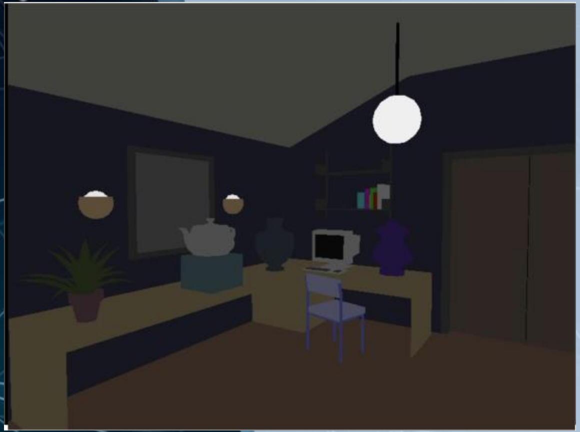
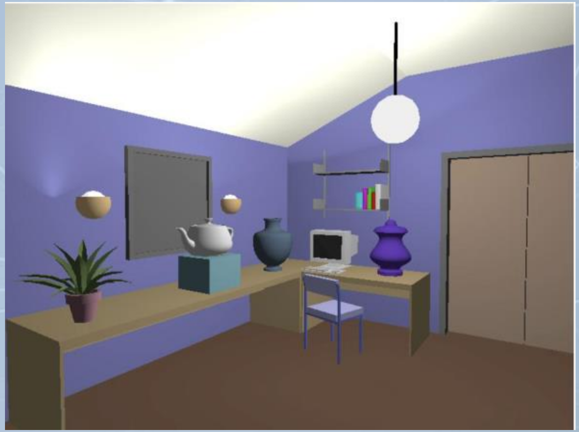
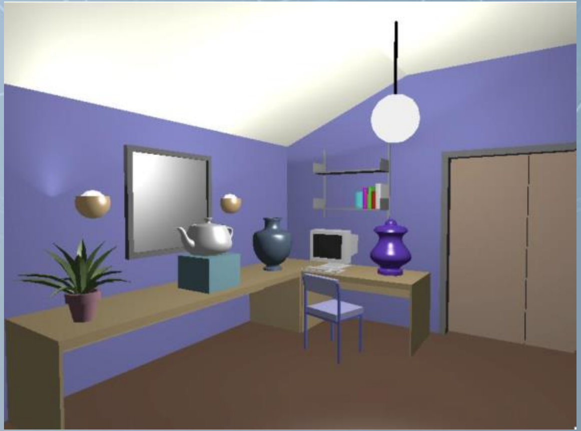
Illumination (Lighting)

CSU0021: Computer Graphics

Illumination

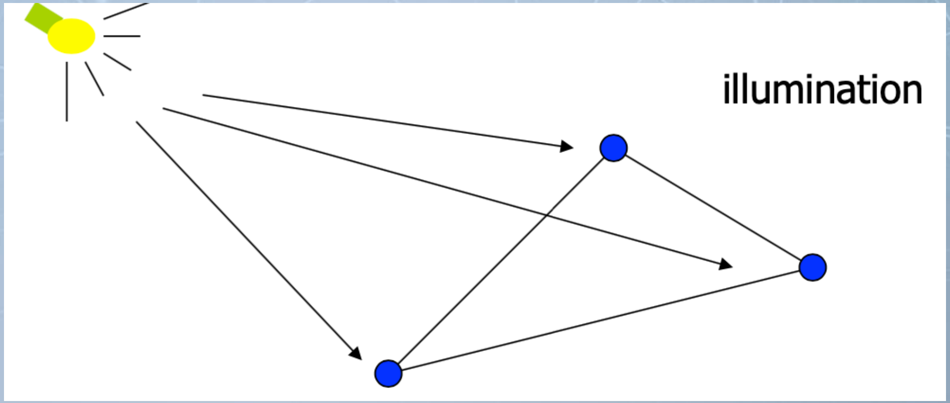






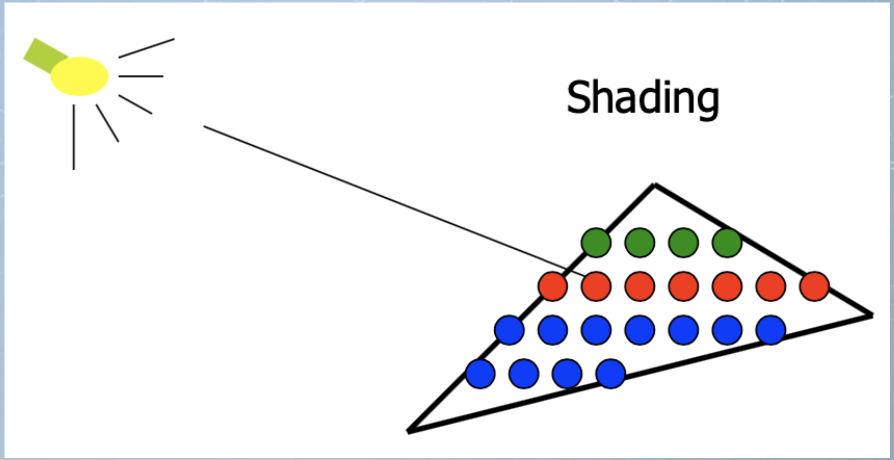
Illumination

- Model the interaction of light with surface points to determine their final color and brightness
- The illumination can be computed either at vertices or fragments



Shading

- Interpolation from the vertex illumination
- Or apply the lighting model at a set of points across entire surface

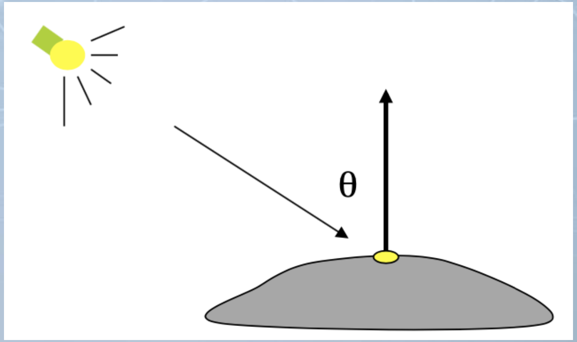


Illumination Model

- The governing principles for computing the illumination
- A illumination model usually considers:
  - Light attributes: light intensity, color, position, direction, shape
  - Object surface attributes: color, reflectivity, transparency, etc.
  - Interaction among lights and objects: object orientation
  - Interaction between objects and eye: viewing direction

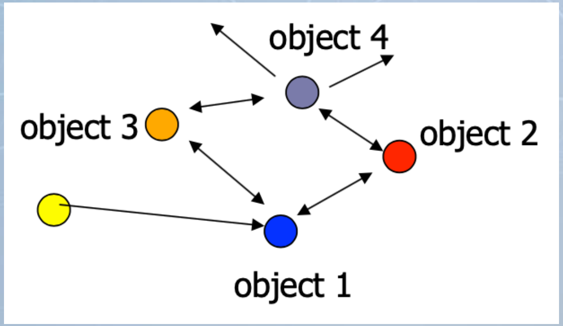
Illumination Calculation

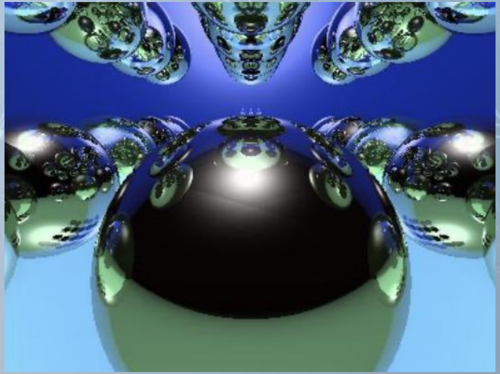
- **Local illumination**: only consider the light, the observer position, and the object material properties



Illumination Models

- Global illumination: take into account the interaction of light from all the surfaces in the scene
  - Ray Tracing (advanced computer graphics)



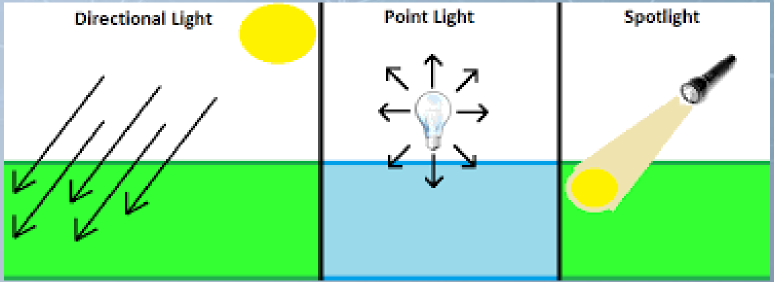


Basic Light Sources

- **Point light**
  - **Emit light to all directions**
  - **E.g. light bulb, fire. Defined by the light position and light color**

- Directional light
  - E.g. sum. Just defined by a direction and light color

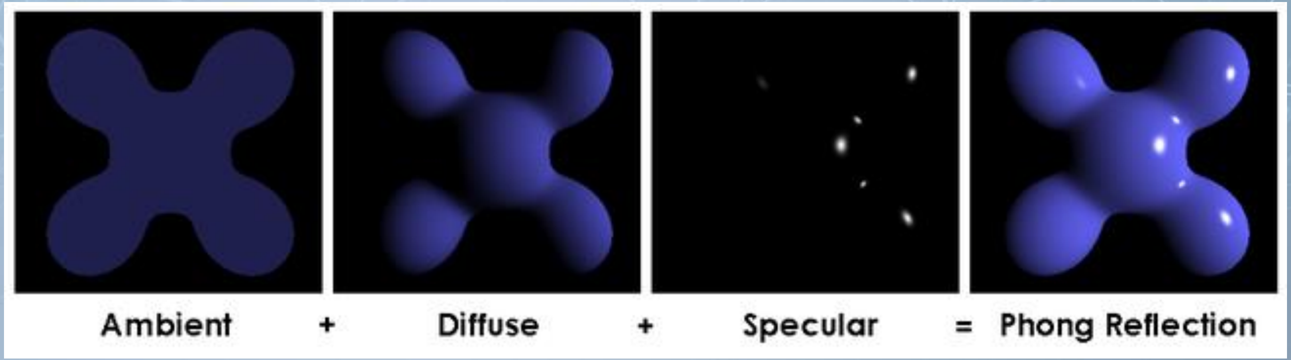
- Spot light



Light intensity can be independent or dependent of the distance between objects and the light source

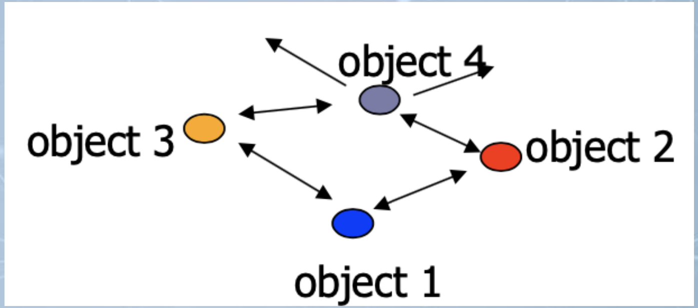
Simple Local Illumination

- Local illumination: https://en.wikipedia.org/wiki/Phong_shading#:~:text=In%203D%20computer%20graphics%2C%20Phong,or%20normal%2Dvector%20interpolation%20shading.
  - Ambient
  - Diffuse
  - Specular
- Consider these three types of light contribution to compute the final illumination of an object



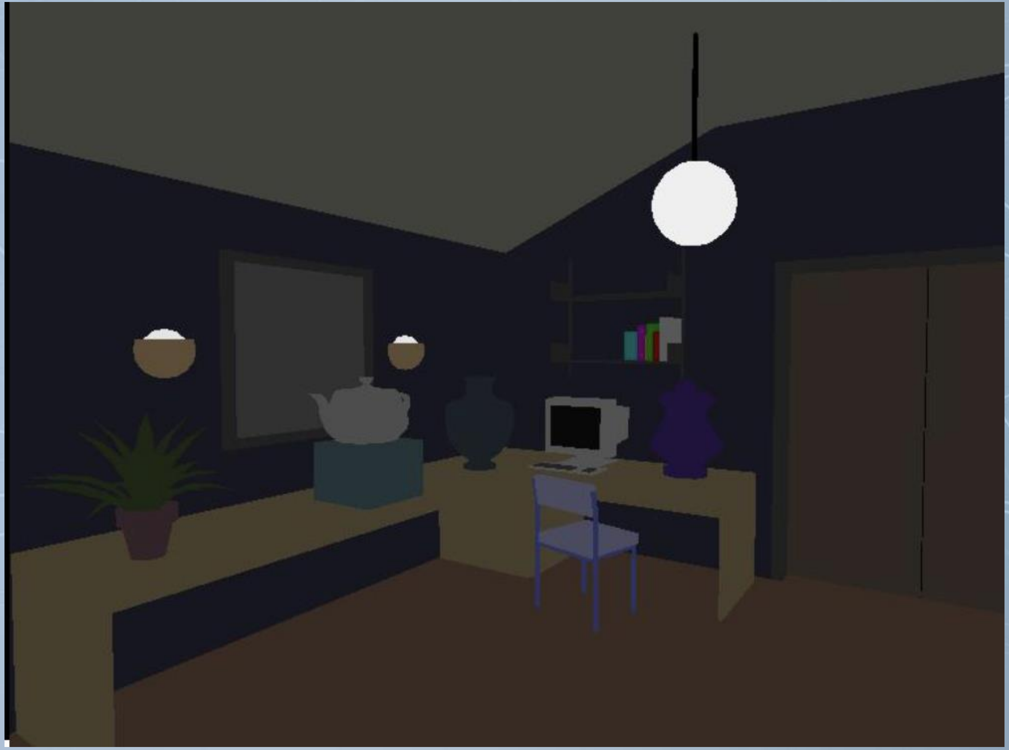
Ambient Light

- Ambient light (background light): the light that is scattered by the environment
- A **very simple approximation** of global illumination



