

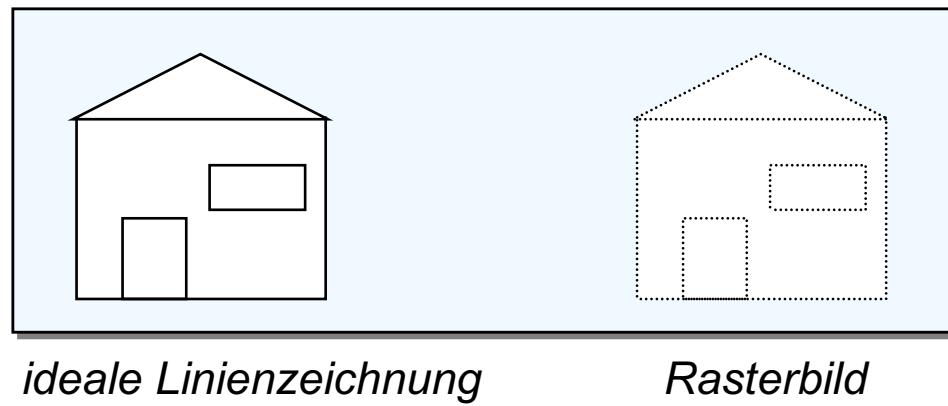


3. Rendering

3.5 Rasterisierung und Antialiasing



- Unter **Rasterisierung (Scankonvertierung)** versteht man die Zerlegung von graphischen Primitiven in Pixel des Bildspeichers.





- Rasterisierung

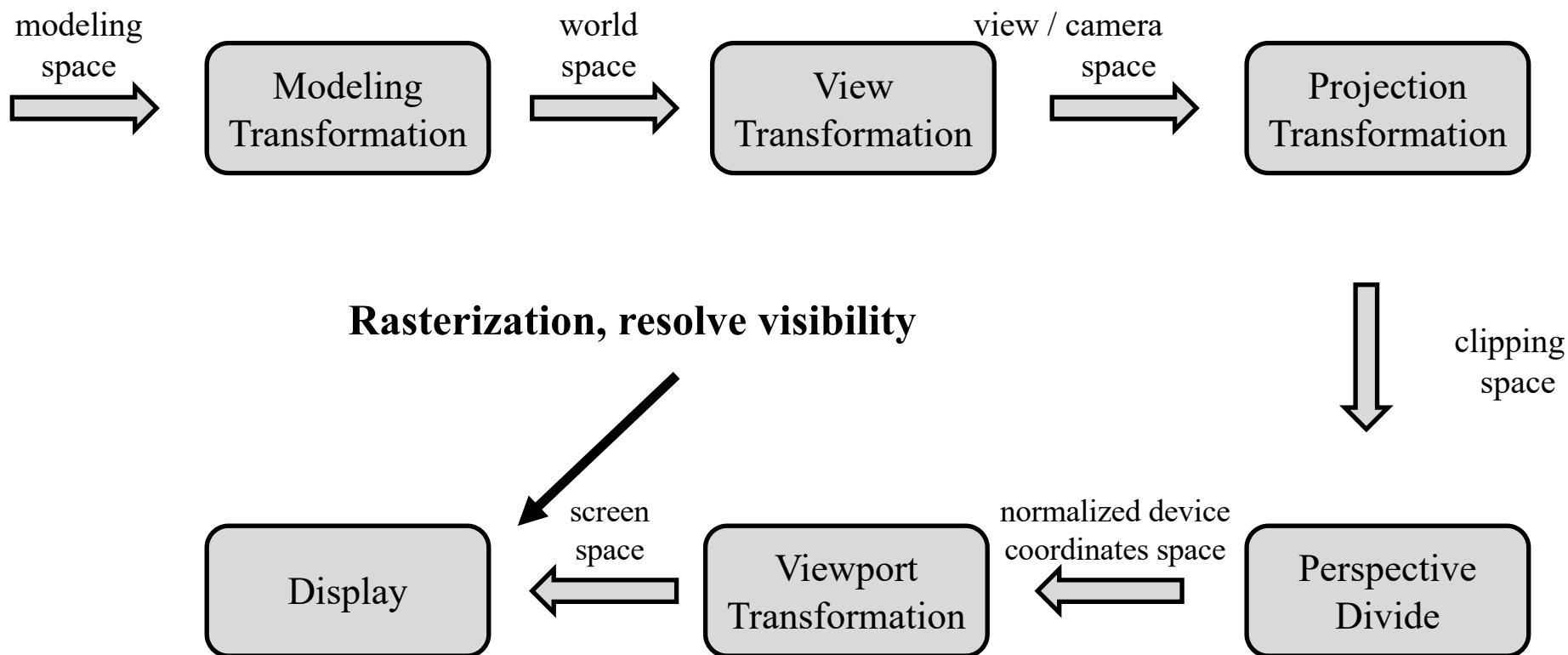


3. Rendering

3.5 Rasterisierung und Antialiasing



Übliche Koordinatensysteme der 3D-Computergrafik

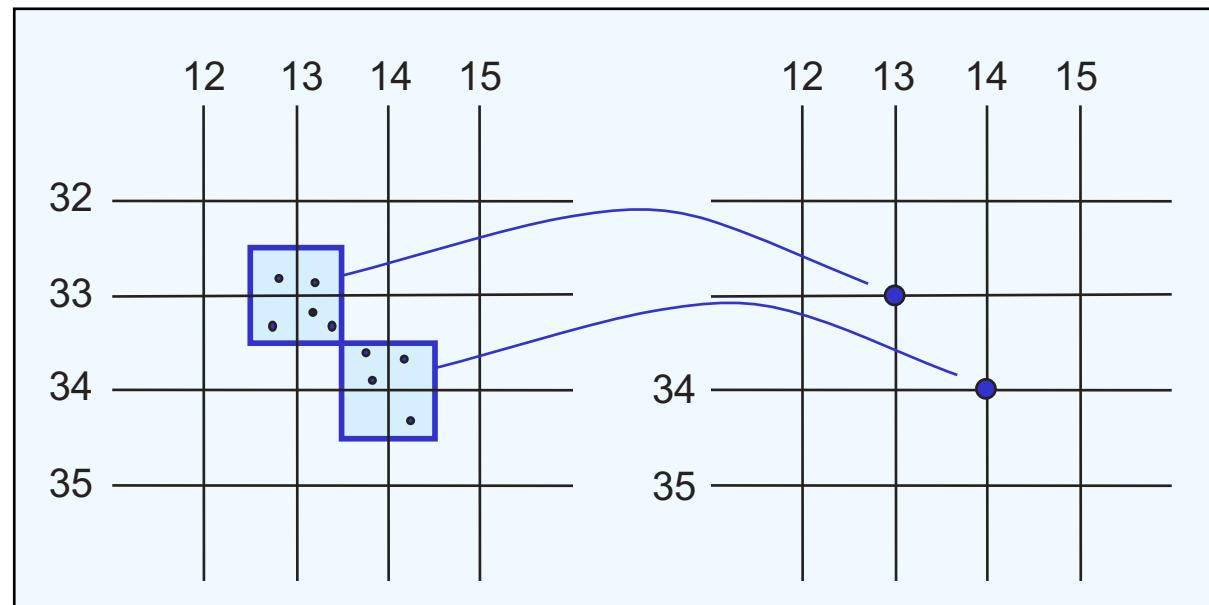


3. Rendering

3.5 Rasterisierung und Antialiasing



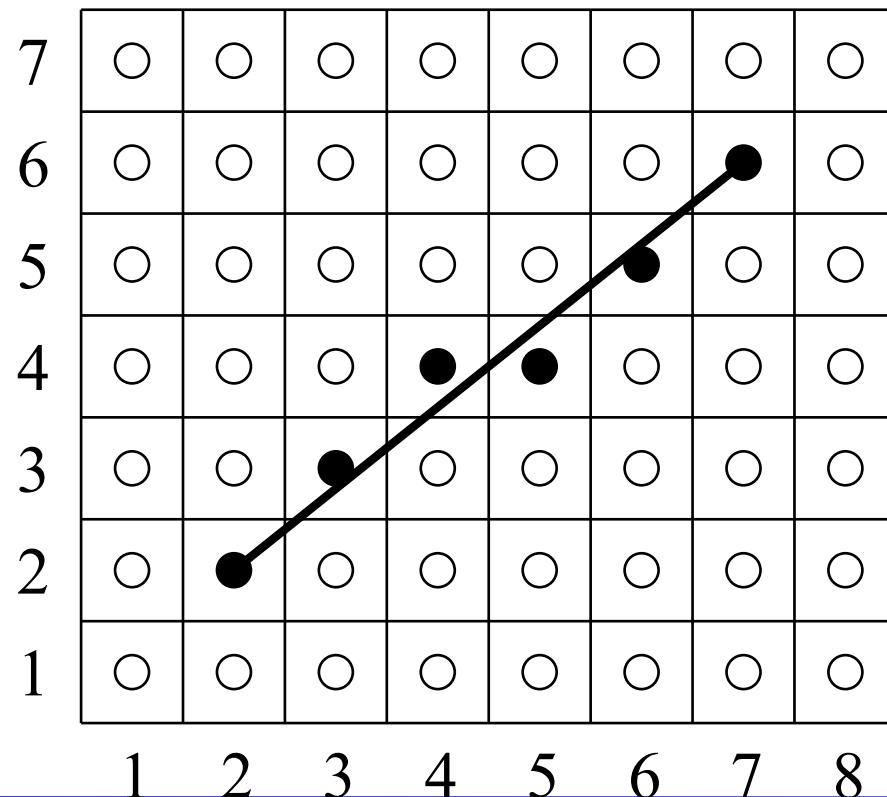
- Die Rasterisierung von Punkten beruht auf der Suche der nächsten Nachbarschaft von Pixelzentren.





- Für die **Rasterisierung von Strecken** wurden spezielle effektive Algorithmen entwickelt.

wichtigster Vertreter: der **Bresenham-Algorithmus**





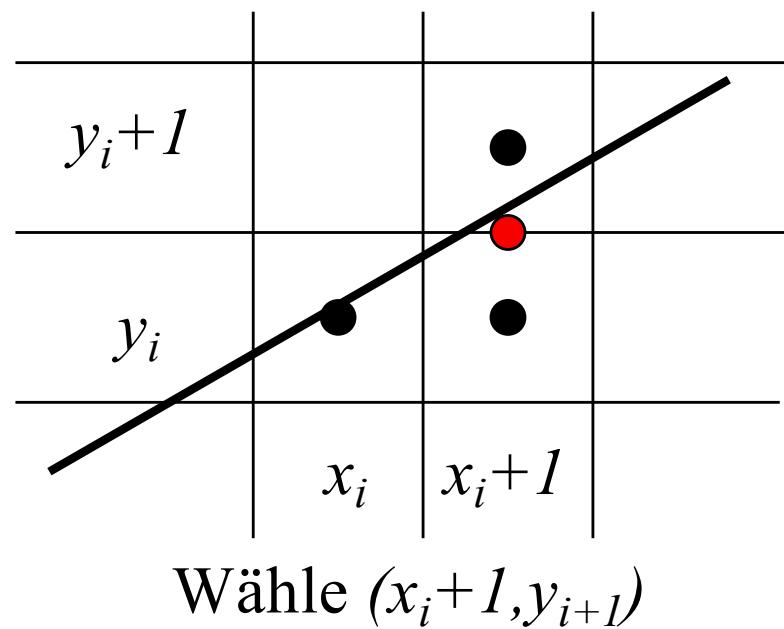
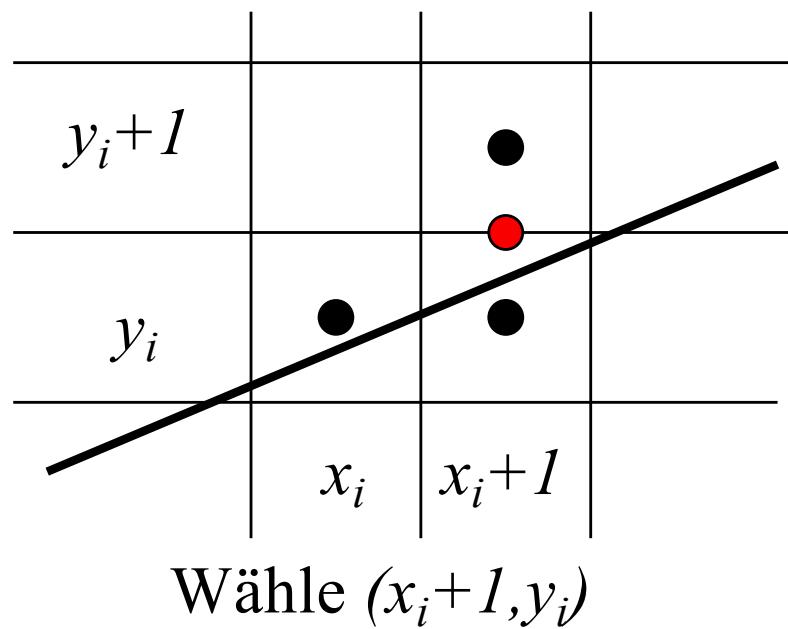
- **Bresenham-Algorithmus: Überblick**

- **Ziel:** zeichne für jedes x das Pixel, dessen y -Wert der Linie am nächsten ist, Annahme: positiver Anstieg ≤ 1
 - Gegeben: (x_i, y_i) , Wahl zwischen (x_i+1, y_i+1) oder (x_i+1, y_i)
- **Idee:** Berechnung einer *Entscheidungsvariablen*
 - Der Wert legt fest, welches Pixel zu zeichnen ist
 - Von einem Pixel zum nächsten einfach zu aktualisieren
- **Bresenham-Algorithmus:** *Midpoint-Algorithmus* für Linien
 - Andere Midpoint-Algorithmen für Kegelschnitte (Kreise, Ellipsen)



- **Midpoint-Methode**

- Berücksichtigung des Midpoints zwischen (x_i+1, y_i+1) und (x_i+1, y_i)
- Ist dieser oberhalb der Linie, wählen wir (x_i+1, y_i) , andernfalls wählen wir (x_i+1, y_i+1)





■ **Midpoint-Entscheidungsvariable**

- Schreiben der Linie in *impliziter Form*:
 - $\Delta x = x_2 - x_1$, $\Delta y = y_2 - y_1$

$$F(x, y) = ax + by + c = \Delta y \cdot x - \Delta x \cdot y + (\Delta x \cdot y_1 - \Delta y \cdot x_1)$$

- Der Wert von $F(x, y)$ gibt an, an welcher Stelle der Linie sich die Pixel befinden
 - $F(x, y) = 0$: der Punkt befindet sich auf der Linie
 - $F(x, y) < 0$: der Punkt befindet sich oberhalb der Linie
 - $F(x, y) > 0$: der Punkt befindet sich unterhalb der Linie
- Die Entscheidungsvariable ist der Wert von
$$d_i = 2F(x_i + 1, y_i + 0.5)$$
 - Der Faktor 2 vereinfacht Berechnungen: keine Brüche



$$d_i = 2\Delta y(x_i + 1) - 2\Delta x y_i - \Delta x + 2c$$

- *Was können wir entscheiden?*
 - d_i negativ => nächster Punkt bei (x_i+1, y_i)
 - d_i positiv => nächster Punkt bei (x_i+1, y_i+1)
 - An jedem Punkt berechnen wir d_i und entscheiden, welches Pixel zu zeichnen ist
 - Wie aktualisieren wir es? Was ist d_{i+1} ?

