Project 3 - Lighting Effects and Shadows

CS 1566 — Introduction to Computer Graphics

Check the Due Date on the CourseWeb

The purpose of this project are as follows:

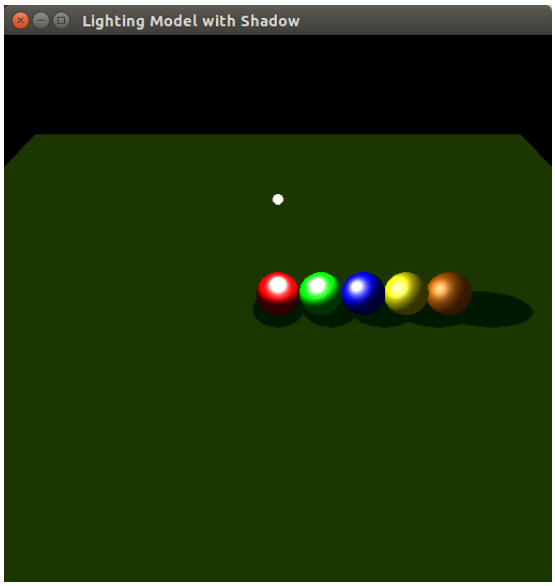
- Creating a lighting effect by applying the lighting model in the vertex shader.
- Creating fake shadow in the vertex shader

Generate a World (Object) Frame

For this project, your world will be a simple pool table with 5 billiard balls, and a light bulb with the following description:

- The pool table is a 20×20 flat surface or it can be a cube where the surface is at the plane $y = 0$ and the color of the surface is green.
- Five billiard balls should be yellow, blue, red, green, and orange balls. A ball is a sphere with radius 0.5. The location of the center of the balls are at $(0.0, 0.5, 0.0)$, $(1.0, 0.5, 0.0)$, $(2.0, 0.5, 0.0)$, $(3.0, 0.5, 0.0)$, and $(4.0, 0.5, 0.0)$. **Note** that these balls are simply sit on the pool table.
- The light bulb is simply a sphere with radius 0.3 and its center located at $(0.0, 3.0, 0.0)$.

Your work should look like the following:



Note that to create the above world, there are two methods:

1. Send a set of vertices of a cube and a set of vertices of a sphere into the graphics pipeline and use vertices of the sphere six times to create each sphere in the scene. Each time, a different transformation matrix must be sent to transform the sphere to its desired size and location.

2. Send a set of vertices of a cube and six sets of vertices of spheres where each set of vertices of a sphere has been scaled and translated into its desired size and location.

Whichever method you want to use is up to you.

Lighting Model

For this project, you will use Phong's lighting model to assign a color to each vertex based on light source location, material properties, etc. This lighting model will be done in the vertex shader. Recall that you need the light parameters for diffuse, specular, and ambient. Set them to all white in this case:

```
vec4 light_diffuse = {1.0, 1.0, 1.0, 1.0};
vec4 light_specular = {1.0, 1.0, 1.0, 1.0};
vec4 light_ambient = {0.2, 0.2, 0.2, 1.0};
```

Since we are going to have multiple materials, it is a good idea to create a structure of material as shown below:

```
typedef struct
{
    vec4 reflect_ambient;
    vec4 reflect_diffuse;
    vec4 reflect_specular;
    GLfloat shininess;
} material;
```

In doing so, you can have an array of materials that you can iterate through in your program:

```
material ball_materials[5] = {
    {{1.0, 0.0, 0.0, 1.0}, {1.0, 0.0, 0.0, 1.0}, {1.0, 1.0, 1.0, 1.0}, 10},
    {{0.0, 1.0, 0.0, 1.0}, {0.0, 1.0, 0.0, 1.0}, {1.0, 1.0, 1.0, 1.0}, 10},
    {{0.0, 0.0, 1.0, 1.0}, {0.0, 0.0, 1.0, 1.0}, {1.0, 1.0, 1.0, 1.0}, 10},
    {{1.0, 1.0, 0.0, 1.0}, {1.0, 1.0, 0.0, 1.0}, {1.0, 1.0, 1.0, 1.0}, 10},
    {{1.0, 0.5, 0.0, 1.0}, {1.0, 0.5, 0.0, 1.0}, {1.0, 1.0, 1.0, 1.0}, 10}};
```

Note that you also need materials for table, light (the 6th sphere), and shadow (if you want to). So, apply the lighting model in your vertex shader as we discussed in class.

Shadows

For this project, you are going to create a shadow of each ball using the same method discussed in the lab. However, these shadow will not be a stationary shadow. It will be changed based on the

location of the ball as well as the location of the light source. To be able to achieve this goal, we have to render shadows in the vertex shader by performing all shadow projection calculations in the vertex shader program.

Recall that the shadow projection calculations need the location of the light source and the location of the vertex that it wants to project. The location of the light source will be sent as a uniform variable as usual. To get the set of vertices to be projected, you can either reuse the original vertices of the sphere or simply send another copy of vertices of the sphere and use them.

