Prof. G. Zachmann
Roland Fischer
Andre Mühlenbrock

Universität Bremen
Fachbereich 3
Arbeitsgruppe CGVR
13. November 2022

Wintersemester 2022/23

Übungen zu Computergrafik - Blatt 4

Abgabe am 20.11.2022, 23:59 Uhr

In diesem Aufgabenblatt beschäftigen wir uns mit Shader, die in OpenGL in der Programmiersprache GLSL implementiert werden. Ladet euch dazu das ShaderFramework herunter.

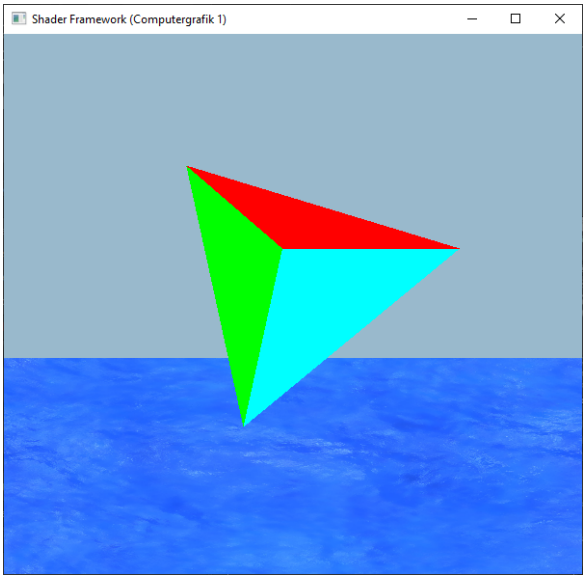

Abbildung 1: Das Fenster des Shader-Frameworks

Aufgabe 1 (Vertex Shader, *4 Punkte*)

In der `main.cpp` wurde ein Tetrahedron-Mesh mit dem Variablennamen `tetrahedron` definiert. Während die Positionen des Tetrahedrons korrekt angegeben wurden, fällt Dir auf, dass die Farben des gerenderten Tetrahedrons irgendwie falsch aussehen, wenn Du das Programm öffnest. Nach einiger Recherche bemerkst Du, dass die Farben nicht in Rot-, Grün und Blau-Werten angegeben wurden, wie OpenGL sie erwartet, sondern dass in den Farben andere Werte kodiert sind. Das Ziel der Aufgabenstellung ist es, die Farben so umzuwandeln, sodass sie korrekt gezeigt werden.

Stelle Dir vor, dass der C++ Source Code nicht verfügbar ist und Du lediglich ein bereits kompiliertes Programm im Binärformat vorliegen hast – d.h. Du darfst den C++ Code **nicht** verändern. Du weißt, dass der Quellcode eines Shaders i.d.R. als externe Textdatei vorliegt und Shader erst zur Laufzeit während der Programmausführung geladen werden. Daher bleibt Dir nur die Möglichkeit, die Farben im Shader umzuwandeln.

Du erhälst den Hinweis, dass sich die kodierten Farbwerte wohl mit dem nachfolgenden Algorithmus in Rot-, Grün und Blau-Werte umwandeln lassen, sodass OpenGL sie korrekt anzeigt:

When $0 \leq \mathrm{H} < 360$, $0 \leq \mathrm{S} \leq 1$ and $0 \leq \mathrm{V} \leq 1$:

$$C = V \cdot S$$

$$X = C \cdot (1 - |(H/60^\circ) \mod 2 - 1|)$$

$$m = V - C$$

$$(R', G', B') = \begin{cases} (C, X, 0) & \text{wenn } 0^\circ \leq H < 60^\circ \\ (X, C, 0) & \text{wenn } 60^\circ \leq H < 120^\circ \\ (0, C, X) & \text{wenn } 120^\circ \leq H < 180^\circ \\ (0, X, C) & \text{wenn } 180^\circ \leq H < 240^\circ \\ (X, 0, C) & \text{wenn } 240^\circ \leq H < 300^\circ \\ (C, 0, X) & \text{wenn } 300^\circ \leq H < 360^\circ \end{cases}$$

$$(R, G, B) = (R' + m, G' + m, B' + m)$$

Implementiere diesen Algorithmus im **Vertex-Shader** `tetrahedron.vert`, sodass genau diese Umwandlung der Eingabefarbe stattfindet, bevor die Farbe an den Fragment-Shader übergeben wird.

Hinweis: Die Variablen `H`, `S` und `V` im Algorithmus sind im Eingabe-Farbvektor in genau dieser Reihenfolge in `x`, `y` und `z` gespeichert.

Aufgabe 2 (Fragment Shader, *3 Punkte*)

Neben den Tetrahedron wird auch eine Ebene mit einem Wasser-/Wellen-Bild im Fenster gezeichnet. Deine Aufgabenstellung ist, dieses Wasser-Bild zu animieren, indem Du den **Fragment-Shader** `plane.frag` bearbeitest. Dort sind die Funktionen `effect1` und `effect2` deklariert. Diese Funktionen erhalten jeweils eine Bildkoordinate und sollen diese verändert zurückgeben. Verändere diese Bildkoordinaten so, dass:

- `effect1` dafür sorgt, dass das Bild zu dem Betrachter hin bewegt wird.
- `effect2` dafür sorgt, dass das Bild abwechselnd in einigen Bereichen leicht gestreckt wird, während es gleichzeitig an anderen Bereichen leicht gestaucht wird – die sin- und cos-Funktionen sind dabei sehr nützlich.

Verwende zur Animation die Uniformvariable `time`. Achte zudem darauf, dass dei Animation nicht zu schnell abläuft bzw. "realistisch" aussieht.

Hinweis 1: Eine Bildkoordinate beschreibt im Fragment-Shader, welcher Pixel des Wasser-Bildes ausgelesen werden soll, allerdings in relativen Koordinaten – also normiert von 0 bis 1. Wenn man in einem $512\,x\,512$ *großem Bild im Fragment-Shader also die Farbwerte des Pixels (330, 500) auslesen möchte, würden die Bildkoordinaten, die man dafür braucht, ungefähr (0.65, 0.98) lauten.*

Hinweis 2: Wenn man Bildkoordinate außerhalb des normierten Bereich [0,1] verwendet, so werden diese Koordinaten automatisch auf dem Bereich [0,1] gemappt. Dies ist vergleichbar mit einer Modulo-Operation, sodass eine Bildkoordinate wie (1.03, 1.07) schließlich den gleichen Farbwert wie die Bildkoordinate (0.03, 0.07) zurückliefert.

Aufgabe 3 (Verständnisfragen zu Shader, *3 Punkte*)

a) Nehme an, die Grafikkarte führt die `main`-Funktion des Fragment-Shader zum Zeichnen eines Dreiecks 125.000 Mal pro Frame aus. Wenn wir nun die Höhe und Breite des Fensters verdoppeln und das Dreieck skaliert entsprechend mit, wie oft wird dann der Fragment-Shader zum Rendern des Dreiecks eines Bildes ausgeführt? Warum? (max. 2-3 Sätze)

b) Nehme an, die Grafikkarte führt den Fragment-Shader zum Zeichnen eines Dreiecks 125.000 Mal pro Frame aus und alle Eckpunkt des Dreiecks liegen in verschiedenen Ecken des Fensters. Wie groß ist die Höhe und Breite des Fensters unter der Annahme, dass das Fenster quadratisch ist? Warum? (max. 2-3 Sätze).

c) Wie oft wird der Vertex-Shader zum Zeichnen eines Dreiecks ausgeführt? Wie verändert sich diese Anzahl, wenn die Fenstergröße verdoppelt wird? Warum? (max. 2-3 Sätze).