Course Updates

- Midterm should be graded by Wednesday

- Assignment 2
  - **Reminder:** Assignment 2 is to be done individually
  - **Theory** is due on Wednesday
  - **Programming** is due a week after Wednesday

- This weeks tutorial will concentrate on the programming portion of Assignment 2 (OpenGL)

Last time…

- Phong reflectance model

$$L(\bar{\boldsymbol{p}},\vec{\boldsymbol{c}}) = \boldsymbol{r}_d \boldsymbol{I}_d \max(0, \vec{\boldsymbol{s}} \cdot \vec{\boldsymbol{n}}) + \boldsymbol{r}_s \boldsymbol{I}_s \max(0, \vec{\boldsymbol{r}} \cdot \vec{\boldsymbol{c}})^{\alpha} + \boldsymbol{r}_a \boldsymbol{I}_a$$

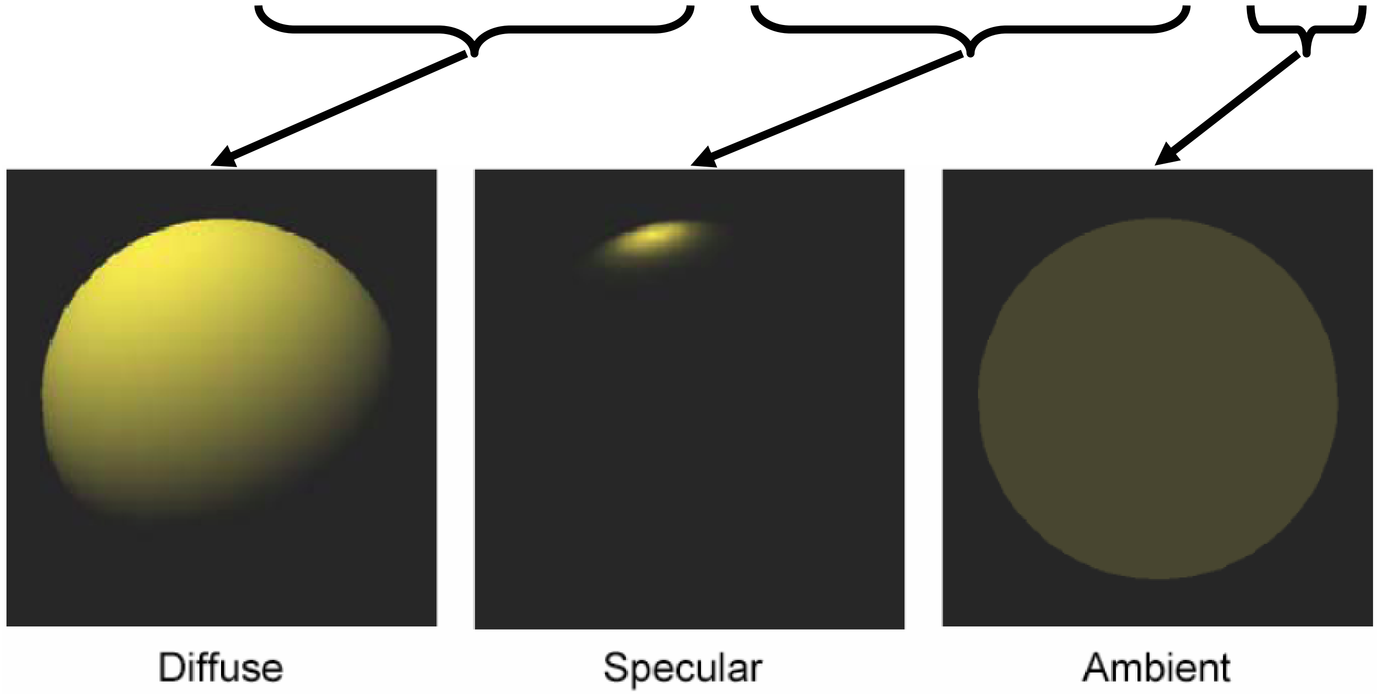


Image from lecture slides of David Breen

Shading

Computer Graphics, CSCD18

Fall 2007

Instructor: Leonid Sigal

Shading

- **Goal:** use light/reflectance model we derived last week to shade/color facets of polygonal mesh

- Last time
  - We know how to color a point on objects surface given a point, a normal at that point, a light source, and a camera position

- But
  - Geometry is not modeled using points (too expensive), it is modeled using polygonal meshes.

- This is why we need **shading**

Basic setup

- Assume we know

$\bar{e}^w$ - center of projection (position of eye) in world coordinates

$\bar{l}^w$ - position of the point light source in world coordinates

I_a, I_d, I_s - intensity of ambient, diffuse, and secular light sources

r_a, r_d, r_s - reflection coefficients of ambient, diffuse, and secular light sources

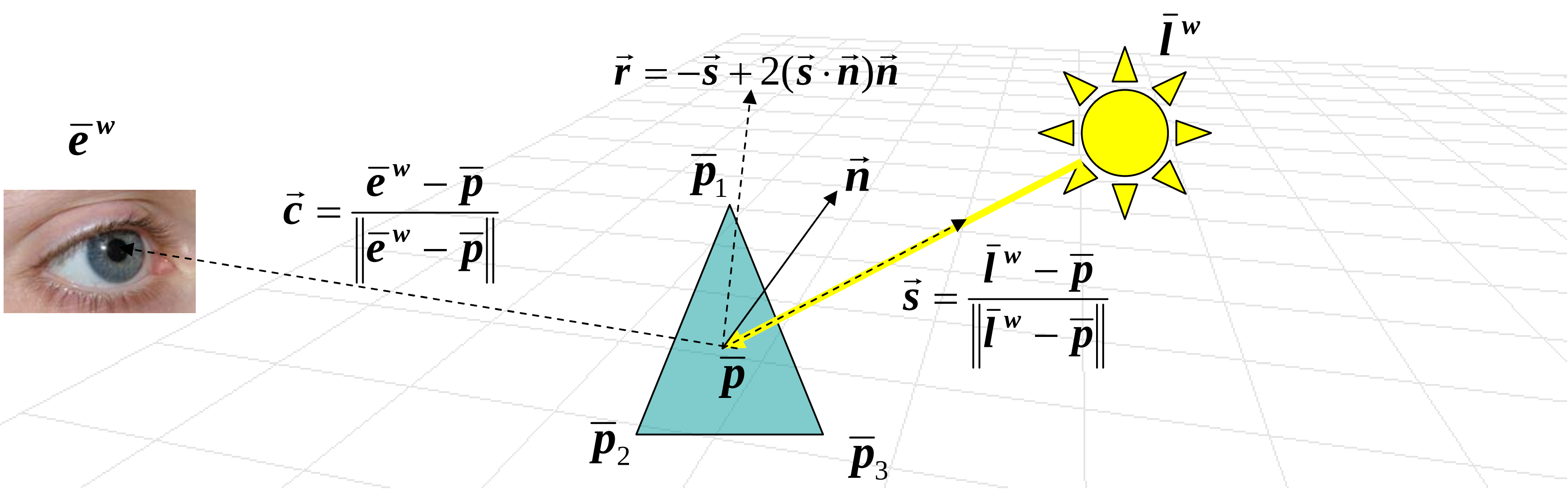
α - exponent controlling width of the specular highlight

- How do we shade a triangle?

