

Implementierung & Features

Unsere Implementierung baut einerseits auf dem aktuellen ECG-Framework auf und andererseits auf unserem unvollständigen CG-Projekt aus dem Vorjahr. Da das schon ein Jahr her ist, hoffen wir, möglichst alle wichtigen Bestandteile noch ausreichend benennen und beschreiben zu können.

Playable

Um aus der simplen Rendering Engine ein Spiel zu entwickeln, haben wir die Kamera des bestehenden Frameworks umgeschrieben, um die für unser Spiel nötige First-Person Ansicht zu bekommen. Des Weiteren haben wir grundlegende Spring-, Lauf- und Slide -Mechanismen implementiert. Außerdem haben wir 3D-Modelle für zwei Levels, einer Waffe und Feinde in Blender erzeugt und mit Assimp in unser Projekt geladen. Näheres dazu in den jeweiligen Unterpunkten.

3D Geometry

Wir benutzen eine erweiterte Geometry Klasse aus dem ECG-Framework, welche mit Assimp Code erweitert wurde, um unter anderem die bereits erwähnten komplexen Modelle – gespeichert als .dae Dateien - zu laden.

Wir benutzen eine Waffe, welche an den Viewport fixiert werden muss, um dem Spieler zu folgen. Dazu haben wir einen zweiten Rendering-Pass in der Game-Loop hinzugefügt, wobei dem Shader hier nur die Projektionsmatrix statt View-Projection-Matrix übergeben wird. Zusätzlich muss wegen der Beleuchtung die Lichtquelle in den Raum der Waffe zurücktransformiert werden.

Die komplexen Modelle (Waffe, Levels, Gegner) wurden mit Blender erzeugt.

Win/Loose Condition

Ziel des Spiels ist es grundsätzlich das Ende des Levels zu erreichen und dabei eine möglichst hohe Wertung zu erzielen. Diese wird durch die Faktoren getroffene Gegner (4,5 Punkte pro Gegner) und Schnelligkeit (als Multiplikator mit den verbleibenden Sekunden eines counters) berechnet.

Am Ende des ersten Levels wird das zweite Level geladen. Am Ende des zweiten Levels wird das Spiel beendet und der Score ausgegeben.

Verloren hat man, wenn der counter auf 0 ist, dann werden Bewegungen nicht mehr erkannt und das Level muss neu gestartet werden.

Intuitive Controls

Die verwendeten Tasten sind im Sinne zeitgemäßer Computerspiele und wurden mit Polling implementiert. Für Bewegungen wurde die berühmte **WASD**-Kombination benutzt. Zum Springen wird die **Leertaste** benutzt und zum Sliden **Links-Alt**. Sliden verkleinert die collision capsule des Spielers und erhöht zusätzlich kurzfristig die Geschwindigkeit. Man kann auch in der Luft sliden, um weiter zu springen. Springen kann man kontinuierlich, es gibt keinen delay, das ermöglicht (zusammen mit den Eigenschaften der Player Physx Capsule) eine relativ leichte Steuerung des Spielers. Hindernisse können so einfach gut erreicht werden. Außerdem gibt es zwei

Bewegungsgeschwindigkeiten, welche auf zweierlei Arten geändert werden können: **Links-Shift** ändert die Geschwindigkeit, solange die Taste gedrückt wird und **Caps-Lock** ändert den Modus pro Klick. Daher haben wir bei letzterer Variante auf Callback statt Polling gesetzt. Mit einem **Linksklick** der Maus kann geschossen werden.

Weitere nützliche Tasten:

Esc – beende das Programm

Enter – setze das Level nach Ablauf der Zeit zurück

F2 – Frametime/FPS-Anzeige togglen

F4 – Normalmapping togglen

Intuitive Camera

Wie bereits erwähnt haben wir die Kamera neu geschrieben, um eine First-Person View zu ermöglichen. Da wir das bereits vor einem Jahr gemacht haben, hoffe ich die Implementierung im Folgenden ausreichend genau zu beschreiben. Als Grundlage diene jedenfalls ein Tutorial [1].

