Shaders

Part 2

Lighting using shaders.

2D Lighting

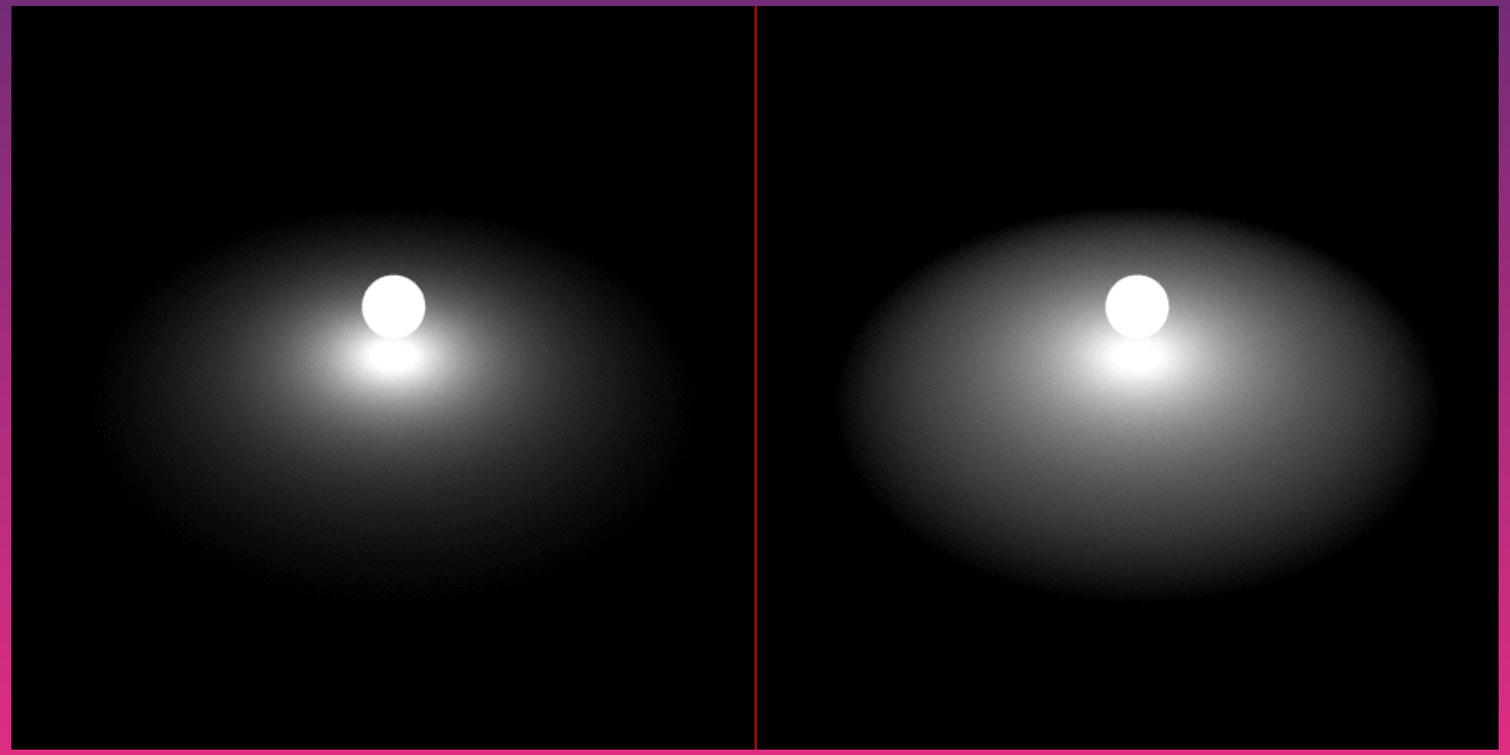

4
4
4
$0


$2854514
x9
x3
40


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.

