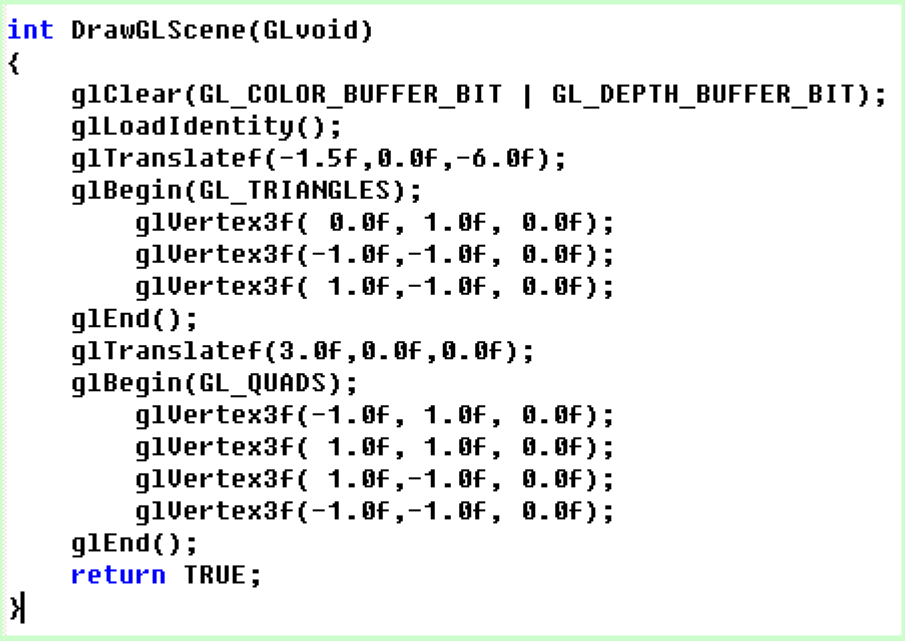
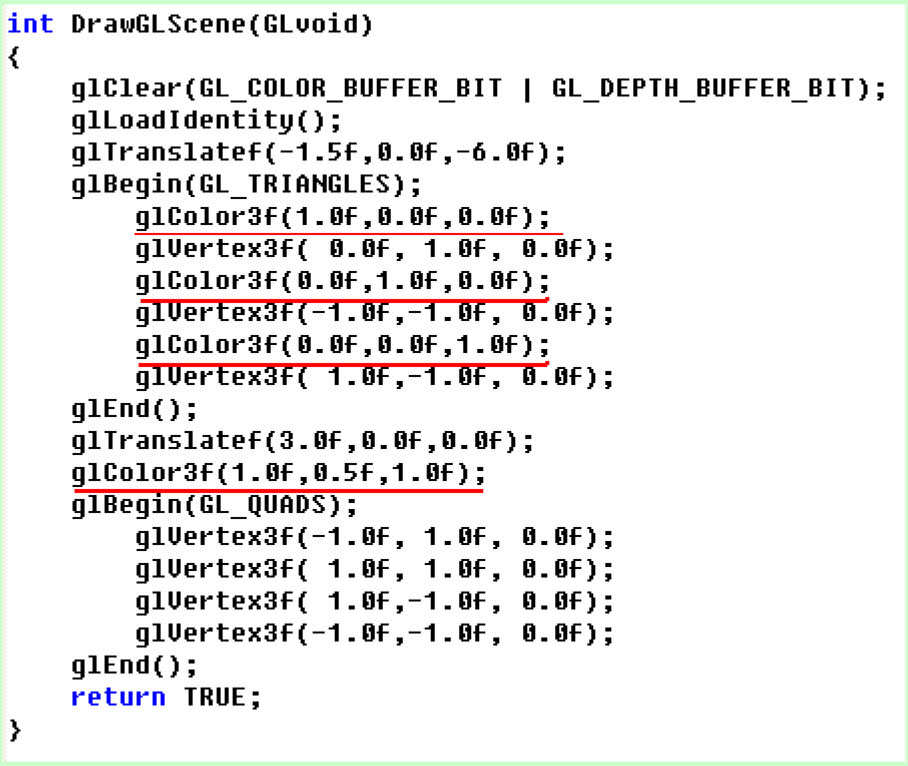
游戏软件开发基础课程讲义

基本 2D 图元绘制



```
int DrawGLScene(GLvoid)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(-1.5f,0.0f,-6.0f);
    glBegin(GL_TRIANGLES);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
    glEnd();
    glTranslatef(3.0f,0.0f,0.0f);
    glBegin(GL_QUADS);
        glVertex3f(-1.0f, 1.0f, 0.0f);
        glVertex3f( 1.0f, 1.0f, 0.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
    glEnd();
    return TRUE;
}
```

为 2D 图元添加颜色



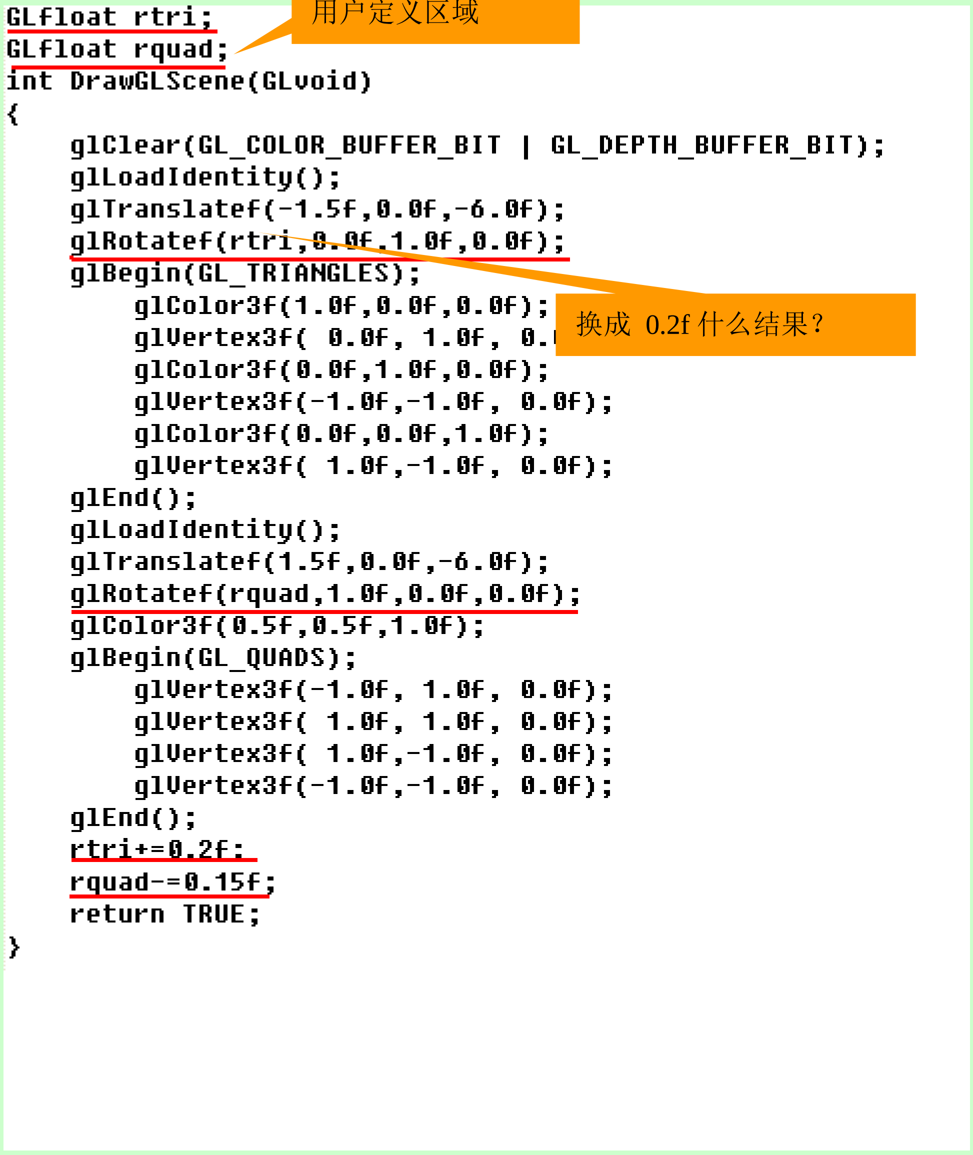
```
int DrawGLScene(GLvoid)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(-1.5f,0.0f,-6.0f);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
    glEnd();
    glTranslatef(3.0f,0.0f,0.0f);
    glColor3f(1.0f,0.5f,1.0f);
    glBegin(GL_QUADS);
        glVertex3f(-1.0f, 1.0f, 0.0f);
        glVertex3f( 1.0f, 1.0f, 0.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
    glEnd();
    return TRUE;
}
```

一些常用的混合色

混合色	红色成分（R）	绿色成分（G）	蓝色成分（B）
黑	0.0	0.0	0.0
红	1.0	0.0	0.0
绿	0.0	1.0	0.0
黄	1.0	1.0	0.0
蓝	0.0	0.0	1.0
紫	1.0	0.0	1.0
青	0.0	1.0	1.0
深灰	0.25	0.25	0.25
浅灰	0.75	0.75	0.75
棕	0.60	0.40	0.12
南瓜橙	0.98	0.625	0.12
粉红	0.98	0.04	0.70
紫红	0.60	0.40	0.70
白	1.0	1.0	1.0

glClearColor 的最后一个参数是 alpha 成分，主要用于混合的特殊效果，如半透明效果等。

为 2D 图元添加旋转



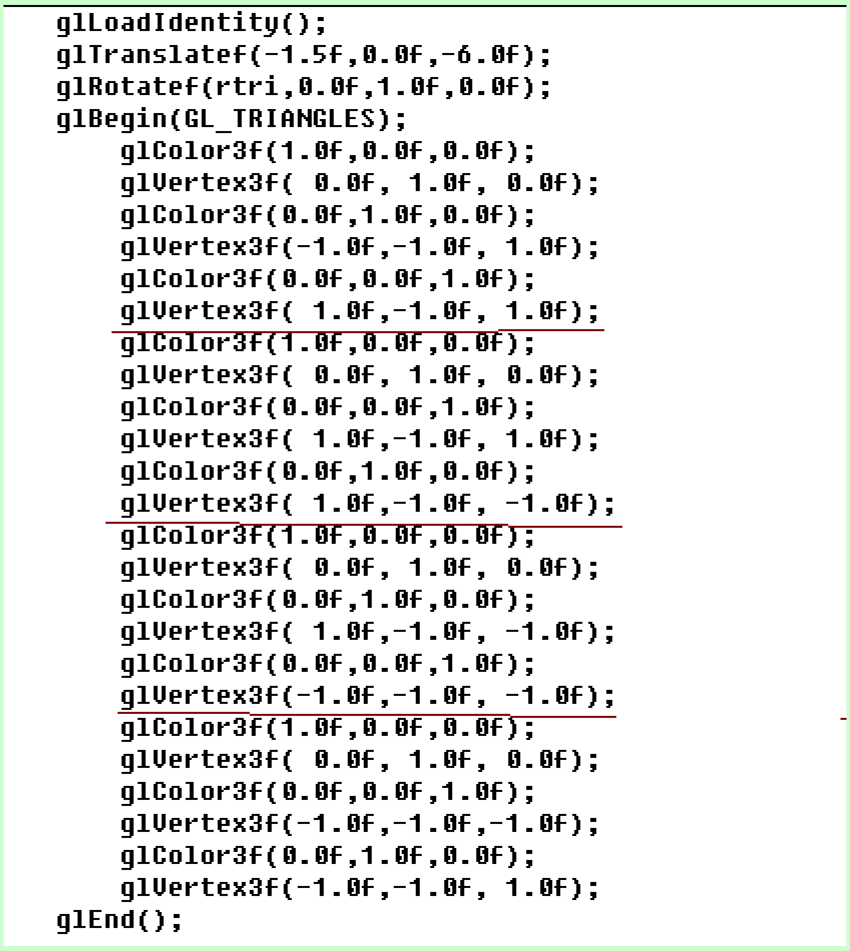
用户定义区域

```
GLfloat rtri;
GLfloat rquad;
int DrawGLScene(GLvoid)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(-1.5f,0.0f,-6.0f);
    glRotatef(rtri,0.0f,1.0f,0.0f);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
    glEnd();
    glLoadIdentity();
    glTranslatef(1.5f,0.0f,-6.0f);
    glRotatef(rquad,1.0f,0.0f,0.0f);
    glColor3f(0.5f,0.5f,1.0f);
    glBegin(GL_QUADS);
        glVertex3f(-1.0f, 1.0f, 0.0f);
        glVertex3f( 1.0f, 1.0f, 0.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
    glEnd();
    rtri+=0.2f;
    rquad-=0.15f;
    return TRUE;
}
```

