教室漫游代码文档

201811153031 18数科 曾锐

[TOC]

程序实现功能

使用OpenGL编程实现室内场景的漫游，要求用户可自由选择观看位置和视线方向。

代码结构

- 库文件

```
#include<gl/glut.h>
#include<windows.h>
#include<math.h>
#include <stdio.h>
#include <stdlib.h>
#pragma warning(disable:4996)
```

- 结构体

```
typedef struct EyePoint
{
    GLfloat x;
    GLfloat y;
    GLfloat z;
}EyePoint;
EyePoint myEye;
EyePoint vPoint;
EyePoint up;
GLfloat vAngle = 0;
```

> Eyepoint结构的作用是确定**当前视角**，包含x，y，z三个GLfloat变量，我们用此结构体来定义三个结构体变量：
>
> - myEye:相当于相机在世界坐标的位置，如果把相机想象成自己的脑袋，那么它就是脑袋所在的位置
> - vPoint:相当于相机镜头对准的物体在世界坐标的位置，或者说是眼睛看的物体的位置
> - up:相机向上的方向在世界坐标中的方向，也就是头顶朝向的方向（因为你可以歪着头看同一个物体）
> - vAngle:角度

- 纹理对象

```
//定义各个纹理对象的名称
GLuint texblackboard, texwindow, texdesk, texsound;
GLuint texceiling, texdoor, texfloor, texbackwall, texpole;
GLuint texairfro, texairback, texhighland, texsdesk, texclock,
texcsc,texcuc,texlmy;
```

> 此处定义了后面要使用的纹理对象的名称

- 各函数功能

```
/*******载入位图作为纹理的相关函数*****/
#define BMP_Header_Length 54//定义BMP头文件大小为54
int power_of_two(int n)//判断一个数是否是2的整数次方
GLuint load_texture(const char* file_name)//载入BMP图作为纹理，并返回纹理编号
/**********************************/
void drawscence()//绘制教室这个大场景
void drawdesks()//绘制桌子
void drawchairs()//绘制椅子
void reshape(int we, int he)//窗口刷新函数，包括视角的刷新
void myDisplay()//显示函数，用来显示教室、桌子以及椅子
GLvoid OnKeyboard(unsigned char key, int x, int y)//响应普通键盘操作，w，s，a，d以及退出esc键
GLvoid OnUPDOWN(int key, int x, int y)////响应特殊键盘操作，即上下左右
```

函数实现

int power_of_two(int n)

```
int power_of_two(int n)
{
    if (n <= 0)
        return 0;
    return (n & (n - 1)) == 0;
}
```

这个函数用来判断一个数是否为2的整数次方，如果是小于等于0，那么直接返回0，说明它不是2的整数次方；否则我们对它进行 **&位运算**，如果它是2的整数次方，那么它的二进制必定是1（0）*的格式，那么这个数减去一的话，二进制就变成了0（1）*的格式，这时对它们进行&运算，如果为0的话，说明这各数是2的整数次方，所以返回0==0，值为1，否则返回0.

GLuint load_texture(const char* file_name)

```
GLuint load_texture(const char* file_name)
{
    GLint width, height, total_bytes;
    GLubyte* pixels = 0;
    GLuint last_texture_ID = 0, texture_ID = 0;

    // 打开文件，如果失败，返回
    FILE* pFile = fopen(file_name, "rb");
    if (pFile == 0)
        return 0;

    // 读取文件中图象的宽度和高度
    fseek(pFile, 0x0012, SEEK_SET);
    fread(&width, 4, 1, pFile);
    fread(&height, 4, 1, pFile);
    fseek(pFile, BMP_Header_Length, SEEK_SET);

    // 计算每行像素所占字节数，并根据此数据计算总像素字节数
    {
        GLint line_bytes = width * 3;
        while (line_bytes % 4 != 0)
            ++line_bytes;
        total_bytes = line_bytes * height;
    }

    // 根据总像素字节数分配内存
    pixels = (GLubyte*)malloc(total_bytes);
    if (pixels == 0)
    {
        fclose(pFile);
        return 0;
    }

    // 读取像素数据
    fread(pixels, total_bytes, 1, pFile);

    // 对就旧版本的兼容，如果图象的宽度和高度不是2的整数次方，则需要进行缩放
    // 若图像宽高超过了OpenGL规定的最大值，也缩放
    {
        GLint max;
        glGetIntegerv(GL_MAX_TEXTURE_SIZE, &max);
        if (power_of_two(width)
            || power_of_two(height)
            || width > max
            || height > max)
        {
            const GLint new_width = 256;
            const GLint new_height = 256; // 规定缩放后新的大小为边长的正方形
            GLint new_line_bytes, new_total_bytes;
            GLubyte* new_pixels = 0;

            // 计算每行需要的字节数和总字节数
            new_line_bytes = new_width * 3;
            while (new_line_bytes % 4 != 0)
                ++new_line_bytes;
            new_total_bytes = new_line_bytes * new_height;
            // 分配内存
            new_pixels = (GLubyte*)malloc(new_total_bytes);
            if (new_pixels == 0)
            {
                free(pixels);
                fclose(pFile);
                return 0;
            }

            // 进行像素缩放
            gluScaleImage(GL_RGB,
                width, height, GL_UNSIGNED_BYTE, pixels,
                new_width, new_height, GL_UNSIGNED_BYTE, new_pixels);

            // 释放原来的像素数据，把pixels指向新的像素数据，并重新设置width和
height
            free(pixels);
            pixels = new_pixels;
            width = new_width;
            height = new_height;
        }
    }

    // 分配一个新的纹理编号
    glGenTextures(1, &texture_ID);
    if (texture_ID == 0)
    {
        free(pixels);
        fclose(pFile);
        return 0;
    }
    // 在绑定前，先获得原来绑定的纹理编号，以便在最后进行恢复
    GLint lastTextureID = last_texture_ID;
    glGetIntegerv(GL_TEXTURE_BINDING_2D, &lastTextureID);
    glBindTexture(GL_TEXTURE_2D, texture_ID);


    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);

    gluBuild2DMipmaps(GL_TEXTURE_2D, 3, width, height, GL_BGR_EXT,
GL_UNSIGNED_BYTE, pixels);

    glBindTexture(GL_TEXTURE_2D, lastTextureID);  //恢复之前的纹理绑定
    free(pixels);
    return texture_ID;
}
```

此函数用于实现载入BMP图作为纹理，并返回纹理编号。 首先定义bmp文件的宽度和高度，像素值，以及两个纹理编号，其中last_Texture_ID用来存储之前绑定的纹理编号，以便在最后进行回恢复。 然后我们打开bmp文件，读取图像的宽度和高度，然后计算每行像素所占的像素值，彩色bmp图为三位，因此将宽度乘3，因为每一行像素值必须是4的倍数，因此不够的我们要进行填充，最后乘上高度就是bmp文件像素值的字节大小了。 根据得到的值为pixels分配内存，然后将bmp图像像素数据存放到pixels中。 在这里，我们需要注意一个情况：如果图像的宽度和高度不是2的整数次方，或者图像的宽高超过了OpenGL的最大值，也需要进行缩放。 因此我们需要比较宽高和OpenGl的最大值max，并且用前面定义的power_of_two来判断宽高是否时候2的整数次方。 如果不满足条件，则我们重新定义一个宽高以及像素值，重新计算字节大小和分配内存。最后使用**gluScaleImage**函数进行像素缩放，这样就得到了符合标准的pixels。 在这之后，我们需要使用**glGenTextures**函数给这个文件分配一个新的纹理编号。在绑定之前，先获得原来的纹理编号，以便在最后进行恢复。 最后使用**glTexParameteri**函数确定如何把纹理象素映射成像素.

> GL_TEXTURE_2D: 操作2D纹理. GL_TEXTURE_WRAP_S: S方向上的贴图模式. GL_TEXTURE_WRAP_T: T方向上的贴图模式. GL_TEXTURE_MIN_FILTER: 缩小过滤 GL_LINEAR_MIPMAP_NEAREST: 使用GL_NEAREST对最接近当前多边形的解析度的两个层级贴图进行采样,然后用这两个值进行线性插值. ...

