Computer Graphics Assignment 2

Student ID: 312551077
Name: 薛祖恩

Implementation

1. TODO#1

I will skip TODO#1-1 and TODO#1-2 since the only thing to do is just comment and uncomment the codes. The following figure is the implementation for TODO#1.

```
// object file parser
std::string line;
std::string prefix;
std::stringstream ss;

// iterate through data
std::vector<glm::vec3> currPositions;
std::vector<glm::vec3> currNormals;
std::vector<glm::vec2> currTextcoords;

glm::vec2 tempVec2;
glm::vec3 tempVec3;

// add foo element, since index of object file starts from 1
currPositions.push_back(tempVec3);
currNormals.push_back(tempVec3);
currTextcoords.push_back(tempVec2);

GLint tempGlint = 0;
```

First, we declare variables that will be use, including *line*, that stores each line in the file; *prefix*, stores the prefix of each line. *currPositions, currNormals, currTextcoords* stores data according to different prefix. The reason to push a data into the vector before the loop is because the index of the object file starts from 1 instead of 0.

```
// read line by line
while (getline(ObjFile, line)) {
  ss.clear();
  ss.str(line);
  ss >> prefix;
  // positions
  if (prefix == "v") {
    ss >> tempVec3.x >> tempVec3.y >> tempVec3.z;
    currPositions.push_back(tempVec3);
  }
  // normals
  else if (prefix == "vn") {
    ss >> tempVec3.x >> tempVec3.y >> tempVec3.z;
    currNormals.push_back(tempVec3);
  }
  //textcoords
  else if (prefix == "vt") {
    ss >> tempVec2.x >> tempVec2.y;
    currTextcoords.push_back(tempVec2);
  }
  // push data
  else if (prefix == "f") {
    int counter = 0;
    while (ss >> tempGlint) {
      if (counter == 0) {
        m->positions.push_back(currPositions[tempGlint].x);
        m->positions.push_back(currPositions[tempGlint].y);
        m->positions.push_back(currPositions[tempGlint].z);
      } else if (counter == 1) {
        m->texcoords.push_back(currTextcoords[tempGlint].x);
        m->texcoords.push_back(currTextcoords[tempGlint].y);
      } else if (counter == 2) {
        m->normals.push_back(currNormals[tempGlint].x);
        m->normals.push_back(currNormals[tempGlint].y);
        m->normals.push_back(currNormals[tempGlint].z);
      }
      // ignore '/' and space
      if (ss.peek() == '/') {
        counter++;
        ss.ignore(1, '/');
      } else if (ss.peek() == ' ') {
        counter++;
        ss.ignore(1, ' ');
      }
      // new vector
      if (counter > 2) {
        counter = 0;
      }
    }
  }
}

// set num of vertex
m->numVertex = m->positions.size();
```

In the loop, we read one line for one iteration. According to each different prefix (“v”, “vn”, “vt”), the data will be stored into variables. If the prefix is “f”, we iterate through each vector, ignore the ‘/’ and the space to parse the object file correctly. After the iteration, we set up the *numVertex* by the size of the *positions*.

2. TODO#2

```
glGenVertexArrays(num_model, VAO);

for (int i = 0; i < num_model; i++) {
  glBindVertexArray(VAO[i]);
  Model* model = ctx->models[i];

  GLuint VBO[3];
  glGenBuffers(3, VBO);

  // pass positions to VBO
  glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->positions.size(), model->positions.data(), GL_STATIC_DRAW);
  glEnableVertexAttribArray(0);
  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);

  // pass normals to VBO
  glBindBuffer(GL_ARRAY_BUFFER, VBO[1]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->normals.size(), model->normals.data(), GL_STATIC_DRAW);
  glEnableVertexAttribArray(1);
  glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);

  // pass texture coordinates to VBO
  glBindBuffer(GL_ARRAY_BUFFER, VBO[2]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->texcoords.size(), model->texcoords.data(), GL_STATIC_DRAW);
  glEnableVertexAttribArray(2);
  glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
}

return programId != 0;
}
```

