes of the vertices

$$p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3$$

$$0 \leq \alpha_1, \alpha_2, \alpha_3 \leq 1$$

$$\alpha_1 + \alpha_2 + \alpha_3 = 1$$

Barycentric Coordinates



$$\mathbf{p}_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \quad \mathbf{p}_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} \quad \mathbf{p}_3 = \begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3$$

$$0 \leq \alpha_1, \alpha_2, \alpha_3 \leq 1 \quad \alpha_1 + \alpha_2 + \alpha_3 = 1$$

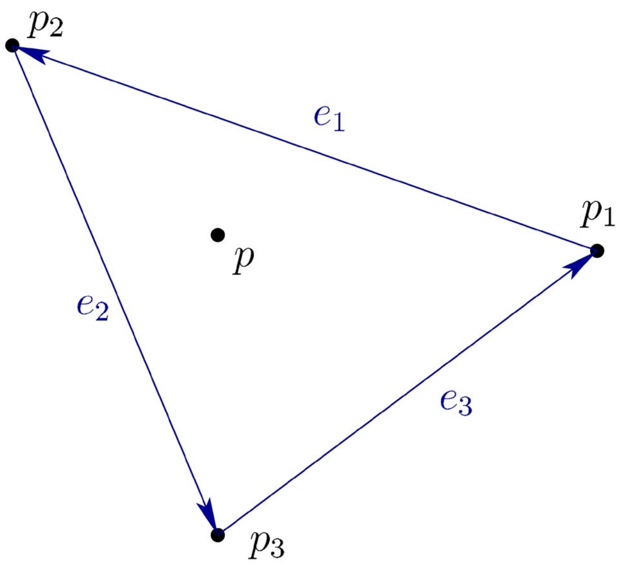
$$\alpha_1 = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)},$$

$$\alpha_2 = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{(y_2 - y_3)(x_1 - x_3) + (x_3 - x_2)(y_1 - y_3)},$$

$$\alpha_3 = 1 - \alpha_1 - \alpha_2.$$

https://en.wikipedia.org/wiki/Barycentric_coordinate_system

Barycentric Coordinates



$$p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3$$

Color

$$R = \alpha_1 R_1 + \alpha_2 R_2 + \alpha_3 R_3$$
$$G = \alpha_1 G_1 + \alpha_2 G_2 + \alpha_3 G_3$$
$$B = \alpha_1 B_1 + \alpha_2 B_2 + \alpha_3 B_3$$

Apply to other attributes, e.g., depth, texture coordinates, alpha value, etc.

Depth Buffer for Visibility Testing

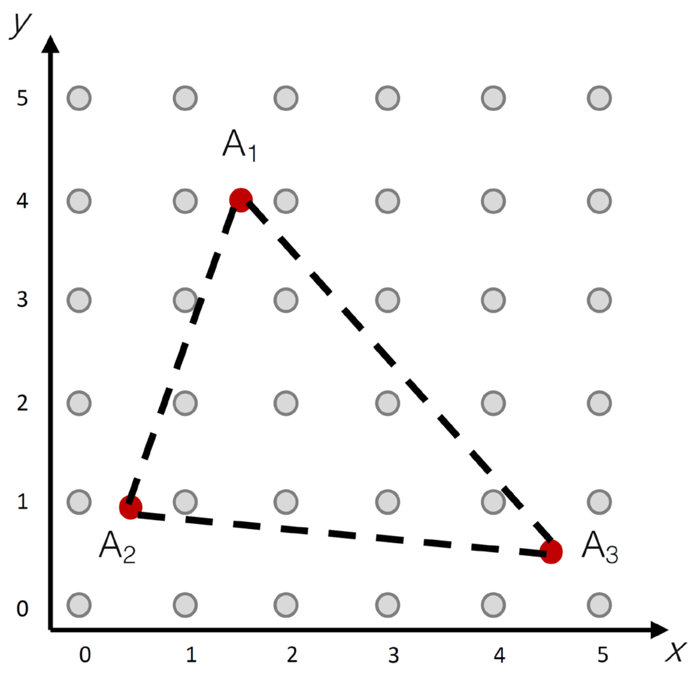
- When drawing multiple triangles, determine which one to draw and which one to discard
- If depth of fragment is **smaller than** the current value is the depth buffer, overwrite color and depth value using the current fragment

color buffer

depth buffer

Lighting and Shading

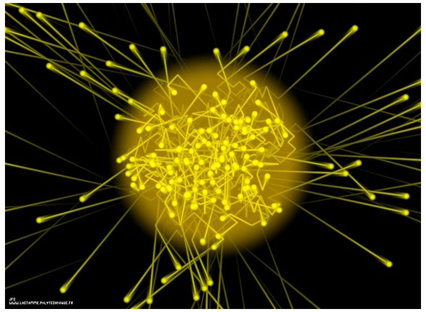
- How to determine color and what attributes to interpolate after rasterization



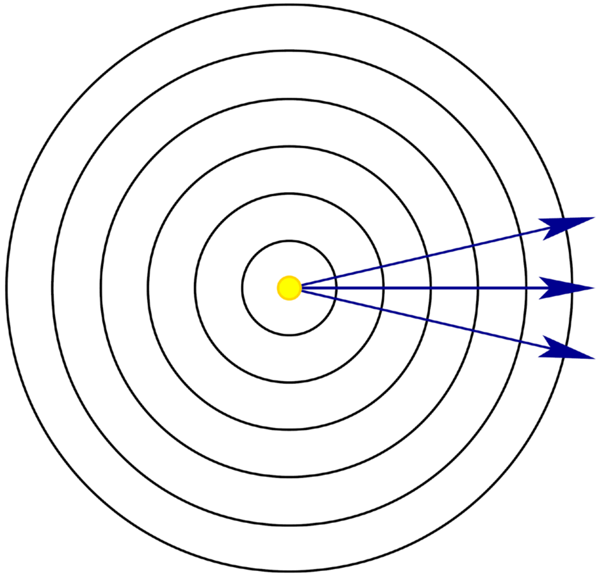
Rasterization: determine which fragments are inside the triangles

Basic Behavior of Light

- Light can be described in three ways
  - Photons: tiny particles of energy moving through space at high speed

