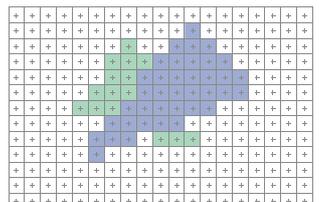





# Computer-Graphik I

## Polygon Scan Conversion

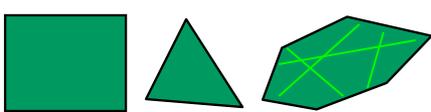


G. Zachmann  
 Clausthal University, Germany  
[zach@in.tu-clausthal.de](mailto:zach@in.tu-clausthal.de)




## Klassifikation der Polygone

Konvex



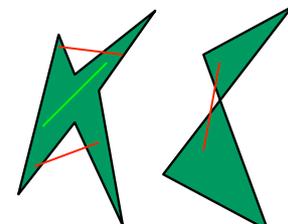
Für jedes Punktepaar in einem konvexen Polygon liegt die Verbindung auch innerhalb des Polygons

Horizontal Konvex



Selbe Definition, gilt hier aber nur für Punkte auf der selben horizontalen Linie

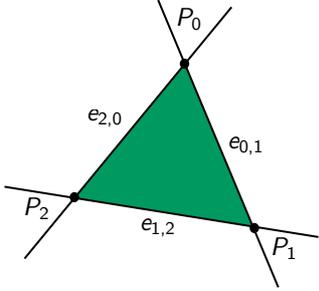
Konkav



G. Zachmann Computer-Graphik 1 - WS 09/10
Scan Conversion: Polygone 2

## Dreiecke

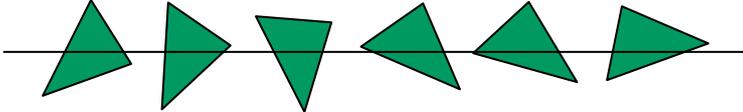
- Dreiecke sind besondere Polygone
- 3D Dreiecke sind immer eben
  - 3 Punkte beschreiben eine Ebene
  - Mit weniger als 3 Punkten kann man keine Fläche beschreiben
- Dreieck = 2D-Simplex ("einfachstes" geom. Objekt, das echt 2-dim. ist)
- Somit sind Dreiecke sehr einfach (sowohl mathematisch wie auch geometrisch)



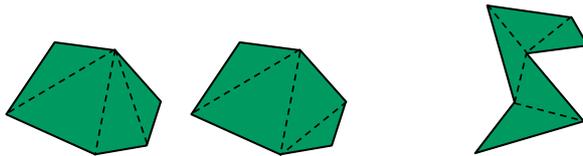
The diagram shows a green triangle with vertices labeled  $P_0$ ,  $P_1$ , and  $P_2$ . The edges are labeled  $e_{2,0}$ ,  $e_{0,1}$ , and  $e_{1,2}$ . Each vertex is also connected to a line extending from the edge it is not on.

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 3

- Dreiecke sind immer konvex  $\rightarrow$  egal wie man ein Dreieck dreht, es gibt nur ein Schnittintervall (= *Span*) für jede Gerade (*Scanline*)



- Satz (I. Bew.):  
Jedes beliebige Polygon kann in eine Menge von Dreiecken zerlegt werden ("Triangulierung")



The left diagram shows a convex pentagon with dashed lines representing internal diagonals, labeled "Konvex". The right diagram shows a concave pentagon with dashed lines representing internal diagonals, labeled "Konkav".

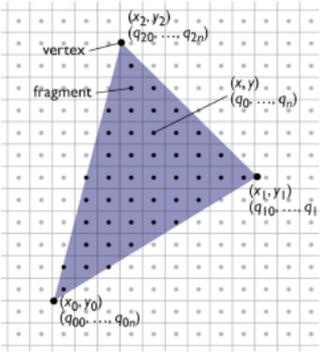
G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 4

- Der Algorithmus zum Rasterisieren erzielt aus den Eigenschaften von Dreiecken Vorteile
- Deswegen
  - ist die Graphik-Hardware optimiert für Dreiecke
  - unterteilen viele Graphikkarten Polygone in Dreiecke
  - Einige Systeme rendern eine Linie, indem sie 2 schmale Dreiecke rendern

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 5

## Rasterisierung von Dreiecken

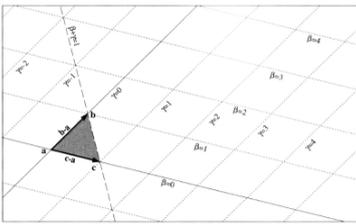
- Eingabe:
  - Drei 2D Punkte (Eckpunkte im Framebuffer / "Pixel-Raum"):  
 $(x_0, y_0); (x_1, y_1); (x_2, y_2)$
  - Mit Attributen  $q$  für jeden Eckpunkt, z.B. Farbe
- Ausgabe:
  - Ganzzahlige Pixel-Koordinaten  $(x, y)$
  - Interpolierte Parameterwerte  $q_{xy}$



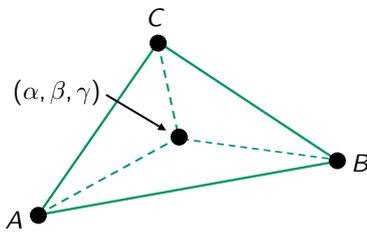
G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 6

### Erinnerung

- Baryzentrische Koordinaten



- Test ob Punkt im Dreieck liegt:
 
$$\alpha > 0 \wedge \beta > 0 \wedge \gamma > 0$$



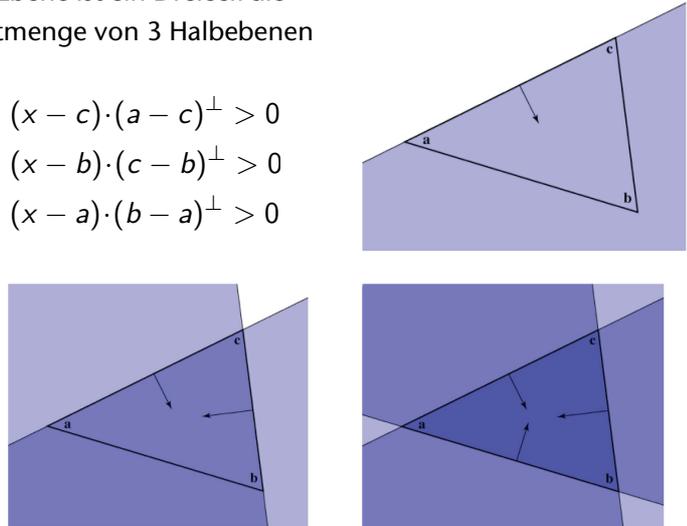
G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 7

### Alternative Betrachtungsweise

- In der Ebene ist ein Dreieck die Schnittmenge von 3 Halbebenen

$$(x - c) \cdot (a - c)^\perp > 0$$

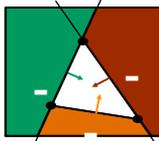
$$(x - b) \cdot (c - b)^\perp > 0$$

$$(x - a) \cdot (b - a)^\perp > 0$$


G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 8

## Algorithmus von Pineda [1988]

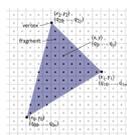
- Idee:
  - Berechne baryzentrische Koordinate für alle Pixel(-mittelpunkte)
 
$$\alpha = F_{BC}(x) = \frac{\mathbf{n}_c \cdot (\mathbf{X} - \mathbf{B})}{\mathbf{n}_c \cdot (\mathbf{C} - \mathbf{B})}, \quad \beta = \dots$$
  - Zeichne Pixel, falls innerhalb des Dreiecks
  - Setze interpolierte Farbe für Pixel:
 
$$\mathbf{c} = \alpha \mathbf{c}_A + \beta \mathbf{c}_B + \gamma \mathbf{c}_C$$



- Algo:
 

```

                for y = y_min ... y_max:
                  for x = x_min ... x_max:
                    berechne alpha, beta, gamma
                    if alpha > 0 and beta > 0 and gamma > 0:
                      c = alpha*c_A + beta*c_B + gamma*c_C
                      zeichne Pixel (x,y) mit Farbe c
            
```



wobei  $x_{min}, x_{max}, y_{min}, y_{max}$  = Bounding-Box von A, B, C

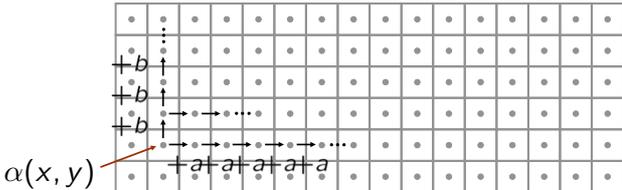
G. Zachmann Computer-Graphik 1 - WS 09/10
Scan Conversion: Polygone 9

## Optimierung

- Beobachtung:  $\alpha$  ist eine **lineare** (eigtl. affine) Funktion in der Ebene, m.a.W.,  $\alpha$  hat die Form
 
$$\alpha = ax + by + c$$

Dito für  $\beta, \gamma$
- Lineare Funktionen können sehr effizient **inkrementell** auf einem Gitter ausgewertet werden:
 
$$\alpha(x + 1, y) = \alpha(x, y) + a$$

$$\alpha(x, y + 1) = \alpha(x, y) + b$$



G. Zachmann Computer-Graphik 1 - WS 09/10
Scan Conversion: Polygone 10

```

linEval(xl, xh, yl, yh, cx, cy, ck):
# setup
compute a, b, c ...
qRow = a*xl + b*yl + c

# traversal
for y = Ymin ... Ymax:
  qPix = qRow
  for x = Xmin ... Xmax:
    draw(x, y, qPix)
    qPix += a
  qRow += b

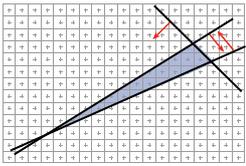
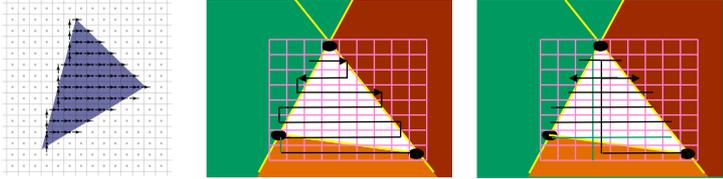
```



$a = .005; b = .005; c = 0$   
(Bildgröße 100x100)

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 11

- Problem: wenn das Dreieck lang und schmal ist, dann werden viele unnötige Berechnungen durchgeführt
- Erinnerung: Dreieck ist konvex
  - Folge: wenn man in der x-Schleife einmal ein Pixel außerhalb erreicht, dann sind alle folgenden in dieser Scanline auch außerhalb

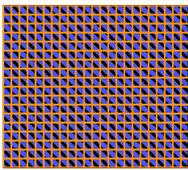



- Weiterer Vorteil des Algorithmus von Pineda: lässt sich relativ leicht parallelisieren

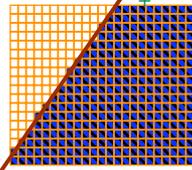
G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 12

## Kurzer Exkurs: Die Pixel-Planes-Architektur

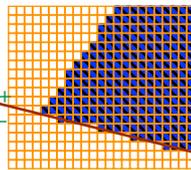
- Eine Geschichte mit "sad end" ...
- Die Idee:
  - Die Berechnung pro Pixel ist extrem einfach, nämlich Auswertung einer linearen Funktion  $Ax + By + C$
  - Also: baue Framebuffer, in dem jedes Pixel ein einfacher Prozessor ist, der solch eine Gleichung für "seine" Koordinaten auswerten kann! ("processor per pixel")
  - Betrachte die 3 Kanten der Reihe nach
  - Lade alle Prozessoren gleichzeitig mit den Koeff. A,B,C der aktuellen Kante



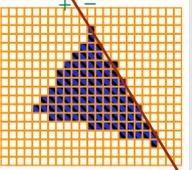
Alle Prozessoren an



Lade Kante 1



Lade Kante 2



Lade Kante 3

G. Zachmann Computer-Graphik 1 - WS 09/10
Scan Conversion: Polygone 13

## Pixel-Planes 4 (1986)

- Features:
  - Full-size (512 by 512 pixel) prototype
  - Used 2048 enhanced memory ICs
  - 1 Geometry Processor
  - 72 bits per pixel
- Performance:
  - 35K triangles/sec
  - Kugeln als Primitive
  - CSG
  - Schatten
- "Lessons Learned":
  - Dreiecke sind klein, daher viele Proc idle
  - Für noch mehr Performance braucht man auch auf dem Geometrie-Level Parallelisierung

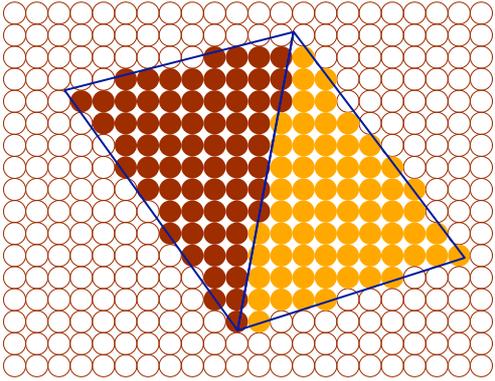




G. Zachmann Computer-Graphik 1 - WS 09/10
Scan Conversion: Polygone 14

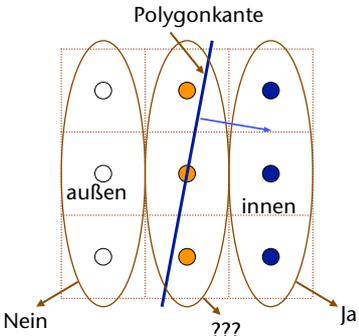
## Vorsicht bei angrenzenden Polygonen

- Behandle angrenzende Polygone korrekt!
  - Vermeide Risse
  - Vermeide Überschneidungen
  - Unabhängig von der Zeichenreihenfolge



G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 15

- Pixel vollständig im Polygon → wird gezeichnet
- Pixel zum Teil im Polygon → ... ?
- Vereinbarung (für den Moment): zeichne nur die Pixel, deren **Zentren im Inneren** des Polygons liegen
- Problem, falls Zentrum des Pixels genau auf der Ecke des Polygons liegt :
  - Nicht zeichnen → Loch
  - Zeichnen → wird möglicherweise 2x gezeichnet (ergibt sog. "z flickering" u.a. Artefakte)



Nein ??? Ja

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 16

■ Problem beim 2x Zeichnen:
 

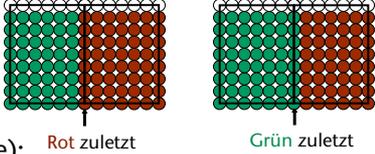
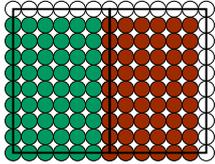
- Das zuletzt gezeichnete Polygon "gewinnt"

■ Mögliche Lösung (gibt noch andere):
 

- Ein Begrenzungspixel (dessen Zentrum genau auf der Kante liegt) gehört **nicht** zu einem Primitiv, wenn das Primitiv links bzw. unterhalb des durch die Kanten aufgespannte Halbraumes liegt (erkennt man an der Normale)
- Verfahre bei konvexen Polygonen genau wie bei Rechtecken

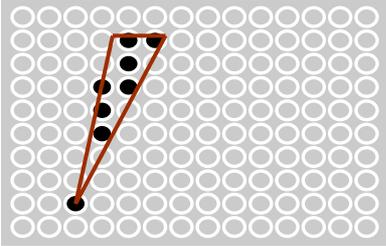
■ Folgerungen für Rechtecke:
 

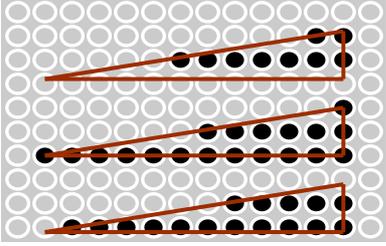
- Spans lassen das rechteste Pixel weg (falls direkt auf Kante)
- Bei jedem Polygon fehlt oberster Span (falls direkt auf Kante)

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 17

## Weiteres Problem

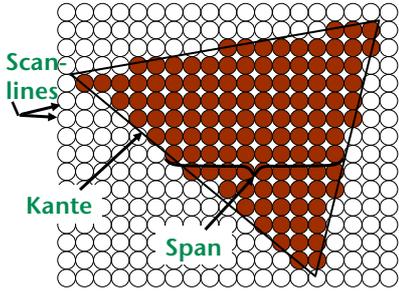
■ Sogenannte "Slivers":
 

■ Moving Slivers:
 

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 18

## Das allgemeine Konzept der Scan Conversion

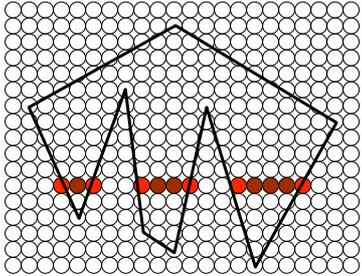
- Gegeben: beliebiges (einfaches) Polygon
- **Span**: Folge **benachbarter** Pixel auf einer Scanline **innerhalb** des Polygons
- Hauptgedanke beim Rasterisieren:
  - Durchlaufe aufeinander folgende Scanlines
  - Berechne pro Scanline alle Spans innerhalb des Polygons
- Allg. Algorithmentechnik:
  - *Sweep-Line-Algorithmus*
  - Im Prinzip nutzt man dabei:
    - räumliche Kohärenz
    - Dimensionsreduktion



G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 19

## Polygon Scan Conversion

- Annahme: gesamtes Polygon ist auf dem Bildschirm / Framebuffer
1. Bestimme alle Punkte auf der Scanline, welche die Kante eines Polygons schneiden
2. Sortiere Schnittpunkte von links nach rechts
3. Gruppierere Schnittpunkte in *Spans* und färbe die Pixel dazwischen

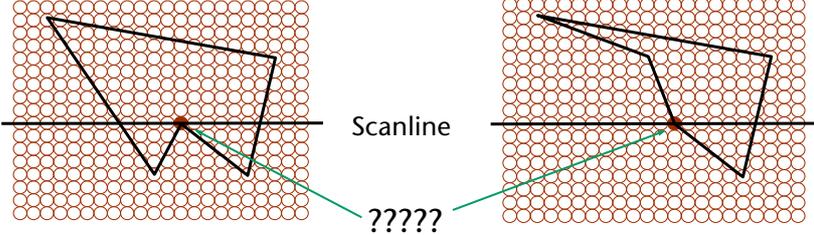


- Schnittpunkte
- Restliche Punkte des Spans

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 20

## Entscheidung "innerhalb" / "außerhalb"

- Wie werten wir Eckpunkte genau auf der Scanline?
  - M.a.W.: begrenzt solch ein Eckpunkt einen Span?



Scanline

?????

- Lösung: zähle einen Eckpunkt genau dann, wenn er das untere Ende der einen, und das obere Ende der anderen Kante ist
- Alternative: "wackle" am Eckpunkt ein wenig ("perturbation")
  - Dann bekommt man im linken Beispiel 2 oder 0 Schnittpunkte, im rechten genau 1 Schnittpunkt

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 21

## Übersicht des Algorithmus'

- Für jede Scanline:
  - berechne Schnittpunkte zwischen Scanline und Kanten des Polygons
  - Führe Scan-Verarbeitung einzeln für jede Scanline durch
- Kann durch Tabelle aller Kanten optimiert werden
- Für jede neue Scanline: schaue in der Tabelle nach, ob eine Kante geschnitten wird
- Haben wir einen Schnitt zwischen Scanline und Kante berechnet, so verwenden wir diese Information in der nächsten Iteration (inkrementell)

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 22

### Inkrementelle Schnittberechnung mit Scanline

- Verwende Kantenkohärenz bei Aktualisierung der X-Schnittpunkte
- Wir kennen  $y = mx + b$ ,  $m = \frac{y_1 - y_0}{x_1 - x_0}$
- Sei  $(x_i, y_i)$  der Schnittpunkt mit der  $i$ -ten Scanline
- Somit
 
$$x_{i+1} = \frac{y_i + 1 - b}{m}$$

$$= \frac{y_i - b}{m} + \frac{1}{m}$$

$$= x_i + \frac{1}{m}$$

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 23

### Die Active Edge Table

- Verwende *active edge table* (AET) zur Verwaltung der Kohärenz
- AET enthält 1 Eintrag pro Schnitt mit der aktuellen Scanline:
  - $y_{max}$ -Wert der Kante
  - x-Wert des Schnittpunktes zwischen Kante und Scanline
  - Schrittweite in x-Richtung (=  $1/m$ )
- Für jede neue Scanline:
  - Berechne **neue** Schnittpunkte für alle Kanten
  - Füge jeden neuen Schnittpunkt zur AET hinzu
  - Entferne Kanten, die nicht mehr geschnitten werden
- Um die AET effizient zu aktualisieren, muß man eine *global edge table* (GET) führen

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 24

## Die Global Edge Table

- GET hat 1 "Slot" für jede Scanline (z.B. 1024)
- Jeder dieser Slots enthält eine Liste von Kanten, deren  $y_{min}$ -Wert gleich dem y-Wert der Scanline dieses Slots sind
- Jede Kante des Polygons hat also in genau 1 Slot der GET einen Eintrag
- Jeder Eintrag in der GET für eine Kante enthält:
  - $y_{max}$ -Wert (= oberes Ende) der Kante
  - $X@y_{min}$ -Wert (der X-Wert beim  $y_{min}$ -Punkt)
  - Schrittweite in X-Richtung (=  $1/m$ )

G. Zachmann Computer-Graphik 1 - WS 09/10
Scan Conversion: Polygone 25

## Beispiel einer Global Edge Table

- Gegeben folgendes Polygon:

(A, B) = [(2, 1), (3, 5)]  
 (B, C) = [(3, 5), (6, 6)]  
 (C, D) = [(6, 6), (3, 8)]  
 (D, E) = [(3, 8), (0, 4)]  
 (E, A) = [(0, 4), (2, 1)]

GET Entries ( $y_{max}$ ,  $X@y_{min}$ ,  $1/m$ )

GET				
8				
7				
6	8	6	-3/2	(C, D)
5	6	3	3	(B, C)
4	8	0	3/4	(D, E)
3				
2				
1	5	2	1/4	(A, B)
0				
		4	2	-2/3 (E, A)

G. Zachmann Computer-Graphik 1 - WS 09/10
Scan Conversion: Polygone 26

Beispiel für die Aktualisierung der AET

Scanline 3

AET

4  $2/3$   $-2/3$

5  $5/2$   $1/4$

$y_{max}$ -Wert der Kante (aus der GET)

x-Wert des aktuellen Schnittpunktes

Schrittweite in x-Richtung ( $\approx 1/m$ ; aus der GET)

(A, B) = [(2, 1), (3, 5)]  
 (B, C) = [(3, 5), (6, 6)]  
 (C, D) = [(6, 6), (3, 8)]  
 (D, E) = [(3, 8), (0, 4)]  
 (E, A) = [(0, 4), (2, 1)]

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 27

Scanline 4

AET

8 0  $3/4$

5  $11/4$   $1/4$

(A, B) = [(2, 1), (3, 5)]  
 (B, C) = [(3, 5), (6, 6)]  
 (C, D) = [(6, 6), (3, 8)]  
 (D, E) = [(3, 8), (0, 4)]  
 (E, A) = [(0, 4), (2, 1)]

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 28

Scanline 5

AET

8	3/4	3/4
6	3	3

$(a, b) = [(2, 1), (3, 5)]$   
 $(b, c) = [(3, 5), (6, 6)]$   
 $(c, d) = [(6, 6), (3, 8)]$   
 $(d, e) = [(3, 8), (0, 4)]$   
 $(e, a) = [(0, 4), (2, 1)]$

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 29

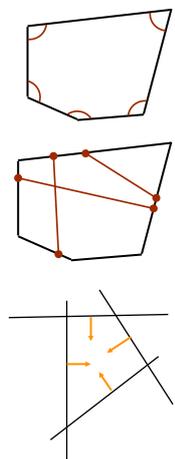
## Der Scan-Line Algorithmus für einfache Polygone

