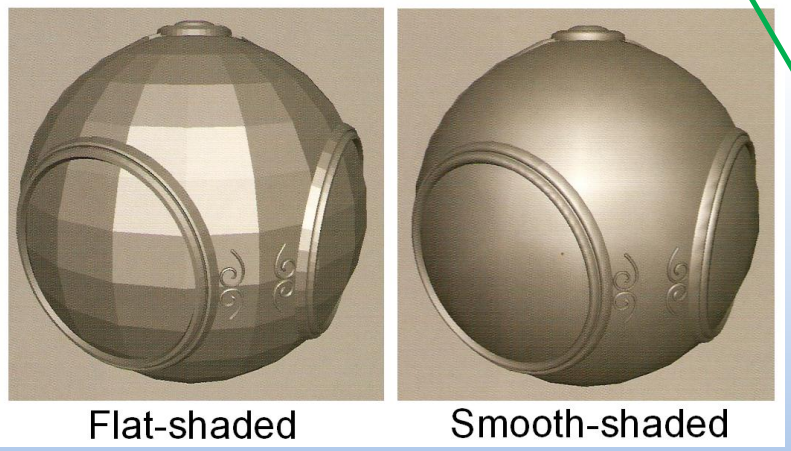
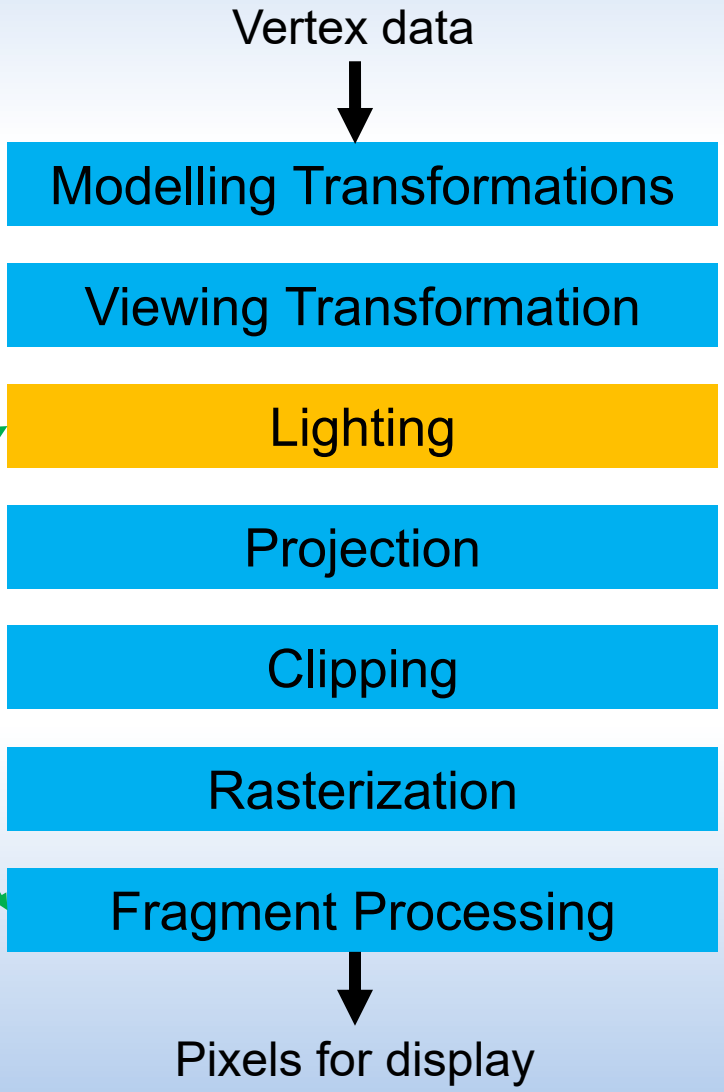
Lighting and Shading

The Computer Graphics Pipeline

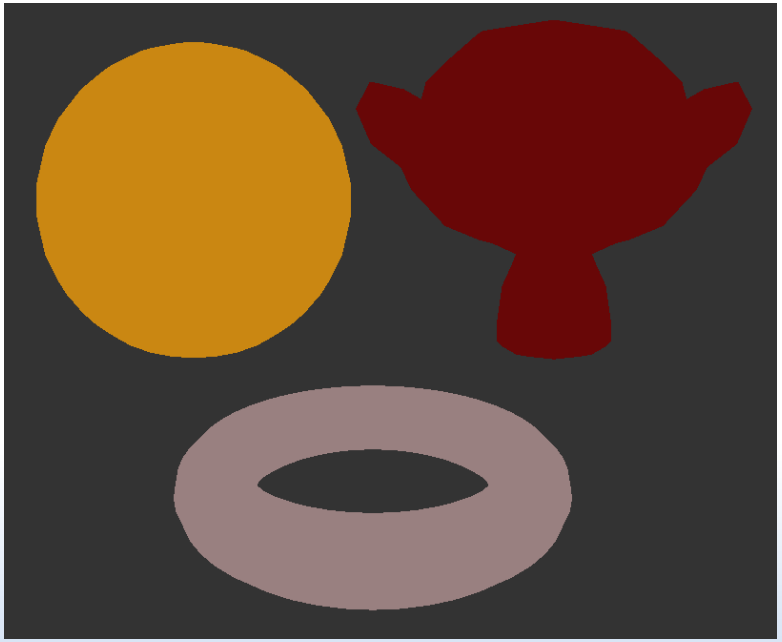
- Lighting
  - ➢ Local lighting model
    - Calculate intensity to shade 3D models
      - – Based on normals, light and material properties, etc.
      - – Per-vertex or per-pixel
  - ➢ Shading model



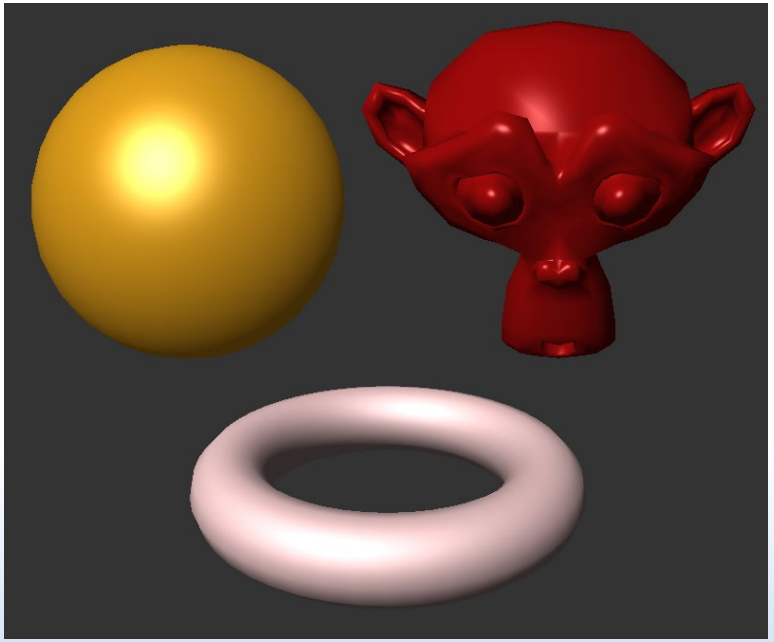


What we want…

- The same objects
  - With and without lighting and shading



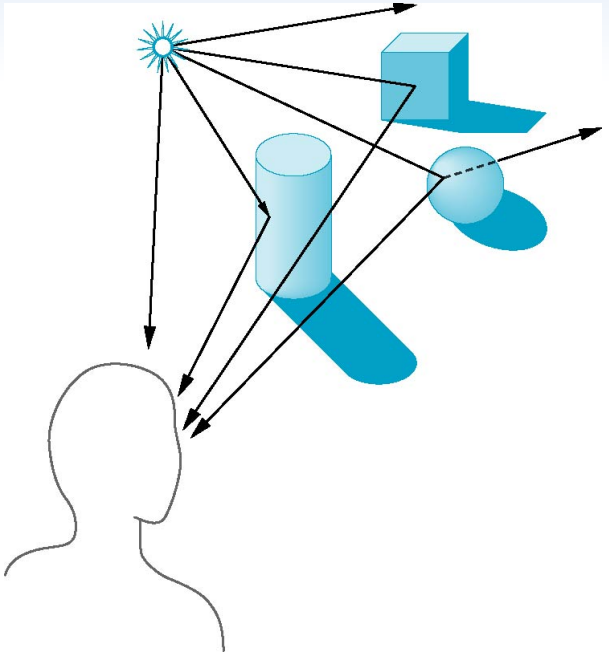
Without



With

Light and Shading

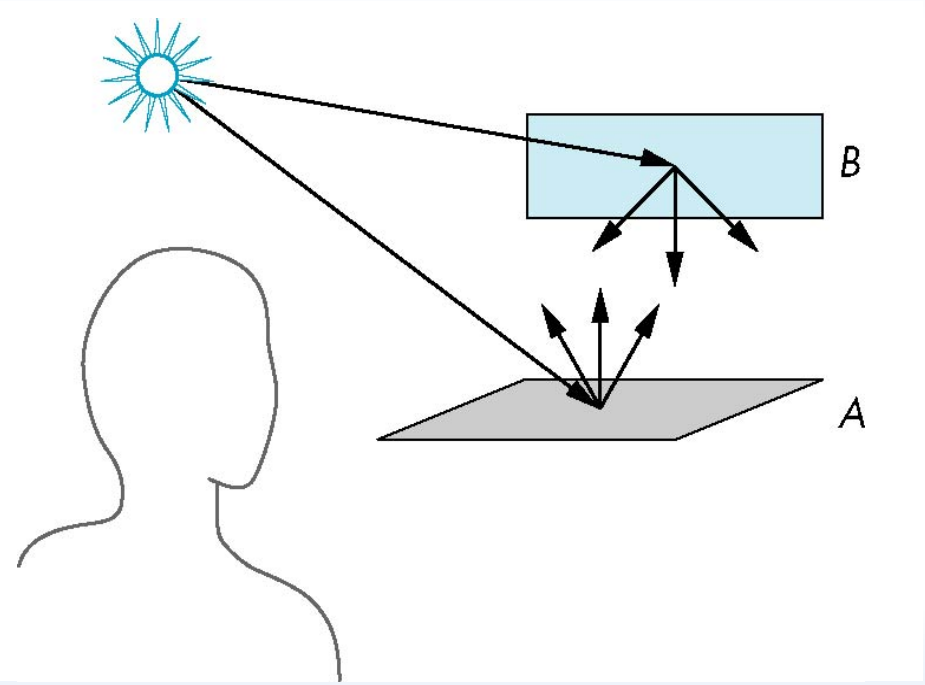
- Light
  - Emitted from sun or other sources
  - Interacts with object in the scene
    - Part is absorbed
    - Part is scattered and propagates in new directions
  - Finally, light enters eye/camera



- Colour seen by the viewer
  - Determined by multiple interactions among light sources and reflective surfaces
    - Light-material interactions cause each point to have a different colour or shade

