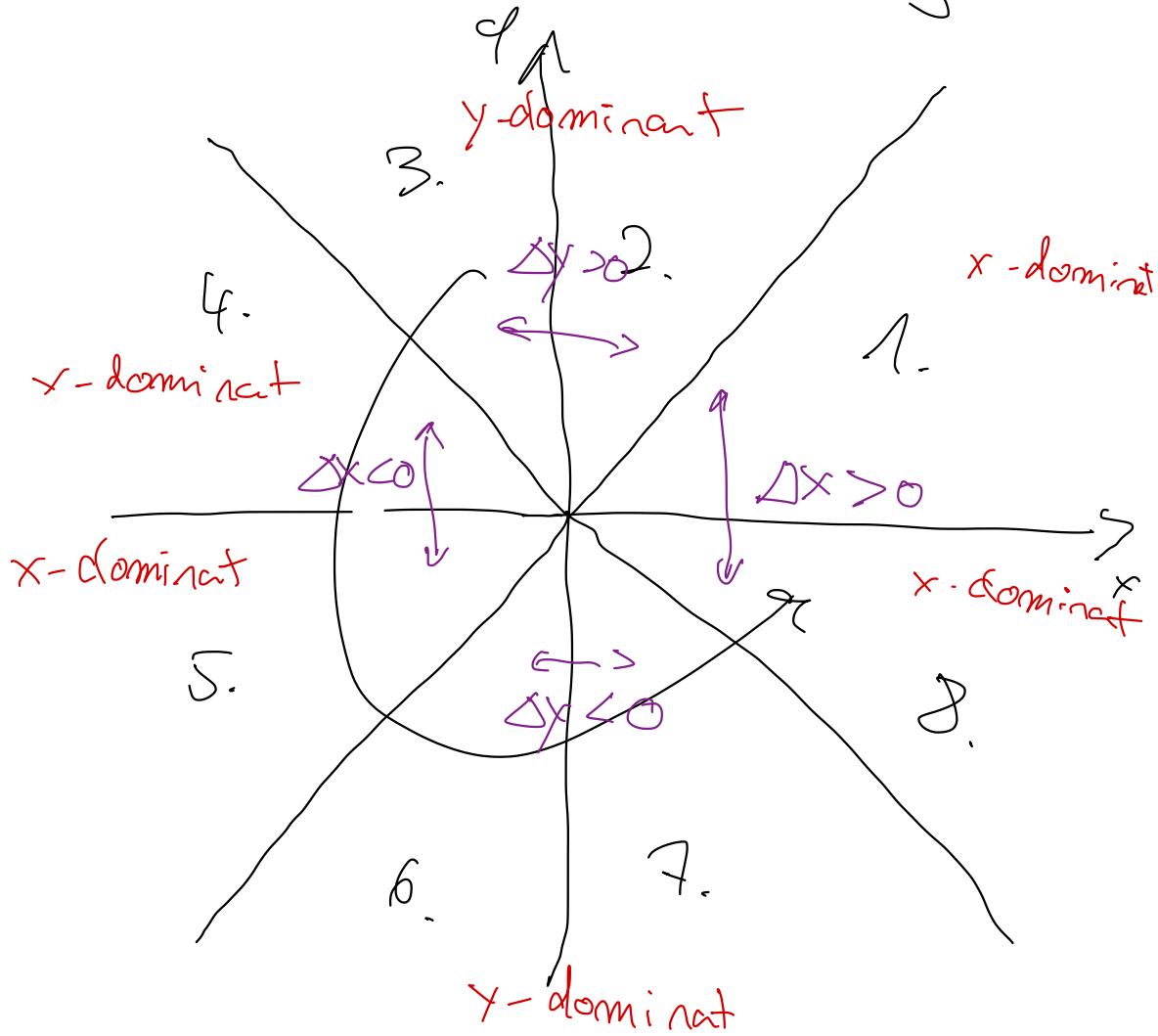


# Bresenham Line Drawing



$x$ -dominant :  $|\Delta x| > |\Delta y|$

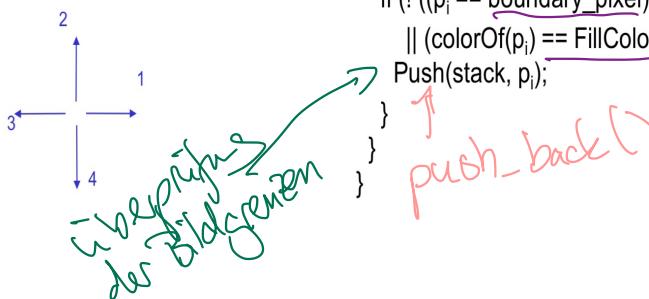
$y$ -dominant :  $|\Delta y| > |\Delta x|$

