中国地质大学（北京）

计算机图形学

实验报告

学　　院：信息工程学院

专　　业：计算机科学与技术

班　　级：10041912

学　　号：1004192118

姓　　名：陈达

联系方式：18269849385

邮　　箱：2229859481@qq.com

指导老师：严红平

日　　期：2021 年 12 月 24 日

一、实验内容

实验要求：要求构建复杂场景构建（内容自选，要求场景要符合一定的故事情节，有合理的构建搭配，包括至少两个组件，光照和纹理映射，全局运动和局部运动动画，运动形变包括三种运动：平移、旋转和缩放）。

大作业思路：开发出了一款简易的 3D 小型图形软件系统，用户通过启动该程序，可以尝试游玩该小游戏，通过右键菜单选择关闭当前“小猪”视角，可以进行全局预览，观看游戏情节外的整体景观。且由于具有一定的逻辑性，所以可以获得一些乐趣。

实现功能：

1、通过设置的 output 方法向屏幕下端打印提示文字,告知需右键点击菜单打开游戏视角即“pig”视角。

2、通过利用函数实现动画和交互效果，进而实现复杂几何模型(小猪)的平移、旋转和缩放以及其全局运动，同时通过绘制小风车实现局部相对独立运动。

3、通过绘制夜明珠、设计围墙、地面、天空、城堡以及门窗等 3D 场景实现了光照、材质和纹理映射功能。

4、通过投影变换以及模型变换等操作，实现了视点转换，可以通过键盘和鼠标等外设交互进行视角转变以及运动场景变换。

5、实现了消隐、迷雾效果，通过定义黑色迷雾实现了一定的压抑气氛，同时通过游戏情节消去迷雾，具有一定的故事性。（自学）

二、作品介绍

我实现的作品名称为《小小猪历险记》，是我结合这段时间学习到的 openGL 技术以及在 B 站和各类博客自学后的产物，我的设计思想是完成一个 3D 场景，具有一定的故事情节，最好是有一定的可玩性，当然受目前所学知识的局限，不能够完全实现所有功能，但也会在之后进一步完善好这个作品。

故事情节介绍：作品想要描述的情节如下：作为主人公的我们需要前往雪地中的城堡获取夜明珠，为此我们首先勘探了周围的环境和情况：可以看到有成片的森林，一直不停旋转的风车，还有瞭望塔，以及大片的遮挡了我们视角的迷雾，给任务带来了种种困难（想象ing），当然，最大的困难是，当我们想要进入城堡获取夜明珠时，我们才发现我们实在是太大了，无法从城堡的大门走进去。为此，我们需要变身成为小小猪，以一头小猪的视角进入到城堡当中，并进行战斗，当战斗结束后我们成功的拿到了塔顶的夜明珠，这时，天空放晴了，正义最终战胜了邪恶，阴森恐怖的迷雾也消散了，天空是那么晴朗，我门站在塔顶，眺望远处的风景，一切的努力和奋斗都显得那么值得，这时我们也可以变回主人公视角，更加宽广的看四面的风景。

功能实现简介：本作品是我结合我所学 openGL 知识以及一些封装好的库函数所实现而成，其中包含了一些基本图形的绘制、以及图形的基本变换、通过投影矩阵等操作实现 3 维场景到 2 维场景的变换、

采用纹理图片绑定等操作使得图片具有真实感、相对运动和全局运动、生活事物的绘制如小猪等等，同时通过一定的逻辑构思编写出一定的情节和交互，使得作品富有情感。

本作品实现的功能如下：

1、3D 建模：通过封装好的 openGL 函数以及中点画线等直线绘制算法绘制出了一些基本图元。

2、视角变换：通过右击菜单实现主人公视角和小猪视角的切换，进行实现不同视角观看场景的功能。

3、相对运动和全局运动：小猪通过键盘操作可以实现前进后退以及转换角度等操作，同时风车与车轮之间的旋转实现了相对运动。

4、光照的实现：通过定义在原点处的灯源实现了对各种物体的光照作用。

5、材质的实现：通过设置不同物体的材质，实现了光照下不同物体颜色的变换。

6、纹理映射的实现：通过纹理映射，使得不同物体拥有不同的纹理图样，使得场景更加丰富逼真。

7、场景变换：通过交互以及设置视角函数等操作，可以实现当前场景的变换，具有多场景等功能。

8、菜单实现：通过右键可以查看菜单，实现其余多种附加功能。

9、交互实现：通过键盘 e 键以及 f 等键可以实现物体的缩放以及迷雾的开关，通过界面提示语提示切换视角开始游戏等，

使得场景功能更富有变换。

实验环境：

操作系统：windows 10 操作系统

版本：Windows 10 家庭中文版

处理器：AMD Ryzen 5 5600H with Radeon Graphics 3.30 GHz

机带 RAM：16.0 GB (15.9 GB 可用)

系统类型 64 位操作系统，基于 x64 的处理器

编译器：Visual Studio 2019

三、代码与结果分析

（一）3D 建模

代码如下：

```
void MidPoint(int x, int y, int z) {

    glTranslatef(x, y, z);
    glColor3f(0.5, 0.5, 0.5);

    glPushMatrix();
    glScalef(0.1, 2, 1);
    glutSolidCube(10);
    glPopMatrix();

    //绘制标志
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texName + 1);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_QUADS);
```

```
    glNormal3f(-1.0f, 0.0f, 0.0f);
    glTexCoord2f(1.0f, 0.0f);
    glVertex3f(0.0f, 10.0f, 5.0f);
    glTexCoord2f(0.0f, 0.0f);
    glVertex3f(0.0f, 10.0f, -5.0f);
    glTexCoord2f(0.0f, 1.0f);
    glVertex3f(0.0f, 15.0f, -5.0f);
    glTexCoord2f(1.0f, 1.0f);
    glVertex3f(0.0f, 15.0f, 5.0f);
    glEnd();
    glDisable(GL_TEXTURE_2D);

    glColor3f(0.0, 0.0, 1.0);
    glLineWidth(3);
    glBegin(GL_LINES);
    for (int i = -3; i < 9; ++i) {
        glVertex3f(-0.5f, i, 5.0f);
        glVertex3f(-0.5f, i, -5.0f);
    }
    for (int i = -5; i < 6; ++i) {
        glVertex3f(-0.5f, 8.0f, i);
        glVertex3f(-0.5f, -3.0f, i);
    }
    glEnd();

    glColor3f(0.0, 1.0, 0.0);
    int Sx = -5;
    int Ex = 5;
    int Sy = 0;
    int Ey = 6;
    glBegin(GL_LINES);
    glVertex3f(-0.6f, Sy, Sx);
    glVertex3f(-0.6f, Ey, Ex);
    glEnd();
    glColor3f(1.0, 0.0, 0.0);
    int dx = Ex - Sx;
    int dy = Ey - Sy;
    int a = Sy - Ey;
    int b = Ex - Sx;
    int Inc_x, Inc_y; //每次增量
    if (Ex - Sx < 0) {
        b = -b;
        Inc_x = -1;
    } else
```

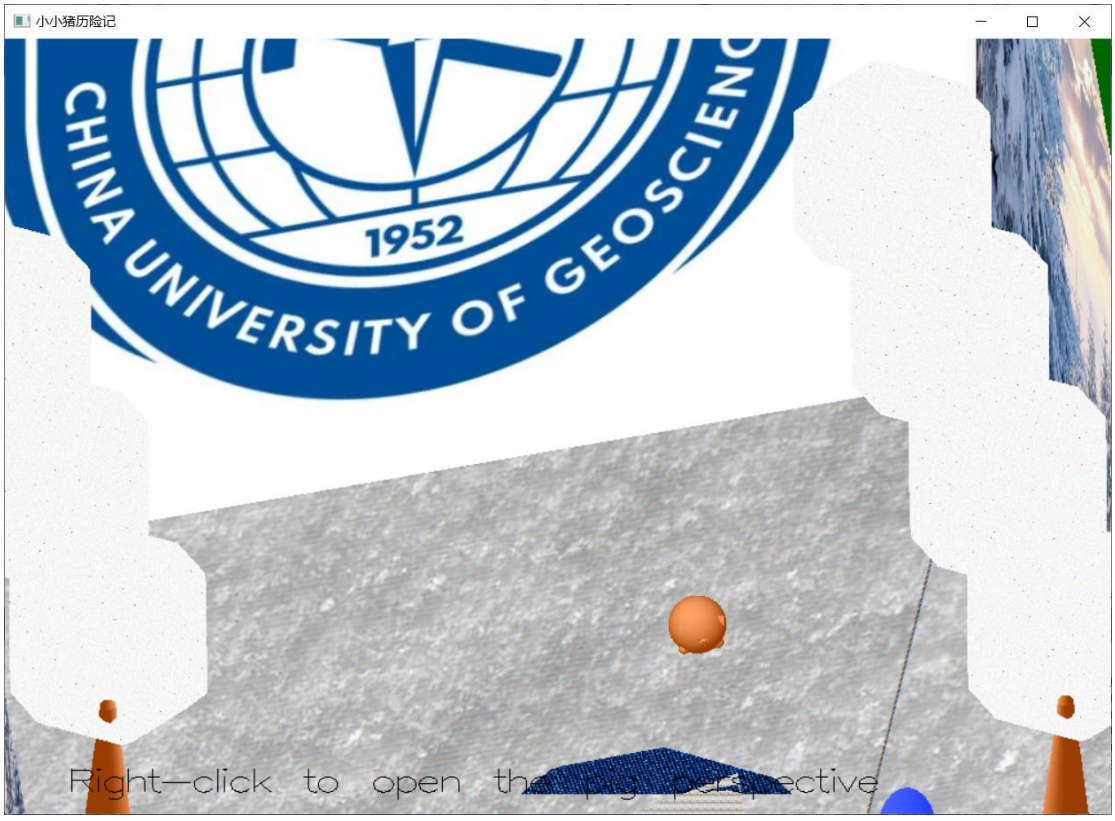
```
        Inc_x = 1;
    if (Ey - Sy < 0) {
        a = -a;
        Inc_y = -1;
    } else
        Inc_y = 1;

    int total; //总增量
    int flag;
    if (abs(dx) >= abs(dy))
        flag = 1;
    else
        flag = 0;

    if (flag)
        total = abs(Ex - Sx);
    else
        total = abs(Ey - Sy);

    int nx = Sx, ny = Sy; //当前点坐标
    glBegin(GL_POINTS);
    glVertex3f(-0.6f, ny, nx);
    glEnd();
    if (!flag) {
        // |k| > 1
        int d = a + 2 * b; //增量（化为整数）
        for (int i = 1; i <= total; i++) {
            if (d >= 0) {
                nx += Inc_x;
                ny += Inc_y;
                d += 2 * a;
                d += 2 * b;
            } else {
                ny += Inc_y;
                d += 2 * b;
            }
            glBegin(GL_POINTS);
            glVertex3f(-0.6f, ny, nx);
            glEnd();
        }
    } else {
        // |k| <= 1
        int d = 2 * a + b; //增量（化为整数）
        for (int i = 1; i <= total; i++) {
```

