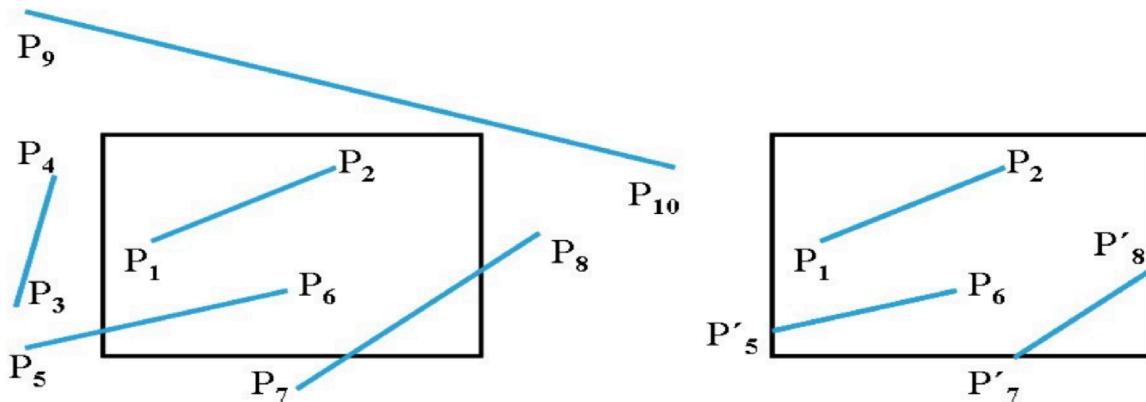


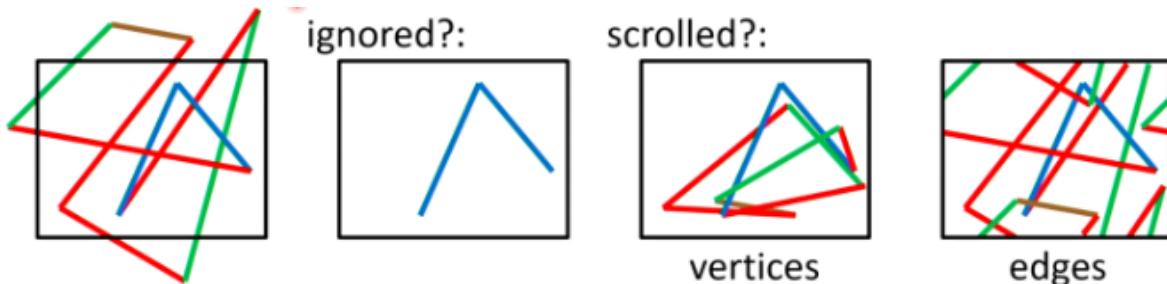
7. Clipping und Antialiasing

Line Clipping:

- **Clipping** bezeichnet das **Abschneiden von Bildteilen**, die außerhalb des Darstellungsfensters liegen.
- Clipping wird durchgeführt, um **unnötige nachfolgende Umformungen** von nicht sichtbaren Teilen zu vermeiden:
 1. **Clipping in Weltkoordinaten:**
 - Analytische Berechnung zum frühestmöglichen Zeitpunkt.
 2. **Clipping in Clipkoordinaten:**
 - Analytische Berechnung an **achsenparallelen Grenzen** (einfacher).
 3. **Clipping bei der Rasterkonversion:**
 - Clipping erfolgt innerhalb des Algorithmus, der ein **Grafikprimitiv** in **Punkte umwandelt**.
- Clipping ist eine sehr **häufige Operation**, daher muss es **einfach und schnell** sein.



Wichtige Fragen: Was soll abgeschnitten werden?



Clippen von Linien: Cohen-Sutherland-Verfahren

- Algorithmen zum **Clippen von Linien** nutzen die Tatsache aus, dass jede Linie in einem rechteckigen Fenster höchstens **einen sichtbaren Teil** besitzt.
- Wichtige Grundprinzipien der **Effizienz**:

1. Häufige einfache Fälle früh eliminieren.
2. Teure Operationen wie Schnittpunkt-Berechnungen vermeiden.
- Ein einfaches Linienculling könnte so aussehen:

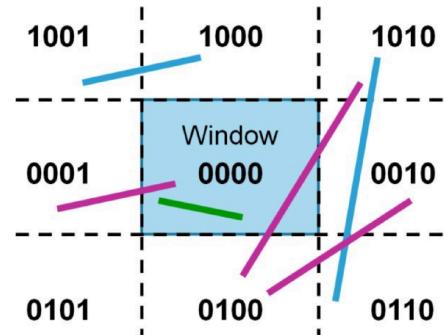
```

1 for endpoints (x0,y0), (xend,yend)
2 intersect parametric representation
3   x = x0 + u * (xend - x0)
4   y = y0 + u * (yend-y0)
5 with window borders:
6   intersection ⇔ 0 < u < 1

```

Der Cohen-Sutherland-Algorithmus klassifiziert zuerst die Endpunkte einer Linie hinsichtlich ihrer Lage zum Clippingfenster: oben, unten, links, rechts, und codiert diese Information in 4 Bit. Nun kann man schnell überprüfen:

1. OR der beiden Codes = 0000 ⇒ Linie ganz sichtbar
2. AND der beiden Codes ≠ 0000 ⇒ Linie ganz unsichtbar
3. andernfalls mit einer relevanten Fensterkante schneiden, und den weggeschmierten Punkt durch den Schnittpunkt ersetzen.
GOTO 1.



- Schnittpunktberechnungen mit vertikalen Fensterkanten:

- Für die linke Kante:

$$y = y_0 + m(x_{wmin} - x_0) \quad y = y_0 + m(x_{wmin} - x_0)$$

- Für die rechte Kante:

$$y = y_0 + m(x_{wmax} - x_0) \quad y = y_0 + m(x_{wmax} - x_0)$$

- Schnittpunktberechnungen mit horizontalen Fensterkanten:

- Für die untere Kante:

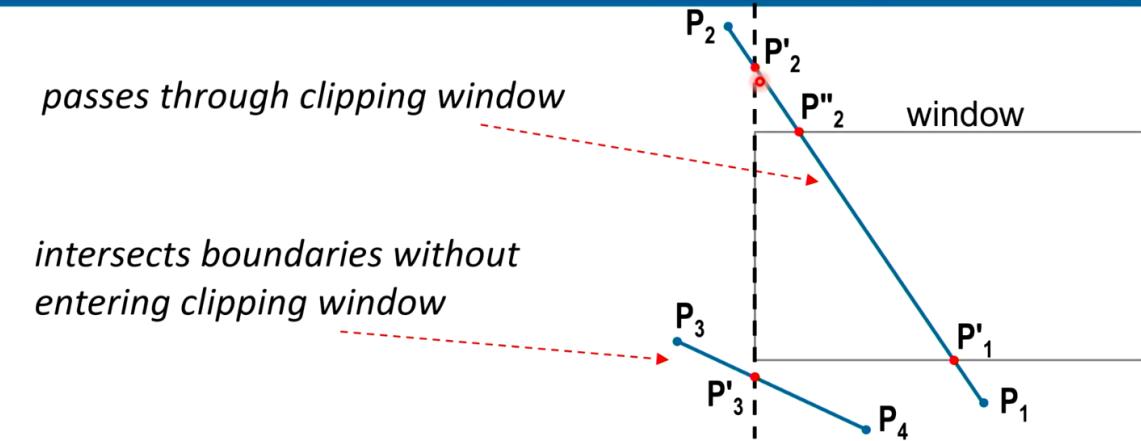
$$x = x_0 + \frac{(y_{wmin} - y_0)}{m} x = x_0 + m(y_{wmin} - y_0)$$

- Für die obere Kante:

$$x = x_0 + \frac{(y_{wmax} - y_0)}{m} x = x_0 + m(y_{wmax} - y_0)$$

- Punkte, die genau auf den Fensterkanten liegen, gelten als innerhalb des Fensters.
- Es sind höchstens 4 Schleifendurchläufe erforderlich, da es höchstens 4 Schnittpunkte gibt.
- Effizienz: Schnittpunktberechnungen werden nur durchgeführt, wenn sie wirklich notwendig sind.
- Clipping von Kreisen:
 - Ähnliches Verfahren wie für Linien.
 - Kreise können beim Clipping in mehrere Teile zerfallen.

Cohen-Sutherland Line Clipping



vertical: $y = y_0 + m(xw_{\min} - x_0)$,
 horizontal: $x = x_0 + (yw_{\min} - y_0)/m$,

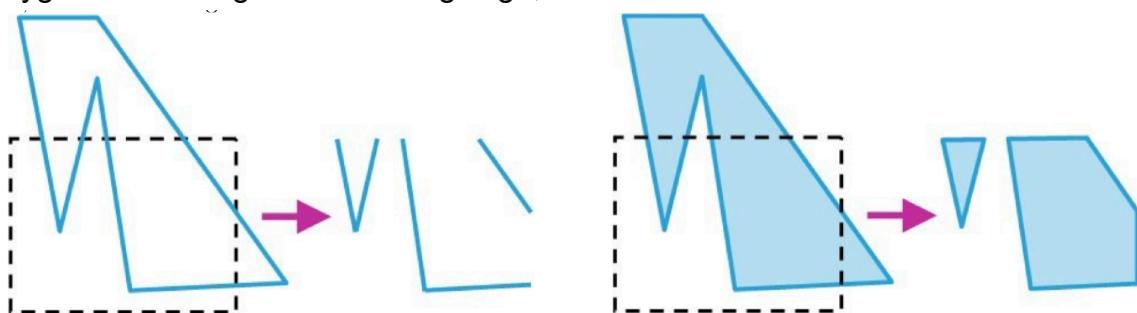
Werner Purgathofer

13



Polygon Clipping:

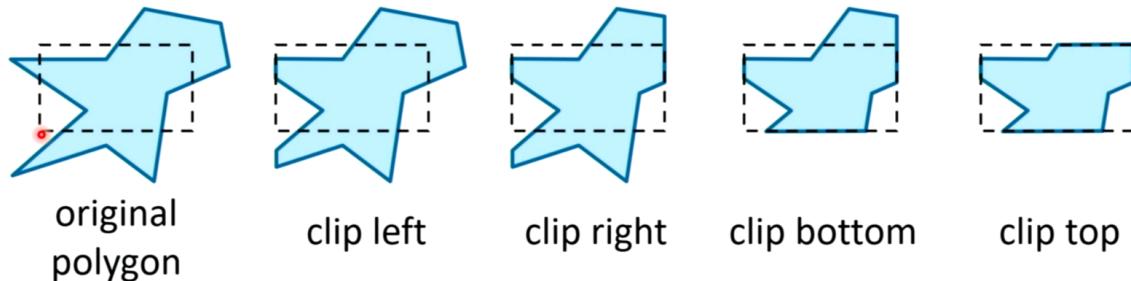
- **Polygon-Clipping** muss sicherstellen, dass nach dem Clipping ein **gültiges Polygon** entsteht, auch wenn der Clipping-Vorgang mehrere Teile erzeugt.
- Beispiel:
 - **Linien-Clipping-Algorithmus:**
 - Das Ergebnis zeigt ein **unvollständiges Polygon**, bei dem nicht mehr erkennbar ist, was innen und was außen liegt.
 - **Korrekte Polygon-Clipping-Verfahren:**
 - Das Polygon zerfällt in **mehrere Teile**, die alle korrekt **gefüllt** werden können.
- **Wichtig:** Auch wenn mehrere Teile entstehen, müssen alle Teile des resultierenden Polygons **korrekt** gefüllt und als gültige, sichtbare Bereiche behandelt werden.



Bei einfachem Linien Clipping könnten Linien zurückkommen deshalb gibt es extra Verfahren für das Polynomclipping:

Sutherland-Hodgman Polygon Clipping

processing polygon boundary as a whole against each window edge
 → output: list of vertices



clipping a polygon against successive window boundaries

Man zerlegt das hier in **4 Schritte** die meist rekursiv aufgerufen werden

Clippen von Dreiecken:

- **Geometrische Daten** bestehen in der Praxis häufig nur aus **Dreiecken**. Der **Renderingprozess** hat dabei kein Wissen mehr über den Zusammenhang der Dreiecke und behandelt diese als eine „**Triangle Soup**“ (Dreiecks-Suppe).
- **Wichtig:**
 - Beim Clipping von Dreiecken muss immer darauf geachtet werden, dass nur Dreiecke entstehen – keine anderen Primitives.
- Beim **Clippen eines Dreiecks** gegen eine Kante gibt es vier mögliche Fälle:
 1. In einigen Fällen kann auch ein **Viereck** entstehen.

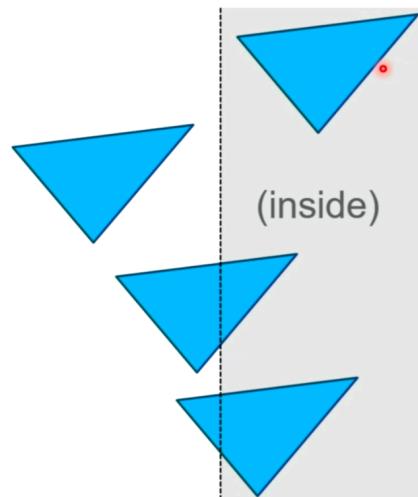
Clipping of Triangles

often b-reps are “triangle soups”

clipping a triangle → triangle(s)

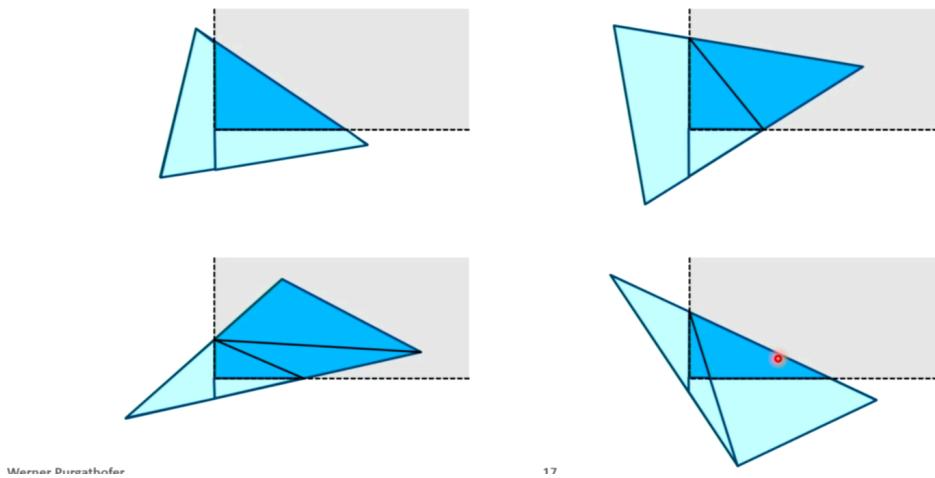
4 possible cases:

- (1) inside
- (2) outside
- (3) triangle
- (4) quadrilateral → 2 triangles



2. Das Viereck muss sofort in **zwei Dreiecke** zerteilt werden, um die Weiterverarbeitung zu ermöglichen.

corner cases need no extra handling!



