Füllen von Primitiven

→ Basisproblem der 2D-Graphik

Anwendung:

- füllen beliebiger Flächen (Polygone, Freiformkurven)
- Darstellung von Buchstaben
- dicke Primitive (Linien, Kreise, Kurven), Teilproblem
- in der 3D-Graphik beim Rendering:
  → Helligkeit eines Objekts wird an Ecken oder Kanten bestimmt
  → Objektfläche wird interpolativ gefüllt

Zwei Grundprobleme zu lösen:

1. Welche Pixel sind zu füllen ?
2. Mit welcher Farbe (evtl. Muster) sind sie zu füllen ?

Ansätze:

- Pixelbasiert, rekursiv (Flood-Fill)
- Kantenbasiert, Scanline-Algorithmen

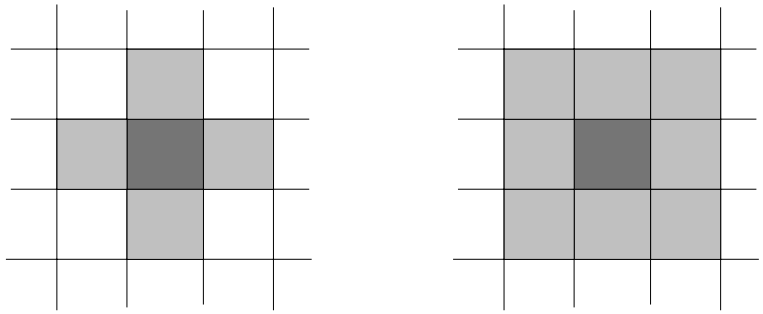
Flood-Fill: Füllen durch Fluten

$\longrightarrow$ Primitiv ist als Bitmap (auf dem Bildschirm) durch Rand mit der Farbe "randFarbe" gegeben

$\longrightarrow$ rekursiver Algorithmus

```
floodFill(x, y, randFarbe, füllFarbe)
{   if (ReadPixel(x, y) ∉ [randFarbe, füllFarbe])
    {  writePixel(x, y, füllFarbe)
       floodFill(x, y-1, randFarbe, füllFarbe)
       floodFill(x, y+1, randFarbe, füllFarbe)
       floodFill(x-1, y, randFarbe, füllFarbe)
       floodFill(x+1, y, randFarbe, füllFarbe)
}}
```

→ obige Version benutzt 4 Nachbar-Pixel
(Variante: Version mit 8 Nachbar-Pixeln)



4 Nachbarpixel 8 Nachbarpixel

→ Ähnlicher Algorithmus in der Bildverarbeitung:

Region Fill (Segmentierung)

Schön am Algorithmus:

- Rand kann beliebig definiert sein

Nachteile des Algorithmus:

- ineffizient (Pixel werden mehrfach besucht)
- Rekursion mit sehr breitem Rekursionsbaum (4 bzw. 8 Aufrufe)
- "Auslaufen" bei unvollständigen Rändern

Füllen von Rechtecken

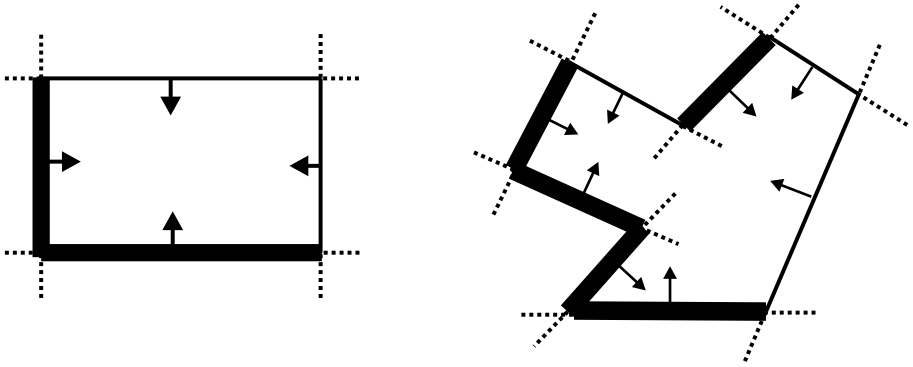
→ Scanlinienverfahren, einfach zu implementieren