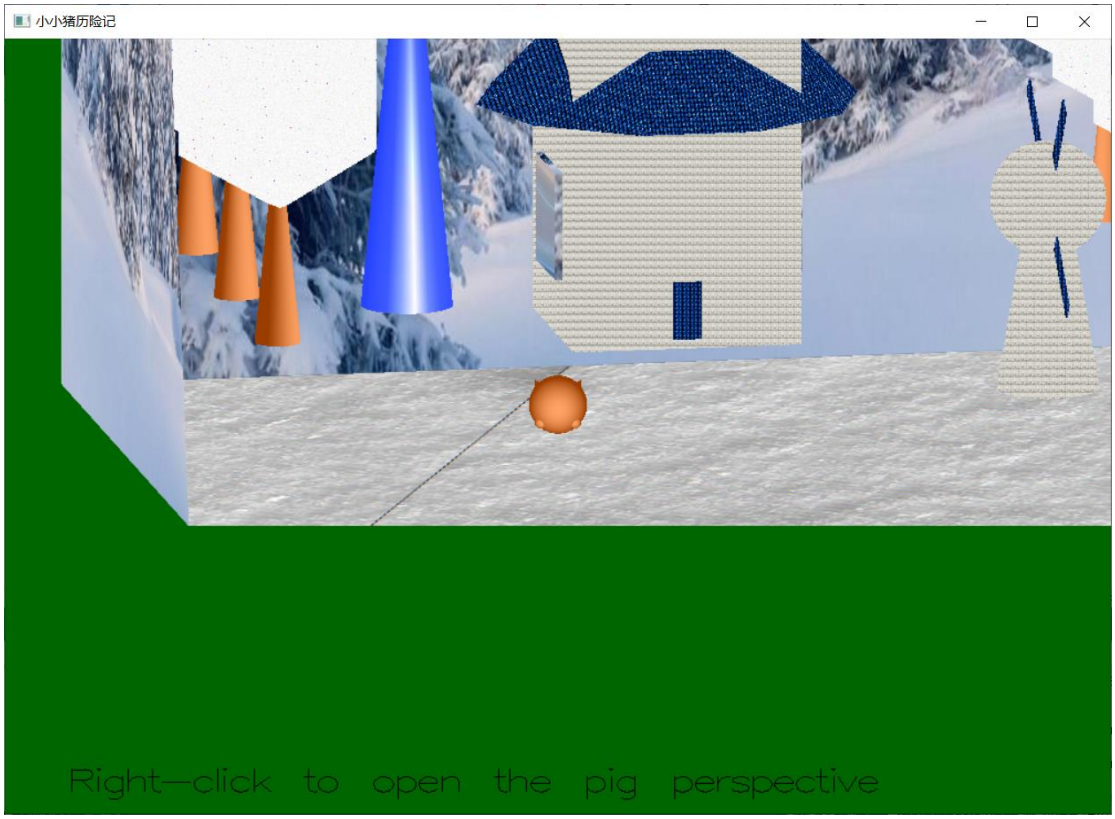
```
            if (d >= 0) {
                nx += Inc_x;
                d += 2 * a;
            } else {
                nx += Inc_x;
                ny += Inc_y;
                d += 2 * a;
                d += 2 * b;
            }
            glBegin(GL_POINTS);
            glVertex3f(-0.6f, ny, nx);
            glEnd();
        }
    }
}
void drawPig() { //画主人公小猪
    glTranslatef(ax, ay, az);
    glScalef(mx, my, mz);
    glRotatef(theta[1], 0.0, 1.0, 0.0);

    glPushMatrix();
    glTranslatef(22.5, -20.0, 18.75);
    glColor3f(1.0, 0.5, 0.5);
    glutSolidSphere(7.5f, 100, 100); //左前脚
    glPopMatrix();

    glPushMatrix();
    glColor3f(1.0, 0.5, 0.5);
    glTranslatef(-22.5, -20, 18.75);
    glutSolidSphere(7.5f, 100, 100); //右前脚
    glPopMatrix();

    glPushMatrix();
    glColor3f(1.0, 0.5, 0.5);
    glTranslatef(22.5, -20, -18.75);
    glutSolidSphere(7.5f, 100, 100); //左后脚
    glPopMatrix();

    glPushMatrix();
    glColor3f(1.0, 0.5, 0.5);
    glTranslatef(-22.5, -20, -18.75);
    glutSolidSphere(7.5f, 100, 100); //右后脚
    glPopMatrix();
```

```
    glPushMatrix();
    glColor3f(1.0, 0.5, 0.5);
    GLUquadricObj* quadratic;
    quadratic = gluNewQuadric();
    gluCylinder(quadratic, 7.5f, 7.5f, 40.5f, 50, 50); //鼻子
    glPopMatrix();

    glPushMatrix();
    glColor3f(1.0, 0.5, 0.5);
    glBegin(GL_TRIANGLES);
    glVertex3f(30.0, 28, 22.5);
    glVertex3f(28.5, 6, 22.5);
    glVertex3f(10.5, 21, 15.0);

    glVertex3f(30.0, 28, 22.5);
    glVertex3f(10.5, 21, 15.0);
    glVertex3f(18, 20, 12.0);

    glVertex3f(30.0, 28, 22.5);
    glVertex3f(28.5, 6, 22.5);
    glVertex3f(18.0, 20, 12);
    glEnd(); //左耳
    glPopMatrix();

    glPushMatrix();
    glColor3f(1.0, 0.5, 0.5);
    glBegin(GL_TRIANGLES);
    glVertex3f(-30, 28, 22.5);
    glVertex3f(-28.5, 6, 22.5);
    glVertex3f(-10.5, 21, 15.0);

    glVertex3f(-30, 28, 22.5);
    glVertex3f(-10.5, 21, 15);
    glVertex3f(-18, 20, 12);

    glVertex3f(-30, 28, 22.5);
    glVertex3f(-28.5, 6, 22.5);
    glVertex3f(-18, 20, 12);
    glEnd(); //右耳
    glPopMatrix();

    glPushMatrix();
    glColor3f(1.0, 0.8, 0.8);
    glutSolidSphere(37.5f, 100, 100);  //身体
```

```
    glPopMatrix();

    glPushMatrix();
    glColor3f(1.0, 0.5, 0.5);
    glBegin(GL_POLYGON);
    for (int i = 0; i < n; ++i)
        glVertex3f(0.06 * cos(2 * PI / n * i), 0.04 * sin(2 * PI / n * i),
0.27f);
    glEnd();//鼻子(表面)
    glPopMatrix();

    glPushMatrix();
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    for (int i = 0; i < n; ++i)
        glVertex3f(-3 + 1.5 * cos(2 * PI / n * i), 1 * sin(2 * PI / n * i),
40.5f);
    glEnd();//右鼻孔
    glPopMatrix();

    glPushMatrix();
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    for (int i = 0; i < n; ++i)
        glVertex3f(3 + 1.5 * cos(2 * PI / n * i), 1 * sin(2 * PI / n * i),
40.5f);
    glEnd();//左鼻孔
    glPopMatrix();

    glPushMatrix();
    glColor3f(0.0, 0.0, 0.0);
    glTranslatef(7.5, 8, 34.5);
    glutSolidSphere(1.95f, 1000, 1000);  //左眼
    glPopMatrix();

    glPushMatrix();
    glColor3f(0.0, 0.0, 0.0);
    glTranslatef(-7.5, 8, 34.5);
    glutSolidSphere(1.95f, 1000, 1000);  //右眼
    glPopMatrix();
}
```

通过以上代码结构绘制出了一个三维小猪形象如下：


小小猪历险记
CHINA UNIVERSITY OF GEOSCIENCES
1952
Right—click to open the pig perspective


小小猪历险记
Right—click to open the pig perspective

（二）视角变换

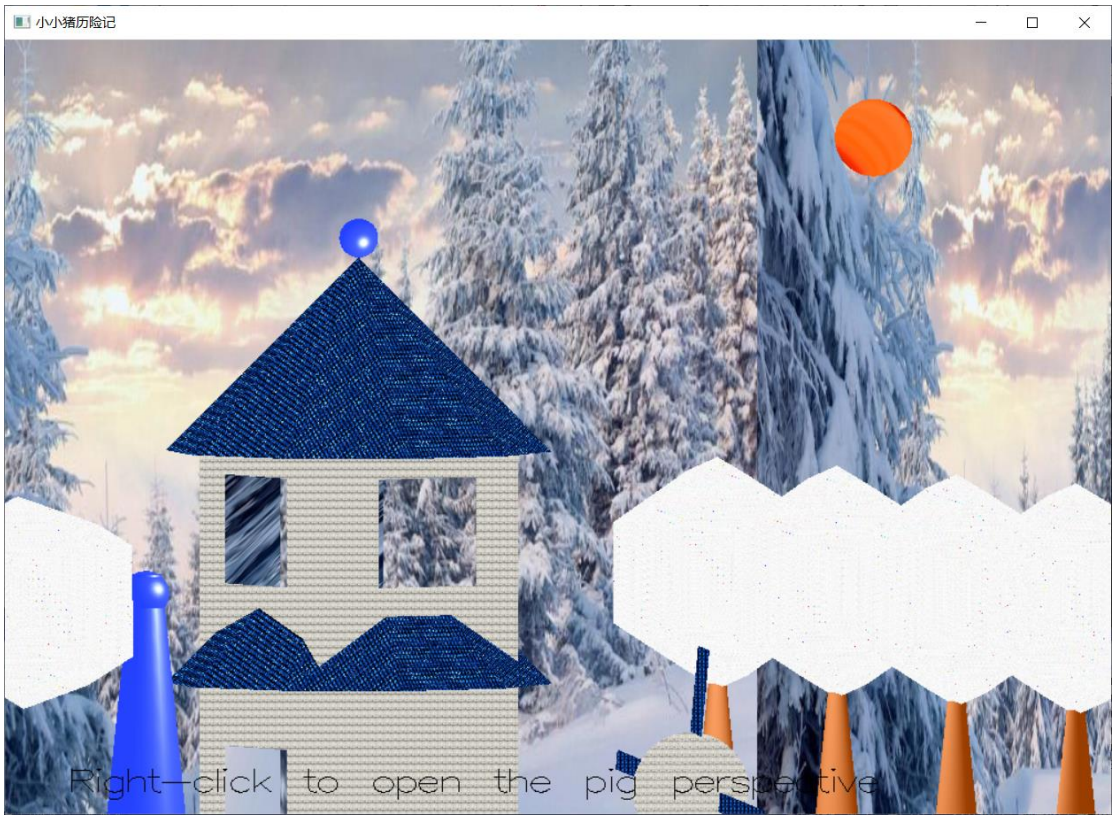
代码如下：

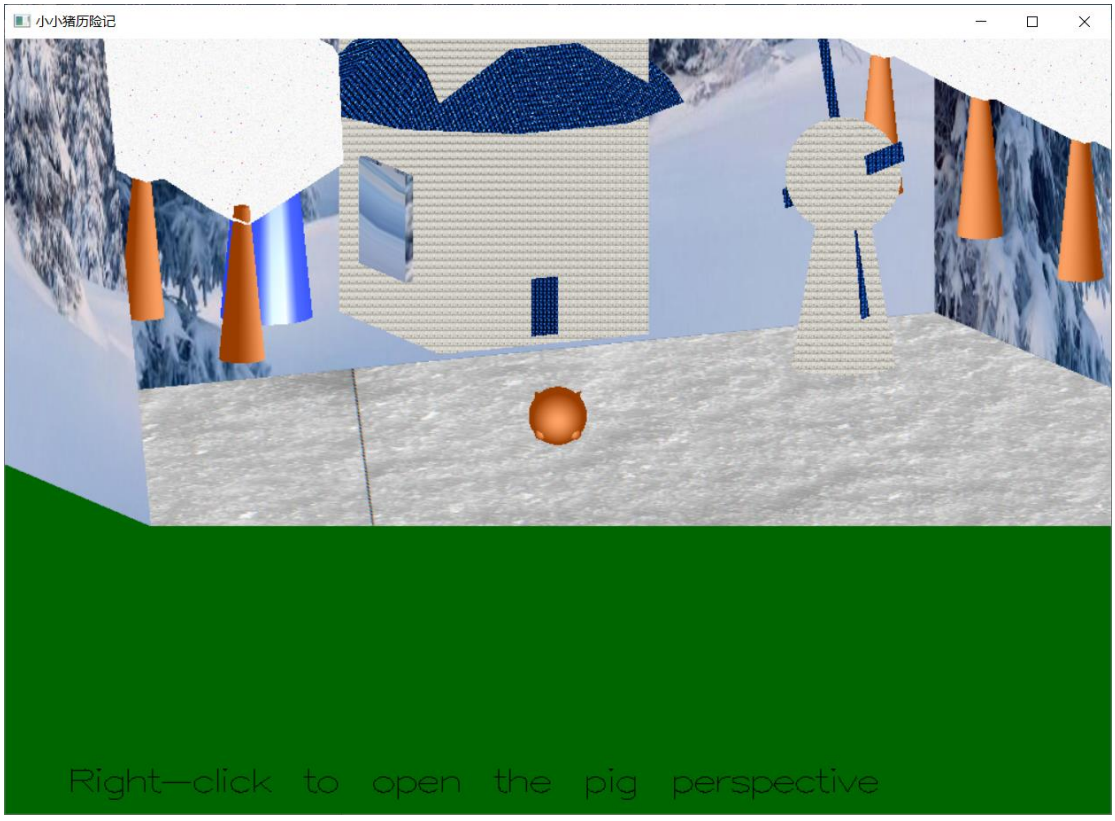
```
//各类变量的定义
int view = 0;
float eye_x[3] = { 0.0, 100.0, 0.0 };              //当前摄像位置
float eye_y[3] = { 0.0, 100.0, 0.0 };
float eye_z[3] = { 0.0, 0.0, 0.0 };
float fx[3] = { 0.0, 0.0, 0.0 };                //观察对象位置
float fy[3] = { 0.0, 0.0, 0.0 };
float fz[3] = { 0.0, 0.0, 0.0 };
float mx = 0.2f;                            //真实小猪与原小猪缩放倍数
float my = 0.2f;
float mz = 0.2f;
float ax = 50.0f;                           //小猪的初始坐标
float ay = -125.0f;
float az = -50.0f;
float theta[3] = { 0.0, 0.0, 0.0 };       //小猪每个轴的旋转角
void SubMenu(GLint Option) {
    switch (Option) {
        case 1:
            MOUSE_MOVE = 0; //此刻无法进行鼠标预览移动了
            view = 1;
            eye_x[view] = ax + 0.1 * sin(theta[1] * PI / 180);
            eye_y[view] = ay + 0.3;
            eye_z[view] = az + 0.01 * sin(theta[1] * PI / 180);

            fx[view] = eye_x[view] + sin(theta[1] * PI / 180);
            fy[view] = eye_y[view] - 0.2;
            fz[view] = eye_z[view] + cos(theta[1] * PI / 180);
            glutPostRedisplay();
            break;
        case 2:

            view = 2;
            get_pearl = 0;
            glutPostRedisplay();
            break;
        case 3:
            On = GL_TRUE;
            glutPostRedisplay();
```

