LEC21: Shading-part1

Ku-Jin Kim

School of Computer Science & Engineering

Kyungpook National University

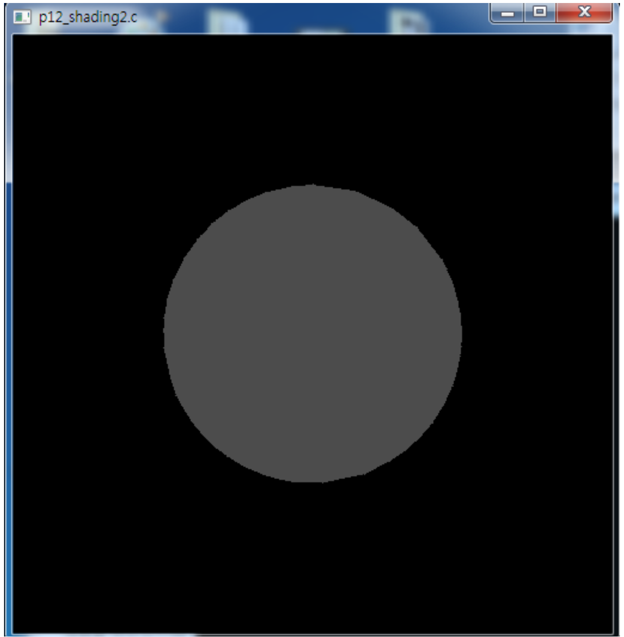
Notice: This PPT slide was created by partially extracting & modifying notes from Edward Angel's Lecture Note for E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

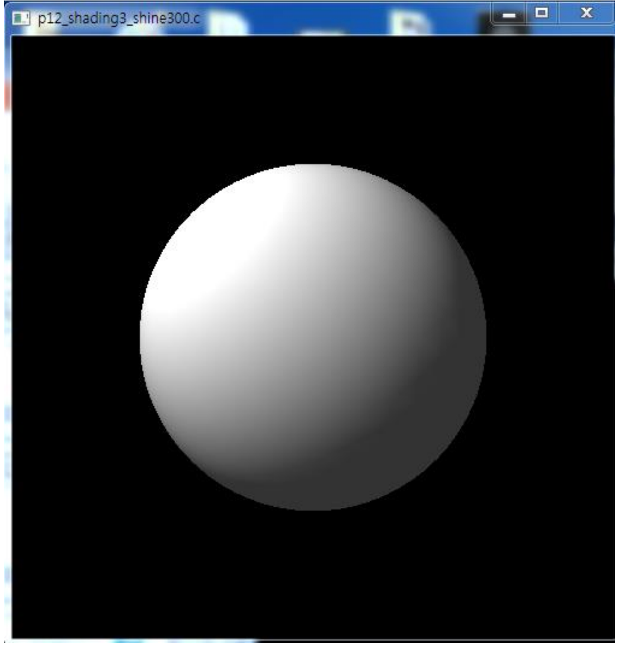
Contents

- Methods for shading objects
- Types of light-material interactions
- Front-face & back-face

Why we need shading

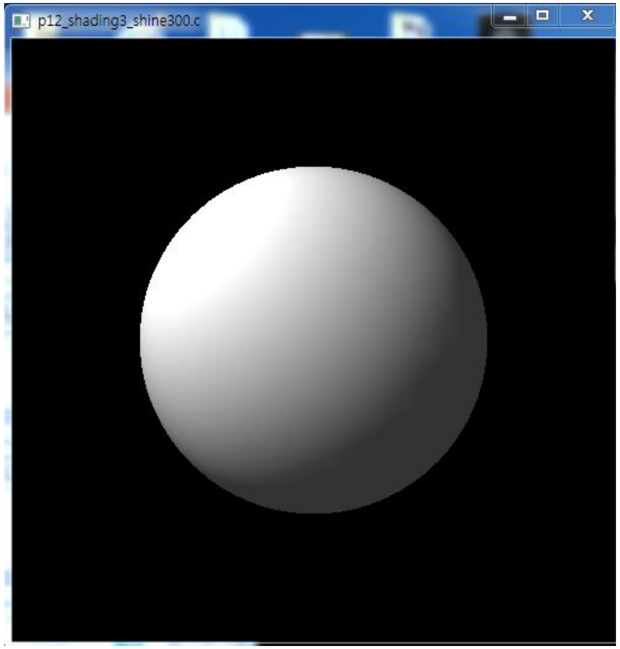
- Suppose we build a model of a sphere using many faces and color them with gl_FragColor. We get something like
- But we want





