VICTORIA UNIVERSITY OF WELLINGTON TE HERENGA WAKA

Lecture 6:

Introduction to lighting

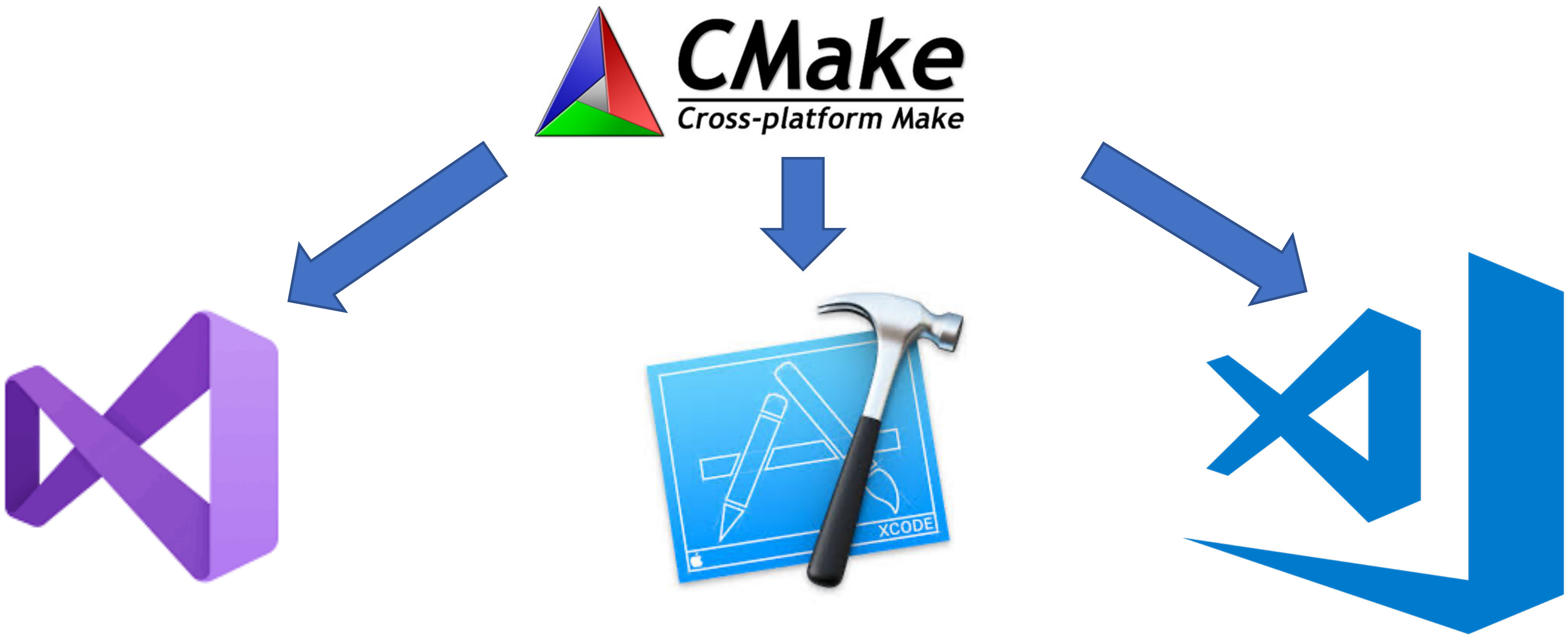
CGRA 354 : Computer Graphics Programming

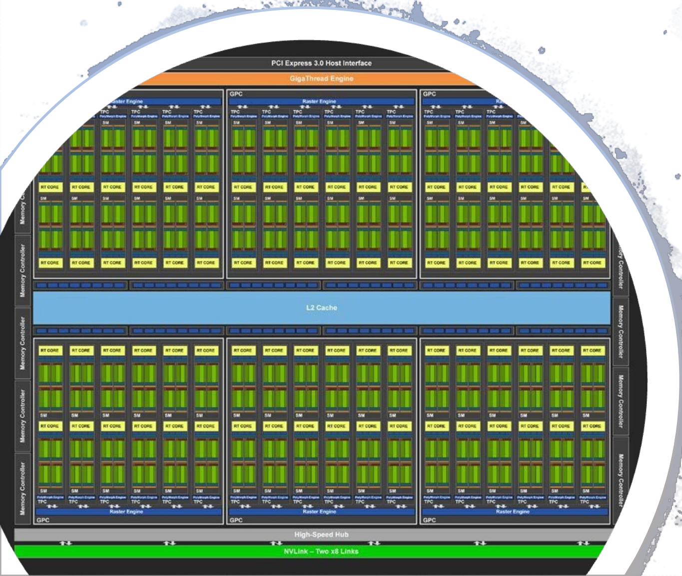
Instructor: Alex Doronin
Cotton Level 3, Office 330
alex.doronin@vuw.ac.nz

With slides from: Steve Marschner, Cornell; Taehyun Rhee, CMIC; Zohar Levi, Mike Bailey, OSU, Ed Angel University of New Mexico

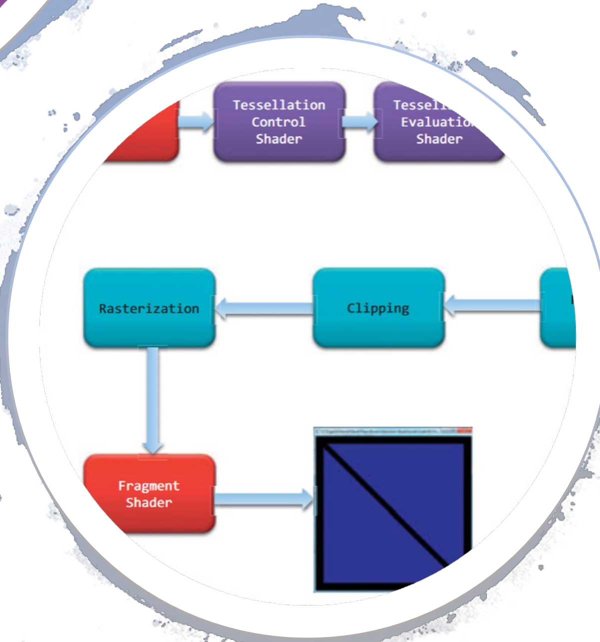
Recap: Building CGRA251 Framework

- *Integrated development environments (IDEs) on Windows, Mac, Linux and CMake tools*