To tell the vertex shader whether to project a set of vertices as a shadow or not, we need to send a type of flag. This can be does by passing another uniform variable of type `int` and use it to decide whether to project to vertices to the plane $y = 0$.

```
uniform int isShadow;

int main()
{
    :
    if(isShadow == 1)
    {
        color = shadow_color;
        float x = ...;
        float y = 0.001;
        float z = ...;
        gl_Position = projection * model_view * vec4(x, y, z, 1.0);
    }
    else
    {
        :
        color = ambient + attenuation * (diffuse + specular);
        gl_position = projection * model_view * vPosition;
    }
    :
}
```

Note that you have to locate the variable `isShadow` in the shader program in your `init()` function first and set it before you tell the OpenGL to draw:

```
GLuint isShadow_location;

void init()
{
    :
    isShadow_location = glGetUniformLocation(program, "isShadow");
    :
}

void display()
{
    :
```

```
    glUniform1i(isShadow_location, 0);
    :
    glDrawArrays(...)
    :
    glUniform1i(isShadow_locaiton, 1)
    :
    glDrawArrays(...)
    :
}
```

Controls

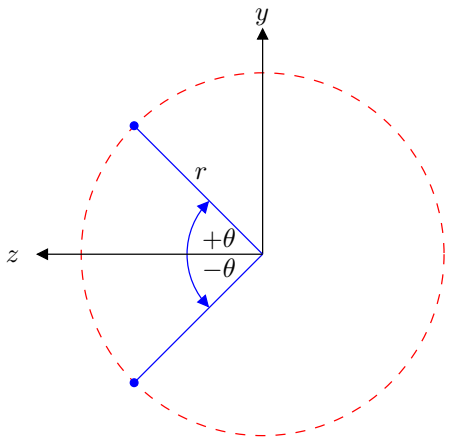
For this project, you must be able to control the location of the viewer (the **at** location of the model view matrix) and the location of the light source.

- **Light Source**: use keyboard to adjust the location of the light source based on x, y, and z axes of the **world** frame. Your program must be able to move the location of the light source to any location in the world frame. **Note** that whenever the location of the slight source is changed, shadows and lighting model must be recalculated. So, you must tell the OpenGL to draw the whole scene again based on the new light source location (see below):

- **Viewer Location**: Similarly, use keyboard to change the location of the viewer. **Note** that the location of the viewer (in the world frame) must be on a surface of a sphere of a specific radius (initially at 10.0 or something) and the viewer always look at the origin. We must be able to adjust the radius as well which will it gives you an effect of moving closer or further from the pool table. **HINT**, the surface of a sphere can be defined in parametric form $f(\theta, \phi, r)$ and

$$x = f_x(\theta, \phi, r)$$
$$y = f_y(\theta, \phi, r)$$
$$z = f_z(\theta, \phi, r)$$

For example, imagine the point $(0, 0, r)$. This point is on the surface of the sphere with radius r. We can move this point up or down by θ degree as shown below (side view):



From the above picture, use Trigonometry to find out the location of the point. Note that the point is still at $x = 0$, the values of y and z depend on the θ and r. Next is to rotate the point about y-axis for ϕ degree. The result will change the x and z values but the value of y is unchanged. Using the parametric form, you can simply use keyboard to adjust θ, ϕ, and r to change the location of the eye point. **Note** that the location of the viewer also effect the lighting model especially the specular part. Thus, you must recalculate colors of the whole scene again:

Animation

For this project, the sphere at the center will be stationary. The rest of the spheres will move around in circle at different speed. Simply use rotate about y-axis. As usual, the location of objects effect the lighting model and shadows. Calculations for all colors are needed.

Uniform Variables

For this project, you are going to have a lot of uniform variables as we discussed in class. Recall that for each uniform variable, you need a location variable of type `GLuint`. So, a part your `init()` function may look like the following:

```
:
ctm_location = glGetUniformLocation(program, "ctm");
model_view_location = glGetUniformLocation(program, "model_view");
projection_location = glGetUniformLocation(program, "projection");
isShadow_location = glGetUniformLocation(program, "isShadow");

ap_location = glGetUniformLocation(program, "AmbientProduct");
dp_location = glGetUniformLocation(program, "DiffuseProduct");
sp_location = glGetUniformLocation(program, "SpecularProduct");
light_position_location = glGetUniformLocation(program, "LightPosition");
LightPosition = light_position;
```

```
ac_location = glGetUniformLocation(program, "attenuation_constant");
al_location = glGetUniformLocation(program, "attenuation_linear");
aq_location = glGetUniformLocation(program, "attenuation_quadratic");

shininess_location = glGetUniformLocation(program, "shininess");
:
```

You also need to send a lot of data before you draw in your `display()` function as shown below:

```
:
glUniform1i(isShadow_location, 0);
glUniformMatrix4fv(ctm_location, 1, GL_FALSE, (GLfloat *) &ctm);
AmbientProduct = product(table_material.reflect_ambient, light_ambient);
DiffuseProduct = product(table_material.reflect_diffuse, light_diffuse);
SpecularProduct = product(table_material.reflect_specular, light_specular);
glUniform4fv(ap_location, 1, (GLfloat *) &AmbientProduct);
glUniform4fv(dp_location, 1, (GLfloat *) &DiffuseProduct);
glUniform4fv(sp_location, 1, (GLfloat *) &SpecularProduct);
glUniform1fv(shininess_location, 1, (GLfloat *) &table_material.shininess);

glDrawArrays(GL_TRIANGLES, 0, num_cube_vertices);
:
```

Note that the `glUniform1i()` function allows you to send one integer to the vertex shader, the `glUniform1fv()` function lets you send an array of floating-point (only one element is sent in the above example), and the `glUniform4fv()` function lets you send an array of four-element vector.

Individual or Team of Two

For this project, you can either work as a two-person team or by yourself. If you decide to work as a team, please email me the name of your team and both your name and the name your teammate as soon as possible.

Submission

The due date of this project is stated on the CourseWeb. Late submissions will not be accepted. Zip all files related to this project into the file named `project2.zip` and submit it via CourseWeb. After the due date, you must demonstrate your project to either TA or me within a week after the due date.