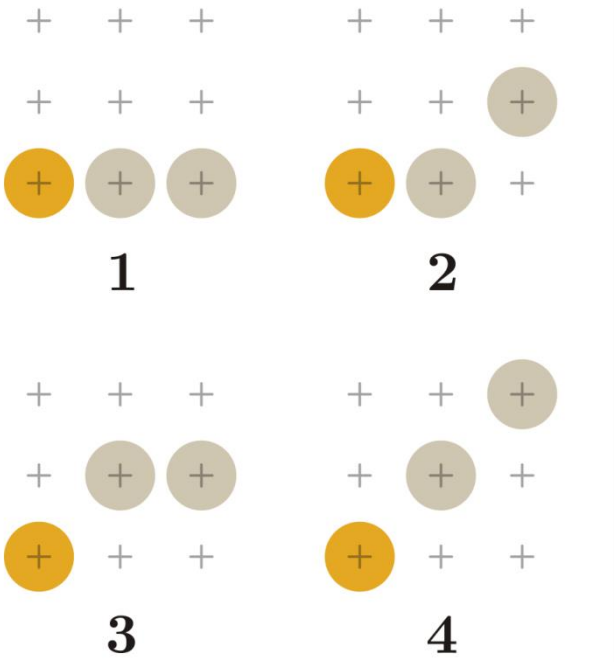
N-Schritt-Verfahren

- Grundidee: Schritte von mehreren Pixeln in x-Richtung machen, alle dazwischen liegende Punkte werden auf einmal eingefärbt

- Doppelschritt-Verfahren: Fallunterscheidung in der letzten Pixelspalte
  - Punkt oben: Muster 4
  - Punkt unten: Muster 1
  - Punkt in der Mitte: Muster 2 oder 3
    - Fallunterscheidung in der vorletzten Spalte




```
// init
dx = x1-x0;
dy = y1-y0;
incr1= 4*dy;
incr2= 4*dy -2*dx;
cond = 2*dy
d = 4*dy - dx;
x = x0;
y = y0;
```

```
while(x < x1) {
   if(d < 0) {           // Pattern 1
      drawPixels(pattern1,x);
      d += incr1;
   } else {
      if(d < cond) {    // Pattern 2
         drawPixels(pattern2,x);
      } else            // Pattern 3
         drawPixels(pattern3,x);
      }
      d += incr2;
   }
   x += 2;
}
```

https://de.wikipedia.org/wiki/Rasterung_von_

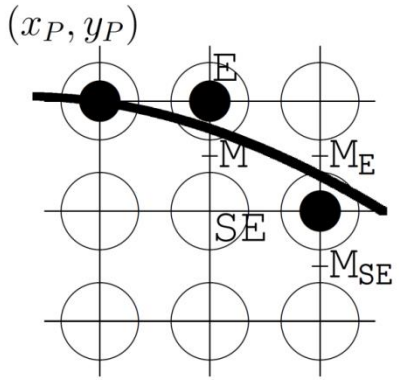
Midpoint Circle Algorithmus

- Konvention:
  - Start bei 12 Uhr
  - Zeichnen bis 1.30 Uhr
- Entsprechend wird die Entscheidungsvariable berechnet

$$F(M) = F\left(x_p + 1, y_p - \tfrac{1}{2}\right) = d$$

$$d = (x_p + 1)^2 + (y_p - \tfrac{1}{2})^2 - R^2$$

- Vorzeichen von d entscheidet über nächsten Punkt auf der Linie:
  - $d \geq 0$: wähle SE
  - $d < 0$: wähle E
- Neue Entscheidungsvariable kann iterativ aus vorheriger Entscheidungsvariable berechnet werden → Inkremente



