UNIVERSITÄT
LEIPZIG

Rasterung

# COMPUTERGRAPHIK

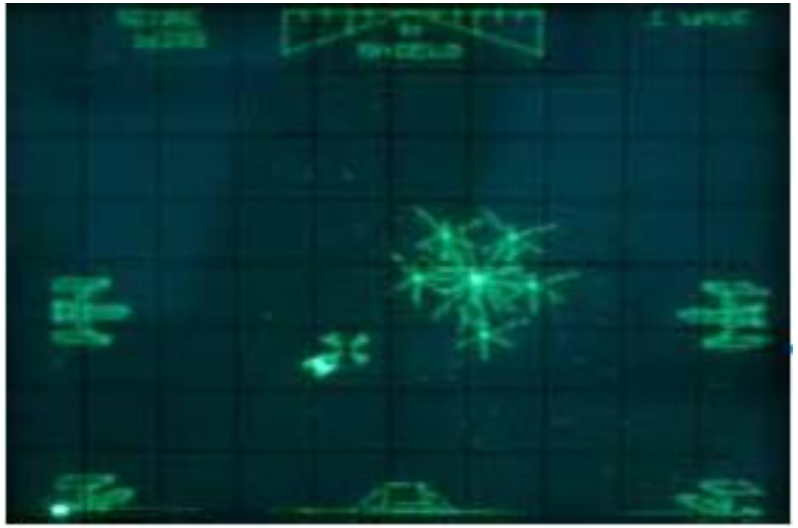
# Inhaltsverzeichnis

## 3 Rasterung

# 3. Rasterung

## Darstellungsmöglichkeiten

- Vektordarstellung
  - Plotter, Elektronenstrahl (Oszilloskop)
  - Einzelne Linien (Vektoren) werden gezeichnet.
  - Bild setzt sich aus Linien zusammen.
  - Eingeschränkte Darstellungsmöglichkeiten




# 3. Rasterung

## Darstellungsmöglichkeiten

- Rasterdisplays
  - Bild wird in Bildpunkte diskretisiert.
  - Erfordert Framebuffer (Bildspeicher)
    - Speichert die Bitmap des angezeigten Bildes
    - Digitale Kopie des Monitorbildes
    - Heutzutage doppelt oder dreifach gepuffert

Speicherbedarf:

1024 × 768 × TrueColor ≈ 2,25 MB

1920 × 1200 × TrueColor ≈ 6,6 MB

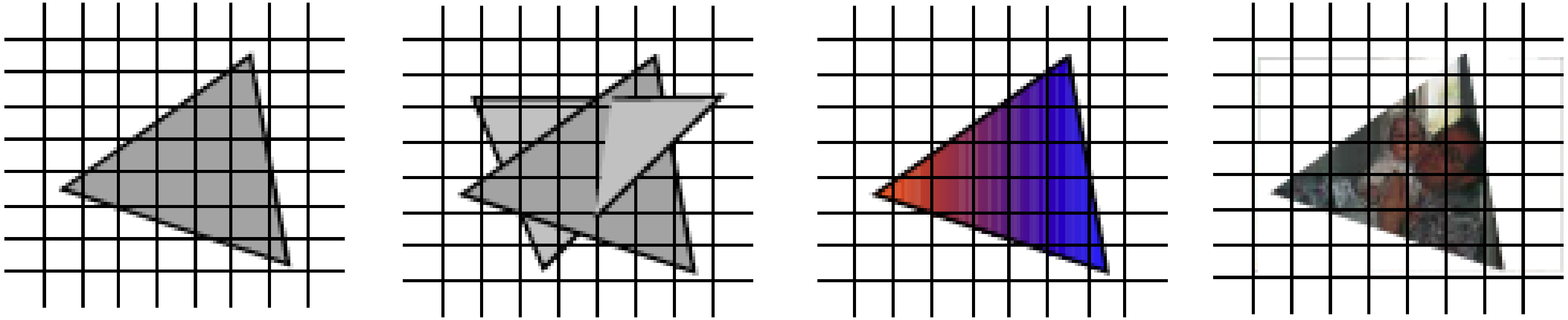
2560 × 1600 × TrueColor ≈ 12 MB

Truecolor:

- 24 Bit
- Je ein Byte (8 Bit) für R, G und B
- 2^24 = 16.777.216

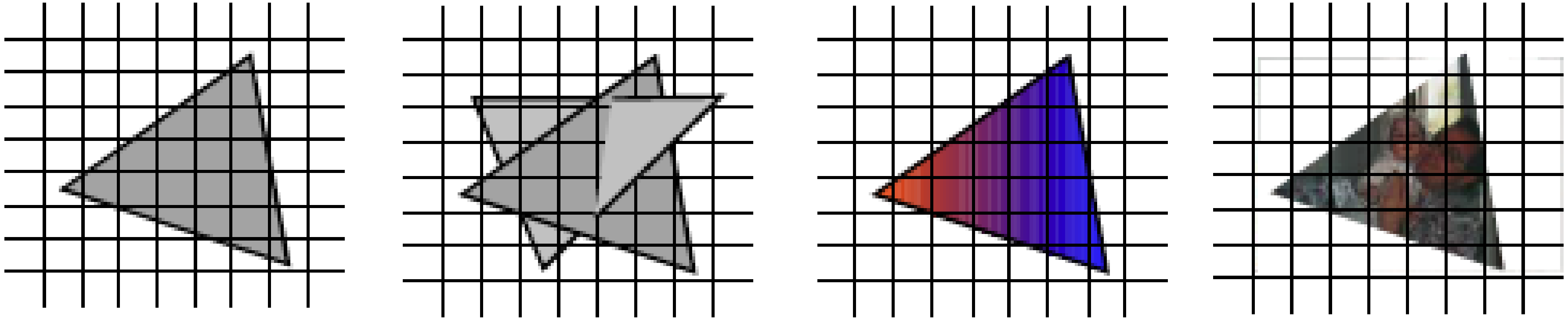
## 3. Rasterung

- Die dominierenden Rasterbildschirme erfordern die „Zerlegung“ aller darzustellenden geometrischen Objekte in Bildschirmpunkte.
- Dieser Prozess wird auch als Rasterung bezeichnet.
- Dies ist die Aufgabe der Bilderzeugungseinheit DPU (Display Processing Unit) des Bildrechners: Rastereinheit/Rasterprozessor



# 3. Rasterung

- Wir beschäftigen uns näher mit:
  - Rasterung von Geraden, Kreisen, Ellipsen, Polygonen
  - Antialiasing von Linien und Polygonen



## 3.1 Rasterung von Geraden

### Problemstellung

– Darstellung einer Linie auf einem Rasterbildschirm erfordert die Bestimmung der „am besten passenden“ Punkte im Raster bzw. Gitter (geeignete ganzzahlige Rundung).

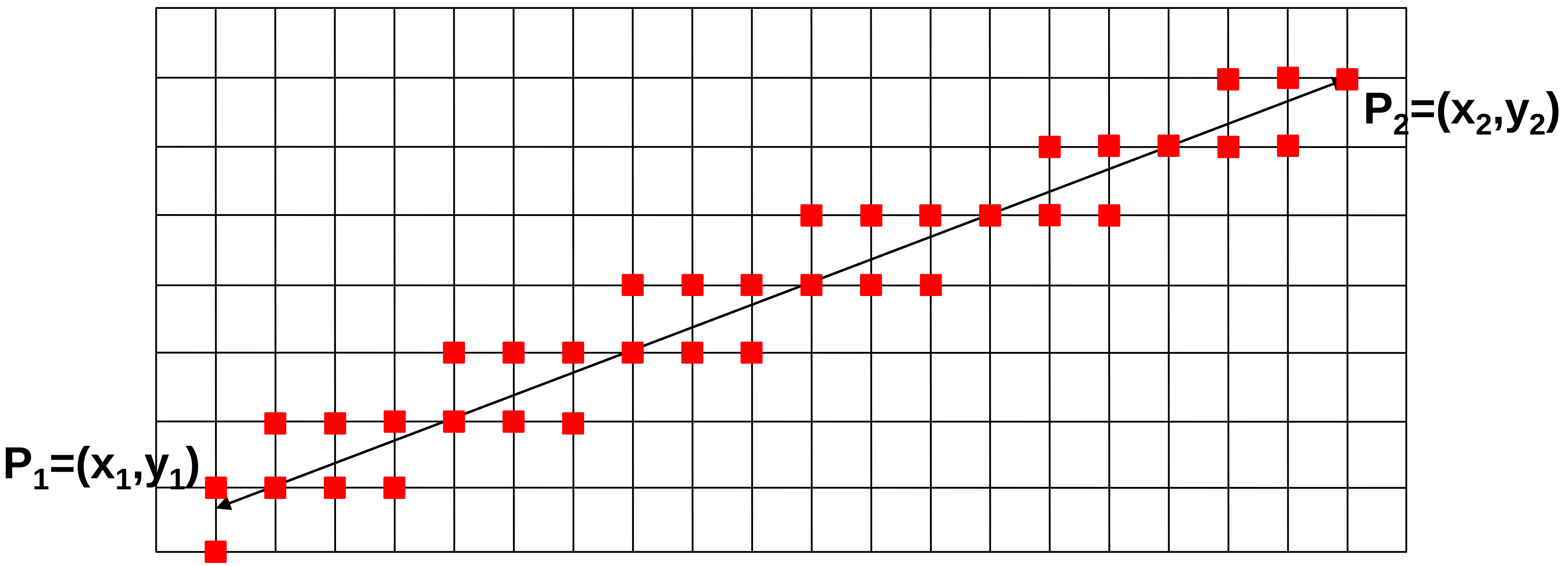



**Abb. 2.1: Mögliche Rasterkandidaten**

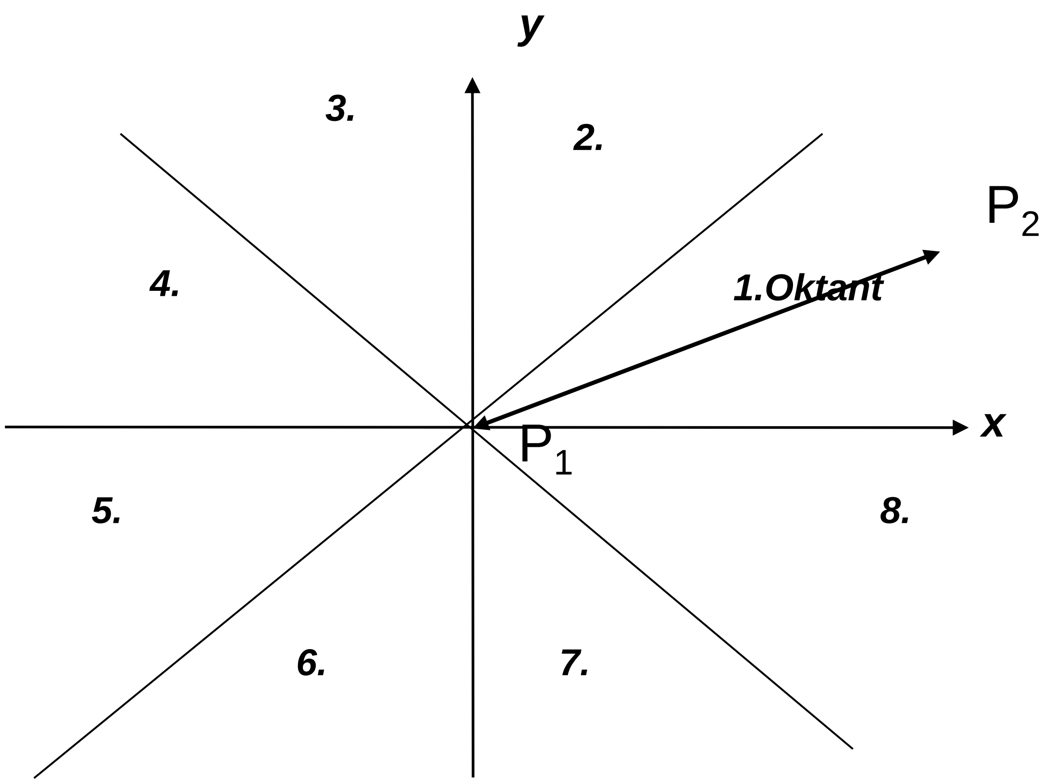
## 3.1 Rasterung von Geraden

### Anforderungen

- Geraden
  - sollen gerade erscheinen
  - sollen gleichmäßig hell erscheinen
  - sollen schnell gezeichnet werden
- Algorithmus
  - soll leicht in Hardware implementierbar sein

## 3.1 Rasterung von Geraden

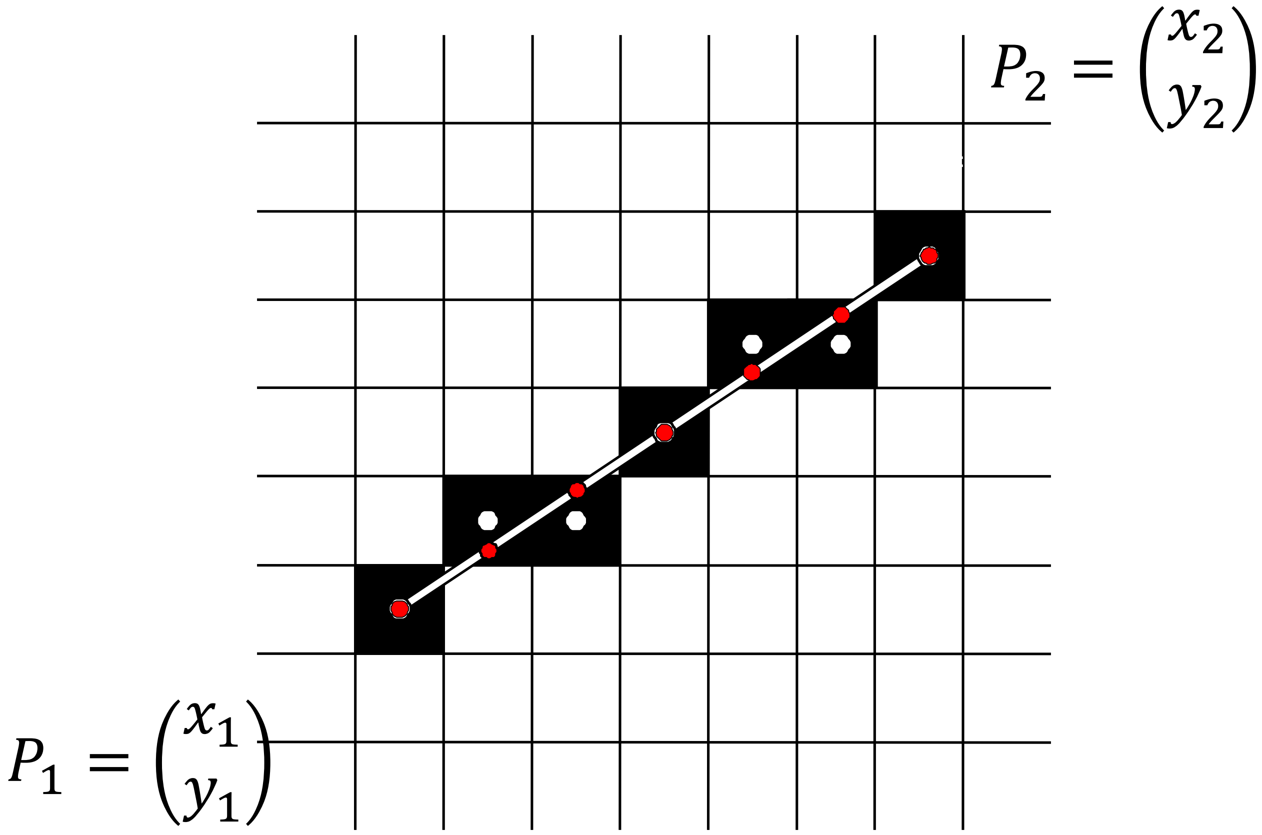
- Fallunterscheidung in Oktanten (nach Steigung)
- Hier:
  - Ursprung des zugeordneten Koordinatensystems in $P_1$
  - 1. Oktant
  - $P_1$ und $P_2$ auf Rastergitter



DDA-Algorithmus für Geraden [1930er]

- DDA
  - Digital Differential Analyzer
  - Digitaler Integrierer

- Ein DDA-Algorithmus generiert eine Kurve (nicht nur Geraden) aus einer beschreibenden Differentialgleichung.



## 3.1 Rasterung von Geraden: DDA

Geradengleichung: $y = ax + b$

$\Delta x =$

$\Delta y =$

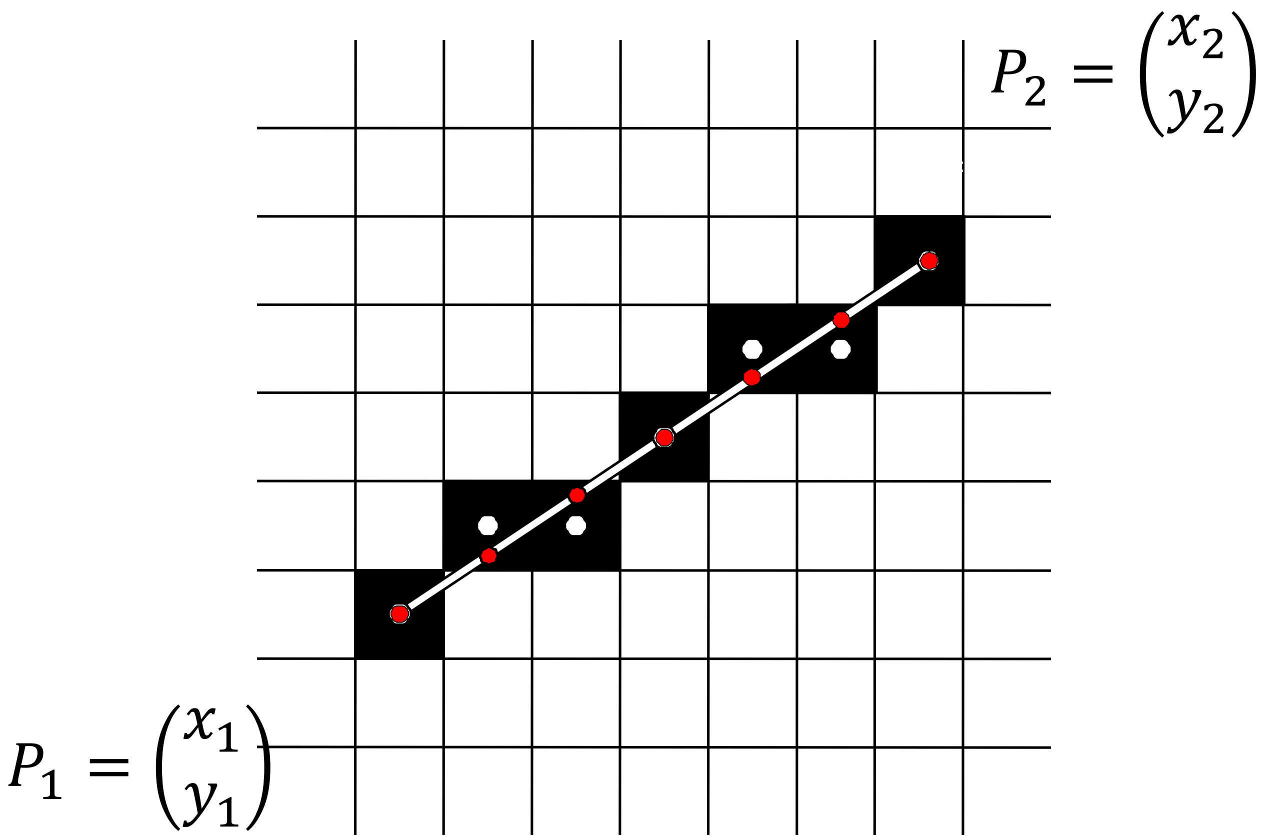
$a \quad =$

$b \quad =$

Direkte Berechnung: $\forall i = 1, \ldots, \Delta x$:

$x_{i+1} =$

$y_{i+1} =$



## 3.1 Rasterung von Geraden: DDA

Geradengleichung: $y = ax + b$

$\Delta x = x_2 - x_1$

$\Delta y = y_2 - y_1$

$a \quad = \frac{\Delta y}{\Delta x}, 0 \leq a \leq 1$

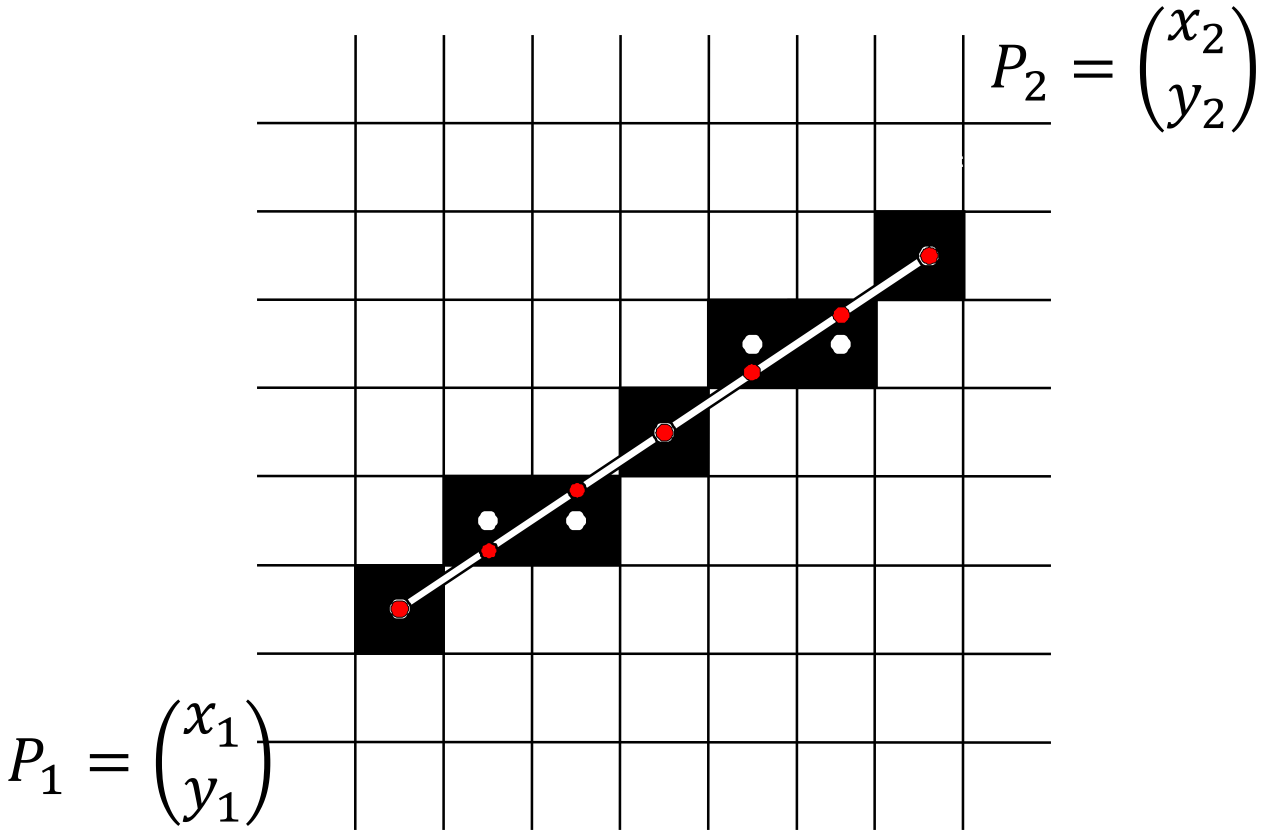
$b \quad = ax_1 - y_1$

Direkte Berechnung: $\forall i = 1, \ldots, \Delta x$:

$x_{i+1} = x_i + 1$

$y_{i+1} = ax_{i+1} + b$

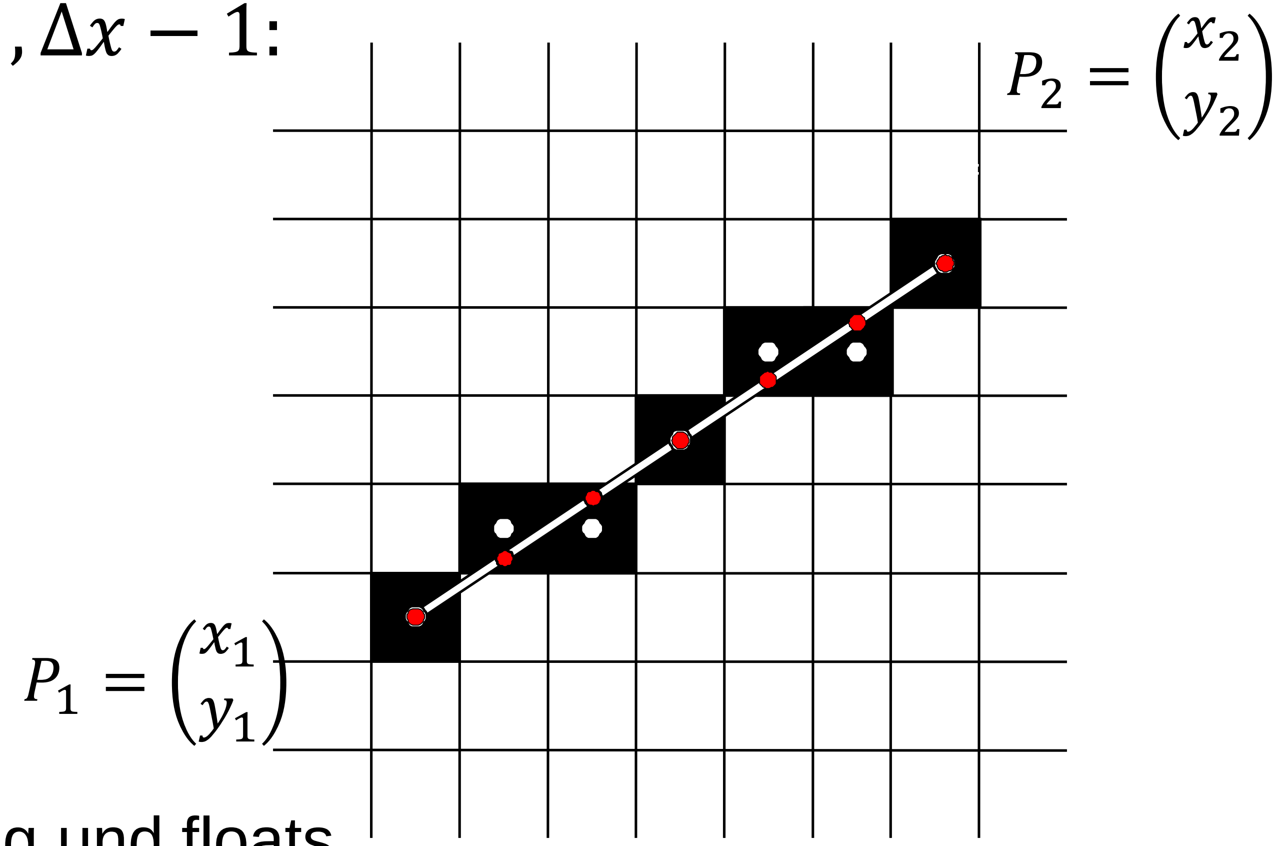
Plus Rundung



## 3.1 Rasterung von Geraden: DDA

Inkrementelle Berechnung: $\forall i = 0, \ldots, \Delta x - 1$:

$y_{i+1}$



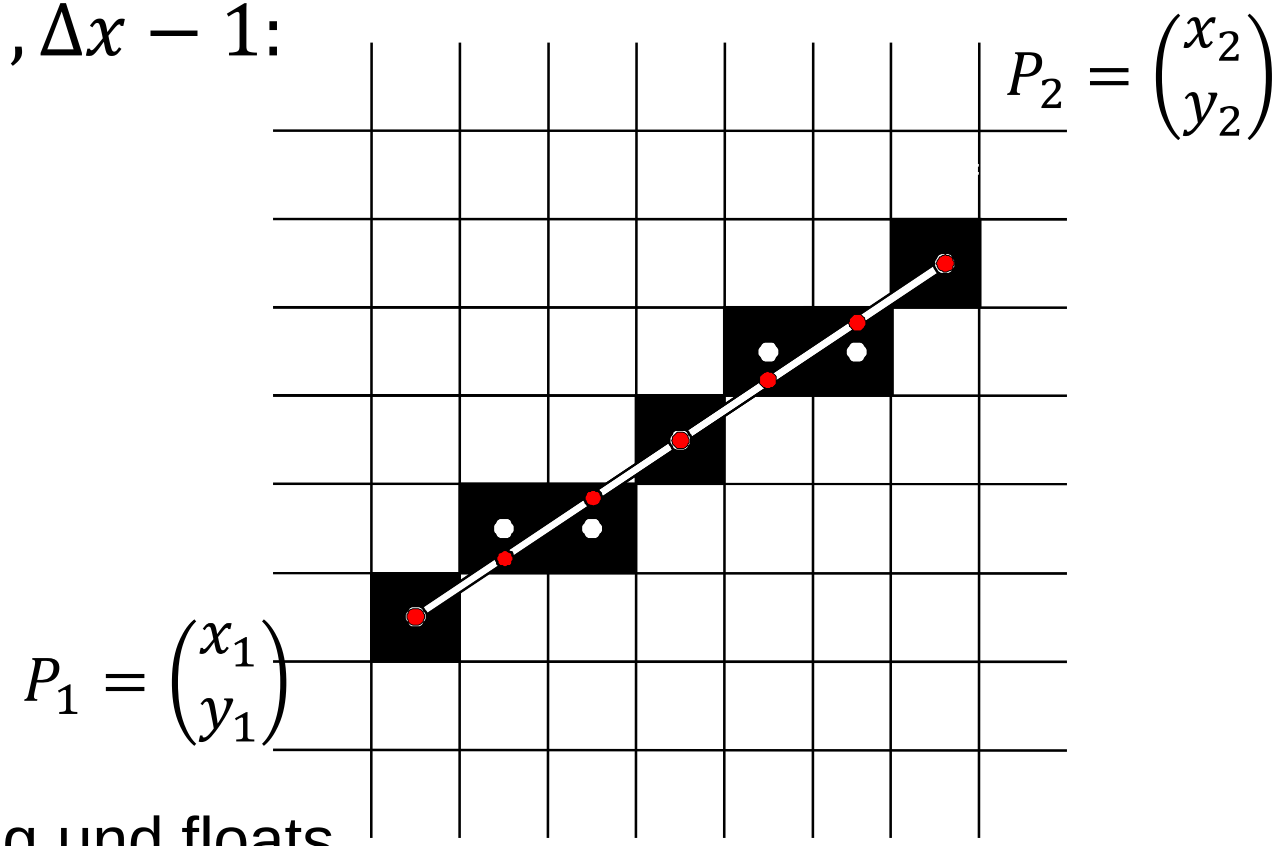

Vorteil: schneller

Nachteil beide: ungenau durch Rundung und floats

## 3.1 Rasterung von Geraden: DDA

Inkrementelle Berechnung: $\forall i = 0, \ldots, \Delta x - 1$:

$$
\begin{aligned}
y_{i+1} &= a x_{i+1} + b \\
&= a(x_i + 1) + b \\
&= a x_i + b + a \\
&= y_i + a \\
&= y_i + \frac{\Delta y}{\Delta x}
\end{aligned}
$$




Vorteil: schneller

Nachteil beide: ungenau durch Rundung und floats

## 3.2 Rasterung von Geraden: Bresenham

Bresenham-Algorithmus für Geraden [1962]

- Idee:
  - Abhängig von der Steigung der Geraden wird die x- oder y-Koordinate immer um genau eine Einheit geändert
  - Die andere Koordinate wird entweder nicht oder ebenfalls um eine Einheit geändert
  - Fallunterscheidung nach der kleineren Abweichung der Geraden zum nächsten Gitterpunkt in Koordinatenrichtung

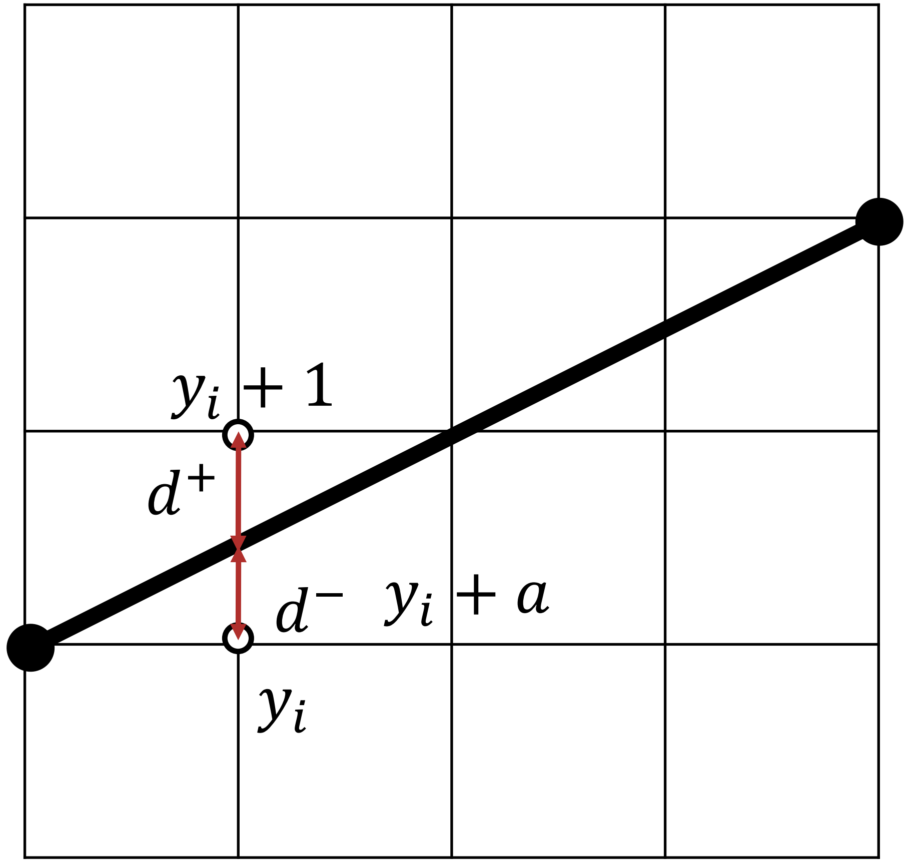
## 3.2 Rasterung von Geraden: Bresenham

- Entweder Punkt 1 oder Punkt 2 wird gezeichnet, je nachdem, welcher näher zur Geraden liegt

$d^-$

$d^+$

- Gehe nach oben, wenn $d^+ < d^-$



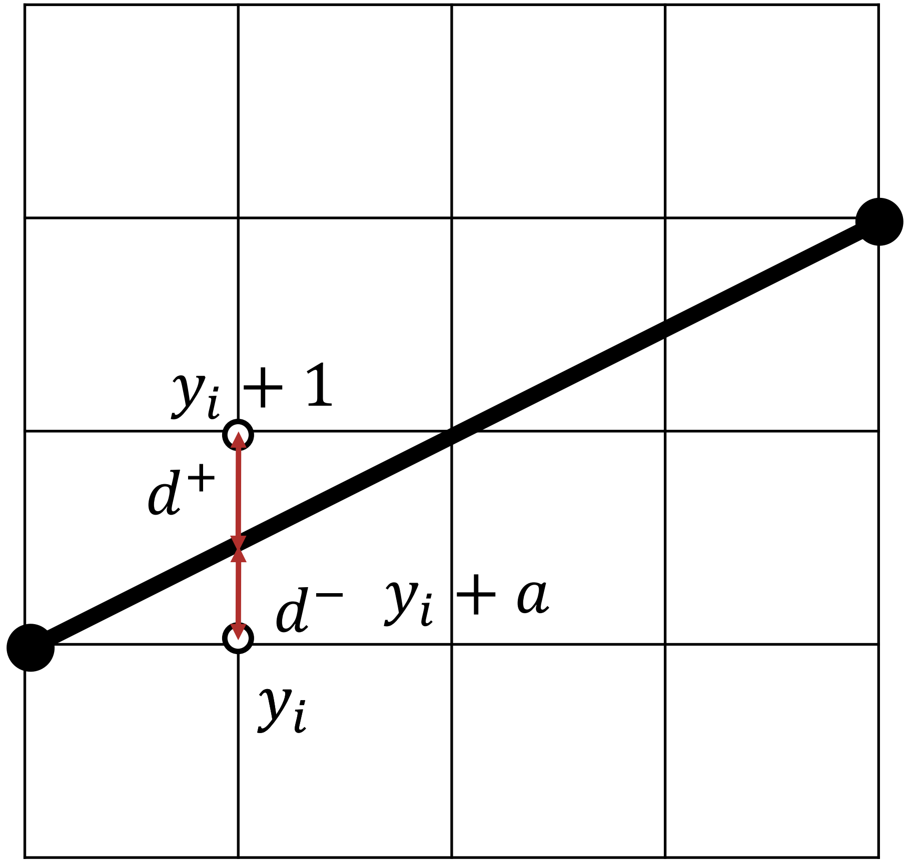
## 3.2 Rasterung von Geraden: Bresenham

- Entweder Punkt 1 oder Punkt 2 wird gezeichnet, je nachdem, welcher näher zur Geraden liegt

$$d^- \quad = y_i + a - y_i$$
$$= a$$

$$d^+ \quad = y_i + 1 - (y_i + a)$$
$$= 1 - a$$

- Gehe nach oben, wenn $d^+ < d^-$



## 3.2 Rasterung von Geraden: Bresenham

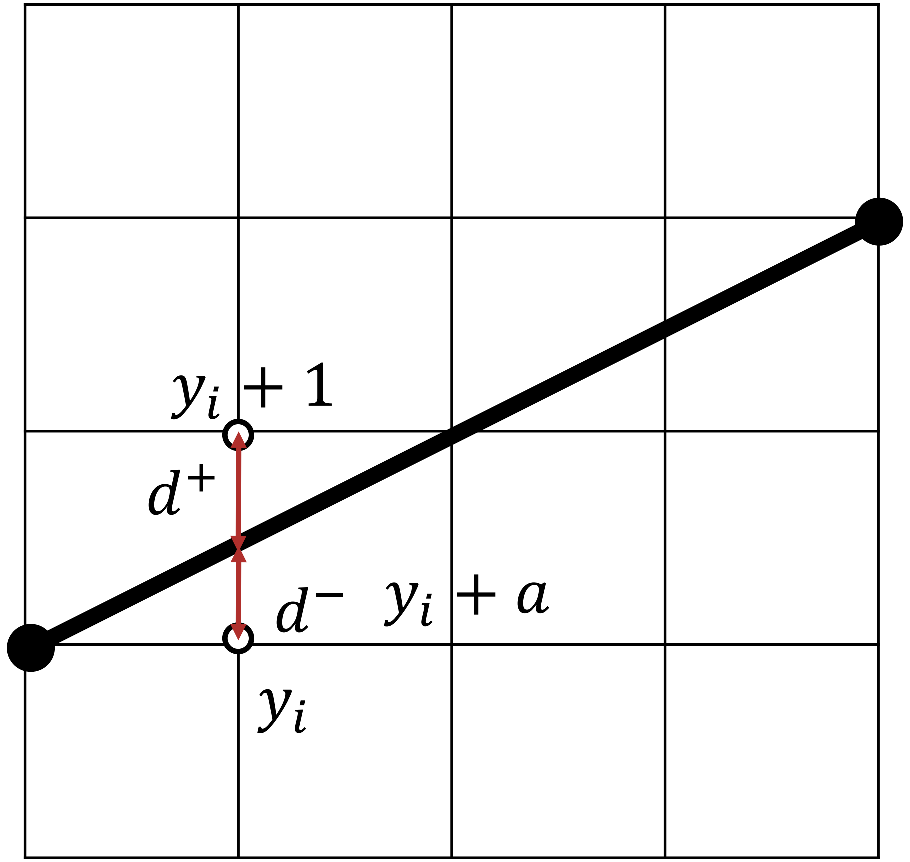
- Zur Realisierung wird eine **Entscheidungsgröße E** eingeführt:

$$E := d^- - d^+$$

- Gehe nach oben, wenn

$$d^+ < d^- \Leftrightarrow E > 0$$

- Das **Vorzeichen von E** dient dann als Kriterium für die Rundung auf den nächsten Rasterpunkt:
  - $E \leq 0: x++$
  - $E > 0: x++, y++$



# 3.2 Rasterung von Geraden: Bresenham

- Zur schnelleren Berechnung, kann E **inkrementell** berechnet werden
- Das **Vorzeichen von E** dient dann als Kriterium für die Rundung auf den nächsten Rasterpunkt und die Aktualisierung von E:
  - $E \leq 0 : x++,$

    $d^+ \mathrel{+}=$

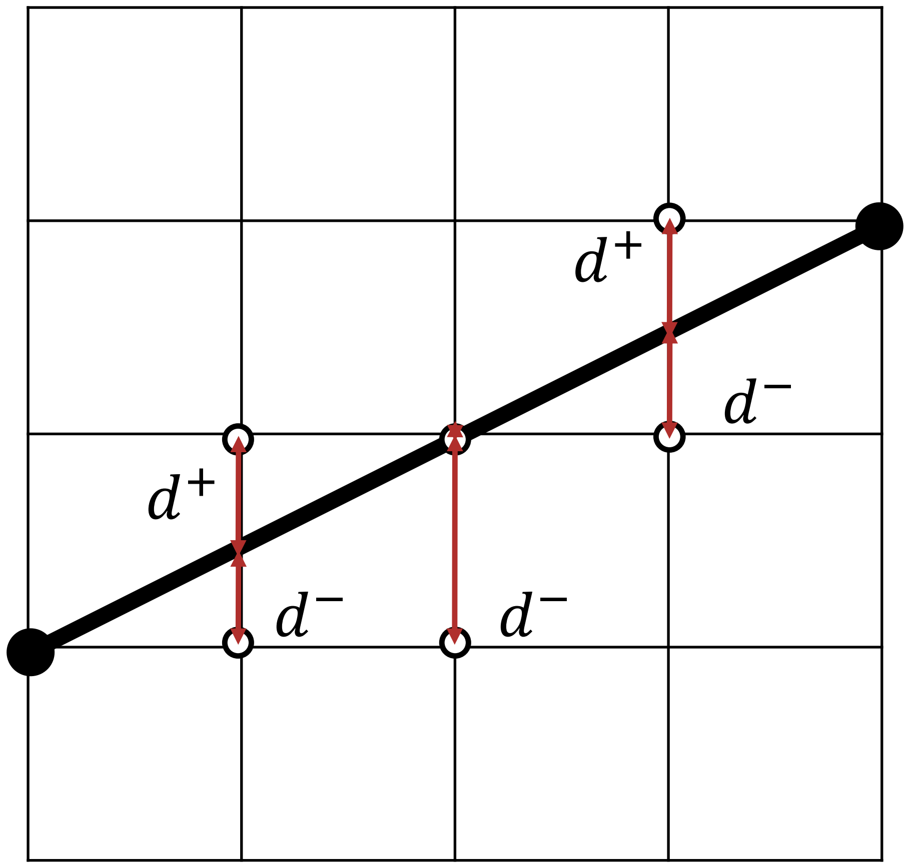
    $d^- \mathrel{-}=$

    $E \;\; \mathrel{+}=$
  - $E > 0 : x++, y++,$

    $d^+ \mathrel{+}=$

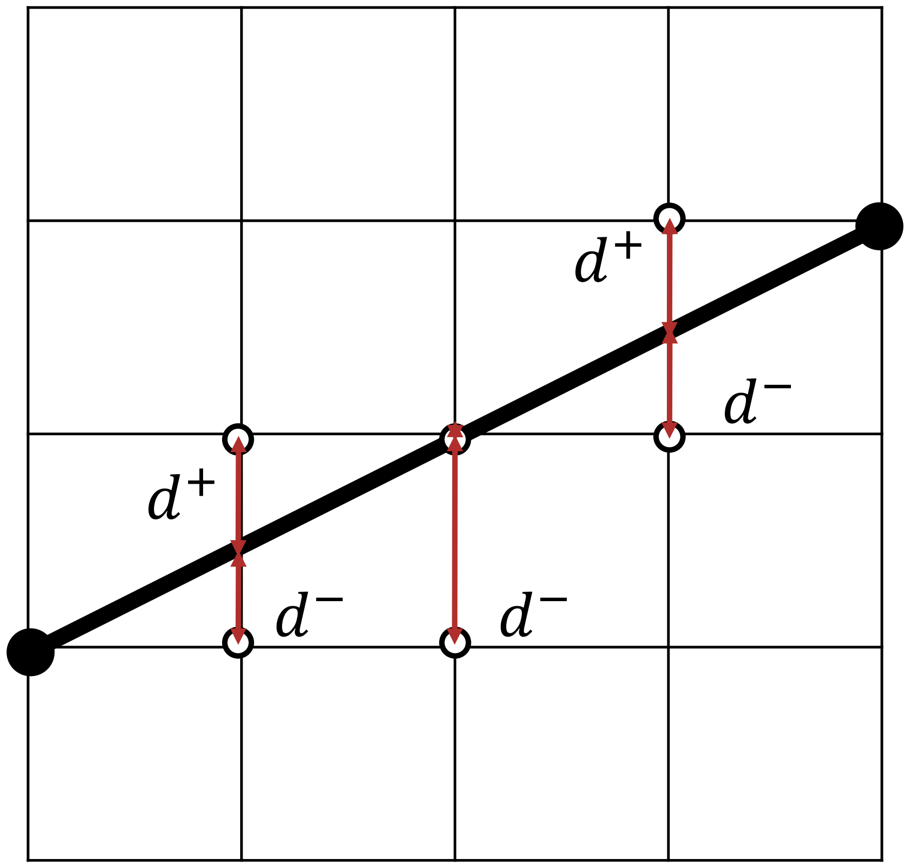
    $d^- \mathrel{-}=$

    $E \;\; \mathrel{+}=$



## 3.2 Rasterung von Geraden: Bresenham

- Zur schnelleren Berechnung, kann E **inkrementell** berechnet werden
- Das **Vorzeichen von E** dient dann als Kriterium für die Rundung auf den nächsten Rasterpunkt und die Aktualisierung von E:
  - $E \leq 0: x++,$
    $$d^+ \mathrel{-}= a$$
    $$d^- \mathrel{+}= a$$
    $$E \mathrel{+}= 2a$$
  - $E > 0: x++, y++,$
    $$d^+ \mathrel{+}= 1 - a$$
    $$d^- \mathrel{-}= 1 - a$$
    $$E \mathrel{+}= 2a - 2$$



# 3.2 Rasterung von Geraden: Bresenham

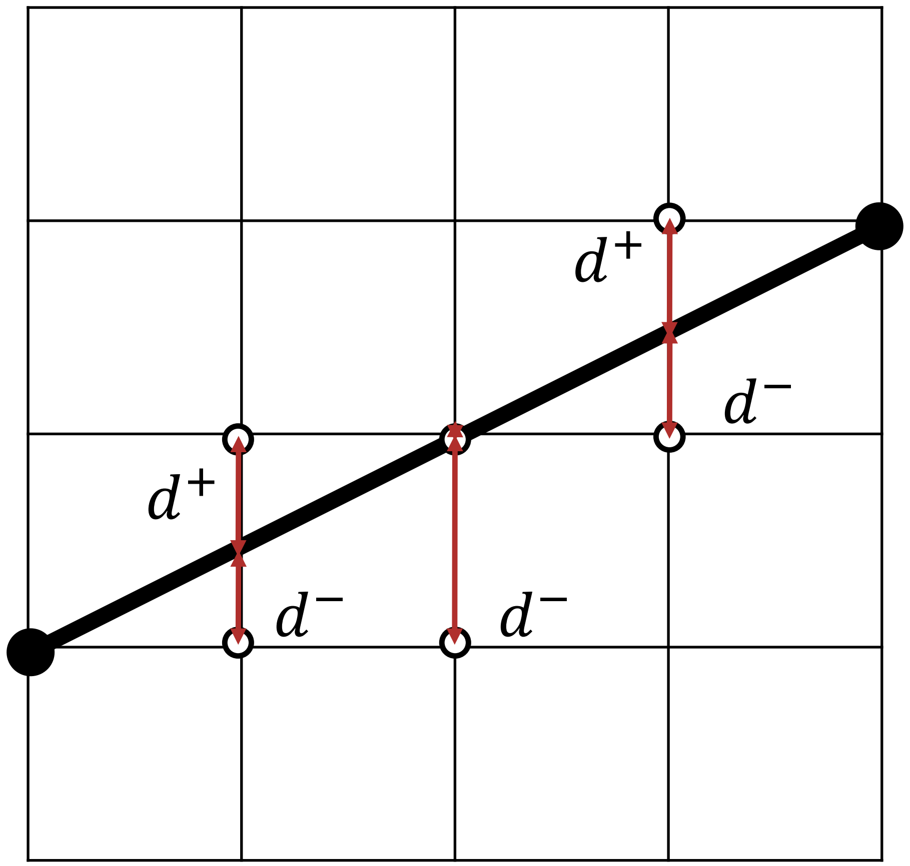
- Zur Vermeidung der Division wird **Entscheidungsgröße e** eingeführt:

$$e := \Delta x E$$

- Gehe nach oben, wenn

$$E > 0 \Leftrightarrow e > 0$$

- Das **Vorzeichen von e** dient dann als Kriterium für die Rundung auf den nächsten Rasterpunkt:
  - $e \leq 0: x++,$
  - $e > 0: x++, y++.$



## 3.2 Rasterung von Geraden: Bresenham

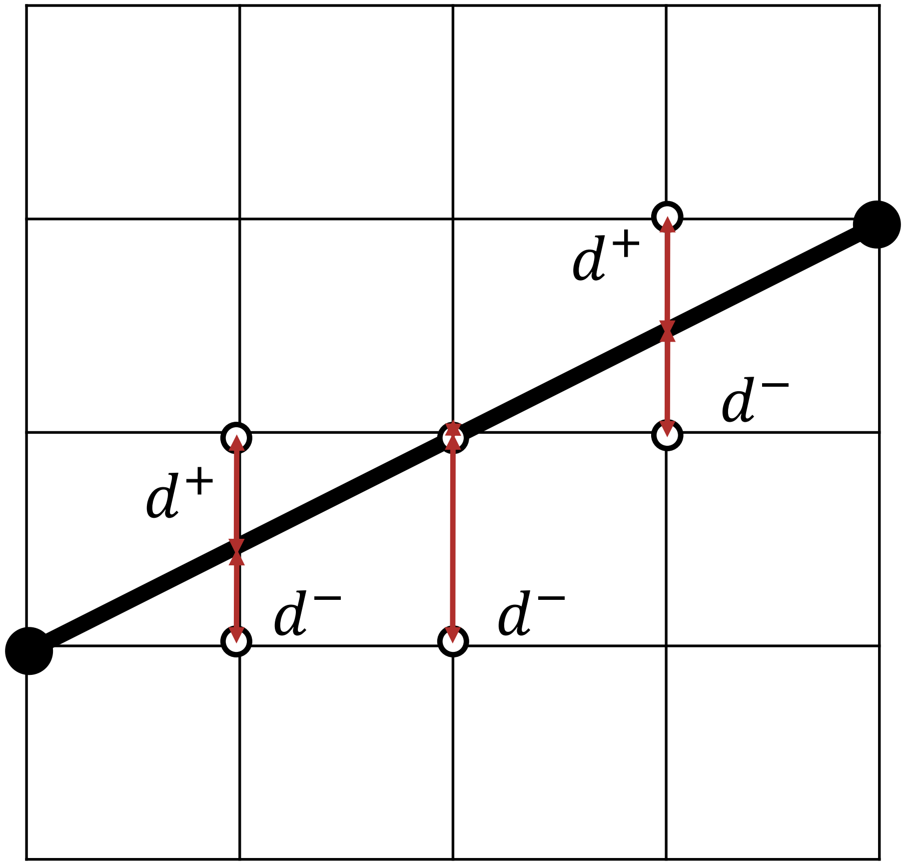
- Zur Vermeidung der Division wird **Entscheidungsgröße e** eingeführt:

$$e := \Delta x E$$

- Gehe nach oben, wenn

$$E > 0 \Leftrightarrow e > 0$$

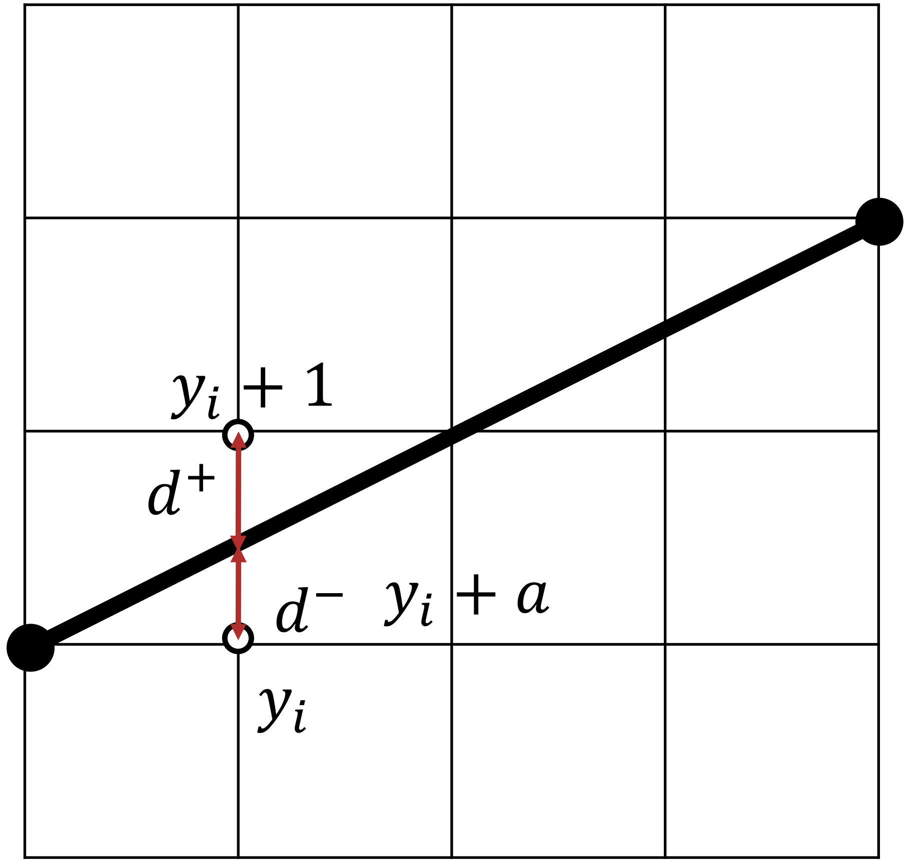
- Das **Vorzeichen von e** dient dann als Kriterium für die Rundung auf den nächsten Rasterpunkt:
  - $e \leq 0: x++, \qquad e += 2\Delta y$
  - $e > 0: x++, y++, e += 2\Delta y - 2\Delta x$



# 3.2 Rasterung von Geraden: Bresenham

- Startwerte:

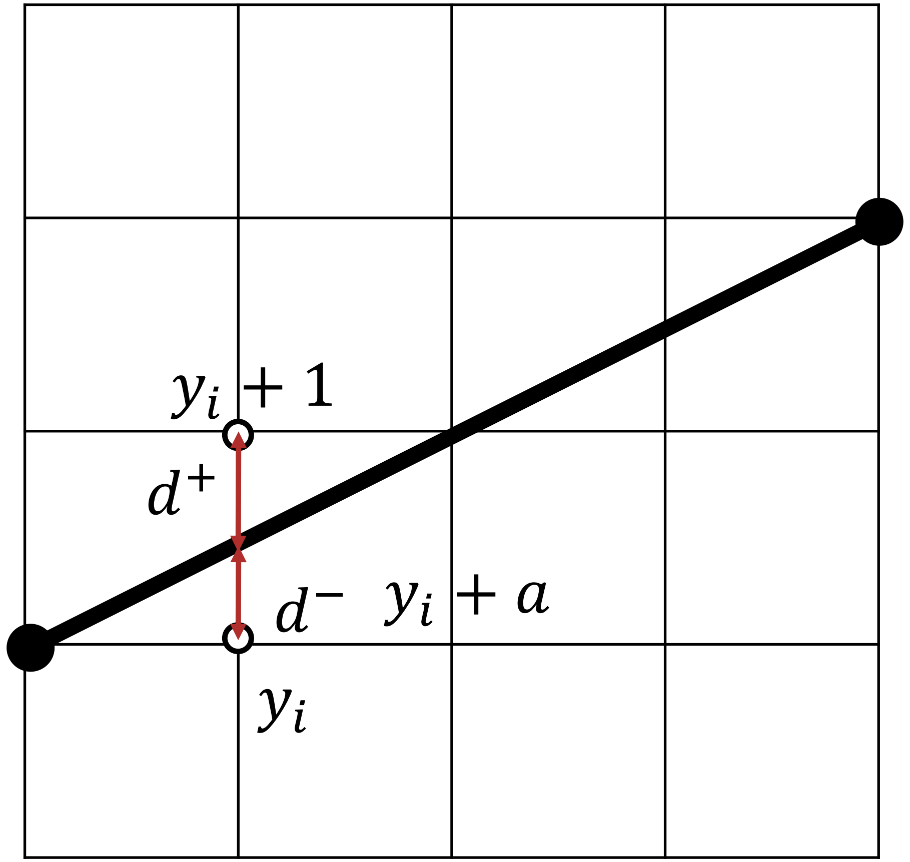
$$e_1$$



## 3.2 Rasterung von Geraden: Bresenham

– Startwerte:

$$\begin{aligned} e_1 &= \Delta x E_1 \\ &= \Delta x(d_1^- - d_1^+) \\ &= \Delta x(a - (1 - a)) \\ &= \Delta x(2a - 1) \\ &= \Delta x(2\frac{\Delta y}{\Delta x} - 1) \\ &= 2\Delta y - \Delta x \end{aligned}$$



# 3.2 Rasterung von Geraden: Bresenham

```
// (x1, y1), (x2, y2) Ganzzahlig
// x1 < x2, y1 < y2
x = x1; y = y1;
dx = x2 - x1; dy = y2 - y1;
e = 2 * dy - dx; // Initialisierung
for(i = 1; i <= dx; i++){
  // Schleife fuer x
  plot(x, y);
  if (e >= 0) {
        // oberen Punkt Zeichnen (y erhoehen)
    ++y;
    e -= 2 * dx;
  }
  ++x;
  e += 2 * dy;
}
plot(x, y);
```

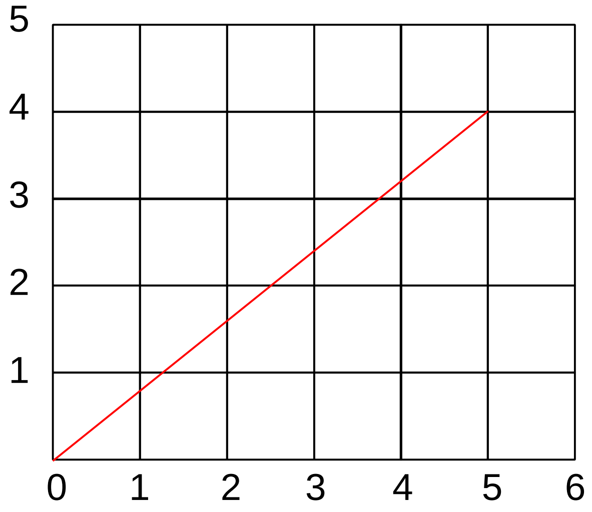
- Erster Oktant
- Ausschließlich ganzzahlige Operanden

# 3.2 Rasterung von Geraden: Bresenham

```
// (x1, y1), (x2, y2) Ganzzahlig
// x1 < x2, y1 < y2
x = x1; y = y1;
dx = x2 - x1; dy = y2 - y1;
e = 2 * dy - dx; // Initialisierung
for(i = 1; i <= dx; i++){
  // Schleife fuer x
  plot(x, y);
  if (e >= 0) {
        // oberen Punkt Zeichnen (y erhoehen)
    ++y;
    e -= 2 * dx;
  }
  ++x;
  e += 2 * dy;
}
plot(x, y);
```

**Beispiel:**

- P1=(0, 0)
- P2=(5, 4)



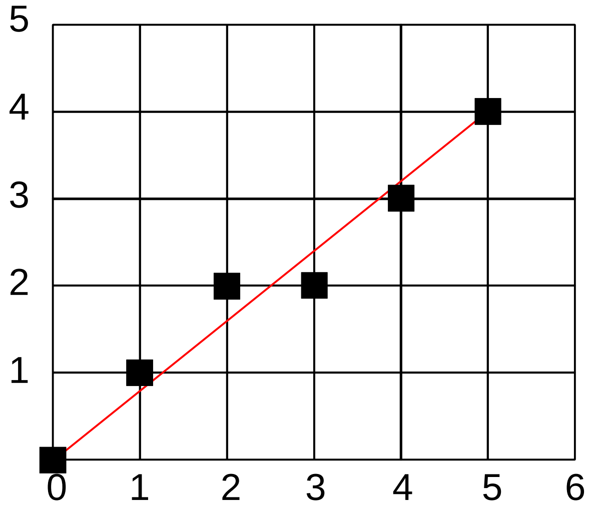

| x | y | e | i | plot |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# 3.2 Rasterung von Geraden: Bresenham

```
// (x1, y1), (x2, y2) Ganzzahlig
// x1 < x2, y1 < y2
x = x1; y = y1;
dx = x2 - x1; dy = y2 - y1;
e = 2 * dy - dx; // Initialisierung
for(i = 1; i <= dx; i++){
  // Schleife fuer x
  plot(x, y);
  if (e >= 0) {
        // oberen Punkt Zeichnen (y erhoehen)
    ++y;
    e -= 2 * dx;
  }
  ++x;
  e += 2 * dy;
}
plot(x, y);
```

**Beispiel:**

- P1=(0, 0)
- P2=(5, 4)

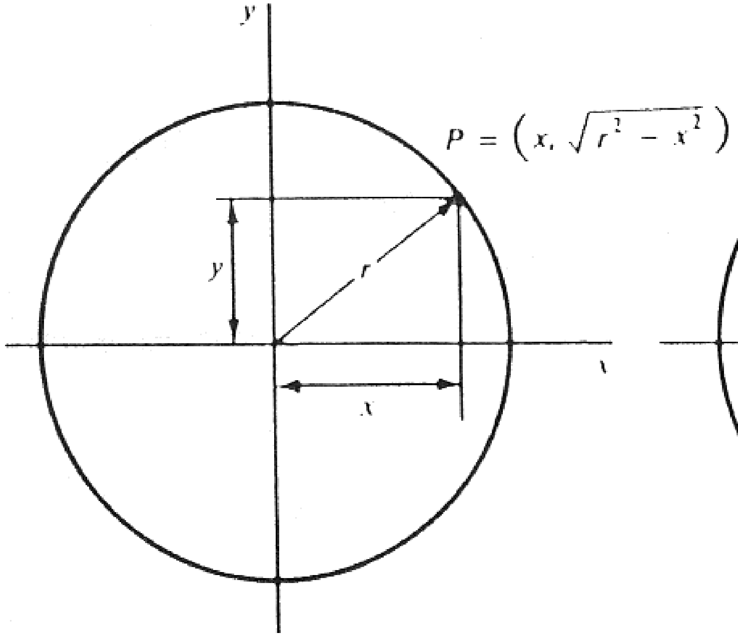


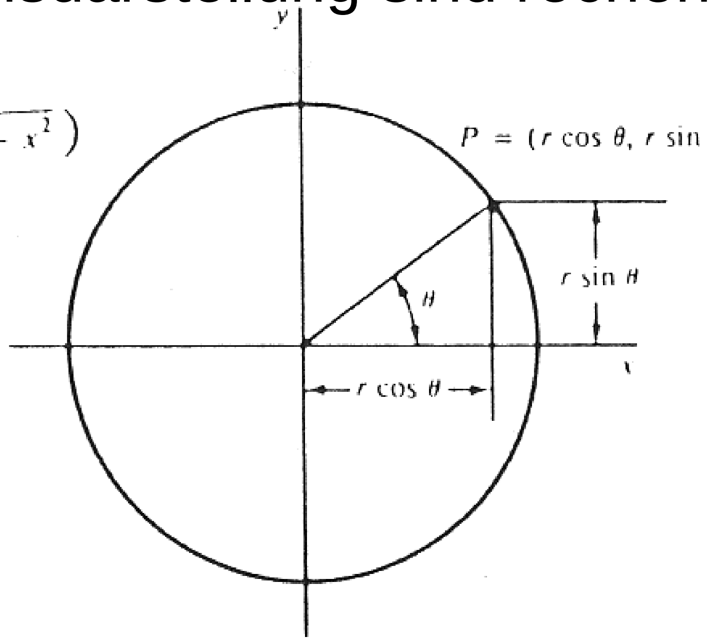

| x | y | e | i | plot |
|---|---|---|---|---|
| 0 | 0 | 3 | 1 | (0, 0) |
| | 1 | -7 | | |
| 1 | | 1 | 2 | (1, 1) |
| | 2 | -9 | | |
| 2 | | -1 | 3 | (2, 2) |
| 3 | | 7 | 4 | (3, 2) |
| | 3 | -3 | | |
| 4 | | 5 | 5 | (4, 3) |
| | 4 | -5 | | |
| 5 | | 3 | 6 | |
| | | | | (5, 4) |

# 3.3 Rasterung von Kreisen

Darstellung eines Kreises mit Mittelpunkt $(x_M, y_M)$ und Radius $r$

- Implizit: $f(x,y) = (x - x_M)^2 + (y - y_M)^2 - r^2 = 0$
- Parameter: $x(\theta) = x_M + r \cdot \cos\theta$, $\mathrm{y}(\theta) = y_M + r \cdot \sin\theta$, $\theta \in [0,2\pi[$
- Nachteil: Beide Methoden zur Kreisdarstellung sind rechenaufwändig



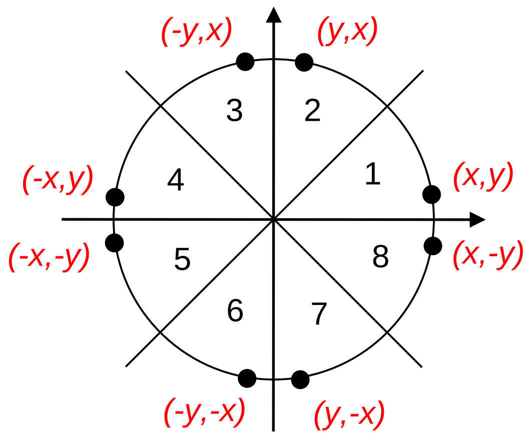



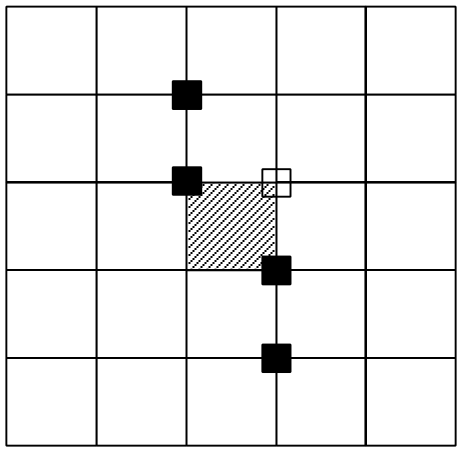
## 3.3 Rasterung von Kreisen

### Darstellung eines Kreises: Bemerkungen

- Mit der Berechnung eines Kreispunktes sind durch **Symmetrie** 7 weitere Kreispunkte gegeben
- Für eine gleichmäßige Rasterung müssen die Pixel entlang des Kreises **möglichst gleichmäßig** verteilt sein
- Die Bewertung der Kreisapproximation im Raster ist subjektiv

- von jedem Rasterquadrant dürfen nur 2 Eckpunkte gesetzt werden



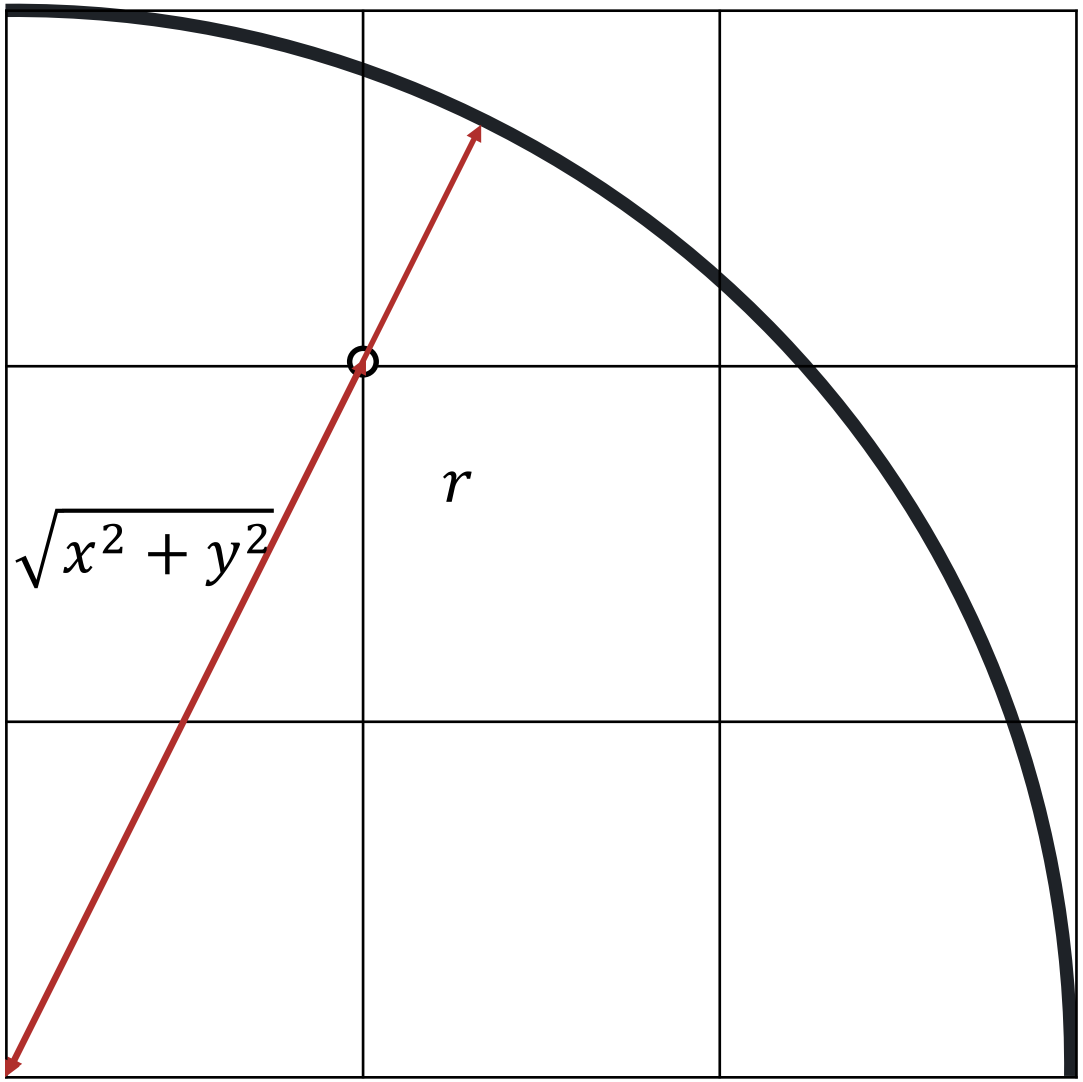



## 3.3 Rasterung von Kreisen

Darstellung eines Kreises: Bemerkungen

- Ein häufig verwendetes Kriterium ist die **Minimierung des Residuums**

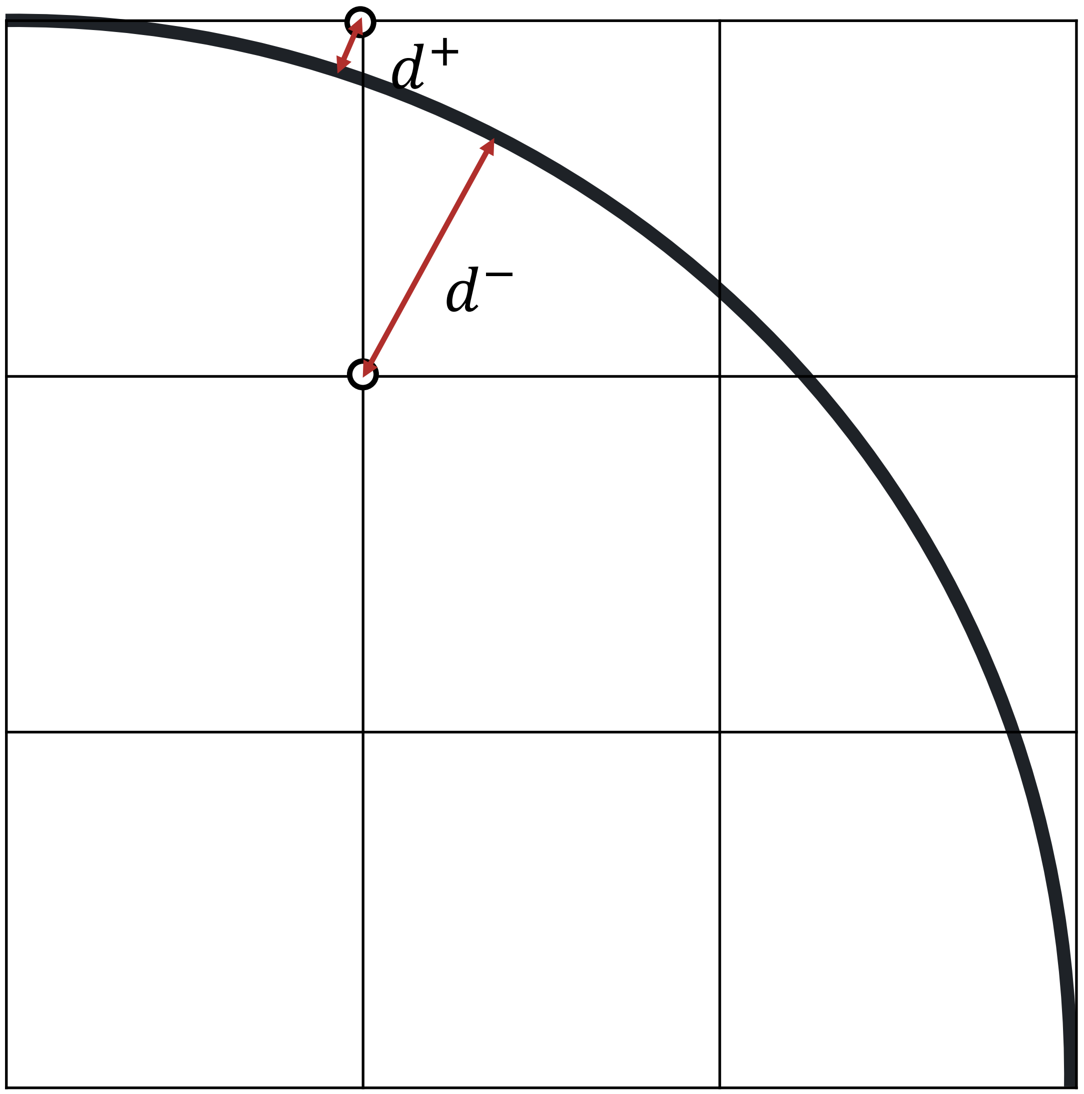
$$d = |x^2 + y^2 - r^2|$$



## 3.3 Rasterung von Kreisen

Bresenham-Algorithmus für Kreise

- Innerhalb des 2. Oktanten
  - Kreisfunktion monoton fallend
  - Steigung zwischen 0 und -1
  - $P^+$ liegt immer außerhalb der Kreislinie
  - $P^-$ liegt immer innerhalb der Kreislinie