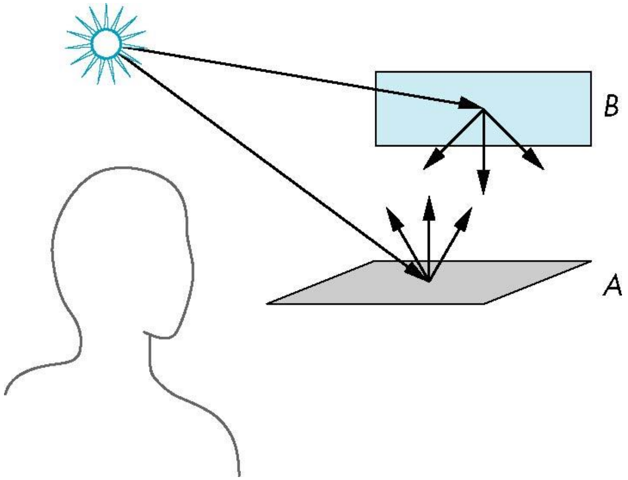
Shading

- Why does this image looks realistic?
- Light-material interactions cause each point to have a different color or shade
- Need to consider
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation



Scattering

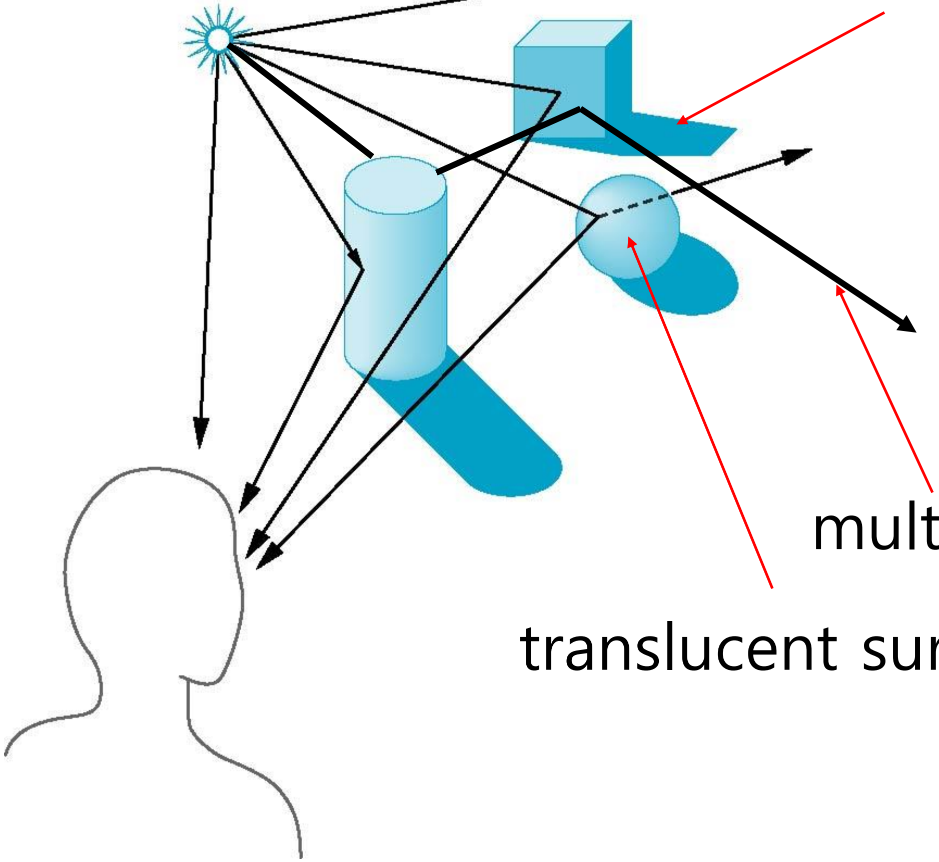
- Light strikes A
  - Some scattered
  - Some absorbed
- Some of scattered light strikes B
  - Some scattered
  - Some absorbed
- Some of this scattered light strikes A and so on



Rendering Equation

- The infinite scattering and absorption of light can be described by the rendering equation
  - Cannot be solved in general
  - Ray tracing is a special case for perfectly reflecting surfaces
- Rendering equation is global and includes
  - Shadows
  - Multiple scattering from object to object