- Independent of the light position, object orientation, observer's position or orientation – ambient light has no direction

Ambient Light Example

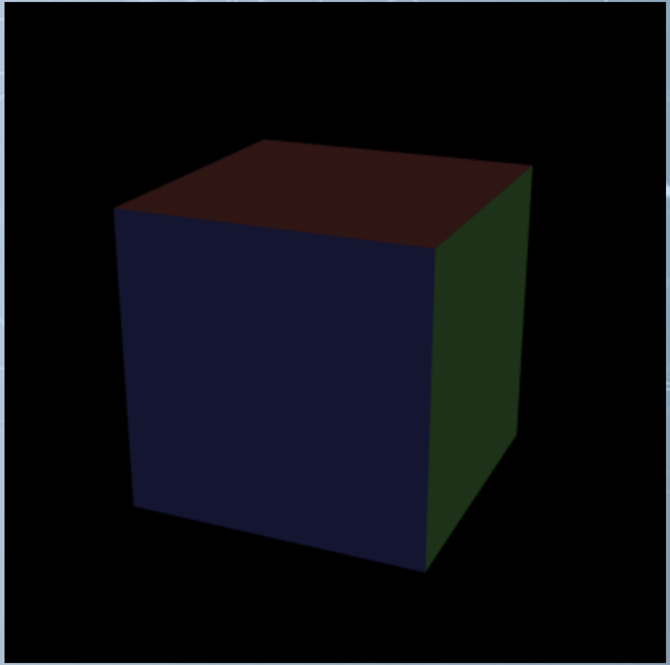


Ambient Light Calculation

- Each light source has an ambient light contribution (I_a)
- Different objects can reflect different amounts of ambient (different ambient reflection coefficient K_a, $0 \leq K_a \leq 1$)
  - Note both I_a and K_a are vectors for (R, G, B)
- So, the amount of ambient light that can be seen from an object is
  - **Ambient = $I_a * K_a$**

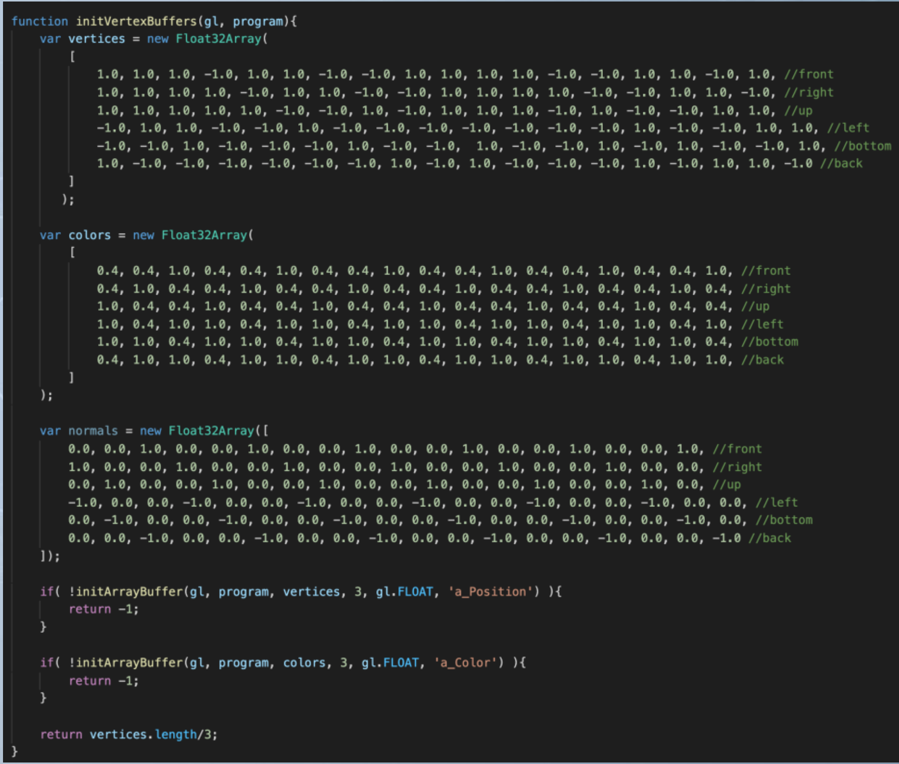
Example (Ex06-1)

- Create a cube and apply **ambient** light on it
- Files
  - index.html
  - WebGL.js
  - cuon-matrix.js



Example (Ex06-1)

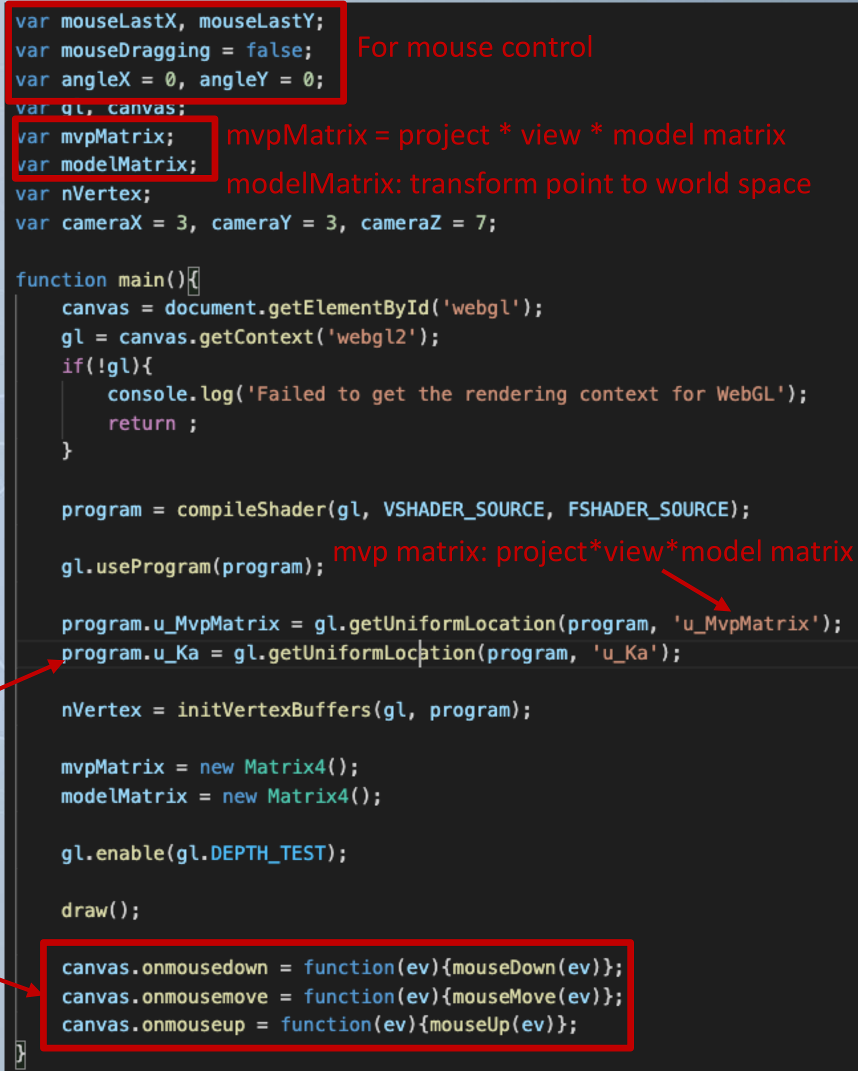
- initVertexBuffers in WebGL.js
- Data of a cube
  - One cube consists of 6 faces, each face consists of 2 triangle, and each triangle has 3 vertices
  - Each vertex has position, color and normal vector (we do not use normal vector in this example)
- Also, create vertex buffer objects in this function



```javascript
function initVertexBuffers(gl, program){
    var vertices = new Float32Array(
        [
            1.0, 1.0, 1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, //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, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, //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, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, //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, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, //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, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, //bottom
            1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0 //back
        ]
    );

    var colors = new Float32Array(
        [
            0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, //front
            0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, //right
            1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, //up
            1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, //left
            1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, //bottom
            0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, //back
        ]
    );

    var normals = new Float32Array([
        0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, //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, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, //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, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, //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, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, //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, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, //bottom
        0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0 //back
    ]);

    if( !initArrayBuffer(gl, program, vertices, 3, gl.FLOAT, 'a_Position') ){
        return -1;
    }

    if( !initArrayBuffer(gl, program, colors, 3, gl.FLOAT, 'a_Color') ){
        return -1;
    }

    return vertices.length/3;
}
```

Example (Ex06-1)

- main() in WebGL.js



```
var mouseLastX, mouseLastY;
var mouseDragging = false;
var angleX = 0, angleY = 0;
var gl, canvas;
var mvpMatrix;
var modelMatrix;
var nVertex;
var cameraX = 3, cameraY = 3, cameraZ = 7;

function main(){
    canvas = document.getElementById('webgl');
    gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

    gl.useProgram(program);

    program.u_MvpMatrix = gl.getUniformLocation(program, 'u_MvpMatrix');
    program.u_Ka = gl.getUniformLocation(program, 'u_Ka');

    nVertex = initVertexBuffers(gl, program);

    mvpMatrix = new Matrix4();
    modelMatrix = new Matrix4();

    gl.enable(gl.DEPTH_TEST);

    draw();

    canvas.onmousedown = function(ev){mouseDown(ev)};
    canvas.onmousemove = function(ev){mouseMove(ev)};
    canvas.onmouseup = function(ev){mouseUp(ev)};
}
```

For mouse control

mvpMatrix = project * view * model matrix

modelMatrix: transform point to world space

mvp matrix: project*view*model matrix

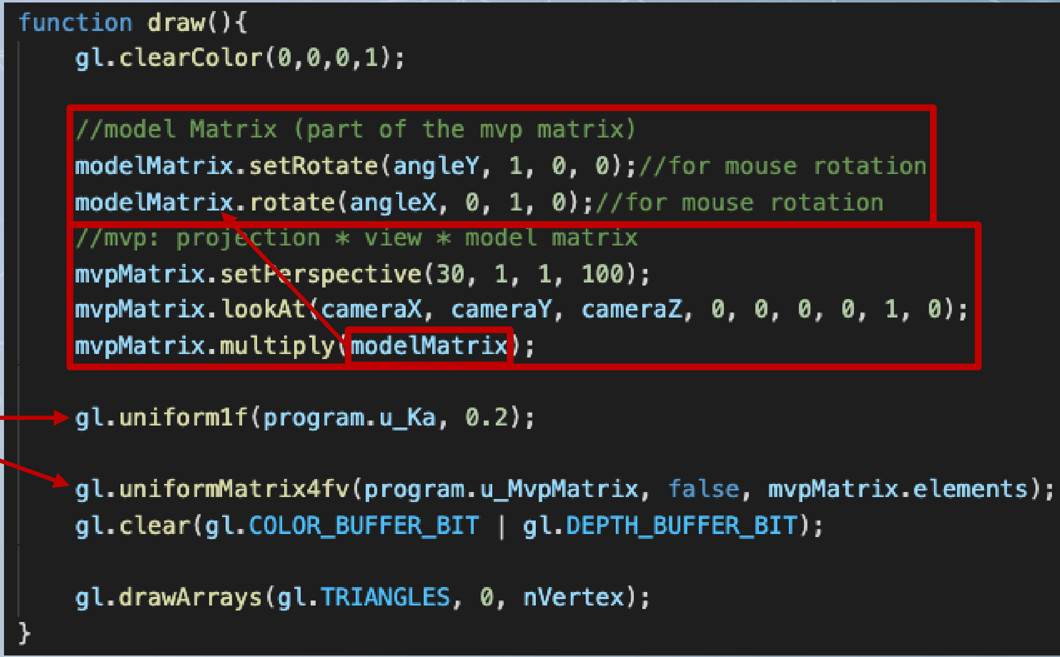
Ambient light factor: each face or each object could have different ambient factor, but I set the same value to them here

