计算机图形学

（Computer Graphics）

实验题目：

目录：

实验报告：

1. 实验内容描述，即场景名称和特点

1.1 场景名称

《飞机上办公的树莓学生》

1.2 场景特点

场景内容为一个穿着西装的程序员（选择了数字媒体方向的）将自己的办公地点（有办公桌和办公椅）设置在了飞机上，然后使用自己的笔记本电脑在上面利用 blender 进行建模工作。

2. 实验功能算法描述，即多个模型是如何建立、如何载入、如何观察的

2.1 多个模型的建立

模型	建模工具
天空球、笔记本电脑及屏幕	Blender
椅子	Solidworks
桌子	3Dmax
飞机	3dmax
程序员	Blender

2.2 多个模型的载入

小组的成员的模型全部先在建模的时候完成纹理贴图和法线调整，然后将模型导出为 obj 文件（在 obj 文件中写入纹理坐标以及法线信息）。

在程序中，使用自己实现的 obj 载入器类的对象解析 obj 文件数据，然后，将不同的 obj 文件的数据存入不同的 VBO(Vertex Buffer Object).这样，就完成了模型数据的载入。

2.3 多个模型的渲染

使用一个 glProgram,传入相同的 World-View Projection 的矩阵、相机位置等所谓模型共用的数据，在每个模型渲染前，绑定对应的 VBO，将纹理采样器绑定对应的纹理单元，设置对应的数据解析结构，然后使用 glDrawArrays 函数进行传入渲染就可以了。

3. 实验 shader 程序描述，即 vertex shader 和 fragment shader 的程序代码及说明

3.1 Vetex shader

```
#version 330

layout(location = 0) in vec3 Position;  //传入的顶点坐标
layout(location = 1) in vec2 TexturePosition;    //传入的纹理坐标
layout(location = 2) in vec3 Normal; //传入的顶点法向量

//传入的变换到眼睛坐标系的变换矩阵
uniform mat4 gWVP;

//光照相关
uniform mat4 gWorld; //传入的变换到世界坐标系的变换矩阵
uniform vec3 gEye; //传入的眼睛的世界坐标系坐标

out vec2 TextureAfterTrans; //输出的纹理坐标
//out vec4 Color;

//光线相关
out vec3 NormalInWorld;
out vec3 PointPosition;
out vec3 EyePosition;

void main()
{
    //顶点位置输出
    gl_Position = gWVP * vec4(Position, 1.0);

    //光照相关
    PointPosition = (vec4(Position, 0.0)).xyz; //世界坐标系顶点坐标传递
    NormalInWorld = (gWorld*vec4(Normal, 0.0)).xyz;//世界坐标系法向量传递
    EyePosition = (vec4(gEye, 0.0)).xyz;//世界坐标系眼睛坐标传递

    TextureAfterTrans = TexturePosition;//纹理坐标系坐标传递
    //Color = vec4(clamp(Position, 0.0, 1.0), 1.0);
}
```

图 1 顶点着色器 vshader.glsl 代码

顶点着色器主要工作：

(a) 将 obj 文件中模型原始的世界坐标系下的坐标，使用 WVP 矩阵将其转换成眼睛（摄像机）坐标系下的坐标进行输出。

```
//顶点位置输出
gl_Position = gWVP * vec4(Position, 1.0);
```

(b) 将眼睛的世界坐标系坐标原封不动的传递给 fragment shader（光照计算时使用）

```
EyePosition = (vec4(gEye, 0.0)).xyz;//世界坐标系眼睛坐标传递
```

(c) 将当前处理的顶点的坐标传递给 fragment shader（光照计算时使用）

```
PointPosition = (vec4(Position, 0.0)).xyz; //世界坐标系顶点坐标传递
```

(d) 将法线坐标进行和模型一样的变换（平移、缩放、旋转）后传递给 fragment shader （光照计算时使用）

```
NormalInWorld = (gWorld*vec4(Normal, 0.0)).xyz;//世界坐标系法向量传递
```

(e) 将纹理坐标原封不动的传递给 fragmentshader（纹理映射时使用）

```
TextureAfterTrans = TexturePosition;//纹理坐标系坐标传递
```

