UNIVERSITÄT
LEIPZIG

Rasterung

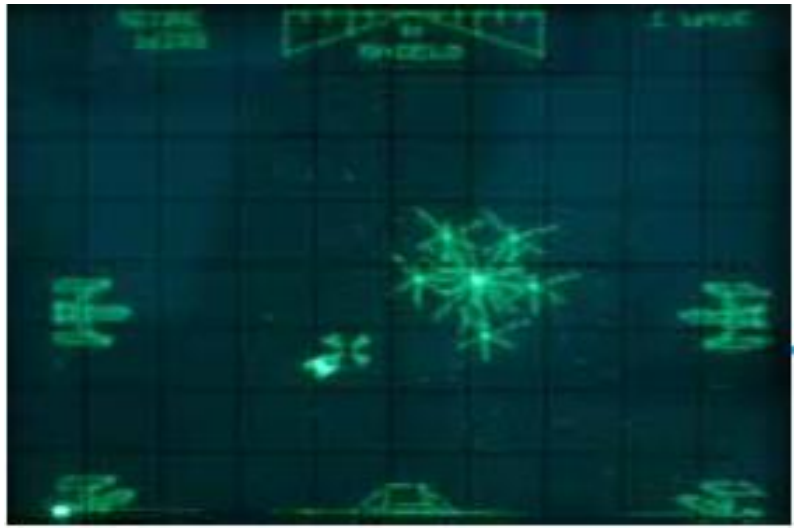
COMPUTERGRAPHIK

Inhaltsverzeichnis

3. Rasterung

Darstellungsmöglichkeiten

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



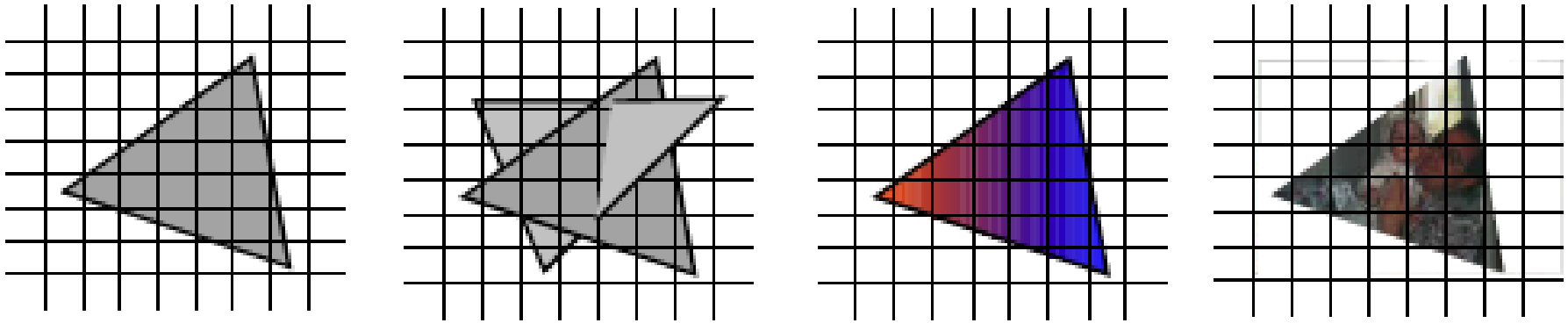
3. Rasterung

Darstellungsmöglichkeiten

- Rasterdisplays
  - Bild wird in Bildpunkte diskretisiert
  - Erfordert Framebuffer (Bildspeicher)

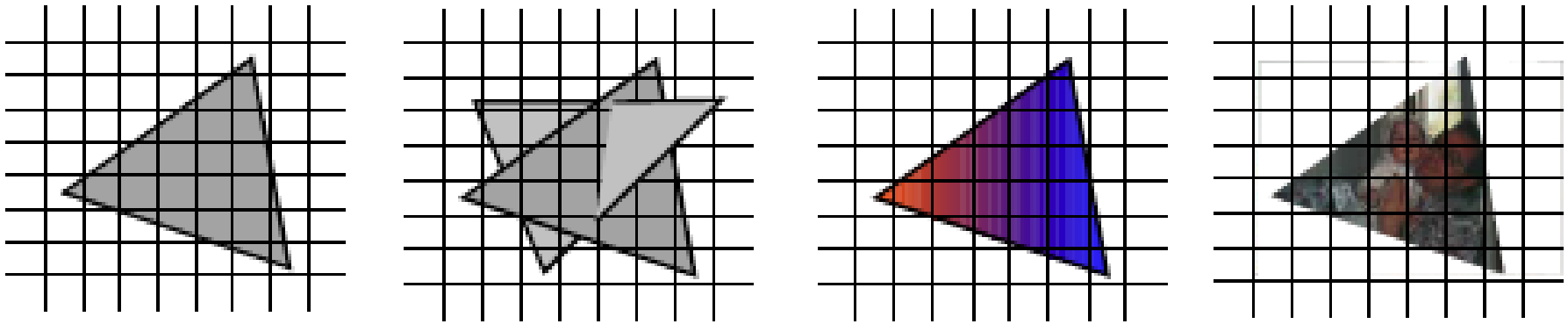
3. Rasterung

- 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 Linien

Problemstellung

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

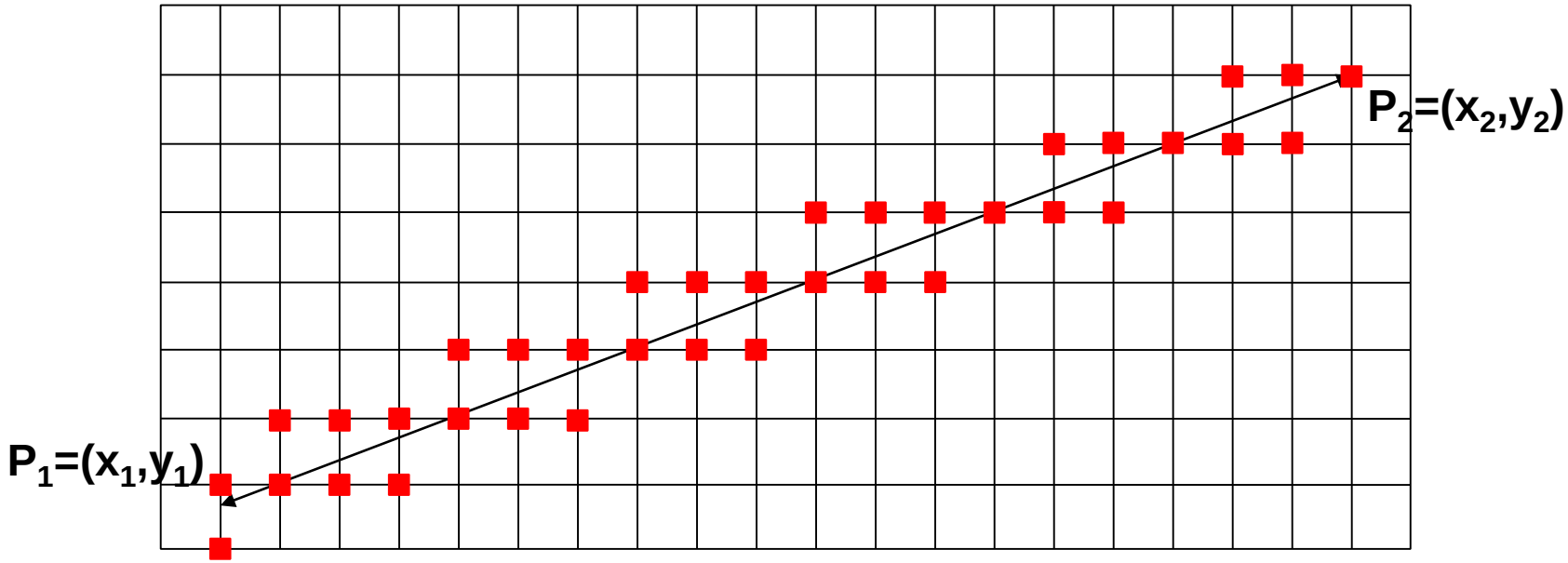


Abb. 2.1: Mögliche Rasterkandidaten

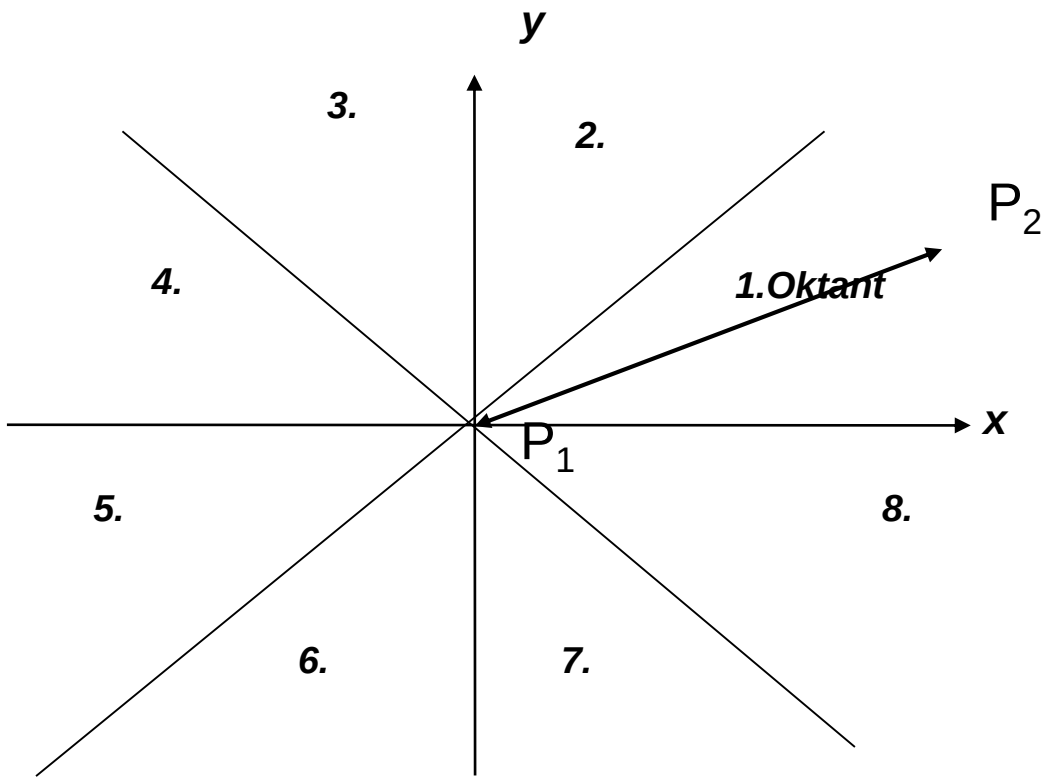
3.1 Rasterung von Linien

Anforderungen

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

3.1 Rasterung von Linien

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

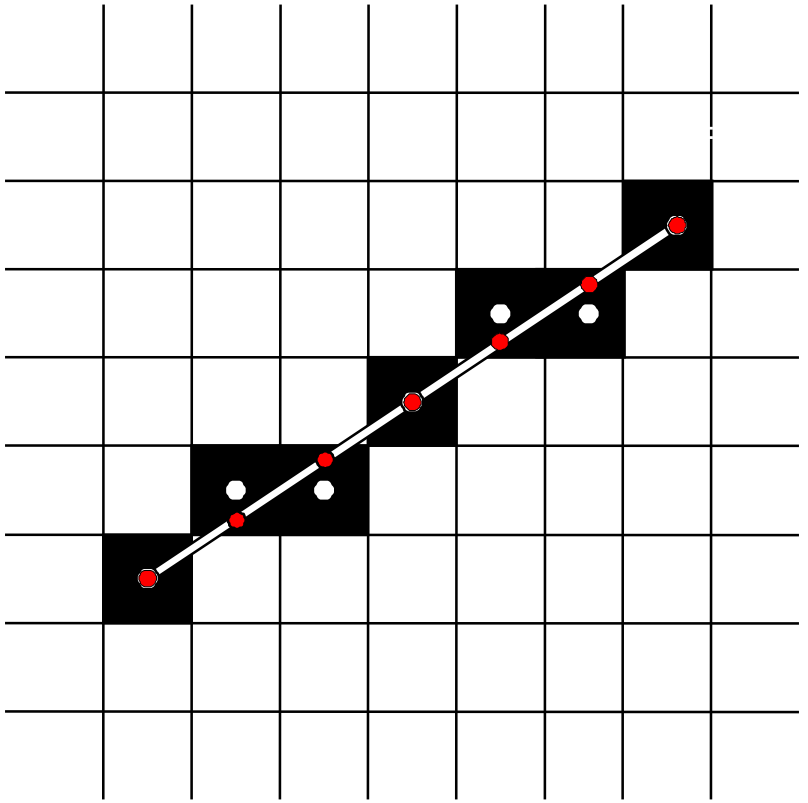


3.1 Rasterung von Linien

DDA-Algorithmus für Geraden [1965]