Global Effects



shadow

multiple reflection

translucent surface

Local vs Global Rendering

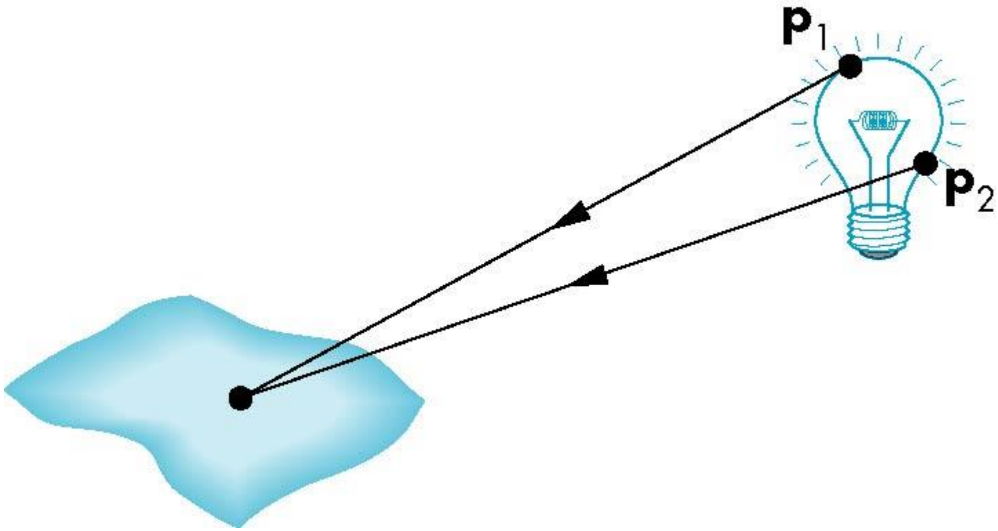
- Correct shading requires a global calculation involving all objects and light sources
  - Incompatible with pipeline model which shades each polygon independently (local rendering)
- However, in computer graphics, especially real time graphics, we are happy if things “look right”
  - Exist many techniques for approximating global effects

Light-Material Interaction

- Light that strikes an object is partially absorbed and partially scattered (reflected)
- The amount reflected determines the color and brightness of the object
  - A surface appears red under white light because the red component of the light is reflected and the rest is absorbed
- The reflected light is scattered in a manner that depends on the smoothness and orientation of the surface

Light Sources

- General light sources are difficult to work with because we must integrate light coming from all points on the source

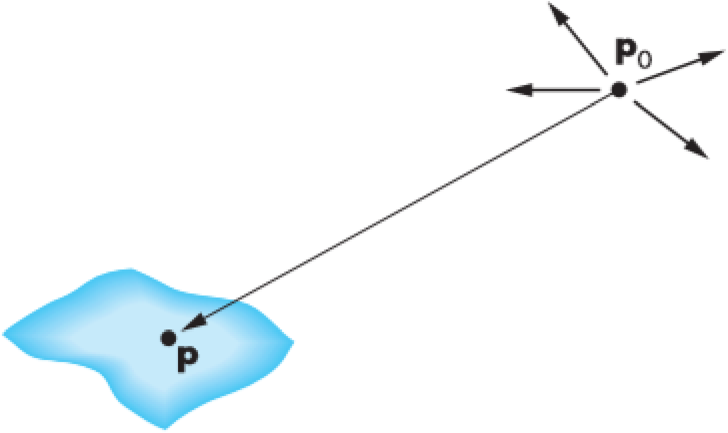


Simple Light Sources (1)

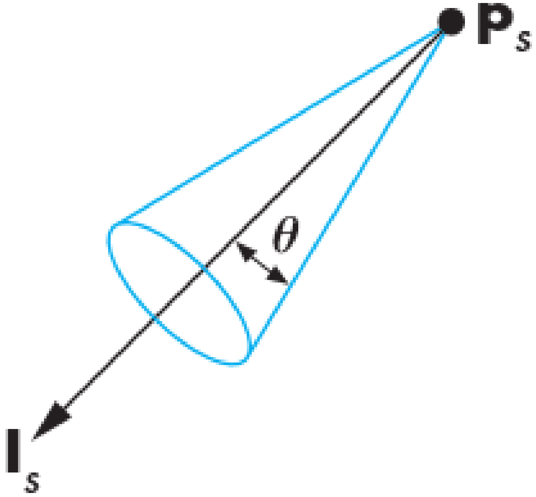
- Point light source
  - Model with position and color
  - Distant source = infinite distance away (parallel)
- Spotlight
  - Restrict light from the point source
- Ambient light
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces

Simple Light Sources (2)

