Computer Graphics

DOcraft

祁　烨　3120000386

徐子诚　3120000073

诸　祺　3130000037

任弘毅　3120000242

2014-12-10

目录

1 简介

1.1 目的

通过该软件，展现浙大宿舍的基本结构与普遍风貌，给未曾造访的人身历其境的体验，帮助他们了解相关情况；给在此居住过的人似曾相识的回忆。

1.2 范围

1.2.1 软件名称

DOcraft

1.2.2 软件功能

1） 基本体素的建模表达能力

2） OBJ 格式三维网格导入功能

3） 基本材质、纹理的现实和编辑能力

4） 基本几何变换功能

5） 基本光照明模型要求和基本的光源编辑

6） 场景漫游、视角变换功能

7） 动画绘制与播放

8） NURBS 曲面建模绘制

9） 漫游时可实时碰撞检测

10）具有一定的对象表达能力

1.2.3 软件应用

基于该项目的漫游体验程序。

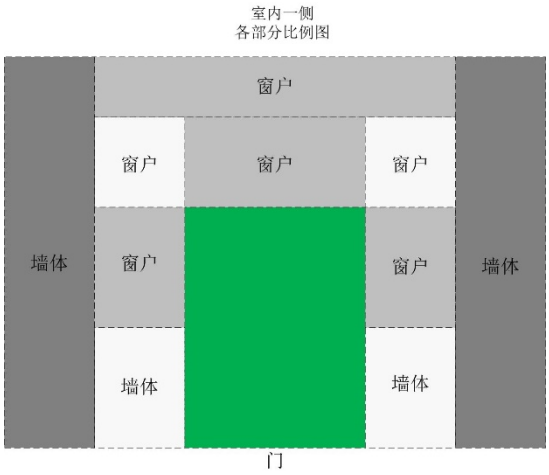
2 分模块说明

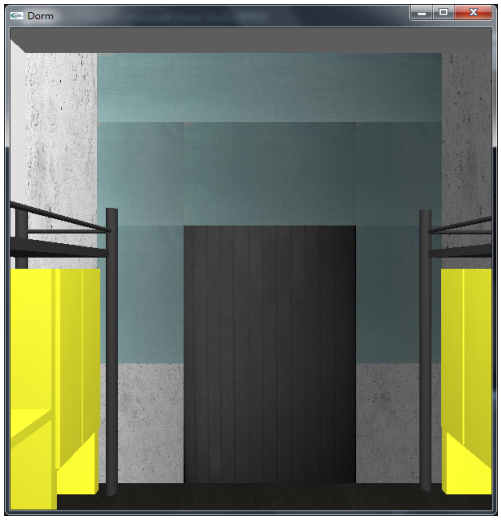
2.1 Draw_Balcony 模块

2.1.1 模块设计说明

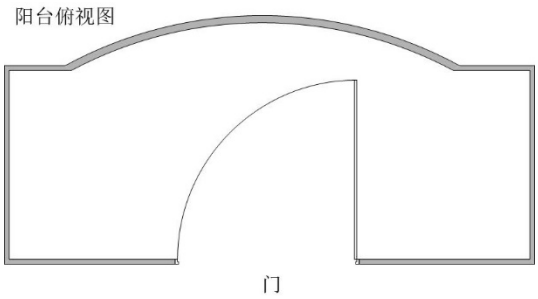
该模块主要绘制了两个部分的内容。

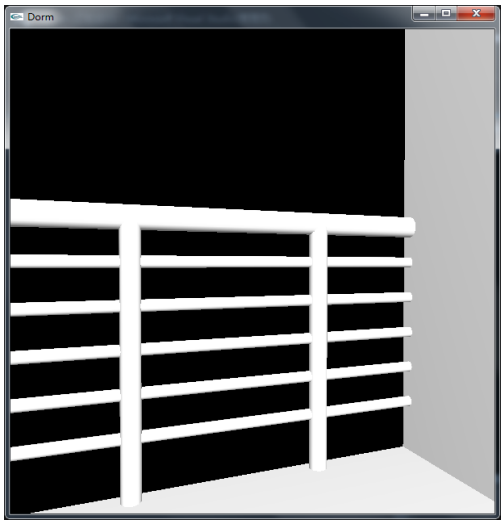
其中阳台与室内间隔墙被分为 11 个部分，窗户实现半透明效果，墙体贴上对应纹理，门调用 render_door 函数进行绘制。





