图形学大程终期报告

李博怡 3140102256
潘哲 3140104209
郑驭聪 3140102384

一、 技术分析

我们一共实现了如下特性（已涵盖所有基本点和两个加分点）

• 基本点

1. 基于 OpenGL，具有基本物体的建模表达能力
2. 具有基本三维网格导入导出功能

我们利用 maya 实现了书房场景的建模，我们构建了类似书房的部分元素，比如书本、门、窗、桌椅等，并实现了后序事件触发时所需要使用的元素。

场景的实现方法是先通过网上搜索一些合适的 obj 模块，利用 maya 将它们整合到一个完整的场景中，然后将整个场景的 obj 导出。

Obj 导入部分我们封装成了一个类 class.h，这个类主要是通过它自己的构造函数来解析 obj 文件和 mtl 文件，读取对应的物品的位置和物品的贴图信息，然后我们把这些信息存储在类的指定成员变量中，等到要画时再调用类中的相应绘制函数实现绘制。

（这部分代码参考与网上的读取 obj 文件代码）

其中读取 obj 部分的代码如下：

```
while (getline(in, line))
{
    if (line.size() == 0 || line[0] == '#') continue;
    istringstream is(line);
    is >> word;
    if (word == "v")
    {
        VERTEXF p;
        is >> p.x >> p.y >> p.z;
        vertexs.push_back(p);
    }
    else if (word == "vt")
    {
        pair<float, float> p;
        is >> p.first >> p.second;
        texcoords.push_back(p);
    }
    else if (word == "vn")
    {
        VERTEXF p;
```

```
            is >> p.x >> p.y >> p.z;
            normals.push_back(p);
        }
        else if (word == "o" || word == "g")
        {
            if (!goname.empty() && !faces.empty())
            {
                Object obj(vertexs.begin(), vertexs.end(),
                    texcoords.begin(), texcoords.end(), normals.begin(),
                        normals.end(), faces.begin(), faces.end(), row, col, mtlname);
                obj.setName(goname);
                object.push_back(obj);
                faces.clear();
            }
            is >> goname;
        }
        else if (word == "f")
        {
            int r = 0, c = 0;
            while (is >> word)
            {
                c = count(word.begin(), word.end(), '/');
                if (c == 0)
                {
                    faces.push_back(atoi(word.c_str()));
                }
                else if (c == 1)
                {
                    faces.push_back(atoi(string(word.begin(),
                            word.begin() + word.find("/")).c_str()));
                    faces.push_back(atoi(string(word.begin()
                            + word.find("/") + 1, word.end()).c_str()));
                }
                else if (c == 2)
                {
                    int a = word.find("/");
                    int b = word.find("/", a + 1);
                    faces.push_back(atoi(string(word.begin(),
                            word.begin() + a).c_str()));
                    faces.push_back(atoi(string(word.begin() + a + 1,
                            word.begin() + b).c_str()));
                    faces.push_back(atoi(string(word.begin() + b + 1,
                            word.end()).c_str()));
                }
```

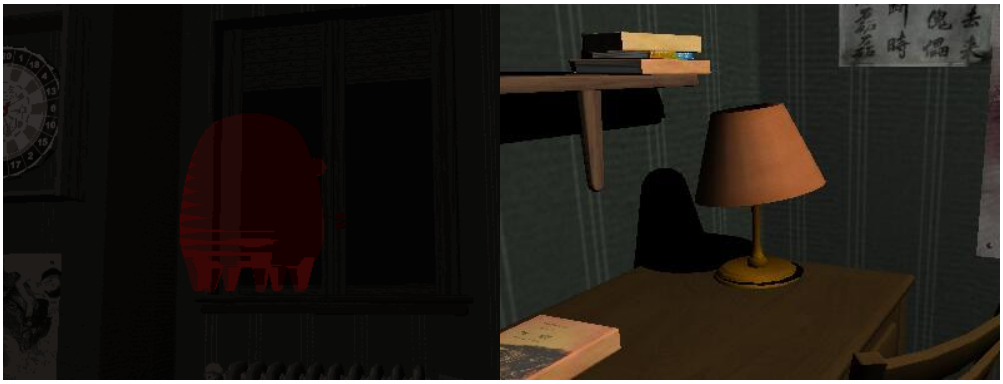
```
                ++r;
            }
            row = r;
            col = c + 1;
        }
        else if (word == "mtllib")
        {
            is >> word;
            ReadMtl(cd, word, material);
        }
        else if (word == "usemtl")
        {
            is >> mtlname;
        }
}
```

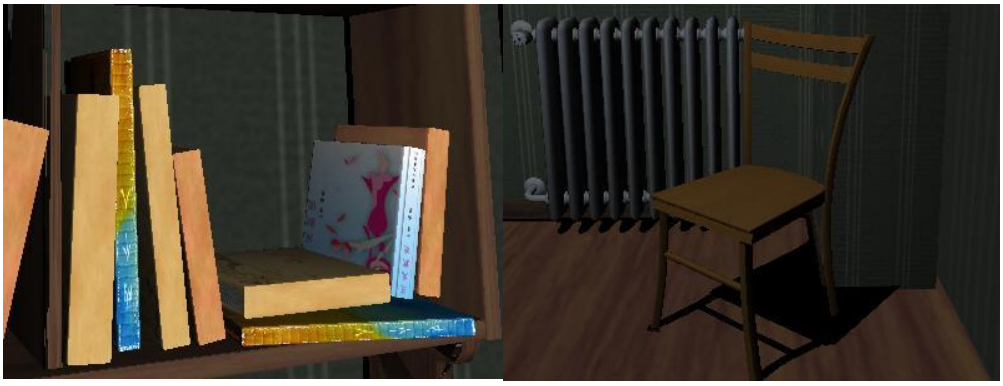
读取 mtl 部分的代码如下：

```
while (getline(in, line))
{
    if (line.size() == 0 || line[0] == '#') continue;
    istringstream is(line);
    is >> word;
    if (word == "newmtl")
    {
        is >> ntname;
        if (!ptname.empty())
        {
            if (hasmap)
            {
                mat.insert(make_pair(ptname, Material(ambient, diffuse,
specular, emission, map)));
            }
            else
            {
                mat.insert(make_pair(ptname, Material(ambient, diffuse,
specular, emission, 0)));
            }
        }
        ptname = ntname;
        hasmap = false;
    }
    else if (word == "Ka")
    {
        is >> ambient[0] >> ambient[1] >> ambient[2];
```

```
    }
    else if (word == "Kd")
    {
        is >> diffuse[0] >> diffuse[1] >> diffuse[2];
    }
    else if (word == "Ks")
    {
        is >> specular[0] >> specular[1] >> specular[2];
    }
    else if (word == "Ke")
    {
        is >> emission[0] >> emission[1] >> emission[2];
    }
    else if (word == "map_Kd")
    {
        is >> fname;
        Texture texture = Texture(cd + "\\" + fname);
        map = texture.getID();
        hasmap = true;
    }
}
```

最后导入以后的场景如下：

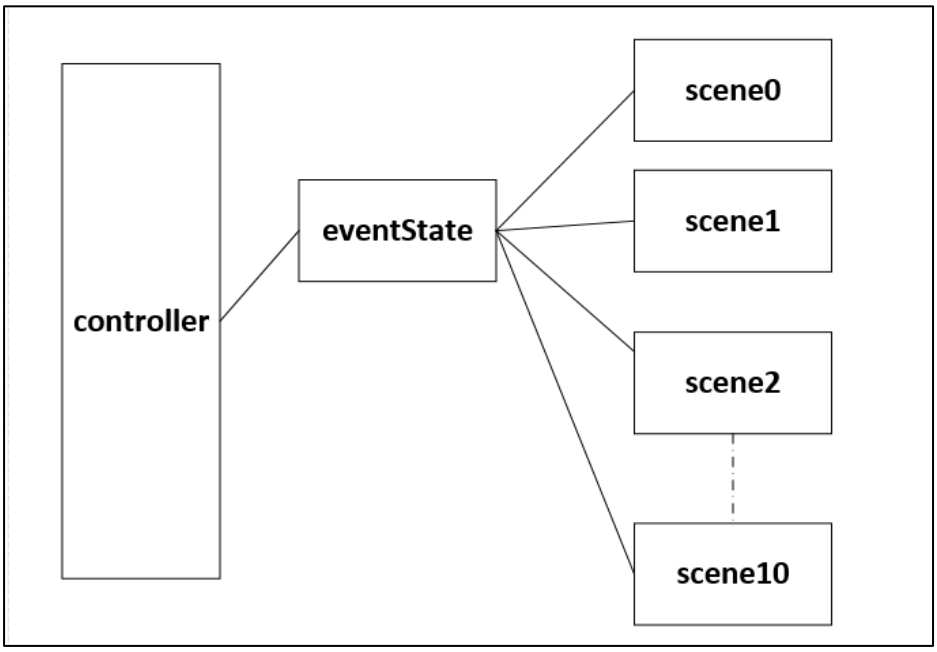




这样的场景我们绘制了 10 个，这是由我们的事件触发机制决定的，我们的每一个场景都是一个对象，我们的控制机制利用一个信号变量来控制，每当点击触发信号更改时，我们会根据信号的值找到对应要重画的场景进行重绘。大致原理图见下：

```
//场景控制信号
#define GAMEBEGINTIP -2
#define GAMESTART -1
#define INITIAL 0
#define NOCLOCK 1
#define NOPICTURE 2
#define NOPICTURE_NOCLOCK 3
#define NOTARGET 4
#define NOTARGET_NOCLOCK 5
#define NOHAMMER 6
#define NOHAMMER_NOCLOCK 7
#define FINAL 8
#define FINALEND 9
#define FINALBEGIN 10
#define GHOST 11
#define PART 12
#define LAMP 13
```

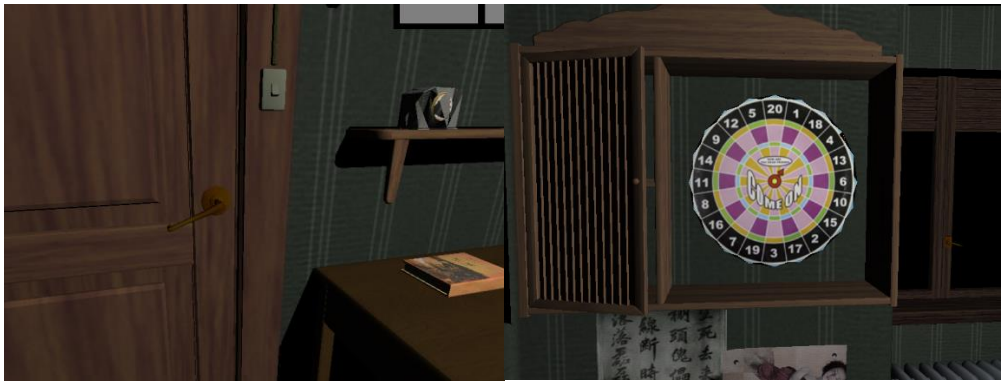
上述这些控制信号分别对应不同的场景，我们通过 controller 产生不同的控制信号改变 eventState，通过 eventState 来改变要绘制的场景，从而实现不同时间段的事件对应不同的场景，控制模式如下：



3. 具有基本材质、纹理的显示和编辑能力

能够将 obj 文件中读取到的材质信息和纹理信息加在具体的每个物品上，由于在本程序中，所有物品均有纹理信息，因此材质信息实际上不需要了。对于纹

理，首先我们打开二维纹理使能，使纹理信息能够绘制到每个点上；第二，导入贴图 bmp 文件，从 bmp 文件上第 18 个字节读取文件的宽度和高度，并从第 54 个字节开始读取具体每个 pixel 的值；第三，使用 glGenTextures 生成纹理的 ID；第四，使用一系列函数设置纹理的属性，比如 glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)能够使纹理和光照同时存在。
以下为我们在场景中加入纹理的截图：



