Institut für Visualisierung
und Interaktive Systeme

Dr. Guido Reina Dr. Michael Krone

10. Übungsblatt zur Vorlesung Computergraphik im WS 2017/18

Abgabe am Mittwoch, 17.01.2018 (12:00 Uhr)
Besprechung am Montag, 22.01.2018 (17:30 - 19:00 Uhr)

Aufgabe 1 *Blending und Tiefentest [2.5 Punkte]*

Wir betrachten im Folgenden einen bestimmtem Pixel auf dem Bildschirm. Dabei findet folgender Ablauf von Ereignissen statt:

1. `glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_COLOR);`
2. `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
3. `glDisable(GL_BLEND);`
4. `glEnable(GL_DEPTH_TEST);`
5. Drawaufruf erzeugt Fragment $(z = 0.6, r = 1.0, g = 0.3, b = 0.7, a = 0.5)$.
6. `glEnable(GL_BLEND);`
7. Drawaufruf erzeugt Fragment $(z = 0.8, r = 0.8, g = 0.2, b = 0.5, a = 1.0)$.
8. `glDisable(GL_DEPTH_TEST);`
9. Drawaufruf erzeugt Fragment $(z = 0.4, r = 0.2, g = 0.5, b = 0.6, a = 0.0)$.
10. `glEnable(GL_DEPTH_TEST);`
11. Drawaufruf erzeugt Fragment $(z = 0.5, r = 0.7, g = 0.9, b = 0.8, a = 1.0)$.
12. `glBlendFunc(GL_DST_ALPHA, GL_ONE_MINUS_SRC_ALPHA);`
13. `glDisable(GL_DEPTH_TEST);`
14. `glDisable(GL_BLEND);`
15. Drawaufruf erzeugt Fragment $(z = 0.1, r = 1.0, g = 1.0, b = 0.5, a = 0.4)$.

Welche Werte nimmt der Pixel jeweils unmittelbar nach den Schritten 5, 7, 9, 11 und 15 an? Geben Sie sowohl Farbe als auch z-Wert, sowie den Rechenweg an. Begründen Sie Ihre Antwort!

Alle Informationen, die für das Lösen der Aufgabe notwendig sind, finden Sie auch in den OpenGL-Referenzen: `https://www.khronos.org/registry/OpenGL-Refpages/gl4/`

Aufgabe 2 *Screen Space Ambient Occlusion in OpenGL [7.5 Punkte]*

Der ambiente Term des (Blinn-)Phong-Beleuchtungsmodell leuchtet alle Punkte in der Szene gleichmäßig aus, unabhängig von Umgebungsverdeckung. Es soll dadurch eine sehr grobe Approximation von globalen Beleuchtungseffekten modelliert werden. In der Realität hängt die globale Beleuchtung eines Punktes aber auch maßgeblich von der Umgebung ab. Auf einen Punkt, der sich in einer Ecke oder Nische befindet, fällt normalerweise weniger Licht als auf eine freie Fläche. Bei der Umgebungsverdeckung (*Ambient Occlusion*) wird dieser Effekt approximiert. Bei *Screen Space Ambient Occlusion* (SSAO) wird für jeden Punkt der Szene, der im Bild sichtbar ist, innerhalb einer Halbkugel über dem Punkt die Umgebung abgetastet. Dadurch lässt sich überprüfen ob sich dort Geometrie befindet, die den Punkt verdeckt. Je mehr Geometrie den Punkt verdeckt, desto mehr wird der Punkt verschattet. Die Intensität des ambienten Terms wird entsprechend gesenkt.

Zunächst wird das Bild in einen Framebuffer gerendert und kann somit für einen weiteren *Render-Pass* als Textur verwendet werden. Um die Umgebungsverdeckung zu berechnen, benötigt man pro Pixel nicht nur die Farbe, sondern auch den zugehörigen Normalenvektor und die Tiefe in Kamerakoordinaten. Diese Werte werden deshalb ebenfalls in eine Textur geschrieben. Anschließend gibt es einen zweiten *Render-Pass*, in dem die Umgebungsverdeckung für jeden Pixel berechnet wird. Bei unserer SSAO-Implementierung werden auch die im zweiten *Render-Pass* berechneten Werte der Umgebungsverdeckung in eine Textur gerendert. Erst in einem dritten Durchlauf werden diese weichgezeichnet, um ein Rauschen zu verringern, und mit den Farbwerten des Objekts multipliziert und auf dem Bildschirm angezeigt.

Weitere Informationen zu einer möglichen Implementierung von SSAO finden Sie hier:
`https://john-chapman-graphics.blogspot.de/2013/01/ssao-tutorial.html`.

Über ILIAS erhalten Sie das Visual-Studio-Projekt SSAO mit einer unvollständigen Implementierung, wobei alle zu bearbeitenden Stellen mit TODO gekennzeichnet sind.

Hinweis: Unterstützt werden Visual Studio 2015 und 2017. Bei der Installation müssen Sie unter „Einzelne Komponenten“ den „NuGet-Paket-Manager“ aktivieren. Sollten Sie Visual Studio 2017 verwenden, brauchen Sie zusätzlich das „Toolset für VC++ 2015.3 v140“. Diese Features lassen sich auch im Nachhinein mit dem Visual Studio Installer hinzufügen. Achten Sie bitte auch darauf, beim ersten Öffnen des Projekts, die verwendete Toolset-Version *nicht* auf v141 umzustellen (kein Upgrade).