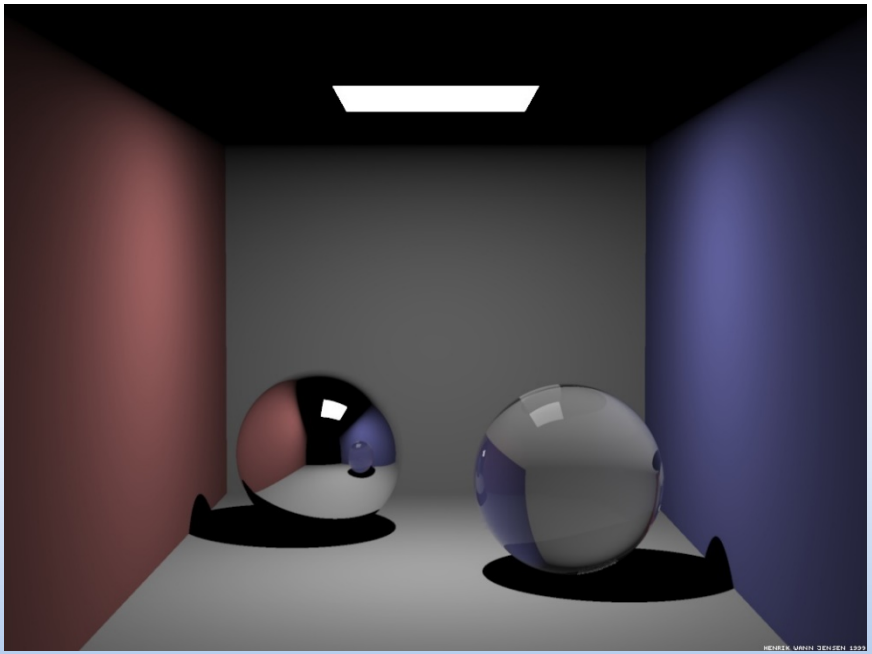
Light and Shading

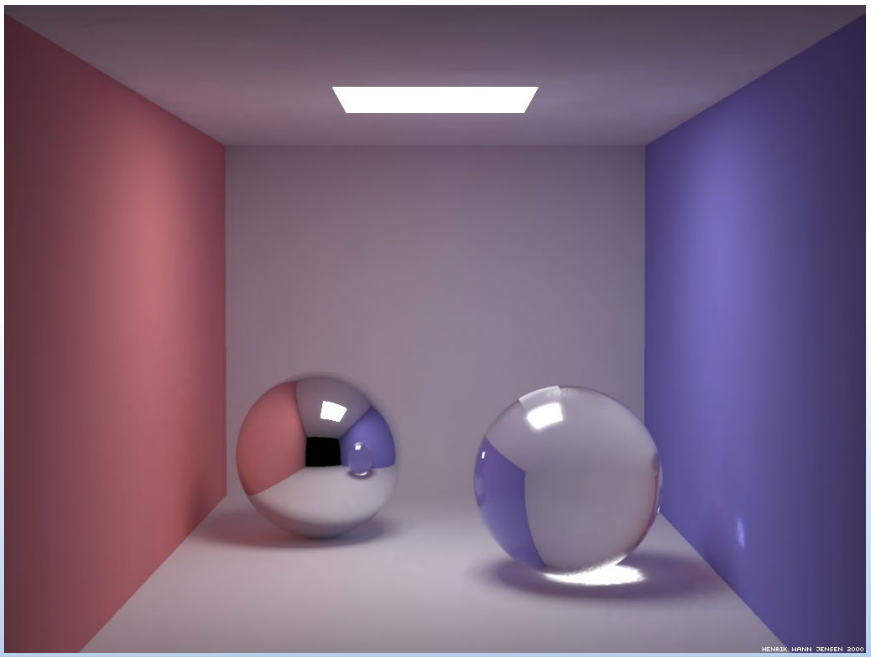
- Recursive process
  - Light strikes surface A
    - Some absorbed
    - Some scattered
  - Some of scattered light strikes surface B
    - Some absorbed
    - Some scattered
  - Some of this scattered light strikes surface A and so on



Light and Shading

- Local vs global illumination
  - Global calculation involving all objects and light sources
  - Incompatible with graphics pipeline
    - Each polygon shaded independently (local)

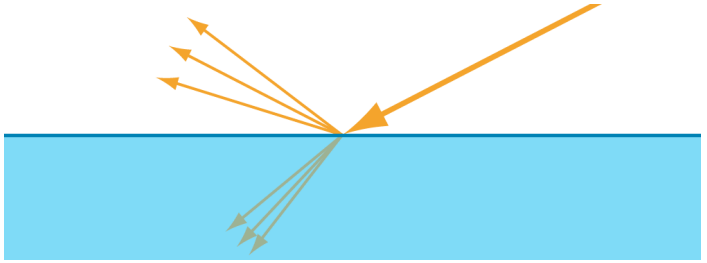




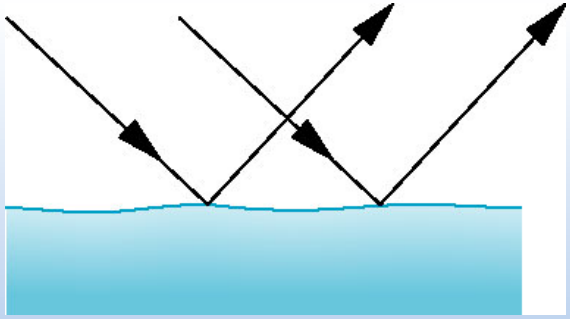
http://graphics.ucsd.edu/~henrik/images/global.html

Light and Shading

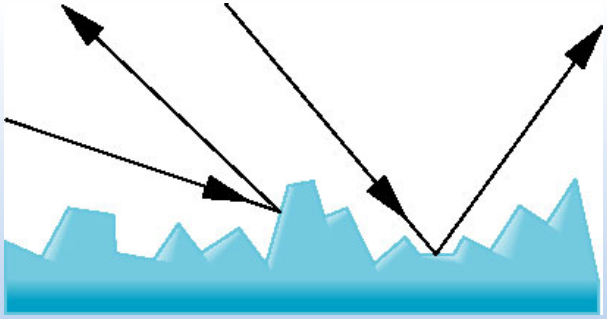
- Light-material interaction
  - ➢ Scattering
    - Away from the surface (**reflection**)
    - Into the surface (**refraction**)



  - ➢ Surface reflection
    - The **smoother** the surface, the more reflected light is concentrated in the direction a perfect mirror would reflected the light
    - A very **rough** surface scatters light in all directions



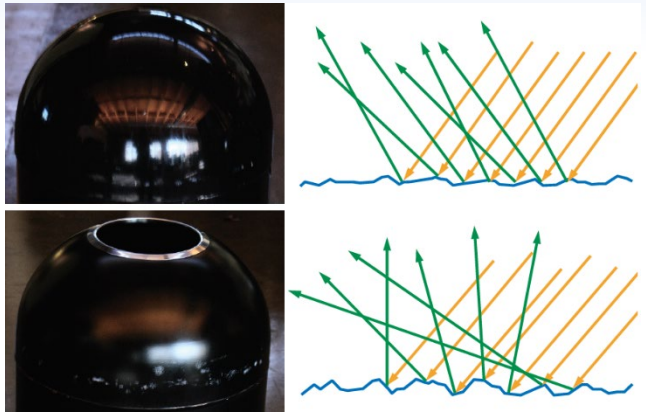
Smooth surface

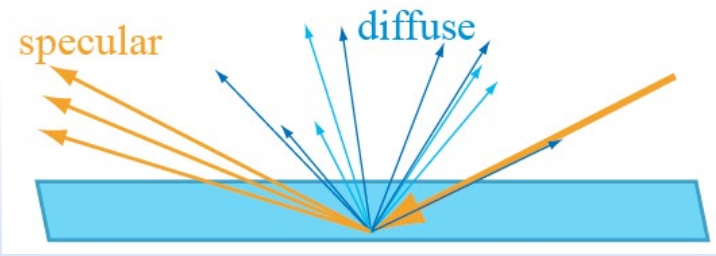


Rough surface

Light and Shading

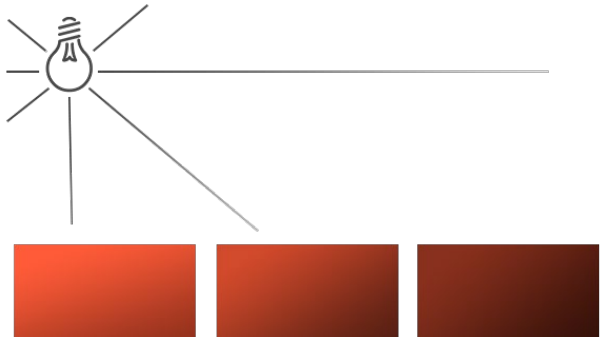
- Light-material interaction
  - **Specular** surfaces
    - Appear shiny as most light reflected in a narrow range
    - Mirrors are perfectly specular surfaces
  - **Diffuse** surfaces
    - Reflected light scattered in all directions
    - Perfectly diffuse surfaces scater light equally in all directions
      - Appear the same to all viewers
  - **Translucent** surfaces
    - Some light enters the surface and emerges from another location





Light and Shading

- Attenuation of light
  - ➢ Decrease in the intensity the further it travels from the source
    - Inversely proportional to the square of the distance between source and surface
      - i.e. 1/*distance*^{2}
        - » Decreases to rapidly
    - For greater control



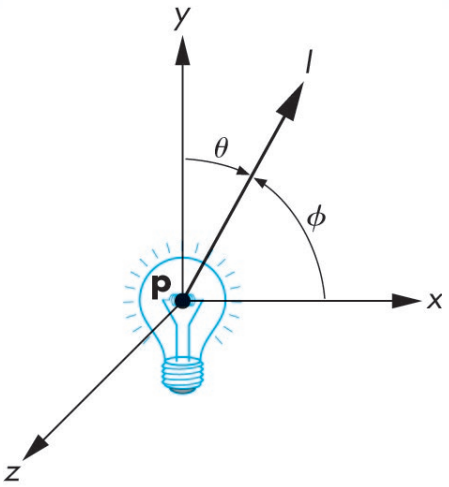
$$f_{attenuation} = \begin{cases} 1.0 & \text{If source at infinity} \\ \dfrac{1}{a + bd + cd^2} & \text{If source is local} \end{cases}$$

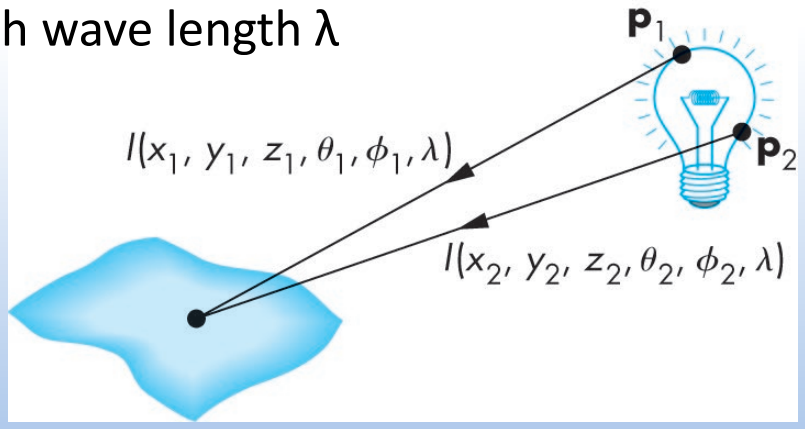
d = distance
a, *b* and *c* are attenuation constants

https://learnopengl.com/Lighting/Light-casters

Light Sources

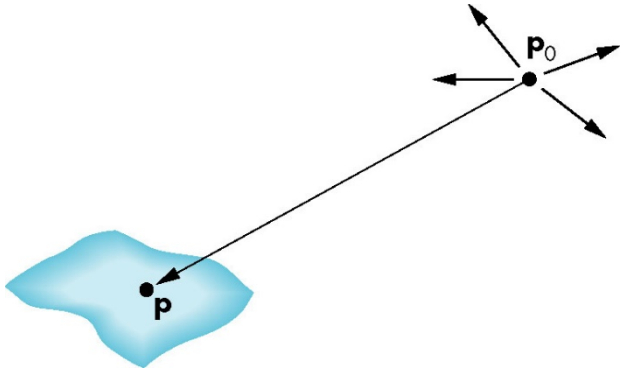
- Light can leave a surface through
  - Self-emission
  - Reflection
- A light source
  - Can be treated as an object with a surface
    - Each point (x, y, z) on the surface can emit light
    - Characterized by the direction of emission (θ, ϕ)
    - Intensity of energy emitted at each wave length λ
      - Use RGB colour model





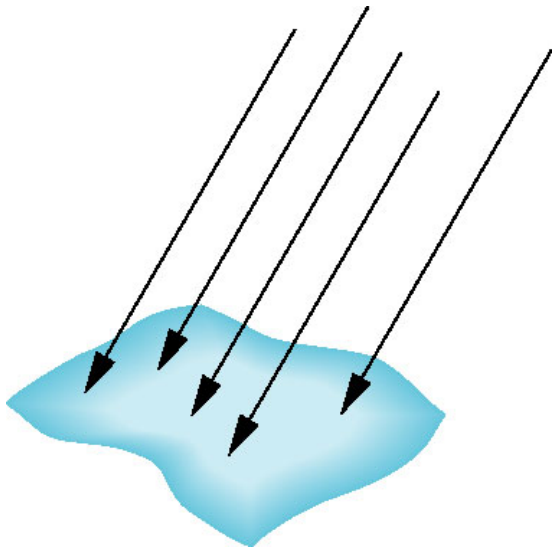
Light Sources

- Point source
  - Ideal point source emits light equally in all directions
  - Mainly used for ease of use rather than resemblance to physical reality
    - Scenes rendered using only point sources tend to have high contrasts
      - i.e. objects appear either bright or dark
  - Model with position and colour
  - Affected by attenuation



