Inhaltsverzeichnis

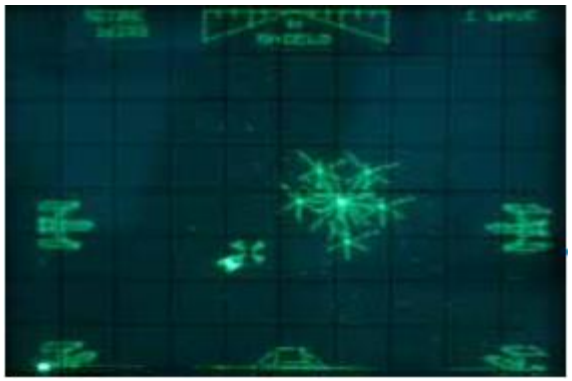
§1 Hardwaregrundlagen
§2 Transformationen und Projektionen
§3 Repräsentation und Modellierung von Objekten
§4 Rasterung
- 4.1 Rasterung von Linien
- 4.2 Rasterung von Geraden
- 4.3 Rasterung von Kreisen
- 4.4 Polygone & Füllalgorithmen
- 4.5 Rasterartefakte und Aliasing

§5 Visibilität und Verdeckung
§6 Rendering
§7 Abbildungsverfahren
§8 Freiformmodellierung
Anhang: Graphiksprachen und Graphikstandards
Anhang: Einführung in OpenGL

4. Rasterung

Darstellungsmöglichkeiten

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



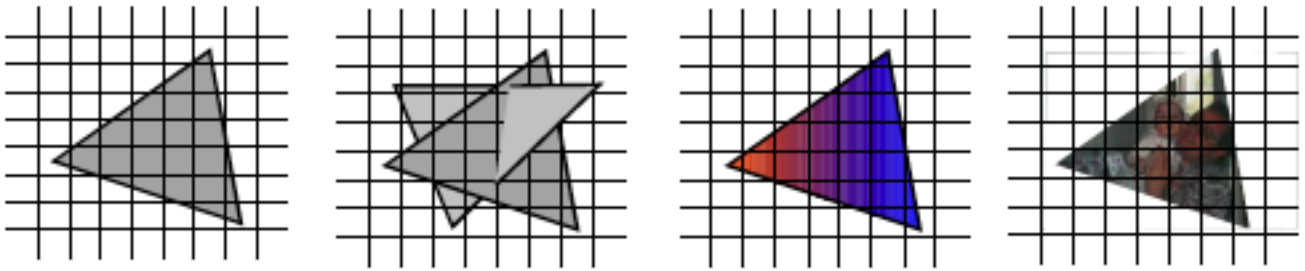
4. Rasterung

Darstellungsmöglichkeiten

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

4. Rasterung

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



4.1 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).

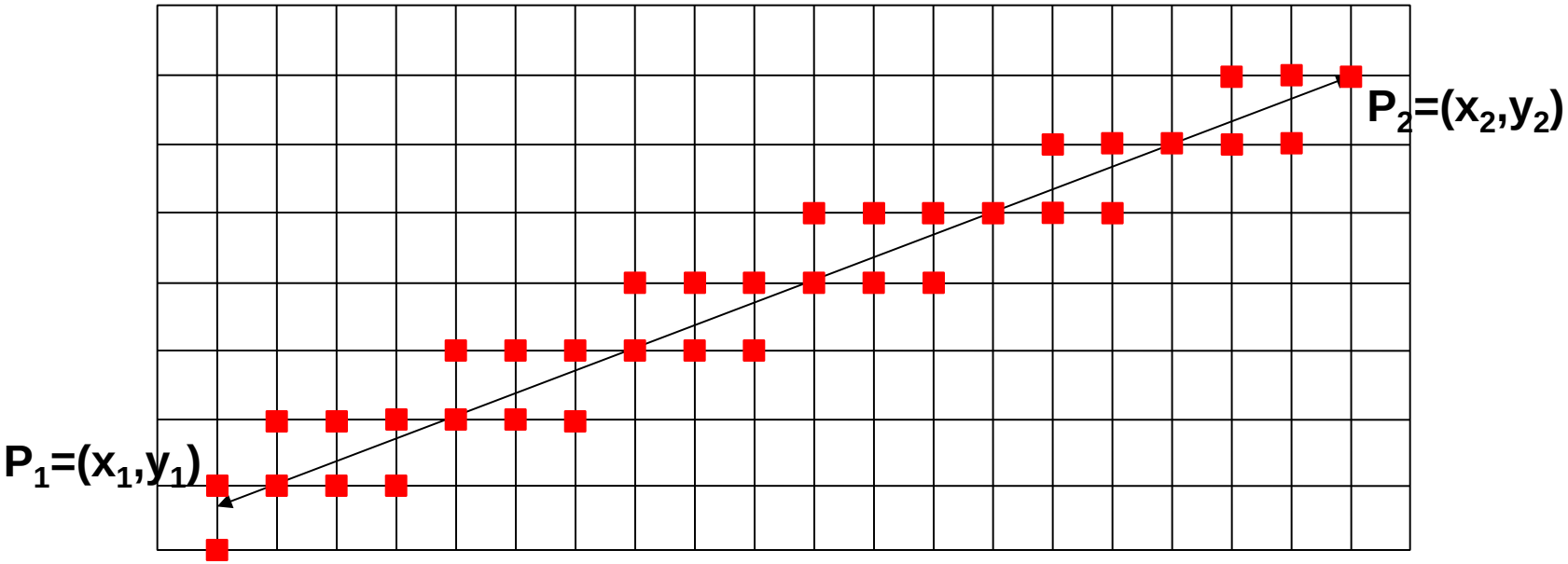
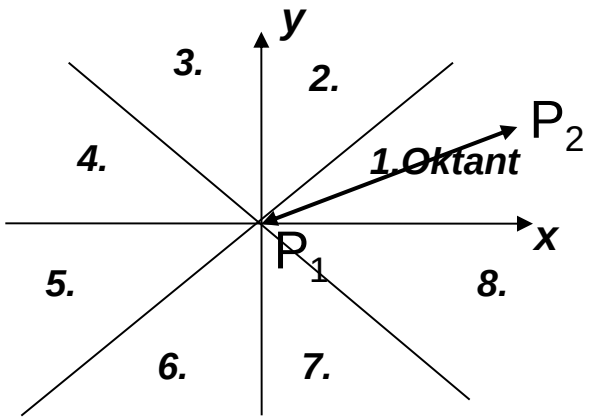


Abb. 2.1: Mögliche Rasterkandidaten

4.1 Rasterung von Linien

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

Fallunterscheidung in Oktanten (nach Steigung)



Hier:

- Ursprung des zugeordneten Koordinatensystems in P_1
- 1. Oktant
- P_1 und P_2 auf Rastergitter

4.1 Rasterung von Linien

DDA-Algorithmus für Geraden [1965]

Digital Differential Analyzer / Digitaler Integrierer

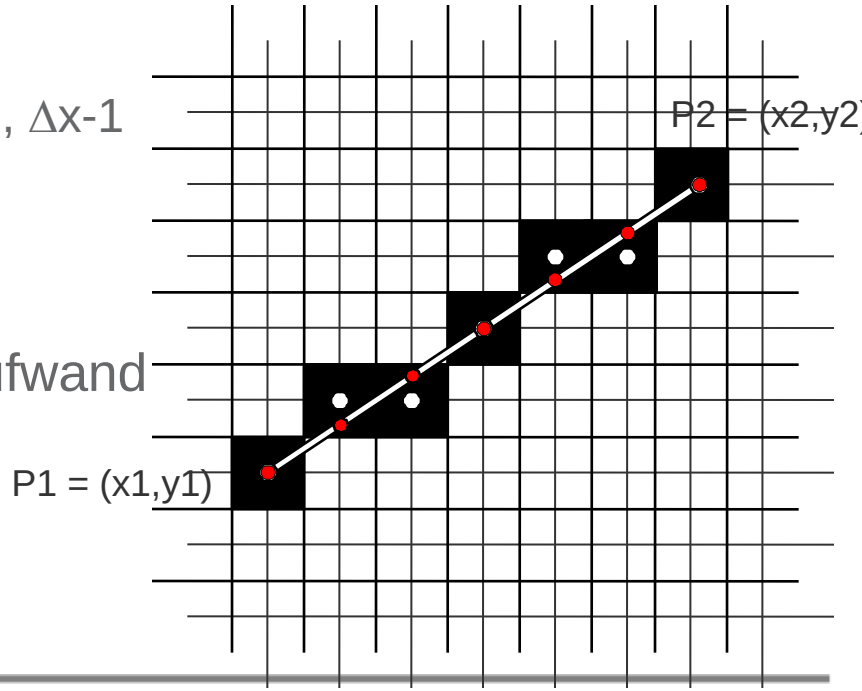
- Ein DDA-Algorithmus **generiert eine Kurve** (nicht nur eine Gerade) aus einer beschreibenden Differentialgleichung

$\Delta x = x_2 - x_1 \; (= x_{i+1} - x_i)$

$\Delta y = y_2 - y_1 \; (= y_{i+1} - y_i)$; $\quad -1 \leq \Delta x / \Delta y \leq 1$

$y_{i+1} = \Delta y / \Delta x \; * x_{i+1} + b$; $\quad x_{i+1} = x_1 + i,\ i=1,\ldots,\ \Delta x-1$

$\text{Plot}(x_i, \text{round}(y_i)) = \text{plot}(x_i, \text{Floor}(1/2 + y_i))$



- **Dezimalbrüche** erhöhen Rechenaufwand
  - Fließkomma-Multiplikation
  - Addition
  - Rundung

