**案例27-正十二面体透视投影算法**

1. **主要知识点**

透视投影视所有投影线都从视点投射，离视点近的物体投影大，离视点远的物体投影小，小到极点消失，成为灭点。

视点：观察者的眼睛的位置，也是观察坐标系的原点。

视心：垂直于屏幕的实现与屏幕的交点，也是屏幕坐标系的原点。

视距：视点到视心的距离。

视径：视点到物体的距离。

透视投影算法分为两部分：从世界坐标系到观察坐标系和从观察坐标系到屏幕坐标系。

世界坐标系变换为观察坐标系，齐次坐标矩阵表示为：，其展开式为：

其中，则有

观察坐标系到屏幕坐标系通过直角三角形的相似关系可以得出下式成立：

（w下标表示世界坐标系，v下标表示观察坐标系，s下标表示屏幕坐标系）

1. **案例描述：**

本案例利用正十二面体透视投影算法，绘制正十二面体。

1. **实现步骤：**
2. 添加基础类与添加绘制立方体的CDodecahedron类。
3. 在CDodecahedron类中计算顶点坐标、读入面表，绘制图形，透视变换参数初始化、设置视点位置以及透视变换。
4. 在CTestView中添加消息响应函数，在OnDraw中调用DoubleBuffer函数。
5. **主要算法**

1. CDodecahedron类：

public:

void ReadPoint();//点表

void ReadFace();//面表

void InitParameter();//参数初始化

void SetViewPoint();//设置视点

CP3d PersPectiveProject(CP3d P);//透视投影

void Draw(CDC\* pDC);//绘制正四面体线框

public:

CP3d P[20];//点表

CFace F[12];//面表

int nClientWidth;//屏幕客户区宽度

int nClientHeight;//屏幕客户区高度

int nHWidth, nHHeight;//屏幕客户区的半宽和半高

double R, Theta, Phi, d;//R,Theta,Phi视点在用户坐标系的球坐标,d视距

double k[9];//运算常量

CTransform tran;

CP3d ViewPoint;//视点

void CDodecahedron::ReadPoint()//点表

{

//顶点的三维坐标(x,y,z)

const double Golden\_Section = (sqrt(5.0) - 1.0) / 2.0;//黄金分割比例

double a = 160;//黄金矩形长半边的边长

double b = a \* Golden\_Section;//黄金矩形短半边的边长

P[0].x = a; P[0].y = a; P[0].z = a;

P[1].x = a + b; P[1].y = 0; P[1].z = b;

P[2].x = a; P[2].y = -a; P[2].z = a;

P[3].x = 0; P[3].y = -b; P[3].z = a + b;

P[4].x = 0; P[4].y = b; P[4].z = a + b;

P[5].x = a + b; P[5].y = 0; P[5].z = -b;

P[6].x = a; P[6].y = a; P[6].z = -a;

P[7].x = b; P[7].y = a + b; P[7].z = 0;

P[8].x = -b; P[8].y = a + b; P[8].z = 0;

P[9].x = -a; P[9].y = a; P[9].z = -a;

P[10].x = 0; P[10].y = b; P[10].z = -a - b;

P[11].x = a; P[11].y = -a; P[11].z = -a;

P[12].x = b; P[12].y = -a - b; P[12].z = 0;

P[13].x = -b; P[13].y = -a - b; P[13].z = 0;

P[14].x = -a - b; P[14].y = 0; P[14].z = b;

P[15].x = -a; P[15].y = a; P[15].z = a;

P[16].x = -a; P[16].y = -a; P[16].z = -a;

P[17].x = 0; P[17].y = -b; P[17].z = -a - b;

P[18].x = -a; P[18].y = -a; P[18].z = a;

P[19].x = -a - b; P[19].y = 0; P[19].z = -b;

}

void CDodecahedron::ReadFace()//面表

{

//面的顶点数和面的顶点索引

F[0].SetNum(5); F[0].vI[0] = 0; F[0].vI[1] = 7; F[0].vI[2] = 8; F[0].vI[3] = 15; F[0].vI[4] = 4;

F[1].SetNum(5); F[1].vI[0] = 6; F[1].vI[1] = 10; F[1].vI[2] = 9; F[1].vI[3] = 8; F[1].vI[4] = 7;

F[2].SetNum(5); F[2].vI[0] = 1; F[2].vI[1] = 5; F[2].vI[2] = 6; F[2].vI[3] = 7; F[2].vI[4] = 0;

F[3].SetNum(5); F[3].vI[0] = 1; F[3].vI[1] = 2; F[3].vI[2] = 12; F[3].vI[3] = 11; F[3].vI[4] = 5;

F[4].SetNum(5); F[4].vI[0] = 11; F[4].vI[1] = 12; F[4].vI[2] = 13; F[4].vI[3] = 16; F[4].vI[4] = 17;