3. Rendering

3.5 Rasterisierung und Antialiasing



- **Aktualisierung der Entscheidungsvariable**

- d_{k+1} ist der alte Wert, d_k , plus Erhöhung:

$$d_{k+1} = d_k + (d_{k+1} - d_k)$$

- Wenn wir $y_{i+1}=y_i+1$ wählen:

$$d_{k+1} = d_k + 2\Delta y - 2\Delta x$$

- Wenn wir $y_{i+1}=y_i$ wählen:

$$d_{k+1} = d_k + 2\Delta y$$

- Was ist d_1 (unter der Annahme ganzzahliger Endpunkte)?

- Hinweis: wir benötigen c nicht mehr

$$d_1 = 2\Delta y - \Delta x$$



- **Bresenham-Algorithmus**
- Bei ganzen Zahlen Anstieg zwischen 0 und 1:
 - $x=x_1, y=y_1, d=2dy - dx$, set pixel (x, y)
 - until $x=x_2$
 - $x=x+1$
 - if $d>0$, then { $y=y+1$; set pixel (x, y) ; $d=d+2\Delta y - 2\Delta x$; }
 - if $d<0$, then { $y=y$; set pixel (x, y) ; $d=d+2\Delta y$; }
- Zu Beginn Berechnung der Konstanten ($2\Delta y - 2\Delta x$ und $2\Delta y$)
 - Innerer loop führt nur Additionen und Vergleiche aus
- Es darf keine Rolle spielen, in welcher Reihenfolge die Endpunkte festgelegt werden (einheitliche Entscheidung treffen, wenn $d==0$)

3. Rendering

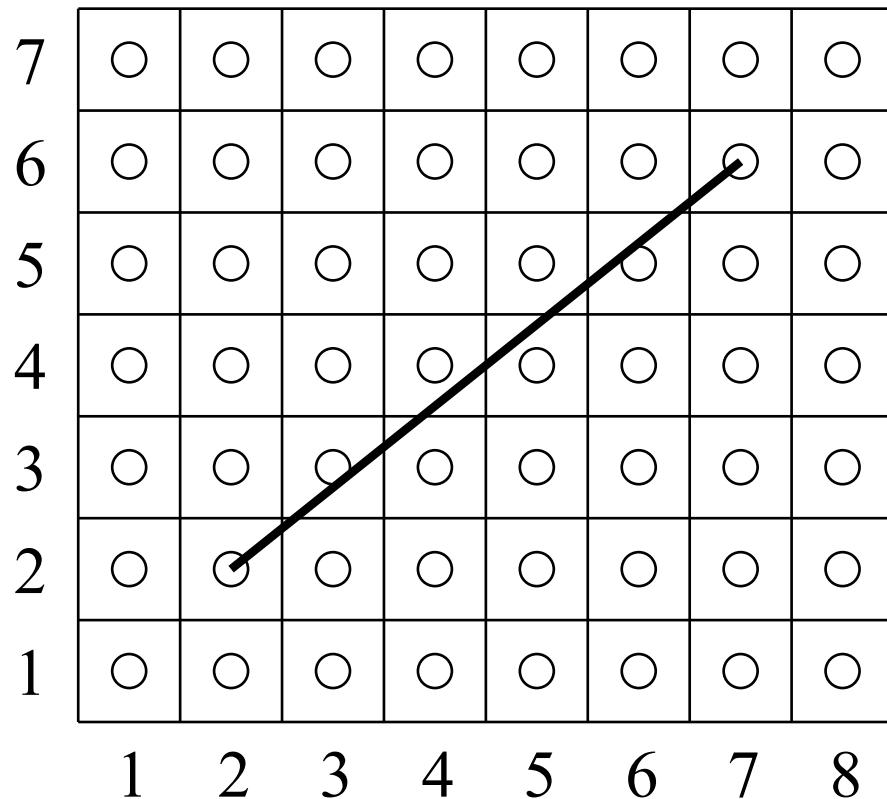
3.5 Rasterisierung und Antialiasing



$$x=x_1, y=y_1, d_1=2dy - dx$$

if $d > 0$, then { $y = y + 1$; set (x, y) ; $d = d + 2\Delta y - 2\Delta x$; }

- Beispiel: (2,2) bis (7,6)
- if $d < 0$, then { $y = y$; set (x, y) ; $d = d + 2\Delta y$; }



$$\Delta x = 5, \Delta y = 4$$

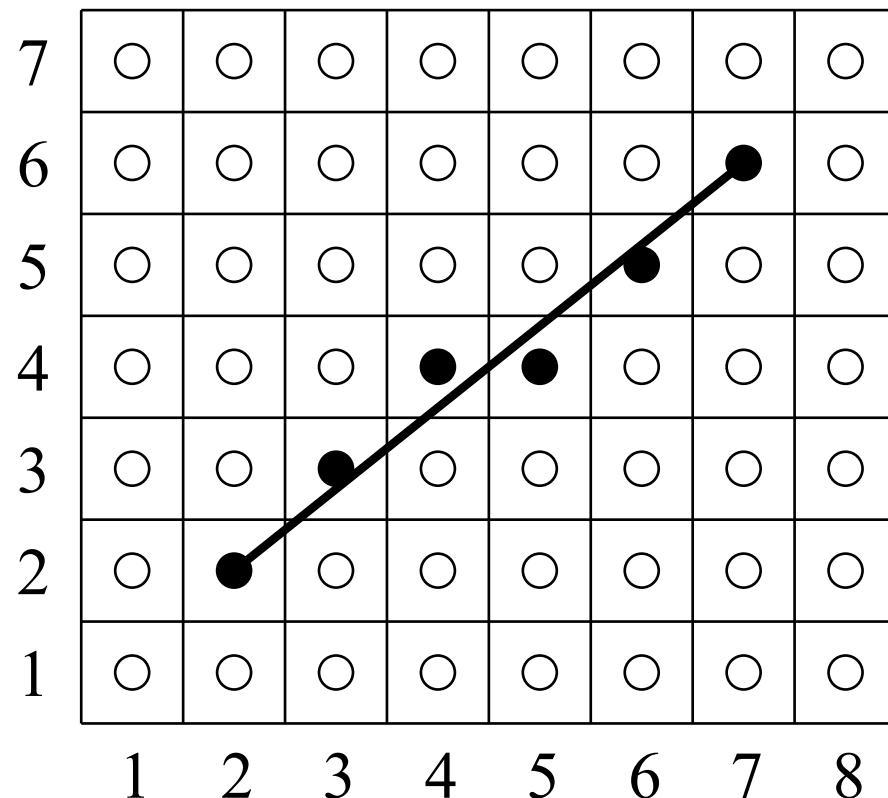
| x | y | d |
|---|---|---|
| | | |

3. Rendering

3.5 Rasterisierung und Antialiasing



- Beispiel: (2,2) bis (7,6)



$\Delta x=5, \Delta y=4$

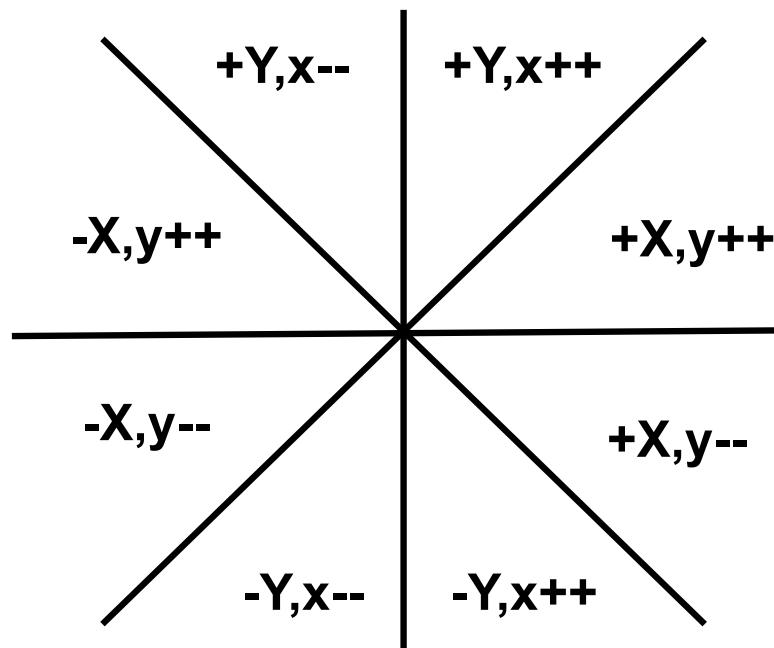
| x | y | d |
|---|---|----|
| 2 | 2 | 3 |
| 3 | 3 | 1 |
| 4 | 4 | -1 |
| 5 | 4 | 7 |
| 6 | 5 | 5 |
| 7 | 6 | 3 |

3. Rendering

3.5 Rasterisierung und Antialiasing



- **Linien: beliebige Richtungen**
- 8 verschiedene Fälle





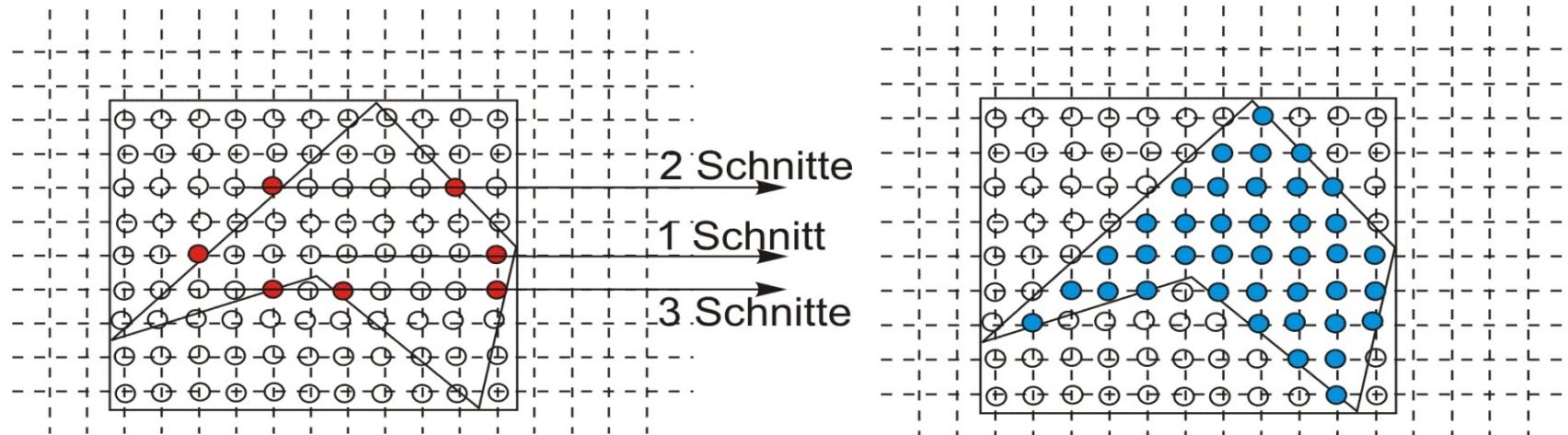
- **Bresenham allgemein:**
- Midpoint-Methode funktioniert gut bei Kreisen, Ellipsen und weiteren implizit definierbaren Kurven
 - Parabeln, Hyperbeln, ...

3. Rendering

3.5 Rasterisierung und Antialiasing



- Eine Prinzipielle Lösung zur Rasterisierung von Polygonen könnte wie folgt aussehen:
 - Bestimmung des umschließenden Rechtecks für das zu konvertierende Polygon.
 - Überprüfung aller innerhalb dieses Rechtecks liegenden Pixel, ob sie innerhalb oder außerhalb des Polygons liegen:



3. Rendering

3.5 Rasterisierung und Antialiasing



- Der Test, ob der Punkt $P(x, y)$ innerhalb des Polygons liegt, lässt sich anhand der Schnittpunkte eines von $P(x, y)$ ausgehenden Strahls mit den Polygonkanten entscheiden.
- Wenn die Anzahl der Schnittpunkte Strahl/Polygonkanten ungerade ist, liegt $P(x, y)$ innerhalb des Polygons, sonst außerhalb. (Tangentielle Berührungen werden nicht gezählt).
- Dieser Test ist aufwendig.
- Es gibt effektivere Algorithmen. Hierzu gehört der **Polygonbasierende Füllalgorithmus**.

3. Rendering

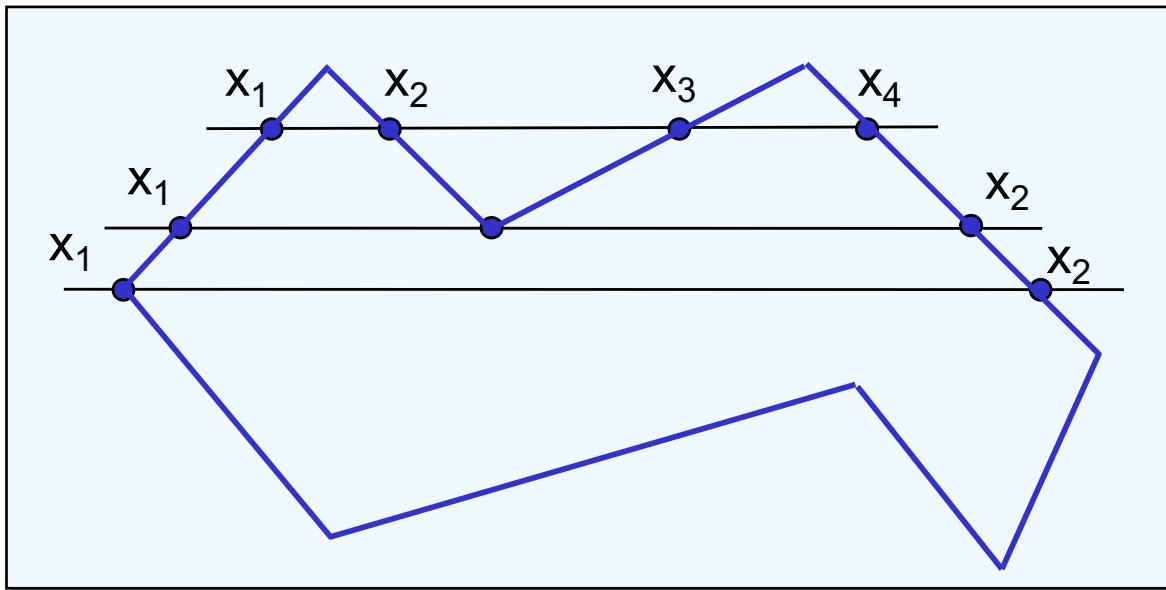
3.5 Rasterisierung und Antialiasing



- **Preprozeß:**
 - Initialisieren der aktuellen Kantenliste
(enthält alle Kanten, die aktuelle Rasterzeile schneiden);
 - Bestimmen des umschließenden Rechtecks des Polygons.
- **Füllen:** Für jede Rasterzeile innerhalb des Rechtecks
 - UPDATE aktuelle Kantenliste;
(Wenn ein Punkt einer Kante den y -Wert der aktuellen Rasterzeile erreicht, wird die Kante in die Liste aufgenommen. Wenn der Endpunkt einer Kante der aktuellen Liste den y -Wert der aktuellen Zeile erreicht, wird diese Kante aus der Liste gestrichen.)
 - Berechnung aller Schnittpunkte mit den Kanten der aktuellen Liste;
 - Sortierung der Schnittpunkte nach $x : x_1, x_2, \dots, x_n$, Setzen der Pixel in den Bereichen (x_{2i+1}, x_{2i+2}) .

3. Rendering

3.5 Rasterisierung und Antialiasing



Rasterisierung von Polygonen



- Pixelbasierende Fülltechniken werden angewendet, wenn keine Informationen über die Kanten des Polygons vorliegen.
 - Bereichsfüllen:
 - Voraussetzung:
 - Alle Pixel eines Bereiches haben dieselbe Farbe F .
 - Alle Pixel, die diesen Bereich begrenzen, haben eine beliebige Farbe ungleich F .
 - Prinzip:

Der Füllalgorithmus ersetzt ausgehend von einem Punkt im Bereichinneren (seed) die Farbe F durch eine neue Farbe $F' \neq F$.

3. Rendering

3.5 Rasterisierung und Antialiasing



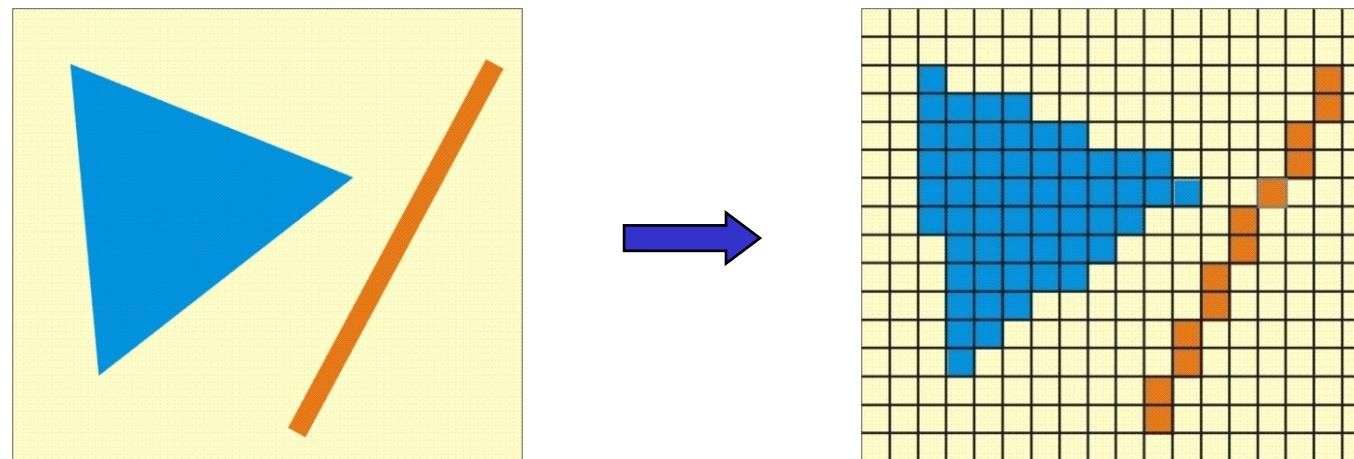
```
procedure fuell_4 (x, y, F, F': integer);
begin if LESE_PIXEL (x, y) = F then
begin
    SCHREIBE_PIXEL (x, y, F');
    {Versuch der Ausbreitung}
    fuell_4 (x, y-1, F, F': integer)
    fuell_4 (x, y+1, F, F': integer)
    fuell_4 (x+1, y, F, F': integer)
    fuell_4 (x-1, y, F, F': integer)
end {fuell_4}
```

3. Rendering

3.5 Rasterisierung und Antialiasing



- Bei der Rasterisierung werden kontinuierliche graphische Objekte in eine Anordnung diskreter Pixel überführt.
- Dabei treten Informationsverluste auf, die als **Aliasing** bezeichnet werden. (Das Wort Aliasing ist vom lateinischen Wort „alias“ abgeleitet.)
- Beispiele für Aliasing sind die bekannten Treppeneffekte bei Linien und Konturen oder das Verlorengehen feiner Details.



Beispiel für Treppeneffekte

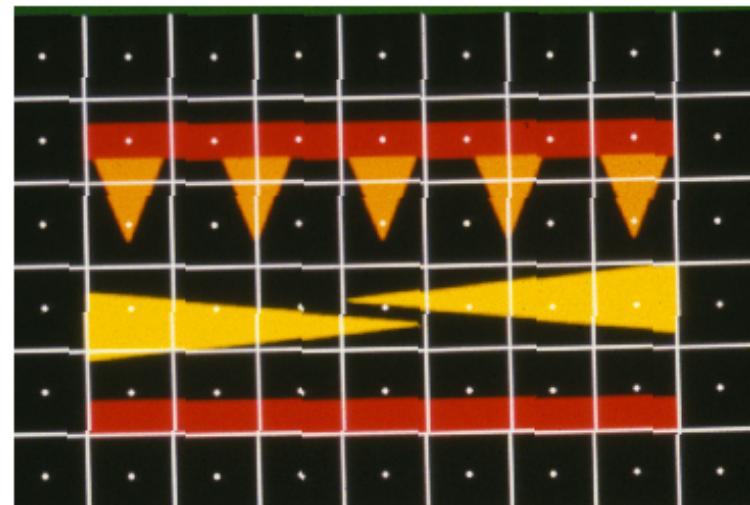
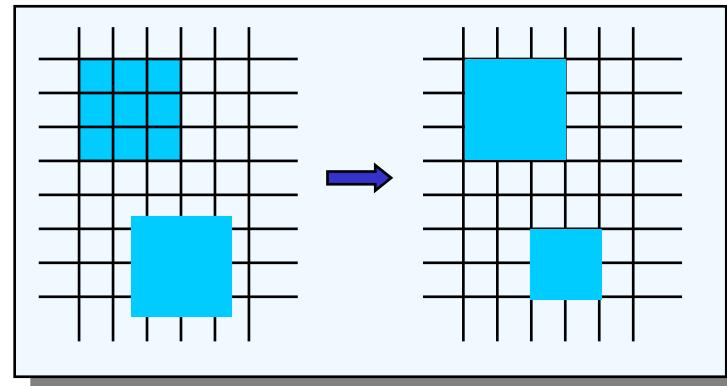
3. Rendering

