CG Practice 7

COLLEGE OF COMPUTING

HANYANG ERICA CAMPUS

Q YOUN HONG (홍규연)

Shading Surfaces

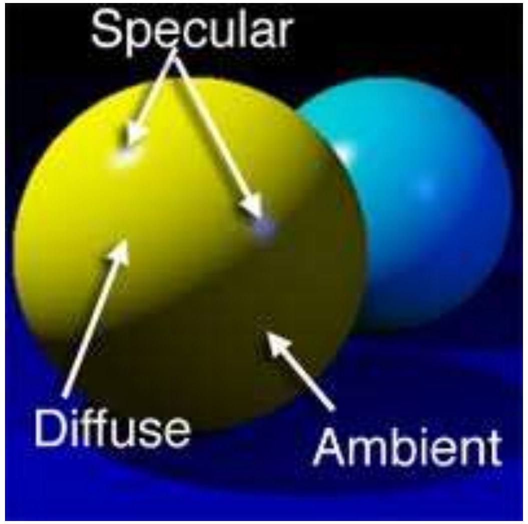
Goal

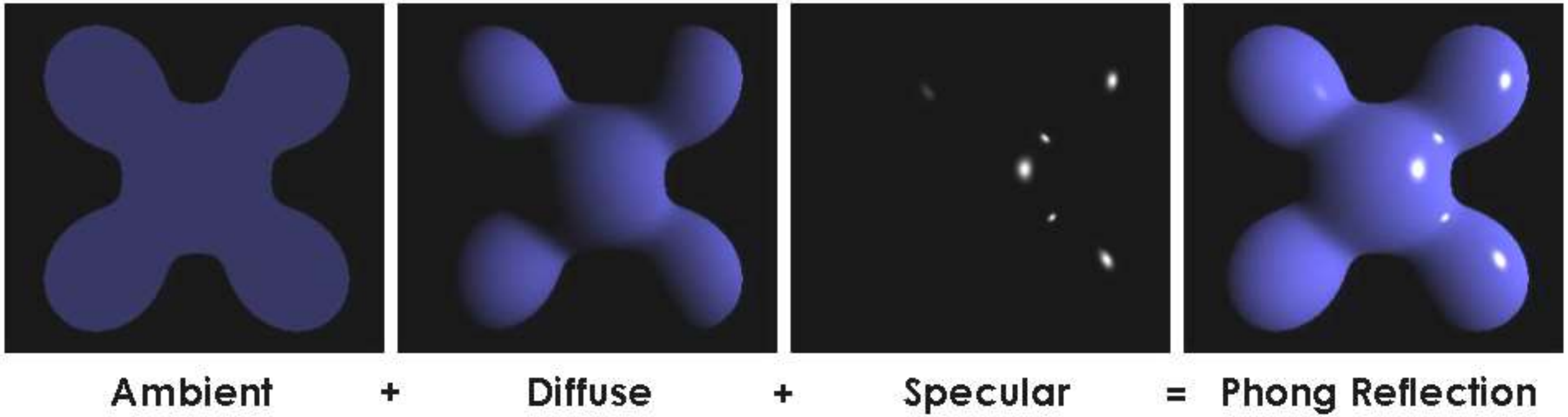
- 간단한 물체를 shading model을 적용해서 그려보기
  - Rotating Cube에 shading model을 적용해서 그려보기

Illumination Equation

- 전체 Illumination Equation

$$I = I_a k_a + I_p(k_d\langle N, L\rangle + k_s\langle R, V\rangle^n)$$





Lighting Parameters

- Light sources (광원)
  - 모든 광원에 대해서 각각의 색상과 위치 (혹은 방향)을 지정해야 함
  - 광원의 색은 다음의 세 요소로 이루어져 있음

```
color4 light_ambient = color4(0.2, 0.2, 0.2, 1.0);
color4 light_diffuse = color4(1.0, 1.0, 1.0, 1.0);
color4 light_specular = color4(1.0, 1.0, 1.0, 1.0);
```

  - 광원의 위치는 다음과 같이 homogeneous coordinate로 지정

```
point4 light_position;
```

→ Point light source의 경우: `point4 light_position = vec4(1, 2, 3, 1);`

→ Directional light source의 경우: `point4 light_position = vec4(1, 2, 3, 0);`

Lighting Parameters

- Materials
  - Material parameter는 object의 종류에 따라 달라짐
  - Front와 back face에 다른 material parameter를 지정할 수도 있음
  - Ambient, diffuse, specular light의 반사율 (k_a, k_d, k_s)를 설정

```
color4 material_ambient = color4(1.0, 0.0, 1.0, 1.0);
color4 material_diffuse = color4(1.0, 0.8, 0.0, 1.0);
color4 material_specular = color4(1.0, 0.8, 0.0, 1.0);
```

  - Shininess: specular component 계산에 사용

```
float material_shininess = 100.0;
```

  - Emission: 스스로 발광하는 물체의 경우

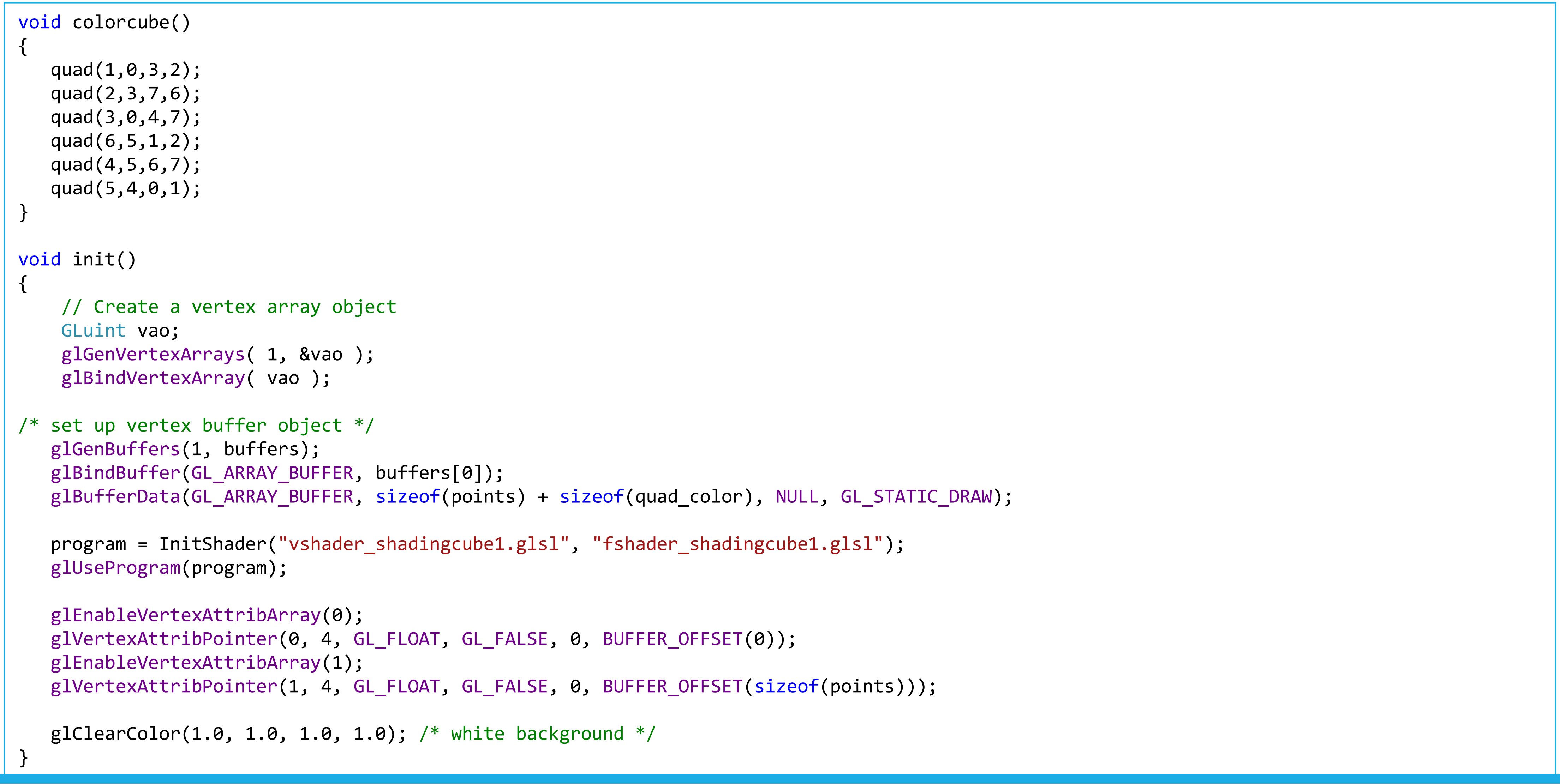
```
color4 material_emission = color4(0.0, 0.3, 0.3, 1.0);
```

Rotating Cube Again

```
#include <vgl.h>
#include <InitShader.h>
#include <mat.h>
#include <cstdio>
#include <cstdlib>
#include <vector>

const int NumVertices = 36; //(6 faces)(2 triangles/face)(3
vertices/triangle)

typedef vec4  point4;
typedef vec4  color4;

int axis = 0;
float theta[3] = {0.0, 0.0, 0.0};
int toggle_spin = 0;

GLuint buffers[2];
GLint matrix_loc;

point4  vertices[8] = {point4(-0.5,-0.5,0.5, 1.0),point4(-0.5,0.5,0.5,
1.0),
   point4(0.5,0.5,0.5, 1.0), point4(0.5,-0.5,0.5, 1.0), point4(-0.5,-
0.5,-0.5, 1.0),
   point4(-0.5,0.5,-0.5, 1.0), point4(0.5,0.5,-0.5, 1.0), point4(0.5,-
0.5,-0.5, 1.0)};
```

```
point4 points[NumVertices];
color4 quad_color[NumVertices];
mat4 ctm;

GLuint program;

// matrix functions

// product of components

vec4 product(vec4 a, vec4 b)
{
   return vec4(a[0]*b[0], a[1]*b[1], a[2]*b[2], a[3]*b[3]);
}
```

Rotating Cube Again



```
void colorcube()
{
   quad(1,0,3,2);
   quad(2,3,7,6);
   quad(3,0,4,7);
   quad(6,5,1,2);
   quad(4,5,6,7);
   quad(5,4,0,1);
}

void init()
{
    // Create a vertex array object
    GLuint vao;
    glGenVertexArrays( 1, &vao );
    glBindVertexArray( vao );

/* set up vertex buffer object */
   glGenBuffers(1, buffers);
   glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);
   glBufferData(GL_ARRAY_BUFFER, sizeof(points) + sizeof(quad_color), NULL, GL_STATIC_DRAW);

   program = InitShader("vshader_shadingcube1.glsl", "fshader_shadingcube1.glsl");
   glUseProgram(program);

   glEnableVertexAttribArray(0);
   glVertexAttribPointer(0, 4, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(0));
   glEnableVertexAttribArray(1);
   glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(sizeof(points)));

   glClearColor(1.0, 1.0, 1.0, 1.0); /* white background */
}
```

Rotating Cube Again

```
void mouse(int btn, int state, int x, int y)
{
    if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
    if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis =
1;
    if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis =
2;
}

void spinCube()
{
    if (toggle_spin == 1) {
      theta[axis] += 0.05;
      if (theta[axis] > 360.0) theta[axis] -= 360.0;
      glutPostRedisplay();
    }
}

void mykey(unsigned char key, int mousex, int mousey)
{
   if(key=='q'||key=='Q') exit(0);
   if (key == ' ')
       toggle_spin = 1 - toggle_spin;
}
```

```
int main(int argc, char** argv)
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_DEPTH |
GLUT_DOUBLE);
    glutInitWindowSize(512, 512);

    glutInitContextVersion( 3, 2 );
    glutInitContextProfile( GLUT_CORE_PROFILE );

    glutCreateWindow("Color Cube");
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    glutIdleFunc(spinCube);
    glutKeyboardFunc(mykey);

    glewInit();
    init();

    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
    return 0;
}
```

Rotating Cube – New Part

- Lighting Parameter 정의

```
vec4 viewer = vec4(0.0, 0.0, 1.0, 0.0);
point4 light_position = point4(0.0, 0.0, -1.0, 0.0);
color4 light_ambient = color4(0.2, 0.2, 0.2, 1.0);
color4 light_diffuse = color4(1.0, 1.0, 1.0, 1.0);
color4 light_specular = color4(1.0, 1.0, 1.0, 1.0);

color4 material_ambient = color4(1.0, 0.0, 1.0, 1.0);
color4 material_diffuse = color4(1.0, 0.8, 0.0, 1.0);
color4 material_specular = color4(1.0, 0.8, 0.0, 1.0);
float material_shininess = 100.0;
```

Rotating Cube – New Part

```
void display( void )
{
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     ctm =
RotateX(theta[0])*RotateY(theta[1])*RotateZ(theta[2]);
     colorcube();

     glBufferSubData( GL_ARRAY_BUFFER, 0,
sizeof(points), points );
     glBufferSubData( GL_ARRAY_BUFFER, sizeof(points),
sizeof(quad_color), quad_color );

     glDrawArrays(GL_TRIANGLES, 0, NumVertices);
     glutSwapBuffers();
}
```

colorcube(): rotation에 따라서 cube의 각 면의 법선 벡터들이 변하고, 법선 벡터가 변하면 diffuse, specular light의 크기가 변함
⇒ display에서 shading 계산을 새로함

Rotating Cube – New Part

- Cube의 각 면의 shading color를 결정
  - Cube의 색은 면에 의해 결정됨 (flat shading)
  - `quad(int a, int b, int c, int d)` – rotation에 의해 바뀌는 vertex position과 vertex normal 계산하고, 이를 이용해서 vertex color를 계산

$$I = I_a k_a + I_p(k_d\langle N, L\rangle + k_s\langle R, V\rangle^n)$$

Rotating Cube – in quad()

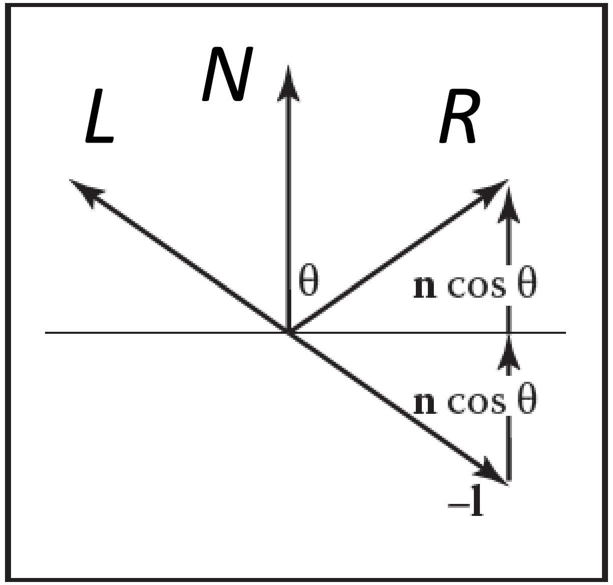
$$I = I_a k_a + I_p (k_d \langle N, L \rangle + k_s \langle R, V \rangle^n)$$

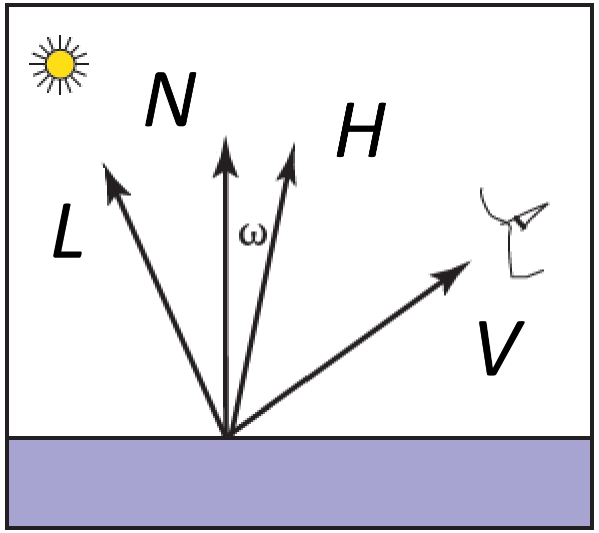
- ambient color 결정

```
ambient_color = product(material_ambient, light_ambient);
```

- diffuse color 결정

```
vec3 n1 = normalize(cross(ctm*vertices[b] -
ctm*vertices[a], ctm*vertices[c] - ctm*vertices[b]));
vec4 n = vec4(n1[0], n1[1], n1[2], 0.0);
if(dd>0.0) diffuse_color = dd*product(light_diffuse,
                                       material_diffuse);
else diffuse_color =  color4(0.0, 0.0, 0.0, 1.0);
```





Rotating Cube – in quad()

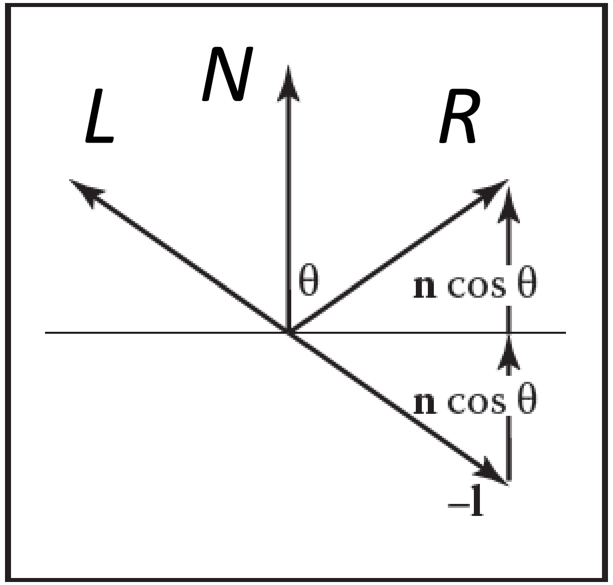
$$I = I_a k_a + I_p(k_d\langle N, L\rangle + k_s\langle R, V\rangle^n)$$

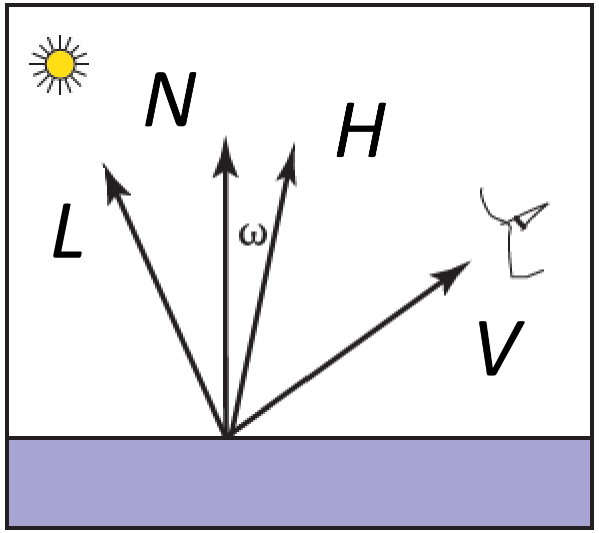
- specular color 결정
  - $\cos\alpha = \langle R, V\rangle$의 근사

$$\rightarrow \cos\omega = \langle N, H\rangle = \left\langle N, \frac{V+L}{|V+L|}\right\rangle$$

```
vec4 half = normalize(light_position+viewer);
dd = dot(half, n);
if(dd > 0.0)
   specular_color =
exp(material_shininess*log(dd))*product(light_specular,
material_specular);
else specular_color = vec4(0.0, 0.0, 0.0, 1.0);
```

dd^s, s = material shininess

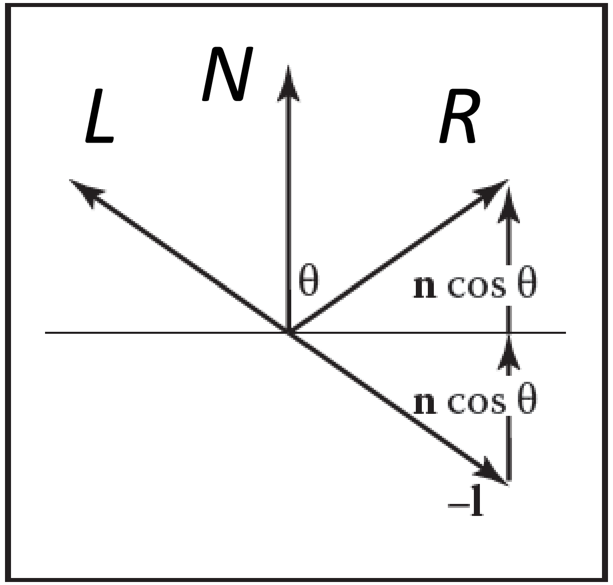


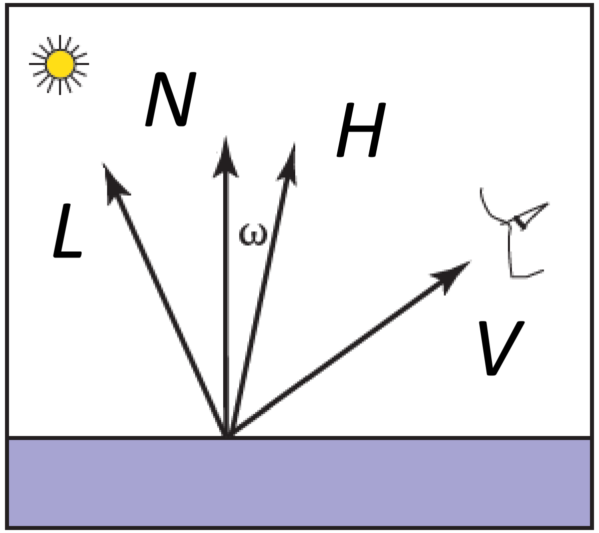


Rotating Cube – in quad()

$$I = I_a k_a + I_p(k_d\langle N,L\rangle + k_s\langle R,V\rangle^n)$$

```
    quad_color[i] = ambient_color + diffuse_color + specular color;
    points[i] = ctm*vertices[a];
    i++;
    quad_color[i] = ambient_color + diffuse_color + specular color;
    points[i] = ctm*vertices[b];
    i++;
    quad_color[i] = ambient_color + diffuse_color + specular color;
    points[i] = ctm*vertices[c];
    i++;
    quad_color[i] = ambient_color + diffuse_color + specular color;
    points[i] = ctm*vertices[a];
    i++;
    quad_color[i] = ambient_color + diffuse_color + specular color;
    points[i] = ctm*vertices[c];
    i++;
    quad_color[i] = ambient_color + diffuse_color + specular color;
    points[i] = ctm*vertices[d];
    i++;
    i%=NumVertices;
```





Rotating Cube

```
#version 330

layout (location = 0) in vec4 vPosition;
layout (location = 1) in vec4 vColor;
out vec4 color;

void main()
{
  gl_Position = vPosition;
  color = vColor;
}
```

```
#version 330

in vec4 color;
out vec4 fColor;

void main()
{
  fColor = color;
}
```

Method 2 - Lighting in Vertex Shader

- model view matrix, projection matrix를 넘기기

```
// Model-view and projection matrices uniform
location
GLuint  ModelView, Projection;
```

```
void
display( void )
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //  Generate tha model-view matrixn

    const vec3 viewer_pos( 0.0, 0.0, 2.0 );
    mat4  model_view = ( Translate( -viewer_pos ) *
 RotateX( Theta[Xaxis] ) *
 RotateY( Theta[Yaxis] ) *
 RotateZ( Theta[Zaxis] ) );

    glUniformMatrix4fv( ModelView, 1, GL_TRUE, model_view );

    glDrawArrays( GL_TRIANGLES, 0, NumVertices );
    glutSwapBuffers();
}
```

Lighting in Vertex Shader

- verex position과 함께, vertex normal도 shader로 넘김

⇒ Initial position에서의 vertex normal 계산

```
void
quad( int a, int b, int c, int d )
{
    // Initialize temporary vectors along the quad's edge to
    //   compute its face normal
    vec4 u = vertices[b] - vertices[a];
    vec4 v = vertices[c] - vertices[b];

    vec3 normal = normalize( cross(u, v) );

    normals[Index] = normal; points[Index] = vertices[a]; Index++;
    normals[Index] = normal; points[Index] = vertices[b]; Index++;
    normals[Index] = normal; points[Index] = vertices[c]; Index++;
    normals[Index] = normal; points[Index] = vertices[a]; Index++;
    normals[Index] = normal; points[Index] = vertices[c]; Index++;
    normals[Index] = normal; points[Index] = vertices[d]; Index++;
}
```

Lighting in Vertex Shader

```
Void init()
{
    colorcube();

    // Create a vertex array object
    GLuint vao;
    glGenVertexArrays( 1, &vao );
    glBindVertexArray( vao );

    // Create and initialize a buffer object
    GLuint buffer;
    glGenBuffers( 1, &buffer );
    glBindBuffer( GL_ARRAY_BUFFER, buffer );
    glBufferData( GL_ARRAY_BUFFER, sizeof(points) + sizeof(normals),
  NULL, GL_STATIC_DRAW );
    glBufferSubData( GL_ARRAY_BUFFER, 0, sizeof(points), points );
    glBufferSubData( GL_ARRAY_BUFFER, sizeof(points),
     sizeof(normals), normals );

    // Load shaders and use the resulting shader program
    GLuint program = InitShader( "vshader_shadingcube2.glsl",
"fshader_shadingcube2.glsl" );
    glUseProgram( program );

    // set up vertex arrays
    glEnableVertexAttribArray(0);
    glVertexAttribPointer( 0, 4, GL_FLOAT, GL_FALSE, 0,
   BUFFER_OFFSET(0) );
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0,
   BUFFER_OFFSET(sizeof(points)) );
```

Vertex normal들도 vertex buffer에 저장하기

Vertex normal들도 vertex attribute으로 설정하기

Lighting in Vertex Shader – init()

- Lighting parameter uniform 변수로 shader에 넘기기

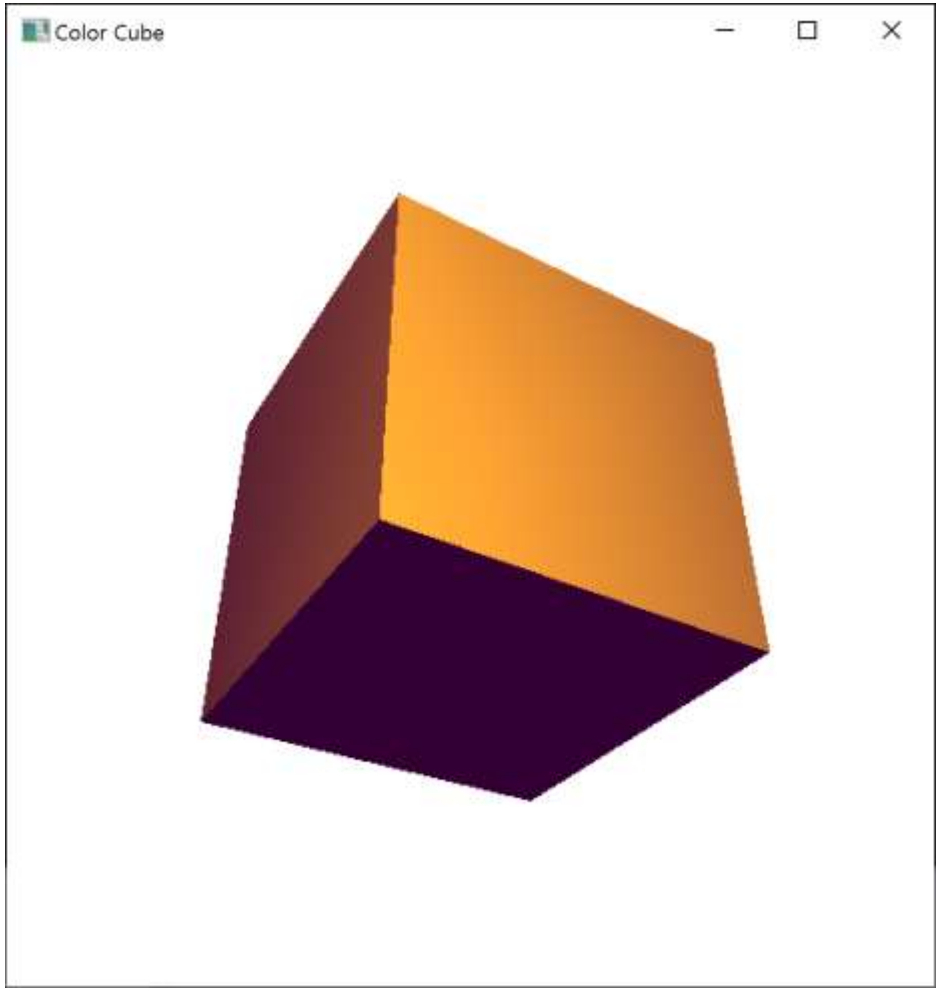
```
// Initialize shader lighting parameters
    point4 light_position( 0.0, 0.0, -1.0, 0.0 );
    color4 light_ambient( 0.2, 0.2, 0.2, 1.0 );
    color4 light_diffuse( 1.0, 1.0, 1.0, 1.0 );
    color4 light_specular( 1.0, 1.0, 1.0, 1.0 );

    color4 material_ambient( 1.0, 0.0, 1.0, 1.0 );
    color4 material_diffuse( 1.0, 0.8, 0.0, 1.0 );
    color4 material_specular( 1.0, 0.8, 0.0, 1.0 );
    float  material_shininess = 100.0;

    color4 ambient_product = light_ambient * material_ambient;
    color4 diffuse_product = light_diffuse * material_diffuse;
    color4 specular_product = light_specular * material_specular;

    glUniform4fv( glGetUniformLocation(program, "AmbientProduct"), 1, ambient_product );
    glUniform4fv( glGetUniformLocation(program, "DiffuseProduct"), 1, diffuse_product );
    glUniform4fv( glGetUniformLocation(program, "SpecularProduct"), 1, specular_product );

    glUniform4fv( glGetUniformLocation(program, "LightPosition"), 1, light_position );
    glUniform1f( glGetUniformLocation(program, "Shininess"), material_shininess );
```

Lighting in Vertex Shader – in vshader

```
#version 330

layout (location = 0) in  vec4 vPosition;
layout (location = 1) in  vec3 vNormal;
out vec4 color;

uniform vec4 AmbientProduct, DiffuseProduct, SpecularProduct;
uniform mat4 ModelView, Projection;
uniform vec4 LightPosition;
uniform float Shininess;
void main()
{
    // Transform vertex  position into eye coordinates
    vec3 pos = (ModelView * vPosition).xyz;

    vec3 L = normalize( (ModelView*LightPosition).xyz - pos );
    vec3 E = normalize( -pos );
    vec3 H = normalize( L + E );

    // Transform vertex normal into eye coordinates
    vec3 N = normalize( ModelView*vec4(vNormal, 0.0) ).xyz;

    // Compute terms in the illumination equation
    vec4 ambient = AmbientProduct;

    float Kd = max( dot(L, N), 0.0 );
    vec4  diffuse = Kd*DiffuseProduct;

    float Ks = pow( max(dot(N, H), 0.0), Shininess );
    vec4  specular = Ks * SpecularProduct;

    if( dot(L, N) < 0.0 ) {
specular = vec4(0.0, 0.0, 0.0, 1.0);
    }

    gl_Position = Projection * ModelView * vPosition;

    color = ambient + diffuse + specular;
    color.a = 1.0;
}
```

왜 transpose(inverse(ModelView))대신 ModelView를 쓸 수 있는가?

Execution Result



Lighting in Fragment Shader

- lighting을 per-vertex가 아닌, per-fragment로 실행
- vertex shader에서 lighting을 계산하는 것보다 더 다양한 lighting 효과를 기대할 수 있음

Lighting in Fragment Shader

- 좀더 복잡한 물체를 그리기 위해, 사면체를 subdivision해서 구생성

```
const int NumTimesToSubdivide = 5;
const int NumTriangles        = 4096;  // (4 faces)^(NumTimesToSubdivide + 1)
const int NumVertices         = 3 * NumTriangles;

#define DivideByZeroTolerance 1e-6

typedef vec4 point4;
typedef vec4 color4;

point4 points[NumVertices];
vec3   normals[NumVertices];
int Index = 0;

void triangle( const point4& a, const point4& b, const point4& c )
{
    vec3  normal = normalize( cross(b - a, c - b) );

    normals[Index] = normal;  points[Index] = a;  Index++;
    normals[Index] = normal;  points[Index] = b;  Index++;
    normals[Index] = normal;  points[Index] = c;  Index++;
}

point4 unit( const point4& p )
{
    float len = p.x*p.x + p.y*p.y + p.z*p.z;

    point4 t;
    if ( len > DivideByZeroTolerance ) {
       t = p / sqrt(len);
       t.w = 1.0;
    }
    return t;
}
```

```
void divide_triangle( const point4& a, const point4& b, const point4& c, int count )
{
    if ( count > 0 ) {
        point4 v1 = unit( a + b );
        point4 v2 = unit( a + c );
        point4 v3 = unit( b + c );
        divide_triangle(  a, v1, v2, count - 1 );
        divide_triangle(  c, v2, v3, count - 1 );
        divide_triangle(  b, v3, v1, count - 1 );
        divide_triangle( v1, v3, v2, count - 1 );
    }
    else {
        triangle( a, b, c );
    }
}

Void tetrahedron( int count )
{
    point4 v[4] = { vec4( 0.0, 0.0, 1.0, 1.0 ), vec4( 0.0, 0.942809, -0.333333, 1.0 ),
                    vec4( -0.816497, -0.471405, -0.333333, 1.0 ), vec4( 0.816497, -
0.471405, -0.333333, 1.0 )
    };

    divide_triangle( v[0], v[1], v[2], count );
    divide_triangle( v[3], v[2], v[1], count );
    divide_triangle( v[0], v[3], v[1], count );
    divide_triangle( v[0], v[2], v[3], count );
}
```

Lighting in Fragment Shader

```
    point4 light_position( 0.0, 0.0, 2.0, 0.0 );
    color4 light_ambient( 0.2, 0.2, 0.2, 1.0 );
    color4 light_diffuse( 1.0, 1.0, 1.0, 1.0 );
    color4 light_specular( 1.0, 1.0, 1.0, 1.0 );

    color4 material_ambient( 1.0, 0.0, 1.0, 1.0 );
    color4 material_diffuse( 1.0, 0.8, 0.0, 1.0 );
    color4 material_specular( 1.0, 0.0, 1.0, 1.0 );
    float  material_shininess = 5.0;
```

```
void
display( void )
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    point4 at( 0.0, 0.0, 0.0, 1.0 );
    point4 eye( 0.0, 0.0, 2.0, 1.0 );
    vec4   up( 0.0, 1.0, 0.0, 0.0 );

    mat4 model_view = LookAt( eye, at, up );
    glUniformMatrix4fv( ModelView, 1, GL_TRUE, model_view );

    glDrawArrays( GL_TRIANGLES, 0, NumVertices );
    glutSwapBuffers();
}
```

Lighting in Fragment Shader

In Vertex Shader – vertex에 따라 변화하는 값만 계산

```
#version 330

layout (location = 0) in   vec4 vPosition;
layout (location = 1) in   vec3 vNormal;

// output values that will be interpretated per-fragment
out  vec3 fN;
out  vec3 fE;
out  vec3 fL;

uniform mat4 ModelView;
uniform vec4 LightPosition;
uniform mat4 Projection;

void main()
{
    fN = vNormal;
    fE = (ModelView*vPosition).xyz;
    fL = LightPosition.xyz;

    if( LightPosition.w != 0.0 ) {
       fL = LightPosition.xyz - vPosition.xyz;
    }

    gl_Position = Projection*ModelView*vPosition;
}
```

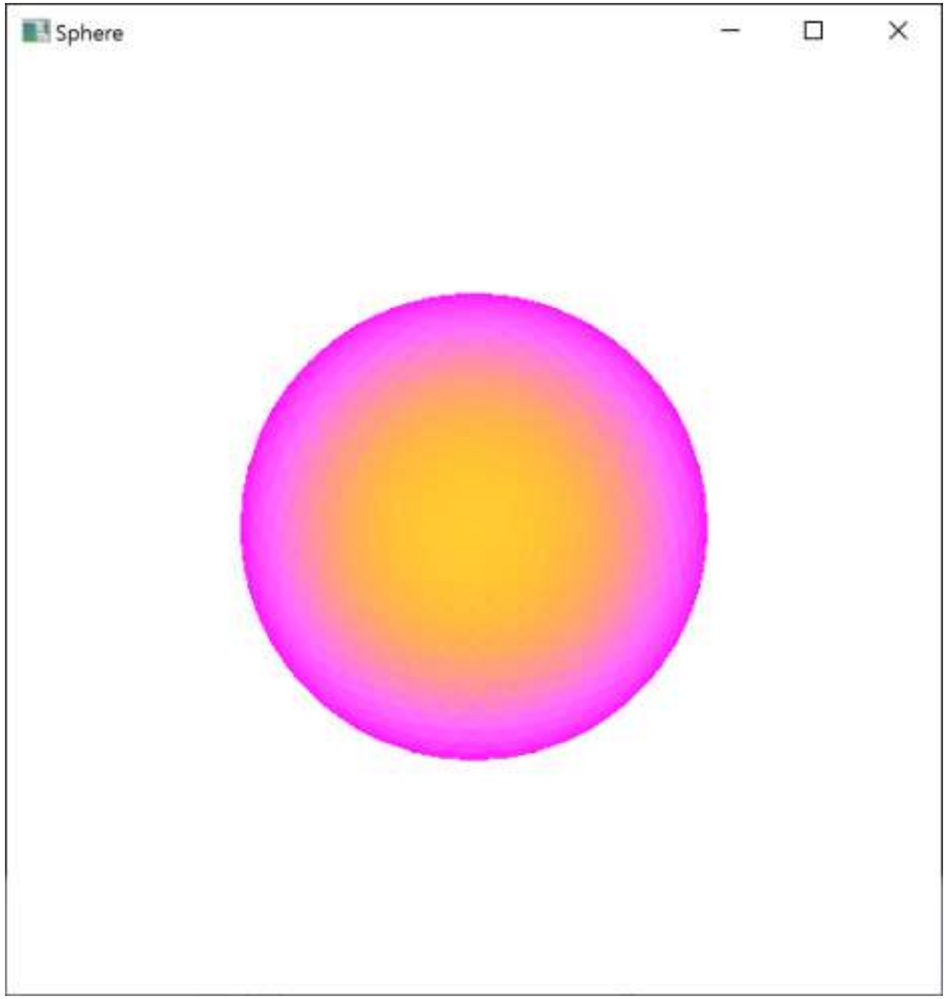
Lighting in Fragment Shader

In Fragment Shader – fragment 별로 color 계산

```
#version 330

// per-fragment interpolated values from the vertex shader
in  vec3 fN;
in  vec3 fL;
in  vec3 fE;

out vec4 fColor;

uniform vec4 AmbientProduct, DiffuseProduct, SpecularProduct;
uniform mat4 ModelView;
uniform vec4 LightPosition;
uniform float Shininess;

void main()
{
    // Normalize the input lighting vectors
    vec3 N = normalize(fN);
    vec3 E = normalize(fE);
    vec3 L = normalize(fL);

    vec3 H = normalize( L + E );
```

```
    vec4 ambient = AmbientProduct;

    float Kd = max(dot(L, N), 0.0);
    vec4 diffuse = Kd*DiffuseProduct;

    float Ks = pow(max(dot(N, H), 0.0), Shininess);
    vec4 specular = Ks*SpecularProduct;

    // discard the specular highlight if the light's behind the
vertex
    if( dot(L, N) < 0.0 ) {
specular = vec4(0.0, 0.0, 0.0, 1.0);
    }

    fColor = ambient + diffuse + specular;
    fColor.a = 1.0;
}
```

Execution Result



ReviewTask 2023-11-10 Shading Meshes

- Bunny.obj에 Shading을 추가하여 렌더링하기
  - GLUT Window의 제목을 자신의 학번으로 할 것
  - main.cpp, vshader, fshader, obj 파일 + (실행파일 및 스크린 샷)을 zip 파일에 압축하여 제출할 것