## 3.3 Rasterung von Kreisen

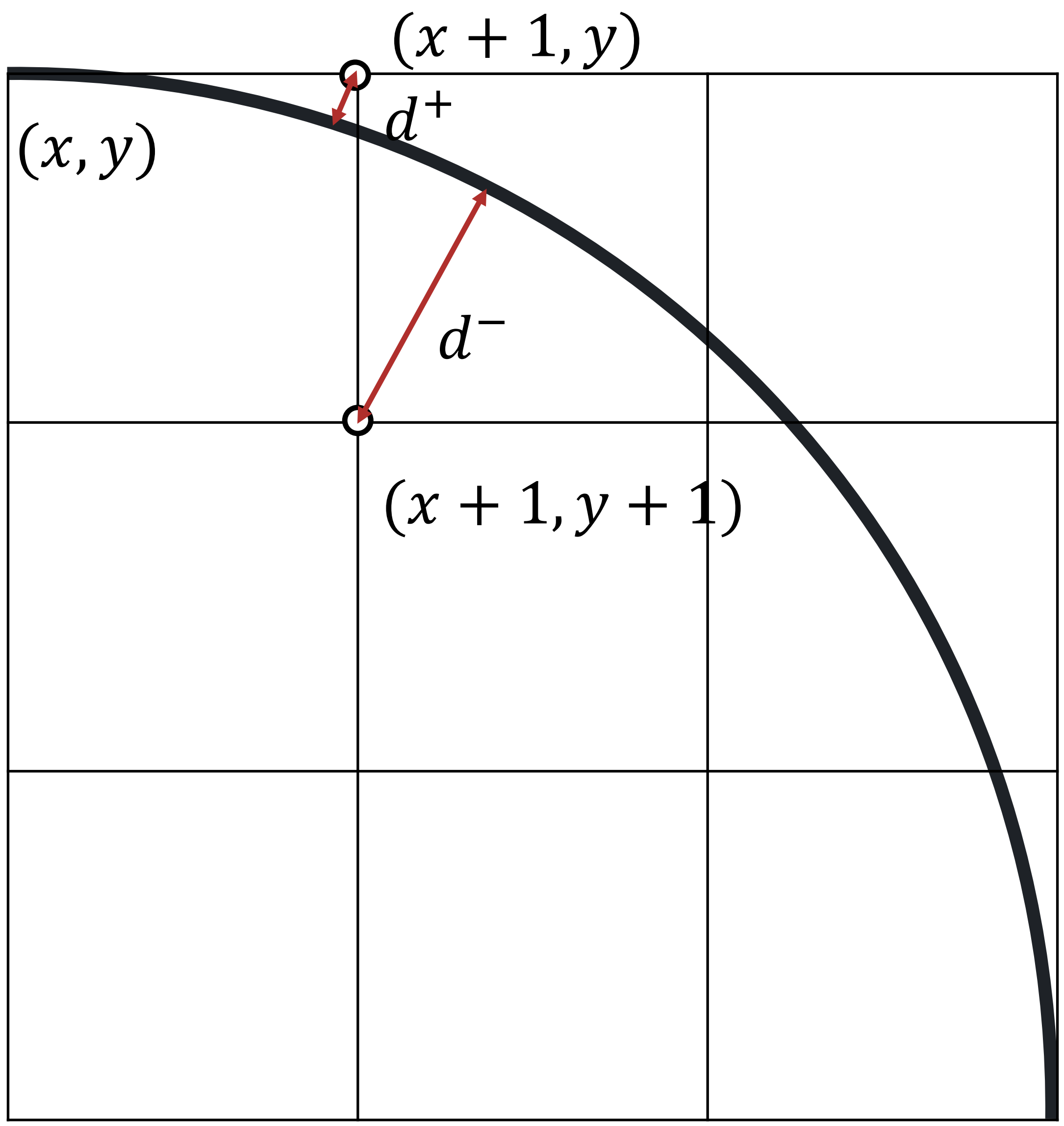
Bresenham-Algorithmus für Kreise

- Hier:
  - $(x_M, y_M) = (0,0)$
  - Startpunkt ist Rasterpunkt
  - $r \in \mathbb{N}$
  - 2. Oktant
- Gehe nach unten, wenn $d^+ > d^-$

$d^+ \quad =$

$\quad =$

$d^- \quad =$



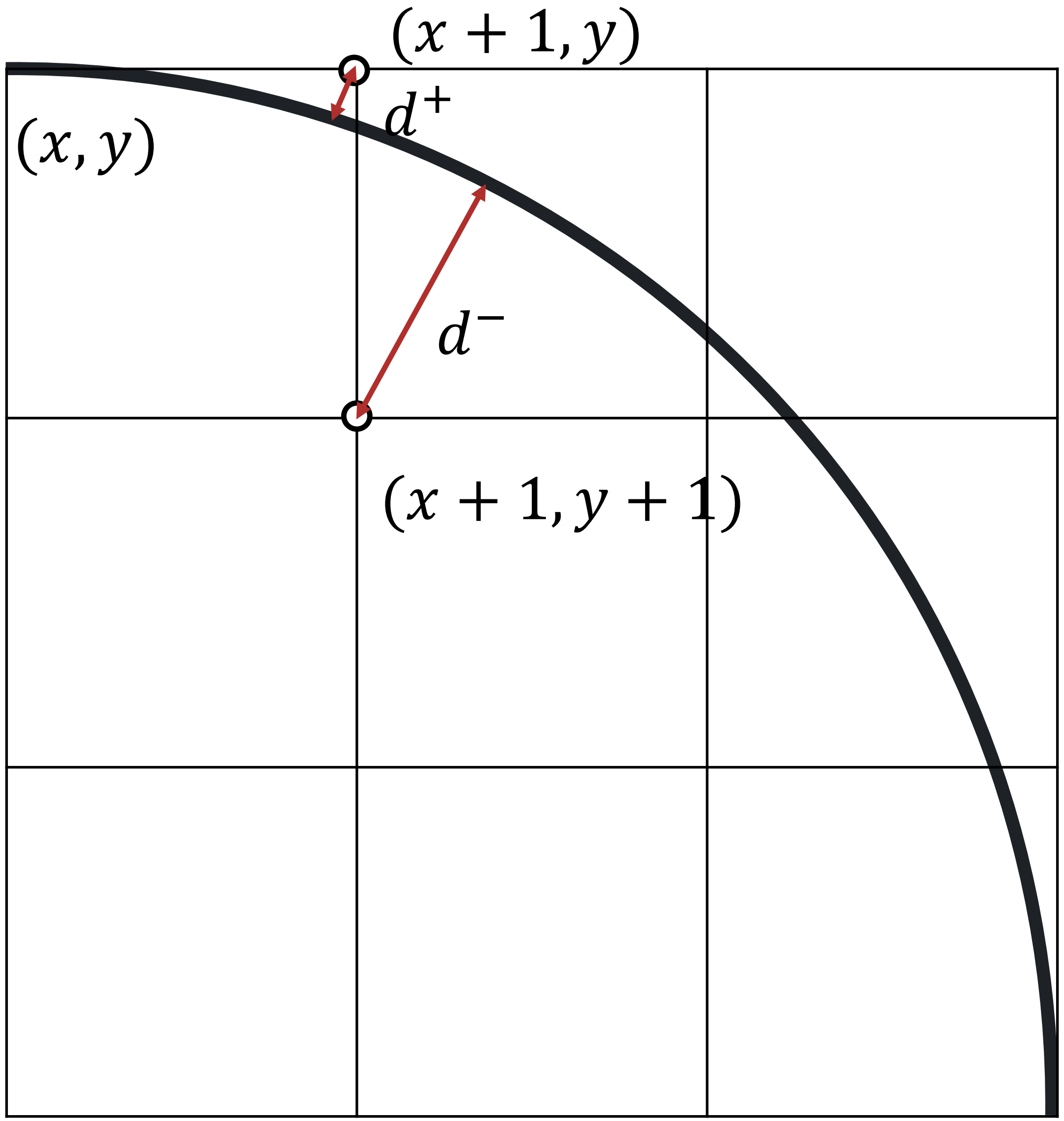
## 3.3 Rasterung von Kreisen

### Bresenham-Algorithmus für Kreise

- Hier:
  - $(x_M, y_M) = (0{,}0)$
  - Startpunkt ist Rasterpunkt
  - $r \in \mathbb{N}$
  - 2. Oktant
- Gehe nach unten, wenn $d^+ > d^-$

$$
\begin{aligned}
d^+ &= |(x_i + 1)^2 + y_i^2 - r^2| \\
&= (x_i + 1)^2 + y_i^2 - r^2
\end{aligned}
$$

$$
\begin{aligned}
d^- &= |(x_i + 1)^2 + (y_i - 1)^2 - r^2| \\
&= r^2 - (x_i + 1)^2 - (y_i - 1)^2
\end{aligned}
$$



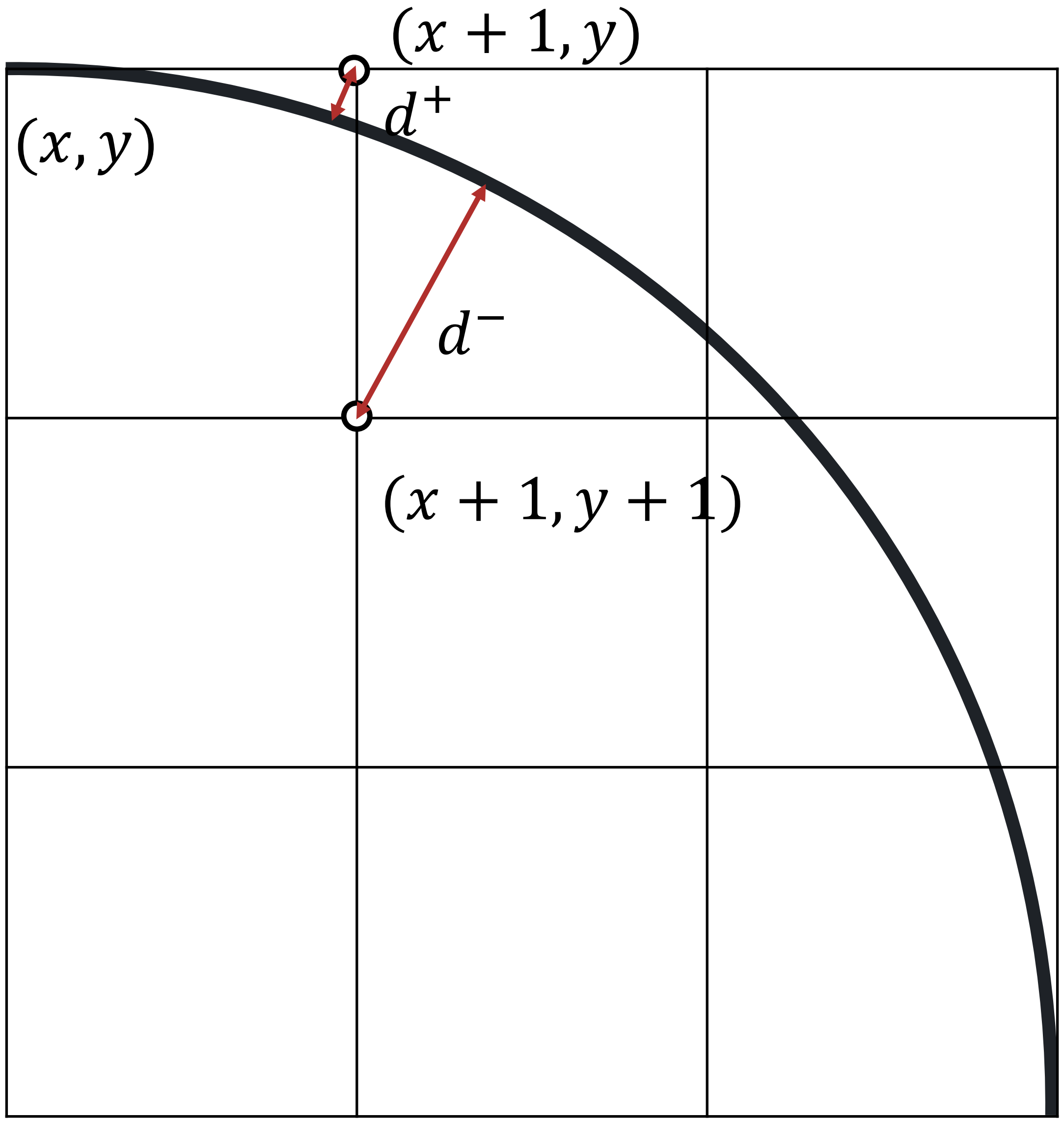
## 3.3 Rasterung von Kreisen

Bresenham-Algorithmus für Kreise

- Beste Approximation über **Entscheidungsgröße E** mittels minimalem Residuum $E = d^+ - d^-$
- Gehe nach unten, wenn $d^+ > d^- \Leftrightarrow E > 0$

$E \leq 0: x++,$

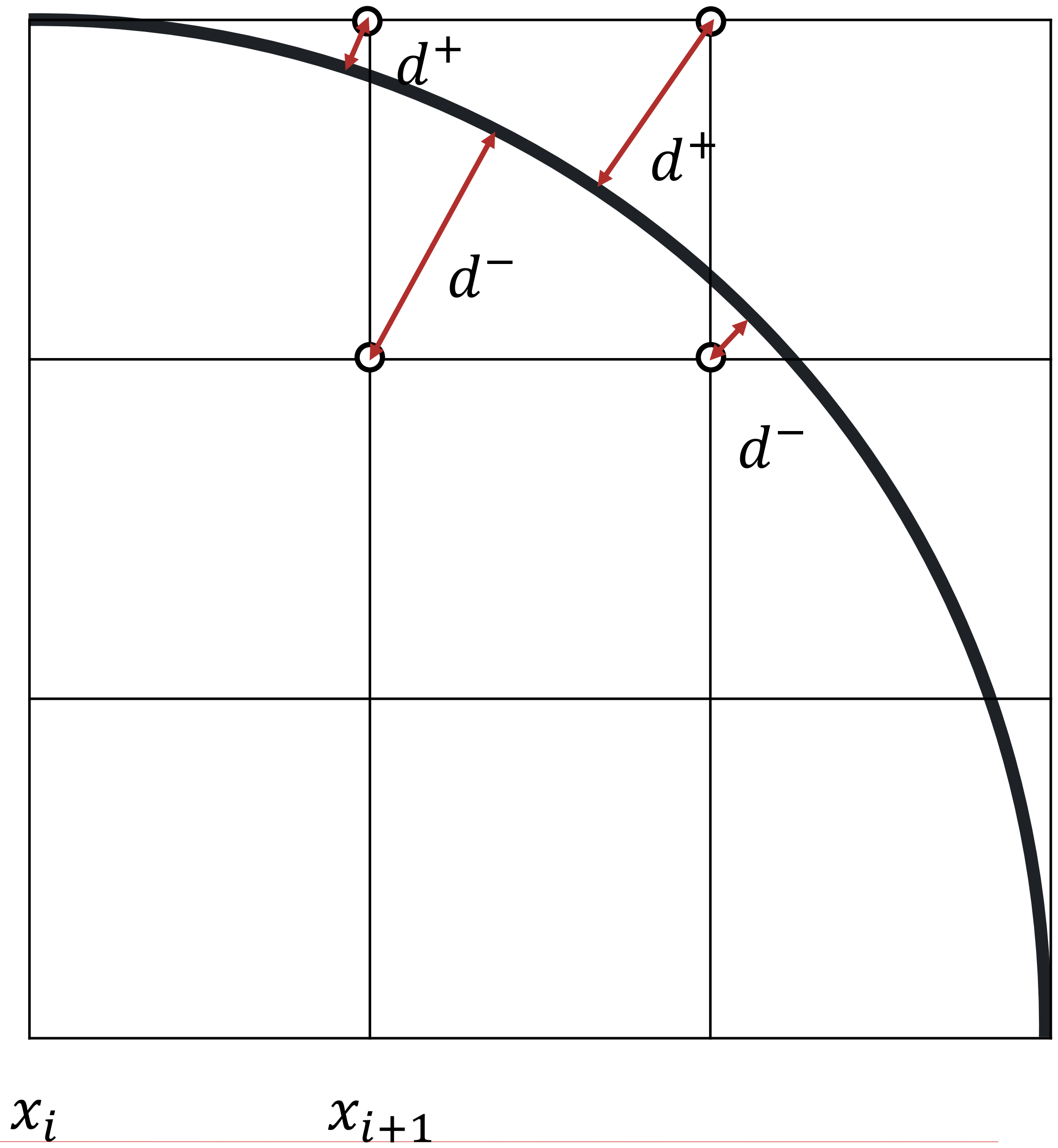
$E > 0: x++, y--,$



## 3.3 Rasterung von Kreisen

# Bresenham-Algorithmus für Kreise

- Zur schnelleren Berechnung kann E **inkrementell** bestimmt werden



# 3.3 Rasterung von Kreisen

## Bresenham-Algorithmus für Kreise

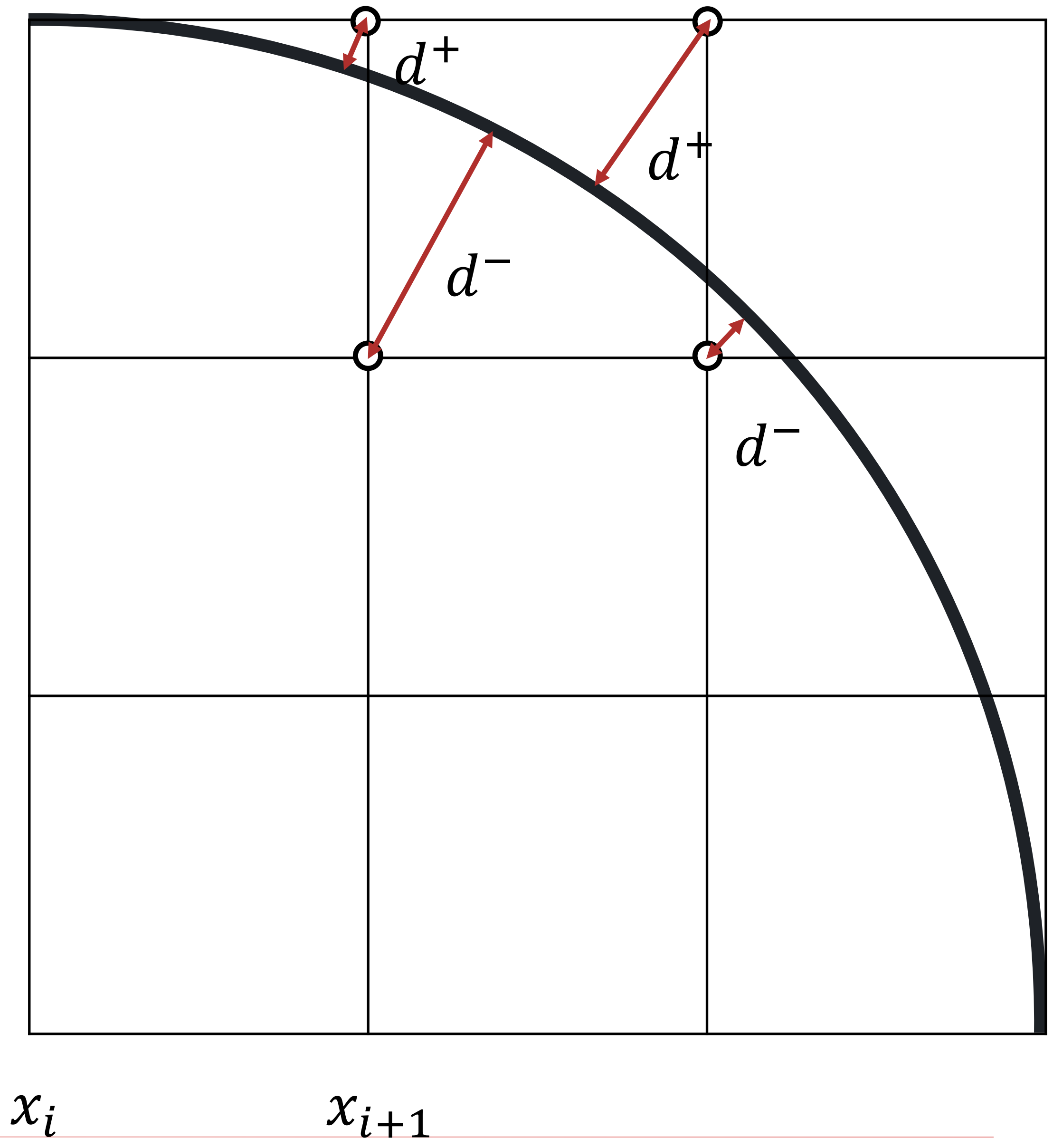
– Zur schnelleren Berechnung kann E **inkrementell** bestimmt werden
– $E \leq 0: x++$

$$
\begin{aligned}
d_{i+1}^{+} &= (x_{i+1}+1)^2 + y_{i+1}^2 - r^2 \\
&= (x_i+1+1)^2 + y_i^2 - r^2 \\
&= (x_i+1)^2 + 2(x_i+1) + 1 + y_i^2 - r^2 \\
&= d_i^{+} + 2x_i + 3 \\
&= d_i^{+} + 2x_{i+1} + 1
\end{aligned}
$$

$$
\begin{aligned}
d_{i+1}^{-} &= r^2 - (x_{i+1}+1)^2 - (y_{i+1}-1)^2 \\
&= r^2 - (x_i+1+1)^2 - (y_i-1)^2 \\
&= r^2 - (x_i+1)^2 - 2(x_i+1) - 1 - (y_i-1)^2 \\
&= d_i^{-} - 2x_i - 3 \\
&= d_i^{-} - 2x_{i+1} - 1
\end{aligned}
$$

$E_{i+1}$

:



## 3.3 Rasterung von Kreisen

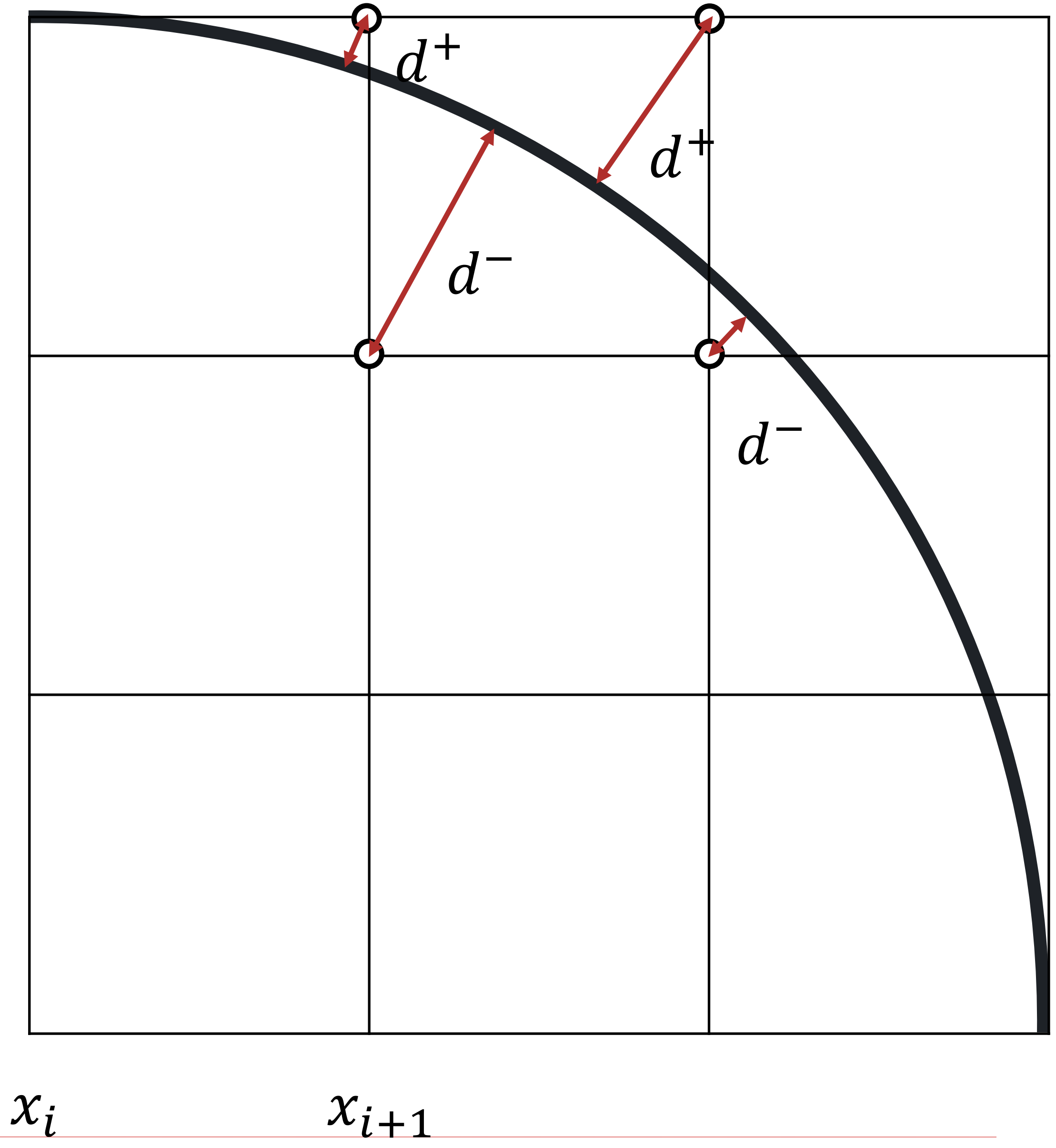
# Bresenham-Algorithmus für Kreise

- Zur schnelleren Berechnung kann E **inkrementell** bestimmt werden
- $E \leq 0: x++$

$$
\begin{aligned}
d_{i+1}^{+} &= (x_{i+1}+1)^2 + y_{i+1}^2 - r^2 \\
&= (x_i+1+1)^2 + y_i^2 - r^2 \\
&= (x_i+1)^2 + 2(x_i+1) + 1 + y_i^2 - r^2 \\
&= d_i^{+} + 2x_i + 3 \\
&= d_i^{+} + 2x_{i+1} + 1
\end{aligned}
$$

$$
\begin{aligned}
d_{i+1}^{-} &= r^2 - (x_{i+1}+1)^2 - (y_{i+1}-1)^2 \\
&= r^2 - (x_i+1+1)^2 - (y_i-1)^2 \\
&= r^2 - (x_i+1)^2 - 2(x_i+1) - 1 - (y_i-1)^2 \\
&= d_i^{-} - 2x_i - 3 \\
&= d_i^{-} - 2x_{i+1} - 1
\end{aligned}
$$

$$
\begin{aligned}
E_{i+1} &= d_{i+1}^{+} - d_{i+1}^{-} \\
&= d_i^{+} + 2x_{i+1} + 1 - d_i^{-} + 2x_{i+1} + 1 \\
&= E_i + 4x_{i+1} + 2
\end{aligned}
$$



