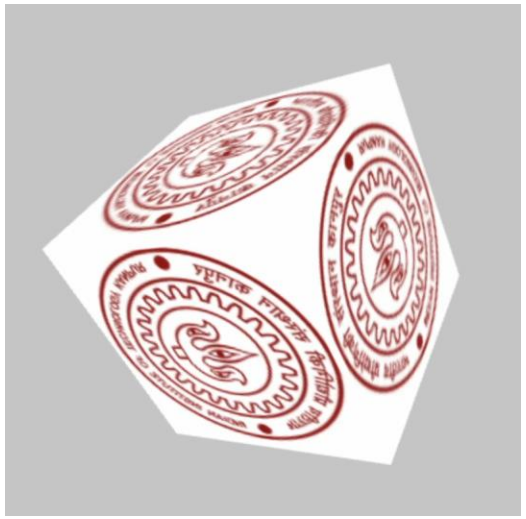


Introduction to Computer Graphics (CS360A)

Instructor: Soumya Dutta

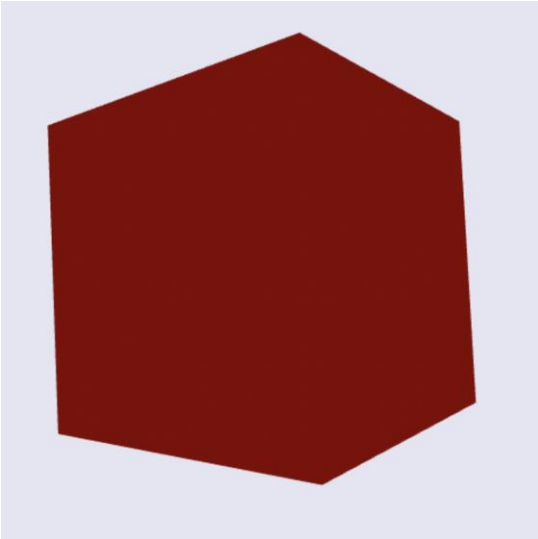
Department of Computer Science and Engineering

Indian Institute of Technology Kanpur (IITK)

email: soumyad@cse.iitk.ac.in

Demo for 3D Viewing and Projection

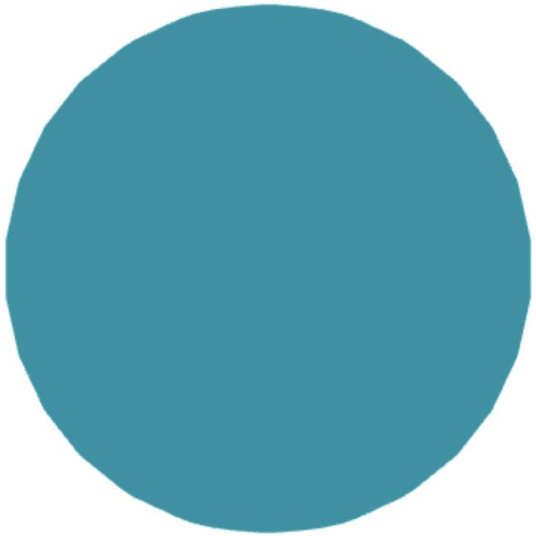
Simple3DCubeRender.js, Simple3DCubeRender.html



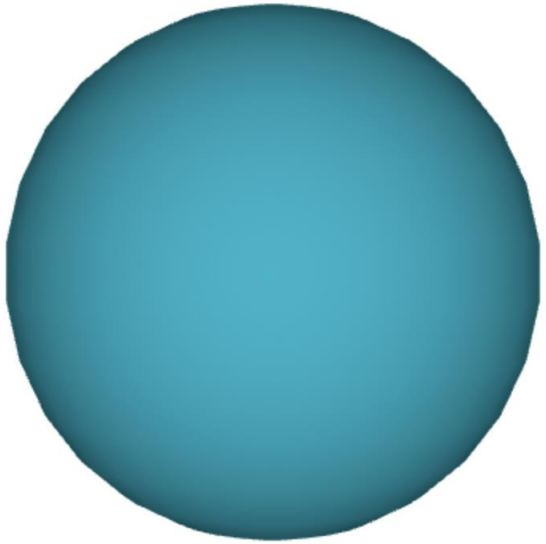
Code shared in HelloIITK

Shading

- Shading is the process of computing final color for each pixel considering its color, opacity, location of the viewer, distance and direction of the light, etc.



Flat 2D (but actually a 3D sphere)



