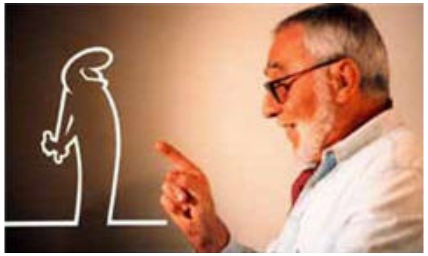
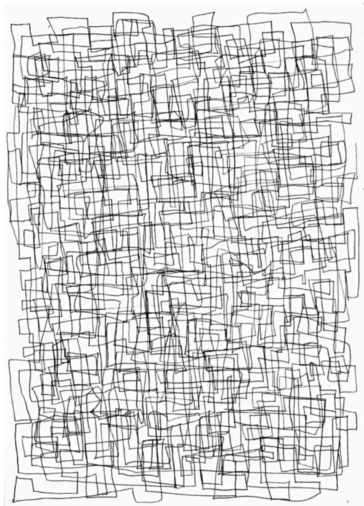
Computergraphik I
Scan Conversion of Lines





G. Zachmann
University of Bremen, Germany
cgvr.informatik.uni-bremen.de

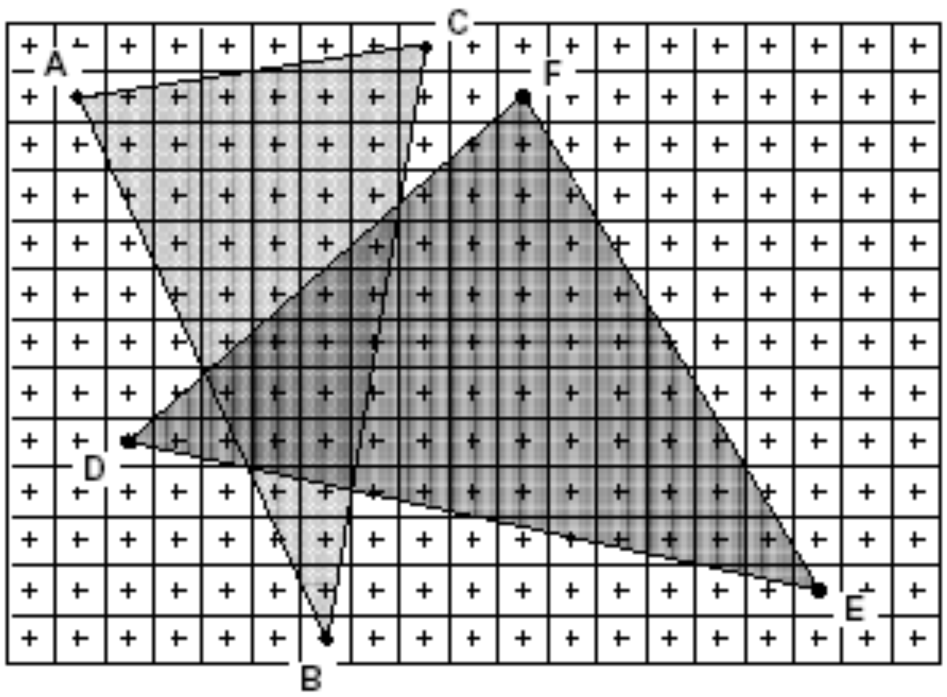
"La Linea"

Das Zeichnen von Linien

- Der Begriff Scan-conversion oder Rasterisierung bezeichnet allgemein das algorithmische Bestimmen, welche Pixel von dem Primitiv überdeckt werden
  - Der Name kommt von der Scan-Technik der Rasterdisplays
- Algorithmus für Linien ist grundlegend für 2D und 3D Computergraphiken
- (Linien zeichnen war einfacher bei Vektor Displays ;-))

Einordnung in die Pipeline

- Rasterisierung der Objekte in Pixel
- Ecken-Werte interpolieren (Farbe, Tiefenwert, …)



Modell Transformation

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

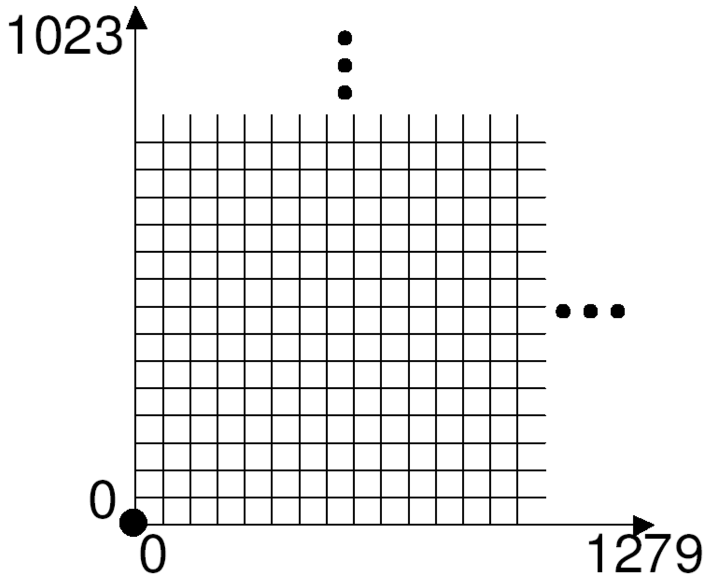
Projektion (in Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

Das Frame-Buffer-Modell

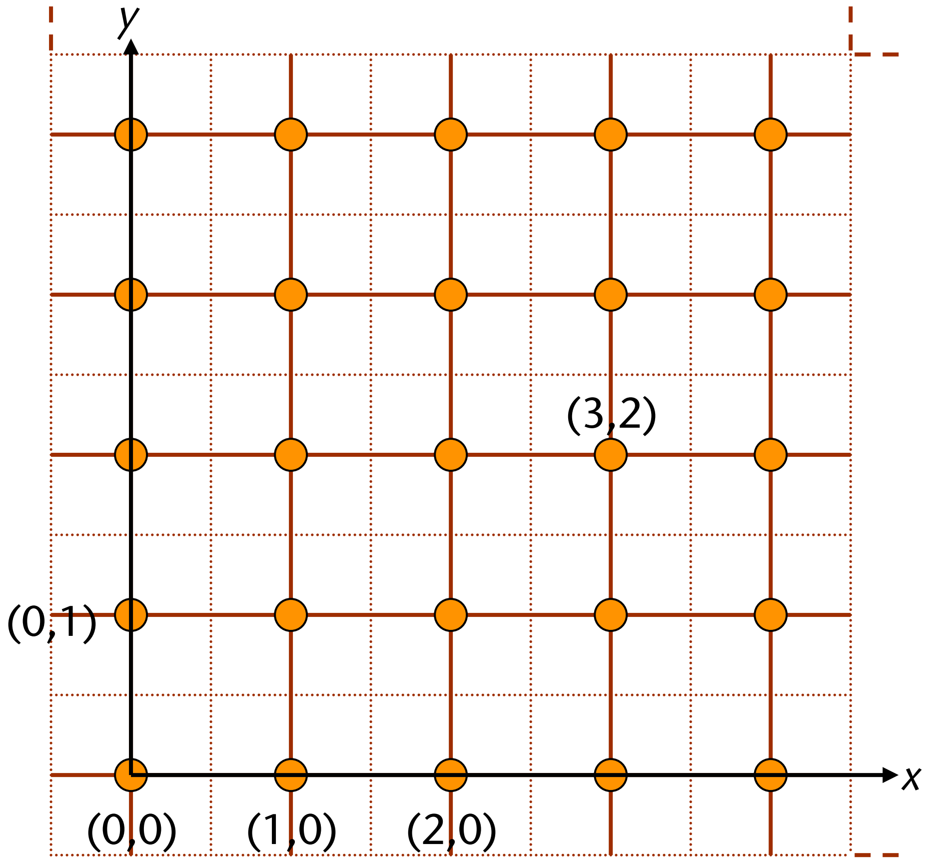
- Adressierung als 2D-Array



- Startadresse:
  - links oben (X11, Java AWT)
  - links unten (Open GL)

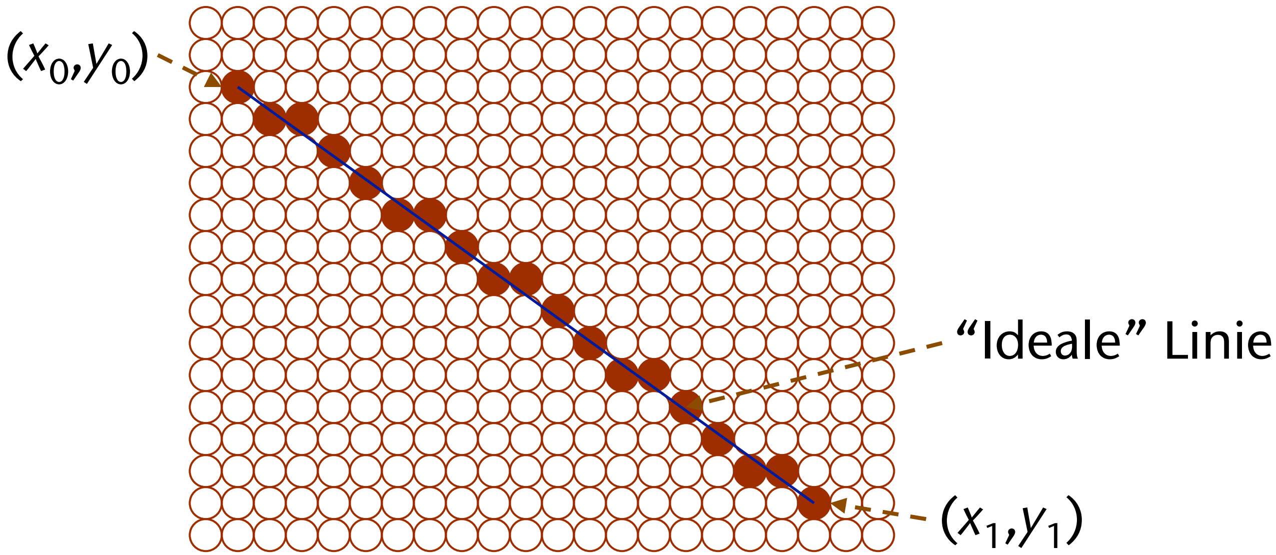
Bildschirmkoordinaten im Frame-Buffer-Modell

- Wir verwenden folgende 2D Bildschirmkoordinaten
  - Ganzzahlige Werte für *Mittelpunkte* der Pixel
  - Senkrecht = Y–Achse, Horizontal = X–Achse



