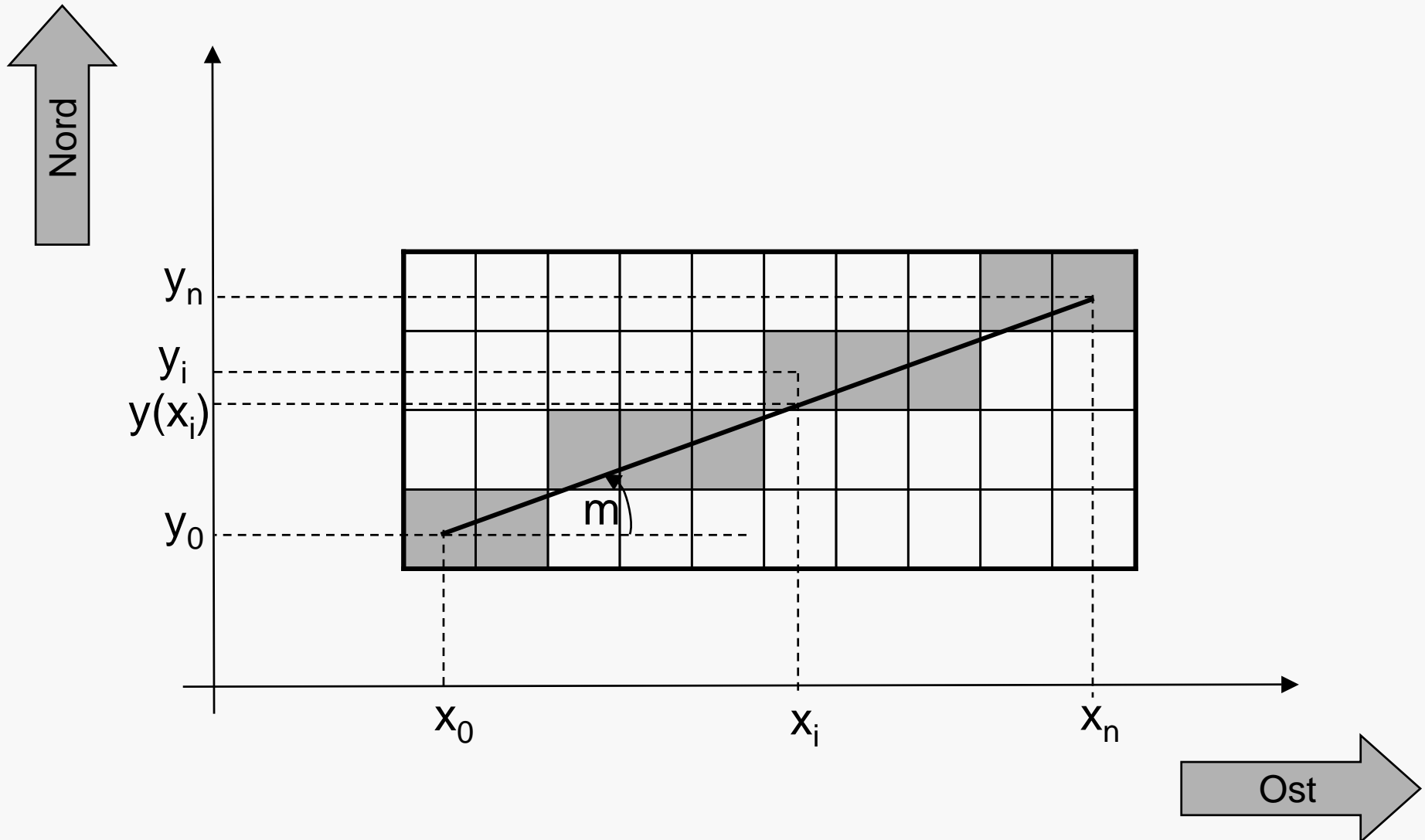
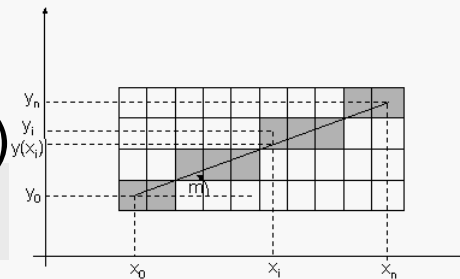


Gerasterte Linien



(n+1) Punkte einer gerasterten Geraden zw. (x_0, y_0) u. (x_n, y_n)
– hier verwendete Notation:

x_i : Abszisse des i-ten Rasterpunktes ($x_i \in \mathbf{N}_0$)
 $x_{i+1} = x_i + 1 \Rightarrow x_{i+j} = x_i + j$



$y(x_i)$: zu x_i , gehörige, mathem. exakte Ordinate ($y(x_i) \in \mathbf{R}$)

y_i : zu x_i gewählte („optimale“) Raster-Ordinate ($y_i \in \mathbf{N}_0$)

Annahme: $y(x_0) = y_0$; $y(x_n) = y_n$

$\Delta x = x_n - x_0$; $\Delta y = y(x_n) - y(x_0) = y_n - y_0$

Geradensteigung: $m = \Delta y / \Delta x = [y(x_i) - y_0] / [x_i - x_0] \quad i=1, \dots, n$

Berechnung der Ordinaten ($\in \mathbf{R}$) aus Pixelgrößen ($\in \mathbf{N}_0$):

$$y(x_i) = m \cdot (x_i - x_0) + y_0$$

Folgepixel: $y(x_{i+1}) = y(x_i + 1) = m \cdot (x_i + 1 - x_0) + y_0$

Gerasterte Linien

$$\begin{aligned}y(x_{i+1}) - y(x_i) &= m (x_i + 1 - x_0 - (x_i - x_0)) + y_0 - y_0 \\ &= m\end{aligned}$$

$$\begin{aligned}\Rightarrow y(x_{i+1}) &= y(x_i) + m \\ &= y_0 + (i+1) \cdot m\end{aligned}\quad \text{Gl.(A)}$$

```
void BruteForceLine (int x0,int y0,int xn,int yn)
{
    int    x=x0;
    float  y=(float)y0, m=(float)(yn-y0)/(xn-x0);

    for (x=x0; x<=xn; x++,y+=m)
        setPixel (x, (int)y);

    return;
}
```

$t \text{ nsec} / \text{floatOp} * f \text{ floatOps} / \text{Umrißpixel}$
 $* p \text{ Umrißpixel} / \text{Dreieck} * d \text{ Dreiecke} / \text{Szene}$
ergeben, falls: $t=10$; $f=1$; $p=100$; $d=100.000$
eine (unnötige) Wartezeit von 0,1 sec / Szene !

Wunsch: Von Pixel-Koordinaten (gerundet, $\in \mathbf{N}_0$) mit Ganzzahl-Arithmetik auf Folgepixel schließen

Gerasterte Linien

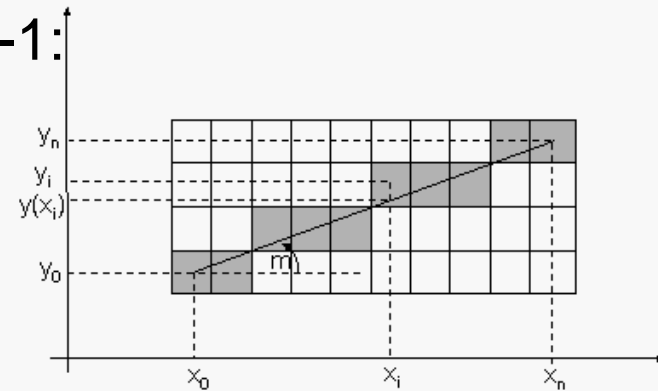
Hinzunahme von Folgepixel (i+1), i=1,...,n-1:

$$\begin{aligned} \text{Gl.(A)} \quad \Rightarrow \quad y(x_{i+1}) &= y(x_0+i+1) \\ &= y_0 + (i+1) \cdot \Delta y / \Delta x \end{aligned}$$

Weiter nach Osten ($y_{i+1} = y_i$),

wenn: $y_0 + (i+1) \cdot \Delta y / \Delta x < y_i + \frac{1}{2}$

– sonst weiter nach Nordosten ($y_{i+1} = y_i + 1$)



[y_i ist (ganzzahlig)
nicht berechenbar!]

Ost, wenn: $(i+1) \cdot \Delta y / \Delta x < y_i - y_0 + \frac{1}{2}$

N := Anzahl bisheriger Nordost-Entscheidungen

$$= y_i - y_0$$

E := Anzahl bisheriger Ost-Entscheidungen

[N, E: nicht berechenbar – aber zählbar !]

$$N + E = i$$

(Wdhlg.): Ost, d.h. $y_{i+1} = y_i$, wenn

$$(i+1) \cdot \Delta y / \Delta x < y_i - y_0 + \frac{1}{2}$$

$$(N+E+1) \cdot \Delta y / \Delta x < N + \frac{1}{2}$$

$$2 \cdot N \cdot \Delta y + 2 \cdot E \cdot \Delta y + 2 \cdot \Delta y < 2 \cdot N \cdot \Delta x + \Delta x$$

$$N \cdot 2 \cdot (\Delta y - \Delta x) + E \cdot 2 \cdot \Delta y + 2 \cdot \Delta y - \Delta x < 0$$

Zur Erinnerung: Bei der ersten Entscheidung $\Leftrightarrow i = N = E = 0$

„Ost-Bedingung“: $2\Delta y - \Delta x < 0$

Bresenham's Strategie: Einrichtung einer „decision variable“ **d**, die mit **$2 \cdot \Delta y - \Delta x$** initialisiert und nach jeder Folge-Entscheidung um **$2\Delta y$** (Ost) bzw. **$2 \cdot (\Delta y - \Delta x)$** (Nordost) erhöht wird.

```
void MidpointLine (int x0,int y0, int xn,int yn)
{ int dx, dy, d;
  int incrE, incrNE;      /*Increments for E & NE*/
  int x=x0, y=y0;         /*Start & current pixel*/

  dx=xn-x0; dy=yn-y0; d=2*dy-dx;
  incrE=2*dy; incrNE=2*(dy-dx); /*Increments */

  for (x=x0; x < xn; x++) /*last pixel omitted!*/
  { SetPixel (x, y);
    if (d < 0) { d += incrE; } /*choose E*/
    else      { d += incrNE; y++; } /*choose NE*/
  }
  return;
}
```

Herleitung für 1. Oktanten (o.B.d.A.): $m \leq 1$

⇒ Anpassung unter Nutzung von Symmetrien! (Übung!)

Besonderheiten / Vorzüge des Bresenham-Algorithmus

(„Bresenham’s algorithm for scan-converting straight lines“):

- Nutzung der Tatsache, daß Pixel diskret sind:

$$x_{i+1} = x_i + 1 \Rightarrow x_{i+j} = x_i + j \Rightarrow y(x_j) = y_0 + j \cdot m$$

- Verknüpfung algorithmischer Vorgänge (Entscheidungen) mit Variablen-Werten:

$$N = y_i - y_0 ; N + E = i$$

- Entscheidung über Hinzunahme von Folgepixel anhand des zuletzt gesetzten Pixels (ohne Geraden-Steigung m)
- Ganzzahlige Addition und Subtraktion
- Je Linie nur 3 ganzzahlige Multiplikationen mit 2 (Hw!)
- Erweiterbar auf Kreise und Ellipsen (in obiger Form)
- Repetierbar u. (grundsätzlich) in der Richtung umkehrbar (Löschung mit Hintergrund-Farbe)
- Läßt in obiger Version den letzten Punkt aus (Polygone: letzter Punkt = erster \Rightarrow Flimmern, Löcher im Papier)

Übung: Implementierung des Bresenham-Algorithmus für alle Oktanten mit ASCII-Zeichen als Bildpunkten:

- (i) Erweiterung des Algorithmus
- (ii) Feststellung / Sicherstellung der Umkehrbarkeit

Zu allen Übungen:

- *.obj-Dateien größtenteils mitgeliefert (pro Fehler-Lokalisierung)
- Aufgaben mit großen Anteilen zur Wiederverwendung (contra Altlast-Ansammlung)

BresenTable.exe