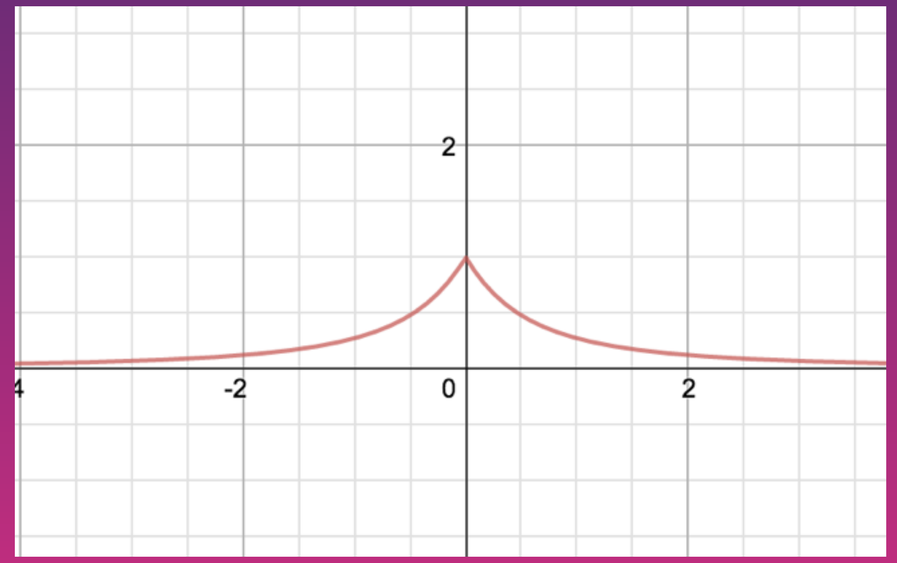


Light attenuation

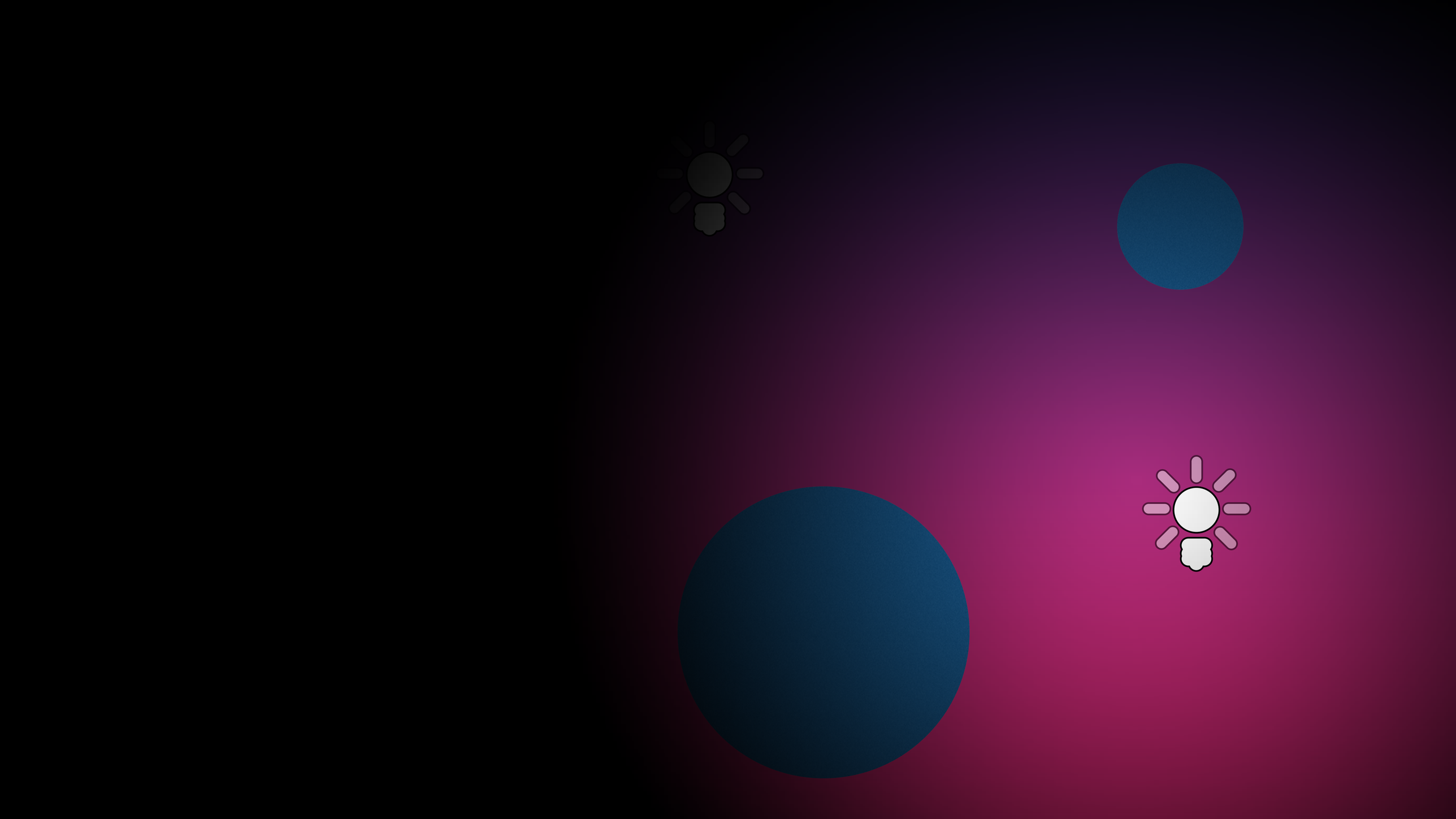
Basic attenuation function.

