Project 3 - Lighting Model, Shadows, Textures, and Transformations

CS 1566 — Introduction to Computer Graphics

Check the Due Date on the Canvas

The purpose of this project are as follows:

- Model multiple objects where each object has a unique textures
- Animate multiple objects where each object has a unique movement
- Apply the lighting model in the fragment shader
- Create fake shadows in the vertex shader
- Allow a user to change the light location using keyboard
- Allow a user to change their eye location using keyboard
- Allow a user to look from behind an object

Generate a World (Object) Frame

For this project, your world will be a simple pool table with 16 billiard balls, and a light bulb with the following description:

- The pool table is a flat surface size 5×5. This can be done by creating two triangle on the plane $y = 0$. For better lighting effect, multiple smaller triangle are preferred but not required. The color of the surface should be green.
- Sixteen billiard balls where each ball has the radius exactly 0.1. Each ball will have a unique texture which will be discussed later. The initial position of each ball should be $(x, 0.1, z)$ such that they are organized as shown in Figure 1. Note that all balls are sitting on top of the pool table.
- The light bulb is simply a sphere with radius 0.05 with the white color and its center should be located at $(0.0, 1.0, 0.0)$. Note that the actual light position (for the lighting model) will also be at $(0.0, 1.0, 0.0)$. This will make the outside surface of the light bulb to be the ambient white (light position is inside the light bulb). To make it show the bright white color instead of ambient white, simply turn normals inward instead of regular outward for a sphere.

For the model view, the eye point should be at $(0.0, 2.0, 5.0)$ and the at point should be at $(0.0, 0.0, 0.0)$. Obviously, the up vector should be $(0.0, 1.0, 0.0)$. For the frustum, simply use -0.1, 0.1, -0.1, 0.1, -0.5, -20.0 for left, right, bottom, top, near, and far, respectively. With the above description, your work should look like the following:

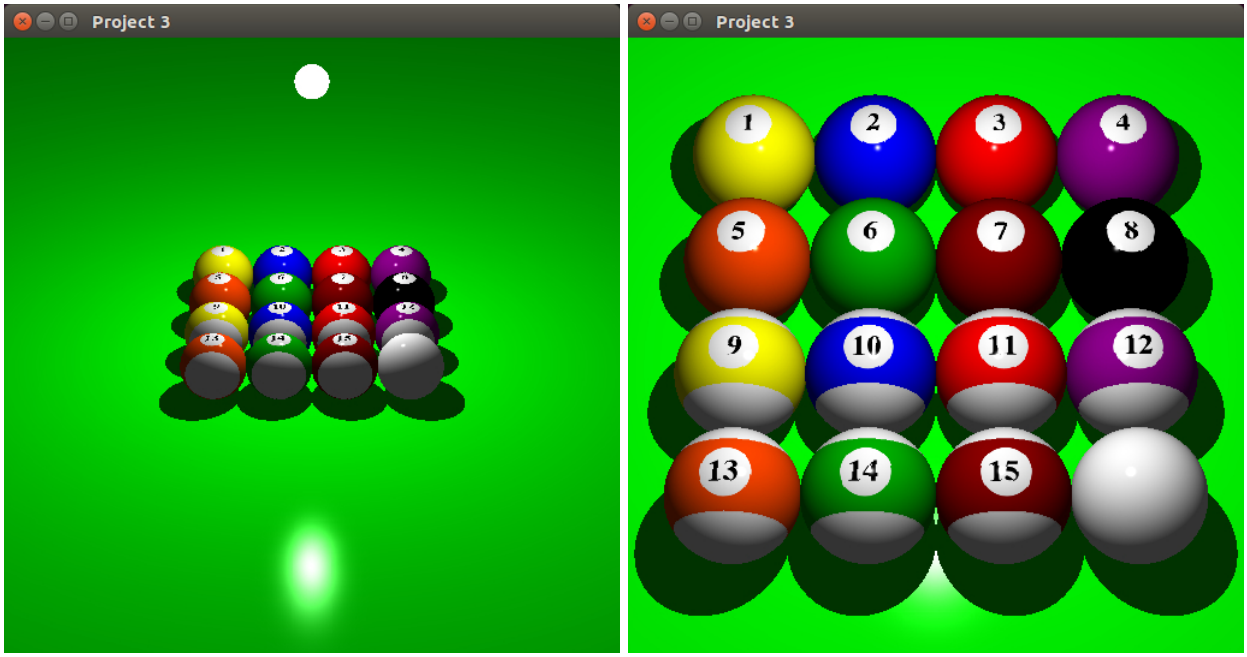
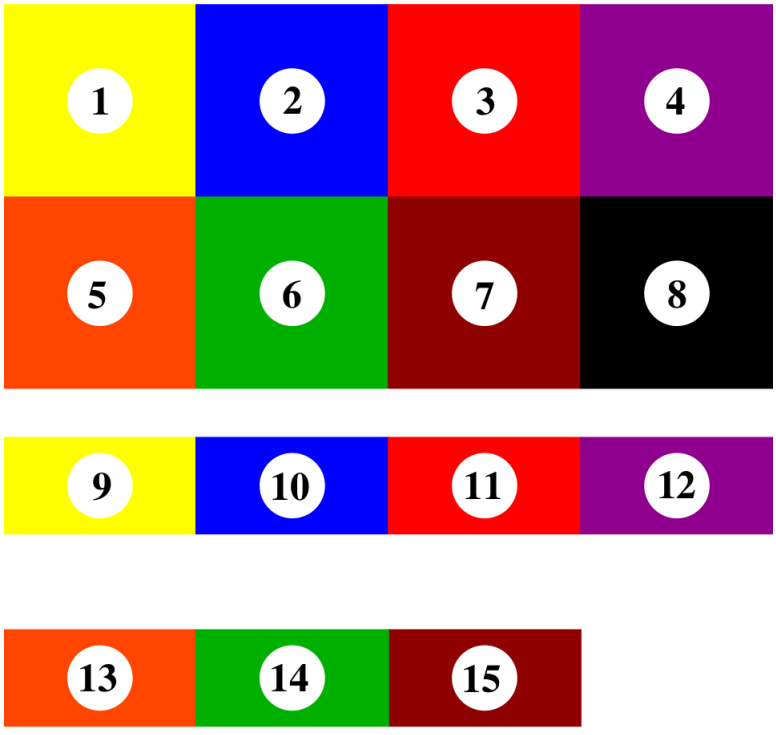


