Graphics and Animation - Student Report

Nicholas Walters - 22243339

CITS3003

University of Western Australia, 2020

Overall Achievements:

For this Graphics and Animation project, I have managed to complete ALL TASKS, as shown in the code. Our program meets all the required specifications (A-J) of the project. However there were some problems with the second light source not reflecting light correctly and being jumpy.

Task A:

From the original skeleton code provided, it had limited functionality.
In the original code, the program had moved the object through the z-direction, which is rotated by the z-axis.
If I want to rotate the camera through the X-Axis and Y-Axis, there must be some rotatable angles on these two axes. So I have set it up in a way where I can rotate the camera direction in any direction/way that I want (The camera variable and multiplication was already set up, making it easy) :

```
mat4 rotate = RotateX(camRotUpAndOverDeg) * RotateY(camRotSidewaysDeg);
view = Translate(0.0, 0.0, -viewDist);
view = view * rotate;
```

Task B:

The mechanism of how this task works is similar to the first question, the original code missed three additional parameters that would make the object rotate. Since the angle array is already defined, I just have to multiply them with the scale and translate structure. To get the object rotation, it is mostly the same as Task A, I need to find out where the translation of the object is. Since the provided code does not have any functionality on rotating the object, I think there might be no rotation on X, Y, Z at all, so we add the same rotation on each with different angles:

```
mat4 rotate = RotateX(sceneObj.angles[0]) * RotateY(sceneObj.angles[1]) *
RotateZ(sceneObj.angles[2]);
mat4 model = Translate(sceneObj.loc) * Scale(sceneObj.scale) * rotate;
```

Changing the Y value to negative didn't produce any results that changed rotation. However, Z-Axis and X-axis are opposite, so they needed to be changed to negative values.

Task C:

This task was a much bigger task which requires me to change 2 or more different functions.
I then need to put these functions into the `setToolCallBacks` function as additional two

parameters.
We need to add another functionality onto this object, this first thing we do is make a button to add this extra functionality:

```
glutAddMenuEntry("Ambient/Diffuse/Specular/Shine", 20);
```

I then set entry port to 20 for when you click this button, and define it in the materialMenu() function:

```
if (id == 20)
	{
	toolObj = currObject;
	setToolCallbacks(adjustAmbientDiffuse, mat2(1, 0, 0, 1),
adjustShineSpecular, mat2(1, 0, 0, 10));
	}
```

After creating the button, and after creating functionality for the button, We then create the appropriate callback functions:

```
static void adjustShineSpecular(vec2 shnSpc)
{
	sceneObjs[toolObj].shine += shnSpc[0];
	sceneObjs[toolObj].specular += shnSpc[1];
}

static void adjustAmbientDiffuse(vec2 ambDif)
{
	sceneObjs[toolObj].ambient += ambDif[0];
	sceneObjs[toolObj].diffuse += ambDif[1];
}
```

Task D:

This task only requires one line of code to complete. The nearDist in the reshape function defines the distance of the camera. We now change it to:

```
GLfloat nearDist = 0.01;
```

Task E:

For this question, In order to reshape the window and shrink objects proportional to it, you have to consider whether the height of the window is greater than the width, or vise versa.
Use width divided by height or the other way around to get the proportional multiplier. I looked up the detailed explanation of the function frustum() from:
https://stackoverflow.com/questions/23309930/what-do-the-arguments-for-frustum-in-opengl-mean
In fructum, the first two parameters determine the left and right view of the view plame, the third and fourth parameters determine the top and bottom view of the view plane. The last two are near and far, they define how far the plane is from the camera. It doesn't affect how translation is made, which means they only affect the view distance from the camera.

If the window height is decreased (width not changed), the left and right view should change with it accordingly.
We must now consider top and bottom also, which adjusts according to the width. If the width is equal to or greater than the height, the top and bottom should be unchanged.

Finally, because ‘near’ and ‘far’ don’t affect translation, we can set the value to: 100.0

```
    //==============PART D & E================//

    GLfloat nearDist = 0.01; // PART D
    GLfloat adjust = nearDist;

    if (width < height)
    {
    adjust = adjust * (float)width / (float)height;
    }

    projection = Frustum(-nearDist * (float)width / (float)height,
                     nearDist * (float)width / (float)height,
                     -nearDist, nearDist,
                     adjust, 100.0);
```

Task F, G, H, and I:

These task's requires editing the shaders of the program. All Calculations are passed through to the fragment shader.
The reason why I linked all these tasks together is because, we calculate the light source and attenuation in the vertex shader, but in Task G (Shown at the start of the code shown) we move them to the fragment shader, which means that all the light calculations are done in the fragment shader. These processes/tasks are highly linked together, thus I must show the main() function of the fragment shader to show how these questions connect to each other.

I will now explain the processes involved with each question:
Note: The start and end of each part is shown in the code below

Part F: involves the reduction in light with regards to distance. We must first calculate the distance of the light position using 'Lvec' and then compare that to the distance of where that light is in the scene, using variables shown below as 'distanceSq'.
After we get the distances of the environment from the light (and its light position), we need to set the colour of the light using 'gl_FragColor'

PART G and I:
This is the harder task to complete, as it deals with light interactions and light as a measure of distance to objects and surfaces. Dealing with these complex calculations requires some understanding of the Blinn-Phong shading calculation, while also combining the other tasks aspects .
I learnt how to implement it through the following resource:
https://learnopengl.com/Advanced-Lighting/Advanced-Lighting

In order to implement the Blinn-Phong model, we need to get the Light direction, View position, and halfway directions of the light respectively. We do this by normalising the subtraction of much of our current positions of the light and the world (Light Position, Fragment Position and View Position) to get the final output.

$$\bar{H} = \frac{\bar{L} + \bar{V}}{\|\bar{L} + \bar{V}\|}$$

-- Normalisation of the light and view vectors to get the halfway vector of light.

PART H:

I referenced the pseudo code from: https://learnopengl.com/Lighting/Basic-Lighting

Where variable 'specular' = Ks1 * light-Brightness * specularProduct * whiteness

We now have specular light for two light sources, because the first light source needs to change with the distance, it needs to multiply by attenuation/whiteness. And while assigning colours, we add the specular light on it.

The below code shows the calculation of the globalAmbient, ambient for both light sources, diffuse for both light sources and specular. Where the colour is the addition of the globalAmbient + diffuse.

Combination of all the code from F, G, H and I. This is inside the fragment shader, and all the questions are closely linked together:

```
void main()
{
      //*****************  PART G  ******************//
      vec4 color;

      // The vector to the light from the vertex
      vec3 Lvec = LightPosition.xyz - globPosition;

      // Unit direction vectors for Blinn-Phong shading calculation
      vec3 L = normalize( Lvec );   // Direction to the light source
      vec3 E = normalize( -globPosition);   // Direction to the eye/camera
      vec3 H = normalize( L + E );  // Halfway vector

      // Transform vertex normal into eye coordinates (assumes scaling
      // is uniform across dimensions)
      vec3 N = normalize( (ModelView*vec4(globNormal, 0.0)).xyz );

      // Compute terms in the illumination equation
      vec3 ambient = AmbientProduct;

      float Kd = max( dot(L, N), 0.0 );
      vec3  diffuse = Kd*DiffuseProduct;

      float Ks = pow( max(dot(N, H), 0.0), Shininess );
```

```
    //**************  PART H  ******************//
    vec3 whiter = vec3(0.5, 0.5, 0.5);
    vec3  specular = Ks * (SpecularProduct + whiter);
    //**************   PART H  ******************//

    if (dot(L, N) < 0.0 ) {
     specular = vec3(0.0, 0.0, 0.0);
    }

    float distanceSq = length(Lvec) * length(Lvec); // Part F
    float a = 1.0;
    float b = 2.0;
    float dist = length(Lvec);
float light = 1.0/(a + b*dist + distanceSq); //Part F

//***************   END OF PART G   *************//

//===============PART I==================//

    // The vector to the light from the ORIGIN
    vec3 Lvec2 = (LightPosition2).xyz;
    float distanceSq2 = length(Lvec2) * length(Lvec2);
    float light2 = 1.0/(a + b*dist + distanceSq2);


    // Unit direction vectors for Blinn-Phong shading calculation
    vec3 L2 = normalize( Lvec2 );   // Direction to the light source
    vec3 H2 = normalize( L2 + E );  // Halfway vector

    float Kd2 = max( dot(L2, N), 0.0 );
    vec3  diffuse2 = Kd2*DiffuseProduct;

    float Ks2 = pow( max(dot(N, H2), 0.0), Shininess );

    vec3  specular2 = Ks2 * (SpecularProduct + whiter);

    if (dot(L2, N) < 0.0 ) {
    specular2 = vec3(0.0, 0.0, 0.0);
    }

//****************  END OF PART I ***************//
```

```
      //*************** PART F ******************//
      // globalAmbient is independent of distance from the light source
      vec3 globalAmbient = vec3(0.1, 0.1, 0.1);
      color.rgb = globalAmbient  + ambient + diffuse + specular + light +
light2;
      //***************  PART F *****************//
      color.a = 1.0;

      gl_FragColor = color * texture2D( texture, texCoord * texScale );

}
```

Task J:

Simply add RGB colour to the second light source.

Nicholas Walters - CITS3003 Project Report, 2020
22243339