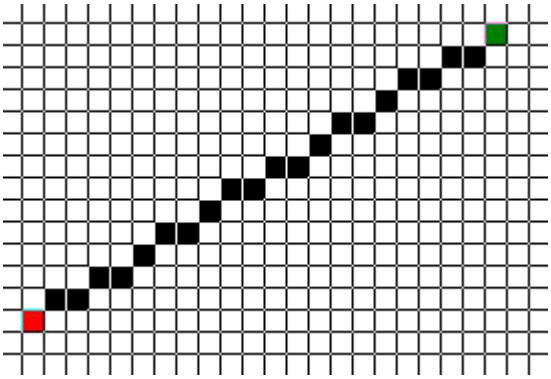
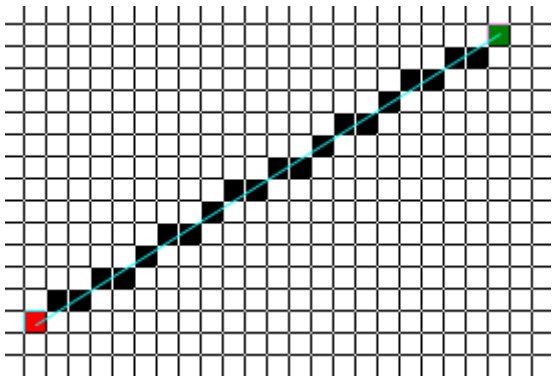
Plotten von Linien (nach Jack Bresenham, 1962)



Eine auf dem Bildschirm darzustellende Linie sieht treppenförmig aus, weil der Computer Linien aus einzelnen (meist quadratischen) Bildpunkten, Pixels = „picture elements“ genannt, erzeugt.

Ein Beispiel zeigt die Grafik rechts, die eine Linie vom roten Punkt P1 zum grünen Punkt P2 darstellt. Die Steigung ist m = 13/21 , sie liegt somit zwischen 0 und 1.



Denkt man sich eine dünne Verbindungslinie von P1 zu P2 gezeichnet, so trifft diese Linie nicht alle Bildpunkte, die der Computer für die Gerade einzeichnet, wie die Grafik rechts zeigt.

Wir erwarten jedoch , dass die vom Computer zu zeichnenden Pixels möglichst nahe an dieser gedachten Linie liegen !

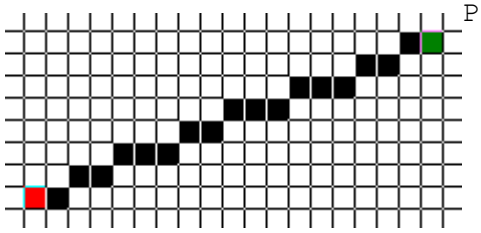
Woher aber „weiß“ der Computer, wie er möglichst nahe an die gedachte Linie herankommen soll bzw. welchen Bildpunkt er als nächsten nach P1 einfärben soll ?

Gesucht ist ein entsprechender Algorithmus zum Zeichnen von Linien.
Der bekannteste Linienalgorithmus ist nach Jack Bresenham benannt.

Der Bresenham-Algorithmus geht so vor, dass er zu jeder x-Koordinate eine y- Koordinate derart sucht, dass der entsprechende Punkt P(x/y) möglichst nahe an der zu zeichnenden Linie liegt.

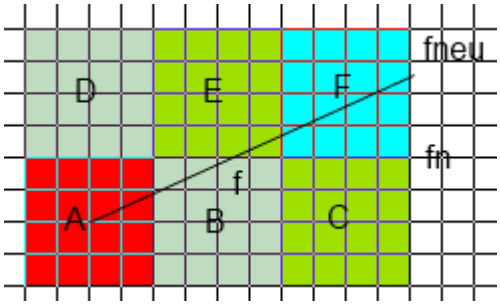
Anmerkung: Zunächst genügt es, den Steigungswinkel auf Werte zwischen 0° und 45° zu beschränken.
Die Steigung m = dy/dx liegt dann zwischen m = 0 und m = 1 .

Beispiel:



Wir gehen davon aus, dass bei jedem Schritt (ausgehend von P1) der x-Wert um 1 erhöht wird.
Man erkennt, dass dann der y-Wert nur in manchen Fällen um 1 erhöht wird !
Wie stellt man nun fest, wann der y-Wert zu erhöhen ist ?

Vorgehensweise:



Wir beginnen beim roten Punkt A (= Anfang der Linie), der bereits eingezeichnet wurde.
Bei Erhöhung des x-Wertes um 1 geht die Linie im allgemeinen nicht mehr exakt durch einen Pixel, denn sie müsste in der Grafik rechts ja sonst durch die Mitte von B verlaufen !
Stattdessen verläuft sie jedoch zwischen B und E .
Es entsteht daher ein "Fehler" f, der von der Geradensteigung m abhängt !

Wir vereinbaren für f folgendes:

f werde (in y-Richtung) gemessen zwischen Linie und der Mitte des zu setzenden Punktes.

Genauer formuliert:
P(xp/yp) sei der zu setzende Punkt.
L(xw/yw) sei der "wahre" Linienpunkt (kein Pixel).
Dann ist f = yw - yp .

Folgerungen:
Verläuft die Linie oberhalb des Punktes, so gilt: $0 < f <= 0{,}5$.
Verläuft die Linie oberhalb des Punktes, so gilt: $-0{,}5 < f < 0$.
Liegt der Punkt auf der Linie, so ist $f = 0$.
f liegt demnach im Intervall $]\ -0{,}5\ ;\ +0{,}5\ ]$.

In der Skizze liegt offensichtlich B näher an der Linie, es gilt also $f > 0$.
B muß daher gesetzt werden, d.h. y wird nicht verändert.

Bei erneuter Erhöhung von x liegt F näher an der Linie als C,
d.h. die Linie verläuft unterhalb des zu setzenden Punktes, f ist negativ !
In diesem Fall ist y um 1 zu erhöhen .

Wie läßt sich nun aber der Fehler f "messen" ?

Wie oben bereits erwähnt, gibt es einen Zusammenhang zwischen f und m.
Erhöht man, beginnend bei A, den x-Wert um 1, so ist f gemäß des Strahlensatzes exakt gleich der Geradensteigung m . Man könnte daher zu Beginn f auf 0 setzen und nach der ersten Erhöhung von x das neue f berechnen mit: $f = f + m$

Wendet man dieses Verfahren auch bei der nächsten Erhöhung von x an, so erhält man eine (positive) Zahl f_n (vgl. Skizze 3).
Da jedoch der Punkt F zu setzen ist, müsste nun f_{neu} ermittelt werden.

Zwischen f_{neu} und f_n besteht der einfache Zusammenhang: $f_{neu} + f_n = 1$.
Dies ist gleichbedeutend mit: $f_{neu} = 1 - f_n$.
Berücksichtigen wir noch, daß f_{neu} nun ein negatives Vorzeichen erhalten muß, so ergibt sich die Beziehung:

$$f_{neu} = -\ (\ 1 - f_n\ ) = f_n - 1$$

Zusammenfassung:

Bei jeder Erhöhung von x um 1 müssen wir f um die Steigung m erhöhen.
Falls f > 0,5 , dann ist f um 1 zu erniedrigen und y um 1 zu erhöhen.

Wie sieht der entsprechende Algorithmus aus,
wenn die Parameter x1, x2, y1, y2 vorgegeben sind ?

Es ergibt sich der folgende Elementar-ALGORITHMUS: ($0 \leq m \leq 1$)

```
dx = x2 - x1
dy = y2 - y1
m = dy / dx
y = y1
f = 0
wiederhole für x von x1 bis x2
        Plotte(x,y)
        f = f + m
        wenn f > 0,5
                dann  y = y + 1
                      f = f - 1
ende wiederhole
```

Dieser Algorithmus besitzt einige Nachteile (z.B. Verwendung von Gleitkommazahlen).

Verbesserung des Linienalgorithmus (Bresenham)

Durch geeignete Transformationen läßt sich der Algorithmus auf
die ausschließliche Verwendung von Ganzzahlen (Integers) zurückführen .

Wir ersetzen f durch 2*dx*(f - 0,5)

```
dx = x2 - x1
dy = y2 - y1
y = y1
f = -dx
wiederhole für x von x1 bis x2
        Plotte(x,y)
        f = f + 2*dy
        wenn f > 0
                dann  y = y + 1
                      f = f - 2*dx
ende wiederhole
```

Hierdurch kommen nun keine Gleitkommazahlen mehr vor !

Bei Verwendung einiger Abkürzungen ergibt sich der

Bresenham-Algorithmus für Steigungswinkel $\alpha \in [\ 0° ; 45° \ ]$.

```
dx = x2 - x1
dy = y2 - y1
a = 2*dy
b = 2*dx
y = y1
f = -dx
wiederhole für x von x1 bis x2
        Plotte(x,y)
        f = f + a
        wenn f > 0
                dann  y = y + 1
                      f = f - b
ende wiederhole
```

Wie muss der Algorithmus für $\alpha \in$] 45° ; 90°] aussehen ?

