**案例23-正三棱柱三视图线框模型算法**

1. **主要知识点**

将视线规定为平行投影线，正对着物体看过去，将可见物体的边界用正投影绘制出来的图形称为视图。一个物体有六个视图。三视图就是主视图、俯视图、侧视图的总称。正面标记为，水平面标记为，侧面标记为。

正三棱柱的三视图归根结底就是正三棱柱线框模型通过矩阵变换来实现的，相应的变换矩阵如下：

(1)主视图投影变换矩阵为：

(2)俯视图投影变换矩阵：

(3)侧视图投影变换矩阵：

(4)斜等测图投影变换矩阵：

1. **案例描述：**

本案例将屏幕静态切分为四个窗格，左上窗格绘制正三棱柱的主视图，左下窗格绘制正三棱柱的俯视图，右上窗格绘制正三棱柱的侧视图，右下窗格绘制正三棱柱的斜等测图；利用正三棱柱三视图线框模型算法，绘制正三棱柱线框模型的三视图。

1. **实现步骤：**
2. 添加基础类：CRGB、CP2i、CP2d、CP3d、Cline、CFace、CTransform3。
3. 添加继承与CView类的CHView、CVView和CWView类.
4. 添加绘制正三棱柱的CTriView类。
5. 在CTriView中计算顶点坐标、读入面表，绘制图形，绘制主视图、俯视图、侧视图、斜等测图。
6. 在CTestView中添加消息响应函数，在OnDraw中调用DoubleBuffer函数。
7. **主要算法**

1.CTriView类：

public:

void ReadPoint();//读入点表

void ReadFace();//读入面表

void RotateY();//绕Y轴旋转变换

void TVMatrix();//读入主视图变换矩阵

void THMatrix();//读入俯视图变换矩阵

void TWMatrix();//读入侧视图变换矩阵

void TOMatrix();//读入斜等测变换矩阵

void MultiMatrix(double T[][4]);//矩阵相乘

void ClearMatrix(double M[4][4]);//矩阵清零

void DrawOView(CDC \*pDC);//绘制斜等侧图

void DrawVView(CDC \*pDC);//绘制主视图

void DrawHView(CDC \*pDC);//绘制俯视图

void DrawWView(CDC \*pDC);//绘制侧视图

void DrawOblique(CDC \*pDC, CP3d \*P);//绘制斜等测图线框模型

void DrawTriView(CDC \*pDC, CP3d \*P);//绘制三视图线框模型

public:

int nClientWidth, nClientHeight;//屏幕客户区宽度和高度

int nHWidth, nHHeight;//屏幕客户区半宽和半高

CP3d P[6];//三棱柱顶点

CP3d V[6];//三视图顶点

CFace F[5];//面表

double TO[4][4];//斜等测变换矩阵

double TV[4][4];//主视图变换矩阵

double TH[4][4];//俯视图变换矩阵

double TW[4][4];//侧视图变换矩阵

void CTriView::ReadPoint()//正三棱柱点表

{

int a = 80, b = 140;//a-底面正三角形边长,b-侧棱长

P[0].x = -a / 2; P[0].y = 0; P[0].z = -b / 2;//P0

P[1].x = 0; P[1].y = sqrt(3) / 2 \* a; P[1].z = -b / 2;//P1

P[2].x = a / 2; P[2].y = 0; P[2].z = -b / 2;//P2

P[3].x = -a / 2; P[3].y = 0; P[3].z = b / 2; //P3

P[4].x = 0; P[4].y = sqrt(3) / 2 \* a; P[4].z = b / 2; //P4

P[5].x = a / 2; P[5].y = 0; P[5].z = b / 2; //P5

}

void CTriView::ReadFace()//正三棱柱面表

{

F[0].SetVNum(4); F[0].vIndex[0] = 0; F[0].vIndex[1] = 3;

F[0].vIndex[2] = 4; F[0].vIndex[3] = 1;

F[1].SetVNum(3); F[1].vIndex[0] = 0; F[1].vIndex[1] = 1;

F[1].vIndex[2] = 2;

F[2].SetVNum(4); F[2].vIndex[0] = 0; F[2].vIndex[1] = 2;

F[2].vIndex[2] = 5; F[2].vIndex[3] = 3;

F[3].SetVNum(4); F[3].vIndex[0] = 1; F[3].vIndex[1] = 4;

F[3].vIndex[2] = 5; F[3].vIndex[3] = 2;

