重慶理工大學 两江国际学院
CQUT-KAIST
Liangjiang International College

Computer Graphics Assignment Report

Assignment:	**Assignment #1 Basic Graph Scan Conversion**
Name:	**游忠敏**
ID#:	**11619370216**
Date of Experiment:	**2019.3.31**
Date of Report:	**2019.4.2**
Submission:	**2019.4.3**

I OBJECTIVES

Using familiar language to write the basic graph rasterization programs with midpoint and Bresenham's algorithm. The graph includes line segment, circle and ellipse.

II THEORETICAL BACKGROUND

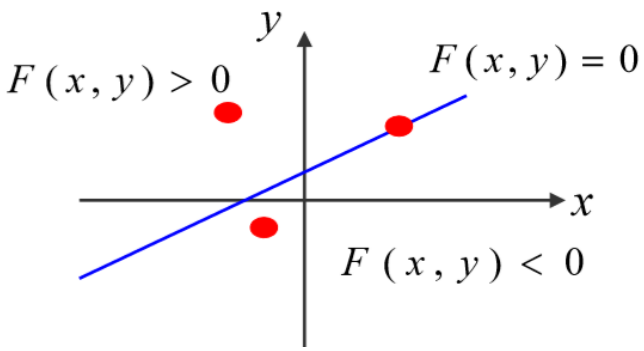
Midpoint algorithm:

The general equation of a straight line:

$$F(x, y) = 0$$

$$Ax + By + C = 0$$

And $A = -(\Delta y)$; $B = (\Delta x)$; $C = -B(\Delta x)$



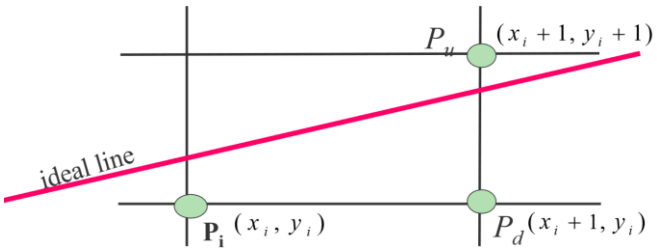
For the point on a line: F(x, y) = 0

For the point above a line: F(x, y) > 0

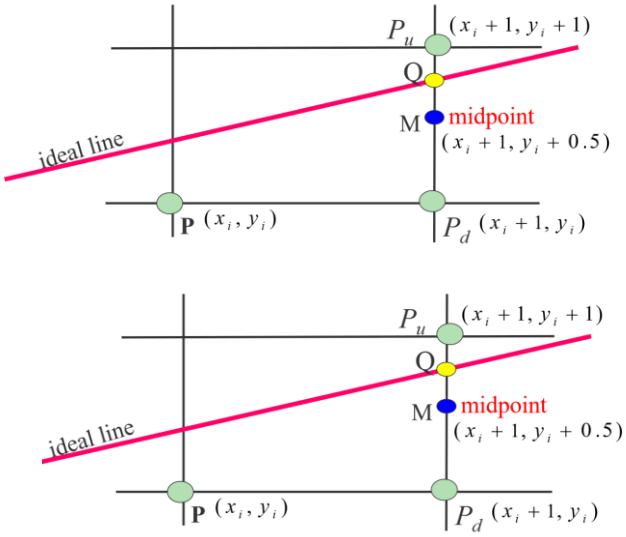
For the point below a line: F(x, y) < 0

Whether one step is taken in the direction of large displacement and the other is taken or not depends on the judgment of the midpoint error term.

Assumption: 0 < | K | < 1. Therefore, it needs to be judged to add 1 in the X direction and 1 in the Y direction at a time.



When M is below Q, P_u is close to the straight line. It should be the next pixel point. When M is above Q, P_d should be the next point.



How to judge whether Q is above or below M? Substitute M into the ideal linear equation:

$$F(x_m, y_m) = Ax_m + By_m + C$$

Next, we will analyze the calculation of the midpoint line drawing algorithm.

$$y = \begin{cases} y + 1 & (d < 0) \\ y & (d \geq 0) \end{cases}$$

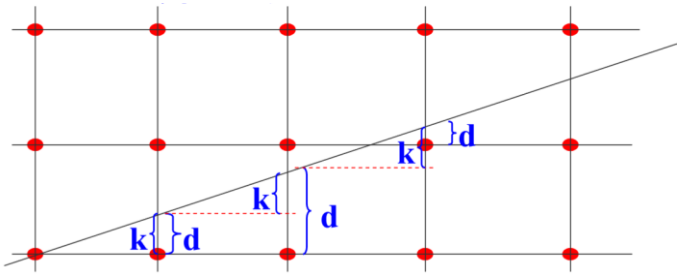
$$d_i = A(x_i + 1) + B(y_i + 0.5) + C$$

Finally, after a series of simplification operations:

$$d_{new} = \begin{cases} d_{old} + A + B & d < 0 \\ d_{old} + A & d \geq 0 \end{cases} \qquad d_0 = A + 0.5B$$

Bensenham's algorithm:

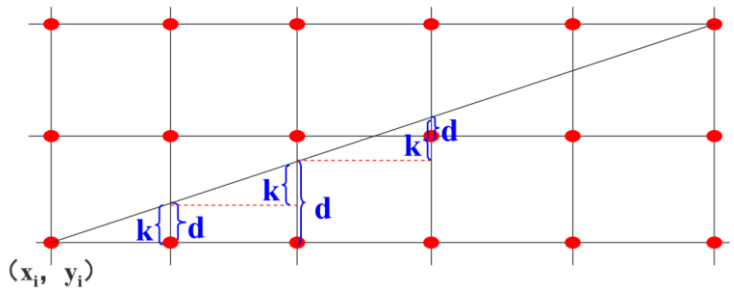
The basic idea of Bresenham's algorithm:



The idea of the algorithm is to construct a group of virtual grid lines by the center of each row and column of pixels. According to the order from the beginning to the end of the line, the intersection points of the line and the vertical grid lines are calculated, and then the pixels

close to the intersection points in the column pixels are determined according to the symbol of the error term.

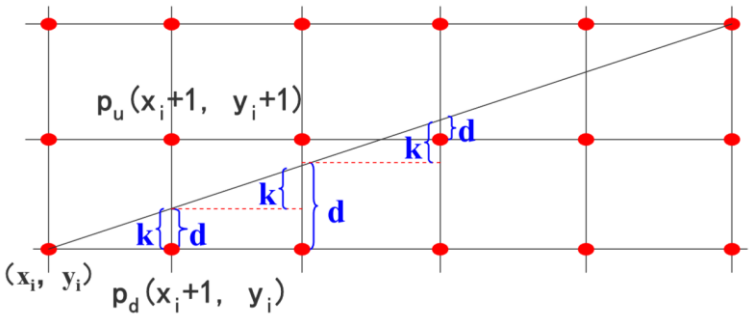
Assuming that the increment (subtraction) of Y is 0 or 1 for each x+1, it depends on the distance between the actual line and the grid point near the grating. The large error of the distance is 0.5.



Initial value d0 of error term d = 0

$$d = d + k,$$

once d is greater than or equal to 1, subtract it from 1 to ensure the relativity of d, and between 0 and 1.



$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i + 1 & (d > 0.5) \\ y_i & (d \leqslant 0.5) \end{cases} \end{cases}$$

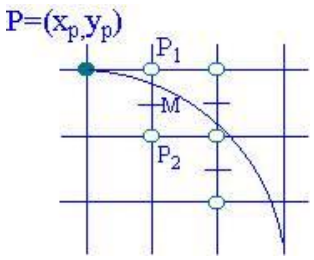
Ⅲ PROCEDURES

The algorithm of midpoint Line and Bresenham's line are showed in "background" of this report.

Midpoint circle:

Suppose the equation of the circle is: $x^2 + y^2 = r^2$

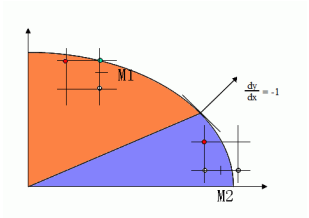
Function F (x, y) = x2 + y2-r2, then F(x, y)>0, corresponding to the outer point of the circle F (x, y) = 0, corresponding to the point on the circle F (x, y) < 0, corresponding to the inner point of the circle.



The core idea of the midpoint circle drawing method is to constantly judge whether the next midpoint is in the garden, if it is, draw a point to the right, if not, draw a point to the lower right corner.

```
void MidpointCircle(int x0, int y0, int r) // (x0, y0) 为圆心坐标， r为半径
{
    int x = x0, y = y0 + r, d, deltax, deltay;
    deltax = 3;
    deltay = 2 - r - r;
    d = 1 - r;
    set_mid(x, y);
    while (x < y)
    {
        if (d < 0)
        {
            d += deltax;
            deltax += 2;
            x++;
        }
        else
        {
            d += (deltax + deltay);
            deltax += 2;
            deltay += 2;
            x++;
            y--;
        }
        set_mid(x, y);
        set_mid(y, x);
        set_mid(x, y0 + y0 - y);
        set_mid(y0 + y0 - y, x);
        set_mid(x0 + x0 - x, y);
        set_mid(y, x0 + x0 - x);
        set_mid(x0 + x0 - x, y0 + y0 - y);
        set_mid(y0 + y0 - y, x0 + x0 - x);
    }
}
```

Midpoint Ellipse:

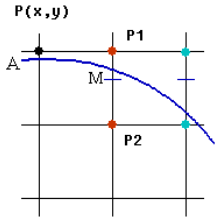


The tangent normal vector N defining a point on an elliptic arc is as follows:

$$N(x,y) = \frac{\partial F(x,y)}{\partial x} i + \frac{\partial F(x,y)}{\partial y} j$$
$$= 2b^2 xi + 2a^2 yj$$

For equation 1, the X and Y partial derivatives are obtained respectively. Finally, the normal vector at (x, y) points on the elliptic arc is ($2b^2x$, $2a^2y$). The point dy/dx = 1 is the boundary point on the elliptic arc. The part above this point (orange-brown part) has a larger y component, i.e. $2b^2 (x + 1) < 2a^2 (y-0.5)$; the part below this point (blue-purple part) has a larger x component, i.e. $2b^2 (x + 1) > 2a^2 (y-0.5)$.

For the upper part of the orange-brown mark in figure 1, the change of Y direction per unit is greater than that of X direction per unit, so the midpoint algorithm needs to step along the X direction to draw points, and x increments are added by 1 each time to find the value of Y. Similarly, for the lower part of the blue-purple mark in figure 1 the neutral point algorithm moves backwards along the Y direction, y minus 1 each time, and calculates the value of X. First, the generation of elliptic arcs in the upper region is discussed, as shown in the figure.

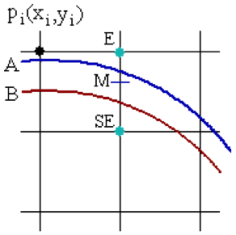



```
/*中点画椭圆*/
//上部区域生成算法的循环终止条件是：2b2(x + 1) >= 2a2(y - 0.5)，下部区域的循环终止条件是y = 0,
void MidpointEllipse(int x0, int y0, int a, int b)
{
    double sqa = a * a;
    double sqb = b * b;

    double d = sqb + sqa * (0.25 - b);
    int x = 0;
    int y = b;
    // 1
    set_mid((x0 + x), (y0 + y));
    // 2
    set_mid((x0 + x), (y0 - y));
    // 3
    set_mid((x0 - x), (y0 - y));
    // 4
    set_mid((x0 - x), (y0 + y));
    while (sqb*(x + 1) < sqa*(y - 0.5))
    {
        if (d < 0)
        {
            d += sqb * (2 * x + 3);
        }
        else
        {
            d += (sqb*(2 * x + 3) + sqa * ((-2)*y + 2));
            --y;
        }
        ++x;
        // 1
        set_mid((x0 + x), (y0 + y));
        // 2
        set_mid((x0 + x), (y0 - y));
        // 3
        set_mid((x0 - x), (y0 - y));
```

Bresenham circle:

Judge the position of point (x, y) according to f (x, y) symbol: assume that point PI (xi, yi) is the current point, and the next point is either point E (xi + 1, yi) or point SE (xi + 1, Yi - 1). Take two midpoints M (xi + 1, yi - 1/2), bring in F (x, y), and select the next given point according to the f (m) symbol.



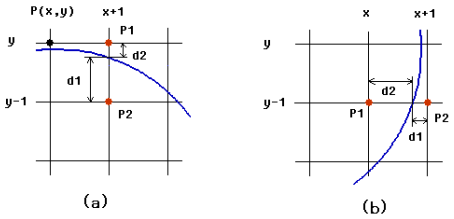

```
/*Bresenham圆*/
void BresenhamCircle(int cx, int cy, int r)
{
    int x = 0, y = r;
    int d = 1 - r;

    set_bre(cx, cy);

    while (x <= y)
    {
        set_bre(cx + x, cy + y);
        set_bre(cx - x, cy + y);
        set_bre(cx + x, cy - y);
        set_bre(cx - x, cy - y);
        set_bre(cx + y, cy + x);
        set_bre(cx - y, cy + x);
        set_bre(cx + y, cy - x);
        set_bre(cx - y, cy - x);
        if (d <= 0) {
            d += (x << 1) + 3;
        }
        else {
            d += ((x - y) << 1) + 5;
            y--;
        }
        x++;
    }
}
```

Bresenham ellipse:

When the upper region of an ellipse is generated, the x-axis is taken as the stepping direction, as shown in the figure.



For each step of x, it is necessary to determine whether y remains unchanged or decreases by 1. The Bresenham algorithm defines the discriminant as follows:

$$D = d1 – d2$$

If D < 0, take P1 as the next point, otherwise, take P2 as the next point. By using discriminant D, the floating-point arithmetic introduced by the midpoint arithmetic due to y-0.5 is avoided, so that the discriminant is reduced to full integer arithmetic, and the efficiency of the arithmetic is greatly improved. According to the elliptic equation, D1 and D2 can be calculated as follows:

$$d_1 = a^2(y_i^2 - y^2)$$

$$d_2 = a^2(y^2 - y_{i+1}^2)$$

Taking (0, b) as the starting point of the upper region of an ellipse and substituting it into discriminant D, the following recursive relations can be obtained:

$$D_{i+1} = D_i + 2b^2(2x_i + 3) \qquad (Di < 0)$$

$$D_{i+1} = D_i + 2b^2(2x_i + 3) - 4a^2(y_i - 1) \qquad (Di >= 0)$$

$$D_0 = 2b^2 - 2a^2b + a^2$$

When the lower part of an ellipse is generated, the Y axis is taken as the stepping direction. As shown in Fig. 5-b, if y is reduced b y one unit per step, it is necessary to determine whether x remains unchanged or whether a unit is added step b y step. For the lower part, D1 and D2 are calculated as follows:

$$d_1 = b^2(x_{i+1}^2 - x^2)$$

$$d_2 = b^2(x^2 - x_i^2)$$

Taking (xp, yp) as the starting point of the lower region of an ellipse and substituting it into discriminant D, the following recursive relations can be obtained:

$$D_{i+1} = D_i - 4a^2(y_i - 1) + 2a^2 \qquad (Di < 0)$$

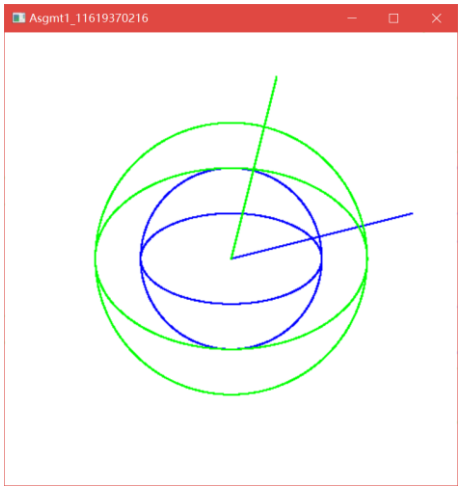
$$D_{i+1} = D_i + 2b^2(x_i + 1) - 4a^2(y - 1) + 2a^2 + b^2 \qquad (Di >= 0)$$

$$D_0 = b^2(x_p + 1)^2 + b^2x_p^2 - 2a^2b^2 + 2a^2(y_p - 1)^2$$

```
/*Bresenham画椭圆*/
void BresenhamEllipse(int x0, int y0, int a, int b)
{
    int x, y;
    float d1, d2, aa, bb;
    aa = a * a;
    bb = b * b;
    d1 = bb + aa * (-b + 0.25);
    glTranslatef((GLfloat)x0, (GLfloat)y0, 0.0f);
    x = 0;
    y = b;
    set_bre(x, y);
    set_bre(-x, y);
    set_bre(-x, -y);
    set_bre(x, -y);
    while (bb * (x + 1) < aa * (y - 0.5))
    {
        if (d1 <= -0.000001)
        {
            d1 += bb * ((x << 1) + 3);

        }
        else
        {
            d1 += bb * ((x << 1) + 3) + aa * (2 - (y << 1));
            --y;
        }
```

IV RESULTS



The blue line used midpoint algorithm, and the green line used Bresenham's algorithm.

V DISCUSSION

Bresenham provides a more general algorithm. This algorithm not only has good efficiency, but also has a wider range of applications.

VI CONCLUSIONS

At first, the difference between Bresenham's algorithm and midpoint algorithm was not clear. Finally, the answer is found: midpoint algorithm is a generalization of Bresenham's algorithm. The reasoning angles are different (although iterations are used), but the final results are the same.

APPENDIX: C/JAVA/Python code

```
#include<GL/glut.h>
```

```
#include<iostream>
#include <stdio.h>
#include<math.h>

using namespace std;

void set_mid(int x, int y) //将用中点画线法的图用蓝色线表示
{
    glColor3f(0.0, 0.0, 1.0);
    glPointSize(2.0f);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}

void set_bre(int x, int y) //将用Bresenham画线法的图用绿色线表示
{
    glColor3f(0.0, 1.0, 0.0);
    glPointSize(2.0f);
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}
/*中点画直线*/
void MidpointLine(int x0, int y0, int x1, int y1)
{
    /*
    用2d代替d的中点画线法的算法步骤（0≤k≤1 ）
    1. 输入直线段的两个端点P1(x1, y1)和P2(x2, y2)。
    2. 初始化：a，b，d=2a+b，x=x1，y=y1，画点(x, y)。
    3. 若x<x2，则执行下列各步，否则算法结束。
    4. 判断d的符号：若d<0，则(x, y)更新为(x+1, y+1)，d更新为d+2a+2b；否则(x, y)更新为(x+1, y)，d更新为d+2a。
    5. 画点(x, y)，返回3。
    */
    if ((x0 != x1) && (y0 != y1))
    {
        int a, b, deltal, delta2, d, x, y;
        a = y0 - y1;
        b = x1 - x0;
        d = 2 * a + b;
        deltal = 2 * a;
```

```
            delta2 = 2 * (a + b);
            x = x0;
            y = y0;
            set_mid(x, y);
            while (x < x1)
            {
                if (d < 0)
                {
                    x++;
                    y++;
                    d += delta2;//when d<0, y=y+1
                }
                else
                {
                    x++;
                    d += delta1;//when d>=0, y=y
                }
                set_mid(x, y);
            }
        }
        else
        {
            int d;
            if (x0 == x1)
            {
                int x = x0, y;
                y = (y0 <= y1) ? y0 : y1;
                d = fabs((double)(y0 - y1));
                while (d <= 0)
                {
                    set_mid(x, y);
                    y++;
                    d--;
                }
            }
        }
    }

    /*中点画圆*/
    void MidpointCircle(int x0, int y0, int r) //（x0, y0）为圆心坐标， r为半径
    {
        int x = x0, y = y0 + r, d, deltax, deltay;
        deltax = 3;
        deltay = 2 - r - r;
```

```
    d = 1 - r;
    set_mid(x, y);
    while (x < y)
    {
        if (d < 0)
        {
            d += deltax;
            deltax += 2;
            x++;
        }
        else
        {
            d += (deltax + deltay);
            deltax += 2;
            deltay += 2;
            x++;
            y--;
        }
        set_mid(x, y);
        set_mid(y, x);
        set_mid(x, y0 + y0 - y);
        set_mid(y0 + y0 - y, x);
        set_mid(x0 + x0 - x, y);
        set_mid(y, x0 + x0 - x);
        set_mid(x0 + x0 - x, y0 + y0 - y);
        set_mid(y0 + y0 - y, x0 + x0 - x);
    }

}

/*中点画椭圆*/
//上部区域生成算法的循环终止条件是：2b2(x + 1) >= 2a2(y  -  0.5)，下部区域的循环终止条件是y = 0，
void MidpointEllipse(int x0, int y0, int a, int b)
{
    double sqa = a * a;
    double sqb = b * b;

    double d = sqb + sqa * (0.25 - b);
    int x = 0;
    int y = b;
    // 1
    set_mid((x0 + x), (y0 + y));
    // 2
    set_mid((x0 + x), (y0 - y));
```

```
// 3
set_mid((x0 - x), (y0 - y));
// 4
set_mid((x0 - x), (y0 + y));
while (sqb*(x + 1) < sqa*(y - 0.5))
{
    if (d < 0)
    {
        d += sqb * (2 * x + 3);
    }
    else
    {
        d += (sqb*(2 * x + 3) + sqa * ((-2)*y + 2));
        --y;
    }
    ++x;
    // 1
    set_mid((x0 + x), (y0 + y));
    // 2
    set_mid((x0 + x), (y0 - y));
    // 3
    set_mid((x0 - x), (y0 - y));
    // 4
    set_mid((x0 - x), (y0 + y));
}
d = (b * (x + 0.5)) * 2 + (a * (y - 1)) * 2 - (a * b) * 2;
// 2
while (y > 0)
{
    if (d < 0)
    {
        d += sqb * (2 * x + 2) + sqa * ((-2) * y + 3);
        ++x;
    }
    else
    {
        d += sqa * ((-2) * y + 3);
    }
    --y;
    // 1
    set_mid((x0 + x), (y0 + y));
    // 2
    set_mid((x0 + x), (y0 - y));
    // 3
```

```
            set_mid((x0 - x), (y0 - y));
            // 4
            set_mid((x0 - x), (y0 + y));
        }
    }

    /*Bresenham画直线*/
    void BresenhamLine(int x1, int y1, int x2, int y2)
    {
        int dx, dy, sx, sy, x, y, a;
        dx = x2 - x1;
        dy = y2 - y1;
        sx = (dx >= 0) ? 1 : (-1);
        sy = (dy >= 0) ? 1 : (-1);
        x = x1;
        y = y1;
        a = 0;

        if (abs(dy) > abs(dx))
        {
            float tdx = dx;
            dx = dy;
            dy = tdx;

            a = 1;
        }

        float p = 2 * (abs(dy)) - abs(dx);

        set_bre(x, y);

        for (int i = 0; i <= abs(dx);i++)
        {
            if (p < 0)
            {
                if (a == 0) {
                    x = x + sx;
                    set_bre(x, y);
                }
                else {
                    y = y + sy;
                    set_bre(x, y);
                }
                p = p + 2 * abs(dy);
```

```
        }
        else
        {
            x = x + sx;
            y = y + sy;
            set_bre(x, y);
            p = p + 2 * abs(dy) - 2 * abs(dx);
        }
    }
}

/*Bresenham画圆*/
void BresenhamCircle(int cx, int cy, int r)
{
    int x = 0, y = r;
    int d = 1 - r;

    set_bre(cx, cy);

    while (x <= y)
    {
        set_bre(cx + x, cy + y);
        set_bre(cx - x, cy + y);
        set_bre(cx + x, cy - y);
        set_bre(cx - x, cy - y);
        set_bre(cx + y, cy + x);
        set_bre(cx - y, cy + x);
        set_bre(cx + y, cy - x);
        set_bre(cx - y, cy - x);
        if (d <= 0) {
            d += (x << 1) + 3;
        }
        else {
            d += ((x - y) << 1) + 5;
            y--;
        }
        x++;
    }
}

/*Bresenham画椭圆*/
void BresenhamEllipse(int x0, int y0, int a, int b)
{
    int x, y;
```

```
float d1, d2, aa, bb;
aa = a * a;
bb = b * b;
d1 = bb + aa * (-b + 0.25);
glTranslatef((GLfloat)x0, (GLfloat)y0, 0.0f);
x = 0;
y = b;
set_bre(x, y);
set_bre(-x, y);
set_bre(-x, -y);
set_bre(x, -y);
while (bb * (x + 1) < aa * (y - 0.5))
{
    if (d1 <= -0.000001)
    {
        d1 += bb * ((x << 1) + 3);

    }
    else
    {
        d1 += bb * ((x << 1) + 3) + aa * (2 - (y << 1));
        --y;
    }
    ++x;
    set_bre(x, y);
    set_bre(-x, y);
    set_bre(-x, -y);
    set_bre(x, -y);
}
d2 = bb * (0.25 * x) + aa * (1 - (y << 1));
while (y > 0)
{
    if (d2 <= -0.000001)
    {
        ++x;
        d2 += bb * ((x + 1) << 1) + aa * (3 - (y << 1));
    }
    else
    {
        d2 += aa * (3 - (y << 1));
    }
    --y;
    set_bre(x, y);
    set_bre(-x, -y);
```

```
            set_bre(-x, y);
            set_bre(x, -y);
        }
    }


void Display()
{
    glClear(GL_COLOR_BUFFER_BIT);

    MidpointLine(0, 0, 400, 100);//Midpoint直线
    MidpointCircle(0, 0, 200);//Midpoint圆
    MidpointEllipse(0, 0, 200, 100);//Mid椭圆
    BresenhamLine(0, 0, 100, 400);//Bresenham直线
    BresenhamCircle(0, 0, 300);//Bresenham圆
    BresenhamEllipse(0, 0, 300, 200);//Bresenham圆

    glEnd();
    glFlush();
}

int main(int argc, char ** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(300, 100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Asgmt1_11619370216");
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-500.0, 500.0, -500.0, 500.0);//坐标轴范围：横坐标（-500，500），纵坐标（-500，500）
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```