- DDA
  - Digital Differential Analyzer
  - Digitaler Integrierer

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

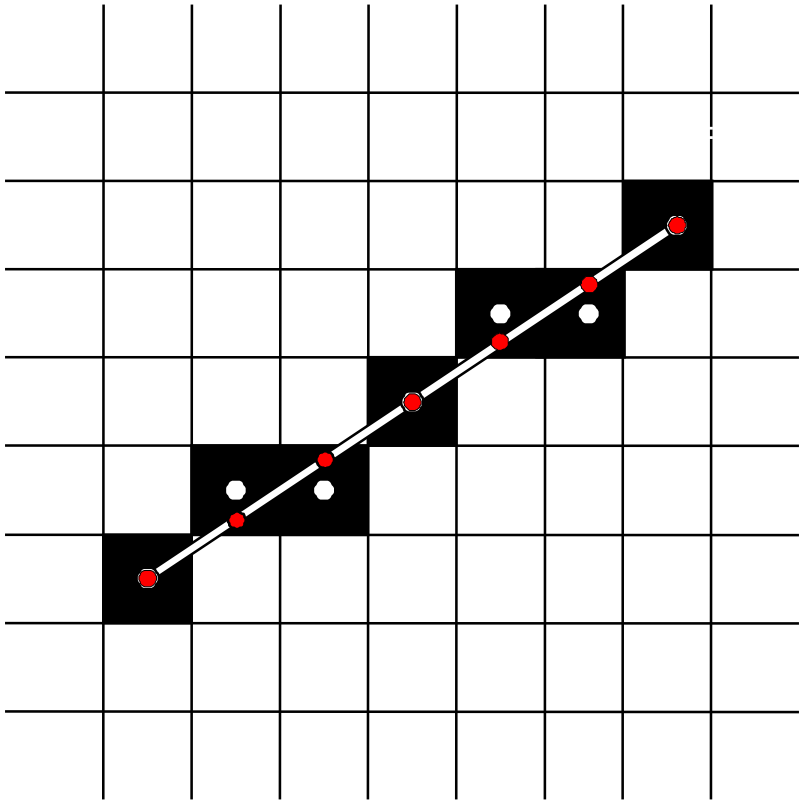


3.1 Rasterung von Linien

DDA-Algorithmus für Geraden [1965]

- $\Delta x = x_2 - x_1 \;\; (= x_{i+1} - x_i)$
- $\Delta y = y_2 - y_1 \;\; (= y_{i+1} - y_i)$
- $-1 \leq \frac{\Delta y}{\Delta x} \leq 1$
- $x_{i+1} = x_1 + i,\ i = 1, \ldots, \Delta x - 1$
- Direkte Berechnung

$$y_{i+1} = \frac{\Delta y}{\Delta x} \cdot x_{i+1} + b$$

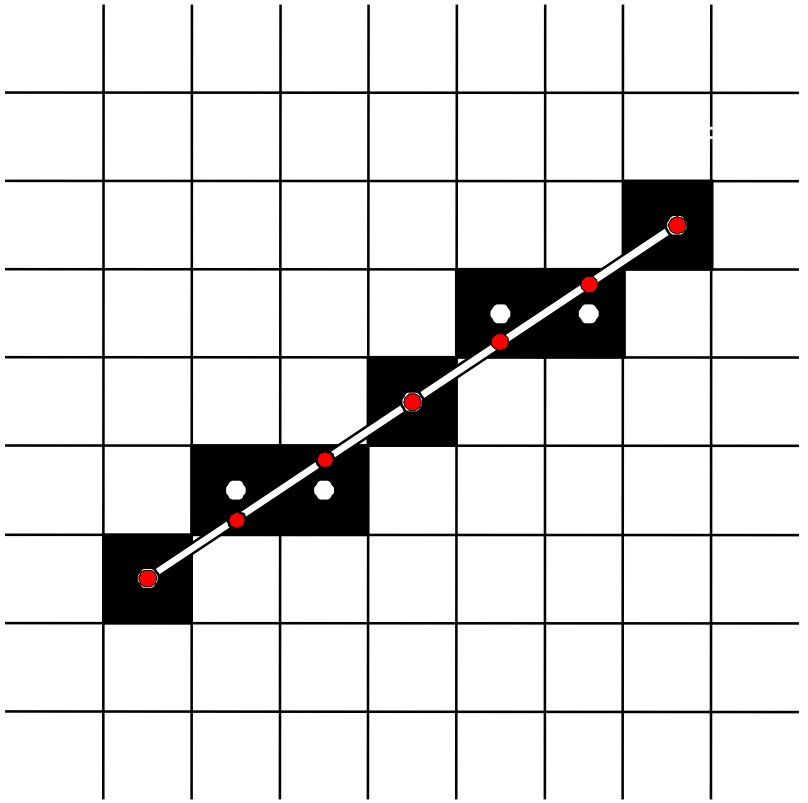


3.1 Rasterung von Linien

DDA-Algorithmus für Geraden [1965]

Inkrementelle Berechnung von y_{i+1}

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



Vorteile inkrementell

- schneller

Vorteil direkt

- genauer

3.2 Rasterung von Geraden

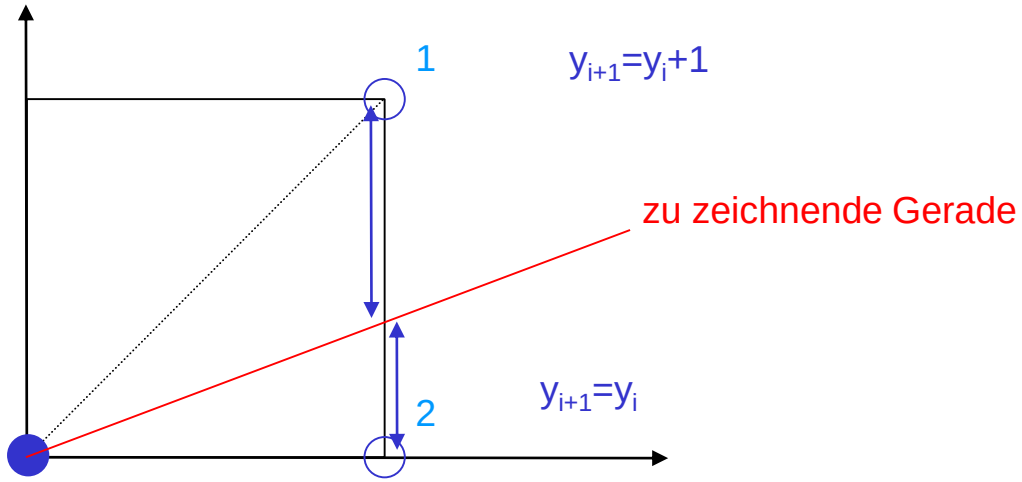
Bresenham-Algorithmus für Geraden [1965]

- 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

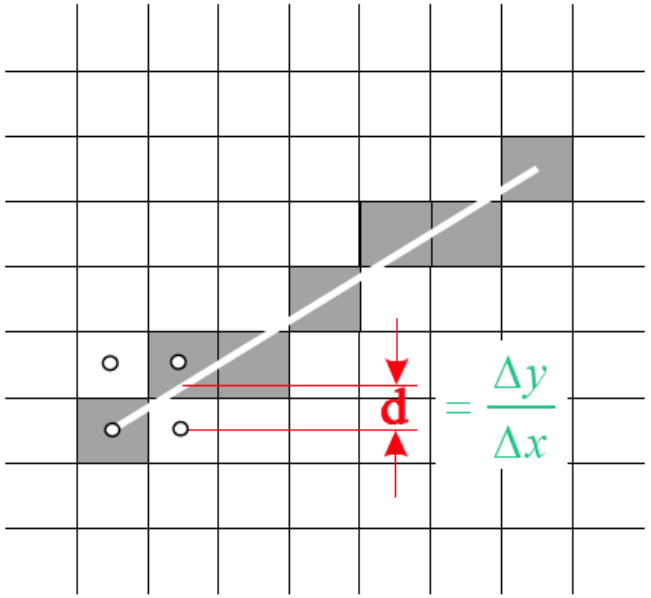
3.2 Rasterung von Geraden

Bresenham-Algorithmus für Geraden [1965]

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



- $y(x) = m \cdot x + c$
- $y'(x) = m = \frac{\Delta y}{\Delta x} = d$



3.2 Rasterung von Geraden

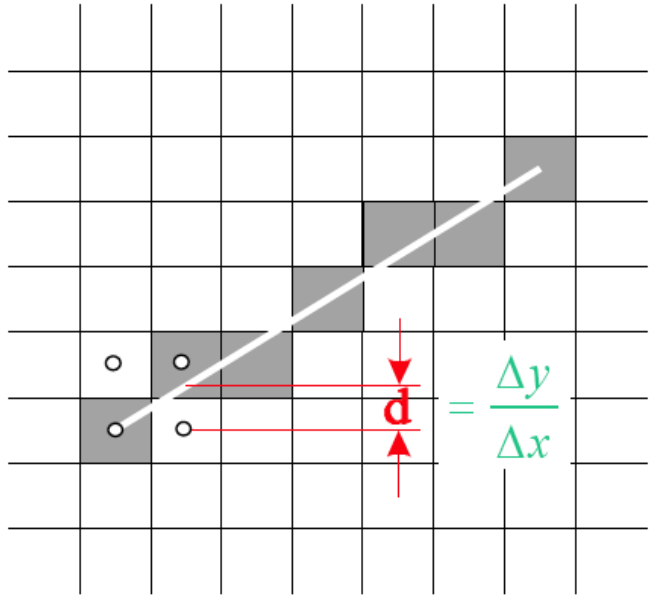
Bresenham-Algorithmus für Geraden [1965]

- Zur Realisierung wird eine **Entscheidungsgröße E** eingeführt, welche die Abweichung zwischen dem exakten Punkt und der Mitte zwischen den beiden möglichen Rasterpunkten angibt:

$$E = \frac{\Delta y}{\Delta x} - \frac{1}{2} = d - \frac{1}{2}$$

- Das **Vorzeichen von E** dient dann als Kriterium für die Rundung auf den nächsten Rasterpunkt:
  - $E > 0$: x++; y++; $E \mathrel{+}= d - 1$
  - $E \leq 0$: x++; $E \mathrel{+}= d$

- $y(x) = m \cdot x + c$
- $y'(x) = m = \frac{\Delta y}{\Delta x} = d$



3.2 Rasterung von Geraden

Bresenham-Algorithmus für Geraden [1965]

