CS100433
Illumination Model and Surface Shading

Junqiao Zhao 赵君峤

Department of Computer Science and Technology

College of Electronics and Information Engineering

Tongji University

Visual cues to 3D geometry

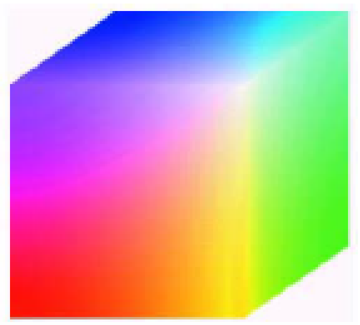
- size (perspective)
- occlusion (depth)
- shading

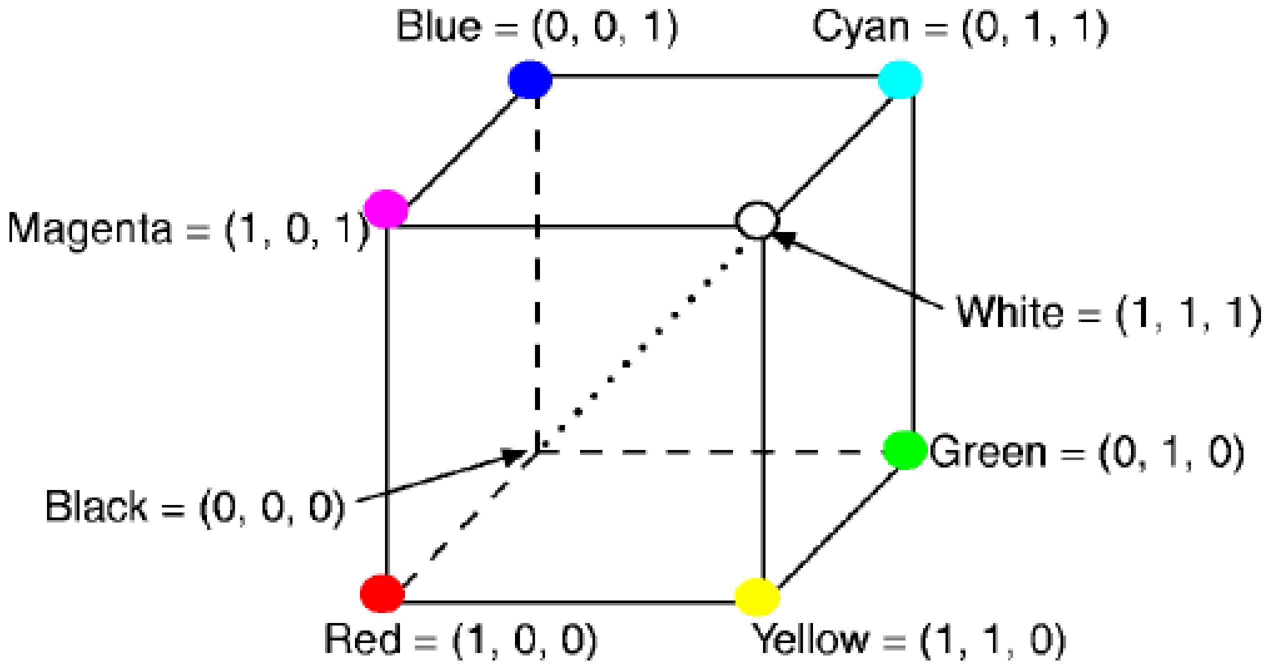
Color

- Color is a property of objects that our minds create –an interpretation of the world around us
- Color of object depends not only on object itself but also on light source illuminating it, on color of surrounding area, and on human visual system (HVS, the eye/brain mechanism)

Color model

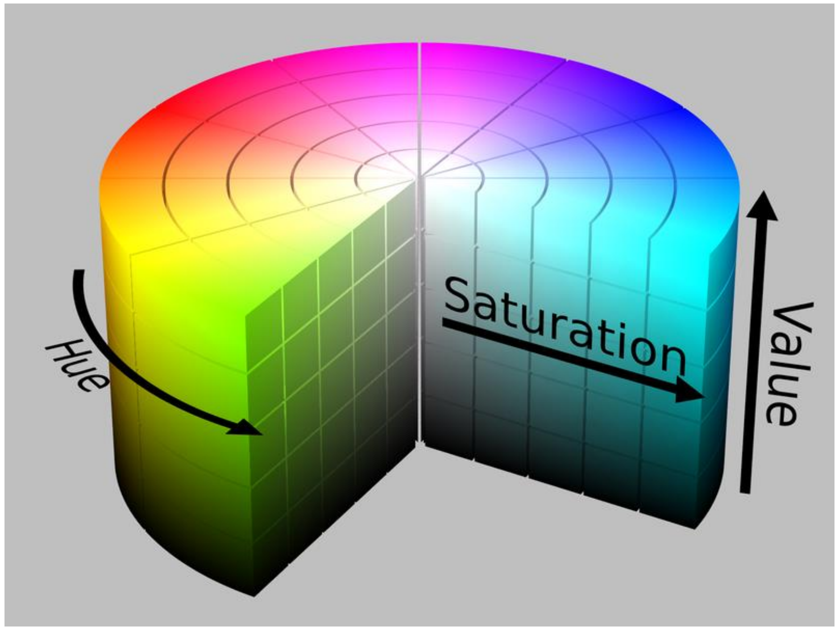
- RGB
- CMY(K)

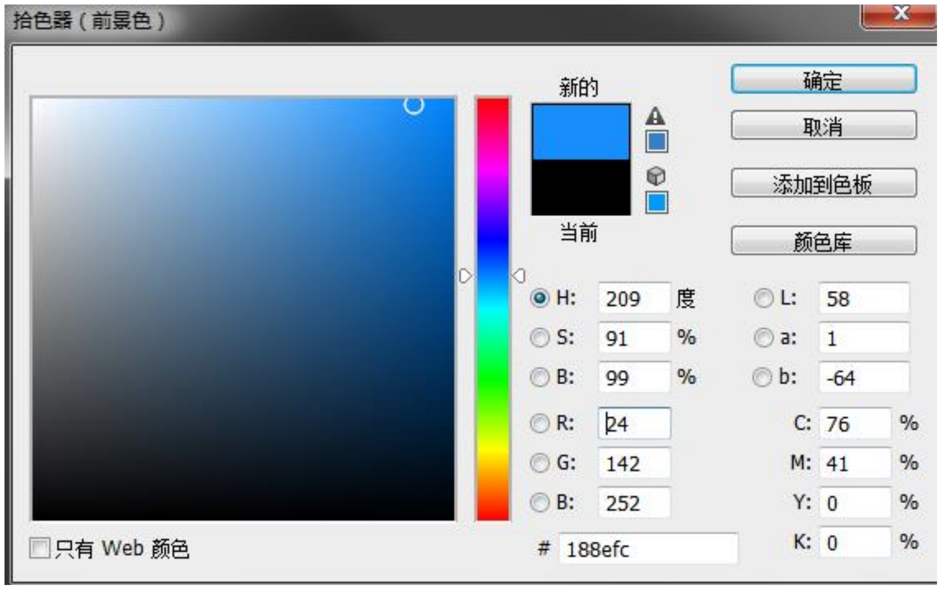




HSV color model

- **H**ue, **S**aturation, and **V**alue (Brightness)
- Rearrange the geometry of RGB in an attempt to be more intuitive and perceptually relevant





H:	209	度	L:	58	
S:	91	%	a:	1	
B:	99	%	b:	-64	
R:	24		C:	76	%
G:	142		M:	41	%
B:	252		Y:	0	%
			K:	0	%

☐ 只有 Web 颜色

# 188efc

Light color vs Reflected color

- The colors we see in real life are not the colors the objects actually have, but are the colors reflected from the object (rejected by the object).
  - If the light color is white: L = (1.0, 1.0, 1.0)
  - An object reflects color: R = (1.0, 0.0, 0.0)
  - The resulting color of the object is L*R = (1.0, 0.0, 0.0) which is Red

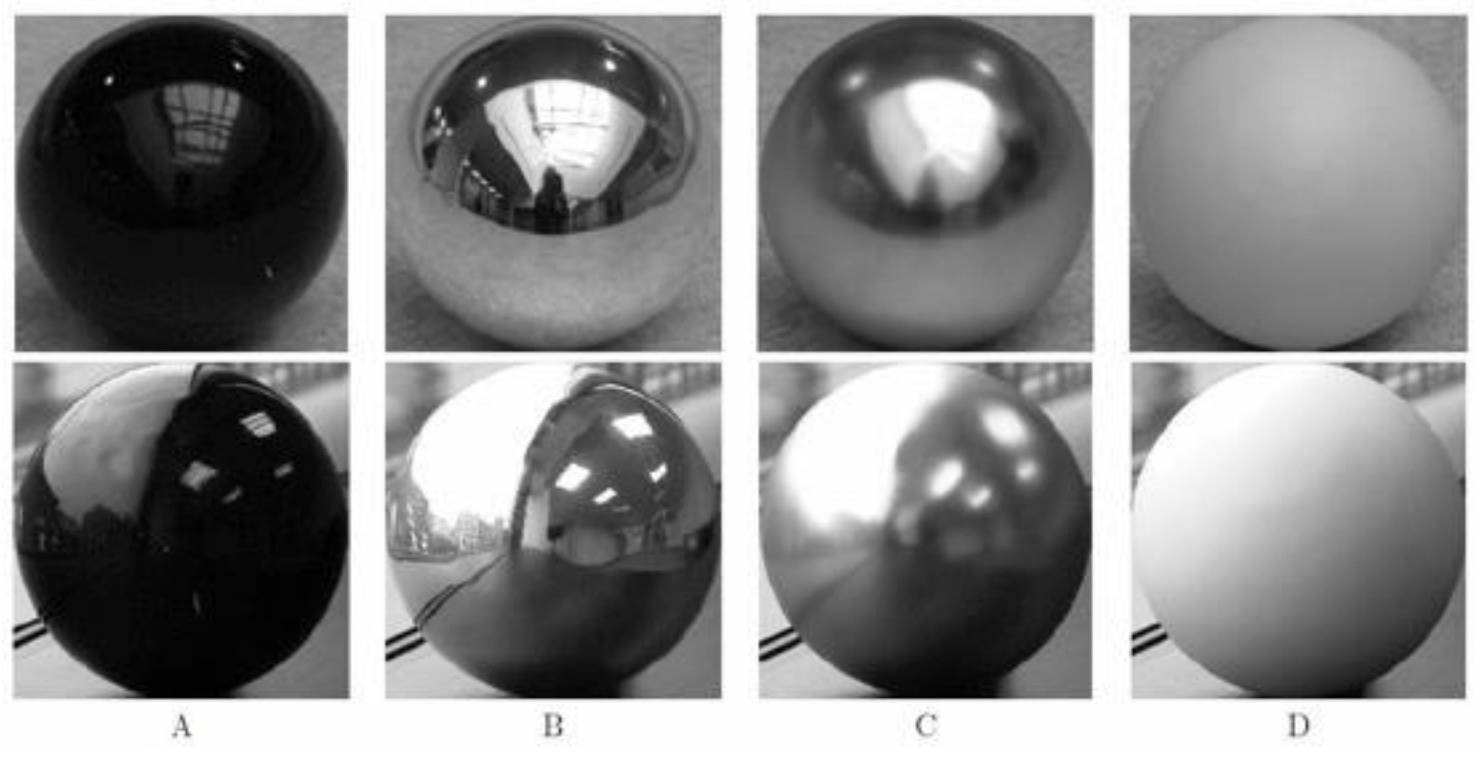
Shading

- Variation in observed color across an object
  - strongly affected by lighting
  - present even for homogeneous material
- caused by how a material reflects light
  - depends on
    - geometry
    - lighting
    - material

KevinMackArt.com

Materials

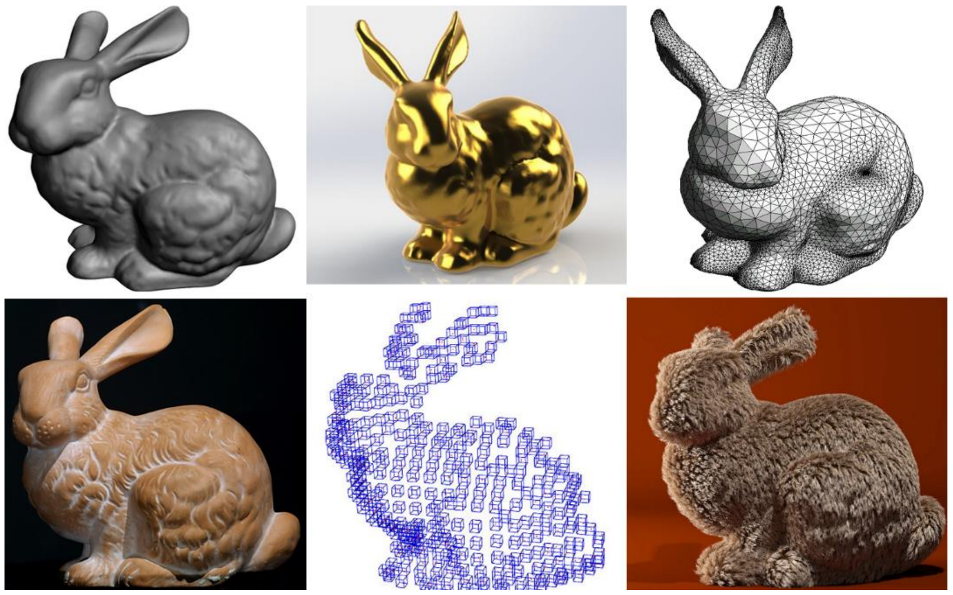
- Human visual system (HVS) is quite good at understanding shading

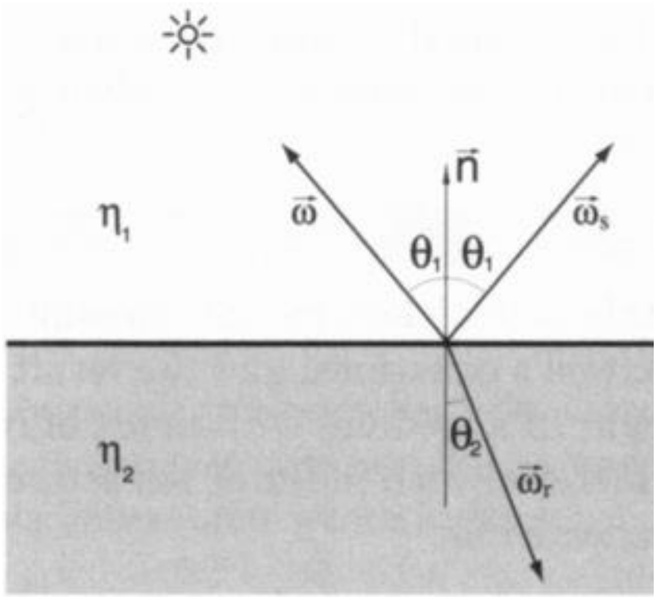


[Dror, Adelson, & Willsky]

Shading problem for CG

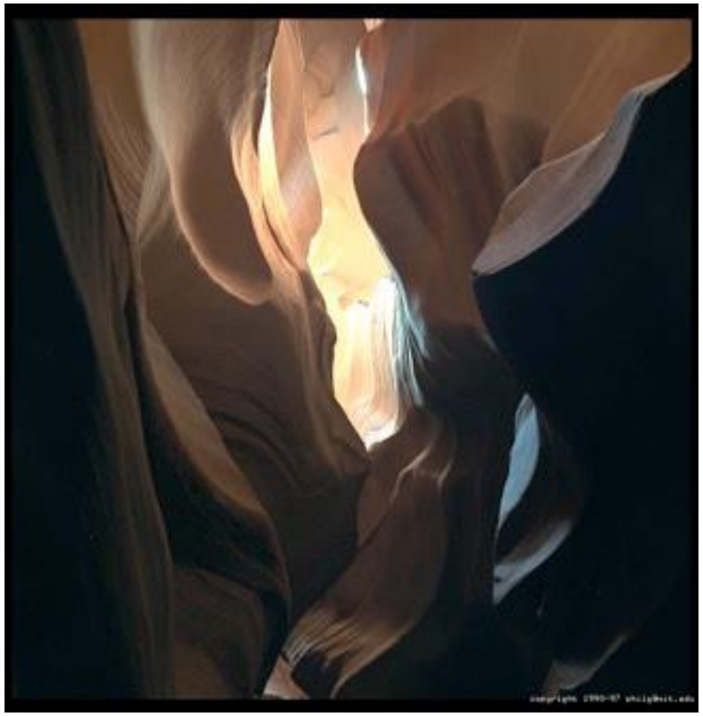
- Need to compute an image
  - of particular geometry
  - under particular illumination
  - from a particular viewpoint
- Basic question: how much light reflects from an object toward the viewer?





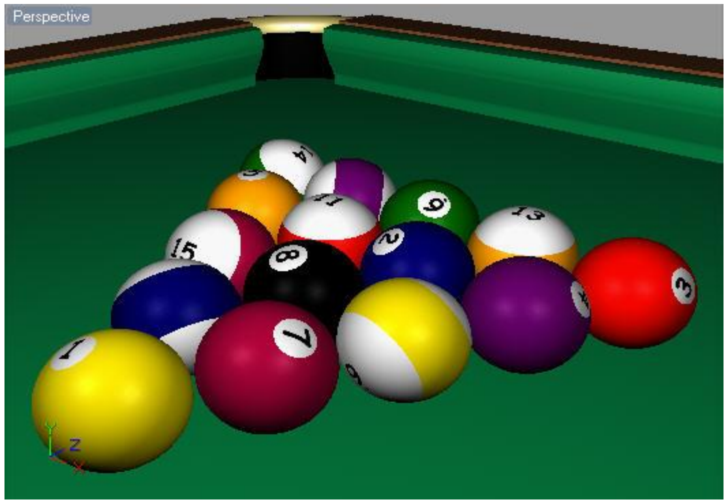
Real Shading

- Complex interactions
  - Even between objects
  - Lot of reflections before reaching the eye
- Scattered light, reflection, absorption etc.
  - Impossible to simulate accurately!



