计算机图形学报告

余婉莹 SA21168286

December 15, 2021

1 依赖包

- glad github 当前稳定版
- glfw github 当前稳定版
- opengl version 3.3
- glm version 0.9.8

2 实现功能

- 地形绘制：选择 puget_tex 模型进行绘制，并用给定的纹理图像进行纹理映射。
- 飞行模拟：采用飞行员视图在地形场景中漫游。
  - i) 左右键控制偏航 (Yaw)；
  - ii) 上下键控制俯仰 (Pitch)；
  - iii) a/d 键控制侧滚 (Roll)；
  - iv) 飞机位置的控制方面，可以按固定速度自动前行 (默认起始速度为 0)，并用 +/-调节速度；
- 物体绘制：在空中放置一个骆驼，物体的材料属性应可配置，通过键盘上的 U、J、I、K 按键进行调控。
- 光照：至少两个光源。i) 要实现平面明暗处理和平滑明暗处理两种方式。平面明暗处理，需为每个面计算一个法向；平滑明暗处理，需为每个顶点计算一个法向。ii) 对两个光源，一个是摄像头方向的手电筒光，一个是在地图四角的灯光。

3 代码解释

3.1 地形绘制

通过对文件数据的加载绘制在模型上，一下为加载的数据操作，其中利用到库文件 <stb_image.h> 帮助进行图形加载和处理。

```
unsigned int loadTexture(const char *path){
    unsigned int textureID;
    glGenTextures(1,\&textureID);
    int width,height,nrComponents;
    unsigned char* data = stbi\_load(path,\&width,\&height,\&
        nrComponents,0);
    if(data){
        GLenum format;
        if(nrComponents==1){
            format=GL_RED;
        }else if(nrComponents==3){
            format = GL_RGB;
        }else if(nrComponents==4){
            format = GL_RGBA;
        }
        glBindTexture(GL_TEXTURE_2D,textureID);
        glTexImage2D(GL_TEXTURE_2D,0,format,width,height,0,
            format,GL_UNSIGNED_BYTE,data);
        glGenerateMipmap(GL_TEXTURE_2D);

        glTexParameteri(GL_TEXTURE_2D,
            GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D,
            GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D,
            GL_TEXTURE_MIN_FILTER,
            GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,
            GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        stbi_image_free(data);
```

```
    }else{
        std::cout << "Texture failed to load at path: " << path << std
            ::endl;
        stbi_image_free(data);
    }
    return textureID;
}
```

3.2 飞行模拟

飞行模拟更多的是强调对于基于当前摄像头方向的把控，主要是由下式进行，侧滚则是在原来的基础上进行旋转，侧滚后的飞行方向基于当前的摄像头方向。

```
    void camera::updateCameraVectors() {
    glm::vec3 front;
    front.x = cos(glm::radians(Yaw)) * cos(glm::radians(Pitch));
    front.y = sin(glm::radians(Pitch));
    front.z = sin(glm::radians(Yaw)) * cos(glm::radians(Pitch));
    Front = glm::normalize(front);
    Right = glm::normalize(glm::cross(Front, WorldUp)); // normalize
        the vectors, because their length gets closer to 0 the more you
         look up or down which results in slower movement.
    Up    = glm::normalize(glm::cross(Right, Front));
    glm::mat4 roll_mat = glm::rotate(glm::mat4(1.0f), glm::radians(Roll)
        , Front);
    Up = glm::mat3(roll_mat) * Up;
    Right=glm::normalize(glm::cross(Front, Up));

}
```

