UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

3D Tetris

David Jäck

Anfängerpraktikum

Hauptseminar Computergrafik und Visualisierung, Heidelberg, Deutschland, Januar 17, 2022

Agenda

- Projekt Motivation
- LWJGL
- Implementierung
- Optimierung
- Demo
- Ausblick

https://www.tz.de/leben/games/playstation-5-ps5-gameplay-eindrucksvolle-tech-demo-zeigt-next-gen-sony-zr-13762689.html

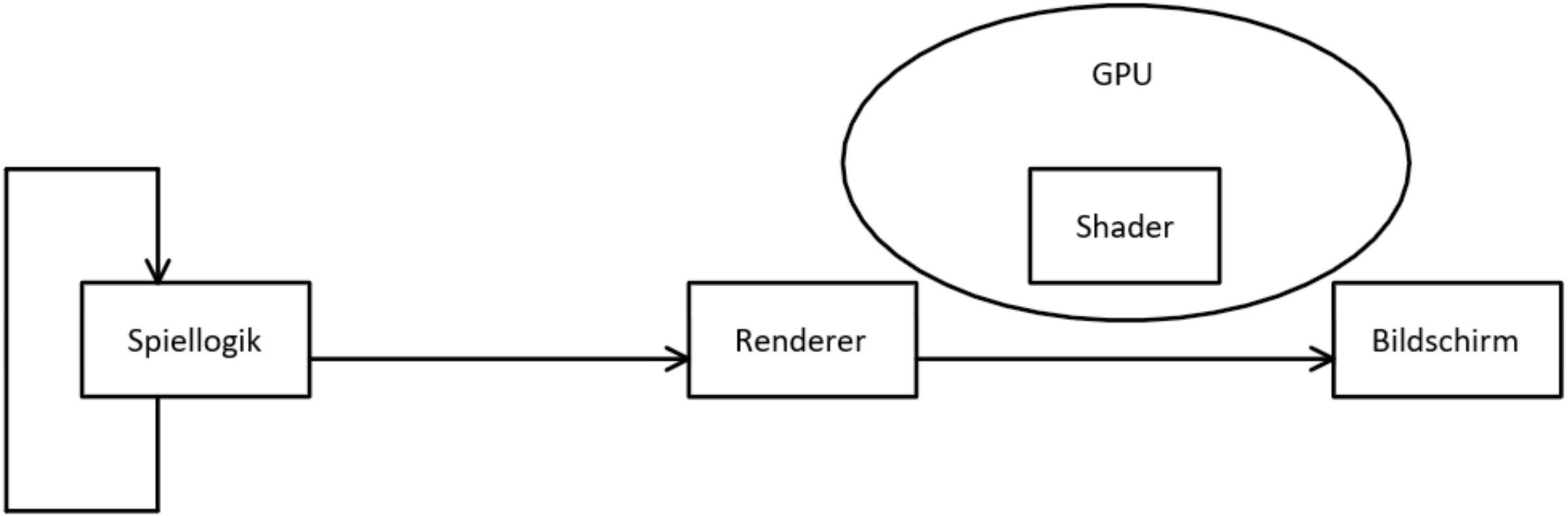
LWJGL/OpenGL

- Grundproblematik:
  - Java ist nicht die performanteste Programmiersprache
  - Grafische Berechnungen sind anspruchsvoll

- Light Weight Java Game Library[1]
  - APIs für die Nutzung von OpenGL

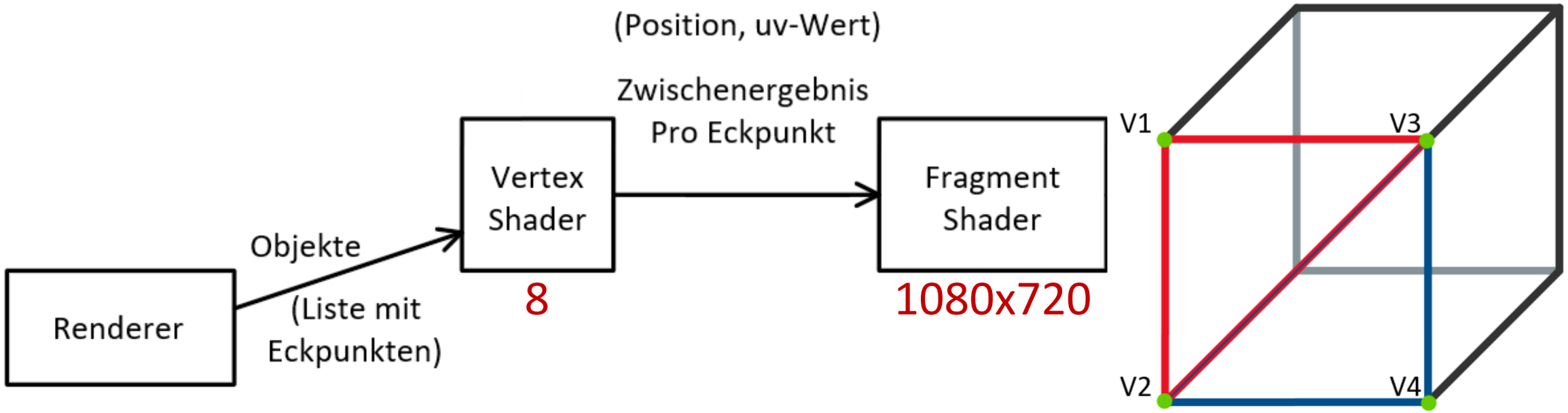
- Open Grafics Library[2]
  - APIs für das rendern von zwei und drei dimensionalen Vector Grafiken (Polygone)

Game Engine Schema



Shader

- Werder auf der GPU ausgeführt
- Berechnen das Aussehen von Objekten durch modellierbare Transformationen
- Liefern Pixel für Pixel den rgb-Farbwert
- Sprache GLSL: OpenGL Shading Language

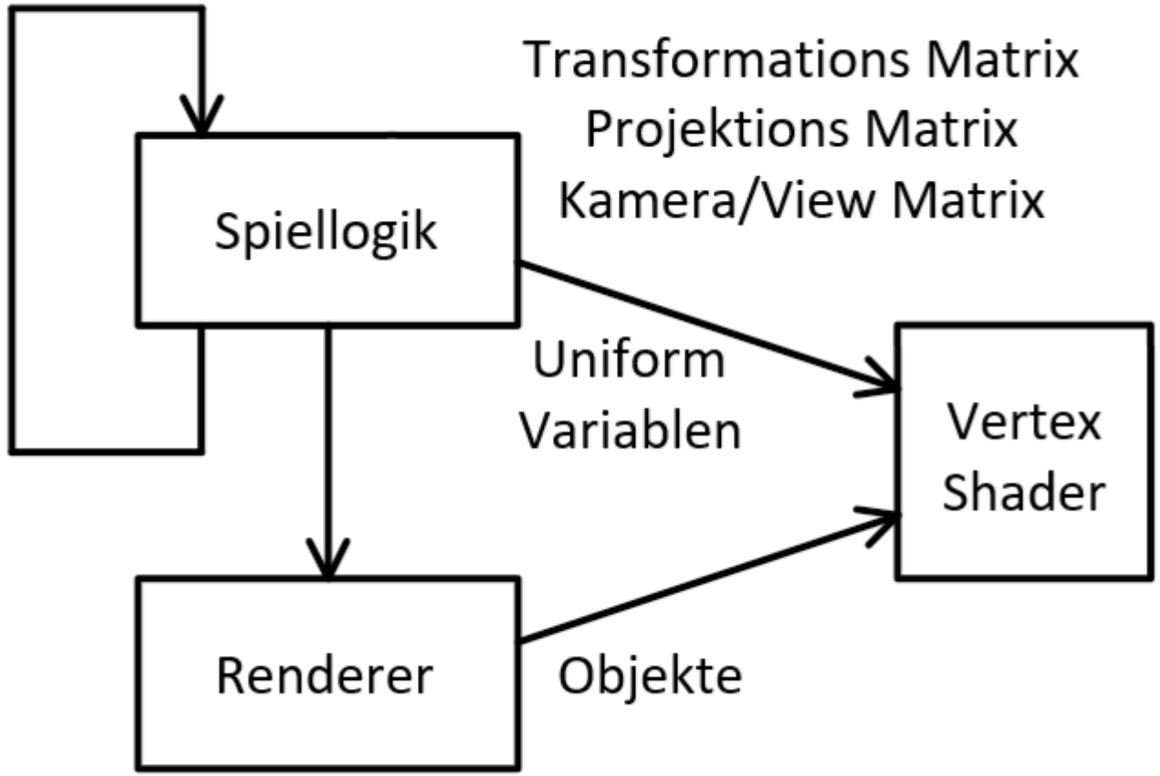


Uniform Variablen

- Objekte auf die GPU zu laden kostet Zeit
- Veränderung am Objekt erfordert
  Erneut es erneut in der GPU zu laden