Illumination model

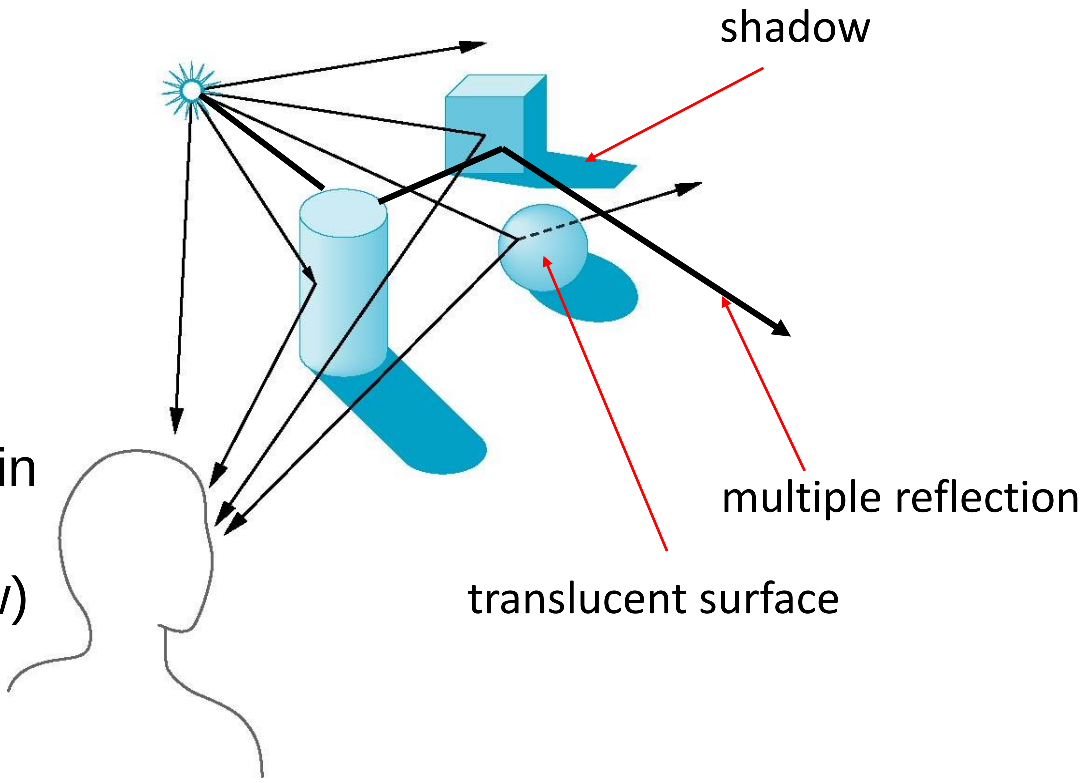
- Global illumination model
  - Simulate not only the direct illuminations but also the indirect illuminations
- Local illumination model
  - Considers light sources and surface properties only

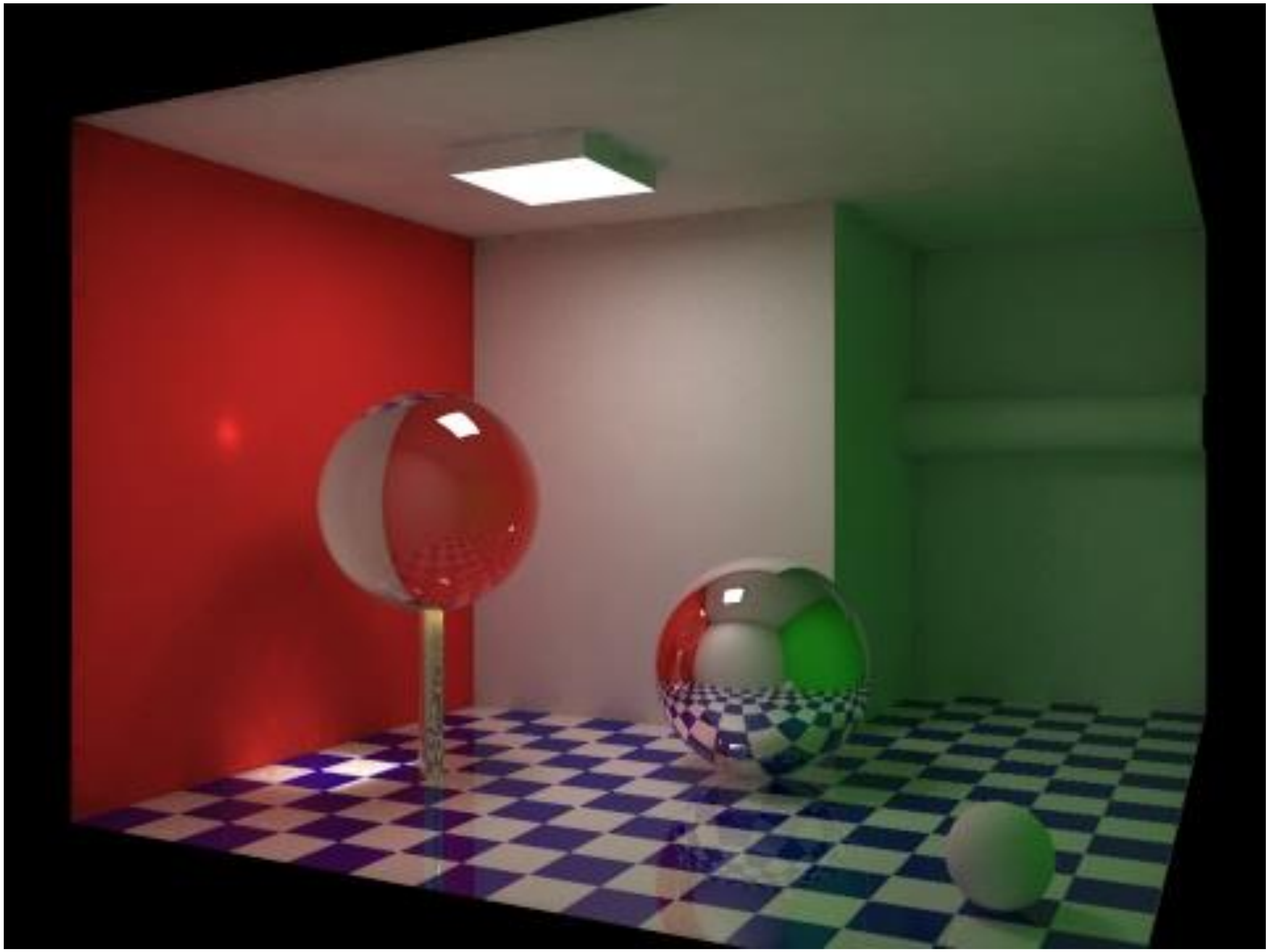


RENDERED USING DALI - HENRIK WANN JENSEN 2000

Global Illumination (GI) methods

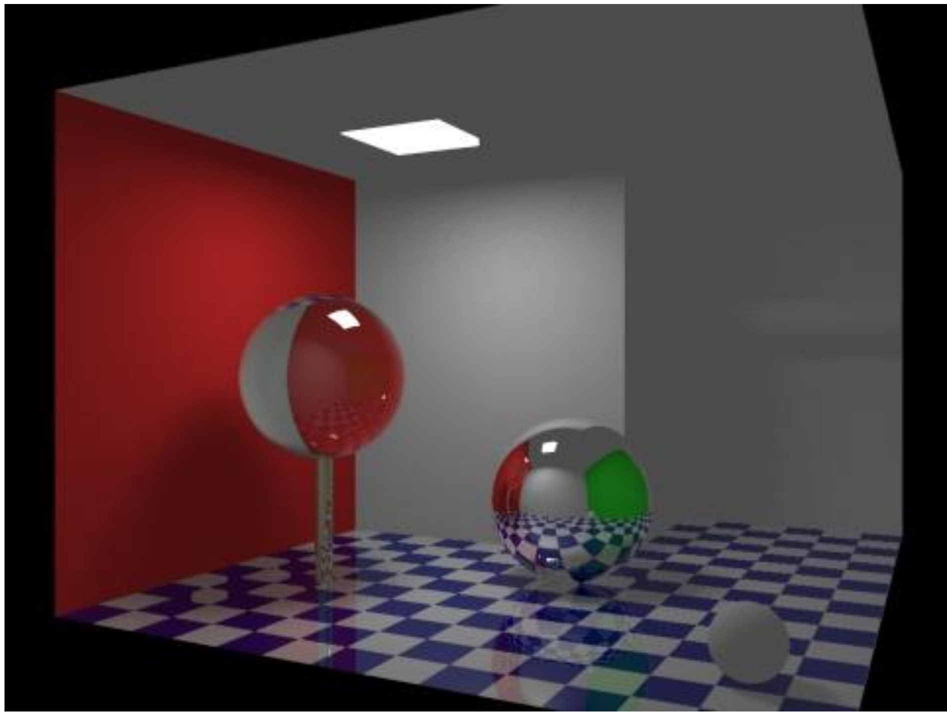
- Ray-tracing
- Radiosity
- Photon Mapping
- Can handle
  - Reflection (one object in another)
  - Refraction (Snell's Law)
  - Shadows
  - Color bleeding
- More computation and slow





Local Illumination methods

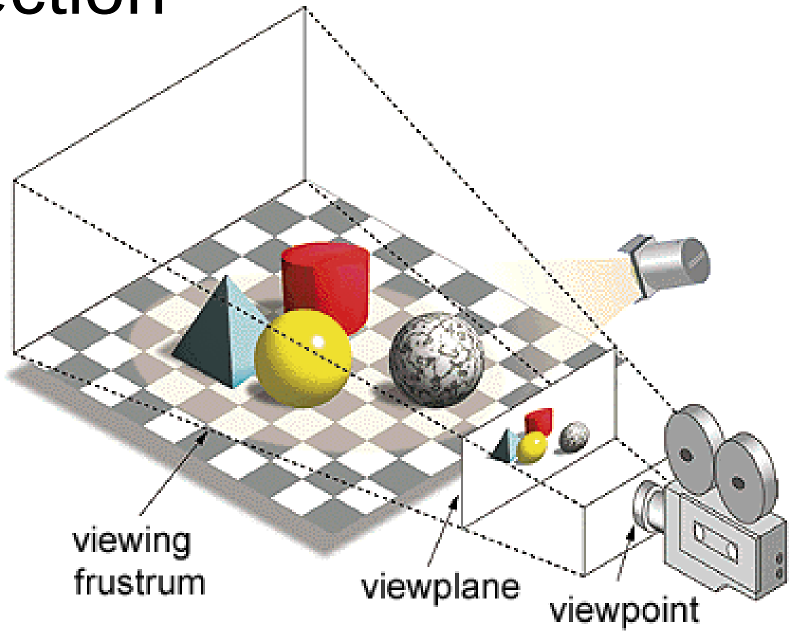
- Gouraud shading
- Phong shading
- Shadow techniques
- Can approximate GI!
  - Environmental Mapping
  - Ambient occlusion
  - Image based lighting
- Fast and real-time
- Not as accurate as GI

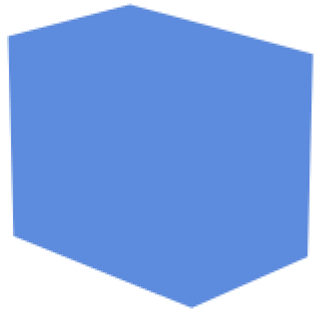


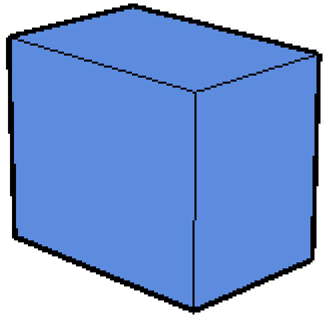
Local illumination model

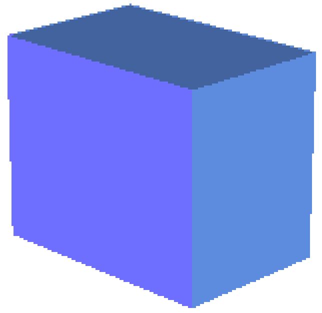
- Light sources
- Geometry
- Material
- Viewing Direction

From Computer Desktop Encyclopedia
Reprinted with permission.
© 1998 Intergraph Computer Systems



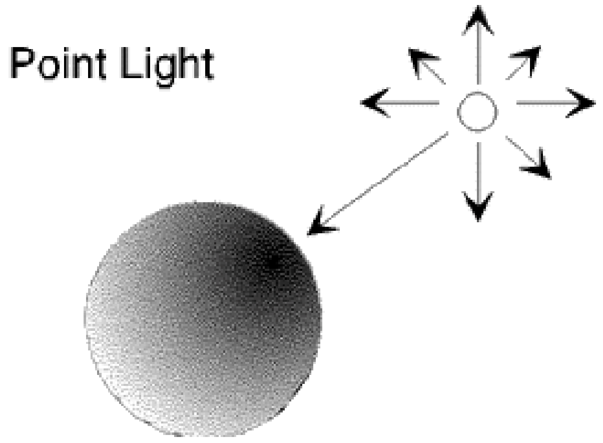






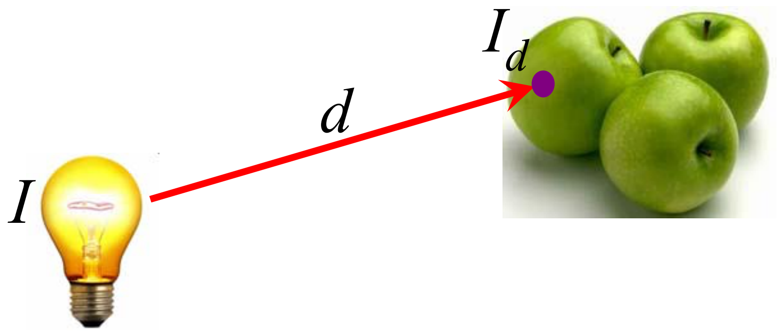
Light sources

- Point Light
- Directional Light
- Spotlight Light
- Area Light
- Volumetric Light



Point light

- The most simple one
- It is omni-directional
- Attributes to specify a point light source
  - Position (*px, py, pz*)
  - Intensity *I* (if it is a chromatic light, three values representing R, G, and B are needed (I_r, I_g, I_b))
  - Coefficients (*a*0, *a*1, *a*2) to specify its attenuation property with distance *d*

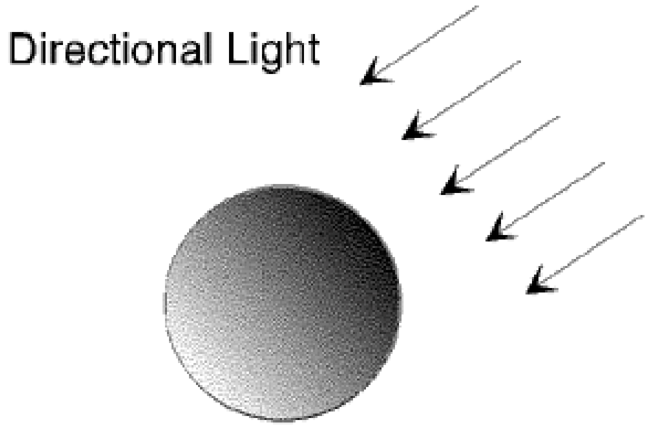


$$f_{radatten} = \frac{1}{a_0 + a_1 d + a_2 d^2}$$

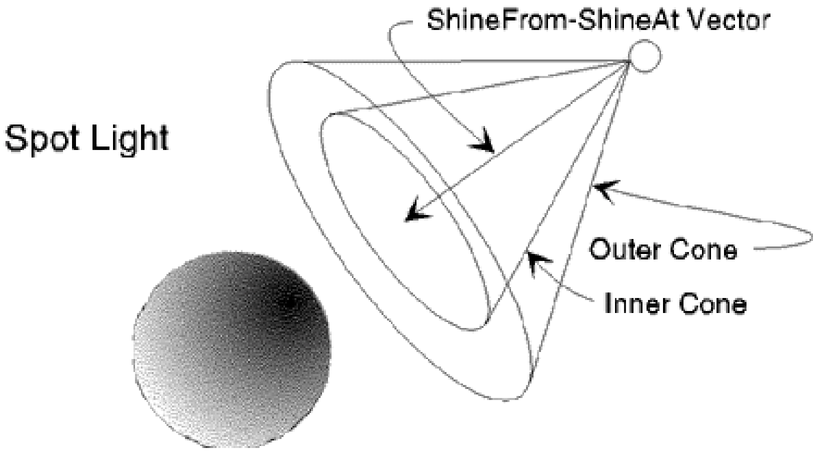
$$I_d = f_{radatten} I$$

Directional light

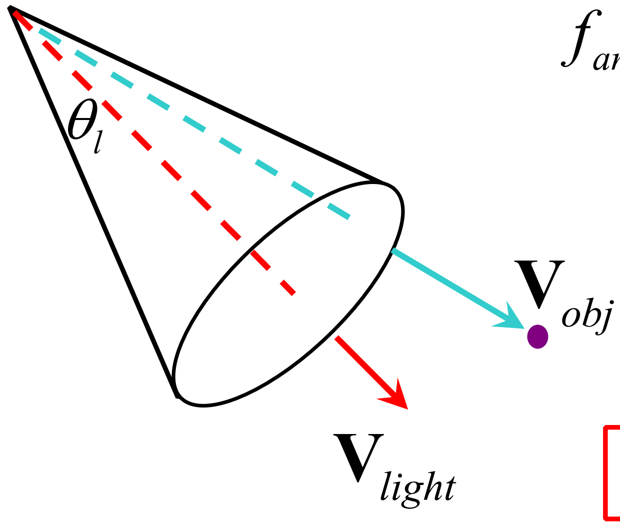
- This type can be imagined as a point light source lying in infinity, e.g. Sun light
- Light rays from such a source are radiated in parallel.
- Attributes to specify a directional light source
  - Direction *V* (vx, vy, vz)
  - Intensity *I*
  - No attenuation Why?



Spotlight



- Besides position and color
- the apex angle of the lighting cone needs to be specified
- (*px*, *py*, *pz*), *I*, (*tx*, *ty*, *tz*) and Theta

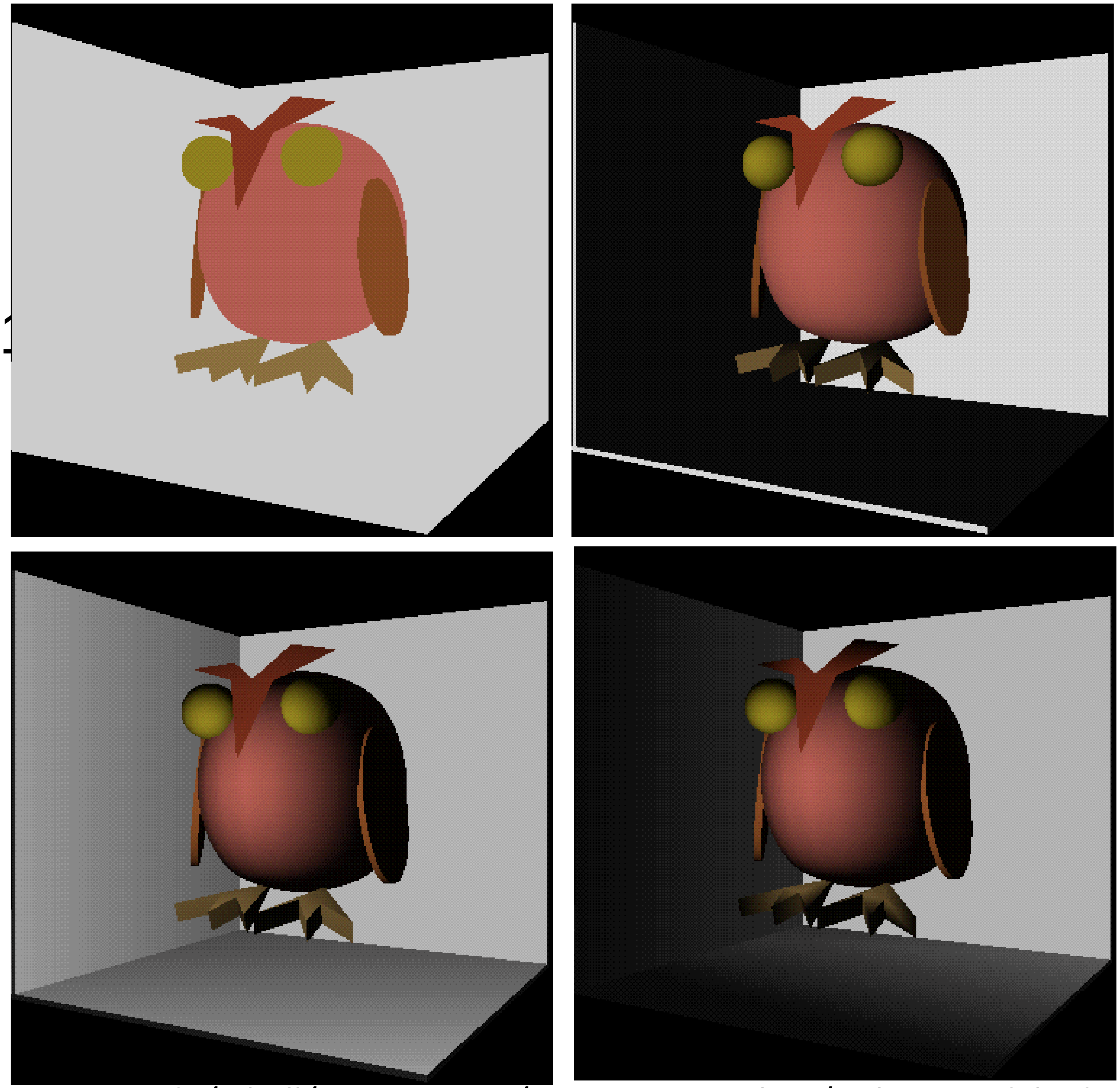


$$f_{angatten} = \begin{cases} 0.0, \textit{when } \mathbf{V}_{obj} \cdot \mathbf{V}_{light} < \cos\theta_l \\ \left(\mathbf{V}_{obj} \cdot \mathbf{V}_{light}\right)^{\alpha}, \textit{otherwise} \end{cases}$$

