Lighting in Computer Graphics

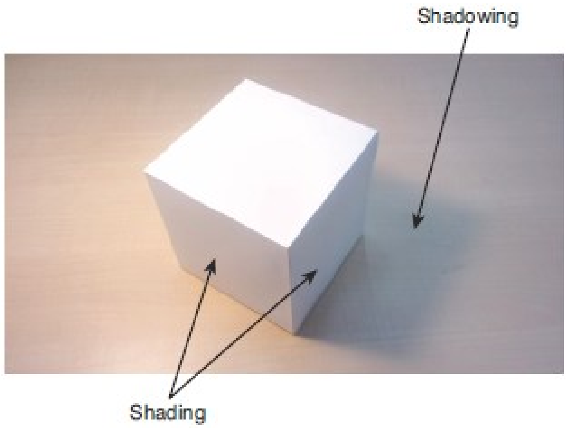
Frederick Li

Durham University

This Lesson

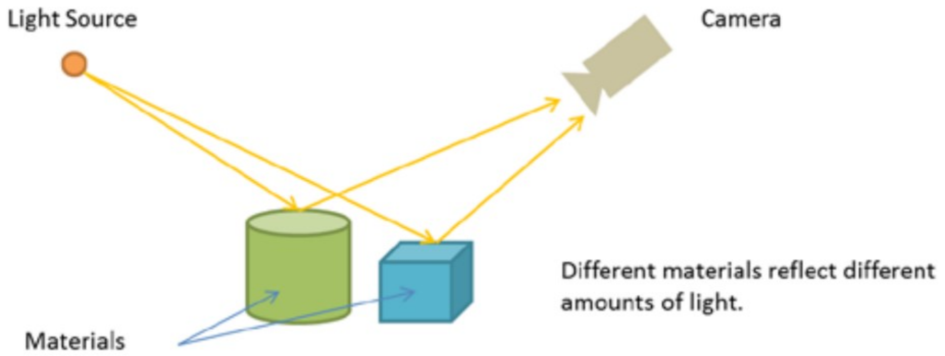
- **Lighting**:
  - Light hits an object, part of it is reflected by the surface of the object. Only after this reflected light enters your eyes can you see the object and distinguish its colour
  - Essential for creating realistic 3D scenes because it helps to give the scene a sense of depth

- Types of lighting
- Surface Normal
- Shading



Shading

- Generally, the process for re-creating the phenomenon where colours differ from surface to surface due to lighting.

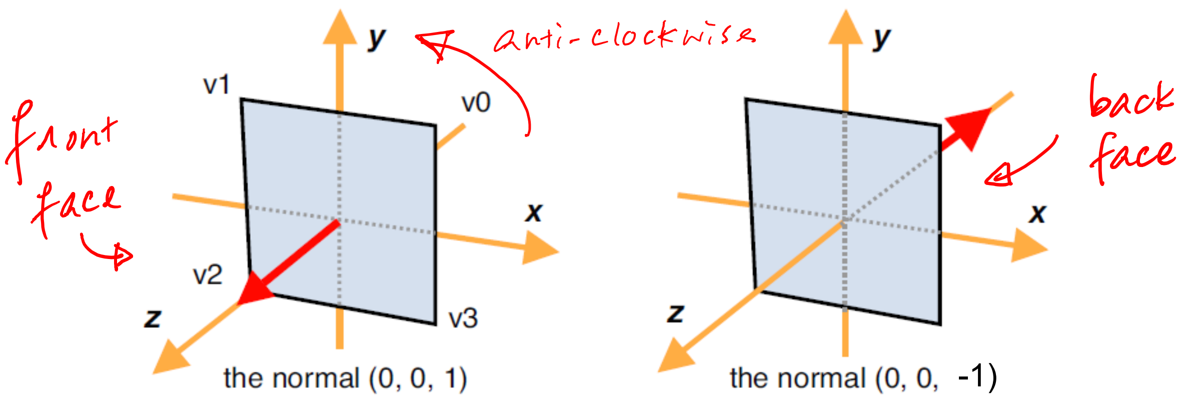


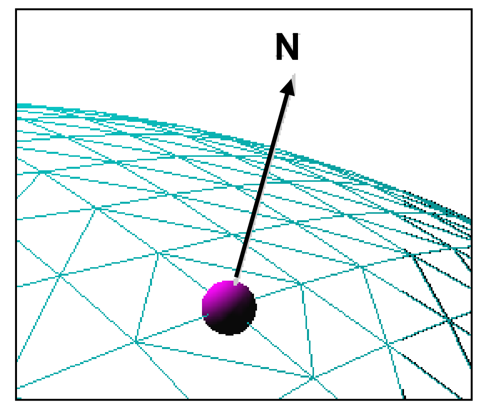
- In CG, shading is the process of altering the colour of an object/surface/polygon, based on:
  - The type of light source that is emitting light
  - How the light is reflected from object surfaces and enters the eye

  to create a photorealistic effect.