F[4].SetVNum(3); F[4].vIndex[0] = 3; F[4].vIndex[1] = 5;

F[4].vIndex[2] = 4;

}

void CTriView::RotateY()//绕y轴旋转变换

{

CTransform3 trans;

trans.SetMat(P, 6);

trans.RotateY(1);

}

void CTriView::TVMatrix()//主视图变换矩阵

{

ClearMatrix(TV);

TV[1][1] = 1;

TV[2][2] = 1;

TV[3][3] = 1;

}

void CTriView::THMatrix()//俯视图变换矩阵

{

ClearMatrix(TH);

TH[0][1] = -1;

TH[2][2] = 1;

TH[3][3] = 1;

}

void CTriView::TWMatrix()//侧视图变换矩阵

{

ClearMatrix(TW);

TW[0][2] = -1;

TW[1][1] = 1;

TW[3][3] = 1;

}

void CTriView::TOMatrix()//斜等测变换矩阵

{

ClearMatrix(TO);

TO[0][0] = 1;

TO[1][1] = 1;

TO[2][0] = -1 / sqrt(2);

TO[2][1] = -1 / sqrt(2);

TO[3][3] = 1;

}

void CTriView::MultiMatrix(double M[][4])//两个矩阵相乘

{

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

{

V[i].x = P[i].x \* M[0][0] + P[i].y \* M[1][0] + P[i].z \* M[2][0] + P[i].w \* M[3][0];

V[i].y = P[i].x \* M[0][1] + P[i].y \* M[1][1] + P[i].z \* M[2][1] + P[i].w \* M[3][1];

V[i].z = P[i].x \* M[0][2] + P[i].y \* M[1][2] + P[i].z \* M[2][2] + P[i].w \* M[3][2];

}

}

void CTriView::ClearMatrix(double M[4][4])//矩阵清零

{

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

for (int j = 0; j < 4; j++)

M[i][j] = 0;

}

void CTriView::DrawOView(CDC \*pDC)//绘制斜等测图

{

pDC->TextOut(20, 30, \_T("立体图"));

MultiMatrix(TO);

DrawOblique(pDC, V);

}

void CTriView::DrawVView(CDC \*pDC)//绘制主视图

{

pDC->TextOut(20, 30, \_T("主视图"));

MultiMatrix(TV);

DrawTriView(pDC, V);

}

void CTriView::DrawHView(CDC \*pDC)//绘制俯视图

{

pDC->TextOut(20, 30, \_T("俯视图"));

MultiMatrix(TH);

DrawTriView(pDC, V);

}

void CTriView::DrawWView(CDC \*pDC)

{

pDC->TextOut(20, 30, \_T("侧视图"));

MultiMatrix(TW);

DrawTriView(pDC, V);

}

void CTriView::DrawOblique(CDC \*pDC, CP3d \*P)//绘制斜等测图线框模型

{

CLine\* line = new CLine;

CP2d Point, t;

for (int nFace = 0; nFace < 5; nFace++)

{

CP3d ViewPoint(sqrt(2) / 2.0, sqrt(2) / 2.0, 1.0);

CVector3 ViewVector(P[F[nFace].vIndex[0]], ViewPoint);//面的视矢量

ViewVector.Normalize();//单位化视向量

CVector3 V01(P[F[nFace].vIndex[0]], P[F[nFace].vIndex[1]]); CVector3 V02(P[F[nFace].vIndex[0]], P[F[nFace].vIndex[2]]); 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

{

for (int nVertex = 0; nVertex < F[nFace].vNum; nVertex++)

//顶点循环

{

Point = P[F[nFace].vIndex[nVertex]];//斜等测投影

if (0 == nVertex)

{

line->MoveTo(pDC, ROUND(nHWidth + Point.x),

ROUND(nHHeight - Point.y), CRGB(0, 0, 1));

t = Point;

}

else

line->LineTo(pDC, ROUND(nHWidth + Point.x),

ROUND(nHHeight - Point.y), CRGB(0, 0, 1), 3);

}

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

CRGB(0, 0, 1), 3);//闭合多边形

}

}

delete line;

}

void CTriView::DrawTriView(CDC\* pDC, CP3d \*P)//绘制三视图线框模型

{

CLine\* line = new CLine;

CP3d Point, t;

for (int nFace = 0; nFace < 5; nFace++)

{

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

{

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

if (0 == nVertex)

