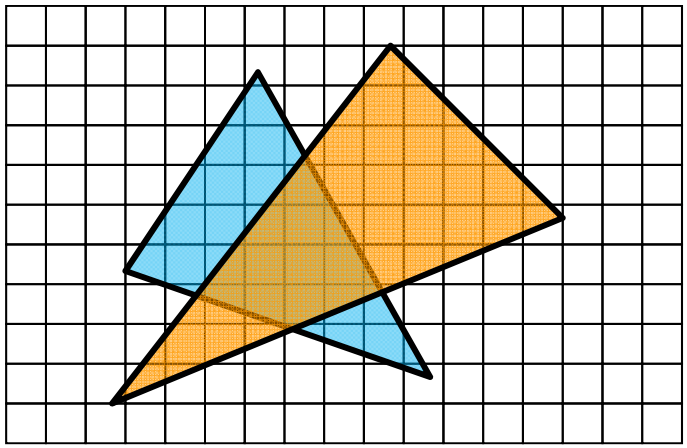
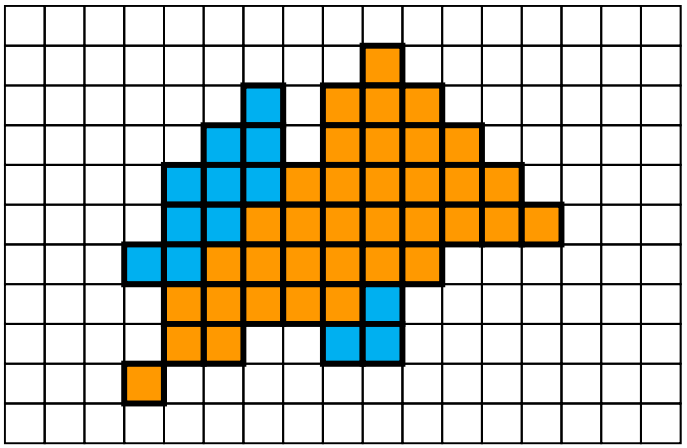
Rendering und Rasterisierung

- 3D-Grafik erzeugt ein Bild aus der Beschreibung einer virtuellen Szene
  - meist werden Oberflächen von Objekten durch Dreiecke beschrieben
  - Bilderzeugung nennt man **Rendering**
- eine Rendering-Technik ist die **Rasterisierung** (rasterization)
  - beinhaltet einige Schritte: Transformationen, Projektionen, Clipping, …
  - wir betrachten zunächst den letzten Schritt davon – Scan Conversion: Bestimmung der Pixel, die von einem grafischen Primitiv bedeckt sind

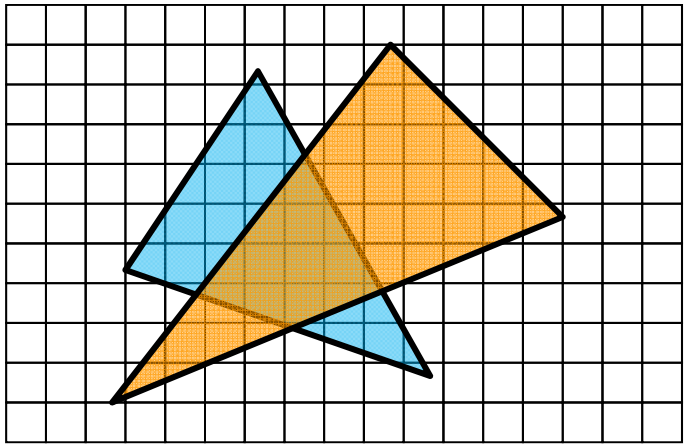


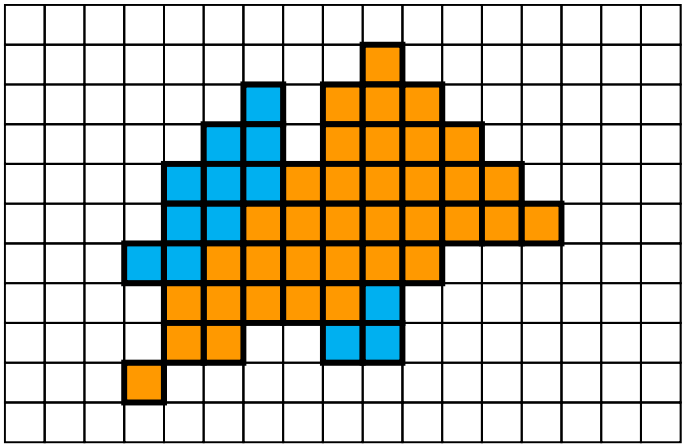


Rasterisierung

- Rasterisierung (scan conversion) von 2D Liniensegmenten und Dreiecken
  - transformiere kontinuierliches Primitiv in diskrete Samples/Pixel
  - zusammenhängende Pixel, keine Lücken
  - wie macht man das akkurat und schnell?