where α is a constant

$f_{radatten}, f_{angatten}$ can be combined

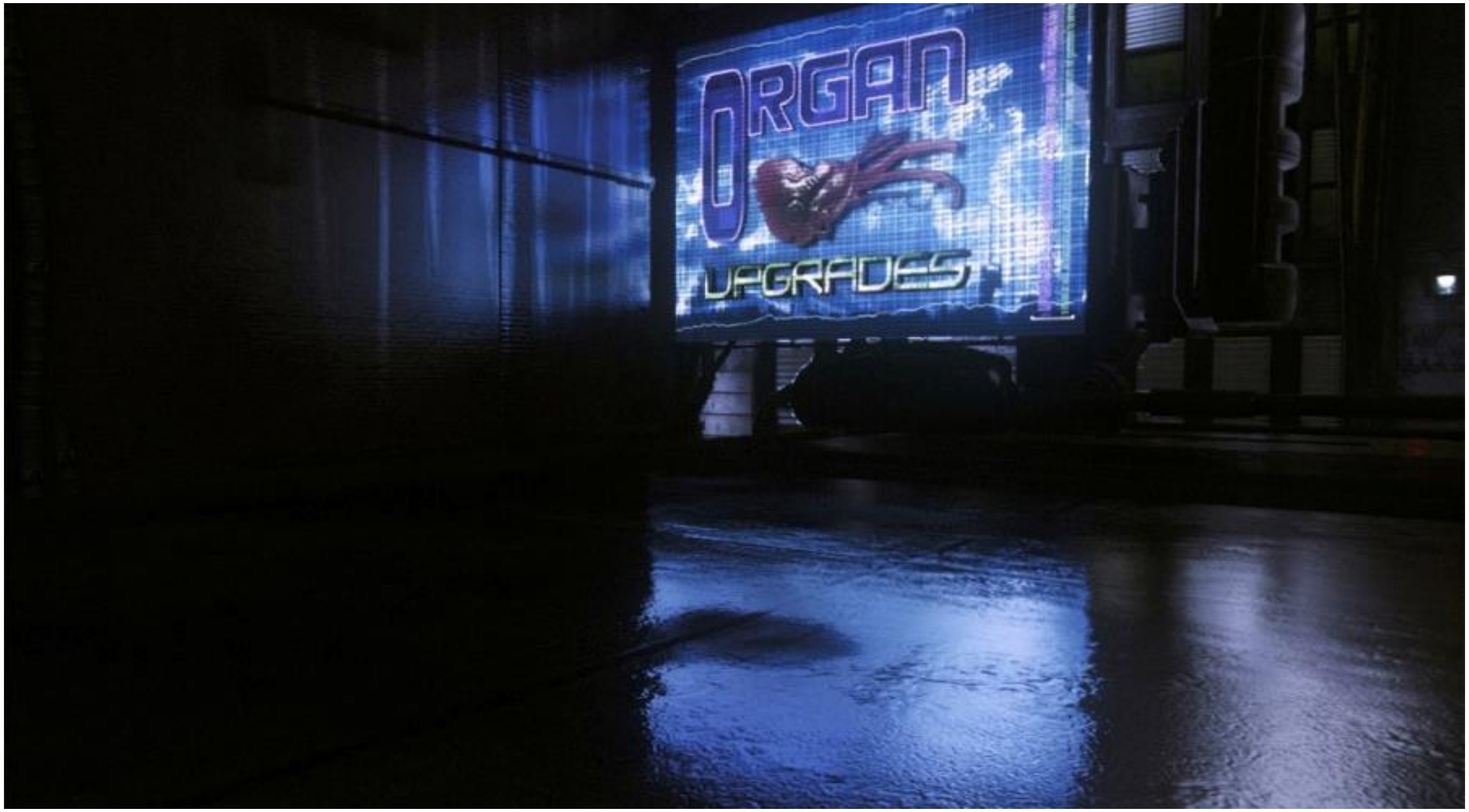
- 1



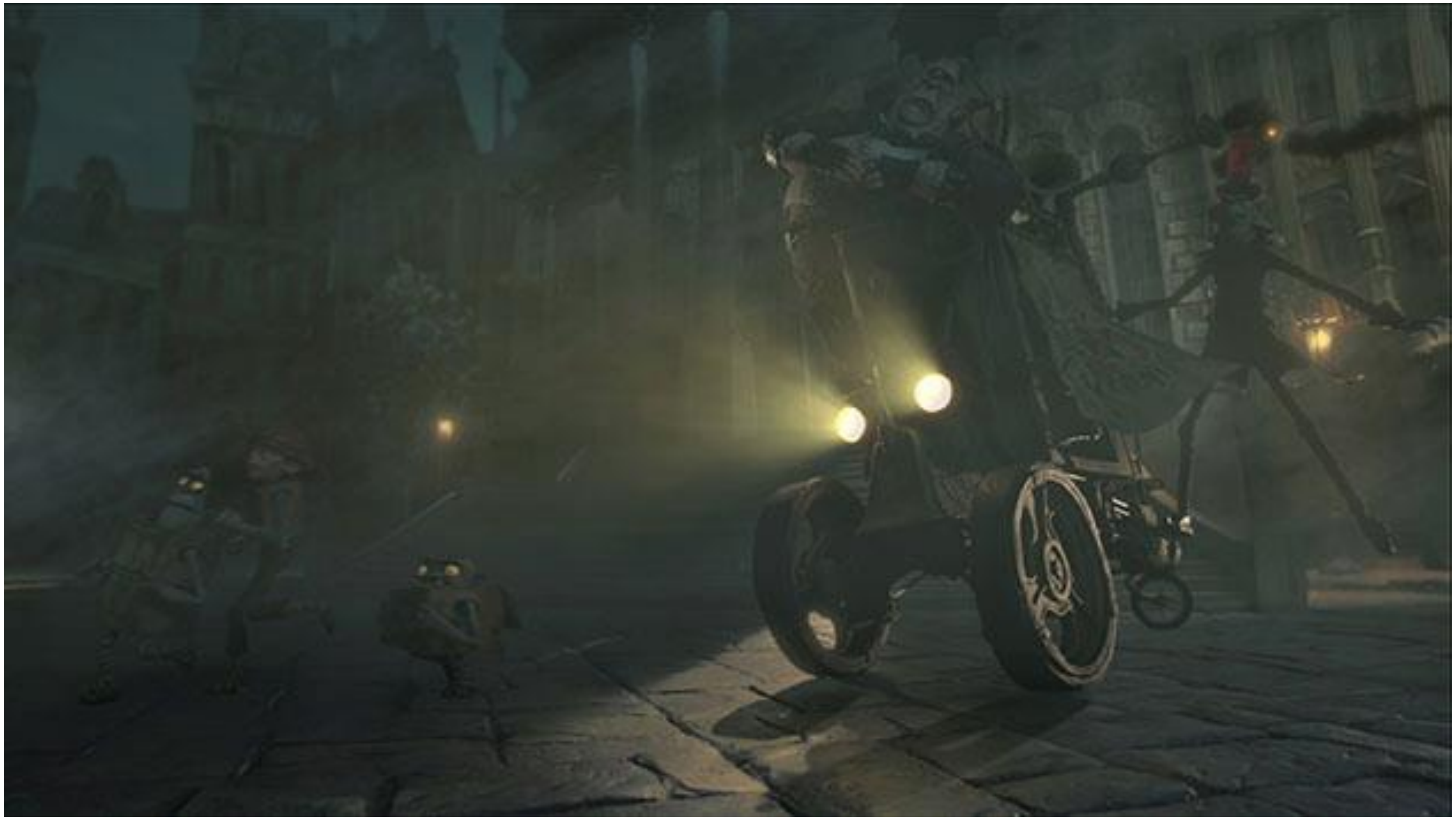
http://www.cs.uic.edu/~jbell/CourseNotes/ComputerGraphics/LightingAndShading.html

- Questions?

Area light source



Volumetric light source

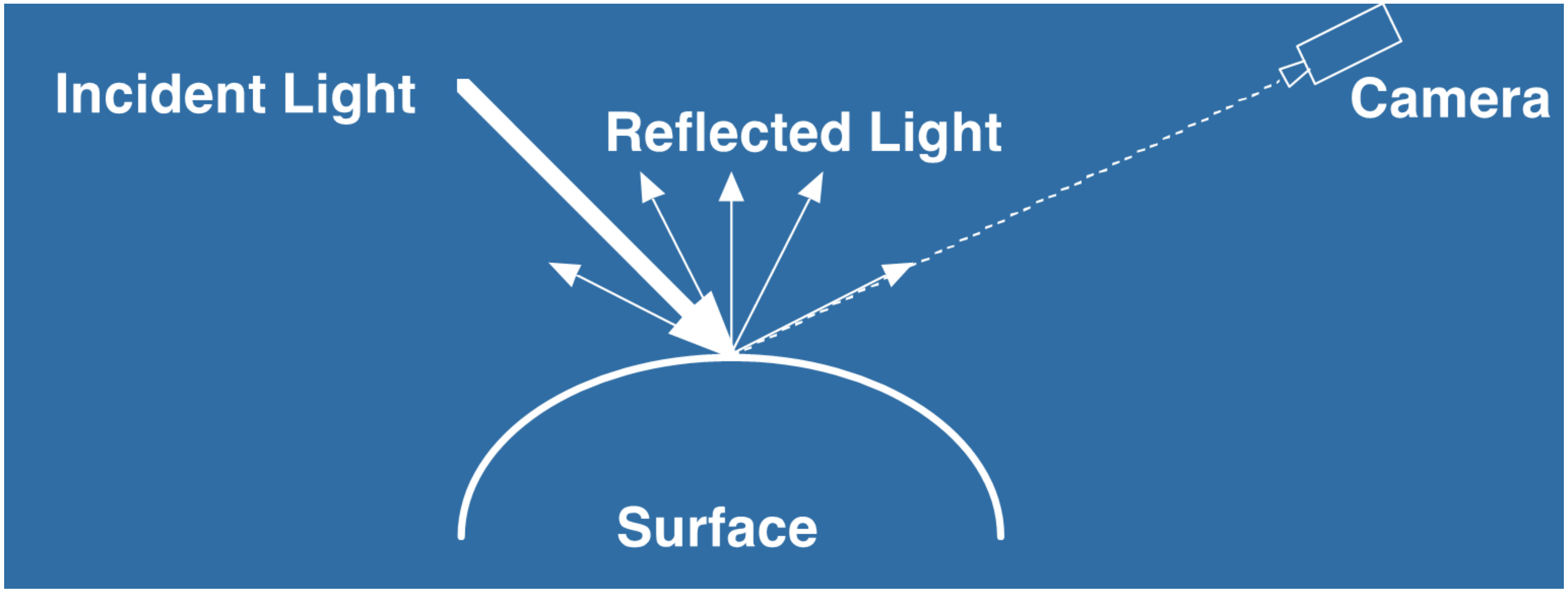


- For any of these light sources, it is easy to compute illumination arriving at a point (e.g., a vertex)
  - How?
- Now we can think about how the incoming light is reflected by a surface

Surface reflection

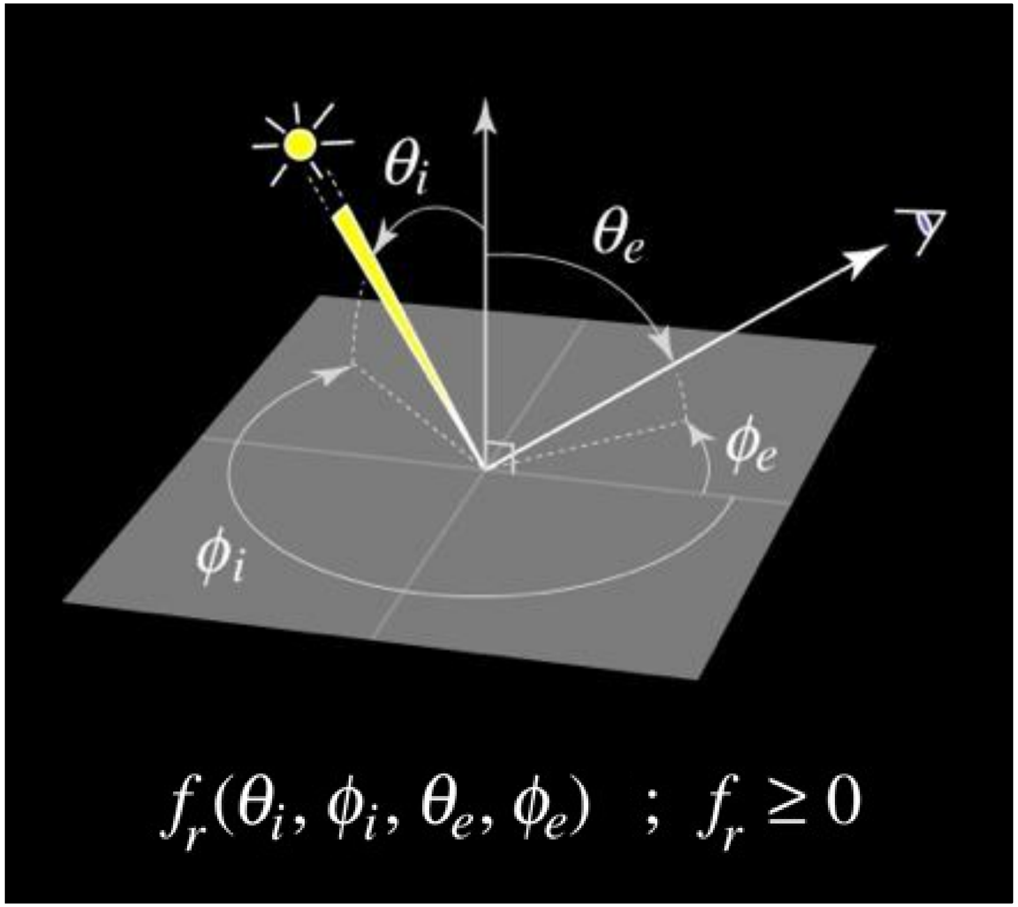
- When light hits an opaque surface some is absorbed, the rest is reflected (some can be transmitted too)
- The reflected light is what we see
- Reflection is not simple and varies with material
  - the surface’s micro structure define the details of reflection
  - variations produce anything from bright specular reflection (mirrors) to dull matte finish (chalk)

Surface reflection



Physical-based Rendering (PBR)

- Bidirectional reflectance distribution function (BRDF)


θ_i
θ_e
ϕ_e
ϕ_i
$f_r(\theta_i, \phi_i, \theta_e, \phi_e) \;\; ; \;\; f_r \geq 0$

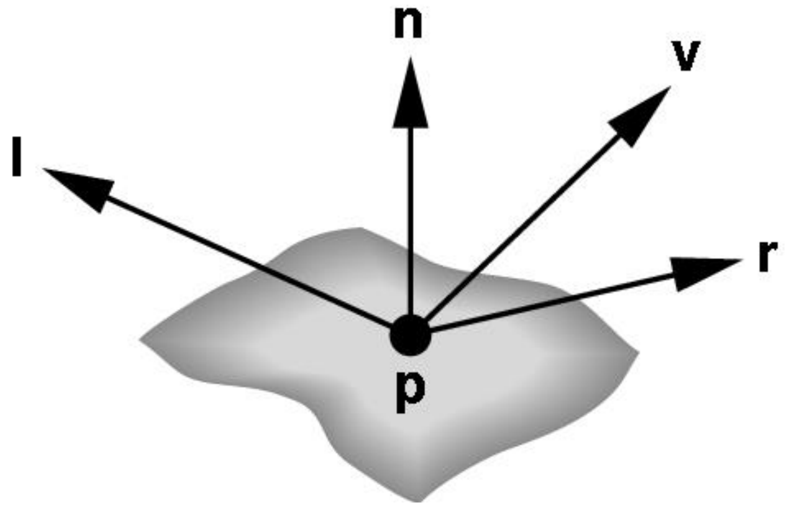
2008 Steve Marschner

Phong reflection model

- Developed by Bui Tuong Phong (裴祥风) at the University of Utah 1973　(Vietnamese, 1942-1975)
- By Phong model, the light energy emitted from a point on an object to the observer’s eye comprises three components
    - Ambient lighting
    - Diffuse reflection
    - Specular reflection

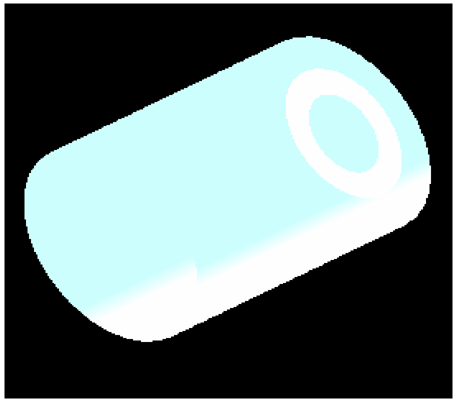
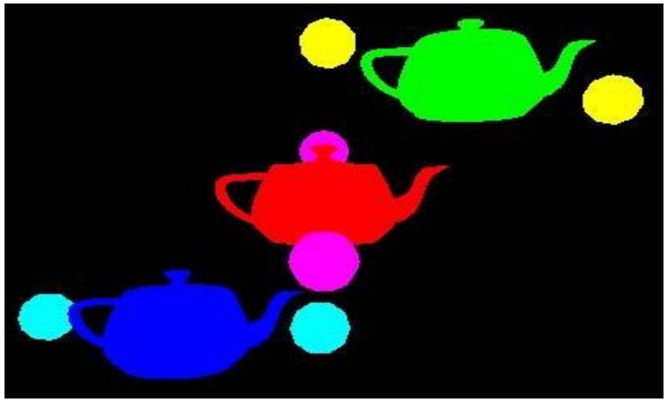
Phong reflection model

- Empirical Model
- Calculate color for arbitrary point on surface
- Basic inputs are material properties and l, n, v:
  - l = vector to light source
  - n = surface normal
  - v = vector to viewer
  - r = reflection of l at p
  (determined by l and n)



Ambient reflection

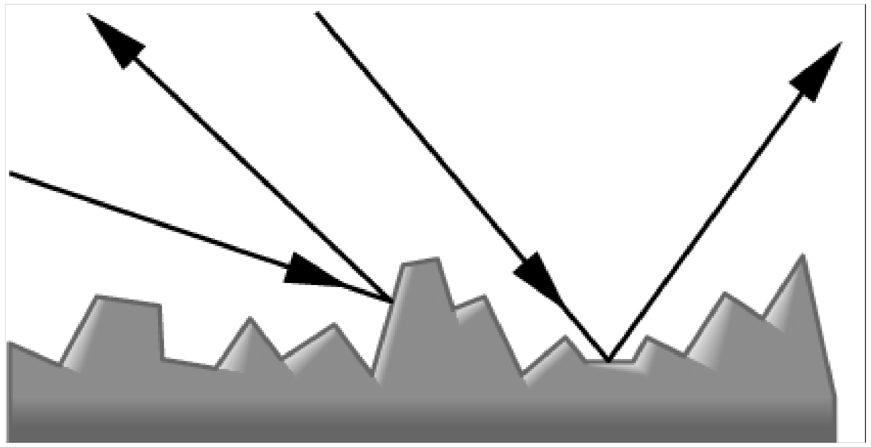
- Ambient reflection is a constant for a scene L_a
- Different surface can have different ambient reflection coefficient k_a ($0 \le k_a \le 1$)
- So, if only consider ambient lighting, the illumination at a point simply is $I_a = L_a * k_a$

Rendering results when only ambient lighting is considered

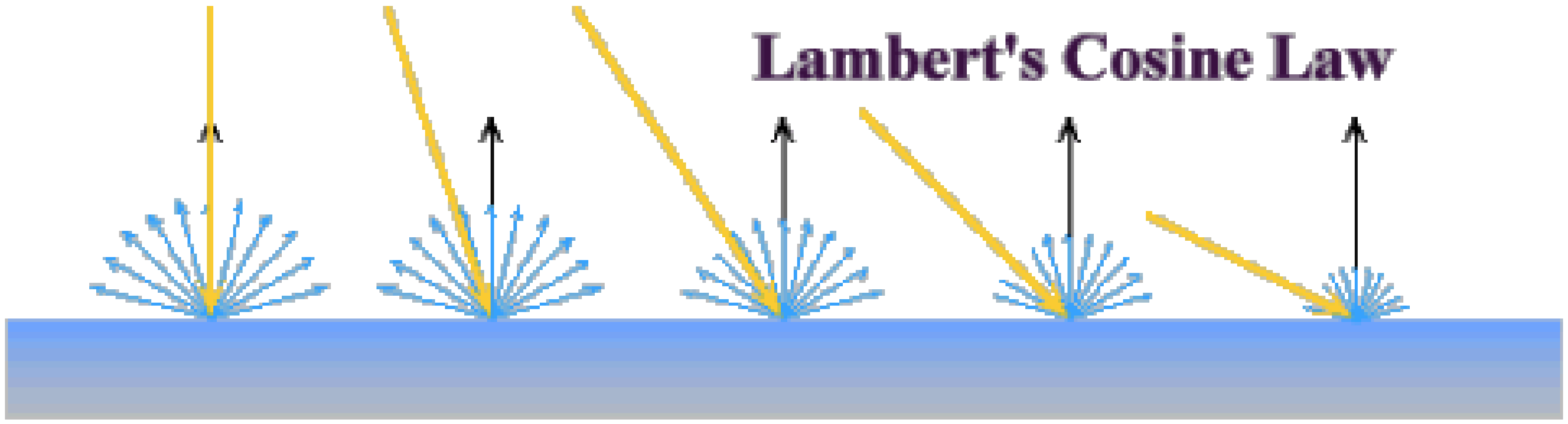
Diffuse reflection

- Diffuse reflector scatters light
- Assume equally all direction
- Called Lambertian surface
- Diffuse reflection coefficient k_d, $0 \leq k_d \leq 1$
- Angle of incoming light still critical



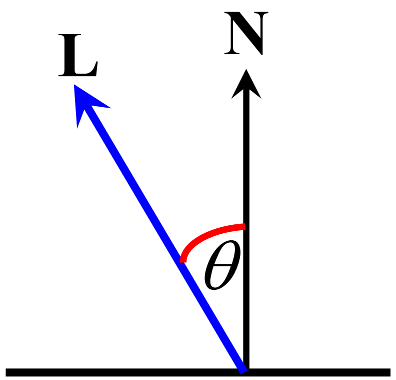
Diffuse reflection

- Lambert’s Law
- Intensity depends on angle of incoming light



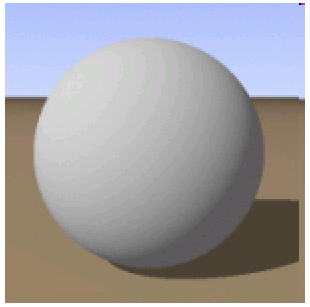
Diffuse reflection

- Lambert's Law
- Intensity depends on angle of incoming light
- Recall
  - l = unit vector to light
  - n = unit surface normal
  - θ = angle to normal
- $\cos\theta = l \bullet n$
- $I_d = k_d\,(l \bullet n)\,L_d$
- If $\theta > 90^\circ$ $(l \bullet n) < 0$, $I_d = 0$

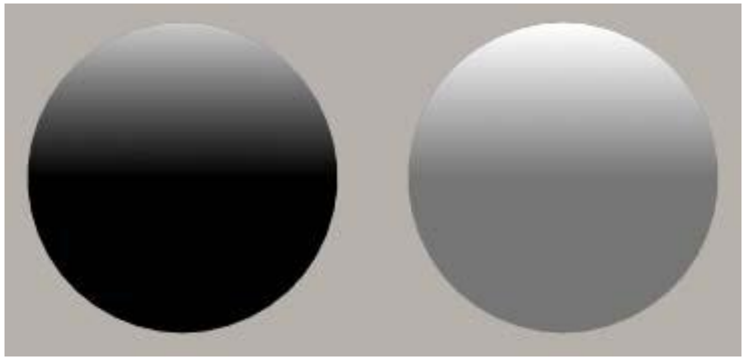


Combine I_a and I_d

- $I = L_a k_a + L_d k_d \max(l \bullet n, 0)$
- Lack of highlight!



