Computer Graphics

7 - Mesh 2, Lighting & Shading 1

Yoonsang Lee
Spring 2022

Midterm Exam Announcement

- Date & time: **Apr 27**, 09:30 - 10:30 am
- Place: IT.BT, 508
- Scope: Lecture 2 ~ 7

- **You cannot leave the room until the end of the exam** even if you finish the exam earlier.

- Please bring your student ID card to the exam.

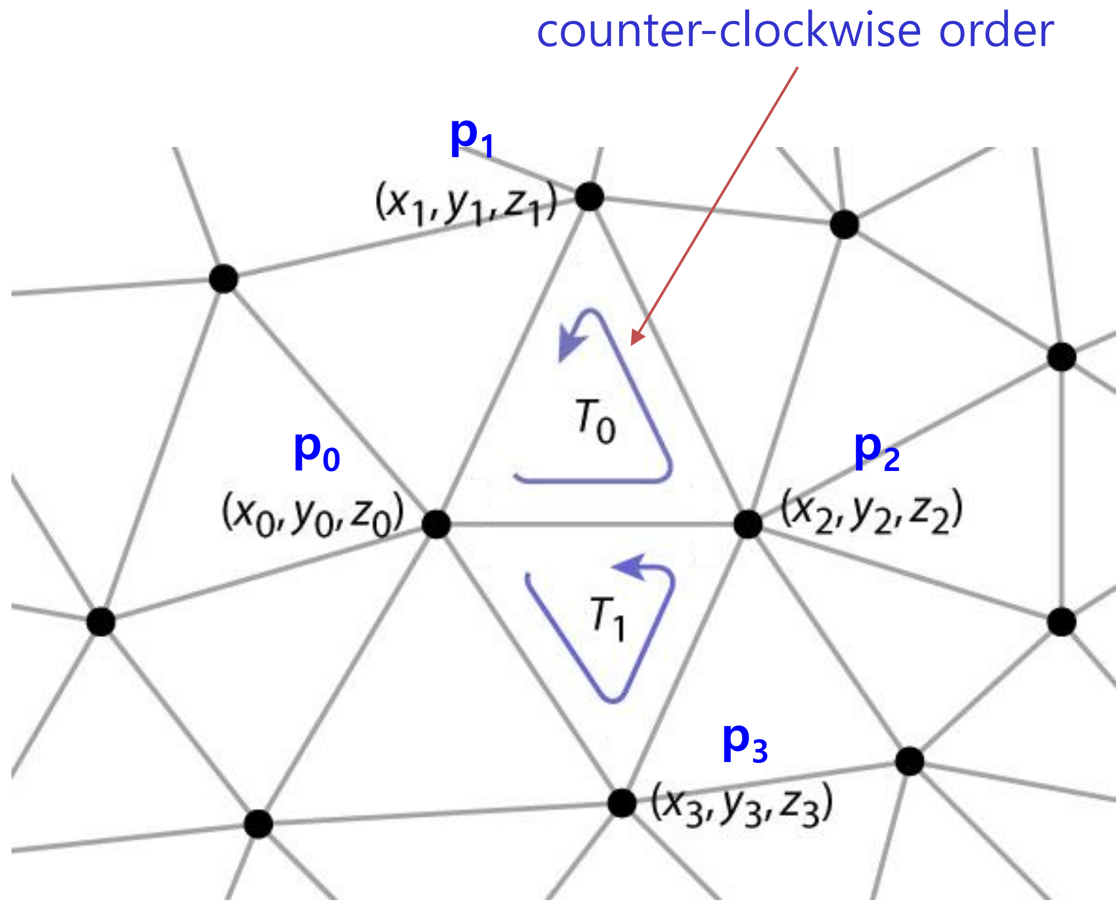
- If you are unable to take the offline exam (stay abroad, corona confirmed, etc.), please contact the TA in advance.
  - Chaejun Sohn (손채준 조교), thscowns@gmail.com
  - You must inform the TA **at least two days before the exam**.

Topics Covered

- Mesh
  - Representations for triangle meshes - Indexed triangle set
  - OBJ file format
- Reflection of Light
- Phong Illumination Model
- Shading
  - Face / Vertex Normal
  - Flat / Goraud / Phong Shading

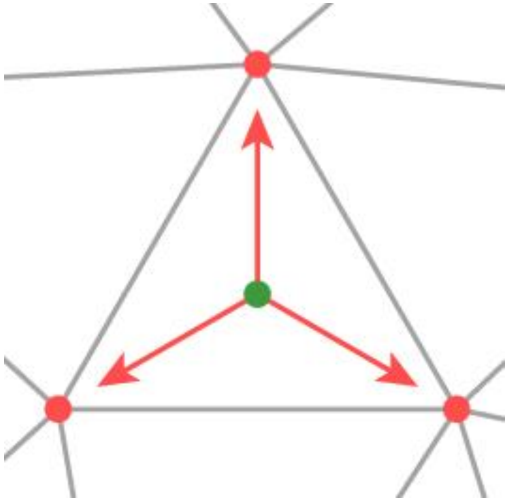
Recall: Separate triangles

	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
⋮	⋮	⋮	⋮

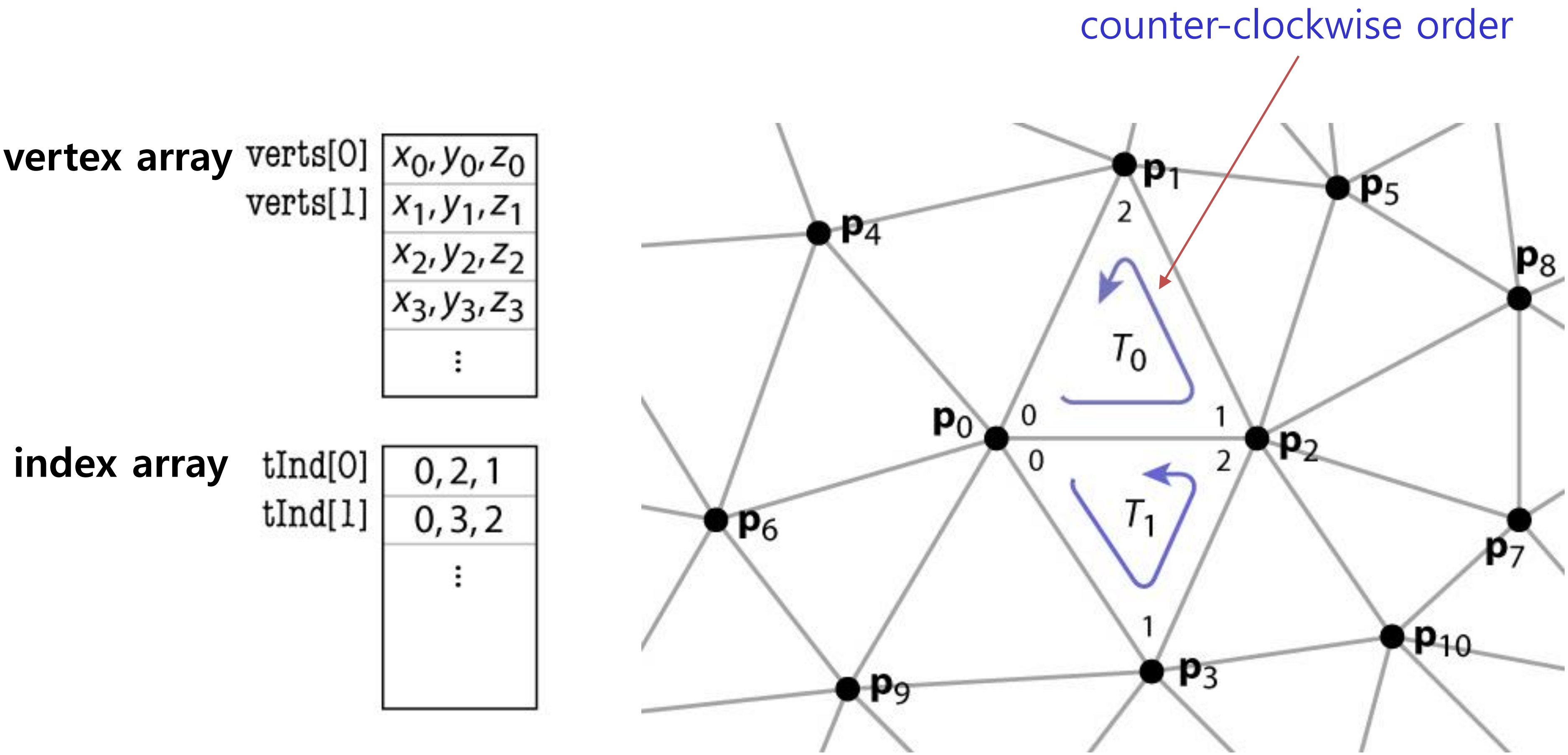


Indexed triangle set

- Store each vertex once
- Each triangle points to its three vertices



Indexed triangle set



Indexed Triangle Set

- Memory efficient: each vertex position is stored only once.
- Represents topology and geometry separately.
- Finding neighbor triangles is at least well defined.
  - Neighbor triangles share same vertex indices.

