

**课程名称** 计算机图形学

**学院** 计算机学院

**专业** 计算机科学与技术

**班级**  2014211306

**学号** 2014211292

**姓名** 张有杰

1. 实验目的

设计和实现一个图形函数库，具有绘制直线段、任意圆弧、椭圆弧、多边形区域的阴影填充和颜色填充等功能。（仅调用画点函数）

熟悉并实现基本的计算机图形学算法。

1. 实验环境

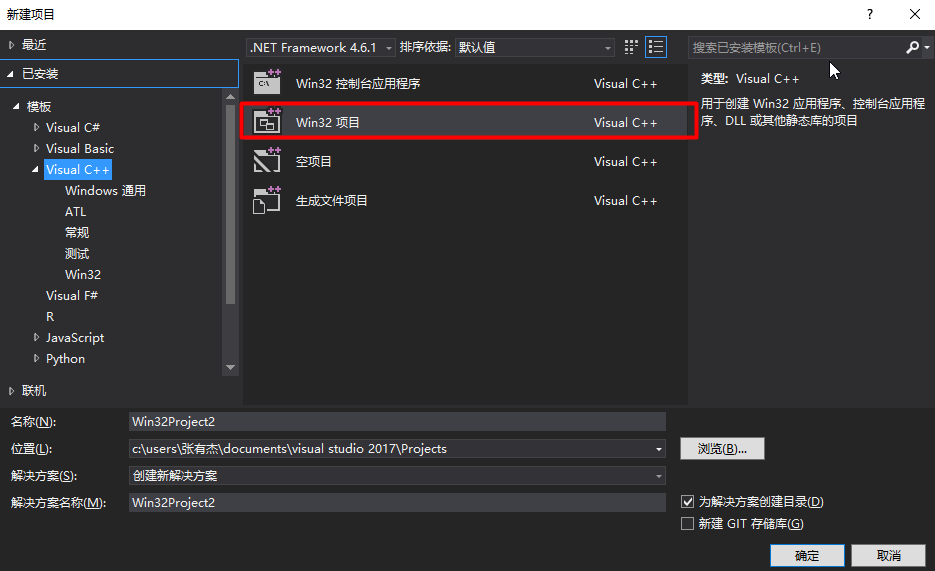
1. 操作系统：Windows 10

2. 开发环境：Visual Studio 2017

1. 实验内容
2. DDA算法画直线
3. Bresenham 算法画直线
4. 画任意圆弧
5. 画任意椭圆弧
6. 多边形区域的阴影填充
7. 多边形区域的颜色填充
8. 画出自己的名字
9. 实验步骤

0.实验准备

在 Visual Studio 2017 中新建项目时选中建立 Win32 项目，如下：



Visual Studio 会自动建立程序框架，我们只需要完成图形函数的编写并在窗口处理函数 WndProc 中调用我们需要的图形函数即可。

1.DDA算法画直线

void DDA(HDC hdc, int x0, int y0, int xn, int yn)

{

/\*确定增量dx,dy\*/

int max;

double dx, dy;

int d\_x = abs(xn - x0);

int d\_y = abs(yn - y0);

if (d\_x >= d\_y)

max = d\_x;

else

max = d\_y;

dx = (xn - x0)\*1.0 / max;

dy = (yn - y0)\*1.0 / max;

/\*绘点\*/

double xi = x0, yi = y0;

int pix\_x = floor(xi), pix\_y = floor(yi);

SetPixel(hdc, pix\_x, pix\_y, RGB(23, 24, 23));

for (int i = 0; i != max; ++i) {

xi += dx;

yi += dy;

pix\_x = floor(xi);

pix\_y = floor(yi);

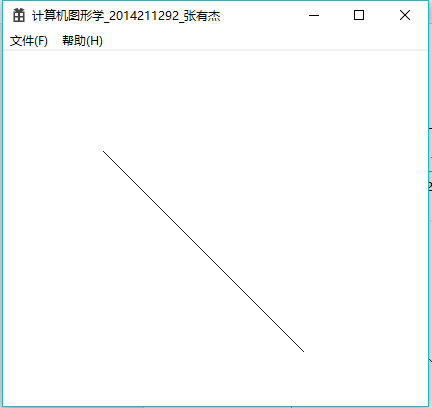
SetPixel(hdc, pix\_x, pix\_y, RGB(23, 24, 23));

}

}

其中 hdc 是设备上下文 Device Context 的句柄，(x0,y0)为起始点坐标，(xn,yn)为终止点坐标。

调用 DDA(hdc,100,100,300,300)，结果如下：



1. Bresenham 算法画直线

void Bresenham(HDC hdc, int x1, int y1, int x2, int y2)

{

int x, y, dx, dy, e;

dx = abs(x2 - x1);

dy = abs(y2 - y1);

e = 2 \* dy - dx;

x = x1;

y = y1;

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

SetPixel(hdc, x, y, RGB(0, 0, 0));

if (e >= 0) {

y++;

e = e - 2 \* dx;

}

x++;

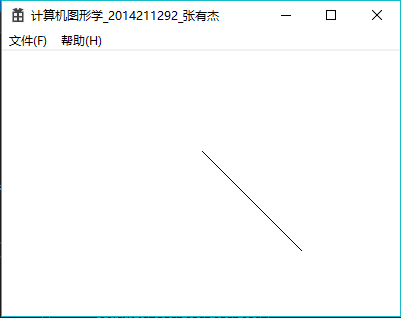
e = e + 2 \* dy;

}

}

其中 hdc 是设备上下文 Device Context 的句柄，(x1,y1)为起始点坐标，(x2,y2)为终止点坐标。

调用 Bresenham(hdc,200,100,100,200)，结果如下：



1. 画任意圆弧

void RoundArc(HDC hdc, int x, int y, int r)

{

int X, Y, d;

X = x;

Y = y - r;

d = (y - Y)\*(y - Y) + (Y + 1 - y)\*(Y + 1 - y) - 2 \* r \* r;

while ((X - x) <= (y - Y))

{

if (d >= 0)

{

Y++;

d = d + 4 \* (X + Y) - 4 \* (x + y) + 10;

}

else

{

d = d + 4 \* X - 4 \* x + 6;

}

SetPixel(hdc, X, Y, RGB(0, 0, 0));

SetPixel(hdc, (2 \* x - X), Y, RGB(0, 0, 0));

SetPixel(hdc, X, (2 \* y - Y), RGB(0, 0, 0));

SetPixel(hdc, (2 \* x - X), (2 \* y - Y), RGB(0, 0, 0));

SetPixel(hdc, (x + (y - Y)), (y - (X - x)), RGB(0, 0, 0));

SetPixel(hdc, (x + (y - Y)), (y + (X - x)), RGB(0, 0, 0));

SetPixel(hdc, (x - (y - Y)), (y - (X - x)), RGB(0, 0, 0));

SetPixel(hdc, (x - (y - Y)), (y + (X - x)), RGB(0, 0, 0));

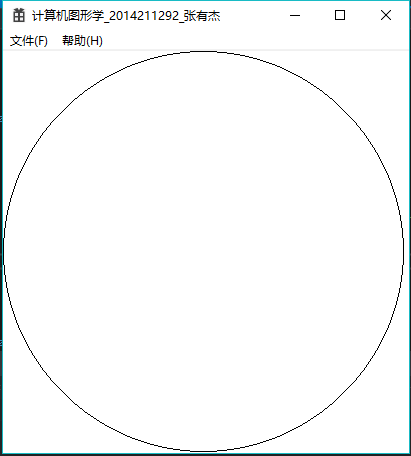
X++;

}

}

其中 hdc 是设备上下文 Device Context 的句柄，(x,y)为圆心坐标，r 为半径。

调用 RoundArc(hdc,200,200,200)结果如下：



1. 画任意椭圆弧

void EllipseArc(HDC hdc, int x, int y, int a, int b)

{

float t = 0, dt = 0.001;

int x1 = x + a, y1 = y, x2, y2;

while (t < 2 \* 3.14159) {

t += dt;

x2 = x + (int)(a \* cos(t));

y2 = y + (int)(b \* sin(t));

Bresenham(hdc, x1, y1, x2, y2);

x1 = x2;

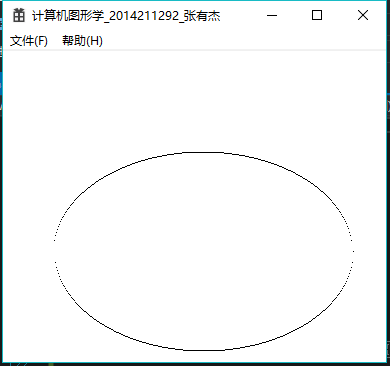
y1 = y2;

}

}

其中 hdc 是设备上下文 Device Context 的句柄，(x,y)为椭圆中心点坐标，a为长轴长，b为短轴长。

调用 EllipseArc (hdc,200,200,150,100)，结果如下：



1. 多边形区域的阴影填充

void shadowFill(HDC hdc, int P[][2], int mn, int m, double h, double a)

{

const double k = 1.0;

const double db = 3.0;

double B[8][2];

int i, j;

for (i = 0; (i + 1) <= (m - 1); ++i) {

B[i][0] = (double)(P[i][1] - k\*P[i][0]);

B[i][1] = (double)(P[i + 1][1] - k\*P[i + 1][0]);

if (B[i][0] > B[i][1]) {

double temp = B[i][0];

B[i][0] = B[i][1];

B[i][1] = temp;

}

}

B[m - 1][0] = (double)(P[m - 1][1] - k\*P[m - 1][0]);

B[m - 1][1] = (double)(P[0][1] - k\*P[0][0]);

if (B[m - 1][0] > B[m - 1][1]) {

double temp = B[m - 1][0];

B[m - 1][0] = B[m - 1][1];

B[m - 1][1] = temp;

}

for (i = m; (i + 1) <= (mn - 1); ++i) {

B[i][0] = (double)(P[i][1] - k\*P[i][0]);

B[i][1] = (double)(P[i + 1][1] - k\*P[i + 1][0]);

if (B[i][0] > B[i][1]) {

double temp = B[i][0];

B[i][0] = B[i][1];

B[i][1] = temp;

}

}

B[mn - 1][0] = (double)(P[mn - 1][1] - k\*P[mn - 1][0]);

B[mn - 1][1] = (double)(P[m][1] - k\*P[m][0]);

if (B[mn - 1][0] > B[mn - 1][1]) {

double temp = B[mn - 1][0];

B[mn - 1][0] = B[mn - 1][1];

B[mn - 1][1] = temp;

}

double Bmin = B[0][0], Bmax = B[0][1];

for (i = 1; i < mn; i++) {

if (B[i][0] < Bmin)

Bmin = B[i][0];

if (B[i][1] > Bmax)

Bmax = B[i][1];

}

double b = Bmin + db;

double D[8][2];

double xp, yp, xq, yq, x, y;

while (b < Bmax) {

for (i = 0; i <= mn - 1; ++i)

for (j = 0; j <= 1; ++j)

D[i][j] = 10000.0;

for (i = 0; (i + 1) <= (m - 1); ++i) {

if ((B[i][0] <= b) && (b < B[i][1])) {

xp = P[i][0];

yp = P[i][1];

xq = P[i + 1][0];

yq = P[i + 1][1];

x = (xp\*yq - yp\*xq + b\*(xq - xp)) / (yq - yp - k\*(xq - xp));

y = k\*x + b;

D[i][0] = x;

D[i][1] = y;

}

}

if ((B[m - 1][0] <= b) && (b < B[m - 1][1])) {

xp = P[m - 1][0];

yp = P[m - 1][1];

xq = P[0][0];

yq = P[0][1];

x = (xp\*yq - yp\*xq + b\*(xq - xp)) / (yq - yp - k\*(xq - xp));

y = k\*x + b;

D[i][0] = x;

D[i][1] = y;

}

for (i = m; (i + 1) <= (mn - 1); ++i) {

if ((B[i][0] <= b) && (b < B[i][1])) {

xp = P[i][0];

yp = P[i][1];

xq = P[i + 1][0];

yq = P[i + 1][1];

x = (xp\*yq - yp\*xq + b\*(xq - xp)) / (yq - yp - k\*(xq - xp));

y = k\*x + b;

D[i][0] = x;

D[i][1] = y;

}

}

if ((B[mn - 1][0] <= b) && (b < B[mn - 1][1])) {

xp = P[mn - 1][0];

yp = P[mn - 1][1];

xq = P[m][0];

yq = P[m][1];

x = (xp\*yq - yp\*xq + b\*(xq - xp)) / (yq - yp - k\*(xq - xp));

y = k\*x + b;

D[i][0] = x;

D[i][1] = y;

}

bubbleSort(D, 8);

for (int k = 0; (D[2 \* k][0] != 10000) && (D[2 \* k + 1][0] != 10000); k++) {

DDA(hdc, D[2 \* k][0], D[2 \* k][1], D[2 \* k + 1][0], D[2 \* k + 1][1]);

}

b = b + db;

}

}

其中 hdc 是设备上下文 Device Context 的句柄，p为多边形上的点的坐标，h 为阴影线的倾斜角，a 为阴影线的间距。

void DrawPolygon(HDC hdc, int p[][2], int mn, int m)

{

int i;

for (i = 0; (i + 1) <= (m - 1); ++i)

DDA(hdc, p[i][0], p[i][1], p[i + 1][0], p[i + 1][1]);

DDA(hdc, p[0][0], p[0][1], p[m - 1][0], p[m - 1][1]);

for (i = m; (i + 1) <= (mn - 1); ++i)

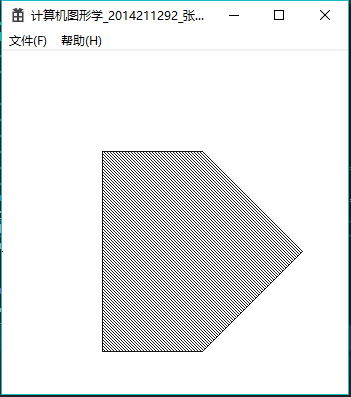
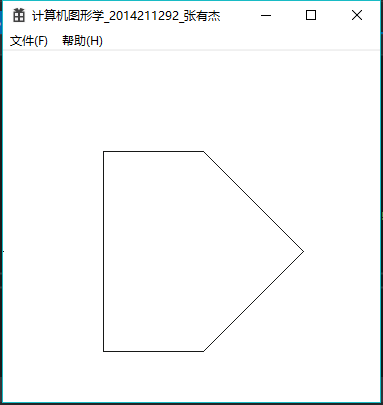
DDA(hdc, p[i][0], p[i][1], p[i + 1][0], p[i + 1][1]);

DDA(hdc, p[m][0], p[m][1], p[mn - 1][0], p[mn - 1][1]);

}

其中 hdc 是设备上下文 Device Context 的句柄，p为多边形上的点的坐标，mn 为多边形的总点数，m 为多边形的内部定点数。

执行结果如下：



1. 多边形区域的颜色填充

非递归：

void ColorFill(HDC hdc, int x, int y, COLORREF old\_color, COLORREF new\_color)

{

struct Pixel {

int x;

int y;

};

static std::list<Pixel> points;

points.push\_back(Pixel{ x,y });

while (points.empty() != true) {

Pixel tmp = points.front();

points.pop\_front();

x = tmp.x;

y = tmp.y;

if (GetPixel(hdc, x, y) == old\_color) {

SetPixel(hdc, x, y, new\_color);

points.push\_back(Pixel{ x + 1, y });

points.push\_back(Pixel{ x, y+1 });

points.push\_back(Pixel{ x, y-1 });

points.push\_back(Pixel{ x - 1, y });

}

}

}

递归：

void ColorFill(HDC hdc, int x, int y, COLORREF old\_color, COLORREF new\_color)

{

if (GetPixel(hdc, x, y) == old\_color) {

SetPixel(hdc, x, y, new\_color);

ColorFill(hdc, x + 1, y, old\_color, new\_color);

ColorFill(hdc, x, y + 1, old\_color, new\_color);

ColorFill(hdc, x, y - 1, old\_color, new\_color);

ColorFill(hdc, x - 1, y, old\_color, new\_color);

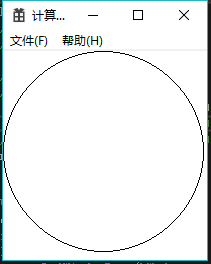
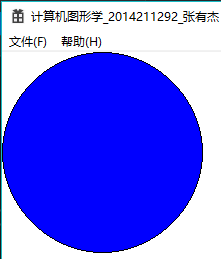
}