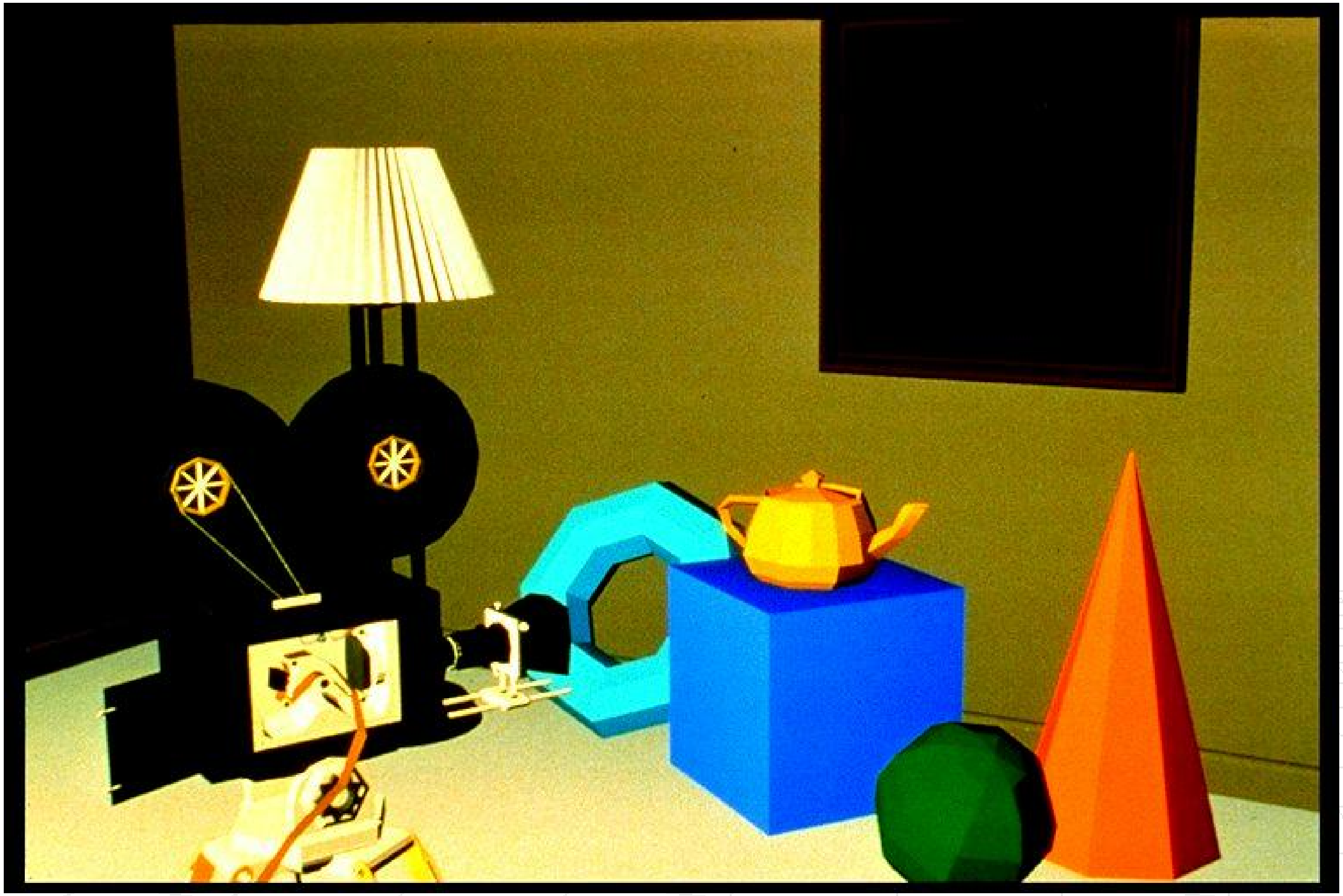
Flat Shading

- **Idea:** Fill each triangle with single color
- Let us assume we have a triangle with vertices $\boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3}$ in CCW order
- We can then compute the normal, $\vec{\boldsymbol{n}} = \frac{(\bar{\boldsymbol{p}}_2 - \bar{\boldsymbol{p}}_1)\times(\bar{\boldsymbol{p}}_3 - \bar{\boldsymbol{p}}_1)}{\left\|(\bar{\boldsymbol{p}}_2 - \bar{\boldsymbol{p}}_1)\times(\bar{\boldsymbol{p}}_3 - \bar{\boldsymbol{p}}_1)\right\|}$
- And shade entire triangle using Phong model

$$\boldsymbol{L}(\bar{\boldsymbol{p}}, \bar{\boldsymbol{e}}^w, \bar{\boldsymbol{l}}^w) = \boldsymbol{r}_d \boldsymbol{I}_d \max(0, \vec{\boldsymbol{s}} \cdot \vec{\boldsymbol{n}}) + \boldsymbol{r}_a \boldsymbol{I}_a + \boldsymbol{r}_s \boldsymbol{I}_s \max(0, \vec{\boldsymbol{r}} \cdot \vec{\boldsymbol{c}})^\alpha$$



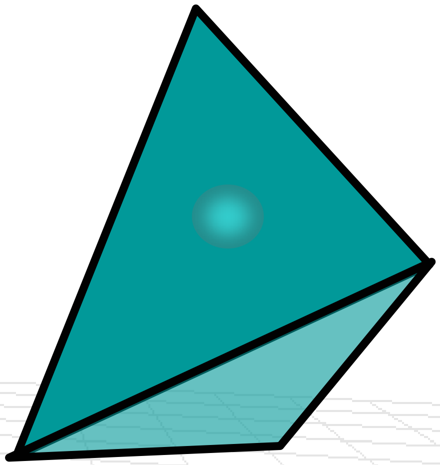
Flat Shading



Foley, van Dam, Feiner, Hughes, Plate II.29

Issues with Flat Shading

- For large faces secularities are impractical, since highlight is often sharp
  - Because of this, typically the secular term is dropped



- Mesh binderies are visible
  - People are very sensitive to this
- **Solutions**
  - Use small patches (but this is inefficient)
  - Use interpolative shading

Interpolative Shading

- **Idea:** Average intensities at vertices of the triangle to smoothly interpolate over pixels within a face

