
Computer Graphics

8 - Lighting & Shading

Yoonsang Lee
Spring 2019

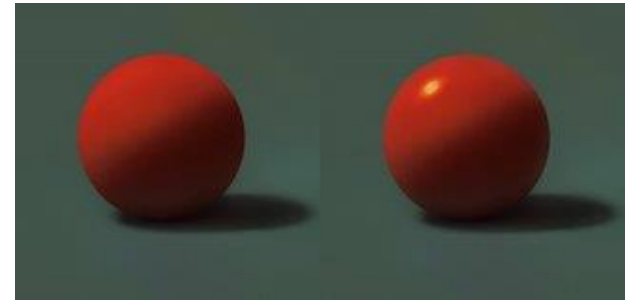
Topics Covered

- Reflection of Light
- Phong Illumination Model
- Shading
 - Face / Vertex Normal
 - Flat / Goraud / Phong Shading
- Lighting & Shading in OpenGL

Reflection of Light

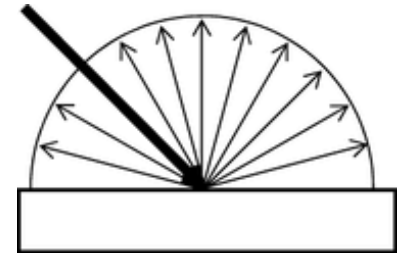
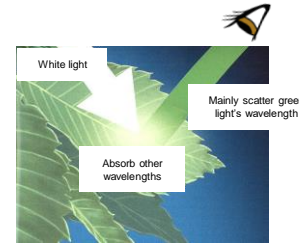
Reflection of Light

- Light can be absorbed(흡수), emitted(발산), scattered(산란), reflected(반사), or refracted(굴절) by objects.
- Scattering and reflection are the main factors in the visual characteristics of the object surface.
 - such as surface color, highlight on surface
- Types of reflection:
 - Diffuse reflection
 - Specular reflection (ideal & non-ideal)
 - In computer graphics, both scattering and reflection are often referred to as "reflection"



Diffuse Reflection

- : Scattering specific light spectrum in all direction
- → Determines surface color
- **View-independent**



strongly scatters
magenta's wavelengths



scatters
wavelengths for all
colors



scatters no colors
(absorbs all colors)

Diffuse Reflection - Lambert's Cosine Law

- The **reflected energy** from a small surface area is proportional to the **cosine of the angle between incident light direction** and the **surface normal**

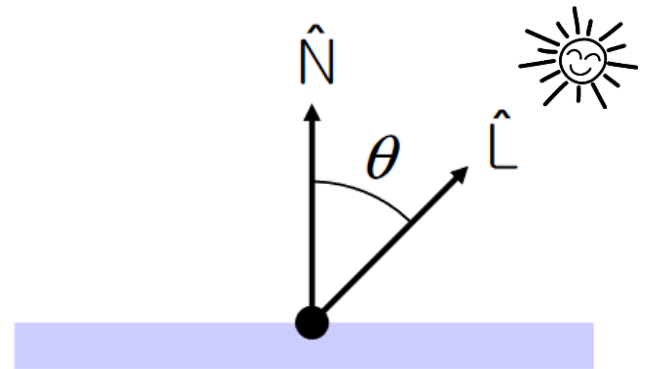
$$\begin{aligned} I_{reflected} &= I_{incident} \cos\theta \\ &= I_{incident} (\hat{\mathbf{N}} \cdot \hat{\mathbf{L}}) \end{aligned}$$

$I_{reflected}$ intensity of reflected ray

$I_{incident}$ intensity of incident ray

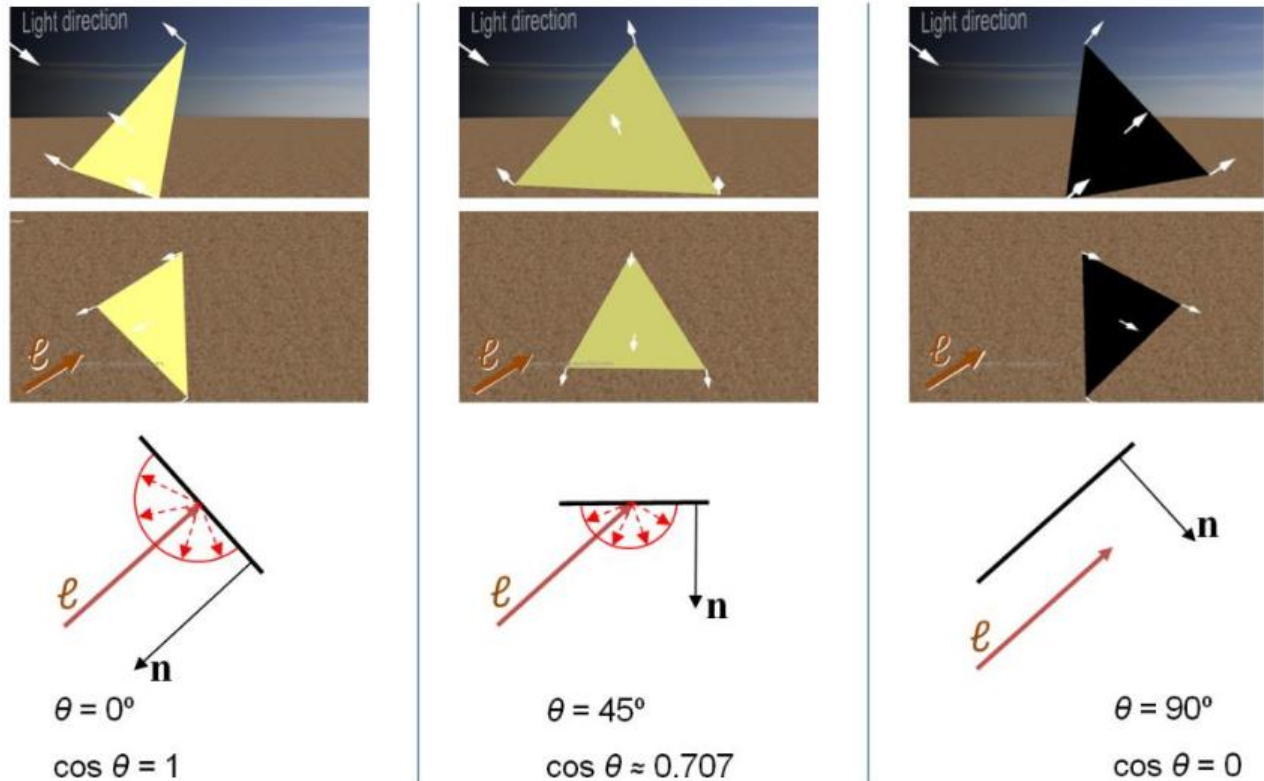
$\hat{\mathbf{N}}$ normal to the reflection surface at the point of the incidence

$\hat{\mathbf{L}}$ normalized light direction vector



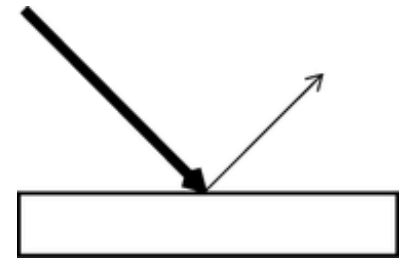
Diffuse Reflection - Lambert's Cosine Law

▶ Visualization of Lambert's law in 2D



Ideal Specular Reflection

- : Mirror-like reflection of light from smooth, polished surface
- → Generate mirrored images
- **View-dependent**



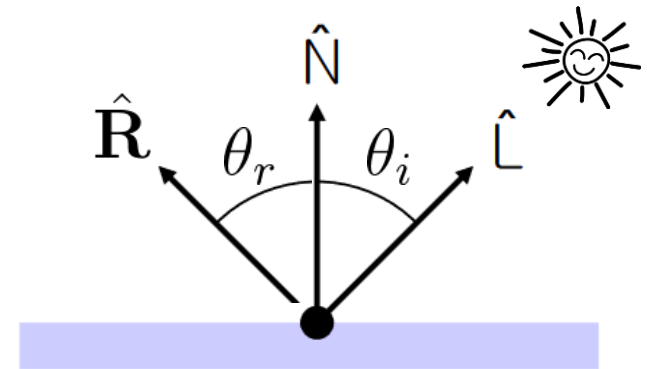
Ideal Specular Reflection - Laws of Reflection (Special case of Snell's Law)

- $\hat{\mathbf{N}}, \hat{\mathbf{L}}, \hat{\mathbf{R}}$ lie in the same plane
- $\theta_r = \theta_i$
- ($\hat{\mathbf{L}}$ and $\hat{\mathbf{R}}$ are on the opposite sides of $\hat{\mathbf{N}}$)

$\hat{\mathbf{N}}$ normal to the reflection surface at the point of the incidence

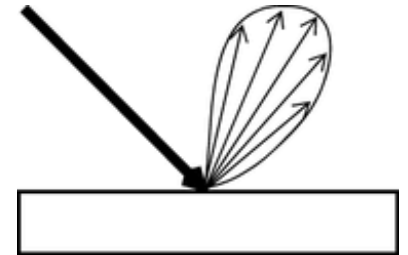
$\hat{\mathbf{L}}$ normalized incident ray direction vector

