Assignment 4

T-511-TGRA Computer Graphics

Andrea Terenziani
andrea23@ru.is

1 Lighting model

The lighting model used implements per-pixel shading, using Lambert for the diffuse component and Phong for the specular; this can also optionally be switched off to have a fully unshaded material that directly uses the diffuse color of the object.

It supports up to 4 point lights in the same scene, each with its own color. Furthermore, the intensity of each light is attenuated up to a certain maximum distance, beyond which it is set to 0; by setting this distance to 0 itself, we can turn off the attenuation entirely.

A fairly doable addition, not included in the project, would have been support for directional lights, implementable with a boolean property of each light.

2 Shader code

2.1 Vertex shader

2.1.1 Uniforms and attributes

```
// Position & Normal attribute of each vertex
attribute vec3 a_position;
attribute vec3 a_normal;

// Model, Projection & View matrix of the current camera
// Used for the projection of the vertex coordinates from its own local coordinates
↪  to projected coordinates
uniform mat4 u_model_matrix;
uniform mat4 u_view_matrix;
uniform mat4 u_projection_matrix;

// World-space position of the camera, used for the Phong specular factor
uniform vec4 u_camera_position;

// World-space positions of each light in the scene
uniform vec4 u_light_position[4];
// Actual number of lights present in the scene
uniform int u_light_count;
```

2.1.2 Varyings

```
// (For each light) s vector from the vertex to the light's position, used for
↪  Lambert and Phong
varying vec4 s[4];
// (For each light) distance from the vertex, used to compute the attenuation factor
varying float dist[4];
// difference between vertex and camera position (v vector)
varying vec4 v;
// normal direction at the current vertex
varying vec4 norm;
```

2.1.3 `main` function

```
void main(void)
{
        // Conversion to homogenous coordinates 4th component of the normal set to 0
        ↪  to keep it unaffected by translation and perspective
        vec4 position = vec4(a_position, 1.0);
        vec4 normal = vec4(a_normal, 0.0);

        // The vertex's position and normal are converted from local to global
        ↪  coordinates using the view matrix
        position = u_model_matrix * position;
        normal = u_model_matrix * normal;

        // Computing light-specific properties (s-vector and distance) c is the
        ↪  minimum between actual and max light count, so that we don't try to
        ↪  compute properties of lights that don't exist
        int c = (u_light_count < 4) ? u_light_count : 4;
        for (int i = 0; i < c; i++) {
                s[i] = u_light_position[i] - position;
                dist[i] = distance(u_light_position[i], position);
        }

        // Computing the v-vector for the Phong specular
        v = u_camera_position - position;
        // Normalization of normal vector (more of a safety check)
        norm = normalize(normal);

        // Transformation of the vertex position into camera space (with the view
        ↪  matrix) and then into projected space (with the projection matrix)
        position = u_projection_matrix * (u_view_matrix * position);

        gl_Position = position;
}
```

2.2 Fragment shader

2.2.1 Uniforms

```
// Diffuse & specular colors of each light
uniform vec4 u_light_diffuse[4];
uniform vec4 u_light_specular[4];
// Maximum distance of each light, beyond which its contribution is attenuated to 0.
// The name "radius" comes from the fact that each light is a point that illuminates
↪  in all directions
uniform float u_light_radius[4];

// Diffuse & specular colors of the material
uniform vec4 u_material_diffuse;
uniform vec4 u_material_specular;
```

```
// Boolean flags to control if the material receives ambient lighting
↪ (u_receive_ambient) and if it uses the full shading calculation (u_unshaded)
uniform bool u_receive_ambient;
uniform bool u_unshaded;

// Shininess factor of the material, used to control the size and sharpness of the
↪ specular highlight
uniform float u_shininess;

// Background ambient color of the scene
uniform vec4 u_ambient;

// Number of lights actually present in the scene
uniform int u_light_count;
```

2.2.2 Varyings

See section 2.1.2. The `varying` variables are repeated to properly "connect" them to the vertex shader and are here computed per-pixel by interpolating the values calculated per-vertex.

```
varying vec4 s[4];
varying float dist[4];
varying vec4 v, h;
varying vec4 norm;
```

2.2.3 Lambert-Phong calculation

```
// This function computes the shaded color for a given light, defined by the input
↪ arguments
vec4 compute_shaded_color(vec4 s_vec, float d, float radius, vec4 light_diffuse, vec4
↪ light_specular)
{
    // If the light is too far away (and its radius is more than 0 to switch on the
    ↪ attenuation) then we directly return black
    if (radius > 0. && d > radius)
        return vec4(0., 0., 0., 1.);

    // Calculate ambient component, tuned down by 0.1 and controlled by the
    ↪ u_receive_ambient flag
    vec4 ambient = (u_ambient * float(u_receive_ambient) * .1);

    // Calculate the diffuse component using Lambert's model
    float lambert = max(0.0, dot(s_vec, norm) / (length(s_vec) * length(norm)));
    vec4 diffuse = light_diffuse * u_material_diffuse * lambert;

    // Calculate the specular component using Phong's model
    vec4 h = (v + s_vec) * .5;
    float phong = max(0.0, dot(h, norm) / (length(h) * length(norm)));
    float shininess = u_shininess;
    vec4 specular = light_specular * u_material_specular * pow(phong, shininess);
```

```
    // Calculate the attenuation due to distance (where 1.0 means no attenuation and
    ↪  0.0 means that the color is fully black)
    float dist_factor = radius <= 0. ? 1. : 1. - (d / radius);

    return (ambient + diffuse + specular) * dist_factor;
}
```

2.2.4 `main` function

```
void main(void)
{
    // If the material is set as unshaded, we simply use the diffuse color as the
    ↪  final fragment color and skip all other calculations
    if (u_unshaded)
    {
        gl_FragColor = u_material_diffuse;
        return;
    }

    // Shading computation, done only for the number of lights actually present in
    ↪  the scene
    int c = (u_light_count < 4) ? u_light_count : 4;
    for (int i = 0; i < c; i++)
    {
        // The fragment color is the sum of the colors computed for each light, which
        ↪  is then normalized by the OpenGL pipeline to a vector of 0 to 1 values
        gl_FragColor += compute_shaded_color(s[i], dist[i], u_light_radius[i],
        ↪  u_light_diffuse[i], u_light_specular[i]);
    }
}
```