Computergrafik 1

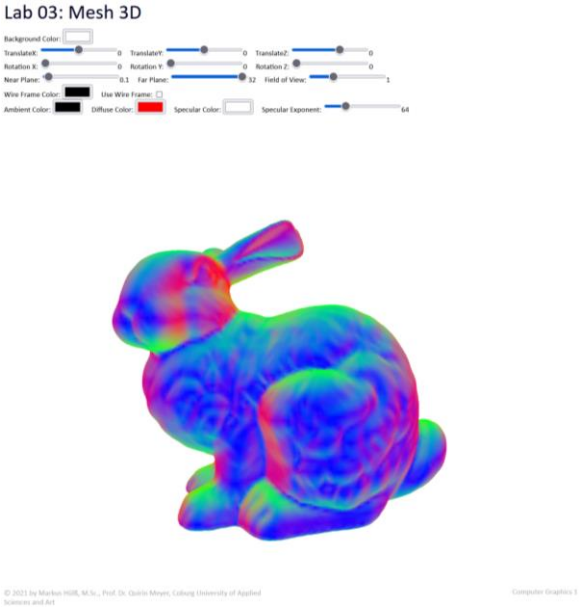
Lab 5

Aufgabe 1 Normalen weiterleiten

(a) Die Klasse `TriangleMeshGL` soll nun für die per-Vertex Normalenvektoren einen Array Buffer anlegen und die Normalen aus dem `simpleMeshIO` dort hineinkopieren, falls diese vorhanden sind. Legen Sie dazu einen WebGL Array Buffer an und stellen Sie sicher, dass dieser auch ordentlich an das passende Vertex Array Object gebunden ist. Vergessen Sie in diesem Zusammenhang nicht, diesen mittels `gl.vertexAttribtPoint` und `gl.enableVertexAttribArray` zu konfigurieren! Die Attribute Location finden Sie in der Konstanten `normalAttributeLocation`!

(b) Lesen Sie die Normalenvektoren als Attribute im Vertex Shader ein. Es empfiehlt sich dieses Attribute `a_normal` zu nennen.

(c) Leiten Sie den Absolutbetrag der per-Vertex Normale als Farbe (`fs_color`) an den Fragment-Shader weiter! Sie sollten folgendes Bild erhalten:

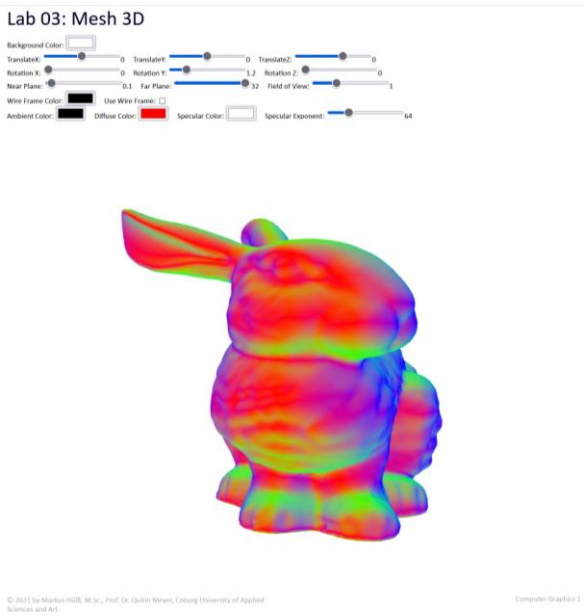


(d) Geben Sie nun die Normale vom Vertex Shader direkt an den Fragment Shader und berechnen Sie die Fragment Farbe im Fragment Shader aus dieser Normalen! Es empfiehlt sich diese Variable `fs_normal` zu nennen.

(e) Normalen sollten immer Einheitslänge haben, d.h. $\|\vec{n}\| = 1$. Allerdings kann der Rasterizer der GPU diese Eigenschaft zerstören und im Fragment-Shader kommen nicht normalisierte Normalen an. Finden Sie eine passende GLSL Funktion um die Normalen im Fragment Shader zu normalisieren!