{

line->MoveTo(pDC, ROUND(nHWidth + Point.z),

ROUND(nHHeight - Point.y), CRGB(0, 1, 0));

t = Point;

}

else

line->LineTo(pDC, ROUND(nHWidth + Point.z),

ROUND(nHHeight - Point.y), CRGB(0, 1, 0), 3);

}

line->LineTo(pDC, ROUND(nHWidth + t.z),

ROUND(nHHeight - t.y), CRGB(0, 1, 0), 3);//闭合多边形

}

delete line;

}

2.CTestView类：

public:

void DoubleBuffer(CDC \*pDC);//双缓冲绘图

protected:

CTriView triview;//三视图类

void CTestView::DoubleBuffer(CDC \* pDC)

{

CRect rect;//定义客户区

GetClientRect(&rect);//获得客户区的大小

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

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

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

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

CDC memDC[4];

CDC \*pViewDC[4];

CMainFrame \*pFrame = (CMainFrame\*)AfxGetMainWnd();//获得框架指针

CVView \*pVView = (CVView\*)pFrame->m\_wndSplitter.GetPane(0, 0);

//获得主视图窗格指针

CWView \*pWView = (CWView\*)pFrame->m\_wndSplitter.GetPane(0, 1);

//获得侧视图窗格指针

CHView \*pHView = (CHView\*)pFrame->m\_wndSplitter.GetPane(1, 0);

//获得俯视图窗格指针

pViewDC[0] = pDC;//多面体屏幕DC

pViewDC[1] = pVView->GetDC();//主视图屏幕DC

pViewDC[2] = pWView->GetDC();//侧视屏幕DC

pViewDC[3] = pHView->GetDC();//俯视图屏幕DC

CBitmap NewBitmap[4], \*pOldBitmap[4];//内存中承载图像的临时位图

for (int i = 0; i < 4; i++)//创建内存DC

{

memDC[i].CreateCompatibleDC(pViewDC[i]);

NewBitmap[i].CreateCompatibleBitmap(pViewDC[i], triview.nClientWidth, triview.nClientHeight);//创建兼容位图

pOldBitmap[i] = memDC[i].SelectObject(&NewBitmap[i]);

memDC[i].FillSolidRect(&rect, pViewDC[i]->GetBkColor());

//按原来背景填充客户区，否则是黑色

}

triview.DrawOView(&memDC[0]);//绘制斜等测图

triview.DrawVView(&memDC[1]);//绘制主视图

triview.DrawWView(&memDC[2]);//绘制侧视图

triview.DrawHView(&memDC[3]);//绘制俯视图

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

{

pViewDC[i]->BitBlt(0, 0, triview.nClientWidth, triview.nClientHeight, &memDC[i], 0, 0, SRCCOPY);

//将内存位图拷贝到屏幕

memDC[i].SelectObject(pOldBitmap[i]);//恢复位图

NewBitmap[i].DeleteObject();//删除位图

ReleaseDC(pViewDC[i]);//释放视图DC

}

}

void CTestView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)

{

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

CTransform3 trans;

trans.SetMat(triview.P, 6);

switch (nChar)

{

case VK\_UP:

trans.RotateX(1);//设定步长

break;

case VK\_DOWN:

trans.RotateX(-1);

break;

case VK\_LEFT:

trans.RotateY(-1);

break;

case VK\_RIGHT:

trans.RotateY(1);

break;

default:

break;

}

Invalidate(FALSE);

CView::OnKeyDown(nChar, nRepCnt, nFlags);

}

void CTestView::OnTimer(UINT\_PTR nIDEvent)

{

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

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

{

triview.RotateY();

Invalidate(FALSE);;

}

CView::OnTimer(nIDEvent);

}

BOOL CTestView::OnEraseBkgnd(CDC\* pDC)

{

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

return TRUE;

}

void CTestView::OnInitialUpdate()

{

CView::OnInitialUpdate();

// TODO: Add your specialized code here and/or call the base class

triview.TOMatrix();

triview.TVMatrix();

triview.THMatrix();

triview.TWMatrix();

triview.ReadPoint();

triview.ReadFace();

