Rastergrafikalgorithmen

Sebastian Kurfürst
s4832338@mail.inf.tu-dresden.de

Proseminar Computergraphik
Lehrstuhl Computergraphik und Visualisierung
Institut für Software- und Multimediatechnik
TU Dresden

Gliederung

- Rasterung von Linien
- Antialiasing
- Zeichnen und Füllen von Polygonen
- Clipping

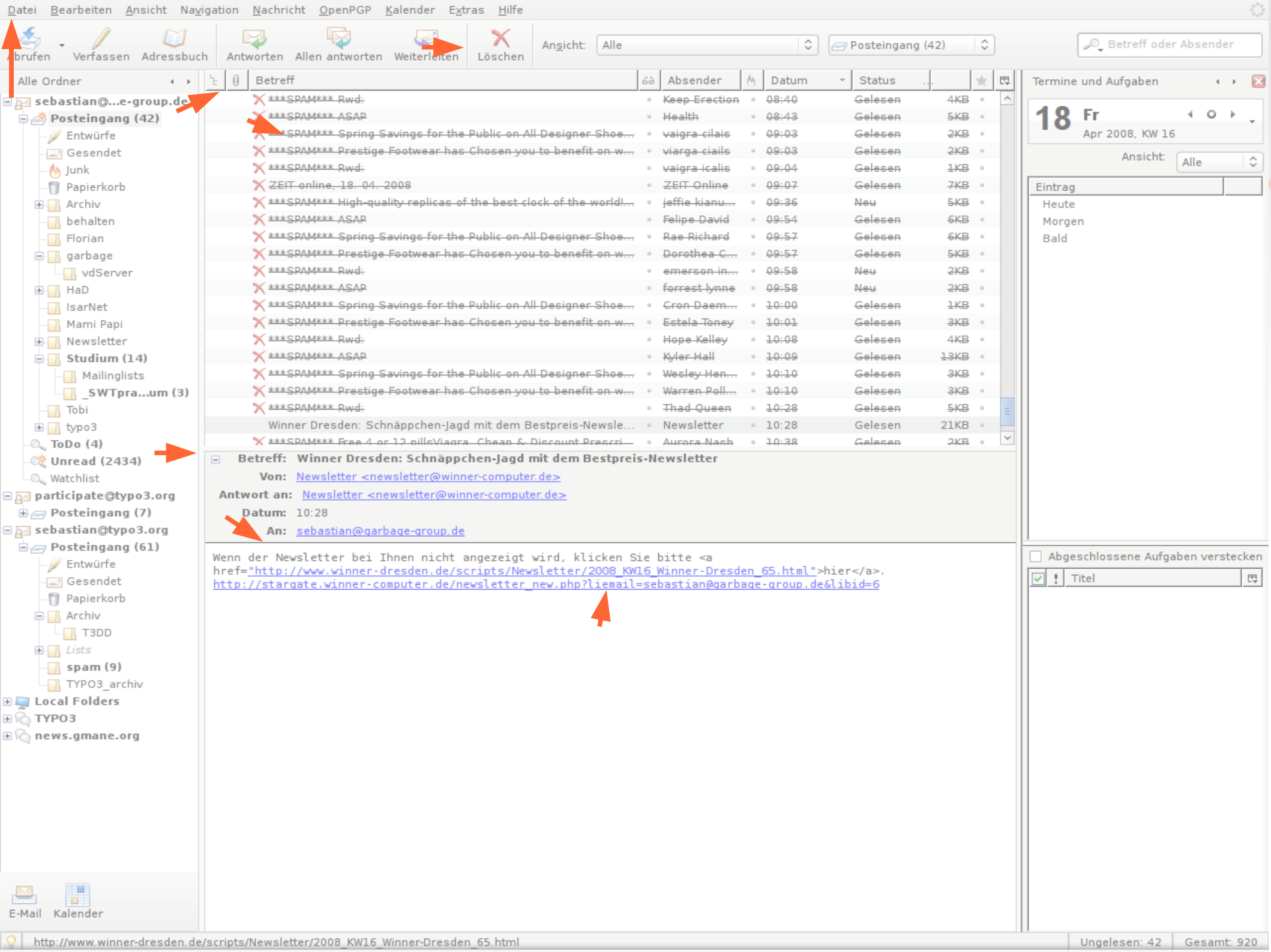
Gliederung

- Rasterung von Linien
- Antialiasing
- Zeichnen und Füllen von Polygonen
- Clipping

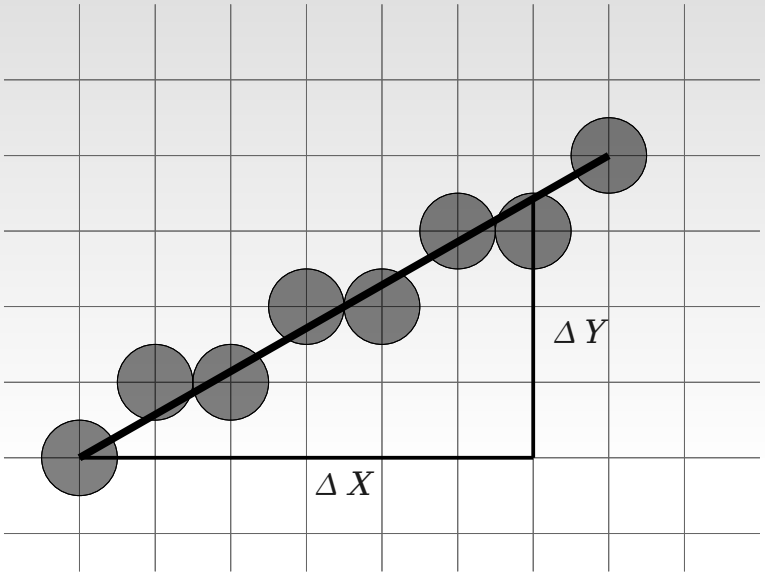
Rasterung von Linien

- Verbindung zweier Punkte (Vektor) ist wichtigstes grafisches Primitivum
- erwünschte Bedingungen:
  - konstante Helligkeit
  - hohe Zeichengeschwindigkeit
- unhabhängig vom Winkel

Warum ist die Linie so wichtig?


