|  |  |  |
| --- | --- | --- |
| 南农大  **计算机图形学实验报告**  XH2 | | |
|  | 姓 名: | 董自经 学号: 19222126 |
|  | 学 院: | 人工智能学院 |
|  | 专 业: | 计算机科学与技术 |
|  | 班 级: | 计科221 |
|  | 指导教师: | 谢忠红 职称： 副教授 |
| 2024 年 12 月 10 日 | | |

计算机图形学实验报告

计科221 董自经

1. 实验目标

对三维图形进行投影、旋转、消隐等操作，并将之展现在屏幕上，初步掌握对三维图形的操作。

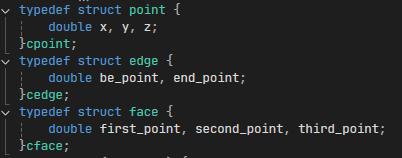
二、实验内容

通过EasyX图形库与Visual Studio 2022结合使用，可以实现对三维图形的投影、旋转和消隐等基本操作。首先，在EasyX中设置窗口和绘图环境，然后利用三维坐标转换算法，对三维物体进行透视投影，将其转换为二维图形。接下来，通过定义旋转矩阵，实现物体的旋转效果。通过这些实践，初步掌握三维图形的操作与展示。

本实验中分三次实现了三个不同的题目：投影、旋转投影、消隐。

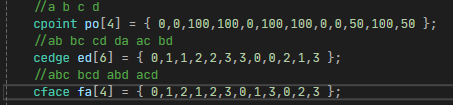
（一）建立三维图形表结构

使用数据表来记录图形元素的数据结构。分别建立顶点表、边表以及面表。



图一：三表结构

之后向其中插入需要进行操作的三棱锥的坐标：



图二：插入数据

（二）投影

此处所进行的投影操作是透视投影，有三角形相似得出投影之后的x、y坐标，分别为：

x’=x\*d/(d-z)、y’=y\*d/(d-z)

由于坐标点较少，在本题中直接使用公式进行了计算。在实际运用中应该使用矩阵进行计算，公式如下：

（x y z 1）\*（a1 a2 a3 a4）=（x’y’z’1）

其中a1 =（1 0 0 0），a2 =（0 1 0 0），a3 =（0 0 0 0），a4 =（0 0 -1/d 1）是透视点为(0,0,d),透视平面O-XY平面的透视投影变换矩阵

（三）旋转投影

绕y轴旋转的公式进集合推导为：

x ´= z\*sinθ + x\*cosθ， y ´= y，z’ = z\*cosθ – x\*sinθ

相应的变换矩阵为：（a1 a2 a3 a4）

其中a1 =（cos(θ) 0 sin(θ) 0），a2 =（0 1 0 0），a3 =（-sin(θ) 0 cos(θ) 0），a4 =（0 0 0 1）

（四）消隐

消隐利用的是向量的矢量内积，当平面的外向法向量与视线向量的乘积N.E<=0 时，θ >=90°，此时的面对于视点而言时不可见，反之面是可见的。

对于平面，可以使用平面的法矢量N = (a,b,c)，以及平面上任一点(x1,y1,z1) 来表示：

ax + by + cz + d = 0，d = -ax1 - by1 - cz1

已知平面内任意三个点的坐标时，可用以下公式求解a、b、c以及d

a=y1(z2-z3)+y2(z3-z2)+y3(z1-z2)

b=z1(x2-x3)+z2(x3-x1)+z3(x1-x2)

c=x1(y2-y3)+x2(y3-y1)+x3(y1-y2)

d=x1(y2z3-z2y3)+y1(z2x3-x2z3)+z1(x2y3-y2x3)

