ARLAB

ME/CprE/ComS 557

Computer Graphics and Geometric Modeling

Light and Material Example

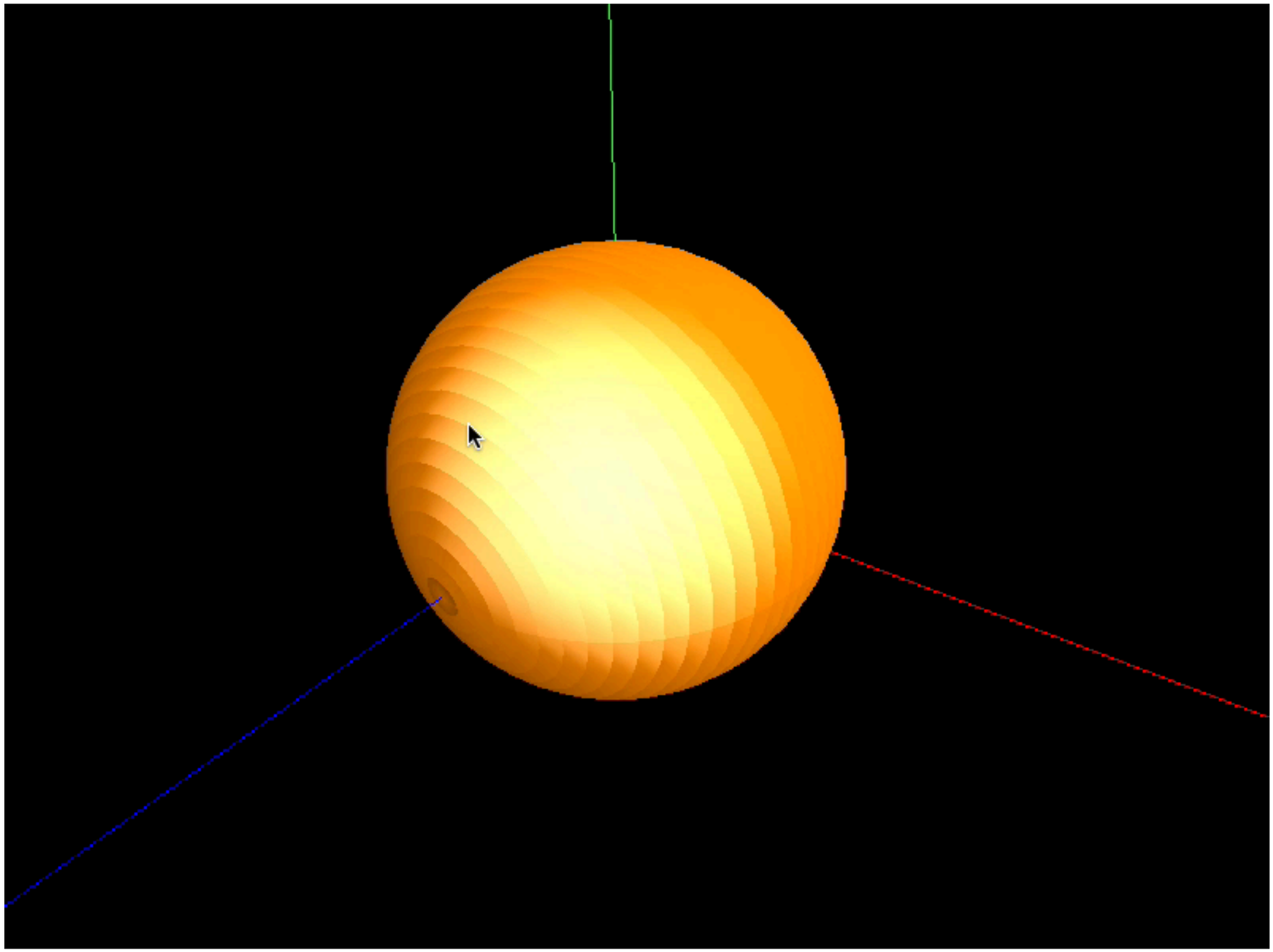
September 24, 2015

Rafael Radkowski

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Sphere Example

ARLAB



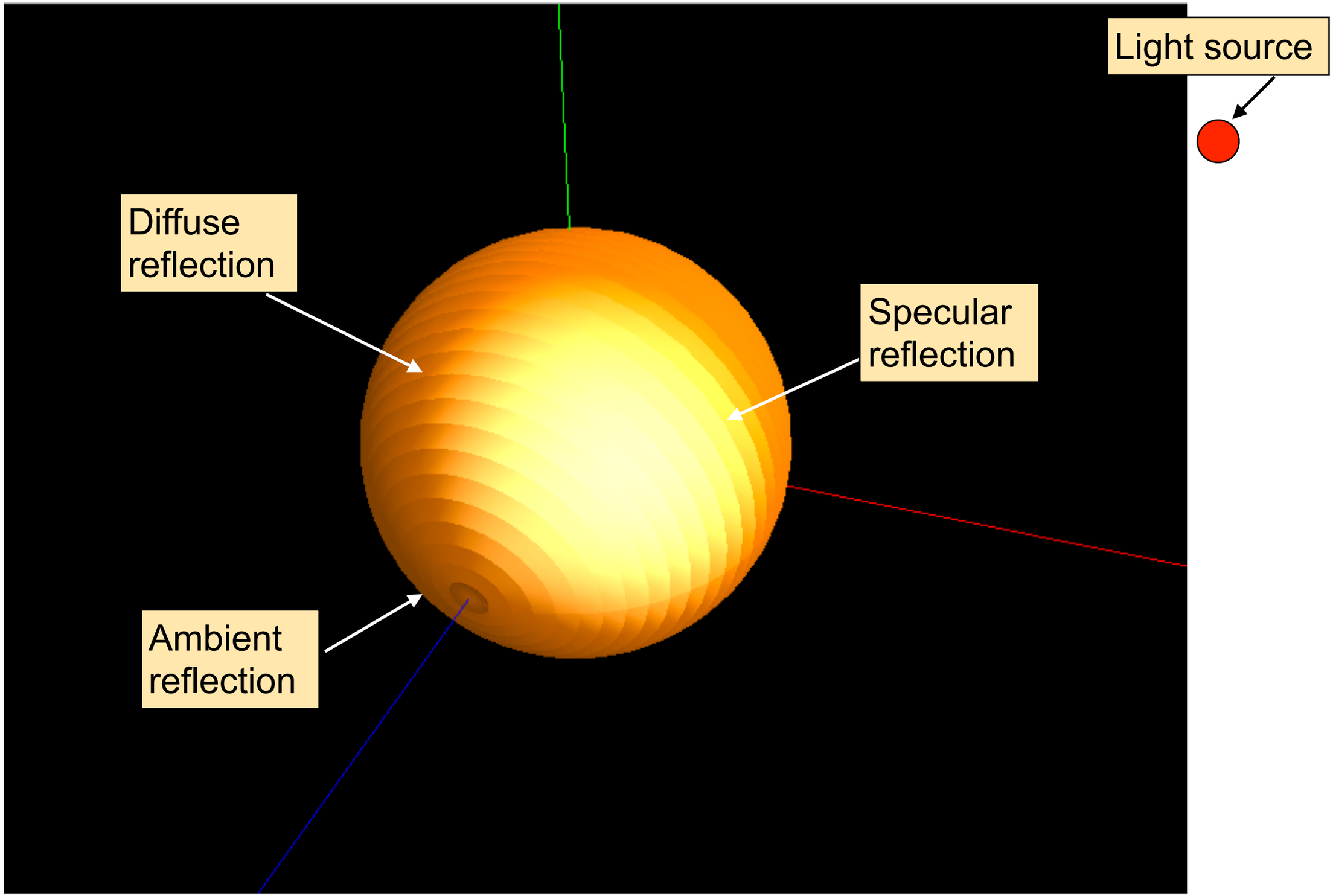
Content

ARLAB

- Creating geometry (sphere), postponed (we will talk about modeling later)
- Material and light, Uniform variables
- Code example
  - Ambient light
  - Diffuse light
  - Specular light

Sphere Example

ARLAB



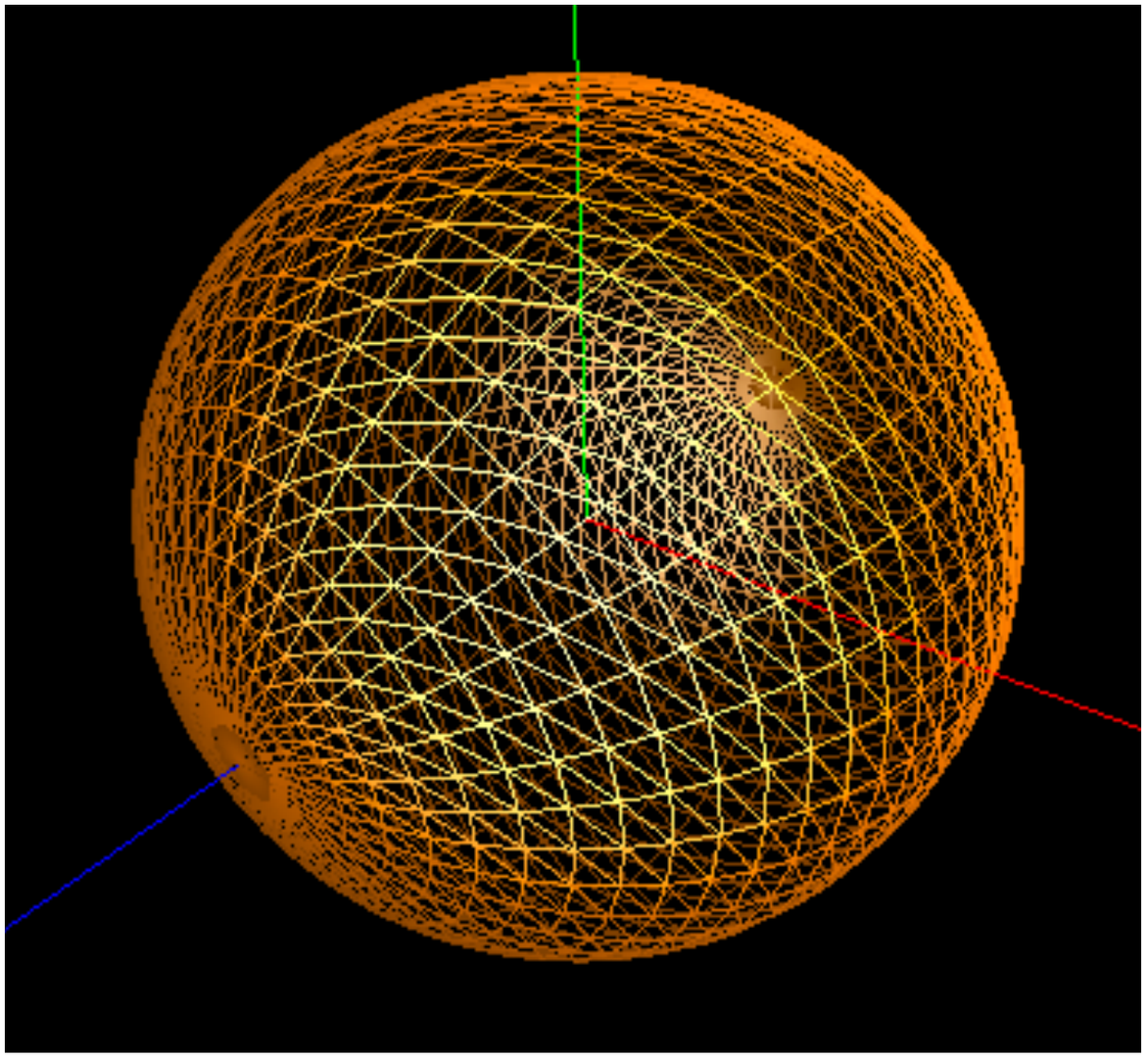
ARLAB

Create the geometry

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Sphere Example

Take the equation of a sphere
and create the points you need.



$$x = x_0 + r\cos\theta\ \sin\varphi$$
$$y = y_0 + r\sin\theta\ \sin\varphi \qquad (0 \leq \theta \leq 2\pi \text{ and } 0 \leq \varphi \leq \pi)$$
$$z = z_0 + r\cos\varphi$$

ARLAB

Material and Light

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Shader Code

```
static const string vs_string_GLSphere_410 =
"#version 410 core                                                 \n"
"                                                                  \n"
"uniform mat4 projectionMatrixBox;
"uniform mat4 viewMatrixBox;
"uniform mat4 modelMatrixBox;
"uniform vec3 diffuse_color;
"uniform vec3 ambient_color;
"uniform vec3 specular_color;                                        \n"
"uniform vec3 light_position;                                        \n"
"uniform float diffuse_intensity;                                        \n"
"uniform float ambient_intensity;                                        \n"
"uniform float specular_intensity;                                        \n"
"uniform float shininess;                                        \n"
"in vec3 in_Position;                                             \n"
"in vec3 in_Normal;                                                \n"
"in vec3 in_Color;                                                \n"
"out vec3 pass_Color;                                             \n"
"                                                                 \n"
"                                                                 \n"
"                                                                 \n"
"                                                                 \n"
"void main(void)                                                  \n"
"{                                                                \n"
"    vec3 normal = normalize(in_Normal);                                                \n"
"    vec4 transformedNormal =  normalize(transpose(inverse(modelMatrixBox)) * vec4( normal, 1.0 ));                \n"
"    vec4 surfacePostion = viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);                \n"
"                                                                                \n"
"    vec4 surface_to_light =   normalize( vec4(light_position,1.0) -  surfacePostion );                \n"
"                                                                                \n"
"    // Diffuse color                                                                \n"
"    float diffuse_coefficient = max( dot(transformedNormal, surface_to_light), 0.0);                \n"
"    vec3 out_diffuse_color = diffuse_color  * diffuse_coefficient * diffuse_intensity;                \n"
"                                                                                \n"
"    // Ambient color                                                                \n"
"    vec3 out_ambient_color = vec3(ambient_color) * ambient_intensity;                \n"
"                                                                                \n"
"    // Specular color                                                                \n"
"    vec3 incidenceVector = -surface_to_light.xyz;                \n"
"    vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);                \n"
"    vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2]);                \n"
"    vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);                \n"
"    float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);                \n"
"    float specular_coefficient = pow(cosAngle, shininess);                \n"
"    vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;                \n"
"                                                                                \n"
"    gl_Position = projectionMatrixBox * viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);                \n"
"                                                                                \n"
"    pass_Color = vec3(out_diffuse_color + out_ambient_color + out_specular_color);                \n"
"}                                                                                \n";
```

incoming and outgoing variables

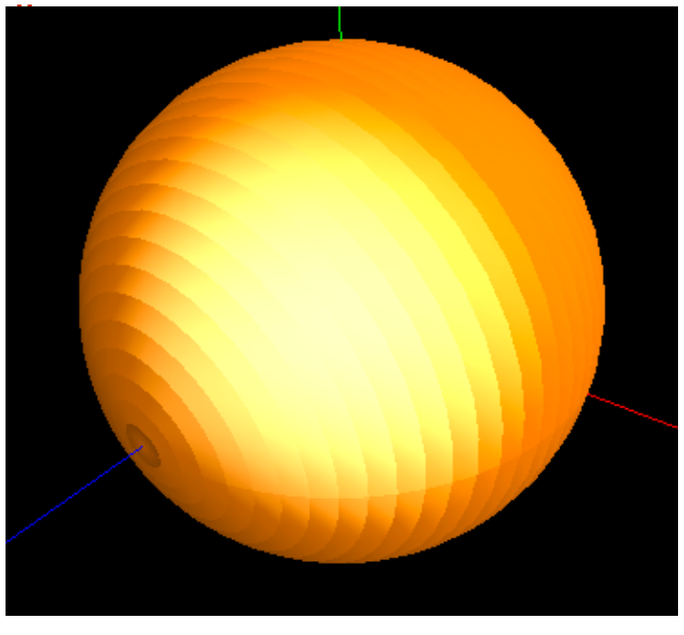
Vertex shader code in GLSphere.cpp

Shader Code

```
static const string vs_string_GLSphere_410 =
"#version 410 core
"
"uniform mat4 projectionMatrixBox;
"uniform mat4 viewMatrixBox;
"uniform mat4 modelMatrixBox;
"uniform vec3 diffuse_color;
"uniform vec3 ambient_color;
"uniform vec3 specular_color;
"uniform vec3 light_position;
"uniform float diffuse_intensity;
"uniform float ambient_intensity;
"uniform float specular_intensity;
"uniform float shininess;
"in vec3 in_Position;
"in vec3 in_Normal;
"in vec3 in_Color;
"out vec3 pass_Color;
"
```

Vertex shader code in GLSphere.cpp

```
      \n"
     \n"
     \n"
      \n"
      \n"
          \n"
          \n"
           \n"
          \n"
          \n"
         \n"
        \n"
        \n"
        \n"
```



We have to represent the “color” as a material and pass all data into the vertex shader program

GLMaterial

ARLAB

```
class GLMaterial
{
public:

    glm::vec3   _specular_material;
    glm::vec3   _diffuse_material;
    glm::vec3   _ambient_material;
    float       _shininess;

    int         _ambientColorPos;
    int         _diffuseColorPos;
    int         _specularColorPos;
    int         _shininessIdx;

    GLMaterial()
    {
        _specular_material = glm::vec3(1.0,0.0,0.0);
        _diffuse_material = glm::vec3(1.0,0.0,0.0);
        _ambient_material = glm::vec3(1.0,0.0,0.0);
    }

};
```

This class allows us to define a material and the associated glsl variable positions

GLLightSource

```
class GLLightSource
{
public:

    float        _specular_intensity;
    float        _diffuse_intensity;
    float        _ambient_intensity;


    // the glsl shader program positions
    int          _specularIdx;
    int          _diffuseIdx;
    int          _ambientIdx;

    glm::vec3    _lightPos;

    GLLightSource():
        _specular_intensity(1.0), _diffuse_intensity(1.0), _ambient_intensity(1.0)
    {
        _lightPos = glm::vec3(0.0,0.0,0.0);
        _specularIdx = _diffuseIdx =_ambientIdx  = -1;
    }

};
```

The class allows us to define a light source.
Right now, a light source is defined by

- intensity values for all components
- the position of the light source

Uniform variables

We define the materials as glsl uniform variables

```
 ////////////////////////////////////////////////////////////////////////////
// Material
_sphereMaterial._diffuse_material = glm::vec3(1.0, 0.5, 0.0);
_sphereMaterial._ambient_material = glm::vec3(1.0, 0.5, 0.0);
_sphereMaterial._specular_material = glm::vec3(1.0, 1.0, 1.0);
_sphereMaterial._shininess = 1.0;


_sphereMaterial._ambientColorPos = glGetUniformLocation(_program, "ambient_color");
_sphereMaterial._diffuseColorPos = glGetUniformLocation(_program, "diffuse_color");
_sphereMaterial._specularColorPos = glGetUniformLocation(_program, "specular_color");
_sphereMaterial._shininessIdx = glGetUniformLocation(_program, "shininess");


// Send the material to your shader program
glUniform3fv(_sphereMaterial._ambientColorPos, 1, &_sphereMaterial._ambient_material[0] );
glUniform3fv(_sphereMaterial._diffuseColorPos, 1, &_sphereMaterial._diffuse_material[0]);
glUniform3fv(_sphereMaterial._specularColorPos, 1, &_sphereMaterial._specular_material[0]);
glUniform1f(_sphereMaterial._shininessIdx, _sphereMaterial._shininess);
```

Two steps:

- Define the variable location and keep the index of the location
- Send the data to your program

glGetUniformLocation

ARLAB

Returns the location of a uniform variable

```
GLint glGetUniformLocation( GLuint program,
                            const GLchar *name);
```

Parameters:

- program: Specifies the program object to be queried.
- name: Points to a null terminated string containing the name of the uniform variable whose location is to be queried.

Note, the variable must already exist in the shader program. The program must be compiled and linked in advance!!

Memory

	Variable	Data
0	ambient_color	
1	diffuse_color	
2	specular_color	
3	shininess	

```
"uniform mat4 projectionMatrixBox;
"uniform mat4 viewMatrixBox;
"uniform mat4 modelMatrixBox;
"uniform vec3 diffuse_color;
"uniform vec3 ambient_color;
"uniform vec3 specular_color;
"uniform vec3 light_position;
"uniform float diffuse_intensity;
"uniform float ambient_intensity;
"uniform float specular_intensity;
"uniform float shininess;
"in vec3 in_Position;
"in vec3 in_Normal;
"in vec3 in_Color;
"out vec3 pass_Color;
```

glGetUniformLocation

Specify the value of a uniform variable for the **current** program object

```
void glUniform1f(   GLint location,
                    GLfloat v0);
```

Parameters:

- program: Specifies the program object to be queried.
- name: Points to a null terminated string containing the name of the uniform variable whose location is to be queried.

Note, the program must be active /activated with

glUseProgram(*[program]*);

Memory

	Variable	Data
0	ambient_color	
1	diffuse_color	Data
2	specular_color	
3	shininess	

```
"uniform mat4 projectionMatrixBox;
"uniform mat4 viewMatrixBox;
"uniform mat4 modelMatrixBox;
"uniform vec3 diffuse_color;
"uniform vec3 ambient_color;
"uniform vec3 specular_color;
"uniform vec3 light_position;
"uniform float diffuse_intensity;
"uniform float ambient_intensity;
"uniform float specular_intensity;
"uniform float shininess;
"in vec3 in_Position;
"in vec3 in_Normal;
"in vec3 in_Color;
"out vec3 pass_Color;
```

Light

We define the light as light intensity and as position

```
////////////////////////////////////////////////////////////////////////////////
// Light

// define the position of the light and send the light position to your shader program
_light_source0._lightPos = glm::vec3(50.0,50.0,0.0);
_light_source0._ambient_intensity = 0.5;
_light_source0._specular_intensity = 1.0;
_light_source0._diffuse_intensity = 1.0;
```

```
_light_source0._ambientIdx = glGetUniformLocation(_program, "ambient_intensity");
_light_source0._diffuseIdx = glGetUniformLocation(_program, "diffuse_intensity");
_light_source0._specularIdx = glGetUniformLocation(_program, "specular_intensity");

// Send the light information to your shader program
glUniform1f(_light_source0._ambientIdx, _light_source0._ambient_intensity );
glUniform1f(_light_source0._diffuseIdx, _light_source0._diffuse_intensity);
glUniform1f(_light_source0._specularIdx, _light_source0._specular_intensity);
glUniform3fv(_lightPosition, 1, &_light_source0._lightPos[0]);
```

Two steps:

- Define the variable location and keep the index of the location
- Send the data to your program

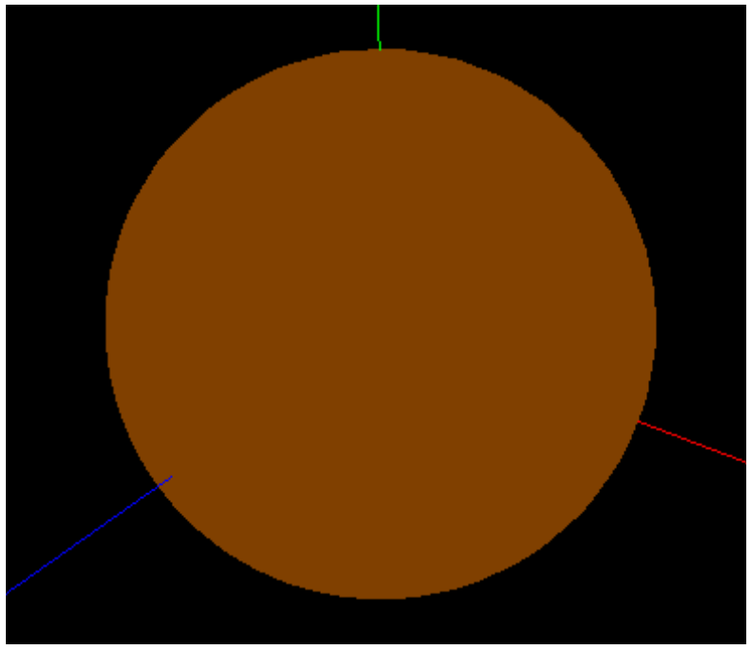
ARLAB

Code example

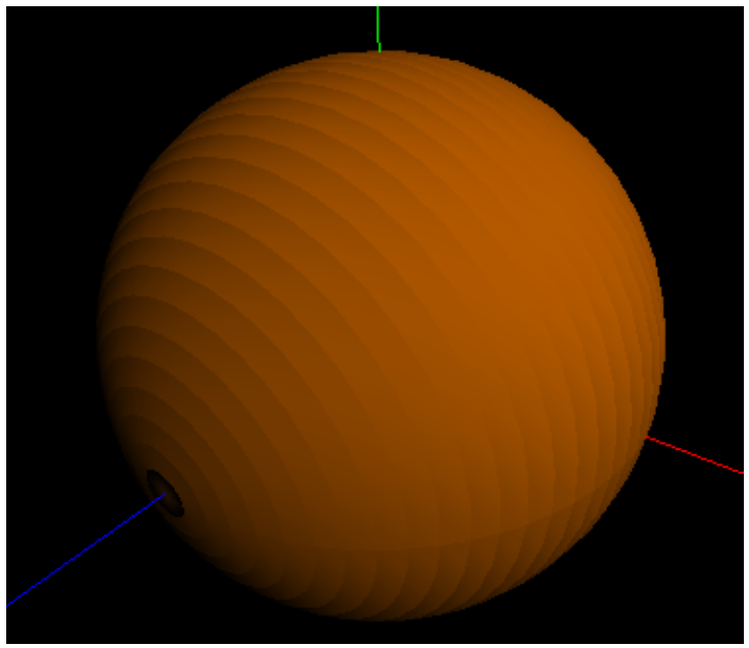
IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Ambient + Diffuse + Specular

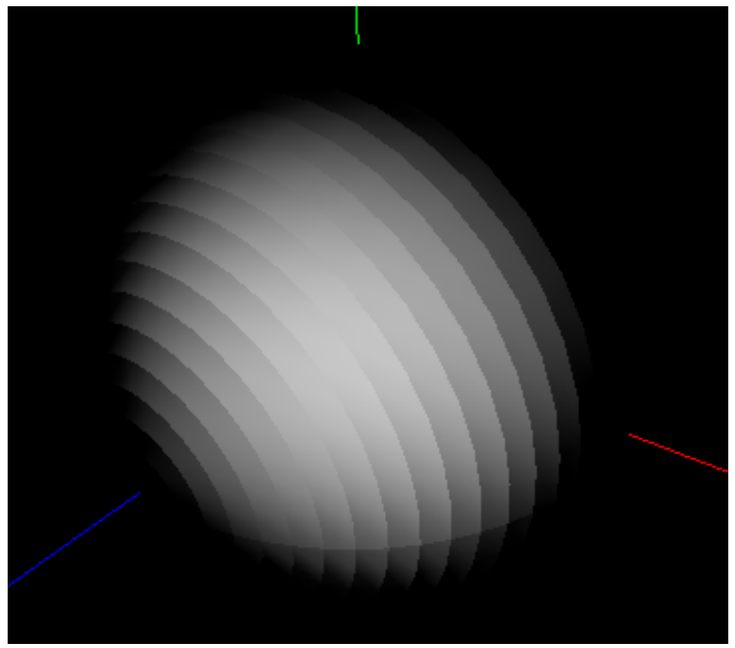
ARLAB



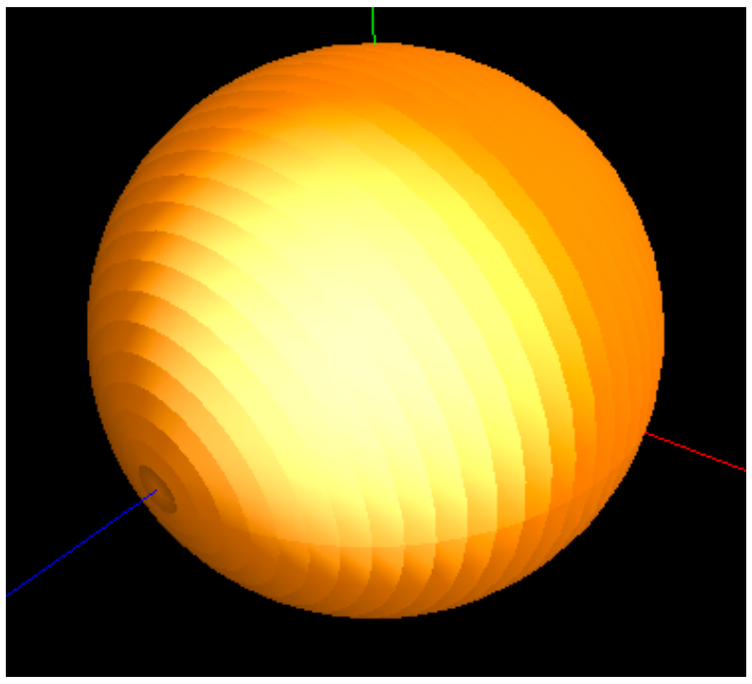
Ambient



Diffuse



Specular

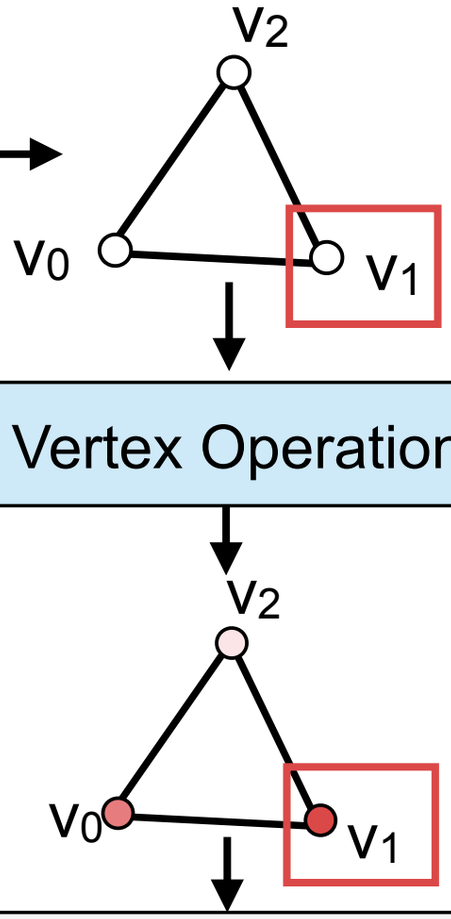


All

Rendering: from model to pixel

ARLAB

Display List of a 3D model



- Calculate vertex colors
- Primitive Assembly
- Shading: fills the primitive (e.g., polygon, quad, etc.) with color

Each primitive (point, line, polygon, quad etc.) is processed individually.

Image data in Frame Buffer

Shader Code

```
static const string vs_string_GLSphere_410 =
"#version 410 core                                                 \n"
"                                                                  \n"
"uniform mat4 projectionMatrixBox;                                 \n"
"uniform mat4 viewMatrixBox;                                       \n"
"uniform mat4 modelMatrixBox;                                      \n"
"uniform vec3 diffuse_color;                                       \n"
"uniform vec3 ambient_color;                                       \n"
"uniform vec3 specular_color;                                      \n"
"uniform vec3 light_position;                                      \n"
"uniform float diffuse_intensity;                                  \n"
"uniform float ambient_intensity;                                  \n"
"uniform float specular_intensity;                                 \n"
"uniform float shininess;                                          \n"
"in vec3 in_Position;                                              \n"
"in vec3 in_Normal;                                                \n"
"in vec3 in_Color;                                                 \n"
"out vec3 pass_Color;                                              \n"
"                                                                  \n"
"                                                                  \n"
"                                                                  \n"
"                                                                  \n"
"void main(void)                                                   \n"
"{                                                                 \n"
"    vec3 normal = normalize(in_Normal);                                                           \n"
"    vec4 transformedNormal =  normalize(transpose(inverse(modelMatrixBox)) * vec4( normal, 1.0 ));  \n"
"    vec4 surfacePostion = viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);               \n"
"                                                                                                  \n"
"    vec4 surface_to_light =   normalize( vec4(light_position,1.0) -  surfacePostion );            \n"
"                                                                                                  \n"
"    // Diffuse color                                                                              \n"
"    float diffuse_coefficient = max( dot(transformedNormal, surface_to_light), 0.0);              \n"
"    vec3 out_diffuse_color = diffuse_color  * diffuse_coefficient * diffuse_intensity;            \n"
"                                                                                                  \n"
"    // Ambient color                                                                              \n"
"    vec3 out_ambient_color = vec3(ambient_color) * ambient_intensity;                             \n"
"                                                                                                  \n"
"    // Specular color                                                                             \n"
"    vec3 incidenceVector = -surface_to_light.xyz;                                                 \n"
"    vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);                      \n"
"    vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2]); \n"
"    vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);                        \n"
"    float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);                           \n"
"    float specular_coefficient = pow(cosAngle, shininess);                                        \n"
"    vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;         \n"
"                                                                                                  \n"
"    gl_Position = projectionMatrixBox * viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);  \n"
"                                                                                                  \n"
"    pass_Color = vec3(out_diffuse_color + out_ambient_color + out_specular_color);                \n"
"}                                                                                                 \n";
```

Shader Code

```
static const string vs_string_GLSphere_410 =
"#version 410 core                                                 \n"
"                                                                  \n"
"uniform mat4 projectionMatrixBox;
"uniform mat4 viewMatrixBox;
"uniform mat4 modelMatrixBox;
"uniform vec3 diffuse_color;
"uniform vec3 ambient_color;
"uniform vec3 specular_color;                                        \n"
"uniform vec3 light_position;                                         \n"
"uniform float diffuse_intensity;                                        \n"
"uniform float ambient_intensity;                                        \n"
"uniform float specular_intensity;                                        \n"
"uniform float shininess;                                         \n"
"in vec3 in_Position;                                              \n"
"in vec3 in_Normal;                                                \n"
"in vec3 in_Color;                                                \n"
"out vec3 pass_Color;                                             \n"
"                                                                 \n"
"                                                                 \n"
"                                                                 \n"
"void main(void)                                                   \n"
"{                                                                 \n"
"    vec3 normal = normalize(in_Normal);                                                \n"
"    vec4 transformedNormal =  normalize(transpose(inverse(modelMatrixBox)) * vec4( normal, 1.0 ));                                                \n"
"    vec4 surfacePostion = viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);                                                \n"
"                                                                  \n"
"    vec4 surface_to_light =   normalize( vec4(light_position,1.0) -  surfacePostion );                                                \n"
"                                                                  \n"
"    // Diffuse color                                                \n"
"    float diffuse_coefficient = max( dot(transformedNormal, surface_to_light), 0.0);                                                \n"
"    vec3 out_diffuse_color = diffuse_color  * diffuse_coefficient * diffuse_intensity;                                                \n"
"                                                                  \n"
"    // Ambient color                                                \n"
"    vec3 out_ambient_color = vec3(ambient_color) * ambient_intensity;                                                \n"
"                                                                  \n"
"    // Specular color                                                \n"
"    vec3 incidenceVector = -surface_to_light.xyz;                                                \n"
"    vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);                                                \n"
"    vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2]);                                                \n"
"    vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);                                                \n"
"    float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);                                                \n"
"    float specular_coefficient = pow(cosAngle, shininess);                                                \n"
"    vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;                                                \n"
"                                                                  \n"
"    gl_Position = projectionMatrixBox * viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);                                                \n"
"                                                                  \n"
"    pass_Color = vec3(out_diffuse_color + out_ambient_color + out_specular_color);                                                \n"
"}                                                                 \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n"
                                                                   \n";
```

incoming and outgoing variables

Vertex shader code in GLSphere.cpp

Vector calculations

Diffuse reflections

Ambient reflections

Specular reflections

All

Shader Code

```
static const string vs_string_GLSphere_410 =
"#version 410 core                                                 \n"
"                                                                  \n"
"uniform mat4 projectionMatrixBox;                                   \n"
"uniform mat4 viewMatrixBox;                                          \n"
"uniform mat4 modelMatrixBox;                                         \n"
"uniform vec3 diffuse_color;                                        \n"
"uniform vec3 ambient_color;                                        \n"
"uniform vec3 specular_color;                                         \n"
"uniform vec3 light_position;                                         \n"
"uniform float diffuse_intensity;                                       \n"
"uniform float ambient_intensity;                                       \n"
"uniform float specular_intensity;                                        \n"
"uniform float shininess;                                          \n"
"in vec3 in_Position;                                               \n"
"in vec3 in_Normal;                                                  \n"
"in vec3 in_Color;                                                 \n"
"out vec3 pass_Color;                                              \n"
"                                                                  \n"
"                                                                  \n"
"                                                                  \n"
"void main(void)                                                   \n"
"{                                                                 \n"
"    vec3 normal = normalize(in_Normal);                           \n"
"    vec4 transformedNormal =  normalize(transpose(inverse(modelMatrixBox)) * vec4( normal, 1.0 ));  \n"
"    vec4 surfacePostion = viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);  \n"
"                                                                  \n"
"    vec4 surface_to_light =   normalize( vec4(light_position,1.0) -  surfacePostion );  \n"
"                                                                  \n"
"    // Diffuse color                                              \n"
"    float diffuse_coefficient = max( dot(transformedNormal, surface_to_light), 0.0);  \n"
"    vec3 out_diffuse_color = diffuse_color  * diffuse_coefficient * diffuse_intensity;  \n"
"                                                                  \n"
"    // Ambient color                                              \n"
"    vec3 out_ambient_color = vec3(ambient_color) * ambient_intensity;  \n"
"                                                                  \n"
"    // Specular color                                             \n"
"    vec3 incidenceVector = -surface_to_light.xyz;                 \n"
"    vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);  \n"
"    vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2]);  \n"
"    vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);  \n"
"    float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);  \n"
"    float specular_coefficient = pow(cosAngle, shininess);        \n"
"    vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;  \n"
"                                                                  \n"
"    gl_Position = projectionMatrixBox * viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);  \n"
"                                                                  \n"
"    pass_Color = vec3(out_diffuse_color + out_ambient_color + out_specular_color);  \n"
"}                                                                 \n";
```

Vertex shader code in GLSphere.cpp

Vector calculations

Vector Calculations

Goal is to prepare the vectors we need for the **per vertex light** calculation

```
vec3 normal = normalize(in_Normal);
vec4 transformedNormal =  normalize(transpose(inverse(modelMatrixBox)) * vec4( normal, 1.0 ));
vec4 surfacePostion = viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);
vec4 surface_to_light =   normalize( vec4(light_position,1.0) -  surfacePostion );
```

1. Normalize the normal vector

$$\mathbf{v_{norm}} = \frac{\mathbf{v}}{|\mathbf{v}|}$$

This has no effect if the vector is already normalized

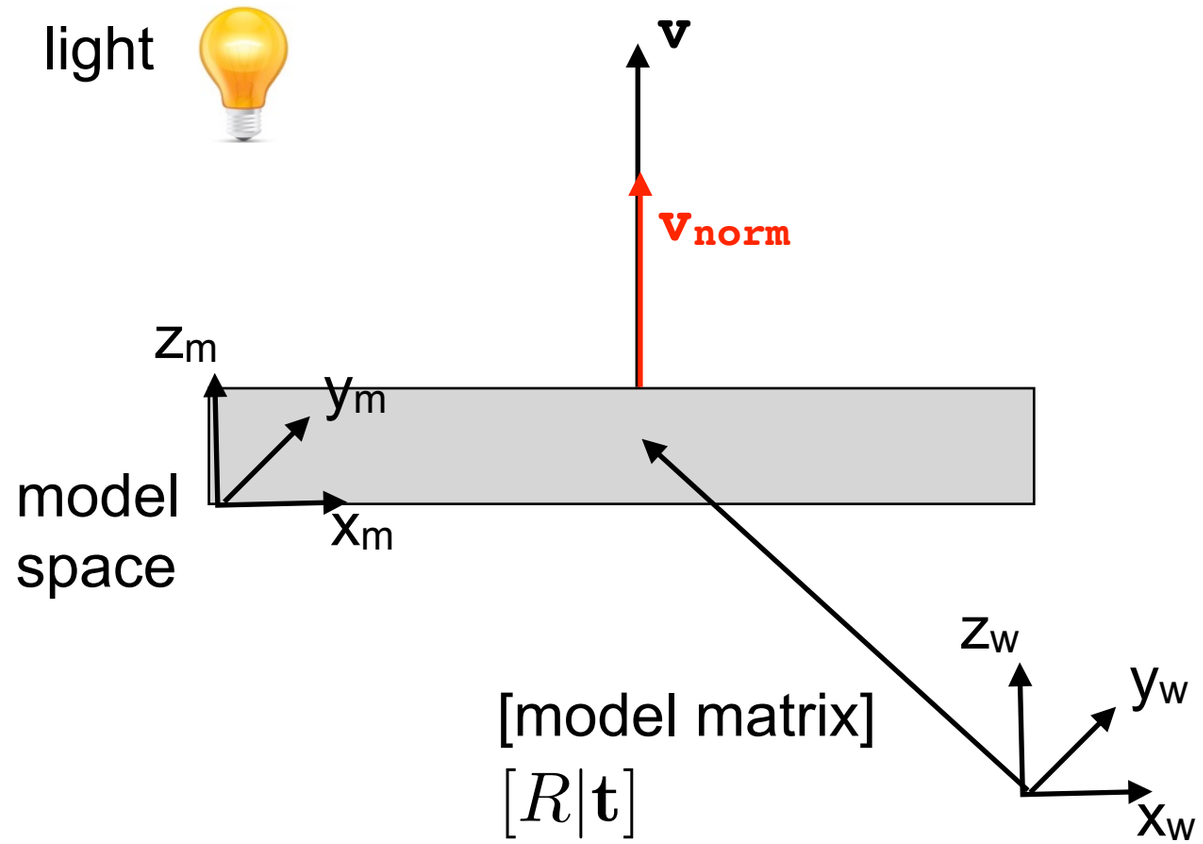
2. Transform the normal vector

Apply the same transformation, which you apply to the vertex, to the normal.

3. Transform all point to world space

We define them in model space

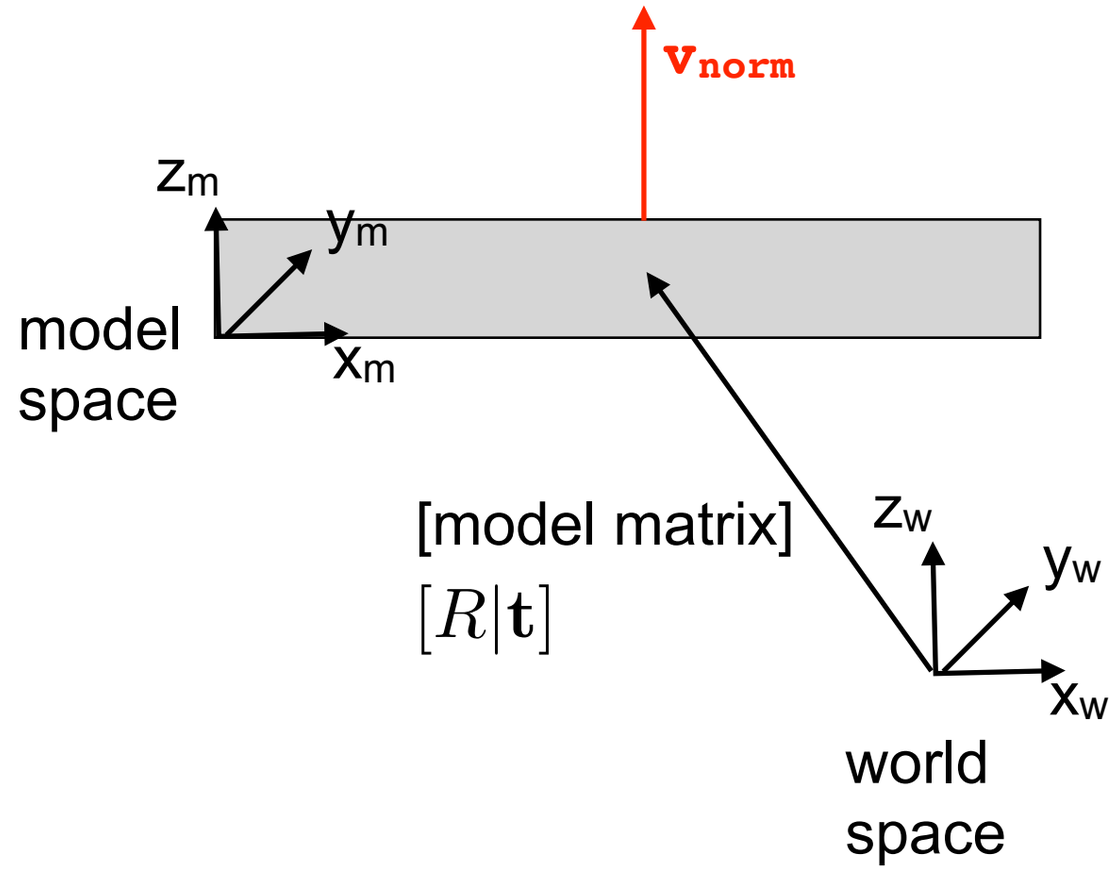
4. Calculate the vector between camera and light



Matrix Transformation Of Normals

ARLAB

- Normals are provided in model space; they are relative to the vertices of the object before a transformations is applied.
- The vector from the surface to the light needs to be calculated in world space.
- If we transform the object, we also have to transform the normals.
- **The problem is that normals are not coordinates, they are unit vectors representing directions.**
- Rotation works, because the rotating a unit vector results in another unit vector,
- Scaling or translating a normal will result in an incorrect normal.



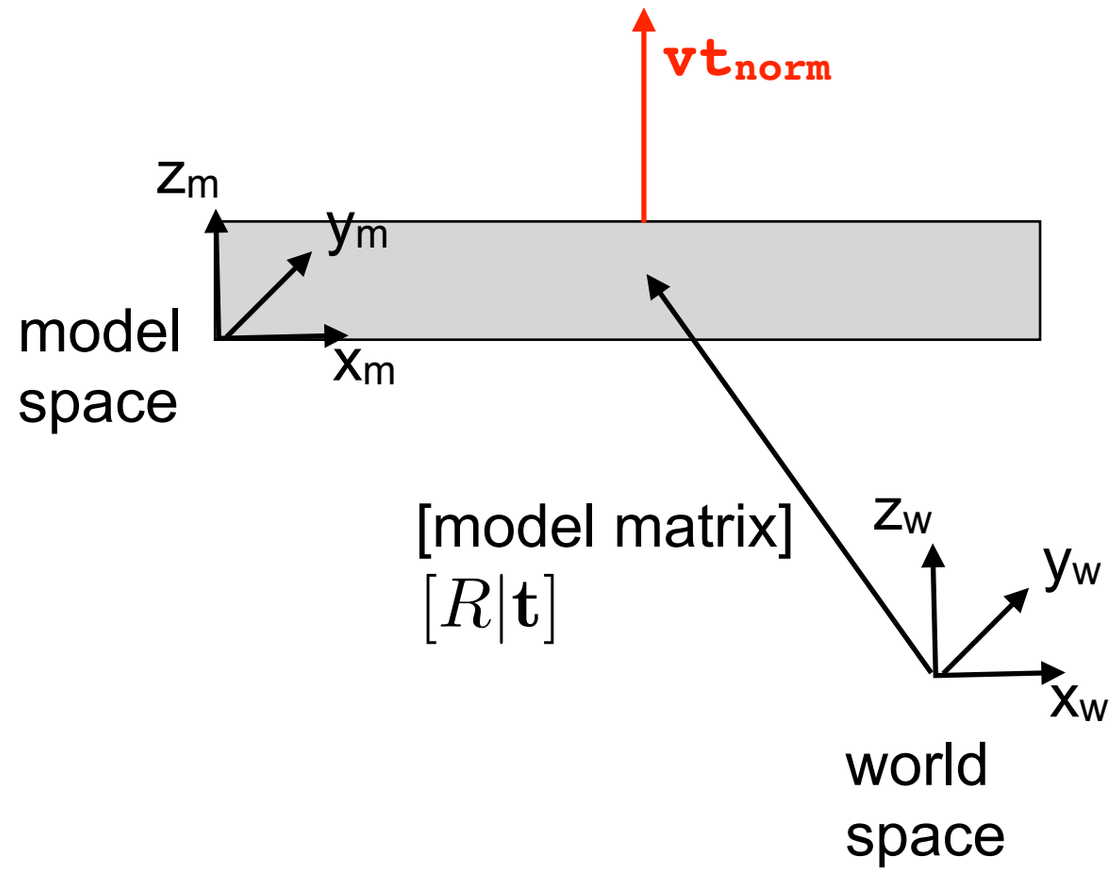
Matrix Transformation Of Normals

- We remove the scaling by transposing and inverting the matrix.
- We also treat the vector like a regular point.
- And we have to normalize the vector again

$$\mathbf{vt_{norm}} = ([R|\mathbf{t}]^{-1})^T \cdot \mathbf{v_{norm}}$$

$$\mathbf{vt_{norm}} = ||\mathbf{vt_{norm}}||_{L2}$$

```
vec4 transformedNormal =
normalize(transpose(inverse(modelMatrixBox))
* vec4( normal, 1.0 ));
```

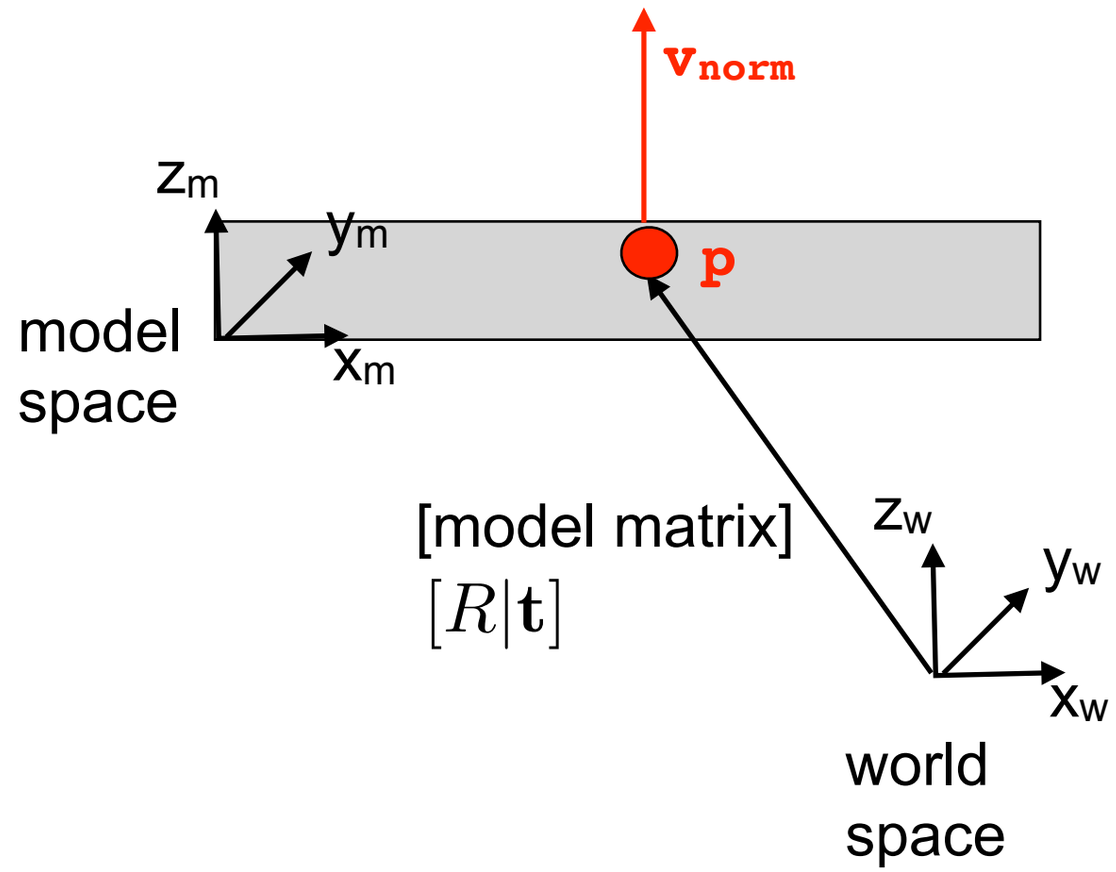


Transform all Points to World Space

ARLAB

- We must move all the model vertex points to world coordinate space.

$$\mathbf{p_w} = [R|\mathbf{t}] \cdot \mathbf{p}$$



Calculate the vector between camera and light source

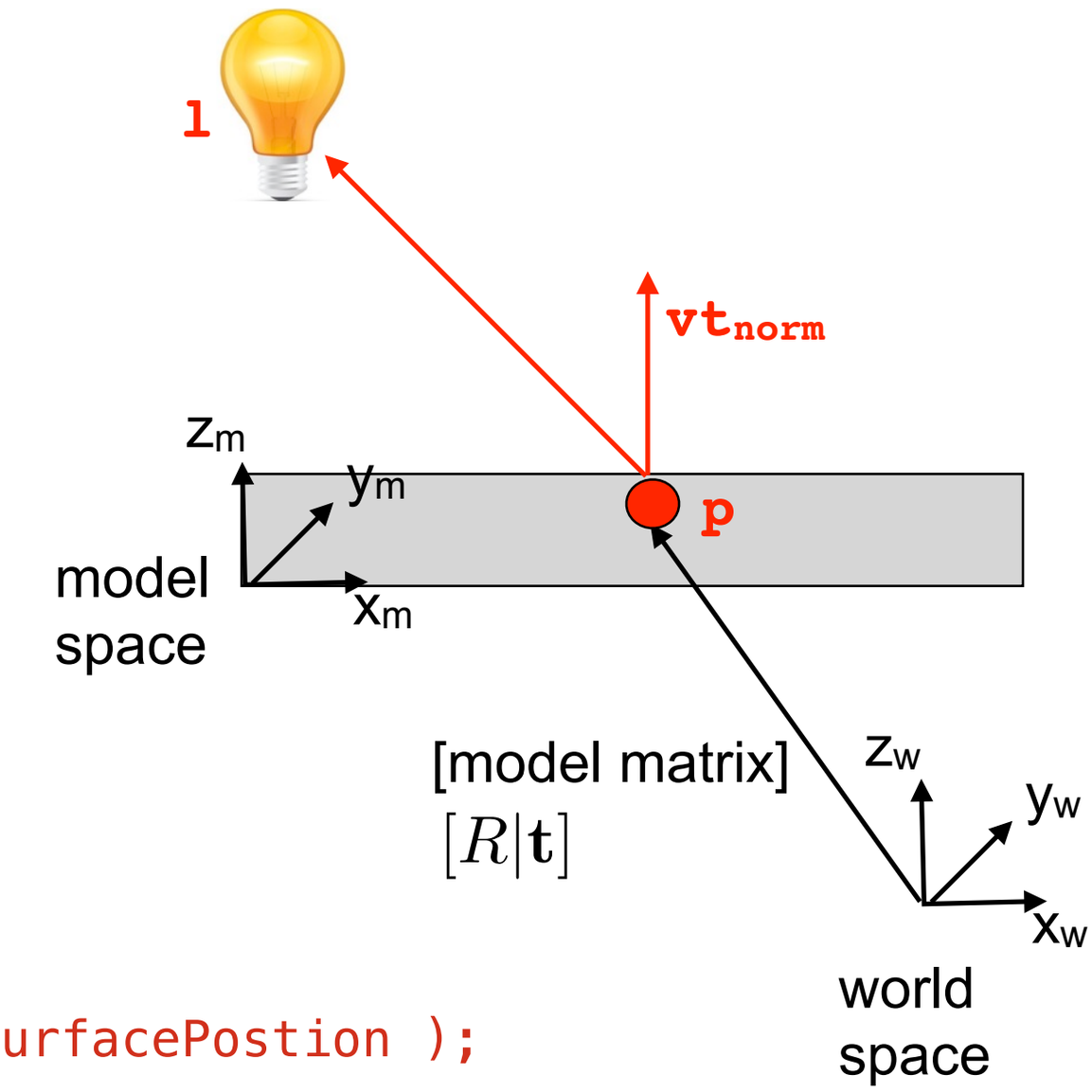
ARLAB

- We must move all the model vertex points to world coordinate space.

$$\mathbf{p_w} = [R|\mathbf{t}] \cdot \mathbf{p}$$

- And we need the vector between the light source and the surface point.
  Note, l, is the position in world coordinates.

$$\mathbf{light} = \mathbf{l} - \mathbf{p}$$




```
vec4 surface_to_light =
normalize( vec4(light_position,1.0) -  surfacePostion );
```

Shader Code

```
static const string vs_string_GLSphere_410 =
"#version 410 core                                                 \n"
"                                                                  \n"
"uniform mat4 projectionMatrixBox;                                   \n"
"uniform mat4 viewMatrixBox;                                          \n"
"uniform mat4 modelMatrixBox;                                         \n"
"uniform vec3 diffuse_color;                                        \n"
"uniform vec3 ambient_color;                                        \n"
"uniform vec3 specular_color;                                         \n"
"uniform vec3 light_position;                                         \n"
"uniform float diffuse_intensity;                                       \n"
"uniform float ambient_intensity;                                       \n"
"uniform float specular_intensity;                                        \n"
"uniform float shininess;                                          \n"
"in vec3 in_Position;                                               \n"
"in vec3 in_Normal;                                                  \n"
"in vec3 in_Color;                                                 \n"
"out vec3 pass_Color;                                              \n"
"                                                                  \n"
"                                                                  \n"
"                                                                  \n"
"void main(void)                                                   \n"
"{                                                                 \n"
"    vec3 normal = normalize(in_Normal);                                 \n"
"    vec4 transformedNormal =  normalize(transpose(inverse(modelMatrixBox)) * vec4( normal, 1.0 ));  \n"
"    vec4 surfacePostion = viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);                  \n"
"                                                                       \n"
"    vec4 surface_to_light =   normalize( vec4(light_position,1.0) -  surfacePostion );             \n"
"                                                                       \n"
"    // Diffuse color                                                    \n"
"    float diffuse_coefficient = max( dot(transformedNormal, surface_to_light), 0.0);                \n"
"    vec3 out_diffuse_color = diffuse_color  * diffuse_coefficient * diffuse_intensity;              \n"
"                                                                       \n"
"    // Ambient color                                                    \n"
"    vec3 out_ambient_color = vec3(ambient_color) * ambient_intensity;                               \n"
"                                                                       \n"
"    // Specular color                                                   \n"
"    vec3 incidenceVector = -surface_to_light.xyz;                                                   \n"
"    vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);                        \n"
"    vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2]);  \n"
"    vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);                          \n"
"    float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);                             \n"
"    float specular_coefficient = pow(cosAngle, shininess);                                          \n"
"    vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;           \n"
"                                                                       \n"
"    gl_Position = projectionMatrixBox * viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);    \n"
"                                                                       \n"
"    pass_Color = vec3(out_diffuse_color + out_ambient_color + out_specular_color);                  \n"
"}                                                                                                   \n";
```

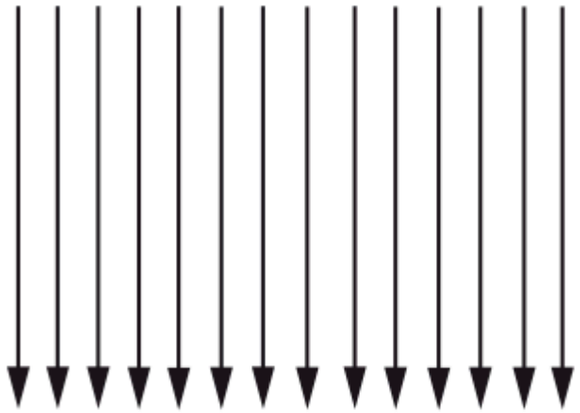
Vertex shader code in GLSphere.cpp

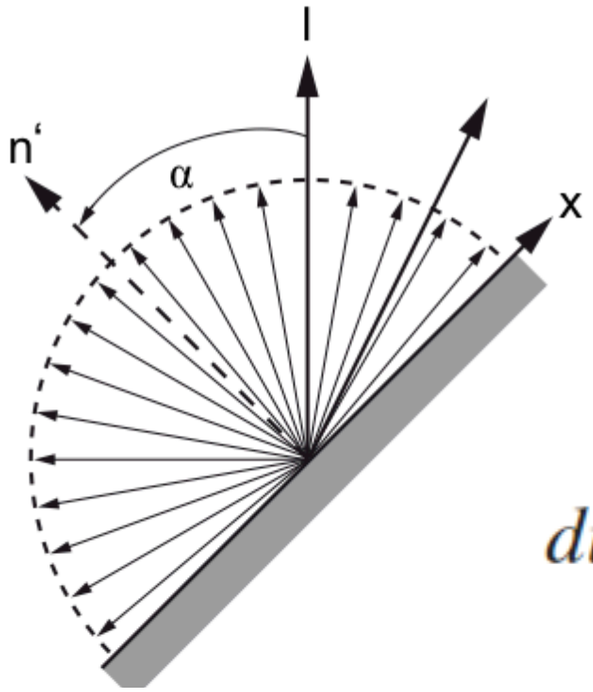
Diffuse reflection

Diffuse Reflection

ARLAB

parallel incoming light

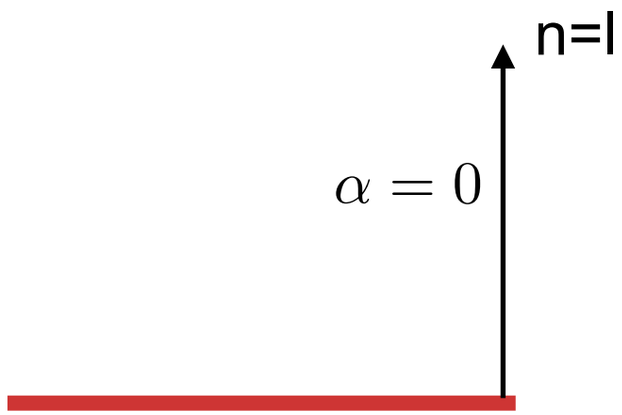


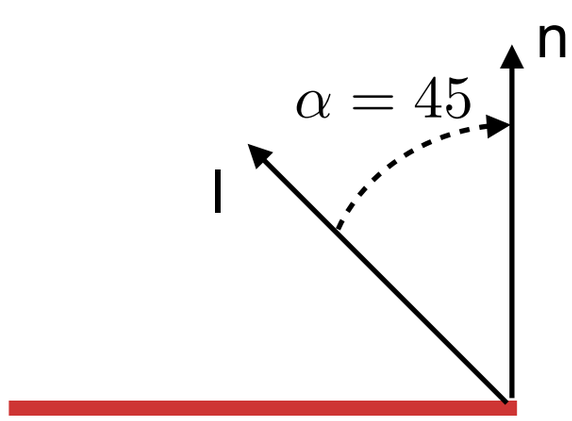


The incoming light I is scattered into all directions.

$$diffus = \max(\boldsymbol{l}\cdot\boldsymbol{n},0)\cdot I_D\cdot C_D$$

$$= \max(\boldsymbol{l}\cdot\boldsymbol{n},0)\cdot\begin{pmatrix} R_{L,D}\cdot R_{C,D} \\ G_{L,D}\cdot G_{C,D} \\ B_{L,D}\cdot B_{C,D} \\ A_{L,D}\cdot A_{C,D} \end{pmatrix}$$





$\boldsymbol{l}\cdot\boldsymbol{n}$:= *dot product*

l: Diffuse light direction vector,
I_D: Diffuse light color
n: Normal,
C_D: Diffuse material color

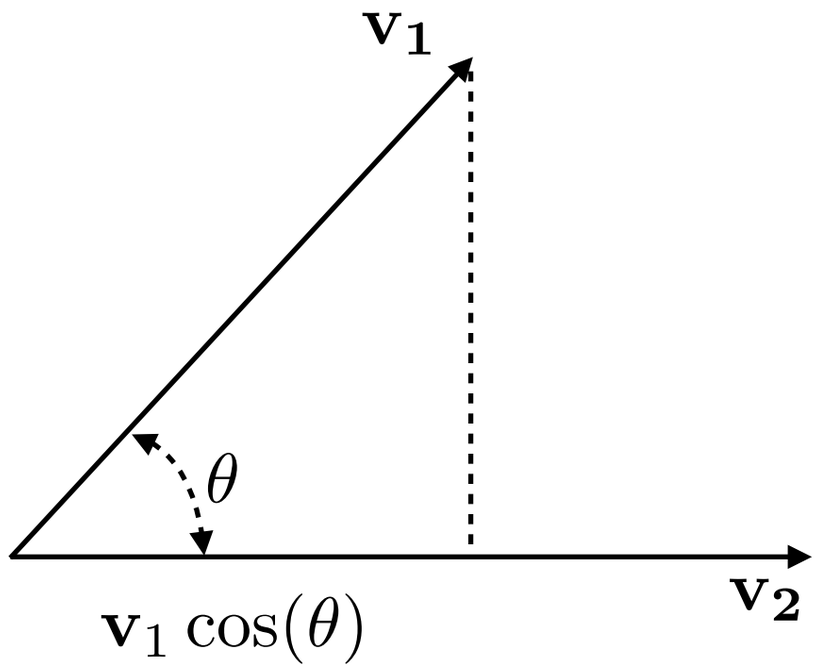
Angle Between Two Vectors: Dot Product

ARLAB

$$\mathbf{v_1} \cdot \mathbf{v_2} = |\mathbf{v_1}||\mathbf{v_2}| \cos(\theta)$$

$$\frac{\mathbf{v_1} \cdot \mathbf{v_2}}{|\mathbf{v_1}||\mathbf{v_2}|} = \cos(\theta)$$

$$\cos^{-1} \frac{\mathbf{v_1} \cdot \mathbf{v_2}}{|\mathbf{v_1}||\mathbf{v_2}|} = \theta$$

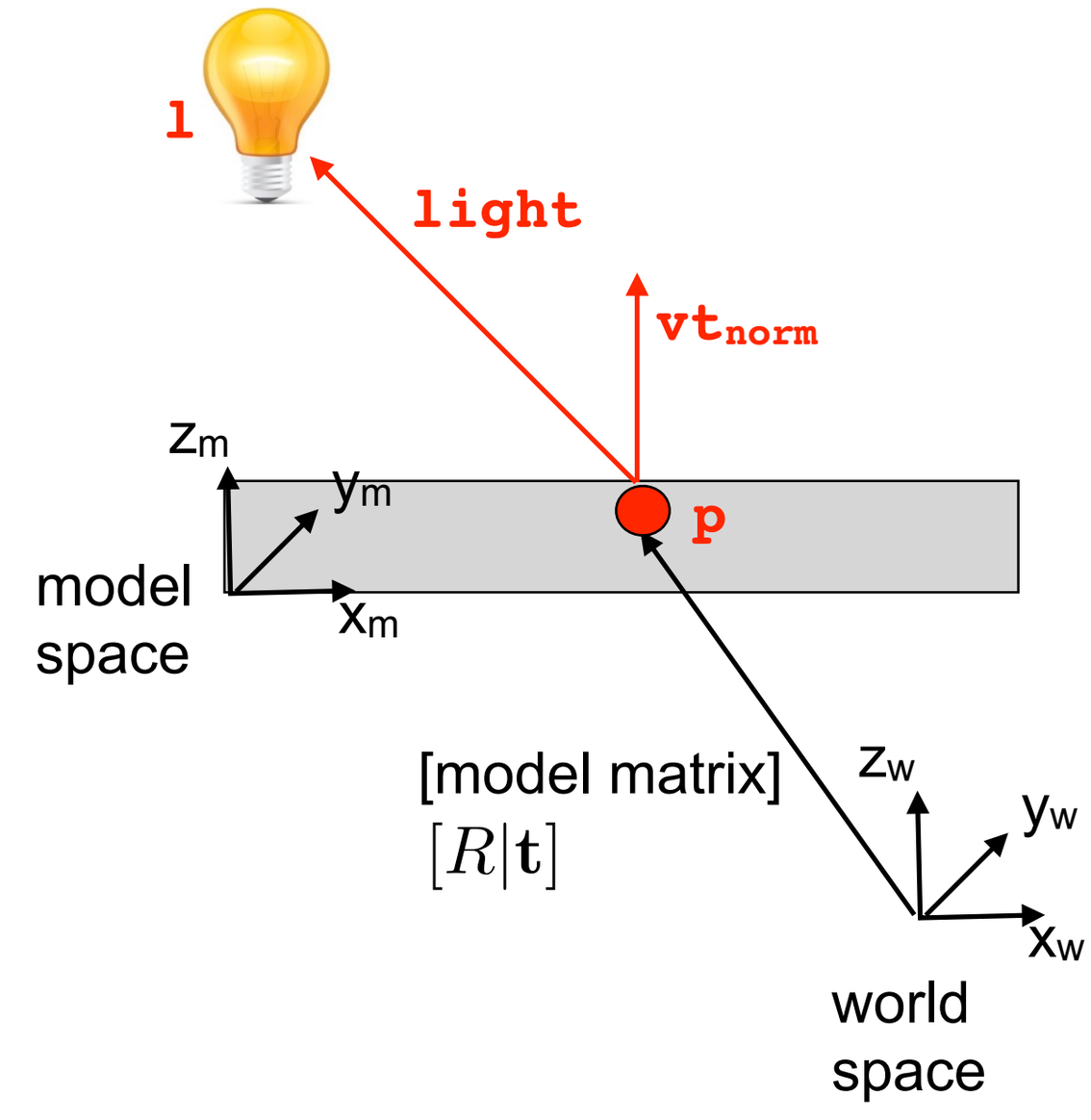


GLSL Code
Diffuse Reflection

ARLAB

```
// Diffuse color
float diffuse_coefficient = max( dot(transformedNormal, surface_to_light), 0.0);
vec3 out_diffuse_color = diffuse_color  * diffuse_coefficient * diffuse_intensity;
```

$$diffuse = \max(\mathbf{vt_{norm}} \cdot \mathbf{light}, 0.0)$$

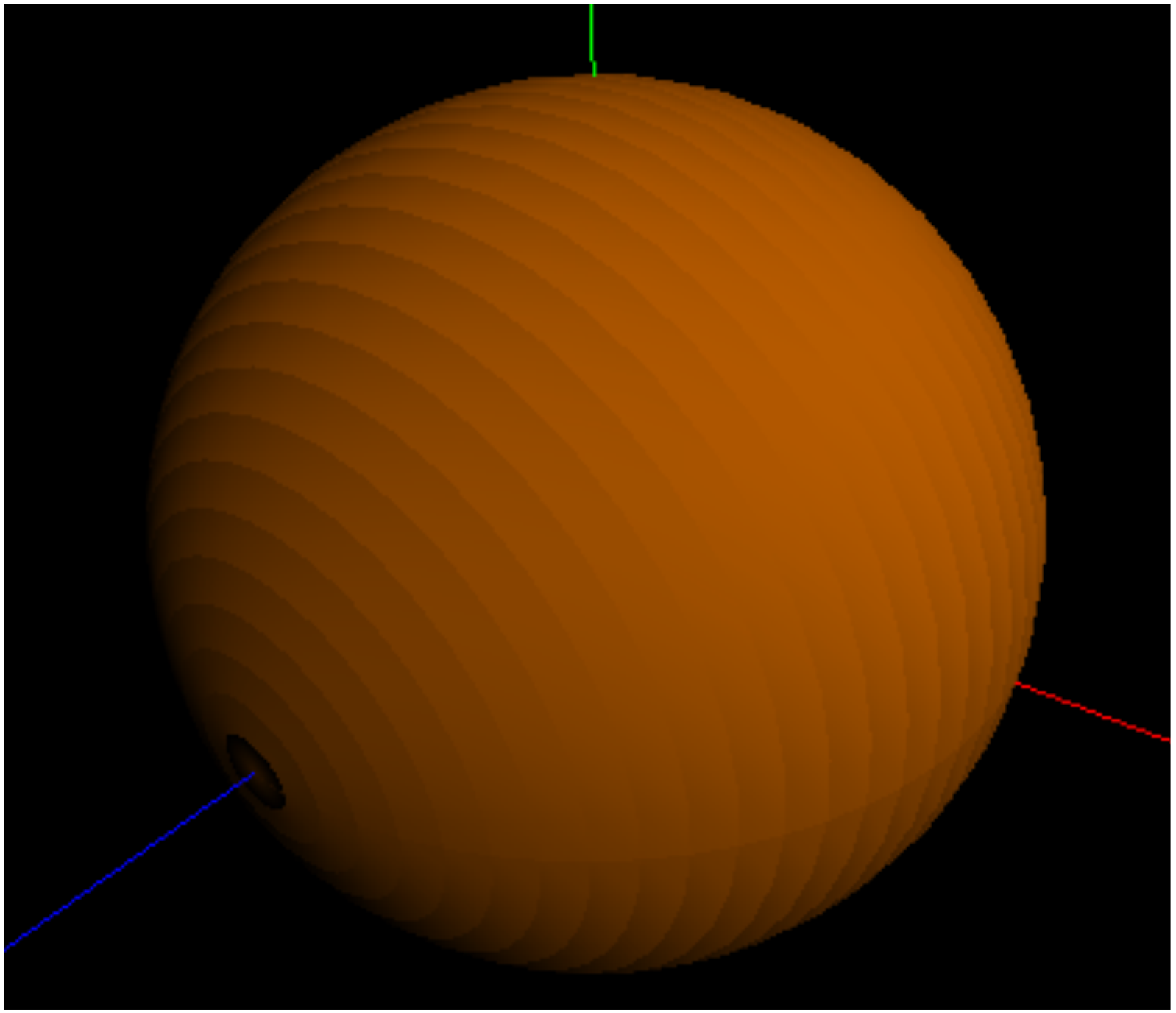


diffuse is the scalar intensity value

`diffuse_coefficient value`
in the program

- diffuse_color: the material color
- diffuse_intensity: the light source intensity

Diffuse Output



Diffuse

Shader Code

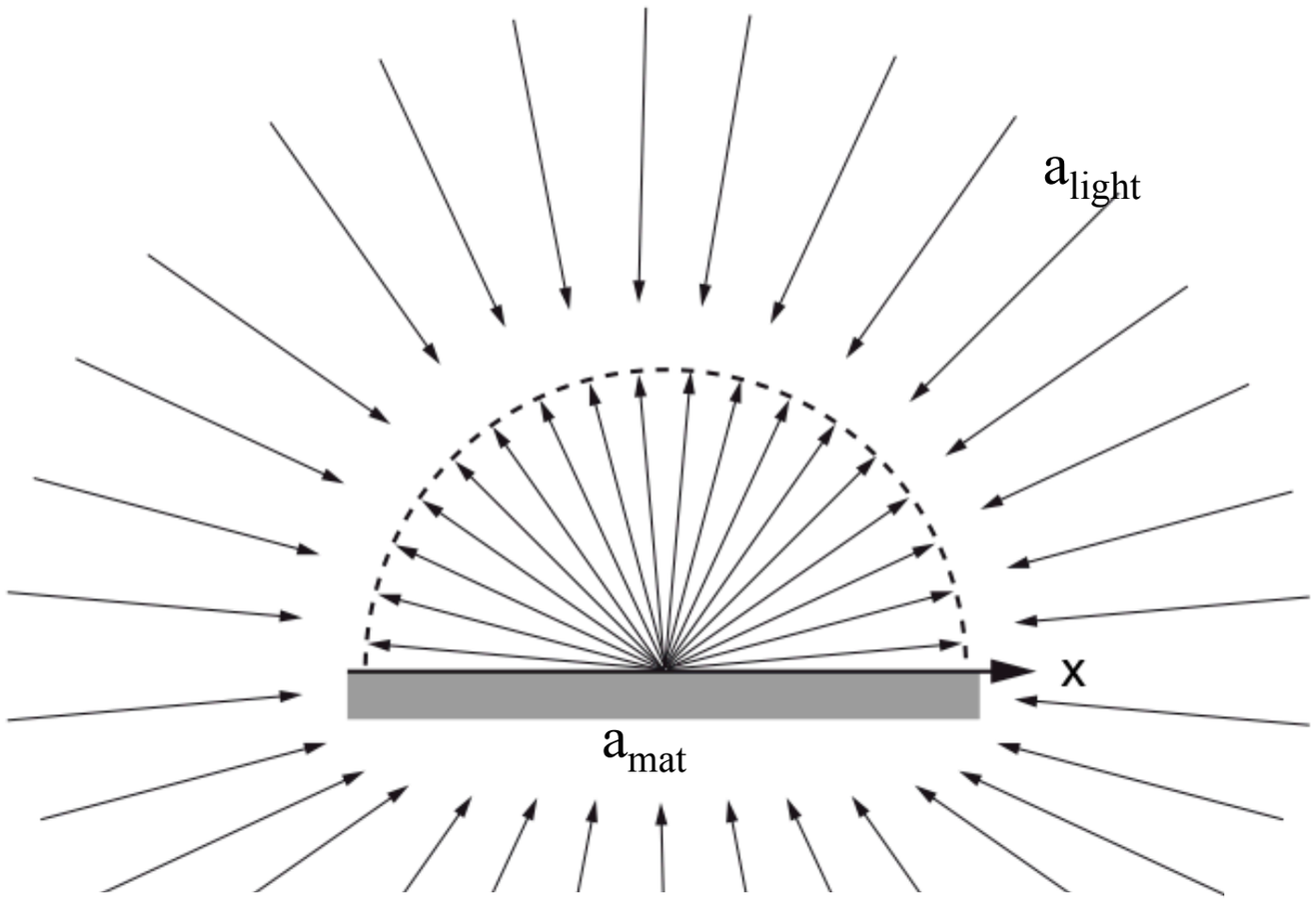
```
static const string vs_string_GLSphere_410 =
"#version 410 core                                                 \n"
"                                                                  \n"
"uniform mat4 projectionMatrixBox;                                   \n"
"uniform mat4 viewMatrixBox;                                          \n"
"uniform mat4 modelMatrixBox;                                         \n"
"uniform vec3 diffuse_color;                                        \n"
"uniform vec3 ambient_color;                                        \n"
"uniform vec3 specular_color;                                         \n"
"uniform vec3 light_position;                                         \n"
"uniform float diffuse_intensity;                                       \n"
"uniform float ambient_intensity;                                       \n"
"uniform float specular_intensity;                                        \n"
"uniform float shininess;                                          \n"
"in vec3 in_Position;                                               \n"
"in vec3 in_Normal;                                                  \n"
"in vec3 in_Color;                                                 \n"
"out vec3 pass_Color;                                              \n"
"                                                                  \n"
"                                                                  \n"
"                                                                  \n"
"void main(void)                                                   \n"
"{                                                                 \n"
"    vec3 normal = normalize(in_Normal);                                \n"
"    vec4 transformedNormal =  normalize(transpose(inverse(modelMatrixBox)) * vec4( normal, 1.0 ));  \n"
"    vec4 surfacePostion = viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);  \n"
"                                                                  \n"
"    vec4 surface_to_light =   normalize( vec4(light_position,1.0) -  surfacePostion );  \n"
"                                                                  \n"
"    // Diffuse color                                                \n"
"    float diffuse_coefficient = max( dot(transformedNormal, surface_to_light), 0.0);  \n"
"    vec3 out_diffuse_color = diffuse_color  * diffuse_coefficient * diffuse_intensity;  \n"
"                                                                  \n"
"    // Ambient color                                                \n"
"    vec3 out_ambient_color = vec3(ambient_color) * ambient_intensity;  \n"
"                                                                  \n"
"    // Specular color                                               \n"
"    vec3 incidenceVector = -surface_to_light.xyz;                      \n"
"    vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);  \n"
"    vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2]);  \n"
"    vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);  \n"
"    float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);  \n"
"    float specular_coefficient = pow(cosAngle, shininess);             \n"
"    vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;  \n"
"                                                                  \n"
"    gl_Position = projectionMatrixBox * viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);  \n"
"                                                                  \n"
"    pass_Color = vec3(out_diffuse_color + out_ambient_color + out_specular_color);  \n"
"}                                                                 \n";
```

Vertex shader code in GLSphere.cpp

Ambient reflection

Ambient Reflection (1/2)

ARLAB



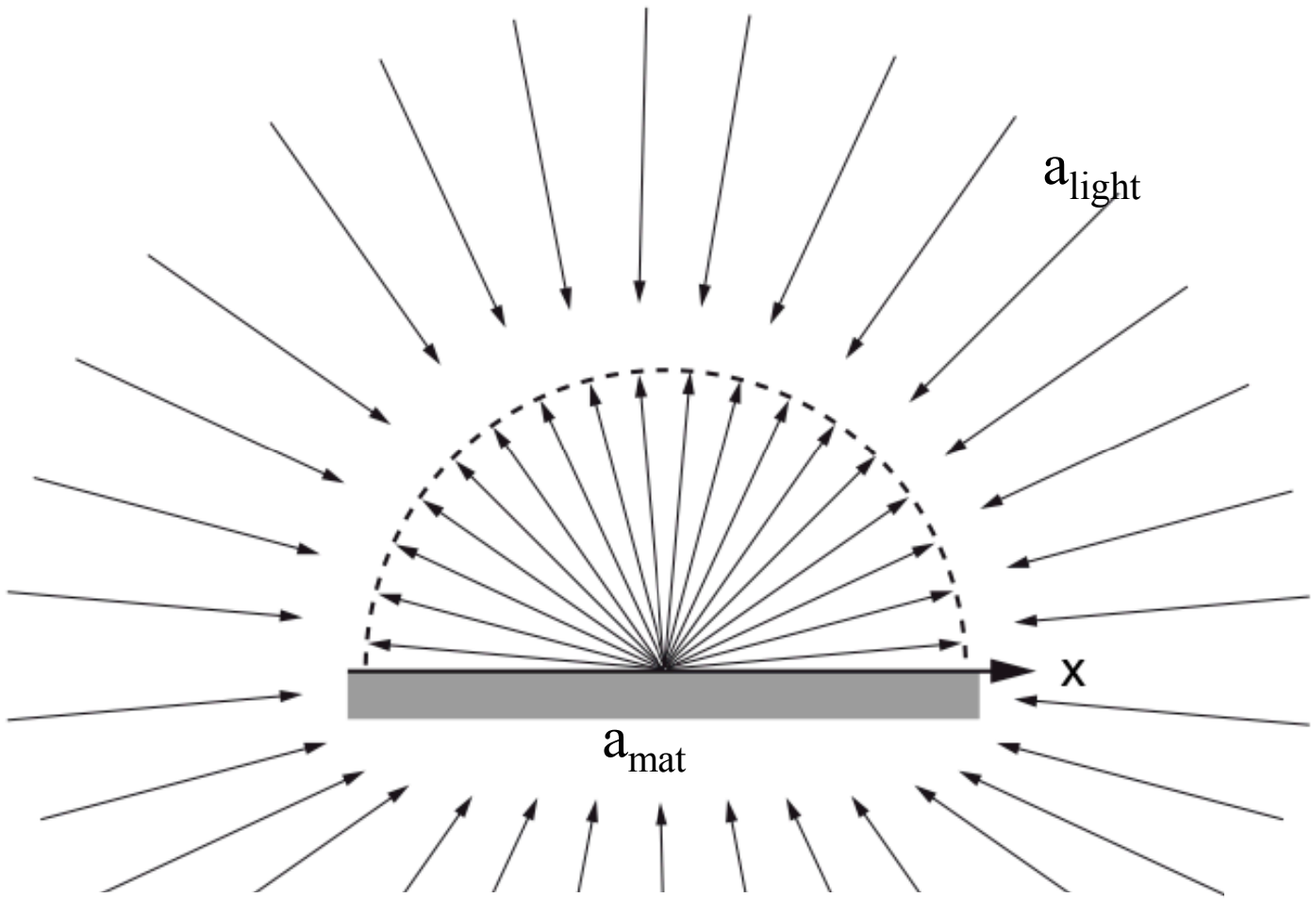
Ambient reflection emulates the effect of omnidirectional light shining on the surface.

A 3D model illuminated only by ambient light and reflect ambient light only, appears plain-colored on the screen.

$$ambient = a_{light} \cdot a_{mat} = \begin{pmatrix} R_{a_{light}} \\ G_{a_{light}} \\ B_{a_{light}} \\ A_{a_{light}} \end{pmatrix} \cdot \begin{pmatrix} R_{a_{mat}} \\ G_{a_{mat}} \\ B_{a_{mat}} \\ A_{a_{mat}} \end{pmatrix}$$

a_{light}: Ambient light color
a_{mat}: Ambient material color

Ambient Reflection



Ambient reflection emulates the effect of omnidirectional light shining on the surface.

A 3D model illuminated only by ambient light and reflect ambient light only, appears plain-colored on the screen.

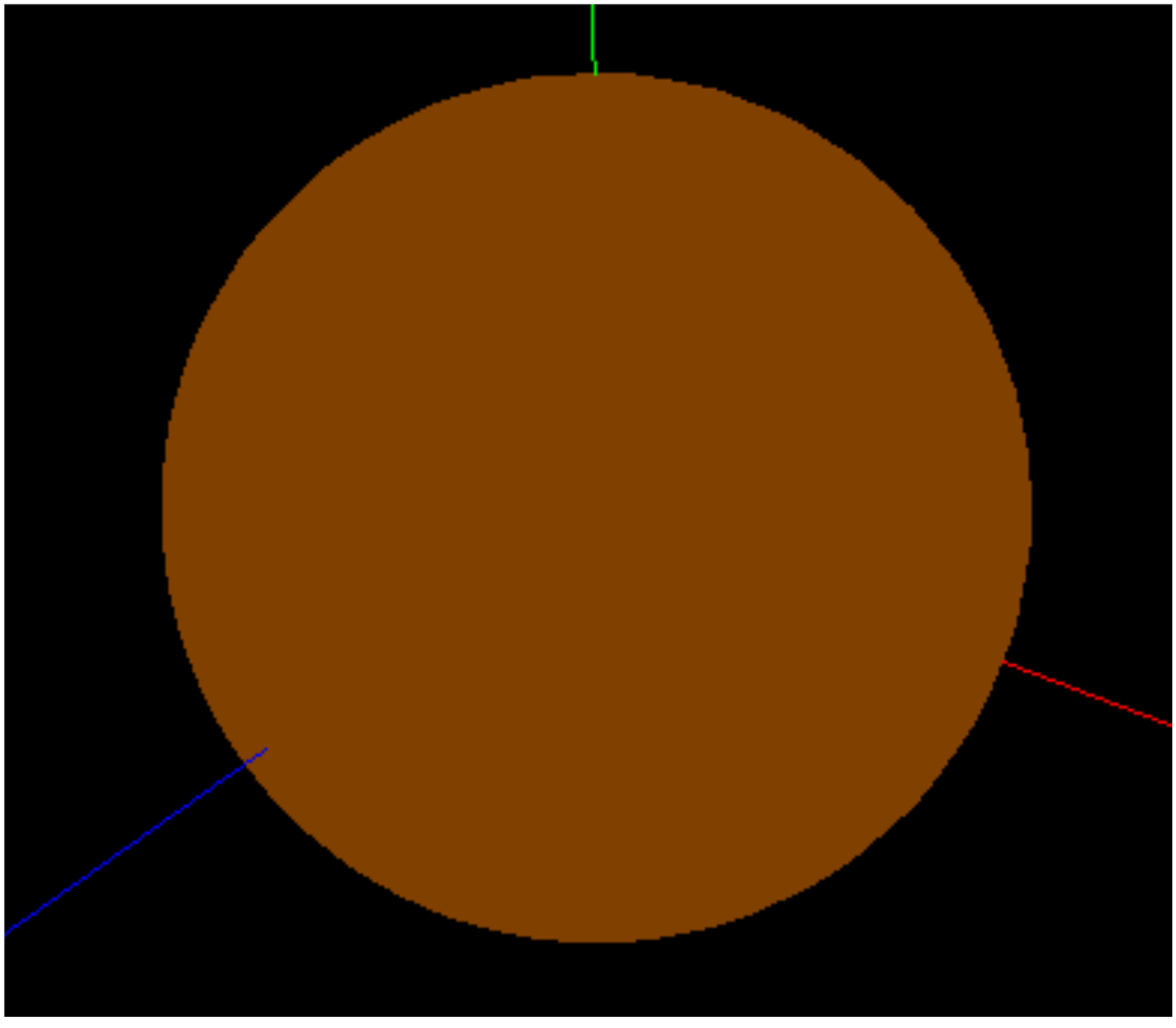
```
// Ambient color
vec3 out_ambient_color = vec3(ambient_color) * ambient_intensity;
```

- ambient_color: the ambient material rgb values
- ambient_intensity: the intensity of the light source

a_{light}: Ambient light color
a_{mat}: Ambient material color

Ambient Output

ARLAB



Ambient

Shader Code

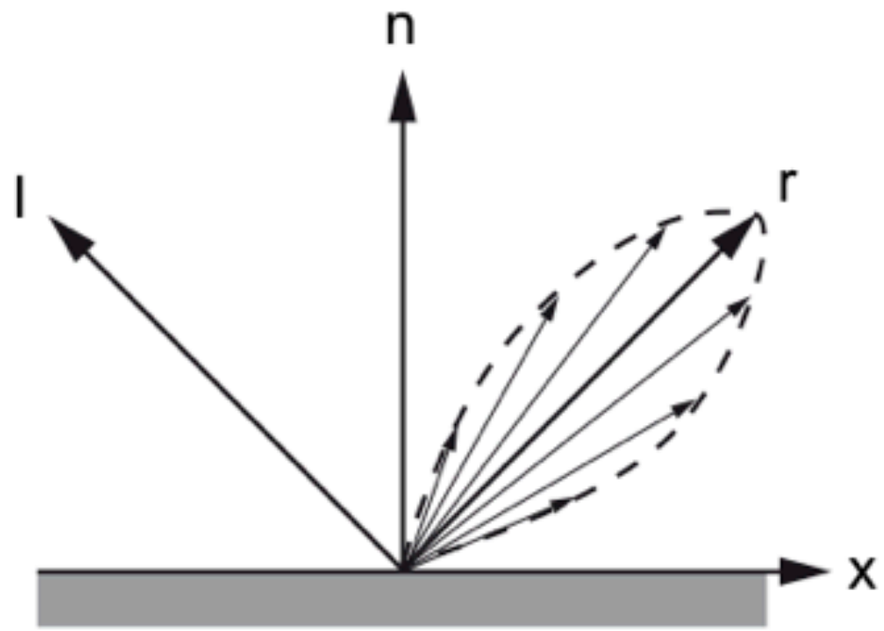
Vertex shader code in GLSphere.cpp

```
static const string vs_string_GLSphere_410 =
"#version 410 core                                                 \n"
"                                                                  \n"
"uniform mat4 projectionMatrixBox;                                   \n"
"uniform mat4 viewMatrixBox;                                          \n"
"uniform mat4 modelMatrixBox;                                         \n"
"uniform vec3 diffuse_color;                                        \n"
"uniform vec3 ambient_color;                                        \n"
"uniform vec3 specular_color;                                         \n"
"uniform vec3 light_position;                                         \n"
"uniform float diffuse_intensity;                                       \n"
"uniform float ambient_intensity;                                       \n"
"uniform float specular_intensity;                                        \n"
"uniform float shininess;                                          \n"
"in vec3 in_Position;                                               \n"
"in vec3 in_Normal;                                                  \n"
"in vec3 in_Color;                                                 \n"
"out vec3 pass_Color;                                              \n"
"                                                                  \n"
"                                                                  \n"
"                                                                  \n"
"void main(void)                                                   \n"
"{                                                                 \n"
"    vec3 normal = normalize(in_Normal);                                                  \n"
"    vec4 transformedNormal =  normalize(transpose(inverse(modelMatrixBox)) * vec4( normal, 1.0 ));  \n"
"    vec4 surfacePostion = viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);       \n"
"                                                                                         \n"
"    vec4 surface_to_light =   normalize( vec4(light_position,1.0) -  surfacePostion );  \n"
"                                                                                         \n"
"    // Diffuse color                                                                     \n"
"    float diffuse_coefficient = max( dot(transformedNormal, surface_to_light), 0.0);     \n"
"    vec3 out_diffuse_color = diffuse_color  * diffuse_coefficient * diffuse_intensity;   \n"
"                                                                                         \n"
"    // Ambient color                                                                     \n"
"    vec3 out_ambient_color = vec3(ambient_color) * ambient_intensity;                    \n"
"                                                                                         \n"
"    // Specular color                                                                    \n"
"    vec3 incidenceVector = -surface_to_light.xyz;                                        \n"
"    vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);             \n"
"    vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2]);  \n"
"    vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);               \n"
"    float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);                  \n"
"    float specular_coefficient = pow(cosAngle, shininess);                               \n"
"    vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;  \n"
"                                                                                         \n"
"    gl_Position = projectionMatrixBox * viewMatrixBox * modelMatrixBox * vec4(in_Position, 1.0);  \n"
"                                                                                         \n"
"    pass_Color = vec3(out_diffuse_color + out_ambient_color + out_specular_color);       \n"
"}                                                                                        \n";
```

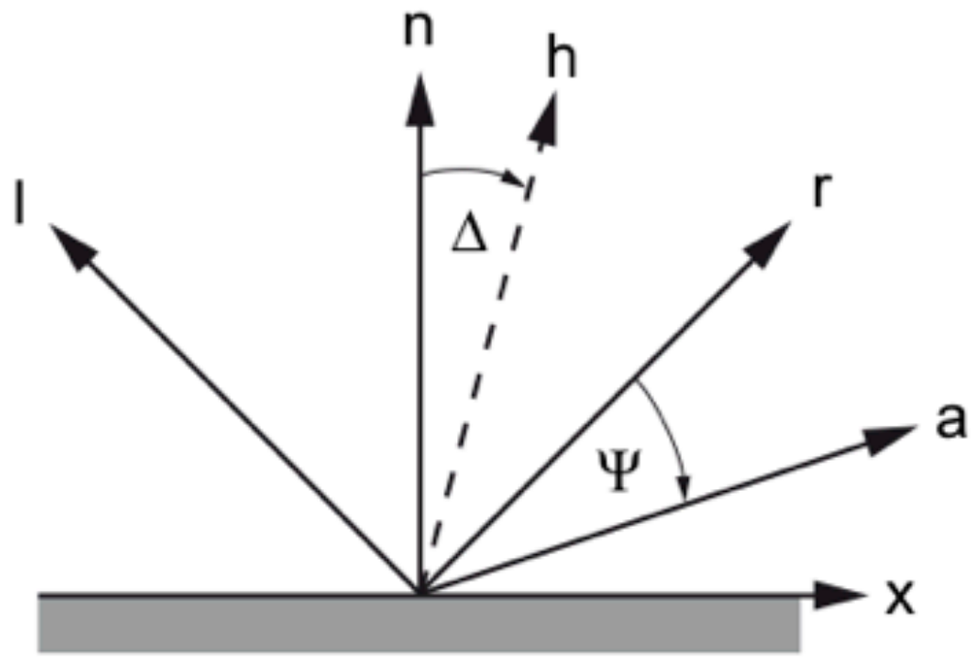
Specular reflection

Specular Reflection

ARLAB



(a)



(b)

The specular reflection component

$$specular = (\max(\boldsymbol{h}\cdot\boldsymbol{n},0)^{S})\cdot I_S\cdot C_S$$

$$= \max(\boldsymbol{h}\cdot\boldsymbol{n},0)^{S}\cdot\begin{pmatrix} R_{L,S}\cdot R_{C,S} \\ G_{L,S}\cdot G_{C,S} \\ B_{L,S}\cdot B_{C,S} \\ A_{L,S}\cdot A_{C,S} \end{pmatrix}$$

half-way vector:

$$\boldsymbol{h} = \frac{(\boldsymbol{l} + \boldsymbol{a})}{|\boldsymbol{l}+\boldsymbol{a}|}$$

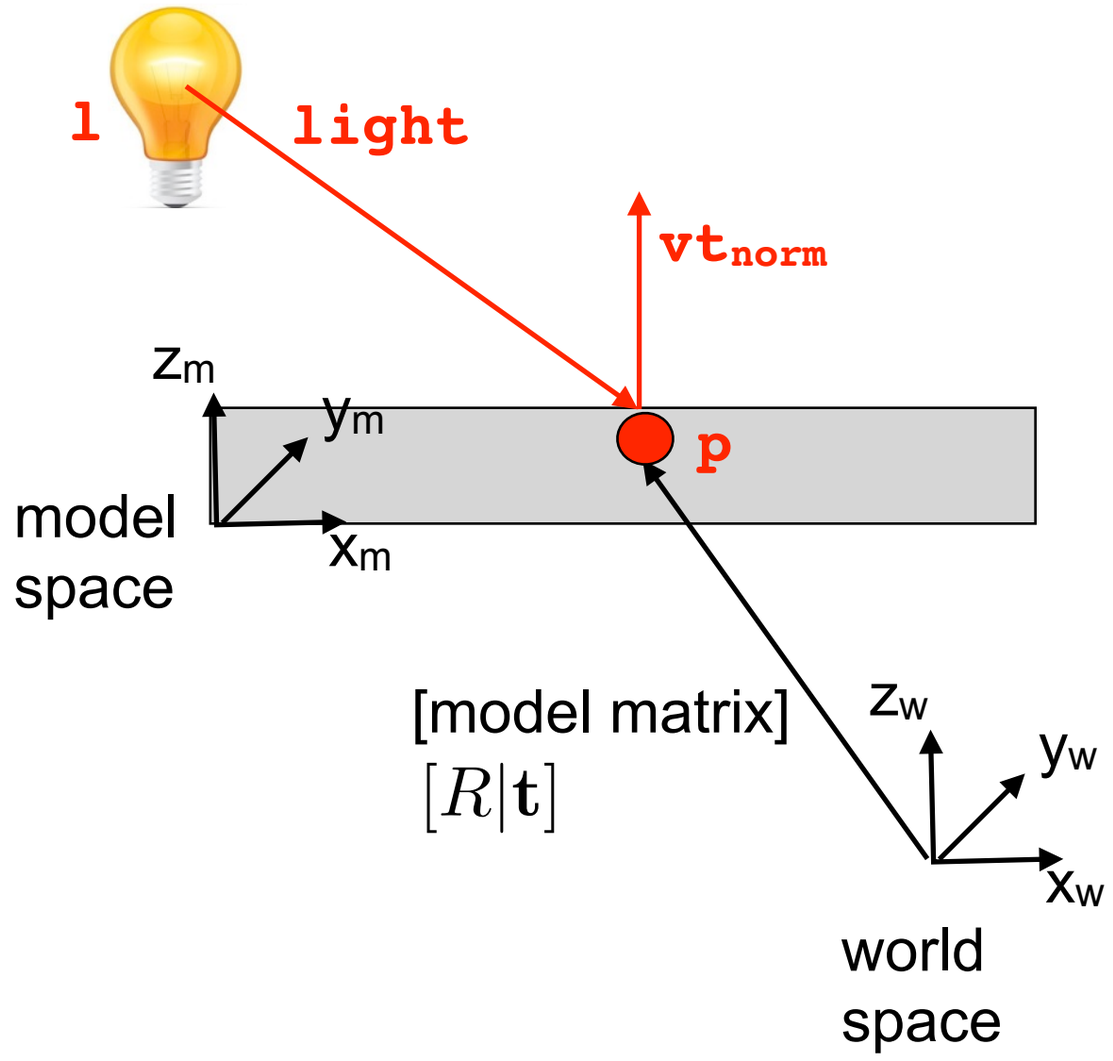
l : Specular light direction vector,
I_S: Specular light color
n: Normal,
a: Eye-point
r: reflection vector
h: Halfway Vector
S: Specular reflection coefficient
C_s:Specular material color

Specular Reflection

ARLAB

```
// Specular color
vec3 incidenceVector = -surface_to_light.xyz;
vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);
vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2])
vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);
float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);
float specular_coefficient = pow(cosAngle, shininess);
vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;
```

- First line: incidenceVector, this the direction from the surface to the light. Note, align the signs!!!!

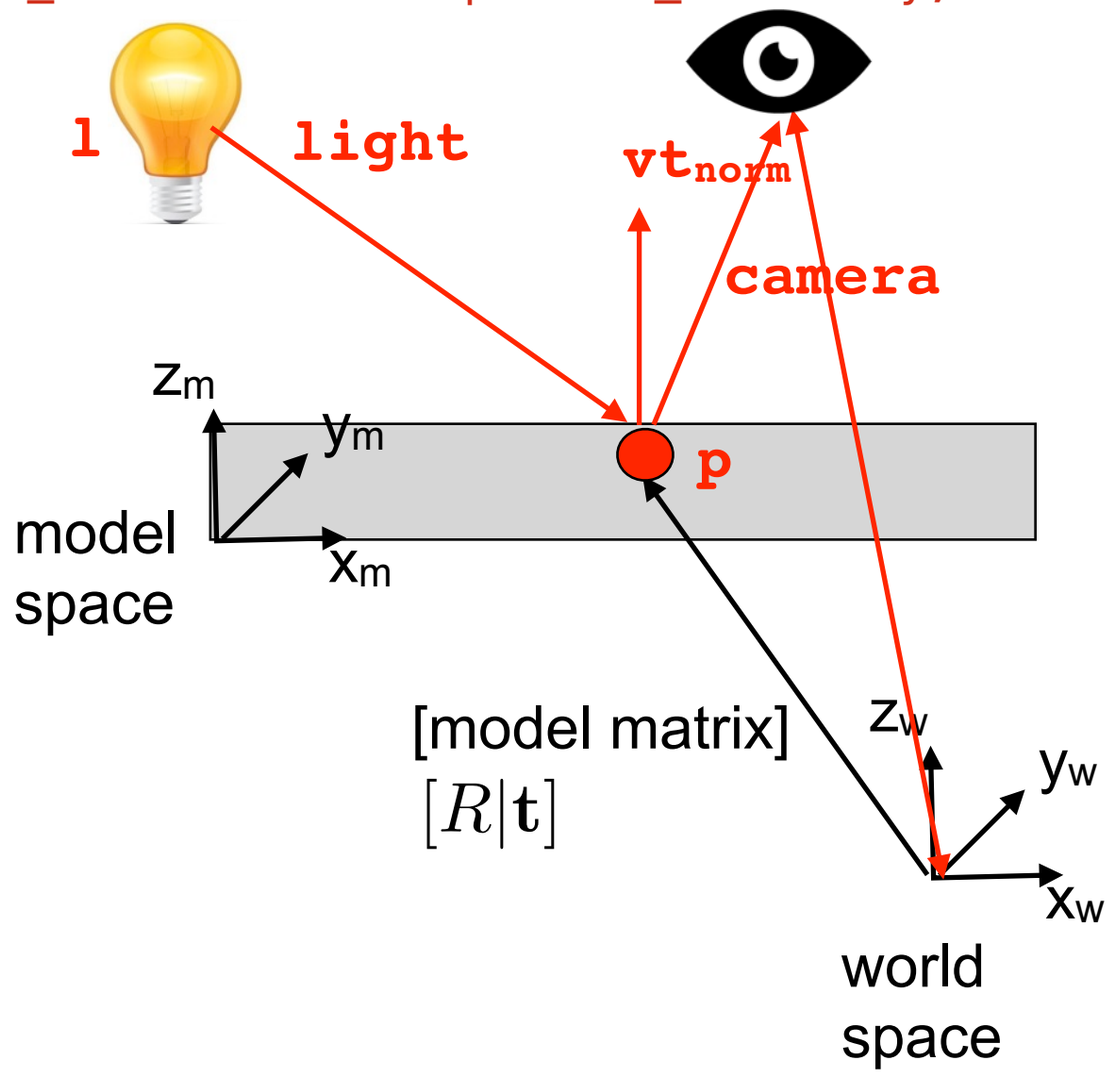


Specular Reflection

ARLAB

```
// Specular color
vec3 incidenceVector = -surface_to_light.xyz;
vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);
vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2])
vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);
float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);
float specular_coefficient = pow(cosAngle, shininess);
vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;
```

- First line: incidenceVector, this the direction from the surface to the light. Note, align the signs!!!!
- Line three and four: calculate the eye vector
- line two: glsl reflect function Calculates the reflection vector

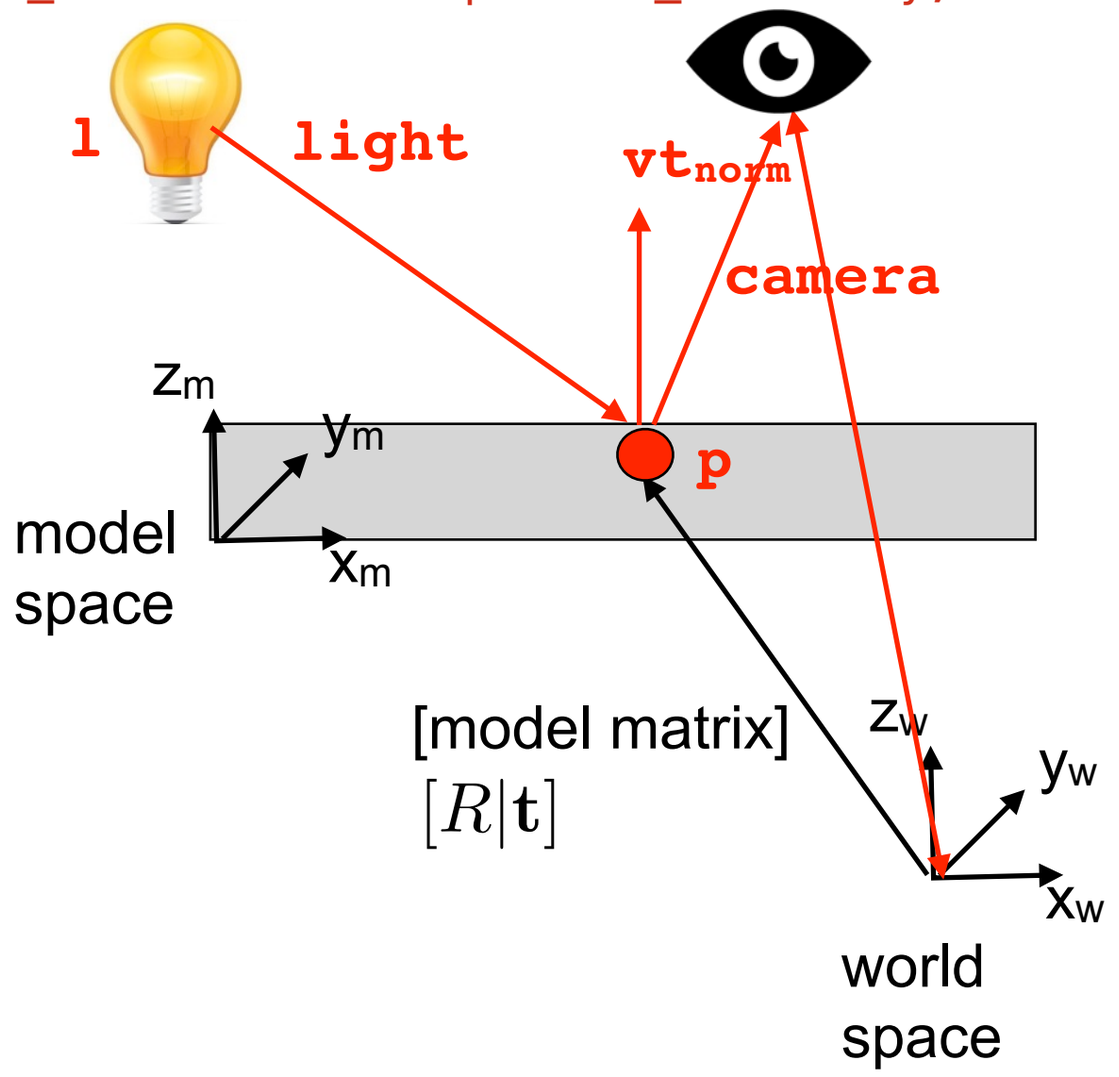


Specular Reflection

ARLAB

```
// Specular color
vec3 incidenceVector = -surface_to_light.xyz;
vec3 reflectionVector = reflect(incidenceVector, transformedNormal.xyz);
vec3 cameraPosition = vec3( -viewMatrixBox[3][0], -viewMatrixBox[3][1], -viewMatrixBox[3][2])
vec3 surfaceToCamera = normalize(cameraPosition - surfacePostion.xyz);
float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);
float specular_coefficient = pow(cosAngle, shininess);
vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;
```

- First line: incidenceVector, this the direction from the surface to the light. Note, align the signs!!!!
- Line three and four: calculate the eye vector
- line two: glsl reflect function Calculates the reflection vector



