实验六 投影变换

一、实验目的

掌握投影的概念以及两种常见投影方式：正交（正平行）投影和透视投影。掌握两种投影方式的特点和实现方法。

二、实验内容

实验六代码文件预计功能为：定义三维点类Point、立方体类Cube和变换矩阵类Matrix；左右方向键按下时，立方体绕与Y轴平行的中心轴旋转；按下空格键，在两种投影模式间切换。但是文件关键代码缺失，请根据给定文件创建控制台项目，按照提示（“//请补充代码”）增加代码，以完成下列题目要求：

1. 【必做】对立方体模型进行正交投影变换（投影平面为观察坐标系的xoy面），将投影结果绘制在窗口中。完成实验报告。

#include <GL/glut.h>

class Point {

public:

float x, y, z;

Point() : x(0.0f), y(0.0f), z(0.0f) {}

Point(float x, float y, float z) : x(x), y(y), z(z) {}

};

class Cube {

public:

Point vertices[8];

Cube(float size) {

vertices[0] = Point(-size / 2, -size / 2, -size / 2);

vertices[1] = Point(size / 2, -size / 2, -size / 2);

vertices[2] = Point(size / 2, size / 2, -size / 2);

vertices[3] = Point(-size / 2, size / 2, -size / 2);

vertices[4] = Point(-size / 2, -size / 2, size / 2);

vertices[5] = Point(size / 2, -size / 2, size / 2);

vertices[6] = Point(size / 2, size / 2, size / 2);

vertices[7] = Point(-size / 2, size / 2, size / 2);

}

};

Cube myCube(2.0f);

bool isOrthoProjection = true;

void drawCube() {

glColor3f(1.0f, 1.0f, 1.0f);

glBegin(GL\_QUADS);

for (int i = 0; i < 6; ++i) {

glVertex3f(myCube.vertices[i].x, myCube.vertices[i].y, myCube.vertices[i].z);

}

glEnd();

}

void display() {

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (isOrthoProjection) {

glOrtho(-5, 5, -5, 5, -10, 10);

}

else {

gluPerspective(45, 1, 1, 10);

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(0, 0, 5, 0, 0, 0, 0, 1, 0);

drawCube();

glutSwapBuffers();

}

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

if (key == ' ') {

isOrthoProjection = !isOrthoProjection;

glutPostRedisplay();

}

}

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

glutInit(&argc, argv);

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

glutCreateWindow("Cube Projection");

glEnable(GL\_DEPTH\_TEST);

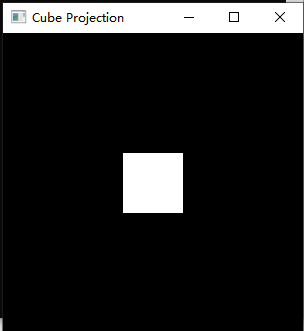
glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}



1. 【必做】修改题目1，对模型进行一点透视投影变换，并把结果显示在窗口中。设投影平面平行于观察坐标系的xoy平面，且与观察坐标系原点距离为d（如d=200，注意是距离）。完成实验报告。

#include <GL/glut.h>

class Point {

public:

float x, y, z;

Point() : x(0.0f), y(0.0f), z(0.0f) {}

Point(float x, float y, float z) : x(x), y(y), z(z) {}

};

class Cube {

public:

Point vertices[8];

Cube(float size) {

vertices[0] = Point(-size / 2, -size / 2, -size / 2);

vertices[1] = Point(size / 2, -size / 2, -size / 2);

vertices[2] = Point(size / 2, size / 2, -size / 2);

vertices[3] = Point(-size / 2, size / 2, -size / 2);

vertices[4] = Point(-size / 2, -size / 2, size / 2);

vertices[5] = Point(size / 2, -size / 2, size / 2);

vertices[6] = Point(size / 2, size / 2, size / 2);

vertices[7] = Point(-size / 2, size / 2, size / 2);

}

};

Cube myCube(2.0f);

bool isOrthoProjection = false; // Set initial projection to perspective

float distanceFromOrigin = 200.0f; // Distance from the origin for perspective projection

void drawCube() {

glColor3f(1.0f, 1.0f, 1.0f);

glBegin(GL\_QUADS);

for (int i = 0; i < 6; ++i) {

glVertex3f(myCube.vertices[i].x, myCube.vertices[i].y, myCube.vertices[i].z);

}

glEnd();

}

void display() {

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

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (isOrthoProjection) {

glOrtho(-5, 5, -5, 5, -10, 10);

}

else {

gluPerspective(45, 1, 1, 1000);

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(0, 0, distanceFromOrigin, 0, 0, 0, 0, 1, 0); // Adjusted for perspective projection

drawCube();

glutSwapBuffers();

}

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

if (key == ' ') {

isOrthoProjection = !isOrthoProjection;

glutPostRedisplay();

}

}

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

glutInit(&argc, argv);

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

glutCreateWindow("Cube Projection");

glEnable(GL\_DEPTH\_TEST);

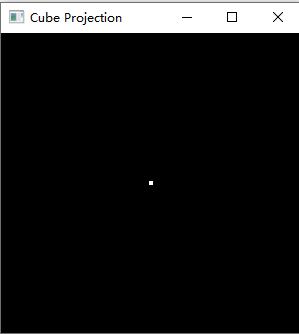
glutDisplayFunc(display);

glutKeyboardFunc(keyboard);

glutMainLoop();

return 0;

}



以上两道题目合并提交一份完整代码即可。

三、实验参考

1. 主视图变换矩阵（透视投影的复合矩阵）

1. 一点透视投影变换矩阵

注意：该矩阵参数的推导，是基于课件中提到的坐标系设定，即投影平面在观察坐标系的Z轴负向。如果采用不同的设定，请自行推导透视投影矩阵并应用。

3.说明：透视投影中，视点位置不能位于所绘制物体的内部，同时要求观察平面位于视点前方，否则无法正确观察。所以在程序中，要注意绘制物体位置、观察平面位置的定义。

4.在进行三维图形投影计算时，物体各顶点的三维局部坐标需要先转换为三维世界坐标，再由世界坐标转换为观察坐标（摄像机坐标），再进行投影变换，最后对二维投影结果进行光栅化。但为了简化实验代码，本实验假定观察坐标系和世界坐标系以及局部坐标系重合。

5.透视投影结果未必和正交投影一样大小。