Rasterization

Rui Ying

Line rasterization

Given conditions `x1 < x2, |y2-y1| < |x2-x1|, y1 < y2`, it's easy to implement line drawing following the pseudo code shown in lecture slides.

Other situations may require some tricks.

x1 > x2

In this situation, we can simply swap (x1,y1) with (x2,y2).

```
int dx, dy, D, inc0, inc1;
dx = x2 - x1;
dy = y2 - y1;
D = 2 * dy - dx;
inc0 = 2 * dy;
inc1 = 2 * (dy - dx);
putPixel(x1, y1);

while (x1 < x2) {
    if (D <= 0) {
        D += inc0;
    }
    else {
        D += inc1;
        y1 += 1;
    }
    x1 += 1;
    putPixel(x1, y1);
}
```

y1 < y2 && |y2-y1| > |x2-x1|

In this situation, we can observe that this line(|y2-y1| > |x2-x1|) and the line(|y2-y1| < |x2-x1|) are symmetric with regard to the 45 degree line starting from (x1,y1).

So we first swap both x's with y's so that the line drawing is now the same as the situation above.

But in order to get the final result, we need to change back the coordinates where `putPixel` draw the pixel.

```
std::swap(x1, y1);
std::swap(x2, y2);
int dx, dy, D, inc0, inc1;
dx = x2 - x1;
```

```
dy = y2 - y1;
D = 2 * dy - dx;
inc0 = 2 * dy;
inc1 = 2 * (dy - dx);
putPixel(y1, x1);

while (x1 < x2) {
    if (D <= 0) {
        D += inc0;
    }
    else {
        D += inc1;
        y1 += 1;
    }
    x1 += 1;
    putPixel(y1, x1);
}
```

y1 > y2 && |y2-y1| < |x2-x1|

In this situation, we can see the line(y1 > y2 && |y2-y1| < |x2-x1|) and the line(y1 < y2 && |y2-y1| < |x2-x1|) are symmetric with the line `y = y1`.

```
y2 = 2 * y1 - y2;
int tmpY1 = y1;
int dx, dy, D, inc0, inc1;
dx = x2 - x1;
dy = y2 - y1;
D = 2 * dy - dx;
inc0 = 2 * dy;
inc1 = 2 * (dy - dx);
putPixel(x1, 2 * tmpY1 - y1);

while (x1 < x2) {
    if (D <= 0) {
        D += inc0;
    }
    else {
        D += inc1;
        y1 += 1;
    }
    x1 += 1;
    putPixel(x1, 2 * tmpY1 - y1);
}
```

y1 > y2 && |y2-y1| > |x2-x1|

Using the same method to transform the line to the situations above solves the problem.

```
y2 = 2 * y1 - y2;
int tmpY1 = y1;
std::swap(x1, y1);
std::swap(x2, y2);
int dx, dy, D, inc0, inc1;
dx = x2 - x1;
dy = y2 - y1;
D = 2 * dy - dx;
inc0 = 2 * dy;
inc1 = 2 * (dy - dx);
putPixel(y1, 2 * tmpY1 - x1);

while (x1 < x2) {
    if (D <= 0) {
        D += inc0;
    }
    else {
        D += inc1;
        y1 += 1;
    }
    x1 += 1;
    putPixel(y1, 2 * tmpY1 - x1);
}
```

Circle rasterization

Follow the pseudo code and it is done.

Except that we need to change `circlePoints` a little bit to `circlePoints(int x, int y, int x0, int y0)` where `(x0,y0)` is the circle center.

```
int x = 0, y = R, D = 1 - R;
circlePoints(x, y, x0, y0);
while (y > x) {
    if (D < 0) {
        D += 2 * x + 3;
    }
    else {
        D += 2 * (x - y) + 5;
        y -= 1;
    }
    x += 1;
    circlePoints(x, y, x0, y0);
}
```

Result