Computergrafik SS 2010

Oliver Vornberger

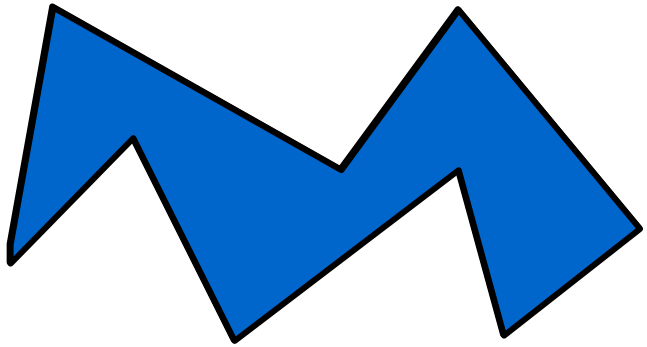
Kapitel 4:
2D-Füllen

Vorlesung vom 19.04.2010

Classroomquiz

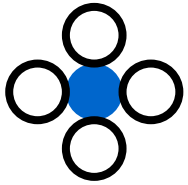
Handyfabrikat	Midlet erfolgreich empfangen	Midlet erfolgreich gestartet	bei Quiz erfolg-reich abgestimmt
HTC HD2	nein	nein	nein
Motorola Milestone (Android)	ja	nein	nein
Motorola Razr v8	ja	nein	-
Nokia E90 (V07.40.1.2)	ja	ja	(vermutlich) ja
O2 XDA Orbit 2	nein	nein	nein
Samsung M1 360 Vodafone	nein (selbst heruntergeladen)	ja	ja
Samsung SGH E900	nein	nein	nein
Sony Ericson Cybershot	ja	ja	unklar
Sony Ericson K850i	ja	ja	ja
Sony Ericsson	ja	ja	ja
Sony Ericsson C702	nein	nein	nein
Sony Ericsson K800i	selbst heruntergeladen	ja	ja
T-mobile MDA	nein	nein	nein
U900Soul	nein (selbst heruntergeladen)	ja	noch nicht teilgenommen

Füllverfahren

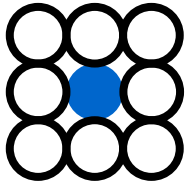


- Universelle Füllverfahren
  (Zusammenhangseigenschaften)
- Scan-Line-Verfahren
  (Geometrie)

Universelle Füllverfahren



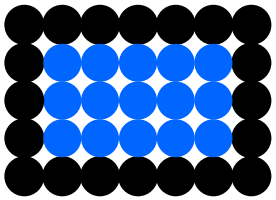
4-way-stepping



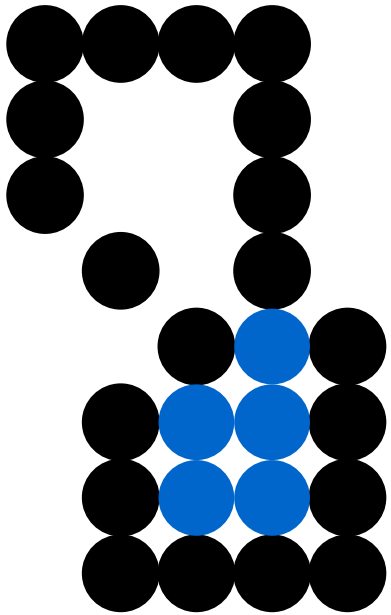
8-way-stepping

Beginnend beim Saatpixel:

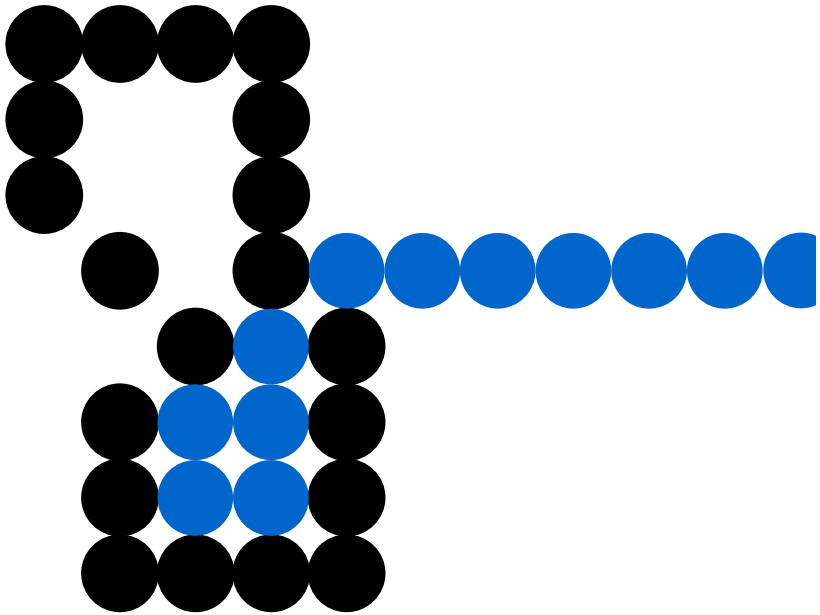
färbe alle Nachbarn, bis Umgrenzung erreicht ist.



Probleme beim universellen Füllen



4-way-stepping



8-way-stepping

Rekursives Füllen

`boolean rangeOK(x,y)` true, falls Punkt x,y innerhalb des Bildbereichs

`boolean getPixel(x,y)` true, falls Vordergrundfarbe an Punkt x,y

`void setPixel(x,y)` setze Vordergrundfarbe an Punkt x,y

```
public void boundaryFill(int x, int y){
if (rangeOk(x,y) && !getPixel(x,y)){
  setPixel(x,y);
  boundaryFill(x+1,y);
  boundaryFill(x,  y+1);
  boundaryFill(x-1,y);
  boundaryFill(x,  y-1);
}
```

Rekursives Leeren

`delPixel(x,y)` setze Hintergrundfarbe an Punkt x,y

```
public void boundaryEmpty(int x, int y){
  if (rangeOk(x,y) && getPixel(x,y)){
    delPixel(x,y);
    boundaryEmpty(x+1,y);
    boundaryEmpty(x+1,y+1);
    boundaryEmpty(x  ,y+1);
    boundaryEmpty(x-1,y+1);
    boundaryEmpty(x-1,y);
    boundaryEmpty(x-1,y-1);
    boundaryEmpty(x  ,y-1);
    boundaryEmpty(x+1,y-1);
  }
}
```

boundaryFill-Implementation

Java-Applet:

~cg/2010/skript/Applets/2D-basic/App.html

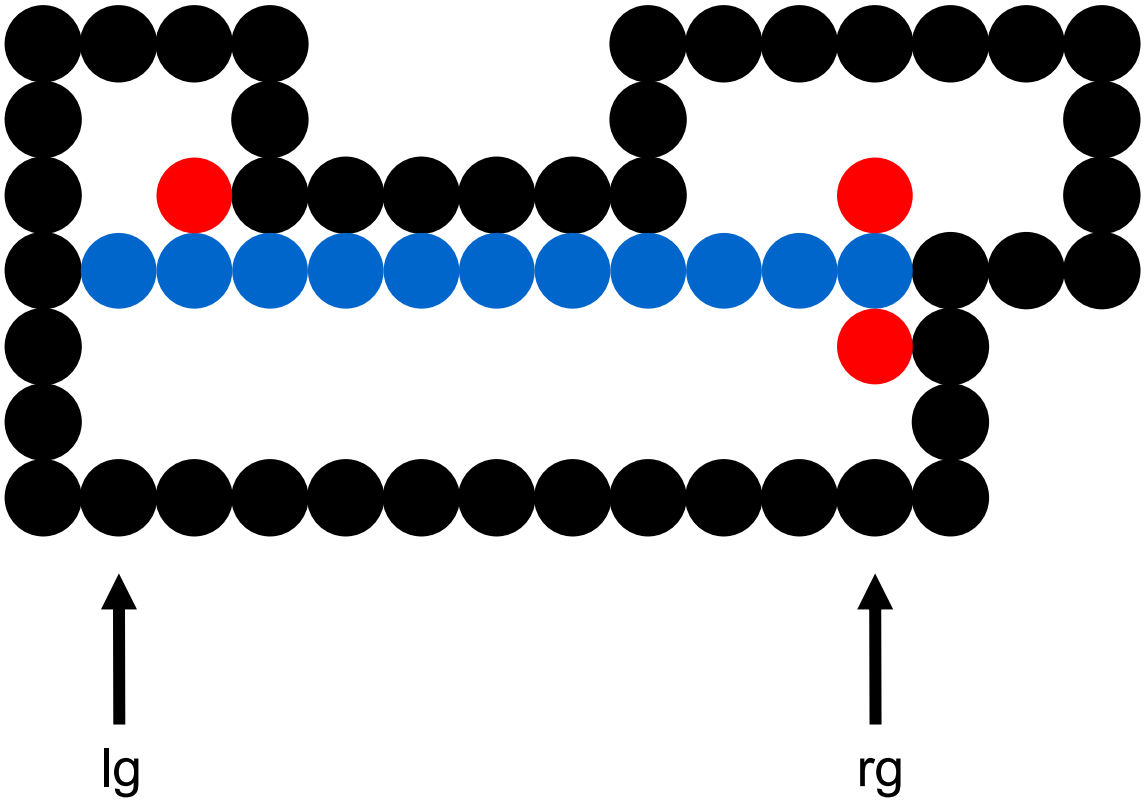
Analyse von boundaryFill

- jedes Pixel (bis auf Randpixel)
  wird 4-mal auf den Keller gelegt

- besser: lege Repräsentant auf Keller

- Repräsentant färbt horizontale Linie
  und legt Repräsentanten für Nachbarlinien
  auf Keller

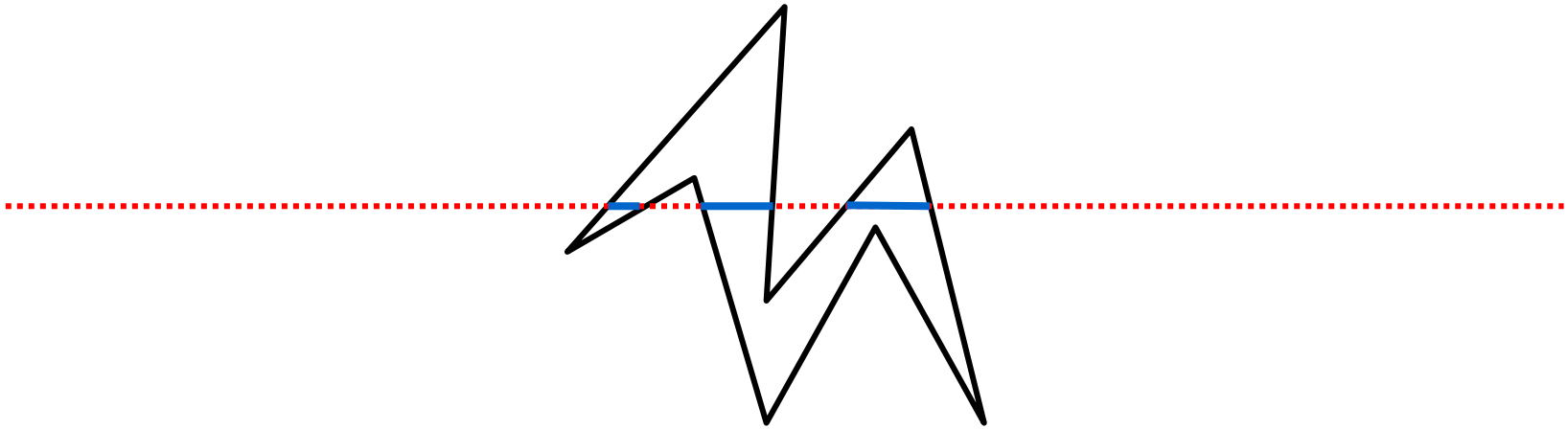
fillRowByRow



fillRowByRow: Algorithmus

```
public void fillRowByRow(int x, int y) {
  int lg;
  int rg;
  int px = x;
  while(!getPixel(x,y)) {setPixel(x,y); x--;}
  lg = x+1;
  x  = px + 1;
  while(!getPixel(x,y)) {setPixel(x,y); x++;}
  rg = x-1;
  for(x = rg; x >= lg; x--) {
    if(!getPixel(x, y-1)) fillRowByRow(x,y-1);
    if(!getPixel(x, y+1)) fillRowByRow(x,y+1);
  }
}
```

Scan-Line-Verfahren



Bewege waagerechte Scan-Line von oben nach unten über das Polygon und färbe entsprechende Abschnittsgeraden

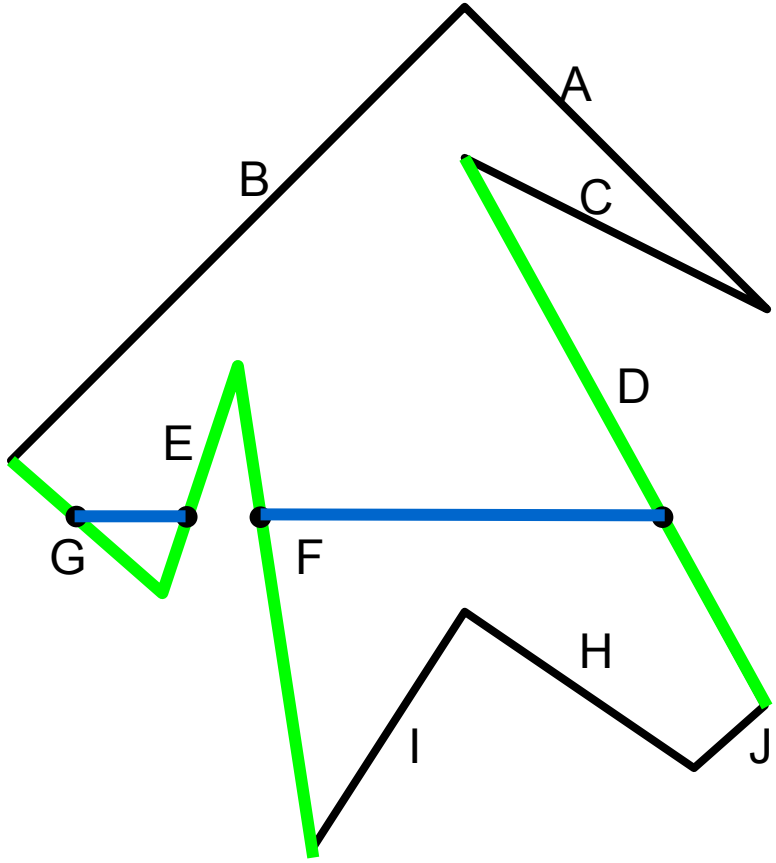
Scan-Line-Verfahren: Detail

1. Sortiere Kanten nach größtem y-Wert

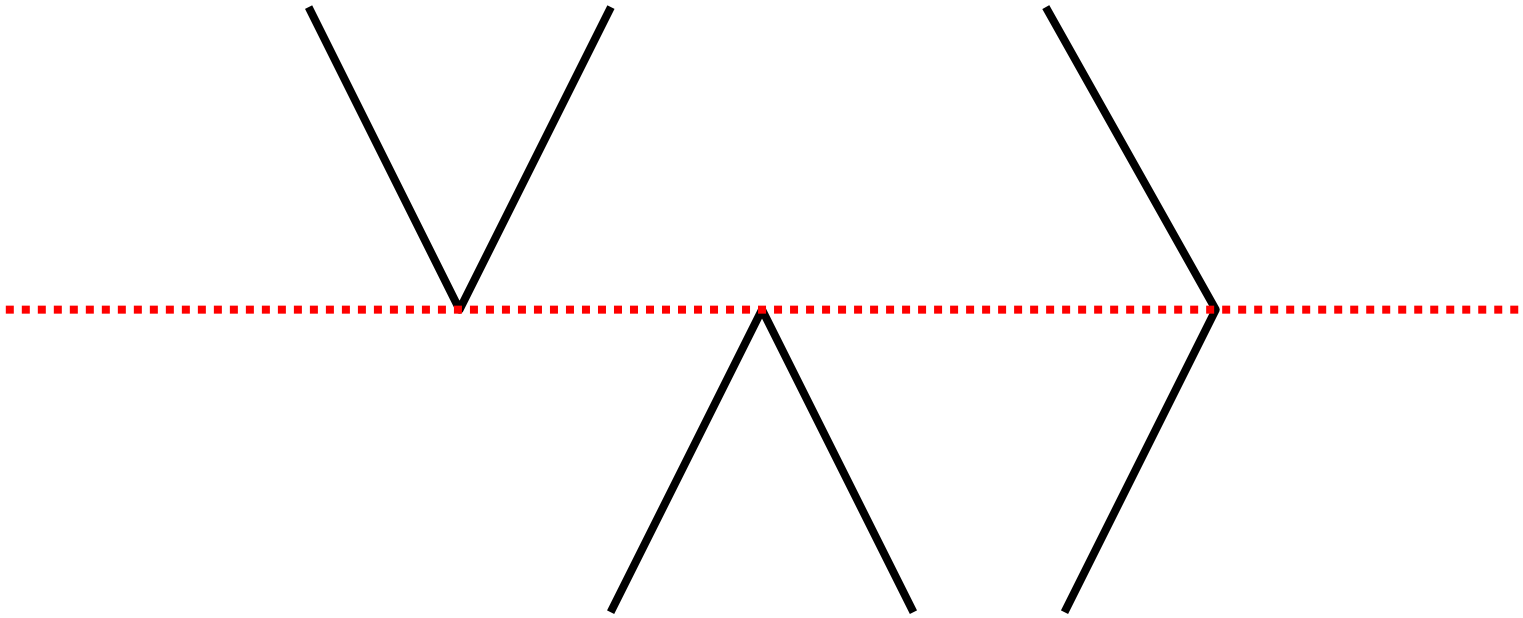
2. Bewege Scan-Line von oben nach unten

3. für jede Position der Scan-Line:
   ermittele aktive Kanten
   berechne Schnittpunkte mit Scan-Line
   sortiere die Schnittpunkte nach x Werten
   färbe abwechselnd zwischen Schnittpunkten

Scan-Line-Verfahren: Beispiel

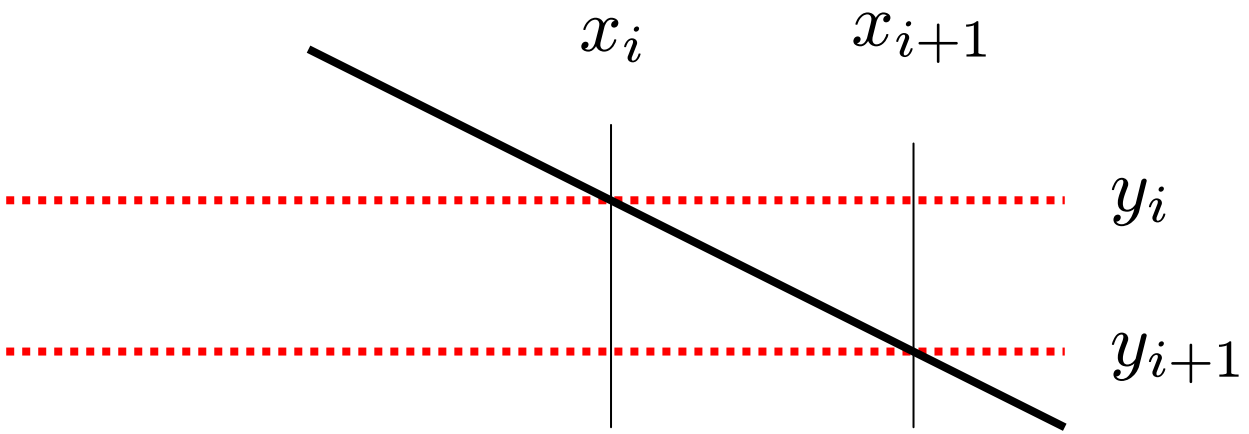


Problemfälle



doppelt doppelt einfach

Schnittpunkte fortschreiben

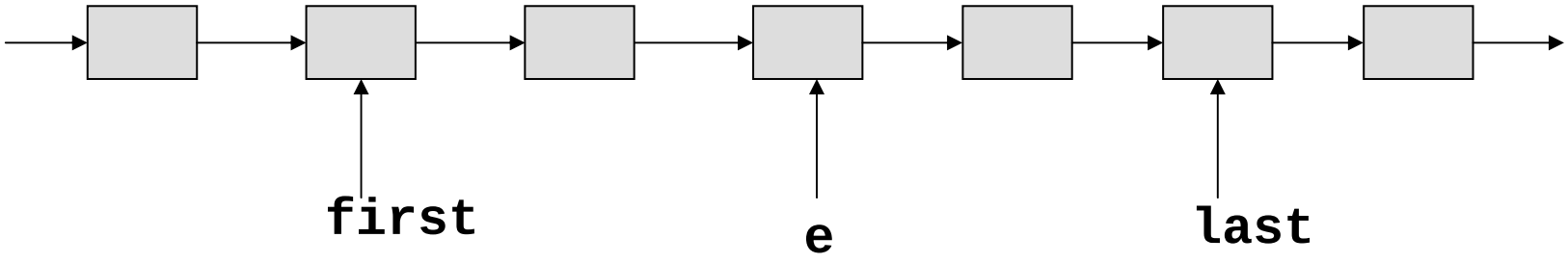


$$s = \frac{y_i - y_{i+1}}{x_i - x_{i+1}} \qquad y_i - y_{i+1} = 1$$

$$x_i - x_{i+1} = \frac{y_i - y_{i+1}}{s} \qquad x_{i+1} = x_i - \frac{1}{s}$$

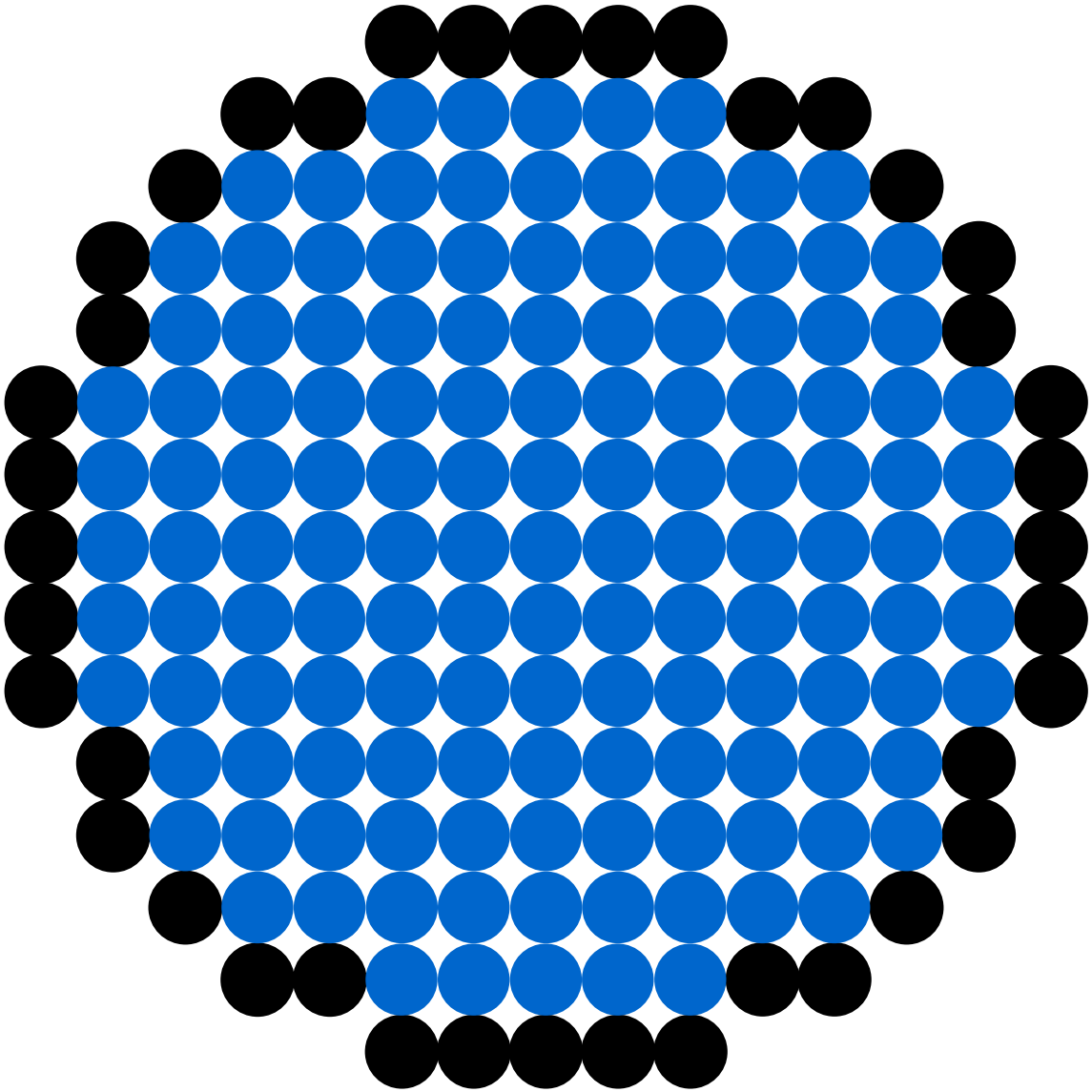
Datenstruktur für Kante

```
public class Edge {
       int y_top;       // groesster y-Wert
       int delta_y;     // Ausdehnung in y-Richtung
       double delta_x; // inverse Steigung
       double x_int;   // errechneter Schnittpunkt
       Edge next;       // Verweis auf naechste Kante
}
```




```
if (e.delta_y) > 0) {
  e.delta_y--;
  e.x_int = e.x_int – e.delta_x;
  e = e.next;
}
```

Scan-Line-Verfahren für Kreis

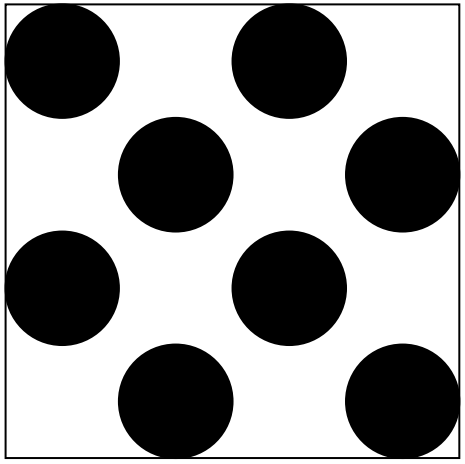


Scan-Line-Implementation

Quelltext:
~cg/2010/skript/Sources/scan.txt

Java-Applet:
~cg/2010/skript/Applets/2D-basic/App.html

Dither-Matrix: Definition

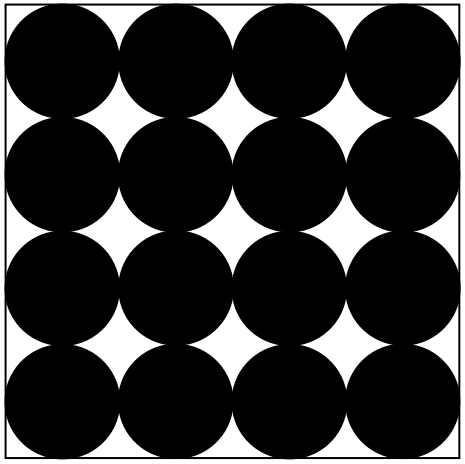


0	8	2	10
12	4	14	6
3	11	1	9
15	7	13	5

Eine *n x n* Dithermatrix enthält gleichmäßig verteilt alle Zahlen aus dem Intervall [$0..n^2 - 1$]

Für Grauwert $0 \leq k \leq n^2$
setze alle Pixel mit Eintrag < k

Dither-Matrix: Beispiel



0	8	2	10
12	4	14	6
3	11	1	9
15	7	13	5

Dither-Matrix: Konstruktion

$$D_0 = (0)$$

U_n = $n \times n$ - Matrix, besetzt mit 1

$$D_n = \begin{pmatrix} 4 \cdot D_{n-1} + 0 \cdot U_{n-1} & 4 \cdot D_{n-1} + 2 \cdot U_{n-1} \\ 4 \cdot D_{n-1} + 3 \cdot U_{n-1} & 4 \cdot D_{n-1} + 1 \cdot U_{n-1} \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix} \qquad D_2 = \begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

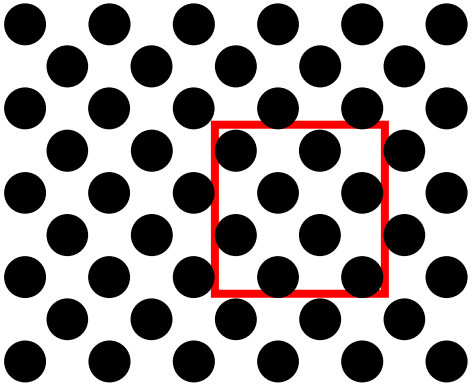
$$\begin{pmatrix} 0 & 8 & 0 & 8 \\ 12 & 4 & 12 & 4 \\ 0 & 8 & 0 & 8 \\ 12 & 4 & 12 & 4 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 \end{pmatrix}$$

Dither-Matrix: Aufruf

Gegeben N × N Dithermatrix D.