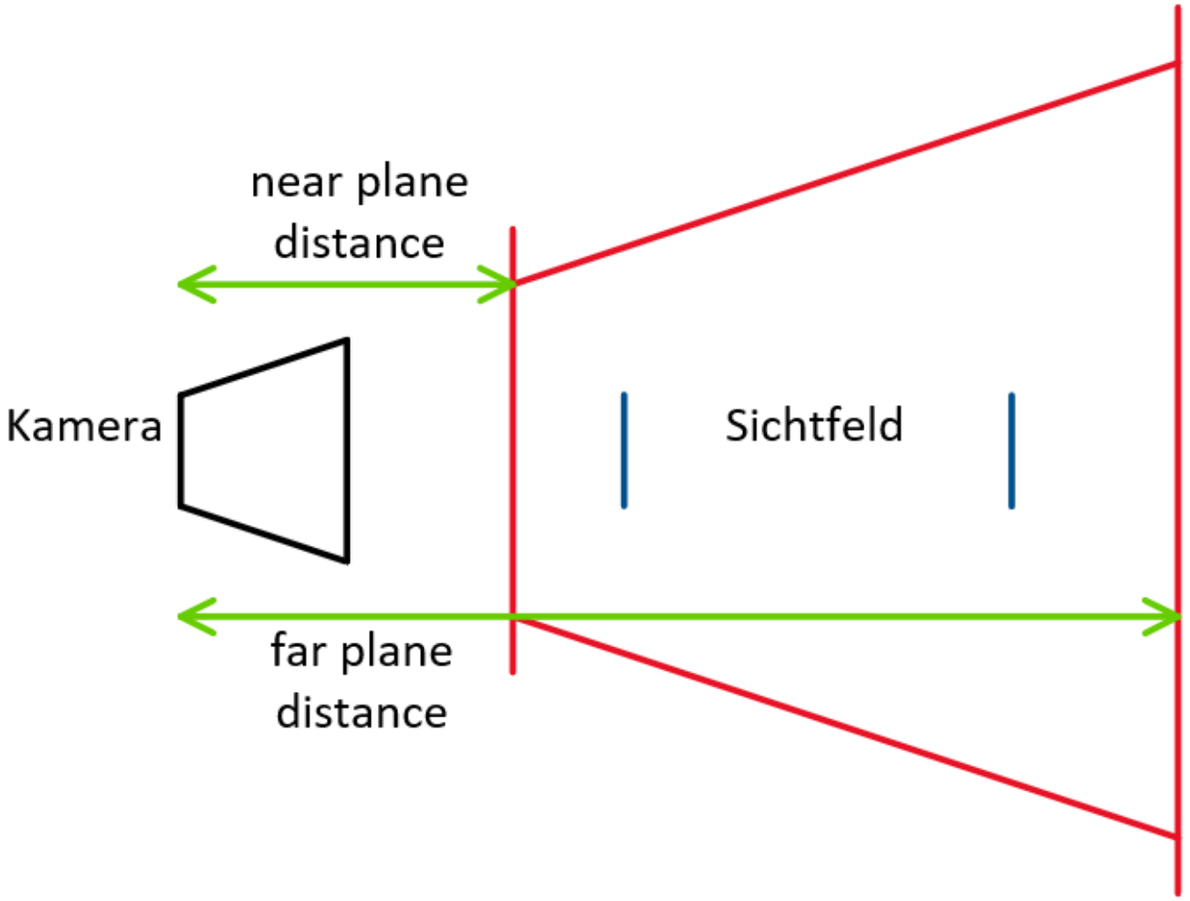
➢ Uniforme Variablen:

- Können von der Spiellogik
  manipuliert werden
- Der Vertex Shader ließt sie aus



Projektions Matrix

- Grundzustand:
  - Nahe und ferne Objekte sehen gleich aus
- Spannt erst dreidimensionales Sichtfeld auf

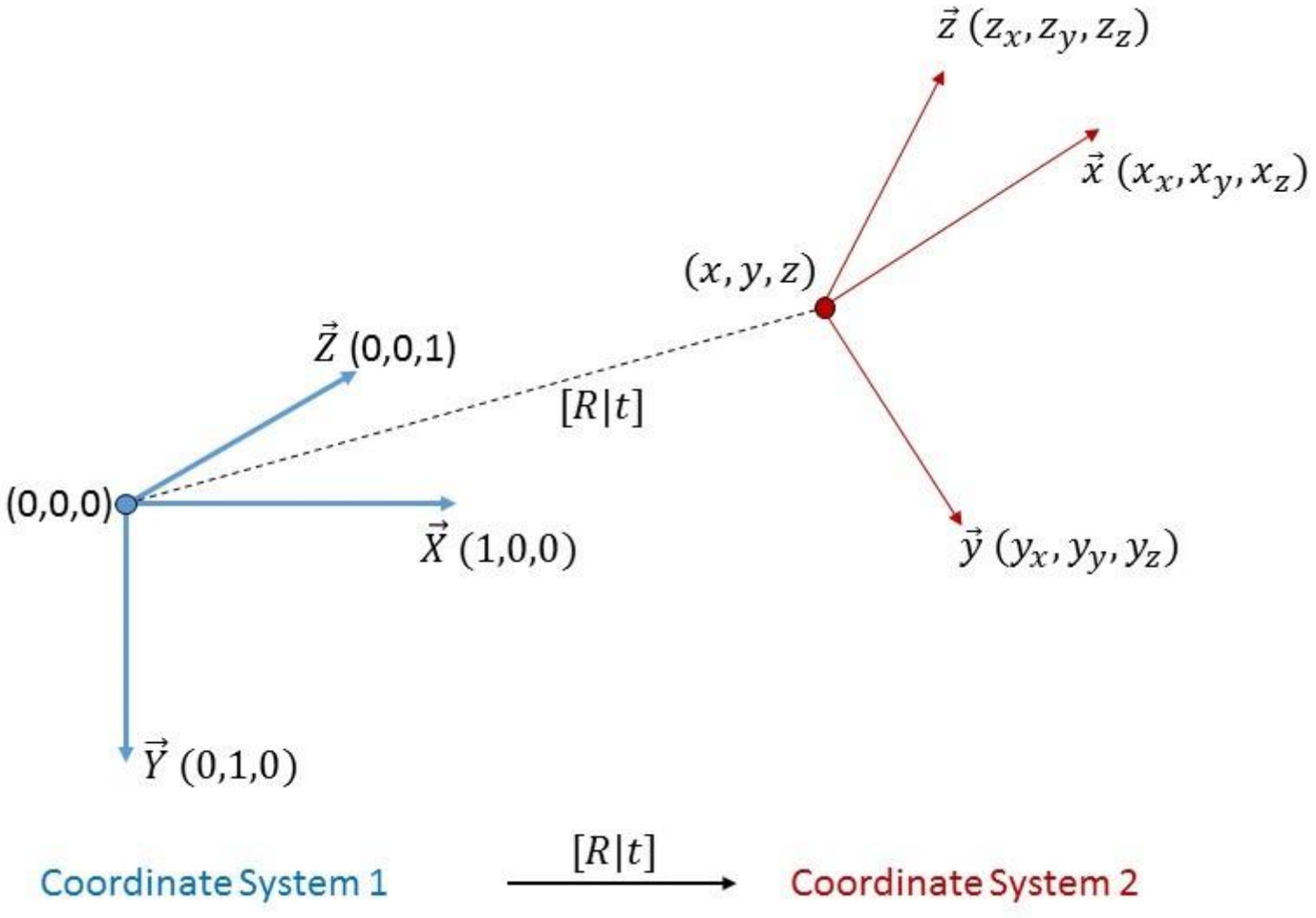


- $p = Mv$

Translations Matrix

- Jedes Objekt besitz:
  - xyz-Position
  - xyz-Orientierung/Rotation
  - Größe/Skalierung
- Ergibt Translation
- Matrix Repräsentation:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

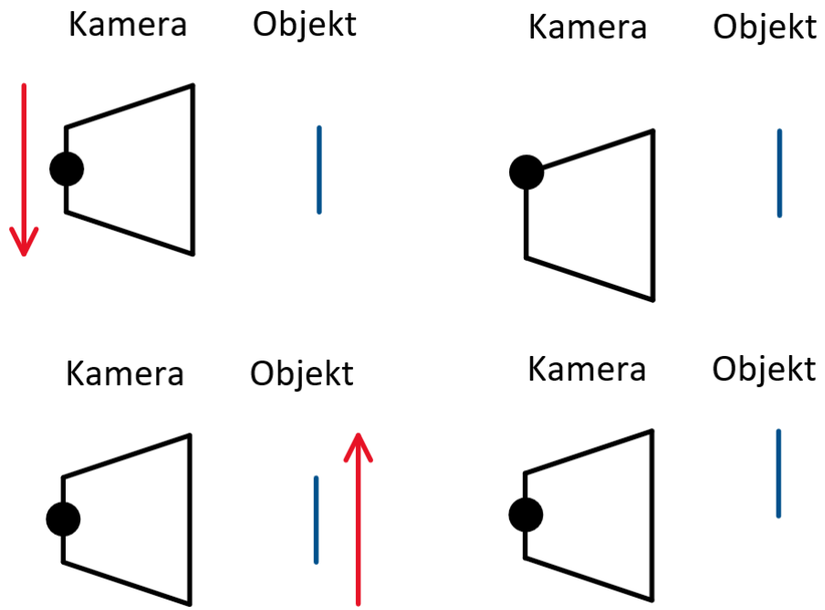


https://www.youtube.com/watch?v=09I15RO49vg

https://stackoverflow.com/questions/34391968/how-to-find-the-rotation-matrix-between-two-coordinate-systems