Figure 1

Note that the center point among balls 6, 7, 10, and 11 is $(0.0, 0.1, 0.0)$.

Texture and Color

The texture that we are going to use in this project is shown below:



The texture file named `pb_512_512.raw` is provided. It is a raw RGB image file (as we discussed in class) of size 512×512. Note that the texture coordinates of each ball texture is slightly off when I created the texture. Make sure to use the following coordinates when you create texture of each ball:

Ball	Top Left Coordinate	Bottom Right Coordinate
1	(0.01, 0.01)	(0.24, 0.24)
2	(0.26, 0.01)	(0.49, 0.24)
3	(0.51, 0.01)	(0.74, 0.24)
4	(0.76, 0.01)	(0.99, 0.24)
5	(0.01, 0.26)	(0.24, 0.49)
6	(0.26, 0.26)	(0.49, 0.49)
7	(0.51, 0.26)	(0.74, 0.49)
8	(0.76, 0.26)	(0.99, 0.49)
9	(0.01, 0.511)	(0.24, 0.741)
10	(0.26, 0.511)	(0.49, 0.741)
11	(0.51, 0.511)	(0.74, 0.741)
12	(0.76, 0.511)	(0.99, 0.741)
13	(0.01, 0.7625)	(0.24, 0.99)
14	(0.26, 0.7625)	(0.49, 0.99)
15	(0.51, 0.7625)	(0.74, 0.99)
White	(0.76, 0.7625)	(0.99, 0.99)

Feel free to fine tune the above coordinates to your liking.

For simplicity, once you have created the set of vertices of a sphere of radius 1.0. Simply create a formula that turn the point (x, y, z) where $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$ to a texture coordinate (s, t) where (s, t) is within the top left coordinate and bottom right coordinate. The value of z should not have any effect.

For this project, there are two objects that do not have a texture, surface of the pool table and the light bulb. For this project, we are going to use both texture and color. In other words, each vertex will contain the following attributes:

- position
- color
- normal
- texture coordinate

The color attribute of each pool ball will contain a bogus color that we are not going to use, and texture coordinate attributes of the pool table and the light bulb will contain a bogus coordinate that we are not going to use. We will tell the fragment shader when to use texture and when to use color using a `uniform` variable:

```
:
varying vec4 color;
varying vec2 texCoord;
:
uniform sample2D texture;
```

```
uniform int use_texture;
:
void main()
{
    vec4 the_color = color;

    if(use_texture == 1)
    {
        the_color = texture2D(texture, texCoord);
    }
    :
    gl_FragColor = ...
}
```

From the above fragment shader code outline, the variable `the_color` is set to the `color` from the color of vertex. However, if `use_texture` is 1, the `the_color` is overwritten by the color from the texture.

