Informatik II

Vorlesung am D-BAUG der ETH Zürich

Vorlesung 7, 11.4.2016

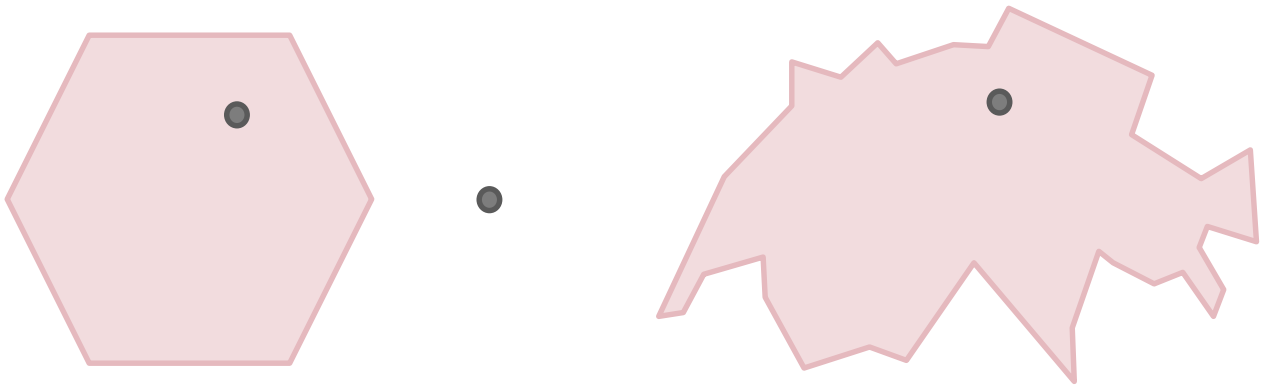
Fallstudie Point-In-Polygon Algorithmus

Diskretisierung: Linien zeichnen

Fallstudie: Point-In-Polygon Algorithmus

Annahme: abgegrenztes Gebiet auf einer Landkarte.

- Repräsentation des Gebietes im Programm?
- Wie entscheiden wir effizient ob ein Punkt «innen» liegt?
- Wie füllen wir das Gebiet mit einer Farbe?



Ein konvexes Gebiet

Ein nicht konvexes Gebiet

Hintergrund: Jordankurven

Schnittfreie Polygone sind *Jordankurven* und zerlegen die Ebene in zwei Gebiete: das Innere und das Äussere

- Das Innere ist beschränkt und
- Das Äussere ist unbeschränkt.

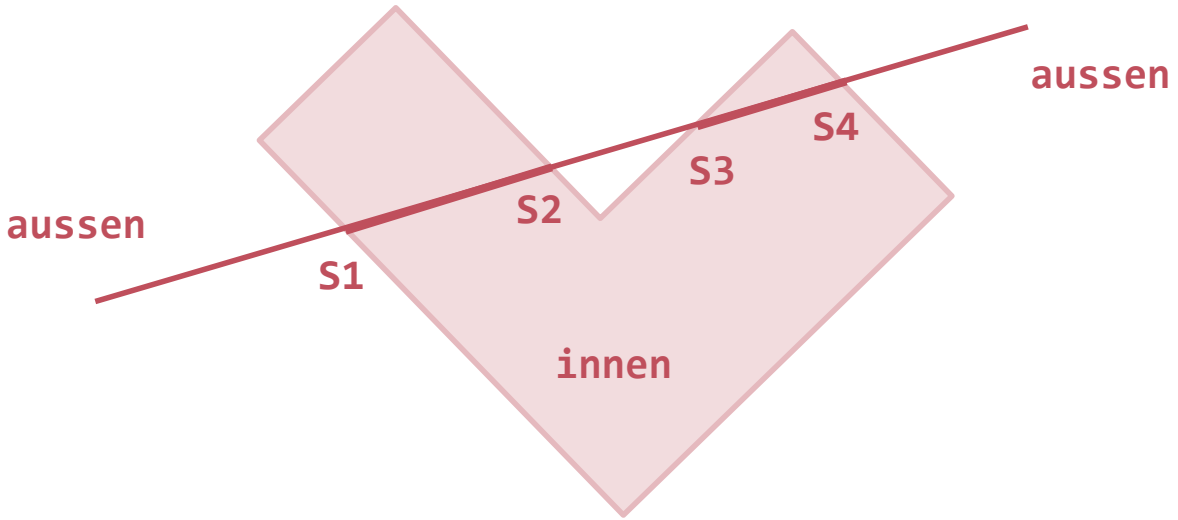
Strikt mathematischer Beweis aus der algebraische Topologie: Abbildung der Einheitskugel

Das kann man ausnutzen, indem man das Polygon mit Geraden schneidet. Man weiss, dass der unbeschränkte Teil der Geraden aussen liegt. Daher funktioniert die folgende Abzählmethode.