View Matrix

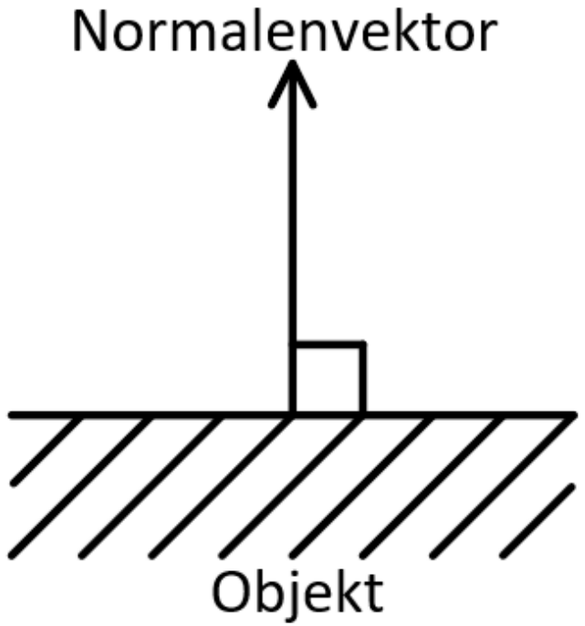
- Modelliert Bewegung der Kamera

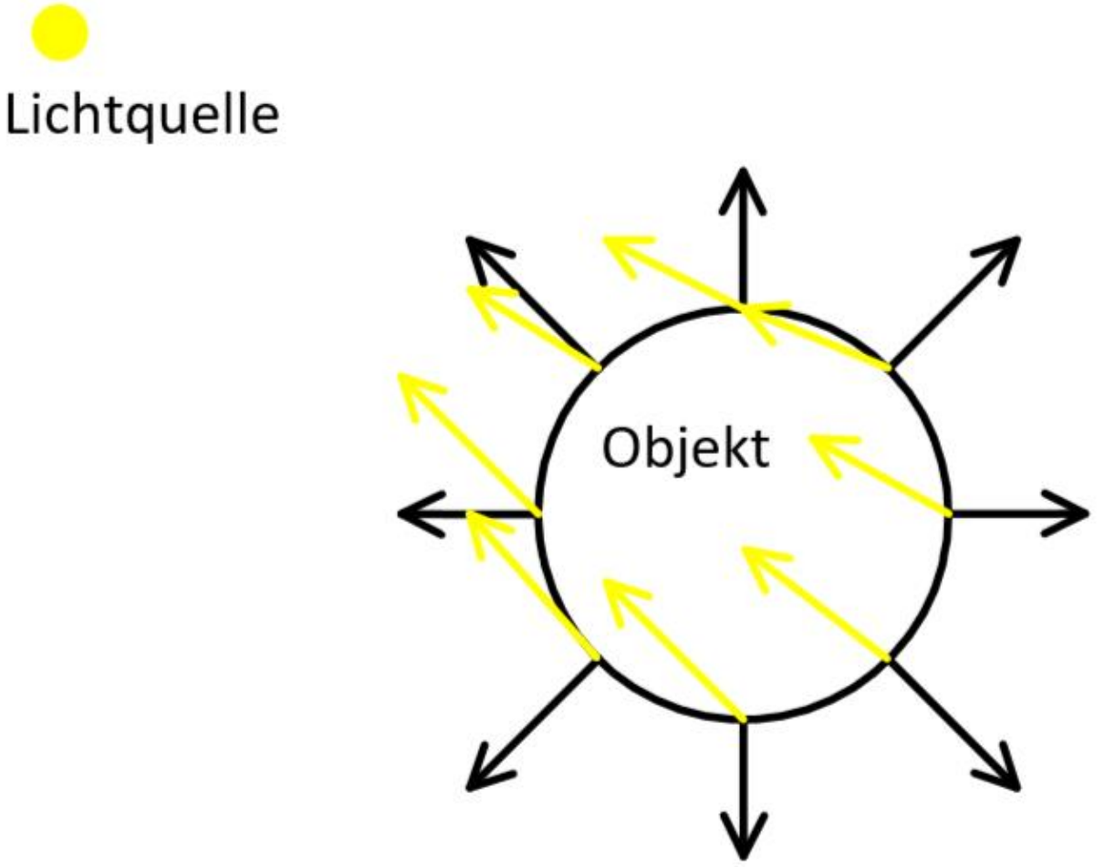


- Die Kamera wird nach rechts bewegt
  - ➢ die Objekte in der Welt/die Welt muss (umgekehrt, also) nach rechts bewegt werden
- View Matrix repräsentiert die Inverse der Kamerabewegung

Licht

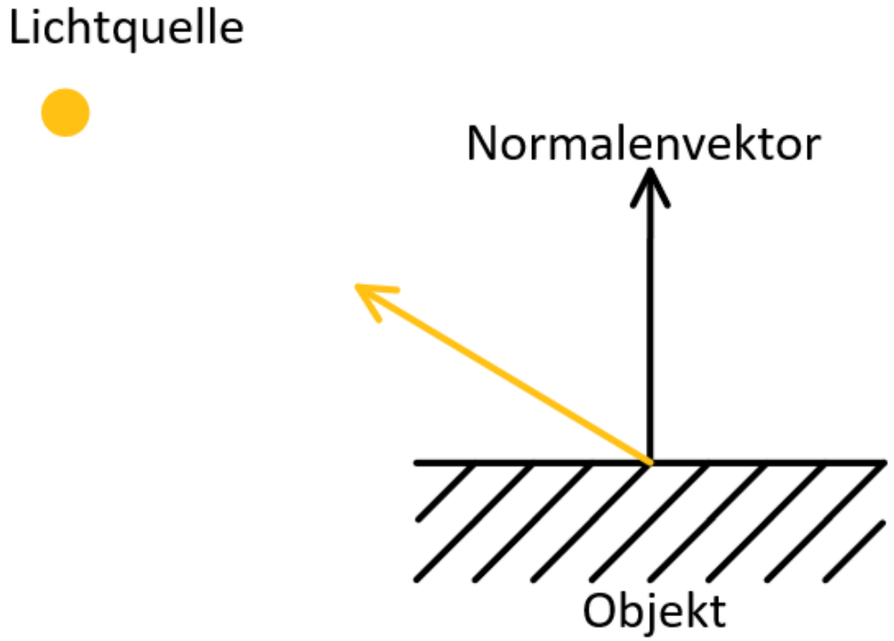
- Benötigt werden die drei Vektoren:
  - Normalenvektor
  - Lichtvektor
  - Kameravektor



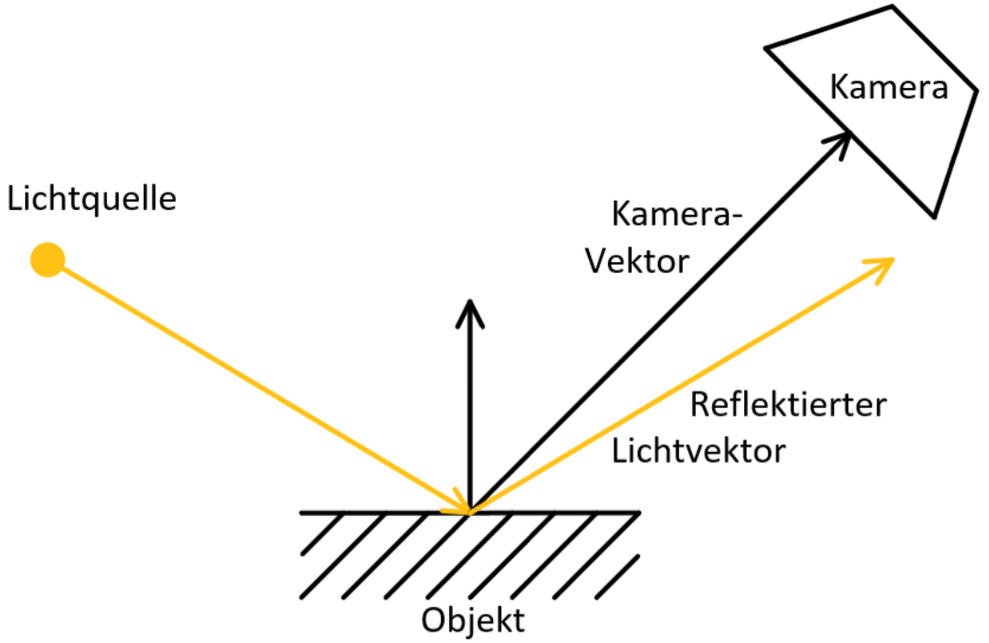


Licht

- Diffuse Lighting:
- Skalarprodukt zwischen Normalen- und Lichtvektor liefert Helligkeit



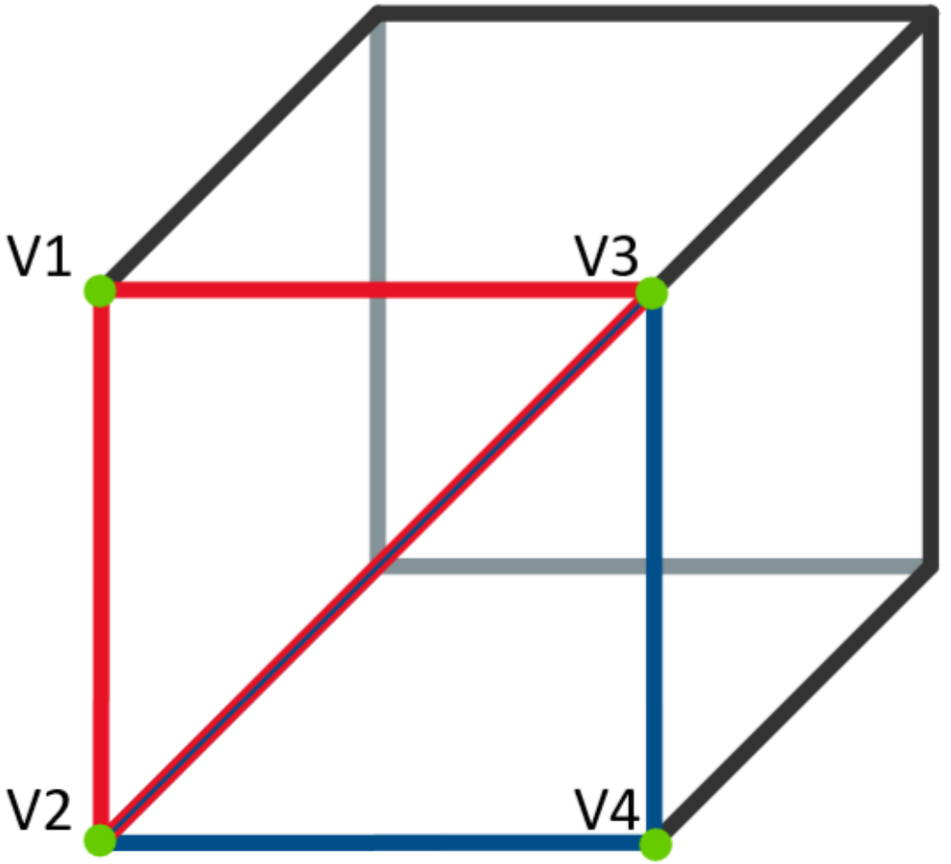
- Specular Lighting:
- Skalarprodukt zwischen Kamera- und reflektiertem Lichtvektor liefert Helligkeit



Index Buffer

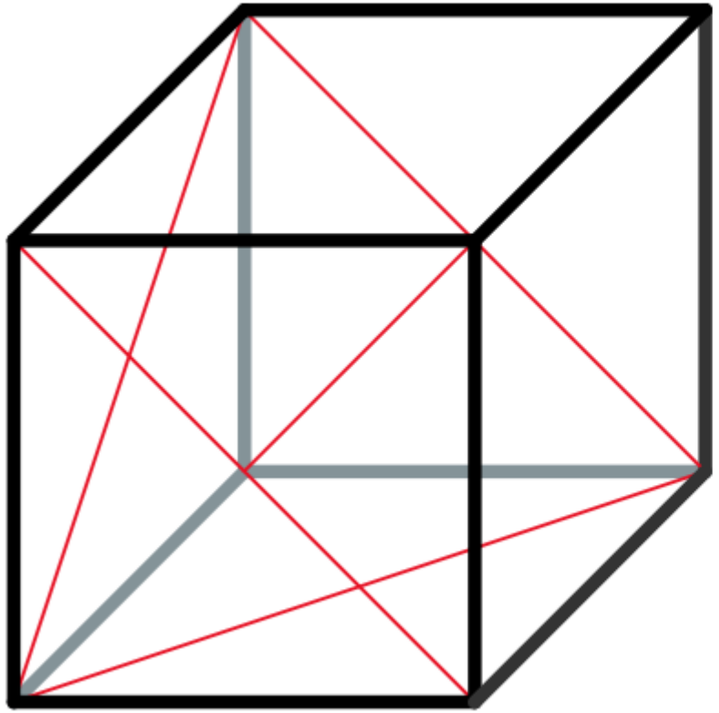
- Objekte = Liste von Polygonen
- Polygone = Liste von Eckpunkten
- Vorderwand: ((V1, V2, V3), (V2, V4, V3))

