

Computergraphik

Computergraphik

Vorlesung im Wintersemester 2013/14

Übung zu Kapitel 1: Rasterisierung

Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie



- ▶ Noch nicht angemeldet?
 - ▶ <http://submit.ibds.kit.edu>
- ▶ Anmeldung für Übungstermin **nicht** notwendig für Abgabe der Übungsblätter!
- ▶ Erstes Übungsblatt: Heute online!
 - ▶ Abgabe am 4.11. (!)
 - ▶ bis 11:00 (!)
- ▶ Für den Schein
 - ▶ Mindestens 60% aller Punkte
 - ▶ Höchstens zwei Abgaben mit 0 Punkten bewertet

Johannes Meng
meng@kit.edu

Florian Simon
florian.simon@kit.edu

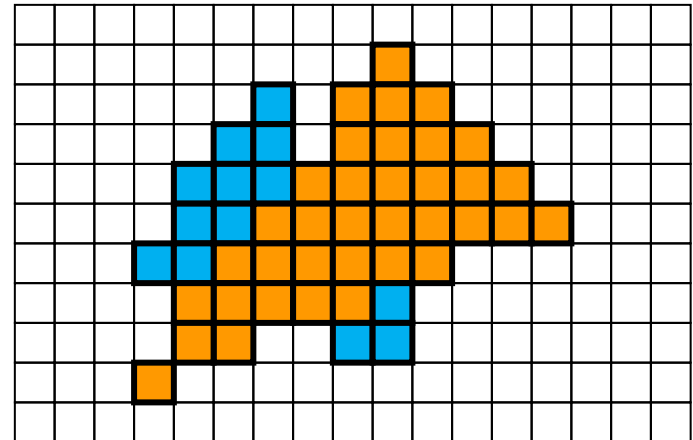
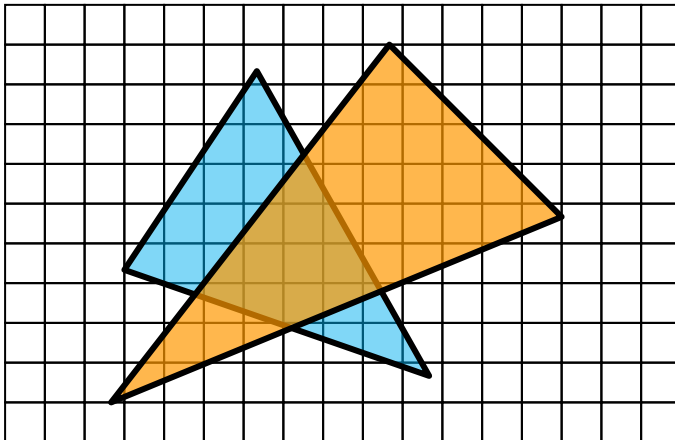
Hauke Rehfeld
hauke.rehfeld@kit.edu

Stephan Bergmann
Stephan.Bergmann@kit.edu

- ▶ Praxisaufgaben mit make (Linux) oder VisualStudio-Projekten (Windows) kompilieren
- ▶ Abgaben **müssen** in der Virtual Machine (Linux)
 - ▶ kompilieren
 - ▶ laufen
- ▶ Wir liefern voraussichtlich für jede Aufgabe Testcases
 - ▶ Richtige Lösungen bestehen alle Testcases
 - ▶ Es gibt aber Teilpunkte

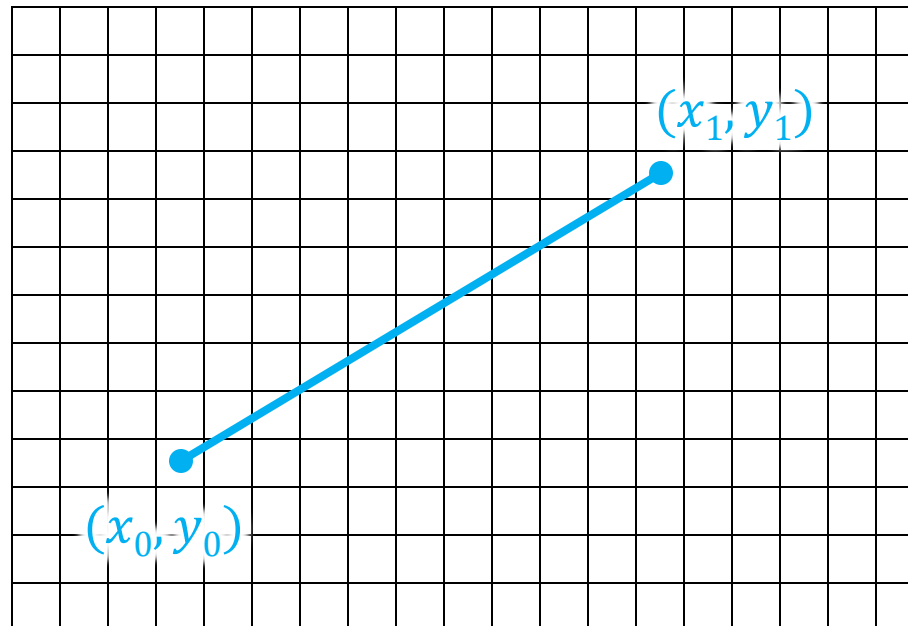
Inhalt

- ▶ Rasterisieren von Linien
- ▶ Polygon-Scanline-Rasterisierung



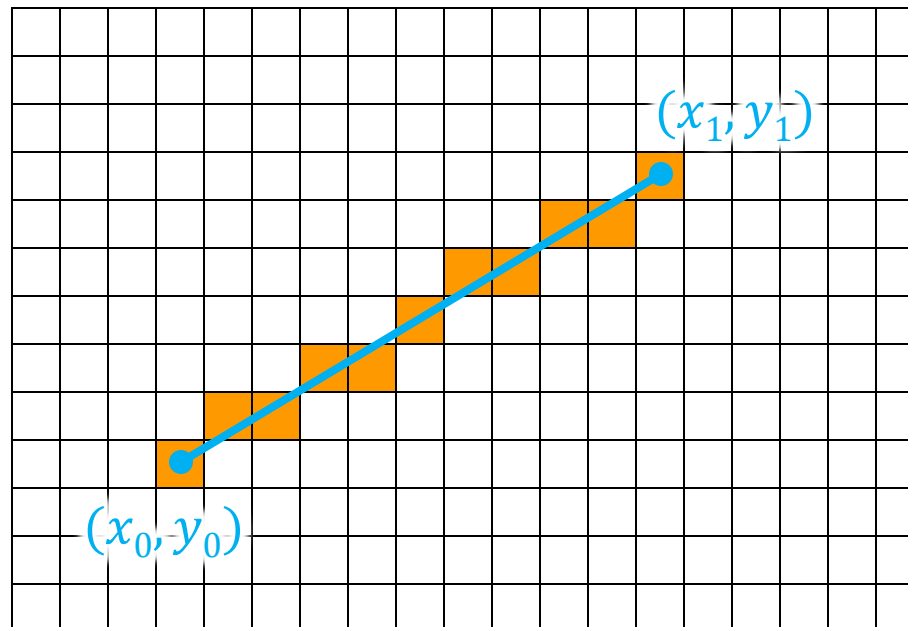
Rasterisierung von Linien

- ▶ geg.: Endpunkte des Liniensegments als Integertupel (x_0, y_0) , (x_1, y_1)
- ▶ ges.: Menge der Pixel, die gesetzt werden sollen
- ▶ optional: wenn die Linie eine bestimmte Dicke hat – welche Pixel muss man mit welcher Intensität setzen



Rasterisierung von Linien

- ▶ Rasterisierung (rasterizing, scan converting) von 2D Liniensegmenten
 - ▶ transformiere kontinuierliches Primitiv in diskrete Samples/Pixel
 - ▶ uniforme Strichstärke und Helligkeit
 - ▶ zusammenhängende Pixel, keine Lücken
 - ▶ wie macht man das akkurat und schnell?



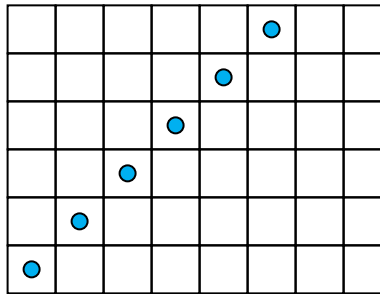
Rasterisierung von Linien

▶ zeichne Linie von $P_0 = (x_0, y_0)$ nach $P_1 = (x_1, y_1)$

▶ Steigung $m = \frac{\Delta y}{\Delta x} = \frac{y_1 - y_0}{x_1 - x_0}$

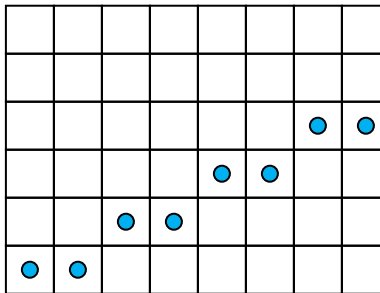
▶ Beispiele

▶ von $(0,0)$ nach $(6,6)$



Steigung $m = 6 / 6$

▶ von $(0,0)$ nach $(8,4)$



Steigung $m = 4 / 8$

▶ Linien-/Geradengleichung: $y = mx + \beta$

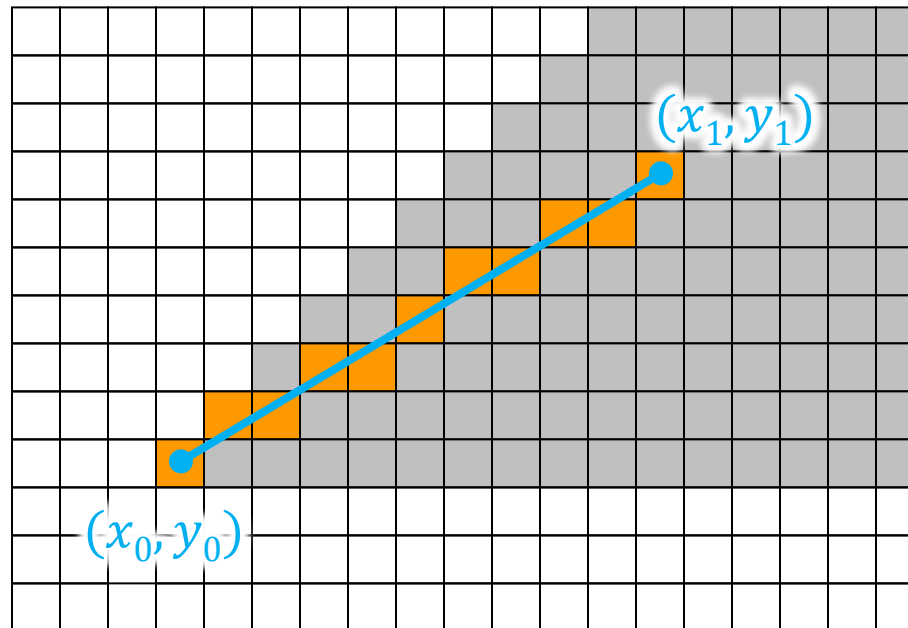
▶ mit $m = (y_1 - y_0) / (x_1 - x_0)$ und $\beta = y_0 - mx_0 = \frac{y_0x_1 - y_1x_0}{x_1 - x_0}$

▶ $\Rightarrow y = mx + \beta = \frac{y_1 - y_0}{x_1 - x_0}x + \frac{y_0x_1 - y_1x_0}{x_1 - x_0}$

▶ Vereinfachung: betrachte nur den Fall $0 \leq m \leq 1$

▶ alle anderen Fälle sind durch Ausnutzung von Symmetrie abgedeckt

▶ betrachte $x_0 \leq x \leq x_1$ mit $y = y_0 + m(x - x_0) = mx + \beta$



Rasterisierung von Linien

► Brute Force Algorithmus

- 3 Gleitkommaoperation pro Pixel: multiply, add, round

```
float m = (y1 - y0) / (x1 - x0);  
float beta = ( y0 * x1 - y1 * x0 ) / ( x1 - x0 );  
  
for ( int x = x0; x <= x1; x++ ) {  
    float y = m * x + beta;  
    set_pixel( x, round( y ) );  
}
```

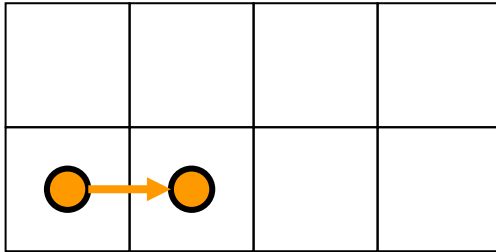
► erste Verbesserung: inkrementelle Berechnung

- $y_n = mx_n + \beta$ und $y_{n+1} = m(x_n + 1) + \beta = y_n + m$
- 2 Gleitkommaoperationen pro Pixel: add, round

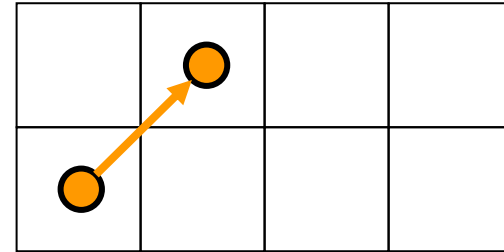
```
float m = (y1 - y0) / (x1 - x0);  
float beta = ( y0 * x1 - y1 * x0 ) / ( x1 - x0 );  
float y = m * x0 + beta;  
  
for ( int x = x0; x <= x1; x++ ) {  
    set_pixel( x, round( y ) )  
    y += m;  
}
```

Rasterisierung von Linien

- ▶ nach wie vor gilt die Annahme: $0 \leq m \leq 1$
- ▶ nur 2 mögliche Fälle, wenn man von einem Pixel x_n zu x_{n+1} geht:



East



North East

- ▶ Ziel des **Bresenham Algorithmus**:
finde Bedingung **cond**, um zu entscheiden welcher Fall eintreten soll

```
int y = y0;  
  
for ( int x = x0; x <= x1; x++ ) {  
    set_pixel( x, y );  
    if ( cond ) y ++;  
}
```

Rasterisierung von Linien

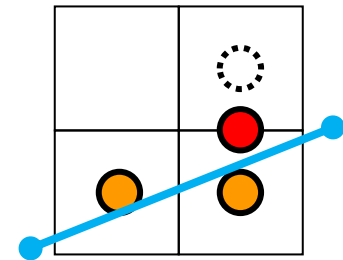
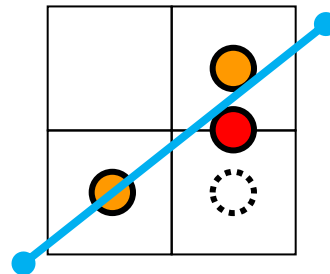
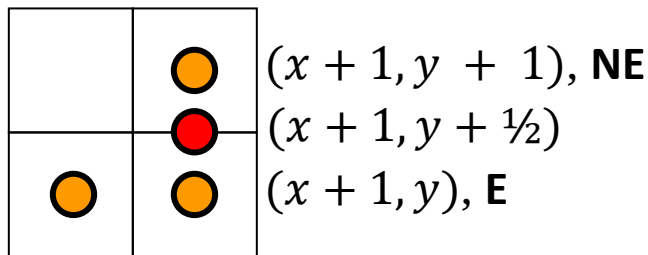
- ▶ implizite Darstellung einer Linie

$$F(x, y) = (y - y_0) - m(x - x_0)$$

$$F(x, y) > 0$$

$$F(x, y) < 0$$

- ▶ werte $F(x, y)$ für die Mitte zw. dem E und NE Pixel aus: $F(x + 1, y + \frac{1}{2})$
 - ▶ verläuft die Linie oberhalb von $(x + 1, y + \frac{1}{2})$: wähle $(x + 1, y + 1)$ (NorthEast)
 - ▶ sonst: wähle $(x + 1, y)$ (East)

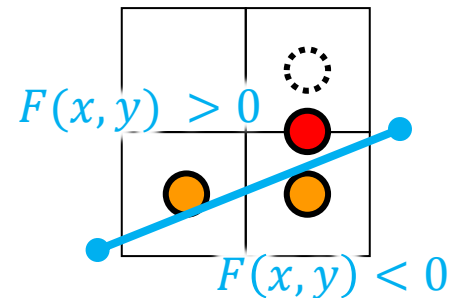
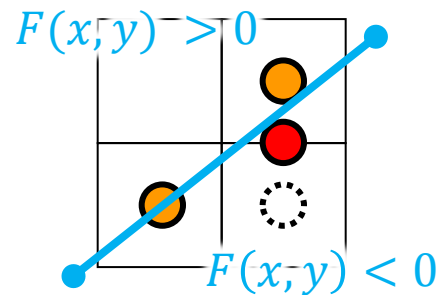
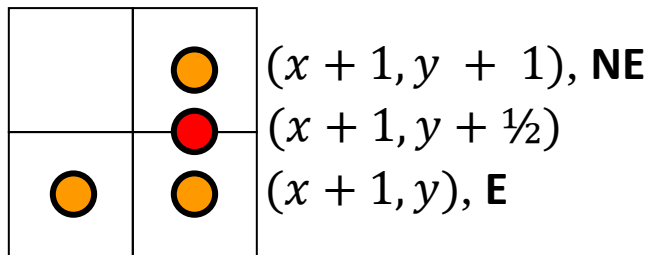


Rasterisierung von Linien

► werte $F(x, y)$ für die Mitte zw. dem E und NE Pixel aus: $F(x + 1, y + \frac{1}{2})$

► Linie oberhalb von $(x + 1, y + \frac{1}{2}) \leftrightarrow F(x + 1, y + \frac{1}{2}) < 0$

► Linie unterhalb von $(x + 1, y + \frac{1}{2}) \leftrightarrow F(x + 1, y + \frac{1}{2}) > 0$



```
int y = y0;
```

```
for ( int x = x0; x <= x1; x++ ) {  
    set_pixel( x, y );  
    if ( F(x+1, y+0.5) < 0 ) y ++;  
}
```

Bresenham Algorithmus

- ▶ Bresenham's Alg. basiert auf der implizierten Darstellung und arbeitet
 - ▶ inkrementell für $F(x, y) = y(x_1 - x_0) + x(y_0 - y_1) + y_1x_0 - y_0x_1$
 - ▶ nur mit Integer-Operationen

Schritt 1: inkrementelle Auswertung

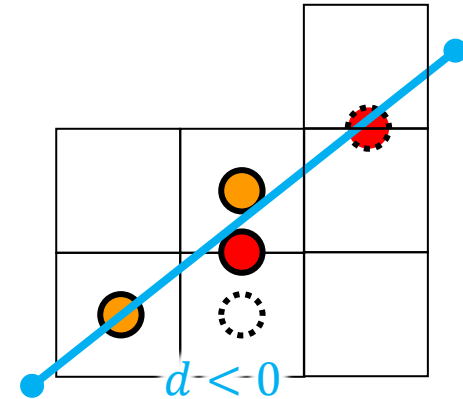
- ▶ Initialisierung: $d := F\left(x_0 + 1, y_0 + \frac{1}{2}\right)$
- ▶ betrachte zunächst nur den ersten Schleifendurchlauf
- ▶ wenn $d < 0$ dann NE: $(x_0, y_0) \rightarrow (x_0 + 1, y_0 + 1)$
 - ▶ der nächste Test wäre dann bei $\left(x_0 + 2, y_0 + 1 + \frac{1}{2}\right)$

$$\begin{aligned} F\left(x_0 + 2, y_0 + \frac{3}{2}\right) &= \\ &= \left(y_0 + \frac{3}{2}\right)(x_1 - x_0) + (x_0 + 2)(y_0 - y_1) + y_1x_0 - y_0x_1 \end{aligned}$$

- ▶ Veränderung von d :

$$F\left(x_0 + 2, y_0 + \frac{3}{2}\right) - F\left(x_0 + 1, y_0 + \frac{1}{2}\right) = (y_0 - y_1) + (x_1 - x_0)$$

- ▶ $\Rightarrow d \leftarrow d + (y_0 - y_1) + (x_1 - x_0)$



Bresenham Algorithmus

Schritt 1: inkrementelle Auswertung

▶ wenn $d < 0$ dann NE: $(x_0, y_0) \rightarrow (x_0 + 1, y_0 + 1)$

▶ ...

▶ wenn $d \geq 0$ dann E: $(x_0, y_0) \rightarrow (x_0 + 1, y_0)$

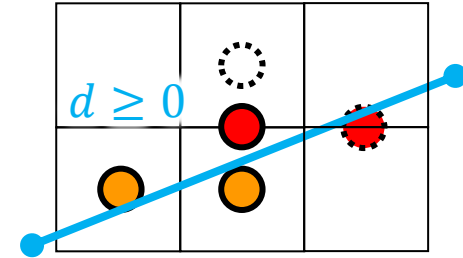
▶ der nächste Test wäre dann bei $(x_0 + 2, y_0 + \frac{1}{2})$

$$\begin{aligned} F\left(x_0 + 2, y_0 + \frac{1}{2}\right) &= \\ &= \left(y_0 + \frac{1}{2}\right)(x_1 - x_0) + (x_0 + 2)(y_0 - y_1) + x_0 y_1 - x_1 y_0 \end{aligned}$$

▶ Veränderung von $F(x, y)$:

$$F\left(x_0 + 2, y_0 + \frac{1}{2}\right) - F\left(x_0 + 1, y_0 + \frac{1}{2}\right) = (y_0 - y_1)$$

▶ $\Rightarrow d \leftarrow d + (y_0 - y_1)$



Bresenham Algorithmus



Schritt 1: inkrementelle Auswertung

```
int y = y0;
float d = F( x0 + 1, y0 + 0.5);

for ( int x = x0; x <= x1; x++ ) {
    set_pixel( x, y );
    if ( d < 0 ) {
        // gehe nach NE
        y = y + 1;
        d = d + (x1 - x0) + (y0 - y1);
    } else {
        // gehe nach E
        d = d + (y0 - y1);
    }
}
```

Integer-Variante des Algorithmus

▶ verwende $d = 2F(x, y)$

▶ möglich, weil nur das Vorzeichen für die Entscheidung relevant ist

$$F(x_0 + 1, y_0 + 1/2) = (y_0 + 1/2)(x_1 - x_0) + (x_0 + 1)(y_0 - y_1) + y_1x_0 - y_0x_1$$

$$2F(x_0 + 1, y_0 + 1/2) = (2y_0 + 1)(x_1 - x_0) + 2(x_0 + 1)(y_0 - y_1) + 2y_1x_0 - 2y_0x_1$$

▶ die inkrementellen Updates sind dann

$$d \leftarrow d + 2(y_0 - y_1) \quad (\text{E})$$

▶ bzw.

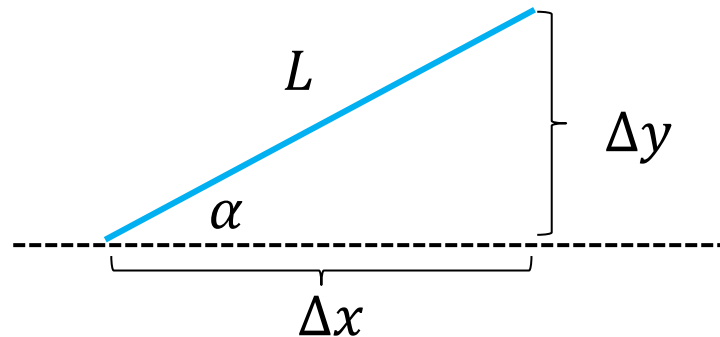
$$d \leftarrow d + 2(y_0 - y_1) + 2(x_1 - x_0) \quad (\text{NE})$$

