nvidia

Visual Computing

2024/2025

Class 6

Illumination and Shading

(and some recap)

deti departamento de electrónica, telecomunicações e informática

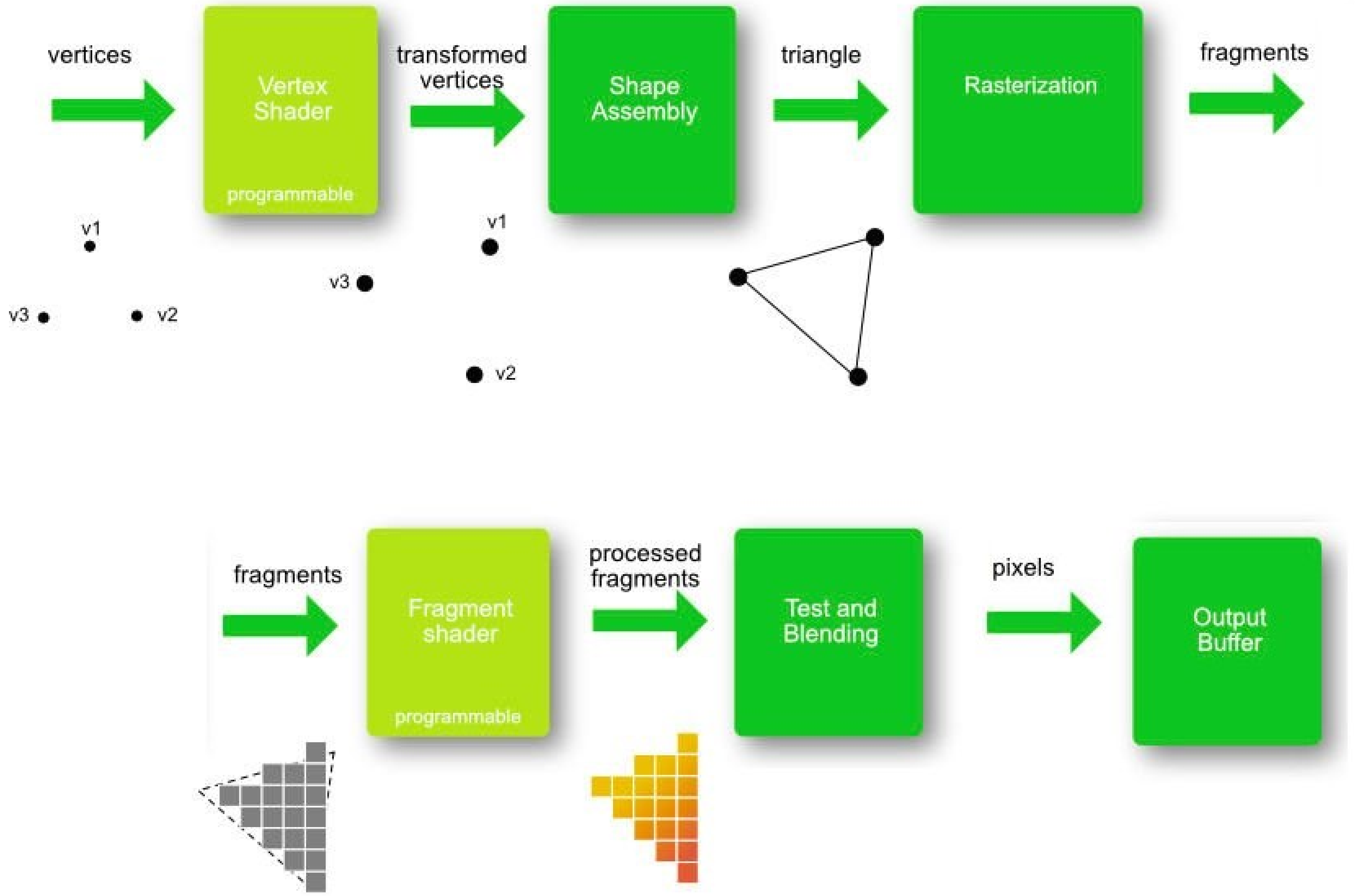
universidade de aveiro

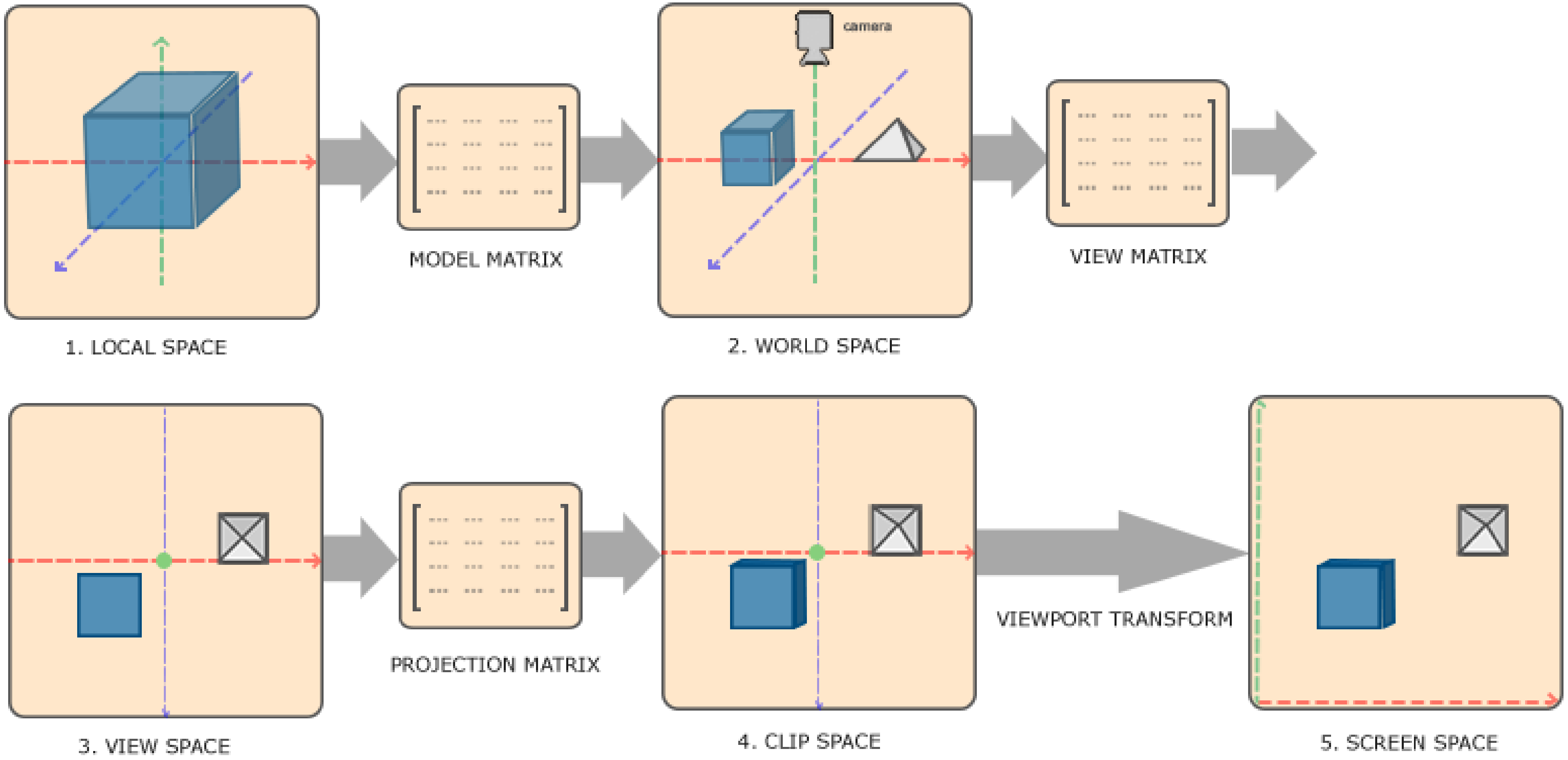
sss_vc_06_v2

Today’s Agenda

- Looking back for a summary
- Illumination
  - Local and Global Illumination
  - Light sources, materials
  - Phong’s Reflection Model
- Shading
  - Per-primitive shading (Flat)
  - Per-vertex shading (Gouraud)
  - Per-fragment shading (Phong)
- Hands On

The Graphics Pipeline


vertices
Vertex Shader
programmable
transformed vertices
Shape Assembly
triangle
Rasterization
fragments
v1
v3
v2
v1
v3
v2
fragments
Fragment shader
programmable
processed fragments
Test and Blending
pixels
Output Buffer


camera
MODEL MATRIX
VIEW MATRIX
1. LOCAL SPACE
2. WORLD SPACE
PROJECTION MATRIX
VIEWPORT TRANSFORM
3. VIEW SPACE
4. CLIP SPACE
5. SCREEN SPACE

Illumination and Shading

Lighting or Illumination

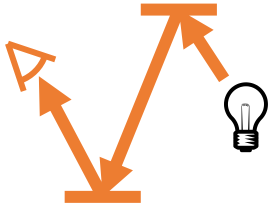
- The process of **computing** the **intensity** and **color** of a sample **point** in a scene as seen by a **viewer**
- It is a function of the **geometry** of the scene
  - Models, lights, camera, and their spatial relationships
- And of **material** properties
  - Reflection, absorption, …

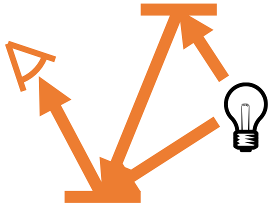
Shading

- The process of **interpolation** of **color** at points **in-between** those with **known lighting** or illumination
  - **Vertices** of triangles in a mesh
- Used in many real time graphics applications (e.g., games)
  - Calculating illumination at a point is usually **expensive** !
- **BUT**, in **ray-tracing** only do lighting for samples
  - Based on pixels (or sub-pixel samples for super-sampling)
  - **No shading** rule

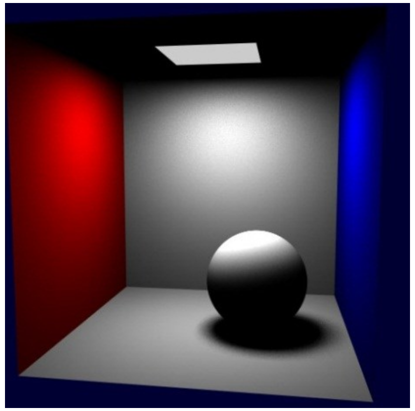
Computing illumination

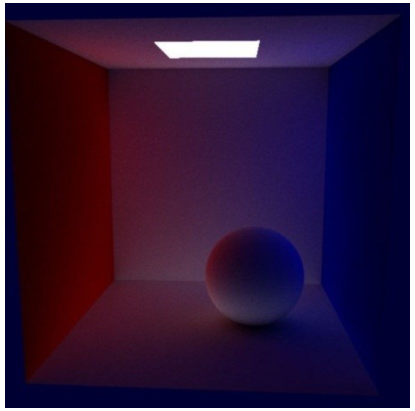
Global Illumination

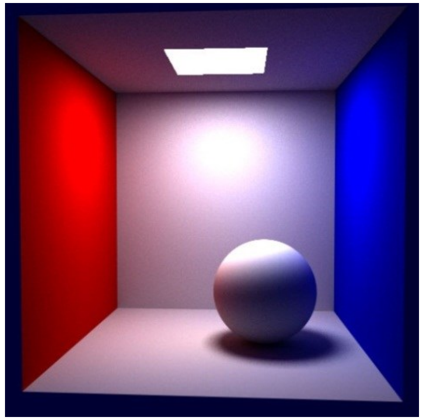




Direct illumination + Indirect illumination = Total illumination



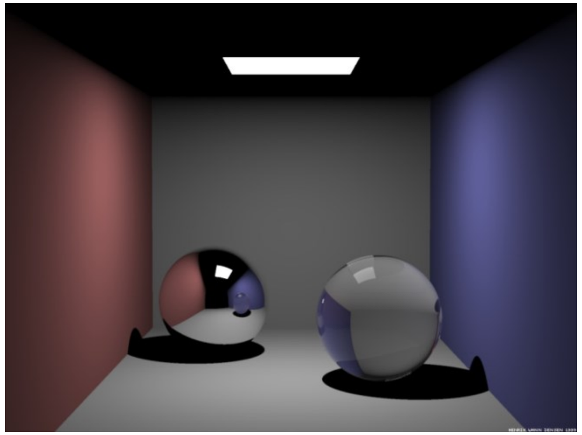




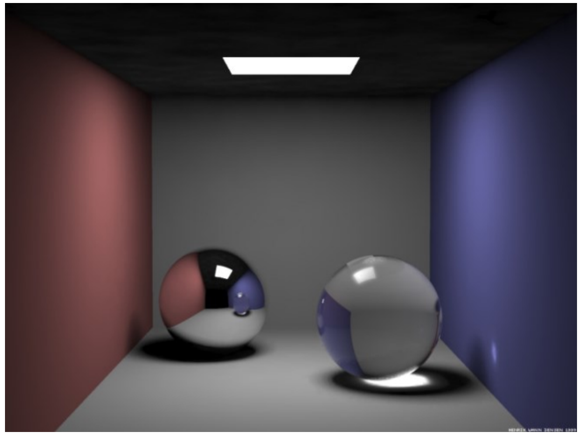
[Andy Van Dam]