Abzählmethode

Zähle auf einer beliebigen Geraden (die den fraglichen Punkt enthält), von unendlich fern kommend, die Anzahl ***echter*** Schnittpunkte mit dem Polygon

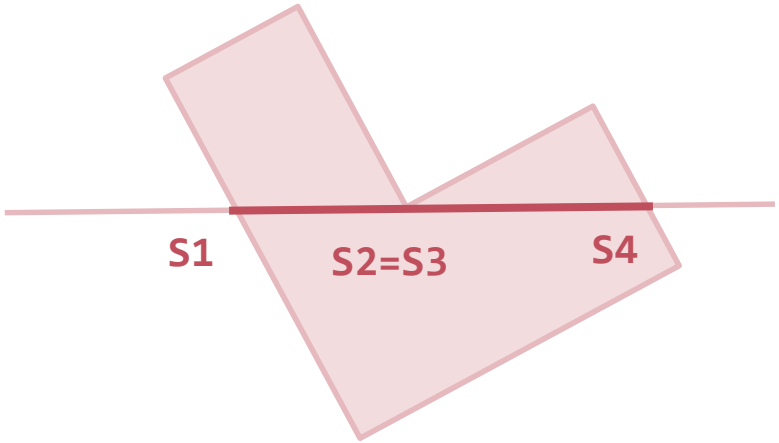
Alle Bereiche der Gerade zwischen ungeradzahligen und geradzahligen Schnittpunkten liegen innen

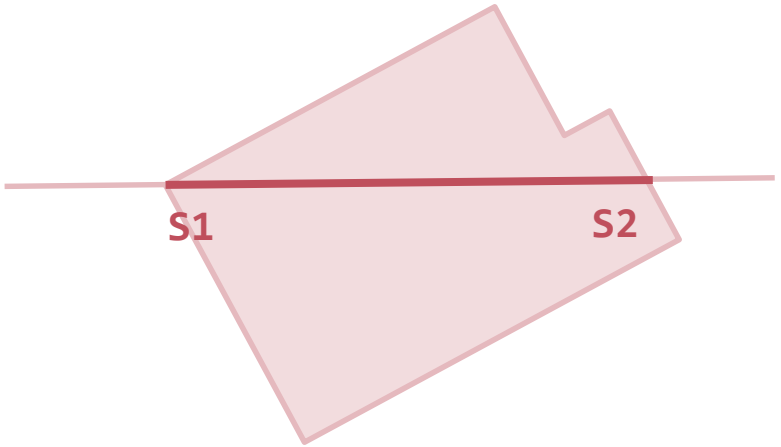


Das funktioniert mit jeder Geraden, wir suchen uns die einfachen Fälle raus (also horizontal oder vertikal)