S. Krömker: Computergrafik I, Universität Heidelberg

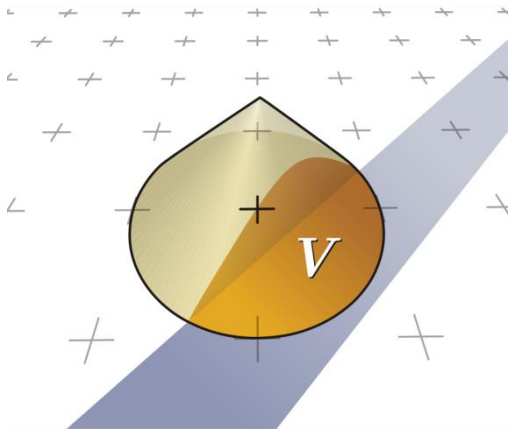
Midpoint Circle Algorithmus

- Vergleich Midpoint Line / Midpoint Circle Algorithmus

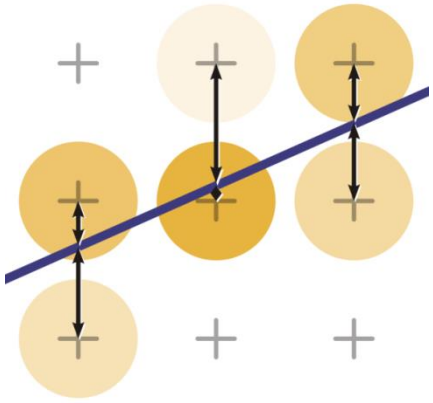
	Midpoint Line	Midpoint Circle
Offset	E: Δy NE: $(\Delta y - \Delta x)$	E: $2x_p + 3$ SE: $2x_p - 2y_p + 5$
Inkremente	konstant	Lineare Funktion
Operationen	Integer	Floating point

Anti-Aliasing von Linien

- Berechnung der Flächenüberdeckung für alle Pixel.
  - Intensität des Pixels = lineare Funktion der am Objekt beteiligten Fläche
- Gupta-Sproull Methode: 3D Kegel als Glättungskern
  - Berechnung des Volumenanteils der die Linie überdeckt

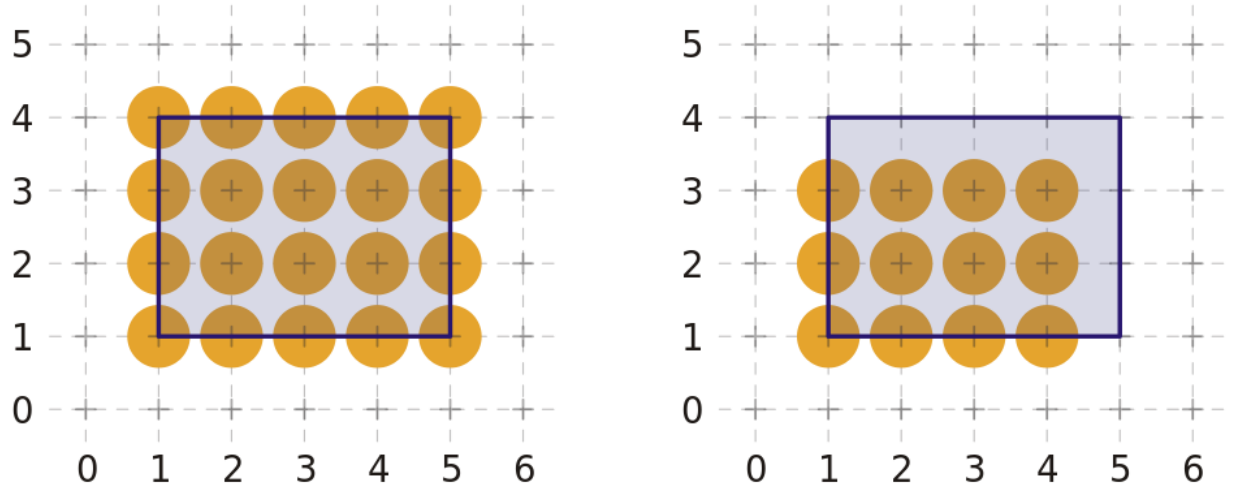


- Wu-Methode
  - Bresenham mit beliebiger Pixelintensität
  - Pixel erhalten Farbwert proportional zur vertikalen Distanz zur idealen Linie.