Examples of Global Models

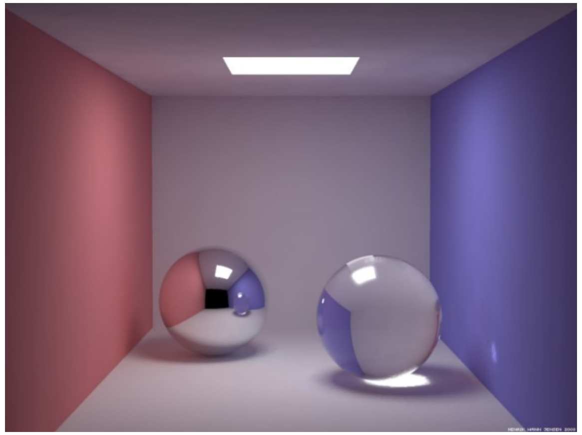
- Take into account **global information** of both **direct** (from emitters) and **indirect illumination** (inter-object reflections)
- Different approximations
  - Advantages and disadvantages; resource requirements
  - More computation gives better results…



Direct illumination + specular reflection Ray trace



+ soft shadows and caustics
Ray trace + caustic photon map



+ diffuse reflection (color bleeding)
Ray trace + caustic and diffuse photon maps

http://graphics.ucsd.edu/~henrik/images/global.html

Computing Illumination: *Light transport simulation*

- Evaluate illumination with enough **samples** to produce final images **without any guessing / shading**
- Often used for high quality renderers, e.g., **FX movies**
  - Can take **days for a single frame**
- **Some** implementations can run in real time on the **GPU**
- Many simulations use **stochastic sampling**

Computing Illumination
Polygon rendering – Shading

- Evaluate illumination at **several samples**
- **Shade** in-between to produce **pixels** in the final image
- **Often used** in real-time, e.g., computer games
- **Lower quality** than light transport simulation !!
    - But **satisfactory** results with various **additions** such as **maps** (bump, displacement, environment)

Computing Illumination
Polygon rendering – Realistic Images

- **perspective projections** of the scene models
- **Natural illumination effects** are obtained using:
    - **illumination model** – compute the **color** to be assigned to each visible **surface point**
    - **surface-rendering method** that applies an illumination model and assigns a **color to every pixel**

Computing Illumination
Illumination Models

- Often an **approximation to the Laws of Physics**
- Describe the **interaction between light & surface**
- **Different types** of illumination models:
  - **simple** models, based on simple **photometric** computations (to reduce the computational cost)
  - more **sophisticated** models, based on the propagation of **radiant energy** (computationally more complex)

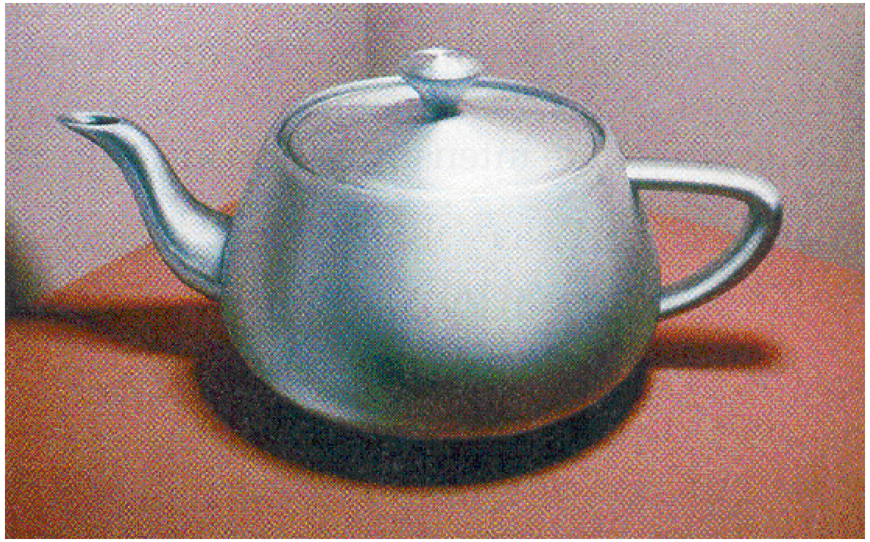
Computing Illumination
Polygon rendering – Materials

- **Photorealistic images** require:

  - precise representation **surface properties**

  - description of the **scene's illumination**

- And might imply modeling:
  - surface **texture**
  - transparency
  - **Reflections,** shadows
  - etc.



Computing Illumination
Shading Pipeline

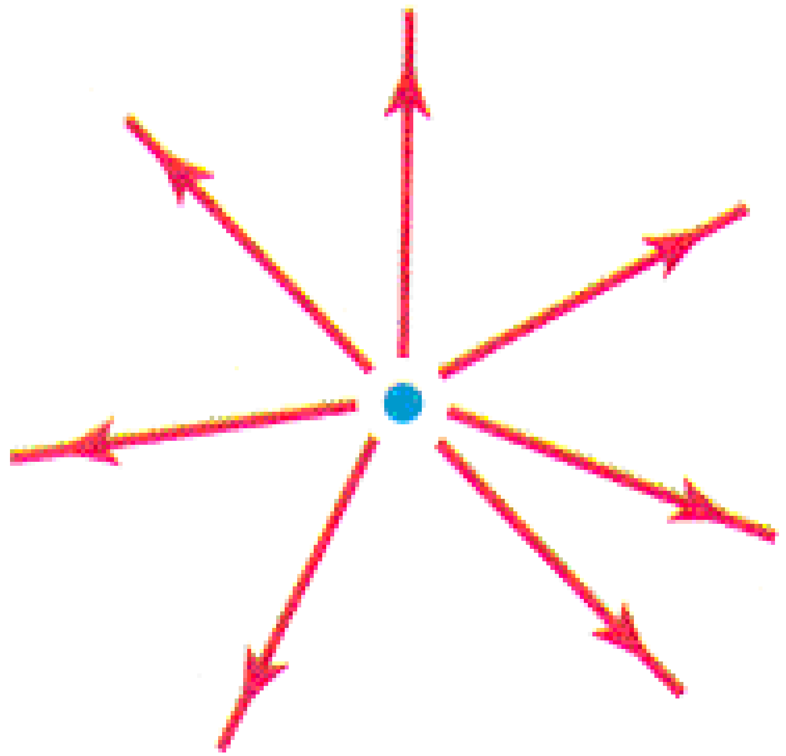
- Compute **surface color** based on
  - Type and number of **light sources**
  - Reflective **surface properties**
  - **Illumination model**
    - **Phong**: ambient + diffuse + specular components
  - Atmospheric effects
    - Fog, smoke
- **Polygons** making up a model surface **are shaded**
  - Realistic representation

Light sources

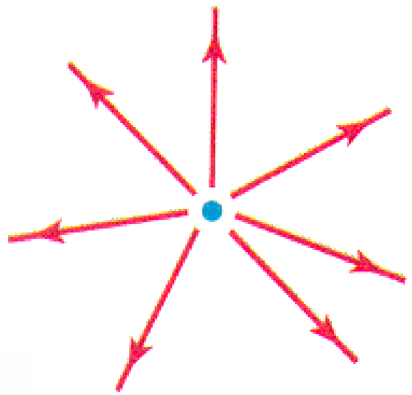
Light Sources

- Objects radiating light and conttributing to the illumination of a scene's objects

- Can be defined by several features:
  - Position,
  - Color of emitted light
  - Emission direction
  - Shape

