Lighting (1)

GPU Programming

2022학년도
2학기

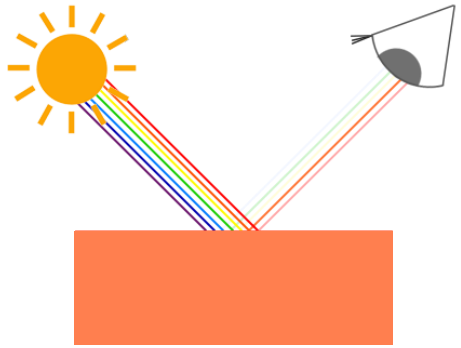
ⓒ nvidia

Colors

Colors

- 현실의 무한한 색을 그래픽스로 표현할 때에는 제한된 디지털 값(24비트 등)에 매핑해야 함

- 디지털 세계에서 색은 RGB (red, green, blue) 요소로 표현
  - 오른쪽 코랄색에 대한 컬러 벡터 `glm::vec3 coral(1.0f, 0.5f, 0.31f);` CPU

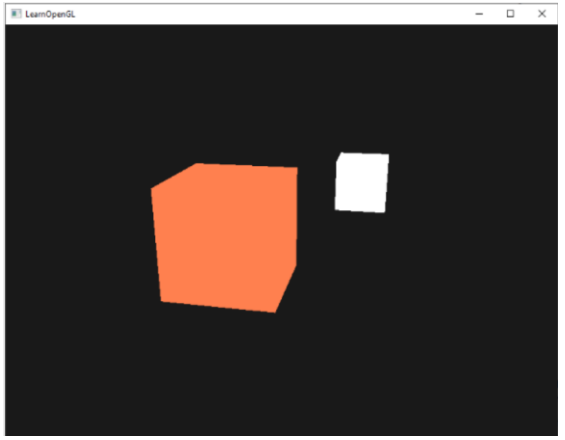


- 실생활에서 보이는 색상
  - 실제로 사물이 어떠한 색상을 가지고 있는 것은 아님
  - 빛에서 색을 표현하는 파장 중에, 사물에 흡수되지 않고 반사된 색이 눈에 들어오는 것
  - 따라서, 빛의 색상이 바뀐다면(예: 붉은 조명) 눈에 보이는 사물의 색상도 이에 따라 붉게 변함

- 빛의 색상과 사물의 색상(=RGB별 반사 정도)을 함께 적용한 예

CPU
```
glm::vec3 lightColor(0.33f, 0.42f, 0.18f);
glm::vec3 toyColor(1.0f, 0.5f, 0.31f);
glm::vec3 result = lightColor * toyColor; // = (0.33f, 0.21f, 0.06f);
```

A Lighting Scene



- 현실의 조명(lighting)을 간단히 시뮬레이션하기 위한 예제 작성
  - 빛을 내는 광원(light source)에 대한 물체(object) 정의
  - 빛을 받아 색상을 내는 또 다른 물체 정의
  - 이 물체들 간의 빛의 이동을 계산 (local illumination)

- 두 물체를 그리기 위해 기존 컨테이너 외에 램프 박스를 추가
  - 컨테이너의 수정이 램프 박스에 영향을 끼치지 않도록, 새로 VAO를 생성하고 바인딩
  - 두 물체는 기존 컨테이너 박스의 vertex data를 공유하므로 VBO는 바인딩만 수행

CPU

```
unsigned int lightVAO;
glGenVertexArrays(1, &lightVAO);
glBindVertexArray(lightVAO);
// we only need to bind to the VBO, the container's VBO's data already contains the data.
glBindBuffer(GL_ARRAY_BUFFER, VBO);
// set the vertex attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
```

A Lighting Scene

- 컨테이너의 설정
  - 색상을 계산하는 fragment shader (FS) 작성

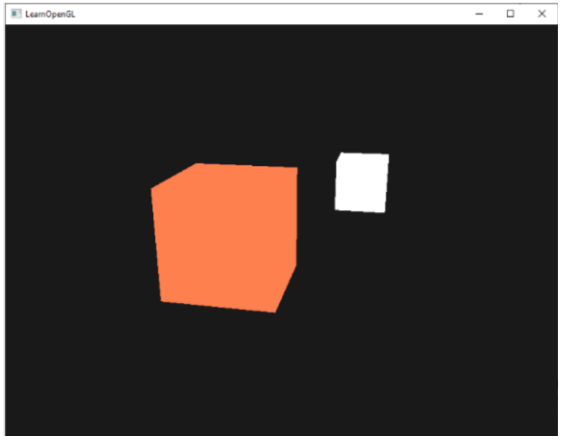
FS

```
#version 330 core
out vec4 FragColor;

uniform vec3 objectColor;
uniform vec3 lightColor;

void main()
{
    FragColor = vec4(lightColor * objectColor, 1.0);
}
```

  - Cpp 코드에서 shader 내 uniform 변수로 전달되는 속성 설정

CPU

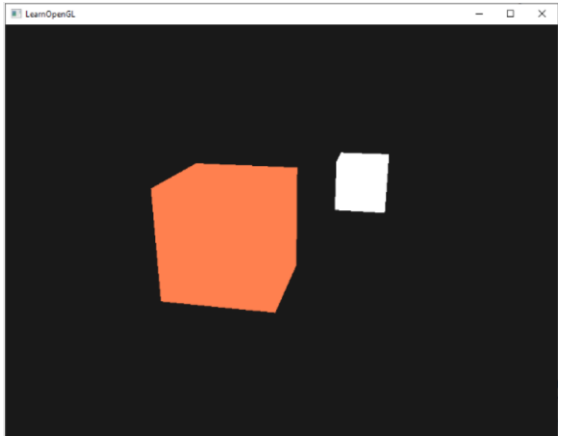
```
// don't forget to use the corresponding shader program first (to set the uniform)
lightingShader.use();
lightingShader.setVec3("objectColor", 1.0f, 0.5f, 0.31f);
lightingShader.setVec3("lightColor",  1.0f, 1.0f, 1.0f);
```



A Lighting Scene

- 램프 박스의 설정
  - 램프의 색상(흰색)을 결정하는 FS 작성

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0); // set all 4 vector values to 1.0
}
```
FS



  - cpp 코드에서 램프 박스의 위치 설정 및 변환

```
glm::vec3 lightPos(1.2f, 1.0f, 2.0f);
model = glm::mat4(1.0f);
model = glm::translate(model, lightPos);
model = glm::scale(model, glm::vec3(0.2f));
```
CPU

  - 램프 박스 그리기

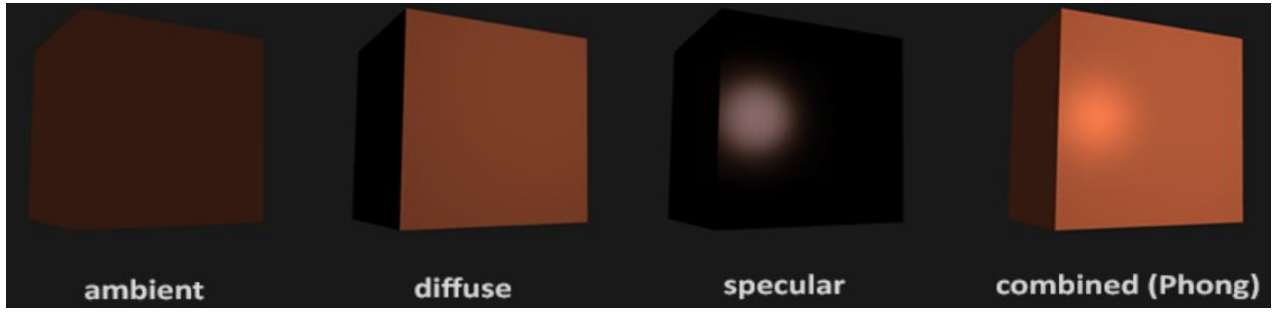
```
lightCubeShader.use();
// set the model, view and projection matrix uniforms
[...]
// draw the light cube object
glBindVertexArray(lightCubeVAO);
glDrawArrays(GL_TRIANGLES, 0, 36);
```
CPU

ⓒ nvidia

Basic Lighting

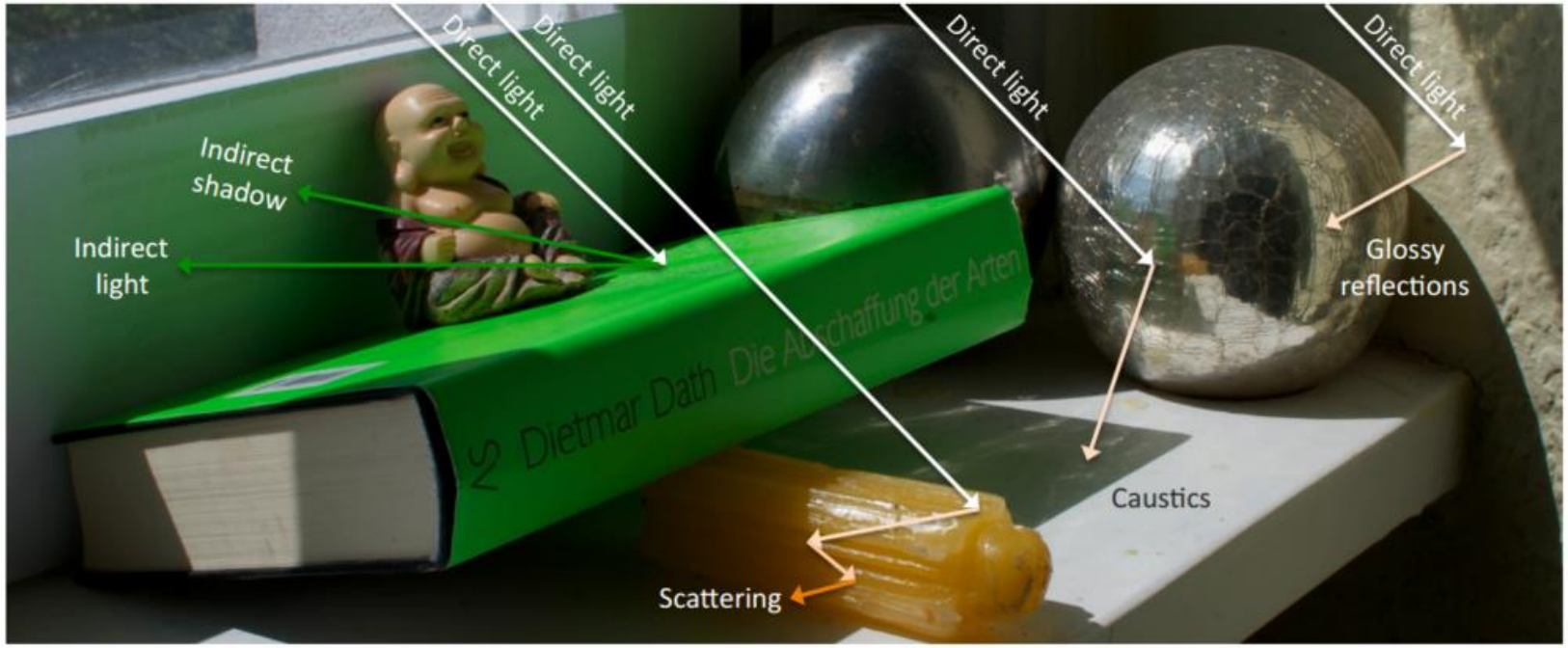
Phong Lighting Model

- Bui Tuong Phong이 1975년에 발표한, 매우 복잡한 실세계의 조명을 3가지 요소로 근사화한 모델
- Ambient lighting
  - 특정한 방향이 없이 주변을 덮고 있는 빛
  - 아주 어두운 환경에서도, 멀리 떨어진 어딘가로부터의 조명(예: 달) 덕분에 무엇인가는 보임
  - 이를 시뮬레이션하기 위해, 물체에 어떠한 색상을 주는 ambient 조명 상수 사용
- Diffuse lighting
  - 일정한 방향으로 광원으로부터 들어와 물체의 표면에서 여러 방향으로 분산되는, 즉 난반사되는 빛
  - 물체의 많은 부분이 광원을 마주보고 있을수록 더 밝아짐
- Specular lighting
  - 일정한 방향으로 광원으로부터 들어와 물체의 표면에서 한 방향으로 정반사되는 빛
  - 빛의 하이라이트를 시뮬레이션함 (이 하이라이트 색상은 주로 광원의 색상에 좌우됨)



Ambient Lighting

- 빛은 일반적으로 하나의 광원으로부터 오지 않고, 우리 주변에 산재한 수많은 광원에서 옴
- 이러한 복잡한 조명을 시뮬레이션 하는 방법을 global illumination(GI)이라고 함
  - GI를 처리하기 위해 ray tracing, radiosity 등 여러 방식이 존재



GISTAR.pdf (jankautz.com)

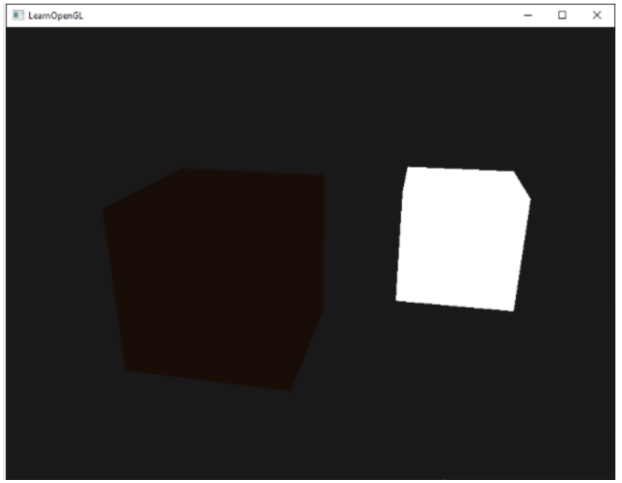
- Ambient lighting은 이를 아주 단순화하여, 항상 퍼지는 빛의 값에 대해 상수값을 부여하는 방식

Ambient Lighting

- Ambient lighting을 적용한 FS 코드

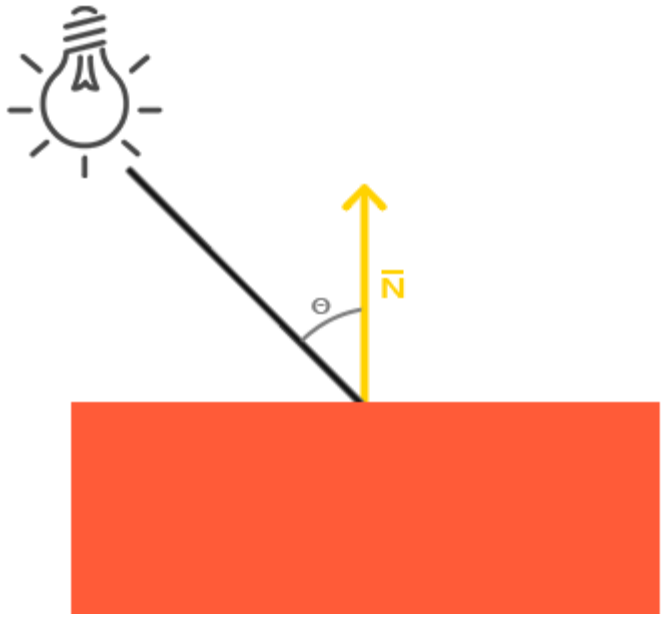
```
void main()                                                    FS
{
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;

    vec3 result = ambient * objectColor;
    FragColor = vec4(result, 1.0);
}
```

- 실행 결과



Diffuse Lighting

- 물체에 중요한 시각적 효과를 주는 요인
  - 빛을 받는 방향이 정면에 가까울수록 더 많은 빛을 받아 더 밝아짐



- Diffuse lighting 계산을 위한 요소
  - 노멀 벡터 (normal vector, 법선 벡터): vertex 및 fragment의 면에 수직인 벡터
  - 방향을 가진 ray (광선): fragment에서 광원의 위치로 향하는 방향 벡터

- Diffuse lighting 계산
  - 노멀 벡터와 ray의 방향 벡터를 모두 정규화한 후 이 둘을 내적
  - 두 벡터가 모두 정규화되었으므로 (길이가 1), 내적의 결과는 cosθ가 됨 (단, 결과값이 음수면 0 이상으로 clipping)

$$v \cdot w = |v| \cdot |w| \cdot \cos(\theta) \overset{|v|=|w|=1}{=} \cos(\theta)$$

https://vis.uni-jena.de/Lecture/ComputerGraphics/Lec8_Lighting_I.pdf

Diffuse Lighting

- 노멀 벡터는 외적을 이용해 구하거나, 미리 계산된 값을 입력
  - 오른쪽은 큐브의 vertex data에 위치와 함께 노멀 값들을 추가한 배열
  - 이를 vertex array에 반영

- 노멀 벡터를 shader에 반영하는 방법
  - VS에 normal 관련 입력 변수 추가
  - vertex 속성으로 normal 추가

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
...
```
VS

```
// normal attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
```
CPU

  - VS에서 FS로 normal값을 그대로 넘겨 줌 (fragment별로 보간된 값이 입력)

```
out vec3 Normal;

void main()
{
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    Normal = aNormal;
}
```
VS

```
in vec3 Normal;
```
FS

```
float vertices[] = {
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

    -0.5f, -0.5f,  0.5f,  0.0f,  0.0f, 1.0f,
     0.5f, -0.5f,  0.5f,  0.0f,  0.0f, 1.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  0.0f, 1.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  0.0f, 1.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  0.0f, 1.0f,
    -0.5f, -0.5f,  0.5f,  0.0f,  0.0f, 1.0f,

    -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
    -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,

     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
     0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,

    -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
     0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
    -0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,

    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f
};
```
CPU

Code Viewer. Source code: lighting/basic_lighting_vertex_data (learnopengl.com)

Diffuse Lighting

- 셰이더에서의 diffuse color 계산
  - FS로 광원의 위치벡터를 넘김

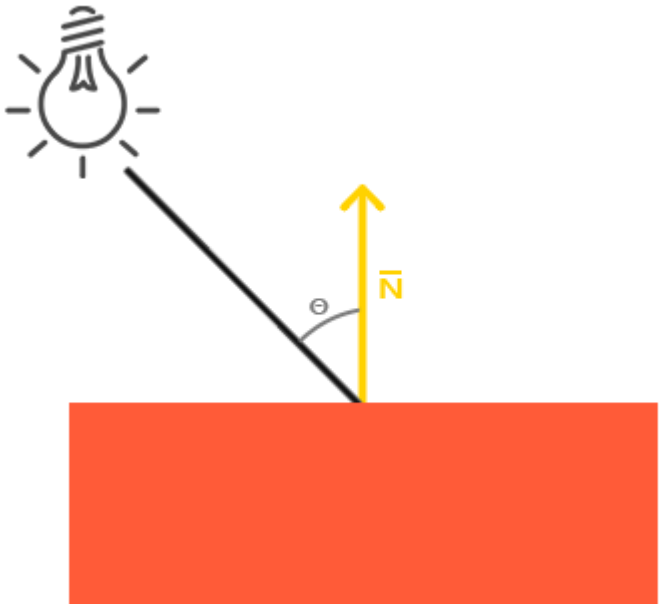
```
uniform vec3 lightPos;   FS
```

```
lightingShader.setVec3("lightPos", lightPos);   CPU
```

  - VS에서 FS로 FragPos (world space 상의 fragment 위치)를 넘겨 줌. FS에서는 이 값이 fragment별로 보간.

```
out vec3 FragPos;                                                    VS
out vec3 Normal;

void main()
{
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    FragPos = vec3(model * vec4(aPos, 1.0));
    Normal = aNormal;
}
```

```
in vec3 FragPos;   FS
```



  - FS에서 Diffuse lighting 계산

```
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;
vec3 result = (ambient + diffuse) * objectColor;
FragColor = vec4(result, 1.0);
```

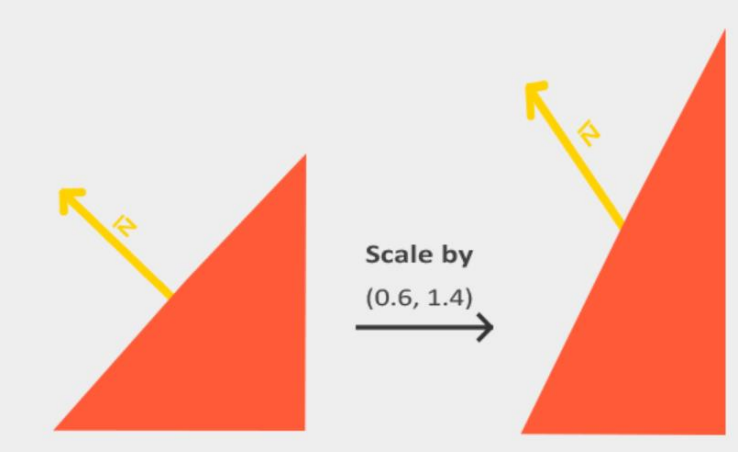
FS

$$v \cdot w = |v| \cdot |w| \cdot \cos(\theta) \overset{|v|=|w|=1}{=} \cos(\theta)$$

Diffuse Lighting - One Last Thing

- 노멀 벡터는 기존 모델 행렬의 translation(이동)에 영향을 받지 않아야 함
  - 노멀 벡터는 방향만을 가지고 위치를 표현하지 않기 때문
  - 변환 행렬의 동차 좌표에서 w값이 0이 됨.
    즉, Translation에 따라 값이 변하지 않으므로, 기존 모델 행렬의 좌 상단 3x3 성분만 가져야 함

- 노멀 벡터에는 기존 모델 행렬의 scaling(확대/축소)을 그대로 적용하면 안 됨
  - 노멀 벡터와 접선 벡터(tangent vector)가 더 이상 수직이 아니게 될 수 있음



- 해법
  - 노멀 벡터를 변환하기 위한
    3x3 크기의 별도 노멀 행렬(normal matrix) 사용
  - VS에서의 노멀 행렬 적용 예

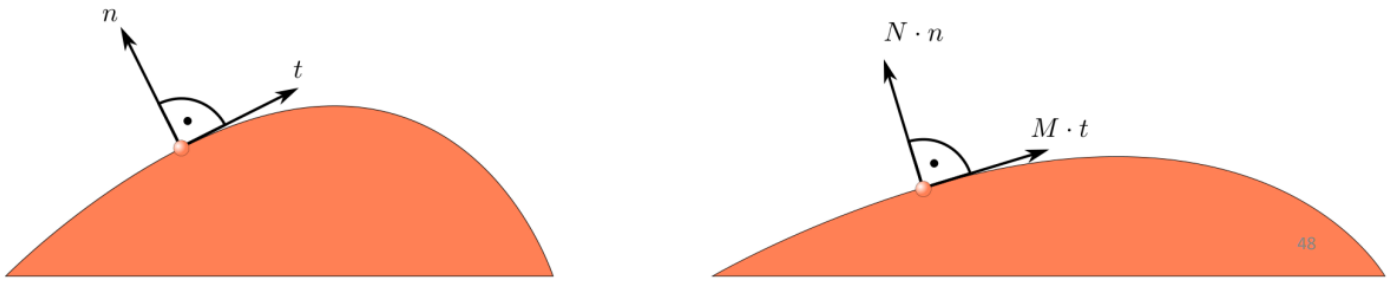
```
Normal = mat3(transpose(inverse(model))) * aNormal;
```
VS

  - 단, 위와 같이 VS에서 직접 역행렬을 계산하는 것 보다는 (고비용 연산),
    CPU에서 이 노멀 행렬을 계산하여 uniform을 통해 셰이더로 전달하는 것이 바람직

Diffuse Lighting - One Last Thing

- 노멀 행렬의 식 유도

```
Normal = mat3(transpose(inverse(model))) * aNormal;
```



$$\langle t, n \rangle = \langle M \cdot t, N \cdot n \rangle = 0$$

$$0 = \langle M \cdot t, N \cdot n \rangle$$

$$= (M \cdot t)^T \cdot (N \cdot n)$$

$$= t^T \cdot M^T \cdot N \cdot n$$

$$= t^T (M^T N) n$$

$$M^T N = Id$$

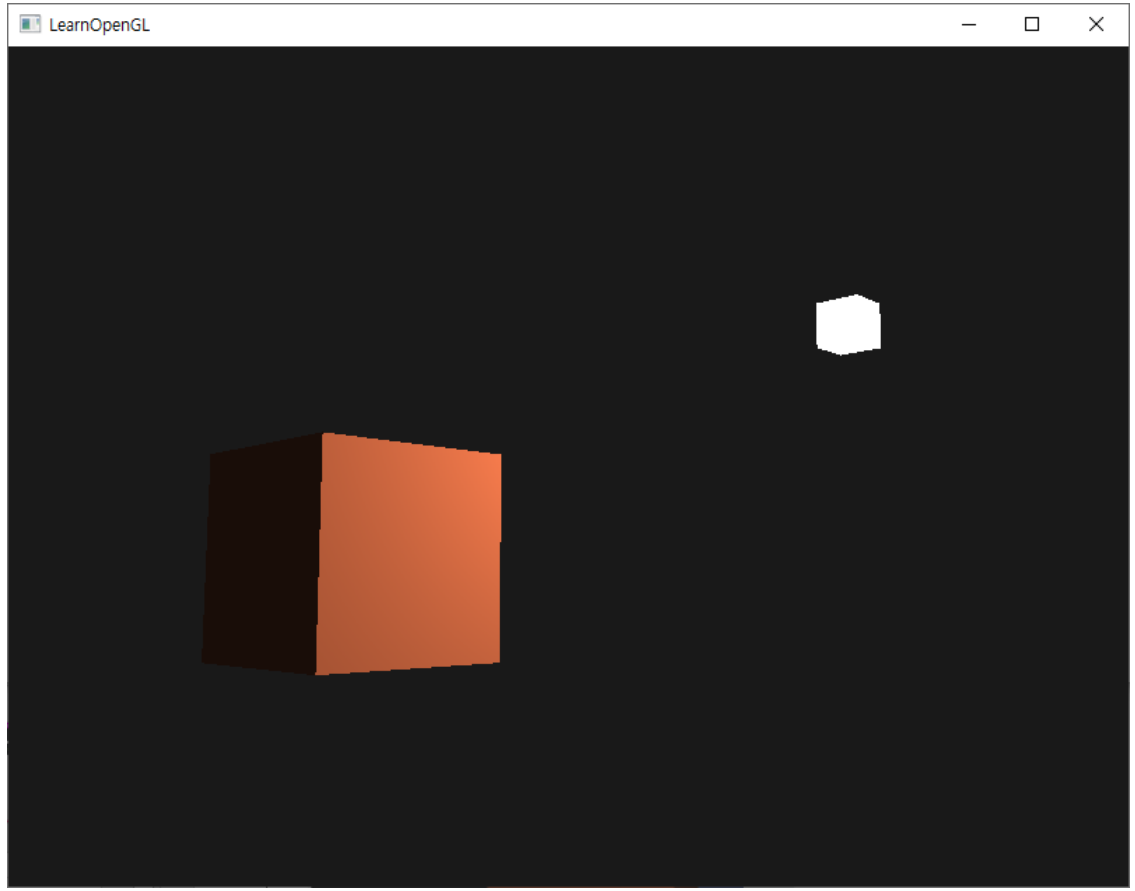
$$N = (M^T)^{-1}$$

$$= (M^{-1})^T$$

https://vis.uni-jena.de/Lecture/ComputerGraphics/Lec8_Lighting_I.pdf

Diffuse Lighting

- 실행 결과

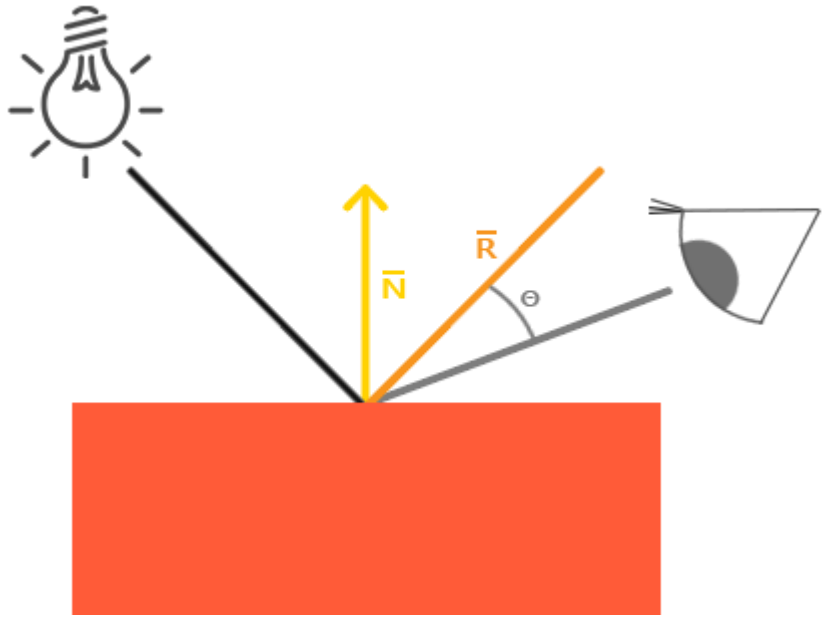


Specular Lighting

- 빛의 하이라이트를 주는 요인
  - 물체를 통해 빛이 거울 반사되는 방향으로 물체를 바라보면, 빛의 하이라이트를 볼 수 있음
  - 따라서 보는 시점에 따라 결과가 달라짐

- Specular lighting 계산을 위한 추가 요소
  - Fragment에서 Viewer로 향하는 방향 (view direction): Viewer의 world space 위치와 fragment의 위치로 계산 가능

- Specular lighting 계산
  - 정규화된 노멀 벡터와 ray의 방향 벡터를 이용하여 반사 벡터 생성
  - 반사 벡터와 view 방향 벡터를 내적한 후, 이를 shininess 계수만큼 제곱



Specular Lighting

- 셰이더에서의 구현
  - FS에 viewPos 변수를 선언하고, 카메라의 위치를 넘겨 줌

```
uniform vec3 viewPos; FS
lightingShader.setVec3("viewPos", camera.Position); CPU
```

  - Specular highlight의 강도 설정

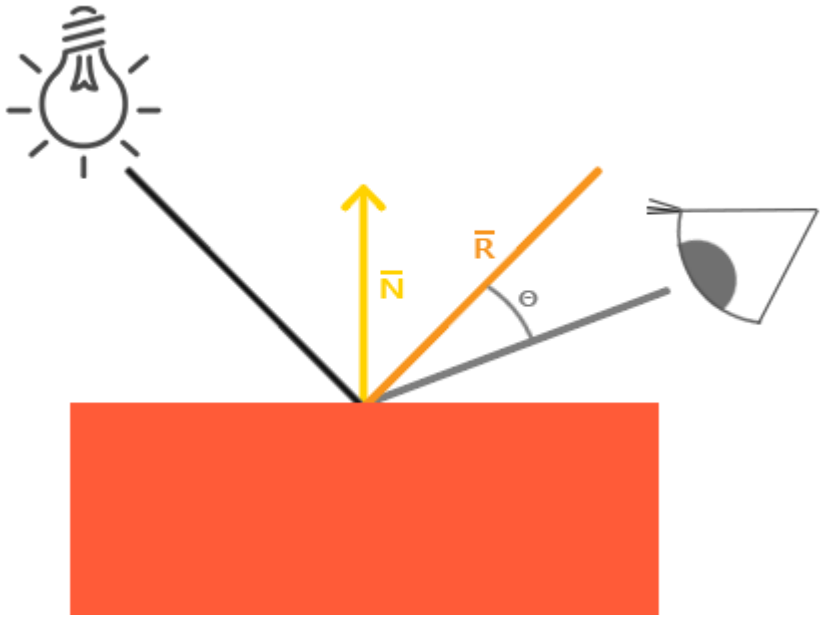
```
float specularStrength = 0.5; FS
```

  - View 방향 벡터와 반사 벡터 계산

```
vec3 viewDir = normalize(viewPos - FragPos); FS
vec3 reflectDir = reflect(-lightDir, norm);
```

  - 위 두 벡터를 내적한 후 이를 0 이상으로 보정하고, shininess 계수만큼 제곱.
    이후 이 값에 광원의 색상과 specular 강도를 곱해 최종 specular 색상 결정

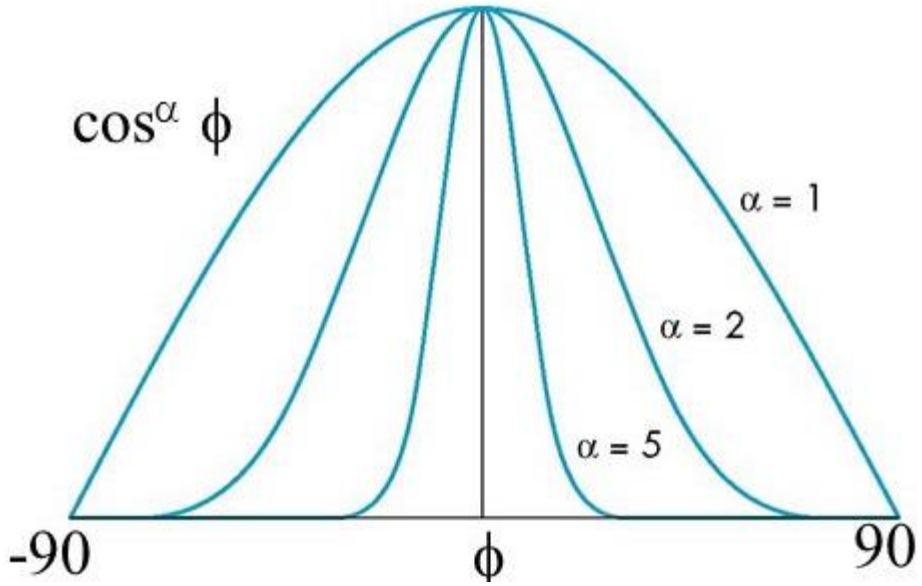
```
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32); FS
vec3 specular = specularStrength * spec * lightColor;
```



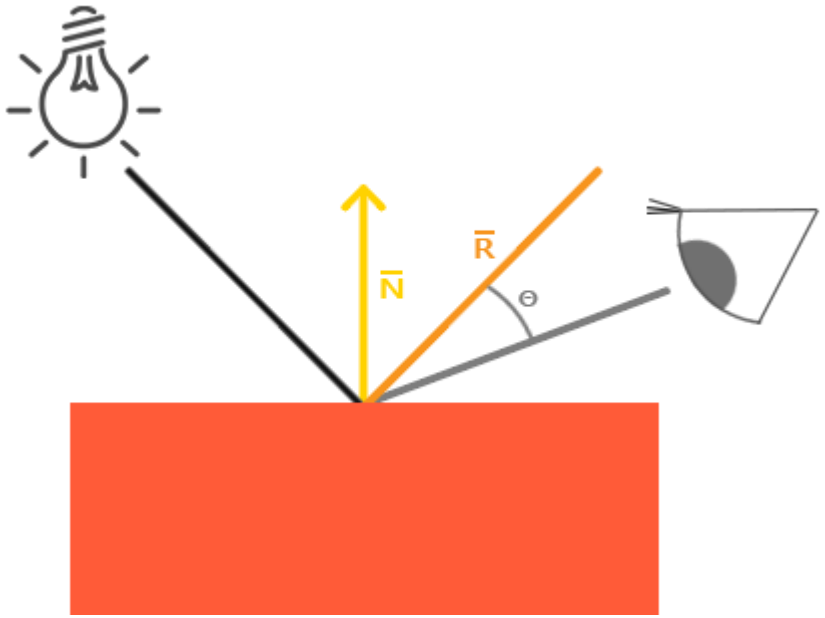
Specular Lighting

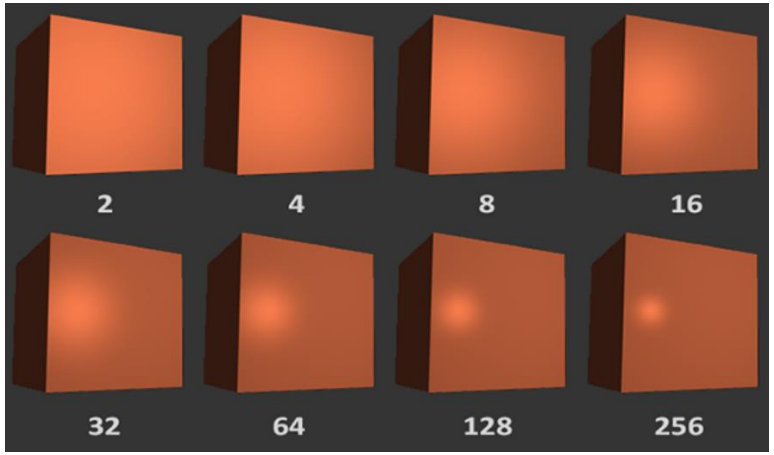
- Shininess coefficient
  - cos 값에 이 계수만큼 제곱을 해 주면, 하이라이트의 크기 지정 가능
  - 5~10이면 플라스틱, 100~200이면 메탈과 같은 느낌

```
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
vec3 specular = specularStrength * spec * lightColor;
```
FS



AngelCG18.ppt (unm.edu)

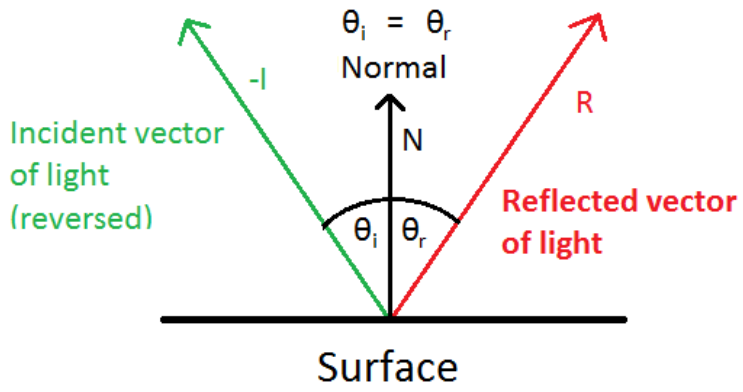


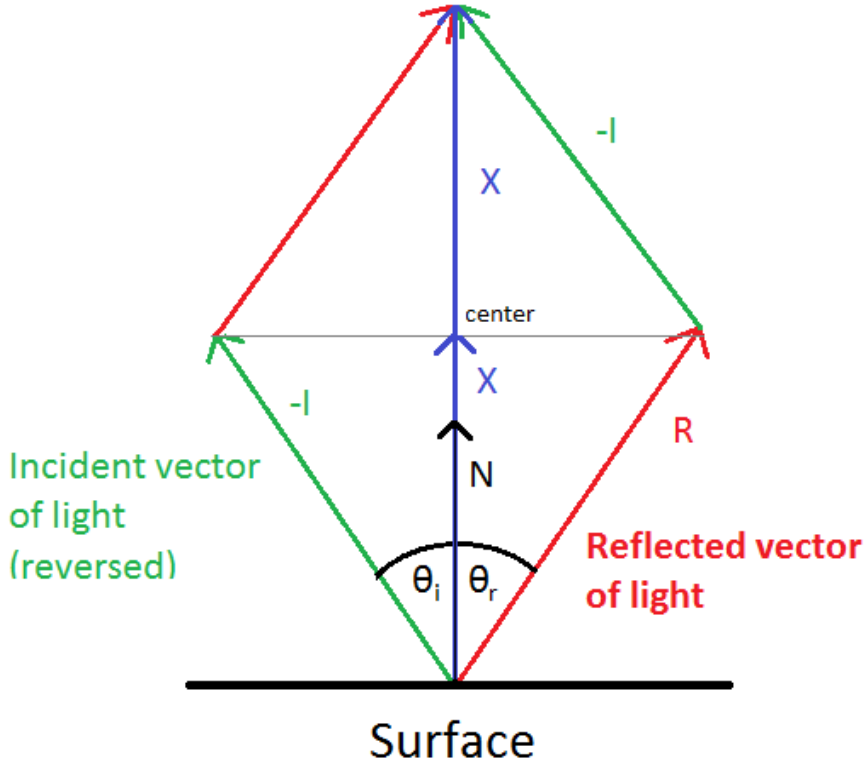


Shininess 값에 따른
하이라이트의 크기 변화

Reflection Equation

- glsl의 reflect(I, N) 함수가 반환하는 값: I - 2.0 * dot(N, I) * N
- 위 반사 식 유도



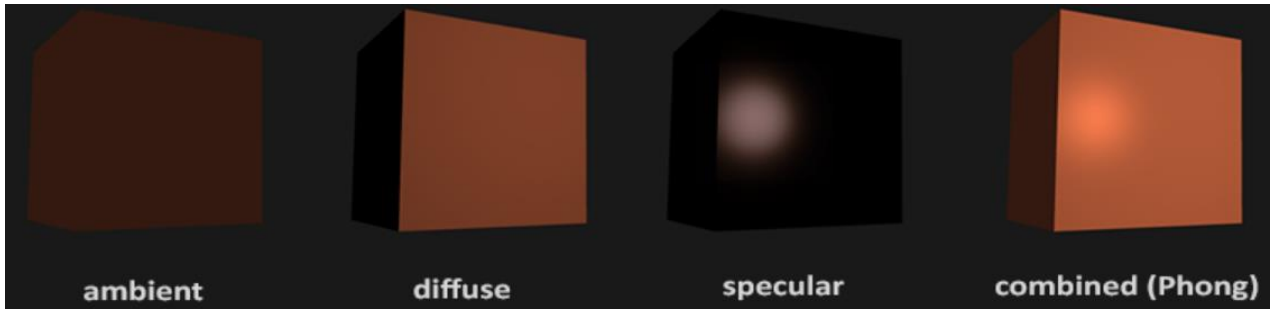


$$R = 2 * (N.(-I)) * N + I =$$
$$= I - 2 * (I.N) * N$$

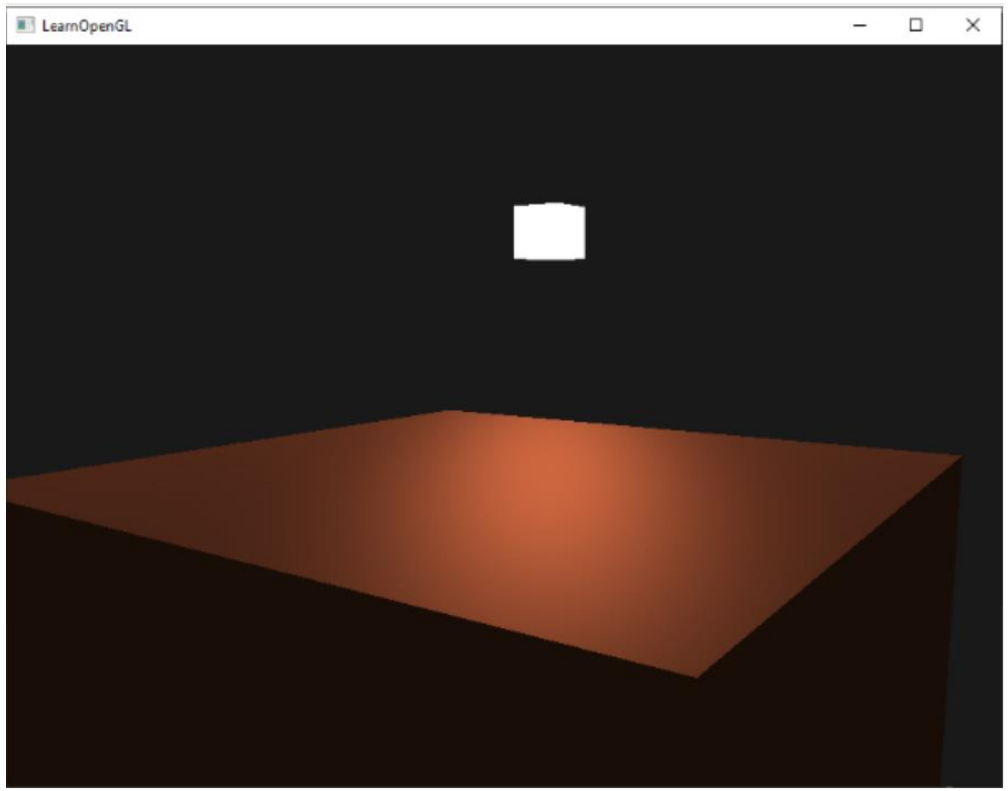
021.) Specular Highlight - OpenGL 4 - Tutorials - Megabyte Softworks (mbsoftworks.sk)

Putting It All Together

- Ambient + diffuse + specular를 합한 결과




```
vec3 result = (ambient + diffuse + specular) * objectColor;
FragColor = vec4(result, 1.0);
```
FS



- Word space가 아닌 view space에서도 lighting 계산 가능
  - Viewer의 위치가 항상 (0,0,0)이므로 따로 계산할 필요가 없음
  - 모든 관련 벡터가 view space로 변환되어야 함

- 광원과 fragment간의 거리에 따른 감쇠(attenuation) 효과는 고려하지 않았음
  - 이는 Chap 16. Light Casters에서 다룸

Shading Models

- Lighting model 적용 방법에 따른 세 가지 shading 분류
  - Flat shading, Gouraud shading, Phong shading

- Flat Shading
  - 면(face) 또는 폴리곤(polygon) 단위로 색상을 결정
  - GPU가 없고 연산 리소스가 부족한 시절 주로 사용
  - 최근에는 레트로 또는 카툰 스타일 게임에서 간혹 사용

Virtua Fighter (1993) - First 3D fighting with zoom and rotated viewpoint - YouTube

Saving the People from the Alien Invasion! - Mugsters Gameplay - YouTube

WHEN TO APPLY LIGHTING MODEL?

per polygon	per vertex	per pixel
"flat shading"	"Gouraud shading"	"per pixel lighting" "Phong shading"

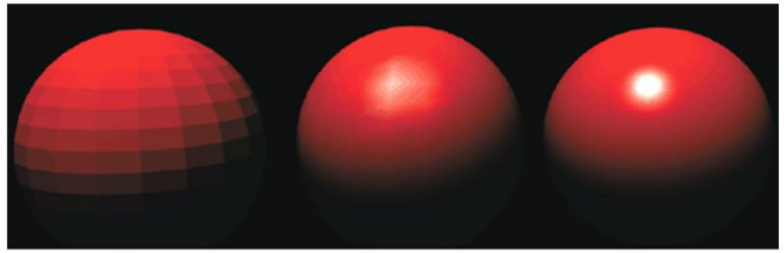


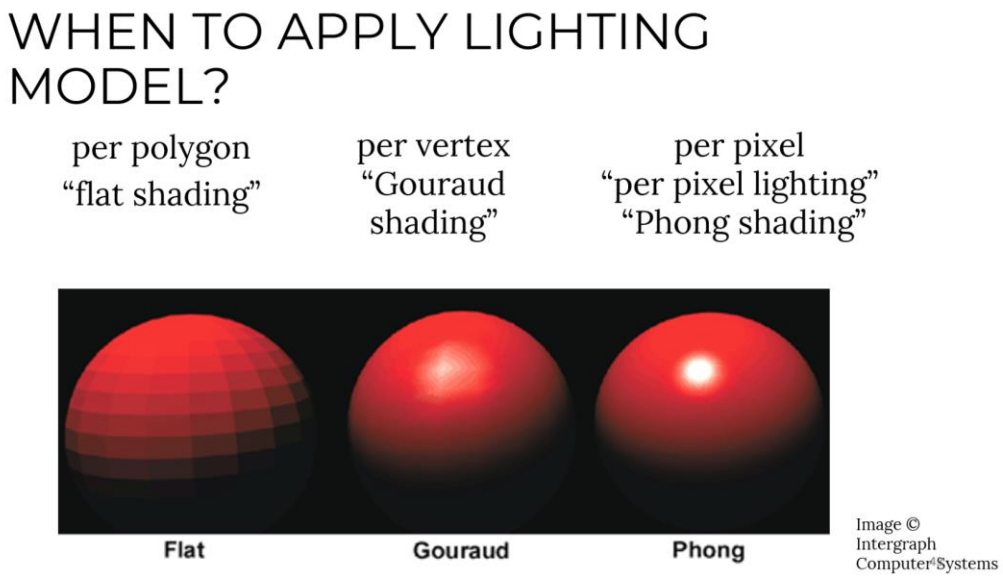
Image © Intergraph Computer Systems

lighting.pptx (live.com)

Shading Models

- Gouraud Shading
  - Phong lighting model을 VS에서 구현할 경우, fragment에서는 lighting이 선형 보간(linear interpolation)됨
  - 보간 기법을 셰이딩에 처음 적용한 Henri Gouraud의 이름을 따서 명명
  - 연산량은 감소할 수 있지만, 다소 부자연스러운 lighting 결과 발생 가능 (특히 highlight)

- Phong shading
  - Phong lighting model을 FS단에서 pixel 단위로 구현
  - 지금까지 구현한 방법



lighting.pptx (live.com)

- Unity의 제안 Unity - Manual: Optimizations (unity3d.com)
  - Adjust pixel light count in **Quality** settings. Essentially only the directional light should be per pixel, everything else - per vertex. Certainly this depends on the game.

ⓒ nvidia

Materials

Materials

- 현실에서 사물은 빛에 각자 다르게 반응
  - 철로 된 사물은 반짝이나, 나무 컨테이너는 빛에 반응하지 않음
  - Specular highlight도 사물마다 많이 퍼지거나 좁게 모아질 수 있음

- 이를 시뮬레이션 하기 위해, 각 객체별로 material 속성을 정의할 필요가 있음
  - Phong lighting의 각 컴포넌트들에 대한 컬러 벡터
  - Specular lighting에 필요한 shininess값

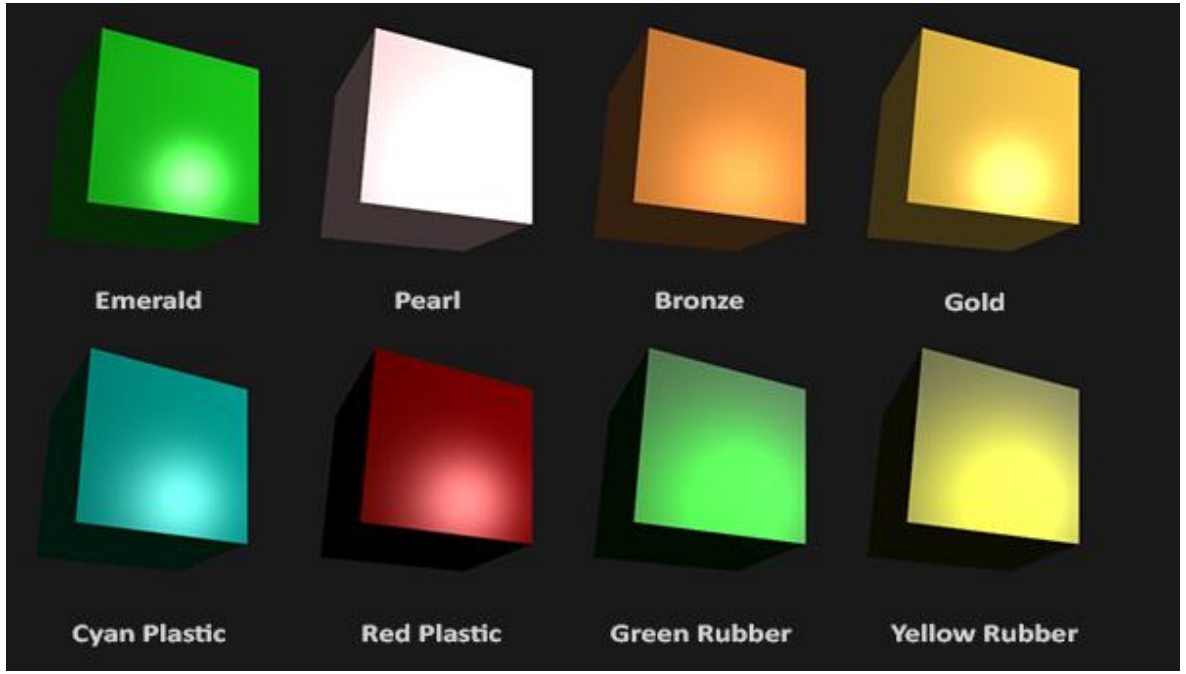
FS
```
#version 330 core
struct Material {
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    float shininess;
};

uniform Material material;
```

Material Table

Name	Ambient			Diffuse			Specular			Shininess
emerald	0.0215	0.1745	0.0215	0.07568	0.61424	0.07568	0.633	0.727811	0.633	0.6
jade	0.135	0.2225	0.1575	0.54	0.89	0.63	0.316228	0.316228	0.316228	0.1
obsidian	0.05375	0.05	0.06625	0.18275	0.17	0.22525	0.332741	0.328634	0.346435	0.3
pearl	0.25	0.20725	0.20725	1	0.829	0.829	0.296648	0.296648	0.296648	0.088
ruby	0.1745	0.01175	0.01175	0.61424	0.04136	0.04136	0.727811	0.626959	0.626959	0.6
turquoise	0.1	0.18725	0.1745	0.396	0.74151	0.69102	0.297254	0.30829	0.306678	0.1
brass	0.329412	0.223529	0.027451	0.780392	0.568627	0.113725	0.992157	0.941176	0.807843	0.21794872
bronze	0.2125	0.1275	0.054	0.714	0.4284	0.18144	0.393548	0.271906	0.166721	0.2
chrome	0.25	0.25	0.25	0.4	0.4	0.4	0.774597	0.774597	0.774597	0.6
copper	0.19125	0.0735	0.0225	0.7038	0.27048	0.0828	0.256777	0.137622	0.086014	0.1
gold	0.24725	0.1995	0.0745	0.75164	0.60648	0.22648	0.628281	0.555802	0.366065	0.4
silver	0.19225	0.19225	0.19225	0.50754	0.50754	0.50754	0.508273	0.508273	0.508273	0.4
black plastic	0.0	0.0	0.0	0.01	0.01	0.01	0.50	0.50	0.50	.25
cyan plastic	0.0	0.1	0.06	0.0	0.50980392	0.50980392	0.50196078	0.50196078	0.50196078	.25
green plastic	0.0	0.0	0.0	0.1	0.35	0.1	0.45	0.55	0.45	.25
red plastic	0.0	0.0	0.0	0.5	0.0	0.0	0.7	0.6	0.6	.25
white plastic	0.0	0.0	0.0	0.55	0.55	0.55	0.70	0.70	0.70	.25
yellow plastic	0.0	0.0	0.0	0.5	0.5	0.0	0.60	0.60	0.50	.25
black rubber	0.02	0.02	0.02	0.01	0.01	0.01	0.4	0.4	0.4	.078125
cyan rubber	0.0	0.05	0.05	0.4	0.5	0.5	0.04	0.7	0.7	.078125
green rubber	0.0	0.05	0.0	0.4	0.5	0.4	0.04	0.7	0.04	.078125
red rubber	0.05	0.0	0.0	0.5	0.4	0.4	0.7	0.04	0.04	.078125
white rubber	0.05	0.05	0.05	0.5	0.5	0.5	0.7	0.7	0.7	.078125
yellow rubber	0.05	0.05	0.0	0.5	0.5	0.4	0.7	0.7	0.04	.078125

OpenGL/VRML Materials (free.fr)

Material 적용 예시



Setting Materials

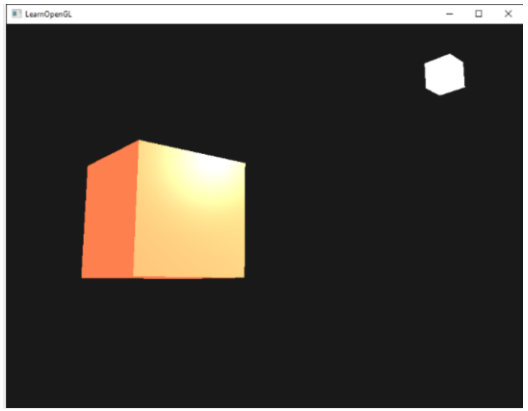
- FS에서는 lighting 계산시
  Material 구조체에 담긴 멤버에 액세스
  - 각 컴포넌트별 계산값에 곱함

- CPU쪽 코드에서는 해당 멤버의 값을 설정
  - 부드러운 갈색, 좁은 하이라이트

```
lightingShader.setVec3("material.ambient", 1.0f, 0.5f, 0.31f);
lightingShader.setVec3("material.diffuse", 1.0f, 0.5f, 0.31f);
lightingShader.setVec3("material.specular", 0.5f, 0.5f, 0.5f);
lightingShader.setFloat("material.shininess", 32.0f);
```

CPU

- 적용 결과



```
void main()
{
    // ambient
    vec3 ambient = lightColor * material.ambient;

    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = lightColor * (diff * material.diffuse);

    // specular
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);
    vec3 specular = lightColor * (spec * material.specular);

    vec3 result = ambient + diffuse + specular;
    FragColor = vec4(result, 1.0);
}
```

FS

Light Properties

- 앞선 결과는 너무 밝은 느낌
  - Ambient, diffuse, specular 색상 모두가 모든 광원으로부터 완전하게 반사되기 때문
  - Lighting 컴포넌트의 세기 벡터를 지정하여 이를 조정 가능
- Material uniform과 유사하게 Light uniform을 만들고 lighting 계산시 이를 사용

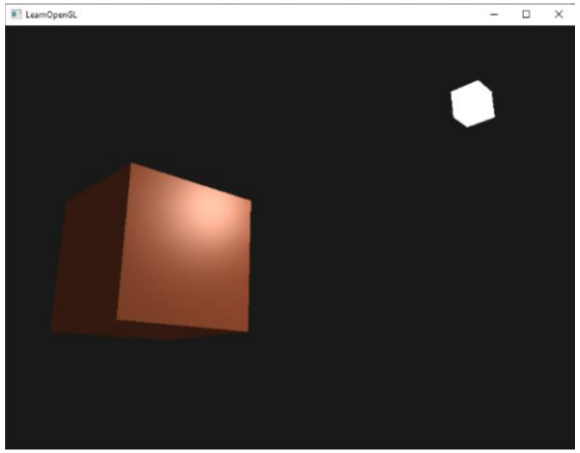
```
struct Light {                                    FS
    vec3 position;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};

uniform Light light;
```

```
vec3 ambient  = light.ambient * material.ambient;                    FS
vec3 diffuse  = light.diffuse * (diff * material.diffuse);
vec3 specular = light.specular * (spec * material.specular);
```

- CPU쪽 코드에서는 해당 멤버의 값을 설정
  - 기존 대비 Ambient는 20%, diffuse는 50%, specular는 100% 수준

```
lightingShader.setVec3("light.ambient",  0.2f, 0.2f, 0.2f);
lightingShader.setVec3("light.diffuse",  0.5f, 0.5f, 0.5f); // darken diffuse light a bit
lightingShader.setVec3("light.specular", 1.0f, 1.0f, 1.0f);                    CPU
```

- 적용 결과



Different Light Colors

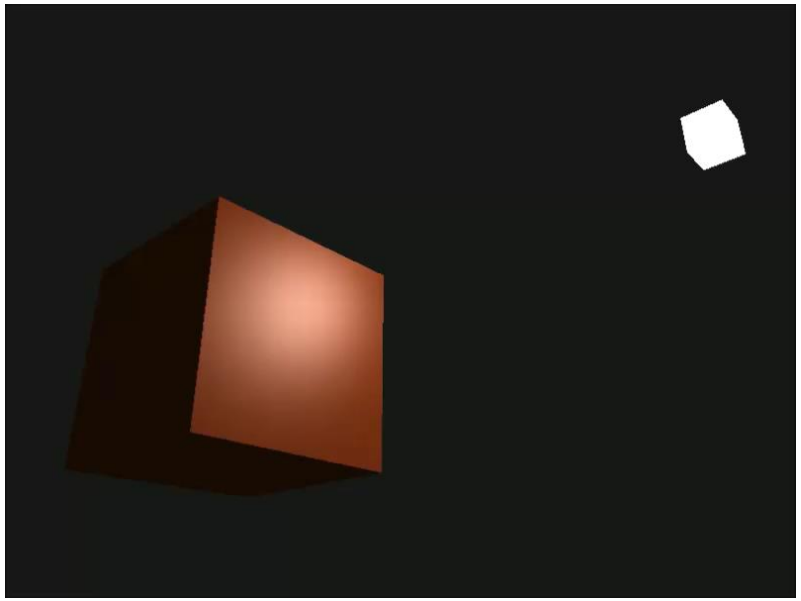
- 흰색 조명이 아닌 다른 색상의 조명도 시뮬레이션 가능

- 시간에 따라 광원의 색상이 바뀌도록 하는 예제

CPU

```
glm::vec3 lightColor;
lightColor.x = sin(glfwGetTime() * 2.0f);
lightColor.y = sin(glfwGetTime() * 0.7f);
lightColor.z = sin(glfwGetTime() * 1.3f);

glm::vec3 diffuseColor = lightColor   * glm::vec3(0.5f);
glm::vec3 ambientColor = diffuseColor * glm::vec3(0.2f);

lightingShader.setVec3("light.ambient", ambientColor);
lightingShader.setVec3("light.diffuse", diffuseColor);
```

- 적용 결과



ⓒ nvidia

Lighting Maps

Lighting Maps

- 현실에서 사물은 여러 개의 material로 이루어져 있음
  - 예) 자동차의 exterior는 glossy하게 빛나는 재질, 창문은 부분적으로 반사/투과, 타이어는 highlight 없이 반사, rim은 강하게 빛남. 또한 모델별로 각기 다른 ambient, diffuse 컬러를 가짐

- 이전 형태의 물체별 Material 속성은, 물체 전체에 대해 공통적으로 적용
  - 현실의 복잡한 material을 표현하기 어려움

- 해법 - Lighting map을 사용하여 물체를 좀 더 정확하게 묘사
  - 물체의 diffuse (보통 ambient도 함께 포함) 컴포넌트와 specular 컴포넌트를 텍스처를 사용하여 표현
  - Diffuse map은 이전에 소개한 물체의 질감을 표현하는 텍스처와 동일
  - Specular map은 specular lighting 속성을 갖는 부분을 표현

Diffuse Maps

- Diffuse map
  - 빛이 존재하는 장면에서는 텍스처 이미지로 물체의 모든 diffuse 색상들을 나타낼 수 있음
  - 따라서 이러한 색상을 담은 map을 일반적으로 diffuse map이라 부름

- Diffuse map 적용 방법
  - Material 구조체의 diffuse 멤버를 sampler2D 형태로 변경하고, 텍스쳐 좌표를 입력 받음
  - 이 예제에서는 ambient 멤버를 생략했으나 필요시 유지. 물체별로 속성을 가지도록 하거나 별도 ambient map도 사용 가능

```
struct Material {          FS
    sampler2D diffuse;
    vec3      specular;
    float     shininess;
};
...
in vec2 TexCoords;
```

container2.png (500×500)
(learnopengl.com)

Diffuse Maps

- Diffuse map 적용 방법 (cont.)
  - Diffuse, ambient 색상 계산시 texture mapping 수행

```
vec3 diffuse = light.diffuse * diff * vec3(texture(material.diffuse, TexCoords));
vec3 ambient = light.ambient * vec3(texture(material.diffuse, TexCoords));
```
FS

  - VS에서는 위치, 노멀과 함께 텍스쳐 좌표도 함께 FS로 넘겨줘야 함

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexCoords;
...
out vec2 TexCoords;

void main()
{
    ...
    TexCoords = aTexCoords;
}
```
VS

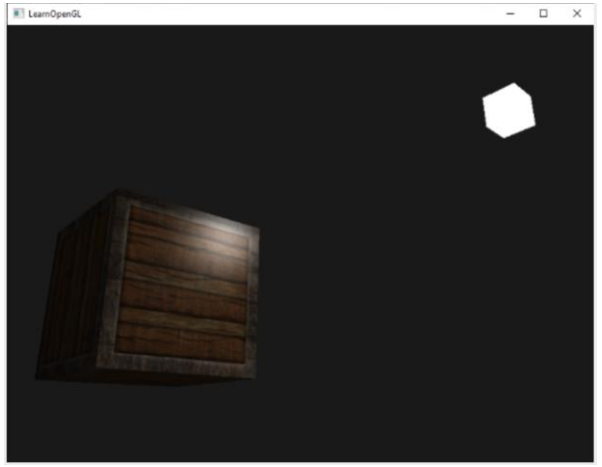
Diffuse Maps

- Diffuse map 적용 방법 (cont.)
  - CPU 쪽에서는 오른쪽에 입력된 텍스쳐 좌표를 처리 가능하도록 VAO를 새로 설정 (3주차 텍스쳐 매핑 참조)
  - Diffuse map도 0번 텍스쳐 유닛에 바인딩하여 FS로 넘겨줌

```
lightingShader.setInt("material.diffuse", 0);
...
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, diffuseMap);
```

CPU

- 실행 결과



Code Viewer. Source code: lighting/vertex_data_textures (learnopengl.com)

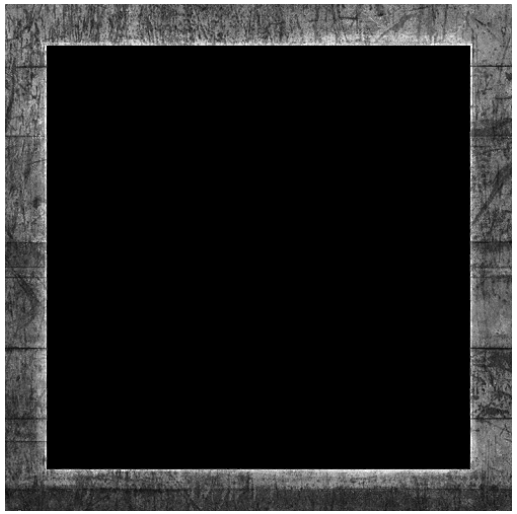
```
float vertices[] = {
    // positions          // normals           // texture coords
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  0.0f, 0.0f,
     0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  1.0f, 0.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  1.0f, 1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  1.0f, 1.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  0.0f, 1.0f,
    -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,  0.0f, 0.0f,

    -0.5f, -0.5f,  0.5f,  0.0f,  0.0f, 1.0f,   0.0f, 0.0f,
     0.5f, -0.5f,  0.5f,  0.0f,  0.0f, 1.0f,   1.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  0.0f, 1.0f,   1.0f, 1.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  0.0f, 1.0f,   1.0f, 1.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  0.0f, 1.0f,   0.0f, 1.0f,
    -0.5f, -0.5f,  0.5f,  0.0f,  0.0f, 1.0f,   0.0f, 0.0f,

    -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,  1.0f, 0.0f,
    -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,  1.0f, 1.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,  0.0f, 1.0f,
    -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,  0.0f, 1.0f,
    -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,  0.0f, 0.0f,
    -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,  1.0f, 0.0f,

     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,  1.0f, 0.0f,
     0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,  1.0f, 1.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,  0.0f, 1.0f,
     0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,  0.0f, 1.0f,
     0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,  0.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,  1.0f, 0.0f,

    -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,  0.0f, 1.0f,
     0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,  1.0f, 1.0f,
     0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,  1.0f, 0.0f,
     0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,  1.0f, 0.0f,
    -0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,  0.0f, 0.0f,
    -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,  0.0f, 1.0f,

    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,  0.0f, 1.0f,
     0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,  1.0f, 1.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,  1.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,  1.0f, 0.0f,
    -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,  0.0f, 0.0f,
    -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,  0.0f, 1.0f
};
```

CPU

Specular Maps

- 왼쪽 아래 diffuse map에 포함된 material은 각기 다른 specular highlight를 가짐
  - 중앙의 나무는 specular highlight를 미약하게 가짐
  - 테두리 철은 specular highlight를 가지며, 그 정도는 부분별로 다름 (오래되어 긁힌 부분은 빛이 덜 반사됨)
- 오른쪽 아래 specular map은 이와 같이 부분별로 다른 specular intensity를 흑백으로 표현
  - 나무 영역은 모두 0이라고 가정, 테두리 철은 부분별로 0~1의 값을 가짐
  - Photoshop이나 Gimp와 같은 이미지 편집 툴에서 쉽게 생성 가능 (일부 영역 삭제 후 흑백 변환)

container2.png (500×500) (learnopengl.com)



container2_specular.png (500×500) (learnopengl.com)

Specular Maps

- Specular map 샘플링
  - Diffuse map과 유사한 과정을 거치나, 이와 다른 텍스처 유닛(아래 예에서는 1번)을 사용해야 함 (diffuse map과 specular map이 동시에 접근 가능해야 하기 때문)

```
lightingShader.setInt("material.specular", 1);
...
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, specularMap);
```
CPU

  - FS의 구조체 및 색상 계산 부분 업데이트

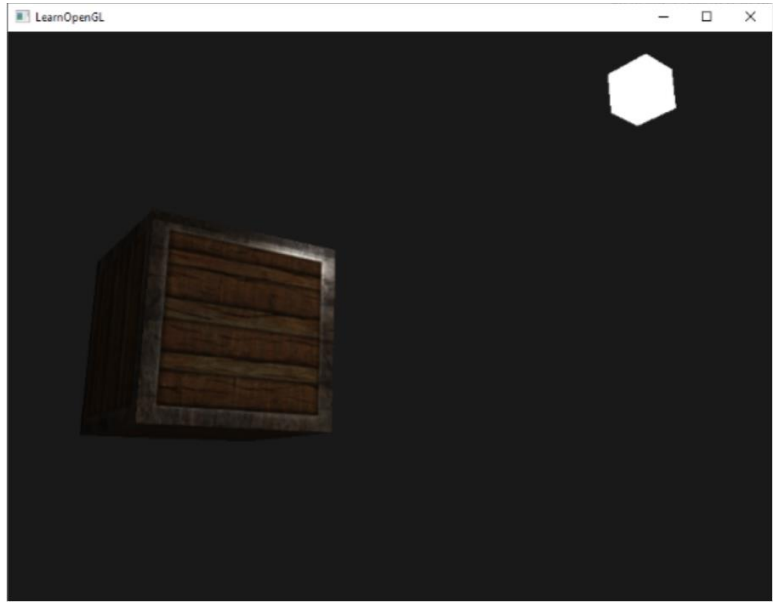
```
struct Material {
    sampler2D diffuse;
    sampler2D specular;
    float     shininess;
};
```
FS

```
vec3 specular = light.specular * spec * vec3(texture(material.specular, TexCoords));
```

- Specular highlight의 색상은 대부분 광원에 의해 결정되므로, specular map에는 3D specular 색상 대신 1D로 specular 세기를 저장

Specular Maps

- 실행 결과



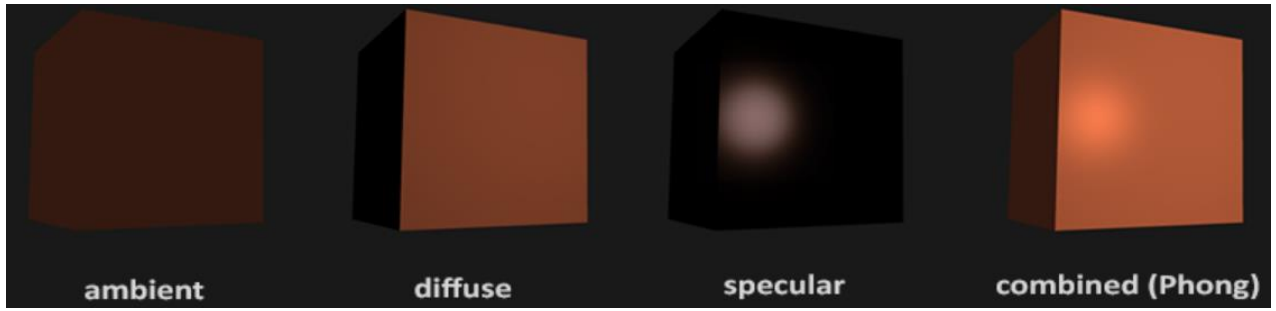
- 다양한 효과를 위해, diffuse/specular map 외에도 다양한 정보를 가진 텍스처 맵을 사용 가능
  - 올록볼록한 효과를 주기 위한 normal/bump map
  - 반사 정보를 담는 reflection map

ⓒ nvidia

마무리

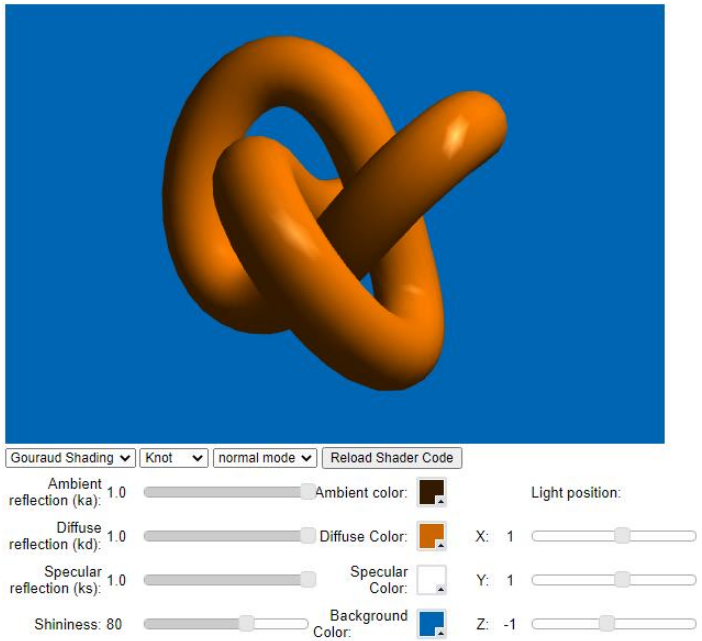
마무리

- 이번 시간에는 아래와 같은 내용을 살펴보았습니다.
  - 빛과 물체의 색상
  - Phong lighting model
  - Material 설정 방법
  - Lighting map의 개념



- Phong lighting model webGL demo
  - Gouraud / Phong shading 선택 가능

WebGL - Phong Shading (toronto.edu)



실습 문제

- 기반 코드
  - 아래 기반 코드를 실행하여 diffuse/specular map이 적용된 컨테이너에서의 lighting 확인 (코드 4.2)
    LearnOpenGL/src/2.lighting/4.2.lighting_maps_specular_map at master · JoeyDeVries/LearnOpenGL

- 문제 1
  - 물체가 스스로 발광(emit)하는 것처럼 보이도록 emission map(matrix.jpg)을 추가로 매핑
    LearnOpenGL/src/2.lighting/4.4.lighting_maps_exercise4 at master · JoeyDeVries/LearnOpenGL

- 문제 2
  - 램프 큐브의 색깔과 빛의 색깔이 함께 주기적으로 바뀌도록 변경. 아래 코드 응용
    Code Viewer. Source code: src/2.lighting/3.1.materials/materials.cpp (learnopengl.com)

- 문제 3
  - 램프 큐브가 회전하도록 설정. 아래 코드 응용
    Code Viewer. Source code: src/2.lighting/2.3.basic_lighting_exercise1/basic_lighting_exercise1.cpp (learnopengl.com)