

SpeedBlasters

Kurzbeschreibung:	Ein Spiel von:	
Ein turbulentes Sci-Fi Rennspiel mit einer Menge an bösen Tricks, wie Laserkanonen.	Omar Ismail 01327702	Christoph Essler 01328166
Links:	GitHub	

Dies ist das in Abgabe 2 geforderte Dokumentations-Dokument.

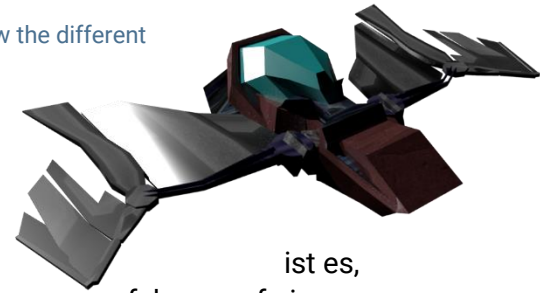
1 IMPLEMENTATION

Brief description of the implementation, in particular a short description of how the different aspects of the requirements (see above) were implemented.

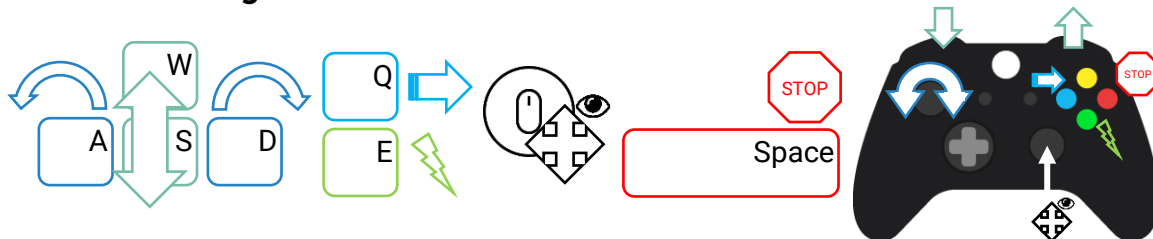
1.1 GAMEPLAY

1.1.1 Beschreibung

SpeedBlasters ist ein Rennspiel, das heißt die primäre Aufgabe ist es, die Strecke vor dem Gegner zu absolvieren. Es sind drei Runden zu fahren, auf einer Strecke mit vielen Kurven, und auch einer Stelle mit Über/Unterführung. Es ist aber nicht nur irgendein konventionelles Rennspiel, denn die beiden Spieler sind mit Laserwaffen ausgestattet. Kurzgefasst, kann der/die Spieler_in gewinnen, indem man drei Runden fährt, oder das andere Raumschiff abschießt bzw. deren Gesundheit auf ≤ 0 bringt.



1.1.2 Steuerung



Ein/Eine Spieler_in verwendet die Tastatur des PCs, der/die andere den Controller. Das Raumschiff auf der linken Seite des Bildschirms wird mit den Tasten WASD gesteuert. A und D verändern dabei den Gier¹-Winkel des Fahrzeugs. Die Leertaste ist eine Vollbremsung, die es einem ermöglicht die momentane Position festzuhalten (auch von der Schwerkraft unbeeinflusst). Q aktiviert den Boost, E den Laser. Das rechte Raumschiff lässt sich mit dem Controller bedienen – für die Controllersteuerung s.o.

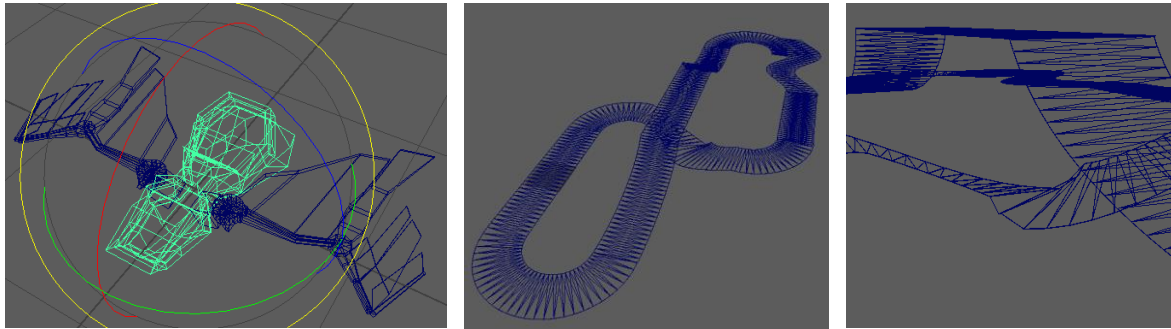
1.2 EFFEKTE

- Environment Mapping
- Spotlights
- Directional Light
- Shadow Mapping
- Normal Mapping

