```
<!DOCTYPE html>
<html>

// Create 4 buttons
<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>
<button id = "ButtonT">Toggle Rotation</button>

// Vertex Shader
<script id="vertex-shader" type="x-shader/x-vertex">
attribute  vec4 vPosition;   // 3D coordinates of a vertex
attribute  vec3 vNormal;     // The normal vector associated the vertex
varying    vec4 fColor;      // The color associated the vertex. The type "varying"
                             // tells the rasterization process that the color should
                             // be interpolated across the polygon it belongs to.

// The "uniform" type variables (or thinking them as properties) will be applied to
// all graphical primitives stored in the current active buffer.
uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
uniform float shininess;

void main()
{

    // vPosition is vec4 type, thus a homogeneous 3D coordinate.
    // The computation "(modelViewMatrix * vPosition)" transforms vPosition from
    // the world coordinates to camera coordinates by the matrix modelViewMatrix.
    // The result is still a homogeneous coordinates in vec4.
    // The operation "( ).xyz" extracts x, y, and z (thus discarding w) components
    // and stores them in the variable of vec3 type. In other words, this is the
    // conversion from homogeneous coordinates to normal coordinates.
    vec3 pos = -(modelViewMatrix * vPosition).xyz;

    vec3 light = lightPosition.xyz;           // conversion from vec4 to vec3

    vec3 L = normalize( light - pos );        // vector pointing from the vertex
                                              // toward the light


    vec3 E = normalize( -pos );    // The viewer vector V in the lecture note.
                                   // That is, the vector pointing from the vertex
                                   // toward the camera (i.e. the origin).

    vec3 H = normalize( L + E );   // Compute the halfway vector, i.e. the average
                                   // (L + E)/2.0 then normalize. Because it is to
                                   // be normalized, division by 2.0 is not necessary.

    vec4 NN = vec4(vNormal,0);     // conversion from vec3 to vec4.

    // Transform vertex normal into eye coordinates

    vec3 N = normalize( (modelViewMatrix*NN).xyz);

    // Compute terms in the illumination equation
    vec4 ambient = ambientProduct;      // The ambient term
```

```
    float Kd = max( dot(L, N), 0.0 );   // if the dot-product is negative, the
                                        // light is from the back side of the
                                        // polygon. So use 0.0.
    vec4  diffuse = Kd*diffuseProduct;  // The diffuse term

    float Ks = pow( max(dot(N, H), 0.0), shininess );
    vec4  specular = Ks * specularProduct;    // The specular term

    if( dot(L, N) < 0.0 ) {             // If the light is from the back,
      specular = vec4(0.0, 0.0, 0.0, 1.0);    // set the specular term to 0's
    }

    gl_Position = projectionMatrix * modelViewMatrix * vPosition;
    fColor = ambient + diffuse +specular;     // Putting the three term together.

    fColor.a = 1.0;                     // fColor = [r, g, b, a] where a is the
                                        // transparency.
}
</script>
// End of Vertex Sahder

<script id="fragment-shader" type="x-shader/x-fragment">

#ifdef GL_ES
precision highp float;
#endif


varying vec4 fColor;

void
main()
{
    gl_FragColor = fColor;
}
</script>

<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="shadedCube.js"></script>

<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```