4. 基本几何变换功能（旋转、平移、缩放）

为了实现这一部分的功能，我们多增加了一个鬼在场景中漫游，这个鬼是通过 maya 制作了一个 obj 文件，然后实现导入，导入原理同上。这个鬼的移动坐标是通过一个新的头文件 ghost.h 来控制，作一定范围内的旋转，并同时自转。鬼的坐标控制段代码如下：

```
private:
    int count;
    double delta;
    bool transOutRoom(float trans[])
    {
        if (trans[2] >= 3 &&
            trans[2] <= 11.5 &&
            trans[0] + 12 >= 0 &&
            trans[0] + 12 <= 3)
            return true;
        return false;
    }
```

```
public:
	Ghost()
	{
		count = 0;
		delta = 0;
	}
	void changePos(float color[], float scale[],
		 float trans[], float rotateGhost[], bool isLightOn)
	{
		count++;
		if (count >= 10)
		{
			count = 0;
			trans[0] = 5 * cos(delta);
			trans[2] = 5 * sin(delta);
			delta += 2 * PI / 360;
			if (delta >= 2*PI)
				delta -= 2 * PI;
		}
		rotateGhost[0] += 0.2;
		if (rotateGhost[0] >= 360)
			rotateGhost[0] -= 360;
	}
```

其实就是不断地把鬼的坐标信息发送过来作更改，然后再送出去，实现重绘。同时这个鬼还由灯光控制，如果灯光一开，鬼就会消失；灯光关上，鬼就重新出现。

鬼的展示如下：

5. 基本光照明模型要求，并实现基本的光源编辑

能够实现基本的光照。比如在下图中，台灯和座椅的明暗变化，以及墙和地板的明暗变化：

光照的实现依赖 OpenGL 自带的光照模型，为了方便使用，我们封装了一个 Light 类，主要用于设置参数和显示。光照方面主要设置的参数如下：position[x, y, z, t]代表光源的位置，其坐标为(x/t, y/t, z/t)。当 t 不为 0 时，光源为点光源；当 t 为 0 时，光源处于无穷远处，因此可以视为方向矢量为(x, y, z)的方向光源。ambient[r, g, b, a]表示表示该光源所发出的光，经过非常多次的反射后，最终遗留在整个光照环境中的颜色；diffuse[r, g, b, a]表示该光源所发出的光，照射到粗糙表面时经过漫反射，所得到的光的颜色；specular[r, g, b, a]表示该光源所发出的光，照射到光滑表面时经过镜面反射，所得到的光的颜色。设置完参数后，调用 glEnable(GL_LIGHTx)（x 代表 0-7 的数字）进行绘制即可。

两个注意事项，第一个是首先需要 glEnable(GL_LIGHTING)，才能使用光照，在绘制三维物品时，不应关闭光照；第二个是应在绘制三维物品纹理时，设置如下属性：glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)。只有 GL_MODULATE 才能将光照的色彩和纹理的色彩进行混合，其他比如 GL_REPLACE 则会用纹理信息覆盖光照信息，这样就显示不出光照的效果了。

光照效果截图如下：

可以看到，对于同样的纹理，能根据光照显示不同的颜色。

6. 能对建模后场景进行漫游

我们的密室逃脱设计是以一个人的眼睛为中心。

游戏玩家通过 wasd 来控制人物的前进后退与左右移动。这里的移动是相对视野的，而不是绝对的 xyz 坐标的移动，例如如果按下按键 w，人物将会朝着视野方向的前方移动。