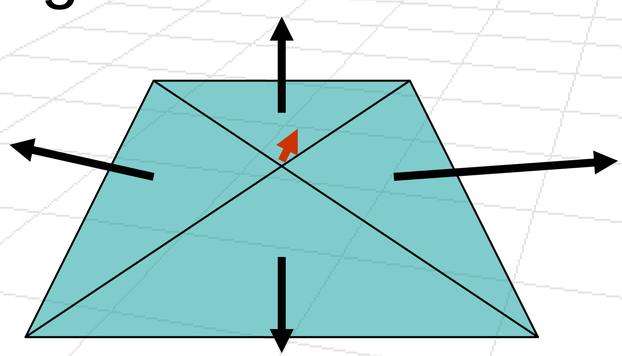
- **Algorithm**, for a triangular face with vertices $\bar{p}_1, \bar{p}_2, \bar{p}_3$
  - Compute normals at each vertex
  - Compute radiance $E_j = L(\bar{p}_j, \bar{e}^w, \bar{l}^w)$ for each vertex point $\bar{p}_j$

$$L(\bar{p}_j, \bar{e}^w, \bar{l}^w) = r_d I_d \max(0, \vec{s}_j \cdot \vec{n}_j) + r_a I_a + r_s I_s \max(0, \vec{r}_j \cdot \vec{c}_j)^\alpha$$

  - Project vertices onto image plane
  - Fill polygon by interpolating radiance along the triangle (scan conversion)

Interpolative Shading in Detail

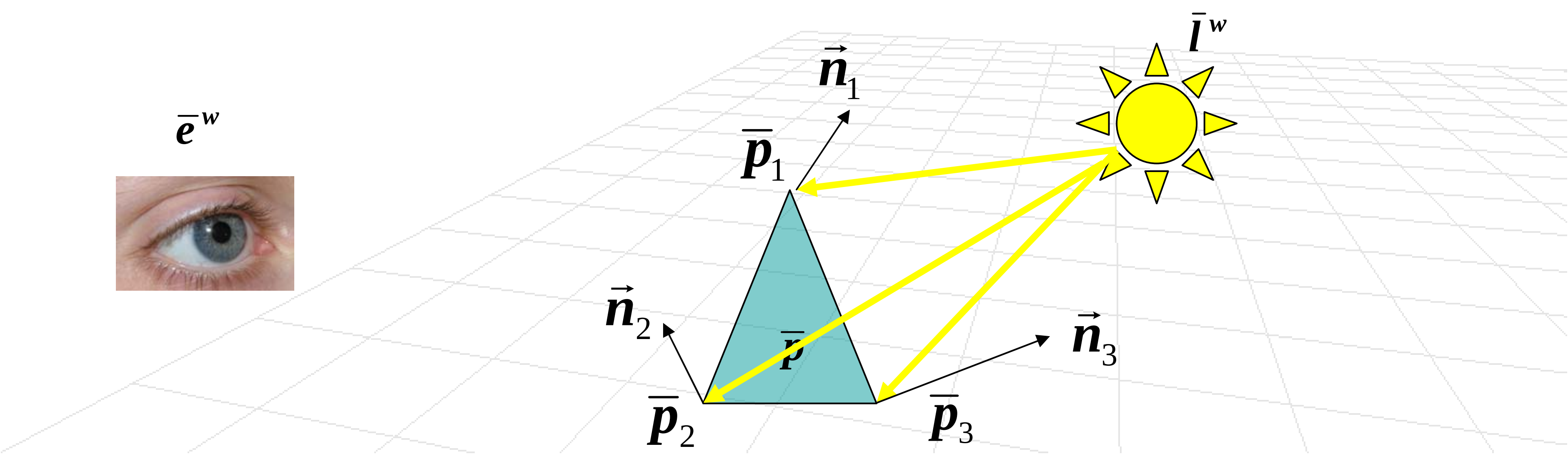
- Compute normals at each vertex
  - Many approaches are possible
  - Given parametric shape, compute normal $\vec{\boldsymbol{n}}_j$ when sampling vertices of the mesh $\overline{\boldsymbol{p}}_j$
    - Implicit form $\vec{\boldsymbol{n}}_j(\overline{\boldsymbol{p}}_j) = \nabla \boldsymbol{f}(\overline{\boldsymbol{p}}_j)$
    - Explicit form $\vec{\boldsymbol{n}}_j(\overline{\boldsymbol{p}}_j) = \left.\frac{\partial \mathbf{s}(\alpha,\beta)}{\partial \alpha}\right|_{\alpha_0,\beta_0} \times \left.\frac{\partial \mathbf{s}(\alpha,\beta)}{\partial \beta}\right|_{\alpha_0,\beta_0}$
  - Let $\vec{\boldsymbol{n}}_j$ be average of "face normals" of all adjacent faces



Interpolative Shading in Detail

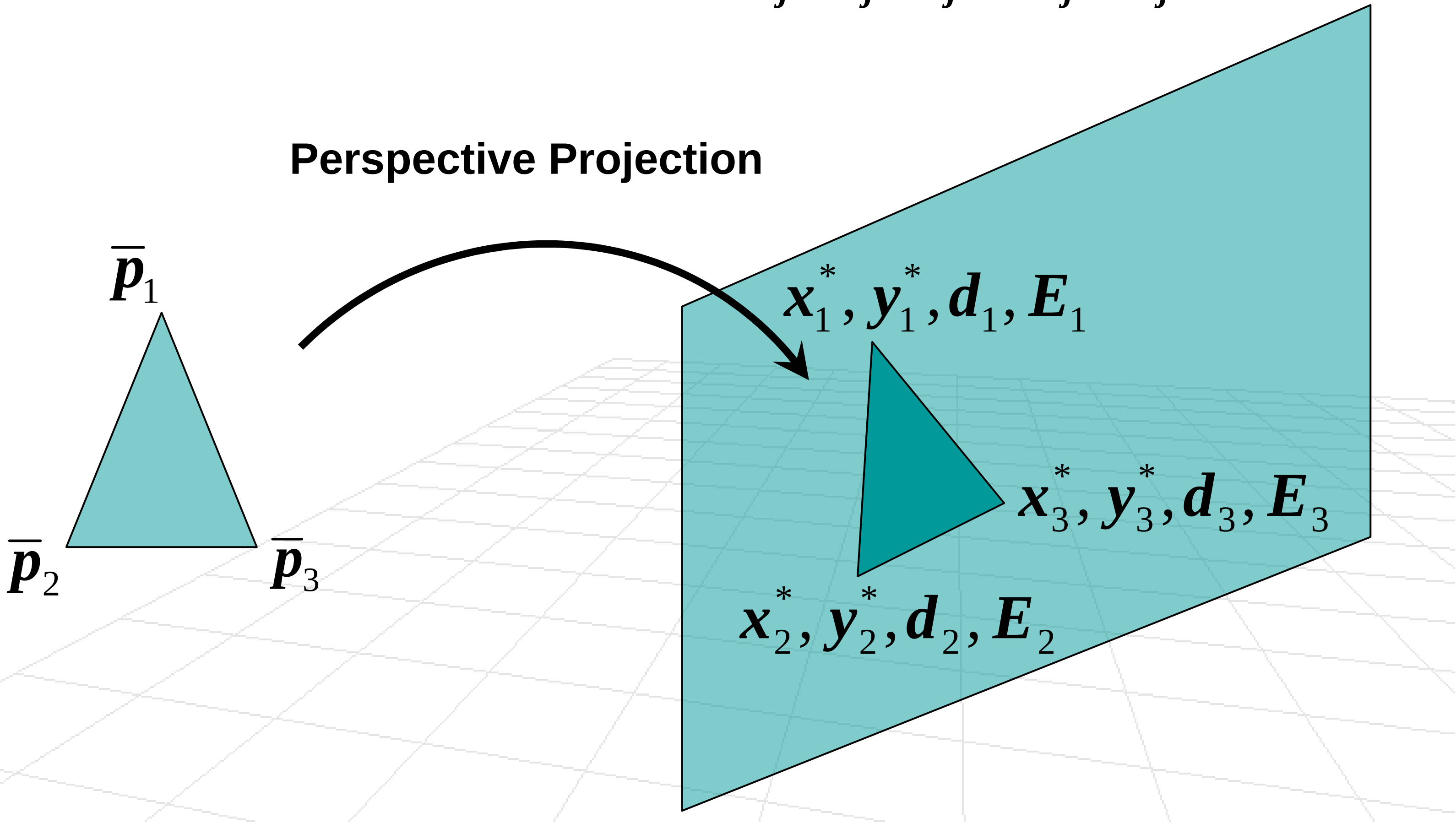
- Compute radiance E_j for each vertex point p_j
  - Same as in flat shading (using Phong model)

