|  |
| --- |
| C:\Users\holzer\Downloads\teapot.obj.png |
| OpenGL 3D Model Viewer  Konzept  **Modul: BZG1310 "Objektorientierte Geometrie" Dozent: Marx Stampfli Autoren: Michael Koch, Joel Holzer**  **Version: 1.0, 23.11.2015** |
| **Berner Fachhochschule**  Technik und Informatik  Informatik |

# Einleitung

Im Rahmen des Moduls „Objektorientierte Geometrie“ an der Berner Fachhochschule erstellen wir eine Java Applikation, welche die Möglichkeit bietet, eine 3D Figur darzustellen und die 3D Figur von unterschiedlichen Seiten zu betrachten. Diese Applikation soll ein GUI beinhalten, über welches .OBJ Dateien eingelesen und angezeigt werden können. OBJ ist ein offenes Dateiformat zum Speichern von dreidimensionalen geometrischen Formen. Die angezeigte Figur kann via Maus-Interaktion von allen Seiten betrachtet werden. Zudem ist es möglich, mit dem Mausrad rein und raus zu zoomen und so die 3D Figur grösser oder kleiner zu betrachten.

Zur Modellierung der 3D Figuren verwenden wir OpenGL und die Shadersprache GLSL. Wir benötigen eine Projektionsmatrix und eine ViewMatrix. Die Projektionsmatrix lässt weit entfernte Objekte (in Richtung Z-Achse) kleiner darstellen und nahe bei der Kamera liegende Objekte grösser. Für die Bewegung der Kamera benötigen wir eine Viewmatrix, welche die aktuelle Position der Kamera beinhaltet.

Optional, jedoch nicht Erfolgs-Relevant ist die Umsetzung von Phong Shading in unserer Applikation.

Wir generieren eine Lichtquelle und berechnen mittels Phong Shading das daraus resultierende Ambient, Diffuse und Spekular Light der Szene.

# Vorgehen

## Tasks

Nachfolgende Auflistung gibt die während der Umsetzung des Projekts anfallenden Tasks wieder, begonnen beim ersten Task. Beide Studierenden sind an der Arbeit aller Tasks beteiligt. Wie die Arbeit innerhalb des Tasks aufgeteilt wird entscheiden die Studierenden vor der Umsetzung des jeweiligen Tasks.

Vorgehen:

1. Einrichtung Entwicklungsumgebung (IntelliJ IDEA) mit GitHub
2. GUI Generierung mit GUI-Designer
3. Realisierung des Uploads für .Obj Dateien
4. Realisierung der Logik für Vertex Array Object (VAO) und Vertex Buffer Object (VBO)
5. Realisierung von Vertex und Fragment Shader
6. Realisierung der Projektion und Viewmatrix
7. Realisierung der Objekt oder Maus Rotiation
8. Finalisierung der Dokumentation
9. Präsentation und Abgabe

## Technologien & Tools

Für die Realisierung des Projektes setzen wir folgende Technologien ein:

|  |  |
| --- | --- |
| **Anwendungsbereich** | **Technologie** |
| Umsetzung des User Interfaces (GUI) | Swing, AWT |
| Umsetzung des .Obj File-Uploader | Java |
| 3D-Modelierung | OpenGL mit GLSL (Programmable Pipeline) |
| OpenGL Toolkit | LWJGL |
| Build Management Tool | Maven |
| Entwicklungsumgebung | IntelliJ IDEA |
| Versionsverwaltung Programmcode | GitHub |
| Betriebssystem Entwicklungscomputer | Windows 7 |

Tabelle 1 Zum Einsatz kommende Technologien

## Methoden

Nachfolgend sind die wichtigsten Methoden, welche bei der Umsetzung der Anwendung eingesetzt werden, kurz erläutert:

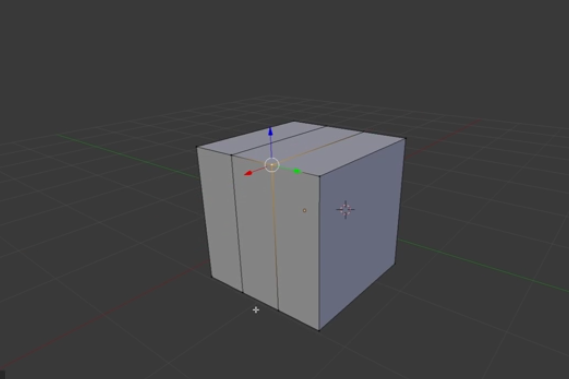
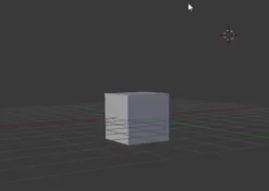
* **Vertex Array Object (VAO) und Vertex Buffer Object (VBO):**Das VBO enthält die eigentlichen Vertexdaten beispielsweise die Positionen, die Normalenvektoren, die Farben oder die Textur-Koordinatenen. Das VAO hingegen enthält die Informationen, in welchem VBO sich die benötigten Daten befinden und in welchem Format sie vorliegen, z.B. dass ein Vertex aus 3 floats für die Position, 3 bytes für den Normalenvektor.  
    
  Der Vorteil vom Einsatz von VAO und VBO ist, dass die Vertexdaten im VRAM der Grafikkarte gespeichert werden, statt wie ohne Einsatz von VAO und VBO im Hauptspeicher der CPU. Das Problem bei einer Speicherung der Vertexdaten im Hauptspeicher ist, dass die Grafikkarte deutlich langsamer auf den Hauptspeicher also auf das eigene VRAM zugreifen kann. Dies wirkt sich negativ auf die Performance der Anwendung aus, weil ein Flaschenhals zwischen CPU und GPU existiert. Dieser Flaschenhals kann mit dem Einsatz mit VAO und VBO umgangen werden.
* **Shader:**GLSL ist eine C-ähnliche Programmiersprache, mit welcher es möglich, selbst Shaders zu schreiben und auf der GPU auszuführen. In der zu realisierenden Anwendung werden 2 verschiedene Shaders benötigt, den Fragment Shader und den Vertex Shader. Der Vertex Shader wird pro Vertex (Knoten) aufgerufen und er dient dazu die Geometrie einer Szene zu manipulieren (View, Projektion, Transformation). Der Fragment Shader bestimmt den Farbwert pro Pixel.
* **Projektion- und View Matrix:**Die Projektionsmatrix lässt weit entfernte Objekte (in Richtung Z-Achse) kleiner darstellen und nahe bei der Kamera liegende Objekte grösser. Für die Bewegung der Kamera benötigen wir eine Viewmatrix.
* **Objekt- und Mausrotation:**Die geladenen Objekte sollen auf einem Boden (Gitternetz) platziert werden. Sobald mit der Maus die Kamera rotiert wird, bewegt sich die ganze Szene, die Kamera hingegen bleibt statisch. Da sich auch der Boden mitbewegt, erhält der Benutzer der Anwendung den Eindruck, als bewege er die Kamera.   
  Nachfolgende Abbildungen zeigt eine mögliche Umsetzung mit einem Boden als Gitternetz. 

Abbildung 1 Mögliche Umsetzung der Kamerarotation mit einem Gitternetz-Boden



# Skizze

Nachfolgende Abbildung zeigt eine Skizze des User Interfaces der zu realisierenden Anwendung.

Im oberen Bereich kann mit Hilfe eines FileOpenDialogs ein .Obj-File ausgewählt und geladen werden. Der Inhalt dieses .Obj-File wird dann von der Anwendung im unteren Bereich interpretiert, d.h. das 3D-Objekt angezeigt.