→ doppelt verschachtelte Schleife:

```
for y = y_min to y_max do begin
    for x = x_min to x_max do begin
        writePixel(x, y, füllFarbe)
    end
end
```

→ Problem: Pixel auf Kante
(Kanten zwischen Polygonen werden reihenfolgeabhängig gemalt)

Kantenbehandlung beim Füllen



- Polygone werden entgegen dem Uhrzeigersinn definiert
- Pixel auf einer Polygonkante gehören nicht zum Polygon, wenn die Halbebene, die durch die gerichtete Kante definiert wird, unterhalb oder links der Kante liegt
- damit: linke und untere Kanten gehören zum Polygon, rechte und obere nicht

Füllen von Polygonen

Idee:

Scanlinie von unten nach oben mit den Kanten schneiden
und zwischen Schnittpunkten im Inneren des Polygons füllen

Problem:

Wann ist man im Inneren?

Lösung:

Ungeradzahligkeitsregel (odd/even - parity)

$\longrightarrow$ Ausgehend vom linken Rand werden die Schnittpunkte einer Scanlinie mit Polygon bestimmt $((x_1, y_1)(x_2, y_2)...(x_w, y_w))$

$\longrightarrow$ dann gilt:

alle Pixel auf der Scanlinie mit $x < x_1$ sind außen

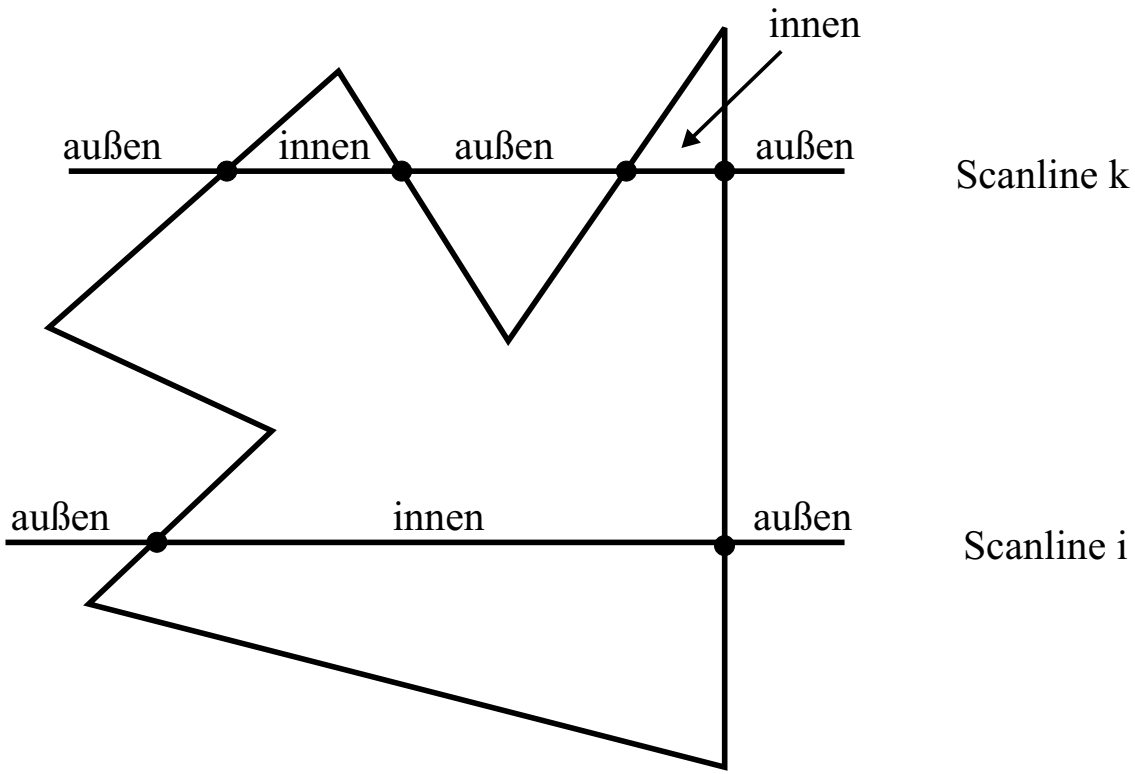
alle Pixel mit $x_{2n+1} < x < x_{2n}$ sind innen