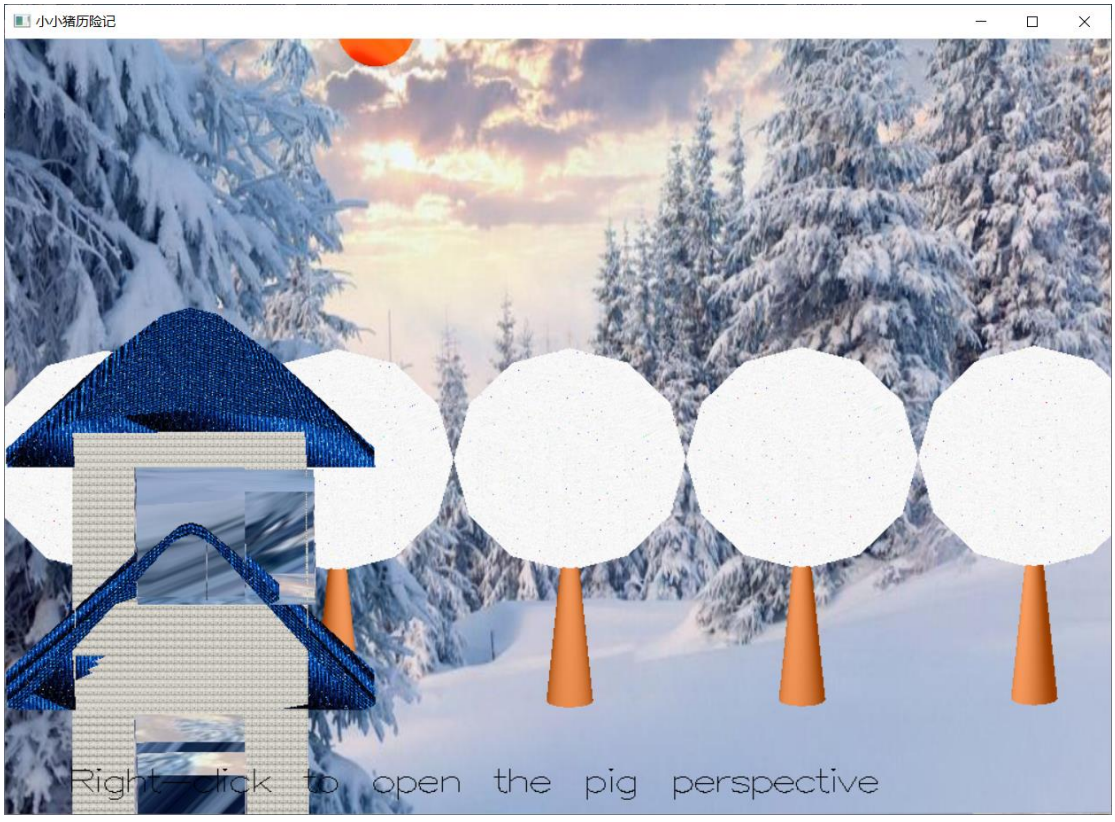
```
            break;
        case 4:
            On = GL_FALSE;
            glutPostRedisplay();
            break;
        default:
            break;
    }
    glutPostRedisplay();
}
```

结果分析：

通过以上代码结构进行视角的转换，可以看到从小猪视角可以切换到全局视角等。




小小猪历险记
Right—click to open the pig perspective


小小猪历险记
Right—click to open the pig perspective

（三）相对运动和全局运动

代码如下：

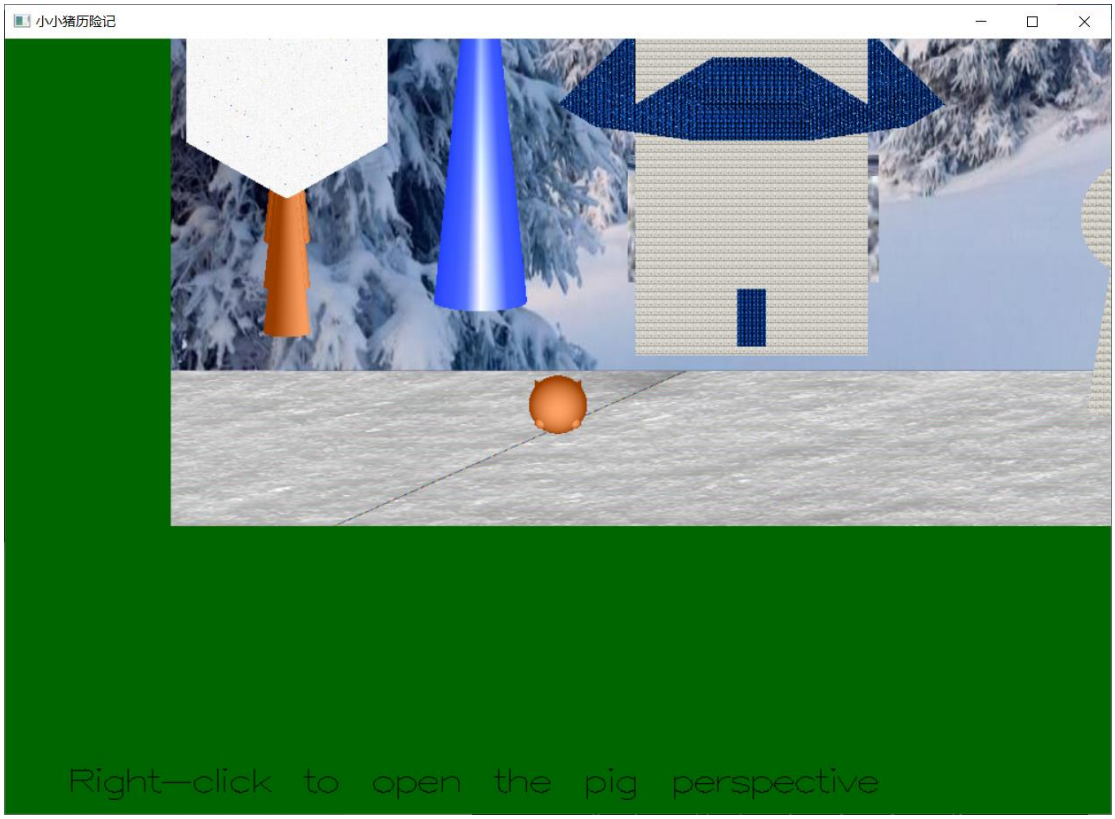
```
//-----------------------------------------
//-------------------实现键盘人机交互------------
void key(unsigned char key, int x, int y) {
    switch (key) {
        //向前走
        case 'w': {
            if (KEY_MOVE == 1 && view == 1) {
                ax += 5 * sin(theta[1] * PI / 180);
                az += 5 * cos(theta[1] * PI / 180);
                if (az > 70 && in_door == 0) {
                    in_door = 1;
                    fog = 0;
                    view = 3;
                    MOUSE_MOVE = 1;
                    get_pearl = 1;//已获取夜明珠
                    Light_change = 1;

                }
            }
            break;
        }
        //向后走
        case 's': {
            if (KEY_MOVE == 1 && view == 1) {
                ax -= 5 * sin(theta[1] * PI / 180);
                az -= 5 * cos(theta[1] * PI / 180);
                if (az > 70 && in_door == 0) {
                    in_door = 1;
                    fog = 0;
                    view = 3;
                    MOUSE_MOVE = 1;
                    get_pearl = 1;//已获取夜明珠
                    Light_change = 1;
                }
            }
            break;
        }
        //向左走
        case 'a': {
```

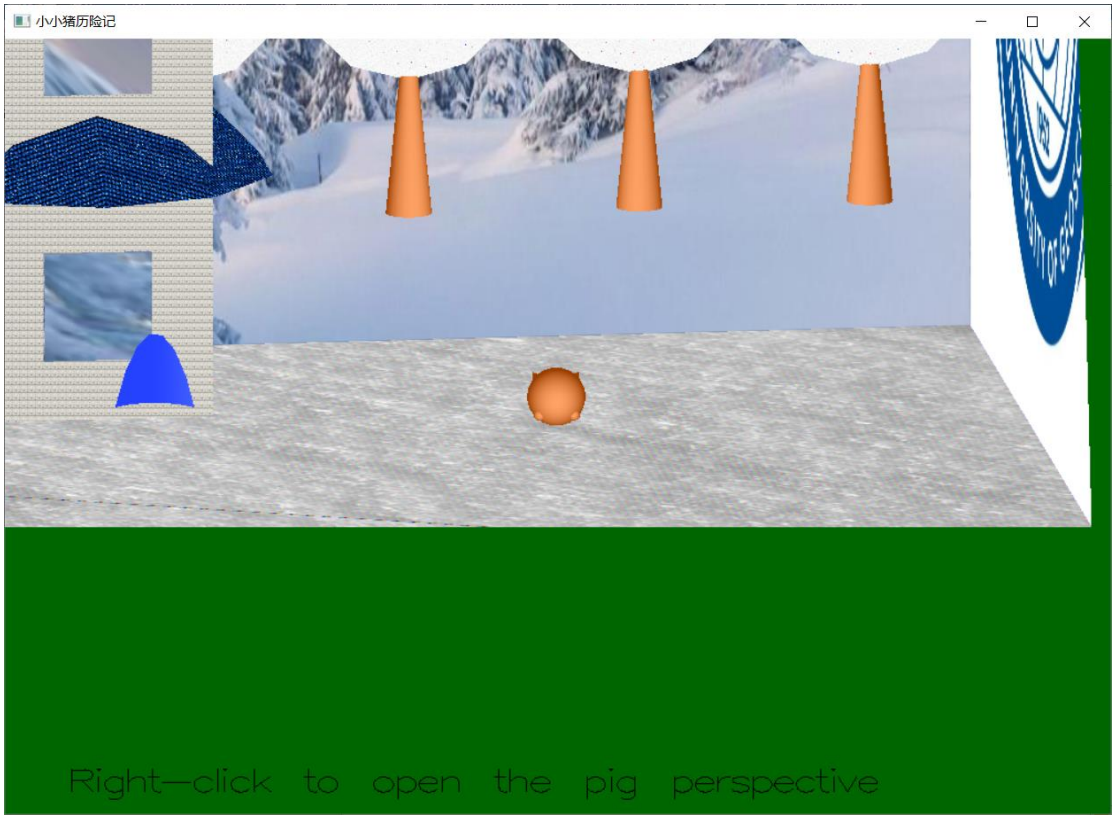
```
            if (KEY_MOVE == 1 && view == 1) {
                axis = 1;
                theta[axis] += 5.0;
                if (theta[axis] > 360.0)
                    theta[axis] -= 360.0;
                eye_x[1] = (eye_x[1] - ax) * cos(5 * PI / 180) + (eye_z[1]
- az) * sin(5 * PI / 180) + ax;
                eye_z[1] = -(eye_x[1] - ax) * sin(5 * PI / 180) + (eye_z[1]
- az) * cos(5 * PI / 180) + az;
                fx[1] = (fx[1] - ax) * cos(5 * PI / 180) + (fz[1] - az) *
sin(5 * PI / 180) + ax;
                fz[1] = -(fx[1] - ax) * sin(5 * PI / 180) + (fz[1] - az) *
cos(5 * PI / 180) + az;
            }
            break;
        }
        //向右走
        case 'd': {
            if (KEY_MOVE) {
                axis = 1;
                theta[axis] -= 5.0;
                if (theta[axis] > 360.0)
                    theta[axis] -= 360.0;
                eye_x[1] = (eye_x[1] - ax) * cos(5 * PI / 180) - (eye_z[1]
- az) * sin(5 * PI / 180) + ax;
                eye_z[1] = (eye_x[1] - ax) * sin(5 * PI / 180) + (eye_z[1]
- az) * cos(5 * PI / 180) + az;
                fx[1] = (fx[1] - ax) * cos(5 * PI / 180) - (fz[1] - az) *
sin(5 * PI / 180) + ax;
                fz[1] = (fx[1] - ax) * sin(5 * PI / 180) + (fz[1] - az) *
cos(5 * PI / 180) + az;
            }
            break;
        }
        case 'e': {
            mx -= 0.01;
            my -= 0.01;
            mz -= 0.01;
            break;
        }
        case 'f': {
            if (fog == 1) {
                fog = 0;
            } else {
```