Point light source

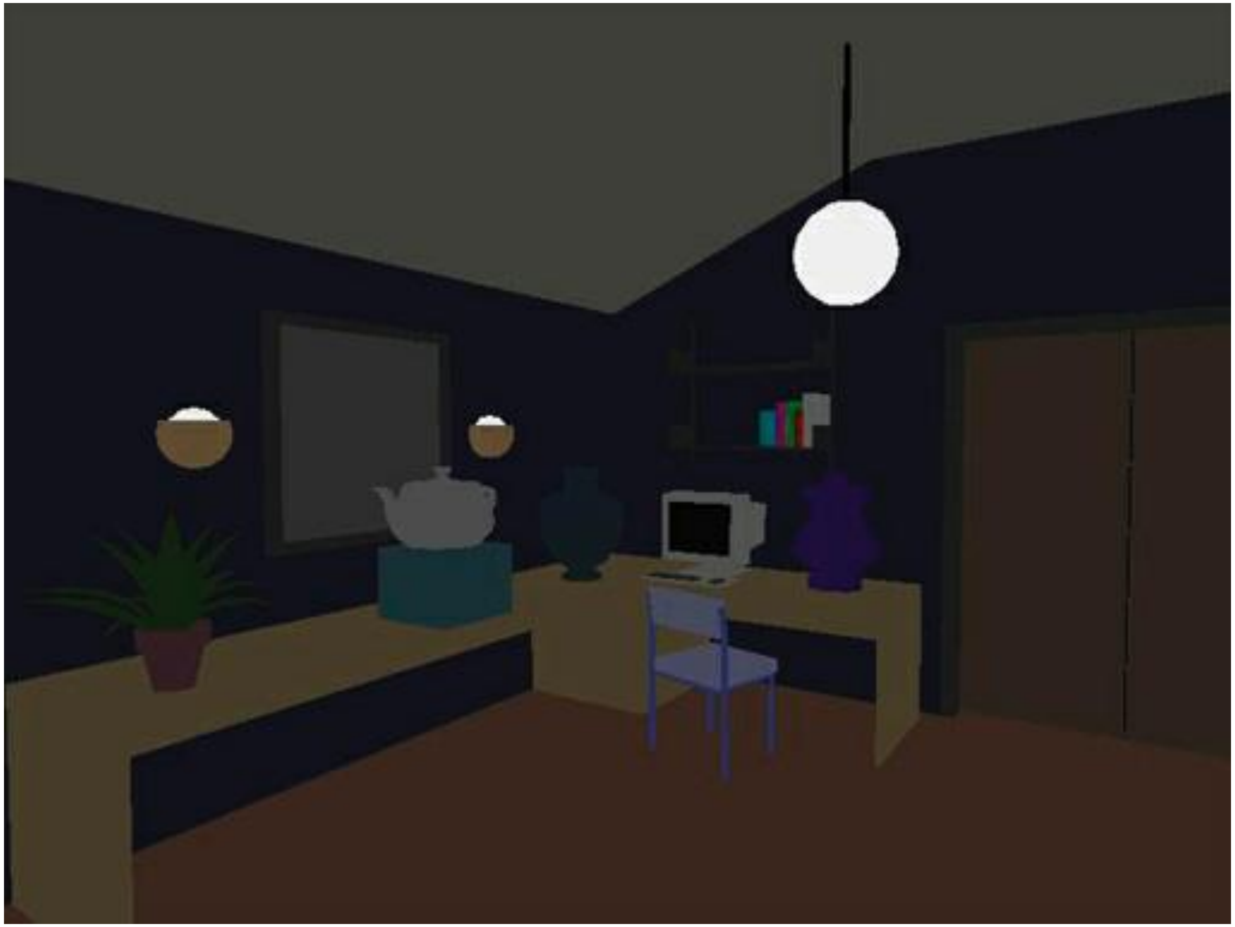


Spot light source



Simple Light Sources (3)