Drawing Indexed Triangles using Vertex & Index Array

- 1. Create a vertex array & **index array** for your mesh
  - The vertex array **should not have duplicate vertex data**

- 2. Specify "pointer" to this vertex array
  - Same with the separate triangles case

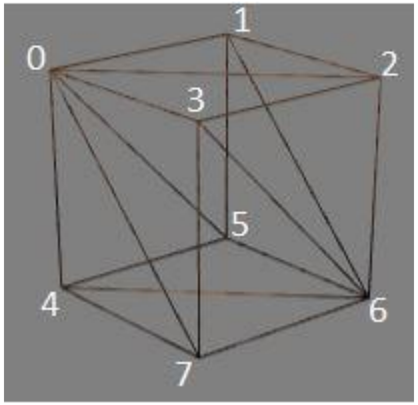
- 3. Render the mesh using the specified "pointer" & the pointer to the **index array** (which contains vertex indices to be rendered)
  - Using **glDrawElements()**

glDrawElements()

- **glDrawElements(mode , count , type , indices)**
- : render primitives from vertex & index array data
  - **mode**: The primitive type to render. GL_POINTS, GL_TRIANGLES, ...
  - **count**: The number of vertex indices to be rendered
  - **type**: The type of the values in **indices**. GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, or GL_UNSIGNED_INT
  - **indices**: The pointer to the index array

[Practice] Drawing Indexed Triangles using Vertex & Index Array

```
def createVertexAndIndexArrayIndexed():
    varr = np.array([
            ( -1 ,  1 ,  1 ), # v0
            (  1 ,  1 ,  1 ), # v1
            (  1 , -1 ,  1 ), # v2
            ( -1 , -1 ,  1 ), # v3
            ( -1 ,  1 , -1 ), # v4
            (  1 ,  1 , -1 ), # v5
            (  1 , -1 , -1 ), # v6
            ( -1 , -1 , -1 ), # v7
            ], 'float32')
    iarr = np.array([
            (0,2,1),
            (0,3,2),
            (4,5,6),
            (4,6,7),
            (0,1,5),
            (0,5,4),
            (3,6,2),
            (3,7,6),
            (1,2,6),
            (1,6,5),
            (0,7,3),
            (0,4,7),
            ])
    return varr, iarr
```



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)

Starts from the "[Practice] Drawing Separate Triangles using Vertex Array" code in the prev. lecture,

```
def render():
    # ...
    drawFrame()
    glColor3ub(255, 255, 255)
    drawCube_glDrawElements()


def drawCube_glDrawElements():
    global gVertexArrayIndexed, gIndexArray
    varr = gVertexArrayIndexed
    iarr = gIndexArray
    glEnableClientState(GL_VERTEX_ARRAY)
    glVertexPointer(3, GL_FLOAT, 3*varr.itemsize, varr)
    glDrawElements(GL_TRIANGLES, iarr.size, GL_UNSIGNED_INT, iarr)


# ...
gVertexArrayIndexed = None
gIndexArray = None

def main():
    # ...
    global gVertexArrayIndexed, gIndexArray

    # ...
    gVertexArrayIndexed, gIndexArray = createVertexAndIndexArrayIndexed()

    while not glfw.window_should_close(window):
    # ...
```

Quiz #1

- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

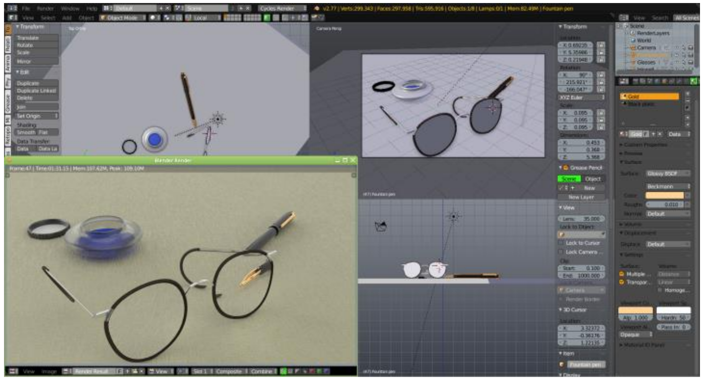
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

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

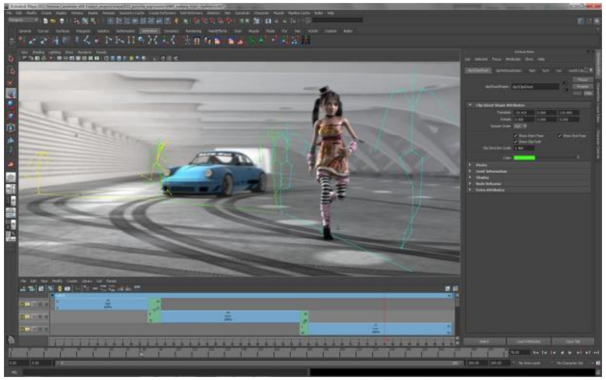
Do we need to hard-code all vertex positions and indices?

- Of course not!

- An *object file* or *model file* storing polygon mesh data is usually created using 3D modeling tools.



Blender



Maya

- Applications (such as games) usually load vertex and index data from an *object file* and draw the object using the loaded data.

3D Model File Formats

- DXF – AutoCAD
  - Supports 2-D and 3-D; binary
- 3DS – 3DS MAX
  - Flexible; binary
- VRML – Virtual reality modeling language
  - ASCII – Human readable (and writeable)
- **OBJ – Wavefront OBJ format**
  - ASCII – Human readable (and writeable)
  - Extremely simple
  - Widely supported
- Let's take a closer look at OBJ format!

OBJ File Format

```
# this is a comment

# List of vertex positions, in (x, y, z) form.
v 0.123 0.234 0.345
v 0.2 0.5 0.3
v ...
...

# List of vertex normals, in (x,y,z) form; normals
might not be unit vectors.
vn 0.707 0.000 0.707
vn ...
...

# List of vertex texture coordinates, in (u, v) form.
vt 0.500 1
vt ...
...
```

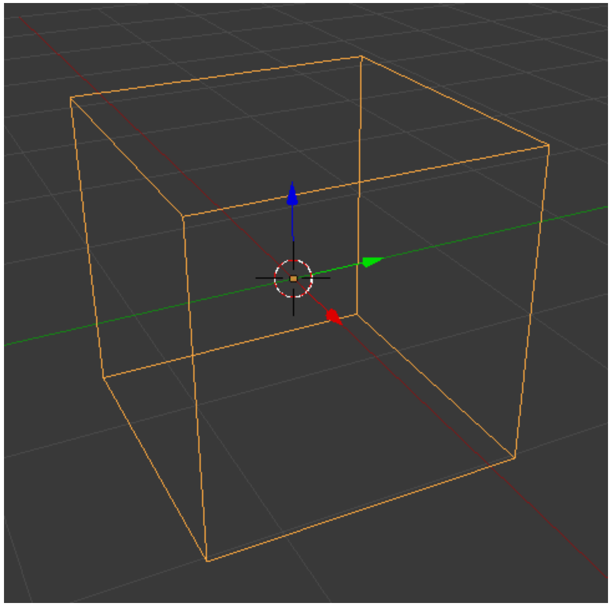
OBJ File Format

```
# List of faces (all argument indices are 1-based indices!)

# with vertex positions only - vertex_position_index
f 1 2 3
f 2 3 4
...

#
vertex_position_index/texture_coordinates_index/vertex_normal_
index
f 6/4/1 3/5/3 7/6/5

# vertex_position_index//vertex_normal_index
f 7//1 8//2 9//3
...

# vertex_position_index/texture_coordinates_index
f 3/1 4/2 5/3
...
```

OBJ File Format

- Other supported featues:
  - for parameter space vertices for free from geometry
    - `vp 0.310000 3.210000 2.100000`
  - for polyline
    - `l 5 8 1 2 4 9`
  - for reference meterials
    - `mtllib [external .mtl file name]`
    - `usemtl [material name]`
  - ...
- You don't need to use these features in this class.