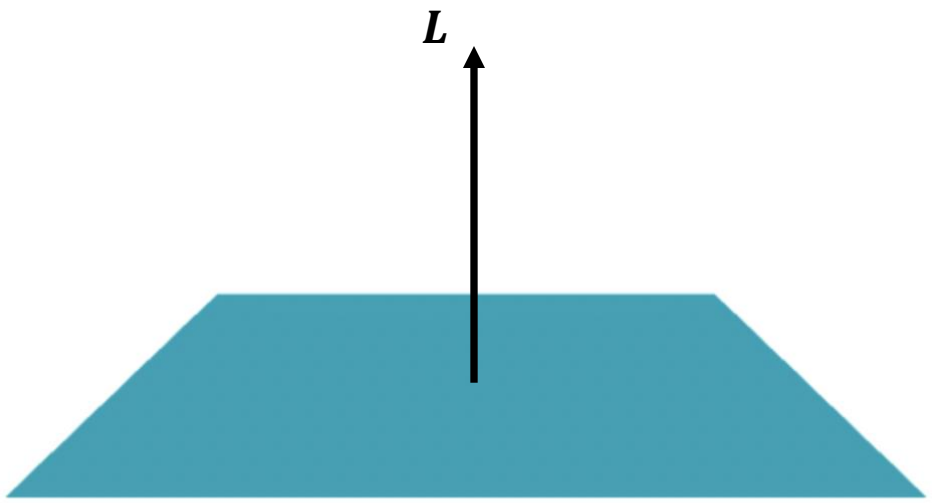
3D Look and feel

Shading

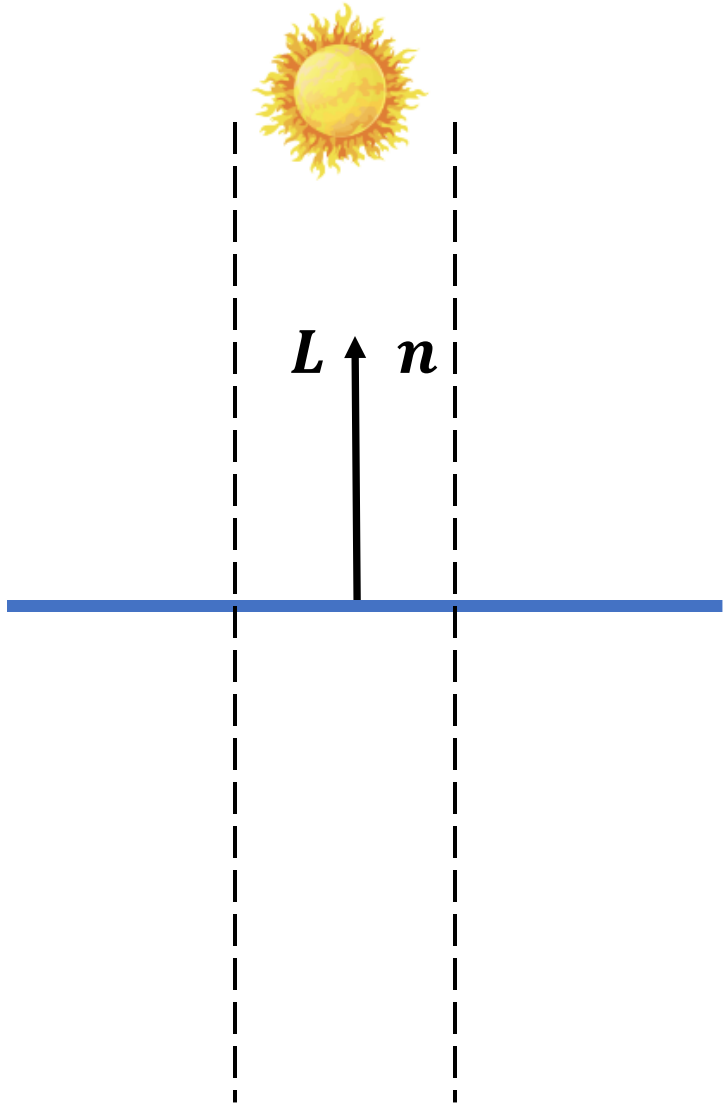
Shading

Shading

Shading

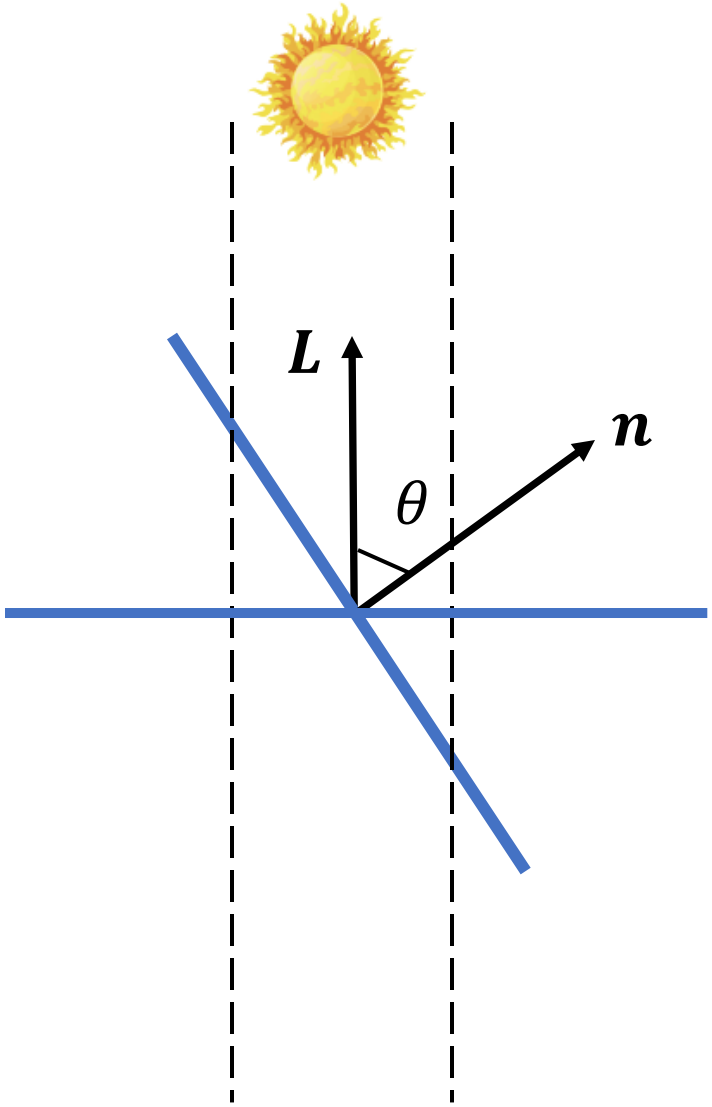


Shading

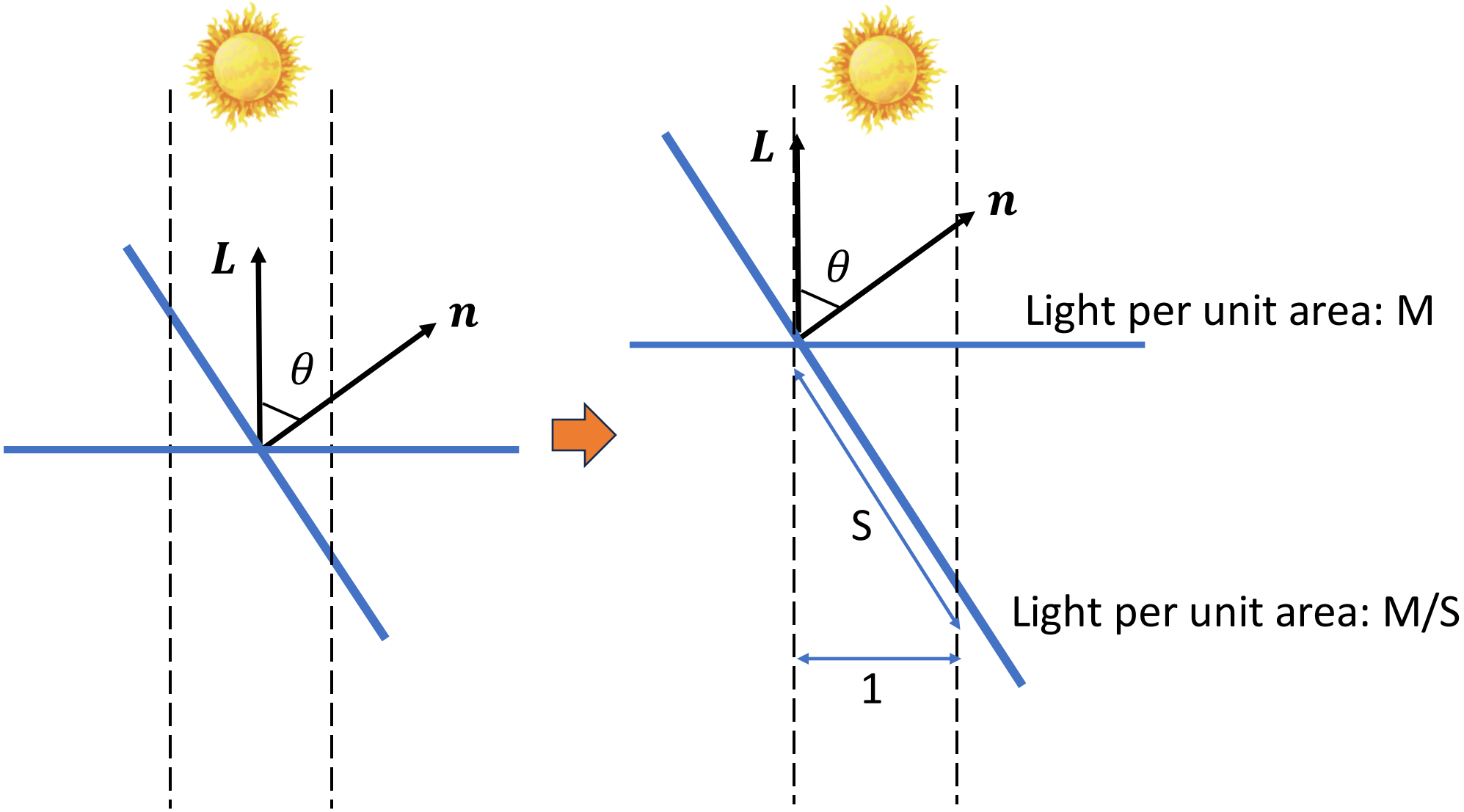


Looking at the surface from the side view

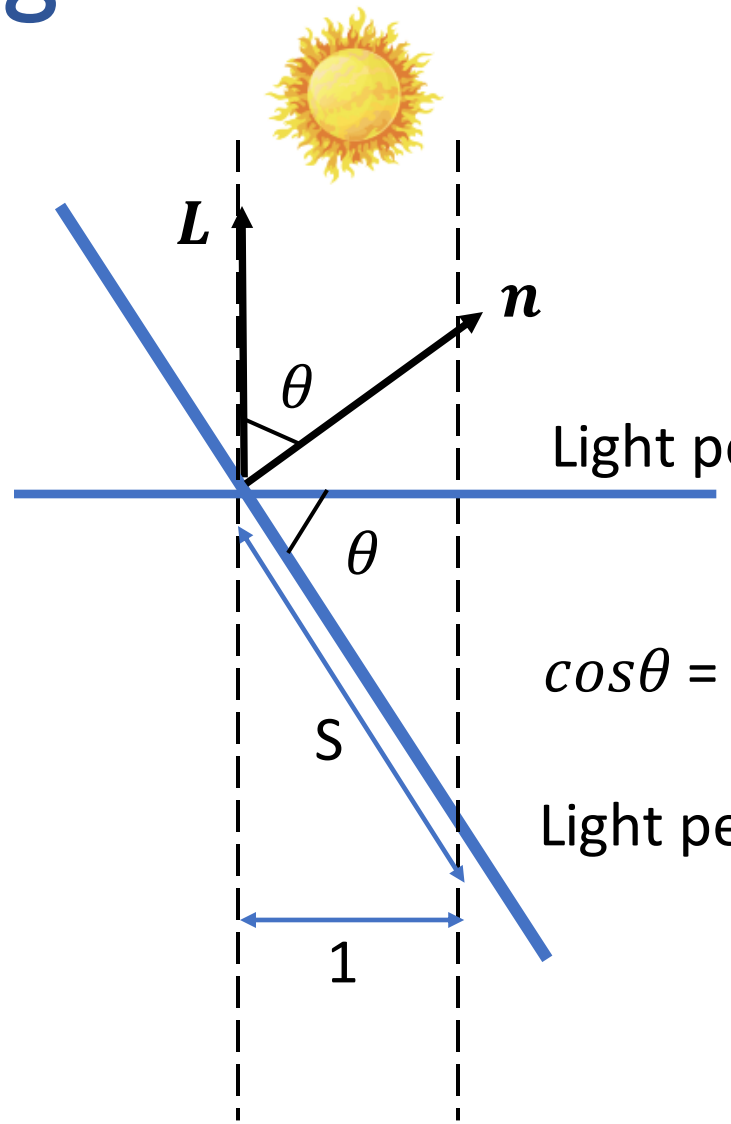
Shading



Shading



Shading

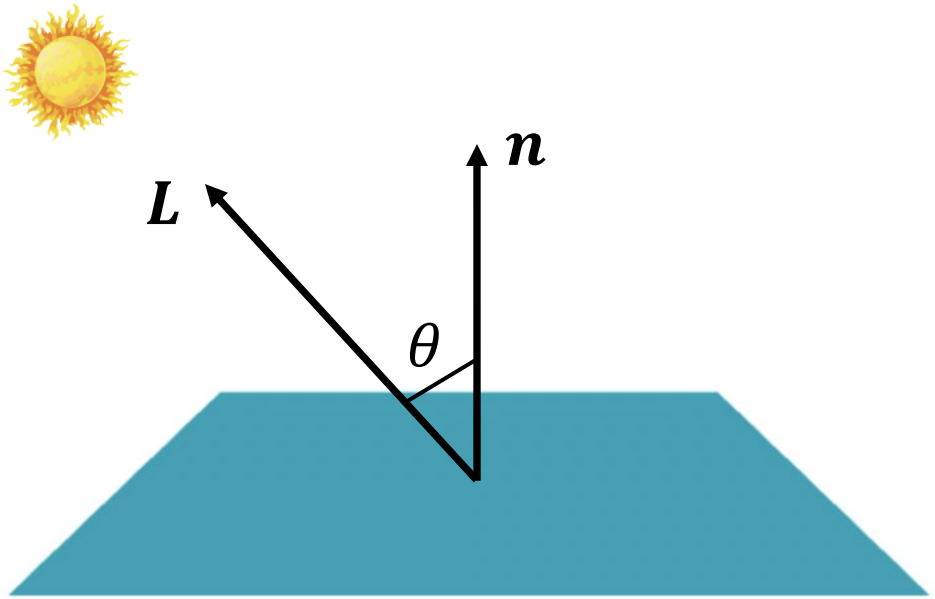


Light per unit area: M

$cos\theta$ = 1/S

Light per unit area: M/S = M $cos\theta$

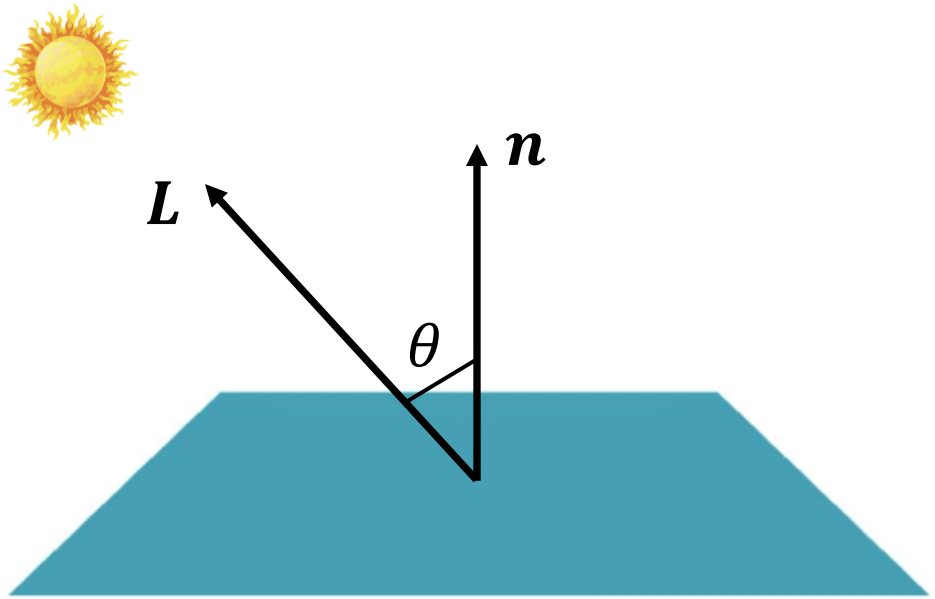
Lambertian Diffuse Material and Lighting