## 3.3 Rasterung von Kreisen

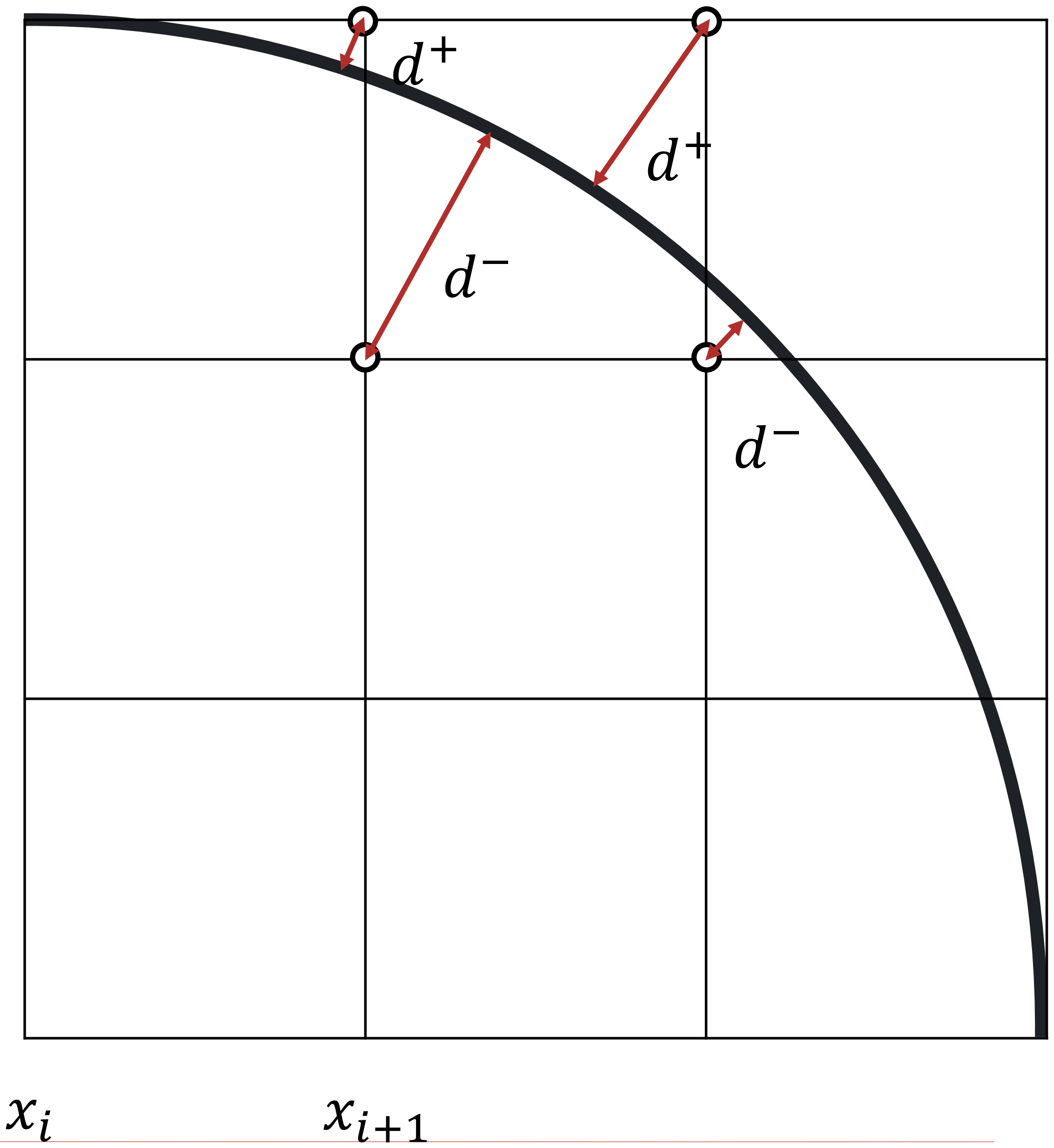
# Bresenham-Algorithmus für Kreise

- Zur schnelleren Berechnung kann E **inkrementell** bestimmt werden
- $E > 0: x++, y--$

$$d_{i+1}^{+} = (x_{i+1}+1)^2 + y_{i+1}^2 - r^2$$
$$= (x_i + 1 + 1)^2 + (y_i - 1)^2 - r^2$$
$$= (x_i+1)^2 + 2(x_i+1) + 1 + y_i^2 - 2y_i + 1 - r^2$$
$$= d_i^{+} + 2x_i - 2y_i + 4$$
$$= d_i^{+} + 2x_{i+1} - 2y_{i+1}$$

$$d_{i+1}^{-} = r^2 - (x_{i+1}+1)^2 - (y_{i+1}-1)^2$$
$$= r^2 - (x_i+1+1)^2 - (y_i - 1 - 1)^2$$
$$= r^2 - (x_i+1)^2 + 2(x_i+1) + 1 - (y_i-1)^2 + 2(y_1 - 1) - 1$$
$$= d_i^{-} - 2x_i + 2y_i - 6$$
$$= d_i^{-} - 2x_{i+1} + 2y_{i+1} - 2$$

$$E_{i+1} = d_{i+1}^{+} - d_{i+1}^{-}$$
$$= d_i^{+} + 2x_{i+1} - 2y_{i+1} - d_i^{-} + 2x_{i+1} - 2y_{i+1} + 2$$
$$= E_i + 4(x_i - y_i) + 2$$



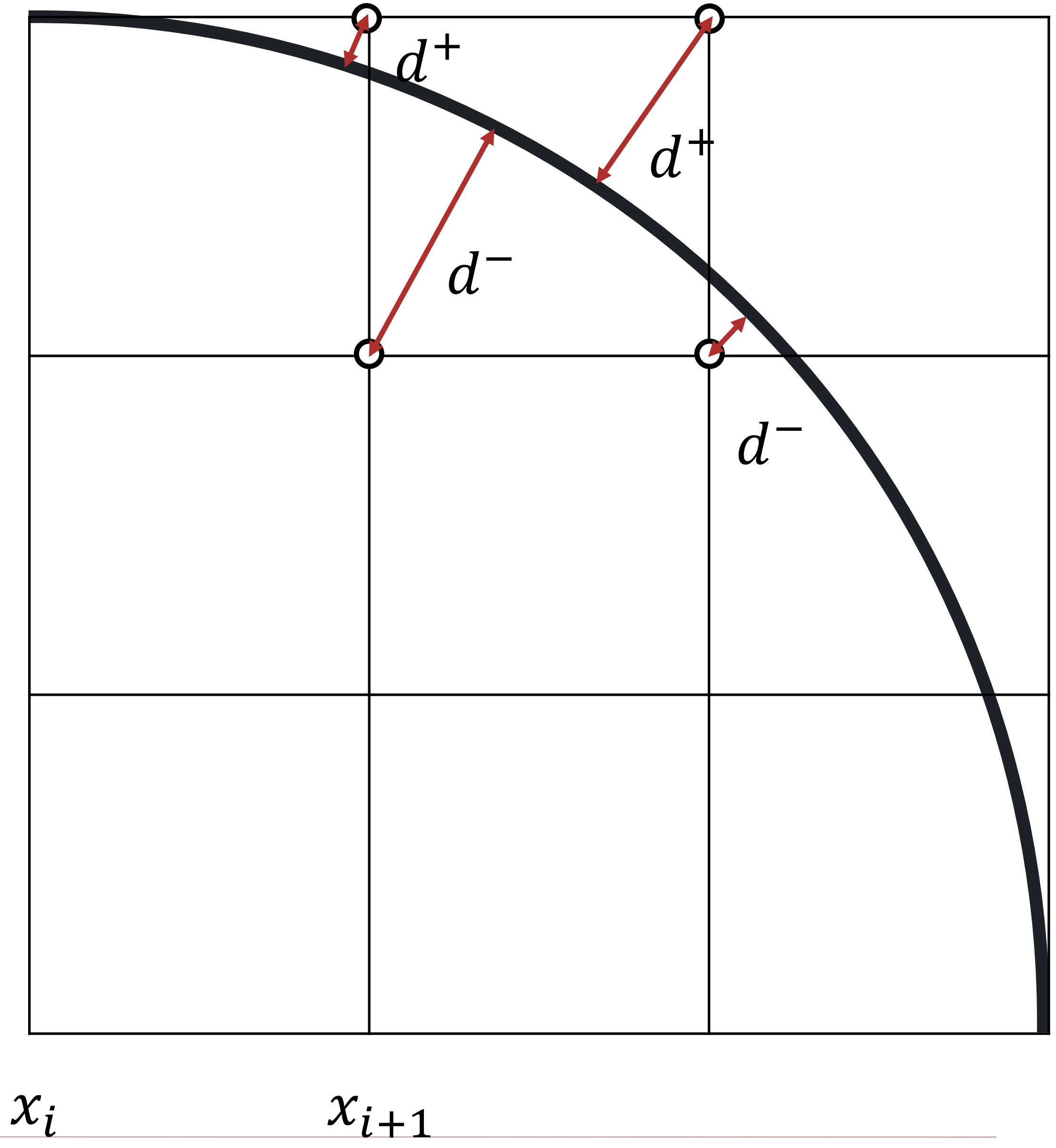
## 3.3 Rasterung von Kreisen

Bresenham-Algorithmus für Kreise

- Aktualisierung:
- $E \leq 0: x++,$

  $$E_{i+1} = E_i + 4x_i + 2$$
- $E > 0: x++, y--,$

  $$E_{i+1} = E_i + 4(x_i - y_i) + 2$$



## 3.3 Rasterung von Kreisen

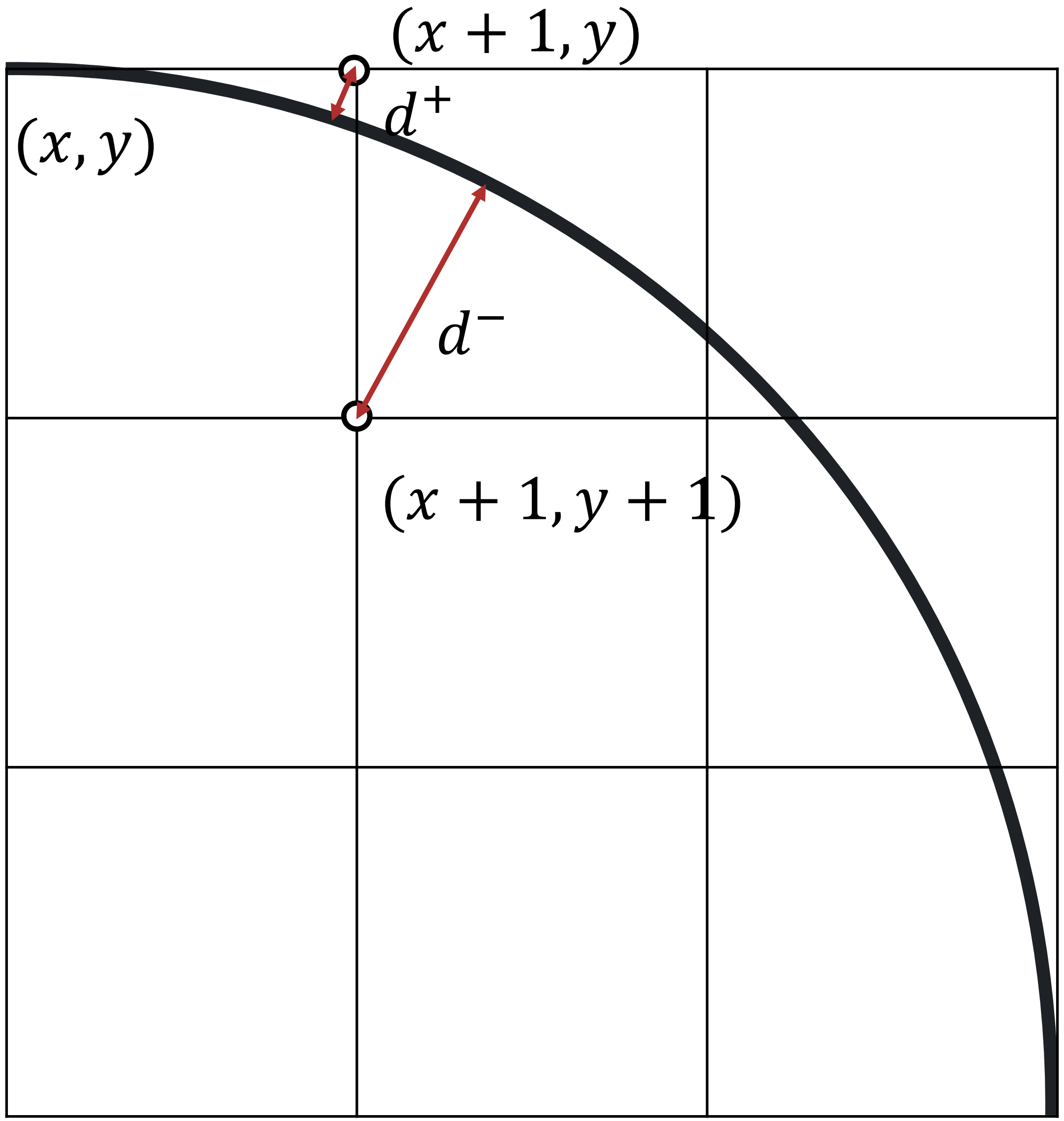
### Bresenham-Algorithmus für Kreise

- Startwerte:
  - $x_1 = 0$
  - $y_1 = r$

$d_1^+$

$d_1^-$

$E_1$



## 3.3 Rasterung von Kreisen

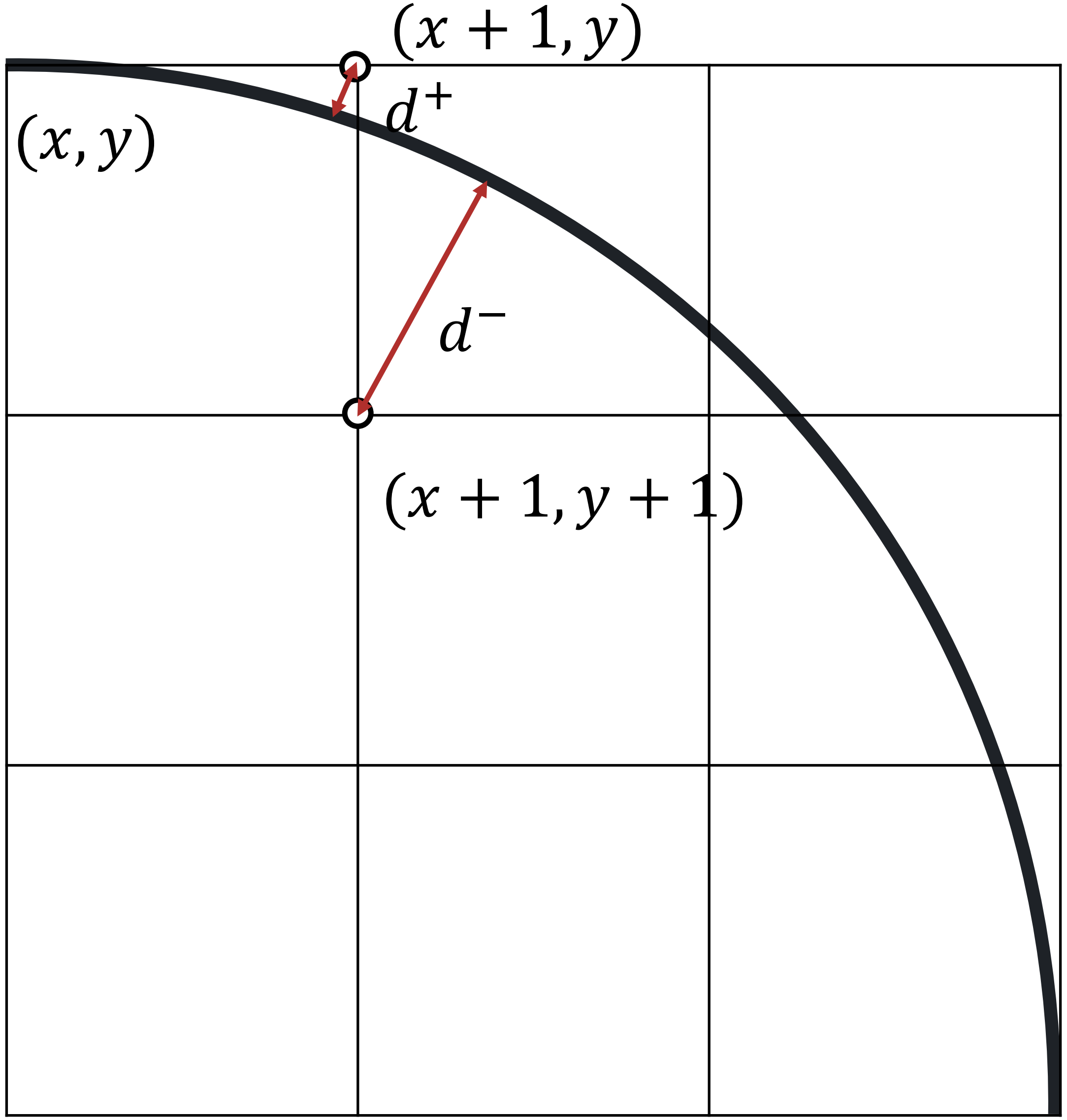
Bresenham-Algorithmus für Kreise

- Startwerte:
  - $x_1 = 0$
  - $y_1 = r$

$$d_1^+ = (x_1+1)^2 + y_1^2 - r^2 = 1 + r^2 - r^2 = 1$$

$$d_1^- = r^2 - (x_1+1)^2 - (y_1-1)^2 = r^2 - 1 - r^2 + 2r - 1 = 2r - 2$$

$E_1$



## 3.3 Rasterung von Kreisen

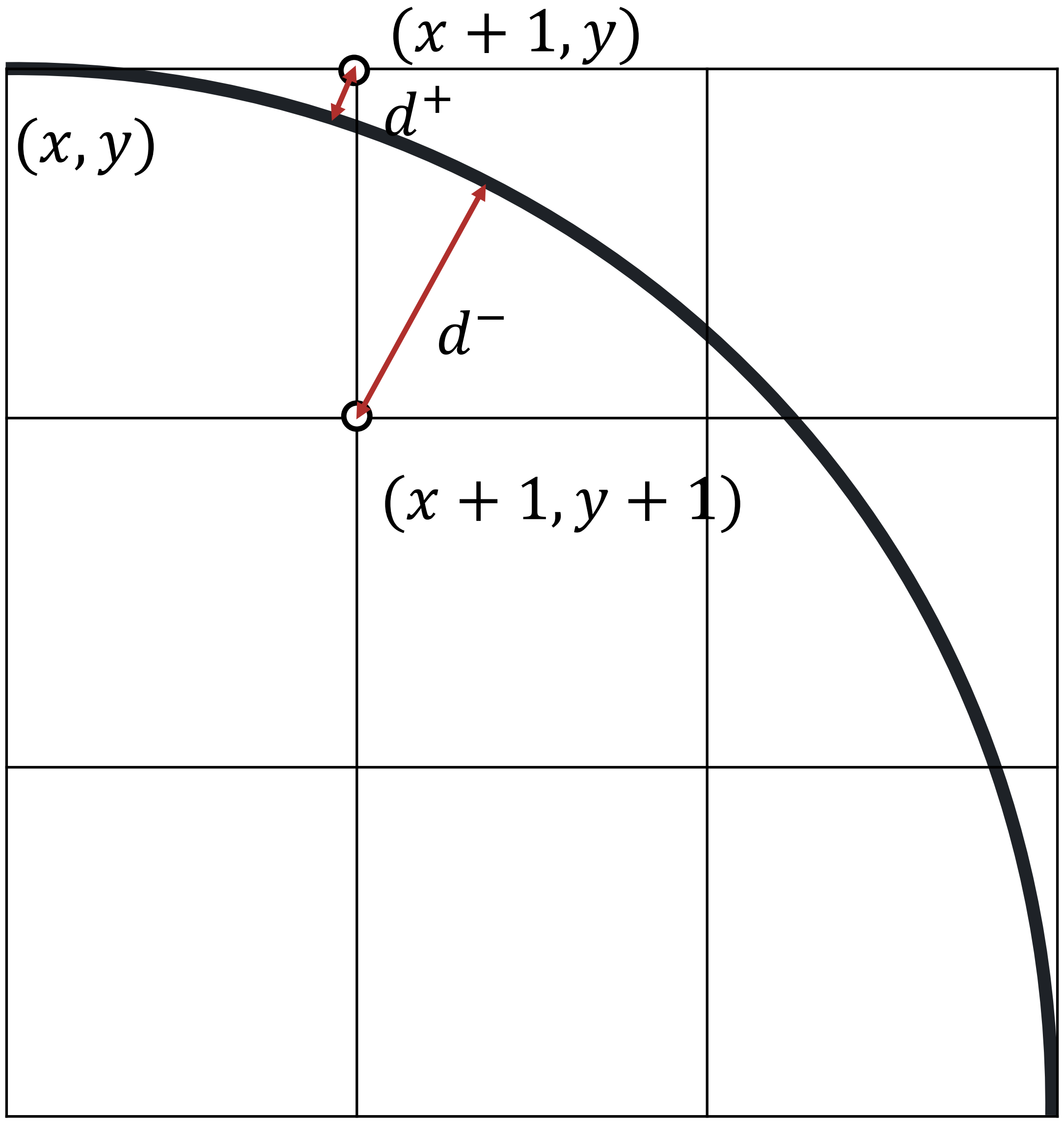
### Bresenham-Algorithmus für Kreise

- Startwerte:
  - $x_1 = 0$
  - $y_1 = r$

$$d_1^+ = (x_1 + 1)^2 + y_1^2 - r^2$$
$$= 1 + r^2 - r^2$$
$$= 1$$

$$d_1^- = r^2 - (x_1 + 1)^2 - (y_1 - 1)^2$$
$$= r^2 - 1 - r^2 + 2r - 1$$
$$= 2r - 2$$

$$E_1 = d_1^+ - d_1^-$$
$$= 1 - 2r + 2$$
$$= 3 - 2r$$



# 3.3 Rasterung von Kreisen

## Bresenham-Algorithmus für Kreise

```
x = 0
y = r
E = 3 - 2*r
zeichneAchtPunkte(x,y)
solange x < y wiederhole:
   x = x + 1
   falls E > 0 dann
      y = y - 1
      E = E + 4(x-y) + 2
   sonst E = E + 4x + 2
   zeichneAchtPunkte(x,y)
ende solange
```

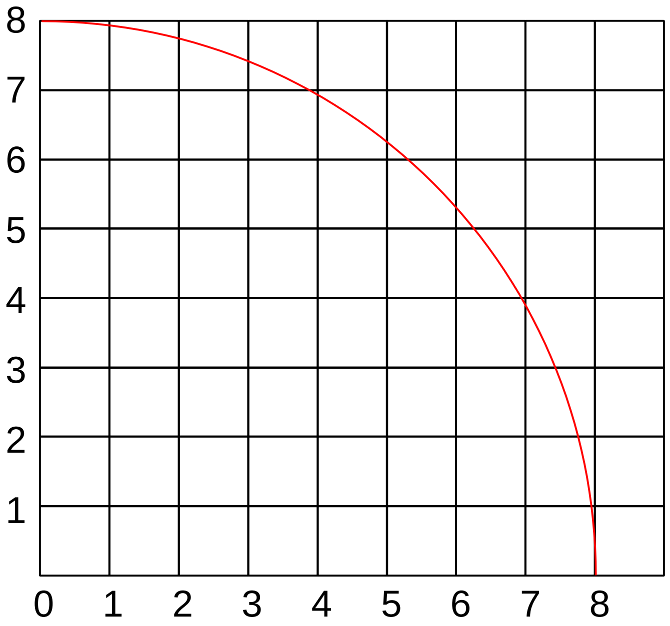
– 2. Oktant
– nur ganzzahlige Operatoren

