

Call of Dirty – Dokumentation

In unserem Spiel geht es darum den ewigwährenden Kampf Student gegen dreckiges Geschirr in eine neue Runde zu führen. Gespielt wird das Ganze aus der Ego-Perspektive. Der Spieler wird gemeinsam mit Unmengen an Geschirr in die Arena geworfen, und ab da liegt es an ihm, in einem Kampf um Leben und Tod so viel dreckige Teller wie nur möglich von ihren Unreinheiten zu befreien. Zur Verfügung steht ihm dabei sein treues und zuverlässiges Gewehr, die ihm als einzige Waffe im Kampf zur Seite steht. Doch Munition ist rar gesät und so muss der Spieler durch die Arena manövrieren, um an den vier festgelegten Munitionsspawnpunkten (markiert durch blaue Zylinder am Boden) seine wieder Waffe nachzufüllen. „Aufgepasst“ heißt es hier nur vor dem Geschirr, das ähnlich wie die Munition, an vier festgelegten Orten (markiert durch rote Ringe am Boden) auf der Karte spawnt, wobei die Spawnrate des Geschirrs abhängig von der bereits gespielten Zeit bis zu einem gewissen Maximum stetig erhöht wird. Wenn die Gegner dem Spieler zu nahekommen (Moving Objects), wird diesem ein Leben abgezogen. Verliert der Spieler 3 Leben (Win/Lose Condition), ist das Spiel in Folge dessen zu Ende und der Score für die Runde wird angezeigt. Punkte werden nicht durchs reine Überleben vergeben, sondern hängt davon ab, wie viel Geschirr gereinigt wurde, daher ist eine offensive Spielweise notwendig. Zusätzlich aktiviert sich jedes Mal, nachdem der 50te Gegner getötet wurde, eine kurze Phase, indem die Feuerrate der Waffe erhöht wird, der Spieler an Geschwindigkeit gewinnt und beim Schießen keine Munition verliert. Diese sollte genutzt werden, um die Karte von allem derzeitigen Gegner zu bereinigen. (Playable)

Wichtige Informationen haben wir größtenteils von folgenden Seiten und Tutorials erhalten:

- *Learn OpenGL* [1]
- *OpenGL-Tutorial* [2]
- *Docs.GL* [3]
- *CPlusPlus* [4]

Implementierung und Features:

Bei der Implementierung wurde darauf geachtet, dass der Code in sinnvolle Klassen und Objekte unterteilt wurde, damit wir mit einem übersichtlich strukturierten und leicht erweiterbarem Code arbeiten können.

Bevor man das Spiel startet, kann man mit einer config file diverse Parameter anpassen. (z.B.: Sound, Helligkeit, Vollbild..) (Adjustable Parameters)

Die Engine kann mit Hilfe Assimp und stb_image.h von komplexe Objekte (3D Geometry) und deren Texturen (Textures) laden, die dann mit einem Phong-Shader, wie wir ihn in der Einführung in die Computergrafik kennengelernt haben, gerendert werden. Bei der Einbindung von Assimp sind wir sehr strikt einem Tutorial [5] gefolgt und dementsprechend gleich bzw. ähnlich sieht der Code auch in unserem Programm aus. Die Beleuchtung besteht aus einem Richtungslicht (Sonne) und vier Punktlichtern an den Positionen der Munitionsspawnpunkte, und setzt sich aus einem diffusen, einem Umgebungs- und einem (wenn auch sehr geringem) Glanzpunktanteil zusammen. Durch ein Array können sehr leicht mehrere Lichtquellen in die Szene eingefügt werden. Die Schatten, die im Spiel zu sehen sind, werden mit Shadow Maps und PCF mit dem Richtungslicht in der Szene für die Arena und die Gegner dynamisch berechnet. (Shadow maps with PCF) Zusätzlich hat auch der gewünschten Cel-Shading Effekt seinen Weg ins Spiel gefunden, der aufgrund der ohnehin schon sehr kantigen Optik des Spiels, nicht allzu viel Unterschied macht. (Cel-Shading)

Gespielt wird in Ego-Perspektive und typisch für diese Art von Spielen wird die Maus verwendet, um sich umzusehen und die Tastatur für die Bewegung durch den Level. (Intuitive Controls + Intuitive Camera)

Die Einbindung für Maus und Tastatur funktioniert über zwei Klassen, die für die Bearbeitung von Eingabebefehlen zuständig sind. Die Steuerung des Spieles kann durch diese sehr einfach umgesetzt werden, da unser Programm zu jeder Zeit in Arrays speichert, welche Tasten gerade gedrückt, losgelassen wurden oder gehalten werden.

die Steuerung sieht wie folgt aus:

- **W** – Vorwärts
- **S** – Rückwärts
- **A** – Links
- **D** – Rechts
- **Left Shift** – Sprinten
- **LMB** – Schießen
- **H** – HUD An/Aus
- **ESC** – Programm wird geschlossen

Um unser Spiel umsetzen zu können, war es notwendig, dass wir eine Physics Engine in unser Spiel einbauen, die unter anderem für die Bewegung der Objekte durch den Level zuständig ist, sowie für das bearbeiten diverser unterschiedlicher Kollisionen. Nach langem Überlegen, haben wir uns gegen das Einbinden von Nvidia's „PhysX“ entschieden, und unsere eigene Engine implementiert. (Physics Engine).

Für die Gegner und die Waffe kommen Partikeleffekte zum Einsatz, einerseits als Explosion nach einem Treffer des Geschirrs und andererseits beim Abfeuern der Waffe. (CPU Particle System)

Für die Dynamik im Bild, sorgt unter anderem die Waffe, die sich je nach Schrittempo auf und ab bewegt. Auch die Munition bei den Spawnpunkten schwebt auf und ab, und rotiert dabei um ihre eigene Achse und die Vorderseite der Gegner, ist immer in Richtung Spieler gerichtet. (Hierarchical Animation)

Für das Rendern der Munition wird zu guter Letzt noch ein Bloom-Effekt verwendet, der auch kurzzeitig aufs ganze Bild eingesetzt wird, nachdem der Spieler 50 Gegner eliminiert hat. (Bloom/Glow)

Um während dem Spiel Überblick über den derzeitigen Stand der Dinge zu haben, wird auch ein HUD mit allen notwendigen Informationen mit Hilfe der library „FreeType“ bereitgestellt. (Heads-Up Display)

Für die musikalische Untermalung kam die Library „irrKlang“ zum Einsatz.

Verwendete Libraries:

- GLFW (<https://www.glfw.org/>)
- GLEW (<http://glew.sourceforge.net/>)
- GLM (<https://glm.g-truc.net/0.9.9/index.html>)
- Assimp (<http://www.assimp.org/>)
- stb_image (https://github.com/nothings/stb/blob/master/stb_image.h)
- irrKlang (<https://www.ambiera.com/irrklang/>)
- FreeType (<https://www.freetype.org/>)

Quellenverzeichnis:

- [1] <https://learnopengl.com/Introduction>
- [2] <http://www.opengl-tutorial.org/>
- [3] <http://docs.gltf.io/>
- [4] <http://www.cplusplus.com/>
- [5] <https://learnopengl.com/Model-Loading/Assimp>