Shaders

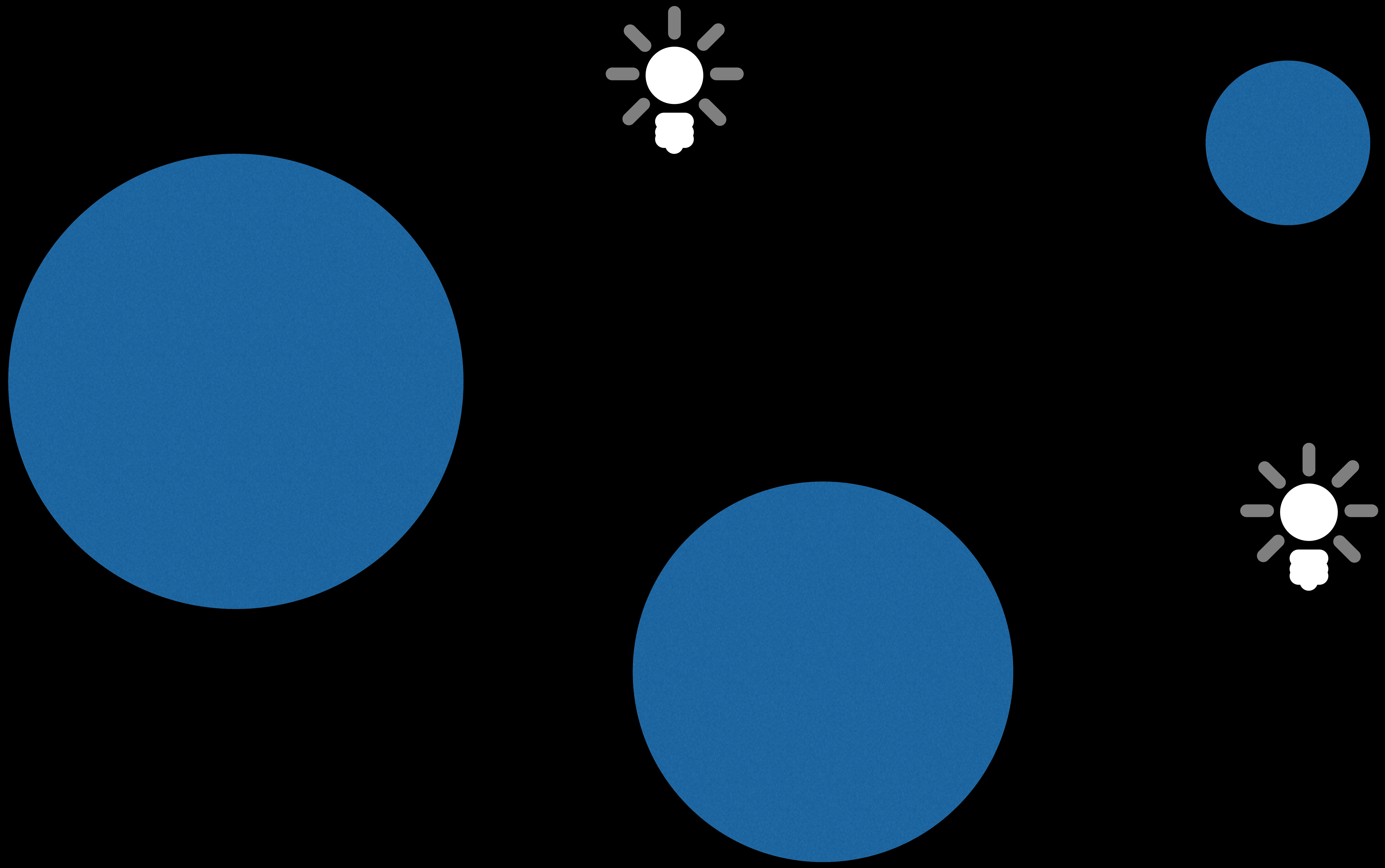
Part 2

Lighting using shaders.