## 3.3 Rasterung von Kreisen

### Bresenham-Algorithmus für Kreise

```
x = 0
y = r
E = 3 - 2*r
zeichneAchtPunkte(x,y)
solange x < y wiederhole:
   x = x + 1
   falls E > 0 dann
      y = y - 1
      E = E + 4(x-y) + 2
   sonst E = E + 4x + 2
   zeichneAchtPunkte(x,y)
ende solange
```

– Beispiel: $r = 8$



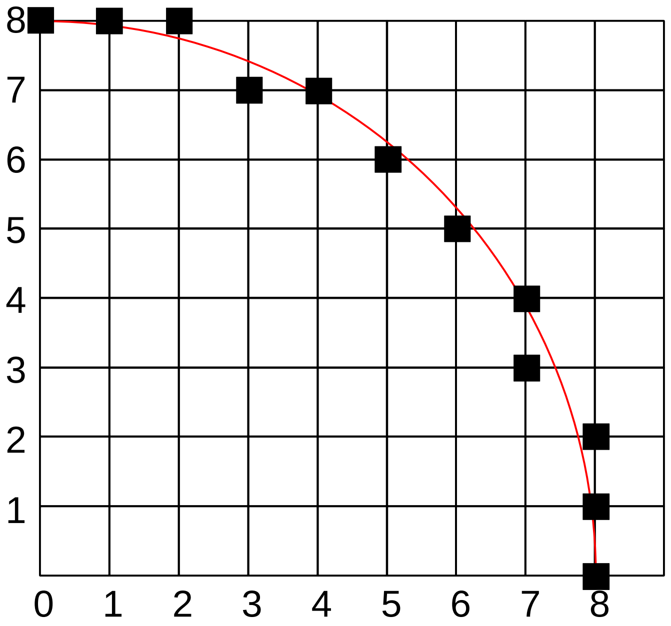

| x | y | E | plot |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

## 3.3 Rasterung von Kreisen

# Bresenham-Algorithmus für Kreise

```
x = 0
y = r
E = 3 - 2*r
zeichneAchtPunkte(x,y)
solange x < y wiederhole:
   x = x + 1
   falls E > 0 dann
      y = y - 1
      E = E + 4(x-y) + 2
   sonst E = E + 4x + 2
   zeichneAchtPunkte(x,y)
ende solange
```

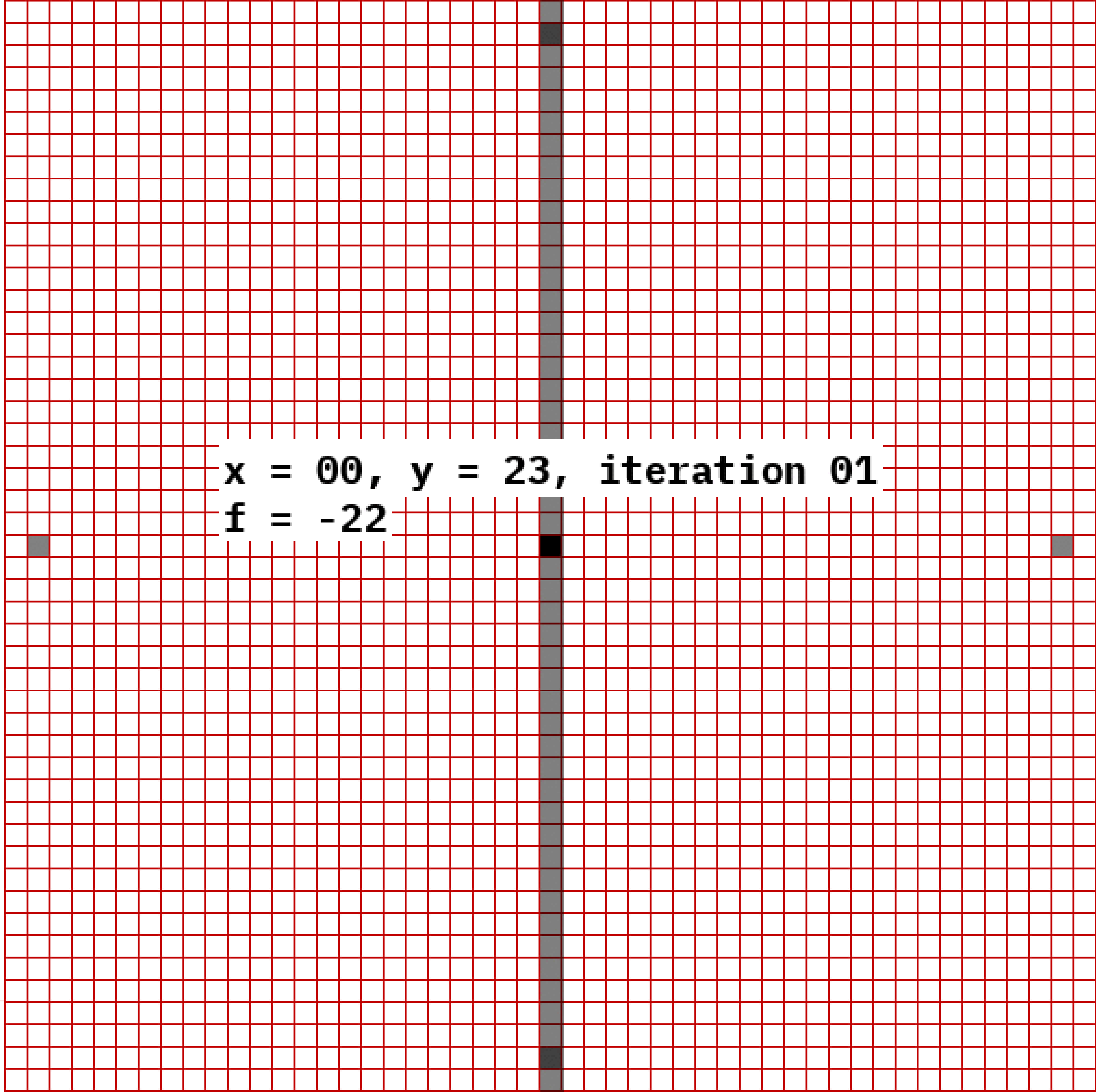
– Beispiel: $r = 8$




| x | y | E | plot |
|---|---|---|---|
| 0 | 8 | -13 | (0, 8) |
| 1 | 8 | -7 | (1, 8) |
| 2 | 8 | 3 | (2, 8) |
| 3 | 7 | -11 | (3, 7) |
| 4 | 7 | 7 | (4, 7) |
| 5 | 6 | 5 | (5, 6) |
| 6 | 5 | 11 | (6, 5) |

## 3.3 Rasterung von Kreisen

Bresenham-Algorithmus für Kreise

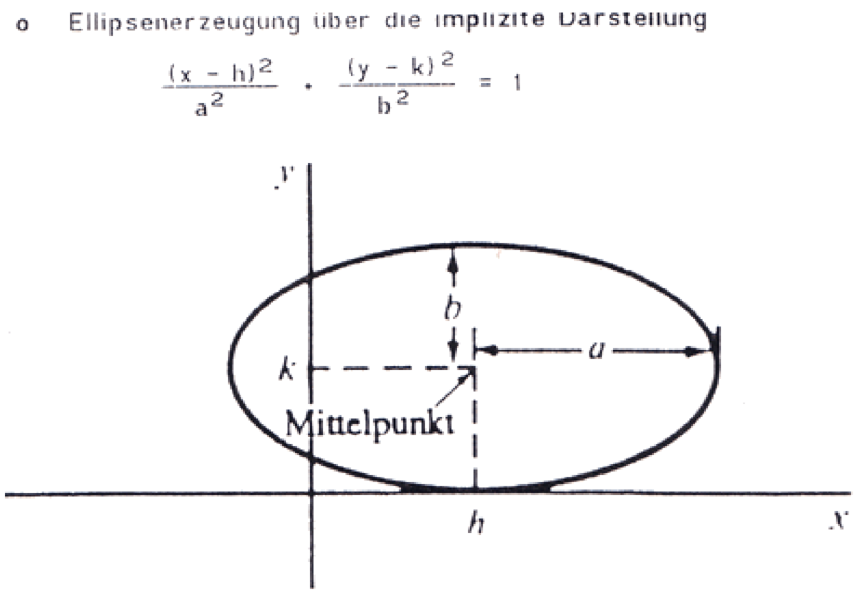


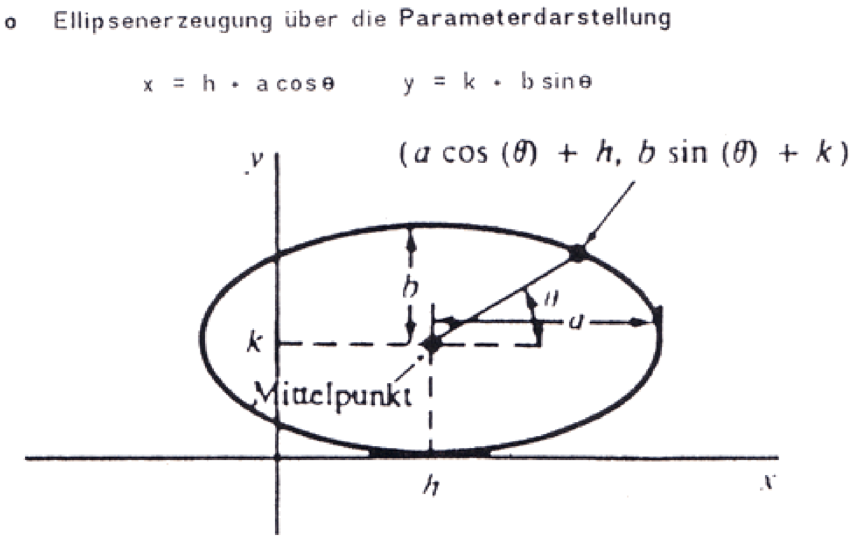
## 3.3 Rasterung von Kreisen

### Ellipsendarstellung

- Zur Rasterung betrachtet man die Ellipsen
  - in **Normalform**
  - mit zu den **Koordinatenachsen parallelen Hauptachsen**

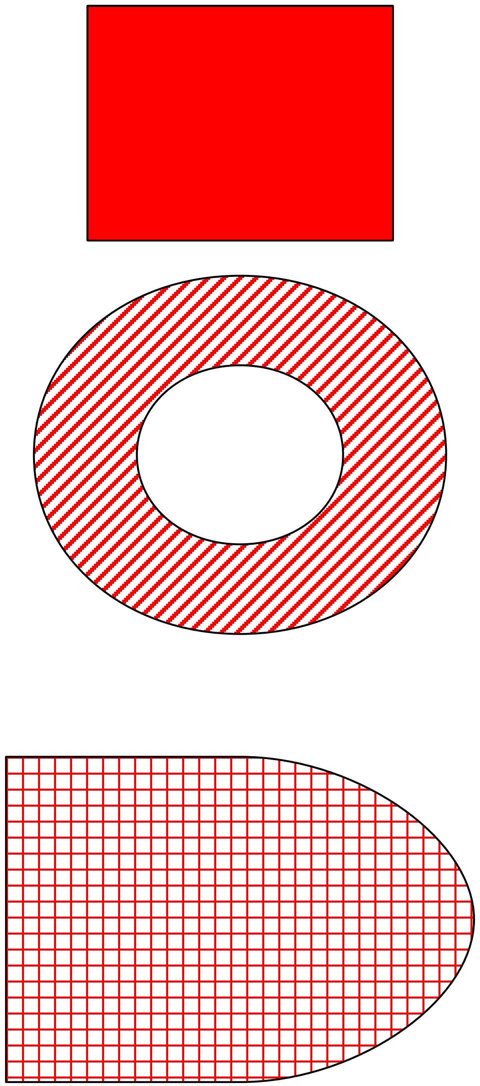
  → Algorithmus von Kappel



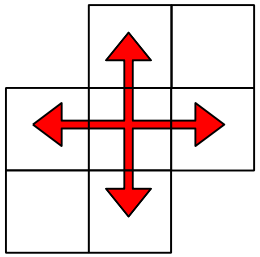
## 3.4 Füllalgorithmen

- Ziel: Füllen bzw. Einfärben eines begrenzten Bereiches oder Gebietes mit einer Füllfarbe oder einem Muster bzw. einer Schraffur
- Beispiele:
  - Balkendiagramme
  - Flächen
  - Körper
- Beschreibung der zu füllenden Gebiete erfolgt geometrisch,
  - Durch Ecken, Strecken, Polygone, Kreise (randdefiniert)
  - Durch Pixel (inhaltsdefiniert)

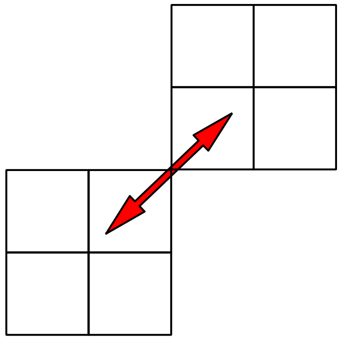


## Zusammenhang von Gebieten

- 4-fach zusammenhängend (nur Horizontal- und Vertikalbewegung)



- 8-fach zusammenhängend (zusätzlich Diagonalbewegung möglich)



## 3.4 Füllalgorithmen

- Bemerkungen
  - Füllalgorithmen mit **8 Freiheitsgraden** (Bewegungsrichtungen) können **auch 4-fach** zusammenhängende Gebiete füllen.
  - Füllalgorithmen mit **4 Freiheitsgraden können keine 8-fach** zusammenhängenden Gebiete füllen.
- Problem:
  4-fach zusammenhängende Gebiete mit gemeinsamen Ecken
- Techniken zum Rastern eines Polygons / Füllen eines Gebietes
  - Scan-Line-Methode
  - Saatkorn-Methode
  - Hybrid-Methoden

## Scan-Line-Methode

- Andere Bezeichnungen
  - Rasterzeilen-Methode
  - Scan-Conversion
- Arbeitet zeilenweise von oben nach unten
- Ein Pixel der aktuellen Zeile (Scan-Line) wird nur dann gezeichnet, wenn es innerhalb des Polygons liegt
- Definition der Gebiete
  - Geometrisch
  - Pixelweise

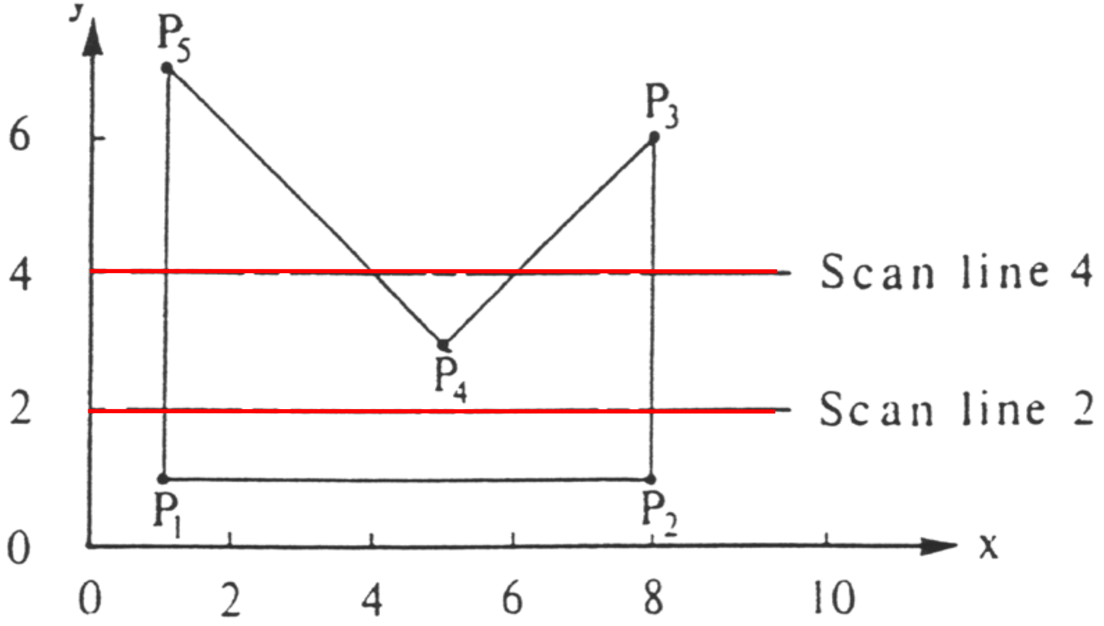
## Scan-Line-Methode

- // einfachster Ansatz:
- for (y = ymin; y <= ymax; y++) {
  - // row, Zeile
  - for (x = xmin; x <= xmax; x++) {
    - // column, Spalte
    - if (Inside(polygon, x, y) {
      - SetPixel(x,y);
    - }
  - }
- }

- Eigenschaften
  - Sehr langsam
  - Verbesserung der Laufzeit durch Ausnutzung von Kohärenz

## Scan-Line-Methode

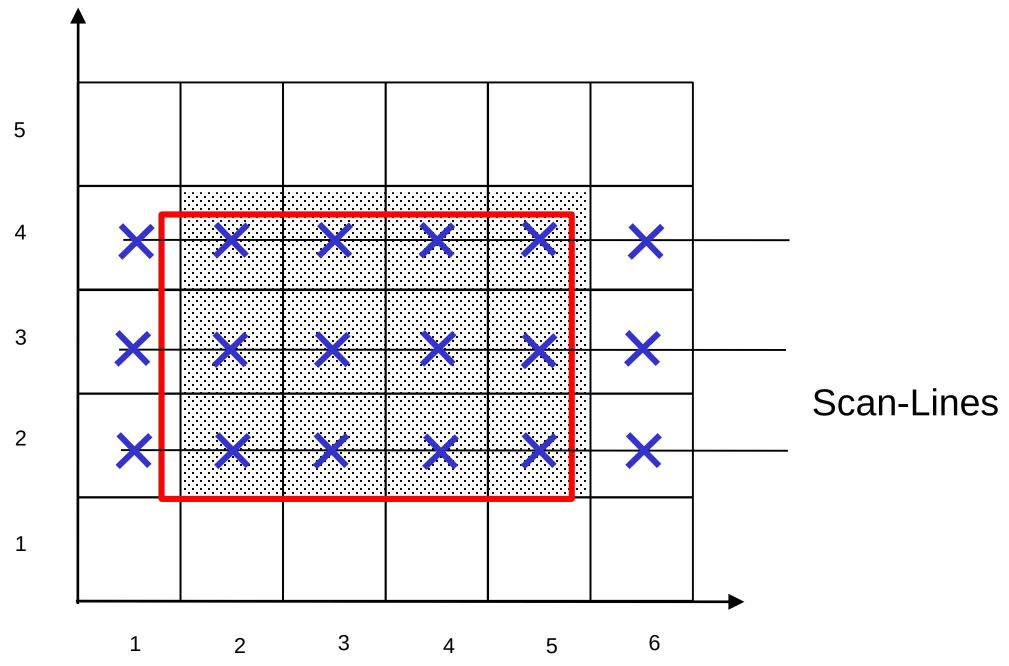
- Ausnutzung von Zeilenkohärenz
  - Benachbarte Pixel auf einer Zeile besitzen höchstwahrscheinlich die gleichen Intensitätswerte.
  - Pixelcharakteristik (Intensität) ändert sich nur dort, wo ein Schnittpunkt einer Polygonkante mit einer Scan Line vorliegt, d.h. der Bereich zwischen zwei Schnittpunkten gehört zum Polygon oder nicht.
- Schnitt mit Polygon
  - $y = 2$: für $x = 1{,}8$
  - $y = 4$: für $x = 1{,}4{,}6{,}8$



## 3.4 Füllalgorithmen

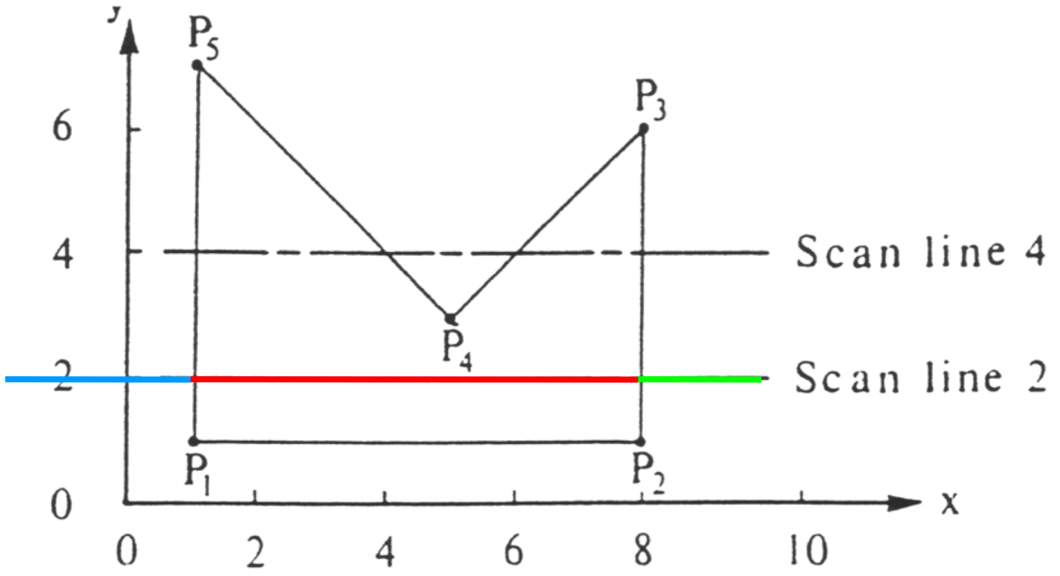
Scan-Line-Methode

- Ausnutzung von Zeilenkohärenz
  - Benachbarte Pixel auf einer Zeile besitzen höchstwahrscheinlich die gleichen Intensitätswerte.
  - Pixelcharakteristik (Intensität) ändert sich nur dort, wo ein Schnittpunkt einer Polygonkante mit einer Scan Line vorliegt, d.h. der Bereich zwischen zwei Schnittpunkten gehört zum Polygon oder nicht.
- Hier sind die Pixelzeilen und -spalten nummeriert, nicht die Koordinaten der Achsen!