Datei Bearbeiten Ansicht Navigation Nachricht OpenPGP Kalender Extras Hilfe
Abrufen Verfassen Adressbuch Antworten Allen antworten Weiterleiten Löschen
Ansicht: Alle Posteingang (42) Betreff oder Absender
Alle Ordner
sebastian@...e-group.de
Posteingang (42)
Entwürfe
Gesendet
Junk
Papierkorb
Archiv
behalten
Florian
garbage
vdServer
HaD
IsarNet
Mami Papi
Newsletter
Studium (14)
Mailinglists
_SWTpra...um (3)
Tobi
typo3
ToDo (4)
Unread (2434)
Watchlist
participate@typo3.org
Posteingang (7)
sebastian@typo3.org
Posteingang (61)
Entwürfe
Gesendet
Papierkorb
Archiv
T3DD
Lists
spam (9)
TYPO3_archiv
Local Folders
TYPO3
news.gmane.org
E-Mail Kalender
Betreff Absender Datum Status
SPAM Rwd: Keep Erection 08:40 Gelesen 4KB
SPAM ASAP Health 08:43 Gelesen 5KB
SPAM Spring Savings for the Public on All Designer Shoe... vaigra cilais 09:03 Gelesen 2KB
SPAM Prestige Footwear has Chosen you to benefit on w... viarga ciails 09:03 Gelesen 2KB
SPAM Rwd: vaigra icalis 09:04 Gelesen 1KB
ZEIT online, 18. 04. 2008 ZEIT Online 09:07 Gelesen 7KB
SPAM High-quality replicas of the best clock of the world!... jeffie kianu... 09:36 Neu 5KB
SPAM ASAP Felipe David 09:54 Gelesen 6KB
SPAM Spring Savings for the Public on All Designer Shoe... Rae Richard 09:57 Gelesen 6KB
SPAM Prestige Footwear has Chosen you to benefit on w... Dorothea C... 09:57 Gelesen 5KB
SPAM Rwd: emerson in... 09:58 Neu 2KB
SPAM ASAP forrest lynne 09:58 Neu 2KB
SPAM Spring Savings for the Public on All Designer Shoe... Cron Daem... 10:00 Gelesen 1KB
SPAM Prestige Footwear has Chosen you to benefit on w... Estela Toney 10:01 Gelesen 3KB
SPAM Rwd: Hope Kelley 10:08 Gelesen 4KB
SPAM ASAP Kyler Hall 10:09 Gelesen 13KB
SPAM Spring Savings for the Public on All Designer Shoe... Wesley Hen... 10:10 Gelesen 3KB
SPAM Prestige Footwear has Chosen you to benefit on w... Warren Poll... 10:10 Gelesen 3KB
SPAM Rwd: Thad Queen 10:28 Gelesen 5KB
Winner Dresden: Schnäppchen-Jagd mit dem Bestpreis-Newsle... Newsletter 10:28 Gelesen 21KB
Betreff: Winner Dresden: Schnäppchen-Jagd mit dem Bestpreis-Newsletter
Von: Newsletter <newsletter@winner-computer.de>
Antwort an: Newsletter <newsletter@winner-computer.de>
Datum: 10:28
An: sebastian@garbage-group.de
Wenn der Newsletter bei Ihnen nicht angezeigt wird, klicken Sie bitte hier.
http://stargate.winner-computer.de/newsletter_new.php?liemail=sebastian@garbage-group.de&libid=6
Termine und Aufgaben
18 Fr
Apr 2008, KW 16
Ansicht: Alle
Eintrag
Heute
Morgen
Bald
Abgeschlossene Aufgaben verstecken
Titel
http://www.winner-dresden.de/scripts/Newsletter/2008_KW16_Winner-Dresden_65.html
Ungelesen: 42 Gesamt: 920

Naiver Algorithmus



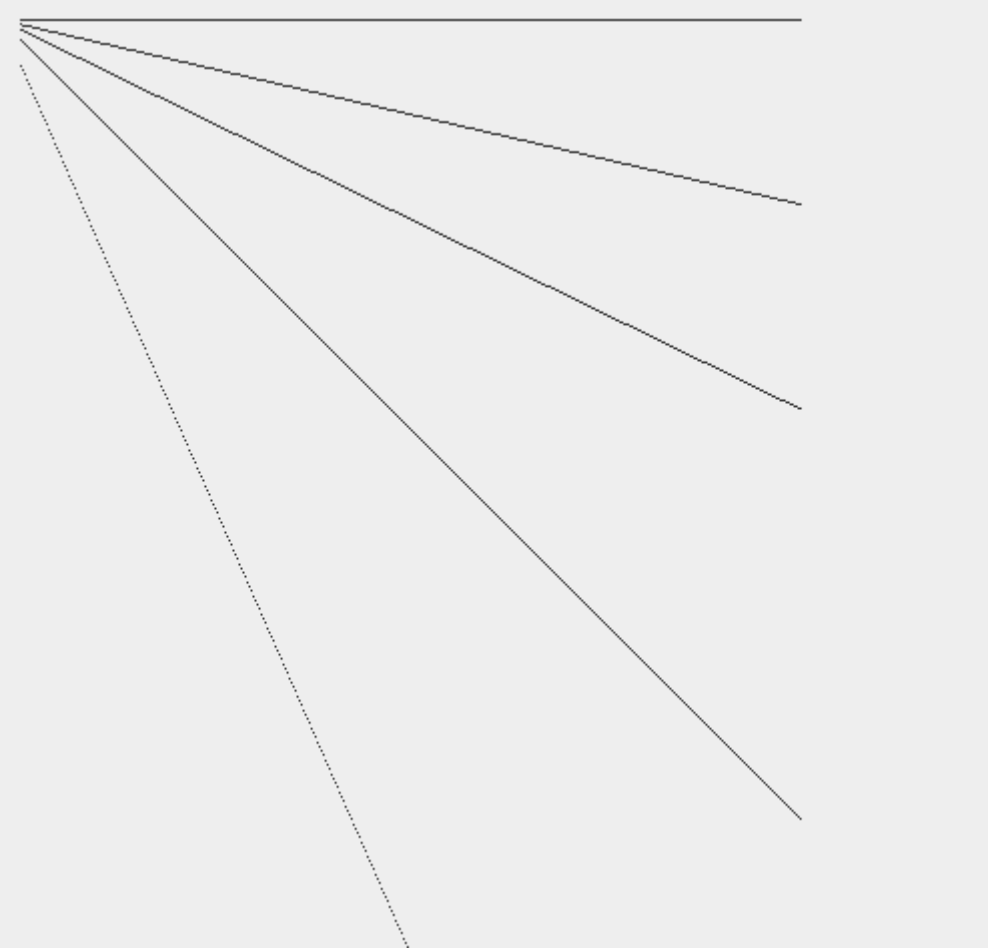
$$y = m * x + n$$
$$Y' = round(y)$$

```
nativeLine(int x0, int y0, int x1, int y1) {
    float y = 0f;
    float m = (float)(y1-y0)/(float)(x1-x0);
    for (int x=x0; x<x1; x++) {
        y = m*x + y0;
        paintDot(x,Math.round(y));
    }
}
```

Naiver Algorithmus - Bewertung

- Nachteile:
  - Floating-Point-Rechnungen, ständiges Runden
- Vorteile
  - extrem einfach zu implementieren

Probleme beim Darstellen von Linien



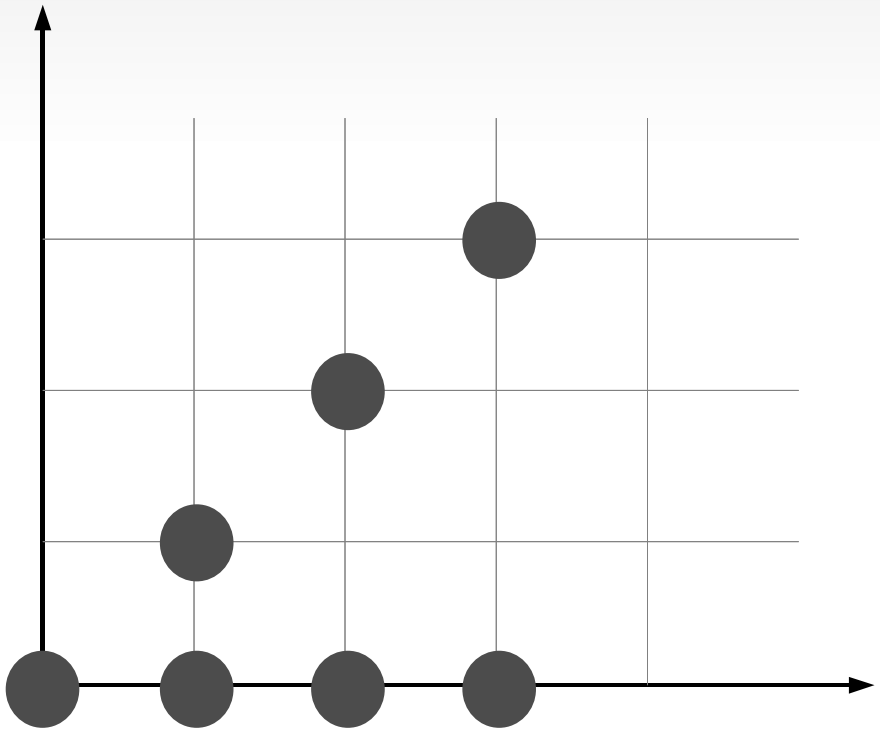
Probleme beim Darstellen von Linien (1)

