Karlsruher Institut für Technologie
Lehrstuhl für Computergrafik

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

2. Übungsblatt zur Vorlesung Interaktive Computergrafik im SS 2014

Besprechung am Mittwoch, 07.05.2013

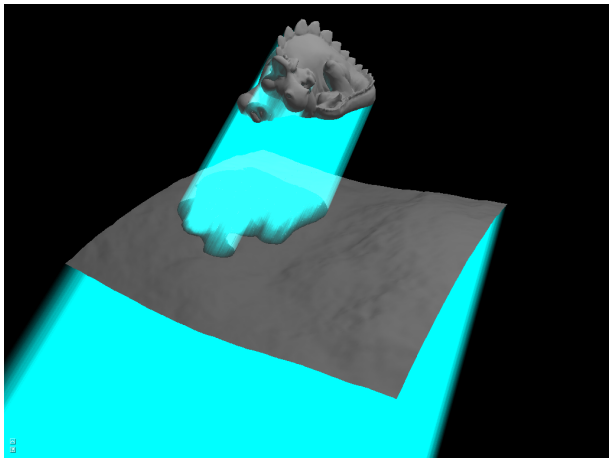

Abbildung 1: Visualisierte Schattenvolumen.

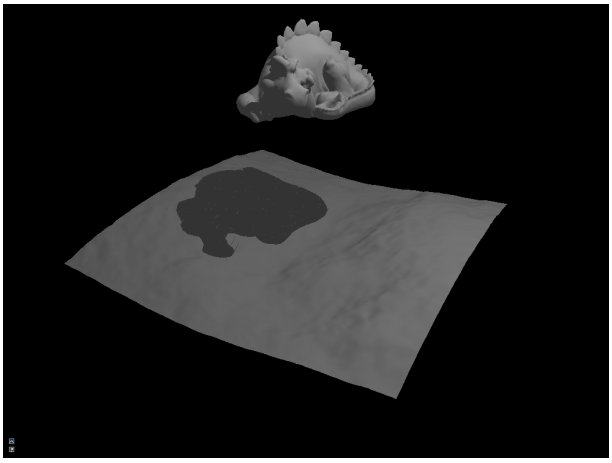

Abbildung 2: Lösungsbild.

In diesem Aufgabenblatt sollen Sie den Schattenvolumen-Algorithmus für Punktlichtquellen, den Sie in der Vorlesung kennengelernt haben, implementieren. Das entsprechende Programmskelett finden Sie auf der Vorlesungswebsite.

In der ersten Aufgabe werden Sie erstmals einen Geometry-Shader verwenden. Dieser kann neben der Anzahl auch die Art der Primitive verändern. Die Deklaration `layout(triangles) in;` gibt z.B. an, dass einzelne Dreiecke als Eingabe dienen. Die Deklaration `layout(triangle_strip, max_vertices = 11) out;` gibt zweierlei an: Zum einen werden Dreiecksstreifen ausgegeben, zum anderen wird die maximale Anzahl der ausgegebenen Vertices auf 11 begrenzt. Letzteres ist notwendig, da zwar eine variable Anzahl Vertices/Primitiven pro Aufruf ausgegeben werden kann, diese Anzahl jedoch durch Hardwarebeschränkungen nach oben hin begrenzt ist.

Zugriff auf die (Fixed-Function-)Eingaben haben Sie über das Array `gl_in[]`, dass eine Reihe von Strukturen mit Ausgabeattributen (wie z.B. `gl_Position`) des Vertex-Shaders enthält. Um Vertices auszugeben, wird der Aufruf `EmitVertex()` verwendet, nachdem die entsprechenden Ausgabeattribute gesetzt wurden. Um das aktuelle Primitiv zu beenden (und evtl. weitere Primitive auszugeben), nutzt man den Aufruf `EndPrimitive()`.

Hinweise:

- Das Programmm verwendet
  - GLFW (2.7.x) (http://www.glfw.org/),
  - GLEW (http://glew.sourceforge.net/),
  - GLM (http://glm.g-truc.net/) und
  - Anttweakbar (http://anttweakbar.sourceforge.net/doc/).
- Unter Windows steht Ihnen eine Visual Studio Solution zur Verfügung, unter Linux kann das Programm mit `make` kompiliert und mit `make start` ausgeführt werden. Das `start`-Skript trägt die kompilierten Libraries in die `LD_LIBRARY_PATH` Variable ein und führt das Programm anschließend aus.
- Falls unter Windows Linker Probleme bestehen, kann es hilfreich sein, die mitgelieferten libs und dlls mit den vorkompilierten Varianten der jeweiligen Projektwebsiete zu ersetzen.