Das Programmskelett greift für die Verwendung von OpenGL auf externe Open-Source-Bibliotheken zu. In der mitgelieferten Visual-Studio-Solution sind die Bibliotheken bereits eingebunden und werden (sofern der NuGet-Paket-Manager installiert ist) automatisch nachgeladen. Falls Sie einen anderen Compiler verwenden wollen, müssen Sie sich selbst um die Einbindung folgender Bibliotheken kümmern:

- glfw: http://www.glfw.org/
- glad: https://github.com/Dav1dde/glad

Wenn Sie das Projekt öffnen und ausführen, bleibt das Bild vorerst schwarz. Die ersten Resultate werden erst nach dem Abschluss der ersten Teilaufgabe sichtbar. Um die Ladezeit zu verkürzen, sollten Sie zudem das Projekt im *Release*-Modus kompilieren.

Mit den Tasten W, A, S, D, *Shift* und *Space* kann man sich im Raum bewegen. Die Pfeiltasten erlauben eine Änderung der Blickrichtung. Über die Tasten 1 bis 5 können einzelne oder zusammengesetzte Methoden betrachtet werden: (1) nur Farbe, (2) Farbe und SSAO, (3) Blinn-Phong (Standardauswahl), (4) Blinn-Phong mit SSAO, (5) SSAO-Effekt.

1. Um überhaupt ein Bild anzuzeigen, muss zuerst die Verwendung des Framebuffers implementiert werden. Dieser wird verwendet, um die Zwischenergebnisse aus den vorherigen Render-Passes abzuspeichern. Im ersten Durchgang werden dabei nur die Normalen und die Tiefen aller Fragmente berechnet. Zudem wird die Farbe des Objekts abgespeichert. Im zweiten Durchlauf wird dann das *Ambient Occlusion* berechnet.

   Die Framebuffer sollen in der Funktion `SSAO_Renderer::UpdateFramebuffers()` initialisiert werden. Hinweise zur Vorgehensweise finden Sie in den Kommentaren.

   Da der Drache einfarbig gehalten ist, lassen sich nach erfolgreicher Implementierung dieser Teilaufgabe nur seine Umrisse und keinerlei Tiefe erkennen. Durch den SSAO-Effekt soll sich dies ändern.

2. Zunächst soll jedoch das Blinn-Phong-Modell für die lokale Beleuchtung im Fragment-Shader `final_frag.glsl` implementiert werden. Die dafür benötigten Größen wurden bereits berechnet. Da für die Lichtquelle angenommen wird, dass sie unendlich weit vom Objekt entfernt ist, ist der Lichtstrahl konstant. Eine Abnehmende Intensität durch die Entfernung entfällt dadurch ebenfalls.

   Nach der erfolgreichen Implementierung, wird der Drache lokal beleuchtet dargestellt.

3. In `ao_frag.glsl` finden Sie die unvollständige SSAO-Implementierung. Darin wird über mehrere Punkte (*Samples*) iteriert, die sich in einer Halbkugel über dem zu schattierenden Punkt in Kamerakoordinaten befinden. Fügen Sie Code hinzu, der für diese Punkte die Tiefendifferenz zur davor bzw. dahinter liegenden Geometrie berechnet und in die Variable `diff` schreibt. Wählen Sie das Vorzeichen der Differenz so, dass eine positive Differenz dafür steht, dass das Sample zur Umgebungsverdeckung beiträgt.

   Hinweis: Die Tiefe eines Punkts ist in der w-Komponente der Textur `tex_normal_depth` enthalten. Um auf die richtige Stelle der Textur zuzugreifen, müssen die *Samples* in normalisierte Gerätekoordinaten transformiert und in den Bereich $[0, 1]^2$ gebracht werden.

   Nach korrekter Lösung dieser Aufgabe ist der SSAO-Effekt sichtbar, siehe Abbildung 1.

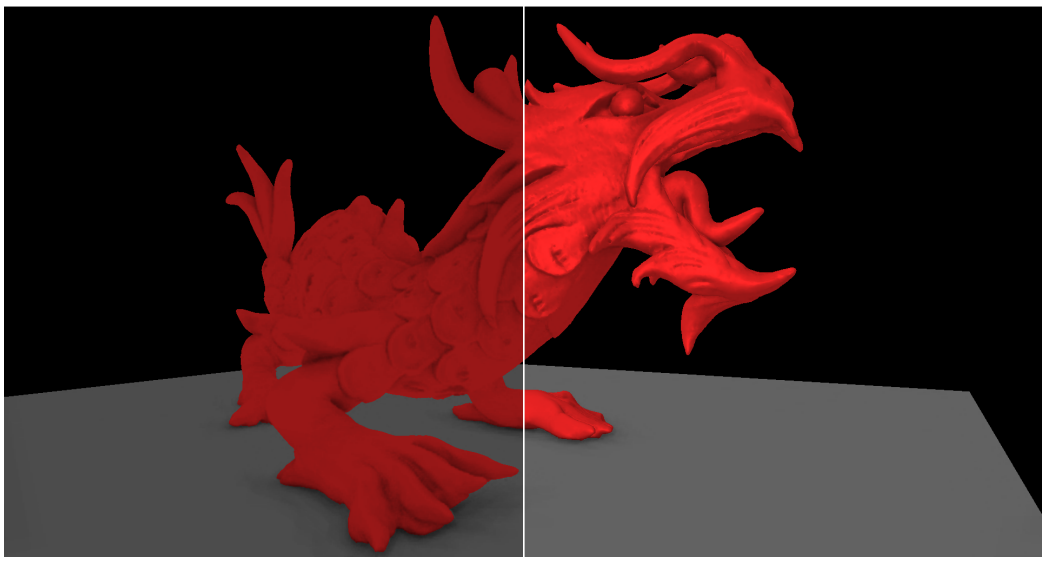

Abbildung 1: Screenshot nach korrekter Lösung dieser Aufgabe. Linke Hälfte zeigt den SSAO-Effekt des roten Drachen, rechte Hälfte Blinn-Phong mit SSAO.