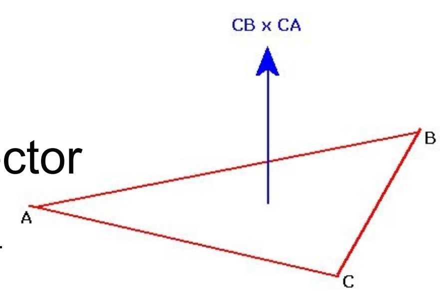
Normal Vector

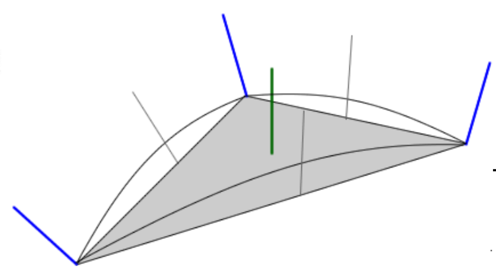
- The orientation of a surface is specified by the direction perpendicular to the surface and is called a **normal** (**normal vector**)
- A surface has a front and a back face, each side has its own normal





A single Normal Vector





For curved surface, normal Vector of surface point is generated by interpolation.

Durham University

Normals of the surfaces of a cube

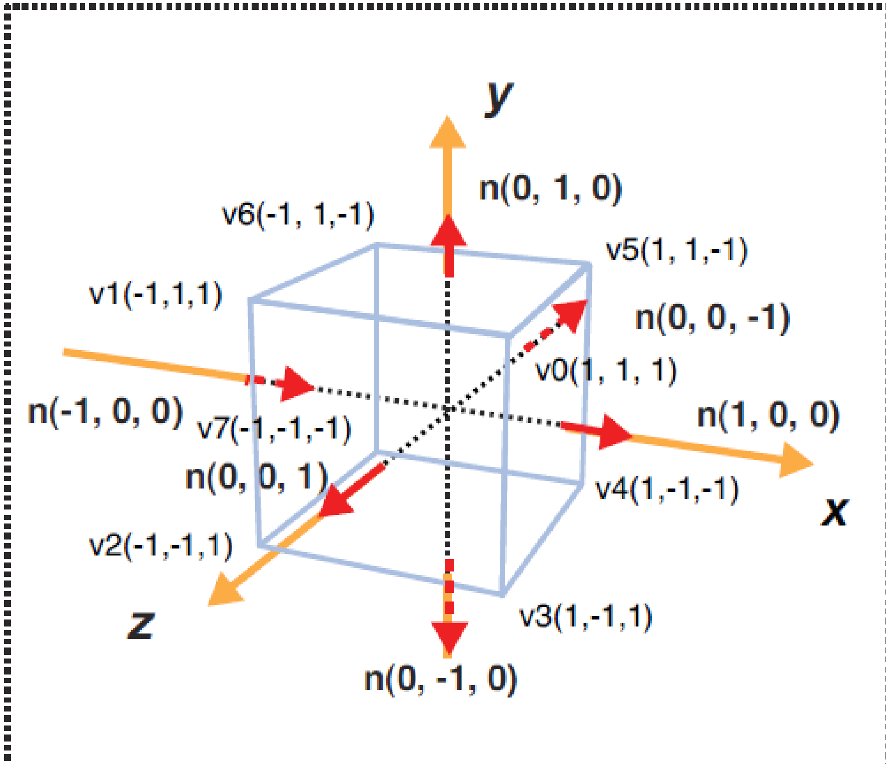
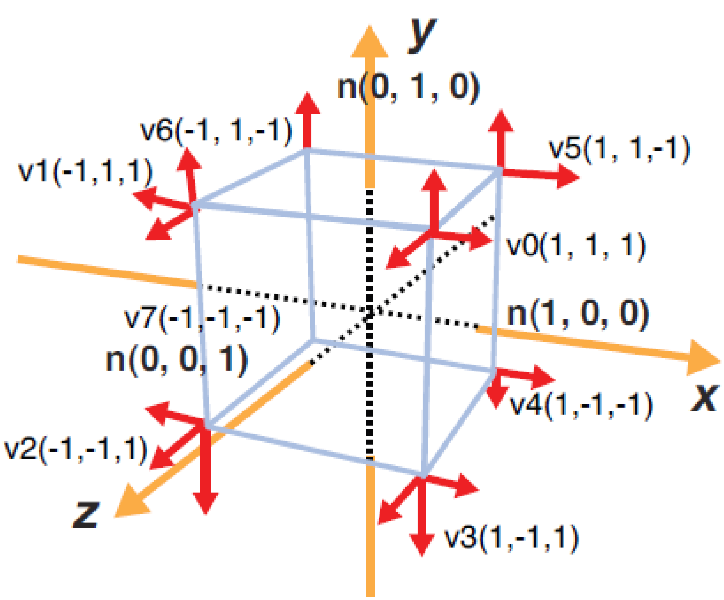


Figure 8.9 Normals of the surfaces of a cube



Normals at the corner vertices are generated by interpolation.