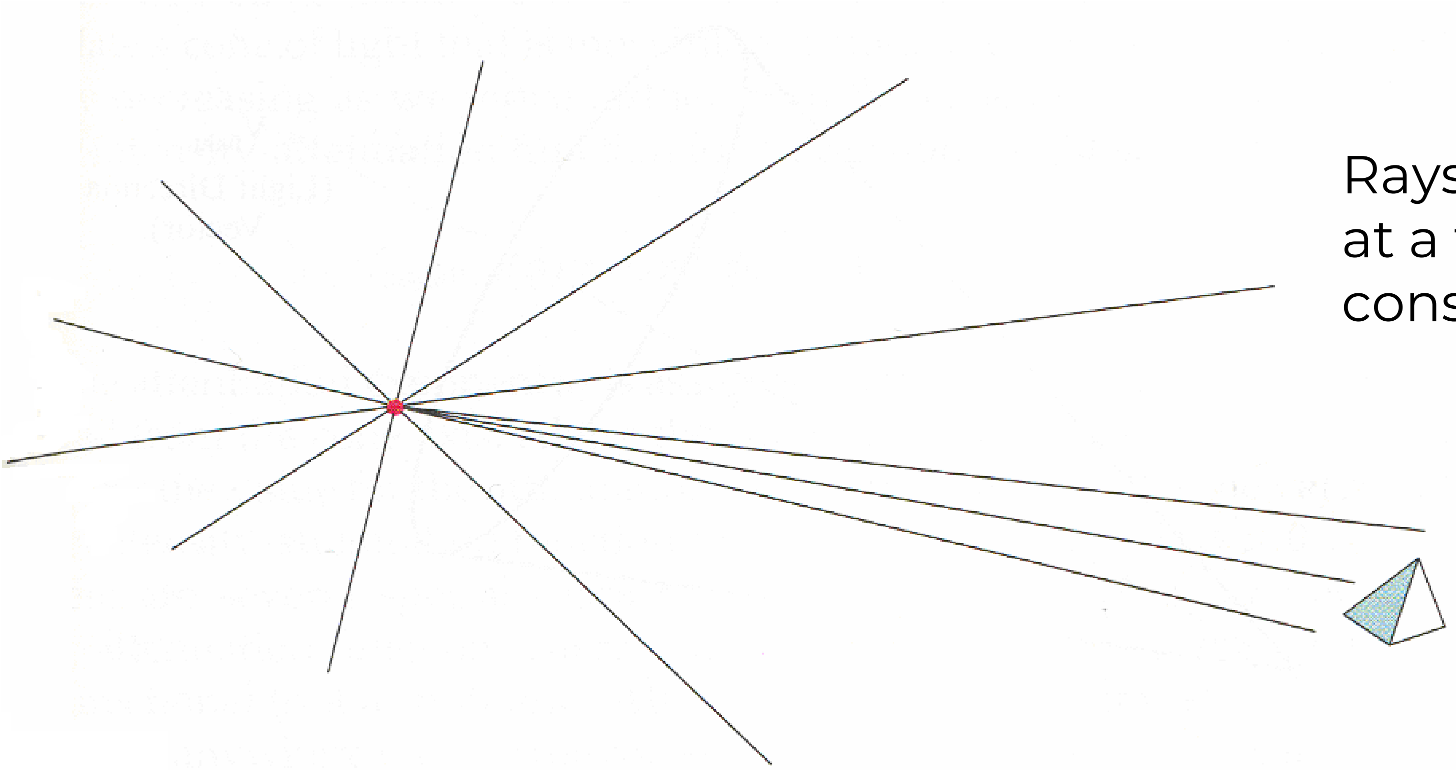


Isotropic point light source



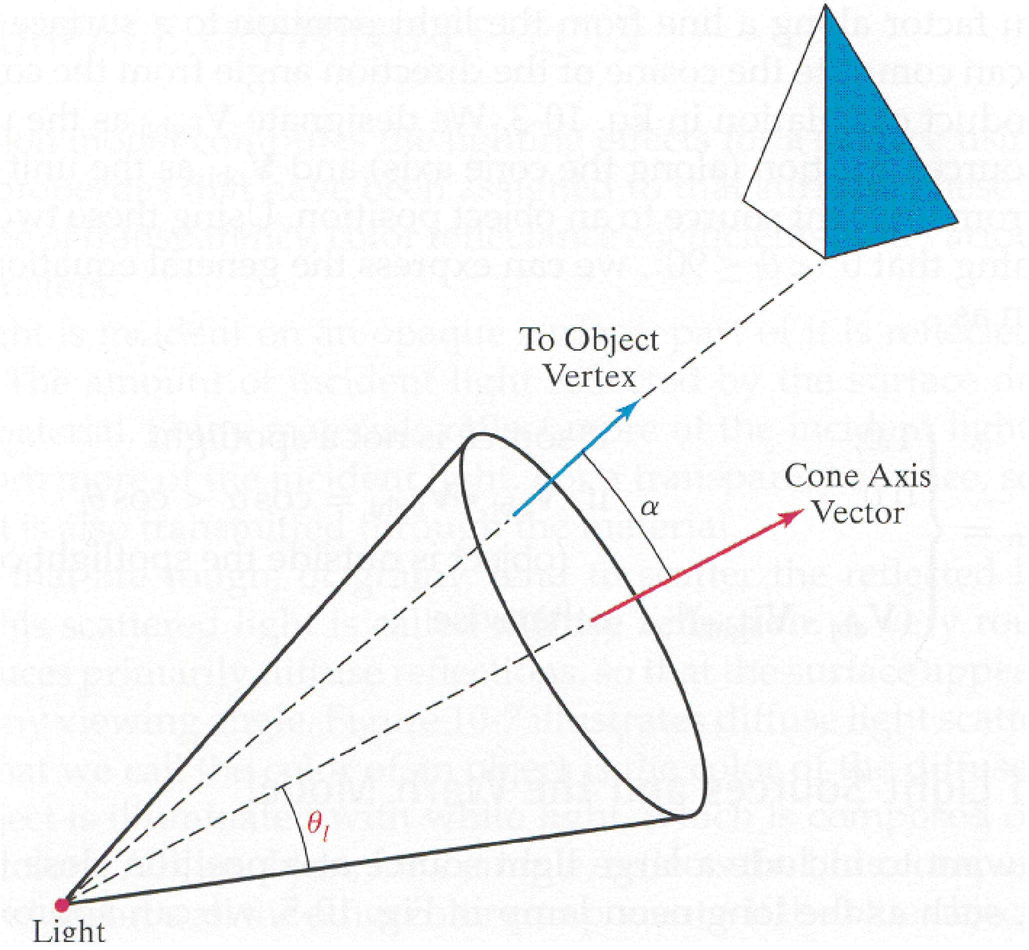
Simplified Light Sources

Light source at an indefinite distance



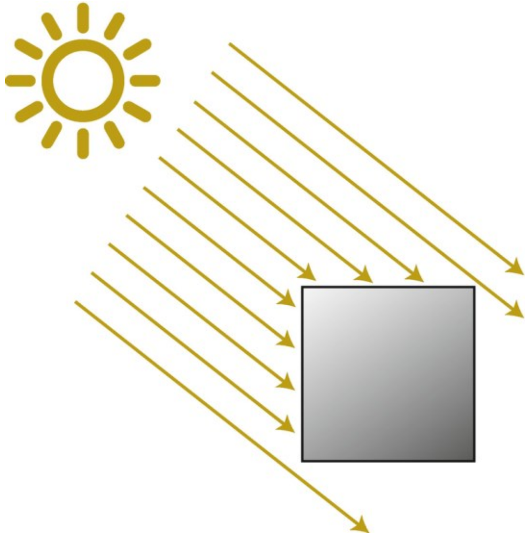
Rays emitted by a light source at a far-away location can be considered as **parallel**

Spotlight – Directional Light Source

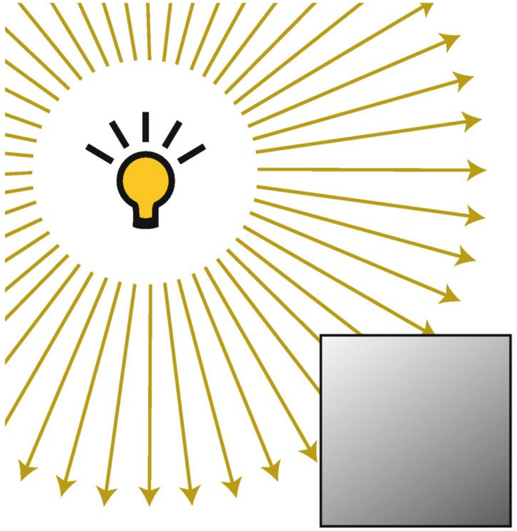


A directional light source is defined by a direction and an emission angle

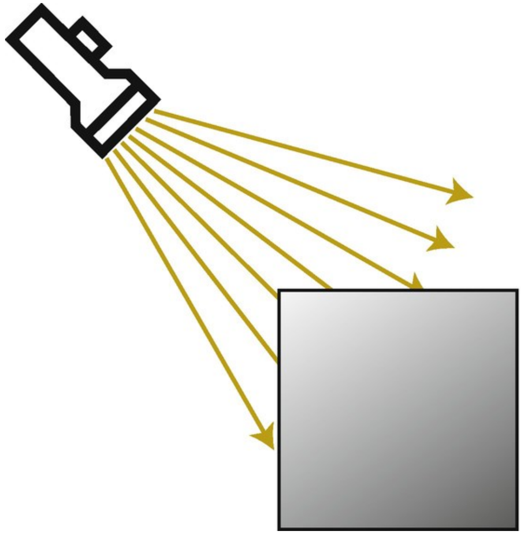
Directional light



Point light



Spotlight



Halladay’s Practical Shader Dev.

Surface features

https://jacobsschool.ucsd.edu/news/release?id=1987

Surface Features

An illumination model takes into account a **surface's optical properties**:

reflection coefficients for each color
degree of transparency
texture parameters

When light is incident on an opaque surface: part is absorbed, part is reflected

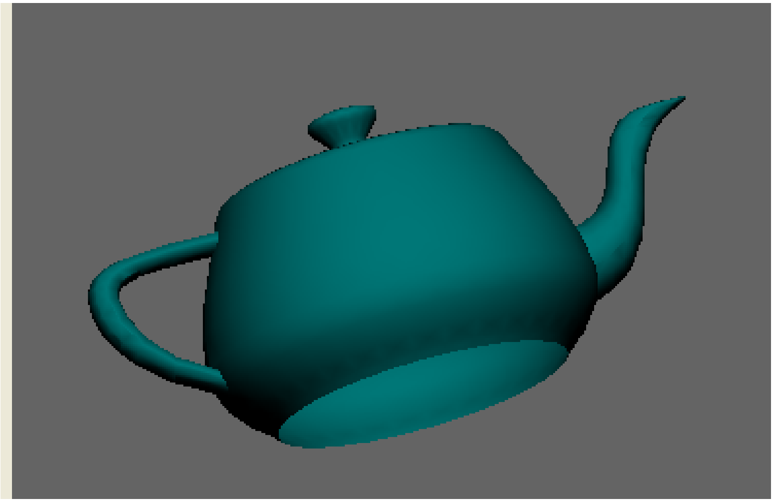
Surface Features

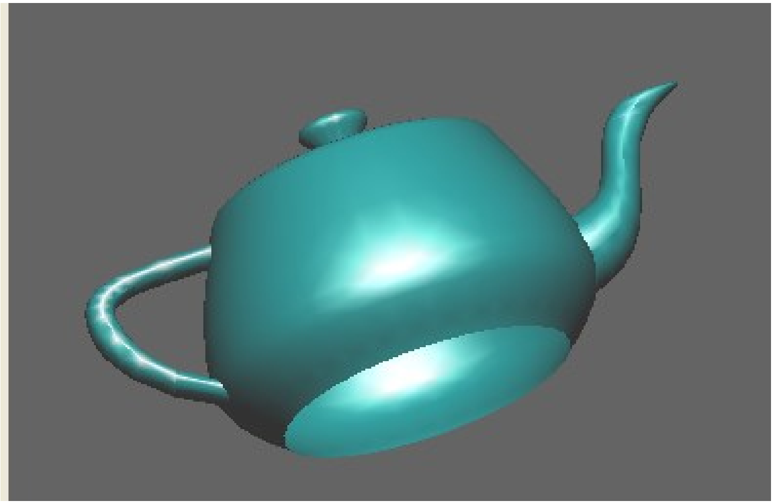
- The amount of reflected light depends on the **surface's features**

  Shiny surfaces reflect more light

  Mate / dull surfaces reflect less light

- **Transparent surfaces** transmit some light

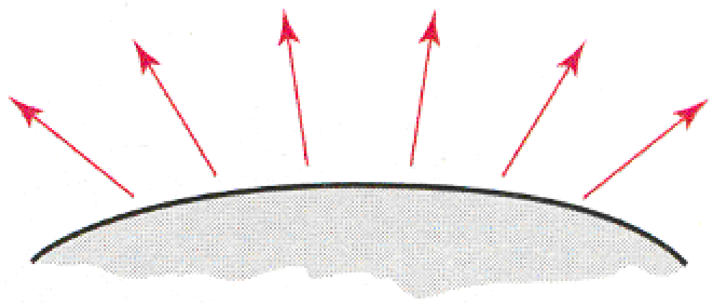




Surface Features

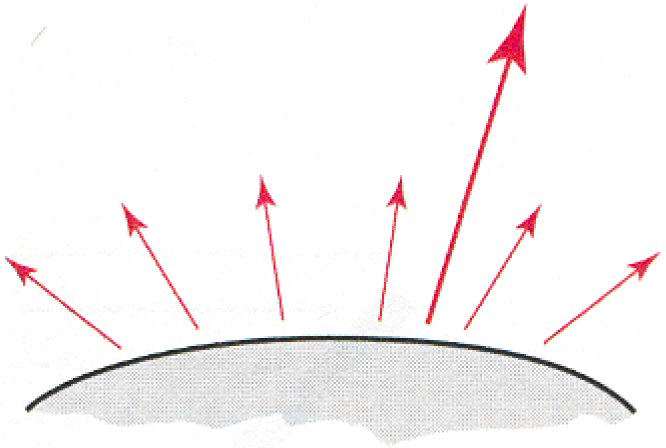
- **Rough surfaces** tend to spread the reflected light in all directions
    - **diffuse reflection**

And look equally shiny from any viewpoint



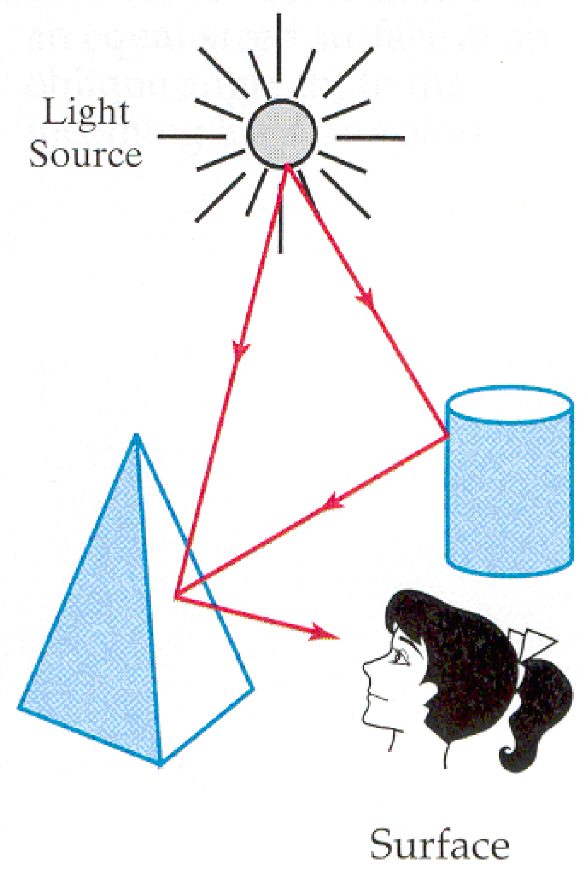
- **Smooth surfaces** reflect more light in particular directions
    - **specular reflection (*highlight*)**