Animation

First thing first, ball movements should be slow. We need to be able to visually inspect the movement of your billiard balls. For this project, there are two set of animations:

1. Move from initial location to lineup location. The initial location is shown in Figure 1. The following is how balls should be lined up:

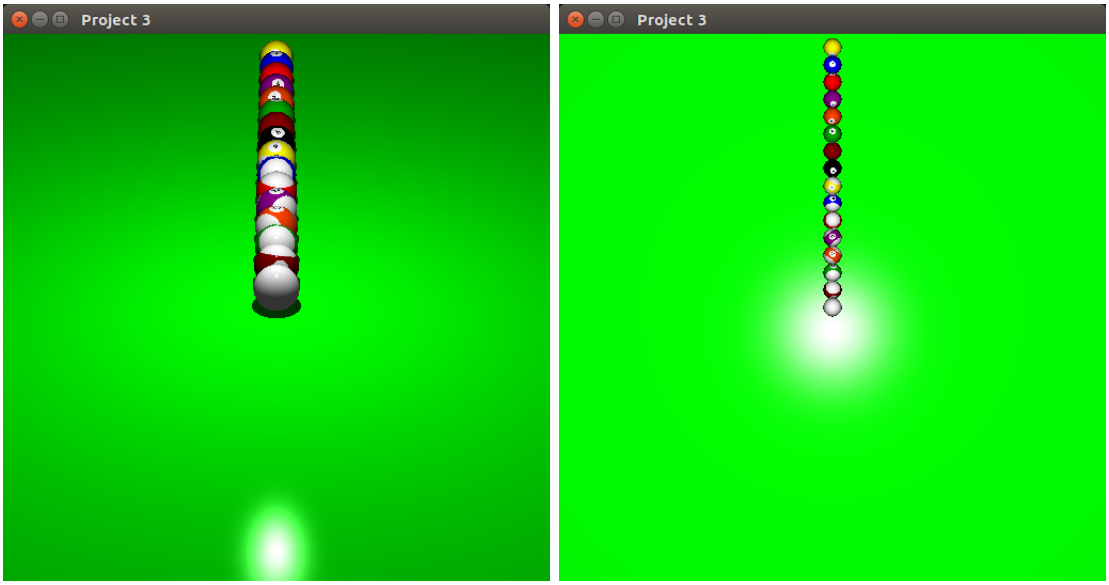


Figure 2

From the above figure, all balls are sit right next to each other on the negative z-axis where the white ball is at $(0.0, 0.1, 0.0)$. Note that since the radius of each ball is 0.1, the location of the first ball (ball 1) is at $(0.0, 0.1, -3.0)$.

2. Orbit around the white ball from their lineup locations. For this animation, all balls are simply rotate about the y-axis. The amount of degree to rotate per iteration should not be the same. The farther the ball from the white ball, the slower (degrees per iteration) it should rotate about y-axis.