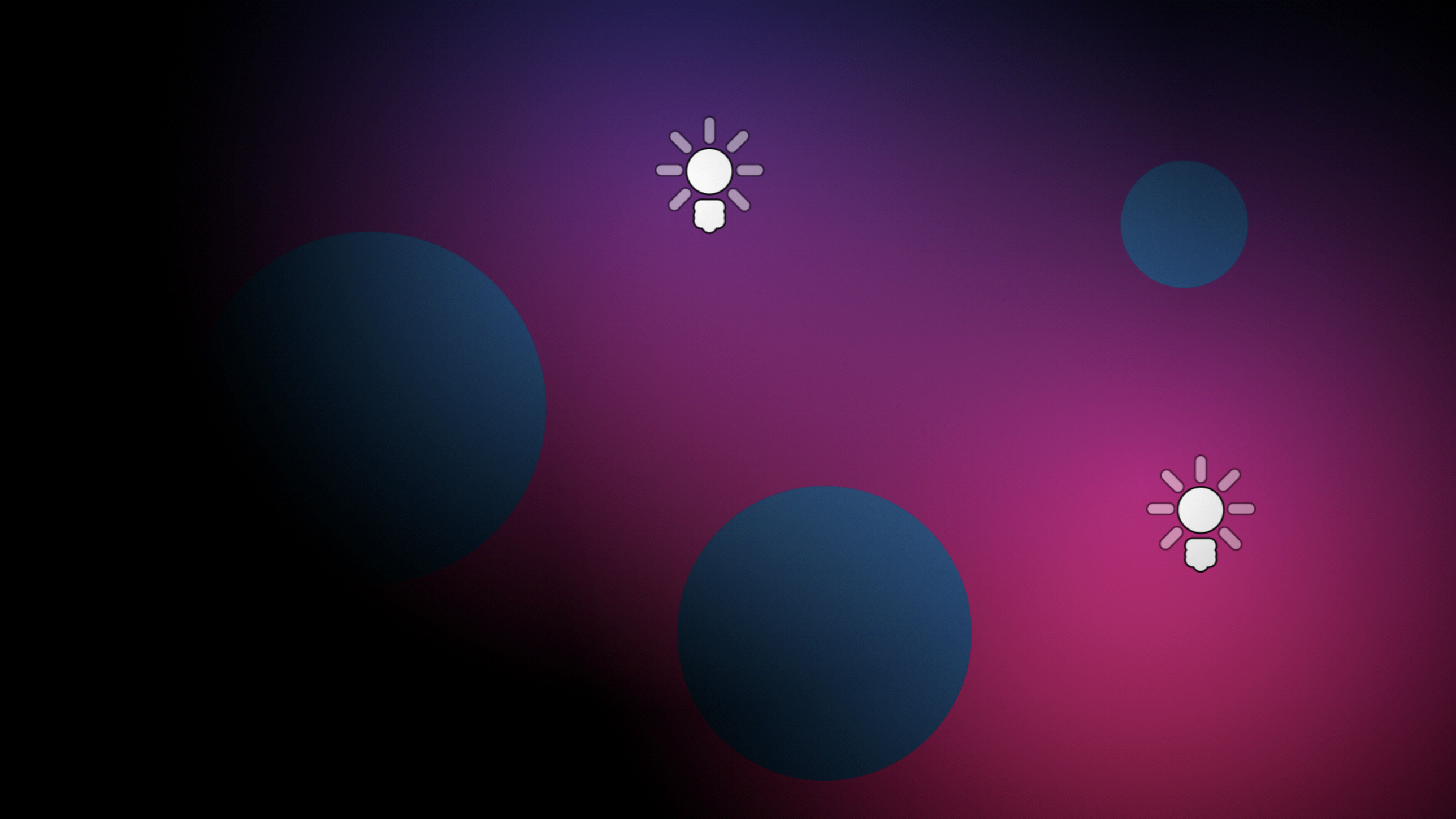
$$\frac{1}{1+a|x|+b|x|^2}$$

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



See how a and b values affect the attenuation graph:
https://www.desmos.com/calculator/nmnaud1hrw





Writing a 2D lighting GLSL shader.

Vertex shader

```
attribute vec4 position;
attribute vec2 texCoord;
uniform mat4 modelView;
uniform mat4 projection;
varying vec2 texCoordVar;
varying vec2 varPosition;

void main()
{
    vec4 p = modelView * position;
    gl_Position = projection * p;
    texCoordVar = texCoord;
    varPosition = p.xy;
}
```

Fragment shader

```
uniform sampler2D texture;
uniform vec2 lightPositions[16];
varying vec2 texCoordVar;
varying vec2 varPosition;

float attenuate(vec2 pixelPosition, vec2 lightPosition) {
        float dist = distance(lightPosition,pixelPosition);
        return 1.0 / (1.0 + 0.0*dist + 10.0*dist*dist);
}

void main()
{
    float brightness = 0.0;
    for(int i=0; i < 16; i++) {
        brightness += attenuate(varPosition, lightPositions[i]);
    }
    gl_FragColor = texture2D( texture, texCoordVar) * brightness;
    gl_FragColor.a = texture2D( texture, texCoordVar).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.

