|  |  |  |
| --- | --- | --- |
| **计算机图形学**  **实验报告** | | |
|  | |  |
| 图片包含 标牌  已生成极高可信度的说明 | | |
|  | | |
| **姓名** |  |  |
| **班级** |  |  |
| **学号** |  |  |
| **日期** | 2023-12-29 |  |

# 一、实验目的

1、了解和掌握OpenGL的基本命令。

2、掌握纹理映射以及利用鼠标与系统进行交互。

# 二、实验内容与要求

地球仪绘制：OpenGL绘制球体，图片作为纹理映射到整个球面上，双点触控缩放球体，拖动旋转球体。

基本要求：

1. 绘制圆球，采用纹理映射的方式将给定的世界地图贴到圆球上（纹理贴图无缝），也可以使用其它世界地图；

2. 需要具有简单的光照和材质效果；

3. 双点触控缩放球体，拖动旋转球体；

附加要求：

4. 增加支架，生成真实的地球仪；

5. 加入光照，阴影，增加逼真度。

# 三、主要函数和功能

1. **initLight():** 初始化场景的光照，指定环境光、漫反射光和镜面光的属性，并设置光源的位置。
2. **initRendering():** 初始化渲染设置，包括创建地球的二次曲面对象（用于绘制球体）和使用**initLight()**初始化光照。
3. **load\_texture(const char file\_name)：**加载BMP图像文件并返回纹理ID。BMP文件用作地球模型的纹理。
4. **drawEarth():** 使用加载的纹理绘制带有纹理的地球。
5. **drawSolidTorus():** 绘制一个带有裁剪平面的实心圆环，以呈现环绕地球的外观。还包括环的顶部和底部各一个圆锥。
6. **drawBase():** 绘制一个锥形底座，用于支撑地球和圆环。
7. **drawScene():** 绘制整个场景，包括地球、圆环和底座。应用旋转和缩放变换。
8. **handleKey(unsigned char key, int x, int y):** 处理键盘输入。按下'w'时，放大地球、圆环，并将底座下移。按下's'时，缩小地球、圆环，并将底座上移。
9. **handleMotion(int x, int y):** 处理鼠标移动，允许用户水平旋转场景。
10. **handleResize(int w, int h):** 处理窗口调整，相应调整视口和投影矩阵。
11. **update(int value):** 更新地球旋转角度以实现动画效果，并触发重绘。设置定时器以实现持续动画。
12. **main():** 主函数设置GLUT窗口，初始化渲染，加载地球纹理，并为显示、键盘输入、窗口调整、鼠标移动和动画设置回调函数。程序进入GLUT主循环，持续渲染场景并响应用户输入。

# 四、代码

#include <GL/glut.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

static float angle = 0.0f;

static int oldX;//鼠标点击位置

static float rotX = 0;

static float rotY = 0;

static float xyz[3] = { 1,1,1 };

static float r = 80;//球体半径

static float rout = 92;

static float rin = 7;//圆环半径

static float incAngle = 15.0f;//倾角

static float basePos = -120.0f;//底座位置

static float axiPos = 90;//轴位置

static GLUquadric\* earth;//球体对象

#define BMP\_Header\_Length 54  //图像数据在内存块中的偏移量

//定义纹理对象编号

GLuint texEarth;

//初始化光照

void initLight() {

    GLfloat ambientLight[] = { 0.2, 0.2, 0.2, 1 };

    GLfloat diffuseLight[] = { 0.8, 0.8, 0.8, 1 };

    GLfloat specularLight[] = { 0.5, 0.5, 0.5, 1 };

    GLfloat posLight[] = { 400, 250, 1, 1 };

    GLfloat specref[] = { 1, 1, 1, 1 };

    glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambientLight);

    glLightfv(GL\_LIGHT0,GL\_POSITION,posLight); //指定光源位置

    glLightfv(GL\_LIGHT0, GL\_AMBIENT, ambientLight);

    glLightfv(GL\_LIGHT0, GL\_DIFFUSE, diffuseLight);

    glLightfv(GL\_LIGHT0, GL\_SPECULAR, specularLight);

    glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 128);

}

//初始化

void initRendering() {

    //创建地球对象

    earth = gluNewQuadric();

    //初始化光照

    initLight();

    glEnable(GL\_LIGHT0);

    glEnable(GL\_LIGHTING);

    glEnable(GL\_DEPTH\_TEST);

    glEnable(GL\_LINE\_SMOOTH);

}

// 判断是不是2的整数次幂

int power\_of\_two(int n) {

    if (n <= 0) {

        return 0;

    }

    return (n & (n - 1)) == 0;

}

// 将一个BMP文件作为纹理载入

GLuint load\_texture(const char\* file\_name) {

    GLint width, height, total\_bytes;

    GLubyte\* pixels = 0;

    GLuint last\_texture\_ID = 0, texture\_ID = 0;

    // 打开文件，如果失败，返回

    FILE\* pFile;

    fopen\_s(&pFile, file\_name, "rb");

    if (pFile == NULL) {

        return 0;

    }

    // 读取文件中图象的宽度和高度

    fseek(pFile, 0x0012, SEEK\_SET);

    fread(&width, 4, 1, pFile);

    fread(&height, 4, 1, pFile);

    fseek(pFile, BMP\_Header\_Length, SEEK\_SET);

    // 计算每行像素所占字节数，并根据此数据计算总像素字节数

    GLint line\_bytes = width \* 3;

    while (line\_bytes % 4 != 0)

        ++line\_bytes;

    total\_bytes = line\_bytes \* height;

    // 根据总像素字节数分配内存

    pixels = (GLubyte\*)malloc(total\_bytes);

    if (pixels == 0) {

        fclose(pFile);

        return 0;

    }

    // 读取像素数据

    if (fread(pixels, total\_bytes, 1, pFile) <= 0) {

        free(pixels);

        fclose(pFile);

        return 0;

    }

    // 缩放

    GLint max;

    glGetIntegerv(GL\_MAX\_TEXTURE\_SIZE, &max);

    if (!power\_of\_two(width) || !power\_of\_two(height) || width > max || height > max) {

        const GLint new\_width = 256;

        const GLint new\_height = 256; // 规定缩放后新的大小为边长的正方形

        GLint new\_line\_bytes, new\_total\_bytes;

        GLubyte\* new\_pixels = 0;

        // 计算每行需要的字节数和总字节数

        new\_line\_bytes = new\_width \* 3;

        while (new\_line\_bytes % 4 != 0)

            ++new\_line\_bytes;

        new\_total\_bytes = new\_line\_bytes \* new\_height;

        // 分配内存

        new\_pixels = (GLubyte\*)malloc(new\_total\_bytes);

        if (new\_pixels == 0) {

            free(pixels);

            fclose(pFile);

            return 0;

        }

        // 进行像素缩放

        gluScaleImage(GL\_RGB, width, height, GL\_UNSIGNED\_BYTE, pixels, new\_width, new\_height, GL\_UNSIGNED\_BYTE, new\_pixels);

        // 释放原来的像素数据，把pixels指向新的像素数据，并重新设置width和height

        free(pixels);

        pixels = new\_pixels;

        width = new\_width;

        height = new\_height;

    }

    // 分配一个新的纹理编号

    glGenTextures(1, &texture\_ID);

    if (texture\_ID == 0) {

        free(pixels);

        fclose(pFile);

        return 0;

    }

    // 绑定新的纹理，载入纹理并设置纹理参数

    // 在绑定前，先获得原来绑定的纹理编号，以便在最后进行恢复

    GLint lastTextureID = last\_texture\_ID;

    glGetIntegerv(GL\_TEXTURE\_BINDING\_2D, &lastTextureID);

    glBindTexture(GL\_TEXTURE\_2D, texture\_ID);

    glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

    glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

    glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

    glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

    glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_REPLACE);

    glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB, width, height, 0, GL\_BGR\_EXT, GL\_UNSIGNED\_BYTE, pixels);

    glBindTexture(GL\_TEXTURE\_2D, lastTextureID);  //恢复之前的纹理绑定

    free(pixels);

    return texture\_ID;

}

//绘制地球

void drawEarth() {

    int loaded = 0;

    //纹理绑定到目标

    glBindTexture(GL\_TEXTURE\_2D, texEarth);

    if (!loaded) {

        //纹理坐标自动生成

        glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_SPHERE\_MAP);

        glTexGeni(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_SPHERE\_MAP);