Mouse call back functions for user to rotate the objects

Example (Ex06-1)

- draw() in WebGL.js

Pass information to shaders



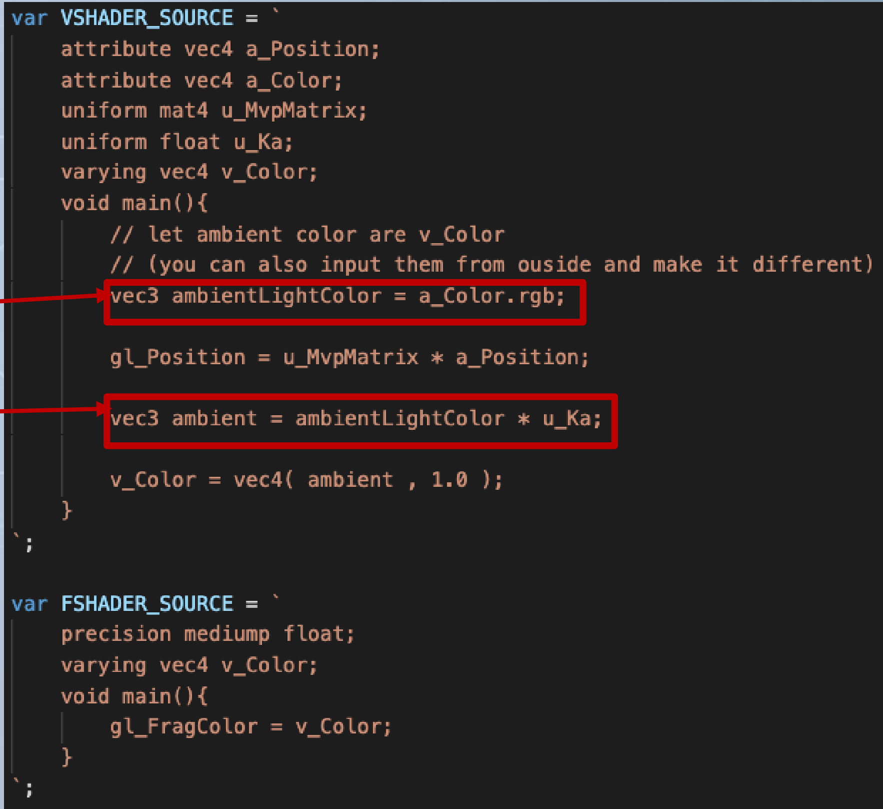
```
function draw(){
    gl.clearColor(0,0,0,1);

    //model Matrix (part of the mvp matrix)
    modelMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
    modelMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
    //mvp: projection * view * model matrix
    mvpMatrix.setPerspective(30, 1, 1, 100);
    mvpMatrix.lookAt(cameraX, cameraY, cameraZ, 0, 0, 0, 0, 1, 0);
    mvpMatrix.multiply(modelMatrix);

    gl.uniform1f(program.u_Ka, 0.2);

    gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpMatrix.elements);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    gl.drawArrays(gl.TRIANGLES, 0, nVertex);
}
```

Example (Ex06-1)

- Shaders in WebGL.js

Use a_Color as the ambient light color

Calculate ambient color



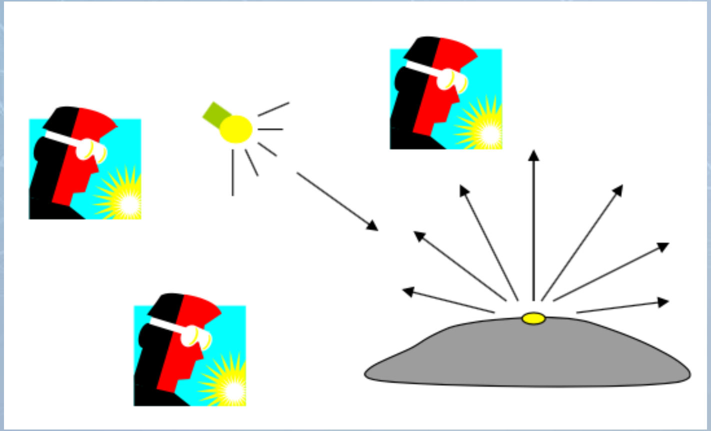
```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Color;
    uniform mat4 u_MvpMatrix;
    uniform float u_Ka;
    varying vec4 v_Color;
    void main(){
        // let ambient color are v_Color
        // (you can also input them from ouside and make it different)
        vec3 ambientLightColor = a_Color.rgb;

        gl_Position = u_MvpMatrix * a_Position;

        vec3 ambient = ambientLightColor * u_Ka;

        v_Color = vec4( ambient , 1.0 );
    }
`;

var FSHADER_SOURCE = `
    precision mediump float;
    varying vec4 v_Color;
    void main(){
        gl_FragColor = v_Color;
    }
`;
```

Let’s try and think (5mins)

- Try to modify the ambient factor
  - Can the three element in ambient factor be different? If so, what happens?

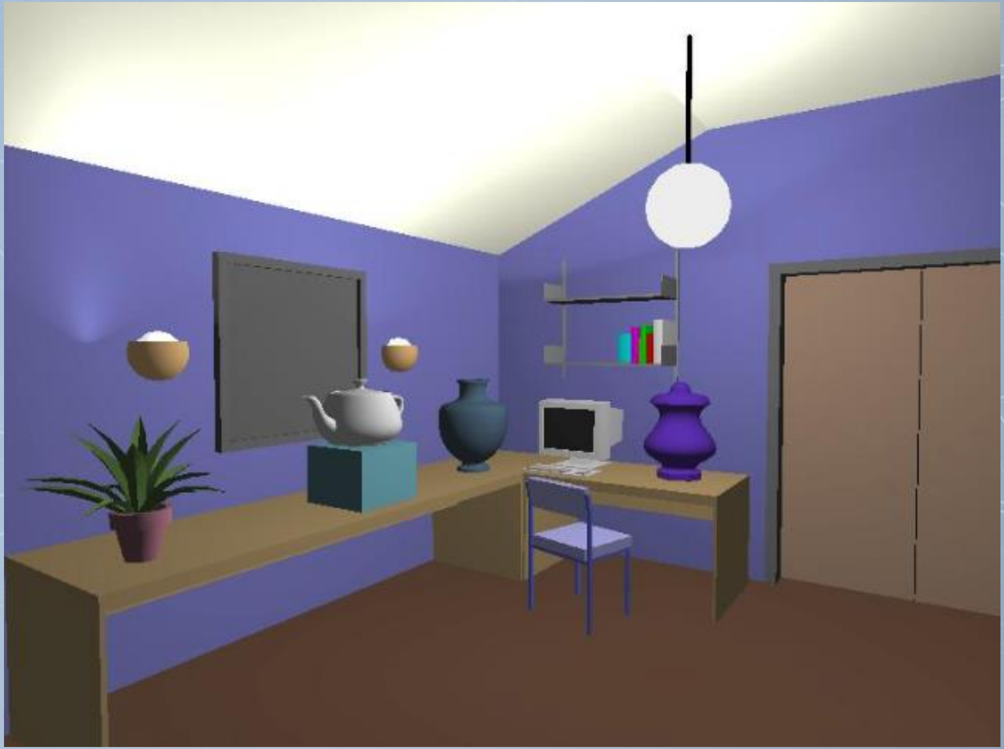
Diffuse Light

- Diffuse light: The illumination that a surface receives from a light source and reflect in all direction



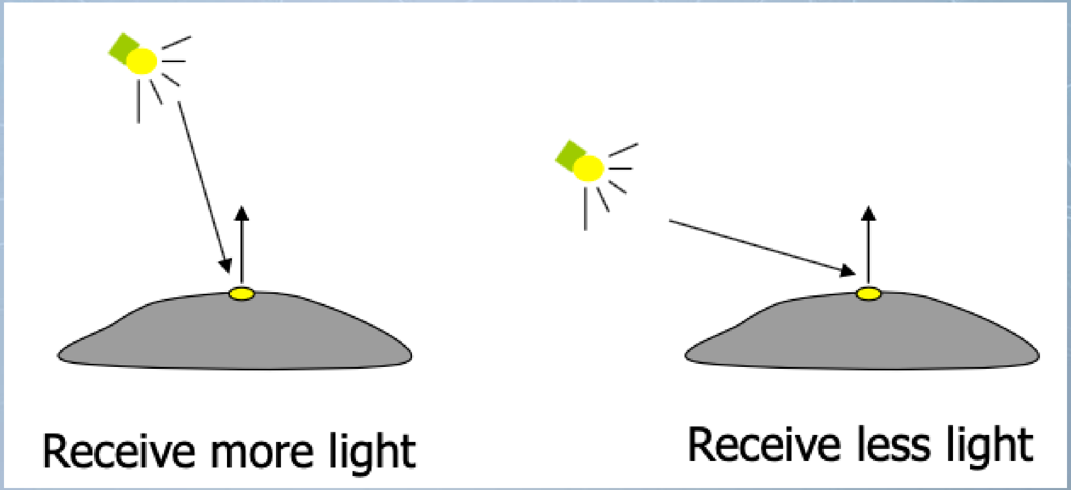
It does not matter where the eye is

Diffuse Lighting Example



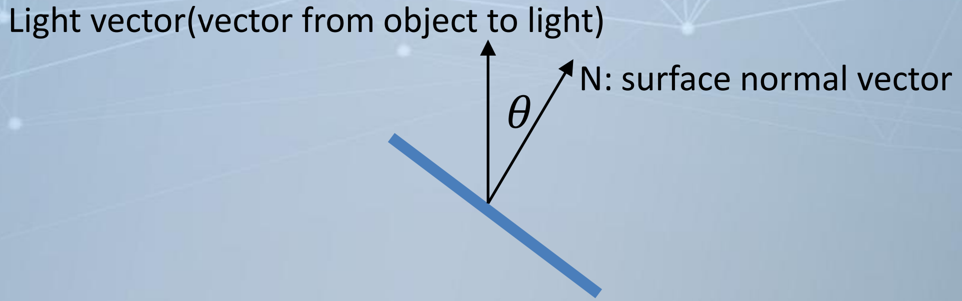
Diffuse Light Calculation

- Need to decide how much light the object point receive from the light source – based on Lambert's law



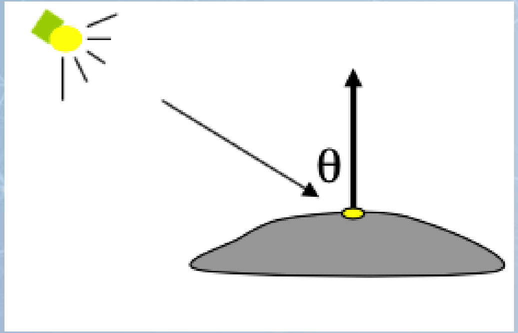
Diffuse Light Calculation

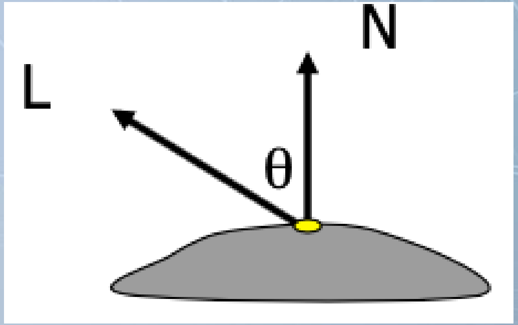
- Lambert's law: the radiant energy D that a small surface patch receives from a light source is $D = I * \cos(\theta)$
  - I: light intensity
  - θ: angle between the light vector and the surface normal



Diffuse Light Calculation

- Different objects can reflect different amount of diffuse light
  - Diffuse reflection coefficient K_d $(0 \leq K_d \leq 1)$
- So, the amount of diffuse light that can be seen is
  - $Diffuse = K_d * I_d * \cos(\theta)$



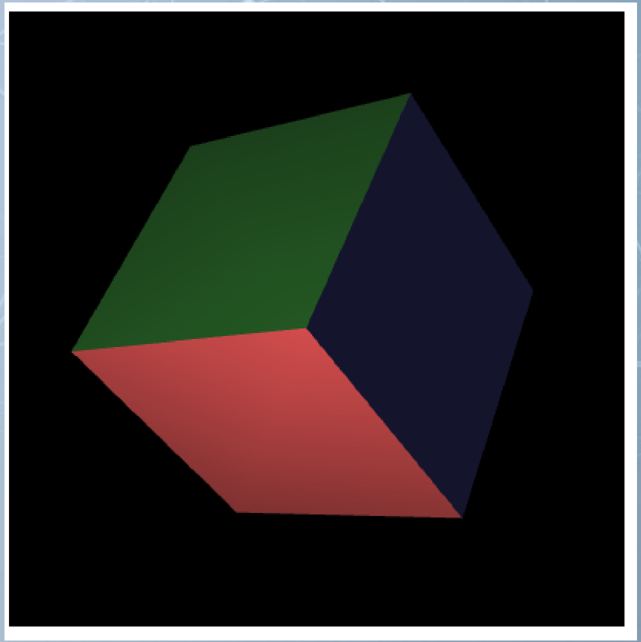


$\cos(\theta) = N \cdot L$

I recommend to calculate "diffuse" in world space.
(this means that you can set or transform all information you need to world space, then calculate)