Erweiterung des Linienalgorithmus (Bresenham)

Liegt der Steigungswinkel α der Linie zwischen 45° und 90° , so muss der Bresenham-Algorithmus modifiziert werden. Das Problem lässt sich recht einfach lösen, indem man bei jedem Schritt die y-Koordinate um 1 erhöht und untersucht, ob die x-Koordinate ihren Wert beibehält oder ebenfalls erhöht werden muß.
In Bezug auf den "Bresenham-Elementaralgorithmus" handelt es sich also lediglich um eine Vertauschung von x und y !
Beachtet werden muss noch, dass zum Fehler f nun nicht mehr die Steigung m, sondern der Kehrwert 1/m = dx / dy der Steigung zu addieren ist !

Aus diesen Überlegungen entsteht folgender

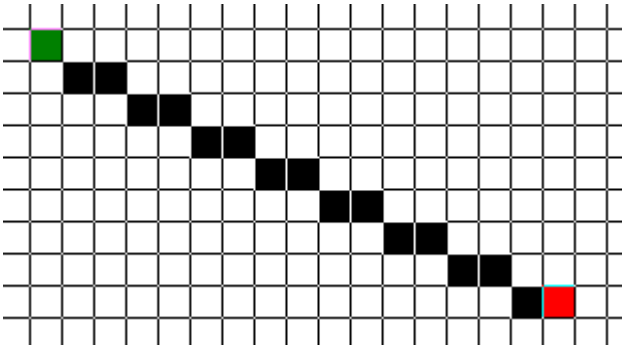
Algorithmus für Steigungswinkel α zwischen 45° und 90° !

```
dx = x2 - x1
dy = y2 - y1
a = 2*dy
b = 2*dx
x = x1
f = -dy
wiederhole für y von y1 bis y2
        Plotte(x,y)
        f = f + a
        wenn f > 0
                dann   x = x + 1
                       f = f - b
ende wiederhole
```

Eine Unterscheidung der beiden Fälle ($0° \le \alpha \le 45°$ sowie $45° < \alpha \le 90°$) ist möglich durch Vergleich der Größen dx und dy.
Gilt $dy \le dx$, so liegt der erste Fall vor, andernfalls der zweite.

Es bleiben noch zwei weitere Fälle zu untersuchen: $dx < 0$ sowie $dy < 0$.
Ist $dx < 0$, d.h. $x2 < x1$, so bewegt man sich nach links statt nach rechts.

P2(x2/y2)



← P1(x1/y1)

Dieses Problem lässt sich lösen, indem die x-Schrittweite auf -1 statt +1 gesetzt wird. Außerdem muss der Betrag von dx gebildet werden, da zum Fehler f nur positive Steigungen addiert werden dürfen (vgl. Elementaralgorithmus). Entsprechend setzt man im Fall $dy < 0$ die y-Schrittweite auf -1 statt +1 und bildet den Betrag von dy.

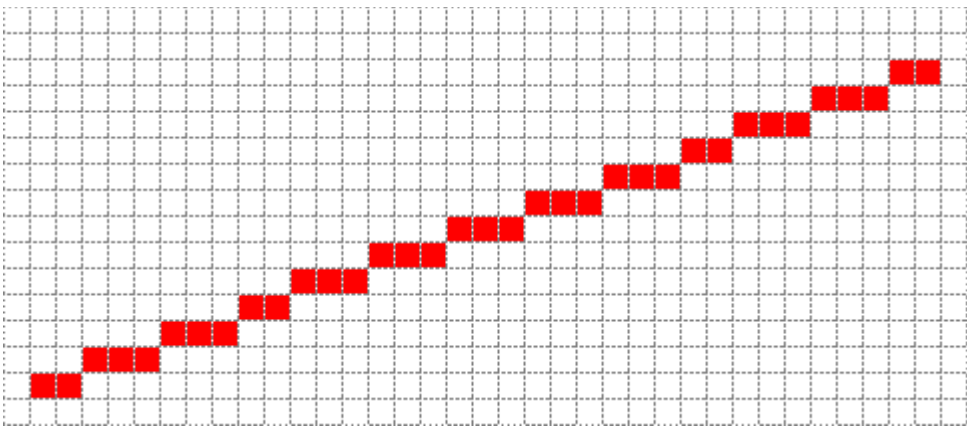
Die 4 betrachteten Fälle lassen sich in einem gemeinsamen Algorithmus folgendermaßen kombinieren:

Linien-Algorithmus von Bresenham (berücksichtigt sämtliche vorkommenden Fälle)
Gezeichnet wird von P(x,y) nach Pe(xe,ye)

```
dx = xe - x
dy = ye - y
xstep = 1
ystep = 1
wenn dx < 0
      dann  dx = -dx
            xstep = -1
wenn dy < 0
      dann  dy = -dy
            ystep = -1
a = 2*dx
b = 2*dy
wenn dy <= dx
      dann  f = -dx
            solange x <> xe wiederhole
                  Plotte(x,y)
                  f = f + b
                  wenn f > 0
                        dann  y = y + ystep
                              f = f - a
                  x = x + xstep
            ende solange
      sonst f = -dy
            solange y <> ye wiederhole
                  Plotte(x,y)
                  f = f + a
                  wenn f > 0
                        dann  x = x + xstep
                              f = f - b
                  y = y + ystep
            ende solange
plotte(x,y)
```

Beispiel: x1=1, y1=14, x2=35, y2=2

Dann gelten: dx=34, xstep=1, dy=12, ystep=-1, a=68, b=24, f=-34 etc.



JAVA-Programm:

```
import java.awt.*;
import java.awt.image.BufferedImage;

import javax.swing.*;

// Ac 24.02.2017

@SuppressWarnings("serial")
public class Bresenham extends JPanel {

   public static void main(String[] args) {
      SwingUtilities.invokeLater(new Runnable() {
       @Override
       public void run() {
             JFrame f = new JFrame();
             f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
             f.setTitle("Bresenham-Algorithmus für Linien");
             f.setResizable(false);
             f.add(new Bresenham(), BorderLayout.CENTER);
             f.pack();
             f.setLocationRelativeTo(null);
             f.setVisible(true);
         }
    });
  }

   public static BufferedImage bufKos;
   public static Graphics2D grBufKos;

   // Konstruktor
   public Bresenham() {
      Dimension dim = new Dimension(640, 640);
      setPreferredSize(dim);
      setBackground(Color.white);
      bufKos = new BufferedImage(dim.width, dim.height, BufferedImage.TYPE_INT_RGB);
      grBufKos = bufKos.createGraphics();
      // "Grafik-Kontext" des Puffers (für Graphics2D)
      grBufKos.setColor(Color.WHITE);
      grBufKos.fillRect(0, 0, dim.width, dim.height);
      // weißer Hintergrund für Puffer

      zeichneBresiLinie(1, 14, 35, 2, Color.RED);

  }   // Ende Konstruktor


   @Override
   public void paintComponent(Graphics grPnl) { // zeichnet auf ein Panel
      Graphics2D g2d = (Graphics2D) grPnl;
      super.paintComponent(g2d);
      g2d.drawImage(bufKos, 0, 0, this);
      // zeichnet bufKos(Puffer) auf "this"=JPanel
      repaint();
   }
```

```
   public void plotPixel(int x, int y, Color col) {
      bufKos.setRGB(x, y, col.getRGB());
   }

  void zeichneBresiLinie(int x1, int y1, int x2, int y2, Color farbe) {
     // Bresenham-Algorithmus für Linien von Ac
     int    x,y,dx,dy,Fehler,a,b,xstep,ystep;
          dx = x2-x1;     dy = y2-y1;
          xstep = 1;      ystep = 1;
          x = x1;         y = y1;
          if (dx < 0) {
              dx = -dx;
              xstep = -1;
          }
          if (dy < 0) {
              dy = -dy;
              ystep = -1;
          }
          a = dx + dx;     b = dy + dy;
          if (dy <= dx) {
               Fehler = -dx;
               while (x != x2) {
                  plotPixel(x,y,farbe);
                   Fehler = Fehler + b;
                   if (Fehler > 0) {
                       y = y + ystep;
                       Fehler = Fehler - a;
                   }
                   x = x + xstep;
               } // while
          } else {
               Fehler = -dy;
               while (y != y2) {
               plotPixel(x,y,farbe);
                   Fehler = Fehler + a;
                   if (Fehler > 0) {
                       x = x + xstep;
                       Fehler = Fehler - b;
               }
                   y = y + ystep;
               }  // while
          }
          plotPixel(x,y,farbe);
  }

}
```