alle Pixel mit $x_{2n} < x < x_{2n+1}$ sind außen

Vereinfacht:

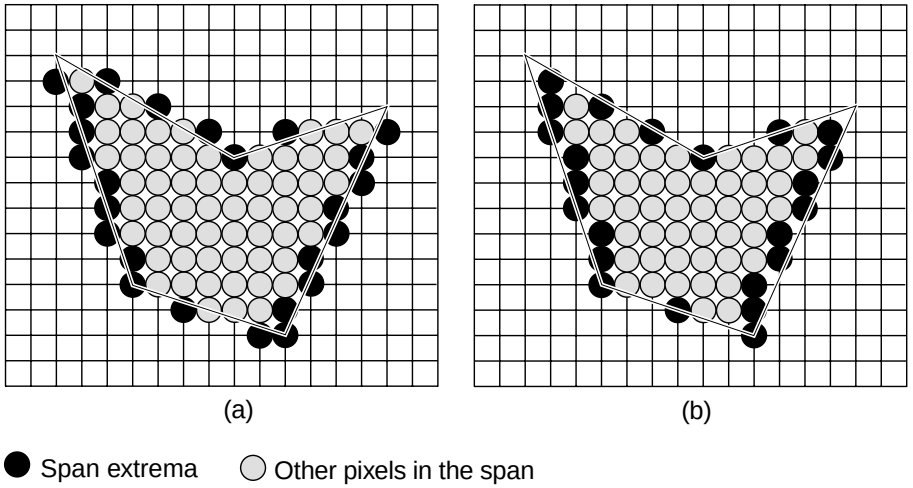
außen anfangen, pro Schnittpunkt: Wechsel zwischen "außen" und "innen"

Beispiel für Ungeradzahligeitsregel mit zwei Scanlinien:



Schema für einen allgemeinen Füllalgorithmus

1. Führe Midpoint-Line-Algorithmus für jede Polygonkante durch
2. Lege generierte Pixel in einer Tabelle für die Spannenenden ab



Problem:

Midpoint-Line-Algorithmus generiert u.U. außenliegende Pixel

Ansatz für einen Algorithmus

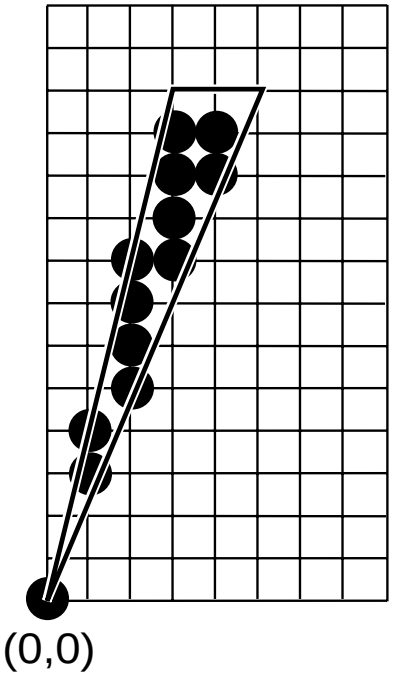
1. Suche alle Schnittpunkte der Scanlinie mit den Polygonkanten
2. Sortiere diese nach aufsteigender x-Koordinate
3. Zeichne nach Ungeradzahligkeitsregel
   - anfangs Flag setzen für “außen”
   - Wechsel nach jedem Schnittpunkt
   - Zeichne nur “innen”

weitere Schwierigkeiten:

1. nichtganzzahlige Schnittpunkte
   - Abrunden, wenn man innen ist
   - Aufrunden, wenn man außen ist
2. ganzzahlige Schnittpunkte
   - am Beginn einer Spanne: innen
   - am Ende einer Spanne: außen
3. doppelte ganzzahlige Endpunkte
   - y_{min} gehört zur Kante
   - y_{max} gehört nicht zur Kante

4. horizontale Kanten
   - untere Kanten zeichnen
   - obere Kanten nicht zeichnen

5. Füllen von "Zacken"
   - es entstehen "leere Spannen"
     $\longrightarrow$ Löcher
   - Problem kann durch Anti-Aliasing behoben werden



$\longrightarrow$ Insgesamt:

Bei entsprechender Implementierung funktioniert der Algorithmus für allgemeine Polygone, bei Selbstüberschneidung sowie bei Kanten der Länge Null

Füllalgorithmus mit Kantentabelle

$\rightarrow$ Ausnutzung der Kanten- und Spannenkohärenz

Spannenkohärenz:

Anzahl der Pixel ändert sich nur selten von Scanlinie i zu$i + 1$

Kantenkohärenz:

Viele Kanten, die von Scanlinie i geschnitten werden, werden auch von Scanlinie $i + 1$ geschnitten

$\Rightarrow$ berechne Schnittpunkt von Scanlinie $i + 1$ mit Kanten aus Schnittpunkten von i