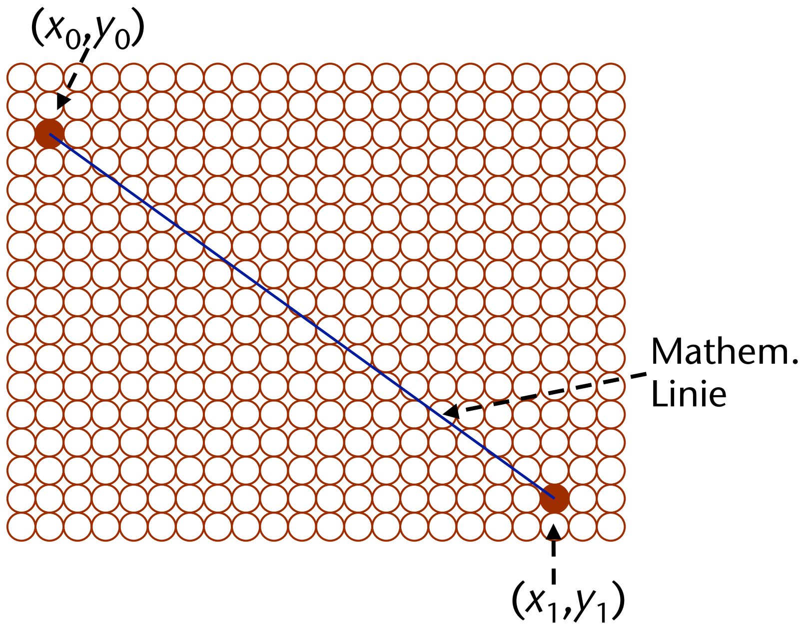
Die Ideale Linie

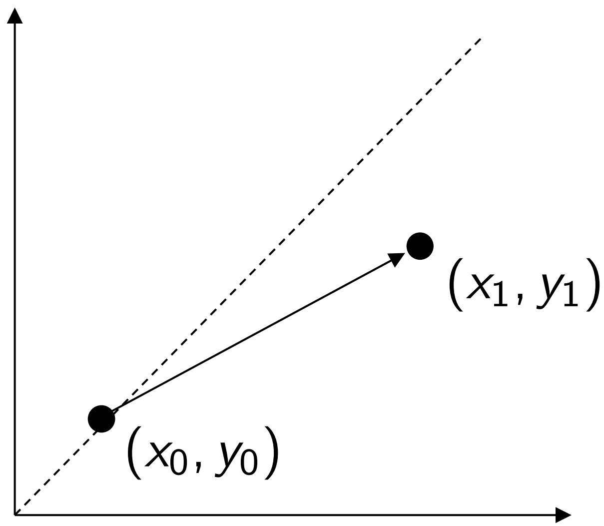
- Keine Unterbrechungen (diagonale Schritte sind erlaubt)
- Einheitliche Stärke und Helligkeit
- Fehlerfreiheit (setze nur die "nähesten" Pixel an der idealen Linie)
- Geschwindigkeit (wie schnell kann die Linie gezeichnet werden?)
- Invarianz gegenüber Zeichenrichtung



Spezifikation der Aufgabe

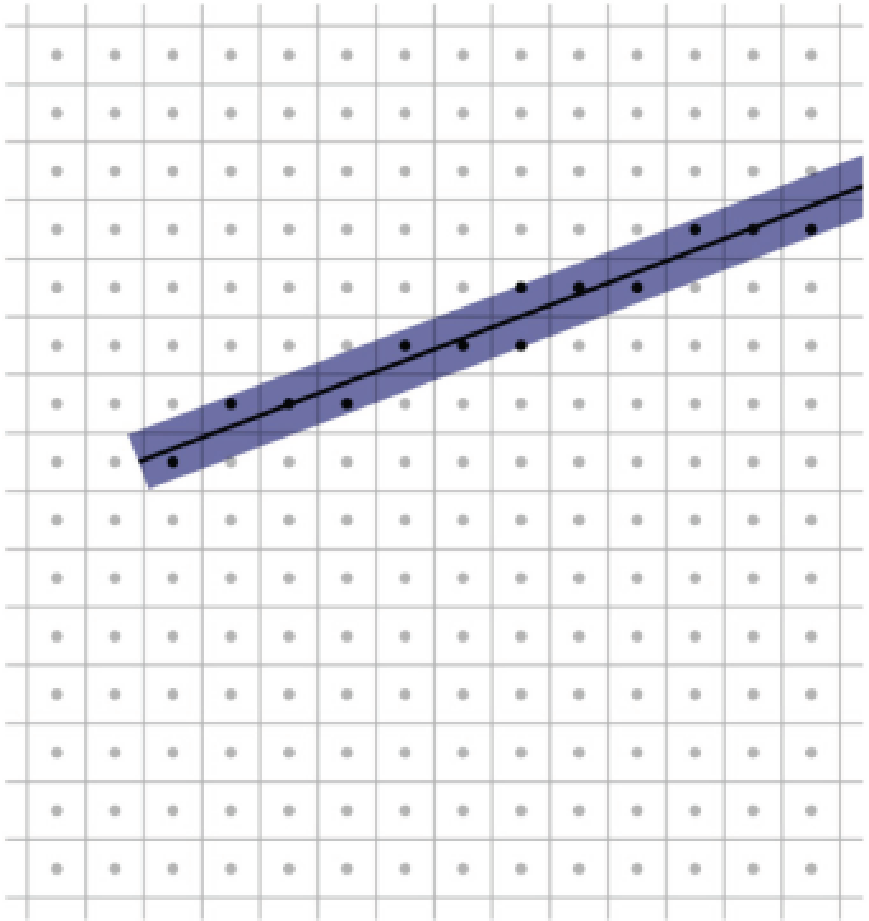
- Gegeben: Endpunktkoordinaten einer Linie
- Vereinfachungen:
  - Ganzzahlige Koordinaten
  - Geradengleichung:
    $$y = mx + b$$
  - Mit $0 \leq m \leq 1$ und $x_0 < x_1$
  - Alle übrigen Fälle bekommt man durch Vertauschen / Spiegeln
- Ausgabe: Folge von Pixeln (= rasterisierte Linie)





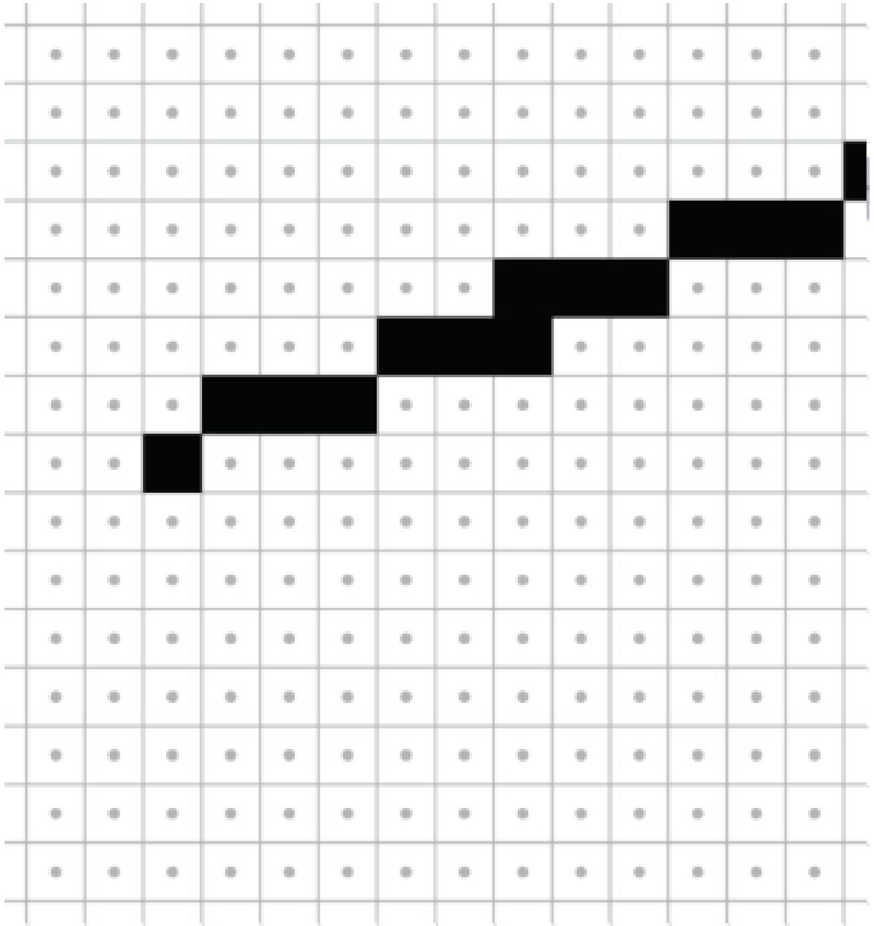
Erster Versuch der Scan-Konv. einer Linie

- Betrachte Linie als schmales Rechteck
- Zeichne alle Pixel, deren Zentrum im Inneren liegt

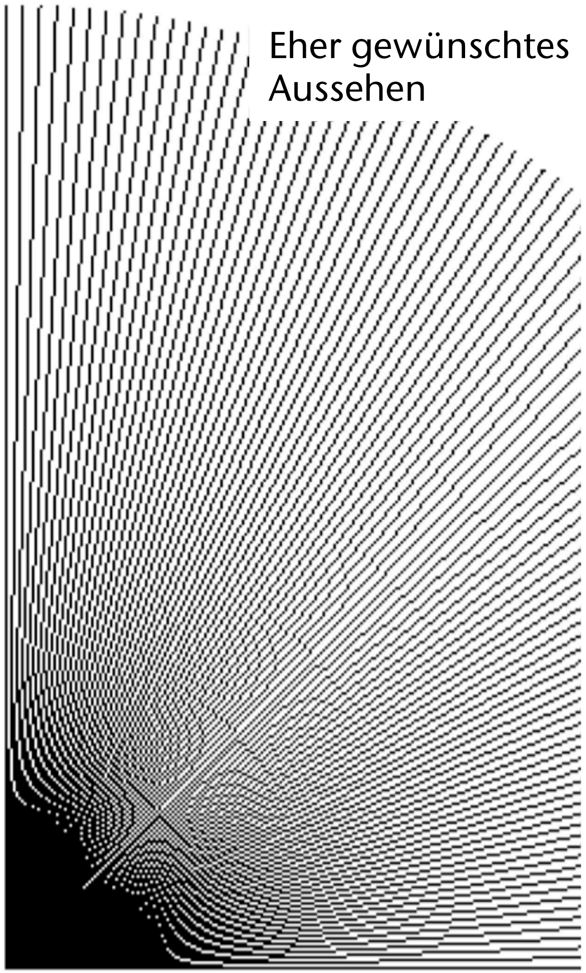


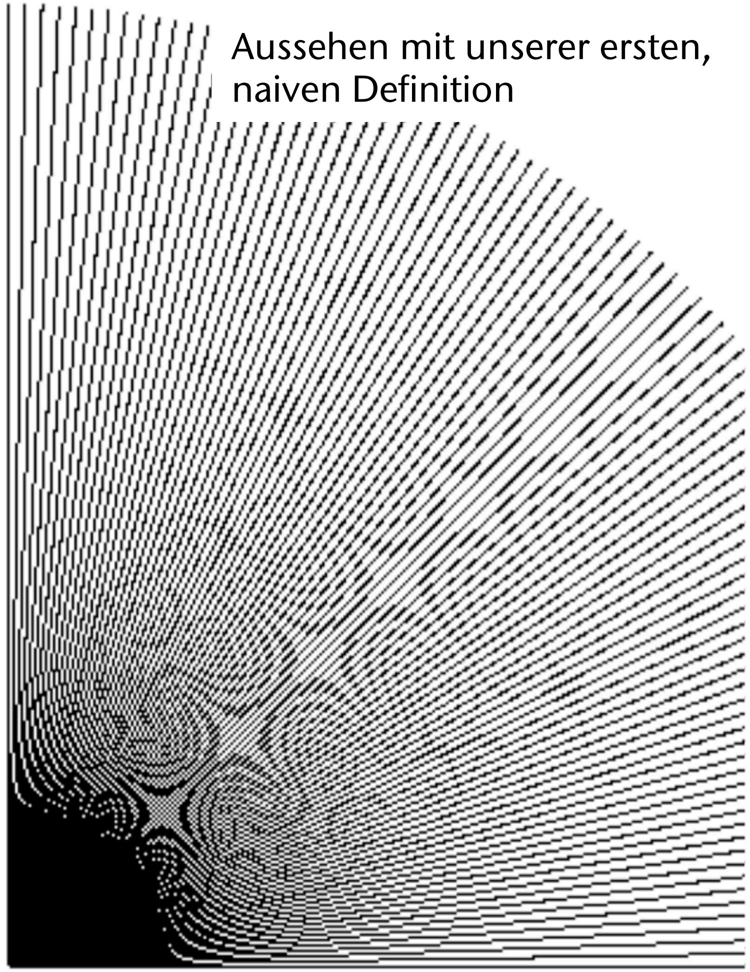
- Betrachte Linie als schmales Rechteck
- Zeichne alle Pixel, deren Zentrum im Inneren liegt