3.5 Rasterisierung und Antialiasing



- Weitere Beispiele:

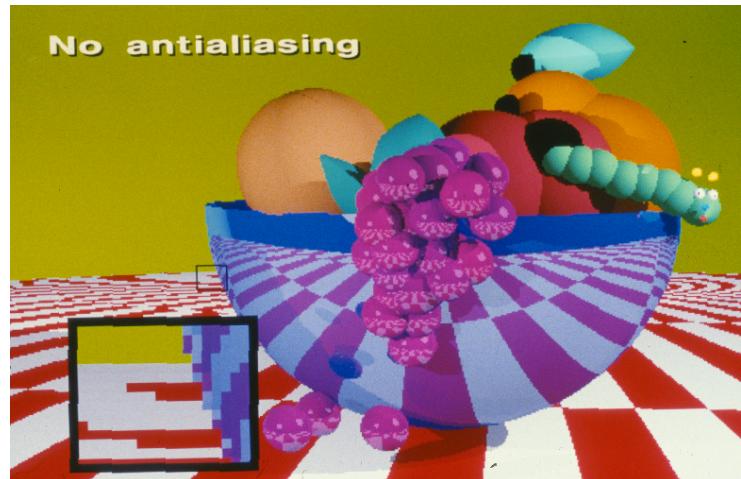
Verlust von Größeninformationen



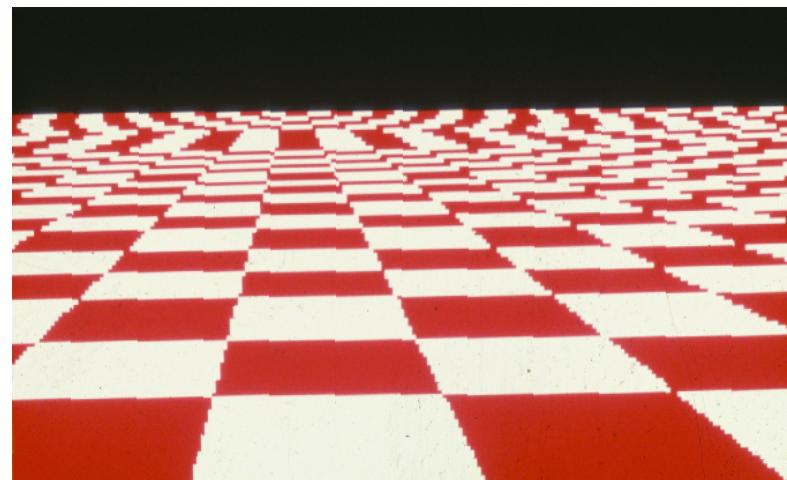
Verlust von Informationen zur Objektform

3. Rendering

3.5 Rasterisierung und Antialiasing



Verlust von Detailinformationen



3. Rendering

3.5 Rasterisierung und Antialiasing



[Chris Fay, youtube]

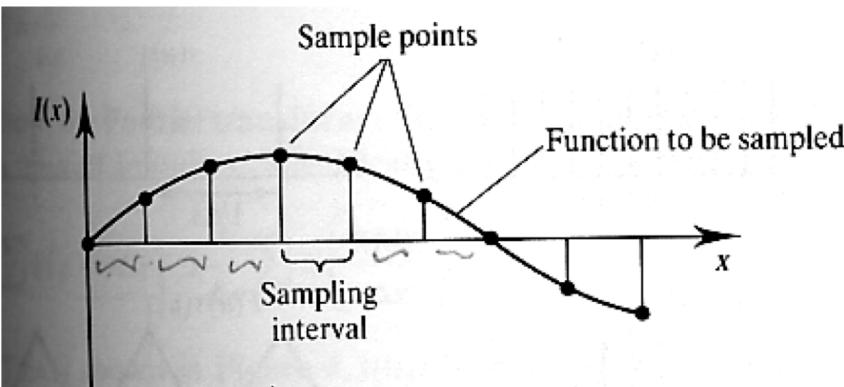
Temporal aliasing

3. Rendering

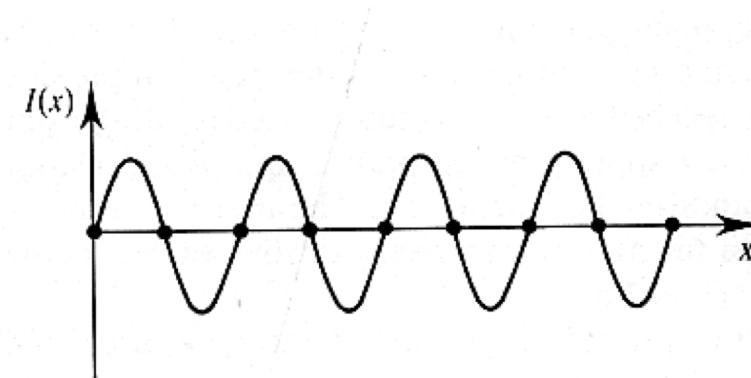
3.5 Rasterisierung und Antialiasing



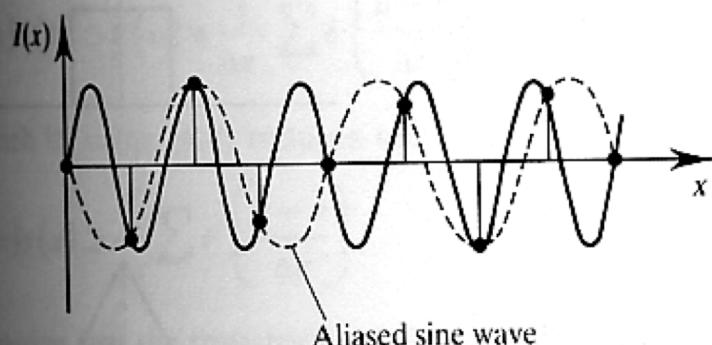
- Nyquist frequenz: hoechste moegliche Frequenz, die repreasentiert werden kann (halbe Abtastfrequenz)