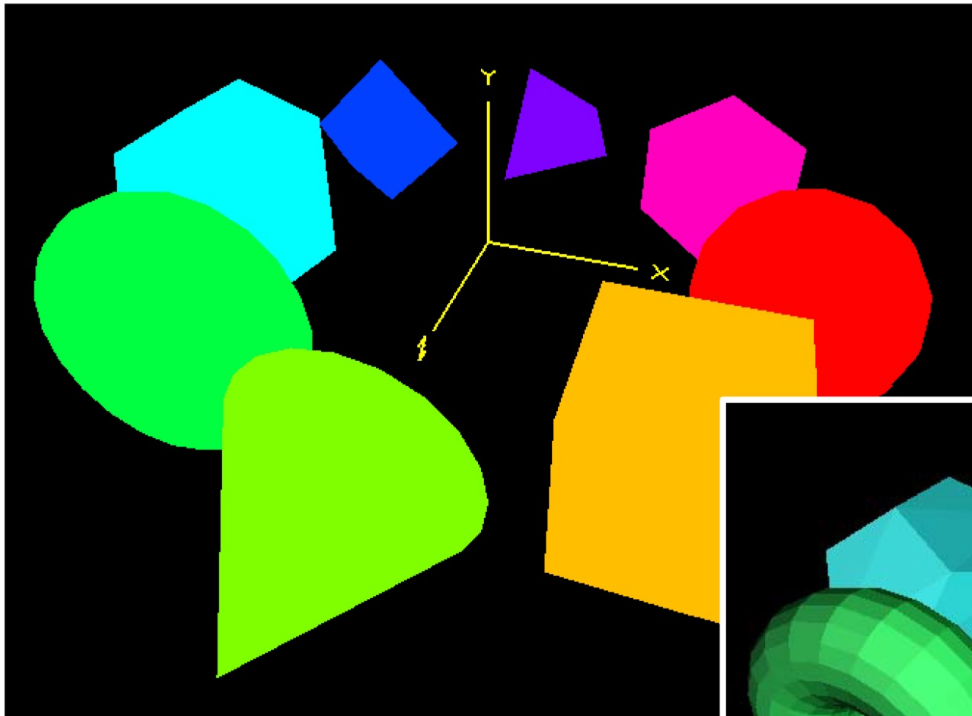
Recap: Shader Program



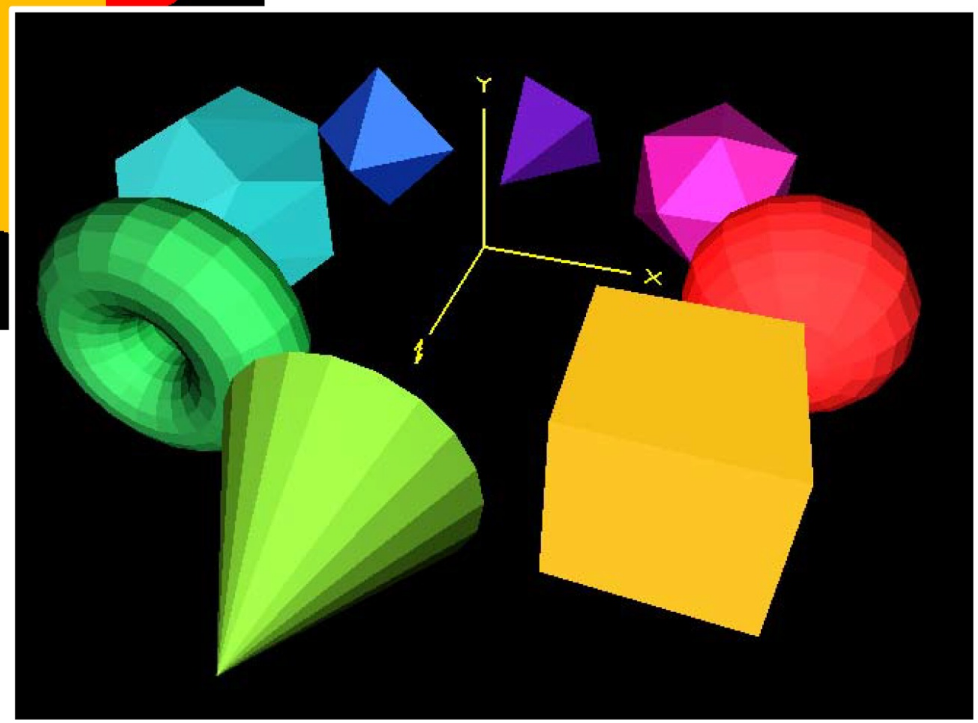
- A small C/C++ style (GLSL) program to control parts of the graphics pipeline
- Consists of 2 (or more) separate parts:
  - **Vertex shader** controls vertex transformation.
  - **Fragment shader** controls fragment shading.

Why Do We Care About Lighting?

Lighting “dis-ambiguates” 3D scenes



Without lighting



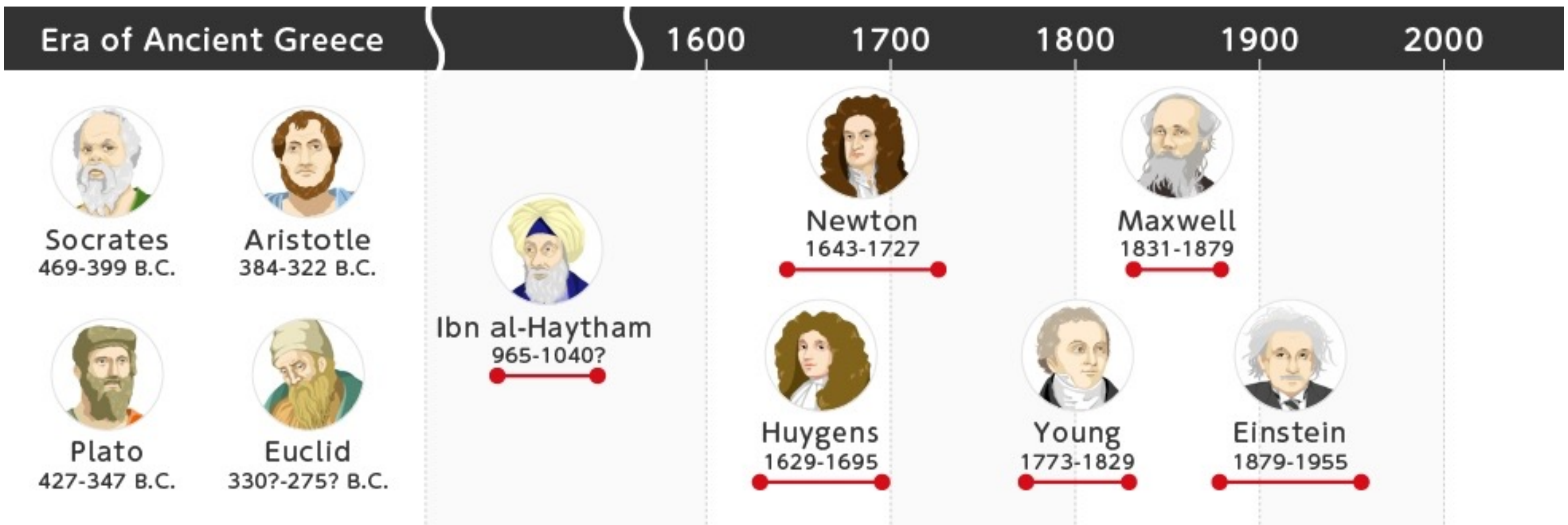
With lighting

What is light ?

- Light is Electromagnetic **Energy**
- Light comes from many different sources
- Some items ***produce*** light while others ***reflect*** light

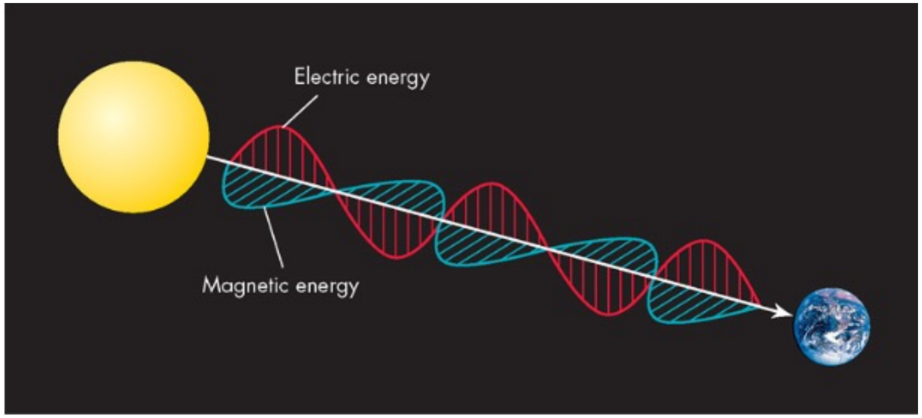
Light: introduction

Light is both a **wave** and a **particle:**



By Hamamatsu at http://photonterrace.net/en/photon/history/

The Nature of light



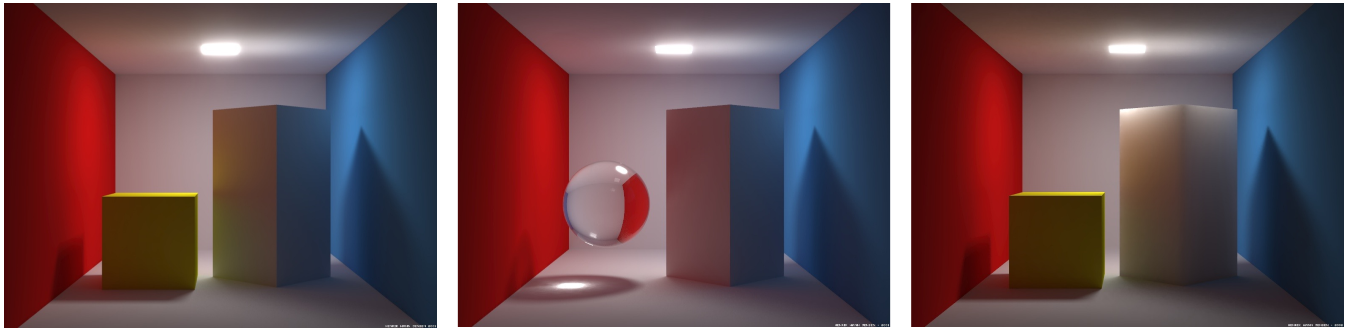
- Light is radiant energy.
- Travels very fast – 300,000 km/sec!
- Can be described either as a wave or as a particle traveling through space.

As a wave:
– A small disturbance in an electric field creates a small magnetic field, which in turn creates a small electric field, and so on
• Light propagates itself "by its bootstraps!"
– Light waves can interfere with other light waves, canceling or amplifying them!
– The color of light is determined by its wavelength

As a particle:
– Particles of light (photons) travel through space.
– These photons have very specific energies. that is, light is quantized.
– Photons strike your eye (or other sensors) like a very small bullet, and are detected.

Lighting

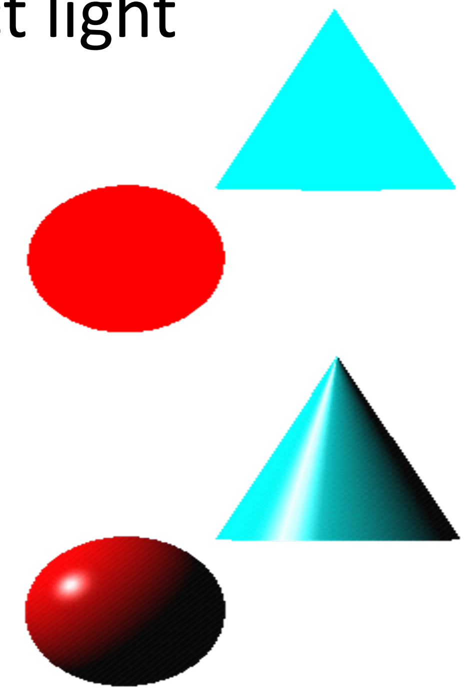
- Lighting or illumination is the deliberate use of light to achieve a practical or aesthetic effect
- Illumination model
  - Models deal with physical interactions between
    - lights, geometry, materials, textures, transparency,
      interaction with (within) surface, etc
  - Simulate light (photons) interacting through the scene



http://graphics.stanford.edu/~henrik/images/cbox.html

Lighting Principles

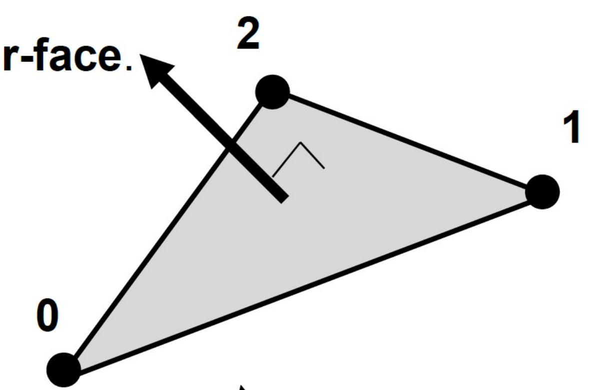
- Lighting simulates how objects reflect light
  - material composition of object
  - light’s color and position
  - global lighting parameters
- Usually implemented in
  - vertex shader for faster speed
  - fragment shader for nicer shading



