CG_HW7

实现思路

阴影映射实现

- 生成深度贴图来计算阴影

```
    // 为渲染的深度贴图创建帧缓冲对象

    GLuint depthMapFBO;
    glGenFramebuffers(1, &depthMapFBO);

    // 创建2D纹理，提供给帧缓冲的深度缓冲使用

    const GLuint SHADOW_WIDTH = 1024, SHADOW_HEIGHT = 1024;
    GLuint depthMap;
    glGenTextures(1, &depthMap);
    glBindTexture(GL_TEXTURE_2D, depthMap);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT,
        SHADOW_WIDTH, SHADOW_HEIGHT, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);

    // 把生成的深度纹理作为帧缓冲的深度缓冲

    glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
    glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,
depthMap, 0);
    glDrawBuffer(GL_NONE);
    glReadBuffer(GL_NONE);
    glBindFramebuffer(GL_FRAMEBUFFER, 0);

        // 渲染深度贴图
        glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
        glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
        glClear(GL_DEPTH_BUFFER_BIT);
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, texture);
```

- 光源空间变换

```
glm::mat4 lightProjection, lightView, lightSpaceMatrix;
float near_plane = 1.0f, far_plane = 7.5f;

if (flag) {
    lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
}
else {
    lightProjection = glm::perspective(glm::radians(45.0f), (GLfloat)SHADOW_WIDTH /
(GLfloat)SHADOW_HEIGHT, near_plane, far_plane);
}
lightView = glm::lookAt(lightPos, glm::vec3(0.0f), glm::vec3(0.0, 1.0, 0.0));
lightSpaceMatrix = lightProjection * lightView;
```

- 渲染至深度贴图

```
// 创建一个负责将顶点变换到光空间的着色器程序
// 顶点着色器
    const GLchar* LightSpaceVertexShaderSource = "#version 330 core\n"
        "layout (location = 0) in vec3 vertex_position;\n"

        "uniform mat4 lightSpaceMat;\n"
        "uniform mat4 model;\n"

        "void main() \n"
        "{\n"
            "gl_Position = lightSpaceMat * model * vec4(vertex_position, 1.0);\n"
        "}\n\0";
// 片段着色器（不工作）
    const GLchar* LightSpaceFragmentShaderSource = "#version 330 core\n"
        "void main()\n"
        "{\n"

        "}\n\0";

// 应用该着色器和前面计算的光空间矩阵渲染深度贴图

        glUseProgram(lightSpace_shader_programme);
        glUniformMatrix4fv(glGetUniformLocation(lightSpace_shader_programme,
"lightSpaceMat") , 1, GL_FALSE, glm::value_ptr(lightSpaceMatrix));

        // 渲染深度贴图
        glViewport(0, 0, SHADOW_WIDTH, SHADOW_HEIGHT);
        glBindFramebuffer(GL_FRAMEBUFFER, depthMapFBO);
        glClear(GL_DEPTH_BUFFER_BIT);
        glActiveTexture(GL_TEXTURE0);
        glBindTexture(GL_TEXTURE_2D, texture);

        // 地面深度贴图
        glm::mat4 model = glm::mat4(1.0f);
        glUniformMatrix4fv(glGetUniformLocation(lightSpace_shader_programme, "model"),
1, GL_FALSE, glm::value_ptr(model));
        glBindVertexArray(planeVAO);
        glDrawArrays(GL_TRIANGLES, 0, 6);

        // 方块深度贴图
        model = glm::scale(model, glm::vec3(0.2f));
        model = glm::rotate(model, float(glfwGetTime()), glm::vec3(0, 1.0, 0));
        glUniformMatrix4fv(glGetUniformLocation(lightSpace_shader_programme, "model"),
1, GL_FALSE, glm::value_ptr(model));
        glBindVertexArray(cubeVAO);
        glDrawArrays(GL_TRIANGLES, 0, 36);

        glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

- 渲染阴影
  - 顶点着色器的主要作用是将物体顶点渲染到光空间
  - 像素着色器主体参照教程用的是Blinn-Phong光照模型渲染场景，此外还需要添加阴影计算部分

```
float ShadowCalculation(vec4 fragPosLightSpace)
{
    // 执行透视除法
    vec3 projCoords = fragPosLightSpace.xyz / fragPosLightSpace.w;
    // 变换到[0,1]的范围
    projCoords = projCoords * 0.5 + 0.5;
    // 取得最近点的深度(使用[0,1]范围下的fragPosLight当坐标)
    float closestDepth = texture(shadowMap, projCoords.xy).r;
    // 取得当前片元在光源视角下的深度
    float currentDepth = projCoords.z;
    // 检查当前片元是否在阴影中
    float shadow = currentDepth > closestDepth  ? 1.0 : 0.0;

    return shadow;
}

// 结合阴影计算最后的光照颜色

"vec3 lighting = (ambient + (1.0 - shadow) * (diffuse + specular)) * color; \n"
```

阴影优化

- 阴影失真

  ```
  // 计算阴影偏离，使所有采样点都获得了比表面深度更小的深度值
  "float bias = max(0.05 * (1.0 - dot(normal, lightDir)), 0.005); \n"
  "shadow += currentDepth - bias > closestDepth ? 1.0 : 0.0; \n"
  ```

- 悬浮没有遇到，或者说肉眼看不出来
- 采样过多

  ```
  // 把深度贴图的纹理环绕选项设置为GL_CLAMP_TO_BORDER
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_BORDER);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_BORDER);
  GLfloat borderColor[] = { 1.0, 1.0, 1.0, 1.0 };
  glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);

  // 在着色器计算阴影时，只要投影向量的z坐标大于1.0，我们就把shadow的值强制设为0.0
  "if (projCoords.z > 1.0)shadow = 0.0; \n"
  ```

- 锯齿边

  ```
  // 应用pcf从纹理像素四周对深度贴图采样，然后把结果平均起来
  "vec2 texelSize = 1.0 / textureSize(shadowMap, 0); \n"
  "for (int x = -1; x <= 1; ++x)\n"
  "{\n"
      "for (int y = -1; y <= 1; ++y)\n"
      "{\n"
          "float pcfDepth = texture(shadowMap, projCoords.xy + vec2(x, y) * texelSize).r; \n"
          "shadow += currentDepth - bias > pcfDepth ? 1.0 : 0.0; \n"
      "}\n"
  "}\n"
  "shadow /= 9.0; \n"
  ```

  事实上，这个方法只解决了透视投影时的锯齿边，使用正交投影阴影仍然存在锯齿边。

正交，透视投影

- 正交投影

  ```
  // 深度贴图着色器中
  fragment_color = vec4(vec3(depthValue), 1.0);

  // 渲染时
  lightProjection = glm::ortho(-10.0f, 10.0f, -10.0f, 10.0f, near_plane, far_plane);
  ```

- 透视投影

  ```
  // 深度贴图着色器中
  "float z = depthValue * 2.0 - 1.0;\n"
  "float coord =  (2.0 * near_plane * far_plane) / (far_plane + near_plane - z *
  (far_plane - near_plane));\n"
  "fragment_color = vec4(vec3(coord / far_plane), 1.0);\n"

  // 渲染时
  lightProjection = glm::perspective(glm::radians(45.0f), (GLfloat)SHADOW_WIDTH /
  (GLfloat)SHADOW_HEIGHT, near_plane, far_plane);
  ```

实现效果见视频