- Bei Winkel > 45° müssen X und Y-Koordinate vertauscht werden

```
nativeLine(int x0, int y0, int x1, int y1) {
    float x = 0f;
    float m = (float)(y1-y0)/(float)(x1-x0);
    for (int y=y0; y<y1; y++) {
        y = m*x + y0;
        paintDot(Math.round(x),y);
    }
}
```

Probleme beim Darstellen von Linien (2)

- Unterschiedliche Helligkeit bei unterschiedlichen Winkeln



Horizontal

Anstieg: 0
Länge: 10
Intensität: 1

Schräg

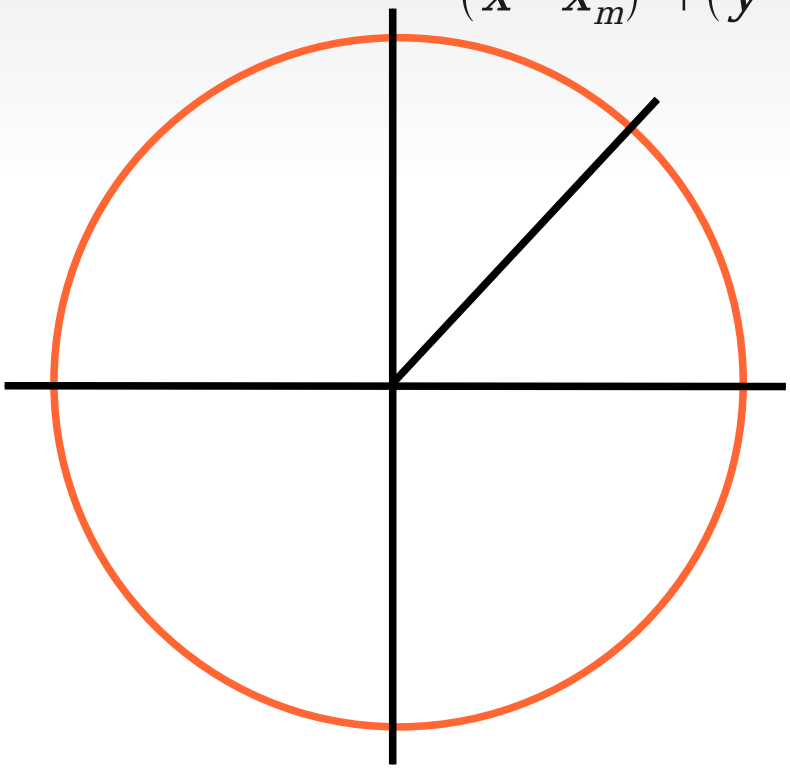
Anstieg: 1
Länge: $10 * \sqrt{2}$
Intensität: $1 \div \sqrt{2}$

Exkurs: Darstellung von Kreisen

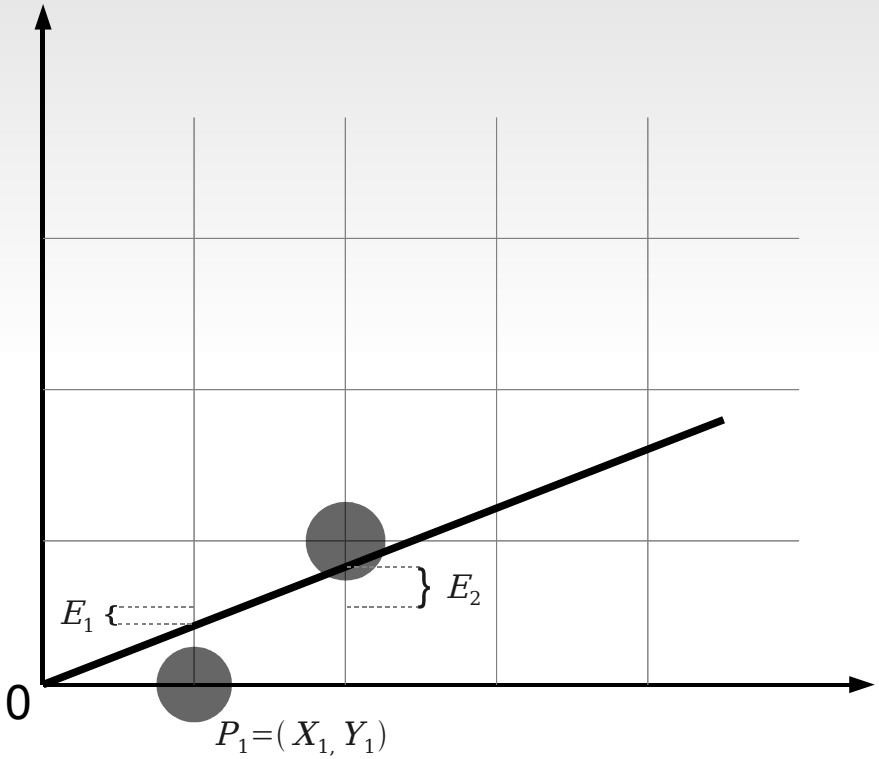
- Anstiegsproblematik
- Idee: Ausnutzung von Symmetrien

$$(x-x_m)^2+(y-y_m)^2=r^2$$

```
writeCirclePoint(x, y) {
    WritePixel(x,y)
    WritePixel(y,x)
    WritePixel(y,-x)
    WritePixel(x,-y)
    WritePixel(-x,-y)
    WritePixel(-y,-x)
    WritePixel(-y,x)
    WritePixel(-x,y)
}
```



