COSC 414/519I: Computer Graphics

2023W2

Shan Du

Lighting the Translated-Rotated Object

- The normal direction may change when coordinate transformations are applied.

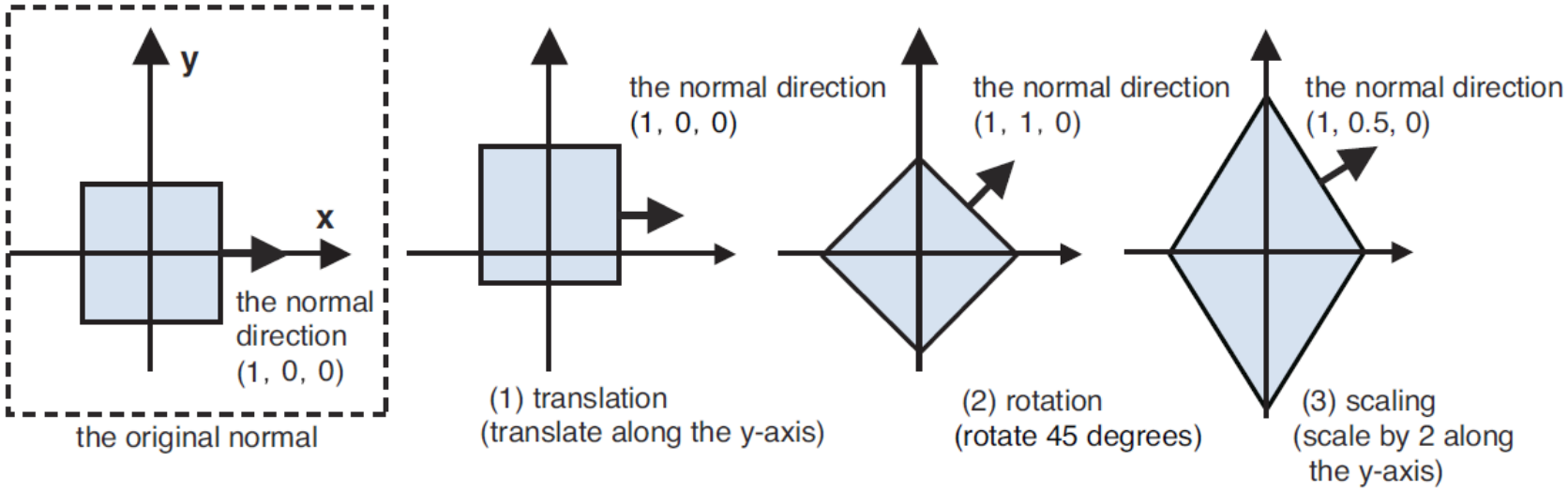


Figure 8.15 The changes of the normal direction due to coordinate transformations

The Magic Matrix: Inverse Transpose Matrix

- The matrix that performs a coordinate transformation on an object is called a model matrix.
- The normal direction can be calculated by multiplying the normal by the inverse transpose matrix of a model matrix. The inverse transpose matrix is the matrix that transposes the inverse of a matrix.

The Magic Matrix: Inverse Transpose Matrix

- The inverse of the matrix *M* is the matrix *R*, where both *R***M* and *M***R* become the identity matrix.
- The term transpose means the operation that exchanges rows and columns of a matrix.

Rule: You can calculate the normal direction if you multiply the normal by the inverse transpose of the model matrix.

- The inverse transpose matrix is calculated as follows:
  1. Invert the original matrix.
  2. Transpose the resulting matrix.