4.1 Rasterung von Linien

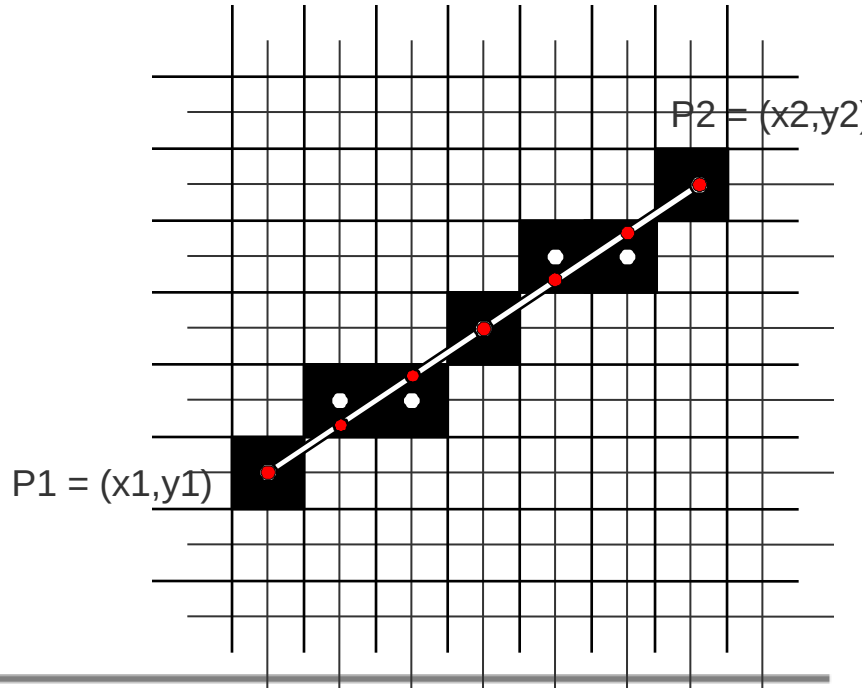
DDA-Algorithmus für Geraden

Digital Differential Analyzer / Digitaler Integrierer

- Verbesserung: **Inkrementeller** Algorithmus

$$y_{i+1} = \Delta y / \Delta x \ * x_{i+1} \ + b$$
$$= \Delta y / \Delta x \ * (x_{i+1} + (x_i - x_i)) + b$$
$$= \Delta y / \Delta x \ * x_i + b + \Delta y / \Delta x \ * (x_{i+1} - x_i)$$
$$= y_i + \Delta y / \Delta x \ *1$$

```
x = x1; y = y1; m=dy/dx;
For (x=x1; x<=x2; x++) {
   Plot(x,y)
   y = round(y+m);
}
```

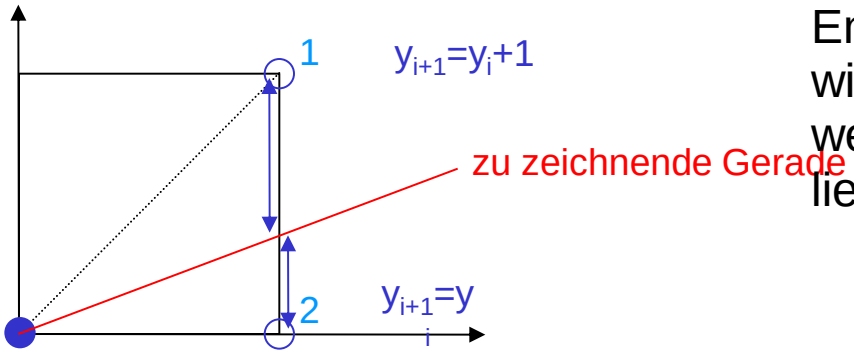


4.2 Rasterung von Geraden

Bresenham-Algorithmus für Geraden [1965]

- Idee: Abhängig von der **Steigung der Geraden** wird die x- oder y-Koordinate immer um 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.

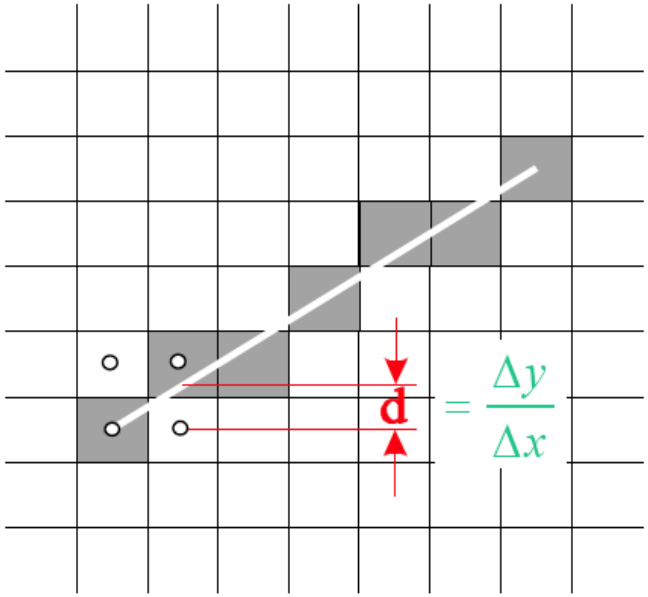
4.2 Rasterung von Geraden



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

$$y(x) = m \cdot x + c = m \cdot x$$

$$y'(x) = m = \frac{\Delta y}{\Delta x} = d$$



- Wenn d> 1/2, d<=1/2

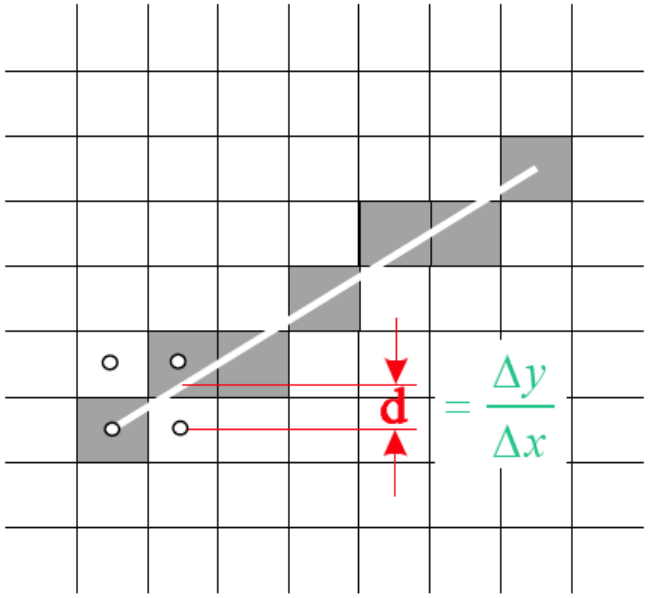
4.2 Rasterung von Geraden

$$y(x) = m \cdot x + c = m \cdot x$$

$$y'(x) = m = \frac{\Delta y}{\Delta x} = d$$

$$d > \frac{1}{2}, d \leq \frac{1}{2}$$

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



- Zur Realisierung wird eine **Entscheidungsgröße E** eingeführt, welche die Abweichung zwischen dem exakten Punkt und der Mitte zwischen den beiden möglichen Rasterpunkten angibt.
- Das **Vorzeichen von E** dient dann als Kriterium für die Rundung auf den nächsten Rasterpunkt:

  E>0: x++; y++; E + Δy/Δx -1

  E≤0: x++; E + Δy/Δx

4.2 Rasterung von Geraden

- Nur lineare Operationen, aber Dezimalbrüche (1/2)
- Skalieren um $2\Delta x$, um Dezimalbrüche zu vermeiden

$E = \Delta y/\Delta x - 1/2 \quad \Rightarrow \quad E' = 2\Delta x(\Delta y/\Delta x - 1/2) = 2\Delta y - \Delta x$

$E' \leq 0$:

x++;
E‘:= E‘ + 2 Δy

$E' > 0$:

x++; y++;

E‘:= E‘ + 2 Δy - 2Δx

4.2 Rasterung von Geraden

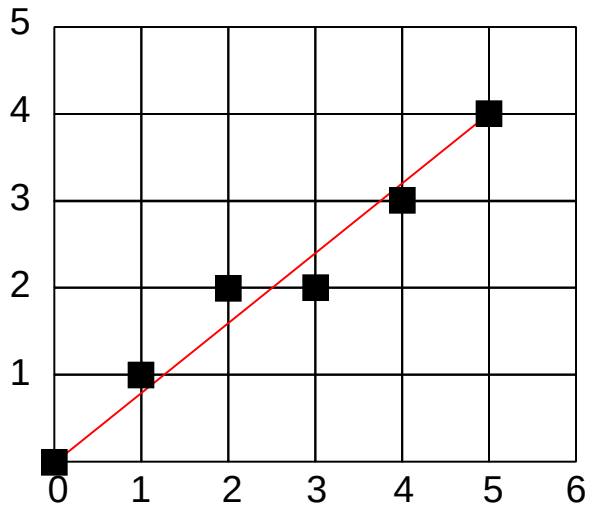
- Bresenham-Algorithmus für den ersten Oktanten, der ausschließlich ganzzahlige Operanden verwendet, mit e:=E‘, dx := Δx, dy := Δy:

```
// (x1, y1), (x2, y2) Ganzzahlig, x1 < x2
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 = y+1;
    e = e-2*dx;
  }
  x = x+1;
  e = e+2*dy;
}
plot(x, y);
```

4.2 Rasterung von Geraden

Beispiel:

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

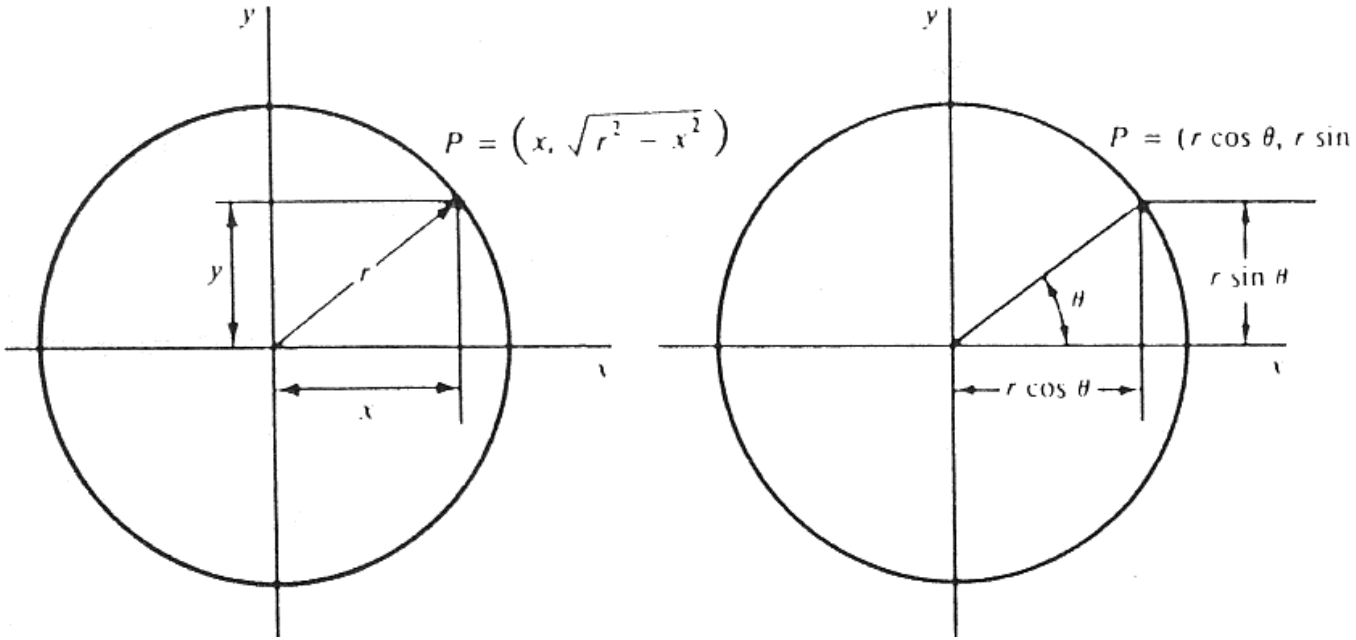


dx	dy	x	y	e	i	plot
5	4	0	0	3	1	(0,0)
			1	-7		
		1		1	2	(1,1)
			2	-9		
		2		-1	3	(2,2)
		3		7	4	(3,2)
			3	-3		
		4		5	5	(4,3)
			4	-5		
		5		3	6	
						(5,4)

4.3 Rasterung von Kreisen

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

- (i) implizit: $f(x,y) = (x-x_M)^2 + (y-y_M)^2 - r^2 = 0$
- (ii) Parameter: $x(\Theta) = x_M + r\cos(\Theta)$, $y(\Theta) = y_M + r\sin(\Theta)$, $\Theta \in [0, 2\pi[$

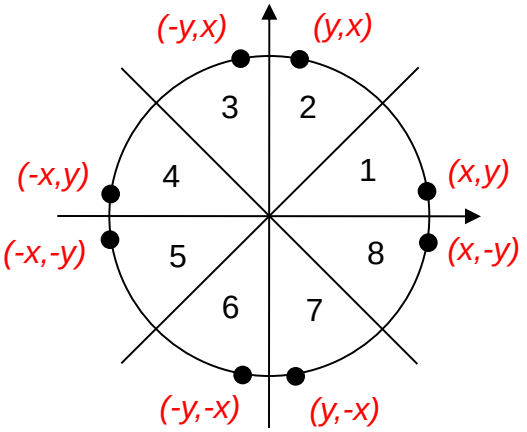


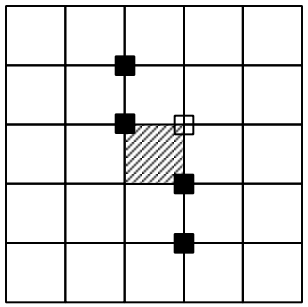
Nachteil: Beide Methoden zur Kreisdarstellung sind rechenaufwändig und erfordern höhere Rechenoperationen.

4.3 Rasterung von Kreisen

Bemerkungen

- Mit der Berechung eines Kreispunktes sind durch **Symmetrie** 7 weitere Kreispunkte gegeben.
- Für eine gleichmäßige Rasterung müssen die Pixel entlang des Kreises **möglichst gleichmäßig** verteilt sein.





von jedem Rasterquadrant dürfen nur 2 Eckpunkte gesetzt werden

- Die Bewertung der Kreisapproximation im Raster ist subjektiv. Ein häufig verwendetes Kriterium ist die **Minimierung des Residuums** $|x_i^2 + y_i^2 - r^2|$.

4.3 Rasterung von Kreisen

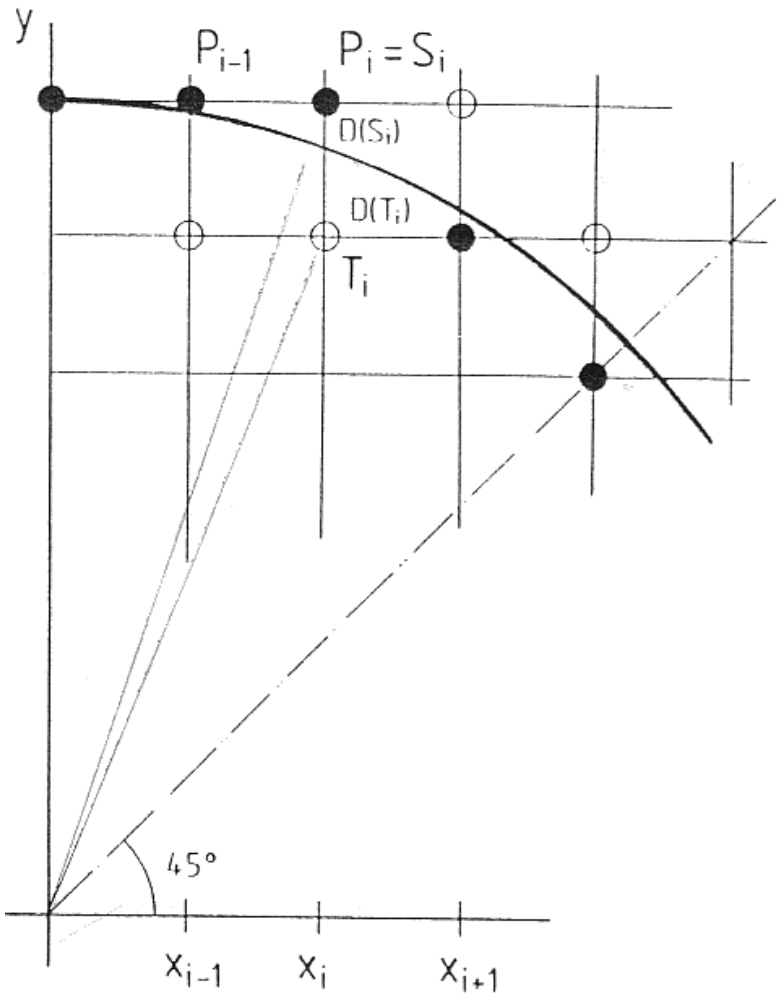
Bresenham-Algorithmus für Kreise

- Beste Approximation über Entscheidungsgröße d_i mittels minimalem Residuum.

- Hier:
  - $(x_M, y_M) = (0, 0)$
  - Startpunkt ist Rasterpunkt
  - $r \in$ IN
  - 2. Oktant

$$D(S_i) = | (x_{i-1}+1)^2 + y_{i-1}^2 - r^2 |$$

$$D(T_i) = | (x_{i-1}+1)^2 + (y_{i-1}\mathbf{-1})^2 - r^2 |$$



4.3 Rasterung von Kreisen

Die Entscheidungsgröße $d_i = D(S_i)-D(T_i)$ misst die Abstände zum oberen und unteren Rasterpunkt

- Für $d_i > 0$:
  wähle $P_i = T_i$ **als nächsten Punkt**, also $x_i = x_{i-1}+1$, $y_i = y_{i-1}-1$
- Für $d_i <= 0$:
  wähle $P_i = S_i$ **als nächsten Punkt**, also $x_i = x_{i-1}+1$, $y_i = y_{i-1}$
- Innerhalb des 2. Oktanten (Kreisfunktion monoton fallend, Steigung zwischen 0 und -1) gilt speziell:
  - $D(S_i)$ unter dem Betrag > 0, dh. S_i liegt immer außerhalb der Kreislinie
  - $D(T_i)$ unter dem Betrag < 0, dh. T_i liegt immer innerhalb der Kreislinie

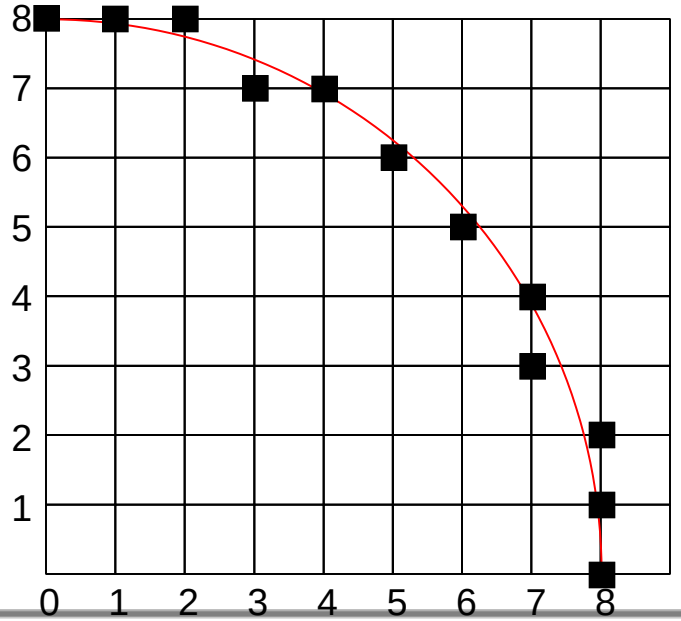
$$d_i = (x_{i-1}+1)^2 + y_{i-1}^2 - r^2 + (x_{i-1}+1)^2 + (y_{i-1}-1)^2 - r^2$$

4.3 Rasterung von Kreisen

und damit rekursiv: (berechne d_{i+1} aus d_i)

- $d_{i+1} = d_i + 4\,(x_i - y_i) + 2$ für di>0
  $d_{i+1} = d_i + 4\,x_i + 2$ für d_i<=0

  mit Startwert $d_1 = 3-2r$ ($x_0=0$, $y_0=r$)
- Beispiel: r=8



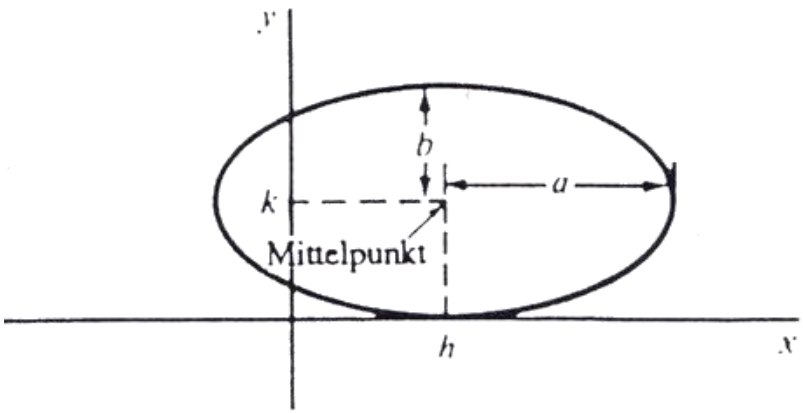
d	x	y
	0	8
-13		
	1	8
-7		
	2	8
3		
	3	7
-11		
	4	7
7		
	5	6

4.3 Rasterung von Kreisen

Ellipsendarstellung

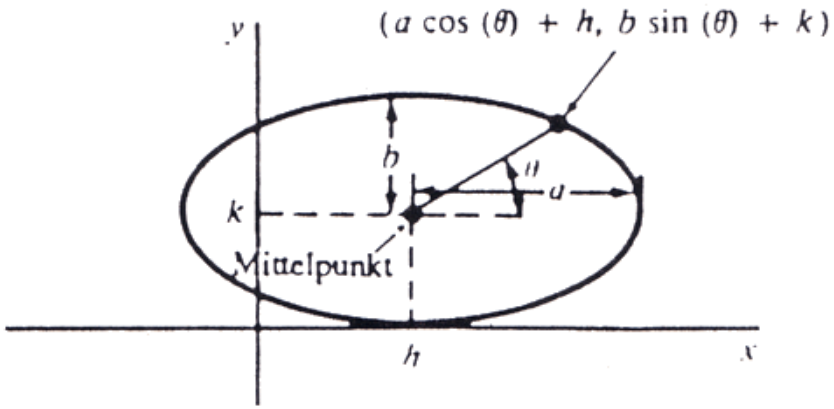
- Ellipsenerzeugung über die implizite Darstellung