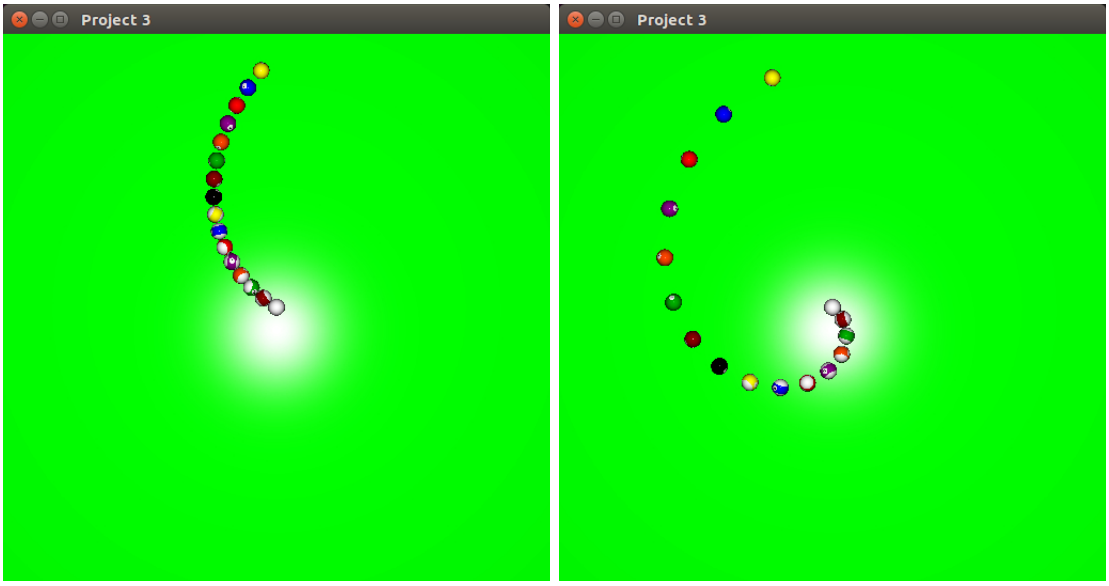


Figure 3

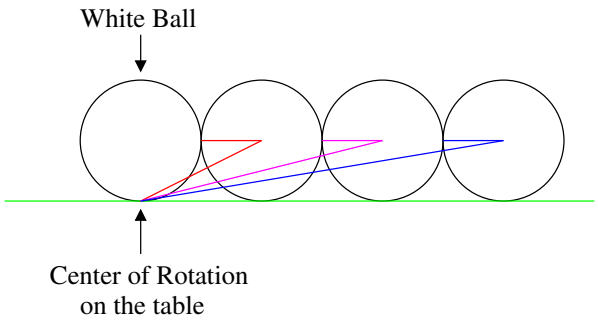
The above image on the left shows locations of ball just after they start orbiting around the white ball. The image on the right shows locations of ball a little bit after the one from the left.

For this project, you are going to make the movement of those ball to be as real as possible. In other words, you are not going to just simply translate the ball from one position to another. You are going to make them look like they are rolling from one position to another.

To make them look realistic, you **MUST** incorporate the rolling effect into balls transformation matrix. Recall that there are two type of movements, straight line and curve.

- For the straight line rolling, the about vector of rotation must be perpendicular with the vector representing the traveling direction and the y-axis. For example, suppose a ball is moving from the point p_1 to p_2, the direction vector is $p_2 - p_1$. The about vector would be the y-axis cross product with $p_2 - p_2$. Once you normalize the about vector, you can use the same method in project 1 to generate the arbitrary rotation matrix.

- For the curve (circle) movement, the about vector of rolling must tilt inward a little bit. The amount of tilt is the same as the degree between the line parallel to the ground and the line from the center of the ball to the center of rotation on the ground as shown below:



The amount of tilt of the about vector for each ball shown above is the amount of degrees between two lines with the same color. Note that the closer to the white ball, the greater the amount of tilt. Again, once you have the about vector, use the same method as in project 1 to calculate the rotation matrix.

The make the rolling effect looks even more realistic, the amount of rotation (in degree) depends on the distance it travel per iteration and the radius of the ball.

Lighting Model

For the lighting model, we are **NOT** going to incorporate materials. We will simply use the mathematics behind ambient, diffuse, and specular of the lighting model discussed in class to modify the value of color. Note that the value of a color in our case can come from the texture or from the vertex attribute. In doing so, it will simplify this project by a lot.

For this project, you **must apply the lighting model in your fragment shader**. You need to calculate the value of ambient, diffuse, and specular for each fragment as shown below:

```
:
    vec4 the_color = color;

    if(use_texture == 1)
    {
        the_color = texture2D(texture, texCoord);
    }

    ambient = the_color * 0.2;
    diffuse = the_color * something;
    specular = vec4(1.0, 1.0, 1.0, 1.0) * something_else;
    gl_FragColor = ambient + diffuse + specular;
    :
}
```

Note that the `use_texture` variable is a uniform variable of type integer. This will allow you to choose whether to use the texture or to use just color (from vertex attribute). Table and light bulb do not have texture. They only have color. Because of this, your vertex attribute of every object must contain the following:

- position

- color
- normal
- texture coordinate

The color of a ball can be any colors since we are not going to use them. Similarly, the texture coordinate of the table or the light bulb can be any coordinates since we do not use them.

Fake Shadow

For this project, you are going to create a shadow of each ball using the same method discussed in class. The calculation can be found in the lab 8. Note that 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.