Einfärben an Position (x,y) mit Grauwert k:

```
if (D[x%N][y%N] < k)
    setPixel(x,y);
else
    delPixel(x,y);
```

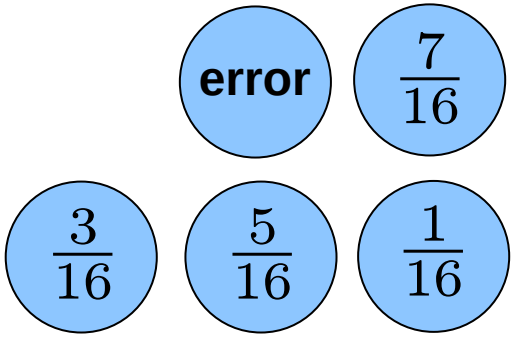


Dither-Matrix-Implementation

Java-Applet:

~cg/2010/skript/Applets/2D-basic/App.html

Floyd-Steinberg-Dithering



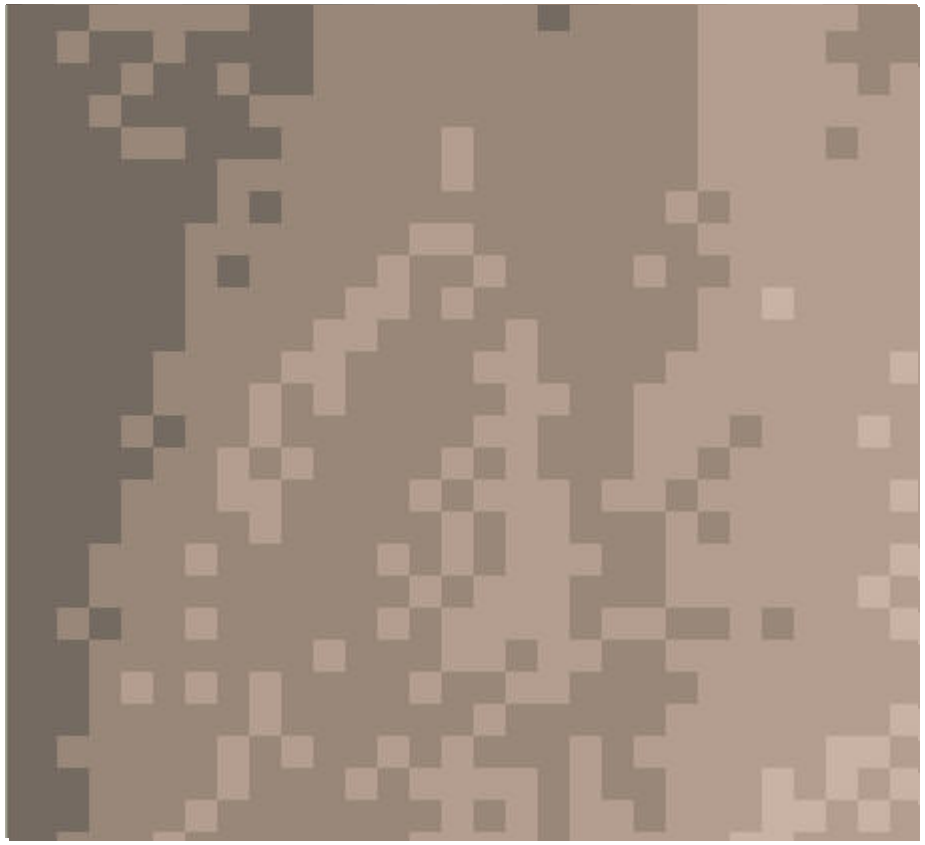

```
for (y=ymax; y>0; y--) {
  for (x=0; x<xmax; x++){
    altfarbe         = pixel[x][y];
    neufarbe         = farbtabelle(altfarbe);
    pixel[x][y]      = neufarbe;
    error            = altfarbe - neufarbe;
    pixel[x+1][y  ] = pixel[x+1][y]   + 7/16 * error;
    pixel[x-1][y-1] = pixel[x-1][y-1] + 3/16 * error;
    pixel[x  ][y-1] = pixel[x  ][y-1] + 5/16 * error;
    pixel[x+1][y-1] = pixel[x+1][y-1] + 1/16 * error;
  }
}
```

Dither-Effekte

True Color

Tabelle
ohne Dither

Tabelle
mit Dither



Punkt in Polygon