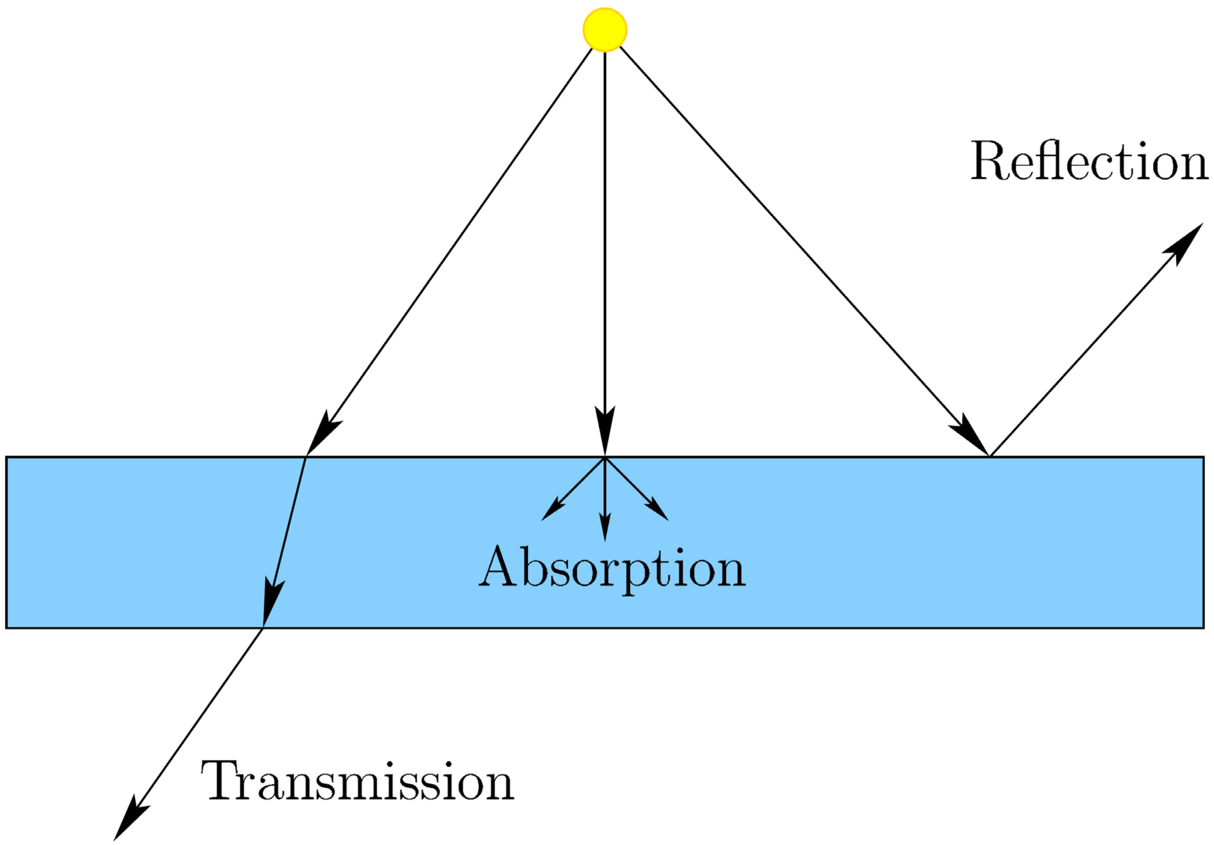


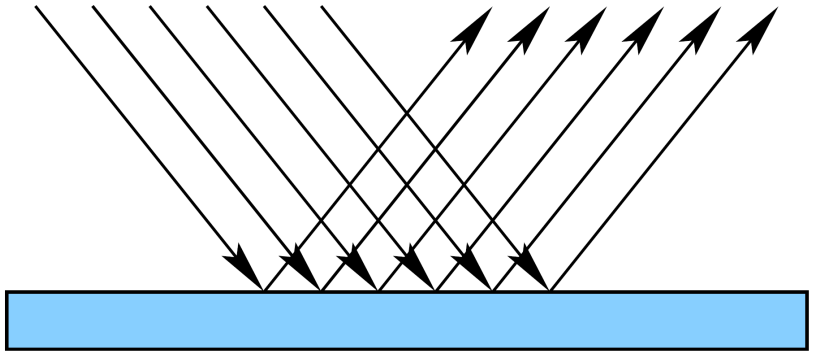
  - Waves: ripples through space



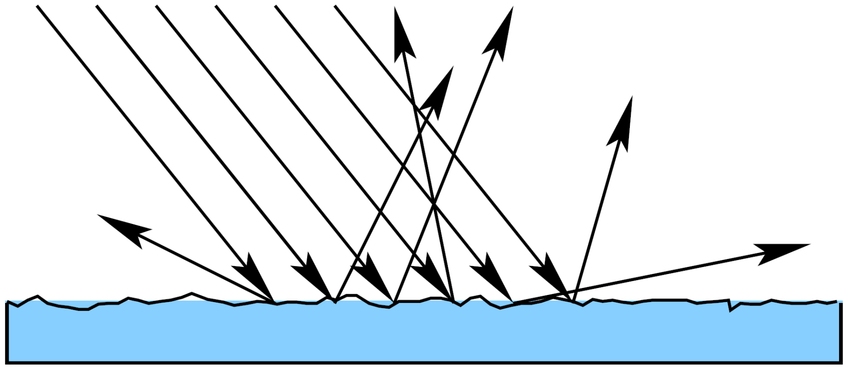
  - Rays: a ray traces the motion of a single hypothetical photon

