

图形学大程实验报告

课程名称：____计算机图形学____ 指导老师：____唐敏____ 成绩：____

同组学生姓名：陈丹彤、刘昊澄、罗昱哲、周阳

(1)总体要求

简单室内场景三维建模及真实感绘制（总分 40 分）（以 4 人（或少于 4 人）为一组实现，教师以给平均分 的形式打分，如果组内不平均分配，则需小组提供组内所有同学都同意的分配方案，报助教或老师同意）

(2)基本要求的实现：

使用工具

Assimp 库
glm 库
glfw 库
irrKlang 音乐播放库

1. 基本三维网格导入导出

a) OBJ 格式

OBJ 文件是一种 3D 模型文件，以文本文件形式、单纯的字典结构进行。以一个简单的 OBJ 格式为例，它包含顶点数据：顶点坐标 v ，顶点法向量 vn ，纹理坐标 vt 。这些关键字出现在每一行的开头，可以说明这一行是什么样的数据。在读入顶点数据时，每一行数据都自动获得一个索引号，即该行在该类数据中的序号。在之后的一行元素：面 f 中就可以通过指定三角形三个顶点的各类数据的索引，绘制出一个面。最终，当所有的面都被绘制出来后，模型也就成功地显示出来了。

b) OBJ 的读入和导出

在导入方面，为了较方便地使用网络上的免费模型构建我们的场景，我们选择了使用 Assimp (Open Asset Import Library) 这一模型加载库来导入模型。Assimp 导入一个模型时，会将整个模型加载进一个场景 Scene 对象。场景 Scene 下的 $mMeshes$ 数组存储了需要用到的数据：网格 Mesh。一个 Mesh 对象包含了其顶点数据、面以及模型的材质。

为了方便我们的绘制，我们需要遍历场景 Scene，对 Scene 中每一个 Mesh 对象进行处理，获取顶点数据、索引以及材质属性存储在我们自己建立的 Mesh 类里，再建立 Model 类用于存储一整个模型。而为了更方便地存储顶点数据，我们也建立了 Vertex 结构进行存储以 glm 提供的向量格式存储顶点坐标、顶点法线与纹理坐标。每一个 Mesh 对象都存有

Vertex 结构的容器，根据面提供的索引可以很方便地进行模型绘制。

在导出方面，由于我们的最终成果是写一个 MiniGame 与场景漫游，功能上不需要实现导出。因而仅仅在 Model 类中实现了一个简单的 Output 函数，导出一个包含其顶点数据、面的 Obj 文件。

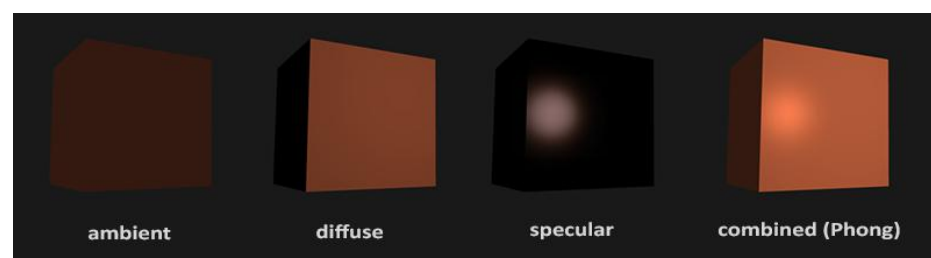
2. 基本材质纹理显示编辑

OBJ 文件不包含材质特性、贴图路径等信息，在实际使用时往往搭配 MTL 文件使用，由 OBJ 文件确定模型，由 MTL 文件确定材质与贴图。MAYA 等 3D 建模软件在导出 OBJ 模型时都会根据材质与贴图自动生成 MTL 文件，OBJ 文件也会自动引用 MTL 材质库。

在先前提到的 Assimp 导入模型过程中，其引用的 MTL 材质库（在同目录下）也会被一并加载至场景对象 Scene 中的 mMaterial 数组中的 aiMaterial 对象。我们在获取 Mesh 信息时一并也将材质信息处理了。aiMaterial 对象内部对每个存有纹理位置数组，通过其成员函数 GetTexture 即可获得文件位置加载贴图，最终存储于我们自己建立的 Texture 结构中。Texture 结构记录了纹理的 id 以及纹理的类型，值得一提的是结构还记录了纹理的路径，读取纹理时可以对路径避免重复读取。每一个 Mesh 对象存有一个 Texture 结构的容器，在绘制时可以很方便地激活、绑定、绘制这些贴图。

