LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LFE Medieninformatik • Prof. Dr. Ing. Axel Hoppe

3D-Computergrafik

Rastergrafik

Notation von Polygonen

- Polygone werden nach 2D abgebildet
- Für weitere Betrachtungen reicht also zunächst:
  - Betrachtung 2D-Prmitiven
  - Erörtern der Aufgaben der Computergrafik im 2D-Fall

Vom Modell zum Bild: Vorüberlegungen

- Spezialfall: 2D-Computergraphik
- Bezug nehmend auf den allgemeinen Prozess der Bilderzeugung mit dem Rechner ergeben sich bei der 2D-Bilderzugung folgende Aufgaben:
  - Abbildung der „Vision eines Bildes“ eines Benutzers in einer geeigneten Modellbeschreibung im Rechner,
  - Überführung des Modells in eine Form, die am Bildschirm ausgegeben werden kann.

Vom Modell zum Bild: Vorüberlegungen

- Voraussetzung: Vision eines 2D-Bildes eines Benutzers durch Positionierung von 2D-Primitiven möglich
- Solche 2D-Primitive sind z. B.
  - Linie,
  - Kreis,
  - Ellipse,
  - Rechteck,
  - Vieleck (Polygon) etc.

Aufgabenstellung: Zielformulierung und Motivation

- Gesucht ist also grundsätzlich

1. eine Datenstruktur, die es erlaubt, die 2D-Primitive im Rechner abzubilden
2. spezielles Programm, dass die 2D-Primitive aus der Modellbeschreibung in die spezifische Beschreibung des Ausgabegeräts überführt

- Am weitesten verbreitet sind hierbei rasterorientiert arbeitende Ausgabegeräte.

Aufgabenstellung: Zielformulierung und Motivation

- Das bedeutet bei einer Ausgabe auf einem rasterorientierten Datensichtgerät (Monitor), Speicherung in einer Bitmap oder Ausgabe auf einen Drucker

1. die Überführung der „idealen“ 2D-Primitive aus der Modellbeschreibung in die entsprechenden Rasterprimitive des Ausgabegeräts und
2. die Transformation der der Koordinaten der 2D-Primitive aus dem Koordinatensystem des Modells in das gerätespezifische Koordinatensystem.

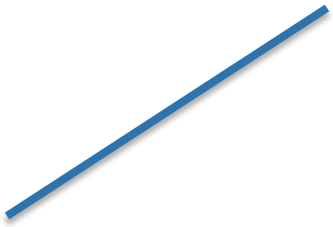
- Ausgegeben wird auf den rasterorientiert arbeitenden Ausgabegeräten ein Rasterbild.

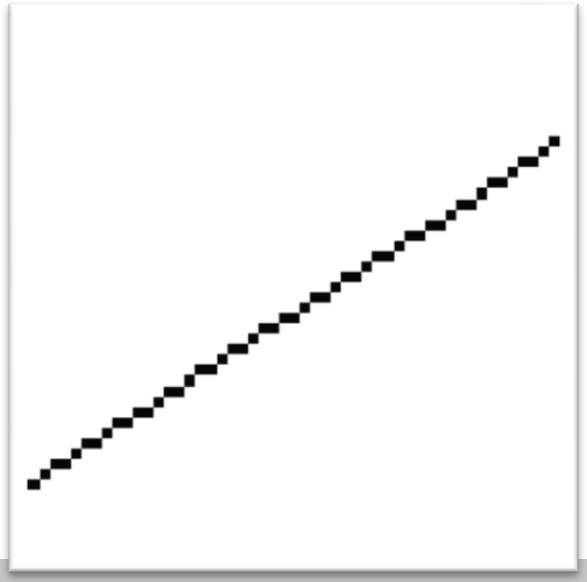
Rasterbilder – Vor- und Nachteile

- Vorteile:
  - Einfache Speicherung (Anordnung der Elemente)
  - Viele Nachbearbeitungsmöglichkeiten (vgl. Bildverarbeitung)

Rasterbilder – Vor- und Nachteile

- Nachteile:
  - Diskretisierung einer geometrischen Beschreibung erforderlich
  - Treppenstufen-Effekte (Aliasing) beim Transformieren (Vergrößern, Rotieren)
  - Hoher Speicherplatzbedarf





Rasterbilder – nähere Betrachtung

- Ein Rasterbild B als Matrix mit R Spalten und L Zeilen: $B = (b(x, y))$
- wobei x der Spaltenindex mit $0 \leq x \leq R - 1$ und
- y der Zeilenindex mit $0 \leq y \leq L - 1$ ist. Das Bildelement besitzt dabei den Wert $b(x, y)$.

Rasterbilder – nähere Betrachtung

- Der Farbwert c ist eine (irgendwie geartete) Kombination von Farbkanälen
- $c = K(v_0, \ldots, v_n)\ v \in V$
- wobei V das zu Grunde liegende Farbmodell repräsentiert.

Rasterbilder – nähere Betrachtung

- Ferner lässt sich ein Bild wie folgt betrachten:
- $B = (b(x, y, n))$
- wobei n der Kanalzähler für ein zu Grunde liegendes Farbmodell mit N Farbkanälen ist mit $0 \leq n \leq N$.

Anforderungen an Algorithmen

- Rasterungsalgorithmen werden oft gebraucht, sind daher zeitkritisch.
- Anforderungen:
  - keine aufwändigen Operationen (Multiplikation, Division sind aufwändiger als Addition),
  - möglichst mittels Integer-Arithmetik implementieren,
  - möglichst einfach strukturierte Algorithmen.

Vorausgehende Betrachtungen

- Wie sollen schließlich später die gerasterten Primitive vorliegen?
- Rasterbild ist aus Pixeln zusammengesetzt. „Pixel“ stammt vom englischen *picture element* und kennzeichnet die kleinste unteilbare Einheit eines Rasterbildes.
- Die Pixel im Rasterbild
  - sind gitterförmig angeordnet in Zeilen und Spalten,
  - sind alle gleich groß,
  - werden angesehen als Quadrate, die eine bestimmte Fläche überdecken,
  - alle Pixel eines Rasterbildes bedecken dieses vollständig (es gibt keine Lücken).

Zusammenfassung: Eigenschaften von Pixeln

- Position in der Bildmatrix,
  - angegeben in Koordinaten der Bildmatrix,
  - also als Wertepaar (Spalte, Zeile),
  - Grundlage hierfür bildet das Koordinatensystem der Bildmatrix
- beliebiger Farbwert,
- Größe (Ausdehnung) der Rechteckfläche des Pixels

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LFE Medieninformatik • Prof. Dr. Ing. Axel Hoppe

2D-Rastergrafik

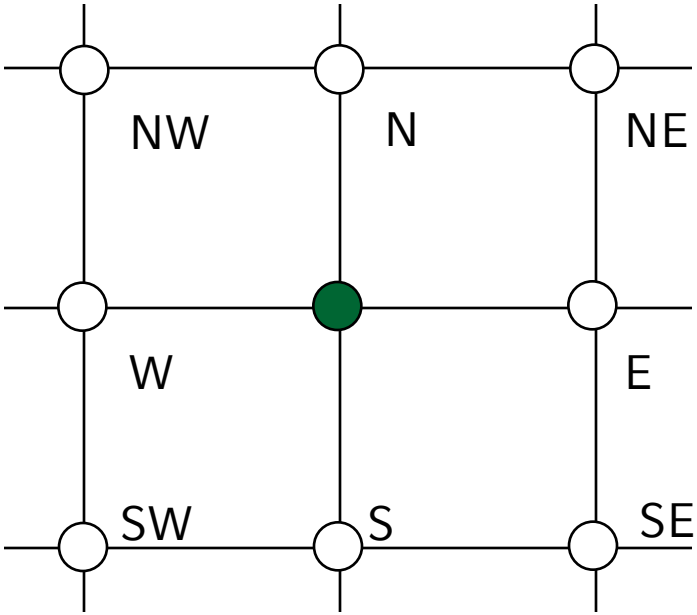
Einführung

Abstraktion: Pixelmodell für Entwurf von Algorithmen

- Konzentration auf das Wesentliche:
  - Bild besteht aus gleichmäßigem Gitter
  - An Gitterkreuzungspunkten befinden sich die Bildelemente
- Vereinfachung der Bildelemente:
  - keine Ausdehnung
  - sind ein- oder ausgeschaltet

Abstraktion: Pixelmodell für Entwurf von Algorithmen

- Bildelemente stehen in Relation zueinander:
  - Vierernachbarschaft
  - Achternachbarschaft
- Nachbarschafts-Beziehung wird gemäß der Himmelsrichtungen kodiert



LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LFE Medieninformatik • Prof. Dr. Ing. Axel Hoppe

2D Rastergrafik

Rastern von Linien

Anforderungen

- Linien sollten gerade aussehen
- Linien sollten gleichmäßig hell erscheinen
- Helligkeit sollte unabhängig von Richtung sein
- Endpunkte sollten exakt sein
- Linien sollen konstante Dicke haben
- Linienalgorithmus soll schnell sein

Linien mathematisch

- Gegeben: zwei Punkte $P_1(x_1, y_1)$ und $P_2(x_2, y_2)$
- Gesucht: Beschreibung der Linie zwischen P_1 und P_2
- setzen:
  - $\Delta x = x_2 - x_1$
  - $\Delta y = y_2 - y_1$
  - Anstieg $m = \Delta y / \Delta x$

Linie: Mathematische Beschreibung

- erste Möglichkeit: parametrische Beschreibung

$$y = x_1 + t(x_2 - x_1) = x_1 + t\Delta x$$

$$y = y_1 + t(y_2 - y_1) = y_1 + t\Delta y$$

- man erhält:

für $t = 0$ den Punkt (x_1, y_1)

für $t = 1$ den Punkt (x_2, y_2)

- Alle Punkte der Linie erhält man, wenn man für t alle Werte aus [0,1] einsetzt.

Linie: Mathematische Beschreibung

- zweite Möglichkeit: Beschreibung über Anstieg

$$y = mx + n$$

$$m = \Delta y / \Delta x$$

n: y-Koordinate des Schnittpunkts mit der y-Achse

- dritte Möglichkeit: implizite Beschreibung der Linie

$$F(x, y) = ax + by + c = 0$$

- Für alle Punkte auf der Gerade ist obige Gleichung erfüllt.

Naiver Algorithmus

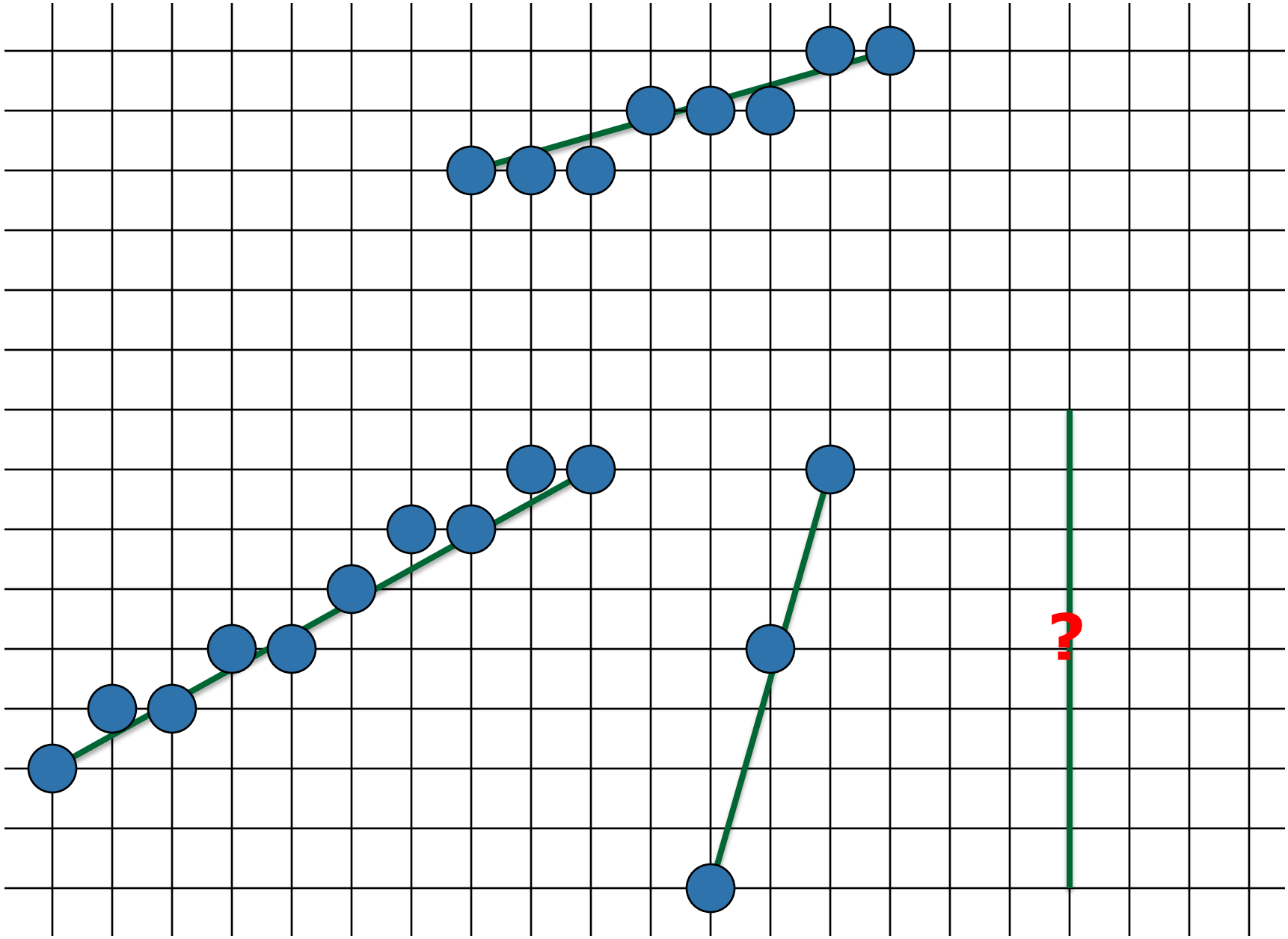
- Gegeben: zwei Punkte $P_1(x_1, y_1)$ und $P_2(x_2, y_2)$ jeweils Integer-Koordinaten
- Gesucht: alle Pixel, die auf der Linie liegen
- erster Schritt: Geradengleichung $y = mx + n$ aufstellen
- Einfachster Weg: Iteration von x_1 nach x_2 im Pixelabstand, entsprechende y-Werte berechnen, runden, zeichnen

```
double m = (y2-y1)/(x2-x1) ;
for (int x = x1; x <= x2; x++) {
    double y = m*x + n;
    setPixel( x, round( y ));
}
```