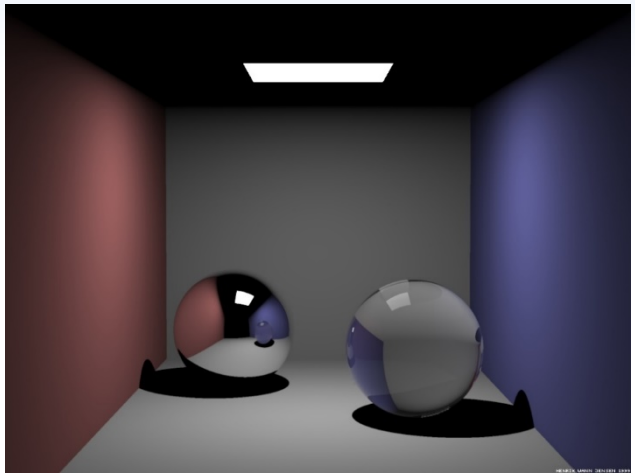
Light Sources

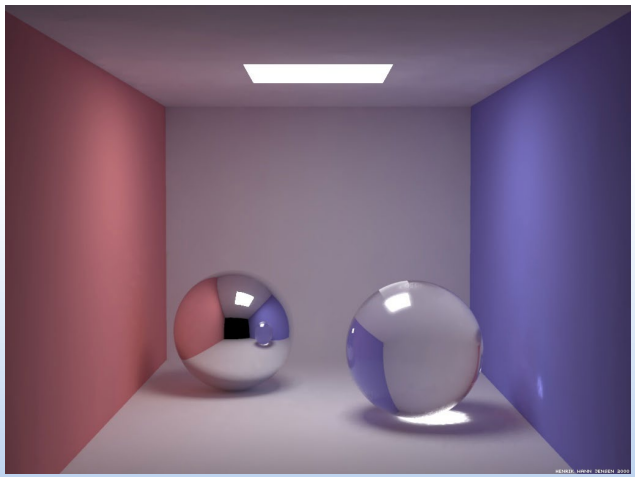
- Directional light sources
  - ➢ Also known as distant light sources
  - ➢ Can be approximated as a point source that is far from the scene
  - ➢ The difference being
    - A point source in the centre of a scene illuminates objects on all sides
    - A distant light source will only illuminate the scene from one direction (parallel)
      - Only need colour and direction (position not required)
    - Ignore attenuation



Light Sources

- Ambient light
  - Provides uniform illumination throughout the environment
  - Models contribution of many sources and reflecting surfaces
    - Local illumination
      - Calculations do not involve all objects and light sources
    - Ignore attenuation
  - Can be used to mitigate the high contrast effect from point source illuminations

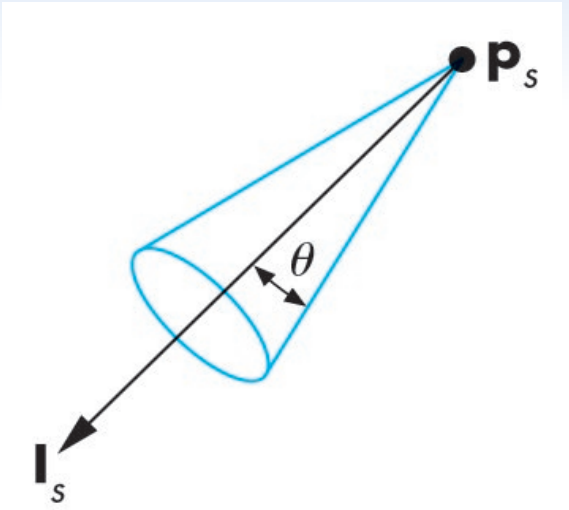


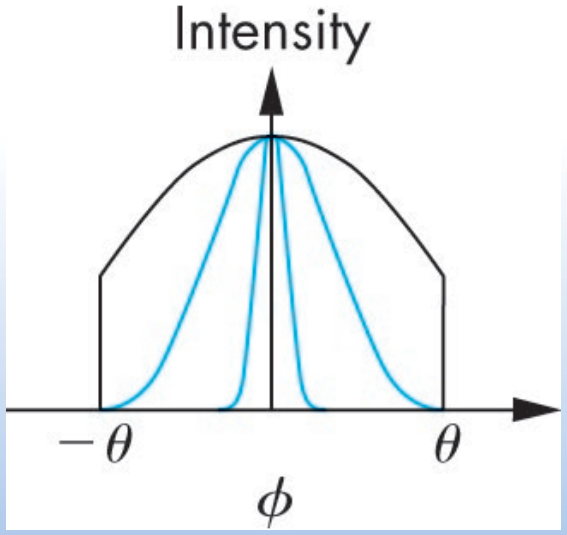


http://graphics.ucsd.edu/~henrik/images/global.html

Light Sources

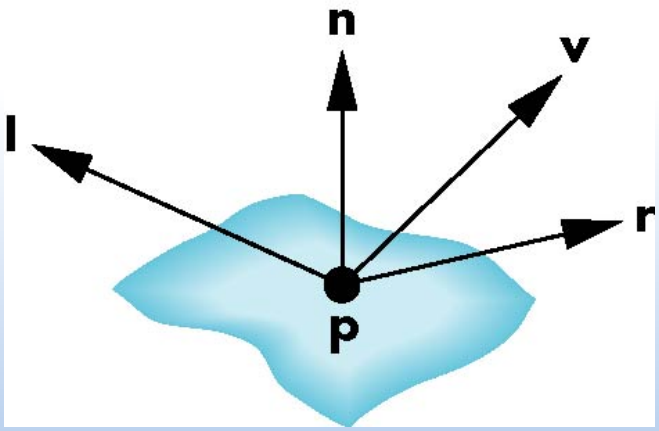
- Spotlights
  - Can construct simple spotlight from point source by restricting angle light
    - If θ = 180^o, spotlight becomes a point source
  - Characterised by distribution of light within cone
    - Most light concentrated in the centre
    - Intensity is a function of angle ϕ, usually defined by $\cos^{\alpha}\phi$
    - Exponent α determines how rapidly light intensity drops off
    - Affected by attenuation





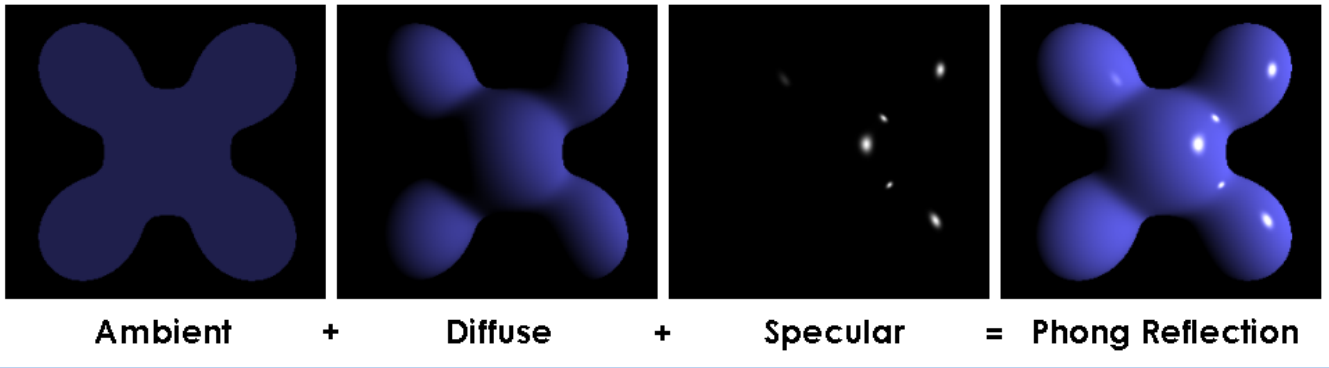
Phong Reflection Model

- Phong reflection model
  - Also known as Phong lighting model
  - Simple model that is computationally efficient
  - Three components
    - **Ambient**, **diffuse**, **specular**
  - To calculate colour for an arbitrary point, p
    - Uses four unit vectors
      - To light source, ***l***
      - To viewer, ***v***
      - Normal, ***n***
      - Perfect reflector, ***r***



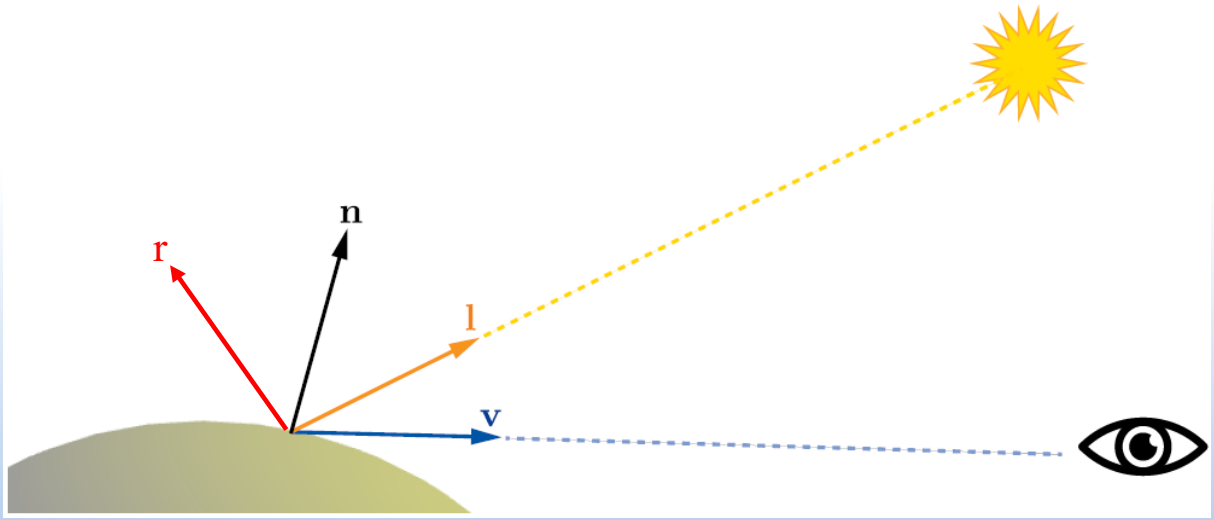
Phong Reflection Model

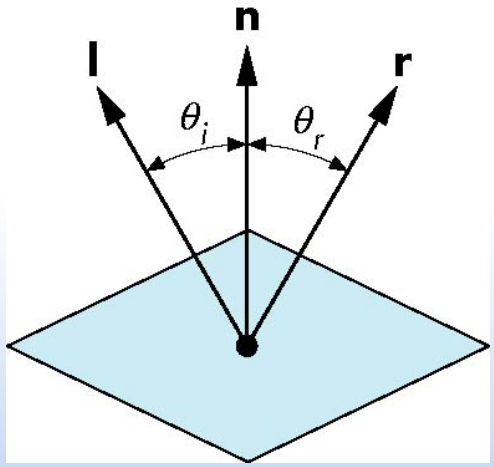
- Ambient term
  - Captures light scattered over the environment
- Diffuse reflection
  - Intensity changes gradually over surface
  - **View independent**
- Specular reflection
  - Shiny surfaces
  - **View dependent**



Phong Reflection Model

- Lighting model vectors
  - Normal vector is determined by local surface orientation
  - Perfect reflector
    - Angle of incidence = angle of reflection
    - The three vectors
      - » *l*, *n* and *r* must be co-planar, i.e. on the same plane





$$\mathbf{r} = 2\,(\mathbf{l} \cdot \mathbf{n})\,\mathbf{n} - \mathbf{l}$$

Phong Reflection Model

- Light-material interactions
  - ➢ Components for light and material
    - Often separated into three primary colours, i.e. RGB
    - Each can have different values

```
struct Light {
    vec3 La;
    vec3 Ld;
    vec3 Ls;
    ...
};
struct Material {
    vec3 Ka;
    vec3 Kd;
    vec3 Ks;
    ...
};
```

L_a intensity of ambient light
L_d intensity of diffuse light
L_s intensity of specular light

k_a ambient reflection coefficient
k_d diffuse reflection coefficient
k_s specular reflection coefficient

Phong Reflection Model

- Light-material interactions
  - ➢ Intensity of reflection
    - Sum reflection contributions of each light
    - Ambient term can be
      - Global, or for individual light sources

$$I = I_{a'} + \sum_{i \in lights} (I_d + I_s + I_a)$$

$$I = \sum_{i \in lights} (I_a + I_d + I_s)$$

I_a intensity of ambient reflection
I_d intensity of diffuse reflection
I_s intensity of specular reflection

Phong Reflection Model

- Ambient reflection
  - Result of multiple interactions between light sources and objects in environment

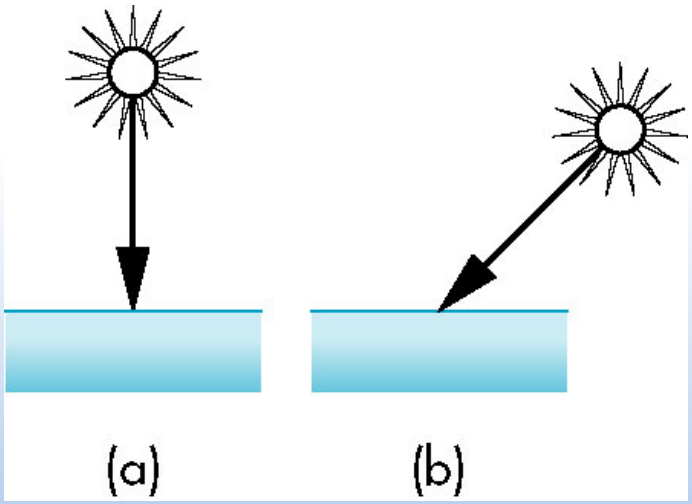
$$I_{\mathrm{a}} = k_{\mathrm{a}} L_{\mathrm{a}}$$

