3D Graphics

The rendering pipeline
Fragment Shader
Vertices & Camera → Indices → Light sources

Fragment Shader

Vertex Shader → Rasterizer → Fragment Shader → Depth Buffer → Image

Compute the color to give each fragments
Fragment Shader

Take a fragment as an input, and output a new fragment with color information.

Simulate how material interact with light.
We want to measure the light arriving at the camera

1. Emit the light
2. Reflect part of the light
3. Measure the light

Normal
Light Behavior

Reflected Light Ray

Emitted Light Ray
Specular material

Parallel emitted light ray stay parallel once reflected
Specular material

Smooth surfaces
Parallel emitted light ray stay parallel once reflected

The camera will receive a lot of light if it is Aligned with the reflected rays
Specular material

Parallel emitted light ray stay parallel once reflected

The camera will receive little light if it is not Aligned with the reflected rays
Example

Specular
Question

What are real life specular surfaces?
(1 minute alone)
(2 minutes with your neighbors)
(5 minutes with the whole group)
Example of specular surfaces

Polished metal
Example of specular surfaces

Polished surfaces
Example of specular surfaces

waxed parquet
Diffuse material

The surface is rough at microscopic level, reflected rays are random.

The camera receive the same amount of light regardless of its orientation.
Diffuse material

The surface is rough, reflected rays are random

The camera receive the same amount of light regardless of its orientation
Example of diffuse surfaces

Rusted metal
Example of diffuse surfaces

Wool, fabric, fur
Why wet surfaces appear more specular that dry one?
(1 minute alone)
(2 minutes with your neighbors)
(5 minutes with the whole group)
Dry material
Wet material

Water fill the rugosity of the material

The more water present in a material the smoother the surface will appear.
Complexe scene : Direct lighting

Light that bounce only once before reaching the camera
Complexe scene : Indirect Lighting

Light that bounce more than once before reaching the camera

Will contribute to the final but is harder to compute
Indirect Lighting Simplification: ambient lighting

We replace indirect lighting with a value for the constant for all the scene

Ambient Lighting \approx \text{average color in the scene}
Is it possible for a scene to have only indirect Lighting? If so what would this scene look like?

(1 minute alone)

(2 minutes with your neighbors)

(5 minutes with the whole group)
Densely Clouded scene

Almost no variation of color
Phong Model

Ambient Light + Diffuse Light + Specular Light = Phong model
Phong Model : Ambient Light

\[ \text{Ambient} = i_a \]
Phong Model: Diffuse Light

$$Diffuse = max\left(\text{dot}\left(\vec{L}, \vec{N}\right), 0\right)$$

N Normal to the surface

L : Light direction

All vectors are normalized
Phong Model: Specular Light

\[ Specular = \max \left( \text{dot} \left( \frac{\vec{R}}{\| \vec{R} \|}, \frac{\vec{V}}{\| \vec{V} \|} \right), 0 \right)^{\alpha} \]

- \( \vec{N} \): Normal vector
- \( \vec{L} \): Light direction
- \( \vec{V} \): View vector, the direction toward the camera
- \( \vec{R} \): Reflected vector direction
- \( \alpha \): Shininess of the material

All vectors are normalized.
Reminder

All vectors are normalized

\[ \vec{R} = 2 \times \text{dot} \left( \vec{L}, \vec{N} \right) \vec{N} - \vec{L} \]
Phong Model: Mixing

\[ \text{phong} = \text{Ambient} \times k_a + \text{Diffuse} \times k_d + \text{Specular} \times k_s \]

\[ k_a \quad k_d \quad k_s \quad \rightarrow \quad \text{Mixing coefficient} \]
Problem:

Currently we only know how to compute:

- The fragment position in screen
- The fragment depth

We need to know:

V, N, L
Normals: stored in vertices

Vertices list
\{v0x,v0y,v0z, n0x,n0y,n0z
v1x,v1y,v1z, n1x,n1y,n1z
v2x,v2y,v2z, n2x,n2y,n2z\}

Indices list
\{0,1,2\}

Unchanged
Vertex shader

\[
\begin{align*}
v_{tmp} &= \begin{bmatrix} \text{proj} \end{bmatrix} \begin{bmatrix} \text{view} \end{bmatrix} v \\
u' &= \frac{v_{tmp}}{v_{tmp} \cdot \mathbf{w}} \\
\mathbf{N} &= \mathbf{n} \\
\mathbf{V} &= \text{cameraPosition} - v \\
\mathbf{L} &= \text{lightPosition} - v
\end{align*}
\]
Vertex shader

\[ v_{tmp} = \begin{bmatrix} proj \end{bmatrix} \begin{bmatrix} view \end{bmatrix} v \]

\[ v' = \frac{v_{tmp}}{v_{tmp} \cdot w} \]

\[ \overrightarrow{N} = n \]

\[ \overrightarrow{V} = \text{cameraPosition} - v \]

\[ \overrightarrow{L} = \text{lightPosition} - v \]
Rasterizer

\[ p = \lambda_0 v_0 + \lambda_1 v_1 + \lambda_2 v_2 \]

\[ f.interploated = \lambda_0 v_0.attributes + \lambda_1 v_1.attributes + \lambda_2 v_2.attributes \]

Warning: for this operation we need to use 3D barycentric coordinates
Fragment Shader

$$\text{phong} = \text{Ambient} \times k_a + \text{Diffuse} \times k_d + \text{Specular} \times k_s$$

$$\text{fragmentColor} = \text{objectColor} \times \text{phong}$$
Results
Next lecture: flipped classroom

In this lecture is dedicated to question answering

To prepare:
   Read all previous lectures
   Look at all the previous practical
   Come with questions