In the load function, I sample from the example.cpp file, then change the index of the VBO, and the parameters in the *glVertexAttribPointer* function.

```
glUseProgram(programId);
int obj_num = (int)ctx->objects.size();
for (int i = 0; i < obj_num; i++) {
  int modelIndex = ctx->objects[i]->modelIndex;
  glBindVertexArray(VAO[modelIndex]);

  Model* model = ctx->models[modelIndex];

  // pass matrices to shaders
  const float* p = ctx->camera->getProjectionMatrix();
  GLint pmatLoc = glGetUniformLocation(programId, "Projection");
  glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, p);

  const float* v = ctx->camera->getViewMatrix();
  GLint vmatLoc = glGetUniformLocation(programId, "ViewMatrix");
  glUniformMatrix4fv(vmatLoc, 1, GL_FALSE, v);

  const float* m = glm::value_ptr(ctx->objects[i]->transformMatrix * model->modelMatrix);
  GLint mmatLoc = glGetUniformLocation(programId, "ModelMatrix");
  glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, m);

  // activave / bind the texture
  glActiveTexture(GL_TEXTURE0 + ctx->objects[i]->textureIndex);
  glBindTexture(GL_TEXTURE_2D, model->textures[ctx->objects[i]->textureIndex]);
  glUniform1i(glGetUniformLocation(programId, "ourTexture"), ctx->objects[i]->textureIndex);

  // draw
  glDrawArrays(model->drawMode, 0, model->numVertex);
}

// unbind
glUseProgram(0);
}
```

In the *doMainLoop* function, I sample from the example.cpp file, then I set up (active and bind) the texture.

3. TODO#3

- TODO#3-1

```
Model* plane = new Model();
// size of table
float sizeX = 8.192f;
float sizeY = 1.0f;
float sizeZ = 5.12f;
// center position of the table
float centerX = 0.0f;
float centerY = 0.0f;
float centerZ = 0.0f;

// Define a helper function to add a vertex to the positions vector, total six surfaces
auto addVertex = [&](float x, float y, float z) {
  plane->positions.push_back(x + centerX);
  plane->positions.push_back(y + centerY);
  plane->positions.push_back(z + centerZ);
};

// Top surface
addVertex(-sizeX / 2, 0, sizeZ / 2);
addVertex(sizeX / 2, 0, sizeZ / 2);
addVertex(sizeX / 2, 0, -sizeZ / 2);
addVertex(-sizeX / 2, 0, -sizeZ / 2);

// Bottom surface
addVertex(-sizeX / 2, -sizeY / 2, -sizeZ / 2);
addVertex(sizeX / 2, -sizeY / 2, -sizeZ / 2);
addVertex(sizeX / 2, -sizeY / 2, sizeZ / 2);
addVertex(-sizeX / 2, -sizeY / 2, sizeZ / 2);

// Front surface
addVertex(-sizeX / 2, -sizeY / 2, sizeZ / 2);
addVertex(sizeX / 2, -sizeY / 2, sizeZ / 2);
addVertex(sizeX / 2, 0, sizeZ / 2);
addVertex(-sizeX / 2, 0, sizeZ / 2);

// Right surface
addVertex(sizeX / 2, -sizeY / 2, sizeZ / 2);
addVertex(sizeX / 2, -sizeY / 2, -sizeZ / 2);
addVertex(sizeX / 2, 0, -sizeZ / 2);
addVertex(sizeX / 2, 0, sizeZ / 2);

// Back surface
addVertex(-sizeX / 2, -sizeY / 2, -sizeZ / 2);
addVertex(-sizeX / 2, 0, -sizeZ / 2);
addVertex(sizeX / 2, 0, -sizeZ / 2);
addVertex(sizeX / 2, -sizeY / 2, -sizeZ / 2);

// Left surface
addVertex(-sizeX / 2, -sizeY / 2, -sizeZ / 2);
addVertex(-sizeX / 2, -sizeY / 2, sizeZ / 2);
addVertex(-sizeX / 2, 0, sizeZ / 2);
addVertex(-sizeX / 2, 0, -sizeZ / 2);
```