```
GLfloat lightPositions[16 * 2];

for(int i=0; i < 16; i++) {
    lightPositions[i*2] = lights[i].x + cameraOffsetX;
    lightPositions[(i*2)+1] = lights[i].y + cameraOffsetY;
}

glUniform2fv(lightPositionUniform, 16, lightPositions);
```

Since in the shader code, our vertex positions are multiplied by the modelview matrix, the light positions must take the view offset (or the view matrix)
into account.

Colored lighting

Same as before, but we add a vec3 array
for light colors and make our brightness a vec3 color.

Fragment shader

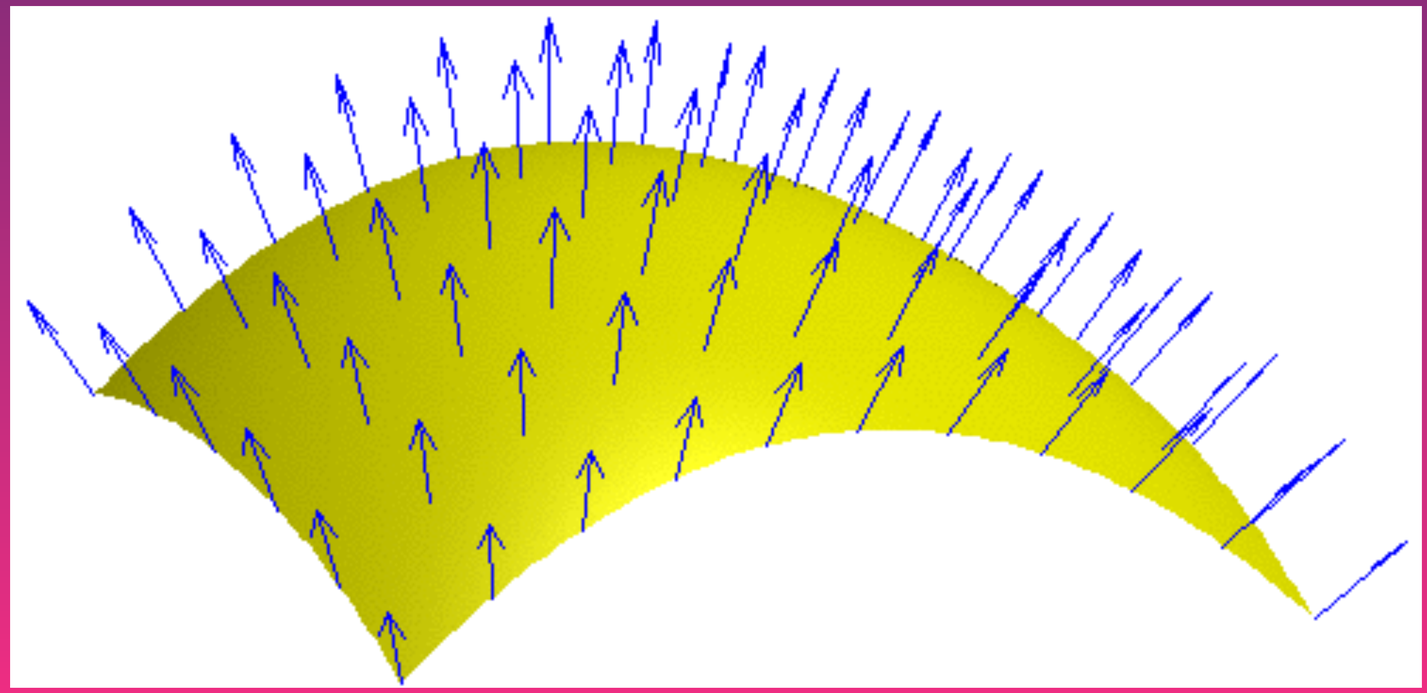
```
uniform sampler2D texture;
uniform vec2 lightPositions[16];
uniform vec3 lightColors[16];
varying vec2 texCoordVar;
varying vec2 varPosition;

float attenuate(vec2 pixelPosition, vec2 lightPosition) {
    float dist = distance(lightPosition,pixelPosition);
    return 1.0 / (1.0 + 0.0*dist + 5.0*dist*dist);
}

void main()
{
    vec3 brightness = vec3(0.0);
    for(int i=0; i < 16; i++) {
        brightness += lightColors[i] * attenuate(varPosition, lightPositions[i]);
    }
    gl_FragColor.xyz = texture2D( texture, texCoordVar).xyz * brightness;
    gl_FragColor.a = texture2D( texture, texCoordVar).a;
}
```

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

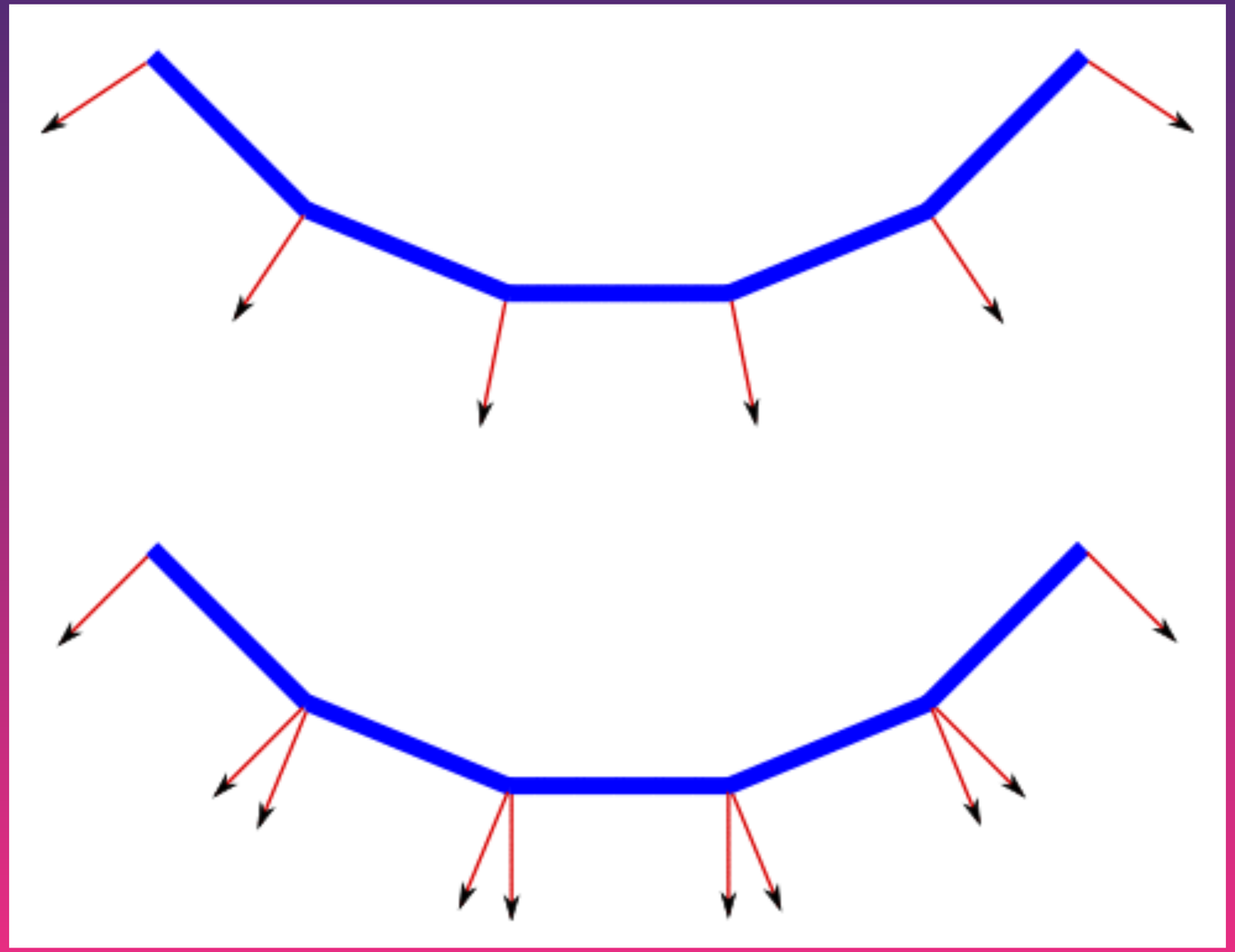
```
GLfloat lightPositions[16 * 2];

for(int i=0; i < 16; i++) {
    lightPositions[i*2] = lights[i].x + cameraOffsetX;
    lightPositions[(i*2)+1] = lights[i].y + cameraOffsetY;
}

glUniform2fv(lightPositionUniform, 16, lightPositions);


GLfloat lightColors[16 * 3];
for(int i=0; i < 16; i++) {
    lightColors[i*3] = lights[i].color_r;
    lightColors[(i*3)+1] = lights[i].color_g;
    lightColors[(i*3)+2] = lights[i].color_b;
}
glUniform3fv(lightColorUniform, 16, lightColors);
```

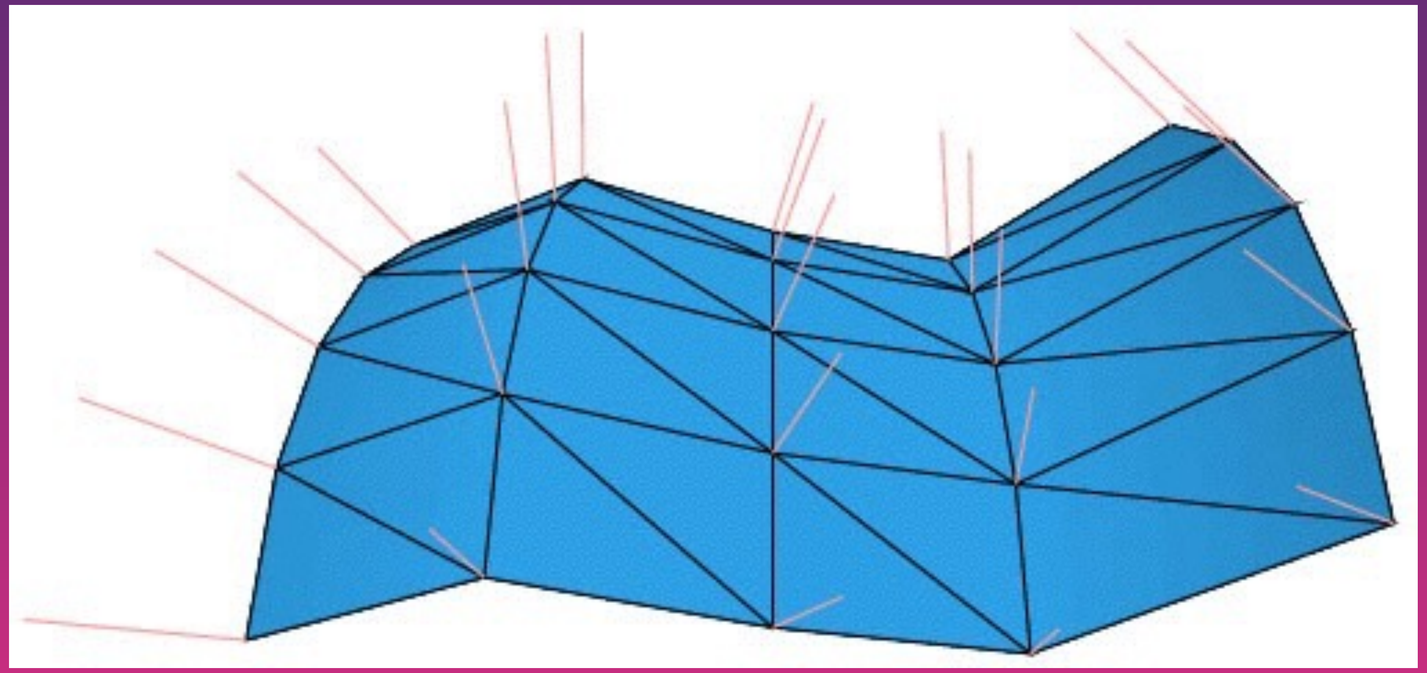
3D Lighting

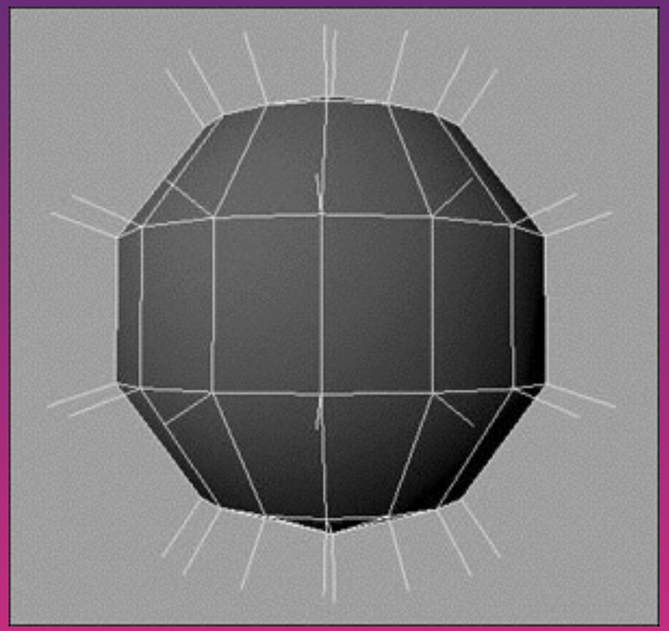
Same exact method as 2D lighting, except light positions are vec3 and we need to take the surface orientation into account.



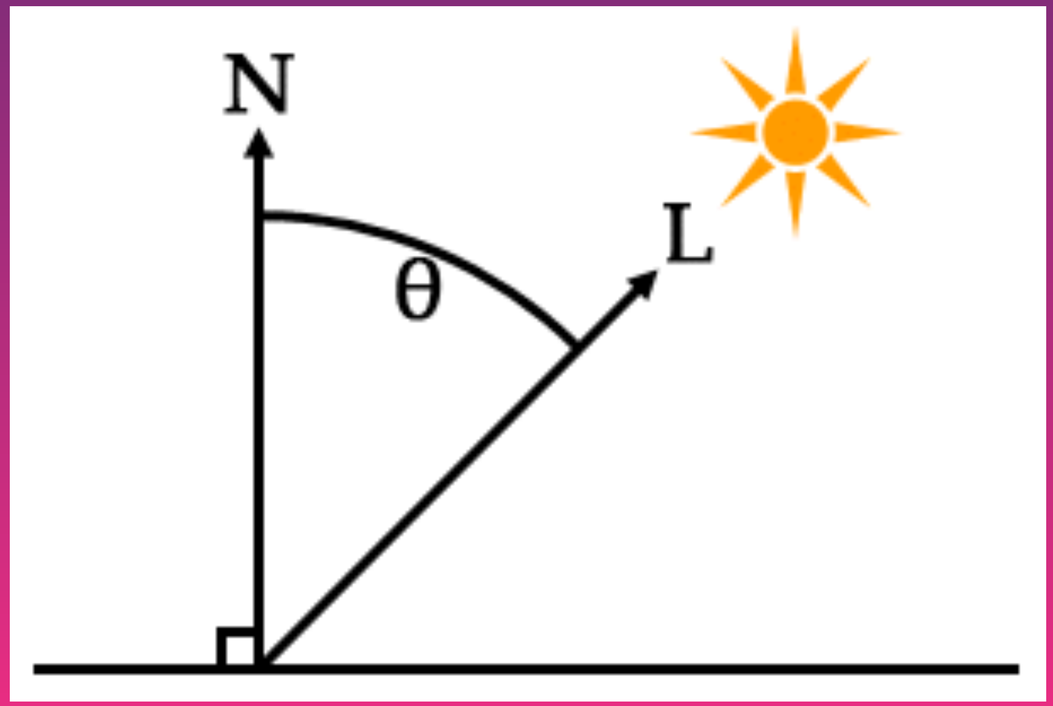
Surface normals.







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



Vertex shader

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

uniform mat4 modelView;
uniform mat4 projection;
uniform mat4 normalMatrix;

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

void main()
{
    vec4 p = modelView * position;
    gl_Position = projection * p;
    texCoordVar = texCoord;
    varPosition = p.xyz;
    varNormal = (normalMatrix * normal).xyz;
}
```

The normal matrix is the transpose of the inverse of the modelview matrix.

Fragment shader

```
uniform sampler2D texture;
uniform vec3 lightPositions[4];
uniform vec3 lightColors[4];
varying vec2 texCoordVar;
varying vec3 varPosition;
varying vec3 varNormal;

float attenuate(vec3 pixelPosition, vec3 lightPosition) {
    float dist = distance(lightPosition,pixelPosition);
    return 1.0 / (1.0 + 0.0*dist + 0.5*dist*dist);
}
void main()
{
    vec3 brightness = vec3(0.0);
    for(int i=0; i < 4; i++) {
        vec3 lightDir = normalize(varPosition - lightPositions[i]);
        float nDotL = dot(varNormal, lightDir);
        brightness += lightColors[i] * attenuate(varPosition, lightPositions[i]) * max(0.0, nDotL);
    }
    gl_FragColor.xyz = texture2D( texture, texCoordVar).xyz * brightness;
    gl_FragColor.a = texture2D( texture, texCoordVar).a;
}
```

Setting normal attributes.

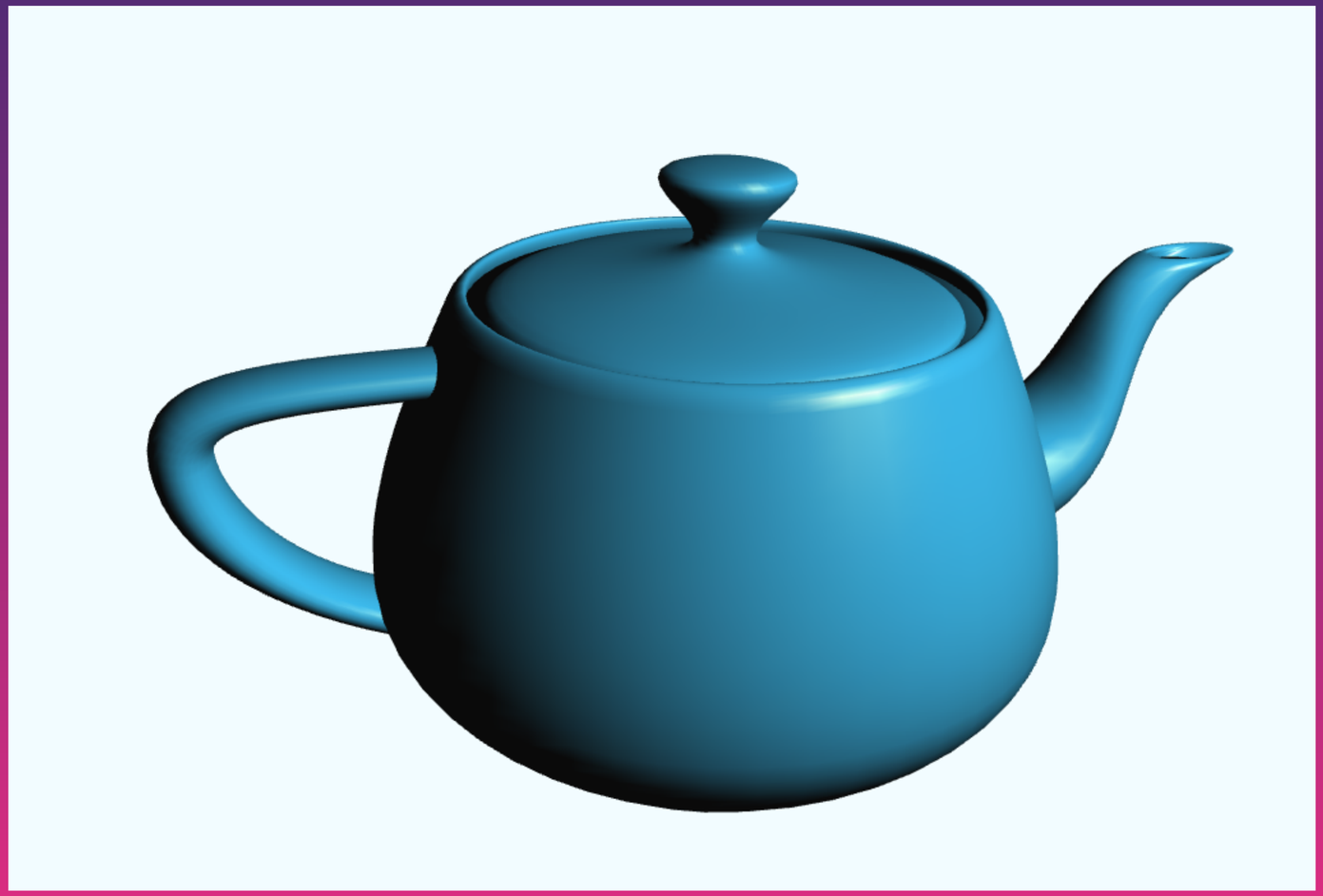
Get the normal attribute location.