Der Algorithmus von Bresenham

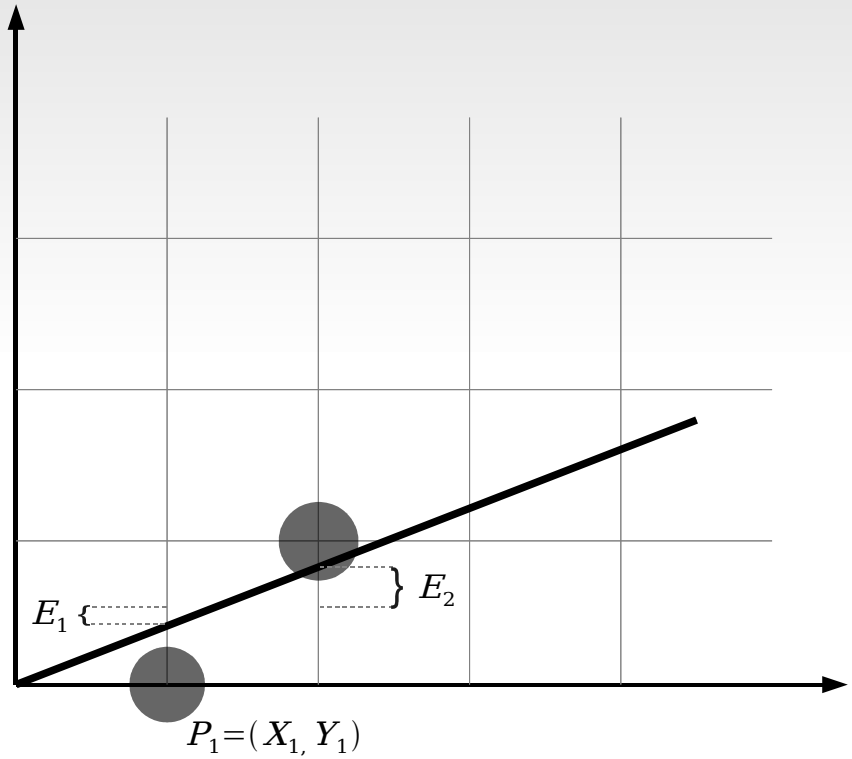


$$m=\frac{\Delta Y}{\Delta X}$$

$$E_1=y_1-\frac{1}{2}=m-\frac{1}{2}$$

$E_1>0$	$E_1\leq 0$
$X_1=X_0+1=1$	$X_1=X_0+1=1$
$Y_1=Y_0+1=1$	$Y_1=Y_0=0$

Der Algorithmus von Bresenham



$E_1 > 0$

$$E_2 = y_2 - Y_1 - \frac{1}{2}$$

$$E_2 = y_1 + m - Y_0 - 1 - \frac{1}{2}$$

$$E_2 = E_1 + m - 1$$

$E_1 \leq 0$

$$E_2 = y_2 - Y_1 - \frac{1}{2}$$

$$E_2 = y_2 - Y_0 - \frac{1}{2}$$

$$E_2 = y_1 + m - \frac{1}{2}$$

$$E_2 = E_1 + m$$

Der Algorithmus von Bresenham

$$E_1 = y_1 - \frac{1}{2} = \frac{\Delta Y}{\Delta X} - \frac{1}{2}$$

$E > 0$	$E \leq 0$
$E := E + m - 1$	$E := E + m$
$E := E - 1 + \frac{\Delta Y}{\Delta X}$	$E := E + \frac{\Delta Y}{\Delta X}$

- Nur Vorzeichen von E wichtig, daher Multiplikation mit $2 * \Delta X$

$$E_1' = 2 * E_1 * \Delta X = 2 * \Delta Y - \Delta X$$

$$E' \geq 0 \rightarrow E' := E' + 2(\Delta Y - \Delta X)$$
$$E' \leq 0 \rightarrow E' := E' + 2(\Delta Y)$$

Fertiger Bresenham-Algorithmus

```
bresenhamLine(int x0, int y0, int x1, int y1) {
  int deltaY = y1 - y0;
  int deltaX = x1 - x0;
  int x = x0;
  int y = y0;
  int e = 2 * deltaY - deltaX;

  for (int i = 1; i < deltaX; i++) {
     paintDot(x, y);
     x++;
     if (e>0) {
        y++;
        e = e + 2 * (deltaY - deltaX);
     } else {
        e = e + 2 * deltaY;
     }
  }
}
```