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



- Ellipsenerzeugung über die Parameterdarstellung

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



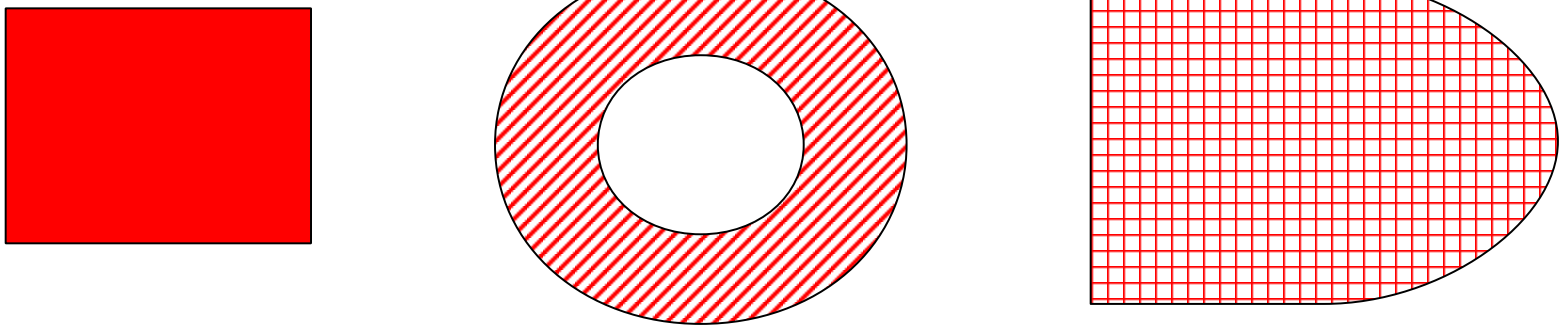
- Zur Rasterung betrachtet man die Ellipsen in **Normalform**, und mit zu den **Koordinatenachsen parallelen Hauptachsen**.

  => Algorithmus von Kappel

4.4 Polygonen & Füllalgorithmen

Ziel: **Füllen** bzw. **Einfärben** eines begrenzten Bereiches oder Gebietes mit einer Füllfarbe oder einem Muster bzw. einer Schraffur

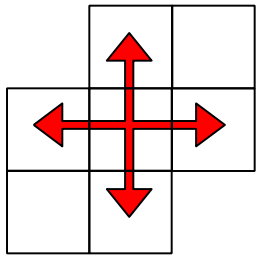
- Beispiele: Balkendiagramme, Flächen, Körper etc.



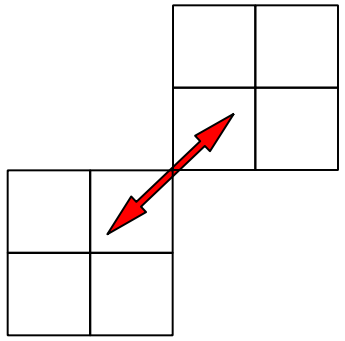
Beschreibung der zu füllenden Gebiete erfolgt geometrisch, zB. durch Ecken, Strecken, Polygone, Kreise, etc. oder durch Pixel (inhalts- oder randdefiniert)

4.4 Polygonen & Füllalgorithmen

- Wichtig: **Zusammenhang** von Gebieten
- Man unterscheidet
  - 4-fach zusammenhängend (nur Horizontal- und Vertikalbewegung)
  - 8-fach zusammenhängend (zusätzlich Diagonalbewegung möglich)



4-fach zusammenhängend



8-fach zusammenhängend

4.4 Polygonen & 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

4.4 Polygonen & Füllalgorithmen

Scan-Line-Methode

- Auch **Rasterzeilen-Methode** oder **Scan-Conversion** genannt
- Arbeitet **zeilenweise** von oben nach unten
- Ein Pixel der aktuellen Zeile (Scan-Line) wird nur dann gezeichnet, **wenn es innerhalb** des Polygons liegt

```
// einfachster Ansatz:
for(y=y_min ; y<=y_max ; y++)          // row, Zeile
  for(x=x_min ; x<=x_max ; x++)        // column, Spalte
    if(Inside(polygon, x, y)
      SetPixel(x,y);
```

- für geometrisch als auch für pixelweise definierte Gebiete
- sehr langsam

=> Verbesserung durch Ausnutzung von Kohärenz

4.4 Polygonen & Füllalgorithmen

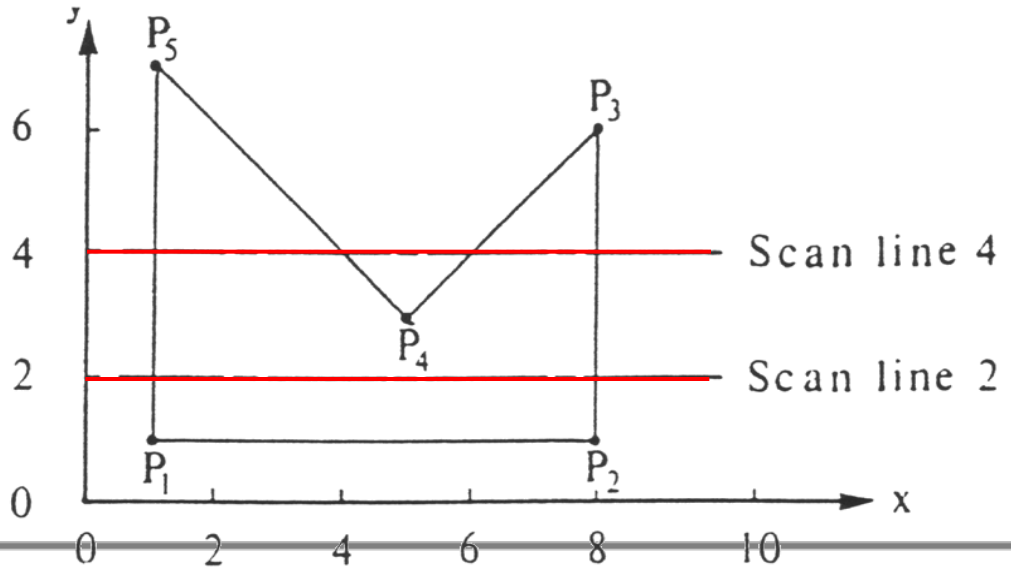
Scan-Line-Methode (Fortsetzung)

- Verfahren basiert auf dem Prinzip der **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

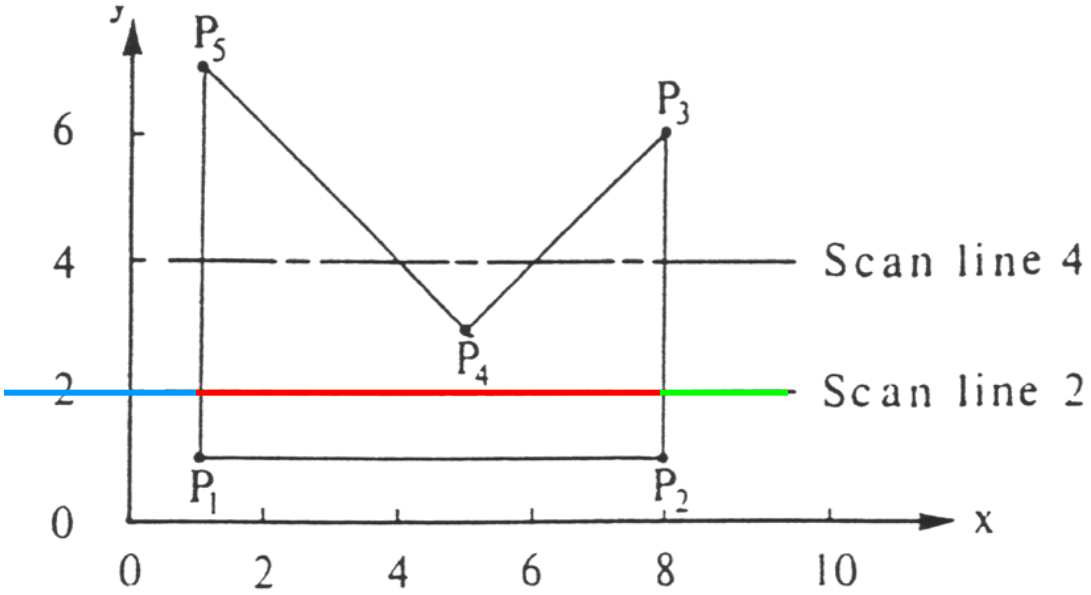
y=2: Schnitt mit Polygon für
x = 1, 8

y=4: Schnitt mit Polygon für
x = 1, 4, 6, 8



4.4 Polygonen & Füllalgorithmen

Scan-Line-Methode (Fortsetzung)



Scan-Line y=2, Unterteilung der Scan-Line in 3 Bereiche:

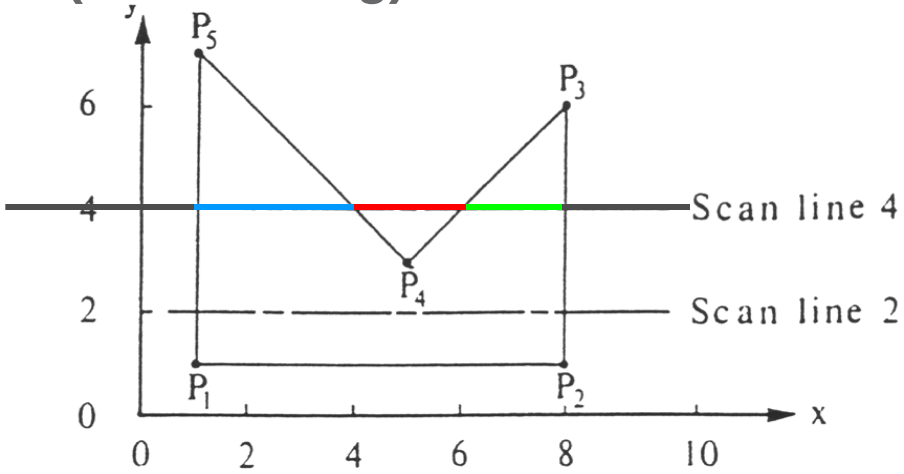
x<1: außerhalb des Polygons

1<=x<=8: innerhalb des Polygons

x>8: außerhalb des Polygons

4.4 Polygonen & Füllalgorithmen

Scan-Line-Methode (Fortsetzung)