$\hat{\mathbf{R}}$ normalized reflected ray direction vector



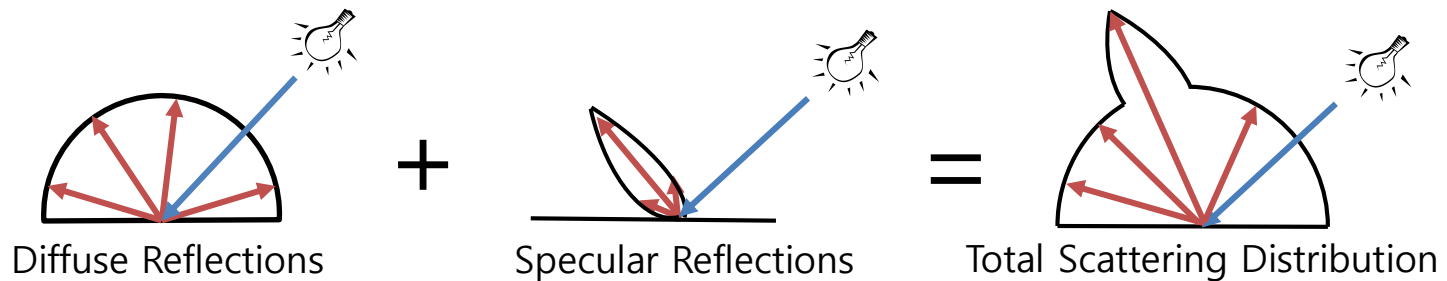
Non-Ideal Specular Reflection (a.k.a. Glossy Reflection)

- : Reflection on shiny & glossy surface, but not as smooth as a mirror
- Reflected rays are “spread out” due to surface roughness
- → Generate bright highlights
- **View-dependent**



Reflection of General Materials

- Many materials' surface have both diffuse reflection and specular reflection.



Quiz #1

- Go to <https://www.slido.com/>
- Join #cg-hyu
- Click “Polls”

- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

Phong Illumination Model

Lighting (or Illumination)

- **Lighting (or Illumination)**: Process of computing effects of lights
- → Computing surface color and highlights of objects.

Phong Illumination Model

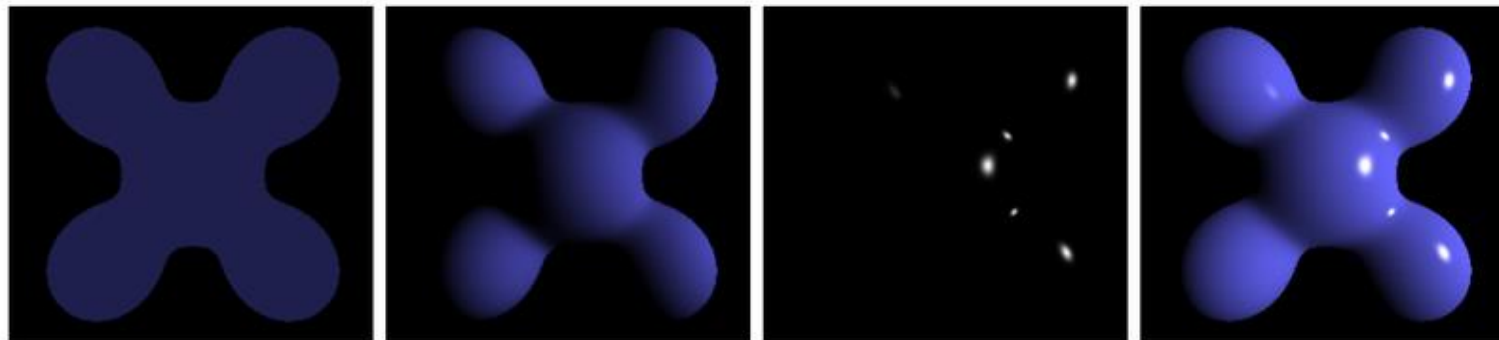
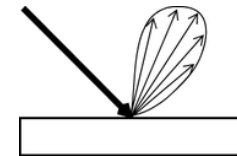
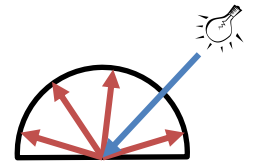
- One of the most commonly used “classical” illumination models in computer graphics
 - Empirical model, not physically based



Bùi Tường Phong
(1942 – 1975)

Phong Illumination Model

- Three components:
- **Ambient**
 - Non-specific constant global lighting
 - Crudest approximation for indirect lighting
- **Diffuse**
 - Color of object under normal conditions using Lambert's model
- **Specular**
 - Highlights on shiny objects
 - Approximation for glossy reflection using $\cos^n(\alpha)$



Ambient + Diffuse + Specular = Phong Reflection

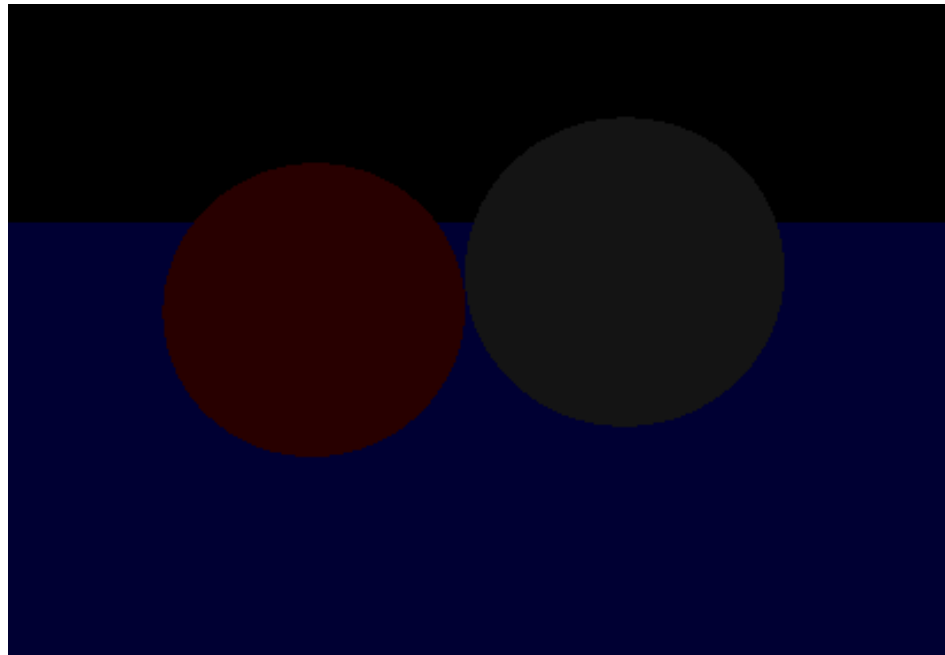
Ambient Light

$$I = k_a C_a$$

- C_a = intensity of ambient light
- k_a = ambient reflection coefficient
- Really 3 equations for 3 C_a s! (C_a^r , C_a^g , C_a^b for Red, Green, Blue)

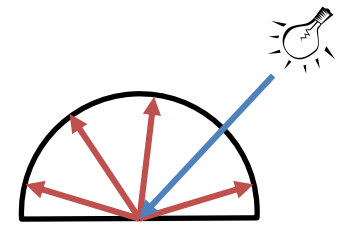
Total Illumination

$$I = k_a C_a$$



Diffuse Light

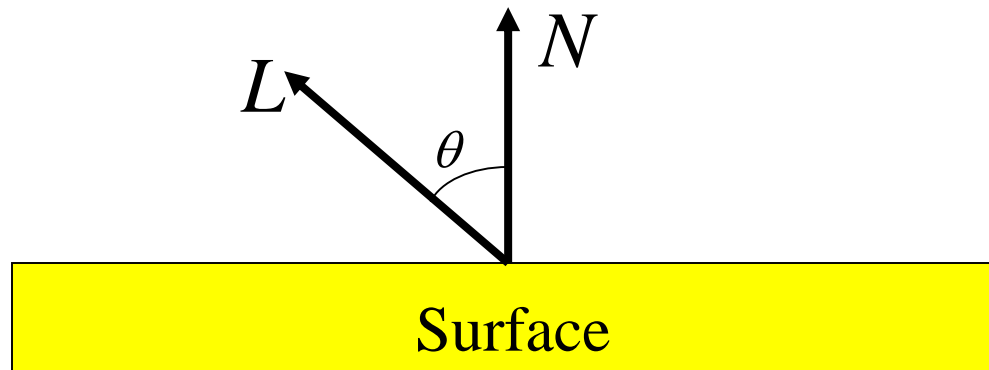
Lambert's Cosine Law



$$I = C_d k_d \cos(\theta) = C_d k_d (L \cdot N)$$

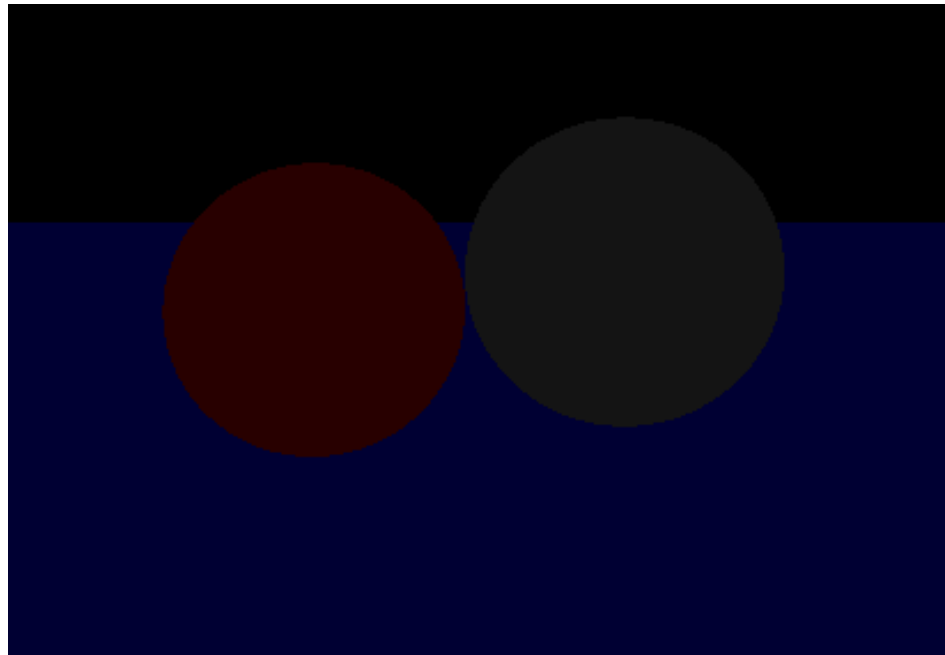
- C_d = intensity of light source (actually 3 equations for C_d^r, C_d^g, C_d^b)
- k_d = diffuse reflection coefficient
- θ = angle between normal and direction to light