I set up the height, width, and length of the plane, then I create a helper function to create 6 different surfaces to render the table. It takes some time to get the correct parameters of the *addVertex* function.

```
// set plane normals, facing upwards
plane->normals = {0.0f, 1.0f, 0.0f,
                  0.0f, 1.0f, 0.0f,
                  0.0f, 1.0f, 0.0f,
                  0.0f, 1.0f, 0.0f};

// set texcoords
float textureSizeX = 4.096f;
float textureSizeY = 2.56f;

// Define a helper function to add texture coordinates to the texcoords vector, total six surfaces
auto addTexCoords = [&](float startX, float startY, float endX, float endY) {
  plane->texcoords.insert(plane->texcoords.end(), {startX, startY, endX, startY, endX, endY, startX, endY});
};

// Texture coordinates for each surface
// Top surface
addTexCoords(0.0f, 0.0f, sizeX / textureSizeX, sizeY / textureSizeY);

// Bottom surface
addTexCoords(0.0f, 0.0f, sizeX / textureSizeX, sizeY / textureSizeY);

// Front surface
addTexCoords(0.0f, 0.0f, sizeX / textureSizeX, sizeY / textureSizeY);

// Right surface
addTexCoords(0.0f, 0.0f, sizeZ / textureSizeX, sizeY / textureSizeY);

// Back surface
addTexCoords(0.0f, 0.0f, sizeX / textureSizeX, sizeY / textureSizeY);

// Left surface
addTexCoords(0.0f, 0.0f, sizeZ / textureSizeX, sizeY / textureSizeY);

// Add texture "../assets/models/Wood_maps/AT_Wood.jpg"
plane->textures.push_back(createTexture("../assets/models/Wood_maps/AT_Wood.jpg"));

// 8 points, each 3 coord.
plane->numVertex = 24;
plane->drawMode = GL_QUADS;

// Add model to ctx.models
ctx.models.push_back(plane);
}
```

The next part is to set up the text coordinates, I also declare a helper function to implement this feature, then I setup the *numVertex* and the *drawMode*.

- TODO#3-2

```
// TODO#3-2: Put the plane into scene
Object* planeObject = new Object(3, glm::translate(glm::identity<glm::mat4>(), glm::vec3(4.096, 0, 2.56)));
ctx.objects.push_back(planeObject);
```

The last step of TODO#3 is to push the object into the vector, the result is shown below.

4. TODO#4

- TODO#4-1

```
void main() {
  TexCoord = texCoord;
  FragPos = vec3(ModelMatrix * vec4(position.x, position.y, position.z, 1.0));
  Normal = vec3(ModelNormalMatrix * vec4(normal.x, normal.y, normal.z, 1.0));
  gl_Position = Projection * ViewMatrix * ModelMatrix * vec4(position.x, position.y, position.z, 1.0);
}
```

In light.vert file, I setup the text coordinate, fragment position, normal vertex, and position. The implementations are done according to the notes.

```
vec4 CalcDirLight(){
    vec3 norm = normalize(Normal);
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 lightDir = normalize(-dl.direction);
    vec3 reflectDir = reflect(-lightDir, norm);

    float diff = max(dot(norm, lightDir), 0.0);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);

    vec4 ambient  = vec4(dl.lightColor * material.ambient, 1.0)  * texture(ourTexture, TexCoord);
    vec4 diffuse  = vec4(dl.lightColor * material.diffuse * diff, 1.0) * texture(ourTexture, TexCoord);
    vec4 specular = vec4(dl.lightColor * material.specular * spec, 1.0) * texture(ourTexture, TexCoord);

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

```
vec4 CalcPointLight(){
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(pl.position - FragPos);
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);

    float diff = max(dot(norm, lightDir), 0.0);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);

    float distance = length(pl.position - FragPos);
    float attenuation = 1.0 / (pl.constant + pl.linear * distance + pl.quadratic * (distance * distance));

    vec4 ambient  = vec4(pl.lightColor * material.ambient, 1.0)  * texture(ourTexture, TexCoord);
    vec4 diffuse  = vec4(pl.lightColor * material.diffuse * diff, 1.0) * texture(ourTexture, TexCoord);
    vec4 specular = vec4(pl.lightColor * material.specular * spec, 1.0) * texture(ourTexture, TexCoord);

    diffuse  *= attenuation;
    specular *= attenuation;