The Magic Matrix: Inverse Transpose Matrix

Table 8.1 Matrix4 Methods for an Inverse Transpose Matrix

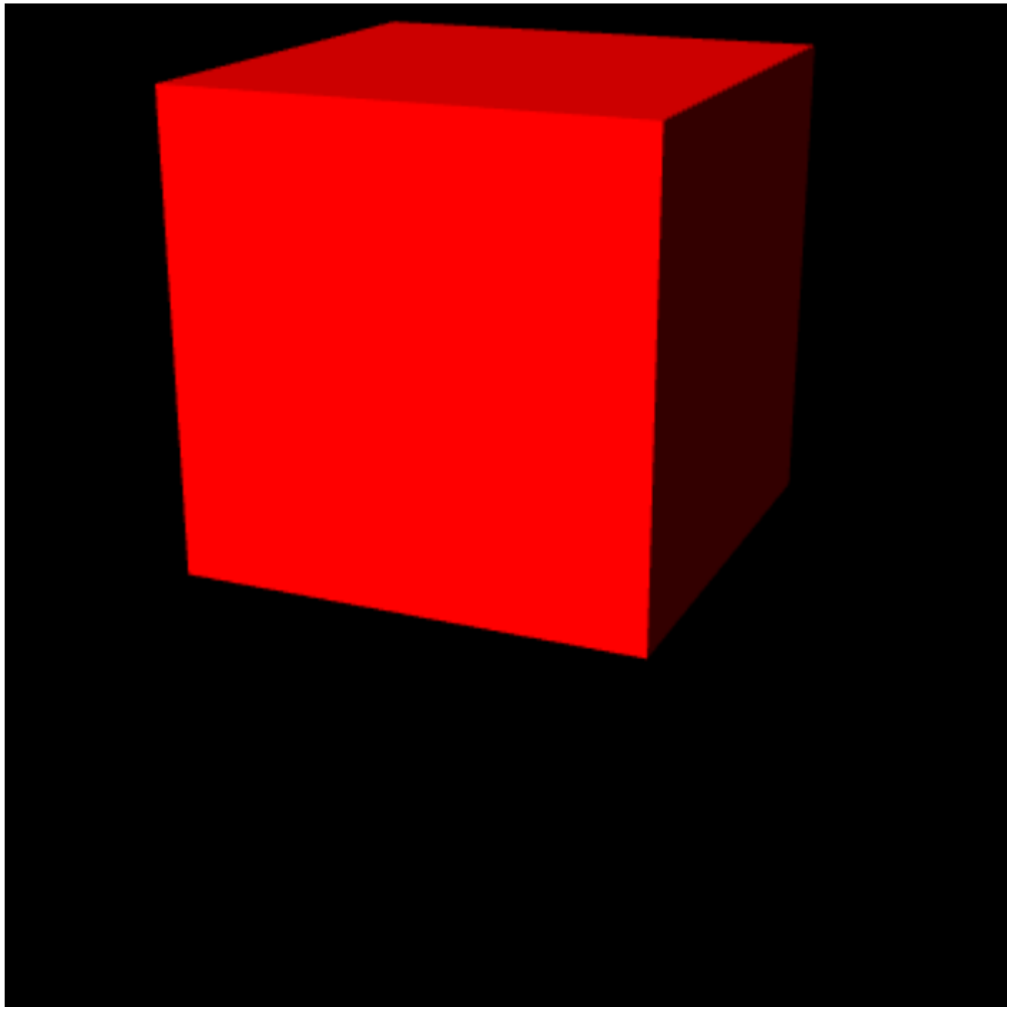
Method	Description
`Matrix4.setInverseOf(m)`	Calculates the inverse of the matrix stored in *m* and stores the result in the `Matrix4` object, where *m* is a `Matrix4` object
`Matrix4.transpose()`	Transposes the matrix stored in the `Matrix4` object and writes the result back into the `Matrix4` object

The Magic Matrix: Inverse Transpose Matrix

- Assuming that a model matrix is stored in *modelMatrix*, which is a *Matrix4* object, the following code snippet will get its inverse transpose matrix. The result is stored in the variable named *normalMatrix* , because it performs the coordinate transformation of a normal:

```
Matrix4 normalMatrix = new Matrix4();
// Calculate the model matrix
...
// Calculate the matrix to transform normal according to the model matrix
normalMatrix.setInverseOf(modelMatrix);
normalMatrix.transpose();
```

Lighted Translated Rotated Cube



Lighted Translated Rotated Cube

var VSHADER_SOURCE =

……

'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of the normal

……

'void main() {\n’ +

……

// Recalculate the normal based on the model matrix and make its length 1.