3.2 Fragment shader

```
#version 330

//in vec4 Color;

in vec2 TextureAfterTrans;  //传入的经过插值的纹理坐标

//光照相关
in vec3 PointPosition; //传入的世界坐标系下物体的坐标
in vec3 NormalInWorld; //传入的世界坐标系下物体的法线坐标
in vec3 EyePosition; //传入的世界坐标系下眼睛的坐标

out vec4 FragColor;

uniform sampler2D gSampler; //传入的采样器全局变量

//平行光源
struct DirectionalLight { ... };

//点光源
struct PointLight { ... };

//漫射光最后的效果计算(光源是点光源)
vec4 CaculateDiffuseColor_pointLight(PointLight light, vec3 PointPosition, vec3 NormalInWorld) { ... }

//漫射光最后的效果计算(光源是平行光源)
vec4 CaculateDiffuseColor_directionLight(DirectionalLight light, vec3 PointPosition, vec3 NormalInWorld) { ... }

//镜面光最后的影响计算(光源是点光源)
vec4 CaculateSpecularColor_pointLight(PointLight light, vec3 PointPosition, vec3 NormalInWorld,vec3 EyePosition ,float SpecularPower) { ... }

//镜面光最后的影响计算(光源是平行光源)
vec4 CaculateSpecularColor_directionLight(DirectionalLight light, vec3 PointPosition, vec3 NormalInWorld, vec3 EyePosition, float SpecularPower) { ... }
```

图 2 片元着色器 fshader.glsl 代码（1）

```glsl
void main()
{

    //白色点光源1
    { ... }

    //白色点光源2
    { ... }

    //白色点光源3
    { ... }

    vec4 MatarialColor = vec4(1.0,1.0,1.0,1.0);//定义材质颜色

    //计算环境光光强
    vec4 AmbientColor = vec4(WhitePointLight1.Color * WhitePointLight1.AmbientIntensity, 1.0f)+
                        vec4(WhitePointLight3.Color * WhitePointLight3.AmbientIntensity, 1.0f)+
                        vec4(WhitePointLight2.Color * WhitePointLight2.AmbientIntensity, 1.0f);

    //计算漫射光强
    vec4 DiffuseColor1 = CaculateDiffuseColor_pointLight(WhitePointLight1, PointPosition, NormalInWorld);
    vec4 DiffuseColor2 = CaculateDiffuseColor_pointLight(WhitePointLight2, PointPosition, NormalInWorld);
    vec4 DiffuseColor3 = CaculateDiffuseColor_pointLight(WhitePointLight3, PointPosition, NormalInWorld);

    //计算镜面反射光
    vec4 SpecularColor1 = CaculateSpecularColor_pointLight(WhitePointLight1, PointPosition, NormalInWorld,EyePosition,1.0);
    vec4 SpecularColor2 = CaculateSpecularColor_pointLight(WhitePointLight2, PointPosition, NormalInWorld,EyePosition,1.0);
    vec4 SpecularColor3 = CaculateSpecularColor_pointLight(WhitePointLight3, PointPosition, NormalInWorld,EyePosition,1.0);

    // ...

    FragColor = texture2D(gSampler, TextureAfterTrans.st) * MatarialColor *(
            AmbientColor +
            DiffuseColor1 + DiffuseColor2 + DiffuseColor3 +
            SpecularColor1 + SpecularColor2 + SpecularColor3);

    //FragColor = texture2D(gSampler, TextureAfterTrans.st);

}
```

图 3 片元着色器 fshader.glsl 代码（2）

片元着色器主要工作：

(a) 定义光源结构体

```glsl
//平行光源
struct DirectionalLight
{
    vec3 Color;//光源颜色
    float AmbientIntensity; //环境光系数
    float DiffuseIntensity;//漫射光光强
    float SpecularIntensity;//镜面反射光光强
    vec3 Direction;//光照方向
};

//点光源
struct PointLight
{
    vec3 Color;
    float DiffuseIntensity; //漫反射系数
    float AmbientIntensity; //环境光系数
    float SpecularIntensity; //反射光系数
    float Attenuation_constant; //常数衰减系数
    float Attenuation_liner; //线性衰减系数
    float Attenuation_exp;  //指数衰减系数
    vec3 position; //点光源位置
};
```

(b) 实现光照计算函数

```
//漫射光最后的效果计算(光源是点光源)
vec4 CaculateDiffuseColor_pointLight(PointLight light, vec3 PointPosition, vec3 NormalInWorld){ ... }

//漫射光最后的效果计算(光源是平行光源)
vec4 CaculateDiffuseColor_directionLight(DirectionalLight light, vec3 PointPosition, vec3 NormalInWorld){ ... }

//镜面光最后的影响计算(光源是点光源)
vec4 CaculateSpecularColor_pointLight(PointLight light, vec3 PointPosition, vec3 NormalInWorld,vec3 EyePosition ,float SpecularPower){ ... }

//镜面光最后的影响计算(光源是平行光源)
vec4 CaculateSpecularColor_directionLight(DirectionalLight light, vec3 PointPosition, vec3 NormalInWorld, vec3 EyePosition, float SpecularPower){ ... }
```