Füllen von Rechtecken

- Färbung der Pixel zwischen (x_{min}, y_{min}) und (x_{max}, y_{max}).
- Interpretation der Koordinaten:



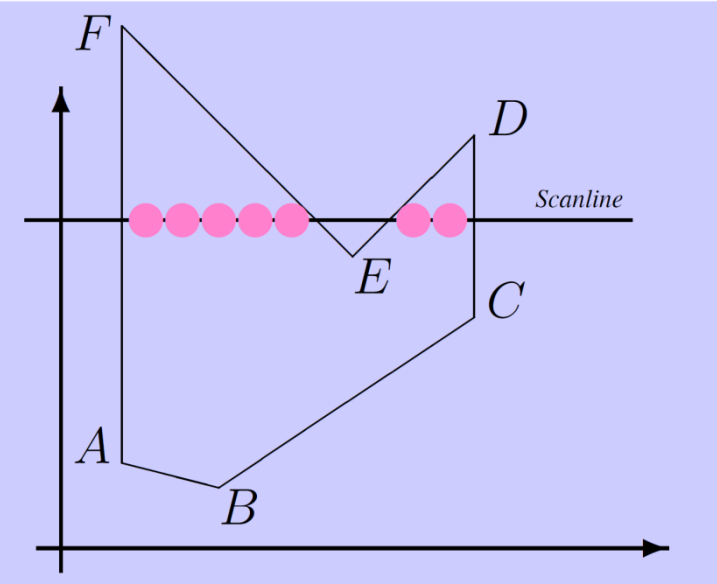
Rechteck größer als definiert

- Abhilfe:
  - Rasterung um einen halben Pixelabstand nach links und nach unten zu verschieben.
  - Rechteck rechts mit den Koordinaten $(0{,}5, 0{,}5)$ und $(4{,}5, 3{,}5)$ wird korrekt gezeichnet.

https://de.wikipedia.org/wiki/Rasterung_von_Polygonen

Füllen von Polygonen

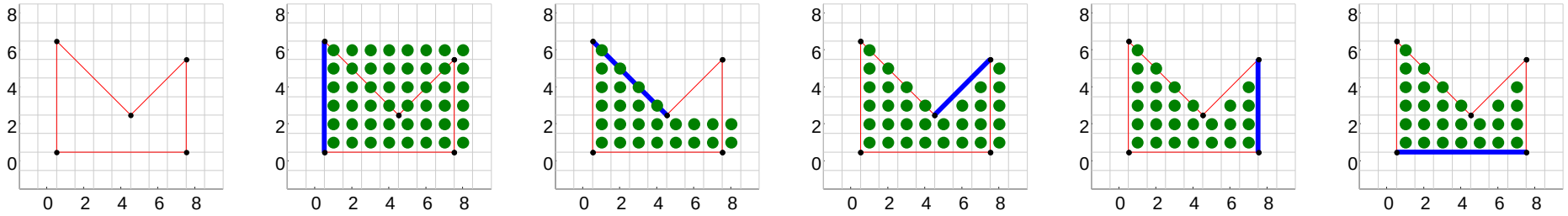
- Beliebige Polygone: Füllung durch Scanline → Schnittpunktberechnung, Füllung zwischen Schtnittpunkten
- Modifikation:
  1. Für jeden Schnittpunkt einer Polygonkante mit einer Bildzeile: Einfärben des ersten Pixels mit $x > s_x + 0{,}5$.
  2. Füllen des Polygons durch Negation von innerhalb/außerhalb



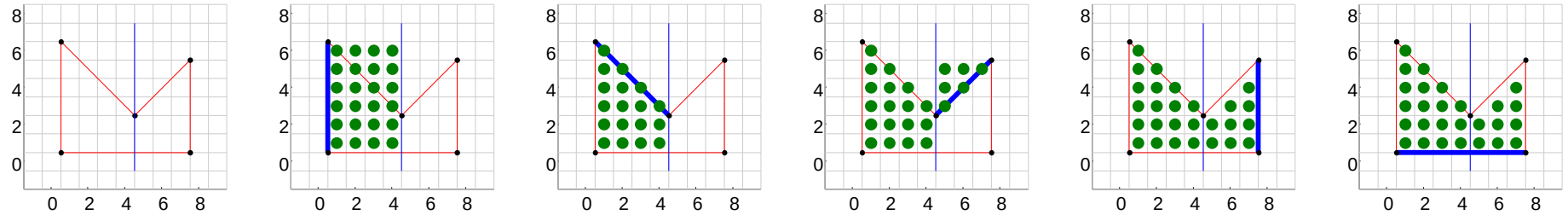

```
Für jede Bildzeile y, die das Polygon schneidet
  Innerhalb = Falsch
  Für jedes x von links bis rechts
    Wenn Pixel (x, y) eingefärbt ist    // Paritätsregel
      Innerhalb negieren
    Wenn Innerhalb
      Pixel (x, y) einfärben
    ansonsten
      Pixel (x, y) auf Hintergrundfarbe zurücksetzen
```