1. Problem: manchmal werden vertikal übereinander liegende Pixel gesetzt → unterschiedliche scheinbare Linienstärke



2. Problem:

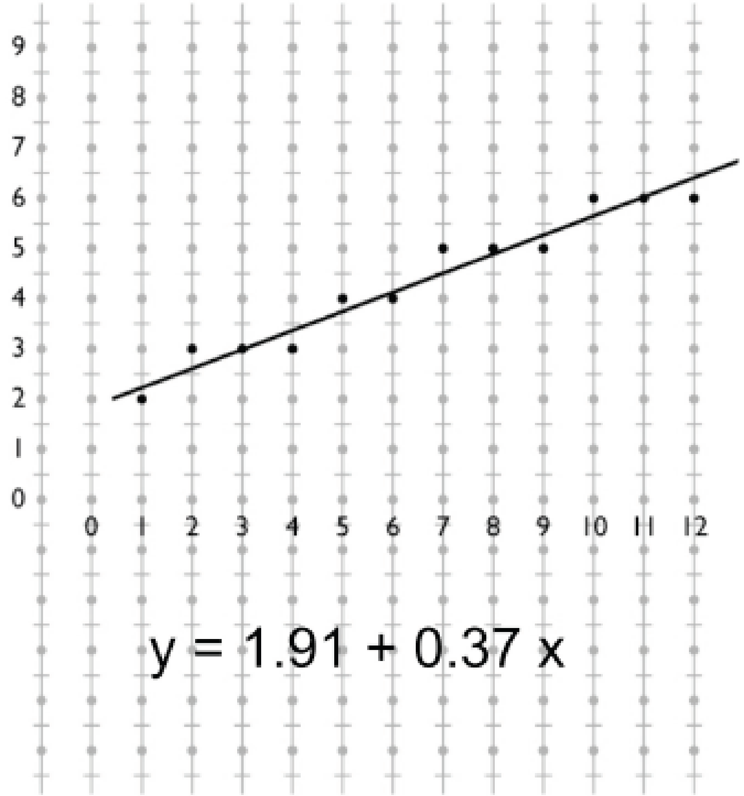

Eher gewünschtes
Aussehen


Aussehen mit unserer ersten,
naiven Definition

Erster einfacher Algorithmus

- Einfacher Algorithmus: werte Gleichung der Linie pro Spalte (pro x-Koord.) 1 Mal aus

```
for x = ceil(x0) .. floor(x1):
   y = b + m*x
   setPixel( x, round(y) )
```



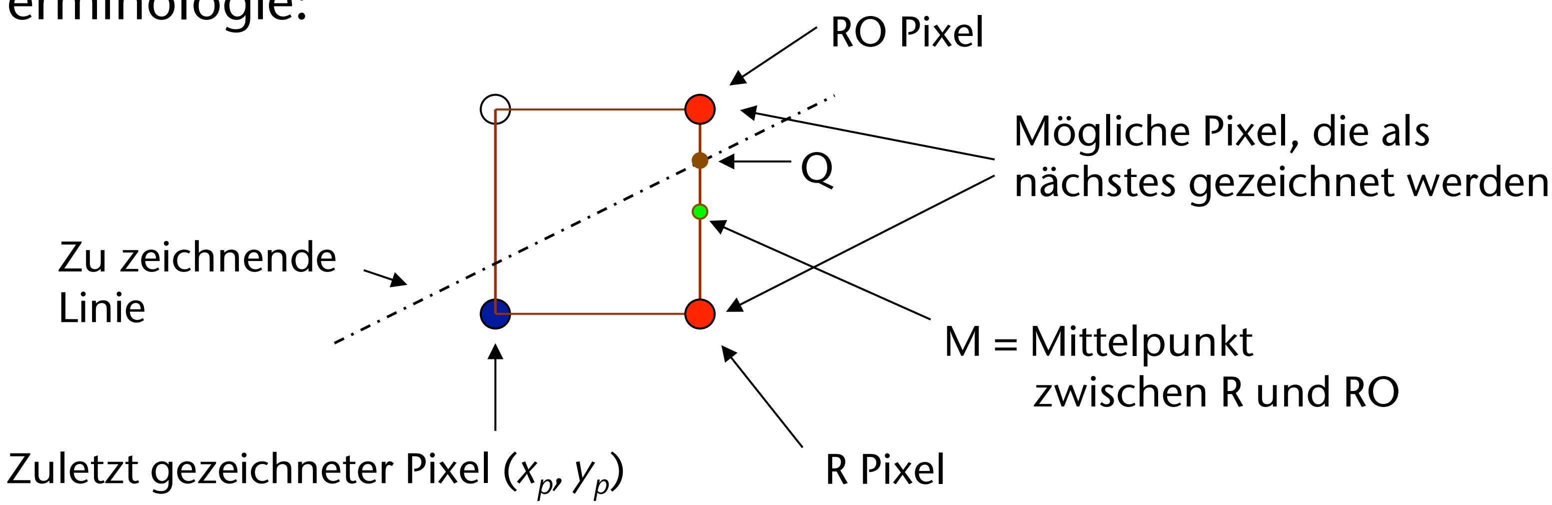
- Probleme:
  - Floating-Point,
  - Mult. und Runden sind (rel.) langsam

Enter Bresenham

Clip from Bresenham's Keynote Talk at WSCG'03

Generelles Vorgehen und Terminologie

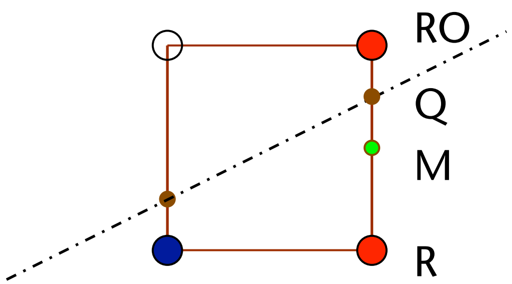
- Bei jedem X-Schritt gibt es nur zwei Möglichkeiten für die Y-Koordinate:
  - entweder bleibt die Y-Koord gleich;
  - oder die Y-Koord erhöht sich um genau 1 Pixel
- Terminologie:



  - Q = Schnittpunkt der Linie mit Gridline $x_P + 1$

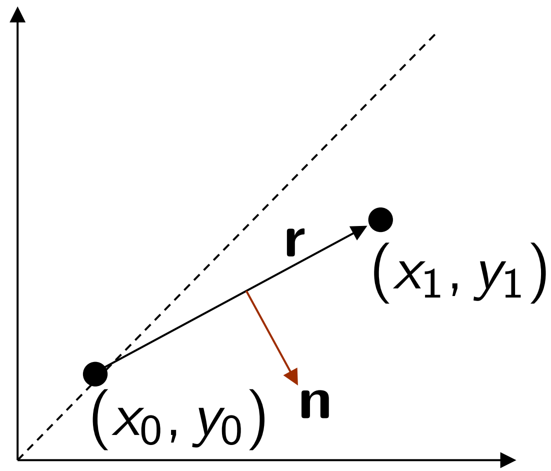
Zwei Varianten des Linien-Zeichen-Algos

- Ursprünglich "Bresenham-Algorithmus" [1962]:
  - Bestimme Distanz zwischen RO und Q und zwischen R und Q
  - Bestimme Differenz zwischen den Distanzen
  - Vorzeichen des Ergebnisses legt fest, welcher Pixel eingefärbt wird
- Heute Midpoint-Algo:
  - Bestimme, auf welcher Seite der Linie liegt Mittelpunkt M
  - M = oberhalb → färbe Pixel R
  - M = unterhalb → färbe Pixel RO



Der Midpoint-Algorithmus

- Wie kann man einfach bestimmen, auf welcher Seite der Linie man sich befindet?
- Verwende implizite Form der Linie:



$$F(x,y) := \mathbf{n} \cdot \binom{x}{y} - c = 0$$

$$\mathbf{r} = \begin{pmatrix} x_1 - x_0 \\ y_1 - y_0 \end{pmatrix} = \text{Richtungsvektor}$$

$$\mathbf{n} = \begin{pmatrix} n_1 \\ n_2 \end{pmatrix} = \begin{pmatrix} y_1 - y_0 \\ x_0 - x_1 \end{pmatrix} \quad \text{ist senkrecht dazu}$$

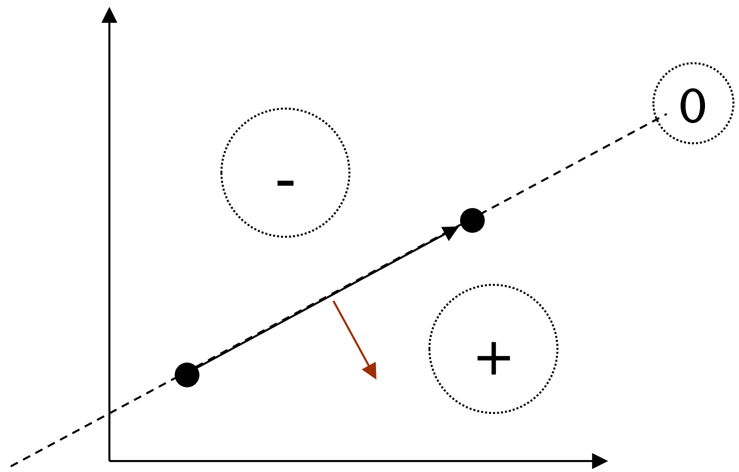
$$F(x_0, y_0) = 0 \;\text{ liefert }\; c = x_0 y_1 - y_0 x_1$$

- Gegeben sei (x, y). Dann ist

  $F(x,y) = 0$, wenn (x,y) auf der Linie liegt

  $F(x,y) < 0$, wenn (x,y) oberhalb der Linie liegt

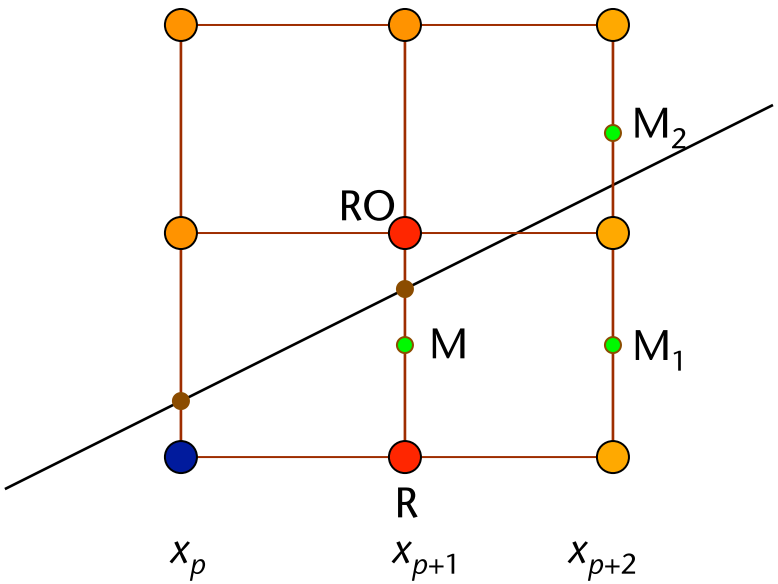
  $F(x,y) > 0$, wenn (x,y) unterhalb der Linie liegt



- Definiere "Entscheidungsvariable" *d*:

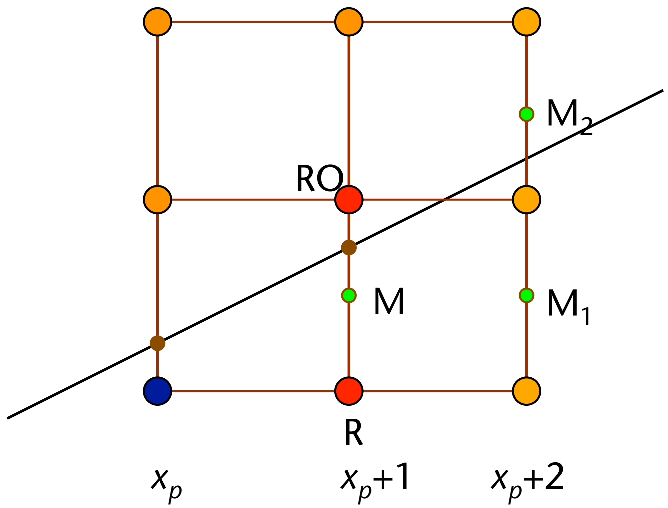
$$d = F(M) = F(x_p + 1, y_p + \tfrac{1}{2})$$

- Für den Midpoint-Algo, beachte das Vorzeichen von *d*:
  - Wenn $d > 0$, färbe RO
  - Wenn $d < 0$, färbe R
- Was ist mit dem nächsten Schritt?
- Annahme: wir haben $d = F(M)$



1. Fall: R wurde ausgewählt → nächstes M ist M_1

$$d_{old} = F(M)$$
$$= F\left(x_p + 1, y_p + \frac{1}{2}\right)$$
$$= n_1 \left(x_p + 1\right) + n_2 \left(y_p + \frac{1}{2}\right) + c$$



und

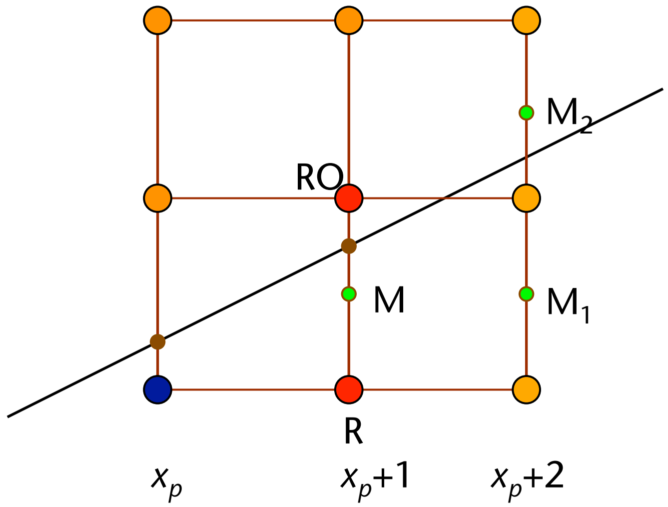
$$d_{new} = F(M_1)$$
$$= F\left(x_p + 2, y_p + \frac{1}{2}\right)$$
$$= n_1 \left(x_p + 2\right) + n_2 \left(y_p + \frac{1}{2}\right) + c$$

somit

$$d_{\text{new}} = d_{\text{old}} + n_1$$

2. Fall: RO wurde ausgewählt → nächstes M ist M_2

$$
\begin{aligned}
d_{old} &= F(M) \\
&= F\left(x_p+1, y_p+\tfrac{1}{2}\right) \\
&= n_1\left(x_p+1\right) + n_2\left(y_p+\tfrac{1}{2}\right) + c
\end{aligned}
$$



und

$$
\begin{aligned}
d_{new} &= F(M_2) \\
&= F\left(x_p+2, y_p+\tfrac{3}{2}\right) \\
&= n_1\left(x_p+2\right) + n_2\left(y_p+\tfrac{3}{2}\right) + c
\end{aligned}
$$

somit

$$
d_{\text{new}} = d_{\text{old}} + n_1 + n_2
$$

- Pseudo-Code des Midpoint-Algo:

```
berechne n1, n2, c
x, y ← x0, y0
```

$$d \leftarrow F(M) = F(x_0 + 1, y_0 + \frac{1}{2}) = n_1 + \frac{n_2}{2}$$

```
setze d1 ← n1, d2 ← n1 + n2
while x ≤ x1:
    zeichne Pixel (x,y)
    x += 1
    if d > 0:
        y += 1
        d += d2
    else:
        d += d1
```

- Achtung: in obigen Pseudo-Code ist evtl. $d = \frac{k}{2}$
  - Lösung: ...

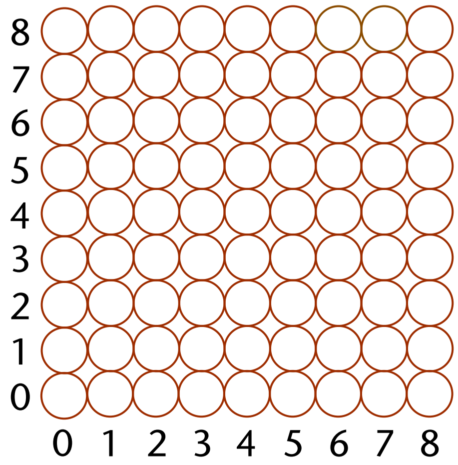
Beispiel

- Zeichne Linie von (1,2) nach (5,5)

```
x, y ← x0, y0
```

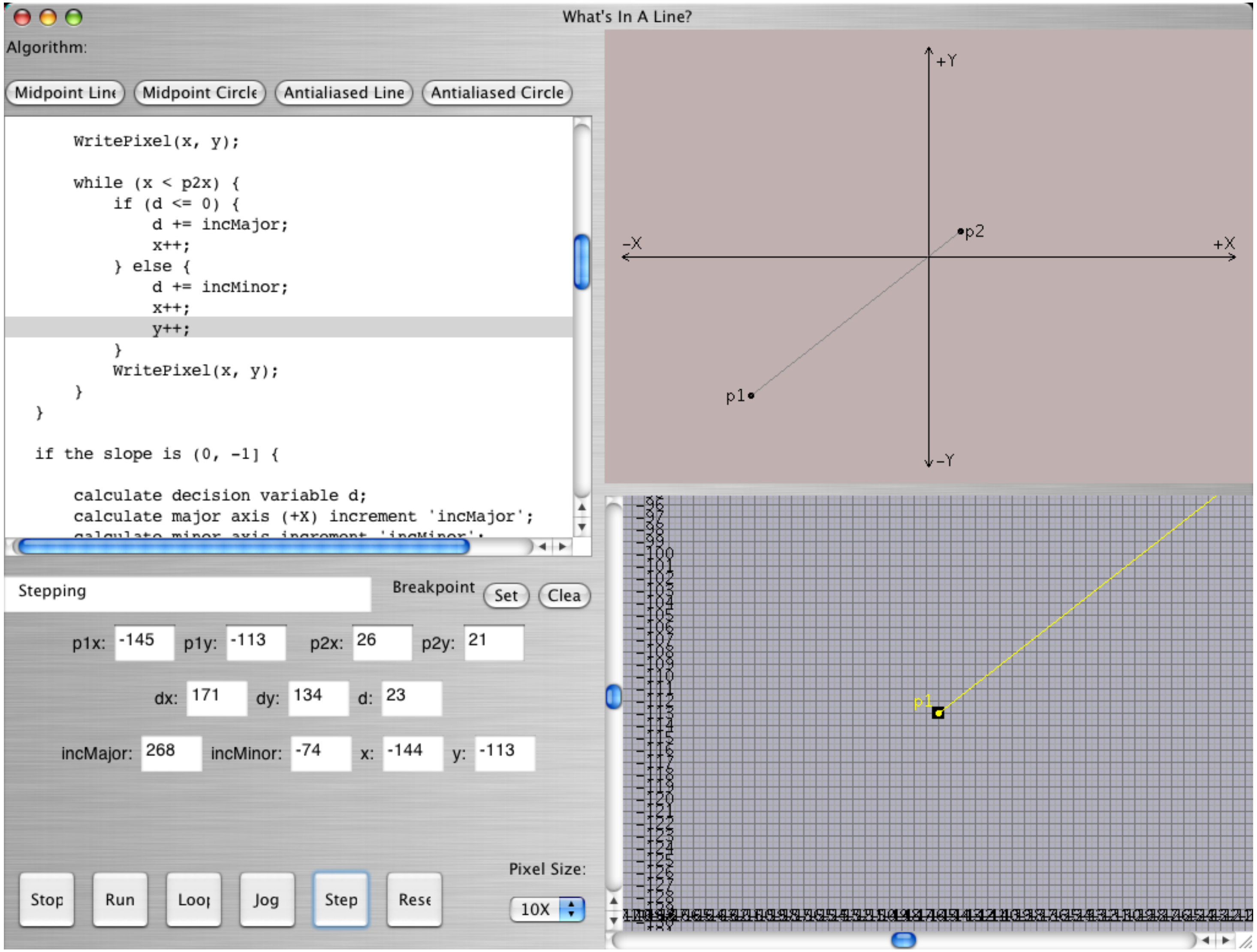
$$d \leftarrow F(M) = F(x_0 + 1, y_0 + \frac{1}{2}) = n_1 + \frac{n_2}{2}$$

```
setze d1 ← n1, d2 ← n1 + n2 while
x ≤ x1:
    zeichne Pixel (x,y)
    x += 1
    if d < 0:
        y += 1
        d += d2
    else:
        d += d1
```



n_1	n_2	d_1	d_2	d	x	y

Demo



http://www.cs.rit.edu/~ncs/whatsInALine/whatsInALine.html

Historische Randnotiz

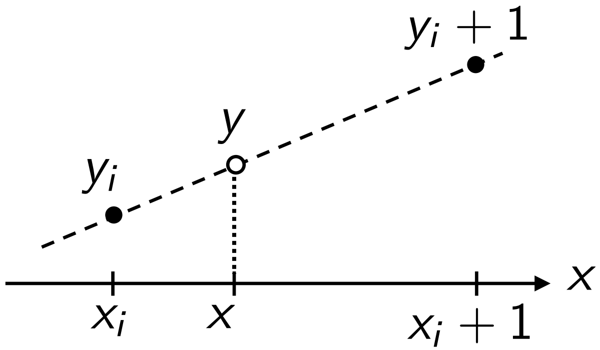
- Bekannt als DDA (digital differential analyzer)