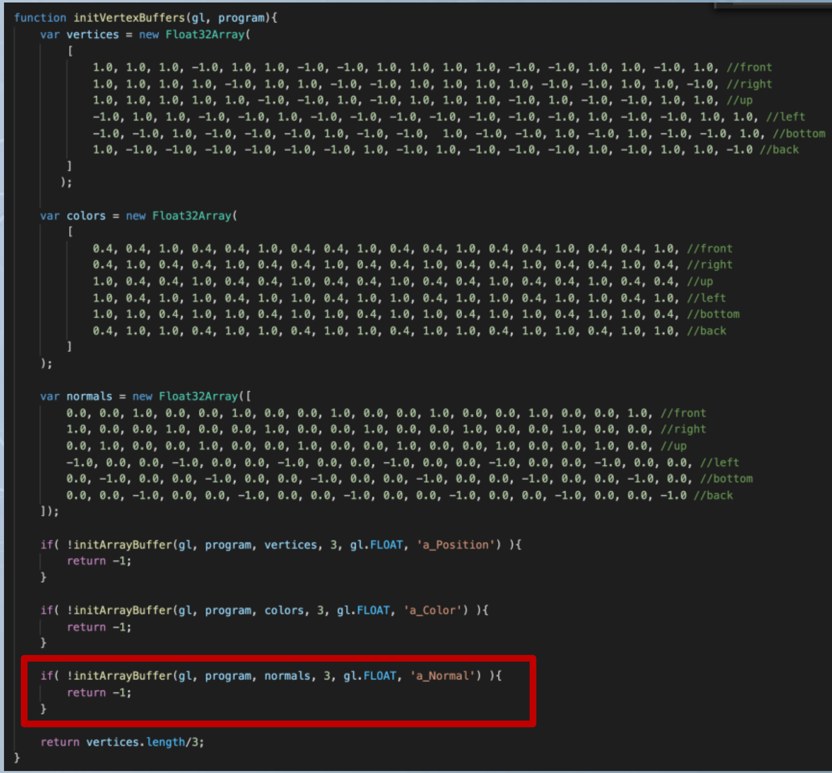
Example (Ex06-2)

- Create a cube and apply **ambient** and **diffuse** light on it
- Files
  - index.html
  - WebGL.js
  - cuon-matrix.js



Example (Ex06-2)

- initVertexBuffers in WebGL.js
- Now, we need normal vector for diffusion



```
function initVertexBuffers(gl, program){
    var vertices = new Float32Array(
        [
            1.0, 1.0, 1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, //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, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, //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, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, //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, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, //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, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, //bottom
            1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0 //back
        ]
    );

    var colors = new Float32Array(
        [
            0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, //front
            0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, //right
            1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, 1.0, 0.4, 0.4, //up
            1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, //left
            1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, //bottom
            0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, 0.4, 1.0, 1.0, //back
        ]
    );

    var normals = new Float32Array([
        0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, //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, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, //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, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, //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, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, //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, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, //bottom
        0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0 //back
    ]);

    if( !initArrayBuffer(gl, program, vertices, 3, gl.FLOAT, 'a_Position') ){
        return -1;
    }

    if( !initArrayBuffer(gl, program, colors, 3, gl.FLOAT, 'a_Color') ){
        return -1;
    }

    if( !initArrayBuffer(gl, program, normals, 3, gl.FLOAT, 'a_Normal') ){
        return -1;
    }

    return vertices.length/3;
}
```

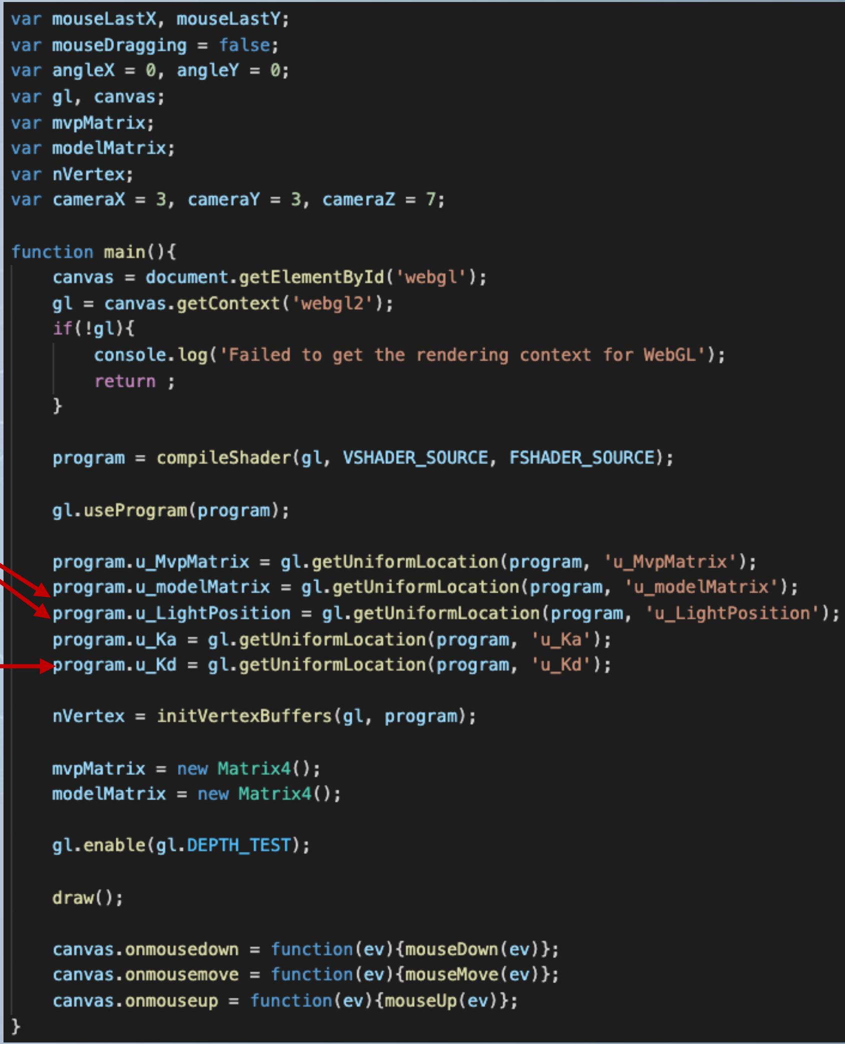
Example (Ex06-2)

- main() in WebGL.js

We have to pass model matrix and light position to shader for diffusion light calculation

Purpose of the model matrix in shader: transform vertices and normal vector to world space

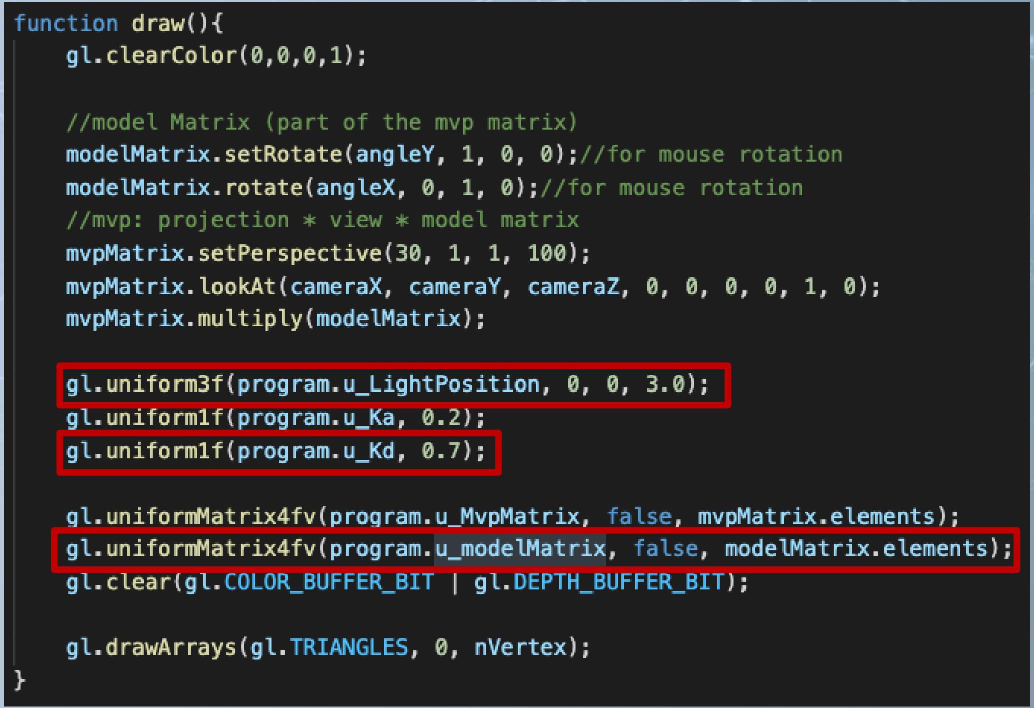
Diffusion factor



```
var mouseLastX, mouseLastY;
var mouseDragging = false;
var angleX = 0, angleY = 0;
var gl, canvas;
var mvpMatrix;
var modelMatrix;
var nVertex;
var cameraX = 3, cameraY = 3, cameraZ = 7;

function main(){
    canvas = document.getElementById('webgl');
    gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

    gl.useProgram(program);

    program.u_MvpMatrix = gl.getUniformLocation(program, 'u_MvpMatrix');
    program.u_modelMatrix = gl.getUniformLocation(program, 'u_modelMatrix');
    program.u_LightPosition = gl.getUniformLocation(program, 'u_LightPosition');
    program.u_Ka = gl.getUniformLocation(program, 'u_Ka');
    program.u_Kd = gl.getUniformLocation(program, 'u_Kd');

    nVertex = initVertexBuffers(gl, program);

    mvpMatrix = new Matrix4();
    modelMatrix = new Matrix4();

    gl.enable(gl.DEPTH_TEST);

    draw();

    canvas.onmousedown = function(ev){mouseDown(ev)};
    canvas.onmousemove = function(ev){mouseMove(ev)};
    canvas.onmouseup = function(ev){mouseUp(ev)};
}
```

Example (Ex06-2)

- draw() in WebGL.js



```
function draw(){
    gl.clearColor(0,0,0,1);

    //model Matrix (part of the mvp matrix)
    modelMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
    modelMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
    //mvp: projection * view * model matrix
    mvpMatrix.setPerspective(30, 1, 1, 100);
    mvpMatrix.lookAt(cameraX, cameraY, cameraZ, 0, 0, 0, 0, 1, 0);
    mvpMatrix.multiply(modelMatrix);

    gl.uniform3f(program.u_LightPosition, 0, 0, 3.0);
    gl.uniform1f(program.u_Ka, 0.2);
    gl.uniform1f(program.u_Kd, 0.7);

    gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpMatrix.elements);
    gl.uniformMatrix4fv(program.u_modelMatrix, false, modelMatrix.elements);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    gl.drawArrays(gl.TRIANGLES, 0, nVertex);
}
```

Example (Ex06-2)

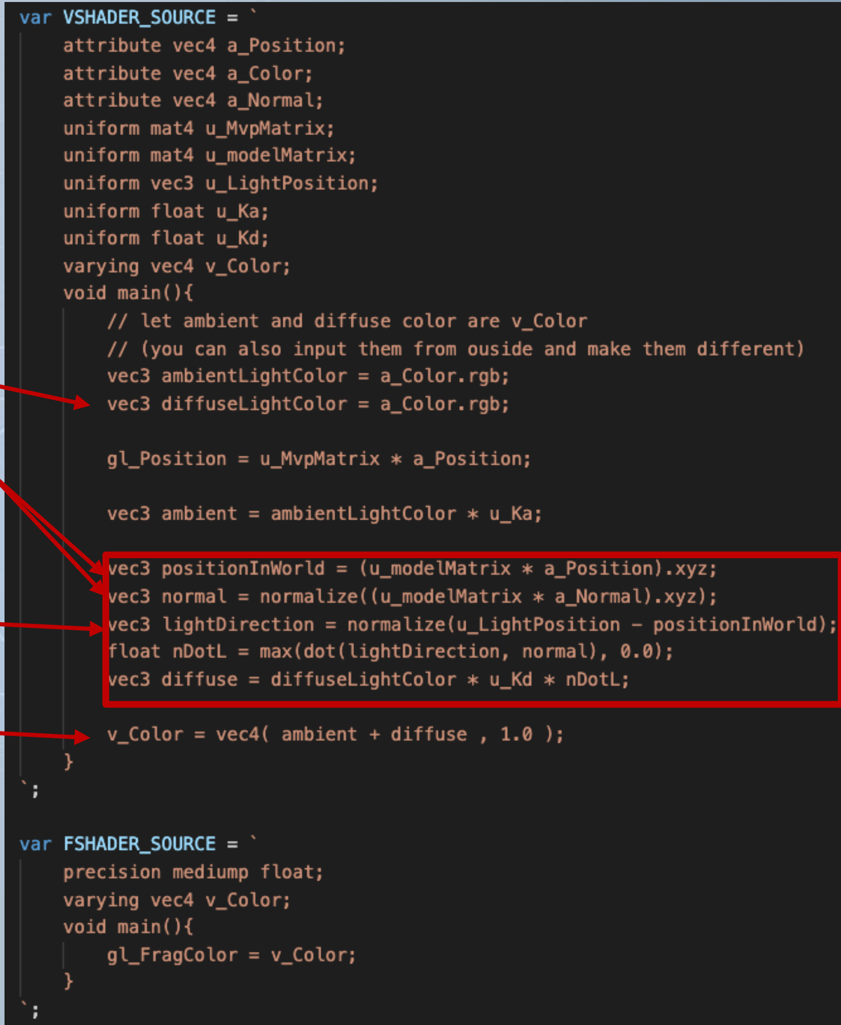
- Shaders in WebGL.js

Set diffuse light color to a_Color

The diffuse light calculation here is in world space.
We need model matrix to transform vertices and normal vector from object space to world space

