CG1 WS14/15 - Übungsblatt 3

Technische Universität Berlin - Computer Graphics
Ausgabe 28. November 2014, **Abgabe** 19. Dezember 2013
Prof. Dr. Marc Alexa, Katrin Lang, Xi Wang.

AUFGABE 1 : Programmierbare Shader

In dieser Übung sollen Sie die Verarbeitung von Dreiecksnetze sowie programmierbare Shader kennenlernen. Auf der ISIS Seite des Kurses steht Ihnen ein `C++` Programmskelett zur Verfügung, welches Sie zur Lösung verwenden sollten.

1. Modelle in Form von Dreiecksnetzen sollen geladen und angezeigt werden können. Als Modellformat soll `OFF` verwendet werden. Dieses ist wie folgt aufgebaut:

Zeile	Inhalt	Beschreibung
1	OFF	Datei-Kennung
2	$V\ F\ E$	V: Anzahl der Dreiecksknoten, F: Anzahl der Polygone, E: Anzahl der Kanten
3	$x\ y\ z$	Koordinate des Knotens mit Index 0
4	$x\ y\ z$	Koordinate des Knotens mit Index 1
...	$x\ y\ z$	Koordinate des Knotens mit Index ...
$V+2$	$x\ y\ z$	Koordinate des Knotens mit Index $V-1$
$V+3$	$n\ i1\ \ldots\ in$	Polygon mit n Knoten in den Positionen der durch i_k referenzierten Einträge
$V+4$	$n\ i1\ \ldots\ in$	Polygon mit n Knoten in den Positionen der durch i_k referenzierten Einträge
...	$n\ i1\ \ldots\ in$	Polygon mit n Knoten in den Positionen der durch i_k referenzierten Einträge
$V+F+2$	$n\ i1\ \ldots\ in$	Polygon mit n Knoten in den Positionen der durch i_k referenzierten Einträge

   Implementieren Sie in der Klasse `TriMesh`, welche ein Dreiecksnetz repräsentiert, eine Funktion `loadOff(...)`, welche Dateien im oben spezifizierten Format einlesen kann. (1 Punkt)

2. Implementieren Sie eine Funktion `TriMesh::draw()`, welche das Dreicksnetz mittels `Vertex Arrays` aus generischen Vertex-Attributen darstellt. Nutzen sie hierzu die in `TriMesh` vordefinierte Konstante `AttribVertex`. Sie können die Korrektheit der Funktionen `loadOff(...)` und `draw()` mittels der in Klasse `ShadingDemo` mitgelieferten Shader `colorShader` und `colorTriangleShader` überprüfen. **Hinweis:** Achten Sie auf das korrekte Polygon Winding. (1 Punkt)

3. Berechnen Sie in einem Geometry-Shader Flächennormalen (face normals), und visualisieren Sie diese analog zum in Klasse `ShadingDemo` mitgelieferten Shader `normalVizShader`. Sie können den Shader `colorTriangleShader` als Vorlage verwenden. **Hinweis:** Im Vertex-Shader geschriebene Variablen sind im Geometry-Shader als Array der Größe 3 abzufragen. (1 Punkt)

4. Berechnen Sie anschliessend in der Klasse `TriMesh` die Normalen pro Vertex als gewichtete Summme der Normalen der Nachbar-Facetten. Sie können die Korrektheit der Funktion mittels des in Klasse `ShadingDemo` mitgelieferten Shaders `normalVizShader` überprüfen. Zum Zeichnen sollen wiederum Vertex-Arrays verwendet werden. Erweitern Sie dazu die Funktion `TriMesh::draw()`. (1 Punkt)

5. Implementieren Sie in der Funktion `ShadingDemo::visualizeLightDir` einen Shader, der im Fragment-Shader die Richtung der Punktlichtquelle `Context::lightSource`, von jedem Pixel aus gesehen, visualisiert. Die Position der Lichtquelle ist in *Welt-Koordinaten* definiert. (1 Punkt)

6. Implementieren Sie eine GLSL-Funktion `blinnPhongReflection`, welche die Beleuchtung mittels des Blinn–Phong-Beleuchtungsmodells für ein Punktlicht bestimmt. Den Lichtabfall (attenuation) vernachlässigen wir. Geeigneterweise soll die Funktion in eine separate Datei `blinnPhongReflection` ausgelagert werden, damit sie in den nächsten Unteraufgaben wiederverwendet werden kann (1 Punkt)

7. Implementieren Sie drei Shader, welche mit Hilfe der soeben implementierten Funktion `blinnPhongReflection`

(a) Flat Shading

(b) Gouraud Shading

(c) Phong Shading

berechnen. Benutzen sie bei die vordefinierten Strukturen `Context::lightSource` und `Context::material`.

Achtung:

- Achten Sie darauf, dass ihr Algorithmus auch anisotrope Skalierung des Modells korrekt behandelt!
- Die „Perspective"- und „Modelview"-Matrix sowie die Lichtquelle dürfen auch dann nicht über builtins abgefragt werden, wenn Sie `#version 120` verwenden!
- Im c++-Code dürfen keine Funktionen verwendet werden, die in OpenGL3+ als `deprecated` ausgewiesenen sind!

Zusatzpunkte (freiwillig):

8. Implementieren Sie in der Klasse `TriMesh` Vertex Buffer Objects (VBO). Bestimmen Sie die Laufzeit sowohl für VBO und Vertex Arrays. Wie unterscheiden sich diese? Was ist das jeweilige Anwendungsszenario?

AUFGABE 2 : Theoriefragen

1. Phänomenologie von Beleuchtungsmodellen.

   (a) Ein Dreieck wird mittels Gouraud-Shading gezeichnet. Welches Bild ist zu erwarten, wenn eine Normale in Richtung Lichtquelle zeigt und die anderen Normalen davon abweichen? (0.5 Punkte)

   (b) Auf welche der Beleuchtungskomponenten ambient, diffus, spekulär hat die Position des Betrachters bei einer ansonsten statischen Szene Einfluss? Begründen Sie Ihre Antwort. (0.5 Punkte)

2. In 1978 führte Jim Blinn „bump mapping" in der Computergrafik ein. Nennen Sie Vor- und Nachteile des Verfahrens. Warum wird es insbesondere im Bereich von Computerspielen häufig verwendet? (1 Punkte)

3. Erläutern Sie die Unterschiede zwischen `uniform`, `attribute` und `varying` Variablen in GLSL. Wie werden diese mit Daten befüllt? Wie verhält es sich in OpenGL3+? (1 Punkt)

4. Die aktuelle Version der OpenGL Pipeline enthält, neben dem Fragment- und dem Vertex-Shader, zwei weitere programmierbare Shader: Der „Geometry Shader" und der „Tesselation Shader". Erklären Sie die Funktionen diesen beiden Shader und nennen Sie jeweils eine Anwendung (ausser Flächennormalen zu berechnen). (1 Punkt)

5. Aus welchem Grund werden beim Aufbau eines BSP Baumes aus einem Dreicksnetz oftmals die enthaltenen Dreiecke selbst zum Spannen der Partitionierungsebene verwendet? Nennen sie mindestens eine weitere Möglichkeit, Partitionierungsebenen zu bestimmen. (1 Punkt)