Illuminated by ambient lighting and diffuse reflection

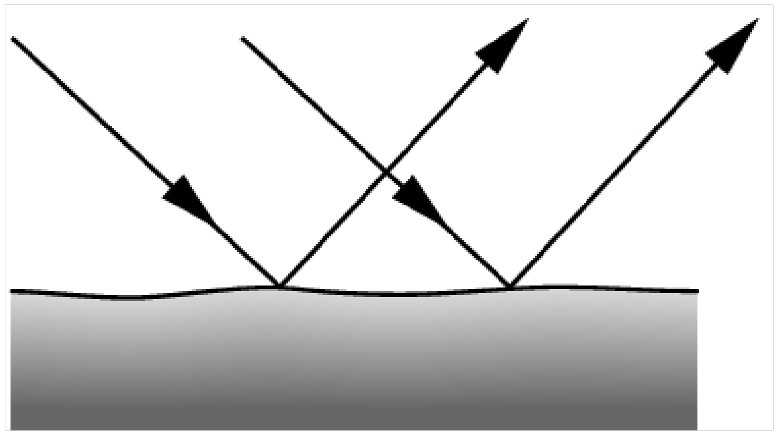
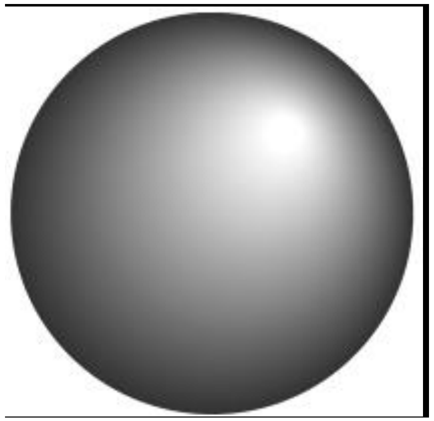


Left: diffuse reflection only

right: ambient + diffuse reflection

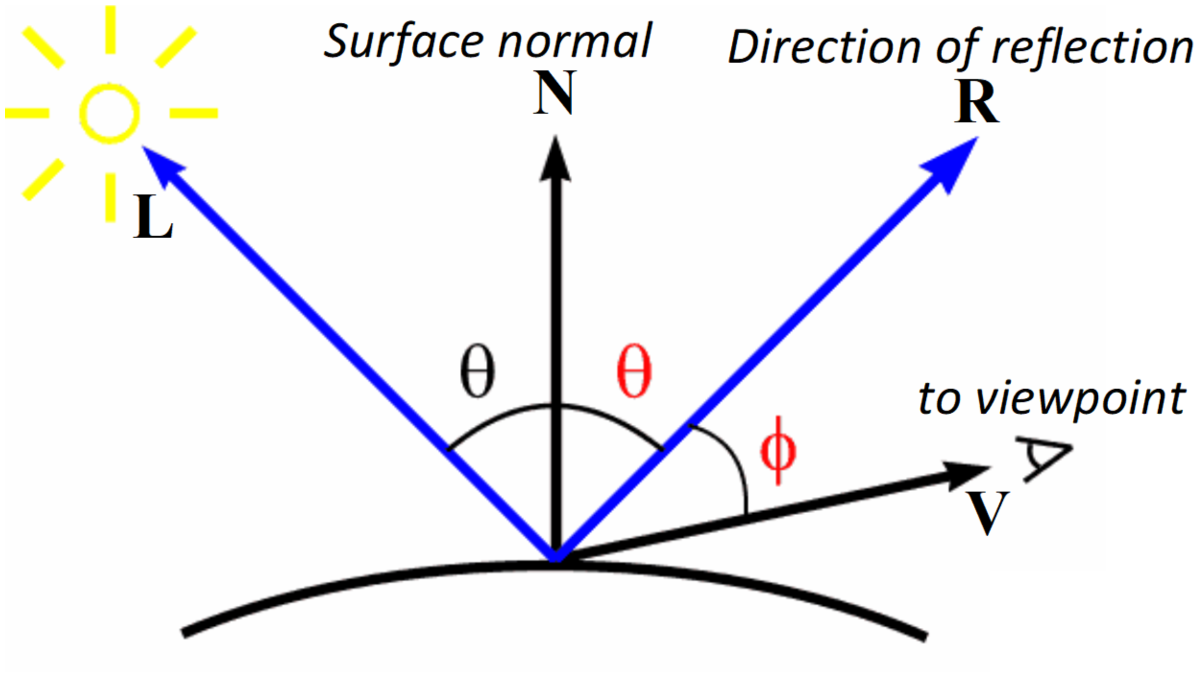
Specular reflection

- Specular reflection coefficient k_s, $0 \le k_s \le 1$
- Shiny surfaces have high specular coefficient
- Used to model specular highlights
- Do not get mirror effect (need other techniques)

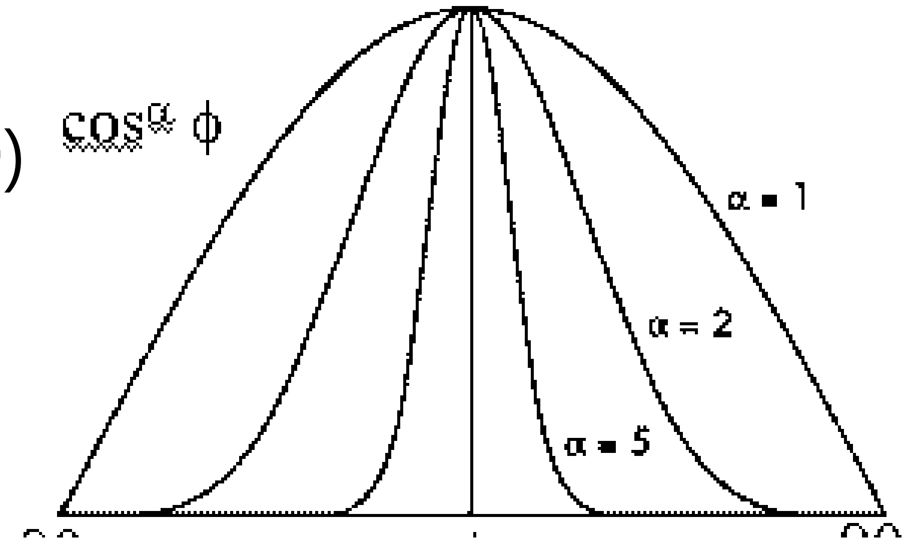
Specular reflection

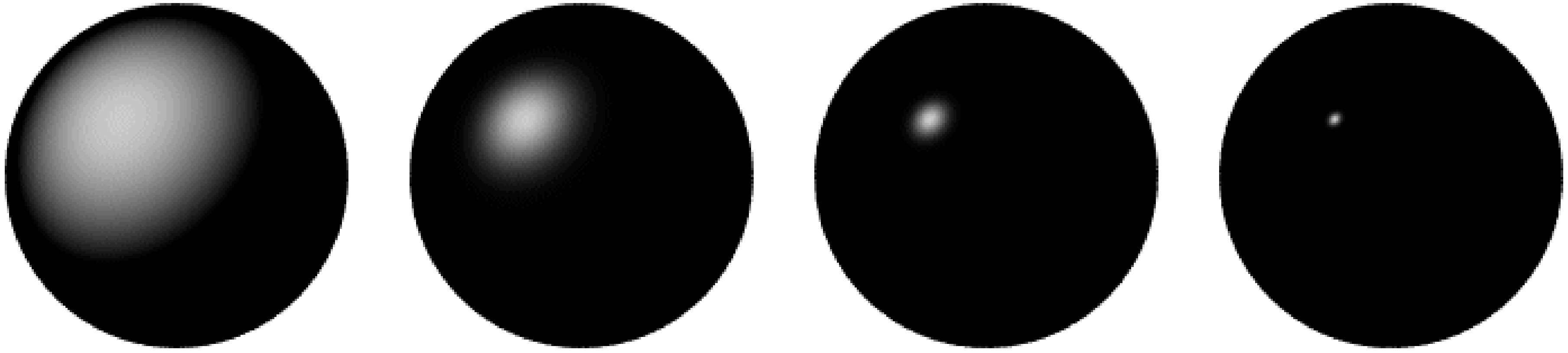
- $I_s = \max(k_s\ L_s \cos^{\alpha}\phi, 0.0)$
- $\cos\phi = r \bullet v$
- *What is α?*



Shininess coefficient

- Higher α is narrower
- $I_s = \max(k_s\ L_s \cos^{\alpha}\phi, 0.0)$





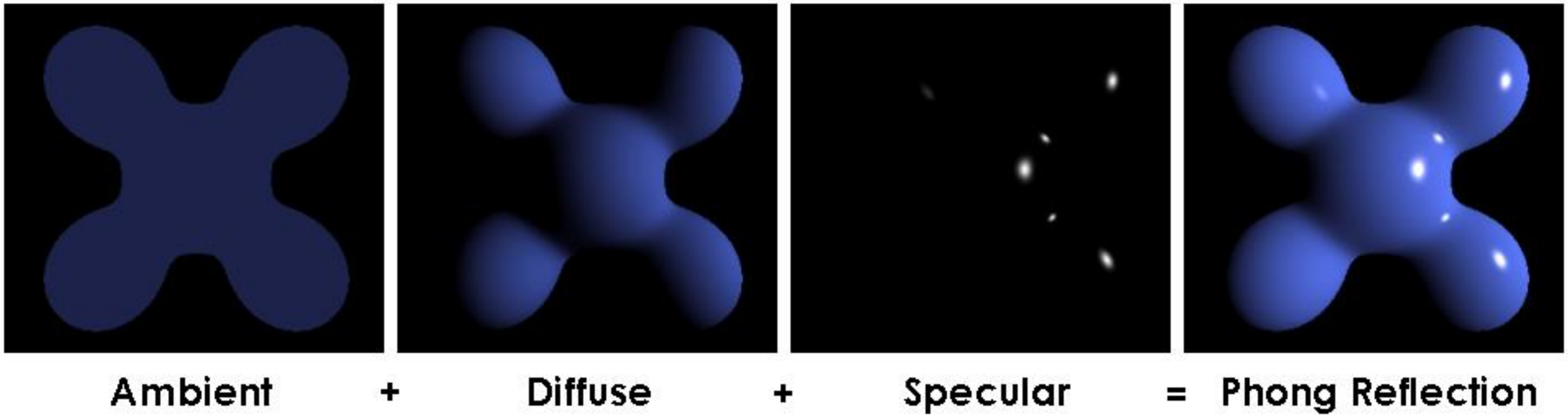
Summary of Phong reflection

- Light components
  - Ambient I_a, Diffuse I_d and Specular I_s
- Material coefficients for each light component
  - k_a, k_d , k_s and α
- Therefore:
- $I = L_a k_a + L_d k_d \max(l \bullet n, 0) + L_s k_s \max(r \bullet v)^{\alpha}, 0)$

Summary of Phong reflection

- Attenuation
- $I_{atten} = I_a + \frac{1}{a_0 + a_1 d + a_2 d^2}(I_d + I_s)$
- How about If there are multiple lights?
- $I = L_a k_a + \sum_{i=1}^{m} (L_{d,i}\, k_d \max(l \bullet n, 0) + L_{s,i}\, k_s \max((r_i \bullet v)^{\alpha}, 0)$
- How about shadows?
- $I = L_a k_a + s_i \sum_{i=1}^{m} (L_{d,i}\, k_d \max(l \bullet n, 0) + L_{s,i}\, k_s \max((r_i \bullet v)^{\alpha}, 0)$

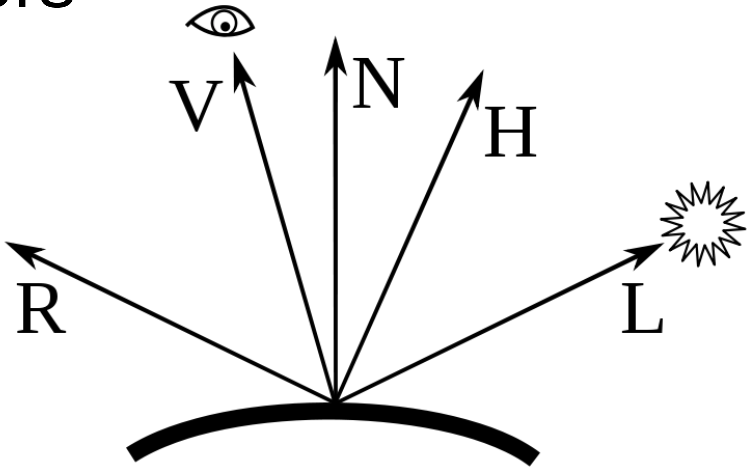
Phong reflection model



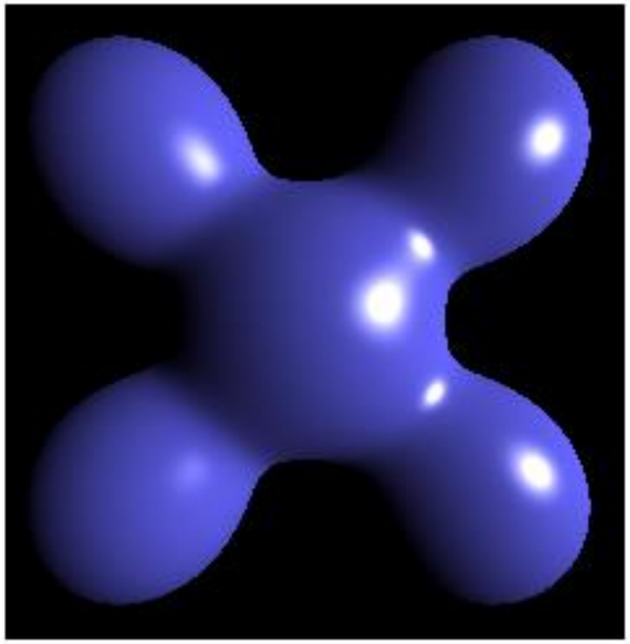
Blinn-Phong reflection model

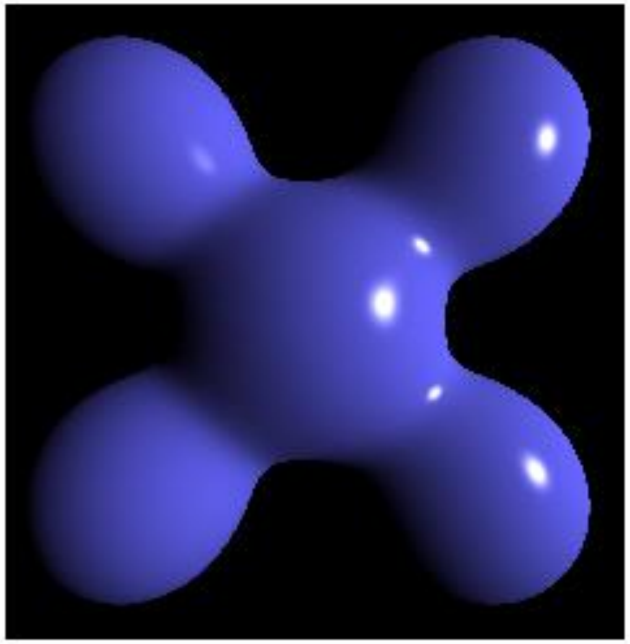
- A modification to the Phong reflection model developed by Jim Blinn
- In Phong model, one must continually recalculate the dot product of r and v. Instead, one calculates a halfway vector between the viewer and light-source vectors

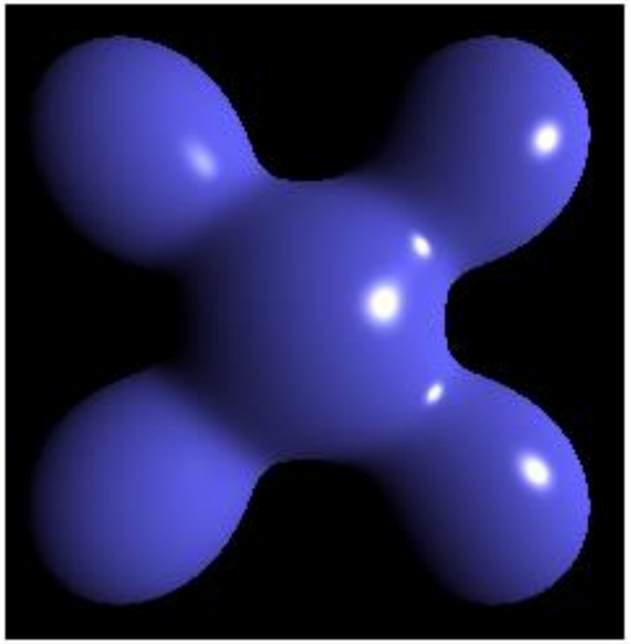
$$\mathbf{H} = \frac{\mathbf{L} + \mathbf{V}}{\left\|\mathbf{L} + \mathbf{V}\right\|}$$



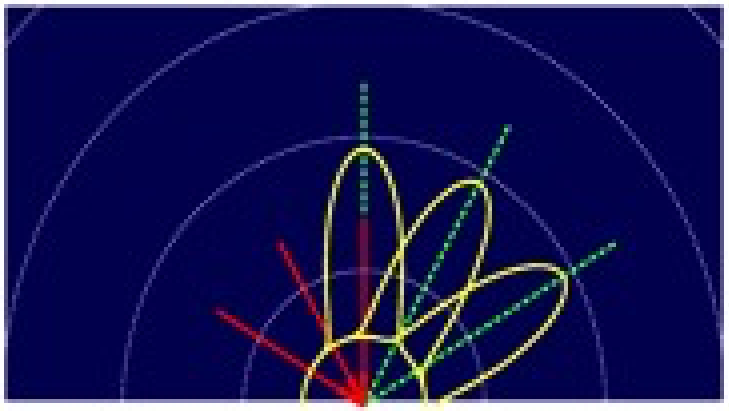
- $L_s k_s(r \bullet v)^{\alpha}$ -> $L_s k_s(n \bullet h)^{\alpha}$


