Aufgabenstellung WS 2014/15

Real-Time Rendering

Prof. Christof Rezk-Salama

– Abgabe spätestens 31.03.2015 –

1 Aufgabenstellung

Ziel des Projektes ist die Umsetzung eines prozeduralen, rekursiven Baum mit Hilfe von Transform Feedback in *OpenGL*. Die Aufgabe ist für die Bearbeitung durch Einzelpersonen gedacht.
Es ist sehr anzuraten bei der Entwicklung eine entsprechend leistungsfähige Grafik-Karte zu verwenden.Die Bearbeitung der Aufgabe erfolgt in C++ und OpenGL für das Rahmenprogramm. Shader in GLSL. Ziel der Aufgabe ist eine möglichst effiziente Umsetzung des unten beschriebenen Prozesses. Im weiteren Verlauf dieses Dokuments wird eine Implementierungs-Skizze für die Aufgabenstellung vorgeschlagen. Diese ist lediglich als Erklärung und Empfehlung vorgegeben. Solange das Ergebnis ähnlich ist, kann durchaus von den konkreten Implementierungsvorschlägen abgewichen werden. Die Konstruktion des Baumes **muss** allerdings im Geometry-Shader erfolgen.

1.1 Transform Feedback

Transform Feedback ist eine Technik, bei der in der Grafik-Pipeline nur der Vertex- und der Geometry-Shader durchlaufen wird und die Berechnung vor der Rasterisierung abgebrochen wird. Das Ergebnis der Berechnung des Vertex- und Geometry-Shaders wird zurück in einen Vertex-Buffer geschrieben wird, der für weitere Passes verwendet wird. Auf diese Weise kann man mit dem Geometry-Shader Geometrien erzeugen und diese im Grafikspeicher zwischenspeichern um sie erst später zu rendern. Das hat den Vorteil, das der Geometry-Shader die Geometrie nur einmal erzeugen muss und nicht jedesmal, wenn die Pipeline durchlaufen wird.
Ausführliche Information zu Transform Feedback in OpenGL finden Sie unter

`https://www.opengl.org/wiki/Transform_Feedback`

1.2 Prozeduraler Baum

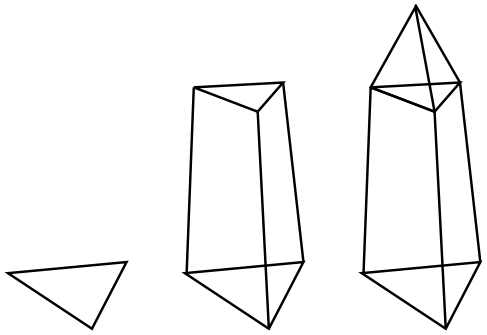
Der Baum, der konstruiert werden soll ist eine 3D-Variante des bekannten Pythagoras-Baumes. Ihre Aufgabe ist aus einem einzigen planaren Dreieck durch rekursives Transform-Feedback einen Baum wachsen zu lassen.
Im ersten Schritt soll nur ein einzelnes Dreieck in die Pipeline geschickt werden. Die Attribute, die sie an die 3 Vertices speichern, dürfen Sie selbst bestimmen. Mein Vorschlag: Die Vertices der Dreiecke speichern neben ihrer *Position* und ihres *Normalenvektors* einen weiteren skalaren Parameter `length`. Dieser Parameter gibt an, wie weit das Dreieck extrudiert werden soll.
Die Aufgabe teilt sich dann in mehrere rekursive Konstruktions-Passes und einen Renderpass.

Konstruktions-Passes:
Die komplette Konstruktion des Baumes findet in Modellkoordinaten statt. Der Vertex-Shader für den Konstruktions-Pass muss daher weder die Positionen noch die Normalenvektoren in irgend einer Weise transformieren.
Der Geometry-Shader bekommt einzelne Dreiecke übergeben und prüft zunächst anhand des `length`-Parameter des ersten Vertex, ob dieses Dreieck extrudiert werden muss oder nicht.

- In dem Falle, dass es nicht extrudiert werden muss, weil der `length`-Parameter gleich 0 ist, gibt der Geometry-Shader das Dreieck einfach genau so aus, wie er es bekommen hat.

- Ist der *length*-Parameter der Vertices jedoch größer als 0, so konstruiert der Geometry aus dem Dreieck zunächst einen Stamm: Es werden zunächst drei neue Vertices bestimmt, indem er die ursprünglichen Vertices um den gegebenen *length*-Betrag entlang ihrer Normalen verschoben und um einen kleinen Betrag zu ihrem gemeinsamen Mittelpunkt hin verschoben werden. Aus den ursprünglichen und den neuen Vertices wird die 'Mantelfläche des Stammes konstruiert. Der `length`-Parameter für diese Vertices soll auf 0 gesetzt werden, damit sie im nächsten Schritt nicht nochmal extrudiert werden.

Überlegen Sie sich vorher genau, wie viele Vertices dabei entstehen, denn dies ist eine wichtige Information, die sowohl der Geometry-Shader benötigt, als auch das Rahmenprogramm beim Anlegen der Vertex-Buffer. Sie benötigen mindestens zwei Vertex-Buffer für das Transform-Feedback, die sie wie eine Art Double-Buffer verwenden: Aus dem einen wird die Geometry gelesen, in den anderen wird das Ergebnis geschrieben. Anschließend werden die beiden Buffer vertauscht.



- Anschließend wird aus den drei neuen Vertices die Seitenflächen einer Pyramide konstruiert, indem der gemeinsame Mittelpunkt der drei neuen Vertices eingefügt, und ein weiteres Stück entlang der Normalen verschoben wird. Die Vertices dieser Pyramide bekommen einen `length`-Parameter, der etwas geringer ist als der ursprüngliche. Auf diese Weise werden die Äste nach oben hin immer kürzer.

- Das Ergebnis dieses Geometry-Passes werden mittels Transform-Feedback in einen freien Vertex-Buffer geschrieben. Dieser Vertex-Buffer wird erneut durch den gleiche Pipeline geschickt, um so rekursiv einen Baum zu erhalten.

Render-Pass:
Nach einigen wenigen Passes durch diese Pipeline, wird das Ergebnis mit einer normalen Vertex/Fragment-Shader-Kombination auf den Bildschirm gezeichnet. Die folgende Abbildung zeigt das Ergebnis nach unterschiedichen Rekursionsschritten. Diese Shader dürfen sie frei gestalten.

2 Schlussbemerkungen

Für die Bearbeitung der Aufgabenstellung sind - wie im späteren Berufsleben - alle Hilfsmittel erlaubt. Als Basis für die Implementierung ist es daher durchaus erlaubt und wünschenswert auf entsprechende Basisprogramme als Framework (z.B. für das Öffnen der Fenster, Erstellen der Rendering-Kontexte). Beispielprogramme sind dabei selbstständig zu recherchieren und in der Dokumentation zu referenzieren.

- Suchen Sie sich Tutorials und Beispielprogramme zu OpenGL, Geometry-Shadern und Transform Feedback.
- Sie dürfen selbstverständlich die Rahmenprogramme aus meinen Vorlesungen nutzen.
- Bitte unterschätzen Sie die Aufgabe nicht! Planen Sie genügend Zeit für die Bearbeitung ein!
- Sollten Sie bei der Bearbeitung der Aufgabe Probleme haben, melden Sie sich bitte bei mir per Email, oder machen Sie einen Gesprächstermin aus! Seien Sie sich aber bewußt, dass ich auch etwas Zeit brauche um alle Mails zu beantworten. 'Dringliche Anfragen paar Stunden vor der Deadline sind daher mit hoher Wahrscheinlichkeit wenig zielführend!

Für die Abgabe wird neben dem Programmcode eine kurze, formlose Projektdokumentation (ca. 5 Seiten) erwartet, die den Aufbau und die Bedienung des Programms erklärt. Für ein Bestehen mit 4.0 sind mehr als 50% der Teilaufgaben erfolgreich zu bearbeiten. Abgaben ohne Dokumentation werden nicht berücksichtigt.

Prozent der Aufgabenstellung	Note
90–100 %	1.0
85–89 %	1.3
80–84 %	1.7
75–79 %	2.0
70–74 %	2.3
65–69 %	2.7
60–64 %	3.3
55–59 %	3.7
51–55 %	4.0

Die Implementierung zusammen mit der Dokumentation laden Sie bitte in einem ZIP-Archiv mit Ihrem Namen gekennzeichnet in den **Hausarbeiten-Ordner** im StudIP hoch! Bitte keine Abgaben per Email!

Viel Erfolg!!

Trier, 15.01.2015

Christof Rezk-Salama