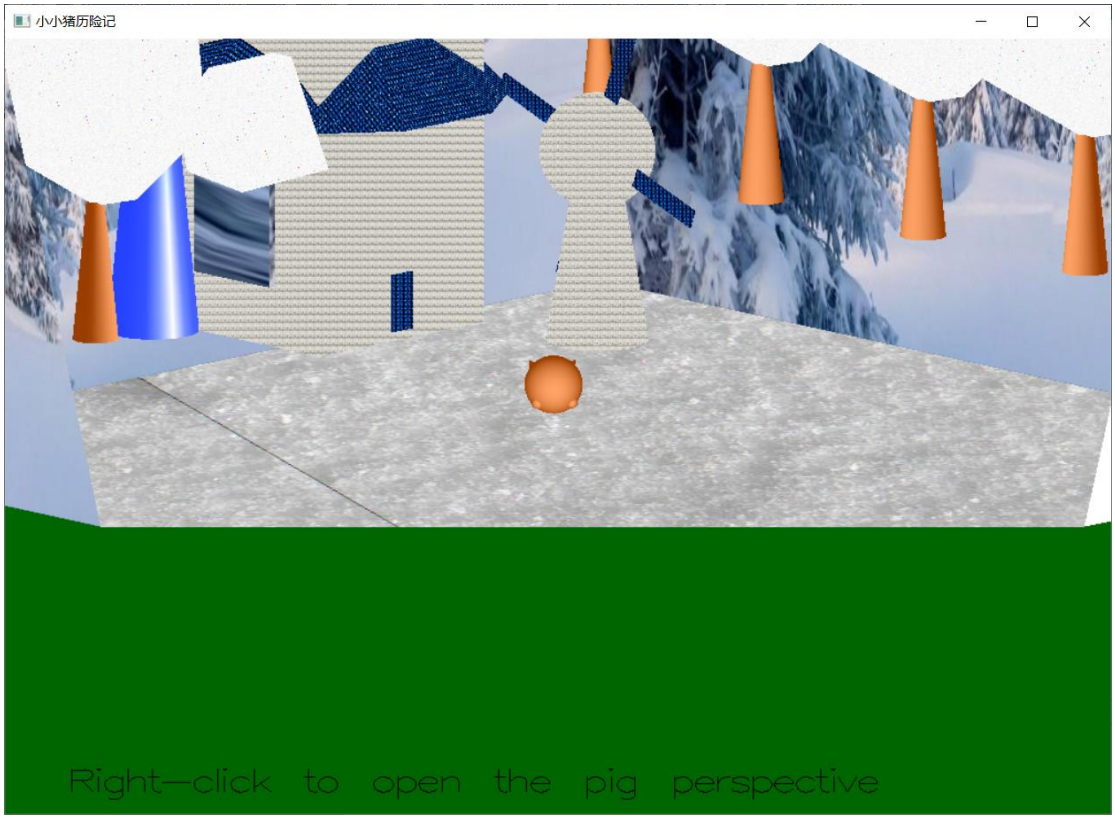
```
                fog = 1;
            }
            break;
        }
    }
}
```

结果分析：

可以看到通过定义的键盘等操作，可以实现小猪的移动，旋转调换方向，以及风车转轮的相对运动等等。




小小猪历险记
Right—click to open the pig perspective


小小猪历险记
Right—click to open the pig perspective

（四）光照的实现

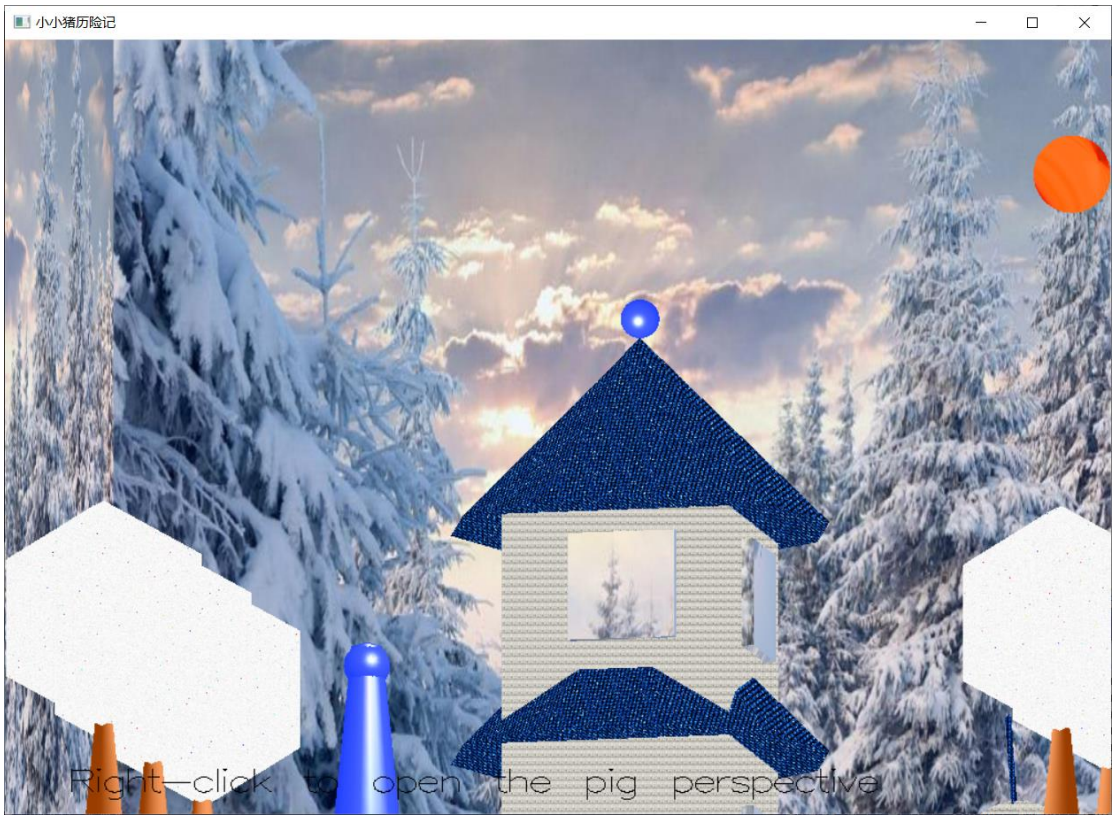
代码如下：

```
//-----------------------------------------------------------------
//---------------------开始光照操作---------------------------------
void loadlight() {
    //默认全局光照为低强度白光

    GLfloat light_ambient[] = { 1.0f, 1.0f, 1.0f, 1.0f };//环境光
    GLfloat light_diffuse[] = { 0.5f, 0.5f, 0.5f, 1.0f };//漫射光    可以将其设置成棕色，映照树干 但水的颜色会变化
    GLfloat light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };//镜面光
    GLfloat light_position[] = { 0.0f, 0.0f, 0.0f, 1.0 }; //xyz 指定光源位置 sun    离场景较近的光源
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);//指定 0 号光源的以上参数
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
}

void tree_material() { //用于画光照树
    GLfloat mat_ambient[] = { 0.5, 0.2, 0.0, 1.0 };
    GLfloat mat_diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
    GLfloat mat_specular[] = { 0.0, 0.0, 0.0, 1.0 };
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);// 材质的环境颜色
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);//材质的散射颜色
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);//材质的镜面反射颜色
}

void pearl_ball(int x, int y, int z) { //用于画塔顶的夜明珠
    glTranslatef(x, y, z);
    glColor3f(0.57f, 0.48f, 0.24f);
    GLfloat mat_ambient[] = { 0.12, 0.22, 0.92, 1.0 };  //材质的环境颜色
    GLfloat mat_diffuse[] = { 0.28, 0.26, 0.81, 1.0 };  //材质的散射颜色
    GLfloat mat_specular[] = { 0.95, 0.94, 0.82, 1.0 }; //材质的镜面反射颜色
    GLfloat mat_shininess[] = { 28.0 };                 //镜面反射指数
```

```
    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);                    //
材质属性中的环境光
    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);                    //
材质属性中的散射光
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);                  //
材质属性中的镜面反
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);                //
材质属性中的发射

    glutSolidSphere(5, 80, 80);
}
```

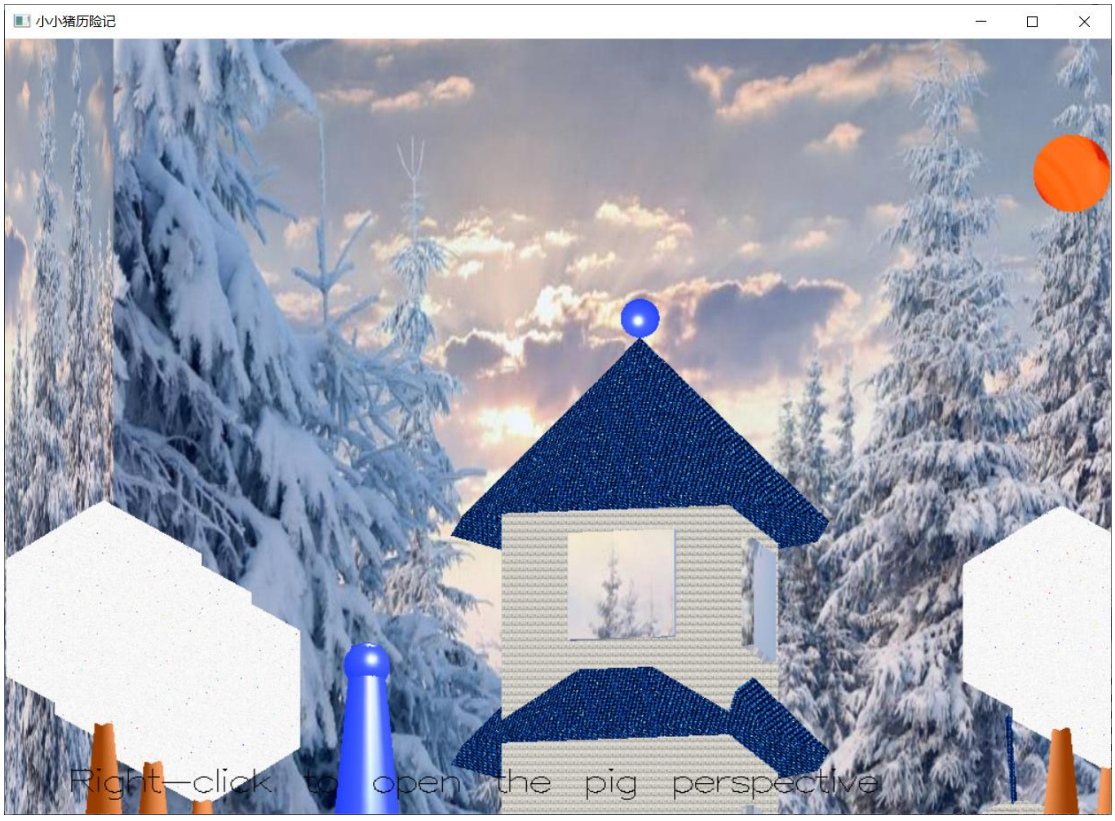
结果分析：

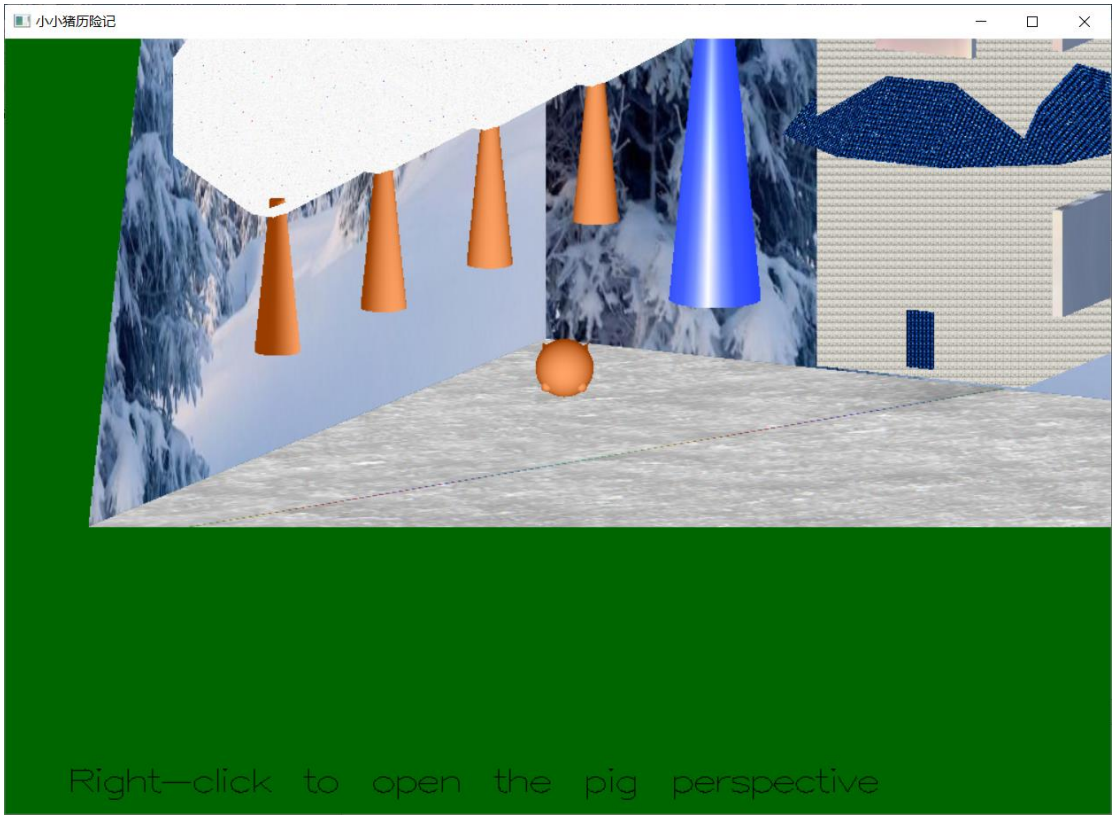
可以看到通过定义的光照以及材质等操作，实现了塔顶夜明珠的光照变换，具有光照的性质。



（五）材质的实现

代码如下：


小小猪历险记
Right—click to open the pig perspective


小小猪历险记
Right—click to open the pig perspective

（六）纹理映射的实现

代码如下：

```
static GLuint texName;                    //纹理组的名称
int get_pearl = 0;                        //是否获取到夜明珠
int Light_change = 0;                     //灯源更替
//依序为太阳、侧面、地面
GLuint drawwalllist, sunlist, rooflist, treelist;//贴环境纹理的显示列表
GLUquadricObj* qobj;
float ctrlpoints[4][4][3];
float texpts[2][2][2] = { {{0.7, 0.7}, {0.7, 1}},
    {{1, 0.7}, {1, 1}}
};
//-------------------------------------------------------------------------
----
//-----------------------开始纹理操作-----------------------------------
void loadtexture() {
    //侧面的纹理
    glGenTextures(8, &texName);//生成纹理的数量
    glBindTexture(GL_TEXTURE_2D, texName);
    {
        Image* image = loadBMP("sun.bmp");//太阳的纹理
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_REPEAT);//REPEAT 为重复模式 如用 CLAMP 截断模式则会出现单色多块填充
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height,
0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);//几维 多分辨率层数 纹理元素的
哪些分量被使用 纹理的宽 高 边框大小  图像的数据格式 数据类型 内存中指定的纹理图
像数据
    }
    {
        Image* image = loadBMP("house.bmp"); //风车的纹理
        glBindTexture(GL_TEXTURE_2D, texName + 1);
        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_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height,
0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
    }
```

```
    {
        Image* image = loadBMP("floor.bmp"); //地面的纹理
        glBindTexture(GL_TEXTURE_2D, texName + 2);
        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_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height,
0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);//纹理映射函数 定义一个二维的纹
理图
    }
    {
        Image* image = loadBMP("wall.bmp");//空间的纹理
        glBindTexture(GL_TEXTURE_2D, texName + 3);
        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_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height,
0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
    }
    {
        Image* image = loadBMP("roof.bmp");//屋顶的纹理
        glBindTexture(GL_TEXTURE_2D, texName + 4);
        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_MAG_FILTER,
GL_LINEAR);//使用 2x2 的纹理元素列阵的加权线性平均值贴图 比 NEAREST 的慢，但质
量高
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height,
0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
    }
    {
        Image* image = loadBMP("tree.bmp");//树的纹理
        glBindTexture(GL_TEXTURE_2D, texName + 5);
        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_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height,
0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
    }
    {
```

```
        Image* image = loadBMP("cugb.bmp");//cugb 墙的纹理
        glBindTexture(GL_TEXTURE_2D, texName + 6);
        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_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height,
0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
    }
    {
        Image* image = loadBMP("sky.bmp");//天空的纹理
        glBindTexture(GL_TEXTURE_2D, texName + 7);
        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_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height,
0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
    }
}
void drawWalls() {

    drawwalllist = glGenLists(1);
    glNewList(drawwalllist, GL_COMPILE);
    glDisable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);

    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);//设定纹理坐
标环境参数
    glBindTexture(GL_TEXTURE_2D, texName + 6);

    glBegin(GL_QUADS);
    //绘制 cugb 那面墙
    glTexCoord2f(0.0, 0.0);
    glVertex3f(-150, -150, -150);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(150, -150, -150);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(150, 150, -150);
    glTexCoord2f(0.0, 1.0);
    glVertex3f(-150, 150, -150);
    glEnd();
```

```
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);//设定纹理坐
标环境参数
    glBindTexture(GL_TEXTURE_2D, texName + 7);
    glBegin(GL_QUADS);
    //绘制天空
    glTexCoord2f(0.5, 0.5);
    glVertex3f(-150, 150, -150);//截一段用作顶部的天空
    glTexCoord2f(0.5, 1.0);
    glVertex3f(150, 150, -150);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(150, 150, 150);
    glTexCoord2f(1.0, 0.5);
    glVertex3f(-150, 150, 150);
    glEnd();

    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);//设定纹理坐
标环境参数
    glBindTexture(GL_TEXTURE_2D, texName + 3);

    glBegin(GL_QUADS);

    glTexCoord2f(0.0, 1.0);
    glVertex3f(150, 150, -150);
    glTexCoord2f(0.0, 0.0);
    glVertex3f(150, -150, -150);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(150, -150, 150);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(150, 150, 150);

    glTexCoord2f(0.0, 1.0);
    glVertex3f(150, 150, 150);
    glTexCoord2f(0.0, 0.0);
    glVertex3f(150, -150, 150);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(-150, -150, 150);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(-150, 150, 150);

    glTexCoord2f(1.0, 1.0);
    glVertex3f(-150, 150, -150);
    glTexCoord2f(0.0, 1.0);
    glVertex3f(-150, 150, 150);
    glTexCoord2f(0.0, 0.0);
```

```
    glVertex3f(-150, -150, 150);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(-150, -150, -150);
    glEnd();

    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);//设定纹理坐
标环境参数
    glBindTexture(GL_TEXTURE_2D, texName + 2);//地面
    glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0);
    glVertex3f(150, -150, -150);
    glTexCoord2f(1.0, 0.0);
    glVertex3f(-150, -150, -150);
    glTexCoord2f(1.0, 1.0);
    glVertex3f(-150, -150, 150);
    glTexCoord2f(0.0, 1.0);
    glVertex3f(150, -150, 150);
    glEnd();

    glDisable(GL_TEXTURE_2D);
    glEnable(GL_LIGHTING);

    glEndList();

}

//------------------------------------------------------------
//--------------------绘制太阳及其纹理---------------------
void sun() {
    sunlist = glGenLists(1);//赋予空的相邻的显示列表，即初始化列表
    glNewList(sunlist, GL_COMPILE);

    glEnable(GL_TEXTURE_GEN_S);//启动 S 坐标的自动生成
    glEnable(GL_TEXTURE_GEN_T);
    glEnable(GL_TEXTURE_2D);//启用纹理

    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);//控制纹理坐标的生
成
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);//GL_SPHERE_MAP
这个模式是在平面上模拟了球面的反射效果

    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);//设定纹理坐
标环境参数
```

```
    glBindTexture(GL_TEXTURE_2D, texName);//建立一个绑定到目标纹理的有名称的纹理

    gluSphere(qobj, 10, 30, 30);//绘制一个球体

    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);
    glEndList();
}
//-------------------------------------------------------------
//--------------------------绘制房顶及其纹理---------------
void roof() {
    rooflist = glGenLists(1);//赋予空的相邻的显示列表
    glNewList(rooflist, GL_COMPILE);

    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
    glEnable(GL_TEXTURE_2D);//启用纹理

    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);//GL_OBJECT_LINEAR 此模式下生成的坐标相当于顶点坐标到特定平面的距离
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);

    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    glBindTexture(GL_TEXTURE_2D, texName + 4);//建立一个绑定到目标纹理的有名称的纹理
    glTranslatef(0, 0, -60);
    glutSolidCone(50, 50, 10, 10);//绘制实心圆锥体
    glTranslatef(0, 0, 60);
    glutSolidCone(50, 50, 10, 10);//再绘制一个


    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);

    glDisable(GL_TEXTURE_2D);

    glEndList();
}

//-------------------------------------------------------
//--------------------绘制几棵小树----------------------
```

```
void tree() {
    treelist = glGenLists(1);//赋予空的相邻的显示列表
    glNewList(treelist, GL_COMPILE);
    glPopMatrix();

    glPushMatrix();
    tree_material();//环境，漫射，镜面反射
    glTranslatef(0, 10, 0);
    glRotatef(-90, 1, 0, 0);
    gluCylinder(qobj, 6, 2, 40, 20, 10);//画圆柱
    glPopMatrix();
    glPushMatrix();
    glTranslatef(0, 75, 0);

    glNewList(sunlist, GL_COMPILE);


    glEnable(GL_TEXTURE_GEN_S);//启动 S 坐标的自动生成
    glEnable(GL_TEXTURE_GEN_T);
    glEnable(GL_TEXTURE_2D);//启用纹理

    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);//控制纹理坐标
的生成
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE,
GL_OBJECT_LINEAR);//GL_SPHERE_MAP 这个模式是在平面上模拟了球面的反射效果

    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);//设定纹理坐
标环境参数
    glBindTexture(GL_TEXTURE_2D, texName + 5);//建立一个绑定到目标纹理的有
名称的纹理

    gluSphere(qobj, 30, 6, 6);//绘制一个球体
    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);


    glDisable(GL_TEXTURE_2D);


    glEndList();
}
void display() {
    glDepthFunc(GL_LESS);   //深度测试 深度缓冲区对照,z 值小被显示
    glClearDepth(0.25);//深度大于数值的多边形不会被显示
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // Clear display
window.颜色与深度  填充窗口
    glMatrixMode(GL_MODELVIEW);//指定当前矩阵 模型变换矩阵

    drawfog();  //雾化效果
    glLoadIdentity();//重置为单位矩阵
    gluLookAt(eye_x[view], eye_y[view], eye_z[view], fx[view], fy[view],
fz[view], 0, 1, 0);

    glPushMatrix();//矩阵压栈

    glMatrixMode(GL_PROJECTION);//指定当前矩阵 投影矩阵
    glLoadIdentity();//重置为单位矩阵

    if (windoww <= windowh)
        glOrtho(-100, 100, -100 * (GLfloat)windowh / (GLfloat)windoww,
                100 * (GLfloat)windowh / (GLfloat)windoww, 0.001, 290.0);
    else
        glOrtho(-100 * (GLfloat)windoww / (GLfloat)windowh,
                100 * (GLfloat)windoww / (GLfloat)windowh, -100, 100, 0.001,
290.0);


    glMatrixMode(GL_MODELVIEW);//指定当前矩阵
    glPopMatrix();//出栈
    glPushMatrix();
    //----------绘制环境-------------------
    glCallList(drawwalllist);
    //----------绘制太阳---------------------
    glTranslatef(-125, 70, 50);//矩阵相乘
    glCallList(sunlist);
    glPopMatrix();



    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
    glEnable(GL_TEXTURE_2D);//启用纹理
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);//控制纹理坐标的生
成
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);//GL_EYE_LINEAR
与 GL_OBJECT_LINEAR 模式具有类似的纹理生成函数
    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);//设定纹理坐
标环境参数
```

```
    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, texName + 1);
    glTranslatef(0, -100, 100);
    glutSolidCube(60.0);//绘制塔的墙面立方体结构
    glTranslatef(0, 60, 0);
    glutSolidCube(60.0);//绘制塔的第二层墙面
    glPopMatrix();

    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);//关闭纹理


    glPushMatrix();
    glTranslatef(0, -10, 100);
    glRotatef(-90, 1, 0, 0);//旋转
    glCallList(rooflist);//绘制屋顶圆锥结构
    glPopMatrix();



    glEnable(GL_TEXTURE_GEN_S);
    glEnable(GL_TEXTURE_GEN_T);
    glEnable(GL_TEXTURE_2D);//启用纹理
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);//控制纹理坐标的生
成
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);//设定纹理坐
标环境参数

    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, texName + 4);
    glTranslatef(0, -120, 70);
    glScalef(0.5, 1, 0.1);
    glutSolidCube(15.0);//绘制一层的门结构
    glPopMatrix();

    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);


    glEnable(GL_TEXTURE_GEN_S);//启动 S 坐标的自动生成
    glEnable(GL_TEXTURE_GEN_T);
    glEnable(GL_TEXTURE_2D);//启用纹理
```

```
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);//控制纹理坐标的生成
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);//GL_SPHERE_MAP这个模式是在平面上模拟了球面的反射效果
    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);//设定纹理坐标环境参数

    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, texName + 3);
    glTranslatef(18, -100, 100);
    glutSolidCube(28.0);//绘制窗户结构
    glTranslatef(-37, 0, 0);
    glutSolidCube(28.0);//一层右边
    glTranslatef(0, 70, 0);
    glutSolidCube(28.0);//二层右边
    glTranslatef(37, 0, 0);
    glutSolidCube(28.0);//二层左边
    glTranslatef(-20, 0, -18);
    glutSolidCube(28.0);//二层前面
    glPopMatrix();

    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);

    //------绘制风车------------
    glEnable(GL_TEXTURE_GEN_S);//启动 S 坐标的自动生成
    glEnable(GL_TEXTURE_GEN_T);
    glEnable(GL_TEXTURE_2D);//启用纹理
    glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);//控制纹理坐标的生成
    glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_EYE_LINEAR);
    glEnable(GL_TEXTURE_2D);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);//设定纹理坐标环境参数

    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, texName + 1);
    glTranslatef(-100, -100, 100);
    glutSolidSphere(15, 100, 100);   //绘制实心球
    glRotatef(90, 1, 0, 0);
    gluCylinder(qobj, 5, 15, 60, 100, 100);   //圆锥 下上面的半径 高度 经纬线（细致程度）
    glPopMatrix();
```

```
    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, texName + 4);
    glTranslatef(-100, -100, 100);
    if (On == GL_TRUE) {
        xz += 10;
        if (xz == 360) xz = 0;
    }
    glRotatef(xz, 1, 0, 0);
    glScalef(0.01, 1.5, 0.1); //缩放
    glutSolidCube(50);   //实心立方体
    glPopMatrix();

    glPushMatrix();
    glBindTexture(GL_TEXTURE_2D, texName + 4);
    glTranslatef(-100, -100, 100);
    glRotatef(xz, 1, 0, 0);
    glScalef(0.01, 0.1, 1.5);
    glutSolidCube(50);
    glPopMatrix();

    glDisable(GL_TEXTURE_GEN_S);
    glDisable(GL_TEXTURE_GEN_T);


    //-------tree-----------
    glPushMatrix();
    glTranslatef(120, -120, 200);//从左上到右上
    glPushMatrix();
    glCallList(treelist);
    glPopMatrix();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(120, -120, 120);
    glPushMatrix();
    glCallList(treelist);
    glPopMatrix();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(120, -120, 60);
    glPushMatrix();
    glCallList(treelist);
```

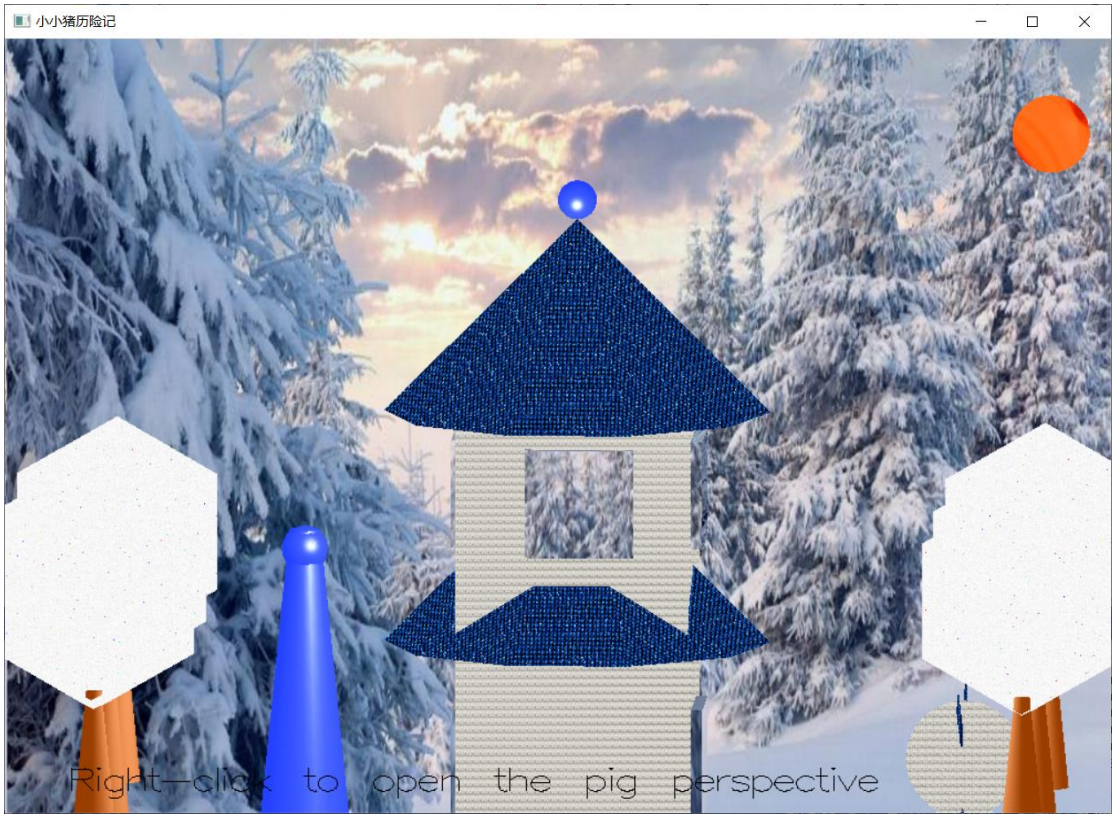
```
    glPopMatrix();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(120, -120, 0);
    glPushMatrix();
    glCallList(treelist);
    glPopMatrix();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(120, -120, -60);
    glPushMatrix();
    glCallList(treelist);
    glPopMatrix();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(120, -120, -120);
    glPushMatrix();
    glCallList(treelist);
    glPopMatrix();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-120, -120, -120);
    glPushMatrix();
    glCallList(treelist);
    glPopMatrix();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-120, -120, -60);
    glPushMatrix();
    glCallList(treelist);
    glPopMatrix();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-120, -120, 0);
    glPushMatrix();
    glCallList(treelist);
    glPopMatrix();
    glPopMatrix();
```

```
    glPushMatrix();
    glTranslatef(-120, -120, 60);
    glPushMatrix();
    glCallList(treelist);
    glPopMatrix();
    glPopMatrix();

    glPushMatrix();
    glTranslatef(-120, -120, 120);
    glPushMatrix();
    glCallList(treelist);
    glPopMatrix();
    glPopMatrix();


    //----------------------画主人公小猪----------------------
    glPushMatrix();
    drawPig();
    glPopMatrix();

    //----------------------画夜明珠------------------
    if (get_pearl == 0) {
        glPushMatrix();
        pearl_ball(0, 45, 100);
        glPopMatrix();
    }

    //画灯
    glPushMatrix();
    lamp();
    glPopMatrix();

    //--------------------------屏幕文字的打印
    glPushAttrib(GL_ENABLE_BIT);
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_LIGHTING);
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glColor4f(0.0, 0.0, 0.0, 1.0);
    glLoadIdentity();
    gluOrtho2D(0, 3500, 0, 3500);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
```

```
    glLoadIdentity();
    outputfont(200, 100, "Right-click to open the pig perspective");
    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glPopAttrib();


    glutSwapBuffers();//交换当前窗口的缓存
}
```

结果分析：

可以看到，通过设计出的纹理函数以及传入的各类 bmp 图片，可以对各类物品如树木以及太阳，房屋进行纹理。



（七）场景变换

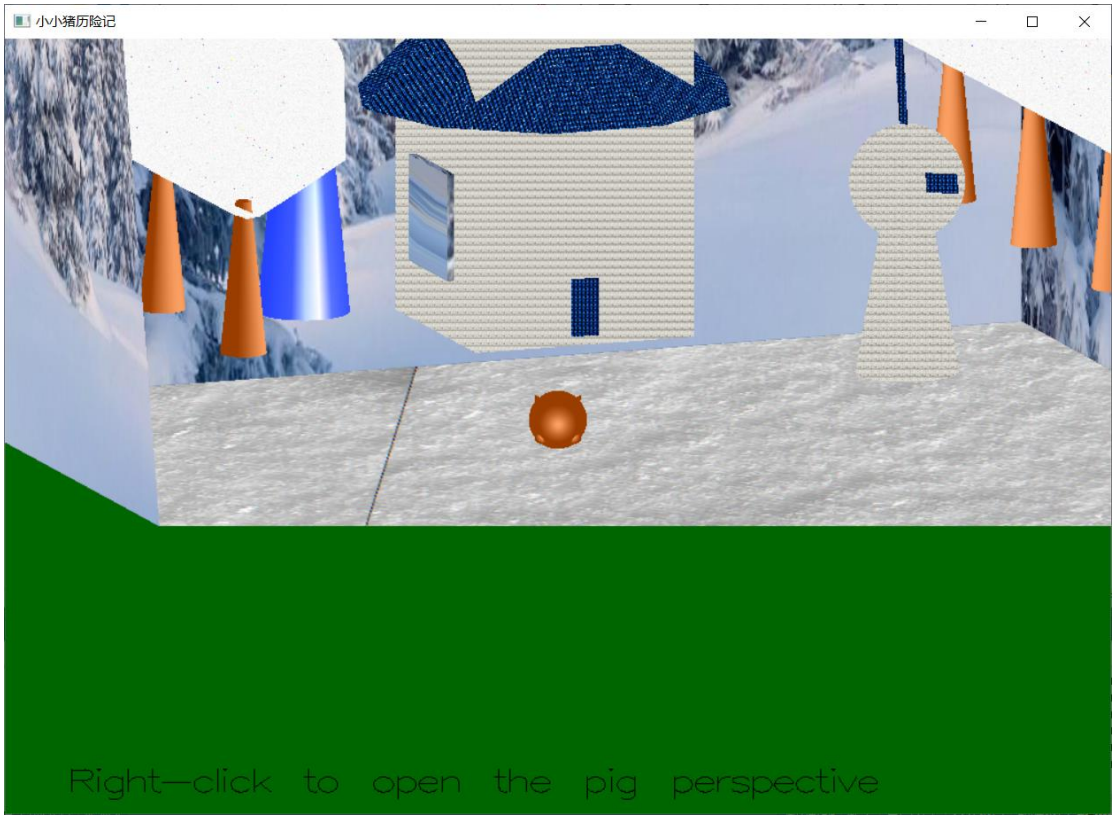
代码如下：

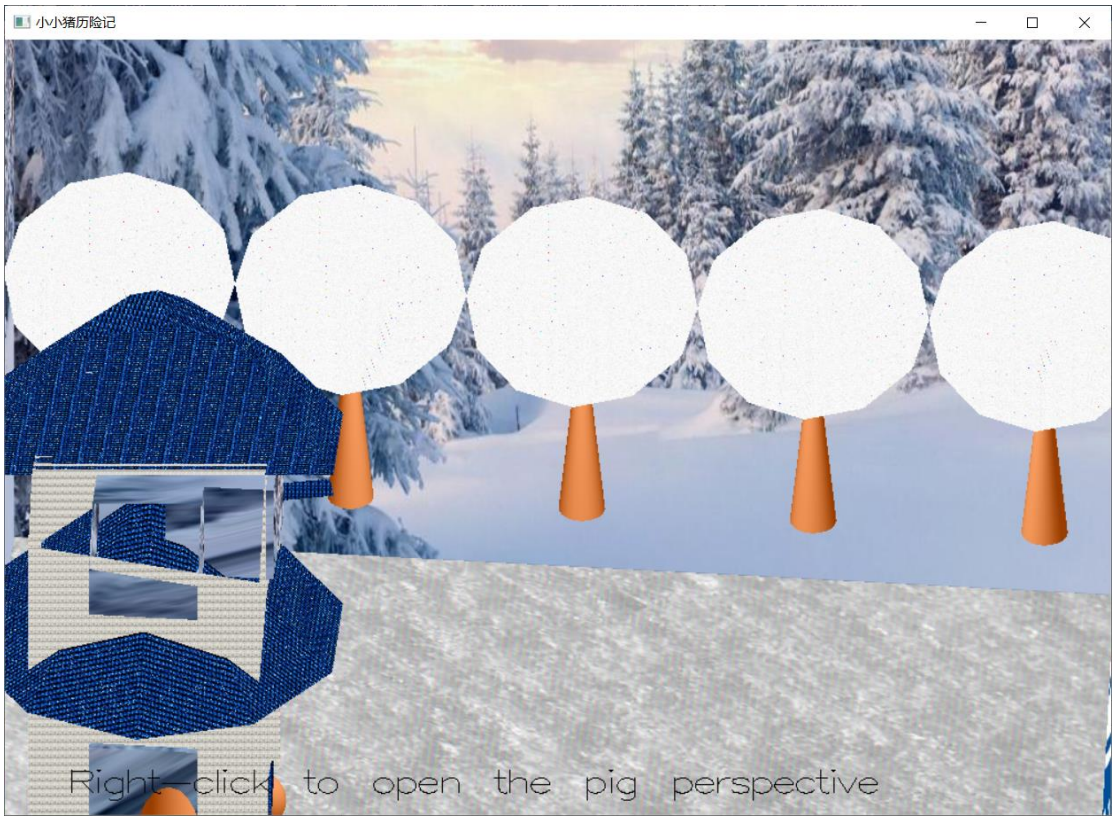
```
void moveMouse(int xMouse, int yMouse) {
    if (MOUSE_MOVE) {
```

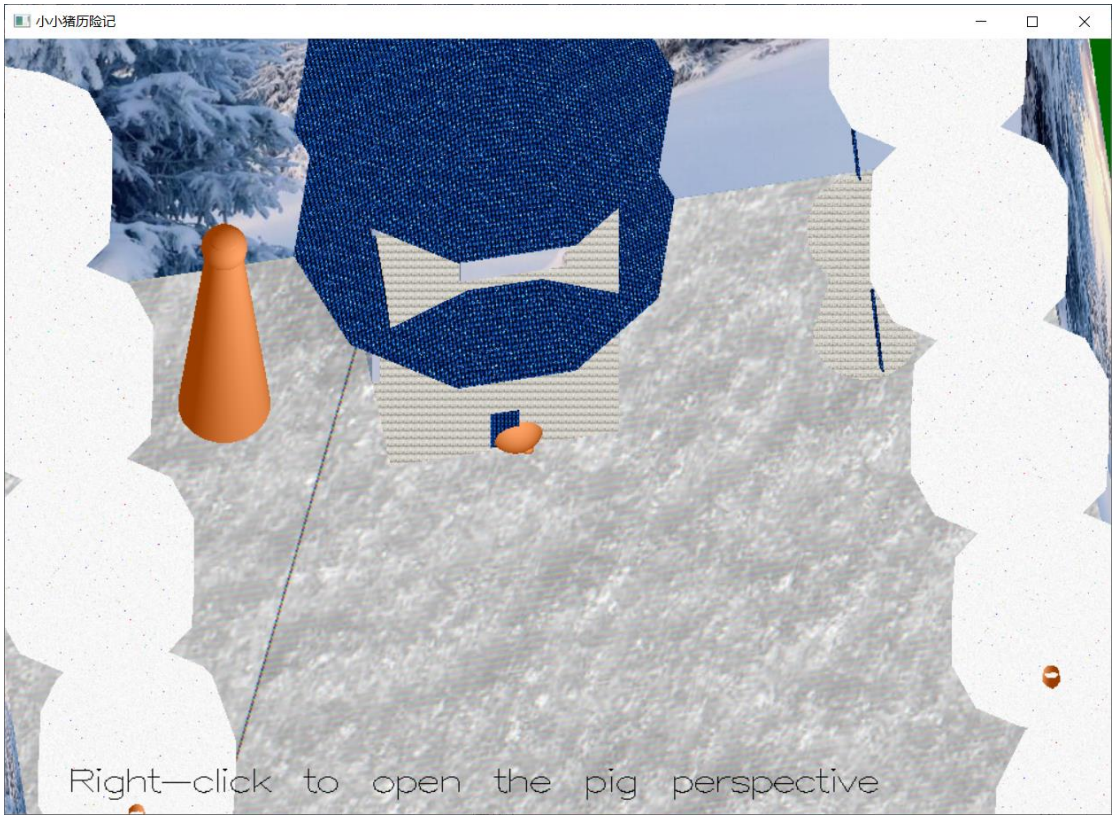
```
        int tmpx = 250;
        int tmpy = 250;
        float setx = xMouse - tmpx;          //鼠标与当前视窗的中心水平偏移
量
        float sety = yMouse - tmpy;          //鼠标与当前视窗的中心竖直偏移
量
        float rate_x = setx / tmpx * PI;     //摄像机水平偏移角度
        float rate_y = sety / tmpy * PI / 2; //摄像机竖直偏移角度
        //水平方向移动分量
        x_temp = 10 * sin(rate_x);
        z_temp = 10 * cos(rate_x);
        //竖直方向移动分量
        fy[view] = 10 * sin(-rate_y);
        float r_castlenght = abs(10 * cos(rate_y));
        //计算视角移动距离
        fx[view] = r_castlenght * x_temp / 10;
        fz[view] = r_castlenght * z_temp / 10;
    }
}
```

部分代码：

```
    switch (Option) {
        case 1:
            MOUSE_MOVE = 0; //此刻无法进行鼠标预览移动了
            view = 1;
            eye_x[view] = ax + 0.1 * sin(theta[1] * PI / 180);
            eye_y[view] = ay + 0.3;
            eye_z[view] = az + 0.01 * sin(theta[1] * PI / 180);

            fx[view] = eye_x[view] + sin(theta[1] * PI / 180);
            fy[view] = eye_y[view] - 0.2;
            fz[view] = eye_z[view] + cos(theta[1] * PI / 180);
            glutPostRedisplay();
            break;
        case 2:

            view = 2;
            get_pearl = 0;
            glutPostRedisplay();
            break;
```


小小猪历险记
Right—click to open the pig perspective


小小猪历险记
Right—click to open the pig perspective



结果分析：

可以看到当进入塔内后，便可更改视角。

（八）菜单实现

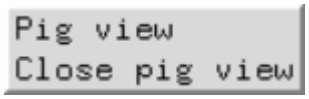
主函数部分代码如下：

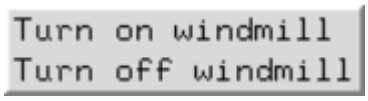
```
    GLint subMenu1 = glutCreateMenu(SubMenu);
    glutAddMenuEntry("Pig view", 1);
    glutAddMenuEntry("Close pig view", 2);


    GLint subMenu2 = glutCreateMenu(SubMenu);
    glutAddMenuEntry("Turn on windmill", 3);
    glutAddMenuEntry("Turn off windmill", 4);



    glutCreateMenu(menu);//创建主菜单
    glutAddSubMenu("Begin to adventure", subMenu1);
    glutAddSubMenu("Windmill rotating", subMenu2);

    glutAttachMenu(GLUT_RIGHT_BUTTON);//指定菜单触发按键
```

对应函数如下：

```
void SubMenu(GLint Option) {
    switch (Option) {
        case 1:
            MOUSE_MOVE = 0; //此刻无法进行鼠标预览移动了
            view = 1;
            eye_x[view] = ax + 0.1 * sin(theta[1] * PI / 180);
            eye_y[view] = ay + 0.3;
            eye_z[view] = az + 0.01 * sin(theta[1] * PI / 180);

            fx[view] = eye_x[view] + sin(theta[1] * PI / 180);
            fy[view] = eye_y[view] - 0.2;
            fz[view] = eye_z[view] + cos(theta[1] * PI / 180);
            glutPostRedisplay();
            break;
        case 2:

            view = 2;
            get_pearl = 0;
            glutPostRedisplay();
            break;
        case 3:
            On = GL_TRUE;
            glutPostRedisplay();
```

```
            break;
        case 4:
            On = GL_FALSE;
            glutPostRedisplay();
            break;
        default:
            break;
    }
    glutPostRedisplay();
}
```