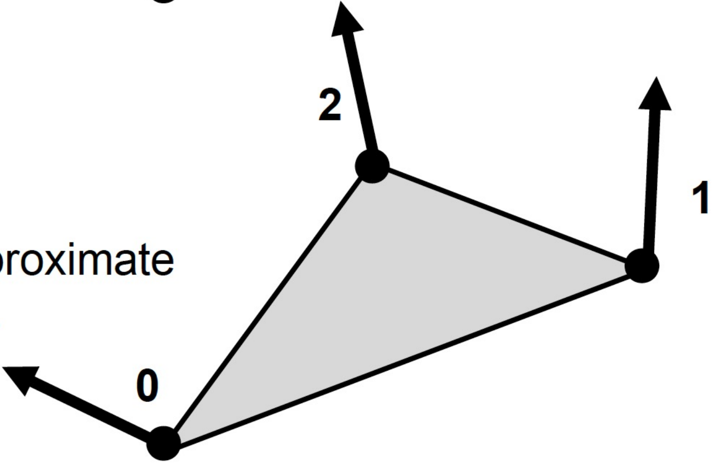
The Normal

A surface normal is a vector perpendicular to the surface.

Sometimes surface normals are defined or computed **per-face**.



Sometimes they are defined **per-vertex** to best approximate the underlying surface that the face is representing.



Recap: Points and Vectors

- The rendering pipeline transforms vertices, normals, colors, texture coordinates
- Points (e.g. vertices) specify a location in space
- Vector (e.g. normal) specify a direction
- Relations between points and vectors
  - point – point = vector
  - point + vector = point
  - vector + vector = vector
  - point + point = not defined
  - **p** = P1 – P2, P1 = P2 + **p**

Homogeneous coordinates:

if w == 1, then the vector (x,y,z,1) is a position in space.

If w == 0, then the vector (x,y,z,0) is a direction.

Dot and Cross Product

- Dot product **a•b:**

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \bullet \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = a_x.b_x + a_y.b_y + a_z.b_z$$

  a•b = 0, if v and w are perpendicular,

  If both are normalized, it is directly the cosine of the angle between them: $\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}||\ ||\mathbf{b}|| \cos\theta$

- Cross product **axb**

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} a_y b_z - b_y a_z \\ a_z b_x - b_z a_x \\ a_x b_y - b_x a_y \end{pmatrix}$$

Results in a vector that is perpendicular to both of them

Vector Normalization

- To compute a new vector pointing in the same direction but unit length
- Normalized vector = unit vector
- Divide each component of **v** by ||**v**||

GLSL examples: vec3

```
vec3 a;
a.x = 10.0; a.y = 20.0; a.z = 30.0;   // a = (10,  20,  30)
a.r = 0.1;  a.g = 0.2;  a.b = 0.3;    // a = (0.1, 0.2, 0.3)
a.s = 1.0,  a.t = 2.0;  a.p = 3.0;    // a = (1, 2, 3)

vec3 b = vec3(4.0, 5.0, 6.0);

vec3 c = a + b; // c = (5, 7, 9)
vec3 d = a - b; // d = (-3, -3, -3)
vec3 e = a * b; // e = (4, 10, 18)
vec3 f = a * 3; // e = (3, 6, 9)
float g = dot(a,b); // g = 32
vec3 h = cross(a,b); // h = (-3,6,-3)
float i = length(a); // i = 3.742
```

Dot: angle between two lines

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \bullet \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = a_x.b_x + a_y.b_y + a_z.b_z$$

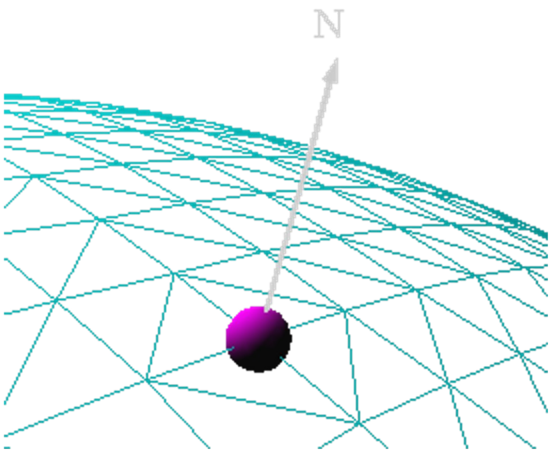
Cross: line perpendicular to two lines:

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \times \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \begin{pmatrix} a_y b_z - b_y a_z \\ a_z b_x - b_z a_x \\ a_x b_y - b_x a_y \end{pmatrix}$$

Length (magnitude): $c = \sqrt{(c_x^2 + c_y^2 + c_z^2)}$

Surface Normals

- Normals define how a surface reflects light
  - Application usually provides normals as a vertex atttribute
  - Current normal is used to compute vertex's color
  - Use unit normals for proper lighting
    - scaling affects a normal's length



Surface Normal

- A surface normal of a triangle can be calculated by taking the vector cross product of two edges of that triangles

$\mathbf{u} = \mathbf{v_2} - \mathbf{v_1}$

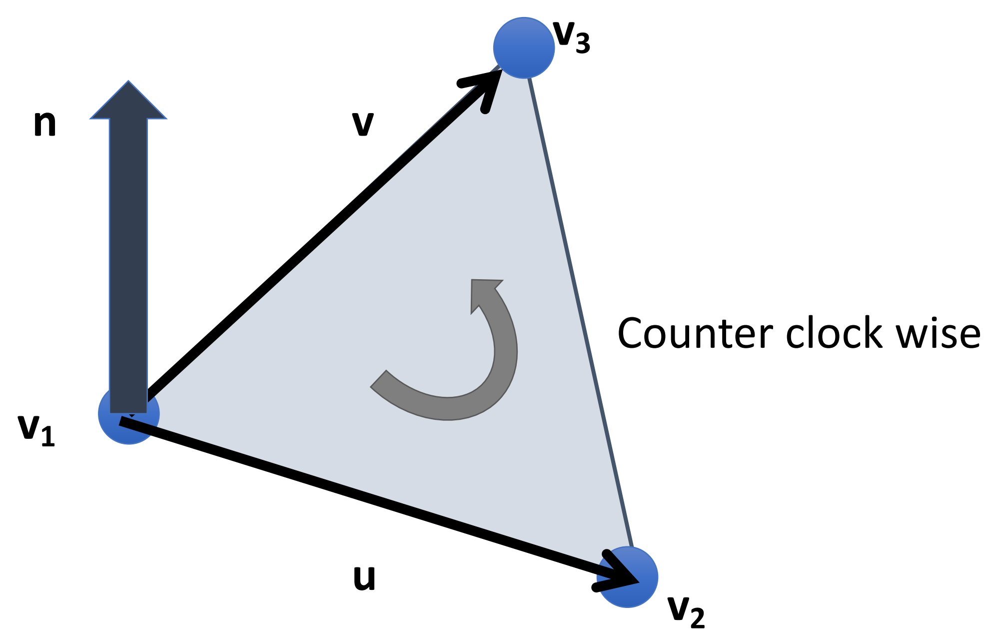
$\mathbf{v} = \mathbf{v_3} - \mathbf{v_1}$

$\mathbf{n} = \mathbf{u} \times \mathbf{v}$

$n_x = u_y v_z - u_z v_y$

$n_y = u_z v_x - u_x v_z$

$n_z = u_x v_y - u_y v_x$



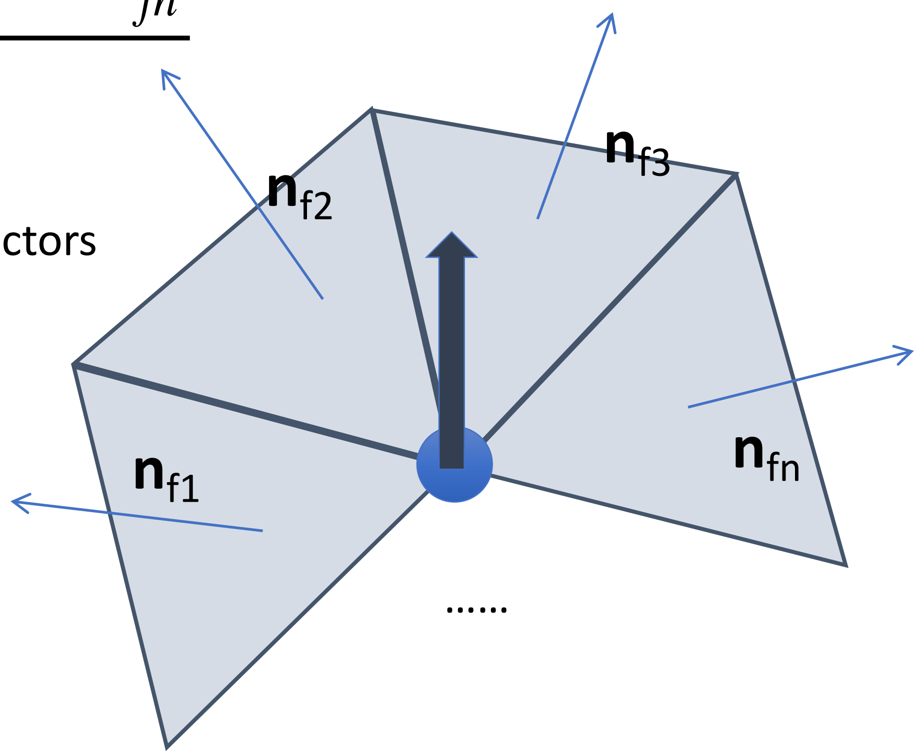
Vertex Normal

- Normalized sum of surface normals at the vertex

$$\mathbf{n} = \frac{\mathbf{n}_{f1} + \mathbf{n}_{f2} + \mathbf{n}_{f3} + \ldots + \mathbf{n}_{fn}}{\|\mathbf{n}\|}$$