$$\cos(\theta) = L \cdot N$$



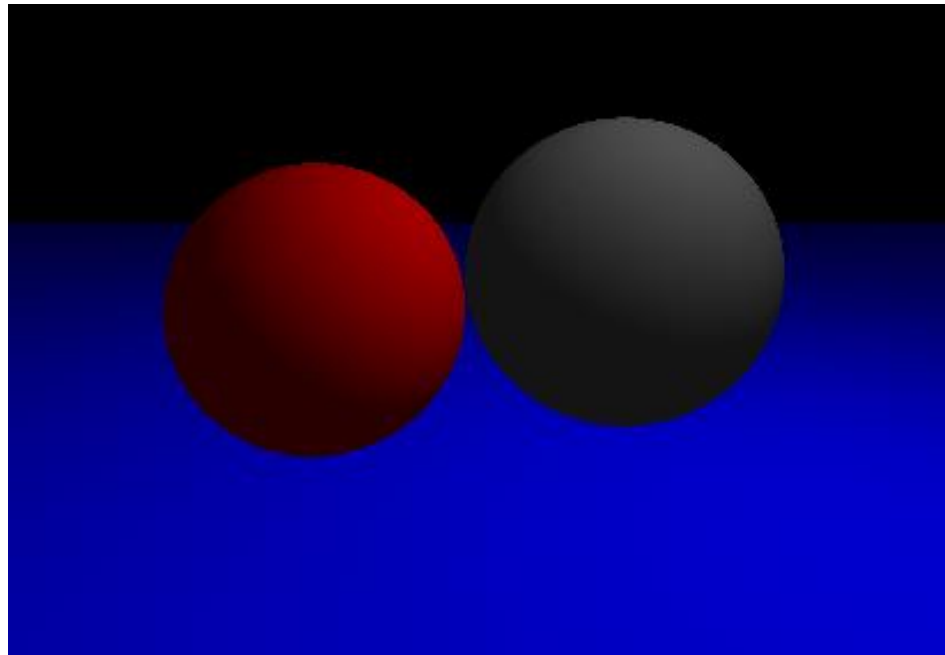
Total Illumination

$$I = k_a C_a$$



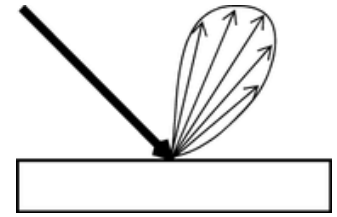
Total Illumination

$$I = k_a C_a + k_d C_d (L \cdot N)$$



Specular Light

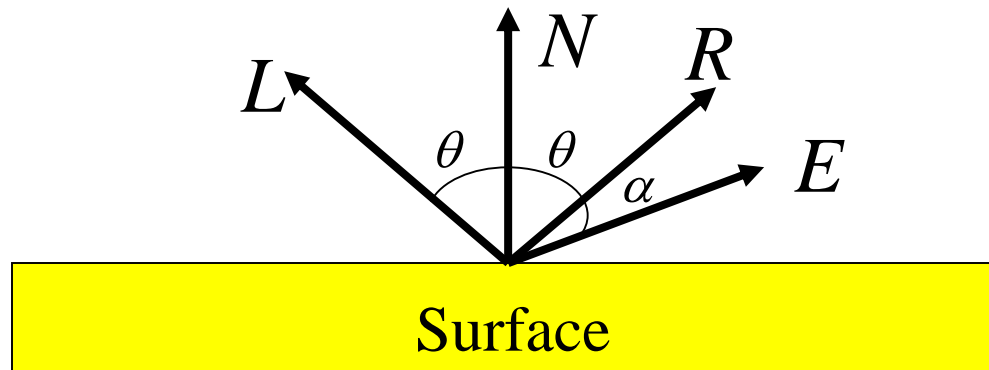
approximation for
glossy reflection



$$I = C_s k_s \cos^n(\alpha) = C_s k_s (R \cdot E)^n$$

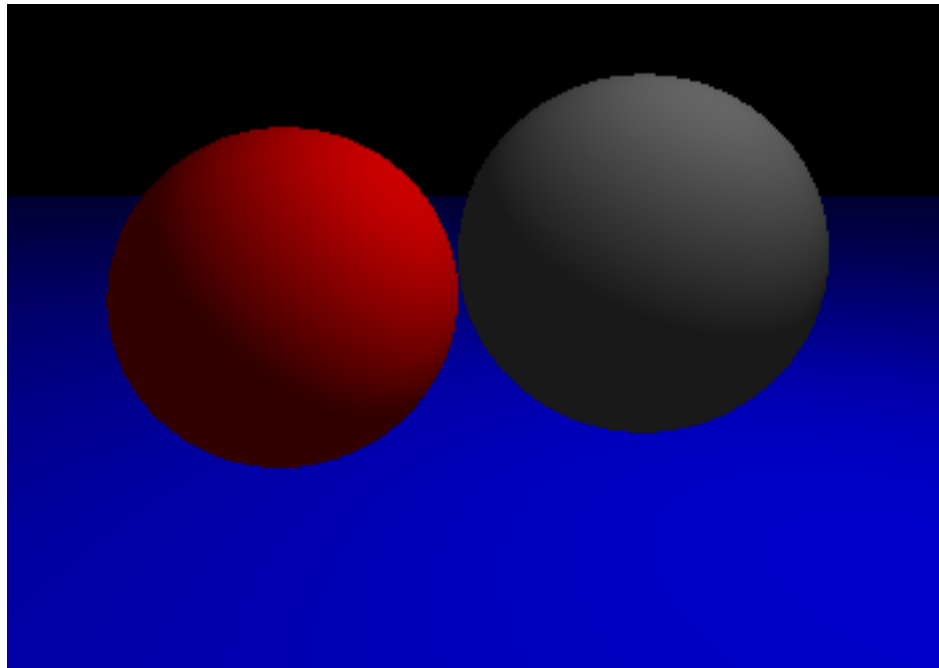
- C_s = intensity of light source (actually 3 eq: C_s^r, C_s^g, C_s^b)
- k_s = specular reflection coefficient
- α = angle between reflected vector (R) and eye (E)
- n = shininess coefficient

