3D-Computergrafik I

Sommersemester 2014 – Aufgabenblatt 6

Hasso-Plattner-Institut
Fachgebiet Computergrafische Systeme

Aufgabe 20: Toon-Shader und Phong-Beleuchtungsmodell (8 Punkte)

Mit Hilfe von Shaderprogrammen lassen sich Renderingeffekte (z.B. Anwendung von Beleuchtungsmodellen) effizient auf der Grafikhardware berechnen. Gegenüber einer rein CPU-basierten Berechnung solcher Effekte ergeben sich hierdurch deutliche Performancesteigerungen. *Vertex Shader* ermöglichen dabei die Manipulation von Vertexinformationen (z.B. Position, Normale), während durch *Fragment-* oder *Pixel Shader* einzelne Fragmente angepasst werden können.

In dieser Aufgabe sollen Sie unter Nutzung von Shadern die folgenden zwei Renderingeffekte realisieren: (a) einen Cartoon-Effekt und (b) die Beleuchtung und Schattierung nach Phong (siehe Abbildung 1).

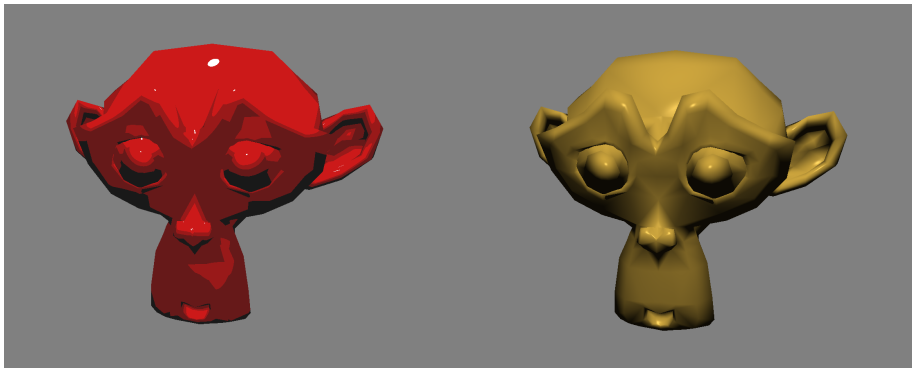
Cartoon-Effekt Um einen Cartoon-Effekt ähnlich dem in der Vorlesung besprochenen *Cel-Shading* zu erreichen ist es wichtig einen diskreten Farbraum mit geringer Anzahl an Farben festzulegen. Die jeweils genutzte Farbe ist dabei abhängig von der *Intensität* I, berechnet anhand der Normalen der Oberfläche und der Richtung der Lichtquelle.

Vervollständigen Sie den Fragment Shader `toon.frag` so, dass eine Diskretisierung der Grundfarbe auf drei Intensitätsbereiche (zwischen $I >= 0,01$ und $I <= 0,99$) erfolgt. Integrieren Sie zusätzlich eine Farbgebung für die Intensitätsbereiche für dunkle schattierte Bereiche ($I < 0,01$), sowie einen Bereich für Highlights ($I > 0,99$) mit einer helleren Farbe.

Phong-Beleuchtungsmodell Das in der Vorlesung vorgestellte Beleuchtungsmodell nach Phong bietet eine effiziente Möglichkeit, lokale Beleuchtung echtzeitfähig zu simulieren. Hierzu werden sowohl die Lichtparameter als auch die Materialeigenschaften anhand einer Gewichtungsfunktion (1) in die Berechnung der Farbwerte miteinbezogen. Die Parameter für das Licht werden dabei pro Lichtquelle durch einen *ambienten* (I_a), einen *diffusen* (I_d), sowie einen *spekularen* Anteil (I_s) simuliert. Für das Material werden analog die *ambienten* (k_a), *spekularen* (k_s) und *diffusen* (k_d) Parameter vergeben um verschiedene Materialeigenschaften simulieren zu können. Zusätzlich wird beim Material ein Koeffizient *shininess* (k_{shi}) für die Stärke der Glanzlichter festgelegt, durch den Oberflächen matter oder spiegelnder gestaltet werden können (Wertebereich 0 bis 128).

Vervollständigen Sie den Fragment Shader `phong.frag` unter Verwendung der gegebenen *uniform*-Variablen so, dass eine korrekte Beleuchtungsberechnung nach Phong erfolgt. Vernachlässigen Sie hierbei die Abschwächung des Lichts (*attenuation*) sowie den Emissionskoeffizienten (k_em).

$$I = I_a k_a + I_d k_d \langle N, L \rangle + I_s k_s \langle R, V \rangle^{k_{shi}} \tag{1}$$



(a) Toon-Shader (b) Phong-Beleuchtungsmodell

Abbildung 1: Unterschiedliche Renderingeffekte am Beispiel eines simplen 3D-Modells.