' vec3 normal = normalize(vec3(u_NormalMatrix * a_Normal));\n’ +

……

'}\n';

Lighted Translated Rotated Cube

```
function main() {
...
  // Get the storage locations of uniform variables and so on
  var u_MvpMatrix = gl.getUniformLocation(gl.program, 'u_MvpMatrix');
  var u_NormalMatrix = gl.getUniformLocation(gl.program, 'u_NormalMatrix');
...
  var modelMatrix = new Matrix4();  // Model matrix
  var mvpMatrix = new Matrix4();    // Model view projection matrix
  var normalMatrix = new Matrix4(); // Transformation matrix for normal

  // Calculate the model matrix
  modelMatrix.setTranslate(0, 1, 0); // Translate to y-axis direction
  modelMatrix.rotate(90, 0, 0, 1);   // Rotate around the z-axis
  // Calculate the view projection matrix
  mvpMatrix.setPerspective(30, canvas.width/canvas.height, 1, 100);
  mvpMatrix.lookAt(-7, 2.5, 6, 0, 0, 0, 0, 1, 0);
  mvpMatrix.multiply(modelMatrix);
  // Pass the model view projection matrix to u_MvpMatrix
  gl.uniformMatrix4fv(u_MvpMatrix, false, mvpMatrix.elements);

  // Calculate matrix to transform normal based on the model matrix
  normalMatrix.setInverseOf(modelMatrix);
  normalMatrix.transpose();
  // Pass the transformation matrix for normal to u_NormalMatrix
  gl.uniformMatrix4fv(u_NormalMatrix, false, normalMatrix.elements);
...
}
```

Using a Point Light Object

- In contrast to a directional light, the direction of the light from a point light source differs at each position in the 3D scene.
- So, when calculating shading, you need to calculate the light direction at the specific position on the surface where the light hits.