Bemerkungen

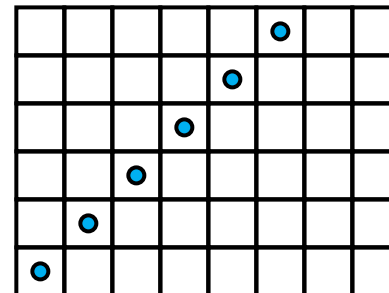
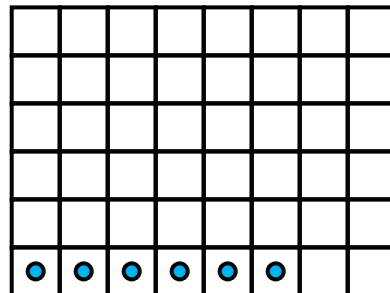
- ▶ Test $0 < m < 1$
 - ▶ 1. Test: $(y_1 - y_0) > 0$
 - ▶ 2. Test: $(x_1 - x_0) > (y_1 - y_0)$
- ▶ für den Fall $m > 1$
 - ▶ vertausche x und y : $y = mx + \beta \Leftrightarrow x = \frac{1}{m}y + \beta'$
- ▶ zwei weitere Fälle: $m < -1$ und $-1 < m < 0$
 - ▶ einfache Erweiterung der bisherigen Fälle
- ▶ in der Praxis:
 - ▶ nur Fallunterscheidungen bei der Initialisierung
 - ▶ Schleife über $\max(|x_1 - x_0|, |y_1 - y_0|)$ Pixel
 - ▶ alle weiteren Berechnungen inkrementell

► Probleme mit den bisherigen Ansätzen

- idealerweise würde für eine Linie der Länge L (gemessen in Pixel) auch L Pixel gesetzt werden, andernfalls sieht die Linie fragmentiert aus
- Bresenham Algorithmus setzt $\max(\Delta x, \Delta y) = L \cos \alpha$ Pixel

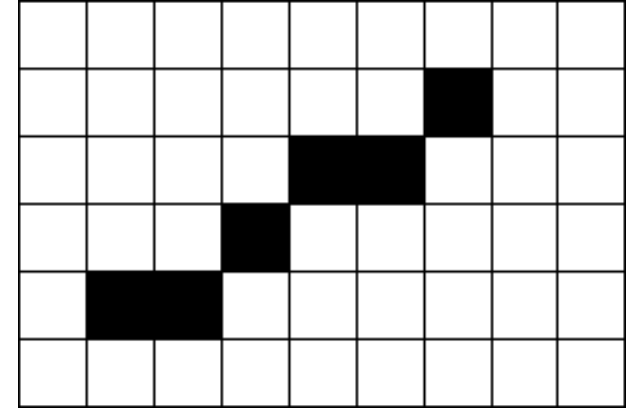


- einfach zu realisieren: erhalte die Gesamthelligkeit durch Erhöhen der Helligkeit um den Faktor $\frac{1}{\cos \alpha}$ ($0 \leq m \leq 1$)

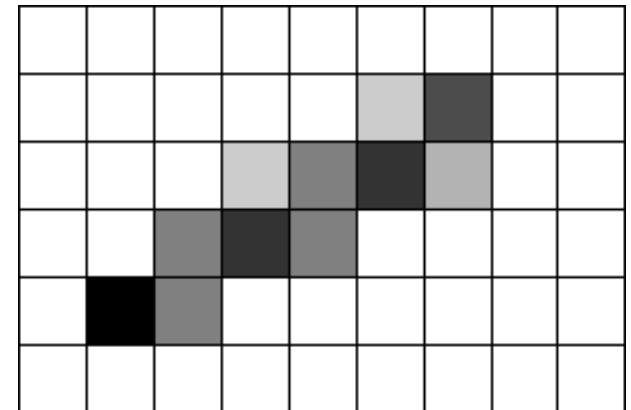


Rasterisierung von Linien mit Antialiasing

- ▶ der original Bresenham Algorithmus setzt einen Pixel pro Inkrement
 - ▶ die Intensität wird nur einem Pixel zugewiesen

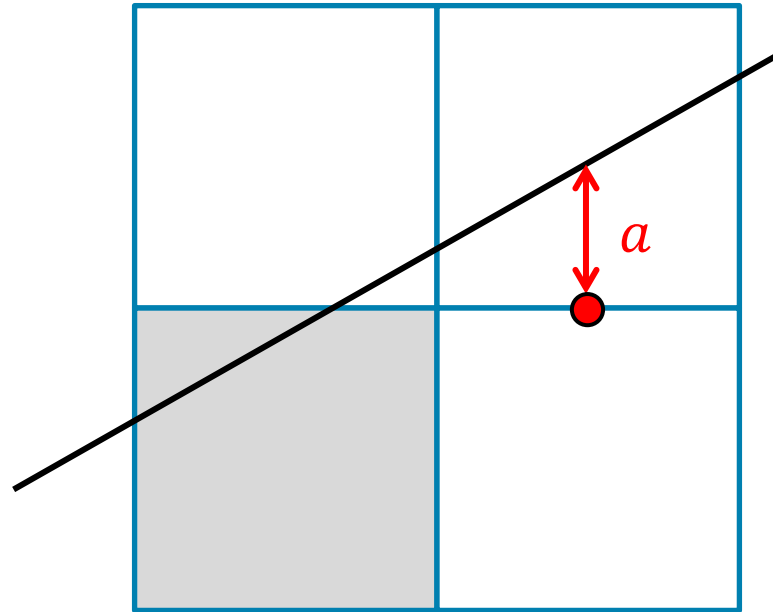


- ▶ modifizierter Bresenham
 - ▶ zwei Pixel werden gesetzt
 - ▶ Intensitäten summieren sich zur gewünschten Gesamtintensität auf



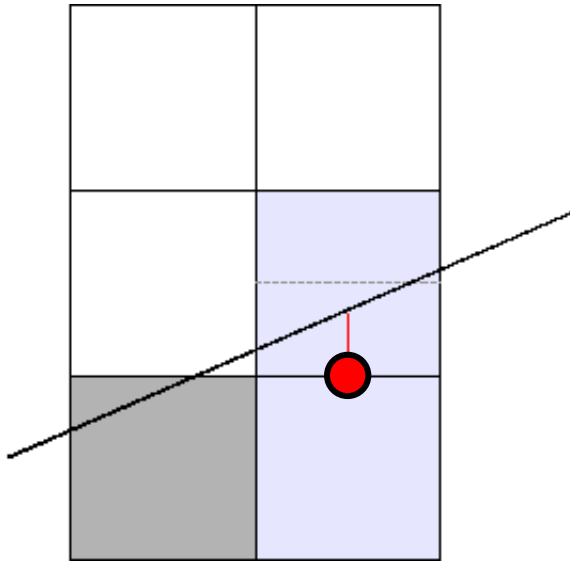
Rasterisierung von Linien mit Antialiasing

- ▶ für die Entscheidung welche Pixel mit welcher Intensität gesetzt werden benötigen wir den (vorzeichenbehafteten) Abstand a zw. der Linie und dem Mittelpunkt zw. dem E und NE Pixel
- ▶ Abstand kann aus Entscheidungsvariable berechnet werden: $a = d/2$



Rasterisierung von Linien mit Antialiasing

- ▶ welche Pixel sollen gesetzt werden?
- ▶ Fall 1: $d < 0$ (der North-East Pixel wird gewählt)

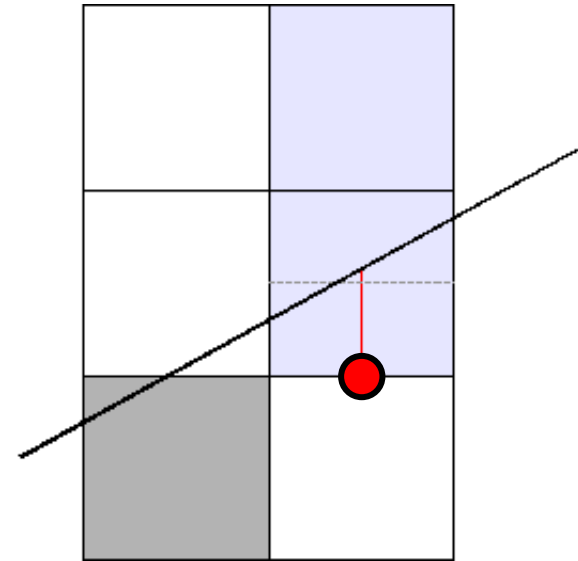


wenn $0 \geq a \geq -1/2$

setze Pixel:

$(x + 1, y)$ mit $\frac{1}{2} + a$

$(x + 1, y + 1)$ mit $\frac{1}{2} - a$



wenn $a < -1/2$

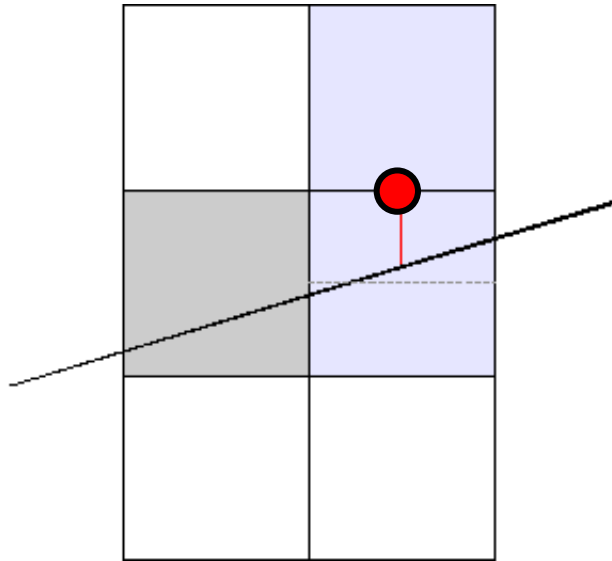
setze Pixel:

$(x + 1, y + 1)$ mit $\frac{3}{2} + a$

$(x + 1, y + 2)$ mit $-\frac{1}{2} - a$

Rasterisierung von Linien mit Antialiasing

- ▶ welche Pixel sollen gesetzt werden?
- ▶ Fall 2: $d \geq 0$ (der East Pixel wird gewählt)

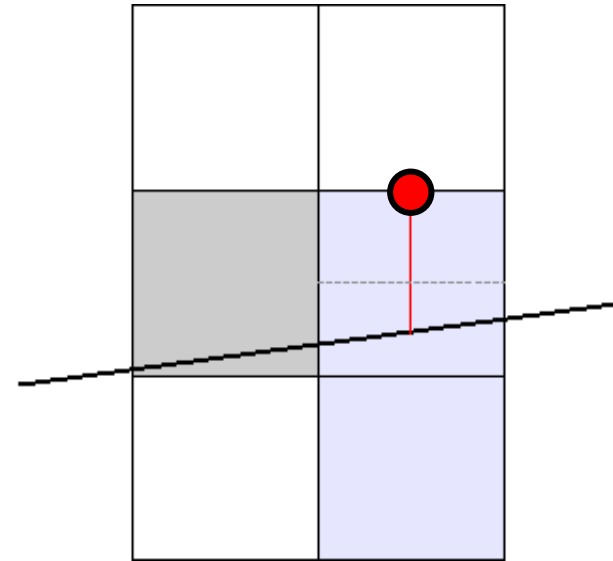


wenn $0 \leq a \leq \frac{1}{2}$

setze Pixel:

$(x + 1, y)$ mit $\frac{1}{2} + a$

$(x + 1, y + 1)$ mit $\frac{1}{2} - a$



wenn $1 \geq a > \frac{1}{2}$

setze Pixel:

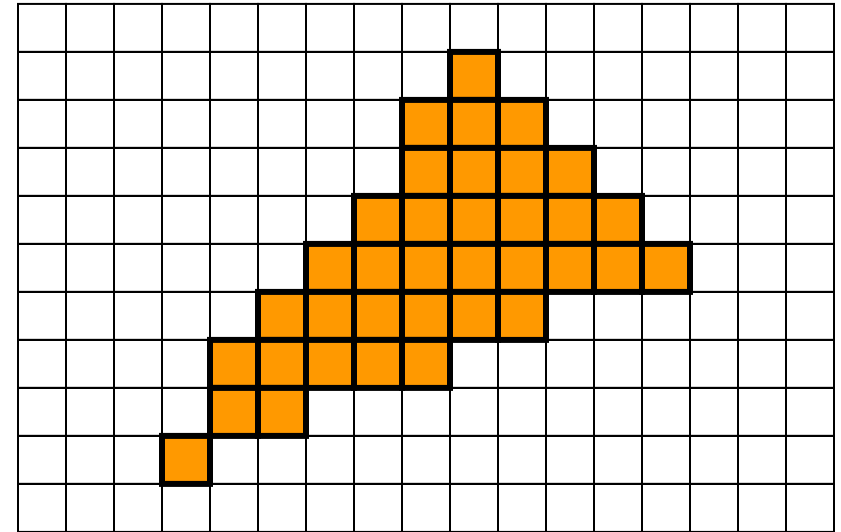
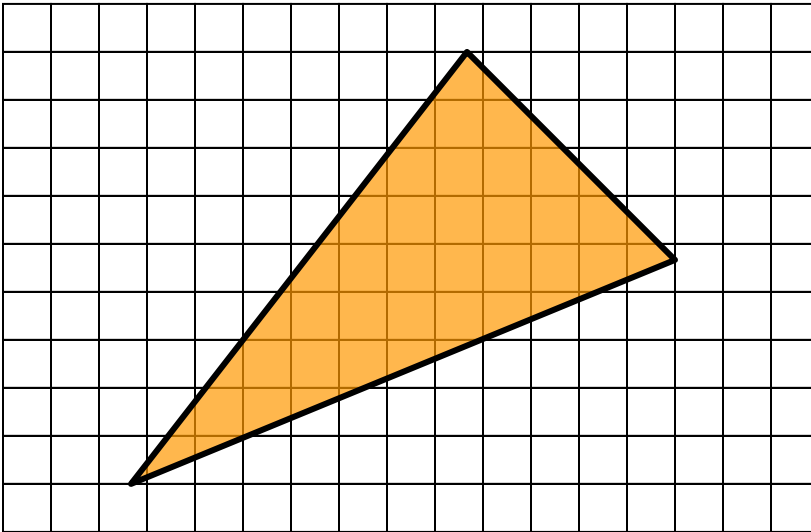
$(x + 1, y)$ mit $\frac{3}{2} - a$

$(x + 1, y - 1)$ mit $a - \frac{1}{2}$

Rasterisierung von Polygonen

▶ Polygon-Rasterisierung:

- ▶ geg. ein 2D-Polygon mit n Eckpunkten $P_1, \dots, P_n$
- ▶ färbe alle Pixel im Inneren des Polygons

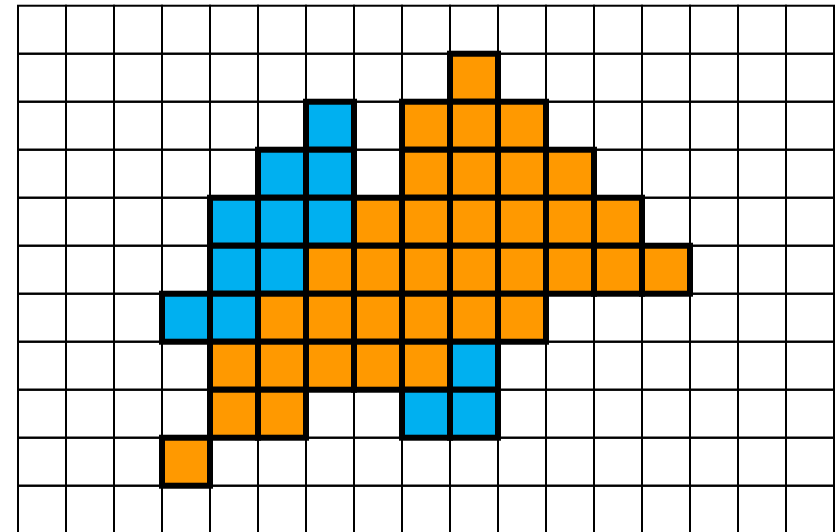
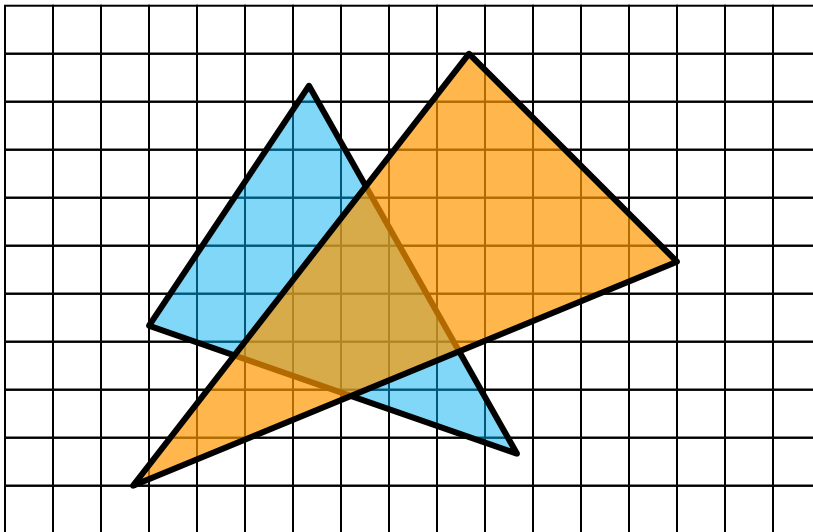


▶ Polygon-Rasterisierung:

- ▶ geg. ein 2D-Polygon mit n Eckpunkten $P_1, \dots, P_n$
- ▶ färbe alle Pixel im Inneren des Polygons

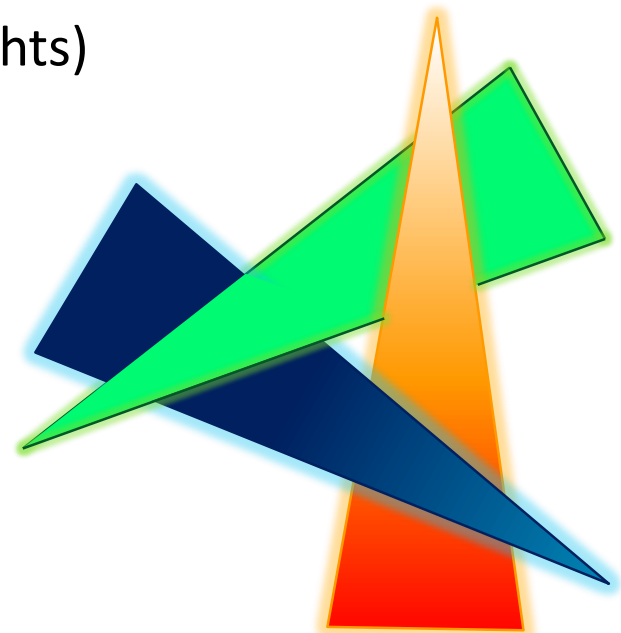
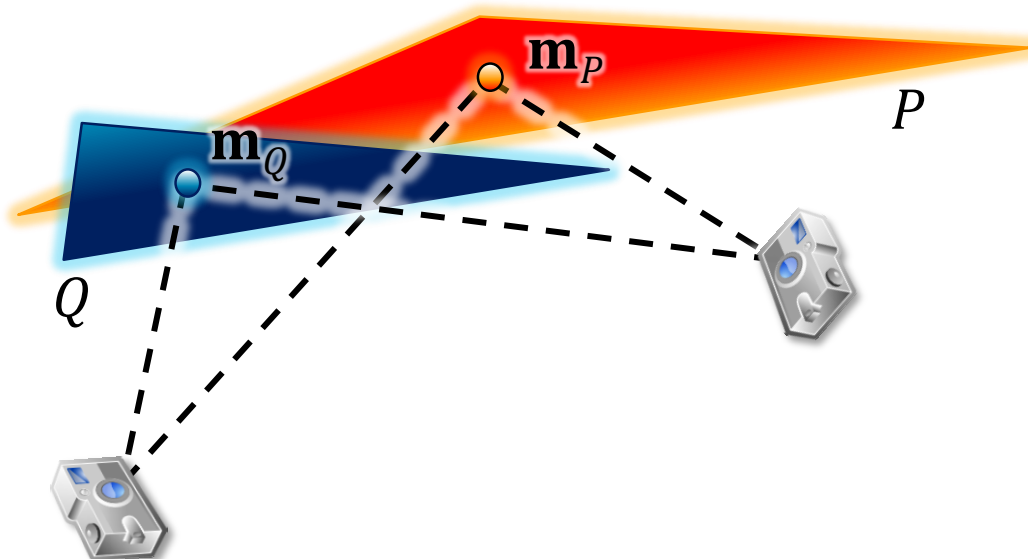
▶ wir möchten 3D Szenen rasterisieren

- ▶ wir müssen die 2D Polygone am Bildschirm bestimmen
- ▶ wir müssen das Verdeckungs- oder Sichtbarkeitsproblem lösen



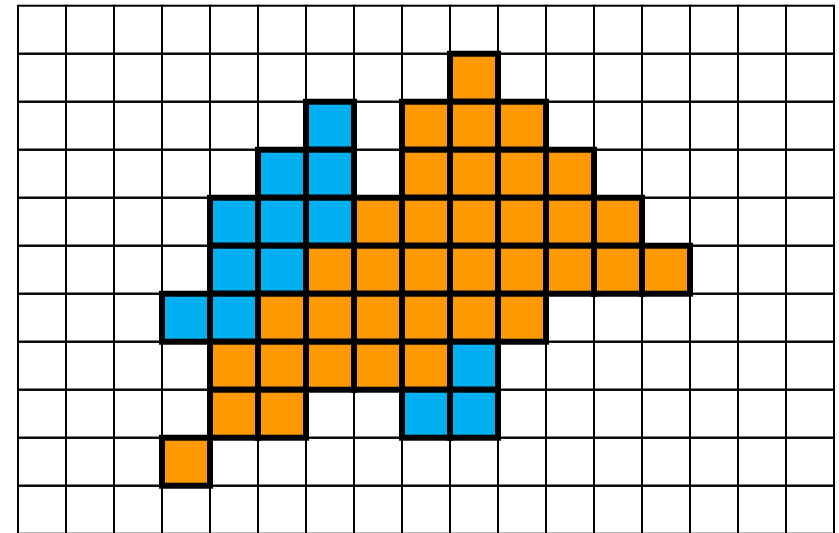
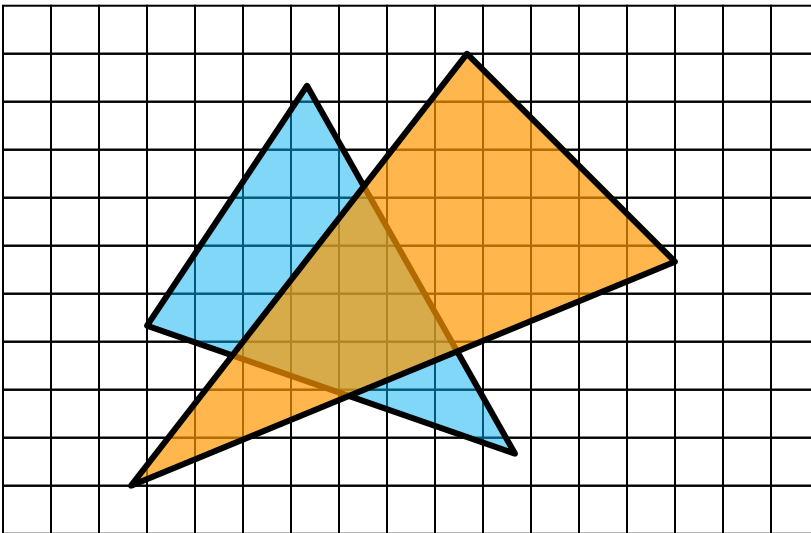
Maler-Algorithmus (Painter's Algorithm)

- ▶ nehmen wir an, wir könnten Polygone rasterisieren...
- ▶ einfachster Ansatz für das Sichtbarkeitsproblem: Maler-Algorithmus
sortiere die Polygone (Dreiecke) nach dem Abstand zur Kamera
(„von hinten nach vorne“) und zeichne sie in dieser Reihenfolge
- ▶ Probleme
 - ▶ nach welchem Abstand soll sortiert werden?
Abstand zu einem Punkt auf dem Polygon (hier: der Mittelpunkt)
liefert nicht immer das richtige Ergebnis (links)
 - ▶ Zyklen können nicht sortiert werden (rechts)



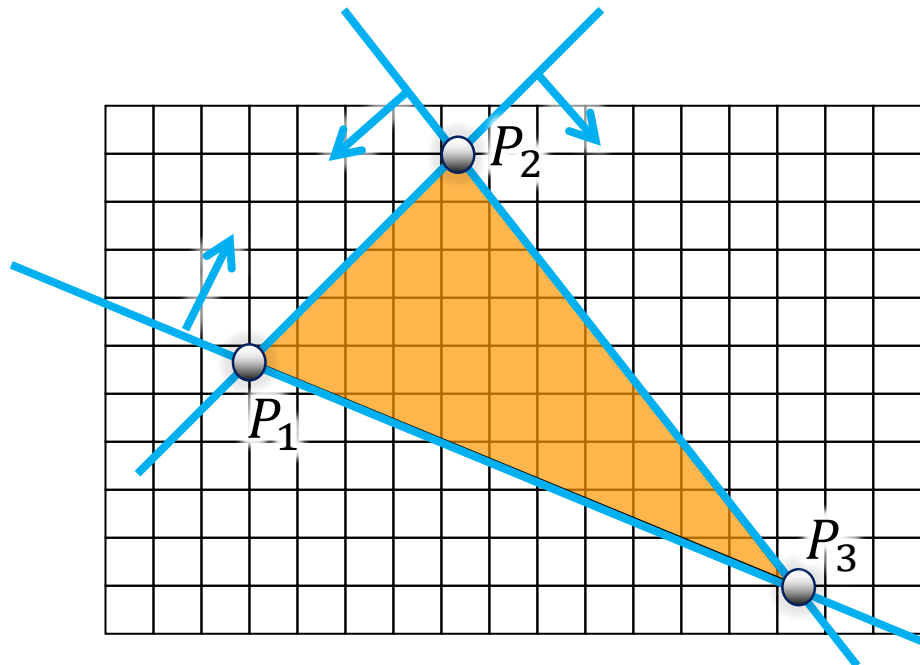
▶ Polygon-Rasterisierung:

- ▶ geg. ein 2D-Polygon mit n Eckpunkten $P_1, \dots, P_n$ (in Window-Koord.)
 - ▶ färbe alle Pixel im Inneren des Polygons
- ## ▶ verschiedene Möglichkeiten
- ▶ Seed-Fill (allgemeiner Flächenfüllalgorithmus, nicht praktikabel)
 - ▶ Brute Force Test: Pixel vs. Geradengleichungen der Kanten
 - ▶ Scanline Verfahren



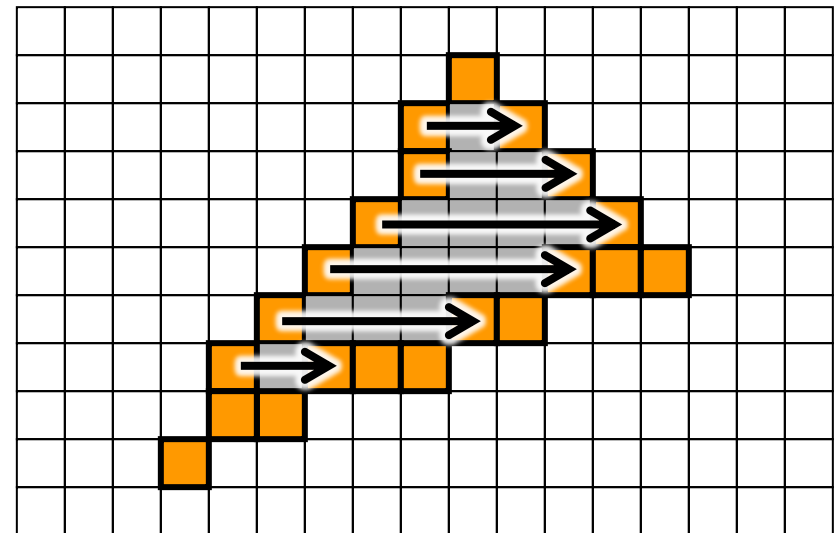
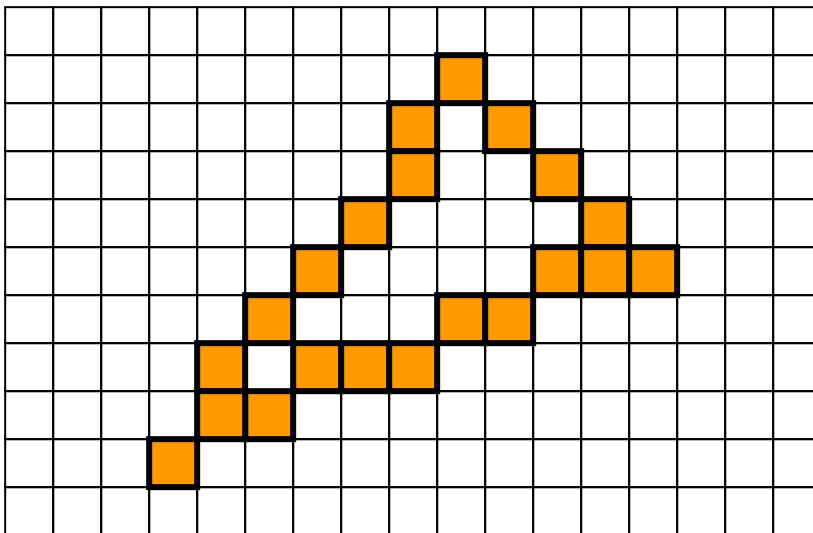
Brute Force Ansatz (nur für konvexe Polygone)

- ▶ stelle die Geradengleichungen der Kanten auf
- ▶ Orientierung so, dass der gerichtete Abstand aller Punkte im Inneren des Dreiecks zu den Kanten positiv ist → Test für jeden Pixel(Mittelpunkt)
- ▶ Konvention: lege fest, dass man ein Polygon „sieht“, wenn die Punkte im oder gegen den Uhrzeigersinn angeordnet sind



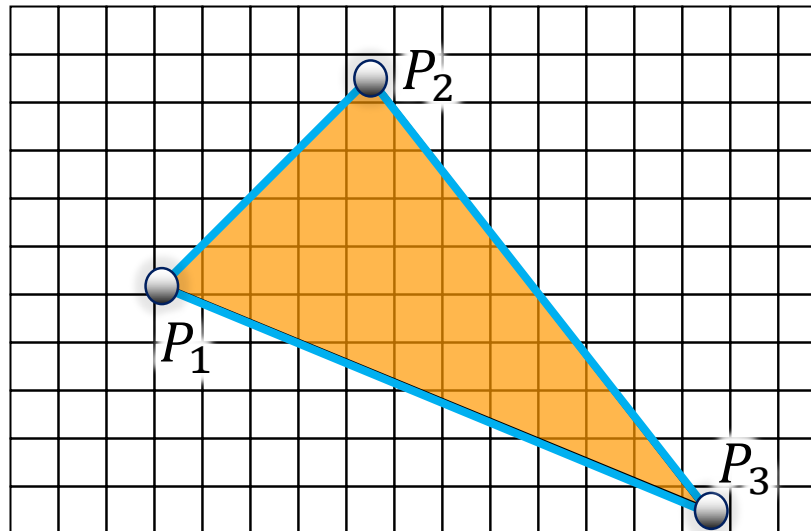
Scanline Polygon Rendering

- ▶ behandle eine Scanline (Pixel-Zeile) nach der anderen
 - ▶ von unten nach oben (oder umgekehrt, je nach Implementation)
- ▶ finde Schnitte der Scanline mit dem Polygon
- ▶ setze die Pixel in diesem Teil
- ▶ wir besprechen im Folgenden das Rasterisieren von konvexen Polygonen
 - ▶ eine Verallgemeinerung auf beliebige Polygone ist mittels sog. Active Edge Tables möglich (nicht Stoff dieser Vorlesung)



Rasterisierung von konvexen Polygonen

- ▶ geg. **umlaufend sortierte** Liste von Eckpunkten $P_1 = (x_1, y_1), P_2, \dots, P_n$
- ▶ finde den obersten und untersten Eckpunkt des Polygons (hier $n = 3$)
 - ▶ kleinste y -Koordinate y_t (hier: P_t mit $t = 2$)
 - ▶ größte y -Koordinate y_b (hier: P_b mit $b = 3$)
- ▶ das Polygon wird darunter zunächst von den an P_t anliegenden Kanten begrenzt: $\overline{P_{t-1}P_t}$ und $\overline{P_{t+1}P_t}$ (Konvention: $P_0 = P_n, P_{n+1} = P_1$)

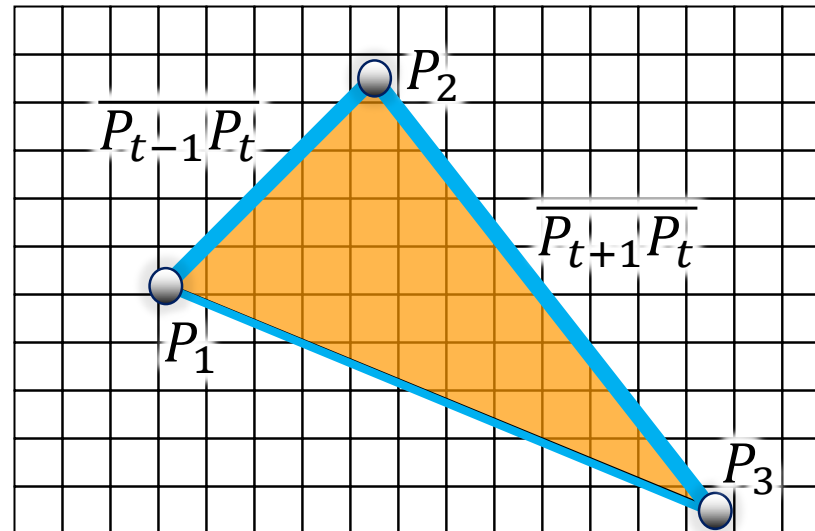


Rasterisierung von konvexen Polygonen

- ▶ das Polygon wird darunter zunächst von den an P_t anliegenden Kanten begrenzt: $\overline{P_{t-1}P_t}$ und $\overline{P_{t+1}P_t}$ (Konvention: $P_0 = P_n$, $P_{n+1} = P_1$)
- ▶ Änderung der x -Koordinate von einer zur nächsten Zeile:

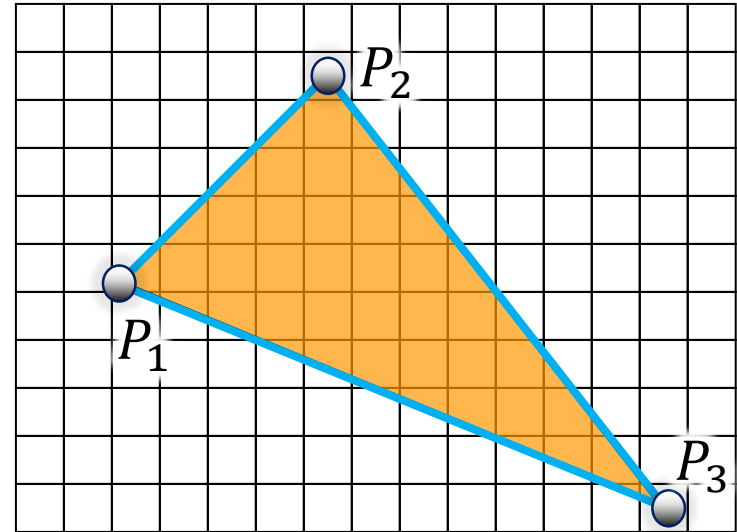
$$\Delta x_{t-1,t} = \frac{x_{t-1} - x_t}{y_{t-1} - y_t} \quad \text{und} \quad \Delta x_{t+1,t} = \frac{x_{t+1} - x_t}{y_{t+1} - y_t}$$

- ▶ hier: Kante $\overline{P_1P_2}$ mit $\Delta x_{12} = \frac{x_1 - x_2}{y_1 - y_2}$ und Kante $\overline{P_3P_2}$ mit $\Delta x_{23} = \frac{x_3 - x_2}{y_3 - y_2}$



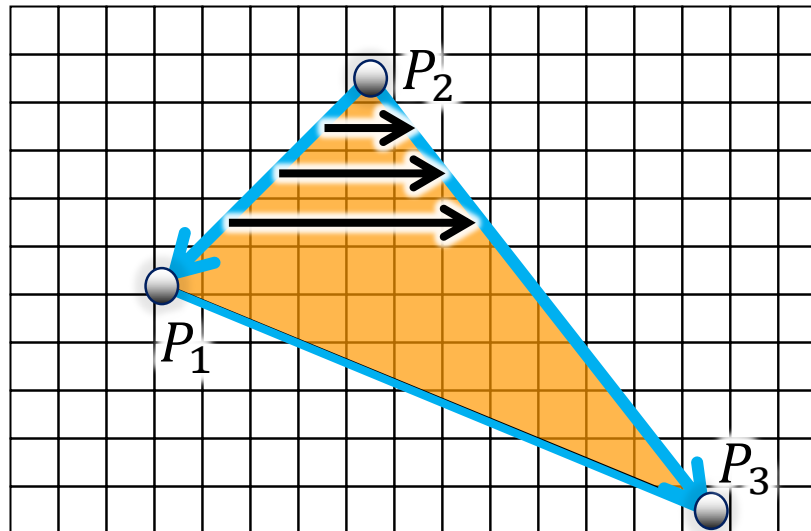
Rasterisierung von konvexen Polygonen (Achtung Spezialfälle!)

- ▶ Start bei $P_t: y = y_t$
 - ▶ linke Kante $x_l = x_t$
 - ▶ Index von Start- und Endpunkt
 $cur_l = t$ und $next_l = t - 1$
 - ▶ rechte Kante $x_r = x_t$
 - ▶ Index von Start- und Endpunkt
 $cur_r = t$ und $next_r = t + 1$
- ▶ do {
 - setze Pixel von x_l bis x_r in Zeile y
 - $y--$;
 - $x_l -= \Delta x_{next_l, cur_l}$;
 - $x_r -= \Delta x_{next_r, cur_r}$;
 - if ($y < y_{next_l}$) { $x_l = x_{next_l}$; $cur_l = next_l$; $next_l--$; }
 - if ($y < y_{next_r}$) { $x_r = x_{next_r}$; $cur_r = next_r$; $next_r++$; }
- } while ($y \geq y_b$);
- ▶ mögl. Optimierung: Bresenham für Kanten



Rasterisierung von konvexen Polygonen

- ▶ Interpolation von Farben, Texturkoordinaten, etc.
 - ▶ interpoliere die Werte an den Vertizes linear entlang der Kanten
 - ▶ analog zur x -Koordinate ($s_l = s_t, \Delta s_{t-1,t} = \dots$)
 - ▶ interpoliere dann entlang jeder Scanline zwischen den Werten an den Kanten
 - ▶ $s = s_l + (x - x_l) \frac{s_r - s_l}{x_r - x_l}, \dots$
- ▶ diese Interpolation ist nicht „perspektivisch korrekt“ (später mehr)



Polygon Rasterisierung

- ▶ bisher haben wir anhand des Pixelmittelpunktes unterschieden, ob der Pixel bei der Rasterisierung eines Polygons gesetzt wird
- ▶ was passiert wenn der Pixelmittelpunkt auf Kanten oder Vertizes liegt?
- ▶ hierfür gibt es Konsistenzregeln bei der Rasterisierung
 - ▶ legen fest, wann ein Pixel in einem der Spezialfälle gesetzt wird
 - ▶ bei benachbarten Dreiecken: verhindert das Pixel gar nicht oder mehrfach gesetzt werden

- ▶ nur bei Interesse:
[http://msdn.microsoft.com/en-us/library/cc627092\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/cc627092(v=vs.85).aspx)