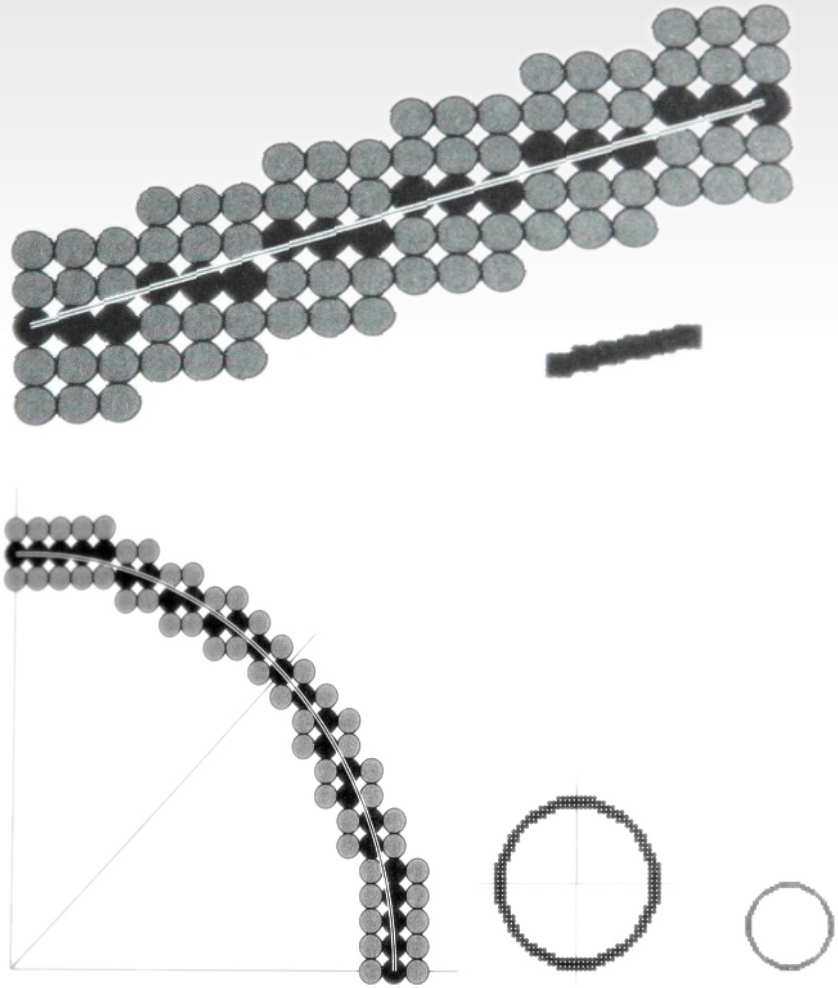
Bewertung Bresenham-Algorithmus

- nur noch Integer-Operationen

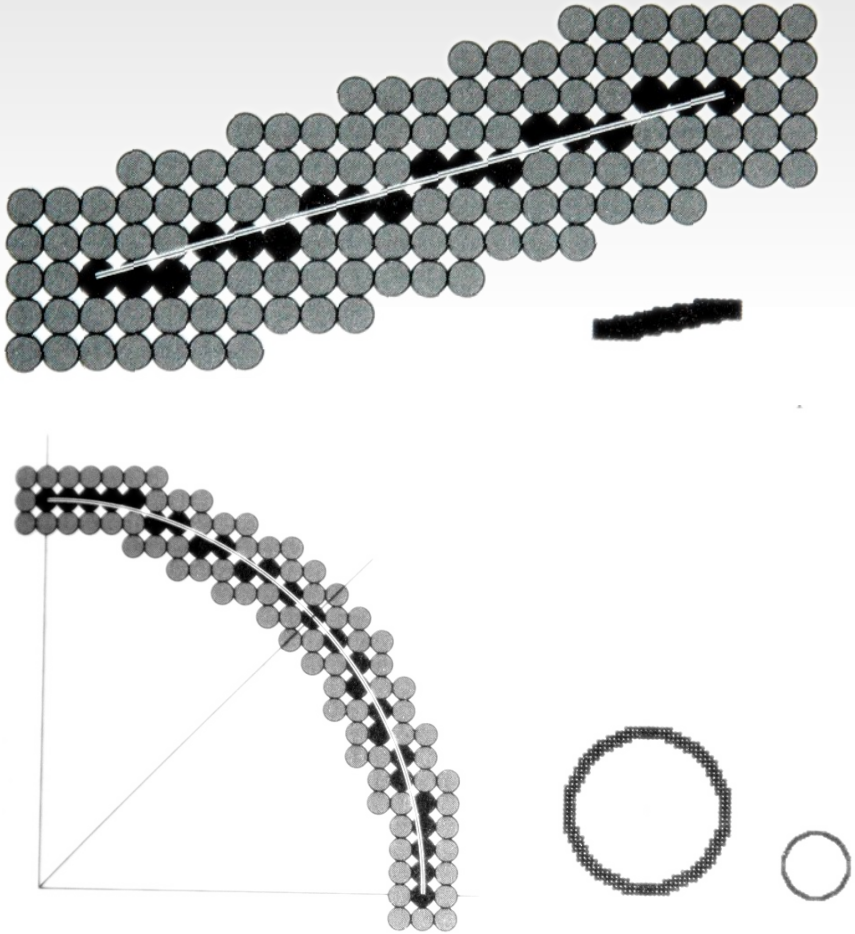
  → keine Rundung mehr nötig
- Benchmarks (100 000 kurze Linien)
  - Naiver Algorithmus: 2,4 s
  - Bresenham-Algorithmus: 1,8 s

Breite Primitive

Spaltenreplikation



Bewegter Stift



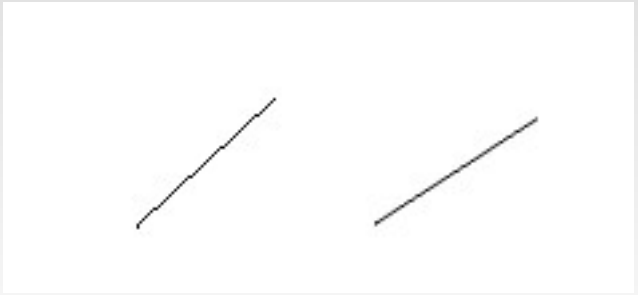
Gliederung

- Rasterung von Linien
- Antialiasing
- Zeichnen und Füllen von Polygonen
- Clipping

Antialiasing - Warum?

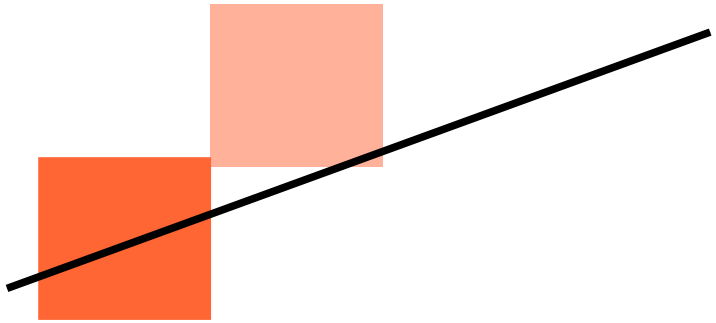
- Visuelles System besonders sensibel für Brüche und harte Kanten

Antialiasing von Linien



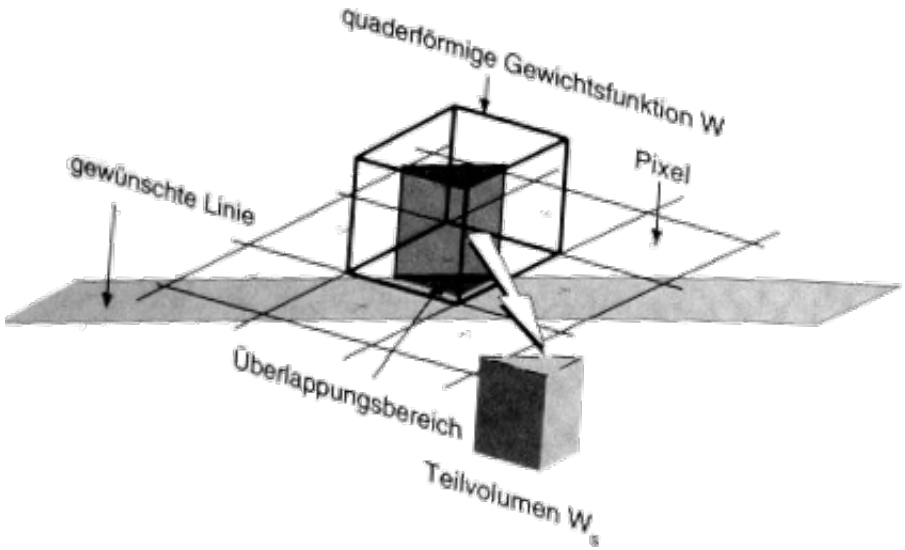
Grundprinzip

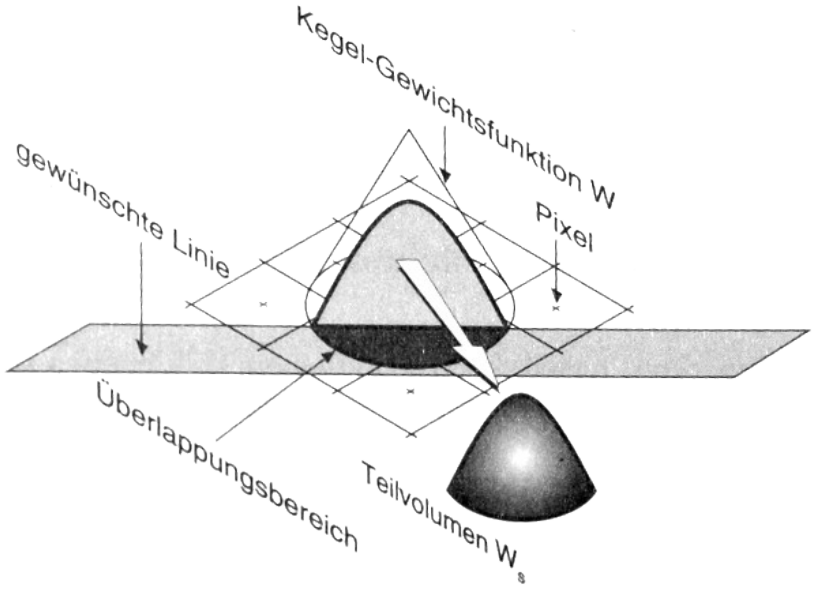
- Die Intensität eines Pixels, welches von einer Kante geschnitten wird, sinkt mit steigendem Abstand zw. Mittelpunkt des Pixels und der Kante.
- Eine Primitive beeinflusst nur die Intensitäten der Pixel, welche sie schneidet.



Antialiasingtechniken

- Ungewichtet
  - Gleich große Überlappungsbereiche bewirken gleich große Intensitäten
- Gewichtet
  - Verwendung von vorberechneten Tabellen

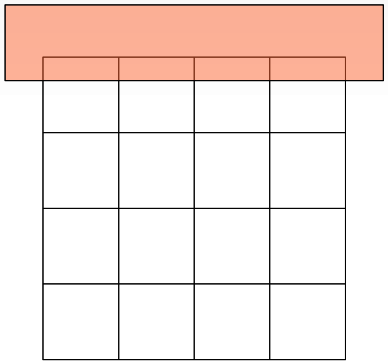


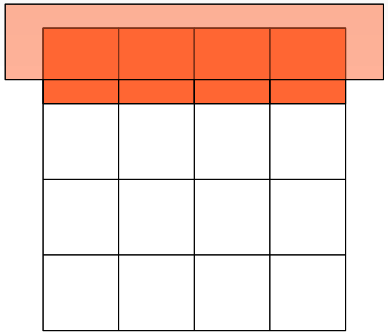


Antialiasing von Polygonkanten – Subpixelmasken

- Unterteilung eines Randpixels in quadratische Subpixel
- Zählung der

  überdeckten Pixel

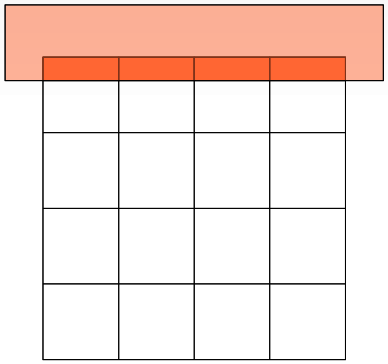




- Probleme:
  - bei zu kleinen Objekten: Flackern bei Bewegungen

Antialiasing von Polygonkanten – Subpixelmasken

- Bessere Alternative:
  - Genaues Ausrechnen der überdeckten Pixelfläche



Gliederung

- Rasterung von Linien
- Antialiasing
- Zeichnen und Füllen von Polygonen
- Clipping

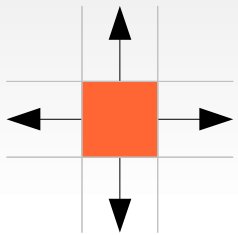
Zeichnen von Polygonen

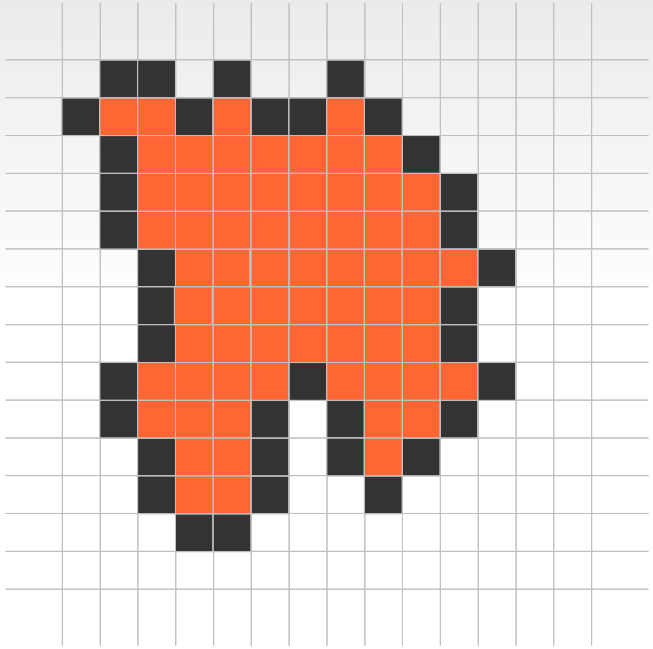
- Für unausgefüllte Polygone: Bresenham-Algorithmus
  - Entscheidungskriterium: kleinster Abstand zum gewünschten Vektor
- Für ausgefüllte Polygone:
  - Liegt ein Pixel außerhalb / innerhalb / auf der Polygonkante?

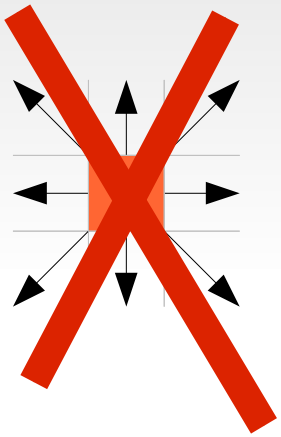
Zeichnen von Polygonen

- Pixel **auf** einer Polygonkante:
  - untere, linke Kanten: zeichnen
  - obere, rechte Kanten: nicht zeichnen

Füllen von Flächen – der Saatfüll-Algorithmus (1)







Füllen von Flächen – der Saatfüll-Algorithmus (2)

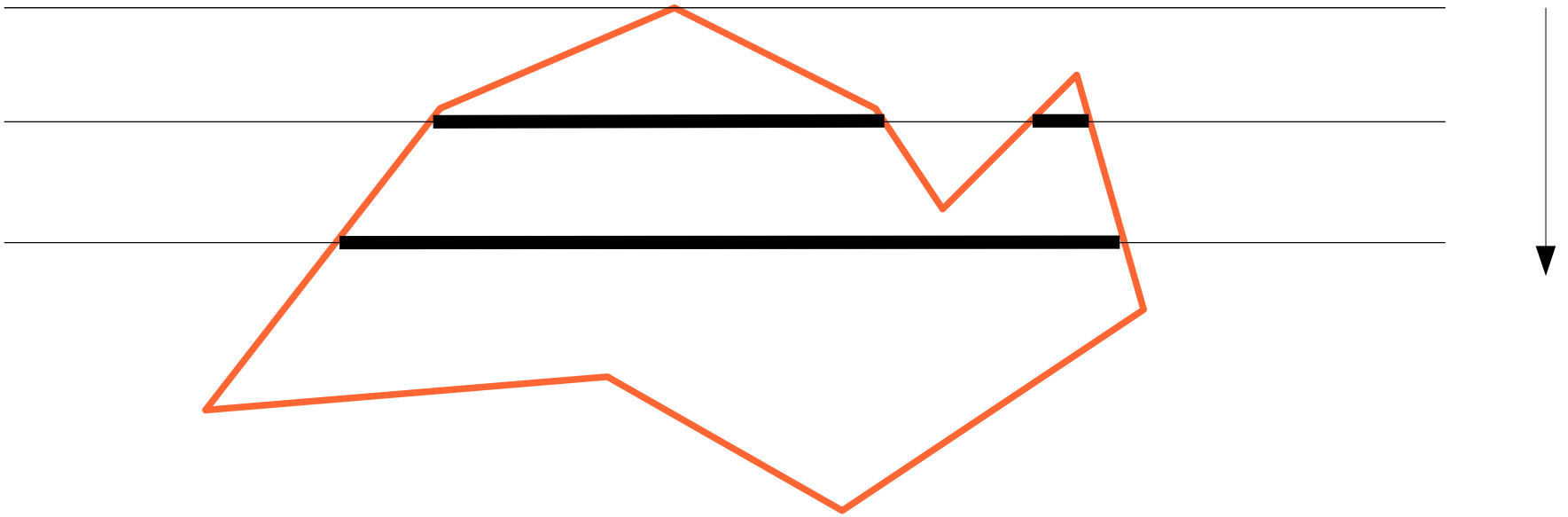
```
seedFill() {
    while( ! list.isEmpty()) {
        // get first element of list
        Point point = list.remove(0);
        // if pixel is white
        if (img.getRGB(point.x, point.y) == -1) {
            paintDot(point.x, point.y);
            list.add(new Point(point.x + 1, point.y));
            list.add(new Point(point.x - 1, point.y));
            list.add(new Point(point.x, point.y + 1));
            list.add(new Point(point.x, point.y - 1));
        }
    }
}
```

Füllen von Flächen – der Saatfüll-Algorithmus (3)

- von einem Saatkorn aus wird die Fläche gefüllt
- implizite (pixelbasierte) Polygondarstellung ausreichend
- mögliche Optimierung: Nutzung von Pixelläufen und damit Füllung kompletter Zeilen

Füllen von Flächen – Scangeraden-Algorithmus

- Polygon muss als Menge von Teilvektoren vorhanden sein



Gliederung

- Rasterung von Linien
- Antialiasing
- Zeichnen und Füllen von Polygonen
- Clipping

Clipping

- Üblicherweise verfügen Programme über ein rechteckiges Fenster (Window), dessen Ränder den interessierenden Bildausschnitt begrenzen
- Außerhalb des Fensters liegende Bildinformationen müssen vor der Bildausgabe abgeschnitten werden.

→ Clipping

Clipping

- Wie teilt man die Bildinformation in sichtbar und unsichtbar ein?
- Naiver Algorithmus:
  - für jeden Bildpunkt, gilt:

$$X_{min} \leq X \leq X_{max} \wedge Y_{min} \leq Y \leq Y_{max} \ ?$$

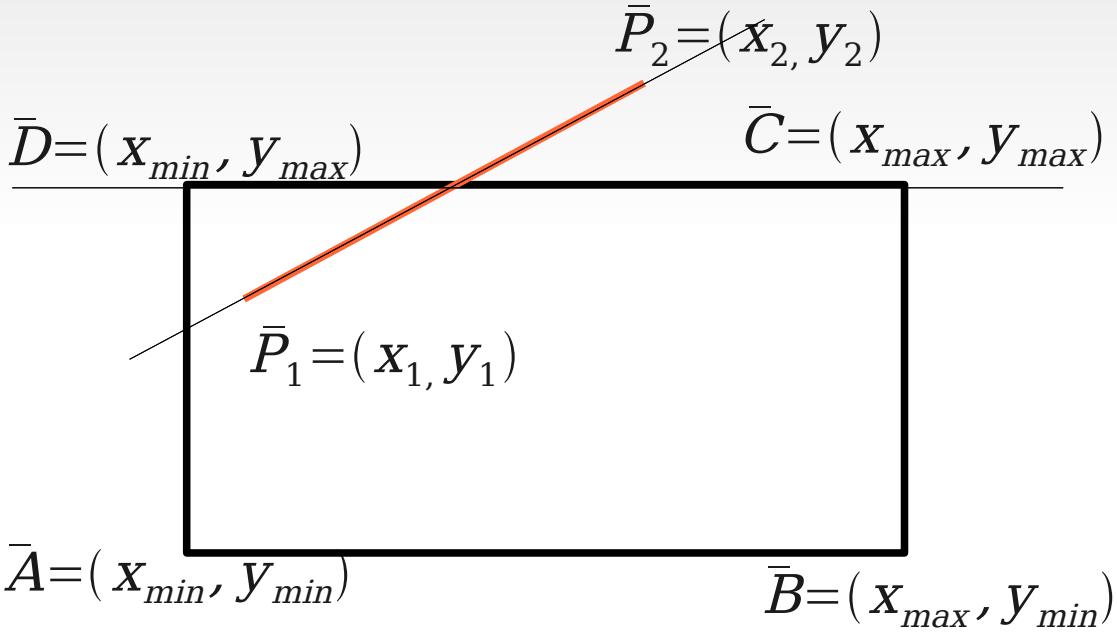
  - Wenn nein, dann liegt Bildpunkt außerhalb von Viewport
- Problem: Zu langsam und ineffizient für Vektorformen

Clipping von Geradensegmenten

- Clipping von $s = \bar{P}_1 \bar{P}_2$
- Fallunterscheidung
  - beide Eckpunkte liegen in Viewport
    - kein Clipping nötig, Gerade muss gezeichnet werden
  - nur einer der Eckpunkte liegt in Viewport
    - Clipping nötig, Gerade muss gezeichnet werden
  - keiner der Eckpunkte liegt in Viewport
    - Clipping nötig, Gerade kann gezeichnet werden

Clipping von Geradensegmenten

- Clippen von $s = \vec{P}_1 + t(\vec{P}_2 - \vec{P}_1), 0 \le t \le 1$



1. Schnittpunkte der Geraden s' und f' ermitteln

$$s' = \vec{P}_1 + t(\vec{P}_2 - \vec{P}_1), -\infty \le t \le \infty$$

$$f' = \vec{Q}_1 + u(\vec{Q}_2 - \vec{Q}_1), -\infty \le u \le \infty$$

$$\vec{P}_1 + t(\vec{P}_2 - \vec{P}_1) = \vec{Q}_1 + u(\vec{Q}_2 - \vec{Q}_1)$$

Clipping von Geradensegmenten

$$\vec{P}_1 + t(\vec{P}_2 - \vec{P}_1) = \vec{Q}_1 + u(\vec{Q}_2 - \vec{Q}_1)$$

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + t\left[\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}\right] = \begin{pmatrix} x_{min} \\ y_{max} \end{pmatrix} + u\left[\begin{pmatrix} x_{max} \\ y_{max} \end{pmatrix} - \begin{pmatrix} x_{min} \\ y_{max} \end{pmatrix}\right]$$

- 2 Gleichungen, 2 Unbekannte: 1 Lösung
- Wenn $0 \leq t \leq 1 \wedge 0 \leq u \leq 1$, so hat man den Schnittpunkt gefunden
- Andernfalls: erneutes Schneiden mit nächster Fensterkante

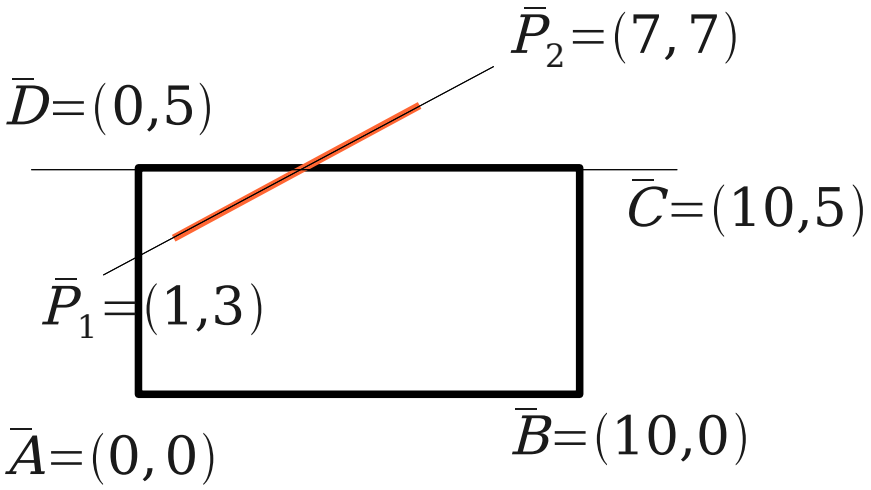
Clipping von Geradensegmenten – Zahlenbeispiel

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + t\left[\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}\right] = \begin{pmatrix} x_{min} \\ y_{max} \end{pmatrix} + u\left[\begin{pmatrix} x_{max} \\ y_{max} \end{pmatrix} - \begin{pmatrix} x_{min} \\ y_{max} \end{pmatrix}\right]$$