F[5].SetNum(5); F[5].vI[0] = 2; F[5].vI[1] = 3; F[5].vI[2] = 18; F[5].vI[3] = 13; F[5].vI[4] = 12;

F[6].SetNum(5); F[6].vI[0] = 0; F[6].vI[1] = 4; F[6].vI[2] = 3; F[6].vI[3] = 2; F[6].vI[4] = 1;

F[7].SetNum(5); F[7].vI[0] = 3; F[7].vI[1] = 4; F[7].vI[2] = 15; F[7].vI[3] = 14; F[7].vI[4] = 18;

F[8].SetNum(5); F[8].vI[0] = 5; F[8].vI[1] = 11; F[8].vI[2] = 17; F[8].vI[3] = 10; F[8].vI[4] = 6;

F[9].SetNum(5); F[9].vI[0] = 9; F[9].vI[1] = 10; F[9].vI[2] = 17; F[9].vI[3] = 16; F[9].vI[4] = 19;

F[10].SetNum(5); F[10].vI[0] = 8; F[10].vI[1] = 9; F[10].vI[2] = 19; F[10].vI[3] = 14; F[10].vI[4] = 15;

F[11].SetNum(5); F[11].vI[0] = 13; F[11].vI[1] = 18; F[11].vI[2] = 14; F[11].vI[3] = 19; F[11].vI[4] = 16;

}

void CTetrahedron::InitParameter()//透视变换参数初始化

{

k[1] = sin(PI\*Theta / 180);

k[2] = sin(PI \* Phi / 180);

k[3] = cos(PI \* Theta / 180);

k[4] = cos(PI \* Phi / 180);

k[5] = k[2] \* k[3];

k[6] = k[2] \* k[1];

k[7] = k[4] \* k[3];

k[8] = k[4] \* k[1];

}

void CTetrahedron::SetViewPoint()//设置视点

{

ViewPoint.x = R \* k[6];

ViewPoint.y = R \* k[5];

ViewPoint.z = R \* k[4];

}

CP3d CTetrahedron::PersPectiveProject(CP3d P)//透视变换

{

CP3d ViewCoorP;//观察坐标系内的点

CP3d ScreenCoorP;//屏幕坐标系内的点

ViewCoorP.x = k[3] \* P.x - k[1] \* P.z;//观察坐标系三维坐标

ViewCoorP.y = -k[8] \* P.x + k[2] \* P.y - k[7] \* P.z;

ViewCoorP.z = -k[6] \* P.x - k[4] \* P.y - k[5] \* P.z + R;

ViewCoorP.c = P.c;

ScreenCoorP.x = d \* ViewCoorP.x / ViewCoorP.z;//屏幕坐标系二维坐标

ScreenCoorP.y = d \* ViewCoorP.y / ViewCoorP.z;

ScreenCoorP.c = ViewCoorP.c;

return ScreenCoorP;

}

void CTetrahedron::Draw(CDC\* pDC)//绘制正四面体线框模型

{

CP3d Vertex[3];

CLine\* line = new CLine;

for (int nFace = 0; nFace < 8; nFace++)//面循环

{

for (int nVertex = 0; nVertex < F[nFace].vN; nVertex++)//顶点循环

Vertex[nVertex] = PersPectiveProject(P[F[nFace].vI[nVertex]]);

CVector3 ViewVector(Vertex[0], ViewPoint);//面的视矢量

ViewVector.Normalize();//单位化视矢量

CVector3 V01(Vertex[0], Vertex[1]);//面对一条矢量边

CVector3 V02(Vertex[0], Vertex[2]);//面对另一条矢量边

CVector3 FNormal = Cross(V01, V02);//面的法矢量

FNormal.Normalize();//单位化法矢量

if (Dot(ViewVector, FNormal) >= 0)//背面剔除

{

line->MoveTo(pDC, ROUND(nHWidth + Vertex[0].x), ROUND(nHHeight - Vertex[0].y));

line->LineTo(pDC, ROUND(nHWidth + Vertex[1].x), ROUND(nHHeight - Vertex[1].y));

line->LineTo(pDC, ROUND(nHWidth + Vertex[2].x), ROUND(nHHeight - Vertex[2].y));

line->LineTo(pDC, ROUND(nHWidth + Vertex[0].x), ROUND(nHHeight - Vertex[0].y));

}

}

delete line;

}

2.CTestView类：

public:

void DoubleBuffer(CDC\* pDC);//双缓存绘图

void DrawObject(CDC\* pDC);//绘图

protected:

BOOL bPlay;//动画开关

CDodecahedron dodecahedron;

void CTestView::DoubleBuffer(CDC \* pDC)

{

CRect rect;//定义客户区

GetClientRect(&rect);//获得客户区的大小

dodecahedron.nClientWidth = rect.Width();//屏幕客户区宽度

dodecahedron.nClientHeight = rect.Height();//屏幕客户区高度

dodecahedron.nHWidth = dodecahedron.nClientWidth / 2;//屏幕客户区半宽

dodecahedron.nHHeight = dodecahedron.nClientHeight / 2;//屏幕客户区半高

CDC memDC;

memDC.CreateCompatibleDC(pDC);