- in der Vorlesung betrachten wir nur einfache Verfahren
  - Übung: effizientere Varianten
  - aktuelle Grafik-Hardware setzt Rasterisierung ein und verarbeitet bis zu 2 Milliarden Dreiecke pro Sekunde

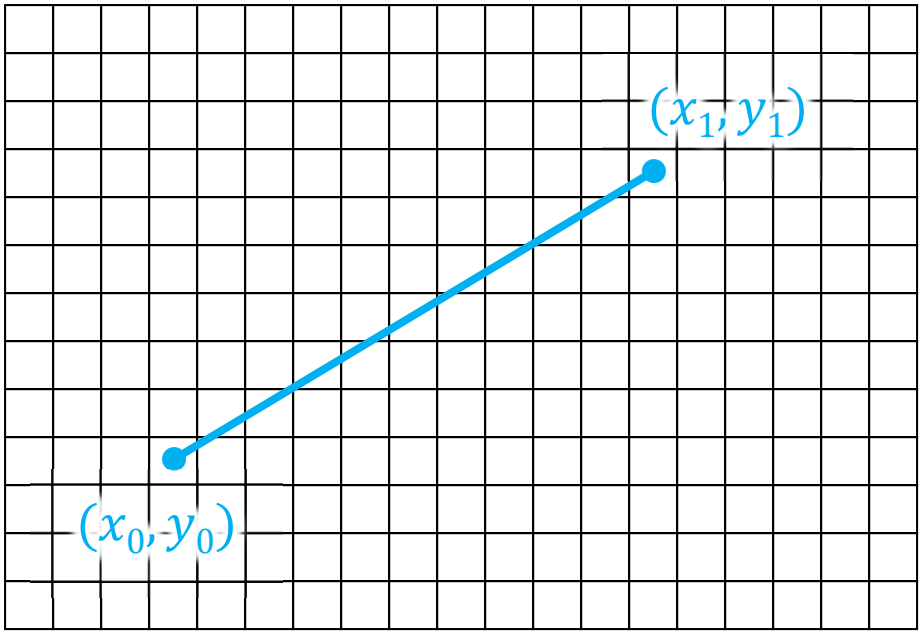




Rasterisierung

Rasterisierung von Linien

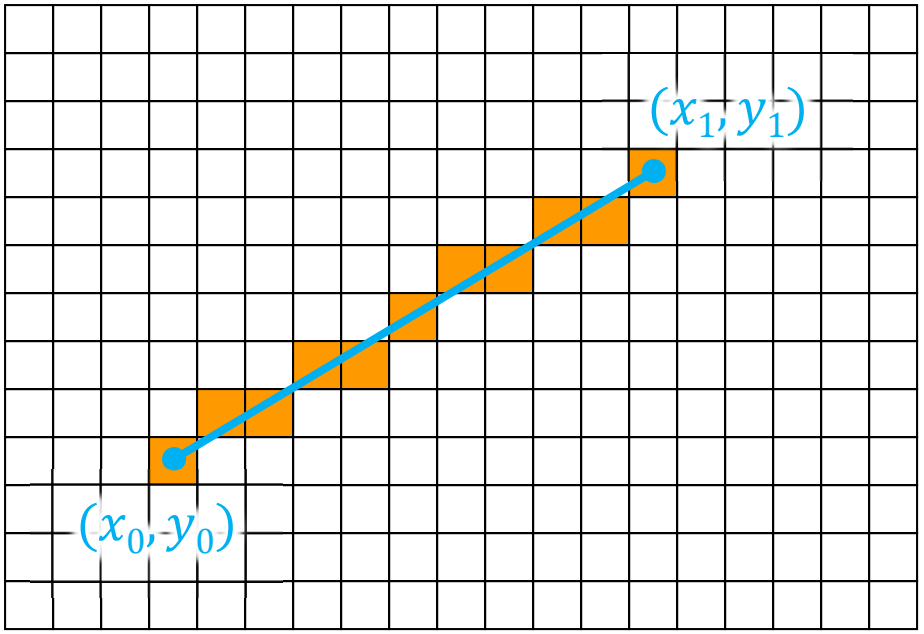
- geg.: Endpunkte des Liniensegments als Integertupel (x_0, y_0), (x_1, y_1)
- ges.: Menge der Pixel, die gesetzt werden sollen
  - optional: wenn die Linie eine bestimmte Dicke hat – welche Pixel muss man mit welcher Intensität setzen



Rasterisierung

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



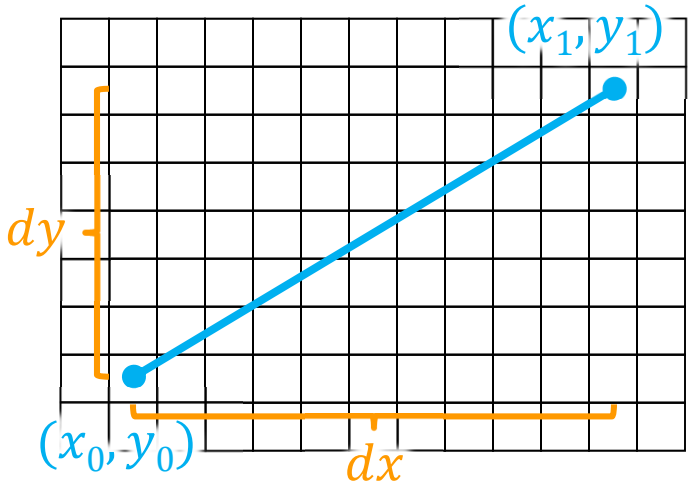
Rasterisierung von Linien

- naiver Algorithmus (Achtung: keine Bereichsüberprüfung etc.)
  - 6 Gleitkommaoperation pro Pixel: multiply, add, round

```
float dx = (x1 – x0);
float dy = (y1 – y0);

// durch diese „Länge“ wird je ein
// Pixel pro Zeile/Spalte gesetzt
float length = max( dx, dy );
dx /= length;
dy /= length;

for ( int i = 0; i <= length; i++ ) {
  float x = x0 + i * dx;
  float y = y0 + i * dy;
  set_pixel( round( x ), round( y ) );
}
```