Scan Line y=4, Unterteilung der Scan Line in 5 Bereiche:

x<1: außerhalb des Polygons

1<=x<=4: innerhalb des Polygons

4<x<6: außerhalb des Polygons

6<=x<=8: innerhalb des Polygons

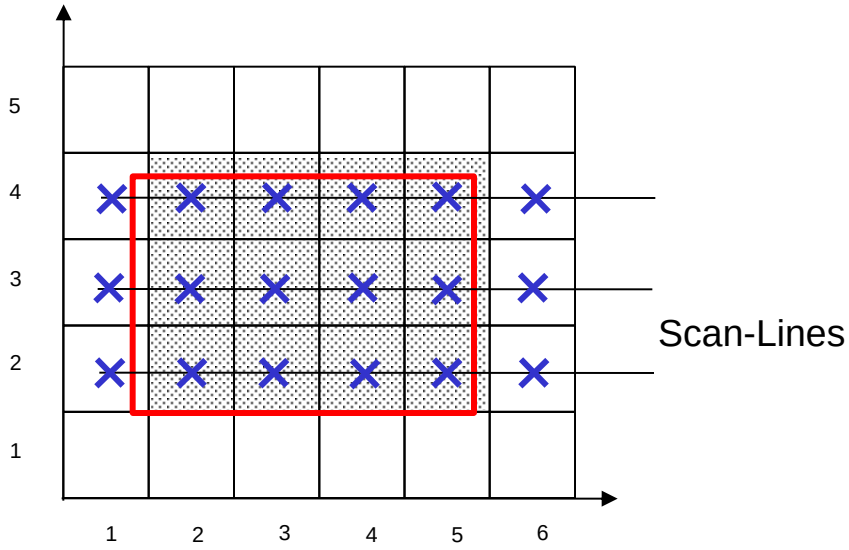
x>8: außerhalb des Polygons

4.4 Polygonen & Füllalgorithmen

Scan-Line-Methode (Fortsetzung)

- **Günstige Wahl** der Scan Lines
- Scan-Lines verlaufen durch die **Pixelmitte**
- Pixelaktivierung erfolgt, falls die Pixelmitte des Pixels **rechts (>=) vom Schnittpunkt** mit dem Polygon (Innenbereich) liegt.

Vorsicht: hier sind die **Pixel-zeilen und -spalten nummeriert**, nicht die Koordinaten der Achsen!



4.4 Polygonen & Füllalgorithmen

Scan-Line-Methode (Fortsetzung)

- Probleme treten bei **Singularitäten** auf,
  dh. die Scan-Line schneidet das Polygon in einer Ecke.

  => Betrachte **lokale Extrema**
- Lokales 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**

4.4 Polygonen & Füllalgorithmen

Scan-Line-Methode (Fortsetzung)

- Einfacher Kanten-Listen-Algorithmus: Ordered Edge List Algorithm
- Funktionsweise: Preprocessing + Scan Conversion
- **Preprocessing**
  - Ermittle (zB. mit Bresenham- / DDA-Algorithmus) für jede Polygonkante die **Schnittpunkte mit den Scan-Lines** in der Pixelmitte.
    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.

4.4 Polygonen & Füllalgorithmen

Scan-Line-Methode (Fortsetzung)

- **Scan-Conversion**
  - Betrachte jeweils zwei direkt **aufeinander folgende Schnittpunkte** (x_1, y_1) und (x_2, y_2) der Liste.
    (dh. Listenelemente 1 und 2, Listenelemente 3 und 4, usw.)
  - Aufgrund des Preprocessing gilt für die Scan-Line y:
    $\mathbf{y = y_1 = y_2}$ und $x_1 <= x_2$
  - Zeichne alle Pixel auf der Scan-Line y, für die gilt:
    $x_1 <= x < x_2$ mit ganzzahligem x

4.4 Polygonen & Füllalgorithmen

Scan-Line-Methode (Fortsetzung)

Beispiel:

- Preprocessing

(1,7), (1,7)

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

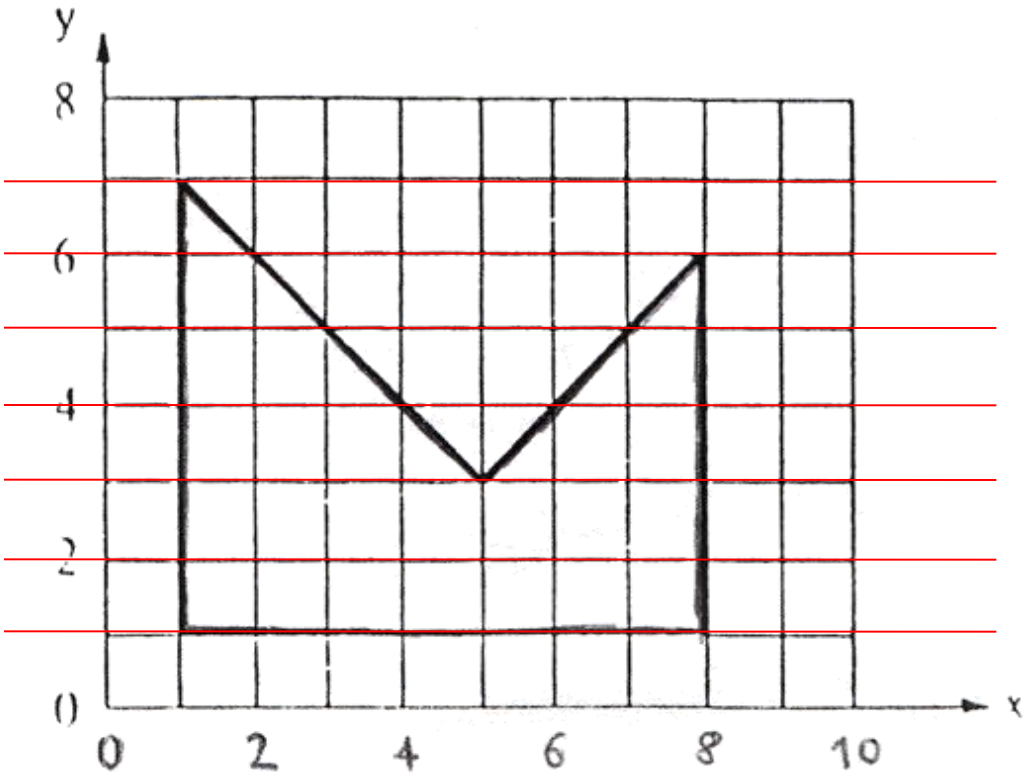
(1,5), (3,5), (7,5), (8,5)

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

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

(1,2), (8,2)

(1,1), (8,1)



(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)

4.4 Polygonen & Füllalgorithmen

Scan-Line-Methode (Fortsetzung)

Beispiel:

- Scan-Conversion

(1,7),(1,7)

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

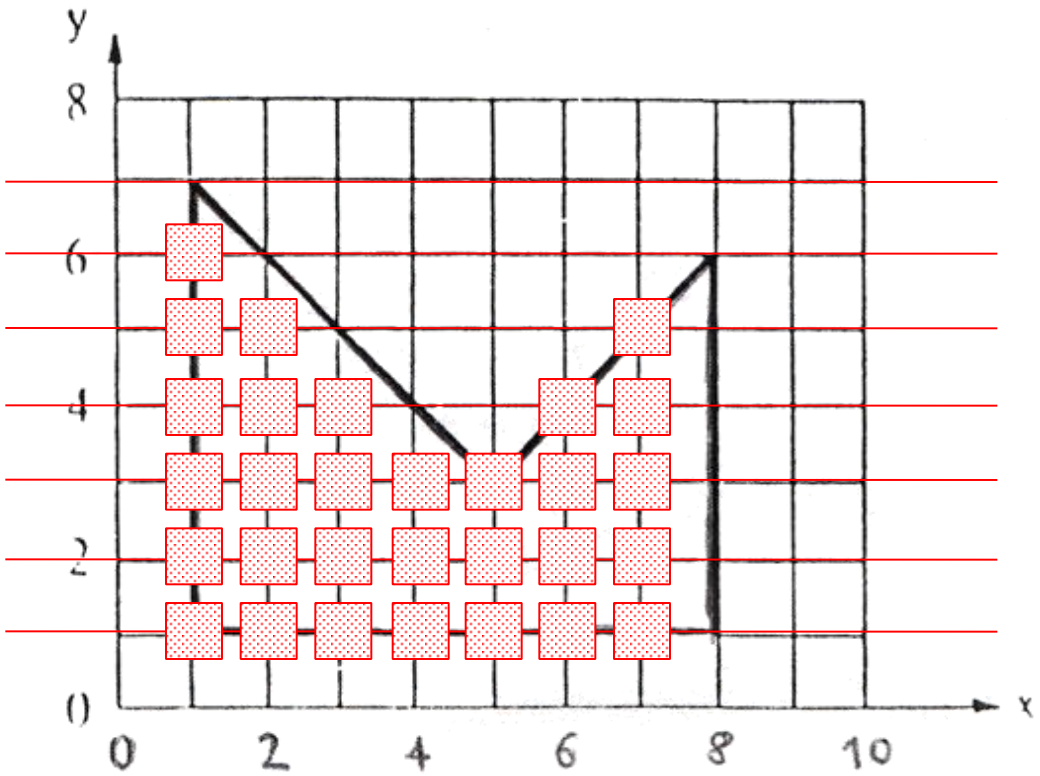
(1,5),(3,5); (7,5),(8,5)

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

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

(1,2),(8,2)

(1,1),(8,1)



Bem.: Ein exakt rechts anschließendes Polygon würde die **Pixel in Spalte 8 färben**!

4.4 Polygonen & Füllalgorithmen

Saatpunkt-Methode / Seed-Fill

- Seed-Fill-Methoden füllen das Gebiet ausgehend von einem **Ausgangspixel** (Saatpunkt, Seed) und
- Eignet sich für **pixelweise definierte Gebiete**, also für Rastergeräte.
- Man unterscheidet nach der Art der Gebietsdefinition:

(i) **Boundary-Fill-Algorithmus** für randdefinierte Gebiete

Input: Startpunkt (Saatpunkt), Farbe der Begrenzungskurve, Füllfarbe oder Muster

Algo.: Wiederhole

Vom **Startpixel ausgehend** werden **rekursiv** Nachbarpixel umgefärbt,

bis Pixel mit der **Farbe der Begrenzungskurve** oder **bereits umgefärbte Pixel** erreicht werden

4.4 Polygonen & Füllalgorithmen

Saatpunkt-Methode / Seed-Fill (Fortsetzung)

(ii) **Flood/Interior-Fill-Algorithmus** für inhaltsdefinierte Gebiete

Input: Startpunkt (Saatkorn), Farbe der umzufärbenden Pixel, Füllfarbe oder Muster

Algo.: Wiederhole

Vom **Startpixel ausgehend** werden rekursiv Nachbarpixel gleicher Farbe umgefärbt,

bis **Pixel mit abweichender Farbe** erreicht werden

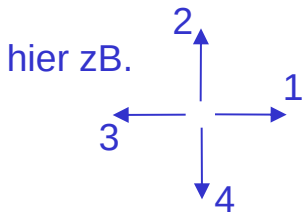
4.4 Polygonen & Füllalgorithmen

Saatpunkt-Methode / Seed-Fill (Fortsetzung)

- Einfacher Saatkorn-Algorithmus
  (4 Bewegungsrichtungen, randdefiniertes Gebiet, FILO/LIFO-Prinzip)

```
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);
  }
}
```

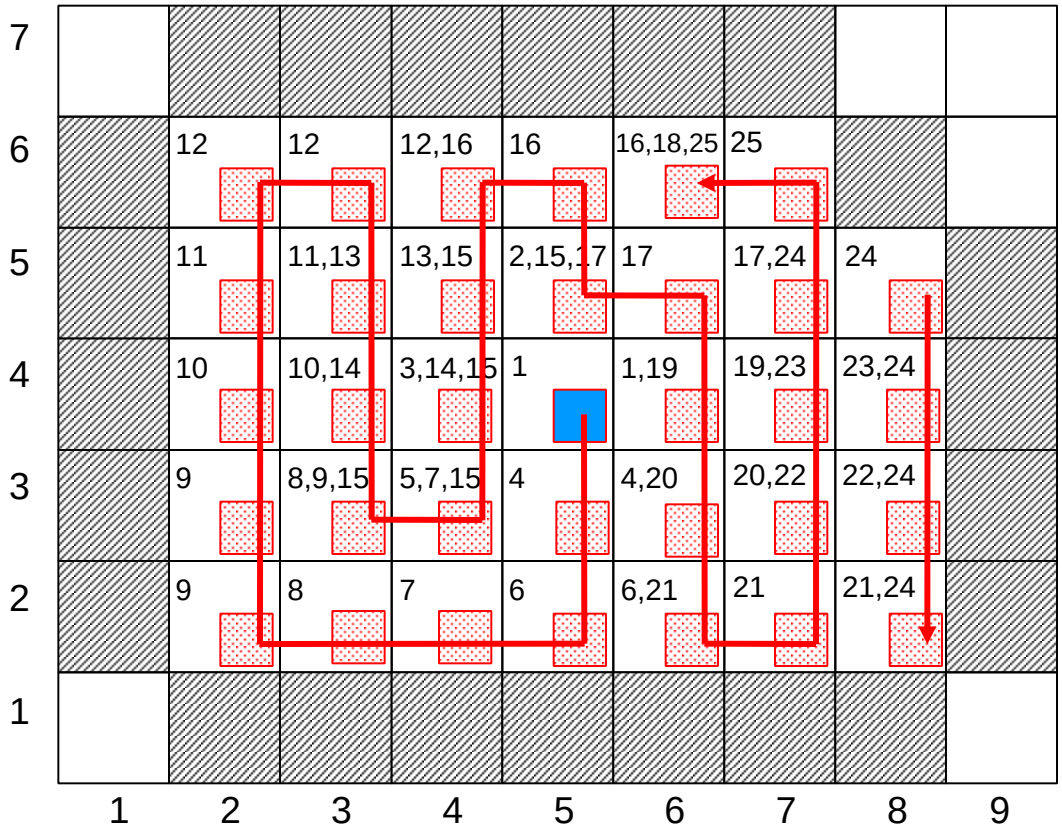


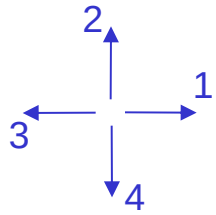
Bemerkung: Eventuell werden Pixel mehrfach im Stack abgelegt (und gefärbt)!

4.4 Polygonen & Füllalgorithmen

Saatpunkt-Methode / Seed-Fill (Fortsetzung)

- Beispiel: (die Zahlen geben die Position der Pixel im Stack an)

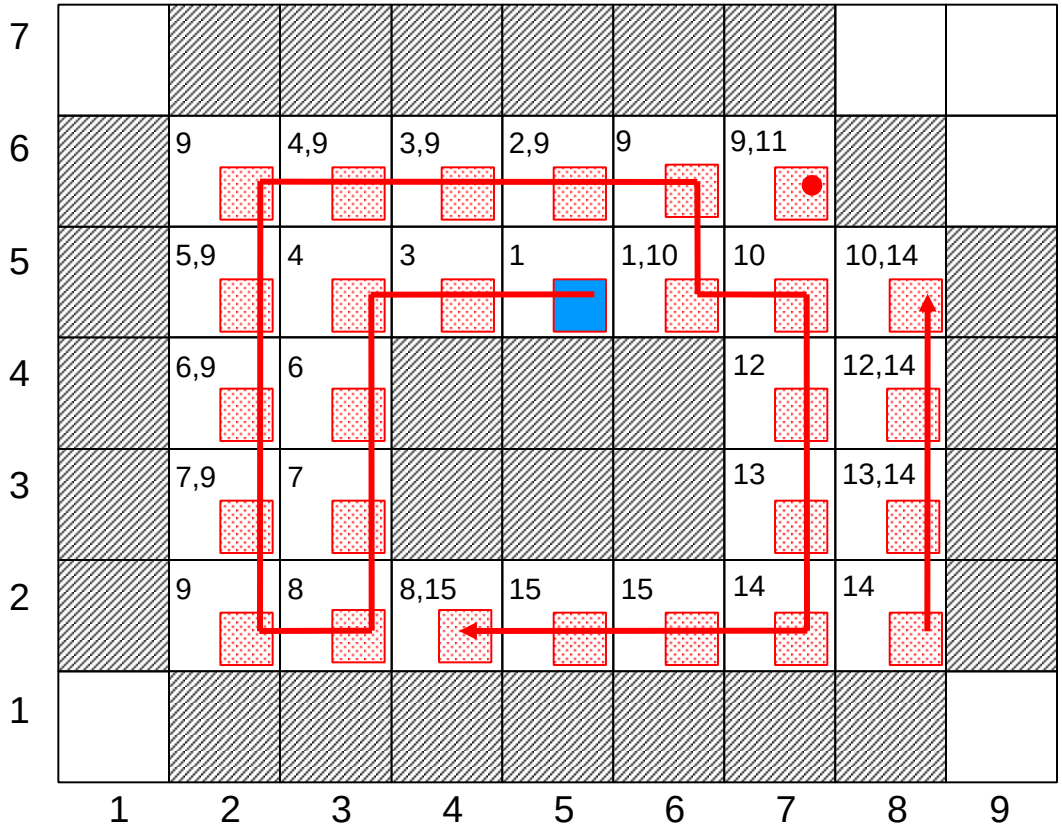


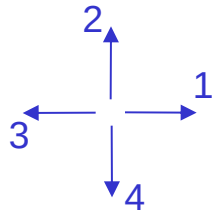


4.4 Polygonen & Füllalgorithmen

Saatpunkt-Methode / Seed-Fill (Fortsetzung)

- Beispiel: Gebiet mit Loch





4.4 Polygonen & Füllalgorithmen

Hybrid-Methoden

- Hybrid-Methoden verwenden die Ideen der **Scan-Line**- und **Saatpunkt**-Methoden gemeinsam.

=> Scan-Line-Seed-Fill-Algorithmus

4.5 Rasterartefakte & Aliasing

Aliasing (Begriff aus der 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)

- Der Begriff **Aliasing** in der Computergraphik
  - Visuelle Artefakte durch **Unterabtastung**, (z.B. Aliasing bei einem Schachbrettmuster)
  - Visuelle Artefakte durch **Rasterkonvertierungseffekte** zurückgehen (z.B. Treppeneffekte bei schrägen Linien).
- **Örtliches und zeitliches Aliasing** (z.B. scheinbar rückwärts drehende Wagenräder im Western).

4.5 Rasterartefakte & Aliasing

Aliasing (Fortsetzung)

- **Anti-Aliasing-Verfahren:** Methoden um Aliasing-Effekten entgegenzuwirken (z.B. Überabtastung, Filterung),
- Ein echtes „Beseitigen“ ist oft (schon theoretisch) **nicht möglich**.
  (z.B. wenn die Signale einfach **nicht bandbegrenzt** sind; hier hilft zwar eine höhere Abtastfrequenz - also **Überabtastung** - aber es beseitigt nicht die Probleme)
- Bei Artefakten der **Rasterkonvertierung** spricht man bei Anti-Aliasing-Verfahren auch von „Verfahren zur (Bild-)**Kantenglättung**“.

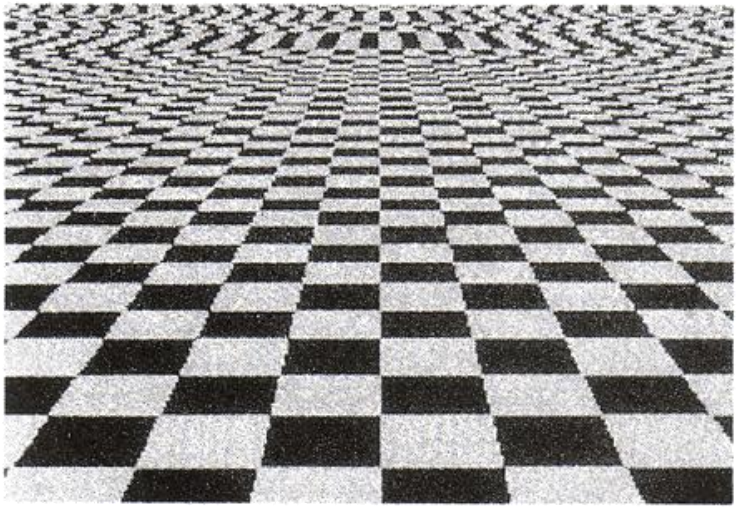
4.5 Rasterartefakte & Aliasing

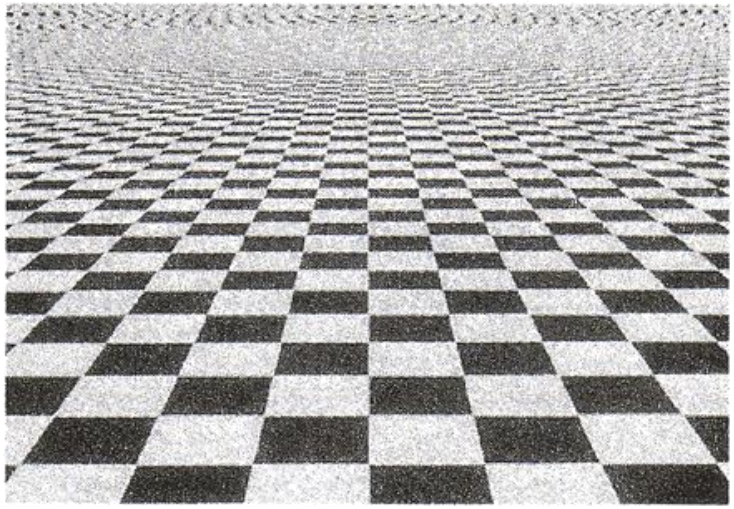
Aliasing (Fortsetzung)

- 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**

4.5 Rasterartefakte & Aliasing

- Beispiel: Textur-Artefakte, unendliches Schachbrettmuster

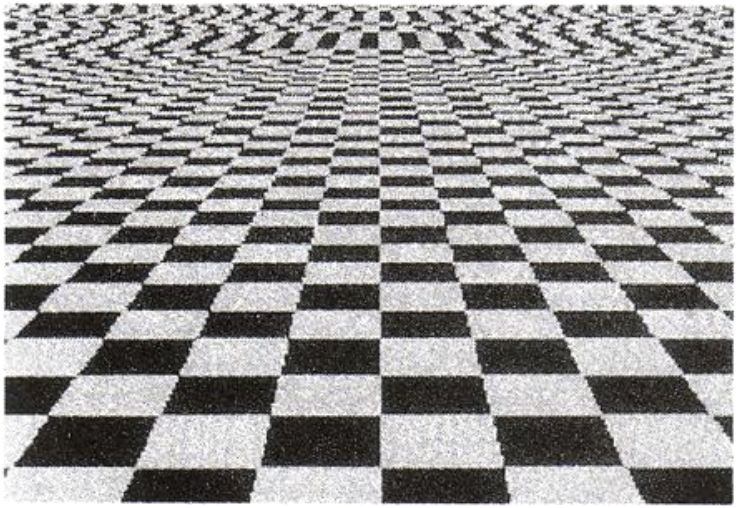


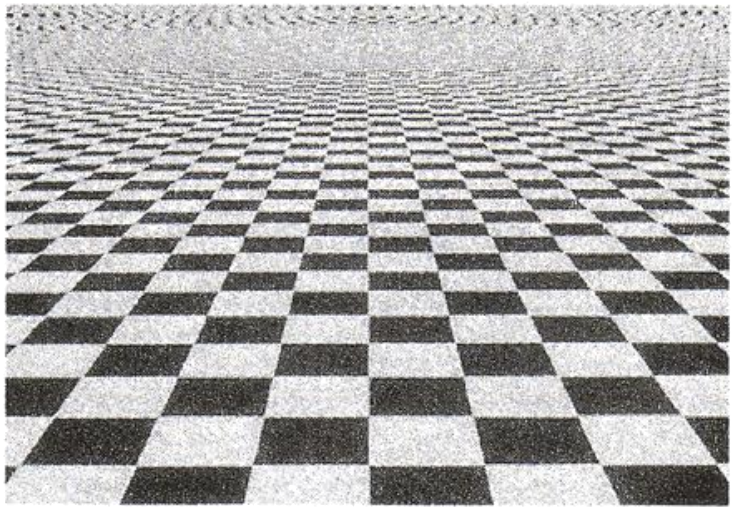


- Üblicherweise treten visuelle Artefakte auf, wenn die **Periodizität** (hier die Kachelmuster) in der Textur die **Größenordnung von Pixel** erreicht.
- Linkes Bild: Am oberen Ende werden die Quadrate immer kleiner, plötzlich aber wieder größer (Aliasing). Dies ist ein Ergebnis zu grober Abtastung

4.5 Rasterartefakte & Aliasing

- Beispiel (Fortsetzung):

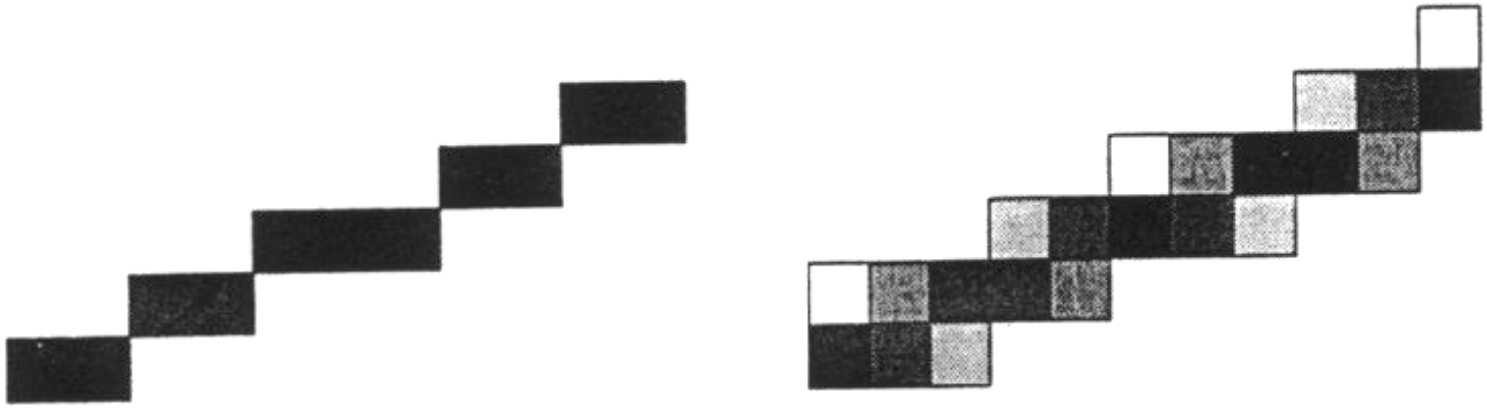




- Rechtes Bild: Mittels **zweifacher Überabtastung** (Abtastung mit doppelter Frequenz, dh. vierfacher Rechenaufwand) können die **Artefakte verringert** werden
  Sie treten aber immer noch auf (bei höheren Frequenzen)
- Es gilt: „Echtes“ Aliasing kann in Computergraphikbildern mittels Überabtastung nicht entfernt, sondern nur verbessert werden (**nicht bandbegrenzt**)

4.5 Rasterartefakte & Aliasing

- Beispiel: Treppeneffekte, Jagged Edges, Jaggies

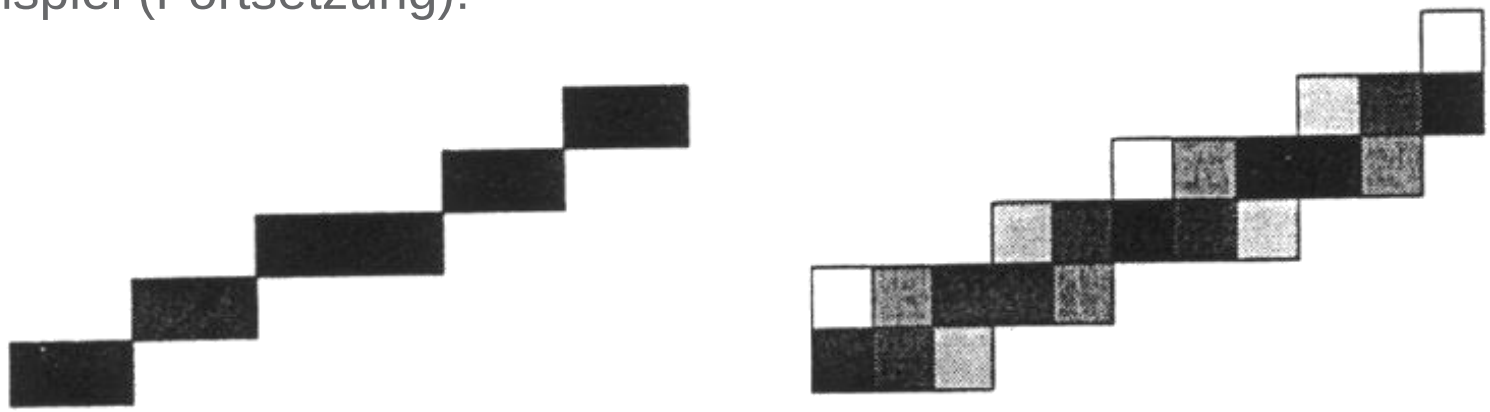


Gegenüberstellung von ungeglättetem und geglättetem Vektor.

- Die bisher beschriebenen Verfahren zur Rasterung von Geraden und Kurven **erzeugen Treppeneffekte** (linkes Bild)
- Zeichnen von Punkten nur **an Rasterpositionen** möglich, Entsprechen i. a. **nicht den tatsächlichen** Positionen (Sollpositionen)

4.5 Rasterartefakte & Aliasing

- Beispiel (Fortsetzung):

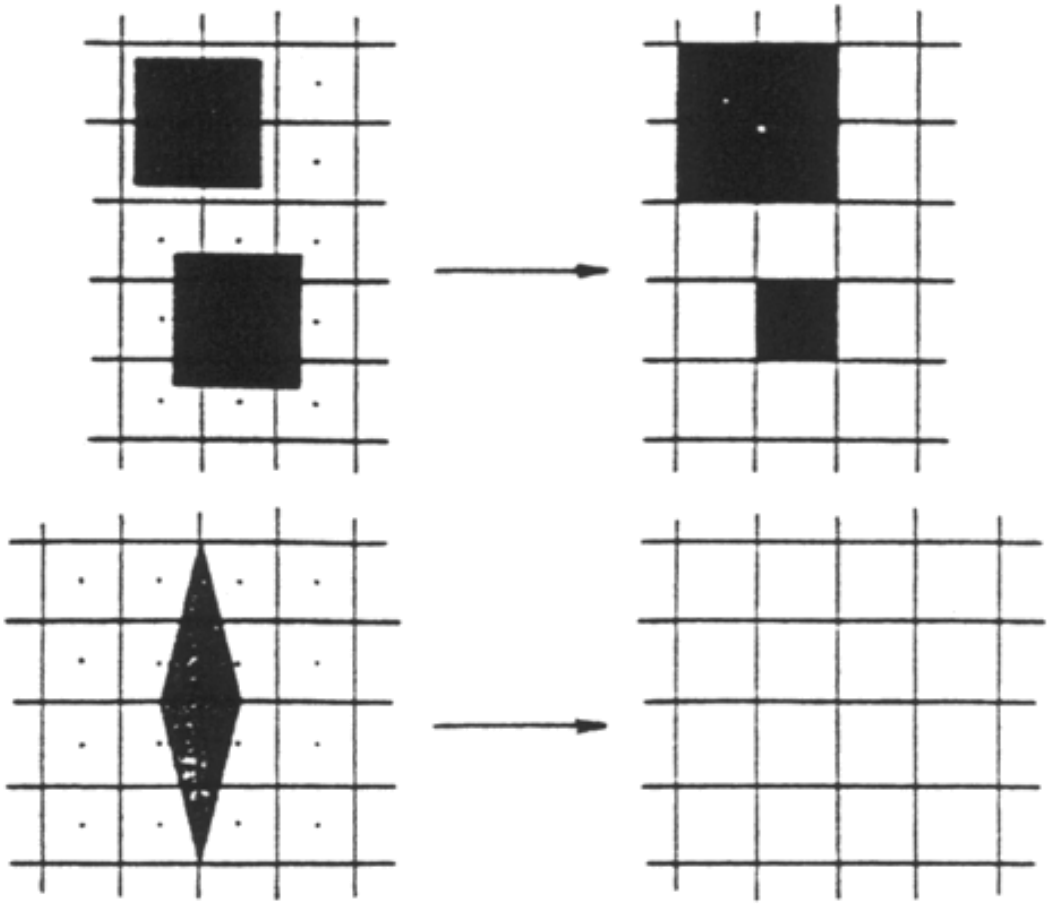


Gegenüberstellung von ungeglättetem und geglättetem Vektor.

- 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 (rechtes Bild).

4.5 Rasterartefakte & Aliasing

- Beispiel: Aliasing bei Polygonen



4.5 Rasterartefakte & Aliasing

Anti-Aliasing

- Ein einfaches globales (d. h. das gesamte Bild betreffende) Anti-Aliasing-Verfahren stellt die **Überabtastung**, auch als Oversampling oder Supersampling bezeichnet, dar.
- 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.
- Folgende Filterkerne (Crow, 1981) werden i. d. R. verwendet:

4.5 Rasterartefakte & Aliasing

Anti-Aliasing (Fortsetzung)

3 x 3

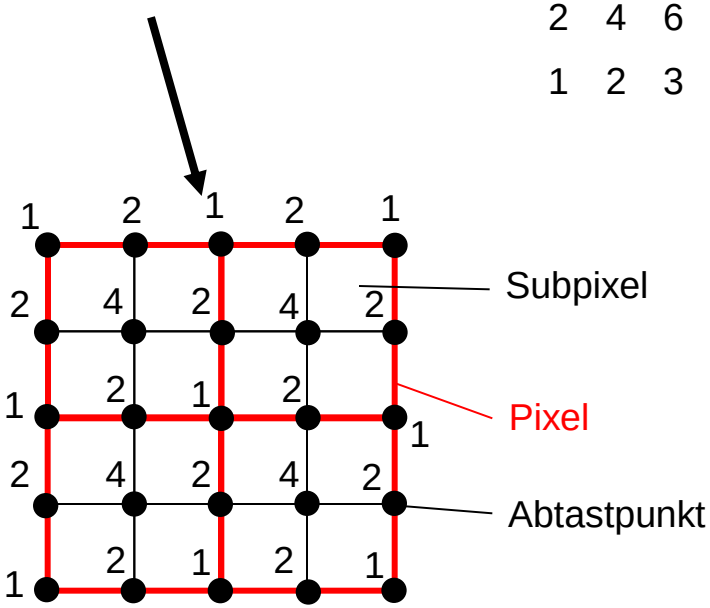
1	2	1
2	4	2
1	2	1

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

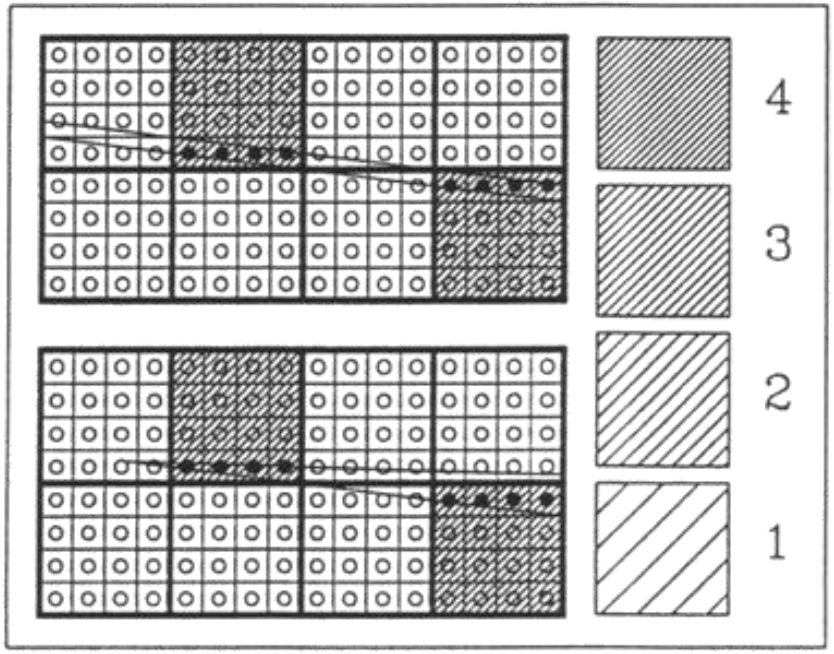
1	2	3	4	3	2	1
2	4	6	8	6	4	2
3	6	9	12	9	6	3
4	8	12	16	12	8	4
3	6	9	12	9	6	3
2	4	6	8	6	4	2
1	2	3	4	3	2	1



4.5 Rasterartefakte & Aliasing

Anti-Aliasing (Fortsetzung)

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



Linie verschwindet stellenweise, da keine Subpixel getroffen werden!

dito für Dreieck!

Bemerkung: Gesetzte Subpixel sind schwarz. Die Schraffur zeigt den Grauwert abhängig von der Anzahl der gesetzten Subpixel.

4.5 Rasterartefakte & Aliasing

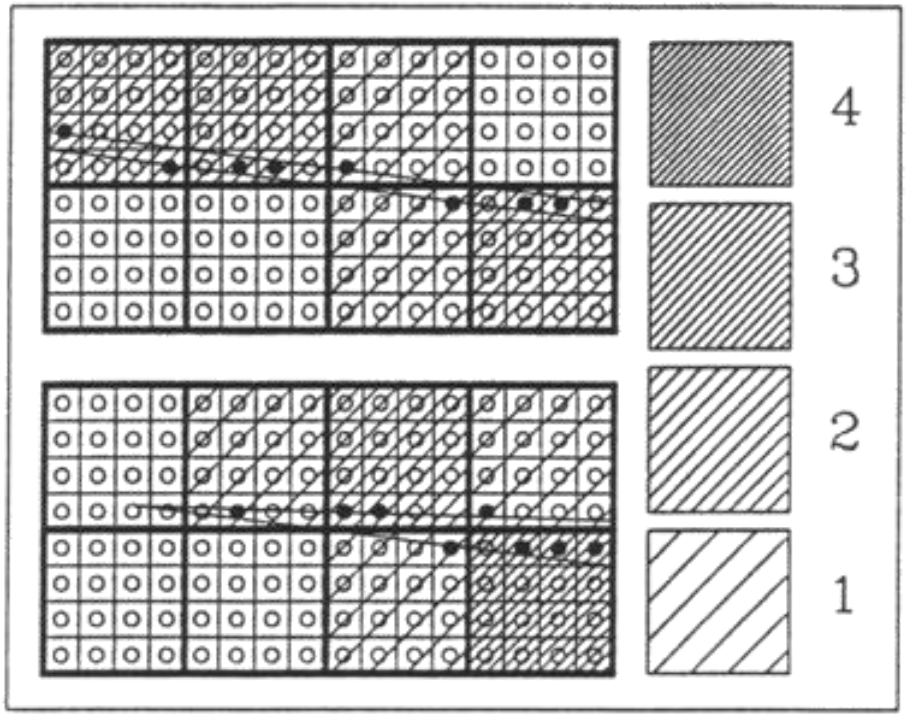
Anti-Aliasing (Fortsetzung)

- 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.)

4.5 Rasterartefakte & Aliasing

Anti-Aliasing (Fortsetzung)

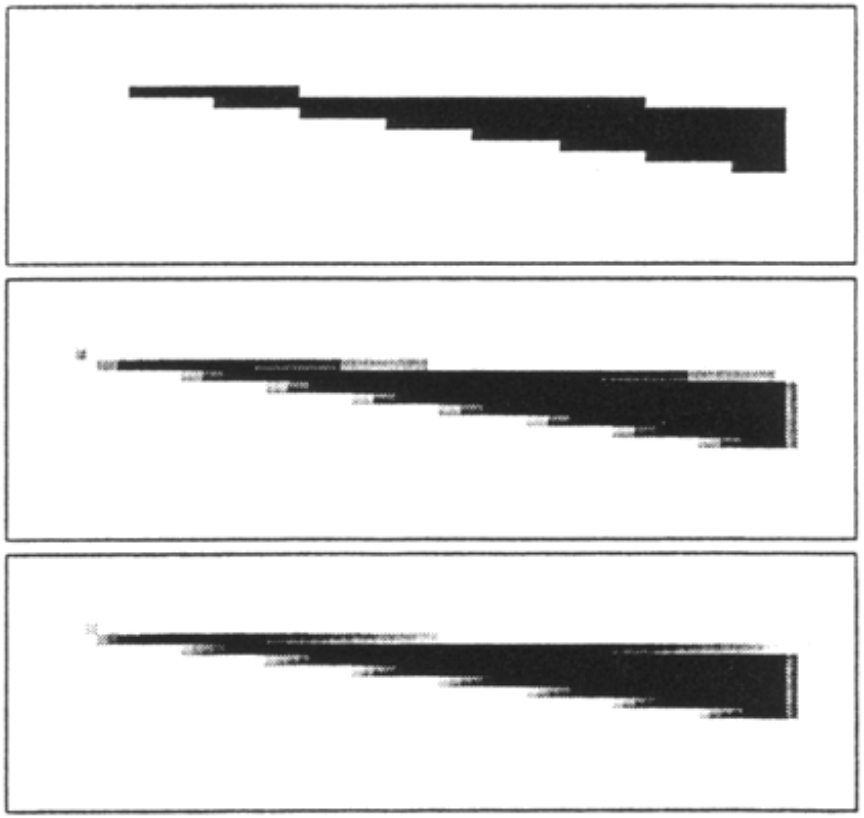
- Die folgenden Abbildungen zeigen die resultierenden Ergebnisse in den zuvor gezeigten Problemfällen Linie und spitzes Dreieck:



4.5 Rasterartefakte & Aliasing

Anti-Aliasing (Fortsetzung)

- Leistungsfähigkeit der vorgestellten Verfahren:



Spitzes Dreieck ohne Glättung

Spitzes Dreieck
mit Oversampling

Spitzes Dreieck
mit korrekter Berechnung
der überdeckten Fläche

4.5 Rasterartefakte & Aliasing

Anti-Aliasing (Fortsetzung)

- Bemerkung: 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)