Dokumentation

CSE Projekt – Martin Steinberger

Bei dem Programm handelt es sich um eine schlichte Demo welche die Anforderungen unserer Übung beinhält. In den folgenden Punkten wird kurz erläutert wie die einzelnen Schritte behandelt wurden.

# Aufbau

Das Programm wurde mit OpenGL in Verbindung mit SDL geschrieben. Grafische Objekte werden im Programm als „Vertex Array Objects“ an die Shader geschickt, dort werden diese Transformiert und der Lichteinfall darauf berechnet. Der Code stammt zu einem großen Teil von <https://learnopengl.com/> da dort Ausgezeichnete Tutorials für die Verwendung von OpenGL in Verbindung mit Shadern vorgestellt werden. Mein Ansatz versucht die in den Tutorials beschriebenen Abläufe Objektorientiert umzusetzen daher ergeben sich einige Klassen welche für die Funktionalitäten des Programms zuständig sind.

## Mesh.h / Mesh.cpp

Die Mesh Klasse ist ein Container welcher Referenzen auf Vertex Array Objects, Texturen sowie eine Transformationsmatrix und Methoden um diese einfach zu bearbeiten enthält. Ich habe versucht die Mesh Klasse so gut wie möglich von OpenGL zu entkoppeln und alle OpenGL Funktionen in den Referenzierten MyVAOs unterzubringen.

Der Vector welcher die Referenzen auf VAOs (Vertex Array Objects) enthält stellt die verschiedenen LODs dar. Bevor ein VAO gezeichnet wird, wird die Entfernung von der Kamera berechnet und dementsprechend ein bereits auf der Grafikkarte hochgeladenes VAO gezeichnet. Bevor das VAO gezeichnet wird, setzt das Mesh-Objekt die Textur, Transformationsmatrix und Transparenz des VAO neu. So können mit demselben VAO auf der Grafikkarte optisch unterschiedliche Objekte gezeichnet werden.

## MyVAO.h / MyVAO.cpp

MyVAO ist eine Klasse welche die OpenGL Funktionen zum Zeichnen von VAOs in Methoden verpackt. Die Klasse lädt in ihrem Konstruktor die benötigten Daten auf die Grafikkarte. In der draw() Methode werden Transformationen, Texturen, Transparenz und Shader gesetzt mit welchen gezeichnet wird. Die Eigenschaften mit welchen das VAO gezeichnet wird können mit den Setter-Methoden der Klasse modifiziert werden.

## Shader.h / Shader.cpp

Eine Containerklasse welche Shader von der Festplatte lesen kann, diese Kompiliert, auf die Grafikkarte lädt und die Adresse davon in einer Membervariable speichert.

## Texture.h / Texture.cpp

Eine Containerklasse welche .dds Texturen von der Festplatte lesen kann, diese auf die Grafikkarte lädt und die Adresse davon in einer Membervariable speichert. Diese Klasse kann derzeit nur .dds Dateien lesen. Sie kann vorhandene Mipmaps dieser auslesen und erstellt diese korrekt. Der Compression-Level der .dds files wird auch ausgelesen und dementsprechend gesetzt.

## VertexShader.glsl

Vertexshader welcher für die Transformationen von VBOs und deren Normals zuständig ist. Kann derzeit Normals nicht richtig skalieren sofern diese nicht gleichmäßig skaliert werden.

## FragmentShader.glsl

Shader zuständig für die Farbe eines Fragments. Die Lichtberechnung nach Phong wird hier durchgeführt und mit Farbe und Textur kombiniert.

# Probleme

Dies war mein erster Versuch mit Shadern zu programmieren und das hat auch sehr gut geklappt, ein Problem konnte ich jedoch nicht lösen, die Transparenz. Zwar werden die Objekte je nach Alpha Wert mehr oder weniger opak gezeichnet doch konnte ich die Reihenfolge in der die einzelnen Dreiecke der VBOs gezeichnet werden nicht mehr beeinflussen nachdem die VBOs als ganze Objekte auf der Grafikkarte liegen. Mein Ansatz war daher die Objekte je nach Entfernung zur Kamera zu ordnen und von hinten nach vorne zu zeichnen. Die Transparenz von Objekten gegenüber anderen Objekten funktioniert so einwandfrei, jedoch werden die Dreiecke eines Objekts in sich in falscher Reihenfolge gerendert und daher Teilweise ausgeblendet.