Naiver Algorithmus

```
double m = (y2-y1)/(x2-x1);
for (int x = x1; x <= x2; x++) {
 double y = m*x + n;
 setPixel( x, round( y ));
}
```

- Rechnen mit Gleitkommawerten bei *y* und *m*
- Division und Multiplikation verbrauchen Zeit
- Runden
- Was ist mit senkrechten Linien –*m* nicht definiert
- Aussehen der Linien bei verschiedenen Anstiegen



DDA (Digital Differential Analyzer)

- DDA = Digital Differential Analyzer
- Man wandert der Linie in kleinen Schritten entlang und zeichnet alle Pixel, in denen ein Schritt endet
- Schrittweite ≤ Pixelbreite, damit Keine Löcher
- Schrittweite erhält man z. B. durch mehrmaliges Halbieren der Gesamtlänge

DDA (Digital Differential Analyzer)

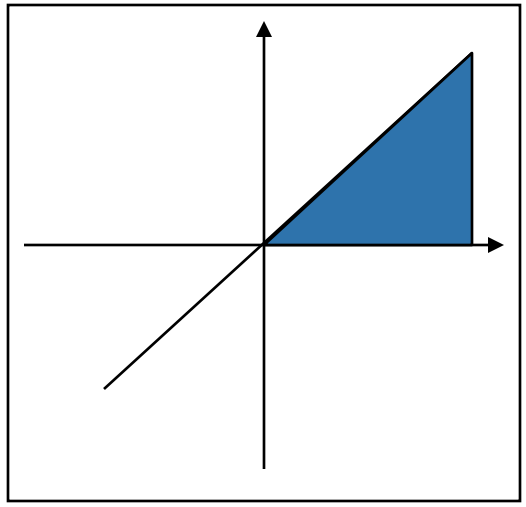
```
int Length, I;
double  X, Y, Xinc, Yinc;

Length = abs(X2 - X1);
if (abs(Y2 - Y1) > Length) Length = abs(Y2-Y1);
Xinc = (X2 - X1) / Length;
Yinc = (Y2 - Y1) / Length;
X = X1;
Y = Y1;
while(X < X2){
     setPixel(Round(X), Round(Y));
     X = X + Xinc;
     Y = Y + Yinc;
}
```

- Probleme:
  - Division
  - Runden

Bresenham-Algorithmus

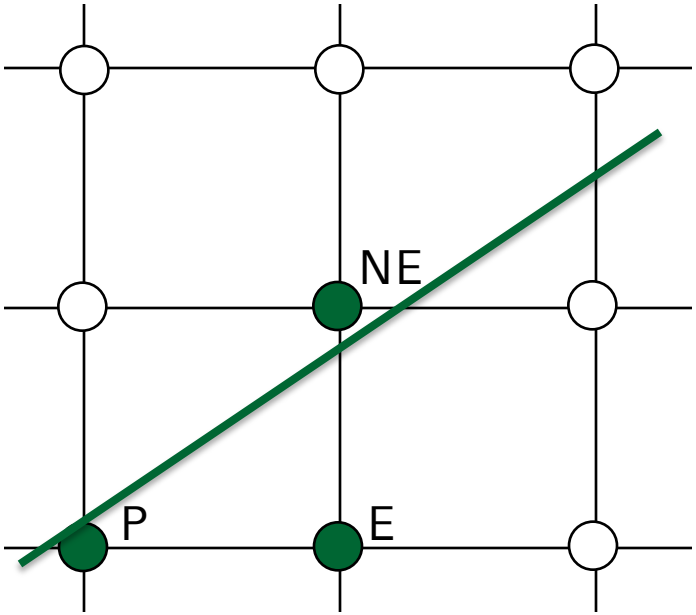
- auch Mittelpunktschema für Linien, Midpoint-Line-Algorithmus genannt
- Lösen der Probleme der anderen Ansätze, also
  - ausschließlich Integer-Arithmetik
  - keine Division, keine komplizierten Multiplikationen
- Einschränkungen:
  - Pixel liegen auf Integer-Gitter (keine wirkliche Einschränkung)
  - Anstieg der Linie kleiner 1, d. h. Linie befindet sich im 1. Oktanten ($0° \leq \alpha \leq 45°$) (auch keine Einschränkung, die eine Verallgemeinerung ausschließt)



Vorüberlegungen

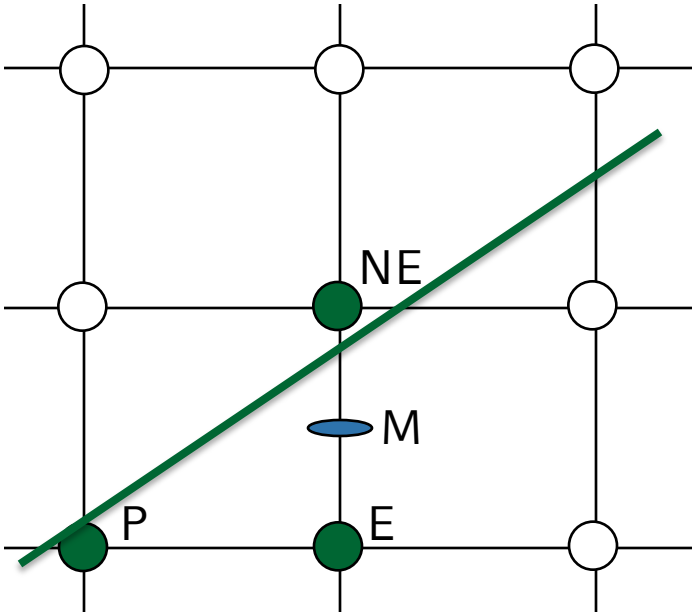
- implizite Geradengleichung $F(x, y) = ax + by + c = 0$
- liefert eine Antwort auf die Frage, auf welcher Seite der Geraden ein gegebener Punkt $P(x_1, y_1)$ liegt
  - ist $F(x_1, y_1) > 0 \rightarrow$ P liegt unterhalb der Geraden
  - ist $F(x_1, y_1) < 0 \rightarrow$ P liegt oberhalb der Geraden
  - ist $F(x_1, y_1) = 0 \rightarrow$ liegt auf der Geraden
- dabei ist:
  - $a = \Delta y$
  - $b = -\Delta x$

Mittelpunktschema für Linien



- Annahme: Steigung der Linie zwischen 0 und 1
- gerade gesetztes Pixel ist P
- Welches der beiden Pixel E oder NE muss als nächstes gesetzt werden?
- Entscheidung, welches der beiden Pixel näher am Schnittpunkt der Linie mit dem Pixelgitter liegt
- das für jedes Pixel

Mittelpunktschema für Linien

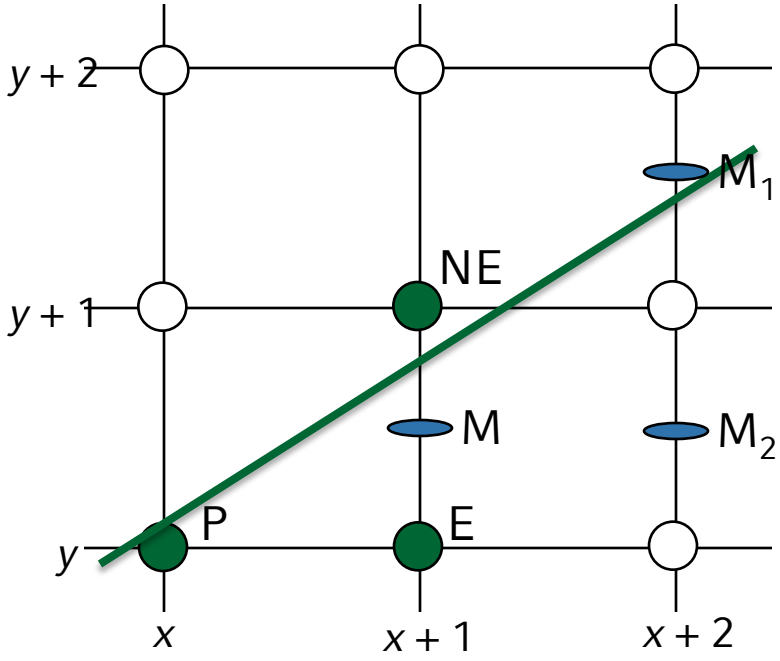


- leichter zu bestimmen: liegt der Mittelpunkt M zwischen E und NE oberhalb oder unterhalb der Linie
- momentane Position P(x, y)
- Koordinaten von M: M($x + 1$, $y + ½$)
- F(M) = F($x + 1$, $y + ½$)
- wenn F(M) > 0, dann geht Linie über M vorbei → NE nächstes Pixel
- wenn F(M) ≤ 0, dann geht Linie unter M vorbei → E nächstes Pixel
- wähle d = F(M)=F($x + 1$, $y + ½$) als *Entscheidungsvariable*

Mittelpunktschema für Linien

- Berechnen von F(M) für ein Pixel, Zeichnen des Pixels, Berechnen von F(M) für das neue Pixel
- Fragestellung: Kann F(M) für das neue Pixel aus dem F(M) für das gerade berechnete Pixel bestimmt werden?
  - möglicherweise einfachere Berechnungen
  - schrittweise Weitergehen von Pixel i zu Pixel $i + 1$
- Inkrementeller Algorithmus

Mittelpunktschema für Linien



- Fall 1: NE als nächstes Pixel ausgewählt
- $F(M) = F((x + 1), (y + ½))$
  $= a(x + 1) + b(y + ½) + c$
- dann ist M_1 der nächste Mittelpunkt
- $F(M_1) = F((x + 2), (y + 3/2))$
  $= a(x + 2) + b(y + 3/2) + c$
- Differenz zwischen beiden:
- $F(M_1) - F(M) = a + b$

→ $F(M_1) = F(M) + a + b$

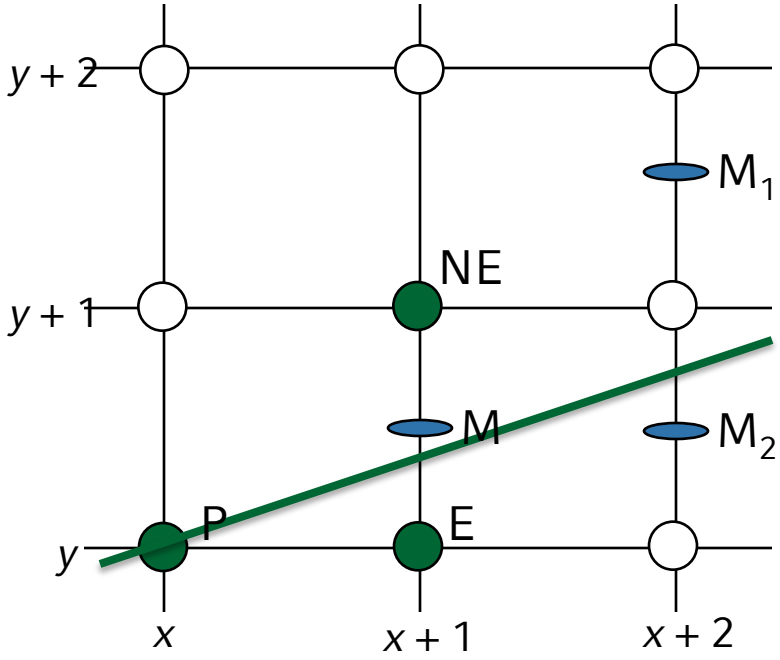
- mit $a = \Delta y$ und $b = -\Delta x$

→ $F(M_1) = F(M) + \Delta y - \Delta x$

→ $d = d + \Delta y - \Delta x$

→ $\Delta_{NE} = \Delta y - \Delta x$

Mittelpunktschema für Linien



- Fall 2: E als nächstes Pixel ausgewählt
- $F(M) = F((x + 1), (y + ½))$
  $= a(x + 1) + b(y + ½) + c$
- dann ist M_2 der nächste Mittelpunkt
- $F(M_2) = F((x + 2), (y + ½))$
  $= a(x + 2) + b(y + ½) + c$
- Differenz zwischen beiden:
- $F(M_2) - F(M) = a$

→ $F(M_2) = F(M) + a$

- mit $a = \Delta y$:

→ $F(M_1) = F(M) + \Delta y$

→ $d = d + \Delta y$

→ $\Delta_E = \Delta y$

Mittelpunktschema für Linien

- Idee für den Algorithmus:
  - beginne beim ersten Pixel
  - berechne die Entscheidungsvariable d=F(M) für dieses Pixel
  - entscheide, ob E oder NE das nächste Pixel ist
  - zeichne das Pixel
  - aktualisiere d entsprechend der Wahl von E oder NE
  - wiederhole, solange der Endpunkt nicht erreicht ist
- Offene Frage: Initialer Wert von d?

