Übung: Interaktive Computergrafik

Heute

- Besprechung des fünften Übungsblatts (Deferred Shading)
- Shading Overdraw
- Temporal Antialiasing

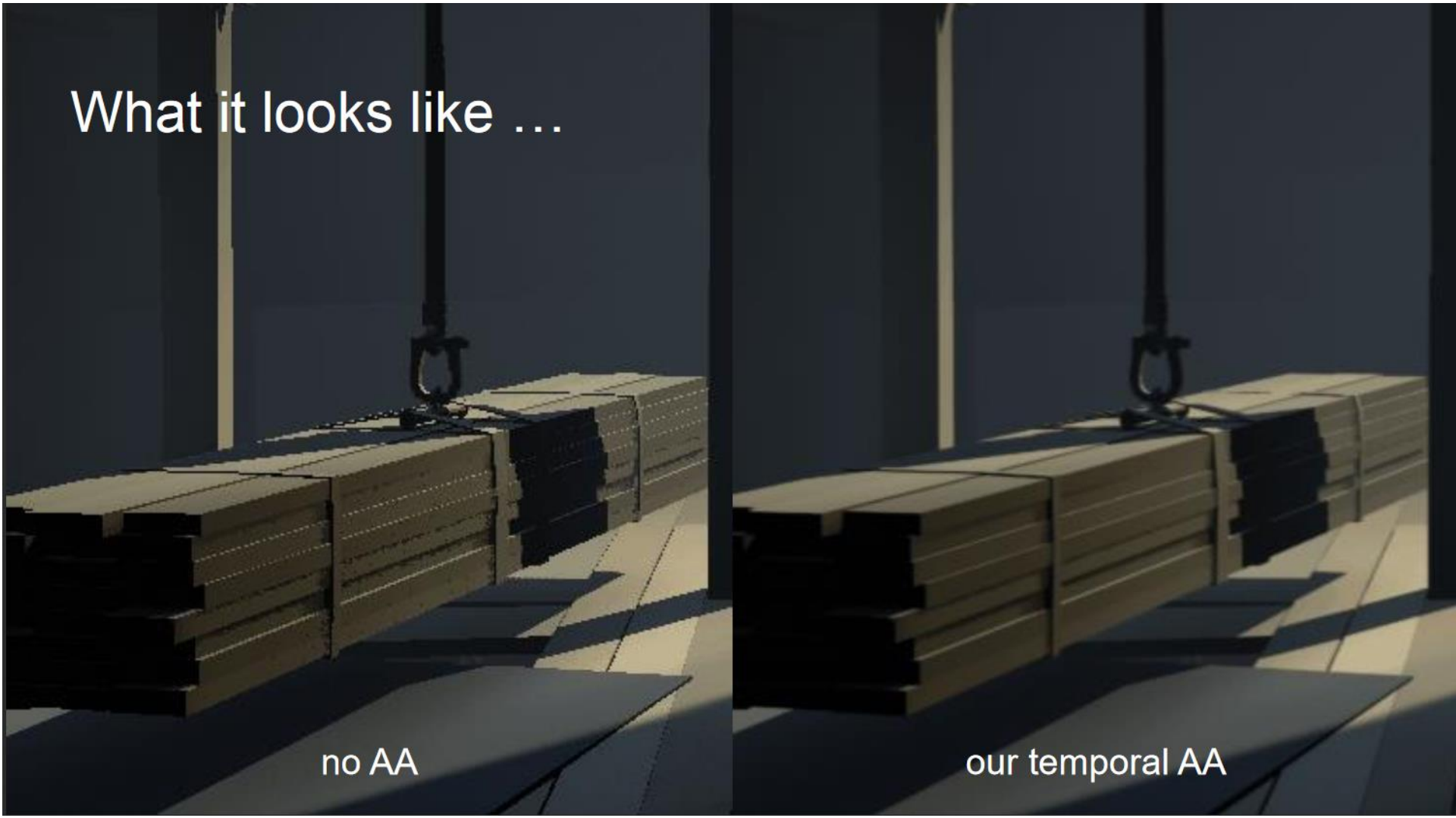
Wdh. Deferred Shading und Antialiasing

- Supersampling in der Praxis zu teuer
  - Speicheraufwand
  - Shading zu teuer

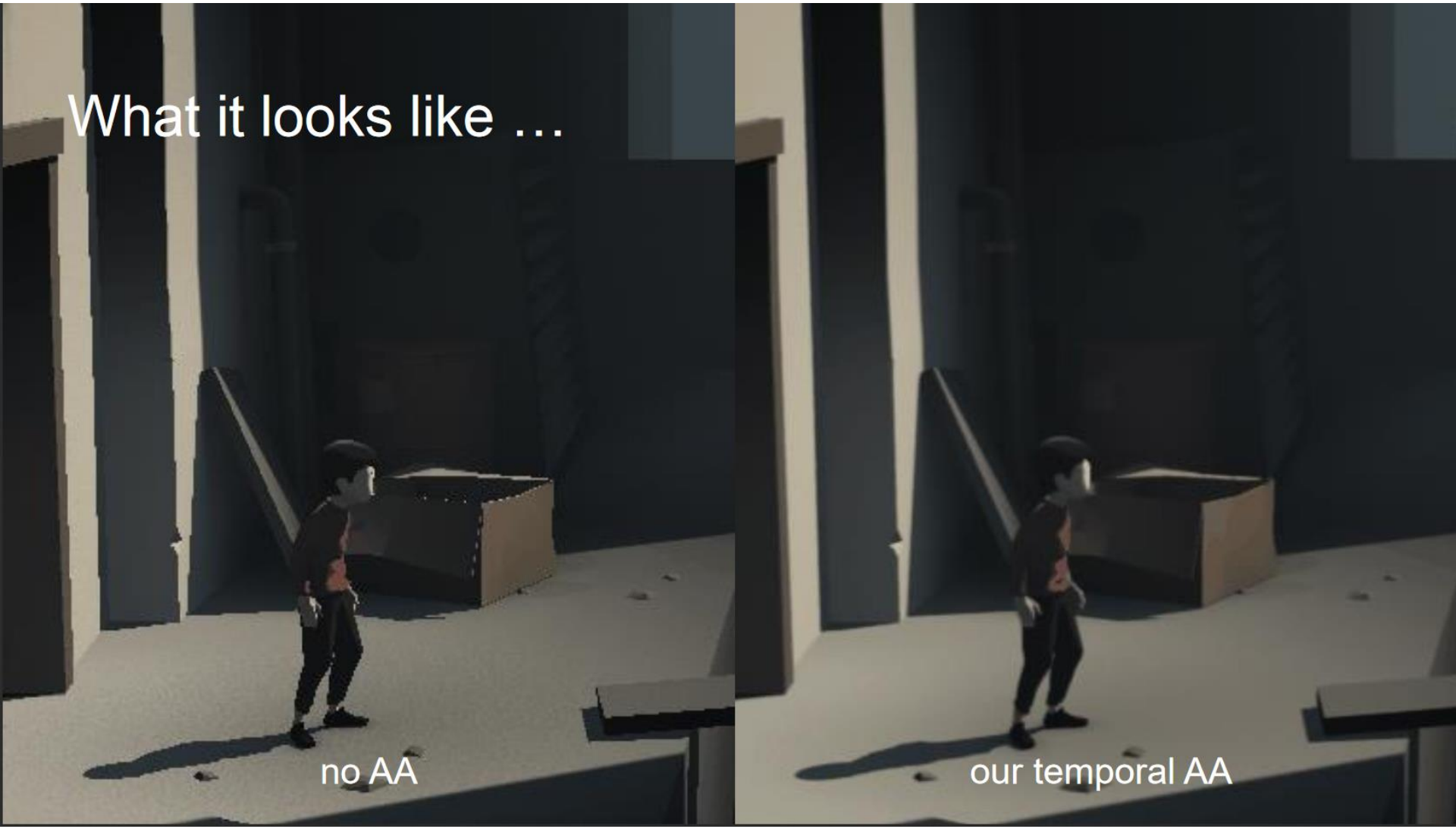
- MSAA
  - Entkoppelung von Shading und Visibility Sampling schwierig
  - Effizientes Shading schwierig (diveregente Ausführung)
  - Hoher Speicherverbrauch

- Morphologisches Antialiasing
  - Kann unterabgetastetes Signal nicht rekonstruieren
  - Hilft nur wenig bei temporaler Stabilität

Temporal antialiasing



Temporal antialiasing



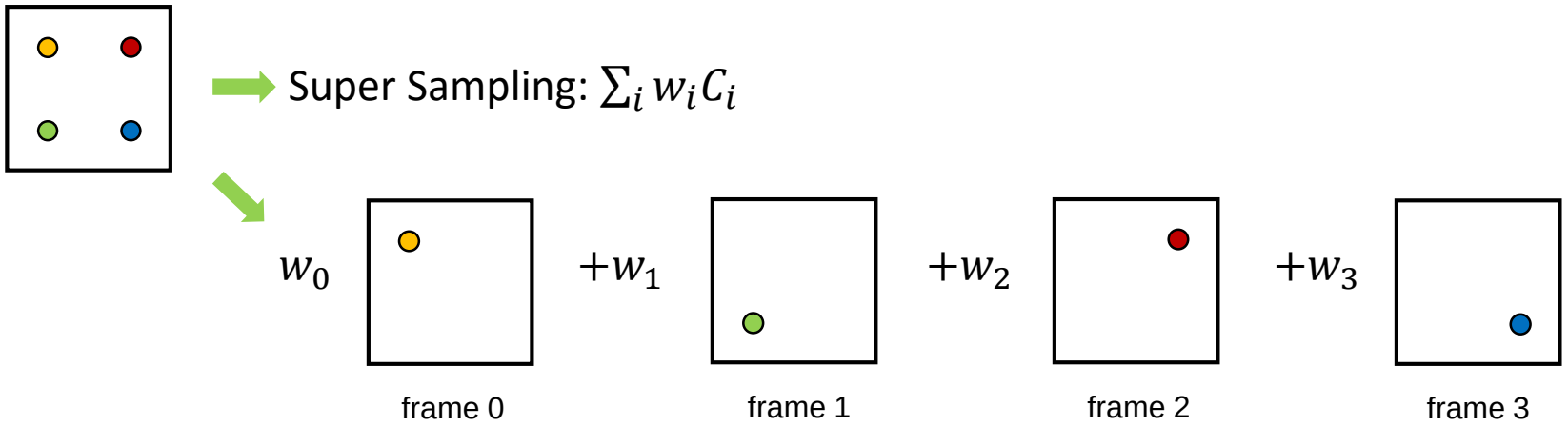
Temporal Antialiasing

- ideale Lösung für Antialiasing: Vorfilterung
  - nicht immer möglich, aber Fortschritte (siehe andere Kapitel der VL)
- billige Lösung: morphologisches Antialiasing
- brute-force Lösung: mehr Samples
  - klassisches, direktes Super Sampling ist teuer
  - besser sind MSAA, CSAA (insb. für Forward Shading)
- temporales Antialiasing (TAA):
  - mehr Samples, aber nicht alle gleichzeitig

Temporales Antialiasing

Amortisation der Samples über mehrere Frames

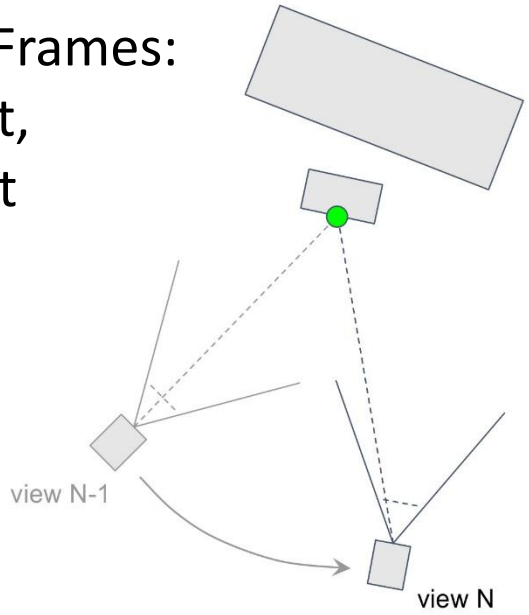
- Grundidee: verwende Samples aus vorhergehenden Frames (dazu leichtes Jittering der Kamera)



Temporales Antialiasing

Amortisation der Samples über mehrere Frames

- Grundidee: verwende Samples aus vorhergehenden Frames (dazu leichtes Jittering der Kamera)
- projiziere Fläche in vorherige Frames:
  Position der Fläche ist bekannt,
  berechne Pixelkoordinaten mit
  Objekt, Kamera- und
  Projektionstransformation
  aus dem vorherigen Frame



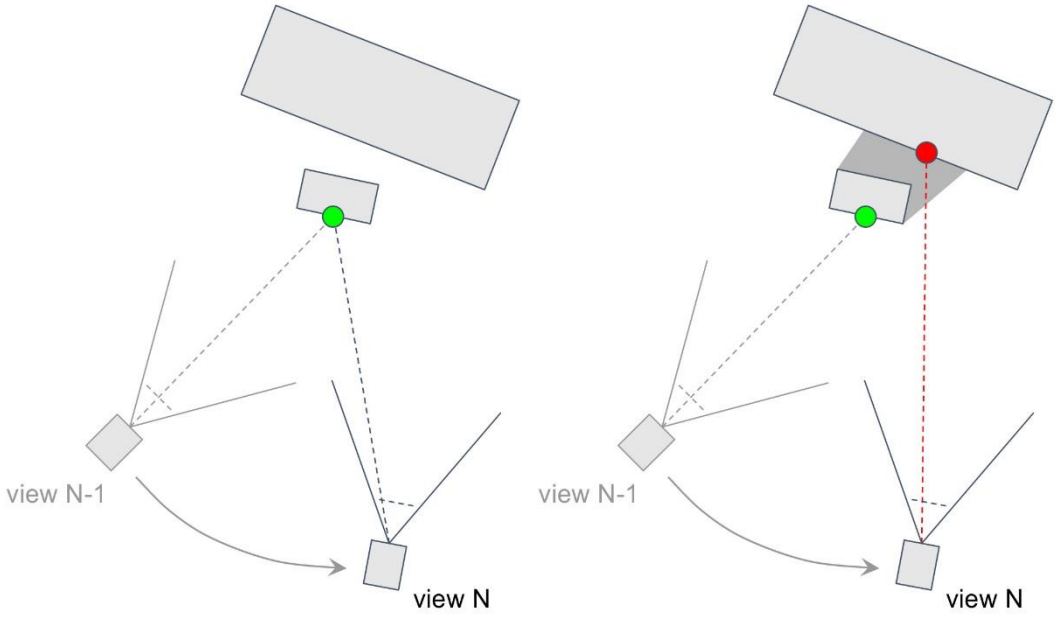
- ursprüngliche TAA-Varianten
  - verwende Samples nur bei
    gleicher Material ID, Tiefe, Normale etc.

Temporales Antialiasing

Temporales Antialiasing

Amortisation der Samples über mehrere Frames

- Grundidee: verwende Samples aus vorhergehenden Frames (dazu leichtes Jittering der Kamera)
- Probleme:
  - Verdeckung (Abb. rechts)
  - veränderte Beleuchtung
  - Bewegung der Objekte
  - keine Glättung der Kanten, nur Antialiasing des Shading (Samples von der „gleichen“ Fläche)



- für einige dieser Probleme kann man sich Lösungen überlegen, allerdings wird das schnell unpraktikabel und anfällig für Artefakte (z.B. Ghosting, Spuren von Glanzlichtern, …)

Temporales Antialiasing

Pragmatischer und robuster Ansatz (so durchgeführt für jeden Pixel)

- projiziere die Fläche (nur) in das vorherige Frame
- Gewichtung der Farbwerte nach exponentiell geglätteten Mittelwert (engl. exponential moving average) mit Parameter α (z.B. $\alpha = 0.1$)
- Pixel-Wert P_n im Frame n aus
  - neu-berechnetem Farbwert C_n und
  - altem Pixel-Wert P_{n-1} mit

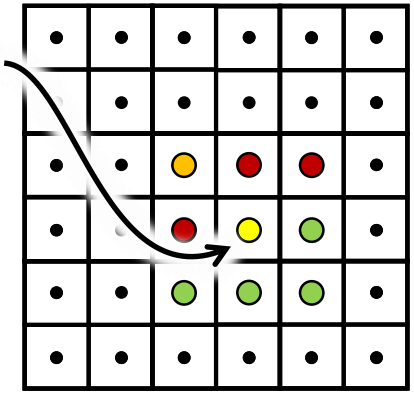
$$P_n = \alpha \cdot C_n + (1 - \alpha) \cdot P_{n-1}$$

Temporales Anti-Aliasing

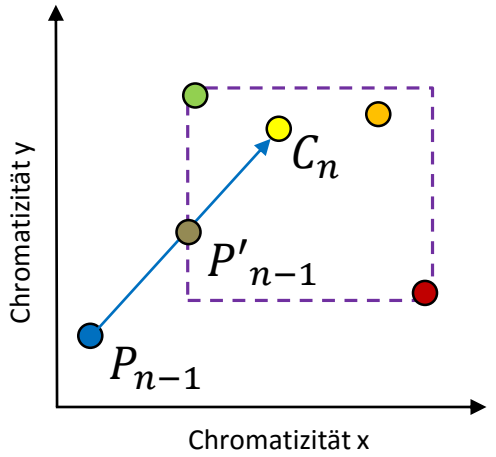
Pragmatischer und robuster Ansatz (so durchgeführt für jeden Pixel)

- man möchte nur Farbwerte aus vorherigen Frames verwenden, die konsistent mit C_n sind (Annahme: große Unterschiede ↔ andere Fläche)

- Idee nennt sich Neighborhood Clipping
  - betrachte Nachbarschaft (z.B. 3×3) des Pixels mit C_n
  - bestimme Hüllkörper oder konvexe Hülle der dort gefundenen Chrominanzen oder RGB-Werte
  - wenn P_{n-1} außerhalb, dann berechne Schnitt $\overline{P_{n-1}C_n}$ und verwende diese Farbe



- durch reine Betrachtung von Farbwerten, werden alle Ursachen von Aliasing adressiert, aber
  - u.U. zu starke Glättung
  - Flackern, bei großflächigem Aliasing (z.B. Nachbarschaft komplett verdeckt durch eine rot-grün flackernde Fläche → Clipping liefert immer rot oder grün)

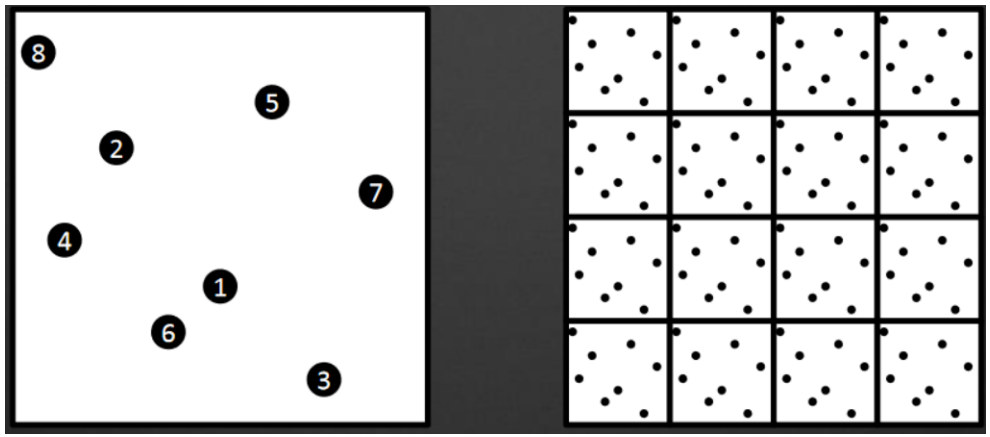


Kamera-Jittering

- Globaler Jitter durch angepasste Projektionsmatrix

```
ProjMatrix[2][0] += (SampleX * 2.0f - 1.0f ) / ViewRect.Width();
ProjMatrix[2][1] += (SampleY * 2.0f - 1.0f ) / ViewRect.Height();
```

- Sample Pattern
  - z.B. Halton(2,3) Sequenz



Wahl des Motionvectors

- Pro-Pixel Motionvectors können nicht direkt benutzt werden
  - Sonst kein Anti-Aliasing zwischen Verdecker und Hintergrund