2D Lighting


4
4
4
$0

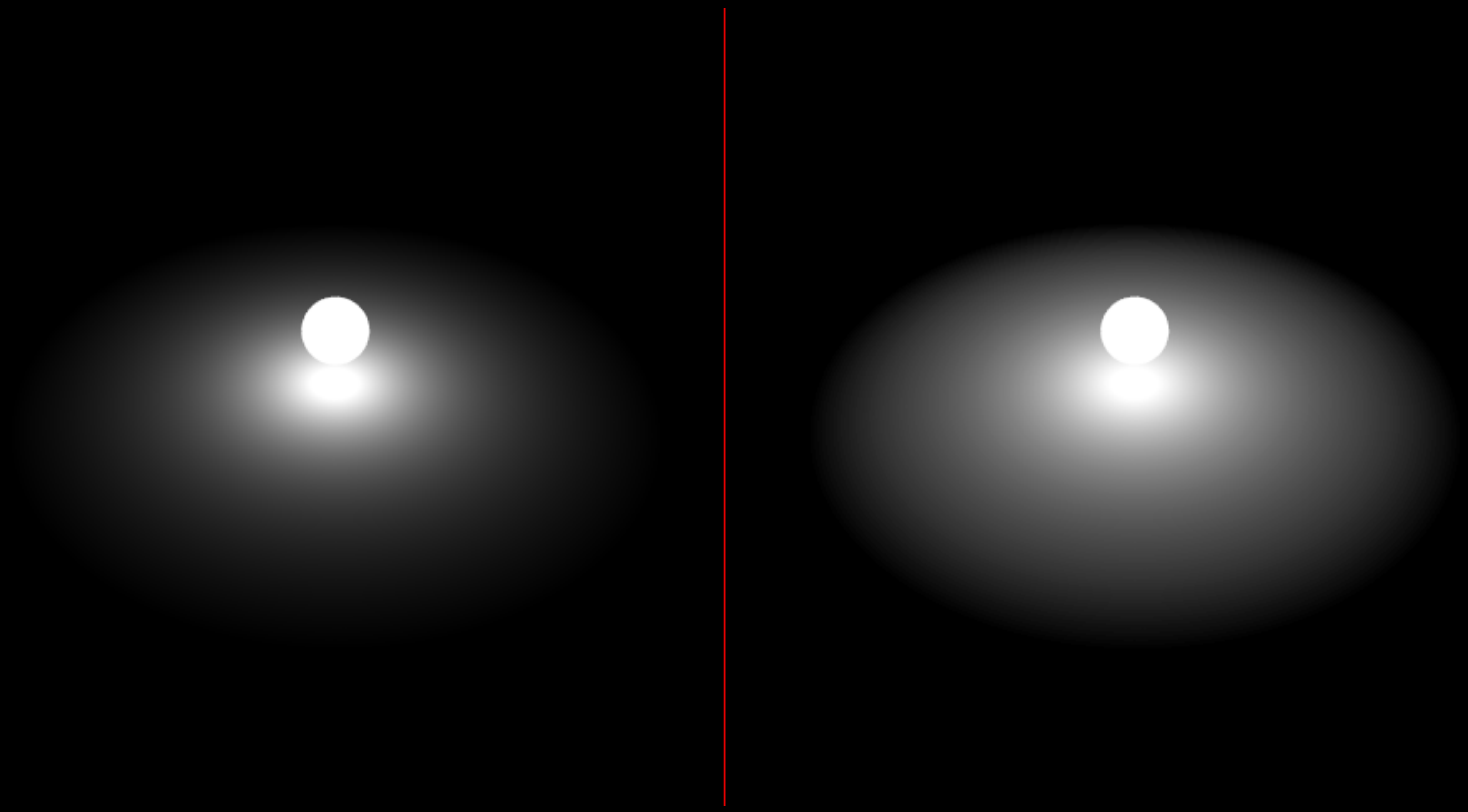

Deadly Red Phaseblade
Life: 156/260
339
145
109



For each pixel, check its **distance to each light** and **increase its brightness** based on that light’s **attenuation**.

Light attenuation

Defines the decrease in brightness based on distance from the light.

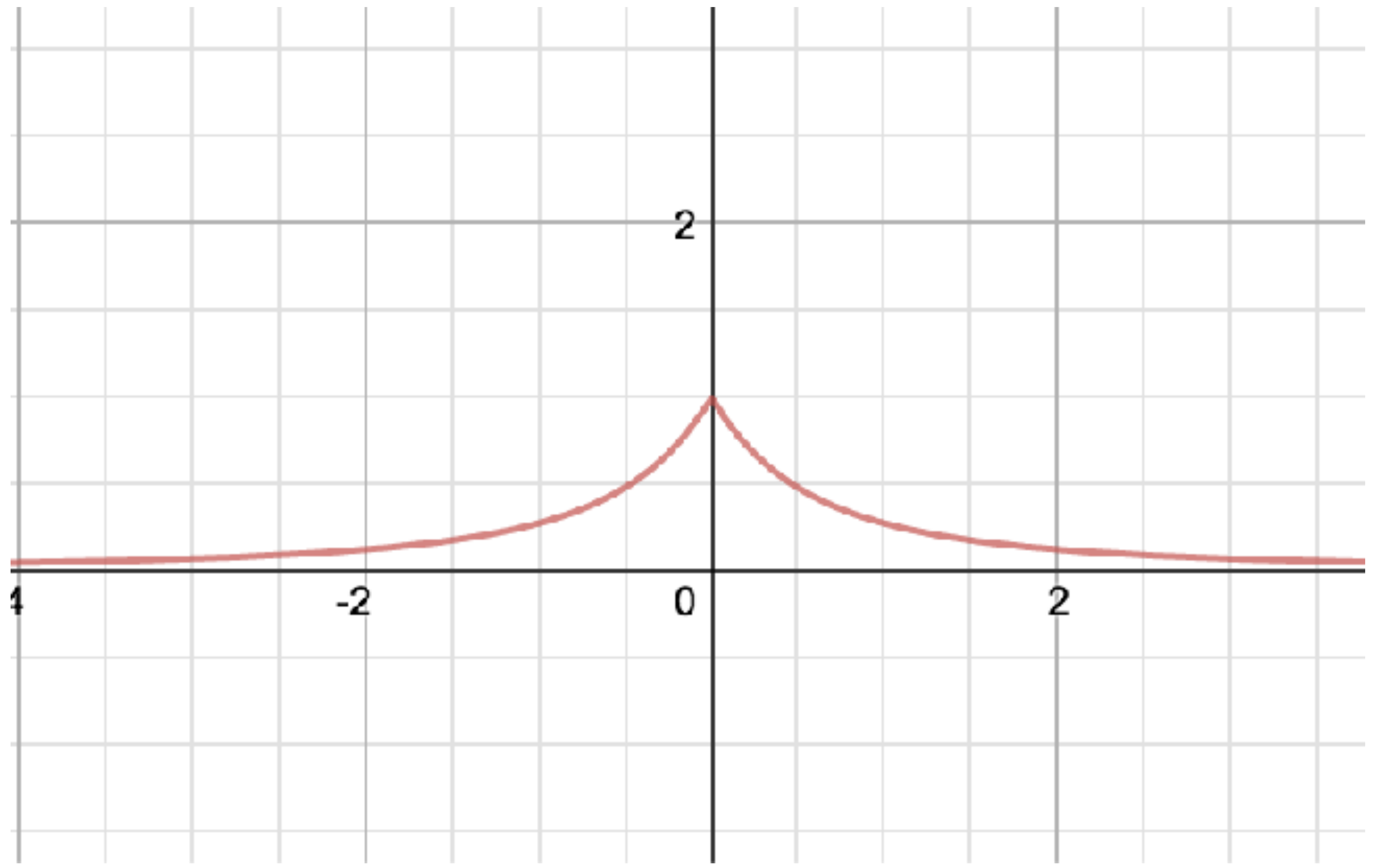


Light attenuation

Basic attenuation function.