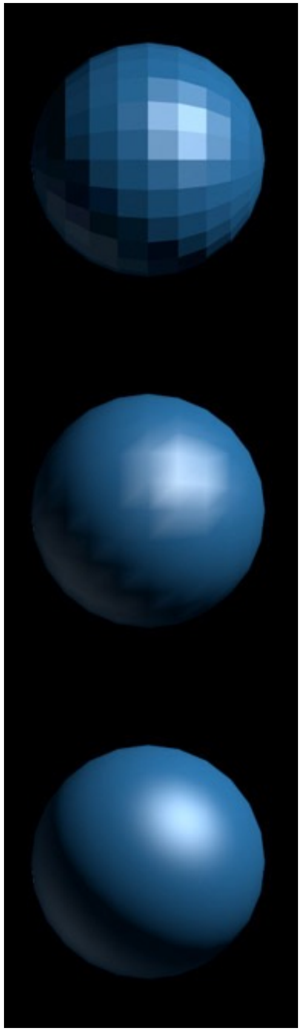
* Each surface normal should be unit vectors

$$\|\mathbf{n}\| = \sqrt{\mathbf{n}_x^2 + \mathbf{n}_y^2 + \mathbf{n}_z^2}$$

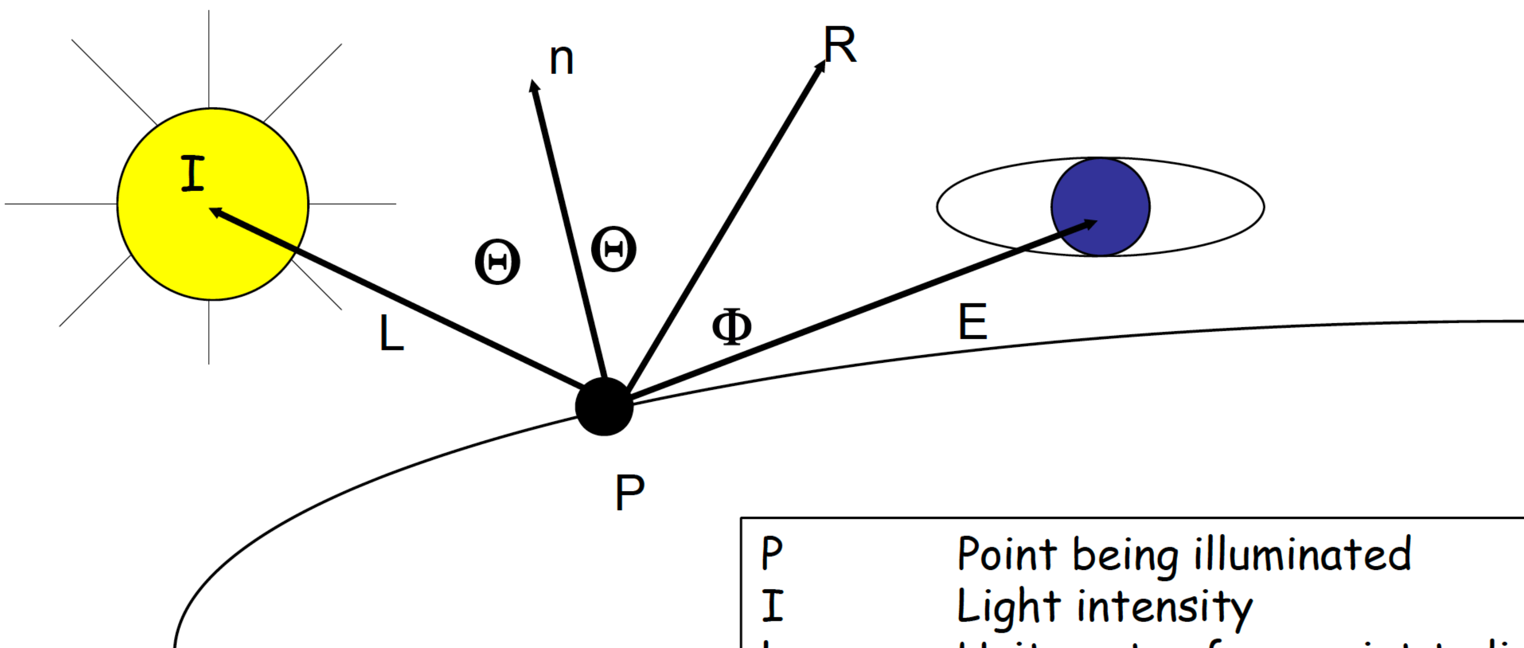


Shading Model

- **Flat shading**
  - Evaluate lighting per vertex using surface normal
- **Gouraud shading**
  - Evaluate lighting per vertex using vertex normal
- **Phong shading**
  - Evaluate lighting per fragment using interpolated normal



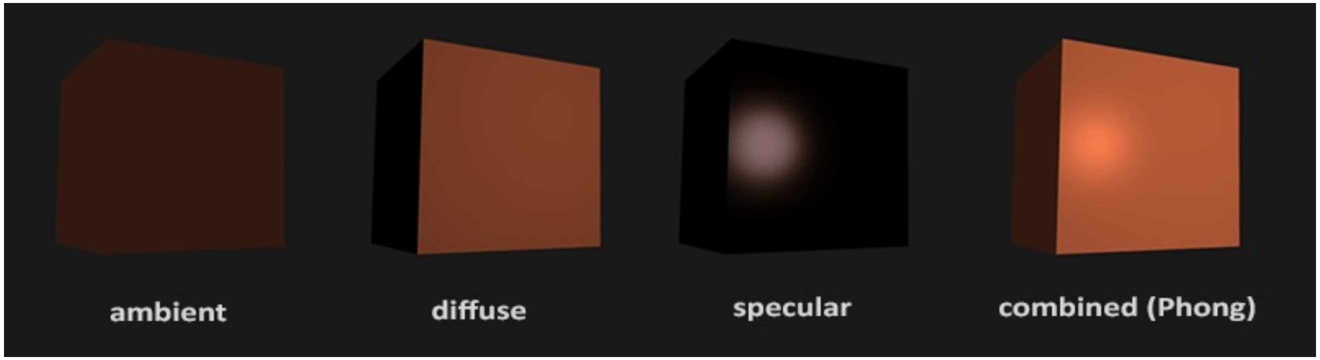
A typical Lighting configuration



P	Point being illuminated
I	Light intensity
L	Unit vector from point to light
n	Unit vector surface normal
R	Perfect reflection unit vector
E	Unit vector to eye position

Simple shading model: components

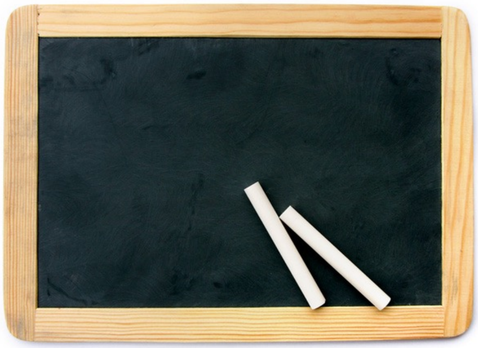
- Illumination model express the components of light “reflected from” or
  “transmitted through”
  (refracted or scattered) a surface
- We will deal with three basic lit components
  - Ambient
  - Diffuse
  - Specular

