Computergrafik II

Dr. Philipp Hurni, Kantonsschule Sursee

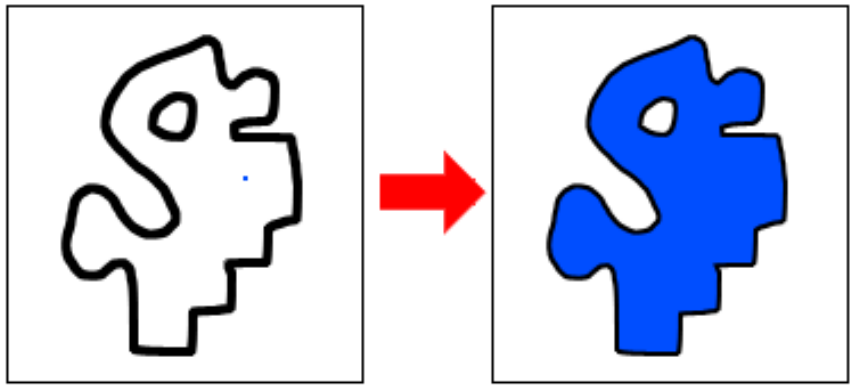
Agenda

Elementare Algorithmen der Computergrafik

- Füllen
  - Der Flood Fill Algorithmus
- Rastern
  - Der Bresenham Algorithmus
- Glätten
  - Antialiasing (Kantenglättung)

Utah Teapot

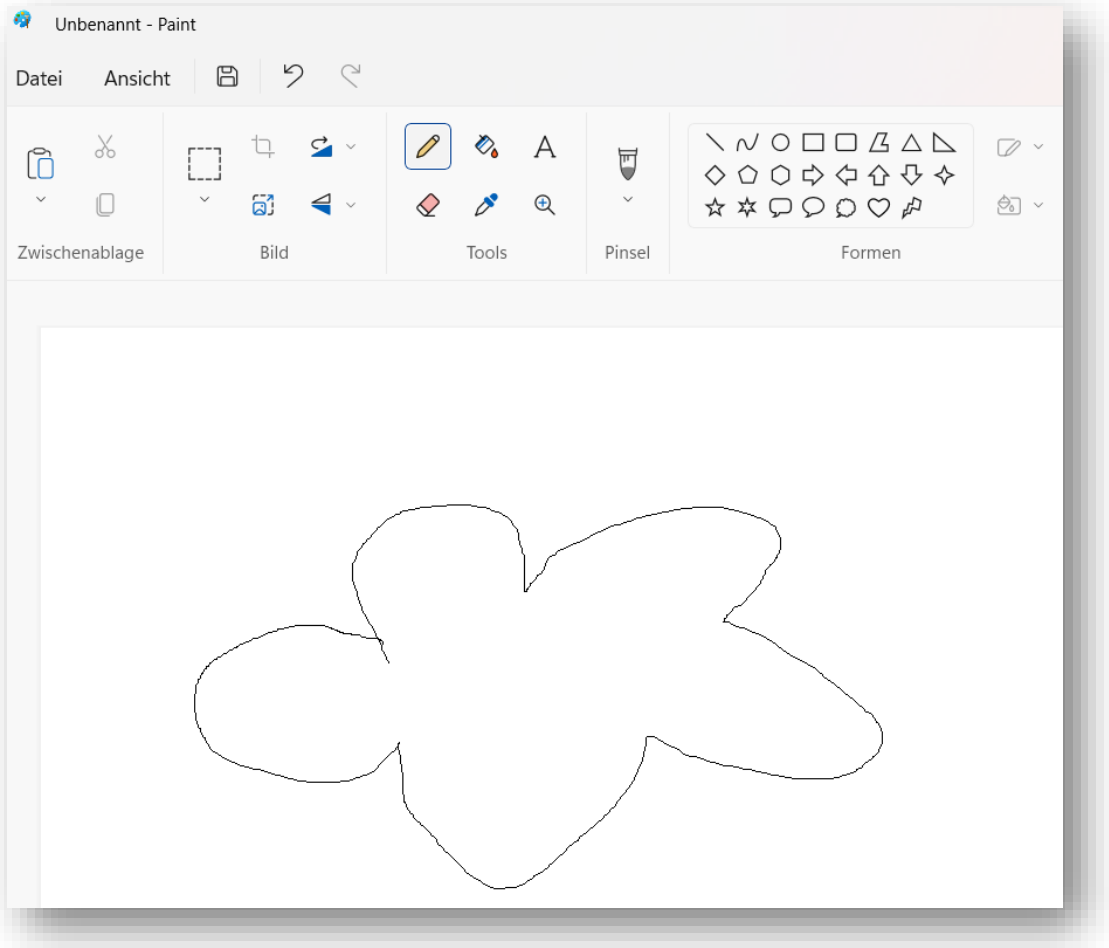
Elementare Algorithmen in der Computergrafik



Der FloodFill Algorithmus

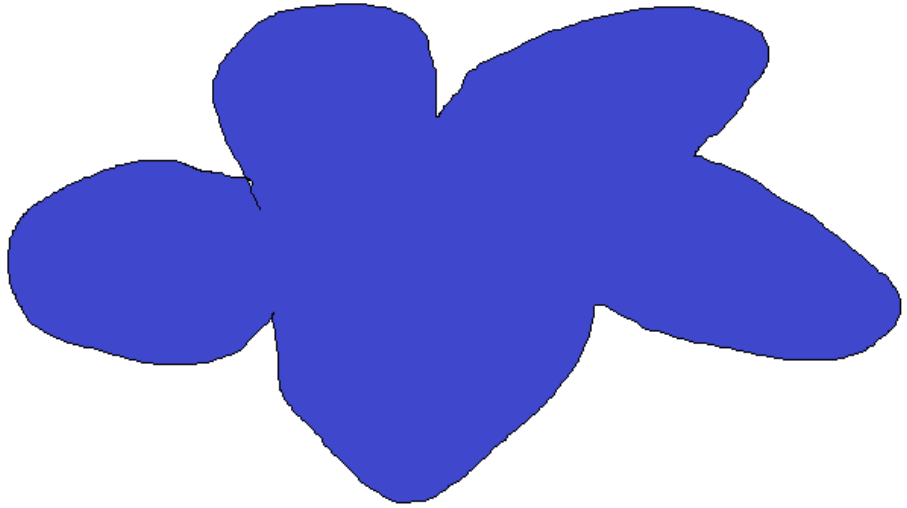
Flood Fill

- Öffnen Sie Paint und zeichnen Sie eine Wolke oder ein anderes, geschlossenes Objekt
- Drücken Sie auf den Knopf der mit «Füllen» benannt ist und wählen Sie eine Farbe (z.B. Blau)
- Drücken Sie innerhalb des Objektes mit «Füllen» und beobachten Sie was passiert

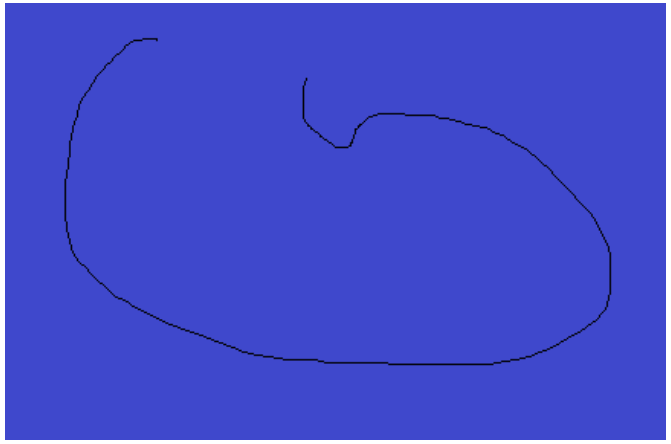


Flood Fill

- Das Objekt wird gefüllt.



- Wiederholen Sie den Vorgang mit einem nicht geschlossenen Objekt – Sie werden merken dass die «Flut» nicht aufgehalten wurde.



FloodFill

- Floodfill bzw. Flutfüllung ein einfacher Algorithmus, um Flächen zusammen-hängender Pixel einer Farbe in einem digitalen Bild zu erfassen und mit einer neuen Farbe zu füllen.
- Ausgehend von einem Pixel innerhalb der Fläche werden jeweils dessen Nachbarpixel darauf getestet, ob diese Nachbarpixel auch die alte Farbe enthalten.
- Jedes gefundene Pixel mit der alten Farbe wird dabei sofort durch die neue Farbe ersetzt, und dasselbe Prozedere wird mit den Nachbarpixeln rekursiv fortgesetzt
- Falls das neue Pixel nicht die alte Farbe enthält, stoppt der rekursive Aufruf von FloodFill.

FloodFill (mit Python)

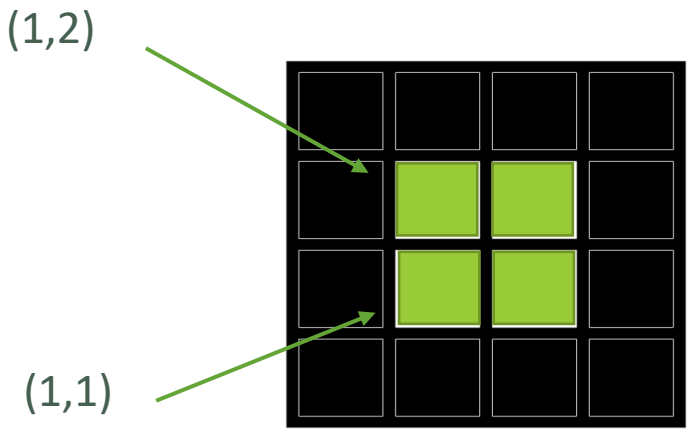
Rekursiver Selbstaufruf!

```
def floodfill(x, y, alteFarbe, neueFarbe):
    print("floodfill(", x,",",y,",alteFarbe, neueFarbe))")
    if getpixelcolor(x, y) == alteFarbe:
        drawpixel(x, y, neueFarbe)
        floodfill(x, y + 1, alteFarbe, neueFarbe)      # oben
        floodfill(x, y - 1, alteFarbe, neueFarbe)      # unten
        floodfill(x - 1, y, alteFarbe, neueFarbe)      # links
        floodfill(x + 1, y, alteFarbe, neueFarbe)      # rechts
    else:
        print("stop floodfill(",x,",",y,")")
        # anhalten!
```

Abbruchbedingung!

Rekursionsbaum

- floodfill(1,1) - **zeichnet (1,1)!**
  - floodfill(1,2) # oben von (1,1) – **zeichnet (1,2)!**
    - floodfill(1,3) # oben von (1,2) – bricht ab
    - floodfill(1,1) # unten von (1,2) – bricht ab
    - floodfill(0,2) # links von (1,2) – bricht ab
    - floodfill(2,2) # rechts von (1,2) – **zeichnet (2,2)!**
      - floodfill(2,3) # oben von (2,2) – bricht ab
      - floodfill(2,1) # unten von (2,2) – **zeichnet (2,1)!**
        - floodfill(2,2) # oben von (2,1) – bricht ab
        - floodfill(2,0) # unten von (2,1) – bricht ab
        - floodfill(1,1) # links von (2,1) – bricht ab
        - floodfill(3,1) # links von (2,1) – bricht ab
      - floodfill(1,2) # links von (2,2) – bricht ab
      - floodfill(3,2) # rechts von (2,2) – bricht ab
  - floodfill(1,0) # unten von (1,1) – bricht ab
  - floodfill(0,1) # links von (1,1) – bricht ab
  - floodfill(2,1) # rechts von (1,1) – bricht ab



FloodFill mit Python

Öffnen Sie die Datei "Algorithmen und Rekursion IV - FloodFill Ia"

Navigieren Sie zu diesem Code hier unten:

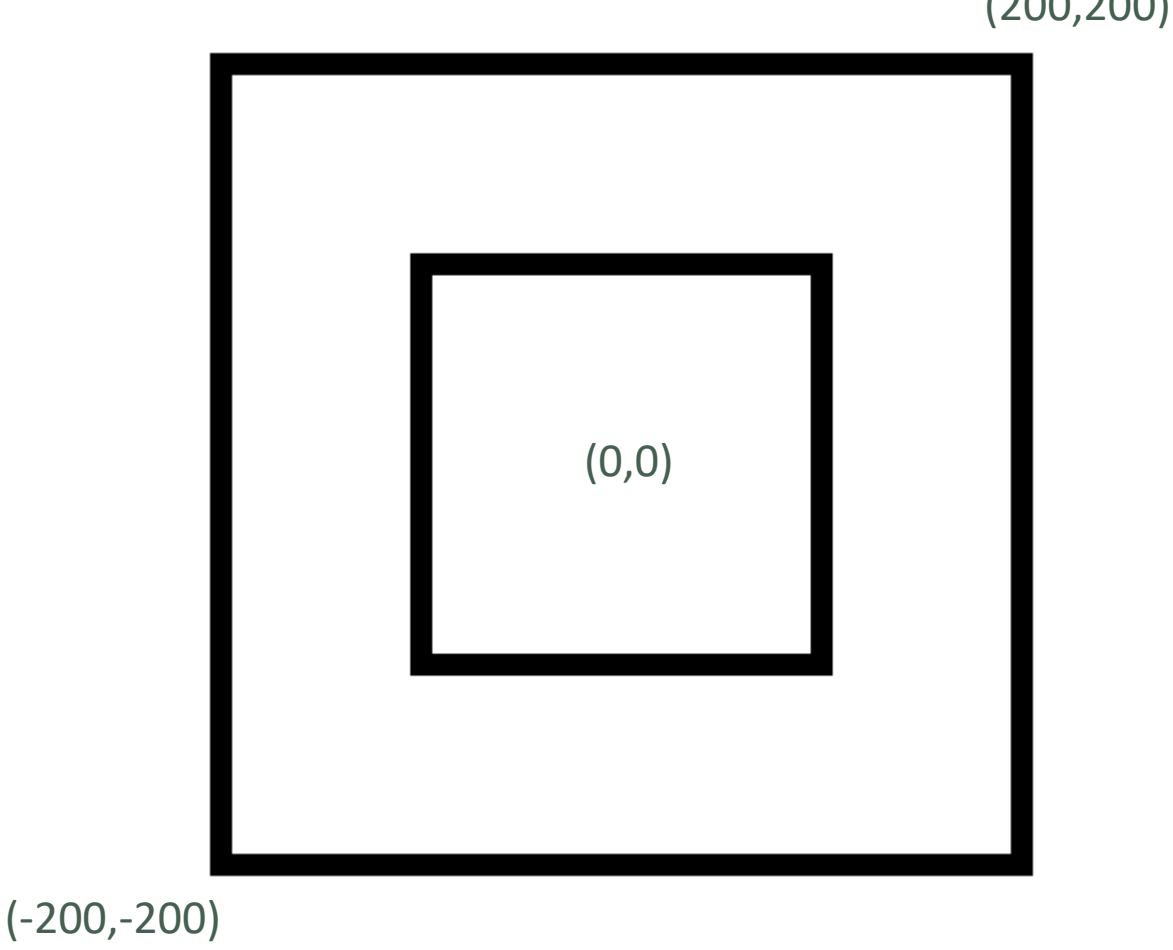
```
def drawing_prepare():
    clear()
    # zeichne das schwarze Quadrat
    drawsquare(20, "black")
    # zeichne das zweite schwarze Quadrat
    drawsquare(40, "black")
    # zeichne das dritte schwarze Quadrat
    drawsquare(200, "black")

def drawing_floodfill(x,y):
    floodfill(x,y,"white", "blue")
```

Rufen Sie nun den Algorithmus FloodFill für das innere Quadrat auf. Das machen Sie indem Sie in die Grafik klicken. Dabei wird x und y als Parameter übergeben

floodfill(x,y,"white","blue")

Beobachten Sie was passiert.



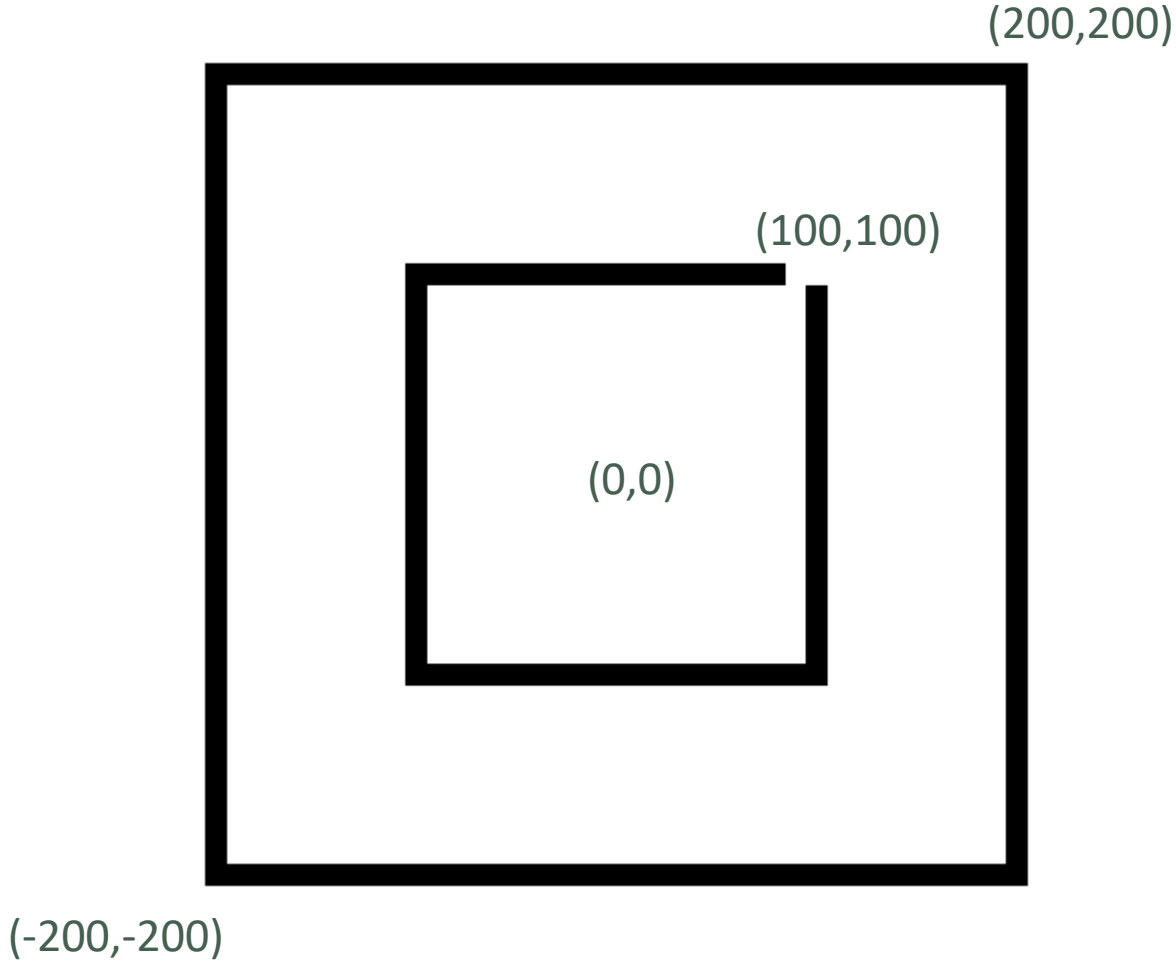
FloodFill mit Python

Öffnen Sie die Datei "Algorithmen und Rekursion IV - FloodFill Ib"

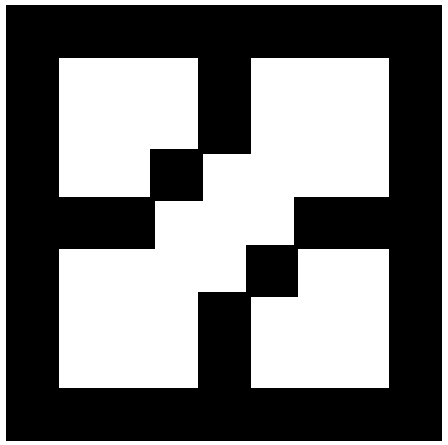
Hier wurden zwei Pixel in der oberen rechten Ecke entfernt:

Rufen Sie nun den Algorithmus FloodFill für das innere Quadrat auf. Das machen Sie indem Sie in die Grafik klicken. Dabei wird x und y als Parameter übergeben

Was erwarten Sie dass passiert?



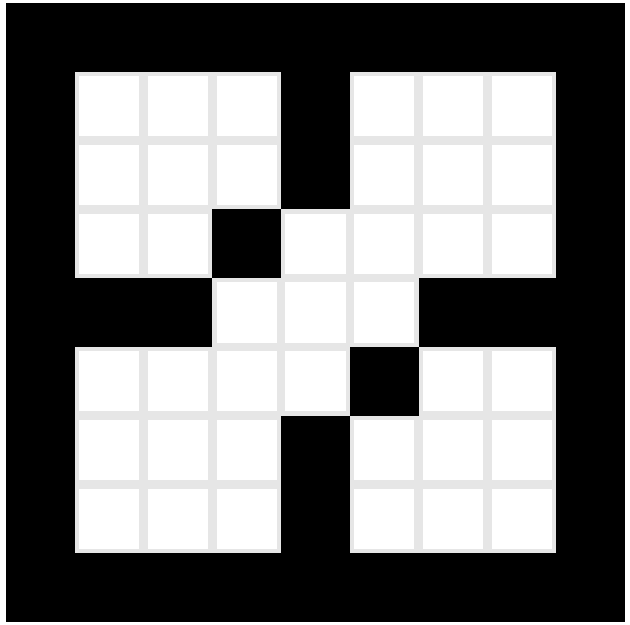
FloodFill mit diagonalen Linien

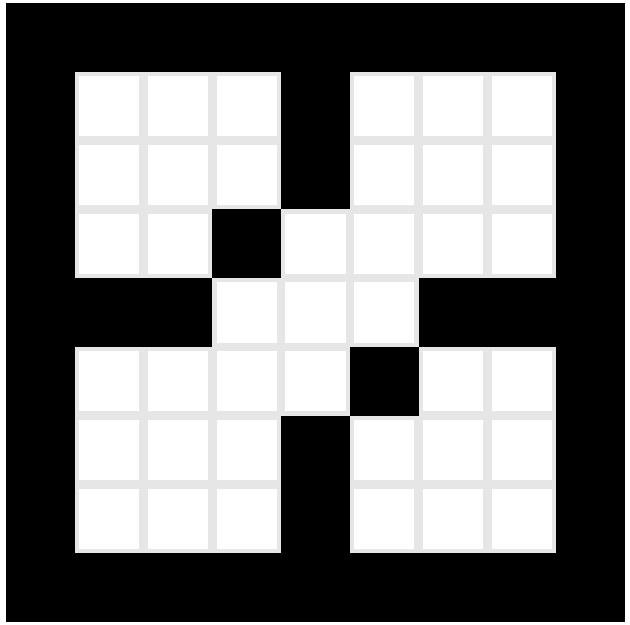


Was passiert, wenn ich FloodFill an der Position (1,1) ausführe?

Probier es aus. Nutze dazu “Algorithmen und Rekursion IV - FloodFill II”.

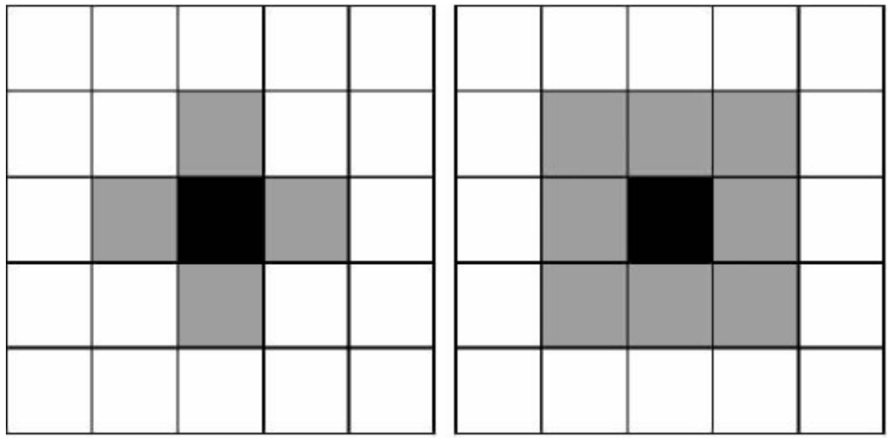
4 Neighbor vs. 8 Neighbors





Aufgabe FloodFill II

Ergänzen Sie den Code in "Computergrafik II - FloodFill II" um die Variante, wo Floodfill mit allen Nachbarn ausgeführt wird (nord, süd, west, ost + nordwest, nordost, südwest, südost)

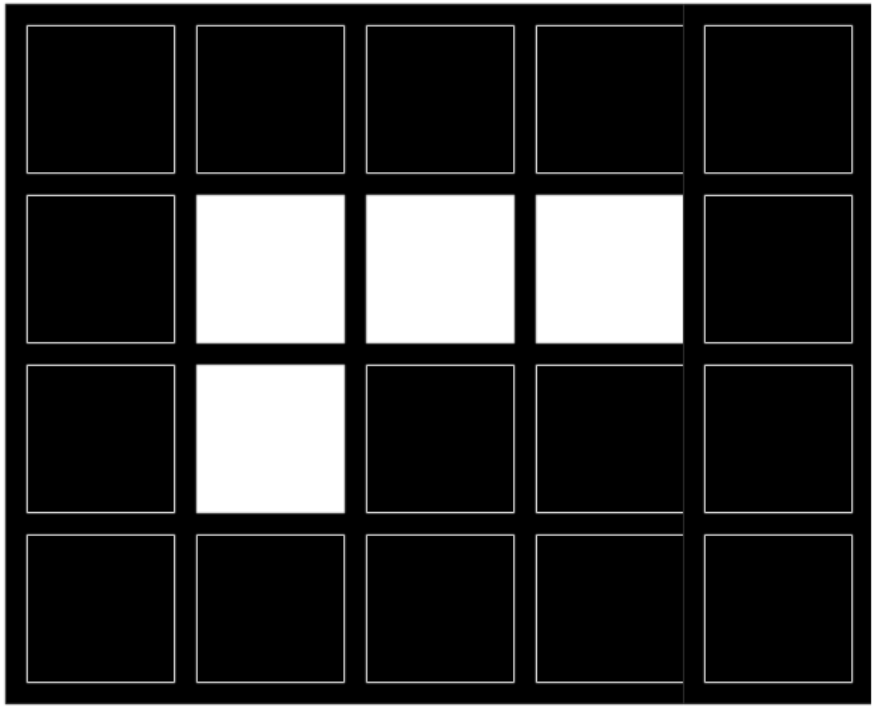


```
    # DIESEN CODE MUSST DU SCHREIBEN

def floodfill8(x, y, alteFarbe, neueFarbe):
    drawpixel(x, y, neueFarbe)
```

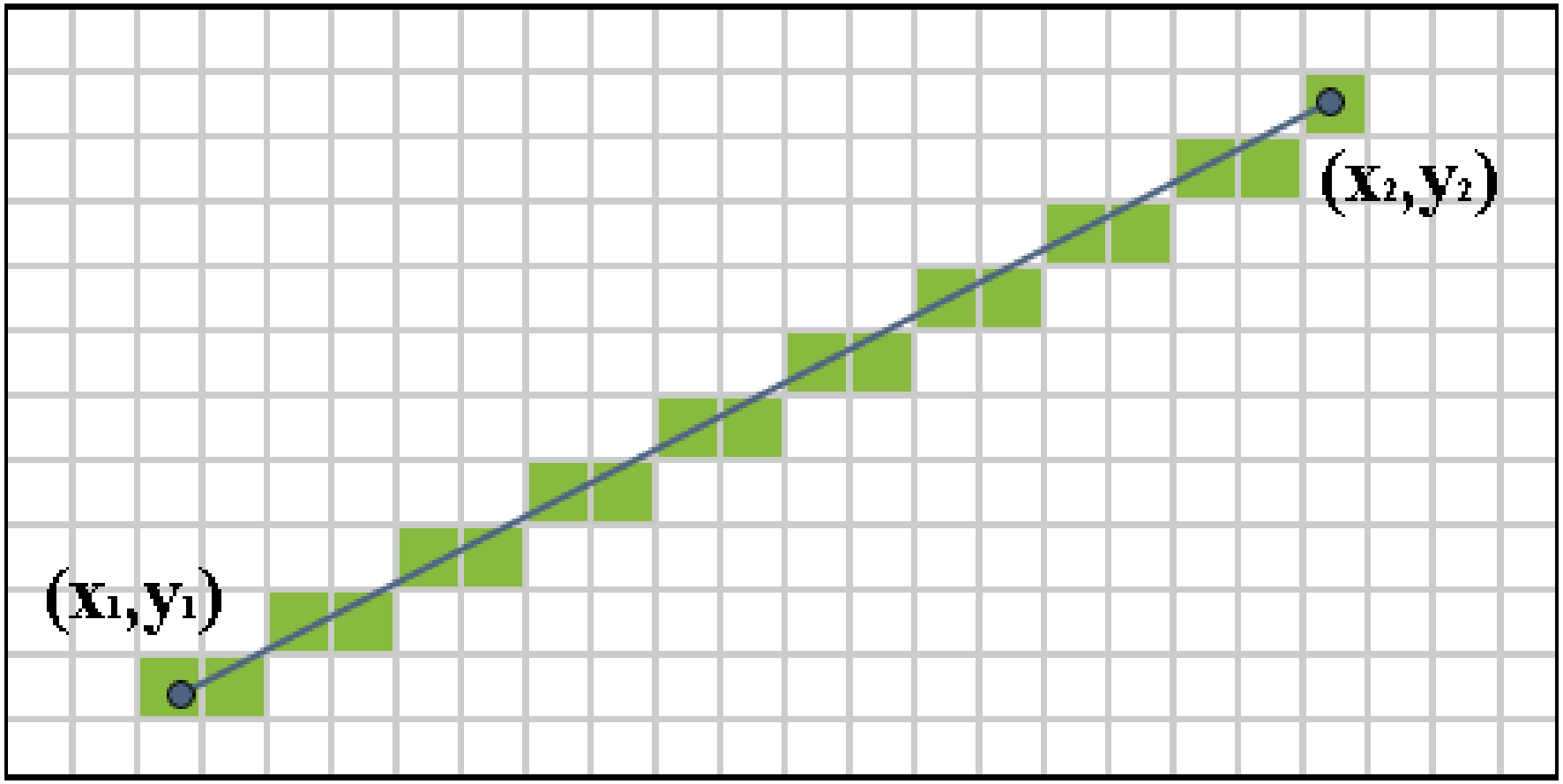
Aufgabe FloodFill III

Beschreiben Sie den Rekursionsbaum für die Figur rechts, wenn Sie FloodFill4 bei (1,1) starten



Rasterung

Bresenham Algorithmus



Bresenham Algorithmus

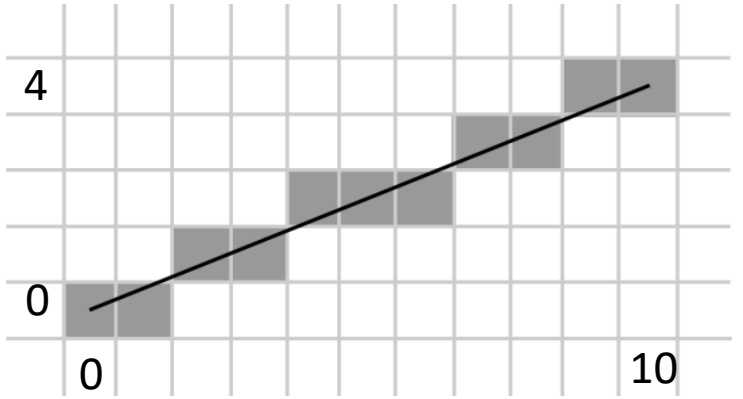
- Der Algorithmus wurde 1962 von Jack Bresenham entwickelt.
- Der Algorithmus ist sehr einfach gehalten, und wird auf zahlreichen Systemen heute verwendet.
- Durch eine geringfügige Erweiterung lässt sich der ursprüngliche Algorithmus, der für Geraden entworfen wurde, auch für die Rasterung von Kreisen verwenden

Bresenham Algorithmus (2)

- Aufgrund dieser Eigenschaften ist die Bedeutung des Bresenham-Algorithmus bis heute ungebrochen, und er kommt unter anderem in Plottern, in den Grafikchips moderner Grafikkarten und in vielen Grafikbibliotheken zur Verwendung.

- Dabei ist er so einfach, dass er nicht nur in der Firmware solcher Geräte verwendet wird, sondern in Grafikchips direkt in Hardware implementiert werden kann.

Bresenham Algorithmus (3)



- Zum Verständnis des Algorithmus beschränkt man sich auf den ersten Oktanten, also eine Linie mit einer Steigung zwischen 0 und 1 von (x0, y0) nach (x1, y1). Seien $dx = x1 – x0$ und $dy = y1 – y0$ mit $0 < dy < dx$.
- Für andere Oktanten muss man später Fallunterscheidungen über Vorzeichen von dx und dy und die Rollenvertauschung von x und y treffen

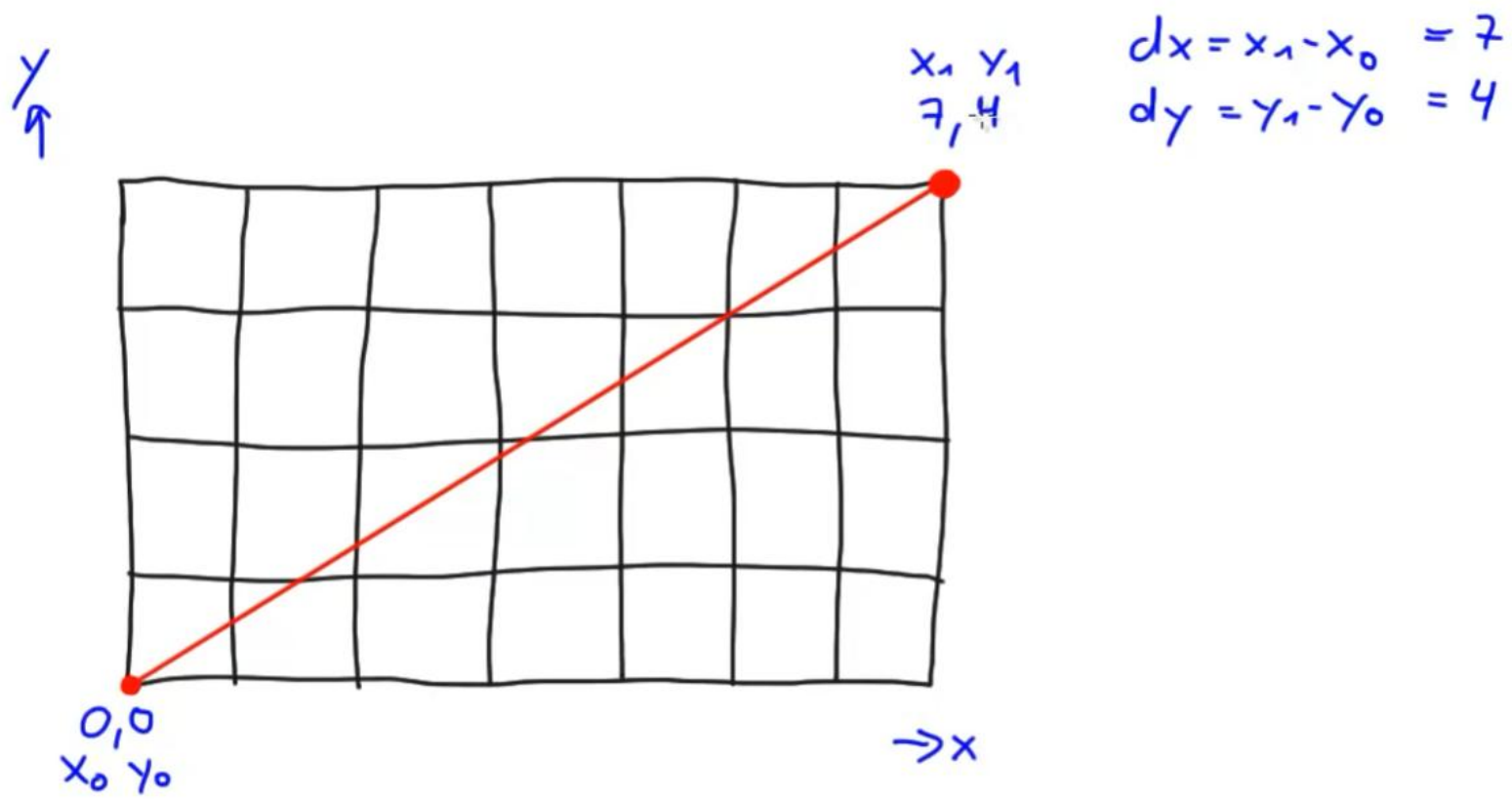
Bresenham Algorithmus (4)

- Der Algorithmus läuft dann so, dass man in der „schnellen“ Richtung (hier die positive x-Richtung) immer einen Schritt macht und je nach Steigung hin und wieder zusätzlich einen Schritt in der „langsameren“ Richtung (hier y).
- Man benutzt dabei eine Fehlervariable, die bei einem Schritt in x-Richtung den (kleineren) Wert dy subtrahiert bekommt. Bei Unterschreitung des Nullwerts wird ein y-Schritt fällig und der (grössere) Wert dx zur Fehlervariablen addiert. Diese wiederholten „Überkreuz“-Subtraktionen und -Additionen lösen die Division des Steigungsdreiecks m=dy/dx in elementar Rechenschritte auf.
- Zusätzlich muss diese Fehlervariable vorher initialisiert werden. Man initialisiert die Fehlervariable mit (dx/2).

Bresenham Algorithmus Ablauf

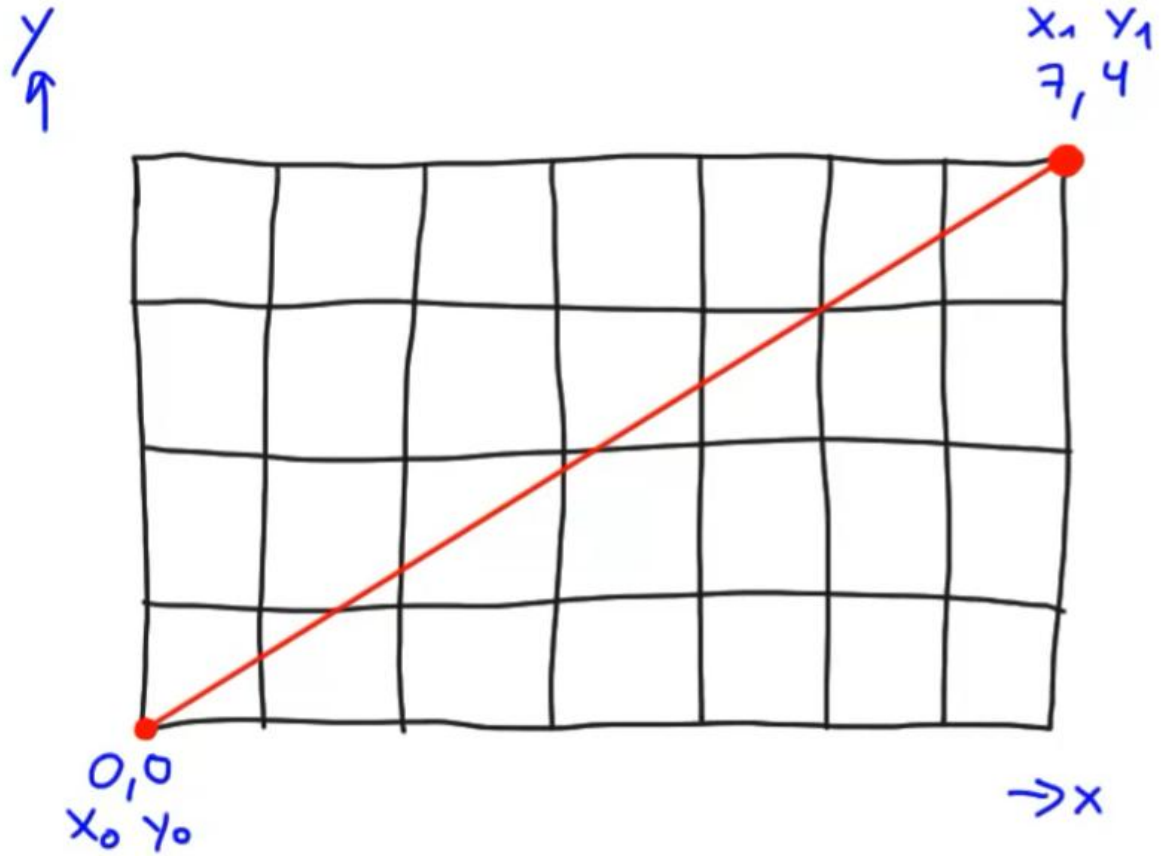
- dx, dy berechnen
- schnelle Richtung bestimmen
- Fehlerterm f = “schnelle Richtung” / 2
- 1. Punkt markieren
- Schleife bis zum vorletzten Punkt der schnellen Richtung:
  - 1 Kästchen in die schnelle Richtung
  - f = f_alt – d(Langsame Richtung)
    - Wenn f < 0 dann:
      - 1 Feld in langsame Richtung
      - f = f_alt + d(Schnelle Richtung)
    - Pixel
- Setze letzten Pixel

Bresenham Algorithmus an einem Beispiel



https://www.youtube.com/watch?v=vlZFSzCIwoc

Bresenham Algorithmus an einem Beispiel



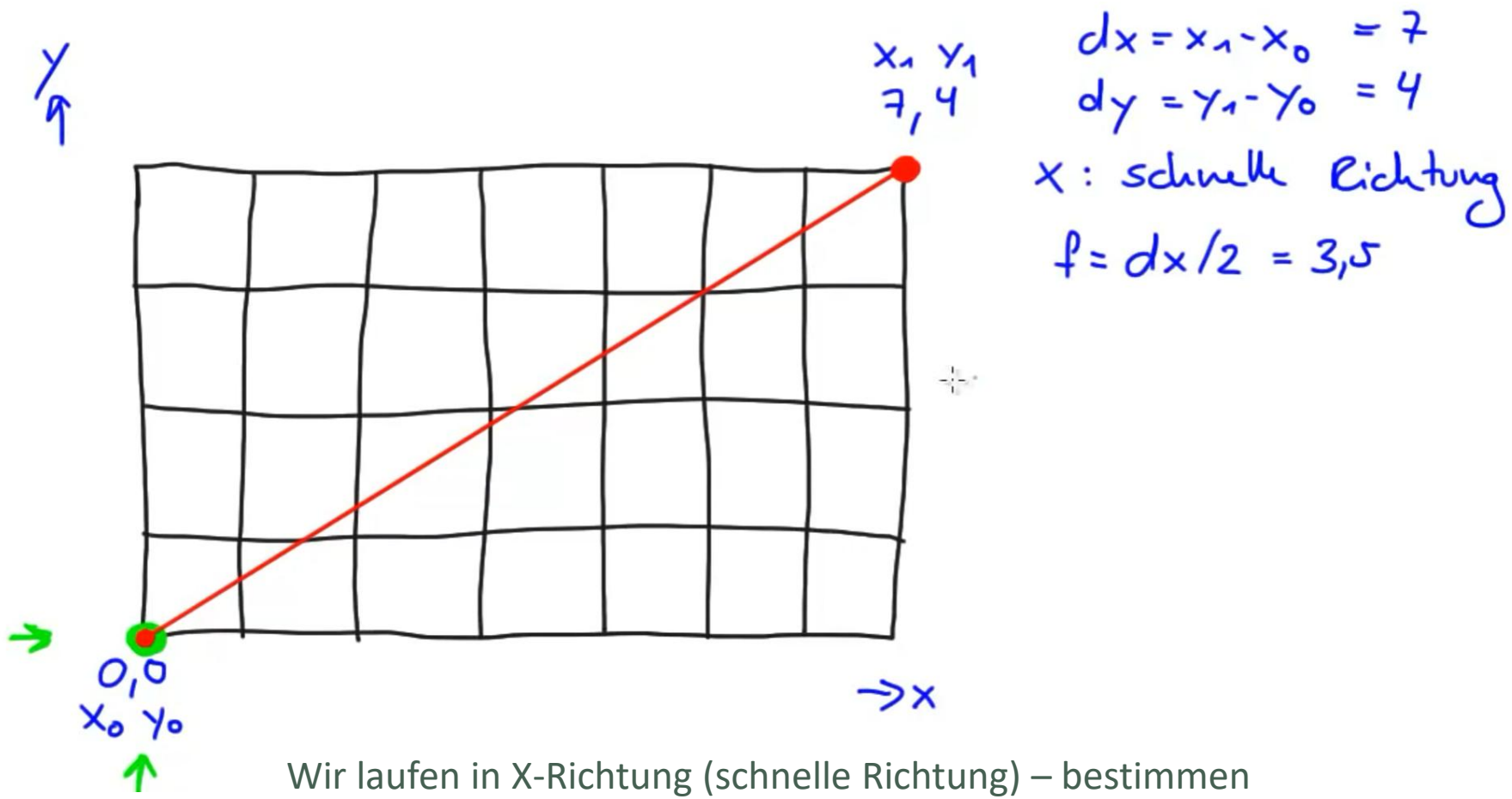
$dx = x_1 - x_0 = 7$

$dy = y_1 - y_0 = 4$

x: schnelle Richtung

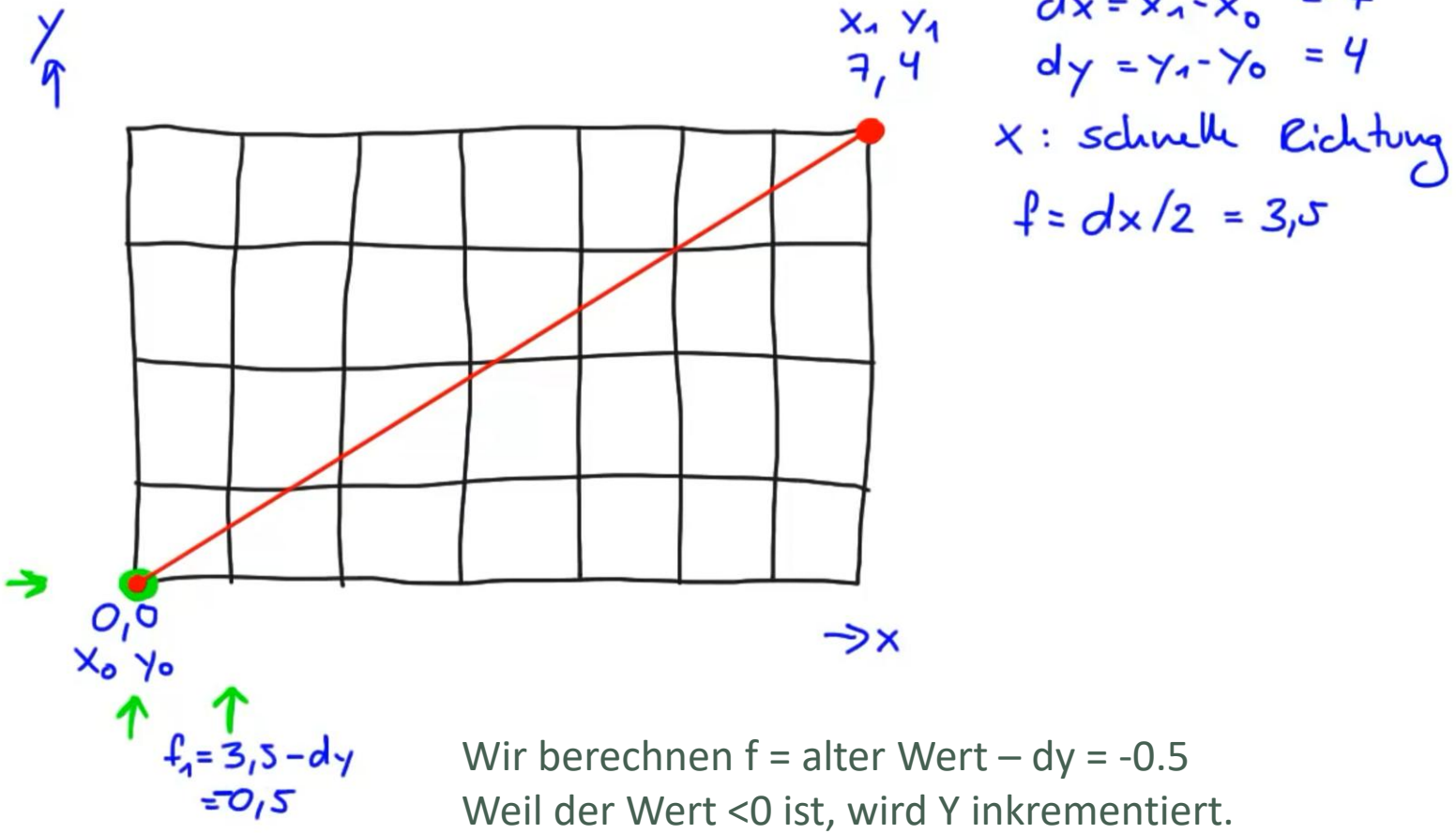
$f = dx/2 = 3{,}5$

Bresenham Algorithmus an einem Beispiel



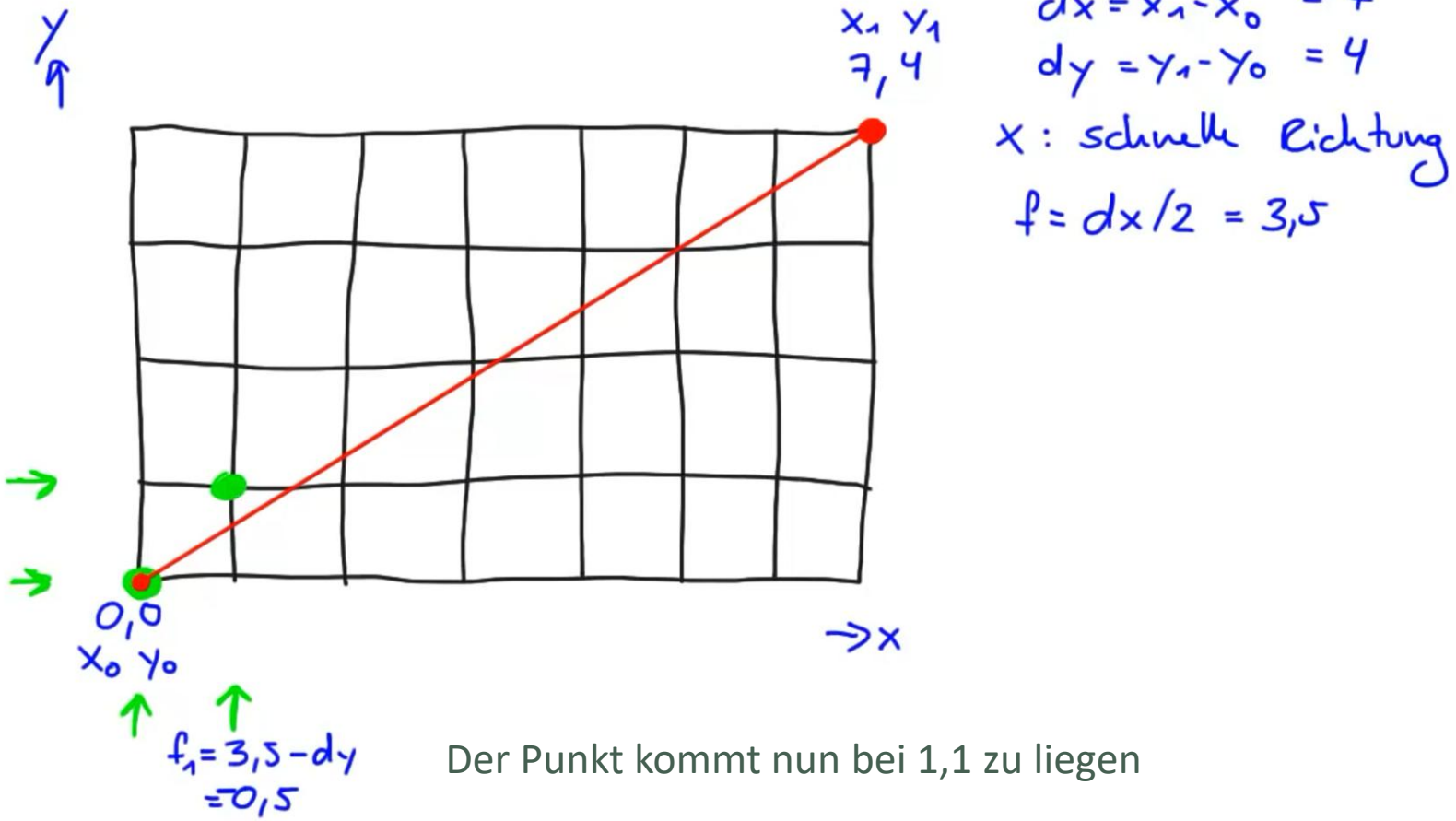
Wir laufen in X-Richtung (schnelle Richtung) – bestimmen jeweils ob wir Y inkrementieren oder nicht

Bresenham Algorithmus an einem Beispiel



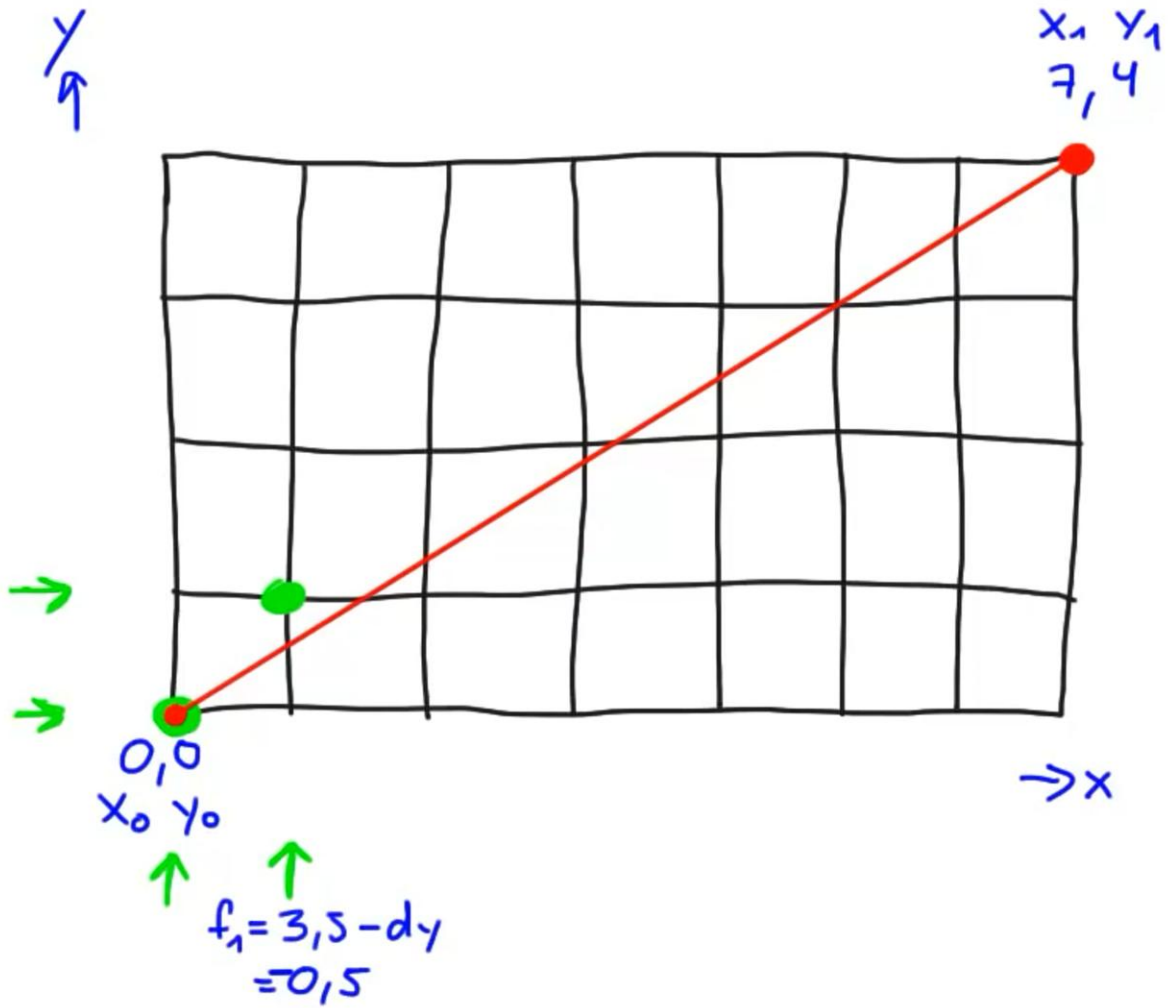
Wir berechnen f = alter Wert – dy = -0.5
Weil der Wert <0 ist, wird Y inkrementiert.

