Rastergrafikalgorithmen

Vortrag über Rastergrafikalgorithmen im Rahmen des Proseminars Computergrafik

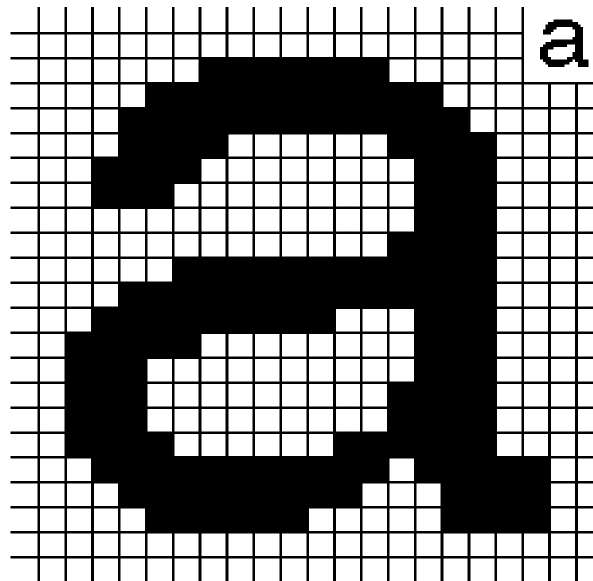
Vortragender: Christian Vonsien

Inhalt

1. **Einleitung** – Was ist eine Rastergrafik?
2. **Rasterung von Linien und Kurven**
3. **Füllen beschränkter Flächen**
4. **Clipping von Linien und Polygonen**

1. Einleitung – Was ist eine Rastergrafik?

- Matrix aus gleichgroßen, regelmäßig angeordneten Pixeln
- Eigenschaften: Zahl der Pixel und Größe des Bildes in Höhe und Breite
- Pixel besitzt Bitebenen (Kanal) → ermöglicht verschiedene Zustände



2. Rasterung von Linien und Kurven

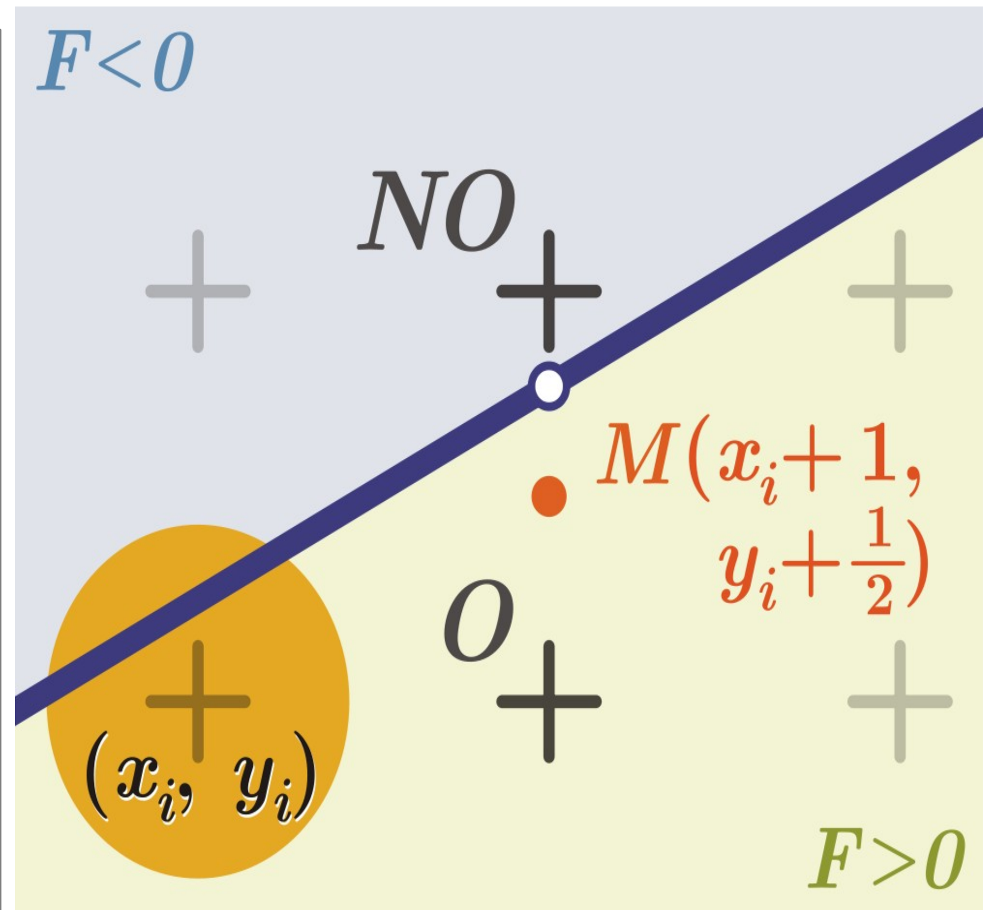
- Problem: Pixelpunkte haben nichtganzzahlige x,y-Werte
- Effekte: Ungenauigkeiten, wie Treppcheneffekt
- Ziel: Originaldarstellung (nahezu) beibehalten

2.1. Rasterung von Linien

- naiver Algorithmus: aus Geradengleichung die y-Werte berechnen und runden
- Nachteil: langsam, da Floatingpoint- und Multiplikations-Operationen
- Multiplikation durch Addition ersetzen
- Steigung am Anfang berechnen → in jedem Schritt auf y addieren
- Rundung durch Kontrollvariable *error* umgehen

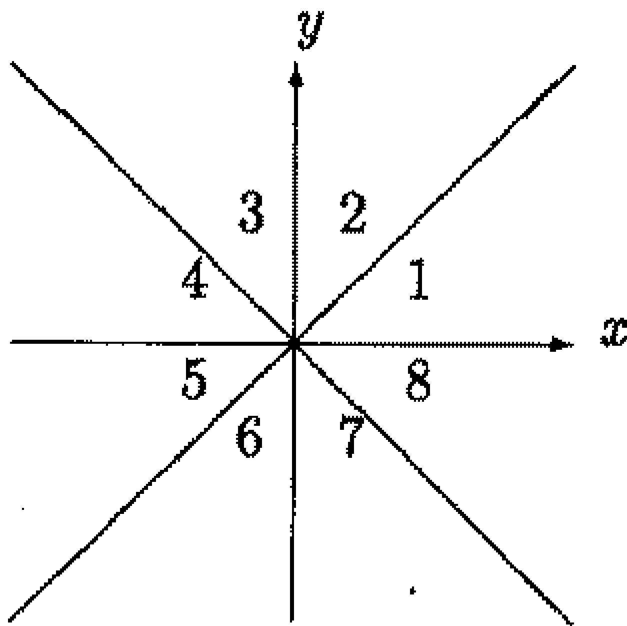
Bresenham-Algorithmus zur Rasterung von Linien

```
error = 2*Δy – Δx
ΔO = 2*Δy
ΔNO = 2*(Δy − Δx)
y = y_a
Pixel (x_a, y_a) einfärben
FOR (x = x + 1; x ≤ x_e)
        IF (error ≤ 0)
        error = error + ΔO
        ELSE
        error = error + ΔNO
        y = y + 1
        Pixel (x, y) einfärben
```



- Floatingpoint-Operationen werden zusätzlich vermieden

- bisher nur gültig bei Steigungen zwischen 0 und 1
- Lösung: Ausnutzung von Symmetrien



Oktand	treibende Achse	andere Achse
1	x	inkrementiert
2	y	inkrementiert
3	y	dekrementiert
4	x	dekrementiert
5	x	inkrementiert
6	y	inkrementiert
7	y	dekrementiert
8	x	dekrementiert

2.2 Algorithmus zur Rasterung von Kreisen

- Ansatz vom Bresenham-Algorithmus abgeleitet

$$r^2 = x^2 + y^2 \Rightarrow x^2 = (x_{vorher} - 1)^2 = x_{vorher}^2 - 2\mathrm{x}_{vorher} + 1$$