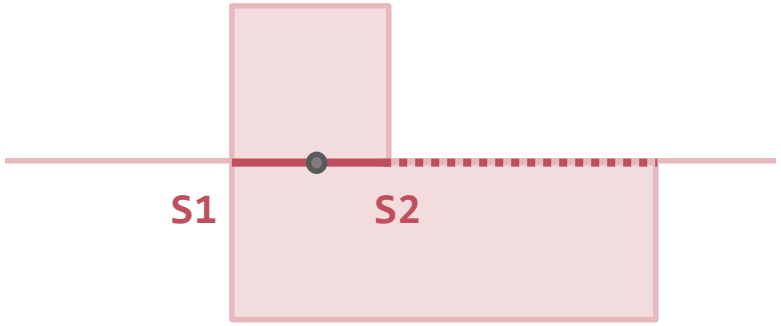
Spezialfälle

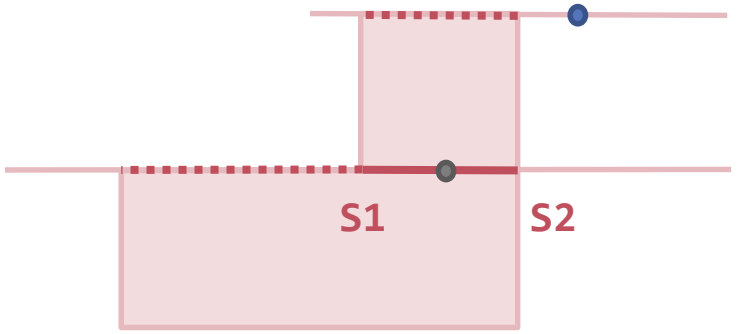
Schnittpunkt = Eckpunkt



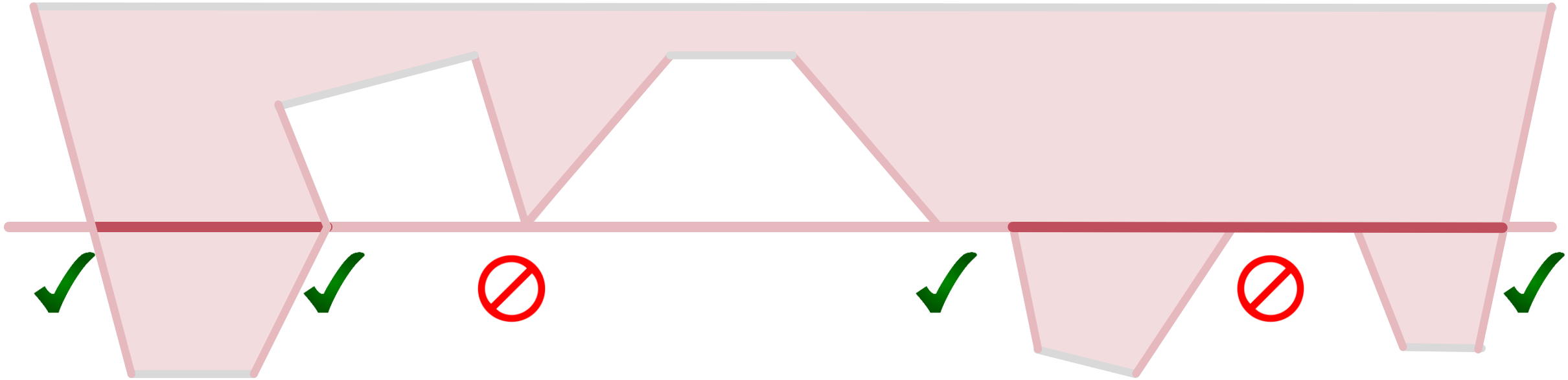


«Schnittpunkt» = Strecke





Echte vs. unechte Schnitte

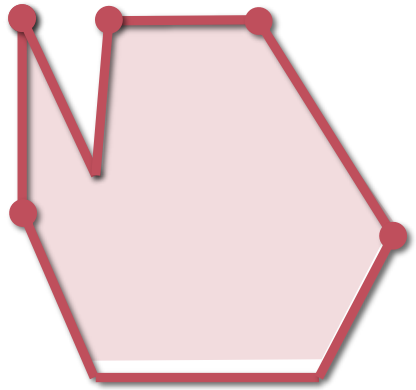


Was charakterisiert echte Schnitte?

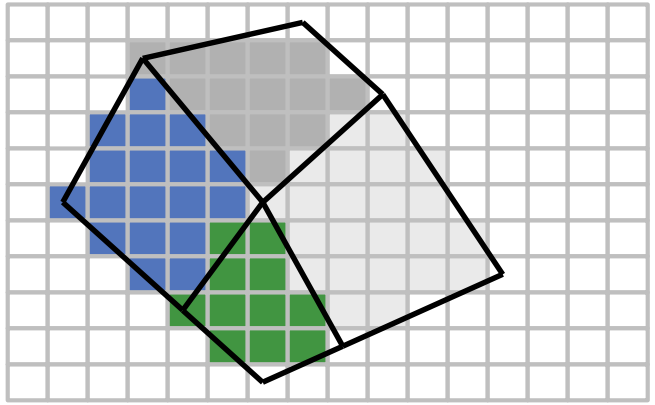
Der Trick

Behandle Eckpunkte einer begrenzenden Kante lageabhängig

- Es werden z.B. nur die oberen Eckpunkte einer Kante mit vertikaler Komponente berücksichtigt
- horizontale Kanten werden ignoriert