Füllen von Polygonen

- Edge-Fill-Algorithmus

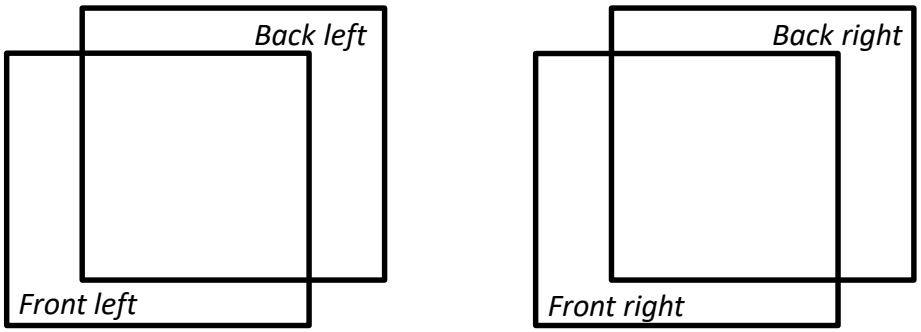


- Erweiterung zum Fence-Fill-Algorithmus

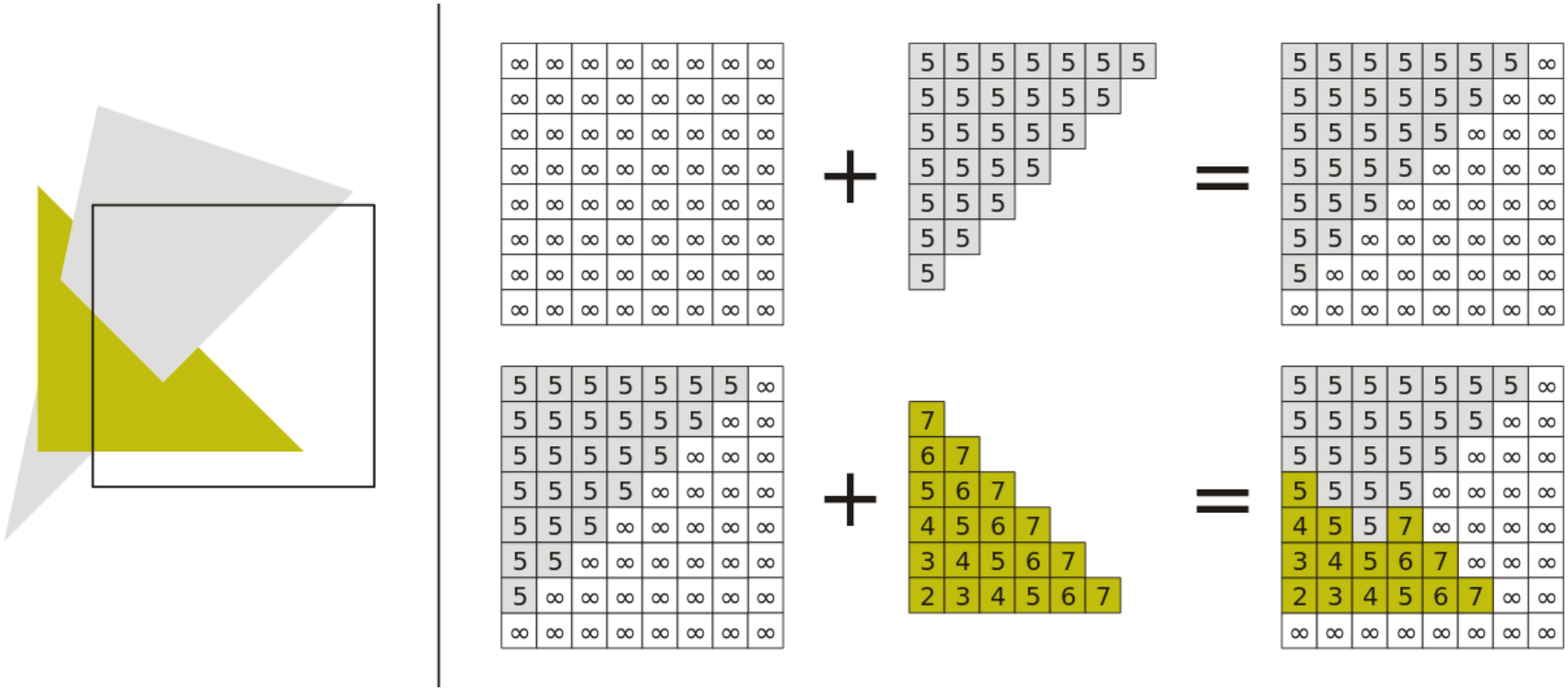


Double / Stereo Buffering

- Double Buffering wirkt „Flackern“ entgegen:
  - Unterteilung des Framebuffers in Front-Buffer und Back-Buffer
  - Front-Buffer wird ausgelesen und auf dem Bildschirm dargestellt
  - Back-Buffer wird zeitgleich mit neu berechnetem Bild gefüllt
  - Anschließend Austausch von Front- und Backbuffer (*Swap*)
- Zeitpunkt des Tauschs beachten: vgl. VSYNC
- Bei Stereo-Darstellungen werden Bilder für das linke und rechte Auge aus unterschiedlicher Kameraposition erzeugt.



Z-Buffer-Algorithmus