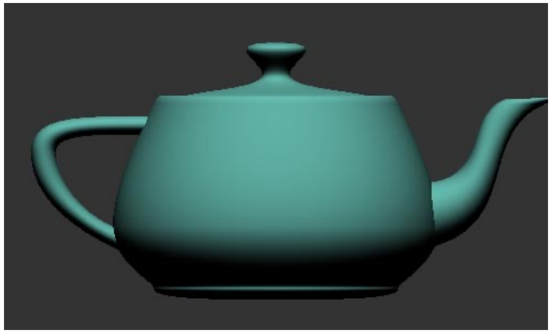


https://learnopengl.com/Lighting/Basic-Lighting

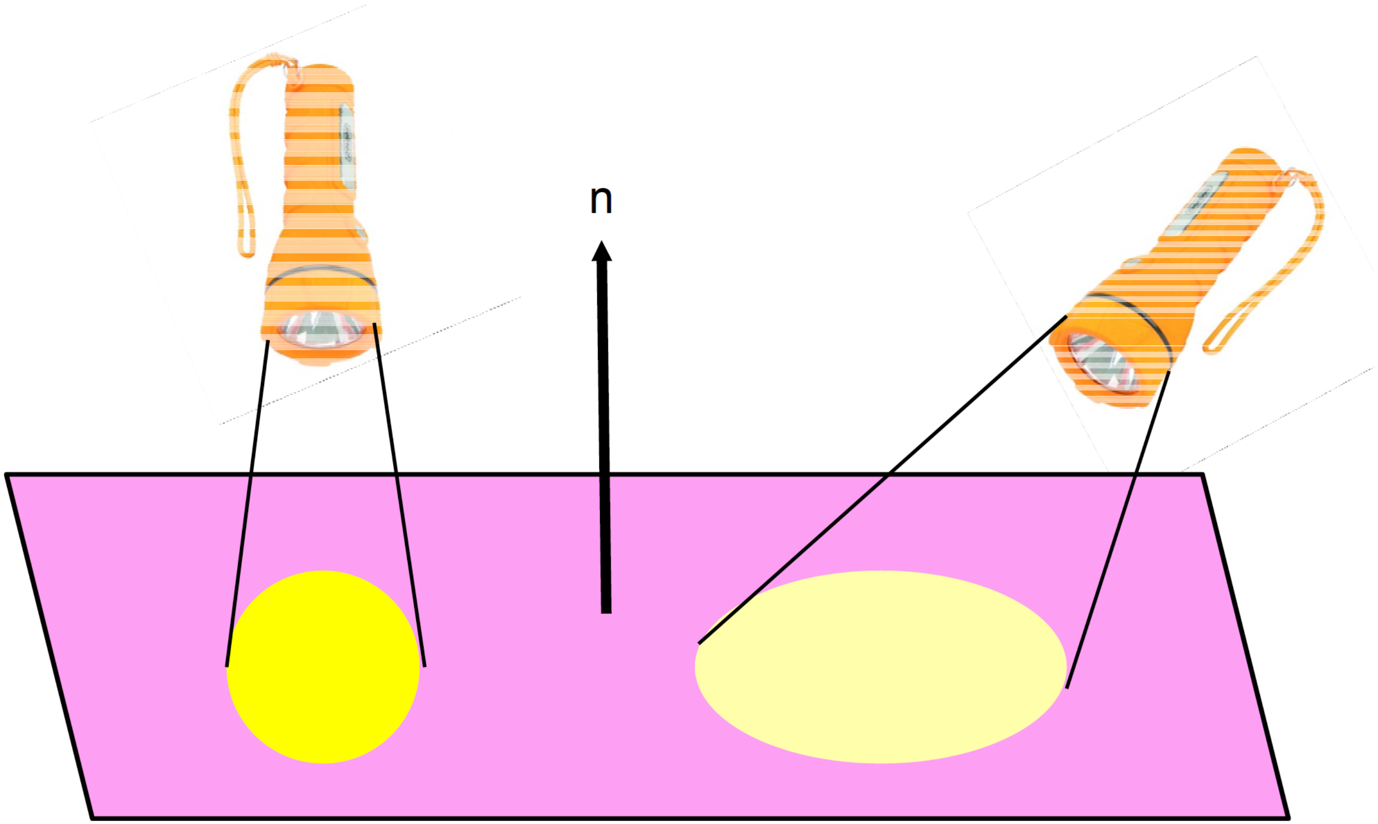
Diffuse reflection

- Diffuse
  - Incident light is reflected into all directions
    - Photons are scattered equally in all directions
  - Diffusely reflected light is typically for dull, matte surface such a paper, chalk or chalkboard





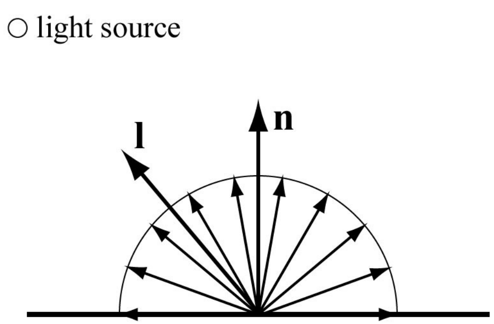
Diffuse reflection



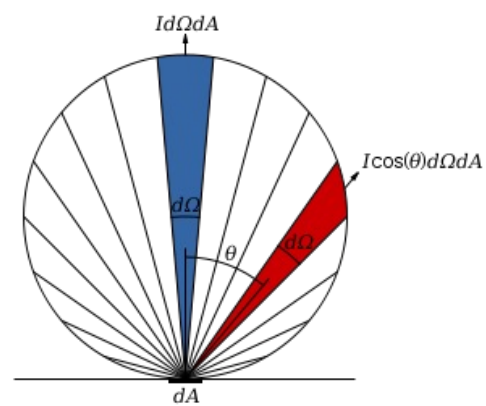
* Spreading out the same amount of light energy across more surface area

Diffuse reflection

- Component of diffuse reflection is based on Lambert's law
  - radiant intensity reflected from a fully diffuse $i_{diff} = \mathbf{n} \cdot \mathbf{l} = \cos\theta$
    surface is proportional to the angle between
    light direction **l** and surface normal **n**



[Image from real time rendering book]



[Image from wikipedia]

In the CGRA251 Framework

default_vert.glsl

default_vert.glsl 〉 No Selection

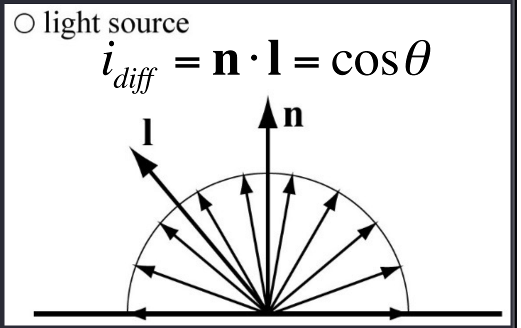
```
#version 330 core

// uniform data
uniform mat4 uProjectionMatrix;
uniform mat4 uModelViewMatrix;

// mesh data
layout(location = 0) in vec3 aPosition;
layout(location = 1) in vec3 aNormal;

// model data (this must match the input of the vertex shader)
out VertexData {
    vec3 position;
    vec3 normal;
} v_out;

void main() {
    // transform vertex data to viewspace
    v_out.position = (uModelViewMatrix * vec4(aPosition, 1)).xyz;
    v_out.normal = normalize((uModelViewMatrix * vec4(aNormal, 0)).xyz);

    // set the screenspace position (needed for converting to fragment data)
    gl_Position = uProjectionMatrix * uModelViewMatrix * vec4(aPosition, 1);
}

```

In the CGRA251 Framework

default_frag.glsl

default_frag.glsl 〉 No Selection

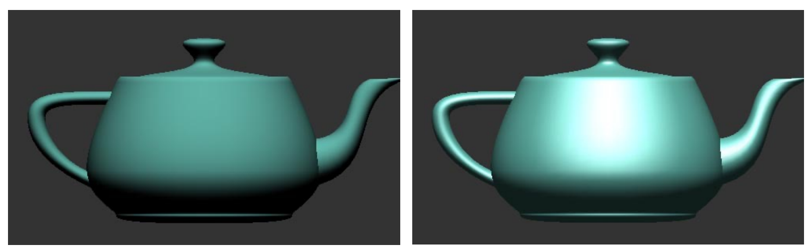
```glsl
#version 330 core

// uniform data
uniform mat4 uProjectionMatrix;
uniform mat4 uModelViewMatrix;

// viewspace data (this must match the output of the fragment shader)
in VertexData {
    vec3 position;
    vec3 normal;
} f_in;

// framebuffer output
out vec4 fb_color;

void main() {
    // calculate shading
    vec3 surfaceColor = vec3(0.066, 0.341, 0.215);
    vec3 eye = normalize(-f_in.position); // direction towards the eye
    float light = abs(dot(normalize(f_in.normal), eye)); // difference between the surface normal and direction towards the eye
    vec3 finalColor = mix(surfaceColor / 4, surfaceColor, light);

    // output to the frambuffer
    fb_color = vec4(finalColor, 1);
}

```



Specular highlights

- Specular
  - Deals with reflection into a dominant direction causing highlights effect on the surface
  - Produce shiny spot on the surface such as billiard ball

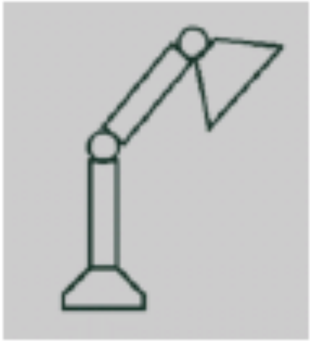
http://www.joshenreborn.com/2013/04/against-all-odds-create.html



[Image from real time rendering book]