An OBJ Example



```
# A simple cube
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 1.000000 1.000000
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
f 1 2 3 4
f 5 8 7 6
f 1 5 6 2
f 2 6 7 3
f 3 7 8 4
f 5 1 4 8
```

[Practice] Manipulate an OBJ file with Blender

- Blender
  - https://www.blender.org/
  - Open source
  - Full 3D modeling/rendering/animation tool

- Install & launch Blender

- Reference for basic mouse actions in Blender
  - https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/3D_View_Windows#Changing_Your_Viewpoint,_Part_One

[Practice] Manipulate an OBJ file with Blender

- Save the obj example in the prev. page as cube.obj (using a text editor)

- Click the "start-up" cube object in the Blender and press Del key to delete it.

- Import cube.obj into Blender (File-Import)
  - Press ‘z’ to render in wireframe mode

- Edit cube.obj somehow (using a text editor)

- Delete the loaded cube and re-import cube.obj into Blender again

- Press ‘tab’ to switch to *Edit mode*

[Practice] Manipulate an OBJ file with Blender

- Click to select a vertex and click "move" icon from the left icons (or press 'G')
- Move the selected vertex by dragging red/blue/green arrows
- Export this mesh to cube.obj (File – Export)
- Open cube.obj using a text editor and check what is changed
- Reference for *Edit mode* in Blender
  - https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Mesh_Edit_Mode
- Reference for *Object mode* in Blender
  - https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Object_Mode

OBJ Sources

- https://free3d.com/
- https://www.cgtrader.com/free-3d-models

- You can download any .obj model files from these sites and open them in Blender.

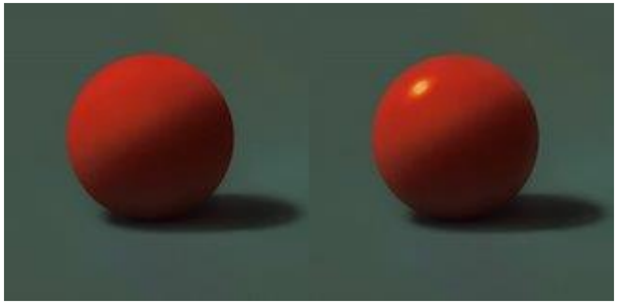
- OBJ file format is very popular:
  - Most modeling programs will export OBJ files
  - Most rendering packages will read in OBJ files

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 a object surface.
  – such as surface color, highlight on surface

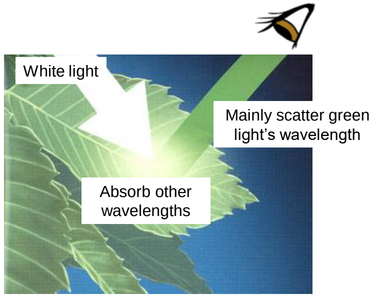


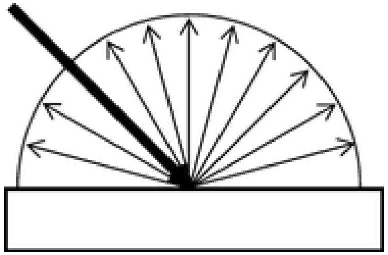
- Types of reflection:
  – Diffuse reflection
  – Specular reflection
    - Ideal specular reflection
    - Non-ideal specular reflection (a.k.a. Glossy reflection)

* 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

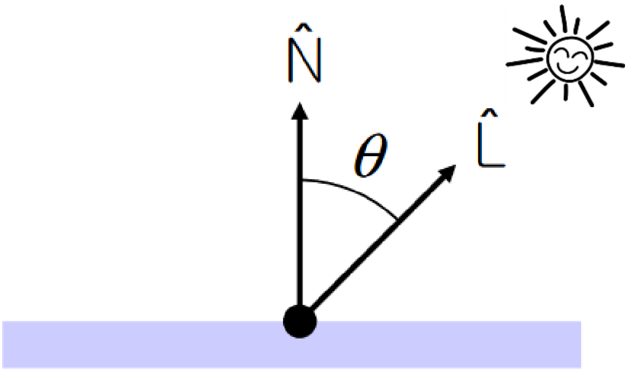
scatter all wavelengths with roughly equal strength

absorb all wavelengths (scatters little)

Diffuse Reflection - Lambert's Cosine Law

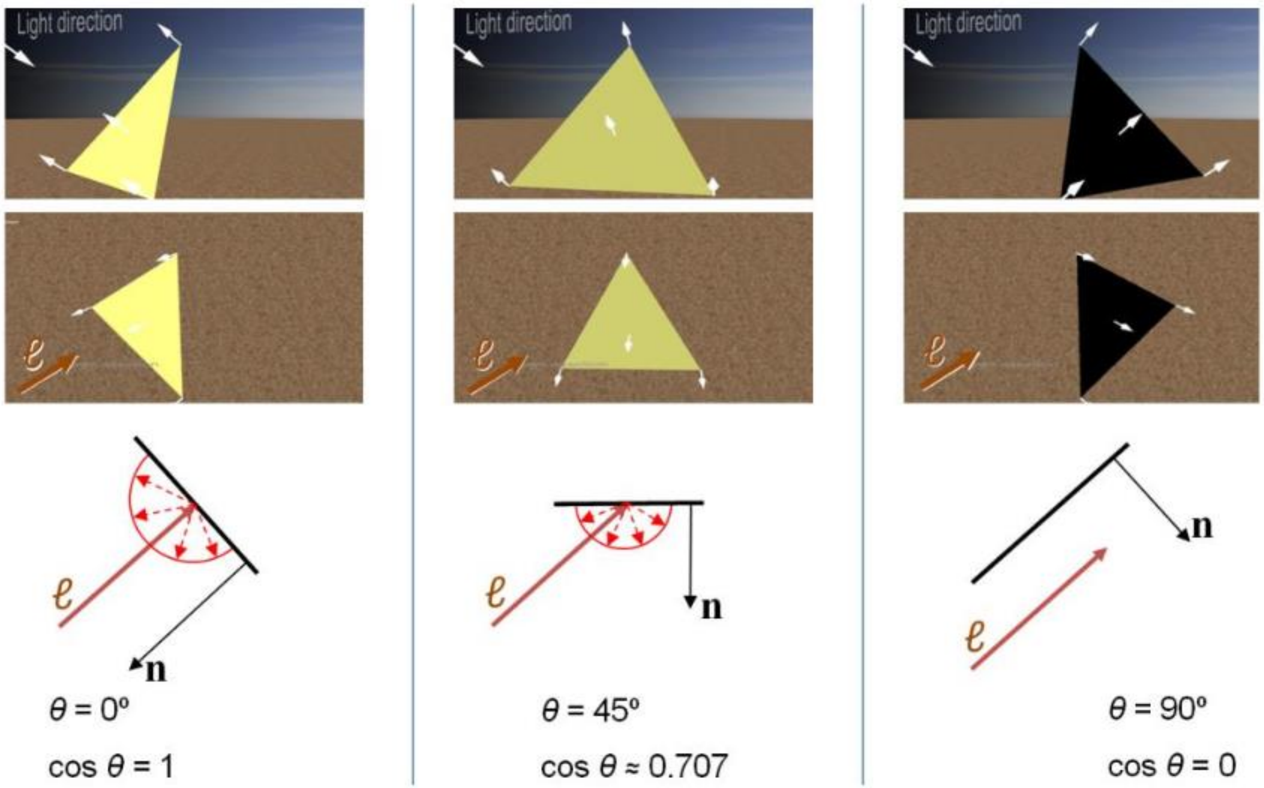
- The **reflected energy** from a small surface area is proportional to the **cosine of the angle** between **incident light direction** and the **surface normal**