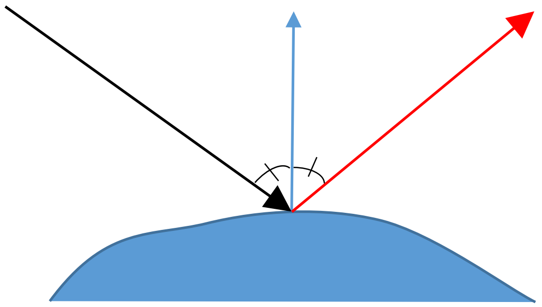
Ambient light

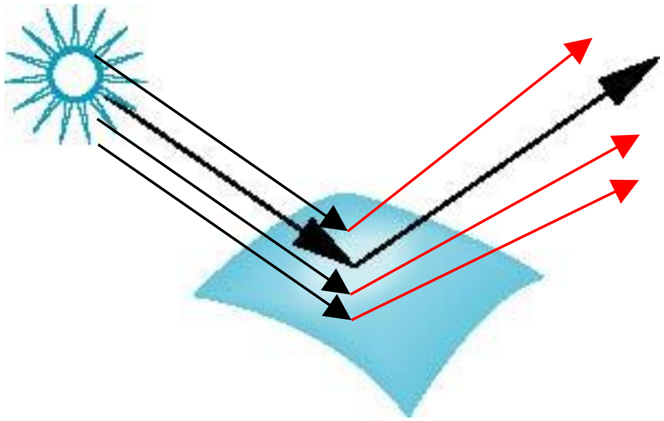


Surface Types – Smooth Surface

- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflected the light

normal vector

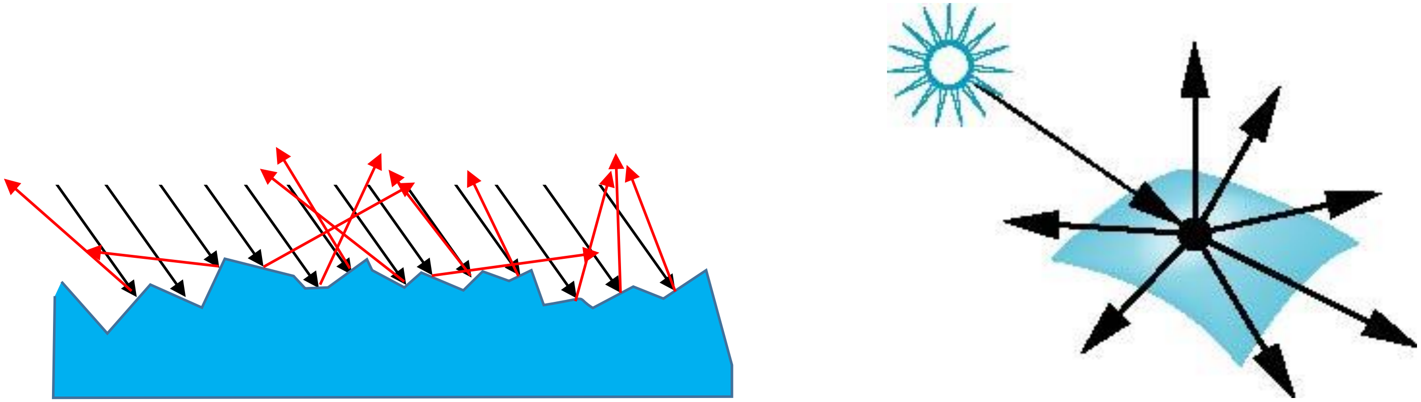




smooth surface

Surface Types – Rough Surface

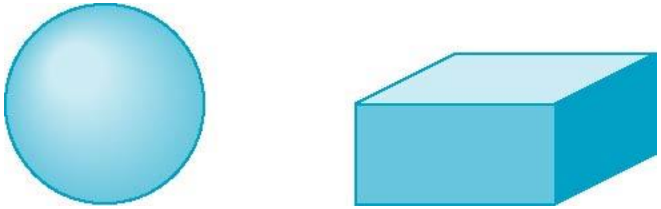
- A very rough surface scatters light in all directions



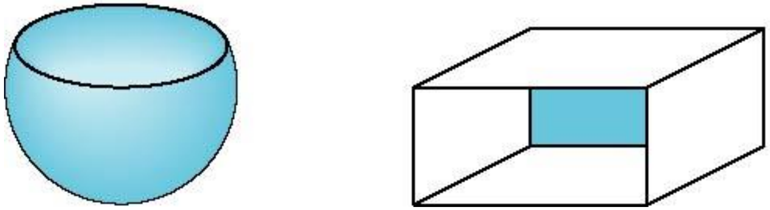
rough surface

Front-Face and Back-Face (1)

- Every face has a front and back
- For many objects, we never see the back face so we don't care how or if it's rendered
- If it matters, we can handle in shader



