

# Shading and GLSL

EDAF80

Michael Doggett



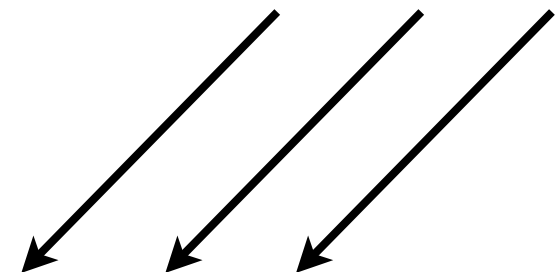
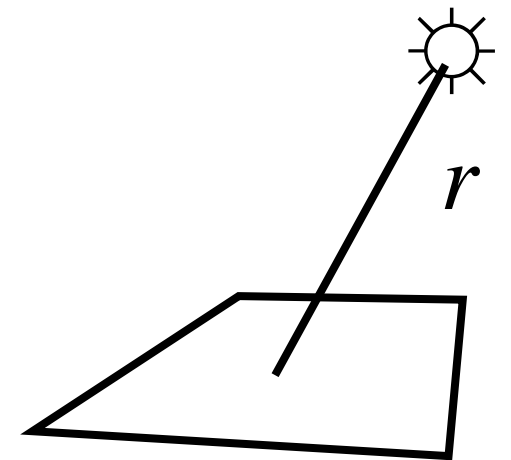
Slides by Jacob Munkberg 2012-13

# Today

- Lighting and Shading
- The Phong shading model
- Texture Mapping

# Light Sources

- Ambient light
  - Equal intensity at all points in scene
- Point light
  - Intensity falloff with distance:  $1/r^2$
- Spotlights
  - Limit point light to a cone
- Directional lights (sun)
  - All light rays parallel



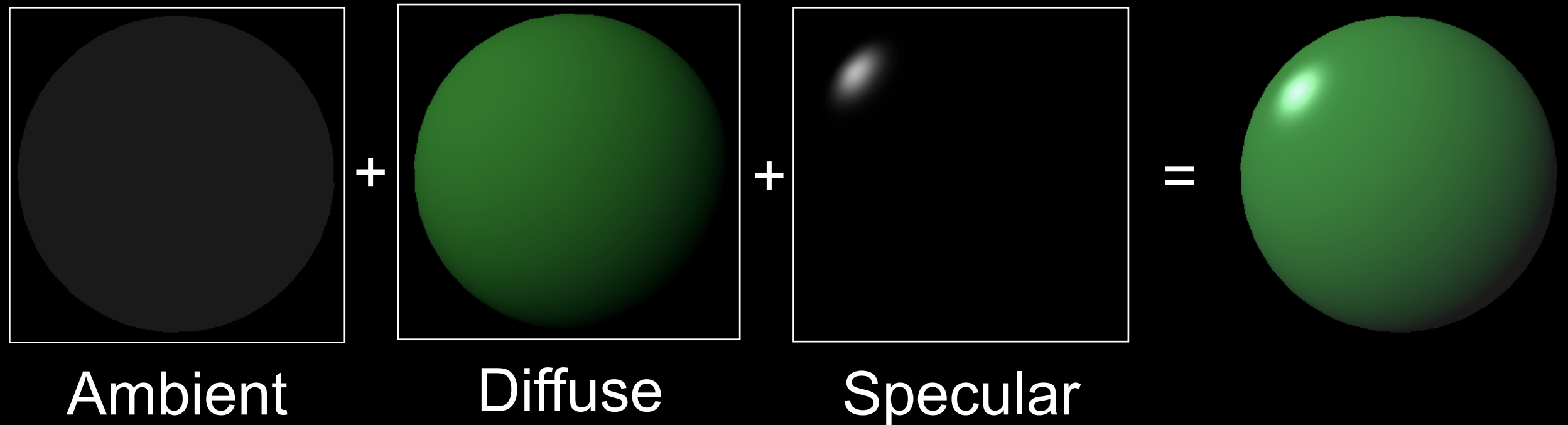
# Shading Theory

- Describe how surfaces interact (reflect/refract/absorb) light
- Light-material interactions:
  - Specular surfaces (mirrors, shiny metals) reflect light in narrow cone
  - Diffuse surfaces (rough metal, paper) reflect incoming light in wide cone
  - Translucent surfaces (glass, water, marble, skin) Some light penetrates the surface

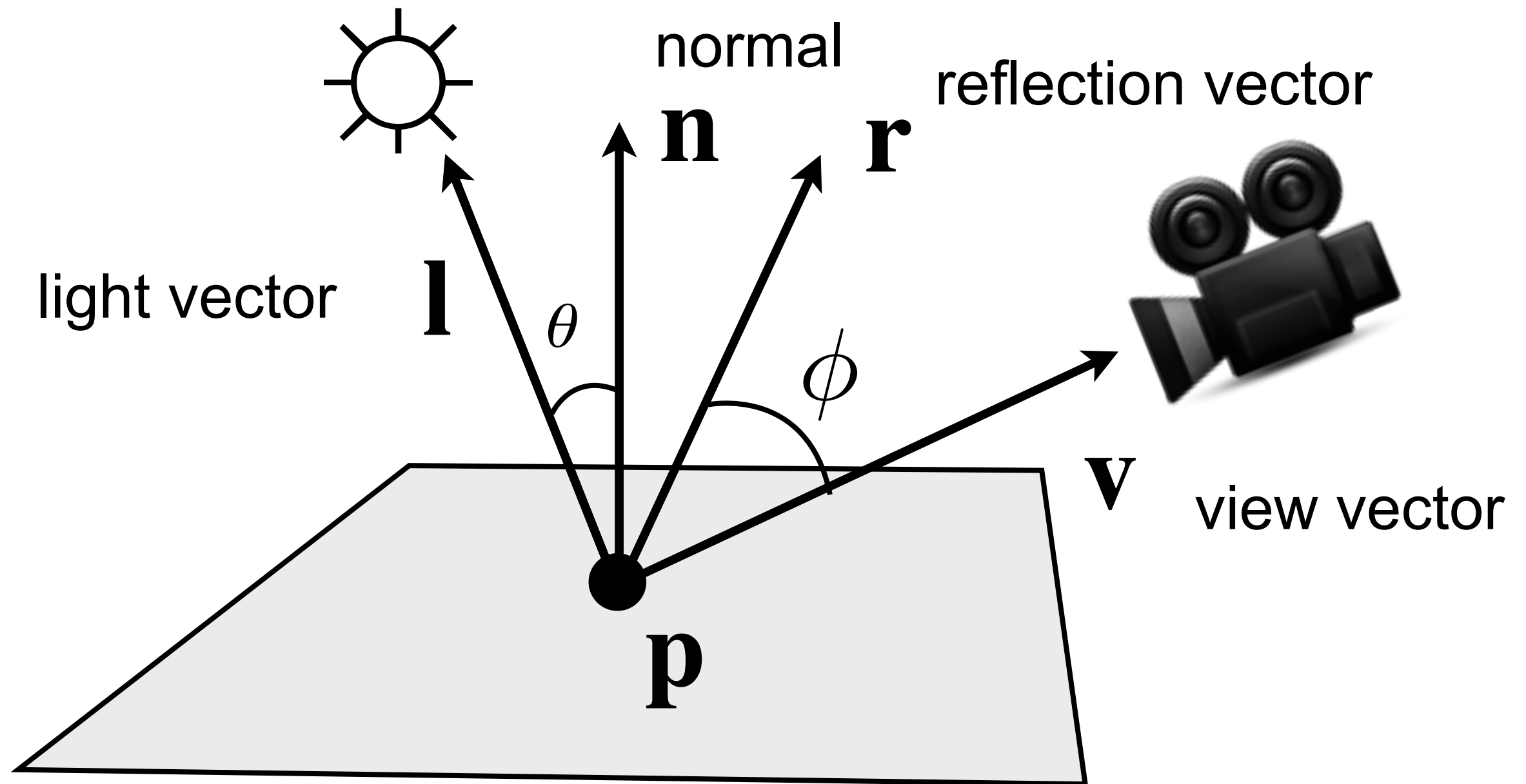


# Phong Reflection Model

- Efficient approximation of materials
  - Very popular in real-time graphics
- Describe materials as a sum of three types of reflection terms:

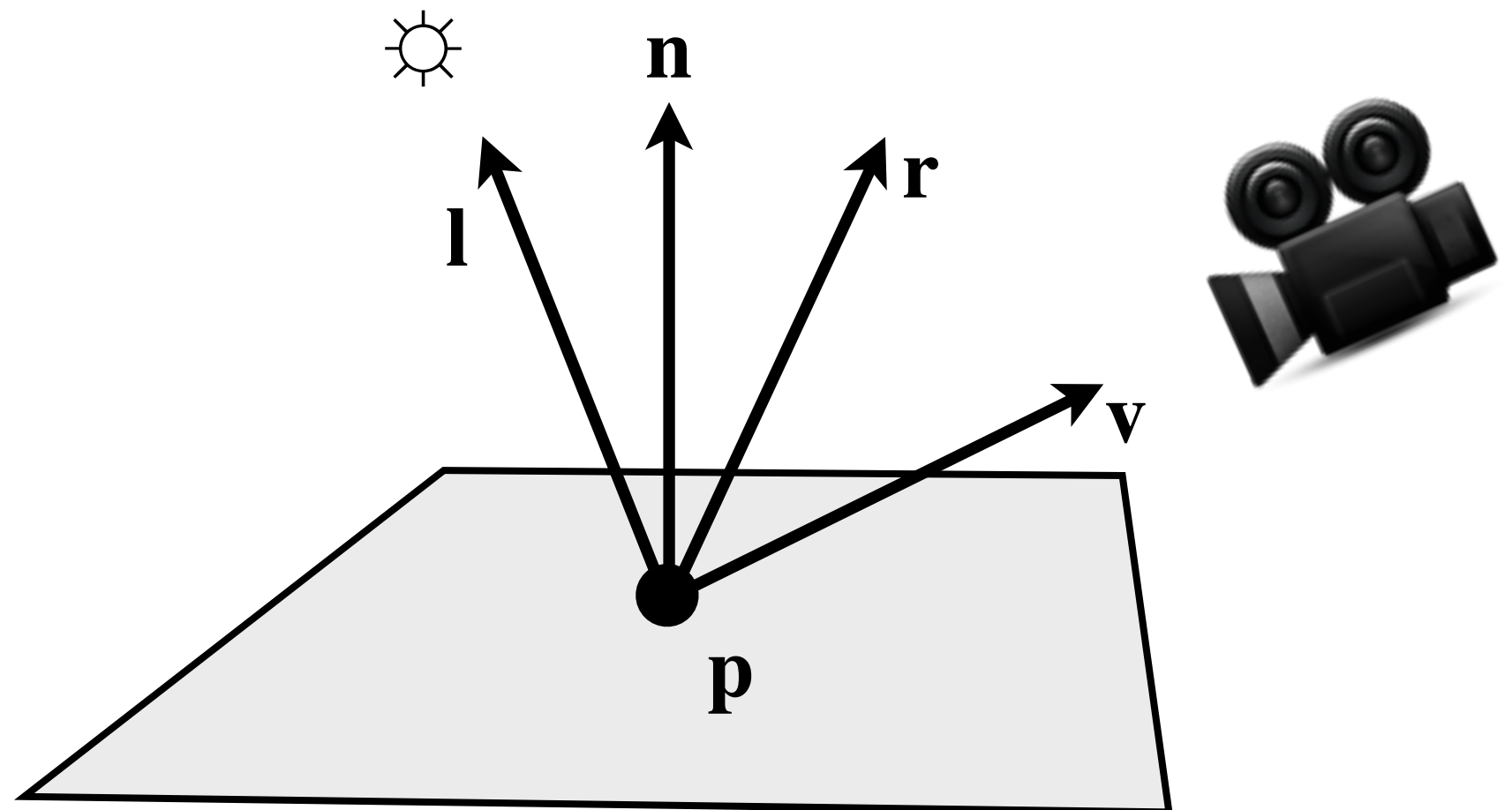


# Phong Shading



# Phong Reflection Model

- Ambient: constant
- Diffuse:  $\propto \mathbf{n} \cdot \mathbf{l}$
- Specular  $\propto (\mathbf{r} \cdot \mathbf{v})^\alpha$

