**案例21-三维图形几何变换算法**

1. **主要知识点**

三维图形几何变换算法原理：三维变换算法与二维变换类似。三维图形几何变换矩阵是一个4×4的矩阵：

其中为3×3阶子矩阵，对物体进行比例、旋转、反射和错切变换：

为1×3阶子矩阵，对物体进行平移变换:

为3×1阶子矩阵，对物体进行投影变换:

为1×1阶子矩阵，对物体进行整体比例变换:

1. **案例描述：**

本案例利用递归函数绘制递归球，并通过几何变换算法，实现递归球的平移变换、比例变换、旋转变换。

1. **实现步骤：**
2. 添加基础类：CRGB、CP2i、CP2d、CP3d、Cline、CFace.
3. 三维变换核心类：CTransform3类，包含了平移变换、比例变化、旋转变换、反射变换、错切变换矩阵。
4. 在CTestView中计算顶点坐标、读入面表，绘制图形，进行递归和边界检测。
5. 在CTestView中添加消息响应函数，在OnDraw中调用SetTimer函数。
6. 在ResourceView中新建对话框资源，静态切分视图框架，并生成继承与CFormView类的CLeftPortion类，在该类中写实现变换的相应函数。
7. **主要算法**

CTestView类：

public:

void ReadPoint();//计算顶点坐标

void ReadFace();//读入面表

void DoubleBuffer();//双缓冲

void DrawObject(CDC\* pDC);//绘制图形

void SubDivide(CDC\* pDC, CP3d p0, CP3d p1, CP3d p2, int n);//递归函数

void Normalize(CP3d &);//矢量扩展函数

void DrawTriangle(CDC\* pDC, CP3d p0, CP3d p1, CP3d p2);//绘制三角形函数

void BorderCheck();//边界检测

protected:

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

CP3d P[7];//点表

CFace F[8];//面表

int directionX,directionY,directionZ;//位移方向控制

double translateX,translateY,translateZ;//平移变换矩阵参数

double scale;//比例变换参数

double rotateX,rotateY,rotateZ;//旋转变换矩阵参数

int facecount;//递归深度

double radius;//球体半径

CTransform3 tran;//变换对象

void CTestView::DoubleBuffer()//双缓冲

{

CDC\* pDC = GetDC();

CRect rect;//定义客户区

GetClientRect(&rect);//获得客户区的大小

nClientWidth = rect.Width();//屏幕客户区宽度

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

nHWidth = nClientWidth / 2;//屏幕客户区半宽

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

CDC memDC;

memDC.CreateCompatibleDC(pDC);

CBitmap NewBitmap, \* pOldBitmap;

NewBitmap.CreateCompatibleBitmap(pDC, nClientWidth, nClientHeight);

pOldBitmap = memDC.SelectObject(&NewBitmap);

ReadPoint();//读入点表

ReadFace(); //读入面表

//平移变换

tran.Translate(translateX, translateY, translateZ);

//相对于任意点的绕X轴旋转变换

tran.RotateX(rotateX, CP3d(translateX, translateY, translateZ));

//相对于任意点的绕Y轴旋转变换

tran.RotateY(rotateY, CP3d(translateX, translateY, translateZ));

//相对于任意点的绕Z轴旋转变换

tran.RotateZ(rotateZ, CP3d(translateX, translateY, translateZ));

//相对于任意点的比例变换

tran.Scale(scale, scale, scale, CP3d(translateX, translateY, translateZ));

DrawObject(&memDC);

BorderCheck();

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

memDC.SelectObject(pOldBitmap);

NewBitmap.DeleteObject();

}

void CTestView::ReadPoint()//读入顶点坐标

{

P[0].x = 0; P[0].y = radius; P[0].z = 0; P[0].w = 1;

P[1].x = 0; P[1].y =-radius; P[1].z = 0; P[1].w = 1;

P[2].x = radius; P[2].y = 0; P[2].z = 0; P[2].w = 1;

P[3].x = 0; P[3].y = 0; P[3].z =-radius; P[3].w = 1;

P[4].x =-radius; P[4].y = 0; P[4].z = 0; P[4].w = 1;

P[5].x = 0; P[5].y = 0; P[5].z = radius; P[5].w = 1;

P[6].x = 0; P[6].y = 0; P[6].z = 0; P[6].w = 1;//球体中心

tran.SetMat(P,7);

}

void CTestView::ReadFace()//面表

{

//第一列为每个面的边数,其余列为面的顶点编号

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

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

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

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

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

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

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

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

}

void CTestView::DrawObject(CDC\* pDC)//绘制图形

{

CP3d Point[3],t;

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

{

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

Point[nVertex] = P[F[nFace].vI[nVertex]];

