Prof. G. Zachmann
Roland Fischer
Andre Mühlenbrock

Universität Bremen
Fachbereich 3
Arbeitsgruppe CGVR
3. November 2022

Wintersemester 2022/23

Übungen zu Computergrafik - Blatt 3

Abgabe am 13.11.2022, 23:59 Uhr

Aufgabe 1 (OpenGL-API (C++), *6 Punkte*)

Im neuen Framework sind im Vergleich zur vorherigen Aufgabe zwei neue Klassen namens `Mesh` und `Shader` sowie zwei Strukturen `Triangle` und `Vertex` hinzugekommen.

- Die `Mesh`-Klasse soll in künftigen Aufgabenblättern dazu dienen, beliebige 2D- oder 3D-Objekte, die aus Dreiecken bestehen, auf die Grafikkarte hochzuladen und mithilfe von OpenGL auf dem Bildschirm zu rendern. Der Konstruktor eines Meshes erhält dazu eine Liste mit Dreiecken, also ein `std::vector<Triangle>` (denke daran, `std::vector` ist vergleichbar mit einer ArrayList und kein mathematischer Vektor).
- Dreiecke (`Triangle`-Struct) enthalten wiederum drei Vertices (`Vertex`-Struct), die wiederum eine Position, eine Normale und eine Farbe enthalten (diese werden jeweils mithilfe unserer Vec4f-Klasse aus dem letzten Übungsblatt gespeichert).
- Die `Shader`-Klasse dient als eine Wrapper-Klasse eines Shaders (bestehend aus Vertex- und Fragment-Shader) – diese Shader-Klasse ist für dieses Aufgabenblatt allerdings nur bedingt relevant und bereits fertig implementiert.

In der `main.cpp` findest Du folgende Stelle, an der ein Mesh-Objekt erstellt wird, welches lediglich ein Dreieck enthält:

```
87    // Create a triangle:
88    std::vector<Triangle> triangles;
89    {
90        // Create vertex a of triangle:
91        Vertex vertexA(Vec4f( 0.866f, -0.5f, 0.0f) * 0.5f,Vec4f(0,0,1,0),Vec4f(1,0,0));
92
93        // Create vertex b of triangle:
94        Vertex vertexB(Vec4f(0,1.f,0) * 0.5f,Vec4f(0,0,1,0),Vec4f(0,0,1));
95
96        // Create vertex c of triangle:
97        Vertex vertexC(Vec4f(-0.866f, -0.5f,0) * 0.5f,Vec4f(0,0,1,0),Vec4f(0,1,0));
98
99        // Create triangle:
100       Triangle triangle(vertexA, vertexB, vertexC);
101
102       // Add triangle to the list of triangles:
103       triangles.push_back(triangle);
104   }
105
106   // Create our mesh containing one triangle:
107   Mesh triangleMesh(triangles);
```

Direkt dahinter wird der vorgegebene Vertex-Shader `simple.vert` und Fragment-Shader `simple.frag` geladen:

```
109     // Load our simple shader:
110     Shader shader(
111                 CMAKE_SOURCE_DIR + "/shader/simple.vert",
112                 CMAKE_SOURCE_DIR + "/shader/simple.frag"
113     );
```

Weiter darunter innerhalb der Main-Loop, die in jedem Frame des OpenGL-Programms aufgerufen wird, werden sowohl dieser Shader gebunden (d.h. für nachfolgendes Rendern aktiviert) und die render-Funktion des Meshes aufgerufen:

```
while (!glfwWindowShouldClose(window))
{
    [...]
    // Bind our shader:
    shader.bind();

    // Draw our triangle:
    triangleMesh.render();
    [...]
}
```