- Vermeidung der Division

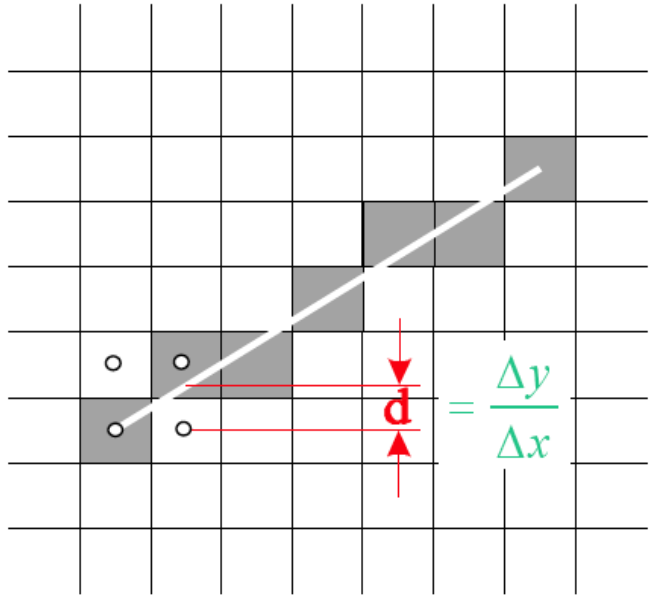
$$E = \frac{\Delta y}{\Delta x} - \frac{1}{2} = \frac{2 \cdot \Delta y - \Delta x}{2 \cdot \Delta x}$$

$$\Downarrow \cdot 2 \cdot \Delta x$$

$$E' = 2 \cdot \Delta y - \Delta x$$

  - $E' > 0$: x++; y++; $E' \mathrel{+}= 2 \cdot \Delta y - 2 \cdot \Delta x$
  - $E' \leq 0$: x++; $E' \mathrel{+}= 2 \cdot \Delta y$

- $y(x) = m \cdot x + c$
- $y'(x) = m = \frac{\Delta y}{\Delta x} = d$



3.2 Rasterung von Geraden

Bresenham-Algorithmus für Geraden [1965]

- Erster Oktant
- Ausschließlich ganzzahlige Operanden
- $e \coloneqq E'$
- $dx \coloneqq \Delta x$
- $dy \coloneqq \Delta y$

