Flying Reeves

Dokumentation

Kurzbeschreibung der Implementierung

Flying Reeves ist ein schnelles Geschicklichkeits-Spiel, bei dem der Spieler sind mit seinem Raumschiff nach vorne bewegt und dabei weder von der Straße abkommen noch auf ein Hindernis treffen darf. Die Strecke wird automatisch generiert und ist unendlich lange, dabei kann zwischen einer gleichbleibenden Strecke und einer auf Basis eines random Seeds erzeugten Strecke unterschieden werden. Während des Spiels kann der Spieler nur nach links und rechts lenken, das Fahrzeug beschleunigt automatisch. Auf der Strecke befinden sich Hindernisse und in größeren abständen Portale. Währen der Spieler durch die Portale fahren muss, um weiter zu kommen, ist es wichtig den Hindernissen auszuweichen, denn trifft der Speier auf ein Hindernis oder fällt von der Strecke, ist das Spiel vorbei.

Zur Implementierung: Das Herz des Spiels ist die Strecke, welche von einem TrackHandler verwaltet wird. Dieser generiert zu beginn des Spiels eine bestimmte Anzahl an verschiedenen Streckenstücken und hängt diese dann aneinander. Ist der Spieler am Ende eines Streckenstücks angekommen, kann dieses weiter vorne wiederverwendet werden. Durch eine hohe Anzahl an verschiedenen Streckenstücken und das zufällige Aneinanderhängen, entstehen immer wieder neue Strecken und Erlebnisse. Auch für Hindernisse und Portale ist der TrackHandler verantwortlich, er kümmert sich darum, in bestimmten Abständen immer wieder ein Hindernis oder Portal zu platzieren. Die Streckenstücke bestehen aus 3 dynamisch erzeugten 3D-Objekten, welche passenden aneinandergelegt werden. In der Mitte befindet sich die dunkle Fahrbahn und an beiden Seiten die etwas helleren, leicht spiegelnden „Flügel“ welche verhindern sollen, dass Raumschiffe zu schnell von der Strecke abkommen. Die Streckenstücke besitzen auch einen auf denselben Vertices gebauten 3D-Körper für die Collision-Detection. Die Hindernisse bestehen aus einfachen Quadern, welche auf der Strecke platziert werden, einige stehen still andere bewegen sich ein wenig. Die Portale, welche man durchquert sind, Sub-Dividend Planes, auf welche eine Vertex-Shader-Animation angewandt wird. Der Portal-Effekt entsteht durch die Kombination der Animation mit dem Environment Mapping. Der Rest der Strecke wird mit Hilfe eines Directional-Lights und zwei gerichteten Point-Lights welche die Scheinwerfer des Fahrzeugs darstellen, in einem physical based shader beleuchtet.

Der Spieler fährt ein Raumschiff, welches darauf angewiesen ist auf der Strecke zu bleiben. Künstliche erzeugt Schwerkraft, welche den Spieler auf die Strecke drückt, sorgt dafür, dass der Spieler nicht vom Kurs abweicht, aber auch dafür, dass sich das Raumschiff außerhalb der Strecke nicht steuern lässt. Die Schwerkraft passt sich dem Verlauf der Strecke im 3-dimensionalen Raum an. Das Modell des Spielers wird über eine Library geladen und auch über den physical based shader beleuchtet. Für Collision detection, die Bewegung des Spielers und die Schwerkraft wird Physx verwendet.

Die Kamera folgt dem Fahrzeugt, der Spieler hat dabei die Chance zwischen verschiedenen perspektiven zu wechseln (hinter-, ober-, vor-, …, dem Fahrzeug). Für eine angenehmere Kamerasteuerung bewegt sich die Kamera ein bisschen langsamer als der Spieler. Auf einem Heads Up Display werden dem Spieler dabei auch Geschwindigkeit und zurückgelegte Distanz angezeigt.

Features

- Automatisch generierte Strecken
- Mehrere Kameraperspektiven
- Heads-Up-Display
- Collision Detection
- Komplexes Model
- Environment Mapping
- Physically Based Rendering
- Vertex Shader Animations

Zusätzlich Libraries

Bibliotheken (Libraries)

- Assimp: eine portable Open-Source-Library, welche zum Importieren verschiedener 3D-Modellformate dient
  https://learnopengl.com/Model-Loading/Assimp
- GLEW: bietet effiziente Laufzeitmechanismen, um festzustellen, welche OpenGL-Erweiterungen auf der Zielplattform unterstützt werden.
  (Aus ECG)
- GLFW: bietet eine einfache API zum Erstellen von Fenstern, Kontexten und Oberflächen, zum Empfangen von Eingaben und Ereignissen.
  (Aus ECG)
- PhysX: Physics engine für collision detection.
  https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Index.html
- Freetype: Library zum Erstellen von Schrift (Glyphen).
  https://www.freetype.org/

Effekte

Environment Mapping:
Wir haben eine Cubemap erstellt, damit es so ausschaut als würde man wirklich durch das Weltall fliegen. Reflektionen der Sterne kann man auf den Seitenrändern der Strecke sehen und natürlich auch auf den Portalen, wo wir eine Mischung aus Reflektionen und Brechungen verwenden.
Hier haben wir auf learnOpenGl ein gutes Tutorial gefunden.
https://learnopengl.com/Advanced-OpenGL/Cubemaps

Physically Based Rendering:
Das Physically Based Rendering wird für alle Bereiche des Games, außer für die Portale verwendet und sorgt für ein realistischere Beleuchtung. Für das Physically Based Rendering haven wir uns zu erst auf die Theorie konzentriert, um das Konzept dahinter zu verstehen. Bei der Implementierung haben wir uns dann an das Tutorial von learnOpenGl gehalten.
https://www.youtube.com/watch?v=j-A0mwsJRmk&feature=youtu.be
https://de45xmedrsdbp.cloudfront.net/Resources/files/2013SiggraphPresentationsNotes-26915738.pdf
https://learnopengl.com/PBR/Theory

Vertex Shader Animation:
Die Vertex Shader Animation wird dazu benutzt die Wellen-Animation für die Portale zu machen. Implementiert wurde die Animation nach einer genaueren Erklärung vom Tutor beim ersten Abgabegespräch auf eigene Faust.