Kapitel 3

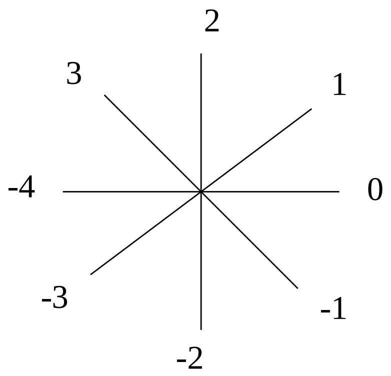
2D-Grundlagen

3.1 Koordinatensysteme

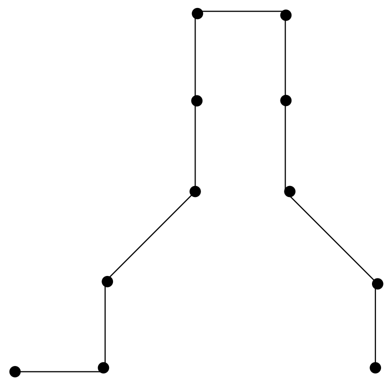
Meistens: Angabe eines Punktes durch kartesische Koordinaten x, y.
Eine Folge von Punkten kann

- absolut oder
- relativ spezifiziert werden.

Absolut bedeutet:	Jeder Punkt der Folge wird durch seine kartesischen Koordinaten beschrieben.
Relativ bedeutet:	Jeder Punkt der Folge wird unter Verwendung der Windrose relativ zum Vorgänger der Folge beschrieben, entweder mit absoluten Richtungscodes oder mit relativen Richtungscodes.



Beispiel: Gegeben sei folgende Figur:



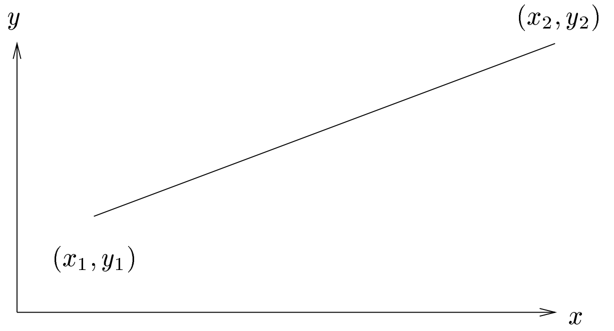
Absoluter Richtungscode ergibt für die Figur die Folge $0, 2, 1, 2, 2, 0, -2, -2, -1 - 2$.
Relativer Richtungscode legt nach jedem Schritt die momentane Richtung als "0" fest. Es ergibt sich für Bild 2 die Folge $0, 2, -1, 1, 0, -2, -2, 0, 1, -1$.
Zur Platzeinsparung werden die "wahrscheinlicheren" Richtungsänderungen mit weniger Bits codiert:

Richtung	Code	Länge
0	00	2
1	10	2
-1	01	2
2	1100	4
-2	1101	4
3	111000	6
-3	111001	6
-4	111011	6

Für Bild 2 entstehen statt $(10 \times 3) = 30$ Bits nur noch 26 Bits.

3.2 Linie

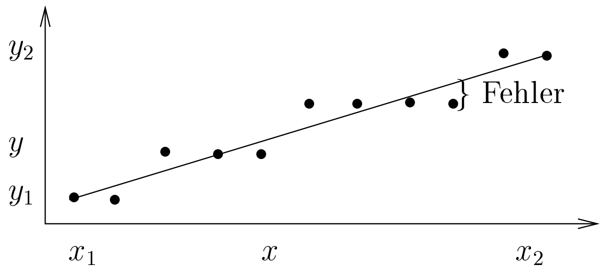
Problem: Gegeben sind Anfangs- und Endkoordinaten einer Linie. Berechne die "anzuschaltenden" Pixel.



1. Möglichkeit: Bestimme Geradengleichung $y = s \cdot x + c$
Es gilt:

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1} \Rightarrow$$

$$y = \frac{y_2 - y_1}{x_2 - x_1} \cdot x + \frac{y_1 \cdot x_2 - y_2 \cdot x_1}{x_2 - x_1}$$



Daraus ergibt sich folgender Algorithmus unter Verwendung der Klasse `java.awt.Point` mit den beiden Integer-Koordinaten `x` und `y`:

```
void plot_line(Point P, Point q)
{
        double s, c;
        Point p = new Point(P);

        s = (double) (q.y-p.y) / (double) (q.x-p.x);
        c = (double) (p.y*q.x - q.y*p.x) / (double) (q.x - p.x );

        while (p.x <= q.x) {
                p.y = (int)(s*p.x + c + 0.5);
                set_pixel(p);
                p.x++;
        }
}
```

Verfahren hat 2 Nachteile:

1. Pixel für steile Geraden sind nicht benachbart.
2. viel Gleitkomma-Arithmetik

Lösung:	Bresenham-Algorithmus
Idee:	Berechne den nächsten y-Wert aus dem vorherigen. Merke den momentanen Fehler.
Zunächst:	Gerade verläuft im 1. Oktanten (d.h. Steigung < 1)

```
void bresenham_linie_1(Point P, Point q)
{
        int dx, dy;
        double s, e;
        Point p = new Point(P);

        dx    = q.x - p.x;
        dy    = q.y - p.y;
        e     = 0.0;
        s     = (double) dy / (double) dx;

        while (p.x <= q.x) {
                set_pixel(p);
                p.x++;
                e += s;
                if (e > 0.5) { p.y++; e-- ; }
        }
}
```

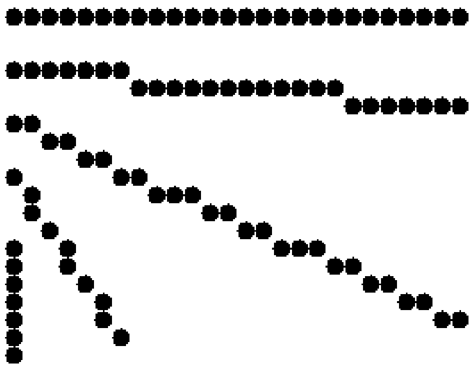
Eliminiere Gleitkommazahlen durch Multiplikation von `s` mit `2*dx`:

```
void bresenham_linie_2(Point P, Point q)
{
        int dx, dy, error, delta;
        Point p = new Point(P);

        dx    = q.x - p.x;
        dy    = q.y - p.y;
        error = 0;
        delta = 2*dy;

        while (p.x <= q.x) {
                set_pixel(p);
                p.x++;
                error += delta;
                if (error > dx) { p.y++; error -= 2*dx; }

        }
}
```

Vergleiche `error` mit 0 und verwende Abkürzung `schwelle` für `-2*dx`

```
void bresenham_linie_3(Point P, Point q)
{
        int dx, dy, error, delta, schwelle;
        Point p = new Point(P);

        dx       = q.x - p.x;
        dy       = q.y - p.y;
        error    = -dx;
        delta    = 2*dy;
        schwelle = -2*dx;

        while (p.x <= q.x) {
                set_pixel(p);
                p.x++;
                error += delta;
                if (error > 0) { p.y++; error += schwelle; }
        }
}
```

Geraden in den anderen 7 Oktanten müssen durch Spiegelung und/oder Vertauschen von x und y auf den 1. Oktanten zurückgeführt werden.

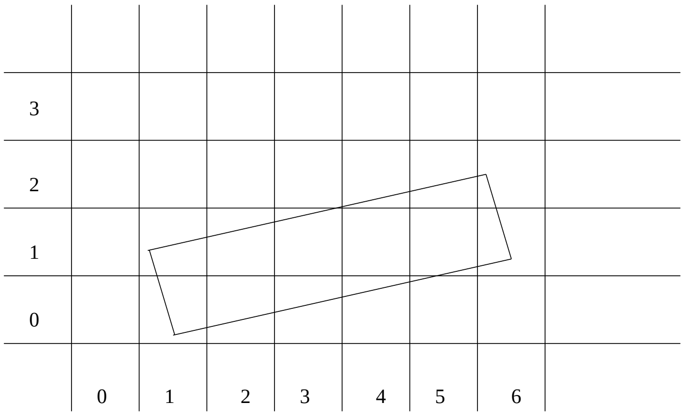


Vom Bresenham-Algorithmus erzeugte Linien

```
/*************************************************************************************/
/*                                                                                   */
/*                   Bresenham-Algorithmus zum Zeichnen einer Linie                  */
/*                                                                                   */
/*************************************************************************************/

private void bresenham_linie(                   // zeichnet mit Bresenham-Algorithmus
        Point P, Point q)                       // eine Linie von Punkt P nach Punkt q
{
        Point p = new Point(P);

        int error, delta, schwelle, dx, dy, inc_x, inc_y;
        dx = q.x - p.x;
        dy = q.y - p.y;

        if (dx>0) inc_x= 1; else inc_x=-1;
        if (dy>0) inc_y= 1; else inc_y=-1;

        if (Math.abs(dy) < Math.abs(dx)) {      // flach nach oben oder flach nach unten

                error = -Math.abs(dx);
                delta = 2*Math.abs(dy);
                schwelle = 2*error;
                while (p.x != q.x) {
                        set_pixel(p);
                        p.x+=inc_x;
                        error = error + delta;
                        if (error >0) { p.y+=inc_y; error = error + schwelle;}
                }
        } else                                  // steil nach oben oder steil nach unten
        {
                error = -Math.abs(dy);
                delta = 2*Math.abs(dx);
                schwelle = 2*error;
                while (p.y != q.y) {
                        set_pixel(p);
                        p.y+=inc_y;
                        error = error + delta;
                        if (error >0) { p.x+=inc_x; error = error + schwelle;}
                }

        }
        set_pixel(q);
}
```

3.2.1 Antialiasing

Bei gleichbleibender Auflösung (d.h. Pixel pro Zeile/Spalte) kann bei Schirmen mit mehreren Grauwerten pro Pixel die Qualität der Linie gesteigert werden. Hierzu werden die Pixel proportional zur Überlappung mit der Ideallinie geschwärzt.



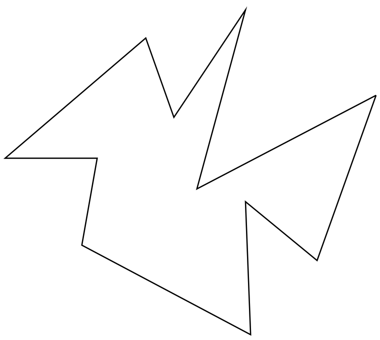
Beispiel: Linie von (1,0) nach (6,1) hat eine Überlappung

von	15 %	mit Pixel (5,2)
von	95 %	mit Pixel (5,1)
von	5 %	mit Pixel (5,0).

Eine diagonale Linie verwendet dieselbe Anzahl von Pixeln wie eine horizontale Linie für eine um den Faktor $\sqrt{2}$ längere Strecke. Daher erscheinen horizontale und vertikale Linien kräftiger als diagonale. Dies kann durch Antialiasing-Techniken behoben werden.

3.3 Polygon

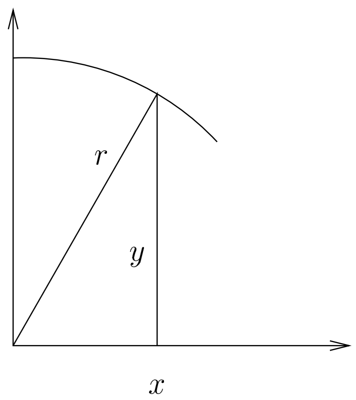
Ein Polygon wird spezifiziert durch eine Folge von Punkten, die jeweils durch Linien verbunden sind. Anfangs- und Endpunkt sind identisch.



3.4 Kreis

Problem: Gegeben Mittelpunkt (x, y) und Radius r. Bestimme die "anzuschaltenden" Pixel für einen Kreis mit Radius r um den Punkt (x, y).