$$\cos(\alpha) = R \cdot E$$



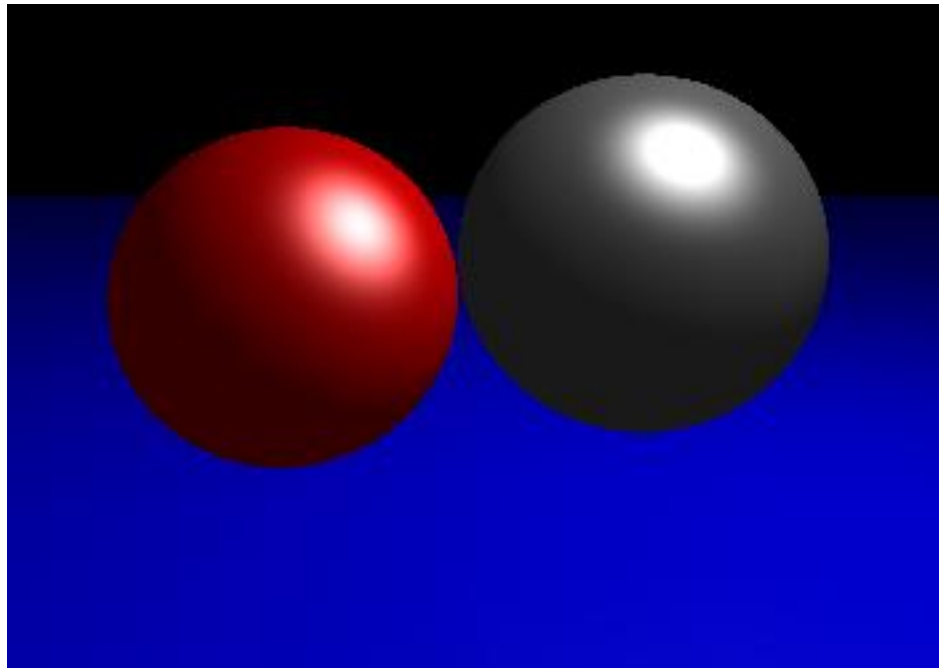
Total Illumination

$$I = k_a C_a + k_d C_d (L \cdot N)$$



Total Illumination

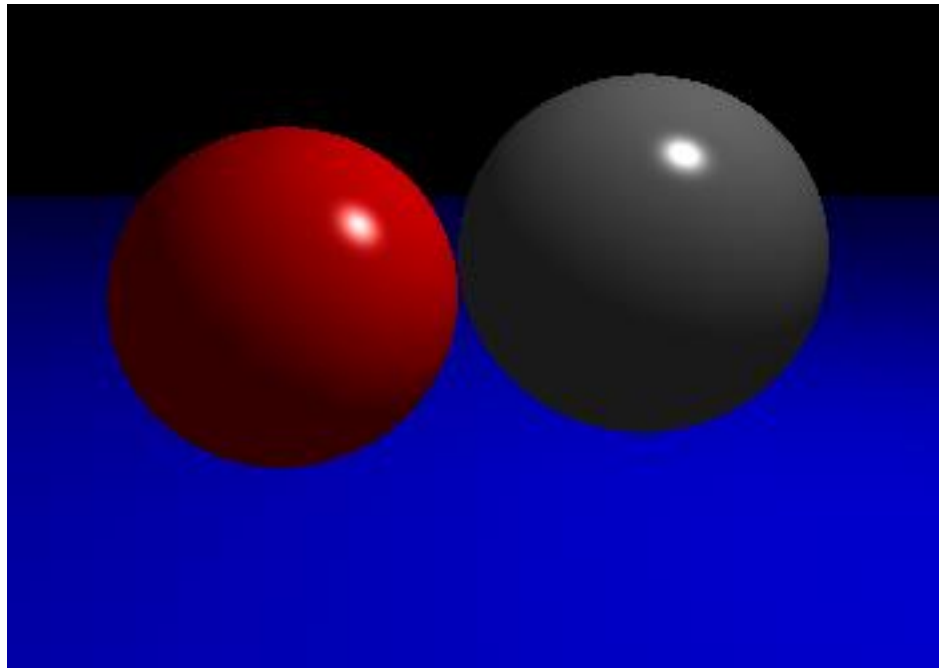
$$I = k_a C_a + k_d C_d (L \cdot N) + k_s C_s (R \cdot E)^n$$



$$n=5$$

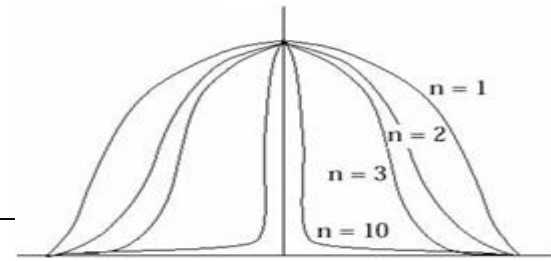
Total Illumination

$$I = k_a C_a + k_d C_d (L \cdot N) + k_s C_s (R \cdot E)^n$$



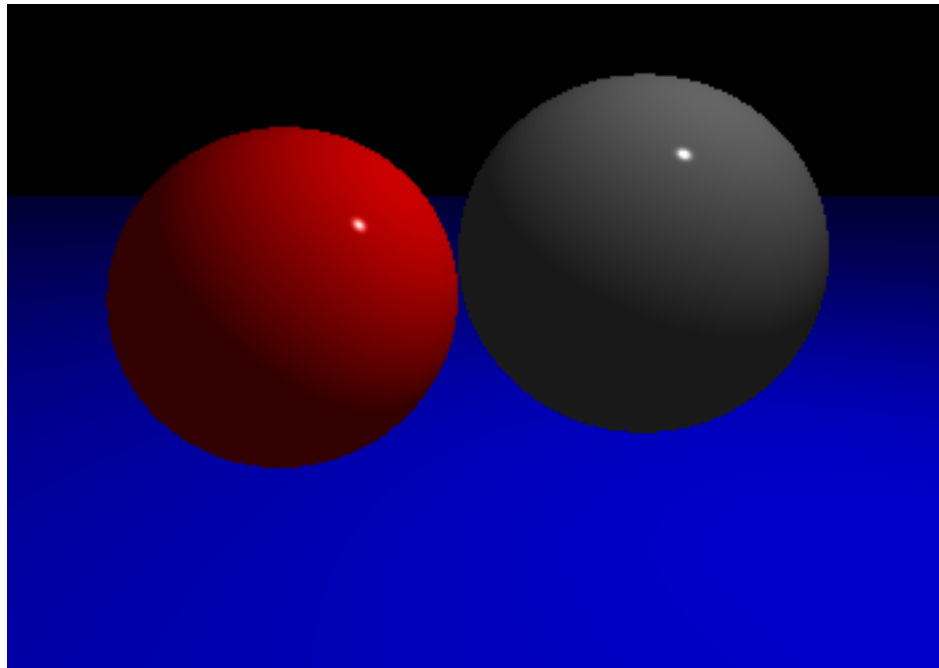
$$n = 50$$

Total Illumination



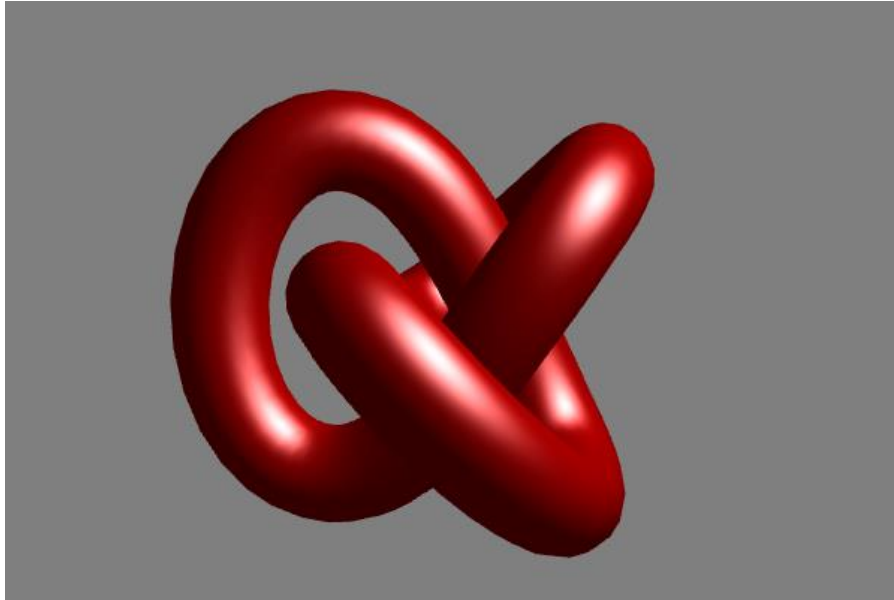
Specular falloff of $(\cos \delta)^n$

$$I = k_a C_a + k_d C_d (L \cdot N) + k_s C_s (R \cdot E)^n$$



$n = 500$

[Practice] Phong Illumination Demo



<http://www.cs.toronto.edu/~jacobson/phong-demo/>

- First set the value of the first drop down box to “Phong Shading”
- Try to change
 - reflection coefficient and color of ambient, diffuse, and specular
 - specular shininess
 - you can also change object type, light position and background color

Quiz #2

- Go to <https://www.slido.com/>
- Join #cg-hyu
- Click “Polls”

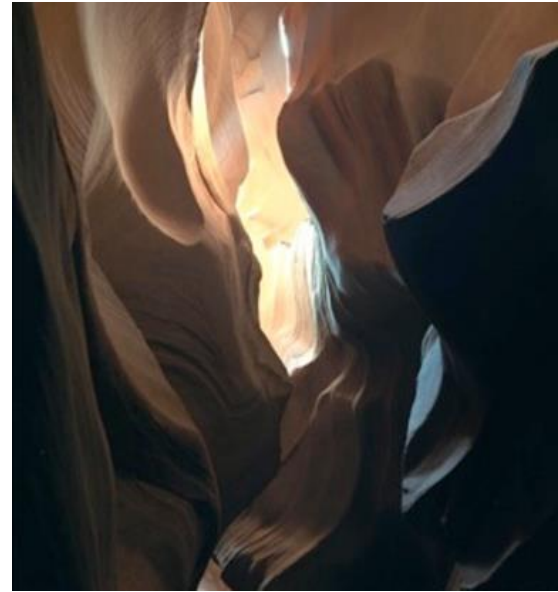
- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

Shading

Shading - General Meaning

- Variation in observed color across an object
 - Strongly affected by lighting



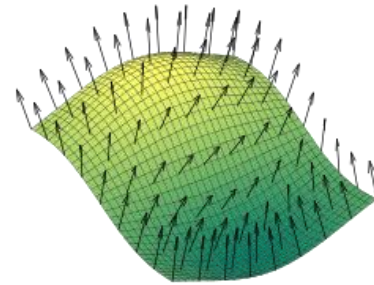
Shading - Meaning in Computer Graphics

- The process of determining **each pixel color in a polygon** based on a illumination model



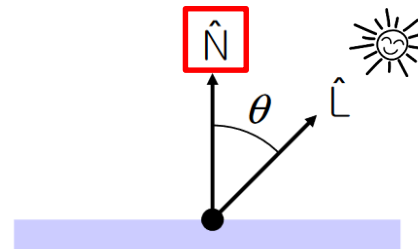
Surface Normal

- A vector that is perpendicular to the surface at a given point
 - A unit normal vector (of length 1) is generally used



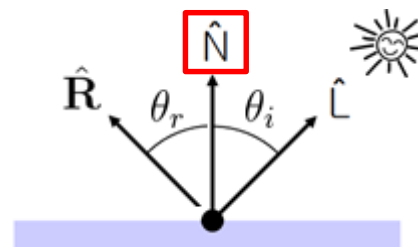
- Plays a key role in shading & illumination process

- Diffuse reflection
 - Lambert's Cosine Law



$$I_{reflected} = I_{incident} \cos \theta$$

- Specular reflection
 - Laws of Reflection



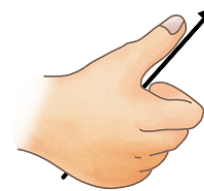
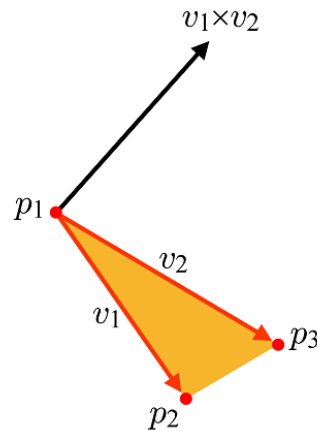
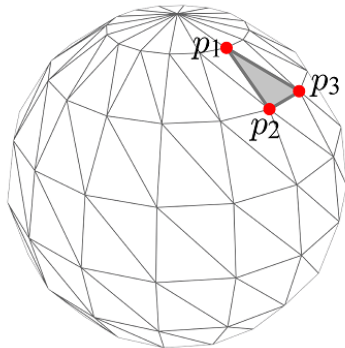
$$\theta_r = \theta_i$$

Face Normal

- How to get the surface normal of a polygonal face

The order does matter!