Bresenham Algorithmus an einem Beispiel



Der Punkt kommt nun bei 1,1 zu liegen

Bresenham Algorithmus an einem Beispiel



$dx = x_1 - x_0 = 7$

$dy = y_1 - y_0 = 4$

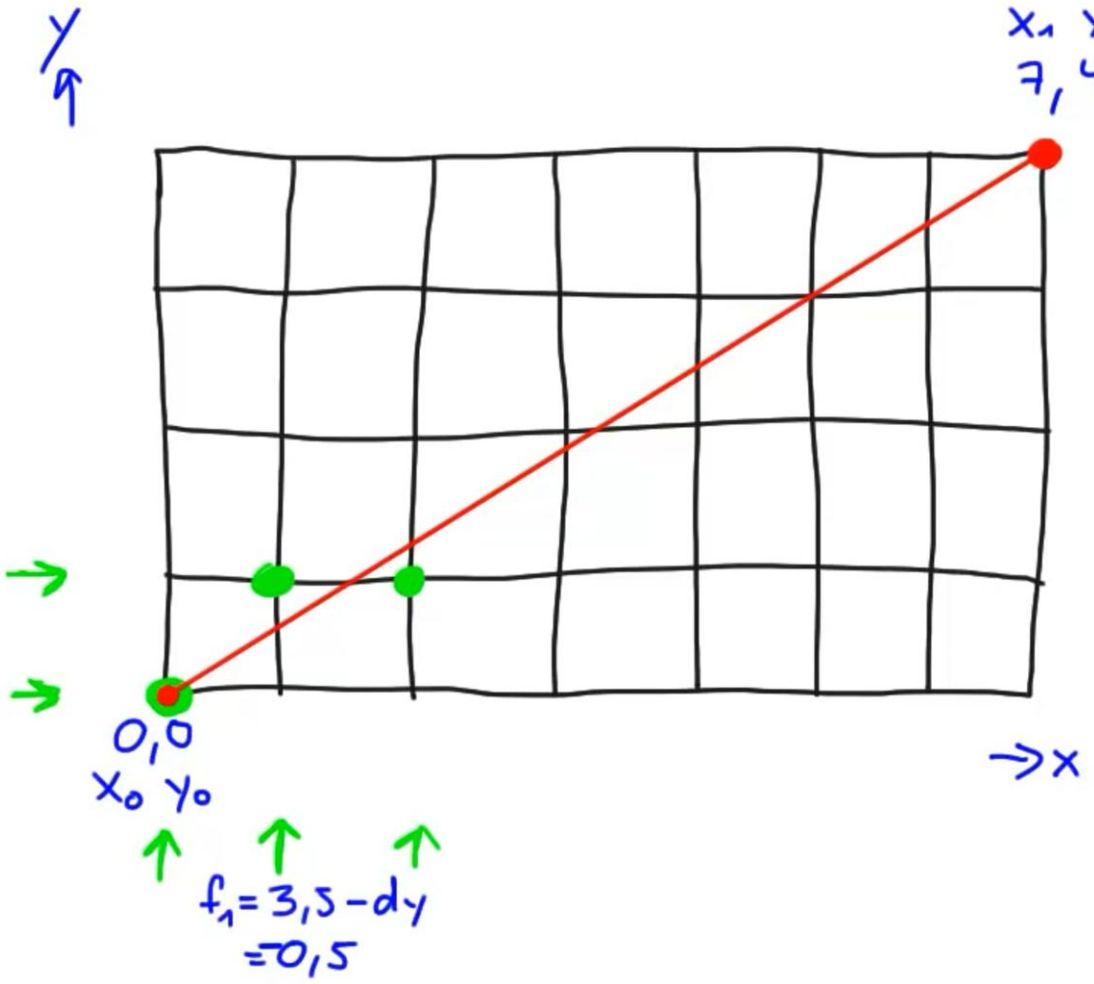
x: schnelle Richtung

$f = dx/2 = 3{,}5$

$f_2 = 6{,}5$

Der Fehlerterm muss nun verändert werden – dx wird hinzuaddiert (-0.5+7 = 6.5)

Bresenham Algorithmus an einem Beispiel



$dx = x_1 - x_0 = 7$

$dy = y_1 - y_0 = 4$

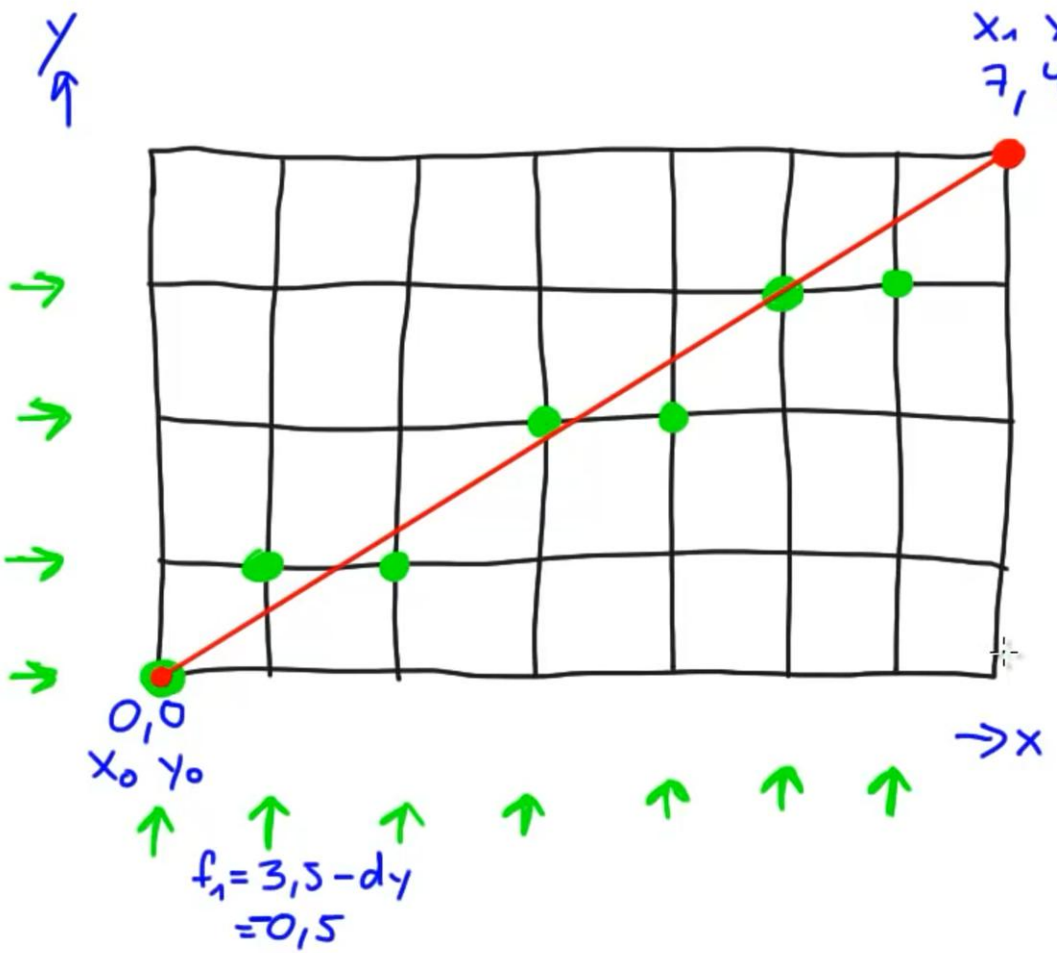
x: schnelle Richtung

$f = dx/2 = 3{,}5$

$f_2 = 6{,}5 - 4 = 2{,}5$

Nun wird wieder wie vorher weitergefahren. Dy wird vom Fehlerterm subtrahiert, der neue Wert ist 2.5. Weil 2.5 > 0 gehen wir nicht nach oben in Y-Richtung.

