

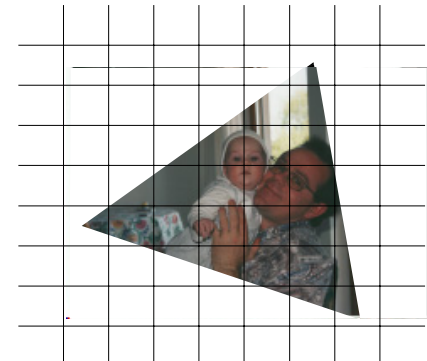
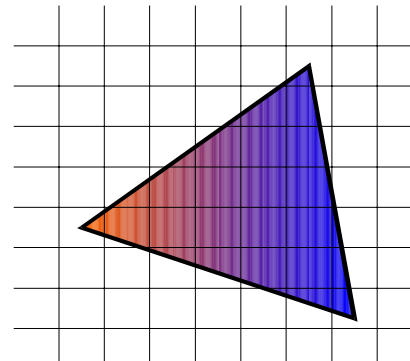
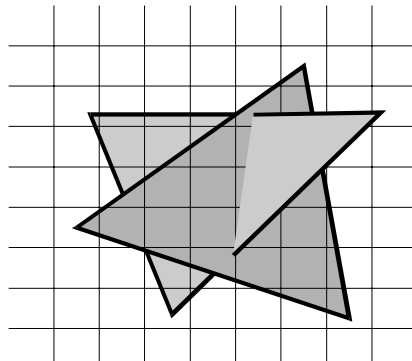
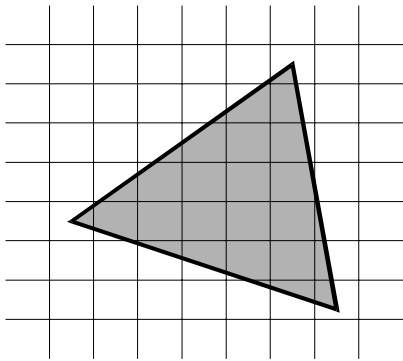
Computer Graphik I

Rasterisierung (Sichtbarkeit, Beleuchtung)

- Anwendung
- Geometrieverarbeitung
 - Perspektivische Transformation, kanonisches Sichtvolumen
 - Clipping
 - Culling (Verdeckungsrechnung im Objektraum)
 - Simulation der Beleuchtung
 - Projektion
- Rasterisierung
- Ausgabe

■ Rasterisierung

- Primitive (Linien, Polygone) werden in Pixel zerlegt



■ Zusätzliche Operationen pro Pixel

- Verdeckungsrechnung (inclusive Transparenz)
- Shading und
- Texturierung

- Betrachte Linien, deren Anfangs- und Endpunkt auf dem Raster liegen.

$$\Delta x = x_2 - x_1, \quad -1 \leq \frac{\Delta x}{\Delta y} \leq 1$$

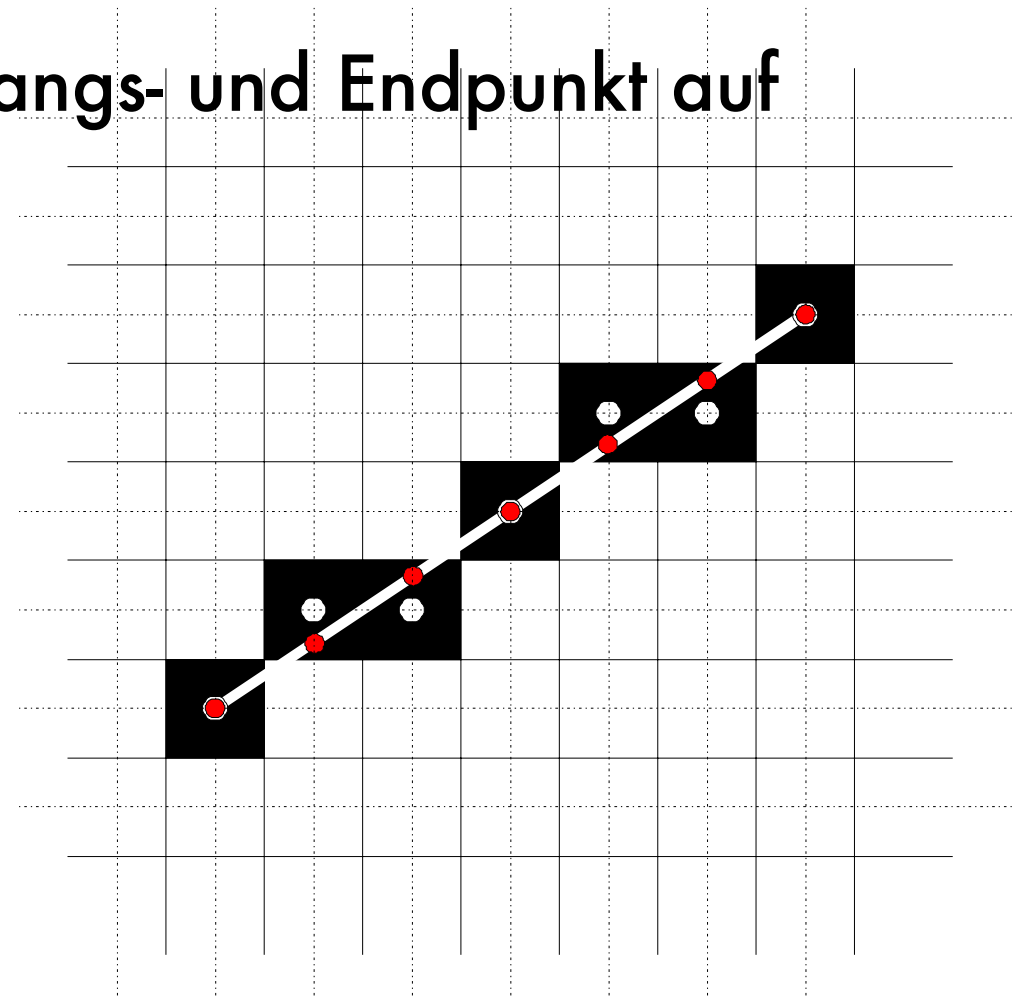
$$\Delta y = y_2 - y_1,$$

- Berechne

$$y_i = \frac{\Delta y}{\Delta x} * x_i + b,$$

$$x_i = x_1 + i, \quad i = 1, \dots, \Delta x$$

- Zeichne Pixel an der Stelle $(x, \text{round}(y))$



- Ineffizient: jede Pixeloperation erfordert
 - floating point Multiplikation,
 - Addition,
 - Rundung

- Idee: Inkrementeller Algorithmus

$$\begin{aligned}
 y_{i+1} &= \frac{\Delta y}{\Delta x} * x_{i+1} + b \\
 &= \frac{\Delta y}{\Delta x} (x_i + (x_{i+1} - x_i)) + b
 \end{aligned}$$

- Mit $x_{i+1} - x_i = 1$ ergibt sich

$$\begin{aligned}
 &= y_i + \frac{\Delta y}{\Delta x} (x_{i+1} - x_i) \\
 y_{i+1} &= y_i + \frac{\Delta y}{\Delta x}
 \end{aligned}$$

$$\Delta x = x_2 - x_1; \Delta y = y_2 - y_1;$$

$$m = \frac{\Delta y}{\Delta x}$$

$$x = x_1; y = x_2;$$

for($x = x_1, x \leq x_2, x++$) {

 Zeichne Pixel (x, y);

$y = \text{round}(y + m);$ }

Rasterisierung von Linien

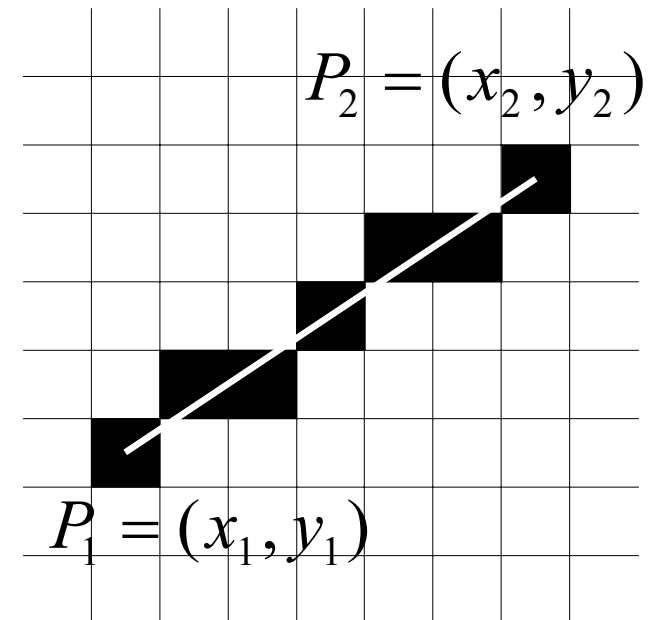
Algorithmus von Bresenham

- Der erste Integer-Algorithmus zum Zeichnen von Linien
 - Bresenham (1965).
- Herleitung:
 - Anfangs- und Endpunkt auf dem Raster
 - Steigung zwischen 0 und 1

$$\Delta x = x_2 - x_1 \geq 0,$$

$$\Delta y = y_2 - y_1 \geq 0,$$

$$\Delta x \geq \Delta y.$$



Rasterisierung von Linien

Algorithmus von Bresenham

- Welcher Pixelmittelpunkt liegt näher zur Geraden? Ist $d \leq 1/2$ oder ist $d > 1/2$?

- Entscheidungsvariable $E := \frac{\Delta y}{\Delta x} - \frac{1}{2}$

$$E' := 2\Delta x E = 2\Delta y - \Delta x$$

$$E \leq 0:$$

$$x := x + 1$$

$$E := E + \frac{\Delta y}{\Delta x}$$

$$E := E + 2\Delta y$$

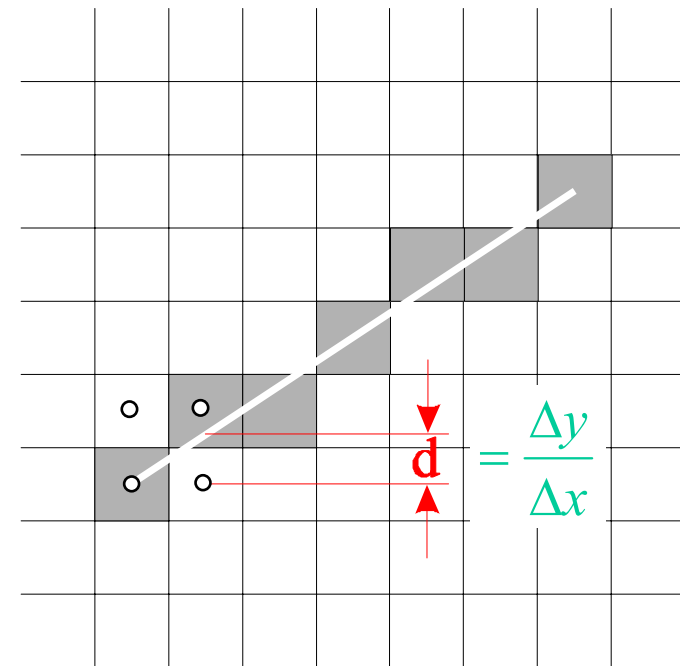
$$E > 0:$$

$$x := x + 1$$

$$y := y + 1$$

$$E := E + \frac{\Delta y}{\Delta x} - 1$$

$$E := E + 2\Delta y - 2\Delta x$$



Rasterisierung von Linien

Mittelpunktsalgorithmus

- Voraussetzungen wie bei der Herleitung des Bresenham-Algorithmus
- Welches Pixel soll als nächstes gewählt werden:

$(x + 1, y)$ oder $(x + 1, y + 1)$?

- Auf welcher Seite der Geraden liegt

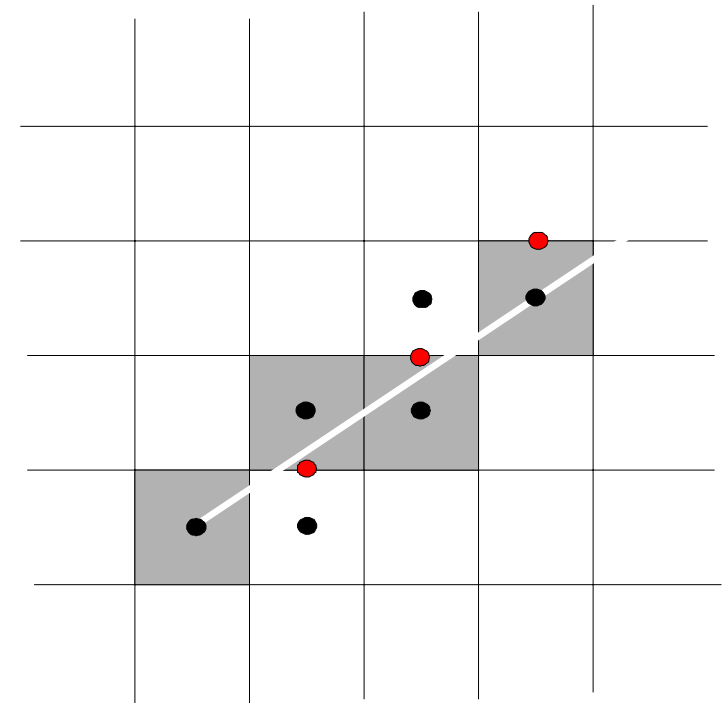
$$M = (x + 1, y + \frac{1}{2})$$

- Implizite Geradengleichung:

$$g(x, y) = \Delta y * x - \Delta x * y + (\Delta x * y_1 - \Delta y * x_1)$$

$$g(x + 1, y + \frac{1}{2}) \leq 0, \text{ wähle Pixel } (x + 1, y)$$

$$g(x + 1, y + \frac{1}{2}) > 0, \text{ wähle Pixel } (x + 1, y + 1)$$



- Inkrementelle Berechnung von $g(x+1, y + \frac{1}{2})$:

$$g(x+1, y + \frac{1}{2}) = \Delta y * (x+1) - \Delta x * (y + \frac{1}{2}) + C, \quad C = \Delta y * x_1 - \Delta x * y_1$$

- Pixel $(x+1, y)$ gewählt

$$g(x+2, y + \frac{1}{2}) = \Delta y * (x+2) - \Delta x * (y + \frac{1}{2}) + C$$

$$g(x+2, y + \frac{1}{2}) = g(x+1, y + \frac{1}{2}) + \Delta y$$

- Pixel $(x+1, y+1)$ gewählt

$$g(x+2, y + \frac{3}{2}) = \Delta y * (x+2) - \Delta x * (y + \frac{3}{2}) + C$$

$$g(x+2, y + \frac{3}{2}) = g(x+1, y + \frac{1}{2}) + \Delta y - \Delta x$$

Rasterisierung von Linien

Mittelpunktsalgorithmus für Kreise

Beschränkung auf $0 \leq x \leq \frac{R}{\sqrt{2}}$:

Implizite Kreisgleichung:

$$f(x, y) = x^2 + y^2 - R^2.$$

$$f\left(x + 1, y - \frac{1}{2}\right) = (x + 1)^2 + \left(y - \frac{1}{2}\right)^2 - R^2$$

Pixel $(x + 1, y)$ gewählt

$$f\left(x + 2, y - \frac{1}{2}\right) = (x + 2)^2 + \left(y - \frac{1}{2}\right)^2 - R^2$$

$$f\left(x + 2, y - \frac{1}{2}\right) = f\left(x + 1, y - \frac{1}{2}\right) + 2x + 3$$

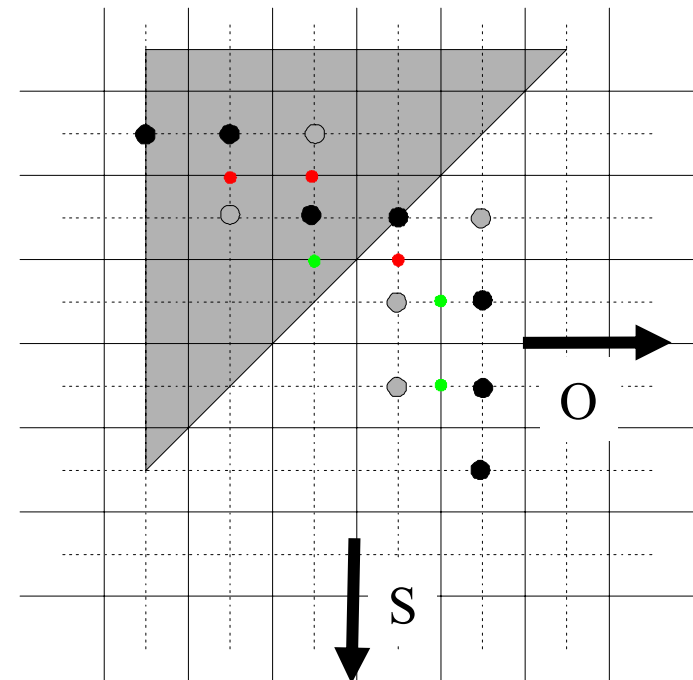
Pixel $(x + 1, y - 1)$ gewählt

$$f\left(x + 2, y - \frac{3}{2}\right) = (x + 2)^2 + \left(y - \frac{3}{2}\right)^2 - R^2$$

$$f\left(x + 2, y - \frac{3}{2}\right) = f\left(x + 1, y - \frac{1}{2}\right) + 2x - 2y + 5$$

M zwischen O und SO

$\left\{ \begin{array}{l} \text{außerhalb: } SO \text{ setzen} \\ \text{sonst: } O \text{ setzen} \end{array} \right.$



Rasterisierung von Linien

Mittelpunktsalgorithmus für Kreise

- Beim Kreis ändern sich die Differenzen in Abhängigkeit von der aktuellen Pixelposition! \Rightarrow Erhöhter Aufwand!

$$f(x+2, y - \frac{1}{2}) = f(x+1, y - \frac{1}{2}) + 2x + 3$$

$$f(x+2, y - \frac{3}{2}) = f(x+1, y - \frac{1}{2}) + 2x - 2y + 5$$

- Idee: Bilde für die Inkremente zweite Differenzen

$$\Delta(x, y) = 2x + 3$$

$$\Delta(x+1, y) = 2(x+1) + 3$$

$$\Delta(x, y) - \Delta_o(x+1, y+1) = 2$$

$$\Delta(x, y) = 2x - 2y + 5$$

$$\Delta(x+1, y) = 2(x+1) - 2y + 5$$

$$\Delta(x, y) - \Delta_{so}(x+1, y+1) = 2$$

Rasterisierung von Linien

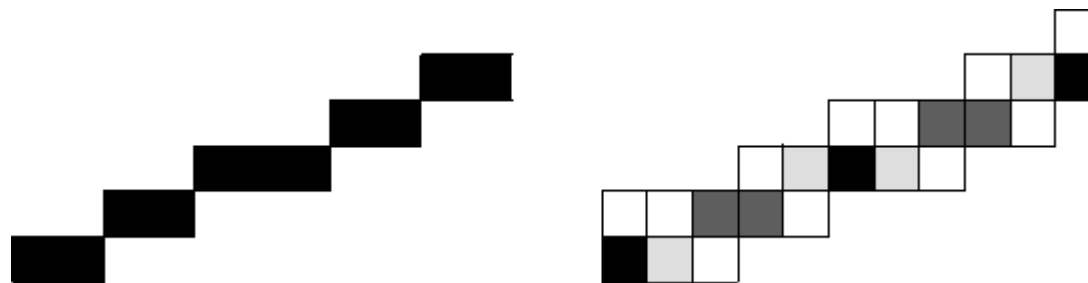
Mittelpunktsalgorithmus für Kreise

```

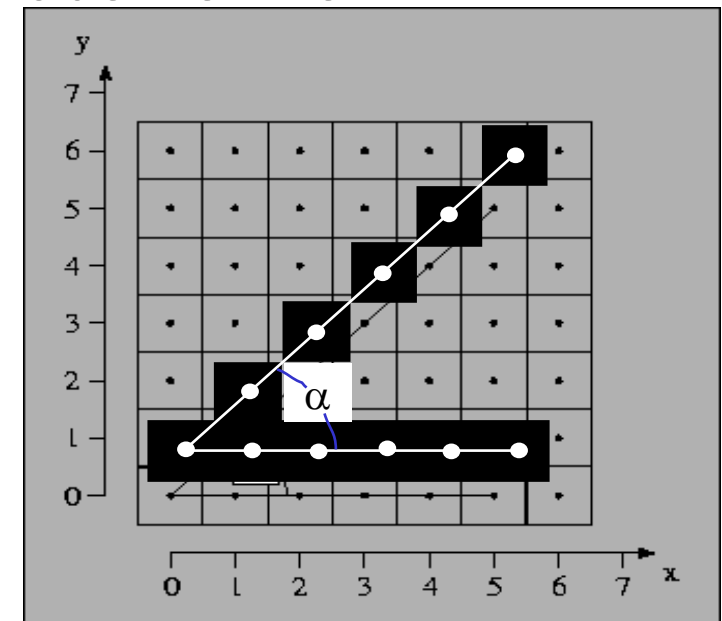
procedure MidpointCircle (radius, value : integer);
var x, y, d, deltaE, deltaSE : integer;
begin
x := 0; y := radius; d := 1 - radius;
deltaE := 3; deltaSE := -2 * radius + 5;
CirclePoints (x, y, value);
while y > x do
  begin
    if d < 0 then                                {Select E}
      begin
        d := d + deltaE;
        deltaE := deltaE + 2;
        deltaSE := deltaSE + 2;
        x := x + 1;
      end
    else                                            {Select SE}
      begin
        d := d + deltaSE;
        deltaE := deltaE + 2;
        deltaSE := deltaSE + 4;
        x := x + 1;    y := y - 1;
      end;
    CirclePoints (x, y, value)
  end; {while}
end; {MidpointCircle}

```

- Für jeden X-Wert (Spalten) werden zwei übereinanderliegende Pixel gesetzt
- Gesamthelligkeit ist in jeder Spalte gleich
- Verteilung entsprechend der vertikalen Entfernungen E der beiden Pixel von der Idealposition
- Pixelhelligkeit (Grauwert) nimmt linear mit steigender Entfernung ab.



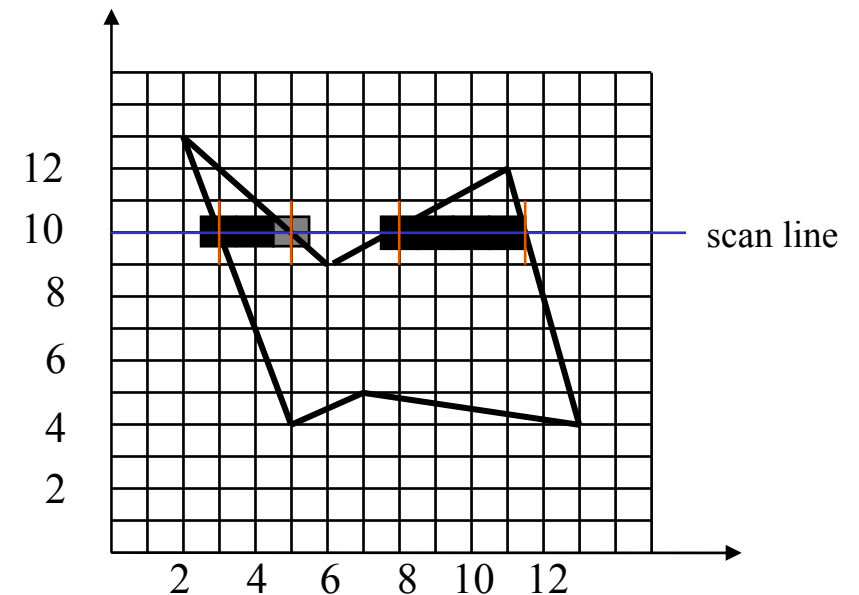
- Gammakorrektur muss vorgenommen werden.
- Bei Vektoren mit geringem Abstand
 - Probleme durch Überzeichnen von Pixeln
 - Einfache Lösung: Nur hellere Werte übernehmen
 - Annahme: Helle Linien auf dunklem Grund
- Grauwert muss mit dem Faktor $1/\cos(\alpha)$ skaliert werden



Rasterisieren von Polygonen

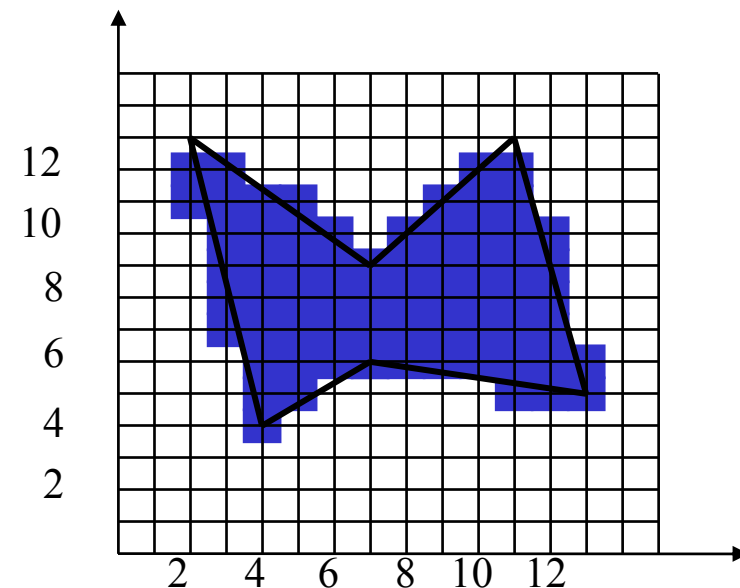
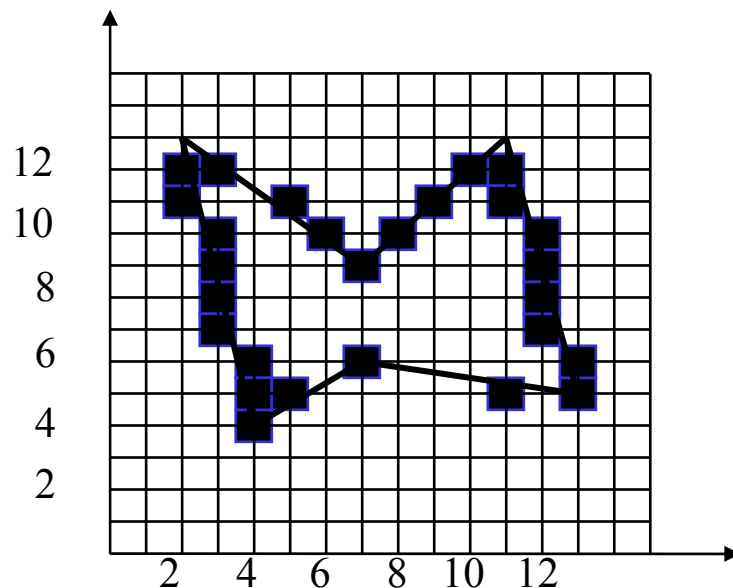
■ Idee: (Scanline Algorithmus)

- Finde Schnittpunkt der scan line mit allen Kanten des Polygons
- Sortiere Schnittpunkte nach wachsender x-Koordinate
- Fülle Pixel zwischen Paaren aufeinanderfolgender Schnittpunkte
 - Regel von der ungeraden Parität
 - Parität ist am Anfang 0
 - mit jedem Schnittpunkt um eins inkrementiert
 - Pixel wird gesetzt falls Parität ungerade



■ Extrema

- MP-Algorithmus auf den Kanten
 - Pixel außerhalb des Polygons
 - Pixel ragen in anderes Polygon hinein

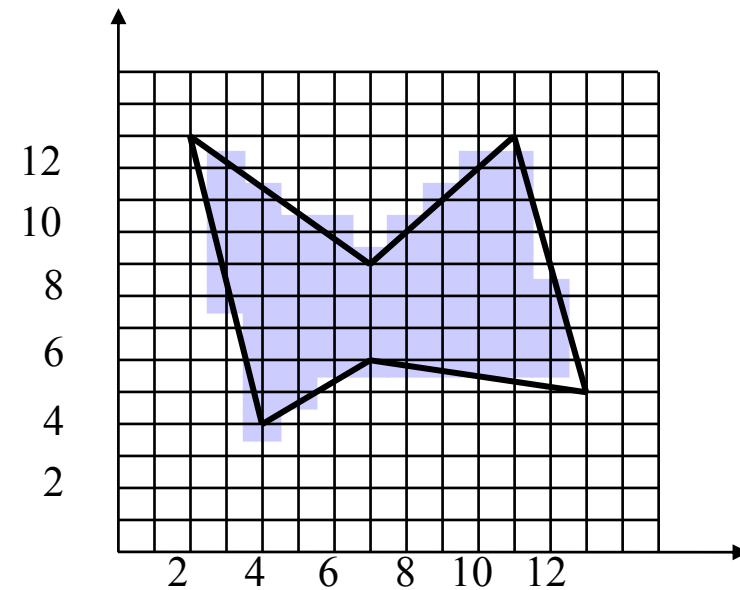
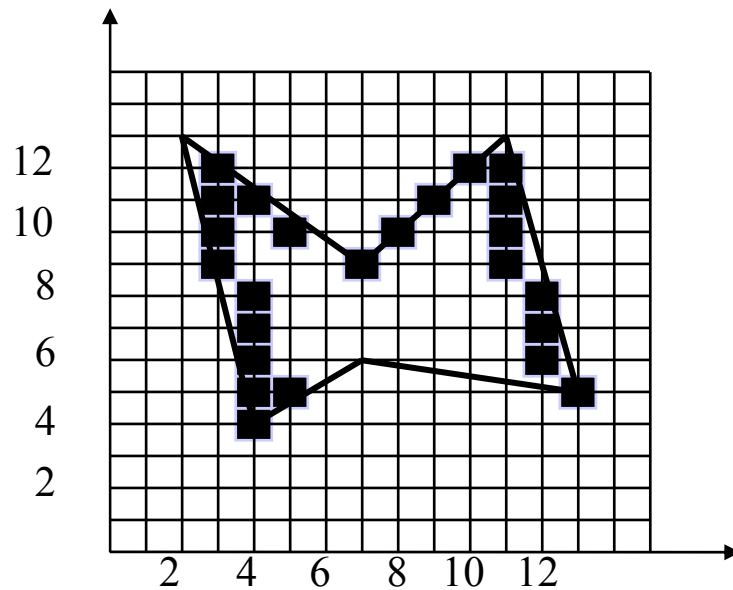


Rasterisieren von Polygonen

Scanline Algorithmus

■ Extrema

- Modifizierter MP-Algorithmus (nur Pixel innerhalb der Polygone zeichnen)

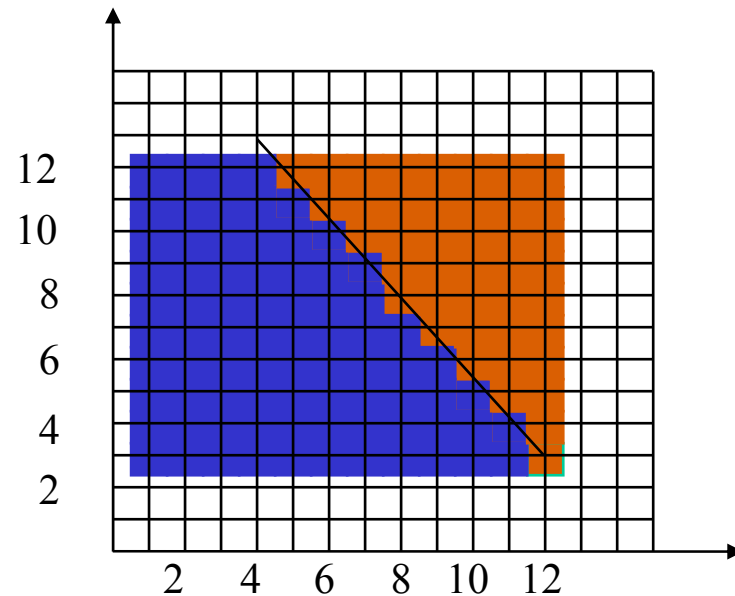
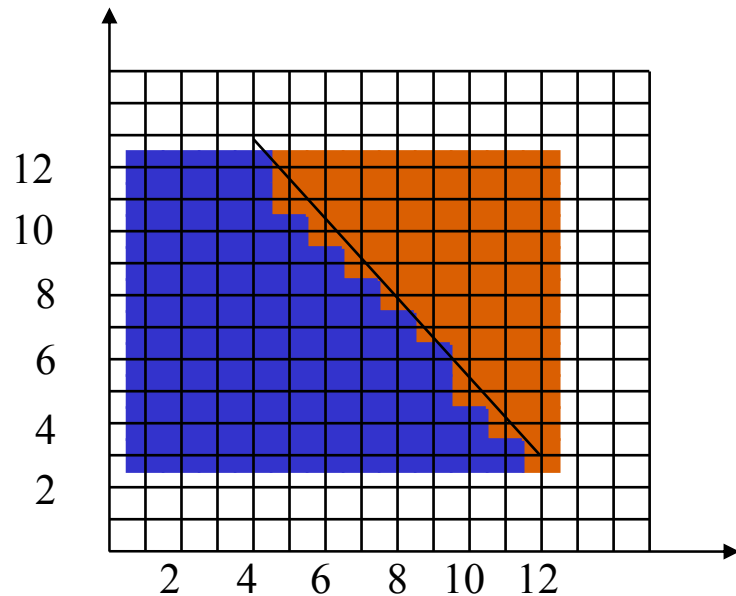


Rasterisieren von Polygonen

Scanline Algorithmus

■ Extrema

- MP-Algorithmus auf den Kanten
 - Pixel ragen in anderes Polygon hinein
- Modifizierter MP-Algorithmus

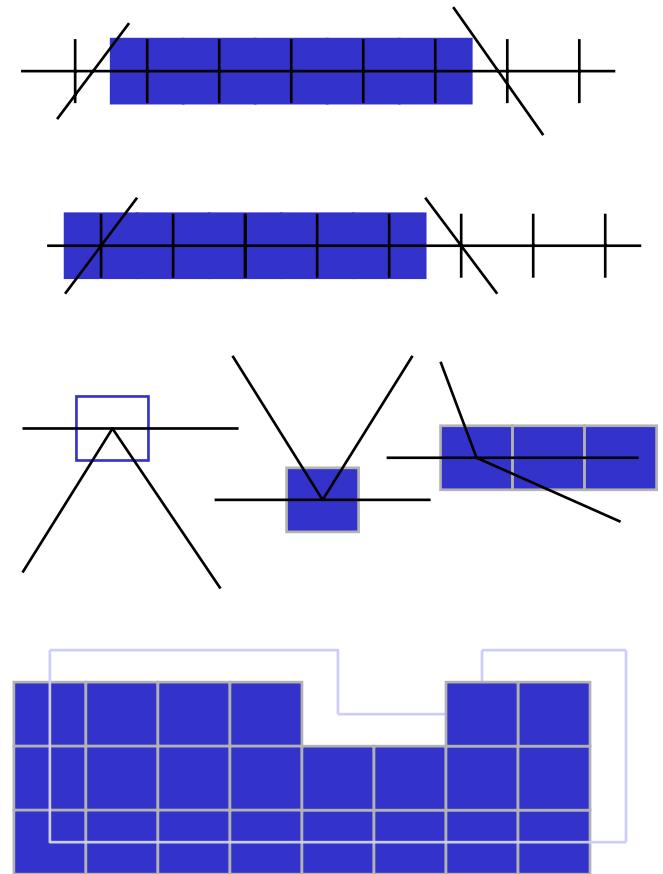


Rasterisieren von Polygonen

Scanline Algorithmus

■ Füllstrategie

- Schnitt Floatingpointwert
 - Innen abrunden, Außen aufrunden
- Schnitt Integerwert
 - Linker Endpunkt eines Spans: Innen
 - Rechter Endpunkt eines Spans: Außen
- gemeinsamer Vertex von Nachbarpolygone
 - Nur y_{\min} -Vertex einer Kante wird zur Berechnung der Parität gezählt
- horizontale Kanten
 - Pixel horizontaler Kanten werden zur Berechnung der Parität nicht gezählt

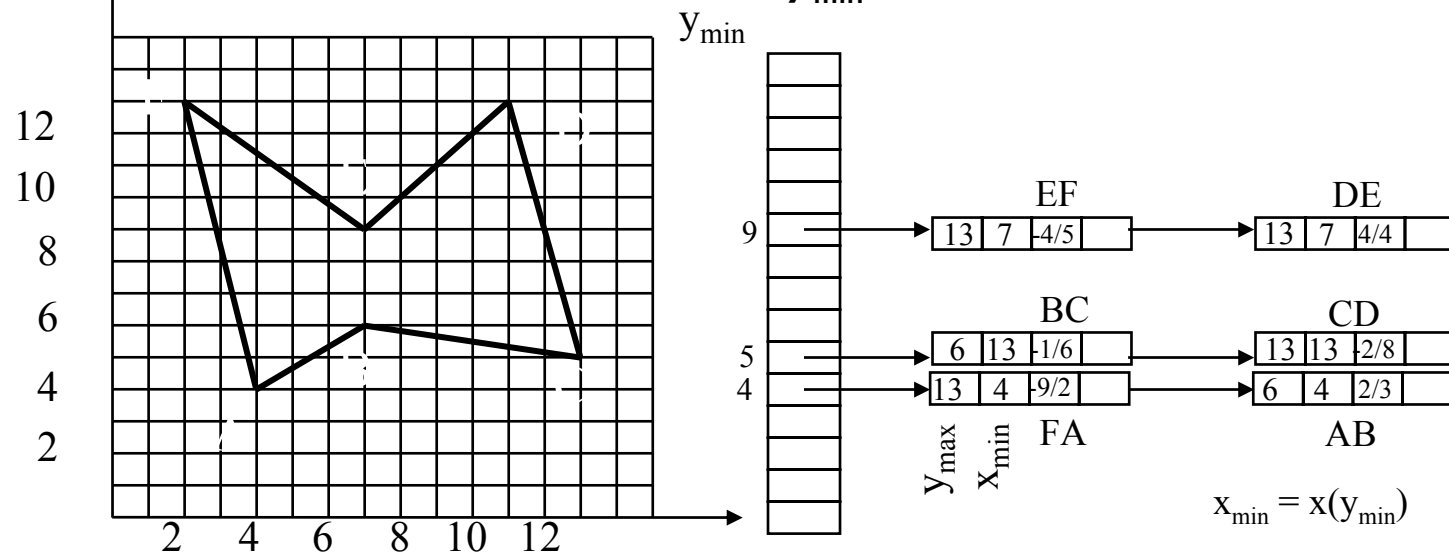


Rasterisieren von Polygonen

Scanline Algorithmus

■ Kanten:

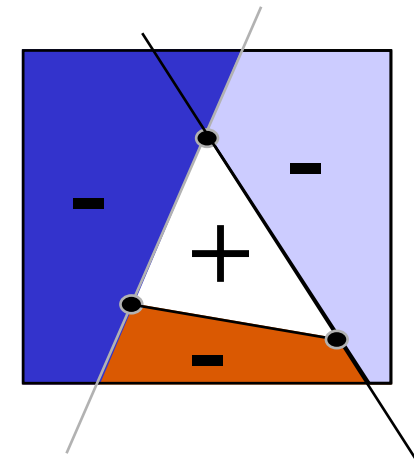
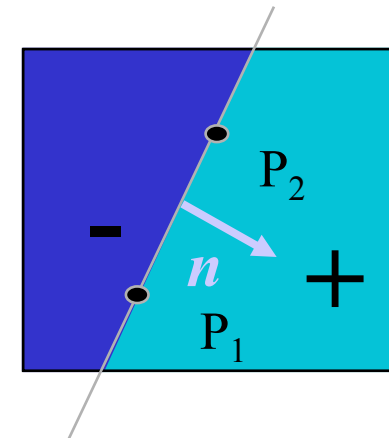
- $e = (x_{\min}, y_{\min}, x_{\max}, y_{\max}, D_y/D_x)$
- AET (Active Edge Table)
 - (e_1, \dots, e_k) sortiert nach x_{Schnitt}
- ET (Edge Table)
 - alle Kanten, sortiert mit Bucket-Sort nach y_{\min}



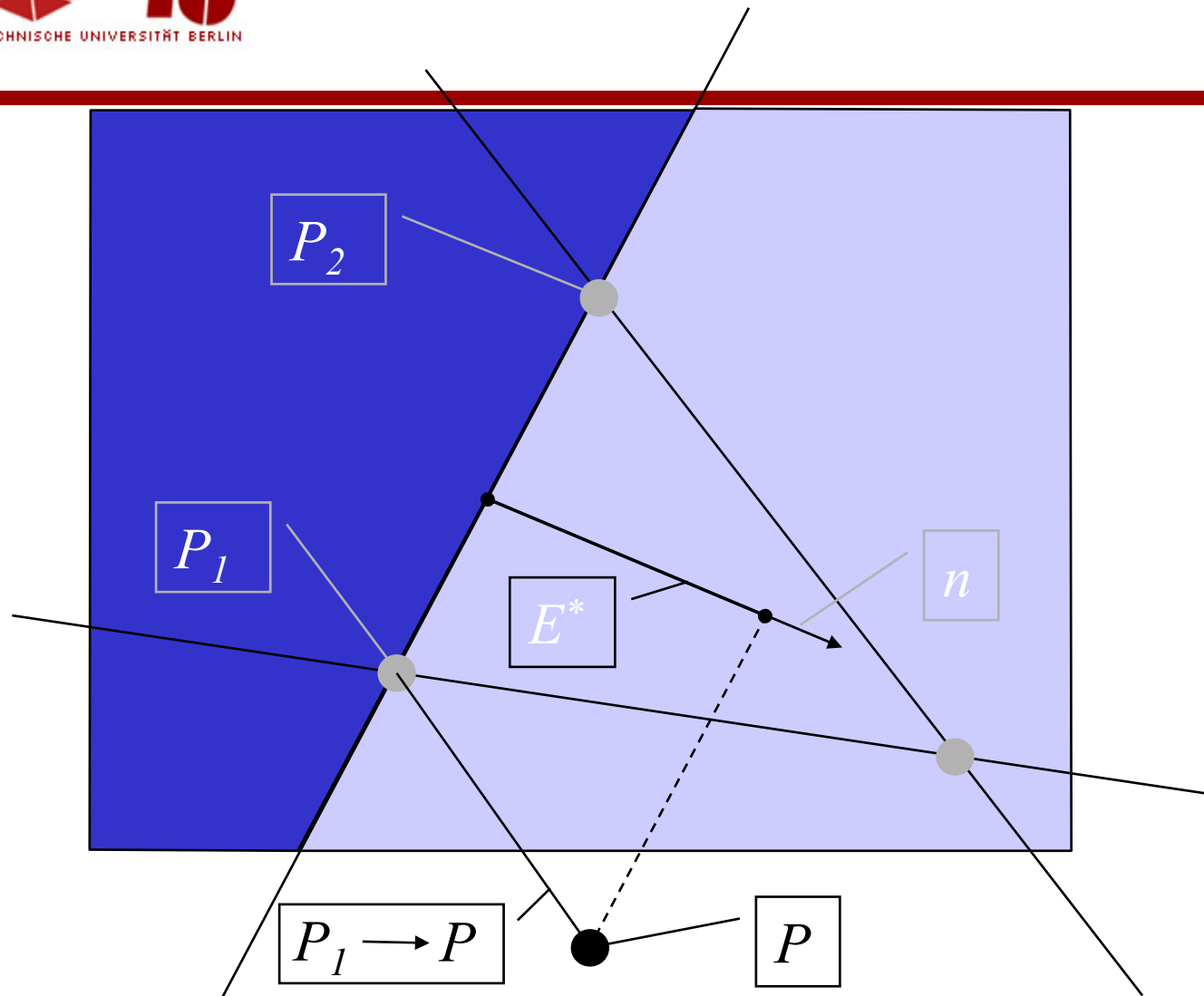
Rasterisieren von Polygonen Dreiecke

$$\begin{aligned}
 P_1 &= (x_1, y_1), P_2 = (x_2, y_2), \\
 n &= (\Delta y, -\Delta x) = (y_2 - y_1, -(x_2 - x_1)), \\
 E(x, y) &= n \cdot (x, y) - n \cdot (x_1, y_1) \\
 &= \Delta y \cdot x - \Delta x \cdot y - (\Delta y \cdot x_1 - \Delta x \cdot y_1)
 \end{aligned}$$

- Falls n normiert wird, so misst E den Abstand zur Kante.



Rasterisieren von Polygonen Dreiecke

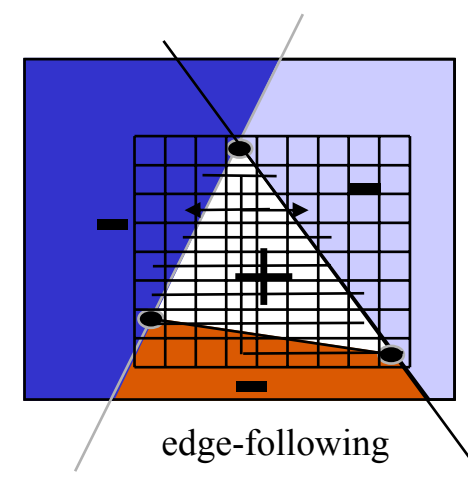
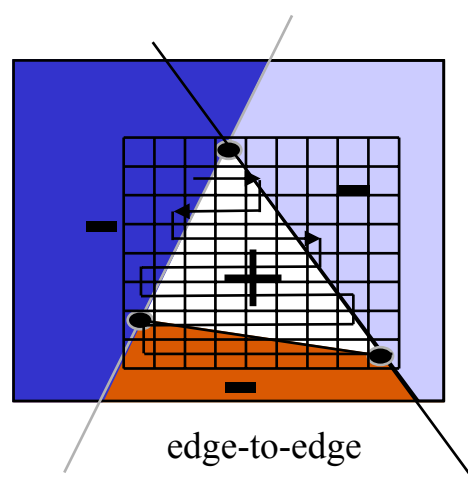
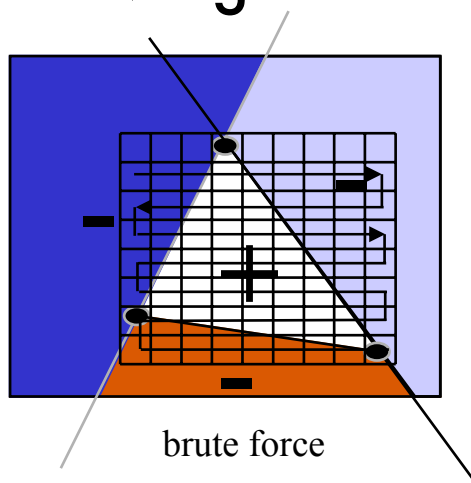


- Die Abstandsfunktion E_i der i -ten Kante kann inkrementell berechnet werden:

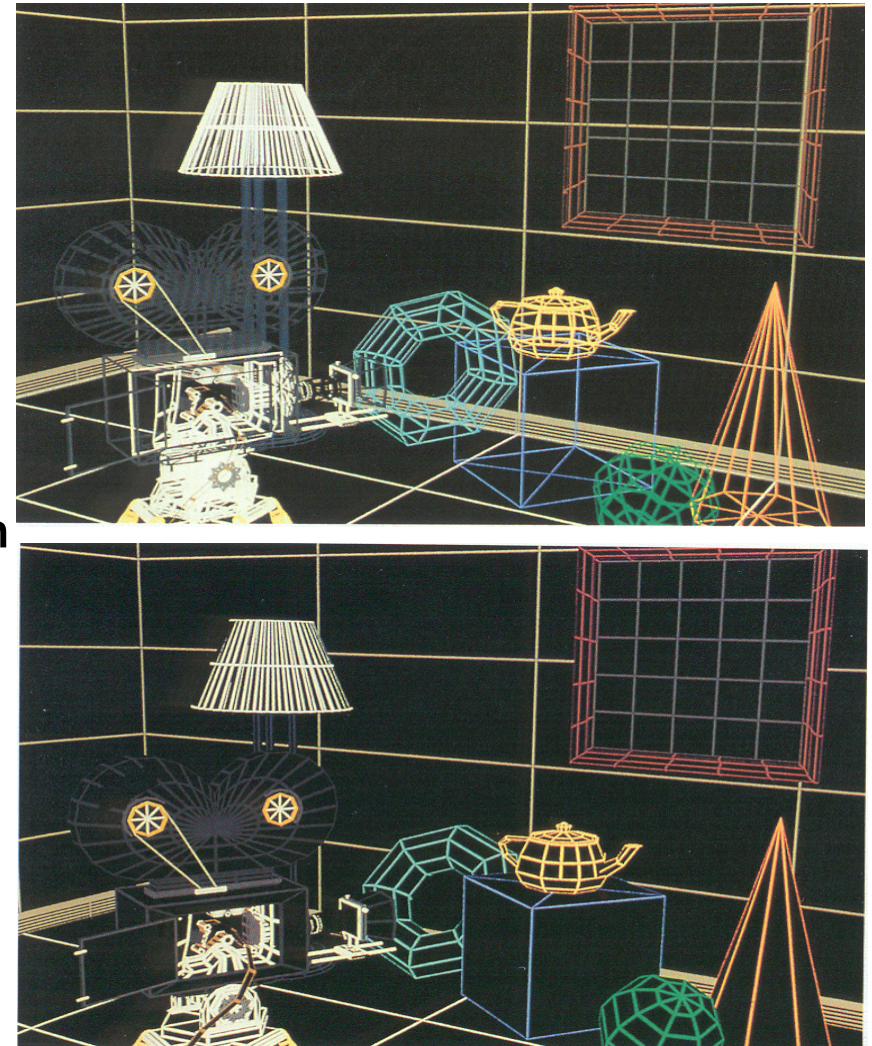
$$E_i(x + 1, y) = E_i(x, y) + \Delta_i y$$

$$E_i(x, y + 1) = E_i(x, y) - \Delta_i x$$

- Traversierungsvarianten (Sonderfälle werden analog zum schon vorgestellten Algorithmus behandelt):

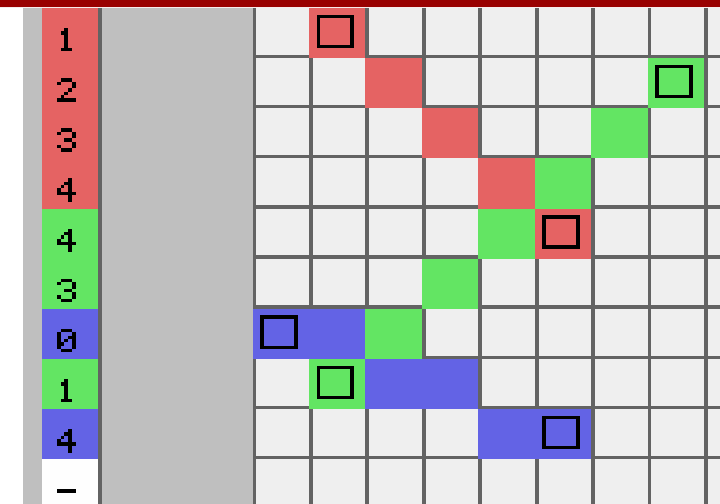


- Verdeckung
 - Mehrere Objekte erhalten durch Projektion gleiche Koordinaten
 - Projektionsäquivalenz
 - Intuitiv: Objekte werden vom gleichen Sehstrahl getroffen
 - Sichtbar ist der dem Auge am nächsten liegende Punkt
 - Ist das Objekt durchsichtig (transparent), wird der dahinterliegende Punkt auch sichtbar, usw.



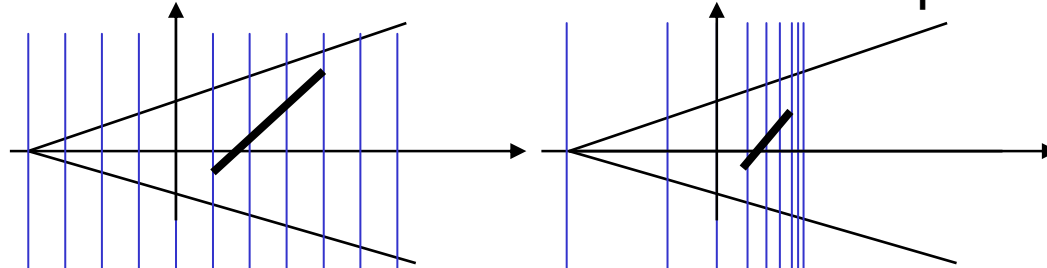
Pixelorientierte Sichtbarkeit

- Einzelne (Bild-)Punkte werden auf Verdeckung bzw. Sichtbarkeit untersucht
- Einfachste Art der Visibilitätsrechnung
- Größte Anzahl von einfachen Tests
- Punkttests liefern Sichtbarkeit explizit
- Linien- und Flächentests liefern die Sichtbarkeit implizit
 - Wenige Punkttests und Ausnutzung von Kohärenz
- Punktorientierte Verfahren sind zweckmäßig für Rasterdisplays
 - Testelement und Ausgabeelement stimmen überein



- Der Tiefenspeicher wurde bereits 1974 vorgeschlagen, aber aus Kostengründen damals nicht realisiert
- Für jeden Bildpunkt wird auch ein z-Wert gespeichert
 - Tiefenbild
- Initialisierung
 - Der Bildspeicher » Hintergrundfarbe
 - z-Speicher » maximaler z-Wert
- Alle Objekte der Szene werden nacheinander gerastert
 - Eine besondere Reihenfolge ist nicht erforderlich

- Für jeden Punkt (x,y) eines darzustellenden Polygons
 1. Berechne $z(x,y)$
 - Perspektivische Transformation erlaubt keine lineare Interpolation mehr!



2. Ist $z(x,y)$ kleiner als der unter (x,y) bereits gespeicherte Wert
 - Schreibe $z(x,y)$ in den z-Speicher und den zugehörigen Farbwert an der Stelle (x,y) in den Bildspeicher.
- Nach der Behandlung aller Objekte steht im Bildspeicher das gewünschte Bild der sichtbaren (Teil-)flächen

- Jede Szene mit jeder Art von Objekten kann behandelt werden
- Komplexität ist unabhängig von der Tiefenkomplexität
- In eine fertige Szene können nachträglich Objekte eingefügt werden
 - Dies ist besonders für die Kombination mit Kameraaufnahmen interessant
- Spezielle Objekte, wie z.B. ein 3D-Cursor, können in der Szene mit korrekter Verdeckung dargestellt werden
- Leicht in Hardware zu realisieren.

- Pro Bildpunkt wird nur ein Objekt gespeichert
 - Daraus resultieren Abtastfehler, die durch Supersampling [höhere Auflösung] verkleinert aber nicht beseitigt werden können
- Transparenz ist prinzipiell nicht realisierbar
- Die Genauigkeit des z-Buffers ist beschränkt
 - Getrennte Objekte erhalten denselben z-Wert
 - Die Farbe wird dann von der Objektreihenfolge bei der Rasterung bestimmt

Beleuchtung eines Primitivs

- Die Bestimmung der Leuchtdichten pro Pixel findet in der Praxis erst während der Rasterisierung statt
- Man kann aber ganz allgemein unterscheiden:
- Flat Shading
 - Normale des Primitivs ergibt einheitliche Helligkeit
- Gouraud Shading
 - Normale in den Eckpunkten ergibt Helligkeitswerte für die Eckpunkte
 - Helligkeitswerte der Eckpunkte werden linear interpoliert
- Phong Shading
 - Eckpunkt-Normalen werden für jeden Punkt linear interpoliert und normiert
 - Helligkeitswert ergibt sich aus interpolierter Normale