Interactions with Materials



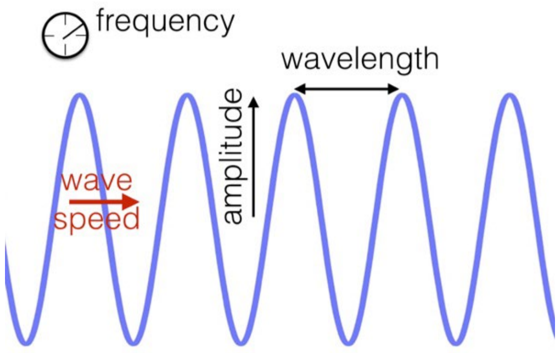


Specular



Diffuse

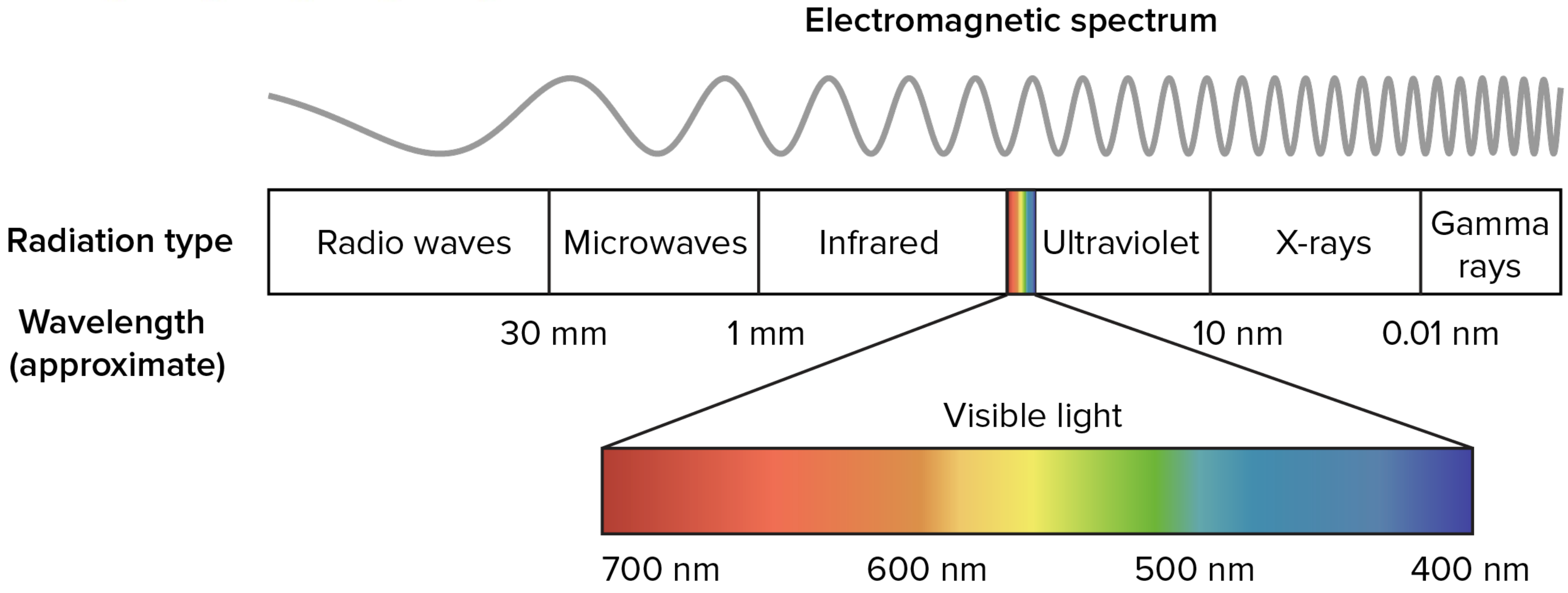
Wavelengths and Colors



Wavelength $\lambda = \frac{v}{f}$

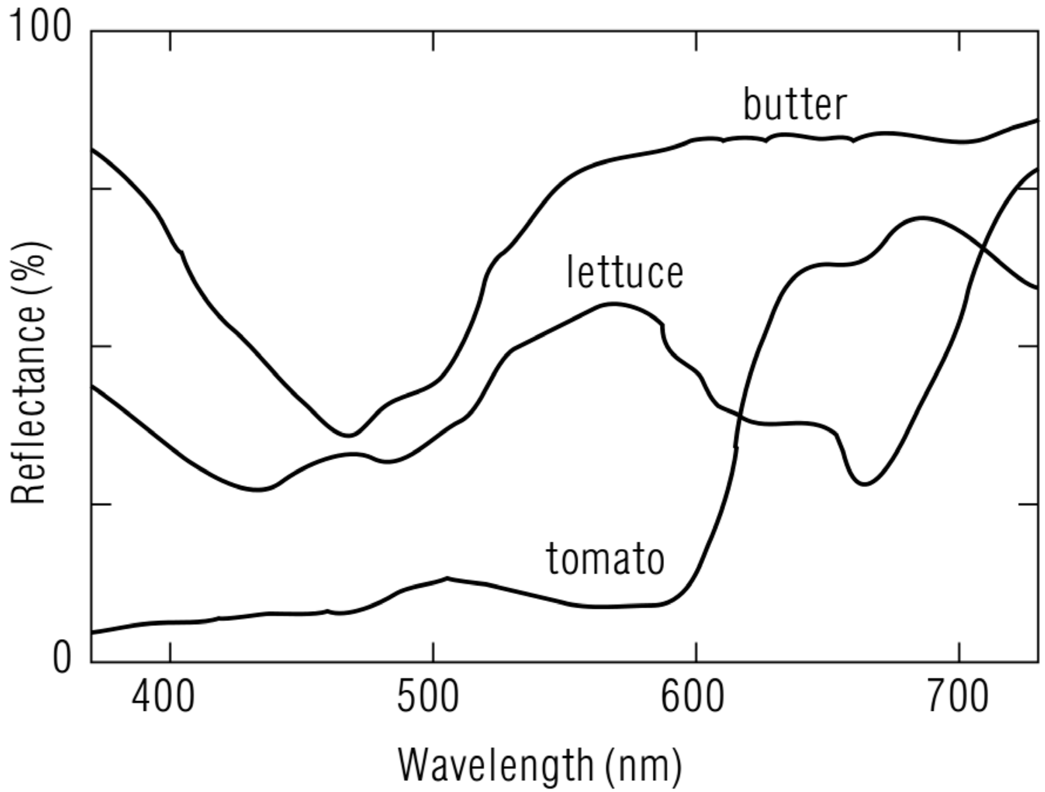
Speed: meters per second

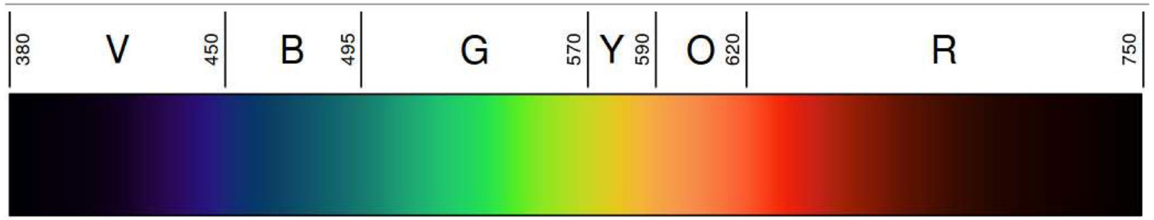
Frequency: how many cycles per second



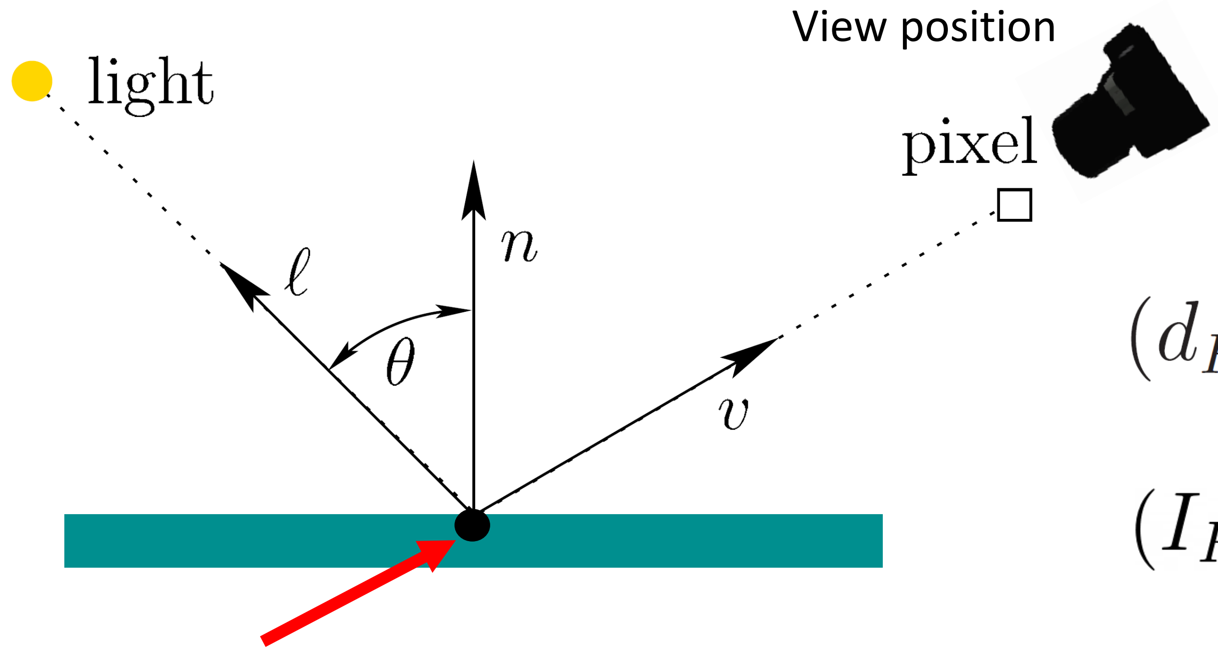
Reflection of Materials

- We see objects with different colors because the materials reflect specific colors differently





Lambertian Lighting



Think about this point as a vertex of a 3D mesh. We want to compute its color on the image

Diffuse reflection

$$R = d_R I_R \max(0, n \cdot \ell)$$
$$G = d_G I_G \max(0, n \cdot \ell)$$
$$B = d_B I_B \max(0, n \cdot \ell)$$
$$n \cdot \ell = \cos\theta$$

(d_R, d_G, d_B) Reflectance property of the material (triangle)

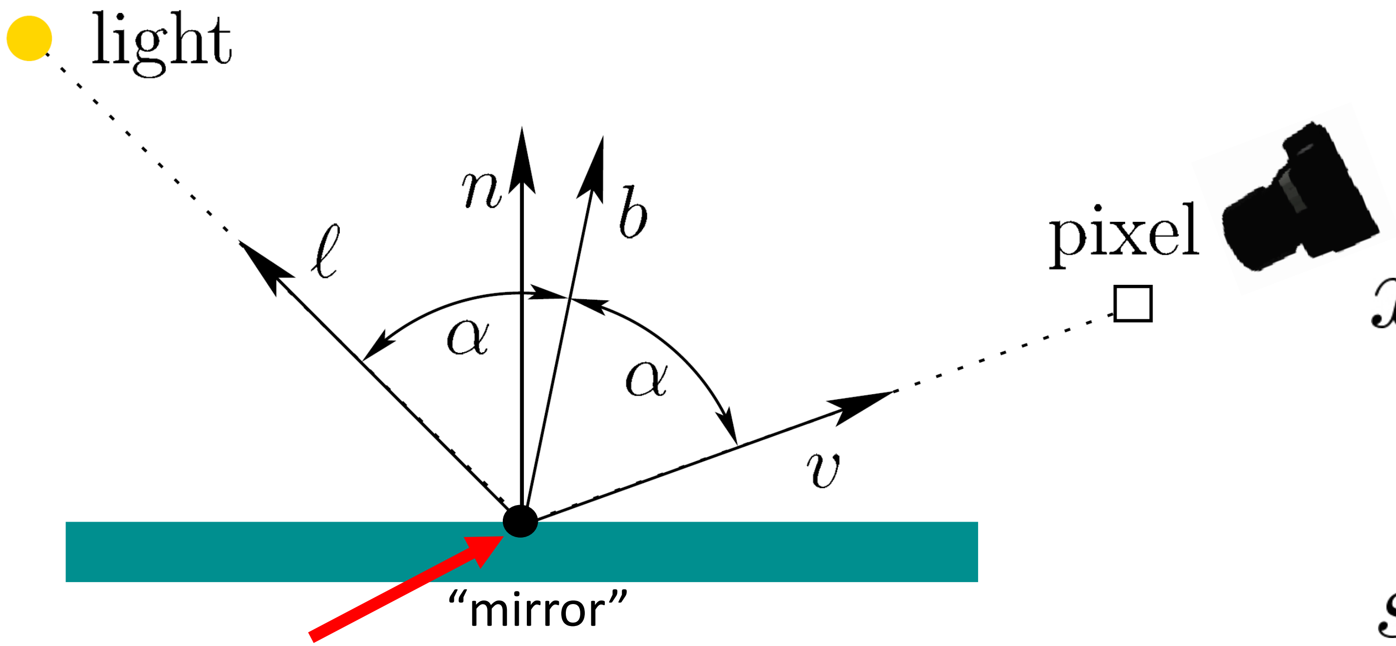
(I_R, I_G, I_B) Spectral power distribution of the light source

$$L = dI \max(0, n \cdot \ell)$$

$n \cdot \ell < 0$

Light behind triangle

Blinn-Phong Lighting



Think about this point as a vertex of a 3D mesh. We want to compute its color on the image

Related to specular reflection

$$b = \frac{\ell + v}{\|\ell + v\|}$$

x Material property that expresses the amount of surface shininess

x=100, mild amount of shininess

x=10000, almost like a mirror

s Specular reflectance property of the material

$$L = dI \max(0, n \cdot \ell) + sI \max(0, n \cdot b)^x$$

Ambient Lighting

- Independent of light/surface position, viewer, normal

- Adding some background color

$$L = dI\ \max(0, n \cdot \ell) + sI\ \max(0, n \cdot b)^x + L_a$$

Ambient light

Multiple Light Sources and Attenuation

- N light sources

$$L = L_a + \sum_{i=1}^{N} dI_i \max(0, n \cdot l_i) + sI_i \max(0, n \cdot b_i)^x$$