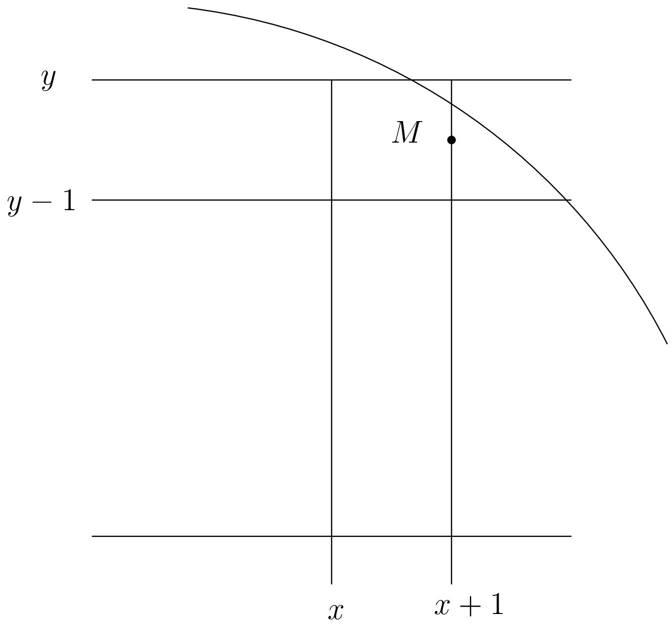
Betrachte zunächst den Verlauf eines Kreises um $(0, 0)$ im 2. Oktanten.
Für einen Punkt (x, y) sei $F(x, y) = x^2 + y^2 - r^2$



Es gilt:

$$\begin{aligned} F(x,y) &= 0 \text{ für } (x,y) \text{ auf dem Kreis,} \\ F(x,y) &> 0 \text{ für } (x,y) \text{ außerhalb des Kreises,} \\ F(x,y) &< 0 \text{ für } (x,y) \text{ innerhalb des Kreises.} \end{aligned}$$

Verwende F als Entscheidungsvariable dafür, ob y erniedrigt werden muß, wenn x erhöht wird. F wird angewendet auf die Mitte M zwischen Zeile y und $y - 1$.



$\Delta = F(x+1, y-\frac{1}{2})$

Falls $\Delta < 0 \Rightarrow M$ liegt innerhalb $\Rightarrow (x+1, y)$ ist ideal
$\Delta \geq 0 \Rightarrow M$ liegt außerhalb $\Rightarrow (x+1, y-1)$ ist ideal
Idee: Entscheide anhand von Δ, ob y erniedrigt werden muß oder nicht, und rechne neues Δ aus.

$$
\begin{array}{rcll}
\text{Sei "altes" } \Delta & = & F(x+1, y-\frac{1}{2}) & = (x+1)^2 + (y-\dfrac{1}{2})^2 - r^2 \text{ gegeben.} \\
\text{falls } \Delta & < & 0 \Rightarrow & \\
\text{"neues" } \Delta' & = & F(x+2, y-\frac{1}{2}) & = (x+2)^2 + (y-\dfrac{1}{2})^2 - r^2 = \Delta + 2x + 3 \\
\text{falls } \Delta & \geq & 0 \Rightarrow & \\
\text{"neues" } \Delta' & = & F(x+2, y-\frac{3}{2}) & = (x+2)^2 + (y-\dfrac{3}{2})^2 - r^2 = \Delta + 2x - 2y + 5 \\
\text{Startwert für } \Delta & = & F(1, r-\frac{1}{2}) & = 1^2 + (r-\dfrac{1}{2})^2 - r^2 = \dfrac{5}{4} - r
\end{array}
$$

Also ergibt sich:

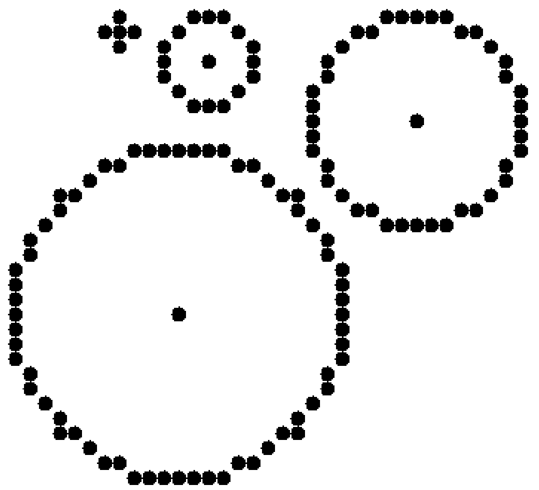
```
void bresenham_kreis_1 (            // zeichnet Kreis um Punkt 0,0
                        int r)      // mit Radius r
{
        double delta;
        int x, y;

        x=0; y=r;
        delta = 5.0/4.0 - r;

        while (y>=x) {

                set_pixel (new Point(x,y));

                if (delta < 0.0 ) { delta += 2*x + 3.0;       x++;       }
                            else { delta += 2*x - 2*y + 5.0; x++; y--; }
        }
}
```

Substituiere `delta` durch `d := delta - 1/4`. Dann ergibt sich
als neue Startbedingung: `d := 5/4 - r - 1/4 = 1 - r`
als neue if-Bedingung: `if (d < 0)`
Da `d` nur ganzzahlige Werte annimmt, reicht der Vergleich mit 0.

Weitere Verbesserung:
Ersetze `2x + 3` durch `dx` mit Initialwert `dx = 3;`
Ersetze `2x - 2y + 5` durch `dxy` mit Initialwert `dxy = -2*r + 5`
Es ergibt sich

```
void bresenham_kreis_2 (           // zeichnet Kreis um Punkt 0,0
                        int r)      // mit Radius r
{
        int x, y, d, dx, dxy;

        x=0; y=r; d=1-r;
        dx=3; dxy=-2*r+5;
        while (y>=x) {

                set_pixel (new Point(x,y));

                if (d < 0) { d += dx;  dx += 2; dxy += 2; x++;      }
                      else { d += dxy; dx += 2; dxy += 4; x++; y--; }
        }
}
```

Um den ganzen Kreis zu zeichnen, wird die Symmetrie zum Mittelpunkt ausgenutzt.
Die Anzahl der erzeugten Punkte des Bresenham-Algorithmus für den vollen Kreis beträgt $4r \cdot \sqrt{2}$ Punkte. Verglichen mit dem Kreisumfang vom $2r\pi$ liegt dieser Wert um ca. 10% zu tief.



Vom Bresenham-Algorithmus erzeugte Kreise

```
/*************************************************************************************/
/*                                                                                   */
/*                  Bresenham-Algorithmus zum Zeichnen eines Kreises                 */
/*                                                                                   */
/*************************************************************************************/

private void bresenham_kreis (                   // zeichnet mit Bresenham-Algorithmus
        Point p,                                 // einen Kreis um den Punkt p
        int r)                                   // mit Radius r
{
        int x,y,d,dx,dxy;
        x=0; y=r; d=1-r;
        dx=3; dxy=-2*r+5;
        while (y>=x)
        {
                set_pixel( new Point( p.x+x, p.y+y) );  // alle 8 Oktanden werden
                set_pixel( new Point( p.x+y, p.y+x) );  // gleichzeitig gezeichnet
                set_pixel( new Point( p.x+y, p.y-x) );
                set_pixel( new Point( p.x+x, p.y-y) );
                set_pixel( new Point( p.x-x, p.y-y) );
                set_pixel( new Point( p.x-y, p.y-x) );
                set_pixel( new Point( p.x-y, p.y+x) );
                set_pixel( new Point( p.x-x, p.y+y) );

                if (d<0) { d=d+dx;  dx=dx+2; dxy=dxy+2; x++;      }
                    else { d=d+dxy; dx=dx+2; dxy=dxy+4; x++; y--; }
        }
}
```