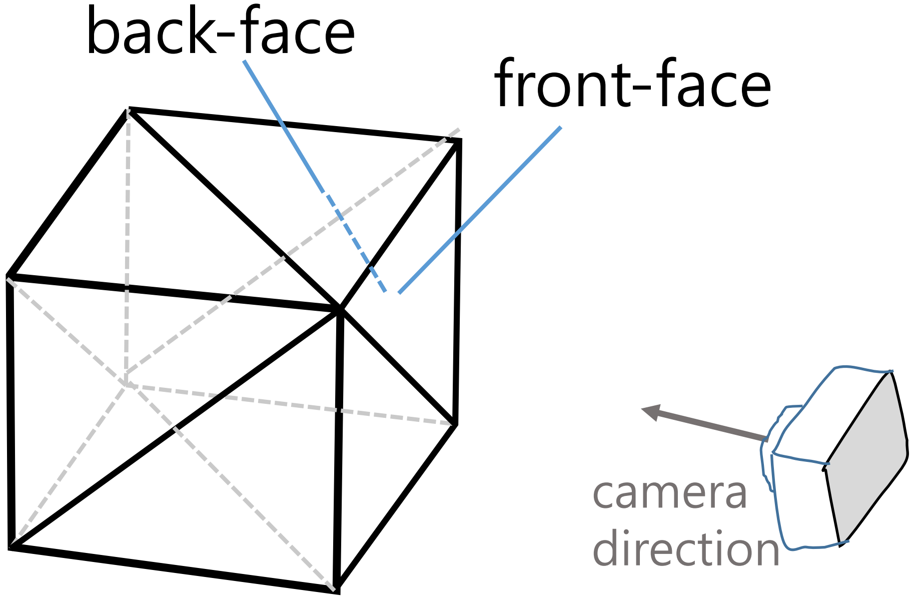
back faces not visible



back faces visible

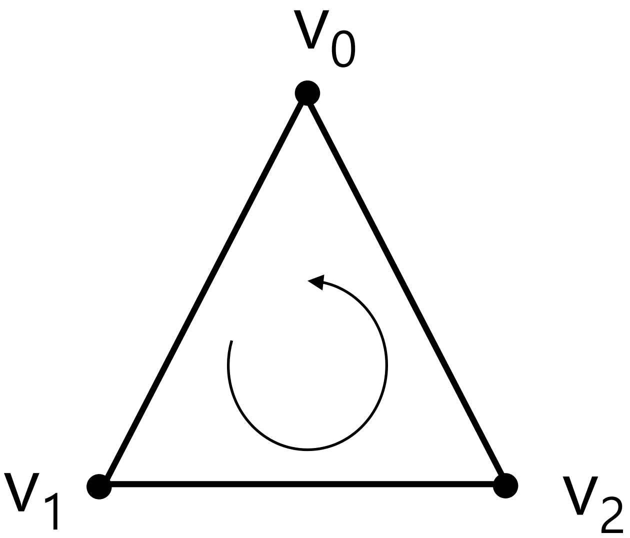
Front-Face and Back-Face (2)

- Back-face culling
  - OpenGL checks all the faces that are front facing towards the viewer and renders those, while discarding all the faces that are back facing
  - Saving us a lot of fragment shader calls.
  - Winding order is used to determine if it's front-face or back-face



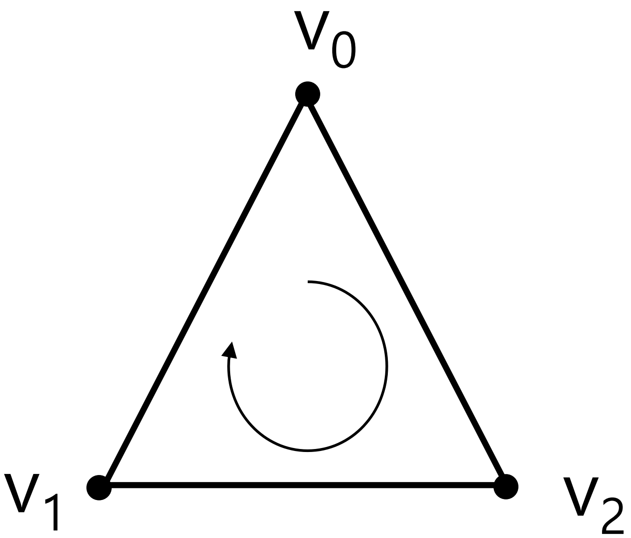
Front-Face and Back-Face (3)