3.基本光照模型要求，并实现基本的光源编辑（如调整光源的位置，光强等参数）；

场景中光照模型是以冯氏光照模型为基础，在模型加载的片段着色器中对顶点颜色的计算包括了环境光、漫反射和镜面光照的叠加。



1.环境光照(Ambient Lighting): 即使在黑暗的情况下，世界上通常也仍然有一些光亮（月亮、远处的光），所以物体几乎永远不会是完全黑暗的。为了模拟这个，我们会使用一个环境光照明量，它永远会给物体一些颜色。环境光照的在场景中的计算很简单，片段着色器从程序中接收光源的颜色，并乘上一个很小的常量环境因子和物体的颜色就得到了最终的环境光分量。

2.漫反射光照(Diffuse Lighting): 模拟光源对物体的方向性影响。它是冯氏光照模型中视觉上最显著的分量。物体的某一部分越是正对着光源，它就会越亮。漫反射分量利用顶点的法向量和光源的位置向量进行点乘角度计算，结果值乘以光的颜色，两个向量之间的角度越大，漫反射分量就会越小。但是为了避免点乘角度大于 90 度时返回负数计算结果需要与 0 值判断大值输出。

3. 镜面光照(Specular Lighting): 模拟有光泽物体上面出现的亮点。镜面光照的颜色相比于物体的颜色会更倾向于光的颜色。镜面光照也通过反射法向量周围光的方向来计算反射向量, 然后计算反射向量和视线方向的角度差, 夹角越小则镜面光分量越强。

在我们的 project 中室内场景的基本光照模型实现的是平行光和点光源的叠加。在模型加载的片段着色器中定义平行光和点光源两个结构体, 并定义相应的计算函数。在点光源的计算中实现随着光线传播距离的增长逐渐削减光的强度的衰减效果时运用非线性衰减公式:

$$F_{att} = \frac{1.0}{K_c + K_l * d + K_q * d^2}$$

并对常数项、一次项和二次项的参数进行配置调整

点光源照射效果。同时对光照增加可键盘调整的强度变量 light_str 乘以光源分量进行光照强度的自定义编辑。最终室内基本光照模型编辑的效果截图如下:

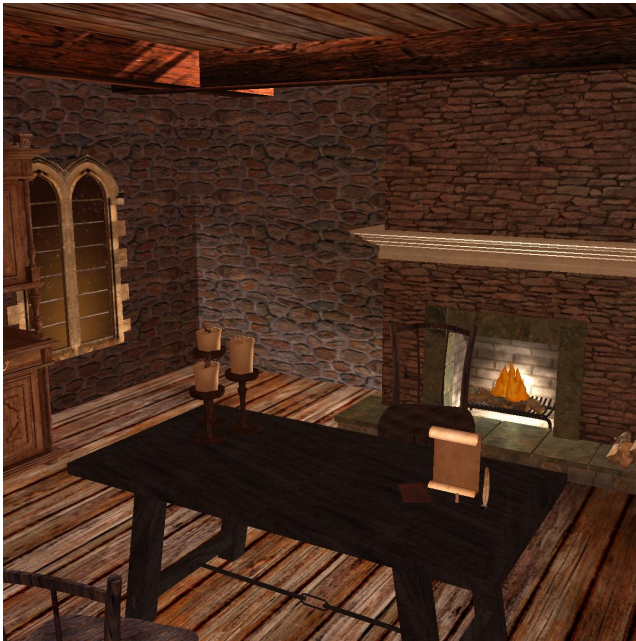
默认光照条件: (光源为壁炉中火焰后的黄色方块, 可以移动编辑观察)



点光源强度调小, 仅保留背侧定向光:



点光源强度调大：



光源位置移动实现场景漫游照明：





3. 场景漫游观察功能

a) Camera

在我们的第一人称漫游/游戏中，我们建立了一个 camera 类，运用 glm 库提供的向量记录相机的位置（Position）、方向（Front/Up/Right）、欧拉角（Yaw/Pitch）、视野（Zoom）；为绘制提供接口获取观察矩阵；并实现相机漫游观察功能的位置角度变换。使用 glm 库的好处是，在具体的功能实现时一些复杂的向量、矩阵运算都可以通过 glm 库的函数轻松完成。例如观察矩阵的计算，只需要通过 glm::LookAt 函数就可以创建，继而传递至 main 函数中进行绘制。

b) 自由移动

这一功能的实现主要可以两个部分：一是处理键盘键入控制的摄像机位置移动，二是处理鼠标移动控制的摄像机方向移动。

通过 GLFW 库的 glfwGetKey 函数，我们可以很方便地检测键盘键入，并将相应的按键参数传入 camera 类进行处理。以按下 W 键为例：main 调用 camera 类中的 ProcessKeyboard 函数，该成员函数根据按下的按键 W 更新相机位置的值 $\text{Position} += \text{velocity} * \text{Front}$ ，相机的位置就发生了改变。需要注意的是，这里的移动速度会考虑到 main 函数中传入的帧渲染时间，从而保证在不同帧渲染时间不同的情况下，移动的速度保持平衡。

另一方面，GLFW 库中的 glfwSetCursorPosCallback 会在鼠标移动后调用，进而可以调用 camera 的成员函数并传入鼠标移动的偏移量进行视角的更新。鼠标在屏幕 x 轴方向的偏移对应 Yaw 值的变化，y 轴的偏移则对应 Pitch 值的变化。由于相机的欧拉角和方向实际上代表的是同一个空间几何意义：相机的姿态。所以在改变欧拉角后一定要及时对方向进行更

新。同时，为了避免第一人称视角下相机的奇怪显示，Pitch 值不能超过 89 度（看向后脑勺后的天空）或是小于 -89 度（看向身后的地面）。

c) 缩放

在 main 函数中的 glm::perspective 构建投影矩阵时，会需要一个参数定义我们的视野。这个参数的数值就储存在我们的 camera 类的 Zoom 中。默认值为 45.0f（角度），值域设定为 [1.0f, 45.0f]。通过改变视野大小即可实现场景缩小放大的效果。类似的，glfwSetScrollCallback 函数会在使用鼠标滚轮后调用，进而调用 camera 类的成员函数。这样我们就可以使用鼠标的滚轮输入来控制 Zoom 值得大小变化了。

d) 轨道（Orbit）

为了实现这一功能，首先要确定：轨道旋转的中心 centre、轨道旋转的轴 axis、旋转的角度 rad。我们最终选择了：相机的位置选择相机局部坐标系的 y 轴进行旋转，旋转的角度由鼠标移动输入控制，相机的姿态始终保持指向旋转前的中心点。这样最终可以获得一个视觉效果较好的旋转观察效果。

在具体的代码中，首先存储当前 camera 指向的中心点，根据 camera 类的方向数据配合 glm 的向量运算函数构建局部坐标系 basis 矩阵，再创建变换矩阵 trans，二者进行矩阵运算后获取旋转后的新相机方向，再根据中心点、新的相机方向获得新的相机位置，最终完成轨道旋转。

4. 能够提供动画播放功能（多帧数据连续绘制）多帧连续绘制指读取连续的 obj 文件（或其他格式的模型文件），对其网格进行多次的绘制；

通过 vector 将连续的 obj 文件读入，读取时只要按序号连续触发 Draw 函数即可。通过 Sleep 函数控制绘制间隔时间，从而达到连续形成动画的效果。



图 1 以开门的连续动画为例







图 2 蜡烛摇曳动画

5.能够提供屏幕截取/保存功能

思路如下：

- 创建文件指针
- 读取现有 bmp 图片
- 用当前时间命名新图片
- 将文件头和信息头复制，并修改宽高数据
- 读取当前画板上的像素数据
- 写入像素数据
- 释放内存关闭文件

比较巧妙的一点在于将现有 bmp 文件信息头直接复制，避免自行输入而出错的几率。通过时间命名图片可以保留多张截图。

 Jun 19 223438 2020.bmp	2020/6/19 22:34	BMP 文件	4,219 KB
 Jun 19 223852 2020.bmp	2020/6/19 22:38	BMP 文件	4,219 KB
 Jun 19 223916 2020.bmp	2020/6/19 22:39	BMP 文件	4,219 KB
 Jun 19 223939 2020.bmp	2020/6/19 22:39	BMP 文件	4,219 KB
 Jun 19 223945 2020.bmp	2020/6/19 22:39	BMP 文件	4,219 KB
 Jun 19 224113 2020.bmp	2020/6/19 22:41	BMP 文件	4,219 KB

