//Bresenham画线

void bresLine(int x1, int y1, int x2, int y2)

{

TextOut(g\_hDc, x2, y2, L"Bresenham画线",11);

int x = x1, y = y1, dx = x2 - x1, dy = y2 - y1, e = -dx;

while (x <= x2)

{

SetPixel(g\_hDc,x, y, RGB(0, 255, 0));

x++;

e = e + (2 \* dy);

if (e > 0)

{

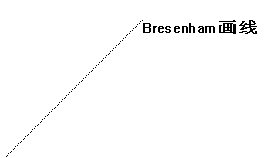
y++;

e = e - (2 \* dx);

}

}

}



//DDA画线

void DDALine(int x1, int y1, int x2, int y2)

{

TextOut(g\_hDc, x2, y2, L"DDA画线",5);

int dx = x2 - x1, dy = y2 - y1, epsl;

float x = x1, y = y1, xIncre, yIncre;

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

epsl = abs(dx);

else

epsl = abs(dy);

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

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

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

{

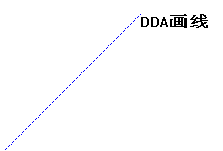
SetPixel(g\_hDc,(int)x + 0.5, (int)y + 0.5, RGB(0, 0, 255));

x += xIncre;

y += yIncre;

}

}



//中点Bresenham画线

void midBresLine(int x1, int y1, int x2, int y2)

{

TextOut(g\_hDc,x2, y2, L"中点Bresenham画线",13);

int dx, dy, d, upIncre, downIncre, x = x1, y = y1;

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;

upIncre = 2 \* dx - 2 \* dy; downIncre = -2 \* dy;

while (x <= x2)

{

SetPixel(g\_hDc,x, y, RGB(0, 255, 255));

x++;

if (d < 0)

{

y++;

d += upIncre;

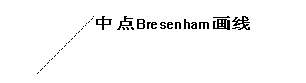
}

else

d += downIncre;

}

}



//库函数画线

void libLine(int x1, int y1, int x2, int y2)

{

TextOut(g\_hDc, x2, y2, L"库函数画线", 5);

MoveToEx(g\_hDc, x1, y1, NULL);

LineTo(g\_hDc, x2, y2);

HPEN hPen = CreatePen(PS\_DASH, 10, RGB(255, 0, 0));//sPen[]笔型；粗细；颜色

SelectObject(g\_hDc, hPen);

MoveToEx(g\_hDc, x1, y1-100, NULL);

LineTo(g\_hDc, x2, y2-100);

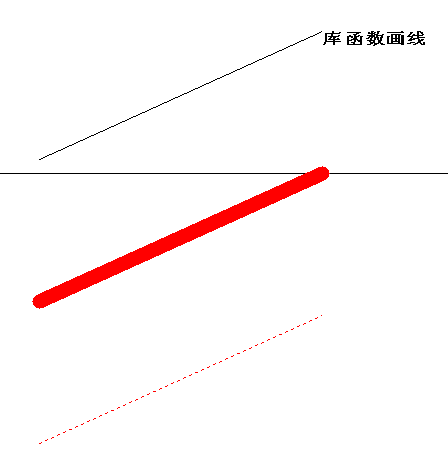
HPEN hPen2 = CreatePen(PS\_DOT, 1, RGB(255, 0, 0));//sPen[]笔型；粗细；颜色

SelectObject(g\_hDc, hPen2);

MoveToEx(g\_hDc, x1, y1 - 200, NULL);

LineTo(g\_hDc, x2, y2 - 200);

}



//中点Bresenham画圆

static void point(int x, int y, int x0, int y0)

{

SetPixel(g\_hDc, x0 + x, y0 + y, RGB(255, 0, 0)); SetPixel(g\_hDc, x0 + y, y0 + x, RGB(255, 0, 0));

SetPixel(g\_hDc, x0 - y, y0 + x, RGB(0, 255, 0)); SetPixel(g\_hDc, x0 - x, y0 + y, RGB(0, 255, 0));

SetPixel(g\_hDc, x0 - x, y0 - y, RGB(0, 0, 255)); SetPixel(g\_hDc, x0 - y, y0 - x, RGB(0, 0, 255));

SetPixel(g\_hDc, x0 + y, y0 - x, RGB(255, 255, 0)); SetPixel(g\_hDc, x0 + x, y0 - y, RGB(255, 255, 0));