Blinn–Phong

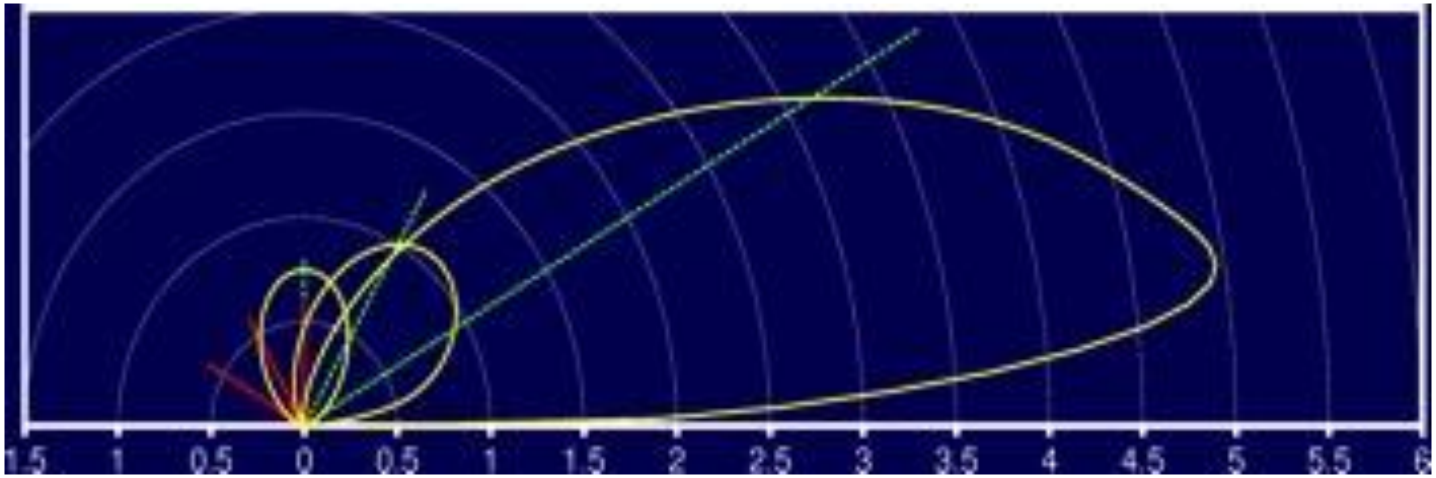

Phong


Blinn–Phong
(higher exponent)

Phong model



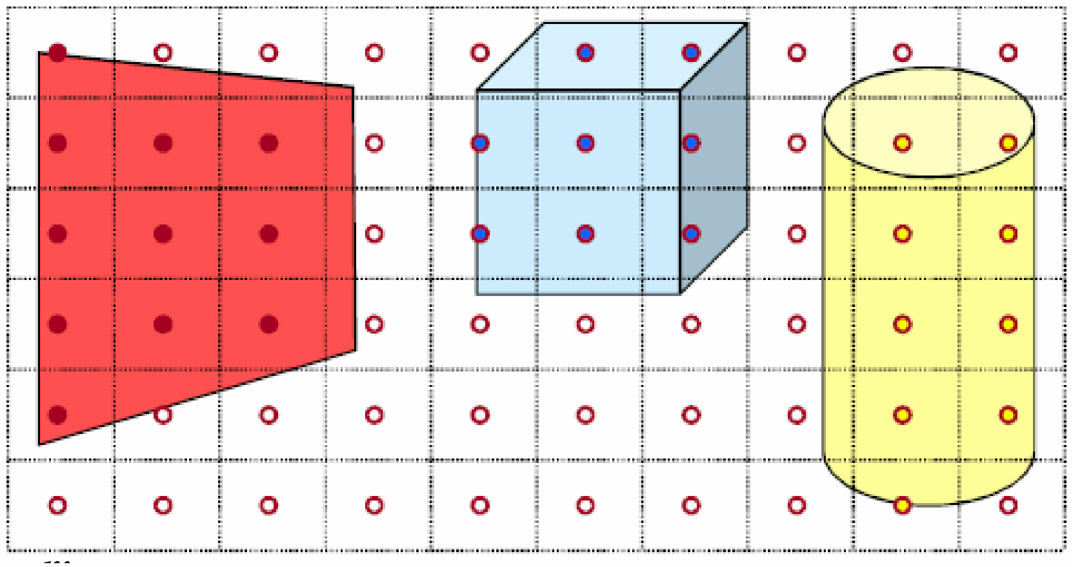
Physics-based model



- Questions?

Surface shading

- How do we choose a color for each filled pixel if the object is represented in a set of polygonal meshes?
- Recall Rasterization
- Simplest shading approach is to perform independent lighting calculation for every pixel



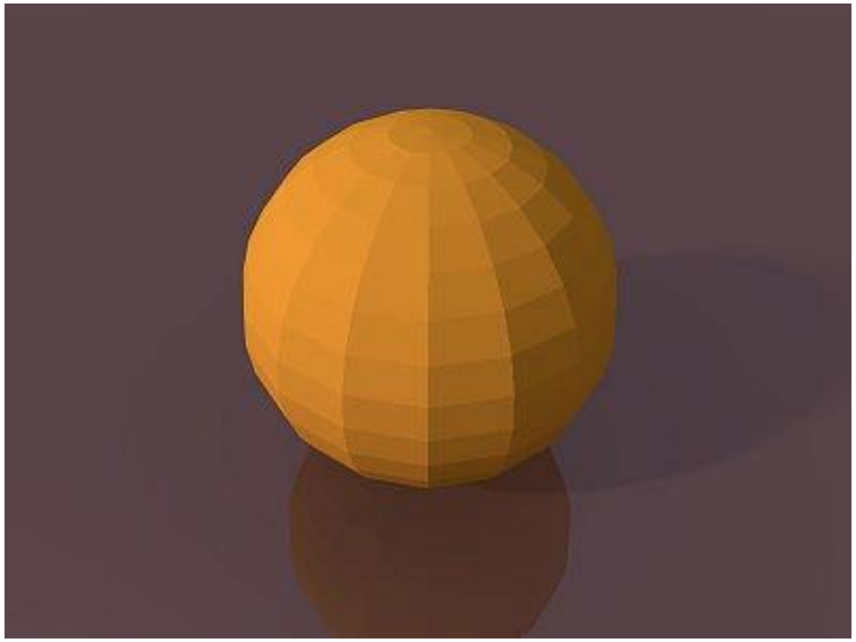
Surface shading

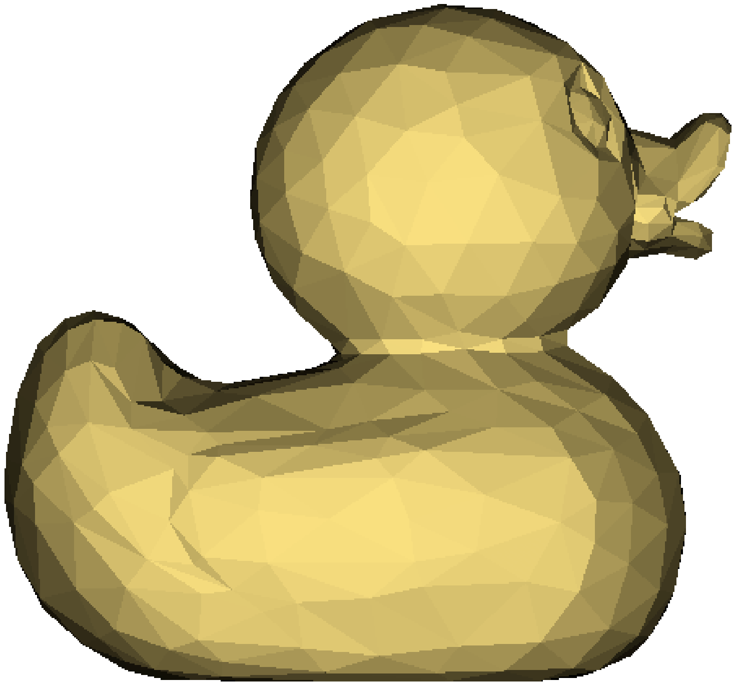
- Common used shading algorithms
  - Flat shading
  - Smooth shading
    - Gouraud shading
    - Phong shading

Flat shading

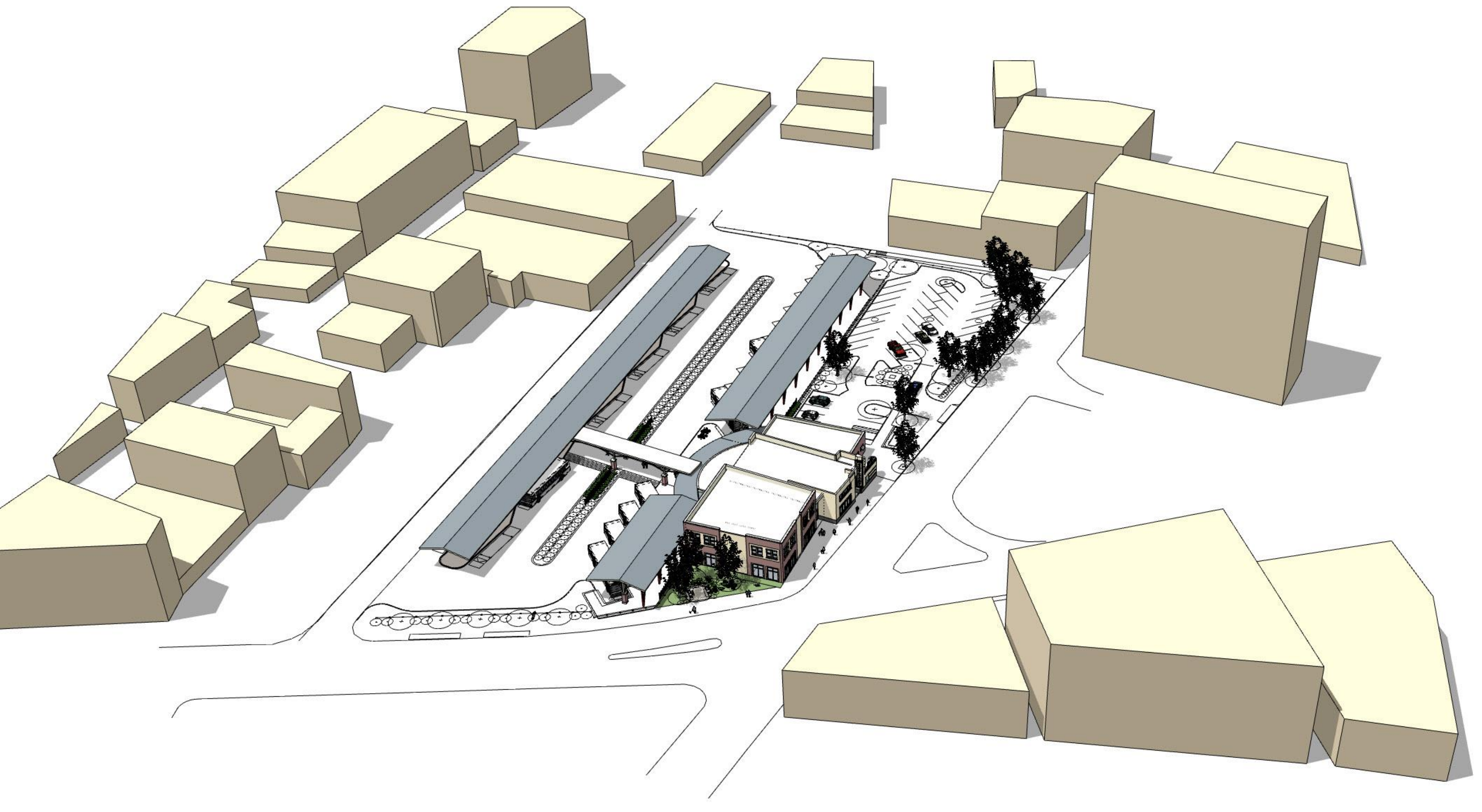
- The simplest one, also called as “constant intensity surface rendering”
- One illumination calculation per polygon
- Assign all pixels inside each polygon the same color, therefore reveal polygons and fast
- OK for polyhedral objects, Not good for smooth surfaces

Examples of flat shading

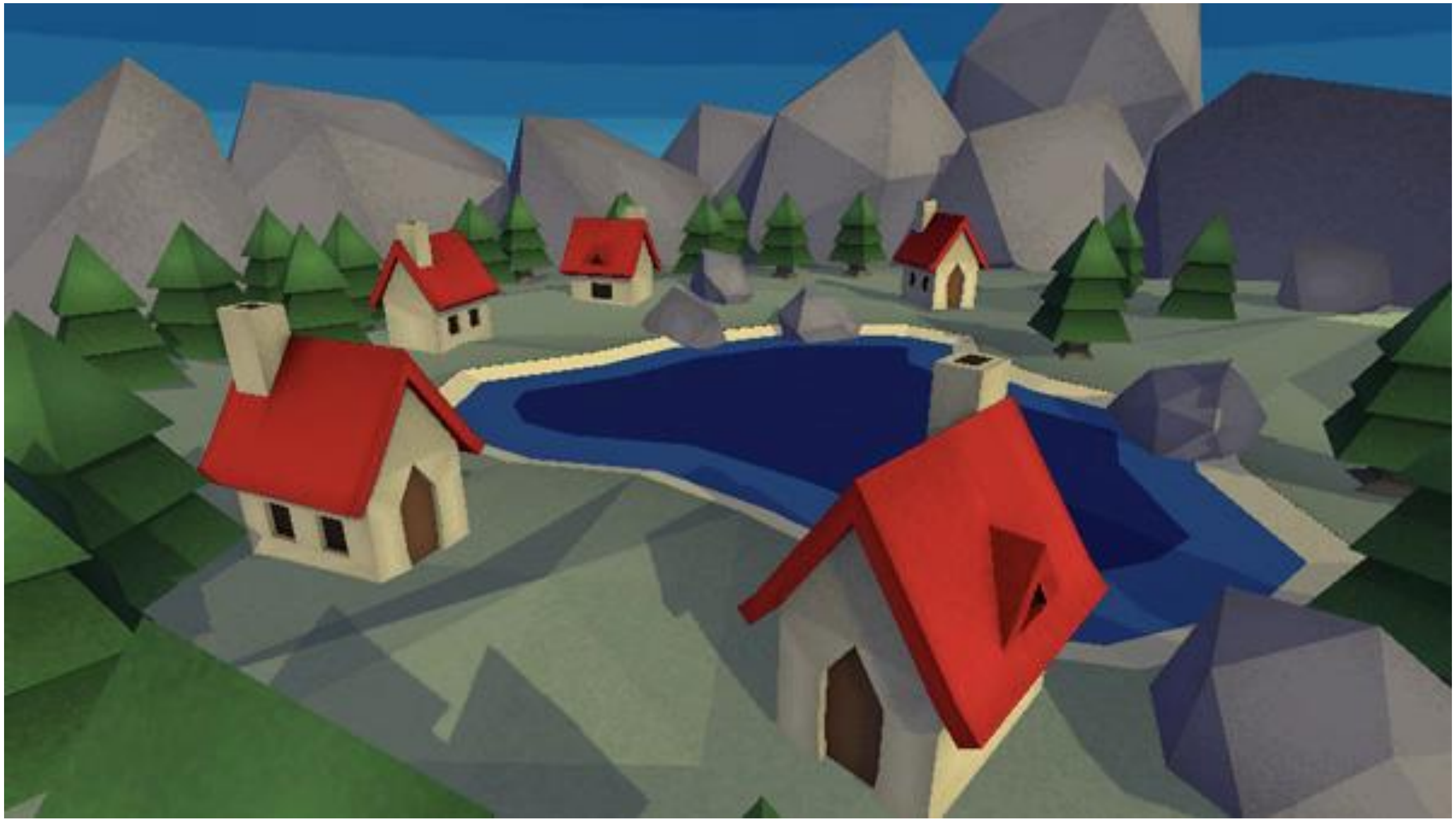




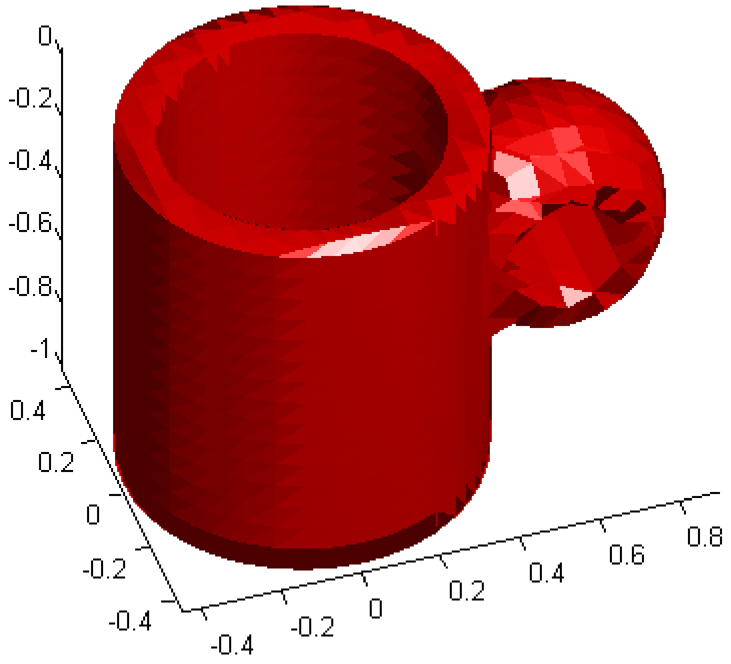
Examples of flat shading

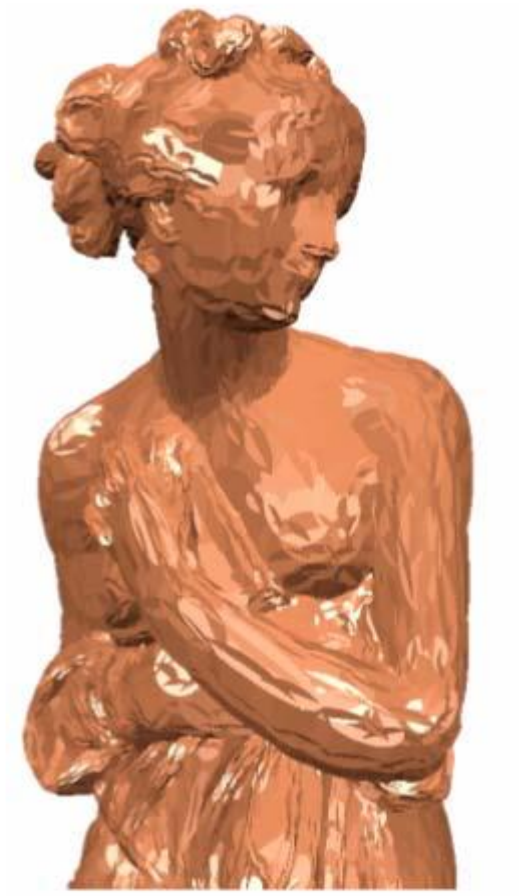


Examples of flat shading



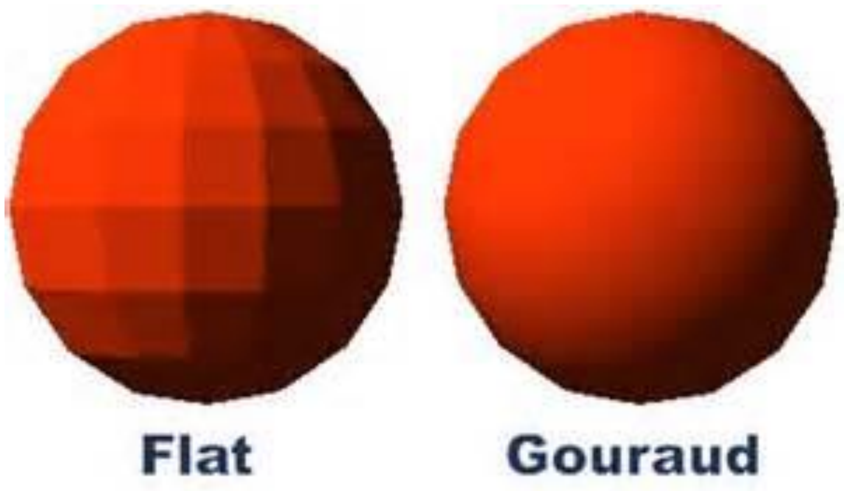
Examples of flat shading





Gourand shading

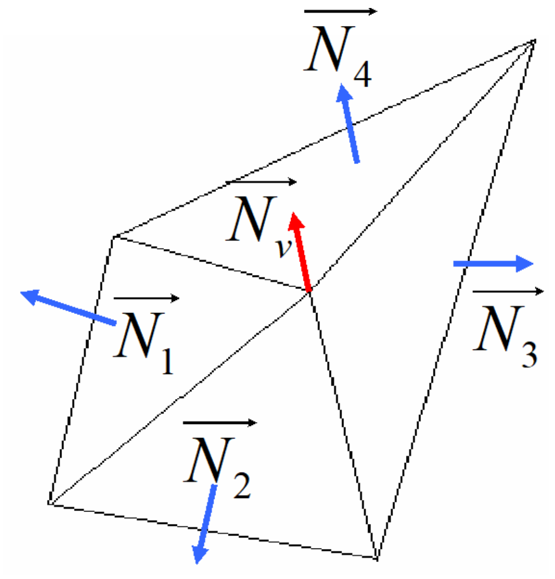
- Named after Henri Gouraud (1971)
- Produce continuous shading of surfaces represented by polygon meshes
- An interpolation method



