Information Visualization

W06: Exercise – Shader Programming

Graduation School of System Informatics

Department of Computational Science

Naohisa Sakamoto, Akira Kageyama

April.25, 2018

Schedule

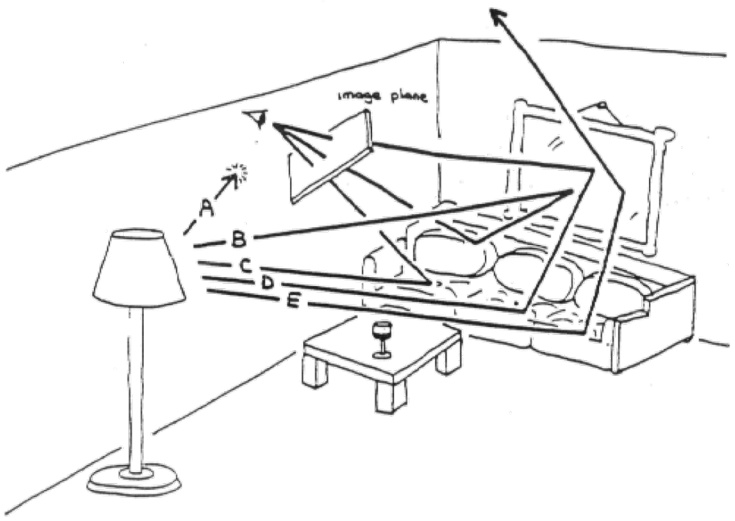
- W01 4/10 Guidance
- W02 4/11 Exercise (Setup)
- W03 4/17 Introduction to Data Visualization
- W04 4/18 Exercise (JavaScript Programming)
- W05 4/24 Computer Graphics
- W06 4/25 Exercise (Shader Programming)
- W07 5/01 Visualization Pipeline
- W08 5/02 Exercise (Data Model and Transfer Function)
- W09 5/08 Volume Visualization
- W10 5/09 Exercise (Isosurfaces and Volume Rendering)
- W11 5/22 Flow Visualization
- W12 5/23 Exercise (Streamlines and Line Integral Convolution)
- W13 5/29 Workshops 1
- W14 5/30 Workshops 2
- W15 6/05 Presentations

Table of Contents

Table of Contents

Illumination

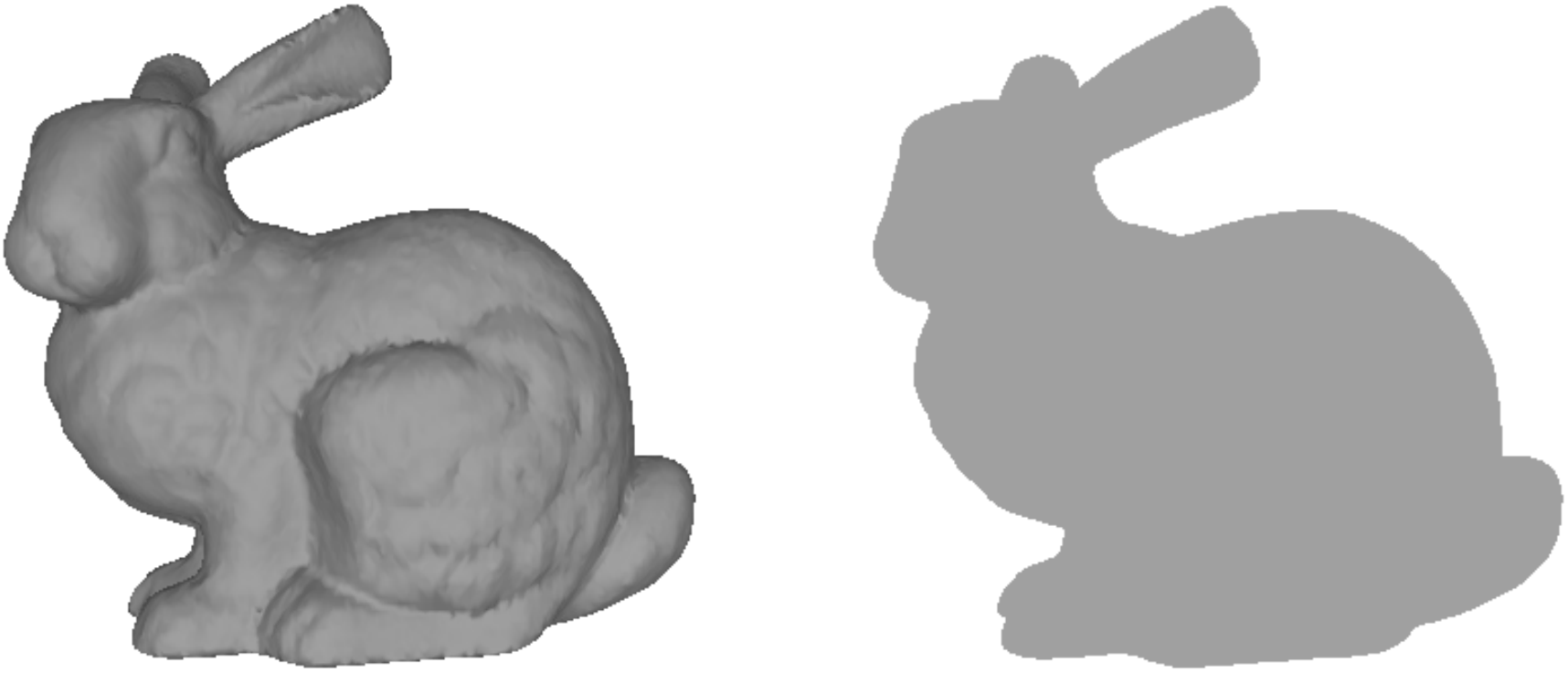
- Illumination model (Lighting)
  - From Physics we can derive models, called "illumination models", of how light reflects from surfaces and produces what we perceive as color.
  - In general, light leaves some light source, e.g. a lamp or the sun, is reflected from many surfaces and then finally reflected to our eyes, or through an image plane of a camera.



https://www.siggraph.org/education/materials/HyperGraph/illumin/illum1.htm

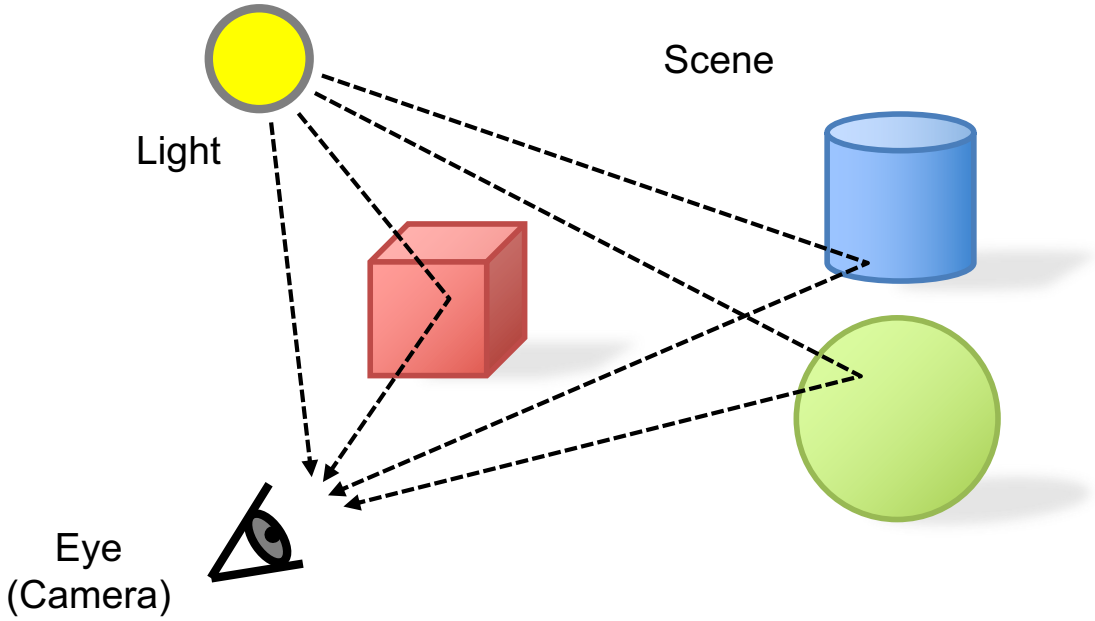
Illumination

- Lighting
  - Stanford bunny with and without lighting effects



Illumination Model

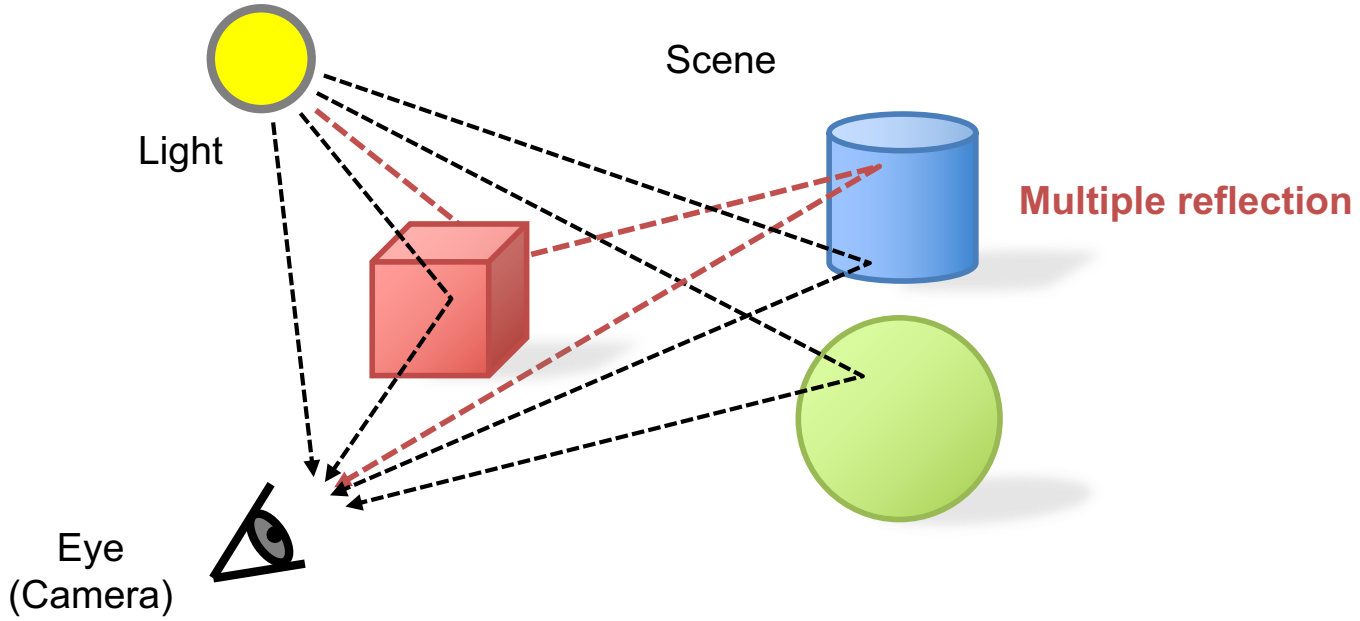
- Local Illumination Model
  - The contribution from the light that goes directly from the light source and is reflected from the surface is called a "local illumination model".



https://www.siggraph.org/education/materials/HyperGraph/illumin/illum1.htm

Illumination Model

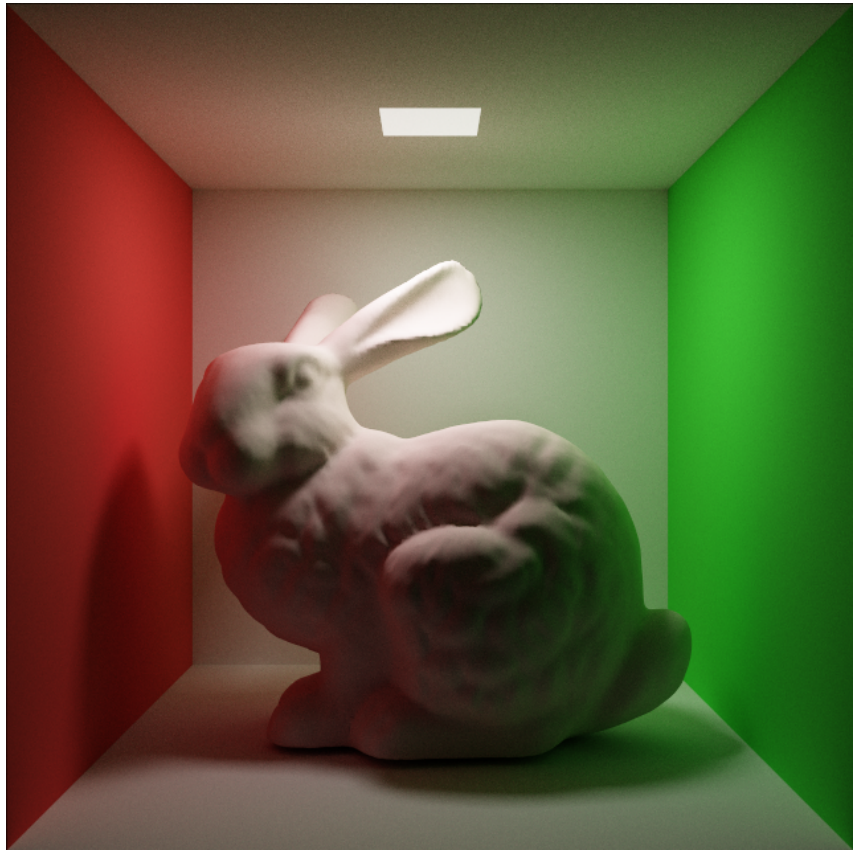
- Global Illumination Model
  - A "global illumination model" adds to the local model the light that is reflected from other surfaces to the current surface.