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

```
vec4 CalcSpotLight(){
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(sl.position - FragPos);
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);

    float diff = max(dot(norm, lightDir), 0.0);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess);

    float distance = length(sl.position - FragPos);
    float attenuation = 1.0 / (sl.constant + sl.linear * distance + sl.quadratic * (distance * distance));

    float theta = dot(lightDir, normalize(-sl.direction));

    vec4 ambient = vec4(sl.lightColor * material.ambient, 1.0) * texture(ourTexture, TexCoord);
    vec4 diffuse = vec4(sl.lightColor * material.diffuse * diff, 1.0) * texture(ourTexture, TexCoord);
    vec4 specular = vec4(sl.lightColor * material.specular * spec, 1.0) * texture(ourTexture, TexCoord);

    diffuse *= attenuation;
    specular *= attenuation;

    if(theta > sl.cutOff){
        return (ambient + diffuse + specular);
    }
    else{
        return vec4(sl.lightColor * material.ambient, 1.0) * texture(ourTexture, TexCoord);
    }
}
```

```
void main() {
    color = vec4(0.0, 0.0, 0.0, 1.0);
    if(dl.enable == 1){
        color += CalcDirLight();
    }
    if(pl.enable == 1){
        color += CalcPointLight();
    }
    if(sl.enable == 1){
        color += CalcSpotLight();
    }
    if(dl.enable == 0 && pl.enable == 0 && sl.enable == 0){
        color = vec4(material.ambient, 1.0) * texture(ourTexture, TexCoord);
    }
}
```

In the light.frag file, several conditions are set, including if directional light, point light, spot light are set, and all are not enabled. The way to implement and the formula are implemented based on the notes.

- TODO#4-2

```
programId = quickCreateProgram(vertProgramFile, fragProgramFIle);

int num_model = (int)ctx->models.size();
VAO = new GLuint[num_model];
glGenVertexArrays(num_model, VAO);

for (int i = 0; i < num_model; i++) {
  glBindVertexArray(VAO[i]);
  Model* model = ctx->models[i];

  GLuint VBO[3];
  glGenBuffers(3, VBO);

  glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->positions.size(), model->positions.data(), GL_STATIC_DRAW);
  glEnableVertexAttribArray(0);
  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);

  glBindBuffer(GL_ARRAY_BUFFER, VBO[1]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->normals.size(), model->normals.data(), GL_STATIC_DRAW);
  glEnableVertexAttribArray(1);
  glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);

  glBindBuffer(GL_ARRAY_BUFFER, VBO[2]);
  glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->texcoords.size(), model->texcoords.data(), GL_STATIC_DRAW);
  glEnableVertexAttribArray(2);
  glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
}

return programId != 0;
}
```

In the load function, the implementation in this function is like the load function in basic.cpp file.