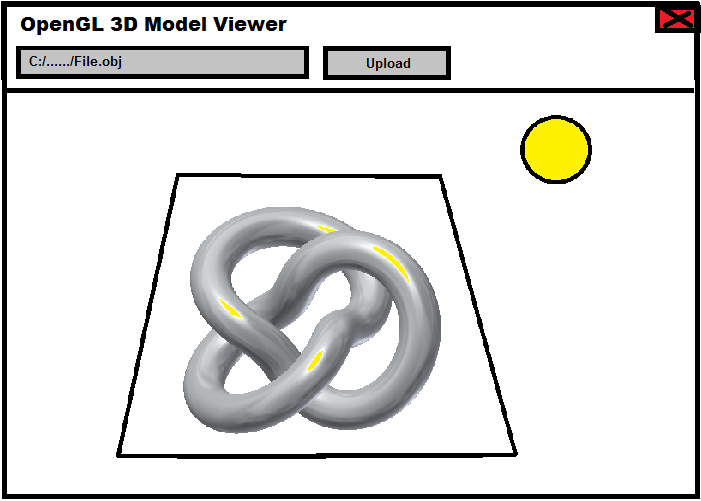


Abbildung 2 Skizze des User Interfaces

# Zeitplan

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tätigkeit** | **Prio.** | **KW 49** | **KW 50** | **KW 51** | **KW 52** | **KW 53** | **KW 01** | **KW 02** | **KW 03** |
| **30.11** | **7.12** | **14.12** | **21.12** | **28.12** | **4.1** | **11.1** | **18.1** |
| Konzept erstellen | Muss |  |  |  |  |  |  |  |  |
| Entwicklungsumgebung einrichten | Muss |  |  |  |  |  |  |  |  |
| Realisierung: DisplayManager und GUI erstellen | Muss |  |  |  |  |  |  |  |  |
| Realisierung: .obj File loader implementieren | Muss |  |  |  |  |  |  |  |  |
| Realisierung: Logik für VAO und VBO implementieren | Muss |  |  |  |  |  |  |  |  |
| Realisierung: GLSL Shaders initialisieren | Muss |  |  |  |  |  |  |  |  |
| Realisierung: Pointlight Ambient, Diffuse, Spekular. (Phong Shading) umsetzen | Kann |  |  |  |  |  |  |  |  |
| Realisierung: Projektion und View-Matrix generieren | Muss |  |  |  |  |  |  |  |  |
| Realisierung: Maus-Rotation implementieren | Muss |  |  |  |  |  |  |  |  |
| Finalisierung der Dokumentation | Muss |  |  |  |  |  |  |  |  |
| Präsentation und Abgabe | Muss |  |  |  |  |  |  |  |  |

Tabelle 2 Zeitplan

# Referenzen

**Wavefront OBJ**  
*Wikipedia, Stand 18.11.2015*[*https://de.wikipedia.org/wiki/Wavefront\_OBJ*](https://de.wikipedia.org/wiki/Wavefront_OBJ) 3

**GLSL**  
*OpenGL Webseite, Stand 21.11.2015*[*https://www.opengl.org/documentation/glsl/*](https://www.opengl.org/documentation/glsl/) 3

**Lightweight Java Game Library 3 (LWJGL)***LWJGL Webseite, Stand 21.11.2015*[*https://www.lwjgl.org*](https://www.lwjgl.org) 3

**OpenGL***OpenGL Webseite, Stand 21.11.2015*[*https://www.opengl.org*](https://www.opengl.org) 3

# Versionskontrolle

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Datum** | **Beschreibung** | **Autor** |
| 0.1 | 09.11.2015 | Dokument erstellt und Dokumentenstruktur festgelegt. | Joel Holzer |
| 0.2 | 23.11.2015 | Dokument überarbeitet: - Erster Entwurf aller Kapitel erstellt. | Michael Koch |
| 0.3 | 23.11.2015 | Dokument überarbeitet:  - Überarbeitung aller Kapitel.  - Kapitel „Methoden“ hinzugefügt. | Joel Holzer |
| 1.0 | 23.11.2015 | Dokument finalisiert | Michael Koch, Joel Holzer |