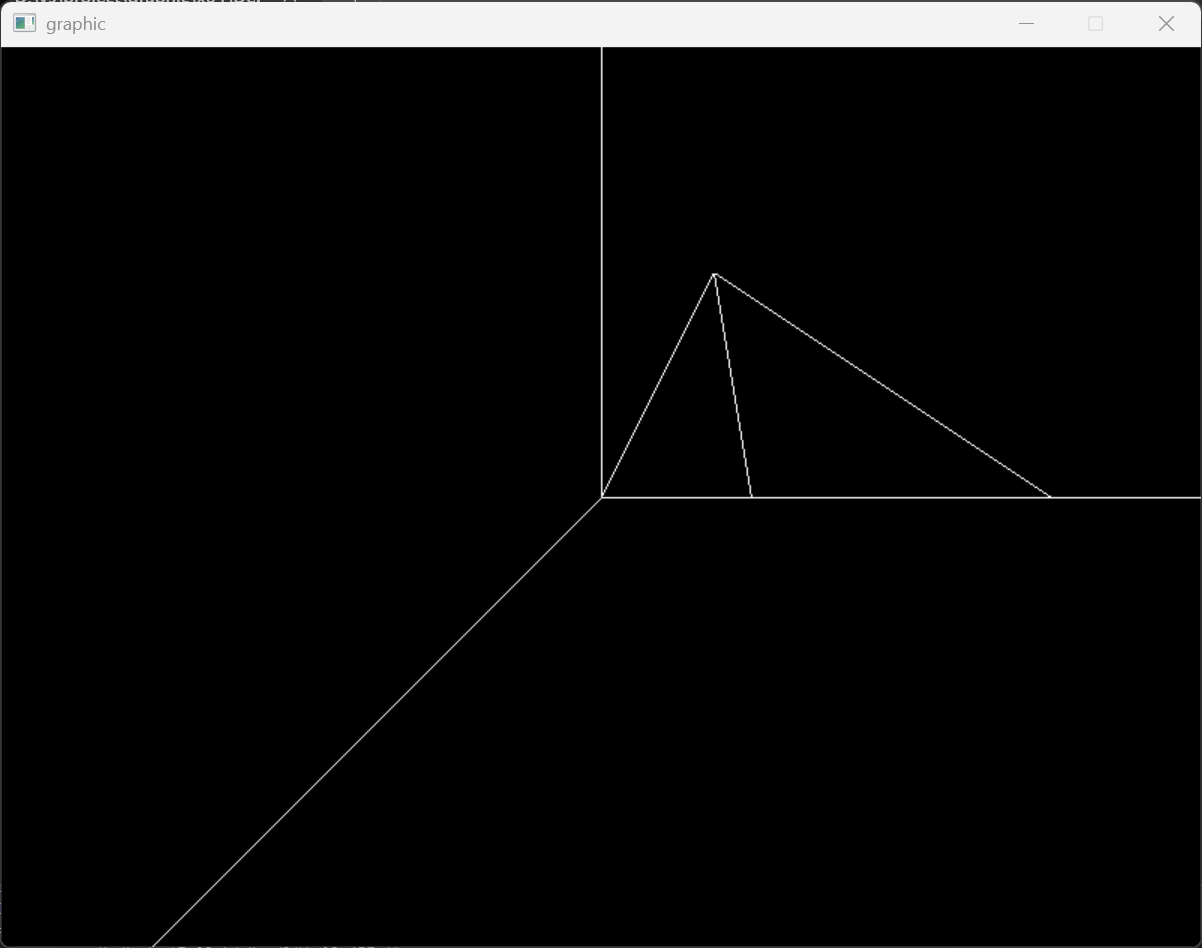
视线方向向量是使用视点坐标减去平面内任意一点坐标得到的，平面内的一点取得是三角形的中心坐标，由此可得视线向量：