## Scan-Line-Methode

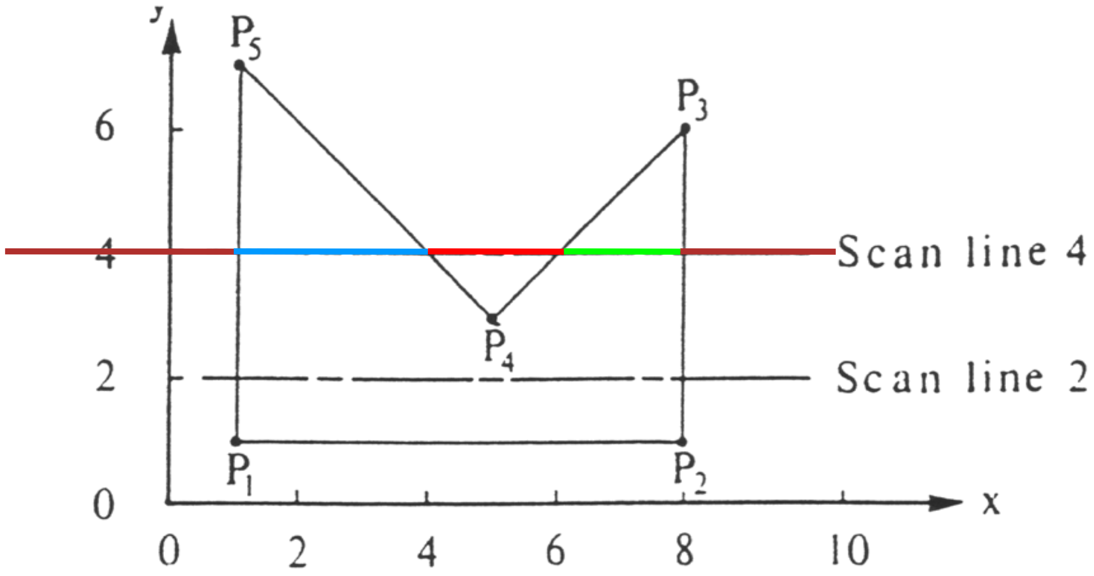
- Scan-Line $y = 2$
- Unterteilung in 3 Bereiche:
  - $x < 1$ außerhalb des Polygons
  - $1 \leq x \leq 8$ innerhalb des Polygons
  - $x > 8$ außerhalb des Polygons



## 3.4 Füllalgorithmen

Scan-Line-Methode

- Scan-Line $y = 4$
- Unterteilung in 5 Bereiche:
  - $x < 1$ außerhalb des Polygons
  - $1 \leq x \leq 4$ innerhalb des Polygons
  - $4 < x < 6$ außerhalb des Polygons
  - $6 \leq x \leq 8$ innerhalb des Polygons
  - $8 < x$ außerhalb des Polygons

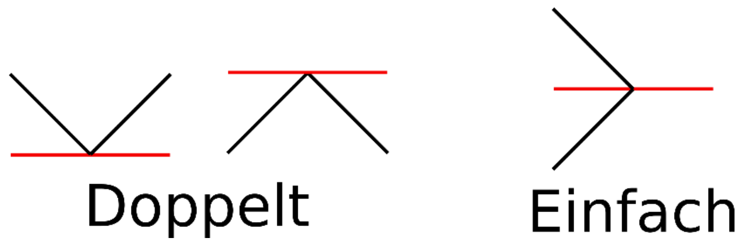


## 3.4 Füllalgorithmen

### Scan-Line-Methode

- Probleme treten auf, wenn die Scan-Line das Polygon in einer Ecke schneidet
  $\rightarrow$ Betrachte **lokale Extrema**
- Lokale Extrema:
  - y-Werte der **Endpunkte** der in dieser Ecke beginnenden Polygonseiten sind **beide größer** oder **beide kleiner** als der y-Wert der **Schnittecke.**
  - Fallunterscheidung:
    - Ist die Ecke ein lokales Extremum, so zählt der **Schnitt zweifach.**
    - Ist die Ecke kein lokales Extremum, so zählt der **Schnitt nur einfach.**

- Einfacher Kanten-Listen-Algorithmus: Ordered Edge List Algorithm
- Funktionsweise:
  - Preprocessing
  - Scan Conversion



# Scan-Line-Methode

- **Preprocessing**
  - Ermittle für jede Polygonkante die **Schnittpunkte mit den Scan-Lines** in der Pixelmitte
    - Bresenham
    - DDA-Algorithmus
  - Ignoriere dabei **horizontale Kanten**
  - **Speichere** jeden Schnittpunkt (x, y) in einer Liste
  - **Sortiere die Liste** dann von oben nach unten und von links nach rechts
- **Scan-Conversion**
  - Betrachte jeweils zwei direkt **aufeinander folgende Schnittpunkte** $(x_1, y_1)$ und $(x_2, y_2)$ der Liste
    - Listenelemente 1 und 2
    - Listenelemente 3 und 4
    - …
  - Aufgrund des Preprocessing gilt für die Scan-Line y

    $y = y_1 = y_2$ und $x_1 \leq x_2$
  - Zeichne alle Pixel auf der Scan-Line y, für die gilt:

    $x_1 \leq x < x_2$ mit ganzzahligem x

## 3.4 Füllalgorithmen

Scan-Line-Methode: Preprocessing

(1,7), (1,7)

(1,6), (2,6), (8,6), (8,6)

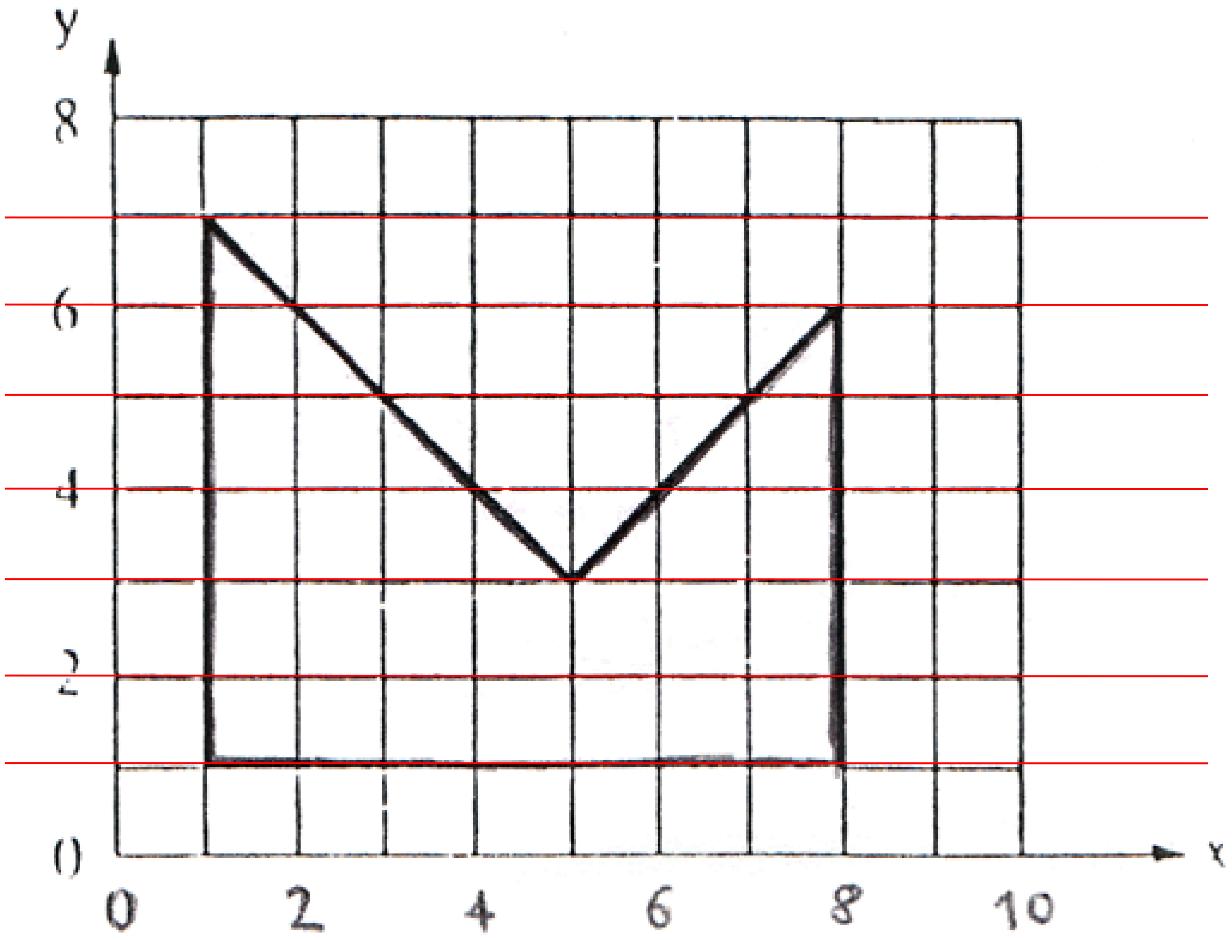
(1,5), (3,5), (7,5), (8,5)

(1,4), (4,4), (6,4), (8,4)

(1,3), (5,3), (5,3), (8,3)

(1,2), (8,2)

(1,1), (8,1)



# 3.4 Füllalgorithmen

## Scan-Line-Methode: Scan-Conversion

(1,7), (1,7)

(1,6), (2,6), (8,6), (8,6)

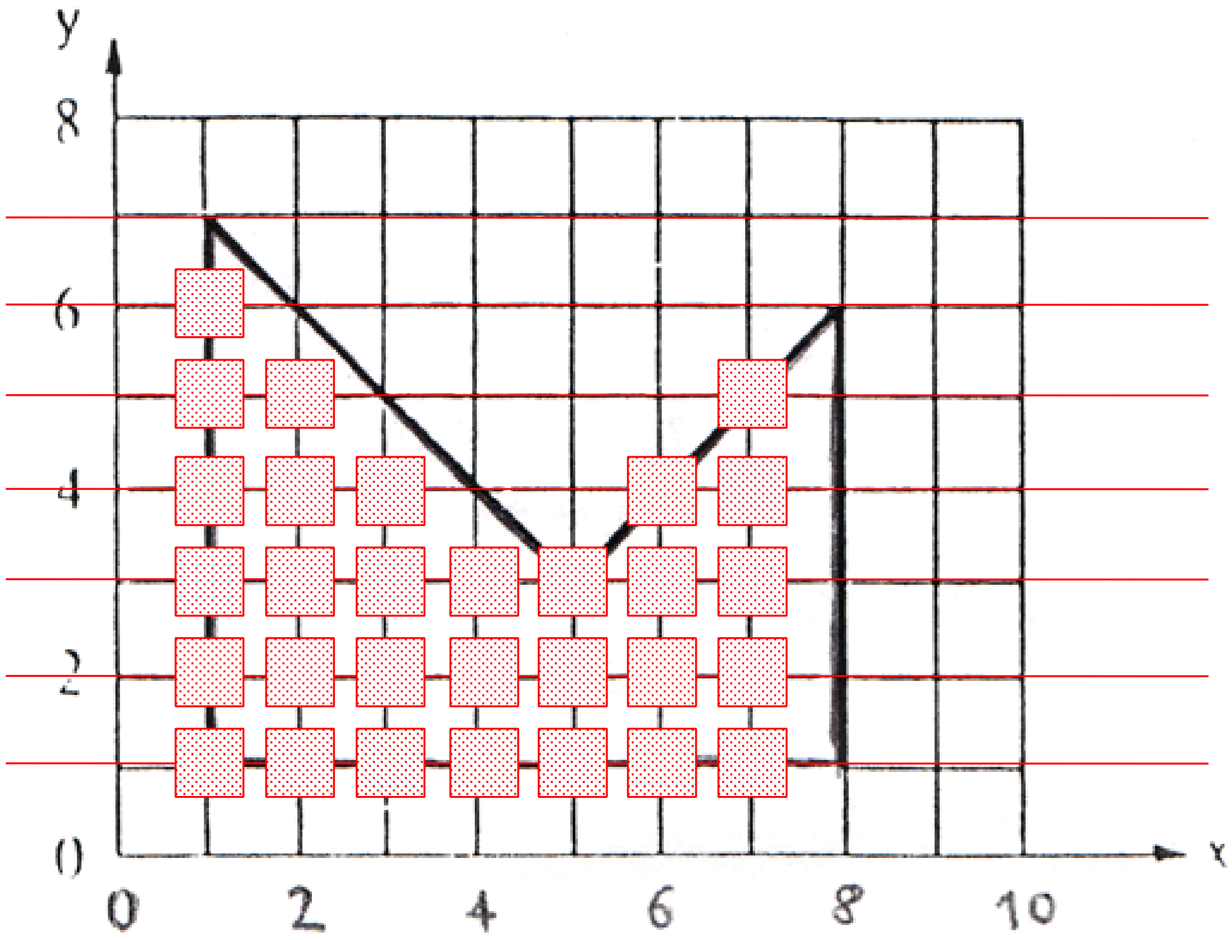
(1,5), (3,5), (7,5), (8,5)

(1,4), (4,4), (6,4), (8,4)

(1,3), (5,3), (5,3), (8,3)

(1,2), (8,2)

(1,1), (8,1)



## Saatpunkt-Methode / Seed-Fill

- Seed-Fill-Methoden füllen das Gebiet ausgehend von einem **Ausgangspixel** (Saatpunkt, Seed)
- Eignet sich für **pixelweise definierte Gebiete**, also für Rastergeräte

- Man unterscheidet nach der Art der Gebietsdefinition:
  1. **Boundary-Fill-Algorithmus** für randdefinierte Gebiete
  2. **Flood/Interior-Fill-Algorithmus** für inhaltsdefinierte Gebiete

# 3.4 Füllalgorithmen

## Saatpunkt-Methode / Seed-Fill

i. Boundary-Fill-Algorithmus
   - Input
     - Startpunkt (Saatpunkt)
     - Farbe der **Begrenzungskurve**
     - Füllfarbe oder Muster
   - Algorithmus
     - Wiederhole
       - Vom Startpixel ausgehend werden rekursiv Nachbarpixel umgefärbt
     - Bis Pixel mit **der Farbe der Begrenzungskurve oder bereits umgefärbte Pixel** erreicht werden

## Saatpunkt-Methode / Seed-Fill

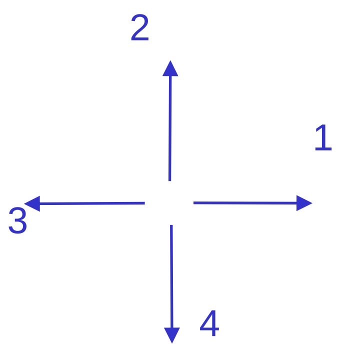
i. Boundary-Fill-Algorithmus
   - Input
     - Startpunkt (Saatpunkt)
     - Farbe der **Begrenzungskurve**
     - Füllfarbe oder Muster
   - Algorithmus
     - Wiederhole
       - Vom Startpixel ausgehend werden rekursiv Nachbarpixel umgefärbt
     - Bis Pixel mit **der Farbe der Begrenzungskurve oder bereits umgefärbte Pixel** erreicht werden

ii. Flood/Interior-Fill-Algorithmus
   - Input
     - Startpunkt (Saatkorn)
     - Farbe der **umzufärbenden Pixel**
     - Füllfarbe oder Muster
   - Algorithmus
     - Wiederhole
       - Vom Startpixel ausgehend werden rekursiv Nachbarpixel umgefärbt
     - Bis Pixel mit **abweichender Farbe** erreicht werden

## 3.4 Füllalgorithmen

### Saatpunkt-Methode / Seed-Fill

- Einfacher Saatkorn-Algorithmus
  - 4 Bewegungsrichtungen
  - randdefiniertes Gebiet
  - FILO/LIFO-Prinzip (Stack)
- Eventuell werden Pixel mehrfach im Stack abgelegt (und gefärbt)

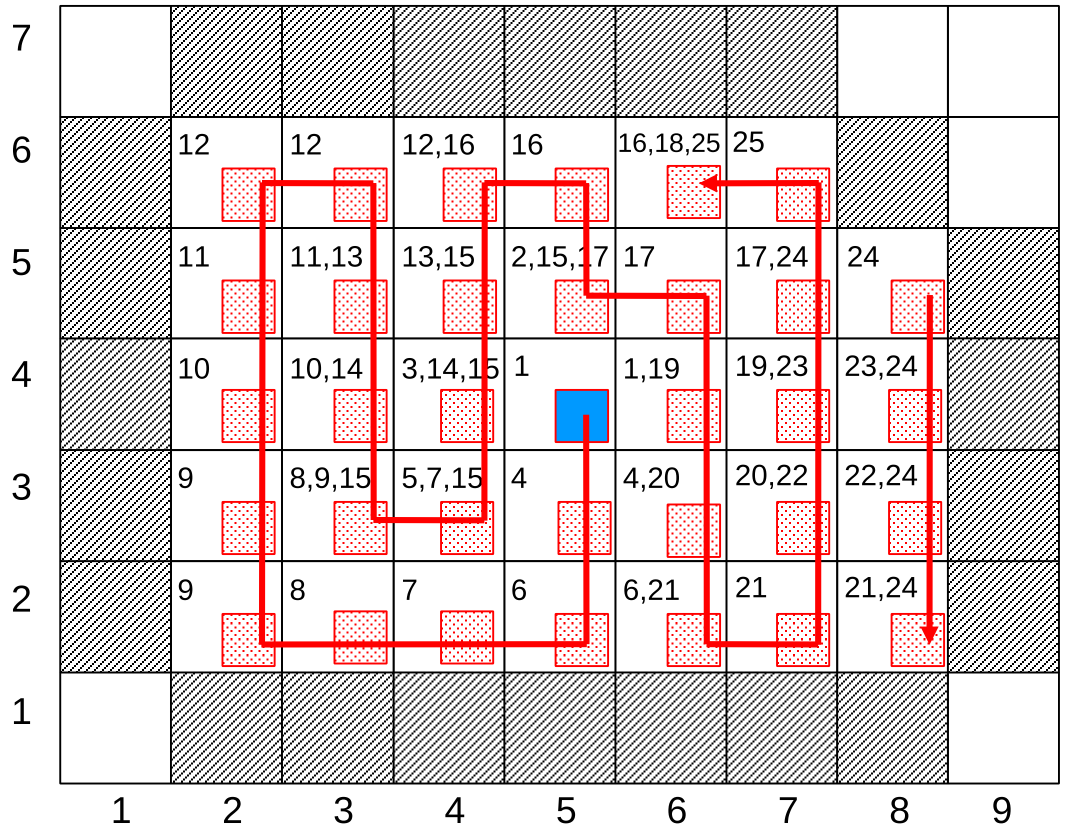


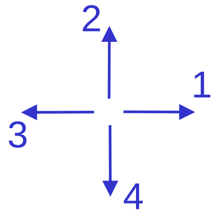

```
Empty(stack);
Push(stack, seed-pixel);

while(stack not empty) {
  pixel = Pop(stack);
  setColor(pixel, FillColor);
  for (each of the 4-connected pixels p_i) {
    if (! ((p_i == boundary_pixel)
        || (colorOf(p_i) == FillColor))) {
      Push(stack, p_i);
    }
  }
}
```

## 3.4 Füllalgorithmen

# Saatpunkt-Methode / Seed-Fill

– Zahlen: Position der Pixel im Stack

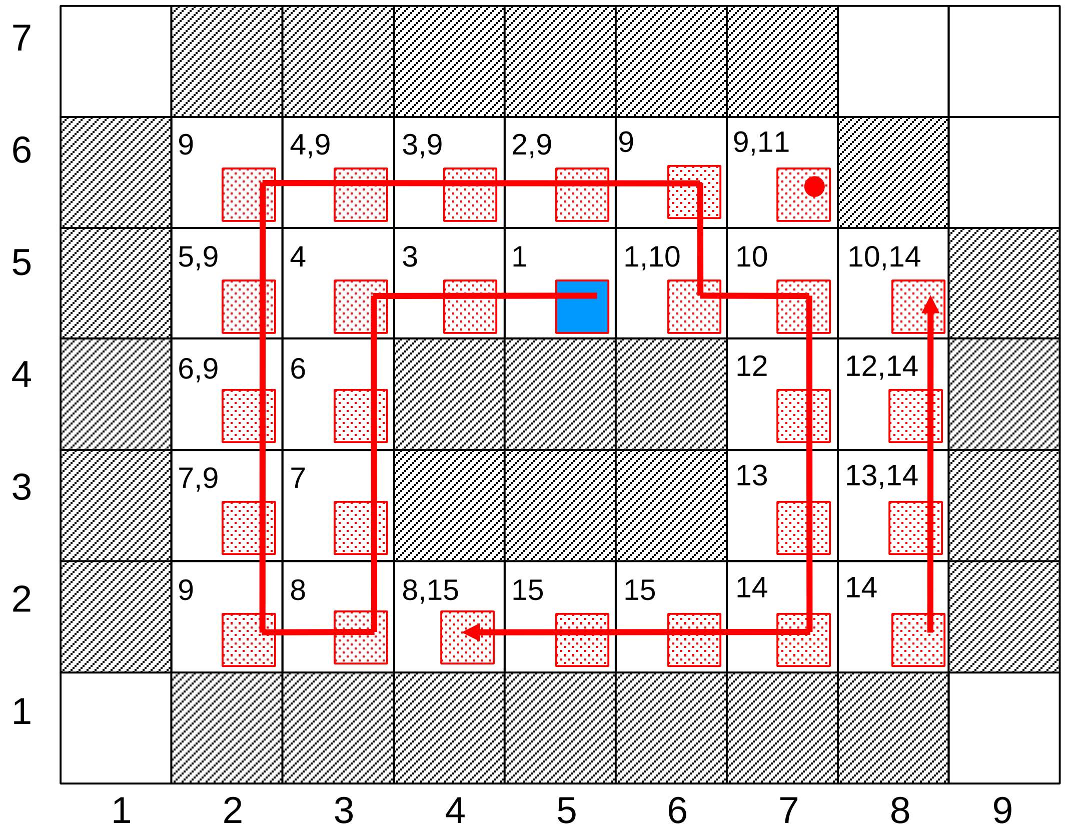


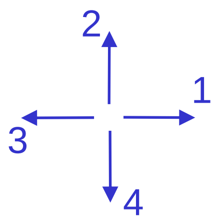



# 3.4 Füllalgorithmen

## Saatpunkt-Methode / Seed-Fill

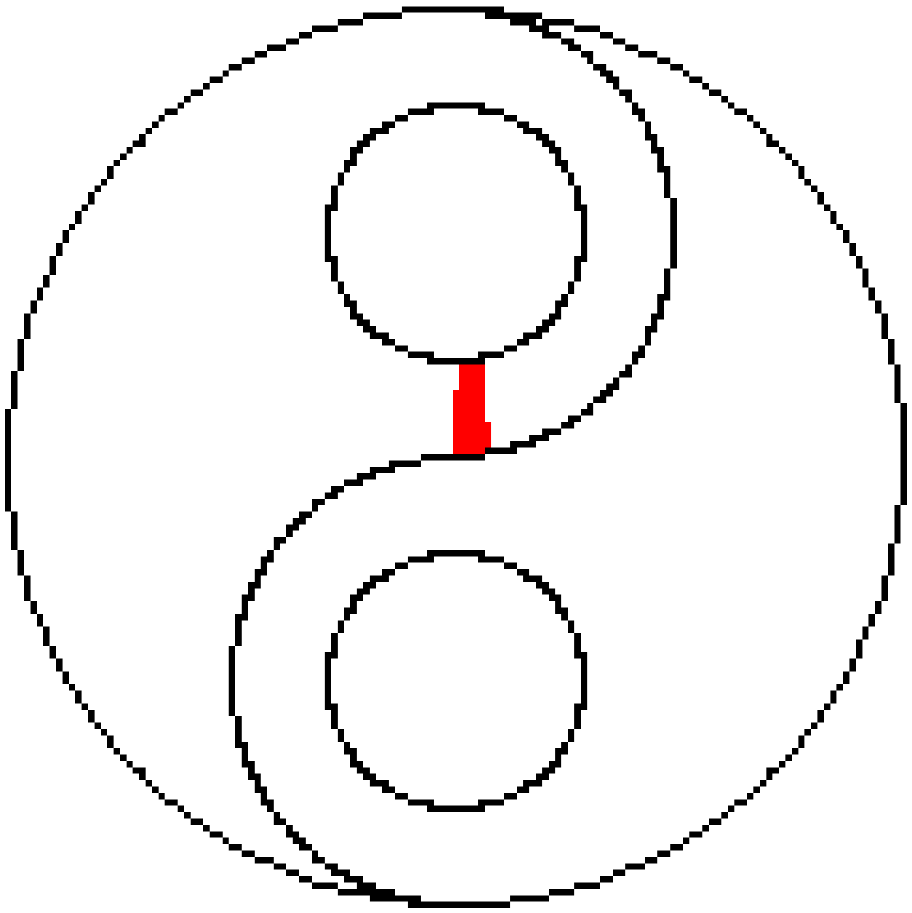
– Gebiet mit Loch (Zahlen: Position der Pixel im Stack)



# 3.4 Füllalgorithmen

Saatpunkt-Methode / Seed-Fill



## 3.4 Füllalgorithmen

- Hybrid-Methoden
  - verwenden die Ideen der Scan-Line- und Saatpunkt-Methoden gemeinsam
  - Scan-Line-Seed-Fill-Algorithmus

# 3.5 Aliasing

## Aliasing (Signaltheorie)

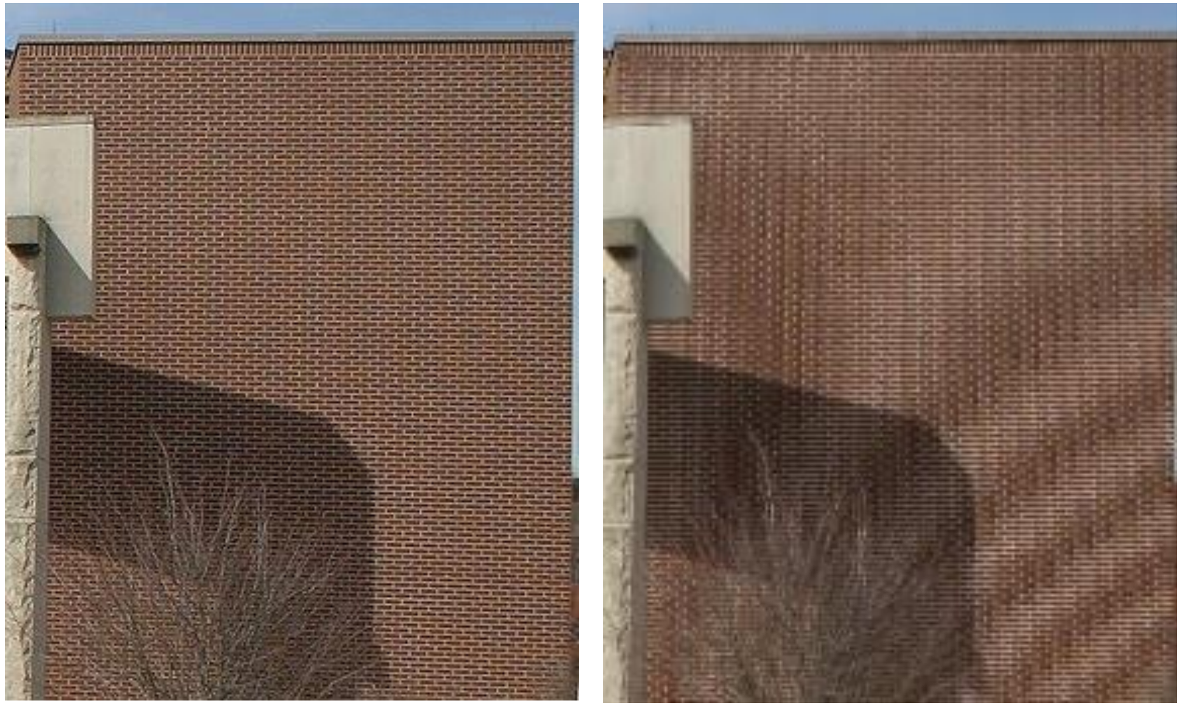
- Allgemein versteht man unter Aliasing-Effekten
  - die **fehlerhafte Rekonstruktion** eines (kontinuierlichen) Ausgangssignals durch eine Abtastung mit **zu geringer Frequenz** (vgl. Nyquist-Theorem)
- Im Frequenzbereich **bandbegrenzte Signale** müssen mit **mehr als der doppelten Grenzfrequenz** abgetastet werden, um eine exakte Rekonstruktion zu ermöglichen

## Aliasing (Computergraphik)

- Visuelle Artefakte durch
  - **Unterabtastung** (z.B. bei einem Schachbrettmuster)
  - **Rasterkonvertierungseffekte** (z.B. Treppeneffekte bei schrägen Linien)
  - **Örtliches und zeitliches Aliasing** (z.B. scheinbar rückwärts drehende Räder)

## Anti-Aliasing-Verfahren

- Methoden, um Aliasing-Effekten entgegenzuwirken:
  - Überabtastung
  - Filterung
- Ein echtes „Beseitigen“ ist oft (schon theoretisch) **nicht möglich,**
  - wenn die Signale **nicht bandbegrenzt** sind.
  - Eine höhere Abtastfrequenz - also **Überabtastung** vermindert die Alias-Effekte, beseitigt sie aber nicht vollständig.
- Bei Artefakten der **Rasterkonvertierung** spricht man von „Verfahren zur (Bild-)**Kantenglättung**“.