$K_d \cos\theta$

K_d =

Lambertian Diffuse Material and Lighting



$K_d\, cos\theta = K_d(\boldsymbol{n}\,.\,\boldsymbol{L})$

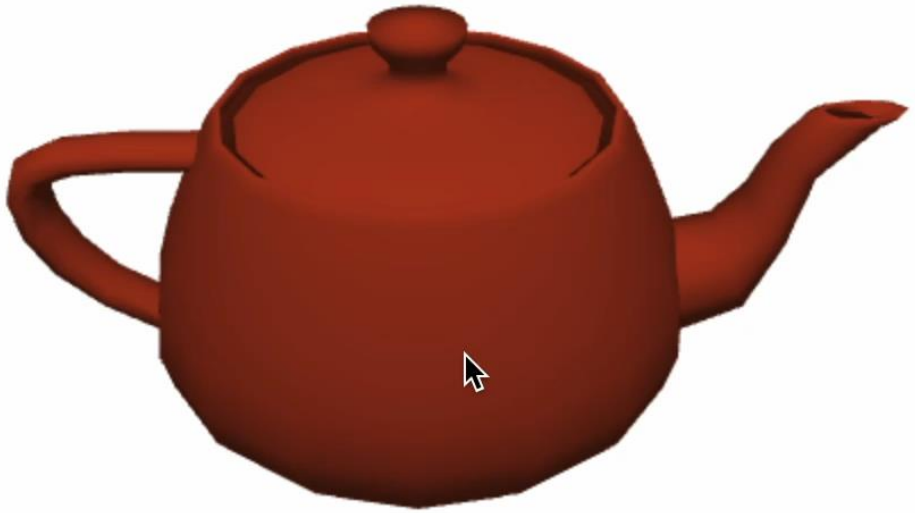
$C_d = IKdcos\theta$

I = Light Intensity

$\boldsymbol{n}$ and $\boldsymbol{L}$ are unit vectors

K_d =

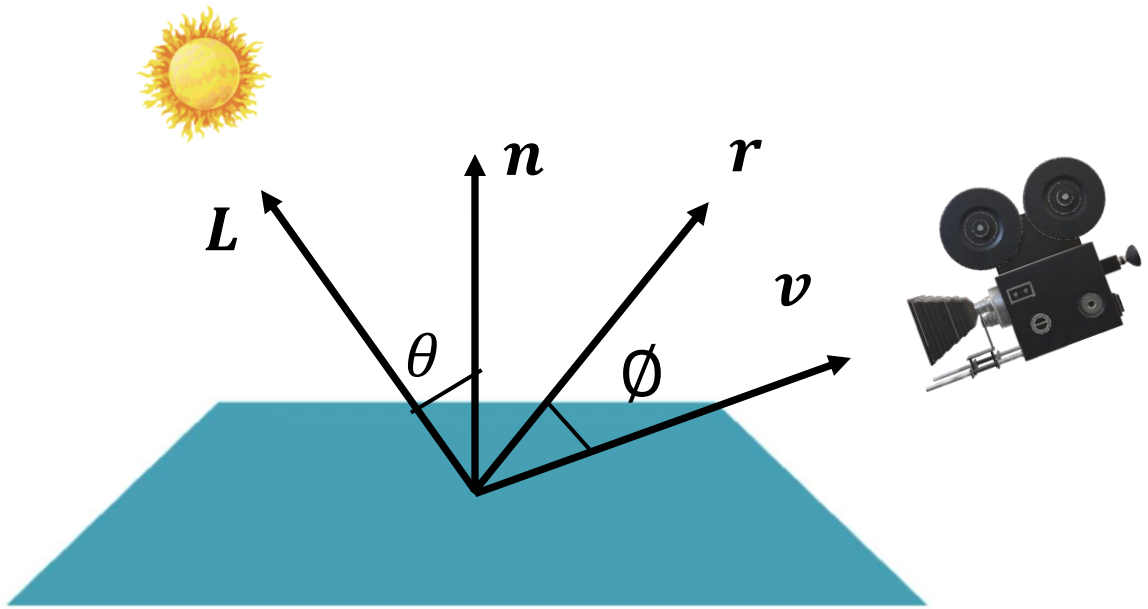
Specular Reflection



Diffuse Only

Diffuse + Specular

Phong Specular Reflection Lighting



$$(cos\emptyset)^{\alpha} = (\boldsymbol{v} \,.\boldsymbol{r})^{\alpha}$$

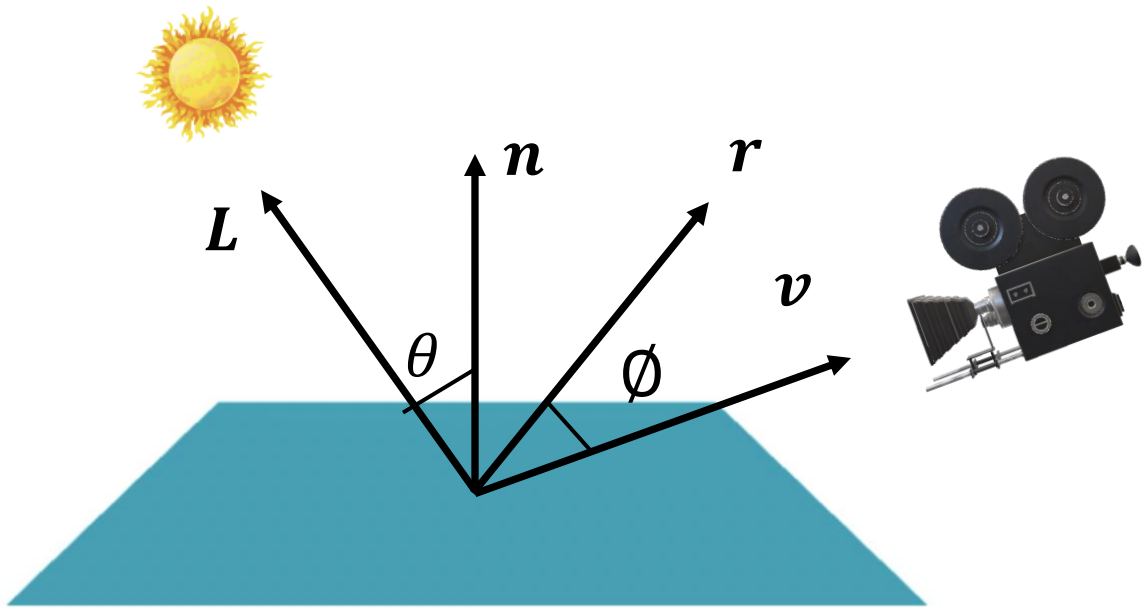
$$K_s(cos\emptyset)^{\alpha} = K_s(\boldsymbol{v} \,.\boldsymbol{r})^{\alpha}$$

$$C_s = IKs(\boldsymbol{v} \,.\boldsymbol{r})^{\alpha}$$

I = Light Intensity

Specular Reflections are reflections of light source

Phong Lighting/Illumination Model



$$C = Cd + Cs$$

$$C = IK_d cos\theta + IK_s(\boldsymbol{v}\,.\boldsymbol{r})^{\alpha}$$

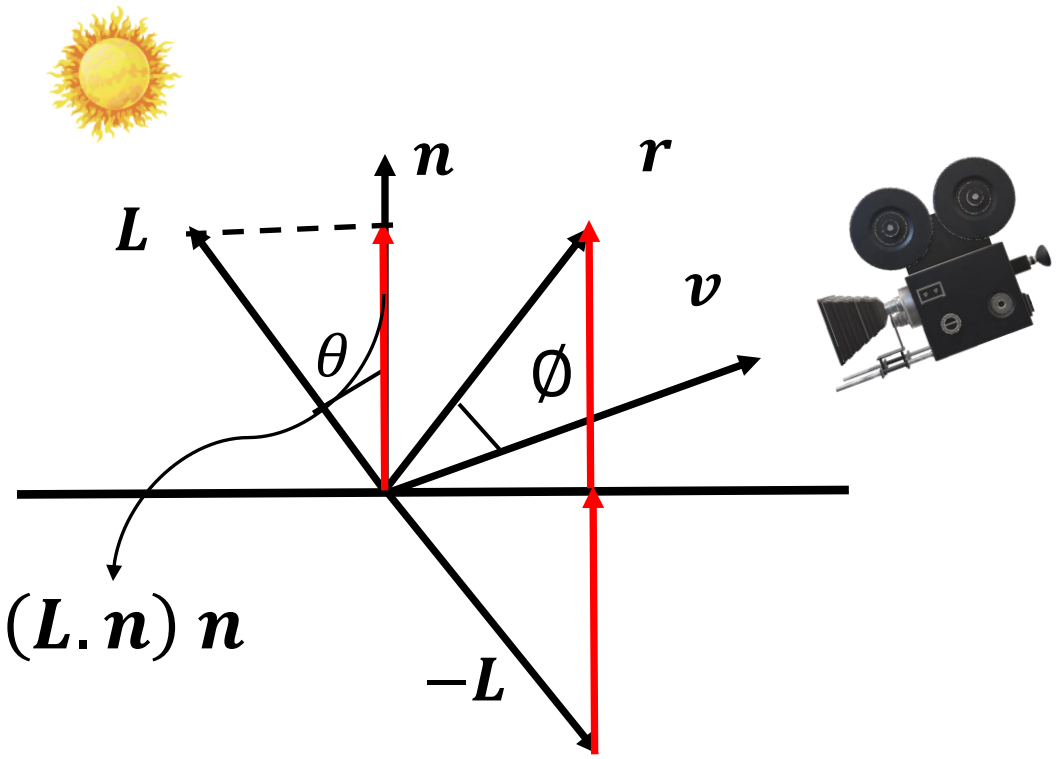
$$C = IK_d cos\theta + IK_s(cos\emptyset)^{\alpha}$$

Handle the case when light is coming from behind

$$C = IK_d max(0, cos\theta) + IK_s(cos\emptyset)^{\alpha}$$

Now, how do we get the reflection vector $\boldsymbol{r}$?

Phong Lighting/Illumination Model

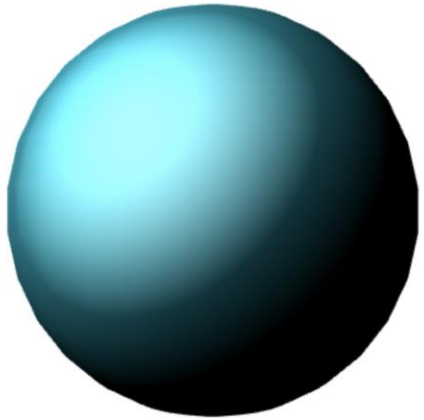


Compute reflection vector ($\boldsymbol{r}$)

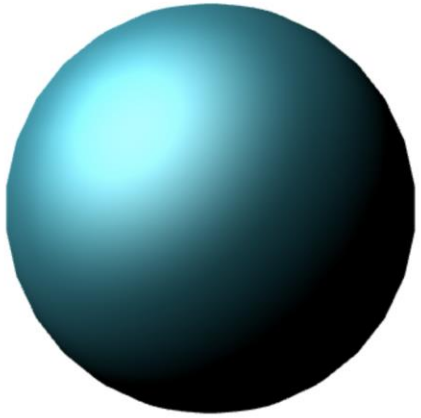
$$\boldsymbol{r} = 2(\boldsymbol{L}.\boldsymbol{n})\,\boldsymbol{n} - \boldsymbol{L}$$

