Allgemeines

- Inhalt der Computergrafik (Grafikverarbeitung, GDV):

  Bildsynthese: Generierung visueller Darstellungen unter Verwendung logisch-mathematischer Methoden

  – z.B. Kuchen-Diagramm mit Wahl-Ergebnissen

  – z.B. fotorealistische Filmtricks

  Gegenstück: **Bildanalyse** (Behandlung bestehender Bilder)
  – Bildverarbeitung

- Fragestellungen und Verfahren der Bildanalyse und -synthese großteils gleich, Grenzen unscharf

  (vgl. autom.Skizzen-Erstellung zu archäologischen Funden)

- Eingangs-/ Ausgangsgröße als Unterscheidungskriterium:

Top-Down-Prozeß

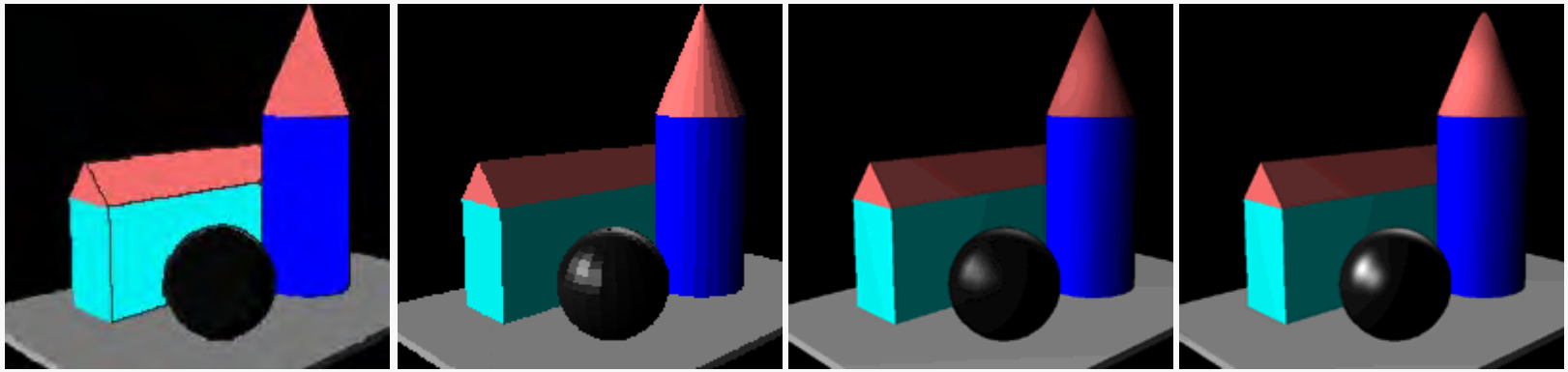
Grafik:	Beschreibung	⇨ Bild
Bildverarbeitung:	Bild	⇨ Beschreibg., Folgerg., ..
Bildbearbeitung:	Bild	⇨ „besser geeignetes“ Bild

Bottom-Up-Prozeß

BOUNCEac.exe

Lineare / Linien-Algorithmen

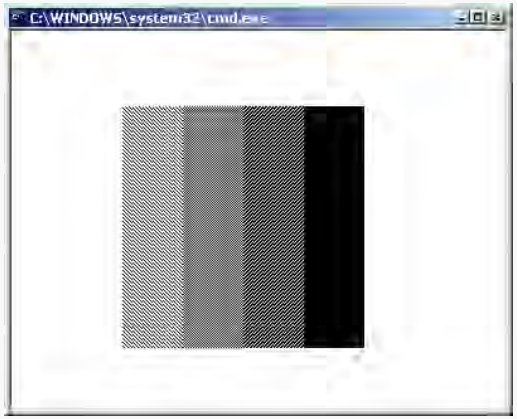
Ziel vieler CG-Anwendungen: Fotorealismus in Echtzeit; wichtiges Mittel: stetige Helligkeits- / Farbübergänge – z.B.:

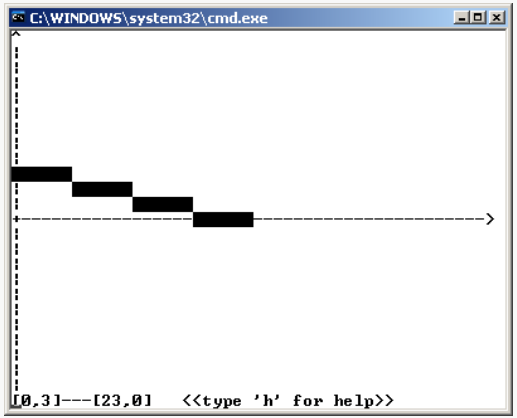


Bilder: www.glossar.de/glossar/z_shading.htm

Meistangewandte math. Methode: lineare Interpolation

Zuordnung von Werten einer abhängigen Var. $y \in \mathbf{N}_0$ (Pixelfarbe, -helligkeit) Werten einer unabhängigen Variablen $x \in \mathbf{N}_0$ (Ort).

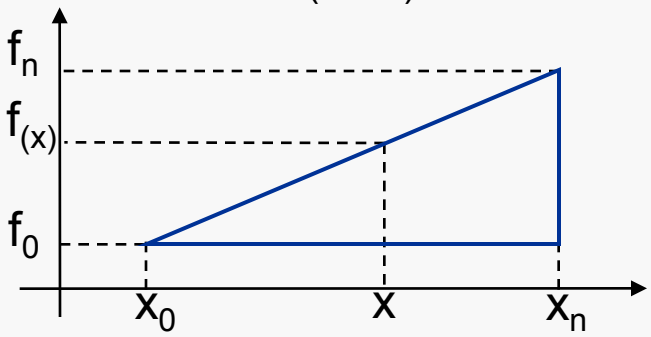




Lineare / Linien-Algorithmen

Zur Erinnerung: Lineare Interpolation im Wertebereich zwischen Anfangswert $f_0 = f_{(x=x0)}$ und Endwert $f_n = f_{(x=xn)}$ über den Definitionsbereich $x_0 \ldots x_n$

(typischer Funktionsname: **`lerp()`**)



$$(f_{(x)} - f_0) : (x - x_0) = (f_n - f_0) : (x_n - x_0)$$

$$f_{(x)} = x \bullet (f_n - f_0) / (x_n - x_0) + f_0 - x_0 \bullet (f_n - f_0) / (x_n - x_0)$$

$$= x \bullet m + b \quad \text{(m=const.; b=const.)}$$

$$f_{(x+1)} = (x+1) \bullet (f_n - f_0) / (x_n - x_0) + f_0 - x_0 \bullet (f_n - f_0) / (x_n - x_0)$$

$$= f_{(x)} + (f_n - f_0) / (x_n - x_0)$$

$$= f_{(x)} + m$$

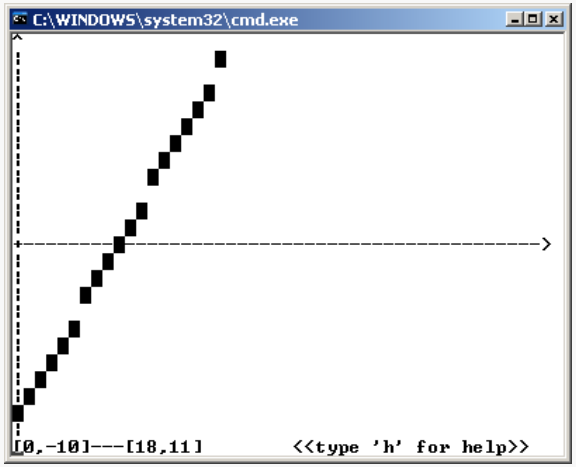
`float`/ `int`-Umwandlung: `BruteForce`-Variante

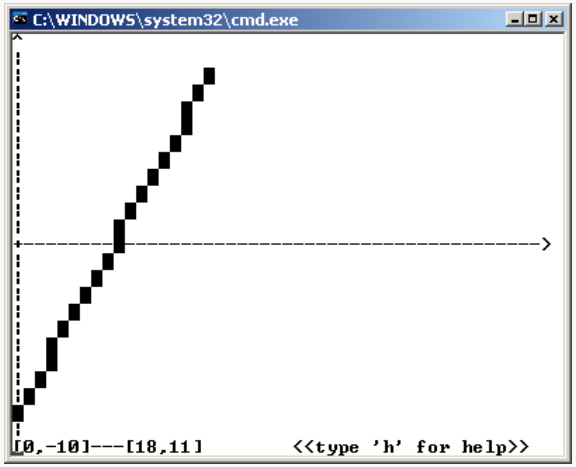
- Wunsch: Vermeidung (langsamer) Floating-Point- durch Einsatz geeigneter Ganzzahl-Arithmetik (vgl. Linien-Algorith.)

Lineare / Linien-Algorithmen

Unterscheidung zwischen Algorithmus zur Linienziehung und zur linearen Interpolation bei diskreten Systemen:

- Interpolation von (x_0,y_0) nach (x_n,y_n) ordnet (linear) jedem x-Wert im Definitionsbereich $x_0 \le x \le x_n$ genau einen y-Wert zu.
- Linie von (x_0,y_0) nach (x_n,y_n) ordnet (linear) jedem x-Wert im Definitionsbereich $x_0 \le x \le x_n$ max. so viele y-Werte zu, daß jedem y-Wert $y_0 \le y \le y_n$ ein x-Wert zugeordnet ist.

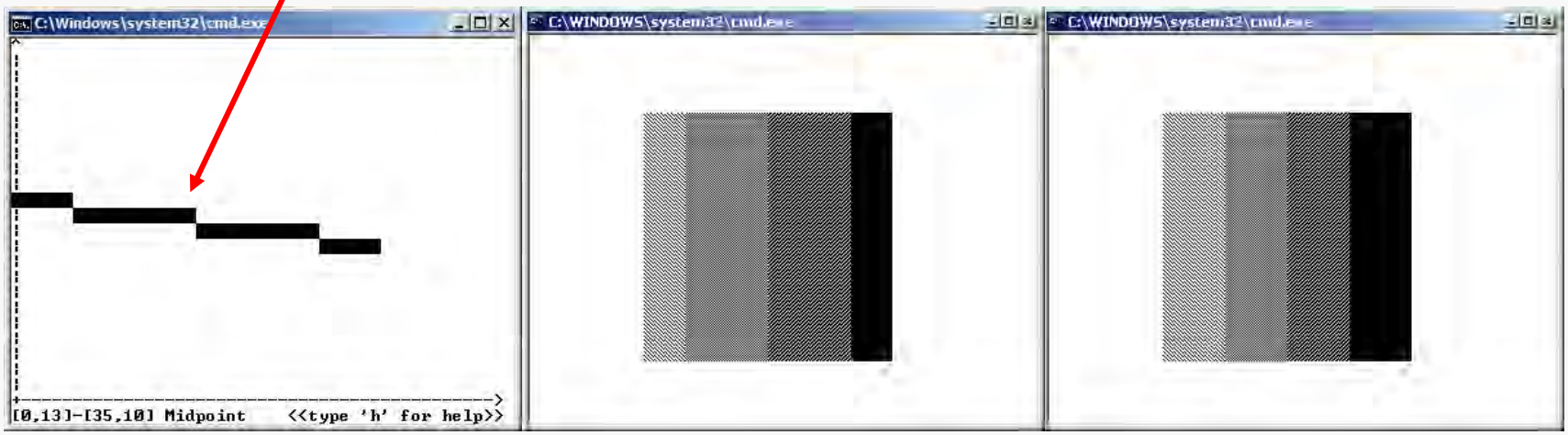




⇨ Bei Anpassung des Bresenham-Algorithmus: geänderte Behandlung des 2. und des 7. Oktanten; außerdem:


Lineare / Linien-Algorithmen

- Weiterer Anpassungsbedarf des „Midpoint Algorithm“: „Plateau“-Bildung im mittleren Teil der Bresenham-Linie:



⇨ Gegeben: (n+1) Werte $x_0,...,x_n$ einer unabhängigen Variablen $x \in \mathbf{N}_0$ mit $x_{i+j}=x_i+j$ ($i,j \in \mathbf{N}_0$, $0 \le i+j \le n$ ⇨ $x_n=x_0+n$); zudem: x_0 u. x_n zugeordnete Werte y_0 u. y_n einer abhängigen Variablen $y \in \mathbf{Z}$

Gesucht: n-1 Werte $y_i \in \mathbf{Z}$ im abgeschlossenen Intervall $[y_0,y_n]$, die den Werten x_i ($i=1,...,n-1$) zugeordnet werden und eine auf ganzzahlige Werte gerundete, intuitiv nachvollziehbare lineare Interpolation zwischen (x_0,y_0) und (x_n,y_n) wiedergeben

Lineare / Linien-Algorithmen

Vorbetrachtungen, Notation, Methodik:

- x_i u. y_i werden in der Intervall-Mitte („Pixel-Mitte“) angenommen
- Mit den max. Werte-Differenzen $d_x=x_n–x_0$ und $d_y=y_n–y_0$:

  Es gilt immer: $d_x > 0$; $x_{i+1} = x_i + 1$

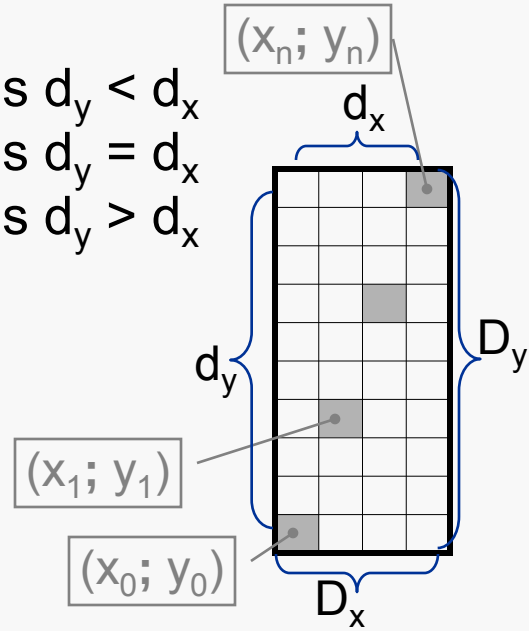
  Für $d_y > 0$: falls $d_y < d_x$: $y_{i+1} \geq y_i$; falls $d_y > d_x$: $y_{i+1} > y_i$

  Für $d_y < 0$: falls $-d_y < d_x$: $y_{i+1} \leq y_i$; falls $-d_y > d_x$: $y_{i+1} < y_i$
- Anzahl möglicher x- bzw. y-Werte (Wertevorrat):

  Wertevorrat in x: $D_x = d_x+1 = x_n–x_0+1 = n+1$

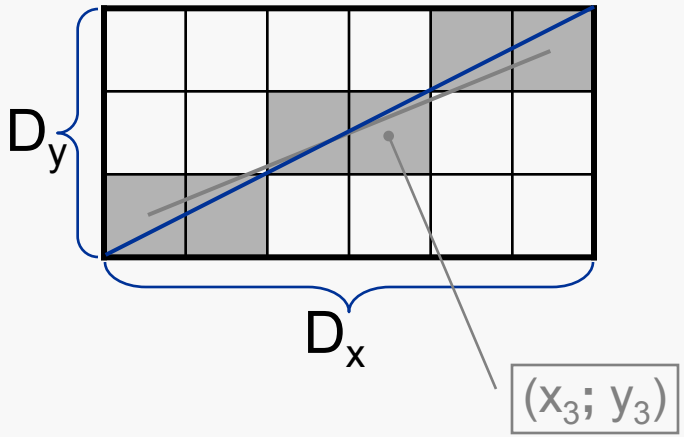
  Wertevorrat in y: $D_y = d_y+1 = y_n–y_0+1$ $\begin{cases} < n+1, \text{ falls } d_y < d_x \\ = n+1, \text{ falls } d_y = d_x \\ > n+1, \text{ falls } d_y > d_x \end{cases}$

