Institut für Visualisierung
und Interaktive Systeme

Prof. Dieter Schmalstieg Dr. Guido Reina

8. Übungsblatt zur Vorlesung Computergraphik im WS 2025/26

Besprechung am Montag, Dienstag und Mittwoch, 19.01., 20.01. & 21.01.2026

Aufgabe 1 *Transformationen, Geometrie, Instancing und Billboards in OpenGL [5 Votierpunkte]*

In dieser Aufgabe wird das Rendern einer simplen Szene mit OpenGL behandelt. Wenn Sie das Projekt öffnen und ausführen, öffnet sich ein Fenster, in dem aber noch nichts gezeichnet wird.

1. **Camera** [1 Votierpunkt]
   In `main.cpp` finden Sie die Funktion `UpdateScene`. Setzen Sie darin die *View*- und die *Projection*-Matrix. Dabei ist `g_cam_pos` die Position der Kamera und `cam_dir` die Blickrichtung, wobei die y-Achse nach oben zeigen soll. Für die Projektion soll das richtige Seitenverhältnis zwischen der Fensterbreite `g_window_width` und der Fensterhöhe `g_window_height` verwendet werden. Wählen Sie außerdem passende Werte für das *Field of View* sowie die Entfernungen zur *Near*- und *Far-Plane* für eine Szenegröße von ungefähr 100m.

   Hinweis: Anstatt die Matrizen von Hand zu berechnen, empfiehlt es sich, die entsprechenden Funktionen der `glm`-Bibliothek zu verwenden.

2. **OpenGL Vertex Data mit Direct State Access** [1 Votierpunkt]
   In der Funktion `CreateRenderBatch` werden für übergebene Index- und Vertexdaten eines Szenenobjekts die entsprechenden OpenGL-Objekte initialisiert. Für Indexdaten ist dies bereits implementiert, jedoch ohne Direct State Access (DSA) Funktionen zu verwenden. Nutzen Sie im Folgenden DSA Funktionen um die Funktion `CreateRenderBatch` zu vervollständigen. Ergänzen Sie die Initialisierung eines *Vertex Buffers* und *Vertex Array Objects* (mit zwei Attributen) für die übergebenen Index- und Vertexdaten.

   Details zu den OpenGL API-Funktionen können Sie in den OpenGL Reference Pages nachschlagen. Für den Umgang mit DSA Funktionen, insbesondere dem Vergleich zur alternativen zustandsorientierten Vorgehensweise, können Sie sich unter anderem hier informieren: Khronos Wiki DSA und Guide-to-Modern-OpenGL-Functions.

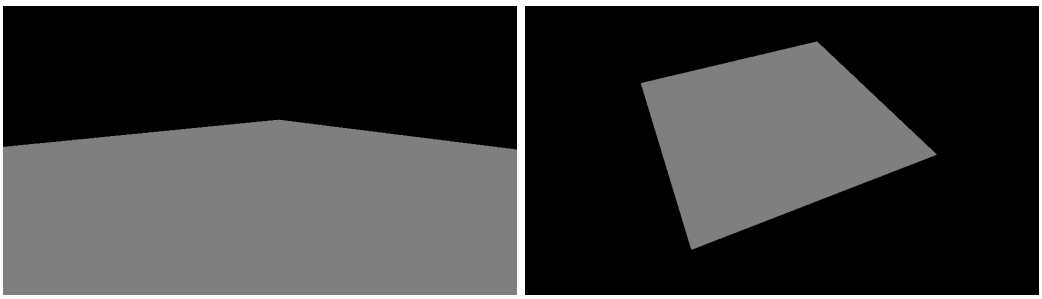

Abbildung 1: Ausgabe nach korrekter Lösung von Teilaufgabe 1.1. und 1.2.

Hinweis: Mit den Tasten *W, A, S, D, Shift* und *Space* kann man sich im Raum bewegen. Die Pfeiltasten erlauben eine Änderung der Blickrichtung. Alternativ kann auch ein Gamepad verwendet werden (z.B. ein Xbox One oder PS4-Controller).

3. **Tree Geometry** [1 Votierpunkt]
   In der Funktion `InitScene` befindet sich der Code, der die Geometrie für den Boden erstellt. Fügen Sie Code hinzu, der analog die Geometrie eines zweidimensionalen Baumes in der xy-Ebene erstellt und die zugehörige globale Variable mit `CreateRenderBatch` initialisiert (vgl. Abbildung 2). Eine passende Größe für die Baumgeometrie ist eine Höhe von 3m bis 4m. Die exakte Farbe der Vertices bleibt Ihnen überlassen.

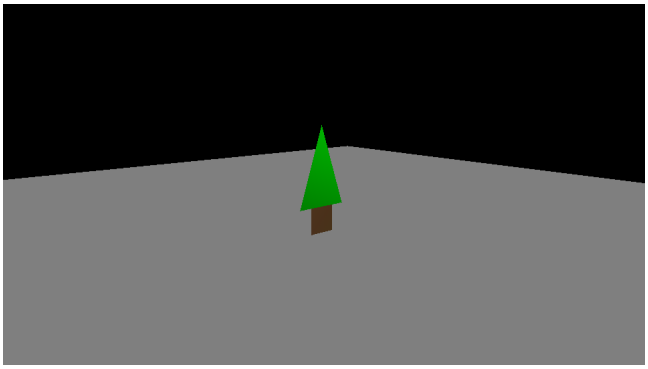

Abbildung 2: Ausgabe nach korrekter Lösung von Teilaufgabe 1.3.

4. **Instancing** [1 Votierpunkt]
   Tatsächlich werden vom Hauptprogramm 192 Instanzen Ihres Baumes gezeichnet. Dabei wird der Vertex-Shader `vert_trees.glsl` verwendet. An diesen wird für jeden der Bäume eine eigene *Model*-Matrix übergeben. Da diese nicht verwendet werden, erscheinen alle Bäume an der selben Stelle. Modifizieren Sie den Shader, sodass die *Model*-Matrizen korrekt verwendet werden (vgl. Abbildung 3).

   Hinweis: Mit `gl_InstanceID` können Sie im Vertex-Shader auf den Index der Instanz des Baums zugreifen. Dieser hat einen ganzzahligen Wert zwischen 0 und 191.

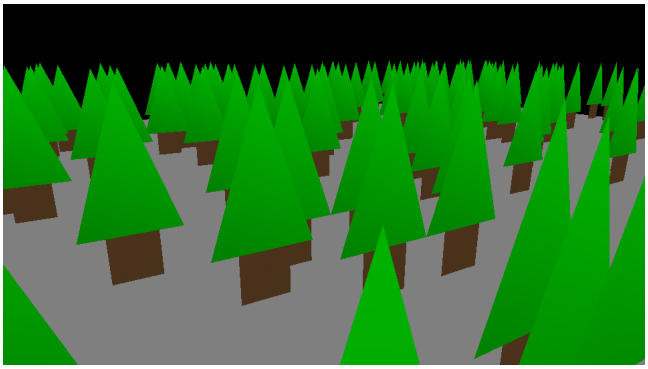

Abbildung 3: Ausgabe nach korrekter Lösung von Teilaufgabe 1.4.

5. **Billboard** [1 Votierpunkt]
   Implementieren Sie ebenfalls in `vert_trees.glsl` den sogenannten *Billboard*-Effekt für die Bäume. Bei diesem Effekt drehen sich 2D-Objekte immer zum Betrachter, egal aus welcher

Richtung dieser die Szene betrachtet. Modifizieren Sie dazu die Rotationskomponente der `model_view`-Matrix, sodass die x-Achse bei dieser Transformation unverändert bleibt.

Hinweis: Da der Baum in der xy-Ebene liegt (mit $z = 0$), kann die Transformation der z-Achse ignoriert werden.

Aufgabe 2 *Texturen und Imposter [5 Votierpunkte (1 Punkt pro Teilaufgabe)]*

Ein weiterer möglicher Einsatzzweck für Billboards ist die Darstellung entfernter Objekte, für die sich eine detaillierte Geometrie weder lohnt noch im Polygon-Budget der Szene Platz findet. Da die Blickrichtung zu ausreichend weit entfernten Objekt zudem auch bei Bewegung nur begrenzt variiert, fallen Billboards oftmals kaum auf. Die bisher verwendete Geometrie der Bäume in der Szene ist allerdings auch auf weite Distanz nur eingeschränkt glaubwürdig. Deswegen sollen nun die Bäume als texturierte Quads (sog. *Imposter*) gerendert werden. Dabei soll der Alpha-Kanal der Textur verwendet werden, um unnötige Fragmente des Quads zu verwerfen (vgl. Vorlesungsfolien OpenGL S. 155).

1. Erzeugen Sie in der Funktion `InitScene` analog zu Aufgabe 1 die Geometrie für ein Quad aus zwei Dreiecken. Das Quad soll im Bereich $[-1.5,1.5] \times [0,3] \times [0,0]$ liegen, mit Texturkoordinaten im Bereich $[0,1] \times [1,0]$. Verwenden Sie die ersten beiden Einträge der Vertex-Farbe um die Texturkoordinaten zu speichern.

2. Ändern Sie den Wert der globalen Variable `g_render_sprites` zu `true`, um das Rendern der Baum-Sprites zu aktivieren (siehe Funktion `RenderFrame`). Kopieren Sie außerdem ihre Lösung von Aufgabe 1.4 und 1.5 in den Vertex-Shader `vert_tree_imposters.glsl`. Anschließend sollten Sie die Quad-Imposter mit dem farblichen Verlauf der Texturkoordianten sehen.

3. In der Funktion `CreateSpriteTexture` wird bereits eine Bilddatei geladen und die notwendigen Informationen zur Verwendung als Textur angegeben. Vervollständigen Sie die Funktion mit der Erzeugung eines OpenGL Textur-Objektes, in das die Bilddaten geladen werden. Nutzen Sie erneut die DSA-Varianten der API-Calls. Die Kommentare im Code liefern Ihnen die Details zu den notwendigen OpenGL API-Calls. Schauen Sie sich außerdem in der Funktion `RenderFrame` an, wie die Textur bereits an den Shader weitergegeben wird, um einen vollständigen Eindruck der notwendigen API-Calls für Texturen zu erhalten.

4. Um die Textur auf den Imposter-Quads anzuzeigen, müssen Sie nun noch im Fragment-Shader `frag_tree_imposters.glsl` die Textur mit den übergebenen Texturkoordinaten (`in vec2 uv;`) auslesen und als Ausgabefarbe verwenden. Wenn der Alpha-Kanal der Textur unterhalb eines gewissen Grenzwertes liegt, verwerfen Sie das Fragment mit dem `discard` Befehl.

5. Bisher haben alle Instanzen des Baums die gleiche Größe. Berechnen Sie im Vertex-Shader `vert_tree_imposters.glsl` einen zufälligen Skalierungsfaktor, mit dem die Position der Eingabe-Vertices multipliziert wird.

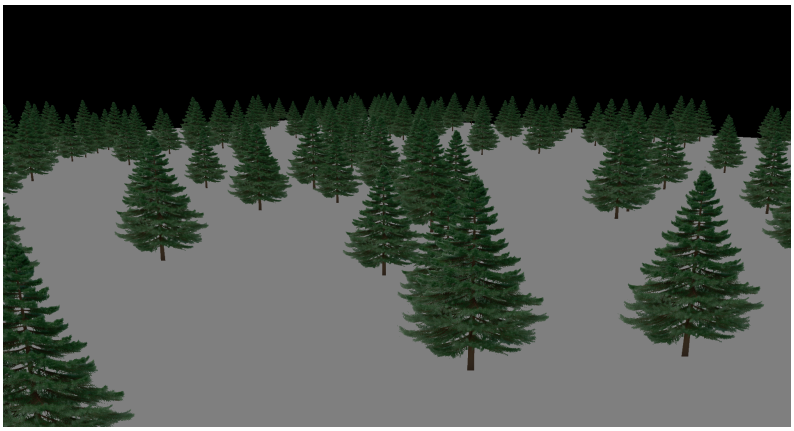

Abbildung 4: Ausgabe nach korrekter Lösung von Aufgabe 2.5.