Phong Lighting/Illumination Model

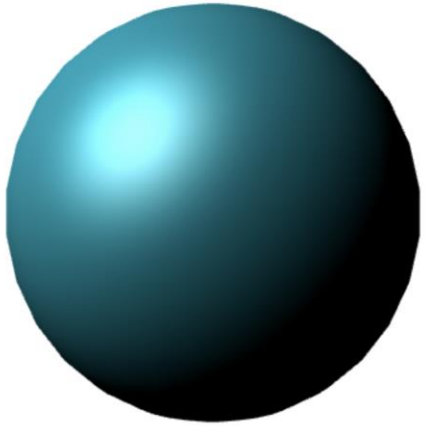
α =1



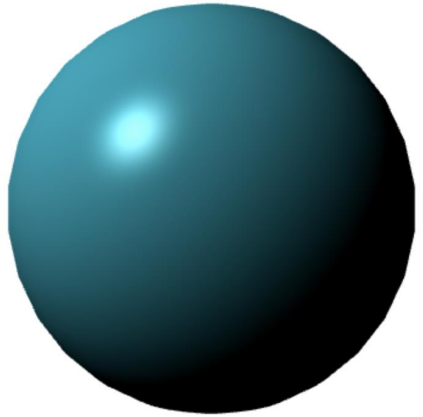
α =2



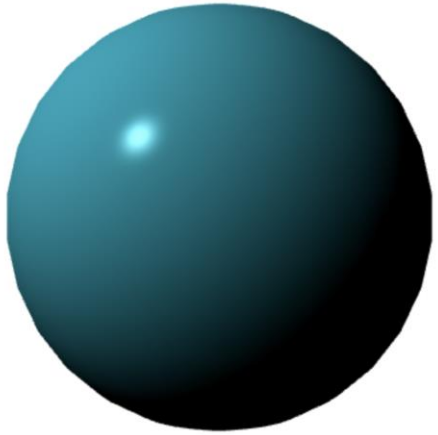
α =5



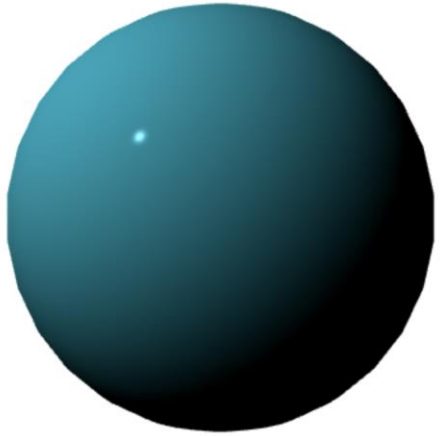
α =25



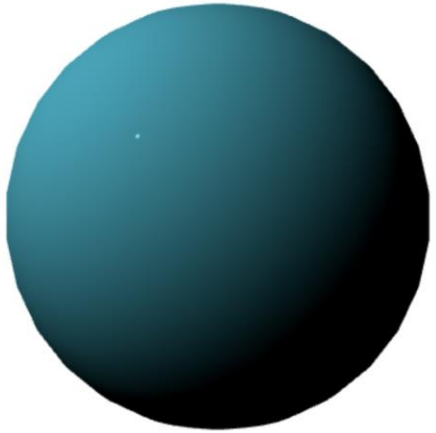
α =100



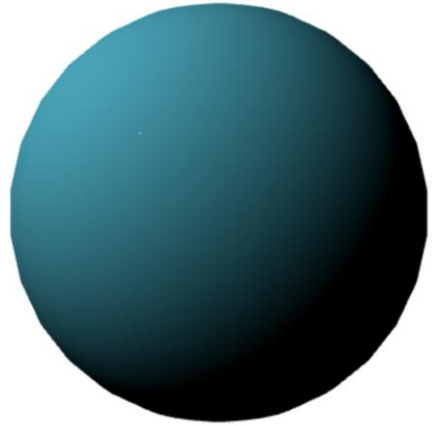
α =1000



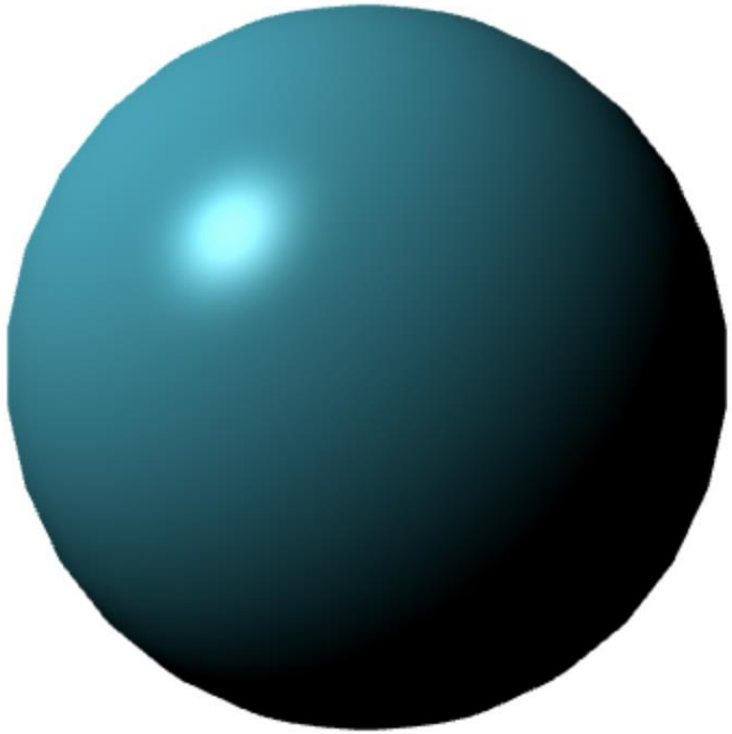
α =10000



α =50000

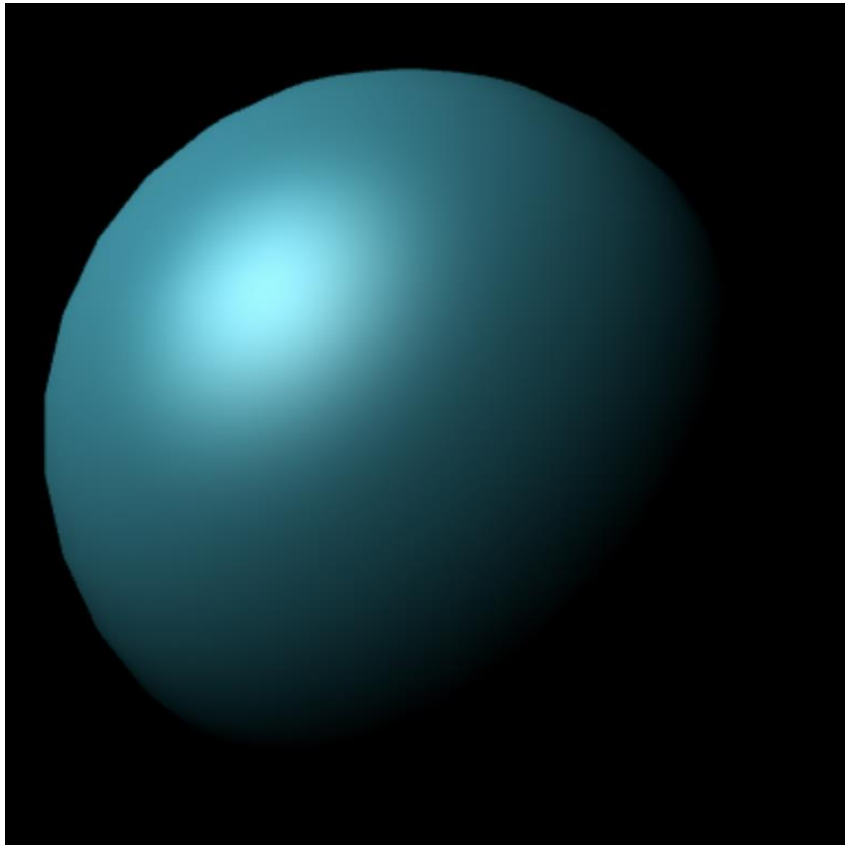


Ambient Light



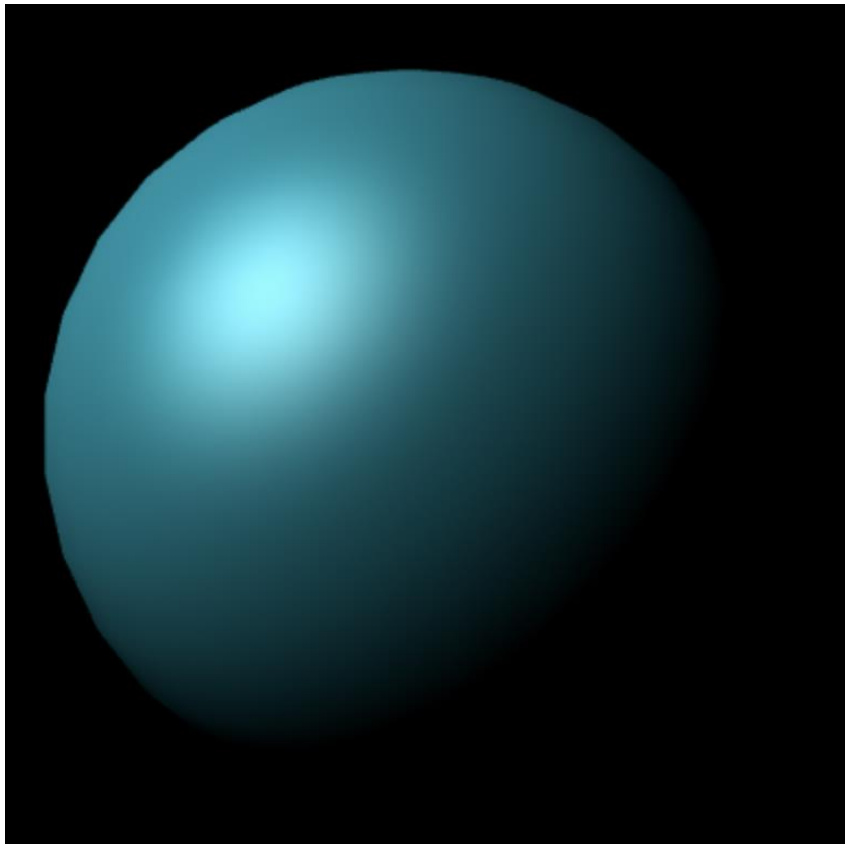
- With this light model, the far side of the object remains very dark
- If we have the object in a black background, the backside is invisible

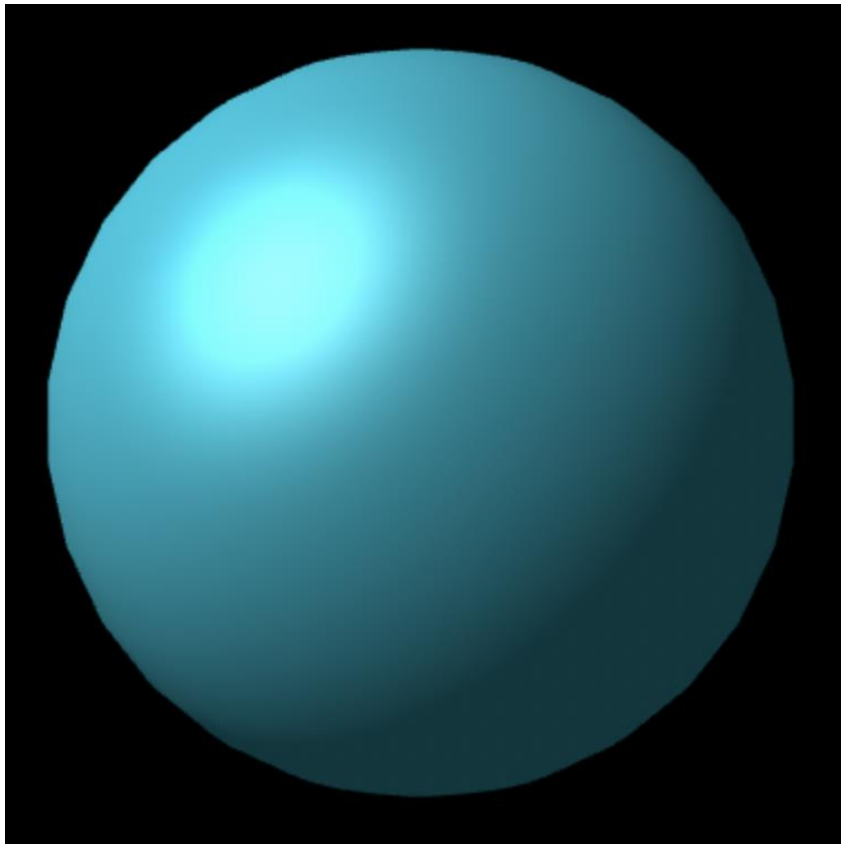
Ambient Light



- With this light model, the far side of the object remains very dark
- If we have the object in a black background, the backside is invisible
- In reality, there is always some global light that will allow us to see the back side of the sphere
- How do we simulate that?

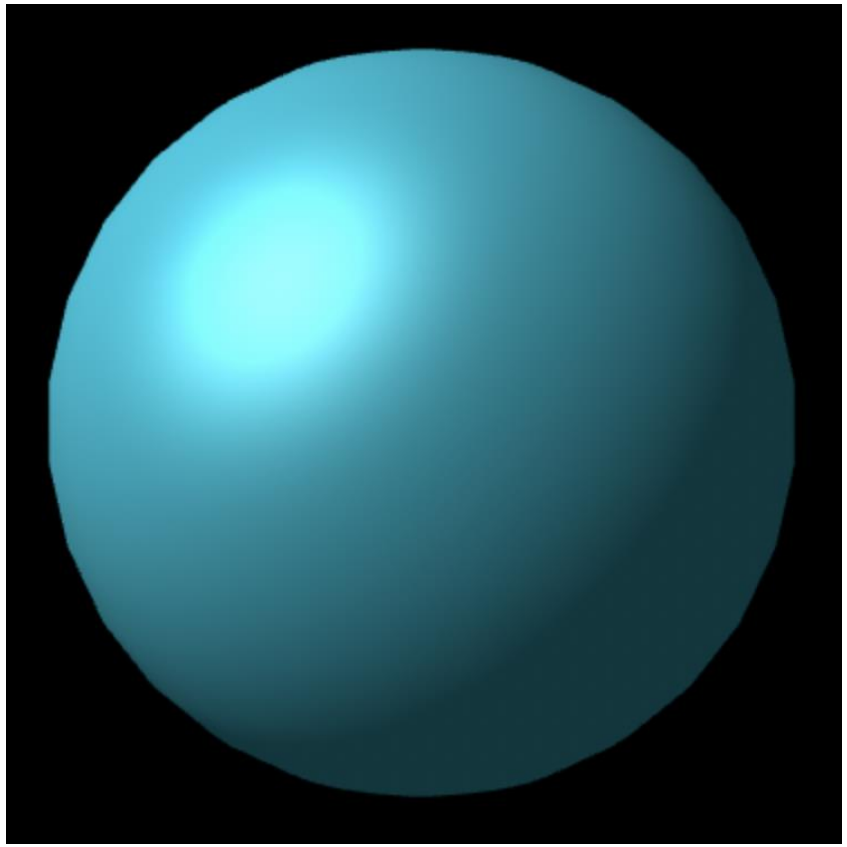
Ambient Light





- Add a small constant light contribution to our object, called ambient light
- An easy approximation of global lighting