$$bzw.\ \ y^2 = (y_{vorher} + 1)^2 = y_{vorher}^2 + 2\mathrm{y}_{vorher} + 1$$

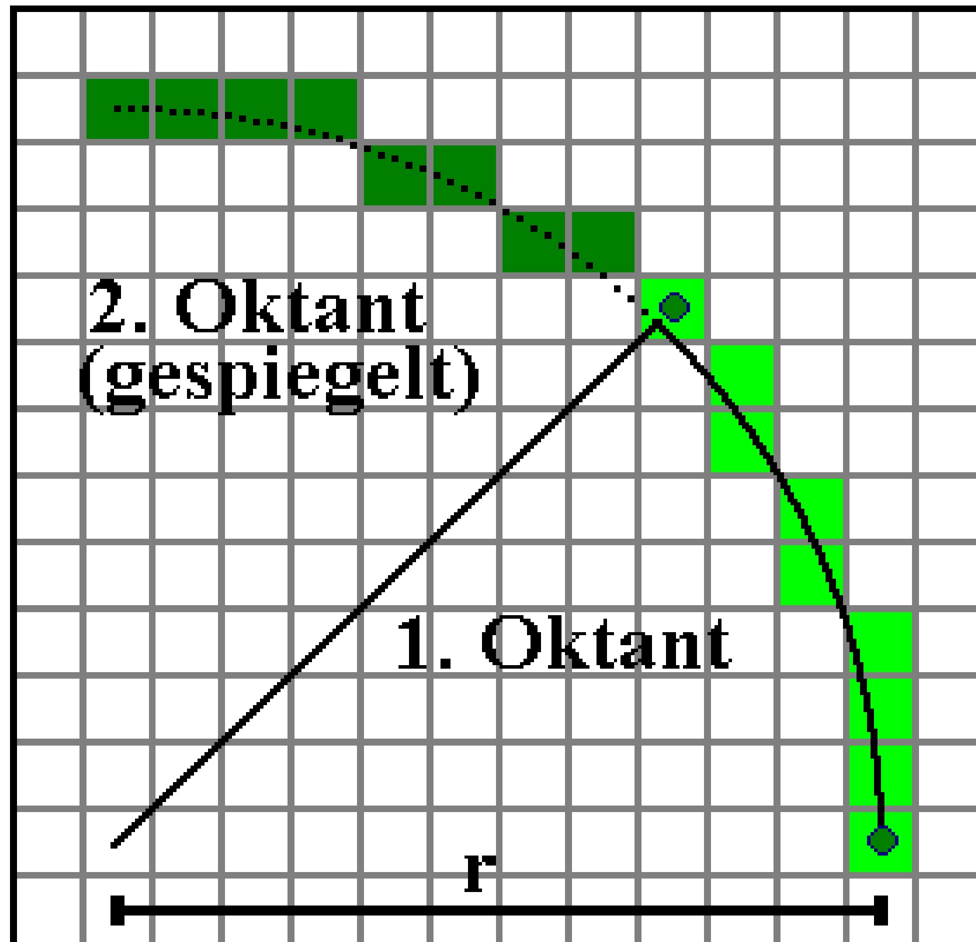
- x^2 wird nicht als Variable genutzt, Differenz fließt beim Fehlerglied ein

- Fehlerglied wird mit r initialisiert

$$y = \sqrt{r^2 - x^2}\ am\ Punkt\ (x_1, y_1)\ soll\ gelten: x_1 = r - 1/2$$

$$\Rightarrow y_1 = \sqrt{r - 1/4}$$

- Berechneter x- bzw. y-Wert wird auf Mittelpunkt aufaddiert und eingefärbt
- Für restliche Oktanden wieder Symmetrie ausnutzen



Bresenham-Algorithmus zu Rasterung von Kreisen

```
x = r
y = 0
error = r
Pixel ( x_mittel + x, y_mittel + y) einfärben
WHILE (y < x)
    dy = y*2 + 1
    y = y + 1
    error = error - dy
    IF error < 0
            dx = 1 - x*2
            x = x - 1
            error = error - dx
    Pixel (x_mittel + x, y_mittel + y) einfärben
    Pixel (x_mittel + y, y_mittel + x) einfärben
```