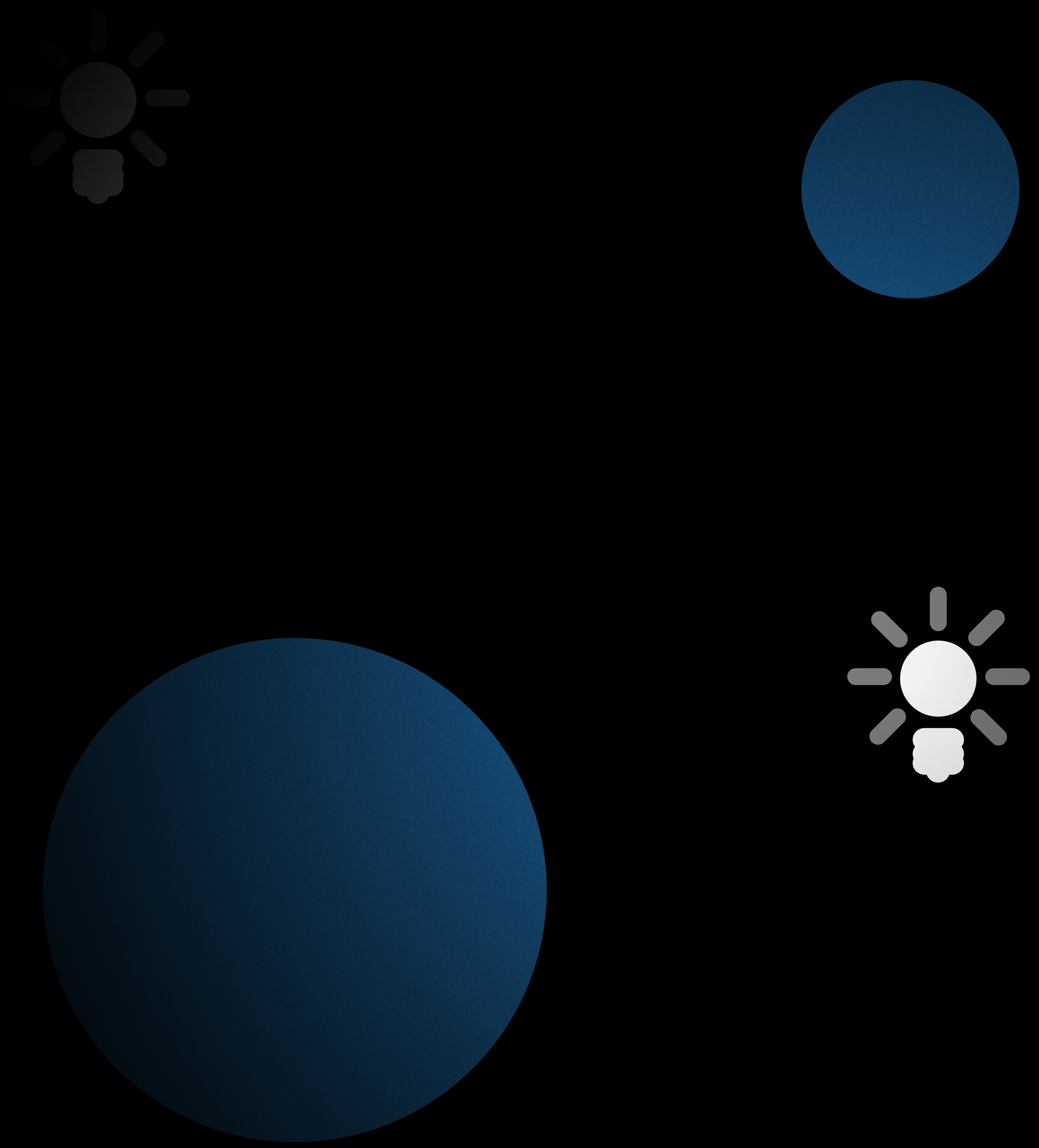
$$\frac{1}{1+a|x|+b|x|^2}$$

```
1.0 / (1.0 + a*distance + b*distance*distance)
```



See how a and b values affect the attenuation graph:

https://www.desmos.com/calculator/nmnaud1hrw



Writing a 2D lighting GLSL shader.

Single light example

Vertex shader

```
attribute vec4 position;
attribute vec2 texCoord;

uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;

varying vec2 texCoordVar;
varying vec2 varPosition;

void main()
{
    vec4 p = modelMatrix  * position;
    texCoordVar = texCoord;
    varPosition = vec2(p.x, p.y);
    gl_Position = projectionMatrix * viewMatrix * p;
}
```

Fragment shader

```
uniform sampler2D diffuse;

uniform vec2 lightPosition;

varying vec2 texCoordVar;
varying vec2 varPosition;

float attenuate(float dist, float a, float b) {
    return 1.0 / (1.0 + a*dist + b*dist*dist);
}

void main()
{
    float brightness = attenuate(distance(lightPosition, varPosition), 4.0, 0.0);
    vec4 textureColor = texture2D(diffuse, texCoordVar);
    gl_FragColor = textureColor * brightness;
    gl_FragColor.a = textureColor.a;
}
```

Multiple light example

Fragment shader

```
uniform sampler2D diffuse;

uniform vec2 lightPositions[6];

varying vec2 texCoordVar;
varying vec2 varPosition;

float attenuate(float dist, float a, float b) {
    return 1.0 / (1.0 + a*dist + b*dist*dist);
}

void main()
{
    float brightness = 0.0;

    for(int i=0; i < 6; i++) {
        brightness += attenuate(distance(lightPositions[i], varPosition), 5.0, 8.0);
    }

    vec4 textureColor = texture2D(diffuse, texCoordVar);
    gl_FragColor = textureColor * brightness;
    gl_FragColor.a = textureColor.a;
}
```

Passing arrays to GLSL

Some GLSL **array types** and their corresponding C++ uniform **binding functions**.

```
float - glUniform1fv(location, count,array_pointer);
vec2 - glUniform2fv(location, count, array_pointer);
vec3 - glUniform3fv(location, count, array_pointer);
vec4 - glUniform4fv(location, count, array_pointer);
```

The count needs to match the array size in GLSL.