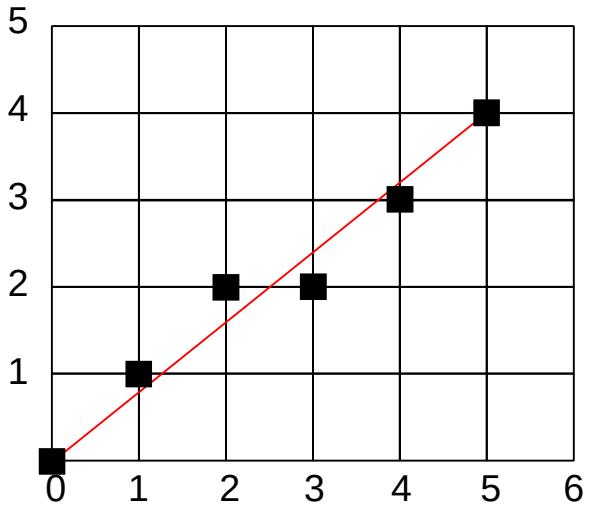
```
// (x1, y1), (x2, y2) Ganzzahlig
// x1 < x2, y1 < y2
x = x1; y = y1;
dx = x2 - x1; dy = y2 - y1;
e = 2 * dy - dx; // Initialisierung
for(i = 1; i <= dx; i++){
  // Schleife fuer x
  plot(x, y);
  if (e >= 0) {
    // oberen Punkt Zeichnen (y erhoehen)
    ++y;
    e -= 2 * dx;
  }
  ++x;
  e += 2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

3.2 Rasterung von Geraden

Bresenham-Algorithmus für Geraden [1965]

Beispiel:

– P1=(0, 0), P2=(5, 4)

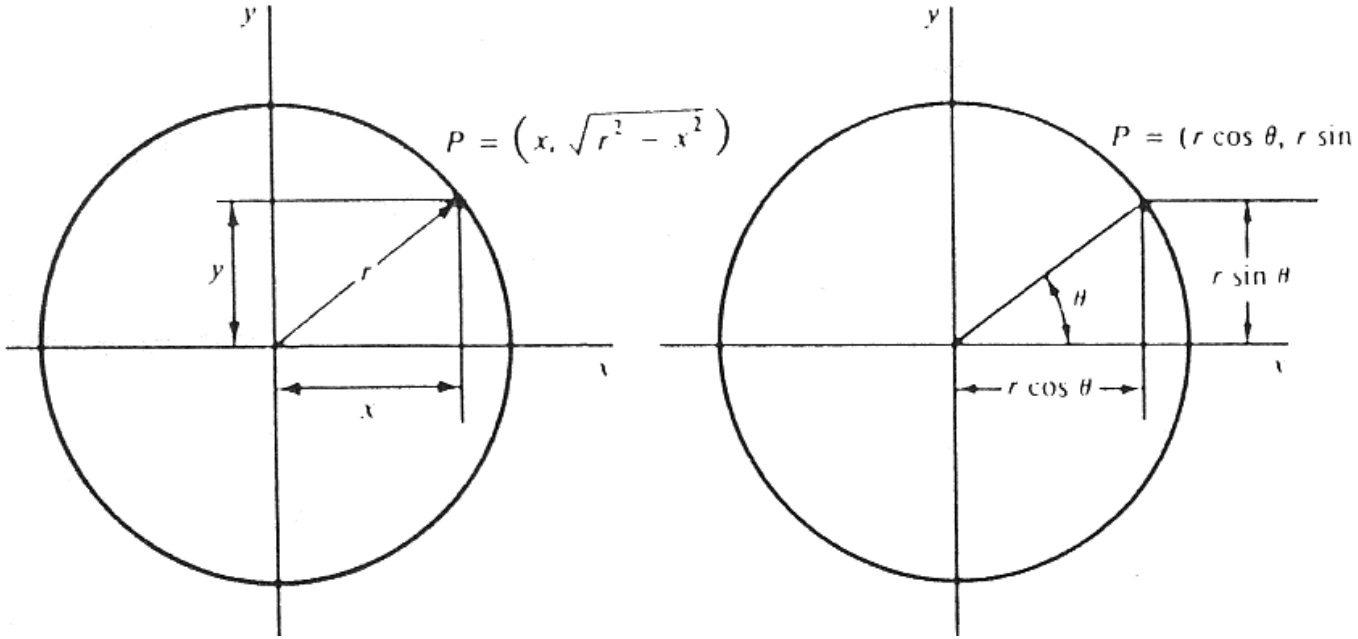


dx	dy	x	y	e	i	plot
5	4	0	0	3	1	(0,0)
			1	-7		
		1		1	2	(1,1)
			2	-9		
		2		-1	3	(2,2)
		3		7	4	(3,2)
			3	-3		
		4		5	5	(4,3)
			4	-5		
		5		3	6	
						(5,4)

3.2 Rasterung von Geraden

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



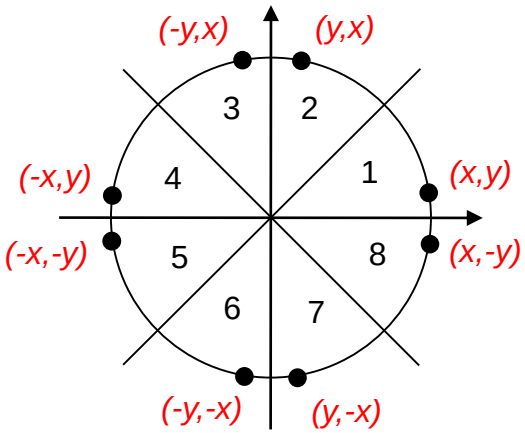
3.2 Rasterung von Geraden

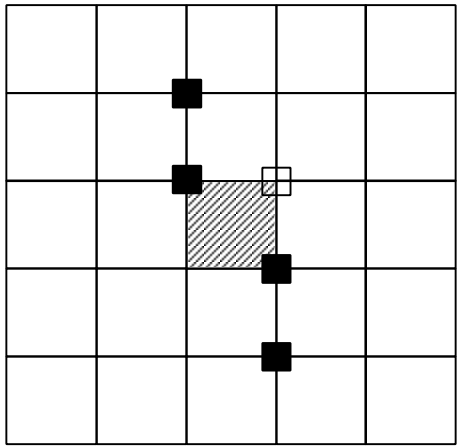
Darstellung eines Kreises: Bemerkungen

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

$$\left|x_i^2 + y_i^2 - r^2\right|$$

- von jedem Rasterquadrant dürfen nur 2 Eckpunkte gesetzt werden

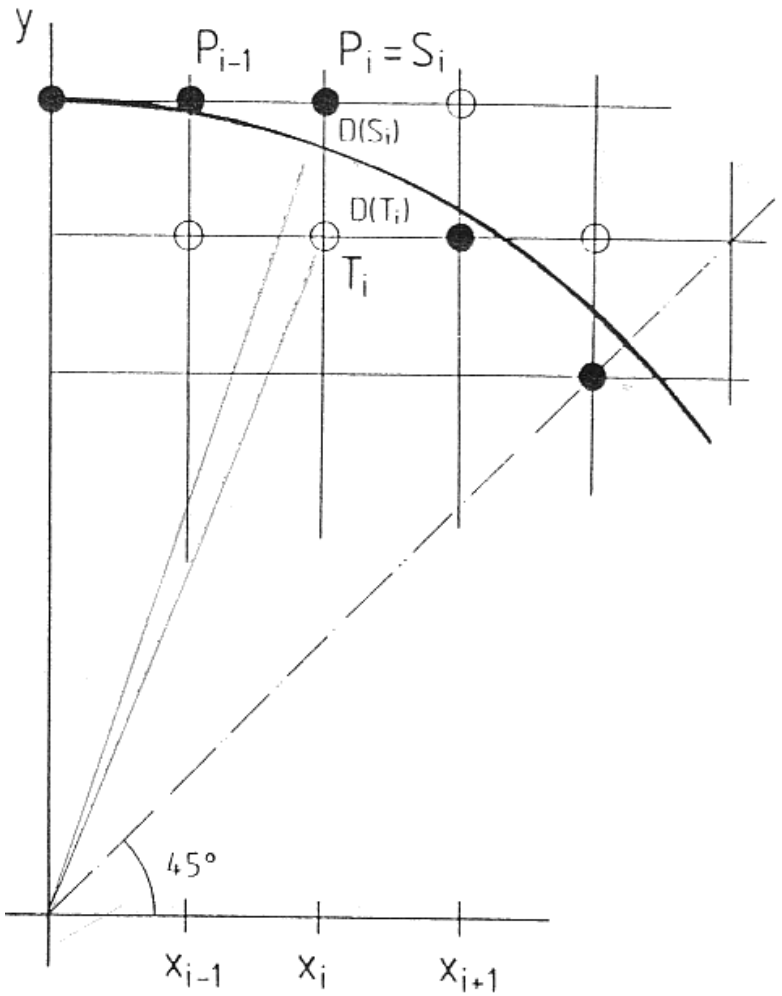




3.2 Rasterung von Geraden

Bresenham-Algorithmus für Kreise

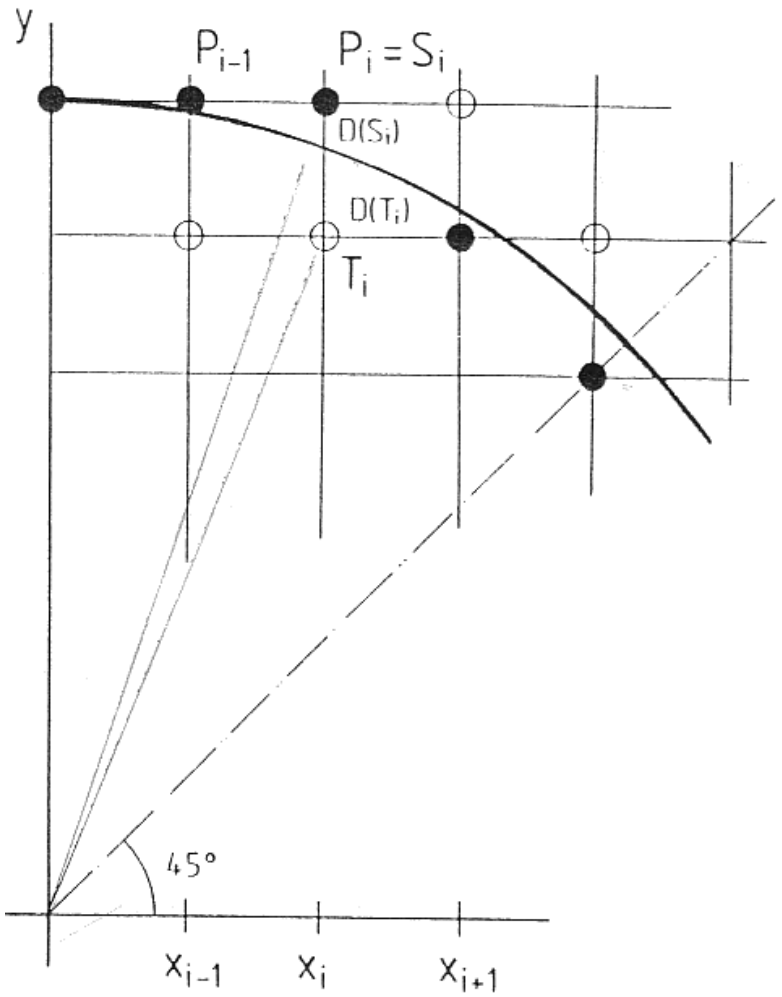
- Beste Approximation über Entscheidungsgröße d_i mittels minimalem Residuum
- Hier:
  - $(x_M, y_M) = (0,0)$
  - Startpunkt ist Rasterpunkt
  - $r \in \mathbb{N}$
  - 2. Oktant
  - $D(S_i) = (x_{i-1} + 1)^2 + y_{i-1}^2 - r^2$
  - $D(T_i) = (x_{i-1} + 1)^2 + (y_{i-1} - 1)^2 - r^2$
  - $d_i = |D(S_i)| - |D(T_i)|$



3.3 Rasterung von Kreisen

Bresenham-Algorithmus für Kreise

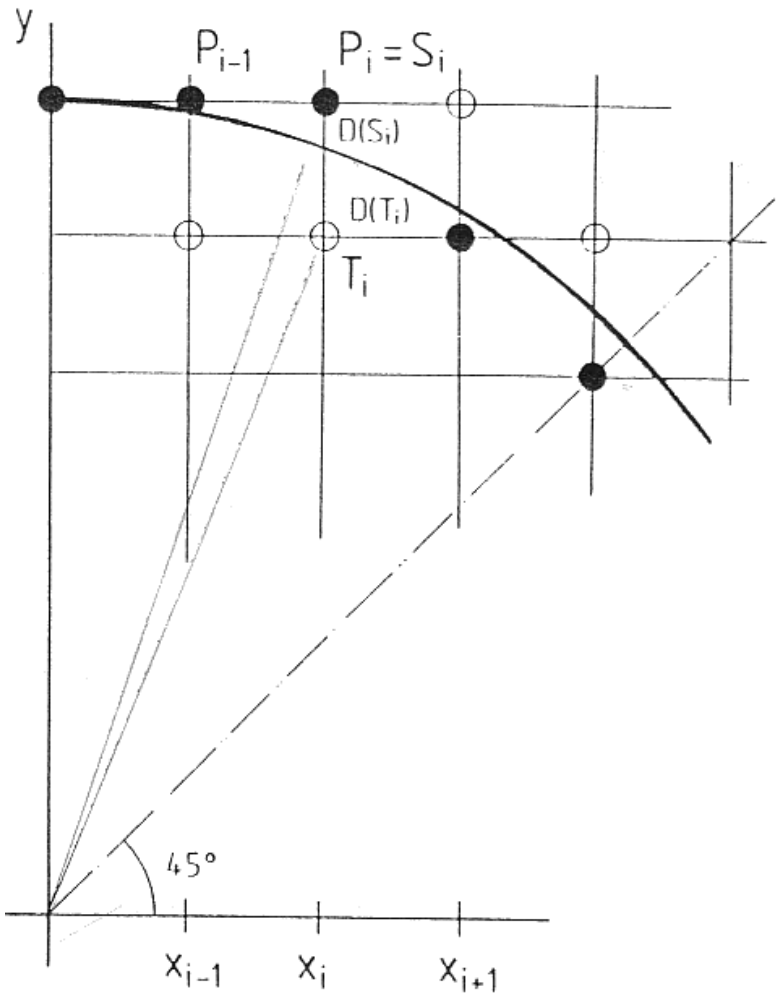
- Die Entscheidungsgröße d_i **misst die Abstände** zum oberen und unteren Rasterpunkt
- $d_i > 0$
  - wähle $P_i = T_i$ **als nächsten Punkt**
  - $x_i = x_{i-1} + 1$
  - $y_i = y_{i-1} - 1$
- $d_i \leq 0$
  - wähle $P_i = S_i$ **als nächsten Punkt**
  - $x_i = x_{i-1} + 1$
  - $y_i = y_{i-1}$



3.3 Rasterung von Kreisen

Bresenham-Algorithmus für Kreise

- Innerhalb des 2. Oktanten
  - Kreisfunktion monoton fallend
  - Steigung zwischen 0 und -1
  - $D(S_i) > 0$: S_i liegt immer außerhalb der Kreislinie
  - $D(T_i) < 0$: T_i liegt immer innerhalb der Kreislinie



3.3 Rasterung von Kreisen

Bresenham-Algorithmus für Kreise

$$
\begin{aligned}
d_i &= (x_{i-1}+1)^2 + y_{i-1}^2 - r^2 + (x_{i-1}+1)^2 + (y_{i-1}-1)^2 - r^2 \\
&= 2 \cdot x_{i-1}^2 + 4 \cdot x_{i-1} + 2 + 2 \cdot y_{i-1}^2 - 2 \cdot y_{i-1} + 1 - 2 \cdot r^2 \\
&= 2 \cdot x_{i-1}^2 + 4 \cdot x_{i-1} + 2 \cdot y_{i-1}^2 - 2 \cdot y_{i-1} + 3 - 2 \cdot r^2
\end{aligned}
$$

- $d_i > 0$:

$$
\begin{aligned}
