Computergrafik 1

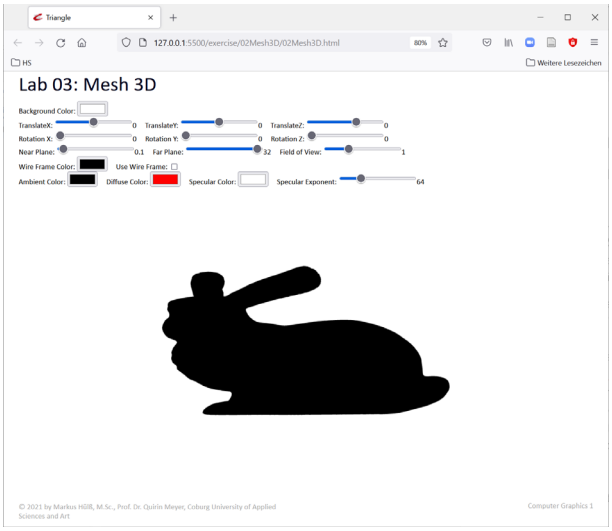
Lab 3

Aufgabe 1 Anzeigen eines 3D Modells

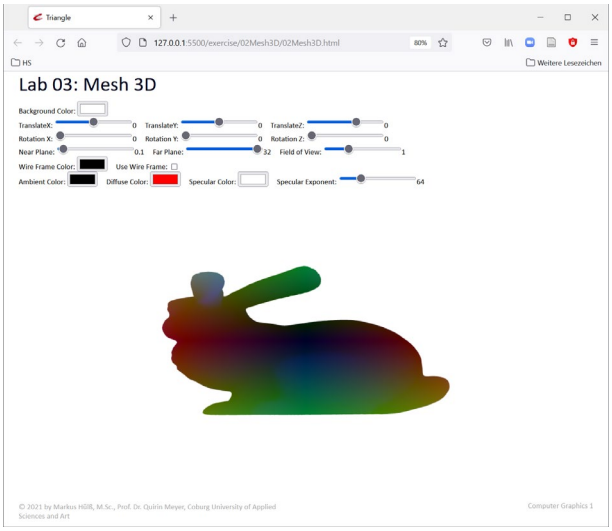
(a) Verwenden Sie die Klasse `TriangleMeshGL` aus dem vorherigen Praktikum. Wenn Sie alles richtig implementiert haben, so sollte diese bereits auch 3D Modelle behandeln können. Allerdings müssen Sie den Rest des Programmes noch etwas erweitern.

(b) Das Mesh `data/bunny.smm` enthält keine Vertex-Farben. Stellen Sie sicher, dass wenn ein Mesh, welches an den Konstruktor von `TriangleMeshGL` übergeben wird, keine Vertex-Farben enthält – das ist dann der Fall, falls `colors == null` gilt – dass auch dann kein WebGL Array-Buffer mit Farben am WebGL Vertex Array Objekt gebunden wird!

Sie sollten dieses Ergebnis erhalten:



(c) Damit etwas Farbe ins Spiel kommt, können Sie die ausgehende Vertex-Farbe im Shader auf den Absolutbetrag (Hier hilft die GLSL Funktion: `abs(vec3 x)`) der eingehenden Position setzen. Dann sollten Sie folgendes Bild sehen:

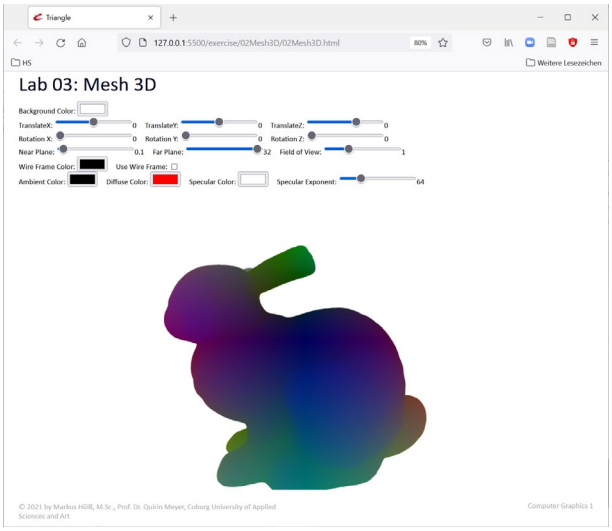


(d) Implementieren Sie in der Datei `Matrix4.js` die Funktionen `translation`, `rotationX`, `rotationY`, `rotationZ`, `multiply` und `perspective` gemäß der Dokumentation in den Kommentaren!

(e) Berechnen Sie in der Funktion `draw` (die Sie innerhalb der Funktion `Mesh3DApp` in der Datei `02Mesh3D.js` finden) eine Model-View-Projection Transformations Matrix `mvp`. Diese soll einen homogenen Punkt (d.h. $[wx, wy, wz, w], w \neq 0$) erst um die z-Achse, dann um die y-Achse und dann um die z-Achse rotieren. Anschließend soll die Matrix diesen homogenen Punkt in x, y und z Richtung verschieben. Abschließend soll die Matrix in den Clip-Space perspektivisch projizieren.
Rufen Sie dazu die geeigneten Methoden aus `Matrix4.js` auf. Die Parameter der Funktion werden bereits aus dem User-Interface in den lokalen Variablen `rotationX`, `rotationY`, `rotationZ`, `translateX`, `translateY`, `translateZ`, `nearPlaneDistance`, `farPlaneDistance` und `fieldOfViewRadians` für Sie bereitgestellt.

(f) Übergeben Sie die 4x4-Matrix `mvp` dem Vertex-Shader. Definieren Sie dazu im Vertex-Shader eine geeignet uniforme Variable `u_mvp`.

(g) Transformieren Sie im Vertex-Shader die eingehende Position mittels der Model-View-Projection Matrix in den Clip-Space. Der Vertex-Shader nimmt Clip-Space Koordinaten in der vordefinierten Variable `gl_Position` entgegen und leitet sie an die nächste Pipeline stufe entsprechend weiter.



(h) Stellen Sie sicher, dass Wire-Frame funktioniert!