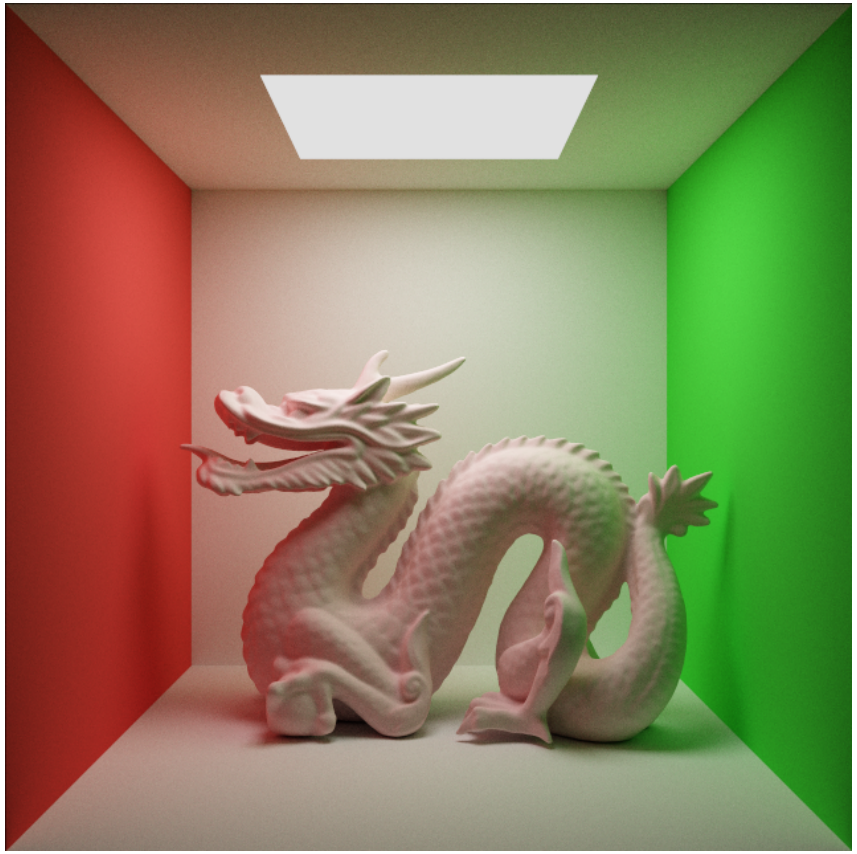


https://www.siggraph.org/education/materials/HyperGraph/illumin/illum1.htm

Illumination

- Global Illumination Model



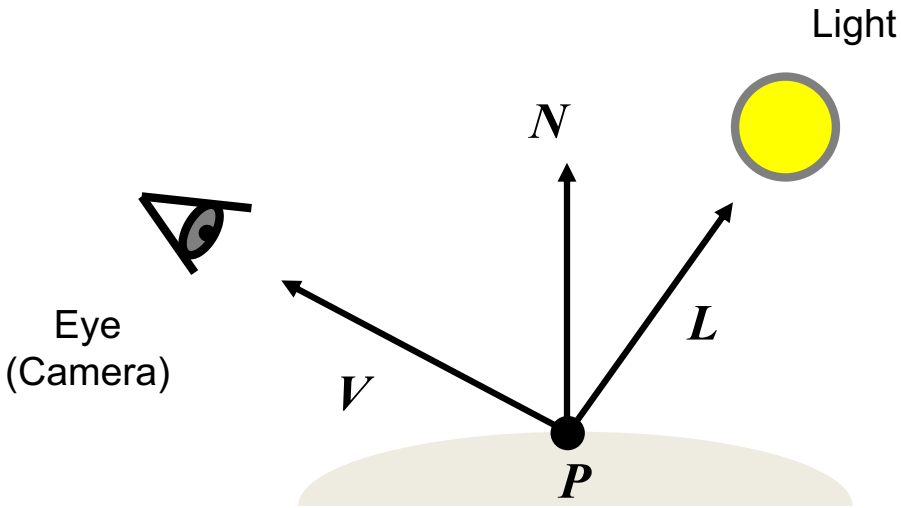


http://srodal.net/oldray/

Table of Contents

Reflection

- The intensity (luminosity) of the reflected light can be calculated by the following components.
  - Ambient reflection
  - Diffuse reflection
  - Specular reflection



P	Point on a surface
N	Normal vector at P
L	Light direction at P
V	Viewing direction at P

Ambient Reflection

- Reflection of light from the environment and assumed to be equal in all directions.
  - Independent of light position, object orientation, camera position or orientation

$$I_{ambient} = I_a \, k_a$$

$I_{ambient}$	Intensity of the ambient reflection
I_a	Ambient intensity
k_a	Ambient reflection coefficient

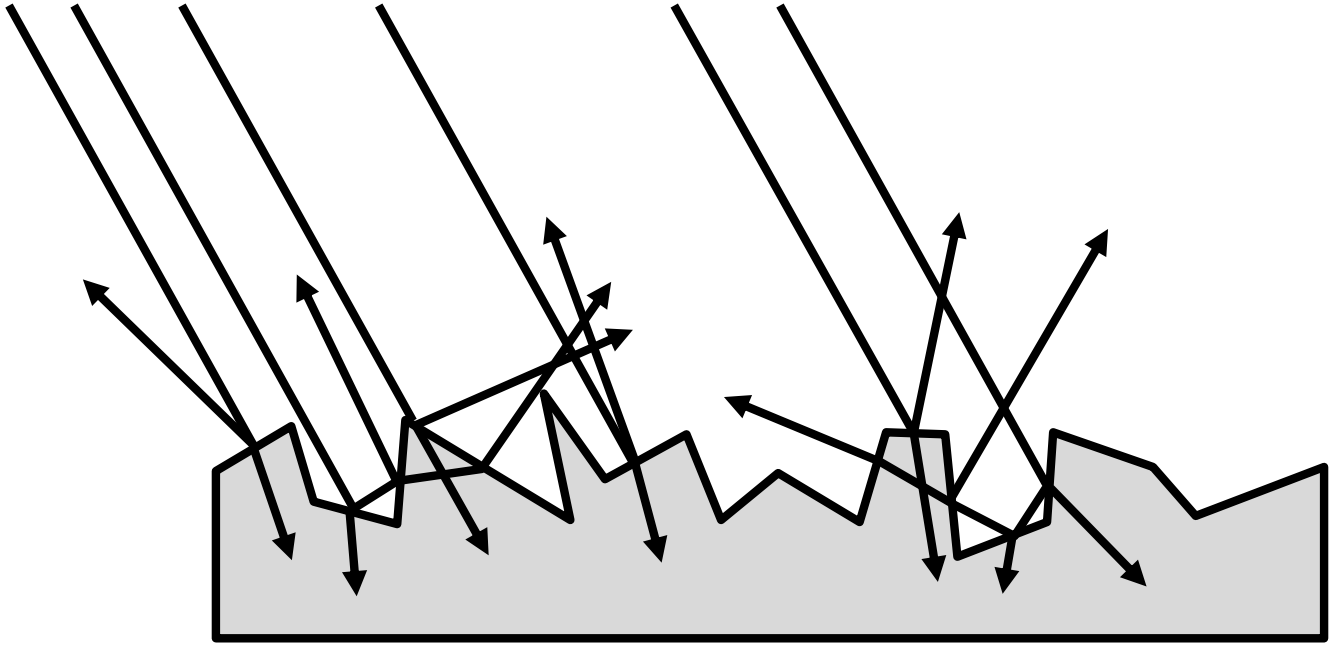
Ambient Reflection

- Effect of ambient reflection coefficient k_a

$k_a = 0.2$ $k_a = 0.6$ $k_a = 1.0$

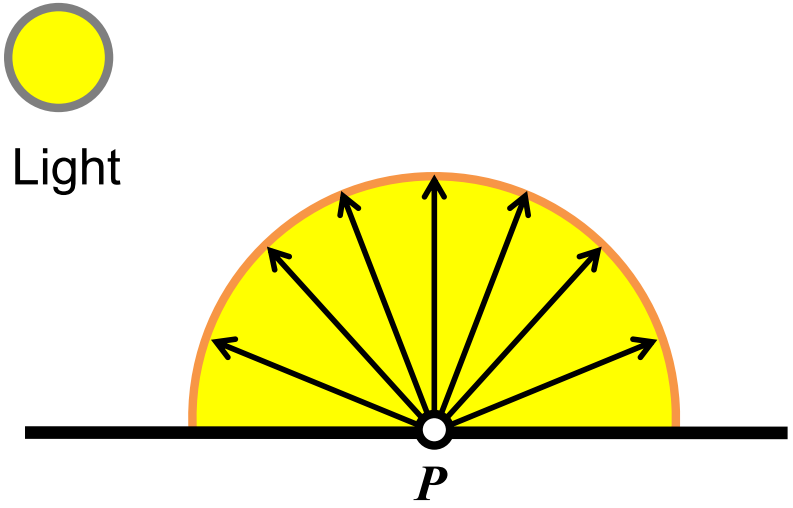
Diffuse Reflection

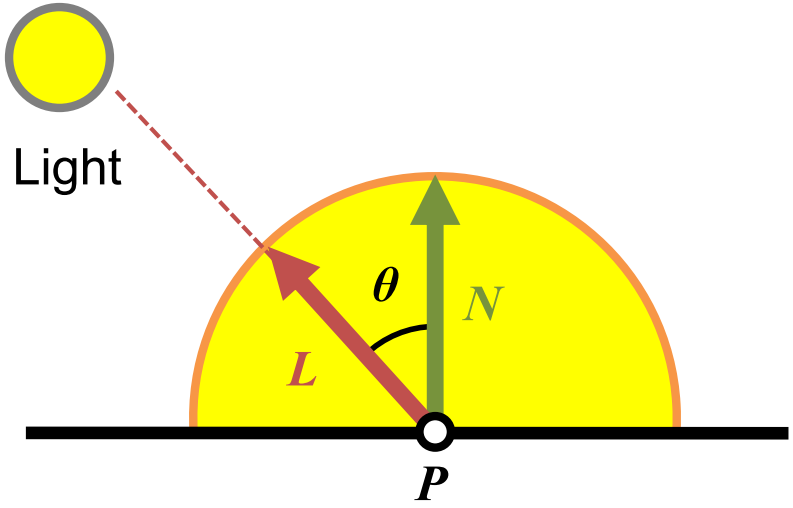
- Reflection of light from a surface such that incident light is reflected into many different directions.



Diffuse Reflection

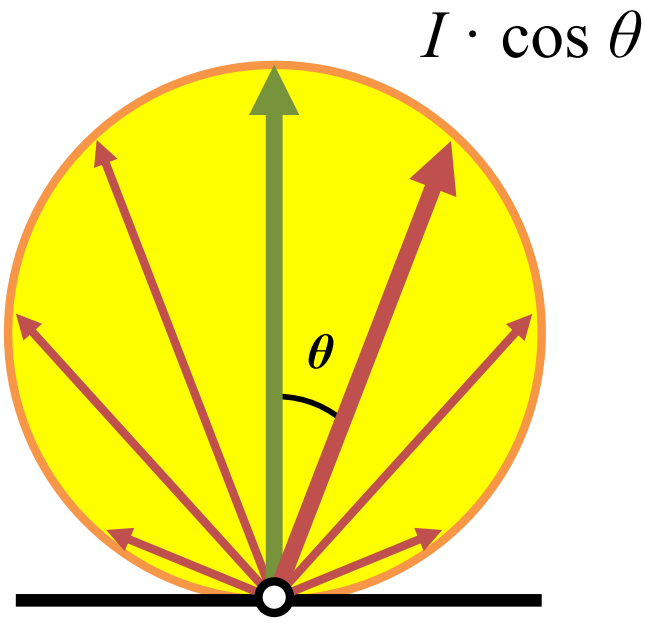
- Reflected intensity does not depend on the viewing direction.
- Incoming light depends on the light direction.





Diffuse Reflection

- Lambert's cosine law
  - Radiant intensity observed from an ideal diffusely reflecting surface is directly proportional to the cosine of the angle θ between the direction of the incident light and the surface normal.



https://en.wikipedia.org/wiki/Lambert%27s_cosine_law

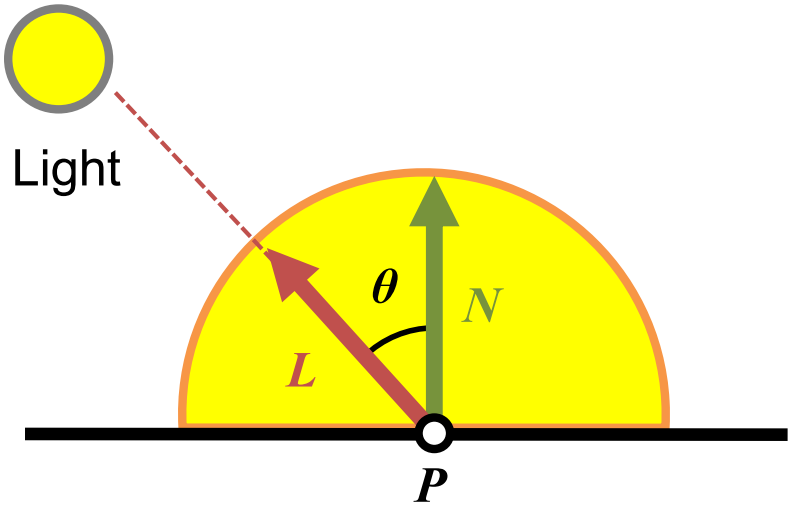
Diffuse Reflection

- Calculation of diffuse reflection

$$I_{diffuse} = I_i \, k_d \cos\theta$$

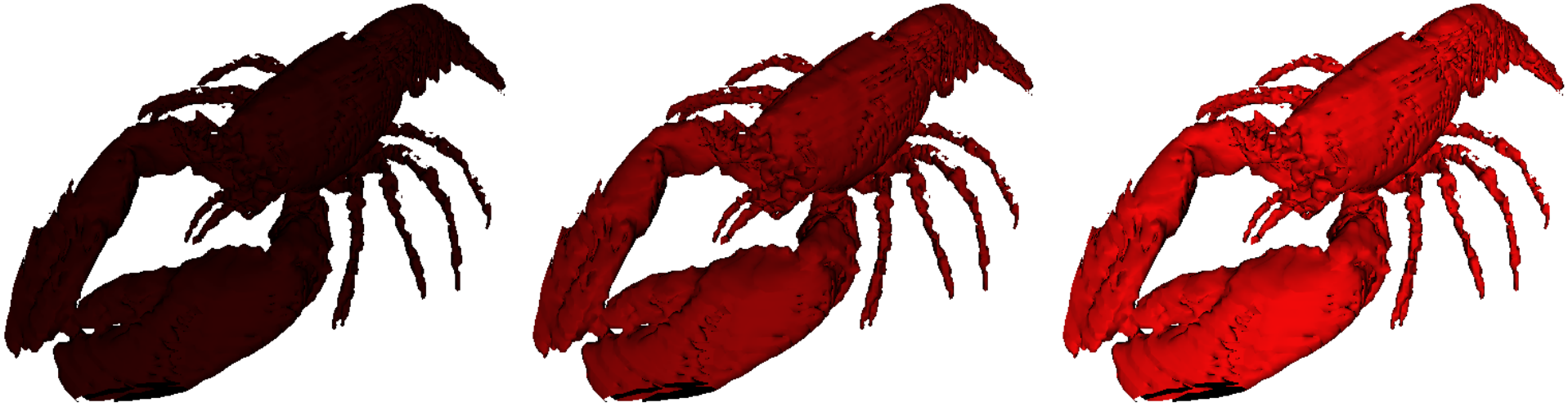
$$= I_i \, k_d \, (\boldsymbol{N} \cdot \boldsymbol{L})$$

$I_{diffuse}$	Intensity of the diffuse reflection
I_i	Intensity of the light source
k_d	Diffuse reflection coefficient
$\boldsymbol{N}$	Normal vector (unit vector)
$\boldsymbol{L}$	Light direction (unit vector)



Diffuse Reflection

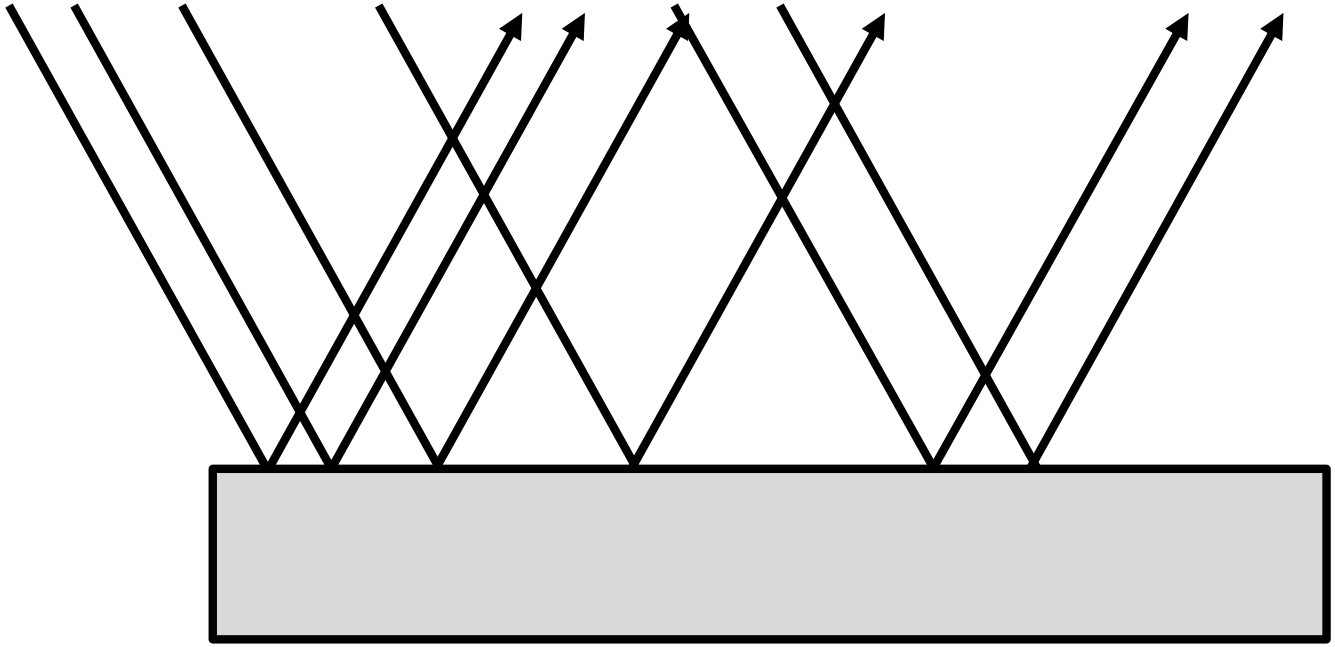
- Effect of diffuse reflection coefficient k_d



$k_d = 0.2$ $k_d = 0.6$ $k_d = 1.0$

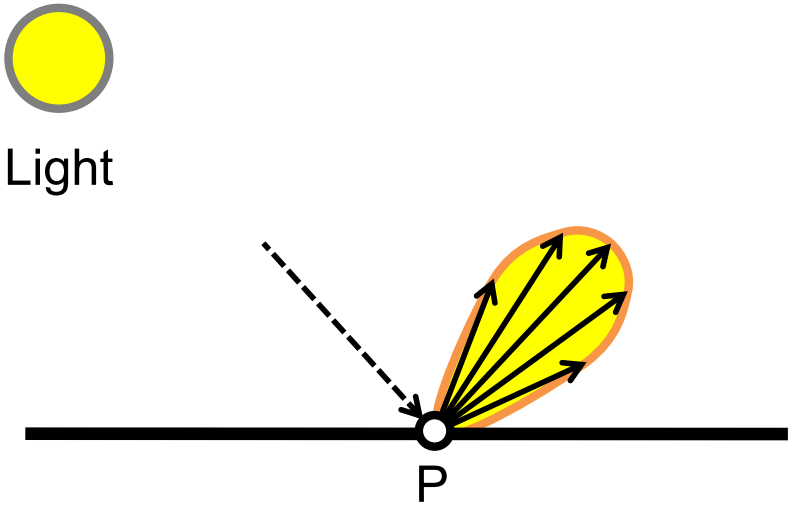
Specular Reflection

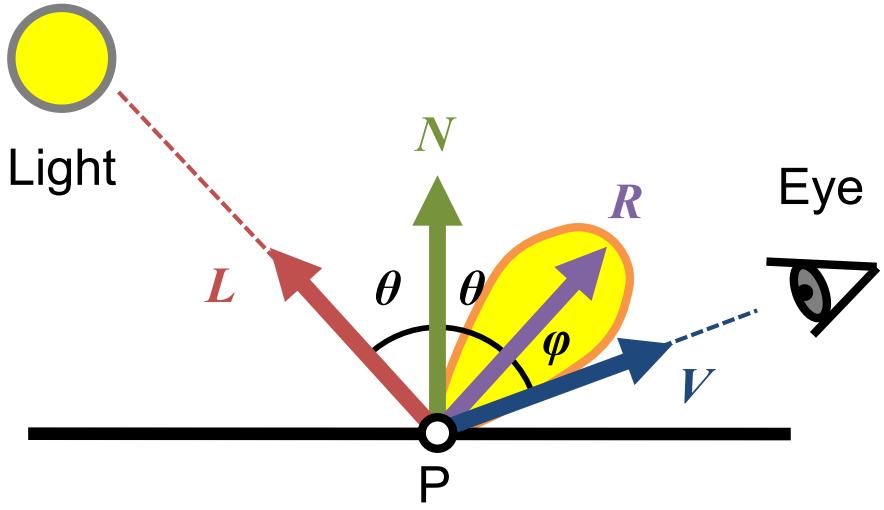
- Mirror-like reflection of light from a surface, in which light from a single incoming direction is reflected into a single outgoing direction.



Specular Reflection

- Reflected intensity does not depend on the viewing direction.
- Incoming light depends on the light direction.





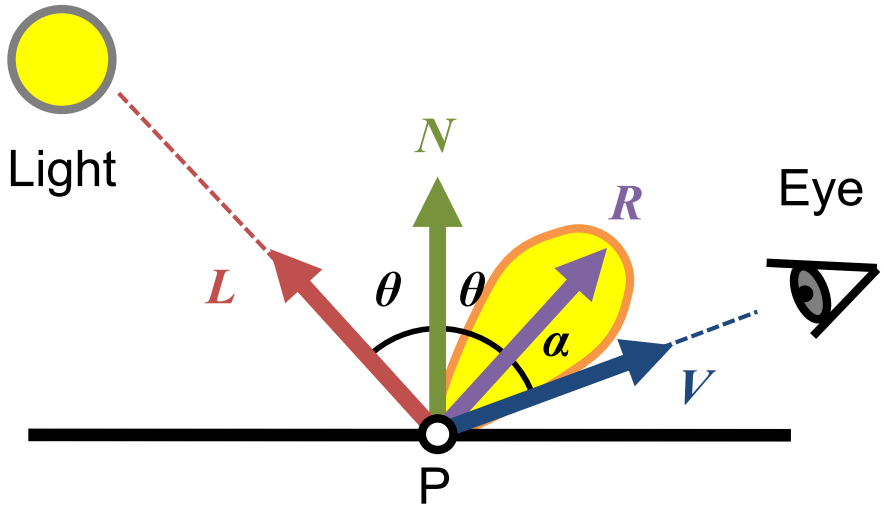
Specular Reflection

- Calculation of specular reflection

$$I_{specular} = I_i \, k_s \cos^n \alpha$$

$$= I_i \, k_s \, (\boldsymbol{V} \cdot \boldsymbol{R})^n$$

$I_{specular}$	Intensity of the specular reflection
I_i	Intensity of the light source
k_s	Specular reflection coefficient
$\boldsymbol{N}$	Normal vector (unit vector)
$\boldsymbol{L}$	Light direction (unit vector)
$\boldsymbol{V}$	Viewing direction (unit vector)
$\boldsymbol{R}$	Reflection vector of $\boldsymbol{L}$

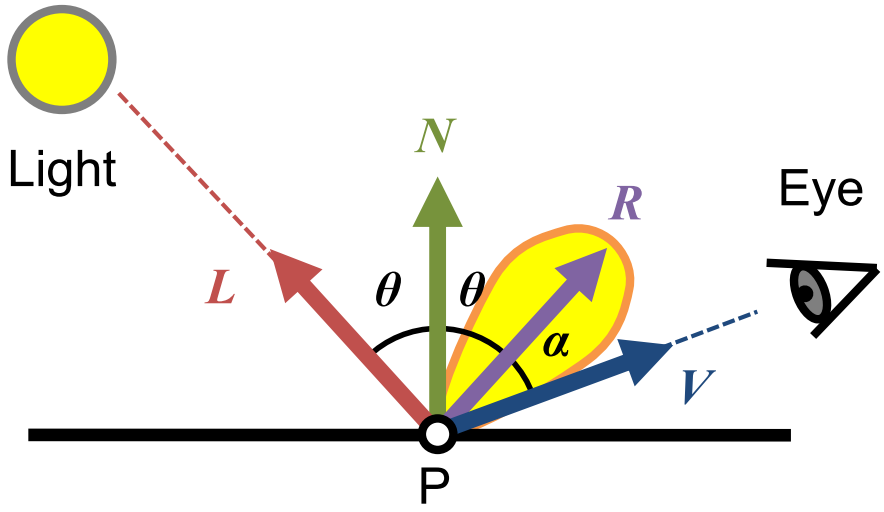


Specular Reflection

- Calculation of reflection vector $\boldsymbol{R}$

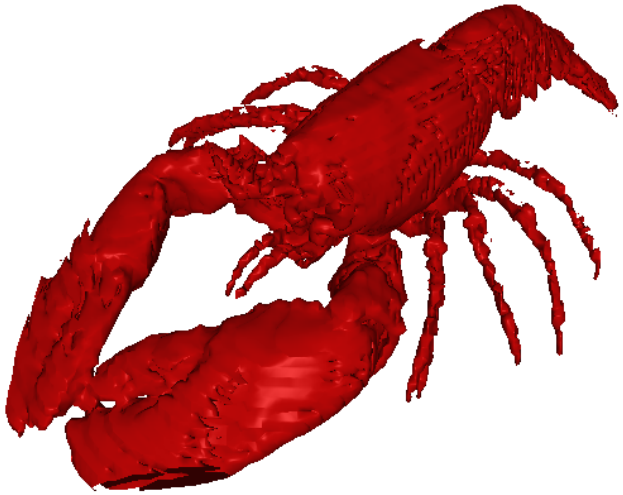
$$\boldsymbol{R} = 2\,(\boldsymbol{N} \cdot \boldsymbol{L})\,\boldsymbol{N} - \boldsymbol{L}$$

$I_{specular}$	Intensity of the specular reflection
I_i	Intensity of the light source
k_s	Specular reflection coefficient
$\boldsymbol{N}$	Normal vector (unit vector)
$\boldsymbol{L}$	Light direction (unit vector)
$\boldsymbol{V}$	Viewing direction (unit vector)
$\boldsymbol{R}$	Reflection vector of $\boldsymbol{L}$

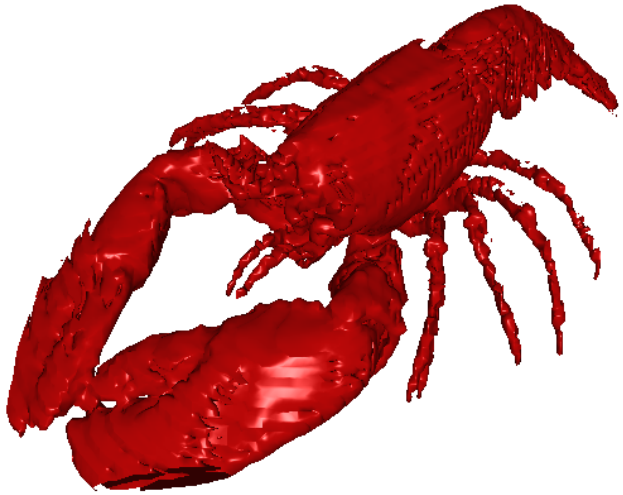


Specular Reflection

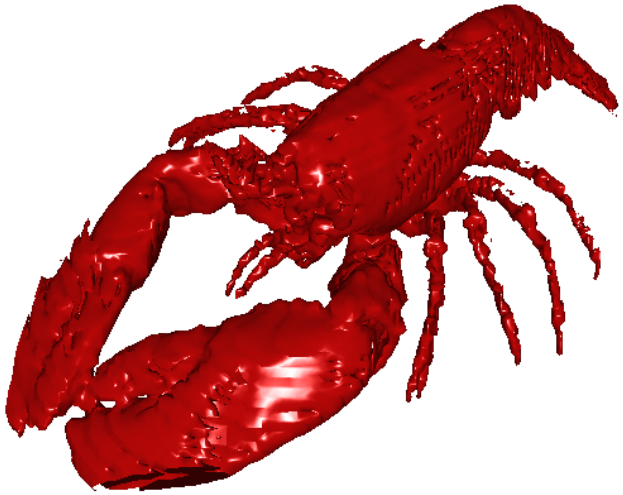
- Effect of specular reflection coefficient k_s



$k_s = 0.2$



$k_s = 0.6$

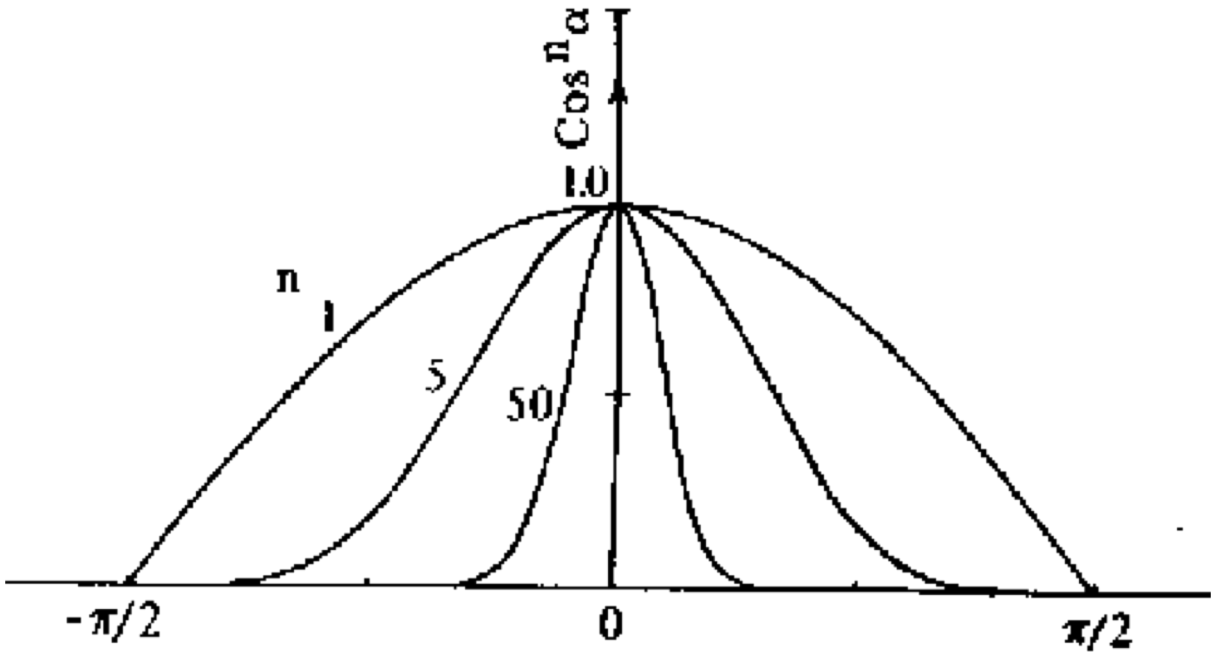


$k_s = 1.0$

Specular Reflection

- Shininess exponent n

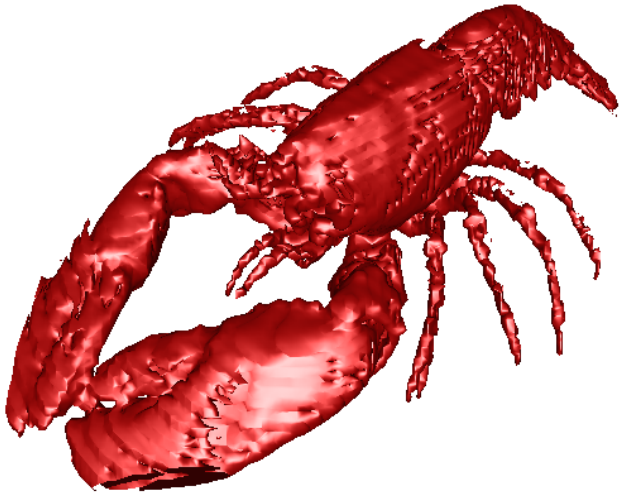
$$I_{specular} = I_i k_s \cos^n \alpha$$



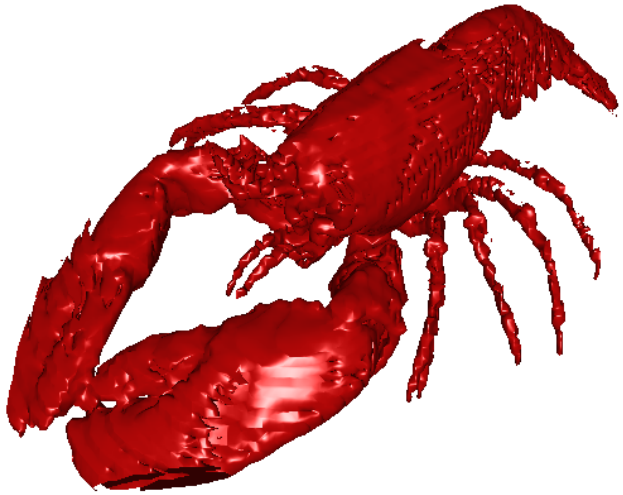
https://www.siggraph.org/education/materials/HyperGraph/illumin/specular_highlights/phong_model_specular_reflection.htm

Specular Reflection

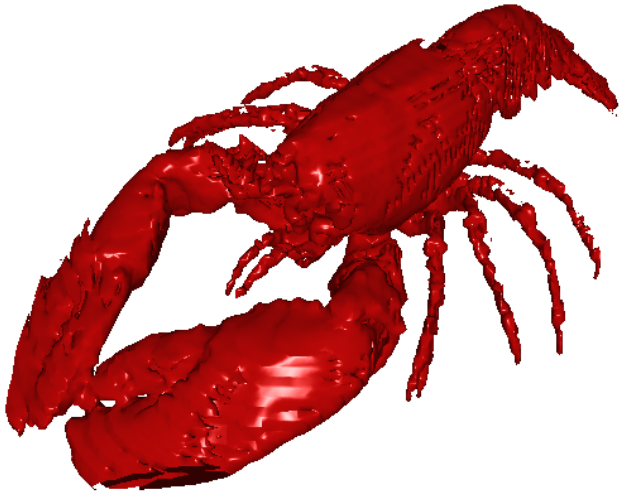
- Effect of shininess exponent n



$n = 10$



$n = 50$



$n = 150$

Reflection Model

- Lambertian reflection model

$$I = I_{ambient} + I_{diffuse}$$

$$= I_a k_a + I_i k_d (\boldsymbol{N} \cdot \boldsymbol{L})$$

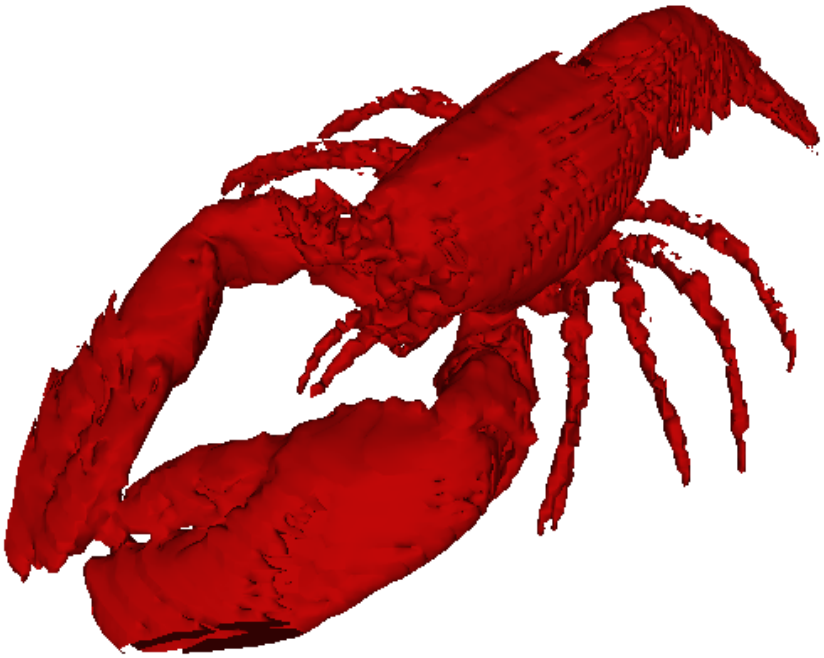
- Phong reflection model

$$I = I_{ambient} + I_{diffuse} + I_{specular}$$

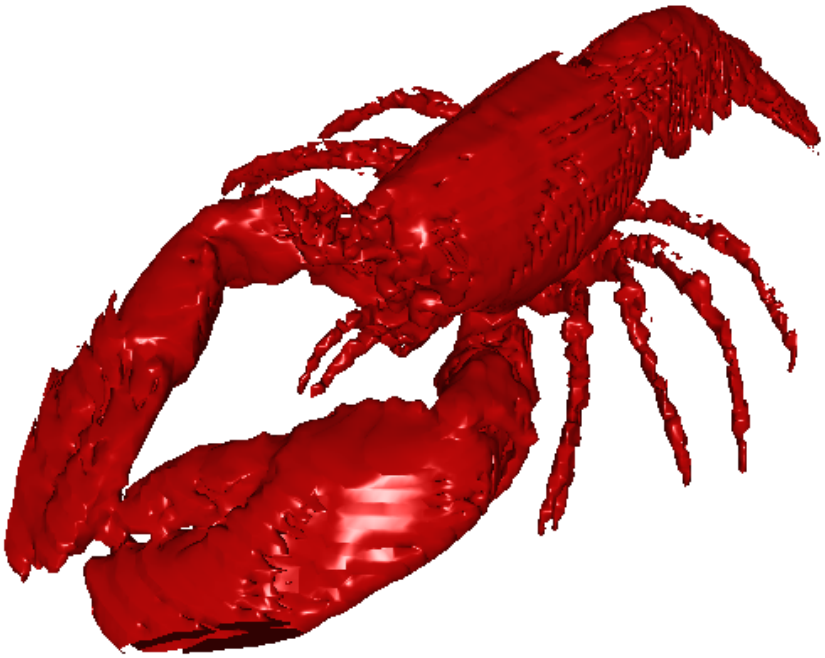
$$= I_a k_a + I_i k_d (\boldsymbol{N} \cdot \boldsymbol{L}) + I_i k_s (\boldsymbol{V} \cdot \boldsymbol{R})^n$$