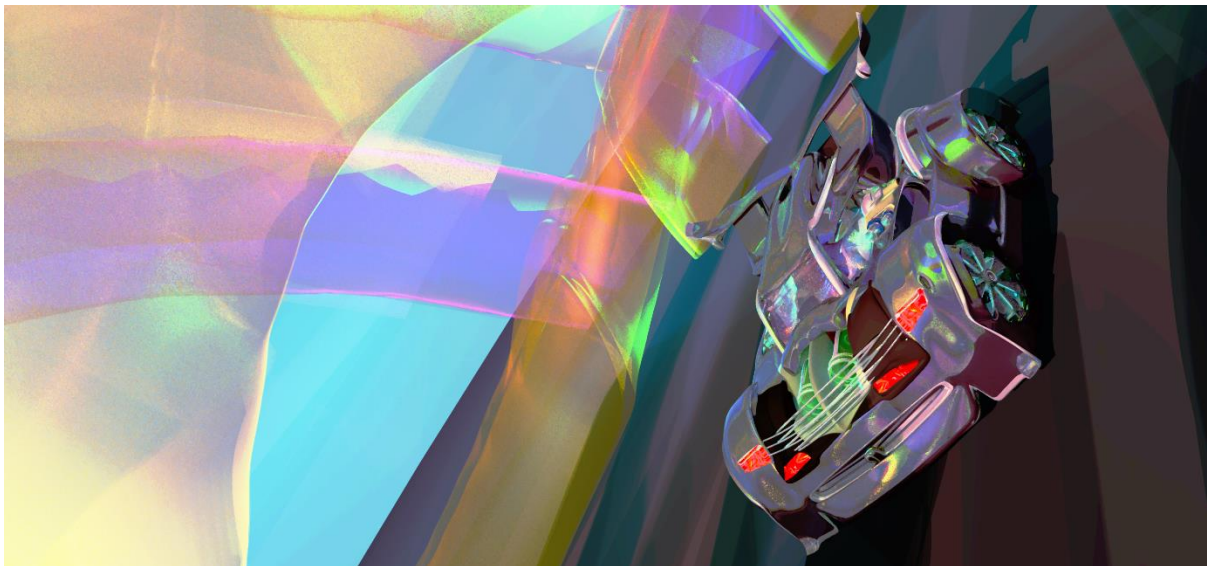
Effekte sind aus der Effekt-Liste gewählt.

¹ Gier bezieht sich auf Rotation um die Y-Achse.

1.3 KOMPLEXE OBJEKTE



Wir verwenden diverse Objekte in SpeedBlasters - alle davon wurden von uns in Maya modelliert. Manche von denen haben es aufgrund zu hoher Komplexität nicht ins Spiel geschafft (s.u.). Das Raumschiffmodell (links) besteht hauptsächlich aus stark veränderten Würfelobjekten, die mit der Extrude Operation verkompliziert wurden. Die Strecke (Mitte und Rechts) ist eine reine Triangle Mesh, die ebenfalls aus Würfeln gebaut wurde – die Kurven wurden mit Maya's Deform erzeugt.



Die Strecke enthält sowohl konkave als auch konvexe Elemente. Tatsächlich hat uns dieser Umstand einige Probleme bereitet, da die einfachste Methode Meshes in Bullet zu importieren und als Kollisionsobjekt zu verwenden eine Konvexe Hülle ist, welche alle konkaven Teile der Strecke relativ sinnlos macht, da die Kollision weit davor abgefangen wurde.

Die Modelle zum Rendern werden via Assimp importiert, und die Kollisionsabfrage passiert mittels Bullet. Beides sind Libraries, welche im entsprechenden Kapitel näher besprochen werden.

1.4 ANIMATION

Wir verwenden eine 3D Animation für beide Raumschiffe. Diese ist rein C++, ist also nicht importiert. Ursprünglich wollten wir Skeletal Animation aus Maya importieren, wir sind aber in zu viele Probleme mit Assimps .fbx, .dae, ... Importern gestoßen. Obwohl die zu importierende Animation bereits fertig ist, haben wir eben in C++ eine Animation "hard coden" müssen.

Bei dieser Animation handelt es sich um Flügelschlagen. Dafür wurde eine CompositePlayer Klasse erstellt, die vom früheren Player Objekt erbt. Das heißt, ein Spieler besteht aus dem Rumpf des Raumschiffs, welches in Player liegt, und den Flügeln, welche in CompositePlayer liegen. CompositePlayer speichert außerdem einen Winkel θ , um welchen die Flügel nach oben gedreht werden. Theta nimmt Werte zwischen -45° und 45° an.

1.5 VIEW-FRUSTUM-CULLING

Unser VFC funktioniert, ist aber auf Grund unserer geringen Objektanzahl fast irrelevant. Jedenfalls kann es mit der Taste **F8** ein und ausgeschaltet werden.

1.6 DEBUG KEY TOGGLES

Für die Features die wir implementiert haben, treffen die gewünschten Shortcuts zu. Das heißt:

- F1** - Help (if available)
- F2** - Frame Time on/off
- F3** - Wire Frame on/off
- F4** - Textur-Sampling-Quality: Nearest Neighbor/Bilinear
- F5** - Mip Mapping-Quality: Off/Nearest Neighbor/Linear
- F6** - Gamma
- F7** - Schatten
- F8** - Viewfrustum-Culling on/off
- F9** - Blending

Beim Umschalten dieser Toggles wird in die Konsole eine Nachricht geprintet. Den restlichen Konsolenoutput haben wir dementsprechend in dieser finalen Version minimiert, damit diese Nachrichten auch herausstechen.

2 FEATURES

“Features” of the game. (Dieser Teil ist zum Großteil aus unserer Dokumentation der letzten Abgabe übernommen)

- Kollisions-Detektion unter Verwendung von Bullet.
- Schwerkraft & Kräfteauswirkung unter Verwendung von Bullet.
- Maussteuerung
- Gute aber einfache Shader
- Shader-Loader
- .obj File Loader unter Verwendung von Assimp
- Diverse Lichtquellen
- Jede Menge Spaß und Action.
- Und inzwischen ist es ein fertiges Rennspiel!

„Bei SpeedBlasters geht es um eine der verrücktesten Rennfahrt-Serien des Jahres 2222 AD. Das Preisgeld von 1.375.000 Global Credits (©) macht das Rennen für alle äußerst attraktiv. Der Unterschied zu anderen ähnlichen Veranstaltungen ist allerdings riesig: ALLES ist erlaubt. Brutalität wird durch Blaster, Fallen, oder allein durch herkömmliche Unfälle gewährleistet. Die Teilnehmen müssen genug Skill, Glück, und ein genug großes Waffenarsenal haben um hier dem Tod zu entgehen, aber für diese Menge an Geld könnte es das wert sein.“

2.1 BELEUCHTUNG

Wir haben ein Directional Light und zwei Spotlights, die an den Fahrzeugen angebracht sind.

2.2 TEXTURIERUNG

Die Strecke, die Flügel der Fahrzeuge und die Skybox sind texturiert.

Das Environment Mapping sieht man auf den Fahrzeugen, das Normal Mapping an den Schrauben auf der Strecke.

2.3 EFFEKTE

2.3.1 Effektliste

Spotlights – Autos

Environment Mapping – Strecke

Shadow Mapping – Autos und Strecke

Normal Mapping – Strecke

2.3.2 Spotlights

Für die Spotlights haben wir das Tutorial <https://learnopengl.com/#!Lighting/Light-casters> verwendet.

2.3.3 Environment Mapping

Für das Environment haben wir das Tutorial <https://learnopengl.com/#!Advanced-OpenGL/Cubemaps> und das ECG Wiki verwendet (<https://lva.cg.tuwien.ac.at/ecg/wiki/doku.php?id=students:cube-mapping>).

2.3.4 Shadow Mapping

Für das Shadow Mapping haben wir das Tutorial <https://learnopengl.com/#!Advanced-Lighting/Shadows/Shadow-Mapping> verwendet.

2.3.5 Normal Mapping

Für das Normal Mapping haben wir das Tutorial <https://learnopengl.com/#!Advanced-Lighting/Normal-Mapping> verwendet.