d_{i+1} &= 2 \cdot x_i^2 + 4 \cdot x_i + 2 \cdot y_i^2 - 2 \cdot y_i + 3 - 2 \cdot r^2 \\
&= 2 \cdot x_{i-1}^2 + 8 \cdot x_{i-1} + 6 + 2 \cdot y_{i-1}^2 - 6 \cdot y_{i-1} + 4 + 3 - 2 \cdot r^2 \\
&= d_i + 4 \cdot x_{i-1} - 4 \cdot y_{i-1} + 10 \\
&= d_i + 4 \cdot x_i - 4 \cdot y_i + 2 \\
&= d_i + 4 \cdot (x_i - y_i) + 2
\end{aligned}
$$

3.3 Rasterung von Kreisen

Bresenham-Algorithmus für Kreise

$$
\begin{aligned}
d_i &= (x_{i-1}+1)^2 + y_{i-1}^2 - r^2 + (x_{i-1}+1)^2 + (y_{i-1}-1)^2 - r^2 \\
&= 2 \cdot x_{i-1}^2 + 4 \cdot x_{i-1} + 2 + 2 \cdot y_{i-1}^2 - 2 \cdot y_{i-1} + 1 - 2 \cdot r^2 \\
&= 2 \cdot x_{i-1}^2 + 4 \cdot x_{i-1} + 2 \cdot y_{i-1}^2 - 2 \cdot y_{i-1} + 3 - 2 \cdot r^2
\end{aligned}
$$

- $d_i \leq 0$:

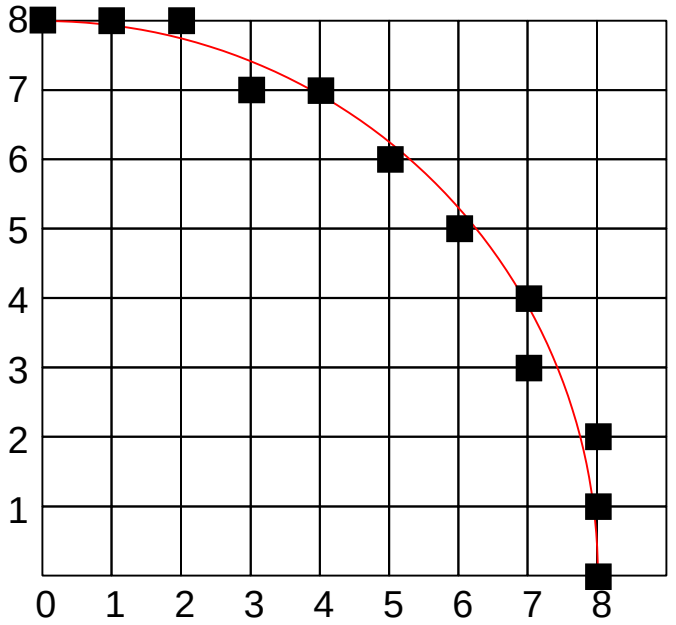
$$
\begin{aligned}
d_{i+1} &= 2 \cdot x_i^2 + 4 \cdot x_i + 2 \cdot y_i^2 - 2 \cdot y_i + 3 - 2 \cdot r^2 \\
&= 2 \cdot x_{i-1}^2 + 8 \cdot x_{i-1} + 6 + 2 \cdot y_{i-1}^2 - 2 \cdot y_{i-1} + 3 - 2 \cdot r^2 \\
&= d_i + 4 \cdot x_{i-1} + 6 \\
&= d_i + 4 \cdot x_i + 2
\end{aligned}
$$

3.3 Rasterung von Kreisen

Bresenham-Algorithmus für Kreise

- Startwerte:
  - $d_1 = 3 - 2 \cdot r$
  - $x_1 = 0$
  - $y_1 = r$
- Beispiel: $r = 8$
- $d_i > 0: d_{i+1} = d_i + 4 \cdot (x_i - y_i) + 2$
- $d_i \leq 0: d_{i+1} = d_i + 4 \cdot x_i + 2$

d	x	y
	0	8
-13		
	1	8
-7		
	2	8
3		
	3	7
-11		
	4	7
7		
	5	6



3.3 Rasterung von Kreisen

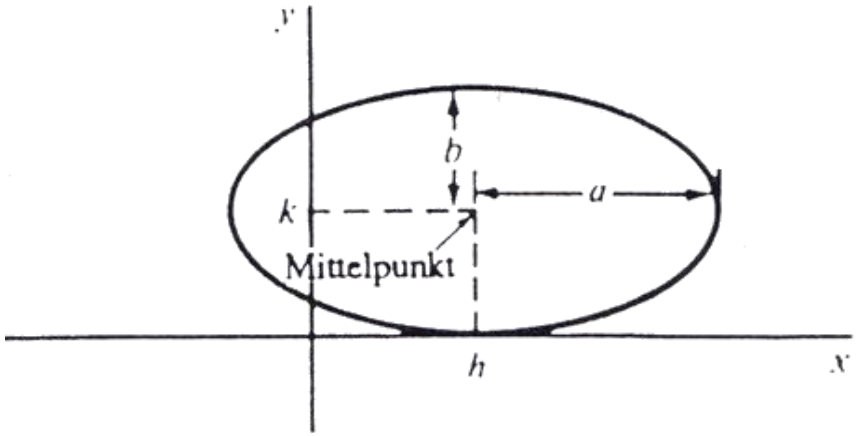
Ellipsendarstellung

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

  → Algorithmus von Kappel

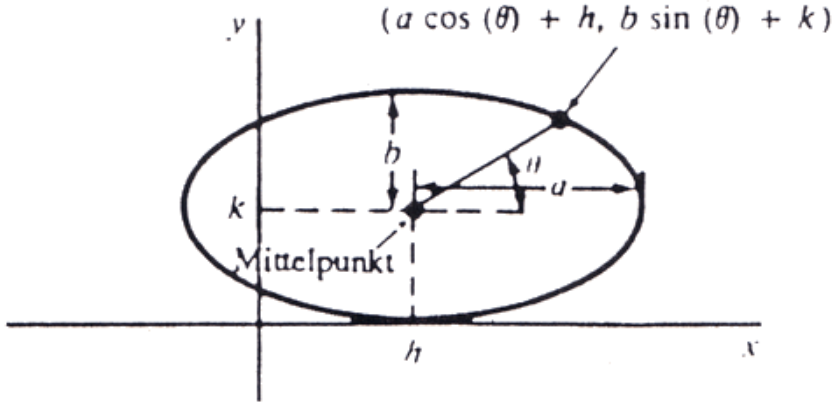
o Ellipsenerzeugung über die implizite Darstellung

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$$



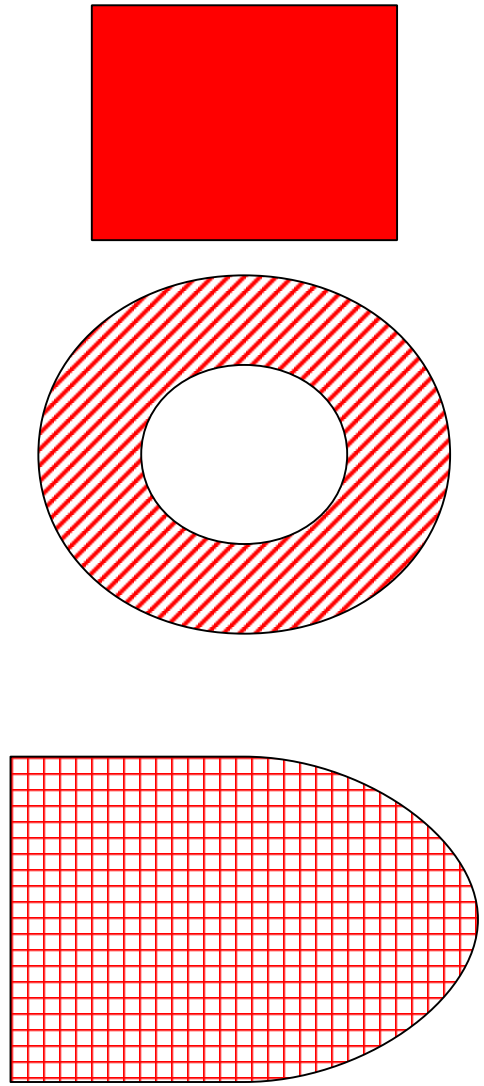
o Ellipsenerzeugung über die Parameterdarstellung

$$x = h + a\cos\theta \qquad y = k + b\sin\theta$$



3.4 Füllalgorithmen

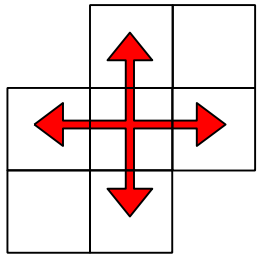
- Ziel: Füllen bzw. Einfärben eines begrenzten Bereiches oder Gebietes mit einer Füllfarbe oder einem Muster bzw. einer Schraffur
- Beispiele:
  - Balkendiagramme
  - Flächen
  - Körper
- Beschreibung der zu füllenden Gebiete erfolgt geometrisch
  - Durch Ecken, Strecken, Polygone, Kreise (randdefiniert)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



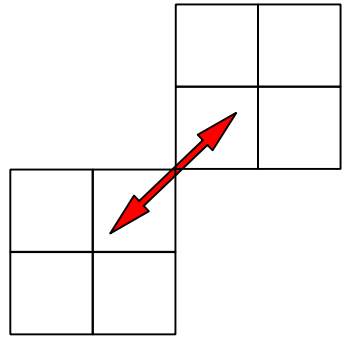
3.4 Füllalgorithmen

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

3.4 Füllalgorithmen

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

3.4 Füllalgorithmen

Scan-Line-Methode

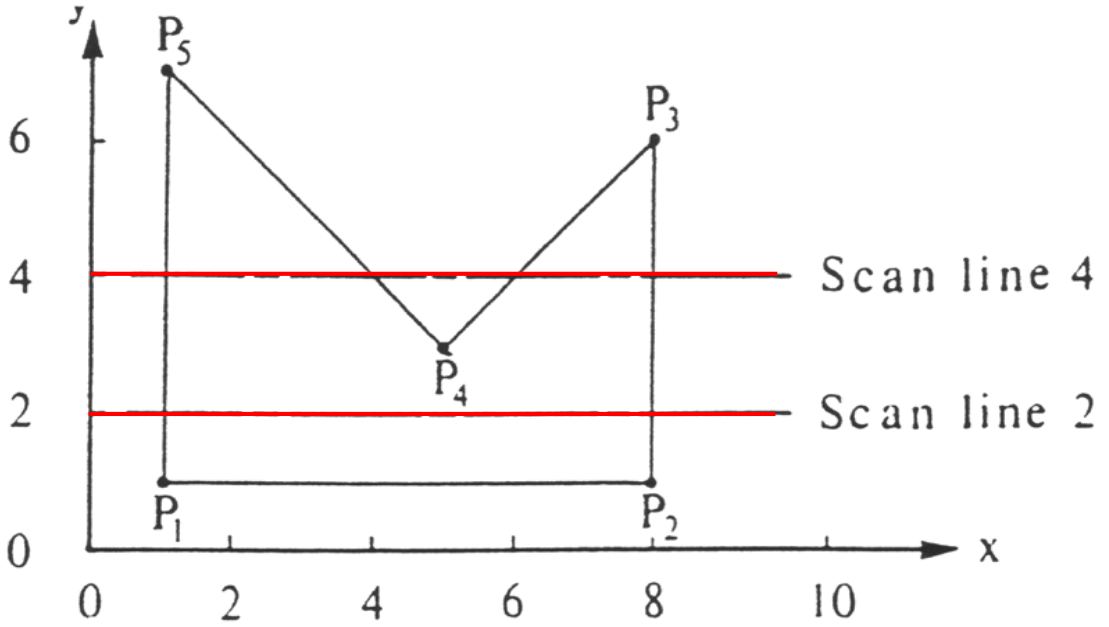
- // einfachster Ansatz:
- for (y = ymin; y <= ymax; y++) {
  - // row, Zeile
  - for (x = xmin; x <= xmax; x++) {