## 3.5 Aliasing

Aliasing-Artefakte in der Computergraphik:

- Textur-Artefakte (z.B. Schachbrettmuster)
- Treppeneffekte beim Rastern von Kurven: **Jagged Edges**
- Verschwinden von Objekten, die kleiner als ein Pixel sind
- Verschwinden von langen, dünnen Objekten
- Detailverlust bei komplexen Bildern
- „Aufblinken“ kleiner Objekte bei Bewegungen / Animationen: **Popping**

- Üblicherweise treten visuelle Artefakte auf, wenn die **Periodizität** (zum Beispiel Kachelmuster) in der Textur die **Größenordnung von Pixeln** erreicht.

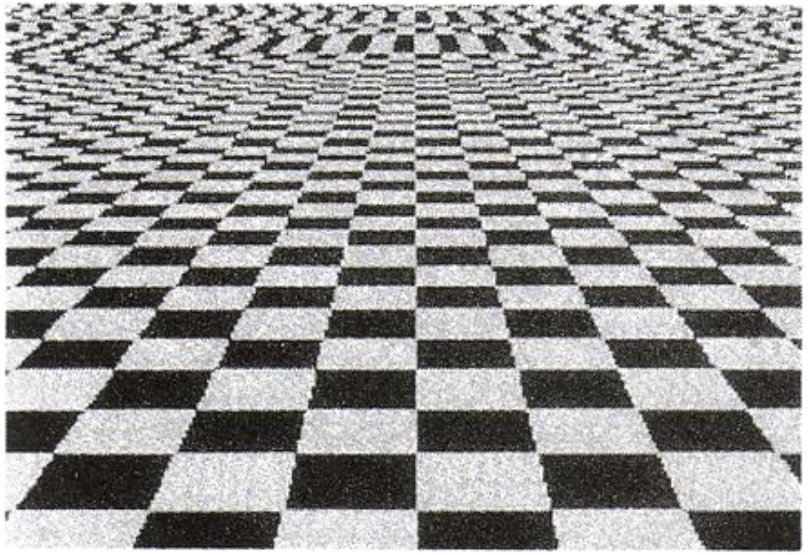
- Es gilt: „Echtes“ Aliasing kann in Computergraphikbildern mittels Überabtastung nicht entfernt, sondern nur verbessert werden (**nicht bandbegrenzt**).

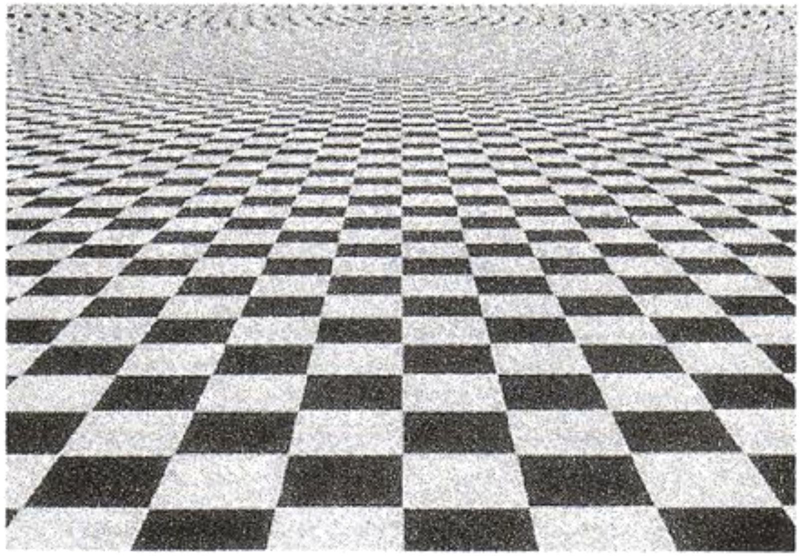
# 3.5 Aliasing

## Textur-Artefakte, unendliches Schachbrettmuster

- Am oberen Ende werden die Quadrate zunächst immer kleiner und dann wieder größer (Aliasing)
- Dies ist ein Ergebnis zu grober Abtastung

- Mittels **zweifacher Überabtastung** (Abtastung mit doppelter Frequenz, d.h. vierfacher Rechenaufwand) können die **Artefakte verringert** werden
- Sie treten aber bei höheren Frequenzen immer noch auf (hier: später, weiter oben)





## 3.5 Aliasing

### Treppeneffekte, Jagged Edges, Jaggies

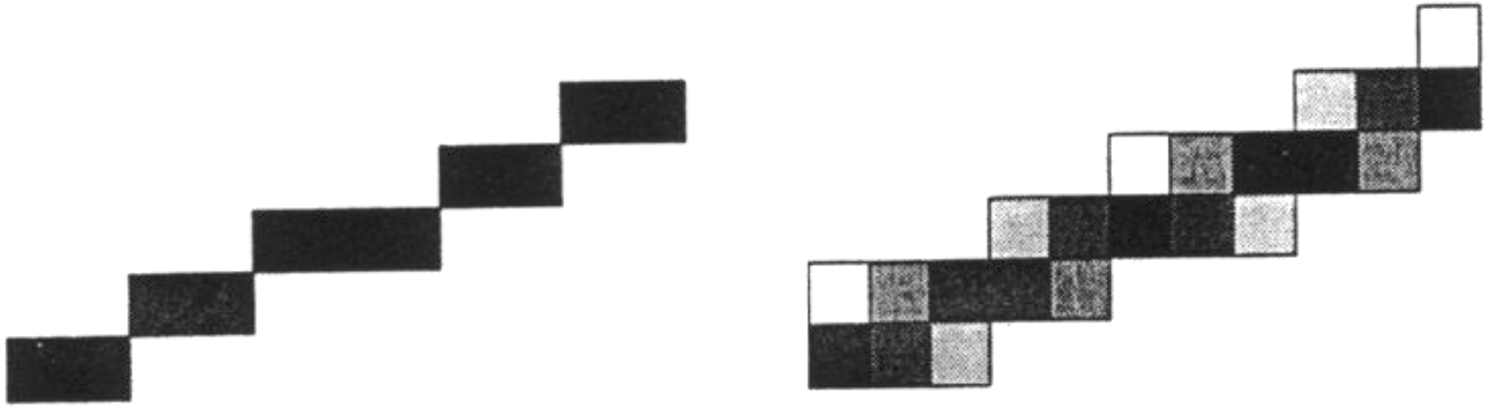
- Die bisher beschriebenen Verfahren zur Rasterung von Geraden und Kurven **erzeugen Treppeneffekte.**
- Zeichnen von Punkten nur **an Rasterpositionen** möglich
- Diese entsprechen im Allgemeinen **nicht den tatsächlichen** Positionen (Sollpositionen).

- Um solchen Aliasing-Effekten entgegenzuwirken, werden **mehrere Intensitäten** zur Erhöhung der visuellen Auflösung benutzt.
- Zum Beispiel verwendet eine Variante des Bresenham-Algorithmus für Geraden (im ersten Oktanten) für jeden x-Wert **zwei Pixel mit Grautönen entsprechend eines Abstandmaßes** zur zu zeichnenden Strecke.

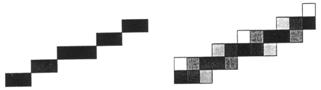
## 3.5 Aliasing

### Treppeneffekte, Jagged Edges, Jaggies

- Treppeneffekte bei einer gerasterten Geraden
- Graustufen, um Treppeneffekte zu verringern



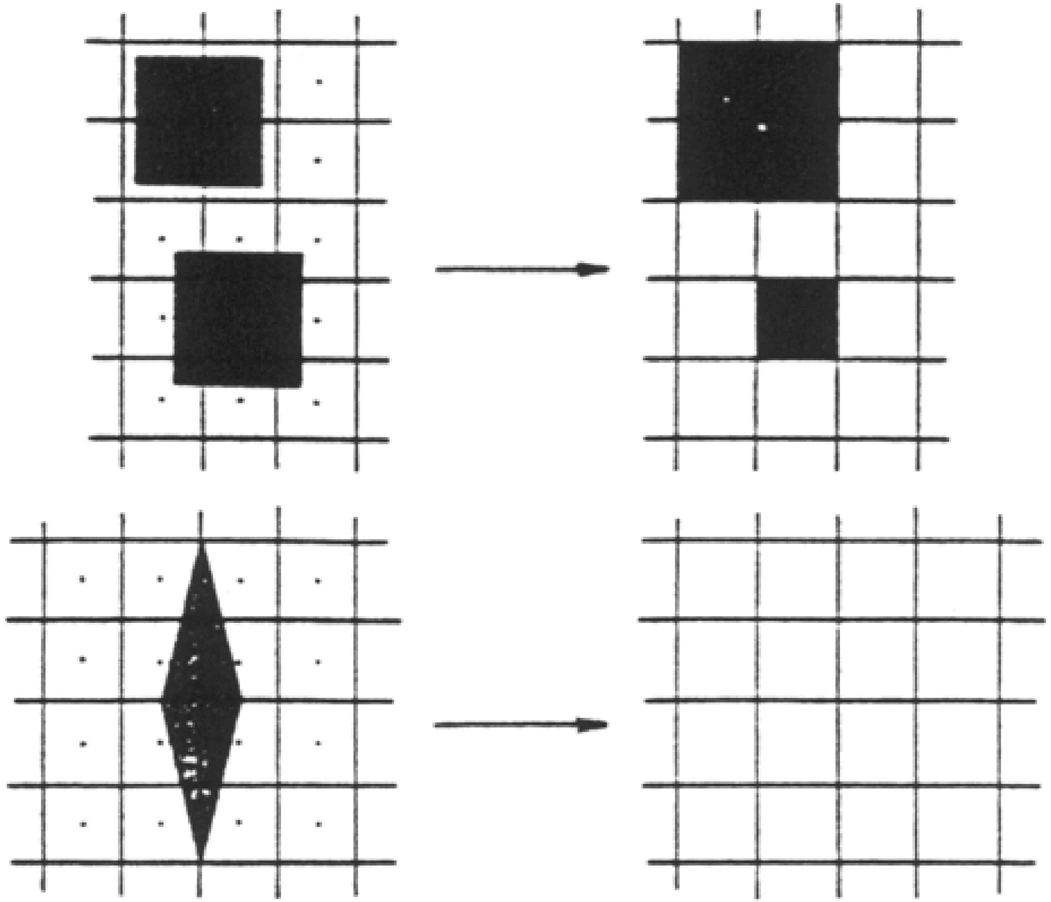
Gegenüberstellung von ungeglättetem und geglättetem Vektor.



Gegenüberstellung von ungeglättetem und geglättetem Vektor.

## 3.5 Aliasing

Aliasing bei Polygonen



# 3.5 Aliasing

## Anti-Aliasing Verfahren: Oversampling

- Ein einfaches globales Anti-Aliasing-Verfahren
- Andere Bezeichnungen
  - Überabtastung
  - Supersampling

- Jedes Pixel wird mit einer höheren Auflösung berechnet, als es schließlich dargestellt wird.

- Das Pixel erhält als eigentliche Grauwertintensität bzw. Farbwert einen gewichteten Durchschnitt der an ihm beteiligten Subpixelwerte.

- Dieses Vorgehen entspricht allgemein einem Filterprozess, dessen theoretische Grundlagen in der Digitalen Signalverarbeitung begründet liegen.

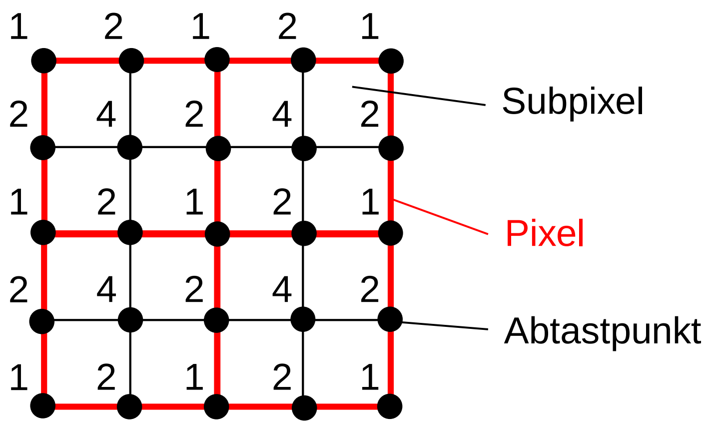
## 3.5 Aliasing

# Anti-Aliasing Verfahren (Filterkerne [Crow, 1981])

**3 x 3**

Faktor: 1/16

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |




**5 x 5**

Faktor: 1/81

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 2 | 1 |
| 2 | 4 | 6 | 4 | 2 |
| 3 | 6 | 9 | 6 | 3 |
| 2 | 4 | 6 | 4 | 2 |
| 1 | 2 | 3 | 2 | 1 |

**7 x 7**

Faktor: 1/256

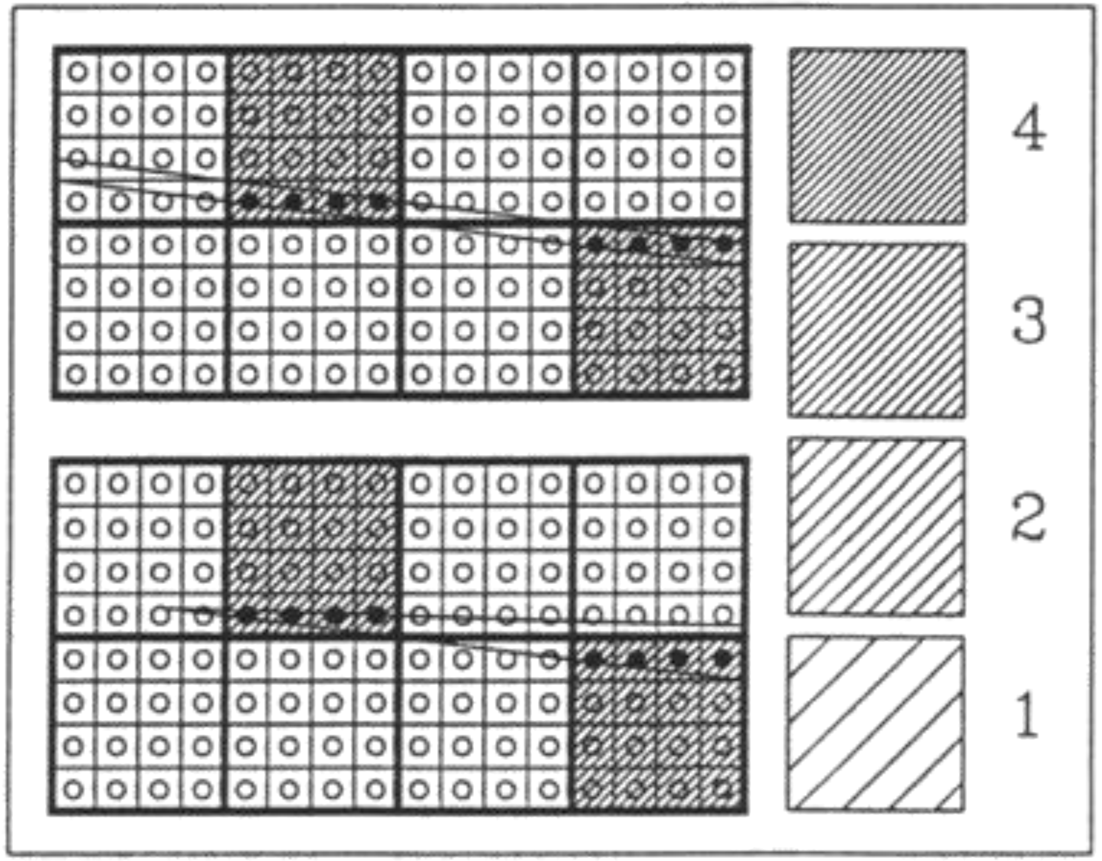
| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 3 | 2 | 1 |
| 2 | 4 | 6 | 8 | 6 | 4 | 2 |
| 3 | 6 | 9 | 12 | 9 | 6 | 3 |
| 4 | 8 | 12 | 16 | 12 | 8 | 4 |
| 3 | 6 | 9 | 12 | 9 | 6 | 3 |
| 2 | 4 | 6 | 8 | 6 | 4 | 2 |
| 1 | 2 | 3 | 4 | 3 | 2 | 1 |

## 3.5 Aliasing

### Anti-Aliasing Verfahren

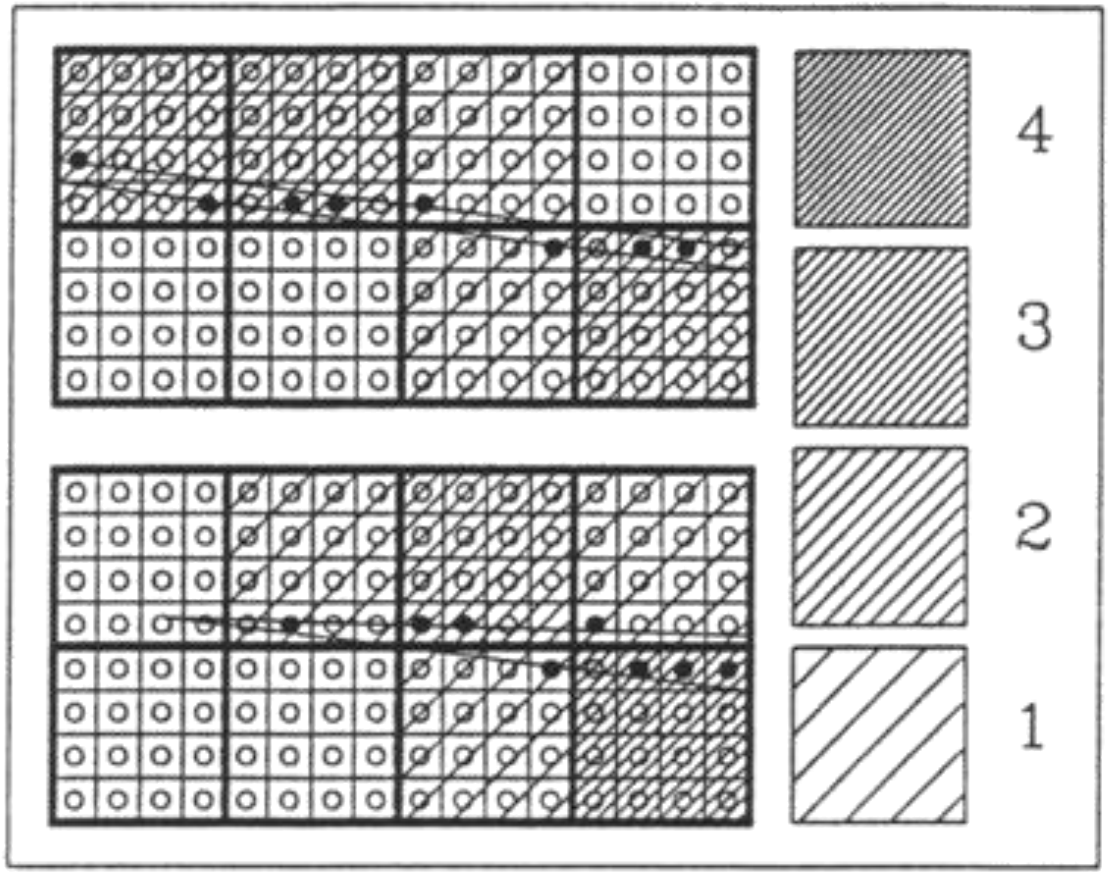
- Bei **Linien und spitzen Dreiecken (dünnen Polygonen)** kann es trotz Supersamplings zu überraschenden Effekten kommen.

- Gesetzte Subpixel sind schwarz.
- Die Schraffur zeigt den Grauwert abhängig von der Anzahl der gesetzten Subpixel.
- Oben:
  Linie verschwindet stellenweise,
  da keine Subpixel getroffen werden!
- Unten:
  Analog für das Dreieck



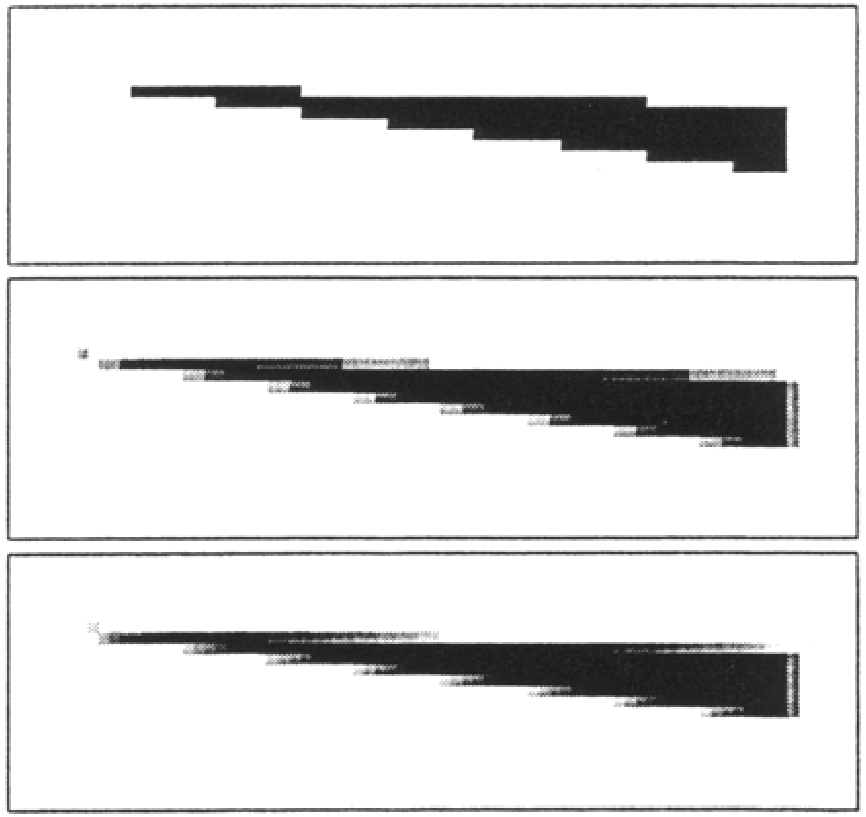
# Anti-Aliasing Verfahren

- Abhilfe für dieses Problem schafft erst eine (korrekte) **Berechnung der überdeckten Fläche** im Pixel.
- Die praktische Anwendung dieser Methode schließt allerdings eine **exakte analytische Berechnung** der wirklich im Pixel überdeckten Fläche **aus.**
- Es existieren hierzu verschiedenste **Näherungsverfahren.**
  - Die überdeckte Fläche lässt sich auch durch eine Anzahl entsprechend gesetzter Subpixel angenähert darstellen.



# 3.5 Aliasing

## Anti-Aliasing Verfahren (Leistungsfähigkeit der vorgestellten Verfahren)



Spitzes Dreieck ohne Glättung

Spitzes Dreieck mit Oversampling

Spitzes Dreieck mit korrekter Berechnung der überdeckten Fläche

# Anti-Aliasing Verfahren: Stochastische Methoden

- Beim stochastischen Abtasten wird das Oversampling mittels Monte-Carlo-Methoden durchgeführt, d.h. die Grauwerte (Intensität) werden an einigen zufälligen Punkten im Pixel ermittelt und das Ergebnis gemittelt.
- Auch bei der Berechnung der vom Polygon im Pixel überdeckten Fläche können Monte-Carlo-Methoden eingesetzt werden.

- Stochastische Methoden
  - erhöhen die Effizienz (schnellere Berechnung)
  - unterdrücken systematische Probleme (z.B. dünne Linie)
  - erlauben adaptives Supersampling
  - neigen aber z.B. zu Problemen bei Animationen, da Objekte flimmern können

## Quellen

- Computergraphik,
  Universität Leipzig
  (Prof. D. Bartz)
- Graphische Datenverarbeitung I,
  Universität Tübingen
  (Prof. W. Straßer)
- Graphische Datenverarbeitung I,
  TU Darmstadt
  (Prof. M. Alexa)
- Computergraphik,
  TU Kaiserslautern
  (Prof. H. Hagen)