Einführung in die Computergrafik

Wintersemester 2018/2019

Prof. Dr. Philipp Jenke

HAW HAMBURG | Department Informatik

Aufgabe 1

Dreiecksnetze

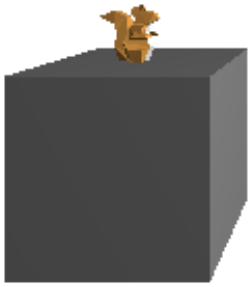

Abbildung 1: Die Spielerfigur (Eichhörnchen) steht auf einem Brick, das durch ein Dreiecksnetz dargestellt wird.

Machen Sie sich zunächst mit der Dokumentation für das Spiel und den vorgegebenen Klassen vertraut. Richten Sie dann das Framework unter IntelliJ (oder Eclipse) ein. Es handelt sich hier um ein Gradle-Projekt, Sie müssen sich also nach dem Importieren nicht weiter um das Auflösen der Abhängigkeiten kümmern.
In diesem Aufgabenblatt ist es Ihre Aufgabe, die Welt aus texturierten Würfeln zu zeichnen.
Schreiben Sie ein Plugin `WorldMeshGeneratorPlugin`, das die Würfelwelt des Spiels darstellt. Die Würfel werden als `Bricks` bezeichnet. Dazu müssen Sie zunächst ein Dreiecksnetz generieren. Für jeden Slot (Zeile, Spalte, Offset) in der Welt wird zunächst geprüft, ob dort ein Brick vorhanden ist. Falls ja, dann werden die notwendigen Vertices und Dreiecke für den Brick einem Dreiecksnetz hinzugefügt. Am Ende beinhaltet das Dreiecksnetz alle notwendigen Vertices und Dreiecke.

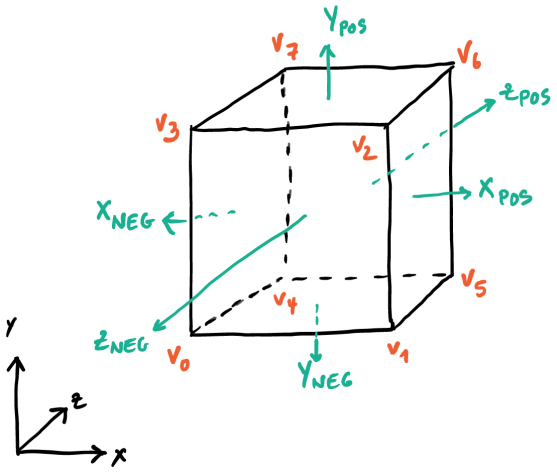

Abbildung 2: Für jeden Würfel werden 8 Eckpunkte und (zunächst) 6 Seiten, also 12 Dreiecke benötigt. Die Vertices sind mit $v_0 \dots v_7$ durchnummeriert, die Seiten haben Bezeichner: $X_{NEG}, X_{POS}, Y_{NEG}, Y_{POS}, Z_{NEG}, Z_{POS}$

Die Vertices jedes Brick-Würfels ergeben sich aus den Einheitspositionen:

```
private Vector CORNER_POINTS[] = {
  new Vector(0,0,0), new Vector(1,0,0),
```

```
new Vector(1,1,0), new Vector(0,1,0),
new Vector(0,0,1), new Vector(1,0,1),
new Vector(1,1,1), new Vector(0,1,1)};
```

Diese müssen für jeden Brick verschoben und skaliert werden:

$$v_i = ll + CORNER\_POINTS[i] \cdot s,$$

wobei ll die Position der linken unteren Ecke des Bricks ist (kann bei `World` abgefragt werden) und s die Kantenlänge eines Bricks angibt (kann ebenfalls bei `World` abgefragt werden).
Aus dem zweidimensionalen Array `FACET_INCIDES` können Sie die Vertices entnehmen, die für eine Seitenfläche benötigt werden:

```
private int[][] FACET_INCIDES = {
    {4, 0, 3, 7}, {1, 5, 6, 2},
    {4, 5, 1, 0}, {3, 2, 6, 7},
    {0, 1, 2, 3}, {5, 4, 7, 6}};
```

Beispiel: Die Seitenfläche Y_{NEG} hat den Index 2 im Enum `Brick.Side`. Daher müssen die Vertices mit den Indices `FACET_INCIDES[2] = {4, 5, 1, 0}` verwendet und durch zwei Dreiecke repräsentiert werden.
Viele der Seiten eines Bricks sind nicht sichtbar, weil Bricks nebeneinander angeordnet sind. Verändern Sie den Code so, dass nur Dreiecke für Seiten generiert werden, die nicht durch einen Nachbar-Brick bedeckt sind. Mit der Methode `hasNeighbor()` der Klasse `World` können Sie prüfen, ob eine solche Nachbarschaft zwischen zwei Bricks entlang einer Seite vorliegt.

Im Rahmen des Praktikums wird über die Aufgaben hinweg ein Jump'n'Run Computerspiel (Platformer) entwickelt. In jedem Aufgabenblatt entwickelt sich das Spiel ein wenig weiter. Das Spiel ist in das Framework für die Lehrveranstaltung integriert. Eine Anleitung zum Einrichten des Frameworks finden Sie in der Dokumentation zum Framework auf der EMIL-Seite zur Veranstaltung. Innerhalb des Frameworks ist bereits Funktionalität zum Spiel vorgegeben. Eine Dokumentation dieser Funktionalität findet sich im doc-Ordner des Packages für den Platformer.
Das Spiel kann sowohl auf dem Smartphone unter Android verwendet werden als auch in einer Desktop-Anwendung. Ich empfehle, im Praktikum auf die Desktop-Entwicklung zu setzen, weil sich diese viel einfacher und schneller debuggen und kompilieren lässt. Informationen zur Einrichtung finden Sie im doc*-Ordner des Frameworks.*