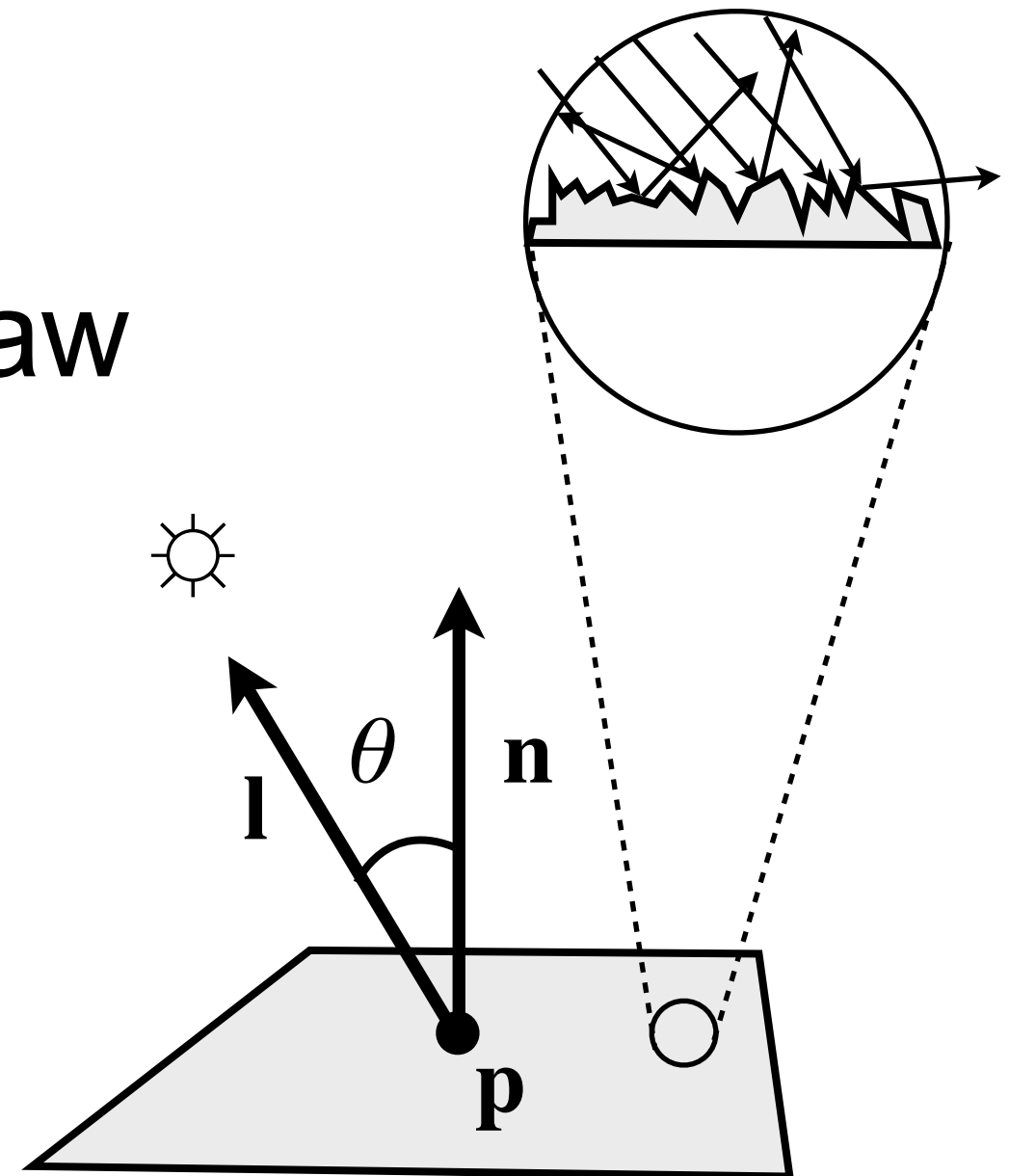


# Diffuse (Lambertian) Term

- Rough surfaces scatter light in all directions
- Model with Lambert's law

$$R_d \propto \cos \theta = \mathbf{l} \cdot \mathbf{n}$$

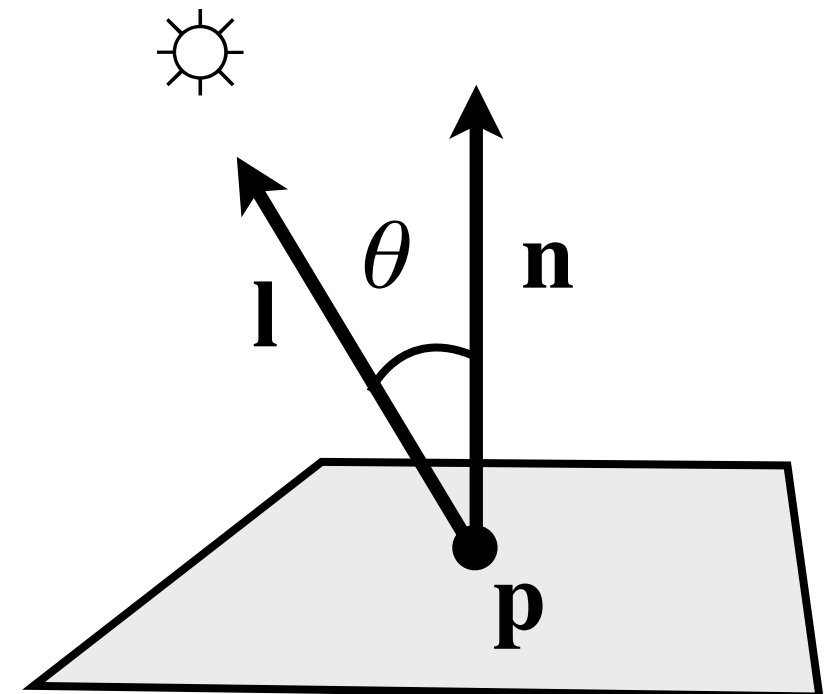
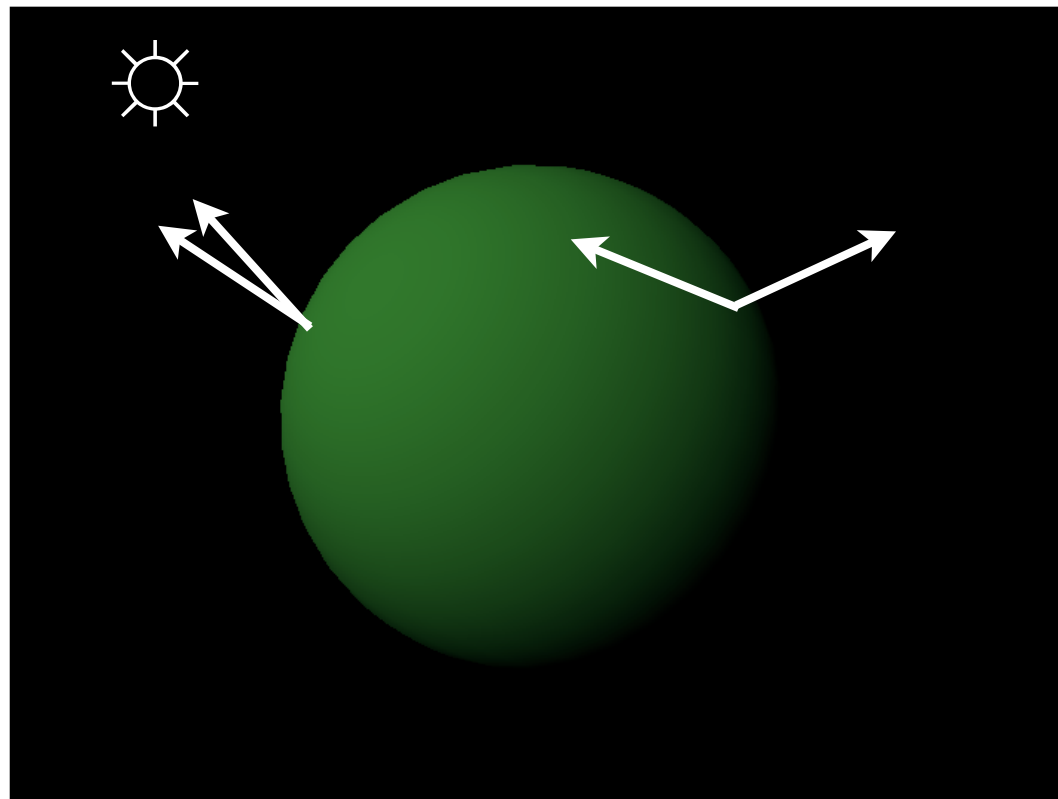
$$|\mathbf{l}| = |\mathbf{n}| = 1$$



# Diffuse (Lambertian) Term

- Let  $k_d$  represent the fraction of incoming diffuse light  $L_d$  that is reflected. The diffuse reflection term is:

$$I_d = k_d (\mathbf{l} \cdot \mathbf{n}) L_d$$

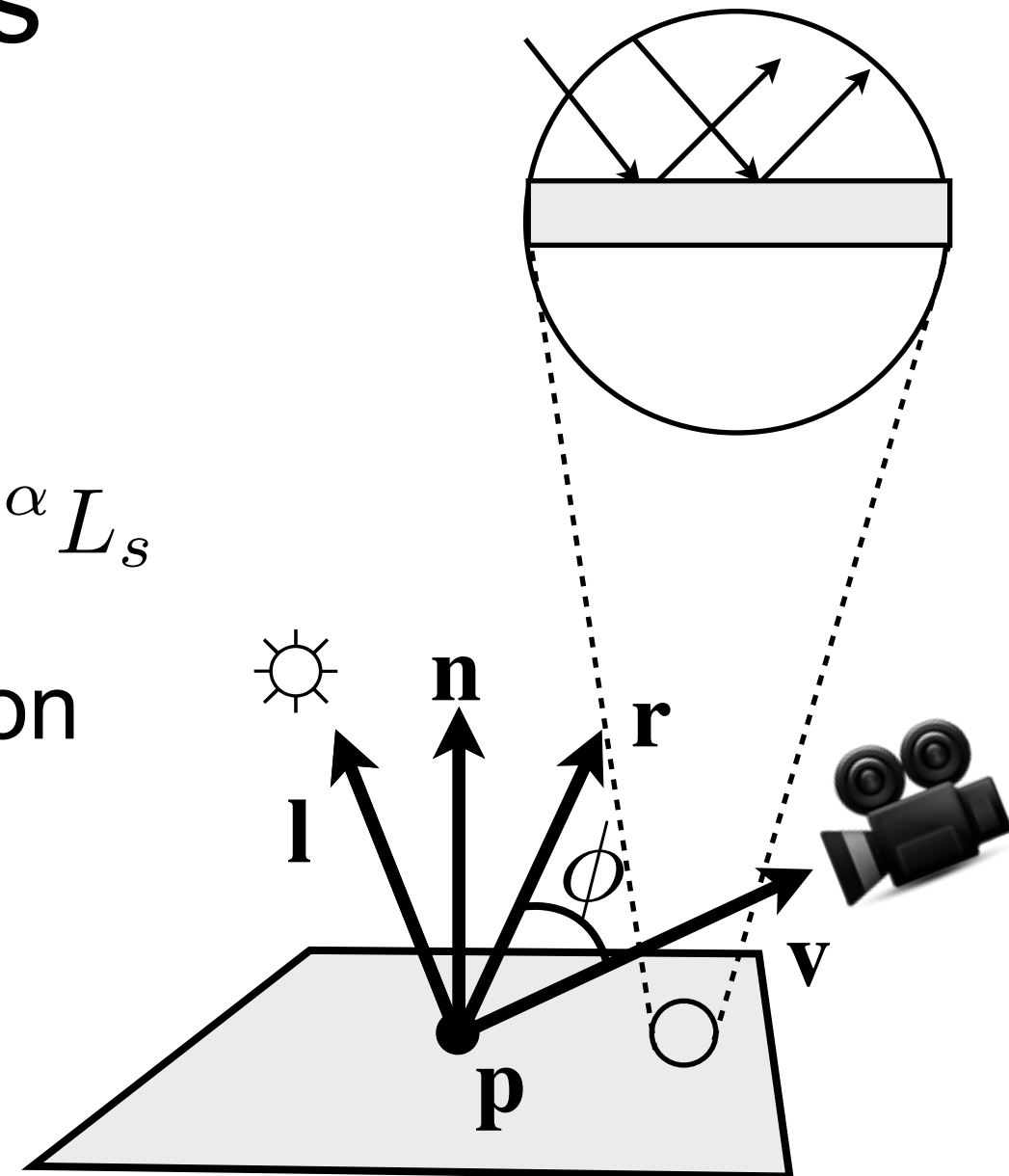


# Specular Reflection

- Smooth surface reflects light in one direction
- Phong specular term

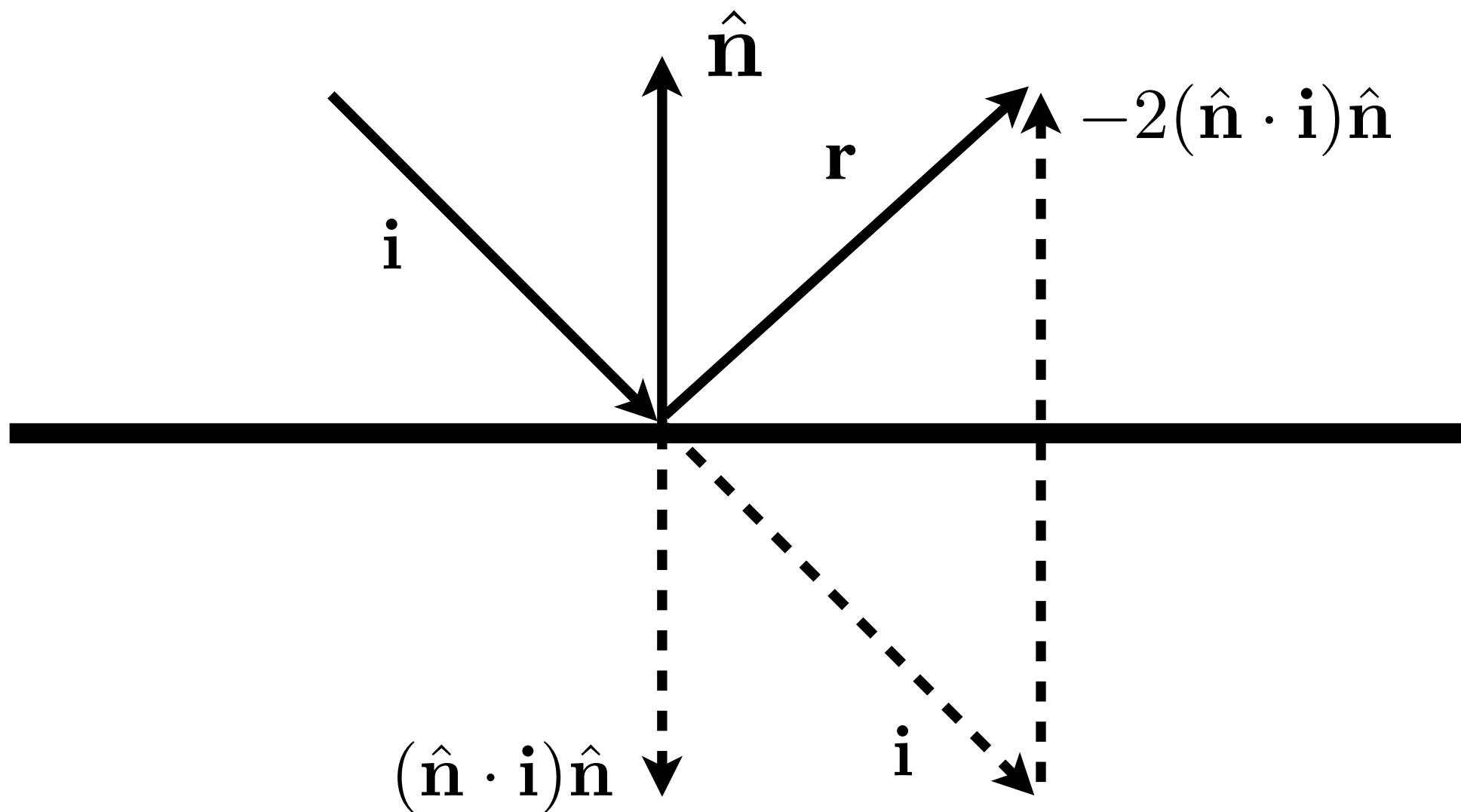
$$I_s = k_s (\cos \phi)^\alpha L_s = k_s (\mathbf{r} \cdot \mathbf{v})^\alpha L_s$$

- where  $k_s$  represent the fraction of incoming specular light  $L_s$  that is reflected.  $\alpha$  represent shininess



# Reflection operator

$$\mathbf{r} = \mathbf{i} - 2(\hat{\mathbf{n}} \cdot \mathbf{i})\hat{\mathbf{n}}$$





# Specular Reflection

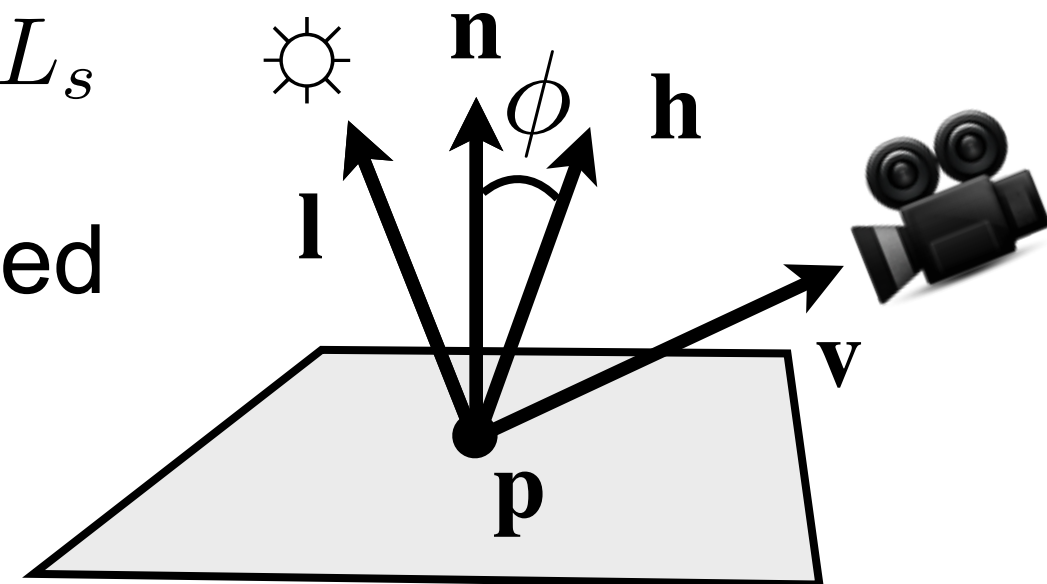
- Alternative specular term:

- Use half vector:  $\mathbf{h} = \frac{\mathbf{l} + \mathbf{v}}{|\mathbf{l} + \mathbf{v}|}$

- Alternative Phong specular term

$$I_s = k_s (\cos \phi)^\alpha L_s = k_s (\mathbf{n} \cdot \mathbf{h})^\alpha L_s$$

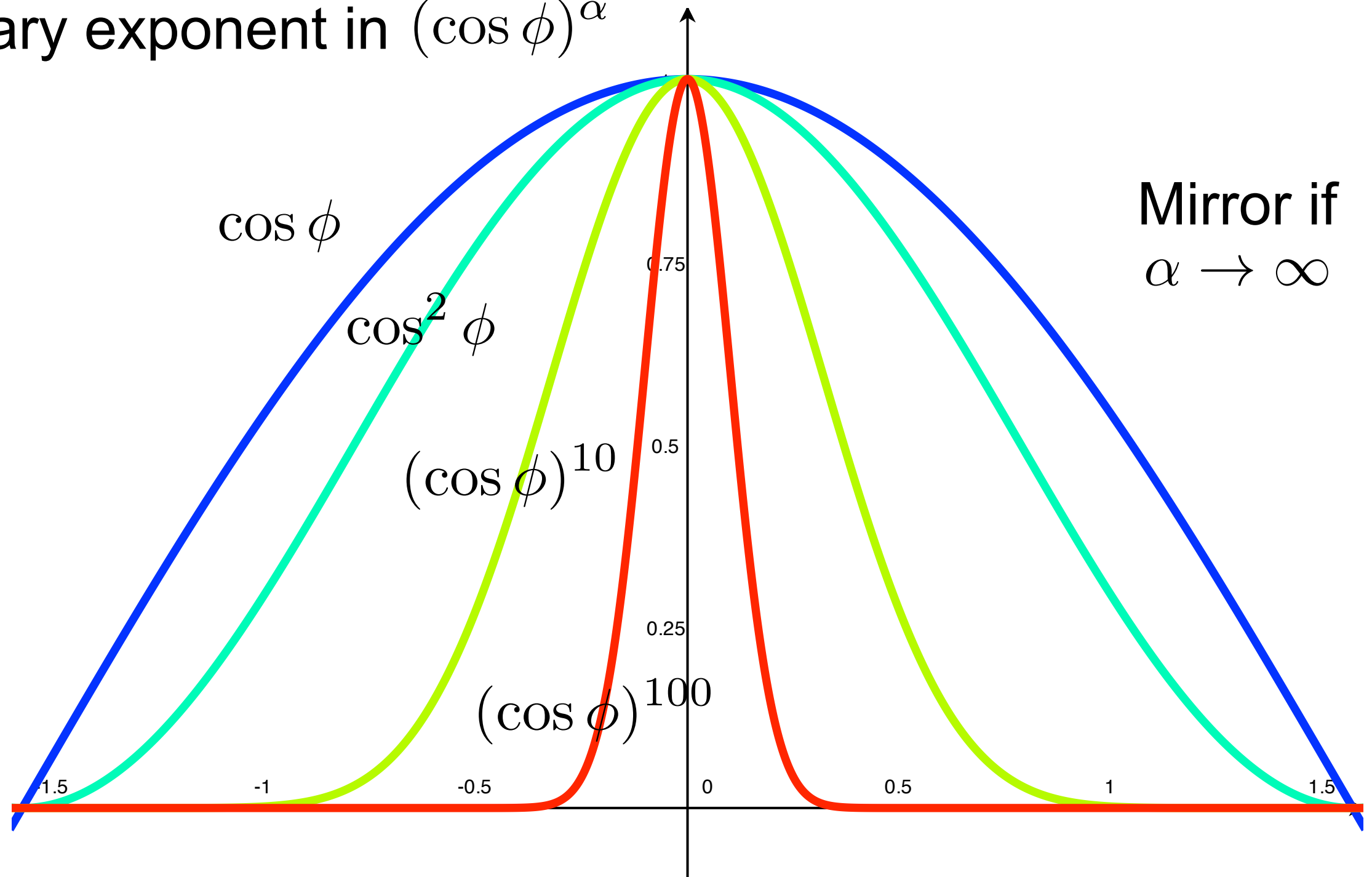
- No reflection operation needed
- Commonly used, called “Blinn-Phong” specular