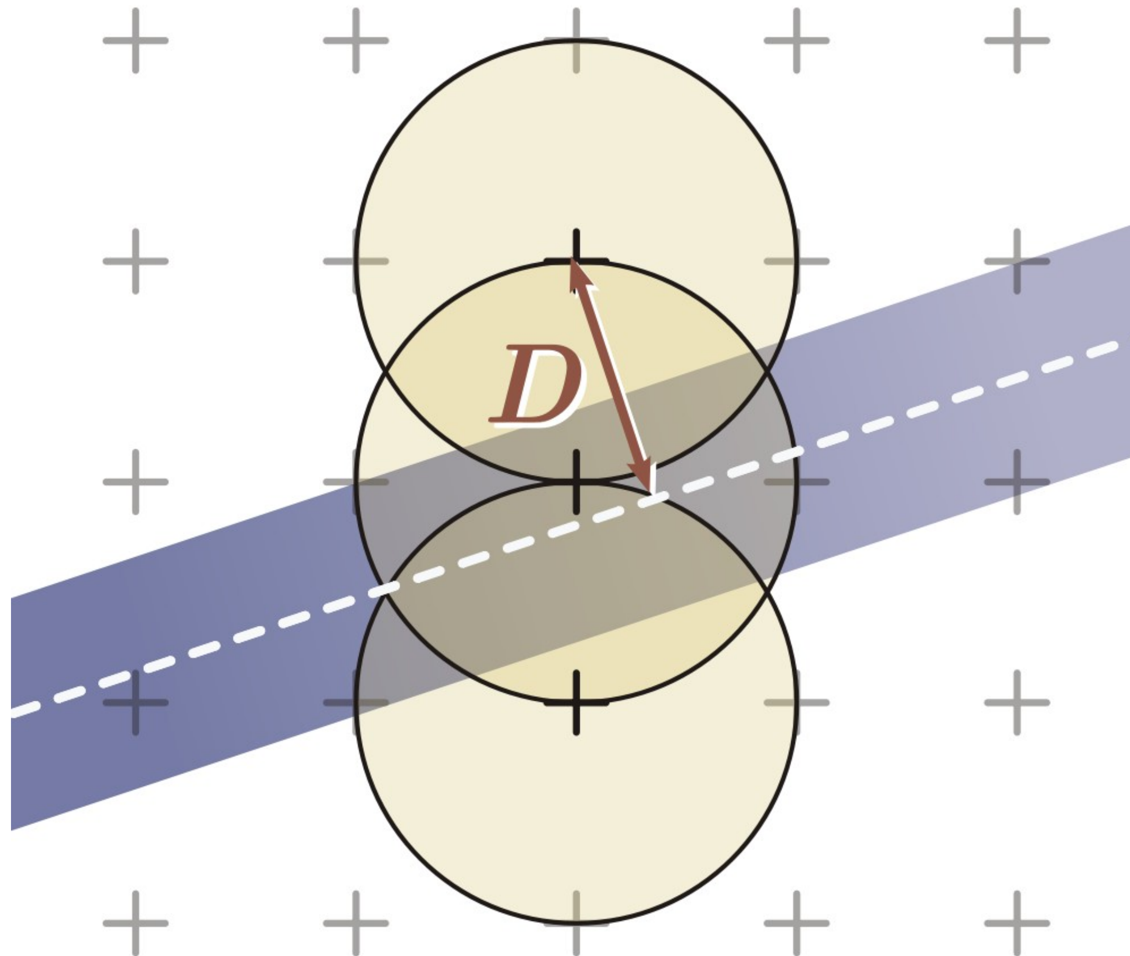
2.3. Antialiasing

- Problem: Linien, Kreise sehen treppenartig aus
- Verbesserung: hellere Pixel um die Linien legen
- Linie wird nicht mehr eindimensional, sondern zweidimensional betrachtet
- 2 Algorithmen kurz beschrieben

2.3.1. Methode von Gupta und Sproull

- Basiert auf dem Bresenham-Algorithmus
- Kegel mit Radius von 1 Pixel als Glättungskern
- Zusätzliche Berechnung von D (Distanz von der Originallinie) in jedem Schritt
- Bei Linien aus 1 Pixel: maximal 3 Pixel, die überlappen
- Helligkeitswerte werden aus Distanztabelle ausgelesen (nur 24 Distanzen)
- Helligkeit der Pixel an Linienenden abhängig von Steigung (→ Tabelle)