# Seed Filling

## 3.4 Füllalgorithmen

### Saatpunkt-Methode / Seed-Fill

- Einfacher Saatkorn-Algorithmus
- 4 Bewegungsrichtungen
- randdefiniertes Gebiet
- FILO/LIFO-Prinzip (Stack)
- Eventuell werden Pixel mehrfach im Stack abgelegt (und gefärbt)

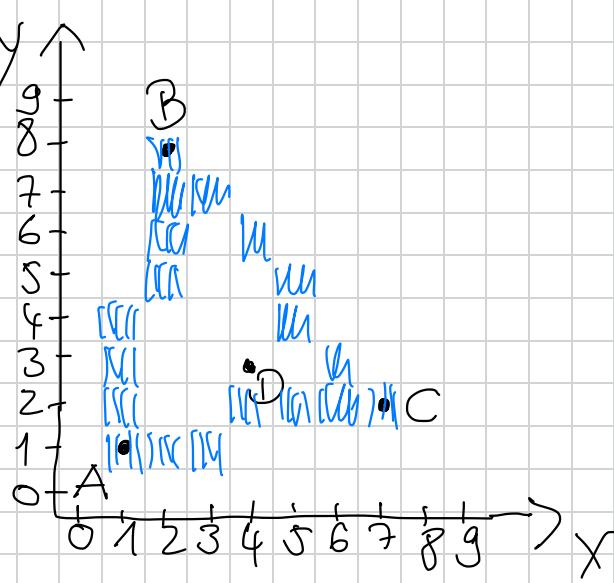


Set Color → SetValue

\* getValues()

getValue in Image Schreiben

Farbvergleich mit !=



$B \rightarrow C$

$$B(2, 8)$$

$$C(7, 2)$$

$$\begin{aligned}\Delta x &= 7 - 2 = 5 \\ \Delta y &= 2 - 8 = -6\end{aligned}$$

$A \rightarrow B$

$$A(1, 1)$$

$$B(2, 8)$$

$$\begin{aligned}\Delta x &= 2 - 1 = 1 \\ \Delta y &= 8 - 1 = 7\end{aligned}$$

O2

$A \rightarrow C$

$$A(1, 1)$$

$$C(7, 2)$$

$$\begin{aligned}\Delta x &= 7 - 1 = 6 \\ \Delta y &= 2 - 1 = 1\end{aligned}$$

O1

## Änderungen

O2, O3, O6, O7 : x & y tauschen

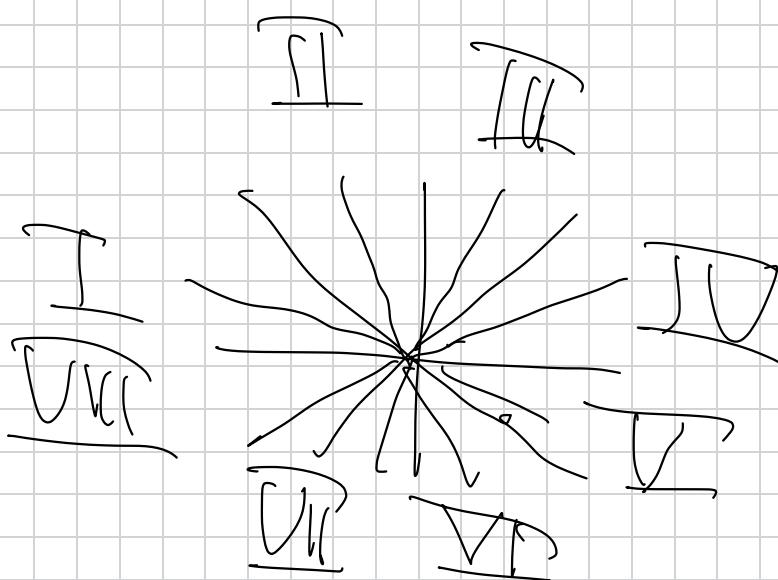
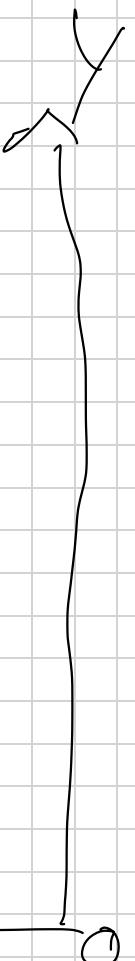
O1, O4, O5, O8 : so wie auf Folien

O3, O4, O5, O6 :  $\Delta x = -\Delta x$ ; --x anstatt +x

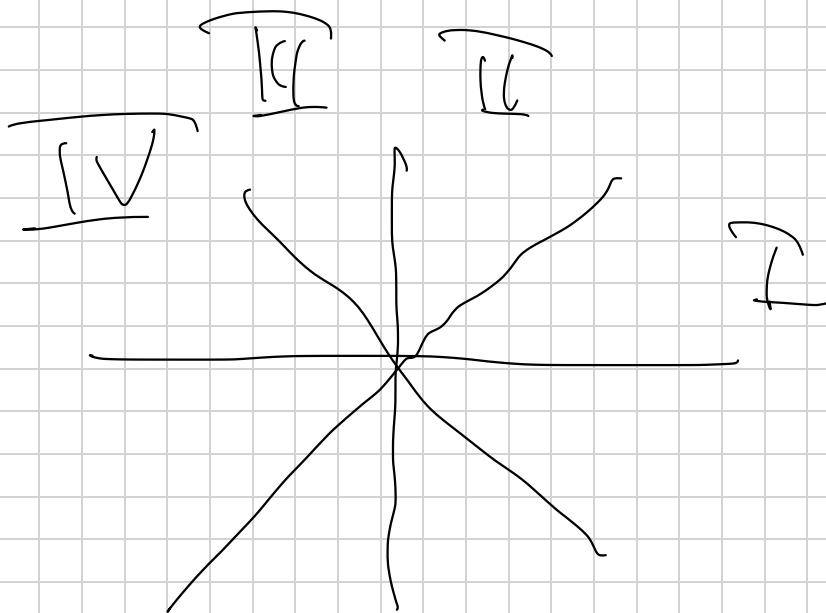
O1, O2, O7, O8 : bleibt  $\Delta x$  & +x

O5, O6, O7, O8 :  $\Delta y = -\Delta y$ ; --y anstatt +y

O1, O2, O3, O4 : bleibt  $\Delta y$  & +y

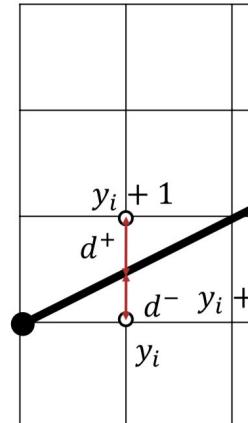


X ← QD

A horizontal line segment with an arrow pointing left, labeled "X" on the left end and "QD" on the right end.

- Startwerte:

$$\begin{aligned}
 e_1 &= \Delta x E_1 \\
 &= \Delta x(d_1^- - d_1^+) \\
 &= \Delta x(a - (1 - a)) \\
 &= \Delta x(2a - 1) \\
 &= \Delta x\left(2 \frac{\Delta y}{\Delta x} - 1\right) \\
 &= 2\Delta y - \Delta x
 \end{aligned}$$



## 3.2 Rasterung von Geraden: Bresenham

```

// (x1, y1), (x2, y2) Ganzahlig
// x1 < x2, y1 < y2
x = x1; y = y1;
dx = x2 - x1; dy = y2 - y1; dy = -dy;
e = 2 * dx - dy; // Initialisierung
for(i = 1; i <= dy; i++){
    // Schleife fuer x
    plot(x, y);
    if (e >= 0) {
        // oberen Punkt Zeichnen (y erhöhen)
        ++x;
        e -= 2 * dy;
    }
    --y;
    e += 2 * dx;
}
plot(x, y);

```

- Erster Oktant
- Ausschließlich ganzzahlige Operanden

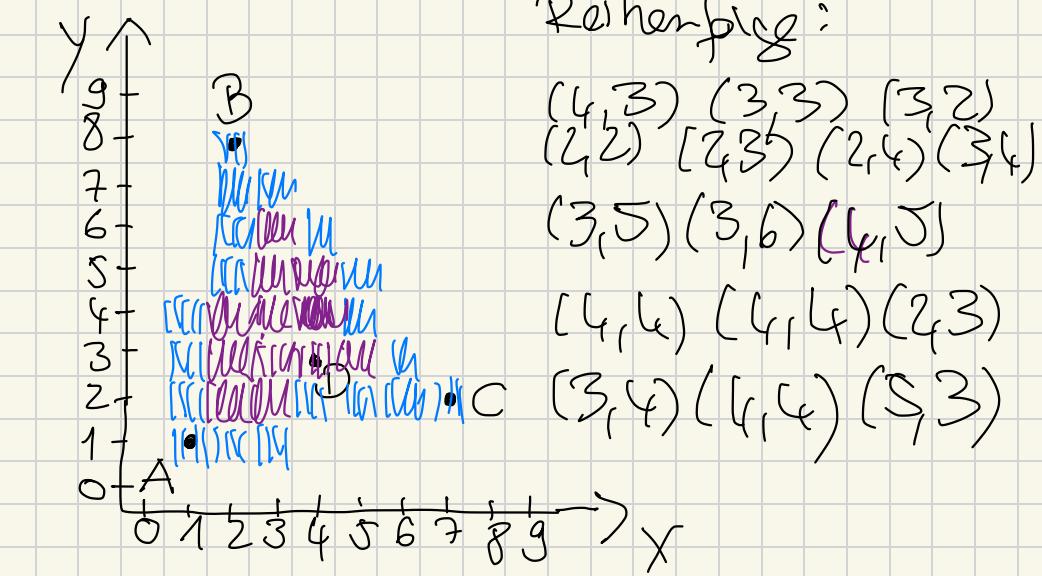
$\Delta x = 5$	$\Delta y = 6$	$(x, y)$	$e$	$plot$
5	4	(2, 8)	8	
4	2	(3, 7)	-8	
6	0	(4, 6)	-10	
5	-2	(5, 5)	-12	
4	-4	(5, 4)	-8	
6	-6	(6, 3)	-6	
7	-4	(7, 2)	2	

## 3.2 Rasterung von Geraden: Bresenham

```
// (x1, y1), (x2, y2) Ganzahlig
```

**Beispiel:**

x



Reihenfolge:

$(4, 3)$   $(3, 3)$   $(3, 2)$   
 $(4, 2)$   $(4, 3)$   $(2, 4)$   $(3, 4)$   
 $(3, 5)$   $(3, 6)$   $(4, 5)$   
 $(4, 4)$   $(4, 4)$   $(2, 3)$   
 $(3, 4)$   $(4, 4)$   $(5, 3)$

~~(3, 6)~~  
~~(4, 5)~~  
~~(3, 5)~~  
~~(4, 4)~~  
~~(3, 4)~~  
~~(2, 4)~~  
~~(2, 3)~~  
~~(2, 2)~~  
~~(3, 2)~~  
~~(2, 3)~~  
~~(3, 4)~~  
~~(3, 3)~~  
~~(4, 4)~~  
~~(5, 3)~~  
~~(4, 3)~~  
Stack

## 3.4 Füllalgorithmen

### Scan-Line-Methode

#### Preprocessing

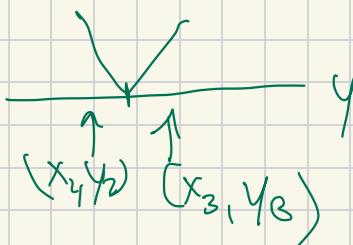
- Ermittle für jede Polygonkante die **Schnittpunkte mit den Scan-Lines** in der Pixelmitte
  - Bresenham
  - DDA-Algorithmus
- Igniere dabei **horizontale Kanten**
- **Speichere** jeden Schnittpunkt  $(x, y)$  in einer Liste
- **Sortiere die Liste** dann von oben nach unten und von links nach rechts

Linien des Polygones

#### Scan-Conversion

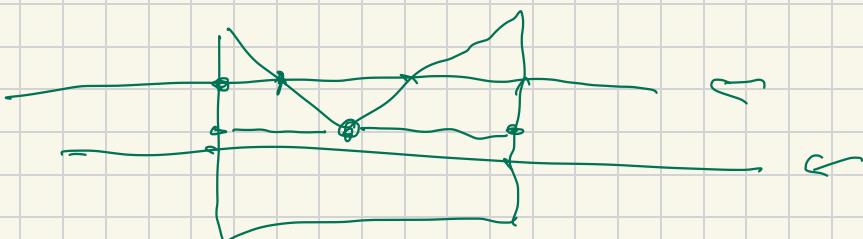
- Betrachte jeweils zwei direkt **aufeinander folgende Schnittpunkte**  $(x_1, y_1)$  und  $(x_2, y_2)$  der Liste
  - Listenelemente 1 und 2
  - Listenelemente 3 und 4
  - ...
- Aufgrund des Preprocessing gilt für die Scan-Line  $y$   
 $y = y_1 = y_2$  und  $x_1 \leq x_2$
- Zeichne alle Pixel auf der Scan-Line  $y$ , für die gilt:  
 $x_1 \leq x < x_2$  mit ganzzahligem  $x$

Fläche füllen



$$y_2 > y \quad \& \quad y_3 > y \\ y_2 < y \quad \& \quad y_3 < y$$

nur Anfang &  
Ende in Liste  
speichern



# Transformationen

P

$M_{\text{Translation}}$

Matrix für die  
Translation

$M_{\text{Scale}}$

Matrix für  
Skalierung

$M_{\text{Rotation}}$

Matrix für  
Rotation

$M_x$  Rotation,  $M_y$  Rotation,  $M_z$  Rotation

Matrizen für Rotation  
um die jeweilige Achse

Aufg. 2

$$r = \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{pmatrix} \rightarrow \text{Rotationsmatrix}$$

$$r = \begin{pmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{pmatrix}$$

$$M_x^{\alpha_x} \times M_y^{\alpha_y} * M_z^{\alpha_z} = M_{\text{Rotation}}$$

$M_{\text{Translation}}$

$M_{\text{Scale}}$

$$S = [s_x, s_y, s_z]$$

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\Rightarrow t = (t_x, t_y, t_z)$$

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M \cdot P = \underline{t + P}$$

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \\ 0 \end{pmatrix}$$

Ortsvektor  
G(Point)

Richtungsvektor  
G(Vector)

$$M_R * M_S * (P + t)$$

$$\begin{aligned} & (M_R * (M_S * (M_T * P))) \\ &= M_R * ((M_S * M_T) * P) \\ &= (M_R * (M_S * M_T)) * P \end{aligned}$$

$M$

M Matrix

update Matrix

get Transformation  
render Scene

Model

$$S = (S_x, S_y, S_z)$$

Set Scale ( $S'$ )

Hintergrund-  
ausführung

$$S = S * S'$$

Ersatz

$$S = S'$$

ist in Grad

Rotation

$$r = (0, 0, 0)$$

sin, cos

brauchen rad

umrechnung

Skalierung

$$S = (1, 1, 1)$$

Translation

$$t = (0, 0, 0)$$

# Scene::intersect

→ Loop über Shaders

- Abfragen des HitRecord Parameter:

- sphereId: index Sphere

- triangleId: -1

- modelId: -1

- bool auf true

→ Loop über Models

→ Loop über triangles

→ Triangle mit transformierten Koordinaten

→ intersectTriangle

- Abfragen des HitRecords

- triangleId: index

- modelId: index

- sphereId: -1

Scene = sphere intersection

$$x^2 + px + q = 0$$

$$t_{1/2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$$

Diskriminante  $d$

$$d < 0$$

$$d = 0$$

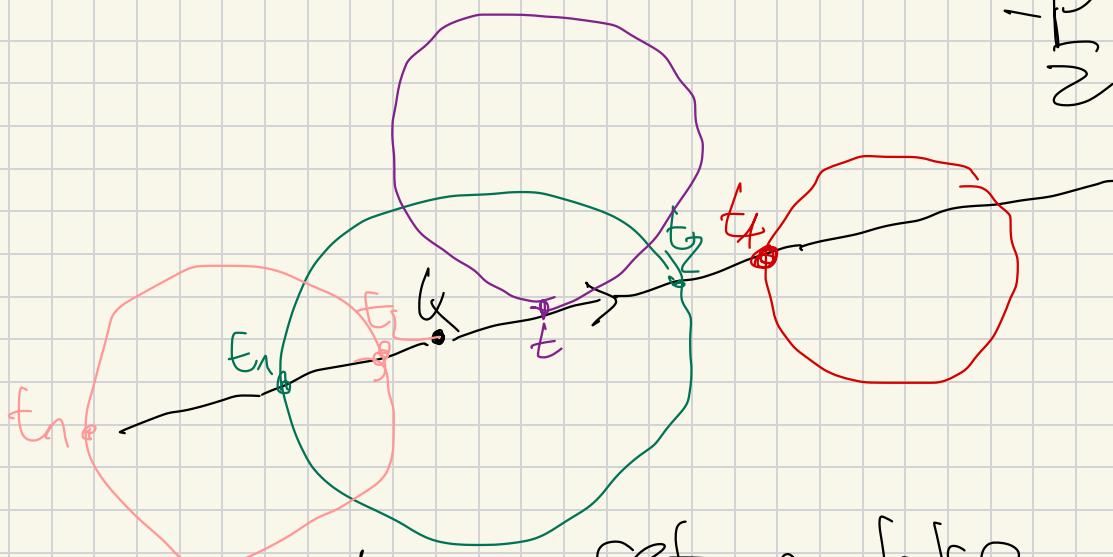
$$d > 0$$

↓  
keine Lösung  
return false

↓  
1 Lösung

↓  
2 Lösungen

$$-\frac{p}{2} - \sqrt{d}$$

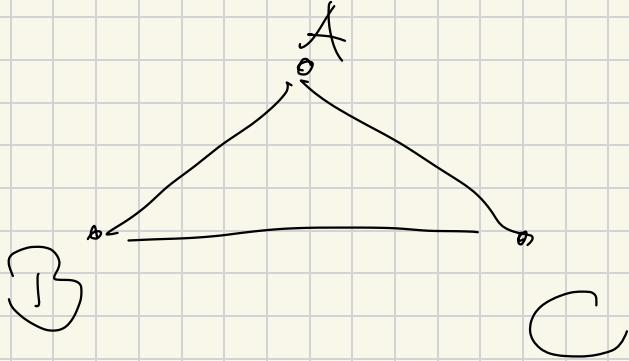


$t < 0$ : return false

$t > 0$ :  $t \rightarrow$  hitRecord.parameter  
else

$S = \text{Ray.Origin} + t \cdot \text{Ray.Direction}$   
intersection Point  
 $t = \text{parameter}$

# Dreieck ABC



$$\text{Flächvektor } \vec{A} + r \cdot \vec{AB} + q \cdot \vec{AC} = \vec{x}$$

$$\text{Strahl: } \vec{e} + t \cdot \vec{v} = \vec{x}$$

Normalenform:  $\vec{n} = \vec{AB} \times \vec{AC}$

$$\vec{B} - \vec{A} \quad \vec{C} - \vec{A}$$

$$(\vec{x} - \vec{A}) \cdot \vec{n} = 0$$

$$\underbrace{(\vec{A} - \vec{e}) \cdot \vec{n}}_{\vec{v} \cdot \vec{n}} = t$$

$$\vec{v} \cdot \vec{n} = 0 \rightarrow \text{Strahl ist parallel zum Dreieck}$$

$\rightarrow$  kein Schnittpunkt  
 $\rightarrow$  false

$t > \text{hitRecord.parameters} \rightarrow \text{false}$

$t \leq 0 \rightarrow \text{false}$

$P = \text{ray.origin} + t \cdot \text{ray.direction}$

$$\alpha = \frac{\Delta PBC}{\Delta ABC} = \frac{(\vec{PB} \times \vec{PC}) \cdot \vec{n}_{ABC}}{(\vec{AB} \times \vec{AC}) \cdot \vec{n}_{ABC}}$$

$$\beta = \frac{\Delta APC}{\Delta ABC} = \frac{(\vec{AP} \times \vec{AC}) \cdot \vec{n}_{ABC}}{(\vec{AB} \times \vec{AC}) \cdot \vec{n}_{ABC}}$$

$$\gamma = \frac{\Delta ABP}{\Delta ABC} = \frac{(\vec{AB} \times \vec{AP}) \cdot \vec{n}_{ABC}}{(\vec{AB} \times \vec{AC}) \cdot \vec{n}_{ABC}}$$

$$\alpha \leq \delta, \beta, \gamma \leq 1$$

$$\alpha + \beta + \gamma = 1$$

$$\gamma = 1 - \alpha - \beta$$

$$\alpha + \beta < 1$$

$\alpha$  berechnen

$$\alpha < 0, \alpha > 1 \rightarrow \text{false}$$

$\beta$  berechnen

$$\beta < 0, \beta > 1 \rightarrow \text{false}$$

$$\alpha + \beta > 0 \rightarrow \text{false}$$

anso gisten  $\rightarrow$  Erreic

ComputeImageRow

- Ray Erzeugen
- HitRecord setzen
- intersect
- shade aufrufen
- pixel einfügen

Shade

- Farbe im HitRecord setzen

→ Sphere Id, model Id, triangle Id = -1  
Color = Background Color Setzen  
Parameter =  $\infty$   
std::limits<double> max

größte double-Zahl,  
die darstellbar ist  
 $c_{\text{limit}}$