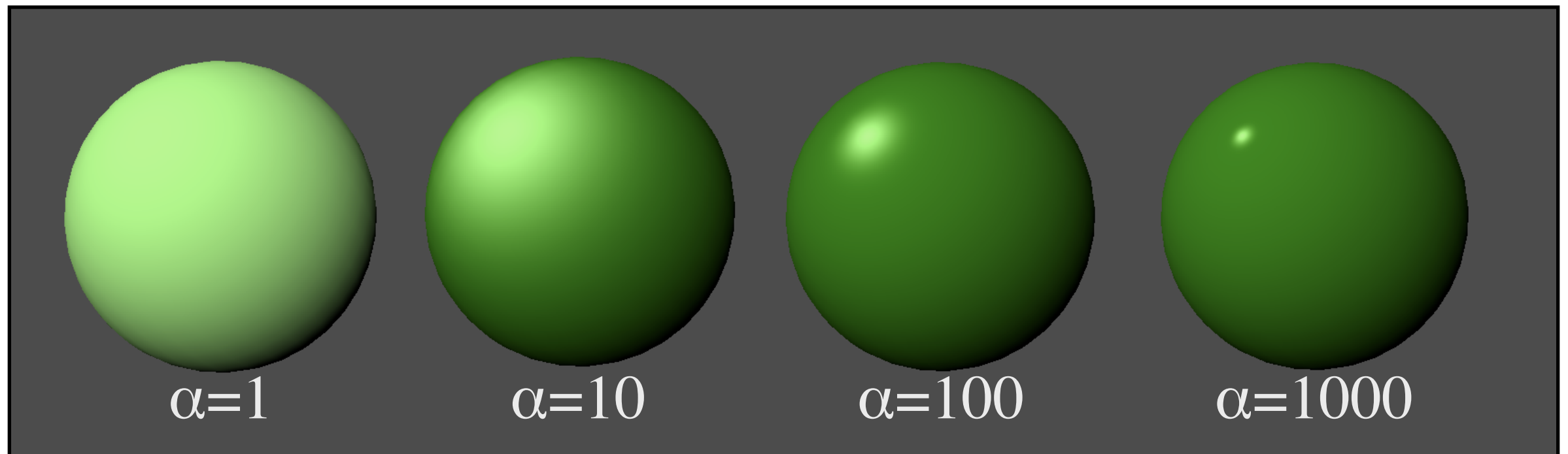
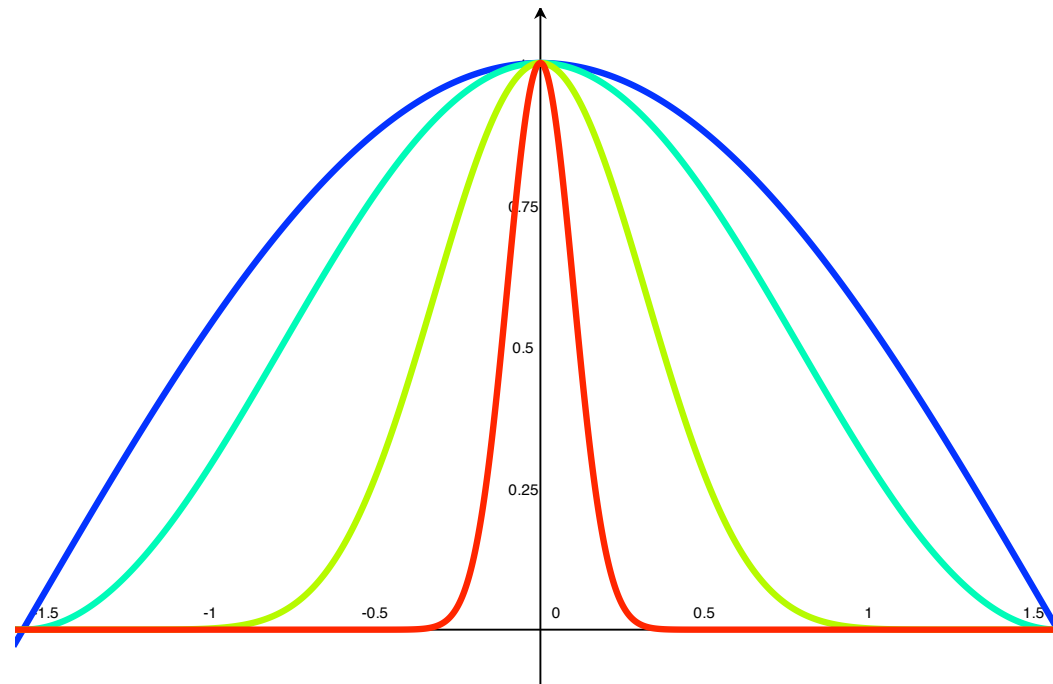


# Shininess

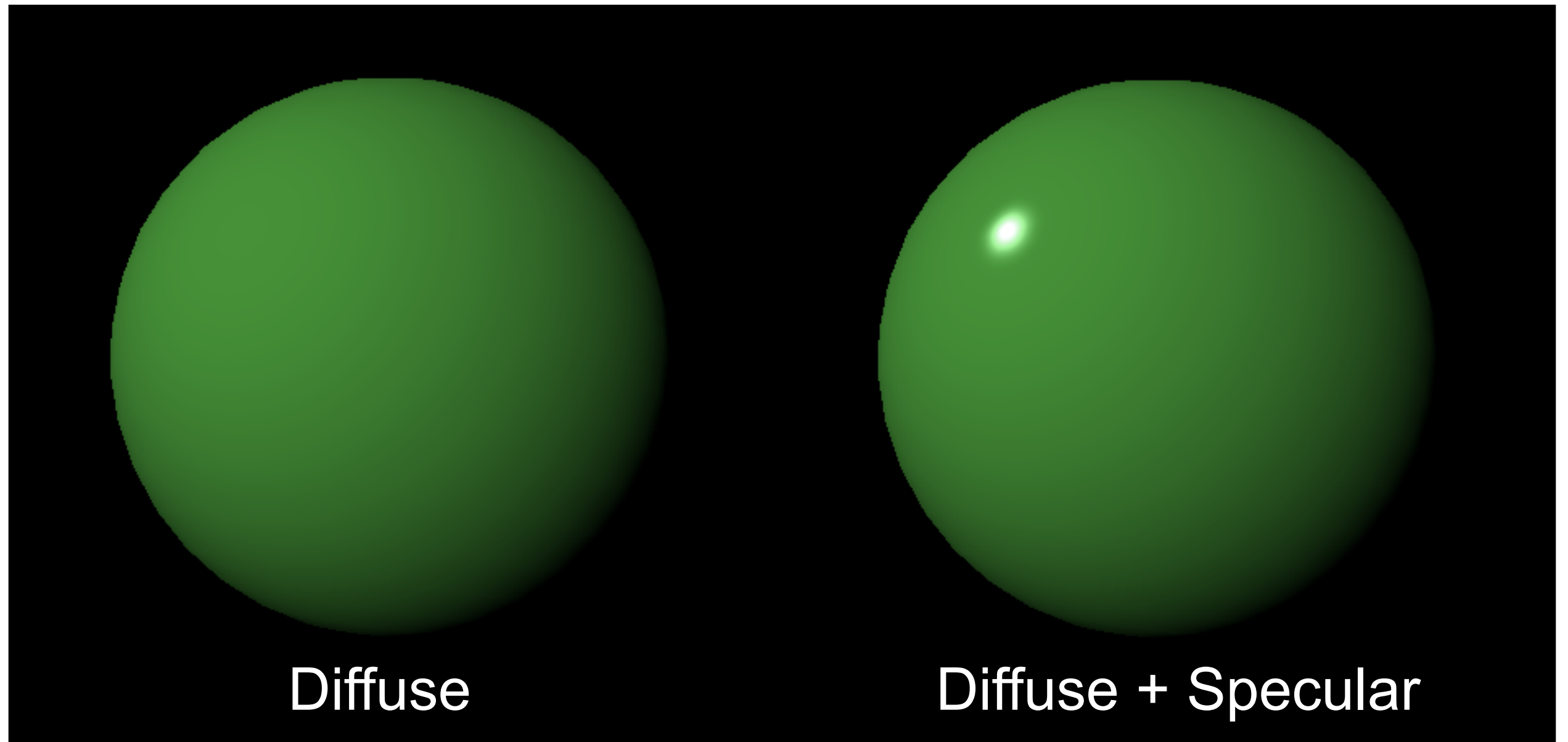
Vary exponent in  $(\cos \phi)^\alpha$



# Shininess



# In practice



# Phong Model

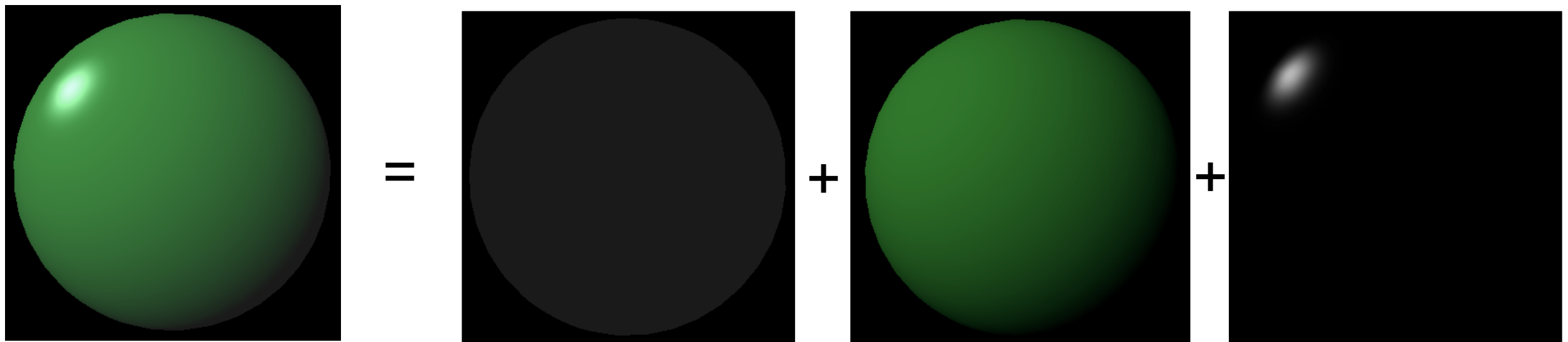
- Combine the three terms

$$I = k_a L_a + k_d L_d \max(\mathbf{l} \cdot \mathbf{n}, 0) + k_s L_s \max((\mathbf{r} \cdot \mathbf{v})^\alpha, 0)$$

Ambient

Diffuse

Specular



- More than one light? Accumulate!

$$C = \sum_i (I_{ambient} + I_{diffuse} + I_{specular})$$

# Phong Materials

- In the Phong model, a material is defined by the coefficients:  $k_a$ ,  $k_d$ ,  $k_s$  and shininess

- Examples:

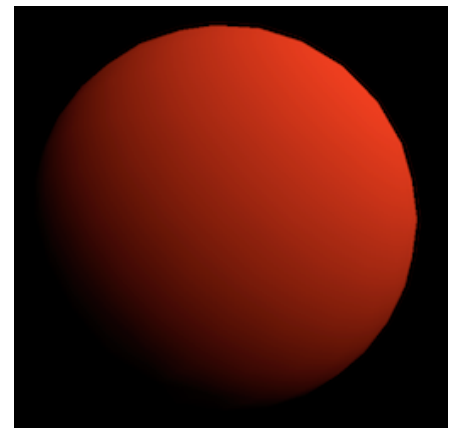
- Red diffuse material:

$$k_a = (0, 0, 0)$$

$$k_d = (1, 0, 0)$$

$$k_s = (0, 0, 0)$$

$$\text{shininess}=0$$



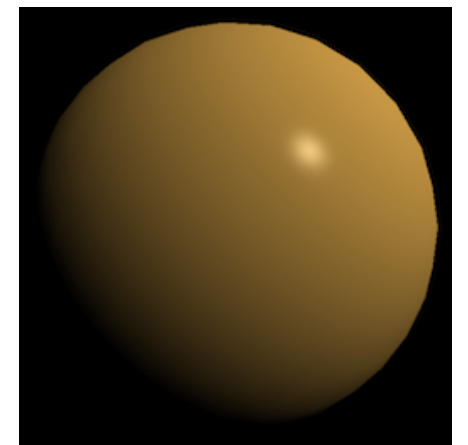
- “Gold”:

$$k_a = (0, 0, 0)$$

$$k_d = (0.8, 0.6, 0.2)$$

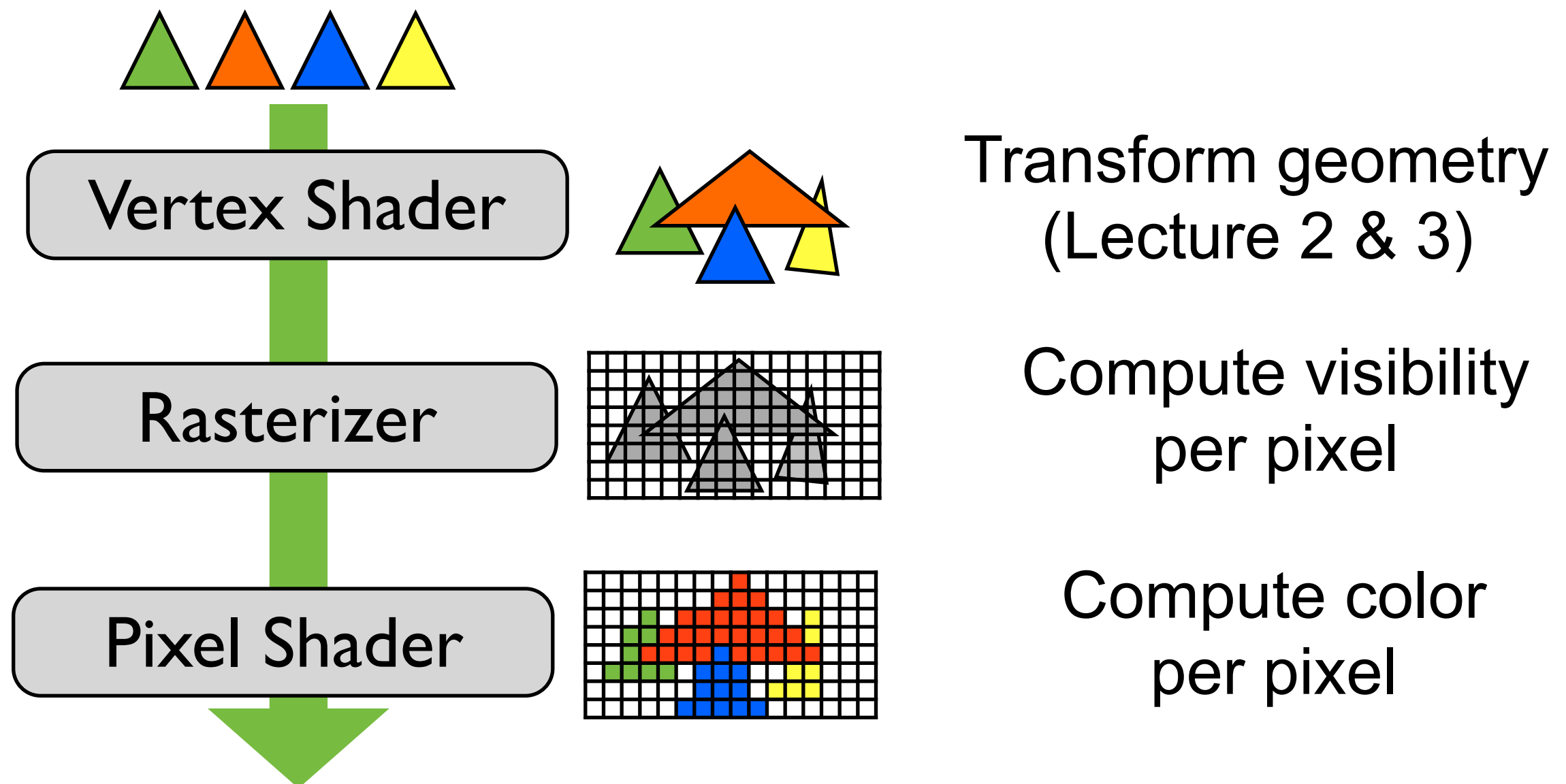
$$k_s = (0.3, 0.3, 0.3)$$

$$\text{shininess}=100$$

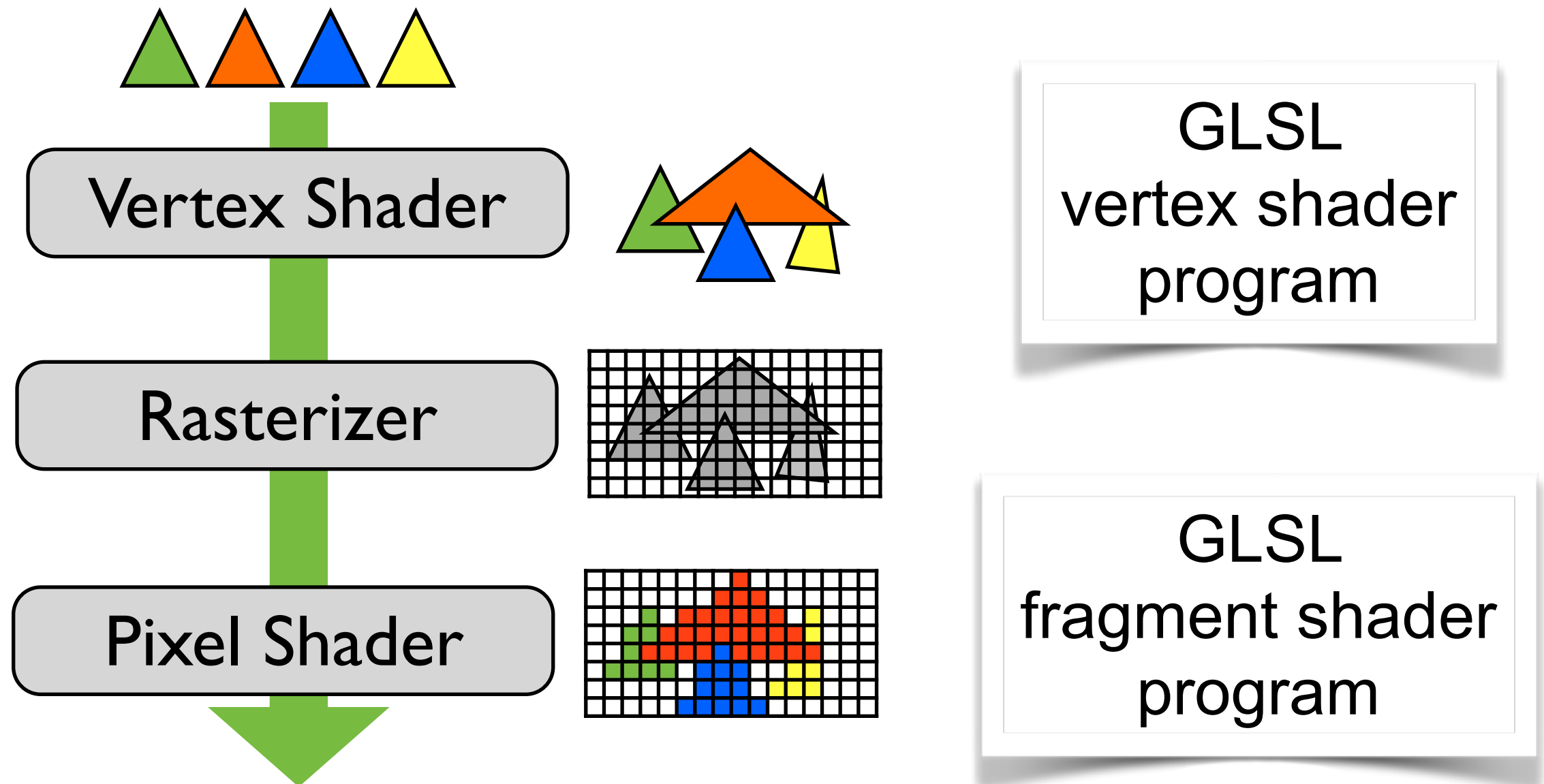


# Graphics Hardware

- Pipeline that accelerates the costly tasks of rendering



# Graphics Hardware





# Passing Geometry Info

- We need position and a set of vectors
- All vectors need to be **defined in the same coordinate space**
  - One of the most common bugs in shaders
  - Angle between vectors requires a common space
- The vertex shader transforms position and vectors
- Pixel shader evaluates Phong model