Gourand shading

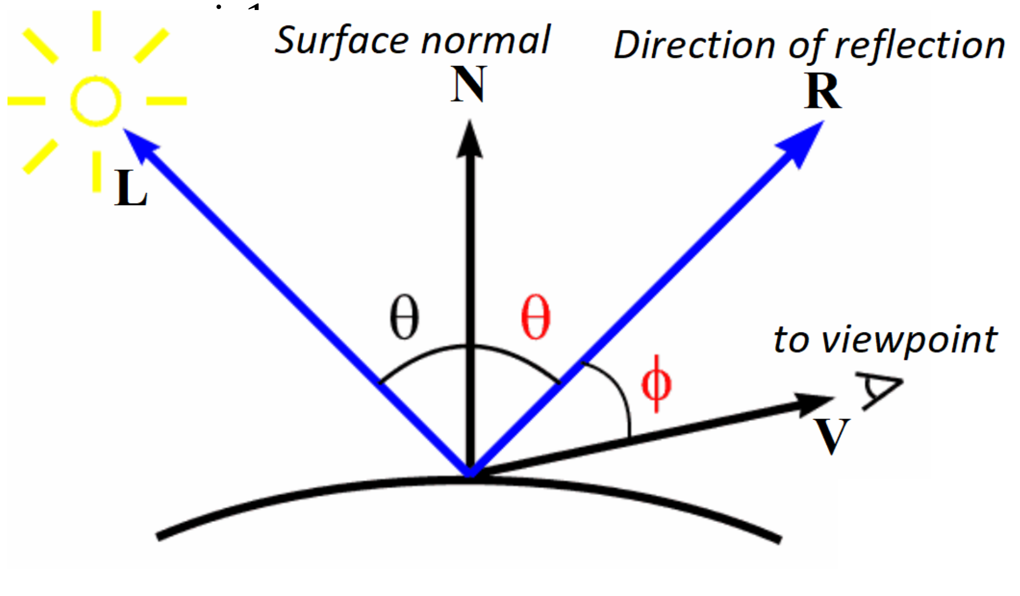
- Step 1: Normal averaging
  - estimate the normal of each vertex by averaging the surface normals of the polygons that meet at each vertex

$$\overrightarrow{N_v} = \frac{\sum_{k=1}^{n} \overrightarrow{N_k}}{\left| \sum_{k=1}^{n} \overrightarrow{N_k} \right|}$$



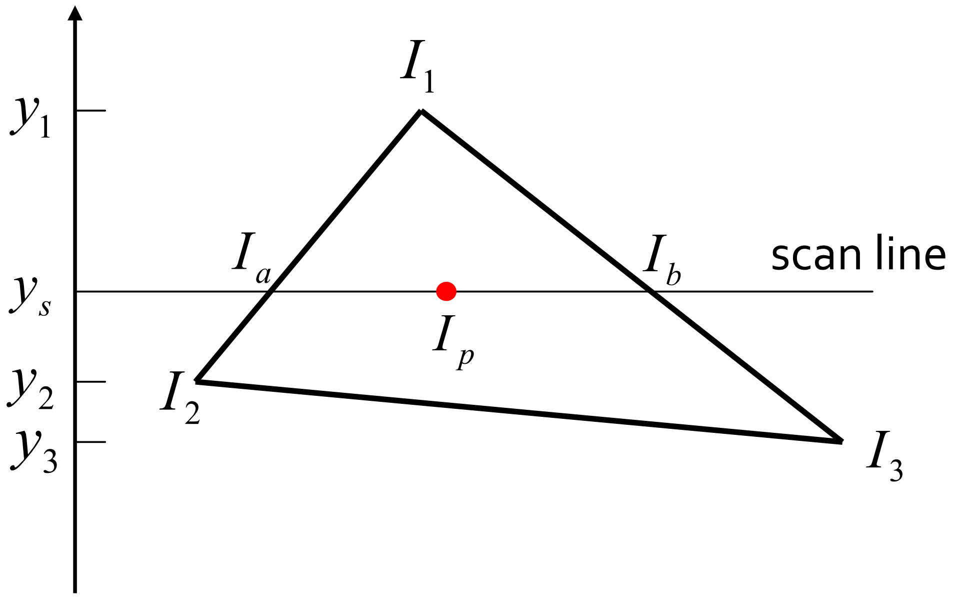
Gourand shading

- Step 2: Vertex Lighting
  - compute the color for each vertex based on a shading model
  - $I = L_a k_a + \sum_{i=1}^{m} (L_{d,i}\, k_d \max(l \bullet n\ ,0) + L_{s,i}\, k_s (h_i \bullet n)^{\alpha})$



Gourand shading

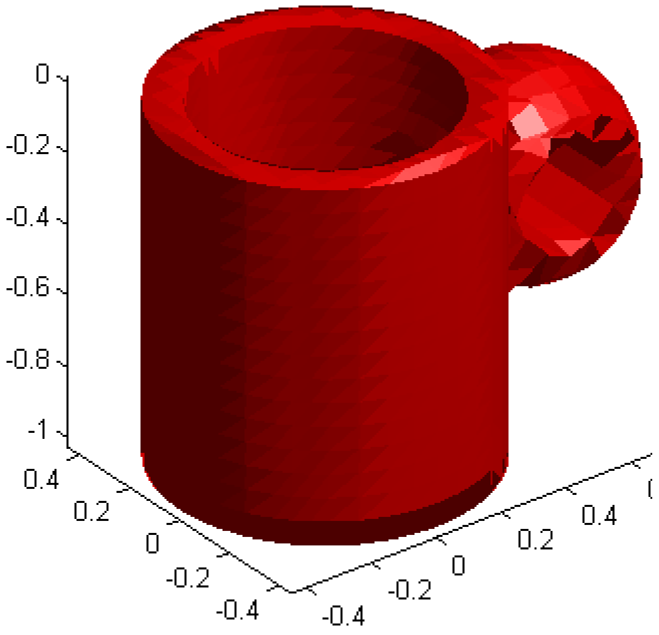
- Step3: Interpolation
  - for each **screen pixel** that represents the polygonal mesh, color is interpolated from the color values calculated at the vertices
  - Bilinear interpolation or barycentric coordinates

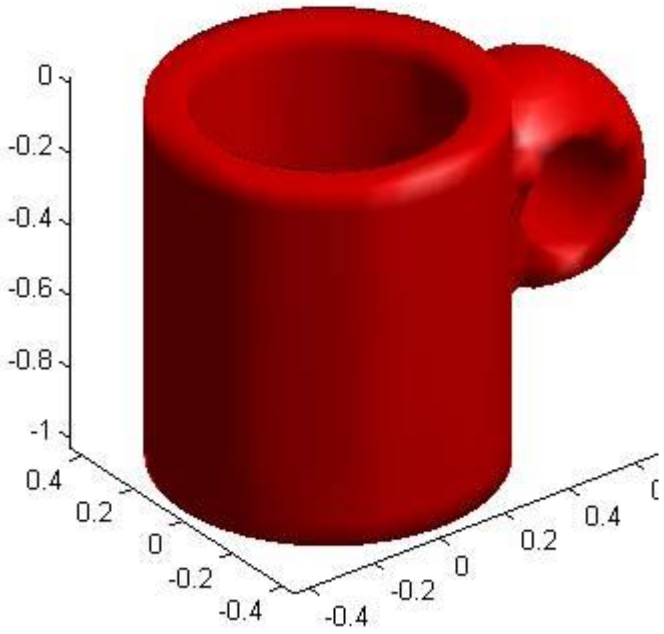


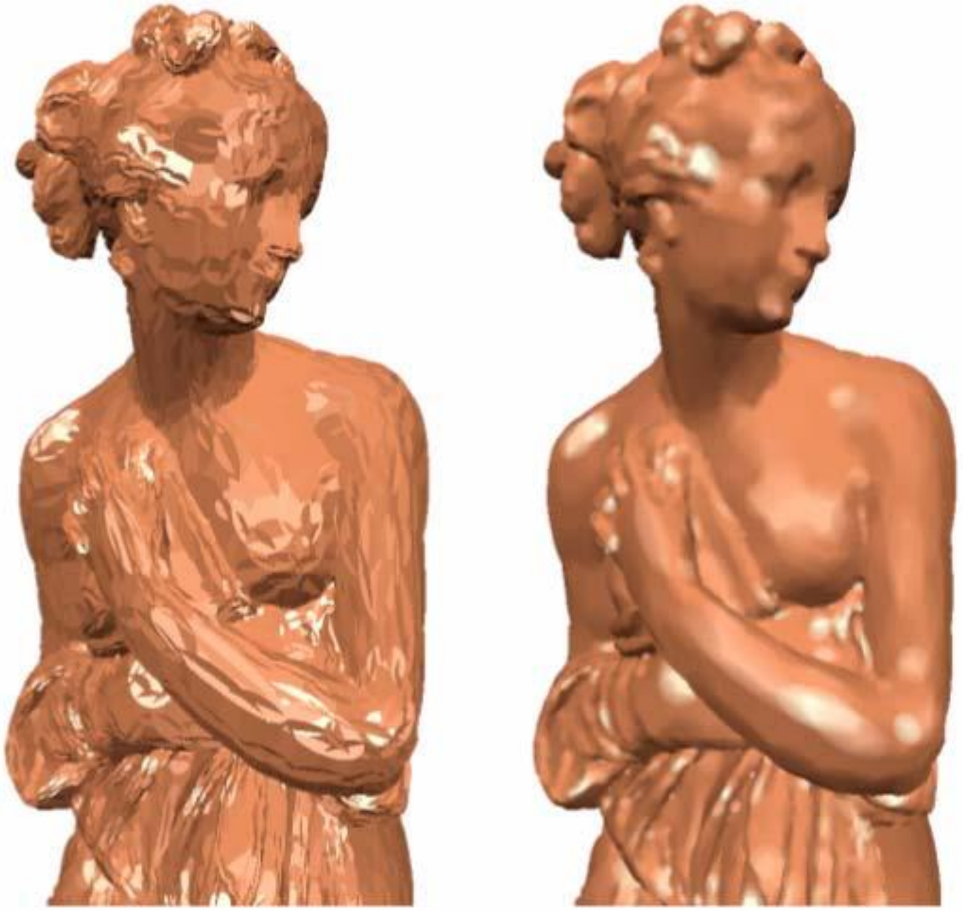
$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2}$$

$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3}$$

$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}$$

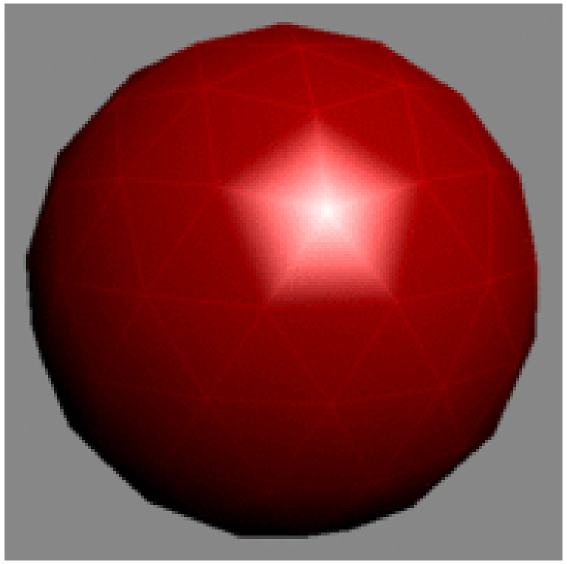

0
-0.2
-0.4
-0.6
-0.8
-1
0.4
0.2
0
-0.2
-0.4
-0.4
-0.2
0
0.2
0.4

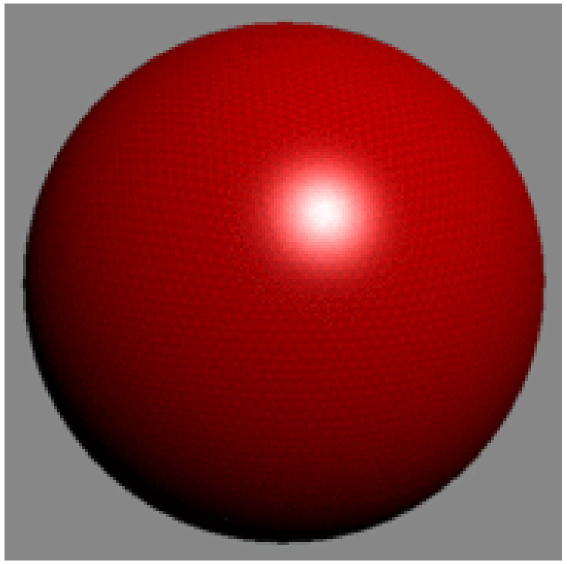

0
-0.2
-0.4
-0.6
-0.8
-1
0.4
0.2
0
-0.2
-0.4
-0.4
-0.2
0
0.2
0.4



Gourand shading

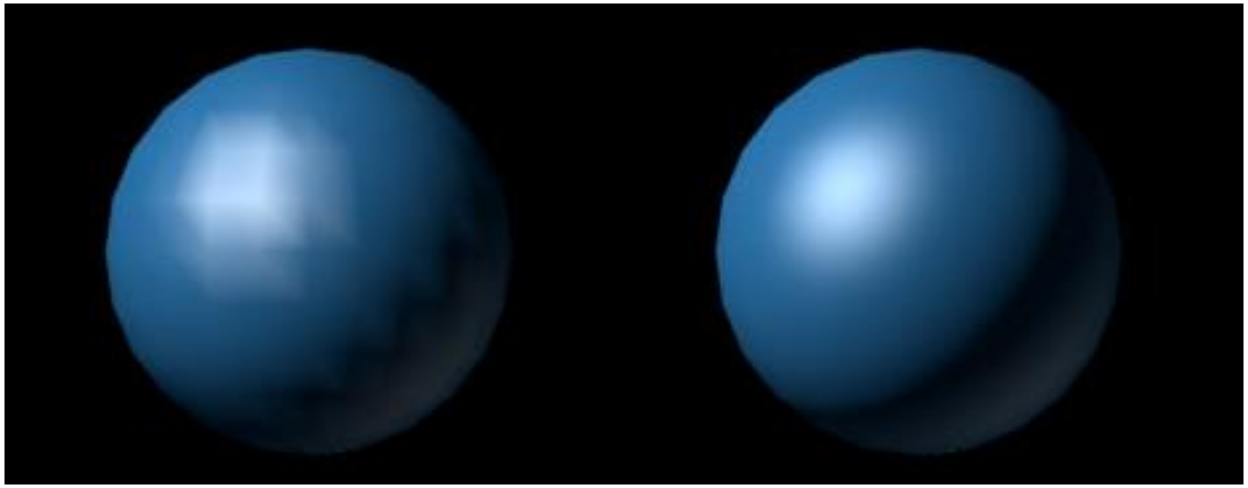
- Not good at highlights





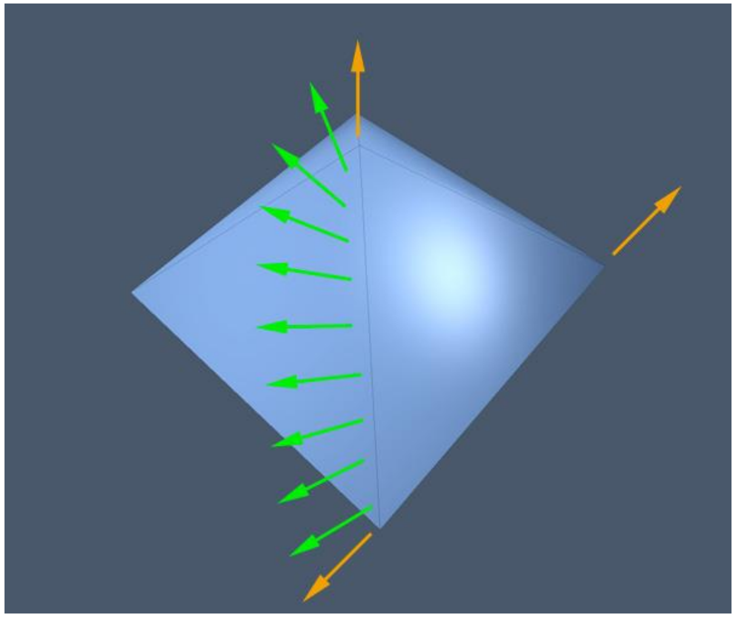
Phong shading

- Named after Bui Tuong Phong (1973)
- Also produce continuous shading of surfaces represented by polygon meshes
- Also an interpolation method
- But



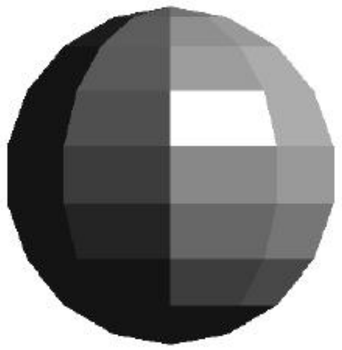
Phong shading

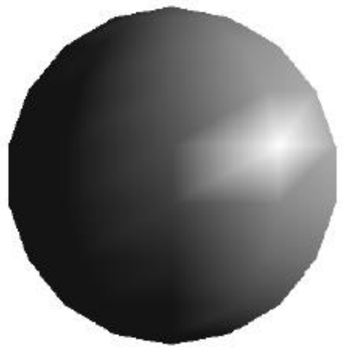
- Different from Gouraud shading, Phong shading interpolate normals rather than colors
- It interpolates surface normals across **rasterized polygons** and computes pixel colors based on the interpolated normals and a reflection model

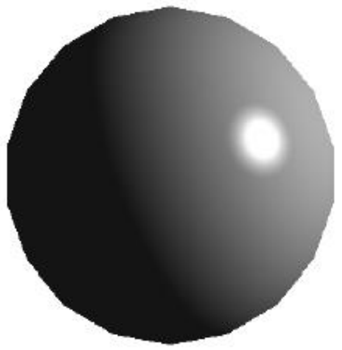


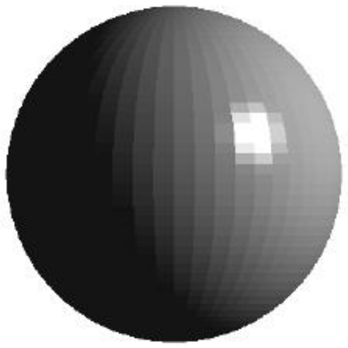
Phong shading

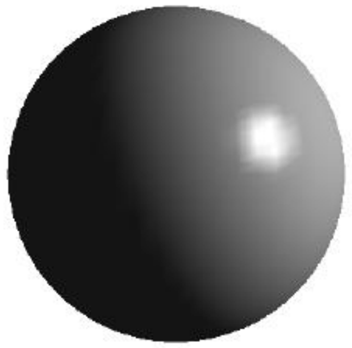
- Always captures specular highlights, but computationally expensive
- At each pixel, *n* is recomputed and normalized Then *I* is computed at each pixel (lighting model is more expensive than interpolation algorithms)
- Not available in fixed pipeline, but now can be implemented in hardware (per fragment shading), How?