3 WERKZEUGE

What additional libraries (e.g. for collision, object-loader, sound, ...) were used, including references (URL) (see restrictions)? What Tools have you used to create the Models (Maya, 3DS MAX, ...).

3.1 LIBRARIES

Wir verwenden ein paar Bibliotheken, um, wie man so schön sagt, das Rad nicht neu zu erfinden.

3.1.1 Windows API

Da die Abgabe allein auf Windows Rechnern laufen muss, ist die Verwendung dieser API keine Beschränkung. Wir verwenden sie für die Hintergrundmusik. Ein .wav File wurde

erstellt – dieses kann Windows nativ im Hintergrund unter der Verwendung von PlaySound() loopen.

Informationen, sowie Tutorials, lassen sich unter [https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx) finden.

3.1.2 Bullet Physics

Unsere verwendete Bullet Version ist bullet-2.82-r2704.

Bullet ist eine Library bzw. Umgebung für Physiksimulation. Es ist zudem recht zugänglich, mit gratis Plugins für wichtige Modellierungssoftware. Unter <http://bulletphysics.org/wordpress/> findet man die Bullet Homepage. Dokumentation ist bei Bullet unter <http://bulletphysics.org/Bullet/BulletFull/> zu finden, wobei einiges nur auf dieser Wiki hier http://bulletphysics.org/mediawiki-1.5.8/index.php/Main_Page erklärt wird.

3.1.3 Assimp

Wir verwenden Assimp 3.3.1, die zurzeit neueste Version, erhältlich unter <https://github.com/assimp/assimp/releases>.

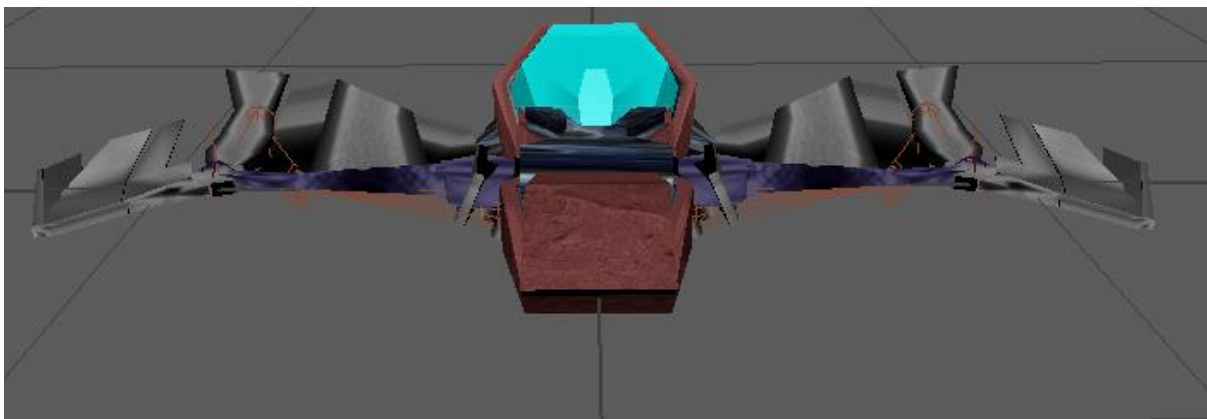
Assimp kann eine Vielzahl an 3D Modellen importieren. Für uns hat das simple .obj Format ausgereicht.

3.1.4 Weitere

- Glm² für GL Mathematik und GL typische Datenstrukturen
- GLEW³
- GLFW⁴(haben früher GLUT⁵verwendet)

4 ZUSÄTZLICHE GRAFIKEN

4.1 FRAMES DER URSPRÜNGLICHEN (SKELETAL) ANIMATION



² Glm.g-truc.net. (2017). OpenGL Mathematics. [online] Available at: <http://glm.g-truc.net/0.9.8/index.html> [Accessed 2 May 2017].

³ Grew.sourceforge.net. (2017). GLEW: The OpenGL Extension Wrangler Library. [online] Available at: <http://glew.sourceforge.net/> [Accessed 2 May 2017].

⁴ Glfw.org. (2017). GLFW - An OpenGL library. [online] Available at: <http://www.glfw.org/> [Accessed 2 May 2017].

⁵ Group, K. (2017). GLUT - The OpenGL Utility Toolkit. [online] Opengl.org. Available at: <https://www.opengl.org/resources/libraries/glut/> [Accessed 2 May 2017].

