专业：__数字媒体技术__
姓名：_____叶怡轩_____
学号：__3170101790__
日期：___2019-06-14___
地点：_____紫金港_____

浙江大学实验报告

课程名称：____计算机图形学___ 指导老师：_____唐敏_____成绩：_____
实验名称：___贪吃人___ 实验类型：_____大作业_____同组学生姓名：_ 郑昌熙_楼晨阳 _

装 订 线

一、实验目的和要求

基于课程对 OpenGL 的学习，完成一个在庭院里吃东西的“贪吃人”游戏。

二、游戏操作与介绍

1. **按键说明：**
   ASD：人物左转，反向和右转。
   ER：扩大、缩小聚光灯视角。
   VB：增强、减弱环境光。
   IJKLNM：聚光光源（月亮）各个方向的移动。
   O：输出 obj 文件。
   空格：暂停
2. **游戏规则：**
   视角和人物会按照默认的方向按一定的速度移动，通过键盘操控来改变人物和视角的移动方向。目标是吃到更多的食物。食物被吃了以后会随机刷新。每吃到一个食物之后，移动速度会有提升。如果碰到建筑物或者栅栏边界即为撞墙死亡。游戏结束。

三、主要仪器设备

Visual Studio C++
glut.zip

四、分工

1. 叶怡轩：obj 文件导入导出，光源编辑（环境光和聚光灯），游戏模式框架搭建，Nurbs 曲面建模，部分场景建模，实验报告。

2. 郑昌熙：部分场景建模，贴图寻找以及导入使用，视角的移动和旋转。

3. 楼晨阳：截屏，场景布置和效果调调整，部分场景建模，碰撞检测

五、实验数据记录和处理

1.obj 文件导入导出

```
class obj3dmodel {

    struct vertex {//点坐标
        double x, y, z;
    };
    struct texvertex {//纹理坐标
        double x, y;
    };
    struct face {
        unsigned int v1, v2, v3, v4;
    };
    struct texface {
        unsigned int v1, v2, v3, v4;
    };
    vector<texface>texfaces;
    vector<vertex>vertexs;
    vector<face>faces;
    vector<texvertex>texvertexs;

public:
    char ss[20] = "1.obj";
    void parse(const char *filename);
    void draw();
    void output();
};
```

对于一个存储三角面 obj 文件中的所有信息都存储在 obj3jiao 类中，四角面的 obj 文件存储在 obj3dmodel 类中。

对于该 obj 文件的绘制、导出、读入都是该类的成员函数。

2.光源编辑：

```
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
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);

glLightfv(GL_LIGHT1, GL_AMBIENT, white);    //设置环境光成分
glLightfv(GL_LIGHT1, GL_SPECULAR, white);  //设置镜面光成分
glLightfv(GL_LIGHT1, GL_DIFFUSE, white);  //设置漫射光成分

glLightfv(GL_LIGHT1, GL_POSITION, spot_pos);
glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, lightangle);            //裁减角度
glLightfv(GL_LIGHT1, GL_SPOT_DIRECTION, shoot_angle);          //光源方向
glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 2.);                    //聚集度

//glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);    // c 系数
//glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);       // l 系数
//glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);   // q 系数


glEnable(GL_LIGHTING);
glEnable(GL_LIGHT1);//开启聚集光源
```

上半部分是环境光源，下半部分是聚光灯。

其中聚光灯的位置与月亮重合（让月亮作为聚光灯）

3.游戏框架

通过 judge()来判断人物是否吃到食物，是否撞到建筑物（死亡）

通过 changefood()在允许的范围内随机生成食物。

4.nurbs 曲面建模

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glPushMatrix();
glEnable(GL_AUTO_NORMAL);
glEnable(GL_NORMALIZE);
glDepthFunc(GL_LESS);    //娣卞害娴嬭瘯

glEnable(GL_DEPTH_TEST);

glEnable(GL_AUTO_NORMAL);
//允许正则化法矢量
glEnable(GL_NORMALIZE);
theNurb = gluNewNurbsRenderer(); // 创建一个NURBS曲面对象
//修改NURBS曲面对象的属性——glu库函数
////采样sampling容错torerance
gluNurbsProperty(theNurb, GLU_SAMPLING_TOLERANCE, 5.0);
gluNurbsProperty(theNurb, GLU_DISPLAY_MODE, GLU_FILL);
```

