// DDA画直线响应函数

void CLine\_ScanView::OnLineDda()

{

// TODO: 在此添加命令处理程序代码

Line l;

l.DoModal();

DDALine(l.x1,l.y1,l.x2,l.y2,NULL);

}

// Bresenham画直线响应函数

void CLine\_ScanView::OnLineBresenham()

{

// TODO: 在此添加命令处理程序代码

Line l;

l.DoModal();

BresenhamLine(l.x1,l.y1,l.x2,l.y2,NULL);

}

// 中点Bresenham画直线响应函数

void CLine\_ScanView::OnLineMidbresenham()

{

// TODO: 在此添加命令处理程序代码

Line l;

l.DoModal();

MidBresenhamLine(l.x1,l.y1,l.x2,l.y2,NULL);

}

// DDA画直线算法

void CLine\_ScanView::DDALine(int x1, int y1, int x2, int y2, int color)

{

int dx,dy,epsl,k;

float x=10,y=10,xIncre,yIncre;

dx = x2-x1;

dy = y2-y1;

k = dy/dx;

y = y1;

if(abs(dx)>abs(dy))

epsl = abs(dx);

else

epsl = abs(dy);

xIncre = (float)dx/(float)epsl;

yIncre = (float)dy/(float)epsl;

CClientDC dc(this);

for(k=0;k<=epsl;k++){

dc.SetPixel((int) x,(int) y,color);

x += xIncre;

y += yIncre;

}

}

// MidBresenham画直线算法

void CLine\_ScanView::MidBresenhamLine(int x1, int y1, int x2, int y2, int color)

{

int dx,dy,d,UpIncre,DownIncre,x,y;

if(x1>x2){

x=x2;x2=x1;x1=x;

y=y2;y2=y1;y1=y;

}

x=x1;

y=y1;

dx=x2-x1;

dy=y2-y1;

d=dx-2\*dy;

CClientDC dc(this);

UpIncre=2\*dx-2\*dy;

DownIncre=-2\*dy;

while(x<=x2){

dc.SetPixel(x,y,color);

x++;

if(d<0){

y++;

d += UpIncre;

}

else

d += DownIncre;

}

}

// Bresenham画直线算法

void CLine\_ScanView::BresenhamLine(int x1, int y1, int x2, int y2, int color)

{

int x,y,dx,dy,e;

dx = x2-x1;

dy = y2-y1;

e = -dx;

x = x1;

y = y1;

CClientDC dc(this);

while(x<=x2){

dc.SetPixel(x,y,color);

x++;

e = e+2\*dy;

if(e>0){

y++;

e=e-2\*dx;

}

}

}

// 中点画圆响应函数

void CLine\_ScanView::OnCircleMidbresenham()

{

// TODO: 在此添加命令处理程序代码

Circle c;

c.DoModal();

MidBresenhamCircle(c.r,NULL);

}

// 中点画圆算法

void CLine\_ScanView::MidBresenhamCircle(int r, int color)

{

int x = 0;

int y = r;

int d = 1-r;

int x0 = 100;

int y0 = 100;

CDC \*pDC = GetDC();

while(x<=y){

pDC->SetPixel(x+x0,y+y0,color);

pDC->SetPixel(-x+x0,y+y0,color);

pDC->SetPixel(-x+x0,-y+y0,color);

pDC->SetPixel(x+x0,-y+y0,color);

pDC->SetPixel(y+x0,x+y0,color);

pDC->SetPixel(-y+x0,x+y0,color);

pDC->SetPixel(-y+x0,-x+y0,color);

pDC->SetPixel(y+x0,-x+y0,color);

if(d<0)

d+=2\*x+3;

else{

d+=2\*(x-y)+5;

y--;

}

x++;

}

}

// 中点画椭圆响应函数

void CLine\_ScanView::OnEllipseMidbresenham()

{

// TODO: 在此添加命令处理程序代码

TuoYuan ty;

ty.DoModal();

MidBresenhamEllipse(ty.a,ty.b,NULL);

}

// 中点画椭圆算法

void CLine\_ScanView::MidBresenhamEllipse(int a, int b, int color)

{

int x = 0;

int y = b;

int x0 = 100;

int y0 = 100;

float d1,d2;

d1 = b\*b + a\*a\*(-b+0.25);

CDC \*pDC = GetDC();

pDC->SetPixel(x+x0,y+y0,color);

pDC->SetPixel(-x+x0,-y+y0,color);

pDC->SetPixel(-x+x0,y+y0,color);

pDC->SetPixel(x+x0,-y+y0,color);

while(b\*b\*(x+1)<a\*a\*(y-0.5)){

if(d1<0){

d1+=b\*b\*(2\*x+3);

x++;

}

else{

d1+=b\*b\*(2\*x+3)+a\*a\*(-2\*y+2);

x++;

y--;

}

pDC->SetPixel(x+x0,y+y0,color);

pDC->SetPixel(-x+x0,-y+y0,color);

pDC->SetPixel(-x+x0,y+y0,color);

pDC->SetPixel(x+x0,-y+y0,color);

}

d2=b\*b\*(x+0.5)\*(x+0.5)+a\*a\*(y-1)\*(y-1)-a\*a\*b\*b;

while(y>0){

if(d2<=0){

d2+=b\*b\*(2\*x+2)+a\*a\*(-2\*y+3);

x++;

y--;

}

else{

d2+=a\*a\*(-2\*y+3);

y--;

}

pDC->SetPixel(x+x0,y+y0,color);

pDC->SetPixel(-x+x0,-y+y0,color);

pDC->SetPixel(-x+x0,y+y0,color);

pDC->SetPixel(x+x0,-y+y0,color);

}

}

// 多边形的扫描转换响应函数

void CLine\_ScanView::OnPolygonScan()

{

// TODO: 在此添加命令处理程序代码

CDC\* pDC = GetDC();

const int POINTNUM=6; //多边形点数.

/\*\*\*\*\*\*定义结构体用于活性边表AET和新边表NET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct XET

{

float x;

float dx,ymax;

XET\* next;

}AET,NET;

/\*\*\*\*\*\*定义点结构体point\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct point

{

float x;

float y;

}polypoint[POINTNUM]={250,50,550,150,550,400,250,250,100,350,100,100};//多边形顶点

//mypoint[POINTNUM]={100,100,200,100,200,200,100,200};//正方形

/\*\*\*\*\*\*计算最高点的y坐标(扫描到此结束)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int MaxY=0;

int i;

for(i=0;i<POINTNUM;i++)

if(polypoint[i].y>MaxY)

MaxY=polypoint[i].y;

/\*\*\*\*\*\*\*初始化AET表\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

AET \*pAET=new AET;

pAET->next=NULL;

/\*\*\*\*\*\*初始化NET表\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NET \*pNET[1024];

for(i=0;i<=MaxY;i++)

{

pNET[i]=new NET;

pNET[i]->next=NULL;

}

/\*\*\*\*\*\*扫描并建立NET表\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for(i=0;i<=MaxY;i++)

{

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

if(polypoint[j].y==i)

{

if(polypoint[(j-1+POINTNUM)%POINTNUM].y>polypoint[j].y)

{

NET \*p=new NET;

p->x=polypoint[j].x;

p->ymax=polypoint[(j-1+POINTNUM)%POINTNUM].y;

p->dx=(polypoint[(j-1+POINTNUM)%POINTNUM].x-polypoint[j].x)/(polypoint[(j-1+POINTNUM)%POINTNUM].y-polypoint[j].y);

p->next=pNET[i]->next;

pNET[i]->next=p;

}

if(polypoint[(j+1+POINTNUM)%POINTNUM].y>polypoint[j].y)

{

NET \*p=new NET;

p->x=polypoint[j].x;

p->ymax=polypoint[(j+1+POINTNUM)%POINTNUM].y;

p->dx=(polypoint[(j+1+POINTNUM)%POINTNUM].x-polypoint[j].x)/(polypoint[(j+1+POINTNUM)%POINTNUM].y-polypoint[j].y);

p->next=pNET[i]->next;

pNET[i]->next=p;

}

}

}

/\*\*\*\*\*\*建立并更新活性边表AET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for(i=0;i<=MaxY;i++)

{

//计算新的交点x,更新AET\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NET \*p=pAET->next;

while(p)

{

p->x=p->x + p->dx;

p=p->next;

}

//更新后新AET先排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//断表排序,不再开辟空间

AET \*tq=pAET;

p=pAET->next;

tq->next=NULL;

while(p)

{

while(tq->next && p->x >= tq->next->x)

tq=tq->next;

NET \*s=p->next;

p->next=tq->next;

tq->next=p;

p=s;

tq=pAET;

}

//(改进算法)先从AET表中删除ymax==i的结点\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

AET \*q=pAET;

p=q->next;

while(p)

{

if(p->ymax==i)

{

q->next=p->next;

delete p;

p=q->next;

}

else

{

q=q->next;

p=q->next;

}

}

//将NET中的新点加入AET,并用插入法按X值递增排序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

p=pNET[i]->next;

q=pAET;

while(p)

{

while(q->next && p->x >= q->next->x)

q=q->next;

NET \*s=p->next;

p->next=q->next;

q->next=p;

p=s;

q=pAET;

}

/\*\*\*\*\*\*配对填充颜色\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

p=pAET->next;

while(p && p->next)

{

for(float j=p->x;j<=p->next->x;j++)

pDC->SetPixel(static\_cast<int>(j),i,RGB(255,0,0));

p=p->next->next;//考虑端点情况

}

}

}

// 多边形的4种子填充响应函数

void CLine\_ScanView::OnPolygonArea4()

{

// TODO: 在此添加命令处理程序代码

DuoBianXing dbx;

dbx.DoModal();

BoundaryFill4(dbx.x, dbx.y, RGB(0,255,0), RGB(255,0,0));

}

// 多边形的4种子填充算法

void CLine\_ScanView::BoundaryFill4(int x, int y, int boundarycolor, int newcolor)

{

CDC\* pDC = GetDC();

CPen pen;

pen.CreatePen(PS\_SOLID,2,RGB(0,255,0));

pDC->SelectObject(&pen);

int a[4][2] = {200,100,180,120,220,120,200,100};

pDC->MoveTo(a[0][0],a[0][1]);

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

pDC->LineTo(a[i][0],a[i][1]);

}

int color = pDC->GetPixel(x,y);

if(color!=newcolor&&color!=boundarycolor){

pDC->SetPixel(x,y,newcolor);

BoundaryFill4(x,y+1,boundarycolor,newcolor);

BoundaryFill4(x,y-1,boundarycolor,newcolor);

BoundaryFill4(x+1,y,boundarycolor,newcolor);

BoundaryFill4(x-1,y,boundarycolor,newcolor);

}

}

// 多边形的8种子填充响应函数

void CLine\_ScanView::OnPolygonArea8()

{

// TODO: 在此添加命令处理程序代码

DuoBianXing dbx;

dbx.DoModal();

BoundaryFill8(dbx.x,dbx.y,RGB(0,255,0),RGB(255,0,0));

}

// 多边形的8种子填充算法

void CLine\_ScanView::BoundaryFill8(int x, int y, int boundarycolor, int newcolor)

{

CDC\* pDC = GetDC();

CPen pen;

pen.CreatePen(PS\_SOLID,2,RGB(0,255,0));

pDC->SelectObject(&pen);

int a[4][2] = {200,100,180,120,220,120,200,100};

pDC->MoveTo(a[0][0],a[0][1]);

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

pDC->LineTo(a[i][0],a[i][1]);

}

int color = pDC->GetPixel(x,y);

if(color!=newcolor&&color!=boundarycolor){

pDC->SetPixel(x,y,newcolor);

BoundaryFill8(x,y+1,boundarycolor,newcolor);

BoundaryFill8(x+1,y+1,boundarycolor,newcolor);

BoundaryFill8(x,y-1,boundarycolor,newcolor);

BoundaryFill8(x-1,y-1,boundarycolor,newcolor);

BoundaryFill8(x+1,y,boundarycolor,newcolor);

BoundaryFill8(x+1,y-1,boundarycolor,newcolor);

BoundaryFill8(x-1,y,boundarycolor,newcolor);

BoundaryFill8(x-1,y+1,boundarycolor,newcolor);

}

}

// 二维变换的比例变换响应函数

void CLine\_ScanView::OnTwoProportion()

{

// TODO: 在此添加命令处理程序代码

CClientDC dc(this);

CPen pen1;

pen1.CreatePen(PS\_SOLID,2,RGB(255,10,255));

int a[11][2]={100,105,90,135,60,135,85,155,75,185,100,165,125,185,115,155,140,135,110,135,100,105};

dc.SelectObject(&pen1);

dc.MoveTo (a[0][0],a[0][1]);

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0],a[i][1]);

}

double sx=0.5,sy=0.5;

dc.MoveTo (a[0][0]\*sx,a[0][1]\*sy);

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0]\*sx,a[i][1]\*sy);

Sleep(5);

}

pen1.DeleteObject();

}

// 二维变换的旋转变换响应函数

void CLine\_ScanView::OnTwoRotating()

{

// TODO: 在此添加命令处理程序代码

CClientDC dc(this);

CPen pen1;

pen1.CreatePen(PS\_SOLID,2,RGB(255,10,255));

int a[11][2]={100,105,90,135,60,135,85,155,75,185,100,165,125,185,115,155,140,135,110,135,100,105};

dc.SelectObject(&pen1);

dc.MoveTo (a[0][0],a[0][1]);

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0],a[i][1]);

}

int t = 600;

dc.MoveTo (a[0][0]\*cos((t/6)\*1.0)-a[0][1]\*sin((t/6)\*1.0),//一个点

a[0][0]\*sin((t/6)\*1.0)+a[0][1]\*cos((t/6)\*1.0));

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0]\*cos((t/6)\*1.0)-a[i][1]\*sin((t/6)\*1.0),//一个点

a[i][0]\*sin((t/6)\*1.0)+a[i][1]\*cos((t/6)\*1.0));

}

pen1.DeleteObject();

}

// 二维变换的错切变换响应函数

void CLine\_ScanView::OnTwoShear()

{

// TODO: 在此添加命令处理程序代码

CClientDC dc(this);

CPen pen1;

pen1.CreatePen(PS\_SOLID,2,RGB(255,10,255));

int a[11][2]={100,105,90,135,60,135,85,155,75,185,100,165,125,185,115,155,140,135,110,135,100,105};

dc.SelectObject(&pen1);

dc.MoveTo (a[0][0],a[0][1]);

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0],a[i][1]);

}

double c=2,d=1.5,u=50;

dc.MoveTo (a[0][0]+c\*a[0][1],a[0][1]-u);

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0]+c\*a[i][1],a[i][1]-u);

}

int s=170,h=200;

dc.MoveTo (a[0][0]+s,d\*a[0][0]+a[0][1]-h);

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0]+s,d\*a[i][0]+a[i][1]-h);

}

pen1.DeleteObject();

}

// 二维变换的对称变换响应函数

void CLine\_ScanView::OnTwoSymmetry()

{

// TODO: 在此添加命令处理程序代码

CClientDC dc(this);

CPen pen1;

pen1.CreatePen(PS\_SOLID,2,RGB(255,10,255));

int a[11][2]={100,105,90,135,60,135,85,155,75,185,100,165,125,185,115,155,140,135,110,135,100,105};

dc.SelectObject(&pen1);

dc.MoveTo (a[0][0],a[0][1]);

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0],a[i][1]);

}

int m=0,b=200;

dc.MoveTo (a[0][0]\*(1-m\*m)/(1+m\*m)+2\*(a[0][1]-b)\*m/(1+m\*m),//一个点

a[0][0]\*(2\*m)/(1+m\*m)+(a[0][1]-b)\*(m\*m-1)/(1+m\*m)+b);

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0]\*(1-m\*m)/(1+m\*m)+2\*(a[i][1]-b)\*m/(1+m\*m), a[i][0]\*(2\*m)/(1+m\*m)+(a[i][1]-b)\*(m\*m-1)/(1+m\*m)+b);

}

pen1.DeleteObject();

}

// 二维变换的平移变换响应函数

void CLine\_ScanView::OnTwoTranslation()

{

// TODO: 在此添加命令处理程序代码

CClientDC dc(this);

CPen pen1;

pen1.CreatePen(PS\_SOLID,2,RGB(255,10,255));

int a[11][2]={100,105,90,135,60,135,85,155,75,185,100,165,125,185,115,155,140,135,110,135,100,105};

dc.SelectObject(&pen1);

dc.MoveTo (a[0][0],a[0][1]);

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0],a[i][1]);

}

int tx=120,ty=100;

dc.MoveTo (a[0][0]+tx,a[0][1]+ty);

for(int i=0;i<11;i++) {

dc.LineTo (a[i][0]+tx,a[i][1]+ty);

}

pen1.DeleteObject();

}

// 三维变换的比例变换响应函数

void CLine\_ScanView::OnThreeProportion()

{

// TODO: 在此添加命令处理程序代码

CClientDC dc(this);

CPen pen1,pen2;

pen1.CreatePen(PS\_SOLID,2,RGB(0,0,0));

pen2.CreatePen(PS\_SOLID,2,RGB(0,255,0));

int a[4][4] = {100,150,0,1,50,200,0,1,150,200,0,1,100,150,0,1};

dc.SelectObject(&pen1);

dc.MoveTo (a[0][0],a[0][1]);

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

dc.LineTo(a[i][0],a[i][1]);

}

int b[4][4] = {2,0,0,0,0,2,0,0,0,0,2,0,0,0,0,1};

int c[4][4];

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

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

c[i][j] = 0;

}

}

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

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

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

c[i][j]=c[i][j]+a[i][k]\*b[k][j];

}

}

}

dc.SelectObject(&pen2);

dc.MoveTo(c[0][0],c[0][1]);

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

dc.LineTo(c[i][0],c[i][1]);

}

pen1.DeleteObject();

pen2.DeleteObject();

}

// 三维变换的旋转变换响应函数

void CLine\_ScanView::OnThreeRotating()

{

// TODO: 在此添加命令处理程序代码

CClientDC dc(this);

CPen pen1,pen2;

pen1.CreatePen(PS\_SOLID,2,RGB(0,0,0));

pen2.CreatePen(PS\_SOLID,2,RGB(255,0,0));

int a[4][4] = {100,150,0,1,50,200,0,1,150,200,0,1,100,150,0,1};

dc.SelectObject(&pen1);

dc.MoveTo (a[0][0],a[0][1]);

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

dc.LineTo (a[i][0],a[i][1]);

}

int t = 600;

dc.SelectObject(&pen2);

dc.MoveTo (a[0][0]\*cos((t/6)\*1.0)-a[0][1]\*sin((t/6)\*1.0),//一个点

a[0][0]\*sin((t/6)\*1.0)+a[0][1]\*cos((t/6)\*1.0));

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

dc.LineTo (a[i][0]\*cos((t/6)\*1.0)-a[i][1]\*sin((t/6)\*1.0),//一个点

a[i][0]\*sin((t/6)\*1.0)+a[i][1]\*cos((t/6)\*1.0));

}

pen1.DeleteObject();

pen2.DeleteObject();

}

// 三维变换的错切变换响应函数

void CLine\_ScanView::OnThreeShear()

{

// TODO: 在此添加命令处理程序代码

CClientDC dc(this);

CPen pen1,pen2;

pen1.CreatePen(PS\_SOLID,2,RGB(0,0,0));

pen2.CreatePen(PS\_SOLID,2,RGB(255,255,0));

int a[4][4] = {100,150,0,1,50,200,0,1,150,200,0,1,100,150,0,1};

dc.SelectObject(&pen1);

dc.MoveTo (a[0][0],a[0][1]);

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

dc.LineTo(a[i][0],a[i][1]);

}

int b[4][4] = {1,2,0,0,1.5,1,0,0,0,0,1,0,0,0,0,1};

int c[4][4];

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

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

c[i][j] = 0;

}

}

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

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

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

c[i][j]=c[i][j]+a[i][k]\*b[k][j];

}

}

}

dc.SelectObject(&pen2);

dc.MoveTo(c[0][0],c[0][1]);

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

dc.LineTo(c[i][0],c[i][1]);

}

pen1.DeleteObject();

pen2.DeleteObject();

}

// 三维变换的对称变换响应函数

void CLine\_ScanView::OnThreeSymmetry()

{

// TODO: 在此添加命令处理程序代码

CClientDC dc(this);

CPen pen1,pen2;

pen1.CreatePen(PS\_SOLID,2,RGB(0,0,0));

pen2.CreatePen(PS\_SOLID,2,RGB(0,255,255));

int a[4][4] = {100,150,0,1,50,200,0,1,150,200,0,1,100,150,0,1};

dc.SelectObject(&pen1);

dc.MoveTo (a[0][0],a[0][1]);

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

dc.LineTo(a[i][0],a[i][1]);

}

int b[4][4] = {0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1}; //沿y=x对称

int c[4][4];

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

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

c[i][j] = 0;

}

}

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

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

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

c[i][j]=c[i][j]+a[i][k]\*b[k][j];

}

}

}

dc.SelectObject(&pen2);

dc.MoveTo(c[0][0],c[0][1]);

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

dc.LineTo(c[i][0],c[i][1]);

}

pen1.DeleteObject();

pen2.DeleteObject();

}

// 三维变换的平移变换响应函数

void CLine\_ScanView::OnThreeTranslation()

{

// TODO: 在此添加命令处理程序代码

CClientDC dc(this);

CPen pen1,pen2;

pen1.CreatePen(PS\_SOLID,2,RGB(0,0,0));

pen2.CreatePen(PS\_SOLID,2,RGB(0,0,255));

int a[4][4] = {100,150,0,1,50,200,0,1,150,200,0,1,100,150,0,1};

dc.SelectObject(&pen1);

dc.MoveTo (a[0][0],a[0][1]);

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

dc.LineTo(a[i][0],a[i][1]);

}

int b[4][4] = {1,0,0,0,0,1,0,0,0,0,1,0,100,100,0,1};

int c[4][4];

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

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

c[i][j] = 0;

}

}

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

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

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

c[i][j]=c[i][j]+a[i][k]\*b[k][j];

}

}

}

dc.SelectObject(&pen2);

dc.MoveTo(c[0][0],c[0][1]);

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

dc.LineTo(c[i][0],c[i][1]);

}

pen1.DeleteObject();

pen2.DeleteObject();

}

// 直线段CS裁剪算法的响应函数

void CLine\_ScanView::OnLineCs()

{

// TODO: 在此添加命令处理程序代码

Line\_Cut lc;

lc.DoModal();

int min\_clip\_x = 100 ,min\_clip\_y = 100 ,max\_clip\_x = 300,max\_clip\_y=300 ;

CClientDC dc(this);

dc.MoveTo(min\_clip\_x,min\_clip\_y);

dc.LineTo(min\_clip\_x,max\_clip\_y);

dc.MoveTo(min\_clip\_x,max\_clip\_y );

dc.LineTo(max\_clip\_x,max\_clip\_y);

dc.MoveTo(max\_clip\_x,max\_clip\_y );

dc.LineTo(max\_clip\_x,min\_clip\_y);

dc.MoveTo(max\_clip\_x,min\_clip\_y );

dc.LineTo(min\_clip\_x,min\_clip\_y);

if(CutLine(lc.x0,lc.y0,lc.x1,lc.y1)){

dc.MoveTo(lc.x0,lc.y0 );

dc.LineTo(lc.x1,lc.y1);

}

}

// 直线段中点分割算法的响应函数

void CLine\_ScanView::OnLineMidpoint()

{

// TODO: 在此添加命令处理程序代码

int x0,y0,x1,y1;

CClientDC dc(this);

int min\_clip\_x = 100 ,min\_clip\_y = 100 ,max\_clip\_x = 300,max\_clip\_y=300 ;

dc.MoveTo(min\_clip\_x,min\_clip\_y);

dc.LineTo(min\_clip\_x,max\_clip\_y);

dc.MoveTo(min\_clip\_x,max\_clip\_y );

dc.LineTo(max\_clip\_x,max\_clip\_y);

dc.MoveTo(max\_clip\_x,max\_clip\_y );

dc.LineTo(max\_clip\_x,min\_clip\_y);

dc.MoveTo(max\_clip\_x,min\_clip\_y );

dc.LineTo(min\_clip\_x,min\_clip\_y);

x0=120;y0=90;x1=310,y1=310;

Mid\_CutLine(x0,y0,x1,y1);

}

/\*void CLine\_ScanView::CompOutCode(int x, int y, CRect\* rect, OutCode\* outCode)

{

outCode->all = 0;

outCode->top = outCode->bottom =0;

if (y < rect->top)

{

outCode->top = 1;

outCode->all += 1;

}

else if (y > rect->bottom)

{

outCode->bottom = 1;

outCode->all += 1;

}

outCode->right = outCode->left = 0;

if (x > rect->right)

{

outCode->right = 1;

outCode->all += 1;

}

else if(x < rect->left)

{

outCode->left = 1;

outCode->all += 1;

}

}\*/

// 直线段的CS裁剪算法

int CLine\_ScanView::CutLine(int& x0, int& y0, int& x1, int& y1)

{

#define CLIP\_CODE\_C 0x0000

#define CLIP\_CODE\_N 0x0008

#define CLIP\_CODE\_S 0x0004

#define CLIP\_CODE\_E 0x0002

#define CLIP\_CODE\_W 0x0001

#define CLIP\_CODE\_NE 0x000a

#define CLIP\_CODE\_SE 0x0006

#define CLIP\_CODE\_NW 0x0009

#define CLIP\_CODE\_SW 0x0005

int xc0 = x0 ,yc0 = y0 , xc1=x1 , yc1=y1 ;

int min\_clip\_x = 100 ,min\_clip\_y = 100 ,max\_clip\_x = 300,max\_clip\_y=300 ;

int p0\_code = 0 ,p1\_code = 0 ;

//确定各个顶点所在的位置代码

if(y0<min\_clip\_y)

p0\_code|=CLIP\_CODE\_N;

else if(y0>max\_clip\_y)

p0\_code|=CLIP\_CODE\_S;

if(x0<min\_clip\_x)

p0\_code|=CLIP\_CODE\_W;

else if(x0>max\_clip\_x)

p0\_code|=CLIP\_CODE\_E;

if(y1<min\_clip\_y)

p1\_code|=CLIP\_CODE\_N;

else if(y1>max\_clip\_y)

p1\_code|=CLIP\_CODE\_S;

if(x1<min\_clip\_x)

p1\_code|=CLIP\_CODE\_W;

else if(x1>max\_clip\_x)

p1\_code|=CLIP\_CODE\_E;

//先检测一些简单的情况

if(p0\_code&p1\_code) //有相同的位置代码，表示在裁剪区外部

return 0 ;

if(p0\_code==0&&p1\_code==0) //表示两个点都在裁剪区内，不需要裁剪

return 1 ;

//判断第一个点的位置代码

switch(p0\_code)

{

case CLIP\_CODE\_C:

break;

case CLIP\_CODE\_N:

{

yc0 = min\_clip\_y ;

xc0 = x0 + 0.5 + (yc0-y0)\*(x1-x0)/(y1-y0);

break ;

}

case CLIP\_CODE\_S:

{

yc0 = max\_clip\_y;

xc0 = x0 + 0.5 + (yc0-y0)\*(x1-x0)/(y1-y0);

break ;

}

case CLIP\_CODE\_W:

{

xc0=min\_clip\_x;

yc0=y0+0.5+(xc0-x0)\*(y1-y0)/(x1-x0);

break;

}

case CLIP\_CODE\_E:

{

xc0=max\_clip\_x;

yc0=y0+0.5+(xc0-x0)\*(y1-y0)/(x1-x0);

break;

}

case CLIP\_CODE\_NE:

{

yc0 = min\_clip\_y;

xc0 = x0 + 0.5 + (yc0-y0)\*(x1-x0)/(y1-y0);

if(xc0<min\_clip\_x||xc0>max\_clip\_x)

{

xc0=max\_clip\_x;

yc0=y0+0.5+(xc0-x0)\*(y1-y0)/(x1-x0);

}

break;

}

case CLIP\_CODE\_SE:

{

yc0 = max\_clip\_y;

xc0 = x0 + 0.5 + (yc0-y0)\*(x1-x0)/(y1-y0);

if(xc0<min\_clip\_x||xc0>max\_clip\_x)

{

xc0=max\_clip\_x;

yc0=y0+0.5+(xc0-x0)\*(y1-y0)/(x1-x0);

}

break;

}

case CLIP\_CODE\_NW:

{

yc0=min\_clip\_y;

xc0 = x0 + 0.5 + (yc0-y0)\*(x1-x0)/(y1-y0);

if(xc0<min\_clip\_x||xc0>max\_clip\_x)

{

xc0=min\_clip\_x;

yc0=y0+0.5+(xc0-x0)\*(y1-y0)/(x1-x0);

}

break;

}

case CLIP\_CODE\_SW:

{

yc0=max\_clip\_y;

xc0 = x0 + 0.5 + (yc0-y0)\*(x1-x0)/(y1-y0);

if(xc0<min\_clip\_x||xc0>max\_clip\_x)

{

xc0=min\_clip\_x;

yc0=y0+0.5+(xc0-x0)\*(y1-y0)/(x1-x0);

}

break;

}

default:

break;

} // end switch(p0\_code)

//判断第二个点的位置代码

switch(p1\_code)

{

case CLIP\_CODE\_C:

break;

case CLIP\_CODE\_N:

{

yc1 = min\_clip\_y ;

xc1 = x1 + 0.5 + (yc1-y1)\*(x1-x0)/(y1-y0);

break ;

}

case CLIP\_CODE\_S:

{

yc1 = max\_clip\_y;

xc1 = x1 + 0.5 + (yc1-y1)\*(x1-x0)/(y1-y0);

break ;

}

case CLIP\_CODE\_W:

{

xc1=min\_clip\_x;

yc1=y1+0.5+(xc1-x1)\*(y1-y0)/(x1-x0);

break;

}

case CLIP\_CODE\_E:

{

xc1=max\_clip\_x;

yc1=y1+0.5+(xc1-x1)\*(y1-y0)/(x1-x0);

break;

}

case CLIP\_CODE\_NE:

{

yc1 = min\_clip\_y;

xc1 = x1 + 0.5 + (yc1-y1)\*(x1-x0)/(y1-y0);

if(xc1<min\_clip\_x||xc1>max\_clip\_x)

{

xc1=max\_clip\_x;

yc1=y1+0.5+(xc1-x1)\*(y1-y0)/(x1-x0);

}

break;

}

case CLIP\_CODE\_SE:

{

yc1 = max\_clip\_y;

xc1 = x1 + 0.5 + (yc1-y1)\*(x1-x0)/(y1-y0);

if(xc1<min\_clip\_x||xc1>max\_clip\_x)

{

xc1=max\_clip\_x;

yc1=y1+0.5+(xc1-x1)\*(y1-y0)/(x1-x0);

}

break;

}

case CLIP\_CODE\_NW:

{

yc1=min\_clip\_y;

xc1 = x1 + 0.5 + (yc1-y1)\*(x1-x0)/(y1-y0);

if(xc1<min\_clip\_x||xc1>max\_clip\_x)

{

xc1=min\_clip\_x;

yc1=y1+0.5+(xc1-x1)\*(y1-y0)/(x1-x0);

}

break;

}

case CLIP\_CODE\_SW:

{

yc1=max\_clip\_y;

xc1 = x1 + 0.5 + (yc1-y1)\*(x1-x0)/(y1-y0);

if(xc1<min\_clip\_x||xc1>max\_clip\_x)

{

xc1=min\_clip\_x;

yc1=y1+0.5+(xc1-x1)\*(y1-y0)/(x1-x0);

}

break;

}

default:

break;

} // end switch(p1\_code)

//进行最后的检测

if(xc0>max\_clip\_x||xc0<min\_clip\_x||

yc0>max\_clip\_y||yc0<min\_clip\_y||

xc1>max\_clip\_x||xc1<min\_clip\_x||

yc1>max\_clip\_y||yc1<min\_clip\_y)

{

//表示全部在裁剪区外部

return 0 ;

}

//将裁减后的数据返回

x0 = xc0 ;

x1 = xc1 ;

y0 = yc0 ;

y1 = yc1 ;

return 1 ;

}

// 直线段的中点分割算法

void CLine\_ScanView::Mid\_CutLine(int x0, int y0, int x1, int y1)

{

#define CLIP\_CODE\_C 0x0000

#define CLIP\_CODE\_N 0x0008

#define CLIP\_CODE\_S 0x0004

#define CLIP\_CODE\_E 0x0002

#define CLIP\_CODE\_W 0x0001

#define CLIP\_CODE\_NE 0x000a

#define CLIP\_CODE\_SE 0x0006

#define CLIP\_CODE\_NW 0x0009

#define CLIP\_CODE\_SW 0x0005

int min\_clip\_x = 100 ,min\_clip\_y = 100 ,max\_clip\_x = 300,max\_clip\_y=300 ;

int p0\_code = 0 ,p1\_code = 0 ;

//确定各个顶点所在的位置代码

if(y0<min\_clip\_y)

p0\_code|=CLIP\_CODE\_N;

else if(y0>max\_clip\_y)

p0\_code|=CLIP\_CODE\_S;

if(x0<min\_clip\_x)

p0\_code|=CLIP\_CODE\_W;

else if(x0>max\_clip\_x)

p0\_code|=CLIP\_CODE\_E;

if(y1<min\_clip\_y)

p1\_code|=CLIP\_CODE\_N;

else if(y1>max\_clip\_y)

p1\_code|=CLIP\_CODE\_S;

if(x1<min\_clip\_x)

p1\_code|=CLIP\_CODE\_W;

else if(x1>max\_clip\_x)

p1\_code|=CLIP\_CODE\_E;

//先检测一些简单的情况

if(p0\_code&p1\_code) //有相同的位置代码，表示在裁剪区外部

{return ;}

else if(p0\_code==0&&p1\_code==0) //表示两个点都在裁剪区内，不需要裁剪

{

CClientDC dc(this);

dc.MoveTo(x0,y0 );

dc.LineTo(x1,y1);

return;

} else

{

int x2,y2;

x2=(x0+x1)/2;

y2=(y0+y1)/2;

if(x0-x2>1||x2-x0>1)

{

Mid\_CutLine(x0,y0,x2,y2);

}

if(x1-x2>1||x2-x1>1)

{

Mid\_CutLine(x2,y2,x1,y1);

}

}

return;

}