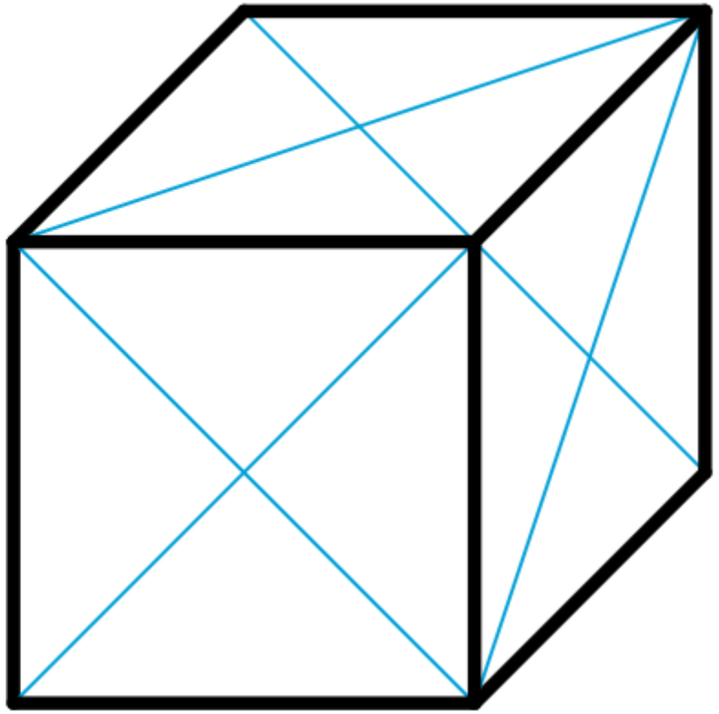
- Index Buffer indiziert Liste von Eckpunkten
  - (V1, V2, V3, V4), ((0,1,2),(1,3,2))
- Speicherverbrauch eines Eckpunktes:
  - 3 Floats + 3 Floats + 2 Floats = 8 Floats
- 6 x 8 = 48 <-> 4 x 8 + 6 = 36
- Spart ca. 30-50% Speicherplatz pro Objekt



Back Face Culling

- Standardmäßig werden alle Polygone gerendert

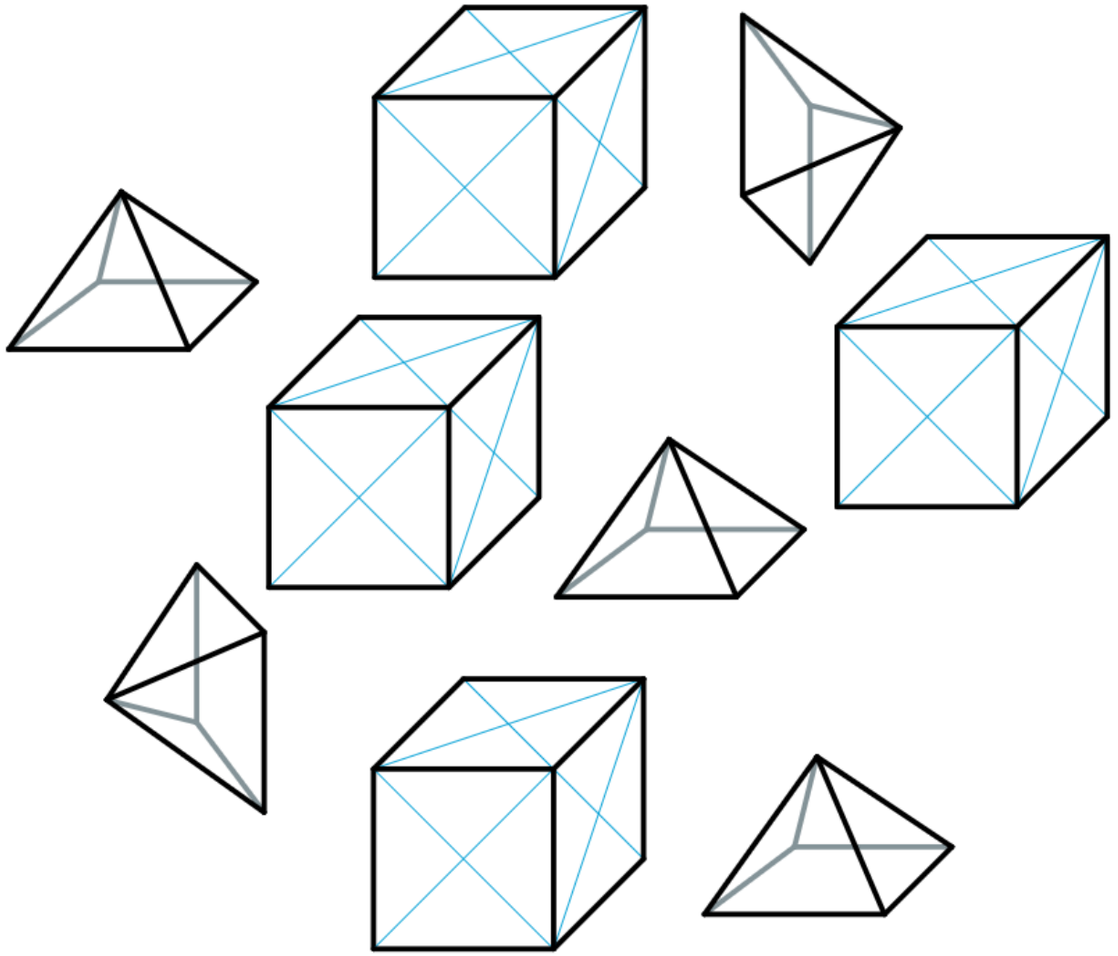




- Verdeckte Polygone können erkannt und dann nicht gerendert werden

Batch Rendering

- Problem:
  - Immer wieder werden Objekte des selben
  - Typs in die GPU geladen
- Lösung:
  - Alle Entitäten einer Objekt klasse werden in eine Liste eingetragen
  - Alle Entitäten einer Liste werden direkt nach einander gerendert

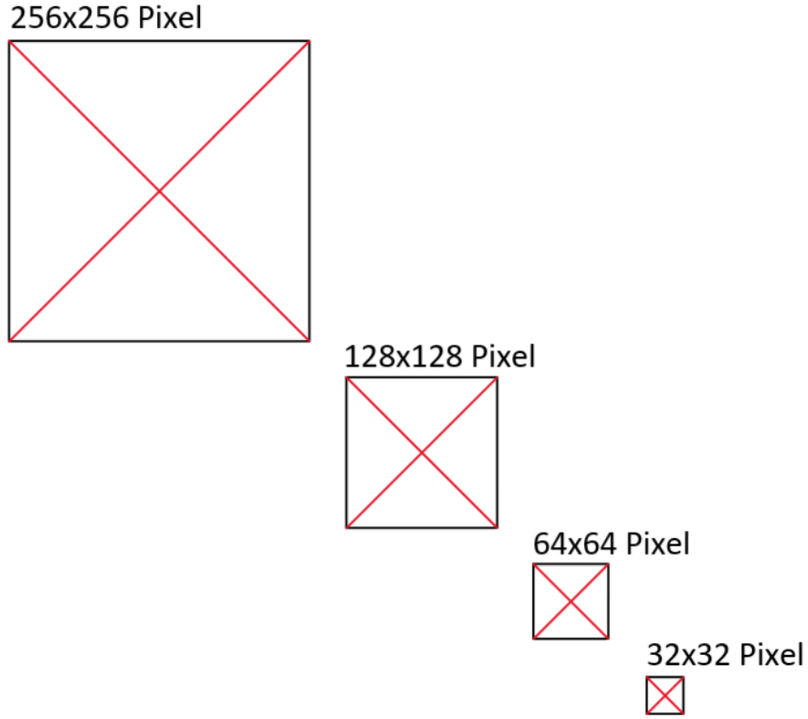


Mipmapping

- Problem:
  - Hoch auflösende Texturen benötigen mehr Speicherplatz und Rechenzeit
  - Weit von der Kamera entfernte Objekte flimmern
- Lösung:
  - Für weit entfernte Objekte werden herunterskalierte Texturen verwendet

➢ Bessere Performance

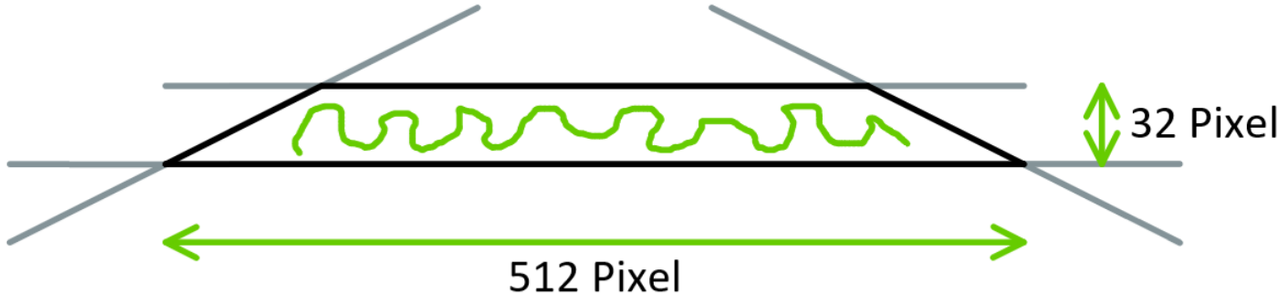
➢ Weit entfernte Objekte sehen besser aus

