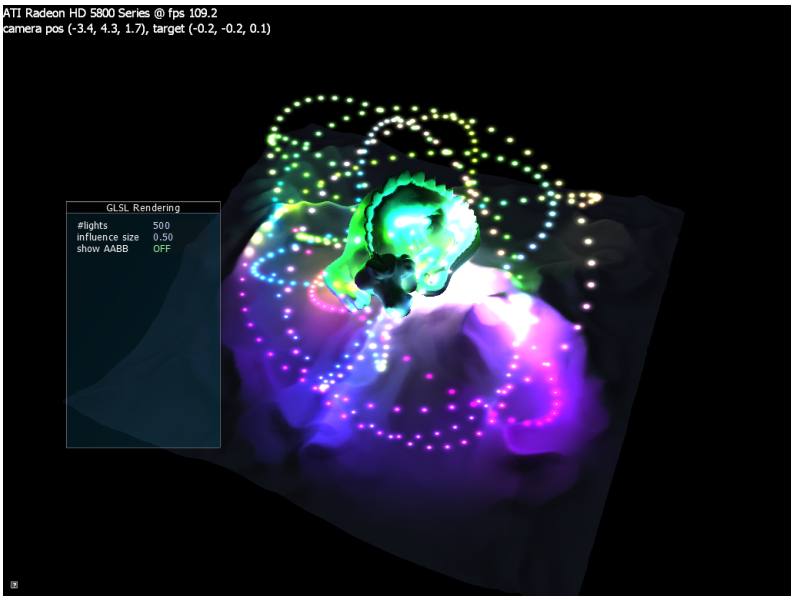
Karlsruher Institut für Technologie
Lehrstuhl für Computergrafik

Prof. Dr.-Ing. Carsten Dachsbacher Dipl.-Inf. Florian Simon

5. Übungsblatt zur Vorlesung Interaktive Computergrafik im SS 2014

Besprechung am Mittwoch, 04.06.2014



Dieses Aufgabenblatt behandelt Deferred Shading. Das Programmskelett finden Sie (wie üblich) auf der Vorlesungswebsite. Kompilieren Sie zunächst die Anwendung und machen sich mit dem Programm vertraut!
Am Ende soll die Anwendung eine Szene darstellen, die mit einer größeren Anzahl von Punktlichtquellen beleuchtet wird. Die Punktlichtquellen werden durch ein einfaches Partikelsystem animiert, sind also dynamisch. Um unnötige Arbeit zu vermeiden (und interaktive Geschwindigkeiten zu erzielen), muss sichergestellt werden, dass für jeden Pixel jede Lichtquelle maximal einmal beachtet wird. Für einen zusätzlichen Geschwindigkeitszuwachs soll weiterhin der Einflußbereich der Lichtquellen begrenzt werden. Beide Ziele sind mit Deferred Shading relativ einfach zu realisieren.

Aufgabe 1 *Erzeugung des G-Buffers*

Deferred Shading ist ein Zwei-Schritt-Verfahren. Im ersten Schritt werden alle Attribute, die für das Shading notwendig sind (Position, Normale usw.), im Bildraum gesammelt und in einem G-Buffer (Geometry-Buffer) gespeichert. In der Praxis handelt es sich dabei um eine Reihe von Texturen, die als Render-Target an die Grafik-Pipeline gebunden werden. Die Framebuffer-Objects hierzu werden im Programmskelett in `main.cpp` in der Methode `createTextures` angelegt.
Das Shader-Programm `createDS.(v|f)p.glsl` ist für die Erzeugung des G-Buffers zuständig. Vervollständigen Sie dieses Programm, damit Position und Normale im G-Buffer gespeichert werden. Verwenden Sie zum Speichern die `out`-Variablen `out_pos` und `out_nrml`. Speichern Sie in `out_nrml.w` zusätzlich den Abstand zur Kamera; dieser wird später beim Shading benötigt.