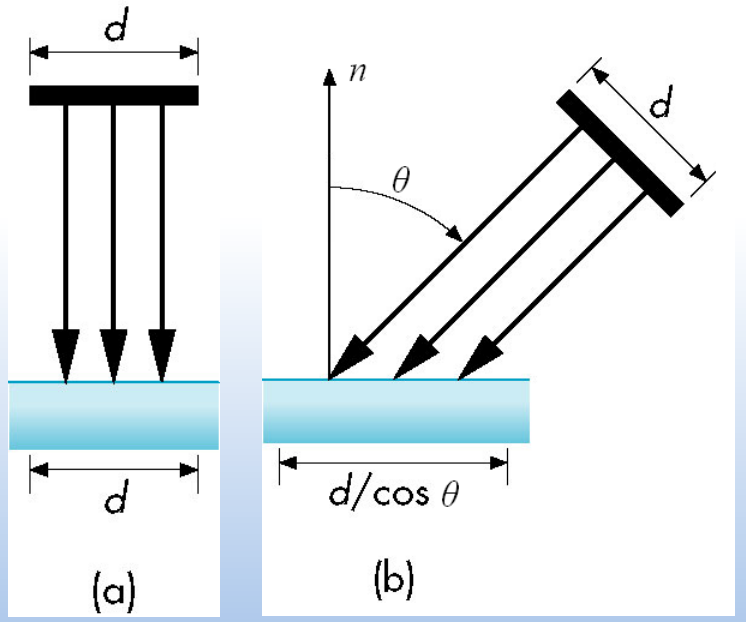
I_a intensity of ambient reflection
k_a ambient reflection coefficient
L_a intensity of ambient light

  - Intensity of ambient light, L_{a}
    - The same at every point on surface
  - Some light absorbed, some is reflected
    - Reflected amount given by ambient reflection coefficient k_{a}
      - Only a positive fraction of light is reflected, $0 \leq k_{\mathrm{a}} \leq 1$

Phong Reflection Model

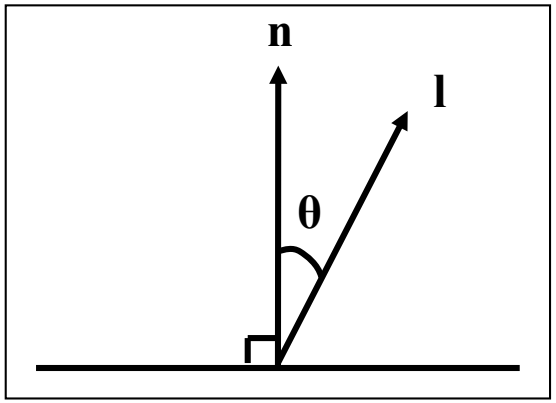
- Diffuse reflection (Lambertian reflection)
  - For perfectly diffuse reflector, diffuse light L_d
    - Scattered equally in all directions
  - Lambert's Law states
    - Intensity of reflected light is proportional to $\cos\theta$





Phong Reflection Model

- Diffuse reflection



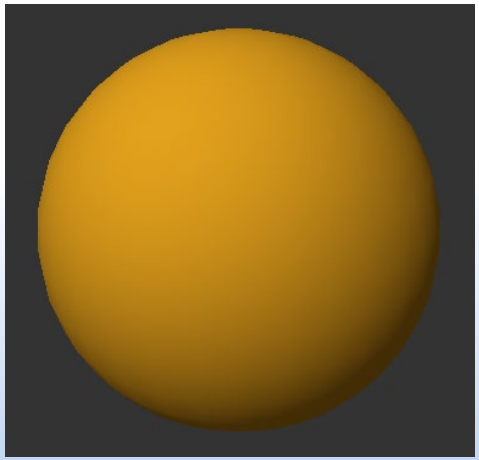
$$I_d = k_d L_d \cos\theta$$
$$= k_d L_d (\mathbf{l}.\mathbf{n})$$

I_d intensity of diffuse reflection
k_d diffuse reflection coefficient
L_d intensity of diffuse light
n unit surface normal
l unit direction to light source

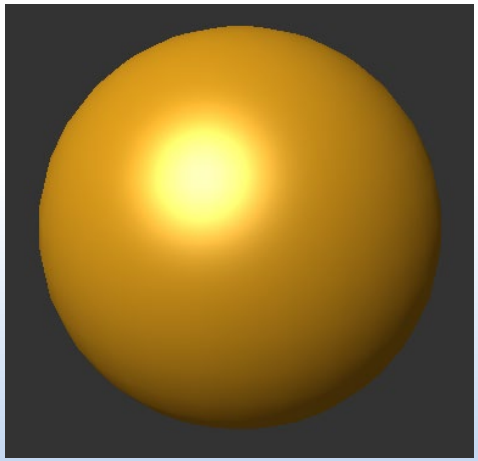
- **l**.**n**, clamped to zero
  - Negative dot product means surface is facing away from the light and should not be illuminated by that light source
- Only a positive fraction of light is reflected, $0 \le k_d \le 1$

Phong Reflection Model

- Specular reflection
  - Most surfaces neither perfectly diffuse nor perfectly specular
  - Smooth surfaces show specular highlights
    - Incoming light reflected in directions close to the direction of a perfect reflection



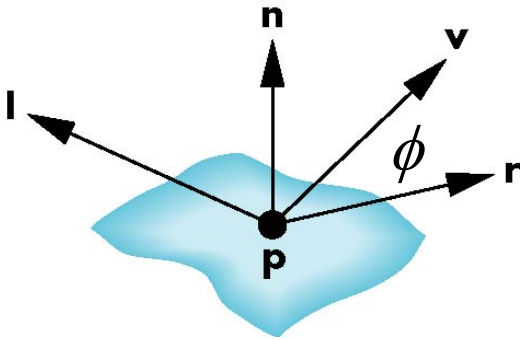
Diffuse reflection



with specular highlight

Phong Reflection Model

- Specular reflection



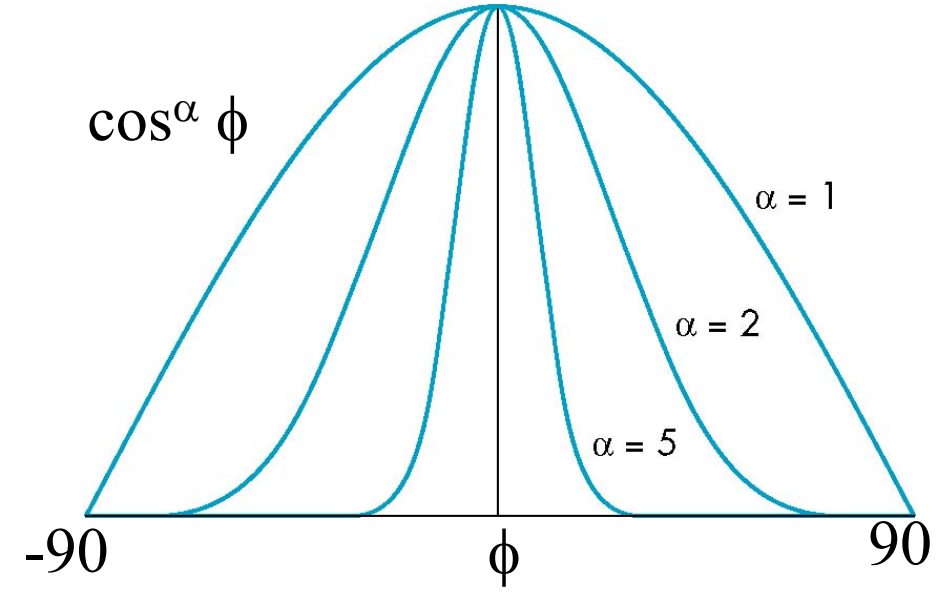
$$I_s = k_s L_s \cos^{\alpha} \Phi$$
$$= k_s L_s (\mathbf{r}.\mathbf{v})^{\alpha}$$

I_s intensity of specular reflection
k_s specular reflection coefficient
L_s intensity of specular light
α shininess exponent
r unit direction of perfect reflector
v unit direction to the viewer

- **r.v** clamped to zero
  - Negative dot product means surface facing away from the light and should not be illuminated by that light source
- Only a positive fraction of light is reflected, $0 \le k_s \le 1$
- α shininess exponent
  - Affects the concentration of specular highlight

Phong Reflection Model

- Specular reflection
  - ➢ Effect of the shininess exponent
    - Values of α between 100 and 500 correspond to metallic surfaces
    - Smaller values < 100 show broad highlights



Phong Reflection Model

- Attenuation
  - For diffuse and specular components
    - To account for decrease in intensity with distance, *d*, between the light source and surface
    - For local light source
      - E.g., point light, spot light

$$I_d = \frac{1}{a + bd + cd^2}\left[k_d L_d \max(\boldsymbol{l}.\boldsymbol{n}, 0)\right]$$

$$I_s = \frac{1}{a + bd + cd^2}\left[k_s L_s \max\left((\boldsymbol{r}.\boldsymbol{v})^{\alpha}, 0\right)\right]$$

Phong Reflection Model

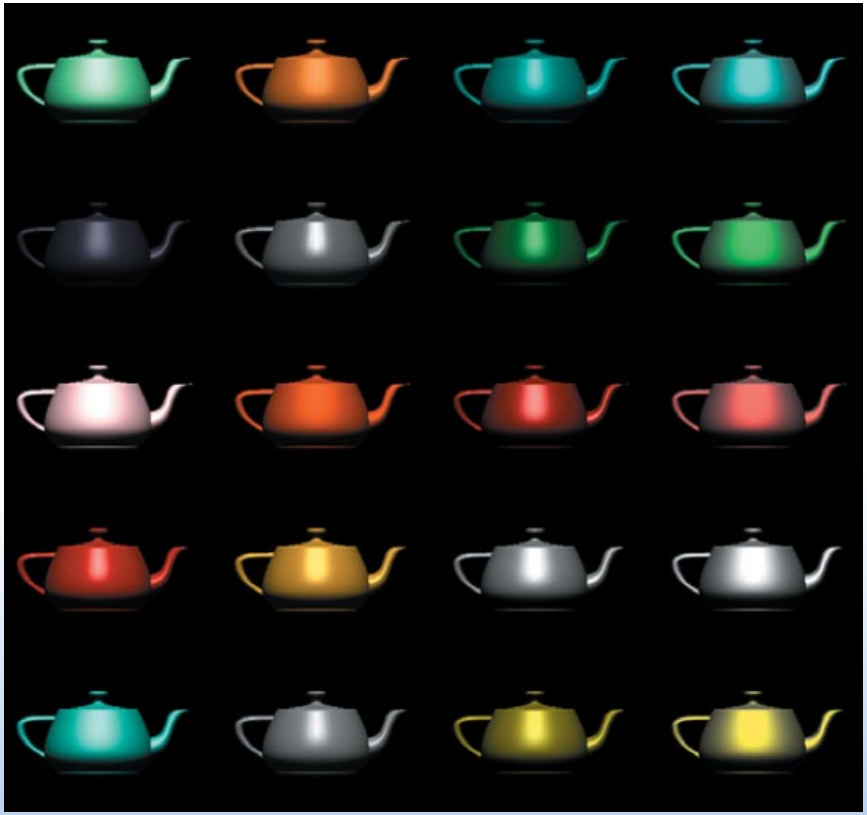
- Phong reflection model
  - For each light

$$I = \frac{1}{a + bd + cd^2}\left[k_d L_d \max(\boldsymbol{l}.\boldsymbol{n}, 0) + k_s L_s \max\left((\boldsymbol{r}.\boldsymbol{v})^{\alpha}, 0\right)\right] + k_a L_a$$

  - Multiple light sources
    - Accumulate the reflection intensities from each light
- For light sources themselves
  - Can add an emission component

Phong Reflection Model

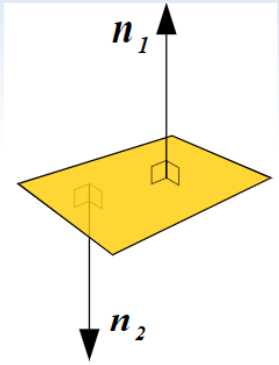
- Same lighting conditions
  - ➢ Different material properties