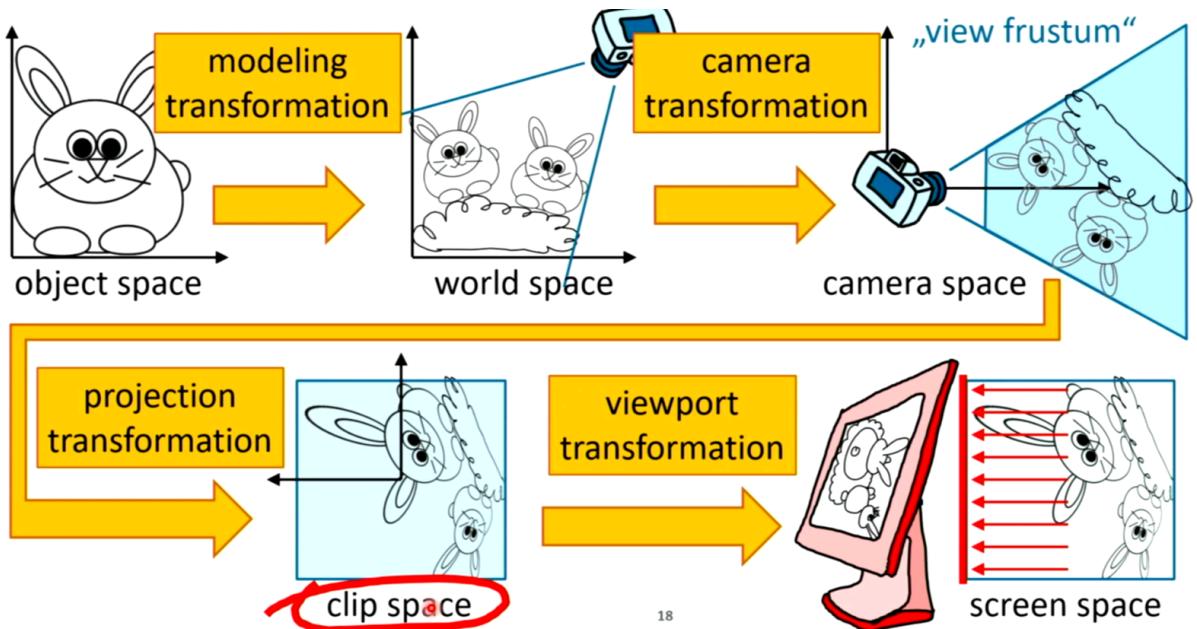
Warner Dursthafer

17

- An den **Ecken des Clip-Fensters** kann es vorkommen, dass mehr Dreiecke erzeugt werden, als tatsächlich notwendig wären (siehe Beispiel links), aber dies wird durch die **Einfachheit des Algorithmus** mehr als kompensiert.

Clipping in Clipkoordinaten:

- Clip-Space:**
 - Die Begrenzungsflächen des **View-Frustums** sind achsenparallel (d.h., die Grenzen sind bei $x = \pm 1$, $y = \pm 1$, $z = \pm 1$).
 - Dies vereinfacht die Feststellung, ob ein Punkt **innerhalb oder außerhalb** des Frustums liegt, da es nur einen **einfachen Vergleich** zwischen zwei Zahlen erfordert.



18

- Clipping vor der Homogenisierung:**

- Um zu vermeiden, dass Punkte hinter dem Kamerapunkt projiziert werden, wird das Clipping schon **vor der Homogenisierung** der Punktkoordinaten durchgeführt.
- Dabei wird an den **Ebenen** $x = \pm h$, $y = \pm h$, $z = \pm h$ geclipppt (was ebenso einfach ist).

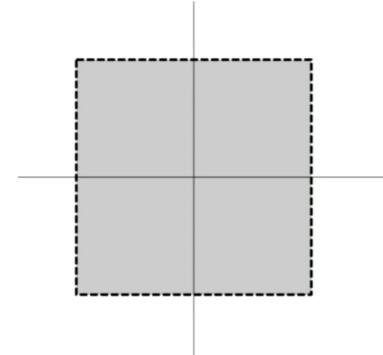
- Vorteile:**

1. Punkte, die hinter dem Kamerapunkt liegen, werden **nicht projiziert**.
2. **Ersparnis der Homogenisierungsdivision** für Punkte, die außerhalb des Clipbereiches liegen.

clipping against $x = \pm 1, y = \pm 1, z = \pm 1$

(x,y,z) inside?

→ only compare one value per border!



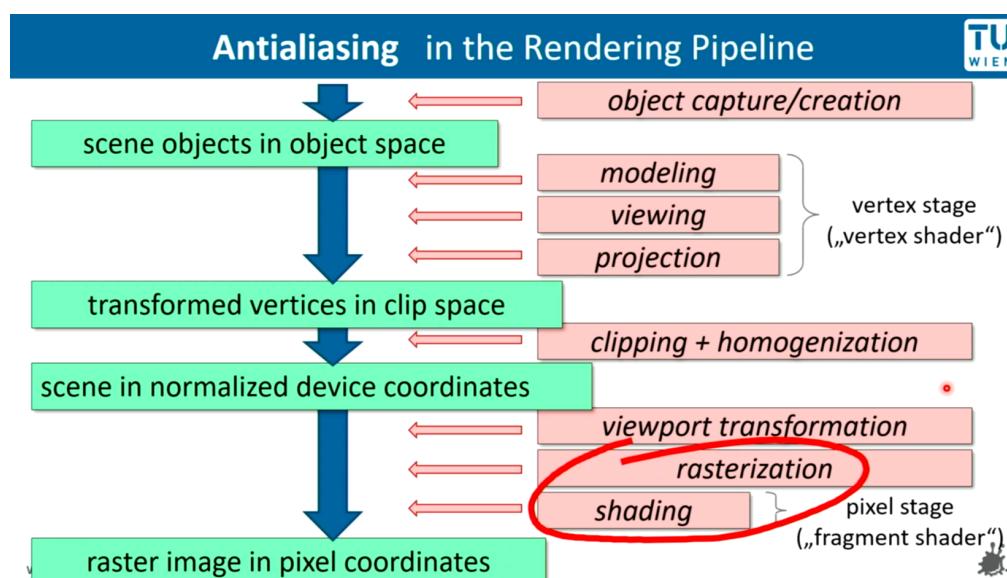
is done *before* homogenization:

clipping against $x = \pm h, y = \pm h, z = \pm h$

clips points that are behind the camera!

reduces homogenization divisions

Aliasing und Antialiasing:



Aliasing

- **Aliasing-Effekte** sind Fehler, die bei der **Umwandlung (Diskretisierung)** von analogen in digitale Informationen auftreten.
- **Ursachen für sichtbare Aliasing-Effekte**:
 1. Zu geringe **Auflösung**
 2. Zu wenige **verfügbare Farben**
 3. Zu wenige **Bilder pro Sekunde** (Frames per second)
 4. **Geometrische Fehler**
 5. **Numerische Fehler**

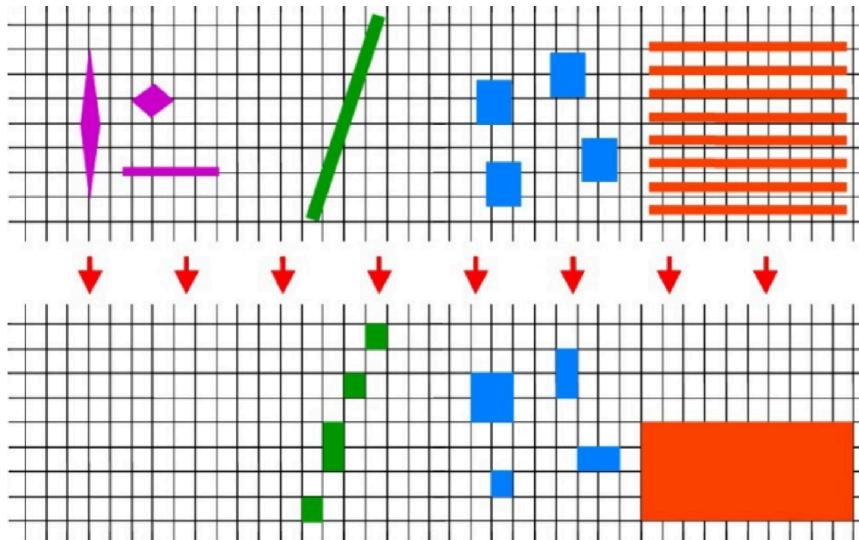
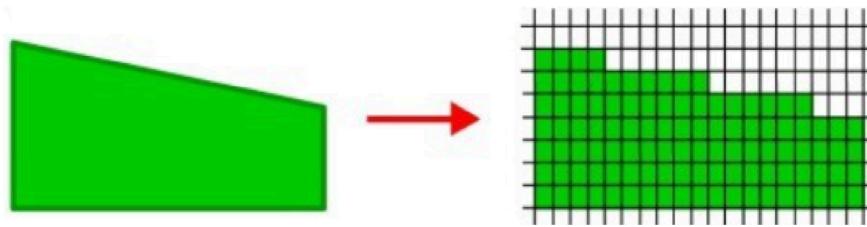
- Ein Beispiel für **Aliasing**: Ein Pixel hat nur einen Wert, stellt aber tatsächlich eine **kleine Fläche** dar, was zu Unschönheiten in Rasterbildern führen kann.

Antialiasing

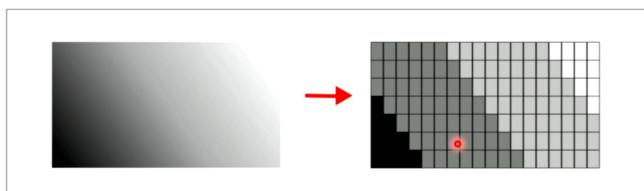
- Antialiasing** bezeichnet Methoden zur **Reduktion** unerwünschter Aliasing-Artefakte.
- Verbesserung der **Hardware** ist meist unrealistisch, daher kommen hauptsächlich **Software-Methoden** zum Einsatz.
- Der Fokus liegt oft auf **Anti-Aliasing** zur Behandlung des **Auflösungsproblems**.

Bekannte Aliasing-Effekte neben dem Treppeneffekt

- Verschwinden kleiner Objekte**
- Unterbrochene, schmale Objekte**
- Unterschiedliche Größen gleicher Objekte**
- Zerstörung feiner Texturen.**



Begriff Aliasing kommt von Alias: Das was angezeigt wird, zeigt sich an etwas anderem...

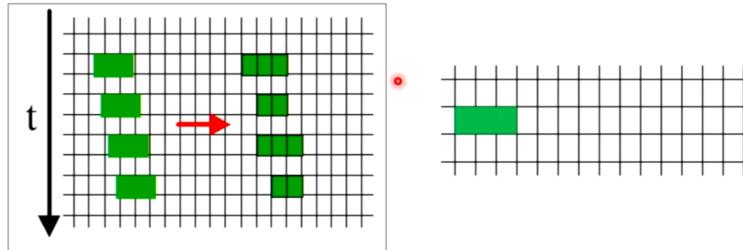


artificial color borders can appear

Aliasing kann auch in Animationen vorkommen:

- jumping images

- "worming"



- backwards rotating wheels



je nach dem welchen Pixel Mittelpunkt man anschaut bekommt man verschiedene Ergebnisse. Außerdem bekommt man dann einen Worming Effekt (also das Objekt wird immer größer und dann wieder kleiner). Ein anderer Effekt wäre, dass sich zum Beispiel in Filmen es so aussieht als würden sich die Räder rückwärts bewegen.

Antialiasing von Linien:

Ursache von Aliasing

- **Aliasing** entsteht durch **ungenügend feine Abtastung** des wahren Bildes, was zu Fehlern in der Rekonstruktion führt.
- **Nyquist-Shannon-Abtasttheorem:**
 - **Theoretische Grundlage:** Eine Information kann nur korrekt rekonstruiert werden, wenn die **Abtastfrequenz (sampling rate)** mindestens doppelt so hoch ist wie die höchste zu übertragende **Informationsfrequenz**.
 - Diese Grenze wird als **Nyquist-Limit** bezeichnet. ([2. Bilddatenahme](#))

Beispiel und Fehlerbehebung

- **Beispiel:** Eine zu grobe Abtastrate eines Signals führt zu einer **falschen Rekonstruktion** (z.B. eine Kurve wird zu einem Polygonzug, siehe Abbildung).
- Fehler können reduziert werden durch:
 1. **Vorfilterung des Signals.**
 2. **Nachbearbeitung des fertigen Bildes** (jedoch unterliegt diese der Vorfilterung und ist weniger effektiv).

Antialiasing für Linien

- **Pixel, die von einer Linie weiter durchkreuzt werden**, sollen mehr Linienfarbe erhalten als Pixel, die nur leicht gestreift werden.
- **Prozess:**

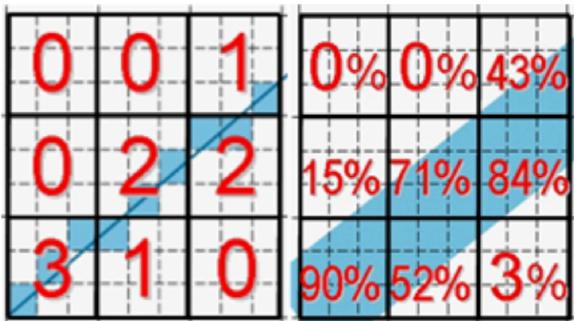
1. **Unterteilung jedes Pixels in Subpixel:** Für jedes Pixel wird gezählt, wie viele Subpixel von der Linie durchkreuzt werden.
2. **Intensitätswahl:** Die Intensität der Linienfarbe wird **proportional zur Anzahl** der Subpixel gewählt, die von der Linie durchquert werden.