Diffuse light calculation

Add ambient and diffuse light together



```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Color;
    attribute vec4 a_Normal;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform vec3 u_LightPosition;
    uniform float u_Ka;
    uniform float u_Kd;
    varying vec4 v_Color;
    void main(){
        // let ambient and diffuse color are v_Color
        // (you can also input them from ouside and make them different)
        vec3 ambientLightColor = a_Color.rgb;
        vec3 diffuseLightColor = a_Color.rgb;

        gl_Position = u_MvpMatrix * a_Position;

        vec3 ambient = ambientLightColor * u_Ka;

        vec3 positionInWorld = (u_modelMatrix * a_Position).xyz;
        vec3 normal = normalize((u_modelMatrix * a_Normal).xyz);
        vec3 lightDirection = normalize(u_LightPosition - positionInWorld);
        float nDotL = max(dot(lightDirection, normal), 0.0);
        vec3 diffuse = diffuseLightColor * u_Kd * nDotL;

        v_Color = vec4( ambient + diffuse , 1.0 );
    }
`;

var FSHADER_SOURCE = `
    precision mediump float;
    varying vec4 v_Color;
    void main(){
        gl_FragColor = v_Color;
    }
`;
```

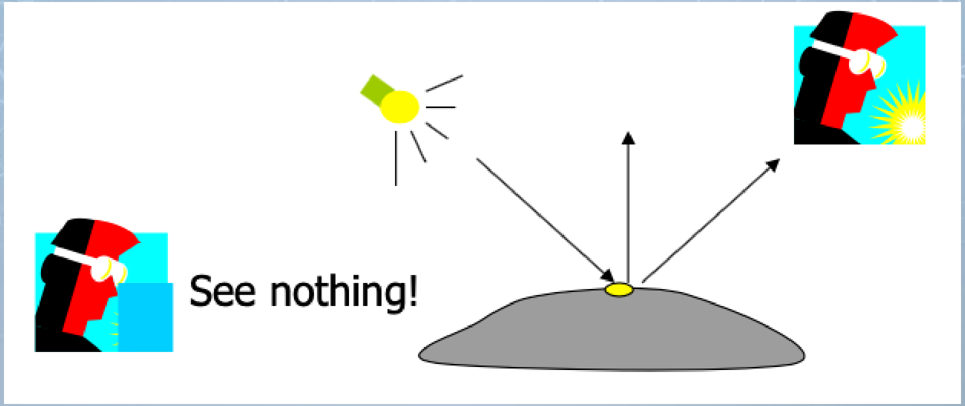
Let’s try and think (5mins)

- Try to modify the diffusion factor

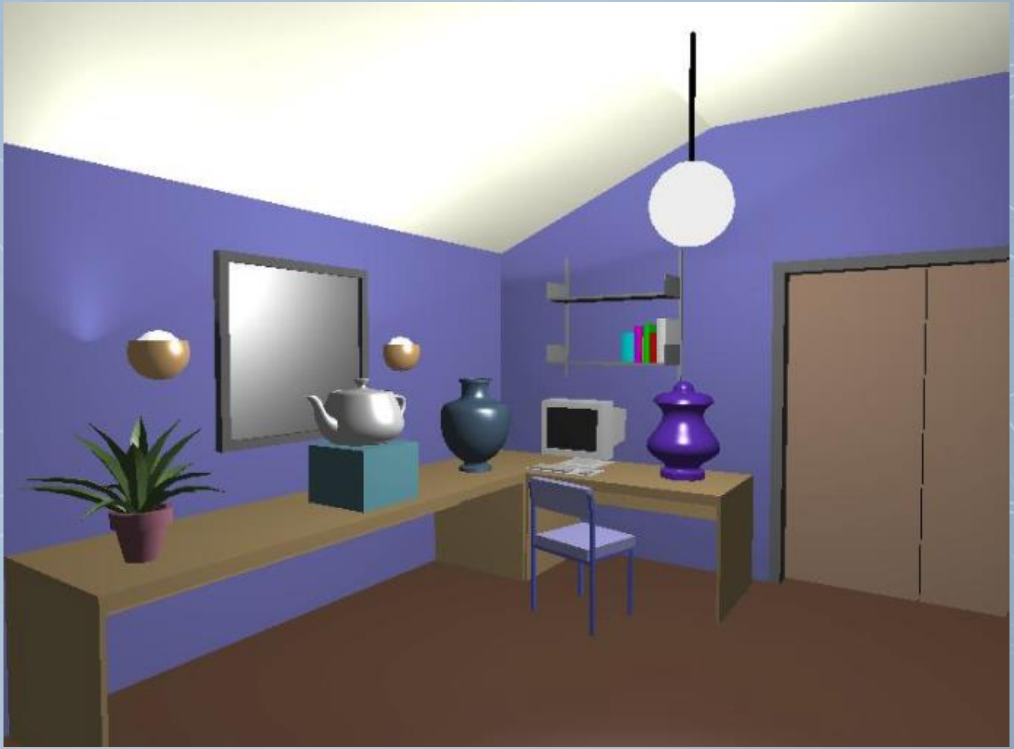
- Try to modify the light position or rotate the object

Specular Light

- The bright spot on the object
- The result of total reflection of the incident light in a concentrate region
  - Camera location matters

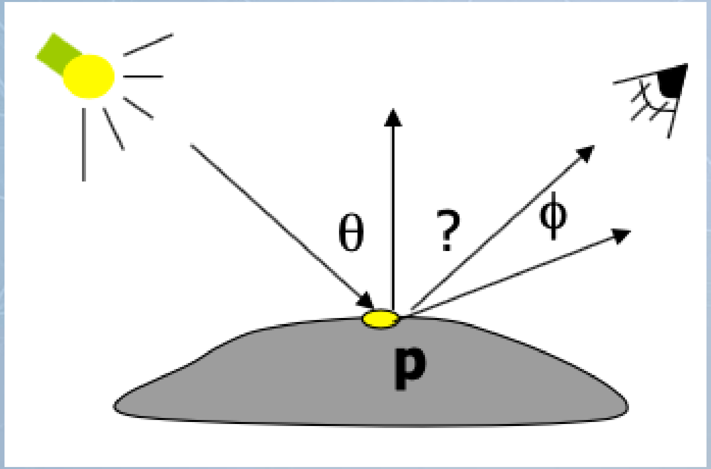


Specular Light Example



Specular Light Calculation

- How much reflection you can see depends on where you are



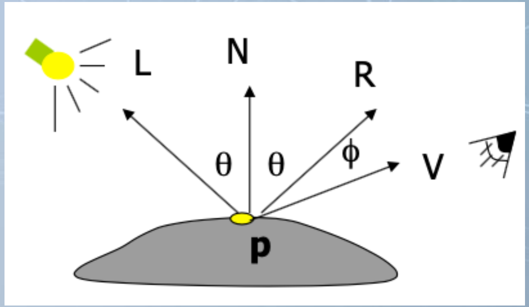
The only position the eye can see specular from p if the object has an ideal reflection surface

For a non-perfect surface you will still see specular highlight when you move a little bit away from the idea reflection direction

When ϕ is small, you see more specular highlight

Specular Light Calculation

- $Specular = K_s * I_s * \cos^n(\phi)$
  - K_s : specular reflection coefficient (a vector for [R, G, B]
  - N: surface normal at P
  - I_s :specular light intensity (a vector)
  - ϕ : angle between V and R
  - $\cos^n(\phi)$: the larger n is, the smaller value of this term is
    - $\cos(\phi) = R \cdot V$

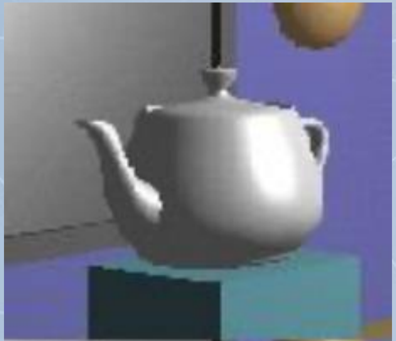


I recommend to calculate “specular” in world space.
(this means that you can set or transform all information you need to world space, then calculate)

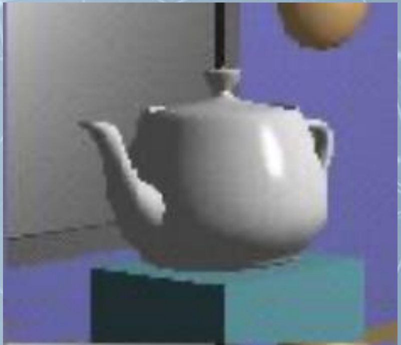
Specular Light Calculation

- The effect of ‘n’ in the model

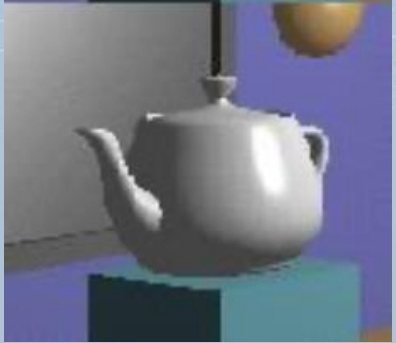
n = 10



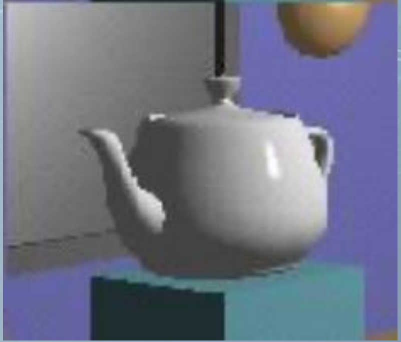
n = 90



n = 30



n = 270



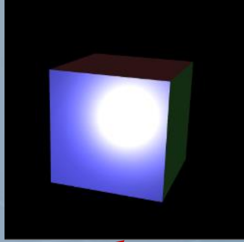
Put It All Together

- Illumination from a light
  - $illu$ = ambient + diffuse + specular

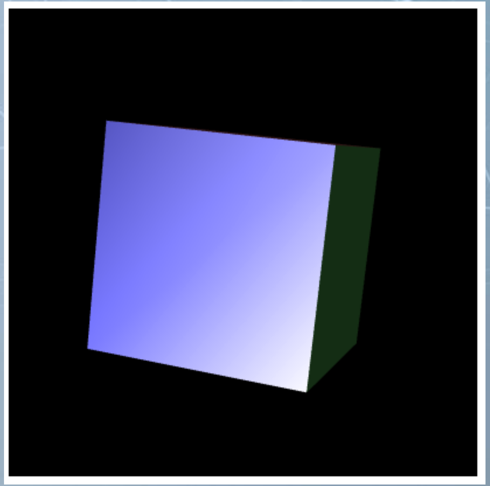
    $= \boldsymbol{K_a} * \boldsymbol{I_a} + K_d * I_d * (N \cdot L) + K_s * I_s * (R \cdot V)^n$

- If there are multiple light sources
  - Total illumination for a point = $\sum(illu)$

Example (Ex06-3)



- Create a cube and apply **ambient, diffuse** and **specular** light color on it
  - This example will **NOT** give you the specular light you expect because we implement all illumination calculation in vertex shader
    - You will implement the better version in your quiz (implement it in fragment shader)