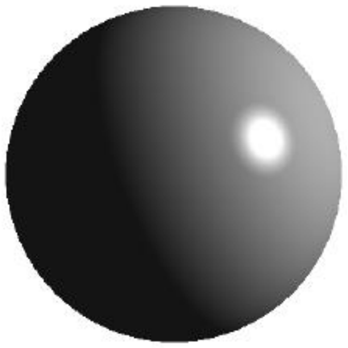

(a_1)

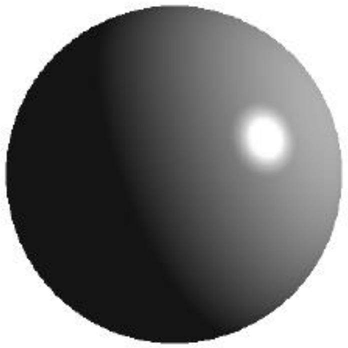

(b_1)

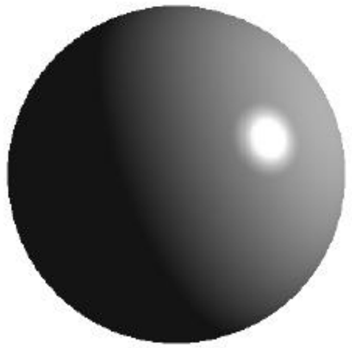

(c_1)

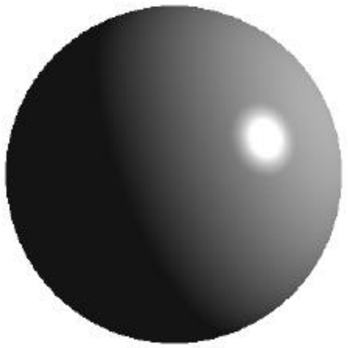

(a_2)


(b_2)

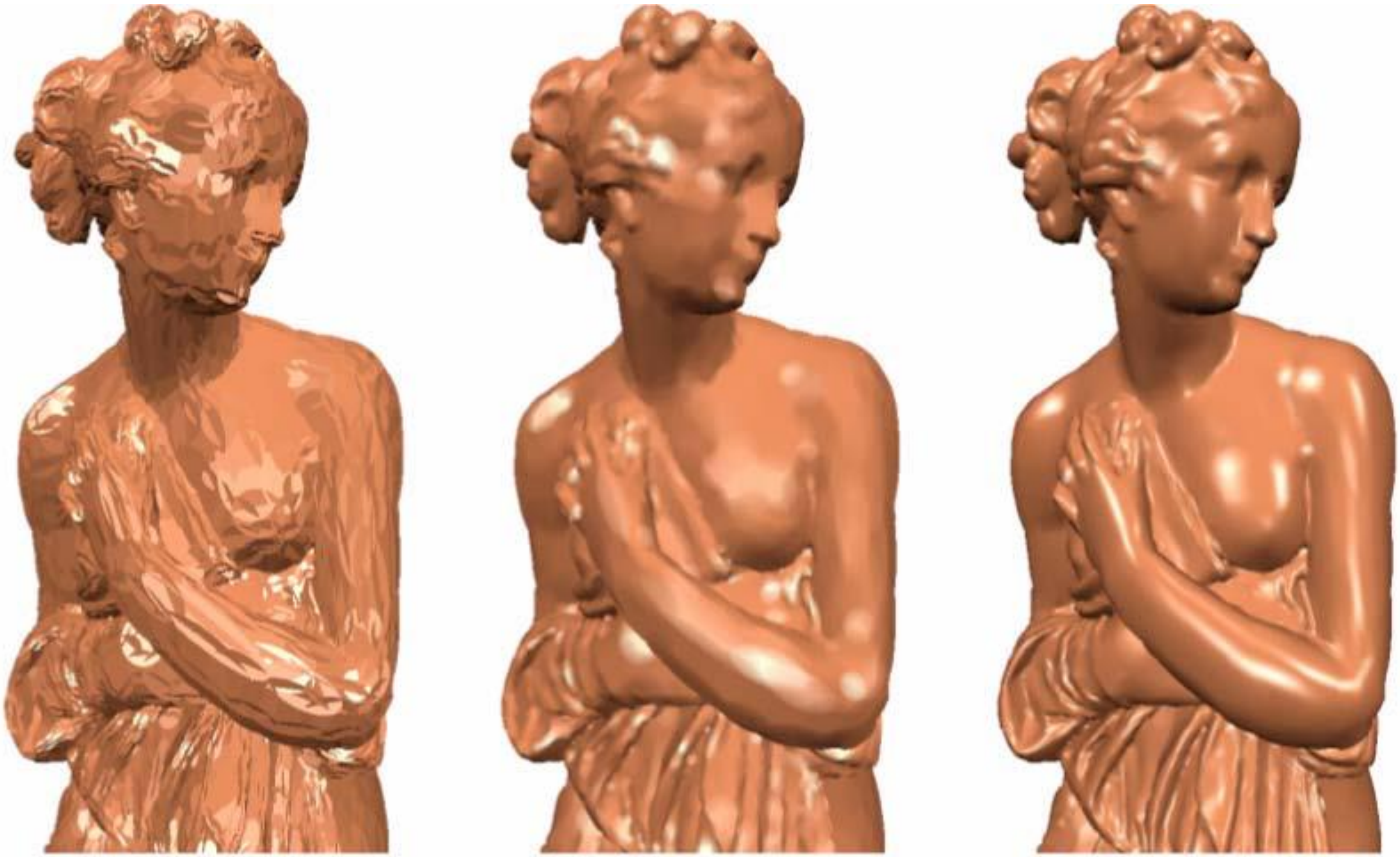

(c_2)

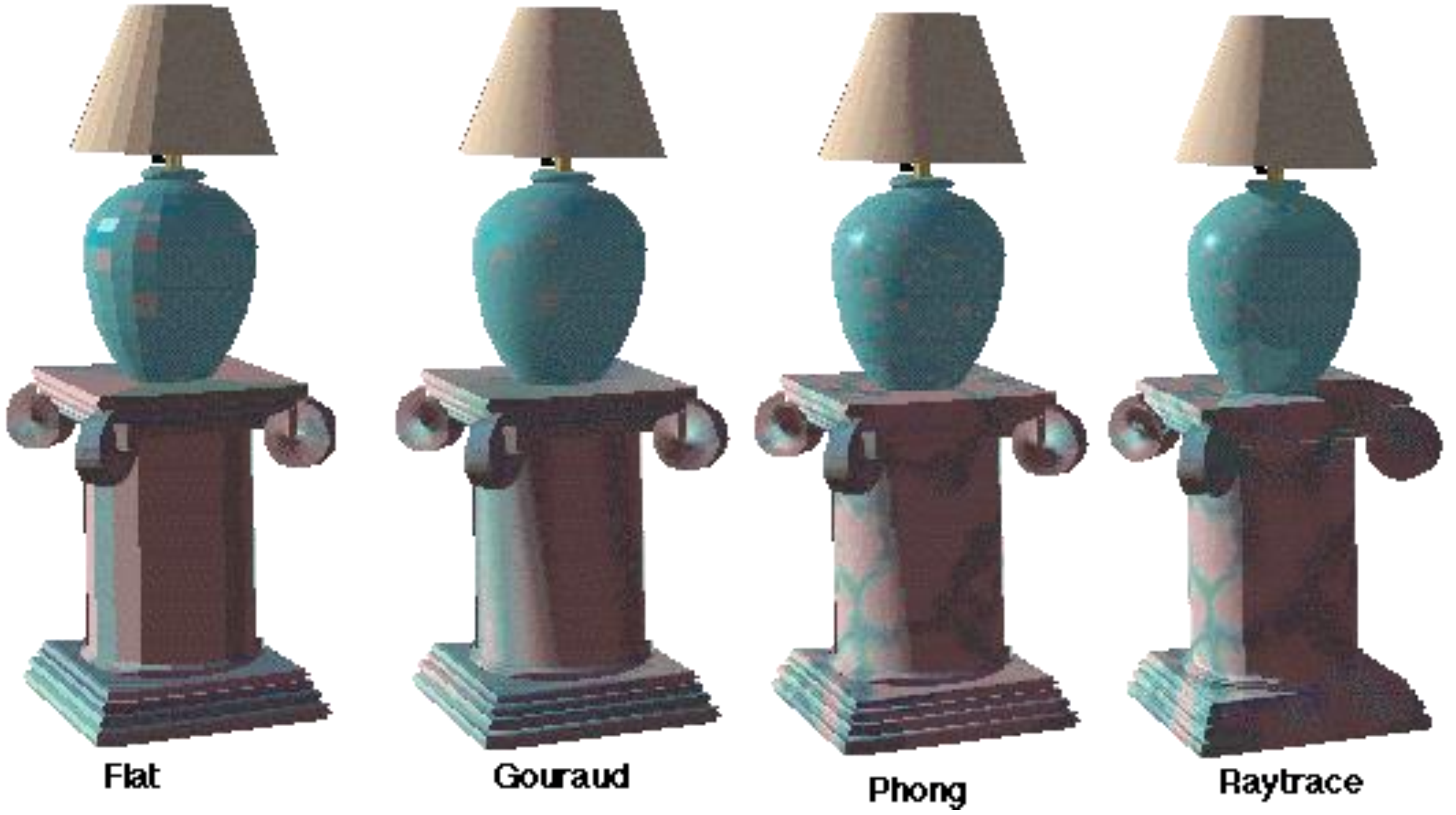

(a_3)


(b_3)


(c_3)

https://cg2010studio.files.wordpress.com

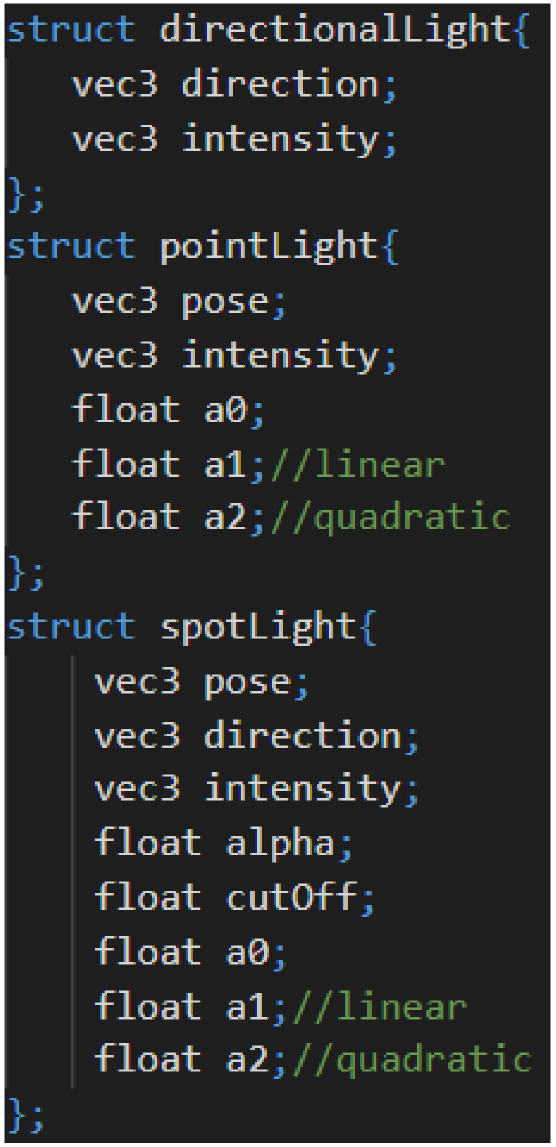




http://www.cosc.brocku.ca/Offerings/3P98

How to implement shading in OpenGL?

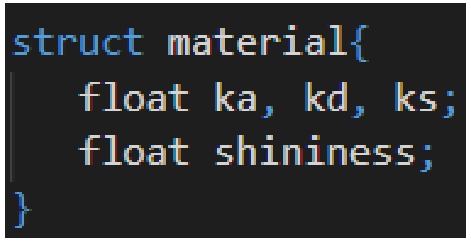
- Lights
  - Defined in vertex or fragment shader as structs
  - Intensity can be detailed defined for each lighting component
    - Ambient
    - Diffuse
    - Specular



```
struct directionalLight{
   vec3 direction;
   vec3 intensity;
};
struct pointLight{
   vec3 pose;
   vec3 intensity;
   float a0;
   float a1;//linear
   float a2;//quadratic
};
struct spotLight{
    vec3 pose;
    vec3 direction;
    vec3 intensity;
    float alpha;
    float cutOff;
    float a0;
    float a1;//linear
    float a2;//quadratic
};
```

How to implement shading in OpenGL?

- Material
  - Material can be defined as textures which is called lighting maps



```
struct material{
    float ka, kd, ks;
    float shininess;
}
```

How to implement shading in OpenGL?

- Reflection
  - Ambient

```
vec3 ambient = material.ka * Light.intensity;
```

  - Diffuse

```
vec3 diffuse = material.kd * max(dot(fragNormal, lightDir), 0.0) * Light.intensity;
```

  - Specular

```
vec3 H = normalize(viewDir + lightDir);
vec3 specular = material.ks * pow(max(dot(H, fragNormal), 0.0), material.shininess) * Light.intensity;
```

How to implement shading in OpenGL?

- Reflection
  - Attenuation

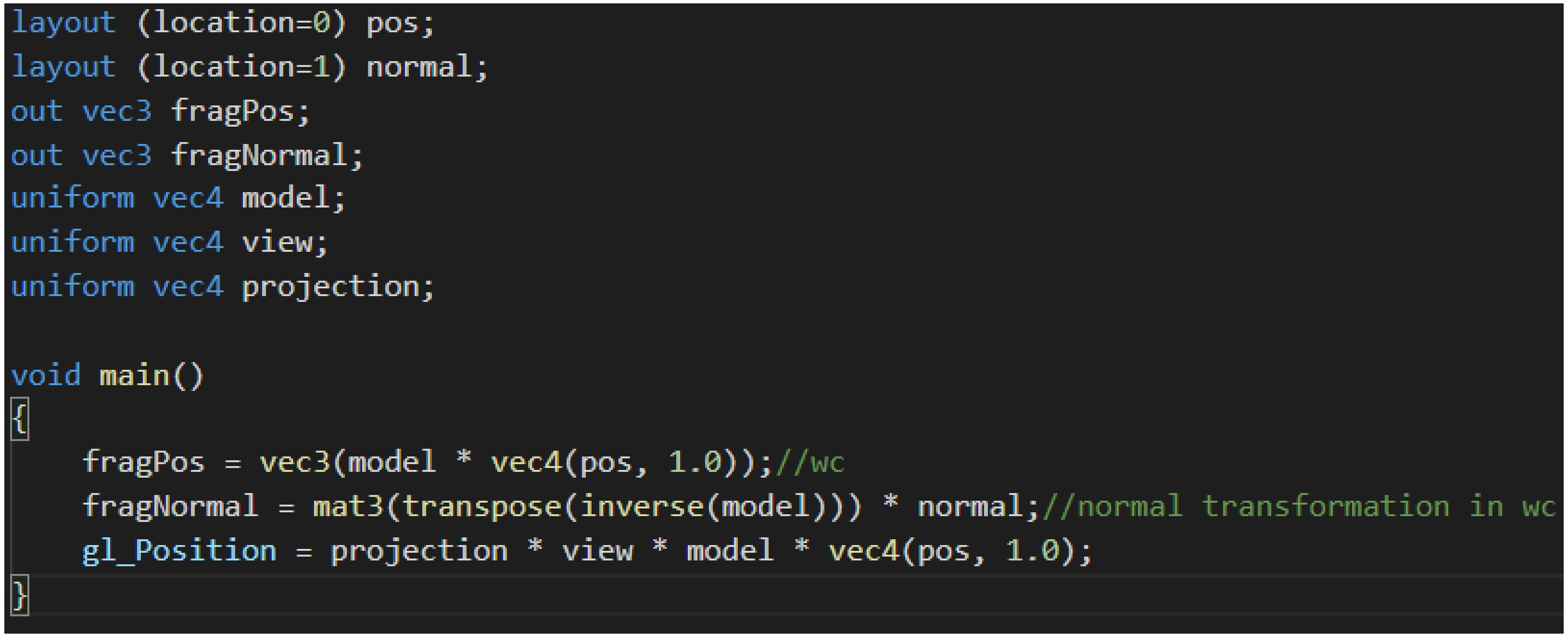
```
float attenuation = 1.0 / (a0 + a1 * distance + a2 *
distance * distance);
```

  - Attenuation

```
float angle = dot(lightDir, lightFragDir);
float angAttenuation = angle > cos(cutOff) ?
pow(angle, alpha) : 0.0;
```

How to implement shading in OpenGL?

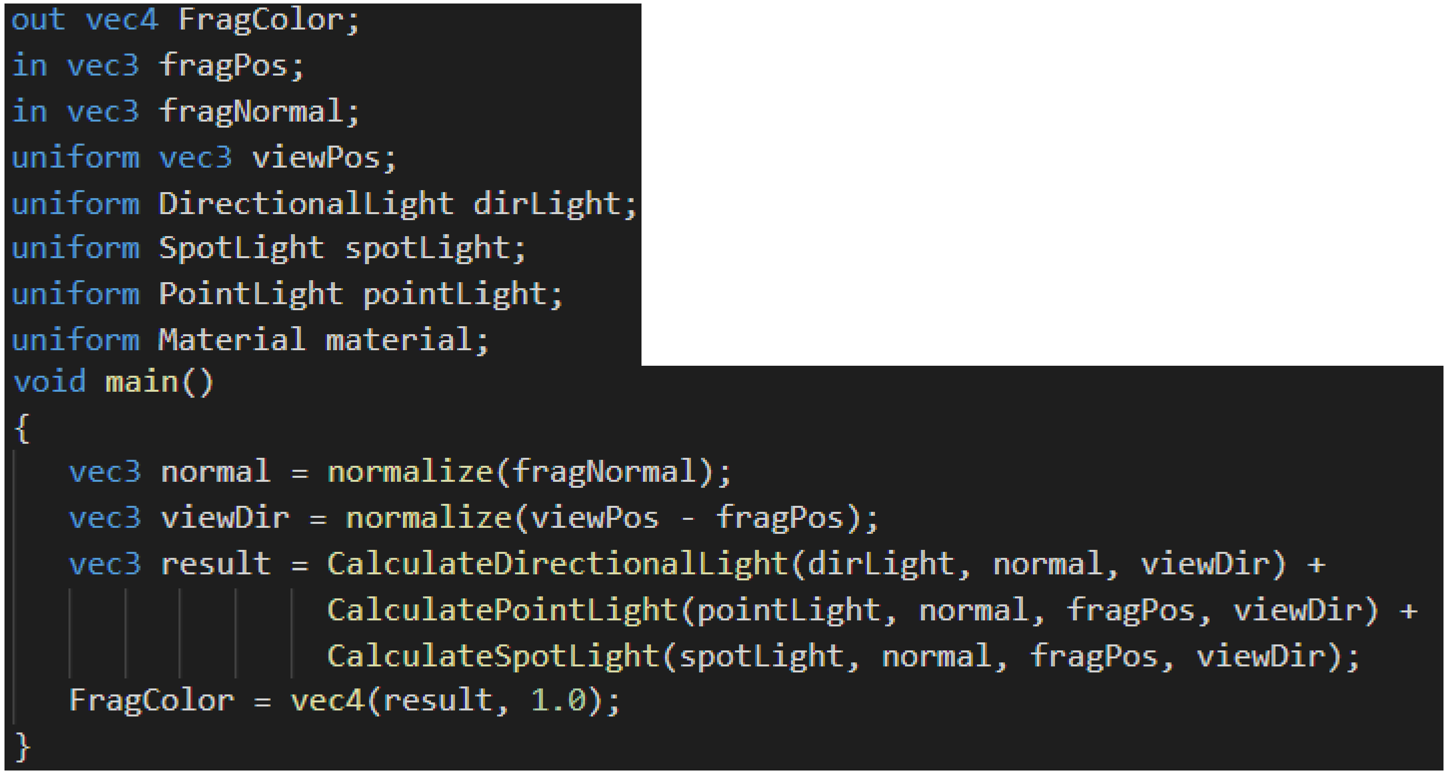
- Shading
  - Vertex shader
  - Shading in WC or in Camera space
  - Normals should be transformed as well



```
layout (location=0) pos;
layout (location=1) normal;
out vec3 fragPos;
out vec3 fragNormal;
uniform vec4 model;
uniform vec4 view;
uniform vec4 projection;

void main()
{
    fragPos = vec3(model * vec4(pos, 1.0));//wc
    fragNormal = mat3(transpose(inverse(model))) * normal;//normal transformation in wc
    gl_Position = projection * view * model * vec4(pos, 1.0);
}
```

How to implement shading in OpenGL?