另外阳台部分分为墙体及栏杆 2 个部分，分别使用 SolidCube 与 Cylinder 进行绘制。





2.2 render_door 模块

2.2.1 模块设计说明

绘制一个由按键控制开关的门。

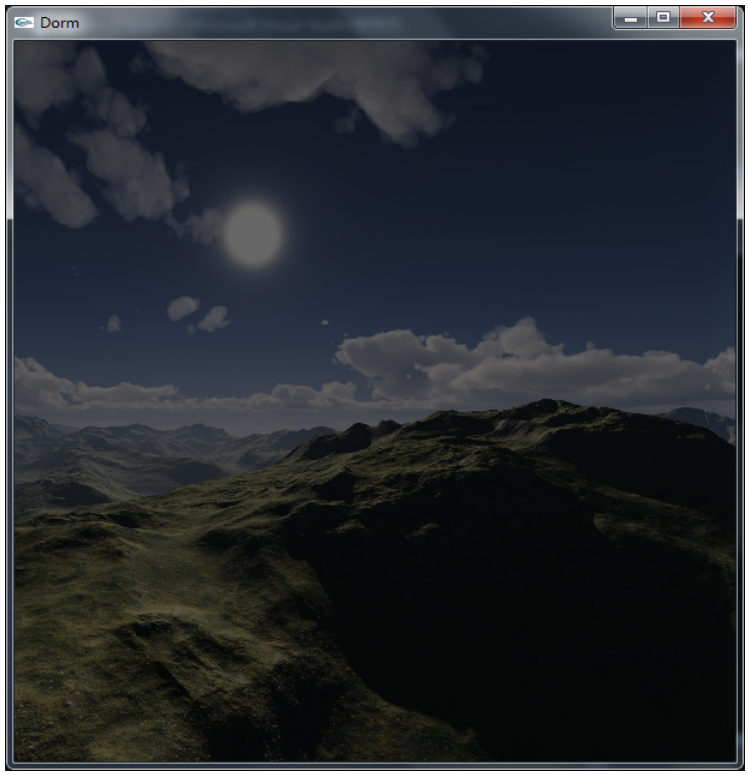
2.2.2 功能实现说明

设定门的转动角度，当按键被按下后，不断累加转动角度实现开门/关门动画。转动角度设定了上/下限值，即门的打开角度被限制。

2.3 Draw_Skybox 模块

2.3.1 模块设计说明

绘制一个长宽高均为 w（视为无穷大）的立方体，在其内部各个面贴上对应纹理，模拟室外环境。



2.3.2 功能实现说明

将单张图片截为 5 个部分，分别贴在立方体的 4 个侧面以及顶面。因图片是连续的且立方体足够大，故视觉效果与远眺效果十分接近。顶面在原始图像中只与其中 1 个侧面邻接，但由于天空的特征较少，在实际使用中顶面纹理到 4 个侧面纹理的过渡十分自然。

2.4 Ball 模块

2.4.1 模块设计说明

在指定范围内绘制一个球体，该球体在到达任一边缘时，弹回该球体以模拟碰撞效果。

2.4.2 功能实现说明

假设球体所碰撞到的边界质量均为无穷大，即可不考虑物体碰撞时的动量变化。又设所有碰撞均为完全弹性碰撞，不损失能量。

分别记录球体运动在各个坐标轴方向上的分速度及界限值。当球体坐标移动到某坐标的界限值时，将球体在该坐标轴方向上的速度反向即可。

通过参数传递各个坐标轴方向上的界限值与初速度，即可实现球体在指定空间范围内的碰撞效果。

2.5 BMP 模块

2.5.1 模块设计说明

读入 bmp 格式的图像文件并载入 texture 数组以备使用。

2.5.2 功能实现说明

添加图像读入函数

```
unsigned char *LoadBitmapFile(char *filename, BITMAPINFOHEADER *bitmapInfoHeader)
```

其中，BITMAPINFOHEADER 是包含了图像的宽高、图像的色深、压缩说明、图像数据的大小和其他一些参数的结构体。

其返回值为 `unsigned char *bitmapImage; // bitmap图像数据`

添加纹理载入函数 `void texload(int i, char *filename)`

```
unsigned char*   bitmapData;                                    // 纹理数据

bitmapData = LoadBitmapFile(filename, &bitmapInfoHeader);
glBindTexture(GL_TEXTURE_2D, texture[i]);
```

将读入的图片信息存入对应的纹理单元并设定相关参数。

添加纹理初始化函数 `void init()`

```
glGenTextures(3, texture);// 第一参数是需要生成标示符的个数, 第二参数是返回标示符的数组
texload(0, "Monet.bmp");
texload(1, "Crack.bmp");
texload(3, "Spot.bmp");
//下面生成自定义纹理
createtexture(); //参考opengl red book，理解后请解释函数的步骤。
glBindTexture(GL_TEXTURE_2D, texture[2]);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1); //设置像素存储模式控制所读取的图像数据的行对齐方式.
glTexImage2D(GL_TEXTURE_2D, 0, 3, 64, 64, 0, GL_RGB, GL_UNSIGNED_BYTE, myImage);
//生成2D纹理，参数分别为：细节级别、颜色组件、宽、高、边框宽度、颜色格式、数据类型、数据指针
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);//使用纹理中坐标最接近的若干个颜色，通过加权平均得到像素颜
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);//图像在物体表面上反复出现
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

完成了从图像转换为纹理的过程。各函数的作用已注释。

2.6 家具绘制模块

2.6.1 模块设计说明

本模块的主要功能是提供了一个 Furniture 类，可以用于绘制学生寝室的床、桌子、橱柜以及进门的柜子梯子，设计的比例基本参照玉泉 8 舍的实物大小进行绘制。Furniture 类封装了 addBed，addDesk，addCupboard，addLadder，addCabinet 等方法，对于这些对象可以指定其生成的位置的元素并将坐标信息存在类中。

2.6.2 模块特点

将绘制物体作为一个类的函数，使代码具有很强的复用性，管理代码更加方便，各个物体的坐标都被存在数据结构中，方便统一地进行管理与查看，将真正的绘制函数作为私有函数，保证代码在整合时的安全性。

2.6.3 模块绘制流程

1) 声明一个 Furniture 类通过
2) 调用方法向内添加物体
3) 为部分物体贴上纹理

2.6.4 功能实现说明

类结构如下：

```
class Furniture
{
private:
    std::vector<int> Type;
    std::vector<float> x;
    std::vector<float> y;
    std::vector<float> z;
    int count;
    void addBed(int type,int x,int y,int z);
    void addDesk(int type,int x,int y,int z,bool sub=true);
```

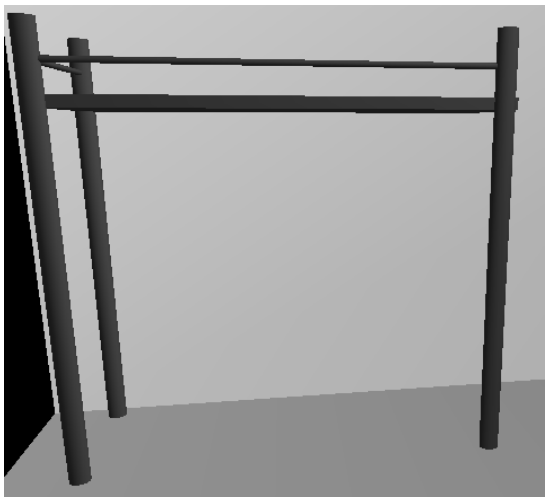
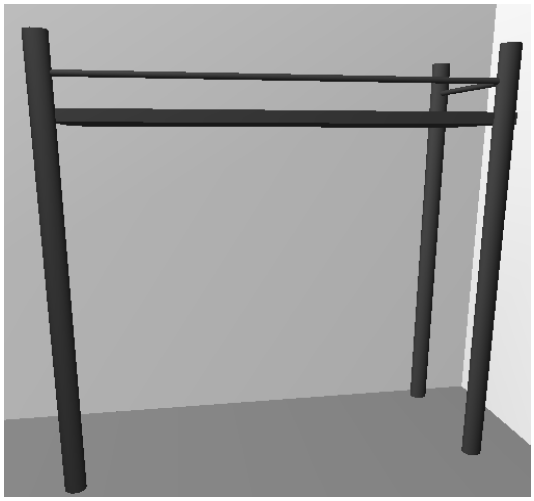
```
    void addCuppboard(int x,int y,int z,bool sub=true);
    void addLadder(int type,int x,int y,int z);
    void addCabinet(int x,int y,int z);
public:
    Furniture();
    ~Furniture();
    void setFurniture(int type,int locationX,int locationY,int
locationZ,int subtype=0,bool sub=false);
    void printFurnitureList();
};
```

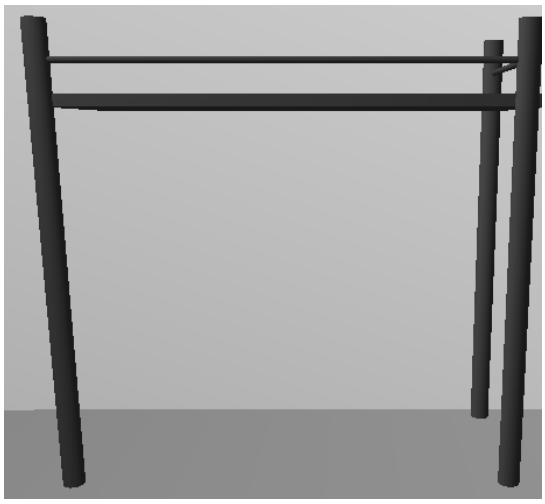
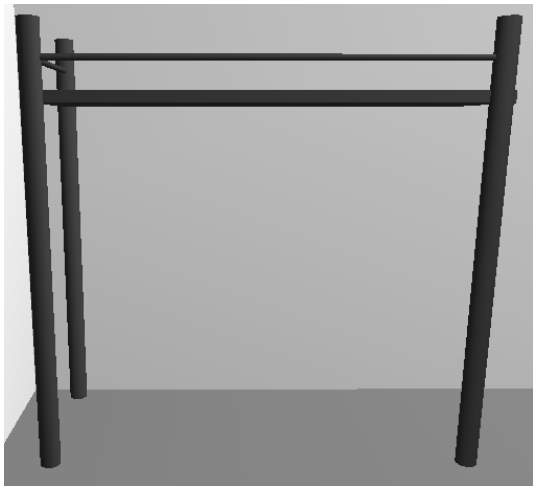
(1) void setFurniture(int type,int locationX,int locationY,int locationZ,int subtype=0,bool sub=false)

setFurniture 是 public 的，是调用 Furniture 类绘制图形时候的调用接口,根据输入 type 不同的调用不同的私有函数进行绘制，sub 参数在这里默认为 false，因为通过调用 setFurniture 是绘制独立的结构，不是附属结构。

(2) void addBed(int type,int x,int y,int z)

addBed 是一个私有函数，只能在 setFurniture 函数中被调用,寝室中的床有四种不同的形态，这四种形态呈镜面对称，并由 type 参数指定，以(x,y,z)为中心绘制一张床，绘制结果如下图所示。

整体效果图：

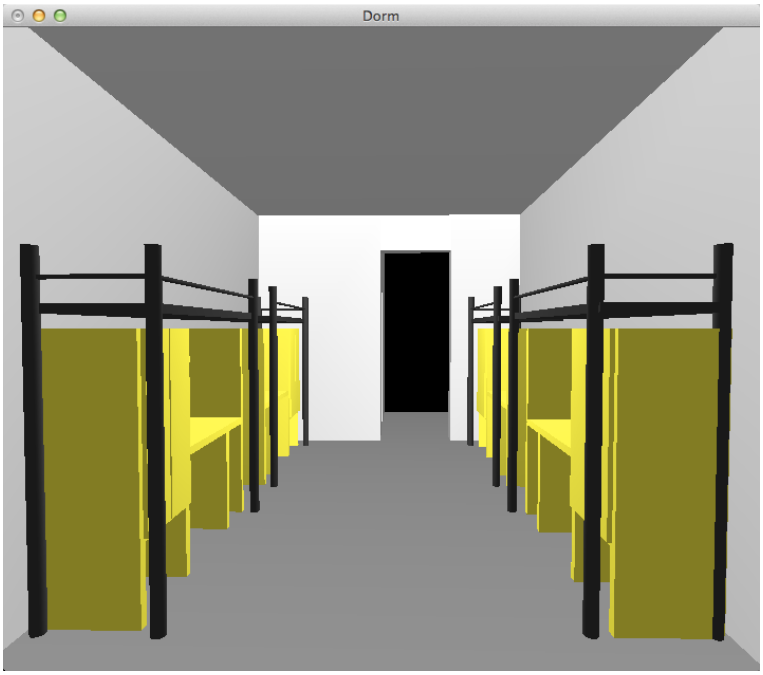
(3)void addDesk(int type,int x,int y,int z,bool sub=true)

addDesk 是一个私有函数，能在 setFurniture 函数和 addBed 中被调用，寝室中的桌子有两种不同的样式，并由 type 参数指定，以(x,y,z)为中心绘制一张桌子，sub 参数默认是 true，意思是附属结构，因为桌子在我们的场景中和床同时出现，当作为附属结构绘制时它的坐标不是由输入的(x,y,z)决定的，而是由 addBed 母函数调用时指定的。当调用 setFurniture 进行单独绘制时，传入参数默认是 false，调用 addDesk 时就会根据 x,y,z 坐标进行单独绘制。

(4)void addCuppboard(int x,int y,int z,bool sub=true)

与 addDesk 类似，在场景中由 addBed 调用绘制作为附属结构，通过 setFurniture 函数调用时进行单独绘制。

寝室的床作为整体结构绘制完成之后，效果如下图所示：

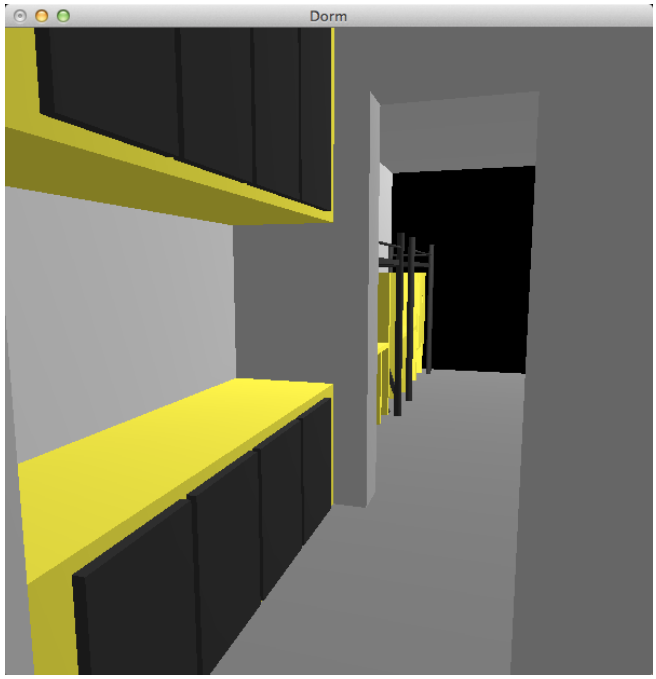


(5)void addLadder(int type,int x,int y,int z)

在(x,y,z)位置绘制阶梯，type=0 时绘制靠右的楼梯，=1 时绘制靠左的楼梯，效果如下：

(6)void addCabinet(int x,int y,int z)

在(x,y,z)位置绘制位于门口的位置的上下两排柜子，效果如下图



(7) `void printFurnitureList()`

该函数打印了目前所有已经添加入 `Furniture` 类的物体，以及其对应的坐标，在场景中输入如下

循环显示如下：

`Type0` 对应床

`Type1` 对应桌子

`Type2` 对应橱柜

`Type3` 对应梯子

`Type4` 对应储物柜

由于实际绘制中桌子和橱柜不是独立结构，属于 `Bed` 的附属结构，故不显示在物体列表中。