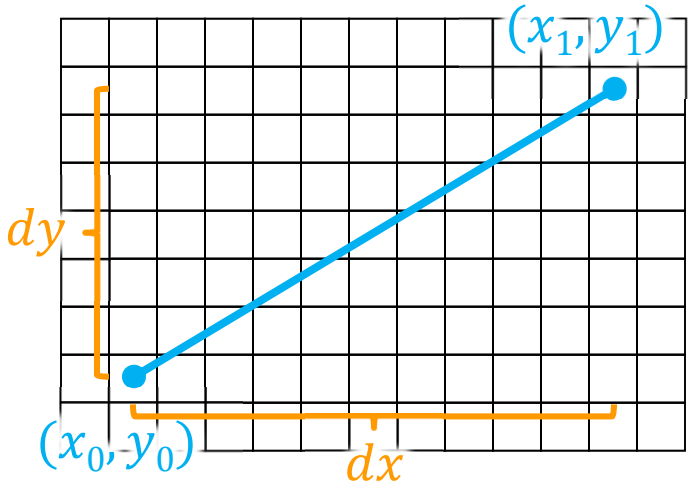
Rasterisierung von Linien

- naiver Algorithmus mit inkrementeller Auswertung
  - 4 Gleitkommaoperation pro Pixel: add, round

```
float dx = (x1 – x0);
float dy = (y1 – y0);

// durch diese „Länge“ wird je ein
// Pixel pro Zeile/Spalte gesetzt
float length = max( dx, dy );
dx /= length;
dy /= length;

float x = x0, y = y0;
for ( int i = 0; i <= length; i++ ) {
  set_pixel( round( x ), round( y ) );
  x += dx; y += dy;
}
```



Rasterisierung von Linien

Bresenham Algorithmus

- effizienter Algorithmus mit inkrementeller Auswertung
  - keine Gleitkommaoperation (interessant für Hardwareumsetzung)
- wir brauchen wir später, aber nicht für einfaches Linienzeichnen

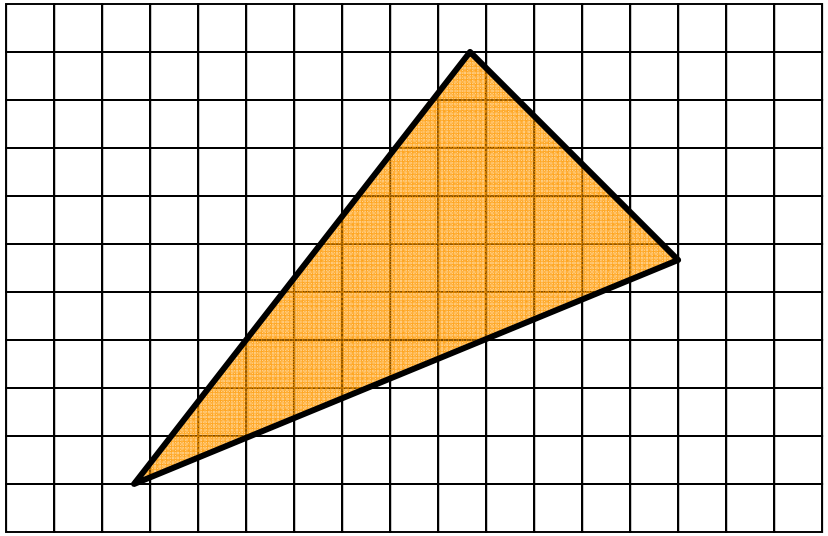
- keine Magie (Erklärung in der Übung):

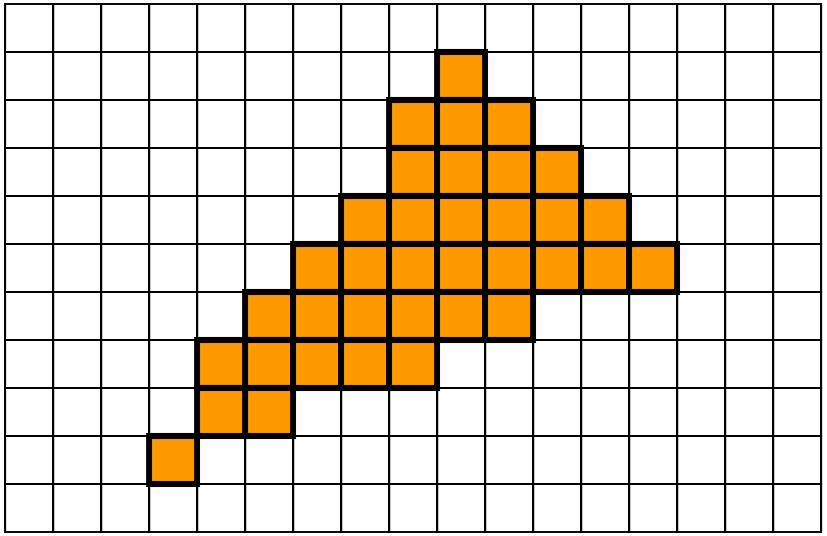
```
int y = y0, dx = 2*(x1-x0), dy = 2*(y0-y1);
int d = (2*y0+1)*(x1-x0)+(x0+1)*dy+2*y1*y0-2*y0*x1;

for ( int x = x0; x <= x1; x++ ) {
  set_pixel( x, y );
  if ( d < 0 ) {
    y = y + 1;
    d = d + dx + dy;
  } else {
    d = d + dy;
  }
}
```