Bresenham Algorithmus an einem Beispiel



$dx = x_1 - x_0 = 7$

$dy = y_1 - y_0 = 4$

x: schnelle Richtung

$f = dx/2 = 3{,}5$

$f_2 = 6{,}5 - 4 = 2{,}5$

$f_3 = 2{,}5 - 4 = -1{,}5$

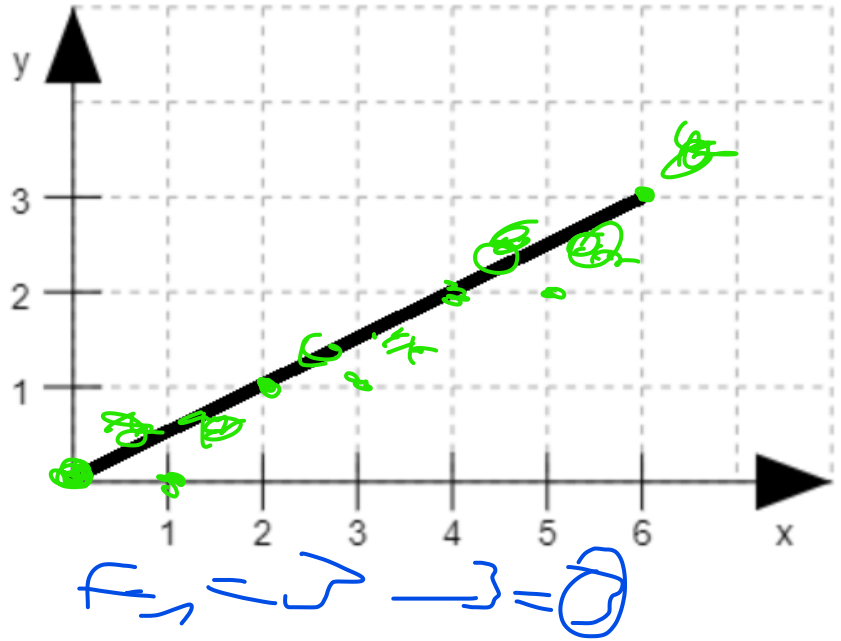
$f_4 = 5{,}5 - 4 = 1{,}5$

$f_5 = 1{,}5 - 4 = -2{,}5$

$f_6 = 4{,}5 - 4 = 0{,}5$

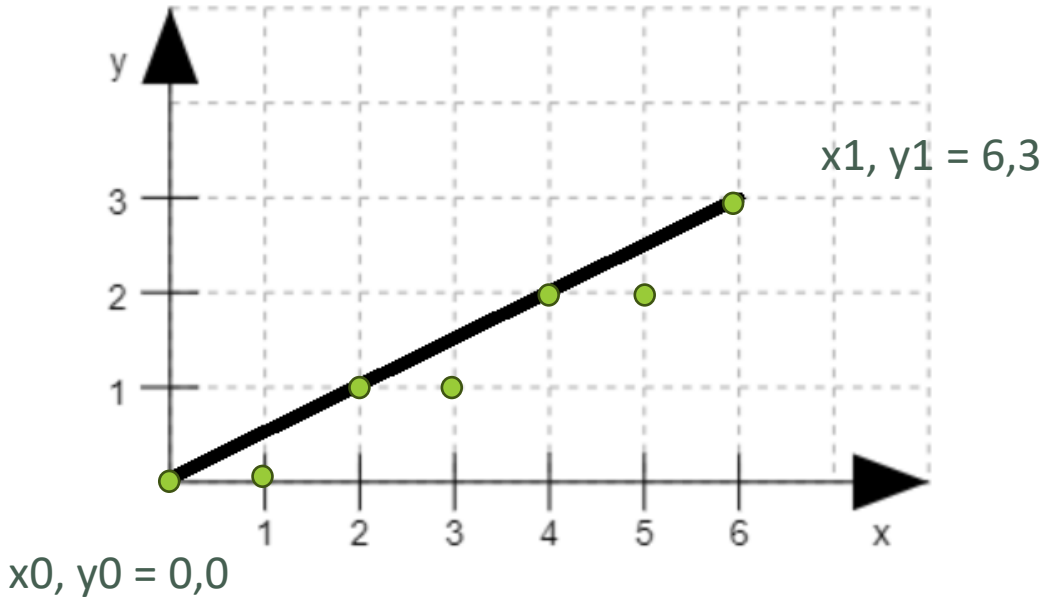
der Verlauf des Fehlerterms

Übung 1 Bresenham Algorithmus



Wende den Bresenham Algorithmus an für die obige Linie. Welche Pixels werden gezeichnet?

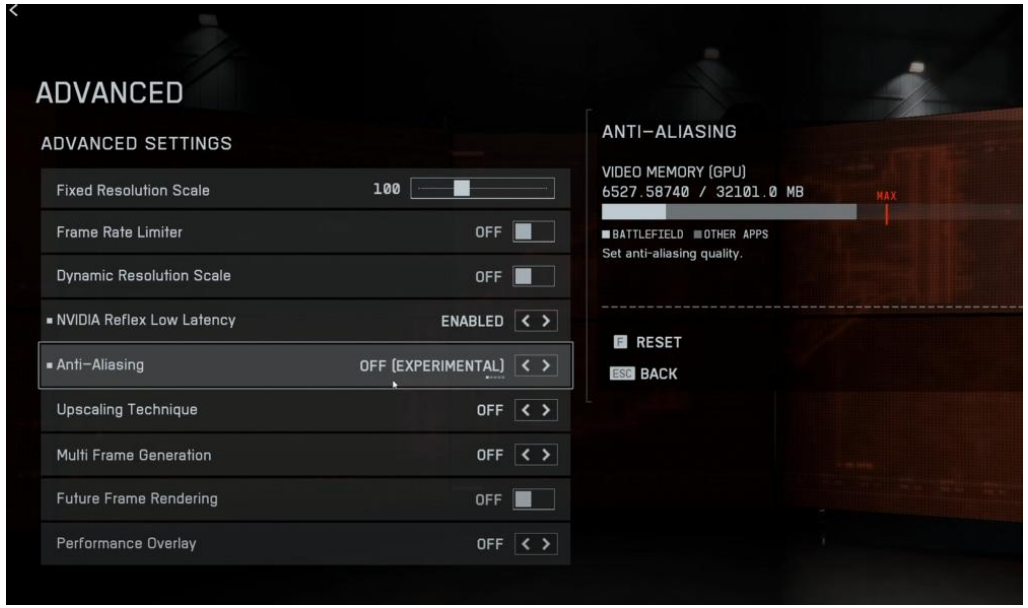
Übung 1 Bresenham Algorithmus



dx = x1-x0 = 6-0 = 6
dy = y1-y0 = 3-0 = 3
x: schnelle Richtung
f = dx/2 = 3
x=1
f1 = 3-dy = 0 wir gehen nicht nach oben
f2 = 0-3 = -3 wir gehen nach oben – Fehlerterm wird nun hinzuaddiert
x=2
y=1
f2 = -3 + 6 = 3
f3 = 3-3 = 0 wir gehen nicht nach oben
x=3
Y=1
f4 = 0-3 = -3 wir gehen nach oben – Fehlerterm wird nun hinzuaddiert
x=4
y=2
f5 = -3 + 6 = 3
f6 = 3 – 3 = 0 wir gehen nicht nach oben
x=5
y=2

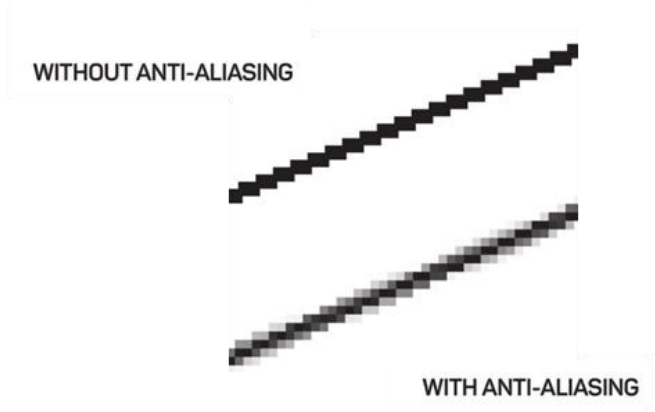
Antialiasing (Kantenglättung)

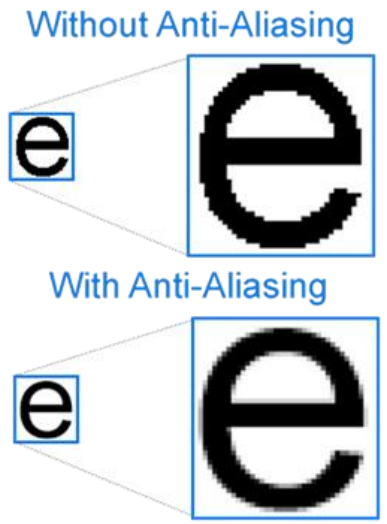
Antialiasing (Kantenglättung)



Antialiasing (Kantenglättung)

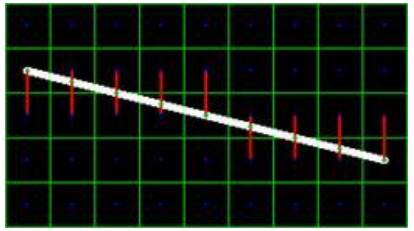
- Der "Treppeneffekt" ist ein unerwünschter Effekt, der entsteht, weil eine Grafik aus sichtbaren, quadratischen Pixeln besteht
- Anti-Aliasing behebt dies, indem es Pixel an den Kanten nicht nur in der Hauptfarbe oder Hintergrundfarbe darstellt, sondern als Mischung aus beiden.
- Das menschliche Auge nimmt diese als angenehmere weicheren Übergänge wahr und die Kanten erscheinen optisch glatter und weniger pixelig.

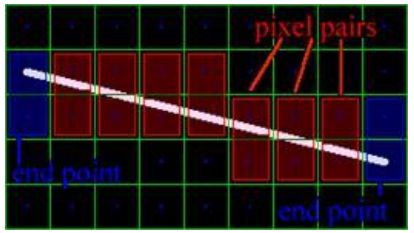




Antialiasing (Kantenglättung)

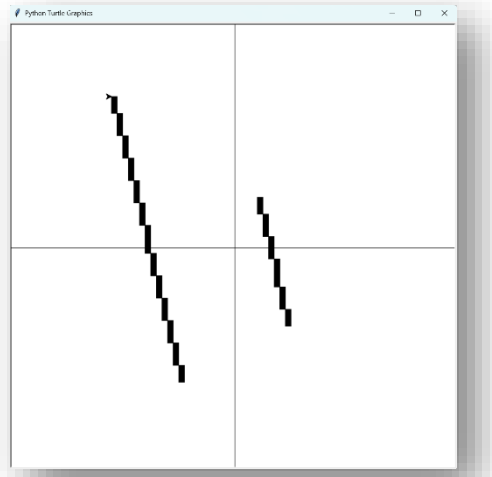
- Der Bresenham-Algorithmus ist bei der Darstellung von Linien zwar besonders schnell, unterstützt aber nicht die Glättung der Linien.
- Xiaolin Wu's Linien-Algorithmus ist ein Algorithmus für das Darstellen von Linien mit Antialiasing (Kantenglättung)
- Wus Algorithmus zeichnet Pixel immer **paarweise** auf je einer Seite der Linie und färbt sie nach ihrem Abstand von der Linie.
- Gesondert behandelt werden die Pixel an den Linienenden sowie Linien mit einer Länge kürzer als ein Pixel.





Experimentieren mit Bresenham und Wu!

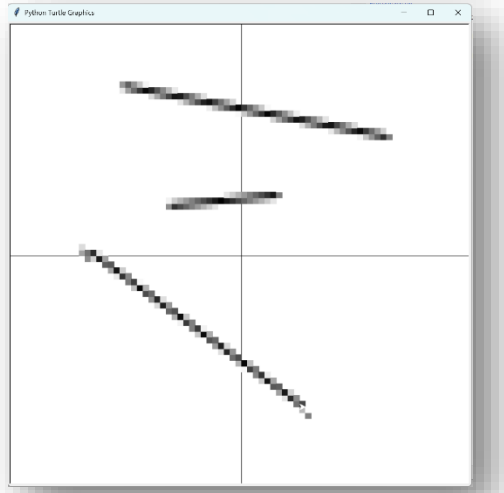
Bresenham vs. Wu



Ihr findet auf dem Teams die Dateien

- Bresenham.py
- Wu.py

Experimentiert damit, spielt damit



Fragen

- Welches Verfahren ist "schneller"?
- Welches Verfahren ist wo besser geeignet?