    - // column, Spalte
    - if (Inside(polygon, x, y) {
      - SetPixel(x,y);
    - }
  - }
- }

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

3.4 Füllalgorithmen

Scan-Line-Methode

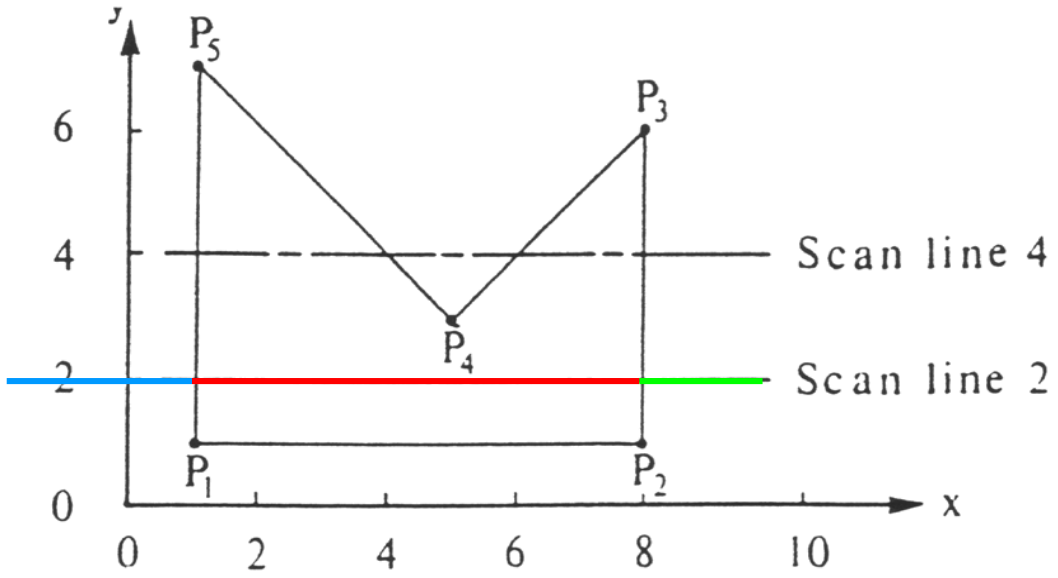
- Ausnutzung von Zeilenkohärenz
  - Benachbarte Pixel auf einer Zeile besitzen höchstwahrscheinlich die gleichen Intensitätswerte
  - Pixelcharakteristik (Intensität) ändert sich nur dort, wo ein Schnittpunkt einer Polygonkante mit einer Scan Line vorliegt, d.h. der Bereich zwischen zwei Schnittpunkten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



3.4 Füllalgorithmen

Scan-Line-Methode

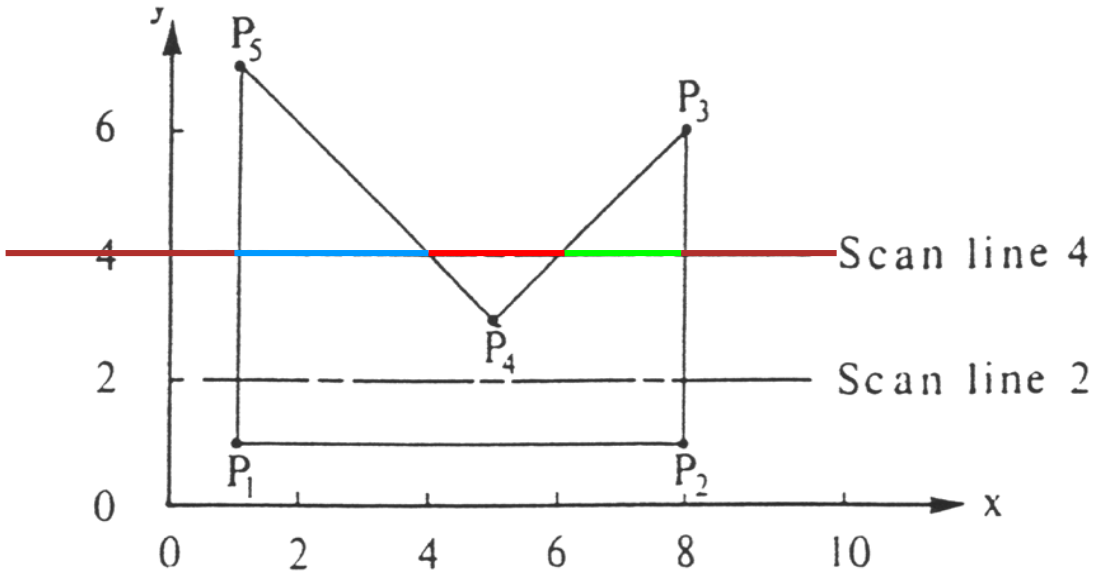
- Scan-Line $y = 2$
- Unterteilung in 3 Bereiche:
  - $x < 1$ außerhalb des Polygons
  - $1 \leq x \leq 8$ innerhalb des Polygons
  - $x > 8$ außerhalb des Polygons



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

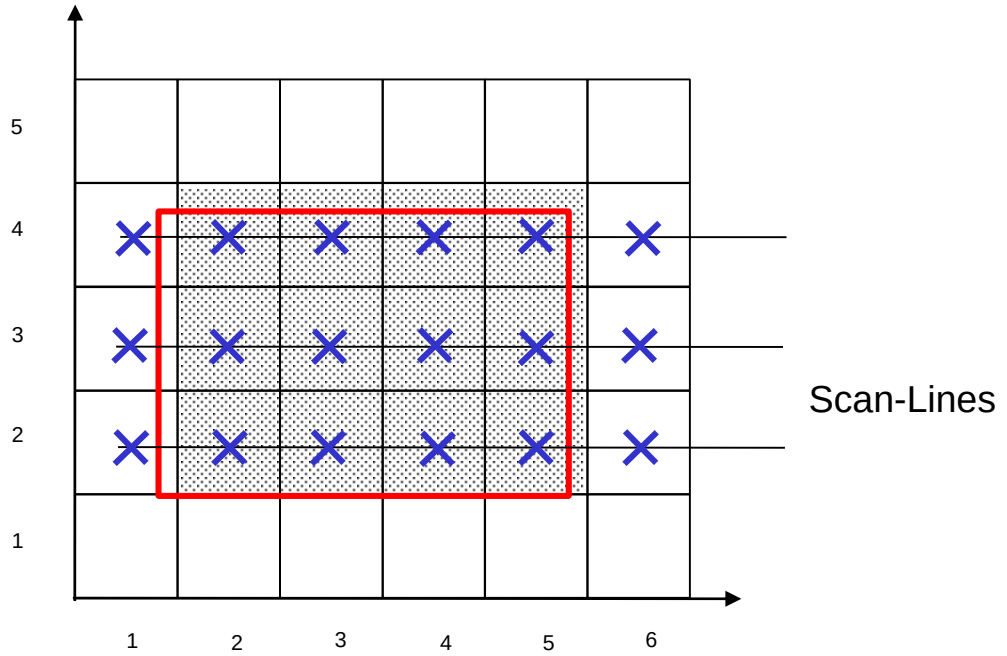


3.4 Füllalgorithmen

Scan-Line-Methode

- 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!



3.4 Füllalgorithmen

Scan-Line-Methode

- Probleme treten auf, wenn die Scan-Line das Polygon in einer Ecke schneidet
  → 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 Extrema, so zählt der **Schnitt zweifach**
    - ist die Ecke kein lokales Extrema, so zählt der **Schnitt nur einfach**
- Einfacher Kanten-Listen-Algorithmus: Ordered Edge List Algorithm
- Funktionsweise:
  - Preprocessing
  - Scan Conversion

3.4 Füllalgorithmen

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)

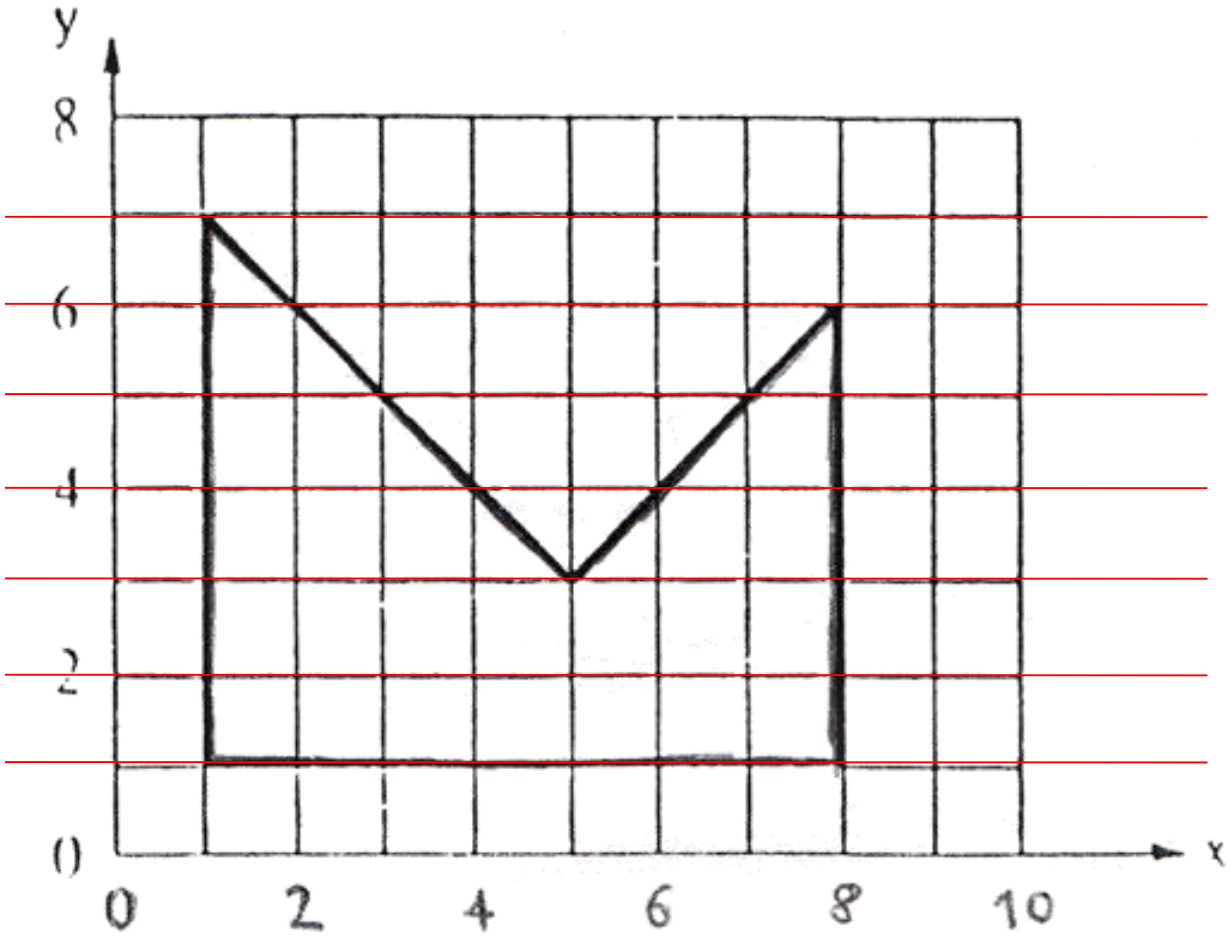
(1,5), (3,5), (7,5), (8,5)

(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)

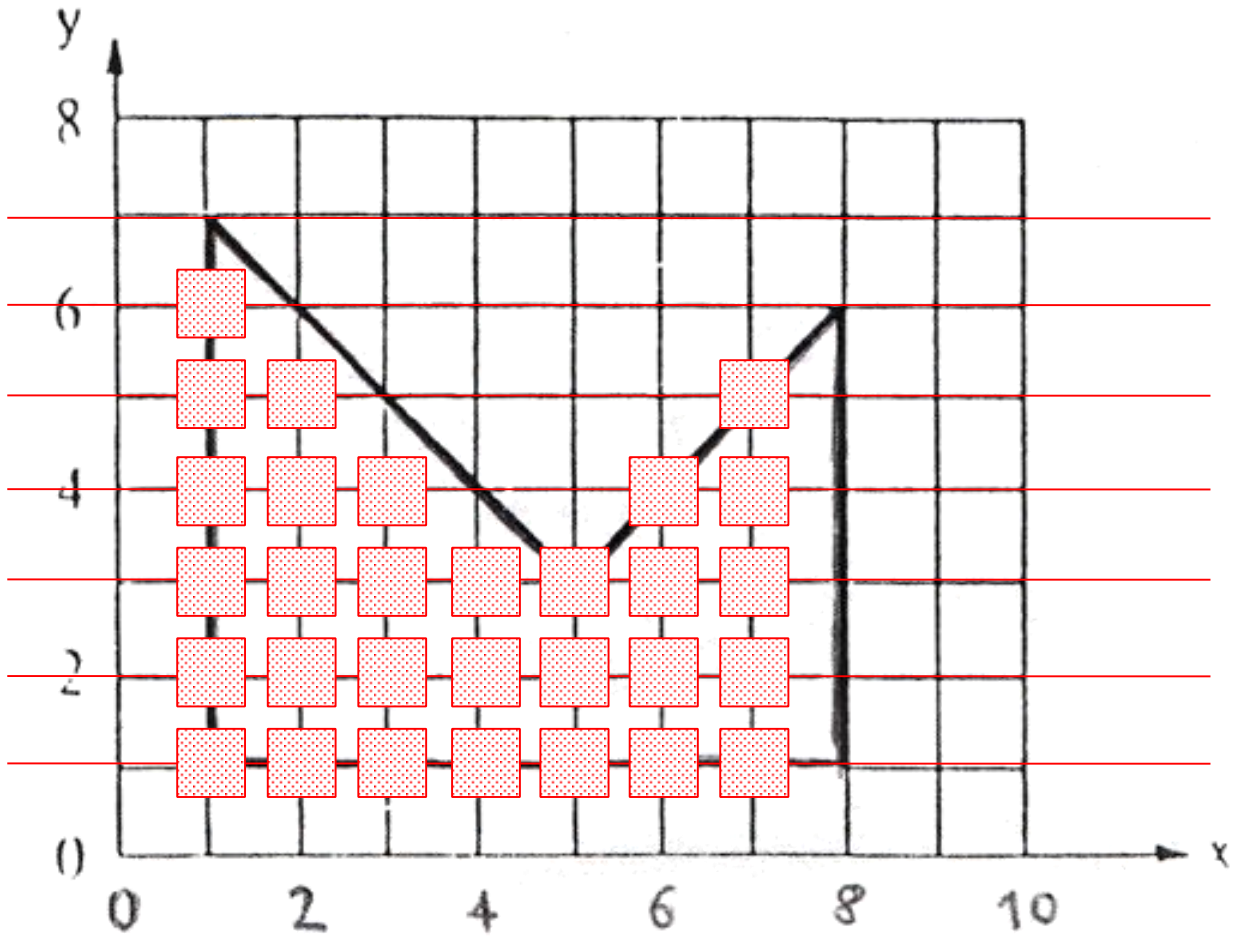
(1,5), (3,5), (7,5), (8,5)

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

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

(1,2), (8,2)

(1,1), (8,1)



3.4 Füllalgorithmen

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:
  i. **Boundary-Fill-Algorithmus** für randdefinierte Gebiete
  ii. **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

3.4 Füllalgorithmen

Saatpunkt-Methode / Seed-Fill

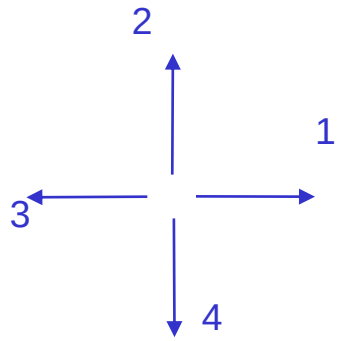
i. Boundary-Fill-Algorithmus
   - Input
     - Startpunkt (Saatpunkt)
     - Farbe der **Begrenzungskurve**
     - Füllfarbe oder Muster
   - Algorithmus
     - Wiederhole
       - Vom Startpixel ausgehend werden rekursiv Nachbarpixel umgefärbt
     - Bis Pixel mit **der Farbe der Begrenzungskurve oder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

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)

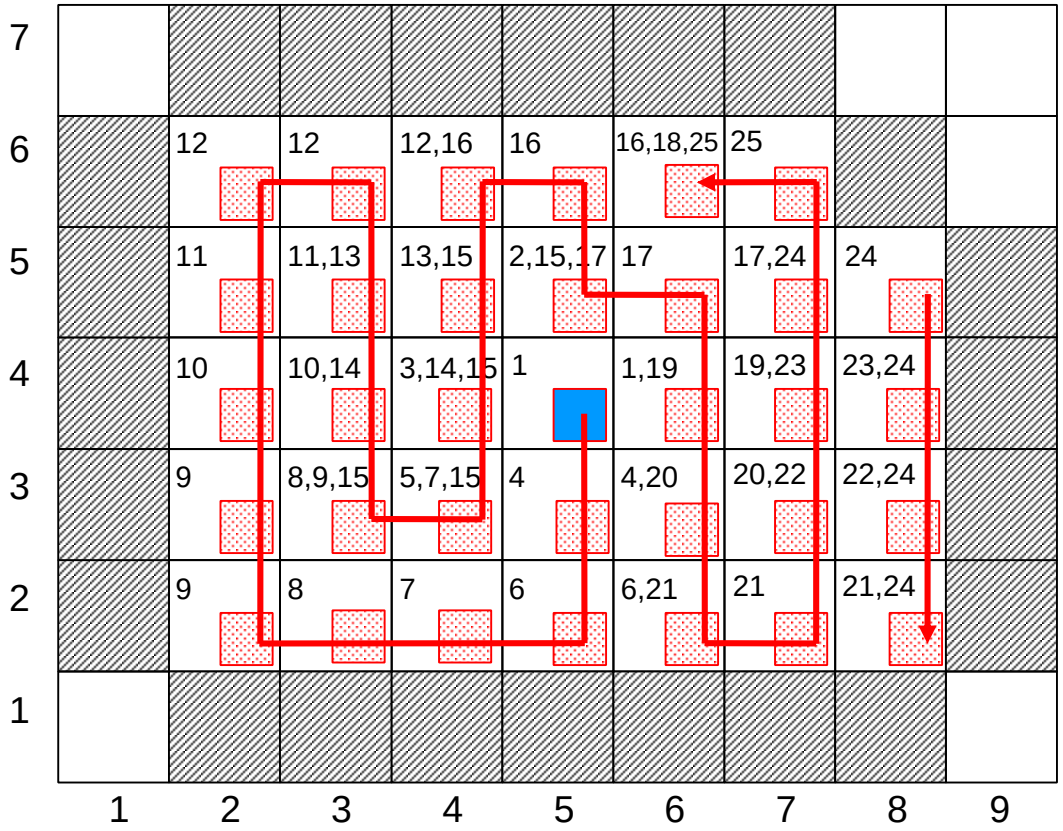


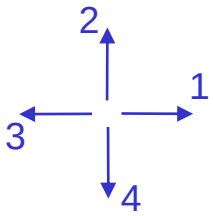