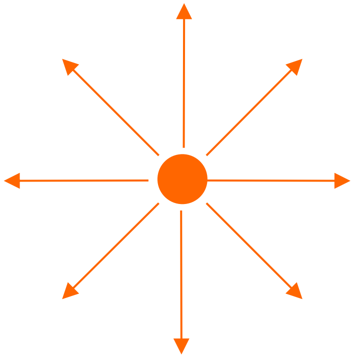
Simple Light Source Models

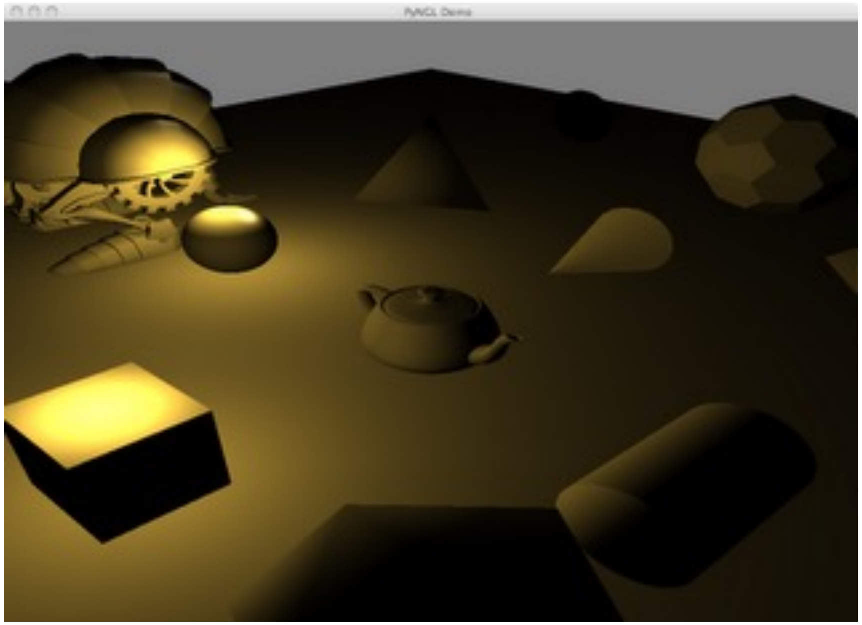
- Simple mathematical models:
  - Point Light
  - Directional Light
  - Spot Light
- Two other light properties
  - Ambient Light
  - Emission



Point Light

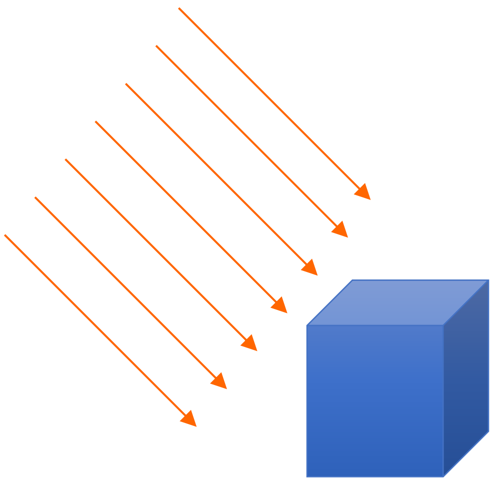
- A light source originating from a zero-volume point in the scene
- Emit light in all direction from a point

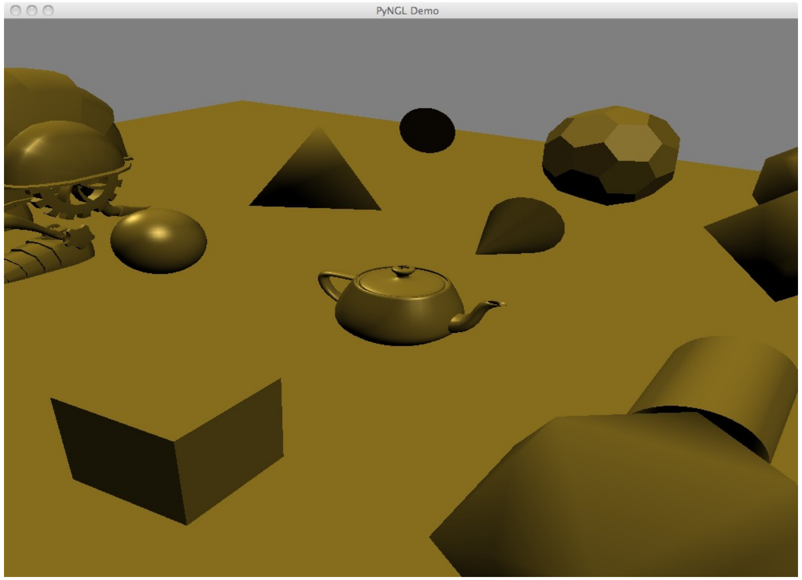




Directional Light

- A light infinitely far away from the scene only having direction
- Often for emulating sunlight

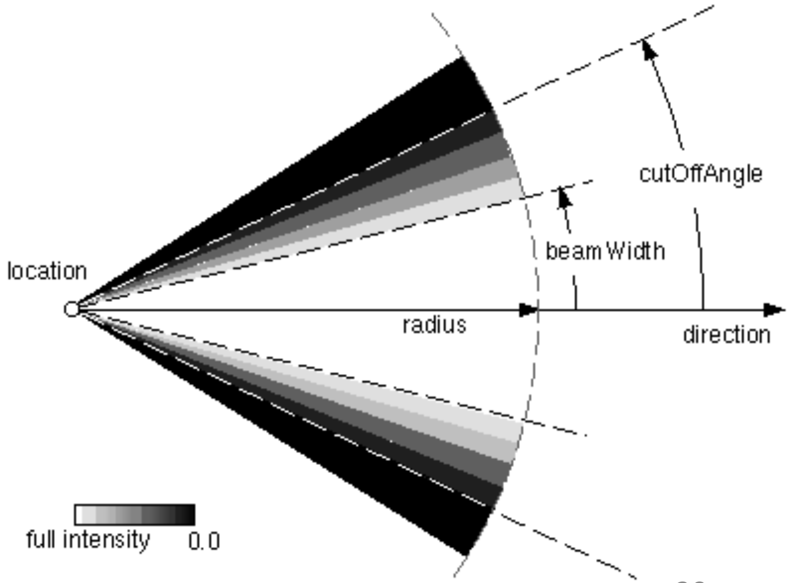




Spot Light

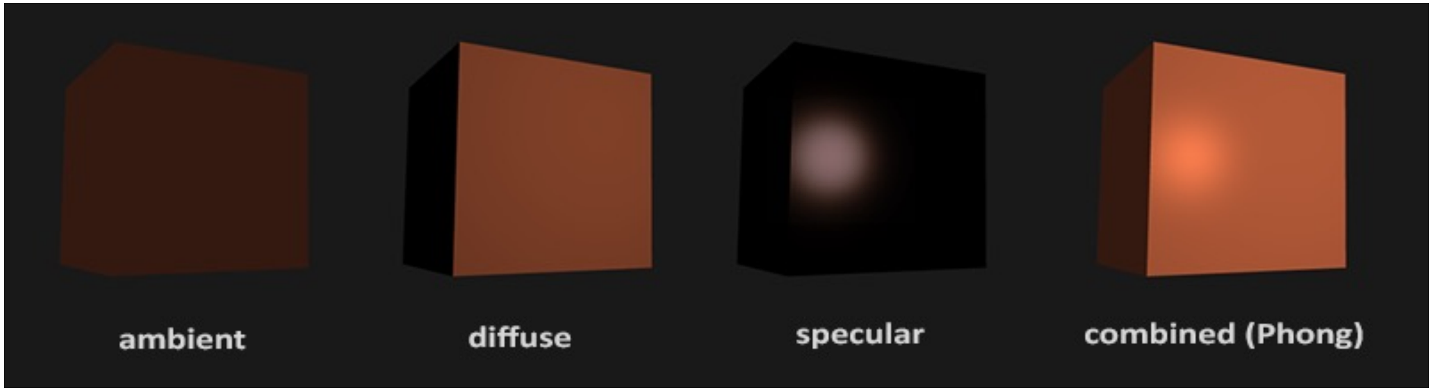
- A light source originating from a zero volume point and direct to the scene
  - Direction : the light is focused on
  - Cutoff : angle that defines light cone
  - Exponent : Concentration of the light

  (Brightest around the center)



Illumination Model in OpenGL

- Illumination model expresses the components of light
  "reflected from" or
  "transmitted through"
  (refracted or scattered) a surface
- We will deal with three basic lit components
  - Ambient
  - Diffuse
  - Specular



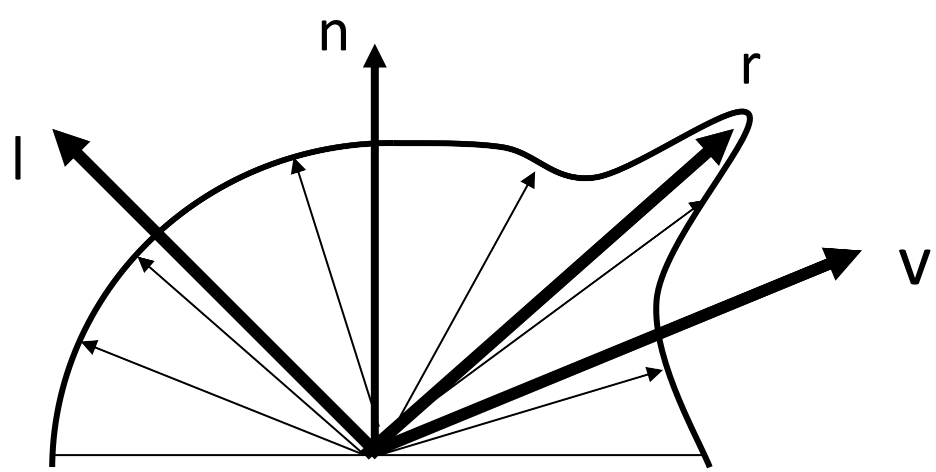
https://learnopengl.com/Lighting/Basic-Lighting