(3)额外要求的实现：

1.漫游时可实时碰撞检测：

本次 cg 程序中我们组采用基于射线与平面求交的方法实现碰撞检测。

原理如下：

我们将射线定义为：射线上的点=射线的远点+t*射线的方向，其中 t 用来描述它距离原点的距离，它的取值范围是[0，无穷远]。

我们将平面描述为 $X_n \cdot X = d$ ，其中 X_n 是平面的法线， X 是平面上的一点， d 是平面到原点的距离，dot 表示向量点积。

我们将射线和平面的方程联立：

$$\text{PointOnRay} = \text{Raystart} + t * \text{Raydirection}$$

$$X_n \cdot X = d$$

如果它们相交，则我们可以解出 t，

$$t = (d - X_n \cdot \text{Raystart}) / (X_n \cdot \text{Raydirection})$$

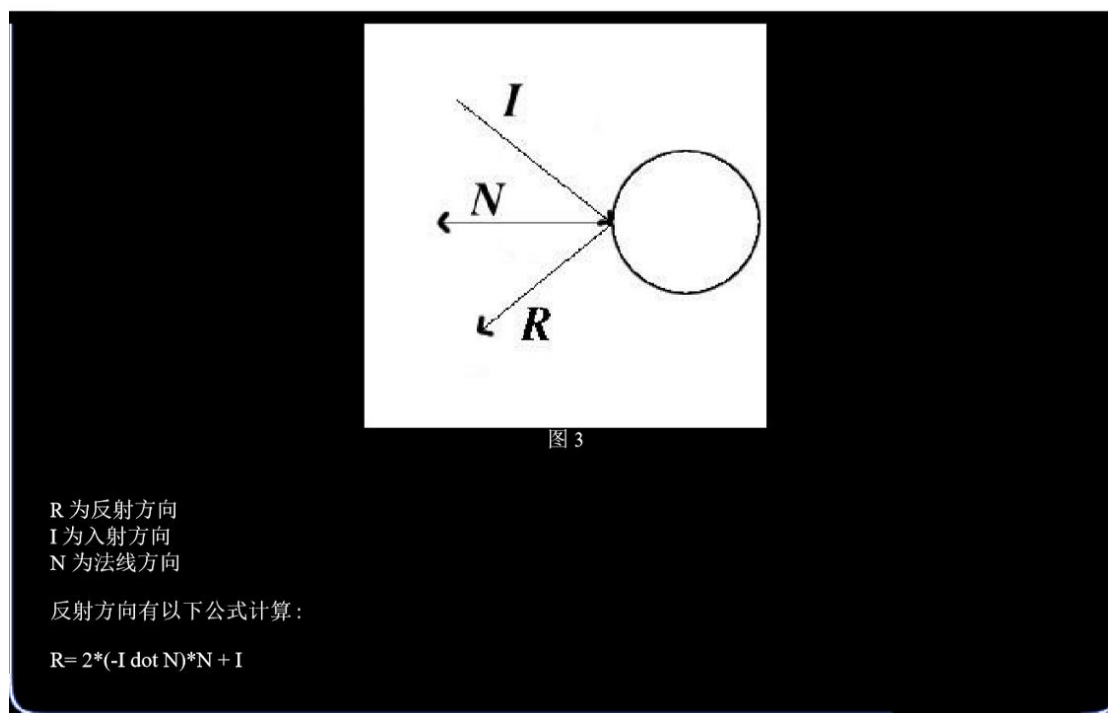
T 代表原点到与平面相交点的参数，把 t 代回原方程我们会得到与平面的碰撞点，需要注意的是如果分母为零，则代表射线与平面平行，如果 t 为负值，则说明交点在射线的相反方向，也不会发生碰撞。

在这里提供我们做碰撞检测的函数接口：

```
int TestIntersionPlane(const Plane& plane, const glm::vec3& position, const glm::vec3& direction, double& lamda, glm::vec3& pNormal)//五个参数分别是平面的结构，当前位置，运动方向，碰撞距离，碰撞法线
```

其中我们需要的是后两个参数，在项目中我们处理碰撞的方式有两种，一种是 camera 在与墙面检测到碰撞后就无法继续沿着当前方向移动；另一种情况是球体在与墙面发生碰撞的时候会发生反弹。