}

其中 hdc 是设备上下文 Device Context 的句柄，(x,y)为填充起始点坐标，old\_color 为旧颜色，new\_color 为新颜色。

由于对于较大的区域进行颜色填充时，使用递归的方法进行填充时会导致栈溢出。因此又使用链表完成了非递归的实现。

首先调用 RoundArc(hdc,100,100,100);在点(100,100)处画一个半径为 100 的圆形。然后调用函数 ColorFill(hdc, 50, 50, RGB(255, 255, 255), RGB(0, 0, 255));从点 50,50 开始将圆形内部颜色填充为蓝色。

 结果如下：

1. 画出自己的名字

void WriteMyName(HDC hdc) {

//写出“张有杰”

//“张”字左半部分

DDA(hdc, 100, 100, 200, 100);

DDA(hdc, 200, 100, 200, 200);

DDA(hdc, 200, 200, 100, 200);

DDA(hdc, 100, 200, 100, 250);

DDA(hdc, 100, 250, 200, 250);

DDA(hdc, 200, 250, 200, 350);

DDA(hdc, 125, 300, 200, 350);

//“张”字右半部分

DDA(hdc, 220, 100, 220, 350);

DDA(hdc, 220, 225, 320, 125);

DDA(hdc, 170, 225, 335, 225);

DDA(hdc, 220, 350, 280, 320);

DDA(hdc, 220, 225, 320, 330);

//“有”

DDA(hdc, 340, 140, 520, 140);

DDA(hdc, 430, 100, 340, 240);

DDA(hdc, 385, 175, 385, 350);

DDA(hdc, 385, 175, 480, 175);

DDA(hdc, 480, 175, 480, 350);

DDA(hdc, 385, 235, 480, 235);

DDA(hdc, 385, 270, 480, 270);

DDA(hdc, 480, 350, 430, 330);

//“杰字”

//“木”

DDA(hdc, 560, 150, 750, 150);

DDA(hdc, 655, 100, 655, 300);

DDA(hdc, 655, 150, 580, 280);

DDA(hdc, 655, 150, 730, 280);

//“四点”

DDA(hdc, 560, 330, 580, 320);

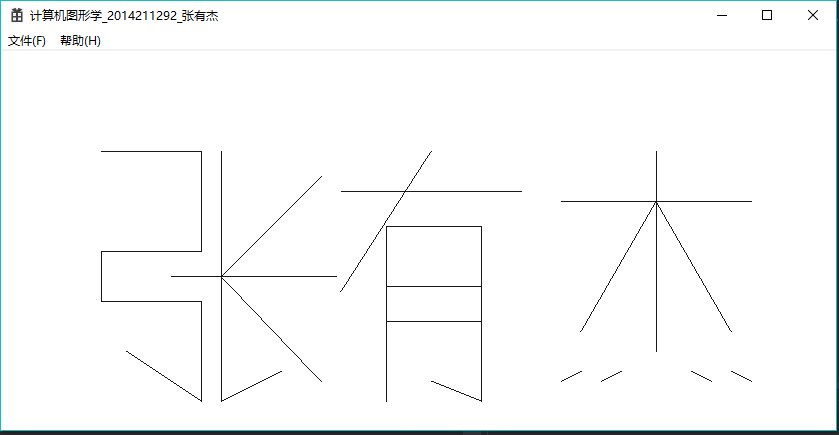
DDA(hdc, 600, 330, 620, 320);

DDA(hdc, 690, 320, 710, 330);

DDA(hdc, 730, 320, 750, 330);

}

其中 hdc 是设备上下文 Device Context 的句柄调用WriteMyName(hdc)，执行结果如下：



1. 实验总结

在此次实验中，我通过使用 Windows API 中的 SetPixel() 函数实现了所学过的几个基本的图形学算法，实现了画直线、圆弧、椭圆、多边形、多边形颜色填充及阴影填充等函数。并使用自己编写的函数写出了自己的名字。

通过此次实验，我对课堂上讲过的计算机图形学算法有了进一步的理解。通过查阅 MSDN 上关于 Windows API 的文档，我实现了几个算法，这锻炼了我的动手能力和解决问题的能力。