SetTimer(1, 50, NULL);//设置定时器

}

1. **实现效果：**

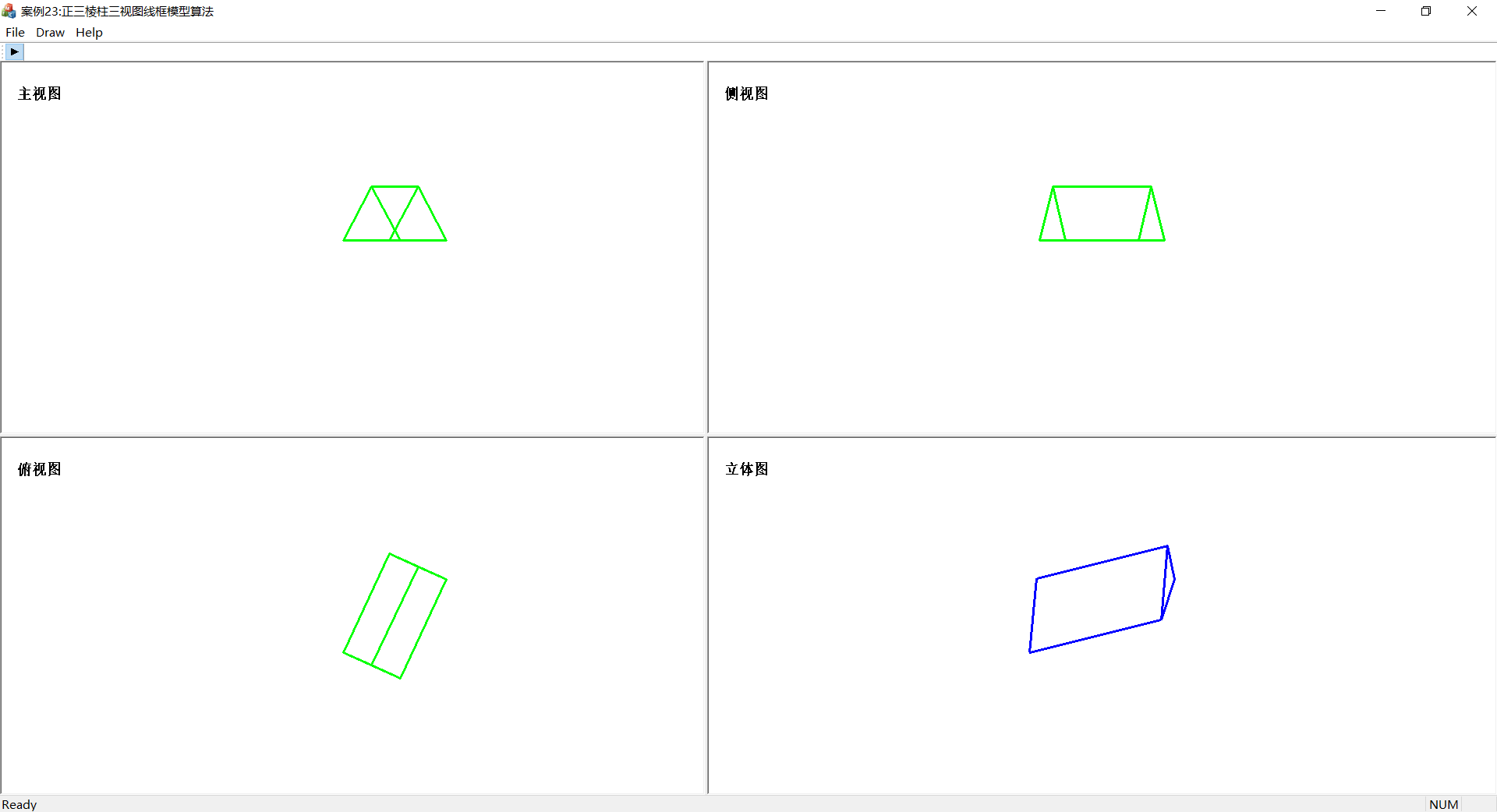


图1正三棱柱三视图线框模型算法效果图

**六、遇到的问题：**

屏幕切分：用到CSplitterWnd类中的[CSplitterWnd::CreateStatic](NULL)方法和[CSplitterWnd::CreateView](NULL)方法。[CSplitterWnd::CreateStatic](NULL)方法的作用是调用创建静态拆分窗口并将其附加到 CSplitterWnd 对象。[CSplitterWnd::CreateView](ms-xhelp://?method=page&id=1e0909b1-c66b-498d-b065-e59ad3a268a5&vendor=Microsoft&topicVersion=140&topicLocale=ZH-CN)方法的作用是调用创建窗格在拆分窗口。

创建继承于CView类的CHView、CVView、CWView类，CView类的作用是为用户定义的视图选件类提供了基本功能。创建过程如下：