Pass in all light positions as a vec2 array.

```
GLint lightPositionsUniform = glGetUniformLocation(program.programID, "lightPositions");

// -----

GLfloat lightPositions[6 * 2];

for(int i=0; i < 6; i++) {
    lightPositions[i*2] = lights[i].x;
    lightPositions[(i*2)+1] = lights[i].y;
}

glUniform2fv(lightPositionsUniform, 6, lightPositions);
```

Color lighting

Same as before, but we add a **vec3** array for **light colors.**

Fragment shader

```
uniform sampler2D diffuse;

uniform vec2 lightPositions[6];
uniform vec3 lightColors[6];

varying vec2 texCoordVar;
varying vec2 varPosition;

float attenuate(float dist, float a, float b) {
    return 1.0 / (1.0 + a*dist + b*dist*dist);
}

void main()
{
    vec3 brightness = vec3(0.0, 0.0, 0.0);

    for(int i=0; i < 6; i++) {
        brightness += attenuate(distance(lightPositions[i], varPosition), 5.0, 8.0) * lightColors[i];
    }

    vec4 textureColor = texture2D(diffuse, texCoordVar);

    gl_FragColor.xyz = textureColor.xyz * brightness;
    gl_FragColor.a = textureColor.a;
}
```

Pass in light positions as a **vec2 array** and light colors as **vec3 array**.

```
GLfloat lightPositions[6 * 2];
for(int i=0; i < 6; i++) {
    lightPositions[i*2] = lights[i].x;
    lightPositions[(i*2)+1] = lights[i].y;
}
glUniform2fv(lightPositionsUniform, 6, lightPositions);


GLfloat lightColors[6 * 3];
for(int i=0; i < 6; i++) {
    lightColors[i*3] = lights[i].r;
    lightColors[(i*3)+1] = lights[i].g;
    lightColors[(i*3)+2] = lights[i].b;
}
glUniform3fv(lightColorsUniform, 6, lightColors);
```

3D Lighting

kapreeza
intimates
DYSFORIA
WAXING
Noodle Boy
BULLFROG PLAZA


ENCODER
Hack Blocked
2
TRANSMITTER
Location Secure
3
OBJECTIVES
Eliminate all EXALT forces.
Protect the capture area.
Block EXALT's hack attempts by occupying the capture zone.
The covert operative can disrupt EXALT comm arrays.
ZISIWE KUUMBA
1
2
3
4
X