- TODO#4-3

```
glUseProgram(programId);
int obj_num = (int)ctx->objects.size();

for (int i = 0; i < obj_num; i++) {
  int modelIndex = ctx->objects[i]->modelIndex;
  glBindVertexArray(VAO[modelIndex]);

  Model* model = ctx->models[modelIndex];

  const float* p = ctx->camera->getProjectionMatrix();
  GLint pmatLoc = glGetUniformLocation(programId, "Projection");
  glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, p);

  const float* v = ctx->camera->getViewMatrix();
  GLint vmatLoc = glGetUniformLocation(programId, "ViewMatrix");
  glUniformMatrix4fv(vmatLoc, 1, GL_FALSE, v);

  const float* m = glm::value_ptr(ctx->objects[i]->transformMatrix * model->modelMatrix);
  GLint mmatLoc = glGetUniformLocation(programId, "ModelMatrix");
  glUniformMatrix4fv(mmatLoc, 1, GL_FALSE, m);

  const float* nm = glm::value_ptr(glm::transpose(glm::inverse(model->modelMatrix)));
  GLint nmmatLoc = glGetUniformLocation(programId, "ModelNormalMatrix");
  glUniformMatrix4fv(nmmatLoc, 1, GL_FALSE, nm);

  glUniform3f(glGetUniformLocation(programId, "viewPos"), ctx->camera->getPosition()[0], ctx->camera->getPosition()[1], ctx->camera->getPosition()[2]);

  // Material Parameters
  glUniform3f(glGetUniformLocation(programId, "material.ambient"), ctx->objects[i]->material.ambient.x, ctx->objects[i]->material.ambient.y, ctx->objects[i]->material.ambient.z);
  glUniform3f(glGetUniformLocation(programId, "material.diffuse"), ctx->objects[i]->material.diffuse.x, ctx->objects[i]->material.diffuse.y, ctx->objects[i]->material.diffuse.z);
  glUniform3f(glGetUniformLocation(programId, "material.specular"), ctx->objects[i]->material.specular.x, ctx->objects[i]->material.specular.y, ctx->objects[i]->material.specular.z);
  glUniform1f(glGetUniformLocation(programId, "material.shininess"), ctx->objects[i]->material.shininess);

  // Direction Light Parameters
  glUniform1i(glGetUniformLocation(programId, "dl.enable"), ctx->directionLightEnable);
  glUniform3f(glGetUniformLocation(programId, "dl.direction"), ctx->directionLightDirection.x, ctx->directionLightDirection.y, ctx->directionLightDirection.z);
  glUniform3f(glGetUniformLocation(programId, "dl.lightColor"), ctx->directionLightColor.r, ctx->directionLightColor.g, ctx->directionLightColor.b);

  // Point Light Parameters
  glUniform1i(glGetUniformLocation(programId, "pl.enable"), ctx->pointLightEnable);
  glUniform3f(glGetUniformLocation(programId, "pl.position"), ctx->pointLightPosition.x, ctx->pointLightPosition.y, ctx->pointLightPosition.z);
  glUniform3f(glGetUniformLocation(programId, "pl.lightColor"), ctx->pointLightColor.r, ctx->pointLightColor.g, ctx->pointLightColor.b);
  glUniform1f(glGetUniformLocation(programId, "pl.constant"), ctx->pointLightConstant);
  glUniform1f(glGetUniformLocation(programId, "pl.linear"), ctx->pointLightLinear);
  glUniform1f(glGetUniformLocation(programId, "pl.quadratic"), ctx->pointLightQuardratic);

  // Spotlight Parameters
  glUniform1i(glGetUniformLocation(programId, "sl.enable"), ctx->spotLightEnable);
  glUniform3f(glGetUniformLocation(programId, "sl.position"), ctx->spotLightPosition.x, ctx->spotLightPosition.y, ctx->spotLightPosition.z);
  glUniform3f(glGetUniformLocation(programId, "sl.direction"), ctx->spotLightDirection.x, ctx->spotLightDirection.y, ctx->spotLightDirection.z);
  glUniform3f(glGetUniformLocation(programId, "sl.lightColor"), ctx->spotLightColor.r, ctx->spotLightColor.g, ctx->spotLightColor.b);
  glUniform1f(glGetUniformLocation(programId, "sl.cutOff"), ctx->spotLightCutOff);
  glUniform1f(glGetUniformLocation(programId, "sl.constant"), ctx->spotLightConstant);
  glUniform1f(glGetUniformLocation(programId, "sl.linear"), ctx->spotLightLinear);
  glUniform1f(glGetUniformLocation(programId, "sl.quadratic"), ctx->spotLightQuardratic);

  // draw
  glActiveTexture(GL_TEXTURE0 + ctx->objects[i]->textureIndex);
  glBindTexture(GL_TEXTURE_2D, model->textures[ctx->objects[i]->textureIndex]);
  glUniform1i(glGetUniformLocation(programId, "ourTexture"), ctx->objects[i]->textureIndex);
  glDrawArrays(model->drawMode, 0, model->numVertex);
}
glBindVertexArray(0);
glUseProgram(0);
}
```

