OpenGL Lighting

> **Complete guide for Section 9 + Lab 3**
> Includes: Lights, Materials, Normals, Shading, Lighting Equation
> **Tips**: Always `glEnable(GL_LIGHTING)` and `GL_LIGHT0`. Use `glNormal3f()` before each vertex. Set material **once** per object. Use `glLightModel*` for global ambient.

1. Enable Lighting

```
glEnable(GL_LIGHTING);        // MUST enable
glEnable(GL_LIGHT0);          // Default white directional light

// Default GL_LIGHT0: White, directional, from viewer, diffuse only
// Up to 8 lights: GL_LIGHT0 to GL_LIGHT7
```

2. Light Types & Properties

2.1 Light Properties (`glLightfv`)

```
float white[] = {1,1,1,1};
float dim[]   = {0.3,0.3,0.3,1};
float pos[]   = {5,5,5,1};   // Point light at (5,5,5)
float dir[]   = {0,0,-1,0};  // Directional light (w=0)

glLightfv(GL_LIGHT1, GL_AMBIENT,  dim);
glLightfv(GL_LIGHT1, GL_DIFFUSE,  white);
glLightfv(GL_LIGHT1, GL_SPECULAR, white);
glLightfv(GL_LIGHT1, GL_POSITION, pos);
```

Property	Meaning
`GL_AMBIENT`	Background glow
`GL_DIFFUSE`	Main color (Lambert)
`GL_SPECULAR`	Shiny highlight
`GL_POSITION`	`[x,y,z,w]` → `w=1`: point, `w=0`: directional

2.2 Global Ambient (Scene-wide)

```
float globalAmb[] = {0.1, 0.1, 0.1, 1};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, globalAmb);
```

---

3. Material Properties

```
float red[]     = {0.8, 0.0, 0.0, 1.0};
float shiny[]   = {1.0, 1.0, 1.0, 1.0};
float none[]    = {0,0,0,1};

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, red);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, shiny);
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 50.0);  // 0–128
glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, none);
```

Notes:

- `GL_AMBIENT_AND_DIFFUSE`: Shortcut (usually same)
- `GL_SHININESS`: Higher = smaller, sharper highlight
- `GL_EMISSION`: Glow (no light cast)

---

4. Normals – Critical for Lighting!

```
glNormal3f(0, 0, 1);  // Front face
glBegin(GL_QUADS);
    glVertex3f(-1,-1, 1);
    glVertex3f( 1,-1, 1);
    glVertex3f( 1, 1, 1);
    glVertex3f(-1, 1, 1);
glEnd();
```

- Normals per face → **Flat shading**
- Normals per vertex → **Smooth shading (average at vertex)**

Auto-Normal for Sphere (Example)

```
void drawSphere() {
    for (int i = 0; i <= stacks; i++) {
        float lat = M_PI * (-0.5 + (float)i / stacks);
        float sinLat = sin(lat), cosLat = cos(lat);
        glBegin(GL_QUAD_STRIP);
        for (int j = 0; j <= slices; j++) {
            float lng = 2 * M_PI * j / slices;
            float x = cos(lng) * cosLat;
            float y = sinLat;
            float z = sin(lng) * cosLat;
            glNormal3f(x, y, z);  // Normal = position (unit sphere)
            glVertex3f(r*x, r*y, r*z);
```

```
            }
            glEnd();
        }
    }
```

---

5. Lighting Equation (OpenGL 1.1)

For each light and each vertex:

```
I = Ambient + Diffuse + Specular
```

Ambient

```
I_amb = light_ambient × material_ambient
```

Diffuse (Lambert)

```
I_diff = light_diffuse × material_diffuse × max(0, N·L)
```

Specular (Phong)

```
R = 2(N·L)N - L
I_spec = light_specular × material_specular × max(0, R·V)^shininess
```

Where:

- `N`: Surface normal (unit)
- `L`: Light direction (from surface to light)
- `V`: View direction (from surface to camera)
- `R`: Reflection vector

---

6. Shading Types

Type	How
Flat	One normal per face → `GL_FLAT`
Smooth	Normals per vertex → `GL_SMOOTH` (default)

```
glShadeModel(GL_FLAT);   // Faceted
glShadeModel(GL_SMOOTH); // Default
```

---

7. Complete Lit Cube Example

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(3,4,5, 0,0,0, 0,1,0);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    // Light
    float lightPos[] = {5,5,5,1};
    float white[] = {1,1,1,1};
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
    glLightfv(GL_LIGHT0, GL_SPECULAR, white);

    // Material
    float red[] = {0.8,0.1,0.1,1};
    float shiny[] = {1,1,1,1};
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, red);
    glMaterialfv(GL_FRONT, GL_SPECULAR, shiny);
    glMaterialf(GL_FRONT, GL_SHININESS, 80);

    drawCube(1.0);  // With normals!

    glutSwapBuffers();
}
```

---

8. Lab 3 – WebGL Shaders (Modern OpenGL)

Use **Lab 3.1.html → Lab 3.2.html → Lab 3.3.html**

Lab 3.1: Solid Color + Rotation

Vertex Shader:

```
attribute vec3 a_position;
uniform mat4 u_modelview;
uniform mat4 u_projection;

void main() {
```

```
    gl_Position = u_projection * u_modelview * vec4(a_position, 1.0);
}
```

Fragment Shader:

```
precision mediump float;
void main() {
    gl_FragColor = vec4(0.5, 0.7, 1.0, 1.0);  // Solid blue
}
```

JS Rotation Example:

```
let angle = 0;
function render() {
    angle += 0.01;
    mat4.identity(modelview);
    mat4.translate(modelview, modelview, [0,0,-5]);
    mat4.rotateY(modelview, modelview, angle);

    gl.uniformMatrix4fv(u_modelview, false, modelview);
    gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED_SHORT, 0);
    requestAnimationFrame(render);
}
```

Lab 3.2: Lambert + Cell Shading

Vertex Shader:

```
attribute vec3 a_position;
attribute vec3 a_normal;
uniform mat4 u_modelview;
uniform mat4 u_projection;
uniform mat4 u_normalMatrix;  // inverse transpose

varying vec3 v_normal;
varying vec3 v_lightDir;

void main() {
    vec4 pos = u_modelview * vec4(a_position, 1.0);
    v_normal = normalize((u_normalMatrix * vec4(a_normal, 0.0)).xyz);
    v_lightDir = normalize(vec3(5,5,5) - pos.xyz);  // Light pos
    gl_Position = u_projection * pos;
}
```

Fragment Shader (Cell Shading):

```
precision mediump float;
varying vec3 v_normal;
varying vec3 v_lightDir;

void main() {
    float NdotL = max(0.0, dot(v_normal, v_lightDir));

    // Cell shading: quantize to 4 levels
    float levels = 4.0;
    float shade = floor(NdotL * levels) / levels;

    vec3 color = vec3(0.2, 0.6, 0.8);  // Base
    gl_FragColor = vec4(color * shade, 1.0);
}
```