Breitere Linien

- Für **breitere Linien** wird der **Prozentsatz der Überdeckung** des Pixels durch die Linie berechnet und die Intensität der Linienfarbe entsprechend angepasst.

Weighted Oversampling

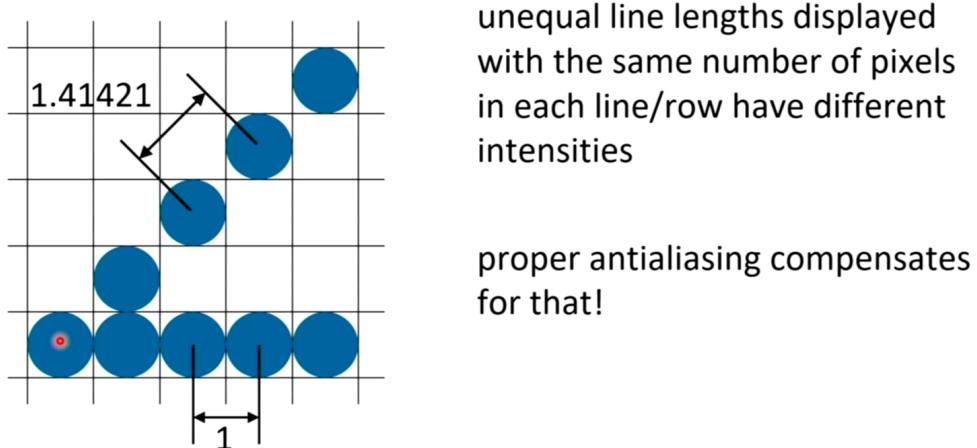
- Basierend auf der Erkenntnis, dass die **Mitte eines Pixels wichtiger** ist als der Rand, werden in einigen Fällen die **Subpixel in der Mitte stärker gewichtet** als die am Rand.
- Diese Technik wird als „**weighted oversampling**“ bezeichnet.



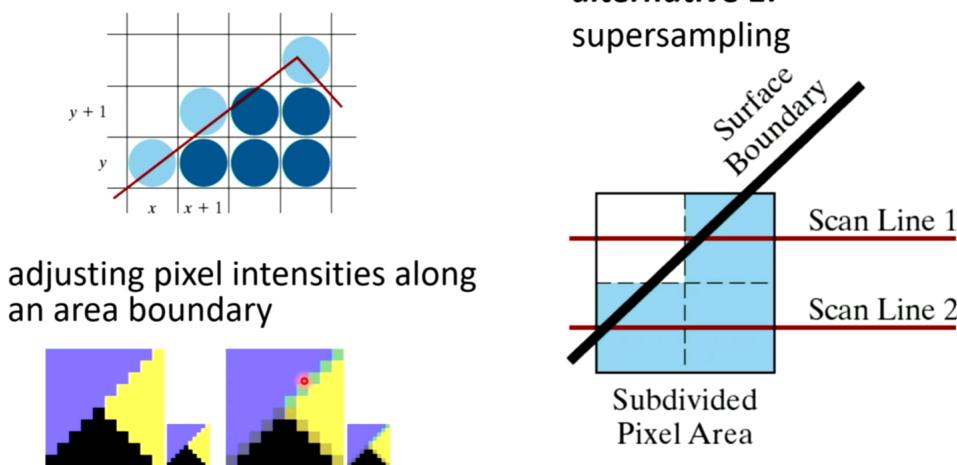
Hier haben wir ein 3 mal 3 Pixelfeld welches dann am Bildschirm angezeigt wird, und die kleinen Pixel im Pixel sind die Subpixel, welche gerendert werden wollen. Man kann allerdings nur die großen Pixel an- und abschalten.

Andere Effekte von Antialiasing:

Verschiedene Längen trotz gleiche Pixelgröße:



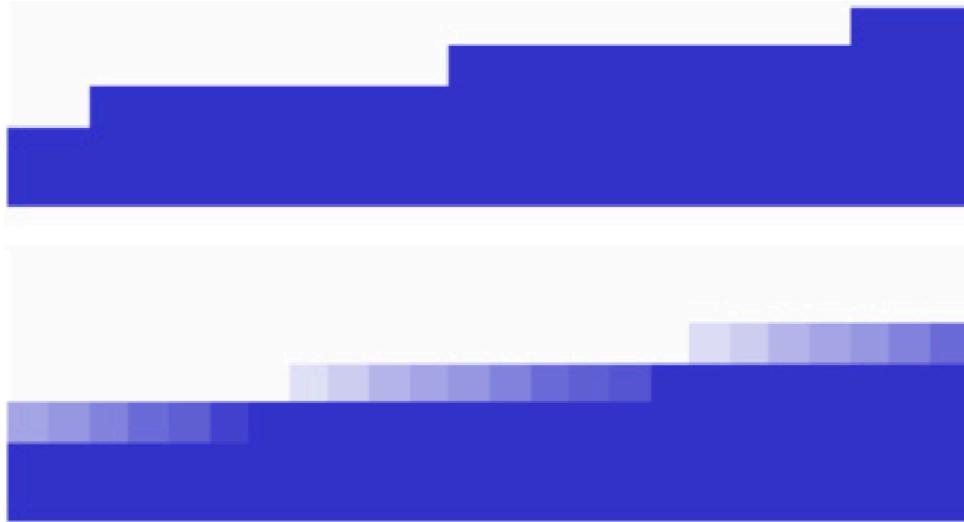
Das kann man mit Area Boundaries behoben:



Antialiasing von Polygonkanten:

1. Ziel des Antialiasings

- **Ziel:** Reduktion der aliasing-bedingten Treppeneffekte an den Kanten von Polygonen.



2. Methoden zur Berechnung des Antialiasings

- **Alternativen:**
 - **Supersampling:** Mehrere Proben pro Pixel werden verwendet.
 - **Überdeckungsgradberechnung:** Der Überdeckungsgrad eines Pixels durch das Polygon wird direkt berechnet.

3. Berechnung des Überdeckungsgrads

- **Rasterkonversion:** Der Überdeckungsgrad wird während der Rasterkonversion berechnet, also beim Erzeugen der Randlinie und Füllung des Polygons.

- **Scanlinien-Füllverfahren:**

- Bei der Berechnung der Endpunkte der Scanlinien (Spans) im Füllverfahren fallen genug Informationen an, um den Überdeckungsgrad fast kostenfrei zu berechnen.

4. Verwendung der Entscheidungsvariable aus dem Bresenham-Algorithmus

- **Bresenham-Linien-Algorithmus:**

(siehe [5. Rasterisierung](#))

- Die Entscheidungsvariable p_k gibt an, welches Pixel als nächstes gezeichnet wird.
- Diese Variable kann so umgewandelt werden, dass ihr Wert den Überdeckungsgrad des letzten Pixels darstellt.

- **Transformation:**

- $p' = y - y_{mid}$,
wobei:

$$y_{mid} = \frac{y_k + y_{k+1}}{2}$$

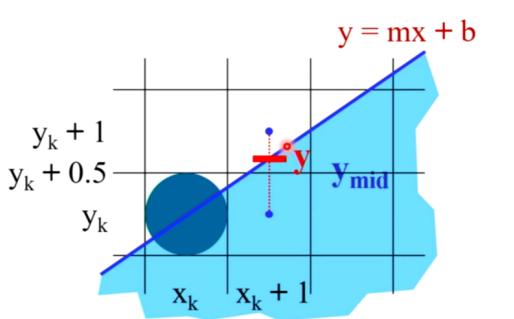
- Das Vorzeichen von p' hat die gleiche Bedeutung wie das Vorzeichen von p_k .

- **Berechnung des Überdeckungsgrads:**

- $p = p' + (1 - m)$, wobei m der Steigungsfaktor ist.
- Der Wert von p liegt im Bereich $0 \leq p \leq 1$ und entspricht dem Überdeckungsgrad an der Stelle x_k .

alternative 2: similar to Bresenham algorithm

$$p' = y - y_{mid} = [m(x_k + 1) + b] - (y_k + 0.5)$$

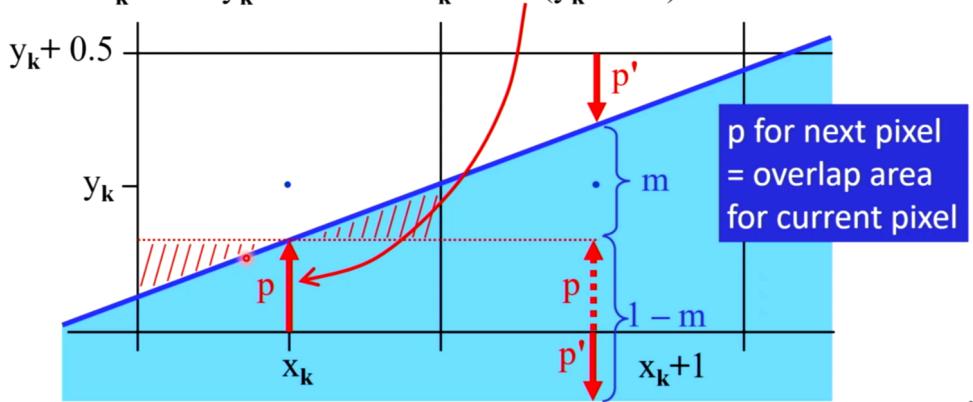


$p' < 0 \Rightarrow y$ closer to y_k

$$\begin{aligned} p &= p' + (1 - m) : \\ p < (1 - m) &\Rightarrow y \text{ closer to } y_k \\ p > (1 - m) &\Rightarrow y \text{ closer to } y_{k+1} \end{aligned}$$

(and $p \in [0,1]$)

$$p = p' + (1 - m) = [m(x_k + 1) + b] - (y_k + 0.5) + (1 - m) = \\ = mx_k + b - y_k + 0.5 = mx_k + b - (y_k - 0.5)$$



5. Effizienz

- **Inkrementelle Berechnung:** Das Antialiasing lässt sich sehr schnell und inkrementell berechnen, was zu einer effizienten Verarbeitung führt.

6. Anpassungen für andere Winkel

- **Drehungen und Spiegelungen:** Für andere Winkel werden Drehungen um 90° und/oder Spiegelungen des Verfahrens verwendet.



Abtastung und Fouriertransformation:

1. Fouriertransformation und Frequenzraum

- **Fouriertransformation:** Beschreibt ein Signal im Ortsraum (z.B. eine Scanlinie in einem Bild) als Summe von Sinusschwingungen im Frequenzraum.
 - **Sinusschwingung:** Durch **Frequenz**, **Phase** und **Amplitude** beschrieben.
 - **Spektrum:** Im Frequenzraum wird ein Signal durch sein Spektrum spezifiziert, also Phase und Amplitude in Abhängigkeit von den Frequenzen ω .
 - **Euler-Identität:** $e^{ix} = \cos(x) + i \sin(x)$, mit der Phase und Amplitude effizient durch imaginäre Zahlen beschrieben werden.
- **Inverse Fouriertransformation:** Wandelt das Spektrum im Frequenzraum zurück in das Signal im Ortsraum.

2. Faltung

- **Faltung:** Kombiniert zwei Funktionen und ergibt das integralgewichtete Summenprodukt der beiden.
 - **Formel:** $f_1 * f_2(x) = \int_{-\infty}^{\infty} f_1(\tau) f_2(x - \tau) d\tau$
- **Faltungstheorem:**
 - Multiplikation zweier Funktionen im Ortsraum entspricht der Faltung ihrer Spektren im Frequenzraum:

$$f_1 f_2 = F_1 * F_2$$
 - Faltung im Ortsraum entspricht der Multiplikation der Spektren im Frequenzraum:

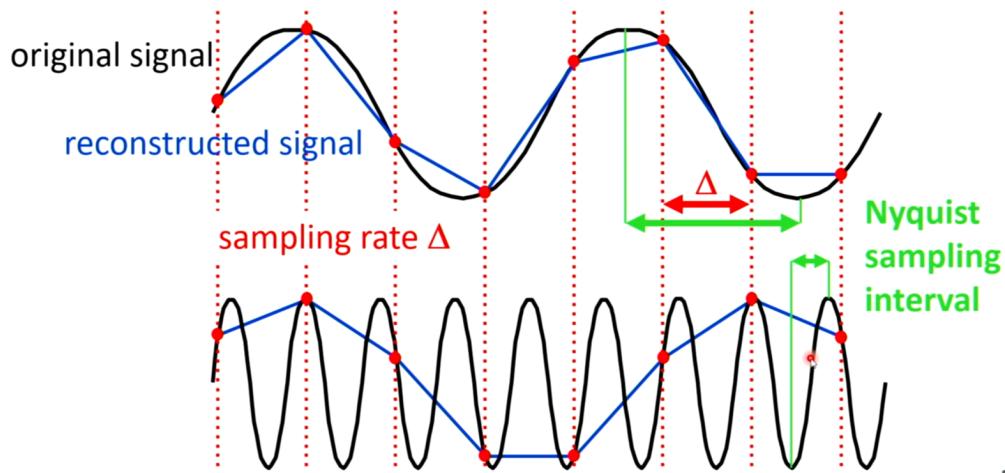
$$f_1 * f_2 = F_1 F_2$$

3. Abtasten und Diskretisierung

- **Abtasten im Ortsraum:** Multiplikation des Signals mit einer Kammfunktion $comb_T$.
 - **Frequenzraum:** Entspricht einer Faltung mit der Kammfunktion $comb_{1/T}$.
 - Die Zahnabstände in $comb_T$ und $comb_{1/T}$ sind invers proportional zueinander.
- **Faltung mit der Kammfunktion:**
 - Führt zu einer **Replikation des Spektrums** im Frequenzraum (Schattenspektren).

4. Nyquist-Limit und Aliasing

- **Nyquist-Limit:** Bei zu niedriger Abtastfrequenz (Abtastintervall T zu groß) sind die Zähne der Kammfunktion $comb_T$ im Ortsraum zu weit auseinander und die Replikationen im Frequenzraum (durch $comb_{1/T}$) zu nah beieinander.
 - Dies führt zu **Aliasing**, da die Schattenspektren sich überlappen und eine fehlerfreie Rekonstruktion unmöglich wird.
- weiteres zu Nyquist: [2. Bilddatenahme](#)



a signal can only be reconstructed without information loss
if the **sampling frequency** is at least
twice the highest frequency of the signal

$$\text{Nyquist sampling frequency: } f_s = 2 f_{\max}$$

$$\Delta x_s = \frac{\Delta x_{\text{cycle}}}{2} \quad \text{with} \quad \Delta x_{\text{cycle}} = 1 / f_{\max}$$

i.e. sampling interval \leq one-half cycle interval

!