Rasterisierung von Polygonen

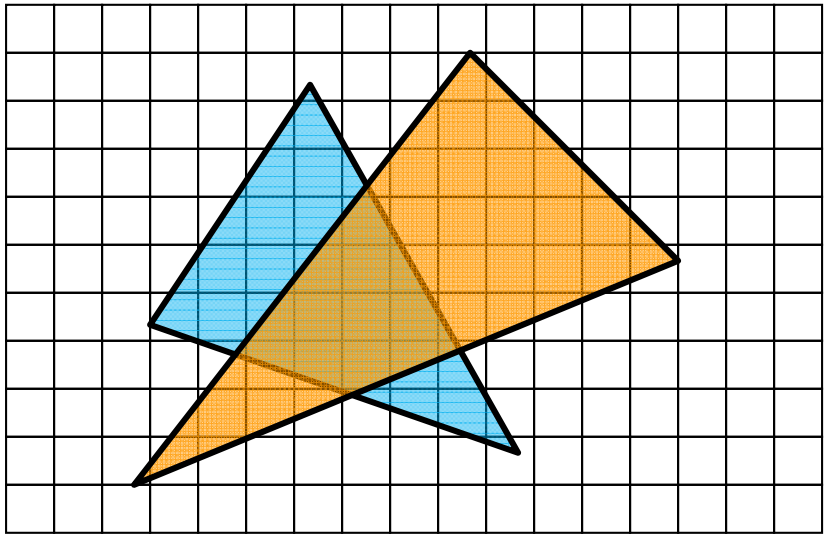
- Polygon-Rasterisierung:
  - geg. ein 2D-Polygon mit n Eckpunkten P_1, …, P_n
  - färbe alle Pixel im Inneren des Polygons

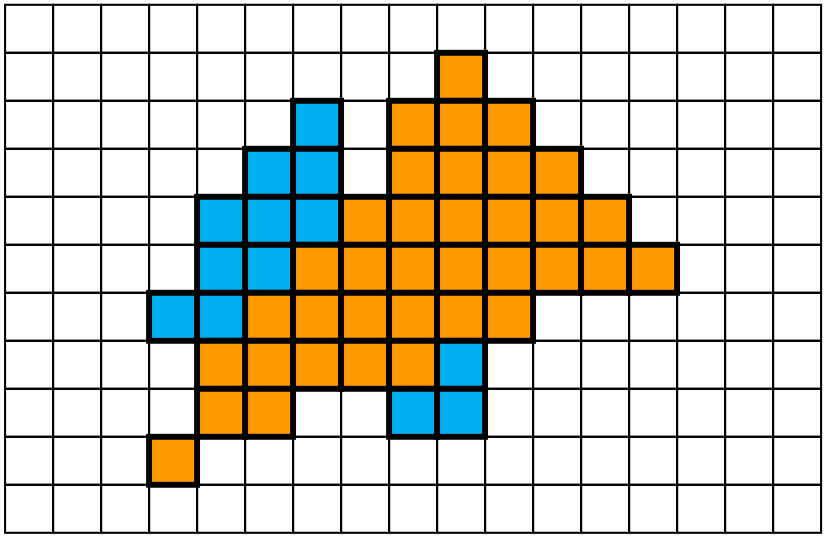




Rasterisierung von Polygonen

- Polygon-Rasterisierung:
  - geg. ein 2D-Polygon mit n Eckpunkten P_1, …, 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
  - einfach: durch Sortierung und Zeichnen „von hinten nach vorne“

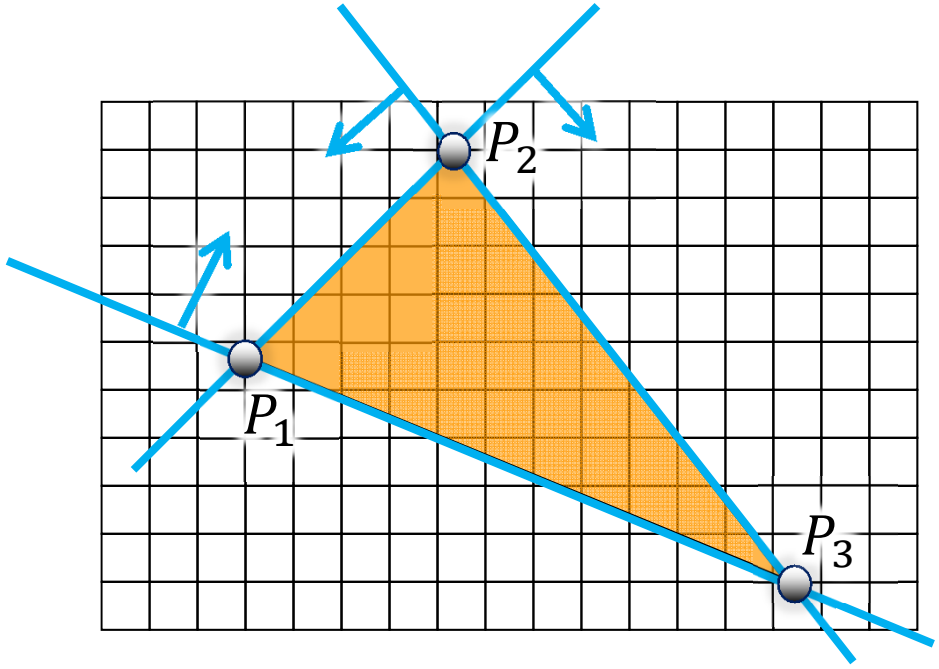




Rasterisierung von Polygonen

Brute Force Ansatz (nur für konvexe Polygone)

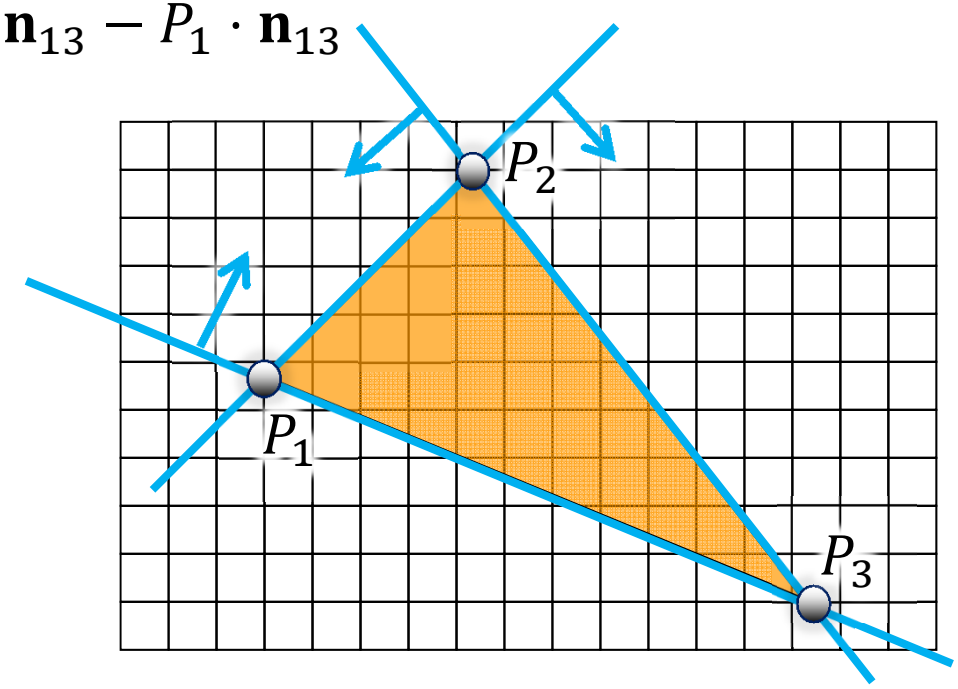
- stelle die Geradengleichungen der Kanten auf
- Orientierung so, dass der gerichtete Abstand aller Punkte im Inneren des Dreiecks zu den Kanten positiv ist → Test für jeden Pixel(mittelpunkt)
- Konvention: lege fest, dass man ein Polygon „sieht“, wenn die Punkte im oder gegen den Uhrzeigersinn angeordnet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



Rasterisierung von Polygonen

Brute Force Ansatz (nur für konvexe Polygone)