(c) 定义光源

```
//白色点光源1
{
    PointLight WhitePointLight1;
    WhitePointLight1.Color = vec3(1, 1, 1);
    WhitePointLight1.AmbientIntensity = 0.3;
    WhitePointLight1.DiffuseIntensity = 0.8;
    WhitePointLight1.SpecularIntensity = 0.8;
    WhitePointLight1.Attenuation_constant = 0.2;
    WhitePointLight1.Attenuation_liner = 0.1;
    WhitePointLight1.Attenuation_exp = 0.1;
    WhitePointLight1.position = vec3(0.0, 9.0, 0.0);
}

//白色点光源2
{ ... }

//白色点光源3
{ ... }
```

(d) 计算最后的光源因子

```
//计算环境光光强
vec4 AmbientColor = vec4(WhitePointLight1.Color * WhitePointLight1.AmbientIntensity, 1.0f)+
                    vec4(WhitePointLight3.Color * WhitePointLight3.AmbientIntensity, 1.0f)+
                    vec4(WhitePointLight2.Color * WhitePointLight2.AmbientIntensity, 1.0f);

//计算漫射光强
vec4 DiffuseColor1 = CaculateDiffuseColor_pointLight(WhitePointLight1, PointPosition, NormalInWorld);
vec4 DiffuseColor2 = CaculateDiffuseColor_pointLight(WhitePointLight2, PointPosition, NormalInWorld);
vec4 DiffuseColor3 = CaculateDiffuseColor_pointLight(WhitePointLight3, PointPosition, NormalInWorld);

//计算镜面反射光
vec4 SpecularColor1 = CaculateSpecularColor_pointLight(WhitePointLight1, PointPosition, NormalInWorld,EyePosition,1.0);
vec4 SpecularColor2 = CaculateSpecularColor_pointLight(WhitePointLight2, PointPosition, NormalInWorld,EyePosition,1.0);
vec4 SpecularColor3 = CaculateSpecularColor_pointLight(WhitePointLight3, PointPosition, NormalInWorld,EyePosition,1.0);
```

(e) 使用纹理取样器取样并把最后的光照因子加上着色

```
FragColor = texture2D(gSampler, TextureAfterTrans.st) * MatarialColor *(
        AmbientColor +
        DiffuseColor1 + DiffuseColor2 + DiffuseColor3 +
        SpecularColor1 + SpecularColor2 + SpecularColor3);
```

4. 其他功能描述，如交互、光照、纹理、类定义、基础库功能等

4.1 纹理

每一个小组成员都再自己的 obj 文件中写入了纹理坐标，然后使用 SOIL(Simple Opengl Image Library)将图片读入，然后将图片数据绑定到为每一

个模型分配的纹理单元(Texture Unit)上。最后在导入每一个模型之前，将纹理着色器绑定到对应的纹理单元上就可以了。

4.2 光照

因为整体的模型背景是在天空中，因此，只在太阳的方向放置了几个点光源来模拟日光效果，但是为了更好的效果，将整体的环境光调亮了，因此日光的漫射、镜面效果不是特别明显。

4.3 交互

定义了一个 bool 值数组 ifShow，分别对应每个模型

```
bool ifShow[8];
```

实现对应回调回调，键盘上的对应字母键会将对应模型的 bool 值取反。

```
static void KeyboardCB(unsigned char Key, int x, int y)
{
    switch (Key) {
        case 'q':
        {
            exit(0);
            break;
        }
        case 'a':
        {
            if (ifShow[0])
            {
                ifShow[0] = false;
            }
            else
            {
                ifShow[0] = true;
            }
            break;
        }
        case 's':
        {
            if (ifShow[1])
            {
                ifShow[1] = false;
            }
            else
            {
                ifShow[1] = true;
            }
            break;
        }
        case 'd':
        {
            if (ifShow[2])
            {
                ifShow[2] = false;
```