RIGA
НАЧ.СТА


НАЧ.СТА

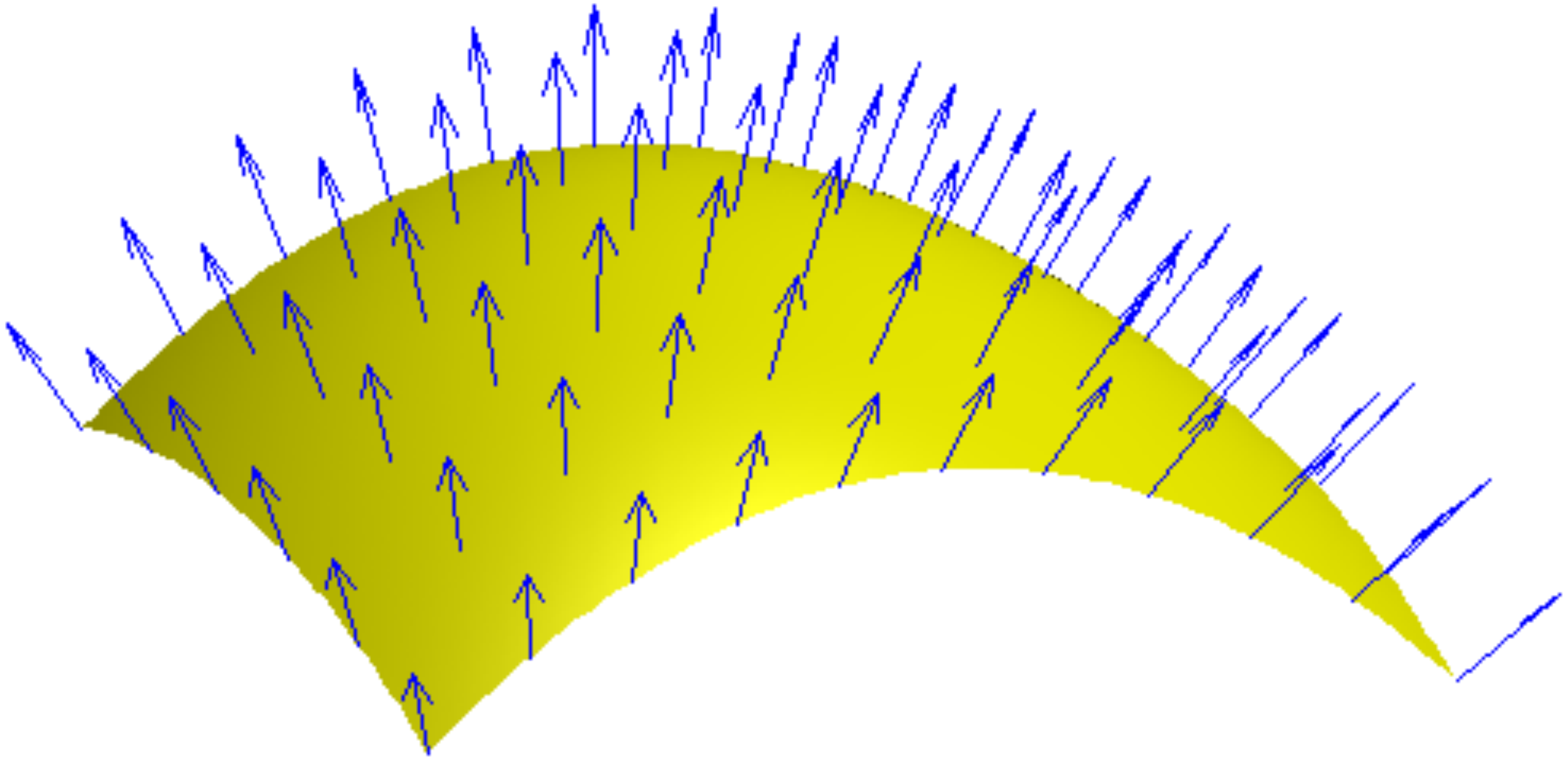
Moving into 3D

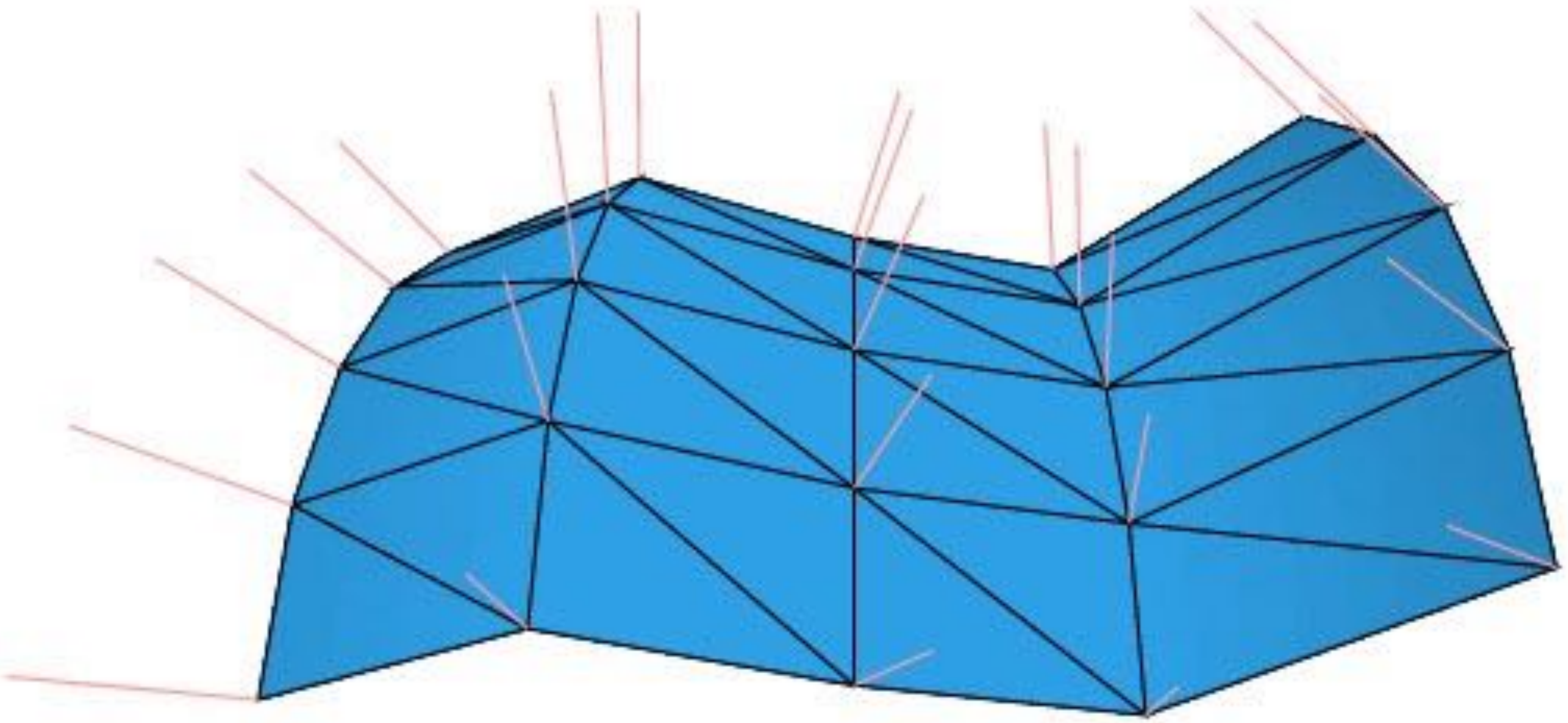
Pass in light positions as a **vec3 array** and light colors as **vec3 array**.

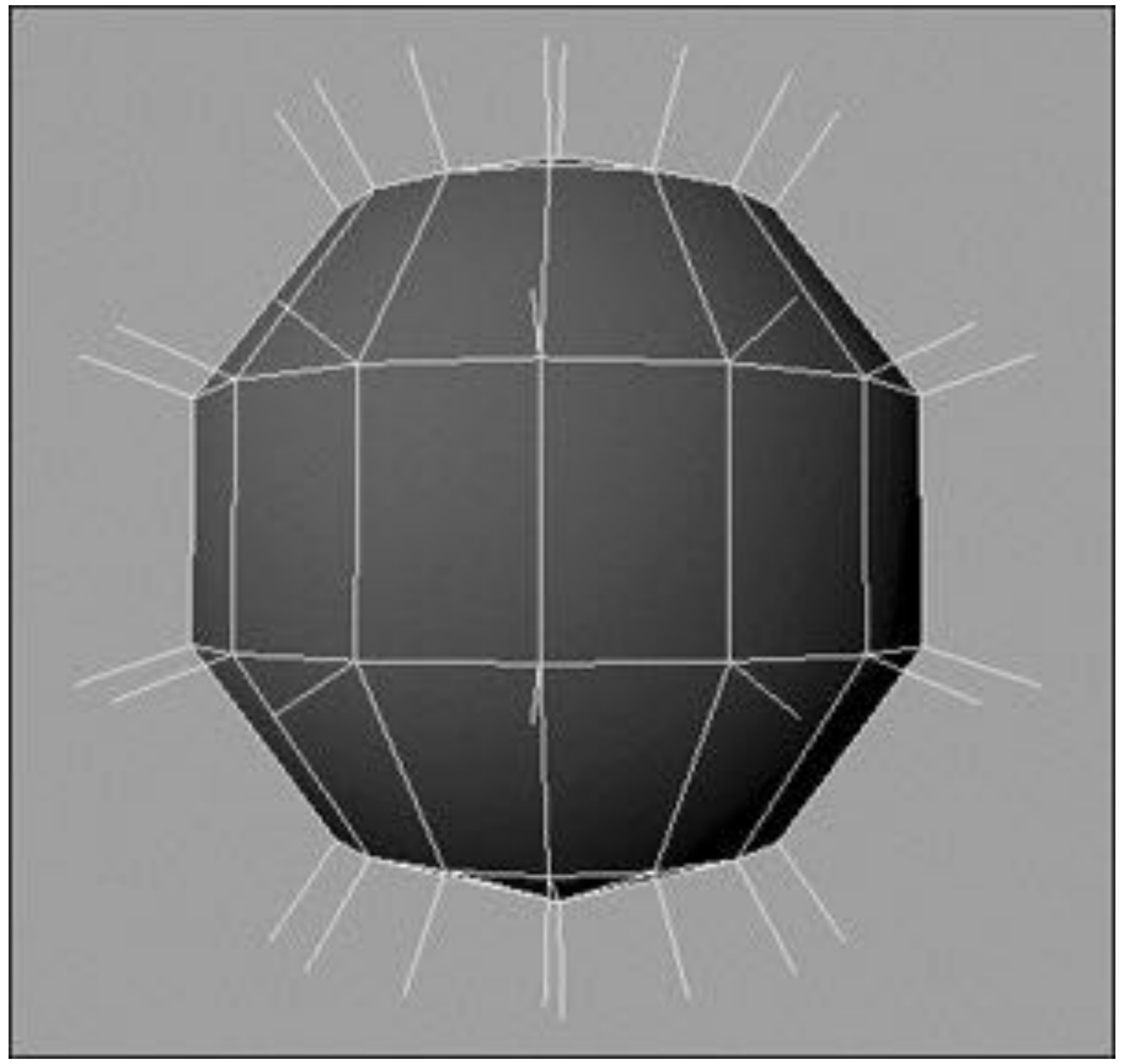
```
GLfloat lightPositions[6 * 3];
for(int i=0; i < 6; i++) {
    lightPositions[i*3] = lights[i].x;
    lightPositions[(i*3)+1] = lights[i].y;
    lightPositions[(i*3)+2] = lights[i].z;
}

glUniform3fv(lightPositionsUniform, 6, lightPositions);


GLfloat lightColors[6 * 3];
for(int i=0; i < 6; i++) {
    lightColors[i*3] = lights[i].r;
    lightColors[(i*3)+1] = lights[i].g;
    lightColors[(i*3)+2] = lights[i].b;
}

glUniform3fv(lightColorsUniform, 6, lightColors);
```

Surface **normals**.

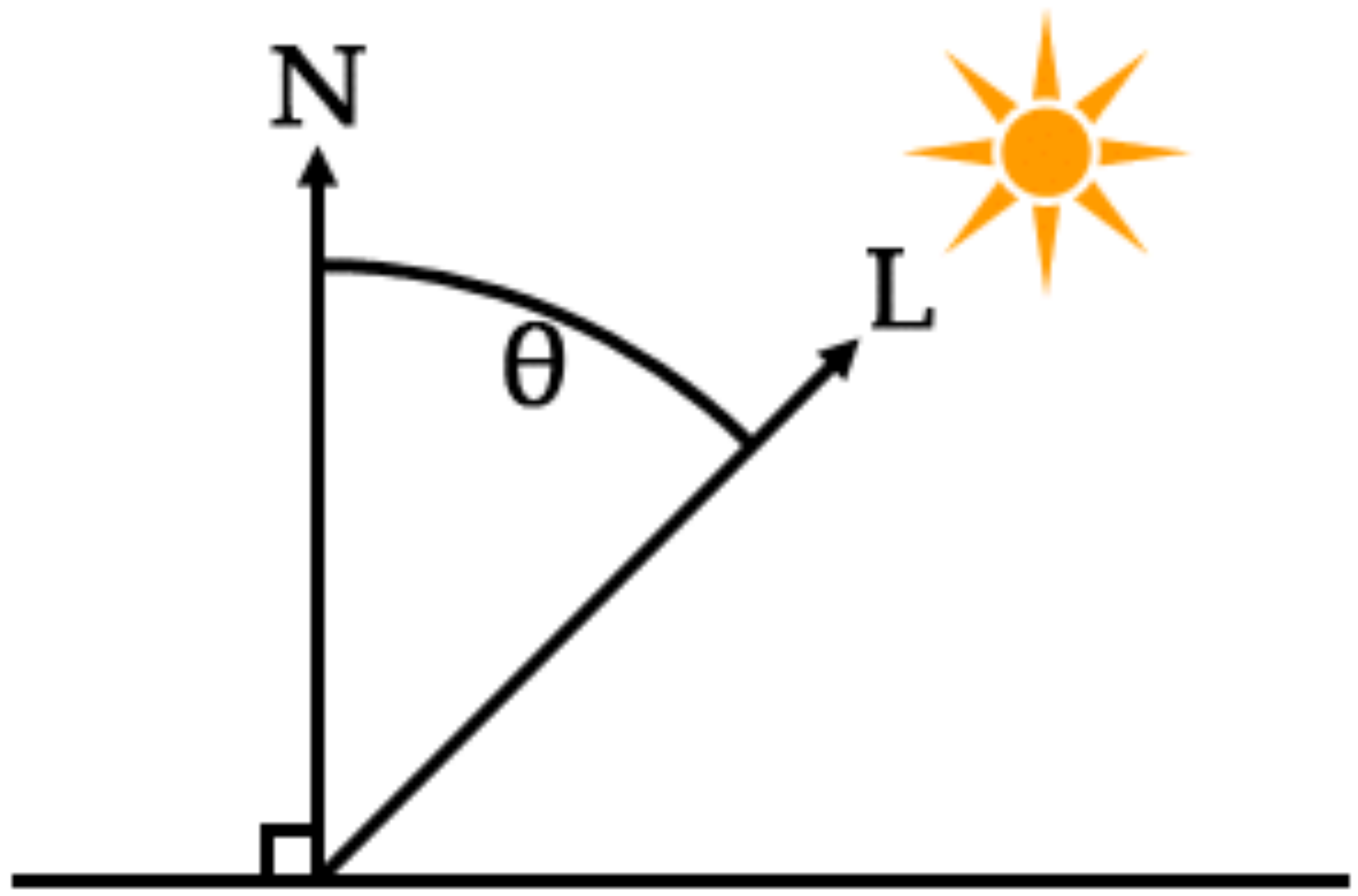
Same exact method as 2D lighting, except light positions are vec3 and we need to take the surface orientation into account.







Angle between **surface normal** and **light direction** defines **how much the light** affects that point on the surface.



Vertex shader

```
attribute vec4 position;
attribute vec3 normal;
attribute vec2 texCoord;

uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;

varying vec2 texCoordVar;
varying vec3 varPosition;
varying vec3 varNormal;

mat3 mat3_emu(mat4 m4) {
    return mat3(
                m4[0][0], m4[0][1], m4[0][2],
                m4[1][0], m4[1][1], m4[1][2],
                m4[2][0], m4[2][1], m4[2][2]);
}

void main()
{
    vec4 p = modelMatrix  * position;
    texCoordVar = texCoord;

    mat3 rotN = mat3_emu(modelMatrix);
    varNormal = normalize(rotN * normal);

    varPosition = vec3(p.x, p.y, p.z);
     gl_Position = projectionMatrix * viewMatrix * p;
}
```

Fragment shader

```
uniform sampler2D diffuse;

uniform vec3 lightPositions[6];
uniform vec3 lightColors[6];

varying vec2 texCoordVar;
varying vec3 varPosition;
varying vec3 varNormal;

float attenuate(float dist, float a, float b) {
    return 1.0 / (1.0 + a*dist + b*dist*dist);
}

void main()
{
    vec3 brightness = vec3(0.0, 0.0, 0.0);

    for(int i=0; i < 6; i++) {

        vec3 lightDir = normalize(lightPositions[i]-varPosition);
        float nDotL = dot(varNormal, lightDir);

        brightness += attenuate(distance(lightPositions[i], varPosition) / 4.0, 3.0, 5.0) * lightColors[i] * max(0.0, nDotL);

    }

    vec4 textureColor = texture2D(diffuse, texCoordVar);
    if(textureColor.a == 0.0) {
        discard;
    }

    gl_FragColor.xyz = textureColor.xyz * brightness;
    gl_FragColor.a = textureColor.a;

}
```

Setting normal attributes.