CBitmap NewBitmap, \*pOldBitmap;

NewBitmap.CreateCompatibleBitmap(pDC, dodecahedron.nClientWidth, dodecahedron.nClientHeight);

pOldBitmap = memDC.SelectObject(&NewBitmap);

memDC.FillSolidRect(rect, pDC->GetBkColor());

DrawObject(&memDC);

pDC->BitBlt(0, 0, dodecahedron.nClientWidth, dodecahedron.nClientHeight, &memDC, 0, 0, SRCCOPY);

memDC.SelectObject(pOldBitmap);

NewBitmap.DeleteObject();

}

void CTestView::DrawObject(CDC \* pDC)

{

dodecahedron.Draw(pDC);

}

1. **实现效果：**

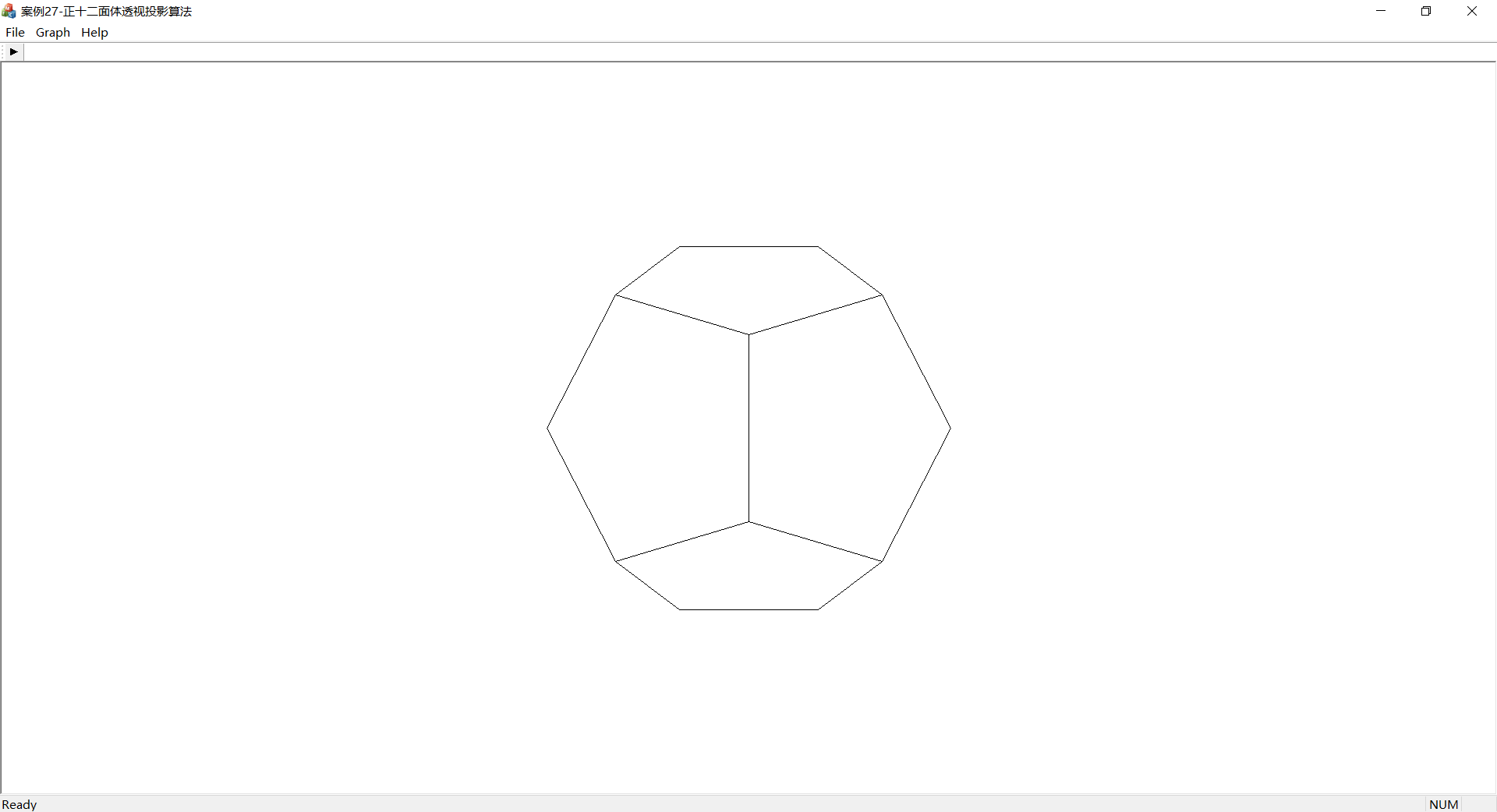


图1正十二面体透视投影算法效果图