And present some shinier areas



Surface Features

- A surface **might not be directly illuminated** and still **be visible**, due to light reflected by other objects in the scene
- **Ambient illumination**
- The amount of light reflected by a surface is the **sum of all contributions** from the light sources and the ambient illumination



Phong’s reflection model

Basic illumination models

- **Sophisticated illumination** models **precisely compute** the interaction effects between the radiating energy and the surface material

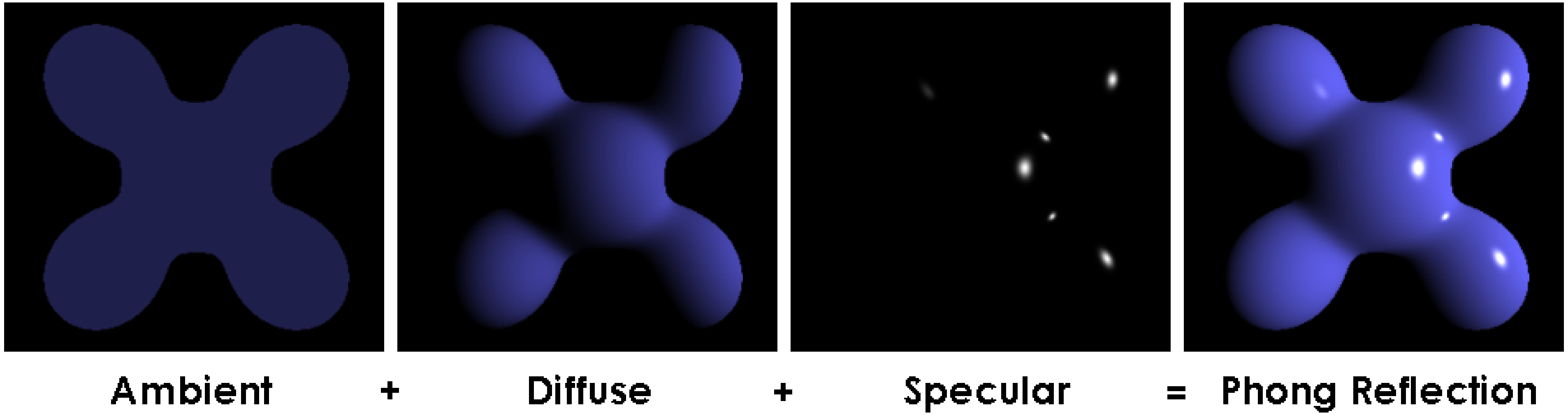
- Basic models use **approximations** to represent the physical illumination effects

- The **empirical model** by **Phong** computes good results for most situations and includes:

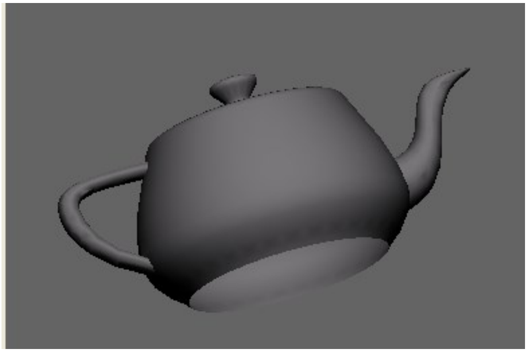
  - **ambient illumination, diffuse reflection, specular reflection**

Phong reflection model – 1973

Reflection is obtained considering three components:

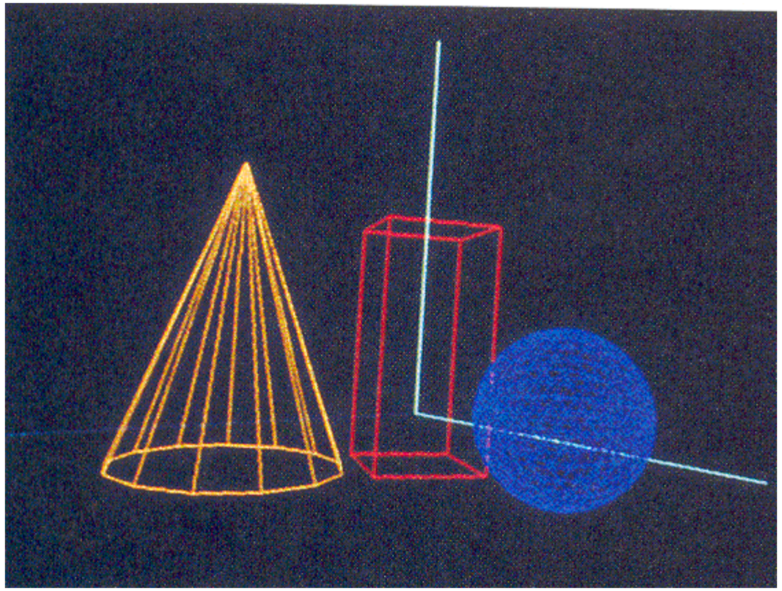


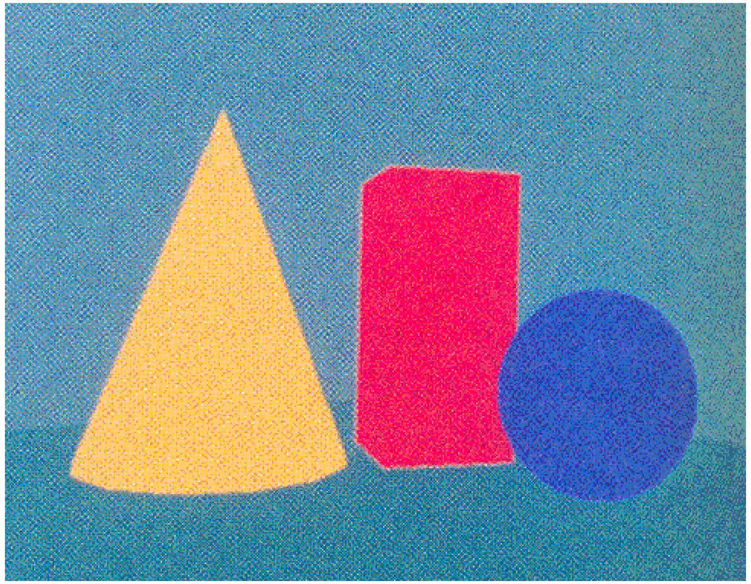
Phong Model – Ambient illumination



- Constant illumination component for each model
- Independent from viewer position or object orientation!
- Take only material properties into account !

Phong Model – Ambient illumination

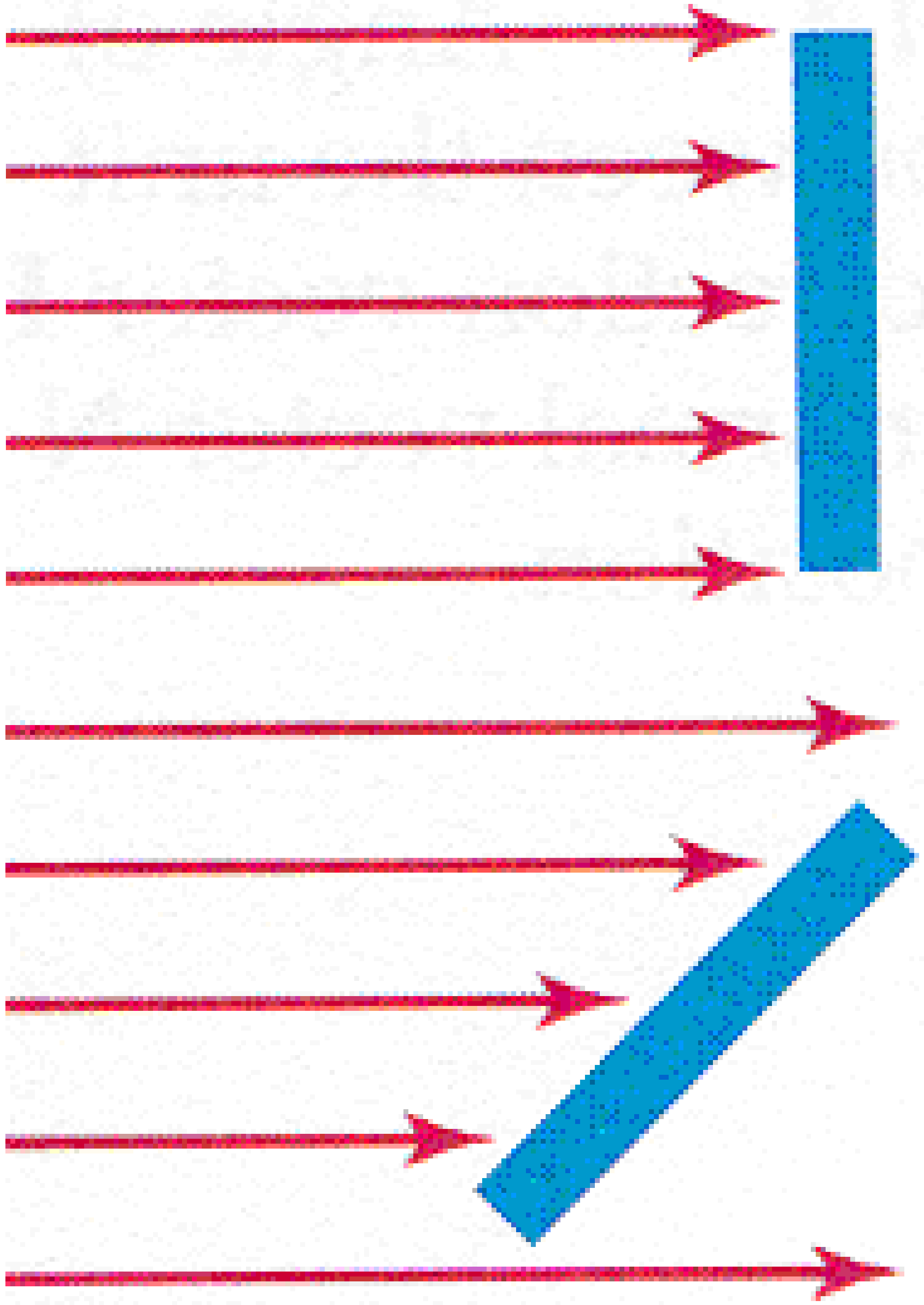




Phong Model – Diffuse Reflection

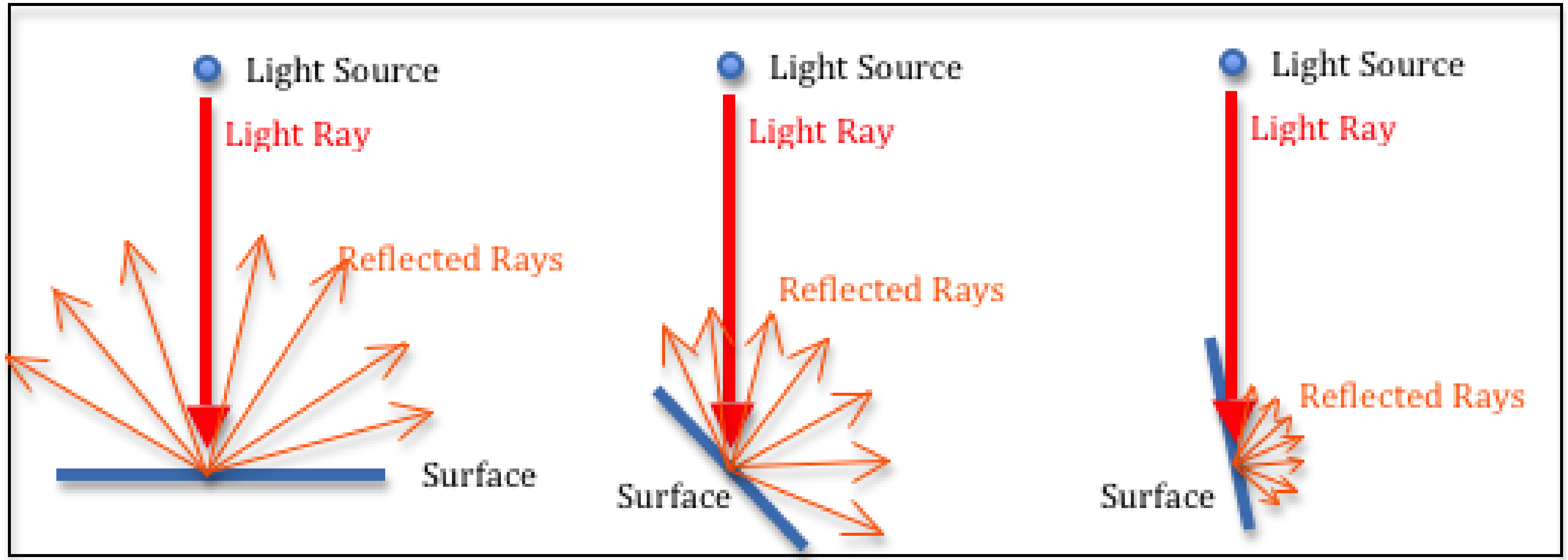
There is, at least, one point light source (usually located at the viewpoint)