1. Erstelle Global Edge Table (GET) mit allen Kanten des Polygons
2.  $Y$  erhält den Wert der kleinsten  $y$ -Koordinate aller Einträge in GET
3. Initialisiere eine leere Active Edge Table (AET)
4. Wiederhole bis GET und AET leer sind:
  1. Aufnehmen aller Kanten aus GET in AET, für die  $y_{min} = Y$
  2. Entferne alle Kanten aus AET, für die  $y_{max} = Y$
  3. Sortiere AET nach  $x$  (kann man einsparen)
  4. Färbe Pixel zwischen den Schnittpunktpaaren (= spans) aus AET
  5. Für jede Kante aus AET, setze  $x = x + 1/m$
  6. Setze  $Y = Y + 1$  zur Aktivierung der nächsten Scanline

G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 30

## Zerlegung konkaver Polygone

- OpenGL kann nur konvexe Polygone zeichnen!
- Definition "konvex":
  - Innenwinkel  $\leq 180^\circ$
  - Alle Verbindungsstrecken zwischen Randpunkten bleiben innerhalb
  - Konvexe Hülle der Eckpunkte
  - Schnitt von Halbebenen
- Ann. im Folgenden: keine kollinearen Punkte!
  - Sonst: Spezialfall mit spezieller Behandlung (z.B. Punkt überspringen)
- Aufgabe im Folgenden:
  1. Teste, ob Polygon konvex / konkav
  2. Zerlege gegebenenfalls das Polygon

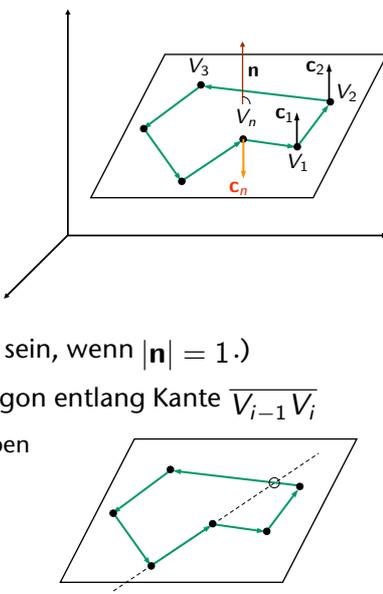


G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 31

## Lösung 1: Kreuzprodukt

- Berechne
 
$$\mathbf{c}_i = (V_i - V_{i-1}) \times (V_{i+1} - V_i)$$
- Teste
 
$$\mathbf{n} \cdot \mathbf{c}_i > 0$$

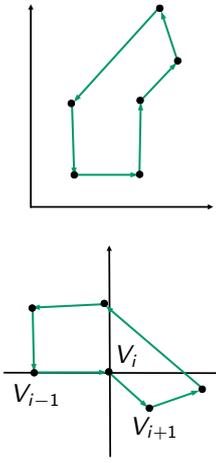
(theoretisch müßte  $\mathbf{n} \cdot \mathbf{c}_i = \pm |\mathbf{c}_i|$  sein, wenn  $|\mathbf{n}| = 1$ .)
- Bei korrekten Kanten: zerlege Polygon entlang Kante  $\overline{V_{i-1}V_i}$ 
  - Achtung kann mehrere Schnitte geben



G. Zachmann Computer-Graphik 1 - WS 09/10 Scan Conversion: Polygone 32

## Lösung 2: „Rotationsmethode“

- Transformiere Polygon in die XY-Ebene
- Für jeden Eckpunkt  $V_i$ :
  - Transformiere/rotiere Polygon so, daß
    - $V_i$  im Ursprung
    - $V_{i-1}$  auf der negativen X-Achse
  - Teste: Y-Koordinate von  $V_{i+1} > 0$
- Für konkave Ecken: schneide Polygon durch X-Achse in 2 oder mehr(!) Teile
  - Zerlege Teile rekursiv weiter



G. Zachmann Computer-Graphik 1 - WS 09/10
Scan Conversion: Polygone 33