Using a Point Light Object

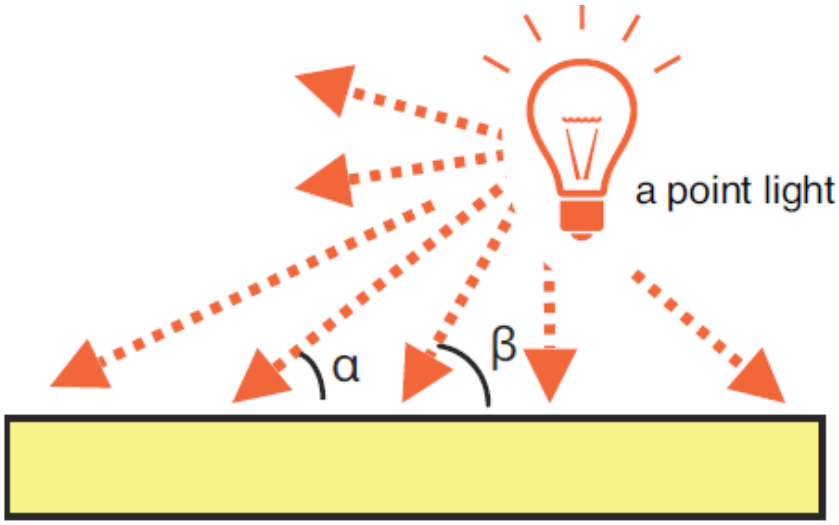
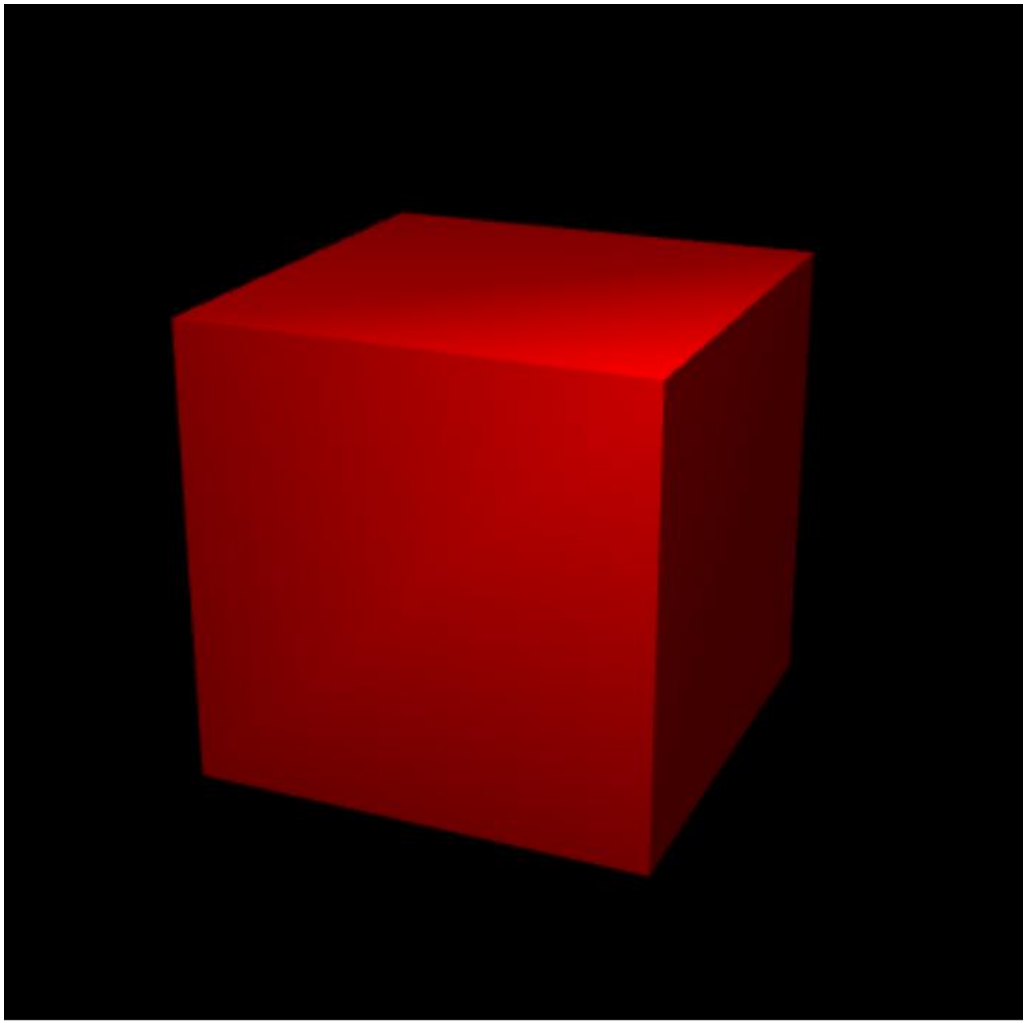


Figure 8.16 The direction of a point light varies by position

Because the light direction changes, you need to pass the position of the light source and then calculate the light direction at each vertex position.