5. Rekonstruktion und Schattenspektra

- **Exakte Rekonstruktion:** Um die durch die Diskretisierung entstandenen Schattenspektra zu entfernen, wird das Spektrum der diskretisierten Funktion im Frequenzraum mit einer **Rechteckfunktion** multipliziert.
 - Das ursprüngliche Spektrum bleibt übrig.
- **Rekonstruktion im Ortsraum:**
 - **Faltung mit Sinc-Funktion:** Die exakte Rekonstruktion erfolgt durch Faltung im Ortsraum mit der **Sinc-Funktion**: $\text{Sinc}(x) = \frac{\sin(x)}{x}$

6. Praktische Rekonstruktion

- Da die Sinc-Funktion über einen unendlichen Bereich nicht null ist, wird für eine praktikable Rekonstruktion:
 - **Rechteckfunktion** (Nächster-Nachbar-Interpolation) oder
 - **Dreiecksfunktion** (lineare Interpolation) verwendet.

Fourier Transform

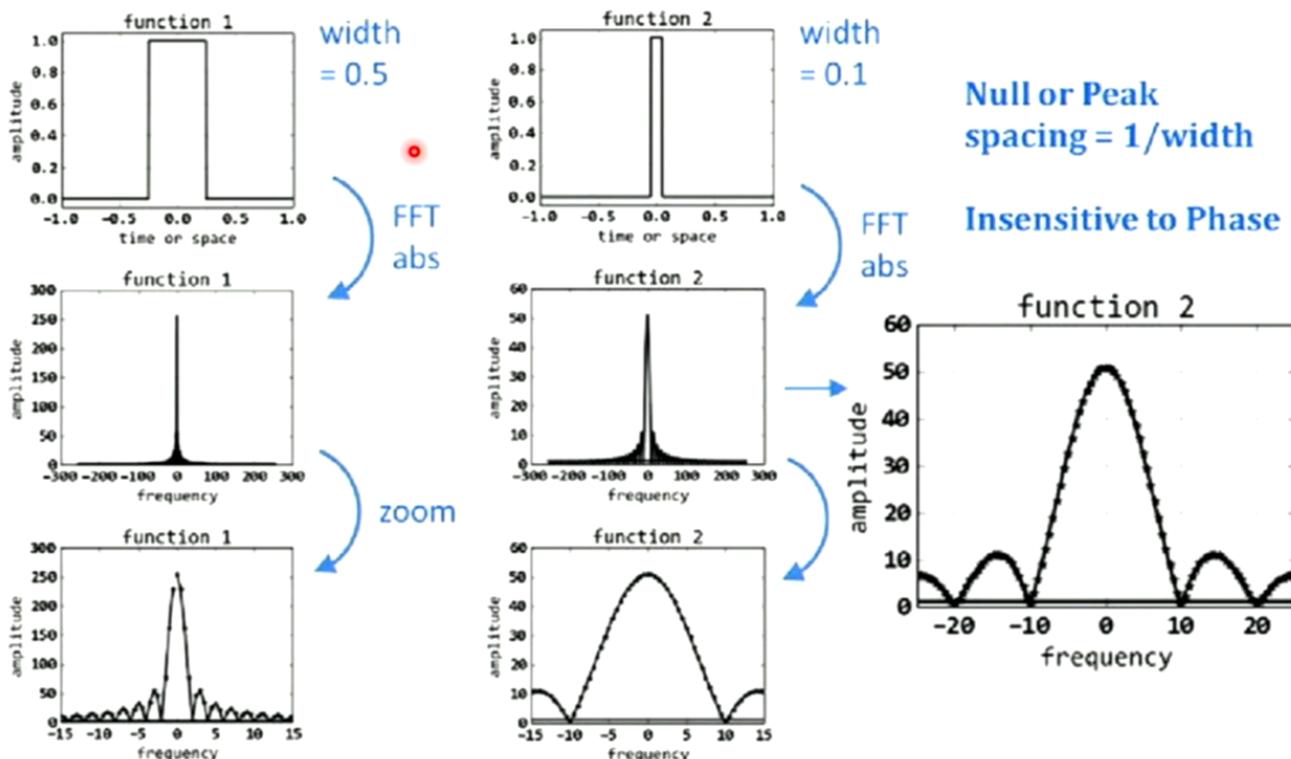
- Link between spatial $f(x)$ and frequency $F(\omega)$ domain

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i \omega x} dx$$

$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{2\pi i \omega x} d\omega$$

$$e^{ix} = \cos x + i \sin x$$

Beispiel zur Fourier Transformation:



- The spectrum of the convolution of two functions is equivalent to the product of the transforms of both input signals, and vice versa

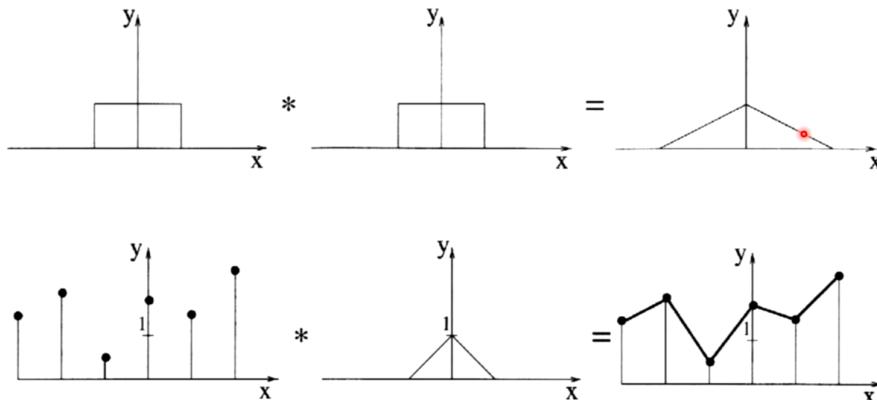
- Convolution $f_1 * f_2(x) = \int_{\mathbb{R}} f_1(\tau) f_2(x - \tau) d\tau$

- Convolution theorem $f_1 * f_2 \equiv F_1 F_2$

$$F_1 * F_2 \equiv f_1 f_2$$

"Wenn ich eine Faltung im einen Raum mache (Orts oder Frequenzraum), ist es im anderen Raum dann zu multiplizieren"