- Für eindeutige Segmentierung bei Kachelung: Entscheidung für die Hinzunahme der einen oder anderen Richtung oben/unten, rechts/links (optional: behandle Kanten separat)



Implementation

Polygon

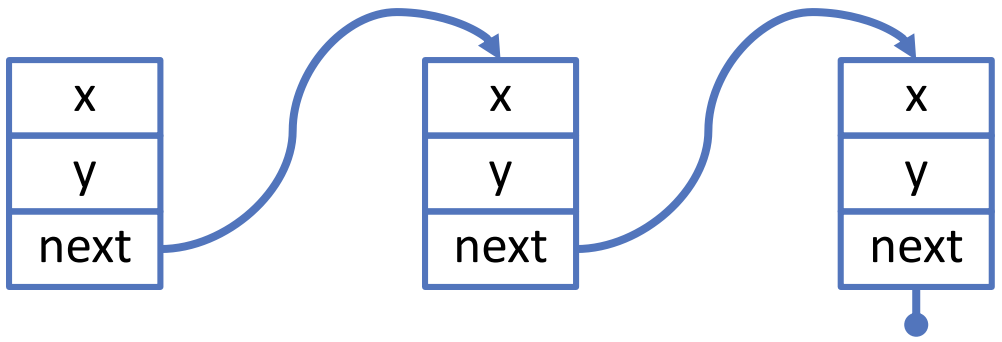
- Folge von Eckpunkten

Benötigte Methoden auf dem Polygon

- Hinzufügen von Eckpunkten
- Abzählen von Schnittpunkten des Polygons mit einer Geraden, dargestellt durch Punkt und Richtung (für die Entscheidung ob Punkt innen liegt)
- Aufzählen von inneren Punkten (für das Füllen)

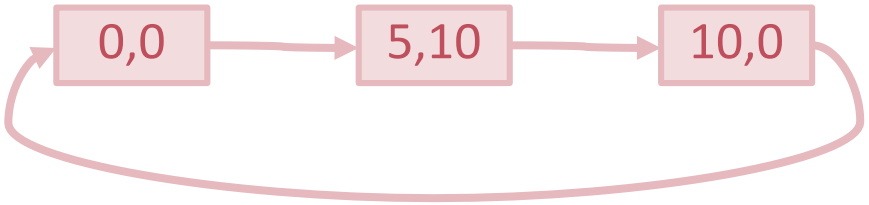
Verkettete Liste von Eckpunkten

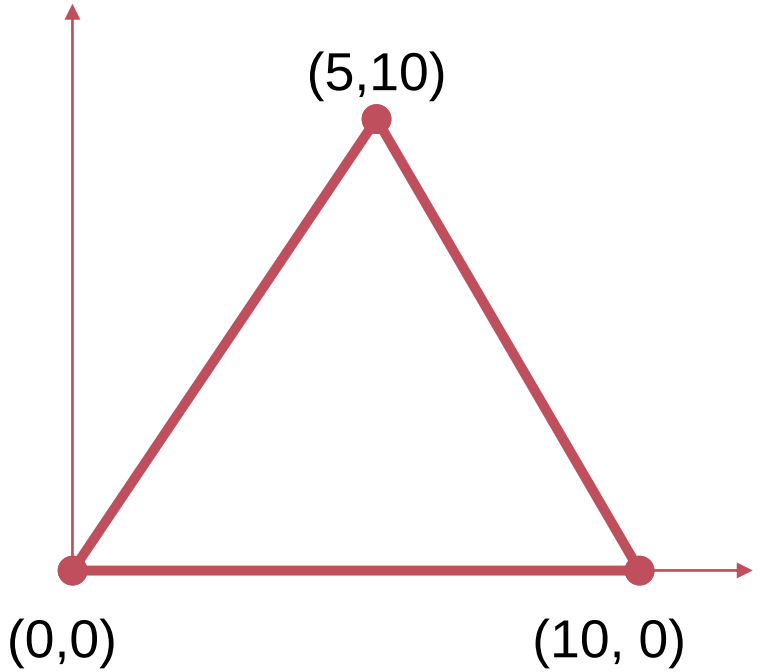
```
public class Vertex {
  public int x, y;
  public Vertex next;

  Vertex(int x0, int y0, Vertex nxt){
    next = nxt;
    x = x0;
    y = y0;
  }
}
```



Polygon: ***zirkuläre*** Liste von Eckpunkten

```
public class Polygon {
  Vertex first, last;
  Polygon() {
    first = null; last = null;
  }
}
```





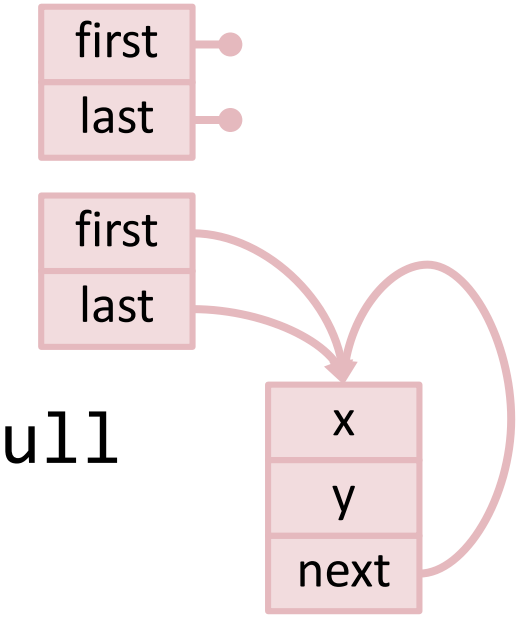
Invarianten der zirkulären Liste

Entweder: **keine Ecke**

```
first == last == null
```

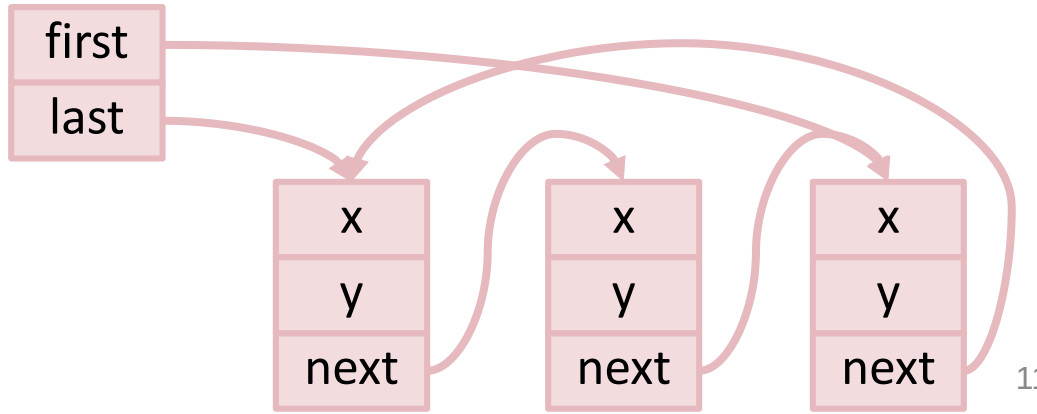
oder: **eine Ecke**

```
first == last && first != null
&& first.next == last
&& last.next = first
```



oder: **mehr als eine Ecke**

```
first != last
&& last.next == first
```



Polygon = Zirkuläre Liste von Eckpunkten

```
public class Polygon {
   Vertex first, last;
   Polygon() {... }

   // add point to the linked circular list
   public void AddPoint(int x, int y) {
      Vertex v = new Vertex(x,y,first);
      if (first == null){
         first = v;
         last = v;
      }
      last.next = v;
      last = v;
   }
}
```

Point in Polygon Algorithmus

Live Coding in der Vorlesung.

Auf separaten Slides als Präsentation.