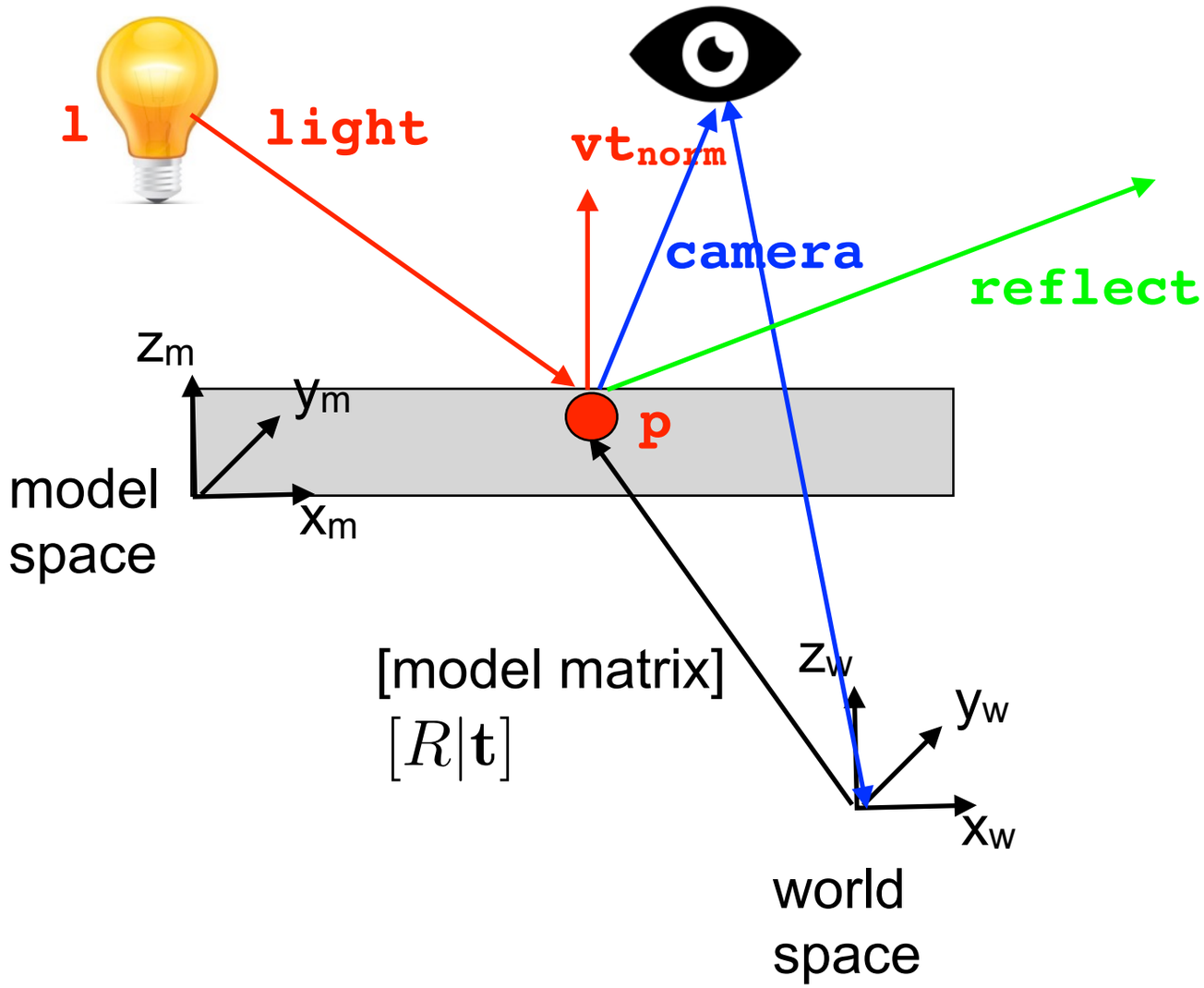
Specular Reflection

We use the GLSL function reflect

```
genType reflect(genType I,
                genType N);
```

Parameters

- I - Specifies the incident vector.
- N - Specifies the normal vector.



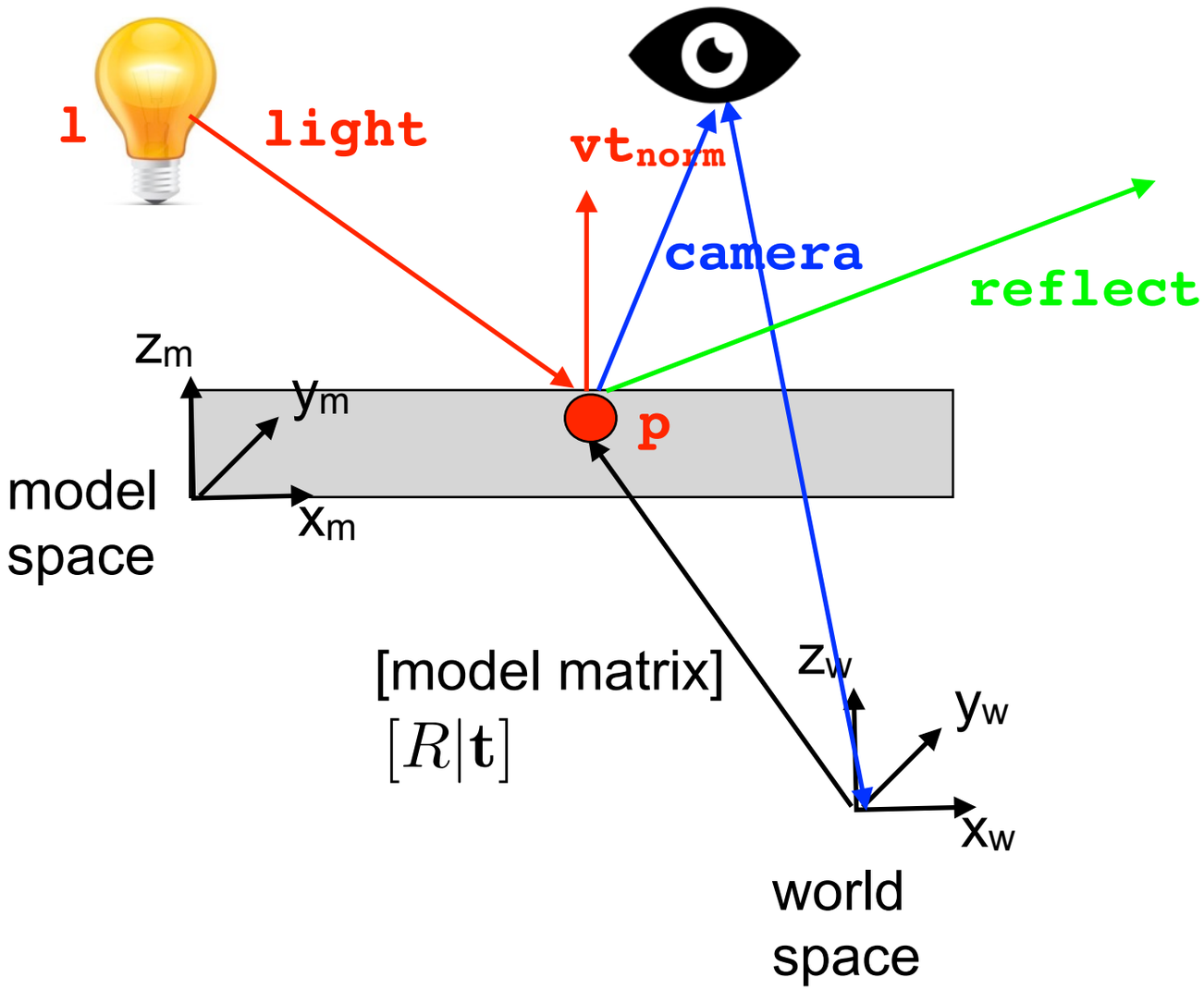
Specular Reflection

ARLAB

Calculate the light vector:

$$\cos(\alpha) = \max(\mathbf{p} \cdot \mathbf{reflect}, 0.0)$$

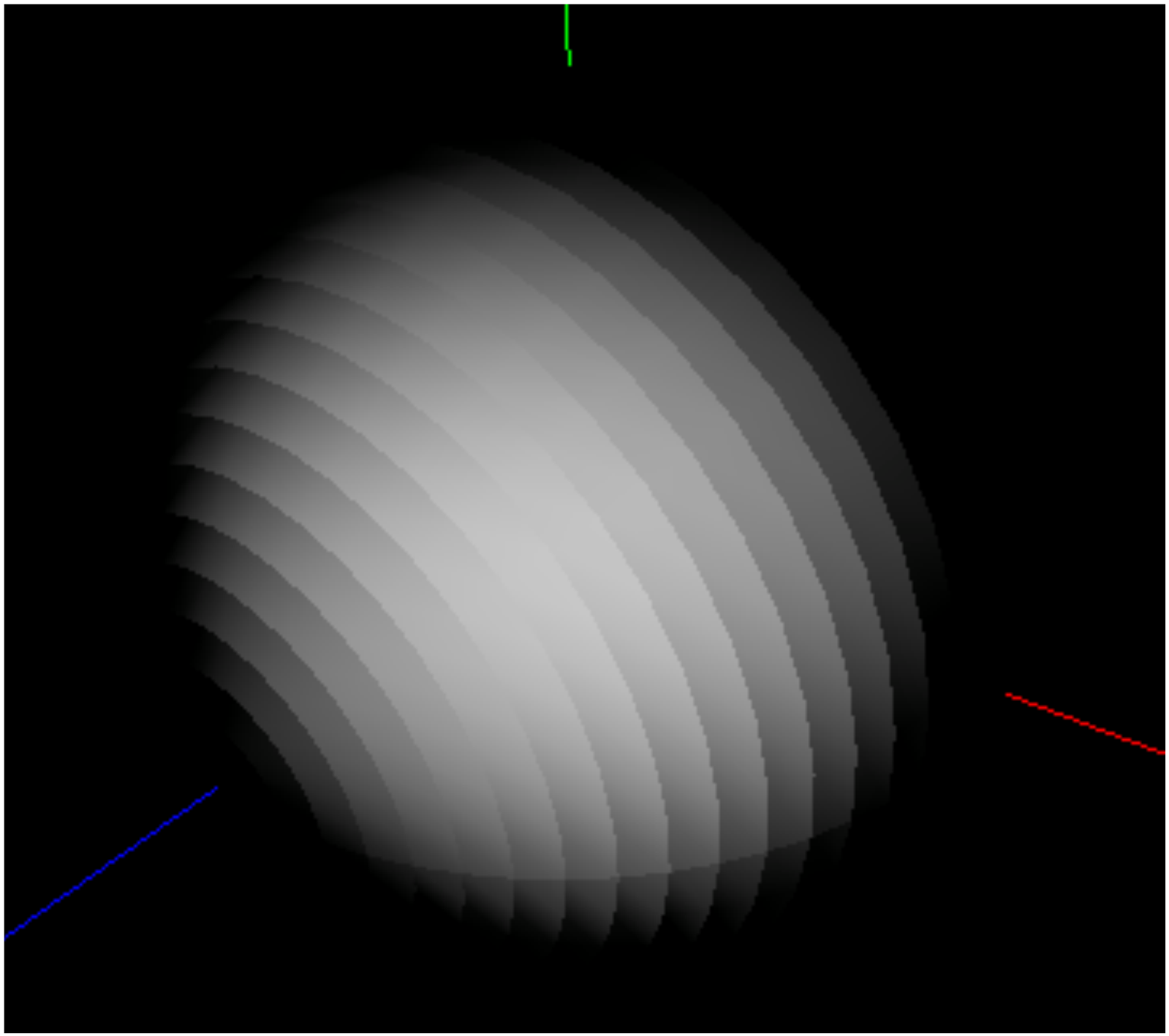
$$specular = \cos(\alpha)^{shininess}$$




```
float cosAngle = max( dot(surfaceToCamera, reflectionVector), 0.0);
float specular_coefficient = pow(cosAngle, shininess);
vec3 out_specular_color = specular_color * specular_coefficient * specular_intensity;
```

Specular Reflection Component

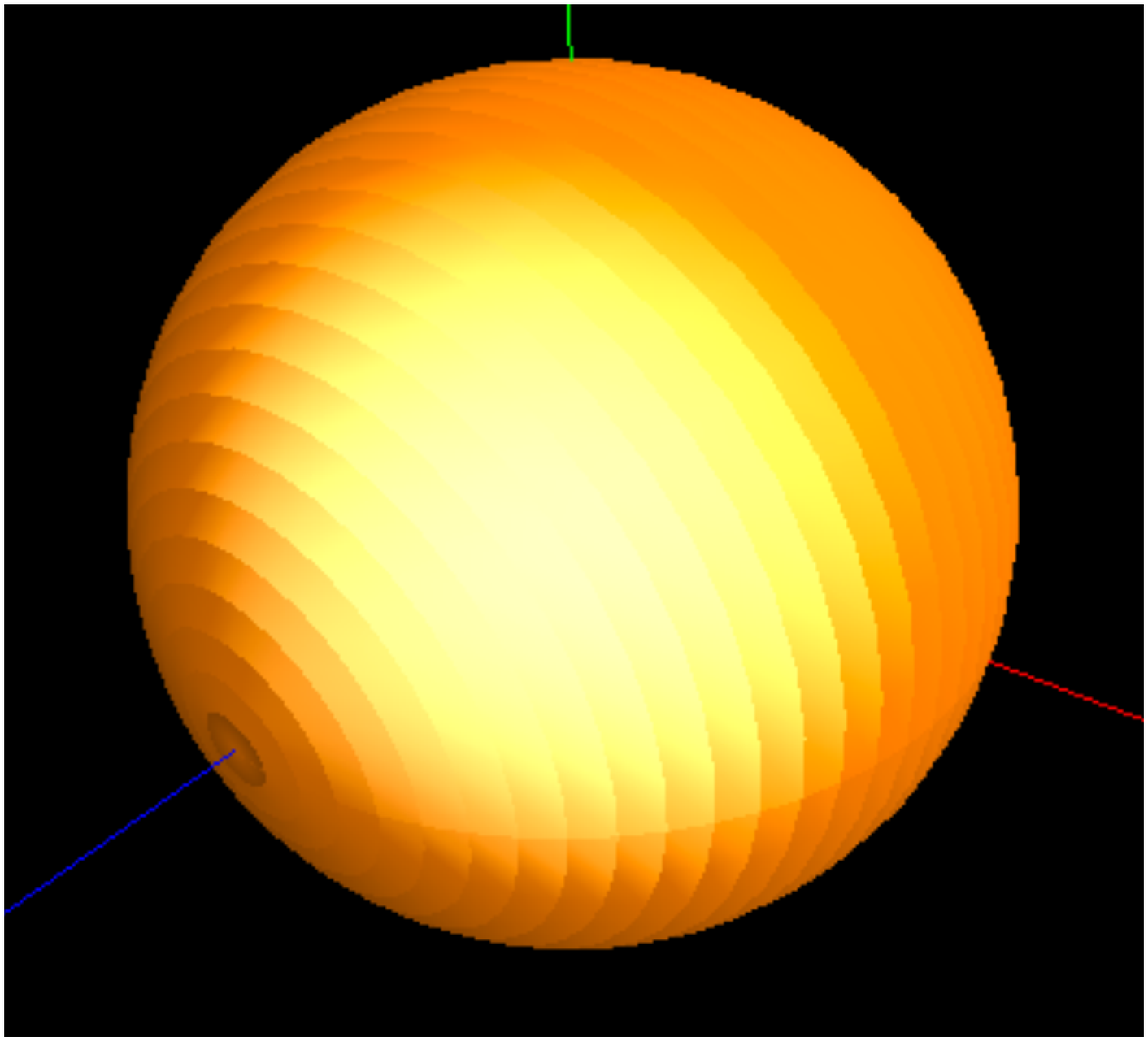
ARLAB



Combination

ARLAB

```
pass_Color = vec3(out_diffuse_color + out_ambient_color + out_specular_color);
```



Thank you!

Questions

Rafael Radkowski, Ph.D.
Iowa State University
Virtual Reality Applications Center
1620 Howe Hall
Ames, Iowa 5001, USA

+1 515.294.5580
+1 515.294.5530(fax)

rafael@iastate.edu

VRAC|HCI

IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY