Project Computer Graphics: OpenGl

Wald Habets & Ruben Ballet

May 20, 2018

1 Inleiding

Dit bestand bevat het verslag van het project OpenGL, door Wald Habets en Ruben Ballet, voor het vak Computer Graphics. Dit verslag bespreekt de gemaakte keuzen betreffende datastructuren, file formaat en ontwerp van het project.

2 Vereisten

Deze sectie bespreekt de implementatie van de vereisten samen met de hierbij nodigge toelichting.

Weergave van een 3D omgeving Als 3D omgeving de eerste level van het spel Mario64, Bob-omb Battlefield, gekozen.

Display Lists Alle objecten worden in de wereld geïnstantiëerd met behulp van display lists, ook de objecten die maar één maal geïnstantiëerd moeten worden. Er is voor deze aanpak gekozen omwille van de manier waarop objecten worden ingeladen in het programma. Door display lists te gebruiken voor alle objecten, kunnen deze objecten worden ingeladen op een uniforme manier. Bij het inladen hoeven we ons dus geen zorgen te maken of een object slechts eenmalig of meerdere keren voorkomt.

Het aanmaken van een display list gebeurt op dezelfde manier als besproken in de cursus (via glNewList, glBegin(GL_TRIANGLES) en glCallList). De elementen die voor elk object worden opgenomen in de display list zijn de vertices met voor elke vertex de bijhorende normaal en texture-coördinaat. Transformaties worden niet opgenomen in de display list, zodat op deze manier elke instantie van een object zijn eigen positie, grootte en rotatie kan hebben.

File Formaat & Hiërarchische relaties Alle objecten zijn .obj-bestanden die worden ingeladen met behulp van de Assimp-library. Welke objecten moeten worden ingeladen, hun textures, positie, rotatie, schalering en onderlinge relatie wordt beschreven in een .json-bestand. Bij het opstarten van het programma wordt dit bestand ingelezen en op basis hiervan wordt de wereld gegenereerd.

Er is gekozen om een bestand te gebruiken dan de volledige wereld beschrijft omdat dit ons toestaat de wereld makkelijk aan te passen zonder dat het nodig is om het programma opniew te compileren.

Dit json-bestand houdt 4 lijsten bij:

- Een lijst van models: dit zijn alle .obj-bestanden die moeten worden geladen. (in het programma vertaald dit zich naar een display list).
- Een lijst van textures: dit zijn alle texture-bestanden die moeten worden geladen.
- Een lijst van groupen: dit is een verzameling entities. Elke group heeft een positie, rotatie en grootte.
- Een lijst van entities: dit zijn de objecten in de wereld. Elke entity heeft een model ID (de display list), een texture ID, een positie, rotatie en grootte en eventueel een group ID. Indien de entity tot een group behoort beschrijven positie, rotatie en grootte hun respectievelijke attributen binnen de group.

Voor de relatie tussen de klassen verwijzen we naar *Afbeelding 1*.

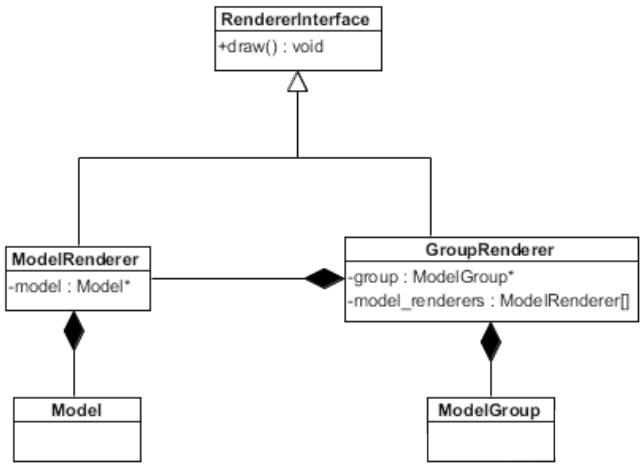

Figure 1: Afbeelding 1

Textures Aan alle objecten kan een texture toegekend worden. Het binden van een texture gebeurt niet in de display list, waardoor een object dat meermaals geïnstantiëerd wordt verschillende textures kan hebben voor zijn verschillende instanties. Het is echter wel zo dat elke instantie ten hoogste één texture kan hebben.

Het aanmaken en binden van textures gebeurt op dezelfde manier als besproken in de cursus (via glGenTextures, glBindTexture, etc). Voor het inladen van de afbeelding is voor de SOIL-library gekozen.

Visualisaties Het programma ondersteunt de vereiste visualisaties: nl. flat-shaded, smooth-shaded, wire-frame (en non-wire-frame). De visualisatie kan worden aangepast via het menu (esc > settings).

Navigatie Het programma ondersteunt navigatie via de muis en het toetsenbord. De sneltoesten kunnen worden geraadpleegd via het menu. (esc > key-bindings).

De camera bestaad uit 3 klassen: één die de positie en rotatie omvat, één die de input van het toetsenbord afhandelt en één die de input van de muis afhandelt.

De camera zelf houdt 2 punten en 4 vectoren bij, waarop alle manipulaties gebeuren. Deze 2 punten zijn de positie en het punt waar naar gekeken wordt. De 4 vectoren zijn:

- een forward-vector die altijd naar voor is gericht (t.o.v. de camera, evenwijdig met xz)
- een right-vector die altijd naar rechts is gericht (t.o.v. de camera, evenwijdig met xz)
- een up-vector die altijd naar boven is gericht (evenwijdig met y)
- een looks-at-vector die de kijkhoek beschrijft (boven onder)

Het gebruik van deze 4 vectoren staat ons toe om alle mogelijke bewegingen op een snelle en uniforme manier te berekenen via matrix transformaties. Elke vector heeft namelijk zijn eigen translatie-, rotatie- en schalerings-functies. Streven van de camera gebeurt namelijk volgens de forward-, right- en up-vectoren en roteren gebeurt door de looks-at-vector te roteren rond deze vectoren.

Lichtbronnen Het programma bevat 2 lichtbronnen, één vast in de wereld en één gemonteerd op de camera. De gemonteerde lichtbron wordt beschreven door de klasse *Headlamp*. Ook de lichtbronnen volgen het model van de cursus (glEnable, glDisable en glLightfv).

Picking Picking doen we op de manier beschreven in de cursus. We maken een viewport op de positie van de muis. Dan tekenen we alle objecten uit en koppelen we een id aan elk object. De id's van de objecten die binnen de viewport vallen worden in een buffer geladen. Vervolgens kunnen we doorheen de buffer gaan en kijken welk object het laatst is getekend, dus aan de voorgrond licht. Wanneer we het model hebben dat geselecteerd is, wordt er een boolean op true gezet. Het behavior van dat model kan dan kijken of het model is gehit en dan zijn behavior aanpassen.