Universität Leipzig
Institut für Informatik
Abteilung für Bild- und Signalverarbeitung
Praktikum Computergrafik

SoSe 2024
Baldwin Nsonga
Lydia Müller

Aufgabenblatt 2

Aufgabe 1 - Wireframe-Rendering

Nutzen Sie Ihre in Aufgabenblatt 1, Aufgabe 2 erstellte Methode, um ein Wireframe-Rendering umzusetzen. Implementieren Sie hierfür den Rumpf der Methode *WireframeRenderer::renderScene* und rufen Sie die Methode in der Main-Methode in *main.cpp* auf. Es genügt hierbei, wenn Sie eine Kamera-unabhängige Parallelprojektion umsetzen und schlicht die z-Komponente der Vertices ignorieren, anstatt eine Projektion auf die Bildebene durchzuführen. Das gewünschte Ergebnis für das Modell *bunny_scaled.ply* ist in Abbildung 1 dargestellt.

Anmerkung: Leider ist das Modell nicht vollständig auf dem Bild zu sehen. Um dies zu ändern, benötigen wir Transformationen. Diese werden anhand der nächsten Aufgaben umgesetzt. Notwendig zum bestehen ist die Darstellung der Modelle mithilfe des in Aufgabenserie 1 implementierten Bresenham Algorithmus.

Aufgabe 2 - Multiplikation mit homogenen Koordinaten

Der zentrale Inhalt dieser Aufgabenserie ist es, Transformationen für die Modelle zu ermöglichen. Um dies zu implementieren, sind die folgenden Vorbereitungen nötig:

1. Informieren Sie sich zur Anwendung homogener Koordinaten in der Computergrafik.
2. Implementieren Sie in der *math.hpp* die Multiplikation einer *GLMatrix* ...
    - mit einem *GLVector*.
    - mit einem *GLPoint*.
    - mit einer *GLMatrix*.

Aufgabe 3 - Transformationen für die Modelle

Fügen Sie in *WireframeRenderer::renderScene* eine Matrixmultiplikation zur Transformation der Vertices hinzu. Die entsprechende Matrix sollte die Rotation, die Translation sowie die Skalierung beinhalten und für jedes Modell der Szene (*mModels*) unabhängig veränderbar sein. Sie soll dementsprechend Attribut der Klasse *Model* sein. Notwendig zum bestehen ist die korrekte Implementierung der Transformationen mithilfe in homogenen Koordinaten. Im Ergebnisbild sollen beide Objekte vollständig im Bild liegen und der Würfel muss erkennbar rotiert sein.

Default-Werte bei der Instanziierung eines Modells sollen sein: Keine Rotation, keine Translation, keine Skalierung. Schreiben Sie zusätzlich drei Methoden, mit denen sich die Rotation, die Translation und die Skalierung eines Modells setzen lassen und eine weitere Methode, die die Matrix eines Modells bei Veränderungen neu berechnet:

- Modell::setRotation(GLVector rotation)
- Modell::setTranslation(GLVector translation)
- Modell::setScale(GLVector scale)
- Modell::updateMatrix()

Anmerkung: Entscheiden Sie selbst, ob mehrfache Aufrufe von Transformations-Methoden gleicher Art (mehrfache Translationen, mehrfache Rotationen oder mehrfache Skalierungen) zur Verkettung mit vorher gesetzten Transformationen gleicher Art führen oder ob lediglich die zuletzt gesetzten drei Transformationen (Translation, Rotation und Skalierung) angewendet werden. Machen Sie sich zudem Gedanken dazu, in welcher Reihenfolge Sie die Transformationen auf das Modell anwenden.

Verwenden Sie die von Ihnen erstellten Setter in der Main-Methode in *main.cpp*, um das Bunny-Modell (*bunny_scaled.ply*) und den Cube (*cube_scaled.ply*) in einem Bild ungefähr gleich groß nebeneinander zu rendern. Fügen Sie zudem für mindestens eines der Modelle eine Rotation hinzu. Ein Beispiel für ein mögliches Ergebnis dieser Aufgabe finden Sie in Abbildung 2.

Anmerkung: Für dieses Rendering wurde der Cube um jeweils 150 Einheiten in x- und 200 Einheiten y-Richtung verschoben. Er wurde um 20 Grad um die x-Achse sowie 45 Grad um die y-Achse rotiert. Zusätzlich wurde der Cube um den Faktor 0.5 in der x- und z-Richtung skaliert. In y-Richtung wurde er um den Faktor 3 skaliert. Das Bunny-Modell wurde um 400 Einheiten auf der x-Achse und um 200 Einheiten auf der y-Achse verschoben, auf 200 Prozent skaliert und um 5 Grad um die y-Achse rotiert.

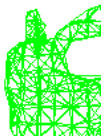

Abbildung 1: Zielsetzung des Wireframe-Renderings

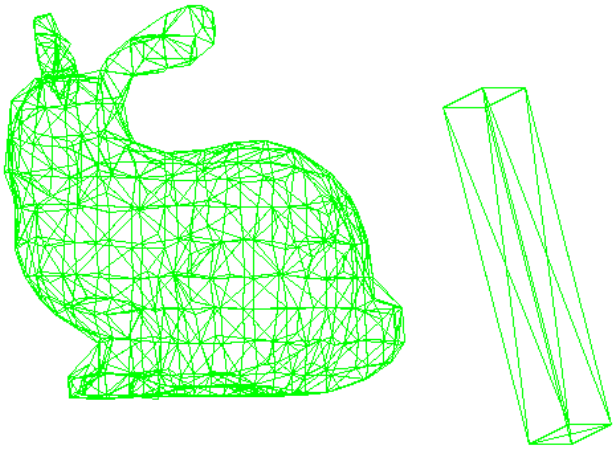

Abbildung 2: Zielsetzung der angewendeten Transformationen (Beispiel)