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 $y = 0.001$. Note that instead of projected to the point $y = 0$, we project to the plane $y = 0.001$ so that the shadow will be just a little higher than the pool table. Otherwise, we may have some flicking on the shadow.

```
:
uniform int is_shadow;
:

void main()
{
    :
    if(is_shadow == 1)
    {
        :
        color = shadow_color;
        vec4 current_position = ctm * vPosition;
        float x = ...;
        float y = 0.001;
        float z = ...;
        gl_Position = projection * model_view * vec4(x, y, z, 1.0);
        :
    }
    else
    {
        :
        color = vColor;
```

```
        gl_position = projection * model_view * ctm * vPosition;
        :
    }
    :
}
```

Note that you have to locate the variable `is_shadow` 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, "is_shadow");
    :
}

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

Light Location

Recall that the initial position of the light bulb (light source) is at $(0.0, 1.0, 0.0)$. For this project, a user must be able to change the location of the light source using keyboard. For this part, pick 6 keys of your choice for the purpose of changing the light position. For simplicity, a key will be used to increase the x component of the position by 0.1, a key will be used to decrease the x component of the position by 0.1, a key will be used to increase the y component of the position by 0.1, and so on.

Once a user press a key to change the light position, the actual position of the light bulb must be changed accordingly. Note that the following must be changed as well:

- diffuse
- specular
- shadow

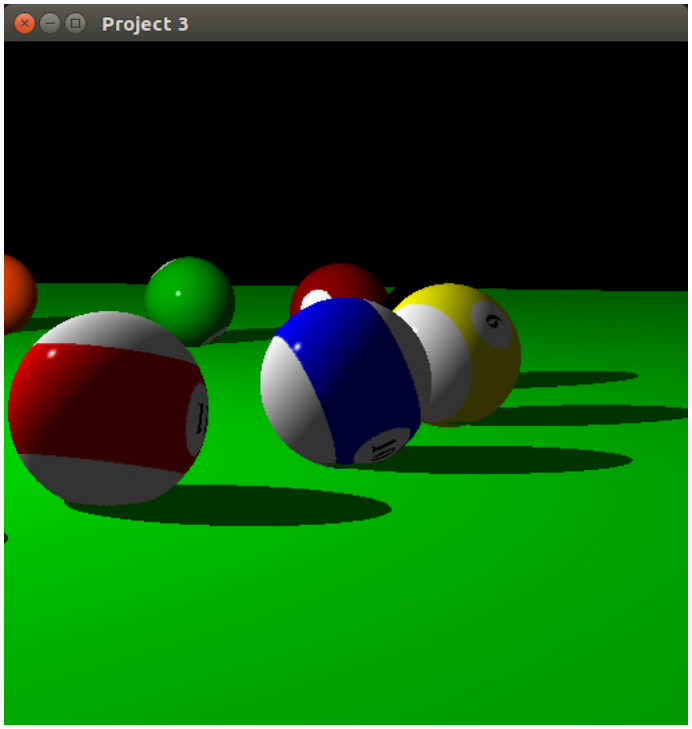
If you implement your lighting model in the fragment shader correctly, by changing the light position (a `uniform` variable) and redraw, you should see the effect immediately.

Eye Location (Look At)

For this project, you can create your own initial eye position. You always look at the origin. Assume that the distance of the eye position to the origin is d. For this project, the eye position can be changed. Pick 6 keys for the purpose of changing the eye position. A key will be used to reduce the value of d (closer) and another will be used to increase the value of d (farther). The rest of four keys will be used to move eye point to left, right, up, and down position while maintaining the distance d. Again, always look at the origin. When moving the eye point left, right, up, or down, just imagine that the eye point is sitting on a glass ball with the radius d.

Follow a Ball

For this part, you must allow a user to view from behind a ball and simply follow the ball around the table. This can be done by keeping the eye position somewhere behind the ball with a fix distance and look at the ball. The following is an example of a view behind the ball number 10:



Since there are 15 balls that moves, pick 15 keys preferably 1 to f to pick which ball do you want to view from behind. Pick another key to reset the view back from afar.

Grading Rubric

The more detail rubric can be found under this project in the Canvas. The following is the list of feature and points:

Feature	Points
Initial world with 16 balls, table, and the light bulb	10
Use texture to all 16 balls	10
Change the eye point around the table using keyboard	10
Change the eye point to behind a ball with tilted up vector during circling	10
Animation	
Initial to Line up	5
Rolling Effect (straight line)	10
Circle Around the White Ball	5
Rolling Effect	10
Rolling Effect with tilt	5
Changing the light position using keyboard	10
Light bulb position is changed accordingly	5
Lighting Model in the fragment shader	10
Diffuse lights change according to the light and object positions	20
Specular lights change according to the light, object, and eye positions	20
Fake Shadows	5
Shadow effects change according to the light and object positions	5
Total	150

Submission

The due date of this project is stated on the Canvas. Late submissions will not be accepted. Zip all files/directories related to this project into the file named `project3.zip` and submit it via Canvas. After the due date/time, you must demonstrate your project to either TA during your recitation session.