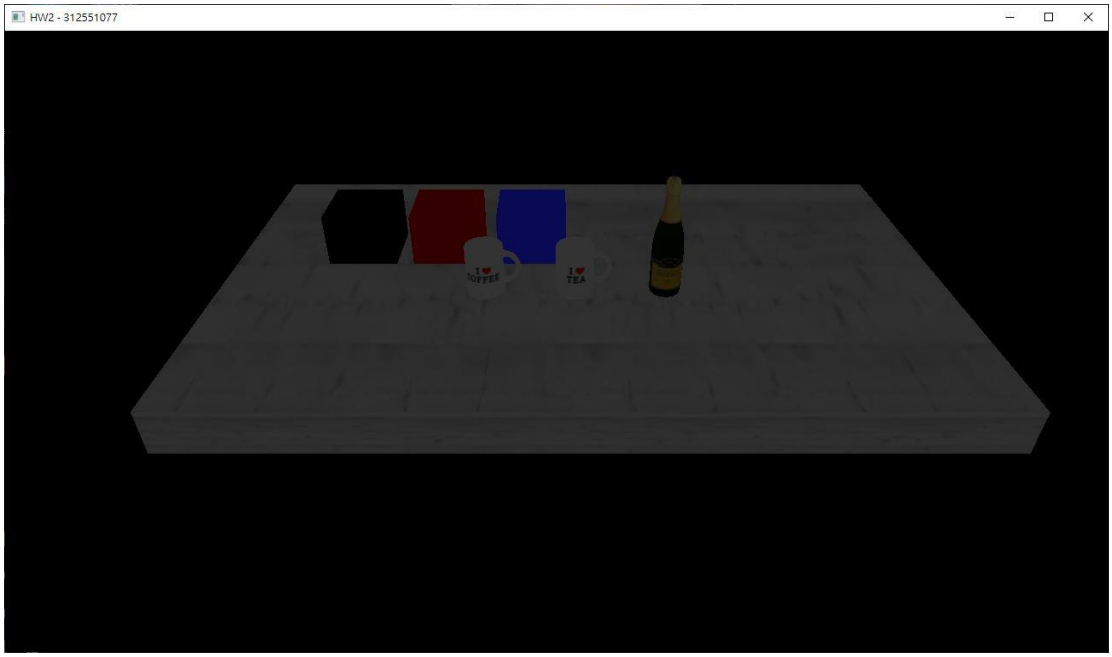
In the *doMainLoop* function, the upper part is the same as the function in basic.cpp.

The lower part is to set up all the attributes of material parameters, the direction light parameters, the point light parameters, and the spotlight parameters. Then I active and bind the textures and draw.

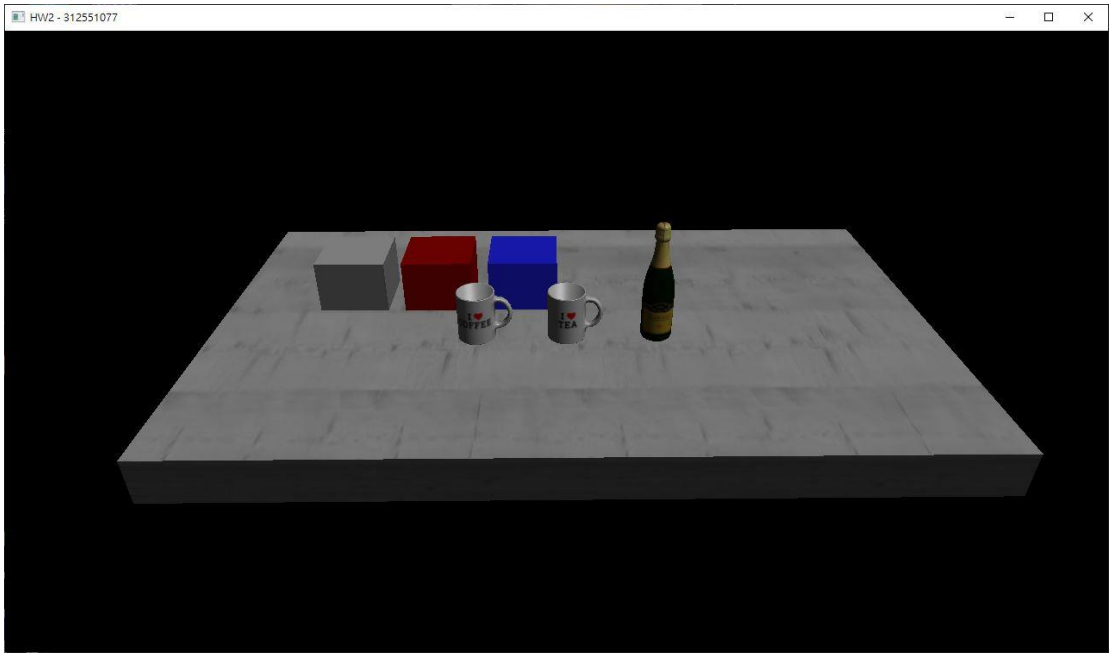
Results

The results are shown below:

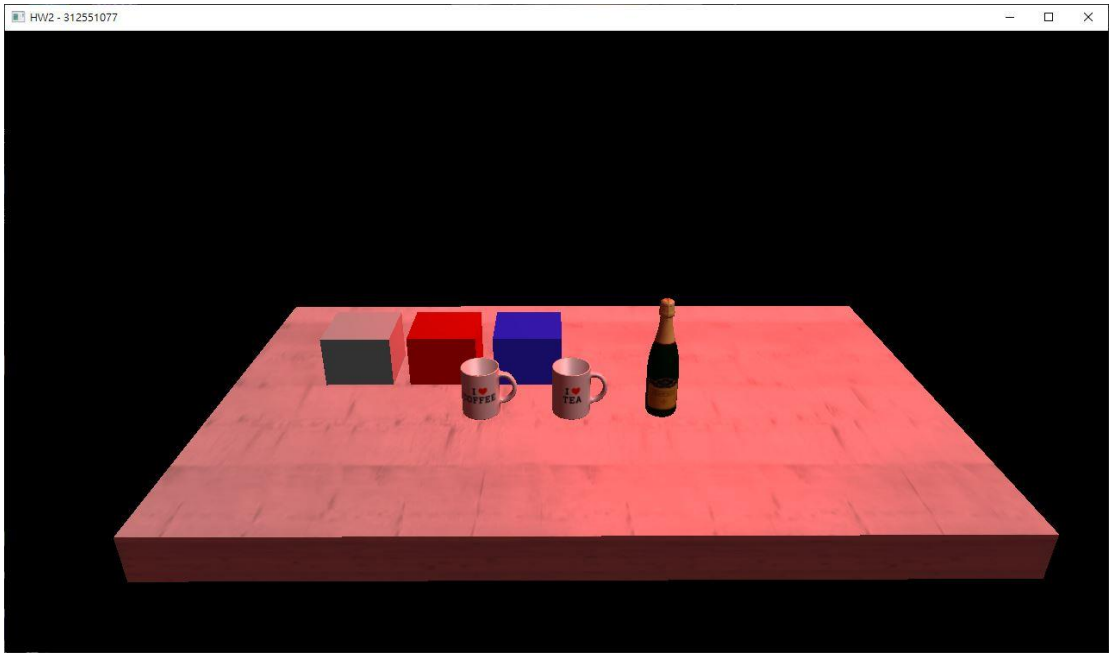
- Basic with no lights



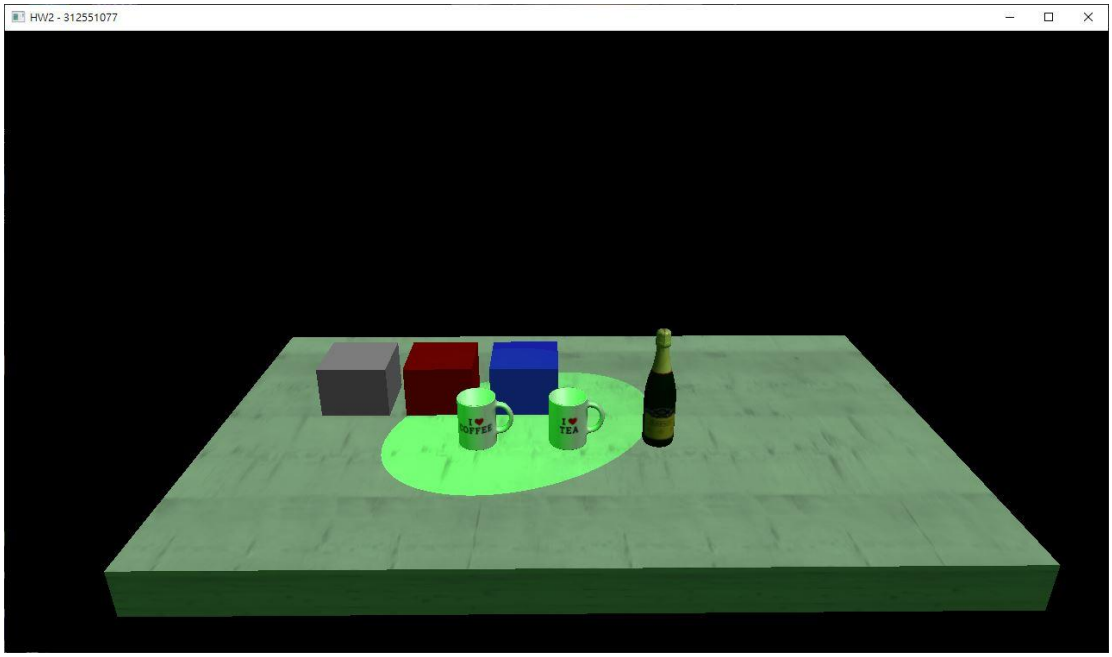
- Direction light



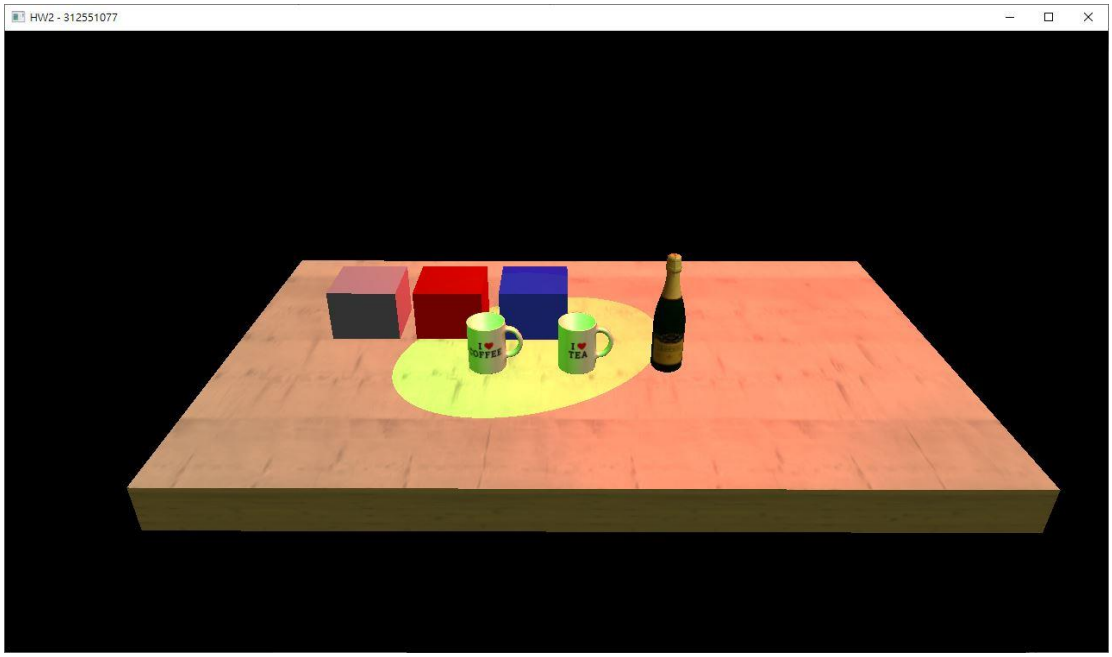
- Point light



- Spotlight



- Direction light with Point light with Spotlight



Problems I encountered

The first problem I encounter is to parse the object file, it took me some time to finish this feature. I added the data into the vector after reading the corresponding prefix, but after I realized it was wrong, I changed the way to add the data to the model. Then the problem was solved.

The second problem I encounter is to finish TODO#4, there are lots of details that must be take care of, and I sometimes make mistake implementing the formula.

Overall, I took some time to finish this assignment, the main reason is because I am not familiar with the functions in OpenGL, thus I don't understand what the structure of the parameters or the formula should be, although there are hints after the TODO description.