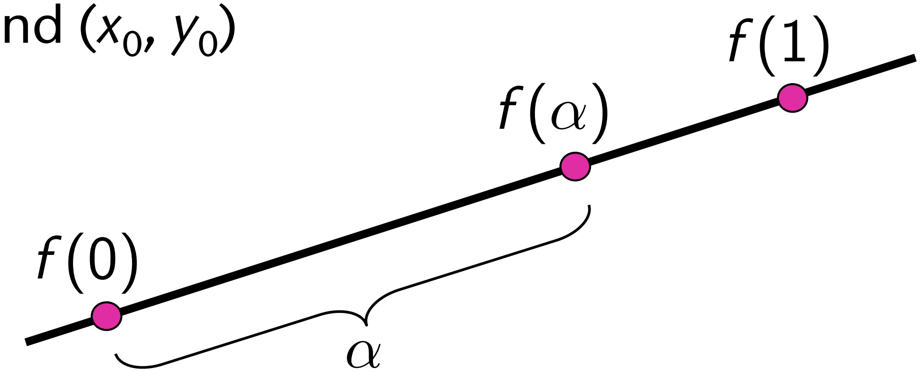
MADDIDA (Magnetic Drum Digital
Differential Analyzer, Northrop Aircraft) 1952

Interpolation von Attributen

- Häufig haben Eckpunkte weitere Attribute (außer der Pos.)
  - Z.B. verschiedene Farben
- Ziel: ein gleichmäßiger Farbverlauf entlang der Linie
- Idee: lineare Interpolation
- Im 1D: $f(x) = (1 - \alpha)y_0 + \alpha y_1$

  mit $\alpha = (x - x_0)/(x_1 - x_0)$
- Im 2D ist α gerade die normierte(!) Distanz zwischen (x, y) und (x_0, y_0)



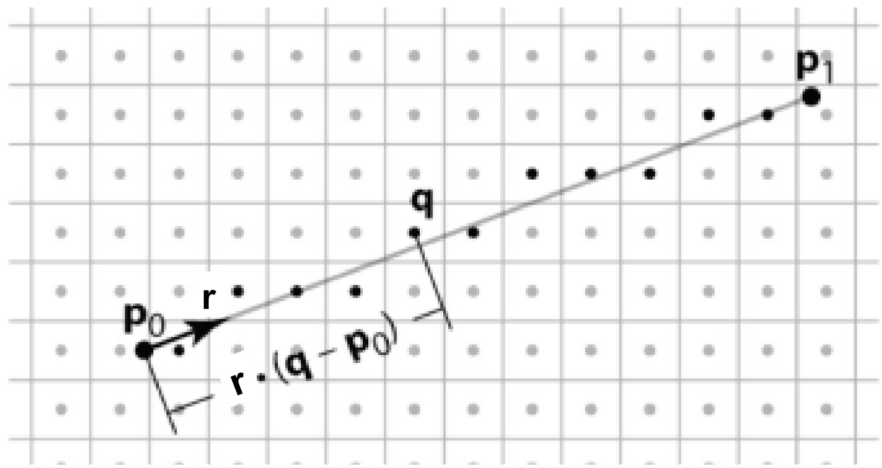


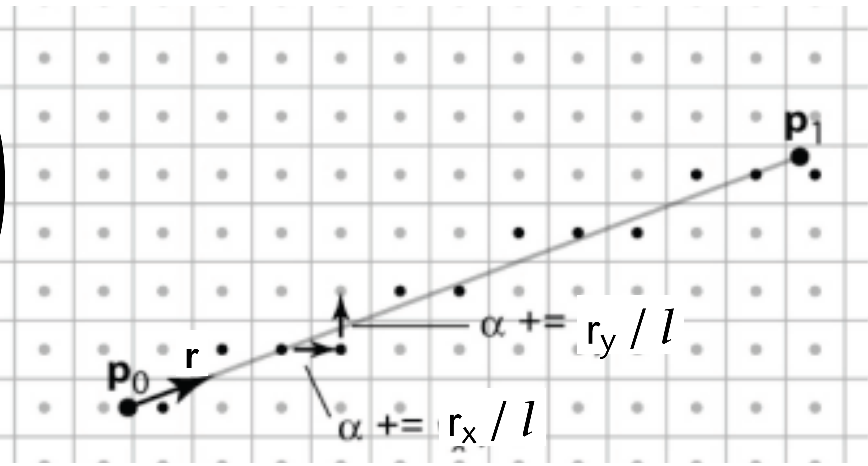
- Problem: Pixel liegen i.A. *nicht* genau auf der Linie
- Definiere 2D Funktion zur Projektion auf die Linie:

$$\alpha = \frac{\mathbf{r}\cdot(\mathbf{Q}-\mathbf{P}_0)}{l}$$

$$l = \|\mathbf{P}_1 - \mathbf{P}_0\|$$

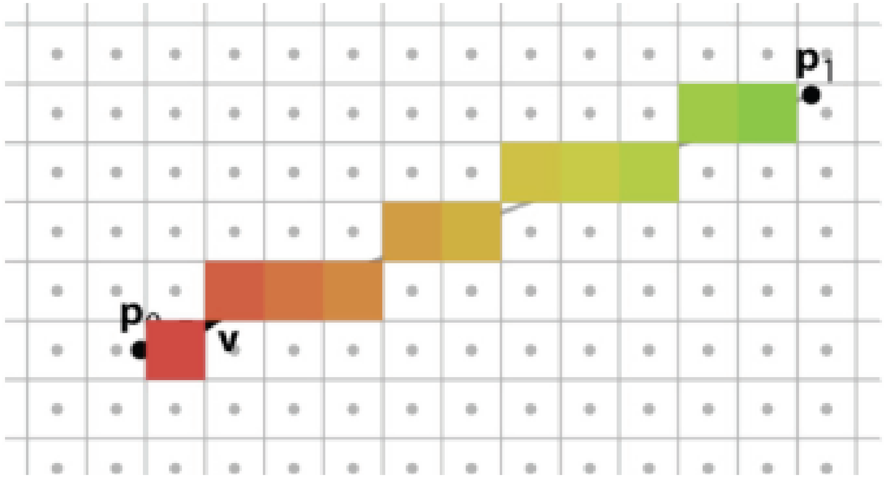
$$f(\alpha) = (1-\alpha)\begin{pmatrix} r_0 \\ g_0 \\ b_0 \end{pmatrix} + \alpha \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix}$$





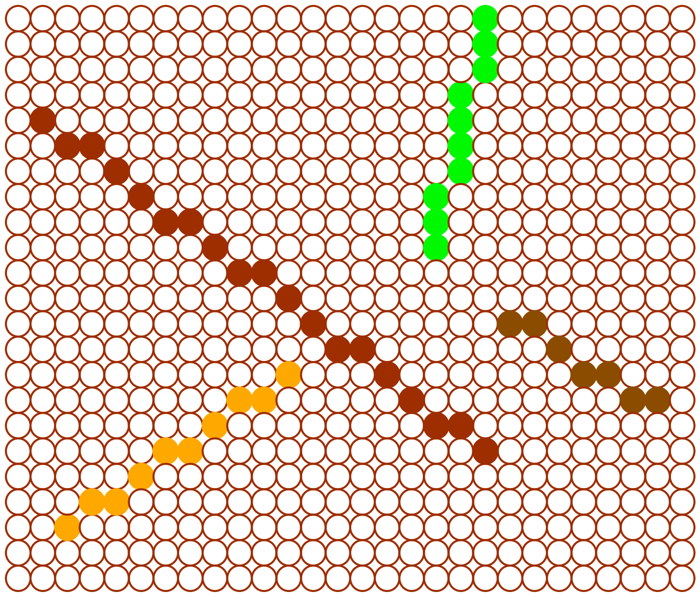
- Beobachtung: *f* ist linear in Q_x und Q_y
- Verwende DDA zur inkrementellen Berechnung von *f*

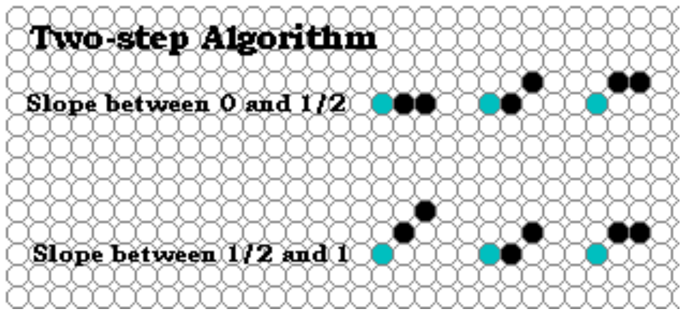
- Resultat:



Geschwindigkeitssteigerung

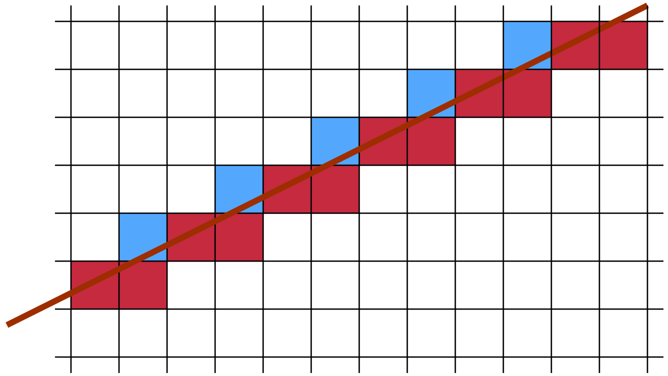
- Sind rasterisierte Linien symmetrisch?
- Abhängig von der Länge:
  - Gerade # an Pixel → ja
  - Ungerade # an Pixel →
    ja, bis auf 1 Pixel
- Idee: zeichne von beiden Seiten
  [Rokne et al., 1990]
- Man kann 2 Pixel zeichnen mit:
  - 1 Vergleich
  - 1 Update der Entscheidungsvariable d
- Weiterhin: mit 1 Test kann man die nächsten 2 Pixel entscheiden
  [Wyvill et al., 1990]





Das letzte(?) Quentchen Geschwindigkeit

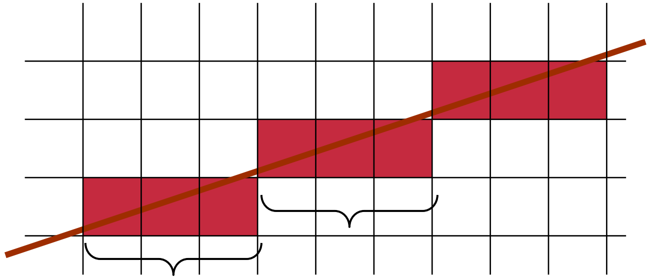
- Vereinfachungen (zunächst):
  1. Betrachte (unendliche) Linien mit $y = mx\ , \quad 0 \le m \le 1$
  2. Betrachte nur die Folge der roten Zellen = Zellen, die an ihrer linken Kante von der Linie geschnitten werden



- Terminologie:
  - Zelle wird identifiziert durch deren linken unteren Eckpunkt (x_j, y_j)
  - Span := Folge von Zellen mit gleicher y-Koord.
  - Länge des j-ten Spans = r_j

- Beobachtung: die diskrete Linie ist vollständig durch die Folge der Span-Längen definiert, denn

$$(x_{j+1}, y_{j+1}) = (x_j + r_j, y_j + 1)$$



- Satz (o. Bew.):