Reflection Model

- Comparison with Lambertian (left) and Phong (right) reflection models



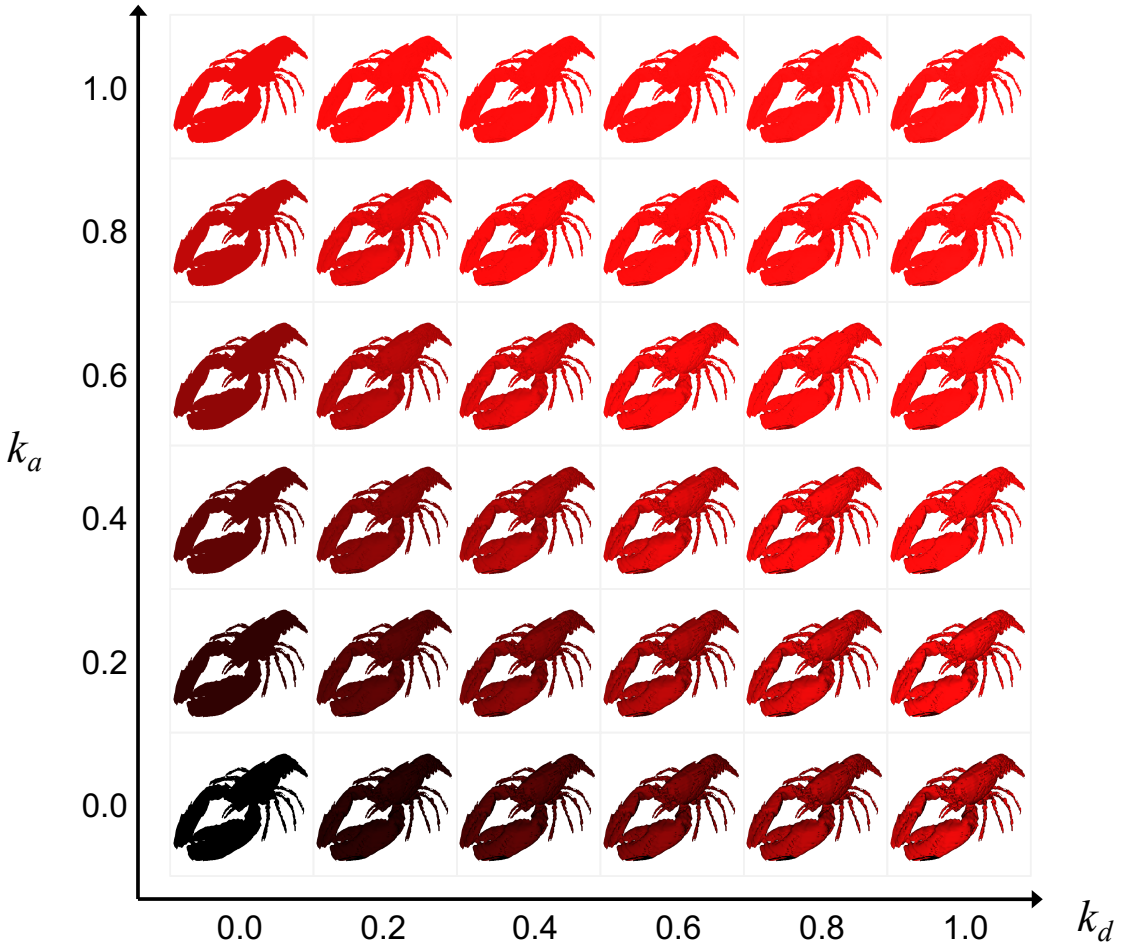
$k_a = 0.2, k_d = 0.6$



$k_a = 0.2, k_d = 0.6, k_s = 0.8, n = 100$

Lambertian Reflection Model

- Effect of k_a and k_d



Phong Reflection Model

- Effect of k_s and n

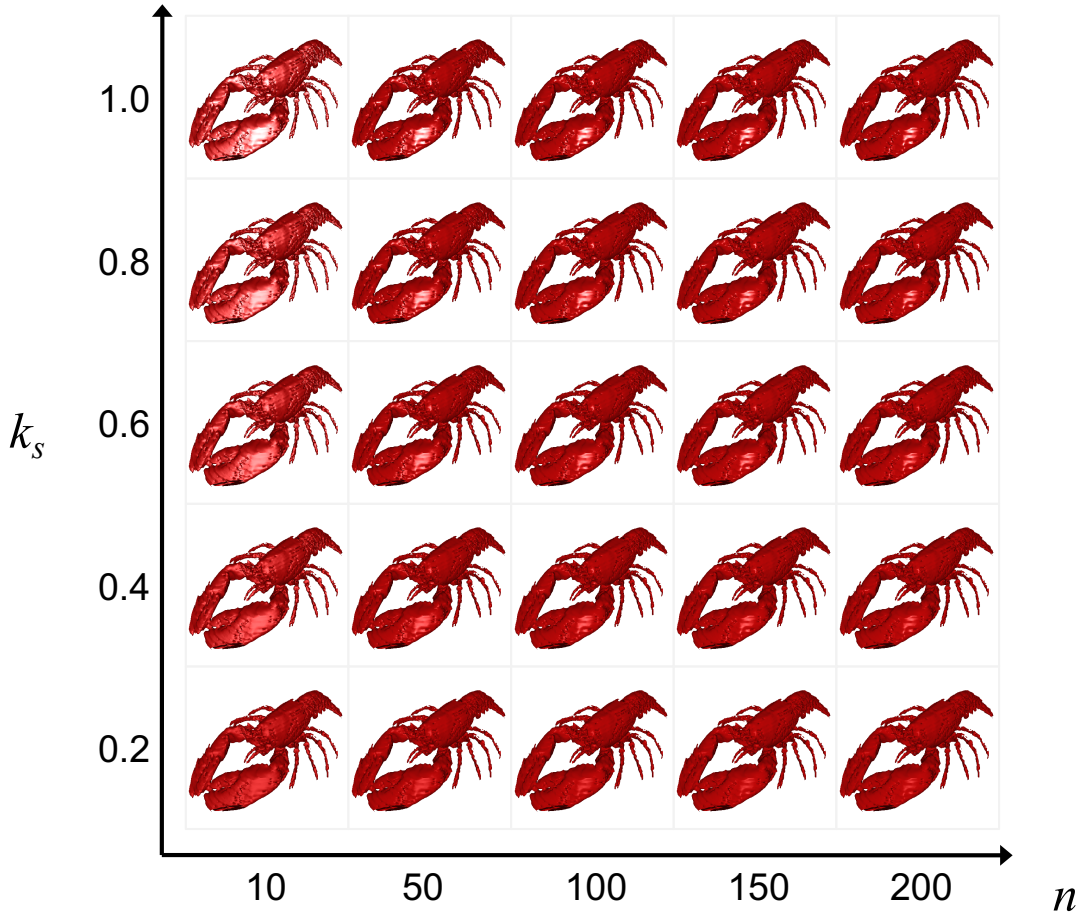


Table of Contents

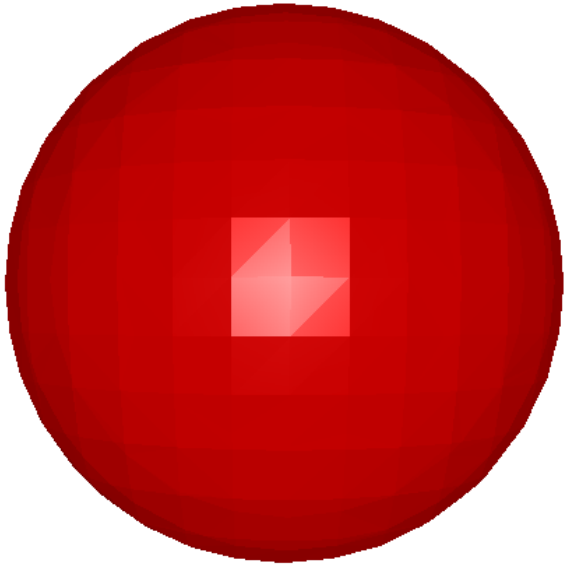
- Illumination Model
- Reflection Model
- Shading Model
- Shader Programming

Shading

- Process of determining the color of a pixel based on the reflection models
  - Flat shading
  - Smooth shading

Flat Shading

- Uses the same color for every pixel in a face
  - Less computationally expensive
  - Edges appear
  - Not suitable for smooth objects

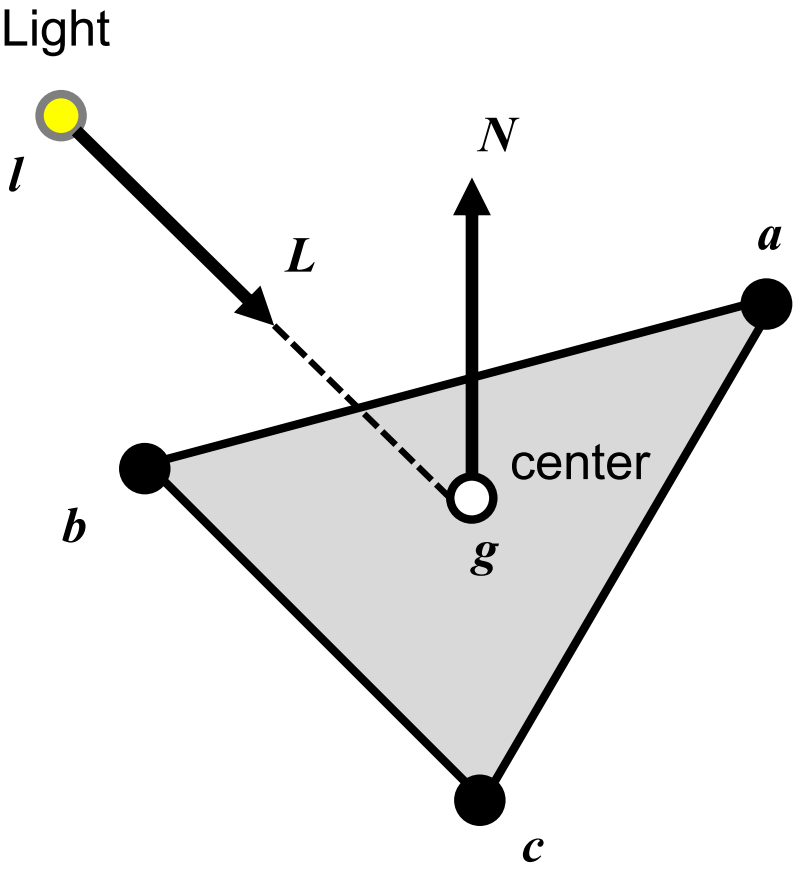


Flat Shading

- Calculation of normal vector $\boldsymbol{N}$ and light direction vector $\boldsymbol{L}$ for a face

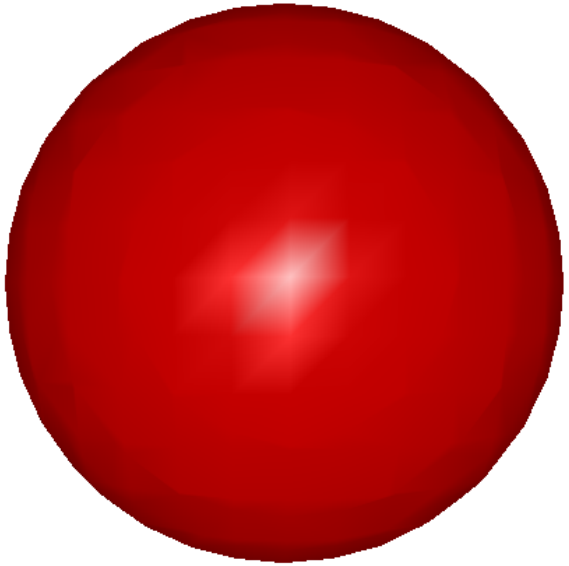
$$\vec{N} = \frac{(\vec{b}-\vec{a}) \times (\vec{c}-\vec{a})}{\left|(\vec{b}-\vec{a}) \times (\vec{c}-\vec{a})\right|}$$

$$\vec{L} = \frac{\vec{l}-\vec{g}}{\left|\vec{l}-\vec{g}\right|}$$



Smooth Shading

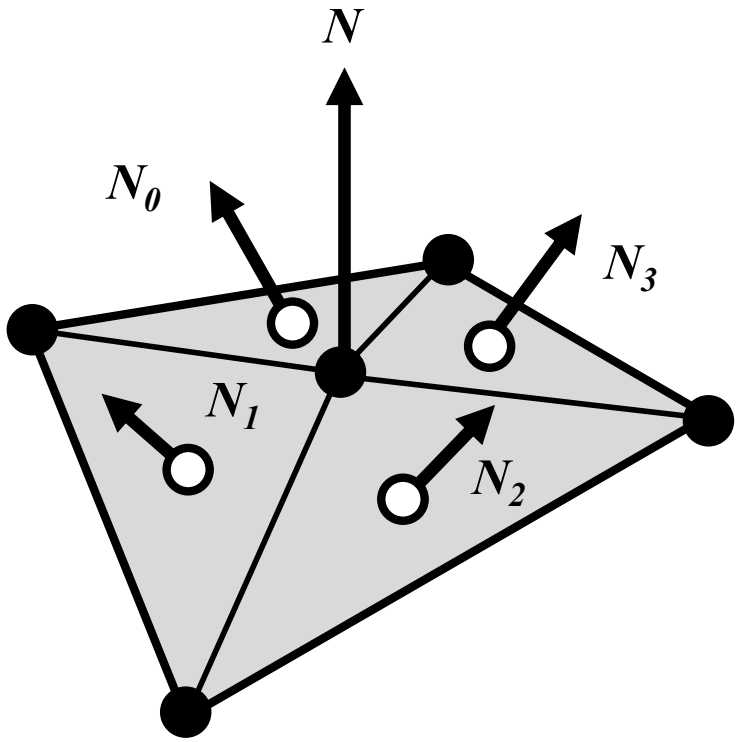
- Uses linear interpolation of colors or normal vectors between vertices
  - More computationally expensive
  - Edges disappear
  - Suitable for any objects