$$x_{i+1} = x_i + \frac{1}{m} \quad y_{i+1} = y_i + 1$$ m: Steigung

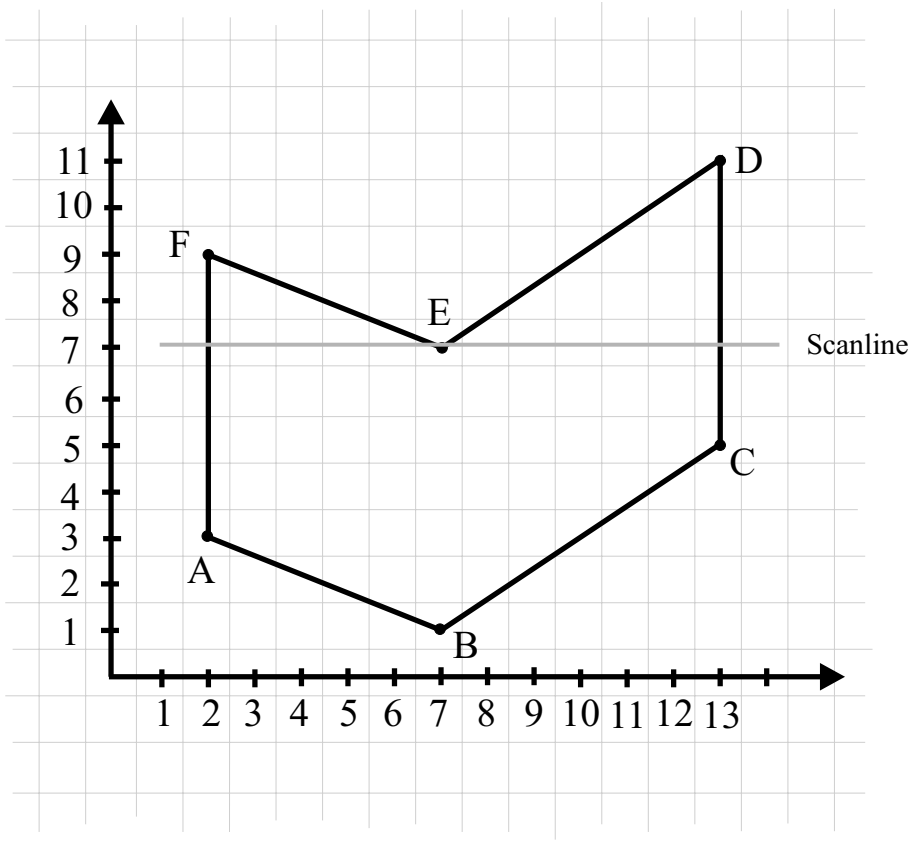
Algorithmenschema

1. Konstruktion einer Kantentabelle (KT) aus den Polygonkanten
   - man speichert pro Kante
     - y_{max}-Koordinate
     - x-Koordinate des Endpunktes mit dem kleineren y-Wert
     - $\frac{1}{m}$ als Inkrement (Inverse Steigung)
   - Kanten in KT nach den y_{min}-Koordinaten sortiert

2. Aufbau einer Aktiv-Kanten-Tabelle (AKT) aus KT,

   Enthält Schnittpunkte der aktuellen Scanlinie mit Polygonkanten nach x sortiert

Beispiel:



für das obige Polygon ergibt sich folgende Kantentabelle

A (2,3), B (7,1), C (13,5), D (13,11), E (7,7), F (2,9)

Aktive Kantentabelle AKT (für die einzelnen Scanlinien):

0	nil
1	$(3, 7, -\frac{5}{2})_{AB}$, $(5, 7, \frac{6}{4})_{BC}$
2	$(3, 7, -\frac{5}{2})_{AB}$, $(5, 7, \frac{6}{4})_{BC}$
3	$(9, 2, 0)_{FA}$, $(5, 7, \frac{6}{4})_{BC}$
4	$(9, 2, 0)_{FA}$, $(5, 7, \frac{6}{4})_{BC}$
5	$(9, 2, 0)_{FA}$, $(11, 13, 0)_{CD}$
6	$(9, 2, 0)_{FA}$, $(11, 13, 0)_{CD}$
7	$(9, 2, 0)_{FA}$, $(9, 7, -\frac{5}{2})_{EF}$, $(11, 7, \frac{6}{4})_{DE}$, $(11, 13, 0)_{CD}$
8	$(9, 2, 0)_{FA}$, $(9, 7, -\frac{5}{2})_{EF}$, $(11, 7, \frac{6}{4})_{DE}$, $(11, 13, 0)_{CD}$
9	$(11, 7, \frac{6}{4})_{DE}$, $(11, 13, 0)_{CD}$
10	$(11, 7, \frac{6}{4})_{DE}$, $(11, 13, 0)_{CD}$
11	nil

Algorithmus insgesamt

$y =$ kleinster in KT vorkommende Wert

AKT als leere Liste initialisieren

Wiederhole bis AKT leer und $y > y_{max}$

- hole aus KT alle die Kanten in AKT, mit $y_{min} = y$ ist;
- Sortiere AKT nach x
- Fülle Pixel der Scanlinie (nach odd/even - parity)
- Entferne alle Einträge mit $y_{max} = y$ aus AKT
- $y := y + 1$
- Aktualisiere x-Werte für alle nicht-vertikalen Kanten
- (Beachte: $x_{i+1} = x_i + \frac{1}{m}$)

Bemerkungen:

- Sonderbehandlung horizontaler Kanten
- AKT wird aus KT abgeleitet, ohne direkte Berechnung der Schnittpunkte!

Füllen von Ellipsen / Kreisen

→ Kreise, Ellipsen und konvexe Polygone haben nur **eine** Spanne pro Scanlinie

Zwei Möglichkeiten:

- Baue Liste aller Spannen, gehe dann diese durch und fülle
- Berechne Spannen nacheinander und fülle sofort

→ Berechnung der Schnittpunkte durch inkrementelle Algorithmen

Füllen mit Mustern

Vorgehensweise:

1. Bestimmen, welche Pixel zu füllen sind
2. Look-up der Farbe im Muster

- Verankerung des Musters

(a) im Ursprung des Koordinatensystems
$\longrightarrow$ Polygone verschieben sich "über" das Muster

(b) in einem festen Punkt im Polygon
$\longrightarrow$ Muster bleibt fix für jede Position des Polygons