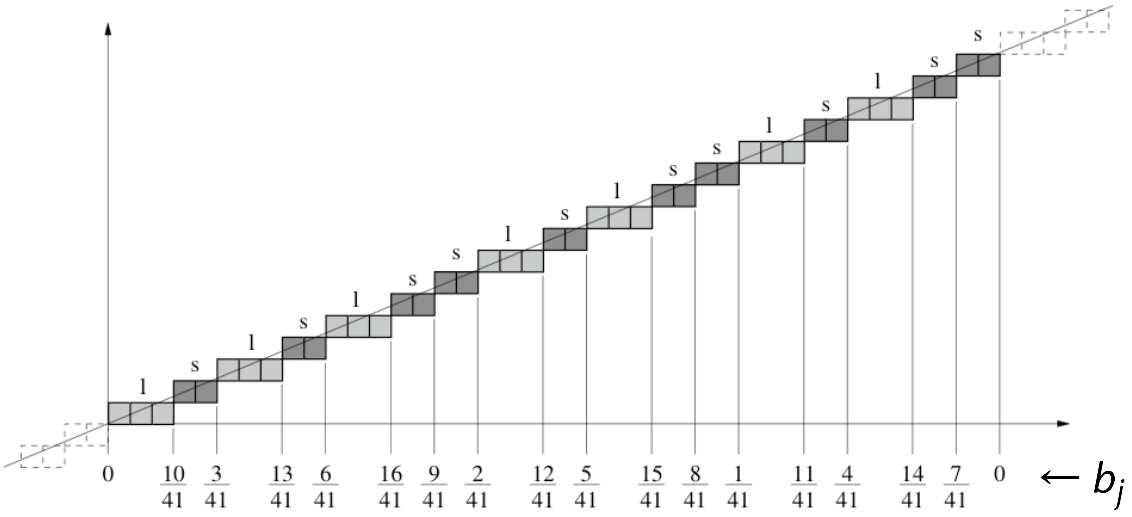
  Alle Spans der diskretisierten Linie haben nur eine von höchstens zwei verschiedenen Längen, nämlich

$$\forall j : \; r_j = r \;\vee\; r_j = r + 1$$

- Klar ist:

$$\frac{1}{2} \leq m \leq 1 \;\Rightarrow\; r = 1$$

- Beispiel:



- Beobachtung: wenn wir ein seeehr langes Segment der Linie betrachten, dann gilt

$$\frac{\#\ \text{Spans}}{\#\ \text{Zellen}} = \frac{\Delta y}{\Delta x} \approx m$$

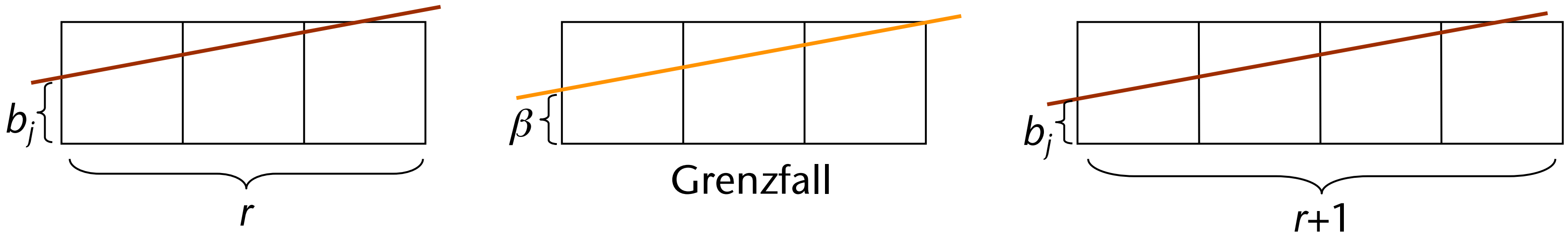
- Folge: aus der Steigung kann man die Span-Länge r (bzw. r+1) berechnen:

$$\frac{1}{m} = \text{ mittlere Span-Länge}$$
$$= \text{Mittelwert von } r \text{ und } r+1 \;\Rightarrow$$

$$r = \left\lfloor \frac{1}{m} \right\rfloor \;,\; r+1 = \left\lceil \frac{1}{m} \right\rceil$$

- Im Folgenden: Berechnung von r_j , m.a.W., Methode zur Entscheidung, ob man einen "langen Span" oder einen "kurzen Span" hat

- Wovon hängt es ab, ob man einen langen / kurzen Span hat?



- Fazit: falls $b_j \geq \beta$, dann kurzer Span, sonst langer Span
- Bestimmung von β: $\quad b_j = mx_j - y_j$

$$b_{j+1} - b_j = mr_j - 1$$

Im Grenzfall ist $b_{j+1} = 0$ und $b_j = \beta$, also

$$\beta = 1 - mr = 1 - m \left\lfloor \frac{1}{m} \right\rfloor$$

- Das nächste b_{j+1} ist also:

  falls kurzer Span → $b_{j+1} = b_j - \beta$

  falls langer Span → $b_{j+1} = b_j + m - \beta$

- Damit hat man einen iterativen, sehr effizienten Algo zur Aufzählung aller Zellen, die von einer Linie getroffen werden
- Weiteres (lästiges) Detail:
  - Bei einem Strahl ist der erste Span i.A. gekürzt
  - Soll hier nicht weiter vertieft werden

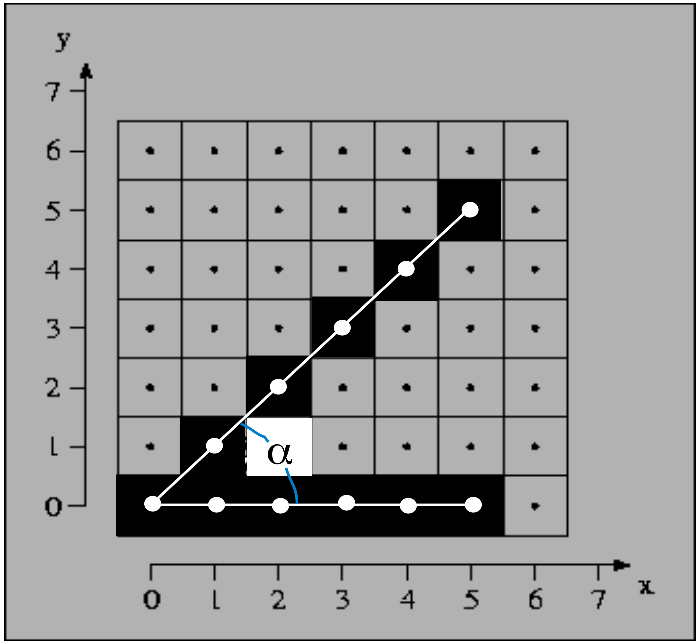
Speedup gegenüber einfachem DDA

- Komplexität:
  O(n) bei DDA (z.B. Midpoint),
  O(n/r) mit der Span-basierten Methode,
  n = Anzahl Zellen auf dem Strahl, r = mittlere Span-Länge
- In Zahlen:
  - Ca. Faktor 2 schneller über alle mögliche Orientierungen des Strahls

Weitere Überlegungen

- Gewünscht: einheitliche Stärke und Helligkeit
- Bei gleicher Pixelzahl sind schräge Linien länger als horizontale
- Ändere Intensität der Linie gemäß der Steigung
- Skaliere den Grauwert um den Faktor

$$\cos(45^\circ - \alpha)\,, \quad \alpha = 0^\circ \dots 45^\circ$$



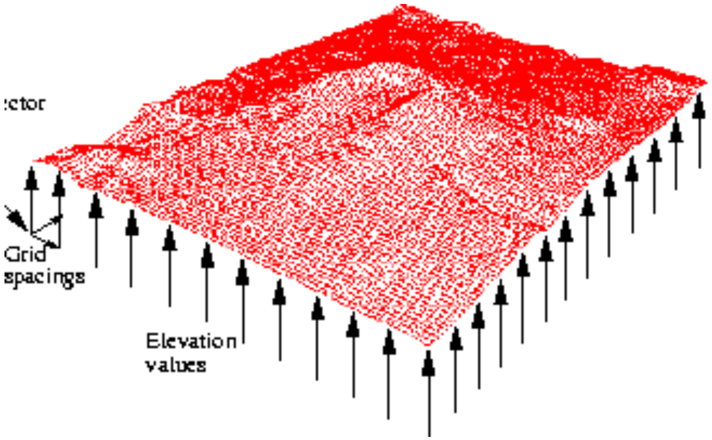
- Was ist bei gemusterten Linien? (gestrichelte Linie, etc.)

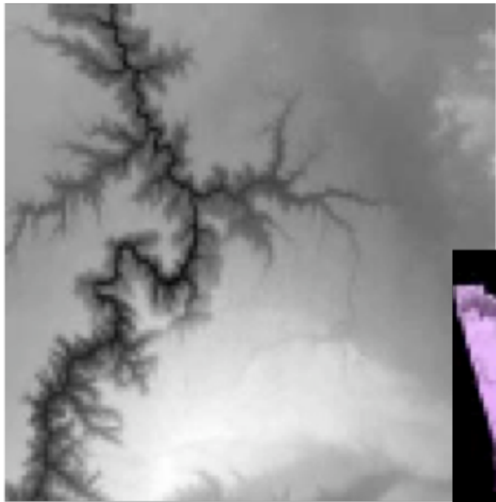
Beispiel für eine Anwendung der DDA-Technik: Ray-Tracing von Height Fields [Henning & Stephenson, 2004]

- Height Field = Alle Arten von Flächen, die sich als Funktion

$$z = f(x, y)$$

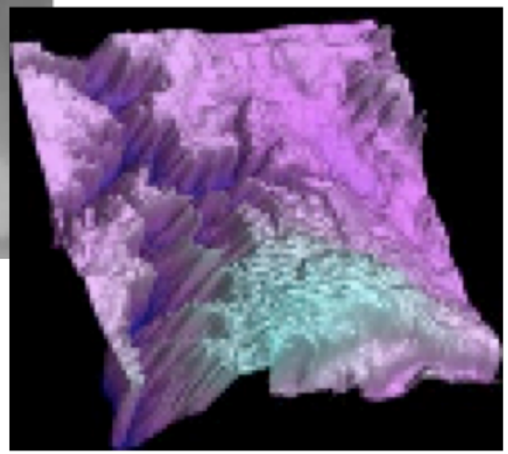
  schreiben lassen
  - Z.B.: Terrain, Meßwerte über einer Ebene, 2D-Skalarfeld, ...





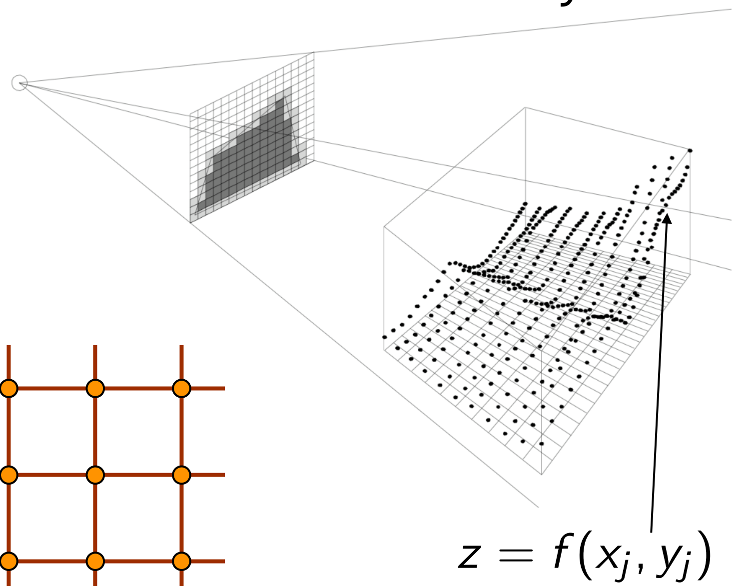
Height field
(= Bitmap)