最后使用**gluBuild2DMipmaps**构建2D纹理，最后恢复之前的纹理编号，释放pixels内存，返回纹理编号。

void drawscence()

```
//绘制教室这个大场景
void drawscence()
{


    /*glTexCoord2f绘制图形时指定纹理的坐标，第一个是X轴坐标，0.0是纹理的左侧，0.5是纹理的中点，1.0是纹理的右侧，
    第二个是Y轴坐标，0.0是纹理的底部，0.5是纹理的中点，1.0是纹理的顶部，
    为了将纹理正确的映射到四边形上，您必须将纹理的右上角映射到四边形的右上角，纹理的左上角映射到四边形的左上角，
    纹理的右下角映射到四边形的右下角，纹理的左下角映射到四边形的左下角，纹理的左上坐标是X：0.0，Y：1.0f，四边形的左上顶点是X：-1.0，Y：1.0。*/


    glEnable(GL_TEXTURE_2D);//开启2D纹理功能
    //天花板
    glBindTexture(GL_TEXTURE_2D, texceiling);//绑定纹理
    glBegin(GL_QUADS);//画四边形
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0f, 30.0f, 30.0f);//左下
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-40.0f, 30.0f, -30.0f);//左上
    glTexCoord2f(1.0f, 1.0f); glVertex3f(40.0f, 30.0f, -30.0f);//右上
    glTexCoord2f(1.0f, 0.0f); glVertex3f(40.0f, 30.0f, 30.0f);//右下
    glEnd();


    //绘制地板
    glBindTexture(GL_TEXTURE_2D, texfloor);
    glBegin(GL_QUADS);
```

```
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-40.0f, 0.0f, 30.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-40.0f, 0.0f, -30.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(40.0f, 0.0f, -30.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(40.0f, 0.0f, 30.0f);
        glEnd();

        //绘制左边墙
        glBindTexture(GL_TEXTURE_2D, texbackwall);
        glBegin(GL_QUADS);
        glNormal3f(1.0f, 0.0f, 0.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-40.0f, 0.0f, 30.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-40.0f, 30.0f, 30.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-40.0f, 30.0f, -30.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0f, 0.0f, -30.0f);
        glEnd();

        //绘制左边窗户
        glBindTexture(GL_TEXTURE_2D, texwindow);
        for (int n = 0; n <= 1; n++)
        {
            glBegin(GL_QUADS);
            glNormal3f(1.0, 0.0f, 0.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f(-39.9, 10, -8 + n * 18);
            glTexCoord2f(1.0f, 1.0f); glVertex3f(-39.9, 20, -8 + n * 18);
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-39.9, 20, -18 + n * 18);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-39.9, 10, -18 + n * 18);
            glEnd();
        }

        //绘制右边墙
        glBindTexture(GL_TEXTURE_2D, texbackwall);
        glBegin(GL_QUADS);
        glNormal3f(-1.0f, 0.0f, 0.0f); //用于定义法线向量
        glTexCoord2f(1.0f, 0.0f); glVertex3f(40.0f, 0.0f, 30.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(40.0f, 30.0f, 30.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(40.0f, 30.0f, -30.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(40.0f, 0.0f, -30.0f);
        glEnd();

        //绘制右边窗户
        glBindTexture(GL_TEXTURE_2D, texwindow);
        glBegin(GL_QUADS);
        glNormal3f(-1.0, 0.0f, 0.0f);                     //用于定义法线向量
        glTexCoord2f(1.0f, 0.0f); glVertex3f(39.5, 10, 10);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(39.5, 20, 10);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(39.5, 20, 0);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(39.5, 10, 0);
        glEnd();

        //绘制后边墙
        glBindTexture(GL_TEXTURE_2D, texbackwall);
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, 1.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0f, 0.0f, 30.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-40.0f, 30.0f, 30.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(40.0f, 30.0f, 30.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(40.0f, 0.0f, 30.0f);
        glEnd();

        //CUC
        glBindTexture(GL_TEXTURE_2D, texcuc);
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, 1.0f); //用于定义法线向量
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-4.8f, 15.0f, 30.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-4.8f, 20.0f, 30.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(4.8f, 20.0f, 30.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(4.8f, 15.0f, 30.0f);
        glEnd();

        //CSC
        glBindTexture(GL_TEXTURE_2D, texcsc);
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, 1.0f); //用于定义法线向量
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-15.0f, 10.0f, 30.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-15.0f, 15.0f, 30.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(15.0f, 15.0f, 30.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(15.0f, 10.0f, 30.0f);
        glEnd();

        //绘制前边墙
        glBindTexture(GL_TEXTURE_2D, texbackwall);
        glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0f, 0.0f, -30.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-40.0f, 30.0f, -30.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(40.0f, 30.0f, -30.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(40.0f, 0.0f, -30.0f);
        glEnd();

        //画钟
        glBindTexture(GL_TEXTURE_2D, texclock);
        glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(23.0f, 18.0f, -29.8f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(23.0f, 20.0f, -29.8f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(25.0f, 20.0f, -29.8f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(25.0f, 18.0f, -29.8f);
        glEnd();

        //绘制空调
        glBindTexture(GL_TEXTURE_2D, texairfro);
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, 1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(33.0f, 0.0f, -26.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(33.0f, 15.0f, -26.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(37.0f, 15.0f, -26.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(37.0f, 0.0f, -26.0f);
        glEnd();

        glBindTexture(GL_TEXTURE_2D, texairback);
```

```
    glBegin(GL_QUADS);
    glNormal3f(-1.0f, 0.0f, 0.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(33.0f, 0.0f, -26.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(33.0f, 15.0f, -26.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(33.0f, 15.0f, -29.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(33.0f, 0.0f, -29.0f);
    glEnd();

    //绘制教室两边石柱前边两根
    glBindTexture(GL_TEXTURE_2D, texpole);
    for (int i = 0; i <= 1; i++)
    {
        glColor3f(1.0f, 1.0f, 1.0f);
        //glColor3f(255, 255, 255);
        //石柱上表面
        glBegin(GL_QUADS);
        glNormal3f(0.0f, -1.0f, 0.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0 + i * 78, 30.0f, -4.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-40.0 + i * 78, 30.0f, -6.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-38.0 + i * 78, 30.0f, -6.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-38.0f + i * 78, 30.0f, -4.0f);
        glEnd();
        //石柱前表面
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, 1.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0 + i * 78, 0.0f, -4.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-40.0 + i * 78, 30.0f, -4.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-38.0 + i * 78, 30.0f, -4.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-38.0 + i * 78, 0.0f, -4.0f);
        glEnd();
        //石柱后表面
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, -1.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0 + i * 78, 0.0f, -6.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-40.0 + i * 78, 30.0f, -6.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-38.0 + i * 78, 30.0f, -6.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-38.0 + i * 78, 0.0f, -6.0f);
        glEnd();
        //石柱右表面
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, -1.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-38.0 + i * 76, 0.0f, -4.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-38.0 + i * 76, 30.0f, -4.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-38.0 + i * 76, 30.0f, -6.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-38.0 + i * 76, 0.0f, -6.0f);
        glEnd();
    }
    //绘制教室两边石柱，后边两根
    for (int j = 0; j <= 1; j++)
    {

        glColor3f(1.0f, 1.0f, 1.0f);
        //glColor3f(255, 255, 255);
        //石柱上表面
        glBegin(GL_QUADS);
        glNormal3f(0.0f, -1.0f, 0.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0 + j * 78, 30.0f,
14.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-40.0 + j * 78, 30.0f,
12.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-38.0 + j * 78, 30.0f,
12.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-38.0f + j * 78, 30.0f,
14.0f);
        glEnd();
        //石柱前表面
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, 1.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0 + j * 78, 0.0f, 14.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-40.0 + j * 78, 30.0f,
14.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-38.0 + j * 78, 30.0f,
14.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-38.0 + j * 78, 0.0f, 14.0f);
        glEnd();
        //石柱后表面
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, -1.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0 + j * 78, 0.0f, 12.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-40.0 + j * 78, 30.0f,
12.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-38.0 + j * 78, 30.0f,
12.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-38.0 + j * 78, 0.0f, 12.0f);
        glEnd();
        //右表面
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, -1.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-38.0 + j * 76, 0.0f, 14.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-38.0 + j * 76, 30.0f,
14.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-38.0 + j * 76, 30.0f,
12.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-38.0 + j * 76, 0.0f, 12.0f);
        glEnd();
    }

    //绘制黑板
```

```
        glBindTexture(GL_TEXTURE_2D, texblackboard);
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, 1.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-20.0, 8.0f, -29.9f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-20.0, 18.0f, -29.9f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(20.0, 18.0f, -29.9f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(20.0, 8.0f, -29.9f);
        glEnd();

        //lmy
        glBindTexture(GL_TEXTURE_2D, texlmy);
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, 1.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-5.0, 20.0f, -29.9f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-5.0, 25.0f, -29.9f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(5.0, 25.0f, -29.9f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(5.0, 20.0f, -29.9f);
        glEnd();

        //绘制教室前边一块高地并贴纹理
        glBindTexture(GL_TEXTURE_2D, texhighland);
        //贴上面
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 1.0f, 0.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-30.0f, 1.5f, -22.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-30.0f, 1.5f, -30.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(30.0f, 1.5f, -30.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(30.0f, 1.5f, -22.0f);
        glEnd();
        //贴左边
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 0.0f, 1.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-30.0f, 0, -22.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-30.0f, 1.5f, -22.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-30.0f, 1.5f, -30.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-30.0f, 0, -30.0f);
        glEnd();
        //贴前边
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 1.0f, 0.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-30.0f, 0, -22.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-30.0f, 1.5f, -22.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(30.0f, 1.5f, -22.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(30.0f, 0, -22.0f);
        glEnd();
        //贴右边
        glBegin(GL_QUADS);
        glNormal3f(0.0f, 1.0f, 0.0f); //用于定义法线向量
        glTexCoord2f(0.0f, 0.0f); glVertex3f(30.0f, 0, -22.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(30.0f, 1.5f, -22.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(30.0f, 1.5f, -30.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(30.0f, 0, -30.0f);
        glEnd();
        //绘制讲台
```

```
    //贴讲台纹理
    glBindTexture(GL_TEXTURE_2D, texsdesk);
    glBegin(GL_QUADS);
    glNormal3f(0.0f, 1.0f, 0.0f); //用于定义法线向量
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-7.5f, 1.5f, -24.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-7.5f, 9.5f, -24.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(7.5f, 9.5f, -24.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(7.5f, 1.5f, -24.0f);
    glEnd();
    glBegin(GL_QUADS);
    glNormal3f(0.0f, 1.0f, 0.0f); //用于定义法线向量
    glTexCoord2f(0.0f, 0.0f); glVertex3f(7.5f, 1.5f, -24.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(7.5f, 9.5f, -24.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(7.5f, 9.5f, -28.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(7.5f, 1.5f, -28.0f);
    glEnd();
    glBegin(GL_QUADS);
    glNormal3f(0.0f, 1.0f, 0.0f); //用于定义法线向量
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-7.5f, 1.5f, -24.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-7.5f, 9.5f, -24.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-7.5f, 9.5f, -28.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-7.5f, 1.5f, -28.0f);
    glEnd();
    glBegin(GL_QUADS);
    glNormal3f(0.0f, 1.0f, 0.0f); //用于定义法线向量
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-7.5f, 9.5f, -24.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-7.5f, 9.5f, -26.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(7.5f, 9.5f, -26.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(7.5f, 9.5f, -24.0f);
    glEnd();
    //画门
    glColor3f(0.521f, 0.121f, 0.0547f);
    glBindTexture(GL_TEXTURE_2D, texdoor);
    glBegin(GL_QUADS);
    glNormal3f(-1.0f, 0.0f, 0.0f); //用于定义法线向量
    glTexCoord2f(0.0f, 0.0f); glVertex3f(39.9f, 0.0f, -25.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(39.9f, 14.0f, -25.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(39.9f, 14.0f, -19.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(39.9f, 0.0f, -19.0f);
    glEnd();

    glDisable(GL_TEXTURE_2D);
}
```

drawscence() 函数用于绘制整个教室和里面的物品装饰（桌子，椅子除外），包括天花板、地板、石柱、四周墙、窗户、黑板、钟、空调、讲台、门、海报等。。。 由于这里每一个物件的绘制都大致相同，因此我们将天花板作为例子来分析：

```
glEnable(GL_TEXTURE_2D);//开启2D纹理功能
    //天花板
glBindTexture(GL_TEXTURE_2D, texceiling);//绑定纹理
```

```
glBegin(GL_QUADS);//画四边形
glTexCoord2f(0.0f, 0.0f); glVertex3f(-40.0f, 30.0f, 30.0f);//左下
glTexCoord2f(0.0f, 1.0f); glVertex3f(-40.0f, 30.0f, -30.0f);//左上
glTexCoord2f(1.0f, 1.0f); glVertex3f(40.0f, 30.0f, -30.0f);//右上
glTexCoord2f(1.0f, 0.0f); glVertex3f(40.0f, 30.0f, 30.0f);//右下
glEnd();
```

首先我们需要用glEnable(GL_TEXTURE_2D)函数来开启2D纹理功能，所以对应的，我们也要在全部纹理贴完后，在代码最后用glDisable(GL_TEXTURE_2D)函数来关闭2D纹理功能。 在开启纹理功能后，我们再用glBindTexture(GL_TEXTURE_2D, texceiling)来绑定对应的天花板纹理， 在这之后，我们需要使用到glTexCoord2f函数和glVertex3f来绘制物体。 glTexCoord2f绘制图形时指定纹理的坐标，第一个是X轴坐标，0.0是纹理的左侧，0.5是纹理的中点，1.0是纹理的右侧， 第二个是Y轴坐标，0.0是纹理的底部，0.5是纹理的中点，1.0是纹理的顶部， 为了将纹理正确的映射到四边形上，您必须将纹理的右上角映射到四边形的右上角，纹理的左上角映射到四边形的左上角， 纹理的右下角映射到四边形的右下角，纹理的左下角映射到四边形的左下角，纹理的左上坐标是X：0.0，Y：1.0f，四边形的左上顶点是X：-1.0，Y：1.0。 glVertex3f用来构造二维平面，它也对应这3维空间里的4个点，并把这四个点连接起来形成一个平面。如果构造对应的几个面，就可以围成一个长方体，依此我们就可以去构造更多的物体，例如：门、石柱、窗户、黑板。。。 在这里特殊的有石柱和窗户两个物体，因为他们的数量不止一个，但他们的形状相同，不同的只是在空间中的位置，因此我们可以使用for循环，来改变它们在空间中的x或y或z坐标，来移动他们的位置。

drawdesks() && drawchairs()

```
//绘制桌子
void drawdesks()
{
    glBegin(GL_QUADS);

    //桌子前面
    glColor3f(0.82,0.68,0.5);
    glVertex3f(-4.0f, 5.0f, 2.0f);
    glVertex3f(2.0f, 5.0f, 2.0f);
    glVertex3f(2.0f, 5.4f, 2.0f);
    glVertex3f(-4.0f, 5.4f, 2.0f);

    //桌子后面
    glVertex3f(-4.0f, 5.0f, -2.0f);
    glVertex3f(-4.0f, 5.0f, -2.0f);
    glVertex3f(2.0f, 5.4f, -2.0f);
    glVertex3f(2.0f, 5.4f, -2.0f);

    //桌子右边
    glVertex3f(2.0f, 5.0f, -2.0f);
    glVertex3f(2.0f, 5.4f, -2.0f);
    glVertex3f(2.0f, 5.4f, 2.0f);
    glVertex3f(2.0f, 5.0f, 2.0f);


    //桌子左边
    glVertex3f(-4.0f, 5.0f, -2.0f);
```

```
        glVertex3f(-4.0f, 5.0f, 2.0f);
        glVertex3f(-4.0f, 5.4f, 2.0f);
        glVertex3f(-4.0f, 5.4f, -2.0f);

        //桌子上边
        glVertex3f(2.0f, 5.4f, 2.0f);
        glVertex3f(-4.0f, 5.4f, 2.0f);
        glVertex3f(-4.0f, 5.4f, -2.0f);
        glVertex3f(2.0f, 5.4f, -2.0f);

        //桌子底部
        glVertex3f(2.0f, 5.0f, 2.0f);
        glVertex3f(-4.0f, 5.0f, 2.0f);
        glVertex3f(-4.0f, 5.0f, -2.0f);
        glVertex3f(2.0f, 5.0f, -2.0f);

        //右前桌腿
        //前
        glColor3f(0.6f, 0.4f, 0.0f);

        glVertex3f(1.8f, 5.0f, 1.6f);
        glVertex3f(1.4f, 5.0f, 1.6f);
        glVertex3f(1.4f, 0.0f, 1.6f);
        glVertex3f(1.8f, 0.0f, 1.6f);

        //后
        glVertex3f(1.8f, 5.0f, 1.2f);
        glVertex3f(1.4f, 5.0f, 1.2f);
        glVertex3f(1.4f, 0.0f, 1.2f);
        glVertex3f(1.8f, 0.0f, 1.2f);

        //右
        glVertex3f(1.8f, 5.0f, 1.6f);
        glVertex3f(1.8f, 5.0f, 1.2f);
        glVertex3f(1.8f, 0.0f, 1.2f);
        glVertex3f(1.8f, 0.0f, 1.6f);

        //左
        glVertex3f(1.4f, 5.0f, 1.6f);
        glVertex3f(1.4f, 5.0f, 1.2f);
        glVertex3f(1.4f, 0.0f, 1.2f);
        glVertex3f(1.4f, 0.0f, 1.6f);

        //右后桌腿
        //前
        glVertex3f(1.8f, 5.0f, -1.2f);
        glVertex3f(1.4f, 5.0f, -1.2f);
        glVertex3f(1.4f, 0.0f, -1.2f);
        glVertex3f(1.8f, 0.0f, -1.2f);

        //后
        glVertex3f(1.8f, 5.0f, -1.6f);
        glVertex3f(1.4f, 5.0f, -1.6f);
        glVertex3f(1.4f, 0.0f, -1.6f);
        glVertex3f(1.8f, 0.0f, -1.6f);

        //右
        glVertex3f(1.8f, 5.0f, -1.6f);
        glVertex3f(1.8f, 5.0f, -1.2f);
        glVertex3f(1.8f, 0.0f, -1.2f);
        glVertex3f(1.8f, 0.0f, -1.6f);

        //左
        glVertex3f(1.4f, 5.0f, -1.6f);
        glVertex3f(1.4f, 5.0f, -1.2f);
        glVertex3f(1.4f, 0.0f, -1.2f);
        glVertex3f(1.4f, 0.0f, -1.6f);

        //左前桌腿
        //前
        glVertex3f(-3.8f, 5.0f, 1.6f);
        glVertex3f(-3.4f, 5.0f, 1.6f);
        glVertex3f(-3.4f, 0.0f, 1.6f);
        glVertex3f(-3.8f, 0.0f, 1.6f);

        //后
        glVertex3f(-3.8f, 5.0f, 1.2f);
        glVertex3f(-3.4f, 5.0f, 1.2f);
        glVertex3f(-3.4f, 0.0f, 1.2f);
        glVertex3f(-3.8f, 0.0f, 1.2f);

        //右
        glVertex3f(-3.8f, 5.0f, 1.6f);
        glVertex3f(-3.8f, 5.0f, 1.2f);
        glVertex3f(-3.8f, 0.0f, 1.2f);
        glVertex3f(-3.8f, 0.0f, 1.6f);

        //左
        glVertex3f(-3.4f, 5.0f, 1.6f);
        glVertex3f(-3.4f, 5.0f, 1.2f);
        glVertex3f(-3.4f, 0.0f, 1.2f);
        glVertex3f(-3.4f, 0.0f, 1.6f);

        //左后桌腿

        //前
        glVertex3f(-3.8f, 5.0f, -1.2f);
        glVertex3f(-3.4f, 5.0f, -1.2f);
        glVertex3f(-3.4f, 0.0f, -1.2f);
        glVertex3f(-3.8f, 0.0f, -1.2f);

        //后
        glVertex3f(-3.8f, 5.0f, -1.6f);
        glVertex3f(-3.4f, 5.0f, -1.6f);
        glVertex3f(-3.4f, 0.0f, -1.6f);
        glVertex3f(-3.8f, 0.0f, -1.6f);

        //右
```

```
    glVertex3f(-3.8f, 5.0f, -1.6f);
    glVertex3f(-3.8f, 5.0f, -1.2f);
    glVertex3f(-3.8f, 0.0f, -1.2f);
    glVertex3f(-3.8f, 0.0f, -1.6f);

    //左
    glVertex3f(-3.4f, 5.0f, -1.6f);
    glVertex3f(-3.4f, 5.0f, -1.2f);
    glVertex3f(-3.4f, 0.0f, -1.2f);
    glVertex3f(-3.4f, 0.0f, -1.6f);



    glEnd();


}
```

```

//绘制椅子
void drawchairs()
{
    glColor3f(0.8, 0.68, 0.5);
    glBegin(GL_QUADS);

    //椅子前面
    glVertex3f(-2.0f, 3.0f, 2.0f);
    glVertex3f(2.0f, 3.0f, 2.0f);
    glVertex3f(2.0f, 3.4f, 2.0f);
    glVertex3f(-2.0f, 3.4f, 2.0f);

    //椅子右边
    glVertex3f(2.0f, 3.0f, -2.0f);
    glVertex3f(2.0f, 3.4f, -2.0f);
    glVertex3f(2.0f, 3.4f, 2.0f);
    glVertex3f(2.0f, 3.0f, 2.0f);

    //椅子后边
    glVertex3f(-2.0f, 3.0f, -2.0f);
    glVertex3f(-2.0f, 3.4f, -2.0f);
    glVertex3f(2.0f, 3.4f, -2.0f);
    glVertex3f(2.0f, 3.0f, -2.0f);

    //椅子左边
    glVertex3f(-2.0f, 3.0f, -2.0f);
    glVertex3f(-2.0f, 3.0f, 2.0f);
    glVertex3f(-2.0f, 3.4f, 2.0f);
    glVertex3f(-2.0f, 3.4f, -2.0f);

    //椅子上面
    glVertex3f(2.0f, 3.4f, 2.0f);
    glVertex3f(-2.0f, 3.4f, 2.0f);
```

```
        glVertex3f(-2.0f, 3.4f, -2.0f);
        glVertex3f(2.0f, 3.4f, -2.0f);

        //椅子底部
        glVertex3f(2.0f, 3.0f, 2.0f);
        glVertex3f(-2.0f, 3.0f, 2.0f);
        glVertex3f(-2.0f, 3.0f, -2.0f);
        glVertex3f(2.0f, 3.0f, -2.0f);

        //桌子前面的桌腿
        //桌腿前
        glVertex3f(1.8f, 3.0f, 1.6f);
        glVertex3f(1.4f, 3.0f, 1.6f);
        glVertex3f(1.4f, 0.0f, 1.6f);
        glVertex3f(1.8f, 0.0f, 1.6f);

        //后
        glVertex3f(1.8f, 3.0f, 1.2f);
        glVertex3f(1.4f, 3.0f, 1.2f);
        glVertex3f(1.4f, 0.0f, 1.2f);
        glVertex3f(1.8f, 0.0f, 1.2f);

        //右
        glVertex3f(1.8f, 3.0f, 1.6f);
        glVertex3f(1.8f, 3.0f, 1.2f);
        glVertex3f(1.8f, 0.0f, 1.2f);
        glVertex3f(1.8f, 0.0f, 1.6f);

        //左
        glVertex3f(1.4f, 3.0f, 1.6f);
        glVertex3f(1.4f, 3.0f, 1.2f);
        glVertex3f(1.4f, 0.0f, 1.2f);
        glVertex3f(1.4f, 0.0f, 1.6f);

        //后面桌腿
        glVertex3f(1.8f, 3.0f, -1.2f);
        glVertex3f(1.4f, 3.0f, -1.2f);
        glVertex3f(1.4f, 0.0f, -1.2f);
        glVertex3f(1.8f, 0.0f, -1.2f);

        //后
        glVertex3f(1.8f, 3.0f, -1.6f);
        glVertex3f(1.4f, 3.0f, -1.6f);
        glVertex3f(1.4f, 0.0f, -1.6f);
        glVertex3f(1.8f, 0.0f, -1.6f);

        //右
        glVertex3f(1.8f, 3.0f, -1.6f);
        glVertex3f(1.8f, 3.0f, -1.2f);
        glVertex3f(1.8f, 0.0f, -1.2f);
        glVertex3f(1.8f, 0.0f, -1.6f);

        //左
        glVertex3f(1.4f, 3.0f, -1.6f);
```

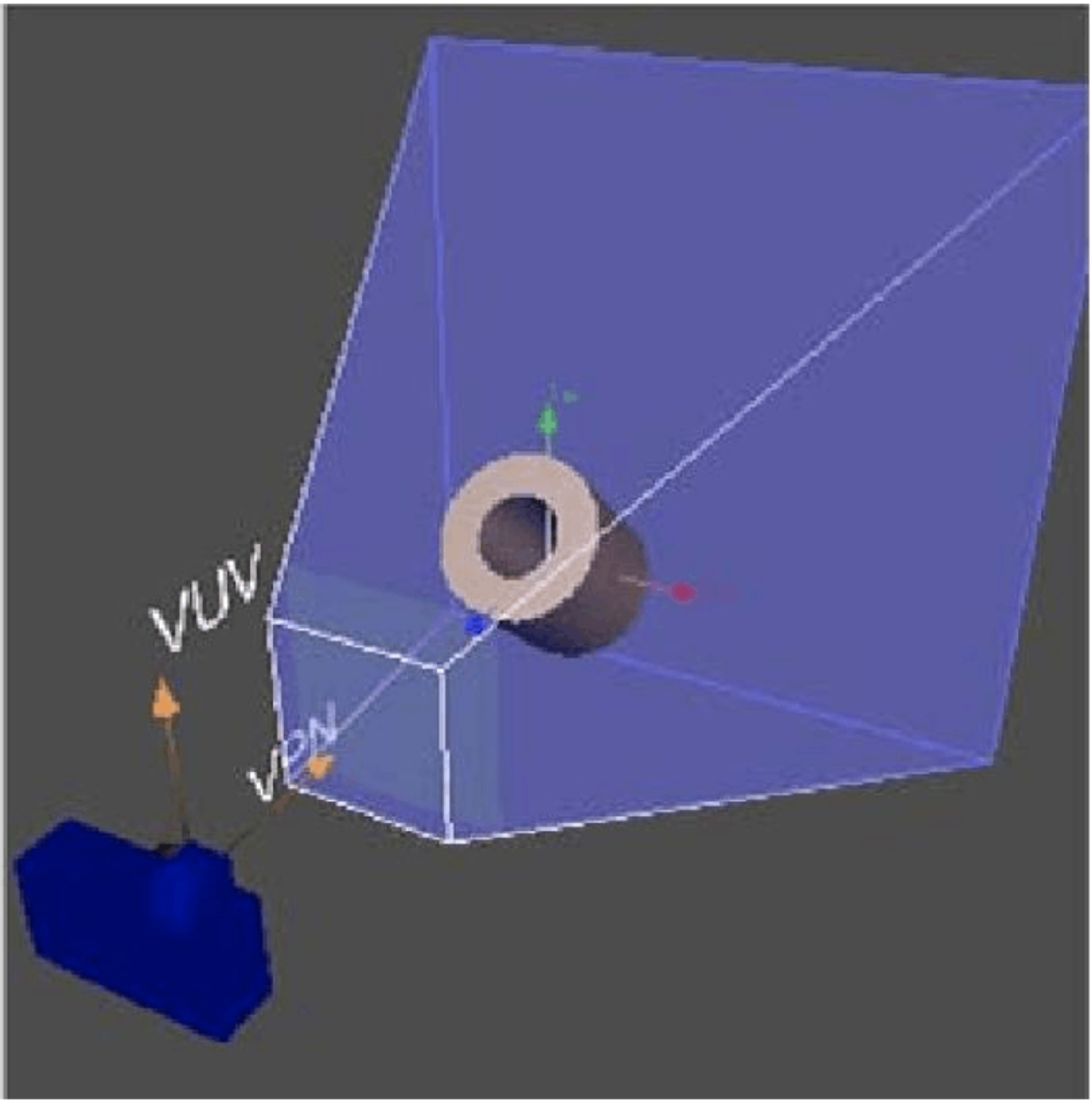
```
    glVertex3f(1.4f, 3.0f, -1.2f);
    glVertex3f(1.4f, 0.0f, -1.2f);
    glVertex3f(1.4f, 0.0f, -1.6f);

    //左前
    glVertex3f(-1.8f, 3.0f, 1.6f);
    glVertex3f(-1.4f, 3.0f, 1.6f);
    glVertex3f(-1.4f, 0.0f, 1.6f);
    glVertex3f(-1.8f, 0.0f, 1.6f);

    //后
    glVertex3f(-1.8f, 3.0f, 1.2f);
    glVertex3f(-1.4f, 3.0f, 1.2f);
    glVertex3f(-1.4f, 0.0f, 1.2f);
    glVertex3f(-1.8f, 0.0f, 1.2f);

    //右
    glVertex3f(-1.8f, 3.0f, 1.6f);
    glVertex3f(-1.8f, 3.0f, 1.2f);
    glVertex3f(-1.8f, 0.0f, 1.2f);
    glVertex3f(-1.8f, 0.0f, 1.6f);

    //左
    glVertex3f(-1.4f, 3.0f, 1.6f);
    glVertex3f(-1.4f, 3.0f, 1.2f);
    glVertex3f(-1.4f, 0.0f, 1.2f);
    glVertex3f(-1.4f, 0.0f, 1.6f);

    //左腿后 前

    //前
    glVertex3f(-1.8f, 3.0f, -1.2f);
    glVertex3f(-1.4f, 3.0f, -1.2f);
    glVertex3f(-1.4f, 0.0f, -1.2f);
    glVertex3f(-1.8f, 0.0f, -1.2f);

    //后
    glVertex3f(-1.8f, 3.0f, -1.6f);
    glVertex3f(-1.4f, 3.0f, -1.6f);
    glVertex3f(-1.4f, 0.0f, -1.6f);
    glVertex3f(-1.8f, 0.0f, -1.6f);

    //右
    glVertex3f(-1.8f, 3.0f, -1.6f);
    glVertex3f(-1.8f, 3.0f, -1.2f);
    glVertex3f(-1.8f, 0.0f, -1.2f);
    glVertex3f(-1.8f, 0.0f, -1.6f);

    //左
    glVertex3f(-1.4f, 3.0f, -1.6f);
    glVertex3f(-1.4f, 3.0f, -1.2f);
    glVertex3f(-1.4f, 0.0f, -1.2f);
    glVertex3f(-1.4f, 0.0f, -1.6f);
    glEnd();
}
```

这里的drawdesks()和drawchairs()在本质上都是通过是glVertex3f函数来贴合四边形，以达到构造物体结构的目的。不同的是在这里使用了glColor3f函数来设置桌子和凳子的颜色。这里的难点并不在于构造物体，而是如何在教室后构造足够数量的桌椅，并把它们整齐地组合在一起。

reshape

```
//窗口刷新函数
void reshape(int we, int he)
{

    WinWidth = we;
    WinHeight = he;
    glViewport(0, 0, (GLsizei)we, (GLsizei)he);/*0 0指定了窗口的左下角位置
    //width,height表示视口矩形的宽度和高度，根据窗口的实时变化重绘窗口。*/
    glMatrixMode(GL_PROJECTION);//声明下一步操作为投影
    glLoadIdentity();//恢复初始坐标系
    gluPerspective(90.0f, (GLfloat)we / (GLfloat)he, 0.01f, 100.0f);
    /*透视投影：90.0f近裁剪平面与远裁剪平面的连线与视点的角度，
    也称视场角,0.01f近裁剪面到相机(视点)的距离,100.0f远裁剪面到相机(视点)的距离*/
    glMatrixMode(GL_MODELVIEW);//声明下一步为对模型视图的操作
    glLoadIdentity();
    /*第一组eyex, eyey,eyez 相机在世界坐标的位置(视点)
    第二组centerx,centery,centerz 相机镜头对准的物体在世界坐标的位置
    第三组upx,upy,upz 相机向上的方向在世界坐标中的方向
    把相机想象成为你自己的脑袋：
    第一组数据就是脑袋的位置
    第二组数据就是眼睛看的物体的位置
    第三组就是头顶朝向的方向（因为你可以歪着头看同一个物体）*/
    gluLookAt(myEye.x, myEye.y, myEye.z, vPoint.x + 30 * sin(vAngle),
vPoint.y, -30 * cos(vAngle), up.x, up.y, up.z);//0.0f, 1.0f, 0.0f);
}
```

reshape()函数用来实现窗口的创建刷新以及视角的变换。其中较为重要的就是gluLookAt()函数，我们可以借助下图来理解



- 第一组数据就是脑袋的位置
- 第二组数据就是眼睛看的物体的位置
- 第三组就是头顶朝向的方向（因为你可以歪着头看同一个物体） 其中第二组数据，我们使用了sin()函数和设定的全局变量vAngle来调整眼睛看的物体的位置。

myDisplay()

```
//显示函数
void myDisplay()
{
    // 清除屏幕
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    //调用绘制函数
    drawscence();
    for (int i = -3; i <= 3; i += 2) {
        for (int j = -3; j <= 3; j += 2) {
            glPushMatrix();
            glTranslatef(i * 5.0f, 3.0f, j * 3.0 + 2.2f);
            glScalef(0.8f, 0.8f, 0.5f);
            drawdesks();
            glPopMatrix();
        }
    }
    for (int i = -3; i <= 3; i += 2) {
        for (int j = -3; j <= 3; j += 2) {
            glPushMatrix();
            glTranslatef(i * 5.0f + 0.2f, 3.0f, j * 3.0 + 5.5f);
            glScalef(0.8f, 0.8f, 0.5f);
            drawchairs();
            glPopMatrix();
        }
    }
    glFlush();//清空缓冲区
}
```

在myDisplay()函数里，主要实现了场景的显示功能。首先调用glClear()函数来清除屏幕，然后调用drawscence()来绘制教室，这里较为重要的是在教室中绘制数量足够多且排列整齐的桌椅。 我们在这里使用了两个for循环，通过i和j来实现对桌椅位置的对称移动。 其中glPushMatrix()函数用来把当前位置入栈，搭配之后的glPopMatrix()函数，将栈顶的位置出栈，实现了保存原始位置的目的。glTranslatef（）函数则通过三个x，y，z参数来移动到相应的位置，glScalef（）搭配drawdesks()使用则是将绘制的desks在相应坐标上伸缩不同的大小。通过这几个函数的搭配，以及嵌套for循环中i、j的变换，实现了对桌子在不同位置上的复制。 同样，我们在对椅子的复制中，只需要将glTranslatef()中的参数适当地进行修改，就可以和桌子整齐地排列在一起了。

OnKeyboard() && OnUPDOWN()

```
//响应普通键盘操作，w，s，a，d以及退出esc键
GLvoid OnKeyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 97://a 向左
        myEye.x -= 0.5;
        vPoint.x -= 0.5;
        if (myEye.x <= -38)
            myEye.x = -38;
        break;
    case 100://d 向右
        myEye.x += 0.5;
        vPoint.x += 0.5;
        if (myEye.x >= 38)
            myEye.x = 38;
        break;
    case 119://w 向前
        myEye.z -= 0.5;
        if (myEye.z <= -28)
            myEye.z = -28;
        break;
    case 115://s 向后
        myEye.z += 0.5;
        if (myEye.z >= 28)
            myEye.z = 28;
        break;
    case 27://esc
        exit(0);

    }
    reshape(WinWidth, WinHeight);//窗口刷新
    glutPostRedisplay();

}
//响应特殊键盘操作
GLvoid OnUPDOWN(int key, int x, int y)//上下左右
{
    switch (key)
    {
    case GLUT_KEY_LEFT:
        vAngle -= 0.05;
        break;
    case GLUT_KEY_RIGHT:
        vAngle += 0.05;
        break;
    case GLUT_KEY_UP:
        myEye.y += 0.5;
        if (myEye.y >= 30)
            myEye.y = 30;

        break;
    case GLUT_KEY_DOWN:
        myEye.y -= 0.5;
        if (myEye.y <= 0)
            myEye.y = 30;
        break;
    }
    reshape(WinWidth, WinHeight);
    glutPostRedisplay();
}
```

OnKeyboard()函数用来响应普通键盘操作，w，s，a，d以及退出esc键，我们对wsad四个键设置了不同的反应，用来控制myEye和vPoint的参数变化，也就是脑袋位置和眼睛所看位置的变化。同时，还加入了碰撞检测，如果超过所设定的边界，则不可以再移动了。ese键则对应exit(0)，用来退出程序。 同样的，OnUPDOWN() 函数用来响应上下左右键，改变vAngle和myEye的值。同样的，我们在上下键中也加入了碰撞检测。

main()

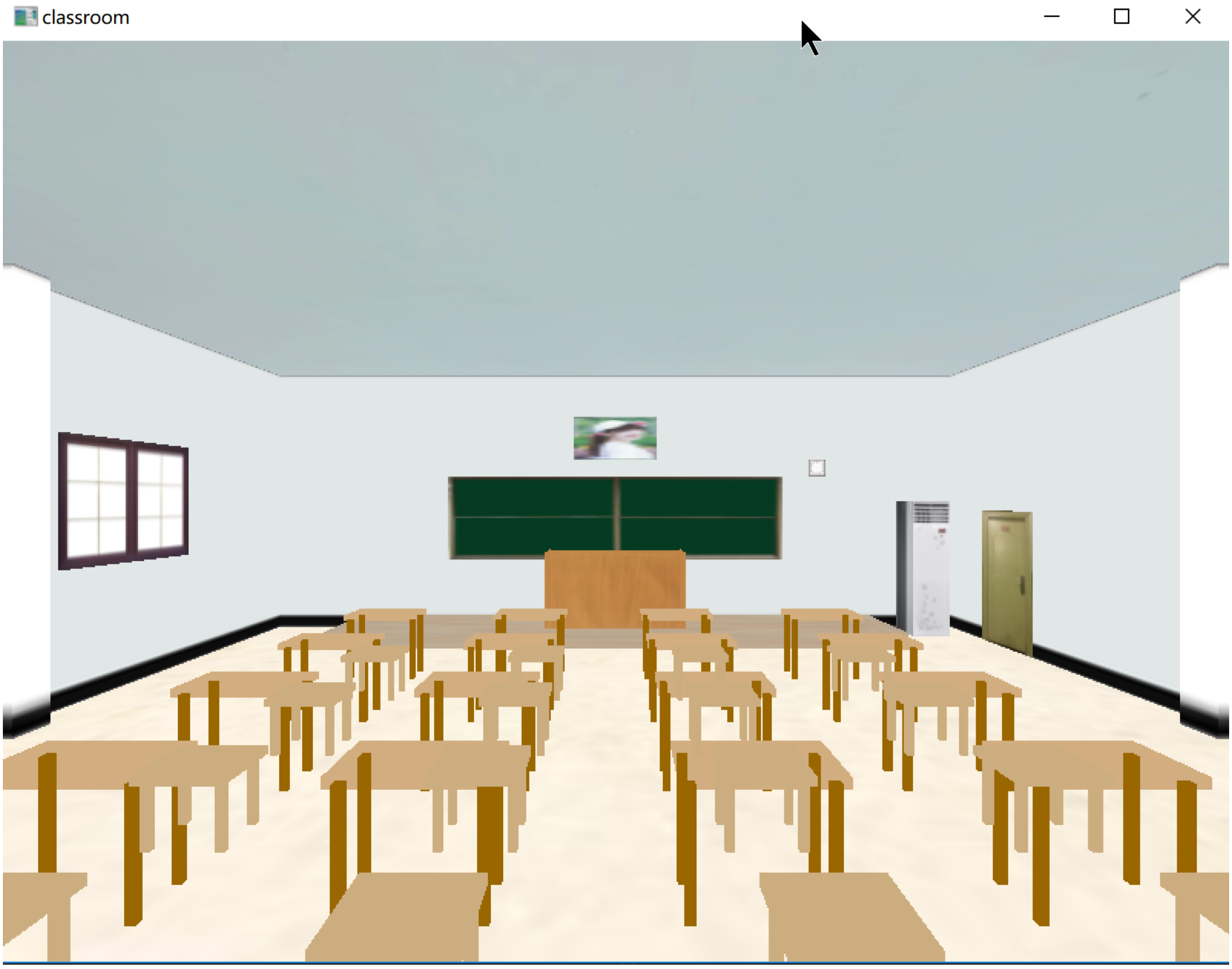
```
int main(int argc, char* argv[])
{
    myEye.x = 0;
    myEye.y = 15;
    myEye.z = 25;
    vPoint.x = 0;
    vPoint.y = 15;
    vPoint.z = -30;
    up.x = 0;
    up.y = 1;
    up.z = 0;
    vAngle = 0;
    glEnable(GL_DEPTH_TEST);//启用了之后，OpenGL在绘制的时候就会检查，当前像素前面是否有别的像素，如果别的像素挡道了它，那它就不会绘制，也就是说，OpenGL就只绘制最前面的一层。
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);//定义显示方式，GLUT_RGBA指定一个RGBA窗口，GLUT_SINGLE.单缓冲区窗口
    glutInitWindowPosition(0, 0);//设置初始窗口的位置
    glutInitWindowSize(800, 600);//设置窗口大小
    glutCreateWindow("classroom");
    glutDisplayFunc(&myDisplay);//注册显示回调函数,包含重绘场景所需的所有代码
    glutReshapeFunc(reshape);//注册窗口改变回调函数
    glutKeyboardFunc(OnKeyboard);//注册键盘响应事件
    glutSpecialFunc(OnUPDOWN);//对键盘上特殊的4个方向按键的响应函数

    texblackboard = load_texture("blackboard.bmp");
    texwindow = load_texture("window.bmp");
    texsound = load_texture("sound.bmp");
    texceiling = load_texture("ceiling.bmp");
    texdoor = load_texture("door.bmp");
    texfloor = load_texture("floor.bmp");
    texbackwall = load_texture("backwall.bmp");
    texpole = load_texture("pole.bmp");
    texairfro = load_texture("airconditionfront.bmp");
    texairback = load_texture("airconditionback.bmp");
    texhighland = load_texture("gaodi.bmp");
    texsdesk = load_texture("sdesk.bmp");
    texclock = load_texture("clock.bmp");
    texcuc = load_texture("CUC.bmp");
    texcsc = load_texture("CSC.bmp");
    texlmy = load_texture("lmy.bmp");

    glutMainLoop();//进入事件处理循环
    return 0;
}
```