# Geometry setup for shading

- Place object
- Place light source (defines  $\mathbf{l}$ -vector)
- Place camera (defines  $\mathbf{v}$ -vector)
- Pick shading coordinate system
- Transform vectors into common system

# Geometry setup for shading

- Place object
- Place light source (defines  $\mathbf{l}$ -vector)
- Place camera (defines  $\mathbf{v}$ -vector)

Application  
code

- Pick shading coordinate system
- Transform vectors into common system

Vertex  
shader

# GLSL

- **OpenGL Shading Language**
- **Domain-specific C-like language**
  - **Scalar types:** `float`, `int`, `bool`
  - **Vectors:** `vec2`, `vec3`, `vec4`
  - **Matrices:** `mat2`, `mat3`, `mat4`
  - **Texture samplers**

# Working with vectors

```
vec4 v(1.0, 2.0, 3.0, 4.0);  
v[1] = v.y = 2.0;  
v.xyz = v.rgb = vec3(1.0, 2.0, 3.0);
```

```
//Swizzle on right side of assignment ok  
vec4 a = v.wxyz //a is now (4.0, 4.0, 1.0, 2.0)
```

```
//Swizzle on left side of assignment is writemask  
// (which component(s) to write to)  
a.xy = vec2(1.0, 2.0) //Ok, update xy  
a.xx = vec2(1.0, 2.0) //Error: Destination swizzle may  
                        // not have duplicate components
```

# GLSL Matrix & Vector ops

```
mat4 M = ...  
vec4 v = ...  
vec4 u = M*v; // Matrix mul
```

```
float f = dot(u.xyz, v.xyz);  
vec3 w = cross(u.xyz, v.xyz);
```

```
vec3 n = ...  
vec3 l = ...  
vec3 r = reflect(-l, n);  
float lambert = dot(n, l);
```

# Qualifiers

- in, out
  - Pass vertex attributes to and from shaders  
`in` vec2 tex\_coord, `out` vec4 color;
- uniform - variables from application
  - `uniform` float time
  - Same value for all vertices / pixels

# Built-in functions

- Use these!
  - Arithmetic: `sqrt`, `pow`, `abs`, `clamp`
  - Trigonometric: `sin`, `cos`, ...
  - Geometry: `length`, `reflect`, `dot`, `cross`, `smoothstep`, ...



# Output from shaders

- Vertex shader output
  - `gl_Position` Mandatory output (position) from vertex shader
  - Add additional per-vertex outputs with `out` keyword. Ex: `out vec4 color`
- Pixel shader output
  - Pixel shader must output a `vec4` color
  - Example: `out vec4 fColor`



# Simple Vertex Shader

- Transform a point
  - A vertex shader **must** output a position in clip coordinates to the rasterizer

```
in vec4 vPos;  
uniform mat4 MVP; // ModelViewProjection mtx  
  
void main()  
{  
    gl_Position = MVP*vPos;  
}
```

$$\text{Matrix}_{[4 \times 4]} * \text{Vector}_{[4 \times 1]} = \text{Vector}_{[4 \times 1]}$$

# Simple Pixel Shader

- Set pixel color to red

```
out vec4 fColor;
```

```
void main()  
{  
    fColor = vec4(1,0,0,1);  
}
```

# Simple Pixel Shader

- Interpolate vertex colors



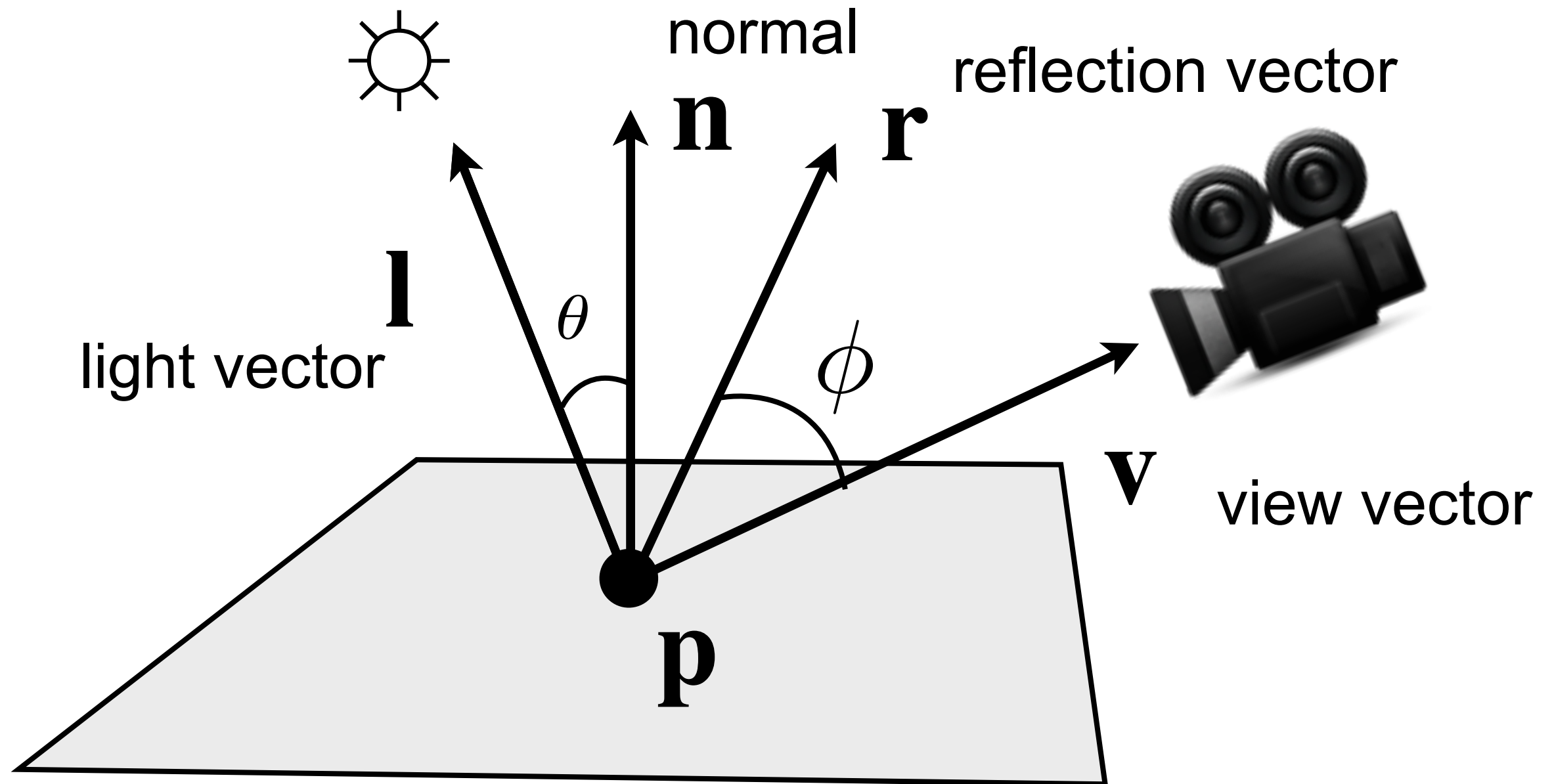
```
in vec3 vertexColor;  
out vec4 fColor;
```

```
void main()  
{  
    fColor = vec4(vertexColor, 1.0);  
}
```

# Coordinate Frames in GLSL

- Object (Model) Coordinates
- World Coordinates
- Eye (camera) coordinates
- Clip coordinates
- Normalized Device Coordinates
- Window (screen) coordinates

# Phong Shading



# Phong Shading in GLSL

## Fragment Shader

```
uniform vec3 ka;           // Material ambient
uniform vec3 kd;           // Material diffuse
uniform vec3 ks;           // Material specular
uniform float shininess;
in vec3 fN;                // Normal (from vertex shader)
in vec3 fL;                // Light vector (from VS)
in vec3 fV;                // View vector (from VS)
out vec4 fColor;

void main()
{
    vec3 N = normalize(fN);
    vec3 L = normalize(fL);
    vec3 V = normalize(fV);
    vec3 R = normalize(reflect(-L,N));
    vec3 diffuse = kd*max(dot(N,L),0.0);
    vec3 specular = ks*pow(max(dot(R,V),0.0), shininess);
    fColor.xyz = ka + diffuse + specular;
    fColor.w = 1.0;
}
```

# Phong Shading in GLSL

## Vertex Shader

```
uniform mat4 ModelViewProj; // Model -> Clip space
uniform mat4 World;         // Model -> World space
uniform mat4 WorldIT;       // Inverse transpose
uniform vec3 LightPos;      // Defined in world space
uniform vec3 CamPos;        // Defined in world space
in vec3 vPos;               // Defined in model space
in vec3 vNormal;            // Defined in model space
out vec3 fN;                // Normal vector
out vec3 fV;                // View vector
out vec3 fL;                // Light vector
```

```
void main()
{
```

## World space version

```
    vec3 worldPos = (World*vec4(vPos,1)).xyz;
    fN = (WorldIT*vec4(vNormal,0)).xyz;
    fV = CamPos - worldPos;
    fL = LightPos - worldPos;
    gl_Position = ModelViewProj*vec4(vPos,1);
}
```



# Phong Shading in GLSL

## Vertex Shader

```
uniform mat4 ModelViewProj; // Model -> Clip
uniform mat4 ModelView;     // Model -> View
uniform mat4 ModelViewIT;   // Inverse Transpose
uniform mat4 View;          // World -> View
uniform vec3 LightPos;      // Defined in world space
in vec3 vPos;               // From application
in vec3 vNormal;            // defined in model space
out vec3 fN;                // Send to Pixel shader
out vec3 fV;
out vec3 fL;
```

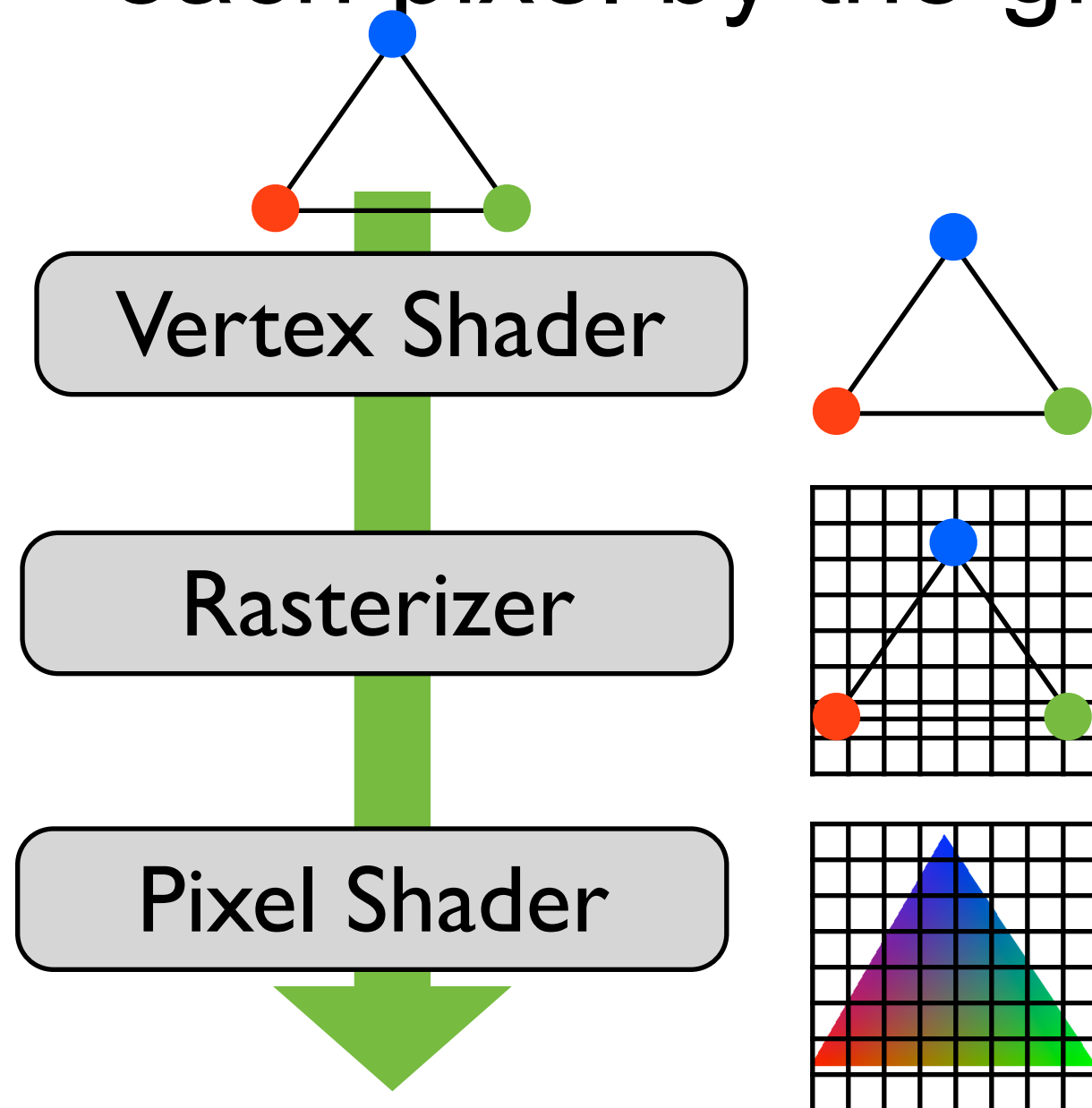
## Camera space version

```
void main()
{
    vec3 posCamSpace = (ModelView*vec4(vPos,1)).xyz;
    fN = (ModelViewIT*vec4(vNormal,0)).xyz;
    fV = -posCamSpace; // vector from point to cam
    fL = (View*vec4(LightPos,1)).xyz - posCamSpace;
    gl_Position = ModelViewProj*vec4(vPos,1);
}
```