```
0 x=141 y=-140 z=-75
0 x=141 y=120 z=-75
3 x=163 y=0 z=-30
3 x=-163 y=0 z=-30
4 x=0 y=0 z=0
0 x=-141 y=-140 z=-75
0
```

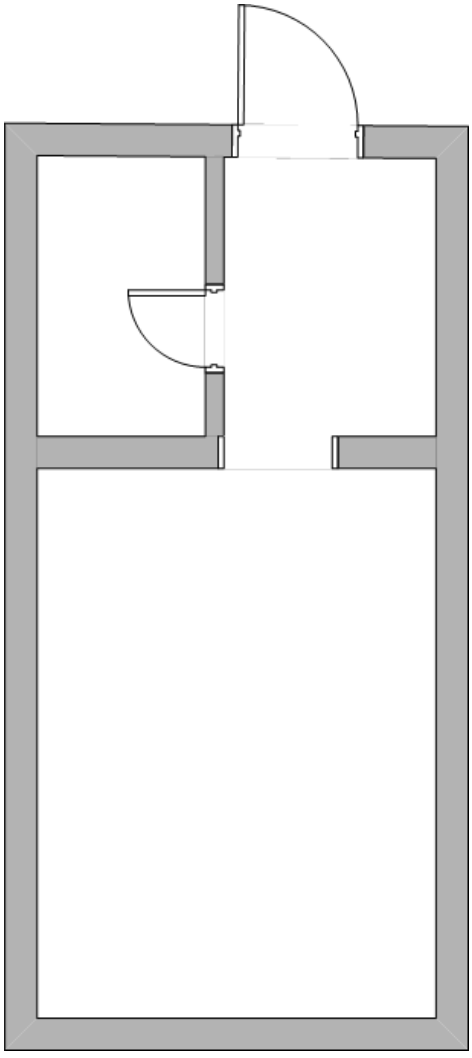
2.7 drawRoom 模块

2.7.1 模块设计说明

此模块对整个寝室的墙壁结构进行建模。能够根据真实的大小比例，表达寝室内各个墙面、天花板、地面、门、玄关、平台等的三维形态。

2.7.2 功能实现说明

为了更好地还原整个结构，我们对房间内的每个尺寸的值进行了实地测量，得到的基本结构图如下：



在具体编程中使用以下的尺寸值来描述上图：