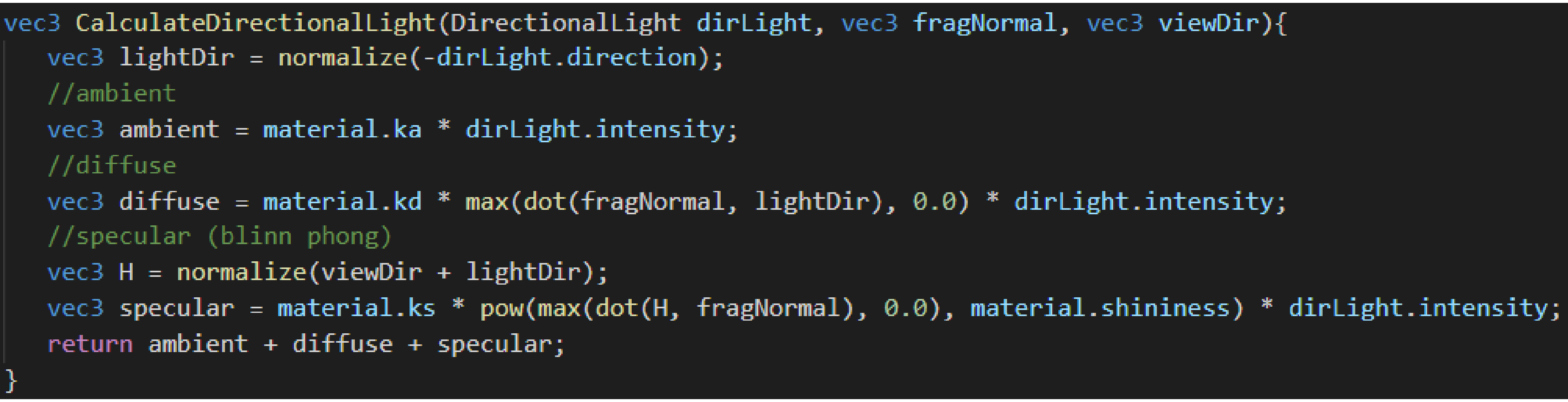
- Shading
  - Fragment shader



```
out vec4 FragColor;
in vec3 fragPos;
in vec3 fragNormal;
uniform vec3 viewPos;
uniform DirectionalLight dirLight;
uniform SpotLight spotLight;
uniform PointLight pointLight;
uniform Material material;
void main()
{
    vec3 normal = normalize(fragNormal);
    vec3 viewDir = normalize(viewPos - fragPos);
    vec3 result = CalculateDirectionalLight(dirLight, normal, viewDir) +
                  CalculatePointLight(pointLight, normal, fragPos, viewDir) +
                  CalculateSpotLight(spotLight, normal, fragPos, viewDir);
    FragColor = vec4(result, 1.0);
}
```

How to implement shading in OpenGL?

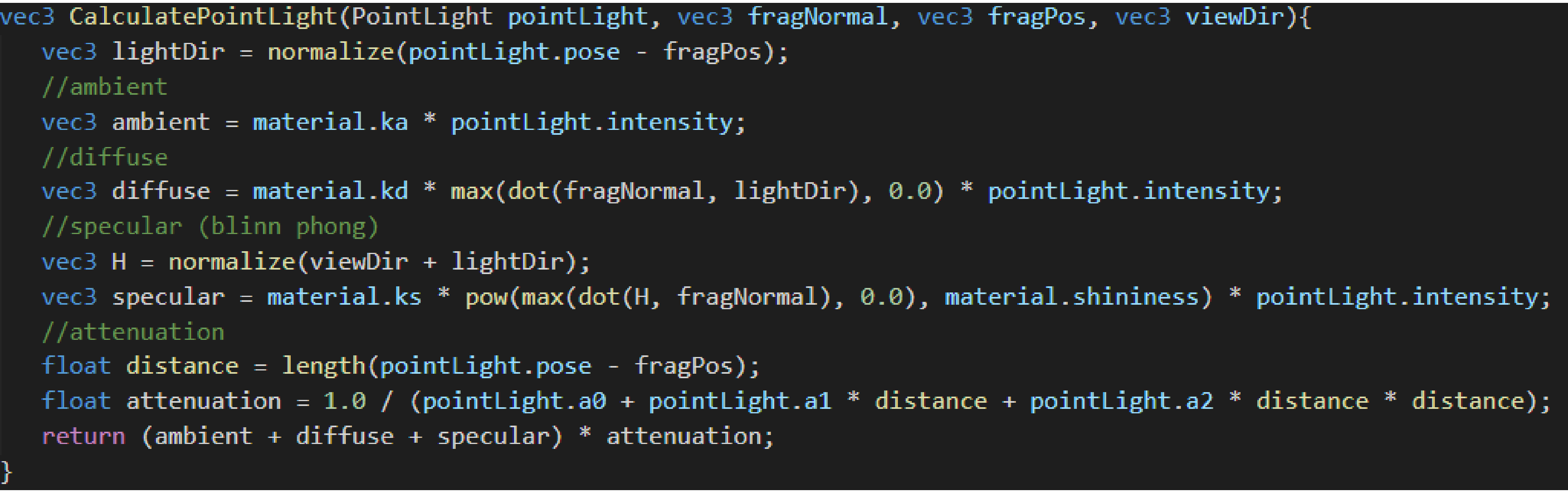
- Shading
  - Fragment shader



```
vec3 CalculateDirectionalLight(DirectionalLight dirLight, vec3 fragNormal, vec3 viewDir){
   vec3 lightDir = normalize(-dirLight.direction);
   //ambient
   vec3 ambient = material.ka * dirLight.intensity;
   //diffuse
   vec3 diffuse = material.kd * max(dot(fragNormal, lightDir), 0.0) * dirLight.intensity;
   //specular (blinn phong)
   vec3 H = normalize(viewDir + lightDir);
   vec3 specular = material.ks * pow(max(dot(H, fragNormal), 0.0), material.shininess) * dirLight.intensity;
   return ambient + diffuse + specular;
}
```

How to implement shading in OpenGL?

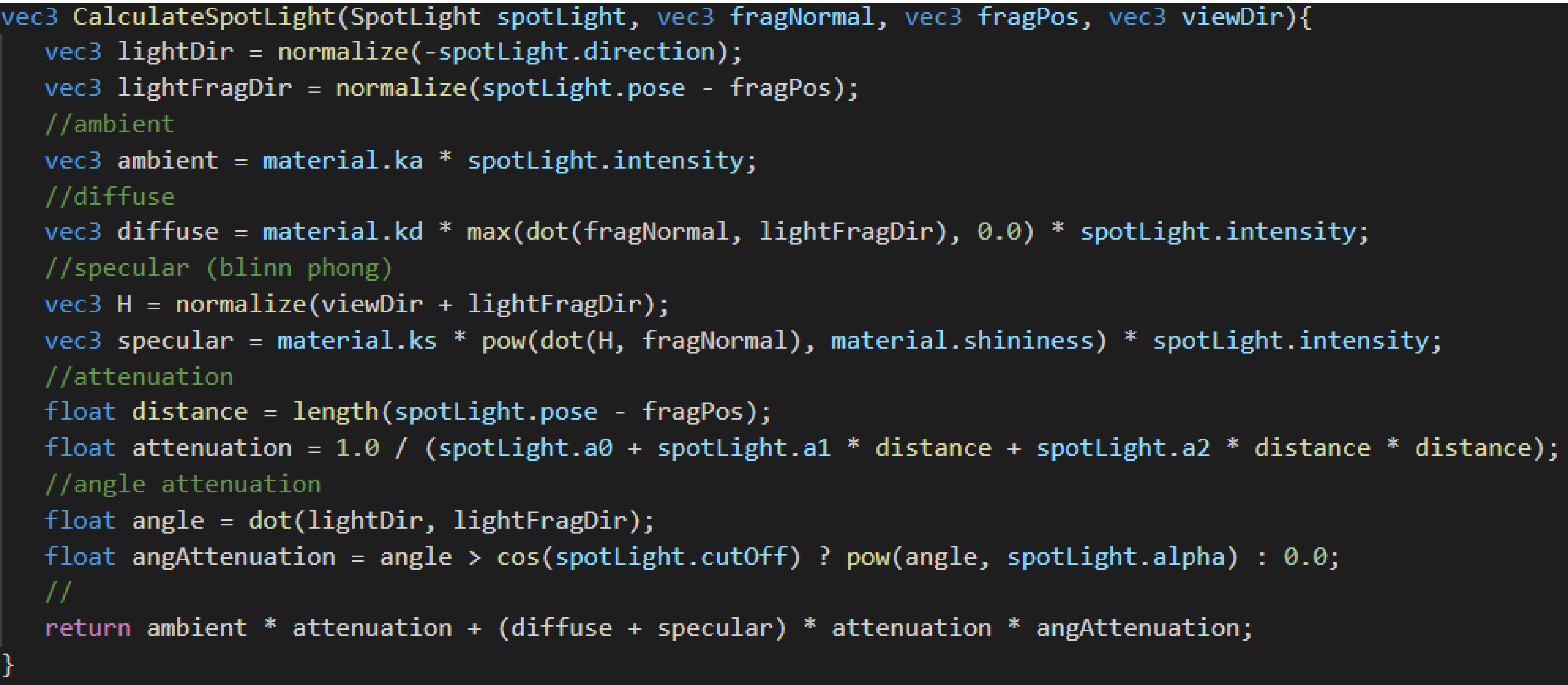
- Shading
  - Fragment shader



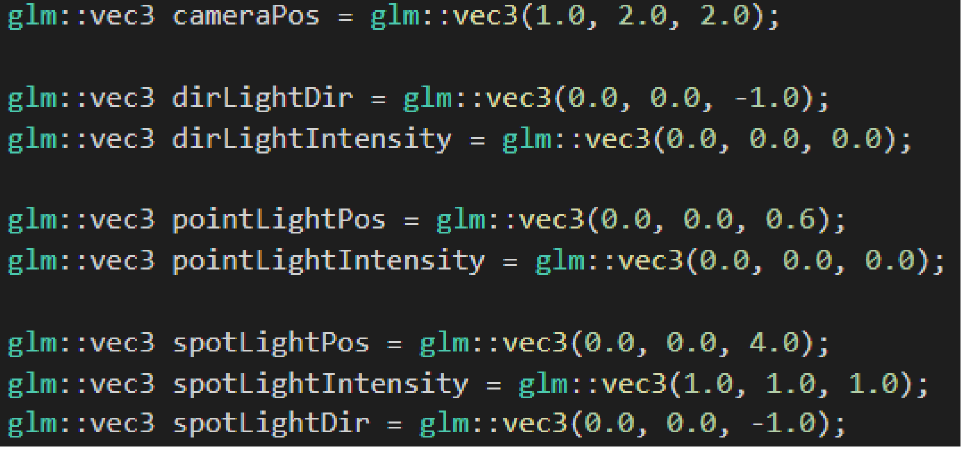
```
vec3 CalculatePointLight(PointLight pointLight, vec3 fragNormal, vec3 fragPos, vec3 viewDir){
   vec3 lightDir = normalize(pointLight.pose - fragPos);
   //ambient
   vec3 ambient = material.ka * pointLight.intensity;
   //diffuse
   vec3 diffuse = material.kd * max(dot(fragNormal, lightDir), 0.0) * pointLight.intensity;
   //specular (blinn phong)
   vec3 H = normalize(viewDir + lightDir);
   vec3 specular = material.ks * pow(max(dot(H, fragNormal), 0.0), material.shininess) * pointLight.intensity;
   //attenuation
   float distance = length(pointLight.pose - fragPos);
   float attenuation = 1.0 / (pointLight.a0 + pointLight.a1 * distance + pointLight.a2 * distance * distance);
   return (ambient + diffuse + specular) * attenuation;
}
```

How to implement shading in OpenGL?

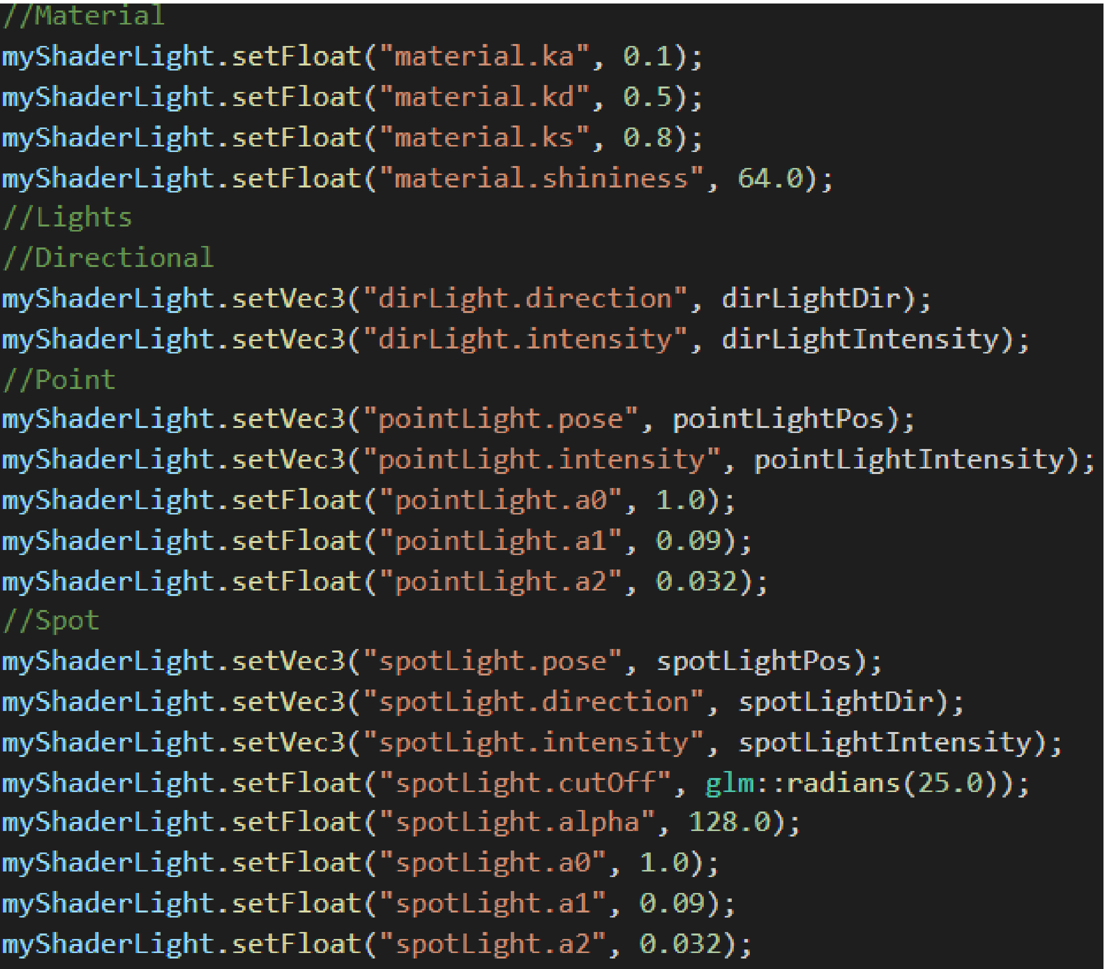
- Shading
  - Fragment shader



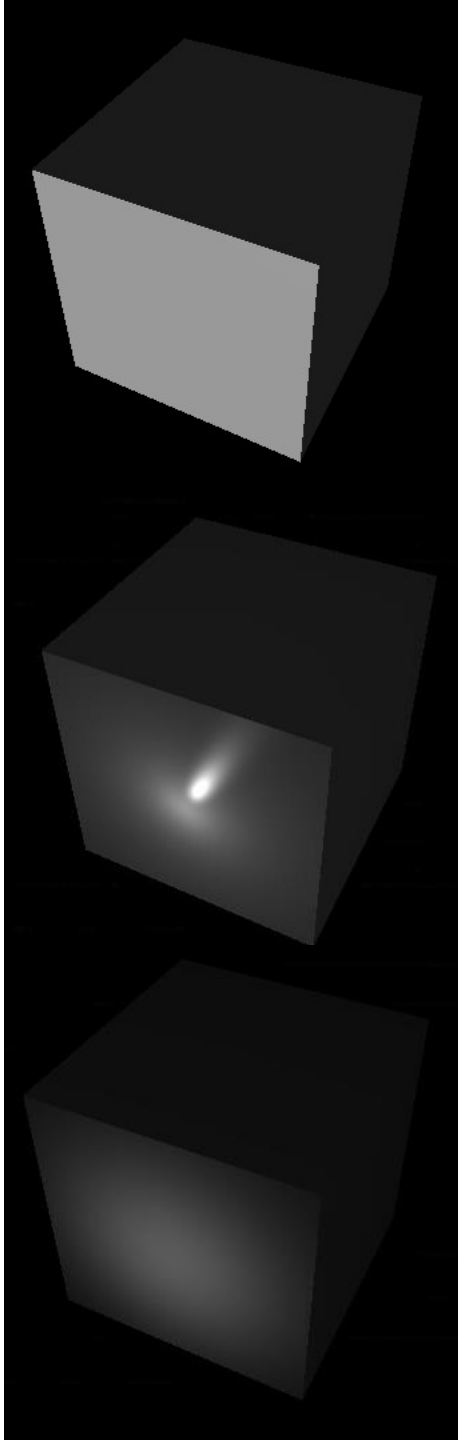
```
vec3 CalculateSpotLight(SpotLight spotLight, vec3 fragNormal, vec3 fragPos, vec3 viewDir){
   vec3 lightDir = normalize(-spotLight.direction);
   vec3 lightFragDir = normalize(spotLight.pose - fragPos);
   //ambient
   vec3 ambient = material.ka * spotLight.intensity;
   //diffuse
   vec3 diffuse = material.kd * max(dot(fragNormal, lightFragDir), 0.0) * spotLight.intensity;
   //specular (blinn phong)
   vec3 H = normalize(viewDir + lightFragDir);
   vec3 specular = material.ks * pow(dot(H, fragNormal), material.shininess) * spotLight.intensity;
   //attenuation
   float distance = length(spotLight.pose - fragPos);
   float attenuation = 1.0 / (spotLight.a0 + spotLight.a1 * distance + spotLight.a2 * distance * distance);
   //angle attenuation
   float angle = dot(lightDir, lightFragDir);
   float angAttenuation = angle > cos(spotLight.cutOff) ? pow(angle, spotLight.alpha) : 0.0;
   //
   return ambient * attenuation + (diffuse + specular) * attenuation * angAttenuation;
}
```



```
glm::vec3 cameraPos = glm::vec3(1.0, 2.0, 2.0);

glm::vec3 dirLightDir = glm::vec3(0.0, 0.0, -1.0);
glm::vec3 dirLightIntensity = glm::vec3(0.0, 0.0, 0.0);

glm::vec3 pointLightPos = glm::vec3(0.0, 0.0, 0.6);
glm::vec3 pointLightIntensity = glm::vec3(0.0, 0.0, 0.0);

glm::vec3 spotLightPos = glm::vec3(0.0, 0.0, 4.0);
glm::vec3 spotLightIntensity = glm::vec3(1.0, 1.0, 1.0);
glm::vec3 spotLightDir = glm::vec3(0.0, 0.0, -1.0);
```



```
//Material
myShaderLight.setFloat("material.ka", 0.1);
myShaderLight.setFloat("material.kd", 0.5);
myShaderLight.setFloat("material.ks", 0.8);
myShaderLight.setFloat("material.shininess", 64.0);
//Lights
//Directional
myShaderLight.setVec3("dirLight.direction", dirLightDir);
myShaderLight.setVec3("dirLight.intensity", dirLightIntensity);
//Point
myShaderLight.setVec3("pointLight.pose", pointLightPos);
myShaderLight.setVec3("pointLight.intensity", pointLightIntensity);
myShaderLight.setFloat("pointLight.a0", 1.0);
myShaderLight.setFloat("pointLight.a1", 0.09);
myShaderLight.setFloat("pointLight.a2", 0.032);
//Spot
myShaderLight.setVec3("spotLight.pose", spotLightPos);
myShaderLight.setVec3("spotLight.direction", spotLightDir);
myShaderLight.setVec3("spotLight.intensity", spotLightIntensity);
myShaderLight.setFloat("spotLight.cutOff", glm::radians(25.0));
myShaderLight.setFloat("spotLight.alpha", 128.0);
myShaderLight.setFloat("spotLight.a0", 1.0);
myShaderLight.setFloat("spotLight.a1", 0.09);
myShaderLight.setFloat("spotLight.a2", 0.032);
```



- Questions?

References

- Nancy Pollard, 15-462: Computer Graphics, CMU
- Steve Marschner, CS4620/5620 Computer Graphics, Cornell
- Cutler and Durand, MIT EECS 6.837
- Tom Thorne, COMPUTER GRAPHICS, The University of Edinburgh
- www.learningopengl.org