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} + t\left[\begin{pmatrix} 7 \\ 7 \end{pmatrix} - \begin{pmatrix} 1 \\ 3 \end{pmatrix}\right] = \begin{pmatrix} 0 \\ 5 \end{pmatrix} + u\left[\begin{pmatrix} 10 \\ 5 \end{pmatrix} - \begin{pmatrix} 0 \\ 5 \end{pmatrix}\right]$$

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} + t\begin{pmatrix} 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 0 \\ 5 \end{pmatrix} + u\begin{pmatrix} 10 \\ 0 \end{pmatrix}$$

$$t\begin{pmatrix} 6 \\ 4 \end{pmatrix} - u\begin{pmatrix} 10 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$



$$6t - 10u = -1$$
$$4t - 0u = 2$$
$$\rightarrow t = \frac{1}{2}$$
$$\rightarrow u = \frac{4}{10}$$

$$\begin{pmatrix} 0 \\ 5 \end{pmatrix} + \frac{4}{10}\begin{pmatrix} 10 \\ 0 \end{pmatrix} = \begin{pmatrix} 4 \\ 5 \end{pmatrix}$$

Clipping von Geradensegmenten

- Nicht besonders effizient
  - sehr viele Schnittpunktberechnungen
- Verbesserung: Cohen-Sutherland-Algorithmus

Cohen-Sutherland-Clipping

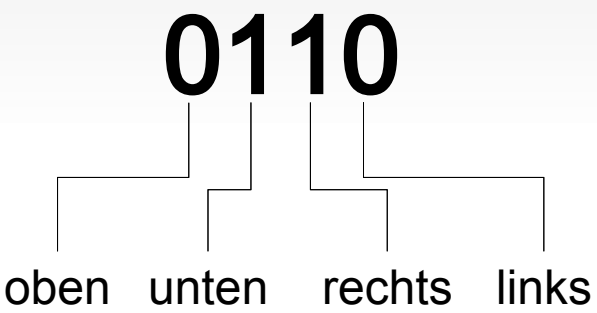
- Jedem Vektorendpunkt wird ein 4-Bit-Code zugeordnet:

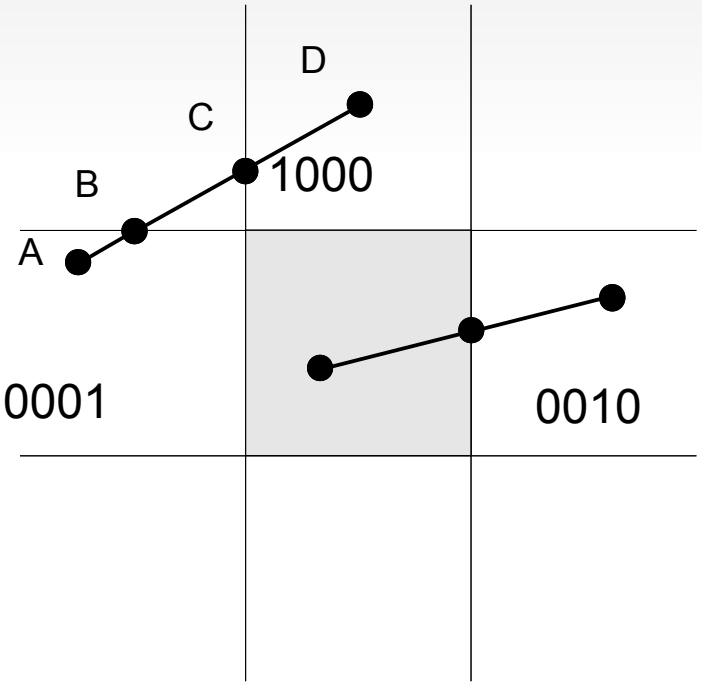
1001	1000	1010
0001	**0000**	0010
0101	0100	0110

- Ein Vektor liegt
  - völlig innerhalb des Fensters, wenn der Code für beide Endpunkte 0000 ist.
  - völlig außerhalb des Fensters, wenn code(A) & code(B) != 0000

Cohen-Sutherland-Clipping

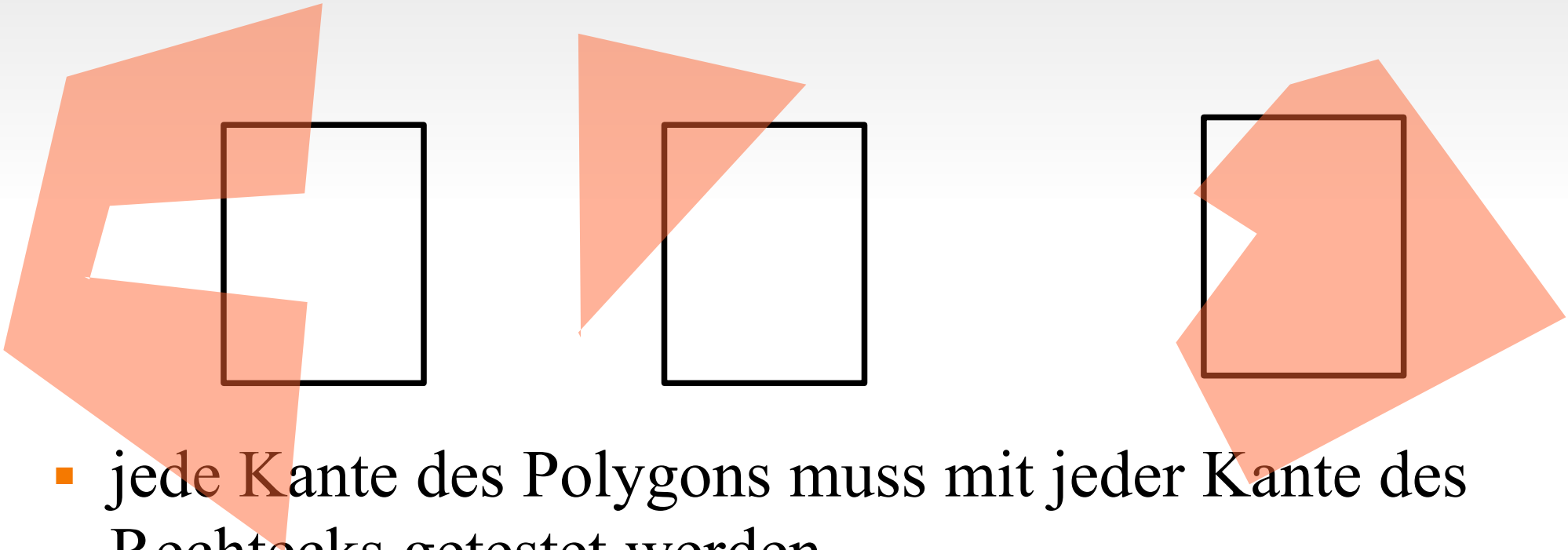
- Berechnung des Schnittpunktes des Vektors mit einer geeigneten Fensterkante





Polygonclipping

- Beispiele



- jede Kante des Polygons muss mit jeder Kante des Rechtecks getestet werden

Quellen

- Foley et al: Grundlagen der Computergraphik (Addison-Wesley, 1994)
- Rauber: Algorithmen in der Computergraphik (Teubner, 1993)
- Encarnação: Graphische Datenverarbeitung (Oldenburg, 1996)
- Pavlidis: Algorithmen zur Grafik und Bildverarbeitung (Heise, 1994)

Vielen Dank!