（0 -（x1 + x2 + x3）/3 , 0 -（y1 + y2 + y3）/3 , d - （z1 + z2 + z3）/3）

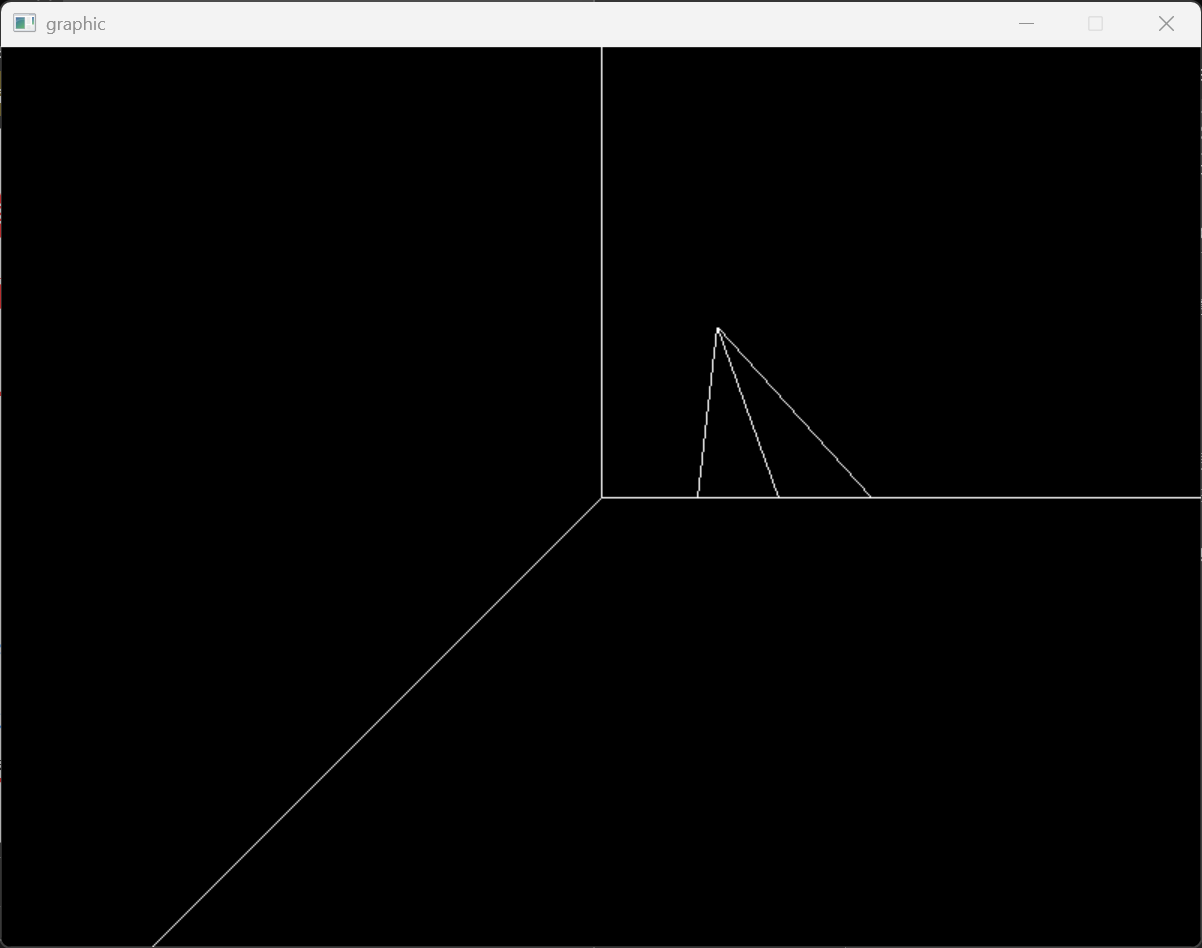
之后计算乘积即可得到面是否可见

1. 实验结果

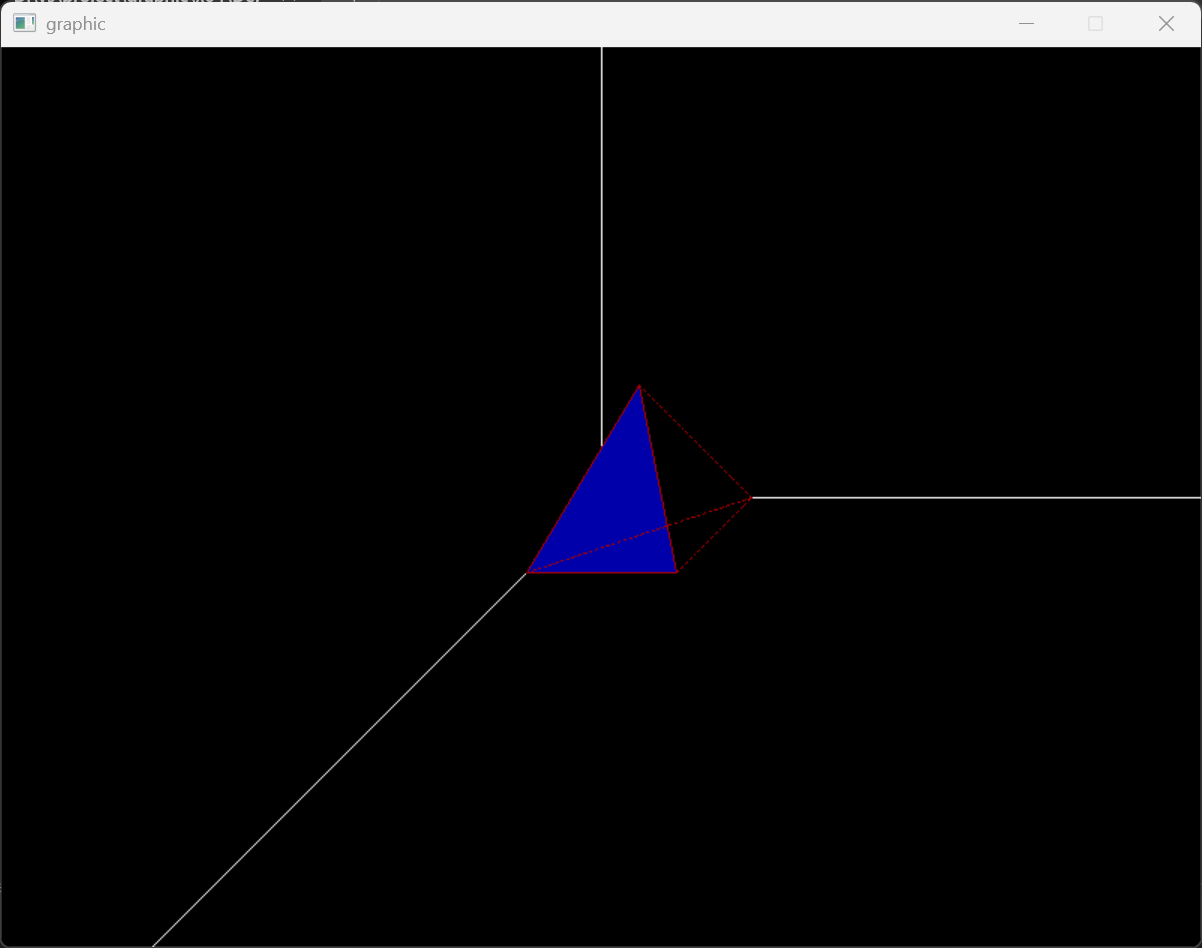
实验是分三次运行不同的代码程序，而可以得到相应的图形：



图三：题（2）投影



图四：题（3）旋转投影



图五：消隐

四、实验不足

本次实验中，主要的问题在于消隐阶段，实际运算出的结果中有两个面是符合显示条件的，但是由于另一个面的遮挡作用，该面实际上是不可视的，在代码中使用了手工屏蔽的方式将该面设置为不可视。

五、实验小结

通过本次实验，利用EasyX图形库与Visual Studio 2022，成功实现了三维图形的投影、旋转和消隐等基本操作。在设置绘图窗口并创建绘图环境后，应用三维坐标转换算法，将三维物体进行透视投影并转换为二维图形，同时定义了旋转矩阵，实现了物体的旋转效果。这一过程加深了我对三维图形变换的理解，特别是在空间坐标系统中的视角变化与物体表面展示的关系。

