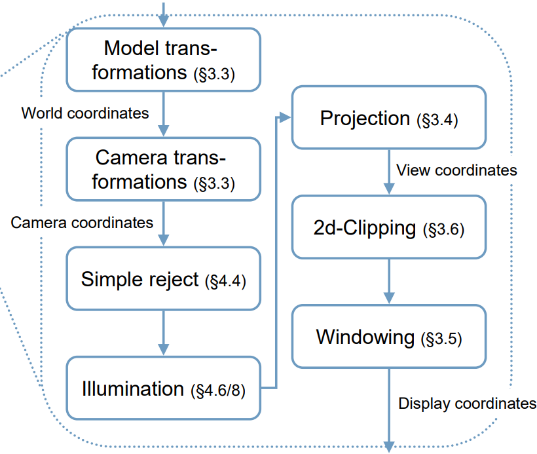
Rendering: Mapping den Geometrische Modellen, Objekten oder der Szene auf ein Bild auf dem Display.

Rendering Pipeline: Konkrete Implementation des Rendering in Soft oder Hardware.

De-Facto Standard Pipeline: Application – Geometry – Rasterization – Fragments – Display

Application

* Receiving and sending of commands and data
* Command buffering/interpretation

Geometry

* Model transformations
* Camera transformations
* Simple reject (Back-Clipping)
* Illumination
* Projection
* 2d-Clipping
* Windowing

Rasterization

* Primitive assembly (Triangles, lines etc)
* Triangle setup (Tiefen/-Farbwerte, Textur Koordinaten)
* Rasterization
* Anti-Aliasing
* Z-Buffer

Fragments

* Transparency
* Texturing

Display

* Frame buffer
* Gamma-correction
* Analog/digital-converter

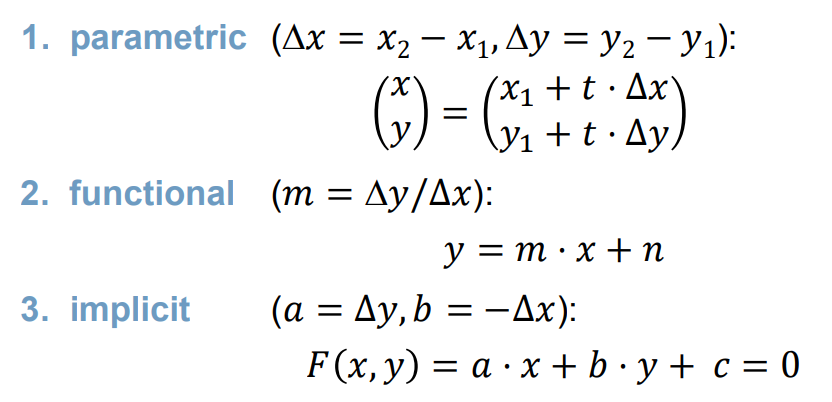
**Kapitel 2 – Rasterization**

Dekomposition von geometrischen 3D Objekten in ein 2D Image mit im Raster angeordneten Pixeln.

Problem: Linie die abgebildet werden soll überlappt die Pixel mal mehr, mal weniger.

Anforderungen:

* Linie muss konstante Dicke, Helligkeit und keine Lücken haben
* Endpunkte muss exakt sein
* Berechnung muss schnell sein, Algorithmus muss sich in Hardware implementieren lassen.

1. Stütz + Richtungsvektor + Parameter t
2. Steigung m \* X-Wert + Y-Offset n
3. Nach Y umstellen, oder Punkt einsetzen

zum Prüfen

**Naive algorithm**:

Probleme:

Floating point values 𝑦 and 𝑚,

divisions and multiplications,

rounding,

vertical lines,

appearance of lines depends on the slope 𝑚.

Use functional representation: 𝑦 = 𝑚 \* 𝑥 + 𝑛

1) Iterate from 𝑥1 to 𝑥2 in pixel-steps

2) Compute 𝑦-values

3) Round

4) Draw

**Bresenham Algorithmus:**

Aufteilung in Oktanten, nur im 1. Zeichnen. Bei Linie in anderen Oktanten die Gerade durch Transformation in den 1. Verlegen.