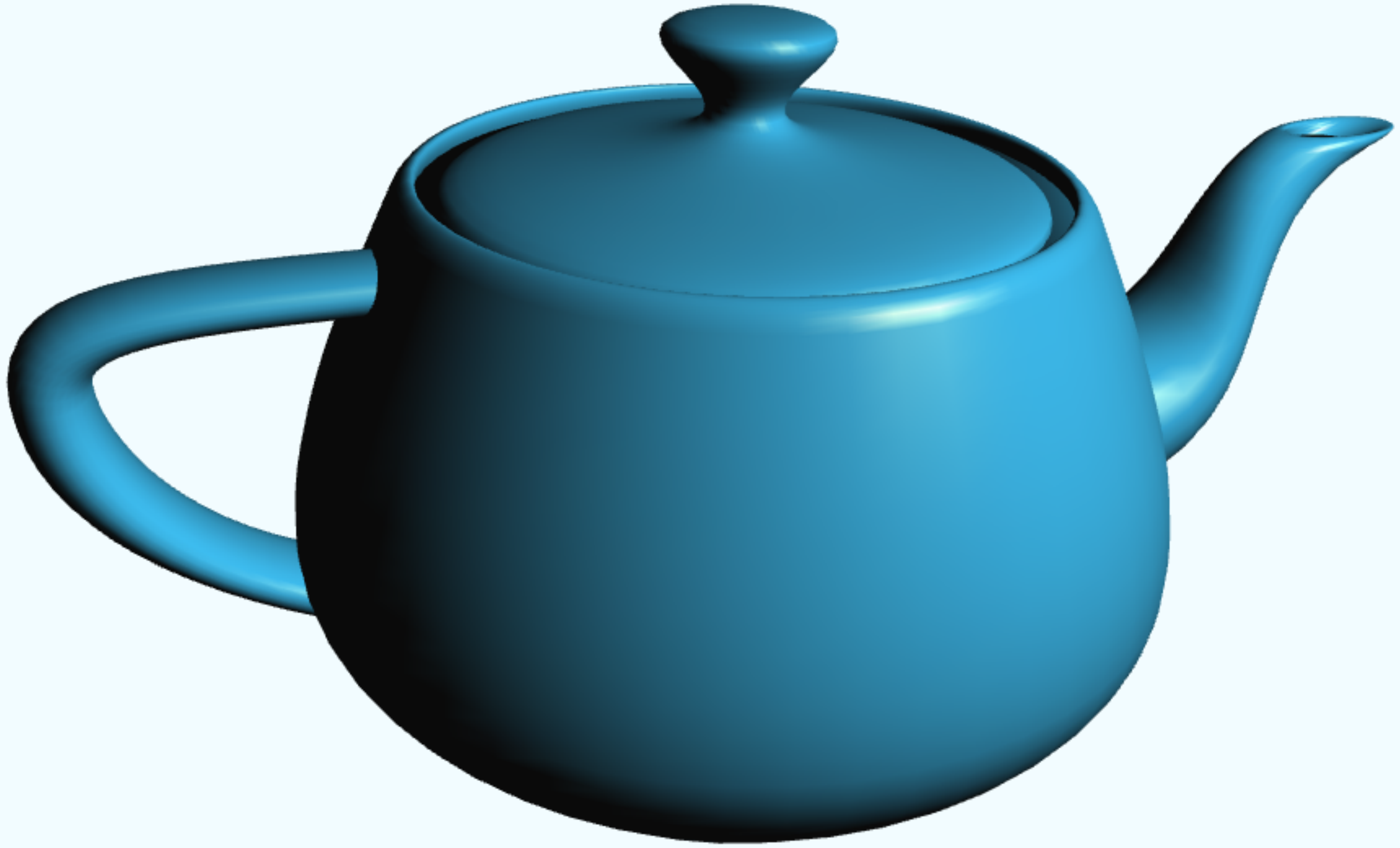
Get the normal attribute location.

```
GLuint normalAttribute = glGetAttribLocation(exampleProgram, "normal");
```

Pass an array of normals to the shader (one for each vertex).