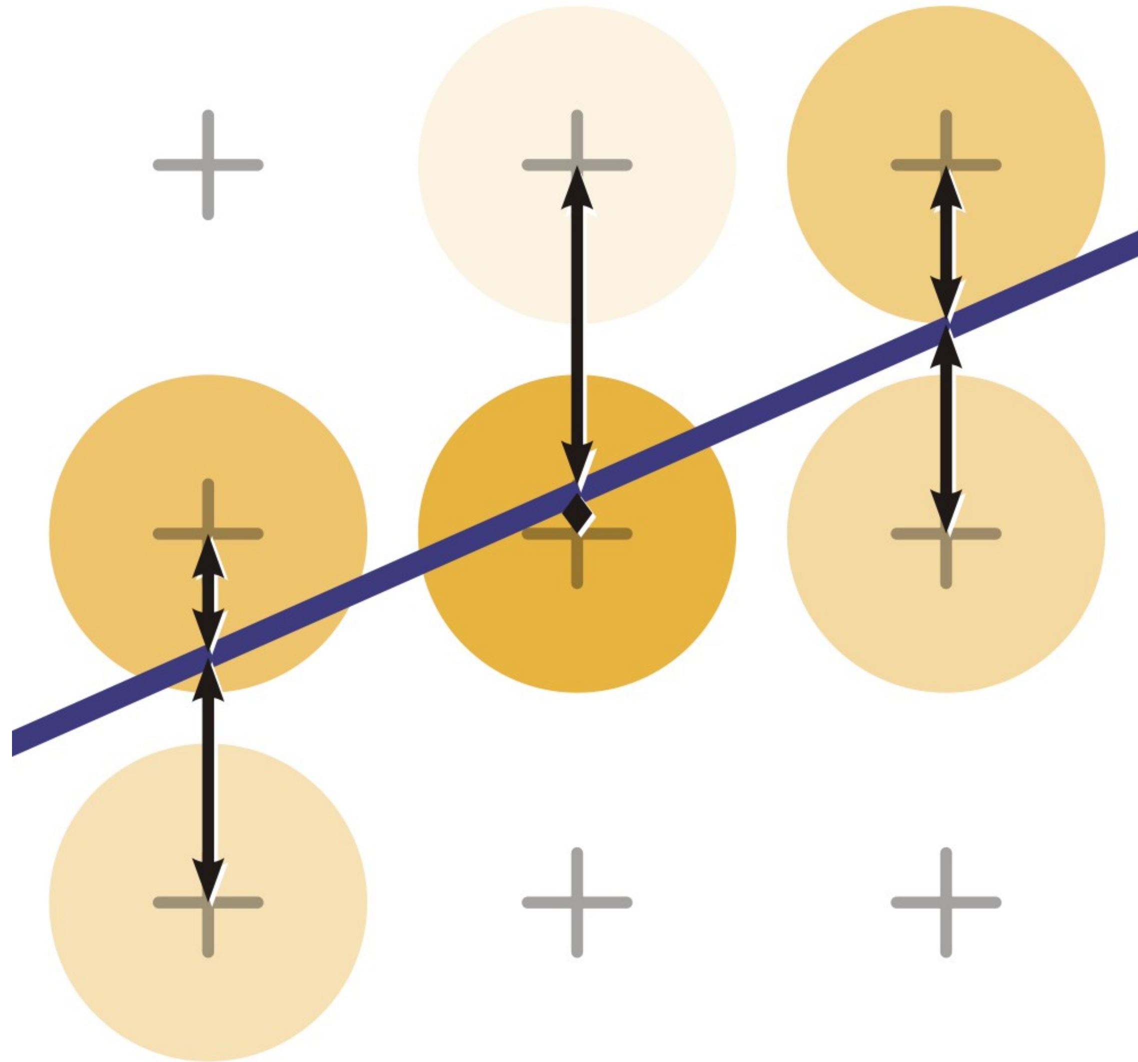
Gupta und Sproull: Glättungskern: Kegel



2.3.2. Methode von Wu

- Basiert nicht auf Glättungskern, sondern auf Fehlermaß
- Fehler lässt sich beseitigen, wenn beliebige Farbwerte zugelassen sind
- In Grundform nur auf unendlich dünne Linien anwendbar
- Bresenham: Fehler zwischen Pixel und Originallinie wird minimiert
- Wu: Pixel ober-, unterhalb der Linie: Helligkeitswerte proportional zur Distanz