Entscheidungsvariable d gibt an, ob Mittelpunkt zwischen nächsten Pixelkandidaten (NE/E) über oder unter der Linie liegt. ( d >= 0 => Mittelpunkt unter Linie, d < 0 => Mittelpunkt über Linie)

Initialisierung: Umformulierte Gleichung, die prüft, ob Steigung > oder < 0.5 ist. \* 2 um Division zu vermeiden.

**Bresenham Algorithmus für Kreise:**

Diesmal im 2. Oktanten, andere Punkte extrapolieren.

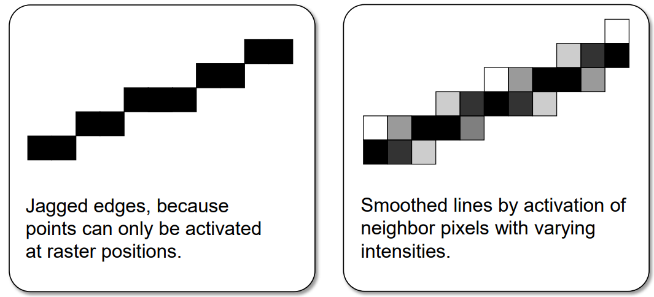
Ähnliches Vorgehen, mit Kreisgleichung prüfen, ob Punkt innerhalb oder Außerhalb vom Kreis liegt.

Implizite Gleichung: 𝐹(𝑥, 𝑦) = (𝑥 – 𝑥𝑀)2 + (𝑦 – 𝑦𝑀)2 − 𝑟2 = 0

Initialisierung von d: d = 5 - 4 \* r;

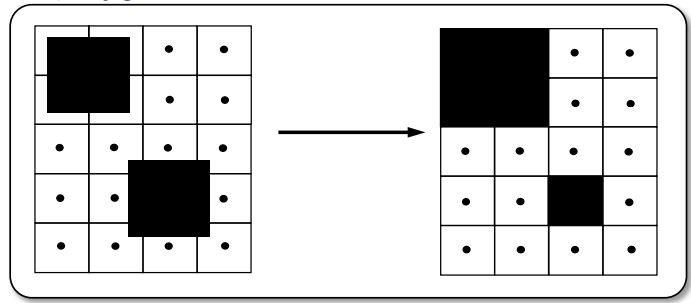
**Aliasing**

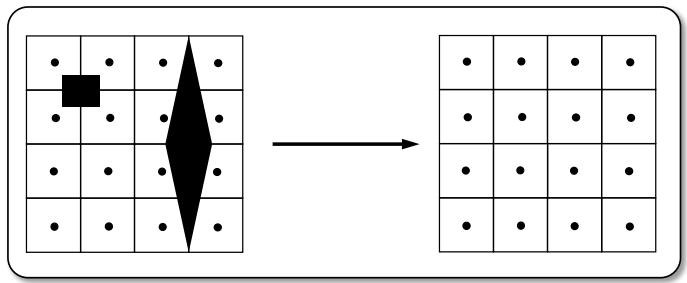
Tritt auf wenn die Abtastfrequenz im Vergleich zum abgetasteten Signal zu klein ist. (Abtastfrequenz = Pixel)

Visual Effects: Aliasing Effekte, ausgelöst durch undersampling. Objekte werden mit niedrigerer, falscher Frequenz dargestellt.

Visual Artifacts (Spatial Aliasing):

* Jagged edges/jaggies

Falsche Größe, Verschwindende Objekte:



Temporal Aliasing: Rad eines Autos scheint während der Fahrt stillzustehen

Anti-Aliasing Techniken:

Edge Smothing: Anti-Aliasing Methoden, die durch Rasterisierung hervorgerufene Artefakte bereinigen

Super-Sampling: Pixel wird in Sub-Pixel unterteilt. Die tatsächliche Pixelfarbe richtet sich nach dem Durchschnitt der Sub-Pixel.

Gewichtete und Ungewichtete overlap area: Korrekten overlap von Line mit Pixeln berechnen => Pixel mit Grauwert\*

Stochastische Methode: Anti Aliasing nur an Zufälligen Punkten. Effizienter, aber kann bei Animationen flackern

FSAA (Full scene anti-aliasing): Super-sampling for every pixels of the scene, very expensive.

TXAA (Temporal anti-aliasing): Filtering of multiple samples inside and outside the pixel with pixels of previous frames.

DLSS (Deep learning super sampling): A NN is trained (an stored on the driver) to render the anti-aliased high resolution image.