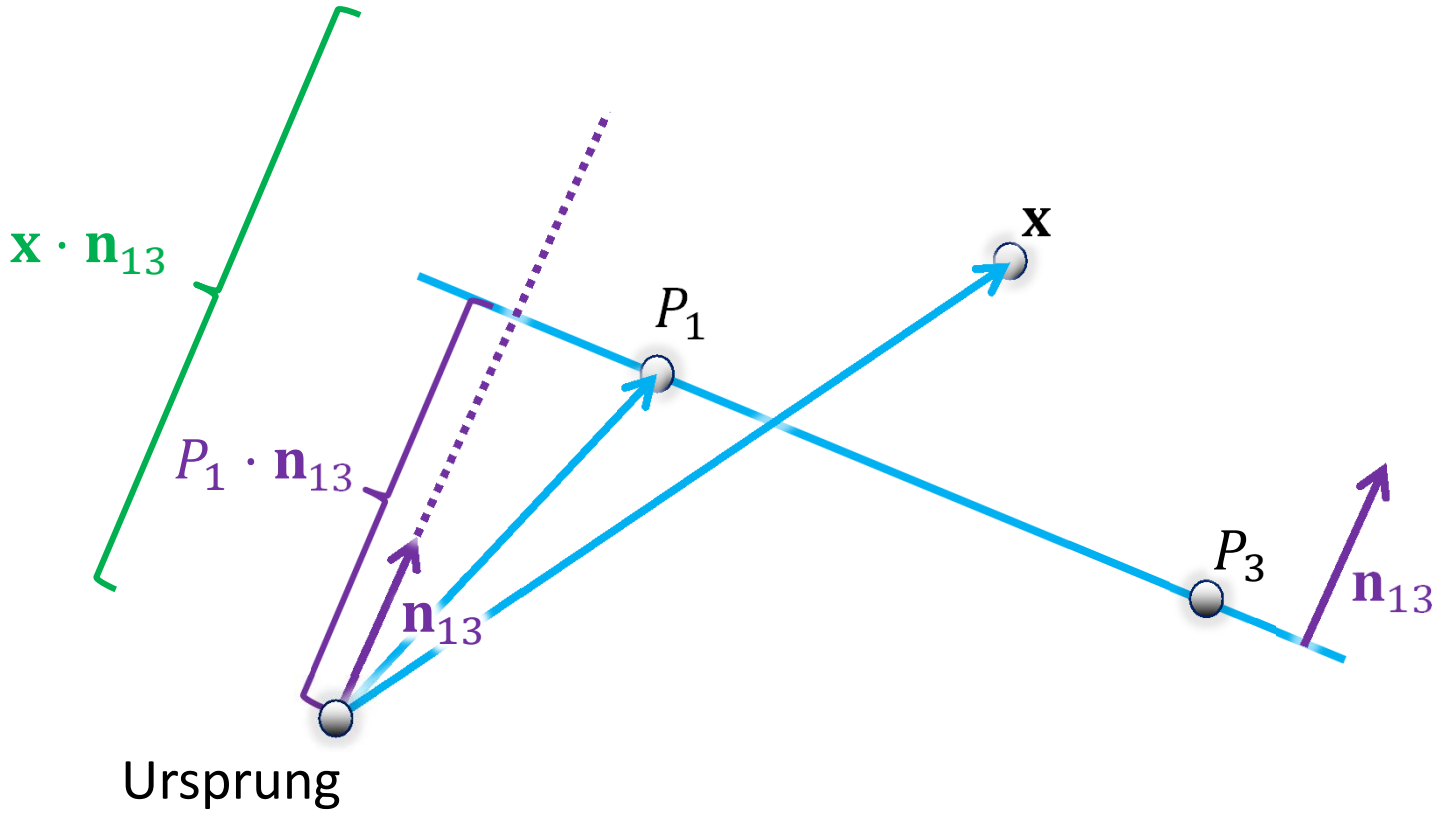
- stelle die Geradengleichungen der Kanten auf
- Orientierung so, dass der gerichtete Abstand aller Punkte im Inneren des Dreiecks zu den Kanten positiv ist → Test für jeden Pixel(mittelpunkt)
- gerichteter Abstand zur Kante $\overline{P_1 P_3}$:
  - Kante $\mathbf{e}_{13} = P_3 - P_1$, Normale $\mathbf{n}_{13} = \left(-\mathbf{e}_{13,y}, \mathbf{e}_{13,x}\right)$
  - Abstand $\mathbf{x} \cdot \mathbf{n}_{13} - P_1 \cdot \mathbf{n}_{13}$



Rasterisierung von Polygonen

Brute Force Ansatz (nur für konvexe Polygone)

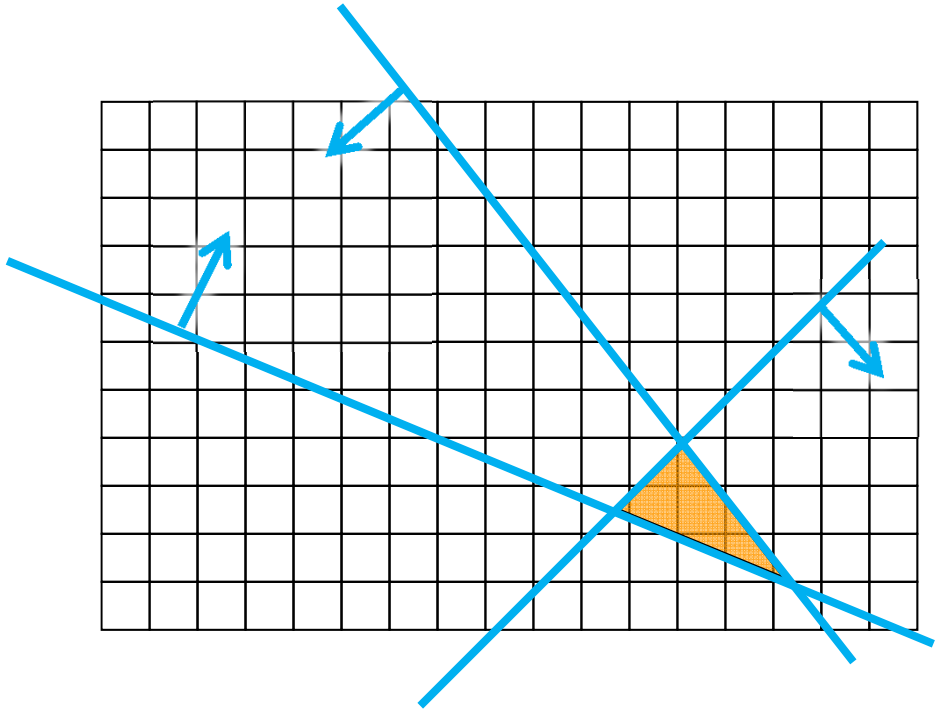
- Kante $\mathbf{e}_{13} = P_3 - P_1$, Normale $\mathbf{n}_{13} = (-\mathbf{e}_{13,y}, \mathbf{e}_{13,x})$
  - wir nehmen o.B.d.A. an $\|\mathbf{n}_{13}\| = 1$
  - $P_1 \cdot \mathbf{n}_{13}$ = Länge des Ortsvektors P_1 projiziert auf Gerade $\mathbf{n}_{13}$ durch den Ursprung



Rasterisierung von Polygonen

Brute Force Ansatz (nur für konvexe Polygone)

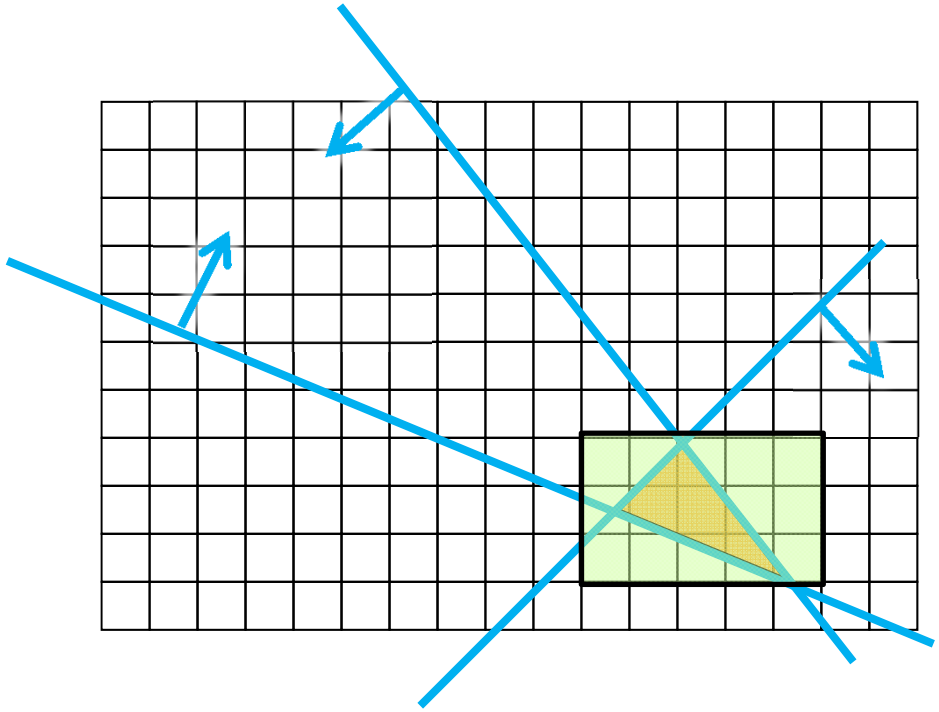
- stelle die Geradengleichungen der Kanten auf
- Orientierung so, dass der gerichtete Abstand aller Punkte im Inneren des Dreiecks zu den Kanten positiv ist → Test für jeden Pixel
- Problem: viele überflüssige Tests bei kleinen Dreiecken



Rasterisierung von Polygonen

Brute Force Ansatz (nur für konvexe Polygone)

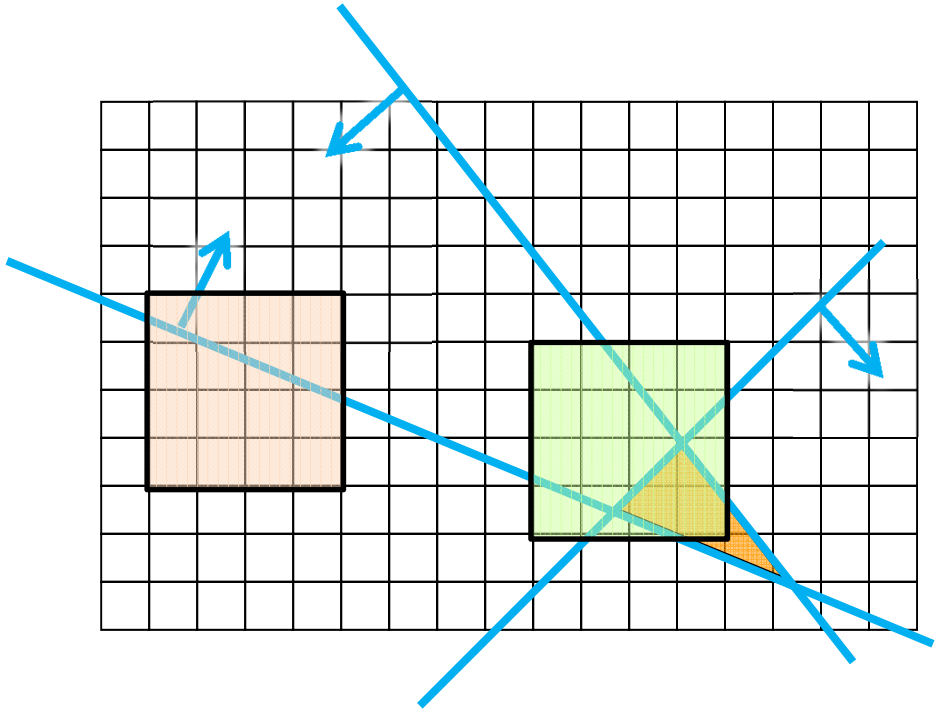
- Problem: viele überflüssige Tests bei kleinen Dreiecken
  - eine mögliche (und einfache) Optimierung:
    berechne Hüll-Rechteck des Dreiecks, teste nur enthaltene Pixel