Mittelpunktschema für Linien

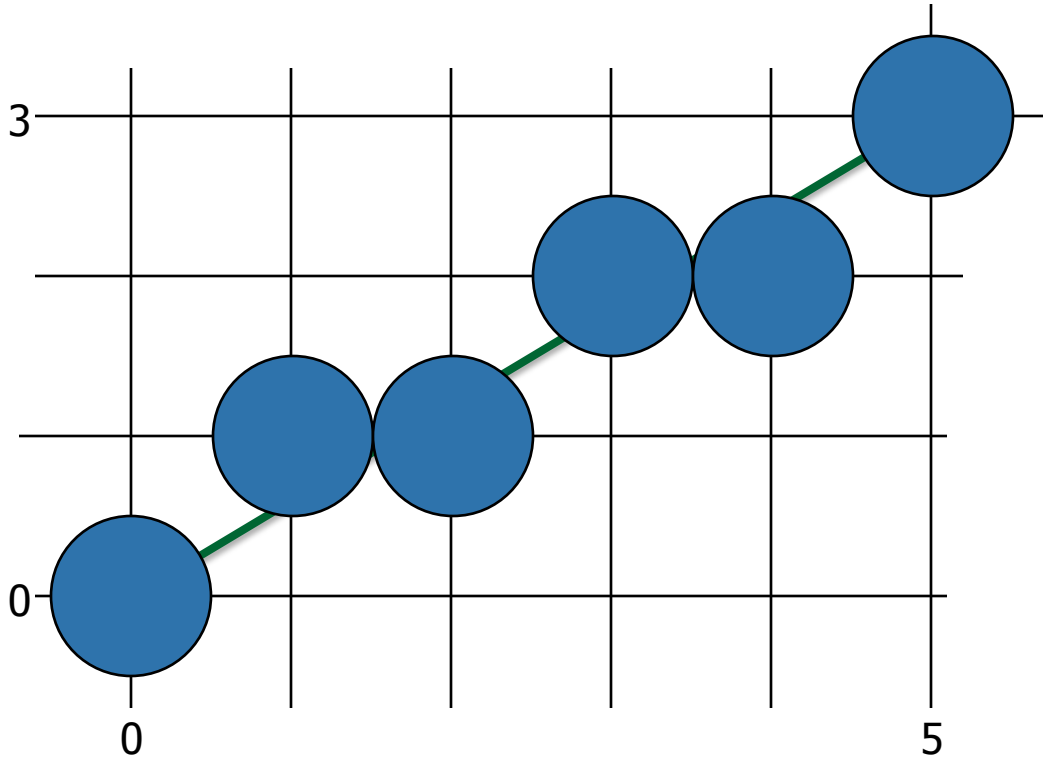
- Initialer Wert von d?
  - Startpunkt der Linie sei (x_0,y_0)
  - erster Mittelpunkt ist dann

$$\begin{aligned} F((x_0+1), (y_0+½)) &= a(x_0+1) + b(y_0+½) + c \\ &= ax_0 + a + by_0 + ½\,b + c \\ &= F(x_0, y_0) + a + ½\,b \end{aligned}$$

  - Startpunkt (x_0, y_0) liegt auf der Linie, also $F(x_0, y_0)=0$
  - Initialer Wert der Entscheidungsvariable $d = a + ½\,b$
- aber: keine Integer-Werte
  - Es interessiert nur das *Vorzeichen* von d.
  - Multiplikation von $F(x, y)$ mit 2 hat keinen Effekt auf das Vorzeichen
  - Initialer Wert der Entscheidungsvariable $d = 2a + b$
  - Inkremente: $\Delta_{NE} = 2\Delta y - 2\Delta x$ und $\Delta_E = 2\Delta y$

```
void midpoint-line(int xo, int yo, int xe, int ye, int color)
{
    int dx = xe - xo;
    int dy = ye - yo;
    int d = 2 * dy - dx;
    int incrE = 2 * dy;
    int icrNE = 2 * (dy - dx);
    int x = xo;
    int y = yo;
    setPixel(xo, yo, color);
    while (x<xe) {
          if (d<=0) {
                    d+=incrE;
                    x++;
          } else {
                    d += incrNE;
                    x++; y++;
          }
          setPixel(x, y, color);
    }
}
```

- Beispiel:
  Linie von (0,0) nach (5,3)
- Initiale Werte:
  $\Delta x = 5$, $\Delta y = 3$,
  $\Delta_E = 6$, $\Delta_{NE} = -4$
- Entscheidungsvariable d = 1



x=0, y=0, setPixel(0,0,color)
d=1>0 → Auswahl NE
→ $d=d+D_{NE}=1-4=-3$
→ x=x+1=1, y=y+1=1
x=1, y=1, setPixel(1,1,color)
d=-3<0 → Auswahl E
→ $d=d+D_E=-3+6=3$
→ x=x+1=2, y=1
x=2, y=1, setPixel(2,1,color)
d=3>0 → Auswahl NE
→ $d=d+D_{NE}=3-4=-1$
→ x=x+1=3, y=y+1=2
x=3, y=2, setPixel(3,2,color)
d=-1<0 → Auswahl E
→ $d=d+D_E=-1+6=5$
→ x=x+1=4, y=2
x=4, y=2, setPixel(4,2,color)
d=5>0 → Auswahl NE
→ $d=d+D_{NE}=5-4=1$
→ x=x+1=5, y=y+1=3
x=5, y=3, setPixel(5,3,color)

Zusammenfassung

- Algorithmus arbeitet ausschließlich mit:
  - Integer-Arithmetik
  - Additionen
- Inkrementeller Algorithmus, d. h.
  - Positionen der Pixel werden nach und nach aus denen der vorherigen berechnet
  - kleine Schleifenkörper
  - Implementierung in Hardware möglich

Probleme und Erweiterungen

- Beschränkung des Anstiegs auf $0° \leq m \leq 45°$
  - kein wirkliches Problem
  - Algorithmus kann erweitert werden auf alle möglichen Anstiege (Vertauschen von *x* und *y*, Vorzeichen, etc.)
- lange Linien werden Pixel für Pixel gezeichnet
  - Erweiterungen, die gleich zwei Pixel im Voraus berechnen
  - Linie setzt sich aus „Mustern“ zusammen → Muster finden und als Gesamtheit setzen

LFE Medieninformatik • Prof. Dr. Ing. Axel Hoppe

Algorithmen der Rastergrafik

Rastern von Kreisen

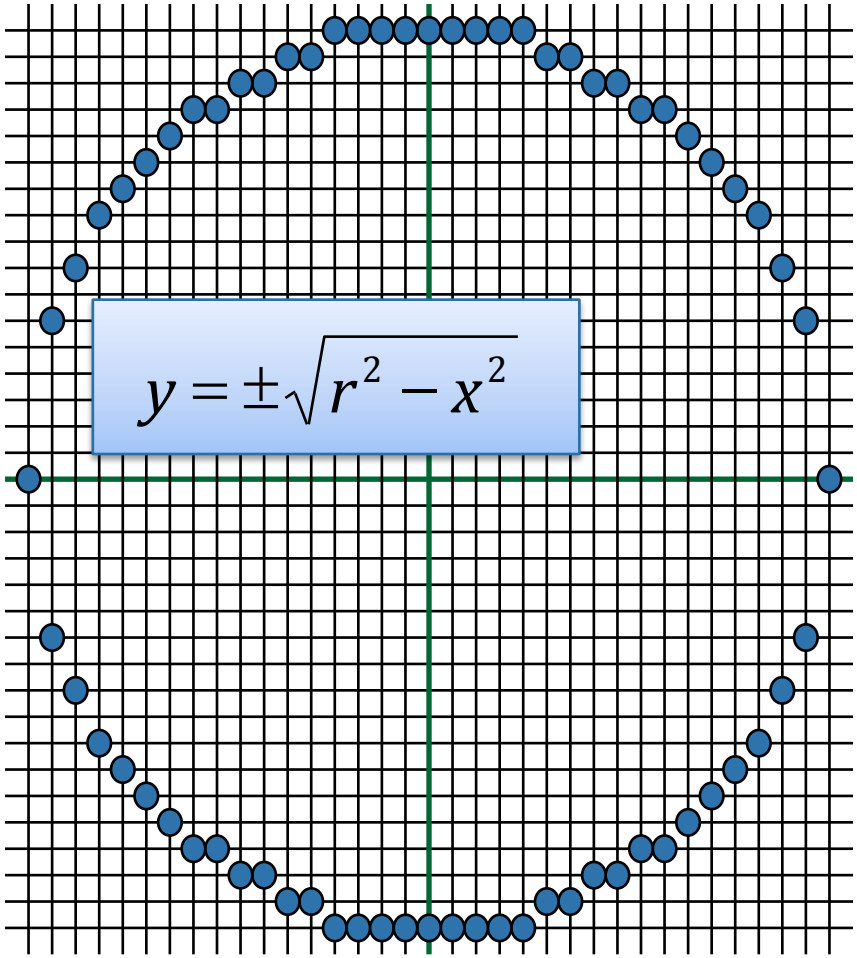
Kreise mathematisch

- Gegeben: Zentrum und Radius
- allgemeine Kreisgleichung für einen Kreis mit dem Zentrum im Ursprung: $F(x, y) = x^2 + y^2 = r^2$

  alle Punkte (x, y), die diese Gleichung erfüllen, liegen auf dem Kreis
- Kreisgleichung, wenn Mittelpunkt nicht im Ursprung liegt, sondern an der Position (x_c, y_c): $F(x, y) = (x - x_c)^2 + (y - y_c)^2 = r^2$

Naiver Algorithmus

- Auflösen der Kreisgleichung nach *y*:
- Schleife über alle *x* von $-r$ bis *r*
- Probleme:
  - aufwändige Berechnung
    - Wurzel
    - Quadrate
  - Lücken in der Umgebung von $|x| = r$

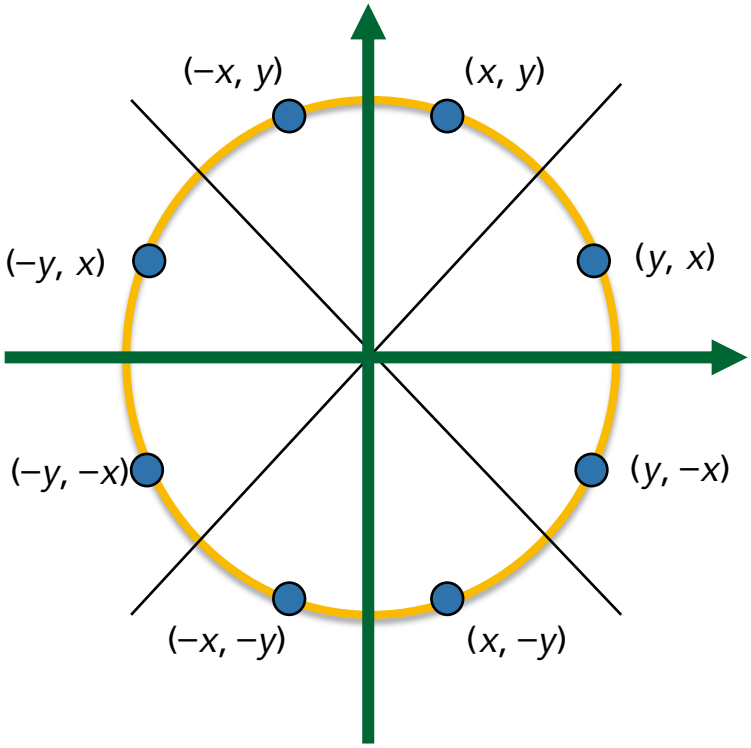


Parametrischer Algorithmus

- löst Problem der Lücken in den Bereichen nahe $|x| = r$
- Idee: Übergang zur Kreisgleichung in Parameterform

  $x = r \cos \varphi$ sowie $y = r \sin \varphi$
- Schleife über φ in gleichen, geeignet groß gewählten Schritten
- Probleme hier:
  - Berechnung der Winkelfunktionen nicht akzeptabel
  - Lösung über Lookup-Tabellen denkbar, jedoch immer noch ineffizient

Symmetrie des Kreises



Verringerung der Anzahl der Rechenoperationen durch Nutzung der Symmetrie des Kreises

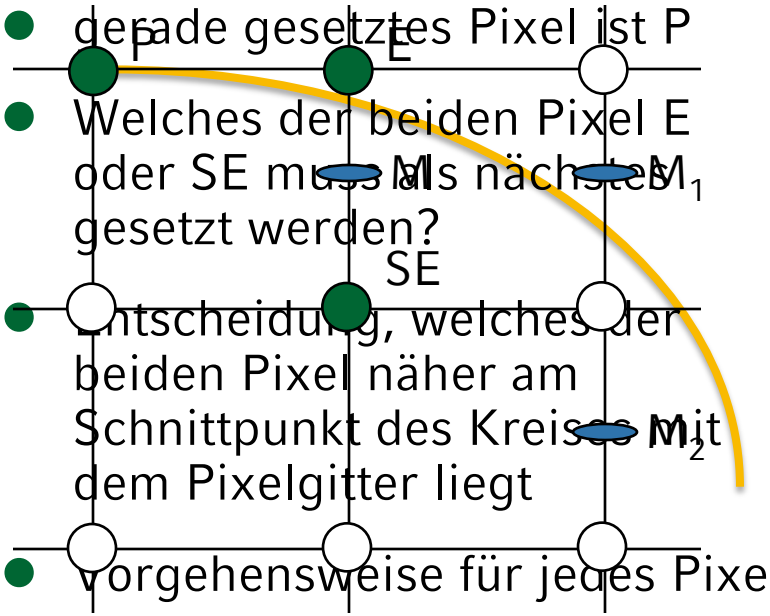
```
void circlePoints( int x, int y )
{
    setPixel( x, y );
    setPixel( y, x );
    setPixel( y, -x );
    setPixel( x, -y );
    setPixel( -x, -y );
    setPixel( -y, -x );
    setPixel( -y, x );
    setPixel( -x, y );
}
```

Bresenham-Algorithmus

- Herangehensweise wie bei Geraden
- Ziele:
  - Integer-Arithmetik
  - Eliminieren aufwändiger Operationen
  - Finden eines inkrementellen Algorithmus
- Basis: implizite Gleichung des Kreises
  - Die Funktion $F(x, y) = x^2 + y^2 - r^2$
  - ist Null für Punkte auf der Kreislinie
  - ist negativ für Punkte innerhalb des Kreises
  - ist positiv für Punkte außerhalb des Kreises