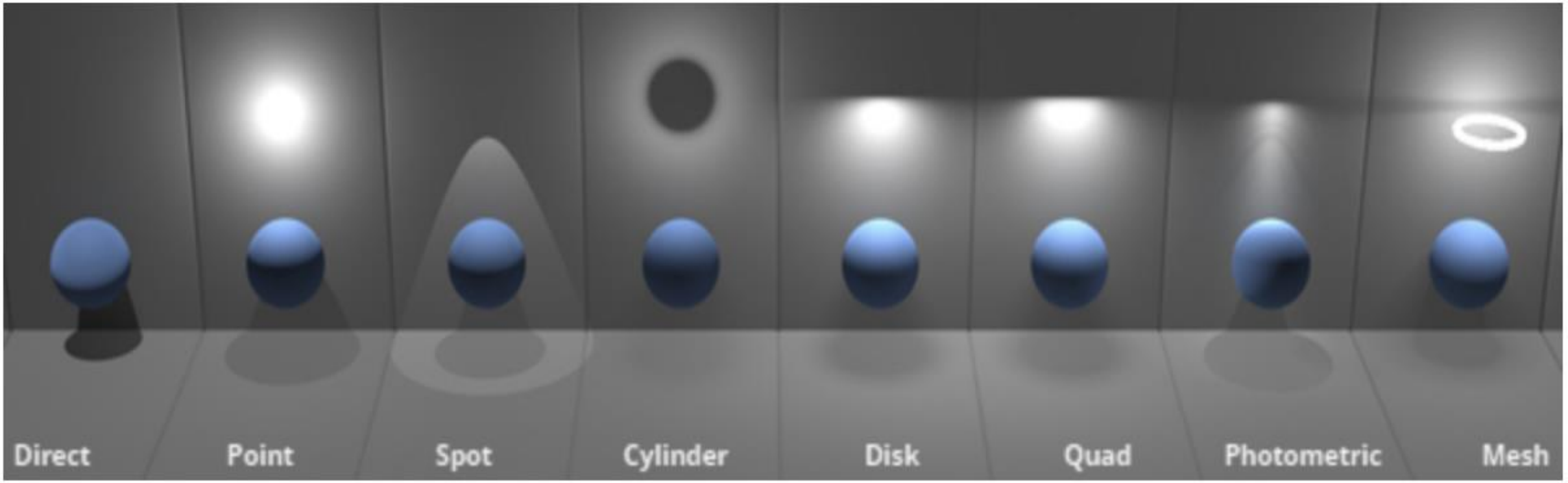
The Complete Phong Model



$$C = IK_d max(0, cos\theta) + IK_s(cos\emptyset)^{\alpha} + \mathrm{IK_a}$$

Most of the time, we assume ambient color is same as diffuse color

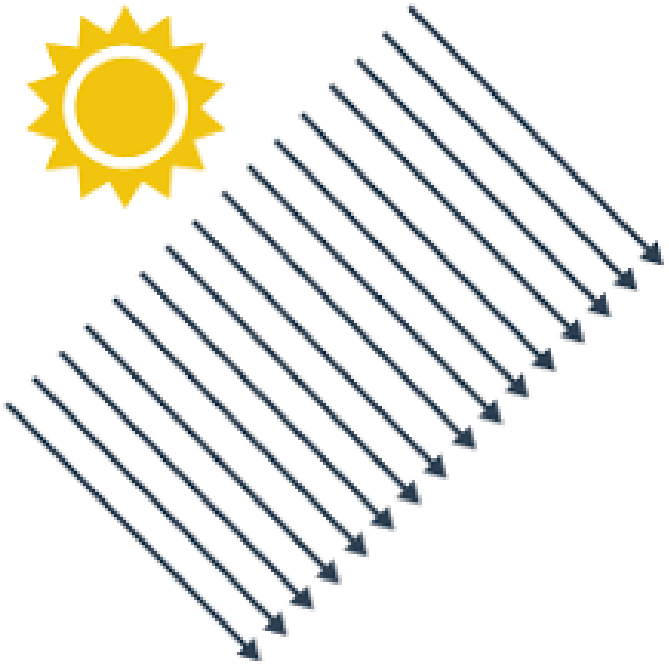
Types of Lights



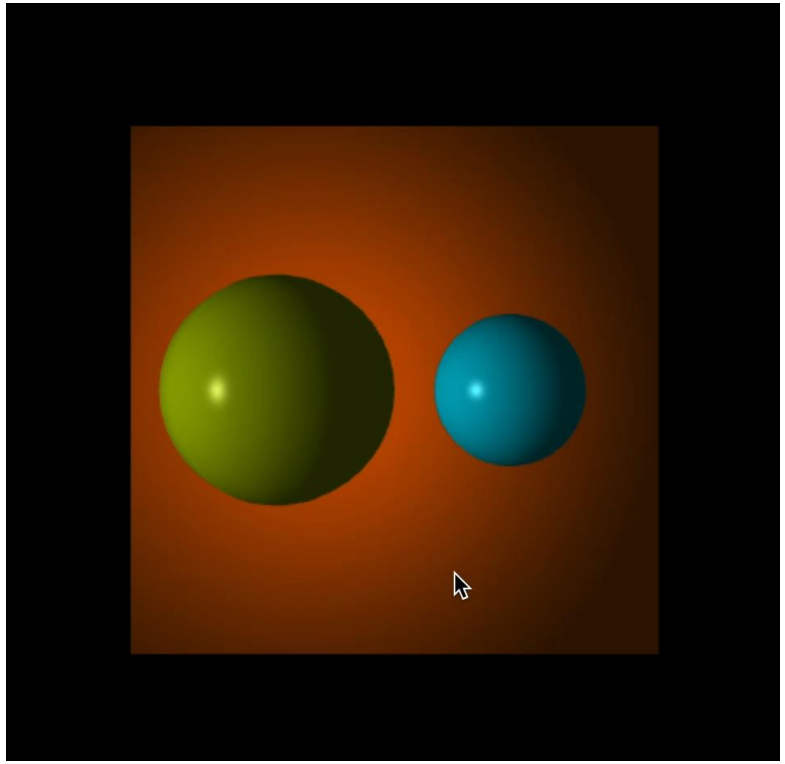
Different types of lights used in Computer Graphics

Directional Light

- Simplest light model
- We assume light is coming from some direction
- Light also has a location