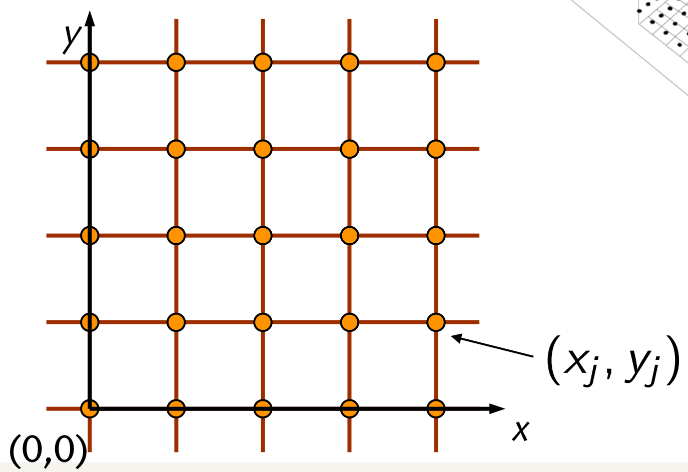
Rendered



Situation

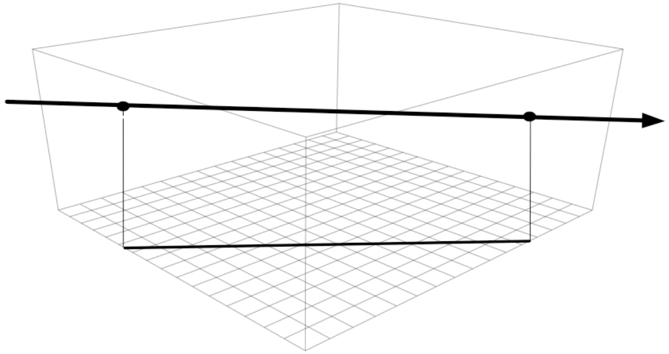
- Die naive Methode, ein Height-Field zu raytracen:
  - Konvertiere das n×n Feld in $2n^2$ Dreiecke, teste Strahl gegen jedes
  - Probleme: langsam, benötigt viel Speicher
- Ziel: direktes Ray-Tracing des Height-Fields aus dem 2D-Array
- Gegeben:
  - Strahl
  - Feld [0…n]×[0…n] als Float-Array
  - Höhenwerte liegen auf den Gitterknoten vor



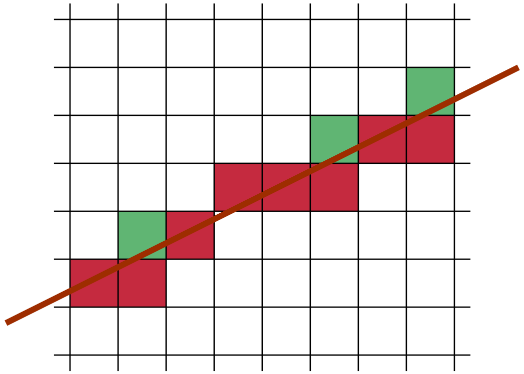


Algorithmus

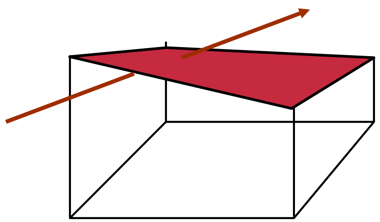
1. Dimensionsreduktion
   - Projiziere Strahl in xy-Ebene



2. Alle Zellen der Reihe nach besuchen, die vom Strahl geschnitten werden (und nur diese)
   - Ähnlich zu Scan-Conversion, aber mit zusätzlichen Zellen

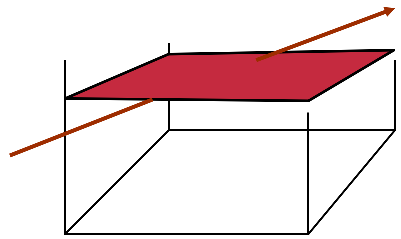


3. Strahl testen gegen das Flächenstück, das von den 4 Höhenwerten an den Ecken aufgespannt wird

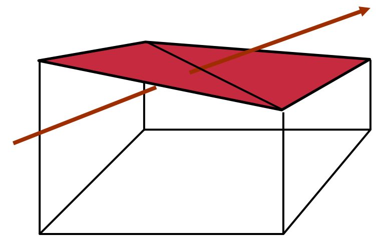


Schnitttest Strahl—Flächenstück in der Zelle

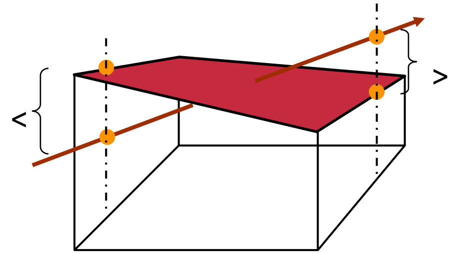
- Naive Methoden:
  - "Nearest-Neighbor":
    - Bestimme mittlere Höhe aus den 4 Höhenwerten an den Ecken
    - Schneide Strahl gegen horizontales Quadrat mit dieser mittleren Höhe
    - Sehr ungenau



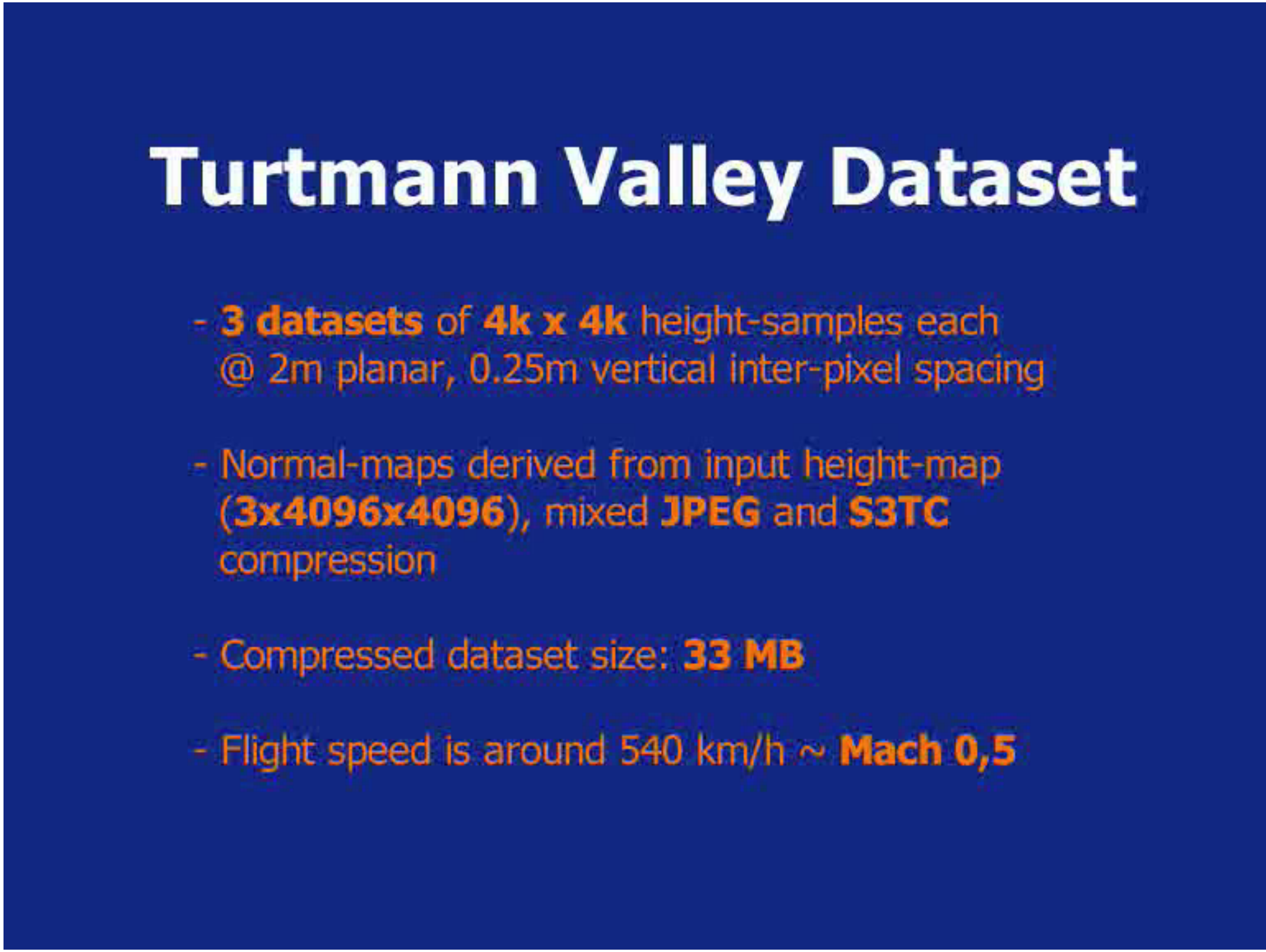
  - "2 Dreiecke":
    - Konstruiere 2 Dreiecke aus den 4 Punkten über den Ecken
    - Knick innerhalb der Zelle, Aufteilung in Dreiecke nicht eindeutig



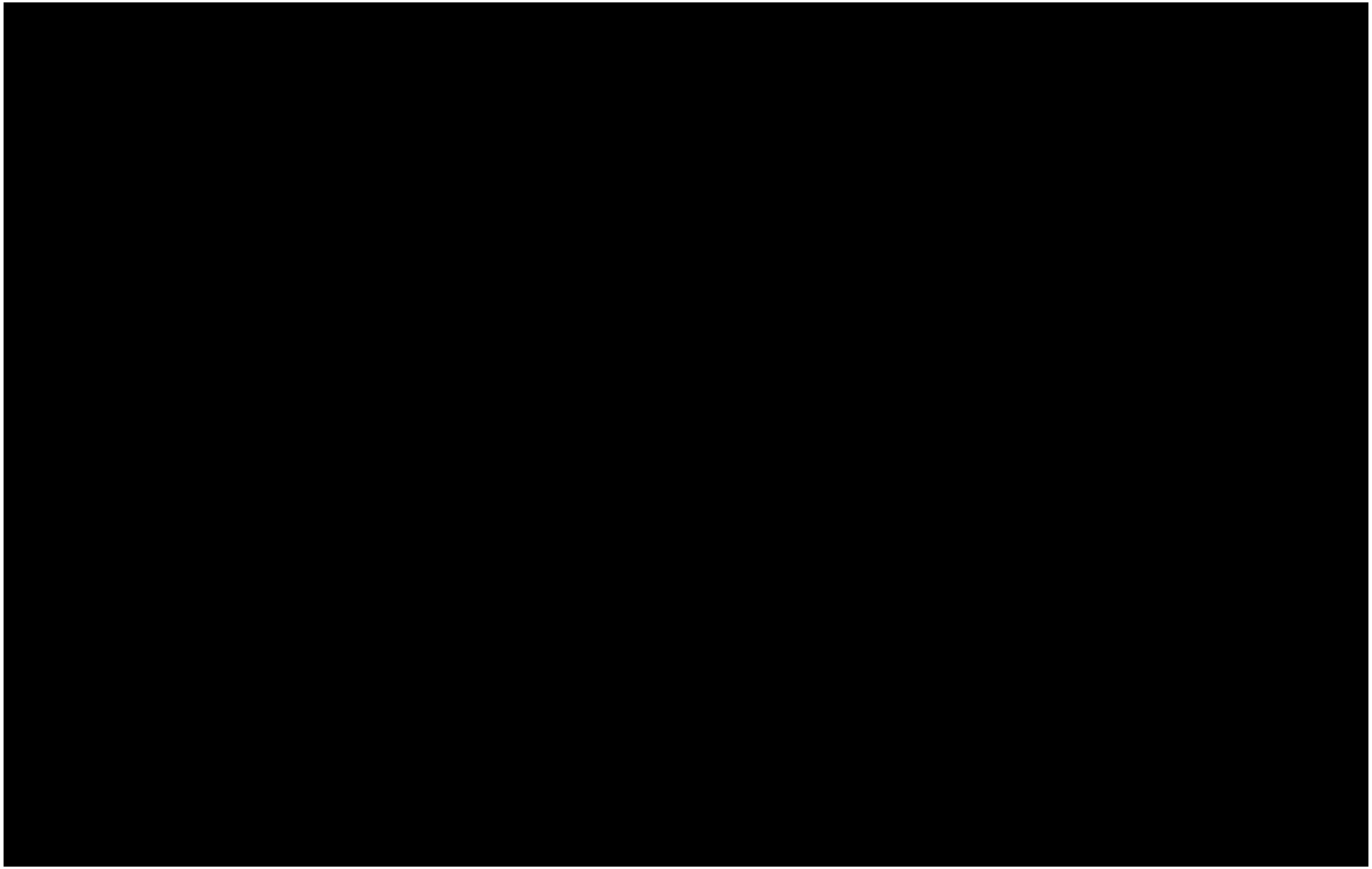
- Besser: "bilineare Interpolation"
  - Betrachte Fläche als parabolisches Hyberboloid
  - Bestimme Höhe am Rand über/unter dem Strahl durch lineare Interpolation
  - Vergleiche Vorzeichen
  - Bestimmt ggf. Schnittpunkt & Normale