- The normal of a triangle $\langle p_1, p_2, p_3 \rangle$ is computed as $v_1 \times v_2$
 - v_1 is the vector connecting p_1 and p_2 , v_2 connects p_1 and p_3

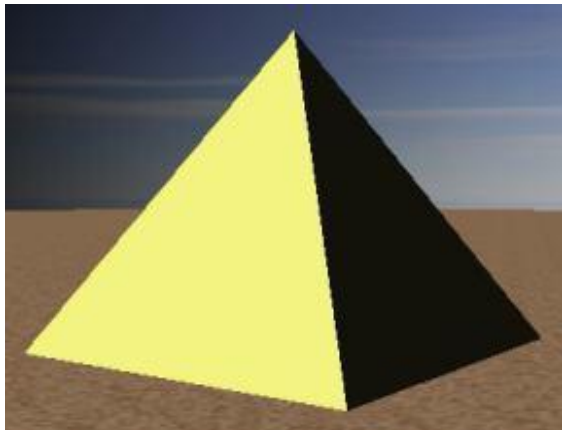


$$\frac{v_1 \times v_2}{\|v_1 \times v_2\|}$$

- That's why we need **counterclockwise** vertex ordering
 - The direction of a face normal determines “outside” of the face

Flat Shading

- Use a single face normal for each polygon
- Calculate color (by illumination) once per polygon
 - Typically use center of polygon
- Fast, but not very desirable for curved shapes
 - Even if we increase the number of polygons, it's still “faceted“

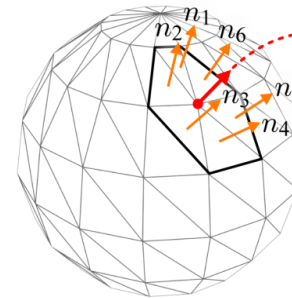


Smooth Shading

- Shading methods for curved shapes
 - Smooth color transition between two adjacent polygons



- Two methods:
 - Gouraud shading
 - Phong shading



$$\frac{n_1+n_2+n_3+n_4+n_5+n_6}{\|n_1+n_2+n_3+n_4+n_5+n_6\|}$$

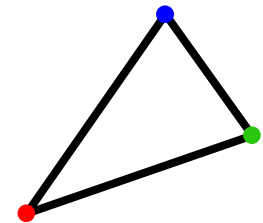
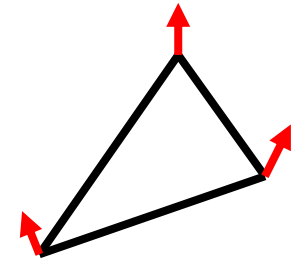
- Use a vertex normal for each vertex
 - For smooth shading, a vertex normal is commonly set to the average of normals of all faces sharing the vertex.

Gouraud Shading



Henri Gouraud
(1944~)

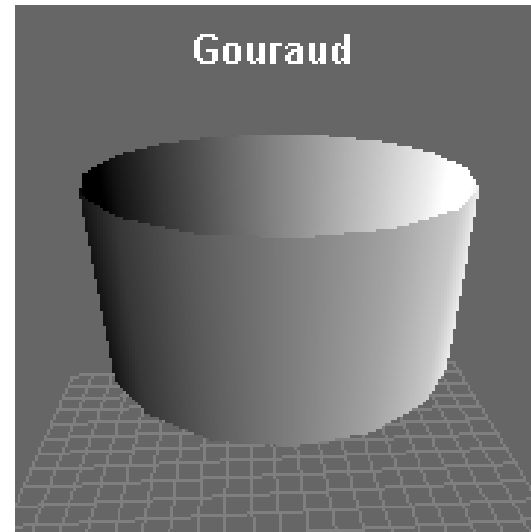
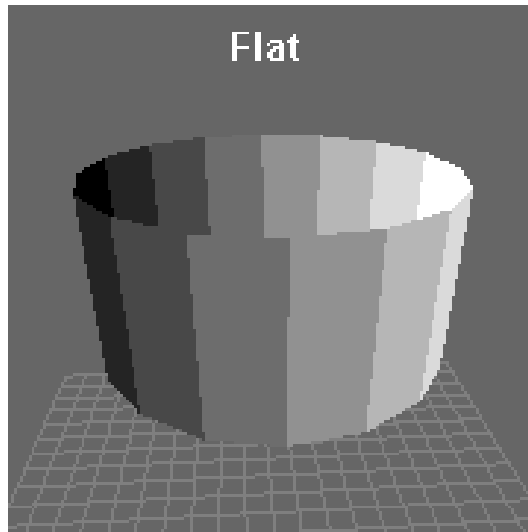
- Use a single vertex normal for each vertex
- Calculate color (by illumination) at each vertex
- Interpolate colors from vertices across polygon
 - Barycentric interpolation



See more for barycentric interpolation:

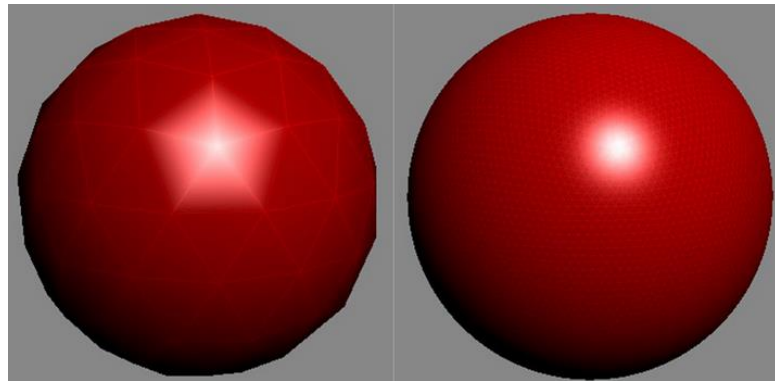
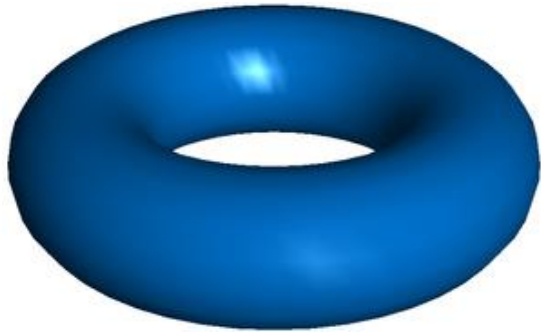
<https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates>

Gouraud Shading



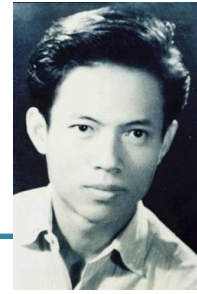
Gouraud Shading

- Problem: poor specular highlight
 - Specular highlights may be distorted or averaged away altogether



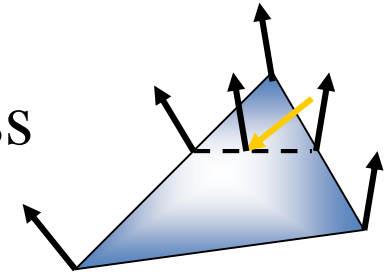
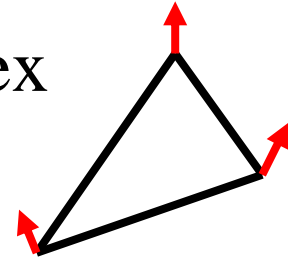
Higher polygon count
reduces this artifact

Phong Shading

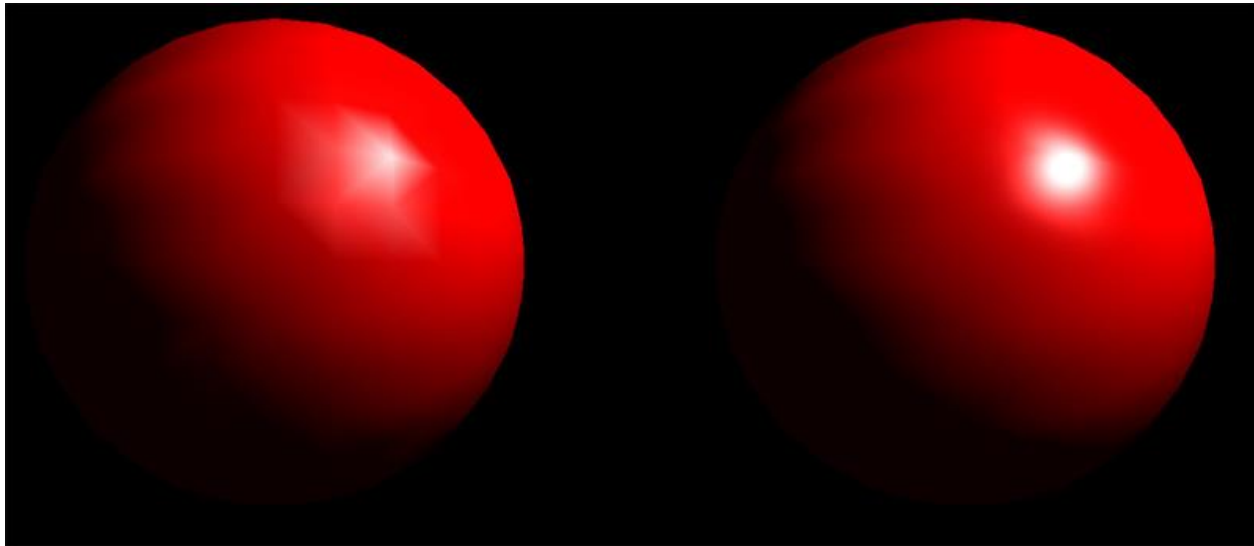


Bùi Tường Phong
(1942 – 1975)

- Use a single vertex normal for each vertex
- Interpolate normals from vertices across polygon
- Calculate color (by illumination) at each pixel in polygon



Phong Shading



Gouraud shading

Phong shading

Phong Shading

- Captures highlights much better
 - The interpolated normal at each interior pixel is more accurate representation of true surface normal at each point
 - Higher quality, but needs more computation
- Not to be confused with Phong's illumination model (developed by the same person)

[Practice] Online Shading Demos

- Flat & Gouraud shading
 - <http://math.hws.edu/graphicsbook/demos/c4/smooth-vs-flat.html>
- Gouraud & Phong shading
 - <http://www.cs.toronto.edu/~jacobson/phong-demo/>

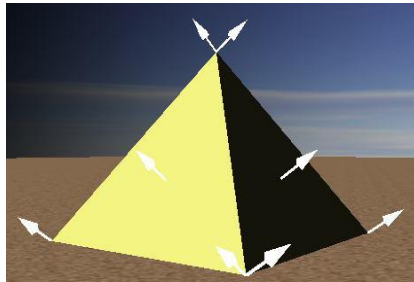
Lighting & Shading in OpenGL

To do Lighting & Shading in OpenGL,

- First, you need to set vertex normal.
- Recall from 2-IntroNumPyOpenGL slides, a vertex has these attributes:
 - Vertex coordinate : specified by `glVertex*()`
 - Vertex color : specified by `glColor*()`
 - **Normal vector : specified by `glNormal*()`**
 - Texture coordinate : specified by `glTexCoord*()`

Shading in OpenGL

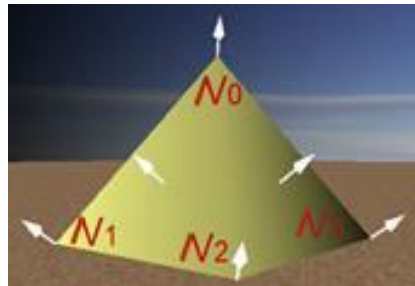
- The shading method is determined by the vertex normal vectors you specify.
- Flat shading: Set each vertex normal to the face normal the vertex belongs to.



The normal at a vertex is the same as the plane normal. Therefore, each vertex has as many normals as the number of planes it belongs

Shading in OpenGL

- Gouraud shading: Set each vertex normal to the average of normals of all faces sharing the vertex.



Only one vertex normal per vertex; average of face normals of the faces the vertex is part of

- Phong shading is not available in legacy OpenGL.

Setting Vertex Normals in OpenGL

- You can specify normals using `glNormal*()` or a vertex array

```
glBegin(GL_TRIANGLES)

glNormal3f(0,0,1) # v0,v2,v1,v0,v3,v2 normal
glVertex3f(-1, 1, 1) # v0 position
glVertex3f( 1, -1, 1) # v2 position
glVertex3f( 1, 1, 1) # v1 position

glVertex3f(-1, 1, 1) # v0 position
glVertex3f(-1, -1, 1) # v3 position
glVertex3f( 1, -1, 1) # v2 position

glNormal3f(0,0,-1)
glVertex3f(-1, 1, -1) # v4
glVertex3f( 1, 1, -1) # v5
glVertex3f( 1, -1, -1) # v6

glVertex3f(-1, 1, -1) # v4
glVertex3f( 1, -1, -1) # v6
glVertex3f(-1, -1, -1) # v7

# ...
], 'float32')
```

```
varr = np.array([
    (0,0,1),          # v0 normal
    (-1, 1, 1),      # v0 position
    (0,0,1),          # v2 normal
    ( 1, -1, 1),     # v2 position
    (0,0,1),          # v1 normal
    ( 1, 1, 1),      # v1 position

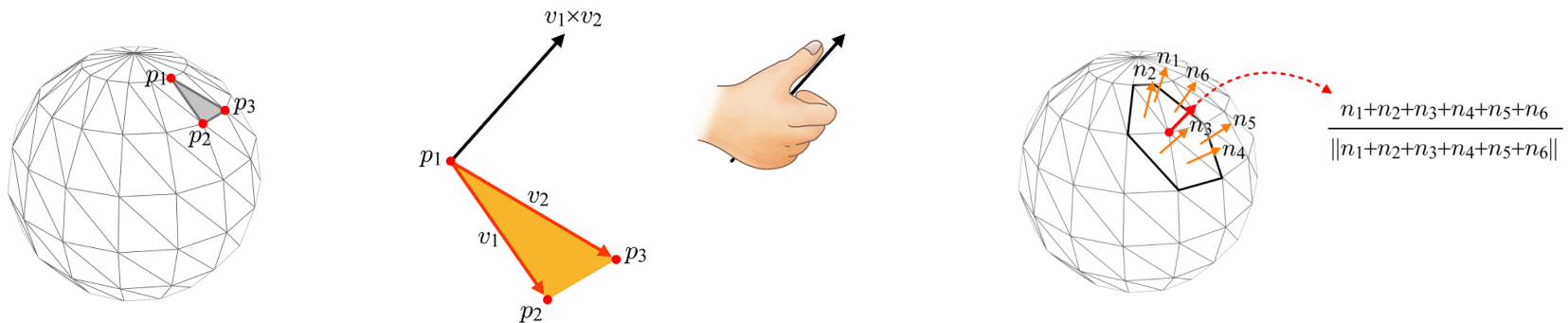
    (0,0,1),          # v0 normal
    (-1, 1, 1),      # v0 position
    (0,0,1),          # v3 normal
    (-1, -1, 1),     # v3 position
    (0,0,1),          # v2 normal
    ( 1, -1, 1),     # v2 position

    (0,0,-1),
    (-1, 1, -1),     # v4
    (0,0,-1),
    ( 1, 1, -1),     # v5
    (0,0,-1),
    ( 1, -1, -1),    # v6

    # ...
], 'float32')
```

Setting Vertex Normals in OpenGL

- You can hard-code normals like prev. page
- or compute normals from vertex positions



- or read normals from a model file such as .obj files (most common case)

Lighting in OpenGL

- Lighting in legacy OpenGL is too restrictive.
 - Only Blinn-Phong illumination model is available.
- **glEnable(GL_LIGHTING)**
 - Enable lighting
- **glEnable(GL_LIGHT0)**
 - Enable 0th light. You can use eight lights in legacy OpenGL (GL_LIGHT0 ~ GL_LIGHT7)

glLightfv()

- **glLightfv(light, pname, param)**
 - **light**: The light to assign
 - GL_LIGHT0 ~ GL_LIGHT7
 - **pname, param**: light properties including light intensity for each color channel, etc

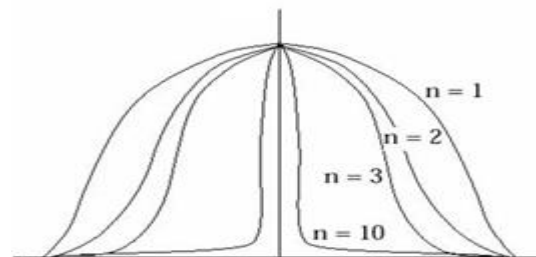
Pname	Def. Value (param)	Meaning
GL_AMBIENT	(0.0, 0.0, 0.0, 0.0)	ambient RGBA intensity of light (ranging from 0.0 to 1.0)
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	diffuse RGBA intensity of light
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	specular RGBA intensity of light
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(x, y, z, w) position of light w=0: directional light w=1: point light (homogeneous coordinates)

glMaterialfv()

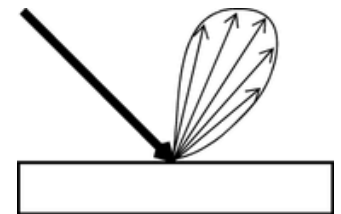
- **glMaterialfv(face, pname, param)**
 - **face**: The face type to assign
 - GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK
 - **pname, param**: material reflectance for each color channel
 - GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR
 - GL_AMBIENT_AND_DIFFUSE
 - GL_SHININESS: Specular exponent (shininess coefficient) (0 ~ 128)

$$I = C_s k_s \cos^n(\alpha)$$

A blue arrow points from the exponent n in the equation to the n parameter in the diagram to its right.