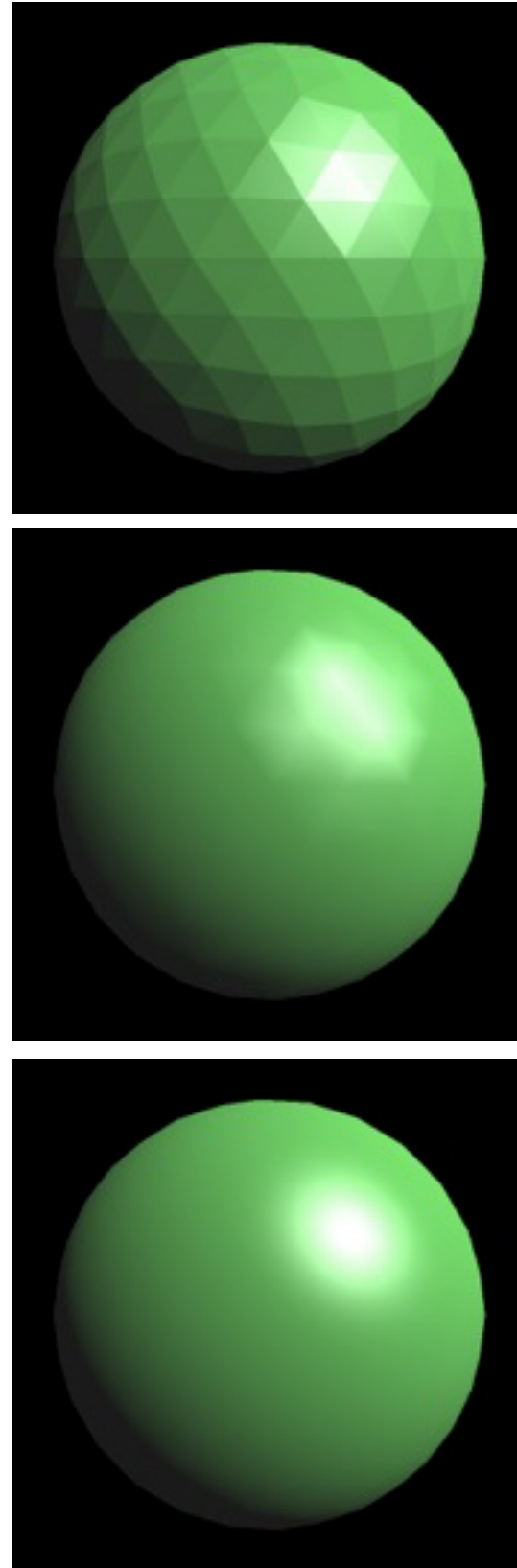
# Hardware Interpolation

- Vertex attributes are interpolated for each pixel by the graphics hardware



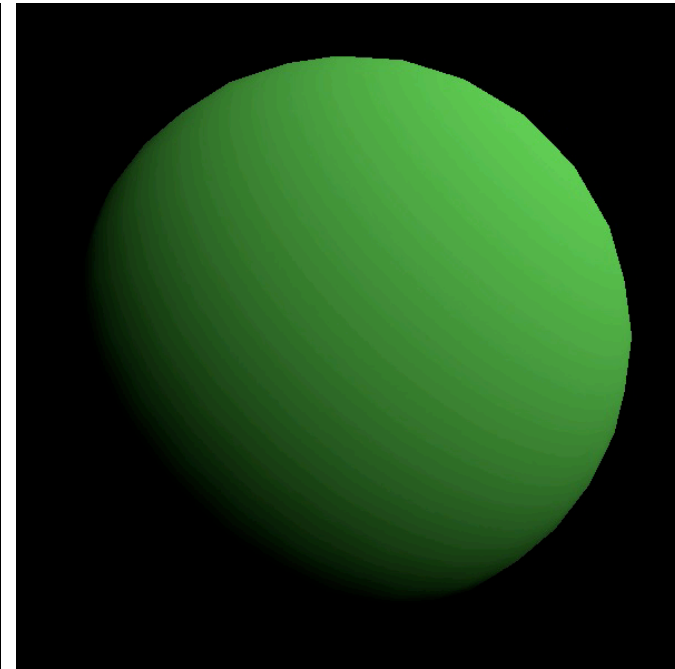
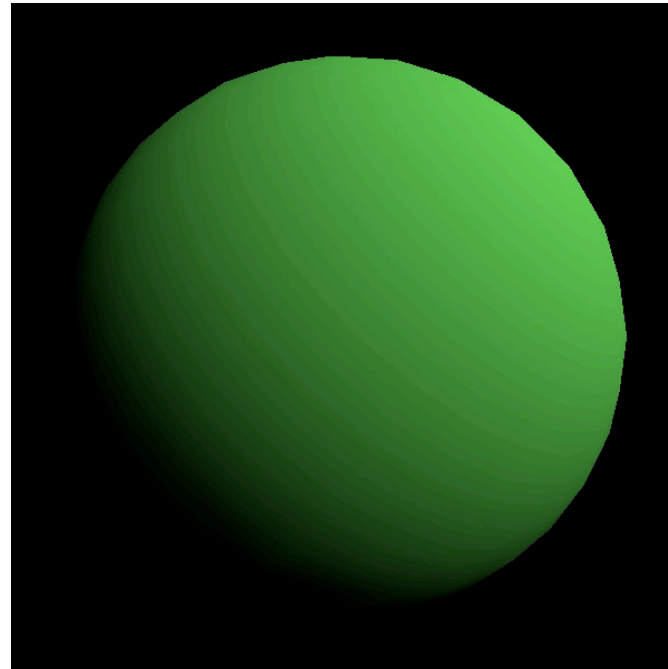
# Polygonal Shading

- Flat Shading
  - $\mathbf{l}$ ,  $\mathbf{n}$ ,  $\mathbf{v}$  constant for **each triangle**
  - Compute shading once per triangle
- Gouraud Shading
  - Shade at **each vertex**
  - Interpolate between the three vertex colors for each fragment within triangle
- Phong Shading/Per-Pixel
  - Compute shading for **each pixel**

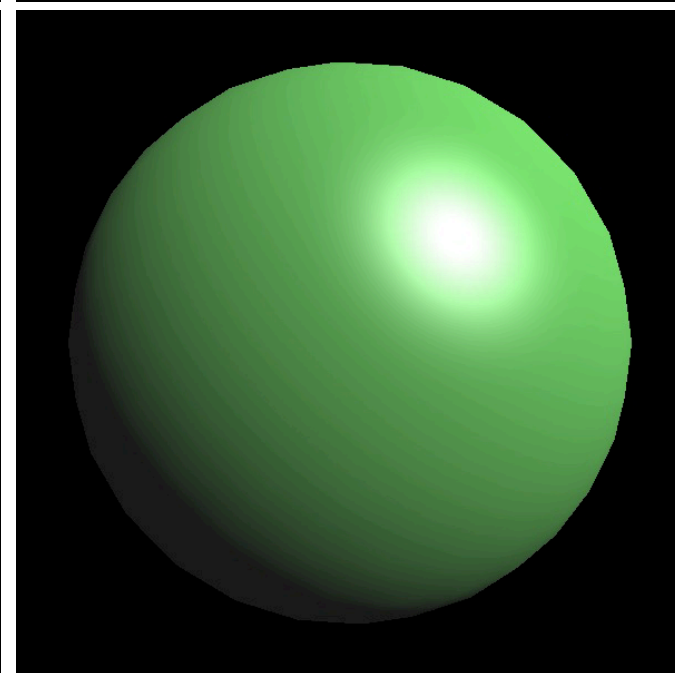
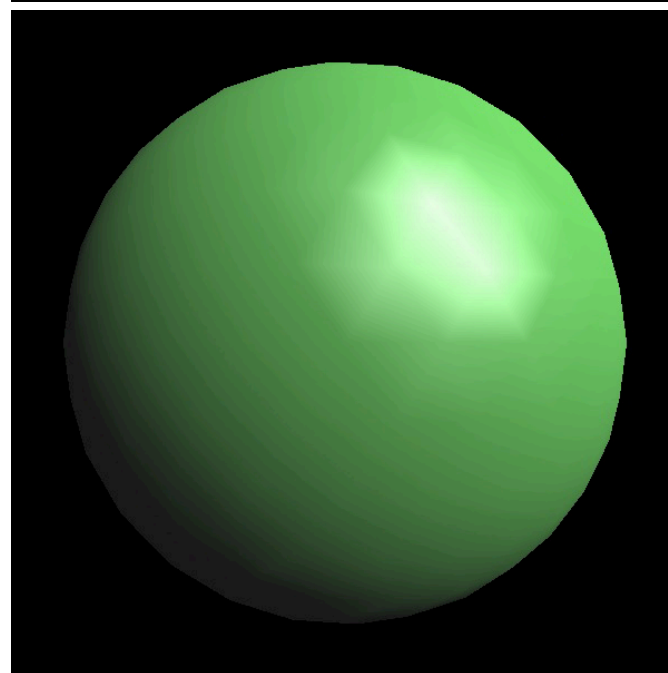


# Gouraud vs Phong

Diffuse



Ambient,  
diffuse &  
specular



Gouraud

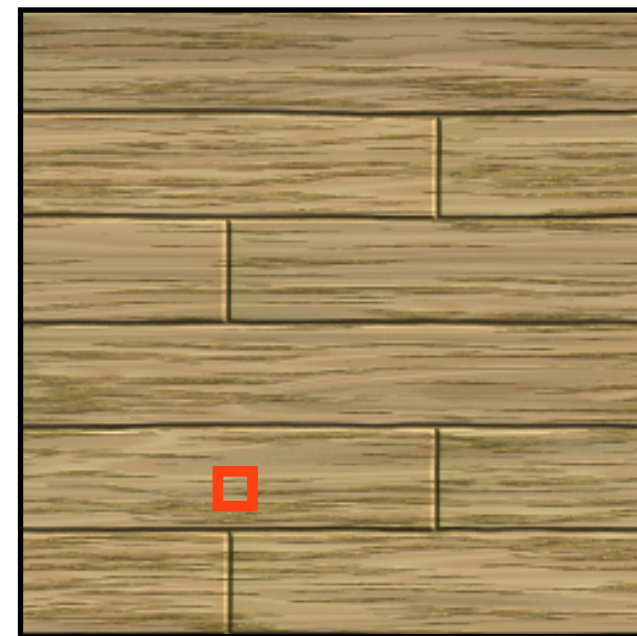
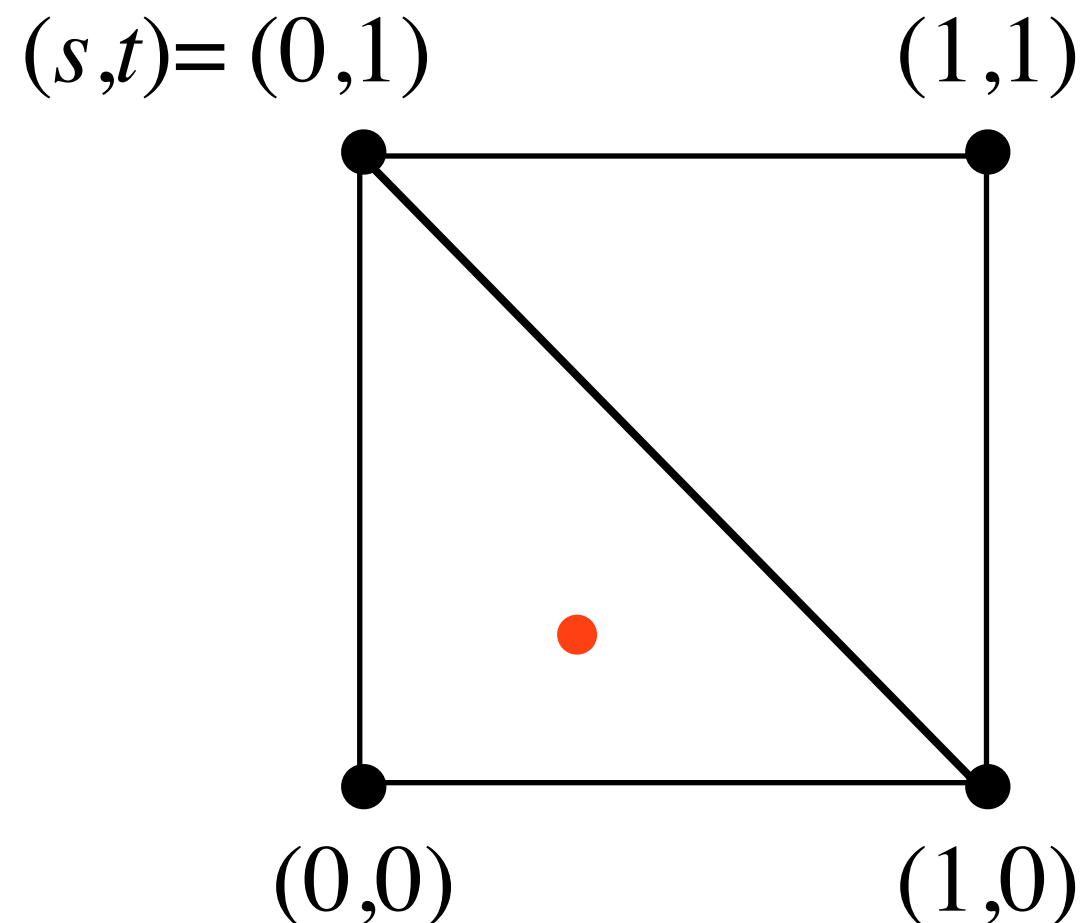
Phong

# Texture Mapping

- “Gift-wrap” geometry
- For each vertex of the mesh, store texture coordinates  $(s, t)$
- Use  $(s, t)$  to index into a texture (image)
- Adds detail to a mesh
- “Paste decals” onto geometry

# Texture Mapping

- Specify tex coordinate at each vertex
  - Hardware interpolates texCoord for each pixel
- Use coordinate to lookup in texture



# Texture Mapping in GLSL

## Vertex shader

```
in vec3 vPos;  
in vec2 vTexCoord;  
  
out vec2 texCoord;  
uniform mat4 MVP;  
  
void main()  
{  
    texCoord = vTexCoord.xy;  
    gl_Position = MVP * vPos;  
}
```

## Pixel shader

```
out vec4 fColor;  
  
uniform sampler2D Sampler;  
in vec2 texCoord;  
  
void main()  
{  
    fColor =  
        texture(Sampler, texCoord);  
}
```

# Textured Phong Model

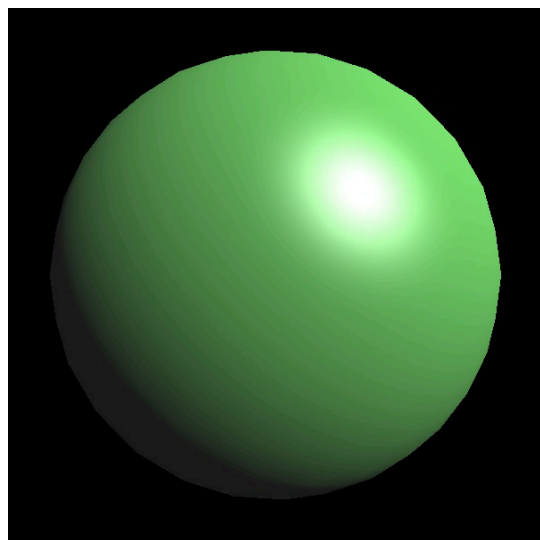
- Change the diffuse factor ( $k_d$ ) per pixel

$$I = k_a L_a + \boxed{k_d} L_d \max(\mathbf{l} \cdot \mathbf{n}, 0) + k_s L_s \max((\mathbf{r} \cdot \mathbf{v})^\alpha, 0)$$

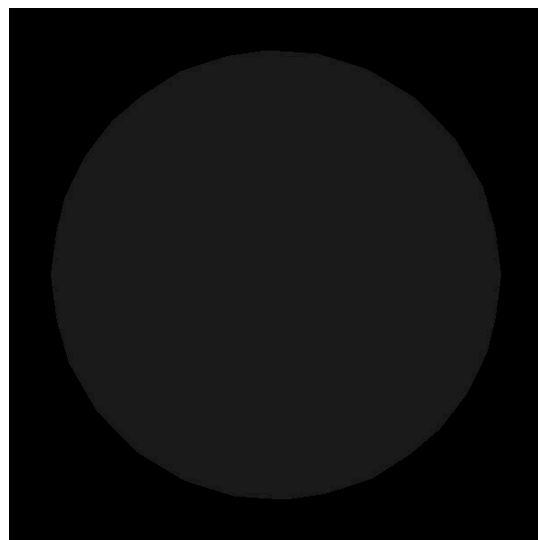
Ambient

Diffuse

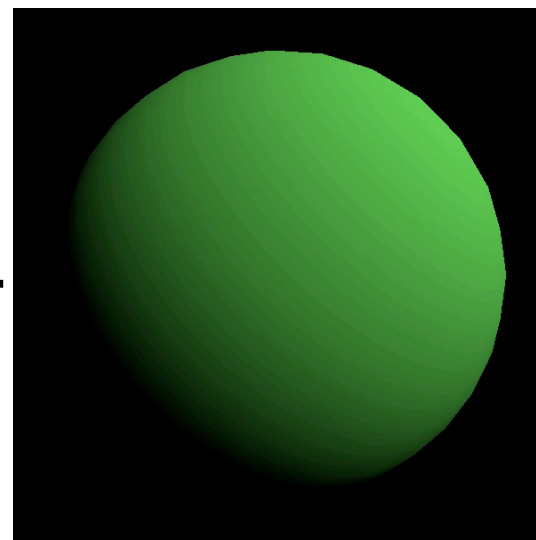
Specular



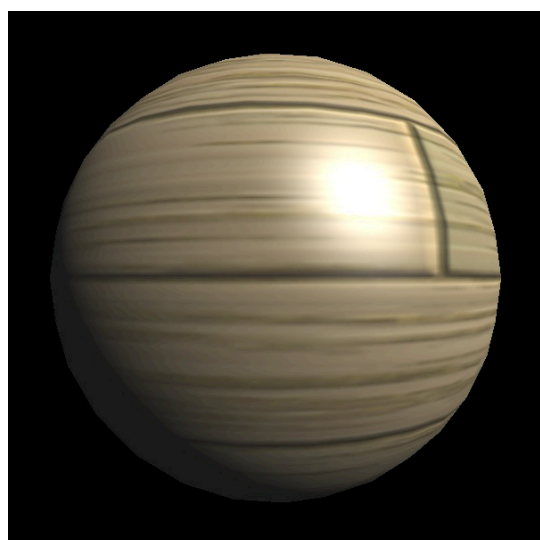
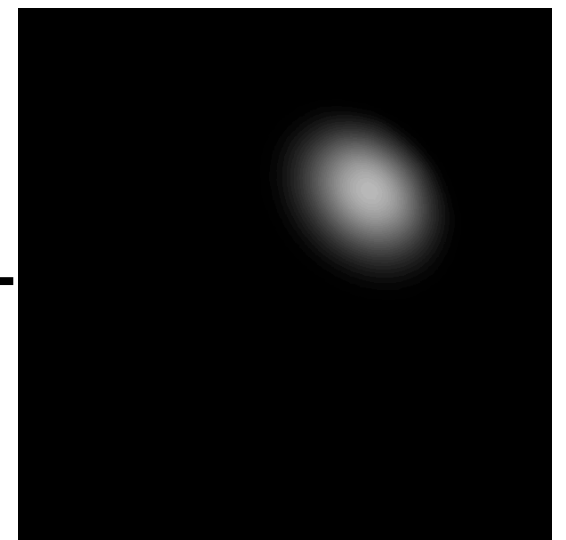
=