$$E_j = L(\bar{p}_j, \bar{e}^w, \bar{l}^w) = r_d I_d \max(0, \vec{s}_j \cdot \vec{n}_j) + r_a I_a + r_s I_s \max(0, \vec{r}_j \cdot \vec{c}_j)^\alpha$$



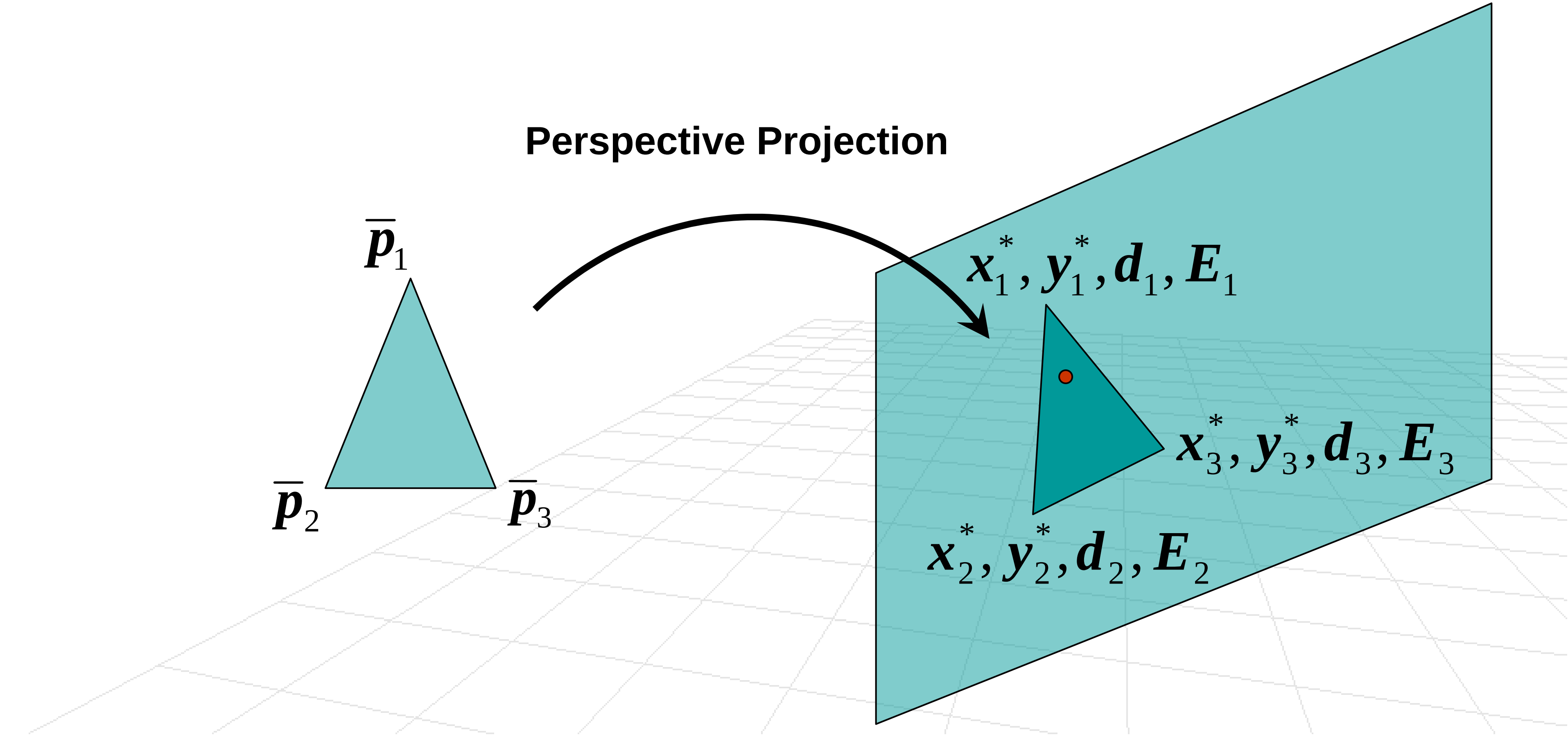
Interpolative Shading in Detail

- Project onto image plane with pseudodepth
  - So for each vertex we have $x^*_j, y^*_j, d_j = z^*_j, E_j$



Interpolative Shading in Detail

- Scan conversion with linear interpolation of both pseudodepth (d_j) and radiance values (E_j)
  - use z-buffer to handle visibility



Algorithm (part 1)

