CITS3003 Project 2022 Sem 1

Kristijan Korunoski - 22966841

Luke Kirkby - 22885101

Most changes for each task can be found by searching for “// Task X” where X is the appropriate task

Task A

We were successful in implementing Task A.

We changed the calculation of the “view” matrix to also depend on the “camRotUpAndOverDeg” and “camRotSidewaysDeg” variables.

Before:

```
view = Translate(0.0, 0.0, -viewDist);
```

After:

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

Task B

We were successful in implementing Task B.

We changed the “model” matrix to be multiplied by a rotation matrix “RotObjXYZ” which contains information on the rotations in the X, Y, and Z directions based on the objects angles.
Before:

```
   mat4 model = Translate(sceneObj.loc) * Scale(sceneObj.scale);
```

After

```
   mat4 RotObjXYZ = RotateX(sceneObj.angles[0]) * RotateY(sceneObj.angles[1]) *
RotateZ(sceneObj.angles[2]);

   mat4 model = Translate(sceneObj.loc) * RotObjXYZ * Scale(sceneObj.scale);
```

In fStart.glsl we added a float texScale that was already passed from the main source code. We changed the calculation of gl_FragColor to be as follows, taking the texScale into calculation.

Before:

```
gl_FragColor = color * texture2D( texture, texCoord * 2.0 );
```

After:

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

Task C

We were successful in implementing Task C.

We did this by creating the functions to adjust lighting variables based on mouse movement.
Added functions:

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

static void adjustSpecularShine(vec2 ss){
   sceneObjs[toolObj].specular += ss[0];
   sceneObjs[toolObj].shine -= 20 * ss[1];
}
```

We also made adjustments to the materialMenu function so that if the option is selected in the material menu we call the above functions:

```
   // You'll need to fill in the remaining menu items here.
   // Part C
   else {
       toolObj = currObject;
       setToolCallbacks(adjustAmbientDiffuse, mat2(1, 0, 0, 1),
                        adjustSpecularShine, mat2(1, 0, 0, 1));
   }
```

Also changed the menu entry to not say unimplimented.

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

Task D

We were successful in implementing Task D.

To do this we changed the near dist value to be much smaller.

```
  GLfloat nearDist = 0.01;
```

Task E

We were successful in implementing Task E.

To implement this task we implemented a condition to determine which way the image is compressed and used the appropriate values in the Frustrum.

```
   if (width >= height) {
       projection = Frustum(-nearDist * (float) width / (float) height,
                            nearDist * (float) width / (float) height,
                            -nearDist,
                            nearDist,
                            nearDist,
                            100.0);
   } else {
       projection = Frustum(-nearDist,
                            nearDist,
                            -nearDist * (float) height / (float) width,
                            nearDist * (float) height / (float) width,
                            nearDist,
                            100.0);
   }
```

Task F

We were successful in implementing Task F.

To complete this task we created “distance” and “scaleDist” variables as such in vShader.glsl:

```
    float distance = sqrt(Lvec[0]*Lvec[0] + Lvec[1]*Lvec[1] + Lvec[2]*Lvec[2]);
    float scaleDist = 1.0/(distance*distance);
```

We then changed color.rgb to have scaleDist multiply on (ambient + diffuse + specular) so that the light from the source is scaled by 1/distance^2

```
    color.rgb = globalAmbient + scaleDist*(ambient + diffuse + specular);
```

Task G

We were successful in implementing Task G.

To do this we commented out the lighting calculations from the vertex shader and put them in the fragment shader. We needed to add some varying variables to communicate between the vertex and fragment shader.

There are comments:

```
// Task G
```

In the fragment shader at points where changes have been made.

Task H

We were successful in implementing Task H.

We did this by creating a bw vector that has no dependence on the red green or blue components of the objects and lights to replace the rgb multiplier when passing SpecularProduct to shaders:

```
        vec3 bw = so.brightness * lightObj1.brightness;
```

When passing SpecularProduct to the shaders we use the “bw” vector so that the specular component of the light shines towards white rather than a certain color:

```
        glUniform3fv(glGetUniformLocation(shaderProgram, "SpecularProduct"), 1,
so.specular * bw);
```

Task I

We were successful in implementing Task I.

In scene-start.cpp we first create a new light object:

```
    addObject(55); // Sphere for the second light
    sceneObjs[2].loc = vec4(3.0, 4.0, 2.0, 1.0);
    sceneObjs[2].scale = 0.1;
    sceneObjs[2].texId = 0; // Plain texture
    sceneObjs[2].brightness = 0.2; // The light's brightness is 5 times this (below).
```

We then create a “lightObj2” variable to access this light and create “lightPosition2” to know the light position in view coordinates.

```
    SceneObject lightObj2 = sceneObjs[2];
    vec4 lightPosition2 = view * lightObj2.loc;
```

We pass to the shader program the light position.

```
    glUniform4fv(glGetUniformLocation(shaderProgram, "LightPosition2"),
                 1, lightPosition2);
    CheckError();
```

We also pass the ambient, diffuse, and specular products for light 2 to the shader.

In the light menu we add the tool callbacks for specific options.

```
    } else if (id == 80) {
        toolObj = 2;
        setToolCallbacks(adjustLocXZ, camRotZ(),
                         adjustBrightnessY, mat2(1.0, 0.0, 0.0, 10.0));
    } else if (id >= 81 && id <= 84) {
        toolObj = 2;
        setToolCallbacks(adjustRedGreen, mat2(1.0, 0, 0, 1.0),
                         adjustBlueBrightness, mat2(1.0, 0, 0, 1.0));
```

We also add the options as menu entries.

```
    glutAddMenuEntry("Move Light 2", 80);
    glutAddMenuEntry("R/G/B/All Light 2", 81);
```

In fStart.glsl we create variables to be passed from scene-start.cpp.

```
uniform vec3 AmbientProduct2, DiffuseProduct2, SpecularProduct2;
uniform vec4 LightPosition2;
```

We set L2vec as the direction of the light to the origin.

```
   vec3 L2vec = LightPosition2.xyz; // Direction of light is to origin not position
```

Set unit vectors for Blinn-Phong shading.

```
   vec3 L2 = normalize( L2vec );  // Direction to light source
   vec3 H2 = normalize( L2 + E );// Halfway vector for light 2
```

Compute ambient, specular, diffuse lights.

```
   vec3 ambient2 = AmbientProduct2;

   float Kd2 = max( dot(L2, N), 0.0 );
   vec3  diffuse2 = Kd2 * DiffuseProduct2;

   float Ks2 = pow( max(dot(N, H2), 0.0), Shininess );
   vec3  specular2 = Ks2 * SpecularProduct2;

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

Set fragment color.

```
color.rgb = globalAmbient + scaleDist*(ambient1 + diffuse1 + specular1) + (ambient2 +
diffuse2 + specular2);
       color.a = 1.0;
```

Task J

We were successful in implementing Task J.

In scene-start.cpp we create duplication and deletion functions:

```
// Task J duplication
static void dupeObject(int id) { ...
// Task J delete
static void deleteObject(int id){ ...
```

Create light object for light 3 with a cutoff specified.

```
   addObject(55); // Sphere for the third light
   sceneObjs[3].loc = vec4(1.0, 1.0, 2.0, 1.0);
   sceneObjs[3].angles[0] = 0.0;
   sceneObjs[3].angles[1] = 90.0;
   sceneObjs[3].angles[2] = 0.0;
   sceneObjs[3].scale = 0.1;
   sceneObjs[3].texId = 0; // Plain texture
   sceneObjs[3].brightness = 0.2; // The light's brightness is 5 times this (below).
   sceneObjs[3].cutOff = 0.9; //Angle at which light is cut off from emitting
```

Variables for light 3 (similar to light 2) now with angles for spotlight cone.

```
   SceneObject lightObj3 = sceneObjs[3];
   float L3pitch = lightObj3.angles[1];
   float L3yaw = lightObj3.angles[2];
   vec4 lightPosition3 = view * lightObj3.loc;
```

Pass variables to shader.

```
   glUniform4fv(glGetUniformLocation(shaderProgram, "LightPosition3"),
                1, lightPosition3);
   // Task J Pass Light 3 specific variables
   glUniform1f(glGetUniformLocation(shaderProgram, "Light3CutOff"), lightObj3.cutOff);
   glUniform1f(glGetUniformLocation(shaderProgram, "L3pitch"), L3pitch);
   glUniform1f(glGetUniformLocation(shaderProgram, "L3yaw"), L3yaw);
   CheckError();
```

Function for light 3 rotation.

```
static void RotateLight(vec2 xz)
{
   sceneObjs[toolObj].angles[2]+=-20*xz[0];
   sceneObjs[toolObj].angles[1]+=-20*xz[1];
}
```

Light menu additions for light 3.

```
   } else if (id == 90) {
       toolObj = 3;
       setToolCallbacks(adjustLocXZ, camRotZ(), // Task J spotlight movement
                        adjustBrightnessY, mat2(1.0, 0.0, 0.0, 10.0));
   } else if (id == 91) {
       toolObj = 3;
       setToolCallbacks(adjustRedGreen, mat2(1.0, 0, 0, 1.0),
                        adjustBlueBrightness, mat2(1.0, 0, 0, 1.0));
   } else if (id == 92){
       toolObj = 3;
       setToolCallbacks(RotateLight, camRotZ(), // Task J spotlight rotation
                        adjustBrightnessY, mat2( 1.0, 0.0, 0.0, 10.0) );
   } else {
```

Light 3 menu entries.

```
   glutAddMenuEntry("Move Light 3", 90);
   glutAddMenuEntry("R/G/B/All Light 3", 91);
   glutAddMenuEntry("Rotate Light 3", 92);
```

In fStart.glsl
Accept variables passed from scene-start.cpp.

```
uniform vec3 AmbientProduct, DiffuseProduct, SpecularProduct;
uniform vec3 AmbientProduct2, DiffuseProduct2, SpecularProduct2;
uniform vec3 AmbientProduct3, DiffuseProduct3, SpecularProduct3;
uniform vec4 LightPosition, LightPosition2, LightPosition3;
```

```
uniform float Light3CutOff;
uniform float L3pitch;
uniform float L3yaw;
```

Calculate L3vec, direction to point.

```
   vec3 L3vec = LightPosition3.xyz - pos;
```

Create L3Direct variable for the angle the light is pointing to

```
   vec3 L3direct; //Directional vector for light 3
   L3direct.x = cos(radians(L3yaw))*cos(radians(L3pitch));
   L3direct.y = sin(radians(L3pitch));
   L3direct.z = sin(radians(L3yaw))*cos(radians(L3pitch));
```

Set unit vectors for Blinn-Phong shading.

```
   vec3 L3 = normalize( L3vec );  // Direction to light source
```

Theta determines if a point is in the spotlight.

```
   float theta = dot(L3, normalize(L3direct));
```

Compute ambient, specular, diffuse lights.

```
   vec3 ambient3 = AmbientProduct3;


   float Kd3 = max( theta, 0.0);
   vec3  diffuse3 = Kd3 * DiffuseProduct3;


   float Ks3 = pow( max(theta, 0.0), Shininess );
   vec3  specular3 = Ks3 * SpecularProduct3;


   if (theta < 0.0 ) {
       specular3 = vec3(0.0, 0.0, 0.0);
   }
```

Set light at point with condition if it is in spotlight

```
   if(theta > Light3CutOff){
       color.rgb = globalAmbient + scaleDist*(ambient1 + diffuse1 + specular1) +
(ambient2 + diffuse2 + specular2) + (ambient3 + diffuse3 + specular3);
       // Testing for Task H
       //color.rgb = globalAmbient + scaleDist*(specular1);
       color.a = 1.0;
   }
   else{
       // Task I & J
       color.rgb = globalAmbient + scaleDist*(ambient1 + diffuse1 + specular1) +
(ambient2 + diffuse2 + specular2);
       color.a = 1.0;
   }
```