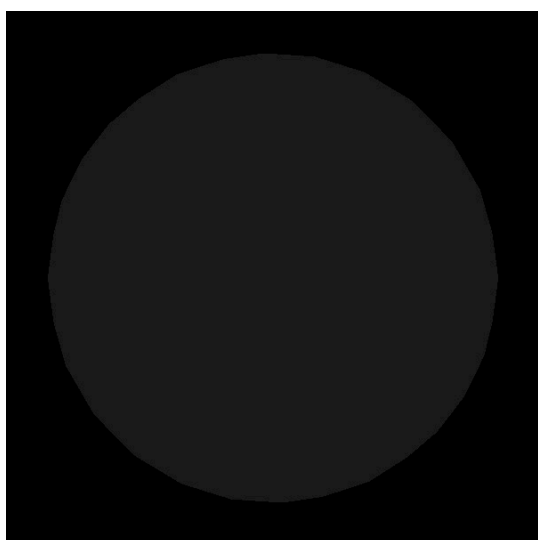
+



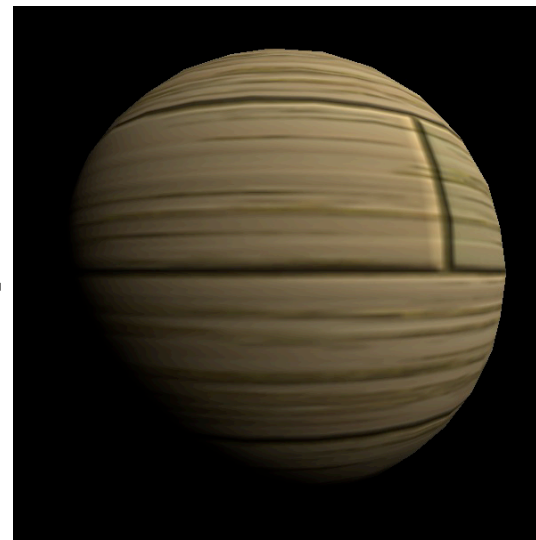
+



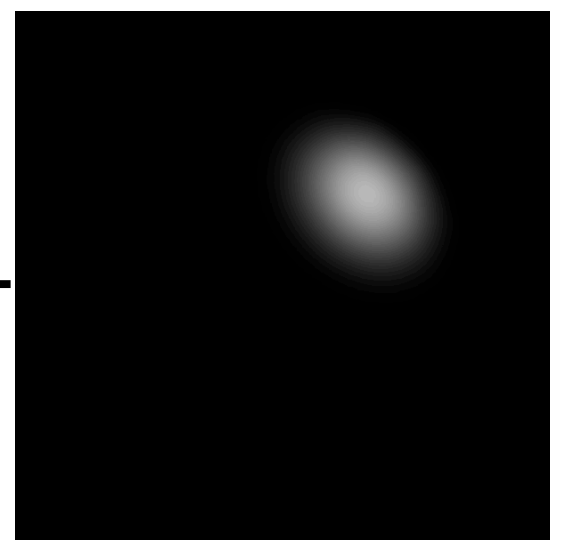
=



+



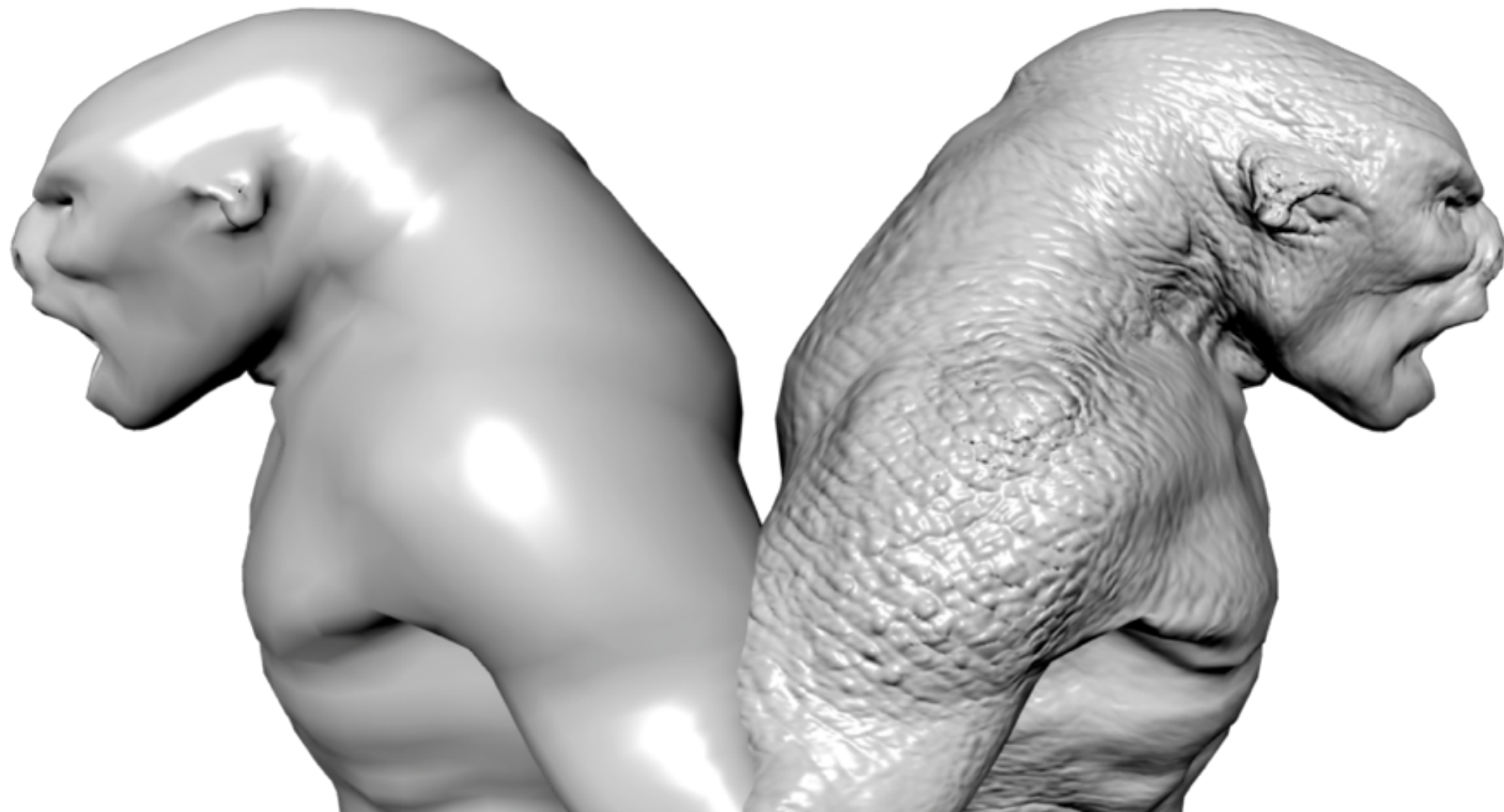
+





# Texturing in Graphics

- Map detail from an image onto a 3D object





# Bump vs Normal Mapping

- Bump mapping

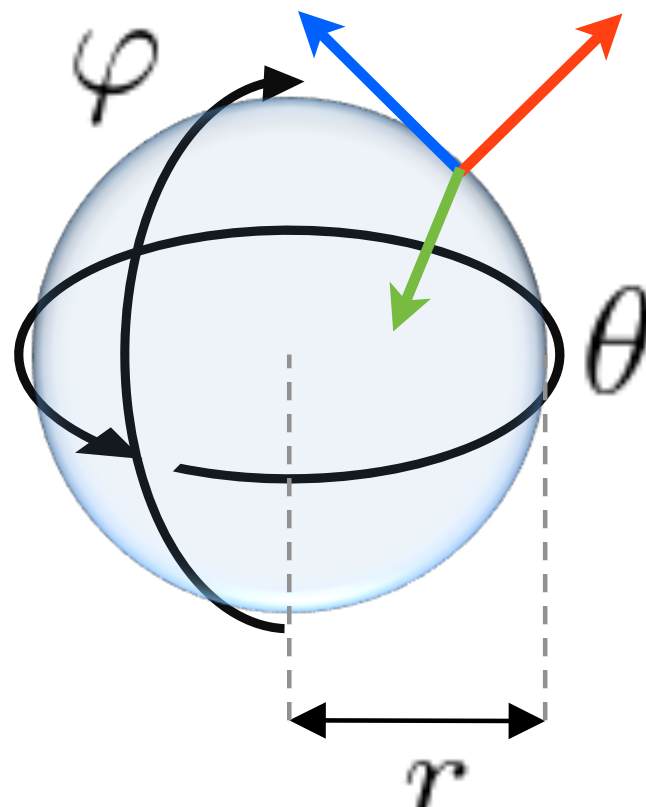
- A Bump Map is a grayscale image used to change a surface
- Can be used to compute a normal (finite differences) to perturb the current normal with
  - Blinn 78

- Tangent space Normal Mapping

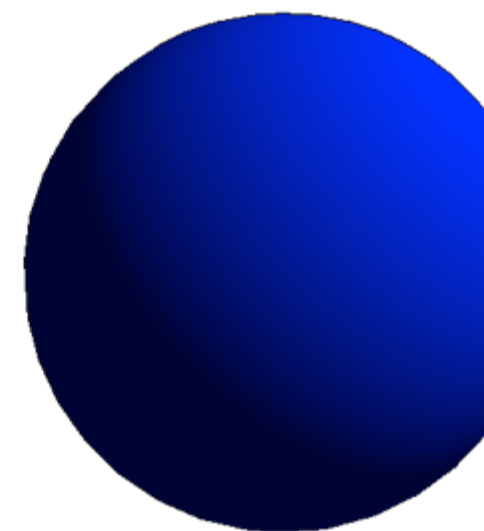
- Reads a normal from a file and replaces the existing normal by transforming it into tangent space
- Most common form of bump mapping

# Tangent $\epsilon^{\Phi(\theta, \varphi)} = \begin{cases} r \sin(\theta) \sin(\varphi) \\ -r \cos(\varphi) \\ r \cos(\theta) \sin(\varphi) \end{cases}$

- Local coordinate system made up of the tangent, binormal
- Unique for each point  $\frac{\partial \Phi}{\partial \theta} = \begin{cases} r \cos(\theta) \sin(\varphi) \\ 0 \\ -r \sin(\theta) \sin(\varphi) \end{cases}$
- Remember the sphere (lecture 3)



$$\frac{\partial \Phi}{\partial \varphi}, \frac{\partial \Phi}{\partial \theta}$$



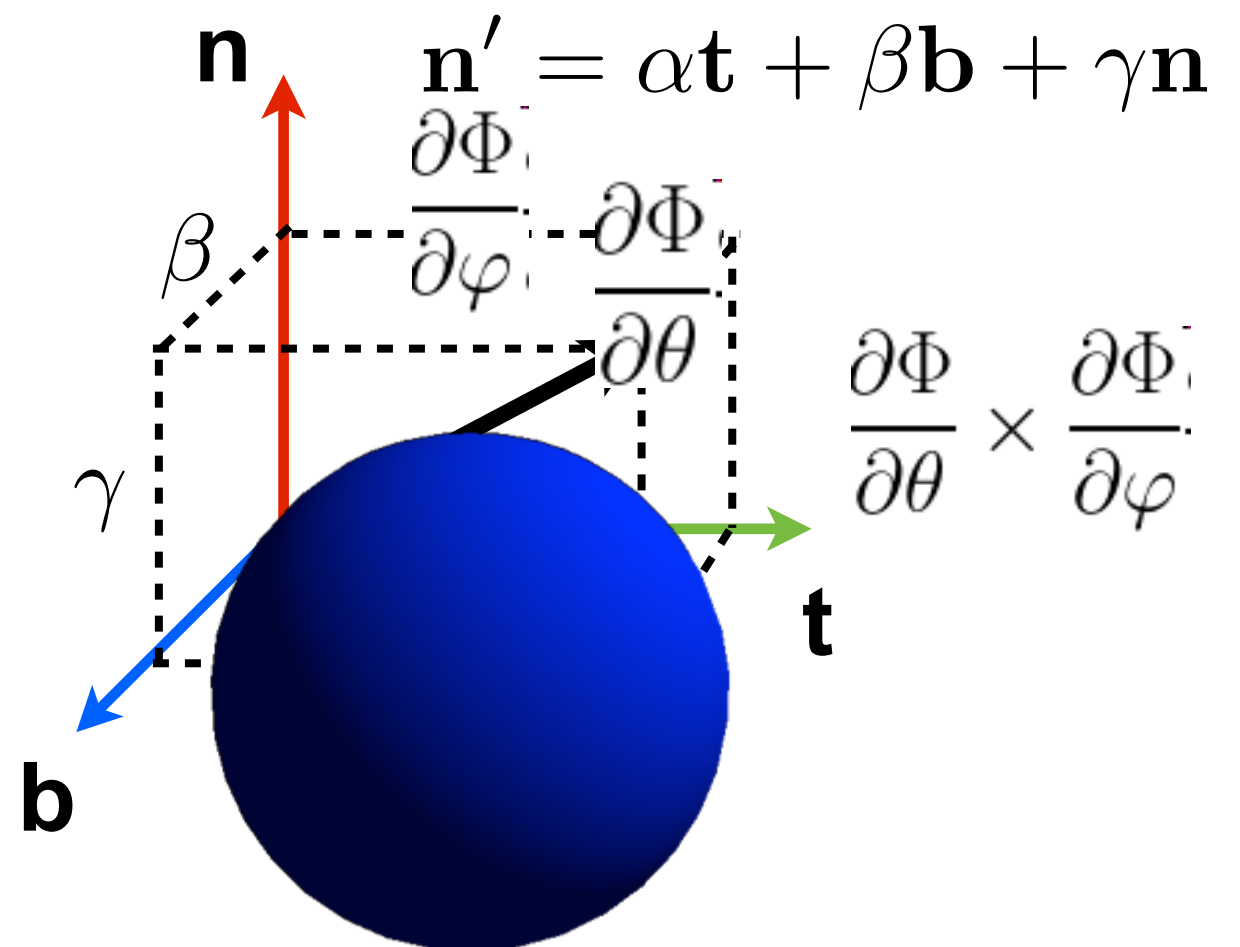
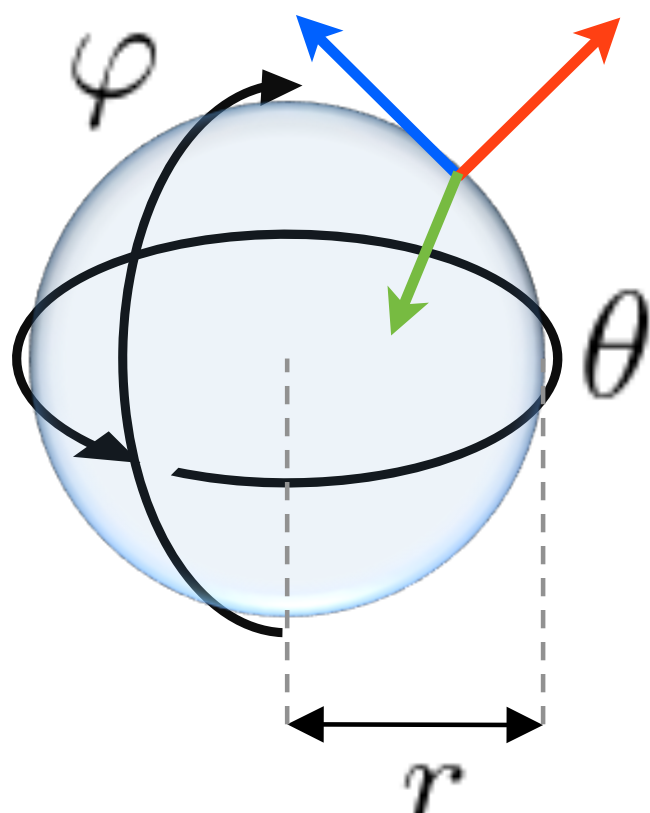
# Nc

$$\Phi(\theta, \varphi) = \begin{Bmatrix} -r \cos(\varphi) \\ r \cos(\theta) \sin(\varphi) \end{Bmatrix}, \quad 0 < \varphi < 2\pi$$