PointInPolygon

```
public boolean PointInPolygon(int px, int py) {
   if (first == last) // keine oder eine Ecke
      return false;
   Vertex v = first;  // Invariante: first.next != first
   boolean b = false; // "Zähl"variable (false -> true -> false ... )
   do {
      if (IntersectHorizontalLeft(px, py, v, v.next))
         b = !b;
      v= v.next;
   }  while (v != first);
   return b;
}
```

Schnitt

```
boolean IntersectHorizontalLeft(int x, int y, Vertex a, Vertex b) {
   if (b.y < y && y <= a.y) { // Kante von oben nach unten
      return (x-a.x) * (b.y - a.y) <= (b.x-a.x)*(y-a.y);
   }
   else if(a.y < y && y <= b.y) { // Kante von unten nach oben
      return (x-a.x) * (b.y - a.y) >= (b.x-a.x)*(y-a.y);
   }
   else
      return false;
}
```

DISKRETISIERUNG: LINIEN ZEICHNEN

Diskretisierung

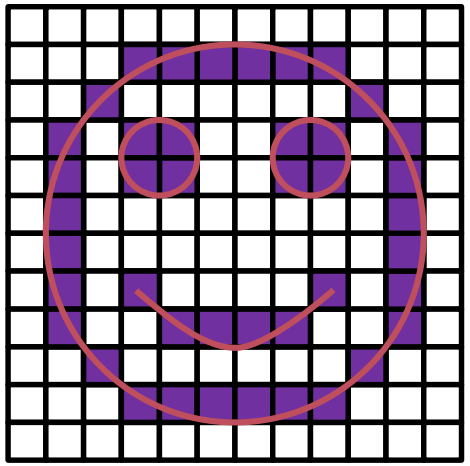
Überführung einer kontinuierlichen Menge Ω in eine diskretisierte, üblicherweise auch endliche, "möglichst passende" Repräsentation M.

Manchmal $M \subset \Omega$.

Diskretisierung: Beispiele

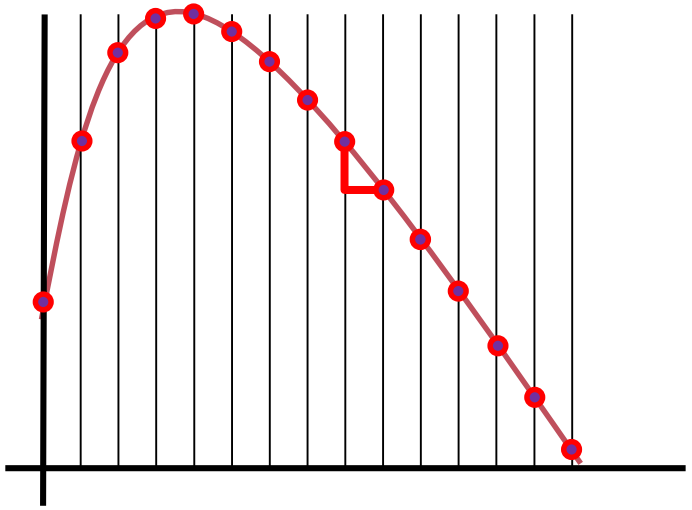
Computergraphik: Abbildungen über $\mathbb{R}^2$ werden zu gefärbten *Pixeln* über einem Rechteck

$$\{0, \dots, w-1\} \times \{0, \dots, h-1\}$$



Numerische Integration:
Diskretisierung des Definitionsbereiches

Differentiale werden zu Differenzen



Linien zeichnen

Angenommen Startpunkt p und Enpunkt q einer Linie im $\mathbb{R}^2$ liegen bereits auf dem Gitter, also

$$p = (p_x, p_y) \in \mathbb{Z}^2$$
$$q = (q_x, q_y) \in \mathbb{Z}^2$$

Linie von p nach q im $\mathbb{R}^2$

$$\overline{pq} = \{p + \lambda(p - q) : 0 \leq \lambda \leq 1\} \subset \mathbb{R}^2$$

Entsprechende Linie im $\mathbb{Z}^2$?