视野是通过鼠标坐标函数实时改变的，鼠标移动将直接使得视野移动。如果鼠标触及边界，将会使得视野朝着那个方向缓慢移动。

按下空格键将会产生跳跃效果，即设置一个重力，模拟人物的弹跳过程。如果人物处于跳跃状态，则可以实现跳到桌子上或椅子上的功能。

与此同时，如果人物从桌子上走下，人物都将模拟自由落体下落。

针对场景漫游我们封装了一个 View 类，View 类定义如下：

```
class View
{
private:
	float velocity;
	bool isJump;
	int count;
	isLegalToMove canMove;
public:
	Vector3 eye;    //视点
	Vector3 neweye;
	Vector3 center;//视角中心
	Vector3 newcenter;
	Vector3 polar;
	static int viewRate;

	View();
	static void setRate(float Rate);
	void viewMove(int direction);
	 void update();
	 void reset();
	 void setView(float x, float y)
}
```

这个类里封装了视点，视角中心的三维坐标。同时，保存了视线的方向。

对于键盘的前进后退左移右移的实现，我们设计了 viewMove 函数，这个函数首先根据中心坐标与视点坐标计算出视线向量，之后找到前进方向和左右方向的单位向量，根据键盘信息将新的中心坐标与视点坐标用向量加法移动到下一个位置。

对于跳跃与走下桌子产生的重力效果，我们在这个类里封装了一个速度，同时配合碰撞检测。我们始终有一个向下的惯性速度，刷新的时候会根据这个惯性速度算出下一个坐标。如果这个坐标可到达，那么速度将会加上一个向下的加速度变量。如果不可到达，则保持原始惯性速度。跳跃的过程即在空格按下时把这个速度设置为向上的一个初速度，之后根据物理效果，速度递减，之后反向回落到地面。

对于鼠标转换视野的方式，我们封装在下列函数里：

```
void setView(float x, float y)
```

```
    {
        x /= viewRate;
        y /= viewRate;
        Vector3 Dir;
        GLfloat alpha, fy;          /*和它的名字一样，不过是单位是弧度*/
        if ((polar.z + y)>5.0f && (polar.z + y)<175.0f)
        {       /*定义竖直偏角只能在 5°到 175°之间*/
            polar.y += x;               /*根据鼠标移动的方向设置新的球坐标*/
            polar.z += y;
            if (polar.y>360.0f) polar.y -= 360.0f;
            if (polar.y<0.0f) polar.y += 360.0f;
                    /*将水平偏角锁定在 0°到 360°之间*/
            alpha = polar.y*PI / 180;
            fy = polar.z*PI / 180;                      /*角度转弧度*/
            Dir.x = -1 * polar.x * sin(fy) * cos(alpha);   /*极坐标转直角坐标*/
            Dir.z = polar.x * sin(fy) * sin(alpha);
            Dir.y = -1 * polar.x * cos(fy);             /*注意：竖直方向的是 y 轴*/
            center.x = eye.x + Dir.x;
            center.y = eye.y + Dir.y;
            center.z = eye.z + Dir.z;
        }
    }
```

这个函数传入参数是鼠标的坐标偏移，其中鼠标的 y 坐标对应实现的 y 轴，鼠标的 x 坐标根据极坐标系变换方程来求解新的实现方向，之后将重心坐标移动到新的位置，从而实现鼠标控制视线变换。

7. Awesomeness 指数

我们小组的场景包括 4 面的墙，天花板和木质地板。室内场景设计十分精致，包括暖气，一个桌子两个凳子，一个木制门，墙上挂着的飞镖盘，海报。桌子上放着台灯书本，书架上的书横七竖八的摆着，栩栩如生。漫游过程模拟人物的真实过程，包括跳跃等处理的相对真实，增加了真实感与可玩性。同时场景配上了比较合适的音乐，让整个氛围更有代入感。在游戏开头和结尾分别加上了开场和结束，能够重复开始，也使得整个游戏更完整。

• 加分点

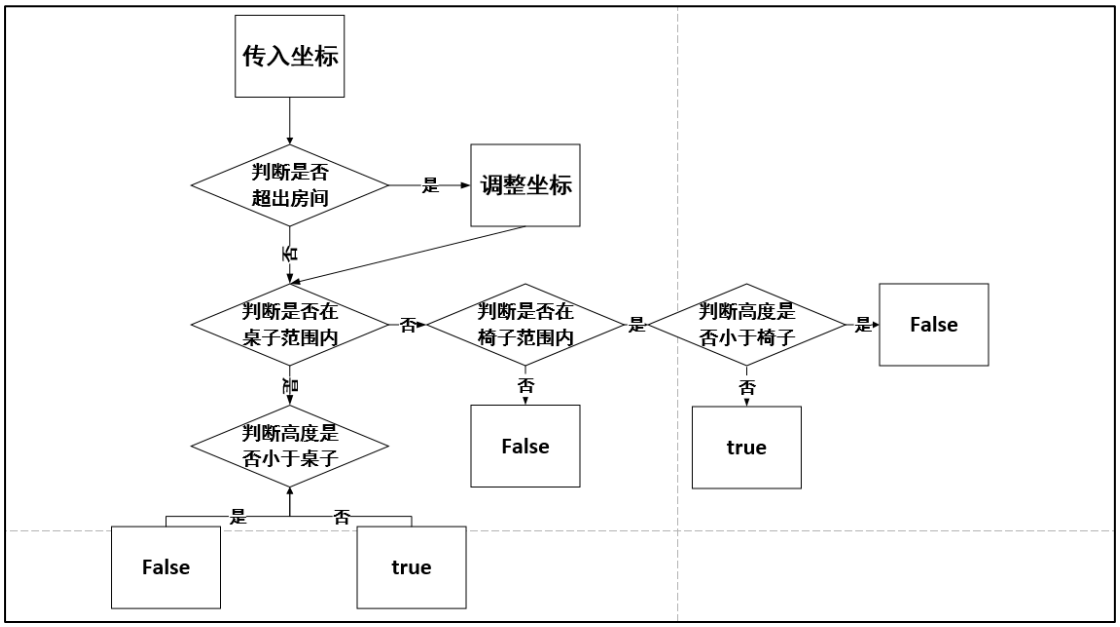
8. 漫游时的碰撞检测

我们小组对整个场景进行了适当的采样，找到了整个场景的边界，桌子、椅子、书架等位置。我们的碰撞检测采用一种试探的方式，对下一个人物位置进行判断。我们的坐标系 xyz 是右手系。首先根据人物位置的 xz 坐标来判断人物的区域，如果在桌子上或椅子上就要保证人物位置高于这个值才能达到。这就需要在跳起的过程中前进到指定的位置。如果在场景外则不能到达，其他位置保持在地面。如果下一个位置能够到达，我们就用新的位置取代之前的位置。通过这种碰撞检测的方式来实现人物位置的锁定。碰撞检测的效果可以在视频录制的地方观看。对于这个碰撞检测，我们单独实现了一个类 CollisionDetection.h，无论是什么情况的碰撞都由这个类来处理，主程序给这个类提交即将到达的下一个位置，这个类通过成员函数来判断出 true 或 false，true 代表可以走到下一个位置，false 代表不能走到下一个位置，部分代码实现如下：

```
void setJump()
{
    isJump = true;
}
void noCollision(Vector3 *tmpeye, Vector3 *tmpcenter, Vector3 eye, Vector3 center)      //房间边界碰撞检测
{
    if ((*tmpeye).x > 3)
    {
        (*tmpcenter).x = center.x + 3 - eye.x;
```

```
            (*tmpeye).x = 3;
        }
        else if ((*tmpeye).x < -12)
        {
            (*tmpcenter).x = center.x + (-12) - eye.x;
            (*tmpeye).x = -12;
        }
        if ((*tmpeye).z < 0)
        {
            (*tmpcenter).z = center.z + (0) - eye.z;
            (*tmpeye).z = 0;
        }
        else if ((*tmpeye).z > 15)
        {
            (*tmpcenter).z = center.z + (15) - eye.z;
            (*tmpeye).z = 15;
        }
}
bool judgeIsOkToMove(Vector3 *neweye, Vector3 *newcenter, Vector3 eye,
Vector3 center)
{
        noCollision(neweye, newcenter, eye, center);
        if (neweye->z >= 3 &&
            neweye->z <= 11.5 &&
            neweye->x + 12 >= 0 &&
            neweye->x + 12 <= 3)//如果在桌子上，且高度小于桌子
        {
            if ((*neweye).y <= TABLEHEIGHT)
                return false;
            else
                return true;
        }
        if (MapChair[(int)(neweye->x) + 12][(int)(neweye->z)] )
        {
            if ((*neweye).y < CHAIRHEIGHT)
                return false;
            else
                return true;
        }
        if (!(noMapInShelf(*neweye) && noMapInTableAndChair(*neweye) &&
((*neweye).y >= BOTTOM)))
            return false;
        return true;
}
```

其中主调函数是 judgeIsOkToMove()函数，用它来分别调用其它的成员函数来判断下一位置的坐标是否合法。它的实现机制大概如下：



先判断下一坐标是否超出房间边界的范围，如果有超出，则调整为允许达到的最大坐标值；继续判断当前坐标是否在桌子对应的矩阵范围内，如果是，则判断高度是否小于桌子，如果高度也小于桌子，则表示没有跳到桌子上，则下一坐标位置不合法，如果跳到桌子上，则下一坐标位置合法；椅子同理。这样使得在地面上时无法进入桌子和椅子的范围，在起跳以后可以达到。同时我们还有高度检测，以确保跳到桌子和椅子的指定高度以后停住。

9. 光照模型细化，可实现实时阴影

我们实现阴影的方式比较简单，即将所有物品按光照方向投影到一个面上并绘制出来。这里最核心的一部分是对每一个投影面产生一个变换矩阵，以后每次绘制阴影时，遍历每一个变换矩阵，将其乘在原坐标系上，将被投影物绘制一遍即可。核心的变换矩阵生成函数 generate_shadow_matrix 如下：

```
void generate_shadow_matrix(GLfloat matrix[4][4], const GLfloat ground[4], const GLfloat light[4])
{
	GLfloat dot = 0;
	int i, j;

	for (i = 0; i < 4; i++) {
		dot += ground[i] * light[i];
	}

	for (i = 0; i < 4; i++) {
		for (j = 0; j < 4; j++) {
			if (i == j) {
				matrix[i][j] = dot - ground[i] * light[j];
			}
```

```
                else {
                    matrix[i][j] = -ground[i] * light[j];
                }
            }
        }
}
```

其中，输入参数 matrix[4][4]为产生的变换矩阵；ground[4]中四个参数为投影面的参数，如果投影面方程为 Ax+By+Cz+D=0，那么四个参数分别对应的就是 A,B,C,D；light[4]中的参数即为光照的 position 参数。

得到变换矩阵后，就可以在绘制的主函数中绘制阴影了，其伪代码如下：

```
// 画 A 物体
A->draw();

// 画 A 物体在地面的阴影
glPushMatrix();
glMultMatrixf((GLfloat*)shadow_matrix_ground);
A->draw();
glPopMatrix();
```

此外需要注意，绘制阴影前需做如下初始化：

```
glDisable(GL_LIGHTING);
glEnable(GL_BLEND);
glBlendFunc(GL_ZERO, GL_SRC_COLOR);
```

其中，glEnable(GL_BLEND)表示打开混合模型，glBlendFunc(GL_ZERO, GL_SRC_COLOR)表示混合方式为[阴影颜色*0+纹理颜色*阴影颜色]（由于阴影颜色在 0 到 1 之间，因此可以视为一个权重值）。这样混合后的阴影会显得更加真实。

阴影效果截图如下：

二、 其它特性

1. 鼠标拣选效果

考虑到我们的密室逃脱要实现点击选中场景中物品的功能，我们专门设计了用于鼠标拣选的函数。

由于场景中物品较多，较为复杂，我们采用将关键物品抽象成中心点的方式，通过计算所点击的位置与目标位置的距离来确定是否点中了目标对象。

我们首先在 Line 的类里封装了一个根据鼠标坐标确定一条从视点出发射向鼠标所指方向的射线的一个函数。函数如下：

```
void getLine(int mouse_x, int mouse_y)
    {
        GLdouble modelview[16];
        GLdouble projection[16];
        GLint viewport[4];
        glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
        glGetDoublev(GL_PROJECTION_MATRIX, projection);
        glGetIntegerv(GL_VIEWPORT, viewport);
         GLdouble world_x, world_y, world_z; // 获取近裁剪面上的交点
        gluUnProject((GLdouble)mouse_x, (GLdouble)mouse_y, 0.0,
            modelview, projection, viewport,
            &world_x, &world_y, &world_z);
        near_point.reset(world_x, world_y, world_z); // 获取远裁剪面上的交点
        gluUnProject((GLdouble)mouse_x, (GLdouble)mouse_y, 1.0,
            modelview, projection, viewport,
            &world_x, &world_y, &world_z);
        far_point.reset(world_x, world_y, world_z);
    }
```

这里我们通过鼠标坐标和视野，确定近视窗与远视窗上鼠标坐标对应的点，两个点确定一条连线。

之后，我们根据几何关系，能够确定这条射线与关键物品中心点的距离，如果距离小于特定值，则能说明鼠标位置在该物品上。我们还需要对视点坐标与目标中心点距离求解，小于一个值表示已经足够靠近。只有两个条件均满足才能够触发事件。通过这种方式我们实现了鼠标拣选功能。

2. 音效效果

针对音效，我们精心挑选了一些合适的音乐。我们从网上搜索到了 FMOD 相关的 OpenGL 音频导出方式，直接调用其中的函数，将背景音乐设置为循环播放，在关键的操作设置了对应的音效效果，使得整个游戏设计更加丰满。

3. Box 窗格实现

在 GUI 中，按钮是不可或缺的。为了显示物品框，同时便于开始和结束画面的设计，我们封装了专门的 Box 类，仿照按钮的功能。

Box 类实现原理为，首先通过如下函数：

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.0, 1.0, 0.0, 1.0, 0.0, 1.0);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

将三维坐标系转化为二维坐标系（由于视线和 z 方向重合，因此 z 方向相当于被降维了）。在二维坐标系中，我们很容易使用绘制带有某种颜色的矩形按钮。对按钮加背景也非常简单，只需要让四个顶点对应纹理贴图的四个角即可。最后，将坐标系恢复为原来的三维坐标系，这样就能在三维坐标系中绘制出二维的物品框了。

Box 类中可以设置矩形的宽(width)、高(height)、距屏幕左侧位置(left)、距屏幕下侧位置(bottom)、背景图片(texturePath_default)，可以通过 isEnable 和 disEnable 函数绘制和消失，可以设置鼠标点击事件(click)，并在 Mouse 函数中检测。Box 类整体和 Button 非常类似，在本次游戏中不同阶段都得到了很好的应用。

Box 截图如下：

三、 使用说明

键盘区：
WASD 控制人的前后左右移动
SPACE 键控制跳跃
ESC 键可在程序进行时直接结束程序
鼠标区：
鼠标移动控制人的上下左右视角改变
鼠标点击触发事件（拾取物品，合成物品，输入密码，打开灯光等）
通关秘籍：

1. 取得书架上的台灯
2. 点击贴画获得密码箱的密码
3. 跳上桌子
4. 点击彩色靶获得输入密码权限
5. 输入我们在贴画里获得的密码
6. 取走掉落的烟灰缸
7. 利用烟灰缸砸碎闹钟获得钥匙
8. 点击钥匙，然后点击门把手
9. 逃出

结束区：
结束区有两个按钮，你可以选择重新开始或是直接退出