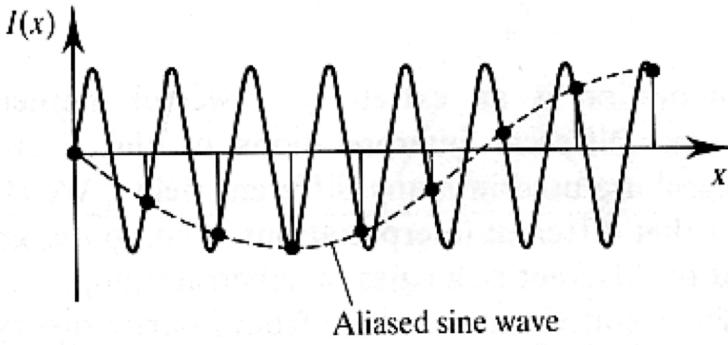
Spatial frequency < Nyquist



**Spatial frequency = Nyquist
2 samples / period**



Spatial frequency > Nyquist



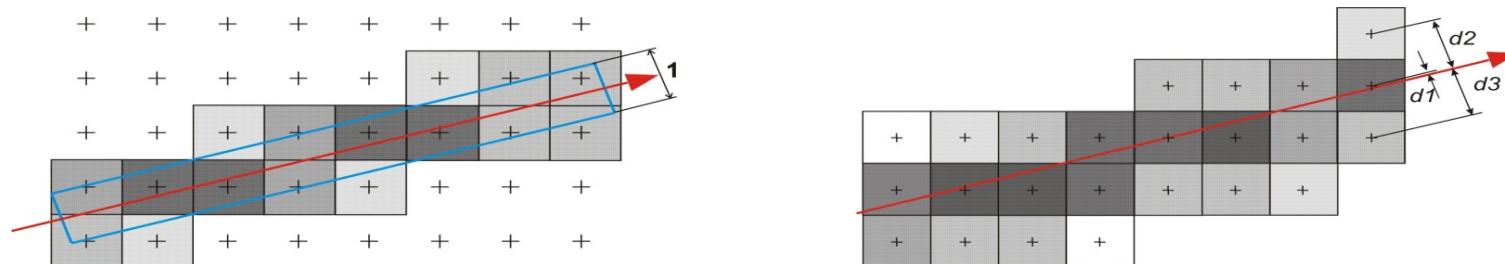
Spatial frequency >> Nyquist

3. Rendering

3.5 Rasterisierung und Antialiasing



- Mit **Antialiasing** bezeichnet man Methoden zur Vermeidung der Aliasing-Effekte in einem Bild. Hierzu gehören:
 - Erhöhung der Auflösung,
 - Bildverbesserung durch Softwaremethoden wie z.B.:
 - Konturglätten,
 - Areasampling,
 - Anwenden eines Faltungsoperators,
 - Supersampling.



*Konturglätten durch Darstellen einer Linie mit einer Linienbreite größer 1
und Variation von Pixelhelligkeiten: in Abhängigkeit der
Pixelüberdeckung (links) bzw. des Abstandes zur Linie (rechts)*

3. Rendering

3.5 Rasterisierung und Antialiasing



*Liniendarstellung ohne
Konturglätten*



*Liniendarstellung mit
Konturglätten*



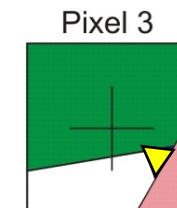
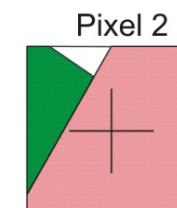
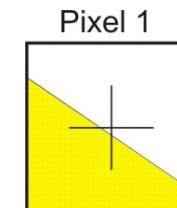
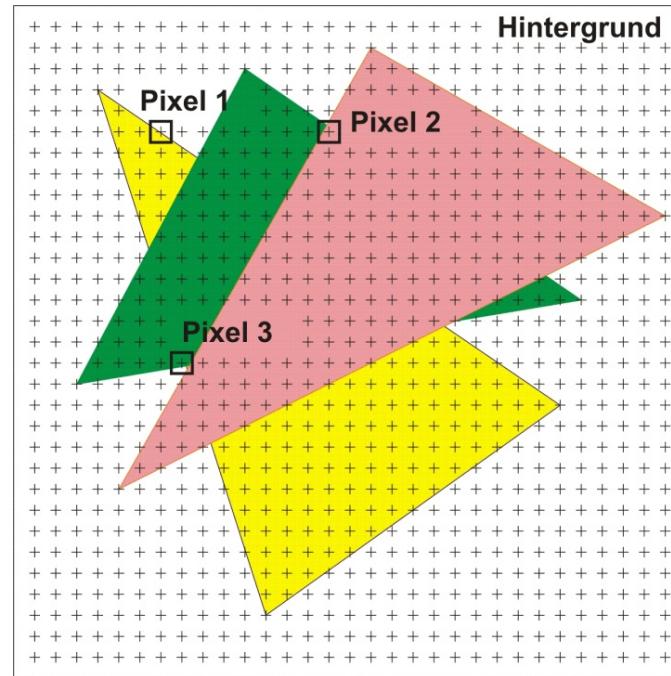
Liniendarstellung mit Konturglätten (vergrößerter Ausschnitt)

3. Rendering

3.5 Rasterisierung und Antialiasing

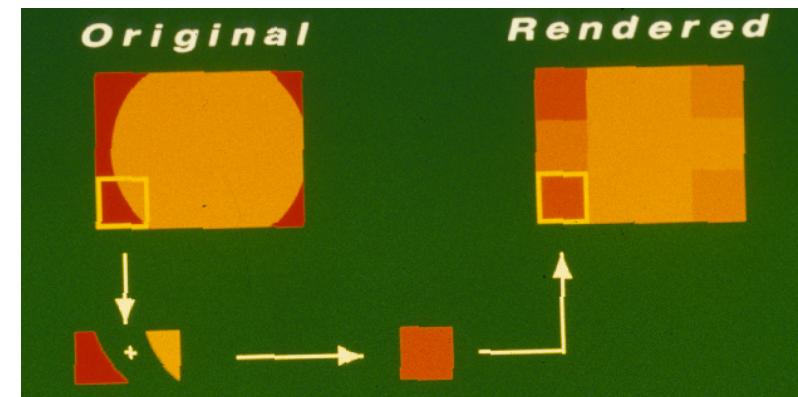
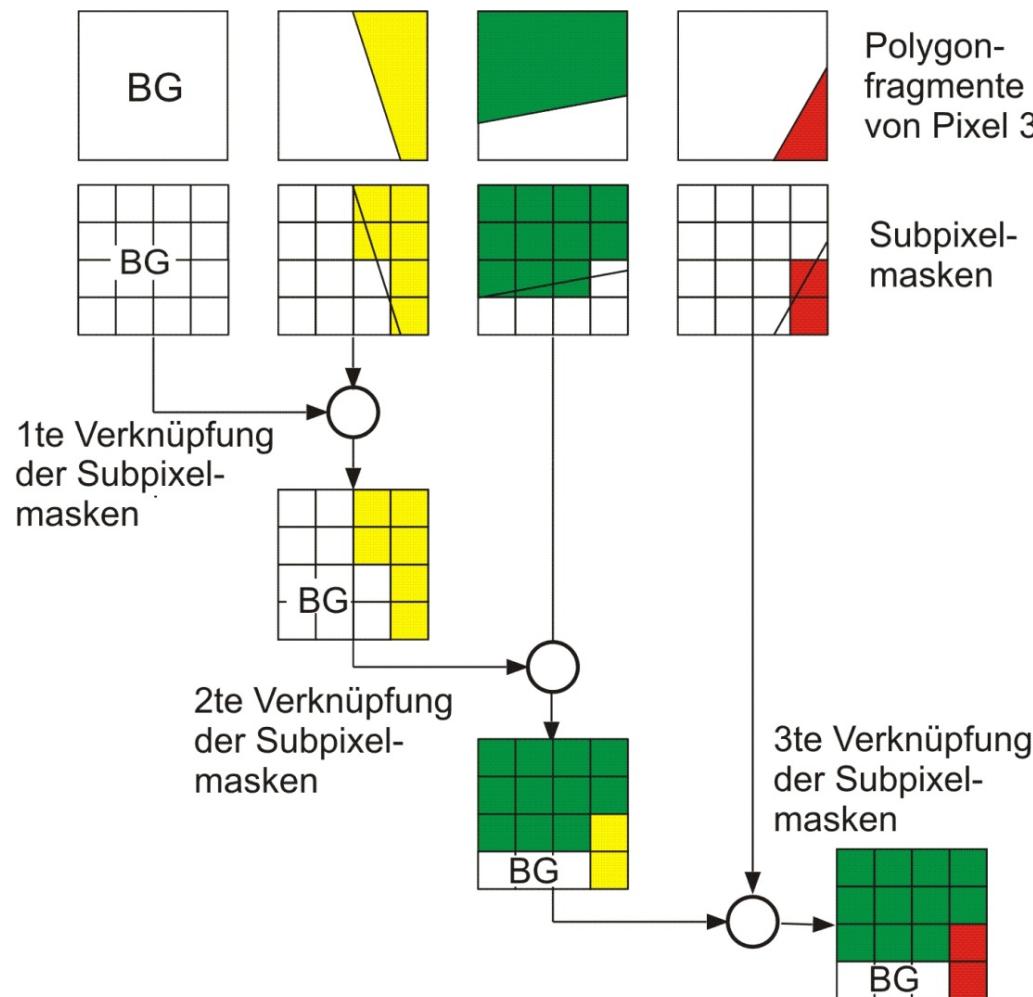


