# Alpin Golf

Anna Berezhinskaya (01233066)
Ole Siemers (01529727)

## Idee

Du bist ein Golfball auf einem Golf Feld in Alpen. Was kann man vom Leben denn noch erwarten? Hey, nicht zu viel von den Bergen abgelenkt werden! Du musst zum Loch und das Erfolg deines Teams hängt jetzt von dir ab! Schaffst du es bis zum Ziel zu rollen? Hm, zuerst musst du das Loch noch finden... Lass dir aber nicht zu viel Zeit, denn die ist begrenzt! Falls du noch Boost brauchst, hol dir einen von Checkpoints!

Viel Erfolg.. und... Let's roll!

## Steuerung

- Bewege die Maus um die Bewegungsrichtung einzustellen.
- Klicke die linke Maustaste um in die Blickrichtung zu beschleunigen.
- Nutze das Gefälle um Geschwindigkeit aufzubauen
- und sammle Boost bei den Checkpoints.
- Erreiche das Ziel bevor die Zeit um ist.
- Drücke Backspace oder rechte Maustaste für Neustart.
- Mit dem Mausrad kann die Kameraentfernung angepasst werden.

## Eingebaute Effekte:

### *- Advanced Physics, Collision Erkennung mit Heightmap und Checkpoints.*

***Beschreibung:*** Das Physic Engine ermöglicht die realistische Bewegung des Balls durch die Gegend. Die Kollisionen mit Heightmap und anderen Objekten werden erkannt. Bei der Kollision mit den Checkpoínt wird diese platzen und der Ball bekommt mehr Boost um weiter zu rollen.

***Implementierung:*** Die Bullet-Physics-Engine wird für die Physiksimulation verwendet. Die zwei Wrapper-Klassen “RigidBody” und “DynamicsWorld” abstrahieren den Zugriff auf die Library. Jedes Objekt vom Typ RigidBody kann mit jedem Anderen in der DynamicsWorld kollidieren. Die Heightmap wird aus einem Graustufenbild geladen.

***Literatur:*** https://pybullet.org/Bullet/BulletFull/

## *- Normal mapping*

***(De-)Aktivierungstaste:*** Wenn man beim Spielen die Taste "8" druckt, kann man die Normale als farbiges Kontrast.

***Implementierung:*** Im Vertex Buffer Object werden zusätzlich zu den Normalvektoren noch die Tangenten und Bitangenten gespeichert. Diese Information wird beim Generieren der Heightmap bzw. von Assimp beim Modelimport berechnet. Mit dieser Information und einer zusätzlichen Textur, werden die kleinen Unebenheiten auf dem Golfball und dem Gelände gezeichnet.

***Literatur:***

*https://learnopengl.com/Advanced-Lighting/Normal-Mapping*

## *- Specular- & Environment mapping*

***Beschreibung:*** Die Oberfläche der Objekte sind haben nicht uniforme Materialeigenschaften, sondern können zusätzlich mit einer Textur gesteuert werden. Außerdem reflektieren der Ball und die Checkpoints die statische Skybox.

***Implementierung:*** Die Texturen werden dem Phong-Shader übergeben, die Speculare-Komponenten mit dem gesampelten Werten multipliziert und ein Reflexionsanteil addiert.

Mit der Taste 5 wird der Reflexionsanteil auf 100% gestellt.

## *- Sprite-HUD*

***Beschreibung:*** HUDs dienen bei unserem Spiel als User Interface. Die verbliebene Zeit und Boost mit Unterschriften werden angezeigt, weiters wird der Spieler am Anfang und am Ende des Spiels mit HUDs über den Status informiert " Start", "you lost", "you won".

***Implementierung:*** Wir haben die einfache Quads erstellt, die auf den richtigen Positionen platziert, skaliert und mit den Texturen bedeckt. Diese Texturen sind einfache Bilder mit entsprechendem Text (schwarz auf weiss). Weiters in unserem Code wird bei den Bildern alpha blending durchgeführt.

## *- Konfigurationsdatei:*

In der Konfigurationsdatei können Parameter, wie Helligkeit und Kontrast des Bildes oder ob das Spiel im Vollbild startet, eingestellt werden

## *- Model-Import*

***Beschreibung:*** ASSIMP Model importer ist eingebaut. Diese Eigenschaft hat uns die Nutzung der Modelle aus Internet bzw. im Blender erstellten Modellen ermöglicht.