在主函数中，我们定义了三个结构体myEye、vPoint、up和vAngle的值，然后开启OpenGL(调用各种初始化函数），设置显示窗口的各项数值，注册显示回调函数,包含重绘场景所需的所有代码，注册窗口改变回调函数，注册键盘响应事件，对键盘上特殊的4个方向按键的响应函数

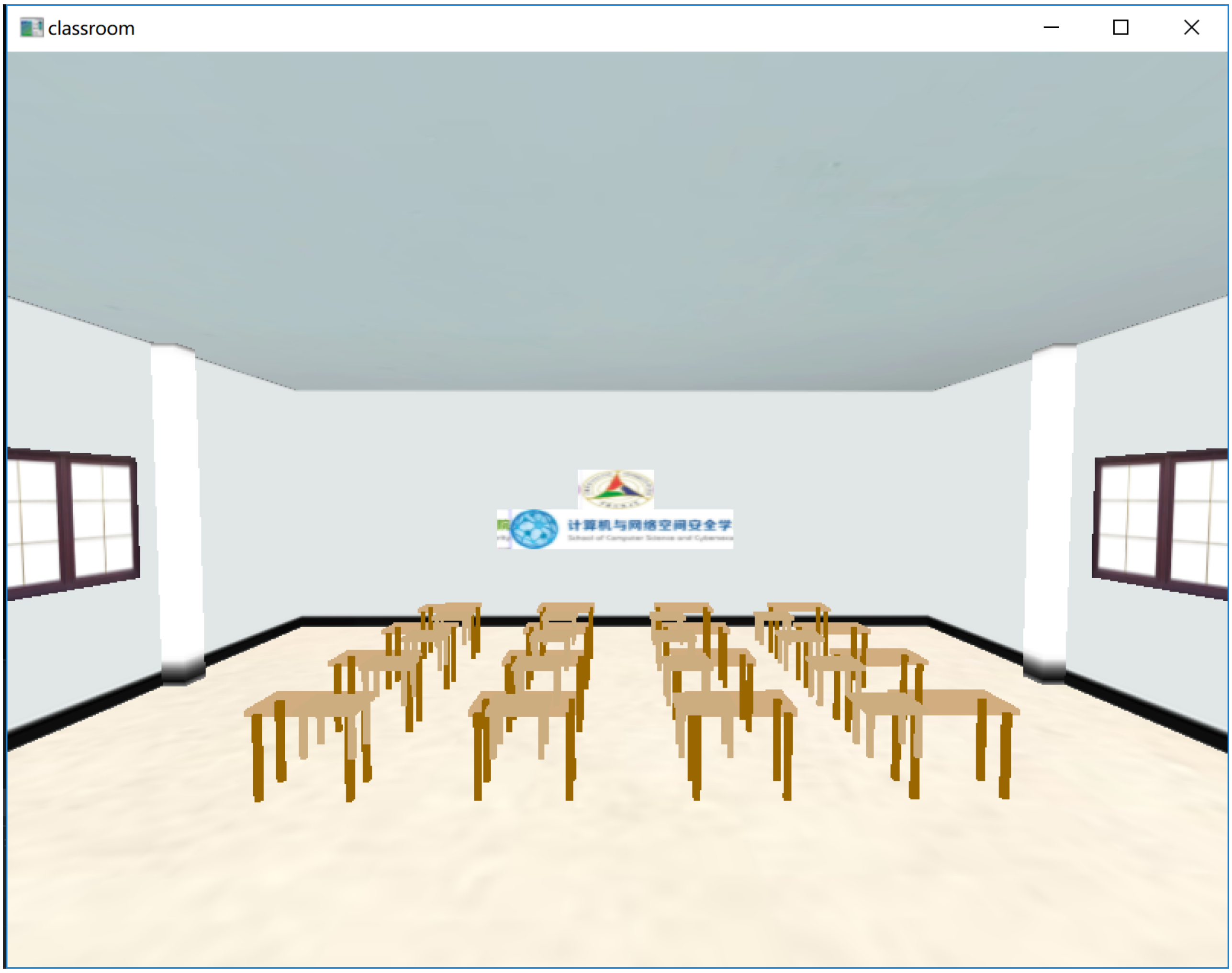
```
st=>start: 定义了三个结构体myEye、vPoint、up和vAngle的值
e=>end
op1=>operation: 开启OpenGL
op2=>operation: 设置显示窗口的各项数值
op3=>operation: 注册显示回调函数
op4=>operation: 注册窗口改变回调函数
op5=>operation: 注册键盘响应事件
op6=>operation: 设置纹理编号
op7=>operation: 进入事件处理循环
cond=>condition: ESC?

st->op1->op2->op3->op4->op5->op6->op7->cond
cond(yes)->e
cond(no)->op1
```