        //表面生成纹理坐标

        gluQuadricDrawStyle(earth, GL\_FILL);

        gluQuadricNormals(earth, GLU\_SMOOTH);

        gluQuadricTexture(earth, GL\_TRUE);

    }

    //生成球体

    glPushMatrix();

    glEnable(GL\_TEXTURE\_2D);

    glRotatef(-90, 1, 0, 0);

    gluSphere(earth, r, 100, 100);

    glDisable(GL\_TEXTURE\_2D);

    glPopMatrix();

}

//绘制圆环

void drawSolidTorus()

{

    // 定义一个裁剪平面，法向量为(0, 1, 0)，通过原点

    GLdouble plane[] = { 1.0, 0.0, 0.0, 0.0 };

    // 启用裁剪平面

    glEnable(GL\_CLIP\_PLANE0);

    // 设置裁剪平面

    glClipPlane(GL\_CLIP\_PLANE0, plane);

    // 绘制圆环

    glutSolidTorus(rin, rout, 8, 50);

    glPushMatrix();

    glTranslatef(0, axiPos, 0);

    glRotatef(90.0f, 1, 0, 0);

    glutSolidCone(7, 20, 200, 200);

    glPopMatrix();

    glPushMatrix();

    glTranslatef(0, -axiPos, 0);

    glRotatef(90.0f, -1, 0, 0);

    glutSolidCone(7, 20, 200, 200);

    glPopMatrix();

    // 禁用裁剪平面

    glDisable(GL\_CLIP\_PLANE0);

}

//绘制圆锥底座

void drawBase()

{

    glPushMatrix();

    glTranslatef(0.0f, basePos, 0.0f);

    glRotatef(90.0f, -1.0f, 0.0f, 0.0f);

    glutSolidCone(30, 20, 100, 200);

    glPopMatrix();

}

//绘制场景

void drawScene() {

    glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

    glMatrixMode(GL\_MODELVIEW);

    glLoadIdentity();

    glTranslatef(400, 240, -200);

    GLUquadric\* quadric = gluNewQuadric();

    gluDisk(quadric, 0.0, 500.0, 30, 4);

    gluDeleteQuadric(quadric);

    glPushMatrix();

    glScalef(xyz[0], xyz[1], xyz[2]);

    glRotatef(rotX / 100, 0, 1, 0);

    glRotatef(-rotY / 100, 1, 0, 0);

    glPushMatrix();

    glRotatef(incAngle, 0, 0, -1);

    //圆环

    drawSolidTorus();

    glPushMatrix();

    glRotatef(angle, 0, 1, 0);

    //球

    drawEarth();

    glPopMatrix();

    glPopMatrix();

    //底座

    drawBase();

    glPopMatrix();

    glutSwapBuffers();

}

//处理按键操作

void handleKey(unsigned char key, int x, int y) {

    switch (key) {

        //按w则放大

    case 'w':

        r += 10;

        rout += 10;

        basePos -= 10.0f;

        axiPos += 10;

        glutPostRedisplay();

        break;

        //按s则缩小

    case 's':

        r -= 10;

        rout -= 10;

        basePos += 10.0f;

        axiPos -= 10;

        glutPostRedisplay();

        break;

    }

}

//处理鼠标滑动

void handleMotion(int x, int y)

{

    int rx = x - oldX;

    printf("%d\n", rx);

    angle += rx;

    //重画

    glutPostRedisplay();

    oldX = x;

}

//窗口调整调用

void handleResize(int w, int h) {

    if (h == 0) {

        h = 1;

    }

    glViewport(0, 0, w, h);

    glMatrixMode(GL\_PROJECTION);

    glLoadIdentity();

    glOrtho(0, w, 0, h, -1000, 1000);

    glMatrixMode(GL\_MODELVIEW);

}

//自转

void update(int value) {

    angle += 1.5f;

    if (angle > 360) {

        angle -= 360;

    }

    glutPostRedisplay();

    glutTimerFunc(16, update, 0);

}

int main(int argc, char\*\* argv) {

    //初始化

    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH);

    glutInitWindowSize(800, 480);

    //创建窗口

    glutCreateWindow("地球仪");

    initRendering();

    //设置用户操作

    texEarth = load\_texture("earth.bmp");

    glutDisplayFunc(drawScene);

    glutKeyboardFunc(handleKey);

    glutReshapeFunc(handleResize);

    glutMotionFunc(handleMotion);

    //动画效果

    glutTimerFunc(16, update, 0);

    glutMainLoop();

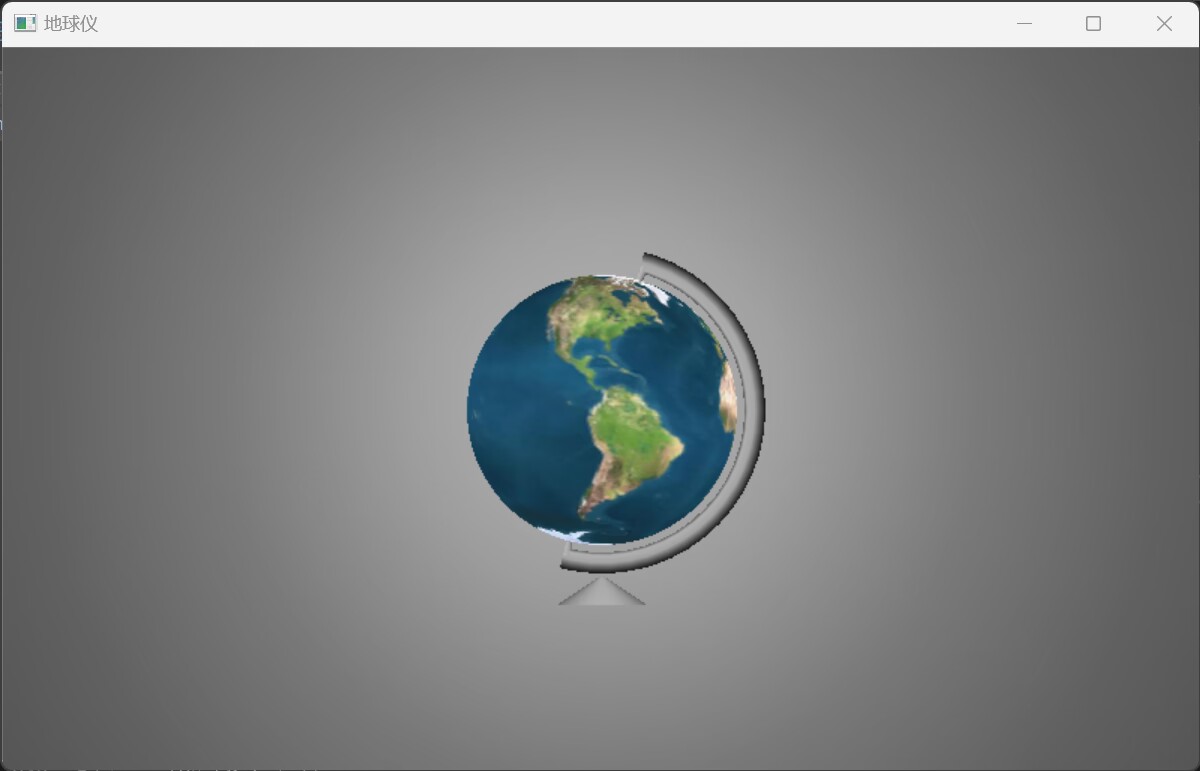
    //退出时删除建模

    gluDeleteQuadric(earth);

    return 0;

}

# 五、运行结果展示



# 六、心得与体会

在本次实验中，我们实现了以下功能：

地球仪绘制： 利用OpenGL库提供的函数，绘制出圆球并成功地将世界地图作为纹理映射到球体表面上，实现了地球仪的基本外观。

光照和材质： 通过使用OpenGL的光照模型和材质属性设置，成功地加入了光影效果。

支架： 绘制了支架和底座，使地球仪更具立体感和真实感。

放大缩小： 实现了通过w和s控制缩放。放大缩小时，支架和底座也会同步放大和缩小，提升了地球仪的真实感。

旋转： 可以通过鼠标拖动旋转地球仪，来观察地球仪不同方向的细节。同时，地球仪还会自动模拟地球的自转。

通过OpenGL地球仪实验，我们对图形学基础、OpenGL库的灵活应用、纹理映射、光照和材质的设置形成了更深刻的认识。这次实验不仅加深了理论理解，也提高了实际编程技能，使我们能够使用OpenGL完成一些简单的图像和功能。