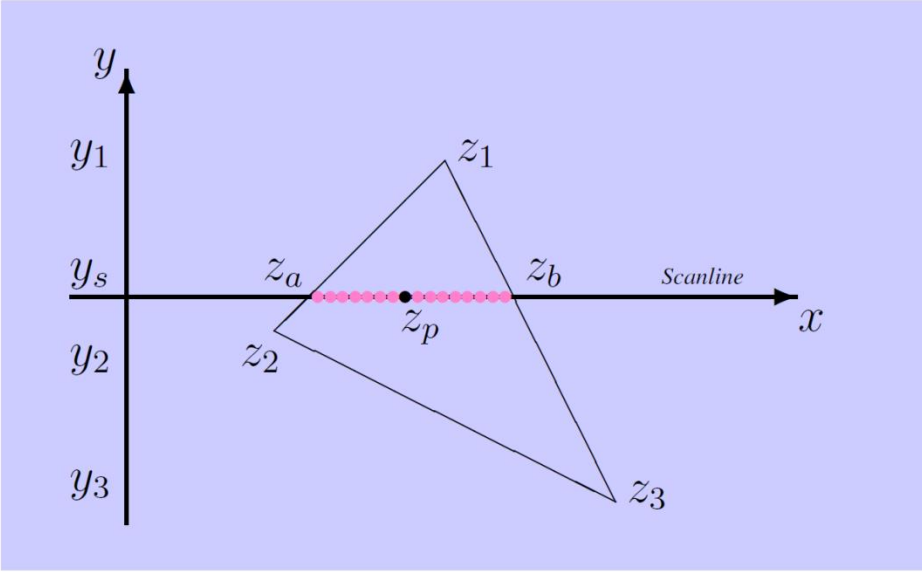
https://de.wikipedia.org/wiki/Z-Buffer

Z-Werte für alle Pixel eines Polygons

- Lineare Interpolation der z-Werte entlang der Scanline:

$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$
$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{x_b - x_a}$$



S. Krömker: Computergrafik I, Universität Heidelberg

Hidden-Line-Darstellung

- Z-Buffer für Verdeckungsberechnung → Stitching
- Polygonoffset: **`glEnable(GL_POLYGON_OFFSET_FILL); / glEnable(GL_POLYGON_OFFSET_LINE);`**
- Für Hidden-Line-Darstellung zwei Möglichkeiten:
  - Offset für gefülltes Polygon >0 → Polygon erscheint weiter hinten
  - Offset für Linie <0 → Linie erscheint weiter vorne