Dokumentation zur Projektaufgabe für das Modul Tool- und Pluginprogrammierung im Sommersemester 2022

Thema und Motivation

Als Thema für meine Modulaufgabe habe ich mich für **Entwicklung einer Mesh Decimation-Anwendung** entischieden, da ich in meinem nächsten privaten Projekt mit 3D-Meshes arbeiten will und ich dann potenziell Verwendung für einen Mesh Decimation Algorithmus finde.

Während meiner Recherche bin ich auf ein Paper mit dem Titel '*Surface Simplification Using Quadric Error Metrics*' von Michael Garland und Paul S. Heckbert gestoßen und habe mich dafür entschieden den in diesem Paper vorgestellten Algorithmus (zumindestens teilweise) zu implementieren.

Da ich am Anfang bisschen überfordert war und nicht wirklich wusste wo ich anfangen soll, habe ich nach anderen Implementionen gesucht, durch die ich den Algorithmus besser verstehen kann. Dabei habe ich ein GitHub Repository (`https://github.com/Meirshomron/MeshSimplification`) gefunden, das mir vor allem bei der Umsetzenung der Fehlerberechnung geholfen hat.

Bibliotheken

In diesem Projekt verwende ich vier bzw. fünf Bibliotheken, die es mir ermöglichen eine grafischen Anwendung zu entwickeln, in der der Algorithmus an einem Modell ausgeführt werden kann.

Für das Erstellen des OpenGL-Kontextes und den Zugriff auf Maus und Tastatur verwende ich **GLFW** (`https://www.glfw.org/`). Ich habe mich dafür entschieden, da ich damit die meiste Erfahrung habe und gut damit klar komme.

Um mir den Umgang mit OpenGL zu vereinfachen, verwende ich das von mir selbst entwickeltes Framework **Ignis** (`https://github.com/oliverjakobs/Ignis`). Grundelegender Bestandteil dieses Frameworks ist die OpenGL-Loading-Library **GLAD** (`https://glad.dav1d.de/`).

Da mein Programm interaktiv sein soll und ich das über eine grafische Benutzeroberfläche umsetzen will, verwende ich die Bibliothek **Dear ImGui** (`https://github.com/ocornut/imgui`).

Damit ich mir keine Gedanken machen muss, ob alle mathematischen Operationen auch wirklich richtig funktionieren (und mit OpenGL kompatibel sind), benutze ich die Mathe-Bibliothek **glm** (`https://github.com/g-truc/glm`).

Designentscheidungen

Meine Designentscheidungen unterteile ich in zwei Abschnitte. Im ersten Abschnitt geht es um allgemeine Entscheidungen, bei denen es um den Umfang des Projektes oder die Benutzerschnittstelle geht. Im zweiten Abschnitt geht es dann um Entscheidungen, die die eigentliche Implementation des Algorithmuses betreffen.

Allgemeine Entscheidungen

Bevor ich mir Gedanken gemacht habe, wie ich den eigentlichen Algorithmus implementiere, habe ich mir überlegt, was das Programm können soll und was nicht.

Das Programm soll ein 3D-Model mit dem Format `.obj` laden und anzeigen können. Dann soll über ein GUI ausgewählt werden können auf wie viele Faces das Mesh reduziert werden soll. Durch einen Button soll dann der Algorithmus mit den vorher festegelegtem Ziel ausgeführt werden. Außerdem soll es einen Button geben, der das Mesh auf seinen ursprünglichen Zustand zurücksetzt.

In dem im ersten Abschnitt erwähnten Paper werden die beiden Vertices für den Edge-Collapse

Zu Beginn habe ich mir Gedanken

Zuerst habe ich mir überlegt, was das Programm machen soll und was nicht.

Als kleines Extra habe ich noch zwei Render-Optionen hinzugefügt, die mit Checkboxen ein- und ausgeschaltet werden können.

Des Weiteren habe ich mich gegen den Export in ein Dateiformat entschieden, da es von der Komlexität eher trivial ist und ich im Normalfall in ein Engine-Spezifisches Format exportieren würde.

Flat-Shading - ermöglicht Vertices mit nur Positionen, außerdem hebt das die Faces hervor Außerdem können beim Flat-Shading die Face-Normals im Fragment-Shader berechnet werden. Dadurch ist es möglich, dass die Vertices nur aus Positionen bestehen. Ein Vertex ist also einfach nur ein *vec3*. Das macht die Implementation des Algorithmuses übersichtlicher.

Entscheidungen zur Algorithmus Implementation

In dem Paper wird das Mesh vereinfacht, indem aus einer Liste aus allen möglichen Paaren das bestmöglichste Paar ausgewählt wird. Dabei ist es nicht nötig, dass zwischen den beiden Verices eines Paares auch eine Kante existiert, solange die Distanz zwischen den Vertices einen festgelegten Threshold nicht überschreitet. Um den Umfang von meinem Projekt nicht zu groß werden zu lassen, habe ich mich entschieden diesen Threshold auf Null zu setzen und nur Paare zuzulassen, zwischen dessen Vertices auch eine Kante existiert.

Für den von mir gewählten Algorithmus braucht ein Vertex eine Position und eine Fehler-Matrix. Das realisiere ich durch zwei Vektoren (einen für die Positionen und einen für die Fehler-Matrizen), indem ich den Vertex durch einen Index darstelle. Um jetzt an die Position eines Vertex v zu gelangen, muss nur der v-te Eintrag in den Vector der Positionen ausgelesen werden. Für die Fehler-Matrix eines Vertex funktioniert das analog.

Pairs - Stellt alle Paare an vertices da, die zu einem neuen Vertex zusammengefasst werden können.

Bei erstellen dieser Paare verwende ich ein `std::unordered_set` um Duplikate zu verhindern. Dabei sind zwei Paare a und b identisch, wenn $(a.first = b.first \wedge a.second = b.second) \vee (a.second = b.first \wedge a.first = b.second)$.

Die Paare werden (mit `std::make_heap`) in einem Min Heap verwaltet, wobei die Sortierung anhand des Fehlers erfolgt.

Vertices werden nicht entfernt - Es werden nur die Indices verändert bzw. entfernt. Das Entfernen der überflüssigen Vertices würde ich erst beim exportieren machen

Bedienung

Im ersten Abschnitt des GUIs (*Mesh*) werden die Anzahl der Vertices und der Faces angezeigt. Außerdem können hier die verfügbaren Meshes ausgewählt und geladen werden. Zu Verfügung stehen ein einfacher Würfel (8 Vertices, 12 Faces), ein Affenkopf (2012 Vertices, 3936 Faces) und die Marsienne Base (13235 Vertices, 27894 Faces) aus Übung03.

Der zweite Abschnitt ist die Bedienung des Simplifiers. Hier kann mit einem Slider die gewünschte Anzahl an Faces ausgewählt werden (der Slider geht von 0 bis zur Anzahl der Faces der aktuellen Version des Meshes). Mit dem Button `Simplify` wird das Mesh so lange vereinfacht, bis die gewünschte Anzahl an Faces erreicht wurde. Mit dem Button `Reset` wird die ursprüngliche Version des Meshes wieder hergestellt.

Im letzten Abschnitt befinden sich zwei Render-Optionen mit denen der Wireframe-Modus ein- und ausgeschaltet bzw. das Backface Culling aktiviert und deaktiviert werden kann.

Mit Hilfe der Maus kann die Kamera auf einem Arcball um das Mesh bewegt werden.

Mit `Esc` kann das Programm beendet werden.

Build

Zur Project Generation benutze ich Premake (`https://premake.github.io/`). Ich habe die entsprechenden Scripts meiner Abgabe beigefügt. Falls also Probleme mit der VisualStudio Solution autreten sollten, kann mit dem folgenden Befehl das Projekt neu generiert werden und so die Probleme hoffentlich gelöst werden:

```
.\premake\premake5.exe [action]
```

Für mein Projekt habe ich `vs2019` als `action` verwendet. Andere Möglichkeiten sind hier `https://premake.github.io/docs/Using-Premake` aufgelistet.