注册对应键盘普通区域的监听函数

```
glutKeyboardFunc(KeyboardCB);    //键盘常规按键（字母键和数字键）
```

最后在渲染时根据 ifShow 数组中的对应 bool 值来决定是否渲染对应模型，以实现使用键盘隐藏模型的功能。

5. 实验结果，要贴实验结果图

图 4 结果(1)

图 5 结果(2)

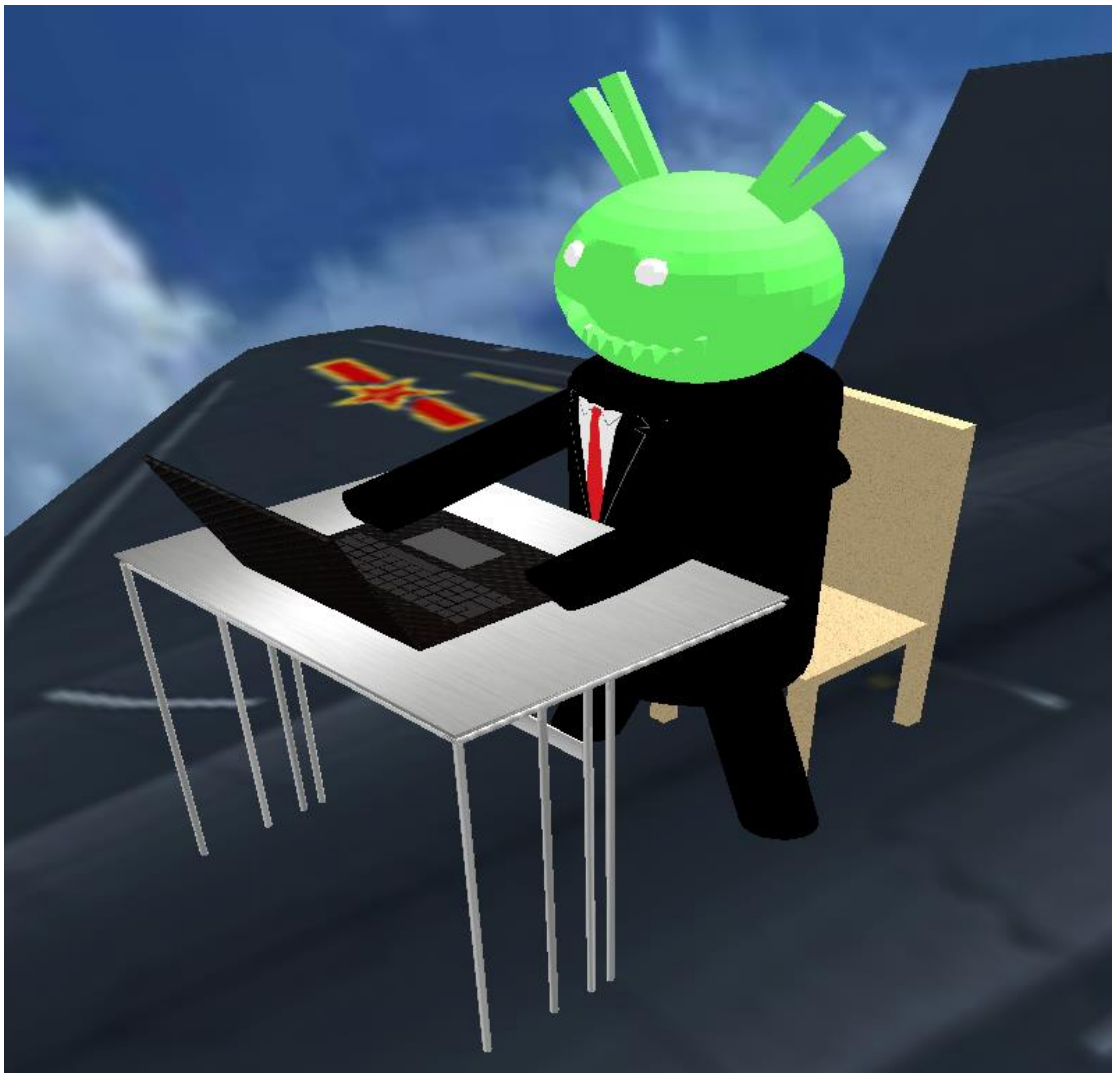

图 6 结果(3)

图 7 结果（4）

6. 小组成员任务分工

姓名	工作
陈纯华	桌子建模贴图
陈俊伟	天空球、笔记本电脑建模贴图，整合模型
黎相鑫	椅子建模贴图
王谦	程序员建模贴图
周俊	飞机建模贴图