- Attenuation: the greater the distance, the low the intensity

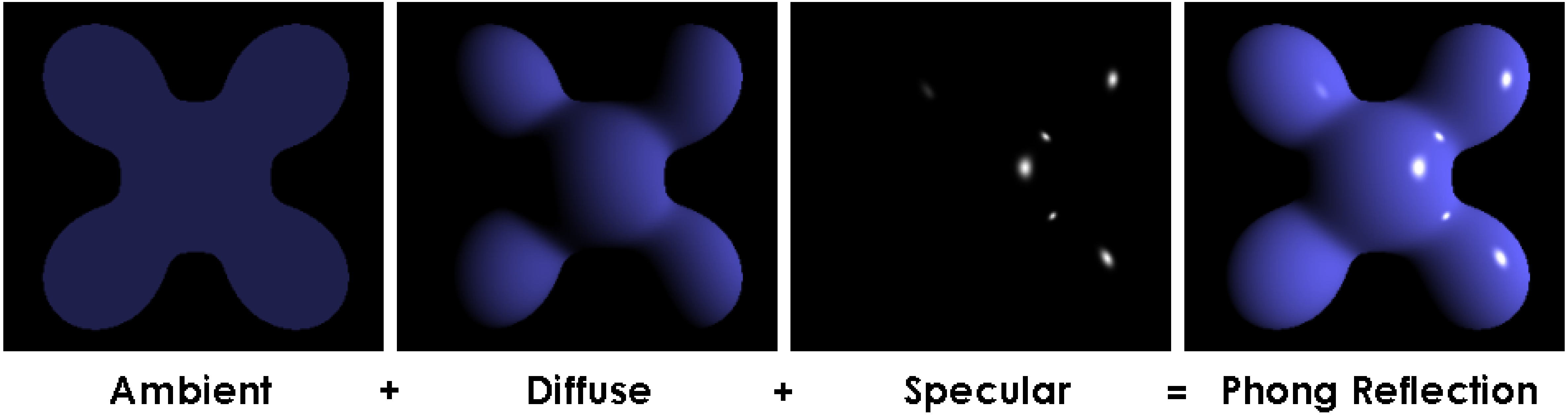
$$L = L_a + \sum_{i=1}^{N} \frac{1}{k_c + k_l c + k_q c^2} \left(dI_i \max(0, n \cdot l_i) + sI_i \max(0, n \cdot b_i)^x \right)$$

constant (for k_c), linear (for k_l), quadratic attenuation (for k_q)

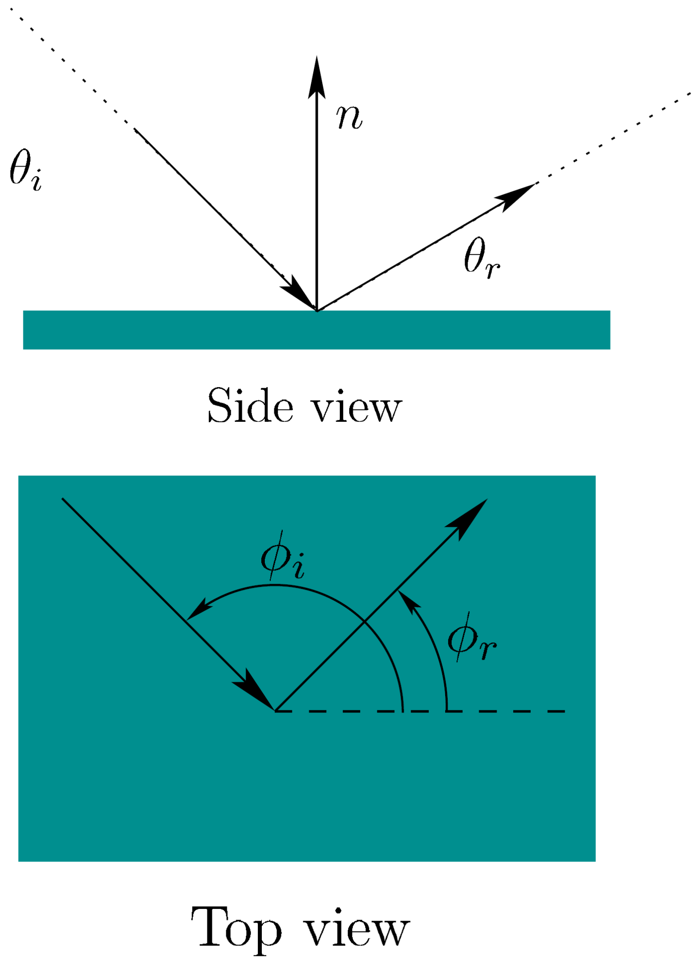
c Light source distance to surface

Used by OpenGL for ~25 years

Phong Reflection Model



Bidirectional Reflectance Distribution Function (BRDF)



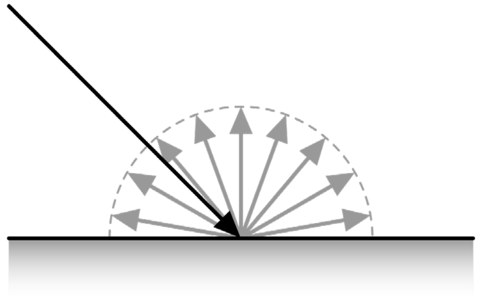
Shading in a more precise and general way

$$f(\theta_i, \phi_i, \theta_r, \phi_r) = \frac{\text{radiance}}{\text{irradiance}}$$

- Radiance: light energy reflected from the surface
- Irradiance: light energy arriving at the surface

For Lambertian shading, BRDF is a constant

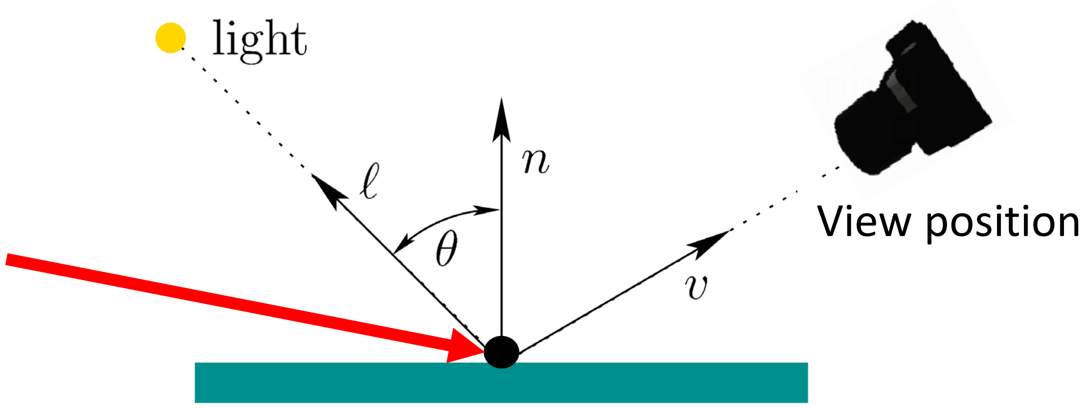
- The surface reflects equally in all directions



Lighting Calculations

- All lighting calculations can happen in world space
  - Transform vertices and normal into world space
  - Calculate lighting, i.e., compute vertex color given material properties, light source color and position, vertex position, normal position, view position

Think about this point as a vertex of a 3D mesh. We want to compute its color on the image



Lighting vs. Shading

- Lighting: interaction between light and surface
  - Different mathematic models exist, e.g., Phong lighting model
  - What formula is being used to calculate intensity/color