Model Transformation

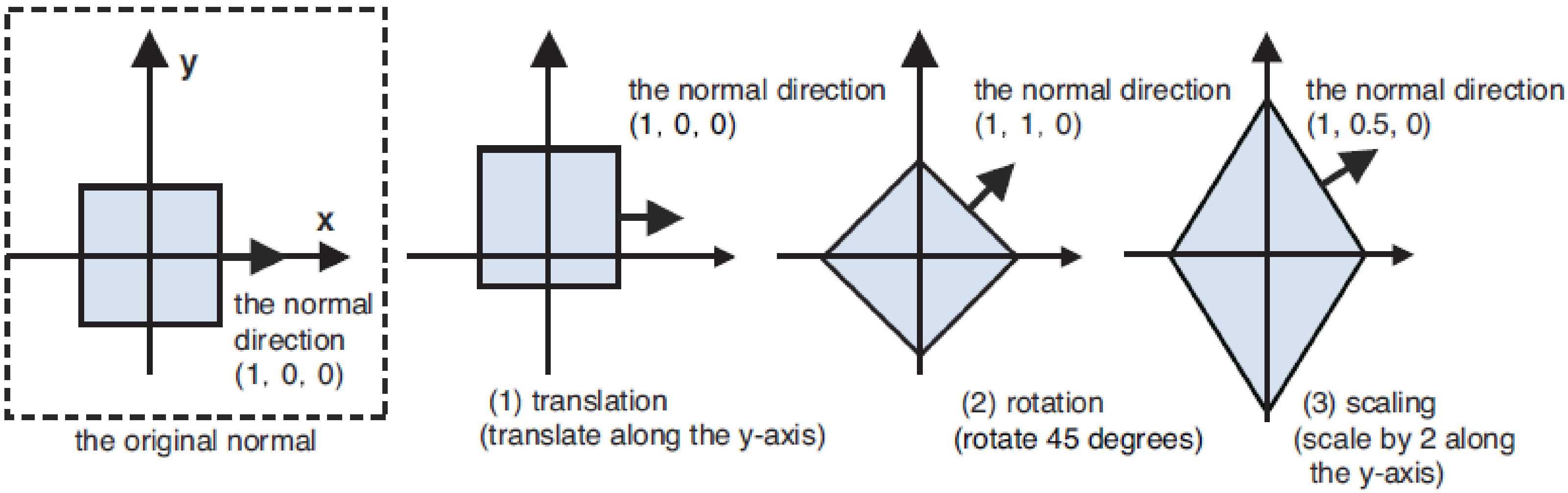


Figure 8.15 The changes of the normal direction due to coordinate transformations

Transformed Surface Normal

- Assuming that a model matrix (comprises a set of transformation operations) is stored in ***modelMatrix***, e.g.

```
// 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
```

- Transformed Normal vector is then calculated by **$(\mathrm{modelMatrix}^{-1})^{T}$**

```
// 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);
```

Reference: WebGL Programming Guide, p.311-314

Types of Shading

Traditionally supported by non-shader-based OpenGL

- **Flat Shading**
  - Assign a single colour to each face (triangle) of an object
- **Gouraud (Smooth) Shading**
  - Apply lighting against the normal vector at each vertex to calculate a vertex colour **[vertex shader]**
  - Colours across a face are generated by interpolating colours obtained at the corner vertices of the face **[rasterization]**
- **Phong Shading**
  - Normal vector at each point over an object surface is obtained by interpolating normal vectors of the corner vertices of the surface **[rasterization]**
  - Colour of each surface point will be calculated by applying lighting against the interpolated normal vector at the point **[fragment shader]**

Types of Light Source in CG

- **Directional light**: like the sun that emits light naturally (from very far away, generating parallel light rays)
- **Point light**: like a light bulb that emits light artificially in all directions from a point
- **Ambient light**: represents indirect light, that is, light emitted from all light sources and reflected by walls or other

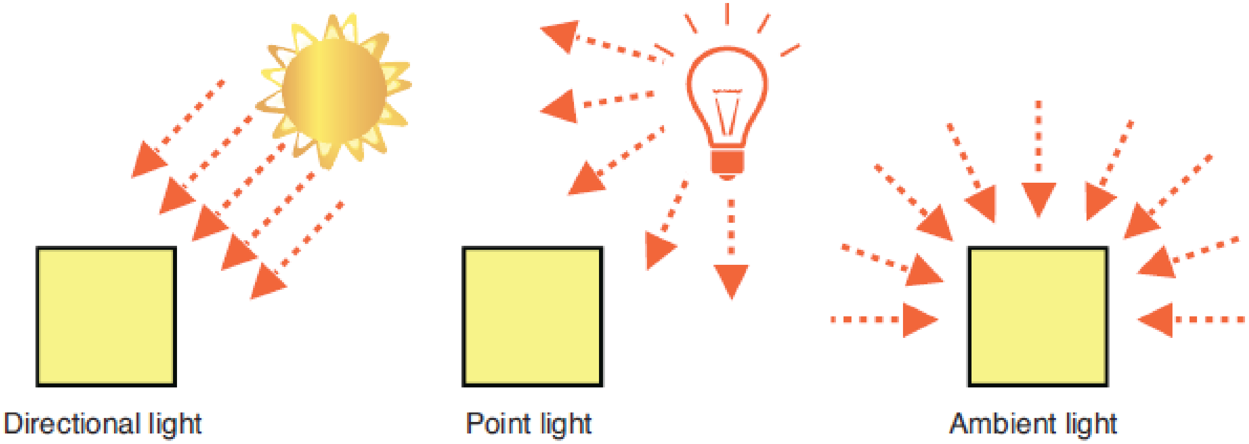


Figure 8.2 Directional light, point light, and ambient light

Types of Reflected Light

- Illuminate objects: How light is reflected by the surface of an object and then enters the eye
- Colour the surface determined by:
  - Type of the light (colour and direction)
  - Type of surface of the object (colour and orientation)
- Two main types: **diffuse reflection** and **environment** (or **ambient**) **reflection**

$$\langle \textit{surface color by diffuse and ambient reflection} \rangle =$$
$$\langle \textit{surface color by diffuse reflection} \rangle + \langle \textit{surface color by ambient reflection} \rangle$$

Ambient Reflection

- Ambient reflection is the reflection of light from another light source
- Illuminates an object equally from all directions with the same intensity, its brightness is the same at any position

$$\langle \textit{surface color by ambient reflection} \rangle = \langle \textit{light color} \rangle \times \langle \textit{base color of surface} \rangle$$

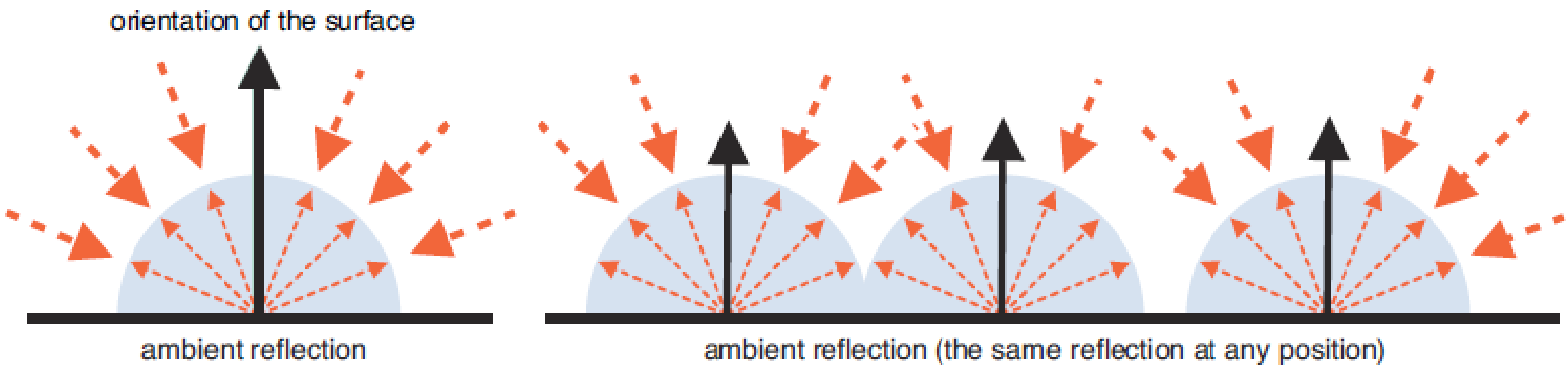


Figure 8.5 Ambient reflection

Diffuse Reflection

- Reflection of light from a directional light or a point light
- Light is reflected equally in all directions from where it hits (due to rough surface)
- θ : Angle between light direction and surface orientation (direction “perpendicular” to the surface)

$$\langle surface\ color\ by\ diffuse\ reflection \rangle = \langle light\ color \rangle \times \langle base\ color\ of\ surface \rangle \times \cos\theta$$

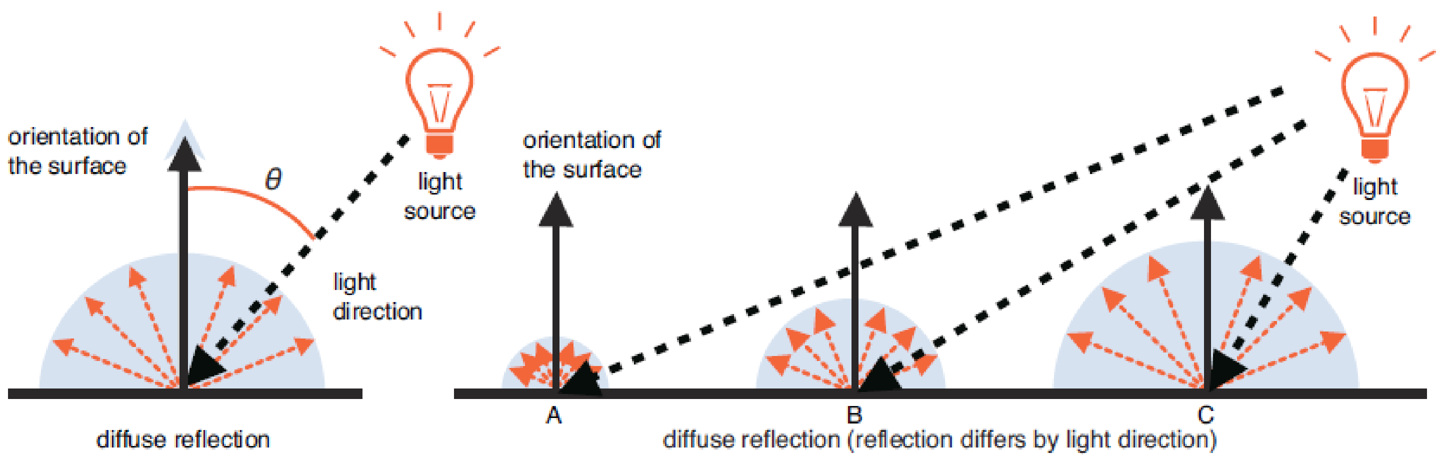


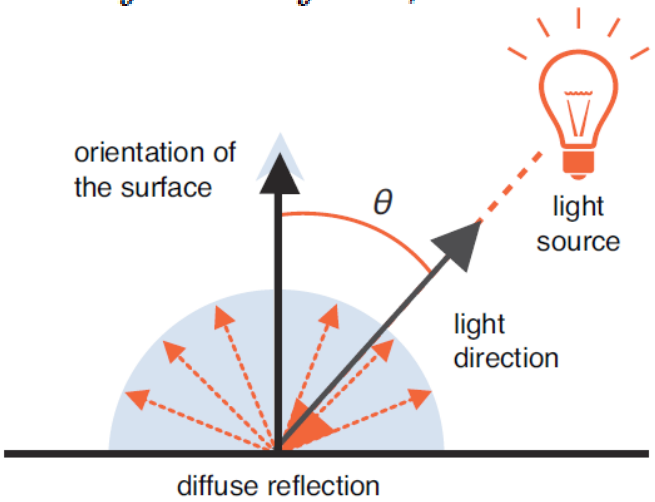
Figure 8.3 Diffuse reflection

Calculate *cos θ*

- *cos θ* is derived by calculating the dot product of the light direction and the orientation of a surface

$$\cos\theta = \langle light\ \ direction\rangle \bullet \langle orientation\ \ of\ \ a\ \ surface\rangle$$

- i.e. Diffusive reflection is:



$$\begin{aligned}&\langle surface\ \ color\ \ by\ \ diffuse\ \ reflection\rangle = \\ &\qquad \langle light\ \ color\rangle \times \langle base\ \ color\ \ of\ \ surface\rangle \times \\ &\qquad\quad (\langle light\ \ direction\rangle \bullet \langle orientation\ \ of\ \ a\ \ surface\rangle)\end{aligned}$$

Calculate *cos θ*

[4] Mathematically, the dot product of two vectors n and l is written as follows:

$n \bullet 1 = |n| \times |1| \times \cos\theta$

where | | means the length of the vector. From this equation, you can see that when the lengths of n and l are 1.0, the dot product is equal to $\cos\theta$. If n is (n_x, n_y, n_z) and l is (l_x, l_y, l_z), then $n_l = n_x * l_x + n_y * l_y + n_z * l_z$ from the law of cosines.

[5] If the components of the vector n are (n_x, n_y, n_z), its length is as follows:

$$\text{length of n} = |n| = \sqrt{nx^2 + ny^2 + nz^2}$$

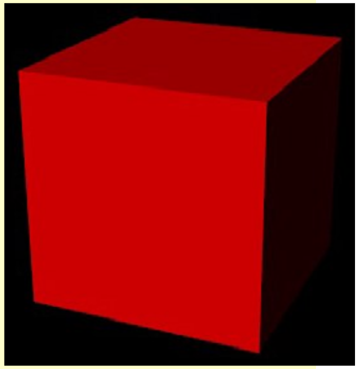
[6] Normalized n is $(n_x/m, n_y/m, n_z/m)$, where m is the length of n. $|n| = \text{sqrt}(9) = 3$. The vector (2.0, 2.0, 1.0) above is normalized into (2.0/3.0, 2.0/3.0, 1.0/3.0).

Directional Lighted Cube

```
Code for Javascript main()

// Set the light color (white)
gl.uniform3f(u_LightColor, 1.0, 1.0, 1.0);

// Set the light direction (in the world coordinate)
var lightDirection = new Vector3([0.5, 3.0, 4.0]);
lightDirection.normalize(); // Normalize
gl.uniform3fv(u_LightDirection, lightDirection.elements););
```

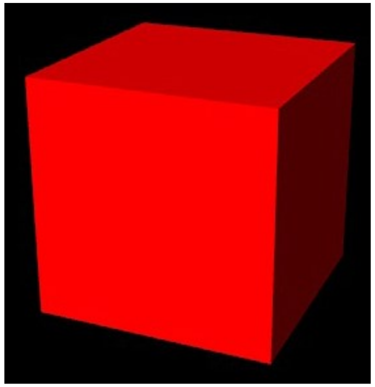


```
V.SHADER:
// Make the length of the normal 1.0
  ' vec3 normal = normalize(vec3(a_Normal));\n' +
// Dot product of light direction and orientation of a surface
  ' float nDotL = max(dot(u_LightDirection, normal), 0.0);\n' +
// Calculate the colour due to diffuse reflection
  ' vec3 diffuse = u_LightColor * vec3(a_Color) * nDotL;\n' +
  ' v_Color = vec4(diffuse, a_Color.a);\n' +
```

Require 1) Surface color, 2) surface orientation, 3) light color, 4) light direction

Ambient Lighting added

$$\langle \textit{surface color by ambient reflection} \rangle = \langle \textit{light color} \rangle \times \langle \textit{base color of surface} \rangle$$



$$\langle \textit{surface color by diffuse and ambient reflection} \rangle = \langle \textit{surface color by diffuse reflection} \rangle + \langle \textit{surface color by ambient reflection} \rangle$$

V.SHADER:

```
// The dot product of the light direction and the normal
 ' float nDotL = max(dot(lightDirection, normal), 0.0);\n' +
// Calculate the color due to diffuse reflection
 ' vec3 diffuse = u_LightColor * a_Color.rgb * nDotL;\n' +
// Calculate the color due to ambient reflection
 ' vec3 ambient = u_AmbientLight * a_Color.rgb;\n' +
// Add surface colors due to diffuse and ambient reflection
 ' v_Color = vec4(diffuse + ambient, a_Color.a);\n' +
```

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

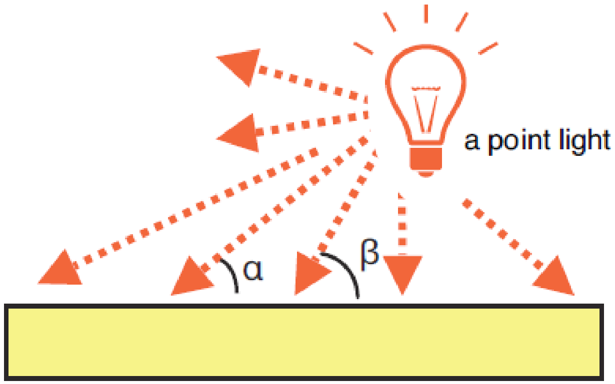
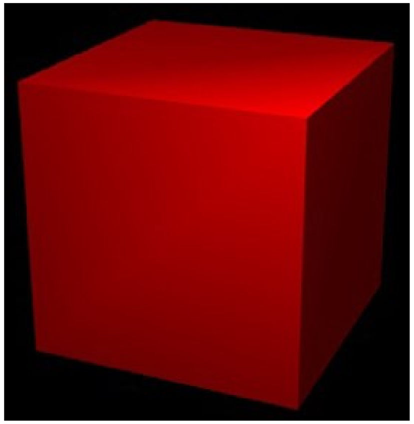


Figure 8.16 The direction of a point light varies by position



- So, when calculating shading, you need to calculate the light direction at the specific position on the surface where the light hits
- **Light direction changes:** pass the position of the light source and then calculate the light direction at each vertex position

Point Light Implementation

V.SHADER:

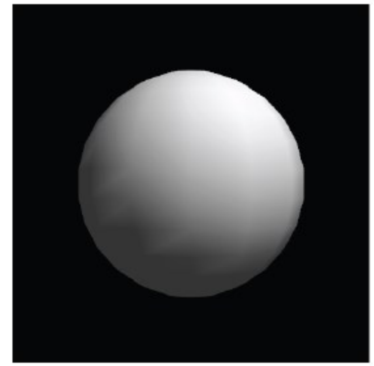
```
// Calculate the world coordinate of the vertex
 ' vec4 vertexPosition = u_ModelMatrix * a_Position;\n' +
// Calculate the light direction and make it 1.0 in length
 ' vec3 lightDirection = normalize(u_LightPosition - vec3(vertexPosition));\n' +

// The dot product of the light direction and the normal
 ' float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
// Calculate the color due to diffuse reflection
 ' vec3 diffuse = u_LightColor * a_Color.rgb * nDotL;\n' +
// Calculate the color due to ambient reflection
 ' vec3 ambient = u_AmbientLight * a_Color.rgb;\n' +
// Add surface colors due to diffuse and ambient reflection
 ' v_Color = vec4(diffuse + ambient, a_Color.a);\n' +
```

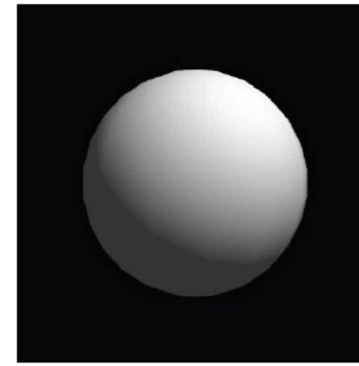
More Realistic Shading
Calculating the Colour per Fragment

VERTEX SHADER:

```
' gl_Position = u_MvpMatrix * a_Position;\n' +
// Calculate the v.pos. in the world coordinate
' v_Position = vec3(u_ModelMatrix * a_Position);\n' +
' v_Normal = normalize(vec3(u_NormalMatrix *
                           a_Normal));\n' +
' v_Color = a_Color;\n' +
```



per-vertex calculation



per-position calculation

FRAGMENT SHADER:

```
'void main() {\n' +
 // Normalize normal because it's interpolated and not 1.0 (length)
 ' vec3 normal = normalize(v_Normal);\n' +
 // Calculate the light direction and make it 1.0 in length
 ' vec3 lightDirection = normalize(u_LightPosition - v_Position);\n' +
 // The dot product of the light direction and the normal
 ' float nDotL = max(dot( lightDirection, normal), 0.0);\n' +
 // Calculate the final color from diffuse and ambient reflection
 ' vec3 diffuse = u_LightColor * v_Color.rgb * nDotL;\n' +
 ' vec3 ambient = u_AmbientLight * v_Color.rgb;\n' +
 ' gl_FragColor = vec4(diffuse + ambient, v_Color.a);\n' +
'}\n';
```

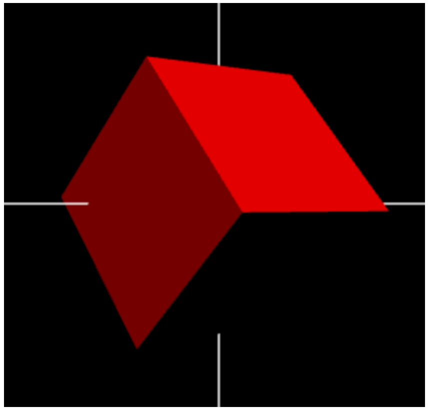
Example: Directional Lighting

Vertex Shader:

1. Update Normal Vectors
2. Calculate dot product between light source and vertex normal
3. Calculate diffuse reflected light multiplying vertex color, light source color and the dot product

```
'void main() {\n' +
'  gl_Position = u_ProjMatrix * u_ViewMatrix * u_ModelMatrix * a_Position;\n' +

'  vec3 normal = normalize((u_NormalMatrix * a_Normal).xyz);\n' +
'  float nDotL = max(dot(normal, u_LightDirection), 0.0);\n' +
   // Calculate the color due to diffuse reflection
'  vec3 diffuse = u_LightColor * a_Color.rgb * nDotL;\n' +
'  v_Color = vec4(diffuse, a_Color.a);\n' +  '  }\n' +
```



Cube: Data Structure

```
function initVertexBuffers(gl) {
  // Create a cube
  //    v6----- v5
  //   /|      /|
  //  v1------v0|
  //  | |     | |
  //  | |v7---|-|v4
  //  |/      |/
  //  v2------v3
  var vertices = new Float32Array([   // Coordinates
     1.0, 1.0, 1.0,  -1.0, 1.0, 1.0,  -1.0,-1.0, 1.0,   1.0,-1.0, 1.0, // v0-v1-v2-v3 front
     1.0, 1.0, 1.0,   1.0,-1.0, 1.0,   1.0,-1.0,-1.0,   1.0, 1.0,-1.0, // v0-v3-v4-v5 right
     1.0, 1.0, 1.0,   1.0, 1.0,-1.0,  -1.0, 1.0,-1.0,  -1.0, 1.0, 1.0, // v0-v5-v6-v1 up
    -1.0, 1.0, 1.0,  -1.0, 1.0,-1.0,  -1.0,-1.0,-1.0,  -1.0,-1.0, 1.0, // v1-v6-v7-v2 left
    -1.0,-1.0,-1.0,   1.0,-1.0,-1.0,   1.0,-1.0, 1.0,  -1.0,-1.0, 1.0, // v7-v4-v3-v2 down
     1.0,-1.0,-1.0,  -1.0,-1.0,-1.0,  -1.0, 1.0,-1.0,   1.0, 1.0,-1.0  // v4-v7-v6-v5 back
  ]);

  ...

  var normals = new Float32Array([    // Normal
    0.0, 0.0, 1.0,   0.0, 0.0, 1.0,   0.0, 0.0, 1.0,   0.0, 0.0, 1.0,  // v0-v1-v2-v3 front
    1.0, 0.0, 0.0,   1.0, 0.0, 0.0,   1.0, 0.0, 0.0,   1.0, 0.0, 0.0,  // v0-v3-v4-v5 right
    0.0, 1.0, 0.0,   0.0, 1.0, 0.0,   0.0, 1.0, 0.0,   0.0, 1.0, 0.0,  // v0-v5-v6-v1 up
   -1.0, 0.0, 0.0,  -1.0, 0.0, 0.0,  -1.0, 0.0, 0.0,  -1.0, 0.0, 0.0,  // v1-v6-v7-v2 left
    0.0,-1.0, 0.0,   0.0,-1.0, 0.0,   0.0,-1.0, 0.0,   0.0,-1.0, 0.0,  // v7-v4-v3-v2 down
    0.0, 0.0,-1.0,   0.0, 0.0,-1.0,   0.0, 0.0,-1.0,   0.0, 0.0,-1.0   // v4-v7-v6-v5 back
  ]);

  // Indices of the vertices
  var indices = new Uint8Array([
     0, 1, 2,   0, 2, 3,    // front
     4, 5, 6,   4, 6, 7,    // right
     8, 9,10,   8,10,11,    // up
    12,13,14,  12,14,15,    // left
    16,17,18,  16,18,19,    // down
    20,21,22,  20,22,23     // back
 ]);
```

Draw function: Draw x, y, z axes

```
function draw(gl, u_ModelMatrix, u_NormalMatrix, u_isLighting) {

  // Clear color and depth buffer
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  gl.uniform1i(u_isLighting, false); // Will not apply lighting

  // Set the vertex coordinates and color (for the x, y axes)

  var n = initAxesVertexBuffers(gl);
  if (n < 0) {
    console.log('Failed to set the vertex information');
    return;
  }

  // Calculate the view matrix and the projection matrix
  modelMatrix.setTranslate(0, 0, 0);  // No Translation
  // Pass the model matrix to the uniform variable
  gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);

  // Draw x and y axes
  gl.drawArrays(gl.LINES, 0, n);

  gl.uniform1i(u_isLighting, true); // Will apply lighting

  // |.... To be add: draw the cube
}
```

→ Next page

Draw function: Draw the Cube

```
// ... codes for drawing x, y, z axes   : last page ←
// now start drawing the cube

gl.uniform1i(u_isLighting, true); // Will apply lighting

// Set the vertex coordinates and color (for the colorful square)
var n = initVertexBuffers(gl);
if (n < 0) {
  console.log('Failed to set the vertex information');
  return;
}

// Rotate, and then translate
modelMatrix.setTranslate(0, 0, 0);  // Translation
modelMatrix.rotate(g_yAngle, 0, 1, 0); // Rotate
modelMatrix.rotate(g_xAngle, 1, 0, 0); // Rotate
modelMatrix.scale(1.5, 1.5, 1.5); // Rotate Scale

// Pass the model matrix to the uniform variable
gl.uniformMatrix4fv(u_ModelMatrix, false, modelMatrix.elements);

// Calculate the normal transformation matrix and pass it to u_NormalMatrix
g_normalMatrix.setInverseOf(modelMatrix);
g_normalMatrix.transpose();
gl.uniformMatrix4fv(u_NormalMatrix, false, g_normalMatrix.elements);

// Draw the cube
gl.drawElements(gl.TRIANGLES, n, gl.UNSIGNED_BYTE, 0);
```

Prepare a "Normal" tranformation matrix

Main() and User Interaction

```
function main() {

  // A lot of initialization to do here .... WebGL init., memory allocation, matrices setup, etc.

  document.onkeydown = function(ev){
    keydown(ev, gl, u_ModelMatrix, u_NormalMatrix, u_isLighting);
  };

  draw(gl, u_ModelMatrix, u_NormalMatrix, u_isLighting);
}

function keydown(ev, gl, u_ModelMatrix, u_NormalMatrix, u_isLighting) {
  switch (ev.keyCode) {
    case 40: // Up arrow key -> the positive rotation of arm1 around the y-axis
      g_xAngle = (g_xAngle + ANGLE_STEP) % 360;
      break;
    case 38: // Down arrow key -> the negative rotation of arm1 around the y-axis
      g_xAngle = (g_xAngle - ANGLE_STEP) % 360;
      break;
    case 39: // Right arrow key -> the positive rotation of arm1 around the y-axis
      g_yAngle = (g_yAngle + ANGLE_STEP) % 360;
      break;
    case 37: // Left arrow key -> the negative rotation of arm1 around the y-axis
      g_yAngle = (g_yAngle - ANGLE_STEP) % 360;
      break;
    default: return; // Skip drawing at no effective action
  }

  // Draw the scene
  draw(gl, u_ModelMatrix, u_NormalMatrix, u_isLighting);
}
```

Options in Vertex Shader

```
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' +
  'attribute vec4 a_Color;\n' +
  'attribute vec4 a_Normal;\n' +        // Normal
  'uniform mat4 u_ModelMatrix;\n' +
  'uniform mat4 u_NormalMatrix;\n' +
  'uniform mat4 u_ViewMatrix;\n' +
  'uniform mat4 u_ProjMatrix;\n' +
  'uniform vec3 u_LightColor;\n' +     // Light color
  'uniform vec3 u_LightDirection;\n' + // Light direction (in the world coordinate, normalized)
  'varying vec4 v_Color;\n' +
  'uniform bool u_isLighting;\n' +
  'void main() {\n' +
  '  gl_Position = u_ProjMatrix * u_ViewMatrix * u_ModelMatrix * a_Position;\n' +
  '  if(u_isLighting)\n' +
  '  {\n' +
  '     vec3 normal = normalize((u_NormalMatrix * a_Normal).xyz);\n' +
  '     float nDotL = max(dot(normal, u_LightDirection), 0.0);\n' +
        // Calculate the color due to diffuse reflection
  '     vec3 diffuse = u_LightColor * a_Color.rgb * nDotL;\n' +
  '     v_Color = vec4(diffuse, a_Color.a);\n' + '  }\n' +
  '  else\n' +
  '  {\n' +
  '     v_Color = a_Color;\n' +
  '  }\n' +
  '}\n';
```

Apply directional lighting

No lighting is applied

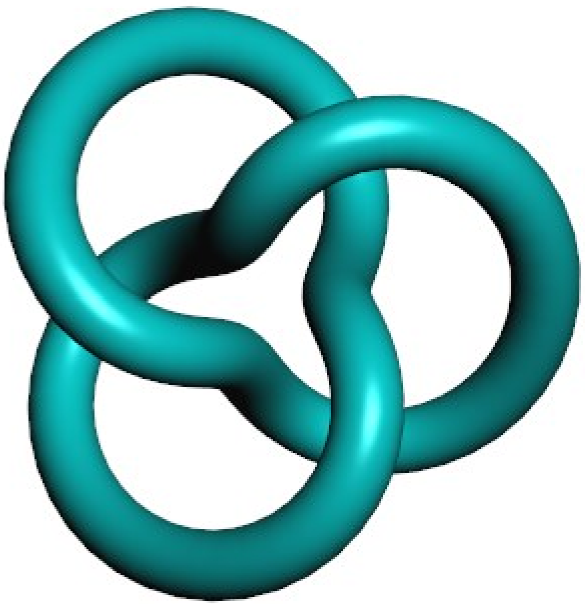
Summary

- Light modelling in CG
- Types of lighting and their interaction with an object surface
- Calculation of surface normal
- Implementation
  - Vertex based (Flat shading, Gouraud shading)
  - Fragment based (Phong shading)

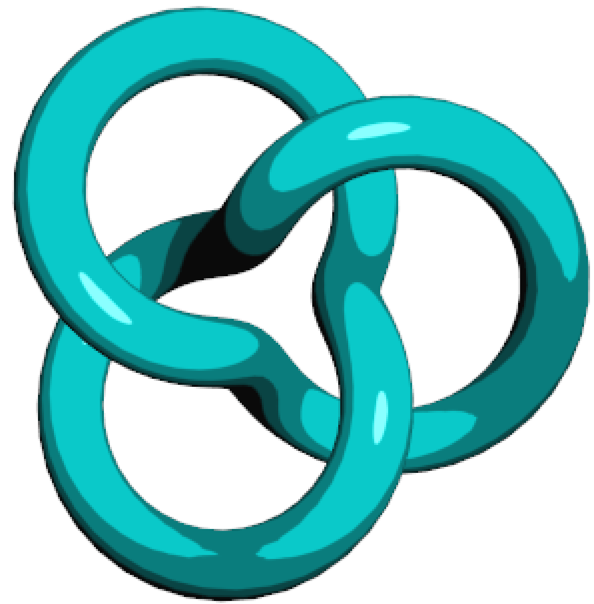
- **Reference:**
  WebGL Programming Guide [Ch. 8]

Appendix

- **Cel Shading**: Cartoon-like shading
  http://prideout.net/blog/?p=22



Normal shading



Cartoon-like shading