- For each edge between (x_b^*, y_b^*, d_b, E_b) and (x_a^*, y_a^*, d_a, E_a) ordered such that $y_a^* > y_b^*$

$$x = x_b^*, \quad \Delta x = (x_a^* - x_b^*)/(y_a^* - y_b^*)$$

$$d = d_b, \quad \Delta d = (d_a - d_b)/(y_a^* - y_b^*)$$

$$E = E_b, \quad \Delta E = (E_a - E_b)/(y_a^* - y_b^*)$$

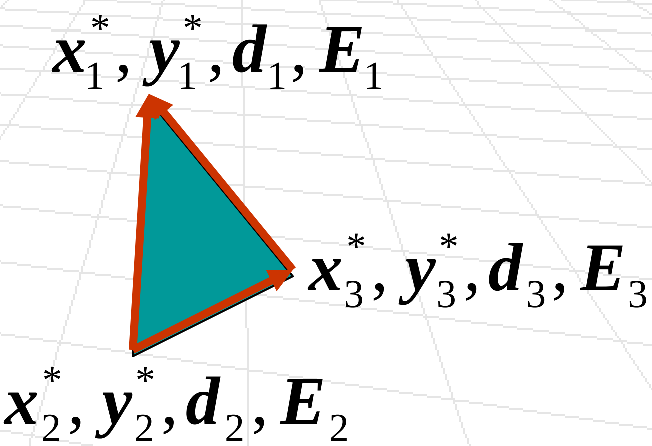
  - For (y=y_b; y<y_a; y++)

    Place (x, d, E) in active edge list (AEL) at scanline y

$$x = x + \Delta x$$

$$d = d + \Delta d$$

$$E = E + \Delta E$$



Algorithm (part 1)

- For each scanline between $\min(y_1, y_2, y_3)$ and $\max(y_1 . y_2, y_3)$
  - Extract (x_a, d_a, E_a) and (x_a, d_a, E_a) from AEL where $x_a > x_b$

    $$d = d_b, \quad \Delta d = (d_a - d_b)/(x_a - x_b)$$

    $$E = E_b, \quad \Delta E = (E_a - E_b)/(x_a - x_b)$$

  - For ($x=x_b$; $x<x_a$; x++)

    if (d < z-buffer(x, y))

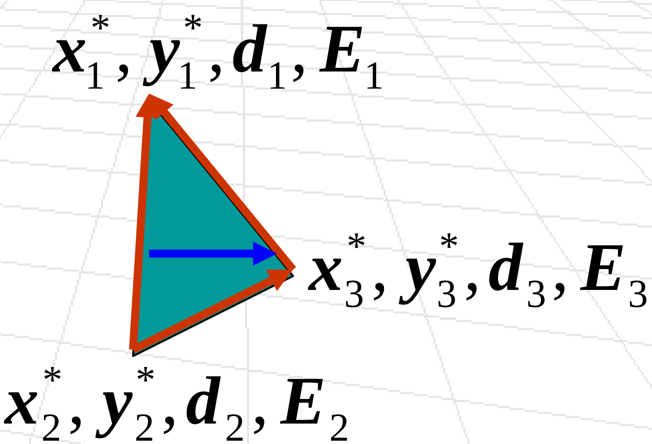
    putpixel(x, y, E)

    z-buffer(x, y) = d

    end

    $$d = d + \Delta d$$

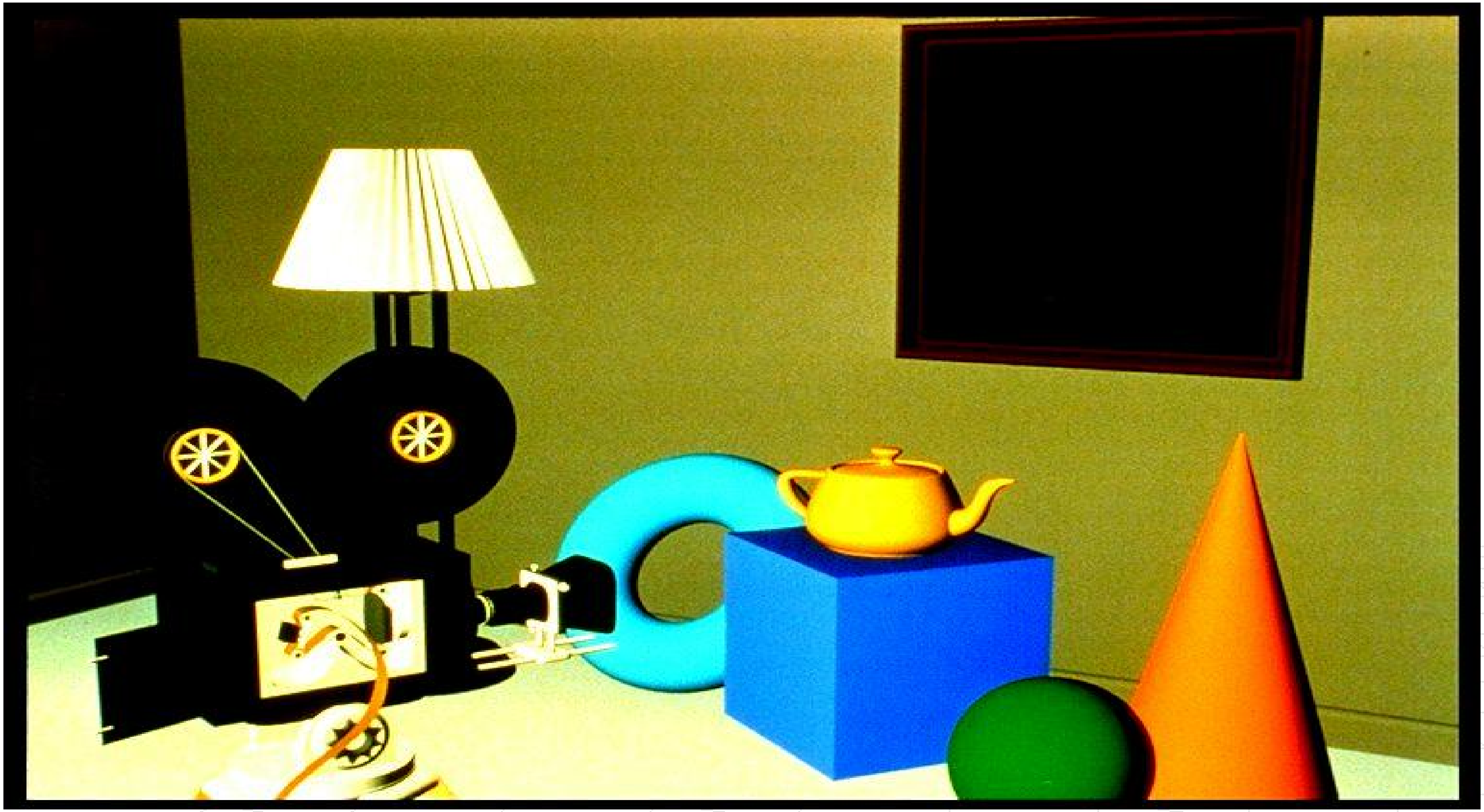
    $$E = E + \Delta E$$



Interpolative Shading in Detail

- What we just described is so called **Gouraud Shading**
- Advantages
  - Does not produce artifacts at face boundaries (i.e. better then flat shading)
- Disadvantages
  - Still hard to handle secular highlights. **Why?**

Gouraud Shading



Foley, van Dam, Feiner, Hughes, Plate II.30

Phong Shading

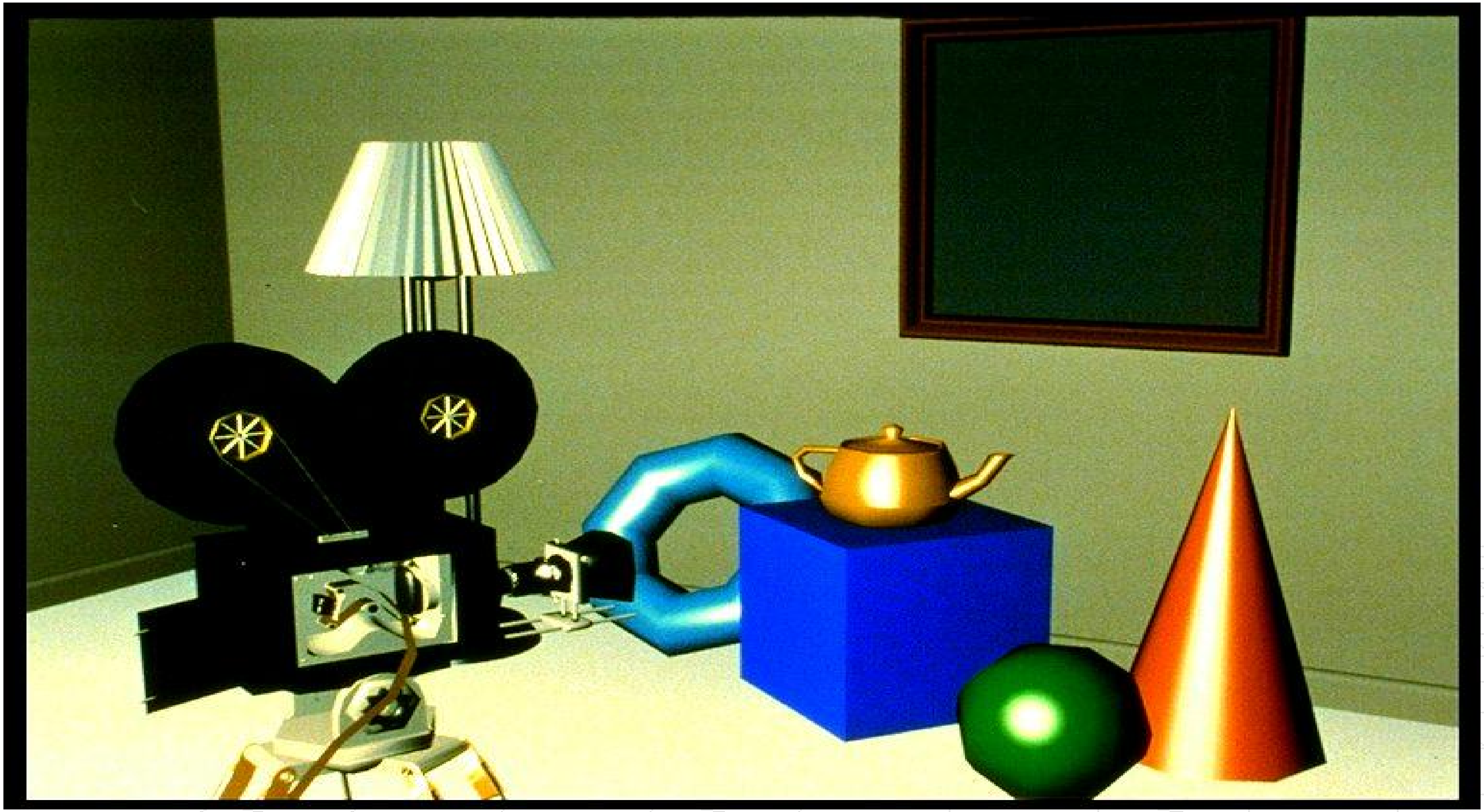
- Note that **phong shading** and **phong lighting** are not one and the same.

- **Idea:** Slightly modify the Gouraud shading algorithm to correctly shade every pixel (with secularities)

- **Algorithm**, for a triangular face with vertices $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$
  - Compute normals at each vertex
  - For each point on a triangle that corresponds to a pixel location interpolate the normal
  - Compute radiance E_j for each pixel in the projected triangle that corresponds to point within the world triangle
  - Project vertices onto image plane

- Why is this batter then just doing Phong lighting?

Phong Shading



Foley, van Dam, Feiner, Hughes, Plate II.32

Phong Shading

- Advantages
  - Produces very accurate shading with specular highlights (better then flat shading and Gouraud shading)
- Disadvantages
  - It's computationally expensive (but not on current graphics hardware)

Texture Mapping

Computer Graphics, CSCD18

Fall 2007

Instructor: Leonid Sigal

Texture Mapping

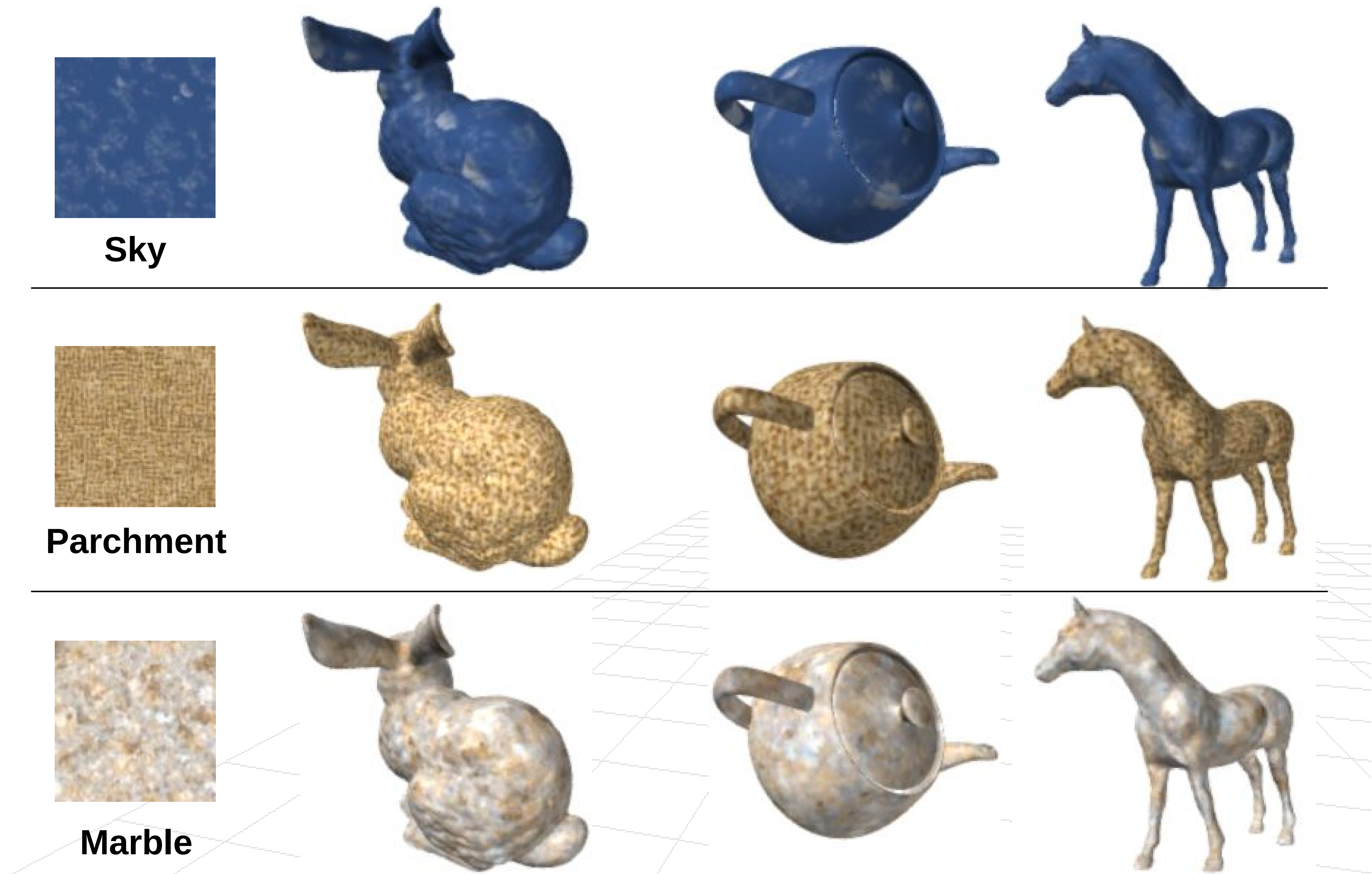
- So far we only considered objects that have consistent color
- If we want to have more realistic variations in reflectance that conveys textures we need to model them
- There are two natural sources of textures
  - **Surface markings** – variations in the total light reflected
  - **Surface relief** – variations in 3D shape which introduce local variability in shading


Why do we need textures?

- An alternative would be to have much more complex models
  - This is expensive computationally
  - The tools for building such high fidelity models are not readily available
- Textures
  - Cheaper to render (especially on current graphics hardware)
  - Reusable
    - Once we have the texture (e.g. wood) we can use it for many different objects

Texture Mapping Examples

From http://www.cs.ualberta.ca/~yang/Projects/texture_analysis_and_synthesis.htm



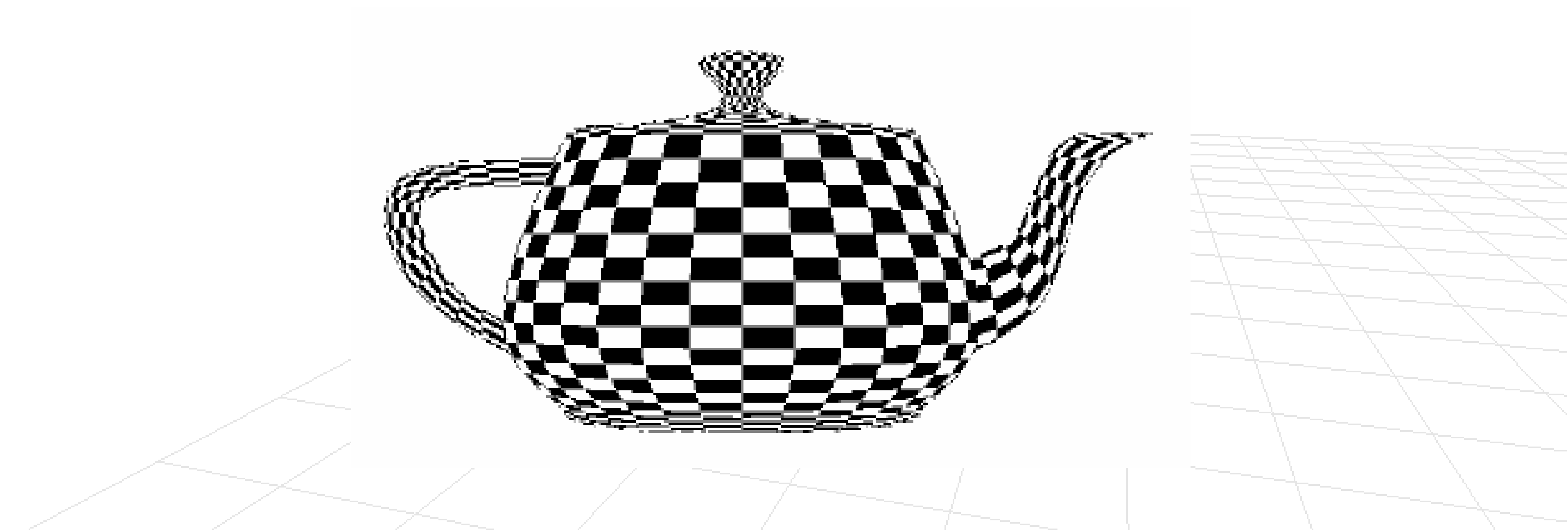
Questions we must address

- Where do textures come from?
- How do we map texture onto a surface?
- How does texture change reflectance properties and shading of the surface
- Scan conversion (how do we actually render texture mapped surface)

Where do we get a texture?

- Textures can be defined procedurally
  - **Input:** point on the surface
  - **Output:** surface albedo at that point

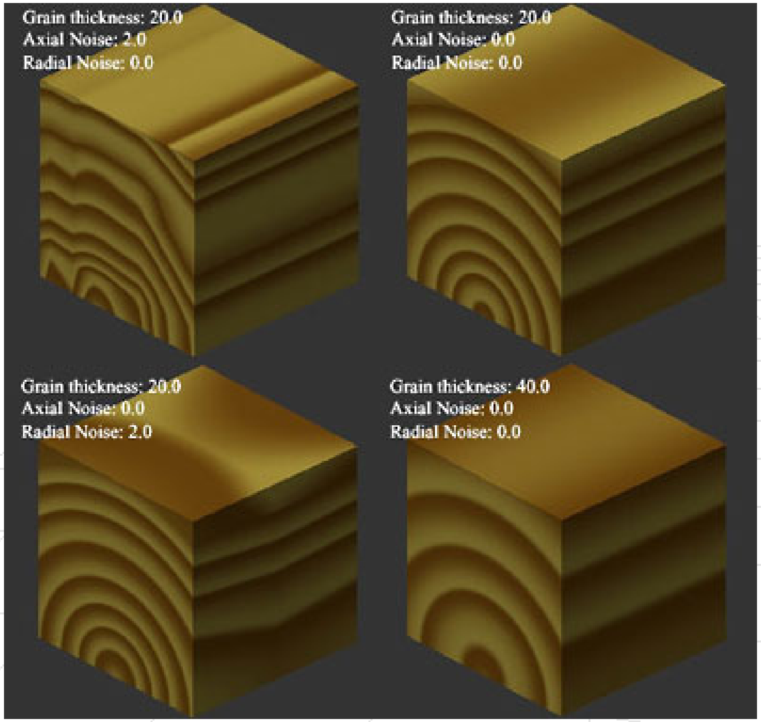
- Example of procedural texture



Where do we get a texture?