换成 0.2f 什么结果？

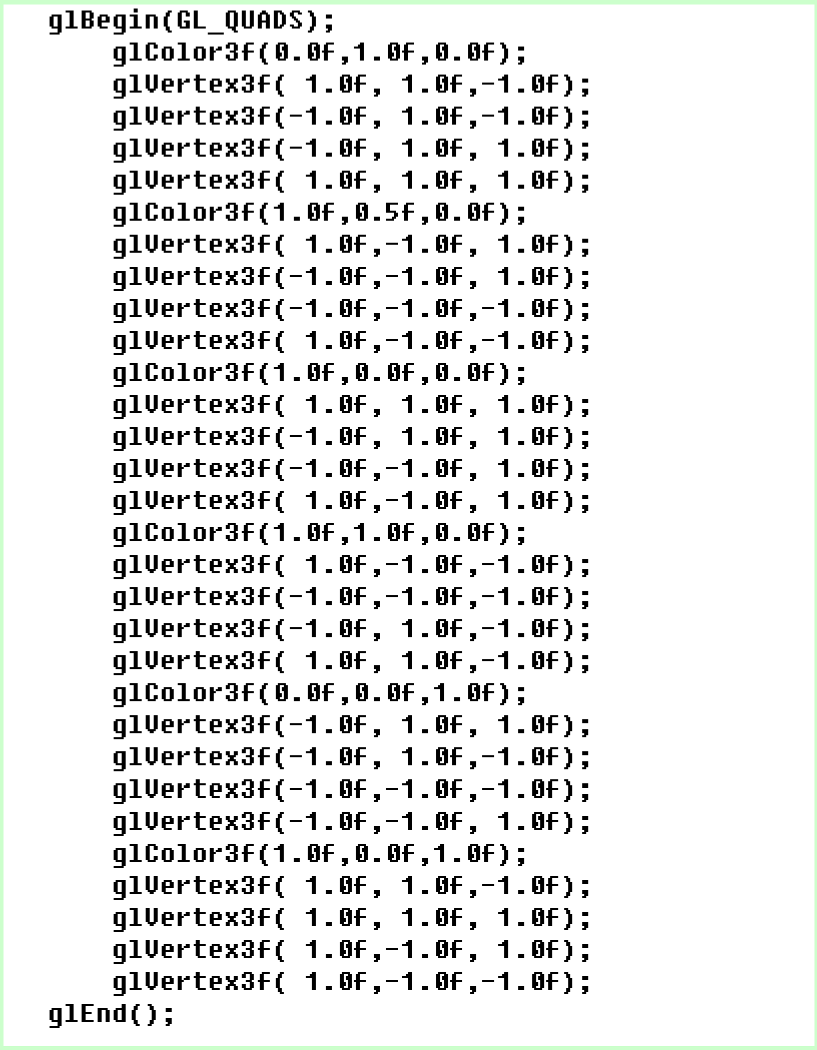
基本 3D 图元绘制

金字塔



```
    glLoadIdentity();
    glTranslatef(-1.5f,0.0f,-6.0f);
    glRotatef(rtri,0.0f,1.0f,0.0f);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f(-1.0f,-1.0f, 1.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f( 1.0f,-1.0f, 1.0f);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f( 1.0f,-1.0f, 1.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f( 1.0f,-1.0f, -1.0f);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f( 1.0f,-1.0f, -1.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f(-1.0f,-1.0f, -1.0f);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f(-1.0f,-1.0f,-1.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f(-1.0f,-1.0f, 1.0f);
    glEnd();
```

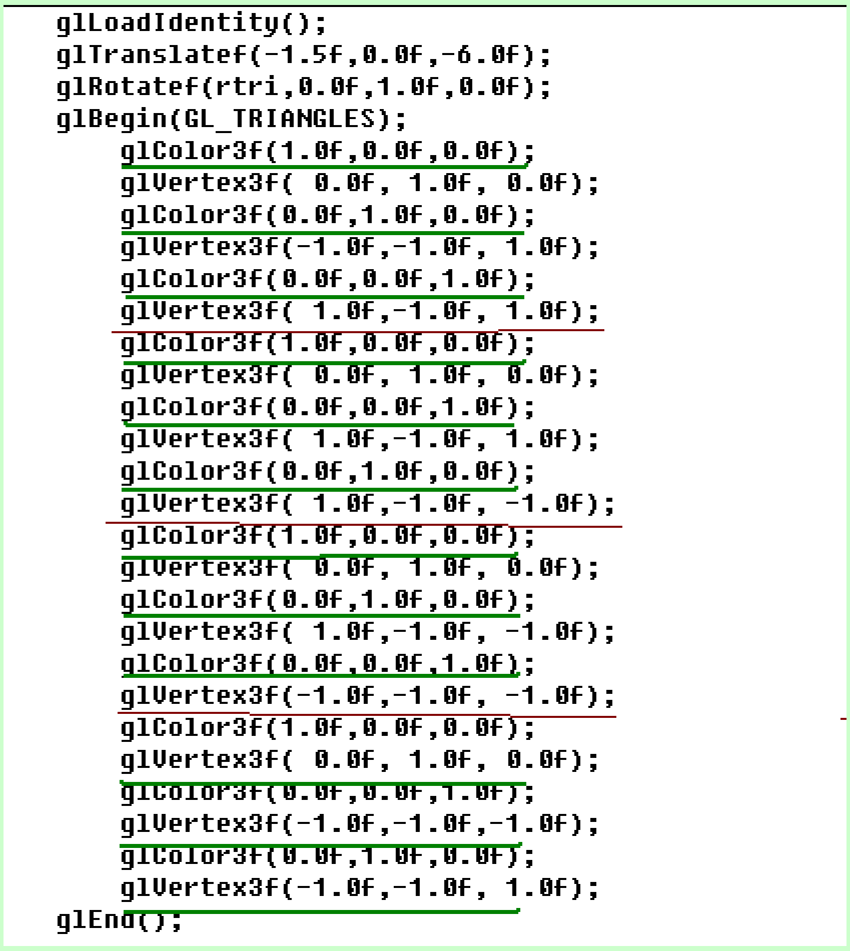
六面体



```
glBegin(GL_QUADS);
    glColor3f(0.0f,1.0f,0.0f);
    glVertex3f( 1.0f, 1.0f,-1.0f);
    glVertex3f(-1.0f, 1.0f,-1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glColor3f(1.0f,0.5f,0.0f);
    glVertex3f( 1.0f,-1.0f, 1.0f);
    glVertex3f(-1.0f,-1.0f, 1.0f);
    glVertex3f(-1.0f,-1.0f,-1.0f);
    glVertex3f( 1.0f,-1.0f,-1.0f);
    glColor3f(1.0f,0.0f,0.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f,-1.0f, 1.0f);
    glVertex3f( 1.0f,-1.0f, 1.0f);
    glColor3f(1.0f,1.0f,0.0f);
    glVertex3f( 1.0f,-1.0f,-1.0f);
    glVertex3f(-1.0f,-1.0f,-1.0f);
    glVertex3f(-1.0f, 1.0f,-1.0f);
    glVertex3f( 1.0f, 1.0f,-1.0f);
    glColor3f(0.0f,0.0f,1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f,-1.0f);
    glVertex3f(-1.0f,-1.0f,-1.0f);
    glVertex3f(-1.0f,-1.0f, 1.0f);
    glColor3f(1.0f,0.0f,1.0f);
    glVertex3f( 1.0f, 1.0f,-1.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glVertex3f( 1.0f,-1.0f, 1.0f);
    glVertex3f( 1.0f,-1.0f,-1.0f);
glEnd();
```

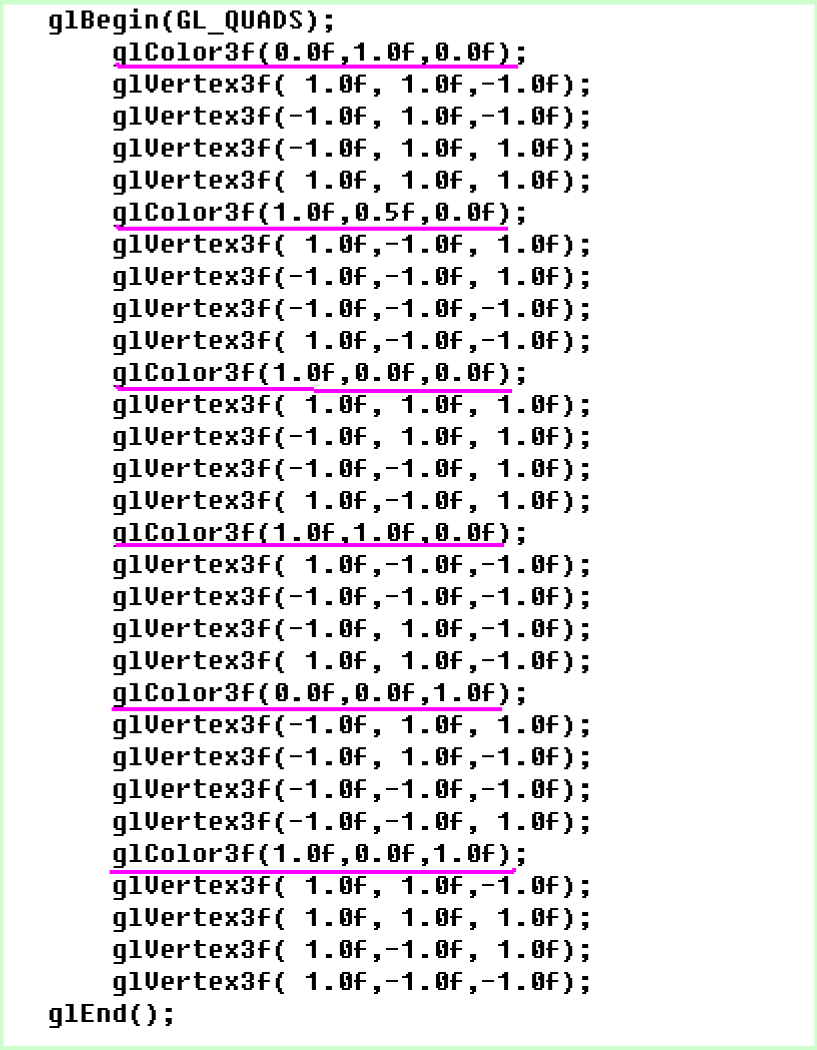
为 3D 图元添加颜色

金字塔



```
    glLoadIdentity();
    glTranslatef(-1.5f,0.0f,-6.0f);
    glRotatef(rtri,0.0f,1.0f,0.0f);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f(-1.0f,-1.0f, 1.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f( 1.0f,-1.0f, 1.0f);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f( 1.0f,-1.0f, 1.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f( 1.0f,-1.0f, -1.0f);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f( 1.0f,-1.0f, -1.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f(-1.0f,-1.0f, -1.0f);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f(-1.0f,-1.0f,-1.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f(-1.0f,-1.0f, 1.0f);
    glEnd();
```

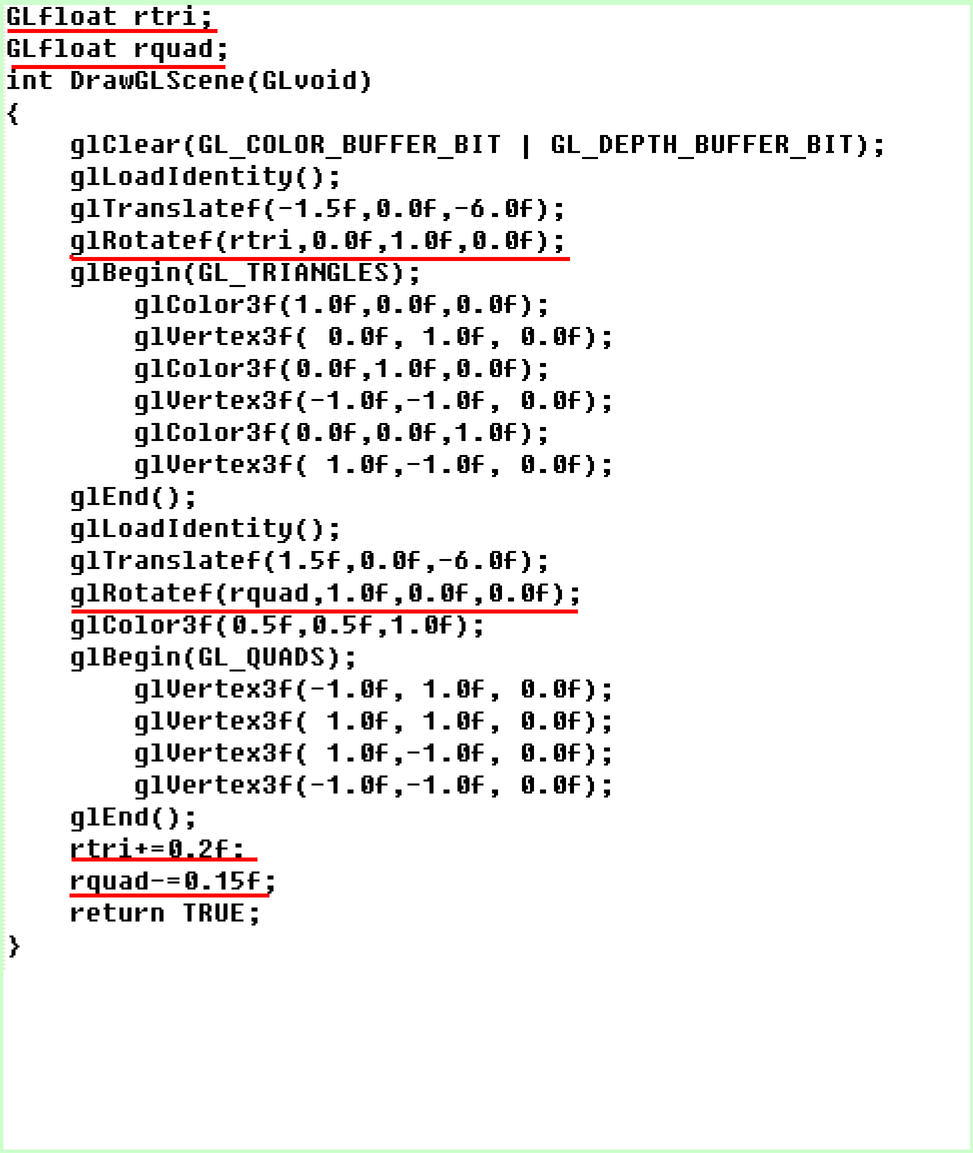
六面体



```
glBegin(GL_QUADS);
    glColor3f(0.0f,1.0f,0.0f);
    glVertex3f( 1.0f, 1.0f,-1.0f);
    glVertex3f(-1.0f, 1.0f,-1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glColor3f(1.0f,0.5f,0.0f);
    glVertex3f( 1.0f,-1.0f, 1.0f);
    glVertex3f(-1.0f,-1.0f, 1.0f);
    glVertex3f(-1.0f,-1.0f,-1.0f);
    glVertex3f( 1.0f,-1.0f,-1.0f);
    glColor3f(1.0f,0.0f,0.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f,-1.0f, 1.0f);
    glVertex3f( 1.0f,-1.0f, 1.0f);
    glColor3f(1.0f,1.0f,0.0f);
    glVertex3f( 1.0f,-1.0f,-1.0f);
    glVertex3f(-1.0f,-1.0f,-1.0f);
    glVertex3f(-1.0f, 1.0f,-1.0f);
    glVertex3f( 1.0f, 1.0f,-1.0f);
    glColor3f(0.0f,0.0f,1.0f);
    glVertex3f(-1.0f, 1.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f,-1.0f);
    glVertex3f(-1.0f,-1.0f,-1.0f);
    glVertex3f(-1.0f,-1.0f, 1.0f);
    glColor3f(1.0f,0.0f,1.0f);
    glVertex3f( 1.0f, 1.0f,-1.0f);
    glVertex3f( 1.0f, 1.0f, 1.0f);
    glVertex3f( 1.0f,-1.0f, 1.0f);
    glVertex3f( 1.0f,-1.0f,-1.0f);
glEnd();
```

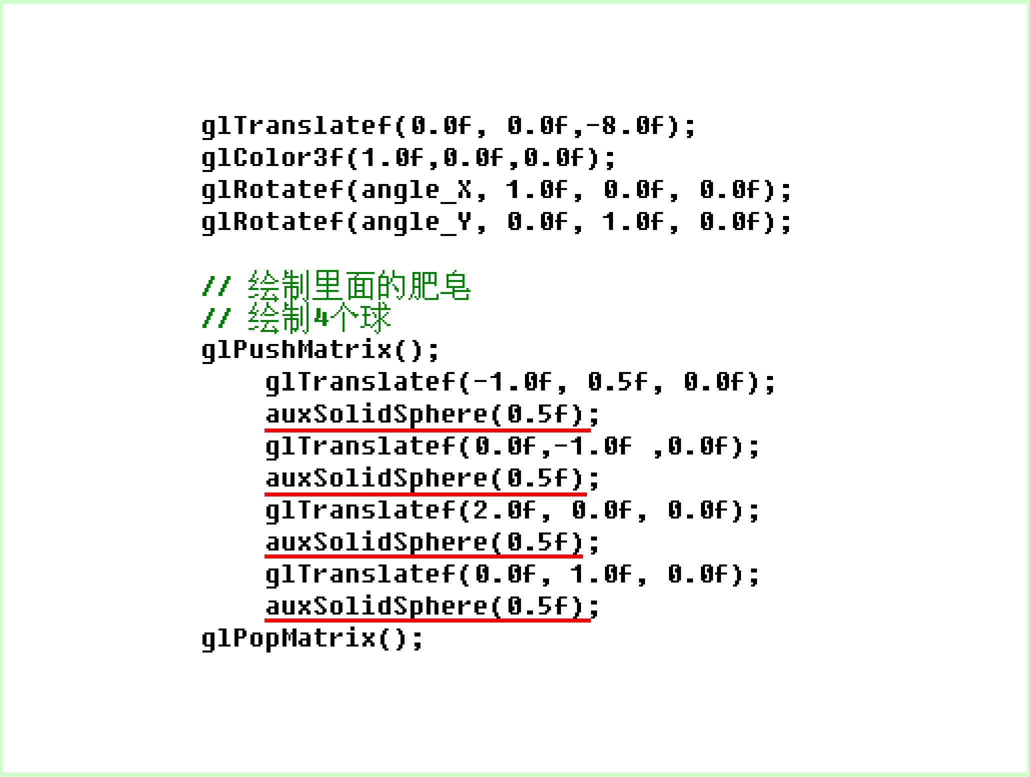
为 3D 图元添加旋转

同 2D 图元



```
GLfloat rtri;
GLfloat rquad;
int DrawGLScene(GLvoid)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(-1.5f,0.0f,-6.0f);
    glRotatef(rtri,0.0f,1.0f,0.0f);
    glBegin(GL_TRIANGLES);
        glColor3f(1.0f,0.0f,0.0f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.0f,1.0f,0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
        glColor3f(0.0f,0.0f,1.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
    glEnd();
    glLoadIdentity();
    glTranslatef(1.5f,0.0f,-6.0f);
    glRotatef(rquad,1.0f,0.0f,0.0f);
    glColor3f(0.5f,0.5f,1.0f);
    glBegin(GL_QUADS);
        glVertex3f(-1.0f, 1.0f, 0.0f);
        glVertex3f( 1.0f, 1.0f, 0.0f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
    glEnd();
    rtri+=0.2f;
    rquad-=0.15f;
    return TRUE;
}
```

使用 aux 库绘制 3D 图元

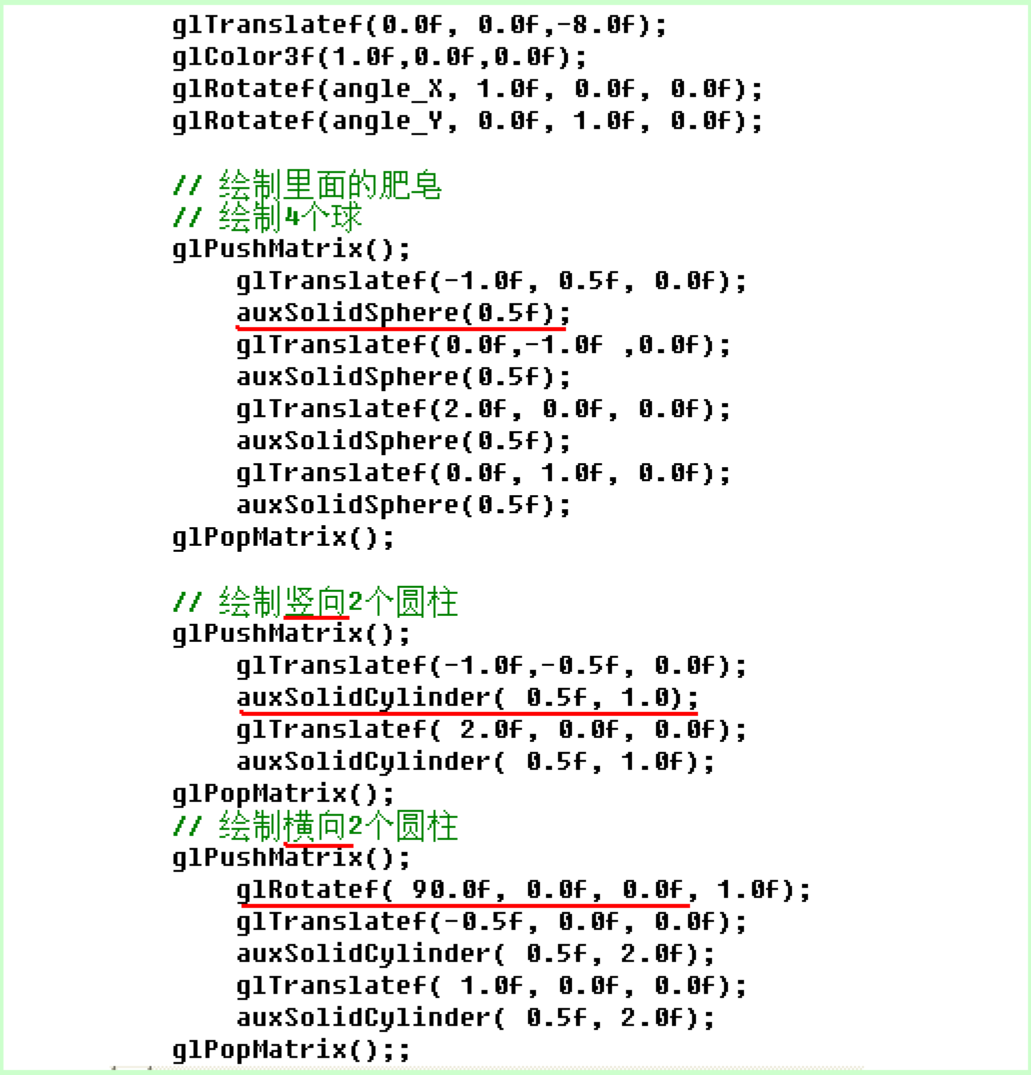


```
glTranslatef(0.0f, 0.0f,-8.0f);
glColor3f(1.0f,0.0f,0.0f);
glRotatef(angle_X, 1.0f, 0.0f, 0.0f);
glRotatef(angle_Y, 0.0f, 1.0f, 0.0f);

// 绘制里面的肥皂
// 绘制4个球
glPushMatrix();
    glTranslatef(-1.0f, 0.5f, 0.0f);
    auxSolidSphere(0.5f);
    glTranslatef(0.0f,-1.0f ,0.0f);
    auxSolidSphere(0.5f);
    glTranslatef(2.0f, 0.0f, 0.0f);
    auxSolidSphere(0.5f);
    glTranslatef(0.0f, 1.0f, 0.0f);
    auxSolidSphere(0.5f);
glPopMatrix();
```

功能	函数
绘制球	void auxWireSphere(GLdouble radius) void auxSolidSphere(GLdouble radius)
绘制立方体	void auxWireCube(GLdouble size) void auxSolidCube(GLdouble size)
绘制长方体	void auxWireBox(GLdouble width,GLdouble height,GLdouble depth) void auxSolidBox(GLdouble width,GLdouble height,GLdouble depth)
绘制环形圆纹面	void auxWireTorus(GLdouble innerRadius,GLdouble outerRadius) void auxSolidTorus(GLdouble innerRadius,GLdouble outerRadius)
绘制圆柱	void auxWireCylinder(GLdouble radius,GLdouble height) void auxSolidCylinder(GLdouble radius,GLdouble height)
绘制二十面体	void auxWireIcosahedron(GLdouble radius) void auxSolidIcosahedron(GLdouble radius)

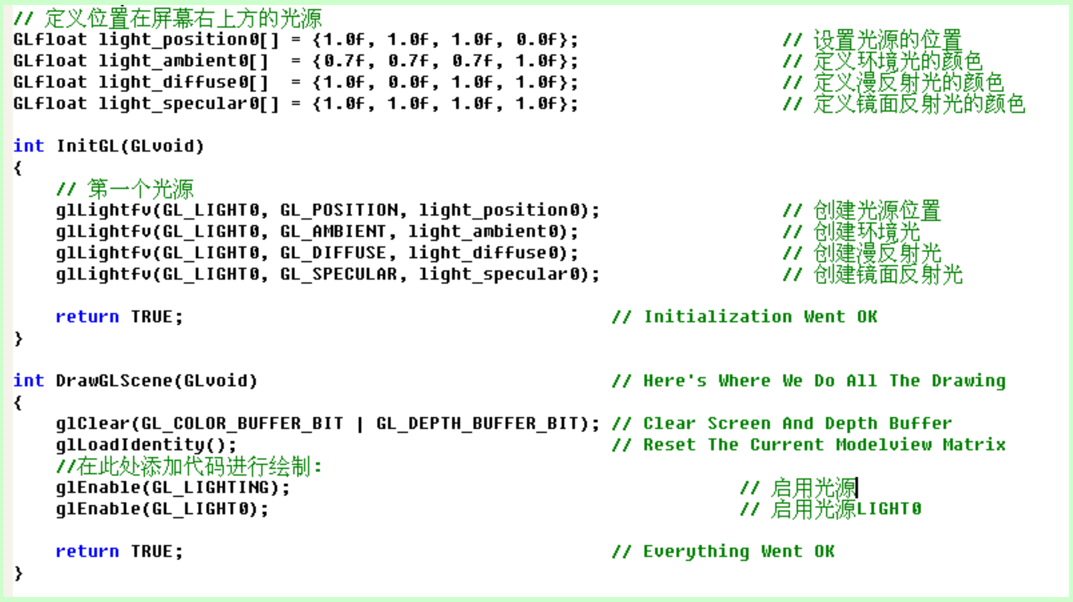
绘制八面体	void auxWireOctahedron(GLdouble radius) void auxSolidOctahedron(GLdouble radius)
绘制四面体	void auxWireTetrahedron(GLdouble radius) void auxSolidTetrahedron(GLdouble radius)
绘制十二面体	void auxWireDodecahedron(GLdouble radius) void auxSolidDodecahedron(GLdouble radius)
绘制圆锥	void auxWireCone(GLdouble radius,GLdouble height) void auxSolidCone(GLdouble radius,GLdouble height)
绘制茶壶	void auxWireTeapot(GLdouble size) void aucSolidTeapot(GLdouble size)

使用 aux 库组合 3D 模型



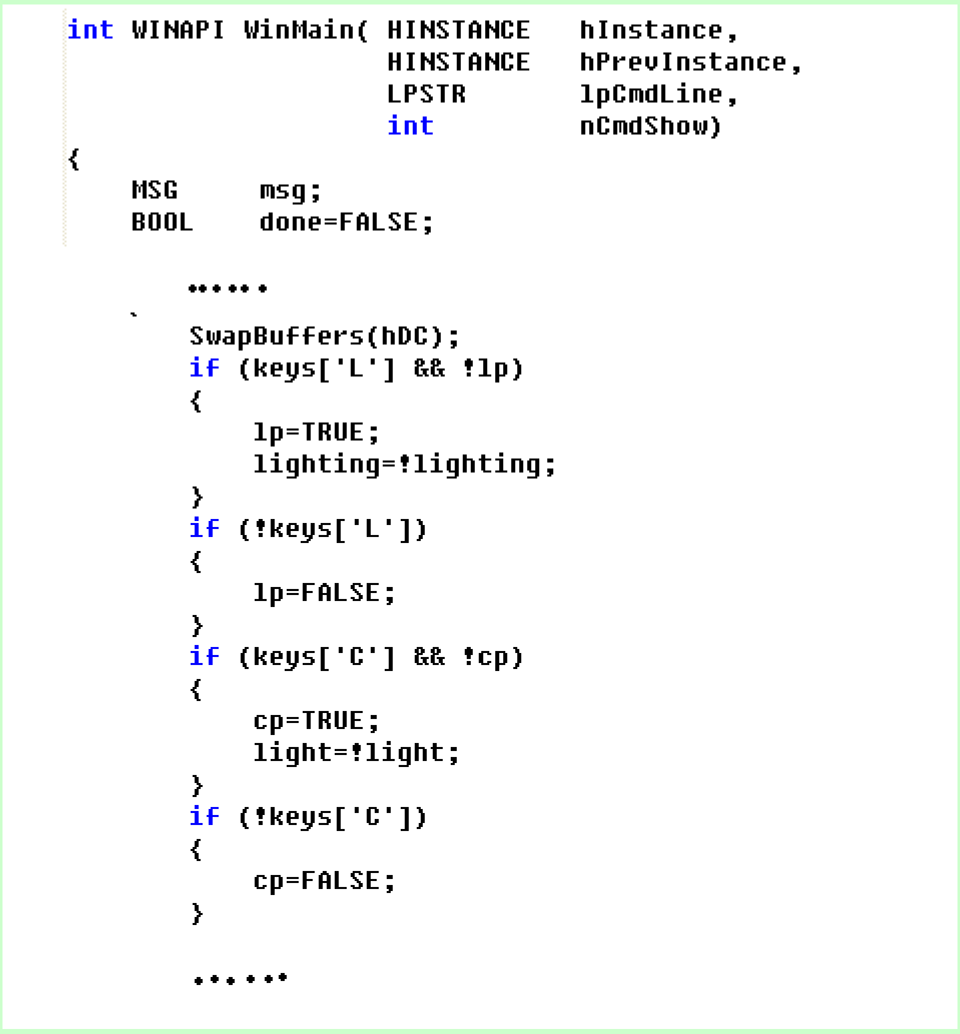
```
glTranslatef(0.0f, 0.0f,-8.0f);
glColor3f(1.0f,0.0f,0.0f);
glRotatef(angle_X, 1.0f, 0.0f, 0.0f);
glRotatef(angle_Y, 0.0f, 1.0f, 0.0f);

// 绘制里面的肥皂
// 绘制4个球
glPushMatrix();
    glTranslatef(-1.0f, 0.5f, 0.0f);
    auxSolidSphere(0.5f);
    glTranslatef(0.0f,-1.0f ,0.0f);
    auxSolidSphere(0.5f);
    glTranslatef(2.0f, 0.0f, 0.0f);
    auxSolidSphere(0.5f);
    glTranslatef(0.0f, 1.0f, 0.0f);
    auxSolidSphere(0.5f);
glPopMatrix();

// 绘制竖向2个圆柱
glPushMatrix();
    glTranslatef(-1.0f,-0.5f, 0.0f);
    auxSolidCylinder( 0.5f, 1.0);
    glTranslatef( 2.0f, 0.0f, 0.0f);
    auxSolidCylinder( 0.5f, 1.0f);
glPopMatrix();
// 绘制横向2个圆柱
glPushMatrix();
    glRotatef( 90.0f, 0.0f, 0.0f, 1.0f);
    glTranslatef(-0.5f, 0.0f, 0.0f);
    auxSolidCylinder( 0.5f, 2.0f);
    glTranslatef( 1.0f, 0.0f, 0.0f);
    auxSolidCylinder( 0.5f, 2.0f);
glPopMatrix();;
```

为场景添加光源



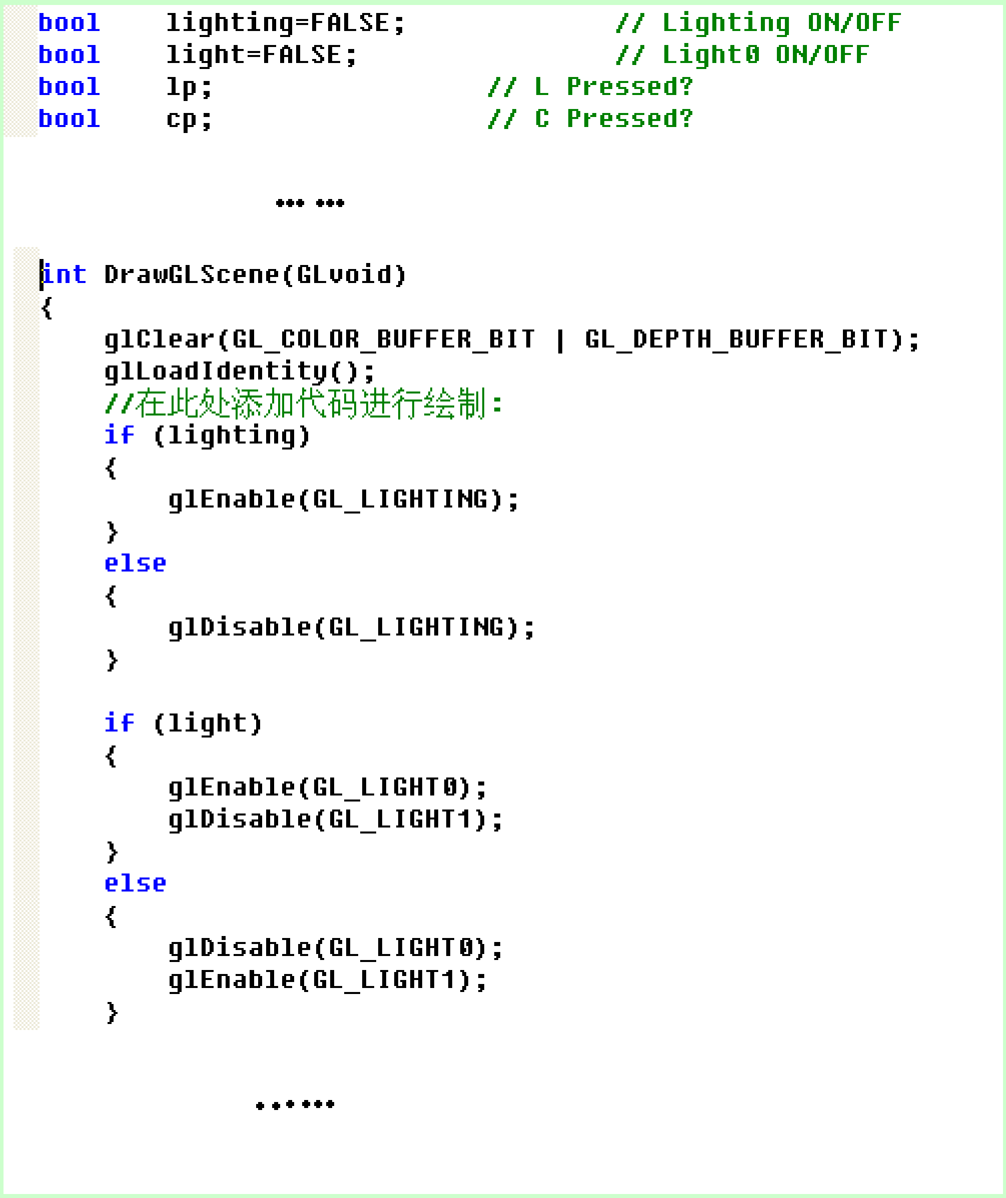
```
// 定义位置在屏幕右上方的光源
GLfloat light_position0[] = {1.0f, 1.0f, 1.0f, 0.0f};                    // 设置光源的位置
GLfloat light_ambient0[]  = {0.7f, 0.7f, 0.7f, 1.0f};                    // 定义环境光的颜色
GLfloat light_diffuse0[]  = {1.0f, 0.0f, 1.0f, 1.0f};                    // 定义漫反射光的颜色
GLfloat light_specular0[] = {1.0f, 1.0f, 1.0f, 1.0f};                    // 定义镜面反射光的颜色

int InitGL(GLvoid)
{
    // 第一个光源
    glLightfv(GL_LIGHT0, GL_POSITION, light_position0);                  // 创建光源位置
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient0);                    // 创建环境光
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse0);                    // 创建漫反射光
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular0);                  // 创建镜面反射光

    return TRUE;                                        // Initialization Went OK
}

int DrawGLScene(GLvoid)                                 // Here's Where We Do All The Drawing
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear Screen And Depth Buffer
    glLoadIdentity();                                   // Reset The Current Modelview Matrix
    //在此处添加代码进行绘制:
    glEnable(GL_LIGHTING);                                           // 启用光源
    glEnable(GL_LIGHT0);                                             // 启用光源LIGHT0

    return TRUE;                                        // Everything Went OK
}
```

添加光源时的键盘响应



```
int WINAPI WinMain( HINSTANCE   hInstance,
                    HINSTANCE   hPrevInstance,
                    LPSTR       lpCmdLine,
                    int         nCmdShow)
{
    MSG      msg;
    BOOL     done=FALSE;

        ……

        SwapBuffers(hDC);
        if (keys['L'] && !lp)
        {
            lp=TRUE;
            lighting=!lighting;
        }
        if (!keys['L'])
        {
            lp=FALSE;
        }
        if (keys['C'] && !cp)
        {
            cp=TRUE;
            light=!light;
        }
        if (!keys['C'])
        {
            cp=FALSE;
        }

        ……
```

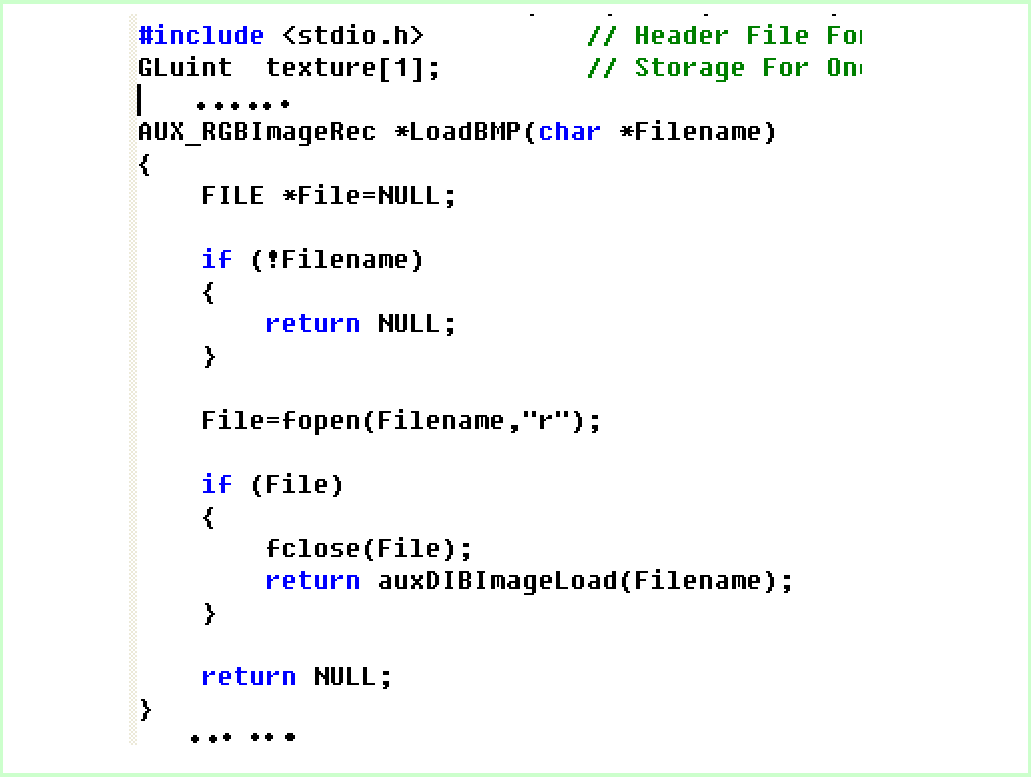
用键盘切换光源



```
bool    lighting=FALSE;              // Lighting ON/OFF
bool    light=FALSE;                 // Light0 ON/OFF
bool    lp;                  // L Pressed?
bool    cp;                  // C Pressed?


                 ••• •••


int DrawGLScene(GLvoid)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    //在此处添加代码进行绘制:
    if (lighting)
    {
        glEnable(GL_LIGHTING);
    }
    else
    {
        glDisable(GL_LIGHTING);
    }

    if (light)
    {
        glEnable(GL_LIGHT0);
        glDisable(GL_LIGHT1);
    }
    else
    {
        glDisable(GL_LIGHT0);
        glEnable(GL_LIGHT1);
    }


                ••••••
```

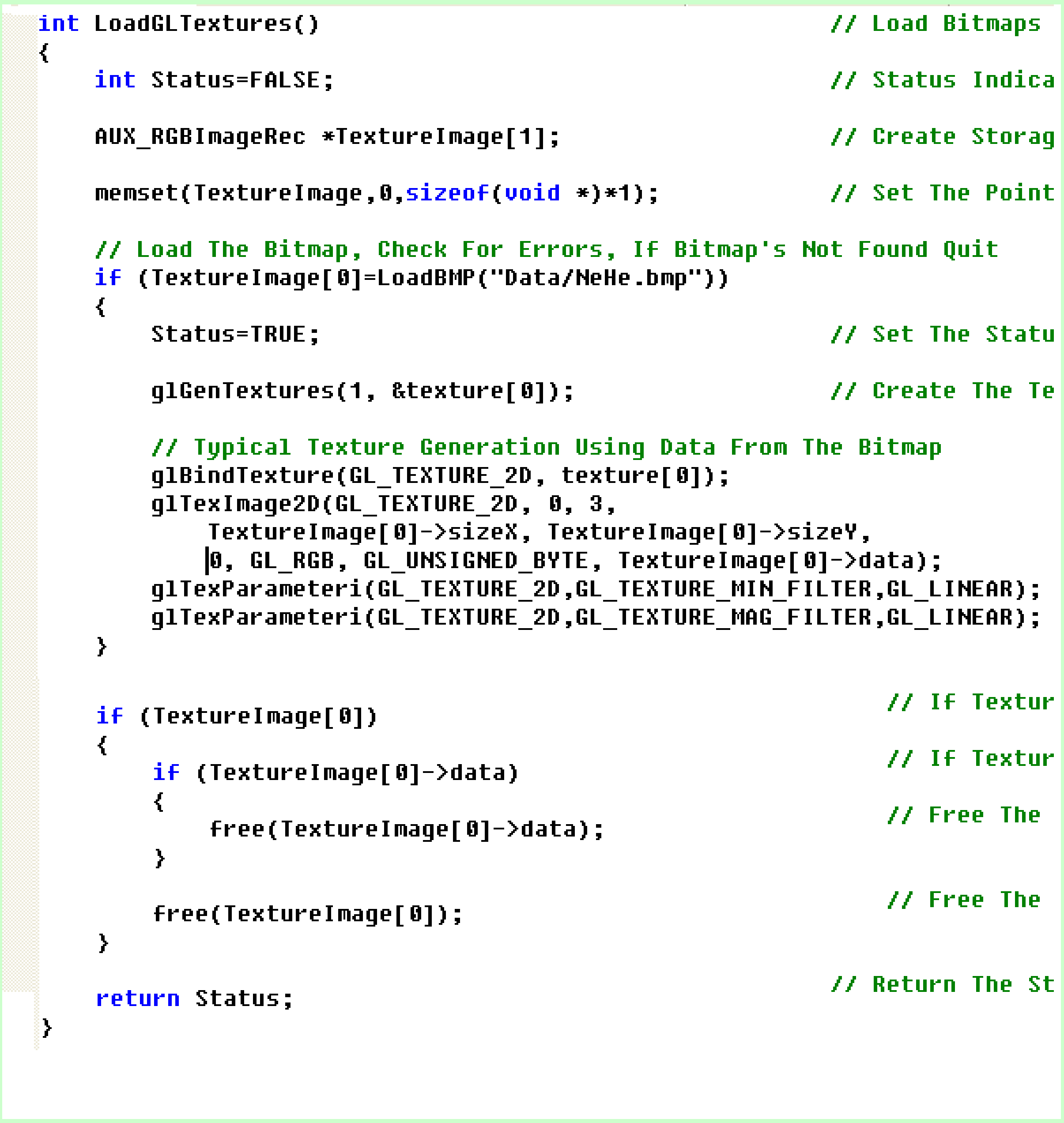
对场景中的某些物体进行单张纹理贴图

加载图像



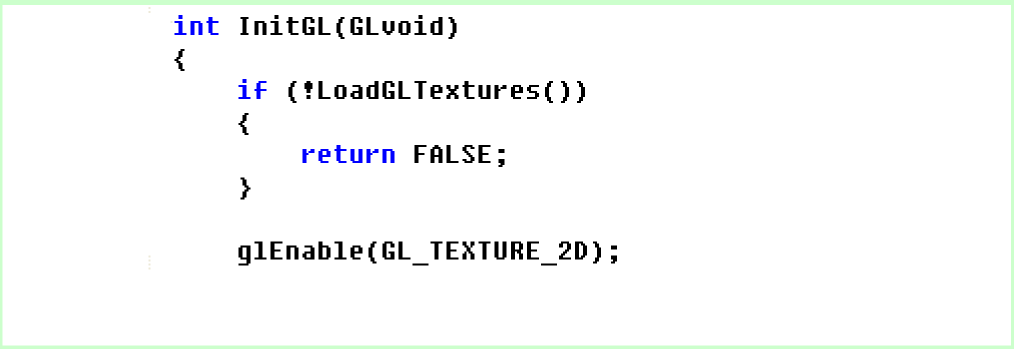
```
#include <stdio.h>            // Header File Fo
GLuint  texture[1];           // Storage For On
|    ......
AUX_RGBImageRec *LoadBMP(char *Filename)
{
    FILE *File=NULL;

    if (!Filename)
    {
        return NULL;
    }

    File=fopen(Filename,"r");

    if (File)
    {
        fclose(File);
        return auxDIBImageLoad(Filename);
    }

    return NULL;
}
   ... ...
```

图像转换成纹理



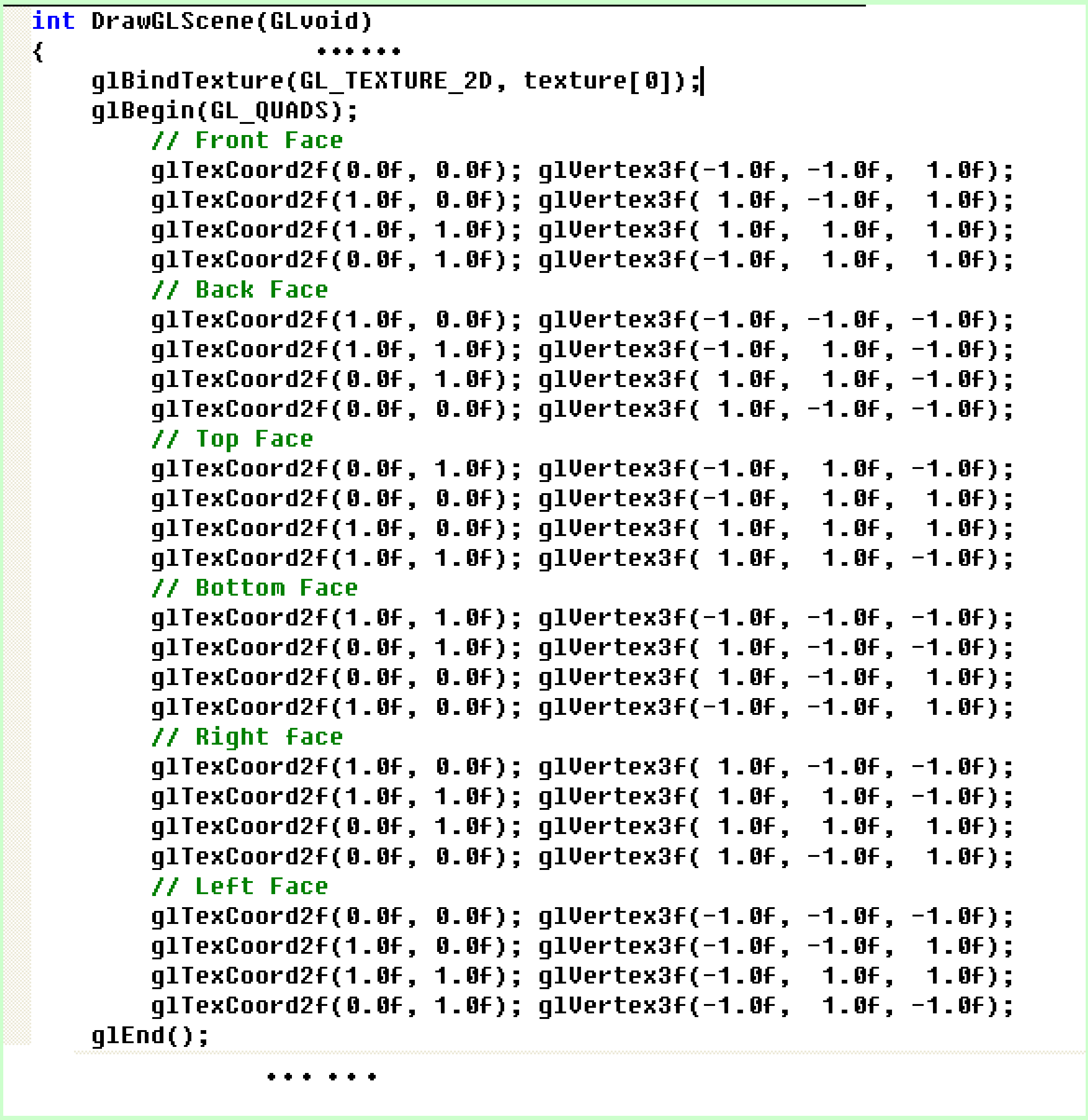
```
int LoadGLTextures()                                    // Load Bitmaps
{
    int Status=FALSE;                                   // Status Indica

    AUX_RGBImageRec *TextureImage[1];                   // Create Storag

    memset(TextureImage,0,sizeof(void *)*1);            // Set The Point

    // Load The Bitmap, Check For Errors, If Bitmap's Not Found Quit
    if (TextureImage[0]=LoadBMP("Data/NeHe.bmp"))
    {
        Status=TRUE;                                    // Set The Statu

        glGenTextures(1, &texture[0]);                  // Create The Te

        // Typical Texture Generation Using Data From The Bitmap
        glBindTexture(GL_TEXTURE_2D, texture[0]);
        glTexImage2D(GL_TEXTURE_2D, 0, 3,
            TextureImage[0]->sizeX, TextureImage[0]->sizeY,
            0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage[0]->data);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
    }

                                                            // If Textur
    if (TextureImage[0])
    {                                                       // If Textur
        if (TextureImage[0]->data)
        {                                                   // Free The
            free(TextureImage[0]->data);
        }
                                                            // Free The
        free(TextureImage[0]);
    }
                                                        // Return The St
    return Status;
}
```

初始化纹理



```
int InitGL(GLvoid)
{
    if (!LoadGLTextures())
    {
        return FALSE;
    }

    glEnable(GL_TEXTURE_2D);
```

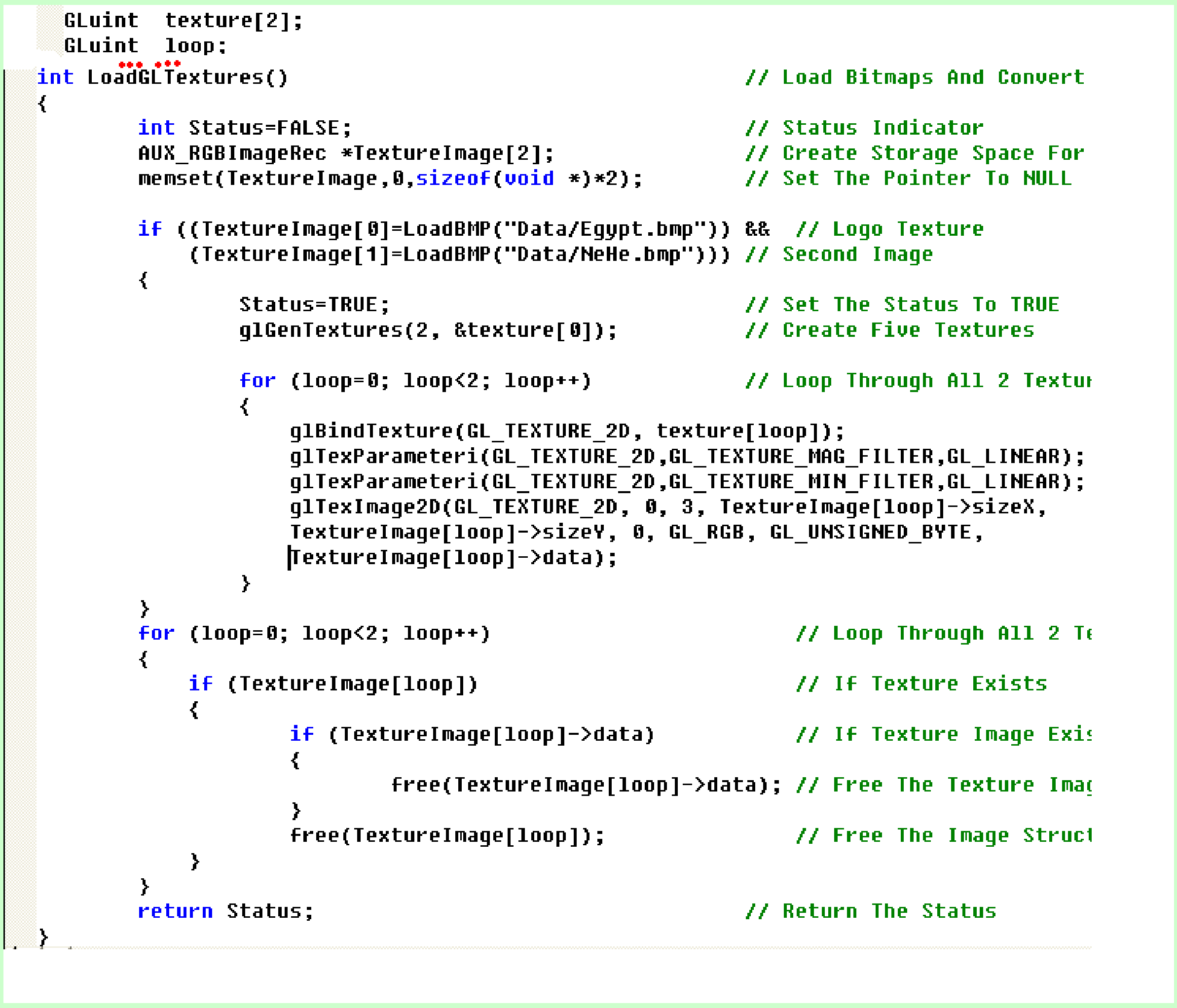
纹理贴图



```
int DrawGLScene(GLvoid)
{                   ••••••
    glBindTexture(GL_TEXTURE_2D, texture[0]);
    glBegin(GL_QUADS);
        // Front Face
        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);
        // Back Face
        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);
        // Top Face
        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);
        // Bottom Face
        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);
        // Right face
        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);
        // Left Face
        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();
                ••• •••
```

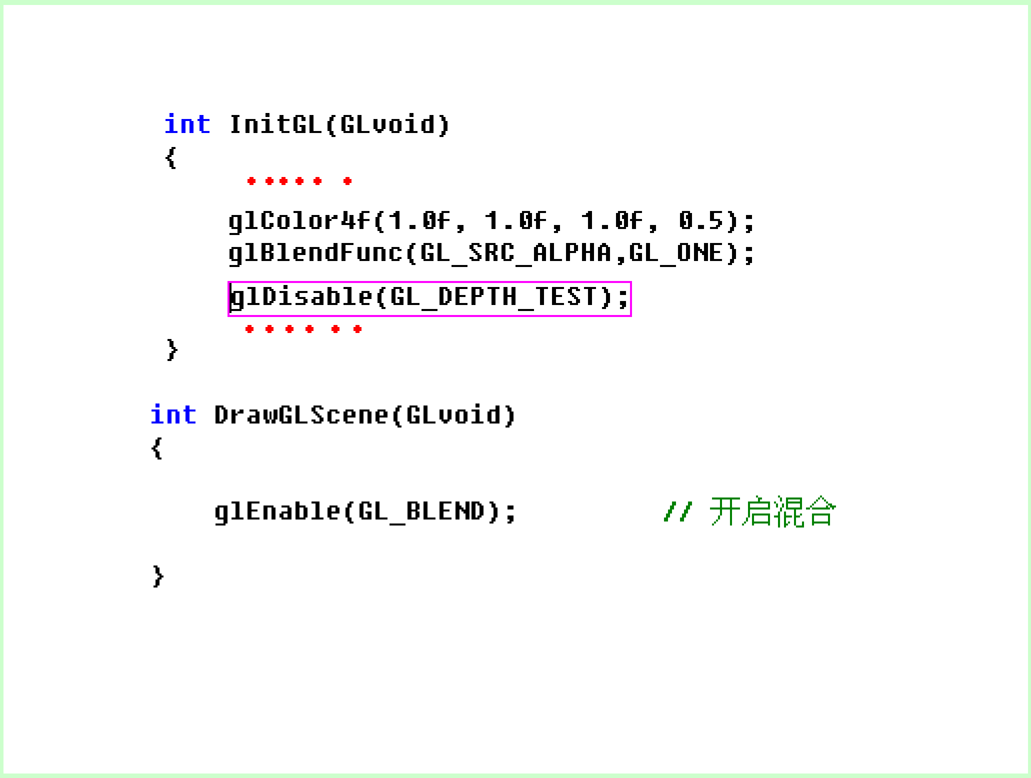
对场景中的某些物体进行多张纹理贴图

只需修改图像转换成纹理部分



```
    GLuint  texture[2];
    GLuint  loop;
int LoadGLTextures()                                   // Load Bitmaps And Convert
{
        int Status=FALSE;                              // Status Indicator
        AUX_RGBImageRec *TextureImage[2];              // Create Storage Space For
        memset(TextureImage,0,sizeof(void *)*2);       // Set The Pointer To NULL

        if ((TextureImage[0]=LoadBMP("Data/Egypt.bmp")) &&  // Logo Texture
            (TextureImage[1]=LoadBMP("Data/NeHe.bmp"))) // Second Image
        {
                Status=TRUE;                           // Set The Status To TRUE
                glGenTextures(2, &texture[0]);         // Create Five Textures

                for (loop=0; loop<2; loop++)           // Loop Through All 2 Textur
                {
                    glBindTexture(GL_TEXTURE_2D, texture[loop]);
                    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, 3, TextureImage[loop]->sizeX,
                    TextureImage[loop]->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,
                    TextureImage[loop]->data);
                }
        }
        for (loop=0; loop<2; loop++)                       // Loop Through All 2 Te
        {
            if (TextureImage[loop])                        // If Texture Exists
            {
                    if (TextureImage[loop]->data)          // If Texture Image Exis
                    {
                            free(TextureImage[loop]->data); // Free The Texture Imag
                    }
                    free(TextureImage[loop]);              // Free The Image Struct
            }
        }
        return Status;                                 // Return The Status
}
```

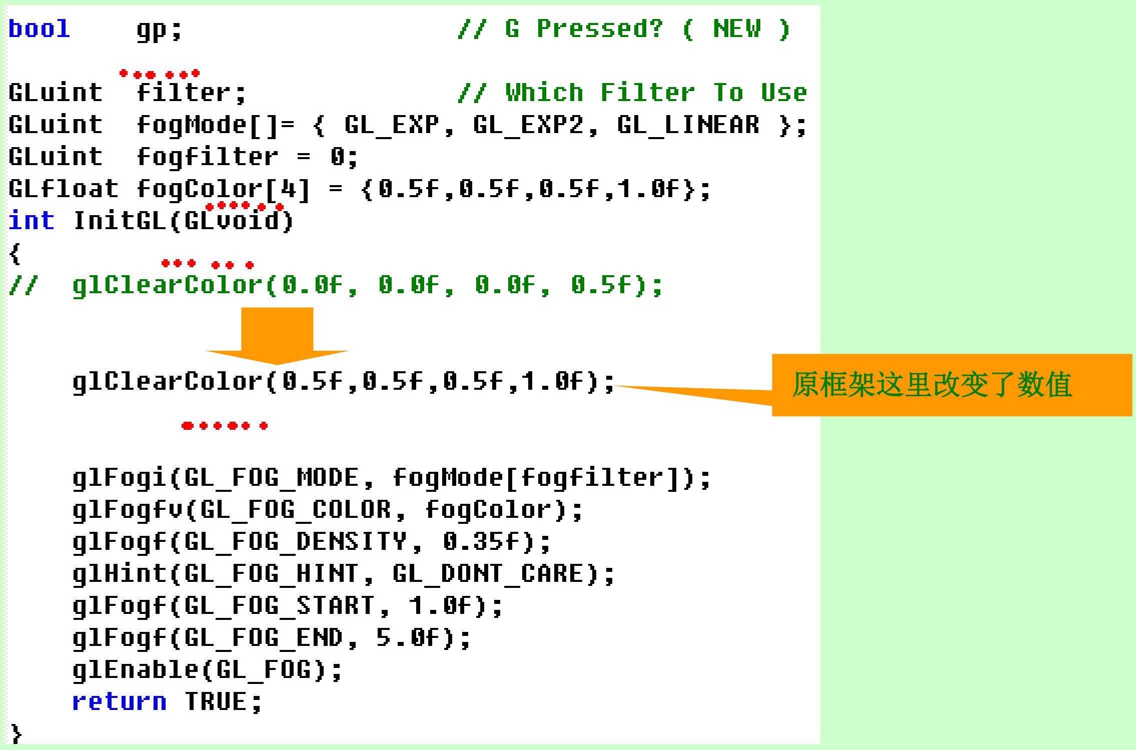
对纹理贴图后的区域实现混合效果



```
int InitGL(GLvoid)
{
    ...... .
    glColor4f(1.0f, 1.0f, 1.0f, 0.5);
    glBlendFunc(GL_SRC_ALPHA,GL_ONE);
    glDisable(GL_DEPTH_TEST);
    ...... ..
}

int DrawGLScene(GLvoid)
{

    glEnable(GL_BLEND);          // 开启混合

}
```

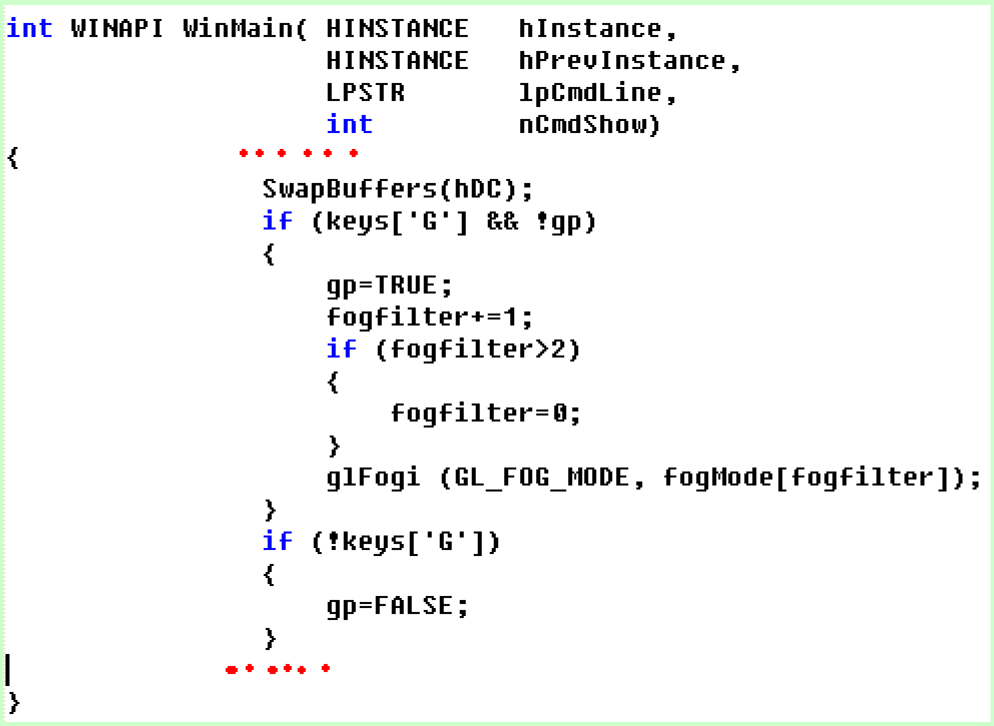
为场景添加雾气

初始化



```
bool    gp;                     // G Pressed? ( NEW )

GLuint  filter;                 // Which Filter To Use
GLuint  fogMode[]= { GL_EXP, GL_EXP2, GL_LINEAR };
GLuint  fogfilter = 0;
GLfloat fogColor[4] = {0.5f,0.5f,0.5f,1.0f};
int InitGL(GLvoid)
{
//  glClearColor(0.0f, 0.0f, 0.0f, 0.5f);


    glClearColor(0.5f,0.5f,0.5f,1.0f);


    glFogi(GL_FOG_MODE, fogMode[fogfilter]);
    glFogfv(GL_FOG_COLOR, fogColor);
    glFogf(GL_FOG_DENSITY, 0.35f);
    glHint(GL_FOG_HINT, GL_DONT_CARE);
    glFogf(GL_FOG_START, 1.0f);
    glFogf(GL_FOG_END, 5.0f);
    glEnable(GL_FOG);
    return TRUE;
}
```

键盘切换雾的种类



```
int WINAPI WinMain( HINSTANCE   hInstance,
                    HINSTANCE   hPrevInstance,
                    LPSTR       lpCmdLine,
                    int         nCmdShow)
{
                SwapBuffers(hDC);
                if (keys['G'] && !gp)
                {
                    gp=TRUE;
                    fogfilter+=1;
                    if (fogfilter>2)
                    {
                        fogfilter=0;
                    }
                    glFogi (GL_FOG_MODE, fogMode[fogfilter]);
                }
                if (!keys['G'])
                {
                    gp=FALSE;
                }

}
```

地月系

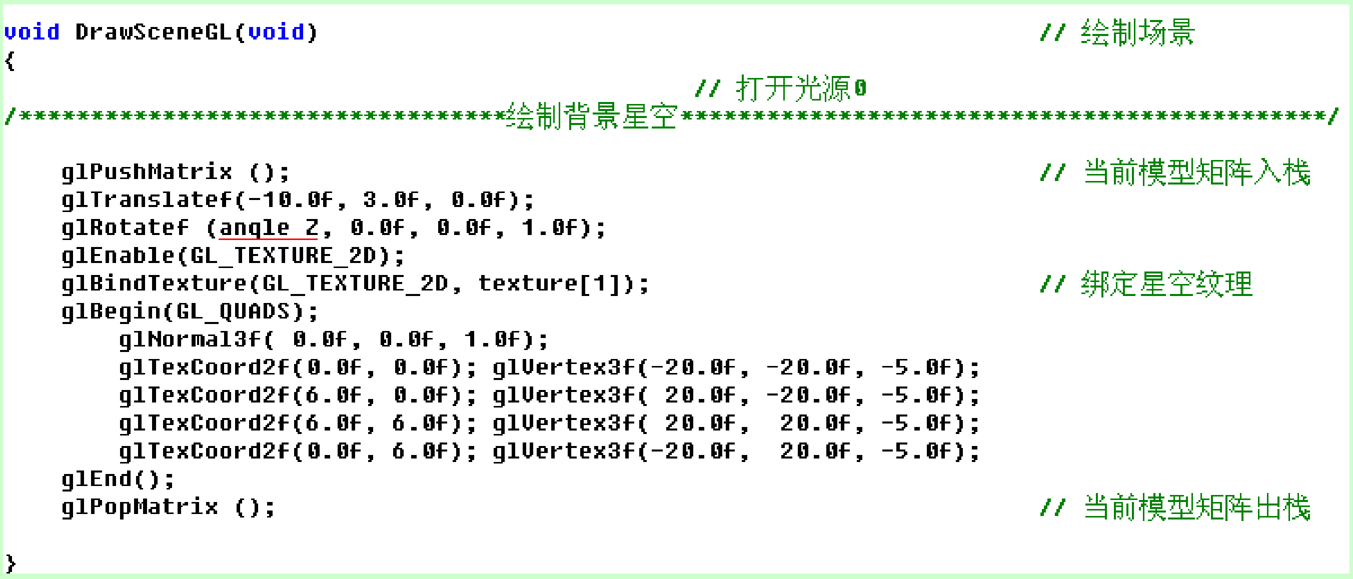
变量定义

```
GLfloat ep_Angle;
GLfloat es_Angle;
GLfloat mp_Angle;
GLfloat ms_Angle;
GLUquadricObj *quadric;
GLuint  texture[4];         // Storage For Our Five Textures
GLfloat angle_Z;

GLuint  loop;               // Generic Loop Variable

GLfloat LightAmbient[] = { 1.0f, 1.0f, 1.0f, 0.0f };
GLfloat LightDiffuse[] = { 1.0f, 1.0f, 1.0f, 0.0f };
GLfloat LightPosition[] = { 0.0f, 0.0f, 0.0f, 1.0f };

GLfloat fogColor[4] = {0.5f, 0.0f, 0.5f, 1.0f};
```

初始化

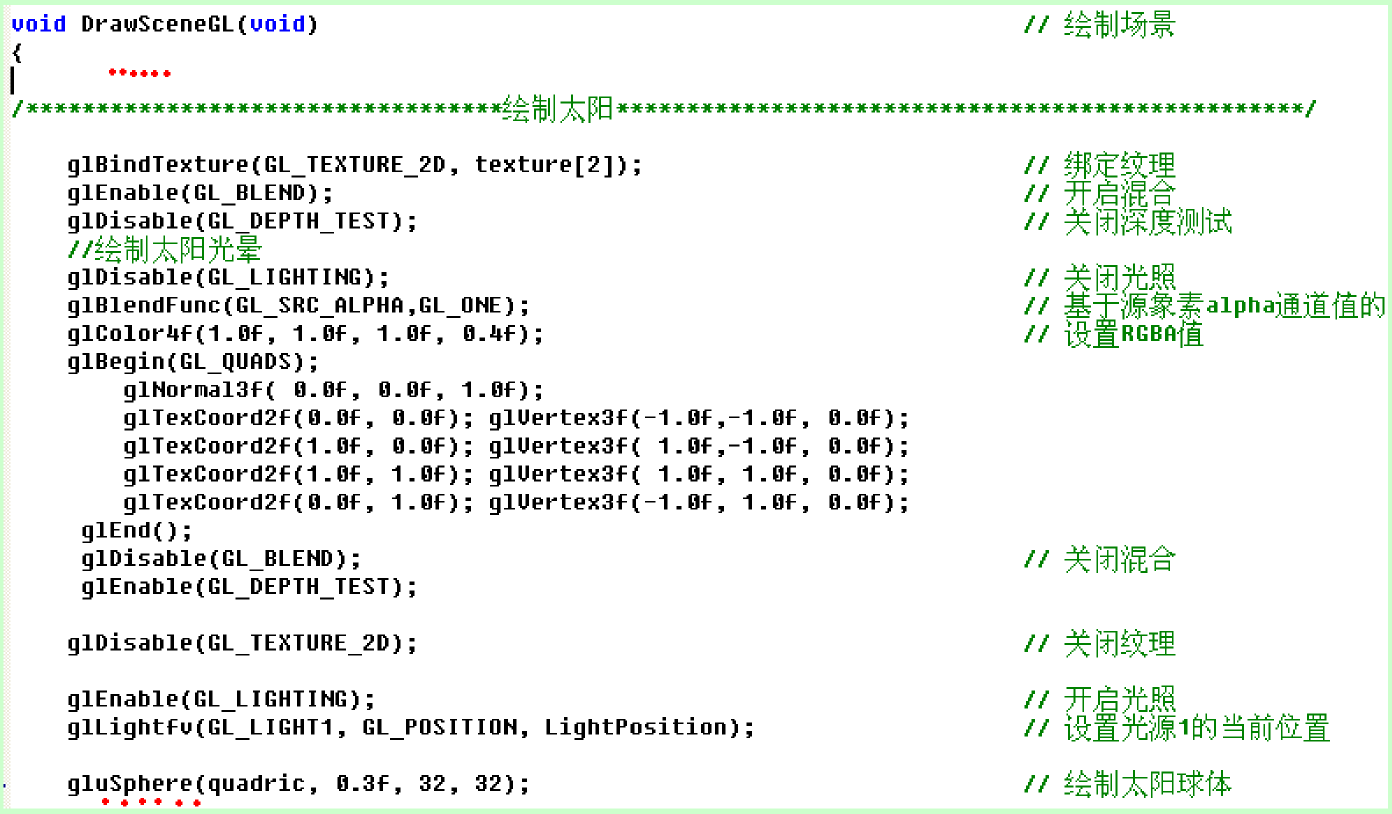
```
int InitGL(GLvoid)
{
    if (!LoadGLTextures())
    {
        return FALSE;
    }

    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClearDepth(1.0);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_TEXTURE_2D);

    glLightfv(GL_LIGHT1, GL_AMBIENT, LightAmbient);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, LightDiffuse);

    quadric = gluNewQuadric();
    gluQuadricTexture(quadric, GLU_TRUE);
    gluQuadricDrawStyle(quadric, GLU_FILL);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT1);
    // 设置雾的各种参数
    glFogi(GL_FOG_MODE, GL_LINEAR);
    glFogfv(GL_FOG_COLOR, fogColor);
    glFogf(GL_FOG_DENSITY, 0.6f);
    glHint(GL_FOG_HINT, GL_DONT_CARE);
    glFogf(GL_FOG_START, 1.0f);
    glFogf(GL_FOG_END, 30.0f);
    glEnable(GL_FOG);
    return TRUE;
}
```

旋转的背景



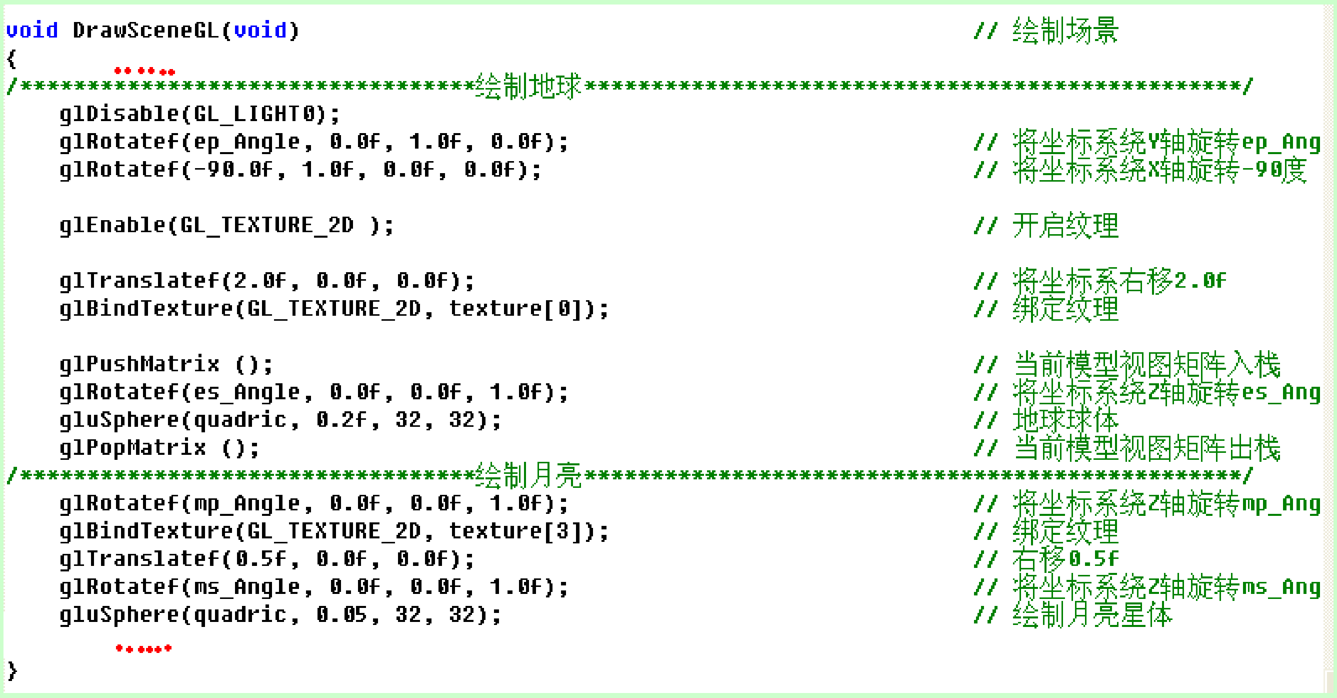
```
void DrawSceneGL(void)                                                   // 绘制场景
{
                                                   // 打开光源0
/***********************************绘制背景星空**********************************************/

    glPushMatrix ();                                                     // 当前模型矩阵入栈
    glTranslatef(-10.0f, 3.0f, 0.0f);
    glRotatef (angle_Z, 0.0f, 0.0f, 1.0f);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, texture[1]);                            // 绑定星空纹理
    glBegin(GL_QUADS);
        glNormal3f( 0.0f, 0.0f, 1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-20.0f, -20.0f, -5.0f);
        glTexCoord2f(6.0f, 0.0f); glVertex3f( 20.0f, -20.0f, -5.0f);
        glTexCoord2f(6.0f, 6.0f); glVertex3f( 20.0f,  20.0f, -5.0f);
        glTexCoord2f(0.0f, 6.0f); glVertex3f(-20.0f,  20.0f, -5.0f);
    glEnd();
    glPopMatrix ();                                                      // 当前模型矩阵出栈

}
```

太阳



```
void DrawSceneGL(void)                                                   // 绘制场景
{
        ......
/***********************************绘制太阳**********************************************/

    glBindTexture(GL_TEXTURE_2D, texture[2]);                            // 绑定纹理
    glEnable(GL_BLEND);                                                  // 开启混合
    glDisable(GL_DEPTH_TEST);                                            // 关闭深度测试
    //绘制太阳光晕
    glDisable(GL_LIGHTING);                                              // 关闭光照
    glBlendFunc(GL_SRC_ALPHA,GL_ONE);                                    // 基于源象素alpha通道值的
    glColor4f(1.0f, 1.0f, 1.0f, 0.4f);                                   // 设置RGBA值
    glBegin(GL_QUADS);
        glNormal3f( 0.0f, 0.0f, 1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f,-1.0f, 0.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f,-1.0f, 0.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 0.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 0.0f);
     glEnd();
     glDisable(GL_BLEND);                                                // 关闭混合
     glEnable(GL_DEPTH_TEST);

    glDisable(GL_TEXTURE_2D);                                            // 关闭纹理

    glEnable(GL_LIGHTING);                                               // 开启光照
    glLightfv(GL_LIGHT1, GL_POSITION, LightPosition);                    // 设置光源1的当前位置

    gluSphere(quadric, 0.3f, 32, 32);                                    // 绘制太阳球体
       ......
```

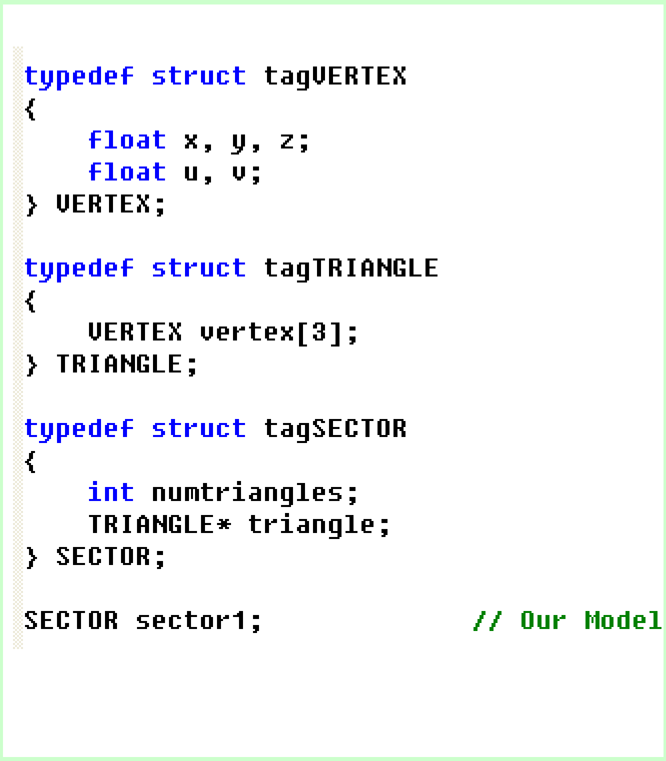
地球和月亮



```
void DrawSceneGL(void)                                                    // 绘制场景
{
        ……
/**********************************绘制地球****************************************************/
    glDisable(GL_LIGHT0);
    glRotatef(ep_Angle, 0.0f, 1.0f, 0.0f);                                // 将坐标系绕Y轴旋转ep_Ang
    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);                                  // 将坐标系绕X轴旋转-90度

    glEnable(GL_TEXTURE_2D );                                             // 开启纹理

    glTranslatef(2.0f, 0.0f, 0.0f);                                       // 将坐标系右移2.0f
    glBindTexture(GL_TEXTURE_2D, texture[0]);                             // 绑定纹理

    glPushMatrix ();                                                      // 当前模型视图矩阵入栈
    glRotatef(es_Angle, 0.0f, 0.0f, 1.0f);                                // 将坐标系绕Z轴旋转es_Ang
    gluSphere(quadric, 0.2f, 32, 32);                                     // 地球球体
    glPopMatrix ();                                                       // 当前模型视图矩阵出栈
/**********************************绘制月亮****************************************************/
    glRotatef(mp_Angle, 0.0f, 0.0f, 1.0f);                                // 将坐标系绕Z轴旋转mp_Ang
    glBindTexture(GL_TEXTURE_2D, texture[3]);                             // 绑定纹理
    glTranslatef(0.5f, 0.0f, 0.0f);                                       // 右移0.5f
    glRotatef(ms_Angle, 0.0f, 0.0f, 1.0f);                                // 将坐标系绕Z轴旋转ms_Ang
    gluSphere(quadric, 0.05, 32, 32);                                     // 绘制月亮星体
        ……
}
```

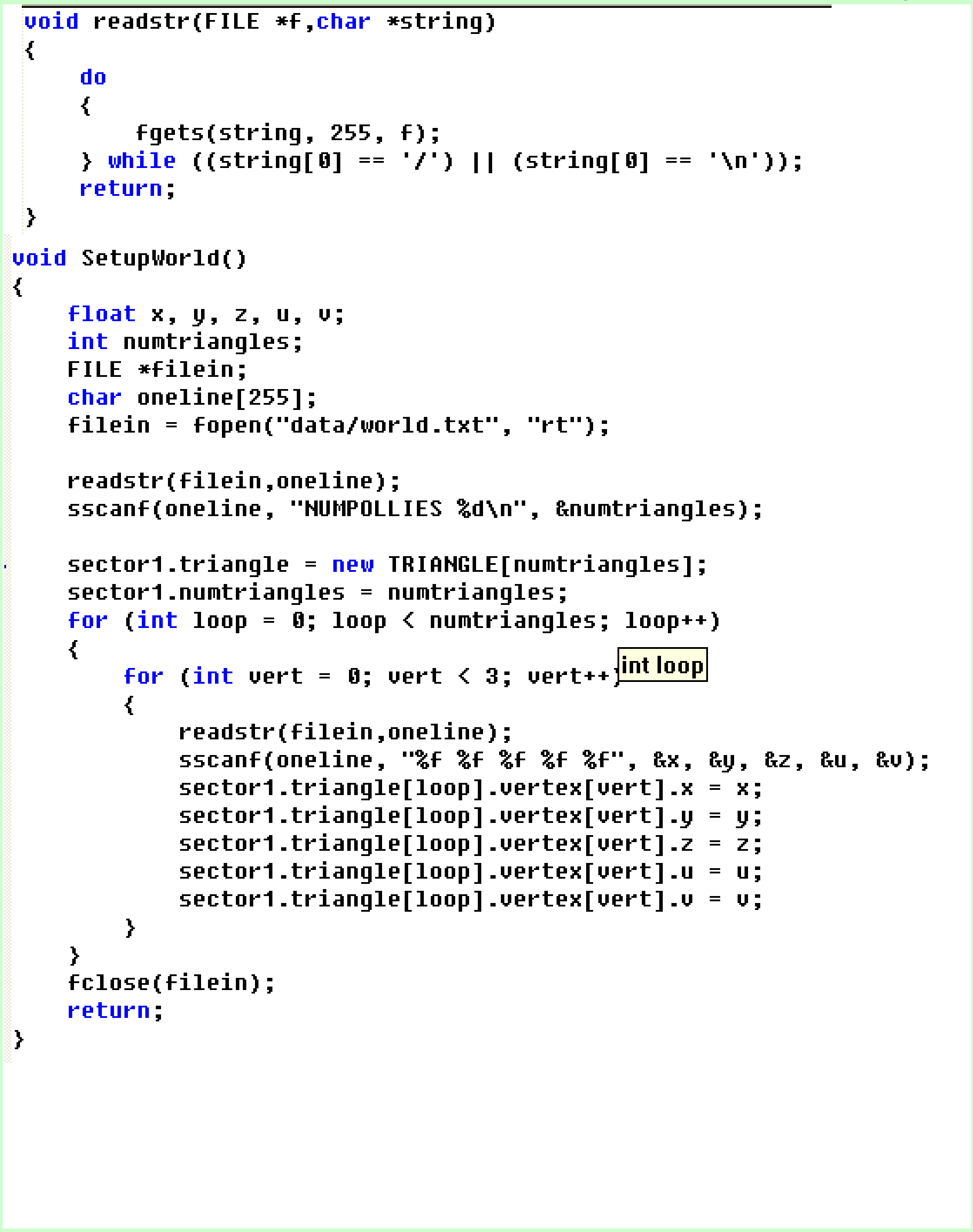
3D 场景

定义结构体（重要!）



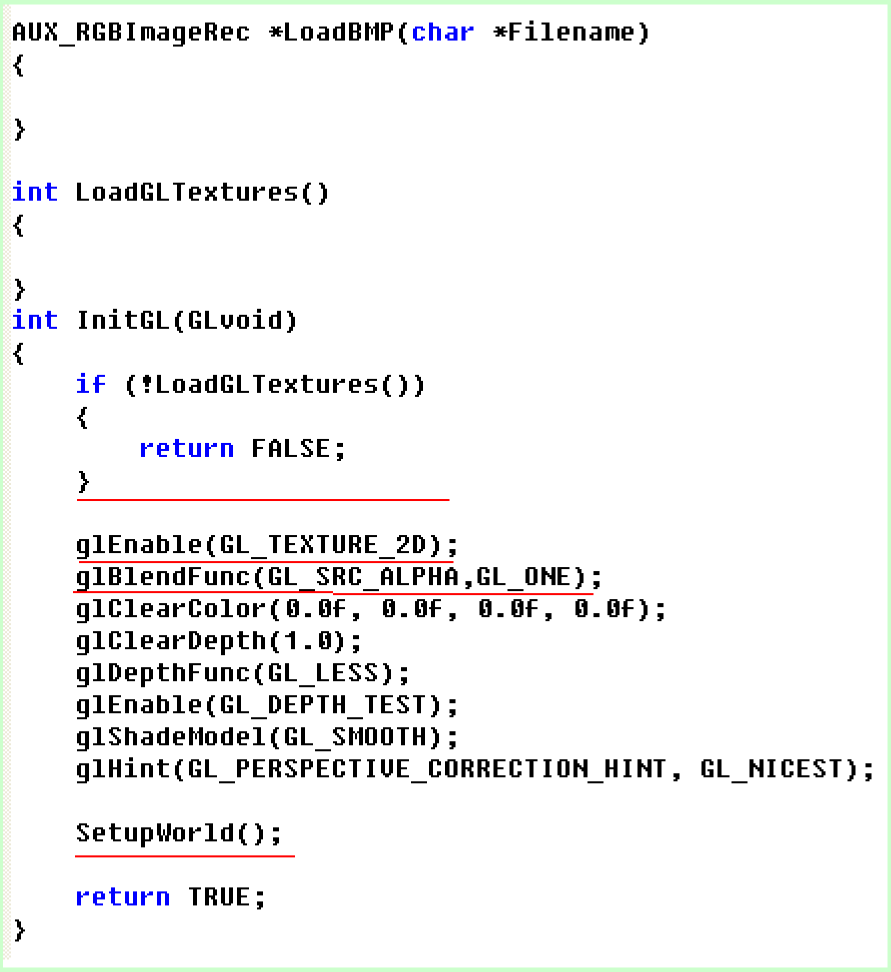
```
typedef struct tagVERTEX
{
    float x, y, z;
    float u, v;
} VERTEX;

typedef struct tagTRIANGLE
{
    VERTEX vertex[3];
} TRIANGLE;

typedef struct tagSECTOR
{
    int numtriangles;
    TRIANGLE* triangle;
} SECTOR;

SECTOR sector1;             // Our Model
```

自定义函数



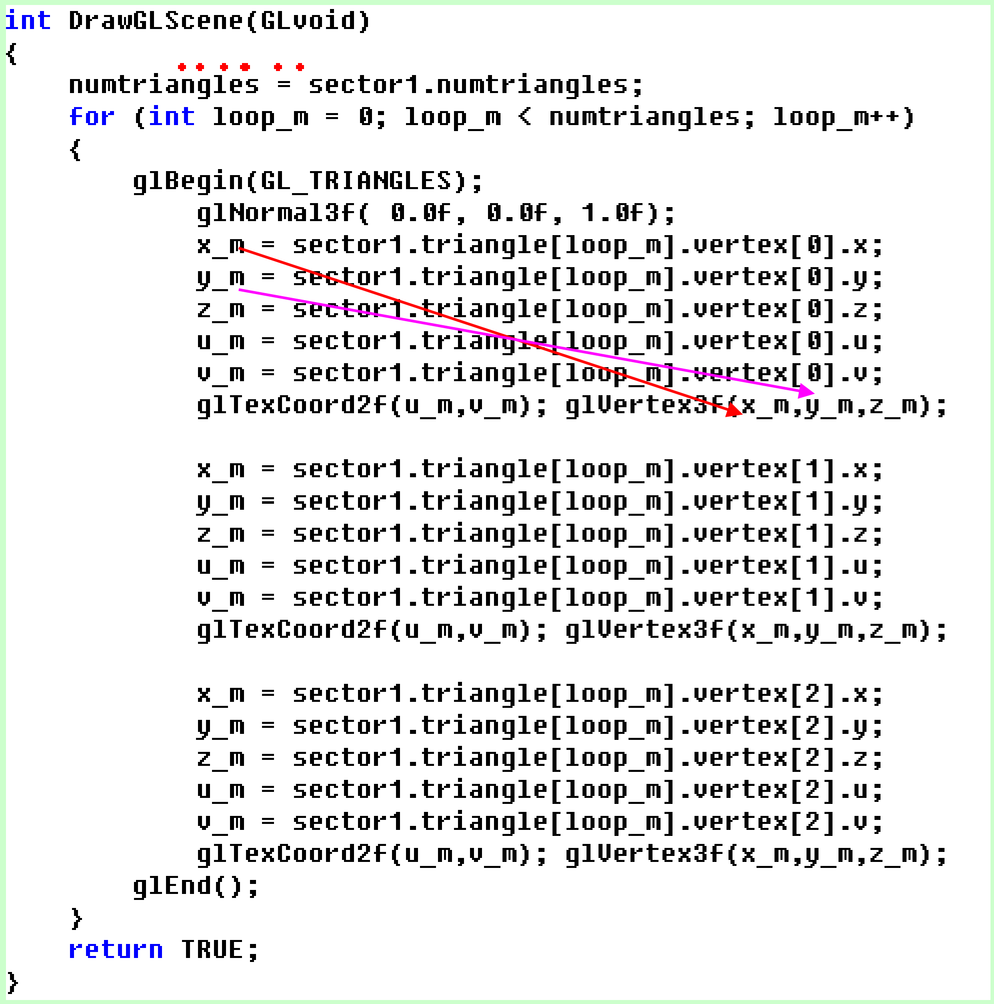
```
void readstr(FILE *f,char *string)
{
    do
    {
        fgets(string, 255, f);
    } while ((string[0] == '/') || (string[0] == '\n'));
    return;
}
void SetupWorld()
{
    float x, y, z, u, v;
    int numtriangles;
    FILE *filein;
    char oneline[255];
    filein = fopen("data/world.txt", "rt");

    readstr(filein,oneline);
    sscanf(oneline, "NUMPOLLIES %d\n", &numtriangles);

    sector1.triangle = new TRIANGLE[numtriangles];
    sector1.numtriangles = numtriangles;
    for (int loop = 0; loop < numtriangles; loop++)
    {
        for (int vert = 0; vert < 3; vert++)
        {
            readstr(filein,oneline);
            sscanf(oneline, "%f %f %f %f %f", &x, &y, &z, &u, &v);
            sector1.triangle[loop].vertex[vert].x = x;
            sector1.triangle[loop].vertex[vert].y = y;
            sector1.triangle[loop].vertex[vert].z = z;
            sector1.triangle[loop].vertex[vert].u = u;
            sector1.triangle[loop].vertex[vert].v = v;
        }
    }
    fclose(filein);
    return;
}
```

初始化



```
AUX_RGBImageRec *LoadBMP(char *Filename)
{

}

int LoadGLTextures()
{

}
int InitGL(GLvoid)
{
    if (!LoadGLTextures())
    {
        return FALSE;
    }

    glEnable(GL_TEXTURE_2D);
    glBlendFunc(GL_SRC_ALPHA,GL_ONE);
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClearDepth(1.0);
    glDepthFunc(GL_LESS);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_SMOOTH);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    SetupWorld();

    return TRUE;
}
```

场景绘制



```
int DrawGLScene(GLvoid)
{
    numtriangles = sector1.numtriangles;
    for (int loop_m = 0; loop_m < numtriangles; loop_m++)
    {
        glBegin(GL_TRIANGLES);
            glNormal3f( 0.0f, 0.0f, 1.0f);
            x_m = sector1.triangle[loop_m].vertex[0].x;
            y_m = sector1.triangle[loop_m].vertex[0].y;
            z_m = sector1.triangle[loop_m].vertex[0].z;
            u_m = sector1.triangle[loop_m].vertex[0].u;
            v_m = sector1.triangle[loop_m].vertex[0].v;
            glTexCoord2f(u_m,v_m); glVertex3f(x_m,y_m,z_m);

            x_m = sector1.triangle[loop_m].vertex[1].x;
            y_m = sector1.triangle[loop_m].vertex[1].y;
            z_m = sector1.triangle[loop_m].vertex[1].z;
            u_m = sector1.triangle[loop_m].vertex[1].u;
            v_m = sector1.triangle[loop_m].vertex[1].v;
            glTexCoord2f(u_m,v_m); glVertex3f(x_m,y_m,z_m);

            x_m = sector1.triangle[loop_m].vertex[2].x;
            y_m = sector1.triangle[loop_m].vertex[2].y;
            z_m = sector1.triangle[loop_m].vertex[2].z;
            u_m = sector1.triangle[loop_m].vertex[2].u;
            v_m = sector1.triangle[loop_m].vertex[2].v;
            glTexCoord2f(u_m,v_m); glVertex3f(x_m,y_m,z_m);
        glEnd();
    }
    return TRUE;
}
```

星星场景

定义

```
bool     twinkle;            // Twinkling Stars
bool     tp;                 // 'T' Key Pressed?

const    num=50;             // Number Of Stars To Draw

typedef struct               // Create A Structure For Star
{
    int r, g, b;             // Stars Color
    GLfloat dist,            // Stars Distance From Center
            angle;           // Stars Current Angle
}
stars;
stars star[num];             // Need To Keep Track Of 'num' Stars

GLfloat zoom=-15.0f;         // Distance Away From Stars
GLfloat tilt=90.0f;          // Tilt The View
GLfloat spin;                // Spin Stars

GLuint  loop;                // General Loop Variable
GLuint  texture[1];          // Storage For One textures
```

初始化

```
int InitGL(GLvoid)
{
    for (loop=0; loop<num; loop++)
    {
        star[loop].angle=0.0f;
        star[loop].dist=(float(loop)/num)*5.0f;
        star[loop].r=rand()%256;
        star[loop].g=rand()%256;
        star[loop].b=rand()%256;
    }
}
```

场景绘制

```
int DrawGLScene(GLvoid)                                  // Here's Where We Do All The Drawing
{   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Clear The Screen And The Depth Buffer
    glBindTexture(GL_TEXTURE_2D, texture[0]);           // Select Our Texture
    for (loop=0; loop<num; loop++)                      // Loop Through All The Stars
    {   glLoadIdentity();                               // Reset The View Before We Draw Each St
        glTranslatef(0.0f,0.0f,zoom);                   // Zoom Into The Screen (Using The Value
        glRotatef(tilt,1.0f,0.0f,0.0f);                 // Tilt The View (Using The Value In 'ti
        glRotatef(star[loop].angle,0.0f,1.0f,0.0f);     // Rotate To The Current Stars Angle
        glTranslatef(star[loop].dist,0.0f,0.0f);        // Move Forward On The X Plane
        glRotatef(-star[loop].angle,0.0f,1.0f,0.0f);    // Cancel The Current Stars Angle
        glRotatef(-tilt,1.0f,0.0f,0.0f);                // Cancel The Screen Tilt
        if (twinkle)
        {   glColor4ub(star[(num-loop)-1].r,star[(num-loop)-1].g,star[(num-loop)-1].b,255);
            glBegin(GL_QUADS);
                glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f,-1.0f, 0.0f);
                glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f,-1.0f, 0.0f);
                glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 0.0f);
                glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 0.0f);
            glEnd();}
        glRotatef(spin,0.0f,0.0f,1.0f);glColor4ub(star[loop].r,star[loop].g,star[loop].b,255);
        glBegin(GL_QUADS);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f,-1.0f, 0.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f,-1.0f, 0.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f, 1.0f, 0.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 0.0f);
        glEnd();
        spin+=0.01f;star[loop].angle+=float(loop)/num;star[loop].dist-=0.01f;
        if (star[loop].dist<0.0f)
        {star[loop].dist+=5.0f;star[loop].r=rand()%256;
         star[loop].g=rand()%256;star[loop].b=rand()%256;}
    }
    return  TRUE;
}
```