```
/**
 ----| ---
```

```
 |   |    |
 |   |    |
 |---|---|  length
 |       |
 |       |
 ---------
   width
 */

const GLfloat generalWallThickness = 28; //普通墙厚度
const GLfloat toiletWallThickness = 16; //厕所右侧墙的厚度

const GLfloat roomHeight = 300; //3米层高
const GLfloat roomWidth = 346;    //空间宽度

const GLfloat roomLength = 754; //门口到阳台
const GLfloat hallwayLength = 242; //门口到玄关
const GLfloat loungeLength = 484;//玄关到阳台

const GLfloat mainDoorWidth = 104;//门口的门宽度
const GLfloat mainDoorToTop = 93;//门到天花板
const GLfloat mainDoorToLeft = 174;//到左边
const GLfloat mainDoorToRight = 68;//到右边

const GLfloat toiletDoorWidth = 67;//厕所门宽度
const GLfloat toiletDoorToTop = 92;//厕所门到天花板
const GLfloat toiletDoorToCorridor = 115;//从里面看到左边
const GLfloat toiletDoorToBalcony = 60;//从里面看到右边
```

```
const GLfloat toiletFlatHeight = 20;//马桶平台的高度
const GLfloat toiletFlatLength = 95;//马桶平台的长
const GLfloat toiletFlatWidth = 80;//马桶平台的宽
const GLfloat toiletBoardWidth = 10;//洗澡间平台宽
const GLfloat toiletBoardHeight = 5;//洗澡间平台高
const GLfloat toiletTubeSize = 20;//卫生间水管边长
const GLfloat toiletCeilingToTop = 65;//吊顶高度

const GLfloat hallwayDoorWidth = 94;//玄关宽度
const GLfloat hallwayDoorToTop = 46;//玄关的门洞到天花板的高度
const GLfloat hallwayDoorToLeft = 162;//玄关到左边
const GLfloat hallwayDoorToRight = 90;//玄关到右边
```

其中每个墙面使用一个 solidcube 来表达，而针对有门洞的墙，使用 3 个 solidcube 拼合而成。

建模时使用右手系坐标，指向屏幕外为 z 轴正方向，取整个寝室的房间正中央为原点。对每一个墙面的绘制流程如下：

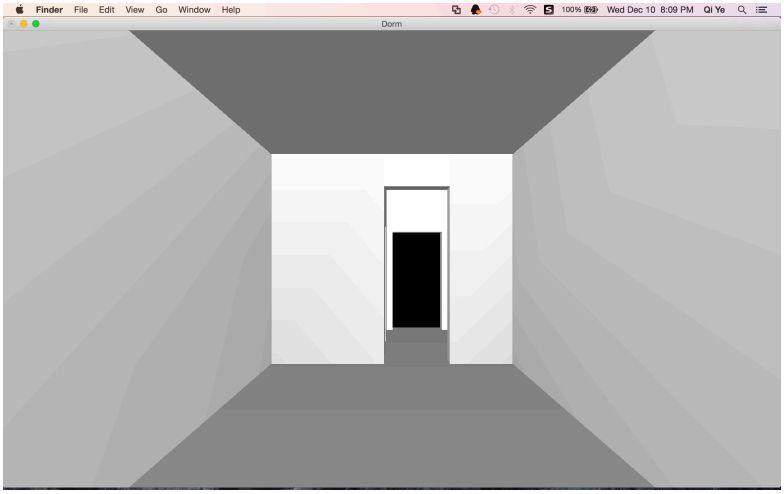
将墙体中心点 translate 到算出的新建的房间坐标系所在位置；

scale 成墙体的大小应有的大小；

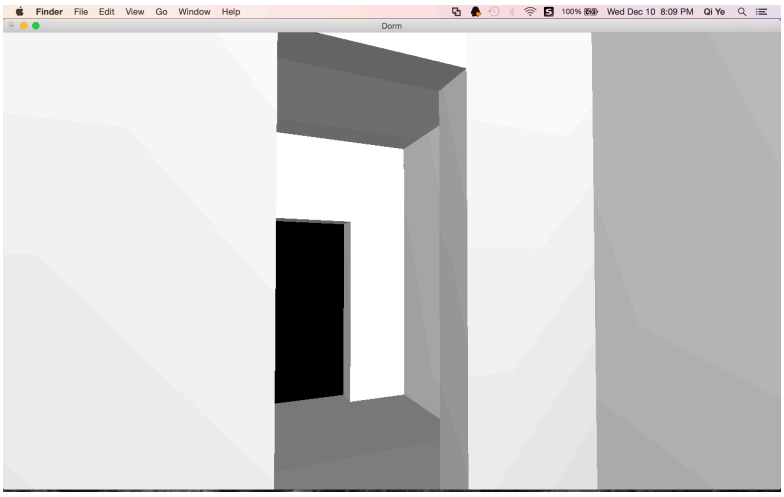
调用 glutSolidCube 画一个单位立方体。

整体绘制时需要主要需要注意 push/pop 的配合，设置 perspective 参数时选取比较符合现实的 60 度，并将前后平面的差值设大一些。最后加上材料的环境光、漫反射光的设置，以及简单的位置光源，呈现出以下效果。

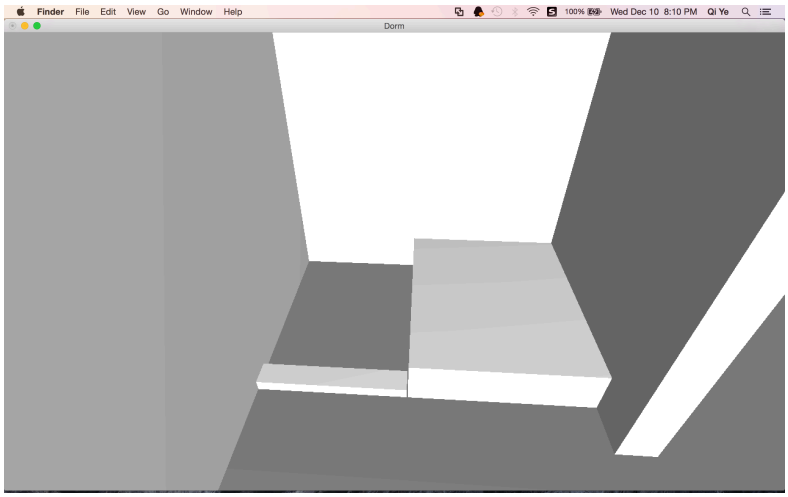
主要生活空间：


Finder File Edit View Go Window Help
Wed Dec 10 8:09 PM Qi Ye
Dorm

玄关：


Finder File Edit View Go Window Help
Wed Dec 10 8:09 PM Qi Ye
Dorm

厕所：


Finder File Edit View Go Window Help
Wed Dec 10 8:10 PM Qi Ye
Dorm

2.8 key 模块

2.8.1 模块设计说明

此模块实现所有键盘交互功能，具体功能表如下：

2.8.2 功能实现说明

对光源部分的控制在实验中已经详细地描述过，这部分只阐述漫游视角变换。

以 cameraRotateLR 与 cameraRotateUD 两个变量来表示旋转的角度，当按下 key 中指定按键后，改变这两个变量的值，redraw 函数中根据它们的变化实时地改变 eye 与 center 的值，以此来实现转换视角的效果。

左右旋转：center 绕着 eye 为中心旋转 cameraRotateLR 度。