```
Empty(stack);
Push(stack, seed-pixel);

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

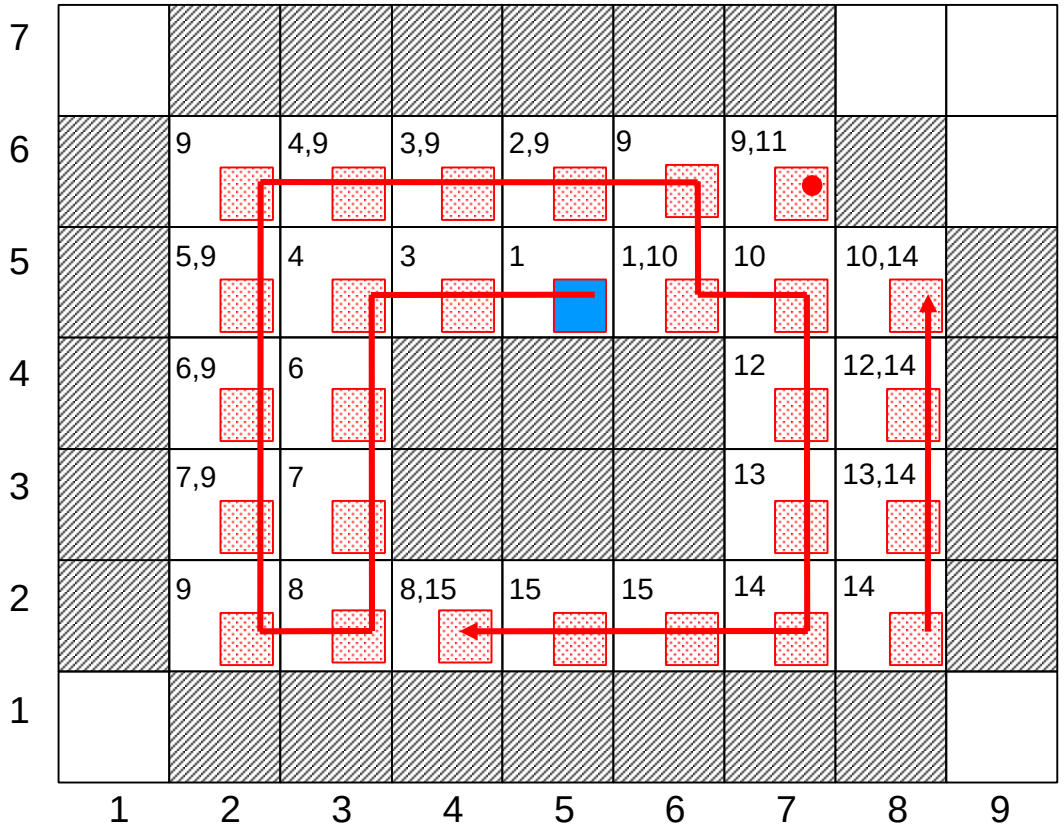


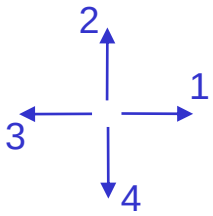


3.4 Füllalgorithmen

Saatpunkt-Methode / Seed-Fill

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





3.4 Füllalgorithmen

- 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)

3.5 Aliasing

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, aber beseitigt sie 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**

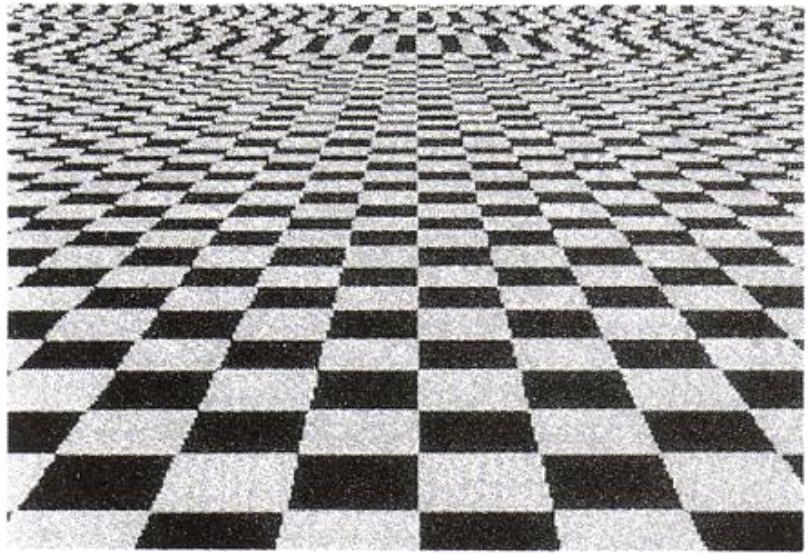
- Üblicherweise treten visuelle Artefakte auf, wenn die **Periodizität** (zum Beispiel Kachelmuster) in der Textur die **Größenordnung von Pixeln** erreicht

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

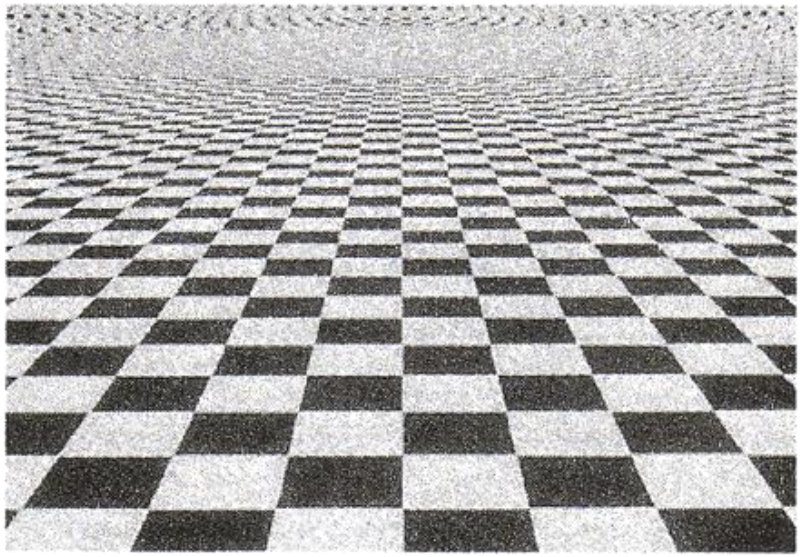
3.5 Aliasing

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) entsprechen
- 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

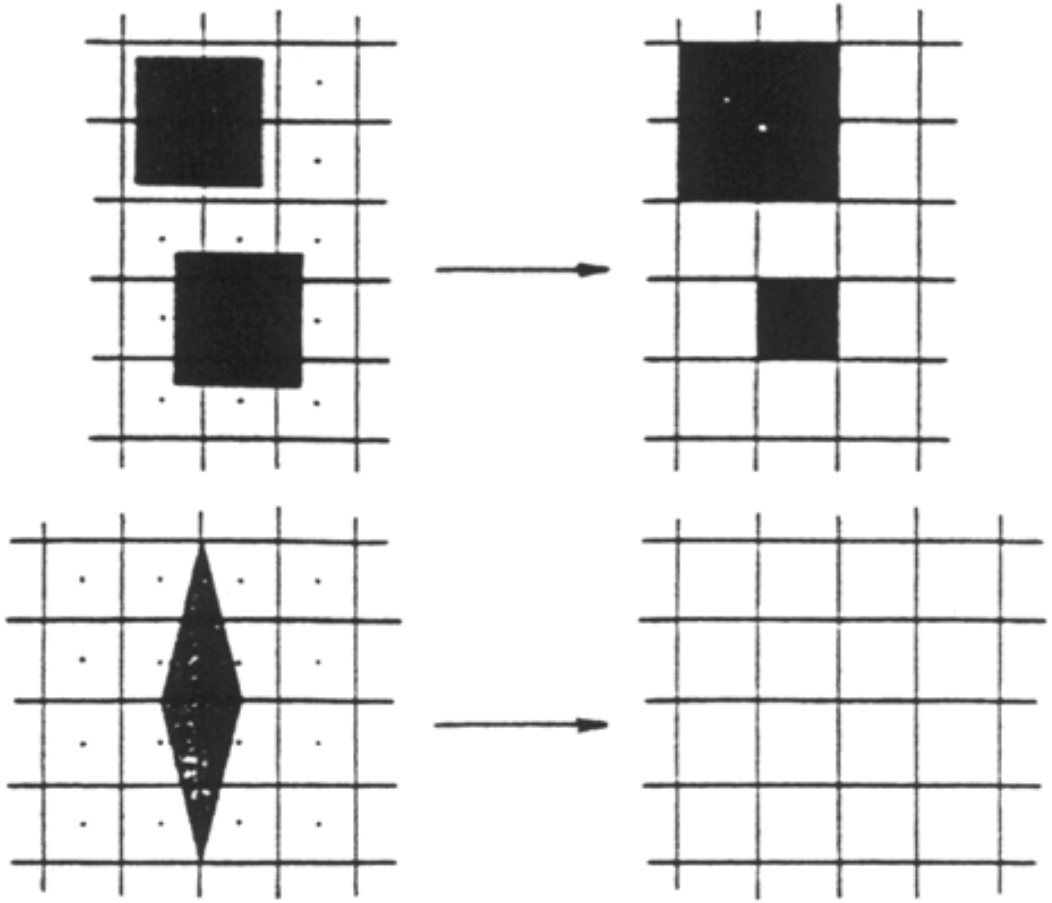
- Treppeneffekte bei einer gerasterten Geraden
- Graustufen, um Treppeneffekte zu verringern