Aufgabe 2 Normalen transformieren

Der Vertex Shader transformiert bisher nur die Position, nicht aber die Normalen. Drehen Sie den Hasen also, sieht das Bild z.B. so aus:



Aus der Vorlesung wissen wir, dass wenn eine Vertex Position $\vec{p}$ mit der Matrix $\boldsymbol{M}$ transformiert wird, d.h.

$$\vec{p}' = \boldsymbol{M} \cdot \vec{p}$$

dann wird eine Normale $\vec{n}$ mit der invers-transponierten Matrix $(\boldsymbol{A}^{-1})^T$ transformiert:

$$\vec{n}' = (\boldsymbol{M}^{-1})^T \vec{n}.$$

folgt

Die Matrix $\boldsymbol{M}$ sollte bei Ihnen wie folge berechnet werden:

$$\boldsymbol{M} = \boldsymbol{T}(\vec{t}) \cdot \boldsymbol{R}_x(\alpha_x) \cdot \boldsymbol{R}_y(\alpha_y) \cdot \boldsymbol{R}_z(\alpha_z)$$

und transformiert die Positionen aus dem Object-Space in den Camera-Space. Beachten Sie, dass hier die Projektionsmatrix bewusst fehlt, den mit dieser würden wir ja in den Clip-Space

denn

transformieren. Dabei ist

- $\boldsymbol{T}(\vec{t}) \in \mathbb{R}^{4\times4}$ eine Translationsmatrix, die einen Punkt um den Translationsvektor $\vec{t}$ verschiebt.
- $\boldsymbol{R}_x(\alpha_x) \in \mathbb{R}^{4\times4}$ eine Rotationsmatrix, die einen Punkt um die x-Achse mit dem Winkel α_x rotiert.
- $\boldsymbol{R}_y(\alpha_y) \in \mathbb{R}^{4\times4}$ eine Rotationsmatrix, die einen Punkt um die y-Achse mit dem Winkel α_y rotiert.
- $\boldsymbol{R}_z(\alpha_z) \in \mathbb{R}^{4\times4}$ eine Rotationsmatrix, die einen Punkt um die z-Achse mit dem Winkel α_z rotiert

(a) Bestimmen Sie allgemein auf Papier die Inversen von $\boldsymbol{T}(\vec{t}), \boldsymbol{R}_x(\alpha_x), \boldsymbol{R}_y(\alpha_y), \boldsymbol{R}_z(\alpha_z)$!

(b) Bestimmen Sie allgemein auf Papier die transponierten Matrizen von $\boldsymbol{T}(\vec{t})$, $\boldsymbol{R}_x(\alpha_x)$, $\boldsymbol{R}_y(\alpha_y)$ und $\boldsymbol{R}_z(\alpha_z)$. Fällt Ihnen was auf?

Aus der Vorlesung *Diskrete Mathematik* sollten Sie über Matrizen $\boldsymbol{A}_i$ folgendes wissen:

$$(\boldsymbol{A}_1 \cdot \boldsymbol{A}_2 \cdot \ldots \cdot \boldsymbol{A}_n)^T = \boldsymbol{A}_n^T \cdot \ldots \boldsymbol{A}_1^T \cdot \boldsymbol{A}_0^T,$$
$$(\boldsymbol{A}_1 \cdot \boldsymbol{A}_2 \cdot \ldots \cdot \boldsymbol{A}_n)^{-1} = \boldsymbol{A}_n^{-1} \cdot \ldots \boldsymbol{A}_1^{-1} \cdot \boldsymbol{A}_0^{-1}.$$

(c) Nutzen Sie diese Erkenntnis um $(\boldsymbol{M}^{-1})^T$ zu bestimmen!

(d) Implementieren Sie die Funktion `transpose` in `Matrix4.js` gemäß der Dokumentation im Kommentar!

(e) Berechnen Sie in der Funktion `draw` (die Sie innerhalb der Funktion `Mesh3DApp` in der Datei `Mesh3D.js` finden) die invers-transponierte Model View Matrix $(\boldsymbol{M}^{-1})^T$ und legen Sie diese in der lokalen Variable `mvInvT` ab.

(f) Übergeben Sie die 4x4-Matrix `mvInvT` dem Vertex-Shader. Deklarieren Sie dazu im Vertex-Shader eine geeignet uniforme Variable `u_mvInvT`.

(g) Transformieren Sie im Vertex-Shader die eingehenden Normalen mittels der inverse-transponierten Model-View- Matrix in den Camera-Space. Allerdings sind die eingehenden Normalen 3D und nicht 4D. Sie müssen also eine geeignete *w* Komponenten hinzufügen! Auch das Ergebnis der Matrix Multiplikation ist ein 4D Vektor, aber die Normale, die Sie an den Fragment-Shader weiterleiten sollte 3D sein. Überlegen Sie sich eine geeignete Konvertierung!

Dann sollten Sie folgendes Bild erhalten: