Karlsruher Institut für Technologie
Lehrstuhl für Computergrafik

Prof. Dr.-Ing. Carsten Dachsbacher
Dipl.-Inf. Christoph Schied

2. Übungsblatt zur Vorlesung Interaktive Computergrafik im SS 2017

Besprechung am **Mittwoch, 31.05.2017**, 10:30 Uhr.

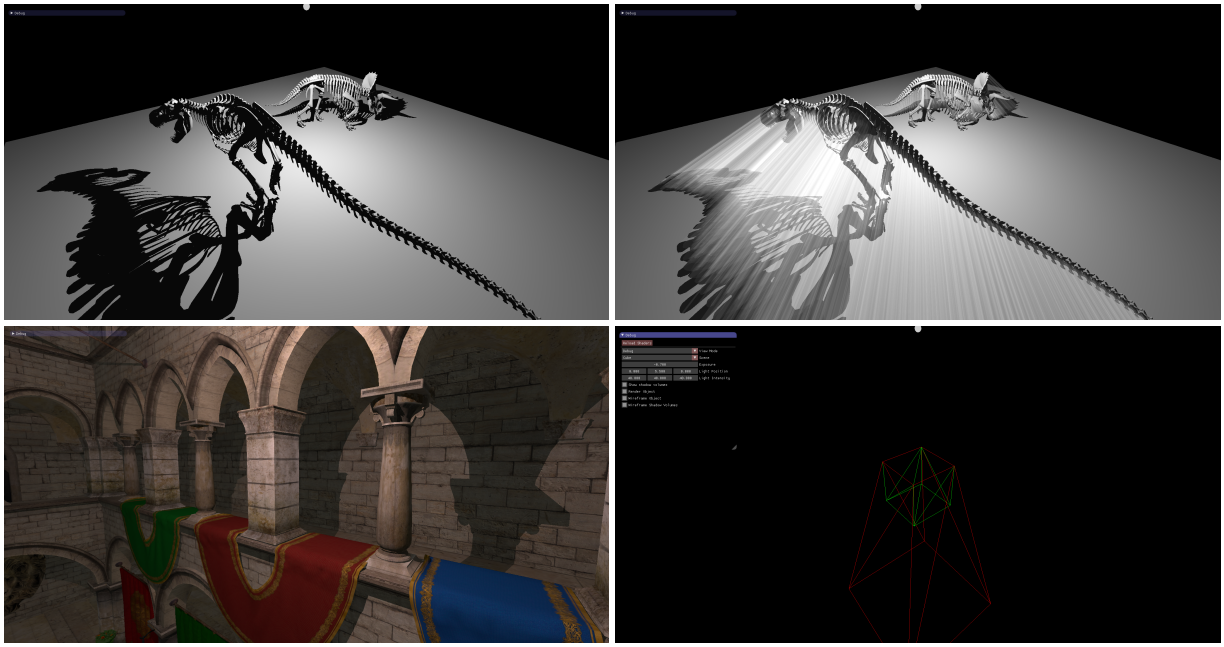

Abbildung 1: Ergebnisbilder des Übungsblattes.

Dieses Übungsblatt beschäftigt sich mit dem Z-Pass Schattenvolumen-Algorithmus den Sie in der Vorlesung kennengelernt haben. Die Ergebnisbilder einer möglichen Implementierung sind in Abbildung 1 dargestellt. Ein entsprechends Programmskelett ist im ILIAS bereitgestellt.

Im ersten Aufgabenteil werden Sie einen Geometry-Shader (`shader/shadow_volumes.geom`) zur Erzeugung der Schattenvolumen verwenden, welcher sowohl Anzahl als auch Art der Primitive verändern kann.

Die Deklaration `layout(triangles) in;` im Geometry-Shader 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()`.

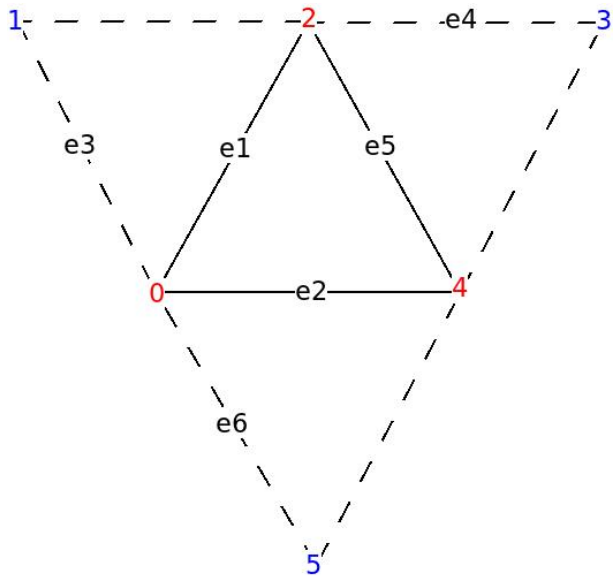

Abbildung 2: Adjazenzinformationen beim OpenGL-Primitiv `GL_TRIANGLE_ADJACENCY`. In rot sind die Vertizes des Dreieckes dargestellt, die blauen Vertizes erlauben die Rekonstruktion der Nachbardreiecke.

Aufgabe 1 *Erzeugung der Schattenvolumen*

Extrudieren Sie im Geoemtry-Shader `shadow_volume.geom` die Silhouetten des Dreiecksnetzes!

Hierfür müssen Sie zuerst Kanten des Dreieckes identifizieren, die Teil der Schattensilhouette sind. Eine Kante ist ein Teil der Silhouette wenn das bearbeitete zur Lichtquelle hingewandt, und das Nachbardreieck mit geteilter Kante von der Lichtquelle abgewandt sind. Implementieren und benutzen Sie hierfür die Funktion `triangle_sign`.

Die benötigten Informationen über Nachbardreiecke können im Geometryshader durch den speziellen Primitivtyp `GL_TRIANGLE_ADJACENCY` bereitgestellt werden (siehe Abbildung 2). Der hierfür benötigte spezielle Indexpuffer wird bereits durch das Framework bereitgestellt.

Aufgabe 2 *Visualisierung der Schattenvolumen*

Um einen Eindruck von der Tiefenkomplexität des Schattenvolumens zu erhalten, sollen Sie nun mehrere Schichten des Volumens darstellen.

Konfigurieren Sie dazu in der `draw_shadowed`-Funktion in `renderers.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.

Aufgabe 3 *Schattentest*

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.

Stellen Sie in der `draw_shadowed`-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`)!

Anschliessend wird das Dreiecksnetz mit einem Shader für die Beleuchtungsberechnung gezeichnet. Konfigurieren Sie den OpenGL Zustand derart dass der Beitrag nur für Punkte, die nicht im Schatten liegen, akkumuliert wird.

Framework

Wir stellen für jedes Übungsblatt ein Framework bereit. Das Framework nutzt C++ 11 und wird unter Linux getestet. Es ist allerdings auch unter Windows mit Visual Studio 2013 lauffähig. Die Datei `Kompilieren.txt` enthält Informationen darüber, wie sie das Framework kompilieren.

Hinweis: Wenn sie Visual Studio verwenden sollten Sie das Projekt im Release oder Release-WithDebug starten um die Ladezeiten zu reduzieren.