Smooth Shading

- Calculation of normal vector N for a vertex

$$\vec{N} = \frac{\vec{N}_0 + \vec{N}_1 + \vec{N}_2 + \vec{N}_3}{\left|\vec{N}_0 + \vec{N}_1 + \vec{N}_2 + \vec{N}_3\right|}$$

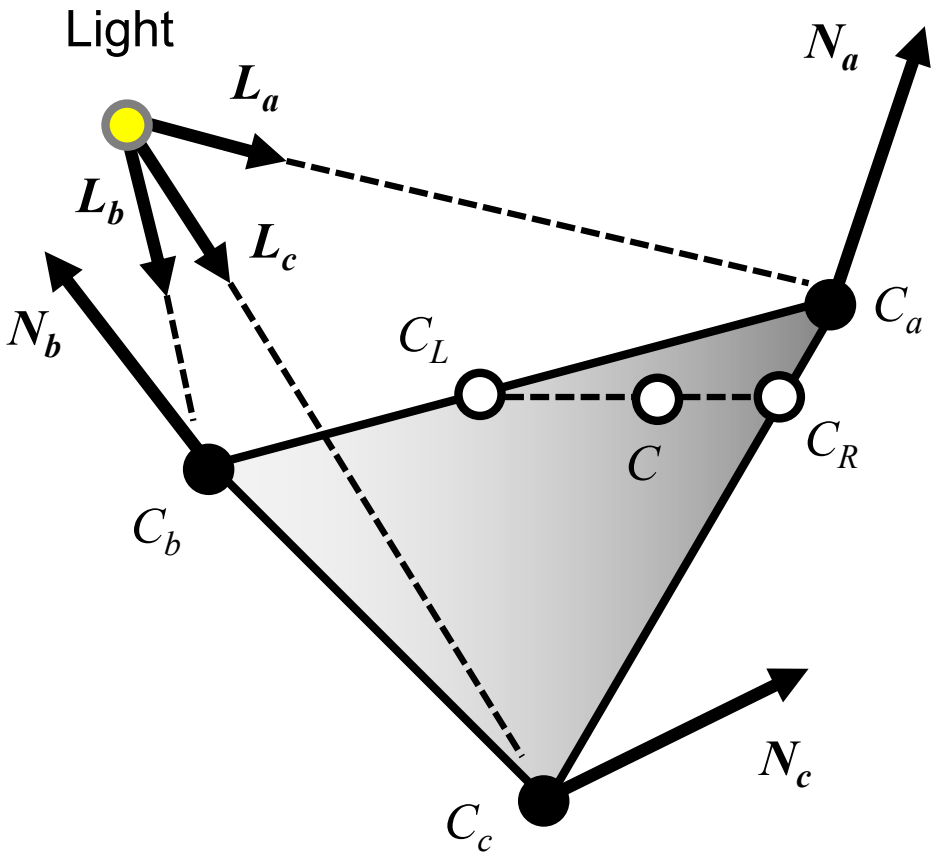


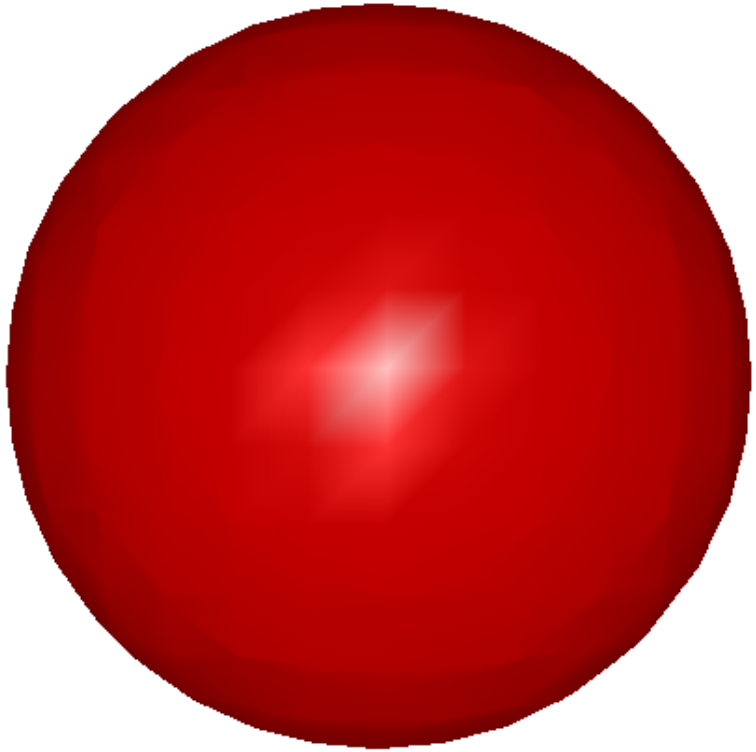
Types of Smooth Shading

- Gouraud shading
  - Color interpolation shading
- Phong shading
  - Normal interpolation shading

Gouraud Shading

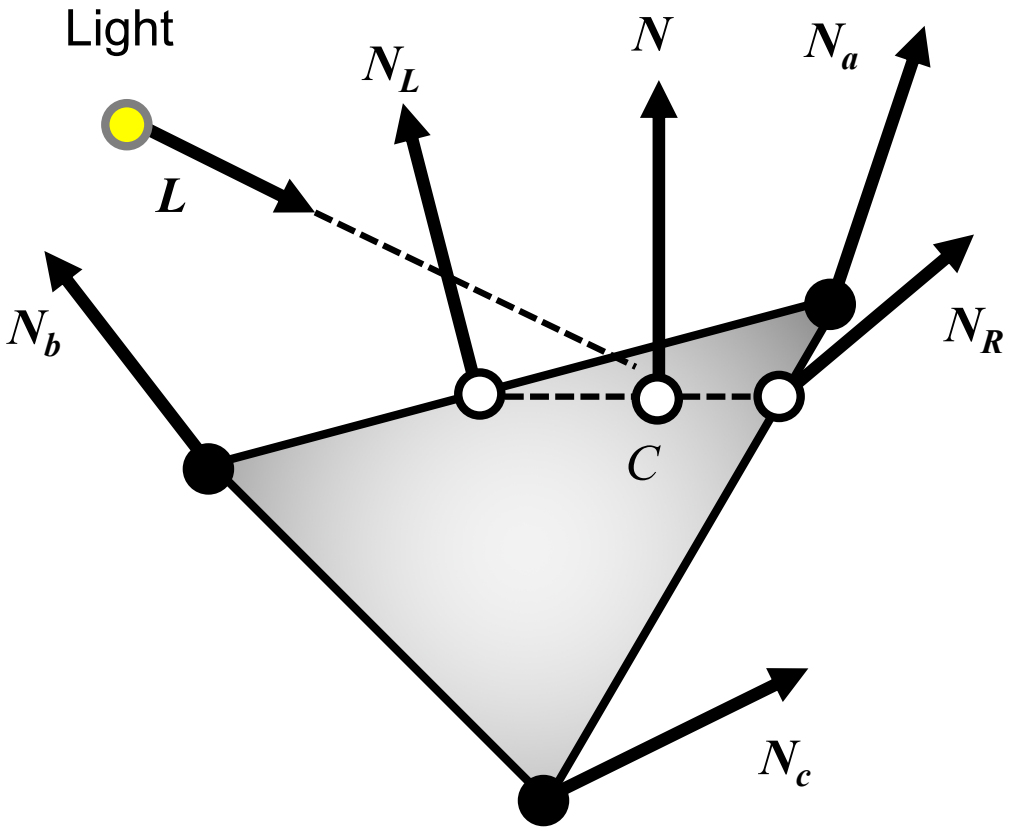
- Colors are interpolated across the face
- Reflection model used at each vertex

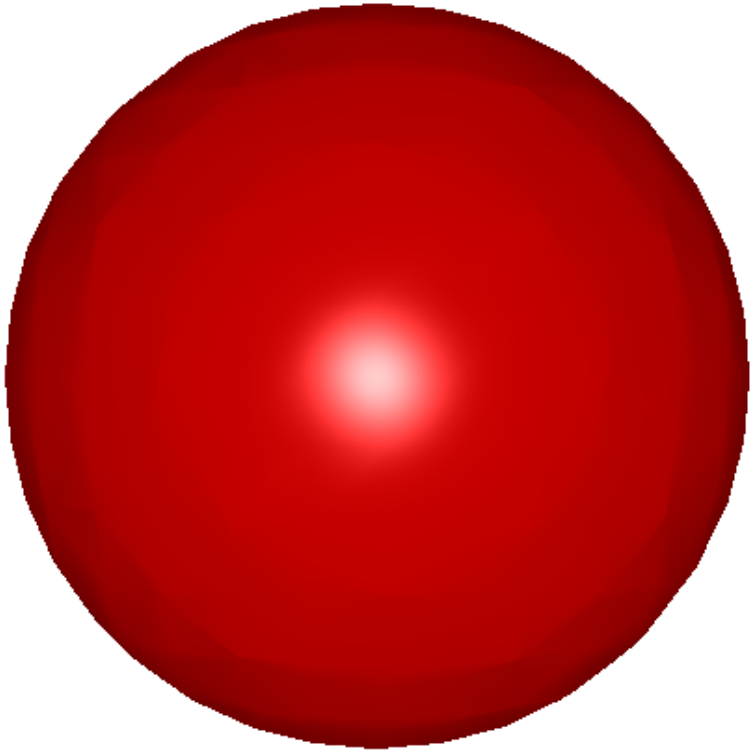




Phong Shading

- Normals are interpolated across the face
- Reflection model used at each pixel





Interpolation

- Linear interpolation

$$S = (1 - t)\, S_0 + t\, S_1$$

Interpolation

- Bilinear interpolation

$$S = (1 - t)\, S_{\mathrm{a}} + t\, S_{\mathrm{b}}$$

$$S_a = (1 - t_a)\, S_0 + t_a\, S_1$$

$$S_b = (1 - t_b)\, S_0 + t_b\, S_2$$

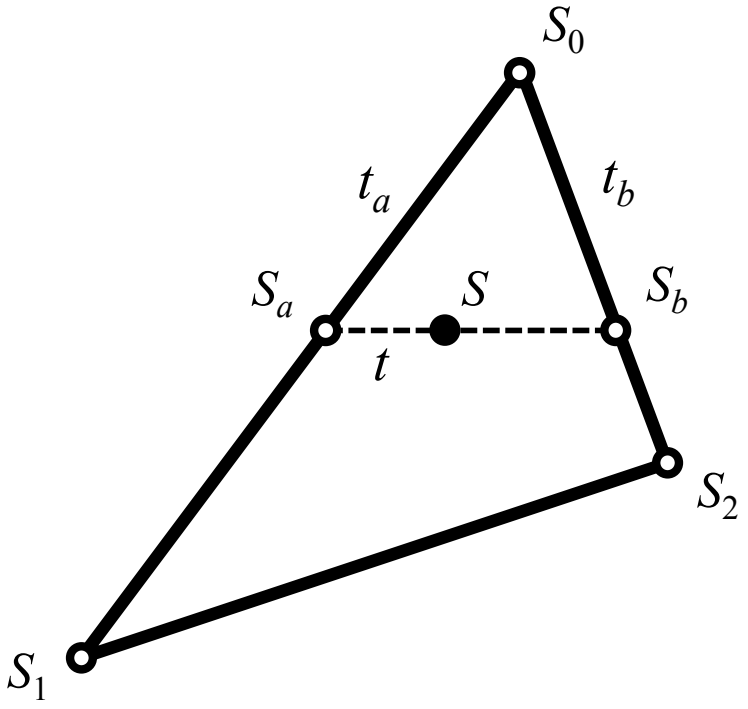


Table of Contents

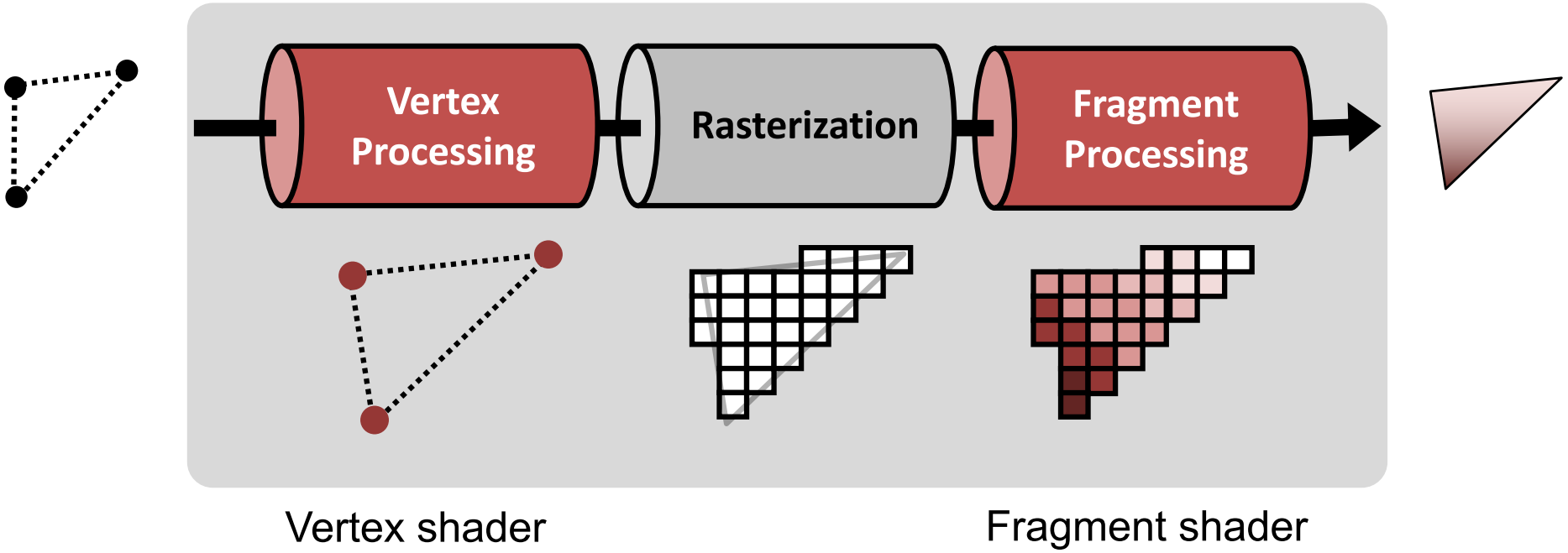
Shader

- OpenGL Shading Language (GLSL)
  - High-level shading language based on the syntax of the C programming language.
  - Created by the OpenGL ARB (OpenGL Architecture Review Board) to give developers more direct control of the graphics pipeline without having to use ARB assembly language or hardware-specific languages.

https://en.wikipedia.org/wiki/OpenGL_Shading_Language

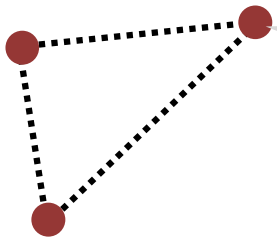
Shader Programming

- Rendering pipeline
  - Vertex shader
  - Fragment shader



Vertex Shader

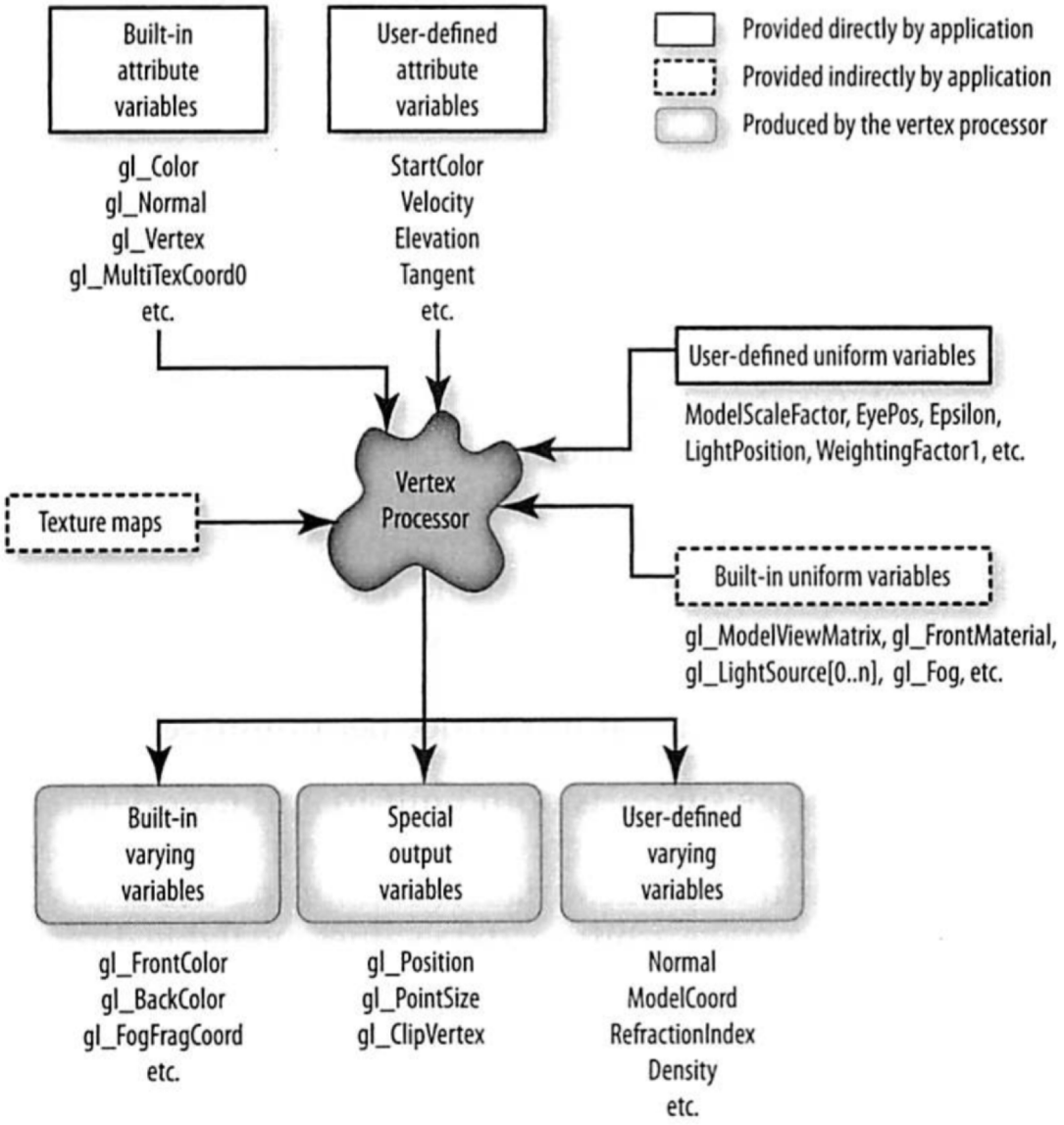
- Programmable Shader stage in the rendering pipeline that handles the processing of individual vertices.



```
void main()
{
    vec4 v = vec4( position, 1.0 );
    mat4 MV = modelViewMatrix;
    mat4 P = projectionMatrix;
    gl_Position = P * MV * v;
}
```

https://en.wikipedia.org/wiki/OpenGL_Shading_Language

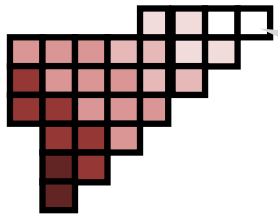
Vertex Shader Inputs and Outputs



[OpenGL Shading Language]

Fragment Shader

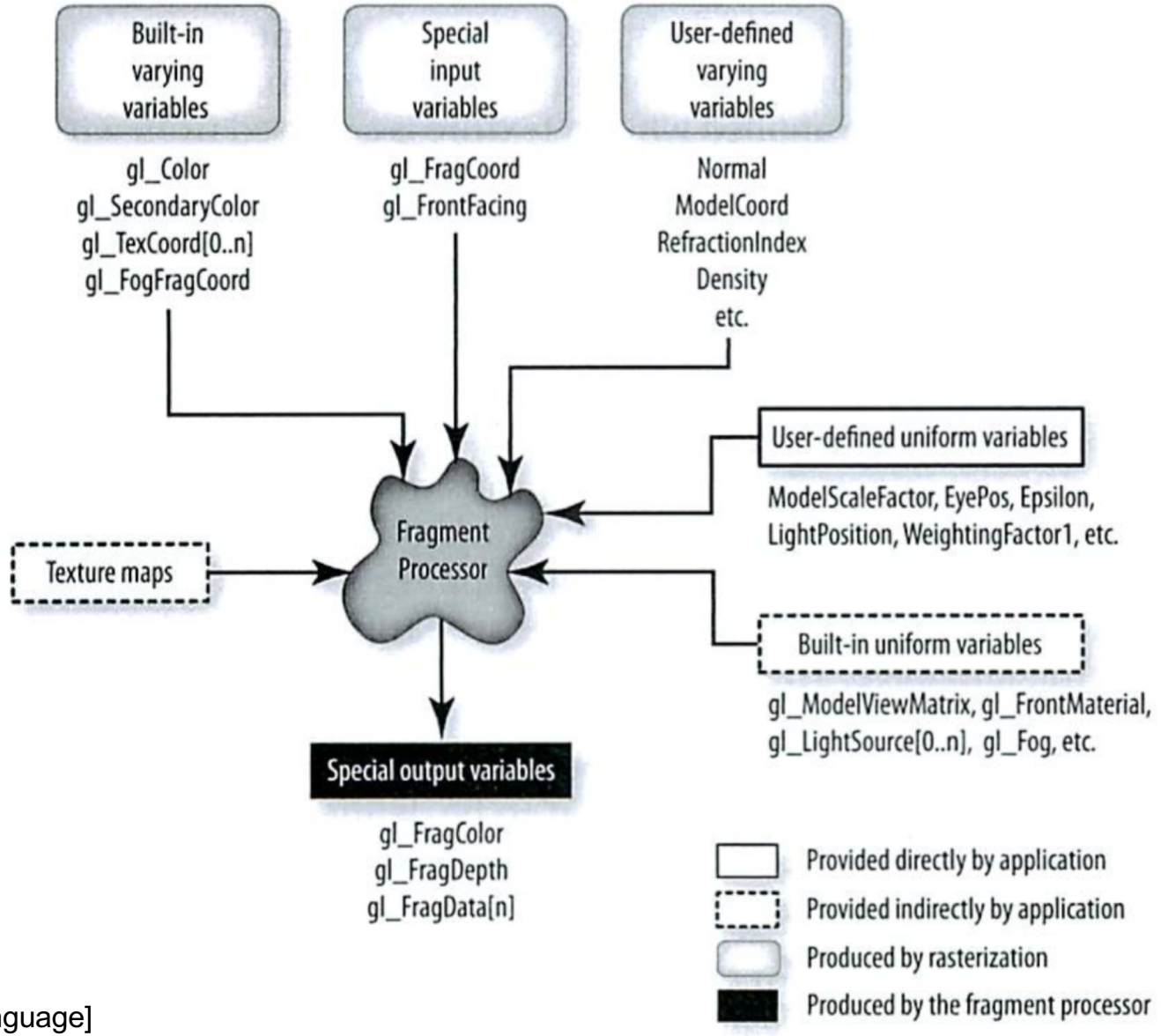
- Programmable Shader stage that will processes a Fragment generated by the Rasterization into a set of colors and a single depth value.



```
void main()
{
    vec4 C = vec4(0.8,0.1,0.1,1.0);
    gl_FragColor = C;
}
```

https://en.wikipedia.org/wiki/OpenGL_Shading_Language

Fragment Shader Inputs and Outputs



[OpenGL Shading Language]

Implementation

- Gouraud shading
  - Vertex shader

```
void main()
{
    point_position = modelViewMatrix * vec4( position, 1.0 );
    normal_vector = normalMatrix * normal;

    vec3 C = color;
    vec3 L = normalize( light_position - point_position.xyz );
    vec3 N = normalize( normal_vector );

    point_color = LambertianReflection( C, L, N );
    gl_Position = projectionMatrix * point_position;
}
```

Implementation

- Gouraud shading
  - Fragment shader

```
void main()
{
    gl_FragColor = vec4( point_color, 1.0 );
}
```

Implementation

- Phong shading
  - Vertex shader

```
void main()
{
    point_color = color;
    point_position = modelViewMatrix * vec4( position, 1.0 );
    normal_vector = normalMatrix * normal;

    gl_Position = projectionMatrix * point_position;
}
```

Implementation

- Phong shading
  - Fragment shader

```
void main()
{
    vec3 C = point_color;
    vec3 L = normalize( light_position - point_position.xyz );
    vec3 N = normalize( normal_vector );

    vec3 shaded_color = LambertianReflection( C, L, N );
    gl_FragColor = vec4( shaded_color, 1.0 );
}
```

GLSL

- Types
  - void, bool, int, float
  - vec2, vec3, vec4
  - bvec2, bvec3, bvec4
  - ivec2, ivec3, ivec4
  - mat2, mat3, mat4
  - sampler2D, sampler3D
  - …

https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf

GLSL

- Vector Components
  - {x, y, z, w} represent points or normals
    e.g.: pos.x, pos.y, pos.z, pos.w
  - {r, g, b, a} represent colors
    e.g.: col.r, col.g, col.b, col.a
  - {s, t, p, q} represent texture coordinates
    e.g.: tex.s, tex.t, tex.p, tex.q
  - Swizzled and replicated
    e.g.: pos.xy, pos.xx, pos.zy, …

GLSL

- Matrix Components
  - Column-major
  - Constructors

    e.g.:
    ```
    mat2(float)
    mat2(vec2, vec2)
    mat2(float,float,float,float)
    ```
  - Access components of a matrix with array subscripting syntax

    e.g.)
    ```
    mat4 m;
    m[1] = vec4(2.0);
    m[0][0] = 1.0;
    ```

GLSL

- Qualifiers
  - `const` constant parameter
  - `attribute` input value of vertex shader
  - `uniform` global value
  - `varying` output value of vertex shader
input value of fragment shader

https://www.khronos.org/files/webgl/webgl-reference-card-1_0.pdf

GLSL

- Built-In Inputs and Outputs (Vertex Shader)
  - Input

    `gl_Position` transformed vertex position

    `gl_PointSize` transformed point size

GLSL

- Built-In Inputs and Outputs (Fragment Shader)
  - Inputs

`gl_FragCoord`	fragment position (framebuffer)
`gl_FrontFacing`	fragment belongs to a front-facing primitive
`gl_PointCoord`	fragment position (point)

  - Oputputs

`gl_FragColor`	fragment color
`gl_FragData[n]`	fragment color for color attachment n

GLSL

- Built-In Functions
  - Angle and Trigonometry Functions
    - `T radians(T degrees)`
    - `T degrees(T radians)`
    - `T sin(T angle)`
    - `T cos(T angle)`
    - `T tan(T angle)`
    - …

GLSL

- Built-In Functions
  - Exponential Functions
    - `T pow(T x, T y)`
    - `T exp(T x)`
    - `T log(T x)`
    - `T sqrt(T x)`
    - …

GLSL

- Built-In Functions
  - Common Functions
    - `T abs(T x)`
    - `T floor(T x), T ceil(T x)`
    - `T min(T x, T y), T max(T x, T y)`
    - `T clamp(T x, T minVal, T maxVal)`
    - `T mix(T x, T y, T a)`
    - …

GLSL

- Built-In Functions
  - Geometric Functions
    - `T length(T x)`
    - `float distance(T p0, T p1)`
    - `float dot(T x, T y)`
    - `vec3 cross(vec3 x, vec3 y)`
    - `T normalize(T x)`
    - `T reflect(T l, T n)`
    - …

Three.js

- Built-in uniforms and attributes
  - Vertex Shader
    - `uniform mat4 modelMatrix;`
    - `uniform mat4 modelViewMatrix;`
    - `uniform mat4 projectionMatrix;`
    - `uniform mat4 viewMatrix;`
    - `uniform mat3 normalMatrix;`
    - `uniform vec3 cameraPosition;`
    - `attribute vec3 position;`
    - `attribute vec3 normal;`
    - …

http://threejs.org/docs/api/renderers/webgl/WebGLProgram.html

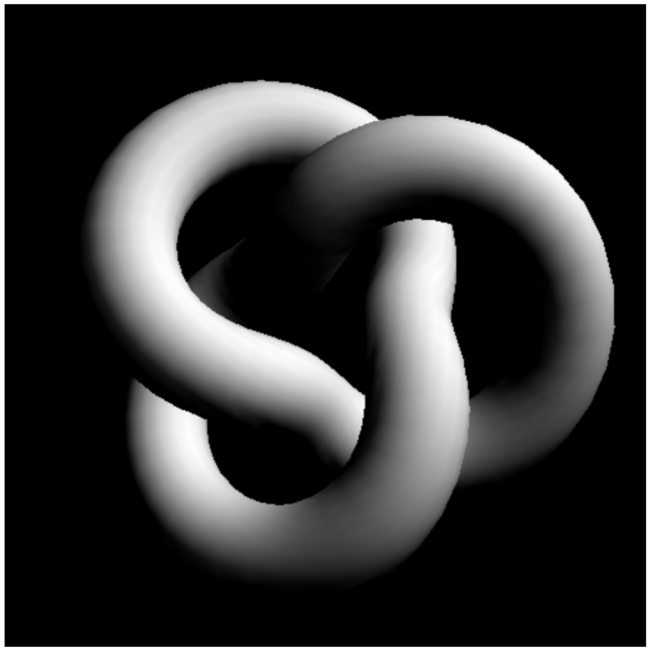
Three.js

- Built-in uniforms and attributes
  - Fragment Shader
    - `uniform mat4 viewMatrix;`
    - `uniform vec3 cameraPosition;`

http://threejs.org/docs/api/renderers/webgl/WebGLProgram.html

Torus Knot

- Download files named as main01.js and w06_ex01.html
- Open w06_ex01.html with your web browser



Torus Knot with Three.js

- THREE.TorusKnotGeometry

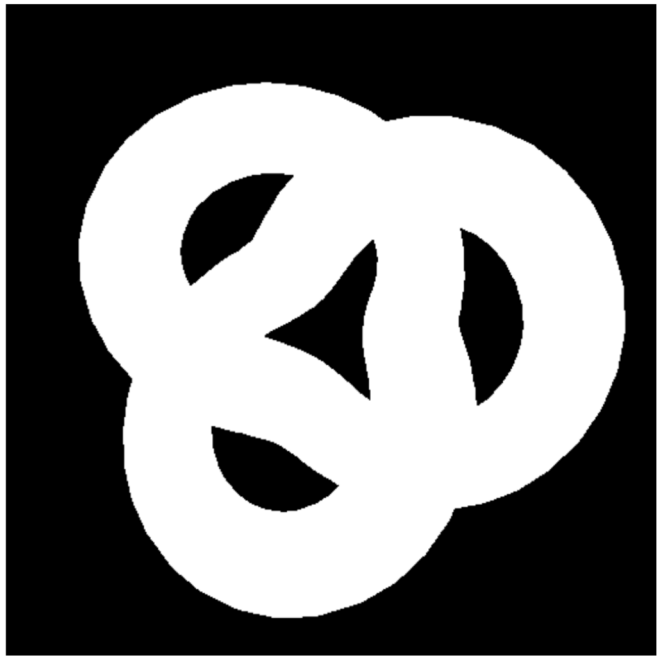
```
var geometry = new THREE.TorusKnotGeometry(1, 0.3, 100,20);
var material = new THREE.MeshLambertMaterial();

var torus_knot = new THREE.Mesh( geometry, material );
scene.add( torus_knot );
```

http://threejs.org/docs/#Reference/Extras.Geometries/TorusKnotGeometry

Torus Knot with Shaders

- Download files named as main02.js and w06_ex02.html
- Open w06_ex02.html with your web browser



Simple Shaders

- Describes shaders with <script>
  - Vertex Shader

```
<script type="x-shader/x-vertex" id="shader.vert">
void main()
{
    gl_Position = projectionMatrix * modelViewMatrix *
vec4( position, 1.0 );
}
</script>
```

Simple Shaders

- Describes shaders with <script>
  - Fragment Shader

```
<script type="x-shader/x-fragment" id="shader.frag">
void main(
{
    gl_FragColor = vec4( 1.0, 1.0, 1.0, 1.0 );
}
</script>
```

Simpler Shaders

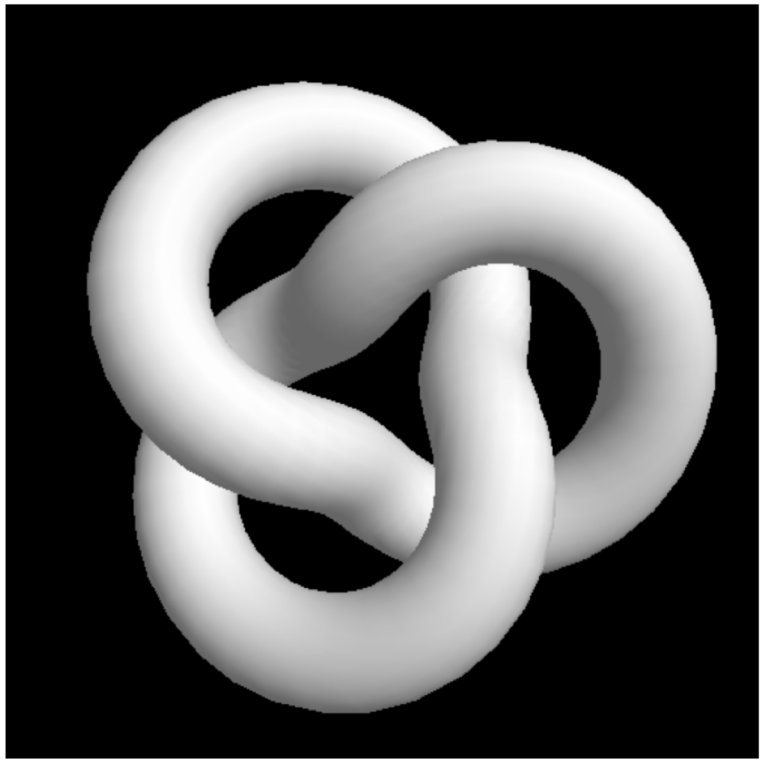
- Uses THREE.ShaderMaterial instead of THREE.MeshLambertMaterial

```
var material = new THREE.ShaderMaterial({
    vertexColors: THREE.VertexColors,
    vertexShader: document.getElementById('shader.vert').text,
    fragmentShader: document.getElementById('shader.frag').text
});
```

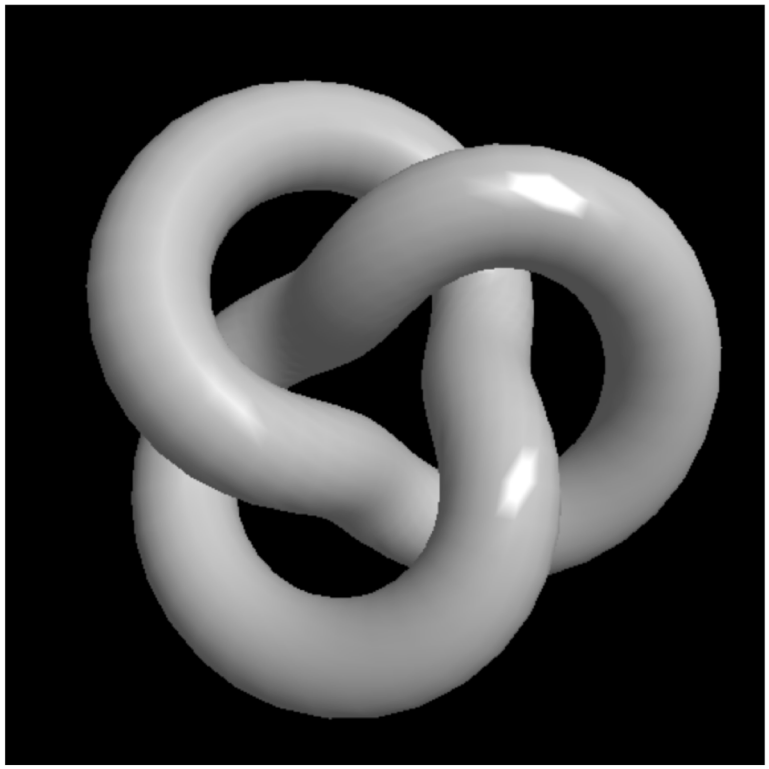
http://threejs.org/docs/#Reference/Materials/ShaderMaterial

Task 1

- Implement Gouraud shading for the rotating torus knot geometry



Lambertian reflection model



Phong reflection model

Hint

- Vertex Shader

```
<script type="x-shader/x-vertex" id="gouraud.vert">
     varying vec3 point_color;
     varying vec4 point_position;
     varying vec3 normal_vector;
     uniform vec3 light_position;

     // LambertianReflection function here

     void main()
     {
          point_position = modelViewMatrix * vec4( position, 1.0 );
          normal_vector = normalMatrix * normal;

          vec3 C = color;
          vec3 L = normalize( light_position - point_position.xyz );
          vec3 N = normalize( normal_vector );
          point_color = LambertianReflection( C, L, N );
          gl_Position = projectionMatrix * point_position;
     }
</script>
```

Hint

- Lambertian Reflection

```
vec3 LambertianReflection( vec3 C, vec3 L, vec3 N )
{
    float ka = 0.4;
    float kd = 0.6;

    float dd = max( dot( N, L ), 0.0 );
    float Ia = ka;
    float Id = kd * dd;
    return C * ( Ia + Id );
}
```

Hint

- Phong Reflection

```
vec3 PhongReflection( vec3 C, vec3 L, vec3 N )
{
    float ka = 0.3;
    float kd = 0.5;
    float ks = 0.8;
    float n = 50.0;

    vec3 R = reflect( -L, N );
    float dd = max( dot( N, L ), 0.0 );
    float ds = pow( max( dot( R, V ), 0.0 ), n );
    if ( dd <= 0.0 ) { ds = 0.0; }

    float Ia = ka;
    float Id = kd * dd;
    float Is = ks * ds;
    return C * ( Ia + Id + Is );
}
```

Hint

- Fragment Shader

```
<script type="x-shader/x-fragment" id="gouraud.frag">
     varying vec3 point_color;

     void main()
     {
          gl_FragColor = vec4( point_color, 1.0 );
     }
</script>
```

Hint

- ShaderMaterial

```
var material = new THREE.ShaderMaterial({
    vertexColors: THREE.VertexColors,
    vertexShader: document.getElementById('gouraud.vert').text,
    fragmentShader: document.getElementById('gouraud.frag').text,
    uniforms: {
        light_position: { type: 'v3', value: light.position }
    }
});
```

Uniform types:

- i int int
- f float float
- v2 THREE.Vector2 vec2
- v3 THREE.Vector3 vec3
- v4 THREE.Vector4 vec4
- c THREE.Color vec3
- …

https://github.com/mrdoob/three.js/wiki/Uniforms-types

Task 2

- Implement Phong shading for the rotating torus knot geometry
- Compare the rendering result with Gouraud shading

Advanced Tasks

- Implement shaders based on the following reflection models
  - Blinn-Phong reflection (Task 3)
  - Cook-Torrance reflection (Task 4)
  - Toon reflection (Task 5)

Polling

- Take the poll
  - Student ID Number
  - Name
  - URL to Task 1
  - URL to Task 2
  - URL to Task 3 (advanced)
  - URL to Task 4 (advanced)
  - URL to Task 5 (advanced)