实 验 报 告

	学　　号
	姓　　名
（实验）	课程名称
	理论教师
	实验教师

实　验　报　告

一、　实验目的：

1) 掌握 OpenGL 环境配置方法；
2) 掌握在空间中绘图的基本方法；
3) 掌握移动物体的基本方法；
4) 掌握键盘事件响应的方法；
5) 掌握投影变换的方法；
6) 掌握照相机移动方法；
7) 掌握颜色、材料和光照的基础知识；
8) 掌握生成简单动画的方法；
9) 掌握键盘事件响应的方法。

二、　实验原理：

OpenGL 是一个 3D 图形和模型库，具有高度的可移植性，并且具有非常快的速度。使用 OpenGL，可以创建优质的 3D 图像。OpenGL 并不像 C 或 C++一样是门编程语言，它更像一个 C 运行时调用的函数库，提供一些预先设定好的功能。程序员可以使用任何语言来编写动画程序，在需要某项绘图功能时调用 OpenGL 所提供的函数即可。因此，当我们提到一个基于 OpenGL 的程序或者一个 OpenGL 应用程序时，实际是指这个程序是用其他编程语言(C,C++,java 等)编写，但它调用了 OpenGL 函数库。

三、　实验内容：

1) 设置编程环境，使用 GLUT；
2) 编写第一个程序:绘制一个三角形；
3) 实现可变窗口大小；
4) 生成一个简单动画，三角形绕固定轴匀速旋转；

   要求：键盘控制三角形颜色和旋转速度；
5) 实现一个反弹运动的正方形的动画；
6) 仿照 5),生成一个可以反弹运动的三角形动画，键盘控制三角形移动速度、颜色实现一个立方体的正投影和透视投影；
7) 实现移动镜头的应用 – 绘制一个雪人世界；

8) 绘制一个三角形，顶点设置不同颜色，对比两种着色模式 -GL_SMOOTH 和 GL_FLAT；
9) 绘制一个旋转的立方体，并向场景中添加光源：环境光、散射光；
10) 采用两种不同方法为在上例中给立方体设置材质属性；
11) 为雪人世界添加光照、设置材质属性；
12) 为 9) 中的立方体添加纹理；

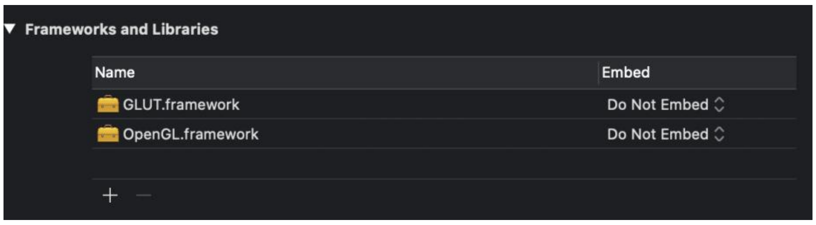
四、 实验器材（设备、元器件）：

个人电脑一台(MacOS,xcode)

五、 实验步骤：

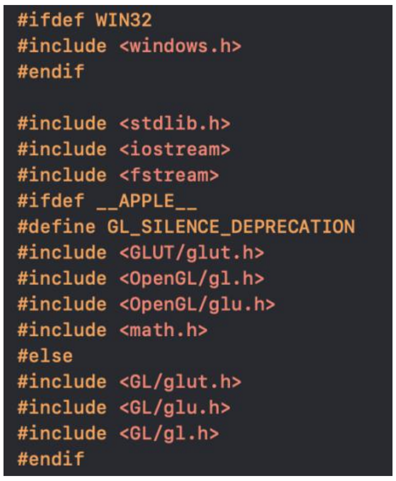
1）生成一个简单动画，三角形绕固定轴匀速旋转；

● 设置编程环境，使用 GLUT；



▼ Frameworks and Libraries

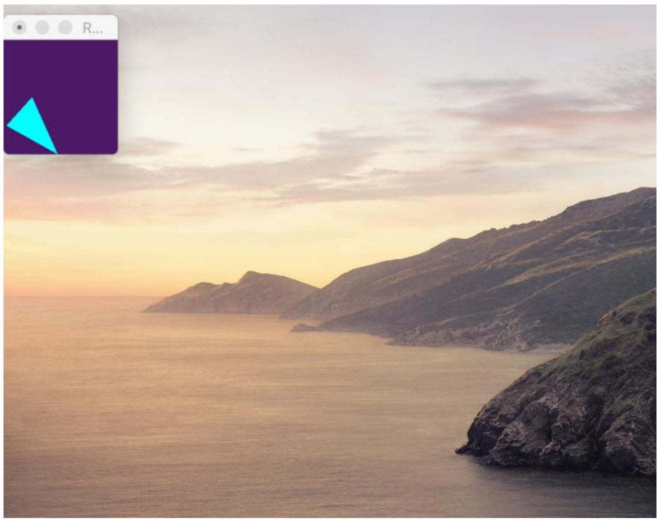
Name	Embed
GLUT.framework	Do Not Embed
OpenGL.framework	Do Not Embed

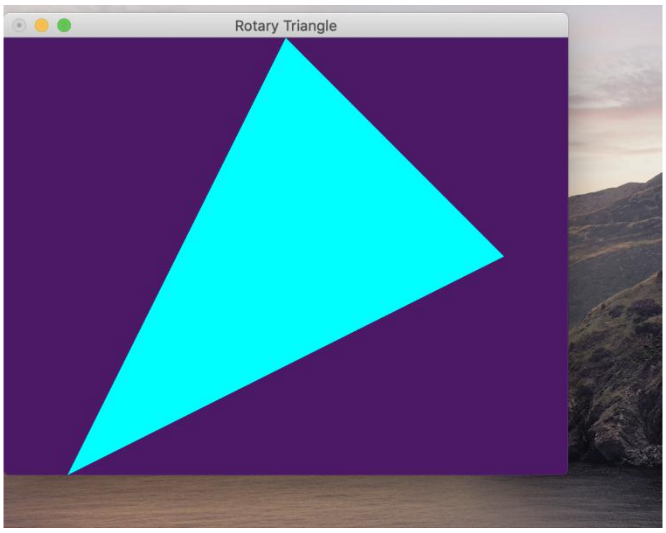


```
#ifdef WIN32
#include <windows.h>
#endif

#include <stdlib.h>
#include <iostream>
#include <fstream>
#ifdef __APPLE__
#define GL_SILENCE_DEPRECATION
#include <GLUT/glut.h>
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#include <math.h>
#else
#include <GL/glut.h>
#include <GL/glu.h>
#include <GL/gl.h>
#endif
```

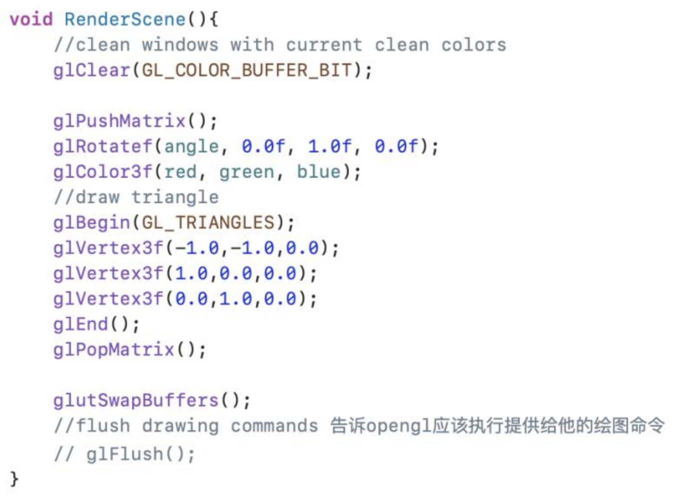
- 实现可变窗口大小;

```
void ChangeSize(int w,int h){
    GLfloat aspectRatio;

    if (h == 0) h = 1;

    glViewport(0, 0, w, h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    aspectRatio = (GLfloat)w/(GLfloat)h;

    if(w <= h){

        glOrtho(-1.0, 1.0, -1.0/aspectRatio, 1.0/aspectRatio, 1.0, -1.0);

    }
    else{
        windowWidth = 1.0 * aspectRatio;
        windowHeight = 1.0;
        glOrtho(-windowWidth, windowWidth, -1.0, 1.0, 1.0, -1.0);
    }

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

}
```