通过这些实践，我认识到三维图形变换不仅涉及数学理论，还与实际应用密切相关，例如虚拟现实、游戏开发和计算机辅助设计等领域。未来，我希望能进一步探索更高级的渲染技术和交互式应用，例如利用光线追踪技术提升图像真实感，或者结合人工智能技术实现更智能化的图形处理。同时，掌握更多的图形编程语言和框架将为我在图形学领域的深耕提供更广阔的视野和可能性。

六、源代码

#undef UNICODE

#undef \_UNICODE

#include<cmath>

#include<conio.h>

#include<codecvt>

#include<graphics.h>

#include<bits/stdc++.h>

using namespace std;

#define pi 3.1415926

typedef struct point {

double x, y, z;

}cpoint;

typedef struct edge {

double be\_point, end\_point;

}cedge;

typedef struct face {

double first\_point, second\_point, third\_point;

}cface;

double swap(double t) {

return t / 180.0 \* pi;

}

int main() {

initgraph(800, 600, 1);

int px = 400, py = 300;

//坐标轴x y z

line(400, py, 800, py); line(px, 0, px, 300); line(px, py, 100, 600);

//(1)三表结构

//a b c d

cpoint po[4] = { 0,0,100,100,0,100,100,0,0,50,100,50 };

//ab bc cd da ac bd

cedge ed[6] = { 0,1,1,2,2,3,3,0,0,2,1,3 };

//abc bcd abd acd

cface fa[4] = { 0,1,2,1,2,3,0,1,3,0,2,3 };

//(2)投影到O-xy，视点为(0,0,d)

//三角形相似 (x,y,z)-->(x',y',z') x'=x\*d/(d-z) y'=y\*d/(d-z) z'=0

double tmp\_p[4][2];

cpoint p = { 0,0,150 };

double d = p.z;

for (int i = 0; i < 4; i++) {

tmp\_p[i][0] = po[i].x \* d / (d - po[i].z);

tmp\_p[i][1] = po[i].y \* d / (d - po[i].z);

}

for (int i = 0; i < 3; i++)

for (int j = i + 1; j < 4; j++)

line(px + tmp\_p[i][0], py - tmp\_p[i][1], px + tmp\_p[j][0], py - tmp\_p[j][1]);

Sleep(1000); cleardevice();

line(400, py, 800, py); line(px, 0, px, 300); line(px, py, 100, 600);

//(3)绕y轴旋转30度，再次投影，视点为(0,0,d)

//先构造矩阵

double angle = 30; angle = swap(angle);

double rotate[][4] = { cos(angle),0,-sin(angle),0,0,1,0,0,sin(angle),0,cos(angle),0,0,0,0,1 };

double tmp[4][4];

for (int i = 0; i < 4; i++) {

tmp[i][0] = po[i].x;

tmp[i][1] = po[i].y;

tmp[i][2] = po[i].z;

tmp[i][3] = 1;

}

//相乘进行旋转

double t[4][4];

for (int i = 0; i < 4; i++) { //4个点，原始矩阵4行

for (int j = 0; j < 4; j++) { //旋转矩阵四列、目的矩阵四列

t[i][j] = 0;

for (int k = 0; k < 4; k++) //每一行有四个乘积

t[i][j] += tmp[i][k] \* rotate[k][j];

}

cout << endl;

}

//投影

for (int i = 0; i < 4; i++) {

tmp\_p[i][0] = t[i][0] \* d / (d - t[i][2]);

tmp\_p[i][1] = t[i][1] \* d / (d - t[i][2]);

}

for (int i = 0; i < 3; i++)

for (int j = i + 1; j < 4; j++)

line(px + tmp\_p[i][0], py - tmp\_p[i][1], px + tmp\_p[j][0], py - tmp\_p[j][1]);

Sleep(1000); cleardevice();

line(400, py, 800, py); line(px, 0, px, 300); line(px, py, 100, 600);