Aufgabe 1 *Schattenvolumen-Erzeugung*

Extrudieren Sie im Geometry-Shader `volume.gs.glsl` die Seitenflächen des Dreiecks. Berechnen Sie dazu die Vektoren von den Eckpunkten zur Lichtquelle.
Der Geometry-Shader ist momentan so programmiert, dass er ein Dreieck als Dreiecksstreifen ausgibt. Erzeugen Sie die Seitenfläche des Schattenvolumens, indem Sie den Dreiecksstreifen berechnen und ausgeben! Dabei können Sie sich an dem bereits vorhandenen Code orientieren. Zunächst werden als Schattenvolumen nur die ursprünglichen Dreiecke rot dargestellt.

Aufgabe 2 *Visualisierung der Tiefenkomplexität*

Um einen Eindruck von der Tiefenkomplexität des Schattenvolumens zu erhalten, sollen Sie nun die Visualisierung aus der letzten Aufgabe so abändern, dass Sie mehrere Schichten des Volumens darstellen.
Konfigurieren Sie dazu in der `OnRender`-Funktion in `user.cpp` vor dem Zeichnen des Schattenvolumens den OpenGL-Zustand so, dass das Volumen mit additivem Alpha-Blending dargestellt wird, d.h. die konstanten Farbwerte der Schattenvolumen im Framebuffer akkumuliert werden.

- Deaktivieren Sie den Tiefentest.
- Deaktivieren Sie den Stencil-Test.
- Aktivieren Sie Blending.
- Setzen Sie die richtige `glBlendFunc` und `glBlendEquation`.
- Geben Sie im Fragment-Shader `volume.fs.glsl` einen Konstanten α-Wert < 1 aus.

Aufgabe 3 *Strahlschnitt mit dem Stencil-Buffer*

Um festzustellen, ob ein Punkt im Schatten liegt, muss man berechnen, wie oft ein Augstrahl das Schattenvolumen schneidet, bis er auf eine Oberfläche trifft: Ist die Anzahl ungerade, befindet sich der Punkt im Schatten, sonst nicht. Die einfachste Art, dass auszunutzen ist der z-Pass-Algorithmus, den Sie im Folgenden implementieren sollen.
Die Anwendung ist bereits so konfiguriert, dass die Beleuchtung nur dort berechnet wird, wo der Stencil-Wert 0 ist. Stellen Sie in der `OnRender`-Funktion den OpenGL-Zustand so ein, dass Front-Faces des Schattenvolumens den Stencil-Wert inkrementieren, Back-Faces den Wert dekrementieren (*Hinweis*: `glStencilOpSeparate`).
Vergessen Sie nicht, den Tiefentest wieder einzuschalten (`glEnable`), aber den Schreibzugriff auf Farb- und Tiefenpuffer zu deaktivieren (`glColorMask`, `glDepthMask`)!
Wenn Sie alles richtig gemacht haben, sollten Sie nun die Szene mit korrekten Schatten sehen können.

Aufgabe 4 *Alpha-Blending statt Stencil-Tests*

Hinweis: Für diese Aufgabe müssen Sie nichts implementieren.

Könnte man den Schattenvolumen-Algorithmus auch ohne Stencil-Buffer und nur mittels Alpha-Blending implementieren? Gehen Sie von folgender Idee aus:

- Zu Beginn wird der α-Wert im Framebuffer auf $1/4$ gesetzt.
- Beim Zeichnen von Front-Faces des Schattenvolumens multipliziert man den α-Wert mit 2.
- Beim Zeichnen von Back-Faces des Schattenvolumens multipliziert man den α-Wert mit $1/2$.

Wie kann man diese Idee nutzen, um festzustellen, ob ein Fragment im Schatten liegt? Wie müssten Sie die OpenGL-Pipeline konfigurieren, um die obigen 3 Punkte umzusetzen?