Diskrete Linien

Betrachte flache Linie mit

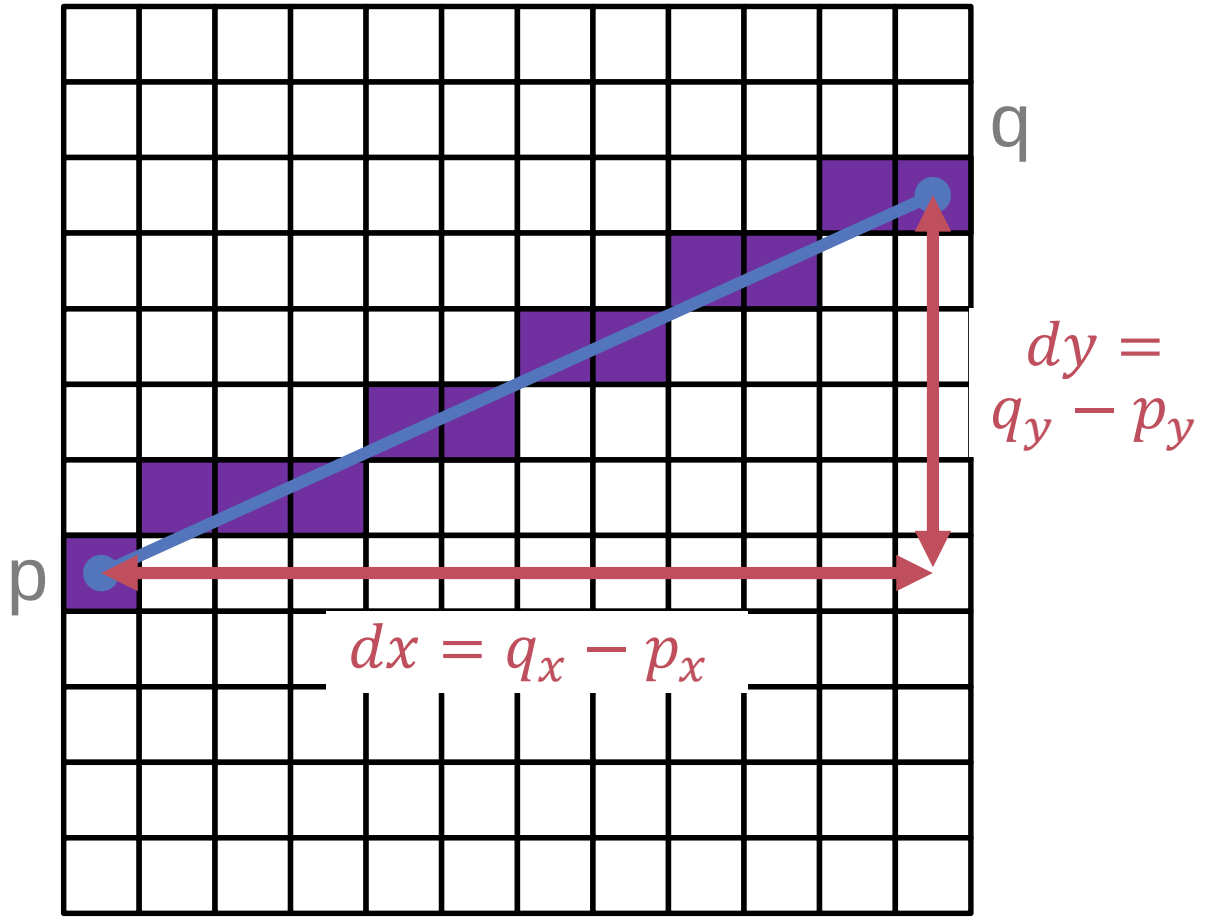
$$|dy| < |dx|$$

Liniengleichung

$$y(x) = p_y + (x - p_x) \cdot \frac{dy}{dx}$$

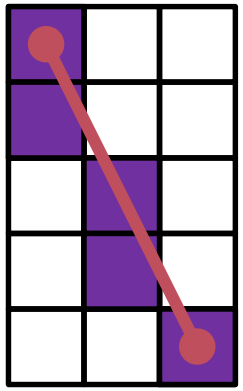
Diskretisiere durch Runden:

$$\hat{y}(x) = p_y + \left\lfloor (x - p_x) \cdot \frac{dy}{dx} + \frac{1}{2} \right\rfloor$$

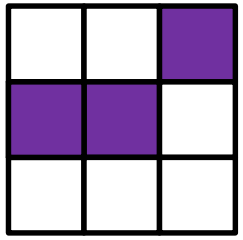


Bemerkungen

Genauso für steile Linien (Tausche Rolle von y und x)

