

OpenGL 조명

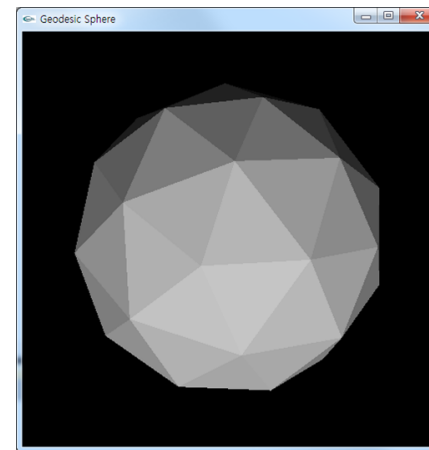
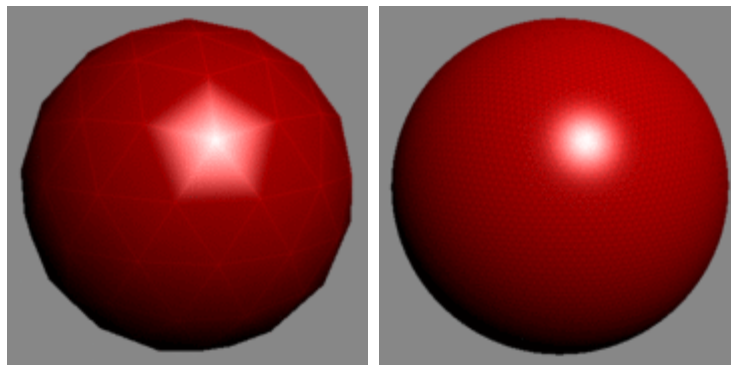
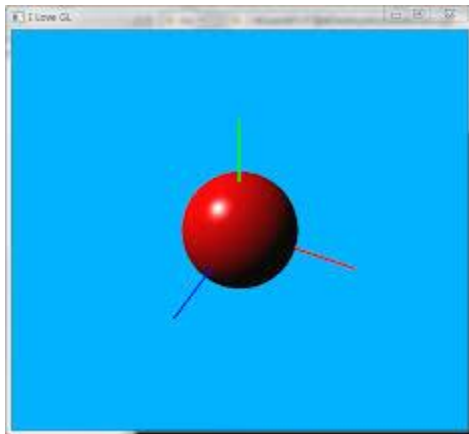
전역 조명, 산란반사 조명, 거울 조명

조명

- 광원은 색(color)과 강도(intensity)를 가지고 있다.
 - 기본색은 백색
 - 우리가 보는 객체의 색은 빛의 색 (light color)과 객체의 재질의 색 (material color)의 합성으로 정해진다.
 - 빛의 색: 조명의 RGB 값
 - 재질이 색: 객체의 표면에서 RGB 값이 얼마나 반사되는지
 - 예)
 - 백색 조명과 빨간색 재질: $(1.0, 1.0, 1.0) * (1.0, 0.0, 0.0) \rightarrow (1.0, 0.0, 0.0)$ 객체의 색은 빨간색
 - 빨간색 조명과 흰색 재질: $(1.0, 0.0, 0.0) * (1.0, 1.0, 1.0) \rightarrow (1.0, 0.0, 0.0)$ 객체의 색은 빨간색
 - 빨간색 조명과 파랑색 재질: $(1.0, 0.0, 0.0) * (0.0, 0.0, 1.0) \rightarrow (0.0, 0.0, 0.0)$ 객체의 색은 검정색
 - 노란색 조명과 청록색 재질: $(1.0, 1.0, 0.0) * (0.0, 1.0, 1.0) \rightarrow (0.0, 1.0, 0.0)$ 객체의 색은 초록색

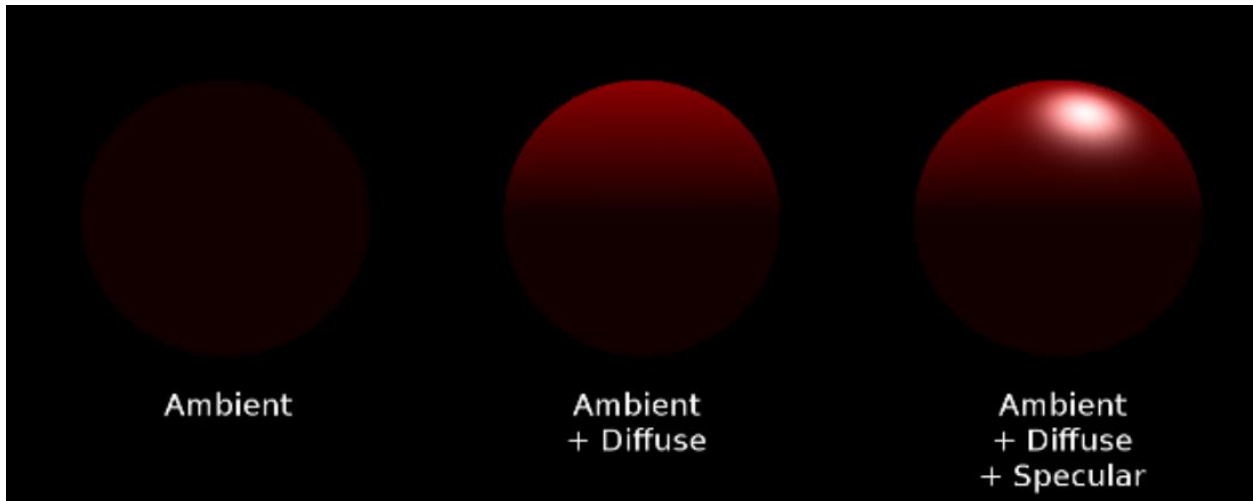
조명

- OpenGL에서 빛 (조명)은
 - 빛의 3원색인 빨간색, 초록색, 파란색의 성분
 - 광원의 색은 광원에서 방출하는 R, G, B 성분의 양에 따라 결정
 - 표면의 재질은 표면에 들어왔다가 다양한 방향으로 반사되는 빛의 RGB 성분의 비율로 결정
 - 따라서, OpenGL의 조명은 **광원 (Lights)과 재질(Materials), 그리고 면의 법선벡터(Normal)**에 의해서 결정된다.
 - 광원의 위치는 현재 변환에 영향을 받는다.
 - 광원은 객체와 같이 변환 행렬에 의해 움직일 수 있다.
 - 버텍스 셰이더 또는 프래그먼트 셰이더에서 객체의 색에 관련된 설정을 정하여 조명 효과를 추가한다.



조명

- 조명
 - 주변 조명 (ambient light)
 - 배경 조명, 전역 조명
 - 모든 방향으로 고루 비춰지는 조명
 - 산란 반사 조명 (diffuse light)
 - 방향과 위치를 가지고 있는 조명으로 빛이 비치는 물체의 면이 밝아진다.
 - 빛을 받는 표면은 그렇지 않은 부분에 비해 밝게 보인다.
 - 거울반사 조명 (specular light)
 - 특정한 방향으로 들어와서 한 방향으로 완전히 반사되는 조명
 - 반짝이는 표면을 모델링할 때 사용됨



주변 조명 (Ambient light)

- 주변 조명 (전역 조명)
 - 간접적으로 들어오는 빛을 나타내는 조명
 - 객체의 위치나 방향과 관계없이 일정한 밝기의 빛이 고르게 퍼져있다고 정함
 - 주변 조명 효과는 광원의 주변조명색과 객체의 색으로 결정함
 - 조명의 컬러와 주변 조명의 세기를 곱해서 주변 조명값으로 정한다.
 - 또는 특정 상수값으로 설정한다.

- 설정

```
uniform vec3 objectColor;
```

```
uniform vec3 ambientLightColor;
```

```
vec3 ambientLight = ambientLightColor;
```

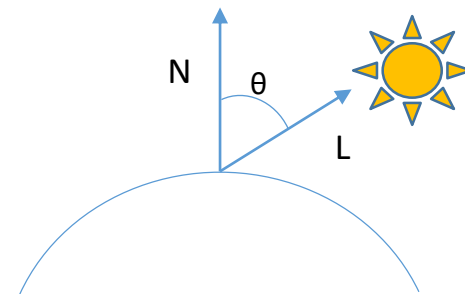
```
vec3 result = ambientLight * objectColor;
```

//--- 객체의 색과 조명의 색상을 곱하여 최종 객체 색상 설정

산란반사조명 (Diffuse light)

• 산란반사조명

- 특정 방향으로 진행하다가 표면에 닿으면 모든 방향으로 동일하게 반사된다.
- 관찰자의 위치와 무관하며 가장 일반적인 조명
- 조명을 받는 부분은 안 받는 부분보다 환하게 보인다.
- 객체의 표면과 광원과의 각도에 따라 조명 값이 달라진다.
 - 물체의 표면이 광원을 향하여 정면으로 향하고 있을 때 가장 많은 빛을 받고, 비스듬하게 놓인 경우에는 상대적으로 적은 양의 빛을 받게 된다.
 - 표면이 받는 빛의 양은 광원과 법선 벡터의 내적에 비례한다.
 - 램버트의 코사인 법칙
 - $\cos\theta = \mathbf{N} \cdot \mathbf{L}$
 - N: 표면의 법선벡터 (정규벡터)
 - L: 광원의 방향벡터 (정규벡터)
 - N과 L 사이의 각도가 클수록 산란반사조명의 값은 어두워진다.



• 설정

```
vec3 normalVector = Normal;  
vec3 lightDir = normalize (lightPos - FragPos);  
float diffuseLight = max (dot (norm, lightDir), 0.0);
```

```
//--- 표면과 조명의 위치로 조명의 방향을 결정한다.  
//--- N과 L의 내적 값으로 강도 조절 (음의 값을 가질 수 없게 한다.)
```

```
float diffuse = diffuseLight * lightColor;
```

```
//--- 산란반사조명값=산란반사값*조명색상값
```

거울반사조명 (Specular light)

- 거울반사조명 (정반사 조명)
 - 반짝이는 하이라이트를 생성함
 - 관찰자가 빛의 입사각과 거의 같은 반사각 부근에 위치할 경우 입사된 빛의 전부를 인식하며 하이라이트가 생긴다.
 - 거울반사조명의 반사각과 관찰자의 각도가 작을수록 많은 빛을 반사한다.
 - 거울 반사 조명은 조명의 색, 객체의 색 외에 재질의 shininess 정도를 추가하여 shininess 가 높으면 작은 면적의 하이라이트가 생성된다.

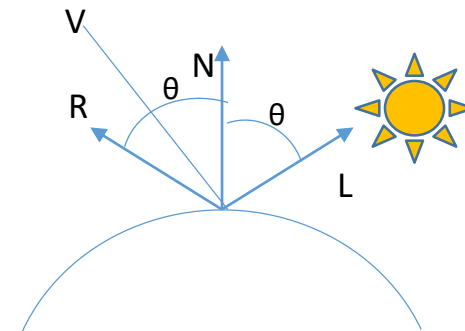
- 설정

```
int shininess = 32;  
vec3 normVector = Normal;  
vec3 lightDir = normalize (lightPos - FragPos);  
vec3 viewDir = normalize (FragPos - viewPos);  
vec3 reflectDir = reflect (-lightDir, normVector);  
float specularColor = max (dot (viewDir, reflectDir), 0.0);
```

```
specularColor = pow(specularColor, shininess);
```

```
// Normal: 표면의 법선 벡터  
// 조명의 방향  
// ViewPos: 관찰자의 위치값, FragPos: 객체의 위치값  
// reflect 함수: 입사 벡터의 반사 방향 계산  
// 거울반사 값 설정: 음수 방지
```

```
// shininess 승을 해주어 하이라이트를 만들어준다.
```



N: normal vector
L: 조명을 가리키는 벡터
R: 빛의 반사 방향
V: 관찰자 위치

최종 조명 효과

- 표면에 조명 효과를 모델링할 수 있는 공식

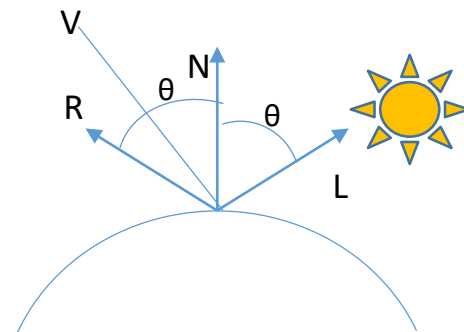
$$I = K_a I_a + K_d (N \cdot L) I_d + K_s (V \cdot R)^n I_s$$

- Gouraud shading (고라우드 셰이딩)

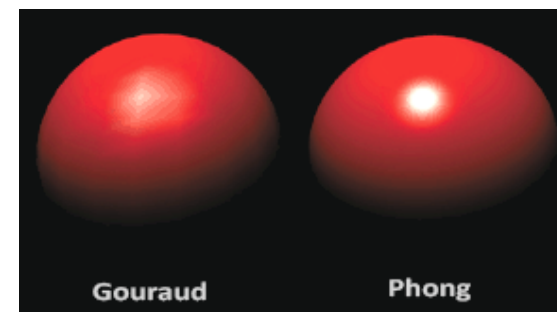
- 버텍스 당 라이팅 값을 계산
- 버텍스 간 결과 색상들을 보간하여 셰이딩에 사용
- 전체 라이팅 공식을 버텍스 셰이더에서 구현
- 각 프래그먼트의 최종 색상은 최종 색상은 프래그먼트 셰이더로 전달
 - 각 프래그먼트 색상은 프래그먼트 셰이더로 전달되기 전에 보간
 - 프래그먼트 셰이더는 단순히 입력 색상을 프레임버퍼에 쓰는 역할
 - 스페큘러 하이라이트 부분에 별 모양의 패턴
 - 삼각형 간의 불일치, 즉 색상 공간 내에서 선형적으로 보간되기 때문

- Phong shading (퐁 셰이딩)

- 버텍스간에 색상값을 보간하는 것이 아니라
- 버텍스 간의 표면 법선벡터를 보간
- 그 결과 법선값을 사용하여 버텍스가 아닌 픽셀에 대해 전체 라이팅 계산을 수행
 - 고라우드 셰이딩 같은 별 모양이 없고 훨씬 좋은 결과물
 - 프래그먼트 셰이더에서 많은 작업을 수행한다.



N: normal vector
L: 조명을 가리키는 벡터
R: 빛의 반사 방향
V: 관찰자 위치



주변 조명 (Ambient light)

- 프래그먼트 셰이더에 조명값을 적용한다.

//--- 프래그먼트 셰이더

#version 330 core

out vec4 FragColor;

uniform vec3 objectColor;

uniform vec3 ambientLightColor;

void main ()

{

vec3 ambientLight = ambientLightColor;

vec3 result = ambientLight * objectColor;

FragColor = vec4 (result, 1.0);

}

//--- 객체의 색과 조명의 색상을 곱하여 최종 객체 색상 설정

산란 반사 조명 (Diffuse light)

- 산란반사 조명 설정

- 응용 프로그램

```
float vertices[] = {           // 꼭지점 속성: x, y, z, normalX, normalY, normalZ
-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,

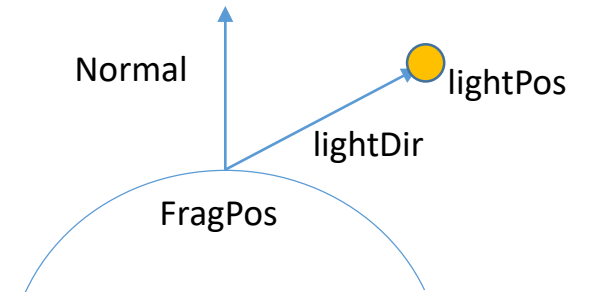
-0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f,      0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f,      0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f,
0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f,      -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f,      -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f,

-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
};
```



산란 반사 조명 (Diffuse light)

- 응용 프로그램

```
unsigned int VBO, VAO;
```

```
glGenVertexArrays(1, &VAO);  
glGenBuffers(1, &VBO);
```

```
glBindVertexArray(VAO);  
glBindBuffer(GL_ARRAY_BUFFER, VBO);  
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);  
glEnableVertexAttribArray(0);  
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));  
glEnableVertexAttribArray(1);
```

```
glUseProgram(shaderProgram);  
int lightPosLocation = glGetUniformLocation(shaderProgram, "lightPos");  
glUniform3f(lightPosLocation, 1.0, 0.5, 0.0);  
int lightColorLocation = glGetUniformLocation(shaderProgram, "lightColor");  
glUniform3f(lightColorLocation, 1.0, 0.0, 1.0);  
int objColorLocation = glGetUniformLocation(shaderProgram, "objectColor");  
glUniform3f(objColorLocation, 0.5, 0.5, 0.5);
```

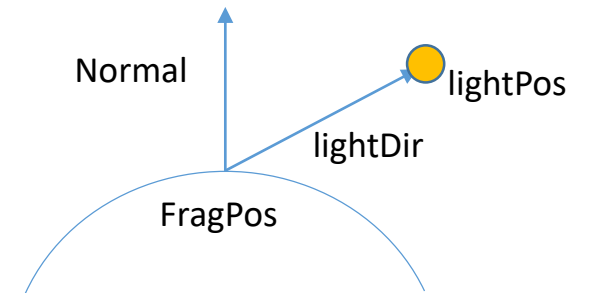
//--- 위치 속성

//--- 노말 속성

//--- lightPos 값 전달

//--- lightColor 값 전달

//--- object Color값 전달



산란 반사 조명 (Diffuse light)

- 셰이더

//--- 버텍스 셰이더

layout (location = 0) in vec3 vPos;

layout (location = 1) in vec3 vNormal;

out vec3 FragPos;

out vec3 Normal;

//--- 객체의 위치값을 프래그먼트 셰이더로 보낸다.

//--- 노멀값을 프래그먼트 셰이더로 보낸다.

//--- projection, view, model 값은 uniform으로 설정한다.

uniform mat4 model;

uniform mat4 view;

uniform mat4 projection;

Void main()

{

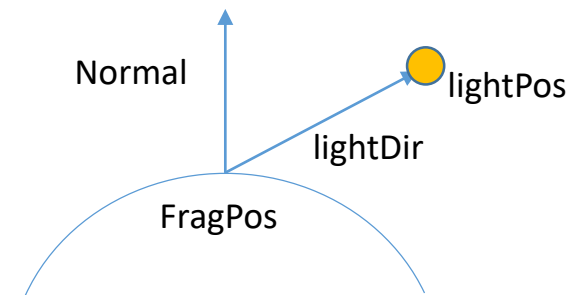
FragPos = vec3(model * vec4(vPos, 1.0));

gl_Position = projection * view * model * vec4(vPos, 1.0);

Normal = normalize (vNormal);

//--- 노멀값을 정규화 하고, 프래그먼트 셰이더로 보낸다.

}



산란 반사 조명 (Diffuse light)

- 셰이더

//--- 프래그먼트 셰이더

#version 330 core

in vec4 FragPos;

in vec3 Normal;

out vec4 FragColor;

uniform vec3 lightPos;

uniform vec3 lightColor;

uniform vec3 objectColor;

void main ()

{

vec3 ambientLight = 0.2;

vec3 ambient = ambientLight * lightColor;

vec3 normalVector = Normal;

vec3 lightDir = normalize (lightPos - FragPos);

float diffuseLight = max (**dot** (norm, lightDir), 0.0);

float diffuse = diffuseLight * lightColor;

vec3 result = (ambient + diffuse) * objectColor;

FragColor = vec4 (result, 1.0);

}

//--- 위치값

//--- 버텍스 셰이더에서 받은 노멀값

//--- 최종 객체의 색 저장

//--- 조명의 위치

//--- 조명의 색

//--- 객체의 색

//--- 전역 조명 세기

//--- 전역 조명 값

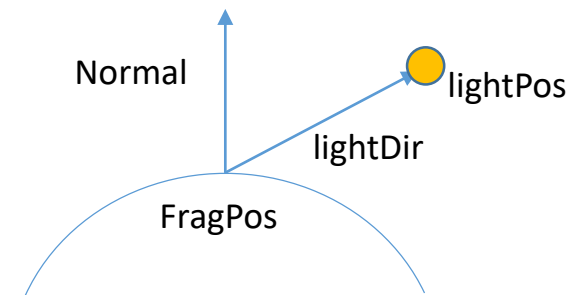
//--- 표면과 조명의 위치로 조명의 방향을 결정한다.

//--- N과 L의 내적 값으로 강도 조절 (음의 값을 가질 수 없게 한다.)

//--- 산란반사조명값=산란반사값*조명색상값

//--- 최종 산란반사조명값 =전역 조명+산란 반사 조명

//--- 픽셀 색을 출력



거울 반사 조명 (Specular light)

- 셰이더

- //--- 버텍스 셰이더

```
out vec3 FragPos;    // 객체의 위치값을 프래그먼트 셰이더로 보낸다.  
out vec3 Normal;     // 노멀값을 프래그먼트 셰이더로 보낸다.
```

```
//--- projection, view, model 값은 uniform으로 설정한다.
```

```
uniform mat4 model;  
uniform mat4 view;  
uniform mat4 projection;
```

```
void main()
```

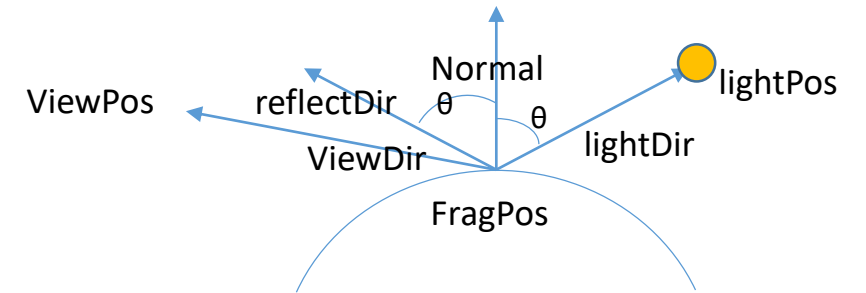
```
{
```

```
    gl_Position = projection * view * model * vec4(vPos, 1.0);
```

```
    FragPos = vec3(model * vec4(vPos, 1.0));
```

```
    Normal = normalize (vNormal);
```

```
}
```



거울 반사 조명 (Specular light)

- 셰이더

//--- 프래그먼트 셰이더

#version 330 core

in vec4 FragPos;

// 노멀값을 계산하기 위해 객체의 위치값을 버텍스 셰이더에서 받아온다.

in vec3 Normal;

out vec4 FragColor

void main ()

{

vec3 ambientLight = 0.2;

//--- 전역 조명 세기

vec3 ambient = ambientLight * lightColor;

//--- 전역 조명 값

vec3 normalVector = Normal;

vec3 lightDir = normalize (lightPos - FragPos);

//--- 표면과 조명의 위치로 조명의 방향을 결정한다.

float diffuseLight = max (dot (norm, lightDir), 0.0);

//--- N과 L의 내적 값으로 강도 조절 (음의 값을 가질 수 없게 한다.)

float diffuse = diffuseLight * lightColor;

//--- 산란반사조명값=산란반사값*조명색상값

int shininess = 32;

vec3 normVector = Normal;

// Normal: 표면의 법선 벡터

vec3 lightDir = normalize (lightPos - FragPos);

// 조명의 방향

vec3 viewDir = normalize (FragPos - viewPos);

// ViewPos: 관찰자의 위치값, FragPos: 객체의 위치값

vec3 reflectDir = reflect (-lightDir, normVector);

// reflect 함수: 입사 벡터의 반사 방향 계산

float specularColor = max (dot (viewDir, reflectDir), 0.0);

// 거울반사 값 설정: 음수 방지

specularColor = pow(specularColor, shininess);

// shininess 승을 해주어 하이라이트를 만들어준다.

vec3 specular = specularColor * lightColor;

// 거울 반사 조명의 색을 결정

vec3 result = (ambient + diffuse + specular) * objectColor;

// 전역 조명, 산란 반사 조명, 거울 반사 조명을 함께 적용

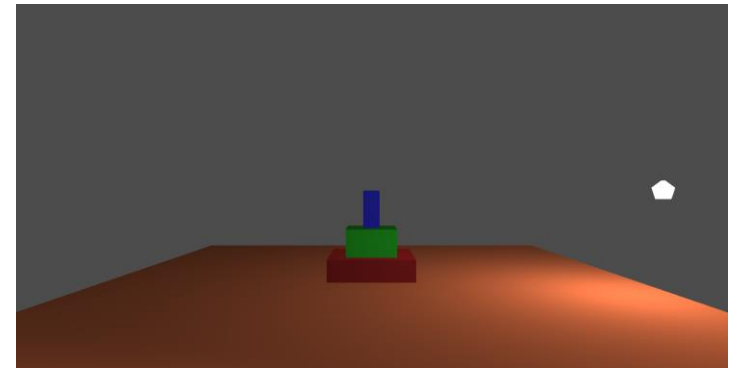
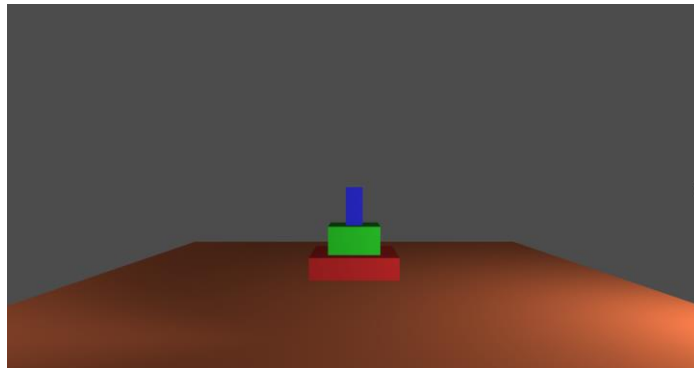
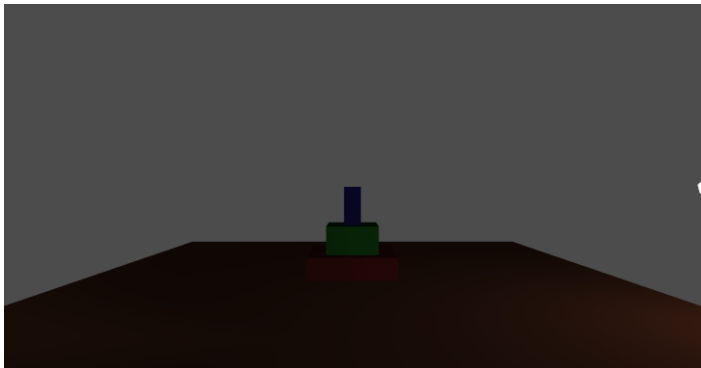
FragColor = vec4 (result, 1.0);

/ / 픽셀 색을 출력

}

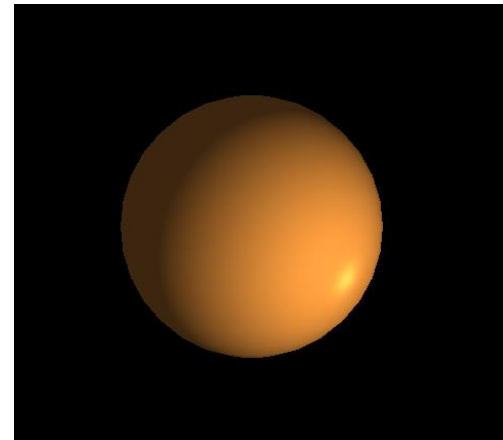
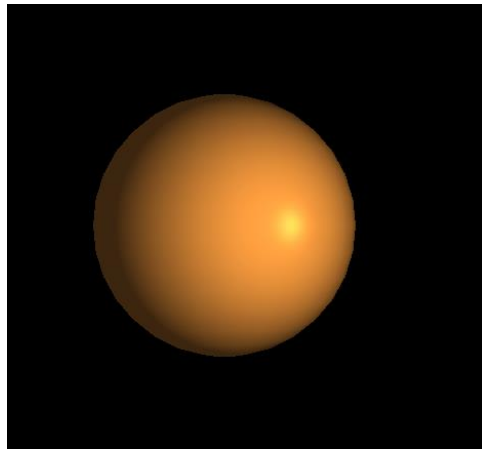
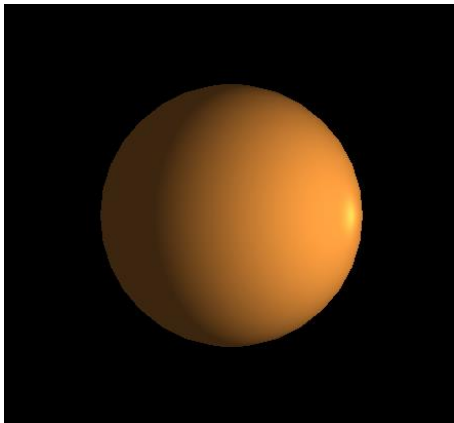
실습 19

- 조명 넣기
 - 크레인 실습(실습 16)에 조명을 적용한다.
 - 바닥에는 크레인이 이동
 - 크레인의 몸체는 각각 다른 색상 (조명)으로 설정한다.
 - 화면의 좌 상단에 조명을 넣는다
 - 키보드 명령
 - m/M: 조명 켜기/끄기
 - r/R: 조명이 중심의 y축 기준으로 양/음 방향으로 회전하기
 - s/S: 회전 멈추기



실습 20

- 조명 넣기
 - 태양, 지구, 달 회전 실습 (실습 13) 에 조명을 적용한다.
 - 조명이 화면 우측에 놓여있다.
 - 키보드 명령
 - 조명 색 바꾸기: 조명 색을 다른 색으로 바꿔도록 한다. 3종류의 다른 색을 적용해본다.
 - 조명 켜기/끄기: 조명 켜기/끄기.
 - 조명 회전하기: 조명의 위치에 따라 다양하게 적용해본다.



실습 21