Sie können sich den Inhalt des G-Buffers mit Hilfe des Texture-Managers anschauen; um ihn zu (de)aktivieren verwenden sie zur Laufzeit die F10-Taste (Scrolling mit F11/F12). Alle Texturen, die über diese Hilfsklasse erzeugt oder geladen werden, tauchen automatisch in dieser Ansicht auf.
(Bei Interesse können Sie in `texmanager.(v|f)p.glsl` bzw. `texturemanager.(h|cpp)` herausfinden, wie der Texture-Manager funktioniert.)

Aufgabe 2 *Point-Sprite-Expansion*

Damit die Beleuchtungsberechnung für alle Partikel/Punktlichquellen durchgeführt wird, muss Geometrie erzeugt werden, um das Fragment-Programm der Beleuchtungsphase anzustoßen. In dieser Aufgabe soll für jedes Partikel ein Quad dargestellt werden.
Den relevanten Code finden Sie im Geometry-Shader `useDS.gp.glsl`; als Eingabe erhält dieser jeweils einen Punkt (`layout(points) in;`), der bereits Position und Intensität/Farbe der Lichtquelle enthält. Da Quads in OpenGL nicht als Ausgabe von Geometry-Shadern erlaubt sind, wird stattdessen ein Triangle-Strip von zwei Dreiecken verwendet (`layout(triangle_strip, max_vertices = 4) out;`). Anfangs gibt der Geometry-Shader noch keine Geometrie aus.
Das Quad soll durch die vier (homogenen) Eckpunkte

$$\begin{pmatrix} \texttt{lightPosSS.x} \pm \texttt{offsetX} \\ \texttt{lightPosSS.y} \pm \texttt{offsetY} \\ -0.5 \\ 1 \end{pmatrix}$$

aufgespannt werden. `lightPosSS` ist die Position der Lichtquelle in Clip-Koordinaten und ist bereits vorgegeben. Sie müssen noch den „Radius“ des Quads in x- und y-Richtung berechnen und in den Variablen `offsetX` bzw. `offsetY` speichern. Der Vektor `up` gibt den Einflussbereich des Lichts in Weltkoordinaten vor, und zeigt von der Lichtquelle aus nach oben. `offsetX` ist die Länge dieses Vektors *in Clip-Koordinaten*. Um sicherzustellen, dass der Einflussbereich quadratisch ist, muss das Seitenverhältnis des Bildschirms (`aspect`) beachtet werden. Demnach sollte `offsetY` das Produkt von `offsetX` und `aspect` sein.
Wenn Sie alles richtig gemacht haben, sollten Sie die Lichtquellen am Bildschirm sehen können.

Aufgabe 3 *Beleuchtung mit vielen Lichtquellen*

Implementieren Sie nun die Beleuchtung im Fragment-Shader `useDS.fp.glsl`! Vorerst stellt dieser nur die Lichtquellen als Glare-Effekt dar und zeichnet deren Einflussradius, sofern letzteres per AntTweakBar-GUI aktiviert ist.
Laden Sie zunächst die geometrischen Attribute aus dem G-Buffer und speichern Sie sie in den Variablen `posWS`, `nrml` und `mat`. Welche Texturkoordinaten benötigen Sie dafür? Müssen die Texturkoordinaten im Geometry-Shader berechnet werden? (*Hinweis:* Was ist mit der Built-in `uniform`-Variablen `gl_FragCoord`?)
Berechnen Sie danach die Beleuchtung (gehen Sie von einem Lambertschen Reflektor aus) und speichern Sie sie in der Variablen `intensity`. Wenn Sie alles richtig gemacht haben, sollten Sie nun die Szene inklusive der Beleuchtung durch das Partikelsystem sehen können.
Experimentieren Sie mit Anzahl und Einflussradius der Lichtquellen! Wie wirkt sich der Radius auf die Füllrate aus? Eine Punktlichtquelle hat theoretisch einen unendlichen Support. Wieso kann man trotzdem irgendwann abschneiden?