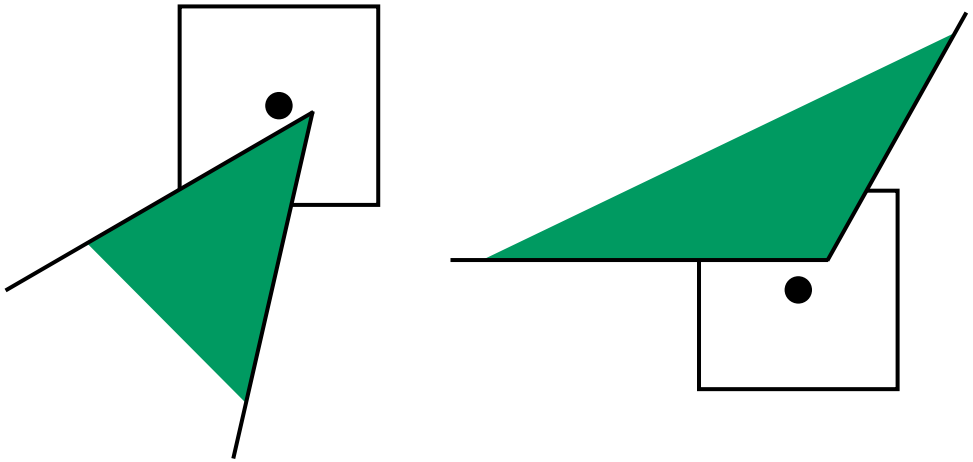
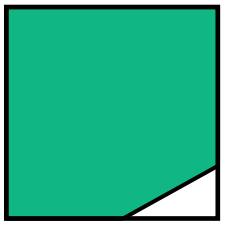
Anti-Aliasing von Polygonen

- Gupta-Sproull-Algorithmus funktioniert nicht so gut:
  1. Flächenanteil an Ecken nicht so einfach

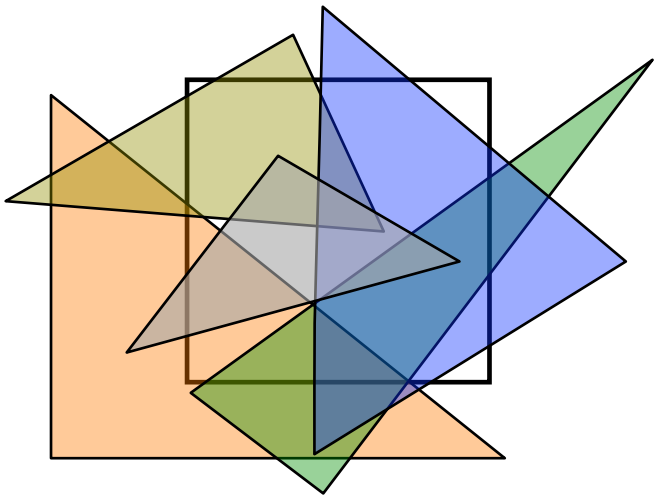


  2. Was tun bei mehreren Polygonen?

- Dieser Fall ist einfach:



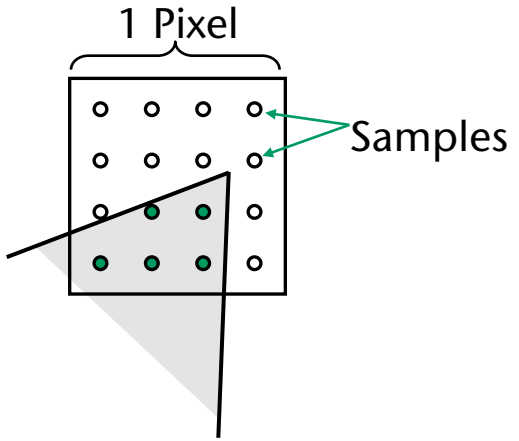
- Wie verfährt man in diesem Fall:

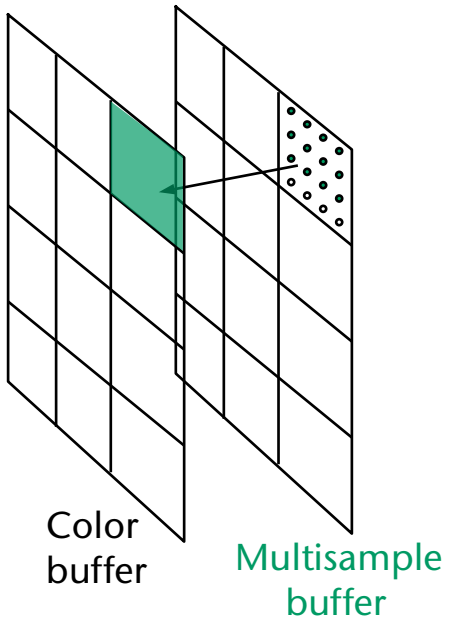


- Ist zur Zeit nur sehr sehr schwer zu lösen

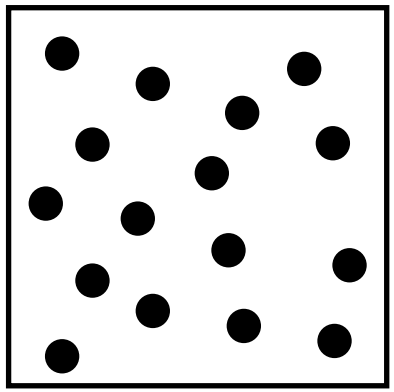
Super-Sampling („Multisampling“)

- Sample Polygon nicht nur an Pixel-Zentrum, sondern an *n·n* Samples
- Approximiere damit den Flächenanteil des Pixels, der vom Polygon überdeckt wird
- Verwalte pro Pixel eine sog. *Coverage Mask* im *Multisample-Buffer*
- Achtung: Color-Buffer liegt i.A. nicht in höherer Auflösung vor!
  - Hardware berechnet automatisch beim Schreiben eines Pixels die Coverage-Mask





Berechnung der Farbe für ein Pixel vom Sample

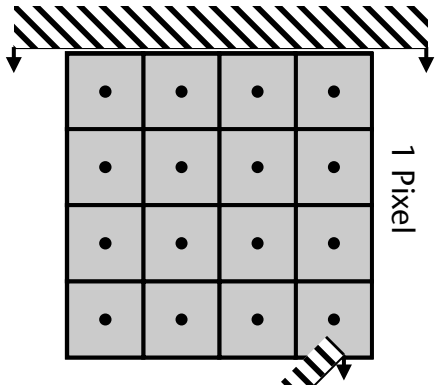


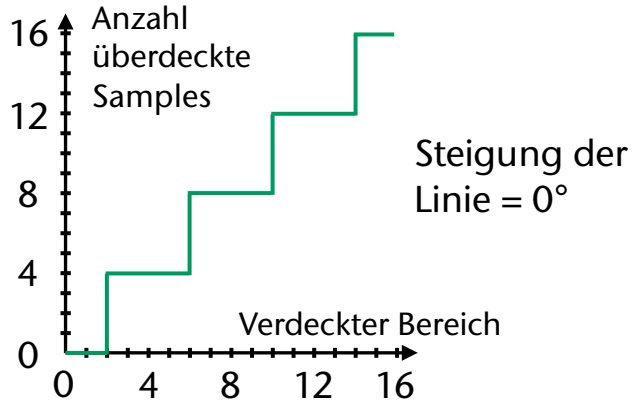
$$p(x,y) = \sum_{i=1}^{n} w_i c(i,x,y) \;, \quad \sum w_i = 1$$

- w_i sind die Gewichte [0,1], abhängig vom verwendeten Filter
- $c(i,x,y)$ Farbe des Sample i im Pixel
- $p(x,y)$ ist die Farbe des Pixel

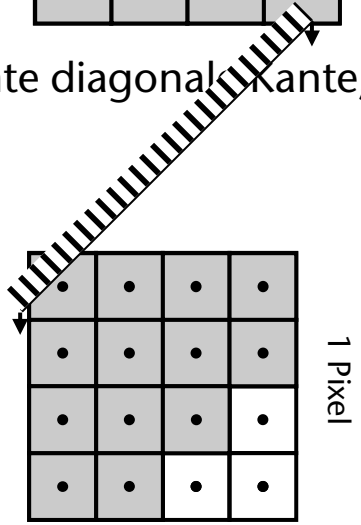
Probleme bei größeren, regelmäßigen Patterns

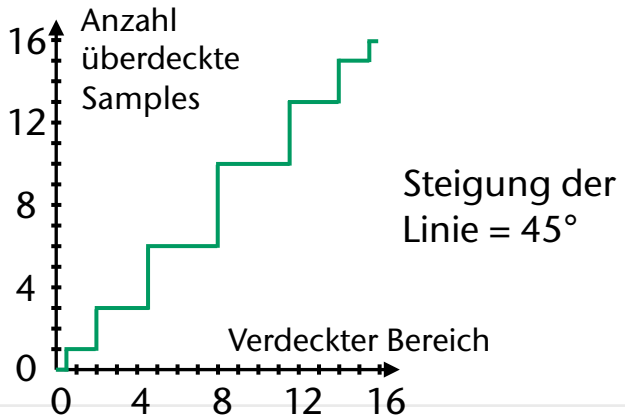
1. Am Sample-Muster ausgerichtete (horizontale, vertikale, diagonale) Kanten:
   - Betrachte horizontale Polygonkanten, die langsam nach unten gleiten



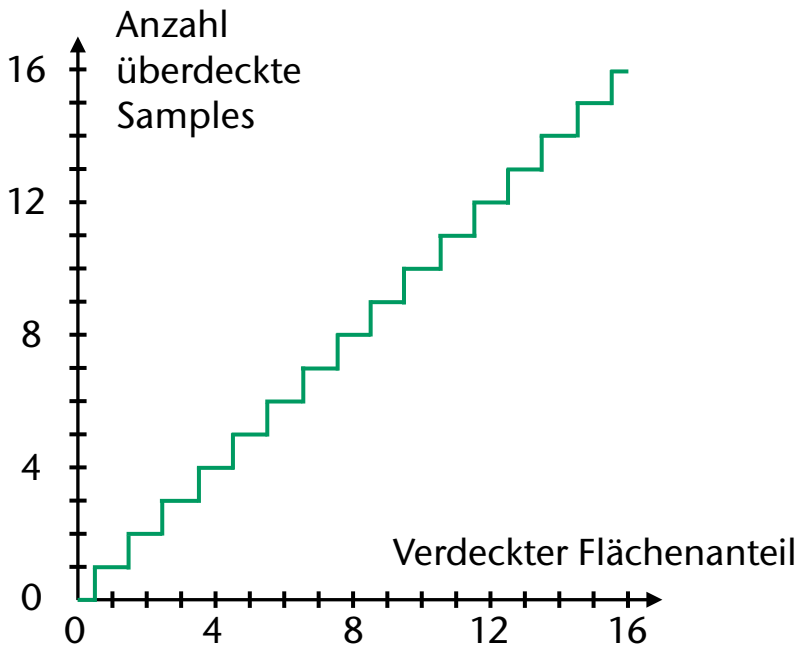


   - Betrachte diagonale Kante, die langsam nach unten gleitet



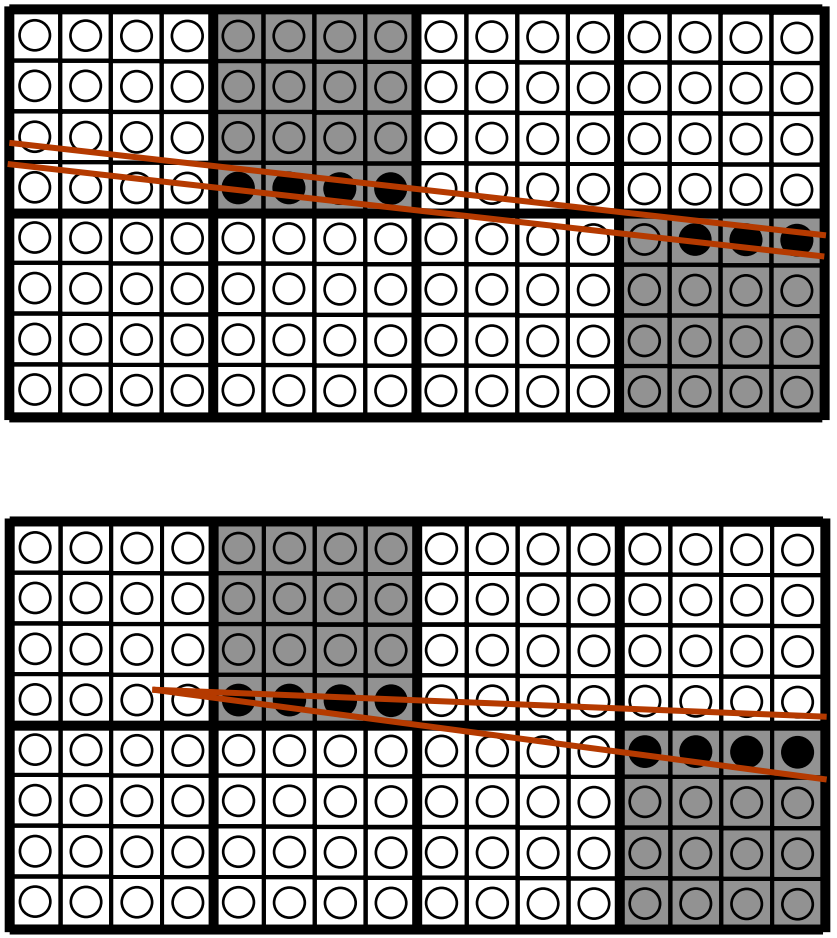


- Was man in allen Fällen gerne hätte, unabhängig von der Steigung (*slope*)

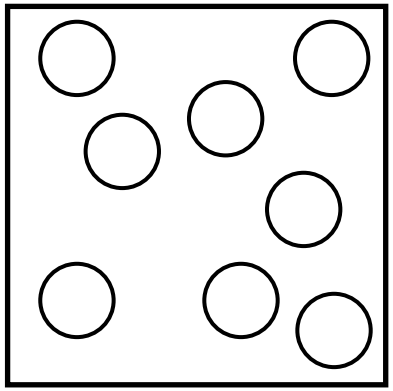


2. Sehr schmale Dreiecke / Rechtecke:

- Lücken (bei manchen Pixeln werden gar keine Samples "getroffen")
- Wenn das Objekt sich langsam bewegt, dann "springen" die Pixel, oder es verschwindet zwischendurch ganz



- Simple Lösung: stochastisches Super-Sampling

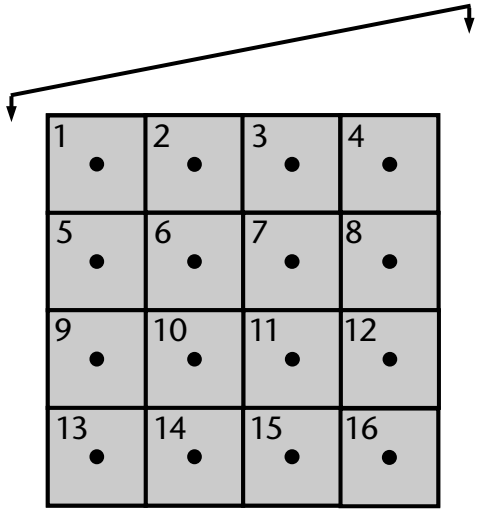


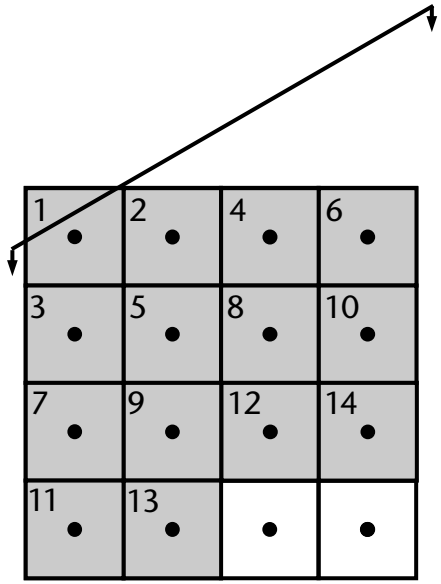
- Aber: dabei wird eigentlich nur ein Artefakt (Aliasing) durch ein anderes (Rauschen) ersetzt

Exact Area Subpixel Masks [Schilling 1991]

1. Beobachtung: es gibt nur eine relativ kleine Anzahl von möglichen Coverage-Masks, da es immer Halbebenen sind
2. Beobachtung: für eine Halbebene mit bestimmter Steigung gibt es nur eine relativ kleine Anzahl von verschiedenen Coverage Masks

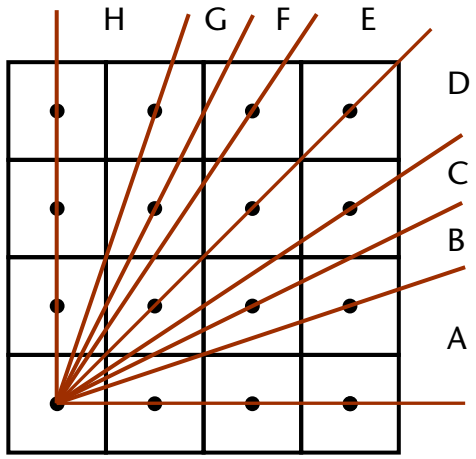
- Beispiele:





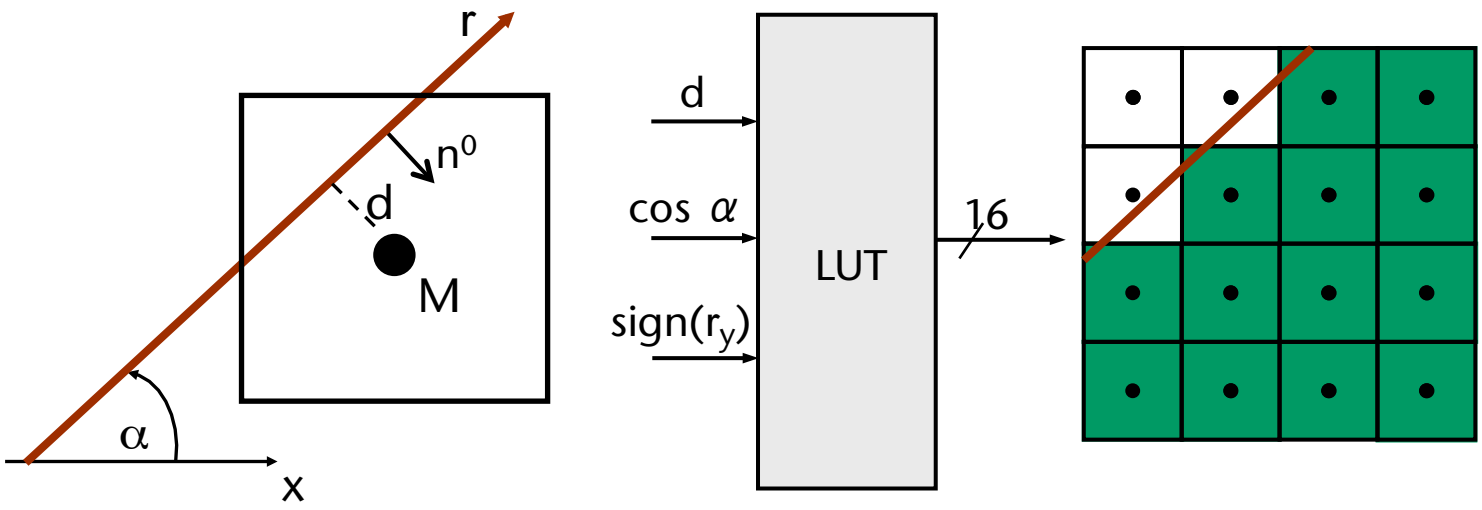
3. Beobachtung: esgibt nur eine relative kleine Teilmenge von Permutationen aus {0, ..., 15}, gemäß denen die Samples "angehen", wenn eine Halbebene über das Pixel streicht

- Fazit: teile alle Geradensteigungen in Äquivalenzklassen ein, wobei alle diejenigen Steigungen in derselben Klasse sind, wenn sie dieselbe Permutation hervorrufen.



Das Verfahren in Hardware

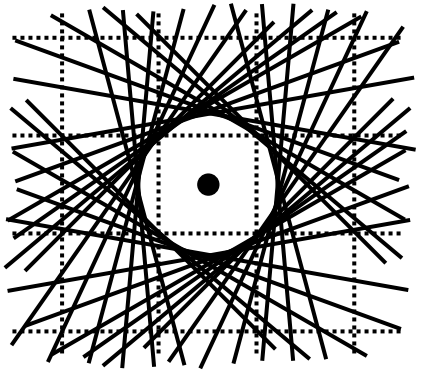
- Precomputation → Lookup-Table



- OBdA ist Steigung im Bereich von - 90° ... + 90°

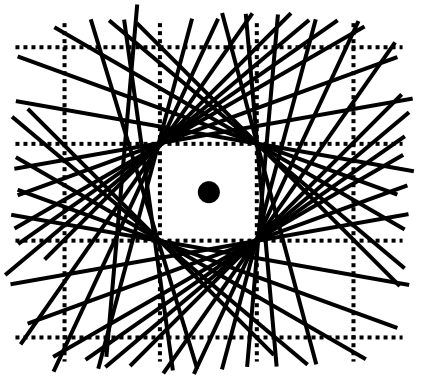
- Effiziente Berechnung der Distanz d:
  - Eigentlich benötigt man den Euklidischen Abstand:

$$d = M \cdot \mathbf{n}^\circ - c = M \cdot \frac{\mathbf{n}}{\sqrt{n_x^2 + n_y^2}} - c$$



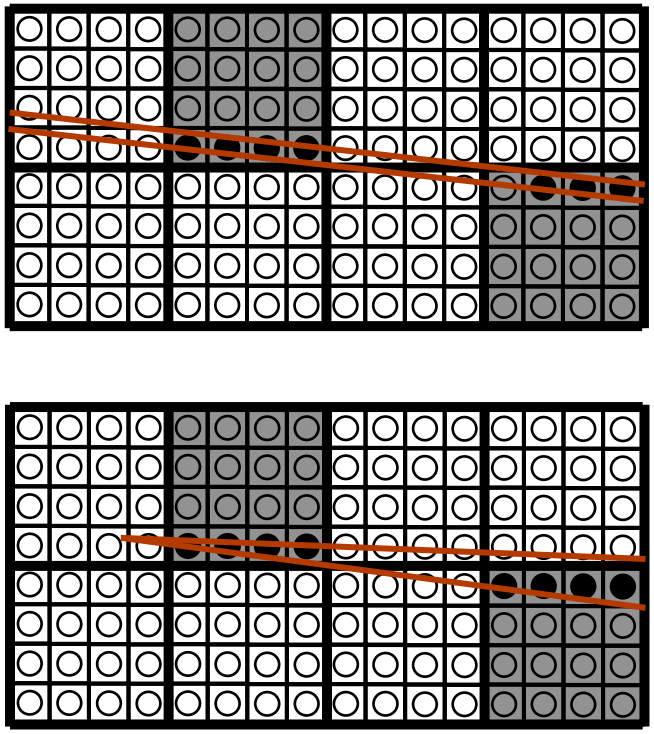
  - Da man eine Lookup-Table hat, kann man aber auch eine andere Norm für die Länge eines Vektors nehmen, z.B.

$$d = M \frac{\mathbf{n}}{|n_x| + |n_y|} - c$$



- Aber eventuell hat man die Normale sowieso mit euklidischer Norm normiert, weil man im Algorithmus von Pineda baryzentrische Koordinaten braucht

Resultate



Ohne Supersampling

Primitives Supersampling

Supersampling wie eben beschrieben