3.3 物体绘制

- 读取着色器文件
- 对不同的物体采用不同的着色器，通过 uniform 设置光照影响，下式分别对灯光、物体、地图进行绘制和 uniform 赋值操作。

```
while (!glfwWindowShouldClose(window))
    {
        float currentFrame = glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        processInput(window);

        // render
        // ------
        glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT|
            GL_DEPTH_BUFFER_BIT);
        s.use();
        s.setVec3("viewPos", c.Position);
        s.setVec3("material.specular", 0.5f, 0.5f, 0.5f);
        s.setFloat("material.shininess", 32.0f);

        s.setVec3("dirLight.direction", -0.2f, -1.0f, -0.3f);
        s.setVec3("dirLight.ambient", 0.01f, 0.01f, 0.01f);
        s.setVec3("dirLight.diffuse", 0.2f, 0.2f, 0.2f);
        s.setVec3("dirLight.specular", 0.05f, 0.05f, 0.05f);

        //point light 1
        s.setVec3("pointLights[0].position", pointLightPositions[0]);
        s.setVec3("pointLights[0].ambient", 0.1f, 0.1f, 0.1f);
        s.setVec3("pointLights[0].diffuse", 0.8f, 0.8f, 0.8f);
        s.setVec3("pointLights[0].specular", 1.0f, 1.0f, 1.0f);
        s.setFloat("pointLights[0].constant", 1.0f);
        s.setFloat("pointLights[0].linear", 0.09);
        s.setFloat("pointLights[0].quadratic", 0.032);

        // point light 2
        s.setVec3("pointLights[1].position", pointLightPositions[1]);
        s.setVec3("pointLights[1].ambient", 0.1f, 0.1f, 0.1f);
        s.setVec3("pointLights[1].diffuse", 0.8f, 0.8f, 0.8f);
```

```
s.setVec3("pointLights[1].specular", 1.0f, 1.0f, 1.0f);
s.setFloat("pointLights[1].constant", 1.0f);
s.setFloat("pointLights[1].linear", 0.09);
s.setFloat("pointLights[1].quadratic", 0.032);

// point light 3
s.setVec3("pointLights[2].position", pointLightPositions[2]);
s.setVec3("pointLights[2].ambient", 0.1f, 0.1f, 0.1f);
s.setVec3("pointLights[2].diffuse", 0.8f, 0.8f, 0.8f);
s.setVec3("pointLights[2].specular", 1.0f, 1.0f, 1.0f);
s.setFloat("pointLights[2].constant", 1.0f);
s.setFloat("pointLights[2].linear", 0.09);
s.setFloat("pointLights[2].quadratic", 0.032);
// point light 4
s.setVec3("pointLights[3].position", pointLightPositions[3]);
s.setVec3("pointLights[3].ambient", 0.1f, 0.1f, 0.1f);
s.setVec3("pointLights[3].diffuse", 0.8f, 0.8f, 0.8f);
s.setVec3("pointLights[3].specular", 1.0f, 1.0f, 1.0f);
s.setFloat("pointLights[3].constant", 1.0f);
s.setFloat("pointLights[3].linear", 0.09);
s.setFloat("pointLights[3].quadratic", 0.032);
// spotLight
s.setVec3("spotLight.position", c.Position);
s.setVec3("spotLight.direction", c.Front);
s.setVec3("spotLight.ambient", 0.0f, 0.0f, 0.0f);
s.setVec3("spotLight.diffuse", 1.0f, 1.0f, 1.0f);
s.setVec3("spotLight.specular", 1.0f, 1.0f, 1.0f);
s.setFloat("spotLight.constant", 1.0f);
s.setFloat("spotLight.linear", 0.09);
s.setFloat("spotLight.quadratic", 0.032);
s.setFloat("spotLight.cutOff", glm::cos(glm::radians(12.5f)));
s.setFloat("spotLight.outerCutOff", glm::cos(glm::radians(15.0f)
    ));

glm::mat4 model          = glm::mat4(1.0f); // make sure to
    initialize matrix to identity matrix first
```

```
glm::mat4 view           = glm::lookAt(c.Position,c.Position+c.
    Front,c.Up);
glm::mat4 projection     = glm::perspective(glm::radians(c.Zoom)
    ,(float)SCR_WIDTH/SCR_HEIGHT,0.1f,100.0f);

s.setMat4("view",view);
s.setMat4("model",model);
s.setMat4("projection",projection);
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D,texture1 );
glBindVertexArray(VAO[0]);
glDrawElements(GL_TRIANGLES,o.S.size(),
    GL_UNSIGNED_INT, 0);

obj.use();
diffuseColor = lightColor    * glm::vec3(diffuse); // decrease
    the influence
ambientColor = diffuseColor * glm::vec3(ambient); // low
    influence
obj.setVec3("material.diffuse", diffuseColor);
obj.setVec3("material.ambient",ambientColor);
obj.setVec3("viewPos", c.Position);
obj.setFloat("material.shininess", 32.0f);

obj.setVec3("dirLight.direction", -0.2f, -1.0f, -0.3f);
obj.setVec3("dirLight.ambient", 0.01f, 0.01f, 0.01f);
obj.setVec3("dirLight.diffuse", 0.2f, 0.2f, 0.2f);
obj.setVec3("dirLight.specular", 0.05f, 0.05f, 0.05f);

//point light 1
obj.setVec3("pointLights[0].position", pointLightPositions[0]);
obj.setVec3("pointLights[0].ambient", 0.10f, 0.1f, 0.1f);
obj.setVec3("pointLights[0].diffuse", 0.8f, 0.8f, 0.8f);
obj.setVec3("pointLights[0].specular", 1.0f, 1.0f, 1.0f);
obj.setFloat("pointLights[0].constant", 1.0f);
obj.setFloat("pointLights[0].linear", 0.09);
```

```
obj.setFloat("pointLights[0].quadratic", 0.032);

// point light 2
obj.setVec3("pointLights[1].position", pointLightPositions[1]);
obj.setVec3("pointLights[1].ambient", 0.1f, 0.1f, 0.1f);
obj.setVec3("pointLights[1].diffuse", 0.8f, 0.8f, 0.8f);
obj.setVec3("pointLights[1].specular", 1.0f, 1.0f, 1.0f);
obj.setFloat("pointLights[1].constant", 1.0f);
obj.setFloat("pointLights[1].linear", 0.09);
obj.setFloat("pointLights[1].quadratic", 0.032);

// point light 3
obj.setVec3("pointLights[2].position", pointLightPositions[2]);
obj.setVec3("pointLights[2].ambient", 0.1f, 0.1f, 0.1f);
obj.setVec3("pointLights[2].diffuse", 0.8f, 0.8f, 0.8f);
obj.setVec3("pointLights[2].specular", 1.0f, 1.0f, 1.0f);
obj.setFloat("pointLights[2].constant", 1.0f);
obj.setFloat("pointLights[2].linear", 0.09);
obj.setFloat("pointLights[2].quadratic", 0.032);
// point light 4
obj.setVec3("pointLights[3].position", pointLightPositions[3]);
obj.setVec3("pointLights[3].ambient", 0.05f, 0.05f, 0.05f);
obj.setVec3("pointLights[3].diffuse", 0.8f, 0.8f, 0.8f);
obj.setVec3("pointLights[3].specular", 1.0f, 1.0f, 1.0f);
obj.setFloat("pointLights[3].constant", 1.0f);
obj.setFloat("pointLights[3].linear", 0.09);
obj.setFloat("pointLights[3].quadratic", 0.032);
// spotLight
obj.setVec3("spotLight.position", c.Position);
obj.setVec3("spotLight.direction", c.Front);
obj.setVec3("spotLight.ambient", 0.0f, 0.0f, 0.0f);
obj.setVec3("spotLight.diffuse", 1.0f, 1.0f, 1.0f);
obj.setVec3("spotLight.specular", 1.0f, 1.0f, 1.0f);
obj.setFloat("spotLight.constant", 1.0f);
obj.setFloat("spotLight.linear", 0.09);
obj.setFloat("spotLight.quadratic", 0.032);
```

```
        obj.setFloat("spotLight.cutOff", glm::cos(glm::radians(12.5f)));
        obj.setFloat("spotLight.outerCutOff", glm::cos(glm::radians(15.0
            f)));

        obj.setMat4("projection", projection);
        obj.setMat4("view", view);
        model = glm::translate(model, glm::vec3(1.0f,2.0f,1.0f));
        obj.setMat4("model", model);
        glBindVertexArray(VAO[1]);
        glDrawElements(GL_TRIANGLES,o1.S.size(),
            GL_UNSIGNED_INT,0);

        light.use();
        light.setMat4("projection",projection);
        light.setMat4("view",view);
        glBindVertexArray(VAO[2]);
        for(unsigned int i=0;i<4;i++){
            model = glm::mat4(1.0f);
            model = glm::translate(model,pointLightPositions[i]);
            model = glm::scale(model,glm::vec3(0.2f));
            light.setMat4("model",model);
            glDrawArrays(GL_TRIANGLES,0,36);
        }

        running();

        // glfw: swap buffers and poll IO events (keys pressed/released,
            mouse moved etc.)
        //
            ---------------------------------------------------------------

        glfwSwapBuffers(window);
        glfwPollEvents();
    }
```

3.4 材料配置

- 根据键盘上 UJIK 四个键分别调整物体的材质

```
if (glfwGetKey(window,GLFW_KEY_U)==GLFW_PRESS)
    if(ambient>0.0005)
        ambient-=0.0005;
if (glfwGetKey(window,GLFW_KEY_J)==GLFW_PRESS)
    if(ambient<1)
        ambient+=0.0005;
if (glfwGetKey(window,GLFW_KEY_I)==GLFW_PRESS)
    if( diffuse>0.0005)
        diffuse-=0.0005;
if (glfwGetKey(window,GLFW_KEY_K)==GLFW_PRESS)
    if( diffuse<1)
        diffuse+=0.0005;
```

3.5 光照效应

光照效应分别设计三种状态下的灯光，然后通过累加得到最终的光线效果。

```
#version 330 core
out vec4 FragColor;

struct Material {
    sampler2D diffuse;
        vec3 specular;
    float shininess;
};

struct DirLight{
    vec3 direction;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};//定向光
```

```
struct PointLight{
    vec3 position;

    float constant;
    float linear;
    float quadratic;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};

struct SpotLight{
    vec3 position;
    vec3 direction;
    float cutOff;
    float outerCutOff;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;

    float constant;
    float linear;
    float quadratic;
};



#define NR_POINT_LIGHTS 4//结构体数组

in vec2 TexCoords;
in vec3 Normal;
in vec3 FragPos;
```

```
uniform PointLight pointLights[NR_POINT_LIGHTS];
uniform DirLight dirLight;
uniform Material material;
uniform SpotLight spotLight;
uniform vec3 viewPos;

vec3 CalcDirLight(DirLight light,vec3 normal,vec3 viewDir);
vec3 CalcPointLight(PointLight light,vec3 normal,vec3 fragPos,vec3
    viewDir);
vec3 CalcSpotLight(SpotLight light,vec3 normal,vec3 fragPos,vec3
    viewDir);

void main()
{
    vec3 norm = normalize(Normal);
    vec3 viewDir = normalize(viewPos-FragPos);

    //第一阶段：定向光照
    vec3 result = CalcDirLight(dirLight,norm,viewDir);
    //第二阶段：点光源
    for(int i=0;i<NR_POINT_LIGHTS;i++)
    result += CalcPointLight(pointLights[i],norm,FragPos,viewDir);

    //聚光
    result+=CalcSpotLight(spotLight,norm,FragPos,viewDir);
    FragColor=vec4(result,1.0);
}
//定向光源
vec3 CalcDirLight(DirLight light,vec3 normal,vec3 viewDir){
    //normal:规范化后的顶点坐标
    //light:
    //观察视角：
    vec3 lightDir = normalize(-light.direction);
    float diff = max(dot(normal,lightDir),0.0);
    vec3 reflectDir = reflect(-lightDir,normal);
    float spec = pow(max(dot(viewDir,reflectDir),0.0),material.
```

```
        shininess);
    //环境光
    vec3 ambient = light.ambient * vec3(texture(material.diffuse,
        TexCoords));
    vec3 diffuse = light.diffuse * diff * vec3(texture(material.diffuse,
        TexCoords));
    vec3 specular = light.specular * spec * material.specular;
    return (ambient+diffuse+specular);

}

//点光源
vec3 CalcPointLight(PointLight light,vec3 normal,vec3 fragPos,vec3
    viewDir){
    vec3 lightDir = normalize(light.position-fragPos);//与实际方向相反
    //漫反射着色
    float diff = max(dot(normal,lightDir),0.0);
    //镜面光着色
    vec3 reflectDir = reflect(-lightDir,normal);
    float spec = pow(max(dot(viewDir,reflectDir),0.0),material.shininess
        );
    //衰减
    float distance = length(light.position-fragPos);
    float attenuation = 1.0/(light.constant+light.linear*distance+light.
        quadratic*(distance*distance));
    //合并结果
    vec3 ambient = light.ambient*vec3(texture(material.diffuse,
        TexCoords));
    vec3 diffuse = light.diffuse*diff*vec3(texture(material.diffuse,
        TexCoords));
    vec3 specular = light.specular*spec*material.specular;

    ambient*=attenuation;
    diffuse*=attenuation;
    specular*=attenuation;
    return (ambient+diffuse+specular);
```

```
}

vec3 CalcSpotLight(SpotLight light,vec3 normal,vec3 fragPos,vec3
    viewDir){
    //漫反射着色
    vec3 lightDir = normalize(light.position-fragPos);
    float diff = max(dot(normal,lightDir),0.0);
    //镜面光着色
    vec3 reflectDir = reflect(-lightDir,normal);
    float spec = pow(max(dot(viewDir,reflectDir),0.0),material.shininess
        );
    //边缘柔化
    float theta = dot(lightDir,normalize(-light.direction));
    float epsilon=(light.cutOff-light.outerCutOff);
    float intensity=clamp((theta-light.outerCutOff)/epsilon,0.0,1.0);

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

    //合并结果
    vec3 ambient = light.ambient*texture(material.diffuse,TexCoords).
        rgb;
    vec3 diffuse = light.diffuse*diff*texture(material.diffuse,
        TexCoords).rgb;
    vec3 specular = light.specular*spec*material.specular;

    diffuse*=intensity;
    specular*=intensity;

    ambient*=attenuation;
    diffuse*=attenuation;
    specular*=attenuation;
    return (ambient+diffuse+specular);
}
```

3.6 法向量计算

- 对于绘制的物体进行法向量计算是实现光照的最重要的一步，该计算公式和数据加载放置在一起，当加载 f 变量时，基于 f 对三面片顶点的选择，计算该三面片法向量。将其设置为该顶点的法向量。

```
glm::vec3 norm=glm::normalize(glm::cross(V[b-1].
    position-V[a-1].position,V[c-1].position-V[b-1].
    position));
V[a-1].normal=norm;
V[b-1].normal=norm;
V[c-1].normal=norm;
```

4 结果展示

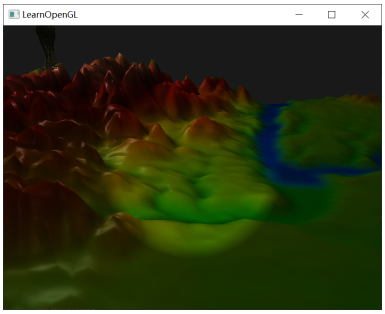

图 1: 实现模型

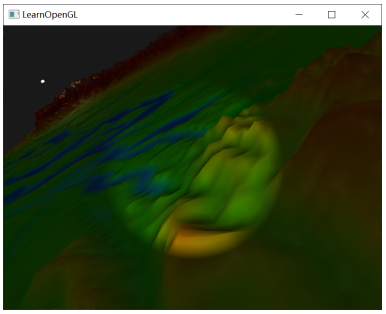

图 2: 侧滚后的效果