- Herleitung mit Ganzzahl-Arithmetik, ausgehend von Konstellationen mit ganzzahligem, intuitivem Ergebnis



Lineare / Linien-Algorithmen

- 1. Oktant ($d_x > d_y$)
  Günstige Konstellation:
  $D_x = p \cdot D_y$; $p \in \mathbf{N}$
  Annahme eines Linienverlaufs
  von $(x_0–½; y_0–½)$ nach $(x_n+½; y_n+½)$
  – steiler als von $(x_0; y_0)$ nach $(x_n; y_n)$

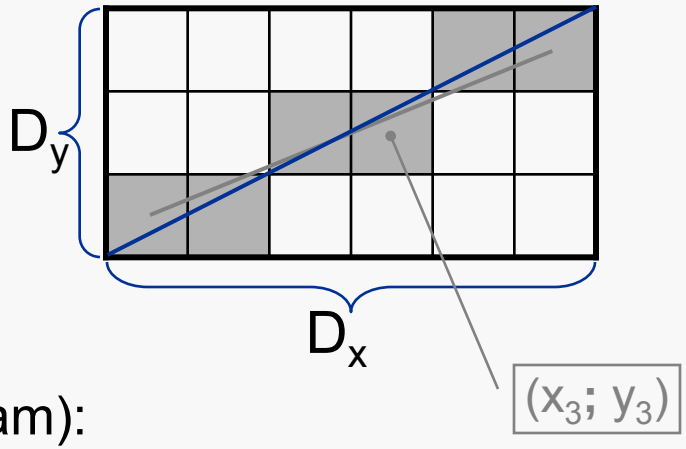


Dem Intervall (x_{i+1}) wird derselbe ganzzahlige Interpolationswert wie dem aktuellen (x_i) zugeordnet ($y_{i+1} = y_i$), wenn der exakte Linienverlauf aus dem aktuellen Intervall $[x_i–½, x_i+½]$ austritt, ohne das nächsthöhere Werteintervall $[y_i+½, y_i+1½]$ erreicht zu haben, d.h.:

$y_{i+1} = y_i$ falls $y(x_i+½) < y_i+½$; andernfalls Zuordnung $(x_{i+1}; y_i+1)$

[Linie: $(x_{i+1}; y_{i+1})$ Ost, falls die Gerade seitlich aus dem Vor-Pixel $(x_i; y_i)$ austritt; Nordost, falls sie durch die Nord-Kante oder Nordost-Ecke von $(x_i; y_i)$ führt.]

Lineare / Linien-Algorithmen



1. Oktant ($d_x > d_y$; Forts.)

$y_{i+1} = y_i$ (Ost), wenn:

$y(x_i+½) < y_i + ½$ (i=1,...,n), d.h.:

$(y_0 – ½) + (i + 1) \cdot D_y / D_x < y_i + ½$

$(i + 1) \cdot D_y / D_x < y_i – y_0 + 1$

Einführung von Variablen (vgl. Bresenham):

$N = y_i – y_0$: Anzahl bisheriger Nordost-Entscheidungen

E : Anzahl bisheriger Ost-Entscheidungen

$N + E = i$

$(N + E + 1) \cdot D_y / D_x < N + 1$, bzw., nach Umformung:

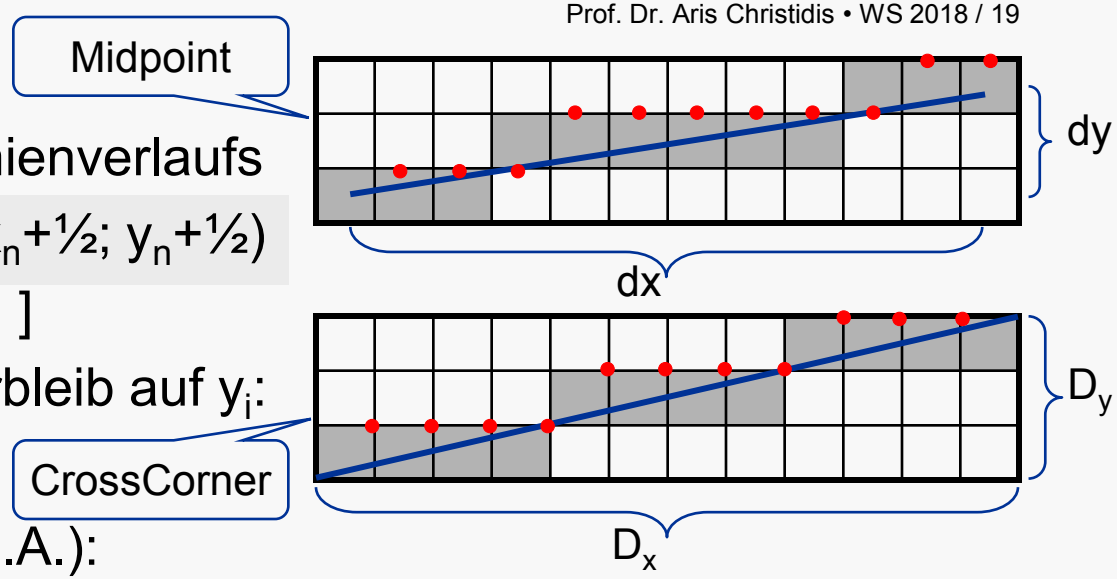
$N \cdot D_y + E \cdot D_y + D_y – N \cdot D_x – D_x < 0$ und daraus:

$N \cdot (D_y – D_x) + E \cdot D_y + D_y – D_x < 0$

Algorithmus-Entwurf in Anlehnung an Bresenham:

Eine „decision variable“ **dv** wird mit **Dy-Dx** initialisiert (Check für y_1). Für **dv<0** wird dem Intervall x_{i+1} der Interpolationswert $y_{i+1}=y_i$ zugeordnet und **dv** um **Dy** erhöht; sonst wird $y_{i+1}= y_i+1$ gesetzt und **dv** um **Dy-Dx** erhöht.

Lineare / Linien-Algorithmen



1. Oktant ($d_x > d_y$)

Annahme geänderten Linienverlaufs

von $(x_0–½; y_0–½)$ nach $(x_n+½; y_n+½)$

[D_y / D_x steiler als dy/dx !]

Kriterien-Wechsel für Verbleib auf y_i:

Pixel-Mitte ⇨ Pixel-Ecke

Herleitung für $m \leq 1$ (o.B.d.A.):

Wachstum gen Osten ($y_{i+1}=y_i$) bei seitlichem Austritt, d.h., wenn gilt:

$$y(x_i+½) = (y_0 – ½) + (i + 1) \cdot D_y / D_x < y_i + ½$$

$$(i + 1) \cdot D_y / D_x < y_i – y_0 + 1$$

Einführung von Variablen nach Bresenham:

N = $y_i–y_0$: Anzahl bisheriger Nordost-Entscheidungen

E : Anzahl bisheriger Ost-Entscheidungen

} N + E = i

$$(N + E + 1) \cdot D_y / D_x < N + 1 \text{ , nach Umformung:}$$

$$\mathbf{N \cdot (D_y – D_x) + E \cdot D_y + D_y – D_x < 0}$$

Lineare / Linien-Algorithmen

Midpoint-Algorithmus:

```
void Midpoint (int x0,
  int y0,int xn,int yn)
{ int dx, dy, d;
  int incrE, incrNE;
  int x=x0,  y=y0;
  dx=xn-x0;  dy=yn-y0;

  d=2*dy-dx; //=dy-dx+dy
  incrE =2*dy;//=dy+dy
  incrNE=2*(dy-dx);

  for (x=x0; x<xn; x++)
  {SetPixel (x,y);
   if (d<0) {d+=incrE; }
   else{y++; d+=incrNE;}
  }
  return; }
```

CrossCorner-Algorithmus:

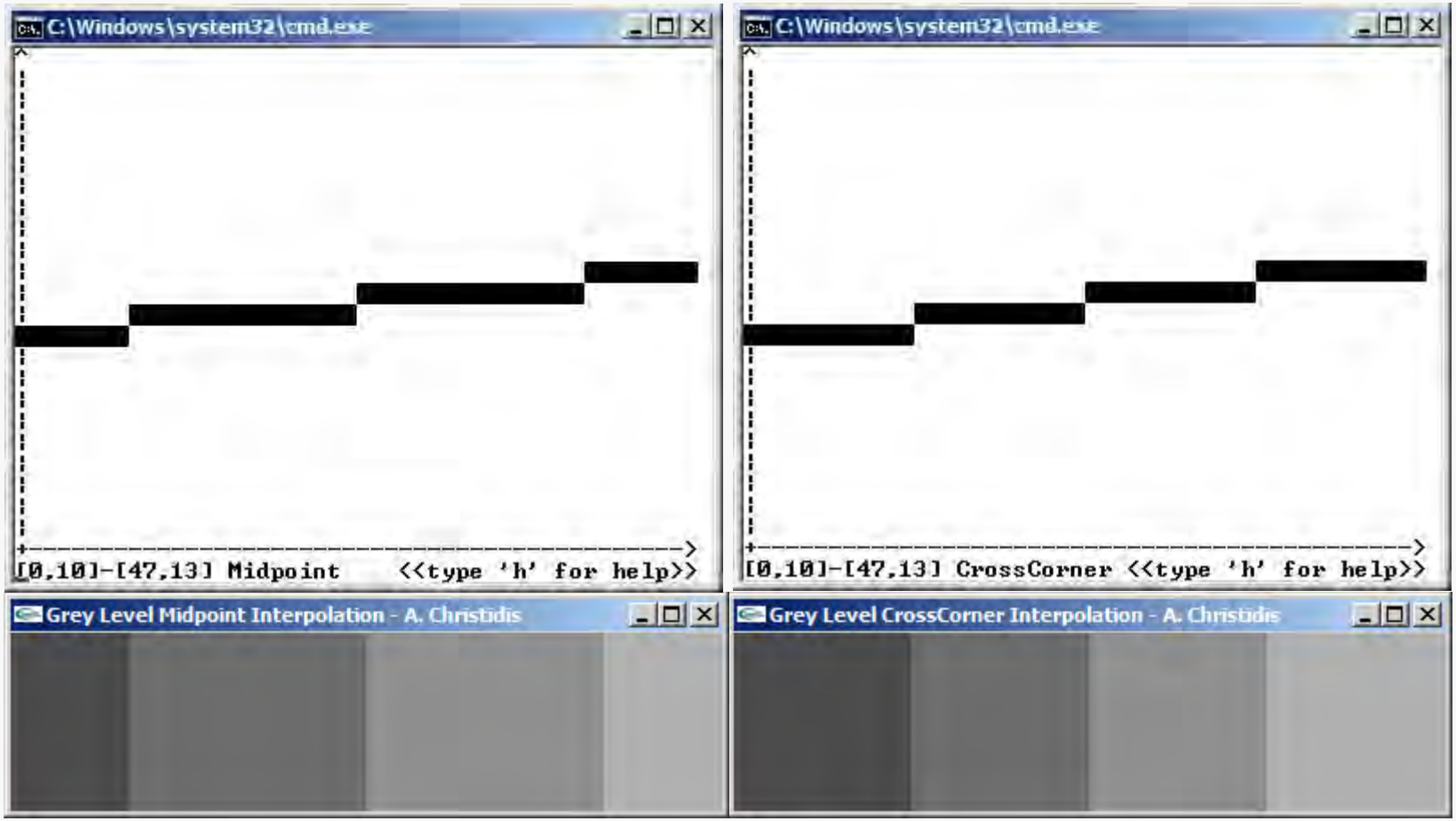
```
void CrossCorner(int x0,
   int y0,int xn,int yn)
{ int dx, dy, dv;
  int incrX, incrY;
  int x=x0, y=y0;
  dx=xn-x0; dy=yn-y0;

  dv=dy-dx;     //=Dy-Dx
  incrX=dy+1;   //=Dy
  incrY=dv;

  for (x=x0; x<xn; x++)
  {SetPixel (x,y);
   if(dv<0) {dv+=incrX;}
   else{y++; dv+=incrY;}
  }
  return; }
```

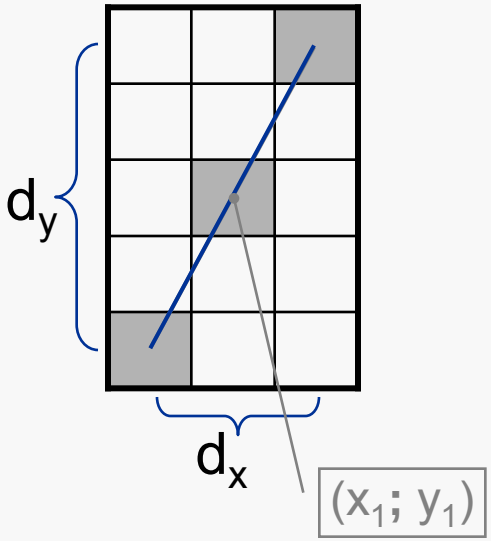
Lineare / Linien-Algorithmen

Vergleich Midpoint vs. CrossCorner:



Lineare / Linien-Algorithmen

- 2. Oktant ($d_x \le d_y$)
  Günstige Konstellation:
  $d_y = q \cdot d_x$; $q \in \mathbf{N}$
  Inkrementierung ($y_{i+1} \Rightarrow y_{i+1}+1$) solange y_{i+1} an der Intervallmitte x_{i+1} unterhalb des exakten Interpolationswerts $y(x_{i+1})$ liegt:
  $\mathbf{y_{i+1} \Rightarrow y_{i+1}+1,\ solange\ \ y_{i+1} < y(x_{i+1})}$
  d.h., wegen $d_y / d_x = [\, y(x_{i+1}) - y_0 \,] / [\, x_{i+1} - x_0 \,]$:
  $y_{i+1} - y_0 \ < \ y(x_{i+1}) - y_0 \ = \ (x_{i+1} - x_0) \cdot d_y / d_x$, bzw.
  $\underbrace{\mathbf{(y_{i+1} - y_0) \cdot d_x \ - \ (x_{i+1} - x_0) \cdot d_y}}_{\mathbf{dv}} \ \mathbf{< \ 0}$



Echtzeit?

Algorithmus-Entwurf in Anlehnung an Bresenham:
Eine „decision variable“ **dv** wird mit 0 initialisiert und für jeden neuen **x**-Wert um **dy** dekrementiert. Solange **dv<0** gilt, werden **y** um **1** und **dv** um **dx** inkrementiert.

Lineare / Linien-Algorithmen

Übung:

Implementierung einer Funktion zum echtzeitfähigen Belegen eines Feldes mit den Werten einer ganzzahligen linearen Interpolation; die Funktion selbst soll nur Ganzzahl-Additionen und Subtraktionen enthalten (s. Übungsblatt).

```
int set_iLerp (int Dx, int y0, int yn, int *yStore)
```

BresenLerp