SubDivide(pDC, Point[0], Point[1], Point[2], facecount);

}

}

void CTestView::SubDivide(CDC\* pDC,CP3d p0, CP3d p1, CP3d p2, int n)//递归函数

{

if (0 == n)

{

DrawTriangle(pDC, p0, p1, p2);

return;

}

else

{

CP3d p01, p12, p20;

p01 = (p0 + p1) / 2;

p12 = (p1 + p2) / 2;

p20 = (p2 + p0) / 2;

Normalize(p01);//扩展模长

Normalize(p12);

Normalize(p20);

SubDivide(pDC, p0, p01, p20, n-1);//递归调用

SubDivide(pDC, p1, p12, p01, n-1);

SubDivide(pDC, p2, p20, p12, n-1);

SubDivide(pDC, p01, p12, p20, n-1);

}

}

void CTestView::Normalize(CP3d &p)//模长标准化函数

{

p -= P[6];

double mag = p.Mag();//模长

if (0 == mag)

return;

p /= mag;

p \*= radius \* scale;

p += P[6];

}

void CTestView::DrawTriangle(CDC\* pDC, CP3d p0, CP3d p1, CP3d p2)//绘制三角形函数

{

CP3d p[3];

p[0] = p0;p[1] = p1;p[2] = p2;

CP3d ViewPoint(P[6].x, P[6].y, P[6].z + 1000);//计算视矢量

CVector3 ViewVector(p0, ViewPoint);//面的视矢量

ViewVector.Normalize();//单位化视矢量

CVector3 V01(p0, p1);//面的一条边矢量

CVector3 V02(p0, p2);//面的另一条边矢量

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

FNormal.Normalize();//单位化法矢量

if (Dot(ViewVector, FNormal) >= 0)

{

CLine\* line = new CLine;

line->MoveTo(pDC, ROUND(nHWidth + p0.x), ROUND(nHHeight- p0.y),

CRGB(1.0, 0.0, 1.0));

line->LineTo(pDC, ROUND(nHWidth + p1.x), ROUND(nHHeight- p1.y),

CRGB(1.0, 0.0, 1.0));

line->LineTo(pDC, ROUND(nHWidth + p2.x), ROUND(nHHeight- p2.y),

CRGB(1.0, 0.0, 1.0));

line->LineTo(pDC, ROUND(nHWidth + p0.x), ROUND(nHHeight- p0.y),

CRGB(1.0, 0.0, 1.0));

delete line;

}

}

void CTestView::OnTimer(UINT nIDEvent)//定时器函数

{

// TODO: Add your message handler code here and/or call default

CTestDoc\* pDoc = GetDocument();

if (((CMainFrame\*)AfxGetMainWnd())->IsPlay)

{

facecount = pDoc->m\_FaceCount;

translateX += pDoc->m\_TranslateX \* directionX;

translateY += pDoc->m\_TranslateY \* directionY;

translateZ += pDoc->m\_TranslateZ \* directionZ;

scale = pDoc->m\_Scale;

rotateX += pDoc->m\_RotateX;

rotateY += pDoc->m\_RotateY;

rotateZ += pDoc->m\_RotateZ;

DoubleBuffer();

}

CView::OnTimer(nIDEvent);

}

void CTestView::BorderCheck()//边界检测

{

double TempR = radius \* scale;

if (fabs(P[6].x) + TempR > nHWidth)

{

directionX \*= -1;

translateX += fabs(fabs(P[6].x) + TempR - nHWidth) \* directionX;

//判断球体水平越界

}

if (fabs(P[6].y) + TempR > nHHeight)

{

directionY \*= -1;

translateY += fabs(fabs(P[6].y) + TempR - nHHeight) \* directionY;

//判断球体垂直越界

}

}

BOOL CTestView::OnEraseBkgnd(CDC\* pDC)//禁止背景刷新

{

// TODO: Add your message handler code here and/or call default

return TRUE;

}

1. **实现效果：**



图1三维图形几何变换算法效果图

1. **知识点补充**

1.静态切分

“静态切分”是指文档窗口在第一次创建时，窗格的次序和数目就已经切分好了，不能再改变，但是可以缩放窗格大小。静态切分视图框架的创建分为以下几个步骤：

1. 再Resource View中新建Dialog对话框资源，并将对话框属性中的Style设置为Child，Border设置为None。
2. 为对话框添加4个GroupBox控件，8个SliderControl控件，9个StaticText控件。
3. 双击对话框，创建继承于CFormView类的CLeftPortion类。

2.Vector类

vector 是将给定类型的元素组织到线性序列中的容器。 它使用户可以快速随机访问任何元素，并动态添加到序列和动态从序列中删除。 vector 是随机访问性能超出限制时的首选序列容器。