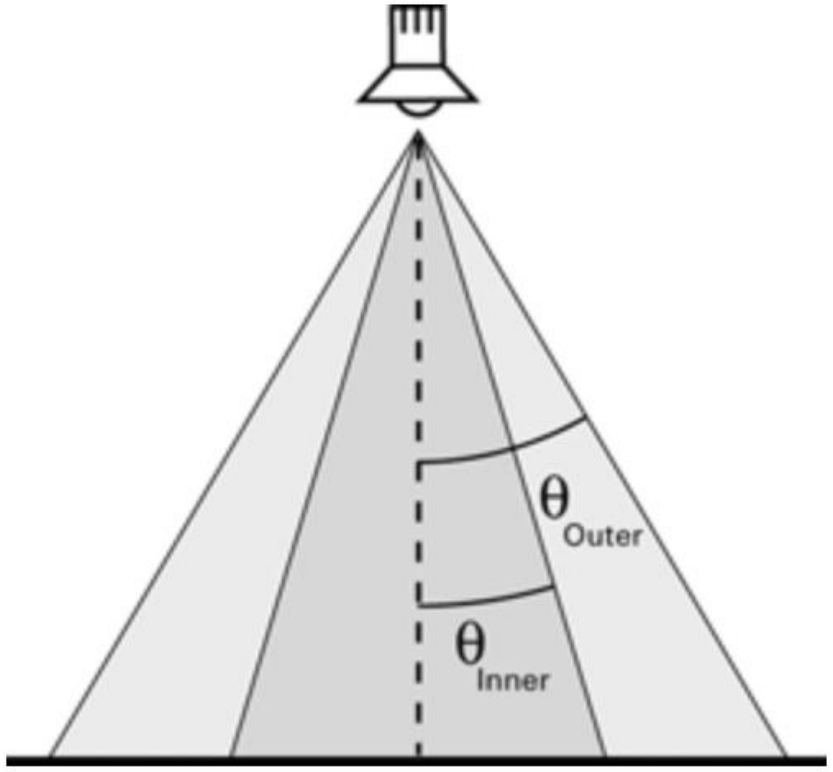


Directional Light

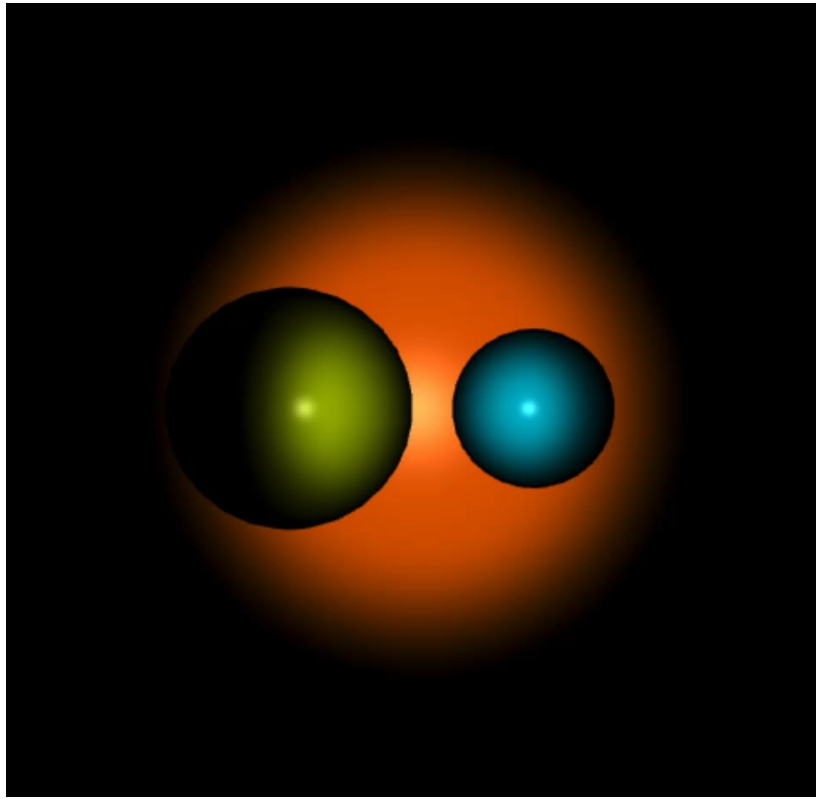


Directional Light

Spotlight



Spotlight



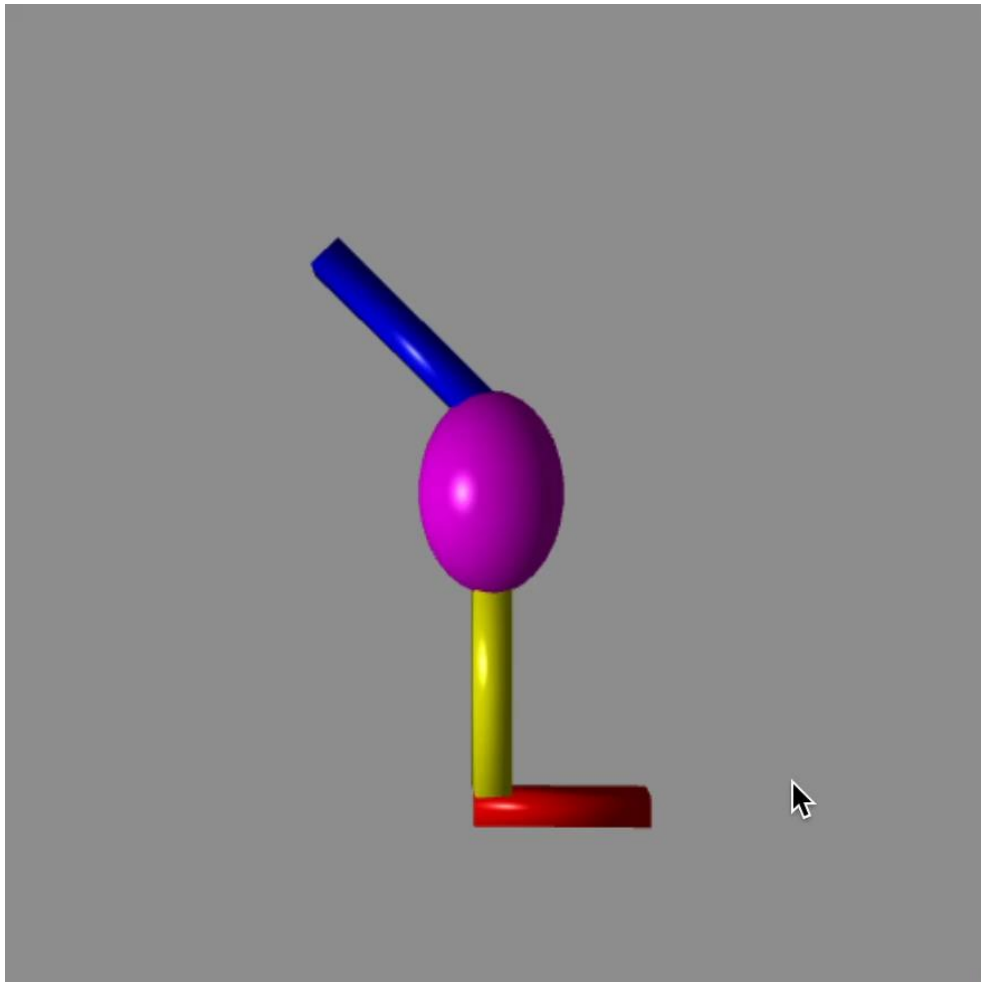
Spotlight

Image-Based Lighting

Source: Nvidia

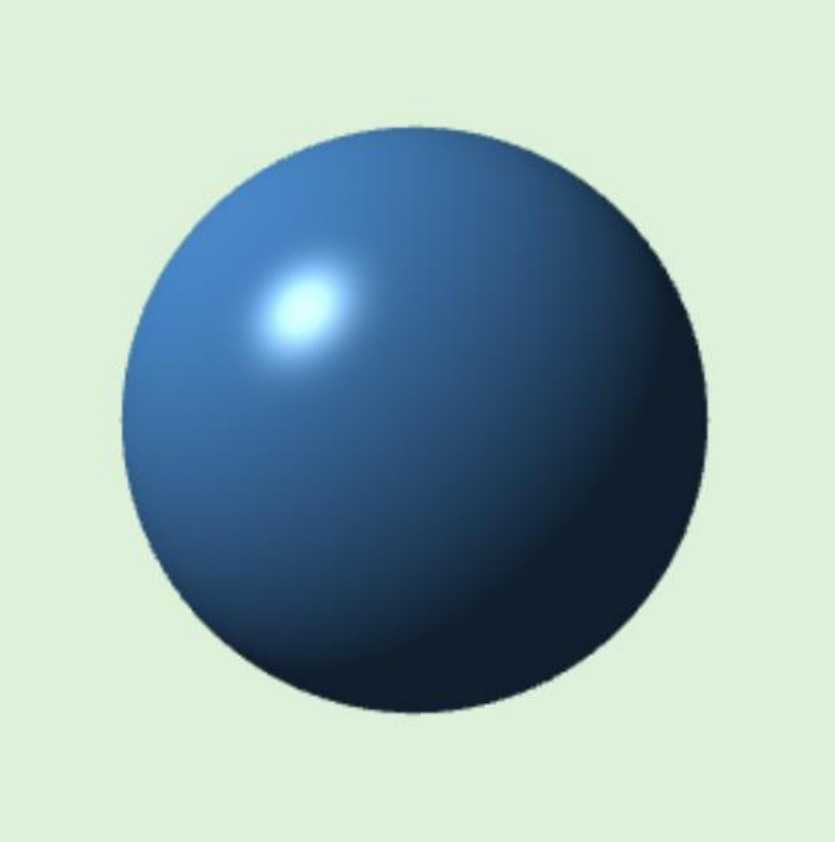
Dealing With Multiple Lights

- Simply add contributions of each light to compute the final color



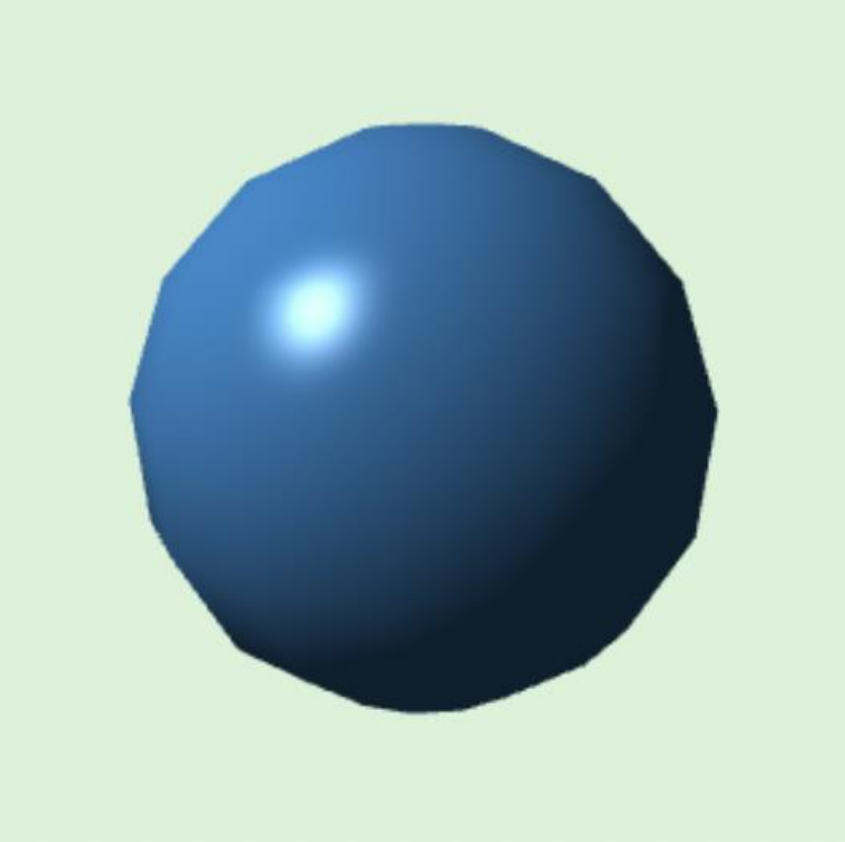
Shading Types

Smooth Shading



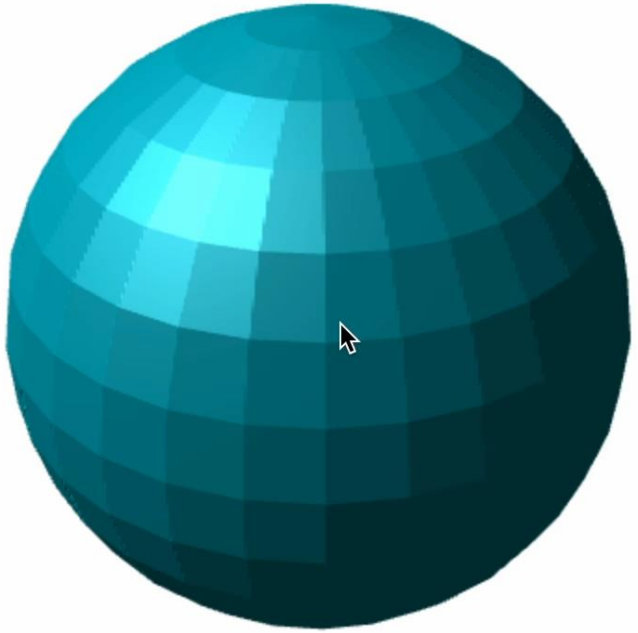
Implicit Surface

Sphere made from an equation



Discrete Surface Mesh

Different Types of Shading



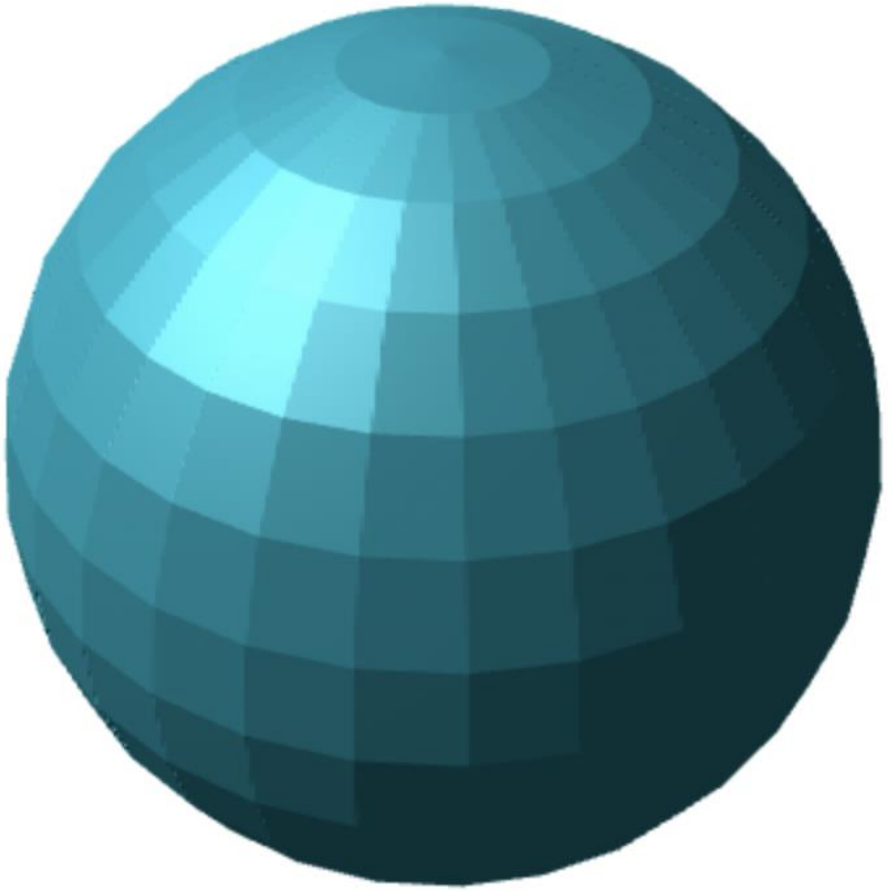
Change Background Color:

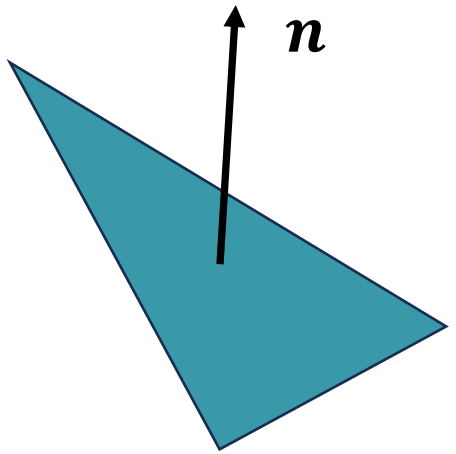
Grey | Black | White

Change Shading/Lighting Mode:

Flat (Per-Face) | Per-Vertex | Per-Fragment | Toon

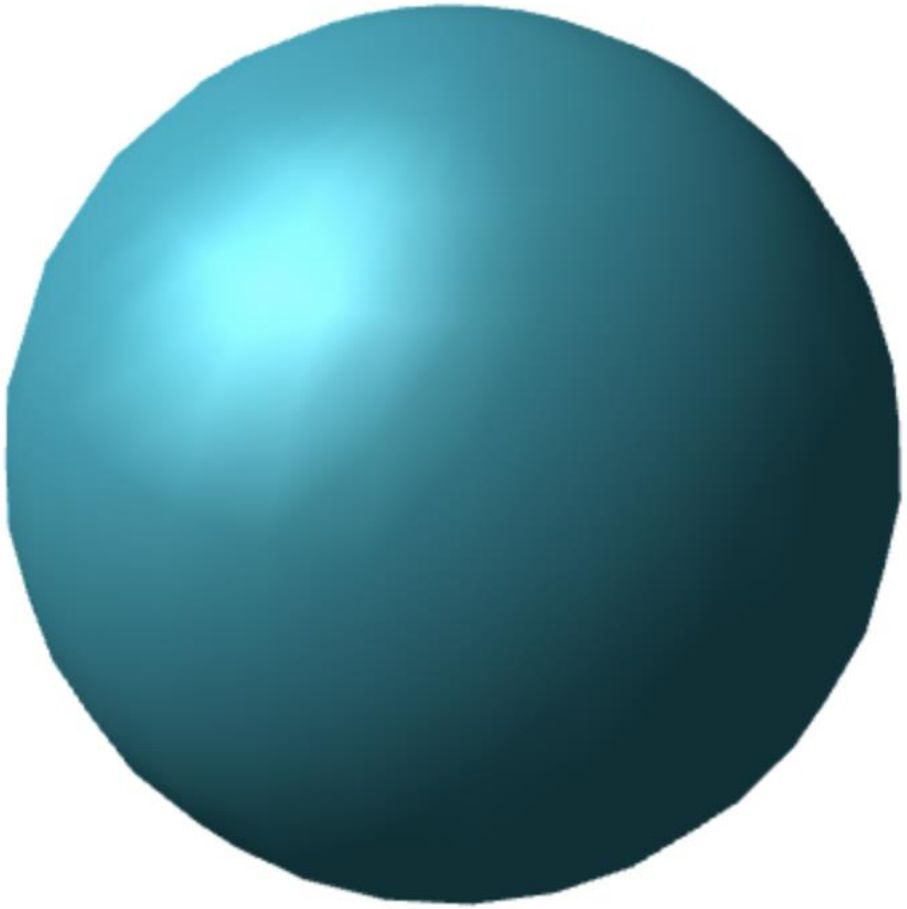
Flat Shading (Per Face Shading)

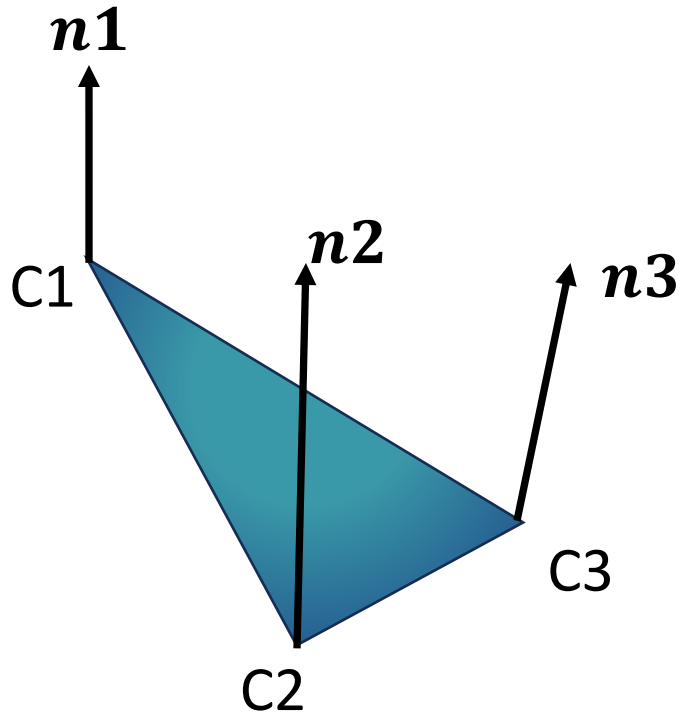




- Constant color inside a triangle
- How to compute normal of a face?

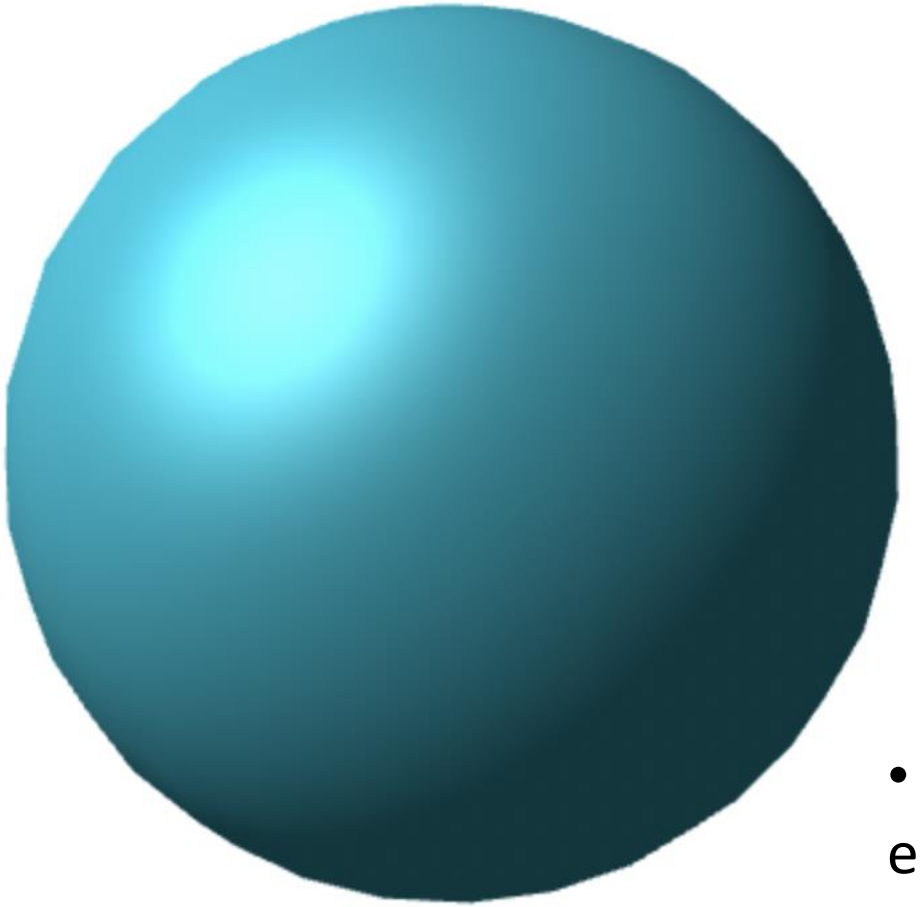
Gouraud Shading (Per Vertex Shading)

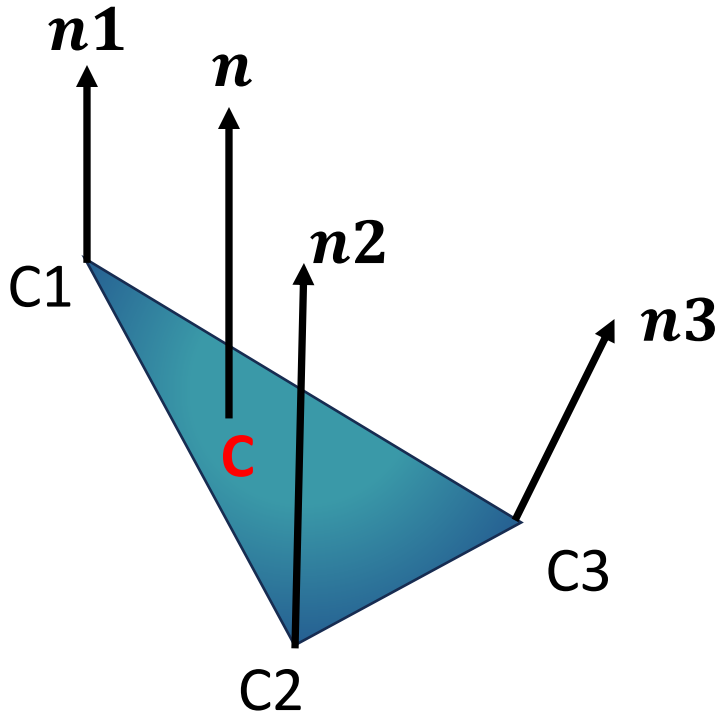




- Compute color at vertices in vertex shader
- Fragment shader will interpolate inside the triangle
- Fast process
- **This is how old graphics hardware shaded objects**

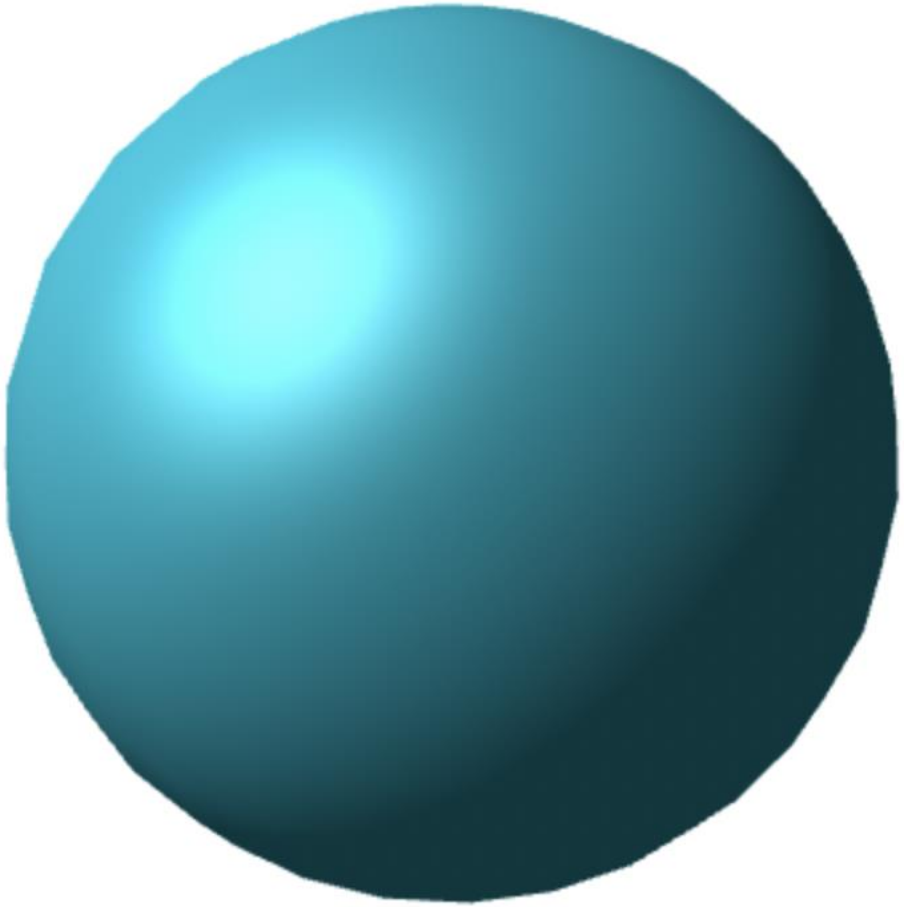
Phong Shading (Per Fragment Shading)

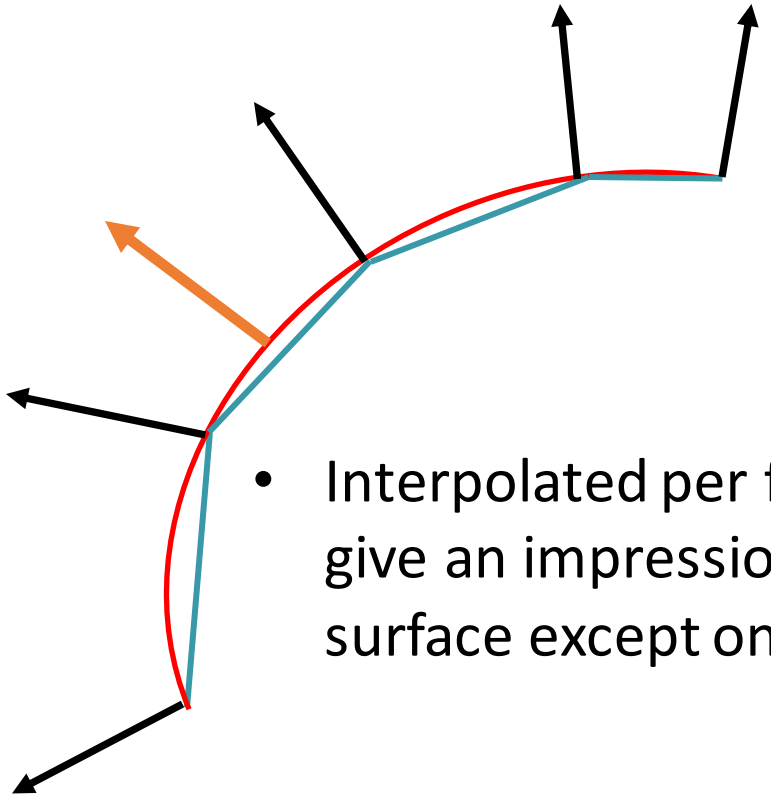




- Compute color at each fragment by finding normal at each fragment
- Normals are interpolated and normalized
- Computationally expensive

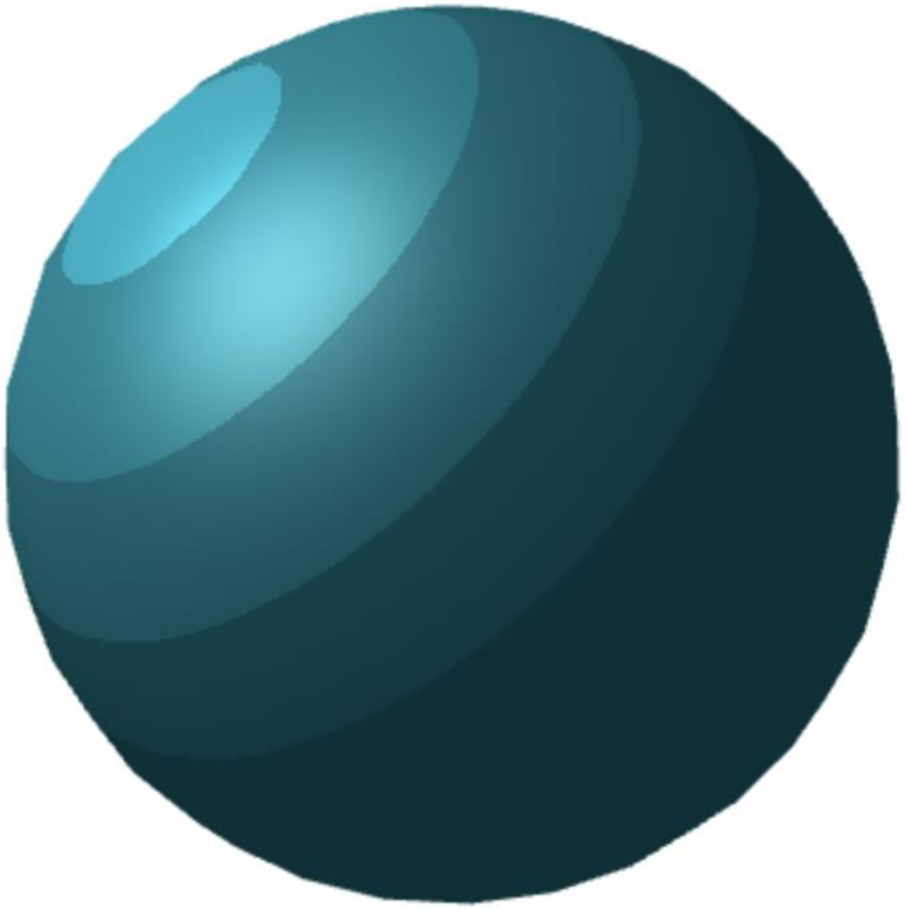
Phong Shading (Per Fragment Shading)





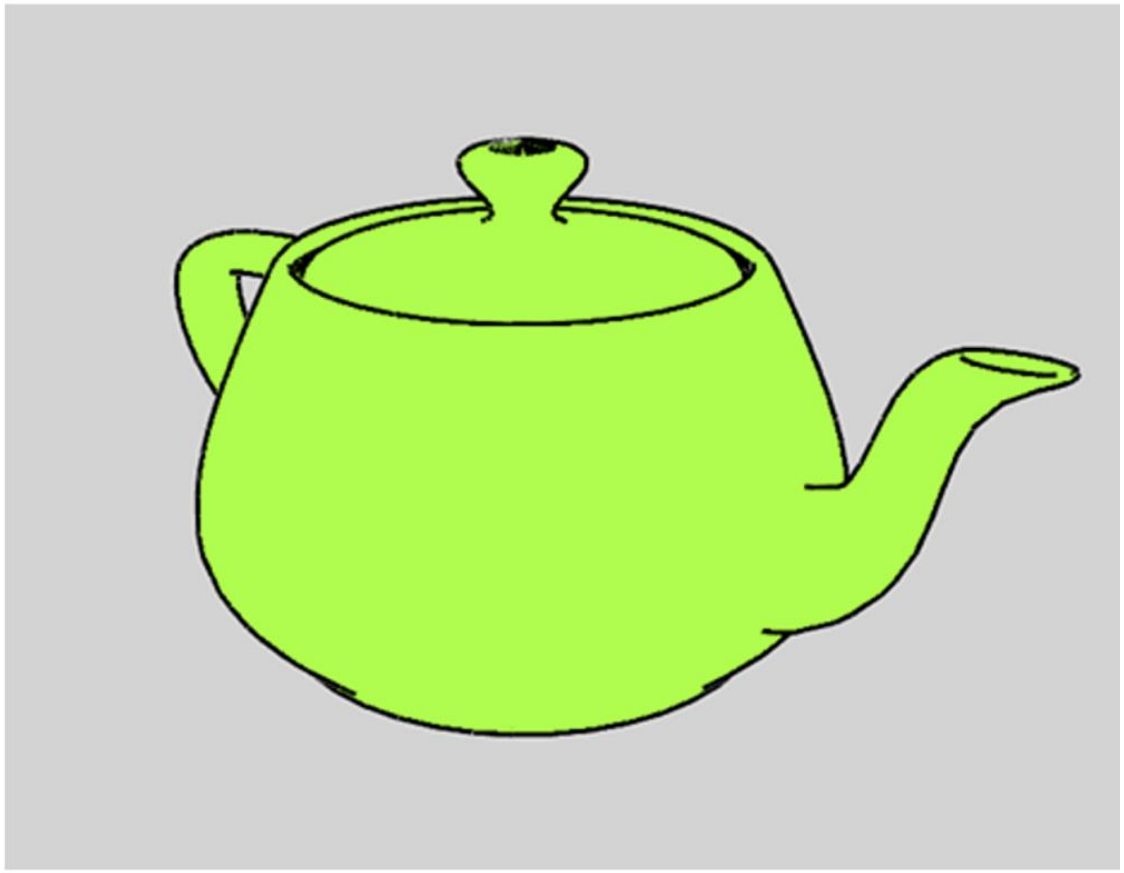
- Interpolated per fragment normals give an impression of smooth surface except on the edges

Toon (Cel) Shading



- Cartoon like shading effect
- Divide the color into several distinct bands so that cartoon like feel appears

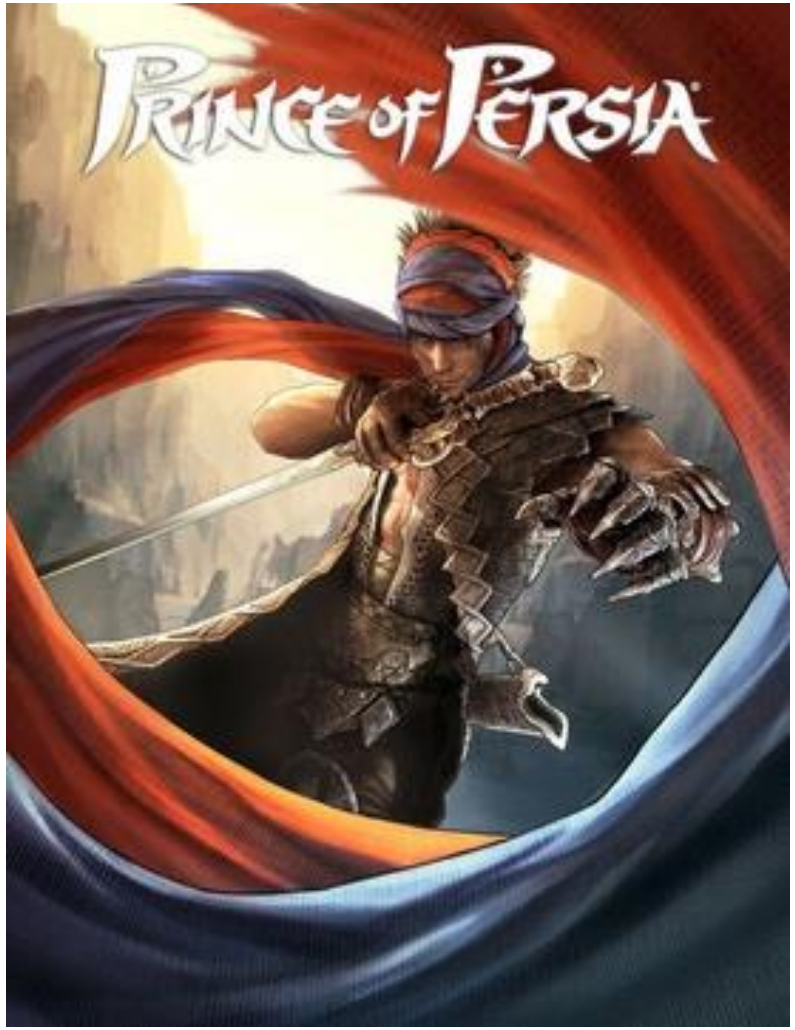
Toon (Cel) Shading

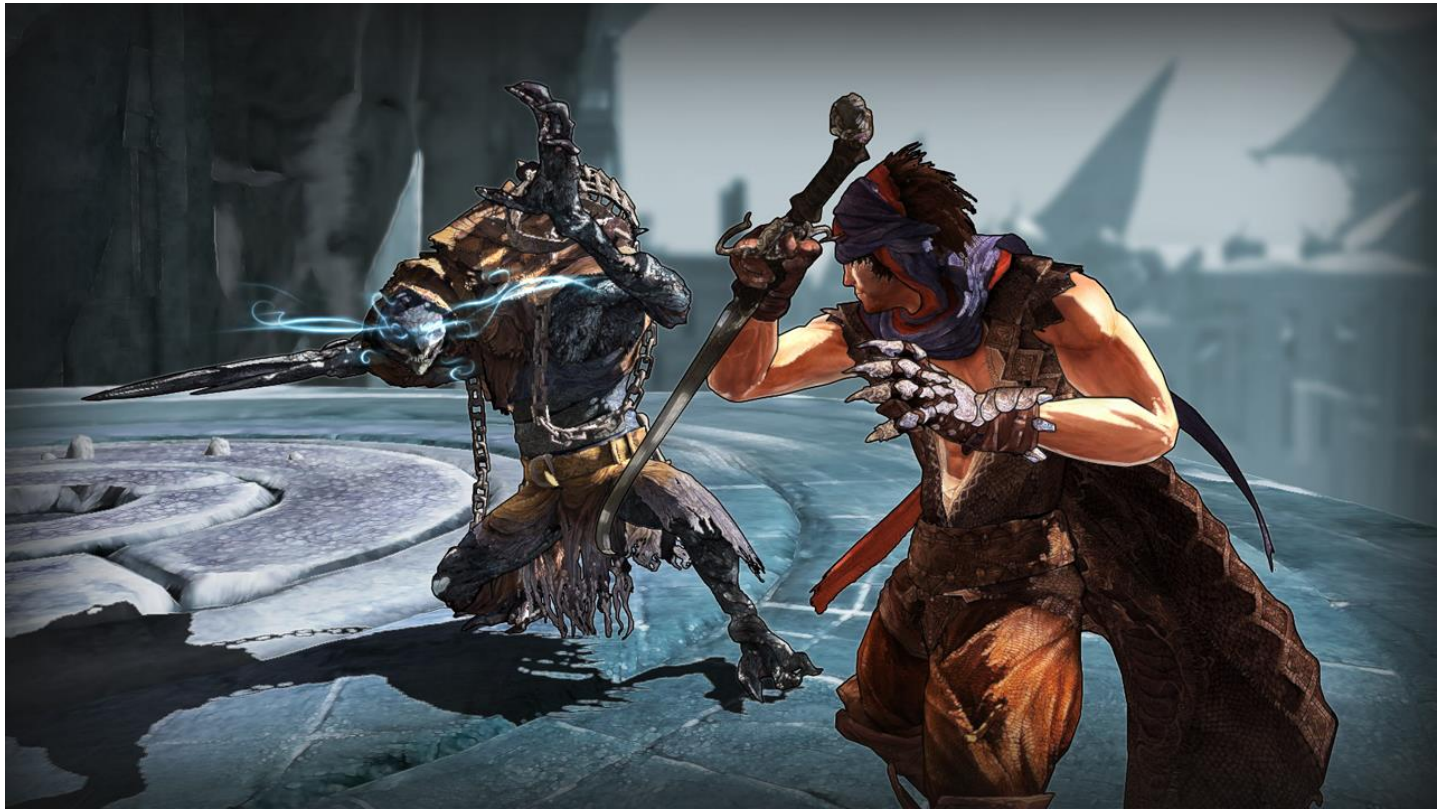


Single Color shading

Toon/Cel shading

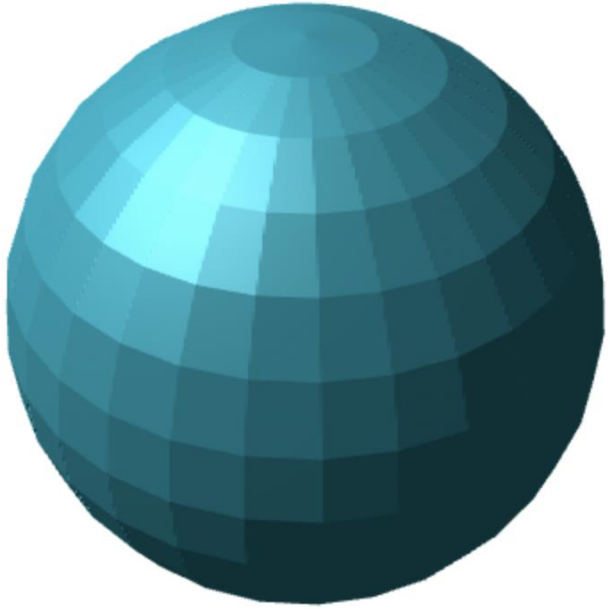
Toon (Cel) Shading



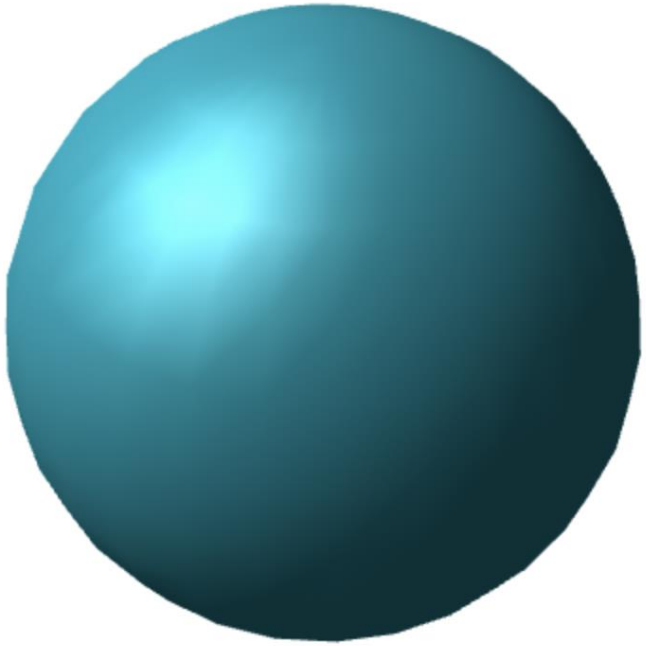


Ubisoft's Prince of Persia: The Two Thrones heavily used cel/toon shading

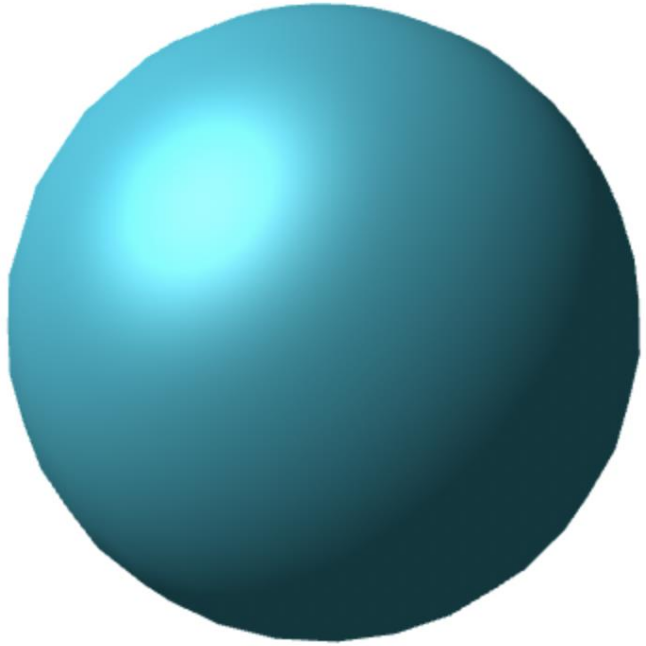
Smooth Shading Methods



Flat Shading



Gouraud Shading



Phong Shading