结果分析：

以上即各类菜单栏。

（九）交互实现（）

代码如下：

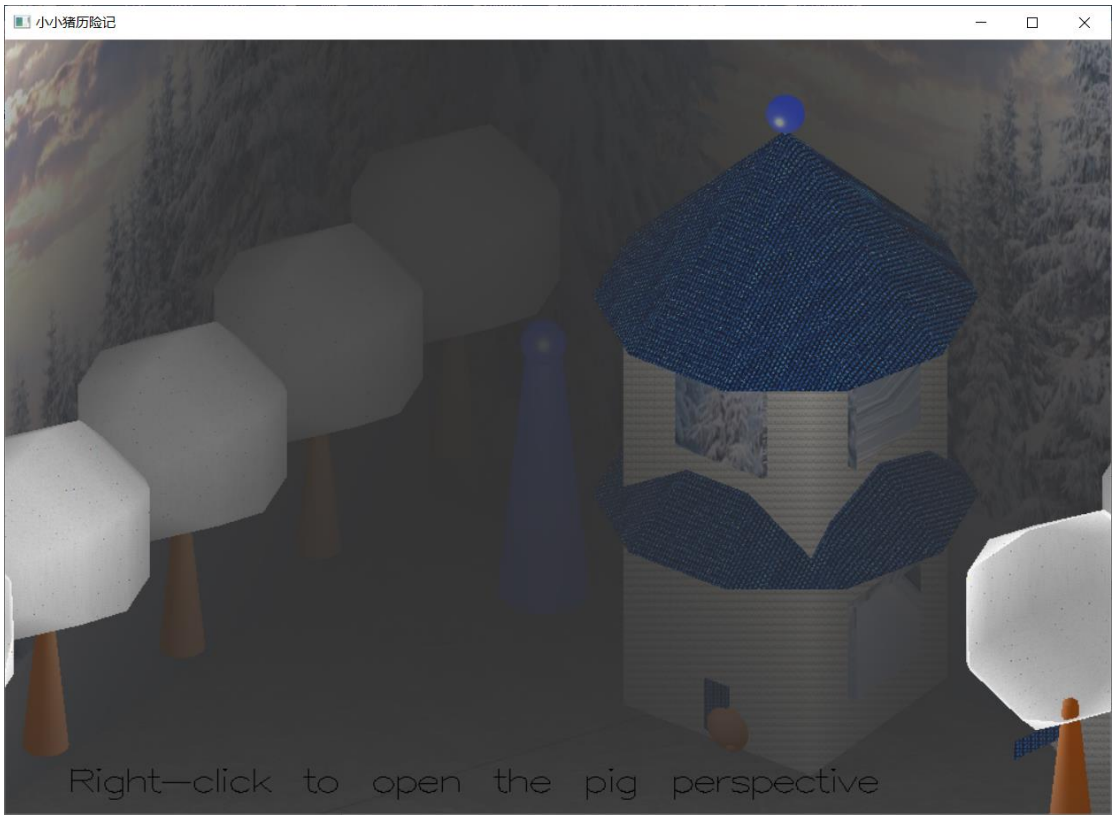
屏幕打印文字功能从博客学习而得。

```
//-------------------------------------------------------------
//----------------------------向屏幕打印文字----------------------
void outputfont(GLfloat x, GLfloat y, const char* format, ...) { //屏幕文字的输出
    va_list args;//定义一具 VA_LIST 型的变量，这个变量是指向参数的指针
    char buffer[200], * p;

    va_start(args, format);//用 VA_START 宏初始化变量刚定义的 VA_LIST 变量
    vsprintf(buffer, format, args);//送格式化输出到串中 用 法: int vsprintf(char *string, char *format, va_list param);     // 将 param 按格式 format 写入字符串 string 中
    va_end(args);//用 VA_END 宏结束可变参数的获取
    glPushMatrix();
    glTranslatef(x, y, 0);
    for (p = buffer; *p; p++)
        glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);//一种字体,仅包括 32 到 127 的 ASCII 字符
    glPopMatrix();
```

```
}
    //--------------------------屏幕文字的打印
    glPushAttrib(GL_ENABLE_BIT);
    glDisable(GL_DEPTH_TEST);
    glDisable(GL_LIGHTING);
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glColor4f(0.0, 0.0, 0.0, 1.0);
    glLoadIdentity();
    gluOrtho2D(0, 3500, 0, 3500);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    outputfont(200, 100, "Right-click to open the pig perspective");
    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glPopAttrib();
```

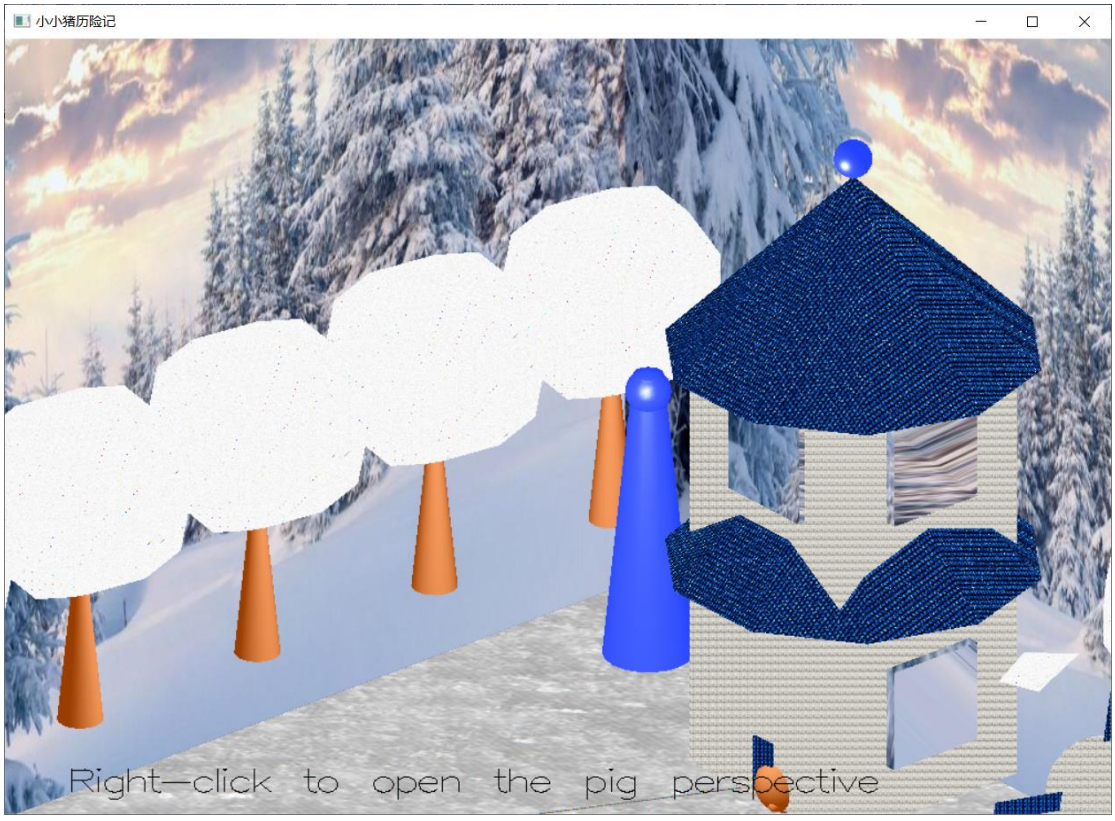
```
//-----------------------------------------------------------------
//-------------------------------画出迷雾----------------------
void drawfog() {
    if (fog == 1) {
        glEnable(GL_FOG);//初始化雾效
        GLfloat fogColor[4] = { 0.2, 0.2, 0.2, 1 }; //雾的颜色

        glFogi(GL_FOG_MODE, fogMode);//定义雾参数 设置雾气的模式
        glFogfv(GL_FOG_COLOR, fogColor);// 设置雾的颜色
        glFogf(GL_FOG_DENSITY, 0.02);// 设置雾的密度
        glHint(GL_FOG_HINT, GL_DONT_CARE);// 设置系统如何计算雾气
        glFogf(GL_FOG_START, 1);// 雾气的开始位置
        glFogf(GL_FOG_END, 5);//结束位置
    } else {
        glDisable(GL_FOG);  //关闭雾气
    }
}

//----------------------创建显示列表------------------------
void loadlist() {
    drawWalls();   //贴环境纹理
    sun();
    roof();
    tree();
```

```
}
```



结果分析：

可以看到通过按钮可以改变迷雾等。



四、结论与展望

心得体会：转眼间，计算机图形学实验课程也即将接近尾声，在这短短的时间里，我从一开始对各类函数和算法具体实现功能的懵懵懂懂，对 openGL 封装库的不熟悉，再到渐渐的领会算法的要点，通过老师发布的实验代码、ppt 以及网络上各类相关知识的学习、课本知识的温习，到最后在接近尾声的时间里较为满意的完成了本次实验报告。在这短短的实验教学经历中，我经历了一开始拿着题目，面对着要求不清楚做什么的迷茫，再到逐渐理清各类函数的作用，慢慢的从 3 维场景的点绘制到线再到面，将图形分解成各个部分逐一分析、

实现。渐渐的，我开始理解和熟练这方面的操作，同时，可以在较短的时间里完成实验原理以及实验内容部分的填写，紧接着就是按照自己的构思和要求一一的完成功能，调通代码，获得具体的图形。当然也有些令人头疼的地方，比如相对坐标系的变换，以及投影矩阵等的操作等等，在进行位置变换时，往往得来来回回尝试很多遍才找对位置，还需要对其一一的甄别。同时，对于图形的设计而言，我一开始的想法是能否在变换后进行逆变换从而抵消变换带来的坐标系变动，比如在执行完一轮绘制后，继续执行其相反操作，让实验尽可能的维持在一个位置绘制。然而结果恰恰相反，这个想法让我在绘制一开始的部分图形时吃了不少苦头：由于大量的逆运算，导致代码量十分庞大，同时容易出现覆盖，使得位置不尽如人意，后来通过glPushMatrix();和glPopMatrix();函数完美的解决了这一问题，通过将当前位置入栈出栈从而实现了全局位置的保存，最终解决了这个问题。因此，我意识到，抱着想要简化操作的心理，而不经过细致的分析，往往得到的结果会适得其反，使得问题变得更加复杂。同时，我也更加清楚了解决问题是一个动手实践的过程，自己设想的并不一定就是恰当的和理的，需要通过不断的实践去检验它，只有通过了数据检验的方法才是真正可行的方法。

这门课的的确确的让我体会到了什么是图形学实验：从一开始听到老师说实验内容自拟时，我不禁思考以这种形式是否可以圆满完成实验任务，是否能够考虑全一个场景完全的符合所有要求。然而，通过了这几天的不断尝试和迭代，我开始逐渐的找到了自己的想法，可

以较为全面的考虑到实验所提出的实验要求，还能通过各类网站、博客上的信息搜寻和思路借鉴找到新的 idea，同时巩固自己的代码能力。我知道圆满完成一个实验的背后往往是许多的汗水以及时间的付出，还记得第一天开始复杂场景构建时，到完完全全绘制出较为成功的场景，我花了满满一整天的时间：从一开始的认识各类函数，到查看软件自带的 openGL 文档，到一步步的实现图形绘制，我发现只有每个环节都亲力亲为才能够在出现问题的时候立即察觉到是哪部分出现错误，也才能在图形位置不正确的时候知晓忘记了什么功能操作，最终完成任务。因此，我发现想要做好实验，培养自主学习、查阅资料和理解实验细节的能力对于我们来说就显得格外重要。同时，我觉得这门课带给我的收获不光是 openGL 和图形绘制方面的知识，还有一个重大收获就是切切实实的增强了我的耐心、知识搜索能力和自学能力，从一次次完成实验的过程中，我体会到了绘制图形的不易，尤其涉及到视角变换、动态交互等方面，稍微分心就会忘记当前的位置。同时我也逐步理解了各种函数的功能、各类操作定义的意义，此外我还学会了如何去回退代码，对错误结果一步步 debug，从而使得实验结果截图契合实验要求，这些都是使得实验能够完成的重要保障条件。

最后，我想感谢老师：虽然最后实验时间并不在课上，我们与老师您进行手把手教学的机会不多，但是这种以自由大作业的形式授课也让我学会了如何操作各种函数，完成各项要求，实现自己想要实现的功能，使得我收益良多。同时也希望在之后的课程中可以继续出现严老师的身影。

至此，表达我的感谢和对本次实验经历的珍重！