```
GLfloat ctrlpoints[4][4][3] = {
    { { -1.5, -1.5, 2.0 },
    { -0.5, -1.5, 2.0 },
    { 0.5, -1.5, -1.0 },
    { 1.5, -1.5, 2.0 } },

    { { -1.5, -0.5, 1.0 },
    { -0.5, 1.5, 2.0 },
    { 0.5, 0.5, 1.0 },
    { 1.5, -0.5, -1.0 } },

    { { -1.5, 0.5, 2.0 },
    { -0.5, 0.5, 1.0 },
    { 0.5, 0.5, 3.0 },
    { 1.5, -1.5, 1.5 } },

    { { -1.5, 1.5, -2.0 },
    { -0.5, 1.5, -2.0 },
    { 0.5, 0.5, 1.0 },
    { 1.5, 1.5, -1.0 } } };
```

```
//////////////////////////////////////////////////////曲面开始
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_2D);
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
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, texture[11]);
glColor3f(0.0, 1.0, 0.0);
//各控制点影响力参数设置
GLfloat knots[8] = { 0.0, 0.0, 0.0, 0.0,
    1.0, 1.0, 1.0, 1.0 }; // NURBS曲面的控制向量


glRotatef(1.0, 0.7, -0.6, 1.0); // 旋转变换

glPushMatrix();
glScaled(0.5, 0.5, 0.5);
gluBeginSurface(theNurb); // 开始曲面绘制
//网络查询：参数GL_MAP2_VERTEX_3的作用？
//将

gluNurbsSurface(theNurb, 8, knots, 8, knots, 4 * 3, 3, &ctrlpoints[0][0][0], 4, 4, GL_MAP2_VERTEX_3); // 定义曲面的数学模型，确定其形状
gluEndSurface(theNurb); // 结束曲面绘制
glPopMatrix();
glDisable(GL_TEXTURE_GEN_S);
glDisable(GL_TEXTURE_GEN_T);

glDisable(GL_TEXTURE_2D);
```

首先申请一个 nurbs 曲面指针，然后定义一个控制点数组来控制 nurbs 曲面的特性。

由于场景的建模主要是由 maya 完成，与课程涉及的内容以及关系较少，故不再赘述，贴图的寻找也类似。

下面主要描述关于贴图的导入和使用以及视角的移动和旋转。

贴图的导入函数:

类似于实验 6 中的部分，首先在 load()函数中调用 glGenTextures(15, texture);并在全局程序开始的地方定义全局变量的 GLU INT 数组 texture[15],而后再逐个调用 texload(texnum, filename);读入纹理图片。texload 本身与实验 6 中部分一样

```
    BITMAPINFOHEADER bitmapInfoHeader;
// bitmap 信息头
    unsigned char*   bitmapData;
// 纹理数据
    bitmapData = LoadBitmapFile(filename, &bitmapInfoHeader);
    glBindTexture(GL_TEXTURE_2D, texture[i]);
    // 指定当前纹理的放大/缩小过滤方式
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D,
        0,          //mipmap 层次(通常为，表示最上层)
        GL_RGB,   //我们希望该纹理有红、绿、蓝数据
        bitmapInfoHeader.biWidth, //纹理宽带，必须是 n，若有边框+2
        bitmapInfoHeader.biHeight, //纹理高度，必须是 n，若有边框+2
        0, //边框(0=无边框, 1=有边框)
        GL_RGB,   //bitmap 数据的格式
        GL_UNSIGNED_BYTE, //每个颜色数据的类型
        bitmapData);    //bitmap 数据指针
```

首先定义一个之前已经定义过的 bmp 文件头以及存放数据内容的指针，并通过 loadbitmapfile 函数读入 bmp 文件的内容并返回文件的指针。在读入后，再将该图片内容与纹理进行绑定。而后便可以在 display（）函数中，通过进行纹理与内容的绑定进行使用纹理。

首先 glEnable(GL_TEXTURE_GEN_S);

glEnable(GL_TEXTURE_GEN_T);

glEnable(GL_TEXTURE_2D);

glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);

glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);

并在需要绑定纹理的部分之前加上：

glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

glBindTexture(GL_TEXTURE_2D, texture[?]);其中 texture[?]为需要使用的纹理。

但是该读取纹理函数在读取过大的文件的时候回出现错误，经猜测可能是 opengl 不支持过大的纹理。

也是由于这个原因，我将 bmp 图片都另外村成了较小的尺寸，如 240*160 大小。

视点移动和视角变化部分：

首先在 display 函数中有 glulookat 函数,这个函数确定了视点所看的方向，同时能够根据其中内容的改变做出相应的调整。

其中，前三个参数为 eye[0],eye[1],eye[2]分别为视点的 x,y,z 位置，中间三个参数为 pos[0],pos[1],pos[2],为所看位置的 x,y,z 位置，最后三个参数为，0,0,1,表明看的方向是正向的，所以相应的只要根据一些操作改变视点和所看物体的坐标的位置，即可达到改变位置和方向的效果。

具体如下：

关于键盘的操作上在 key 调用函数中加入，实现对于世界角度上的上下左右移动

```
case 'w':
    eye[1] += 1.0;
    pos[1] += 1.0;
    break;
case 'd':
    moveangle += 5;
    if (moveangle > 360) {
        moveangle -= 360;
    }
    break;
case 'a':
    moveangle -= 5;
        if (moveangle < 0) {
            moveangle += 360;
        }
    break;
case 's':
    moveangle -= 180;
    if (moveangle < 0) {
        moveangle += 360;
    }
    break;
```

```
    case 'z':
        eye[2] -= 1.0;
        pos[2] -= 1.0;
        cz -= 1;
        break;
    case 'c':
        eye[2] += 1.0;
        pos[2] += 1.0;
        cz += 1;
          break;
```

关于鼠标移动视角的调用：

定义全局变量

```
bool mouseLeftDown;
bool mouseRightDown;
bool mousemidDown;
float mouseX, mouseY;
float cameraDistance;
float cameraAngleX;
float cameraAngleY;
```

关于鼠标的参数

```
void mouseCB(int button, int state, int x, int y)//关于鼠标状态的判断
```

的函数，x,y 表示鼠标当前所在位置，button 是左键，右键或者是中键，在 opengl

中被定义为特殊的宏，state 为状态，即按下或者是松开的状态。

```
{
    mouseX = x;
    mouseY = y;

    if (button == GLUT_LEFT_BUTTON)
    {
        if (state == GLUT_DOWN)
        {
            mouseLeftDown = true;
        }
        else if (state == GLUT_UP)
            mouseLeftDown = false;
    }

    else if (button == GLUT_RIGHT_BUTTON)
    {
        if (state == GLUT_DOWN)
        {
            mouseRightDown = true;
        }
        else if (state == GLUT_UP)
```

```
            mouseRightDown = false;
        }
        else if (button == GLUT_MIDDLE_BUTTON)
        {
            if (state == GLUT_DOWN)
            {
                mousemidDown = true;
            }
            else if (state == GLUT_UP)
                mousemidDown = false;
        }
}

void mouseMotionCB(int x, int y)//关于鼠标移动操作视点的函数
{
    if (mouseLeftDown)//当鼠标左键按下时
    {
        cameraAngleX += (x - mouseX);//判断 x 方向上走过的角度
        pos[0] = sin(cameraAngleX*3.14 / 180) * 120 + eye[0];//以视点为圆心，120 为半径，将视点 x 移动相应的角度
        cameraAngleY += (y - mouseY);//判断 y 方向上走过的角度
        pos[1] = cos(cameraAngleY*3.14 / 180) * 120 + eye[1];
```

```
        pos[2] = eye[2] - cos(cameraAngleY*3.14 / 180) * 120 -
cos(cameraAngleX*3.14 / 180) * 120;;//对于 y 的操作也是类似的
        mouseX = x;//更新当前的 mousex 和 mousey
        mouseY = y;
    }

    glutPostRedisplay();
    }
```

至此关于鼠标移动的函数告一段落

只需要在主函数中注册并调用这两个函数即可

```
glutMouseFunc(mouseCB);
glutMotionFunc(mouseMotionCB);
```

即可通过移动鼠标改变视角。

截屏功能模块：

```
#define WindowWidth  800
#define WindowHeight 600

#define BMP_Header_Length 54
void grab(void)
{
    FILE*    pDummyFile;  //指向另一bmp文件，用于复制它的文件头和信息
```

头数据

```
    FILE*    pWritingFile;  //指向要保存截图的bmp文件
    GLubyte* pPixelData;    //指向新的空的内存，用于保存截图bmp文件数
据
    GLubyte  BMP_Header[BMP_Header_Length];
    GLint    i, j;
    GLint    PixelDataLength;   //BMP文件数据总长度

                        // 计算像素数据的实际长度
    i = WindowWidth * 3;   // 得到每一行的像素数据长度
    while (i % 4 != 0)       // 补充数据，直到i是的倍数
       ++i;
    PixelDataLength = i * WindowHeight;  //补齐后的总位数

                              // 分配内存和打开文件
    pPixelData = (GLubyte*)malloc(PixelDataLength);
    if (pPixelData == 0)
       exit(0);

    pDummyFile = fopen("starsky.bmp", "rb");//只读形式打开
    if (pDummyFile == 0)
       exit(0);
```

```
    pWritingFile = fopen("grab.bmp", "wb"); //只写形式打开
    if (pWritingFile == 0)
        exit(0);

    //把读入的bmp文件的文件头和信息头数据复制，并修改宽高数据
    fread(BMP_Header, sizeof(BMP_Header), 1, pDummyFile);  //读取文件头和信息头，占据54字节
    fwrite(BMP_Header, sizeof(BMP_Header), 1, pWritingFile);
    fseek(pWritingFile, 0x0012, SEEK_SET); //移动到0X0012处，指向图像宽度所在内存
    i = WindowWidth;
    j = WindowHeight;
    fwrite(&i, sizeof(i), 1, pWritingFile);
    fwrite(&j, sizeof(j), 1, pWritingFile);

    // 读取当前画板上图像的像素数据
    glPixelStorei(GL_UNPACK_ALIGNMENT, 4);  //设置4位对齐方式
    glReadPixels(0, 0, WindowWidth, WindowHeight,
        GL_BGR_EXT, GL_UNSIGNED_BYTE, pPixelData);

    // 写入像素数据
```

```
    fseek(pWritingFile, 0, SEEK_END);
    //把完整的BMP文件数据写入pWritingFile
    fwrite(pPixelData, PixelDataLength, 1, pWritingFile);

    // 释放内存和关闭文件
    fclose(pDummyFile);
    fclose(pWritingFile);
    free(pPixelData);
}
```

截屏功能的实现利用了已有的 bmp 文件，通过已有的 bmp 文件获取并复制文件头和信息头，写入到新创建的截屏文件中，然后用 glReadPixels 函数从帧缓冲区中读取数据，前两个参数指定从帧缓冲区读取的第一个像素的窗口坐标，三、四两个参数指定读取像素矩形的尺寸，第五个参数指定像素数据的格式，第六个参数指定像素数据的数据类型，最后一个参数指定了返回像素数据的地址。然后将读取出的图像数据保存到文件中。

六、实验结果与分析

七、讨论、心得

本实验我们综合了所有的本学期所学的 **OpenGL** 知识。之前对于个别知识点还有一些模糊不清，但是在做这个 **project** 的过程中都一一明确了。除了对于已学知识的应用，我们还自学了 **nurbs** 曲面建模、**obj** 导入导出、漫游之类的知识，并运用到我们的实验中。感觉实验让我们收获颇丰，很有成就感！