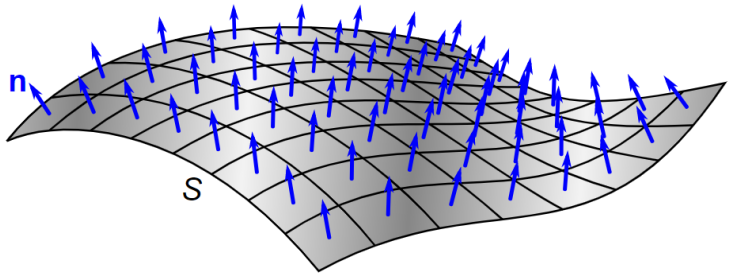
Material	GL_AMBIENT	GL_DIFFUSE	GL_SPECULAR	GL_SHININESS
Brass	0.329412	0.780392	0.992157	27.8974
	0.223529	0.568627	0.941176	
	0.027451	0.113725	0.807843	
	1.0	1.0	1.0	
Bronze	0.2125	0.714	0.393548	25.6
	0.1275	0.4284	0.271906	
	0.054	0.18144	0.166721	
	1.0	1.0	1.0	
Polished	0.25	0.4	0.774597	76.8
Bronze	0.148	0.2368	0.458561	
	0.06475	0.1036	0.200621	
	1.0	1.0	1.0	
Chrome	0.25	0.4	0.774597	76.8
	0.25	0.4	0.774597	
	0.25	0.4	0.774597	
	1.0	1.0	1.0	
Copper	0.19125	0.7038	0.256777	12.8
	0.0735	0.27048	0.137622	
	0.0225	0.0828	0.086014	
	1.0	1.0	1.0	

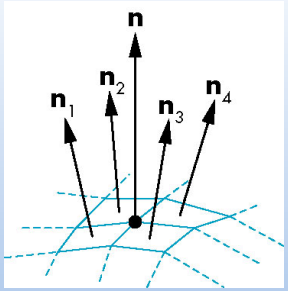
https://www.opengl.org/archives/resources/code/samples/sig99/advanced99/notes/node153.html

Phong Reflection Model



- OpenGL
  - Vertex normals
  - Material properties
  - Light properties
- Normal vectors (or normals)
  - Orientation of the surface
    - Defines how surface reflects light
  - Needs to be of unit length for correct lighting calculations
    - Note: length can be affected by transformations
  - Surface normal vs vertex normals





https://upload.wikimedia.org/wikipedia/commons/a/a5/Normal_vectors2.svg
Chetvorno, CC0, via Wikimedia Commons

Phong Reflection Model

- Point light
  - In vertex shader

```
layout(location = 0) in vec3 aPosition;
layout(location = 1) in vec3 aNormal;

// uniform input data
uniform mat4 uModelViewProjectionMatrix;
uniform mat4 uModelMatrix;
uniform mat3 uNormalMatrix;

// output data
out vec3 vPosition;
out vec3 vNormal;
```

Phong Reflection Model

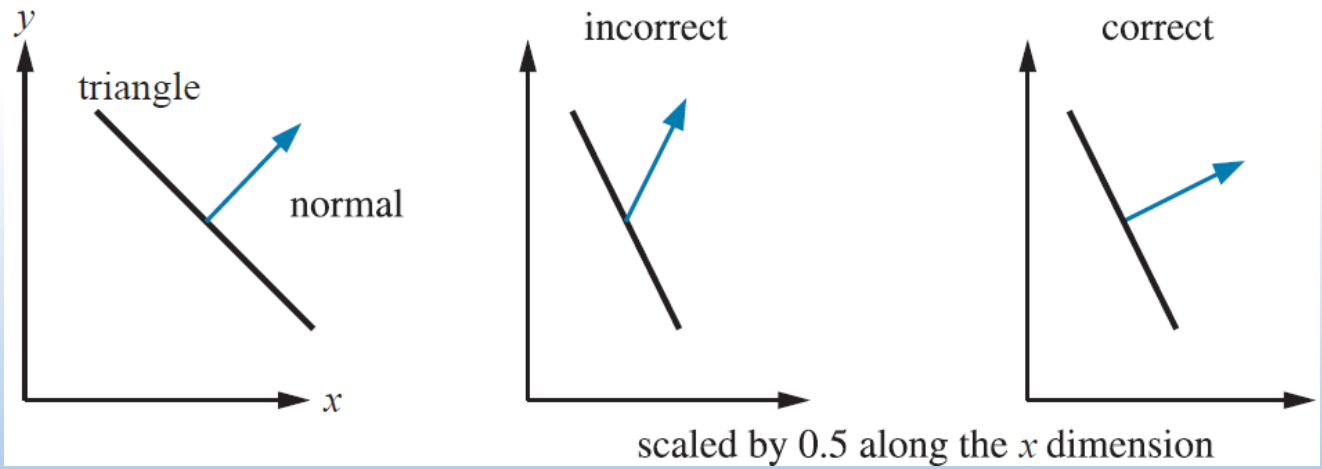
- Point light
  - In vertex shader

```
void main()
{
   gl_Position = uModelViewProjectionMatrix
       * vec4(aPosition, 1.0f);

   vPosition = (uModelMatrix * vec4(aPosition, 1.0f)).xyz;
   vNormal = uNormalMatrix * aNormal;
}
```

Phong Reflection Model

- Normal vectors
  - Length can be affected by transformations
  - Pass normal matrix to shader

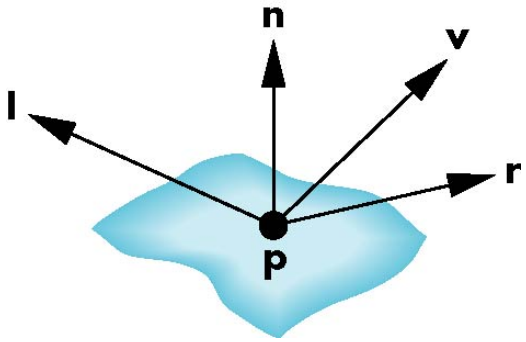
```
glm::mat3 normalMatrix =
   glm::mat3(glm::transpose(glm::inverse(gModelMatrix)));

gShader.setUniform("uNormalMatrix", normalMatrix);
```



Phong Reflection Model

- Point light
  - In fragment shader

```
uniform vec3 uViewpoint;
uniform Light uLight;
uniform Material uMaterial;

void main()
{
    vec3 n = normalize(vNormal);
    vec3 v = normalize(uViewpoint - vPosition);
    vec3 l = normalize(uLight.pos - vPosition);
    vec3 r = reflect(-l, n);
```



fragment's position

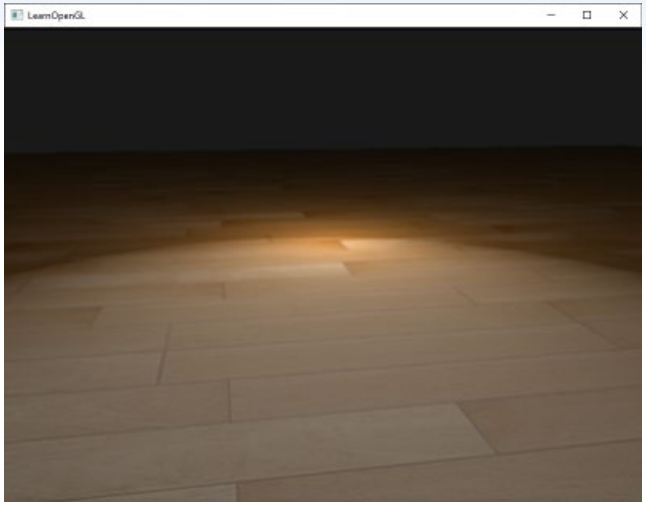
Phong Reflection Model

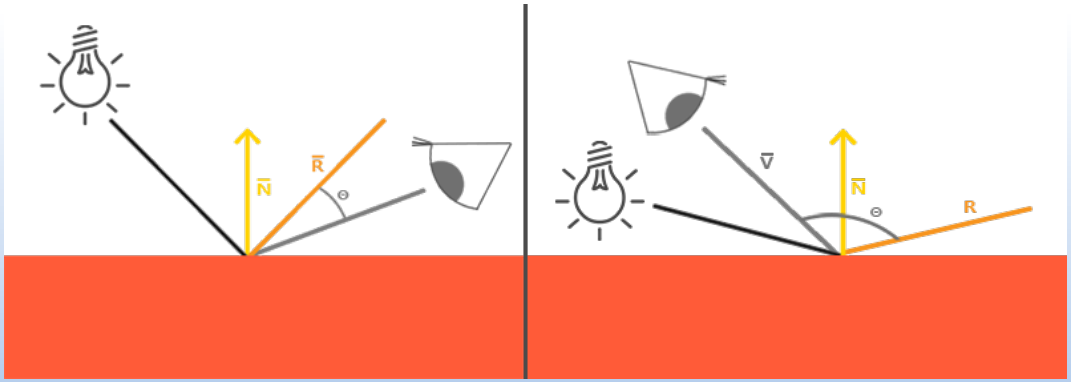
```
    vec3 Ia = uLight.La * uMaterial.Ka;
    vec3 Id = uLight.Ld * uMaterial.Kd
        * max(dot(l, n), 0.0f);
    vec3 Is = uLight.Ls * uMaterial.Ks
        * pow(max(dot(v, r), 0.0f), uMaterial.shininess);

    // attenuation
    float dist = length(uLight.pos - vPosition);
    float attenuation = 1.0f / (uLight.att.x
        + dist * uLight.att.y + dist * dist * uLight.att.z);

    fColor = attenuation * (Id + Is) + Ia;
}
```

$$I = \frac{1}{a + bd + cd^2}\left[k_d L_d \max(\boldsymbol{l}.\boldsymbol{n}, 0) + k_s L_s \max\left((\boldsymbol{r}.\boldsymbol{v})^{\alpha}, 0\right)\right] + k_a L_a$$

Blinn-Phong Reflection Model

- Phong reflection model
  - ➢ Potential issues
    - **r.v** clamped to zero
      - Assumes angle between perfect reflector and viewer cannot exceed 90^o
      - Otherwise dot product will be negative and clamped to 0
    - Efficiency
      - Requires the calculation of **r**



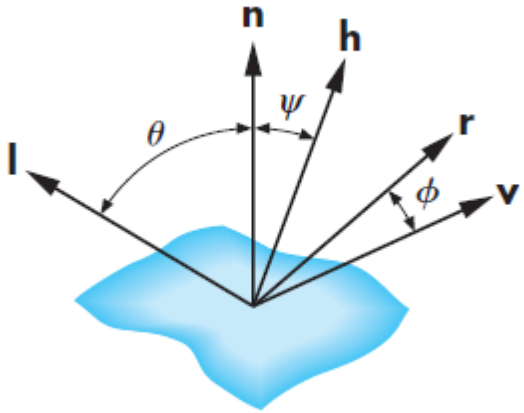


https://learnopengl.com/Advanced-Lighting/Advanced-Lighting

Blinn-Phong Reflection Model

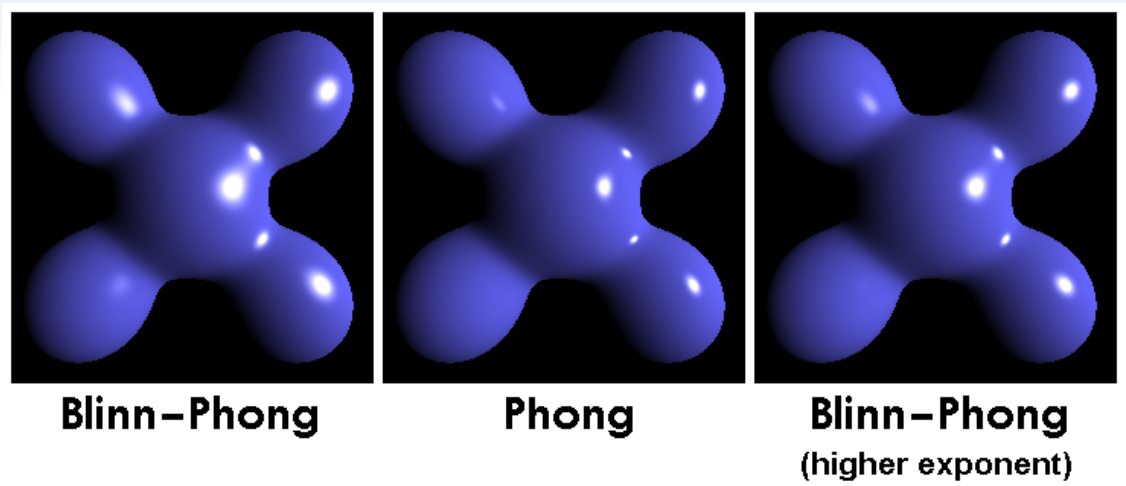
- Modified Phong
  - Approximation using
    - **Halfway** vector in calculation of specular term