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

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

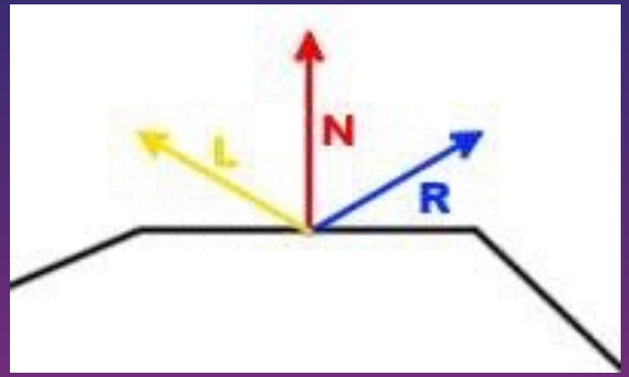
```
glVertexAttribPointer(normalAttribute, 3, GL_FLOAT, false, 0, normals.data());
glEnableVertexAttribArray(normalAttribute);
```

Specular highlights

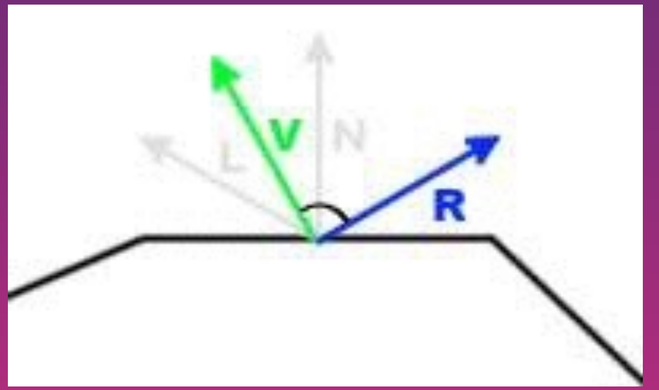


Phong shading

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 texture;
uniform vec3 lightPositions[4];
uniform vec3 lightColors[4];
varying vec2 texCoordVar;
varying vec3 varPosition;
varying vec3 varNormal;
float attenuate(vec3 pixelPosition, vec3 lightPosition) {
    float dist = distance(lightPosition,pixelPosition);
    return 1.0 / (1.0 + 0.0*dist + 0.5*dist*dist);
}
void main()
{
    vec3 brightness = vec3(0.0);
    vec3 specular = vec3(0.0);
    for(int i=0; i < 4; i++) {
        vec3 lightDir = normalize(varPosition - lightPositions[i]);
        vec3 lightReflection = -reflect(lightDir, varNormal);
        vec3 eyeDir = normalize(-varPosition);
        float nDotL = dot(varNormal, lightDir);
        float attenuation = attenuate(varPosition, lightPositions[i]);
        brightness += lightColors[i] * attenuation * max(0.0, nDotL);
        specular += attenuation * pow(max(0.0,dot(lightReflection, eyeDir)), 5.0);
    }
    gl_FragColor.xyz = (texture2D( texture, texCoordVar).xyz * brightness) + specular;
    gl_FragColor.a = texture2D( texture, texCoordVar).a;
}
```