- Winding order
  - When the viewer (camera) is watching



Counterclockwise order

$v_0 \rightarrow v_1 \rightarrow v_2$

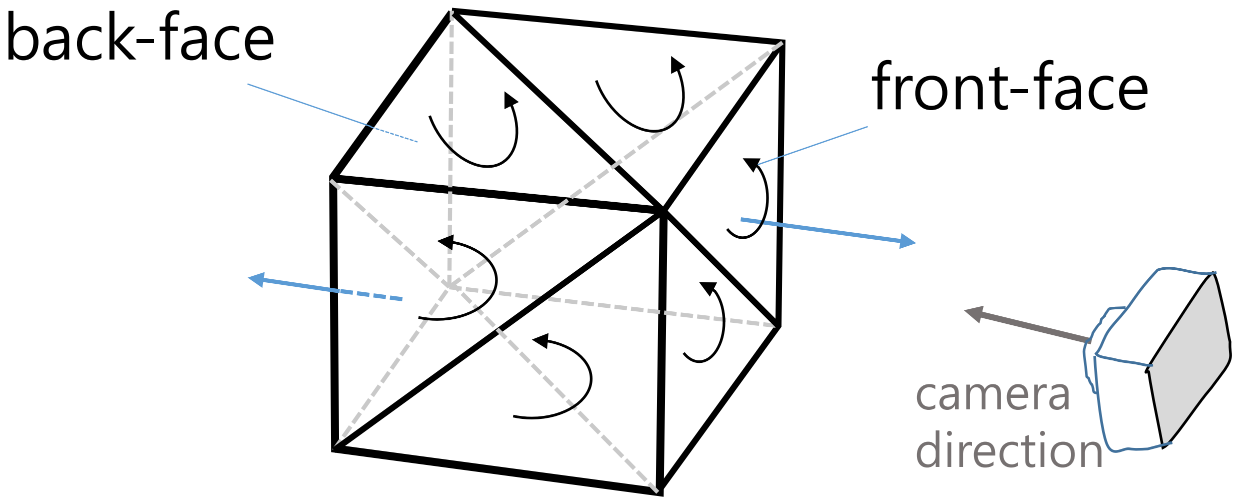


Clockwise order

$v_0 \rightarrow v_2 \rightarrow v_1$

Front-Face and Back-Face (4)

- By default, triangles defined with counter-clockwise vertices are processed as front-facing triangles.
- We assume that front-face is given as counter-clockwise
- Normal vector must come out from the front-face



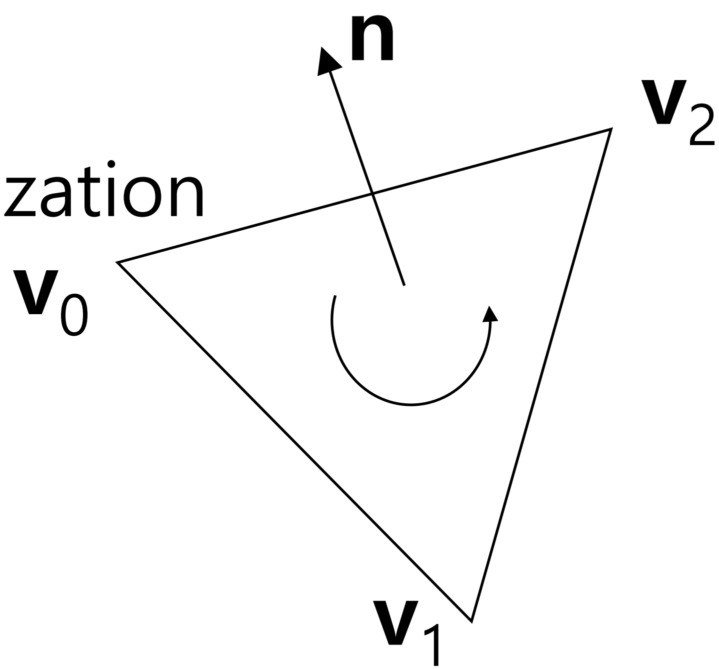
Face Normal

- Face is given by indices: 0, 1, 2
  - counter-clockwise order
- Normal can be obtained by

$$(\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_2 - \mathbf{v}_0)$$

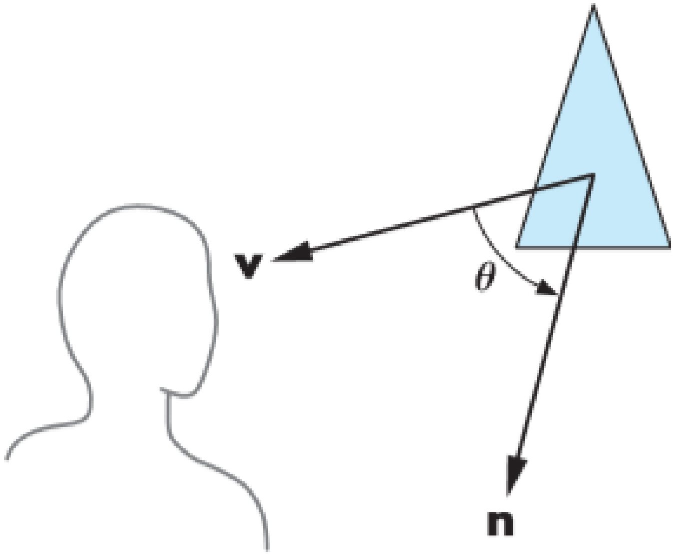
- Normal vector after normalization

$$\mathbf{n} = \frac{(\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_2 - \mathbf{v}_0)}{\|(\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_2 - \mathbf{v}_0)\|}$$