```
eyeCameraDistance = sqrt((eye[0] - center[0]) * (eye[0] - center[0])
+ (eye[2] - center[2]) * (eye[2] - center[2]));
center[0] = eye[0] - eyeCameraDistance * sin(cameraRotateLR);
center[2] = eye[2] - eyeCameraDistance * cos(cameraRotateLR);
```

上下旋转：人不能抬头超过 90 度，且移动视角时看的内容也在同一深度上，所以这里只用调整 y 轴的大小。

```
center[1] = eye[1] + eyeCameraDistance * sin(cameraRotateUD);
```

以 eye 指向 center 的向量为方向前进与后退：

```
//前进
eye[2] -= reboundFlag * walkSpace * eyeCameraSlopeZ;
center[2] -= reboundFlag * walkSpace * eyeCameraSlopeZ;
eye[0] -= reboundFlag * walkSpace * eyeCameraSlopeX;
center[0] -= reboundFlag * walkSpace * eyeCameraSlopeX;
//后退
eye[2] += reboundFlag * walkSpace * eyeCameraSlopeZ;
center[2] += reboundFlag * walkSpace * eyeCameraSlopeZ;
```

```
eye[0] += reboundFlag * walkSpace * eyeCameraSlopeX;
center[0] += reboundFlag * walkSpace * eyeCameraSlopeX;
```

以于前进方向正交，垂直于 eye 指向 center 的向量为方向向左与向右：

```
//向左
eye[0] -= reboundFlag * walkSpace * eyeCameraSlopeZ;
center[0] -= reboundFlag * walkSpace * eyeCameraSlopeZ;
eye[2] += reboundFlag * walkSpace * eyeCameraSlopeX;
center[2] += reboundFlag * walkSpace * eyeCameraSlopeX;
//向右
eye[0] += reboundFlag * walkSpace * eyeCameraSlopeZ;
center[0] += reboundFlag * walkSpace * eyeCameraSlopeZ;
eye[2] -= reboundFlag * walkSpace * eyeCameraSlopeX;
center[2] -= reboundFlag * walkSpace * eyeCameraSlopeX;
```

2.9 mouse&motion 模块

2.9.1 模块设计说明

此模块实现了鼠标的交互，能够追踪鼠标的踪迹，实现视角的旋转。

2.9.2 功能实现说明

这里用到了 OpenGL 中与鼠标有关的两个函数 glutMouseFunc 和 glutMotionFunc。

glutMouseFunc：

调用函数 mouse 记录下鼠标按下时的初始值。

鼠标坐标 mouseX，mouseY，标识鼠标已按下的标志 mousePressed，并且记录下刚按下时旋转角度 cameraRotateLR0，cameraRotateUD0。

glutMotionFunc：

调用函数 motion 跟踪鼠标的运动。

设从窗口最左端移到最右端变换的角度为 180 度，则变换过程如下。

```
cameraRotateLR = cameraRotateLR0 + (1.0 * (x- mouseX) / wWidth) * PIE ;
cameraRotateUD = cameraRotateUD0 + (1.0 * (y- mouseY) / wHeight) *
PIE ;
```

2.10 isCollided 模块

2.10.1 模块设计说明

此模块实现了实时碰撞检测，在每一个前进后退与左右移动的操作时进行检测，并返回一个 bool 型的标志，根据此值设置 reboundFlag：reboundFlag 为 1 表示按原方向前进，为-1 时与原前进方向相反。

2.10.2 功能实现说明

以下为检测碰撞的函数 isCollided 原型：

```
bool isCollided(int x, int y, int z);
```

参数 x，y，z 为经历变换后的坐标。

返回值为改变后的坐标点是否被墙体内部围住。

2.11 OBJ 模型与粒子效果模块

2.11.1 0OBJ 数据结构

一个.obj 文件里的记录数据见下方。

例子：