Phong Illumination Model

- Phong illumination model is combination of
  - Ambient i_{amb}+ Diffuse i_{diff} + Specular terms i_{sepc}
  - Developed by Bui Tuong Phong at Univ. Utah 1973

$$\mathbf{I} = k_a i_a + k_d i_d (\mathbf{n} \bullet \mathbf{l}) + k_s i_s (\mathbf{r} \bullet \mathbf{v})^{m_{shi}}$$

  - k_a k_d k_s are material properties having RGB components



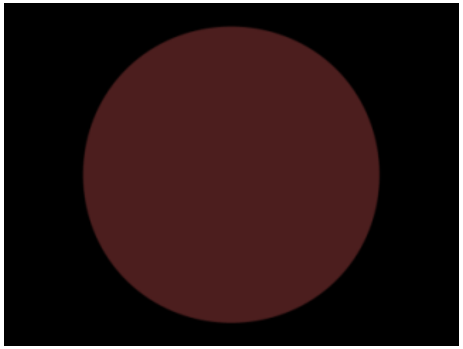
Ambient Lighting

- Light incident to surface is not only along direct path from light sources. Many inter-reflections are modeled as a lumped omnidirectional source

  → Indirect lighting (global illumination)

- Ambient light approximate indirect illumination using a constant intensity from all directions
- Ambient lights in OpenGL

$$\mathbf{i}_{amb} = \mathbf{m}_{amb} \otimes \mathbf{s}_{amb}$$



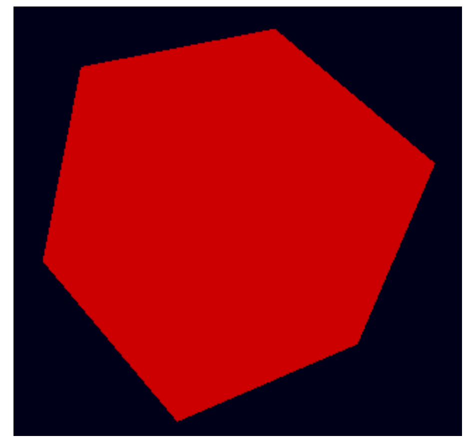
$\mathbf{m}_{amb}$ is the color of the object
$\mathbf{s}_{amb}$ is the color of the light source

Ambient Lighting

Fragment shader:

```
#version 330 core

out vec4 color;

const vec3 lightColor = vec3(1, 1, 1);
const vec3 objectColor = vec3(1, 0, 0);

void main() {
    float ambientStrength = 0.8;
    vec3 ambient = ambientStrength * lightColor;

    vec3 result = ambient * objectColor;
    color = vec4(result, 1.0);
}
```

Result:

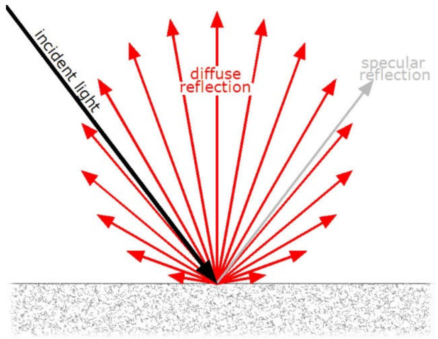


$$\mathbf{i}_{amb} = \mathbf{m}_{amb} \otimes \mathbf{s}_{amb}$$

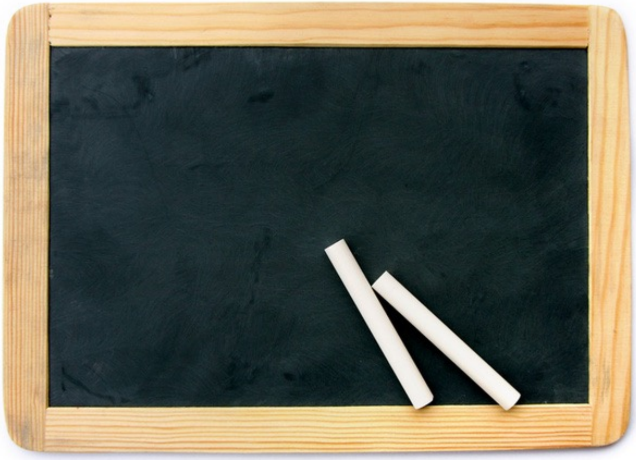
$\mathbf{m}_{amb}$ is the color of the object

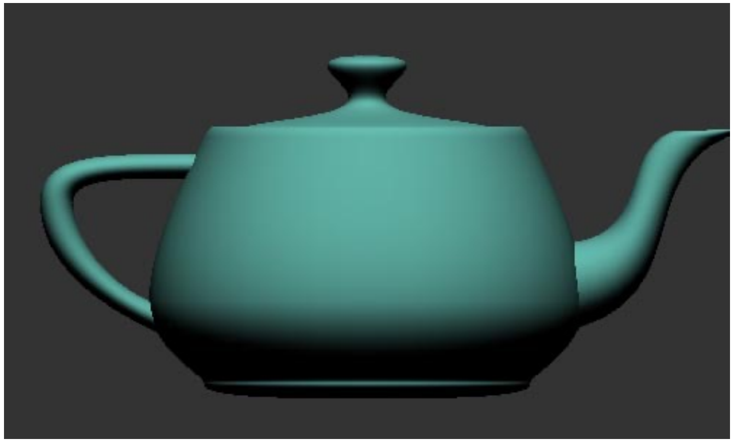
$\mathbf{s}_{amb}$ is the color of the light source

Diffuse



- Diffuse
  - Incident light is reflected into all directions
    - Photons are scattered equally in all directions
  - Diffusely reflected light is typically for dull, matte surface such a paper, chalk or chalkboard

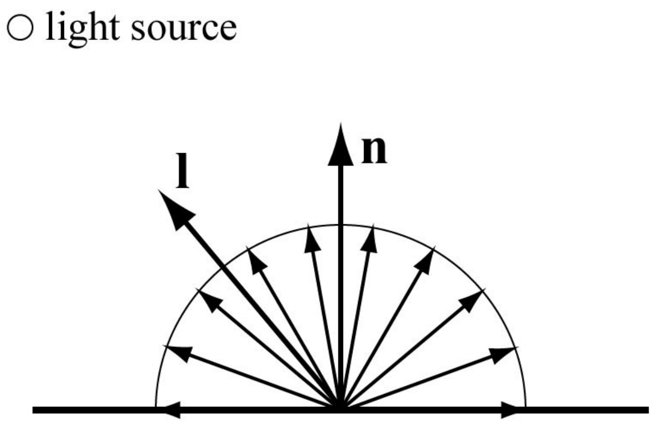




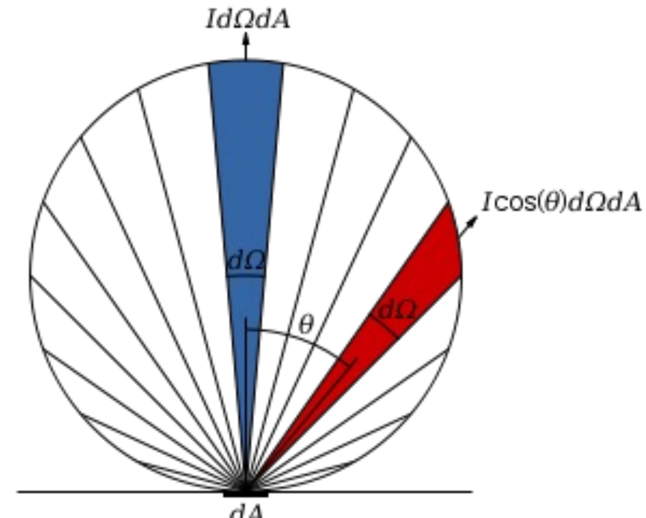
[Image from real time rendering book]

Diffuse reflection

- Component of diffuse reflection is based on Lambert's law $i_{diff} = \mathbf{n} \cdot \mathbf{l} = \cos\theta$
  - radiant intensity reflected from a fully diffuse surface is proportional to the angle between light direction **l** and surface normal **n**



[Image from real time rendering book]



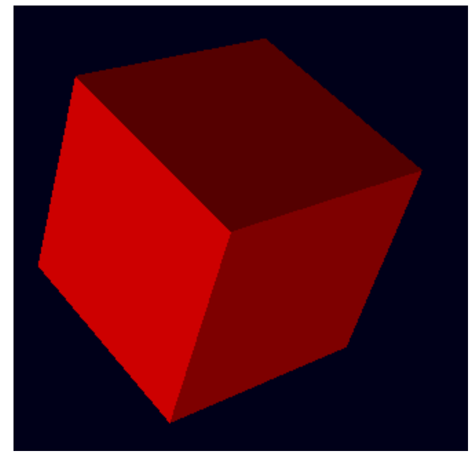
[Image from wikipedia]

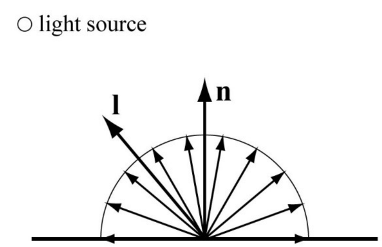
Diffuse Lighting

Fragment shader:

```
#version 330 core

out vec4 color;

in vec3 fragNormal;

const vec3 lightDir = vec3(0.25, 0.25, -1);
const vec3 lightColor = vec3(1, 1, 1);
const vec3 objectColor = vec3(1, 0, 0);

void main() {
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;

    vec3 norm = normalize(fragNormal);
    vec3 lightDir = normalize(-lightDir);

    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    vec3 result = (ambient + diffuse) * objectColor;

    color = vec4(result, 1.0);
}
```