效果展示

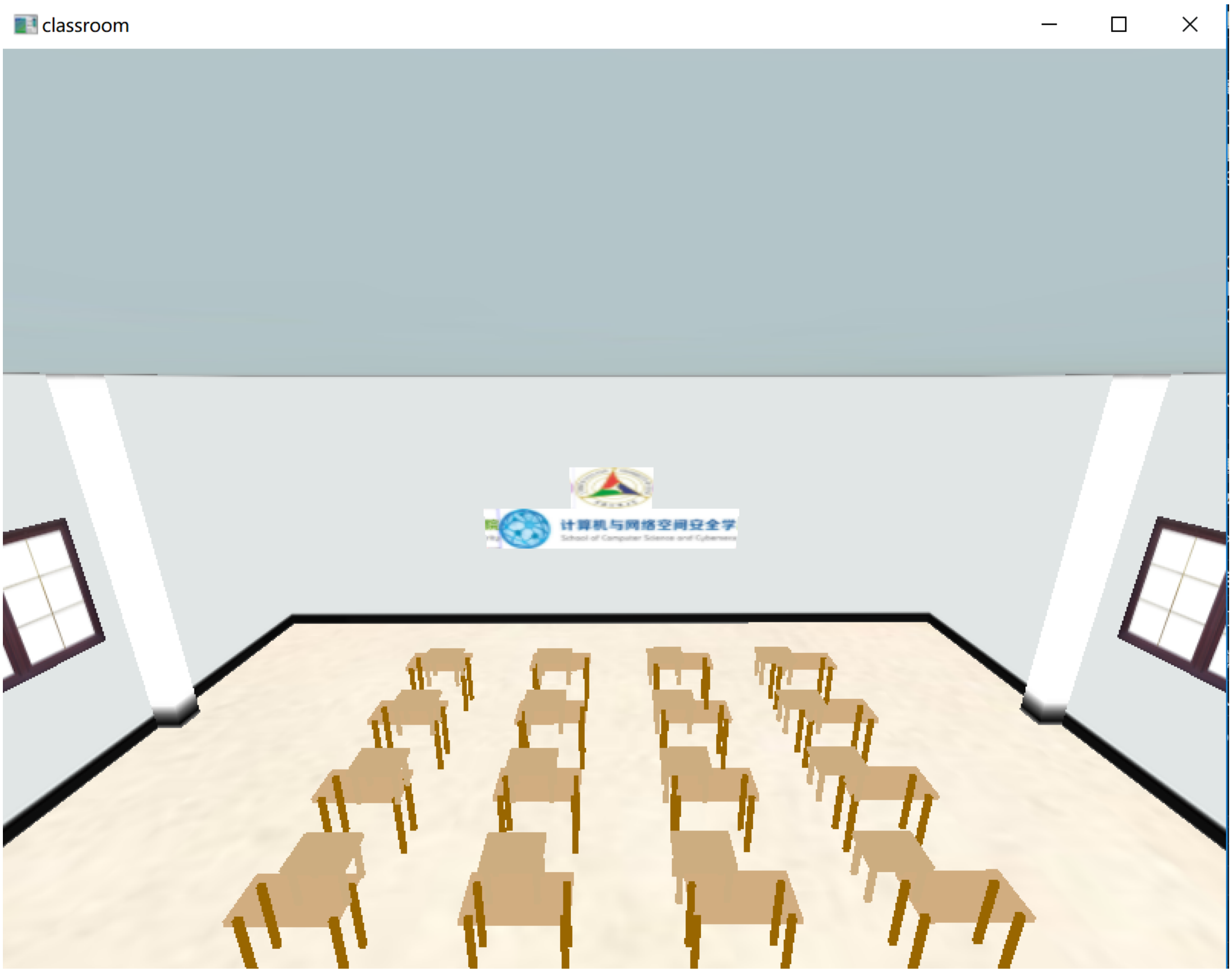
- 进入教室



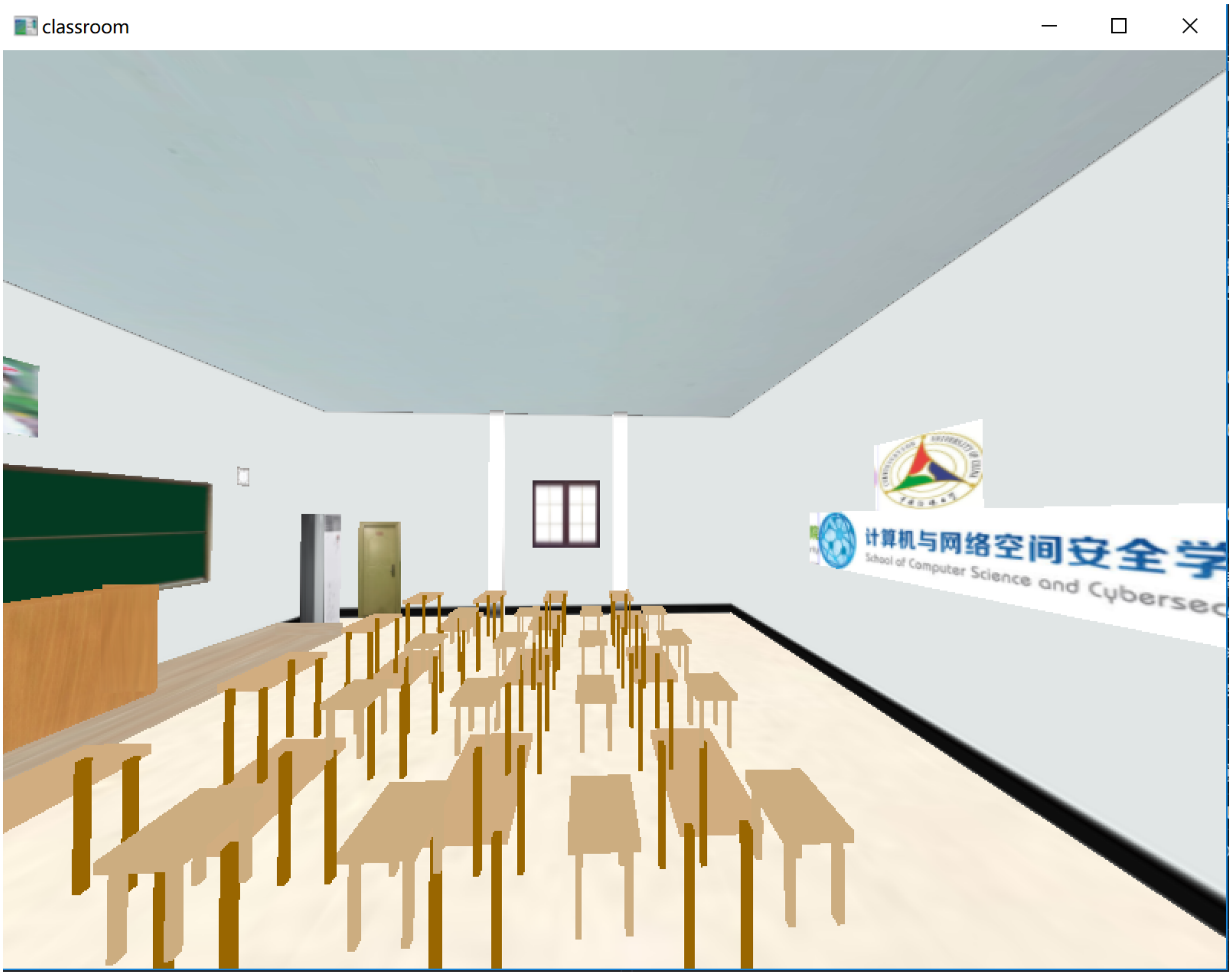
- 后方



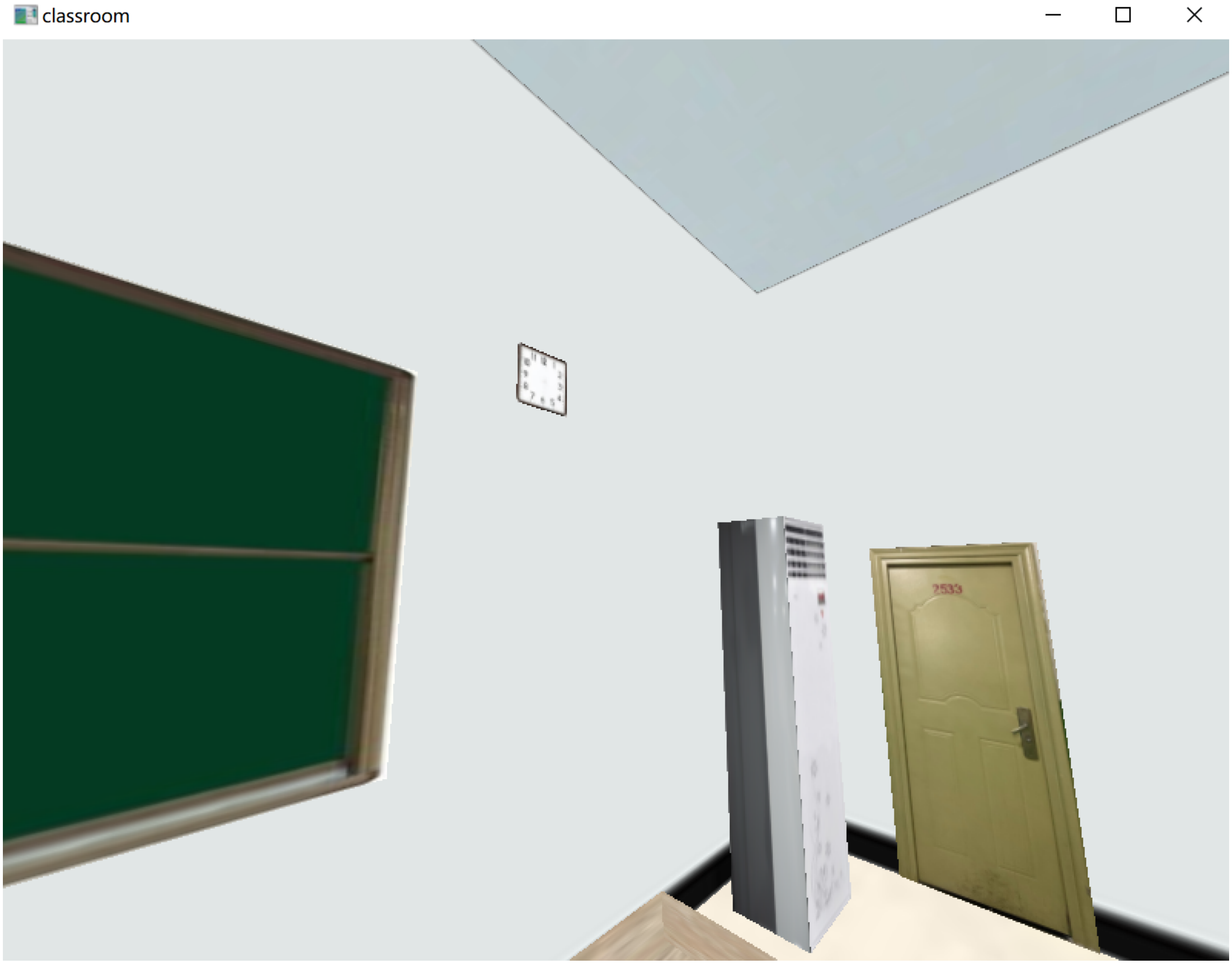
- 俯视


classroom
计算机与网络空间安全学
School of Computer Science and Cybersecu

- 教室左侧


classroom
计算机与网络空间安全学
School of Computer Science and Cybersec

- 教室一角



- 讲台纹理细节