图形学大作业报告

项目名称：基于OpenGL的3D迷宫

组员姓名及学号：

3180105260 王旭龙

3180104180 姚喆雨

3170101547 周青鑫

指导教师：童若锋

日期：2019年12月28日

设计背景

计算机图形学与OpenGL

OpenGL是用于渲染2D以及3D矢量图形的跨平台应用程序编程接口(API)，由几百个不同的函数组成。其流水线式的作业方式广泛的应用于计算机辅助设计、电子游戏以及可视化程序当中。

计算机图形学作为一个计算机图形学相关领域的入门课程，使用了OpenGL作为教学使用的程序编程接口，在整个课程中，使用OpenGL去绘制图形，了解相关底层函数的实现方法，并且将他们运用到对于一般图形的使用过程中，力图绘制看起来更为真实的图片。

游戏背景

迷宫是一款比较经典的3D游戏。玩家需要通过在主菜单中观察迷宫的大体形状，在进入游戏后找到路径走向出口，通过WASD控制就可以在迷宫中演各个方向进行移动。

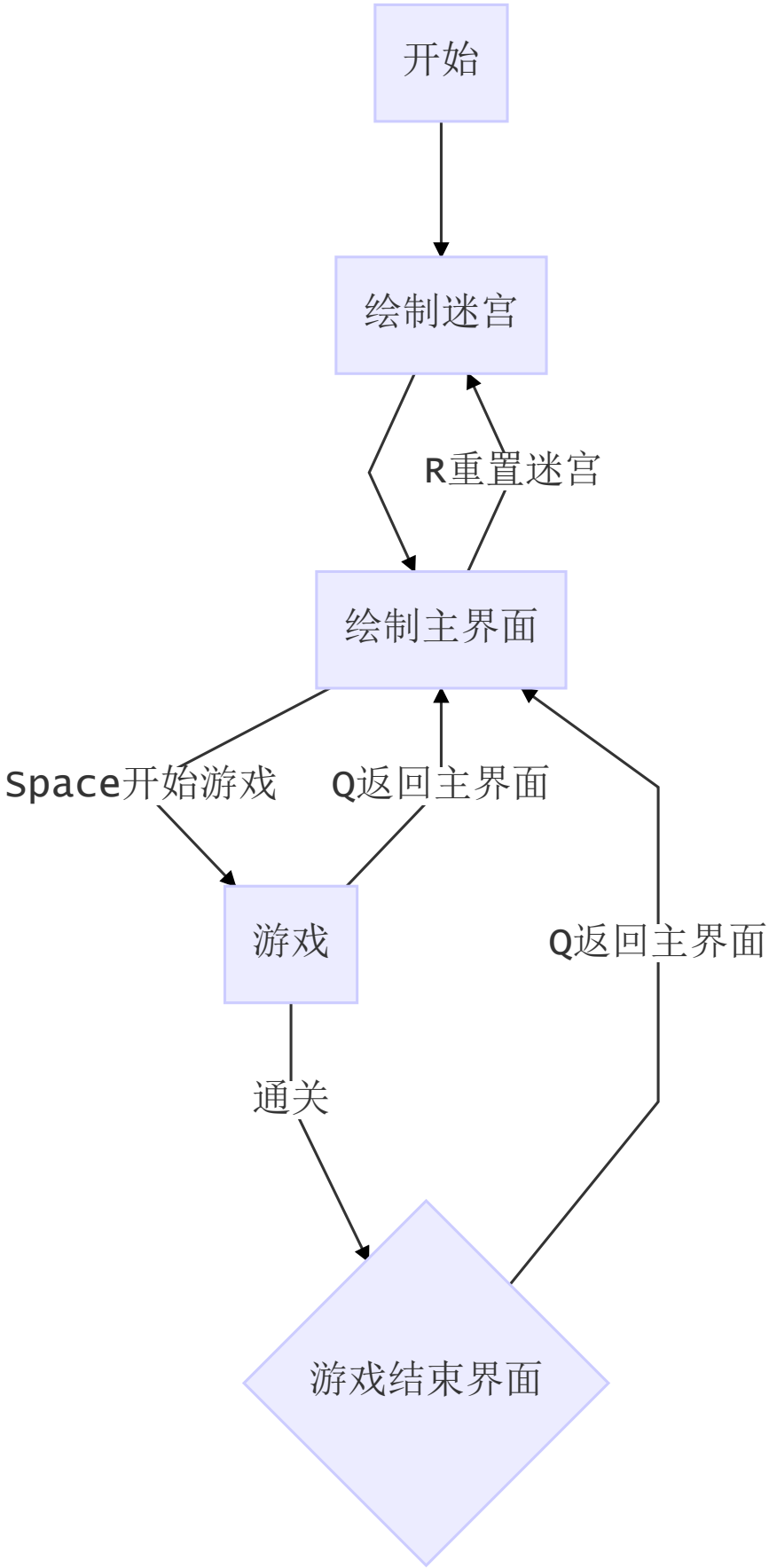
操作介绍

为保持良好的体验，请开启垂直同步并在60帧下运行

这次的大作业主要是利用OpenGL的函数进行绘制各种各样的图形。并且使用键盘相应函数对于键盘输入的信息作出反应。主要的按键如下

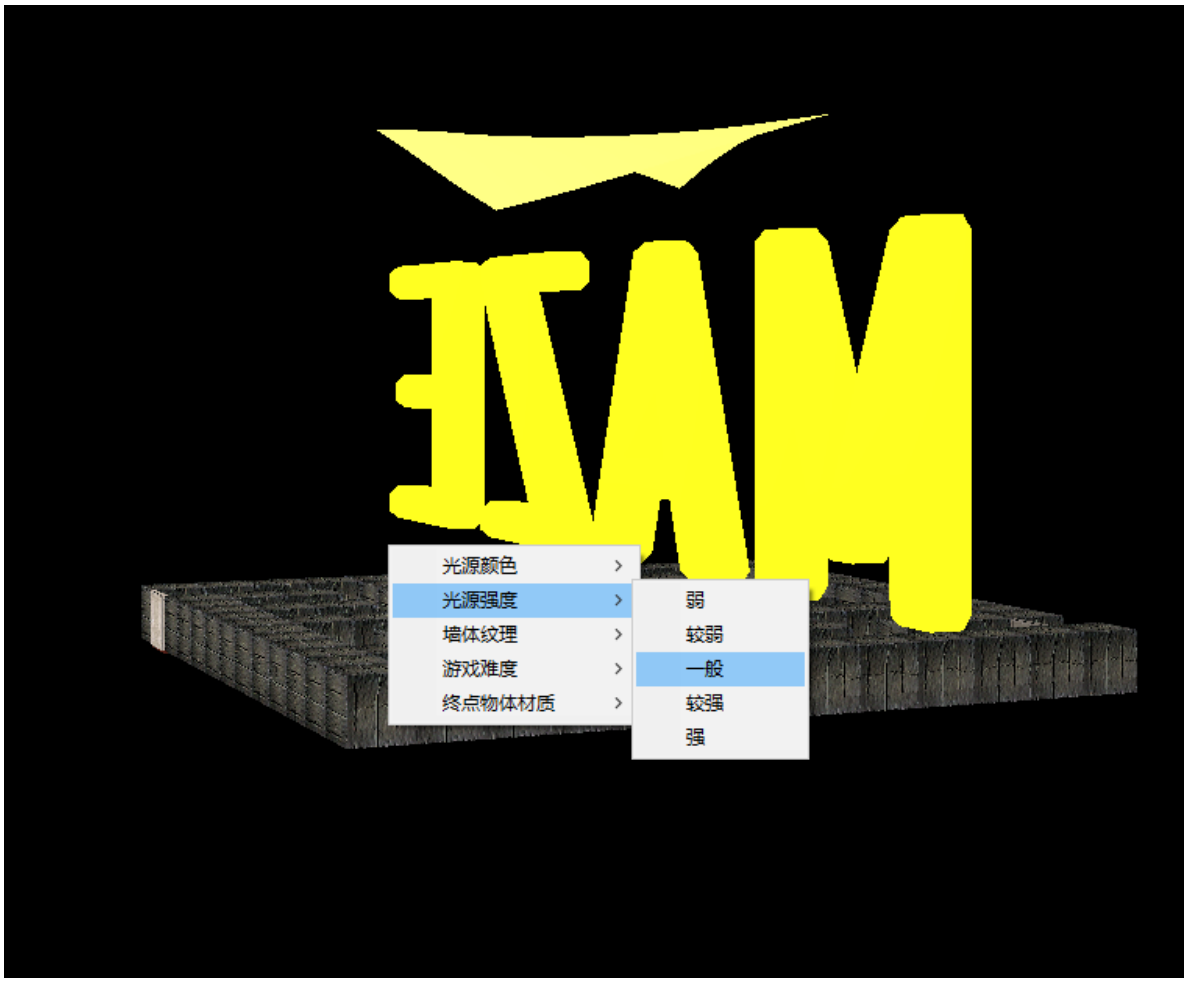
按键	功能
-	截图并保存（保存在游戏目录下）
鼠标右键	打开控制菜单，修改光强、光色、纹理、游戏难度、终点处物体的材质等参数
UIOJKL	控制光照的位置
Space	开始游戏
=	改变MAZE的显示方式（轮廓--渲染）
Esc	离开游戏
R	刷新迷宫
W	前进
S	后退
A	左移
D	右移
N	向左转动视角
M	向右转动视角
Q	返回主菜单
Alt+W/S	加速前进/后退

设计思路

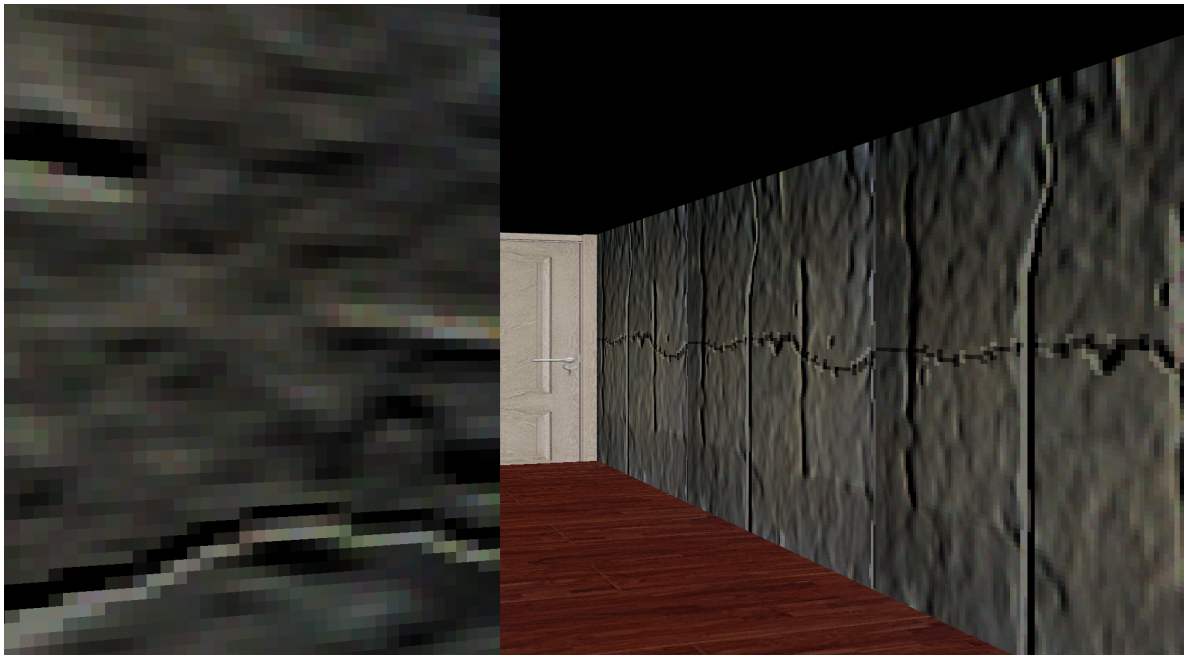

开始
绘制迷宫
R重置迷宫
绘制主界面
Space开始游戏
Q返回主界面
游戏
Q返回主界面
通关
游戏结束界面

程序演示

游戏主界面，四个字母是从OBJ文件中读入的字符，并且输出在迷宫正上方。上面的曲面是采用NURBS建模的曲面，并且通过调整控制节点的方法实现一种类似于波浪状的效果。

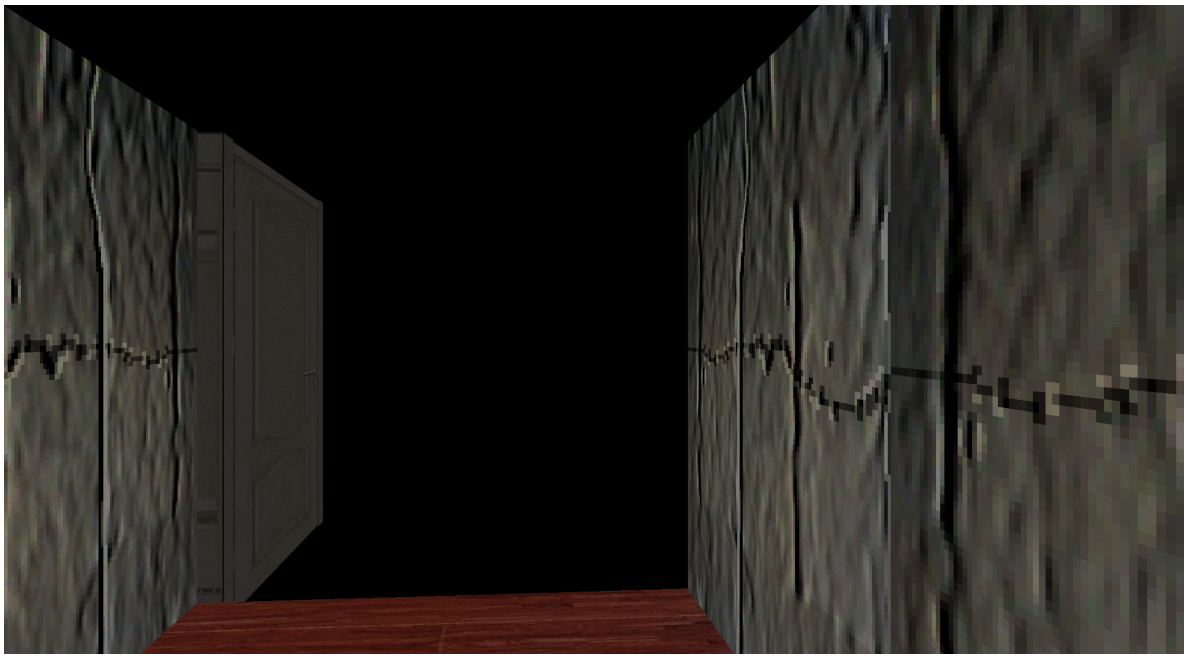


在程序运行过程中，我们可以使用右键来调出控制菜单，来修改光源强度、颜色、墙体纹理以及游戏难，终点物体的材质度等参数。在主界面按空格键即可进入游戏，或者按R更换迷宫。视角会渐进地移动到游戏开始的位置上。

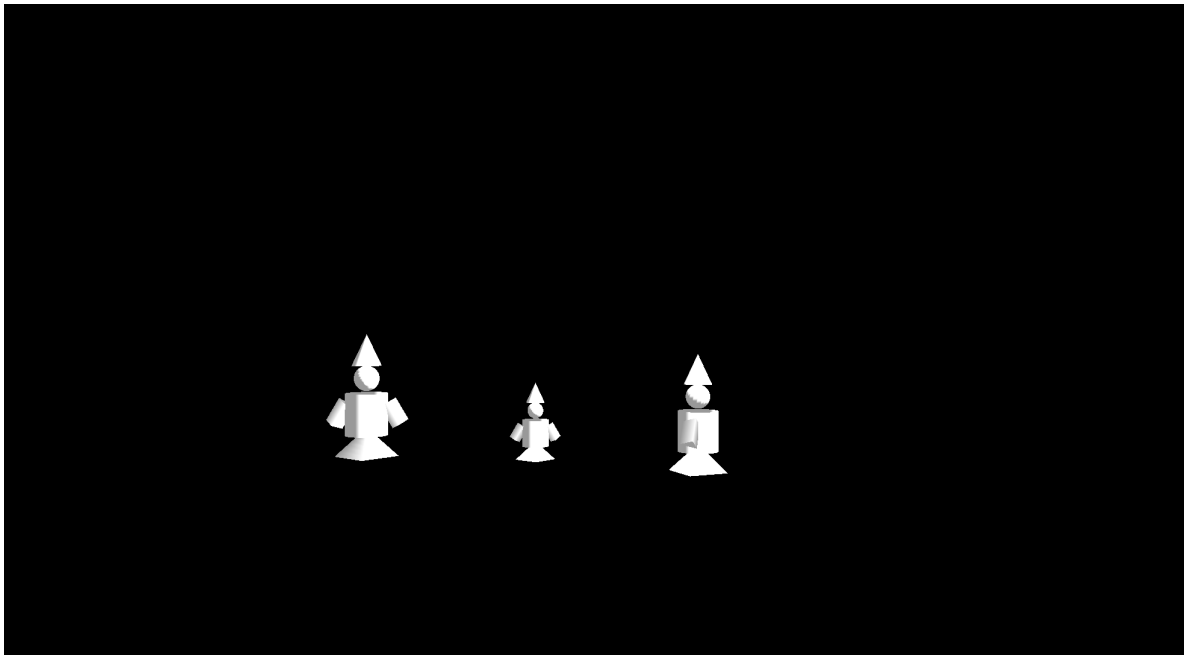


在游戏过程中，我们可以通过WASD修改位置并且用NM控制视角的旋转。在游戏过程中。会检测与墙的距离，如果距离太过于接近，则不能向这个方向前进，避免穿模的情况发生。

在游戏过程中按Q可以直接返回主界面。



接近终点的时候，终点的大门会打开，再向前前进，就可以到达终点页面。



这是终点物体的演示页面，展示了基本物体的构型以及光照模型的建立，可以使用UIOJKL调整光源的位置，并且可以用右键调整集中预制的材质类型。

这时按Q键可以返回主界面，重新开始游戏。

实现功能

1. 基于OpenGL（程序中未使用其他任何库，除实验课使用的Opengl库以及c++自带的库函数），具有基本体素（立方体、球、圆柱、多面棱柱、多面棱台）的建模表达能力。（在游戏结束界面绘制了所要求的基本体素）
2. 具有基本三维网格导入导出功能（主界面迷宫的MAZE是由OBJ文件中读入的，并且导出了字符的OBJ文件）
3. 具有基本材质、纹理的显示和编辑能力（在程序运行中，可以通过右键打开菜单对纹理进行编辑，对终点物体进行基本材质的选择能力）
4. 基本光照明模型要求，并实现基本的光源编辑（如可以使用UIOJKL调整光源的位置，用右键打开菜单调整光强等参数）
5. 具有基本几何变换功能（在最终的结束页面实现了旋转、平移、缩放的操作）。
6. 能对建模后场景进行漫游如**Zoom In/Out， Pan, Orbit, Zoom To Fit**等观察功能。（在主界面按空格键开始游戏时有Zoom To Fit的效果，并且在游戏过程中可以自由移动，也是一种漫游）
7. 能够提供动画播放功能（游戏可以流畅地运行），能够提供屏幕截取/保存功能（使用“-”按键可以截图并且保存到游戏目录下）。
8. 具有NURBS曲面建模能力（开始菜单中MAZE上方地曲面是用16个控制点建立的，我们修改控制点实现了波动地效果）
9. 构建了基于此引擎的完整三维游戏，具有可玩性（整体设计为一个3D迷宫游戏，具有良好的游戏体验）。
10. 具有一定的对象表达能力，在我们的程序中表达了门、墙等物体对象。
11. 利用自己构建的函数实现了在迷宫中较为简单的碰撞检测，不会初现穿模的情况。

数据结构

在构建整个迷宫的时候，使用了并查集这种数据结构。

而在储存点的位置的时候，使用了pair<double, double>的方式来储存一个二维的坐标，对于三位的坐标，使用了pair<pair<double,double>,double>来储存。

这样可以简化代码中对于一些数据处理的难度。

```
#define TriFloatPair pair<pair<float,float>,float>
#define TriIntPair pair<pair<int,int>,int>
#define QuadIntPair pair<pair<int,int>,pair<int,int>>
#define X first.first
#define Y first.second
#define Z second
#define P second.first
#define Q second.second
```

而OBJ文件读入之后储存在了一个叫做ObjFile的结构体中

```
struct ObjFile {//OBJ文件读入区域
    char name[101][101]; //The name of each object in OBJ file
    int st[101], ed[101]; //The Start of the Object surface ID
    int vc;//vertexs number
    int sc;//surface number
    int mc;//object number
    TriFloatPair vertex[200001];//coordinary of point
    QuadIntPair surface[200001];//surface which consists of 4 points
}letter;
```

实现过程及主要代码

MakeMap函数中，对于一个迷宫，我们将每一个行列标号均为奇数的点设置为可以走的区域，随后对于每一面墙设置一个随机数作为权值，随后就可以使用最小生成树算法和并查集来处理整个迷宫，连通起始点和终点即可。这种方法可以起到比较好的随即效果，并且随着时间的变化可以随机出不同的地图。

```
void MakeMap(int a[][MAXN], int n)
{
    start[0] = 0;
    start[1] = 1;
    finish[0] = n-1;
    finish[1] = n-2;
  srand(time(0));
  int id[MAXN][MAXN],idx=0;
  int edge[MAXN*MAXN][3],cnt = 0;
  if (!(n&1)) n|=1;
  for (int i=0; i<n; ++i)
    for (int j=0; j<n; ++j)
      a[i][j] = 0;
  memset(id,0,sizeof id);
  for (int i=0; i<n; ++i)
    for (int j=0; j<n; ++j)
      if (!(i&1) || !(j&1)) a[i][j] = 1;
      else id[i][j] = ++idx;
  for (int i=1; i<=idx; ++i) fa[i] = i;
  a[start[0]][start[1]] = 0;
  a[finish[0]][finish[1]] = 0;
  for (int i=1; i<n-1; ++i)
    for (int j=1; j<n-1; ++j)
      if (a[i][j] == 1 && ((i&1)||(j&1))){
        edge[cnt][0] = i;
        edge[cnt][1] = j;
```

```
        edge[cnt][2] = rand();
        cnt++;
      }
  for (int i=0; i<cnt; ++i)
    for (int j=i+1; j<cnt; ++j)
      if (edge[j][2] < edge[i][2]){
        swap(edge[j][2], edge[i][2]);
        swap(edge[j][1], edge[i][1]);
        swap(edge[j][0], edge[i][0]);
      }
  for (int i=0; i<cnt; ++i){
    int u,v;
    if (edge[i][0] & 1) {
      u = id[edge[i][0]][edge[i][1]-1];
      v = id[edge[i][0]][edge[i][1]+1];
    }
    else {
      u = id[edge[i][0]-1][edge[i][1]];
      v = id[edge[i][0]+1][edge[i][1]];
    }
    if (getfa(u) != getfa(v)) {
      fa[getfa(u)] = getfa(v);
      a[edge[i][0]][edge[i][1]] = 0;
    }
  }
}
```

PrtScrn()是用于截图的函数，在这个函数里，用glReadPixels去读取显示区域内每一个位置的像素值，然后输出到bmp文件中，这里的bmp文件用time(NULL)来作为文件名，避免了重复的情况发生。

```
void PrtScrn()
{
    static void * screenData;
    int len = WinHeight * WinWidth * 3;
    screenData = malloc(len);
    memset(screenData,0,len);
    glReadPixels(0, 0, WinWidth, WinHeight, GL_RGB, GL_UNSIGNED_BYTE,
screenData);

    time_t tm = 0;
    tm = time(NULL);
    char lpstrFilename[256] = {0};
    sprintf_s(lpstrFilename,sizeof(lpstrFilename),"%d.bmp",(int)tm);
    WriteBitmapFile(lpstrFilename,WinWidth,WinHeight,(unsigned
char*)screenData);
    free(screenData);
}
```

而BMP文件的读取等内容可以参考

BMP读入

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

BMP输出

```
void WriteBitmapFile(char * filename,int width,int height,unsigned char *
bitmapData)
```

这些都是基本的BMP文件格式下的读入与输出，是图像信息处理课程的相关知识，这里不再赘述。

纹理映射,利用LoadBitmapFile()导入经处理后符合要求（长宽相等，且为4的倍数，位深为24）的BMP图像。再通过texload()加载纹理。通过数组存放各个图像的地址指针。

```
void texload(int i, char *filename)
```

这是绘制迷宫的主要函数部分，对于每一个位置，如果这个位置不可以通过，即为墙，就绘制一个立方体作为障碍物。

```
void Draw_Cube_List()
{
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture[status]);

    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glPushMatrix();
    for(int i=0;i< MAP_SIZE;i++)    {
        for(int j=0;j< MAP_SIZE;j++)
        {
            if(Map[i][j]==1)
            {
                glCallList(cubeList);
            }
            glTranslatef(1, 0, 0);
        }
        glTranslatef(-MAP_SIZE, 0, -1);
    }
    glPopMatrix();
    glDisable(GL_TEXTURE);
}
```

除了迷宫外，我们还给通道处加上“地板”，即厚度很薄的立方体。在绘制立方体结束后还需要贴上对应的纹理。

```
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture[3]);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glPushMatrix();
    glTranslatef(0, -0.5, 0);
    glScalef(1, 0.1, 1);
    for (int i = 0; i < MAP_SIZE; i++) {
        for (int j = 0; j < MAP_SIZE; j++)
        {
            if (Map[i][j] == 0)
            {
                glCallList(cubeList);
            }
            glTranslatef(1, 0, 0);
        }
        glTranslatef(-MAP_SIZE, 0, -1);
    }
```

```
    glPopMatrix();
    glDisable(GL_TEXTURE);
```

而在游戏中移动的过程当中，需要实时判断是否碰撞到了墙壁，如果碰撞到了墙壁就不再允许向该方向移动。这部分的实现如下

```
bool CanMove(double a, double b)
{
    pair <int, int> pos = FindPosition(a, b);
    for (int i=0; i<3; ++i)
        for (int j = 0; j < 3; ++j) {
            if (OutRange(a + boundry[i], b + boundry[j])||IsWall(FindPosition(a
+ boundry[i], b + boundry[j]))) {
                return false;
            }
        }
    return true;
}

bool IsWall(pair <int, int> pos)
{
    if (Map[pos.second][pos.first] == 0) return false;
    return true;
}

bool OutRange(double a, double b)
{
    if (a<-0.5 || a>MAP_SIZE - 0.5 || b > 0.61 || b < 0.5 - MAP_SIZE) return
true;
    else return false;
}
```

对于我们即将移动到的位置，我们采样了目标点周围的九个点，如果其中有一个点是落在墙壁的范围内或者超出了整个迷宫的边界，那么我们认为这个点距离墙壁过于接近，就不能移动到该位置，返回false。否则就判定可以向该方向移动，并且返回true。

对于判断最终的位置，也是同样的道理，只需要判断当前的位置并且与重点做比较就可以了。

读取OBJ文件，这个函数是从一个包含了所有字母的OBJ文件中读取模型并且储存在结构体中

```
void getobj(const char * filename, struct ObjFile * obj)
{
    FILE * fin;
    fin = fopen(filename, "r");
    if (fin == NULL) {
        printf("Error! Can't find the file\n");
    }

    obj->vc = 0;
    obj->sc = 0;
    obj->mc = 0;

    char buf[1 << 10];
    while (fscanf(fin, "%s", buf) != EOF) {
        switch (buf[0]) {
        case '#':
            getline(fin, buf);
```

```
            printf("The Message %s\n", buf);
            break;
        case 'g':
            getline(fin, obj->name[obj->mc + 1]);
            printf("Find a name %s\n", obj->name[obj->mc + 1]);
            obj->ed[obj->mc] = obj->sc;
            obj->st[obj->mc + 1] = obj->sc + 1;
            obj->mc++;
            break;
        case 'v':
            ++(obj->vc);
            fscanf(fin, "%f", &((obj->vertex[obj->vc]).X));
            fscanf(fin, "%f", &((obj->vertex[obj->vc]).Y));
            fscanf(fin, "%f", &((obj->vertex[obj->vc]).Z));
            break;
        case 'f':
            ++(obj->sc);
            fscanf(fin, "%d", &((obj->surface[obj->sc]).X));
            fscanf(fin, "%d", &((obj->surface[obj->sc]).Y));
            fscanf(fin, "%d", &((obj->surface[obj->sc]).P));
            fscanf(fin, "%d", &((obj->surface[obj->sc]).Q));
            break;
        }
    }
    obj->ed[obj->mc] = obj->sc;
}
```

在程序中绘制OBJ文件，并且对于OBJ的绘制位置进行了一些改动，使得其可以美观的显示在程序中，位移的数组来自delta数组。

```
void OutPutObj(struct ObjFile * obj, int id, float * delta)
{
    glColor3f(1, 1, 0);
    for (int i = obj->st[id]; i <= obj->ed[id]; ++i) {
        if (ObjPrintLine) {
            glBegin(GL_LINES);
            glVertex3f(obj->vertex[obj->surface[i].X].X + delta[0], obj-
>vertex[obj->surface[i].X].Y + delta[1], obj->vertex[obj->surface[i].X].Z +
delta[2]);
            glVertex3f(obj->vertex[obj->surface[i].Y].X + delta[0], obj-
>vertex[obj->surface[i].Y].Y + delta[1], obj->vertex[obj->surface[i].Y].Z +
delta[2]);
            glVertex3f(obj->vertex[obj->surface[i].Y].X + delta[0], obj-
>vertex[obj->surface[i].Y].Y + delta[1], obj->vertex[obj->surface[i].Y].Z +
delta[2]);
            glVertex3f(obj->vertex[obj->surface[i].P].X + delta[0], obj-
>vertex[obj->surface[i].P].Y + delta[1], obj->vertex[obj->surface[i].P].Z +
delta[2]);
            glVertex3f(obj->vertex[obj->surface[i].P].X + delta[0], obj-
>vertex[obj->surface[i].P].Y + delta[1], obj->vertex[obj->surface[i].P].Z +
delta[2]);
            glVertex3f(obj->vertex[obj->surface[i].Q].X + delta[0], obj-
>vertex[obj->surface[i].Q].Y + delta[1], obj->vertex[obj->surface[i].Q].Z +
delta[2]);
            glVertex3f(obj->vertex[obj->surface[i].Q].X + delta[0], obj-
>vertex[obj->surface[i].Q].Y + delta[1], obj->vertex[obj->surface[i].Q].Z +
delta[2]);
```

```
            glVertex3f(obj->vertex[obj->surface[i].X].X + delta[0], obj-
>vertex[obj->surface[i].X].Y + delta[1], obj->vertex[obj->surface[i].X].Z +
delta[2]);
            glEnd();
        }
        else {
            glBegin(GL_POLYGON);
            glVertex3f(obj->vertex[obj->surface[i].X].X + delta[0], obj-
>vertex[obj->surface[i].X].Y + delta[1], obj->vertex[obj->surface[i].X].Z +
delta[2]);
            glVertex3f(obj->vertex[obj->surface[i].Y].X + delta[0], obj-
>vertex[obj->surface[i].Y].Y + delta[1], obj->vertex[obj->surface[i].Y].Z +
delta[2]);
            glVertex3f(obj->vertex[obj->surface[i].P].X + delta[0], obj-
>vertex[obj->surface[i].P].Y + delta[1], obj->vertex[obj->surface[i].P].Z +
delta[2]);
            glVertex3f(obj->vertex[obj->surface[i].Q].X + delta[0], obj-
>vertex[obj->surface[i].Q].Y + delta[1], obj->vertex[obj->surface[i].Q].Z +
delta[2]);
            glEnd();
        }
    }
}
```

输出OBJ文件，可以把保存在结构体中的OBJ文件输出到文件当中去。

```
void OutObjFile(char * s, ObjFile * letter)
{
    FILE * fin;
    fin = fopen(s, "w");
    for (int i = 1; i <= letter->vc; ++i) fprintf(fin, "v %.6f %.6f %.6f\n",
letter->vertex[i].X, letter->vertex[i].Y, letter->vertex[i].Z);
    for (int i = 1; i <= letter->sc; ++i) fprintf(fin, "f %d %d %d %d\n",
letter->surface[i].X, letter->surface[i].Y, letter->surface[i].P, letter-
>surface[i].Q);
    fclose(fin);
    fin = NULL;
}
```

NURBS曲面渲染，这段代码的主要作用是绘制一个NURBS曲面，修改控制点并且渲染出整个曲面。

```
void render_nurbs()
{
    GLfloat knots[8] = { 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0 };
    init_surface();
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture[2]);

    theNurb = gluNewNurbsRenderer();
    gluNurbsProperty(theNurb, GLU_SAMPLING_TOLERANCE, 25.0);
    gluNurbsProperty(theNurb, GLU_DISPLAY_MODE, GLU_FILL);
    gluBeginSurface(theNurb);
    glTexCoord2i(1, 1); glVertex3i(ctlpoints[3][3][0], ctlpoints[3][3][1],
ctlpoints[3][3][2]);
```

```
    glTexCoord2i(1, 0); glVertex3i(ctlpoints[3][0][0], ctlpoints[3][0][1],
ctlpoints[3][0][2]);
    glTexCoord2i(0, 0); glVertex3i(ctlpoints[0][0][0], ctlpoints[0][0][1],
ctlpoints[0][0][2]);
    glTexCoord2i(0, 1); glVertex3i(ctlpoints[0][3][0], ctlpoints[0][3][1],
ctlpoints[0][3][2]);
    gluNurbsSurface(theNurb,
        8, knots, 8, knots,
        4 * 3, 3, &ctlpoints[0][0][0],
        4, 4, GL_MAP2_VERTEX_3);
    gluEndSurface(theNurb);

    if (showPoints) {
        glPointSize(5.0);
        glDisable(GL_LIGHTING);
        glColor3f(1.0, 1.0, 0.0);
        glBegin(GL_POINTS);
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                glVertex3f(ctlpoints[i][j][0],
                    ctlpoints[i][j][1], ctlpoints[i][j][2]);
            }
        }
        glEnd();
        glEnable(GL_LIGHTING);
    }
    glPopMatrix();
    glDisable(GL_TEXTURE);
}
```

在迷宫的入口和出口绘制了门，门形状的绘制利用调用drawCube()和glRotatef()共同完成，然后进行纹理映射，并通过观察点和门的相对位置控制门的开合。

```
void drawdoor()
{
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture[4]);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glPushMatrix();
    glTranslated(0.5, 0, -0.5);
    glRotatef(doorRotate1, 0, 1, 0);
    glTranslatef(0.5, 0, 0);
    glScalef(1, 1, 0.1);
    drawCube();
    glPopMatrix();

    glPushMatrix();
    glTranslated(0.5+MAP_SIZE-3, 0, -0.5-MAP_SIZE+1);
    glRotatef(doorRotate2, 0, 1, 0);
    glTranslatef(0.5, 0, 0);
    glScalef(1, 1, 0.1);
    drawCube();
    glPopMatrix();

    glDisable(GL_TEXTURE);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_ADD);
}
```

不同界面渲染逻辑简要介绍：

用InMenu,InGame,gameover三个变量来控制渲染何种模式，不同模式采取调用不同的子函数和采取不同的视角。光源的编辑在render()函数中实现，其中light_color数组为光源颜色数组，light_pos数组为光源位置数组。光源颜色可以在鼠标右键菜单中设置，光源位置可以通过键盘调整。

```
void render()//主渲染入口
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glClearColor(0, 0, 0, 1);
    glLoadIdentity();                                    // Reset The Current
Modelview Matrix
    gluLookAt(eye[0], eye[1], eye[2],
        eye[0] + cos(faceang), eye[1], eye[2] + sin(faceang),
        0, 1, 0);               // 场景（0，0，0）的视点中心 (0, 5, 50)，Y轴向上

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);

    //开灯
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_color);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_color);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_color);
    glLightfv(GL_LIGHT0, GL_POSITION, light_pos);

    glEnable(GL_LIGHT0);

    if (MenuGameFit) {//渐变效果
        FitLookAt();
        MenuGameFit--;
    }
    else if (gameover) {//如果游戏结束
        render_gameover();
    }
    else if (InGame) {//如果在游戏当中
        render_game();
    }
    else if (InMenu) {//如果在目录上
        render_menu();
    }
    glutSwapBuffers();
}

void render_game(){...}

void render_maze(){...}

void render_menu(){...}
```

鼠标右键菜单设置：
在main函数中构建鼠标右键菜单的内容

```
//构建子菜单1的内容
    subMenu1 = glutCreateMenu(submenufunc1);
    glutAddMenuEntry("Gray", 0);
    glutAddMenuEntry("White", 1);
```

```
    glutAddMenuEntry("Red", 2);
    glutAddMenuEntry("Green", 3);
    glutAddMenuEntry("Blue", 4);
    glutAttachMenu(GLUT_RIGHT_BUTTON);

    //构建子菜单2的内容
    subMenu2 = glutCreateMenu(submenufunc2);
    glutAddMenuEntry("弱", 1);
    glutAddMenuEntry("较弱", 2);
    glutAddMenuEntry("一般", 3);
    glutAddMenuEntry("较强", 4);
    glutAddMenuEntry("强", 5);
    glutAttachMenu(GLUT_RIGHT_BUTTON);

    //构建子菜单3的内容
    subMenu3 = glutCreateMenu(submenufunc3);
    glutAddMenuEntry("水波", 1);
    glutAddMenuEntry("裂缝", 2);
    glutAttachMenu(GLUT_RIGHT_BUTTON);

    //构建子菜单4的内容
    subMenu4 = glutCreateMenu(submenufunc4);
    glutAddMenuEntry("新手模式", 1);
    glutAddMenuEntry("简单模式", 2);
    glutAddMenuEntry("一般模式", 3);
    glutAddMenuEntry("困难模式", 4);
    glutAddMenuEntry("噩梦模式", 5);
    glutAttachMenu(GLUT_RIGHT_BUTTON);

    //构建子菜单5的内容
    subMenu5 = glutCreateMenu(submenufunc5);
    glutAddMenuEntry("Gray", 0);
    glutAddMenuEntry("White", 1);
    glutAddMenuEntry("Red", 2);
    glutAddMenuEntry("Green", 3);
    glutAddMenuEntry("Blue", 4);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    //构建主菜单的内容
    mainMenu = glutCreateMenu(menufunc);

    //将两个菜单变为主菜单的子菜单
    glutAddSubMenu("光源颜色", subMenu1);
    glutAddSubMenu("光源强度", subMenu2);
    glutAddSubMenu("墙体纹理", subMenu3);
    glutAddSubMenu("游戏难度", subMenu4);
    glutAddSubMenu("终点物体材质", subMenu5);
    //点击鼠标右键时显示菜单
    glutAttachMenu(GLUT_RIGHT_BUTTON);
```

针对每个子菜单编写对应的响应函数，实现光源颜色、光源强度、墙体纹理、游戏难度、终点物体材质的编辑。由于代码过长，本文档中只给出了响应函数的函数头。

```
//第一个子菜单响应函数
void submenufunc1(int data) {}

//第二个子菜单响应函数
void submenufunc2(int data) {}
```

```
//第三个子菜单响应函数
void submenufunc3(int data) {}

//第四个子菜单响应函数
void submenufunc4(int data) {}

//第五个子菜单响应函数
void submenufunc5(int data) {}
```

在main函数中通过调用PlaySound播放游戏背景音乐。

```
PlaySound(TEXT(".\\rise.wav"), NULL, SND_FILENAME | SND_ASYNC | SND_LOOP);
```

总结

从本次图形学大作业中，我们以三维迷宫游戏为载体，实践了在计算机图形学课程中学到的各种相关知识，如建模表达基本体素，纹理映射，三维观察，光照和材质，平移、旋转和缩放的基本几何变换，NURBS曲面的生成，三维网格的导入导出，表达了门，墙等对象要素以及漫游时实时进行碰撞检测。

在大作业的完成过程中，三名队友通力合作，都做出了很大的贡献。

最后，感谢童若锋老师由浅入深的教导，以及三位助教在实验过程中给予的指导，通过计算机图形课程的学习，我们收获良多。