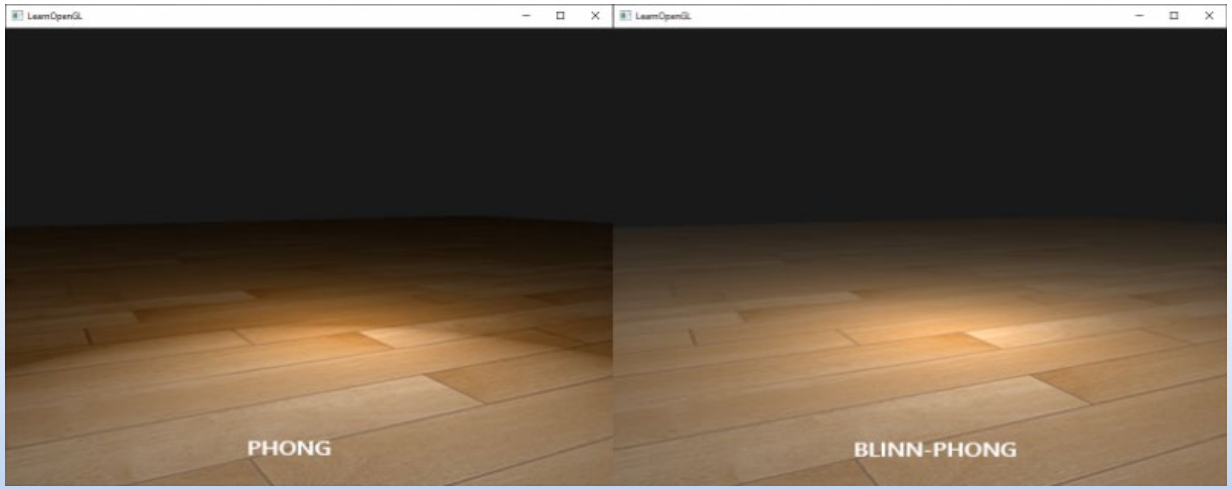
$$\mathbf{h} = \frac{\mathbf{l} + \mathbf{v}}{|\mathbf{l} + \mathbf{v}|}$$



  - Blinn-Phong reflection model
    - Replace $(\mathbf{r}.\mathbf{v})^{\alpha}$ with $(\mathbf{n}.\mathbf{h})^{\beta}$ in calculation of specular term
    - To be visually similar to the Phong reflection model
      - The value of the exponent must be higher
      - In general, between 2 and 4 times the Phong shininess value

Blinn-Phong Reflection Model



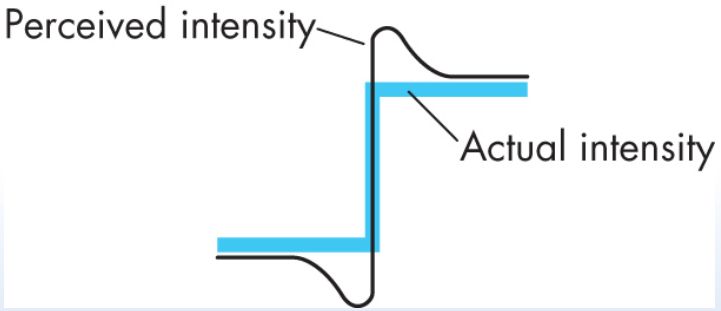


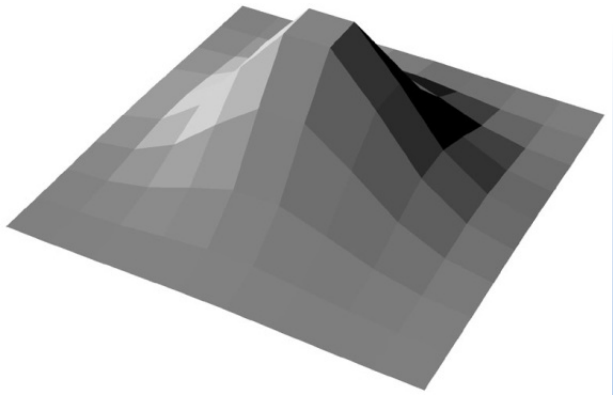
https://learnopengl.com/Advanced-Lighting/Advanced-Lighting

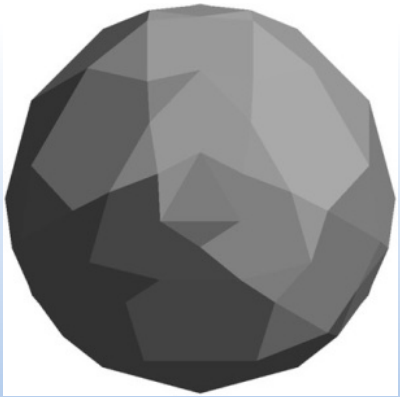
Polygonal Shading

- Flat shading
  - Unattractive, can see individual polygons
  - Mach bands
  - Sometimes useful in design or applications that want to identify individual polygonal facets

Mach bands

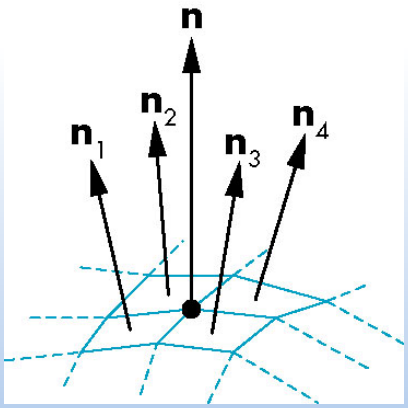






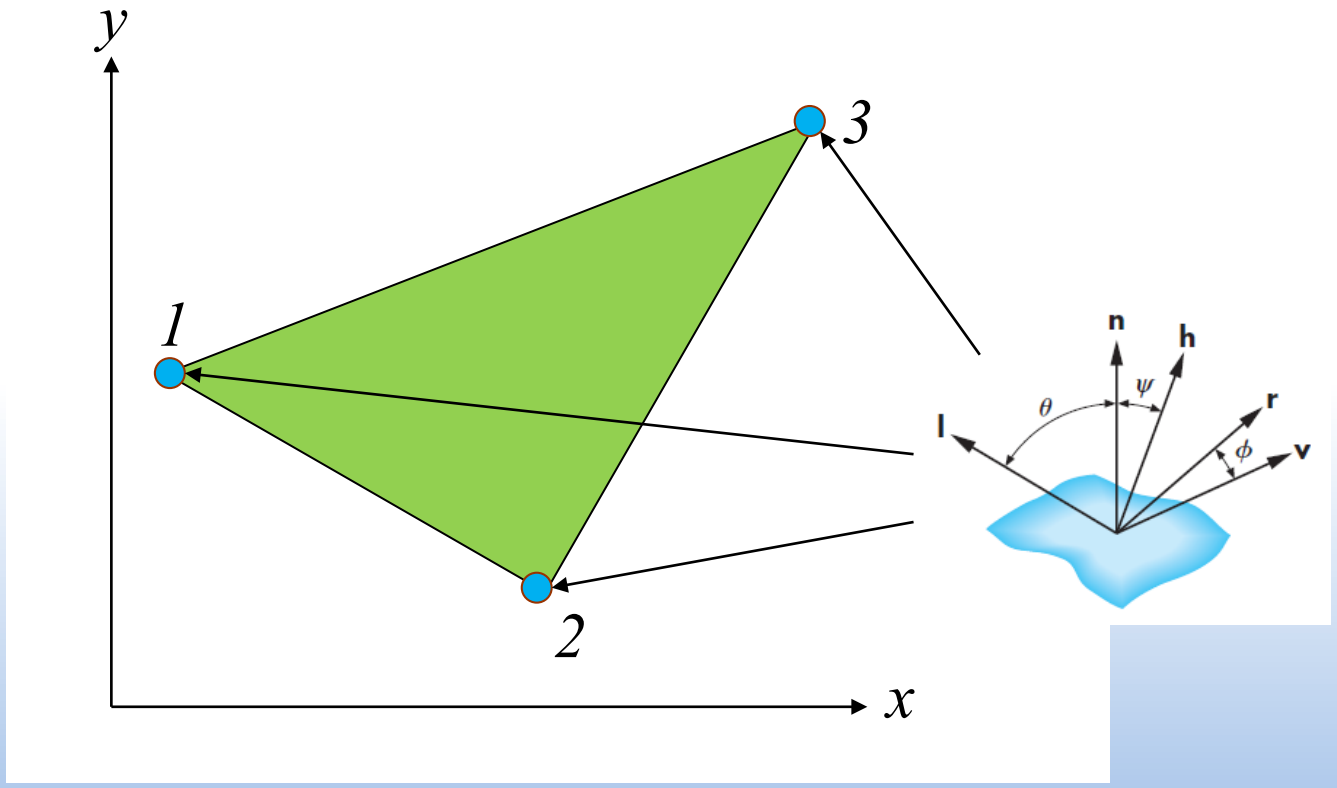
Polygonal Shading

- Gouraud shading (**per vertex**)
  - Also known as smooth shading
  - Interpolative vertex intensity values across polygon faces
    - Determine average unit normal vector at each polygon vertex
    - Apply an **illumination model at each polygon vertex** to obtain the light intensity at that position
    - Interpolate the **vertex intensities over the polygon**
  - Vertex normals
    - If normals supplied as per polygon rather than per vertex
    - Obtained vertex normal by averaging normals of all polygons that share that vertex



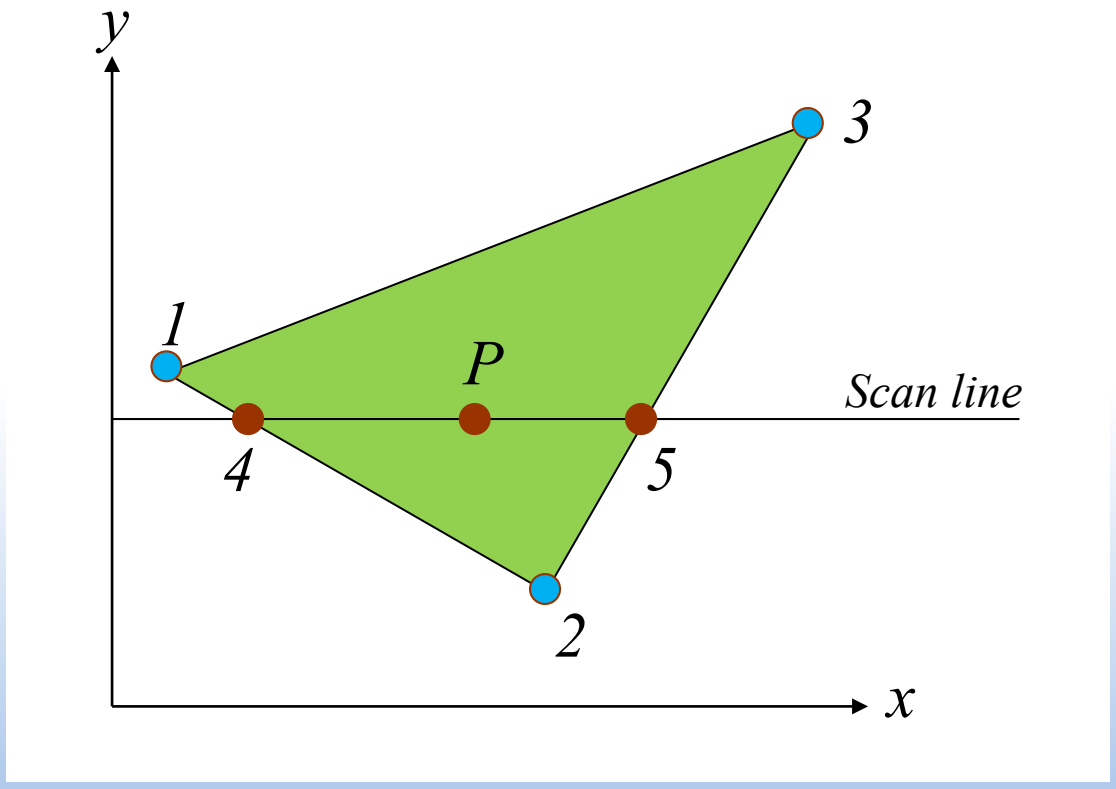
Polygonal Shading

- Gouraud shading
  - ➢ Per vertex lighting
    - Apply an **illumination model at each polygon vertex**



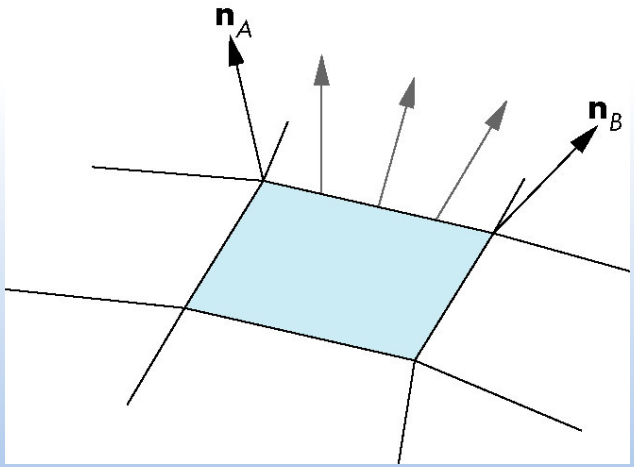
Polygonal Shading

- Gouraud shading
  - ➢ Per vertex lighting
    - Interpolate the **vertex intensities over the polygon**

