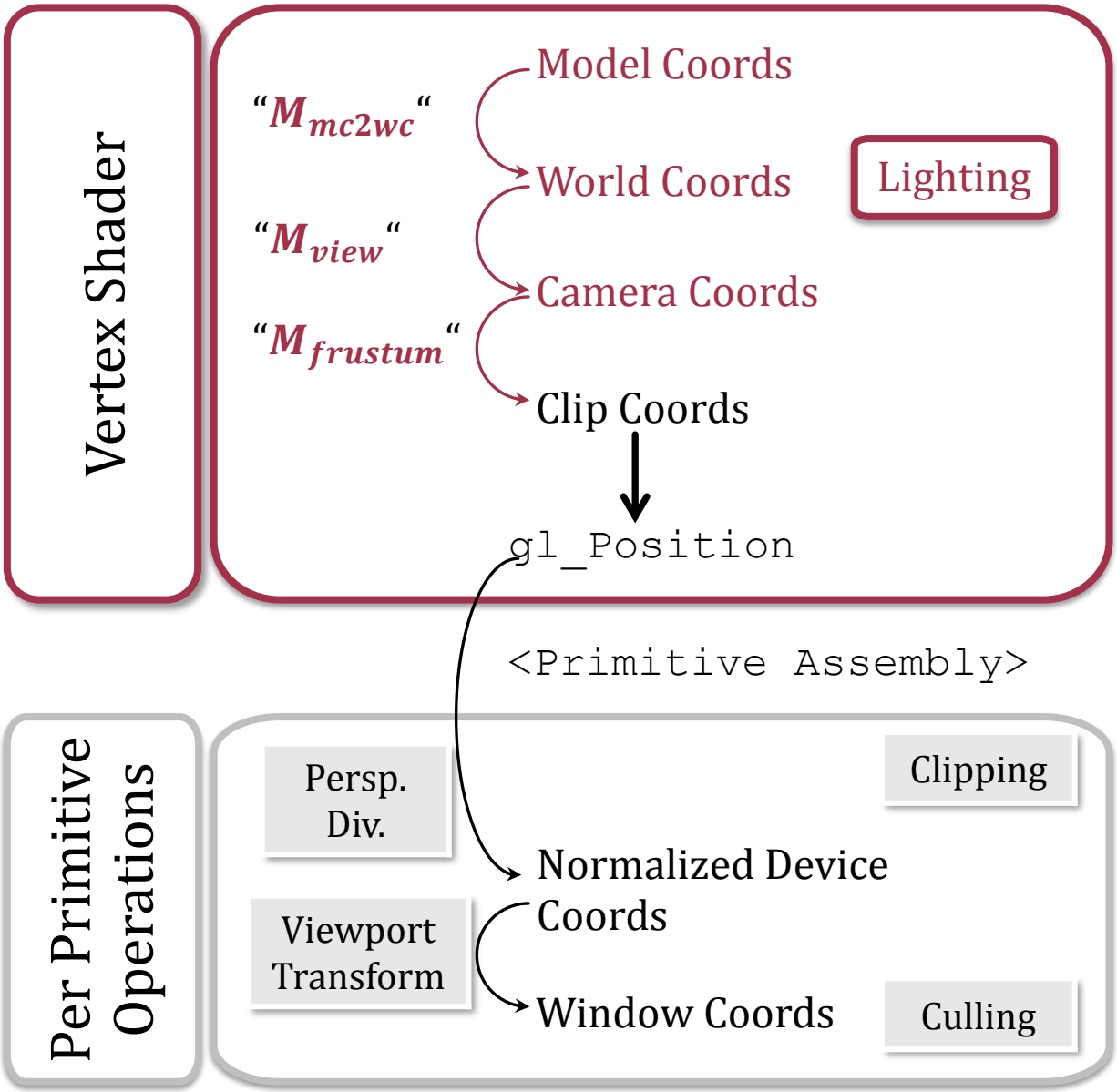
Computergrafik

Universität Osnabrück, Henning Wenke, 2012-06-11

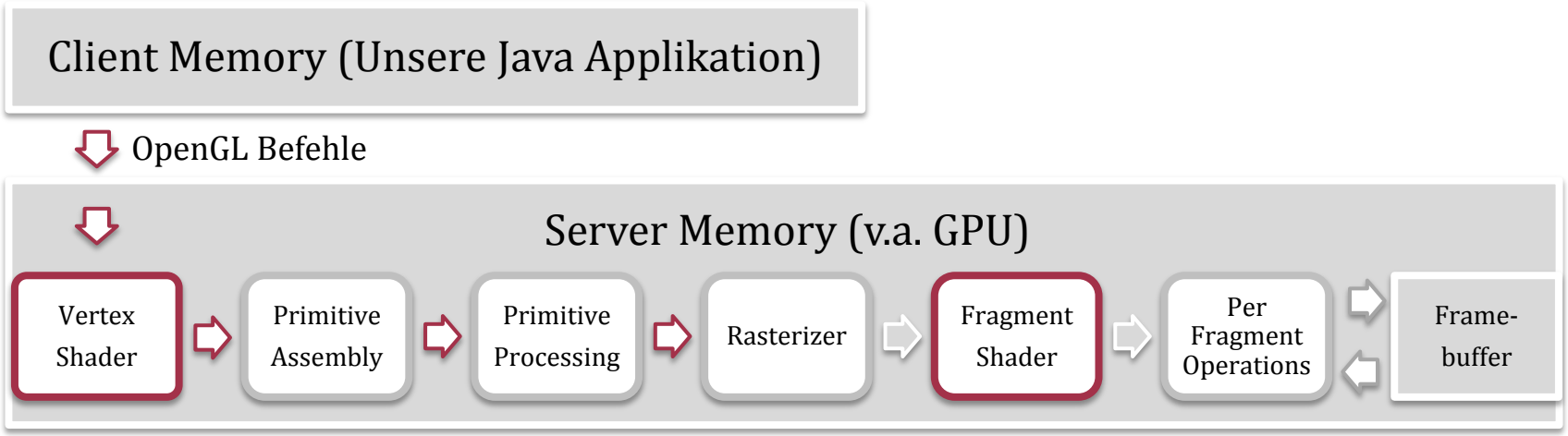
Was bisher geschah



Kapitel X:

Rasterization

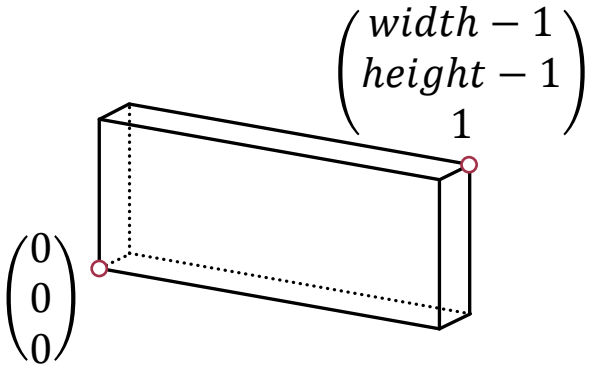
OpenGL Graphics Pipeline

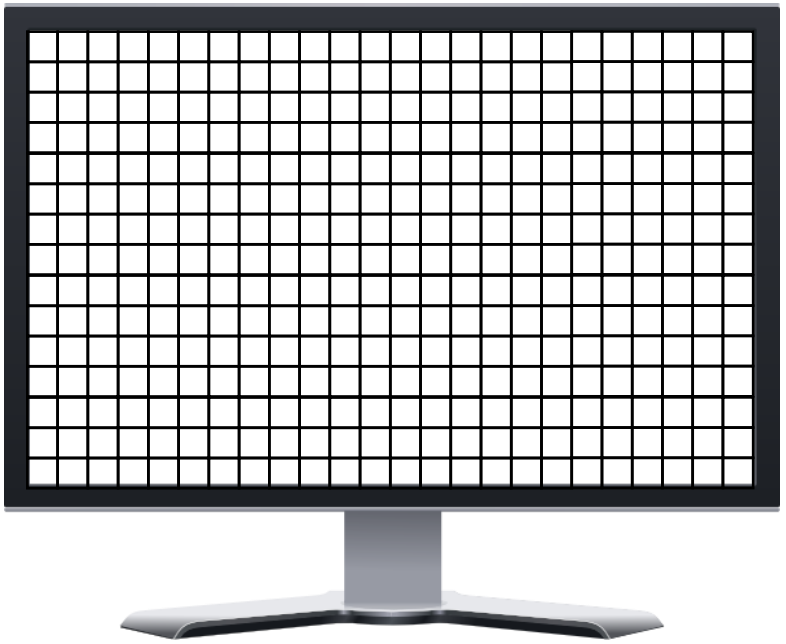


Legende

Vertices | Fragments | Pixel Data | Programmable Stage | Fixed Stage | Memory

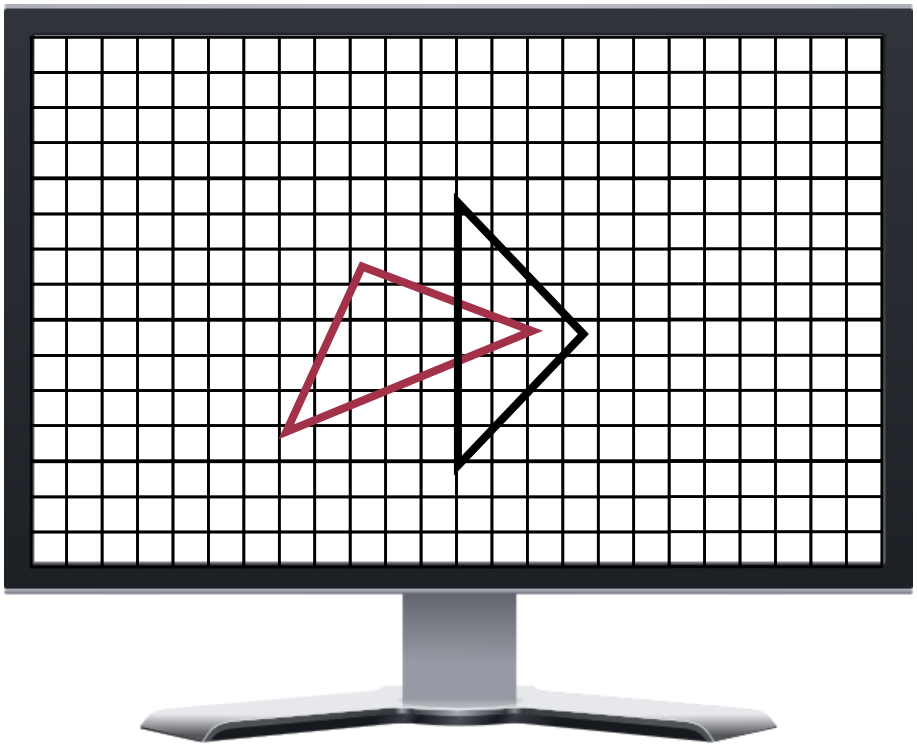
Window Coordinates vs. Monitor





Bestimme einzufärbende Pixel

- Pixel direkt einfärbbar?
  - Nein
- Pixel können von mehreren Primitives überlappt werden
- Primitives werden parallel und auch nacheinander verarbeitet
- Eindeutige Zuordnung unmöglich
- Was pro Pixel erzeugen?
  - „Pixelvorstufe“…
  - …mit Tiefeninformation:
  - Fragment



Vertices vs. Pixel: Koordinaten

Vertices: 1-3

- Koordinaten x & y
  - Window Coords
  - Kontinuierlich
  - Dreieck: 3 Koordinaten
- Koordinate z
  - Vorhanden

Pixel

- Koordinaten x & y
  - Window Coords
  - Diskret
  - Eine Koordinate
- Koordinate z
  - Undefiniert

Fragment

- Koordinaten x & y
  - Die des überlappten Pixels…
  - Berechnet aus den Koordinaten der Vertices
- Koordinate z
  - Interpolierter Wert für diese Pixelkoordinate

Vertices vs. Pixel: Farbe

Vertices: 1-3

- Farbe?
  - "Nein",
  - Keine Build-In VS-Out `gl_Color`

Pixel

- Farbe?
  - Immer

$\Longrightarrow$ Fragment kann initial keine Farbe erhalten

Vertices vs. Pixel: Eigene Daten

Vertices: 1-3

- Beliebige Daten, z.B.
  - Farbe
  - Normale
  - Deformierbarkeit
  - Position in WC
  - …
  - Nur wir kennen deren Bedeutung!

Pixel

- Beliebige Daten
  - Keine

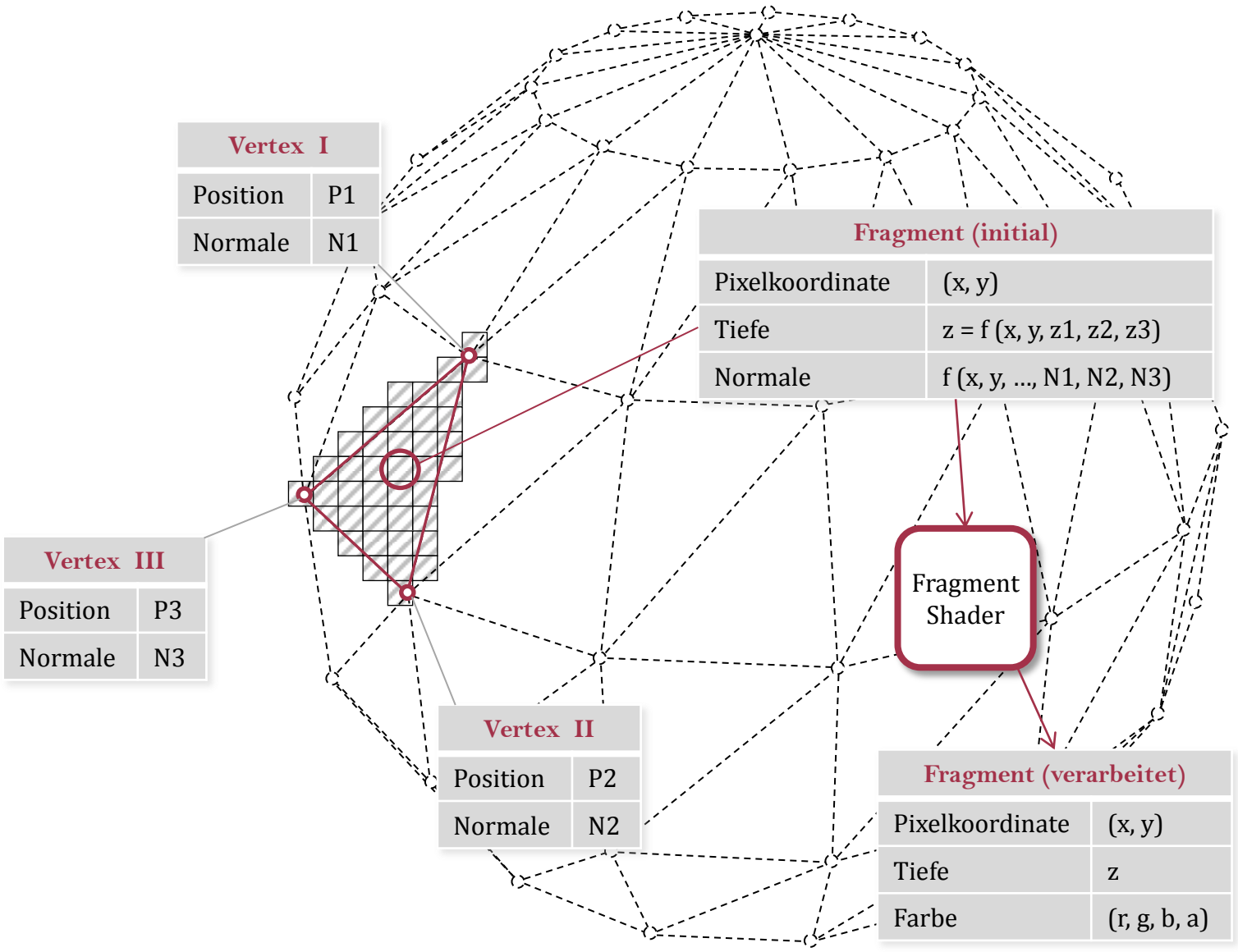
Fragment

- Erhält für seine Pixelposition interpolierte Vertexdaten
- Hier:
  - Farbe($x_{pixel}, y_{pixel}, Farbe_0, Farbe_1, Farbe_2$),
  - $Normale\ (x_{pixel}, y_{pixel}, Normale_0,\ Normale_1, Normale_2)$, …

3 Stufen im Leben eines Fragments

- Initial
  - Pixelposition
  - Tiefeninformation
  - Für diese Position interpolierte Daten
  - Fragment-Pixel Verhältnis?
  - $(0 - n)$ zu 1
- Verarbeitung (typisch)
  - Berechne, basierend auf Fragmentdaten, eine Farbe
  - Fragment Shader (morgen)
- Per Fragment Ops?
  - Lege, basierend auf allen zu einem Pixel gehörigen Fragments dessen Farbe fest

Beispiel


Vertex I
Position P1
Normale N1
Fragment (initial)
Pixelkoordinate (x, y)
Tiefe z = f (x, y, z1, z2, z3)
Normale f (x, y, ..., N1, N2, N3)
Vertex III
Position P3
Normale N3
Fragment Shader
Vertex II
Position P2
Normale N2
Fragment (verarbeitet)
Pixelkoordinate (x, y)
Tiefe z
Farbe (r, g, b, a)

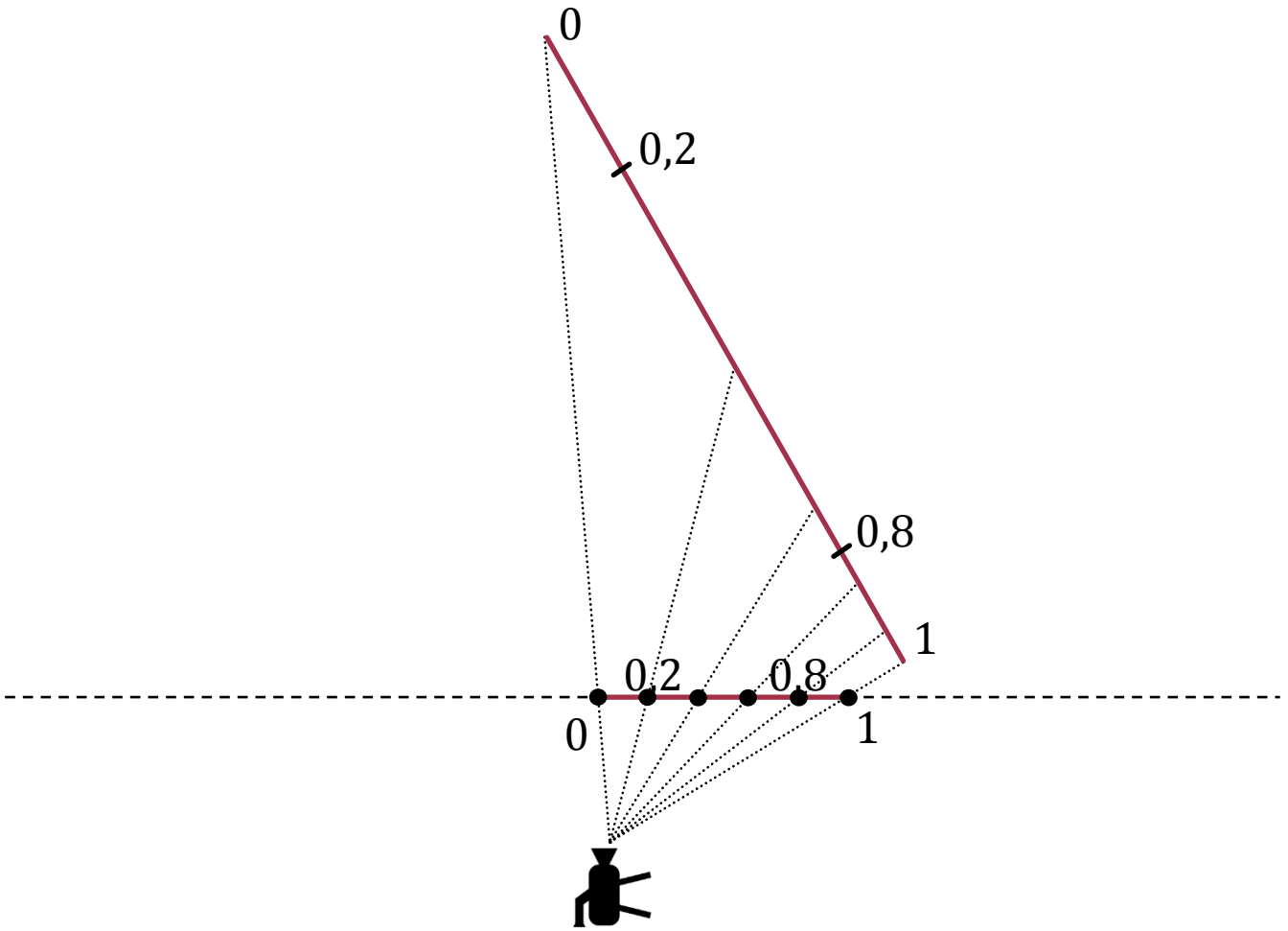
Definitionen

- Rasterizer
  - Algorithmus
  - Bestimmt von Primitiv überlappte Pixel
  - Erzeugt für jeden davon ein Fragment
- Fragment
  - Datenstruktur
  - Informell: Vorstufe eines Pixels mit Tiefeninformation
  - Erhält initial für diese Pixelposition interpolierte Daten der Vertices des Primitivs

X.1

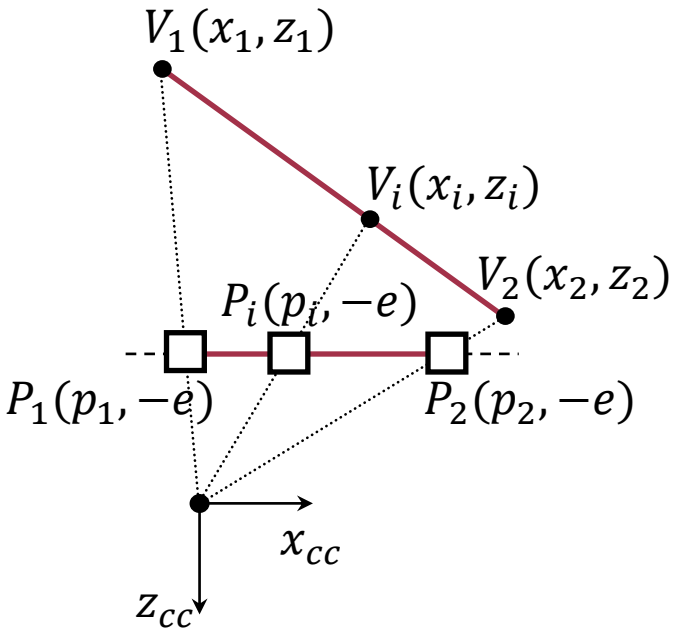
Einschub: Perspektivische Vertex Attribut Interpolation

Problemstellung



Depth Interpolation I

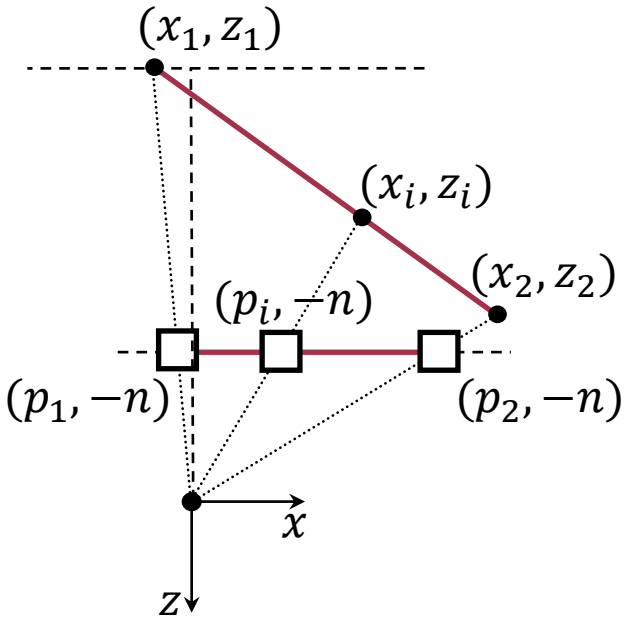
- Erinnerung: Z in NDC:
  - $z_{ndc} = -\frac{2nf}{f-n}\left(-\frac{1}{z_{cc}}\right) + \frac{f+n}{f-n}$
  - $= const_1 \frac{1}{z_{cc}} + const_2$
- Gegeben:
  - Linie definiert durch Punkte V_1 und V_2
  - Projiziert auf: Linie definiert durch Punkte P_1 und P_2 in xy-Bildebene bei $z = -e$
- Zu zeigen:
  - Für einen Punkt P_i auf dieser Ebene kann $1/z_{cc}$ linear zwischen P_1 und P_2 interpoliert werden und
  - Ergebnis mit Interpoliertem Punkt V_i zwischen V_1 und V_2 identisch
  - Und damit perspektivisch korrekt



Dann können in NDC bzw. Window Coords z-Werte linear zwischen beliebigen Punkten interpoliert werden

Depth Interpolation II

- Geradengleichung $z = mx + b$
- Strahlensatz $\frac{p}{x} = \frac{-n}{z} \Rightarrow x = \frac{pz}{-n}$
- Einsetzen in Geradengleichung $z = \frac{mpz}{-n} + b$
- Nach $\frac{1}{z}$ auflösen $\Rightarrow \frac{1}{z} = \frac{1}{b} + \frac{mp}{bn}$



- Setze $z = z_i$, $p = (1-t)p_1 + t \cdot p_2$ ein

$$\Rightarrow \frac{1}{z_i} = \frac{1}{b} + \frac{m \cdot ((1-t) \cdot p_1 + t \cdot p_2)}{b \cdot n} \Rightarrow \frac{1}{z_i} = \left(\frac{m \cdot p_1}{b \cdot n} + \frac{1}{b}\right)(1-t) + \left(\frac{m \cdot p_2}{b \cdot n} + \frac{1}{b}\right) \cdot t$$

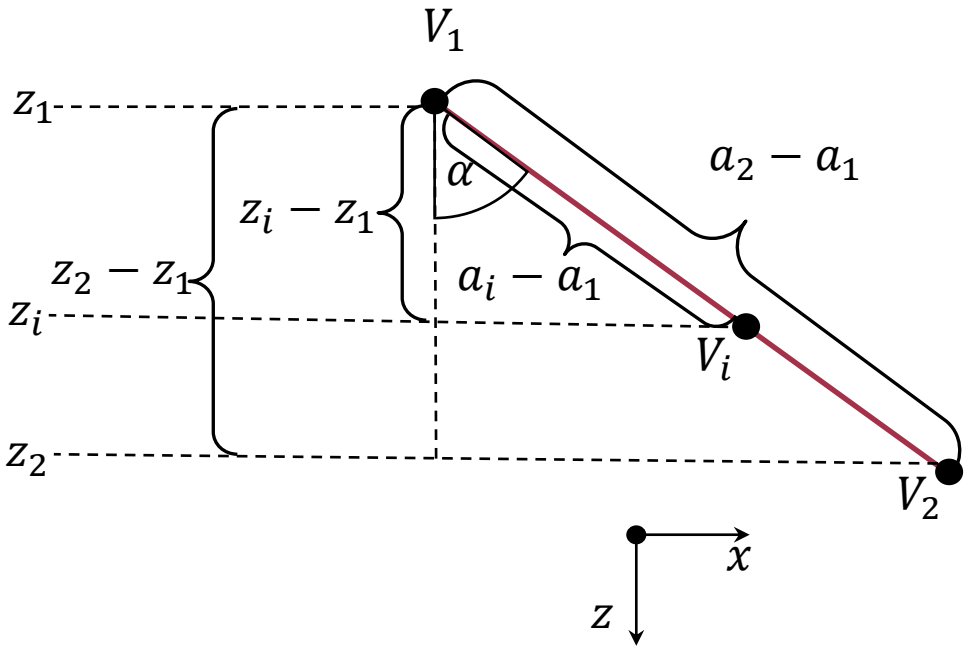
$$\Rightarrow \frac{1}{z_i} = \frac{1}{z_1}(1-t) + \frac{1}{z_2} \cdot t$$

$\Rightarrow$ Lineare Interpolation von $\frac{1}{z}$ in x-Richtung ist korrekt $\square$

(y analog)

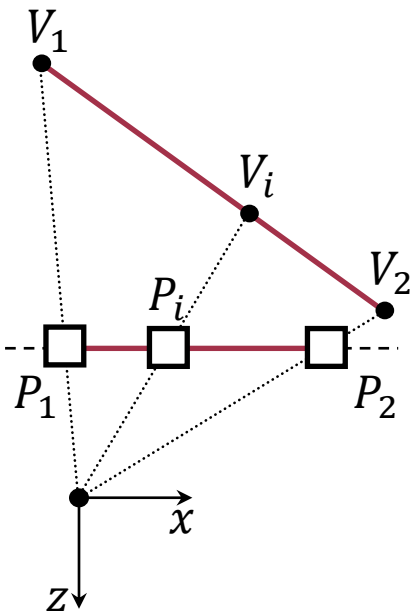
Attribute Interpolation I

- Gegeben Vertices mit Attribut a:
  $V_1: \{x_1, z_1, a_1, \dots\}$,
  $V_2: \{x_2, z_2, a_2, \dots\}$
- Gesucht: Interpoliertes Attribut
  $V_i.a$ bzw. $P_i.a$
- Idee: Beschreibe Interpolation der Attribute durch Verhältnis zur z-Interpolation
- Es gilt:
  - $\cos(\alpha) = \frac{a_i - a_1}{z_i - z_1} = \frac{a_2 - a_1}{z_2 - z_1}$
  - $\Leftrightarrow \frac{a_i - a_1}{a_2 - a_1} = \frac{z_i - z_1}{z_2 - z_1}$
- Hinweis:
  Interpolationsverhältnis gilt auch für mehrdimensionale Attribute

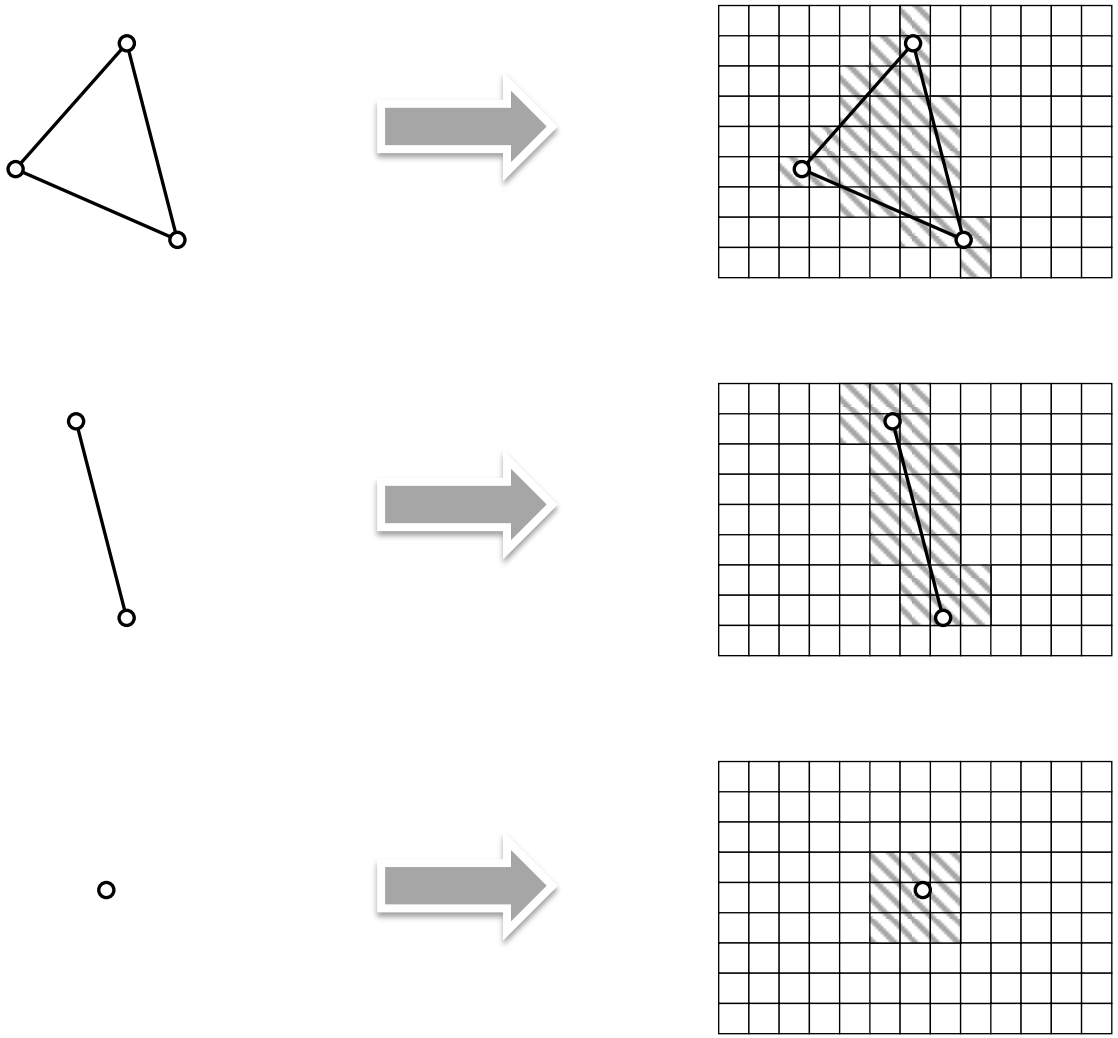


Attribute Interpolation II

- Gegeben: $V_1 : \{x_1, z_1, a_1, \dots\}, V_2 : \{x_2, z_2, a_2, \dots\}$
- Gesucht: $P_i.a$ bzw. $V_i.a$
- Wir erwarten:
  - $\frac{a_i - a_1}{a_2 - a_1} = \frac{z_i - z_1}{z_2 - z_1}$ (I)
- Z_i linear interpoliert zwischen P_1 und P_2:
  - $z_i = 1 / \left(\frac{1}{z_1}(1-t) + \frac{1}{z_2} t \right)$ (Folie 16)
- Einsetzen in (I) und etwas umformen:
  - $a_i \left(\frac{1}{z_1}(1-t) + \frac{1}{z_2} t \right) = \frac{a_1}{z_1}(1-t) + \frac{a_2}{z_1} t$
  - $\frac{a_i}{z_i} = \frac{a_1}{z_1}(1-t) + \frac{a_2}{z_2} t$
- a/z linear zwischen P_1 und P_2 interpolierbar
- $a_i = z_i \left(\frac{a_1}{z_1}(1-t) + \frac{a_2}{z_2} t \right)$



Überblick: Rastern der GL-Primitives

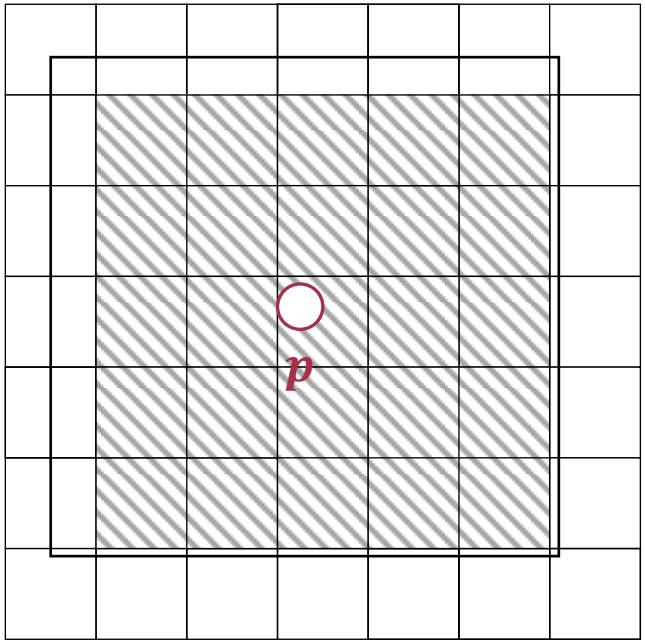


X.2

Rastern von Punkten

Erzeugen der Fragments

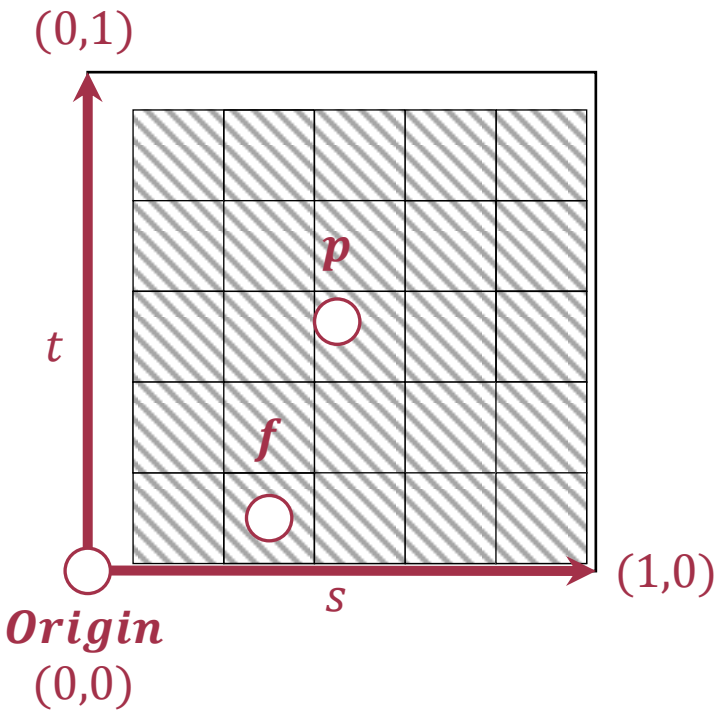
- Gegeben: Punkt mit:
  - Vertex $\boldsymbol{p}(p_x, p_y)$ als Mittelpunkt
  - Ausdehnung $size$ in Pixeln
- Ermittle $\boldsymbol{size} \cdot \boldsymbol{size}$ Pixel großen Bereich um $\boldsymbol{p}$
- Erzeuge Fragments für alle Pixel, deren Mittelpunkte überlappt sind
- Attribute?
  - Fragments erhalten genau die Attribute des Vertex



$size = 5{,}5$

Point Coordinates

- Erzeuge für Fragments PointCoords s, t ausgehend vom Ursprung des Einflussbereichs
- Mit dem Vertex $\boldsymbol{p}(p_x, p_y)$ und dem Fragment $\boldsymbol{f}(f_x, f_y)$ sind diese:
  - $s = \frac{1}{2} + \frac{(f_x + 0.5 - p_x)}{size}$
  - $t = \frac{1}{2} + \frac{(f_y + 0.5 - p_y)}{size}$
- Hinweis: Origin kann sich auch linksoben befinden



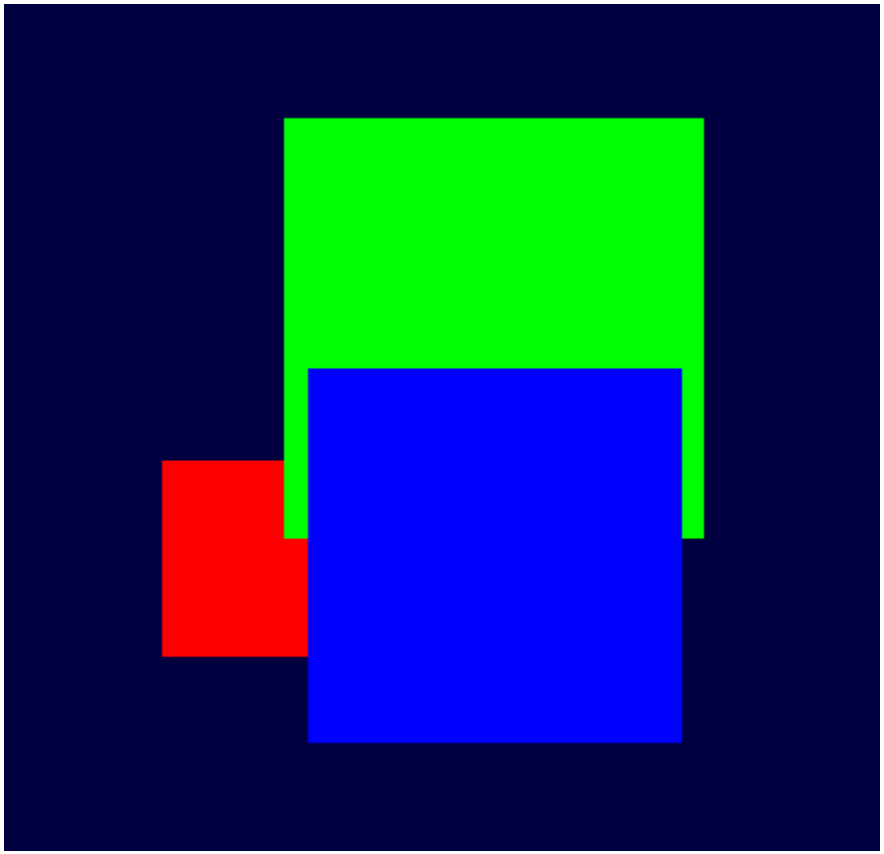
OpenGL Point Rasterizer State

```
// Aktivieren einer variablen Größe des Einflussbereichs für
// alle GL_POINTs
glEnable(GL_PROGRAM_POINT_SIZE);

// Dann kann diese als Build-In Variable, etwa im VS, gesetzt werden.
float gl_PointSize; // Breite und Höhe des Einflussbereichs in Pixeln
```

```
// Alternative: Setzen einer konstanten Größe für alle GL_POINTs
glDisable(GL_PROGRAM_POINT_SIZE); // Variable Größe deaktivieren
glPointSize(
  float size   // Breite und Höhe des Einflussbereichs in Pixeln
);
```

Demonstration

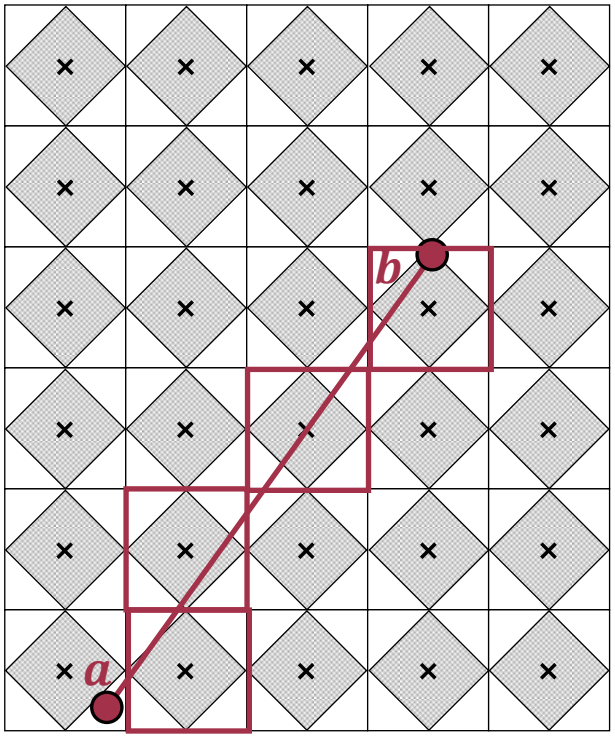


X.3

Rastern von Linien

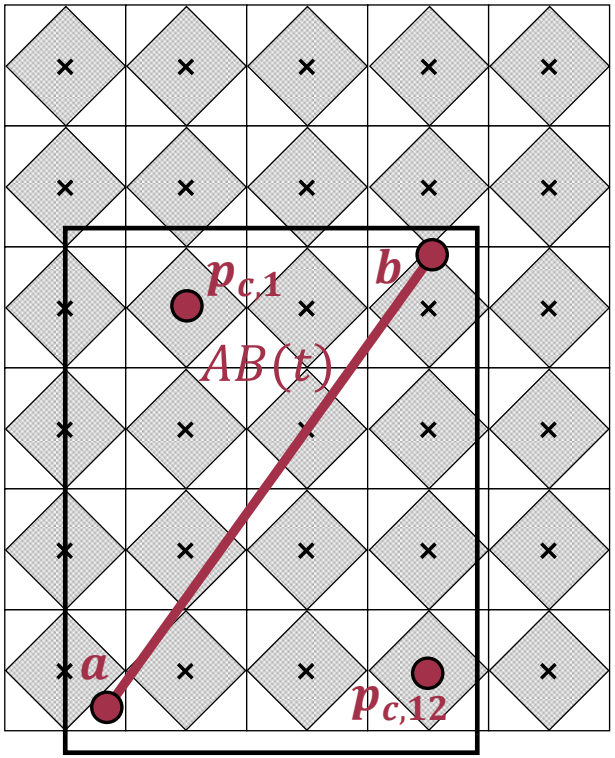
Erzeugen der Fragments (OpenGL Spec)

- Gegeben: Linie mit Eckunkten $\boldsymbol{a}, \boldsymbol{b}$
- Bestimme Mittelpunkte (x_p, y_p) der „Kandidatenpixel“ p_c
- Definiere $\forall p_c$ Bereich R_p
  - $R_p = \{(x, y) | \, |x - x_p| + |y - y_p| < \frac{1}{2}\}$
- Erzeuge Fragments für p_c f.d.g.:
  - Linie schneidet R_p
  - Punkt $\boldsymbol{b} \notin R_p$, da dieses auch für nächstes Liniensegment des Linienzuges erzeugt wird
- Breite $\neq 1$? Keine Vorgaben in der OpenGL Specification



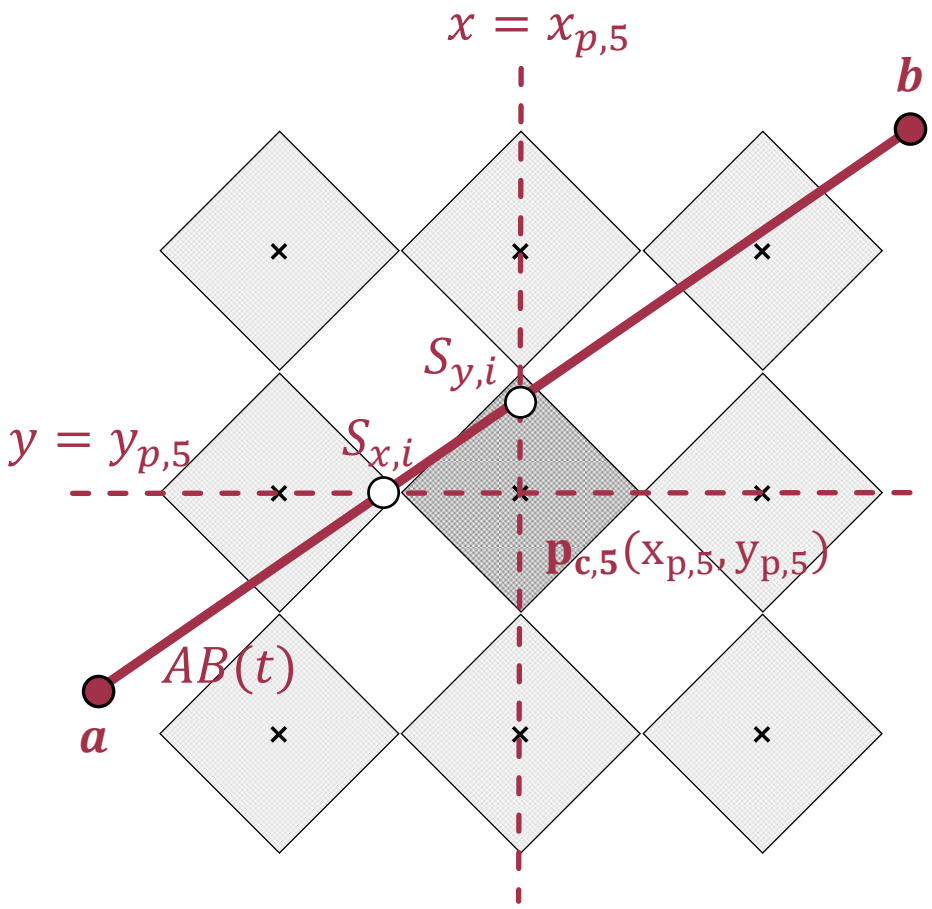
Erzeugen der Fragments (mögl. Impl. I)

- Berechne Linie von A nach B
  - $\mathrm{AB}(t) = \boldsymbol{a} + t \cdot (\boldsymbol{b} - \boldsymbol{a}), t \in [0,1]$
- Bestimme Kandidatenpixel $\boldsymbol{p_{c,i}}(x_{p,i}, y_{p,i})$
- Z.B. alle Pixel, die in rechteckigem Bereich um $\boldsymbol{a}$ und $\boldsymbol{b}$ liegen



Erzeugen der Fragments (mögl. Impl. II)

- Beobachtung: Pixelausdehnung in x-Richtung bei $y = y_{p,i}$ und in y-Richtung bei $x = x_{p,i}$ am größten
- Führe $\forall \boldsymbol{p_{c,i}}$ folgende Schritte aus:
- Bestimme, sofern existent, Schnittpunkte $S_{y,i}$ von $AB(t)$ mit senkrechter Gerade bei $x_{p,i}$ und $S_{x,i}$ mit waagerechter Gerade bei $y_{p,i}$
- Teste, ob $S_{x,i}$ oder $S_{y,i} \in R_p$ ist
  - Ja: Erzeuge Fragment für dieses $\boldsymbol{p_{c,i}}$
  - Nein: Teste Sonderfall



Erzeugen der Fragments (mögl. Impl. III)

- Sonderfall: Linie endet in einem Pixelviertel und schneidet daher Achsen nicht
- In diesem Fall liegt/liegen $\boldsymbol{a}$ und/oder $\boldsymbol{b}$ in $R_{p,i}$
- Es muss demnach das Fragment zu einem Kandidatenpixel $\boldsymbol{p_{c,i}}$ erzeugt werden, wenn eine Bedingung erfüllt ist aus:
  - $S_{x,i}$ existiert und $S_{x,i} \in R_p$
  - $S_{y,i}$ existiert und $S_{y,i} \in R_p$
  - $\boldsymbol{a} \in R_p$
  - $\boldsymbol{b} \in R_p$