***Implementierung:*** Um das Import von Modellen zu ermöglichen, haben wir eine Klasse "Modell" erstellt. Diese Klasse hat Funktionen, die das .obj File lesen können, die entsprechende .mtl und Texture Files finden und das Model von ASSIMP Format für unseres Programm "übersetzt".

***Literatur:***

*http://assimp.sourceforge.net/lib_html/data.html*

*https://learnopengl.com/Model-Loading/Assimp*

*https://www.udemy.com/course/graphics-with-modern-opengl/*

## *- Bloom and Blur*

***Beschreibung:*** Die Bereiche, in denen "brightness" eine gewisse Konstante überschreitet (0.65) werden in eine Texture gespeichert, mit hilfe des Gaussian curve werden die Grenzen von den Bereichen breiter und gleichzeitig unscharfer. Das erzeugt “glow” Effekt.

***(De-)Aktivierungs Taste:*** Wird die Taste "9" gedrückt, so ist das Effekt ausgeschaltet. Wird die Taste "0" gedrückt, so sieht man nur die Textur, die für Blur benutzt wird angezeichnet.

***Implementierung:*** Für die Bloom Teil wird ein Framebuffer mit zwei color Buffers benutzt. Mit Hilfe des Multiple Render Targets werden im ersten Schritt zu einem color buffer das Anfangsbild gezeichnet und ins andere das Bild, in welchen nur die helle Bereiche über gewisse brightness value gezeichnet werden. Die helle Bereiche werden in nächsten Schritt zu “pingpong” Buffers geschickt. Das sind die Framebuffer, die das rendern in color buffer voneinander durchführen und dabei einen spezifischen Shader benutzen (mit gaussian blur). Das Bild wird vom einen in den anderen Buffer gerendert und wieder zurück gerendert. Als Resultat entsteht ein Blur Effect. Nun werden die beide Texturen (das normale Bild und das mit gauss blur) zusammengelegt.

***Literatur****:*

https://learnopengl.com/Advanced-Lighting/Bloom

## *- Shadow Maps (Lightmap using In-Game Calculation)*

***Beschreibung:*** Die Schatten werden gezeichnet

***Implementierung:*** Es wird ein framebuffer benutzt, der das Bild von der Perspektive der Lichtquelle rendert und dabei die depth values in einer Texture gespeichert. Das sind Werte im Bereich zwischen 0 und 1. Nun wird das Bild nochmal von camera Perspektive gerendert, wobei die erzeugte Shadow Map Texture zum Bestimmen der Schatten benutzt wird. Die depth Werte von shadow map werden dabei verglichen werden um zu sehen, was näher an der Lichtquelle ist und was von anderen Objekten verdeckt wird. Auf diese Weise werden die Schatten erkannt und gezeichnet.

***Literatur:***

https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

http://www.opengl-tutorial.org/ru/intermediate-tutorials/tutorial-16-shadow-mapping/

## *- CPU Particle System*

***Beschreibung:*** Wird der Ball das Finish erreichen, so wird eine Menge von Teilchen frei geworfen.

***(De-)Aktivierungstaste:*** Wenn man die Taste “6” gedrückt, werden jedenfalls die Teilchen von dem Ball freigesetzt.

***Implementierung:*** Bei uns sind die Teilchen kleine Quadrate, die wir mit einer einfacher Textur bedecken. Wir haben eine Klasse “Particle” erstellt, jedes Objekt dieser Klasse hat eine Lebensdauer, Geschwindigkeit, Position und “alive” boolean Indikator. Zuerst bei der Spiel Initialisierung wird ein Container aus “Particle” Objekte erstellt, wobei die alle Particles als “nicht alive” gekennzeichnet werden. Wird der Ball eingelocht, so wird die Funktion “spawnParticles” aufgerufen. Dabei werden die Teilchen als alive gesetzt und die Geschwindigkeit und Positionen übergeben. In jedem Render Durchgang wird die Funktion “renderParticles” aufgerufen, welche die lebendige (“alive” gesetzte Particles) zeichnet.

## *- Vertex shader animation*

***Beschreibung:*** Die Flagge beim Ziel ist animiert.

***Implementierung:*** Die Flagge wird mit einem getrennten shader gezeichnet. Dabei werden in Fragment shader die Positionen und Normale transformiert. Dafür benutzt man die ebene Wellengleichung.

***Literatur:*** David Wolff - OpenGL 4.0 Shading Language Cookbook

# Zusätzliche Bibliotheken

- Assimp
- Bullet Continuous Collision Detection and Physics Library
- stb Image