Wu: Intensitäten



3. Füllen beschränkter Flächen

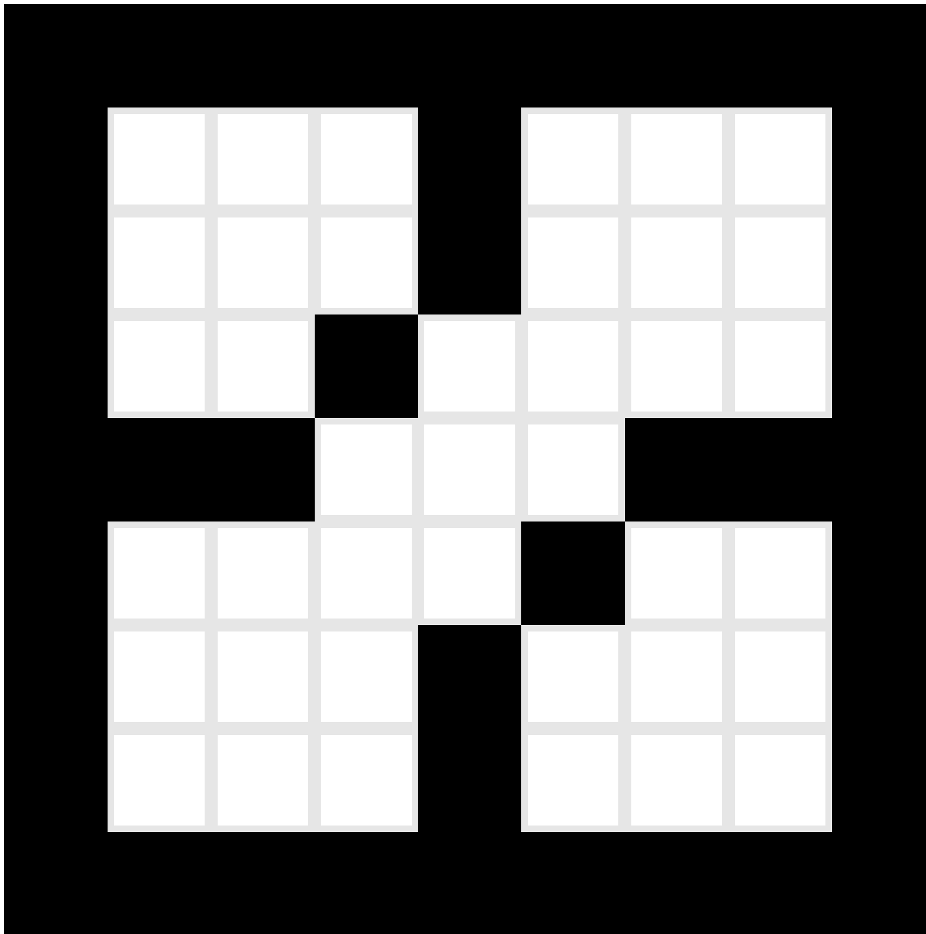
3.1. Scangeraden-Methode

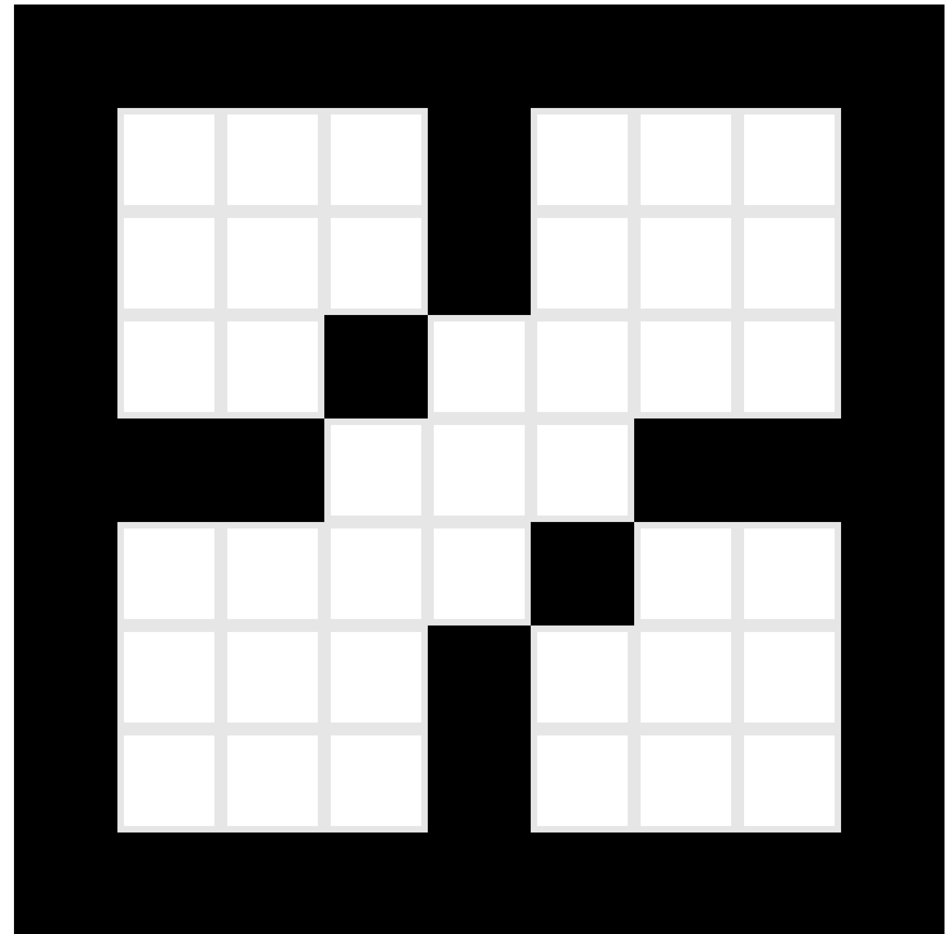
- Voraussetzung: Polygon geometrisch beschrieben
- Gerade parallel zur x-Achse fährt über Objekte
- Für jede Lage werden Schnittpunkte von Gerade und Objekt berechnet
- Pixel zwischen 2 aufeinander folgenden Paaren von x-Werten werden gefärbt
- konkav: 2 oder keine Schnittpunkte, konvex: gerade Anzahl an Schnittpunkten
- Polygone über den Bildschirmrand hinaus → Rand ist zusätzliche Kante

3.2. Saatfüllen

- Voraussetzung: Begrenzung durch andere Bildschirmobjekte
- Pixelfarbe wird als Grenze genutzt, dazwischen wird aufgefüllt
- Beginnt mit einem Pixel, färbt Nachbarn ein, wenn es nicht die Grenzfarbe hat
- Diagonalpixel sind keine Nachbarpixel, vgl. Bresenham-Algorithmus
- Kann bei engen Polygonen stoppen → zweite Saat setzen

Saatfüll-Algorithmus mit Bezug zu Bresenham





Saatfüll-Implementation

```
void floodfill(int x, int y, int borderColour, int newColour) {

    if (getPixel(x, y) != borderColour){

                colourPixel(x, y, newColour);

                floodfill(x, y + 1, borderColour, newColour); // unten
                floodfill(x - 1, y, borderColour, newColour); // links
                floodfill(x, y - 1, borderColour, newColour); // oben
                floodfill(x + 1, y, borderColour, newColour); // rechts

    }
}
```

4. Clipping von Linien und Polygonen

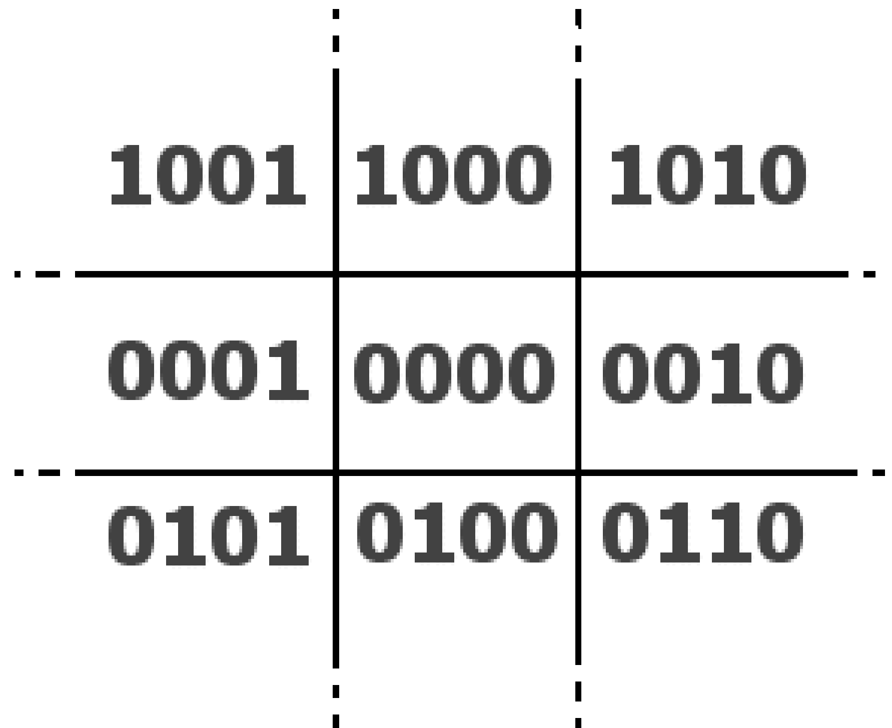
- Oft angewandte Technik, meist um Geraden an Polygonen abzuschneiden
- 2 Varianten: Clipping während der Darstellung oder analytisches Clipping
- Clipping während der Darstellung: jedes Pixel überprüfen
- Analytisches Clipping: Schnittpunkte berechnen, dann clippen

4.1. Clipping von Linien an rechteckigen Clip-Bereichen

- Zur Begrenzung von Linien innerhalb eines Bildschirmbereiches benutzt
- Sind beide Endpunkte innerhalb, ist auch die Linie innerhalb
- Mindestens ein Eckpunkt außerhalb → Schnittpunkte berechnen
- Vorgehen: Parametergleichung → Schnittpunkte mit dem Polygon bestimmen

Algorithmus von Cohen und Sutherland

- Problem: Gerade kann durch das Polygon verlaufen, aber auch daran vorbei
- Anzahl an Schnittpunktberechnungen verringern
- Kennzeichnung von 9 Bereichen in Bezug auf das Clipping-Polygon



- Bitmuster von benachbarten Bereichen unterscheiden sich um 1 Bitstelle
- Endpunkten Bitcodes der Bereiche zuordnen
- Bitweise Oder-Verknüpfung = 0 → Gerade liegt innerhalb des Fensters
- Bitweise Und-Verknüpfung ≠ 0 → Gerade geht am Fenster vorbei
- Ansonsten: Gerade an Fenstergeraden schneiden, Teil außerhalb löschen
- Fortführen, bis alle Geraden auf einen der 2 Fälle zurückgeführt wurden

Implementation von Cohen und Sutherland

```
void clip_CS (float x1, float y1, float x2, float y2, float xmin, float xmax,
    float ymin, float ymax){
int c1, c2;
float xs, ys;
c1 = code(x1, y1);
c2 = code(x2, y2);
if (c1 | c2 == 0x0)
    Draw_Line(x1, y1, x2, y2);
elseif (c1 & c2 != 0x0)
    return;
else
    intersect(xs, ys, x1, y1, x2, y2, xmin, xmax, ymin, ymax);
    if (is_outside(x1, y1))
        clip_CS(xs, ys, x2, y2, xmin, xmax, ymin, ymax);
    else
        clip_CS(x1, y1, xs, ys, xmin, xmax, ymin, ymax);
}
```

4.2. Clipping von Polygonen an rechteckigen Clip-Bereichen

- Algorithmus von Sutherland und Hodgman
- Rückführen auf einfaches Problem → Polygon an unendlicher Geraden clippen
- Algorithmus gibt Schnittpunkte von unendlicher Geraden und Polygon zurück
- Wird für jede Fenstergrenze durchgeführt
- Menge an Schnittpunkten beschreibt das neue Polygon
- Aufwendiger als Cohen und Sutherland, kann aber jedes Polygon clippen

5. Quellen

- Piesch, Dieter: Skript Rastergrafik, Uni Regensburg, 2002
- Rauber, Thomas: „Algorithmen in der Computergraphik",1993
- Fellner, Dieter W.: „Computergrafik", 1992
- Vandevenne, Lode: „Computer Graphics Tutorial", 2004
- Wikpedia-User: Phrood, PeterFrankfurt und André Karwath