//(4)消隐 视点为(0,0,d)

//平面表示：ax+by+cz+d=0

//法向量N(a,b,c) d=-ax1-by1-cz1

//已知三点：P1(x1,y1,z1) P2(x2,y2,z2) P3(x3,y3,z3)

//a = y1(z2 - z3) + y2(z3 - z2) + y3(z1 - z2)

//b = z1(x2 - x3) + z2(x3 - x1) + z3(x1 - x2)

//c = x1(y2 - y3) + x2(y3 - y1) + x3(y1 - y2)

//d = x1(y2z3 - z2y3) + y1(z2x3 - x2z3) + z1(x2y3 - y2x3)

//a.b=|a|.|b|cosθ = axbx + ayby + azbz 大于0可见，小于0不可见

//构建每个面的法向量

// 构建检验向量,取每个三角形的重心

double N[4][3]; double check[4][3];

//abc abd acd bcd

int cnt = 0;

double x1, y1, z1, x2, y2, z2, x3, y3, z3;

for (int i = 0; i < 2; i++) {

x1 = po[i].x; y1 = po[i].y; z1 = po[i].z;

for (int j = i + 1; j < 3; j++) {

x2 = po[j].x; y2 = po[j].y; z2 = po[j].z;

for (int k = j + 1; k < 4; k++) {

x3 = po[k].x; y3 = po[k].y; z3 = po[k].z;

N[cnt][0] = y1 \* (z2 - z3) + y2 \* (z3 - z2) + y3 \* (z1 - z2);

N[cnt][1] = z1 \* (x2 - x3) + z2 \* (x3 - x1) + z3 \* (x1 - x2);

N[cnt][2] = x1 \* (y2 - y3) + x2 \* (y3 - y1) + x3 \* (y1 - y2);

check[cnt][0] = 0 - (x1 + x2 + x3) \* 1.0 / 3;

check[cnt][1] = 0 - (y1 + y2 + y3) \* 1.0 / 3;

check[cnt][2] = d - (z1 + z2 + z3) \* 1.0 / 3;

cnt++;

}

}

}

//检验可视,并对边打标签

//在图像中，x,y不变，z为原来一半

//abc abd acd bcd

for (int i = 0; i < 4; i++) {

if (po[i].z != 0) {

po[i].x -= po[i].z / 2.0;

po[i].y -= po[i].z / 2.0;

}

}

int flag[4]; double sum = 0;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 3; j++)

sum += N[i][j] \* check[i][j];//sum大于0表示可视

POINT pts[3] = { {px+po[int(fa[i].first\_point)].x,py-po[int(fa[i].first\_point)].y},

{ px+po[int(fa[i].second\_point)].x,py-po[int(fa[i].second\_point)].y},

{ px+po[int(fa[i].third\_point)].x,py-po[int(fa[i].third\_point)].y} };

if (sum > 0 && i != 3) {

if(i==0)

setfillcolor(RED);

else if(i==1)

setfillcolor(YELLOW);

else if(i==2)

setfillcolor(BLUE);

else

setfillcolor(GREEN);

solidpolygon(pts, 3);

flag[i] = 1;

}

else {

flag[i] = 0;

}

}

//ab bc cd da ac bd

int flag1[6] = {1,0,0,1,0,1};

//对z处理

setlinecolor(RED);

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

if (flag1[i] == 1) {

setlinestyle(PS\_SOLID);

line(px + po[int(ed[i].be\_point)].x, py - po[int(ed[i].be\_point)].y, px + po[int(ed[i].end\_point)].x, py - po[int(ed[i].end\_point)].y);

}

else {

setlinestyle(PS\_DASH);

line(px + po[int(ed[i].be\_point)].x, py - po[int(ed[i].be\_point)].y, px + po[int(ed[i].end\_point)].x, py - po[int(ed[i].end\_point)].y);

}

}

//Sleep(1000); cleardevice();

\_getch();

closegraph();

return 0;

}