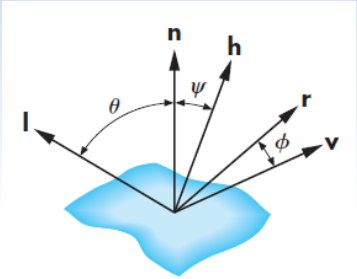


Polygonal Shading

- Phong shading (**per fragment**)
  - Instead of interpolating vertex intensities, interpolate vertex normal
    - Determine average unit normal vector at each polygon vertex
    - Interpolate the **vertex normals over the polygon**
    - Apply an **illumination model at each fragment** based on the interpolated vertex normal

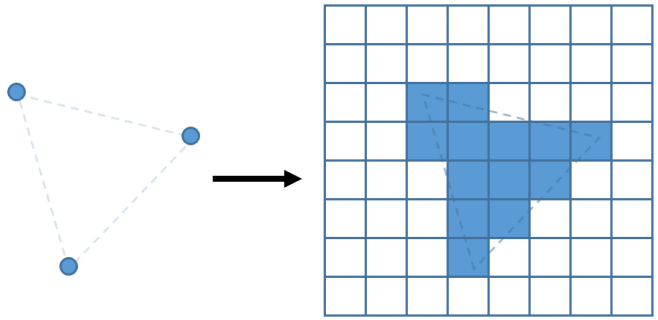


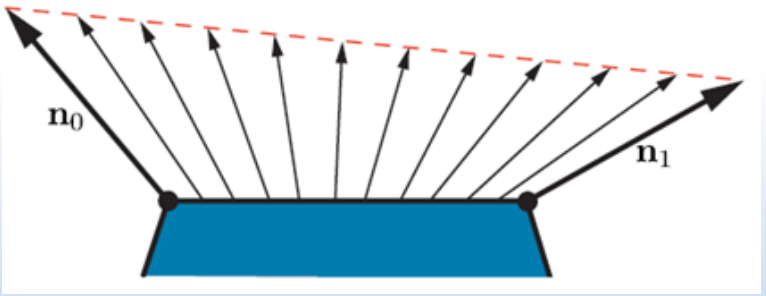
Polygonal Shading



- Phong shading
  - Requires much more computation than Gouraud shading
  - Lighting computation performed in the fragment shader
  - Must **normalise** normal vectors after interpolation

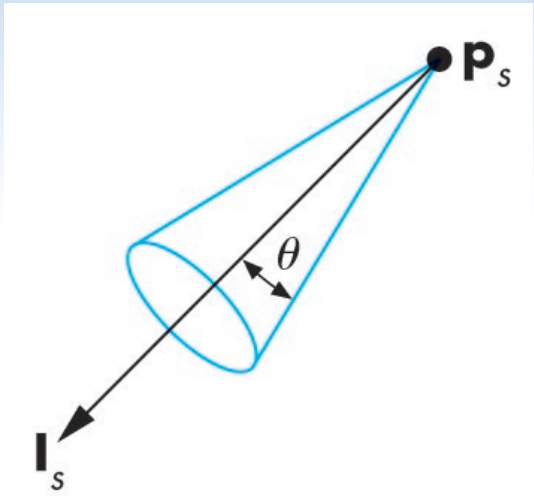
```
vec3 n = normalize(vNormal);
```

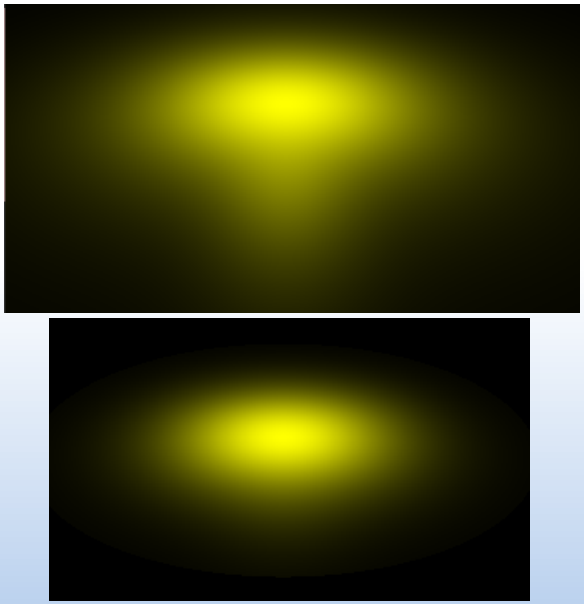


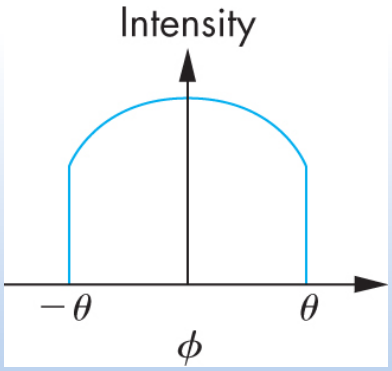


Spotlight

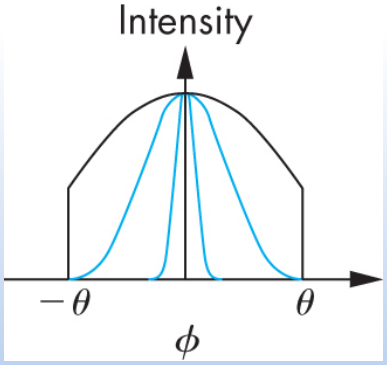
- Spotlight
  - ➢ Modified point light source with
    - Spotlight position and direction
    - Spotlight cutoff angle
      - If 180°, becames point light
      - No illumination outside this angle
    - Spotlight exponent
      - Decrease intensity away from centre







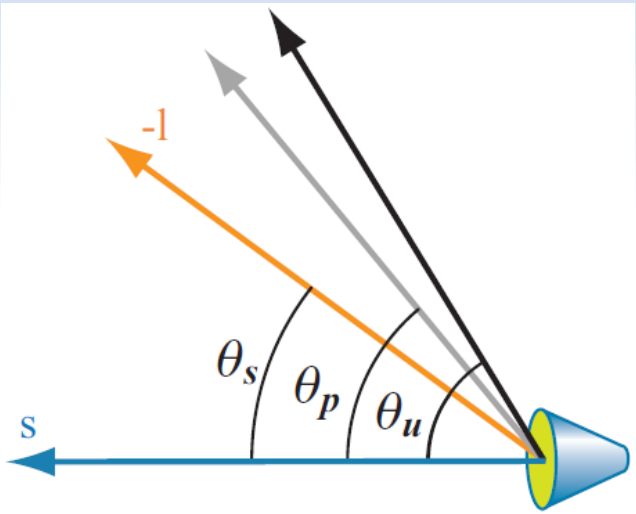
Attenuation

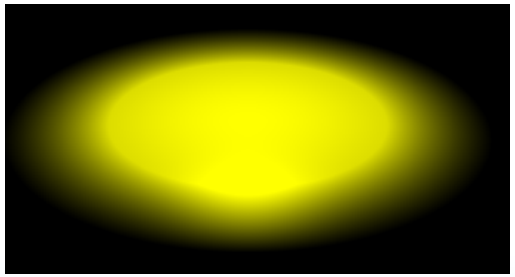


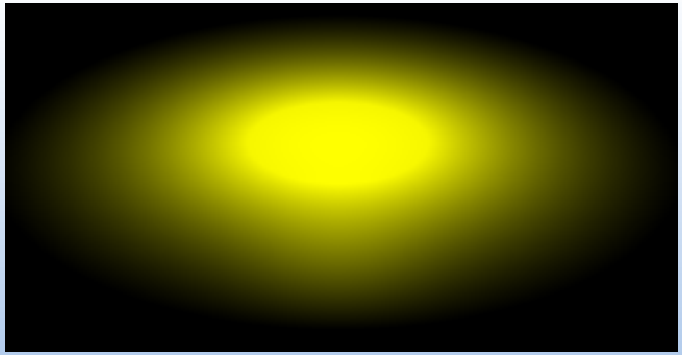
Exponent

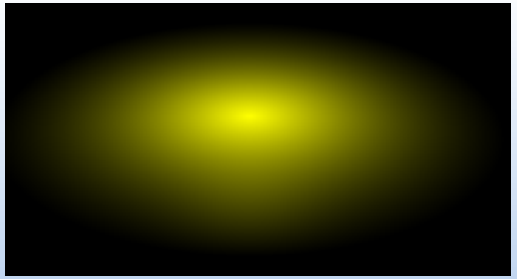
Spotlight

- Spotlight
  - ➢ Alternatively
    - Spotlight position and direction
    - Spotlight cutoff angles
      - – Inner angle
        - » Maximum intensity within inner angle
      - – Outer angle
        - » No illumination outside this angle
      - – Decrease intensity from inner to outer angle





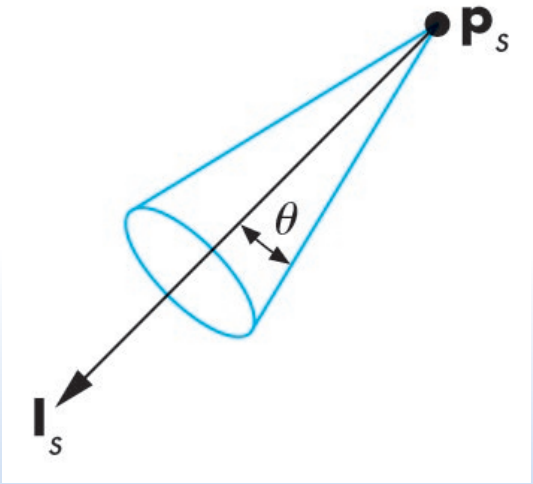


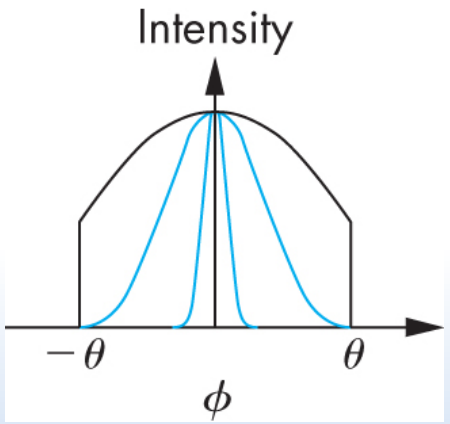


Spotlight

- Spotlight
  - Exponent approach

```
float cosAngle = dot(l, normalize(-uLight.dir));
float falloff = pow(cosAngle, uLight.exponent);

Id = ... * falloff;
Is = ... * falloff;
```



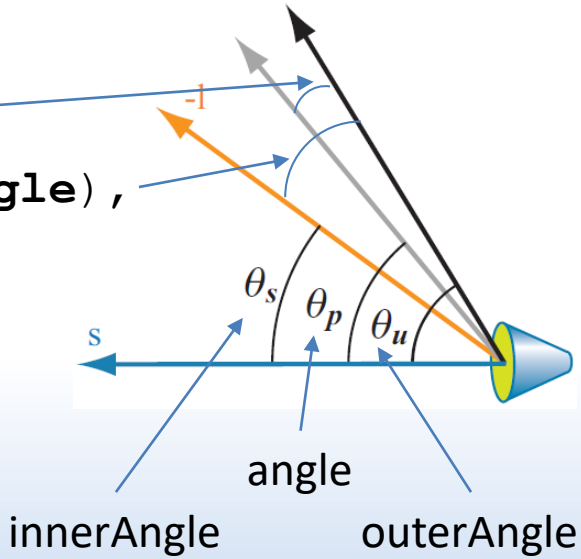


Exponent

Spotlight

- Spotlight
  - Inner/outer angle approach

```
float cosAngle = dot(l, normalize(-uLight.dir));
float angle = acos(cosAngle);
float falloff = clamp(
       (uLight.outerAngle - angle)/
       (uLight.outerAngle - uLight.innerAngle),
       0.0f, 1.0f);

Id = ... * falloff;
Is = ... * falloff;
```



Multiple Lights

- Multiple light sources
  - Apply illumination model for each light source
  - Accumulate the reflection intensities from each light

```
#define MAX_LIGHTS 7

uniform int uNumLights;
uniform Light uLight[MAX_LIGHTS];
uniform Material uMaterial;

vec3 point_light(int index, vec3 n, vec3 v);
vec3 directional_light(int index, vec3 n, vec3 v);
vec3 spotlight(int index, vec3 n, vec3 v);
```