- Modify surface

- Store tangent texture

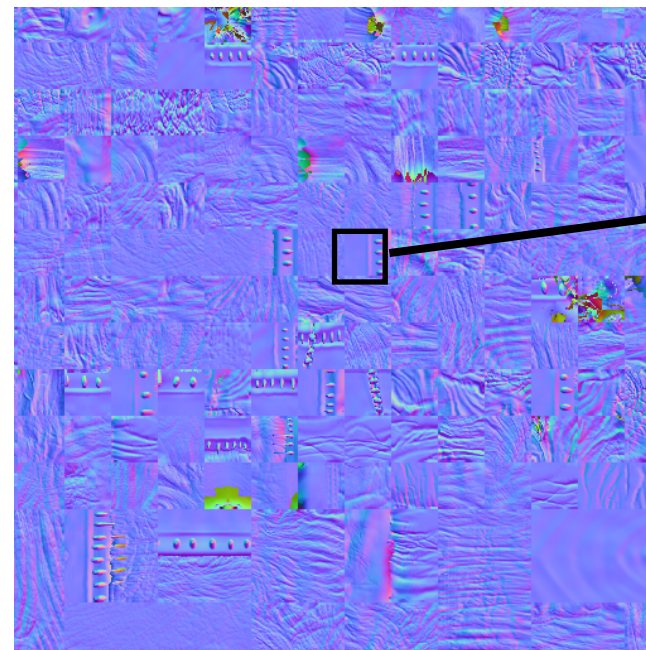
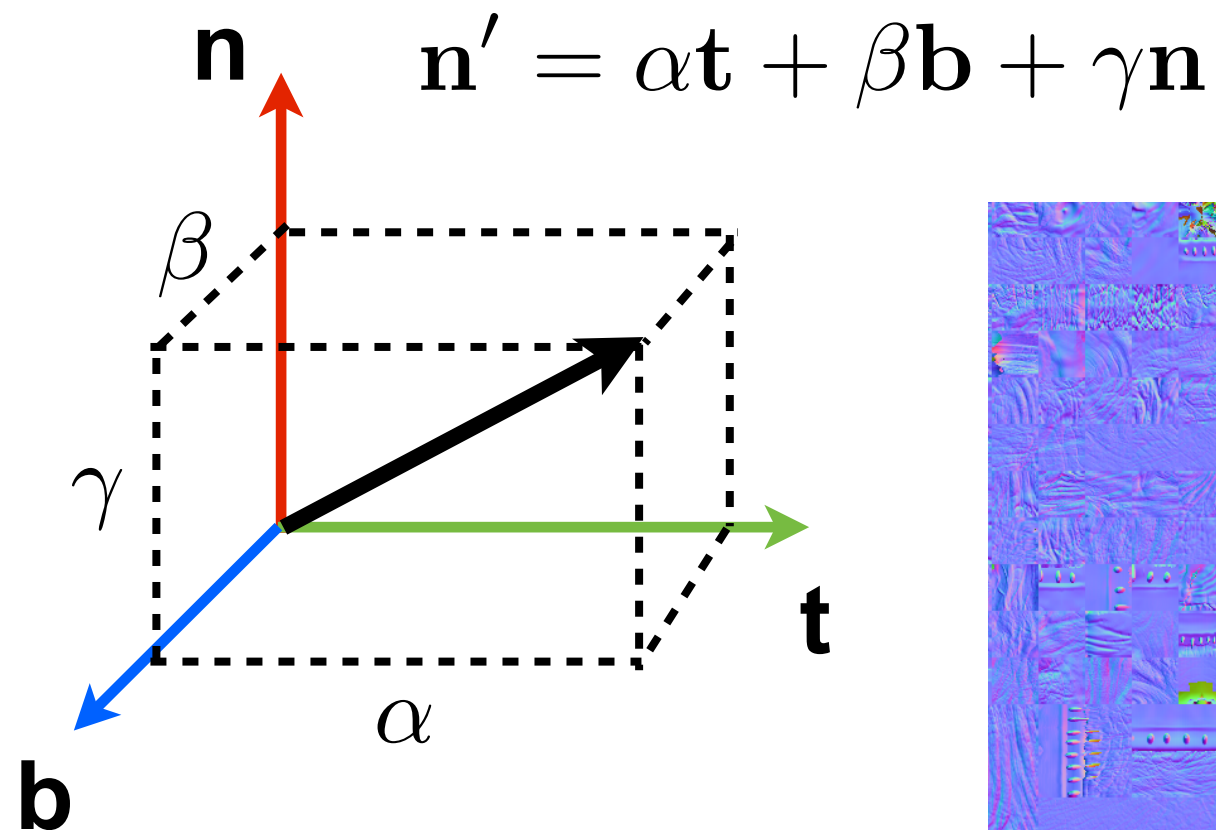
$$\frac{\partial \Phi}{\partial \theta} = \begin{Bmatrix} r \cos(\theta) \sin(\varphi) \\ 0 \\ -r \sin(\theta) \sin(\varphi) \end{Bmatrix}, \quad \frac{\partial \Phi}{\partial \varphi} = \begin{Bmatrix} r \sin(\varphi) \\ r \cos(\theta) \cos(\varphi) \\ r \sin(\theta) \cos(\varphi) \end{Bmatrix}$$



# Normal Mapping

- Modify surface normal for each pixel
- Store **tangent space** normal in a texture

$$(R, G, B) \rightarrow (\alpha, \beta, \gamma)$$
$$[0, 255] \rightarrow [-1, 1]$$



$(127, 127, 255) \rightarrow (0, 0, 1)$

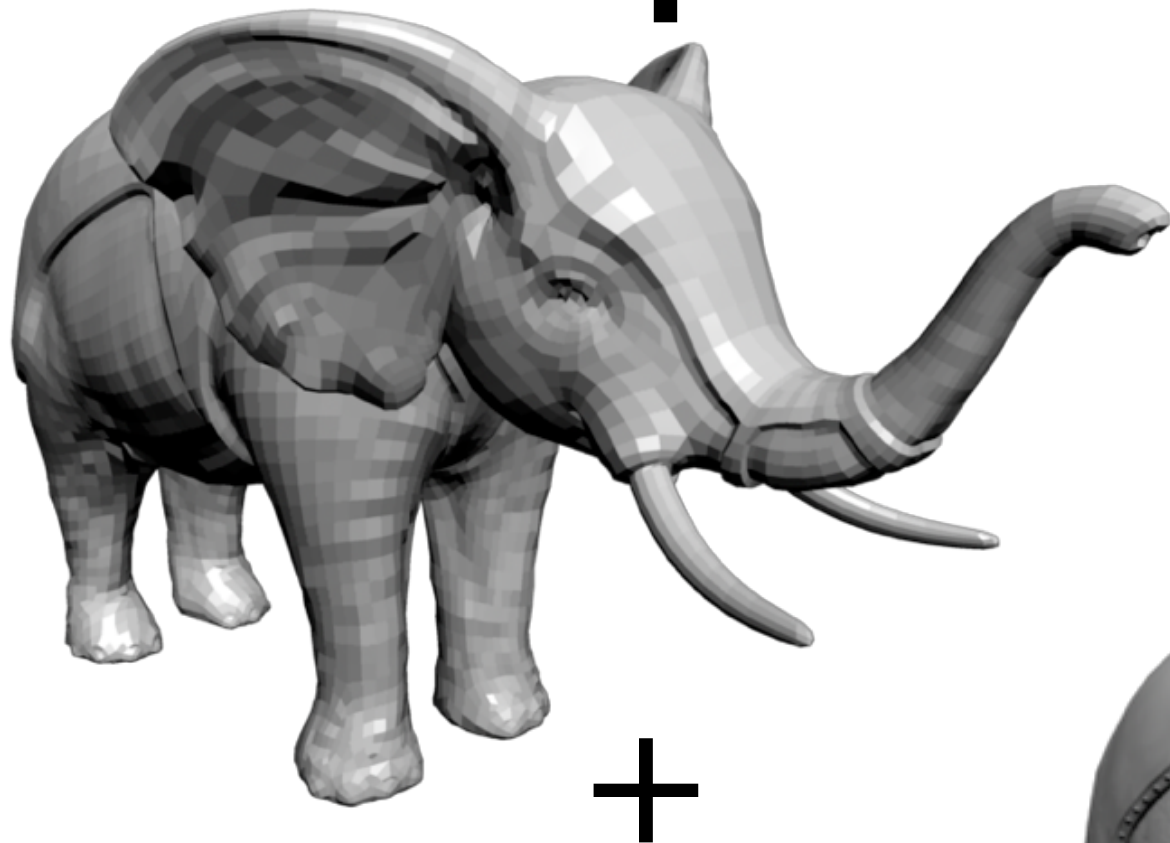
# Coordinate Transform

- From tangent space to object space
  - Normal in tangent space:  $(\alpha, \beta, \gamma)$
  - Basis vectors (defined in object space):  $\mathbf{t}$ ,  $\mathbf{b}$ ,  $\mathbf{n}$
  - Normal in object space:

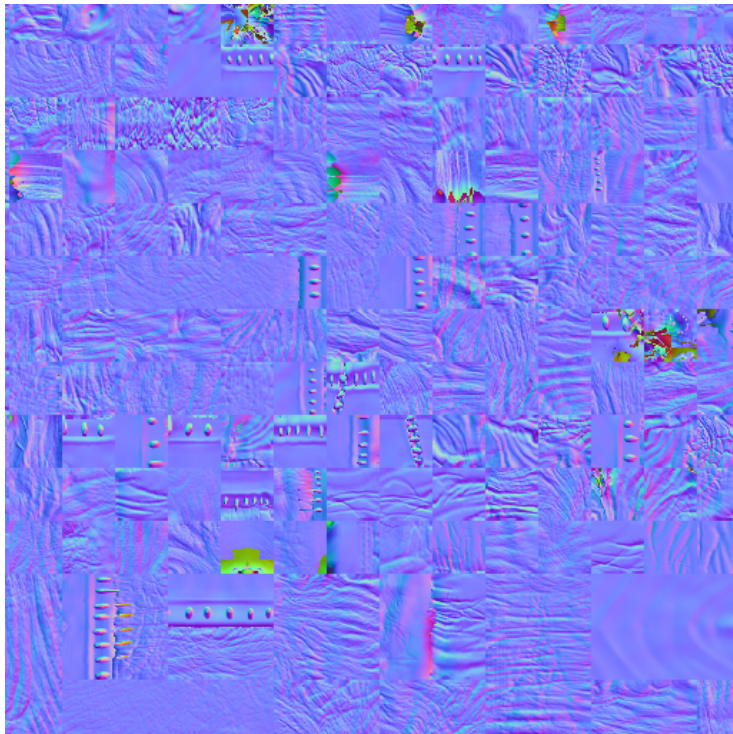
$$\alpha\mathbf{t} + \beta\mathbf{b} + \gamma\mathbf{n} = \begin{bmatrix} | & | & | \\ \mathbf{t} & \mathbf{b} & \mathbf{n} \\ | & | & | \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}$$



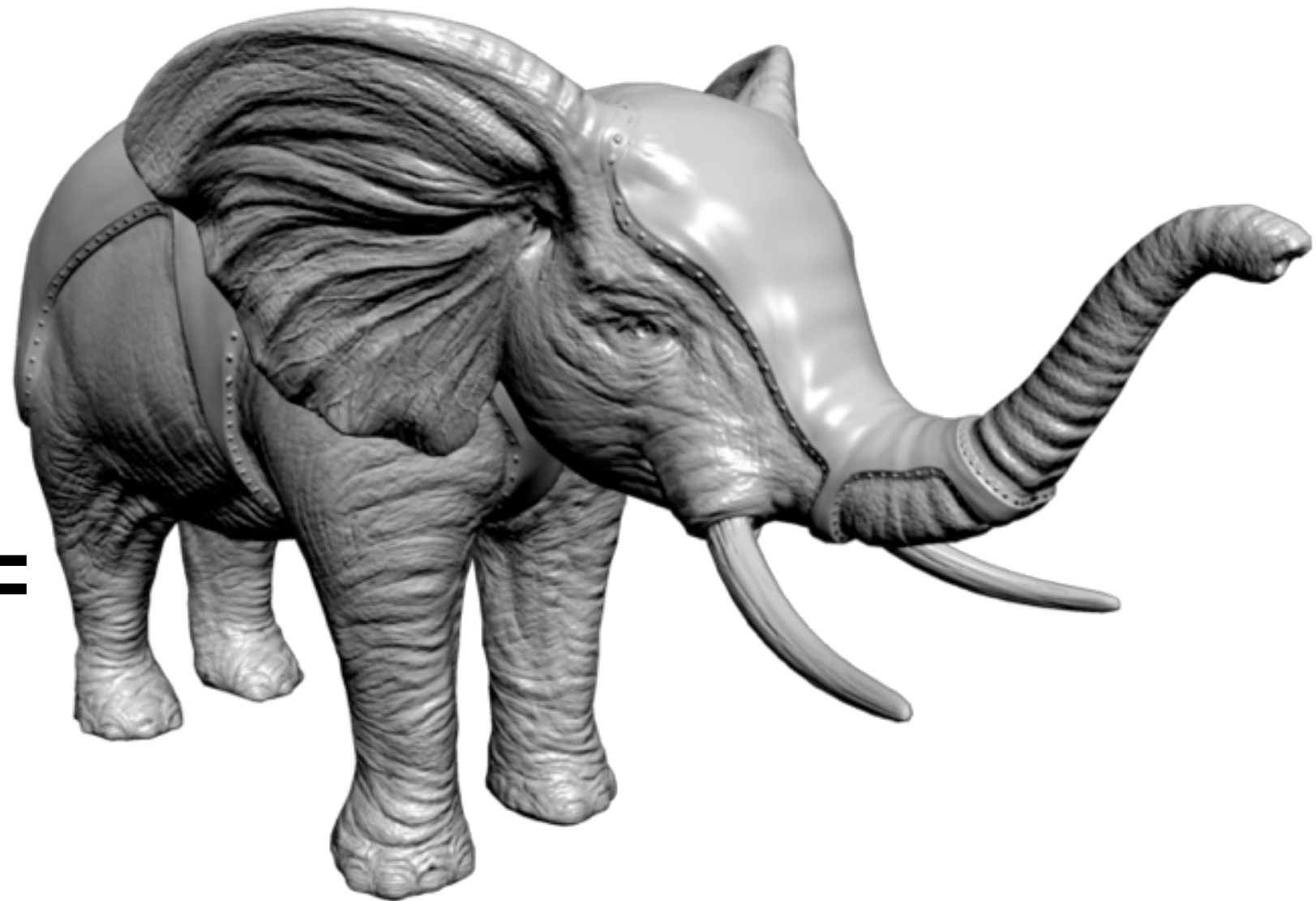
# Example : Normal Mapping



+



=

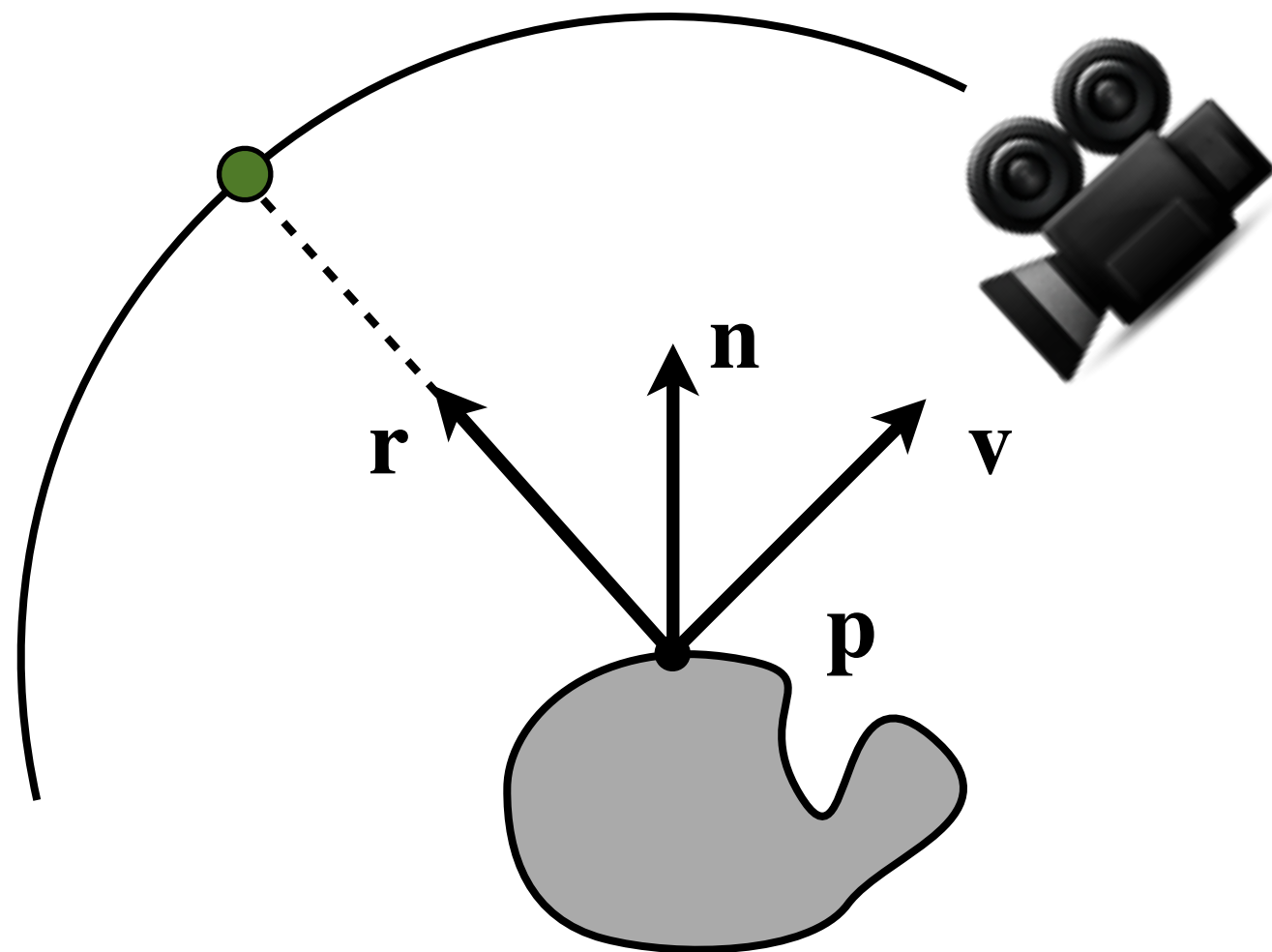


# Normal Mapping Summary

- Derive tangent space
  - Compute tangent, binormal and normal in vertex shader and pass to pixel shader
- Find tangent space normal
  - Lookup  $(\alpha, \beta, \gamma)$  from texture, and remap from  $[0, 1]$  to  $[-1, 1]$  (Colors in GLSL:  $[0, 1]$  instead of  $[0, 255]$ )
  - Express as  $\mathbf{n}' = \alpha \mathbf{t} + \beta \mathbf{b} + \gamma \mathbf{n}$  (object space)
- Transform normal from object space to world space and shade