Beispiele für Terrain



Bonn University



Valles Marineris, Mars - http://mars.jpl.nasa.gov

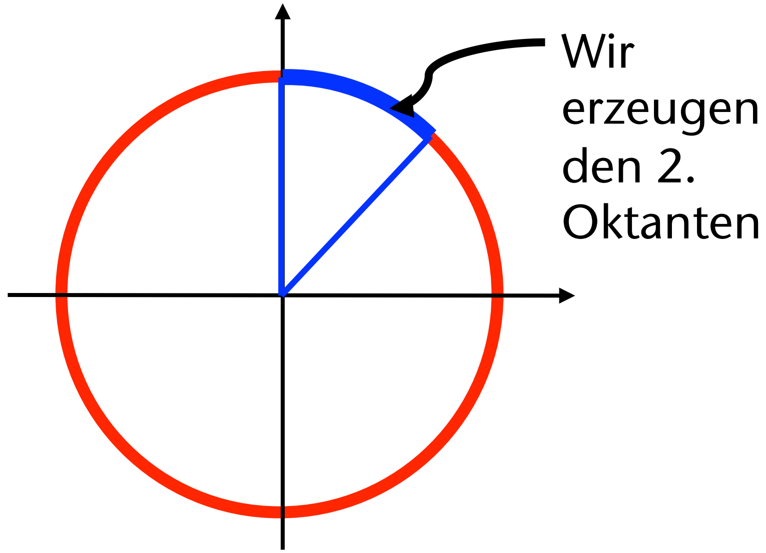
Scan-Konvertierung von Kreisen

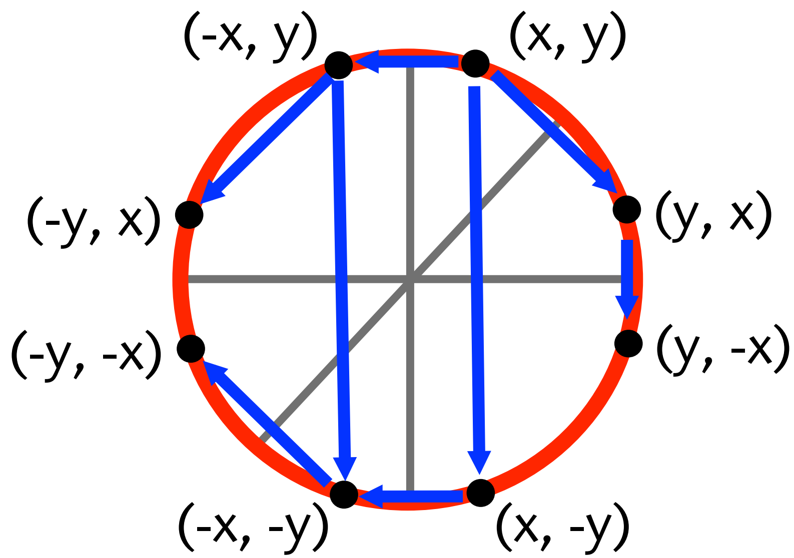
Nutze 8-Punkt-Symmetrie aus

- Berechne nur 1/8 eines Kreises mit $0 \leq x \leq \frac{r}{\sqrt{2}}$

```
drawCirclePoint(x,y):
    drawPixel(x,y)
    drawPixel(y,x)
    drawPixel(y,-x)
    :
    :
```

- Außerdem: oBdA ist Mittelpunkt = Ursprung





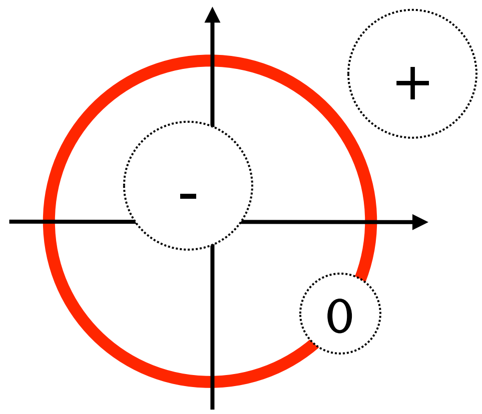
Implizite Kreisgleichungen

- Definiere $F(x,y) = x^2 + y^2 - r^2$

  Für (x,y) auf dem Kreis: F(x,y) = 0

  falls F(x,y) > 0 $\Rightarrow$ (x,y) außerhalb

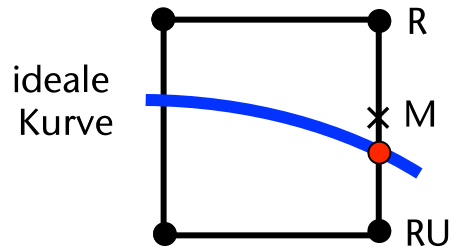
  und F(x,y) < 0 $\Rightarrow$ (x,y) innerhalb

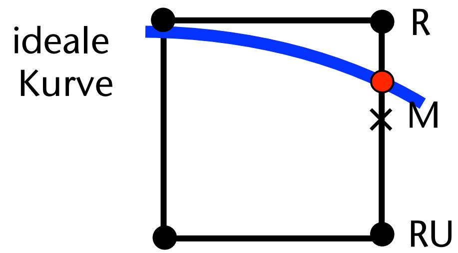


- In jedem x-Schritt: wähle R oder RU

  also: $F(M) \geq 0 \Rightarrow$ RU

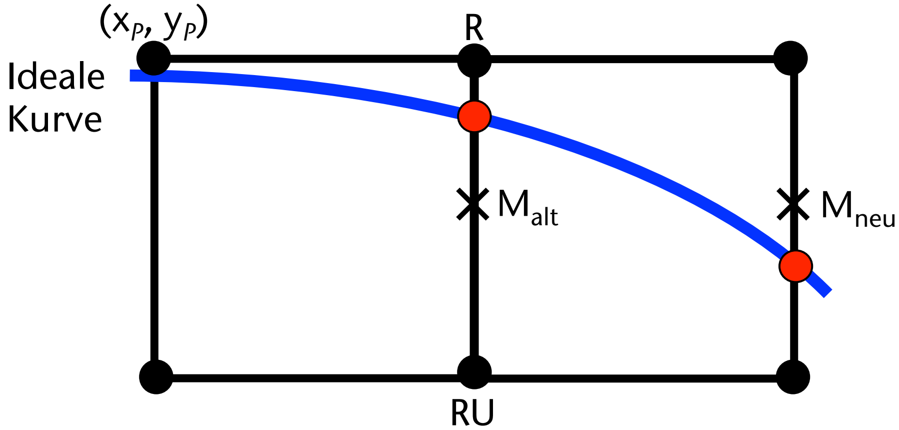
  und: $F(M) < 0 \Rightarrow$ R





- Definiere wieder eine Entscheidungsvariable $d = F(M)$

1. Fall: $d_{alt} < 0 \rightarrow R$



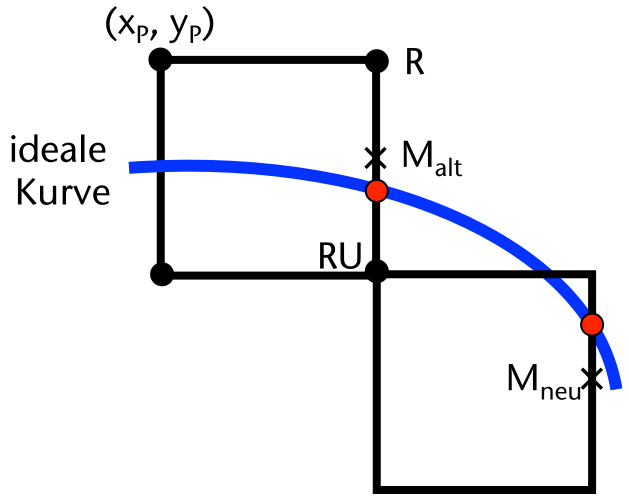
$$d_{alt} = F(x_P + 1, y_P - \tfrac{1}{2}) = (x_P + 1)^2 + \left(y_P - \tfrac{1}{2}\right)^2 - r^2$$

$$d_{neu} = F(x_P + 2, y_P - \tfrac{1}{2}) = (x_P + 2)^2 + \left(y_P - \tfrac{1}{2}\right)^2 - r^2$$

$$d_{neu} = d_{alt} + (2x_P + 3)$$

$$d_{\text{neu}} = d_{\text{alt}} + \Delta_R, \quad \text{mit } \Delta_R = 2x_P + 3$$

2. Fall: $d_{alt} \geq 0 \rightarrow$ RU



$$d_{neu} = F(x_P + 2, y_P - \tfrac{3}{2}) = (x_P + 2)^2 + \left(y_P - \tfrac{3}{2}\right)^2 - r^2$$

$$d_{neu} = d_{alt} + (2x_P - 2y_P + 5)$$

$$d_{\text{neu}} = d_{\text{alt}} + \Delta_{RU}, \quad \text{mit } \Delta_{RU} = 2x_P - 2y_P + 5$$

- Kleines Problem: Δ_{RU} und Δ_R hängen von x_P und y_P ab!
- Idee: Bilde inkremente Differenzen 2-ter Ordnung
- Aktualisiere in beiden Fällen jeweils Δ_{RU} und Δ_R :

Fall R	Fall RU
$\begin{aligned} \Delta_R(x,y) &= 2x+3 \\ \Delta_R(x+1,y) &= 2(x+1)+3 \\ &= \Delta_R(x,y)+2 \end{aligned}$	$\begin{aligned} \Delta_R(x,y) &= 2x+3 \\ \Delta_R(x+1,y-1) &= 2(x+1)+3 \\ &= \Delta_R(x,y)+2 \end{aligned}$
$\begin{aligned} \Delta_{RU}(x,y) &= 2x-2y+5 \\ \Delta_{RU}(x+1,y) &= 2(x+1)-2y+5 \\ &= \Delta_{RU}(x,y)+2 \end{aligned}$	$\begin{aligned} \Delta_{RU}(x,y) &= 2x-2y+5 \\ \Delta_{RU}(x+1,y-1) &= 2(x+1)-2(y-1)+5 \\ &= \Delta_{RU}(x,y)+4 \end{aligned}$