3D-Computergrafik I

Sommersemester 2015 – Aufgabenblatt 5

Hasso-Plattner-Institut
Fachgebiet Computergrafische Systeme

Aufgabe 15: Projektionen (2 Punkte)

Projektionen bezeichnen die Abbildungsvorschrift einer 3D-Szene auf eine 2D-Ebene (Canvas). In der 3D-Computergrafik werden vor allem zwei Arten von Projektion häufig benutzt: Perspektische und orthogonale Projektionen. Implementieren Sie sechs orthogonale Projektionen. Diese sollen Ansichten in Richtung der folgenden Achsen erzeugen: +x , -x , +y, -y, +z, -z.

Definieren Sie hierzu die orthogonale Projektionsmatrix in der Methode `resize(int width, int height)` in `exercise16.cpp`: Verwenden Sie dabei die Membervariable `m_orthoProjection` (vom Typ `QMatrix4x4`) und deren Funktion `ortho()`. Beachten Sie hierbei das Seitenverhältnis (`aspectRatio`).

Definieren Sie zusätzlich unter Verwendung der Membervariable `m_views` in der Methode `setupViews()` die jeweiligen LookAt-Parameter für die orthogonalen Projektionen.

Aufgabe 16: Explosionseffekt durch Transformationen (7 Punkte)

In dieser Aufgabe sollen Sie ein 3D-Objekt "explodieren" lassen (siehe Abbildung 1), welches bereits vom Programmrahmen aus einer `.obj` ausgelesen wird. Das 3D-Objekt ist durch ein Dreiecksnetz definiert. Um einen Körper "explodieren" zu lassen, sollen Sie die einzelnen Dreiecke eines Körpers von seinem Ursprung weg translieren und um sich selbst rotieren lassen. Dies soll nicht zeitgesteuert, sondern durch Nutzerinteraktion passieren. Dazu sollen Sie mittels eines Sliders den Explosionsfortschritt selbst steuern.

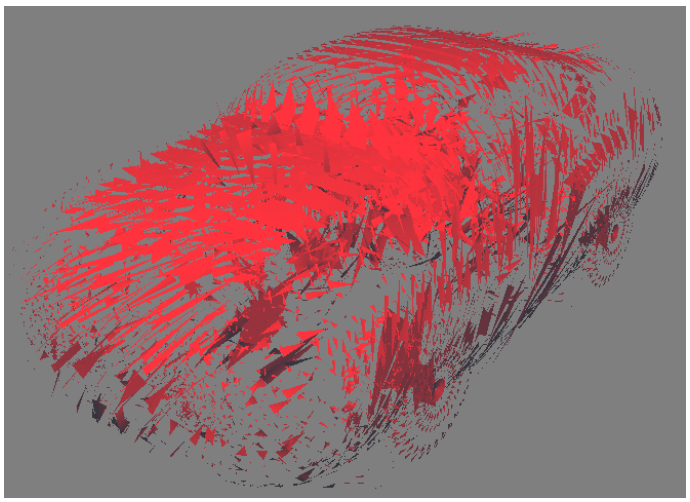

Abbildung 1: Ein "explodierendes" 3D-Objekt.

Passen Sie hierzu den Funktionsrumpf von `main()` im Shader `model.geom` an. Der Explosionsfortschritt soll aus der Variable `animationFrame` ($\in [0; 1]$) ausgelesen werden. Die genauen Translations- und Rotationsrichtungen stehen Ihnen dabei frei. Die oben genannten Kriterien sollen jedoch erfüllt werden. Beachten Sie, dass Matrizen in GLSL **column-major** definiert werden.

Aufgabe 17: Globale Deformationen (8 Punkte)

In der Vorlesung haben Sie das Konzept der globalen Deformationen kennengelernt. Hierzu wird ein solider Körper zunächst in einer benötigten Feinheit tesseliert, um dann eine Transformationsbeschreibung (mit beliebiger Komplexität) pro Vertex anzuwenden. Basierend auf dem Paper von *Blanc* arbeiten diese sowohl auf einem kartesischen als auch einem zylindrischen Koordinatensystem (siehe Skript F. 85). Ihre Aufgabe ist es die folgenden globalen Deformationen zu implementieren (siehe Abbildung 2):

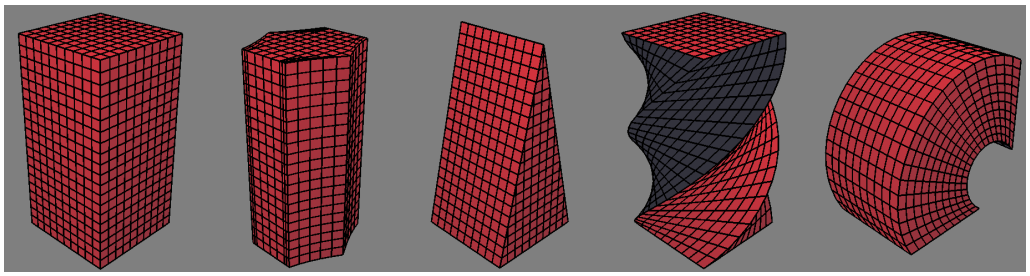

Abbildung 2: Globale Deformationen (von links nach rechts): Original, Mold, Pinch, Twist, Bend.