Point Lighted Cube



Point Lighted Cube

```
1 // PointLightedCube.js
2 // Vertex shader program
3 var VSHADER_SOURCE =
4   'attribute vec4 a_Position;\n' +
   ...
8   'uniform mat4 u_ModelMatrix;\n' +  // Model matrix
9   'uniform mat4 u_NormalMatrix;\n' + // Transformation matrix of normal
10  'uniform vec3 u_LightColor;\n' +   // Light color
11  'uniform vec3 u_LightPosition;\n' +  // Position of the light source (in the
                                                    ➥world coordinate system)
12  'uniform vec3 u_AmbientLight;\n' +   // Ambient light color
13  'varying vec4 v_Color;\n' +
14  'void main() {\n' +
15  '  gl_Position = u_MvpMatrix * a_Position;\n' +
16     // Recalculate normal with normal matrix and make its length 1.0
17  '  vec3 normal = normalize(vec3(u_NormalMatrix * a_Normal));\n' +
18    // Calculate the world coordinate of the vertex
19  '  vec4 vertexPosition = u_ModelMatrix * a_Position;\n' +
20     // Calculate the light direction and make it 1.0 in length
21  '  vec3 lightDirection = normalize(u_LightPosition - vec3(vertexPosition));\n' +
22     // The dot product of the light direction and the normal
23  '  float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
24     // Calculate the color due to diffuse reflection
25  '  vec3 diffuse = u_LightColor * a_Color.rgb * nDotL;\n' +
26     // Calculate the color due to ambient reflection
27  '  vec3 ambient = u_AmbientLight * a_Color.rgb;\n' +
28     // Add surface colors due to diffuse and ambient reflection
29  '  v_Color = vec4(diffuse + ambient, a_Color.a);\n' +
30  '}\n';
```

Point Lighted Cube

```
// Get the storage locations of uniform variables and so on
var u_ModelMatrix = gl.getUniformLocation(gl.program, 'u_ModelMatrix');
...
var u_LightColor = gl.getUniformLocation(gl.program,'u_LightColor');
var u_LightPosition = gl.getUniformLocation(gl.program, 'u_LightPosition');
...

gl.uniform3f(u_LightColor, 1.0, 1.0, 1.0);
// Set the position of the light source (in the world coordinate)
gl.uniform3f(u_LightPosition, 0.0, 3.0, 4.0);
...
var modelMatrix = new Matrix4();  // Model matrix
var mvpMatrix = new Matrix4();    // Model view projection matrix
var normalMatrix = new Matrix4(); // Transformation matrix for normal

// Calculate the model matrix
modelMatrix.setRotate(90, 0, 1, 0); // Rotate around the y-axis
// Pass the model matrix to u_ModelMatrix
gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);
...
```

Specular Reflection

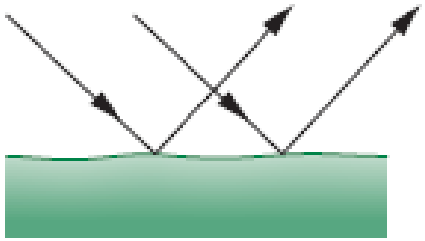
- A specular surface is smooth.
- The smoother the surface is,

the more it resembles a mirror.

- As the surface gets smoother,

the reflected light is concentrated in a smaller range of angles centered about the angle of a perfect reflector.

Specular high-lights.



Specular surface.

Specular Reflection

- Modeling specular surfaces realistically can be complex because the pattern by which the light is scattered is not symmetric. It depends on the wavelength of the incident light, and it changes with the reflection angle.
- Phong proposed an approximate model that can be computed with only a slight increase over the work done for diffuse surfaces.

Specular Reflection

- The model adds a term for specular reflection. Hence, we consider the surface as being rough for the diffuse term and smooth for the specular term.
- The amount of light that the viewer sees depends on the angle Ø between *r*, the direction of a perfect reflector, and *v*, the direction of the viewer.

Specular Reflection

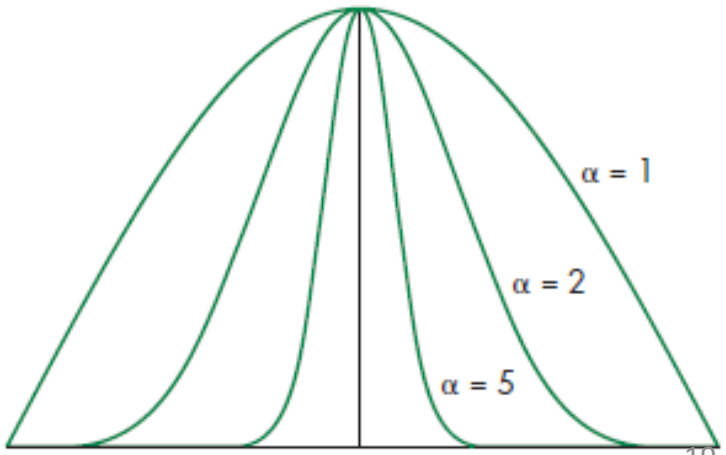
- The Phong model uses the equation

$$I_s = k_s L_s cos^{\alpha} \emptyset$$

  - The coefficient $k_s (0 \leq k_s \leq 1)$ is the fraction of the incoming specular light that is reflected.
  - The exponent α is a shininess coefficient.

Specular Reflection

- As α is increased, the reflected light is concentrated in a narrower region centered on the angle of a perfect reflector. In the limit, as α goes to infinity, we get a mirror; values in the range 100 to 500 correspond to most metallic surfaces, and smaller values (<100) correspond to materials that show broad highlights.



Effect of shininess coefficient.

Specular Reflection

- The computational advantage of the Phong model is that if we have normalized *r* and *v* to unit length, we can again use the dot product, and the specular term becomes

$$I_s = k_s L_s \max\big((r \cdot v)^{\alpha}, 0\big)$$

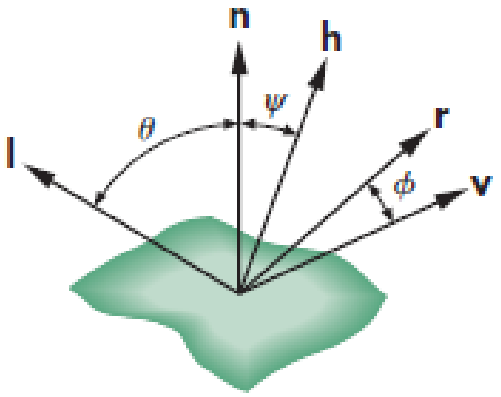
- We can add a distance term

$$I = \frac{1}{a + bd + cd^2}\left(k_d L_d \max(l \cdot n, 0)\right.$$
$$\left. + k_s L_s \max\big((r \cdot v)^{\alpha}, 0\big)\right) + k_a L_a$$

This formula is computed for each light source and for each primary.

The Modified Phong Model

- If we use the Phong model with specular reflections in our rendering, the dot product $r \cdot v$ should be recalculated at every point on the surface.

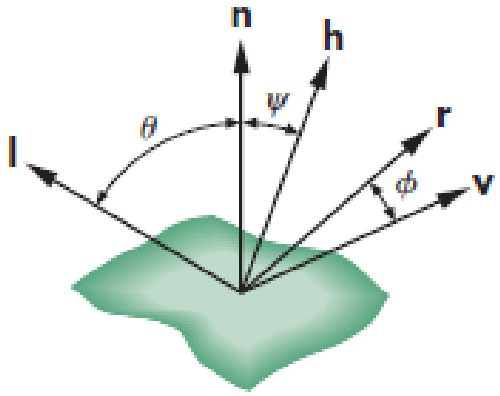


Determination of the halfway vector.

- We can obtain an interesting approximation by using the unit vector halfway between the viewer vector and the light-source vector:

The Modified Phong Model

$$h = \frac{l + v}{|l + v|}$$



Determination of the halfway vector.

Ψ is the angle between *n* and *h*, the halfway angle.

- When *v* lies in the same plane as *l*, *n*, and *r*, we have 2Ψ=∅.
- If we replace $r \cdot v$ with $\mathrm{n} \cdot h$, we can avoid calculation of *r*.

The Modified Phong Model

- Since the halfway angle Ψ is smaller than Ø, and if we use the same exponent e in $(n \cdot h)^e$ that we used in $(r \cdot v)^e$, then the size of the specular highlights will be larger.
- We can mitigate this problem by replacing the value e with a value e' so that $(n \cdot h)^{e'}$ is closer to $(r \cdot v)^e$.
- When we use the halfway vector in the calculation of the specular term, we are using the Blinn-Phong, or modified Phong, lighting model.