The amount of incident light depends on the surface orientation regarding the direction of the light source



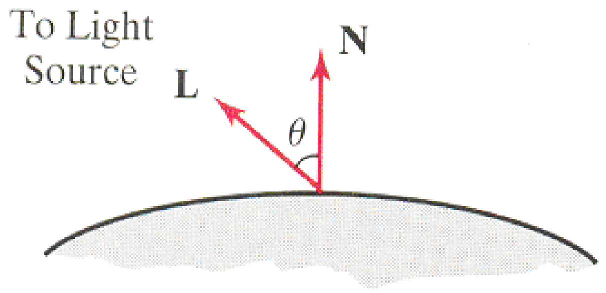
Phong Model – Diffuse Reflection

Reflected light depends on surface orientation



Phong Model – Diffuse Reflection

θ is the **incidence angle** (between the surface normal and the light direction)



Given a light source I_l, the **amount of diffusely reflected light** by a surface is:

$$I_{l,\text{diff}} = k_d I_l \cos\theta$$

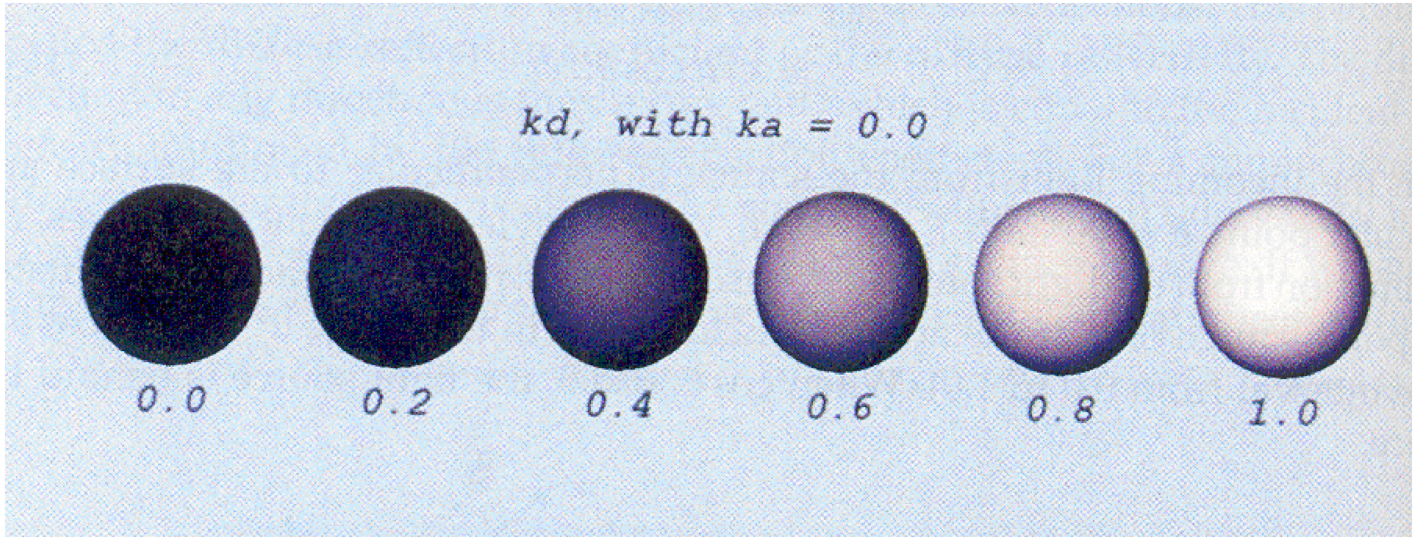
$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$$

Using **unit vectors**:

N → surface normal
L → light source direction

$$I_{l,\text{diff}} = \begin{cases} k_d I_l (\mathbf{N} \cdot \mathbf{L}), & \text{se} \quad \mathbf{N} \cdot \mathbf{L} > 0 \\ 0.0, & \text{se} \quad \mathbf{N} \cdot \mathbf{L} \le 0 \end{cases}$$

Phong Model – Diffuse reflection



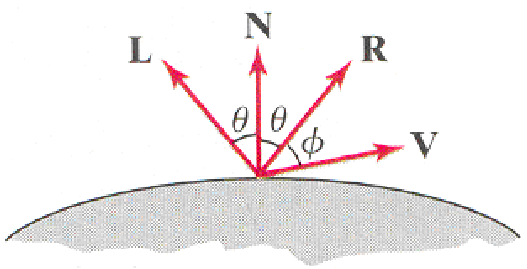
Diffuse reflection for a sphere illuminated by a white point light source, with 0 < Kd < 1, and without ambient illumination (Ka = 0)

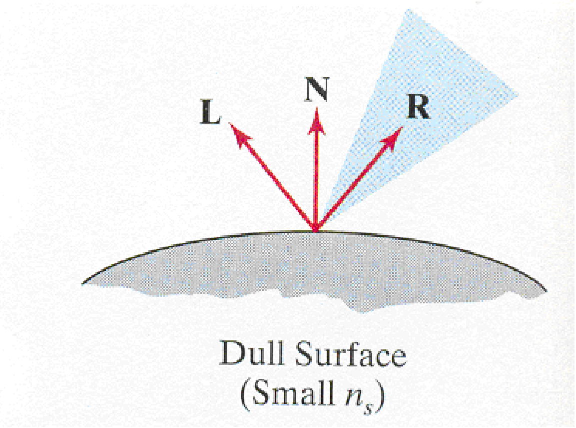
Adding the ambient and diffuse components:

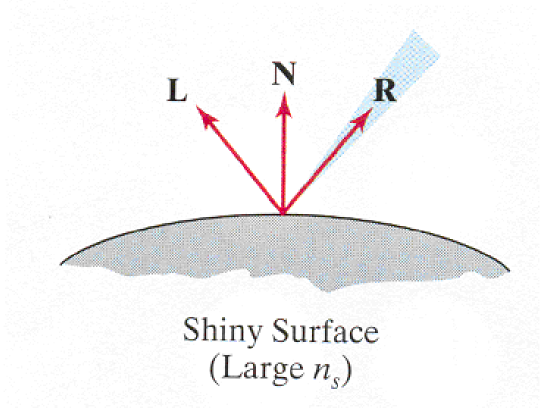
$$I_{\text{diff}} = \begin{cases} k_a I_a + k_d I_l\,(\mathbf{N} \cdot \mathbf{L}), & \text{if} \quad \mathbf{N} \cdot \mathbf{L} > 0 \\ k_a I_a, & \text{if} \quad \mathbf{N} \cdot \mathbf{L} \leq 0 \end{cases}$$

Phong Model – Specular reflection

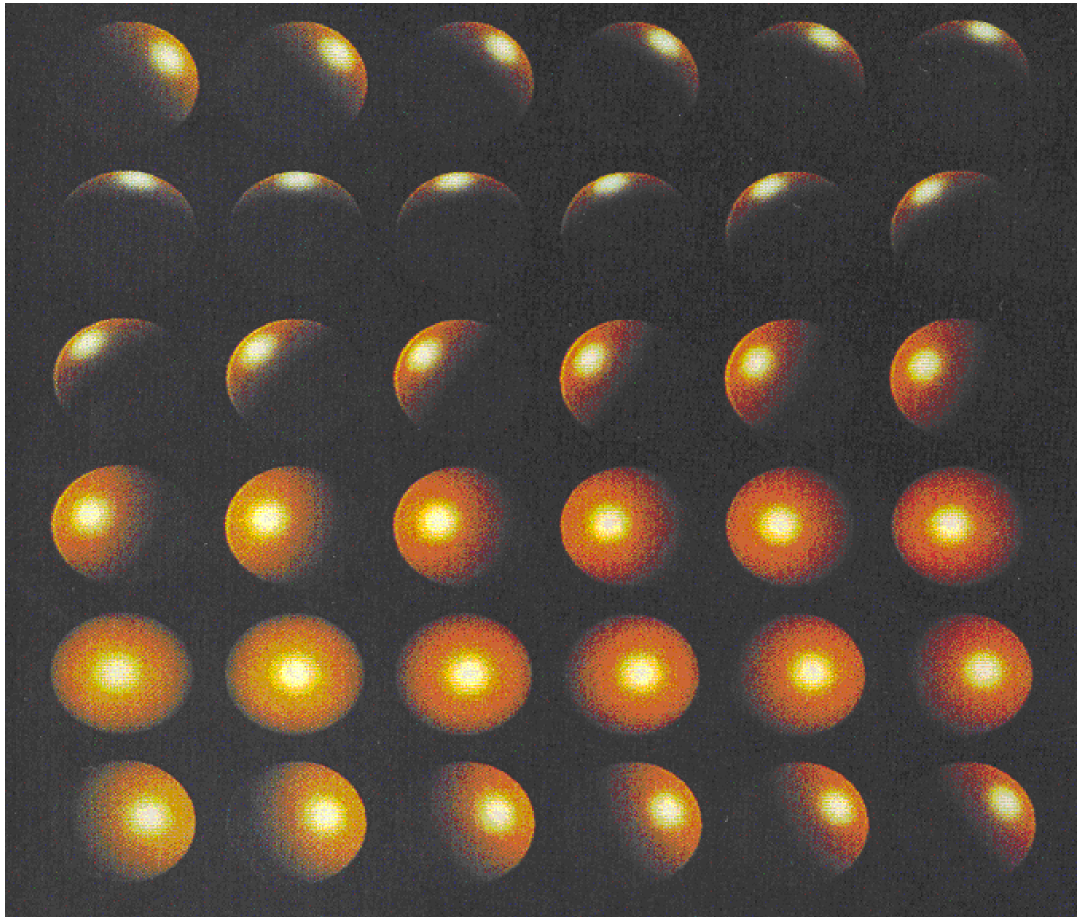
- Important for shiny model surfaces
  - How to model **shininess** ?
- Take into account **viewer position**



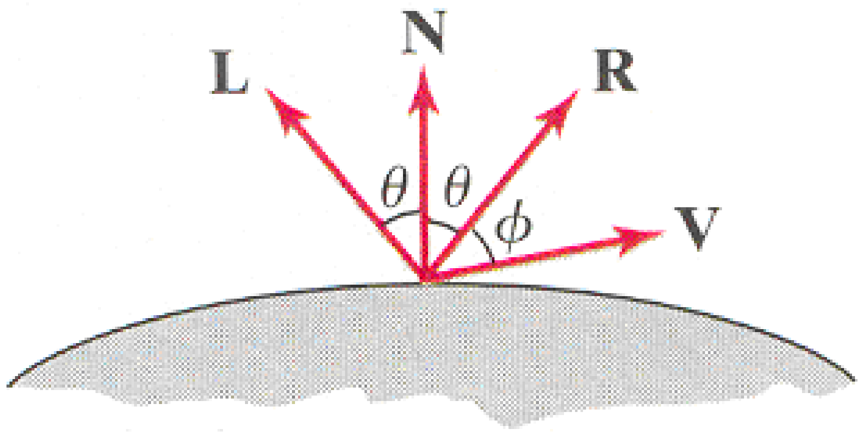




Phong Model – Specular reflection



Phong Model – Specular reflection



$$I_{l,\text{spec}} = \begin{cases} k_s I_l (\mathbf{V} \cdot \mathbf{R})^{n_s}, & \text{if } \mathbf{V} \cdot \mathbf{R} > 0 \quad \text{and} \quad \mathbf{N} \cdot \mathbf{L} > 0 \\ 0.0, & \text{if } \mathbf{V} \cdot \mathbf{R} < 0 \quad \text{or} \quad \mathbf{N} \cdot \mathbf{L} \le 0 \end{cases}$$

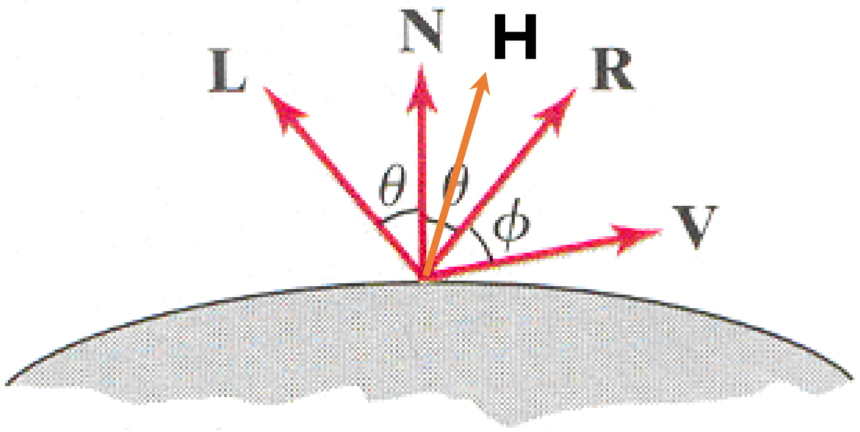
Considering unit vectors!!

Blinn-Phong Model – Specular reflection

Halfway vector

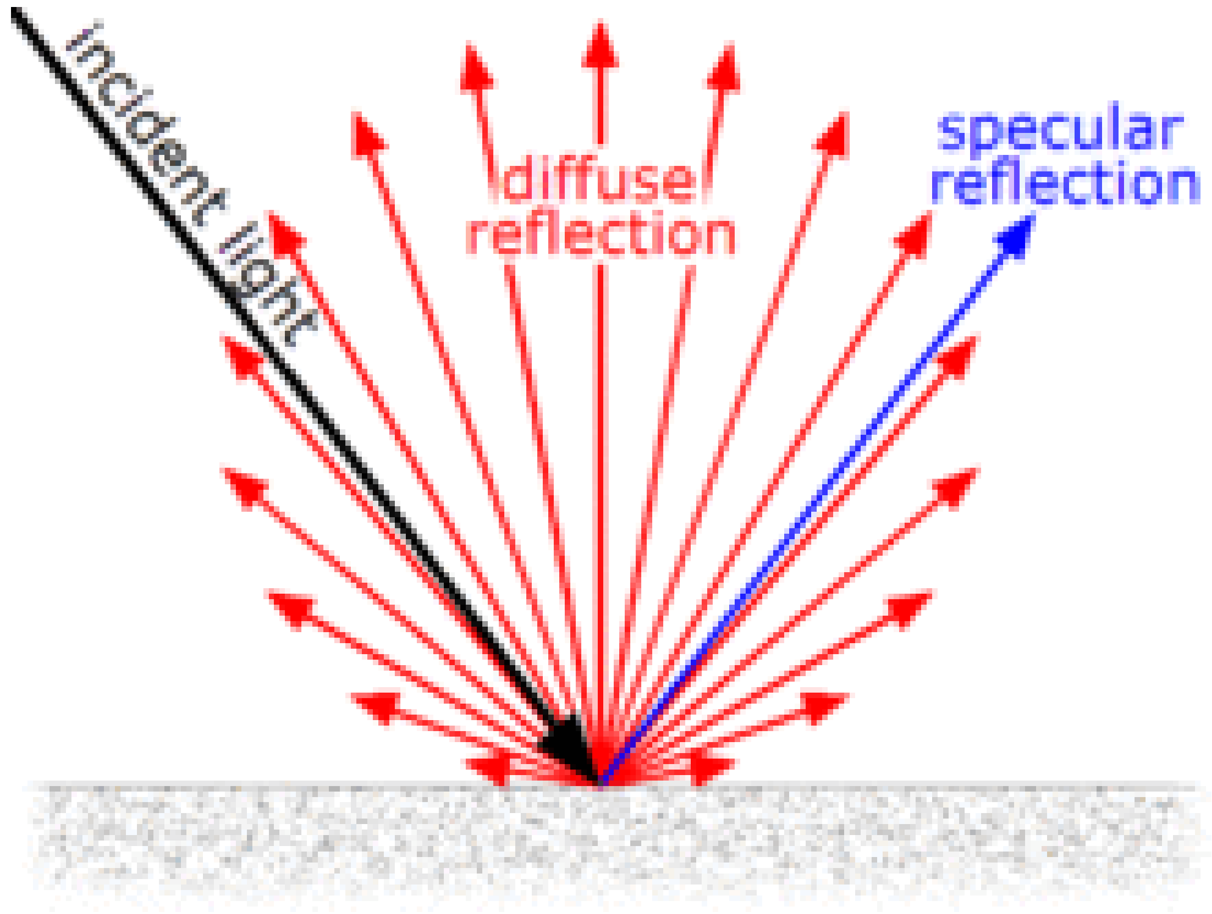
Computing **R** involves some computational resources that can be reduced by computing the **halfway vector**

H = (L + V) / || L + V ||



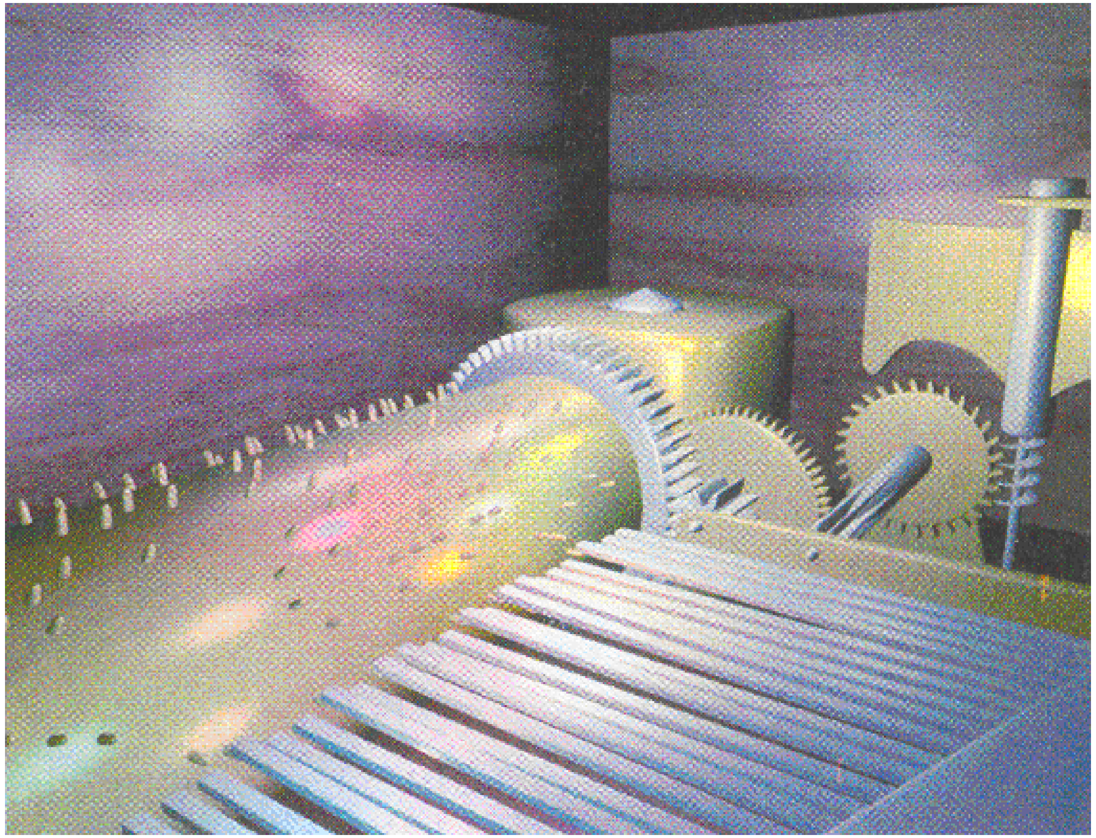
$$I_{l,\text{spec}} = \begin{cases} k_s I_l (\mathbf{H} \cdot \mathbf{N})^{n_s} \\ 0.0, \end{cases}$$

Considering unit vectors!!


incident light
diffuse reflection
specular reflection

More than one light source

- Add the diffuse and specular components



$$I = k_a I_a + \sum_{l=1}^{n} I_l [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{N} \cdot \mathbf{H})^{n_s}]$$

Material properties

Material properties

There are tables of the parameters that can be used to approximate the visual properties for several types of materials

Material	ambient (k_a)	diffuse (k_d)	specular (k_s)	specular exponent (m)	translucency (σ)
Brass	0.329412 0.223529 0.027451	0.780392 0.568627 0.113725	0.992157 0.941176 0.807843	27.8974	1.0
Bronze	0.2125 0.1275 0.054	0.714 0.4284 0.18144	0.393548 0.271906 0.166721	25.6	1.0
Polished Bronze	0.25 0.148 0.06475	0.4 0.2368 0.1036	0.774597 0.458561 0.200621	76.8	1.0
Chrome	0.25 0.25 0.25	0.4 0.4 0.4	0.774597 0.774597 0.774597	76.8	1.0
Copper	0.19125 0.0735 0.0225	0.7038 0.27048 0.0828	0.256777 0.137622 0.086014	12.8	1.0
Polished Copper	0.2295 0.08825 0.0275	0.5508 0.2118 0.066	0.580594 0.223257 0.0695701	51.2	1.0
Gold	0.24725 0.1995 0.0745	0.75164 0.60648 0.22648	0.628281 0.555802 0.366065	51.2	1.0
Polished Gold	0.24725 0.2245 0.0645	0.34615 0.3143 0.0903	0.797357 0.723991 0.208006	83.2	1.0

$$I = k_a I_a + \sum_{l=1}^{n} I_l [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{N} \cdot \mathbf{H})^{n_s}]$$

[https://people.eecs.ku.edu/~jrmiller/Courses/672/InClass/3DLighting/MaterialProperties.html]

THIS IS the Phong Reflection Model

$$I = k_a I_a + \sum_{l=1}^{n} I_l [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{N} \cdot \mathbf{H})^{n_s}]$$

i.e., how illumination is computed at a certain point based on light and material properties


shading

Illumination and Shading

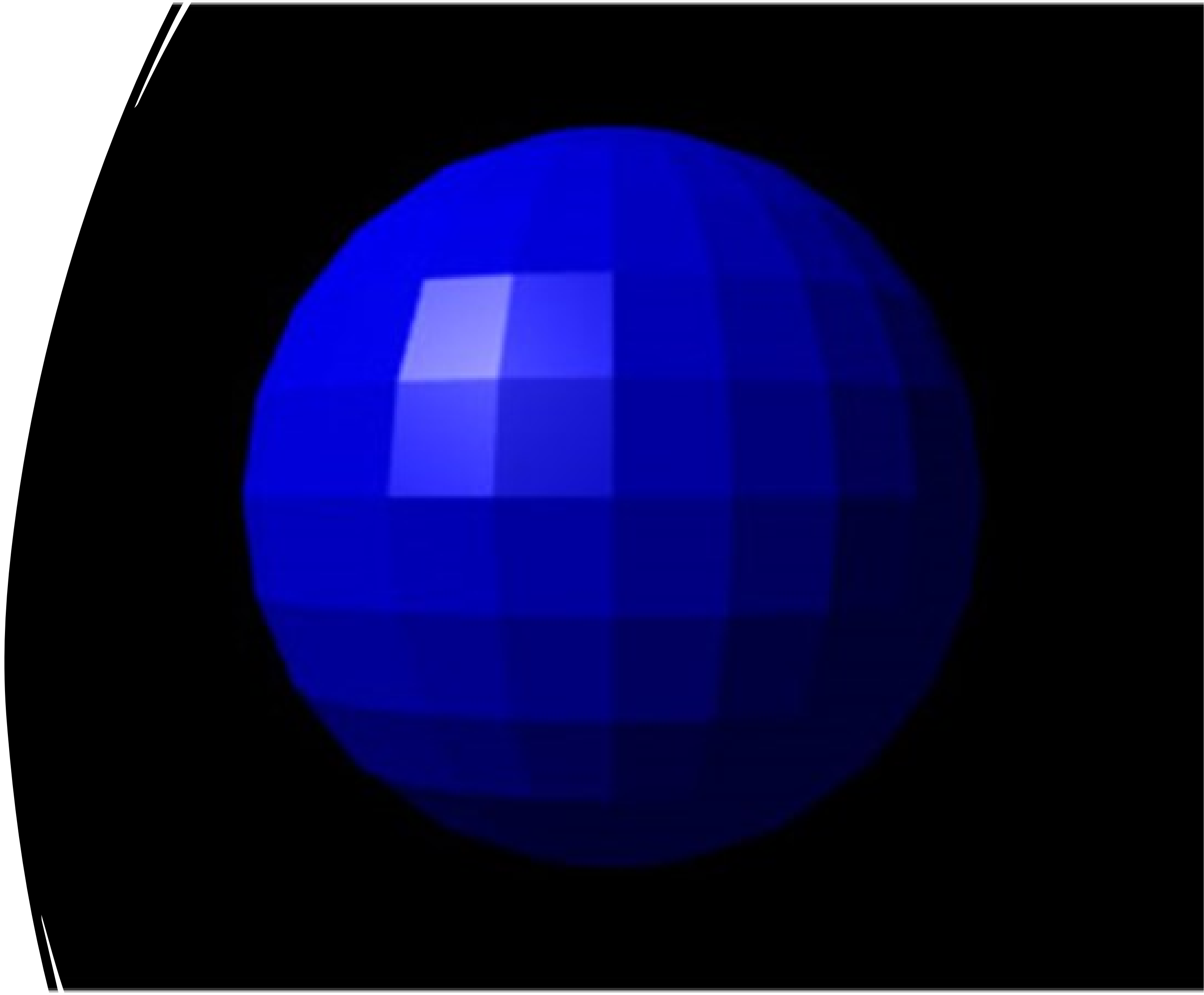
- How to optimize?
  - Fewer light sources
  - Simple **shading** method
- BUT, less computations mean less realism
  - Wireframe representation
  - **Flat** shading
  - **Gouraud** shading
  - **Phong** shading

Flat-Shading

- For each triangle / polygon
  - Apply the illumination model **just once** !
  - All **pixels** have the **same color**

- Fast

- But objects seem “**blocky**”



FLAT SHADING

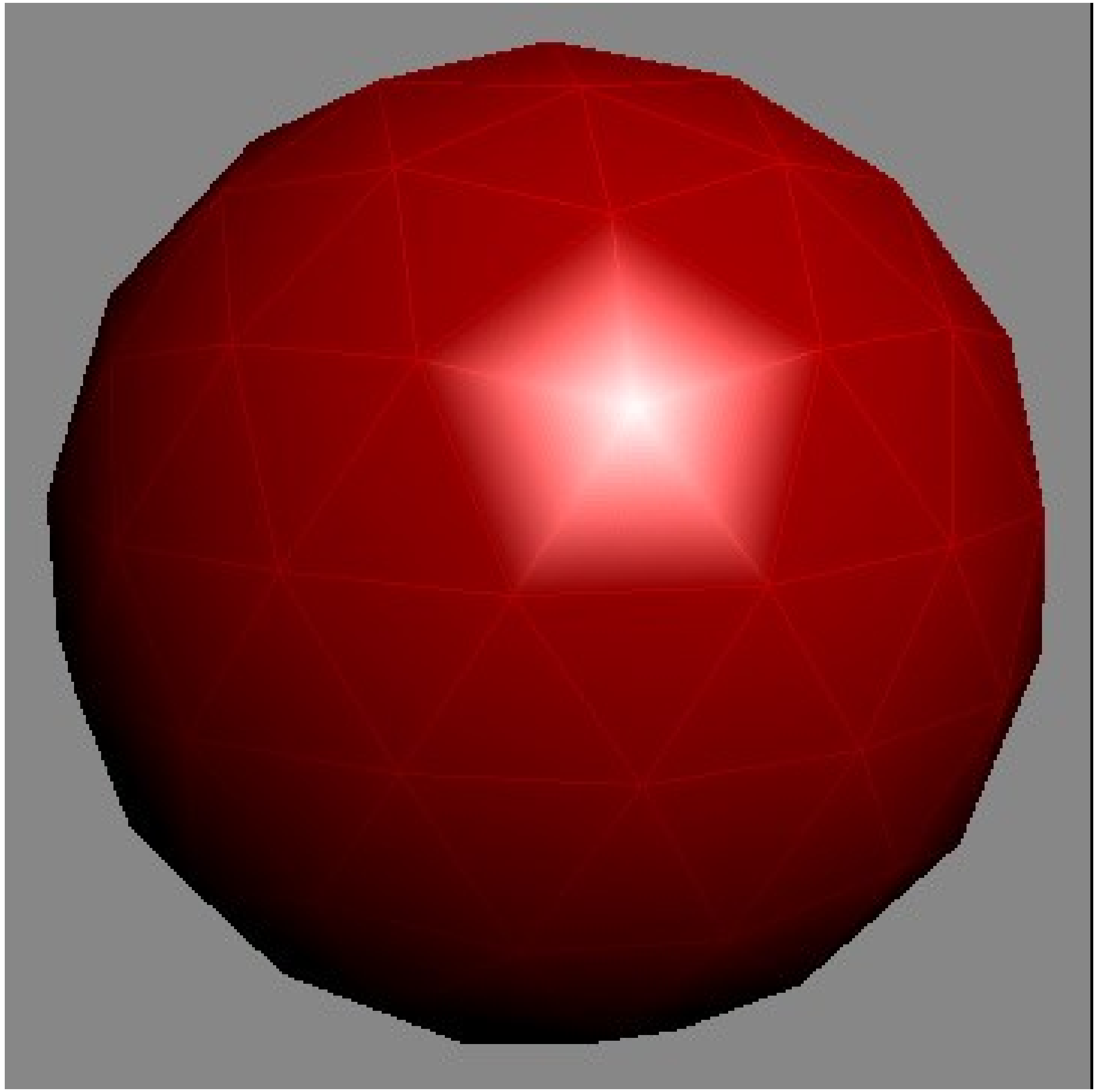
Gouraud Shading – 1971

- For each triangle / polygon
  - Apply the illumination model at **each vertex**
  - **Interpolate** color to shade each pixel

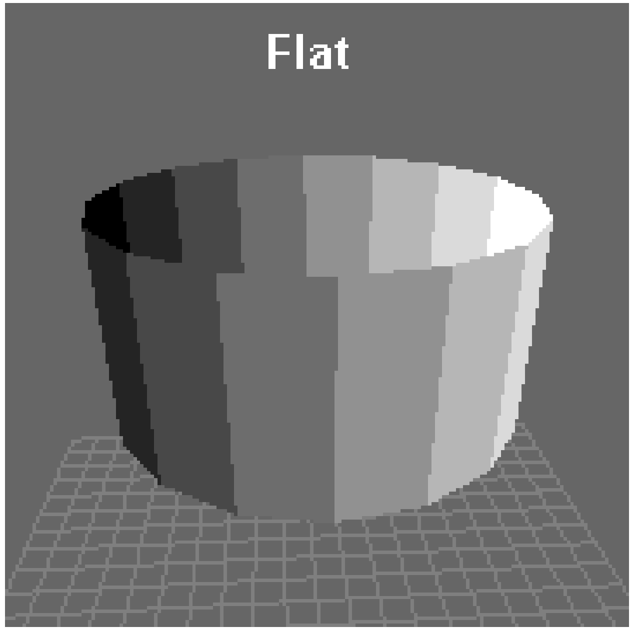
- Better than flat-shading

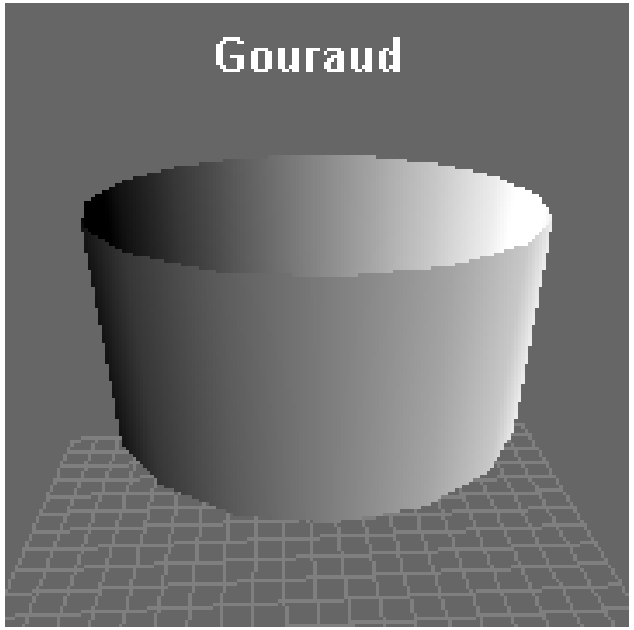
- Problems with highlights

- Mach-banding effect



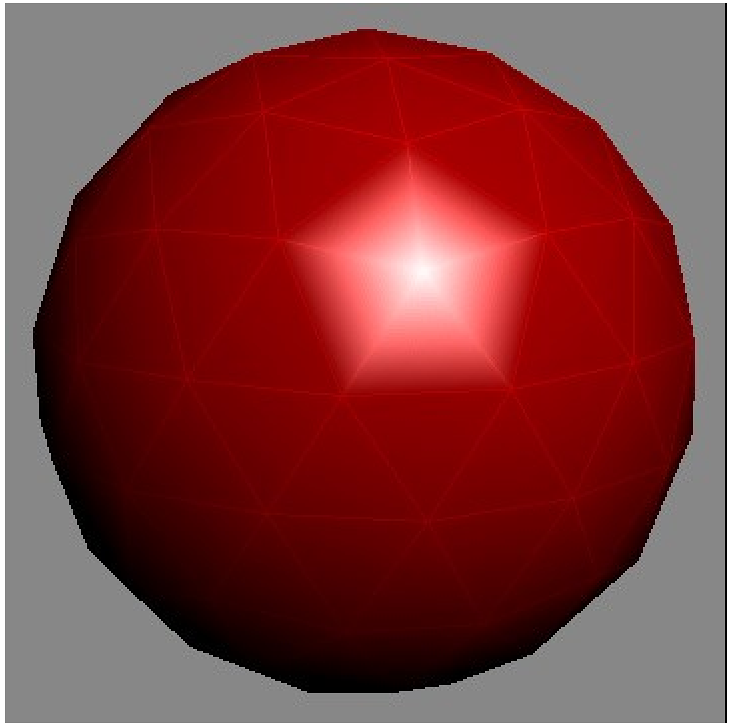
Flat-Shading vs Gouraud Shading

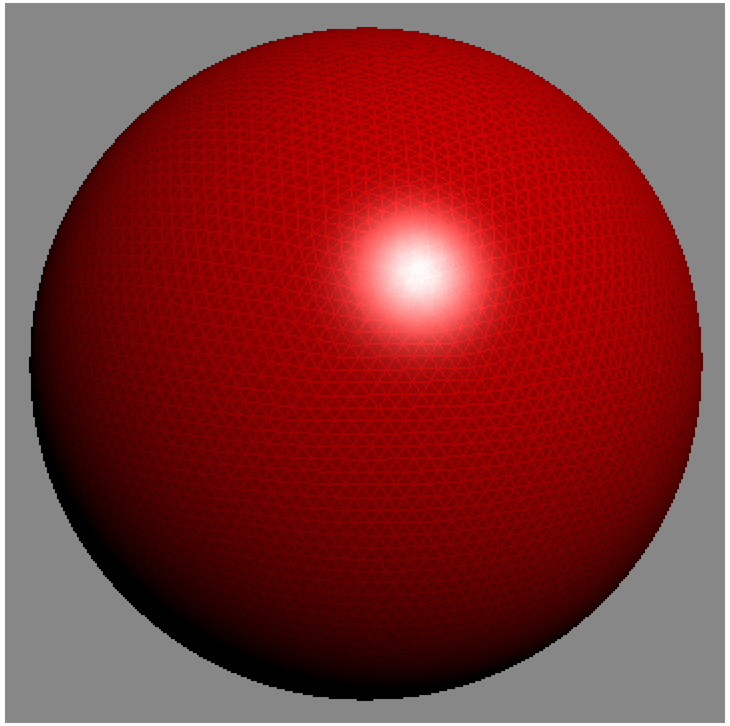




[Wikipedia]

Gouraud Shading – More triangles !