- Textures can be defined procedurally
  - **Input:** point on the surface
  - **Output:** surface albedo at that point

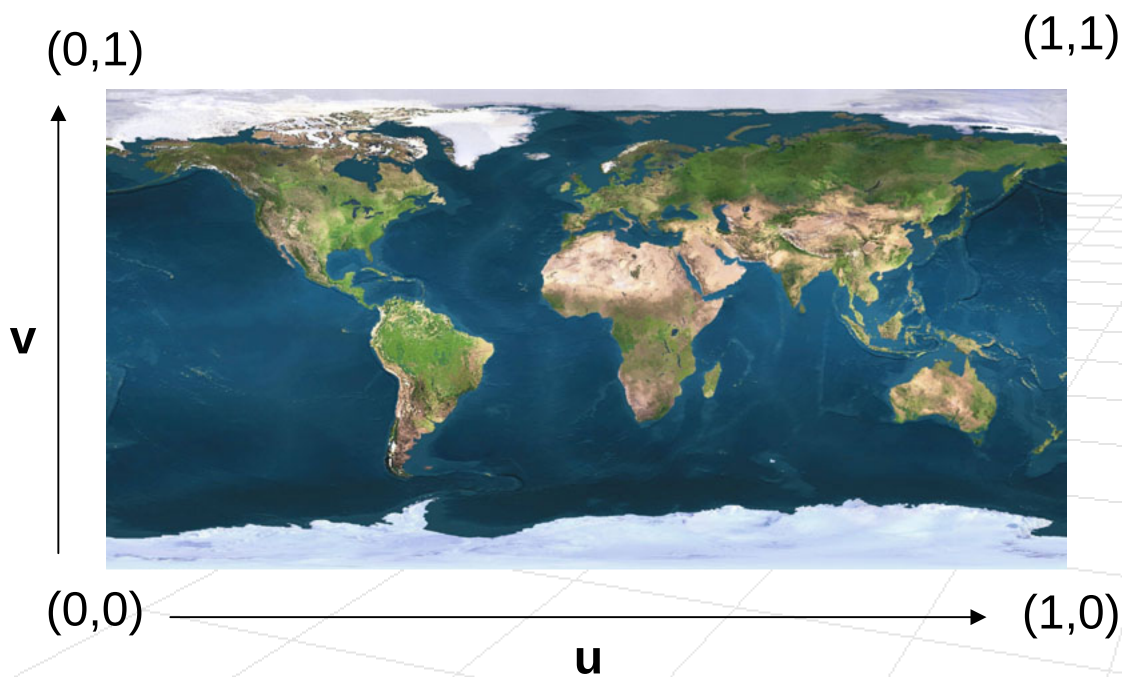
- Example of procedural texture (in 3D)



Where do we get a texture?

- We can also use digital images as textures
  - Imagine gluing a 2D picture over a 3D surface

- How do we do this?
  - map a point on the arbitrary geometry to a point on an abstract unit square (we call this texture space)
  - map a point on abstract unit square to a point on the image of arbitrary dimension



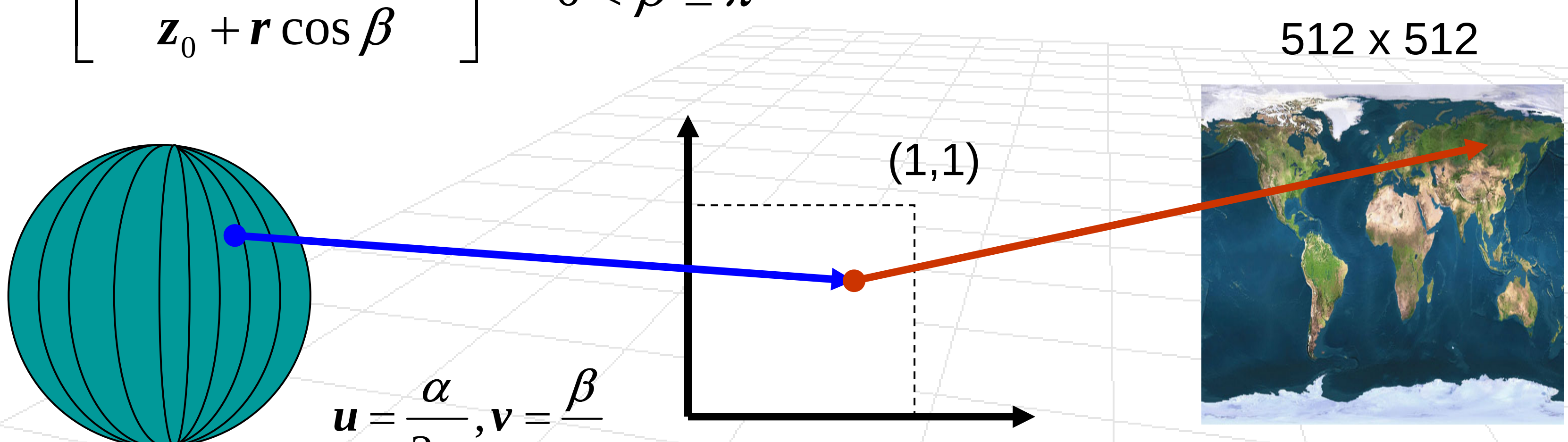
Texture Mapping Details

- Simplest approaches to texture mapping
  - For each face of the mesh, specify a point ($\boldsymbol{u_i}$, $\boldsymbol{v_i}$) for each vertex point $\boldsymbol{p_i}$
  - Continuous mapping from parametric form of the surface onto texture, for example for sphere

Texture Mapping Details

- Simplest approaches to texture mapping
  - For each face of the mesh, specify a point ($\boldsymbol{u_i}$, $\boldsymbol{v_i}$) for each vertex point $\boldsymbol{p_i}$
  - Continuous mapping from parametric form of the surface onto texture, for example for sphere

$$\boldsymbol{s}(\alpha,\beta)=\begin{bmatrix} \boldsymbol{x}_0+\boldsymbol{r}\cos\alpha\sin\beta \\ \boldsymbol{y}_0+\boldsymbol{r}\sin\alpha\sin\beta \\ \boldsymbol{z}_0+\boldsymbol{r}\cos\beta \end{bmatrix}, \quad \begin{matrix} 0\le\alpha\le 2\pi \\ 0<\beta\le\pi \end{matrix}$$



$$\boldsymbol{u}=\frac{\alpha}{2\pi},\boldsymbol{v}=\frac{\beta}{\pi}$$

Texture Mapping

- Texture mapping is also a great way to create artificial objects