Rasterisierung von Polygonen

Brute Force Ansatz (nur für konvexe Polygone)

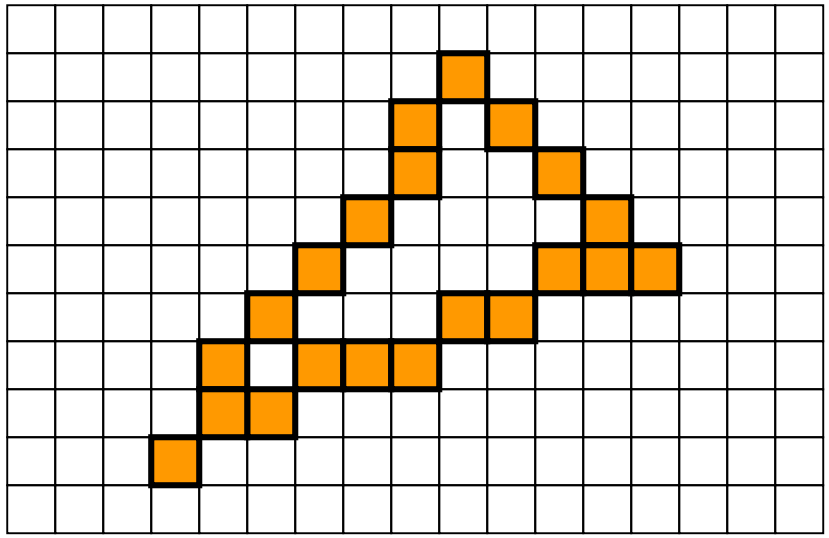
- Problem: viele überflüssige Tests bei kleinen Dreiecken
  - weitere Optimierung: „Binning“
    - teste zuerst größere rechteckige Bereiche, ob sie das Dreieck schneiden oder umschließen (= kein trivial reject)
    - teste dann kleinere Teilbereiche oder einzelne Pixel

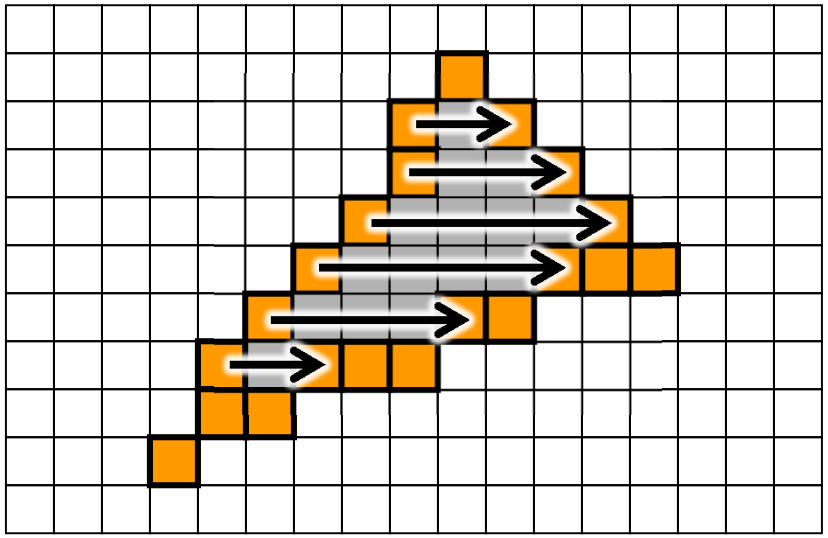


Polygon Rasterisierung

Scanline Polygon Rendering

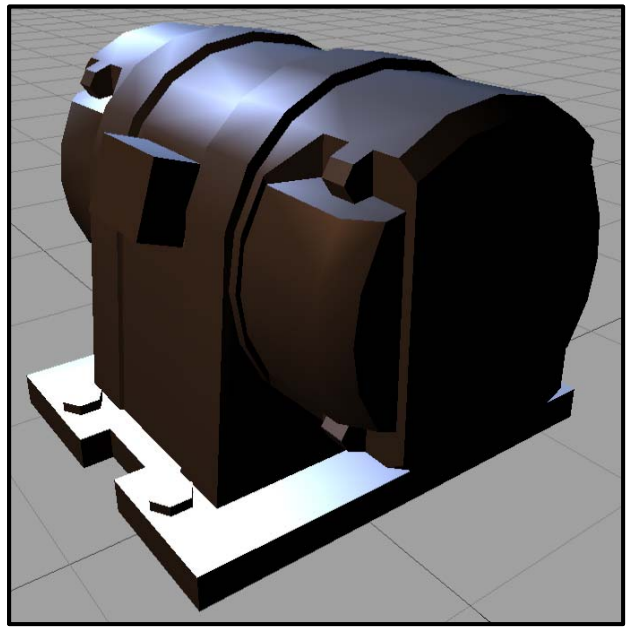
- behandle eine Scanline (Pixel-Zeile) nach der anderen
  - von unten nach oben (oder umgekehrt, je nach Implementation)
- finde Schnitte der Scanline mit dem Polygon
- setze die Pixel in diesem Teil





Polygon Rasterisierung

Scanline Polygon Rendering: weitere Aspekte



- Interpolation von Farben (und Attributen) an den Eckpunkten

- lückenlose und konsistente (= von der Reihenfolge der Dreiecke unabhängige) Rasterisierung an gemeinsamen Kanten