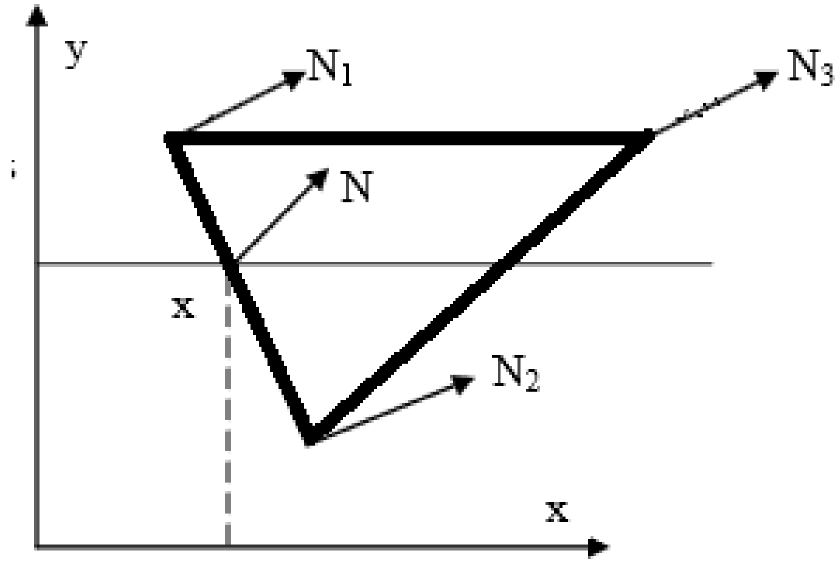
[Wikipedia]

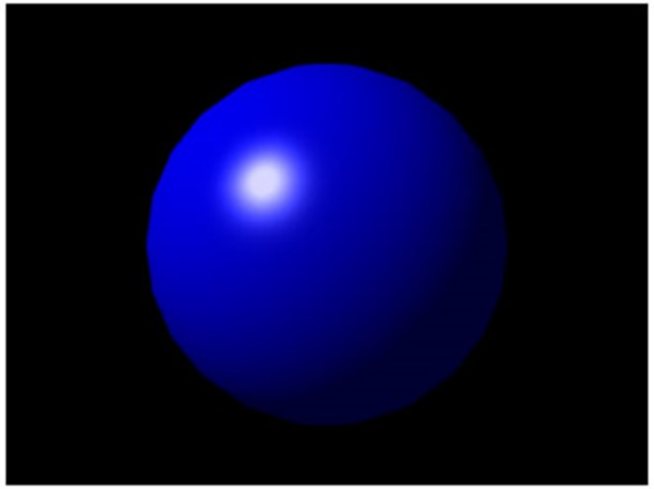
Phong Shading – 1973

- For each triangle / polygon
  - **Interpolate normal vectors** across rasterized polygons

- Better than Gouraud shading

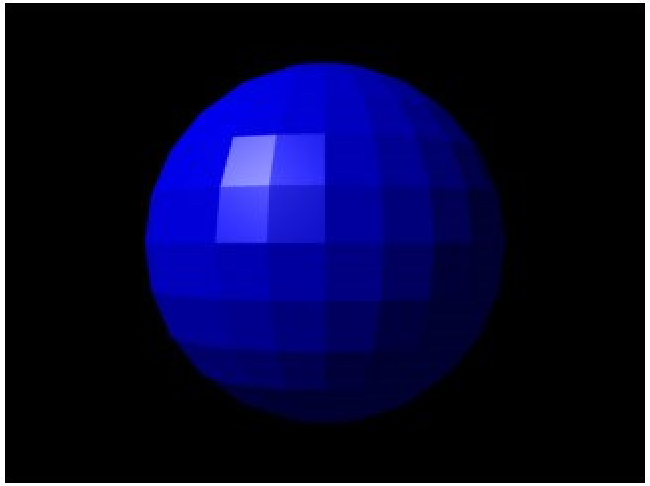
- BUT, more **time consuming**



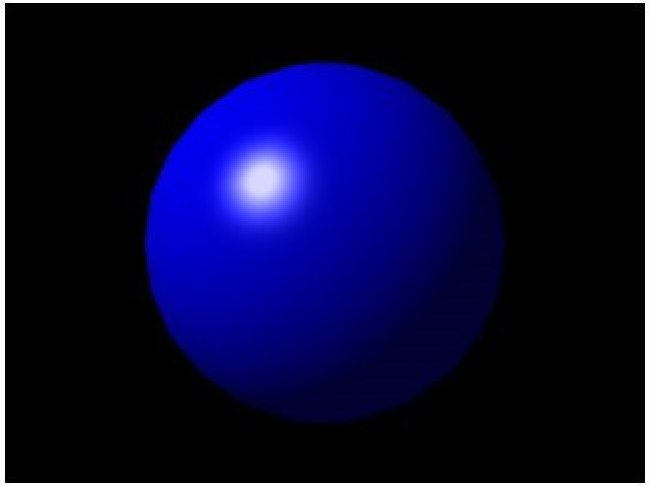


PHONG SHADING

Flat-Shading vs Phong Shading



FLAT SHADING



PHONG SHADING

What happens in the shaders?

Basic (no) Shading

vertex shader

- a **fixed color is passed** to the fragment shader
- **NO** illumination is computed

fragment shader

- **color is applied everywhere**

Flat Shading

vertex shader

Illumination computed here

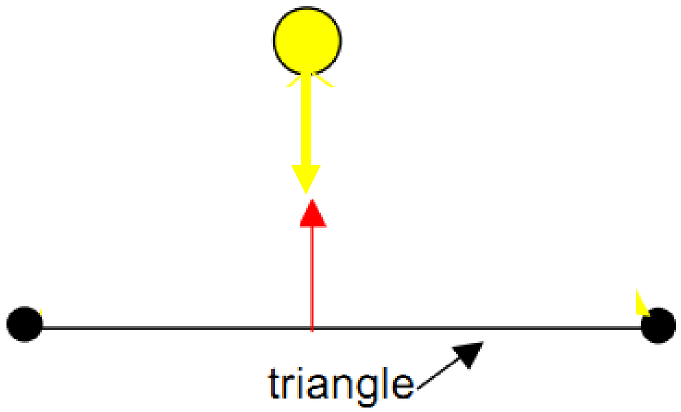
- a **color is computed per polygon** considering its normal, the light position, and the viewer position

- For this, the color information for the **provoking vertex** is used

- the **color is passed to the fragment shader** as representative of the polygon color

fragment shader

- the fragment shader **does not interpolate vertex colors**. It applies the fixed color to all fragments



Gouraud Shading

vertex shader

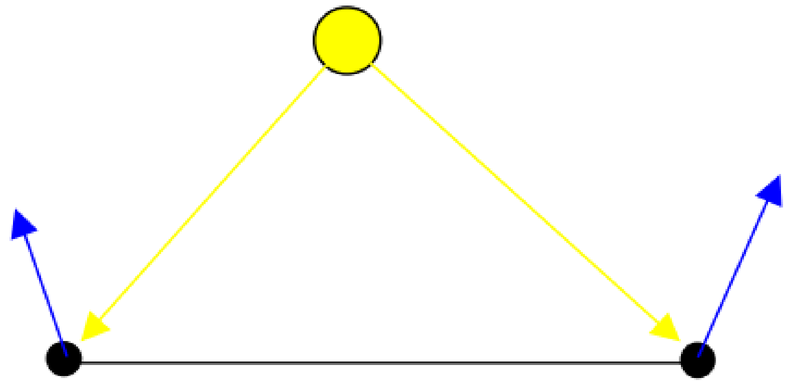
Illumination computed here

- a **color is computed per vertex** considering its normal, the light position, and the viewer position

- the **color is passed to the fragment shader** as representative of the vertex color

fragment shader

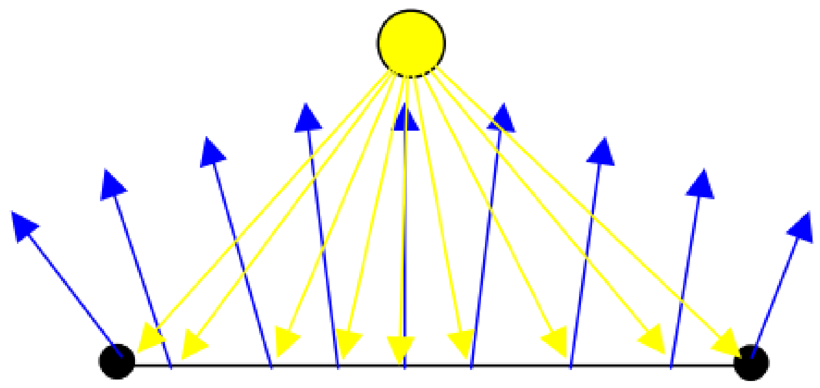
- **interpolates** the received **colors** in-between vertices for each position



Phong Shading

vertex shader

- **position** and **normal** of **each vertex** are passed to the fragment shader



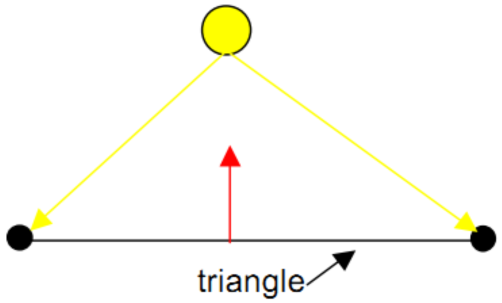
fragment shader

Illumination computed here

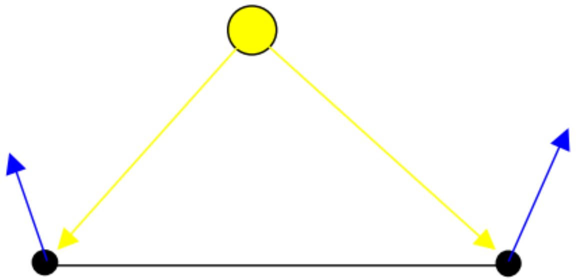
- interpolates the **normals** and vertex positions

- Computes the **color for each point from interpolated normals** and positions

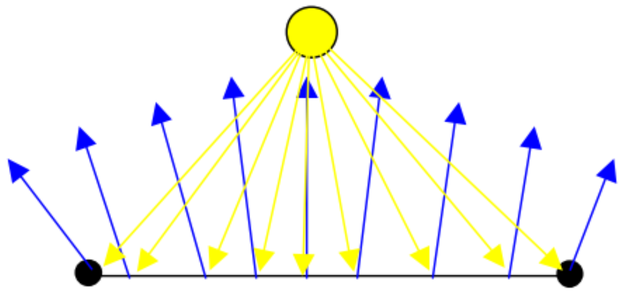
Shading Alternatives



Flat



Gouraud



Phong

https://opengl-notes.readthedocs.io/en/latest/topics/lighting/shading.html


Bibliography

- Steve Marshner, Peter Shirley, “Fundamentals of Computer Graphics”, 5th ed., chapter 5, advanced:…, 2021 https://learning.oreilly.com/library/view/fundamentals-of-computer/9781000426359/
- Kyle Halladay, “Practical Shader Development: Vertex and Fragment Shaders for Game Developers”, chapters 8 and 9, advanced: chapter 12, Apress, 2019 https://learning.oreilly.com/library/view/practical-shader-development/9781484244579/


Hands On!

In the hands-on you will be able to check the different types of shading....

You will not be asked to implement it from scratch, but **pay close attention to the code inside the shaders** and how it relates to the phong reflection model