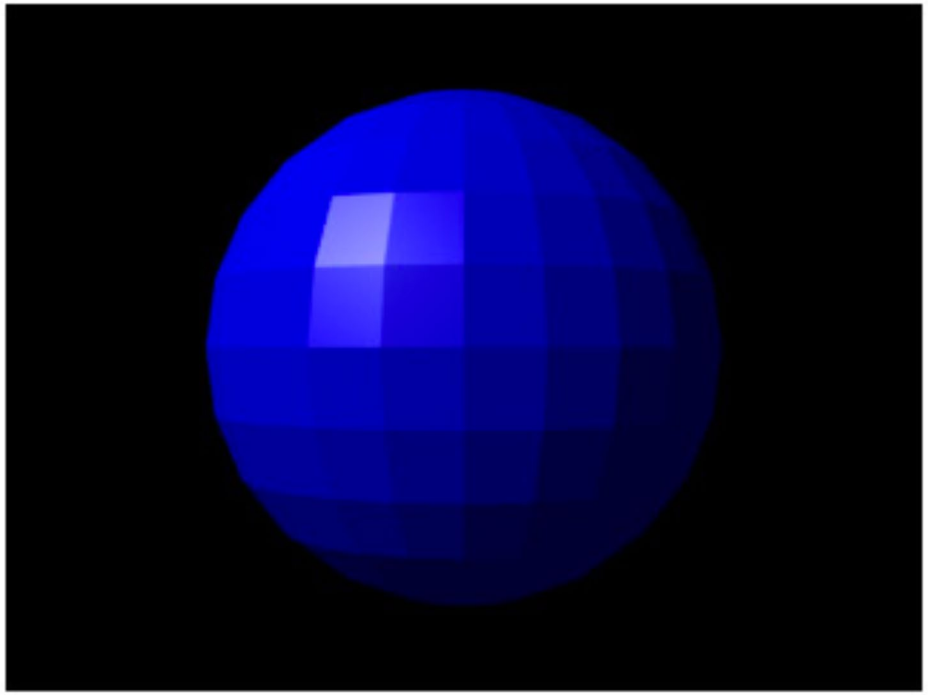
- Shading: how to compute color for each fragment
  - What attributes to interpolate
  - Where to do lighting calculation

Flat Shading

- Compute color only once per triangle (i.e., with Phong lighting)
  - Compute color for the first vertex or the centroid

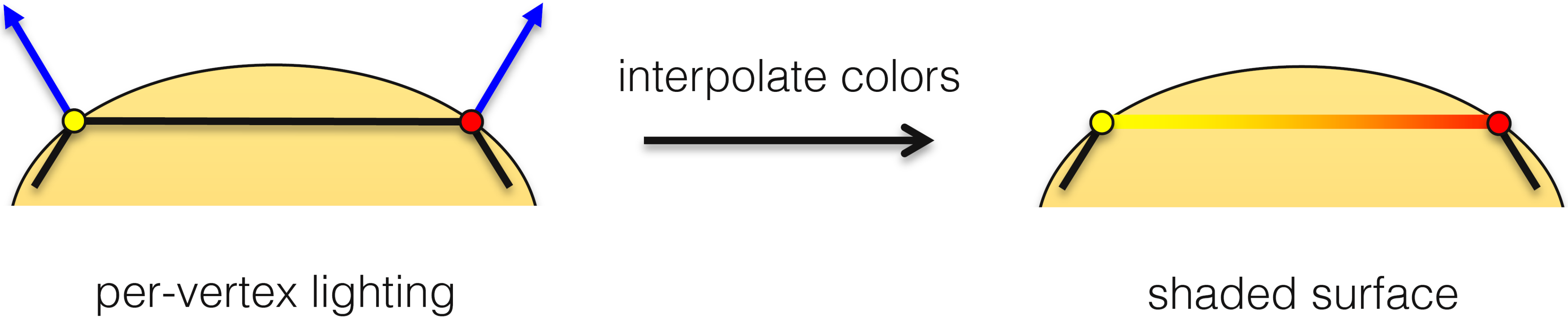
- Pro: fast to compute

- Con: create a flat, unrealistic appearance

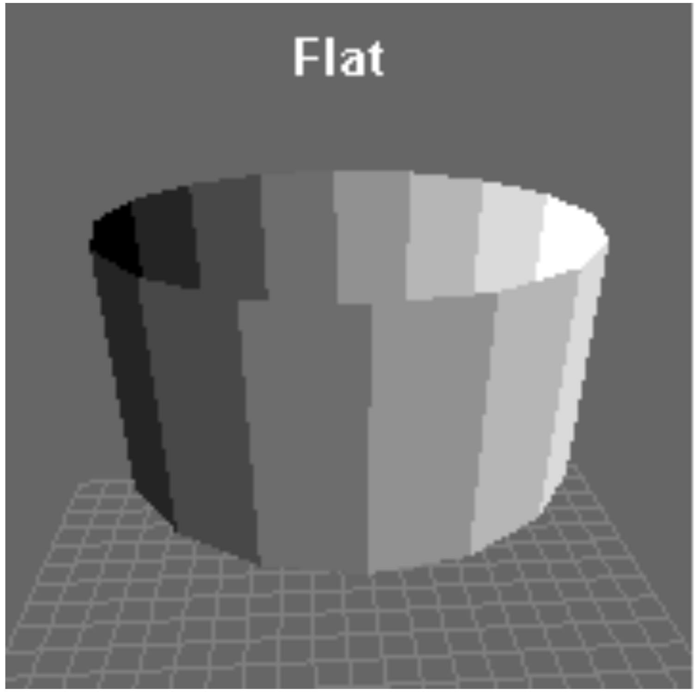


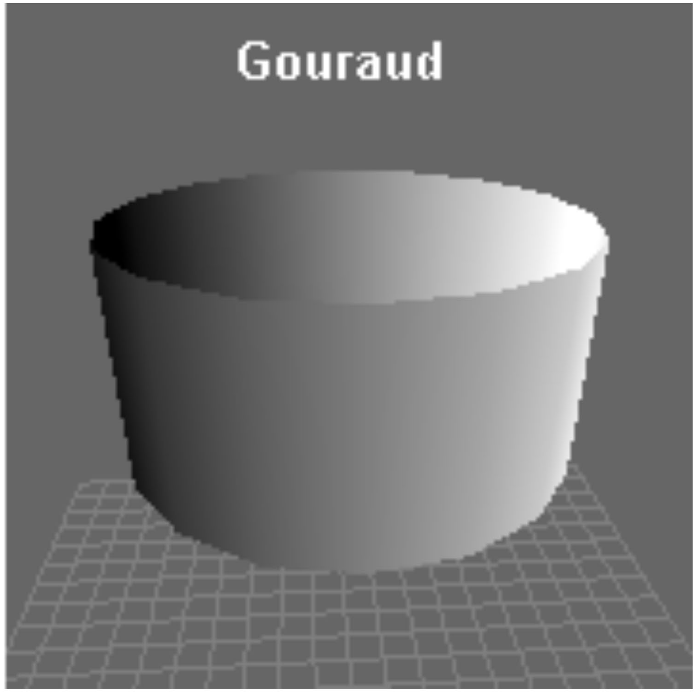
Gouraud or Per-vertex Shading

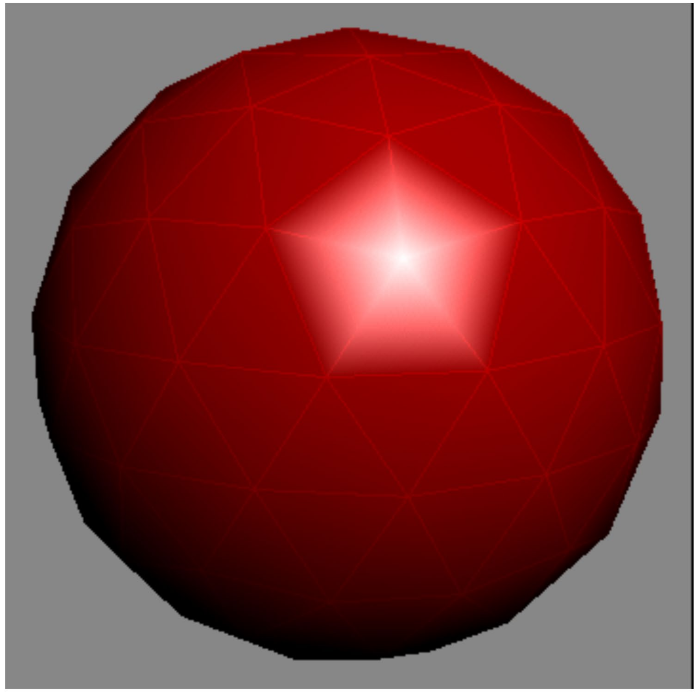
- Compute color only once per vertex (i.e., with Phong lighting)
- Interpolate per-vertex color to all fragments within the triangle
- Pro: fast to compute
- Con: flat, unrealistic specular highlights