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

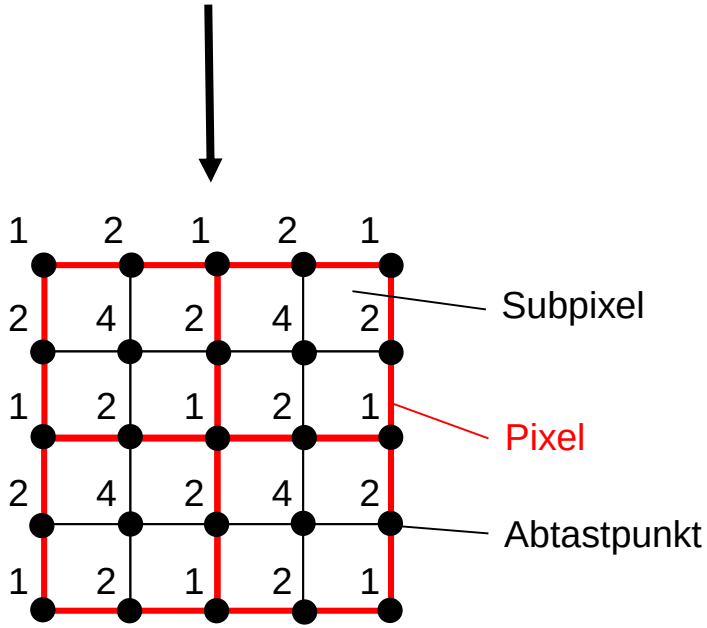
1	2	1
2	4	2
1	2	1

5 x 5

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

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

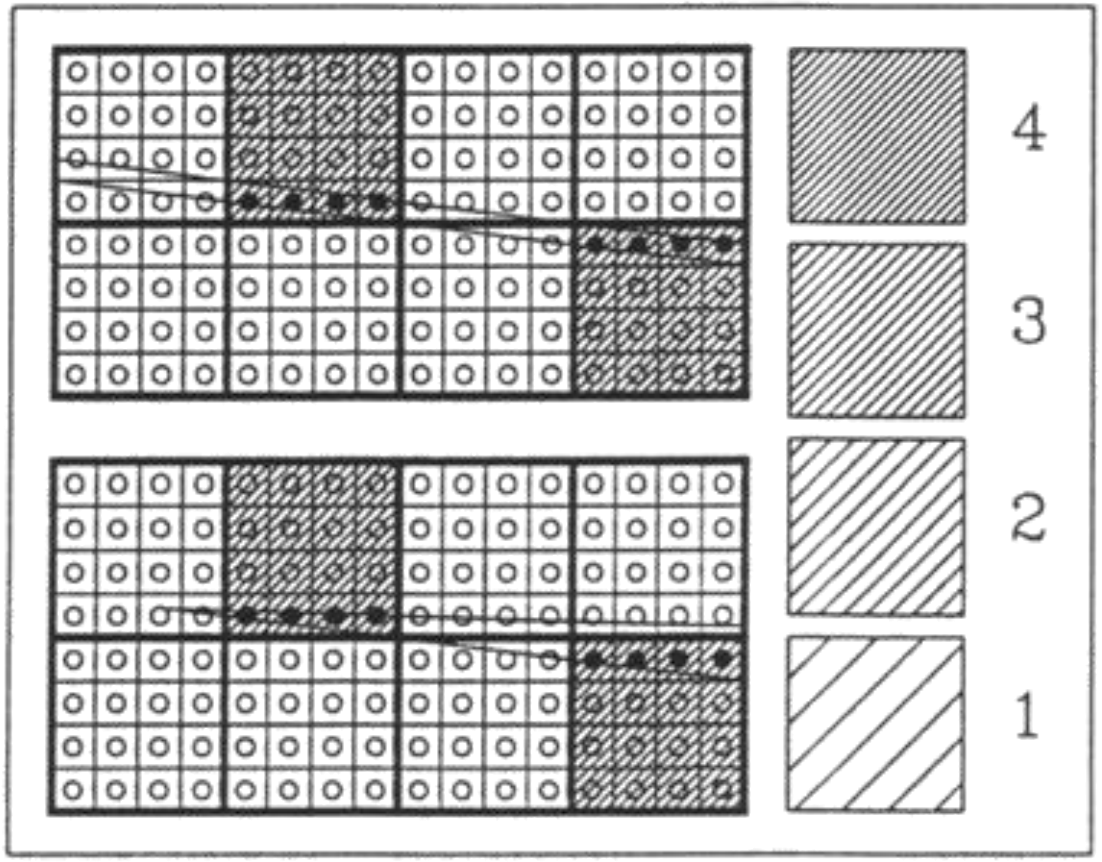


3.5 Aliasing

Anti-Aliasing Verfahren

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

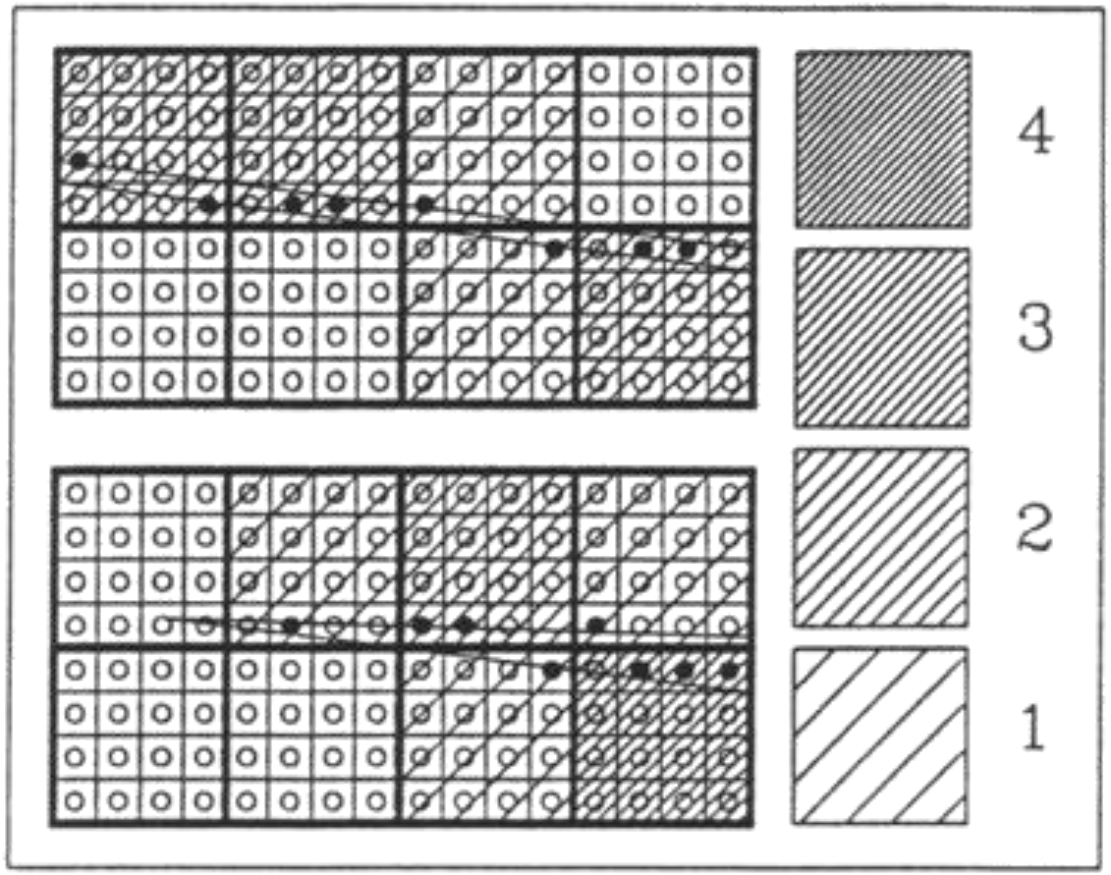
- Gesetzte Subpixel sind schwarz.
- Die Schraffur zeigt den Grauwert abhängig von der Anzahl der gesetzten Subpixel
- Oben:
  Linie verschwindet stellenweise,
  da keine Subpixel getroffen werden!
- Unten:
  Analog für das Dreieck



3.5 Aliasing

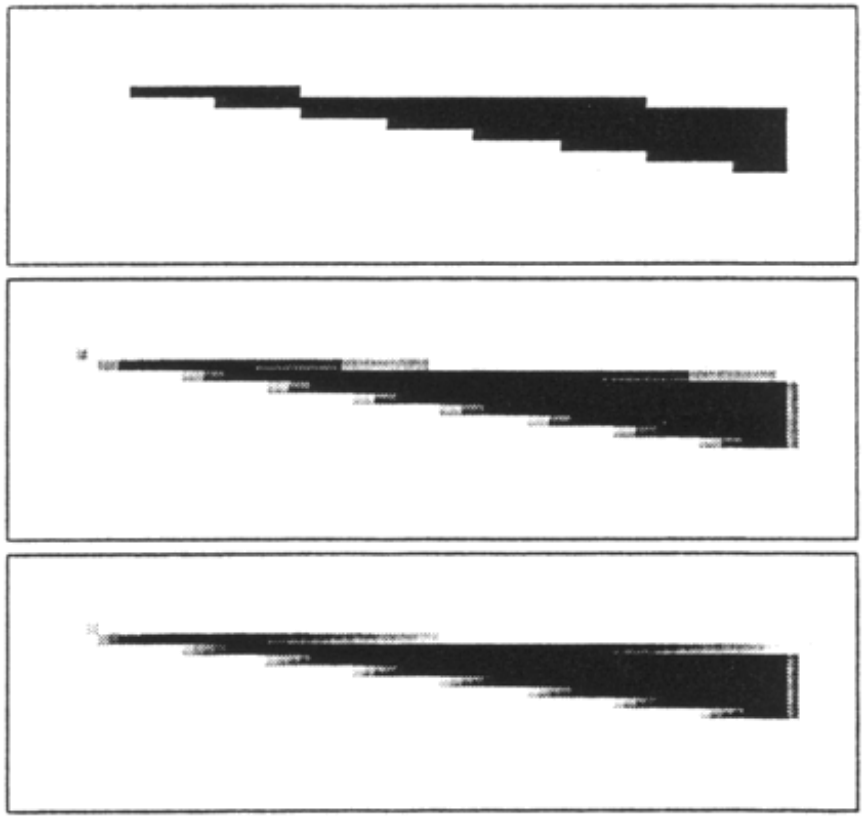
Anti-Aliasing Verfahren

- 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

3.5 Aliasing

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)
  - 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)