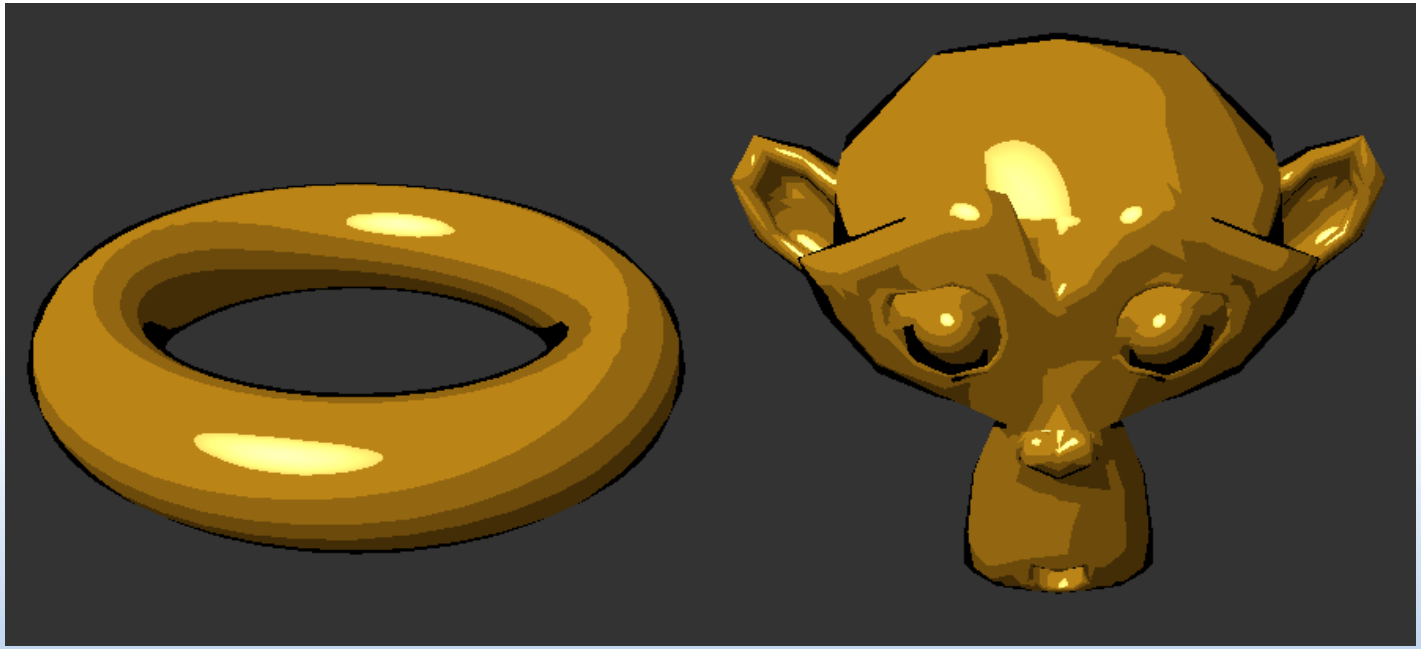
Multiple Lights

- Multiple light sources

```
vec3 n = normalize(vNormal);
vec3 v = normalize(uViewpoint - vPosition);

vec3 color = vec3(0.0f);   // accumulated color

for(int i = 0; i < uNumLights; i++){
   if(uLight[i].type == 1)        // point light
      color += point_light(i, n, v);
   else if(uLight[i].type == 2)   // directional light
      color += directional_light(i, n, v);
   else if(uLight[i].type == 3)   // spotlight
      color += spotlight(i, n, v);
}
```

Non-photorealistic Rendering

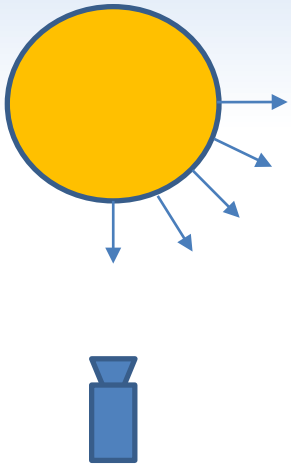
- Non-photorealistic rendering (NPR)
  - Stylistic rendering
  - E.g., toon shading

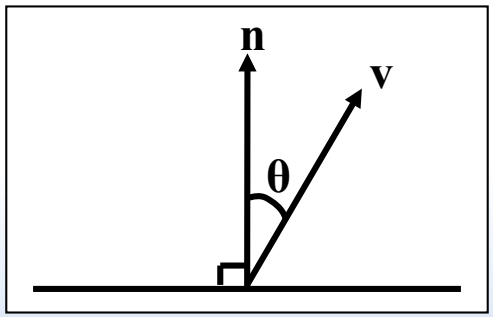


Non-photorealistic Rendering

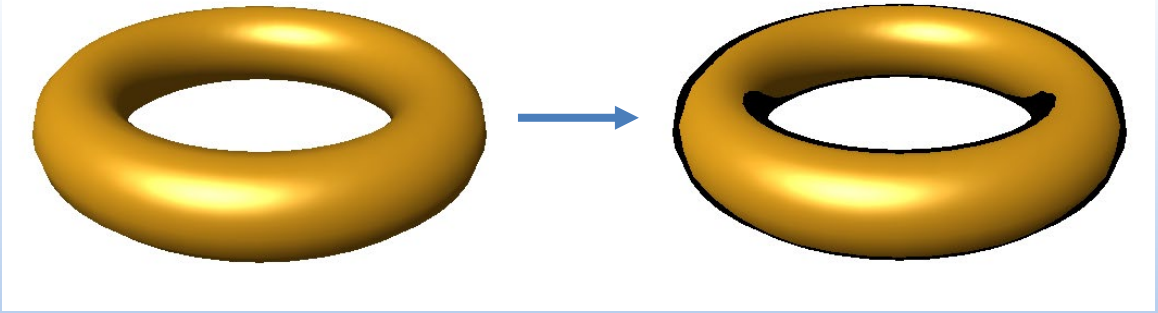
- Toon shading
  - ➢ Outline
    - Find the edge

```
if(dot(v, n) < uEdgeThickness)
{
        fColor = vec3(0.0f);
        return;
}
```





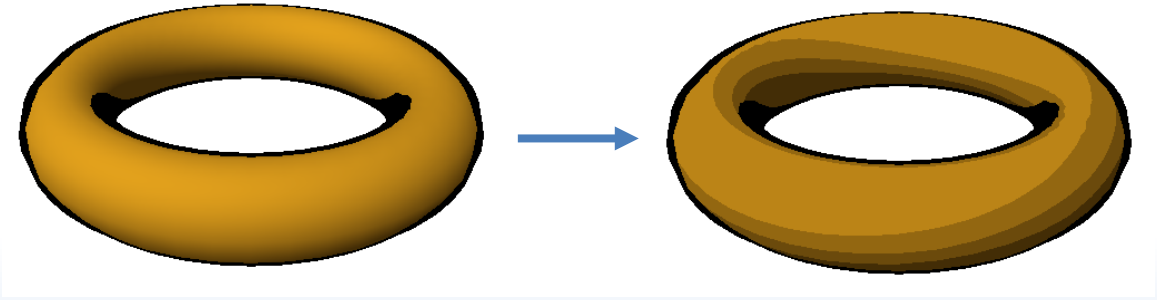
$$\cos\theta = \boldsymbol{v}.\boldsymbol{n}$$

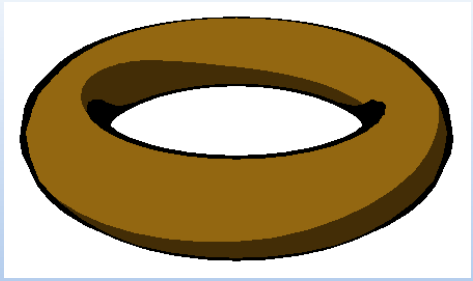


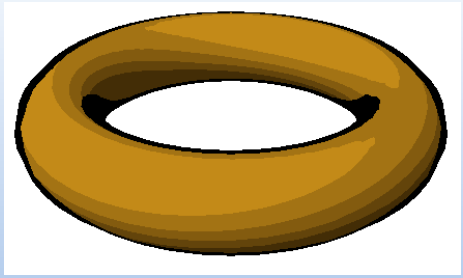
Non-photorealistic Rendering

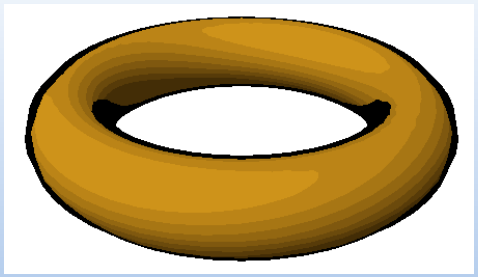
- Toon shading
  - Threshold the diffuse colour

```
float thresholdFactor = floor(dotLN * uNumOfThresholds)/
                                    uNumOfThresholds;
Id = uLight.Ld * uMaterial.Kd * thresholdFactor;
```





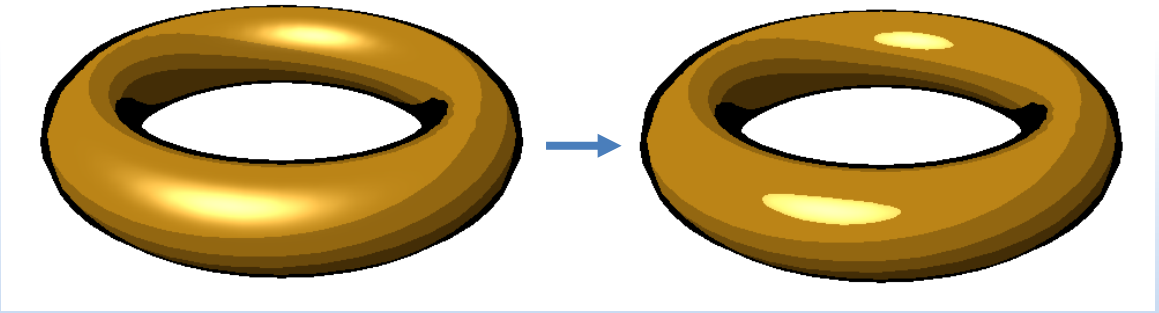




Non-photorealistic Rendering

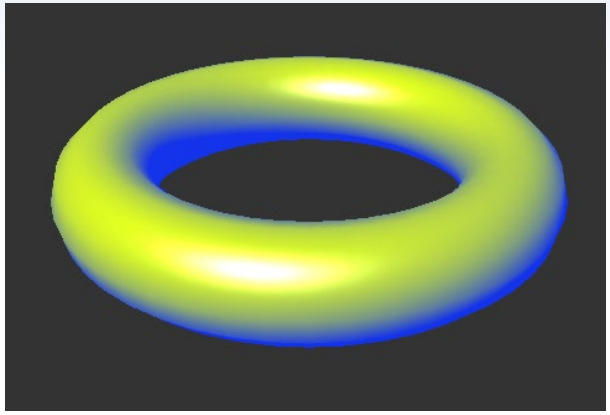
- Toon shading
  - Specular cutoff

```
float specularFactor = pow(max(dot(n, h), 0.0f),
    uMaterial.shininess);

if(specularFactor > uSpecularCutoff)
    Is = uLight.Ls * uMaterial.Ks * specularFactor;
```

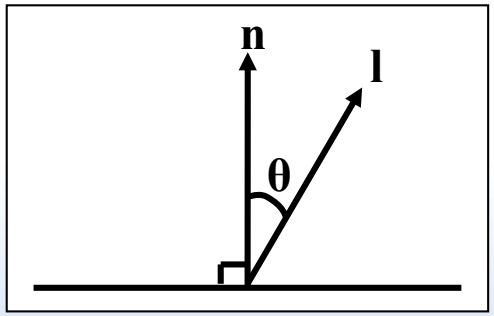


Non-photorealistic Rendering

- Gooch shading
  - ➢ Transition
    - From “Warm” colour
      - Facing the light
    - To “Cool” colour
      - Facing away from the light



```
float dotLN = max(dot(l, n), 0.0f);

vec3 kCool = uMaterial.coolColour
       + uAlpha * uMaterial.surfaceColour;
vec3 kWarm = uMaterial.warmColour
       + uBeta * uMaterial.surfaceColour;
vec3 colour = mix(kCool, kWarm, dotLN);
```



$$\cos\theta = \boldsymbol{l.n}$$

References

- Among others, material sourced from
  - Hearn, Baker & Carithers, “Computer Graphics with OpenGL”, Prentice-Hall
  - Angel & Shreiner, “Interactive Computer Graphics: A Top-Down Approach with OpenGL”, Addison Wesley
  - Akenine-Moller, Haines & Hoffman, “Real Time Rendering”, A.K. Peters
  - Rost, “OpenGL Shading Language”, Addison Wesley
  - Joey de Vries, “Learn OpenGL,” https://learnopengl.com/
  - https://www.khronos.org/opengl/wiki/
  - http://en.wikipedia.org/wiki/