处理球体反弹时，我们需要计算其反射方向，在这里给出示意图和公式：



通过公式计算我们便可以获得球体在下一帧的运动方向，而在处理球的速度时，我们采取的手段是水平方向速度不变，垂直方向速度反向。

另外，为了实现门的对象表达能力，我们对门做了碰撞检测，在门闭合的情况下我们是无法通过门的，只有当我们通过按键打开门之后，我们才能通过门进入房间内部。

2. 三维游戏

基于我们先前所建的三维场景，我们构想出了一个中世纪背景下的探索收集游戏。玩家需要收集房屋内的所有金币取得游戏的胜利。在敌人（幽灵）的设定上我们借鉴了经典的超级马里奥中的布布鬼的游戏机制：幽灵始终面朝玩家；当玩家面向幽灵的时候幽灵不会移动，但当玩家背对幽灵时幽灵就会朝着玩家移动；一旦幽灵接触到了玩家，游戏就宣告失败。

游戏的两个主要元素金币与幽灵，我们都采用了面向对象的程序设计。金币类是动画类的子类，幽灵类则是模型类的子类。二者都是 camera 类的友元类。游戏功能的实现基本封装于对应的类内，并为 main 提供相应接口。

在金币的收集上我们采用了较为简单的实现方式：每当玩家按下交互键（空格）时，遍历所有的金币对象调用成员函数 Update，通过金币自身位置与 camera 类的位置判断玩家是否处于收集范围内，若是，则播放对应金币收集的动画，播放完毕后金币消失，main 函数中记录金币收集数量的 winCount++，收集完毕后 main 函数进行胜利画面的绘制。



图 1 金币

在幽灵类中，我们设定了幽灵的位置、欧拉角用于绘制幽灵时使用。每一帧绘制前 main 函数都会调用幽灵类的 Update 函数：幽灵始终面朝玩家方向通过 camera 的位置向量减去幽灵的位置向量得到，并通过公式转化为欧拉角供绘制使用；玩家与幽灵的面对背对关系通过 camera 的方向与幽灵的方向的点积正负号判断，若为背对关系则更新幽灵的位置；当二者距离太近时返回对应参数给 main 函数，main 函数继而进行失败画面的绘制。



图 2 幽灵

总体来说，游戏在具体的功能实现其实并不难，配合上风格化的模型场景、优秀的渲染画质、恰到好处的背景音乐、合适的玩法机制与数值设定，整个游戏可以说得上是精而小的。

3.光照模型细化：

在光照模型的细化方面我们实现的是室外场景光源实时阴影的编辑，我们采用的是阴影贴图的实现方法。以光的位置为视角透视渲染场景的深度贴图，计算光空间变换矩阵，将每个世界空间坐标变换到光源处所见到的那个空间。在渲染至深度贴图时采用的为 simpleDepthShader 顶点与片段着色器，仅仅将顶点变换到光空间。新建一个帧缓冲对象，把生成的 2D 深度纹理作为帧缓冲的深度缓冲。

在生成深度贴图后在正式模型绘制时，在顶点着色器中进行光空间的变换，片段着色器中根据深度贴图判断顶点是否在阴影中进行阴影的绘制。之后在对阴影贴图进行进一步细化，首先对深度贴图应用修正后的偏移量解决阴影失真的问题，然后通过从纹理四周对深度贴图采样进行简单的 PCF 实现降低阴影的锯齿块，提升阴影的真实感。在实时阴影的测试中，发现对面数较低的模型表现良好，但是由于我们 project 中场景的物体大多面数破万，深度贴图的计算效果不是特别理想，有待进一步改进，目前实时阴影贴图的绘制效果如下，通过按键可以对产生阴影的光线方向进行调整：





4.具有一定的对象表达能力，能够表达门、窗、墙等要求具有实际功能上的不同，如 PUBG 的模型中：关闭的门不可通行，可以按键打开，打开后可通行；窗可打碎、翻越；墙始终不可通行等

游戏模式中，关闭的门无法穿过，按空格键可以开门。



图 3 门关闭



图 4 门开启

在室内无法通过关闭的窗或墙穿出。



图 5 窗

图 6 靠近窗也无法穿过

(4)分工安排:

陈丹彤：动画播放、屏幕截取/保存功能

刘昊澄：obj 导入导出功能、场景漫游、基本材质纹理的显示和编辑、游戏功能整合

罗昱哲：基本光照模型以及光照细化

周阳：碰撞检测：