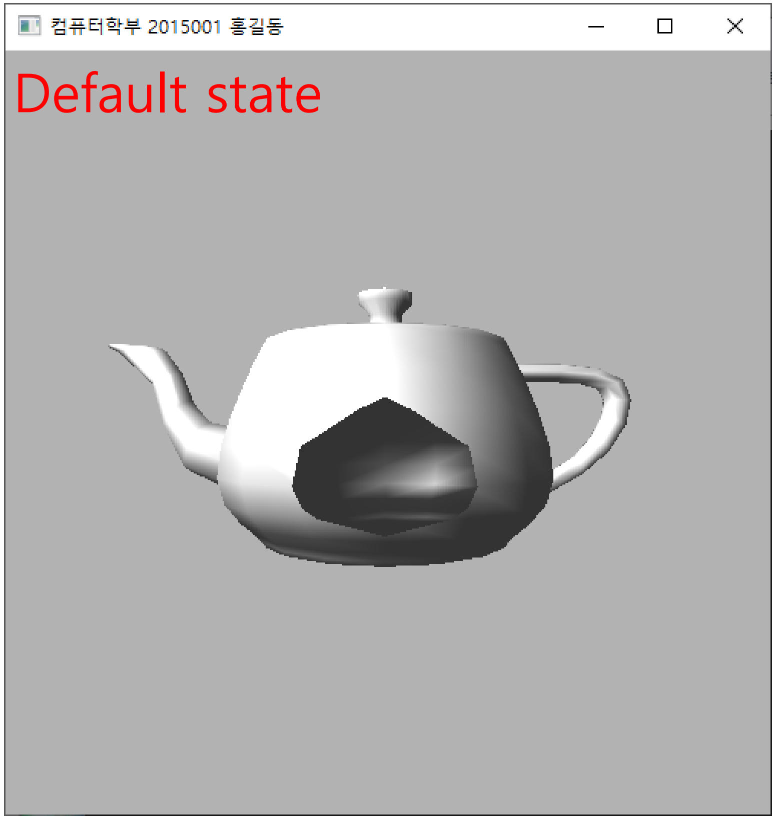
Back-Face Test

- The test for culling a back-facing polygon
- θ : the angle between the normal and the viewer
  - the polygon is facing forward if and only if $-90^\circ \le \theta \le 90^\circ$ or, equivalently, $\cos\theta \ge 0$.
- $\cos\theta \ge 0$ is easy to test
  - the dot product: $n \cdot v \ge 0$.

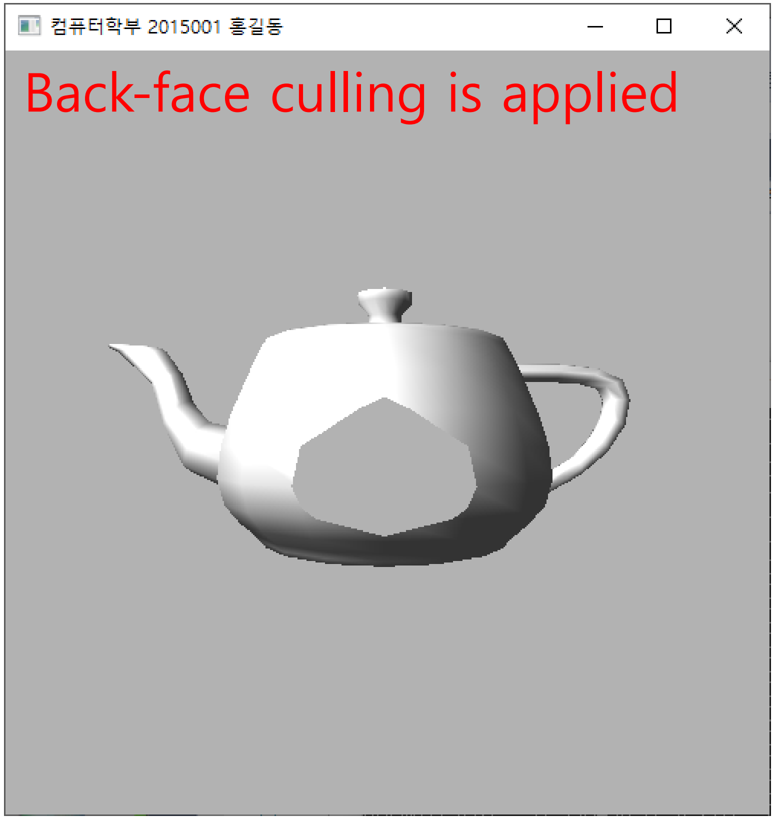


Back-Face Culling Example

- Back-face culling commands
  - glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);



Default state



Back-face culling is applied

Quiz #3. LEC17-LEC20

- Due date: 5/23(Sun) 6:00pm (2 hours)
  - LEC17. Viewing-part1
  - LEC18. Viewing-part2
  - LEC19. Normalization & Orthogonal Projection
  - LEC20. Perspective Projection