Specular falloff of $(\cos \delta)^n$



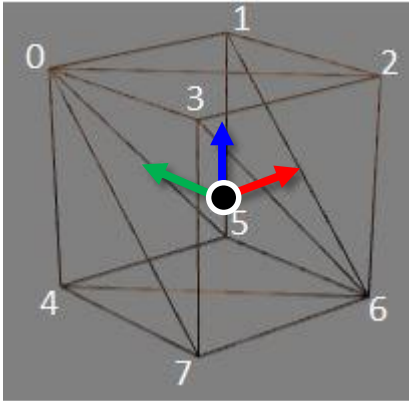
Good Settings for `glLightfv()` & `glMaterialfv()`

- `glLightfv()`
 - `GL_DIFFUSE` & `GL_SPECULAR`: Color of the light source
 - `GL_AMBIENT`: The same color, but at much reduced intensity (about 10%)
- `glMaterialfv()`
 - `GL_DIFFUSE` & `GL_AMBIENT`: Color of the object
 - `GL_SPECULAR`: White (1,1,1,1)
- Final polygon color is the sum of ambient, diffuse, specular components, each of which is formed by multiplying the `glMaterial` color by the `glLight` color.

Normals with Lighting

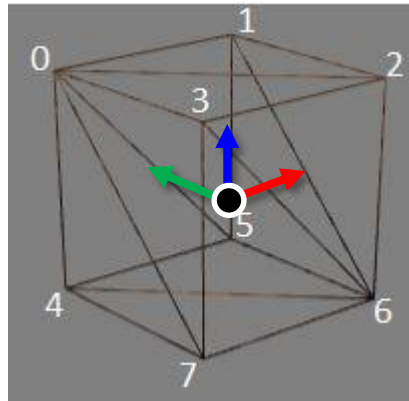
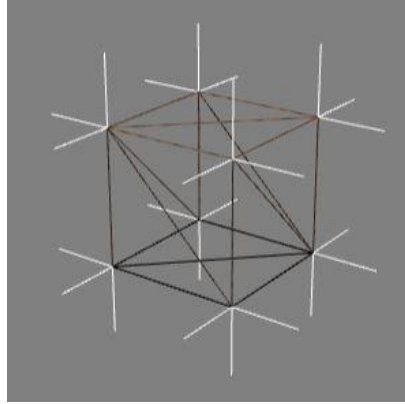
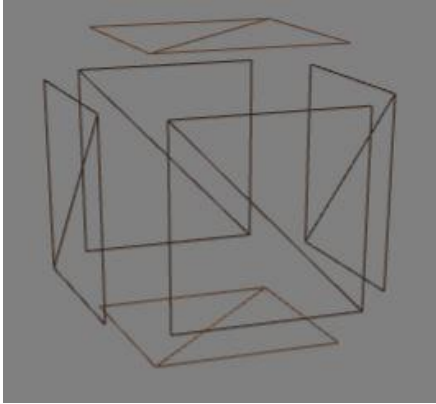
- In OpenGL, normal vectors should have *unit length*.
- Normal vectors are transformed by `GL_MODELVIEW` matrix, so they may not have unit length, especially if scaling are included.
- You need to use one of these:
 - `glEnable(GL_NORMALIZE)`
 - `glEnable(GL_RESCALE_NORMAL)`
 - To use this, normal vectors must be initially supplied as unit vectors.
 - More efficient

Example: a cube of length 2 again



vertex index	position
0	(-1 , 1 , 1)
1	(1 , 1 , 1)
2	(1 , -1 , 1)
3	(-1 , -1 , 1)
4	(-1 , 1 , -1)
5	(1 , 1 , -1)
6	(1 , -1 , -1)
7	(-1 , -1 , -1)

Normals of the Cube for Flat Shading



vertex index	position	normal
0	(-1, 1, 1)	(0,0,1)
2	(1, -1, 1)	(0,0,1)
1	(1, 1, 1)	(0,0,1)
0	(-1, 1, 1)	(0,0,1)
3	(-1, -1, 1)	(0,0,1)
2	(1, -1, 1)	(0,0,1)
4	(-1, 1, -1)	(0,0,-1)
5	(1, 1, -1)	(0,0,-1)
6	(1, -1, -1)	(0,0,-1)
4	(-1, 1, -1)	(0,0,-1)
6	(1, -1, -1)	(0,0,-1)
7	(-1, -1, -1)	(0,0,-1)
0	(-1, 1, 1)	(0,1,0)
1	(1, 1, 1)	(0,1,0)
5	(1, 1, -1)	(0,1,0)
0	(-1, 1, 1)	(0,1,0)
5	(1, 1, -1)	(0,1,0)
4	(-1, 1, -1)	(0,1,0)
3	(-1, -1, 1)	(0,-1,0)
6	(1, -1, -1)	(0,-1,0)
2	(1, -1, 1)	(0,-1,0)
3	(-1, -1, 1)	(0,-1,0)
7	(-1, -1, -1)	(0,-1,0)
6	(1, -1, -1)	(0,-1,0)
1	(1, 1, 1)	(1,0,0)
2	(1, -1, 1)	(1,0,0)
6	(1, -1, -1)	(1,0,0)
1	(1, 1, 1)	(1,0,0)
6	(1, -1, -1)	(1,0,0)
5	(1, 1, -1)	(1,0,0)
0	(-1, 1, 1)	(-1,0,0)
7	(-1, -1, -1)	(-1,0,0)
3	(-1, -1, 1)	(-1,0,0)
0	(-1, 1, 1)	(-1,0,0)
4	(-1, 1, -1)	(-1,0,0)
7	(-1, -1, -1)	(-1,0,0)

[Practice] OpenGL Lighting

```
import glfw
from OpenGL.GL import *
from OpenGL.GLU import *
import numpy as np
from OpenGL.arrays import vbo
import ctypes

gCamAng = 0.
gCamHeight = 1.

def drawCube_glVertex():
    glBegin(GL_TRIANGLES)

        glNormal3f(0,0,1) # v0, v2, v1, v0, v3, v2
normal
        glVertex3f( -1 , 1 , 1 ) # v0 position
        glVertex3f( 1 , -1 , 1 ) # v2 position
        glVertex3f( 1 , 1 , 1 ) # v1 position

        glVertex3f( -1 , 1 , 1 ) # v0 position
        glVertex3f( -1 , -1 , 1 ) # v3 position
        glVertex3f( 1 , -1 , 1 ) # v2 position

        glNormal3f(0,0,-1)
        glVertex3f( -1 , 1 , -1 ) # v4
        glVertex3f( 1 , 1 , -1 ) # v5
        glVertex3f( 1 , -1 , -1 ) # v6

        glVertex3f( -1 , 1 , -1 ) # v4
        glVertex3f( 1 , -1 , -1 ) # v6
        glVertex3f( -1 , -1 , -1 ) # v7
```

```
glNormal3f(0,1,0)
glVertex3f( -1 , 1 , 1 ) # v0
glVertex3f( 1 , 1 , 1 ) # v1
glVertex3f( 1 , 1 , -1 ) # v5

glVertex3f( -1 , 1 , 1 ) # v0
glVertex3f( 1 , 1 , -1 ) # v5
glVertex3f( -1 , 1 , -1 ) # v4

glNormal3f(0,-1,0)
glVertex3f( -1 , -1 , 1 ) # v3
glVertex3f( 1 , -1 , -1 ) # v6
glVertex3f( 1 , -1 , 1 ) # v2

glVertex3f( -1 , -1 , 1 ) # v3
glVertex3f( -1 , -1 , -1 ) # v7
glVertex3f( 1 , -1 , -1 ) # v6

glNormal3f(1,0,0)
glVertex3f( 1 , 1 , 1 ) # v1
glVertex3f( 1 , -1 , 1 ) # v2
glVertex3f( 1 , -1 , -1 ) # v6

glVertex3f( 1 , 1 , 1 ) # v1
glVertex3f( 1 , -1 , -1 ) # v6
glVertex3f( 1 , 1 , -1 ) # v5

glNormal3f(-1,0,0)
glVertex3f( -1 , 1 , 1 ) # v0
glVertex3f( -1 , -1 , -1 ) # v7
glVertex3f( -1 , -1 , 1 ) # v3

glVertex3f( -1 , 1 , 1 ) # v0
glVertex3f( -1 , 1 , -1 ) # v4
glVertex3f( -1 , -1 , -1 ) # v7
glEnd()
```



```

def createVertexArraySeparate():
    varr = np.array([
        (0,0,1),          # v0 normal
        (-1, 1, 1),      # v0 position
        (0,0,1),          # v2 normal
        ( 1, -1, 1),      # v2 position
        (0,0,1),          # v1 normal
        ( 1, 1, 1),       # v1 position

        (0,0,1),          # v0 normal
        (-1, 1, 1),      # v0 position
        (0,0,1),          # v3 normal
        (-1, -1, 1),     # v3 position
        (0,0,1),          # v2 normal
        ( 1, -1, 1),     # v2 position

        (0,0,-1),
        (-1, 1, -1),     # v4
        (0,0,-1),
        ( 1, 1, -1),     # v5
        (0,0,-1),
        ( 1, -1, -1),    # v6

        (0,0,-1),
        (-1, 1, -1),     # v4
        (0,0,-1),
        ( 1, -1, -1),    # v6
        (0,0,-1),
        (-1, -1, -1),    # v7

        (0,1,0),
        (-1, 1, 1),      # v0
        (0,1,0),
        ( 1, 1, 1),      # v1
        (0,1,0),
        ( 1, 1, -1),     # v5

        (0,1,0),
        (-1, 1, 1),      # v0
        (0,1,0),
        ( 1, 1, -1),     # v5
        (0,1,0),
        (-1, 1, -1),    # v4

        (0,-1,0),
        (-1, -1, 1),     # v3
        (0,-1,0),
        ( 1, -1, -1),    # v6
        (0,-1,0),
        ( 1, -1, 1),     # v2
    ])

```

```

        (0,-1,0),
        (-1, -1, 1),    # v3
        (0,-1,0),
        (-1, -1, -1),   # v7
        (0,-1,0),
        ( 1, -1, -1),   # v6

        (1,0,0),
        ( 1, 1, 1),     # v1
        (1,0,0),
        ( 1, -1, 1),    # v2
        (1,0,0),
        ( 1, -1, -1),   # v6

        (1,0,0),
        ( 1, 1, 1),     # v1
        (1,0,0),
        ( 1, -1, -1),   # v6
        (1,0,0),
        ( 1, 1, -1),    # v5

        (-1,0,0),
        (-1, 1, 1),     # v0
        (-1,0,0),
        (-1, -1, -1),   # v7
        (-1,0,0),
        (-1, -1, 1),    # v3

        (-1,0,0),
        (-1, 1, 1),     # v0
        (-1,0,0),
        (-1, 1, -1),    # v4
        (-1,0,0),
        (-1, -1, -1),   # v7
    ], 'float32')
    return varr

```

```

def drawCube_glDrawArray():
    global gVertexArraySeparate
    varr = gVertexArraySeparate
    glEnableClientState(GL_VERTEX_ARRAY)
    glEnableClientState(GL_NORMAL_ARRAY)
    glNormalPointer(GL_FLOAT, 6*varr.itemsize, varr)
    glVertexPointer(3, GL_FLOAT, 6*varr.itemsize,
        ctypes.c_void_p(varr.ctypes.data + 3*varr.itemsize))
    glDrawArrays(GL_TRIANGLES, 0, int(varr.size/6))

```

```

def render():
    global gCamAng, gCamHeight

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)

    glEnable(GL_DEPTH_TEST)

    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45, 1, 1,10)

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

    gluLookAt(5*np.sin(gCamAng),gCamHeight,5*np.cos(
gCamAng), 0,0,0, 0,1,0)

    drawFrame()

    glEnable(GL_LIGHTING)    # try to uncomment:
no lighting
    glEnable(GL_LIGHT0)

    glEnable(GL_RESCALE_NORMAL) # try to
uncomment: lighting will be incorrect if you
scale the object
    # glEnable(GL_NORMALIZE)

    # light position
    glPushMatrix()

    # glRotatef(t*(180/np.pi),0,1,0) # try to
uncomment: rotate light
    lightPos = (3.,4.,5.,1.)    # try to change
4th element to 0. or 1.

```

```

glLightfv(GL_LIGHT0, GL_POSITION, lightPos)
glPopMatrix()

# light intensity for each color channel
lightColor = (1.,1.,1.,1.)
ambientLightColor = (.1,.1,.1,1.)
glLightfv(GL_LIGHT0, GL_DIFFUSE,
lightColor)
glLightfv(GL_LIGHT0, GL_SPECULAR,
lightColor)
glLightfv(GL_LIGHT0, GL_AMBIENT,
ambientLightColor)

# material reflectance for each color
channel
objectColor = (1.,0.,0.,1.)
specularObjectColor = (1.,1.,1.,1.)
glMaterialfv(GL_FRONT,
GL_AMBIENT_AND_DIFFUSE, objectColor)
glMaterialfv(GL_FRONT, GL_SHININESS, 10)
glMaterialfv(GL_FRONT, GL_SPECULAR,
specularObjectColor)

glPushMatrix()
# glRotatef(t*(180/np.pi),0,1,0) # try
to uncomment: rotate object
# glScalef(1.,.2,1.) # try to uncomment:
scale object

glColor3ub(0, 0, 255) # glColor*() is
ignored if lighting is enabled

# drawCube_glVertex()
drawCube_glDrawArray()
glPopMatrix()

glDisable(GL_LIGHTING)

```

```

def drawFrame():
    glBegin(GL_LINES)
    glColor3ub(255, 0, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([1.,0.,0.]))
    glColor3ub(0, 255, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([0.,1.,0.]))
    glColor3ub(0, 0, 255)
    glVertex3fv(np.array([0.,0.,0]))
    glVertex3fv(np.array([0.,0.,1.]))
    glEnd()

def key_callback(window, key, scancode,
action, mods):
    global gCamAng, gCamHeight
    if action==glfw.PRESS or
action==glfw.REPEAT:
        if key==glfw.KEY_1:
            gCamAng += np.radians(-10)
        elif key==glfw.KEY_3:
            gCamAng += np.radians(10)
        elif key==glfw.KEY_2:
            gCamHeight += .1
        elif key==glfw.KEY_W:
            gCamHeight += -.1

```

```

gVertexArraySeparate = None
def main():
    global gVertexArraySeparate

    if not glfw.init():
        return
    window =
glfw.create_window(640,640,'Lecture13',
None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.set_key_callback(window,
key_callback)
    glfw.swap_interval(1)

    gVertexArraySeparate =
createVertexArraySeparate()

    while not
glfw.window_should_close(window):
        glfw.poll_events()
        render()
        glfw.swap_buffers(window)

    glfw.terminate()

if __name__ == "__main__":
    main()

```

glNormalPointer()

- **glNormalPointer(type, stride, pointer)**
- : specifies the location and data format of an array of **normals**
 - **type**: The data type of each coordinate value in the array. GL_FLOAT, GL_SHORT, GL_INT or GL_DOUBLE.
 - **stride**: The number of bytes to offset to the next normal
 - **pointer**: The pointer to the first coordinate of the first normal in the array
- **c.f.) glVertexPointer(size, type, stride, pointer)**
- : specifies the location and data format of an array of **vertex coordinates**
 - **size**: The number of vertex coordinates, 2 for 2D points, 3 for 3D points
 - **type**: The data type of each coordinate value in the array. GL_FLOAT, GL_SHORT, GL_INT or GL_DOUBLE.
 - **stride**: The number of bytes to offset to the next vertex
 - **pointer**: The pointer to the first coordinate of the first vertex in the array

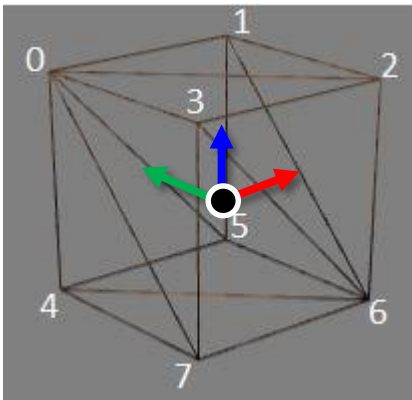
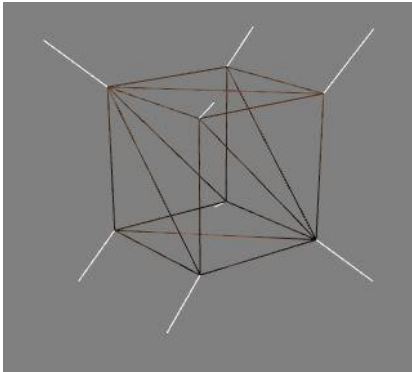
Quiz #3

- Go to <https://www.slido.com/>
- Join #cg-hyu
- Click “Polls”

- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

Normals of the Cube for Smooth Shading



vertex index	position	normal
0	(-1 , 1 , 1)	(-0.5773502691896258 , 0.5773502691896258 , 0.5773502691896258)
1	(1 , 1 , 1)	(0.8164965809277261 , 0.4082482904638631 , 0.4082482904638631)
2	(1 , -1 , 1)	(0.4082482904638631 , -0.4082482904638631 , 0.8164965809277261)
3	(-1 , -1 , 1)	(-0.4082482904638631 , -0.8164965809277261 , 0.4082482904638631)
4	(-1 , 1 , -1)	(-0.4082482904638631 , 0.4082482904638631 , -0.8164965809277261)
5	(1 , 1 , -1)	(0.4082482904638631 , 0.8164965809277261 , -0.4082482904638631)
6	(1 , -1 , -1)	(0.5773502691896258 , -0.5773502691896258 , -0.5773502691896258)
7	(-1 , -1 , -1)	(-0.8164965809277261 , -0.4082482904638631 , -0.4082482904638631)

Lighting in Modern OpenGL

- Legacy OpenGL
 - Only allows Gouraud shading & Blinn-Phong illumination model.
 - Rendering quality is not good.
- Modern OpenGL:
 - No specific lighting & shading model in modern OpenGL
 - Programmers have to implement Phong or other illumination model in vertex shader or fragment shader.
 - Example: the shader code in this online demo
<http://www.cs.toronto.edu/~jacobson/phong-demo/>

Next Time

- Lab in this week:
 - Lab assignment 8
- Next lecture:
 - 9 - Orientation & Rotation
- **Class Assignment #2**
 - **Due: 23:59, May 20, 2019**
- Acknowledgement: Some materials come from the lecture slides of
 - Prof. Andy van Dam, Brown Univ., <http://cs.brown.edu/courses/csci1230/lectures.shtml>
 - Prof. Jinxiang Chai, Texas A&M Univ., http://faculty.cs.tamu.edu/jchai/csce441_2016spring/lectures.html
 - Prof. Steve Marschner, Cornell Univ., <http://www.cs.cornell.edu/courses/cs4620/2014fa/index.shtml>
 - Prof. JungHyun Han, Korea Univ., <http://media.korea.ac.kr/book/>