# Reflection Mapping

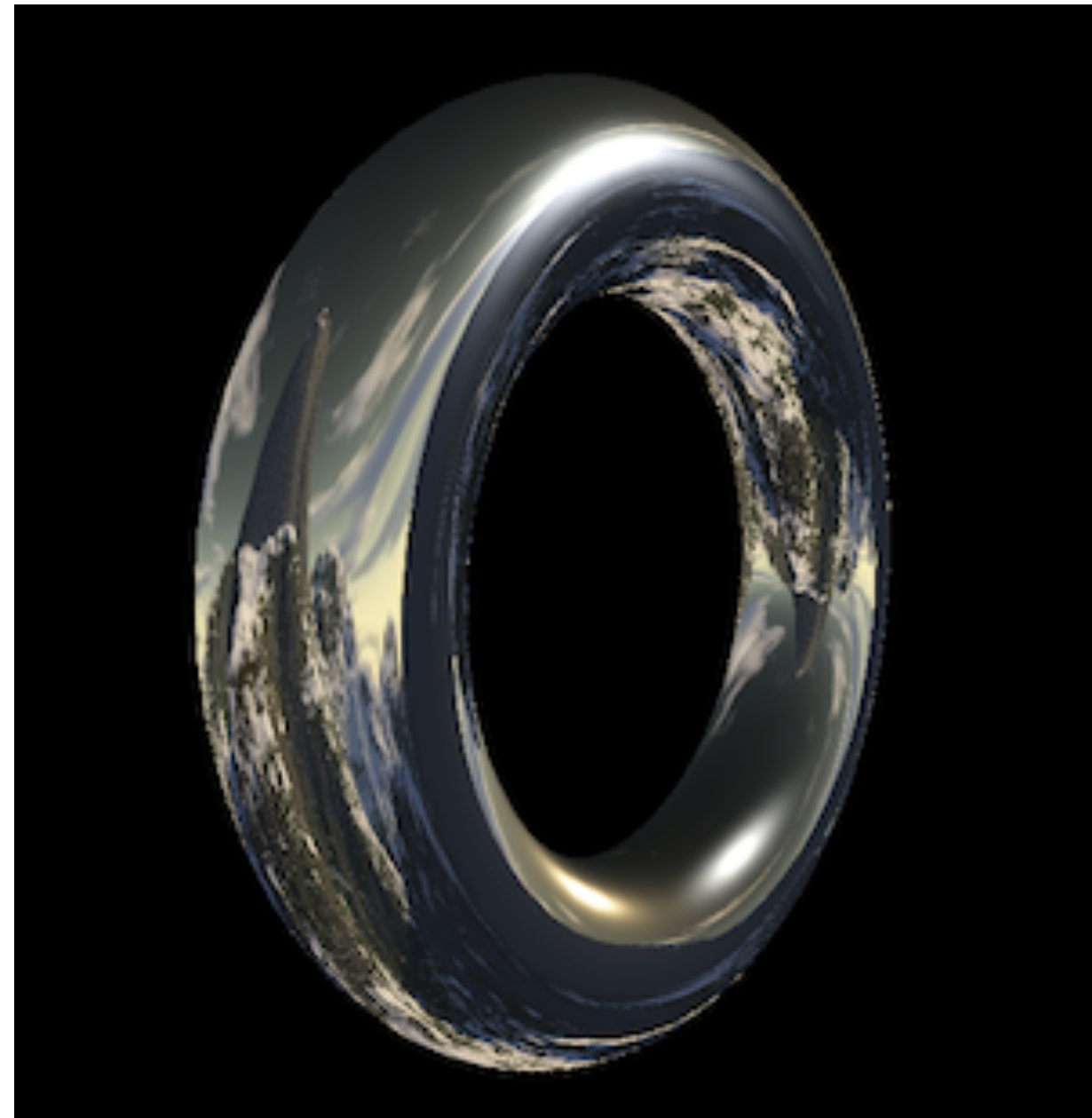
- Fast (but incorrect) way to simulate reflection
- Use reflection vector,  $\mathbf{r}$ , to lookup in a cube map texture
- Instead of  $(s, t)$ , use a 3D direction
  - Direction gives a point on unit sphere





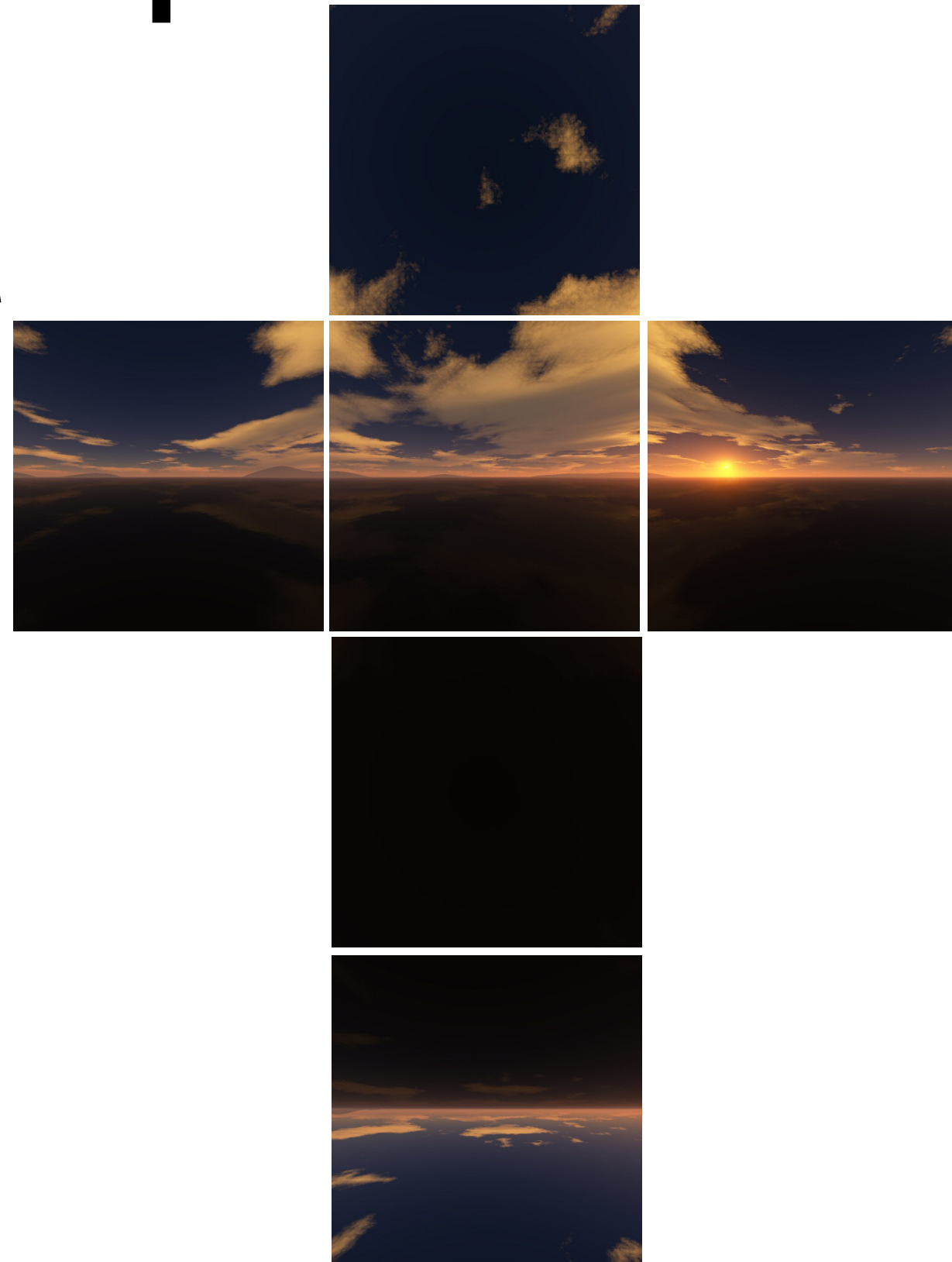
# Reflection Mapping

- Simulate highly reflective surfaces, such as chrome, mirrors and metals
- How do we express environment?



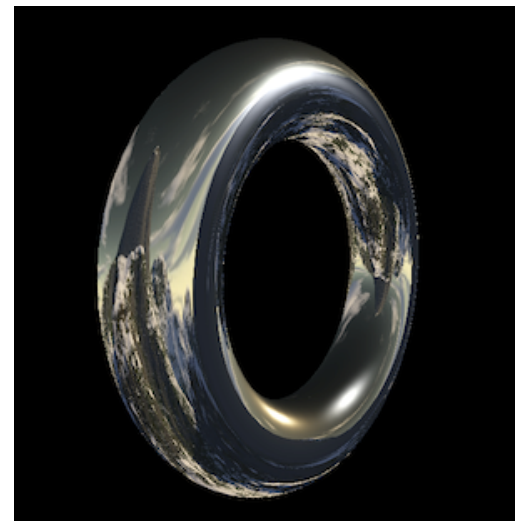
# Cube Map

- Take six photos from the same point
  - With camera looking: Front, back, left, right, up, down
- The cube approximates the spherical view

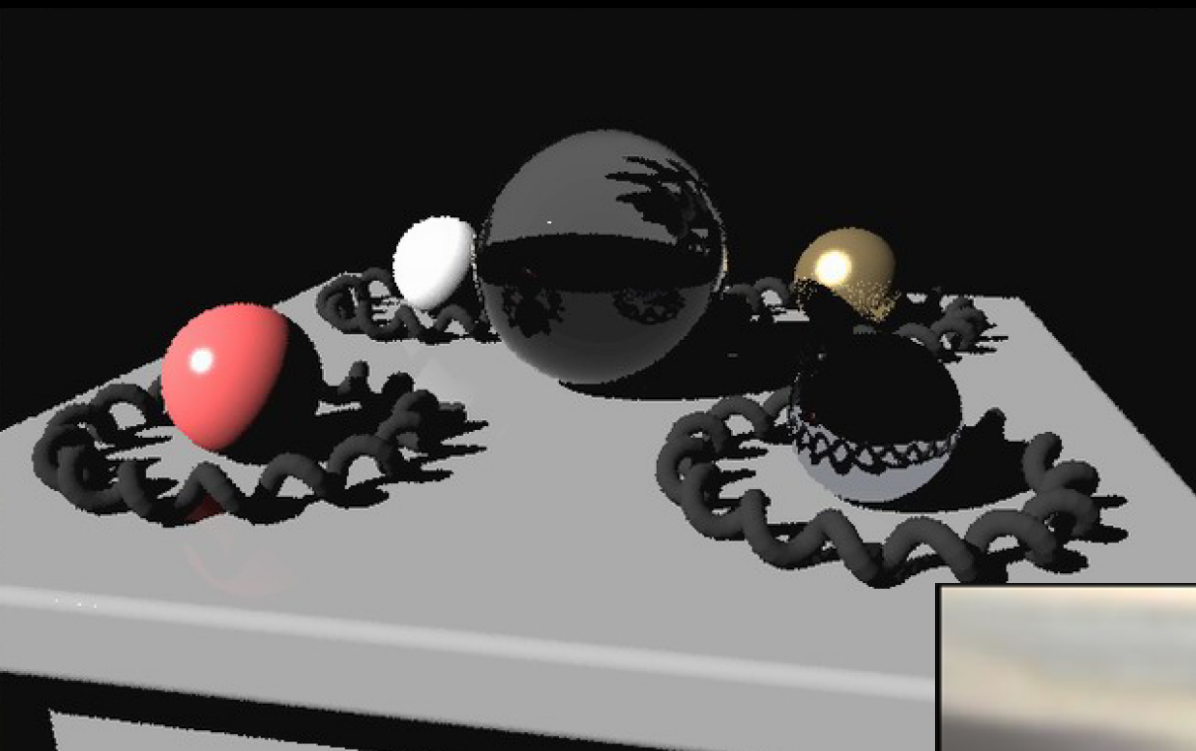


# Reflection Mapping in GLSL

```
out vec4 fColor;  
uniform samplerCube SkyboxTexture;  
  
void main()  
{  
    vec3 V = normalize(...);  
    vec3 N = normalize(...);  
    vec3 R = reflect(-V, N);  
    vec4 reflection =  
        texture(SkyboxTexture, R);  
    fColor = reflection;  
}
```







Funston  
Beach



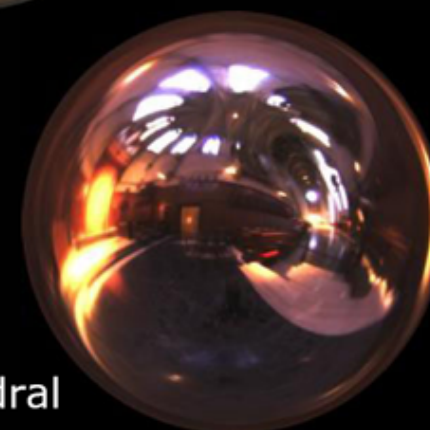
Eucalyptus  
Grove



Uffizi  
Gallery

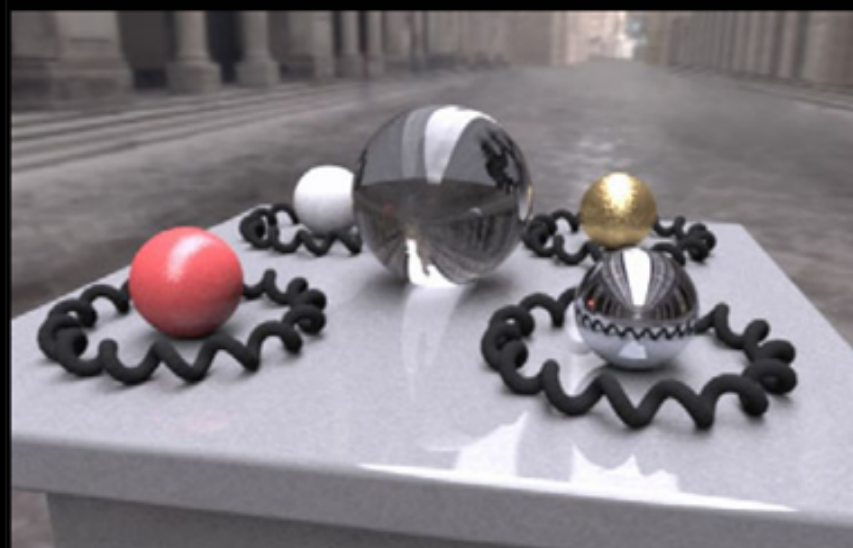
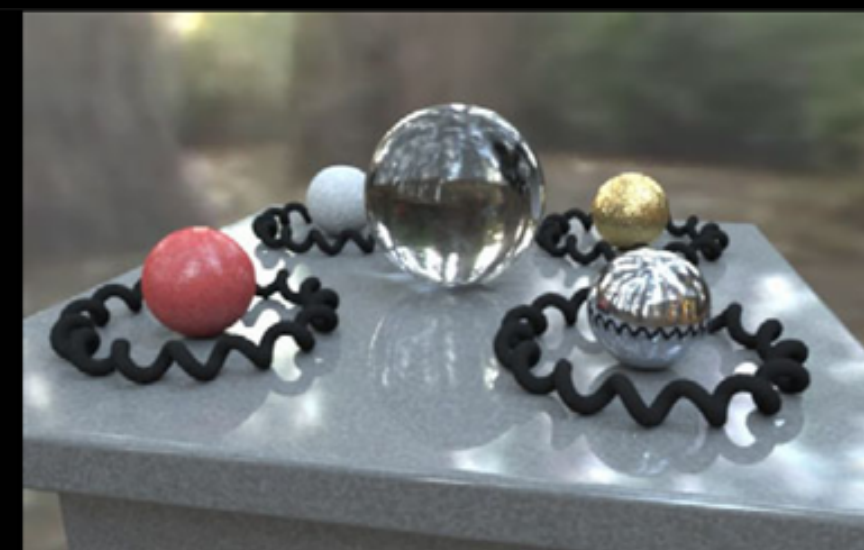
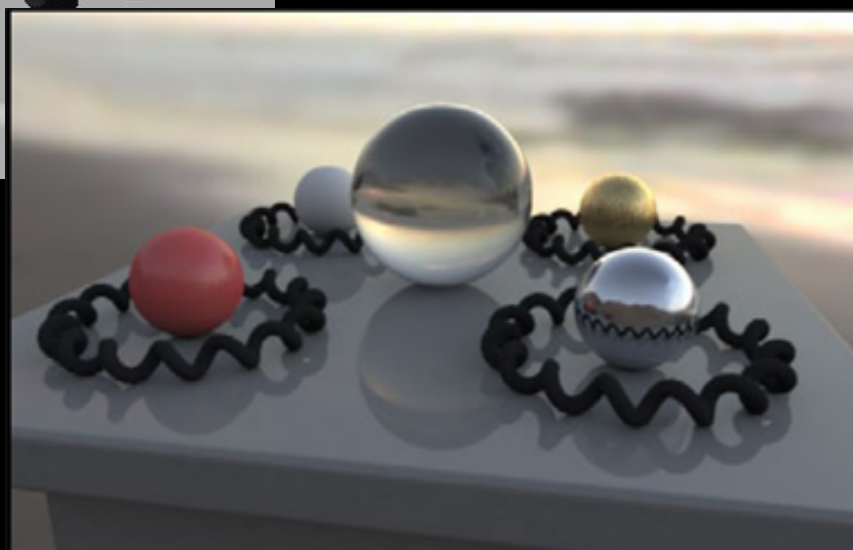


Grace  
Cathedral



Images courtesy  
of Paul Debevec

<http://pauldebevec.com/Probes/>



# Next

- Wednesday Lab3 seminar