```
v 0.500000 -0.500000 -0.500000
vt 0.001992 0.001992
vn 0.000000 0.000000 1.000000
```

f 1/1/1 2/2/2 3/3/3

V：代表顶点。格式为 V X Y Z，V 后面的 X Y Z 表示三个顶点坐标。浮点型

VT：表示纹理坐标。上面的立方体有 24 个纹理坐标，因为每个三角形面的三个顶点，都需要指定一个纹理坐标。格式为 VT TU TV。浮点型

VN：法向量。同样，上面立方体也有 24 个法向量，因为每个三角形的三个顶点都要指定一个法向量。格式为 VN NX NY NZ。浮点型

F：面。面后面跟着的整型值分别是属于这个面的顶点、纹理坐标、法向量的索引。面的格式为：

F Vertex1/Texture1/Normal1

Vertex2/Texture2/Normal2

Vertex3/Texture3/Normal3

针对以上文件格式，我们设计了这样的数据结构用来储存 obj 文件里的数据：

```
/*********************OBJ Structure************************/
typedef struct{
    int v_num; //记录点的数量
    int vn_num;//记录法线的数量
    int vt_num;//记录纹理坐标的数量
    int f_num; //记录面的数量
    GLfloat *vArr[10000000]; //存放点的二维数组
    GLfloat *vnArr[10000000];//存放法线的二维数组
    //GLfloat *vtArr[10000000];//存放纹理坐标的二维数组
    int *fvArr[10000000]; //存放面顶点的二维数组
    int *fnArr[10000000]; //存放面法线的二维数组
    int *ftArr[10000000]; //存放面纹理坐标的二维数组
  }OBJ;
```

2.11.2 文件读入模块（getLineNum，initOBJ）

由于最开始，代码未经优化，我们设计的文件读入模块共分为三部分，第一个 getLineNum 是读取 OBJ 模型点面等共有多少行的模块，第二个 readFile 是将每一行不同结构读入相应数组中的模块。最后一个 initOBJ 则是整合以上两者的模块，并设置好绘制 OBJ 的 OpenGL 环境变量的函数。但由于在实际过程中，我们发现这样子读取实在太慢，所以采用用空间换时间的方法，不再事先计算出点面等的总数。整合了 readFile 和 getLineNum 函数，所以现在只有 getLineNum 和 initOBJ 两个模块。

在读取 OBJ 文件的模块中，首先读取一行数据，判断首字母，然后对相应数组进行 new 操作，并对接下来数据赋值。同时对计数变量做自增操作（这个在绘制 OBJ 模型模块中需要用到）。要注意的是，f 开头的行读取方式与 v、vt、vn 不同。然而由于文件过大，我们找到

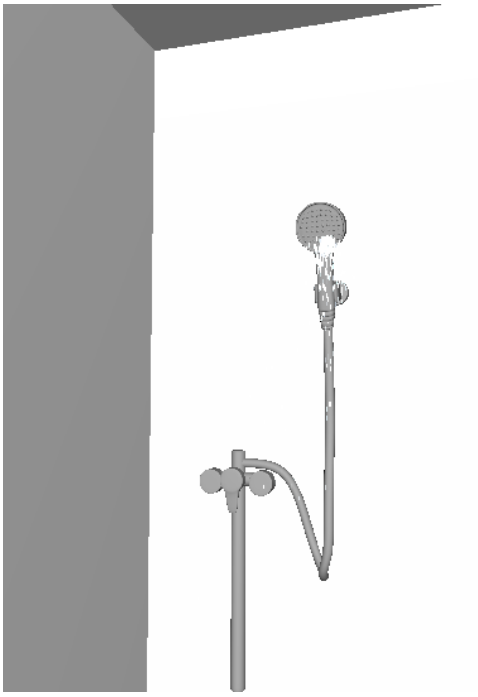
的 OBJ 中有几个超过了 10MB，所以在一开始加载时可能需要较长的时间，约 1 分半左右。

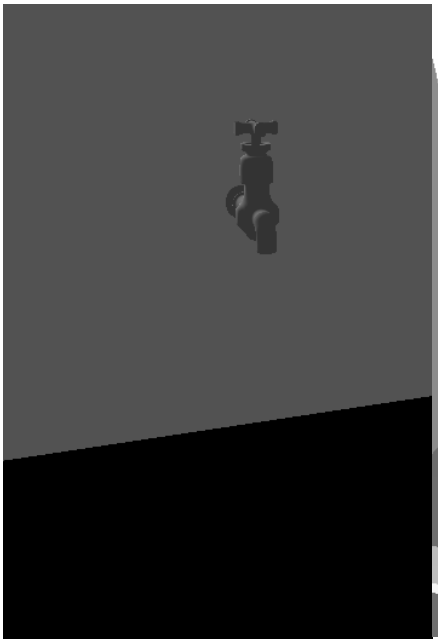
2.11.3 绘制模型模块

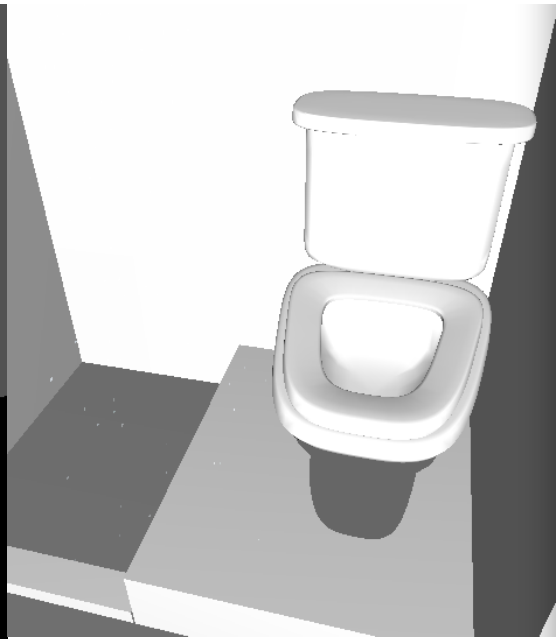
由于 OBJ 模型可以理解为是很多个三角面片拼合而成，所以我们可以用画三角形的方法遍历每个面的顶点来画整个模型。同时，为了表示每个点的法向量，也需要用 glNormal3f 和 OBJ 结构里的 vn 数组来协同完成绘制。

而在 redraw 模块中绘制出 OBJ 图像需要调用此函数。例：Draw_OBJ(&Chair);

以下为我们绘制的效果：







2.11.4 粒子效果的产生

2.11.4.1 粒子结构

粒子效果需要参数见下，其中 particle 结构类型为每一个粒子个体的参数，粒子池表示所有粒子存储的地方。每个粒子只有在激活状态为 1 时才会绘制，激活状态由生命周期与衰退速度共同决定，当生命周期为 0 时又变回不激活，进入粒子池重新初始化。当绘制时，每个粒子的绘制取决于 XYZ 坐标与 RGB 颜色。RGB 颜色从颜色变换池中有规律选择。XYZ 的下一个坐标由当前坐标与当前速度决定，下一个速度由当前速度与加速度决定，从而模仿自然的粒子变化。