- Beim **Areasampling** wird anstelle des Pixelzentrums die Pixelfläche betrachtet. Es wird überprüft, welche Objekte diese Fläche in welchem Umfang überdecken. Hierbei arbeitet man i. allg. mit Subpixelmasken.



3. Rendering

3.5 Rasterisierung und Antialiasing



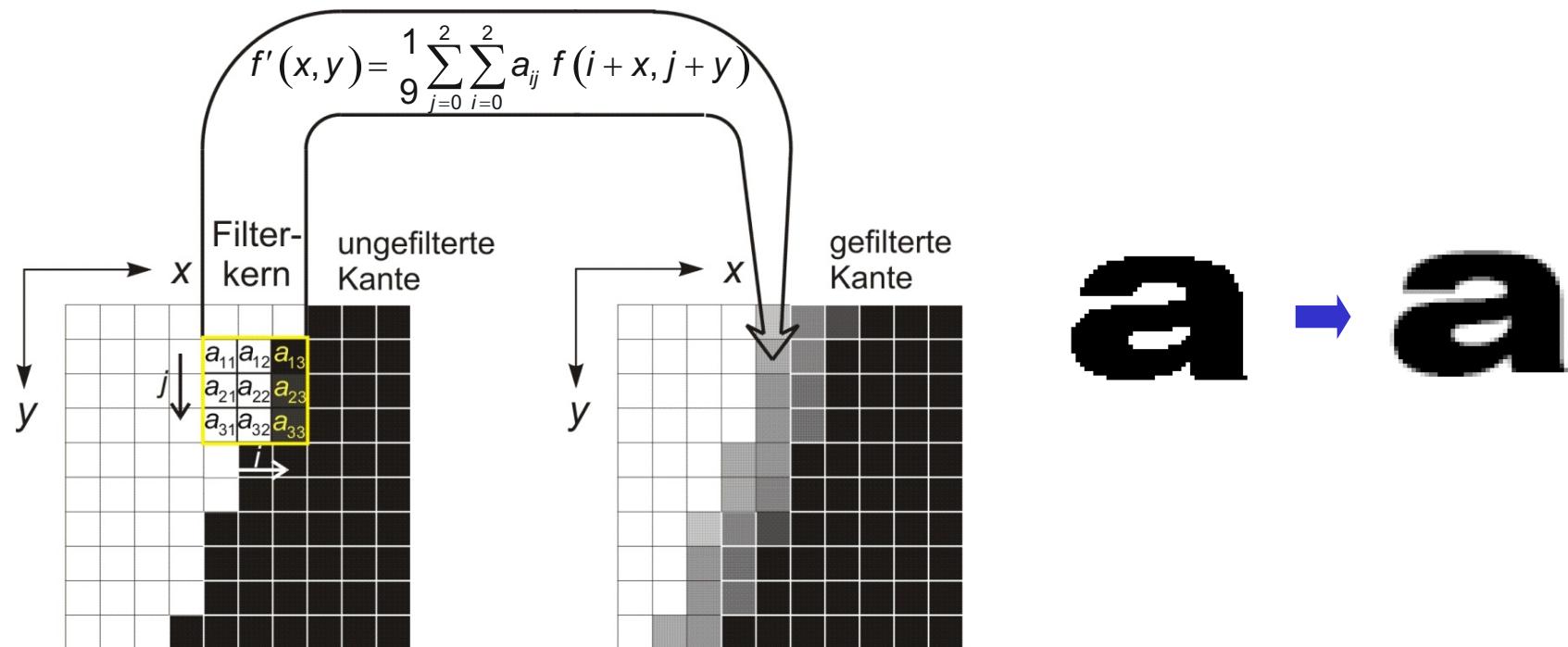
$$\text{Pixelfarbe} = \frac{2}{16} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \frac{11}{16} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + \frac{3}{16} [BG]$$

3. Rendering

3.5 Rasterisierung und Antialiasing

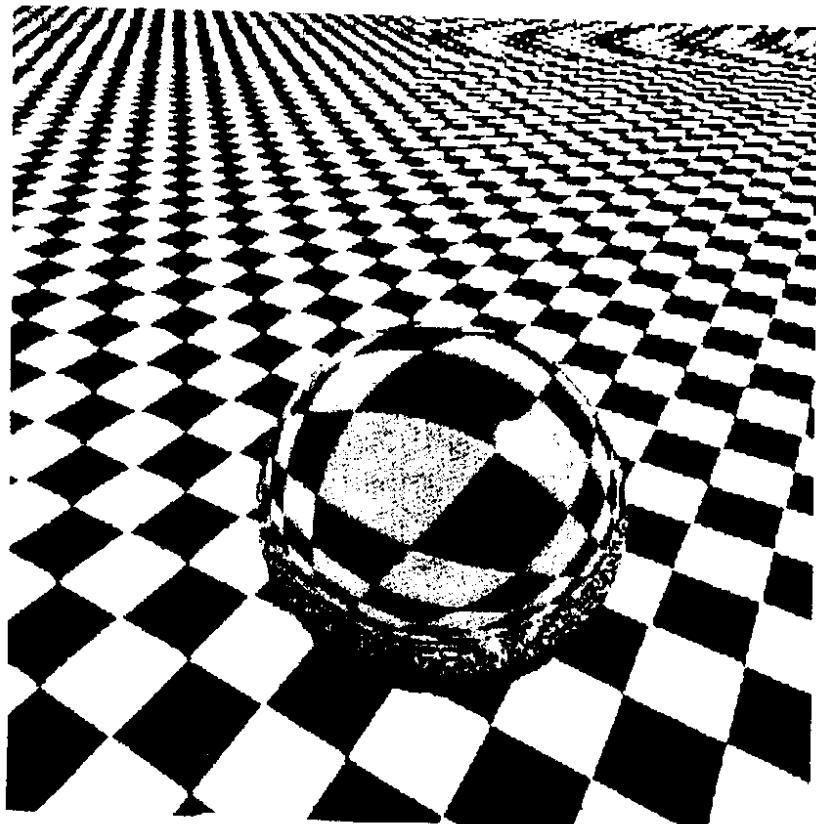


- Beim **Antialiasing mit einem Faltungsoperator** wird eine Filtermaske über das Bild geschoben. Die überdeckten Pixelwerte werden mit den Filterwerten multipliziert und daraus ein Gesamtwert berechnet.





- **Antialiasing mit Supersampling** bedeutet, dass pro Pixel die Farbinformation mehrerer Punkte ausgewertet wird.



*Ohne (rechts) und mit Antialiasing
(Jittersampling, links)*