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



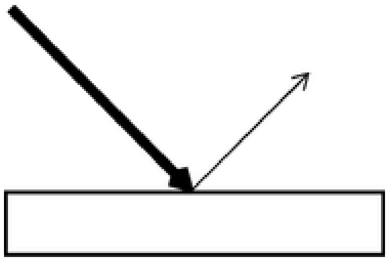
$I_{reflected}$	intensity of reflected ray
$I_{incident}$	intensity of incident ray
$\hat{\mathbf{N}}$	normal to the reflection surface at the point of the incidence
$\hat{\mathbf{L}}$	normalized light direction vector

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

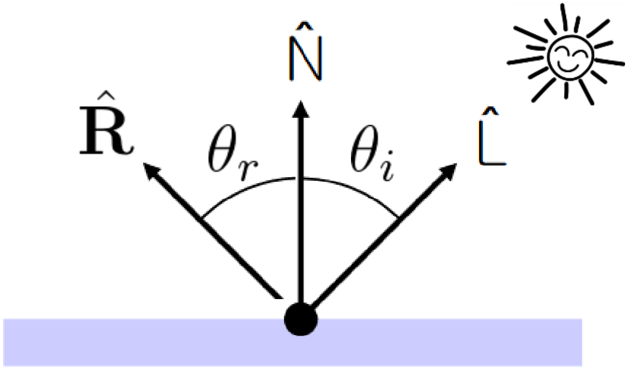


- **View-dependent**

Ideal Specular Reflection - Laws of Reflection

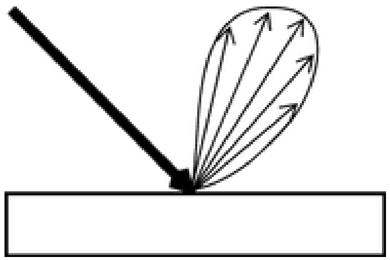
- $\hat{\mathbf{N}}$, $\hat{\mathbf{L}}$, $\hat{\mathbf{R}}$ lie in the same plane
- $\theta_r = \theta_i$
- ($\hat{\mathbf{L}}$ and $\hat{\mathbf{R}}$ are on the opposite sides of $\hat{\mathbf{N}}$)

$\hat{\mathbf{N}}$	normal to the reflection surface at the point of the incidence
$\hat{\mathbf{L}}$	normalized indicent ray direction vector
$\hat{\mathbf{R}}$	normalized reflected ray direction vector



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

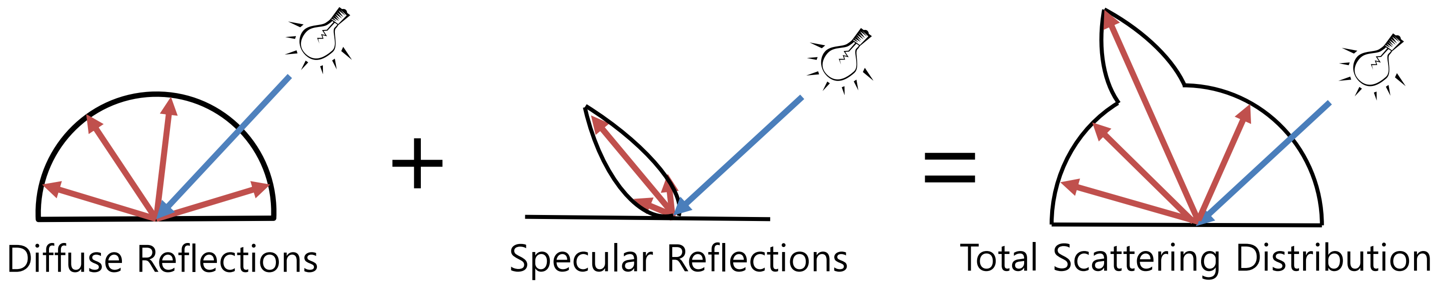
- : Reflection on shiny & glossy surface, but not as smooth as a mirror
- Reflected rays are “spread out” due to surface roughness
- → Generate bright highlights



- **View-dependent**

Reflection of General Materials

- Many materials' surface have both diffuse reflection and (non-ideal) specular reflection.



Quiz #2

- Go to https://www.slido.com/
- Join #**cg-ys**
- 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)

- In computer graphics, **lighting** (or i**llumination**) refers to the process of computing the effects of lights.

- → Computing surface color and highlights of objects.

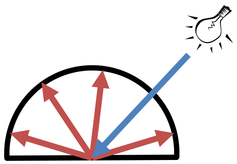
Phong Illumination Model

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

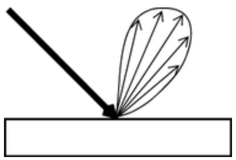
Bùi Tường Phong
(1942 – 1975)

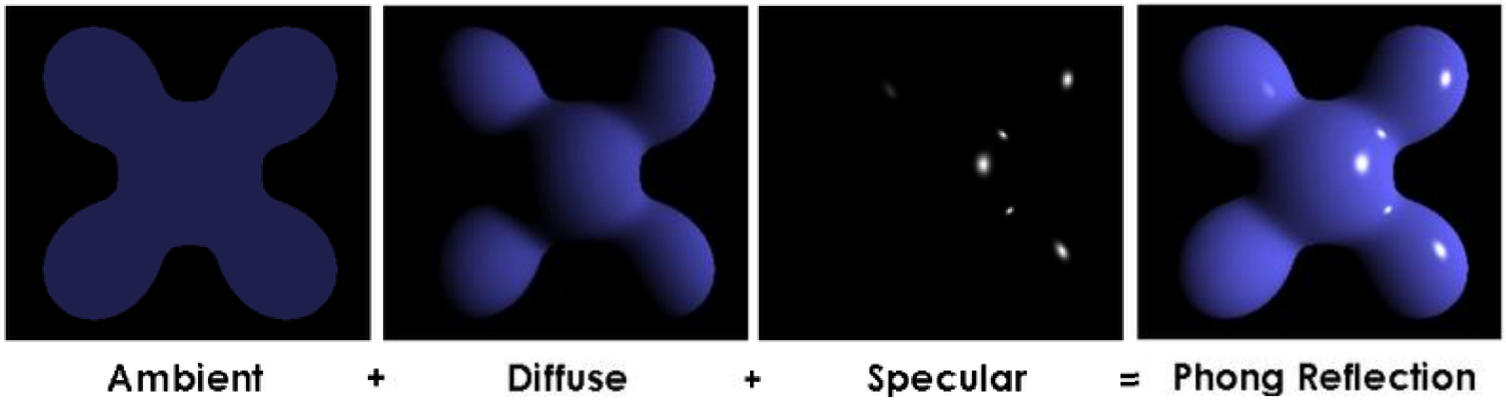
Phong Illumination Model

- 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 Light

$$I = k_a C_a$$

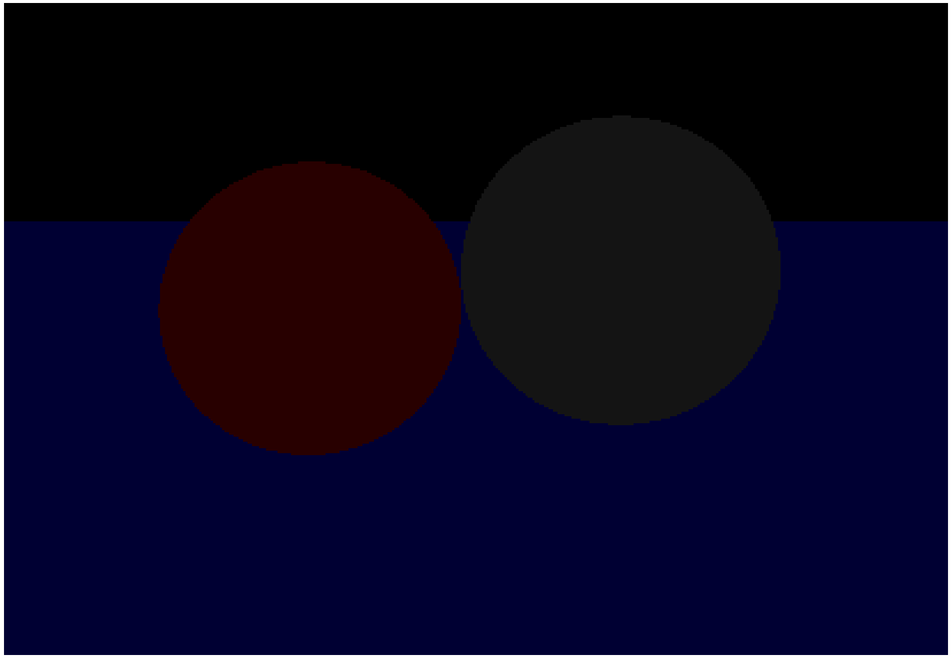
- C_a =intensity of ambient light
- k_a=ambient reflection coefficient

- Actually 3 equations for 3 C_as! (C_a^r, C_a^g, C_a^b for Red, Green, Blue)

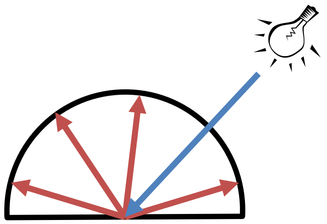
- **Intensity I is calculated for any point on the surface of the object.**
  - **for a polygon vertex**
  - **or for any interior point in a polygon (corresponds to a pixel in the film space).**

Total Illumination

$$I = k_a C_a$$



Diffuse Light

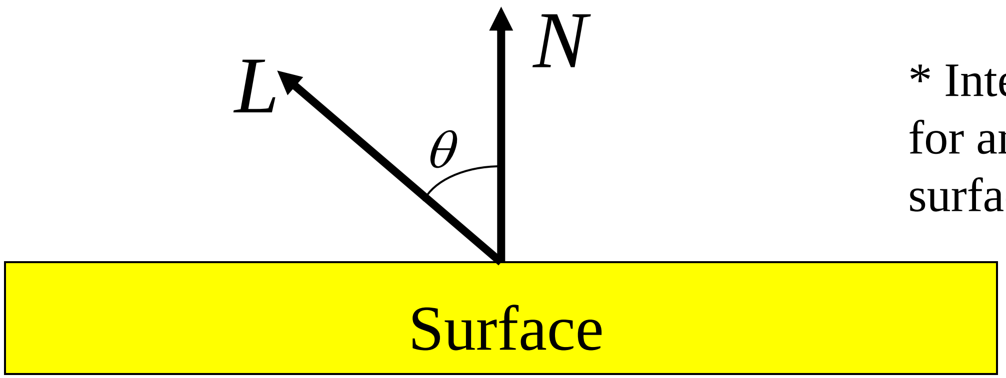


Lambert's Cosine Law

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

- C_d = intensity of diffuse light (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

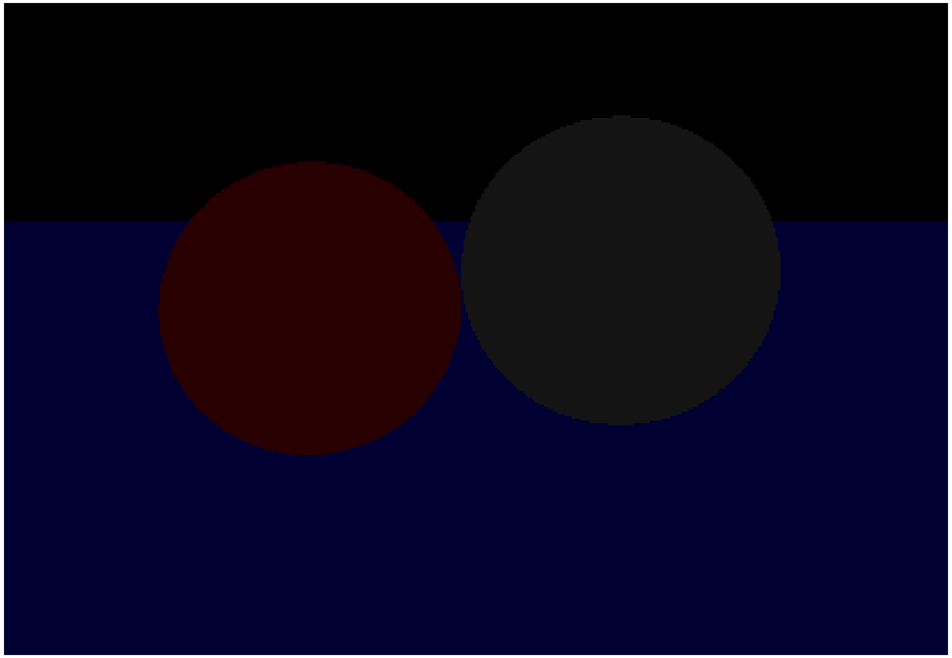
$$\cos(\theta) = L \cdot N$$



* Intensity I is calculated for any point on the surface of the object.

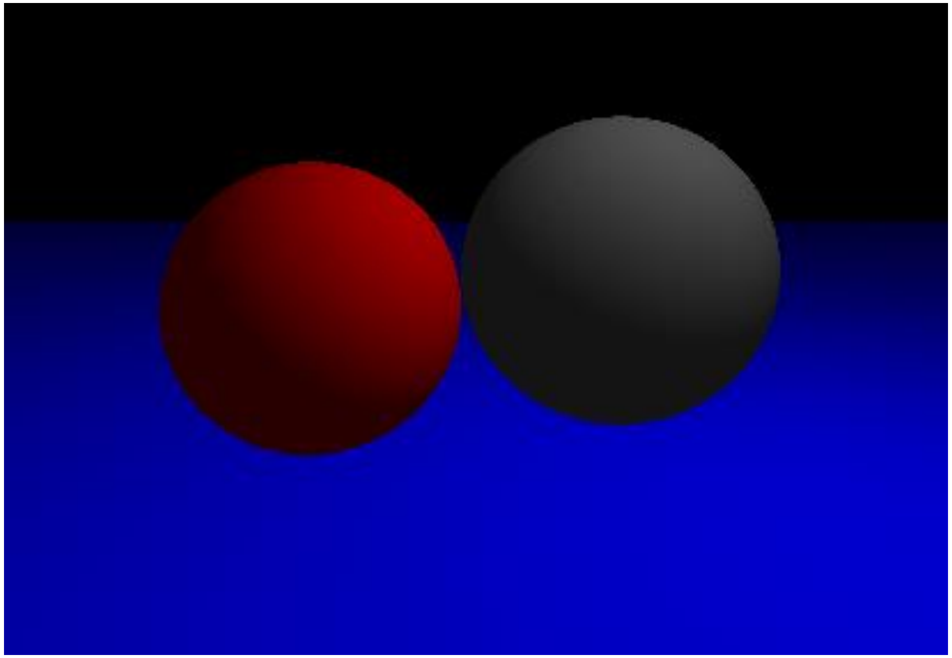
Total Illumination

$$I = k_a C_a$$

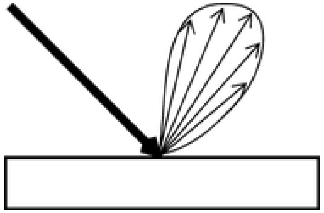


Total Illumination

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



Specular Light

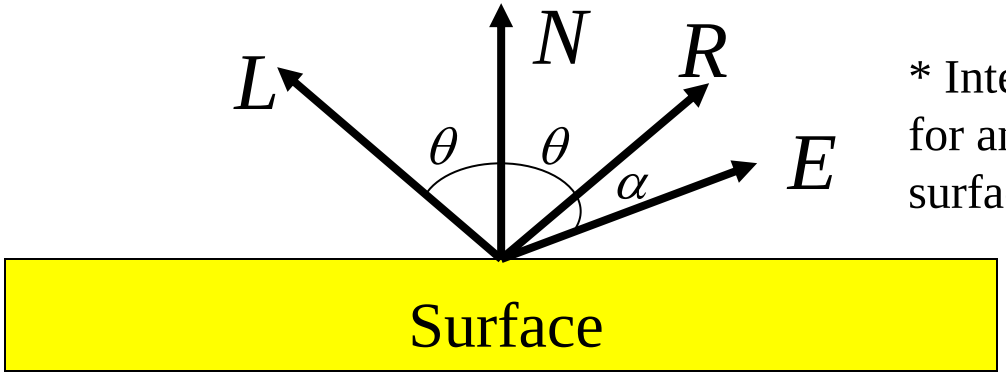


approximation for glossy reflection

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

- C_s = intensity of specular light (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

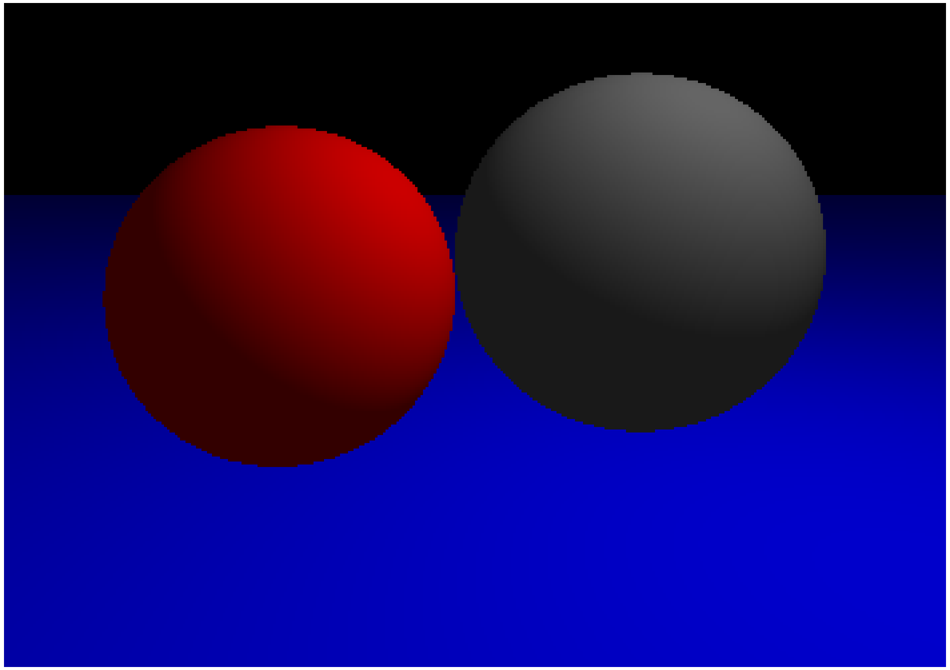
$$\cos(\alpha) = R \cdot E$$



* Intensity I is calculated for any point on the surface of the object.

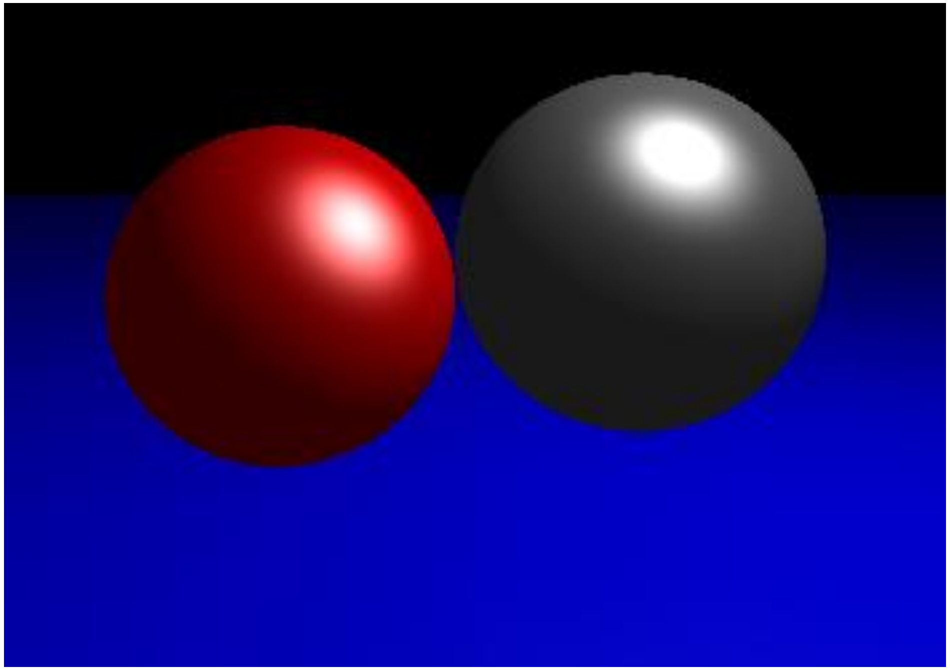
Total Illumination

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



Total Illumination

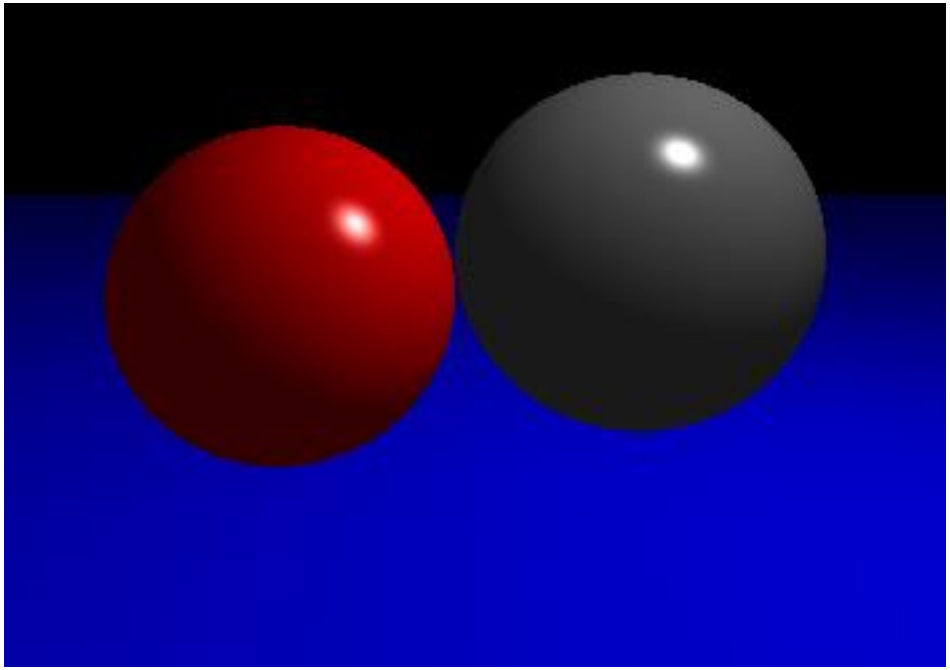
$$I = k_a C_a + k_d C_d (L \cdot N) + k_s C_s (R \cdot E)^n$$



$n = 5$

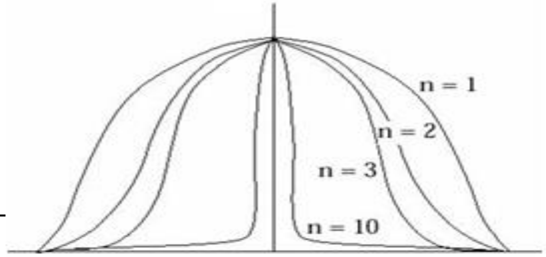
Total Illumination

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



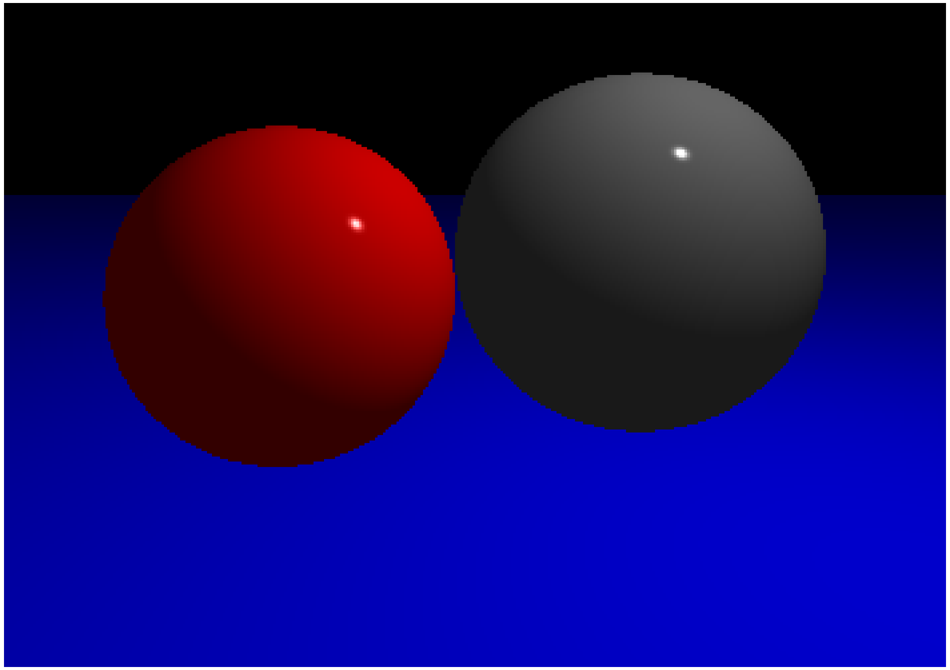
$$n = 50$$

Total Illumination



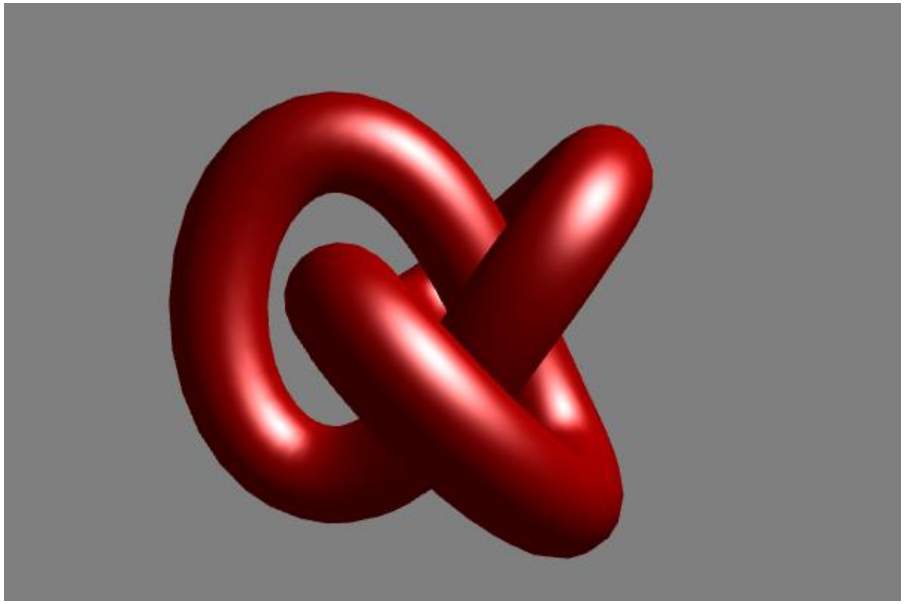
Specular falloff of $(\cos \delta)^n$

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



$n = 500$

[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 #3

- Go to https://www.slido.com/
- Join #**cg-ys**
- 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”.

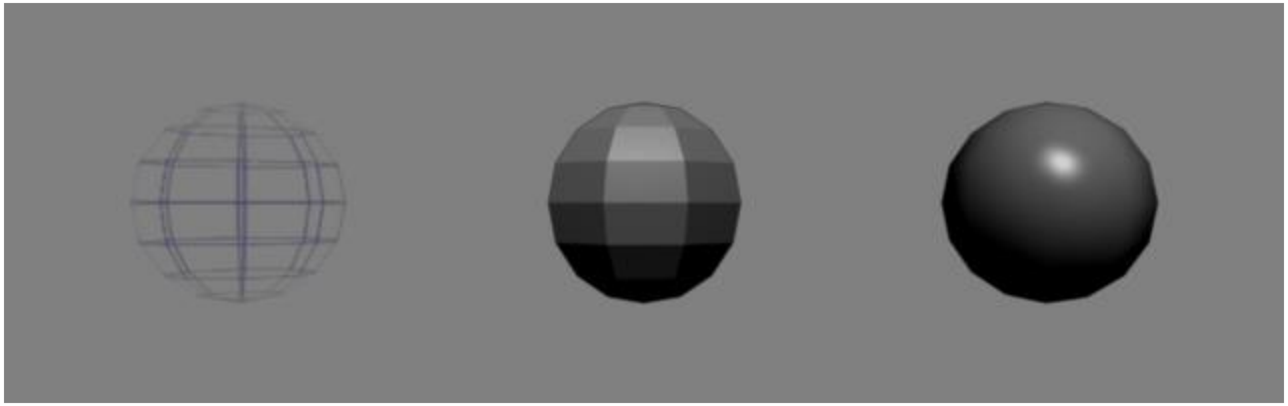
Shading

Shading - General Meaning

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

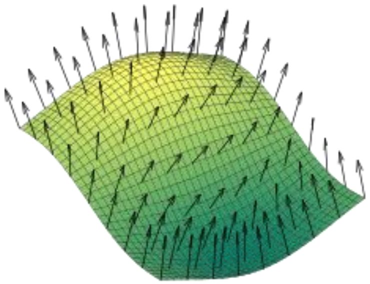
Shading - Meaning in Computer Graphics

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



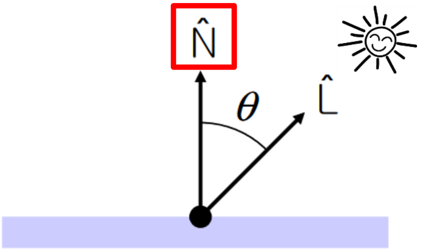
Surface Normal

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



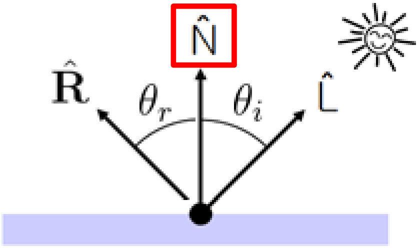
- Plays a key role in shading & illumination process

- Diffuse reflection
  - Lambert's Cosine Law



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

- Specular reflection
  - Laws of Reflection



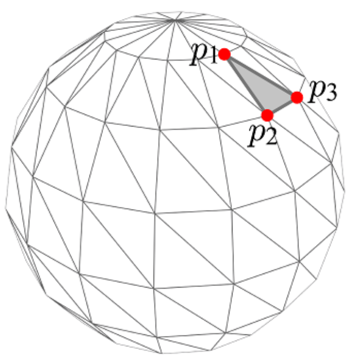
$$\theta_r = \theta_i$$

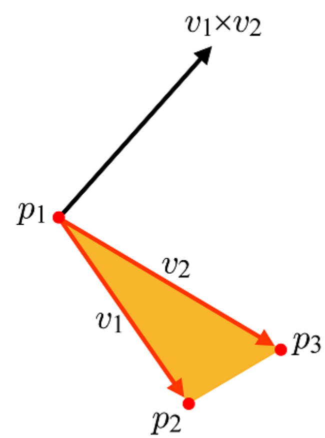
Face Normal

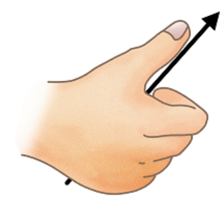
- How to get the surface normal of a polygonal face?

The order does matter!

- The normal of a triangle **<p1, p2, p3>** is computed as $v1 \times v2$
  - v1 is the vector connecting p1 and p2, v2 connects p1 and p3





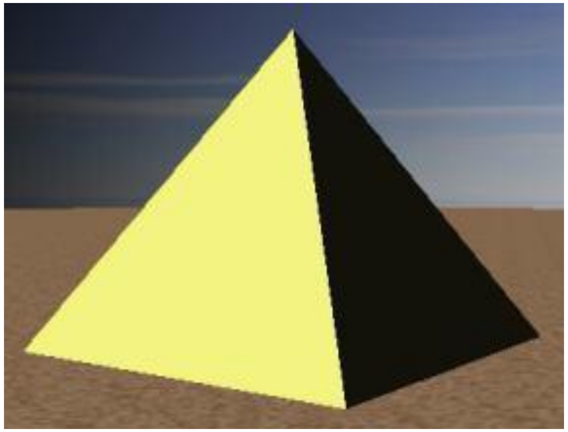


$$\frac{v_1 \times v_2}{||v_1 \times v_2||}$$

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

Flat Shading

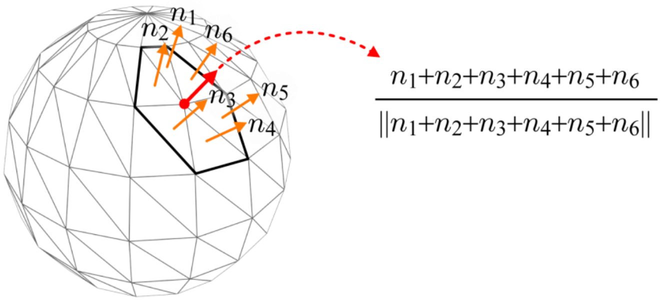
- Use a single face normal for each polygon
- Calculate color (by illumination) once per polygon
  - Typically use center of polygon
- Fast, but not very desirable for curved shapes
  - Even if we increase the number of polygons, it's still "faceted"



Smooth Shading

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

- Two methods:
  - Gouraud shading
  - Phong shading

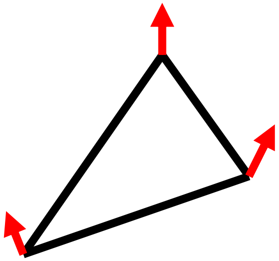


- Use a vertex normal for each vertex
  - For smooth shading, a vertex normal is commonly set to the average of normals of all faces sharing the vertex.

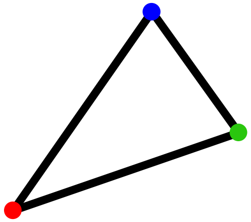
Gouraud Shading

Henri Gouraud
(1944~)

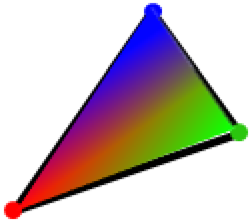
- Use a single vertex normal for each vertex



- Calculate color (by illumination) at each vertex

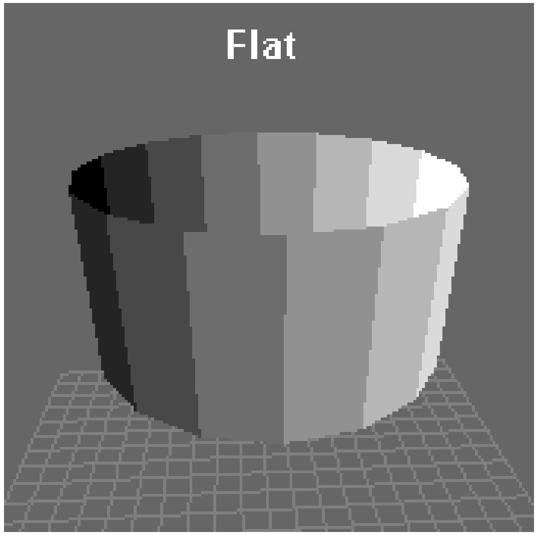


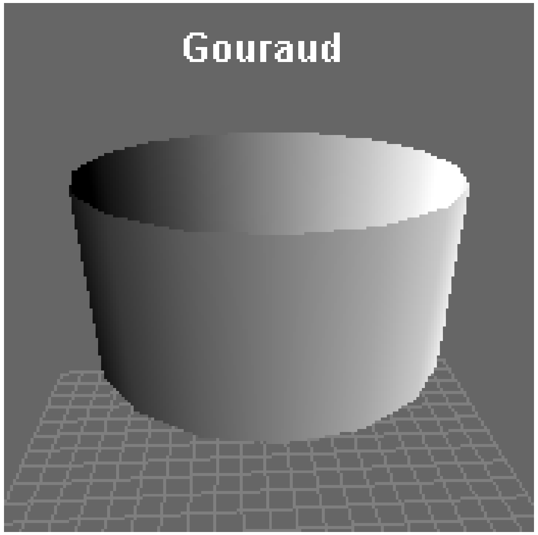
- Interpolate vertex colors across polygon
  – Barycentric interpolation



See more for barycentric interpolation:
https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates

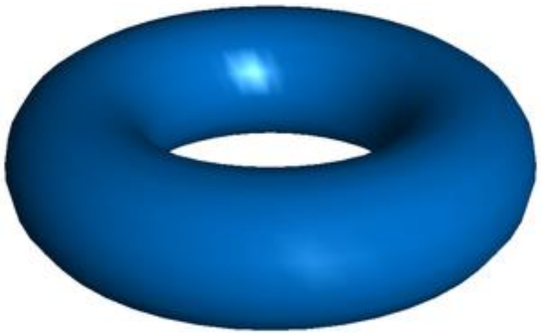
Gouraud Shading

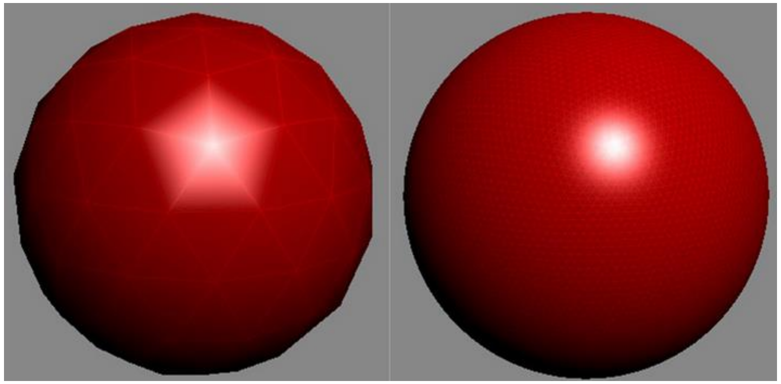




Gouraud Shading

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



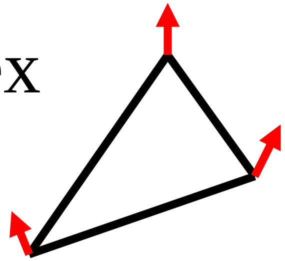


Higher polygon count reduces this artifact

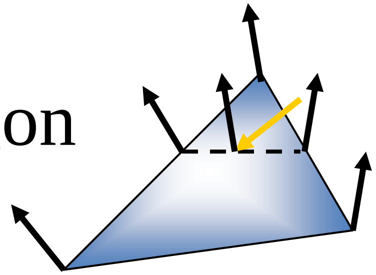
Phong Shading

Bùi Tường Phong
(1942 – 1975)

- Use a single vertex normal for each vertex

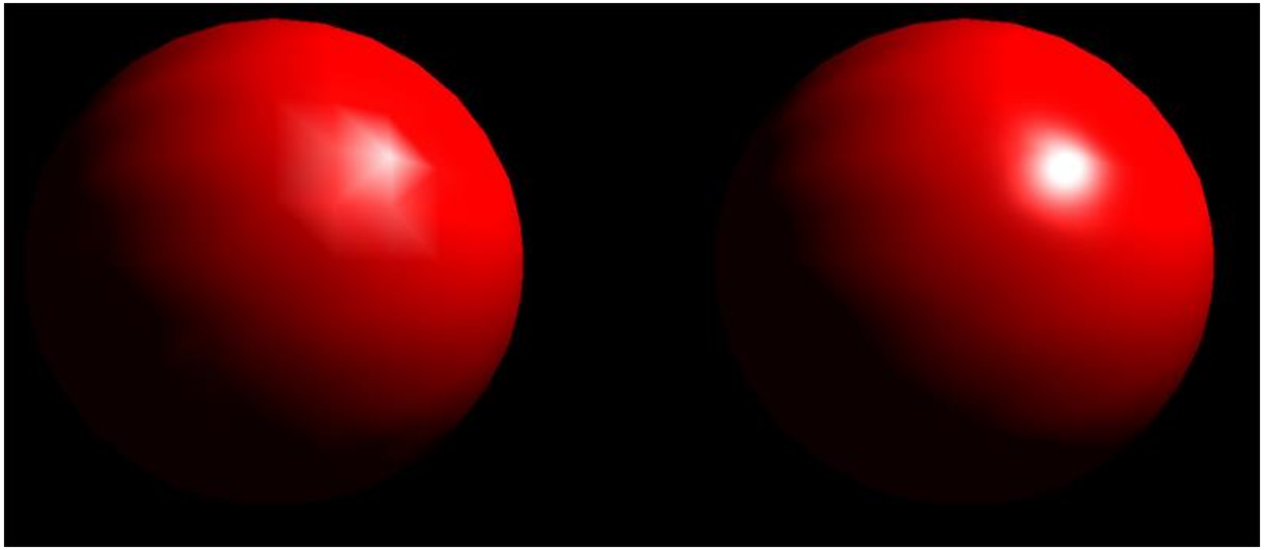


- Interpolate vertex normals across polygon



- Calculate color (by illumination) at each pixel in polygon using the interpolated normal

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/

Next Time

- Lab for this lecture (next Monday):
  - Lab assignment 7
- Next lecture:
  - 8 - Lighting & Shading 2, Hierarchical Modeling

- 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/