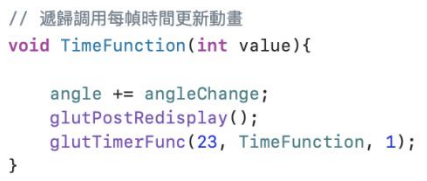


● 渲染函数



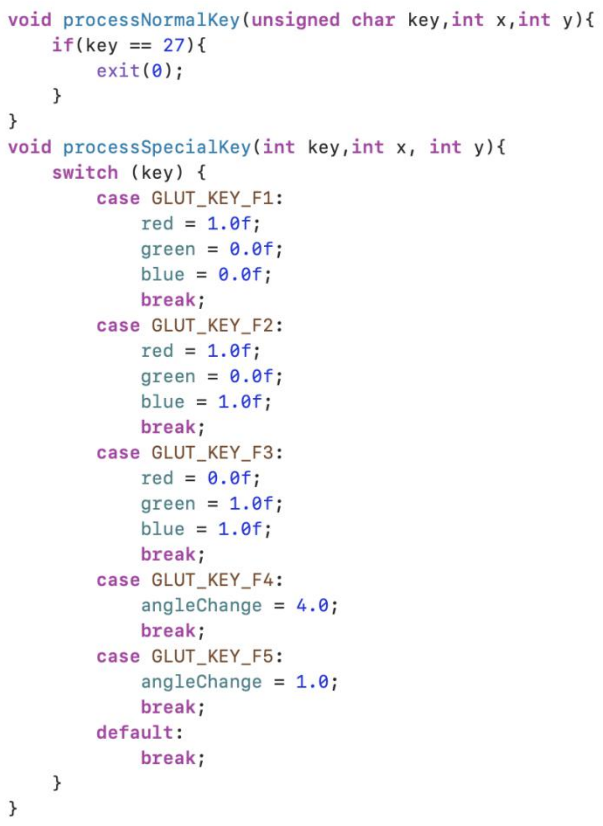
```
void RenderScene(){
    //clean windows with current clean colors
    glClear(GL_COLOR_BUFFER_BIT);

    glPushMatrix();
    glRotatef(angle, 0.0f, 1.0f, 0.0f);
    glColor3f(red, green, blue);
    //draw triangle
    glBegin(GL_TRIANGLES);
    glVertex3f(-1.0,-1.0,0.0);
    glVertex3f(1.0,0.0,0.0);
    glVertex3f(0.0,1.0,0.0);
    glEnd();
    glPopMatrix();

    glutSwapBuffers();
    //flush drawing commands 告诉opengl应该执行提供给他的绘图命令
    // glFlush();
}
```

● 定时更新函数



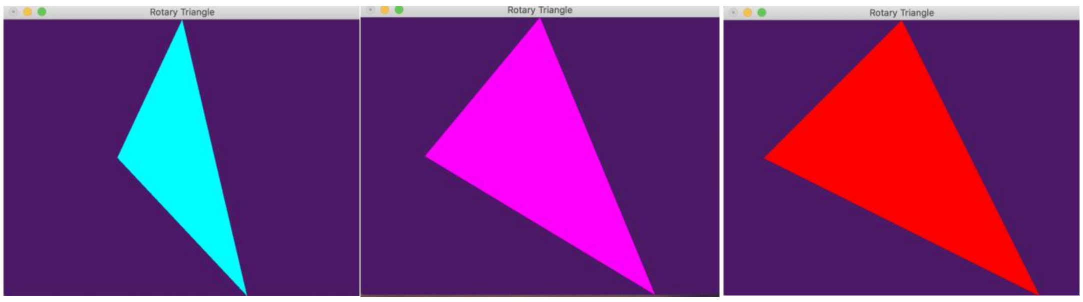
```
// 遞歸調用每幀時間更新動畫
void TimeFunction(int value){

    angle += angleChange;
    glutPostRedisplay();
    glutTimerFunc(23, TimeFunction, 1);
}
```

● 键盘处理事件（接受键盘事件更改三角形的颜色和旋转速度）



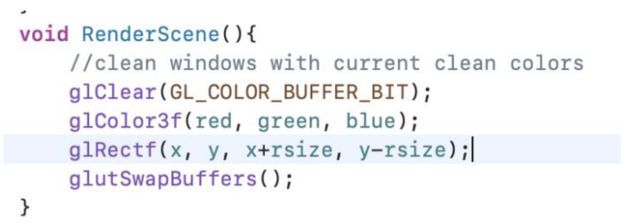
```
void processNormalKey(unsigned char key,int x,int y){
    if(key == 27){
        exit(0);
    }
}
void processSpecialKey(int key,int x, int y){
    switch (key) {
        case GLUT_KEY_F1:
            red = 1.0f;
            green = 0.0f;
            blue = 0.0f;
            break;
        case GLUT_KEY_F2:
            red = 1.0f;
            green = 0.0f;
            blue = 1.0f;
            break;
        case GLUT_KEY_F3:
            red = 0.0f;
            green = 1.0f;
            blue = 1.0f;
            break;
        case GLUT_KEY_F4:
            angleChange = 4.0;
            break;
        case GLUT_KEY_F5:
            angleChange = 1.0;
            break;
        default:
            break;
    }
}
```

效果：



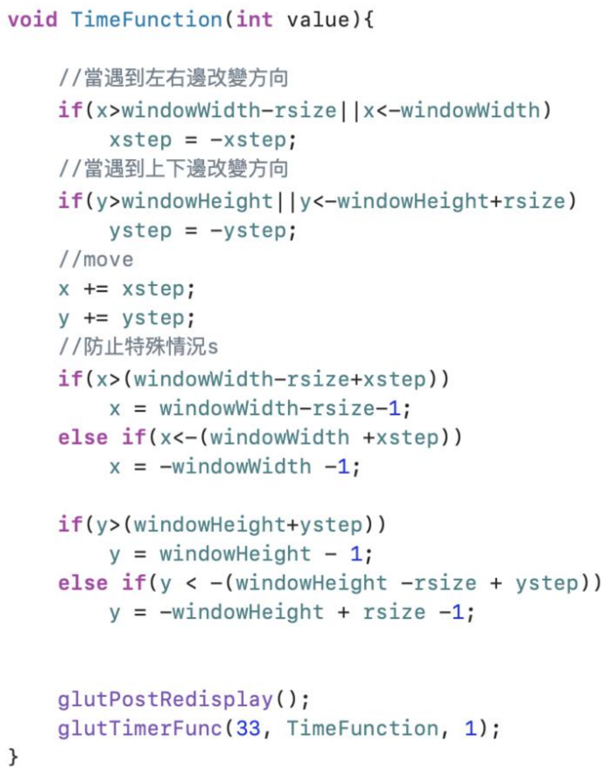
2）实现一个反弹运动的正方形的动画；

- 渲染函数



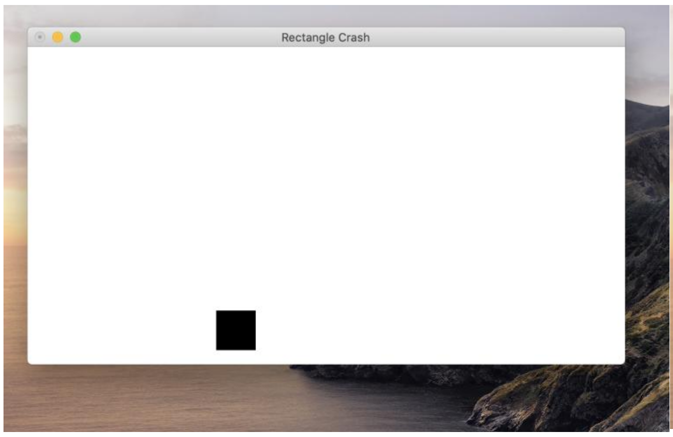
```
void RenderScene(){
    //clean windows with current clean colors
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(red, green, blue);
    glRectf(x, y, x+rsize, y-rsize);
    glutSwapBuffers();
}
```

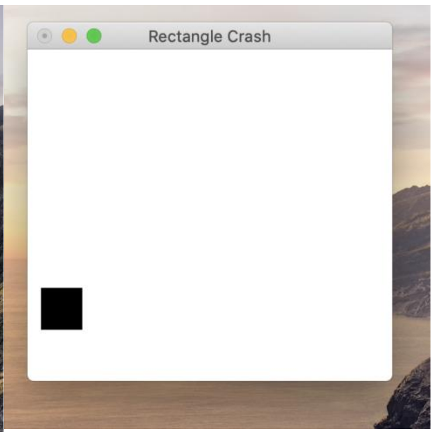
- 时间调用函数（当正方形碰壁回弹）



```
void TimeFunction(int value){

    //當遇到左右邊改變方向
    if(x>windowWidth-rsize||x<-windowWidth)
        xstep = -xstep;
    //當遇到上下邊改變方向
    if(y>windowHeight||y<-windowHeight+rsize)
        ystep = -ystep;
    //move
    x += xstep;
    y += ystep;
    //防止特殊情況s
    if(x>(windowWidth-rsize+xstep))
        x = windowWidth-rsize-1;
    else if(x<-(windowWidth +xstep))
        x = -windowWidth -1;

    if(y>(windowHeight+ystep))
        y = windowHeight - 1;
    else if(y < -(windowHeight -rsize + ystep))
        y = -windowHeight + rsize -1;


    glutPostRedisplay();
    glutTimerFunc(33, TimeFunction, 1);
}
```

效果：





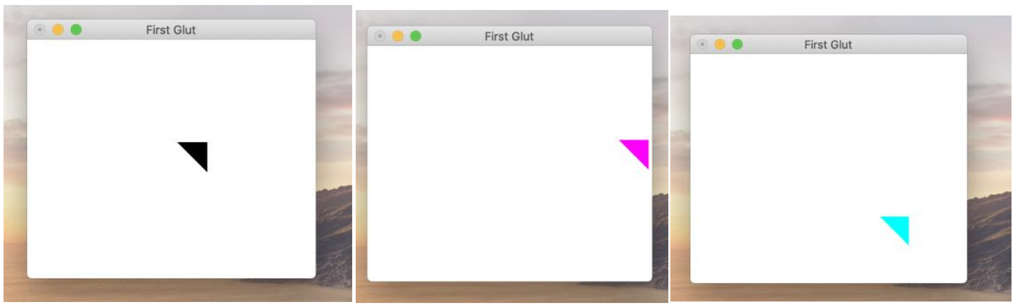
3) 仿照 2),生成一个可以反弹运动的三角形动画,键盘控制三角形移动速度、颜色实现一个立方体的正投影和透视投影;

- 渲染函数

```
void RenderScene(){
    //clean windows with current clean colors
    glClear(GL_COLOR_BUFFER_BIT);

    glPushMatrix();

    glColor3f(red, green, blue);
    //draw triangle
    glBegin(GL_TRIANGLES);
    glVertex3f(x+rsize,y,0);
    glVertex3f(x,y,0.0);
    glVertex3f(x+rsize,y-rsize,0.0);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();

    //flush drawing commands 告诉opengl应该执行提供给他的绘图命令
    // glFlush();
}
```

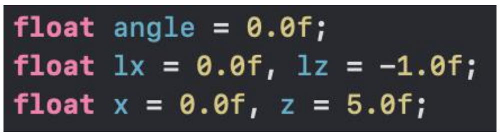
- 键盘响应函数（更改移动速度和颜色）

```
void processNormalKey(unsigned char key,int x,int y){
    if(key == 27){
        exit(0);
    }
}
void processSpecialKey(int key,int x, int y){
    switch (key) {
        case GLUT_KEY_F1:
            red = 1.0f;
            green = 0.0f;
            blue = 0.0f;
            break;
        case GLUT_KEY_F2:
            red = 1.0f;
            green = 0.0f;
            blue = 1.0f;
            break;
        case GLUT_KEY_F3:
            red = 0.0f;
            green = 1.0f;
            blue = 1.0f;
            break;
        case GLUT_KEY_F4:
            if(xstep >0)xstep = 2.0f;
            else xstep = -2.0f;
           if(ystep >0)ystep = 2.0f;
            else ystep = -2.0f;
            break;
        case GLUT_KEY_F5:
           if(xstep >0)xstep = 3.0f;
            else xstep = -3.0f;
            if(ystep >0)ystep = 3.0f;
            else ystep = -3.0f;
            break;
        default:
            break;
    }
}
```



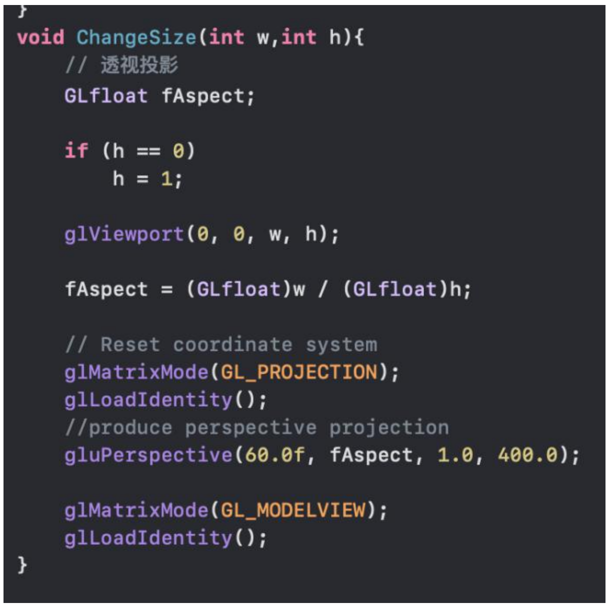
4) 实现移动镜头的应用 – 绘制一个雪人世界;

● 初始化变量



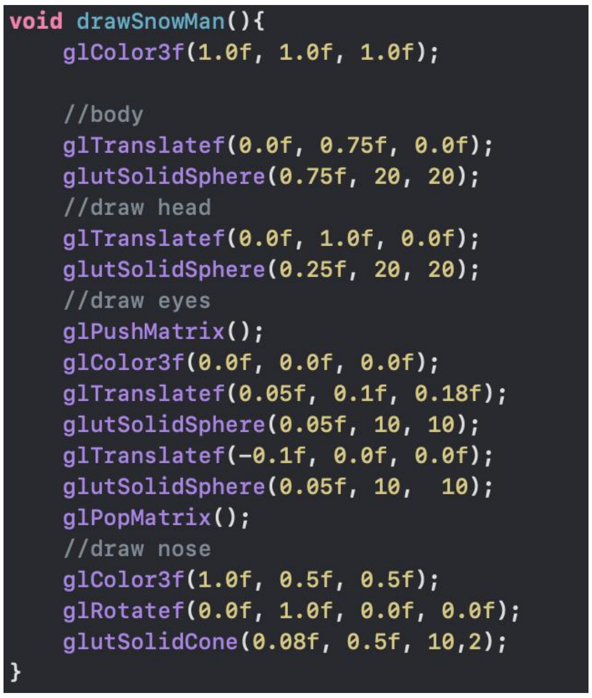
```
float angle = 0.0f;
float lx = 0.0f, lz = -1.0f;
float x = 0.0f, z = 5.0f;
```

● 透视投影



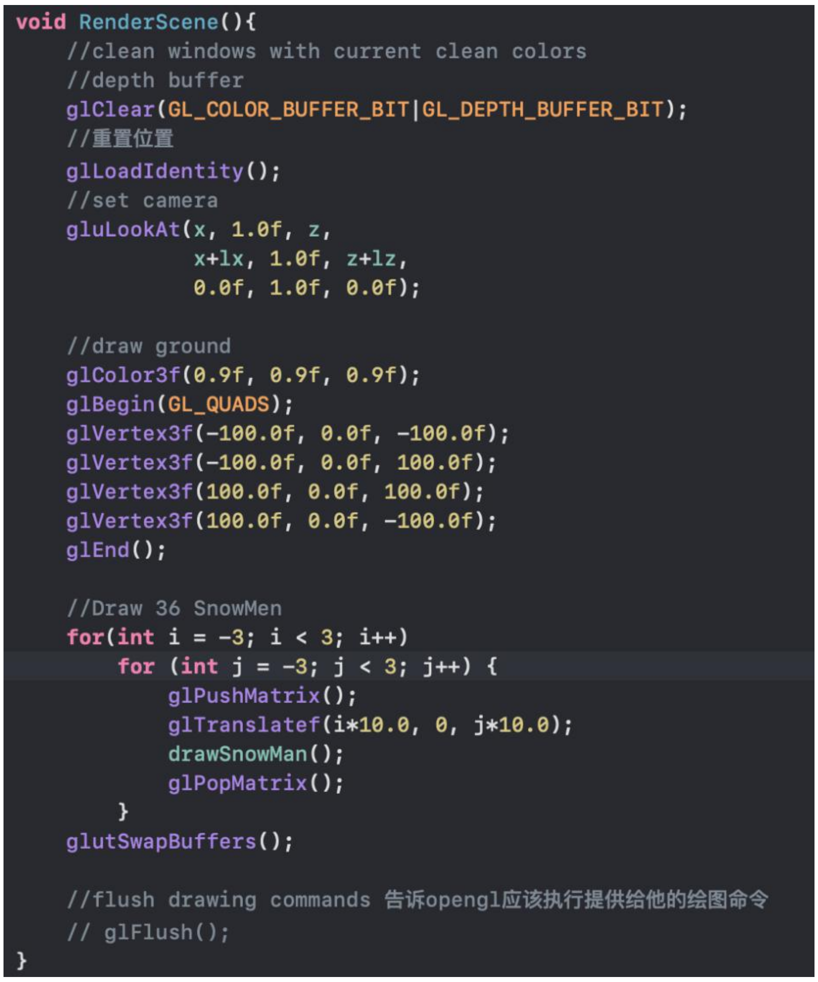
```
}
void ChangeSize(int w,int h){
    // 透视投影
    GLfloat fAspect;

    if (h == 0)
        h = 1;

    glViewport(0, 0, w, h);

    fAspect = (GLfloat)w / (GLfloat)h;

    // Reset coordinate system
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    //produce perspective projection
    gluPerspective(60.0f, fAspect, 1.0, 400.0);

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

● 绘制雪人世界



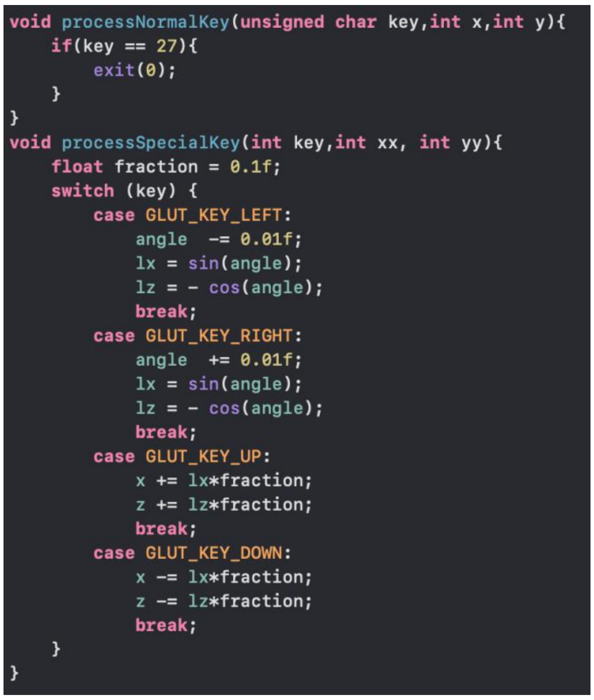
```
void drawSnowMan(){
    glColor3f(1.0f, 1.0f, 1.0f);

    //body
    glTranslatef(0.0f, 0.75f, 0.0f);
    glutSolidSphere(0.75f, 20, 20);
    //draw head
    glTranslatef(0.0f, 1.0f, 0.0f);
    glutSolidSphere(0.25f, 20, 20);
    //draw eyes
    glPushMatrix();
    glColor3f(0.0f, 0.0f, 0.0f);
    glTranslatef(0.05f, 0.1f, 0.18f);
    glutSolidSphere(0.05f, 10, 10);
    glTranslatef(-0.1f, 0.0f, 0.0f);
    glutSolidSphere(0.05f, 10,  10);
    glPopMatrix();
    //draw nose
    glColor3f(1.0f, 0.5f, 0.5f);
    glRotatef(0.0f, 1.0f, 0.0f, 0.0f);
    glutSolidCone(0.08f, 0.5f, 10,2);
}
```

● 渲染函数



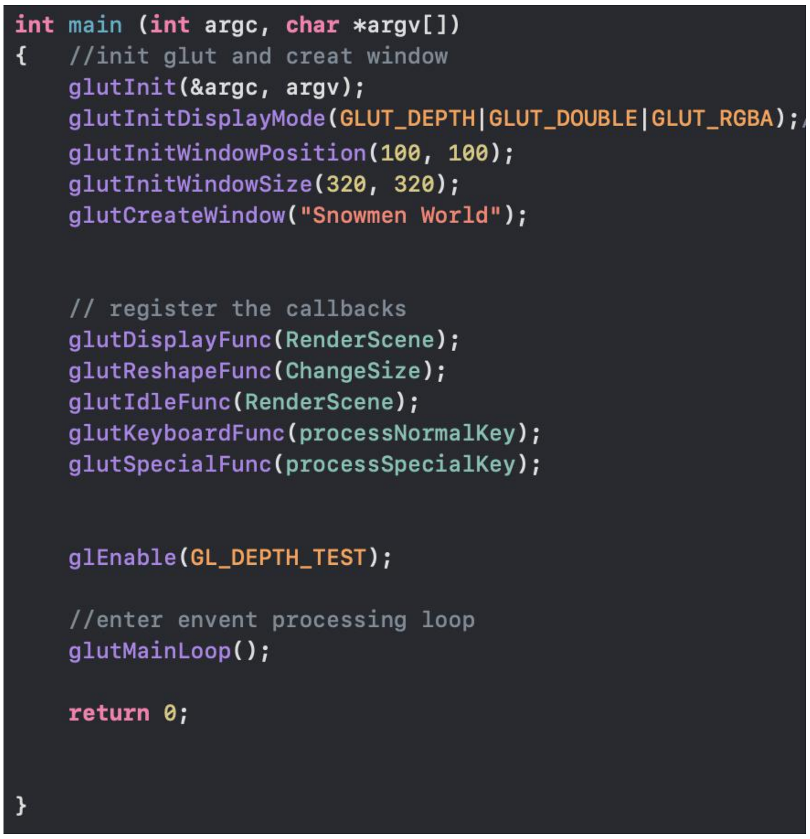
```
void RenderScene(){
    //clean windows with current clean colors
    //depth buffer
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    //重置位置
    glLoadIdentity();
    //set camera
    gluLookAt(x, 1.0f, z,
              x+lx, 1.0f, z+lz,
              0.0f, 1.0f, 0.0f);

    //draw ground
    glColor3f(0.9f, 0.9f, 0.9f);
    glBegin(GL_QUADS);
    glVertex3f(-100.0f, 0.0f, -100.0f);
    glVertex3f(-100.0f, 0.0f, 100.0f);
    glVertex3f(100.0f, 0.0f, 100.0f);
    glVertex3f(100.0f, 0.0f, -100.0f);
    glEnd();

    //Draw 36 SnowMen
    for(int i = -3; i < 3; i++)
        for (int j = -3; j < 3; j++) {
            glPushMatrix();
            glTranslatef(i*10.0, 0, j*10.0);
            drawSnowMan();
            glPopMatrix();
        }
    glutSwapBuffers();

    //flush drawing commands 告诉opengl应该执行提供给他的绘图命令
    // glFlush();
}
```

- 键盘调用函数（上下左右键控制视角左右前后移动）



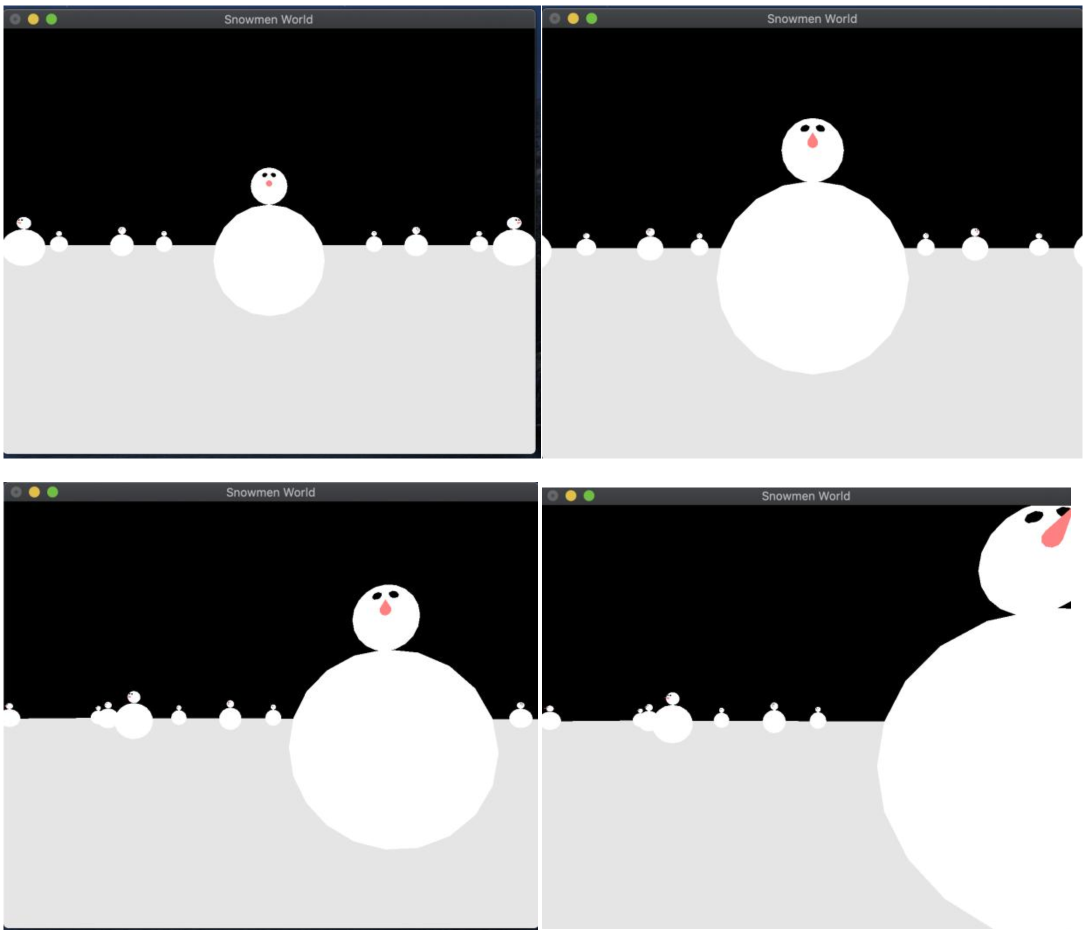
```
void processNormalKey(unsigned char key,int x,int y){
    if(key == 27){
        exit(0);
    }
}
void processSpecialKey(int key,int xx, int yy){
    float fraction = 0.1f;
    switch (key) {
        case GLUT_KEY_LEFT:
            angle  -= 0.01f;
            lx = sin(angle);
            lz = - cos(angle);
            break;
        case GLUT_KEY_RIGHT:
            angle  += 0.01f;
            lx = sin(angle);
            lz = - cos(angle);
            break;
        case GLUT_KEY_UP:
            x += lx*fraction;
            z += lz*fraction;
            break;
        case GLUT_KEY_DOWN:
            x -= lx*fraction;
            z -= lz*fraction;
            break;
    }
}
```

- main 函数



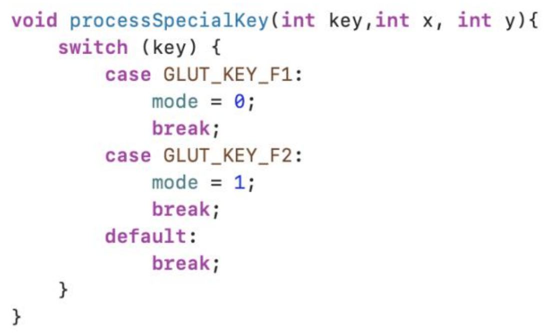
```
int main (int argc, char *argv[])
{   //init glut and creat window
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH|GLUT_DOUBLE|GLUT_RGBA);/
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(320, 320);
    glutCreateWindow("Snowmen World");


    // register the callbacks
    glutDisplayFunc(RenderScene);
    glutReshapeFunc(ChangeSize);
    glutIdleFunc(RenderScene);
    glutKeyboardFunc(processNormalKey);
    glutSpecialFunc(processSpecialKey);


    glEnable(GL_DEPTH_TEST);

    //enter envent processing loop
    glutMainLoop();

    return 0;


}
```

效果:

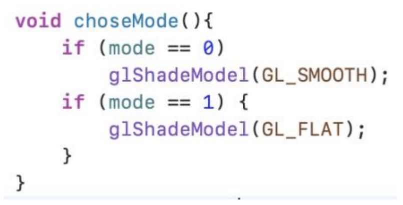


8)绘制一个三角形，顶点设置不同颜色，对比两种着色模式 - GL_SMOOTH 和 GL_FLAT;

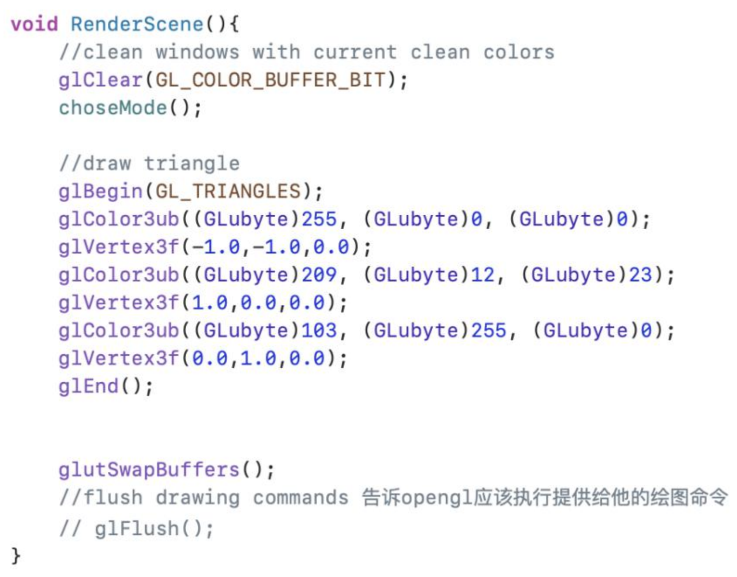
- 设置键盘相应函数改变两种着色模式，并且在渲染函数中使用



```
void processSpecialKey(int key,int x, int y){
    switch (key) {
        case GLUT_KEY_F1:
            mode = 0;
            break;
        case GLUT_KEY_F2:
            mode = 1;
            break;
        default:
            break;
    }
}
```



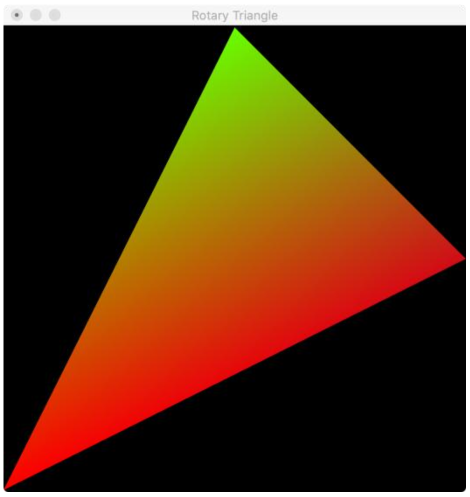
```
void choseMode(){
    if (mode == 0)
        glShadeModel(GL_SMOOTH);
    if (mode == 1) {
        glShadeModel(GL_FLAT);
    }
}
```



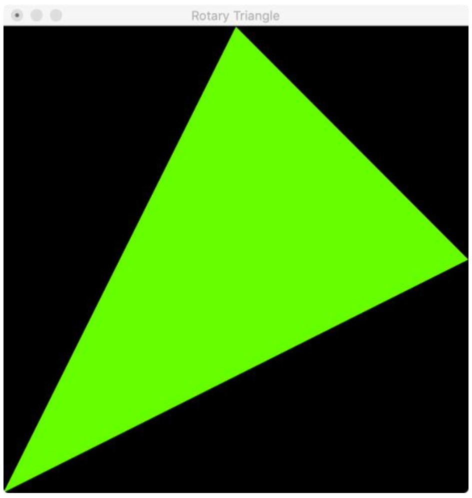
```
void RenderScene(){
    //clean windows with current clean colors
    glClear(GL_COLOR_BUFFER_BIT);
    choseMode();

    //draw triangle
    glBegin(GL_TRIANGLES);
    glColor3ub((GLubyte)255, (GLubyte)0, (GLubyte)0);
    glVertex3f(-1.0,-1.0,0.0);
    glColor3ub((GLubyte)209, (GLubyte)12, (GLubyte)23);
    glVertex3f(1.0,0.0,0.0);
    glColor3ub((GLubyte)103, (GLubyte)255, (GLubyte)0);
    glVertex3f(0.0,1.0,0.0);
    glEnd();


    glutSwapBuffers();
    //flush drawing commands 告诉opengl应该执行提供给他的绘图命令
    // glFlush();
}
```

效果:

GL_SMOOTH:



GL_FLAT:



区别: opengl 默认是将制定的两点颜色进行插值, 绘制之间的其他点

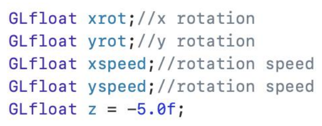
如果两点的颜色相同，使用两个参数效果相同。

如果两点颜色不同，GL_SMOOTH 会出现过渡效果，GL_FLAT 则只是以指定的某一点的单一色绘制其他所有点。

9)绘制一个旋转的立方体，并向场景中添加光源：环境光、散射光；

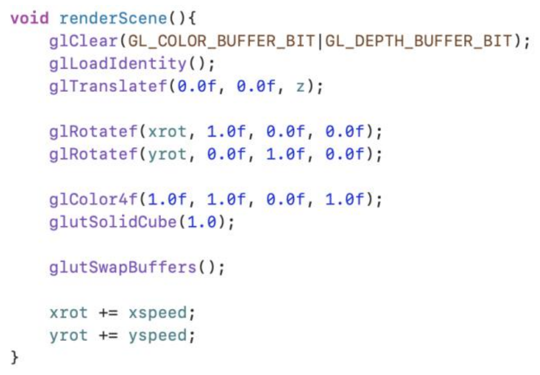
- changSize() 是透视投影（可见雪人世界）
- 旋转正方体（InitGL()中下边部分是实例化灯光）

设置数据



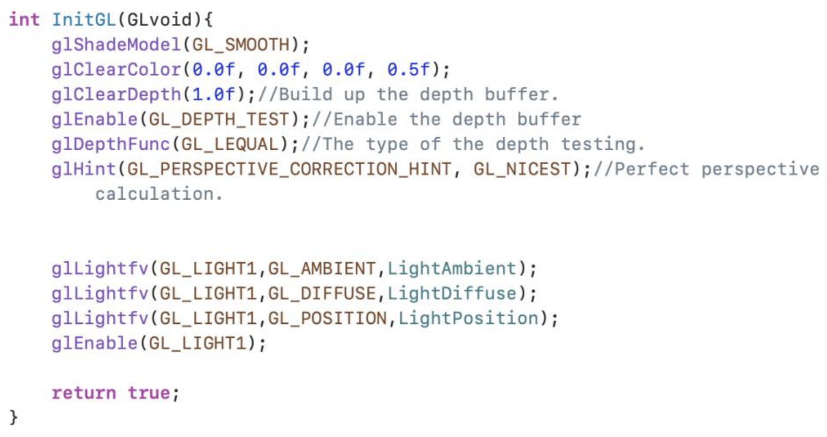
```
GLfloat xrot;//x rotation
GLfloat yrot;//y rotation
GLfloat xspeed;//rotation speed
GLfloat yspeed;//rotation speed
GLfloat z = -5.0f;
```

设置渲染函数



```
void renderScene(){
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, z);

    glRotatef(xrot, 1.0f, 0.0f, 0.0f);
    glRotatef(yrot, 0.0f, 1.0f, 0.0f);

    glColor4f(1.0f, 1.0f, 0.0f, 1.0f);
    glutSolidCube(1.0);

    glutSwapBuffers();

    xrot += xspeed;
    yrot += yspeed;
}
```

初始化函数



```
int InitGL(GLvoid){
    glShadeModel(GL_SMOOTH);
    glClearColor(0.0f, 0.0f, 0.0f, 0.5f);
    glClearDepth(1.0f);//Build up the depth buffer.
    glEnable(GL_DEPTH_TEST);//Enable the depth buffer
    glDepthFunc(GL_LEQUAL);//The type of the depth testing.
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);//Perfect perspective
        calculation.


    glLightfv(GL_LIGHT1,GL_AMBIENT,LightAmbient);
    glLightfv(GL_LIGHT1,GL_DIFFUSE,LightDiffuse);
    glLightfv(GL_LIGHT1,GL_POSITION,LightPosition);
    glEnable(GL_LIGHT1);

    return true;
}
```

改变旋转速度

```
void processSpecialKey(int key, int x, int y){
    if (key == GLUT_KEY_PAGE_UP) {
        z -= 0.02f;
    }
    if (key == GLUT_KEY_PAGE_DOWN) {
        z += 0.02f;
    }
    if (key == GLUT_KEY_PAGE_UP) {
        xspeed -= 0.01f;
    }
    if (key == GLUT_KEY_PAGE_DOWN) {
        xspeed += 0.01f;
    }
    if (key == GLUT_KEY_LEFT) {
        yspeed -= 0.01f;
    }
    if (key == GLUT_KEY_RIGHT) {
        yspeed -= 0.01f;
    }
}
```

- 灯光设置

参数设置

```
GLfloat LightAmbient[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat LightDiffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};
GLfloat LightPosition[] = {0.0f, 0.0f, 2.0f, 1.0f};
GLfloat light = 1;
```

实例化在 InitGL()中

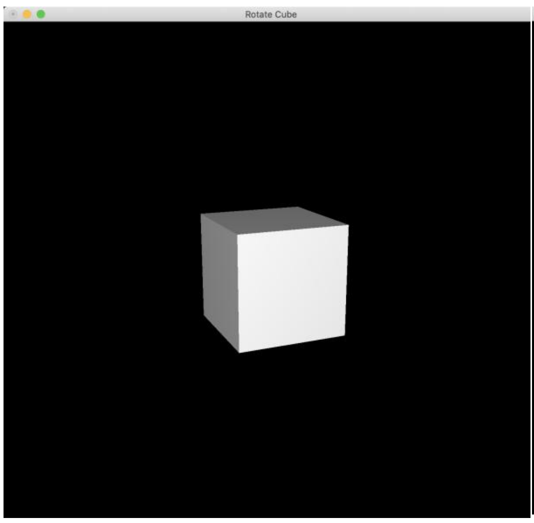
开灯关灯：

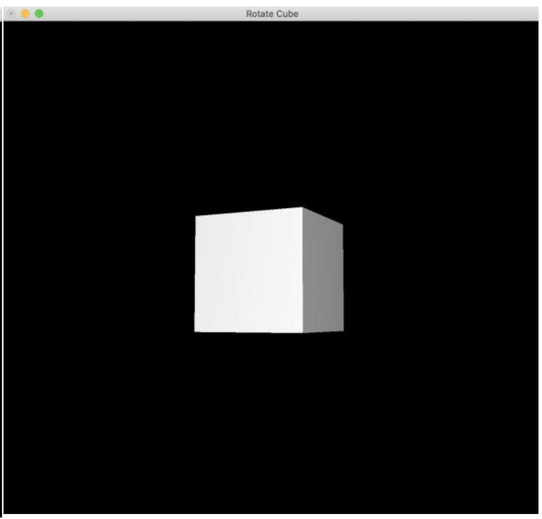
```
void processNomalKey(unsigned char key, int x, int y){
    switch (key) {
        case 27:
            exit(0);
            break;
        case 'l':
            light =!light;
            light?glEnable(GL_LIGHTING):glDisable(GL_LIGHTING);


        default:
            break;
    }

}
```

- main 函数

```
int main(int argc, char **argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DEPTH|GLUT_DOUBLE|GLUT_RGBA);//使用雙緩衝區有利於動畫的流暢性
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(640, 640);
    glutCreateWindow("Rotate Cube");

    // register the callbacks
    glutDisplayFunc(renderScene);
    glutIdleFunc(renderScene);
    glutReshapeFunc(ChangeSize);
    glutSpecialFunc(processSpecialKey);
    glutKeyboardFunc(processNomalKey);

    InitGL();

    //enter envent processing loop
    glutMainLoop();

    return 0;
}
```

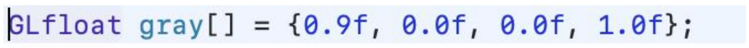
效果:



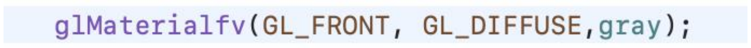


10)采用两种不同方法为在上例中给立方体设置材质属性;

（1） glMaterialfv 函数设置材质属性

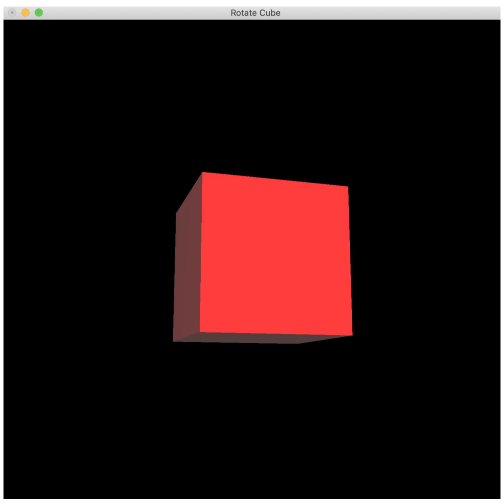


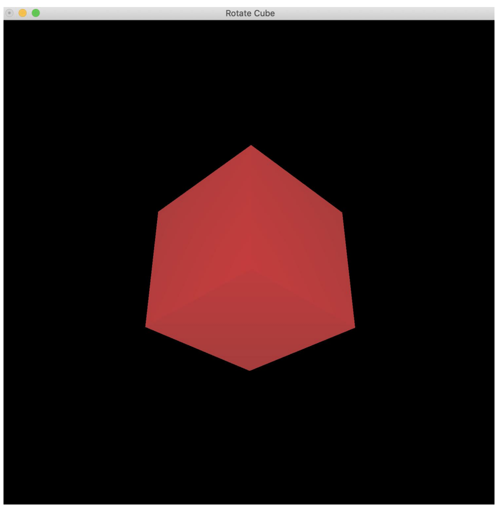
```
GLfloat gray[] = {0.9f, 0.0f, 0.0f, 1.0f};
```



```
    glMaterialfv(GL_FRONT, GL_DIFFUSE,gray);
```

效果:





（2）　glColorMaterial(GL_FRONT, GL_DIFFUSE);

在 InitGL()中 enable

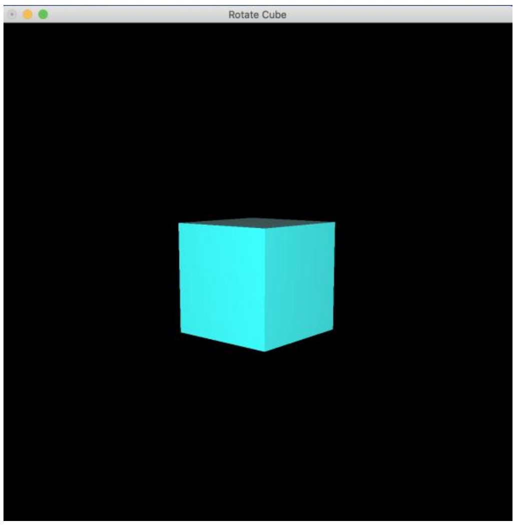
```
//2.material
glColorMaterial(GL_FRONT, GL_DIFFUSE);
glEnable(GL_COLOR_MATERIAL);
```

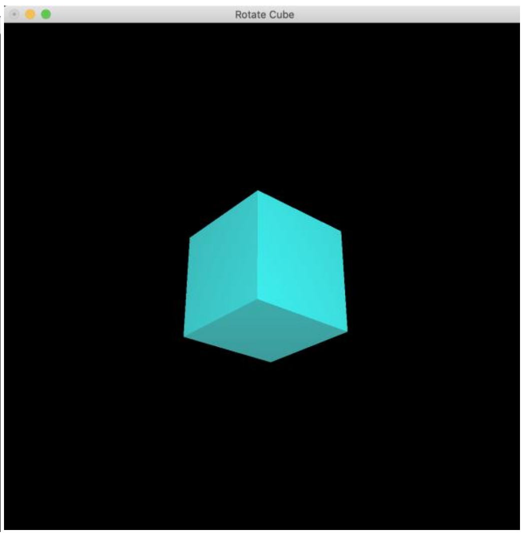
在绘制之前声明颜色

```
//2. material
glColor4f(0.0f, 1.0f, 1.0f, 1.0f);


glutSolidCube(1.0);
```

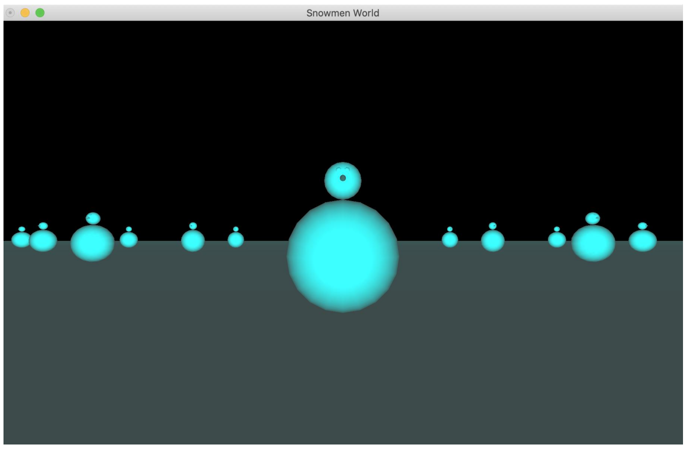
将材质与灯光融合

效果：



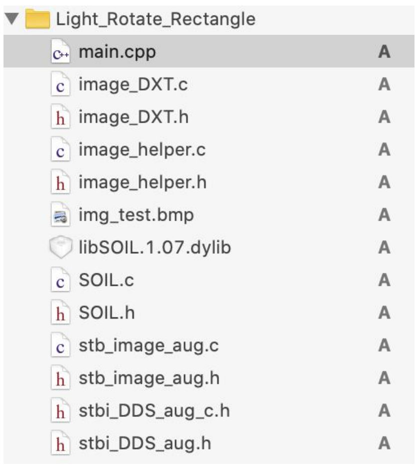


11)为雪人世界添加光照、设置材质属性；(用相同的方式加上灯光和材质)(怪吓人的…… 正在看谁是受害者，开始设置红色的灯光，差点吓死 TAT)



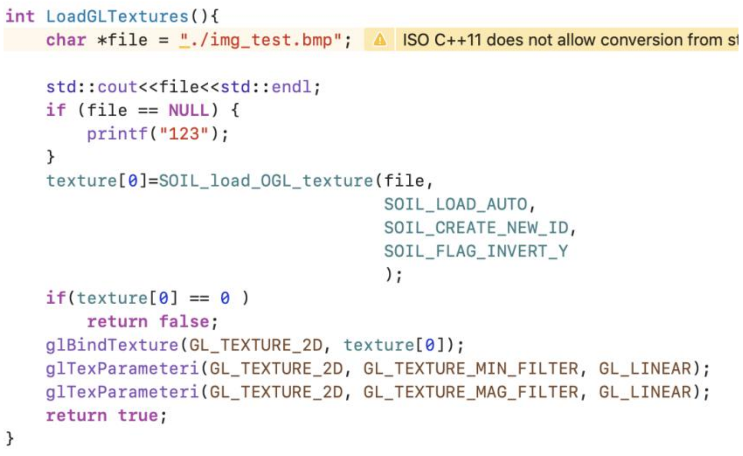
12)为 9）中的立方体添加纹理；

先链接 SOIL.h（不仅要导入 SOIL.h 的链接文件，还要导入它的动态库和.bmp 素材）



```
Light_Rotate_Rectangle
    main.cpp              A
    image_DXT.c           A
    image_DXT.h           A
    image_helper.c        A
    image_helper.h        A
    img_test.bmp          A
    libSOIL.1.07.dylib    A
    SOIL.c                A
    SOIL.h                A
    stb_image_aug.c       A
    stb_image_aug.h       A
    stbi_DDS_aug_c.h      A
    stbi_DDS_aug.h        A
```

创建读取纹理函数



```
int LoadGLTextures(){
    char *file = "./img_test.bmp";  ⚠ ISO C++11 does not allow conversion from st

    std::cout<<file<<std::endl;
    if (file == NULL) {
        printf("123");
    }
    texture[0]=SOIL_load_OGL_texture(file,
                                     SOIL_LOAD_AUTO,
                                     SOIL_CREATE_NEW_ID,
                                     SOIL_FLAG_INVERT_Y
                                     );
    if(texture[0] == 0 )
        return false;
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    return true;
}
```

需要注意资源的引入路径，可以直接用绝对路径，也可以设置 user working path 设置：Product -> Scheme ->Edit Scheme ->Run -> Options ->User working path 为根目录或者什么

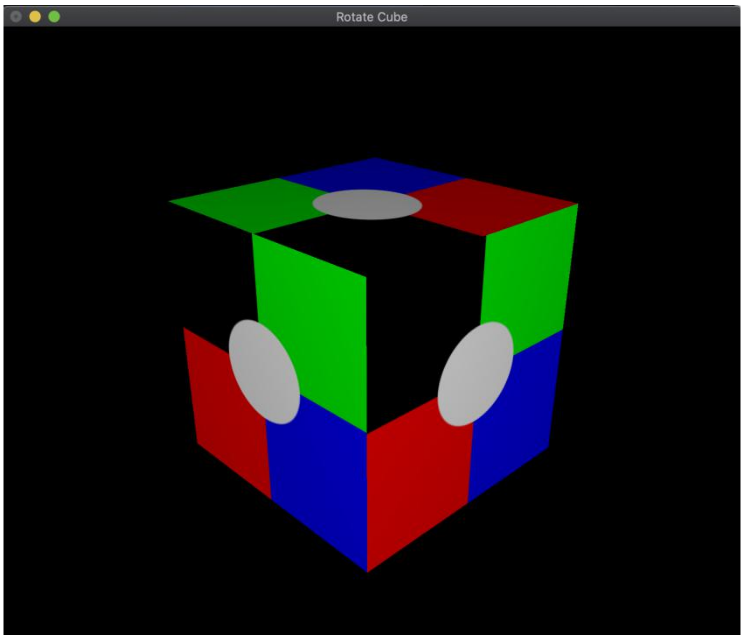
Enable 贴图。在绘制之前声明即可，可在 InitGL()中也可在 renderScene 中。

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture[0]);
```

渲染函数

```
void renderScene(){
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, z);

    glRotatef(xrot, 1.0f, 0.0f, 0.0f);
    glRotatef(yrot, 0.0f, 1.0f, 0.0f);
    //2. material
    //glColor4f(0.0f, 1.0f, 1.0f, 1.0f);
    //glutSolidCube(1.0); (no texture)
// texture
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    glBegin(GL_QUADS);
    glNormal3f(0.0f, 0.0f, 1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, -1.0f,1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f,1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f,1.0f);
    glNormal3f(0.0f, 0.0f, -1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,-1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f,-1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(1.0f, 1.0f,-1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(1.0f, -1.0f,-1.0f);
    glNormal3f(0.0f, 1.0f, 0.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f,-1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, 1.0f,1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, 1.0f,1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f,-1.0f);
    glNormal3f(0.0f, -1.0f, 0.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f,-1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(1.0f, -1.0f,-1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(1.0f, -1.0f,1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,1.0f);
    glNormal3f(1.0f, 0.0f, 0.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, -1.0f,-1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f,-1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(1.0f, 1.0f,1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(1.0f, -1.0f,1.0f);
    glNormal3f(-1.0f, 0.0f, 0.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,-1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, 1.0f,1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f,-1.0f);
    glEnd();

    glutSwapBuffers();

    xrot += xspeed;
    yrot += yspeed;
}
```

效果：


Rotate Cube