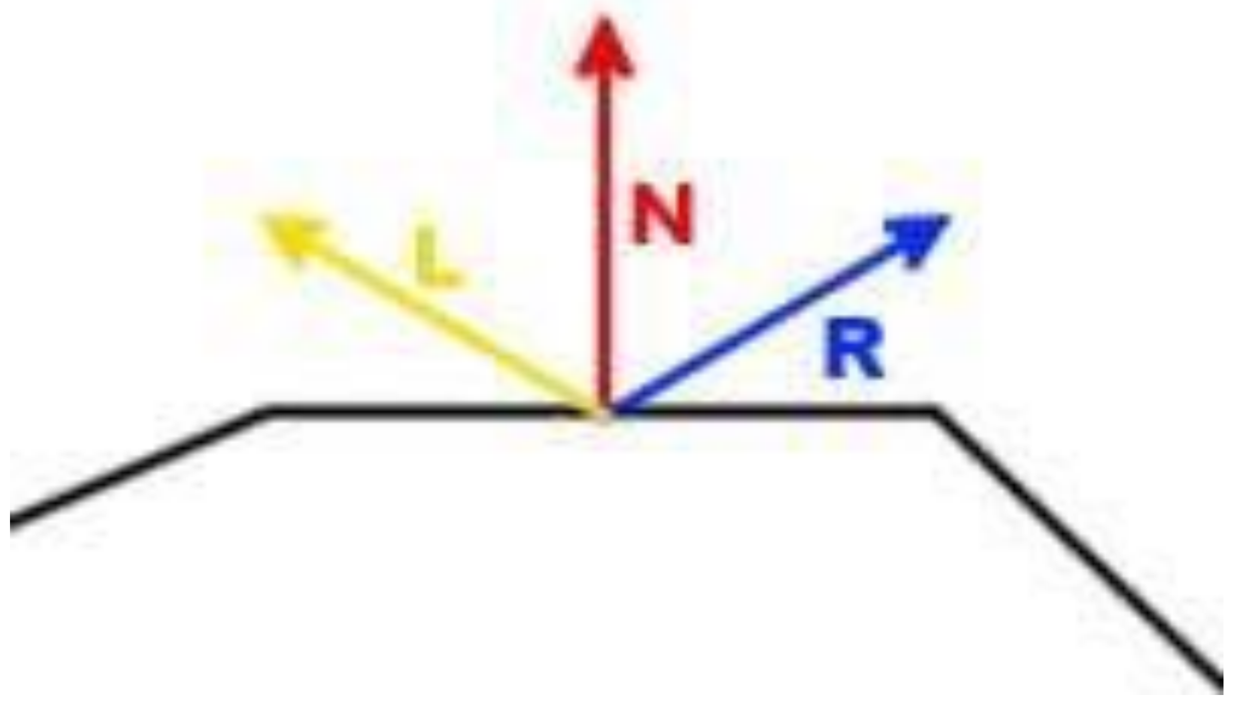
```
glVertexAttribPointer(normalAttribute, 3, GL_FLOAT, false, 0, normals.data());
glEnableVertexAttribArray(normalAttribute);
```

Specular highlights

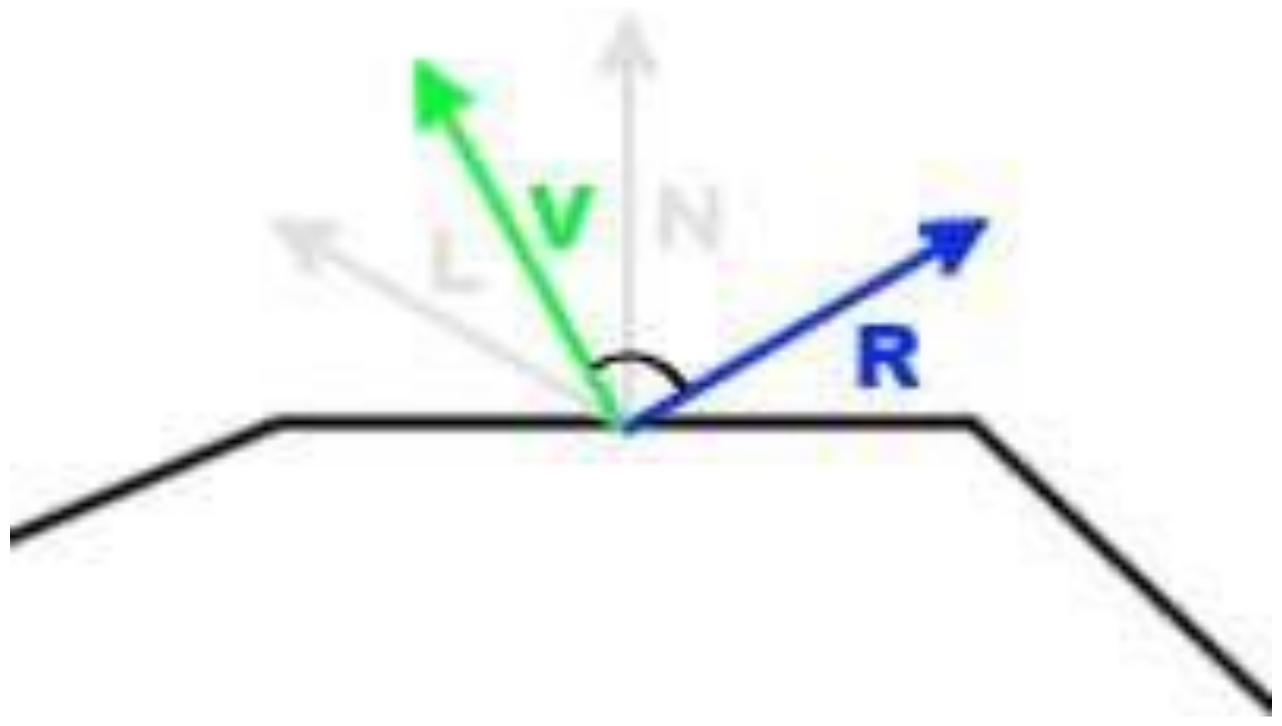


Phong shading

Calculate the **reflection vector** of the **light direction**.



Compare it with the **view direction**.



The closer the reflected light vector is to the view direction, the more brightly it will be lit.

```
uniform sampler2D diffuse;

uniform vec3 lightPositions[6];
uniform vec3 lightColors[6];

varying vec2 texCoordVar;
varying vec3 varPosition;
varying vec3 varNormal;

uniform vec3 eyePosition;

float attenuate(float dist, float a, float b) {
    return 1.0 / (1.0 + a*dist + b*dist*dist);
}

void main()
{
    vec3 brightness = vec3(0.0, 0.0, 0.0);
    vec3 specular = vec3(0.0, 0.0, 0.0);

    for(int i=0; i < 6; i++) {

        vec3 lightDir = normalize(lightPositions[i]-varPosition);
        float nDotL = dot(varNormal, lightDir);
        vec3 lightReflection = -reflect(lightDir, varNormal);
         vec3 eyeDir = normalize(varPosition-eyePosition);

        float attenuation = attenuate(distance(lightPositions[i], varPosition) / 4.0, 1.0, 3.0);

        brightness += attenuation * lightColors[i] * max(0.0, nDotL);
        specular += attenuation * lightColors[i] * pow(max(0.0,dot(lightReflection, eyeDir)), 30.0);
    }

    vec4 textureColor = texture2D(diffuse, texCoordVar);
    if(textureColor.a == 0.0) {
        discard;
    }

    gl_FragColor.xyz = textureColor.xyz * brightness + specular;
    gl_FragColor.a = textureColor.a;

}
```