Mold: Der Abstand der Vertices von der xy-Ebene wird entsprechend des Winkels zwischen dem jeweiligen Vertex und der z-Achse skaliert. Dabei soll mit steigendem Winkel auch der Skalierungsfaktor steigen.

Pinch: Der Körper selbst wird mit steigender Höhe (y-Achse) in einer dazu orthogalen Achse (x-Achse) (gegen 0 gehend) skaliert. Hierzu werden die bei der Tesselierung erhaltenen Vertices mit höheren y-Werten stärker entlang der x-Achse transliert als die mit geringeren.

Twist: Der Körper selbst wird beliebigfach in sich verdreht. Dazu sollen die Vertices je nach Höhe im Körper um die y-Achse rotiert werden.

Bend: Der Körper wird um einen fixen Punkt gebogen.

Implementieren Sie hierzu die Methoden:

- `vec3 mold(vec3 v, float moldPlateau)`
- `vec3 pinch(vec3 v, float pinchPlateau)`
- `vec3 twist(vec3 v, float maxAngle)`
- `vec3 bend(vec3 v, float maxAngle)`

im Shader `box.vert`, wobei die Parameter `moldPlateau`, `pinchPlateau` und `maxAngle` über die Stärke der Deformation entscheiden sollen.

Aufgabe 18: Haloed Lines (4 Punkte)

Implementieren Sie die in der Vorlesung vorgestellten Haloed Lines zur Wireframe-Darstellung (siehe Abbildung 3). Vervollständigen Sie hierzu in der `exercise18.cpp` die Methode `initialize()` um einen geeigneten Aufruf von `glDepthFunc()` und die Methode `render()` um geeignete Aufrufe von `glLineWidth`, `glColorMask` und `m_drawable->draw()`. Außerdem soll die Liniendicke in Abhängigkeit von der Membervariable `m_animationFrame` definiert werden.

Ferner passen Sie den Shader `haloedlines.geom` so an, dass die Geometrie als Wireframe dargestellt wird. Außerdem soll im Shader `haloedlines.frag` die Farbe in Abhängigkeit von der Tiefe angepasst werden.

Allgemeine Hinweise:

- Die Aufgaben sollen maximal zu zweit bearbeitet werden; Ausnahmen müssen mit den Übungsleitern abgesprochen werden.
- Bitte reichen Sie Ihre Lösungen bis Dienstag, den **23.06.2015**, um **13.15** Uhr ein.
- Tragen Sie Ihre Matrikelnummern in die Quellcode-Dateien ein. Beachten Sie, dass nur die vollständigen Quelltexte und Projektdateien geschickt werden sollen. Senden Sie uns keinesfalls ausführbare Dateien oder bereits kompilierte Binär- oder Temporärdateien (*.obj, *.pdb, *.ilk, *.ncb etc.) zu! Testen Sie vor dem Verschicken, ob die Projekte aus den Zip-Dateien fehlerfrei kompiliert und ausgeführt werden können.

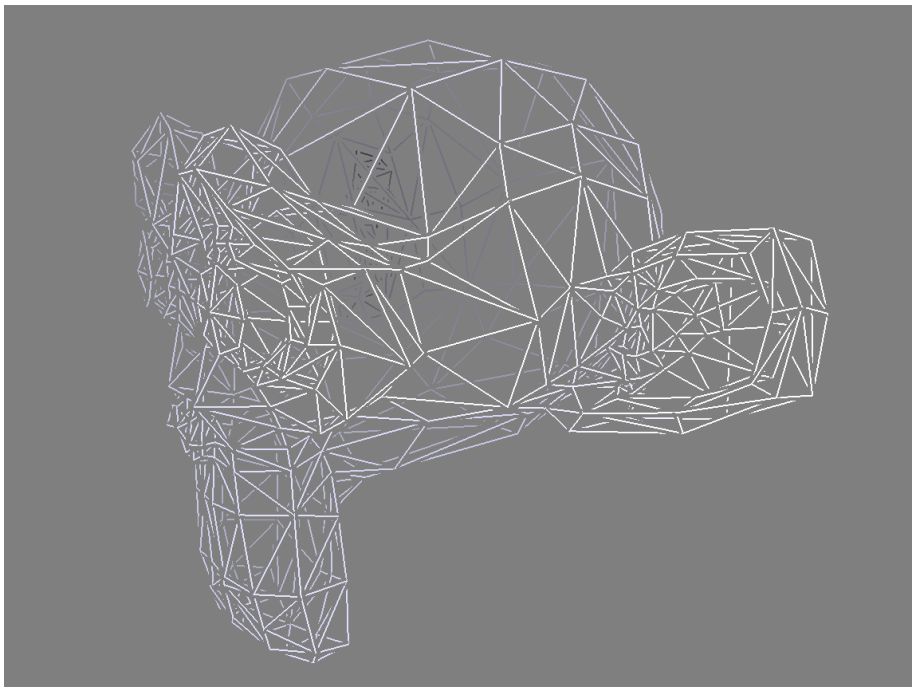

Abbildung 3: Beispiel: Die Intensität der Haloed Lines nimmt mit zunehmender Tiefer zur Kamera ab.

- Zippen Sie Ihre Lösungen in **eine** Zip-Datei. Geben Sie der Zip-Datei einen Namen nach folgendem Schema:

  cg1_blatt5_matrikelnummer1_matrikelnummer2.zip.

- Die Zip-Datei laden Sie dann bei moodle hoch.