}

void midBresCircle(int x0, int y0, int r)

{

TextOut(g\_hDc, x0, y0, L"Bresenham画圆",11);

int x = 0, y = r, d = 1 - r;

while (x <= y)

{

point(x, y, x0, y0);

if (d < 0)

d += 2 \* x + 3;

else

{

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

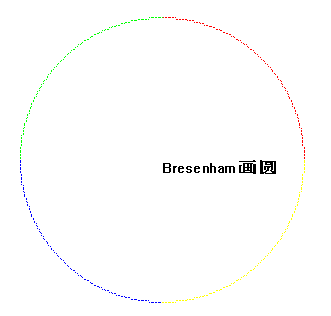
y--;

}

x++;

}

}



//中点Bresenham椭圆

static void point2(int x, int y, int x0, int y0)

{

SetPixel(g\_hDc, x0 + x, y0 + y, RGB(0, 0, 255)); SetPixel(g\_hDc, x0 - x, y0 - y, RGB(0, 0, 255));

SetPixel(g\_hDc, x0 - x, y0 + y, RGB(0, 0, 255)); SetPixel(g\_hDc, x0 + x, y0 - y, RGB(0, 0, 255));

}

void midBresEllipse(int x0, int y0, int a, int b)

{

TextOut(g\_hDc, x0, y0, L"中点Bresenham椭圆",13);

int x = 0, y = b;

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

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

point2(x, y, x0, y0);

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

{

if (d1 <= 0)

{d1 += b\*b\*(3 \* x + 3); x++;}

else

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

x++; y--;}

point2(x, y, x0, y0);

}

while (y > 0)

{

if (d2 <= 0)

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

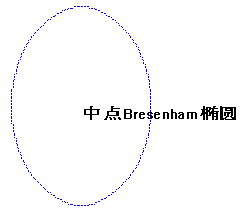
x++; y--;}

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

point2(x, y, x0, y0);

}

}



//矩形绘制

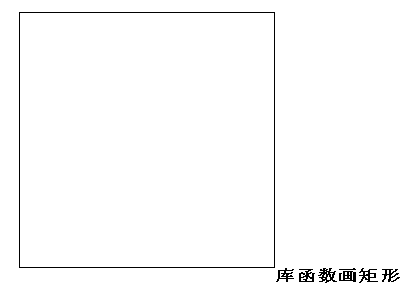
void glRect(int x0, int y0, int x1, int y1)

{

TextOut(g\_hDc,x1, y1, L"库函数画矩形",6);

Rectangle(g\_hDc,x0, y0, x1, y1);

}



//多边形绘制

void polygon(int\* arrX, int\* arrY, int n)

{

TextOut(g\_hDc, arrX[0], arrY[0], L"多边形绘制", 5);

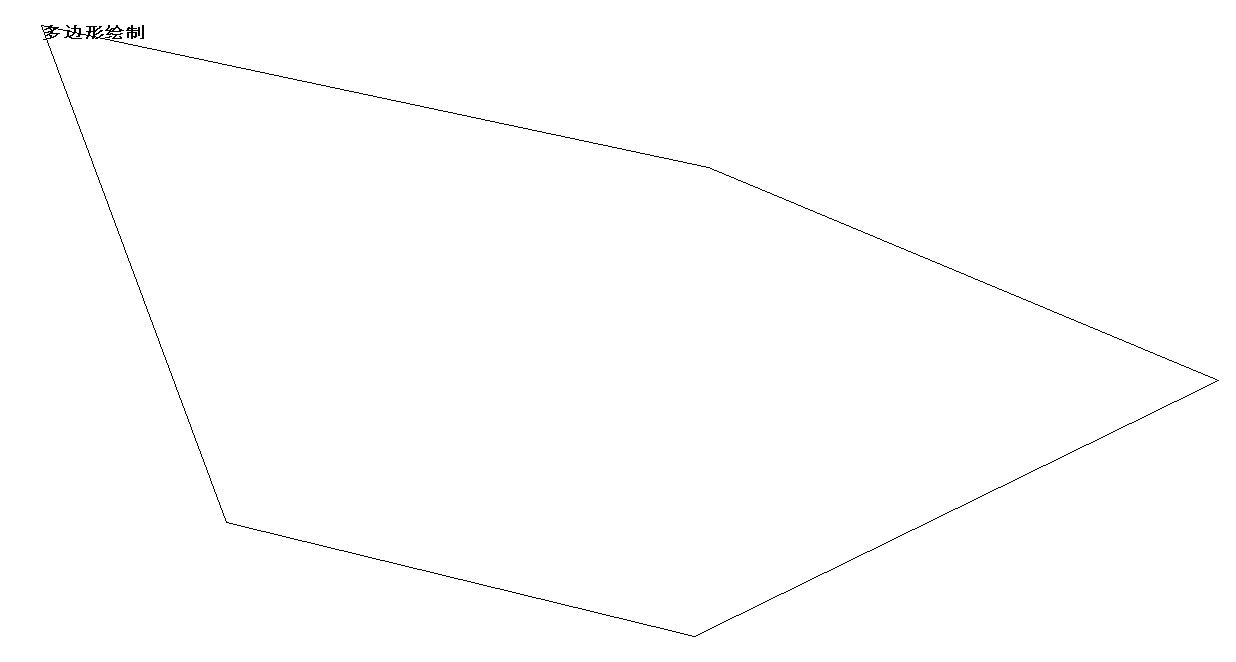
MoveToEx(g\_hDc, arrX[0], arrY[0], NULL);

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

LineTo(g\_hDc, arrX[i], arrY[i]);

LineTo(g\_hDc, arrX[0], arrY[0]);

}



//绘制弧线

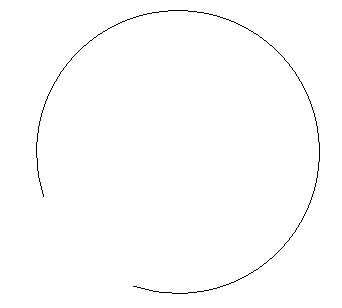
void arc(int\* arrX,int\* arrY)

{

Arc(g\_hDc, arrX[0],arrY[0],arrX[1],arrY[1],

arrX[2],arrY[2],arrX[3],arrY[3]);

}



//椭圆的平移变换

void moveEllipse(int x0, int y0, int x1, int y1)

{

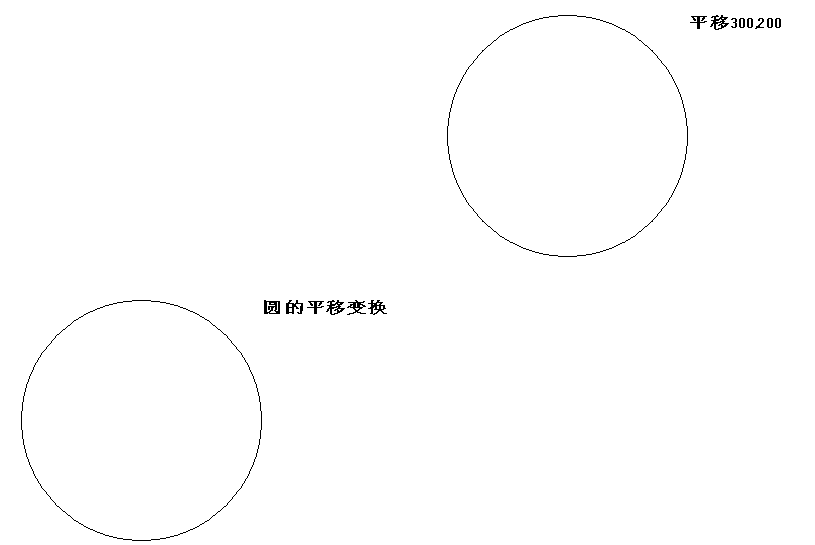
TextOut(g\_hDc,x0, y0, L"圆的平移变换",6);

Ellipse(g\_hDc,x0, y0, x1, y1);

TextOut(g\_hDc, x0 + 300, y0 + 200, L"平移300,200后", 9);

Ellipse(g\_hDc, x0 + 300, y0 + 200, x1 + 300, y1 + 200);

}



//椭圆的比例变换

void rateEllipse(int x0, int y0, int x1, int y1)

{

int Sx = 2, Sy = 2;

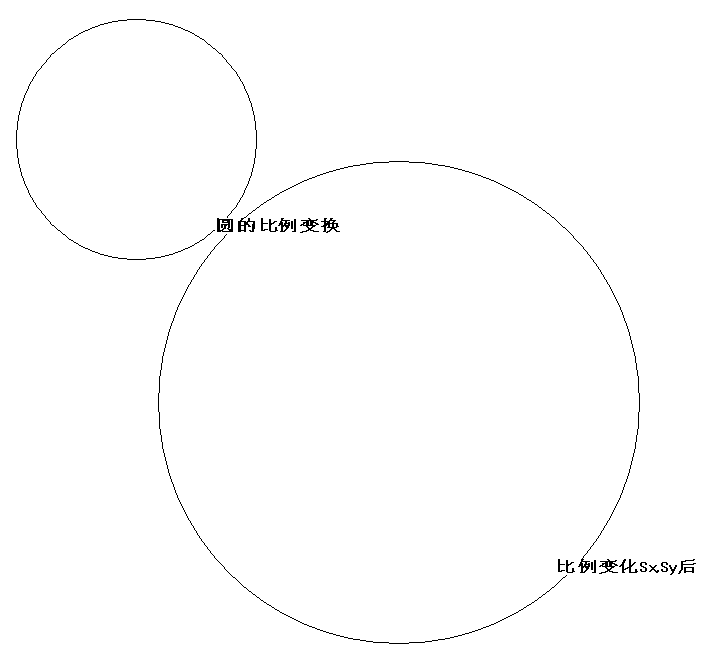
Ellipse(g\_hDc, x0, y0, x1, y1);

Ellipse(g\_hDc, x0\*Sx+300, y0\*Sy-300, x1\*Sx+300, y1\*Sy-300);

TextOut(g\_hDc, x0\*Sx, y0\*Sy, L"圆的比例变换", 6);

TextOut(g\_hDc, x0\*Sx\*Sx + 300, y0\*Sy\*Sy - 300, L"比例变化Sx,Sy后", 10);

}



//错切变换Y方向

void dislocatRect(int x0, int y0, int x1, int y1)

{ int dirY = 1;

Rectangle(g\_hDc, x0, y0, x1, y1);

int resultX[4] = { x0 + 300, x0 + 300, x1 + 300, x1 + 300 };

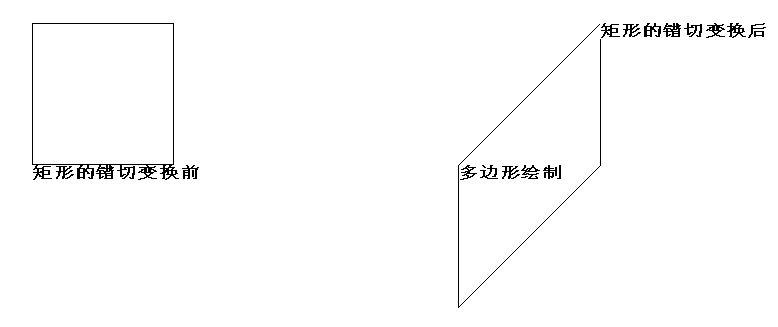
int resultY[4] = { y0, y0 + dirY\*x1, y1 + dirY\*x1, y1 };

polygon(resultX, resultY, 4);

TextOut(g\_hDc, x0, y0, L"矩形的错切变换前", 8);

TextOut(g\_hDc, x1 + 300, y1, L"矩形的错切变换后", 8);

}



//矩形的对称变换

void symmRect(int x0, int y0, int x1, int y1)

{

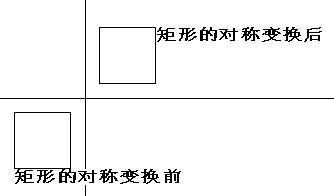
TextOut(g\_hDc, x0, y0, L"矩形的对称变换前", 8);

Rectangle(g\_hDc, x0, y0, x1, y1);

TextOut(g\_hDc, -x0, -y0, L"矩形的对称变换后", 8);

Rectangle(g\_hDc, -x0, -y0, -x1, -y1);

}



//旋转

void turnRect()

{

float st = -30 \* 3.14 / 180.0;

int arrX[4] = { -100, 100, 100, -100 };

int arrY[4] = { 100, 100, -100, -100 };

polygon(arrX, arrY, 4);

TextOut(g\_hDc, arrX[1], arrY[1], L"旋转前", 3);

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

{

arrX[i] = arrX[i] \* cos(st) - arrY[i] \* sin(st);

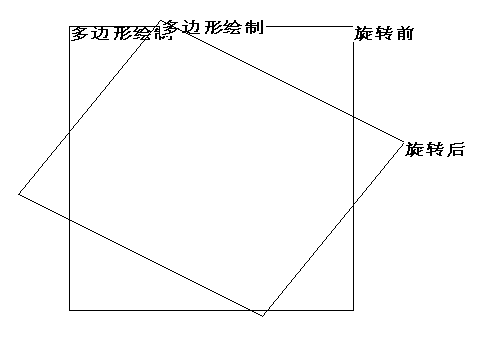
arrY[i] = arrX[i] \* sin(st) + arrY[i] \* cos(st);

}

polygon(arrX, arrY, 4);

TextOut(g\_hDc, arrX[1], arrY[1], L"旋转后", 3);

}



//复合平移

void compMove(int x0, int y0, int x1, int y1)

{

Rectangle(g\_hDc, x0, y0, x1, y1);

Rectangle(g\_hDc, x0, -y0, x1, -y1);

Rectangle(g\_hDc, -x0, -y0, -x1, -y1);

TextOut(g\_hDc, x0, y0, L"矩形的平移变换前", 8);

TextOut(g\_hDc, x0, -y0, L"矩形的平移变换1", 8);

TextOut(g\_hDc, -x0, -y0, L"矩形的平移变换2", 8);

}



//复合比例

void compRate(int x0, int y0, int x1, int y1)

{

float Sx = 0.5, Sy = 0.5;

float S2x = 2, S2y = 2;

Rectangle(g\_hDc, x0\*S2x, y0\*S2y, x1\*S2x, y1\*S2y);

Rectangle(g\_hDc, x0, y0, x1, y1);

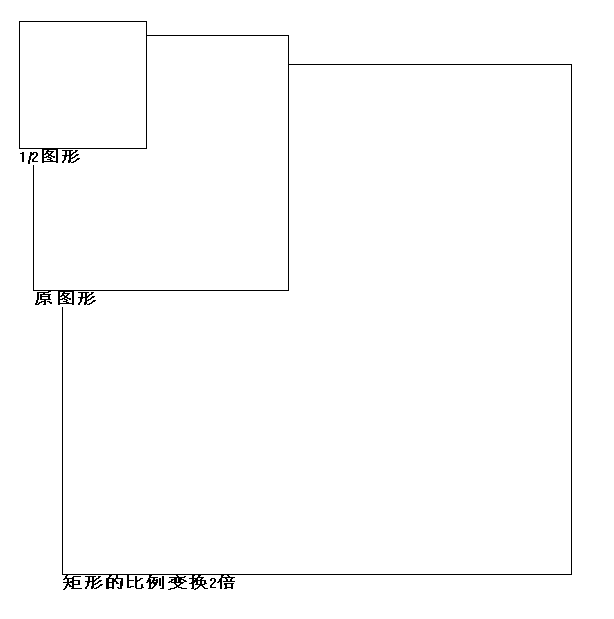
Rectangle(g\_hDc, x0\*Sx, y0\*Sy, x1\*Sx, y1\*Sy);

TextOut(g\_hDc, x0\*S2x, y1\*S2y, L"矩形的比例变换2倍", 9);

TextOut(g\_hDc, x0, y1, L"原图形", 3);

TextOut(g\_hDc, x0\*Sx, y1\*Sy, L"1/2图形", 5);

}



//复合旋转

void compTurn()

{

float st = -30 \* 3.14 / 180.0;

int arrX[4] = { -100, 100, 100, -100 };

int arrY[4] = { 100, 100, -100, -100 };

polygon(arrX, arrY, 4);

TextOut(g\_hDc, arrX[1], arrY[1], L"旋转前", 3);

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

{

arrX[i] = arrX[i] \* cos(st) - arrY[i] \* sin(st);

arrY[i] = arrX[i] \* sin(st) + arrY[i] \* cos(st);

}

polygon(arrX, arrY, 4);

TextOut(g\_hDc, arrX[1], arrY[1], L"旋转1后", 4);

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

{

arrX[i] = arrX[i] \* cos(st) - arrY[i] \* sin(st);

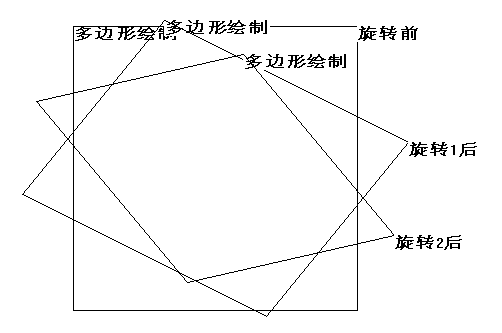
arrY[i] = arrX[i] \* sin(st) + arrY[i] \* cos(st);

}

polygon(arrX, arrY, 4);

TextOut(g\_hDc, arrX[1], arrY[1], L"旋转2后", 4);

}



//任意点旋转变换

void arbitPointRate(int x0, int y0, int x1, int y1, int r, int x, int y)

{ double k = (r\*3.1415926) / 180; int a, b;

MoveToEx(g\_hDc,0, 0,NULL);

LineTo(g\_hDc,x1, y1);

LineTo(g\_hDc,x0, y0);

LineTo(g\_hDc,0, 0);

x0 = x0 - x; y0 = y0 - y; x1 = x1 - x; y1 = y1 - y;

a = -x; b = -y;

x0 = x0\*cos(k) - y0\*sin(k); y0 = x0\*sin(k) + y0\*cos(k);//0

x1 = x1\*cos(k) - y1\*sin(k); y1 = x1\*sin(k) + y1\*cos(k);//1

a = a\*cos(k) - b\*sin(k); b = a\*sin(k) + b\*cos(k);

x0 = x0 + x; y0 = y0 + y; x1 = x1 + x; y1 = y1 + y;

a = a + x; b = b + y;

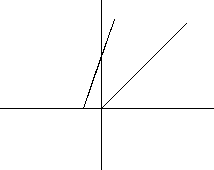
MoveToEx(g\_hDc, a, b,NULL);

LineTo(g\_hDc, x1, y1);

LineTo(g\_hDc, x0, y0);

LineTo(g\_hDc, a, b);

}



//任意线旋转变换

void arbitLineRate(int\* m)

{

int arrX[4], arrY[4];

arrX[0] = 1.5\*m[0] + 0.5\*m[3]; arrY[0] = 0.5\*m[0] + 1.5\*m[3];

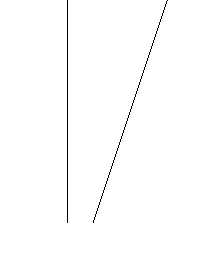
arrX[1] = 1.5\*m[0] + 0.5\*m[1]; arrY[1] = 0.5\*m[0] + 1.5\*m[1];

arrX[2] = 1.5\*m[2] + 0.5\*m[1]; arrY[2] = 0.5\*m[2] + 1.5\*m[1];

arrX[3] = 1.5\*m[2] + 0.5\*m[3]; arrY[3] = 0.5\*m[2] + 1.5\*m[3];

polygon(arrX,arrY,4);

}



//一次bezier曲线

void bezier1()

{

Pt3D p[2] = { { 80, 40, 0 }, { 10, 90, 0 } };

Pt3D pCurr;

TextOut(g\_hDc, p[0].x, p[0].y, L"一次bezier曲线", 10);

line(p[0].x, p[0].y, p[1].x, p[1].y);

for (float i = 0; i < 1; i += 0.001) //精度0.01(100个点)

{

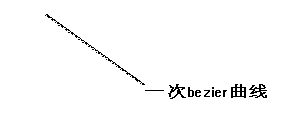
pCurr.x = (1 - i)\*(p[0].x) + i\*p[1].x;

pCurr.y = (1 - i)\*(p[0].y) + i\*p[1].y;

SetPixel(g\_hDc,pCurr.x, pCurr.y, RGB(0, 0, 0));

}

}



//二次bezier曲线

void bezier2()

{

Pt3D p[3] = { { 180, 40, 0 }, { 110, 90, 0 }, { 310, 90, 0 } };

Pt3D pCurr;

TextOut(g\_hDc, p[0].x, p[0].y, L"二次bezier曲线", 10);

line(p[0].x, p[0].y, p[1].x, p[1].y);

line(p[1].x, p[1].y, p[2].x, p[2].y);

for (float i = 0; i < 1; i += 0.001) //精度0.01(100个点)

{

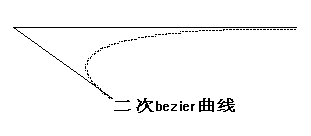
pCurr.x = (p[2].x - 2 \* p[1].x + p[0].x)\*i\*i + 2 \* (p[1].x - p[0].x)\*i + p[0].x;

pCurr.y = (p[2].y - 2 \* p[1].y + p[0].y)\*i\*i + 2 \* (p[1].y - p[0].y)\*i + p[0].y;

putpixel(pCurr.x, pCurr.y, RGB(0, 0, 0));

}

}



//三次bezier曲线

void bezier3()

{

Pt3D p[4] = { { 180, 150, 0 }, { 110, 200, 0 }, { 310, 200, 0 }, { 200, 340, 0 } };

Pt3D pCurr;

TextOut(g\_hDc, p[0].x, p[0].y, L"三次bezier曲线", 10);

line(p[0].x, p[0].y, p[1].x, p[1].y);

line(p[1].x, p[1].y, p[2].x, p[2].y);

line(p[2].x, p[2].y, p[3].x, p[3].y);

for (float i = 0; i < 1; i += 0.001) //精度0.01(100个点)

{

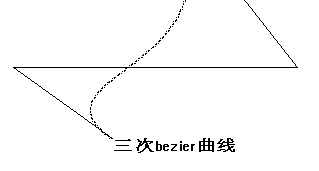
pCurr.x = pow(1 - i, 3)\*(p[0].x) + 3 \* i\*pow(1 - i, 2)\*p[1].x + 3 \* i\*i\*(1 - i)\*p[2].x + i\*i\*i\*p[3].x;

pCurr.y = pow(1 - i, 3)\*(p[0].y) + 3 \* i\*pow(1 - i, 2)\*p[1].y + 3 \* i\*i\*(1 - i)\*p[2].y + i\*i\*i\*p[3].y;

putpixel(pCurr.x, pCurr.y, RGB(0, 0, 0));

}

}



//三次B样条曲线

void thirdBCurve()

{

int p1[2] = { 20, 20 };

int p2[2] = { 20, -10 };

int p3[2] = { 10, -10 };

int pCurr[2];

TextOut(g\_hDc, p1[0], p1[1], L"三次B样条曲线", 7);

//line(p1[0], p1[1], p2[0], p2[1]);

//line(p2[0], p2[1], p3[0], p3[1]);

for (double i = 0; i < 1; i += 0.01)

{

pCurr[0] = (-pow(i, 3) + 3 \* i\*i - 3 \* i + 1)\*p1[0] + (3 \* pow(i, 3) - 6 \* i\*i + 4)\*p2[0]

+ (-3 \* pow(i, 3) \* 3 \* i\*i + 3 \* i + 1)\*p3[0];

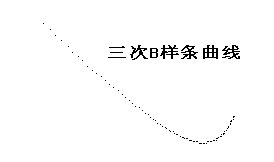
pCurr[1] = (-pow(i, 3) + 3 \* i\*i - 3 \* i + 1)\*p1[1] + (3 \* pow(i, 3) - 6 \* i\*i + 4)\*p2[1]

+ (-3 \* pow(i, 3) \* 3 \* i\*i + 3 \* i + 1)\*p3[1];

putpixel(pCurr[0], pCurr[1], RGB(0, 0, 0));

}

}



// Hermite三次样条曲线

void thirdHermite()

{ int p1[2] = { -200, 200 };int p2[2] = { 200, -300 };

int p3[2] = { 100, -200 };int pCurr[2]; float h[4];

TextOut(g\_hDc, p1[0], p1[1], L"三次Hermite样条曲线", 13);

for (float t = 0; t < 1; t += 0.01)

{ h[0] = 2 \* t\*t\*t - 3 \* t\*t + 1;

h[1] = -2 \* t\*t\*t + 3 \* t\*t;

h[2] = t\*t\*t - 2 \* t\*t + t;

h[3] = t\*t\*t - t\*t;

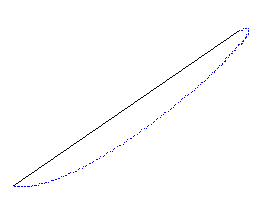
pCurr[0] = p1[0] \* h[0] + p1[0] \* h[1] + p3[0] \* h[2] + p3[0] \* h[3];

pCurr[1] = p1[1] \* h[0] + p1[1] \* h[1] + p3[1] \* h[2] + p3[1] \* h[3];

SetPixel(g\_hDc, pCurr[0], pCurr[1], RGB(255, 0, 0));

}

}



// 曲线的拼接设计

void compCurve()

{

CPoint P1, P2, P3, P4;

P1.x = -50; P1.y = 50;

P2.x = -100; P2.y = -200;

P3.x = 160; P3.y = 40;

MoveToEx(g\_hDc,P1.x,P1.y,0);

LineTo(g\_hDc,P2.x,P2.y);

LineTo(g\_hDc, P3.x,P3.y);

CPoint P;

float t; int i;

for (i = 0; i <= 400; i++)

{

t = (float)(i / 400.0);//400.0必须是小数

P.x = (P1.x\*pow(1 - t, 2) + 2 \* t\*(1 - t)\*P2.x + t\*t\*P3.x);

P.y = (P1.y\*pow(1 - t, 2) + 2 \* t\*(1 - t)\*P2.y + t\*t\*P3.y);

SetPixel(g\_hDc,P.x, P.y, RGB(255, 0, 255));

}

CPoint P5, P6, P7, P8;

P5.x = 160; P5.y = 40;

P6.x = 220; P6.y = 150;

P7.x = 290; P7.y = 10;

P8.x = 370, P8.y = 170;

MoveToEx(g\_hDc,P5.x,P5.y,0);

LineTo(g\_hDc,P6.x,P6.y);

LineTo(g\_hDc, P7.x, P7.y);

LineTo(g\_hDc, P8.x, P8.y);

for (i = 0; i <= 400; i++)

{

t = (float)(i / 400.0);//400.0必须是小数

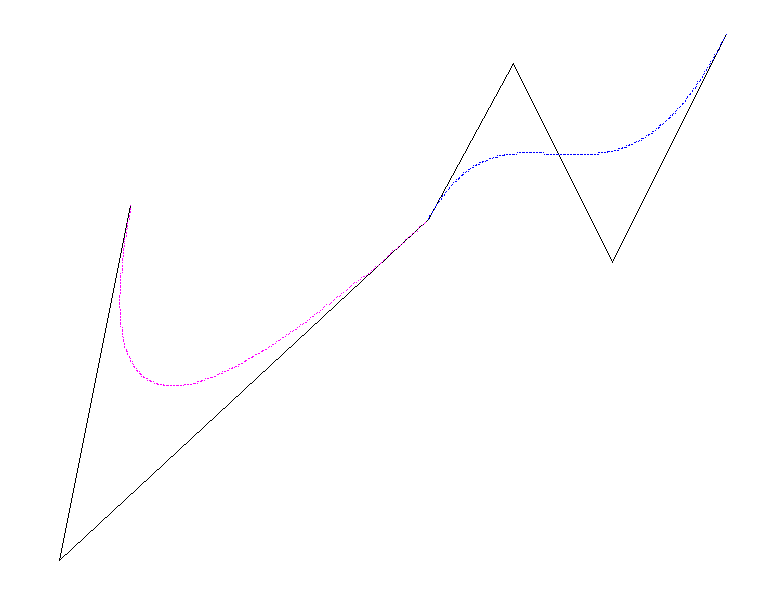
P.x = (int)(P5.x\*pow(1 - t, 3) + P6.x \* 3 \* t\*pow(1 - t, 2) + P7.x \* 3 \* t\*t\*(1 - t) + P8.x\*pow(t, 3));

P.y = (int)(P5.y\*pow(1 - t, 3) + P6.y \* 3 \* t\*pow(1 - t, 2) + P7.y \* 3 \* t\*t\*(1 - t) + P8.y\*pow(t, 3));

SetPixel(g\_hDc,P.x, P.y, RGB(0, 0, 255));

}

}



//图形的填充

void fillRect(int x0,int y0, int x1, int y1)

{

HBRUSH hBrush;

hBrush = (HBRUSH)GetStockObject(DKGRAY\_BRUSH);

CRect area = { x0, y0, x1, y1 };

FillRect(g\_hDc,&area, hBrush);

}