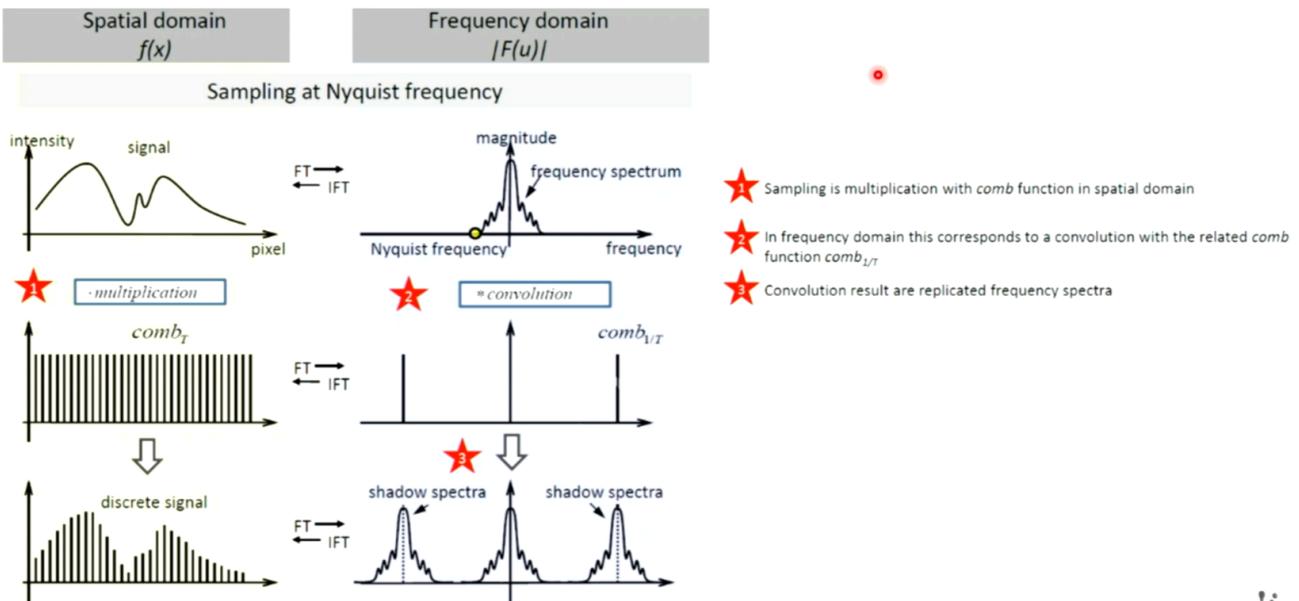
Hier ein Beispiel zu dem Convolution theorem: (Mehr dazu siehe: [7. Globale Operationen](#))



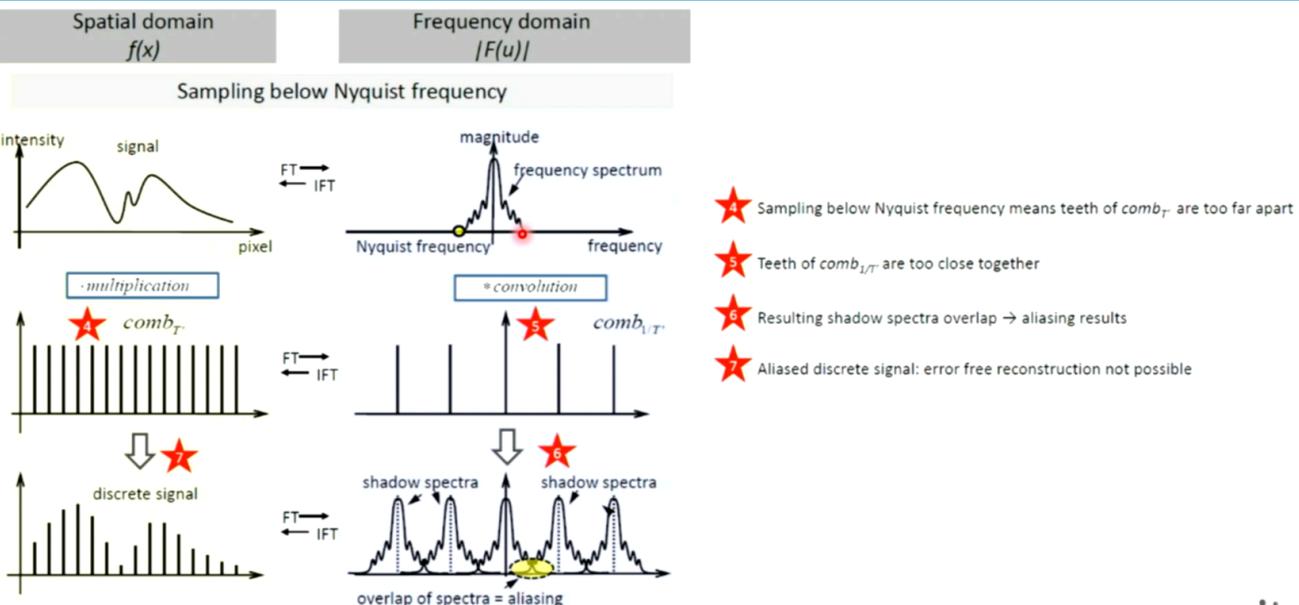
damit das 2. geht, muss der Abstand der Samplepunkten genau doppelt so groß sein wegen Nyquist (das bitte nochmal Fact-checken, er hat schnell gesprochen)

Cheatsheet für alles was Sampling betrifft:

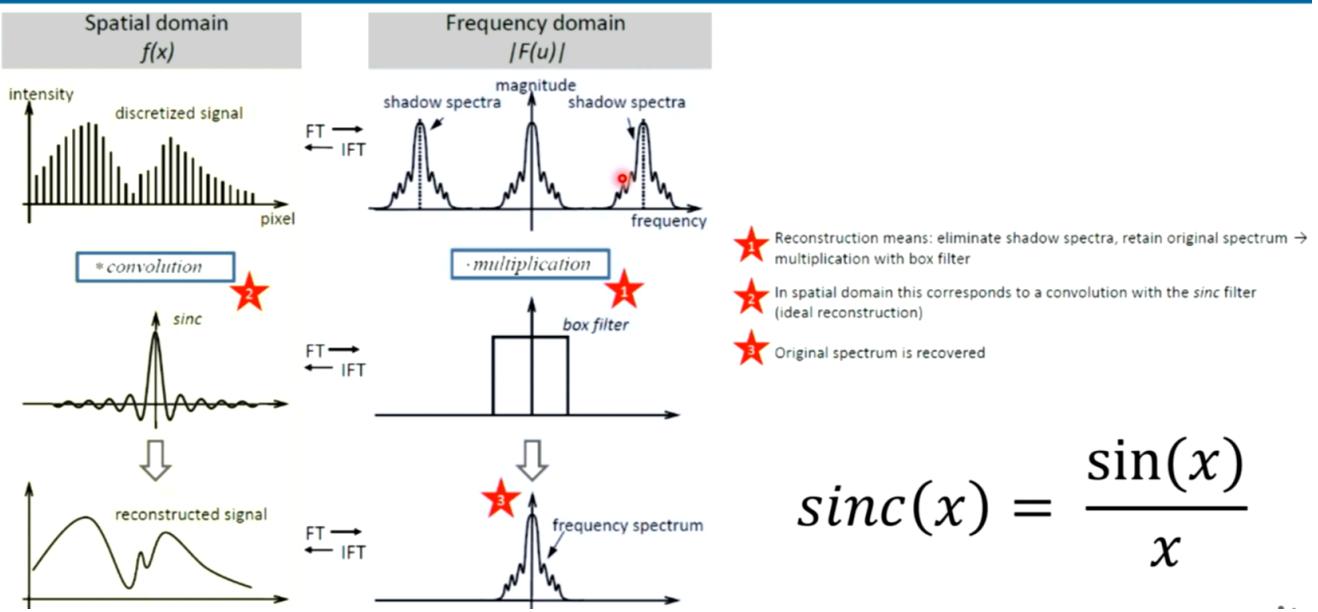
Sampling at the Nyquist Frequency



Sampling Below the Nyquist Frequency



Reconstruction



hier nochmal hochauflösend aus dem Skriptum:

