Übungsblatt 7

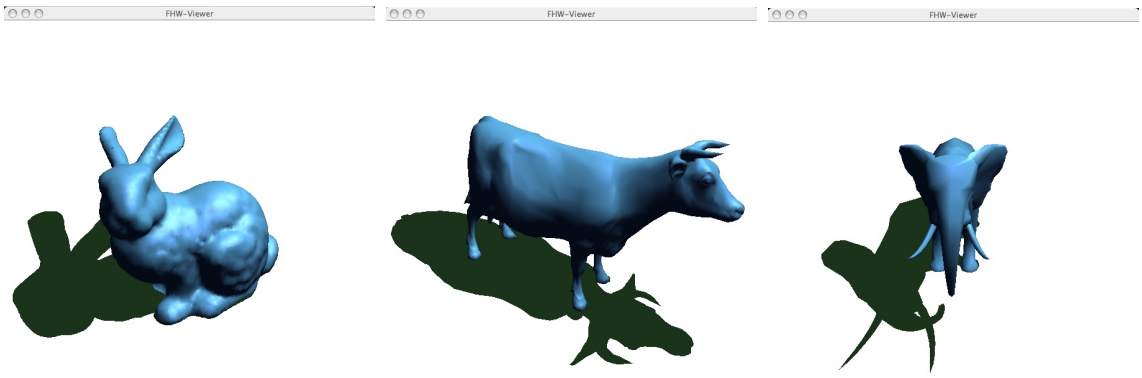
Willkommen zur siebten Übung der Veranstaltung *Generative Computergrafik*. Ziel dieses Übungsblatts ist, dass Sie sich etwas intensiver mit OpenGL vertraut machen.

Dieses Blatt enthält die zweite von drei verpflichtenden Abgaben. Es ist bis Montag, den 12. Juni 2023 um 23:55 Uhr über `https://read.mi.hs-rm.de` abzugeben. Spätere Abgaben oder Abgaben auf anderem Wege, wie z.B. per Email können leider nicht berücksichtigt werden und müssen mit 0 Punkten bewertet werden. Sie sollten daher **frühzeitig mit der Arbeit an diesem Blatt beginnen** und evtl. auch schon früh erste lauffähige Versionen ihrer Lösung hochladen.

> Um das Praktikum zu bestehen müssen Sie *insgesamt (über alle drei Abgaben) mindestens 50% aller erreichbaren Punkte* erreichen, wobei *bei jeder der drei Abgaben mindestens 25% der erreichbaren Punkte* erreicht werden müssen.

Aufgabe 1. Schreiben Sie ein OpenGL-Programm, das ein Dreiecks-Netz, welches im Objekt-Fileformat **obj** als Indexed Faceset gespeichert ist, einlesen und darstellen kann. Ihr Programm soll glfw (siehe `https://www.glfw.org`) als GUI Toolkit verwenden. Starten Sie dabei mit dem Skript `oglTemplate.py`.

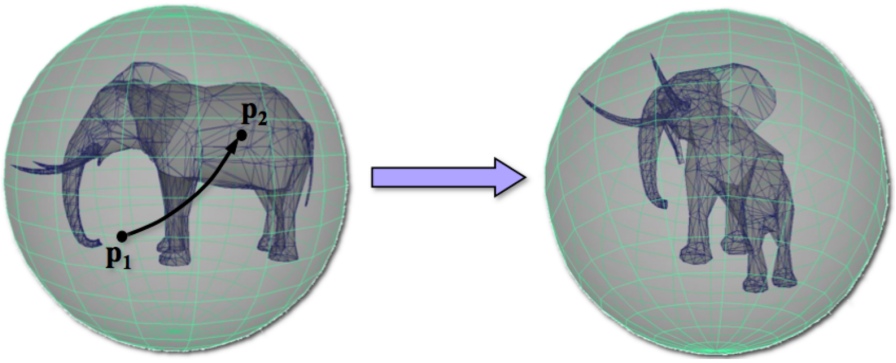
Ihr Programm soll beim Aufruf von `python oglViewer.py object.obj` die Daten aus der Datei `object.obj` einlesen und das zugehörige Modell in einem Fenster darstellen (siehe nachfolgende Abbildungen). Mindestens die fünf Modelle *Stanford-Bunny*, *Cow*, *Elephant*, *Squirrel*, and *Squirrel_ar* (gegeben im **obj**-Fileformat) soll ihr Viewer korrekt darstellen.



Ihr ***obj**-Viewer* soll dabei (mindestens) die folgenden Features besitzen:

- (2 Punkte) *Einlesen eines obj-Files und Darstellung dessen Inhalts als Polygonnetz (**nur Kanten des Dreiecksnetzes werden angezeigt** (Wireframe-Darstellung)), zentriert in der Mitte des Canvas, mit Hilfe von VBA/VBOs.*

- (6 Punkte) *Schattierte Darstellung* eines *beleuchteten Modells* (siehe obige Abbildungen). Die Eigenschaften von Lichtquelle(n) und Objektmaterial(ien) können Sie dabei beliebig wählen. Durch Druck auf die Taste S zwischen **Gouraud-Shading**, **Phong-Shading** und **Wireframe** umgeschaltet werden können.
- *Umschalten* (2 Punkte) zwischen **Orthogonal-** und **Zentral- Projektion** durch Druck auf die Taste P. In beiden Fällen soll die Kamera geeignet angepasst werden, wenn die Fenstergröße geändert wird, damit keine Verzerrungen entstehen.
- (10 Punkte) *Navigation* mit *Maus* und *Tastatur*
  - (5 Punkte) Bei gedrückter *linker Maustaste* soll das 3D Modell mittels *Arcball-Metapher* rotiert werden können. Die *Arcball-Metapher* erlaubt die intuitive 3D Rotation eines Objekts auf Basis von entsprechenden 2D Eingaben mit der Maus. Die wesentliche Erkenntnis dabei ist, dass durch zwei Punkte $\mathbf{p}_1$ und $\mathbf{p}_2$ auf der **Einheitskugel** eine Rotation um die Achse $\mathbf{v} = \mathbf{p}_1 \times \mathbf{p}_2$ und den Winkel $\alpha = \arccos\langle p1, p2\rangle$ beschrieben wird.



Ein Druck auf die *linke Maustaste* definiert dabei den Punkt $\mathbf{p}_1$. Wird die Maus nun bei gedrückter linker Maustaste weiter bewegt, definiert die jeweils aktuelle Mausposition den Punkt $\mathbf{p}_2$. Die Koordinaten $(mx, my) \in [0, WIDTH-1] \times [0, HEIGHT-1]$ des Mauszeigers können dabei mittels

```
def projectOnSphere(x, y, r):
  x, y = x - WIDTH/2.0, HEIGHT/2.0 - y
  a = min(r*r, x**2 + y**2)
  z = sqrt(r*r - a)
  l = sqrt(x**2 + y**2 + z**2)
  return x/l, y/l, z/l
```

auf eine entsprechende Kugel über dem Canvas projiziert werden.

- (2 Punkte) Bei gedrückter *mittlerer Maustaste* soll an das Modell **heran- bzw. weg-gezoomt** werden können.
- (2 Punkte) Bei gedrückter *rechter Maustaste* soll das Modell parallel zur Bildebene **verschoben** werden können.
- (1 Punkt) **Rotation um die x, y, z Achse** (um einen von Ihnen zu wählenden Winkelschritt) bei Druck auf die X, Y, Z Taste.

Zusatzaufgabe (hiermit können Sie *weitere 5 Punkte* sammeln, die auf alle Abgaben angerechnet werden.):
Das dargestellte Modell soll zusätzlich einen *Schatten* auf eine Ebene auf der es steht werfen (siehe Beispiele auf Seite 1). Integrieren Sie hierzu das in `https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping` beschriebene *Shadow Mapping* in ihren Viewer. Dabei wird die Szene aus der Sicht der Lichtquelle gerendert. Alles, was man aus der Perspektive des Lichts sieht, ist beleuchtet und alles, was man nicht sehen kann, muss im Schatten liegen (siehe dazu auch Abbildung 1)

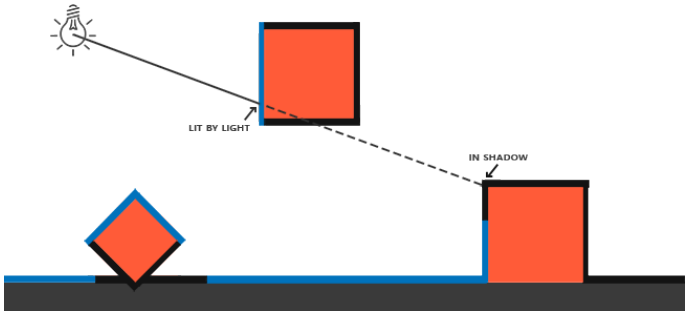


Abbildung 1: Prinzip des Shadow Mappings: Alles, was man aus Sicht einer Lichtquelle sieht, ist beleuchtet. [Bildquelle: `https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping`]

Beim Shadow Mapping rendert man nun die Szene zweimal. Zuerst aus der Perspektive des Lichts und speichert dabei die resultierenden Tiefenwerte (ähnlich wie bei der "Tiefenprüfung") in einer Textur, der sogenannten *Depth Map (Tiefenkarte) oder auch Shadow Map (Schattenkarte)*. Die Tiefenwerte das erste Fragment, das aus der Perspektive des Lichts sichtbar ist. Im zweiten Durchgang rendert man die Szene normal und verwenden die erzeugte Tiefenkarte, um zu berechnen, ob sich Fragmente im Schatten befinden.