```
float  slowdown=2.0f;              // 减速系数
float  xspeed;                     // x轴初始速度
float  yspeed=-30.0f;              // y轴初始速度
float  zoom=-40.0f;              // 缩小系数

typedef struct
{
    bool  active;                  // 激活粒子标志
    float life;                  // 粒子生命周期
    float fade;                  // 衰退速度
    float r,g,b;                   // RGB
    float x,y,z;                   // XYZ轴坐标
    float xi,yi,zi;                    // XYZ方向速度
    float xg,yg,zg;                    // XYZ加速度

}particles;

particles particle[MAX_PARTICLES];   // 粒子池

static GLfloat colors[12][3]=    // 颜色变换池
{};
```

2.11.4.2 粒子生成器

粒子的初始化其实就是对粒子池的粒子进行一次遍历，并对其赋初值。

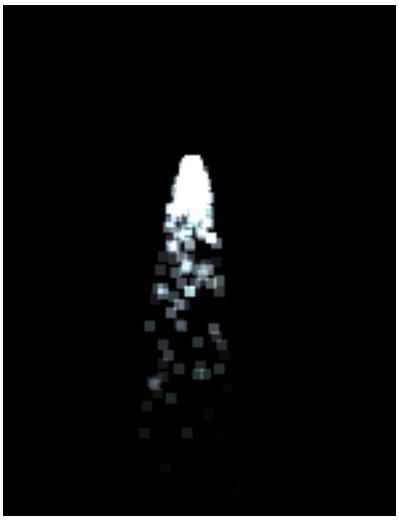
2.11.4.3 粒子绘制器

在我们的室内模拟中，使用到了粒子模块的时淋浴喷头的水花效果，所以在绘制中采用在 xyz 坐标上绘制一个很小的方片，来模拟每一滴水滴。用 GL_TRIANGLE_STRIP 来完成绘制。

在每一次循环中，做以下几件事：1.用 xyz 绘制图形；2.用 xyz 速度更新 xyz，；3.用 xyz 加速度更新 xyz 速度；4.判断粒子是否到了生命周期，如果生命周期终结，则重新初始化粒子。

另外，由于要实现粒子水花的透明效果，使用了颜色混合功能，并且需要关闭深度检测功能 glEnable(GL_BLEND);glDisable(GL_DEPTH_TEST);

以下为粒子绘制效果（为了便于识别，我们使用黑色作为背景）



3 键盘操作说明

按键	作用
W, S, A, D	向前、后、左、右四个方向移动
Z, C	视角上升、下降
Q, E	视角左转、右转
2, 3	抬头、低头
[,]	移动速度加速、减速
0	开启/关闭日光灯
1	开启/关闭手电筒
j, l, i, k, m, .	调整手电筒的方向
/	开启/关闭阳台的门

4 缩略语清单

4.1 OBJ 缩略语

O B J	`int v_num;`	记录点的数量
	`int vn_num;`	记录法线的数量
	`int vt_num;`	记录纹理坐标的数量
	`int f_num;`	记录面的数量
	`GLfloat *vArr[10000000];`	存放点的二维数组
	`GLfloat *vnArr[10000000];`	存放法线的二维数组
	`GLfloat *vtArr[10000000];`	存放纹理坐标的二维数组
	`int *fvArr[10000000];`	存放面顶点的二维数组
	`int *fnArr[10000000];`	存放面法线的二维数组
	`int *ftArr[10000000];`	存放面纹理坐标的二维数组

`void getLineNum(string addrstr, OBJ* obj)`：

`addrstr`：obj文件路径；`obj`：OBJ变量名。

```
void initOBJ(string addrstr,OBJ* obj);
```

addrstr：obj文件路径；obj：OBJ变量名。

```
void Draw_OBJ(OBJ* obj);
```

obj：需要绘制的OBJ变量名。

4.2 粒子缩略语

`float slowdown=2.0f;`	减速系数
`float xspeed;`	x轴初始速度
`float yspeed=-30.0f;`	y轴初始速度
`float zoom=-40.0f;`	缩放系数
MAX_PARTICLES	粒子池粒子数
`static GLfloat colors[12][3]`	颜色变换池

Particles	
`bool active`	激活粒子标志
`float life`	粒子生命周期
`float fade`	衰退速度
`float r,g,b`	RGB
`float x,y,z`	XYZ轴坐标
`float xi,yi,zi`	XYZ方向速度
`float xg,yg,zg`	XYZ方向加速度