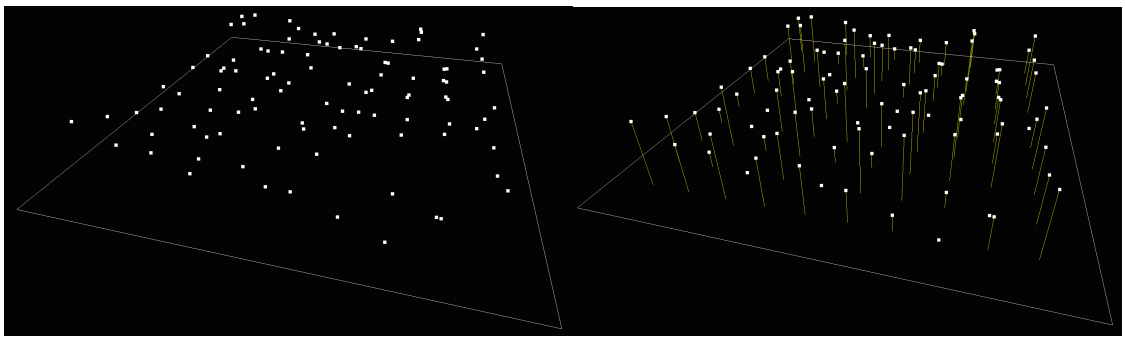
Aufgabe 21: Höhenfeld-Triangulierung und Bézierflächen (7 Punkte)

Eine häufig verwendete Repräsentationsform von Geländemodellen sind sogenannte Höhenfelder. Hierbei handelt es sich um zweidimensionale Skalarfelder, die ein Höhenrelief beschreiben. In dieser Aufgabe soll ein solches Höhenfeld auf verschiedene Arten visualisiert werden.

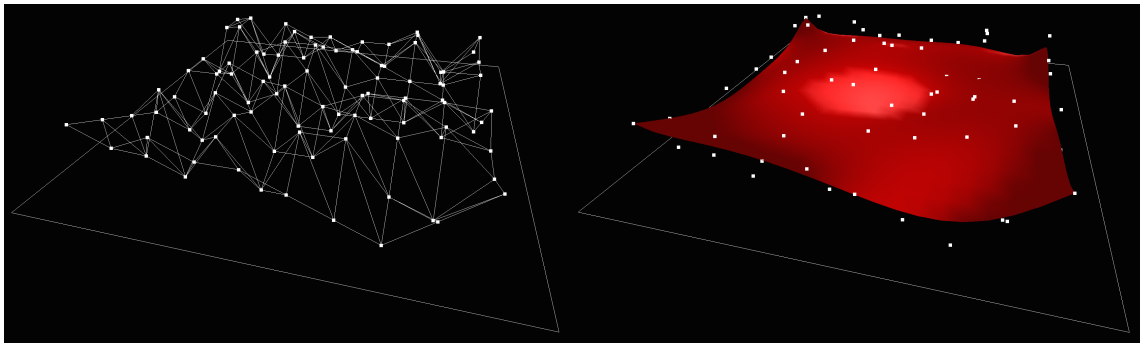
Das Höhenfeld ist als regelmäßiges Raster (genauer: ein multidimensionales Array) `m_heightField` gegeben, das `m_gridSize` verschiedene Höhenwerte enthält. Zur Visualisierung des Höhenfelds implementieren sie in der Klasse `Exercise21` die folgenden Methoden:

- `drawHeightFieldPoints()`. In dieser Methode sollen die einzelnen Höhenwerte in Form von *GL_POINTS* gerendert werden. Die Punktgröße soll auf einen von Ihnen gewählten sinnvollen Wert gesetzt werden (siehe Abbildung 2a).
- `drawHeightFieldLines()`. Visualisieren Sie das Höhenfeld mit Hilfe von *GL_LINES*, die senkrecht auf der xz-Ebene stehen und deren Länge dem jeweiligen Höhenwert entspricht (siehe Abbildung 2b).
- `drawTriangulatedHeightField()`. In dieser Methode soll das Höhenfeld trianguliert werden. Verwenden Sie *GL_TRIANGLE_STRIP*s als Renderingprimitive (siehe Abbildung 2c).

Abschließend sollen die Höhenwerte als Kontrollpunkte einer Bézierfläche interpretiert werden (siehe Abbildung 2d). Nutzen Sie zum Rendering der Bézierfläche einen zweidimensionalen OpenGL-Evaluator[1]. Initialisieren Sie diesen in der Methode `initializeGL()` unter Verwendung der Funktionen `glMap2f()` und `glMapGrid2f()`. Implementieren Sie zudem die Methode `drawHeightFieldBezierPatch()` um die Bézierfläche durch Verwendung der Funktion `glEvalMesh2()` zu rendern.



(a) Höhenwerte als Punkte (b) Höhenwerte als Linien



(c) Trianguliertes Höhenfeld (d) Bézierfläche

Abbildung 2: Verschiedene Visualisierungen eines Höhenfelds.

[1] http://www.glprogramming.com/red/chapter12.html

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 **14.07.2014**, 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.
- Zippen Sie Ihre Lösungen in **eine** Zip-Datei. Geben Sie der Zip-Datei einen Namen nach folgendem Schema:

  cg1_blatt6_matrikelnummer1_matrikelnummer2.zip.
- Die Zip-Datei laden Sie dann bei moodle hoch.