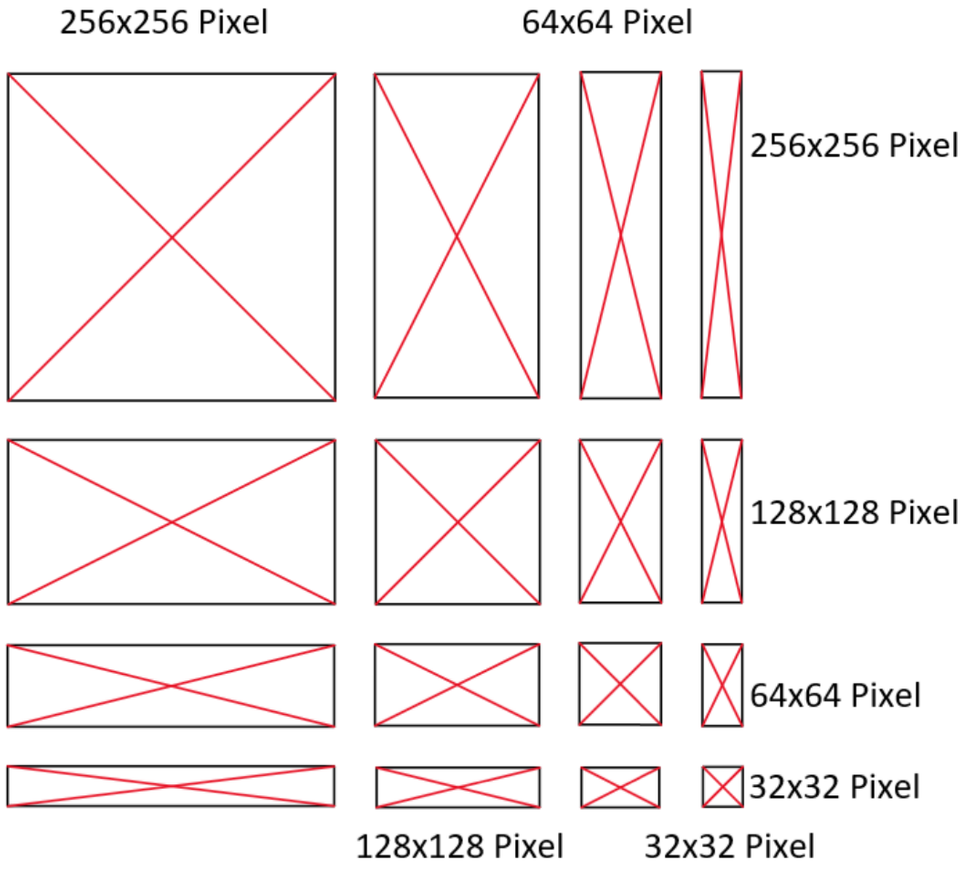


Anisotropic Filtering

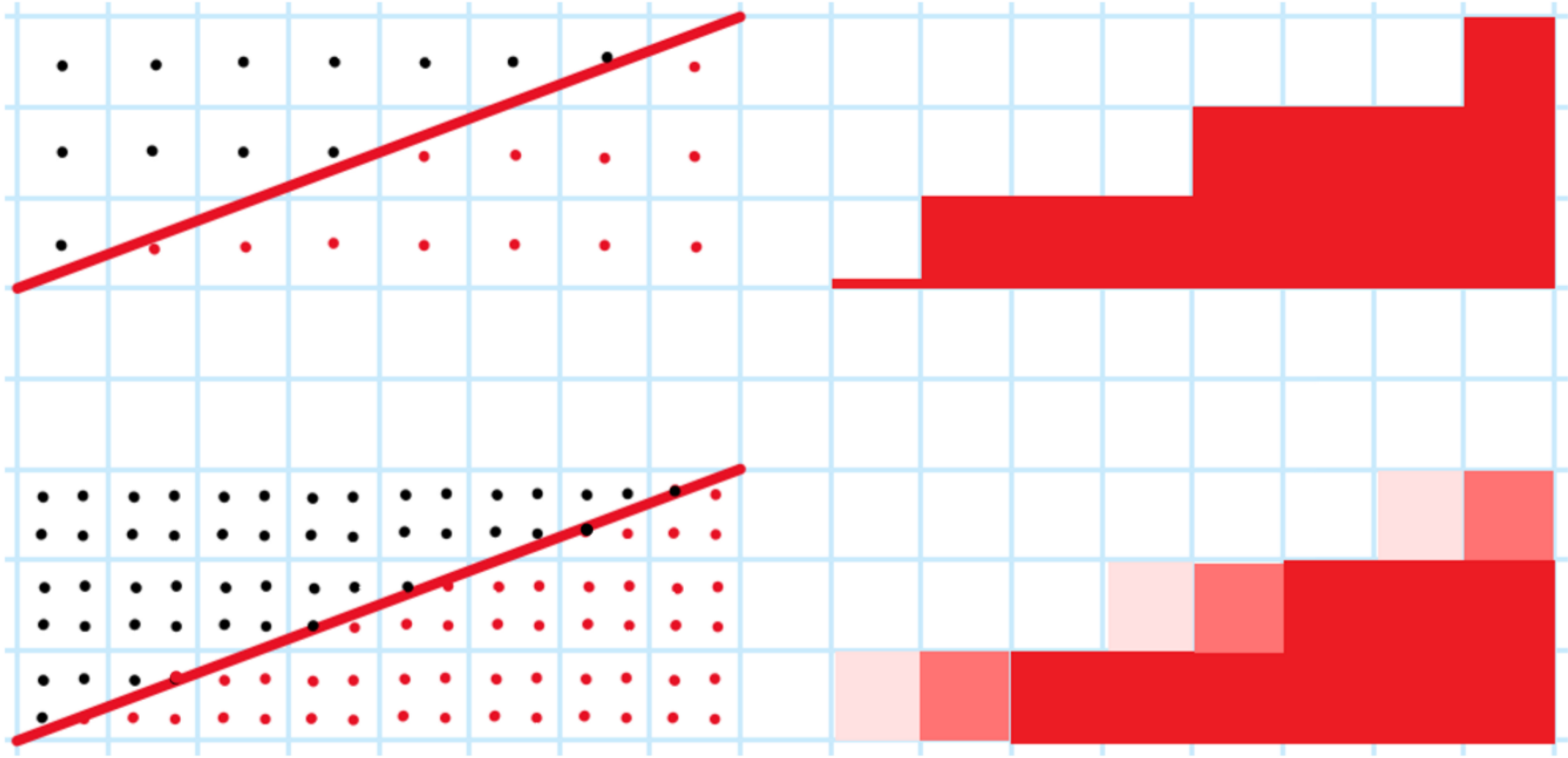
- Problem:
  - Bei Flachen Kamerablickwinkeln auf Objekte sind die Oberflächen verschwommen
  - Niedrig auflösende Textur wird hoch skaliert
- Lösung:
  - Texturen werden auf verschiedene
  - Seitenverhältnisse herunter skaliert

➢Keine Unscharfheiten bei flachen Blickwinkeln mehr



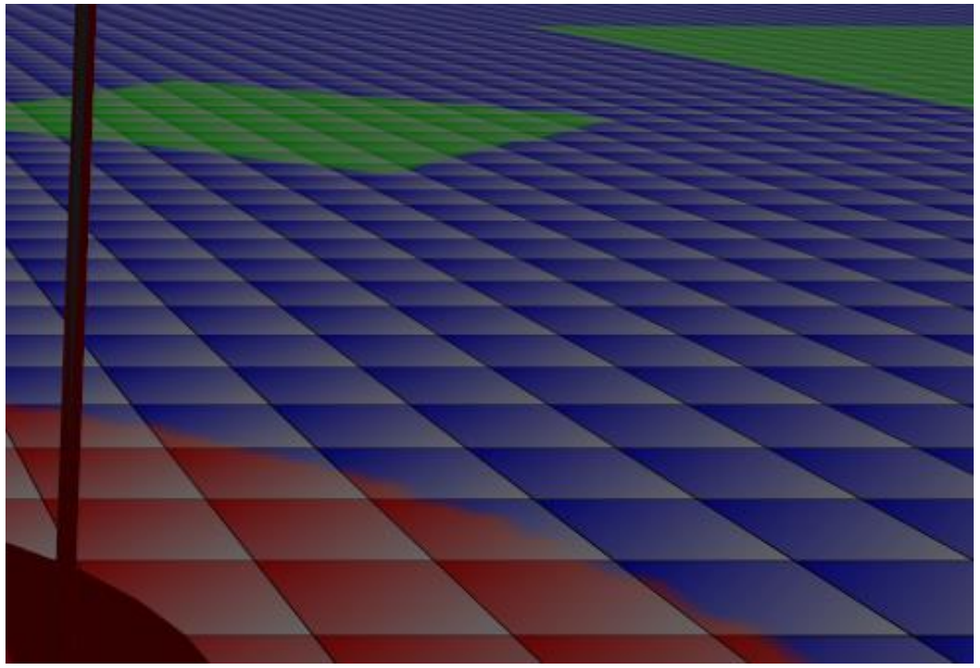


Antialiasing

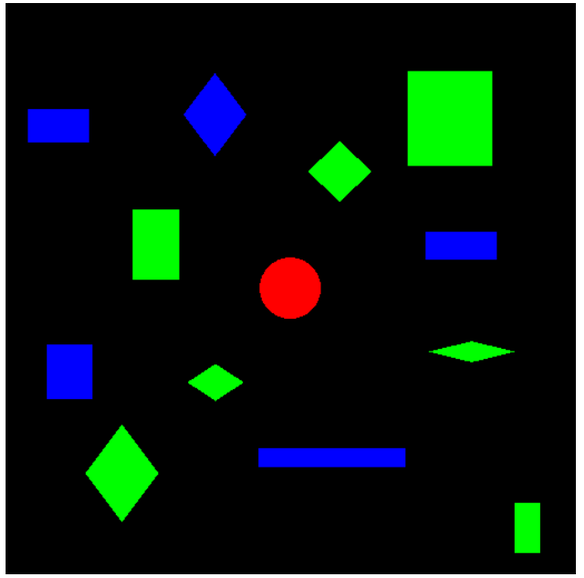


Tiling + Multitextureing

- Problem:
  - Werde hoch auflösende noch hoch skalierte Textur ist zielführend
- Lösung:
  - Selbe Textur immer wieder wiederholen

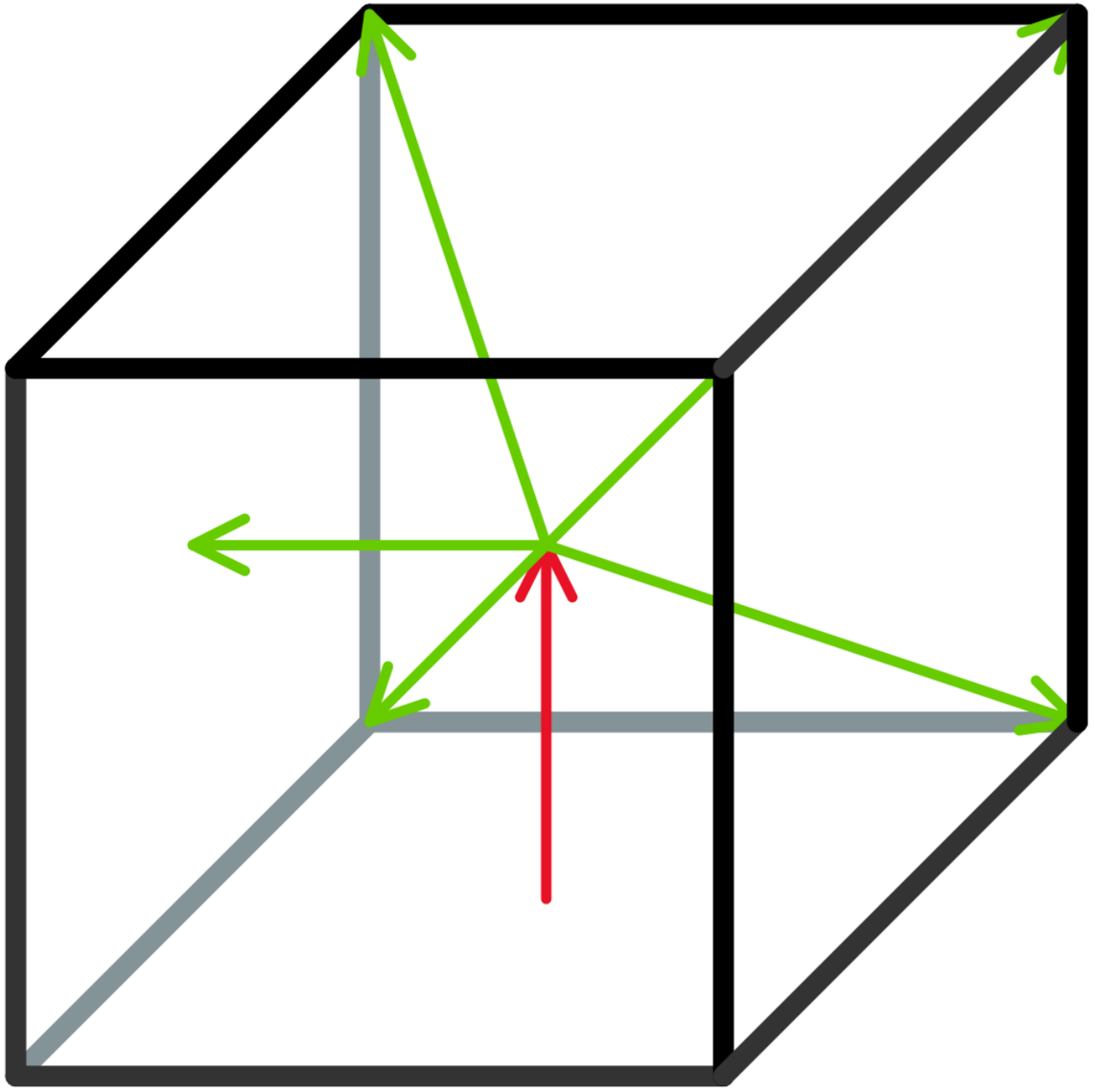


- Problem:
  - Verschiedene Texturen sollen für die selbe Oberfläche verwendet werden
- Lösung:
  - Eine Blendmap spezifiziert wo welche Textur anzuwenden ist



Skybox

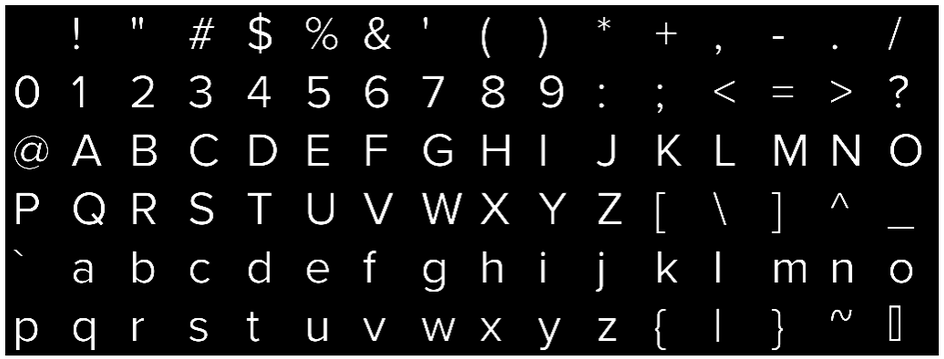
- Konstruierter Würfel stellt Himmel dar
- 6 Bilder konstruieren Cube Map
- Cubemap liefert Textur
- 3-dimensionale Blickrichtung liefert
- Rgb-Farbwert der Cubemap

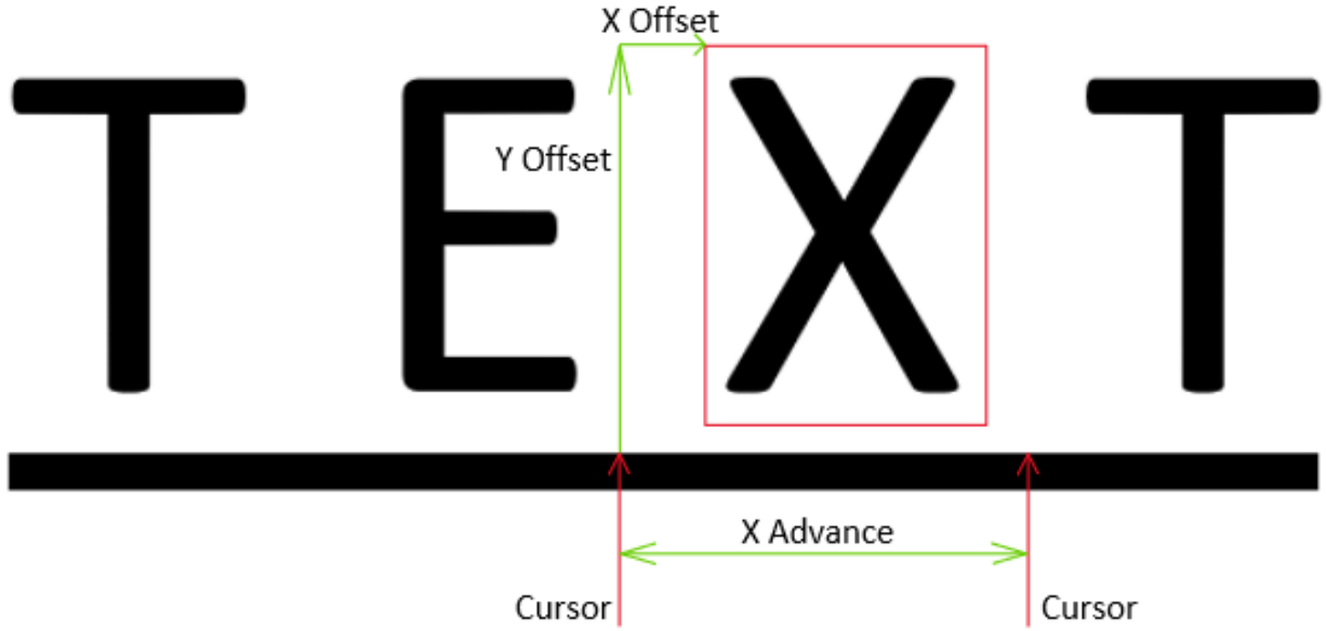


Font Rendering

- Texture Atlas enthält Zeichen
- Shader sampelt korrespondierende Textur
- Bekannt sein müssen:
  - X und Y Position im Texture Atlas
  - Höhe und Breite im Texture Atlas
  - X und Y Offset
  - X Advance

➢ Font File





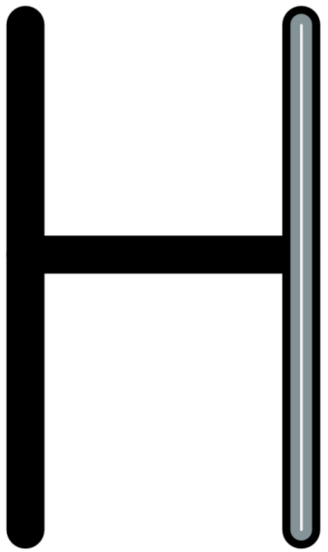
https://www.dreamler.com/blog/font-rendering/

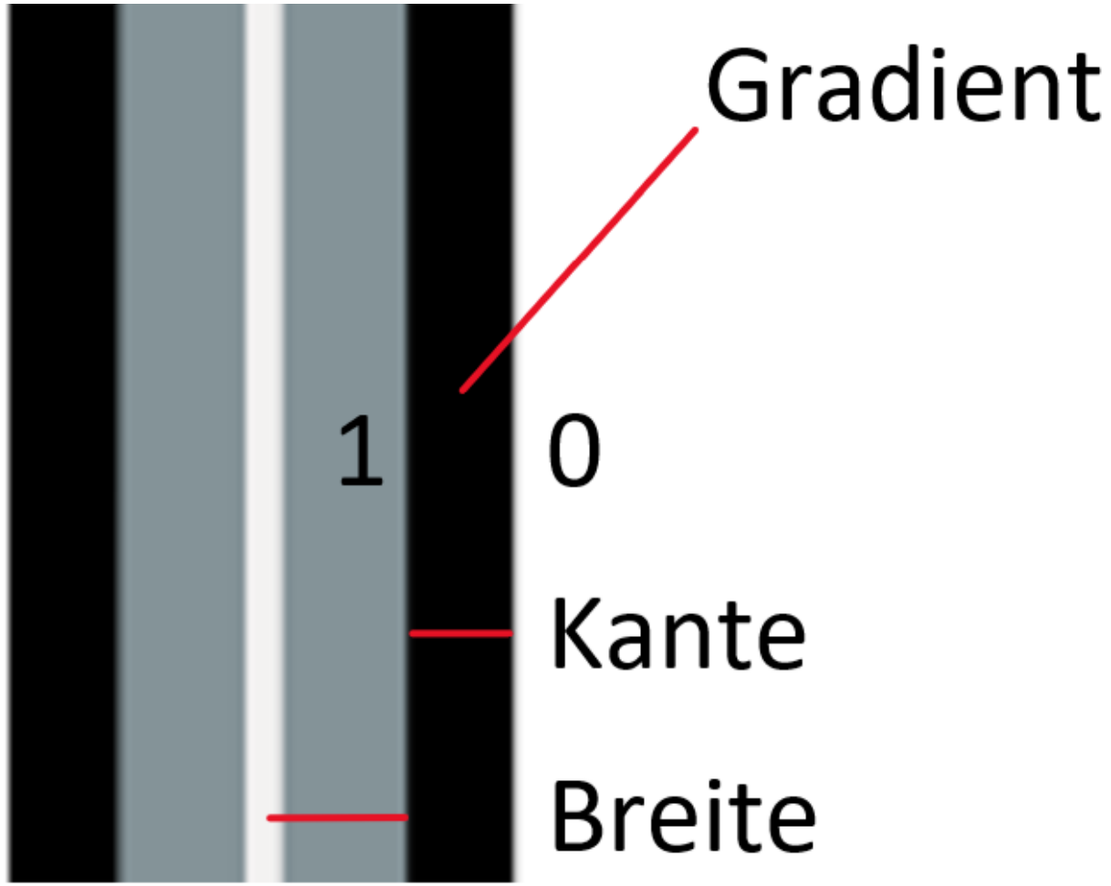
Font Rendering

- Grundproblematik:
  - Verschieden hohe Auflösungen

➢ Signed Distance fields ermöglichen:

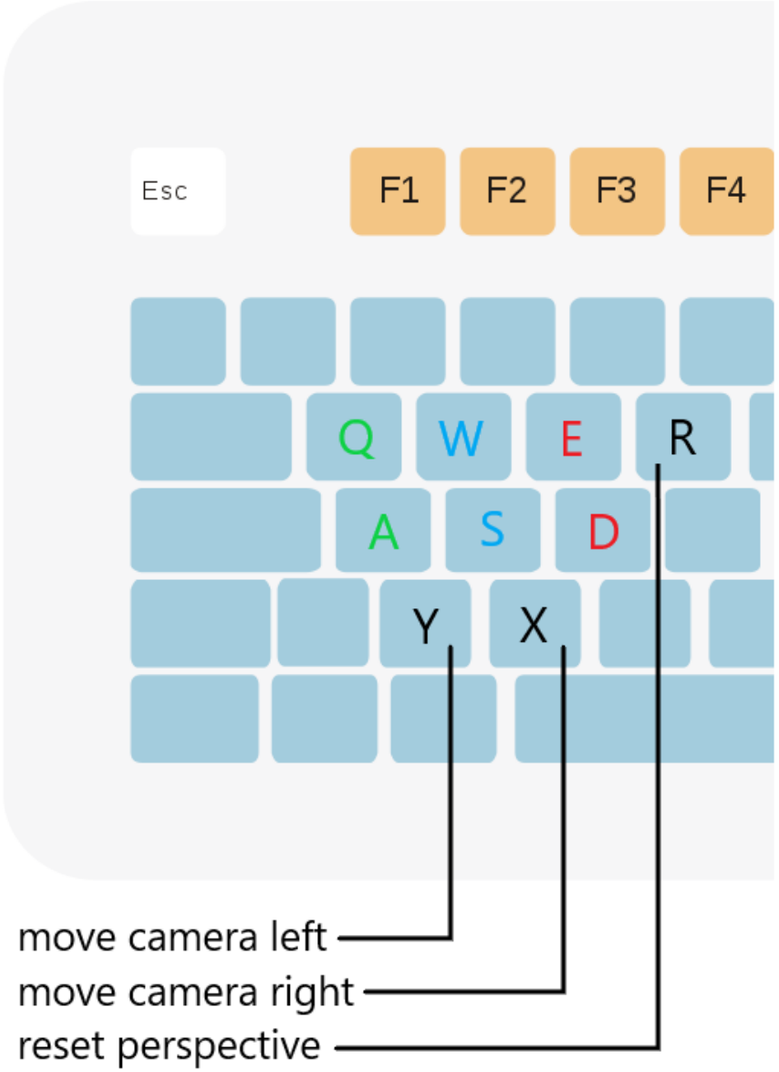
- Modellierbare Skalierung
- Effekte wie Umrandung, Leuchten, Schatten

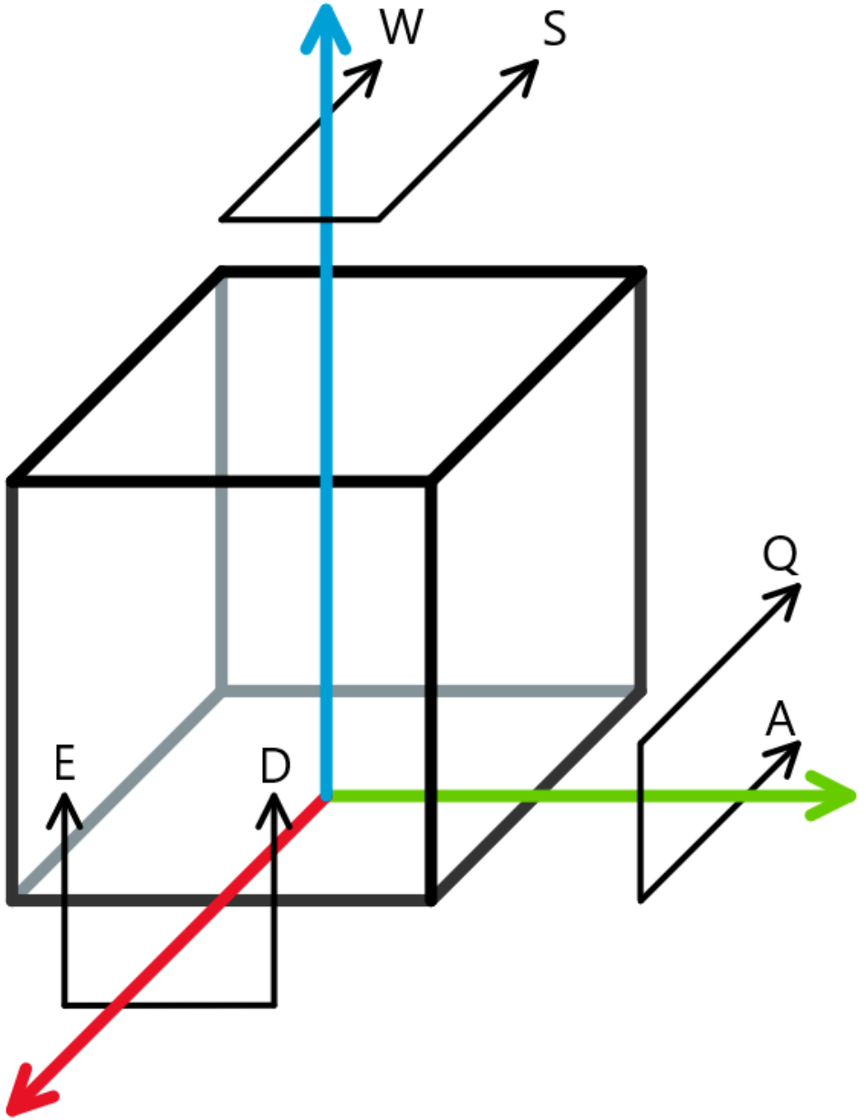


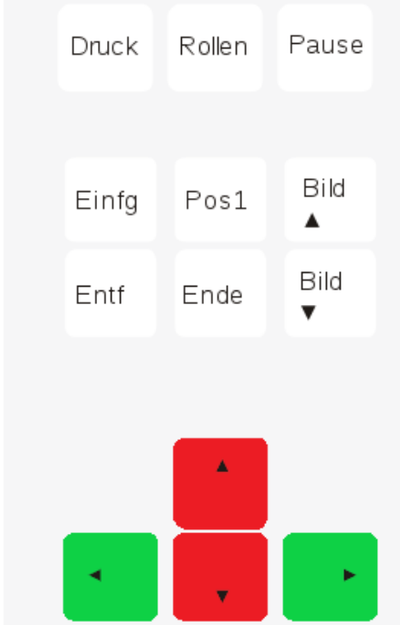


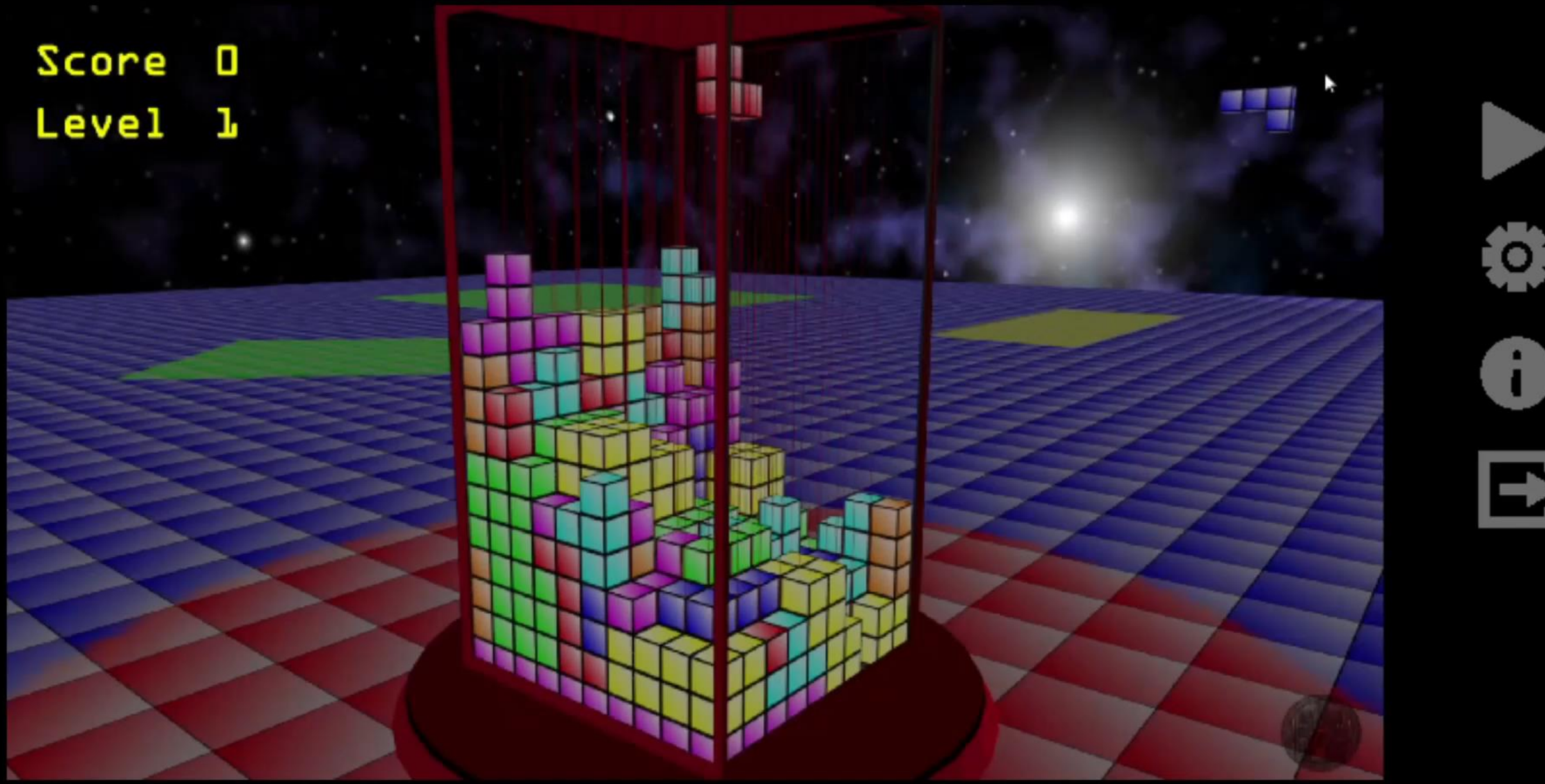
https://lambdacube3d.files.wordpress.com/2014/11/sdf-corners.png

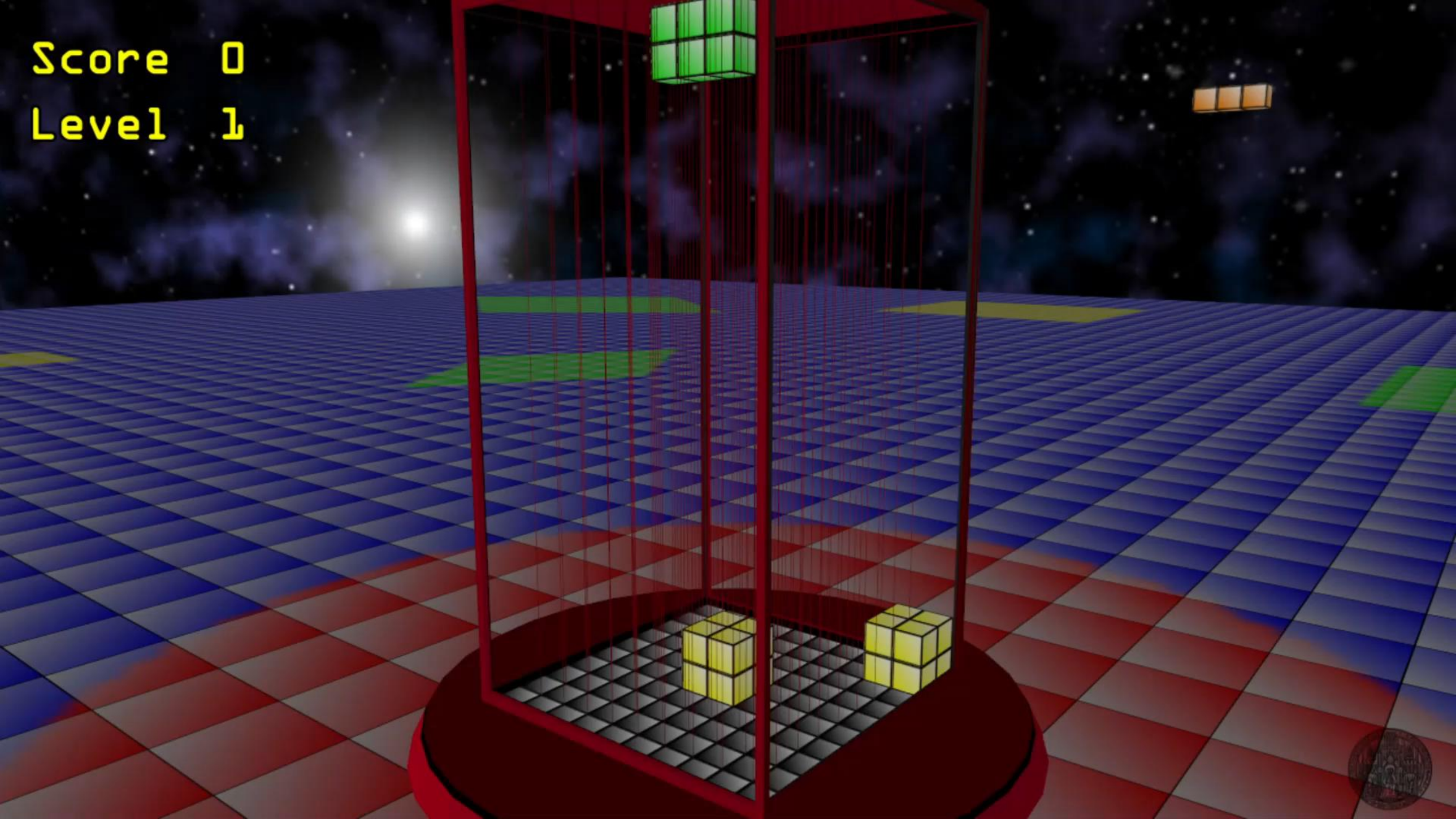
Steuerung

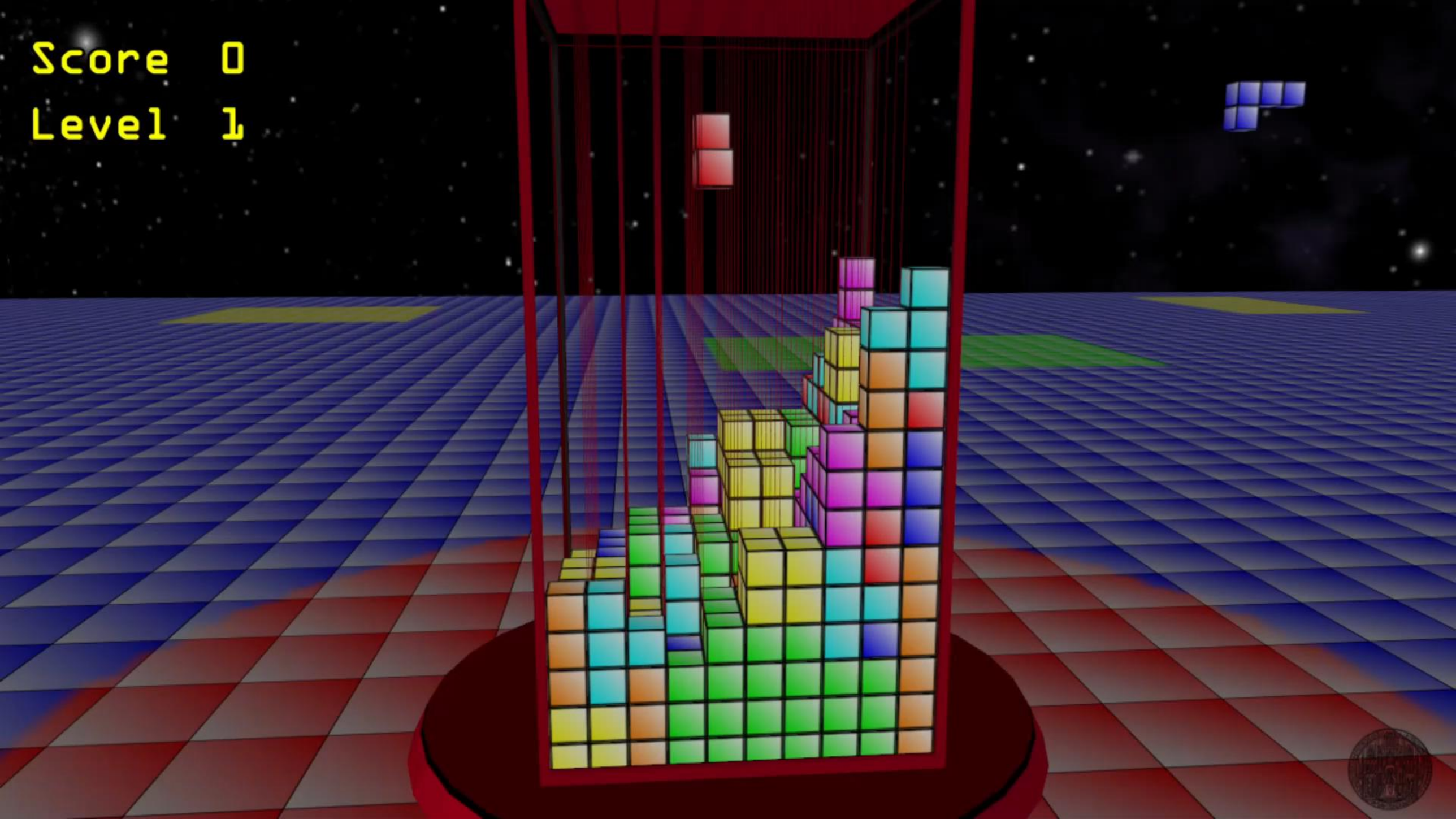

Esc
F1
F2
F3
F4
Q
W
E
R
A
S
D
Y
X
move camera left
move camera right
reset perspective


W
S
Q
A
E
D


Druck
Rollen
Pause
Einfg
Pos1
Bild ▲
Entf
Ende
Bild ▼


Score 0
Level 1


Score 0
Level 1


Score 0
Level 1

Ausblick

- Steuerung
- Fensterauflösung
- Detailliertere Modelle
- Schönere Texturen
- Modultypen
- Partikel
- Soundeffekte
- Highscores

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Vielen Dank für Ihre Aufmerksamkeit!

Einzelnachweise

- [1] https://www.lwjgl.org/
- [2] https://www.opengl.org//