GLM bietet mit der `lookAt` Funktion die Möglichkeit, die Kamera mit 3 Vektoren zu definieren. Der erste Vektor repräsentiert die Position der Kamera, der zweite zeigt in Richtung der Blickrichtung der Kamera und der Dritte zeigt von der Kamera nach oben. Bewegungen sind dadurch sehr einfach, da zu der Position der Kamera nur ein **skalierter Vektor** in die jeweilige Richtung der Bewegung addiert werden muss. Um an die jeweiligen Vektoren zu kommen, muss der Input der Maus miteinbezogen werden. Dazu wird in jedem Frame der Unterschied zur vorherigen x- und y-Position der Maus berechnet und zu den Euler-Winkeln yaw und pitch hinzuaddiert. Etwas Trigonometrie-Magie später haben wir dann unseren Front-Vektor, aus welchem wir mit 90°-Rotationen auf die Up- und Right-Vektoren schließen können, wodurch wir mit der Information der Position der Kamera auf die View-Matrix kommen.

Für die Framerate Independency wird die Geschwindigkeit der Bewegung mit der Delta-Time multipliziert. Rotationen der Kamera mit Pixelwerten sind per Definition Framerate Independent.

Mit dem zuvor erwähnten skalierten Vektor kann je nach Größe des Skalars die Bewegungsgeschwindigkeit gesteuert werden. Durch Änderungen der y-Koordinate der Position haben wir Springen und Sliden implementiert.

Textures

Bei den Texturen haben wir keine Änderungen im Vergleich zum ECG-Framework vorgenommen. Daher ist mipmapping und trilineares filtering aktiviert.

Moving Objects

In Level 1 haben wir zwei Gegner eingebaut, welche sich auf einer fest vorgegebenen Flugbahn bewegen. Dazu wird die Position der Gegner in jedem Frame um einen bestimmten Wert verschoben und ab Erreichen eines Thresholds die Richtung der Bewegung geändert. Für die Framerate

Independency wird der fixe Wert mit der Delta-Time multipliziert. Dasselbe gilt für die Bewegung der Waffe bei einem Schuss.

Adjustable Parameters

Wie im ECG-Framework können Parameter mit der settings.ini Datei angepasst werden. Wir haben dabei zwei verschiedene Werte für Auflösung, um schneller den Vollbild-Modus an- und auszuschalten. Die Refresh-Rate und Helligkeit können hier auch angepasst werden, die Namen sind selbsterklärend.

Physics Engine

Wir haben Physx 4.1 nach dem bereitgestellten Tutorial implementiert. Die Collision Detection wurde für den Character Controller bereits implementiert, welcher der Kamera folgt. Dadurch kann nicht mehr durch Wände gegangen werden und die Schwerkraft wirkt auch auf den Spieler ein. Zusätzlich wurde eine Collision Detection zwischen der geschossenen Kugel und den Feinden und dem Spieler und den Feinden implementiert. Werden diese Feinde mit einer Kugel getroffen, fliegen sie auf den Boden und bleiben dort liegen. Für den Spieler sind diese Objekte dann blockierend.

Zusätzlich verursacht eine Trigger Shape am Ende des Levels den Übergang in das nächste Level.

View-Frustum Culling

Leider war die Lösung unserer Probleme mit dem VFC nicht in Sicht und hat uns unzählige Stunden gekostet. Da dieser Effekt nur optional ist, haben wir uns schweren Herzens dafür entschieden, nicht mehr daran weiter zu arbeiten, auch wenn wir gefühlt nur 1,2 Zeilen von der Lösung entfernt waren. Die Implementierung ist noch vorhanden, wird aber default-mäßig nicht angewendet.

Text Rendering

Um Informationen wie beispielweise die Frametime, die Anzahl gecullter Objekte oder den Time-Counter darzustellen, haben wir FreeType in unser Projekt entsprechend eines Tutorials [4] inkludiert. Da die Glyphen über die 3D-Szene gelegt werden, haben wir Alpha Blending aktiviert. Außerdem waren eigene Shader nötig, welche in unserem Projekt text.vert und text.frag heißen. Diesen Shadern wird pro Buchstaben eine eigene Textur übergeben, was natürlich nicht sehr performant ist. Aufgrund der Nebensächlichkeit in Bezug auf diese LVA und der bisherigen akzeptablen Frameraten halten wir diesen Umstand aber für tolerierbar.

Assimp

Wie bereits erwähnt, werden komplexe Modelle mit Assimp geladen. Die Implementierung davon ist in der Geometry Klasse und erstellt ein Modell aus den Mesh-Daten, die in den .dae Dateien gespeichert sind. Dabei haben wir uns an einem Tutorial orientiert [5].

Effekte

Hierarchical Animation

In Level 1 besitzt einer der Feinde 4 Satellitenobjekte, welche sich in einem Orbit um ihn herumbewegen. Das funktioniert grundsätzlich über eine Liste an Child-Objekten, die einer Geometry zugewiesen werden können. Die Transformationsmatrix dieser Children wird hier zuerst von der Parent-Geometry übernommen und dann mit einer eigenen Transformation ergänzt. Dadurch folgen in dem erwähnten Fall die Satelliten dem Feind und drehen sich zusätzlich um diesen herum.

Physically based shading

Statt dem simplen Phong Modell, haben wir das ausgeklügeltere Cook-Torrance Beleuchtungsmodell verwendet. Dabei sind wir im Großen und Ganzen nach einem Tutorial vorgegangen [6].

Lightmaps mit separaten Texturen

Wurden mit Blender erzeugt und werden einfach im Shader für die Berechnung der Beleuchtung herangezogen.

Bloom

Wurde grundsätzlich nach einem Tutorial implementiert [7]. Verändert wurde die Implementierung in dem Sinne, dass wir nur einzelne meshes in der Scene haben, welche den bloom shader benützen und im fragment shader (auch im cook_torrance shader) abgecheckt wird, ob der pixel im gerenderten quad eine R-Kanal value von > 1.0 hat, in diesem Fall entspricht dieser Wert unserer Material Textur von den Bloom Objekten (Knall-Rot).

Simple Normal Mapping

Zusätzlich zum physically based shading haben wir normal mapping nach [8] implementiert, welche wir einfach aus ästhetischen Gründen eingebaut haben. Die Texturen wurden mit Photoshop erzeugt. Mit F4 kann das normal mapping getoggled werden. Bis auf Level 1 benutzen alle Objekte eine default normal map.

Specular map

Zur Berechnung der spekularen Komponente wurde eine specular map für Level 1 in Photoshop erzeugt. Alle anderen Objekte haben eine default specular map. Im Shader wird die spekulare Komponente über diese Map berechnet.

Beleuchtung & Texturierung

Momentan wird nur ein Licht (ein Point Light) in der Szene unterstützt (für den Cook Torrance shader), der Bloom shader benötigt keine Lichtquelle. Es kann vorkommen, dass einige Bereiche in der Szene sehr dunkel sind, was aber auch im Sinne der Atmosphäre des Spiels so gewählt wurde. Die Bloom Objekte selbst strahlen kein Licht ab, helfen jedoch für die Orientierung in der Scene.

Für die Texturierung wurden für das Level eine Wand-Textur, für die Gegner ein simple Rot-Textur und für die Waffe eine spezielle eigene Textur verwendet. Bis auf das Mesh von Level 1 besitzen alle Objekte keine spezielle normal und specular map.

Referenzen

- [1] <https://learnopengl.com/Getting-started/Camera>
- [2] http://cgvr.informatik.uni-bremen.de/teaching/cg_literatur/lighthouse3d_view_frustum_culling/index.html
- [3] <http://www.crownandcutlass.com/features/technicaldetails/frustum.html>
- [4] <https://learnopengl.com/In-Practice/Text-Rendering>
- [5] <http://ogldev.atspace.co.uk/www/tutorial22/tutorial22.html>
- [6] <https://learnopengl.com/PBR/Lighting>
- [7] https://learnopengl.com/code_viewer_gh.php?code=src/5.advanced_lighting/7.bloom
- [8] https://www.gamasutra.com/blogs/RobertBasler/20131122/205462/Three_Normal_Mapping_Techniques_Explained_For_the_Mathematically_Uninclined.php