Deine Aufgabe ist es, die `Mesh-Klasse` zu vervollständigen – insbesondere das Hochladen der Geometrie-Daten vom Arbeitsspeicher auf den Grafikspeicher. Dazu musst Du im Konstruktor mithilfe der gegebenen Dreiecks-Informationen ein `VertexArrayObject` und drei `VertexBufferObject` erstellen und die IDs der entsprechenden Buffer in den vorgegeben Attributen `vao`, `vbo_position`, `vbo_normal` und `vbo_color` speichern. Im ersten `VertexBufferObject` sollen die Positionen (Index 0), im Zweiten die Normalen (Index 1) und im Dritten die Farbe aller Vertices (Index 2) gespeichert sein. Achte darauf, dass deine Mesh-Klasse auch mit mehr als ein Dreieck funktioniert!

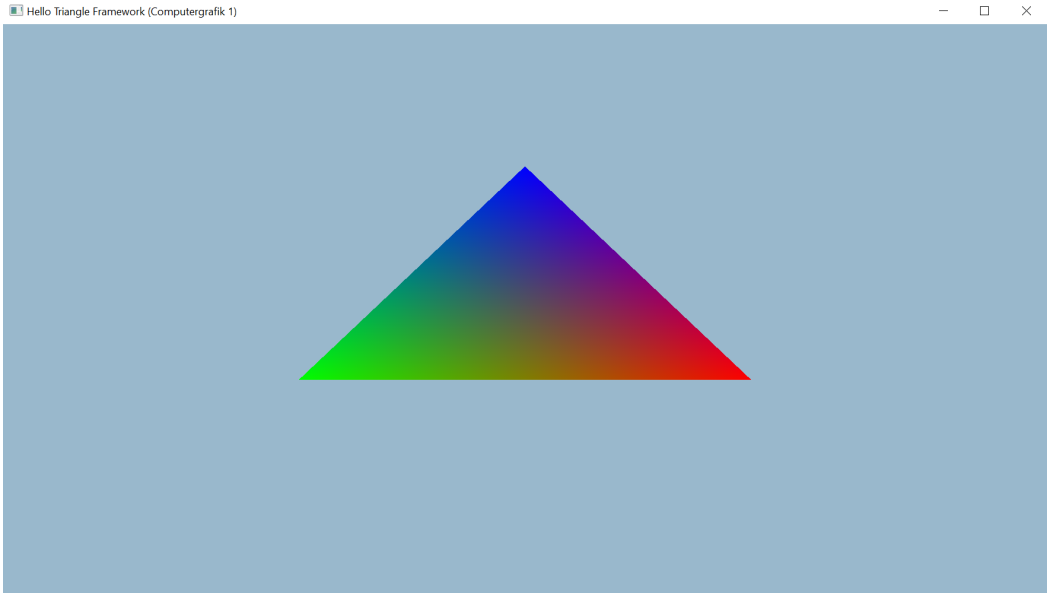

Abbildung 1: Das Fenster nach korrekter Implementierung der Mesh-Klasse

Hinweis: Zur Bearbeitung der Aufgabenstellung musst Du die OpenGL-Funktionen `glGenBuffers`, `glGenVertexArrays`, `glBindVertexArray`, `glBindBuffer`, `glBufferData`, `glVertexAttribPointer` *und* `glEnableVertexAttribArray` *– ggf. auch mehrfach – benutzen. Diese sind hier sortiert nach ihrem erstmaligen Auftreten in der Musterlösung, müssen allerdings nicht zwangsläufig in dieser Reihenfolge ausgeführt werden.*

Aufgabe 2 (Algorithmus: Schnitt-Gerade-Dreieck, *4 Punkte*)

Gegeben seien drei verschiedene Punkte $P_1, P_2, P_3 \in \mathbb{R}^3$ die ein Dreieck im $\mathbb{R}^3$ bilden, sowie eine Gerade $g : X = S + \lambda \mathbf{r}$. Skizziere einen Algorithmus, der feststellt, ob das Dreieck von der Gerade geschnitten wird und bei einer positive Antwort den Schnittpunkt ausgibt.