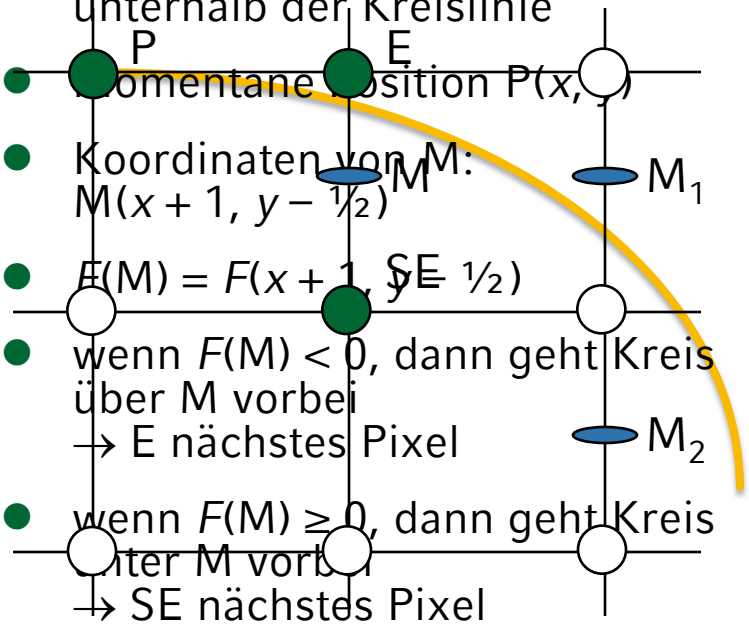
Grundidee

- Berechnung für den zweiten Oktanten
- derade gesetztes Pixel ist P
- Welches der beiden Pixel E oder SE muss als nächstes gesetzt werden?
- Entscheidung, welches der beiden Pixel näher am Schnittpunkt des Kreises mit dem Pixelgitter liegt
- Vorgehensweise für jedes Pixel
- erzeuge alle anderen Oktanten über Symmetrie



Grundidee

- Liegt M innerhalb oder außerhalb des Kreises entspricht in diesem Oktanten: liegt M ober- oder unterhalb der Kreislinie
- Momentane Position P(x, y)
- Koordinaten von M: M($x + 1$, $y - ½$)
- F(M) = $F(x + 1, y - ½)$
- wenn F(M) < 0, dann geht Kreis über M vorbei
  → E nächstes Pixel
- wenn F(M) ≥ 0, dann geht Kreis unter M vorbei
  → SE nächstes Pixel
- wähle d = F(M) = $F(x + 1, y - ½)$ als **Entscheidungsvariable**



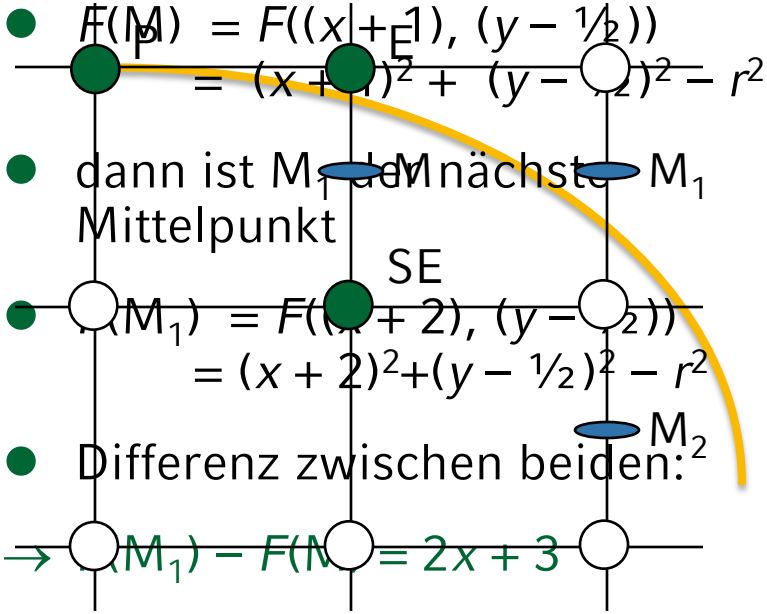
Grundidee

- Fall 1: $d < 0 \rightarrow$ E als nächstes Pixel ausgewählt
- $F(M) = F((x + 1), (y - ½))$
  $= (x + 1)^2 + (y - ½)^2 - r^2$
- dann ist M_1 der nächste Mittelpunkt
- $F(M_1) = F((x + 2), (y - ½))$
  $= (x + 2)^2 + (y - ½)^2 - r^2$
- Differenz zwischen beiden:

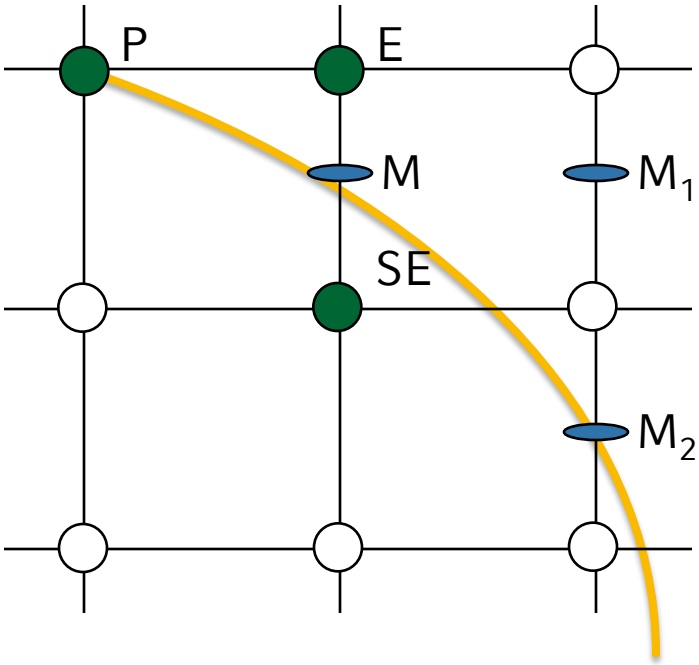
$\rightarrow F(M_1) - F(M) = 2x + 3$

$\rightarrow d = d + (2x + 3)$

$\rightarrow \Delta_E = (2x + 3)$



Grundidee



- Fall 2: $d \geq 0 \rightarrow$ SE als nächstes Pixel ausgewählt
- $F(M) = F((x + 1), (y - ½))$
  $= (x + 1)^2 + (y - ½)^2 - r^2$
- dann ist M_2 der nächste Mittelpunkt
- $F(M_2) = F((x + 2) = (y - {}^3/_2))$
  $= (x + 2)^2 + (y - {}^3/_2)^2 - r^2$
- Differenz zwischen beiden:

$\rightarrow F(M_2) - F(M) = 2x - 2y + 5$

$\rightarrow d = d + (2x - 2y + 5)$

$\rightarrow \Delta_{SE} = (2x - 2y + 5)$

Grundidee

- Initialer Wert von d
  - erstes Pixel des Kreises (besser: des berechneten Kreisausschnittes) ist $P(0, r)$
  - Erster Mittelpunkt ist damit $M(1, r - ½)$

    $$d = F(1, r - ½) = 1^2 + (r - ½)^2 - r^2 = {}^5/_4 - r$$
- allerdings: kein Integer-Wert
  - neue Entscheidungsvariable $h = d - {}^1/_4$
  - Anfangswert: $h_{start} = 1 - r$
  - Entscheidungskriterium: $d < 0$ wird zu $h < -¼$
  - da die Berechnung von h mit ganzzahligen Werten beginnt und h nur ganzzahlig (durch Δ_E, Δ_{SE}) inkrementiert wird, kann $h \leftarrow ¼$ durch $h < 0$ ersetzt werden

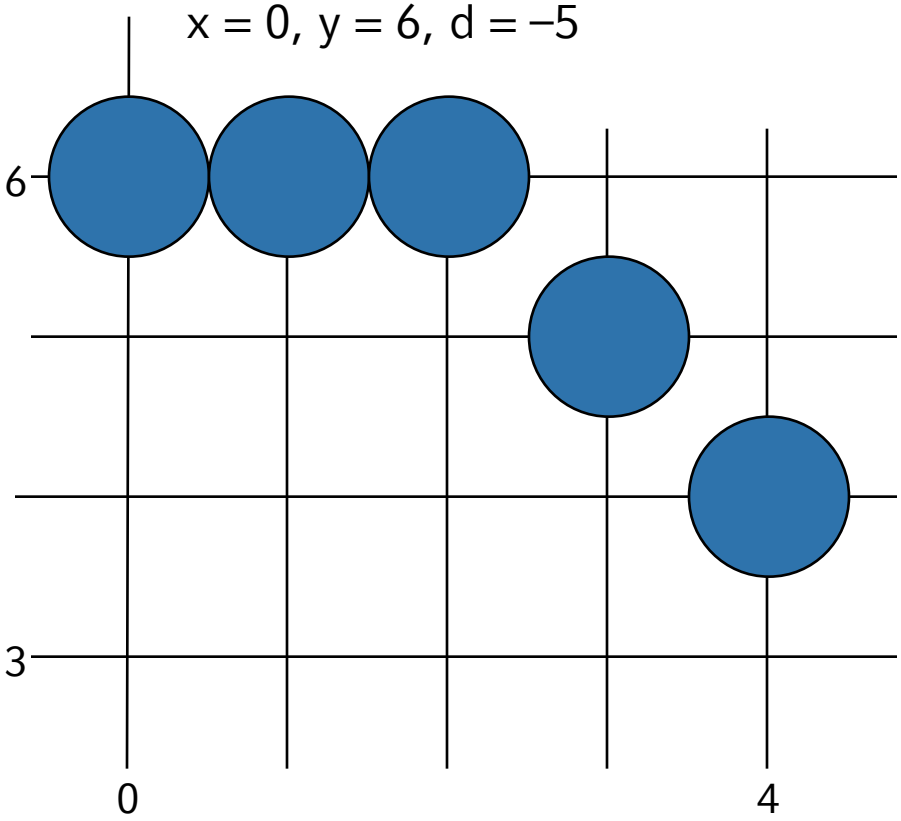
Algorithmus

```
void MidpointCircle (int radius)
{
    int x = 0; /* Initialisierung */
    int y = radius;
    int d = 1 - radius;
    circlePoints (x, y);

    while (y > x) {
           if (d < 0)
           { /* Auswahl von E */
                      d = d + 2  x +3;
                      x = x + 1
           } else { /* Auswahl von SE */
                      d := d + 2  (x-y) + 5;
                      x := x + 1;
                      y := y - 1;
           }
    circlePoints (x, y)
    }
}
```

Beispiel

- Beispiel:
  Kreis um (0,0) Radius 6
- Initialisierung:
  x = 0, y = 6, d = –5



x=0, y=6, d=-5, circlePoints(0,0)

d=-5<0 → Auswahl E
→ d=d+2x+3=-2
→ x=x+1=1

circlePoints(1,6)

d=-2<0 → Auswahl E
→ d=d+2x+3=5
→ x=x+1=2

circlePoints(2,6)

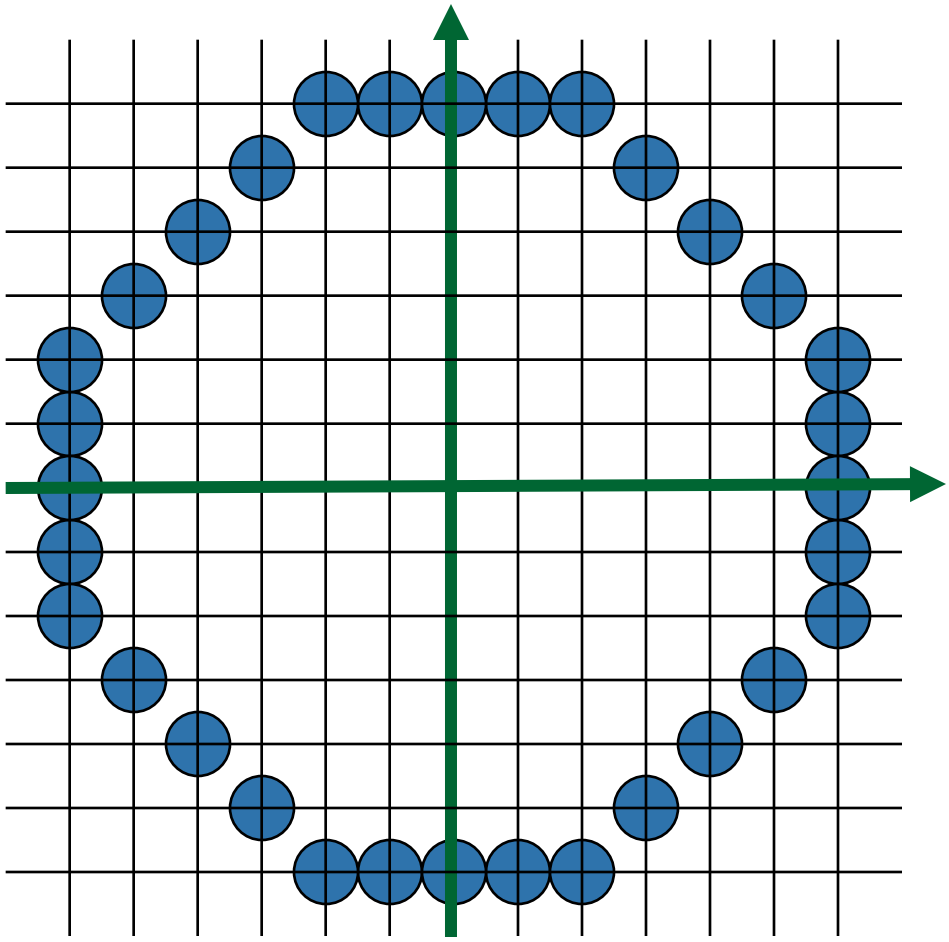
d=5>0 → Auswahl SE
→ d=d+2(x-y)+5=2
→ x=x+1=3, y=y-1=5

circlePoints(3,5)

d=5>0 → Auswahl SE
→ d=d+2(x-y)+5=3
→ x=x+1=4, y=y-1=4

circlePoints(4,4)

Beispiel



(x,y)–(y,x)–(y,-x)–(x,-y)–(-x,-y)–(-y,-x)–(-y,x)–(-x,y)