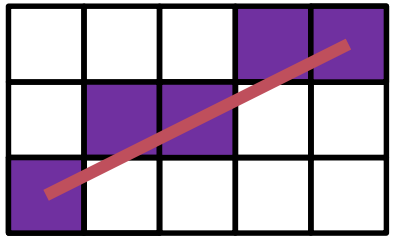


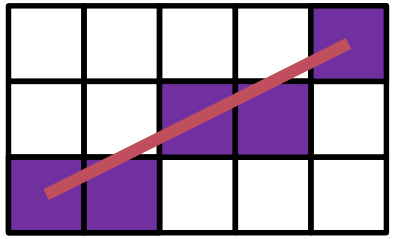
Linien sind 8-verbunden, d.h. benachbarte diskrete Punkte der Linie liegen in der 8er-Nachbarschaft



8 verbunden

Für flache Linien wählt man alle Punkte im Gitter, deren vertikaler Abstand zur Linie maximal 0.5 beträgt.
Bei Gleichheit entweder nur den oberen oder nur den unteren Punkt

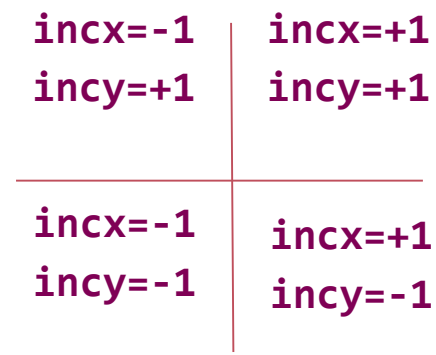


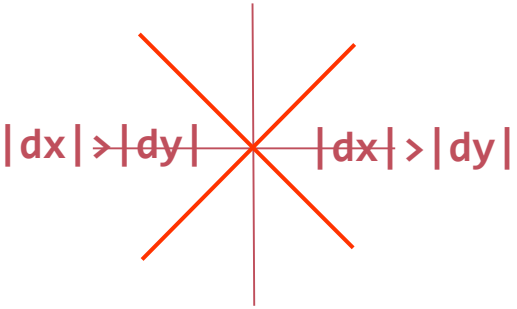


Live Coding: Linien zeichnen

Draw Line Implementation

```
public static void Line(ImageViewer v, int px, int py, int qx, int qy) {
   int dx = (qx-px);
   int dy = (qy-py);
   int incx = 1; if (dx < 0) incx = -1;
   int incy = 1;  if (dy < 0) incy = -1;
   if (dx*incx > dy*incy) { // horizontal
      for (int x = px; x <= qx; x+= incx) {
         int y = py + (int)((double)(x-px) * (dy) / dx + 0.5);
         v.dot(x, y);
      }
   } else { // vertical
      for (int y = py; y <= qy; y+= incy) {
         int x = px + (int)((double)(y-py) * (dx) / dy + 0.5);
         v.dot(x, y);
      }
   }
}
```





Fliesskommaarithmetik ➔ Integer Arithmetik

```
int dx = (qx-px);

int dx2 = dx / 2;

...

Ersetze

  int y = py + (int)((double)(x-px) * (dy) / dx + 0.5);

durch

  int y = py + ((x-px)*dy+dx2)/dx;
```

Superkompakt: Bresenham Algorithmus

```
int dx =  Math.abs(x1-x0);
int sx = x0<x1 ? 1 : -1;
int dy = -Math.abs(y1-y0);
int sy = y0<y1 ? 1 : -1;
int err = dx+dy;

while(x0 != x1 || y0 != y1) {
   dot(x0, y0);
   if (2*err > dy) {
      err += dy;
      x0 += sx;
   }
   if (2*err < dx) {
      err += dx;
      y0 += sy;
   }
}
dot(x1,y1);
```

Idee des Algorithums:

Geradengleichung im $\mathbb{R}^2$:

$$dy \cdot (x - p_x) - dx \cdot (y - p_y) = 0$$

Ständige Korrektur des Diskretisierungsfehlers

$$\varepsilon = dy \cdot (x - p_x) - dx \cdot (y - p_y)$$

während des Durchlaufens der Geraden.

Wir vertiefen das hier nicht.