Result:





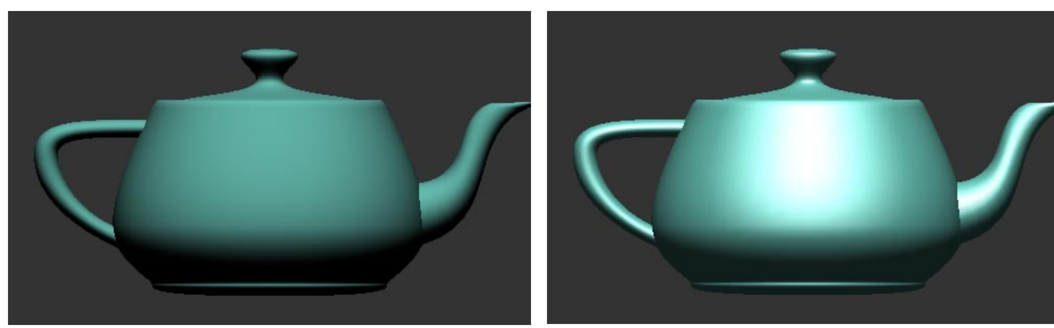
$$i_{diff} = \mathbf{n} \cdot \mathbf{l} = \cos\theta$$

Specular reflection

- Make a surface look shiny by creating highlights
  - Highlight visualize surface curvature
  - Highlight is determined by location of light and view

  →Shape from shading (computer vision)
- Diffuse vs Specular
  - Deals with reflection into a dominant direction causing highlights effect on the surface
  - Produce shiny spot on the surface such as billiard ball

http://www.joshenreborn.com/2013/04/against-all-odds-create.html



[Image from real time rendering book]

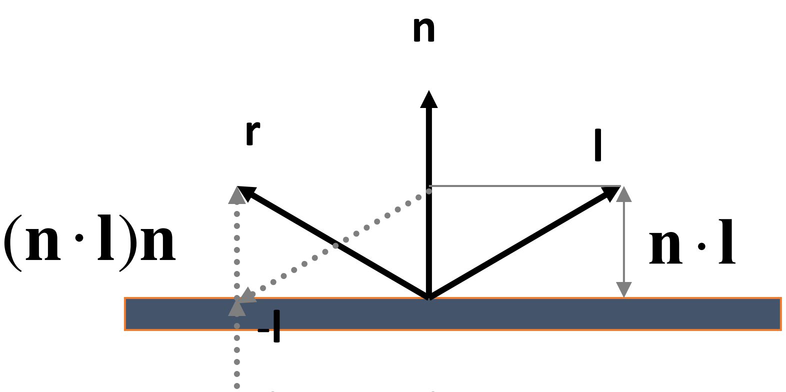
Phong Model: Specular reflection

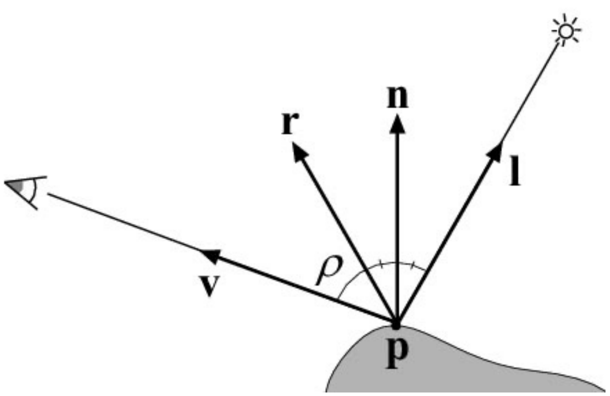
- For shiny surface, incident photons tend to bounce off in the reflection direction *r*

$$i_{spec} = (\mathbf{r} \cdot \mathbf{v})^{m_{shi}} = (\cos \rho)^{m_{shi}}$$

  - If r is closer to v, specularity gets stronger → view dep.

- **r** needs to be computed
  - $\mathbf{r} = 2(\mathbf{n} \cdot \mathbf{l})\mathbf{n} - \mathbf{l}$, n and l is normalized
  - If $(\mathbf{n} \cdot \mathbf{l}) < 0$, surface faces away from light → no effect



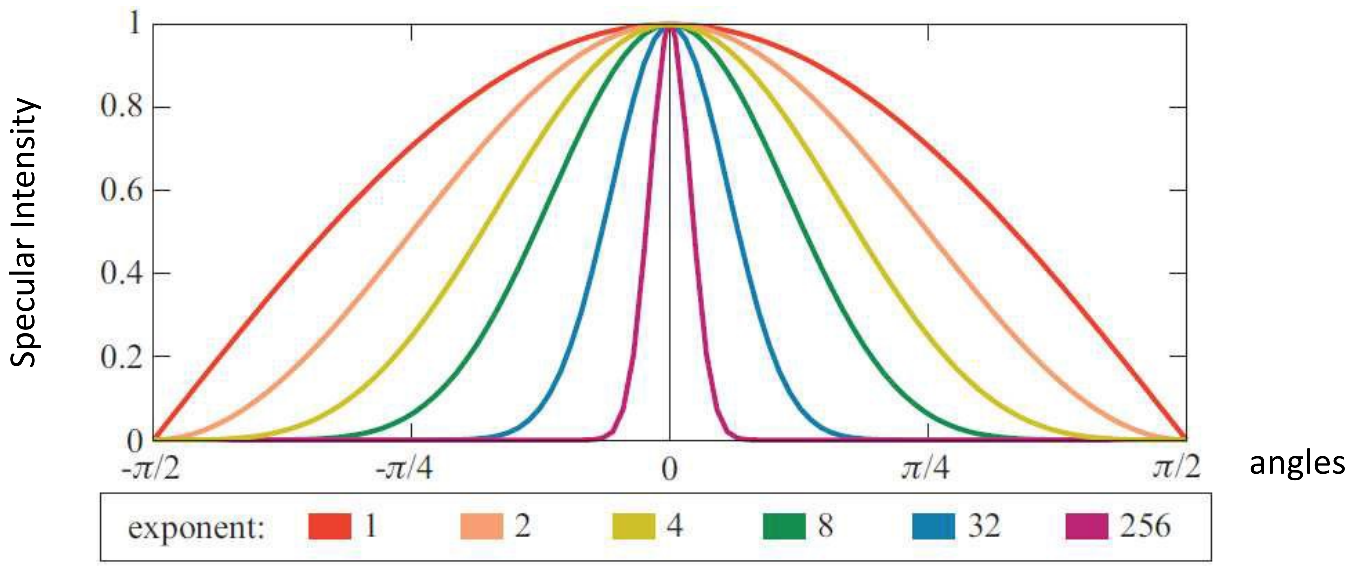


[Image from real time rendering book]

https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/reflection-refraction-fresnel

Shininess control

- m_{shi} controls shininess
  - If m_{shi} is 1, cosine curve is produced between two vectors (**r**, **v**) or (**n**, **h**)
  - When m_{shi} gets larger, small but strong highlight
  - Look reasonable but may not accurate



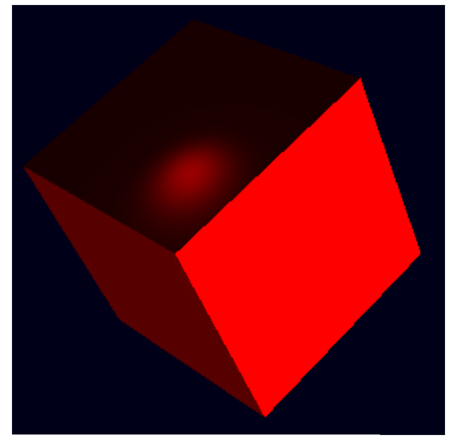
[Image from real time rendering book]

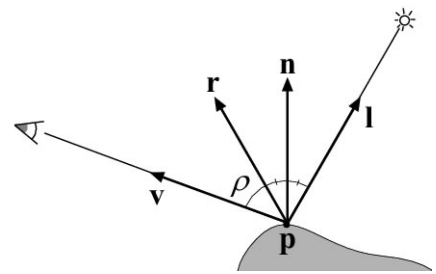
Specular Lighting

Fragment shader:

```
#version 330 core

out vec4 color;

in vec3 fragPosition;
in vec3 fragNormal;

const vec3 lightDir = vec3(0.25, 0.25, -1);
const vec3 lightColor = vec3(1, 1, 1);
const vec3 objectColor = vec3(1, 0, 0);

void main() {
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;

    vec3 norm = normalize(fragNormal);
    vec3 lightDir = normalize(-lightDir);

    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    float specularStrength = 0.5;
    vec3 reflectDir = reflect(-lightDir, norm);
    vec3 viewDir = normalize(-fragPosition);

    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
    vec3 specular = specularStrength * spec * lightColor;

    vec3 result = (ambient + diffuse + specular) * objectColor;

    color = vec4(result, 1.0);
}
```

Result:





$$i_{spec} = (\mathbf{r}\cdot\mathbf{v})^{m_{shi}} = (\cos\rho)^{m_{shi}}$$