(0,6)–(6,0)–(6,-0)–(0,-6)–(-0,-6)–(-6,-0)–(-6,0)–(-0,6)
(1,6)–(6,1)–(6,-1)–(1,-6)–(-1,-6)–(-6,-1)–(-6,1)–(-1,6)
(2,6)–(6,2)–(6,-2)–(2,-6)–(-2,-6)–(-6,-2)–(-6,2)–(-2,6)
(3,5)–(5,3)–(5,-3)–(3,-5)–(-3,-5)–(-5,-3)–(-5,3)–(-3,5)
(4,4)–(4,4)–(4,-4)–(4,-4)–(-4,-4)–(-4,-4)–(-4,4)–(-4,4)

Erweiterungen

- Algorithmus ist relativ effizient, weil:
  - inkrementeller Algorithmus
  - Integer-Arithmetik
  - nur ein Achtel des Kreises muss berechnet werden
- Aber:
  - immer noch Multiplikationen notwendig, um die Inkremente zu berechnen
  - zur Erinnerung: $\Delta_E = (2x + 3)$ und $\Delta_{SE} = (2x - 2y + 5)$
  - Linienalgorithmus arbeitet mit konstanten Inkrementen
- Geht das hier auch?
  - Idee: Berechnen der Inkremente Δ_E und Δ_{SE} auch inkrementell
  - dazu: betrachten nicht nur einen Pixelschritt, sondern zwei

Differenzen zweiter Ordnung

- Fall 1: im aktuellen Schritt wurde E gewählt
  - Bewertungspunkt (aktuelles Pixel) wandert von (x, y) nach $(x + 1, y)$
  - Inkremente am „alten“ Pixel (x, y) waren:
    - $\Delta_{E_{alt}} = 2x + 3$
    - $\Delta_{SE_{alt}} = 2x - 2y + 5$
  - Inkremente am neuen Pixel wären:
    - $\Delta_{e_{neu}} = 2(x + 1) + 3$
    - $\Delta_{SE_{neu}} = 2(x + 1) - 2y + 5$
  - Differenzen zwischen beiden:
    - $\Delta_{e_{neu}} - \Delta_{E_{alt}} = 2$
    - $\Delta_{SE_{neu}} - \Delta_{SE_{alt}} = 2$

Differenzen zweiter Ordnung

- Fall 2: im aktuellen Schritt wurde SE gewählt
  - Bewertungspunkt (aktuelles Pixel) wandert von (x, y) nach $(x + 1, y - 1)$
  - Inkremente am „alten“ Pixel (x, y) waren
    - $\Delta_{E_{alt}} = 2x + 3$
    - $\Delta_{SE_{alt}} = 2x - 2y + 5$
  - Inkremente am „neuen“ Pixel wären:
    - $\Delta_{E_{neu}} = 2(x + 1) + 3$
    - $\Delta_{SE_{neu}} = 2(x + 1) - 2(y - 1) + 5$
  - Differenzen zwischen beiden:
    - $\Delta_{E_{neu}} - \Delta_{E_{alt}} = 2$
    - $\Delta_{SE_{neu}} - \Delta_{SE_{alt}} = 4$

Algorithmus

```
void MidpointCircle(int radius)
{
     int x = 0;
     int y = radius;
     int d = 1 - radius;
     int deltaE = 3;
     int deltaSE = -2 * radius + 5;
     circlePoints(x, y); /* draws 8 points */
     while (y > x) {
             if (d < 0) { /* select E */
                          d += deltaE;
                          deltaE += 2;
                          deltaSE += 2;
             } else { /* select SE */
                          d += deltaSE;
                          deltaE += 2;
                          deltaSE += 4;
                          y--;
             }
             x++;
             circlePoints(x, y);
     }
}
```

LFE Medieninformatik • Prof. Dr. Ing. Axel Hoppe

Algorithmen der Rastergrafik

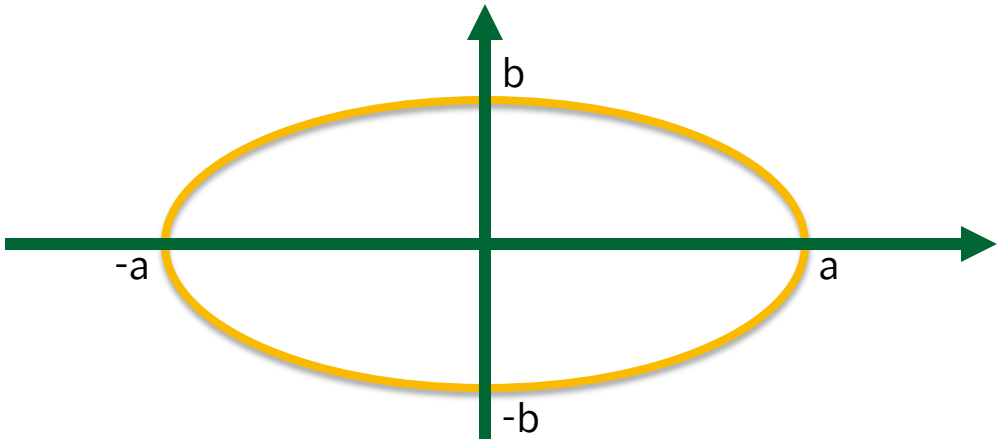
Rastern von Ellipsen

Grundlegendes

- Gleichung der Ellipse mit Mittelpunkt im Ursprung

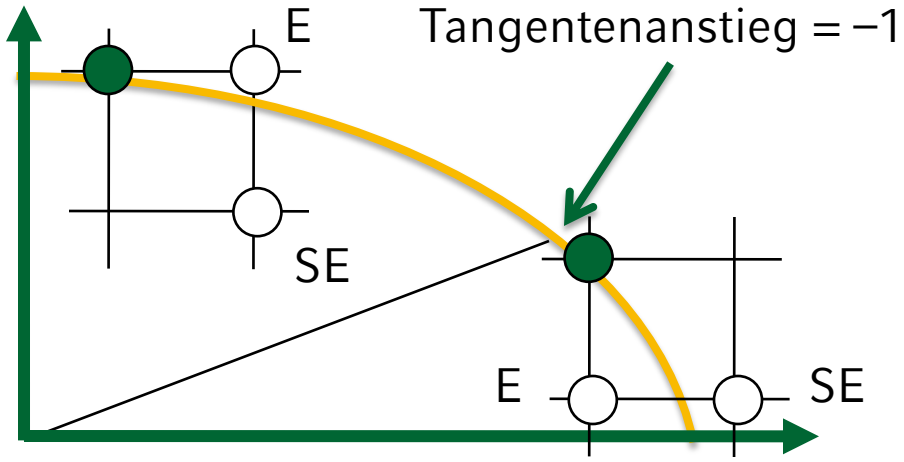
$$F(x, y) = b^2 x^2 + a^2 y^2 - a^2 b^2 = 0$$

- hier nur „achsenparallele“ Ellipsen
- Algorithmus für ersten Quadranten reicht aus, Rest durch Spiegelungen
- Ziel: inkrementeller Mittelpunkt-Algorithmus



Herleitung für den Algorithmus

- Schwierigkeit: zwei Gebiete mit
  - entweder Auswahl zwischen E und SE
  - oder Auswahl zwischen SE und S
- Grenze: Punkt, an dem die Ellipse den Anstieg –1 hat
- hier ohne Herleitung:
  - Übergang, der erste Mittelpunkt, für den gilt:
  - $y^2(y - ½) \leq b^2(x - 1)$



Herleitung für Algorithmus

- Gebiet 1: Mittelpunkt liegt zwischen E und SE

$$d_{alt} = F(x+2,\ y-½) = b^2(x+1)^2 + a^2(y-½)^2 - a^2b^2$$

  - Wahl von E als nächstes Pixel:

$$d_{neu} = F(x+2,\ y-½) = b^2(x+2)^2 + a^2(y-½)^2 - a^2b^2$$

$$\Delta_E = d_{neu} - d_{alt} = b^2(2x+3)$$

  - Wahl von SE als nächstes Pixel:

$$d_{neu} = F(x+2,\ y-3/2) = b^2(x+2)^2 + a^2(y-3/2)^2 - a^2b^2$$

$$\Delta_E = d_{neu} - d_{alt} = b^2(2x+3) + a^2(-2y+2)$$

- Gebiet 2: Mittelpunkt liegt zwischen SE und S – analog

Herleitung für Algorithmus

- Initialer Wert für d
  - es ist $(0, b)$ das erste Pixel, damit $(1, b - ½)$ der erste Mittelpunkt

    $$F(1, b - ½) = b^2 + a^2(b - ½)^2 - a^2b^2 = b^2 + a^2(-b + ¼) = d$$

  - Beim Wechsel der Gebiete ist ein neuer Initialwert für d zu berechnen, er liegt zwischen S und SE bei $(x + ½, y - 1)$

    $$F(x + ½, y - 1) = b^2 (x + ½)^2 + a^2 (y - 1)^2 - a^2b^2 = d$$

  - sonst Algorithmus wie beim Kreis
  - Ausnutzen der vierfachen Symmetrie

Algorithmus

```
void MidpointEllipse(int a, int b, int color)
{
     int x = 0; int y = b;
     double d1 = b*b-a*a*b-(a*a/4);
   ellipsePoints( x, y );
     while( a*a*(y-0.5) > b*b*(x-1)) {
             if d1 < 0 { d1 += b*b*(2*x+3)}
             else {
                          d1 += (b*b*(2*x+3)+a*a*(-2*y+2));
                          y--;
             }
             x++
             ellipsePoints( x, y );
     }
     d2 = b*b*(x+.5)*(x+.5)+a*a*(y-1)*(y-1)-a*a*b*b;
     while(y > 0) {
             if d2 < 0 {
                          d2 += b*b*(2*x+2)+a*a*(-2*y+3);
                          x++;
             } else { d2 += a*a*(-2*y+3) }
             y--;
             ellipsePoints( x, y );
     }
}
```

Zusammenfassung Ellipse

- Algorithmus kann natürlich noch erweitert werden:
  - Ellipsen nicht im Ursprung (trivial)
  - Ellipsen nicht achsenparallel (schwierig)
  - Differenzen zweiter Ordnung (trivial)
  - Float-Berechnungen (schwieriger)
- aber auch hier
  - inkrementeller Algorithmus
  - relativ einfache Berechnungen
- Aufgrund der Eigenschaften der Ellipsen machen sich eigenständige Algorithmen für sehr schmale Ellipsen (*a* oder *b* sehr klein) bezahlt.

Zusammenfassung Rasteralgorithmen

- werden oft aufgerufen, daher besondere Anforderungen:
  - schnell
  - einfach
  - Integer-Arithmetik
- inkrementelle Algorithmen
  - nach dem Mittelpunkt-Schema
  - Linien, Kreise, Ellipsen
  - erweiterbar für allgemeine Lagen der Primitive
- Beispiel für Herleitung von Algorithmen in der Rastergraphik

Offene Fragen

- Rasterung allgemeiner Kurven → würde Rahmen sprengen
- Bisher waren alle Linien genau ein Pixel breit, was passiert, wenn man dickere Linien haben möchte? → nächstes Kapitel
- Beim Rastern wurden bisher alle Pixel mit einer Farbwert gesetzt, das führt zu unschönen Artefakten – Aliasing – Wie kann man dem begegnen? → Antialiasing – übernächstes Kapitel
- Was passiert mit dem Inneren von Figuren – Binnenstrukturen? → Füllen – überübernächstes Kapitel

Rasteralgorithmen

Breite bzw. dicke Primitive

Breite Primitive

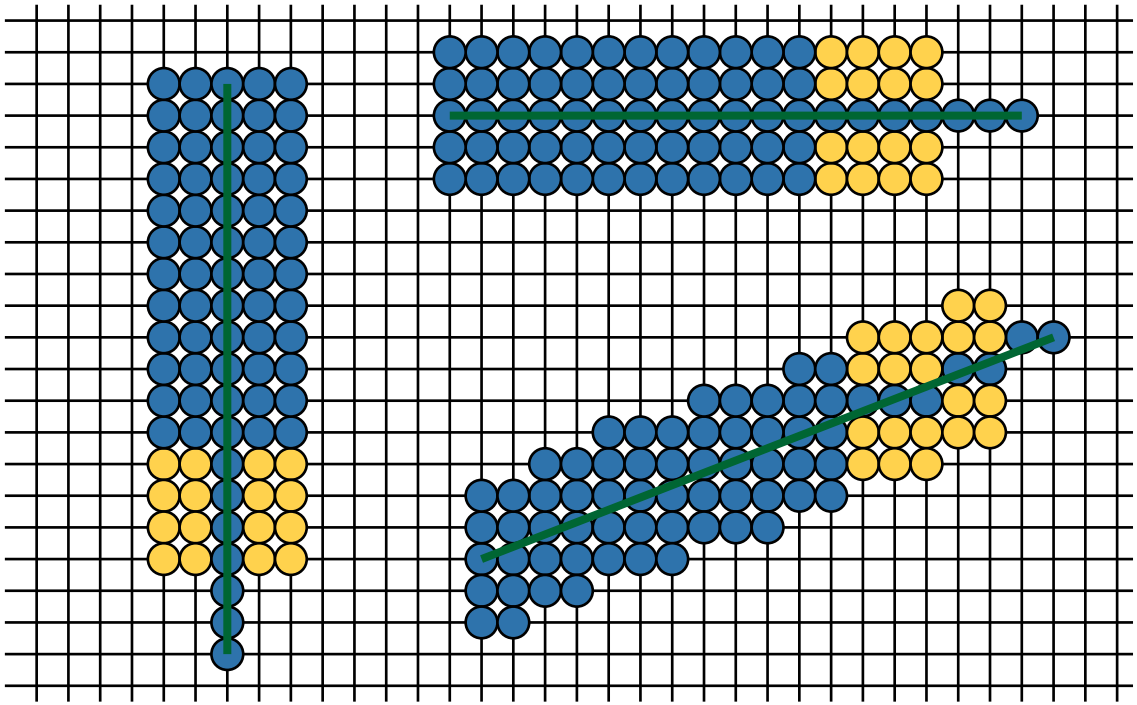
- bisher:
  - Alle Linien sind genau 1 Pixel breit.
  - vorgestellte Algorithmen eignen sich sehr gut dafür
- aber:
  - Wie wird verfahren, wenn die Linien dicker sein sollen?
  - Welche Probleme treten dabei auf?
  - Wie kann man diese lösen?

Breite Primitive

- *Ansatz:* anstelle eines Pixels werden an jedem Punkt mehrere Pixel gezeichnet – Fragen:
  - Wie sieht die Pixelgruppe aus, die gezeichnet werden soll (rund, rechteckig, etc.)
  - Wie ist die Orientierung der Pixelgruppe – insbesondere, wenn sie nicht rund ist?
  - Wie sehen die Linienenden aus?
  - Wie sehen Knotenpunkte aus, an denen zwei Linien aneinanderstoßen?
- Drei Methoden für breite Primitive:
  - Pixelwiederholung
  - Bewegliche Pinsel
  - Ränder zeichnen und dazwischen füllen

Pixelwiederholung

- Duplizieren von Pixeln in Zeilen oder Spalten
- funktioniert gut für Linien
  - spaltenweises Duplizieren für Anstieg $-1 < m < 1$
  - zeilenweises Duplizieren sonst

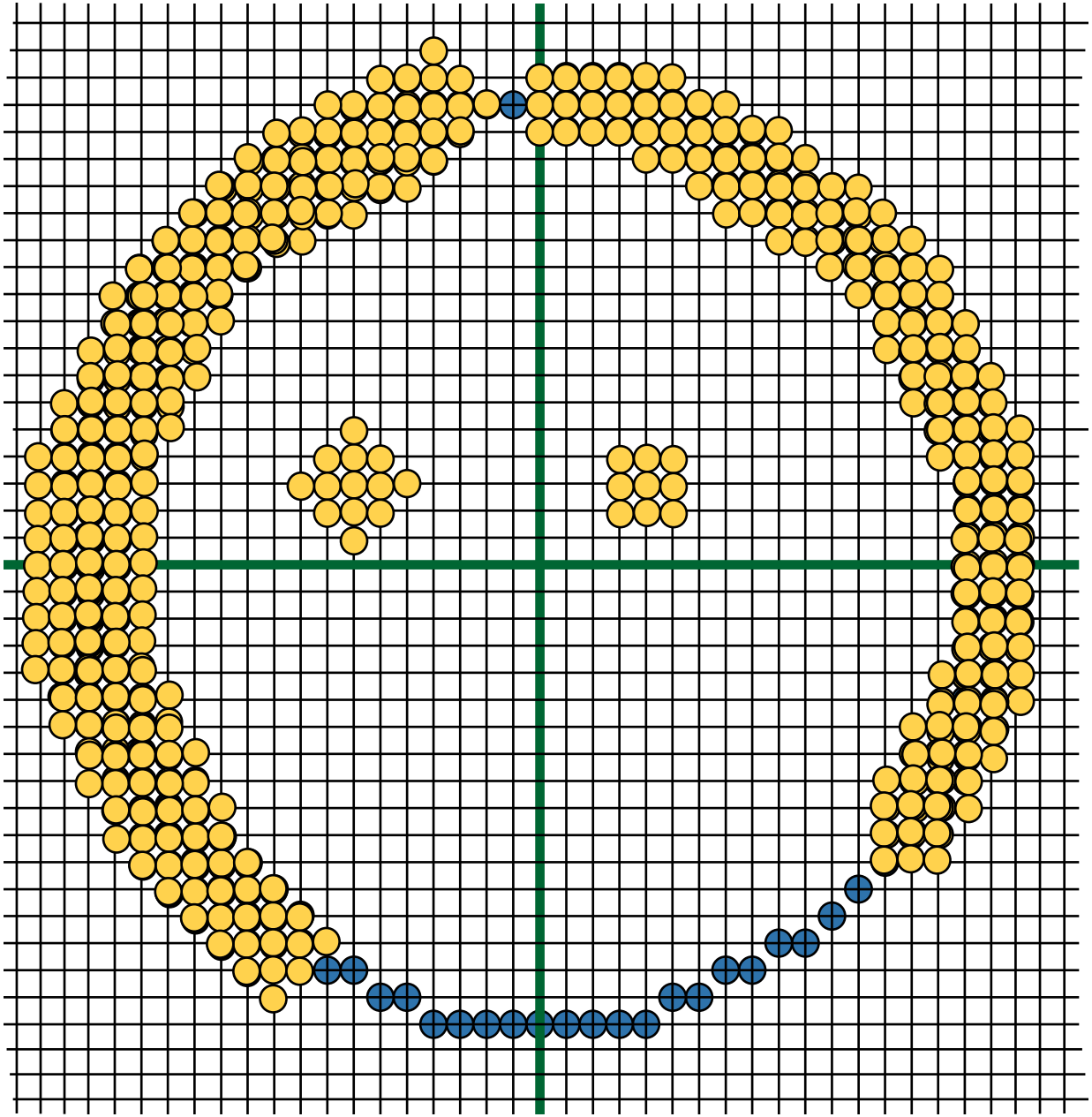


Probleme

- Linienenden sind immer horizontal oder vertikal gerade.
- Horizontale und vertikale Linien haben eine sichtbar unterschiedliche Breite im Vergleich zu schrägen Linien.
  - Sei t die Breite einer Linie (in Pixeln)
  - Horizontale und vertikale Linien haben die Breite t.
  - 45°-Linien haben eine Breite von $t/\sqrt{2}$.
- Geradzahlige Linienbreiten
  - Welches Pixel liegt genau auf der berechneten Linie?
  - Ober- und Unterkante (rechte/linke Kante) fallen aus

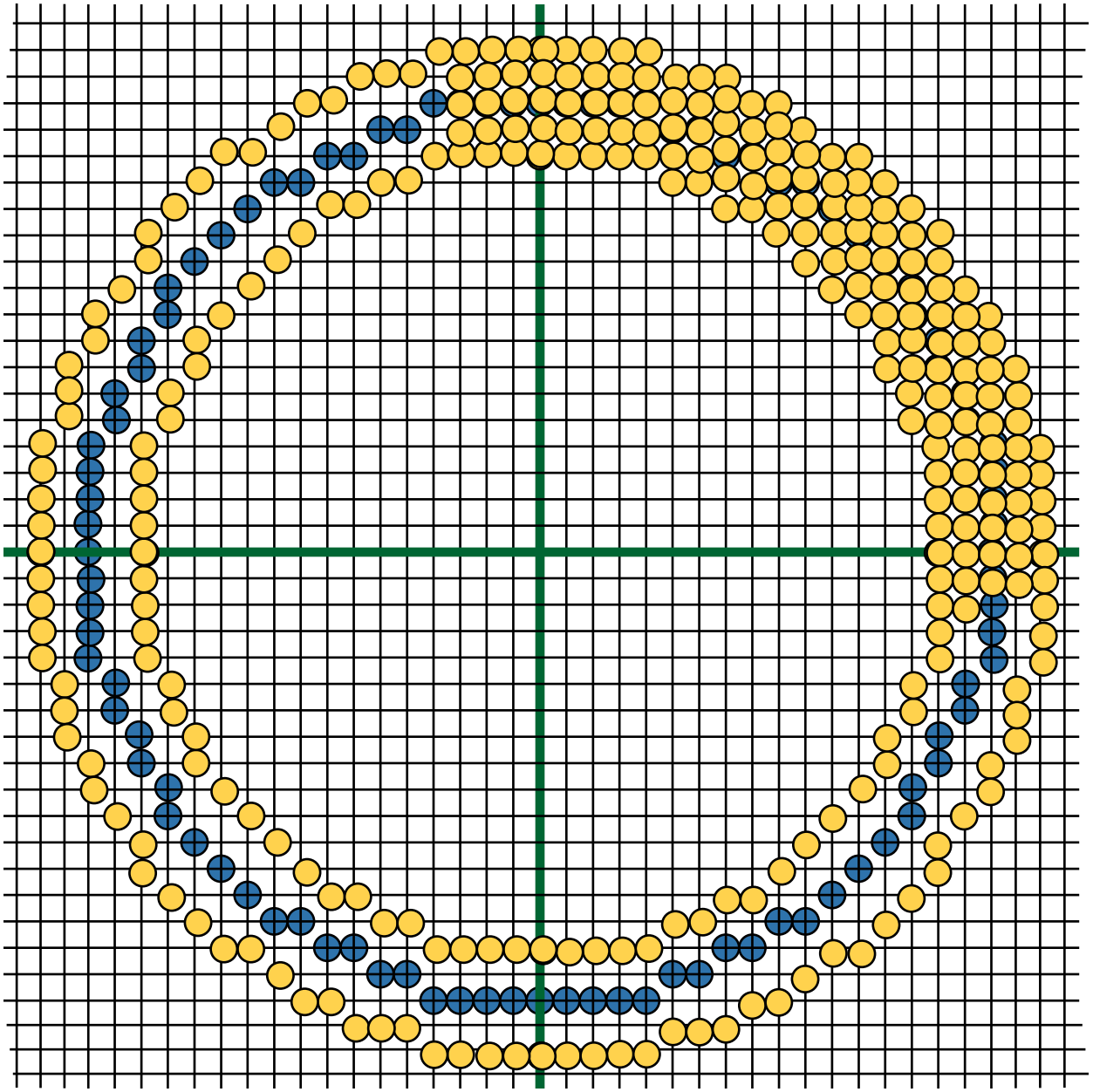
Bewegliche Pinsel

- Pinsel hinterlässt ein spezifisches Pixelmuster (Footprint)
- Bewegen des Pinsels entlang der gerasterten Linie und Zeichnen des Musters an jeder Stelle
- Probleme:
  - Linienenden nicht an der genauen Pixelposition
  - Unterschiedliche Breite der Linien bei unterschiedlichen Anstiegen
  - Viele Pixel werden mehrfach gesetzt
  - Aussehen der Linie stark abhängig vom Aussehen des Pixelmusters



Ränder zeichnen und Füllen

- Anstelle der Rasterung eines Primitivs werden zwei gerastert, mit Abstand $\pm t/2$ vom idealen Primitiv.
- Der Raum zwischen den beiden Primitiven wird dann gefüllt.
- Eigenschaften bzw. auftretende Probleme:
  - funktioniert auch mit geradzahligen Linienbreiten
  - Breite nicht abhängig vom Anstieg
  - Probleme beim Finden der „versetzten“ Linien
    - Geraden sind unproblematisch
    - Kreise sind unproblematisch
    - Ellipsen sind problematisch – Kurven, die $\pm t/2$ von einer Ellipse entfernt gezeichnet werden, sind keine Ellipsen
    - allgemeine Kurven funktionieren nicht



LFE Medieninformatik • Prof. Dr. Ing. Axel Hoppe

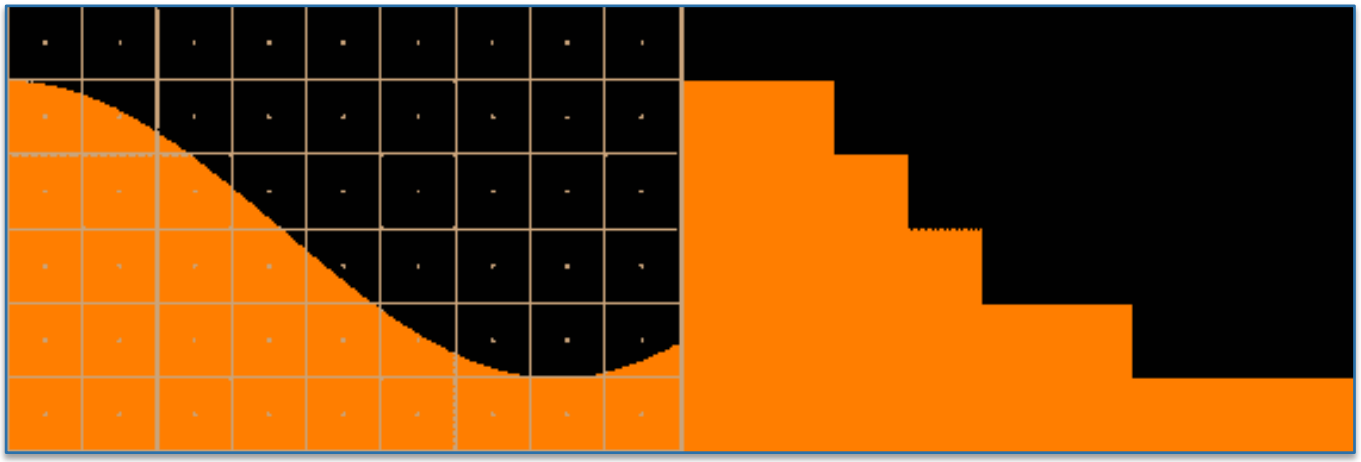
Rasteralgorithmen

Kantenglättung – Antialiasing

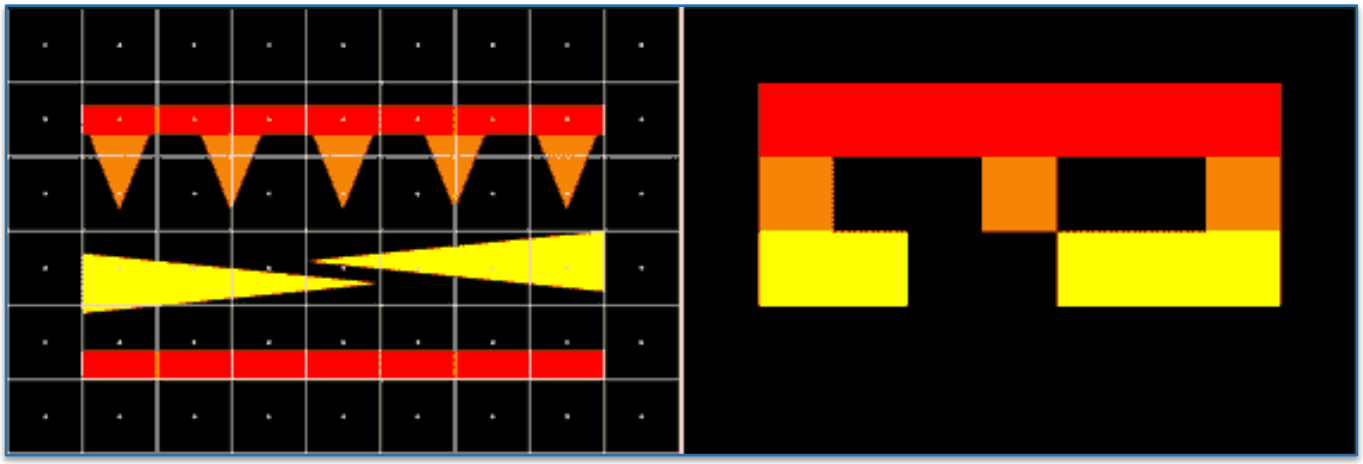
Antialiasing

- Neues Problem: Treppenstufen-Effekte beim Zeichnen von Linien, Details gehen verloren, etc.
- Aliasing
- Ursachen:
  - Diskretisierung eines kontinuierlichen Signals
  - unterschiedliche Abtastfrequenzen (Abtasttheorem)
- Lösung: **Antialiasing**
  - verschiedene Verfahren je nach Anwendung
  - hier insbesondere: Antialiasing beim Rastern von Linien
- erhöht die Qualität von Graphiken
- in Hardware verfügbar

Aliasing-Beispiele

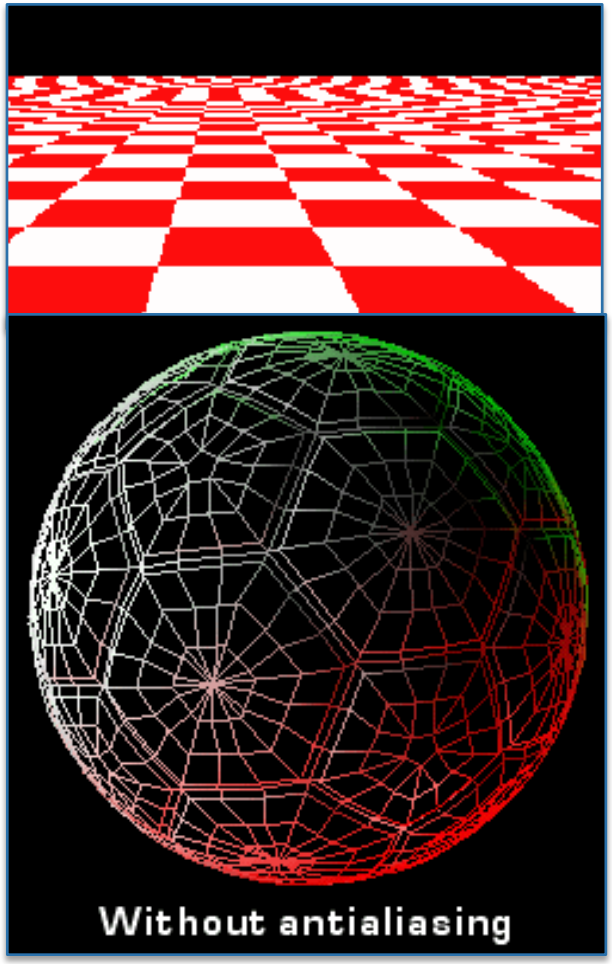


Treppenstufen

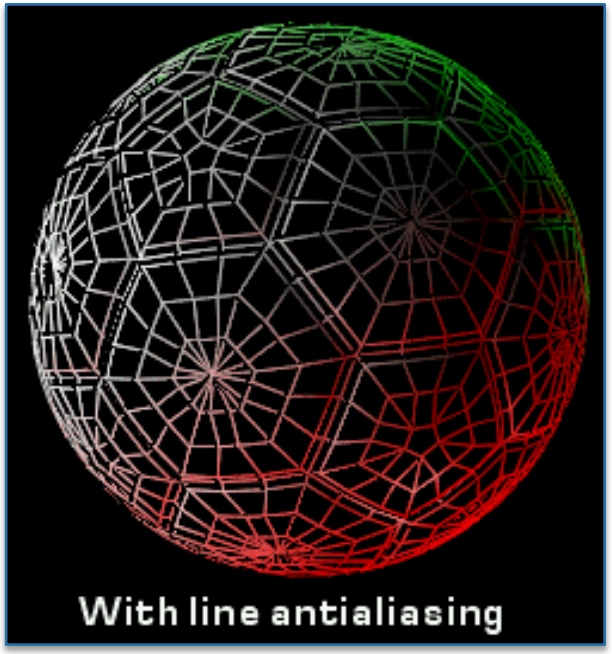


Binnen-Strukturen: Details gehen verloren

Aliasing-Beispiele



Moiré-Muster und Verwischen bei Texturen



Methoden

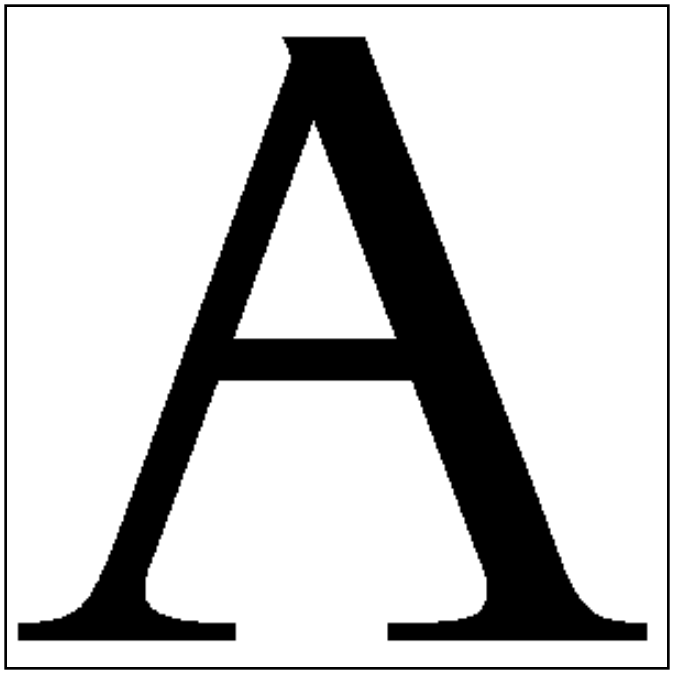
- Allgemeines Verfahren:
  - Erhöhen der Auflösung des Zielbildes und Zusammenfassen mehrerer Pixel zu einem
  - Supersampling
- Einbau in die Rasterungs-Algorithmen
  - Adaptive Anpassung der Pixelfarbe (Grauwert) an die Entfernung von der exakten Linie
  - Gewichtete Flächenbewertung
  - Ungewichtete Flächenbewertung

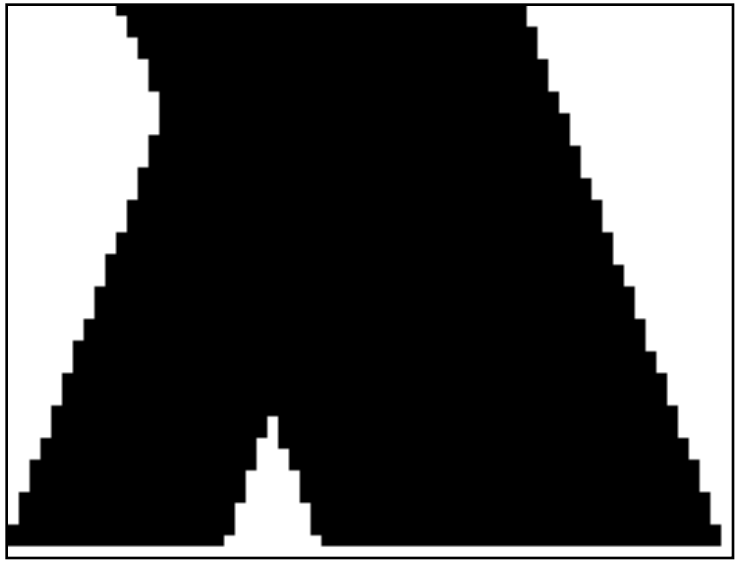
Supersampling

- einfachste Methode:
  - Bild wird in einer höheren Auflösung gesampled (berechnet)
  - danach: Downsampling auf die gewünschte Auflösung (gewichtetes Mittel der Sub-Pixel, die auf einen Pixel abgebildet werden)
- Praktikable Ansätze unterscheiden sich in:
  - Auswahl der „Super-Auflösung“
  - Lage der Subpixel zum Pixel an sich (Gitter)
  - Auswahl des Downsampling-Filters
    - einfaches Averaging
    - Box-Filter
    - Convolution

Einfaches Supersampling

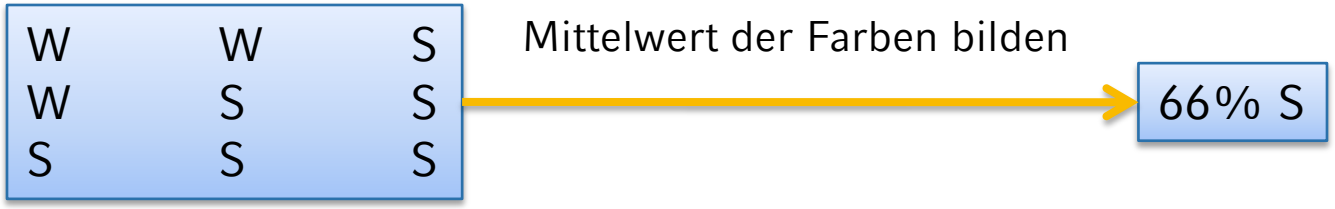
- wähle eine höhere Auflösung für ein „virtuelles Bild“
  - z. B. dreifache Auflösung
  - stelle das Bild mit dieser höheren Auflösung dar





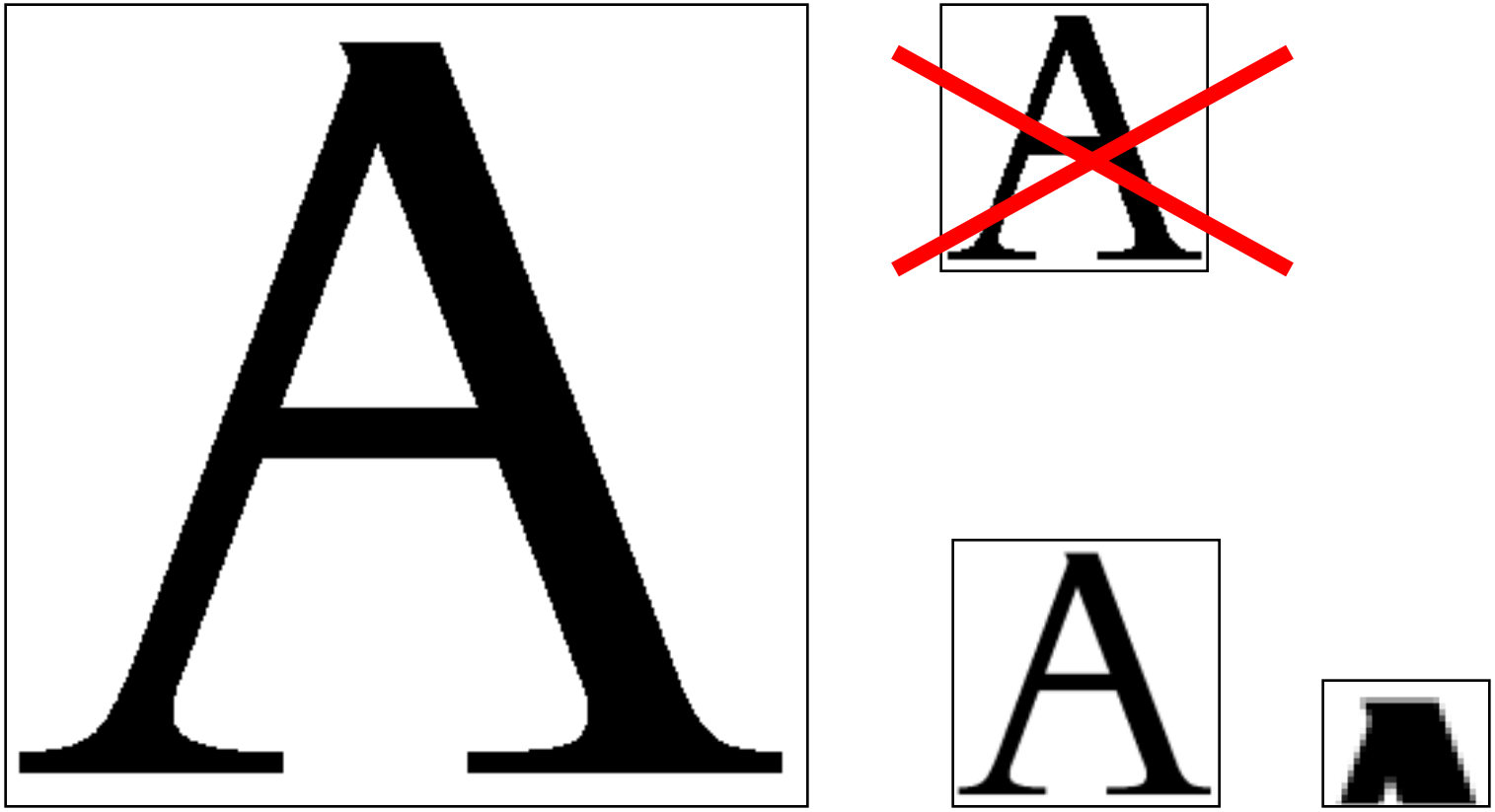
Einfaches Supersampling

- fasse jeweils *n* Pixel des „virtuellen Bildes“ zu einem Pixel des Ergebnisbildes zusammen (im Beispiel jeweils 3 × 3 Pixel)



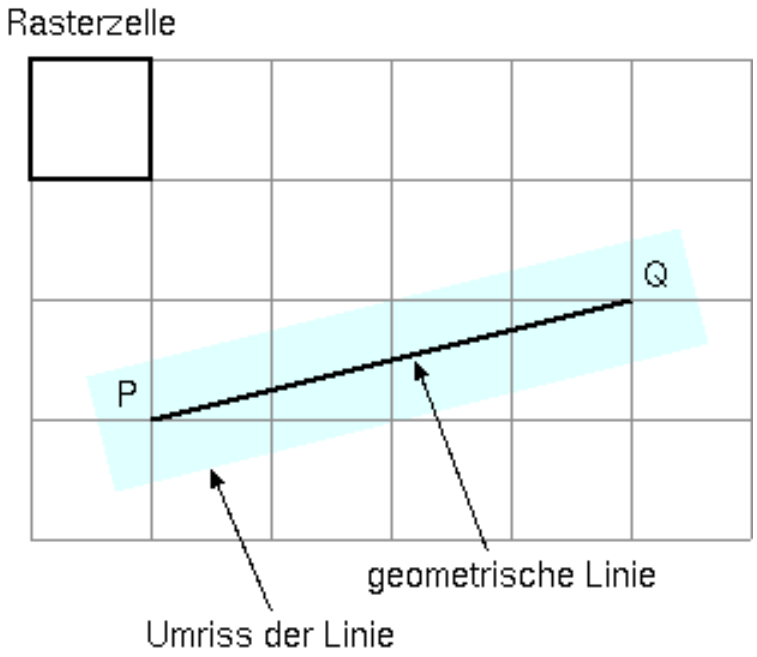
Einfaches Supersampling

- Ausgabe des Ergebnisbildes



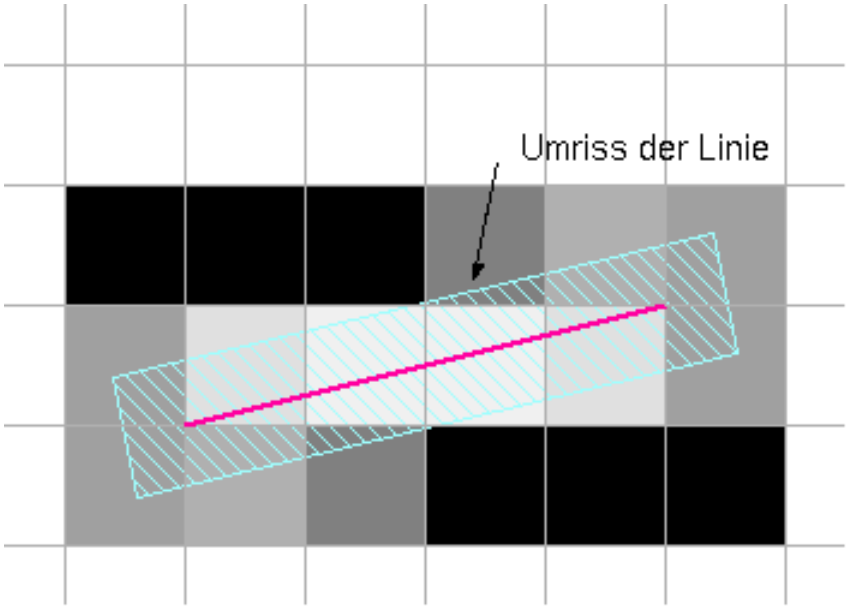
Flächenbewertungs-Algorithmen

- Anwendung direkt bei der Rasterkonvertierung von Primitiven (Linien, etc.)
- Geometrische Linie hat keine Breite, a*ber*: eine gerasterte Linie ist (wenigstens) 1 Pixel breit
- Geometrischer Umriss einer Linie = langes, schmales Rechteck
- Rechteckfläche überlappt Pixelgrundflächen teilweise, ungleichmäßig
- Bisherige Vereinfachung: Pixel gesetzt oder nicht zu grob → führt zu Treppenstufen-Effekten



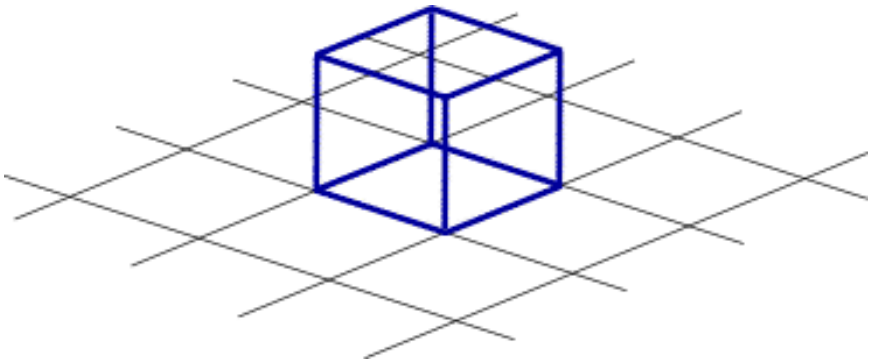
Ungewichtete Flächenbewertung

- Idee: Pixelhelligkeit proportional zum Anteil der Überdeckung dem Linienrechteck dosieren
- Annahme: Jedes Pixel hat quadratische Grundfläche der Größe 1 × 1 = Rasterzelle.

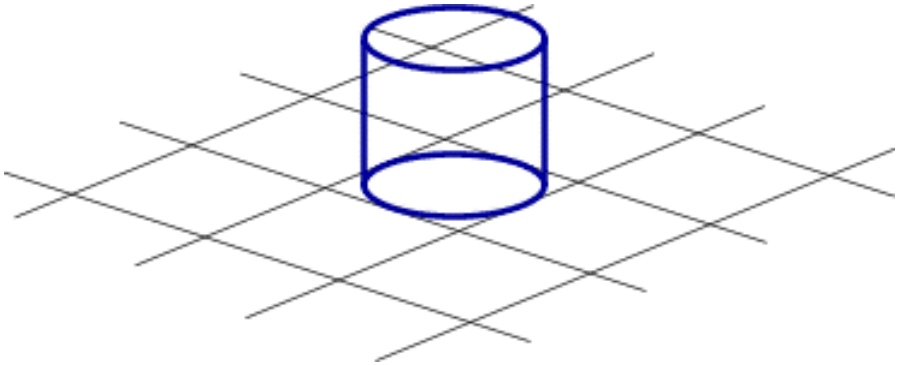


Grad der Überdeckung
Rasterzelle/Linienumriss
= Pixel-Helligkeit

Gewichtete Flächenbewertung

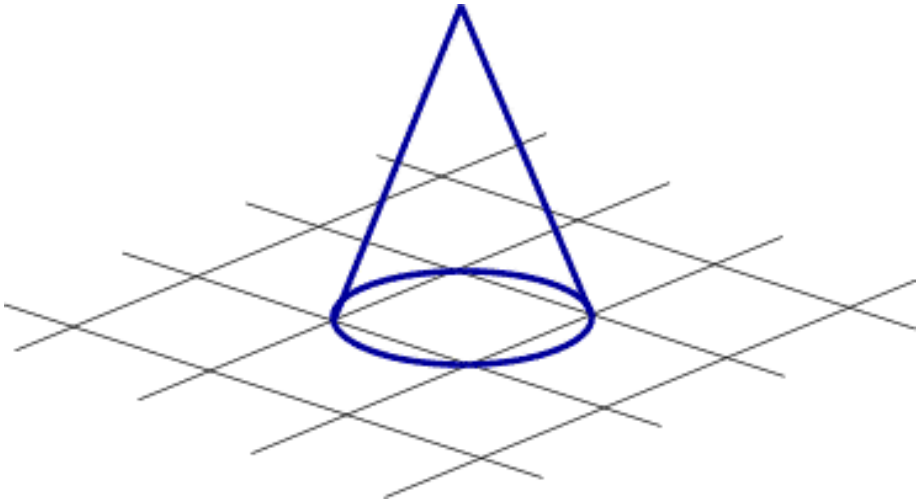


Helligkeitsbestimmung bei ungewichteter Flächenbewertung: Leuchtvolumen als Würfel, Anteil der überdeckten Linienfläche

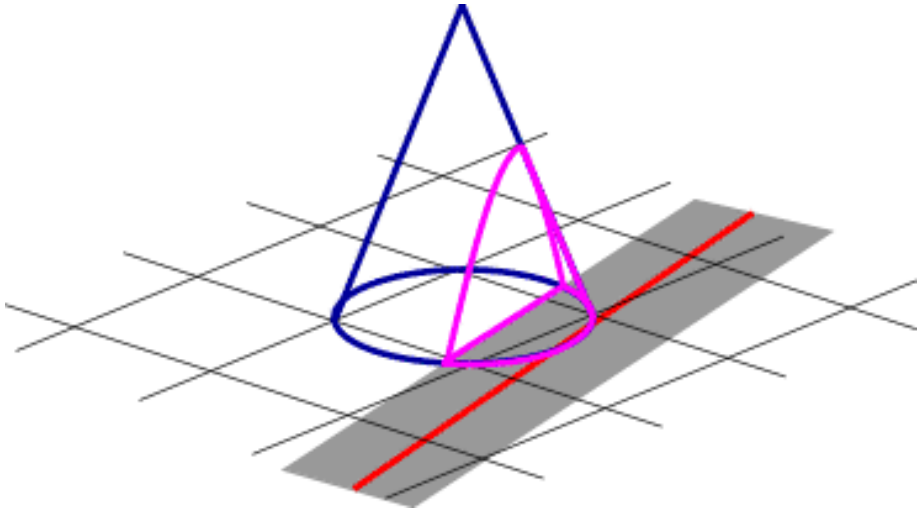


Realität: Pixelgrundflächen sind keine Quadrate sondern Kreise, daher Leuchtvolumen eher als Zylinder ansehen

Gewichtete Flächenbewertung



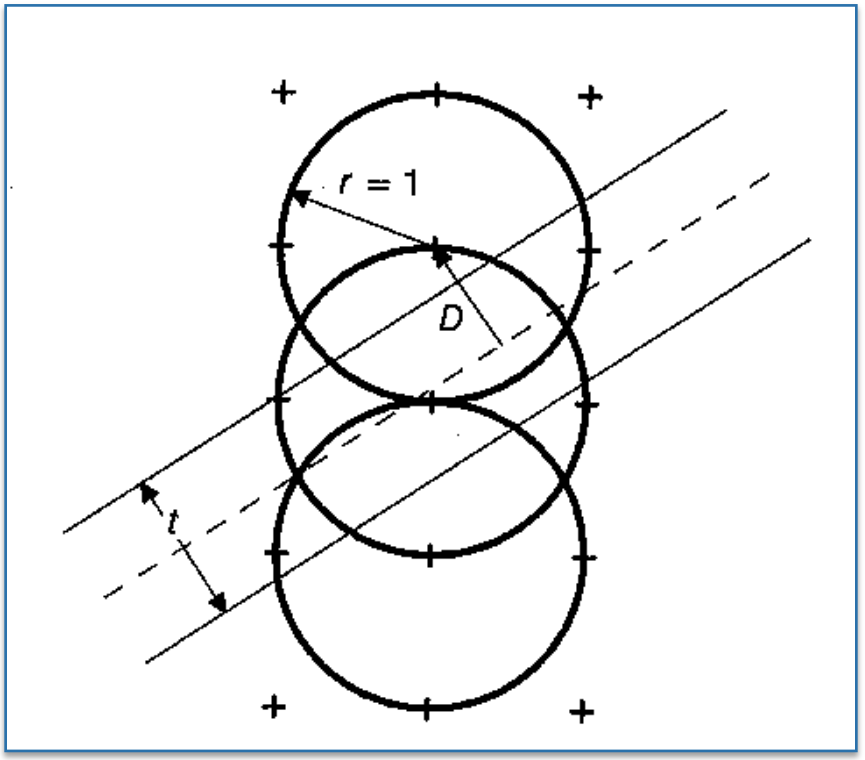
Pixelflächen auch nicht gleichmäßig hell ausgeleuchtet, heller im Zentrum, Licht fällt auch in Nachbarpixelzellen, daher Leuchtvolumen als Kegel mit größerer Grundfläche



Gewichtete Flächenbewertung: Anteil der
Pixelhelligkeit entspricht abgeschnittenem Volumenanteil des Kegels

Gupta-Sproull-Linien

- Verbindung von gewichteter Flächenbewertung mit dem Bresenham-Algorithmus für Linien
- Geometrisches Modell:
  - Mittelpunkte der Pixel auf Gitterpunkten mit Abstand 1
  - Pixel besitzen Einfluss-Bereich, der einer Kreis-scheibe mit Radius 1 entspricht
  - Linie besitzt Dicke t
  - Abstand Pixel zu Gerade wird durch den Abstand Pixelmittelpunkt zur Mittellinie des Primitivs bestimmt



Gupta-Sproull-Linien

- Ablauf des Verfahrens
  1. Berechne Rasterpunkt (Bresenham)
  2. Berechne Distanz D zwischen Rasterpunkt und Linie
  3. Gehe einen Schritt entlang der y-Achse aufwärts und berechne Du
  4. Gehe einen Schritt entlang der y-Achse abwärts und berechne Dl
  5. mit größerem D, Dl, Du ergibt sich kleinerer Grauwert
  6. Jede Linie ist somit 3 „Pixel dick“
- Grauwerte können vorberechnet werden
  1. max. Abstand ist 1.5, unterteilen je nach Anzahl der Graustufen (16, 256, etc.)
  2. Abspeichern in Tabelle und Lookup über Abstand

Zusammenfassung Antialiasing

Vorteile	Nachteile
• Eckige Kanten und harte Übergänge werden reduziert • Pixel- und Linien-Flimmern wird reduziert	• Verringerte Schärfe • Kleine Schriften werden „beschädigt“ • Größere Datenmengen • Längere Rechenzeiten

Quellen

- Foley, van Dam, Feiner, Hughes: *Computer Graphics, Principles and Practice*. Zweite Auflage, Addison Wesley. ISBN 0-201-84840-6.
- Bernhard Preim: *Computergraphik 1*. Universität Magdeburg, Vorlesungsskript, Juli 2005.