Gouraud or Per-vertex Shading

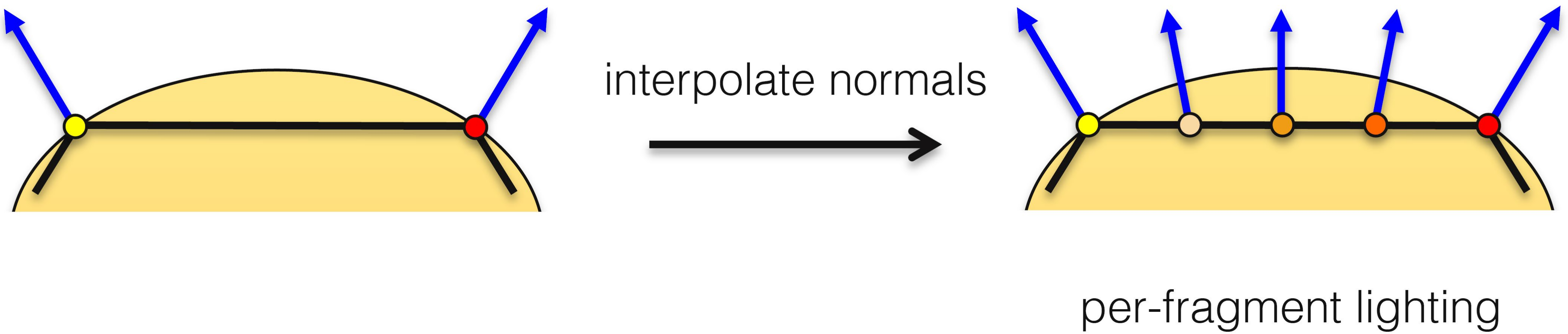






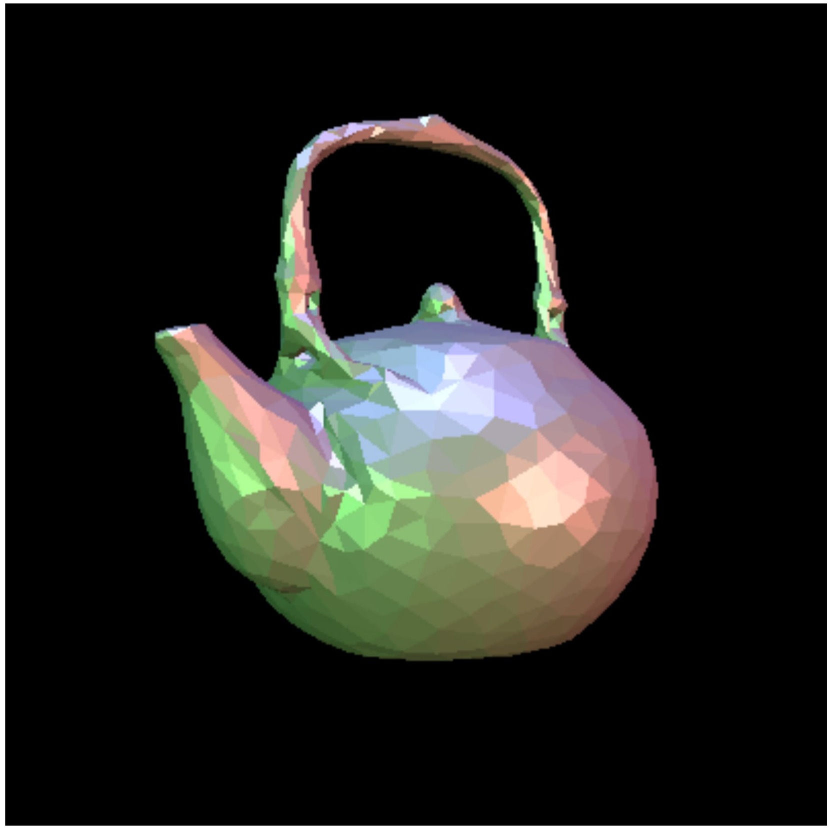
Phong Shading or Per-fragment Shading

- Compute color only once per fragment (i.e., with Phong lighting)
- Need to interpolate per-vertex normal to all fragments to do the lighting calculation
- Pro: better appearance of specular highlights
- Con: slower to compute

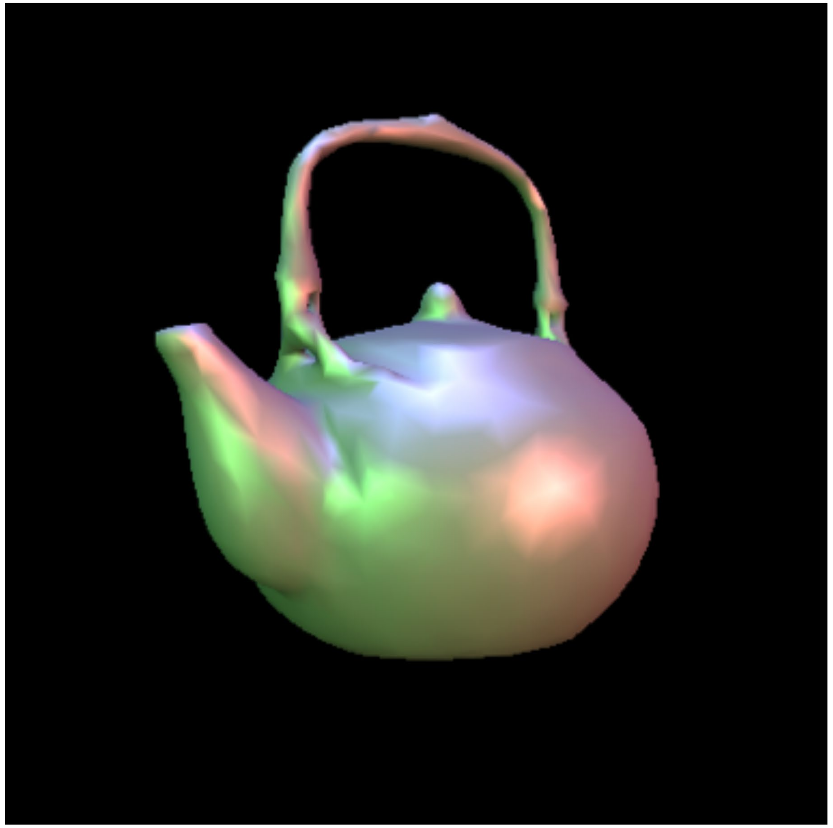


Shading

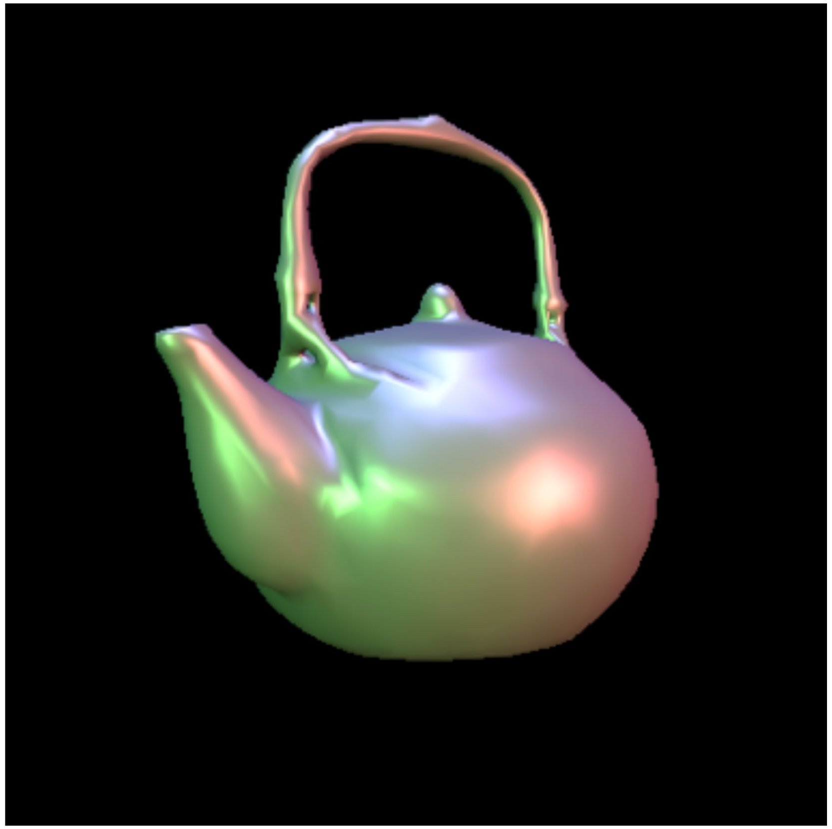
Flat Shading



Gouraud Shading

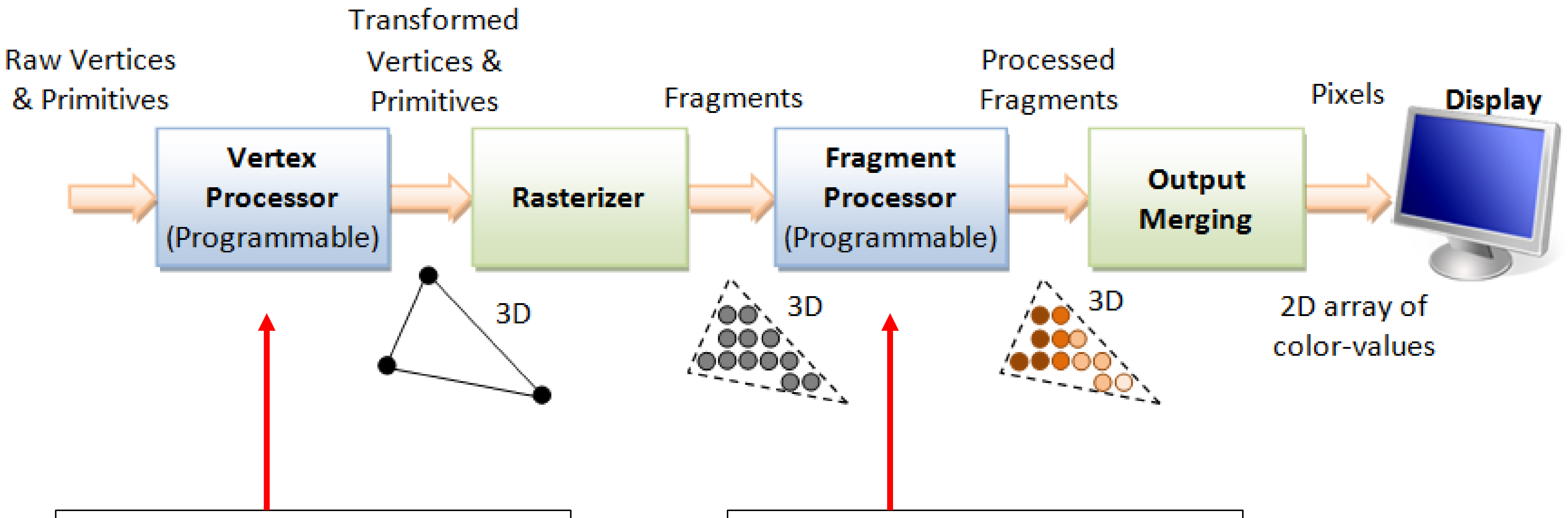


Phong Shading



http://www.decew.net/OSS/timeline.php

Shader



Vertex shader
- Lighting computation for each vertex

Fragment shader
- Lighting computation for each fragment

Shader

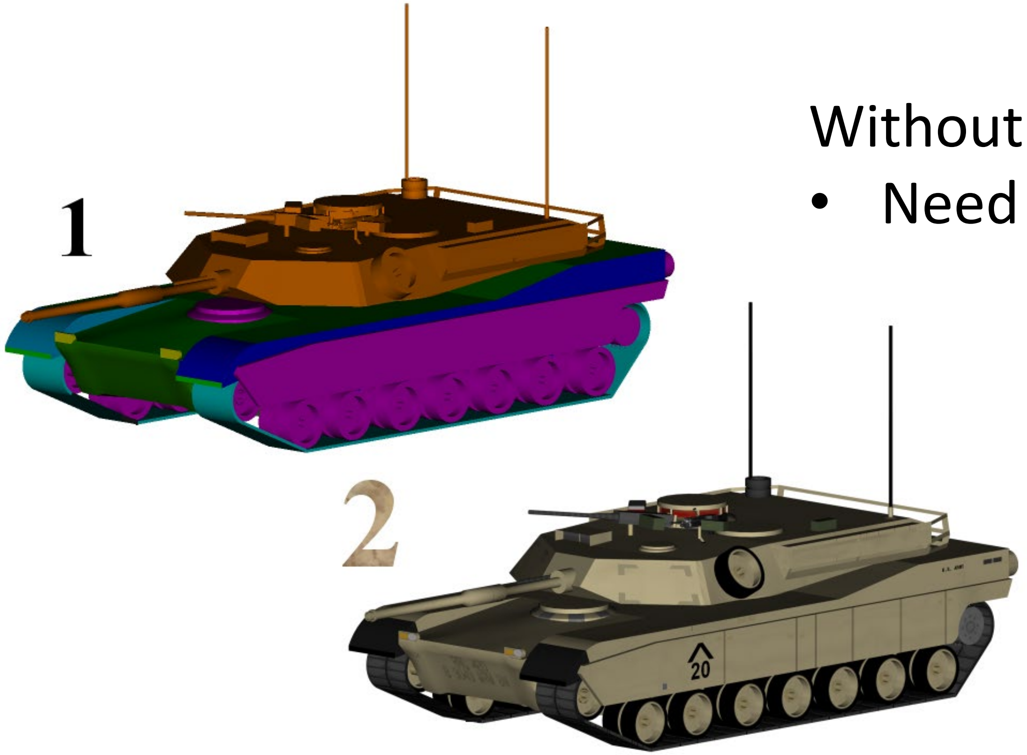
- Shaders are small programs that are executed in parallel on GPUs for each vertex (vertex shader) or each fragment (fragment shader)

- Vertex shader (before rasterization)
  - Modelview projection transform of vertex and normal
  - If per-vertex lighting, compute lighting for each vertex

- Fragment shader (after rasterization)
  - If per-vertex lighting, assign color to each fragment
  - If per-fragment lighting, compute lighting for each fragment

Texture Mapping

- Map textures (2D images) to 3D models

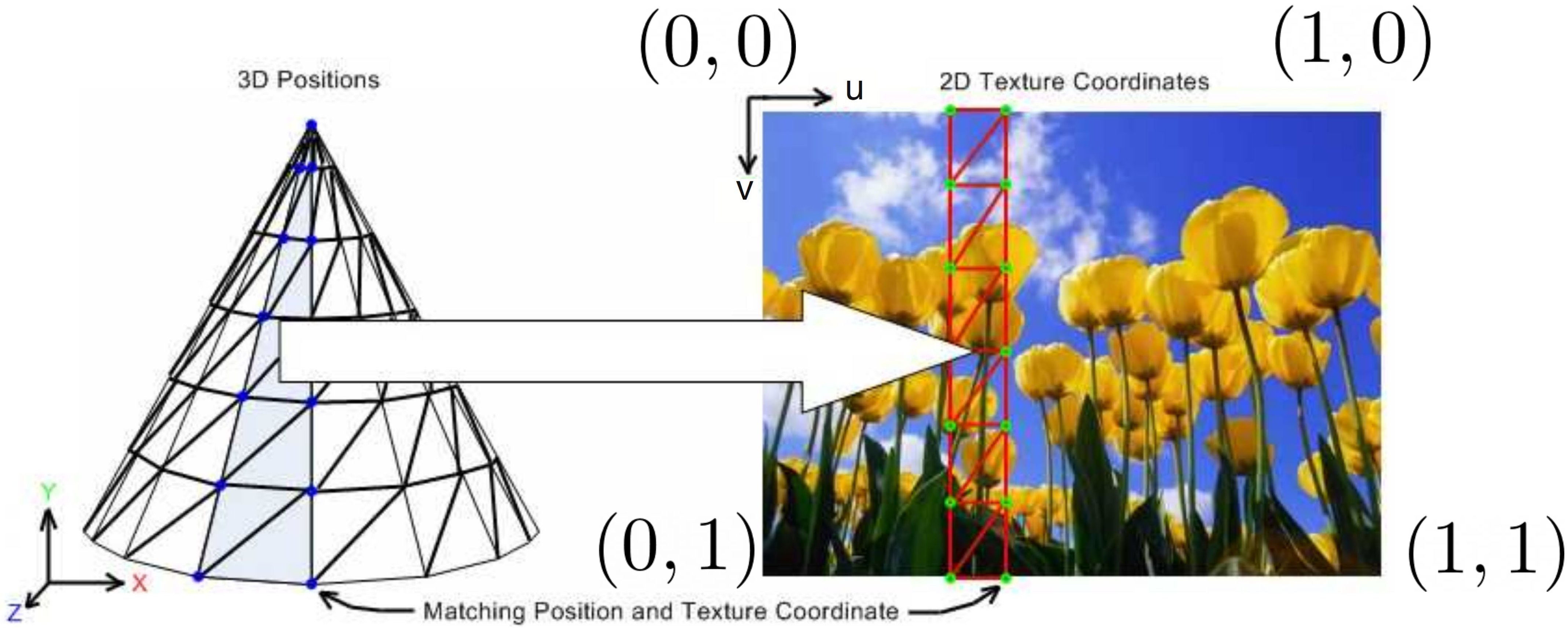


Without texture
- Need to specify vertex colors

With texture
- Vertex colors from texture

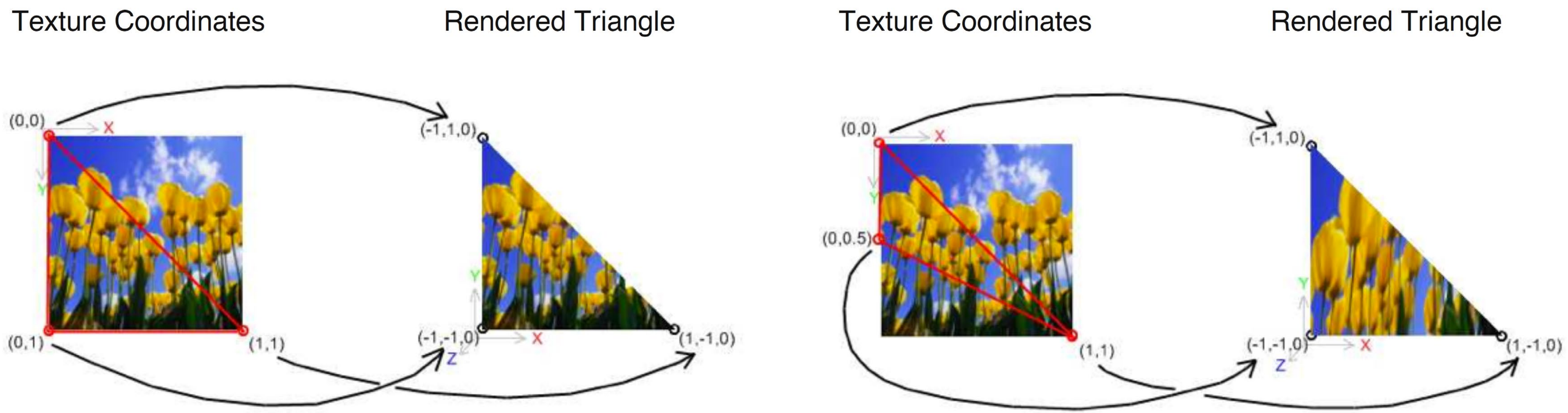
Texture Mapping

- UV coordinates (normalized)



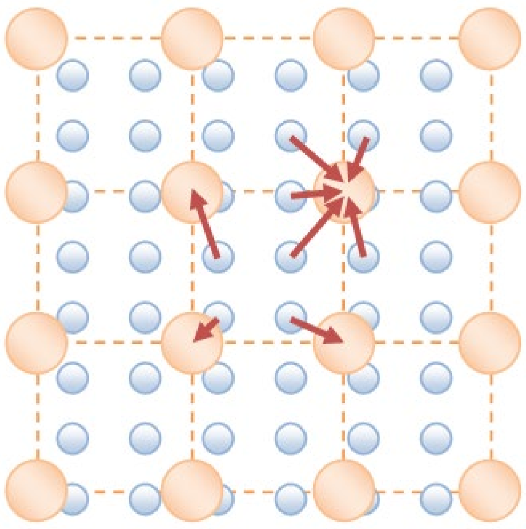
Texture Mapping

- Same texture, different UV coordinates for mapping

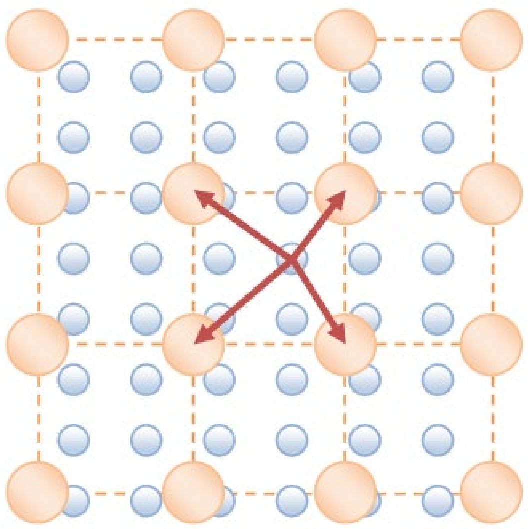


Texture Mapping

- Texture filtering: the resolution of the texture image is different from the displayed fragment
  - Magnification
  - Minification

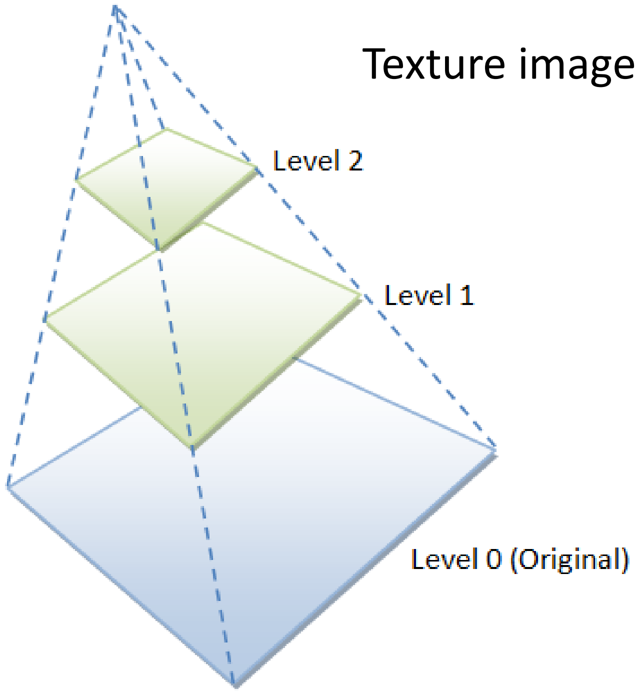


Magnification – Nearest Point Sampling



Magnification – Bilinear Interpolation

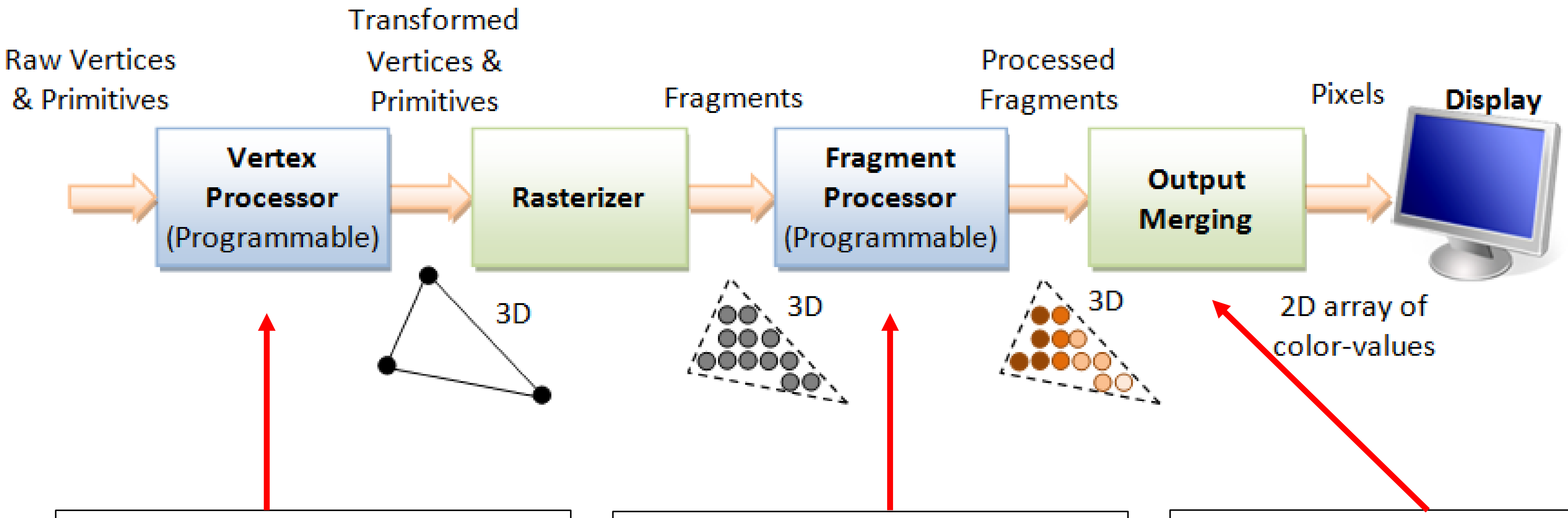
Texture image is larger



Minmaping

Texture Mapping

Review of the Graphics Pipeline



Vertex shader
- Vertex transforms
- Per-vertex lighting

Fragment shader
- Texturing
- Per-fragment lighting

Combine the fragments of all primitives into 2D color-pixel for display

Further Reading

- 3D graphics with OpenGL, Basic Theory https://www3.ntu.edu.sg/home/ehchua/programming/opengl/CG_BasicsTheory.html

- Textbook: Shirley and Marschner "Fundamentals of Computer Graphics", AK Peters, 2009