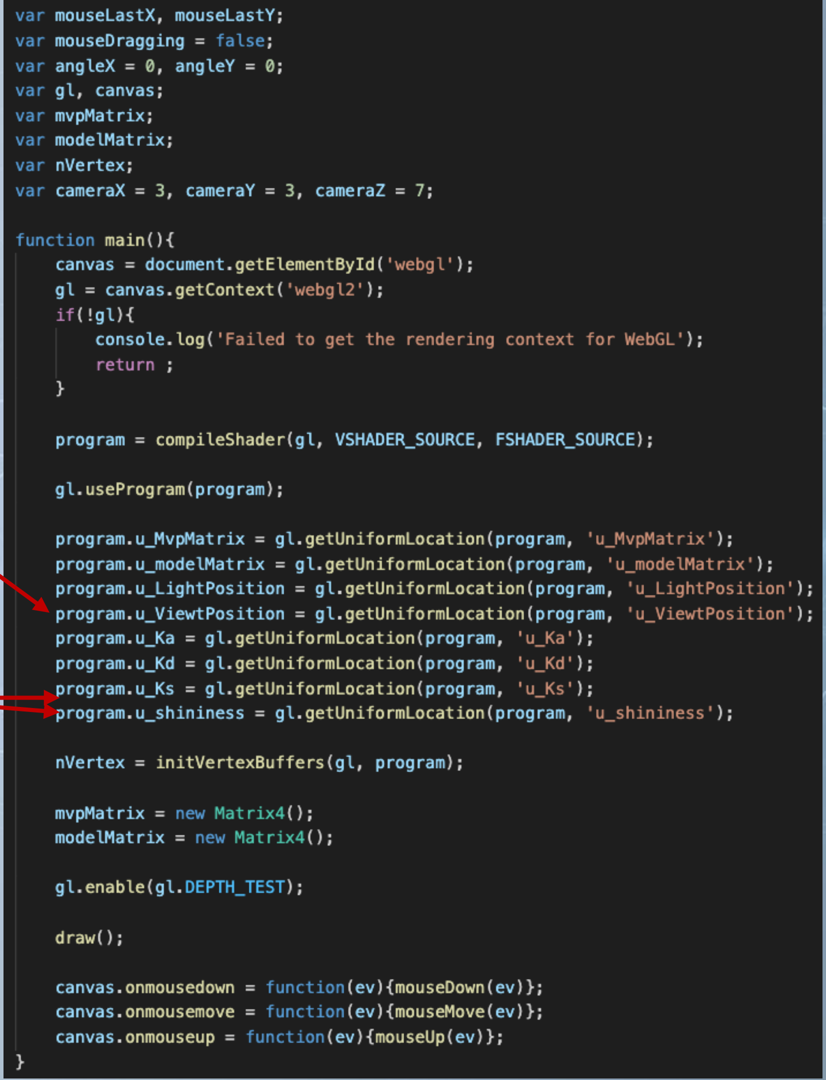
- Files
  - index.html
  - WebGL.js
  - cuon-matrix.js

Example (Ex06-3)

- main() in WebGL.js

We need view (camera) position (in world space) to calculate specular light color

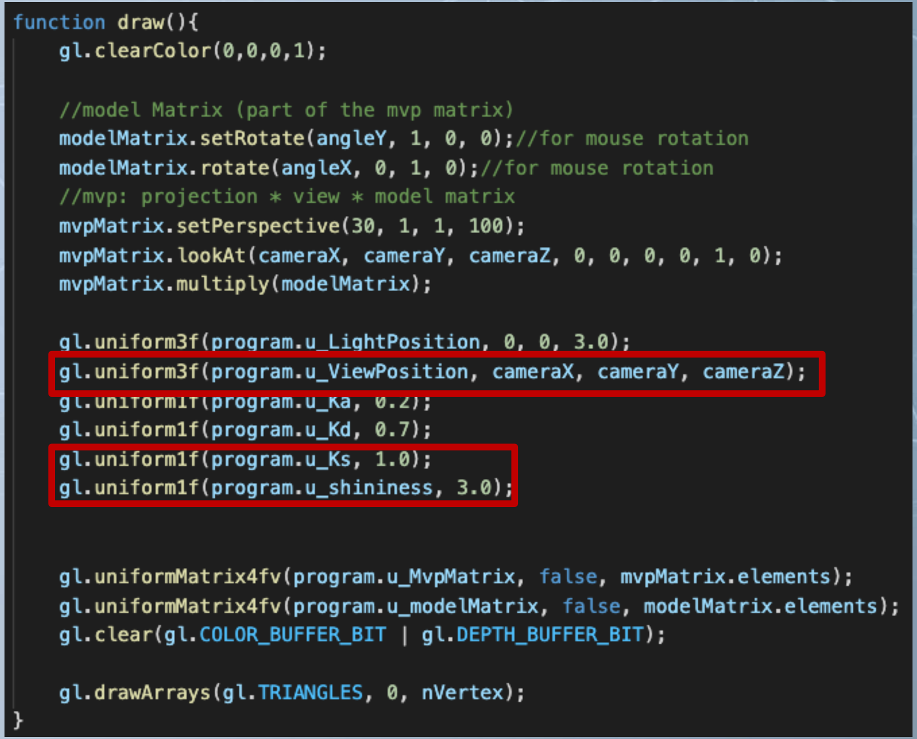
specular factor



```
var mouseLastX, mouseLastY;
var mouseDragging = false;
var angleX = 0, angleY = 0;
var gl, canvas;
var mvpMatrix;
var modelMatrix;
var nVertex;
var cameraX = 3, cameraY = 3, cameraZ = 7;

function main(){
    canvas = document.getElementById('webgl');
    gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);

    gl.useProgram(program);

    program.u_MvpMatrix = gl.getUniformLocation(program, 'u_MvpMatrix');
    program.u_modelMatrix = gl.getUniformLocation(program, 'u_modelMatrix');
    program.u_LightPosition = gl.getUniformLocation(program, 'u_LightPosition');
    program.u_ViewtPosition = gl.getUniformLocation(program, 'u_ViewtPosition');
    program.u_Ka = gl.getUniformLocation(program, 'u_Ka');
    program.u_Kd = gl.getUniformLocation(program, 'u_Kd');
    program.u_Ks = gl.getUniformLocation(program, 'u_Ks');
    program.u_shininess = gl.getUniformLocation(program, 'u_shininess');

    nVertex = initVertexBuffers(gl, program);

    mvpMatrix = new Matrix4();
    modelMatrix = new Matrix4();

    gl.enable(gl.DEPTH_TEST);

    draw();

    canvas.onmousedown = function(ev){mouseDown(ev)};
    canvas.onmousemove = function(ev){mouseMove(ev)};
    canvas.onmouseup = function(ev){mouseUp(ev)};
}
```

Example (Ex06-3)

- draw() in WebGL.js



```
function draw(){
    gl.clearColor(0,0,0,1);

    //model Matrix (part of the mvp matrix)
    modelMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
    modelMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
    //mvp: projection * view * model matrix
    mvpMatrix.setPerspective(30, 1, 1, 100);
    mvpMatrix.lookAt(cameraX, cameraY, cameraZ, 0, 0, 0, 0, 1, 0);
    mvpMatrix.multiply(modelMatrix);

    gl.uniform3f(program.u_LightPosition, 0, 0, 3.0);
    gl.uniform3f(program.u_ViewPosition, cameraX, cameraY, cameraZ);
    gl.uniform1f(program.u_Ka, 0.2);
    gl.uniform1f(program.u_Kd, 0.7);
    gl.uniform1f(program.u_Ks, 1.0);
    gl.uniform1f(program.u_shininess, 3.0);


    gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpMatrix.elements);
    gl.uniformMatrix4fv(program.u_modelMatrix, false, modelMatrix.elements);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    gl.drawArrays(gl.TRIANGLES, 0, nVertex);
}
```

Example (Ex06-3)

- Shaders in WebGL.js

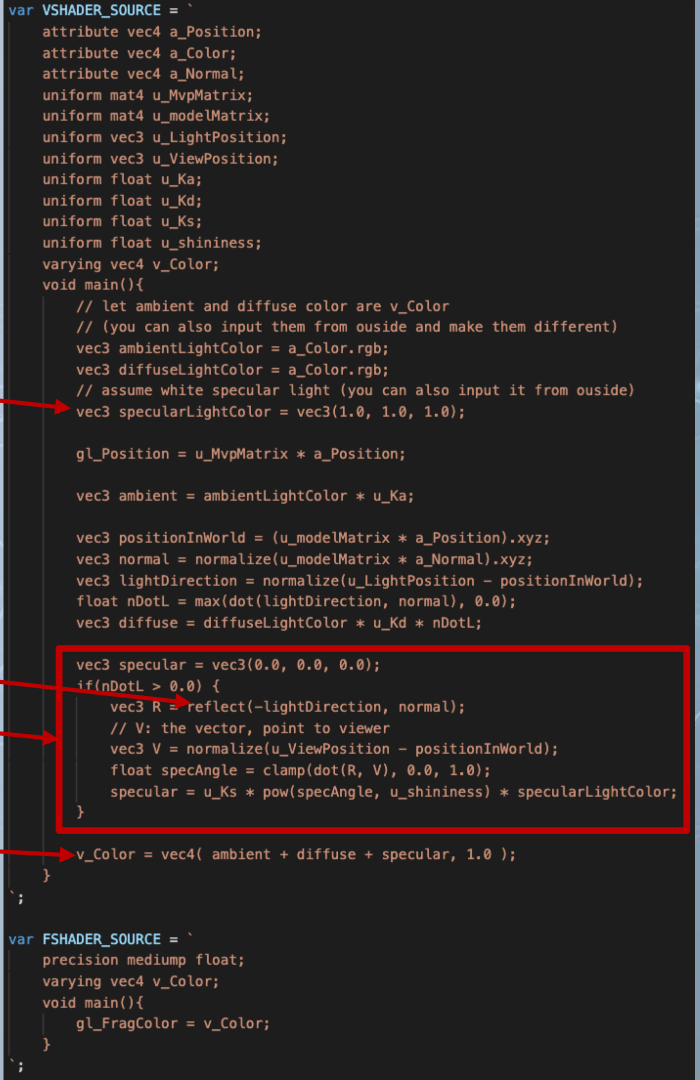
We directly set white light to specular light color

The specular calculation is in world space. So, we have to transform all information we need to world space.

GLSL build-in function to calculate the reflection vector

specular light calculation

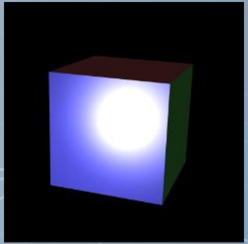
Add ambient, diffuse and specular light together



```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Color;
    attribute vec4 a_Normal;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform vec3 u_LightPosition;
    uniform vec3 u_ViewPosition;
    uniform float u_Ka;
    uniform float u_Kd;
    uniform float u_Ks;
    uniform float u_shininess;
    varying vec4 v_Color;
    void main(){
        // let ambient and diffuse color are v_Color
        // (you can also input them from ouside and make them different)
        vec3 ambientLightColor = a_Color.rgb;
        vec3 diffuseLightColor = a_Color.rgb;
        // assume white specular light (you can also input it from ouside)
        vec3 specularLightColor = vec3(1.0, 1.0, 1.0);

        gl_Position = u_MvpMatrix * a_Position;

        vec3 ambient = ambientLightColor * u_Ka;

        vec3 positionInWorld = (u_modelMatrix * a_Position).xyz;
        vec3 normal = normalize(u_modelMatrix * a_Normal).xyz;
        vec3 lightDirection = normalize(u_LightPosition - positionInWorld);
        float nDotL = max(dot(lightDirection, normal), 0.0);
        vec3 diffuse = diffuseLightColor * u_Kd * nDotL;

        vec3 specular = vec3(0.0, 0.0, 0.0);
        if(nDotL > 0.0) {
            vec3 R = reflect(-lightDirection, normal);
            // V: the vector, point to viewer
            vec3 V = normalize(u_ViewPosition - positionInWorld);
            float specAngle = clamp(dot(R, V), 0.0, 1.0);
            specular = u_Ks * pow(specAngle, u_shininess) * specularLightColor;
        }

        v_Color = vec4( ambient + diffuse + specular, 1.0 );
    }
`;

var FSHADER_SOURCE = `
    precision mediump float;
    varying vec4 v_Color;
    void main(){
        gl_FragColor = v_Color;
    }
`;
```

Let’s try and think (5mins)

A better one



- Try to modify the specular factor and shininess factor

- We still have two more problems
  - Think why this implementation does **not** give you a good enough specular light color
  - Add

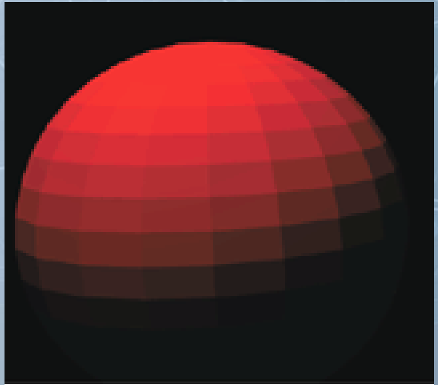
    ```
    modelMatrix.translate(0.0, 0.0, -1.0);
    modelMatrix.scale(1.0, 0.5, 2.0);
    ```

    to model matrix (add the above two line right after “modelMatrix.rotate(angleX, 0, 1, 0);”)

Polygon Shading Model – Flat Shading

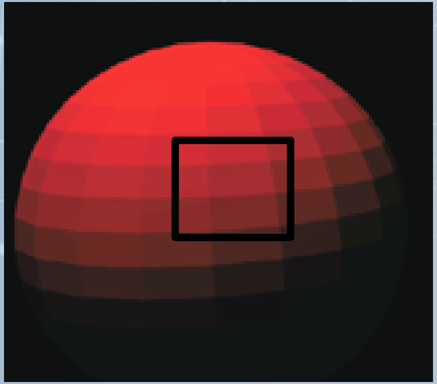
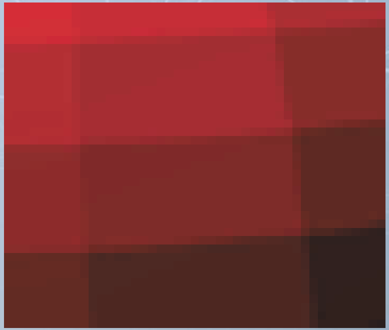
- **Flat shading** – compute lighting once and assign the color to the whole polygon
- Only use one vertex (usually the first one) normal and material property to compute the color for the polygon

- Benefit: fast
- It is used when:
  - The polygon is small enough
  - The light source is far away (why?)
  - The eye is very far away (why?)


Flat

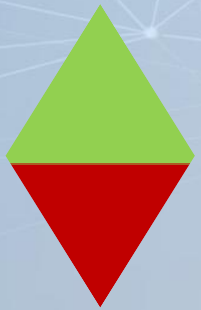
Polygon Shading Model – Flat Shading

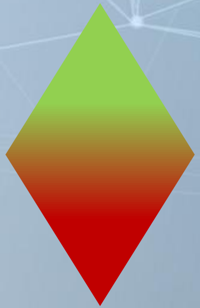
- Mach Band Effect
  - Flat shading suffers from “mach band effect”
  - Mach band effect – human eyes accentuate the discontinuity at the boundary

Smooth Shading

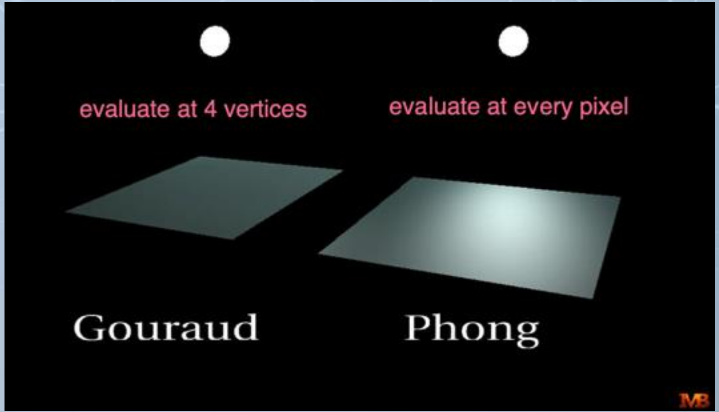
- Fix the mach band effect – remove edge discontinuity
- Compute lighting for more points on each face


Flat shading


Smooth shading

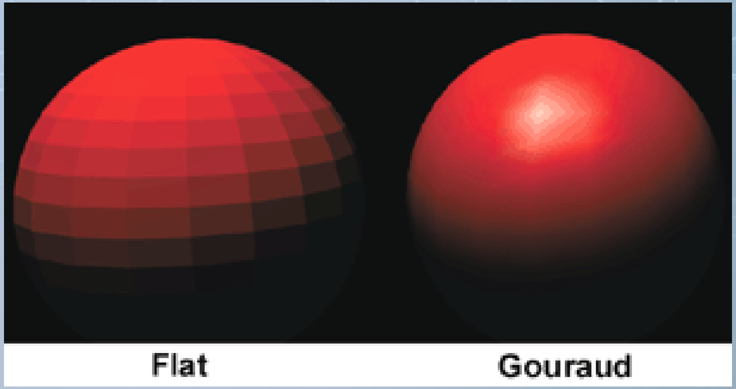
Smooth Shading

- Two popular methods:
  - Gouraud shading
  - Phong shading (better specular highlight)



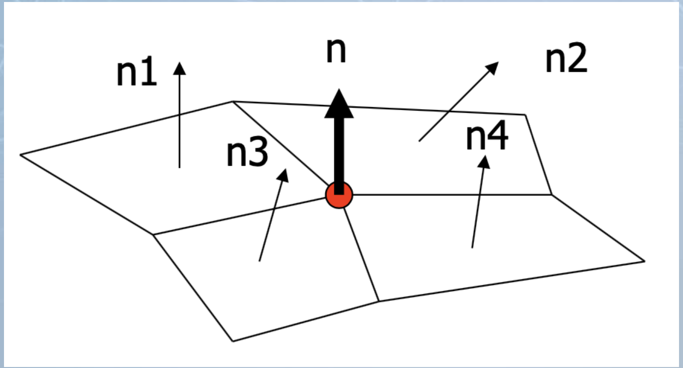
Gouraud Shading

- A smooth shading algorithm
- Lighting is calculated for each of the polygon vertices
- Colors are interpolated for interior pixels



Gouraud Shading

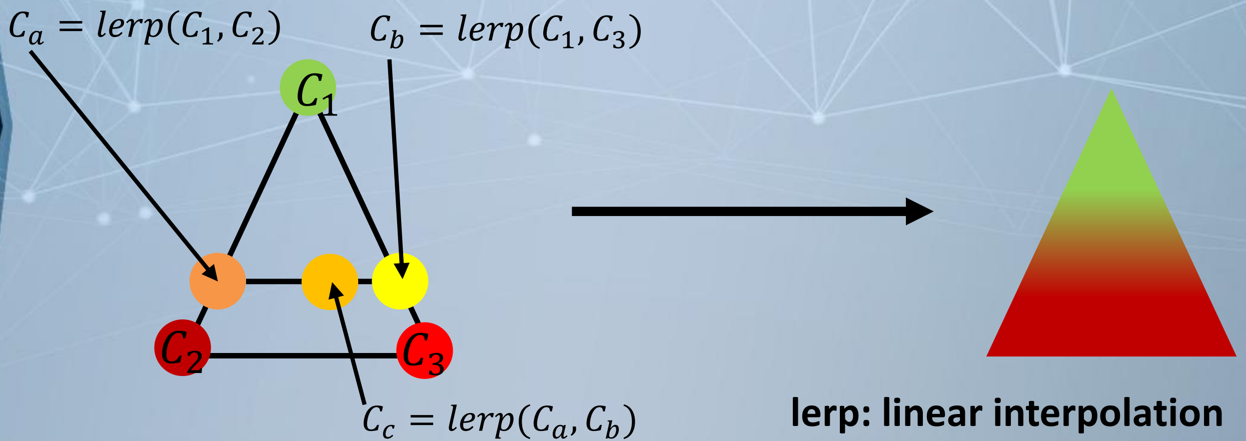
- Per-vertex lighting calculation
- Normal is needed for each vertex
- **Per-vertex normal could be computed by averaging the adjust face normals**



n = (n1+n2+n3+n4)/4.0

Gouraud Shading

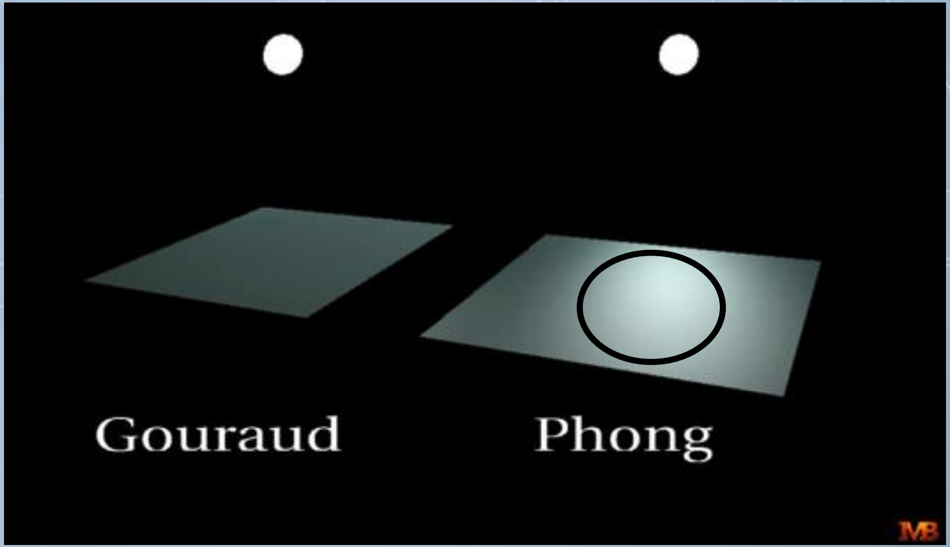
- Compute vertex illumination (color) before the projection transformation
- Shade interior pixels: color interpolation (normal are not needed)



lerp: linear interpolation

Gouraud Shading

- Problem: lighting in the polygon interior can be inaccurate



Phong Shading

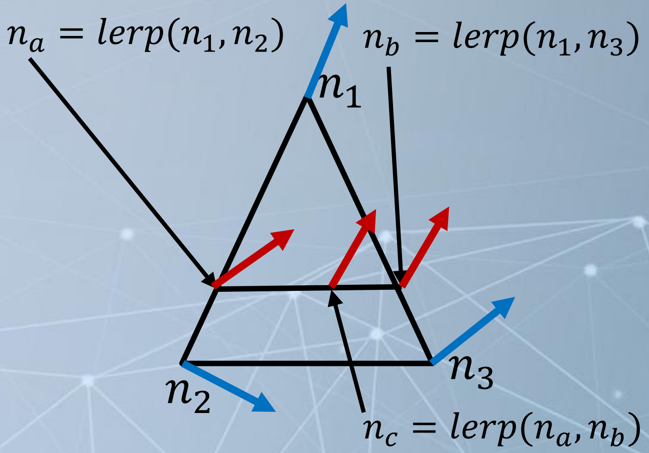
- Instead of interpolation, we calculate lighting for each pixel inside the polygon (per pixel lighting)

- We need to have normal for all the pixels – not provided by the user

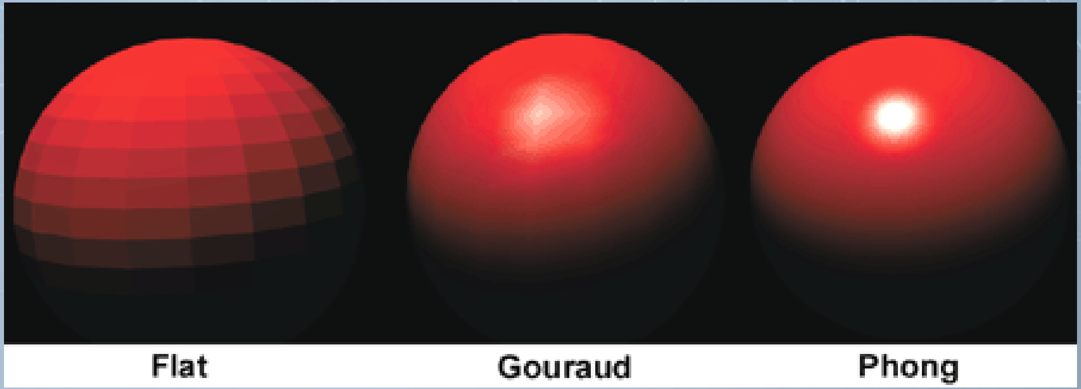
- Phong shading algorithm interpolates the normal and compute lighting during rasterization
  - Need to map the normal back to world or eye space though

Phong Shading

- Normal interpolation



- Good image quality, but need more computation

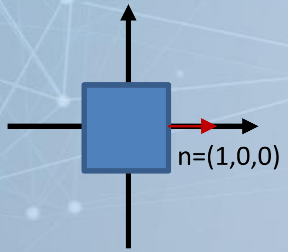


Phong Shading Implementation Concept

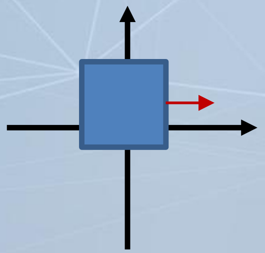
- Vertex shader
  - As usually transform vertex to clip space (gl_Position)
  - Transform **“vertex normal vector”** and **“vertex position”** to **world space**. Put them in varying variables and pass to the fragment shader
- Fragment shader
  - Because WebGL interpolate information you put in varying variables, you will receive **per-fragment ”normal vector”, “position”** in **world space**
  - With **“light position”** and **“eye position”** in **world space** (you may pass them into shaders by uniform variables), you can calculate illumination for every fragment

Transformation of Normal Vector

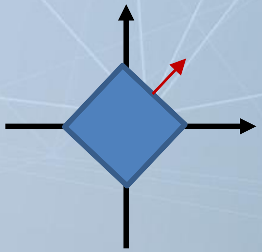
- Normal vector is one of the key components in illumination calculation
  - In the beginning, the normal vector is also defined in the object space
  - Usually you need to transform normal vectors to the world space with the object for illumination calculation
- But it may NOT be always correct if you simply apply the model matrix to normal vector directly



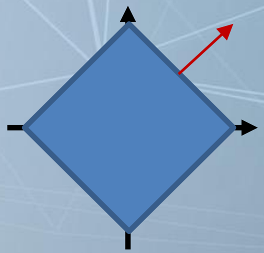
If we directly apply model matrix to normal vector



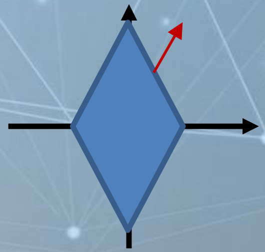
Translate along Y by 1
n = (1,1,0)
incorrect!



Rotate 45 degrees
n = (1,1,0)
OK!



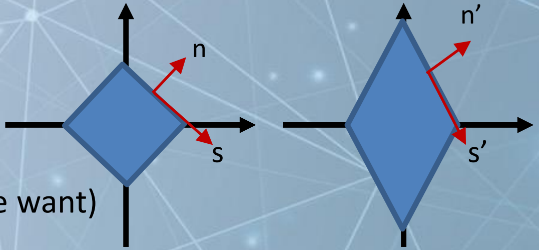
After the rotation
Scale by (sx=2, sy=2)
n = (2,2,0)
OK! If you normalize n



After the rotation
Scale by (sx=1, sy=2)
n = (1,2,0)
incorrect!

Transformation of Normal Vector

- The right matrix to transform the normal vectors
  - If M is your model matrix, use $(M^{-1})^T$ to transform the normal vectors
- Proof:
  - n: the normal vector before transformation
  - s: the surface(edge) before transformation
  - n': the **correct** normal vector after transformation
  - s': the surface(edge) after transformation
  - M: the model matrix
  - M': the **correct** matrix to transform normal vector (what we want)



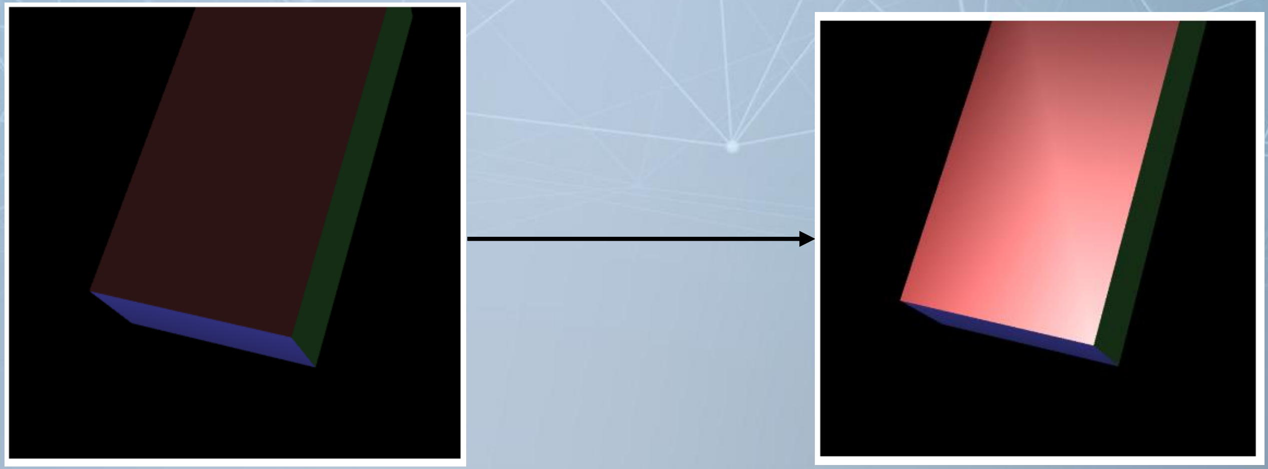
$n' \cdot s' = 0$ (they should be perpendicular with each other)
$\Rightarrow (M' * n) \cdot (M * s) = 0$
$\Rightarrow (M' * n)^T * (M * s) = 0 \quad (\because a \cdot b = a^T * b,$ where a, b are vectors)
$\Rightarrow n^T * M'^T * M * s = 0 \quad (\because (A * B)^T = B^T * A^T)$

$n \cdot s = 0$
$\Rightarrow n^T * s = 0$ $(\because a \cdot b = a^T * b,$ where a, b are vectors)

$M'^T * M = I \quad (\because n^T * M'^T * M * s' = 0,\ n^T * s = 0)$
$\Rightarrow M' = (M^{-1})^T$

Example (Ex06-4)

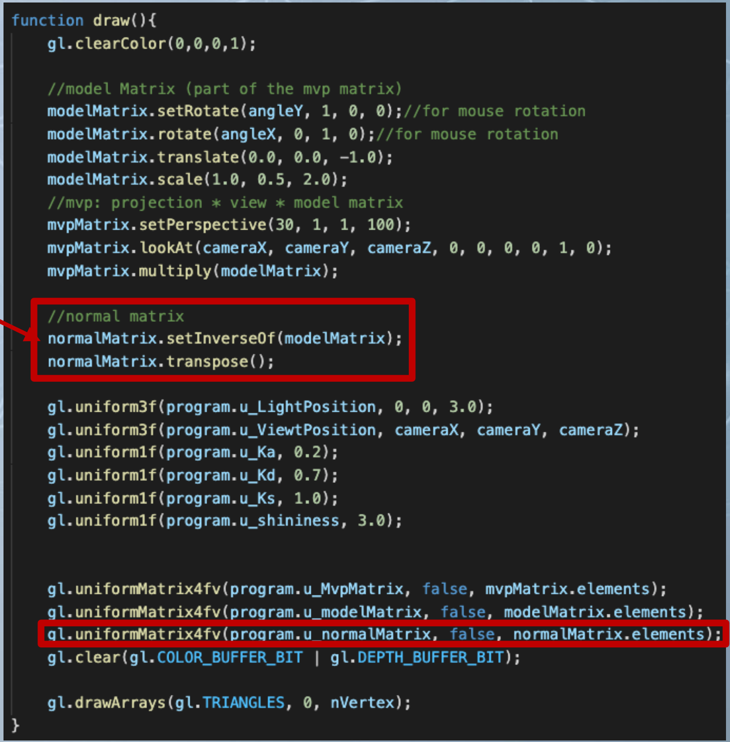
- Correct normal vector transformation and illumination
- Files
  - index.html
  - WebGL.js
  - cuon-matrix.js



Example (Ex06-4)

- draw() in WebGL.js

Calculate $(M^{-1})^T$ for normal vector transformation



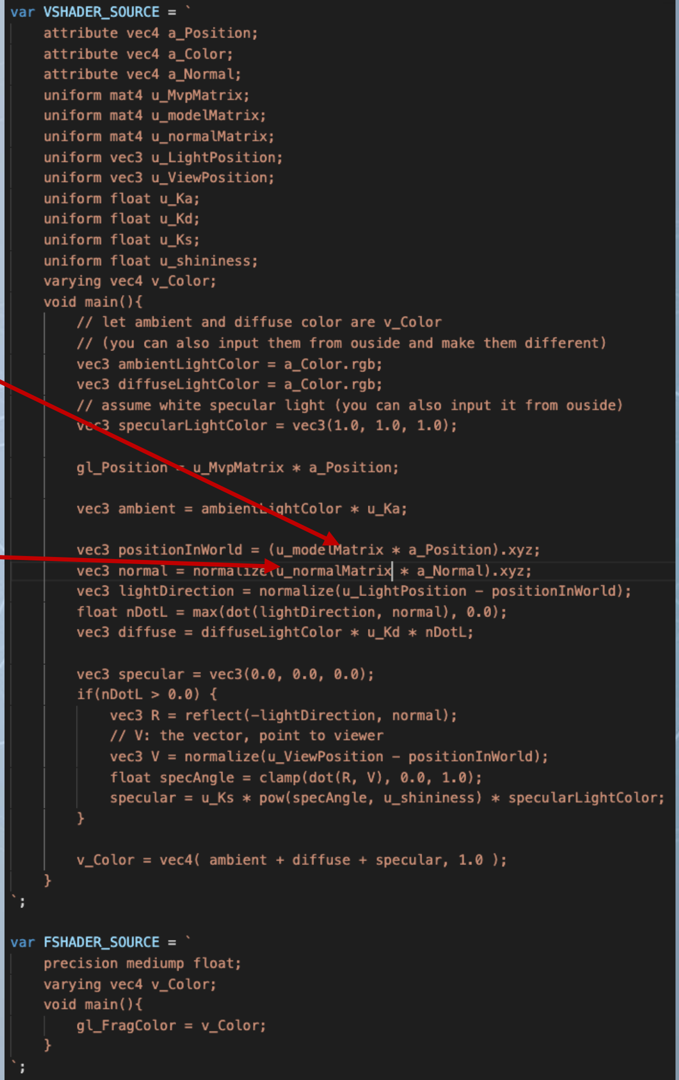
```
function draw(){
    gl.clearColor(0,0,0,1);

    //model Matrix (part of the mvp matrix)
    modelMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
    modelMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
    modelMatrix.translate(0.0, 0.0, -1.0);
    modelMatrix.scale(1.0, 0.5, 2.0);
    //mvp: projection * view * model matrix
    mvpMatrix.setPerspective(30, 1, 1, 100);
    mvpMatrix.lookAt(cameraX, cameraY, cameraZ, 0, 0, 0, 0, 1, 0);
    mvpMatrix.multiply(modelMatrix);

    //normal matrix
    normalMatrix.setInverseOf(modelMatrix);
    normalMatrix.transpose();

    gl.uniform3f(program.u_LightPosition, 0, 0, 3.0);
    gl.uniform3f(program.u_ViewtPosition, cameraX, cameraY, cameraZ);
    gl.uniform1f(program.u_Ka, 0.2);
    gl.uniform1f(program.u_Kd, 0.7);
    gl.uniform1f(program.u_Ks, 1.0);
    gl.uniform1f(program.u_shininess, 3.0);


    gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpMatrix.elements);
    gl.uniformMatrix4fv(program.u_modelMatrix, false, modelMatrix.elements);
    gl.uniformMatrix4fv(program.u_normalMatrix, false, normalMatrix.elements);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    gl.drawArrays(gl.TRIANGLES, 0, nVertex);
}
```

Example (Ex06-4)

- Shaders in WebGL.js

We still use modelMatrix to transform object vertices to world space

But, we have to use “u_normalMatrix” to correctly transform normal vectors to world space



```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec4 a_Color;
    attribute vec4 a_Normal;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform mat4 u_normalMatrix;
    uniform vec3 u_LightPosition;
    uniform vec3 u_ViewPosition;
    uniform float u_Ka;
    uniform float u_Kd;
    uniform float u_Ks;
    uniform float u_shininess;
    varying vec4 v_Color;
    void main(){
        // let ambient and diffuse color are v_Color
        // (you can also input them from ouside and make them different)
        vec3 ambientLightColor = a_Color.rgb;
        vec3 diffuseLightColor = a_Color.rgb;
        // assume white specular light (you can also input it from ouside)
        vec3 specularLightColor = vec3(1.0, 1.0, 1.0);

        gl_Position = u_MvpMatrix * a_Position;

        vec3 ambient = ambientLightColor * u_Ka;

        vec3 positionInWorld = (u_modelMatrix * a_Position).xyz;
        vec3 normal = normalize(u_normalMatrix * a_Normal).xyz;
        vec3 lightDirection = normalize(u_LightPosition - positionInWorld);
        float nDotL = max(dot(lightDirection, normal), 0.0);
        vec3 diffuse = diffuseLightColor * u_Kd * nDotL;

        vec3 specular = vec3(0.0, 0.0, 0.0);
        if(nDotL > 0.0) {
            vec3 R = reflect(-lightDirection, normal);
            // V: the vector, point to viewer
            vec3 V = normalize(u_ViewPosition - positionInWorld);
            float specAngle = clamp(dot(R, V), 0.0, 1.0);
            specular = u_Ks * pow(specAngle, u_shininess) * specularLightColor;
        }

        v_Color = vec4( ambient + diffuse + specular, 1.0 );
    }
`;

var FSHADER_SOURCE = `
    precision mediump float;
    varying vec4 v_Color;
    void main(){
        gl_FragColor = v_Color;
    }
`;
```

Comparing with Ex06-3, this two points are the only differences