

Layers of Sorrow

Dokumentation

Manuel Wieser (01633066)

Sebastian Karall (01635881)

Allgemein

Bei "Layers of Sorrow" handelt es sich um einen Dungeon-Crawler, bei dem der Hauptcharakter sich einen Weg durch Monster schlagen muss, um ans Ende in das Licht zu gelangen.

Features

- *HUD*: Die Lebenspunkte der Spielfigur und der (anvisierten) NPCs, sowie mögliche Attacke werden in 2D über der 3D-Welt und dem eigentlichen Spielgeschehen angezeigt. Außerdem ist ein FPS Counter sichtbar.
- *Frei bewegliche Kamera*: Diese wurde als observierende 3rd-Person-Kamera mittels Maussteuerung (Drag'n'Drop) und den WASD-Tasten zur Steuerung der Spielfigur umgesetzt. Sie verfolgt somit die Hauptcharakter in einem durch Zoom variablen Abstand
- *Ray-Tracing*: Gegner können mittels PhysX anvisiert werden, um Aktionen nur auf einen gegner zu wirken.
- *Texture Mapping*: Es werden mithilfe von FreeImage eine Vielzahl an Bilddateitypen unterstützt.
- *Model Import*: Levels können in Blender erstellt und als .obj exportiert werden. Durch assimp wird zwar auch das .blend Dateiformat unterstützt jedoch reicht das .obj Format aus.
- *Mesh Cooking*: Aus Vertices und deren Index werden Formen mit PhysX modelliert, welche auch zur Kollisionsdetektion verwendet werden können. Somit sind komplexe Boden- und Wandformen im Spiel möglich.
- *Ingame Lightmapping*: Auf statische Modelle wird eine Lightmap beim Starten des Spiels berechnet. Dazu werden mit einem Compute Shader die Light-Texturen berechnet und zur Laufzeit im Fragment-Shader kombiniert.

- *Partikel mit Compute Shader*: Der Flächenattacke-Effekt wird mittels Compute Shader berechnet. Dazu werden die Positionen der Partikel berechnet und im Anschluss einem Vertex/Geometry/Fragment Shader übergeben, um eine Textur darauf zu zeichnen.
- *Keyframe Animation*: Die Spielfigur besitzt mehrere Posen, welche in Blender erstellt wurden. Ein Compute Shader interpoliert zwischen den Posen, um eine Laufanimation darzustellen.
- *Gegner Animation mit Physx*: Die Gegner im Spiel werden mit Physx bewegt, damit sie mit dem Spieler sowie mit der Umgebung kollidieren können.
- *View Frustum Culling*: Um bessere Performance zu Gewinnen, werden Objekte welche sich nicht im View Frustum befinden erst gar nicht an die GPU übergeben.

Auswahl der Gegner

Damit ein Gegner anvisiert werden kann, wird mittels Physx ein Ray von der Camera Position geworfen. Dazu wurde die Bildschirmposition zurück in World-Koordinaten gerechnet. Um die Richtung zu eruieren konnte die z-Koordinate einmal auf -1 und auf 1 gesetzt werden, da der exakte z Startpunkt keine auswirkung auf die Richtung hat. Mittels der Position des Pixels in World-Koordinaten und der Richtung wird nun ein Ray geworfen und überprüft ob es sich bei dem getroffenen Actor um einen Gegner handelt bzw. ob überhaupt etwas getroffen wird.

View Frustum Culling

Damit mit schon frühzeitig erkannt wird, ob sich ein Objekt im View Frustum befindet wird ein Algorithmus angewandt, welcher durch Geometrie bzw. Bounding Boxen feststellt, ob es sinnvoll ist das Objekt an die GPU zu übergeben. Dabei wurde der Algorithmus nach dem Tutorial von http://cgvr.informatik.uni-bremen.de/teaching/cg_literatur/lighthouse3d_view_frustum_culling/index.html angewant.

Model Import

Bei dem Model Import wird grundlegend zwischen dem Level und dem Charakter unterschieden. Die von Assimp gelieferte Struktur wird in die eigene Umgewandelt ebenso informationen wie Material, Textur und Parrent/Child Beziehungen. Die Art

der Objekte wird anhand eines Präfixes im Namen unterschieden. So gibt es einen Präfix für Modelle dessen Mesh "gecookt" werden soll, einen für Gegner und einen für Objekte wenn mit denen Kollidiert wird soll der Gewinn Bildschirm ausgegeben werden.

Beleuchtete/Texturierte Objekte

- Das "Level", sprich eine Höhle bestehend aus
 - Boden
 - Tunnel
 - Anfangs- sowie Endwand
- Spielfigur
- NPCs, zurzeit eine einzige Gegnerklasse
- Aufschrift an der Anfangswand
- Tür am Ende des Levels

Alle Objekte bis auf die Spielfigur befinden sich in main_scene.obj. Es werden im Spiel jedoch die einzelnen Objekte anhand eines Präfix im Namen unterschieden. So kann durch den Präfix bestimmt werden ob ein Objekt ein gegner ist, eine Kollision ein Event auslöst oder ob darauf Mesh cooking angewandt werden soll.

Die Materialkoeffizienten werden aus der .obj zugehörigen .mat Datei ebenfalls mit Assimp eingelesen und in eine eigene Datenstruktur gespeichert.

Alle Objekte haben ein zugewiesenes Material, oberflächen Normalen und UV Koordinaten. Somit kann ein Phong Shading Modell angewendet werden. Bei statischen Objekten wird eine Lightmap beim start des Spiel berechnet und im Anschluss im Fragment Shader mit der Textur kombiniert.

Lichter sind statisch im Code definiert und befinden sich am Eingang und am Ausgang. Auf der Start- und Endwand sowie den angrenzenden Wänden der Höhle ist der Lightmapping effekt gut zu sehen. Die Lightmap kann mittels F4 ein und ausgeschaltet werden.

Steuerung und Gameplay

Effekt	Tasten
Bewegung der Spielfigur	W, A, S, D
Bewegung der Kamera mit Spieler	Rechte Maustaste (Drag'n'Drop)
Bewegung der Kamera ohne Spieler	Linke Maustaste (Drag'n'Drop)
Kamera Zoom	Mausrad
Gegner anvisieren/auswählen	Linke Maustaste
Normale Attacke	1 (nach dem Anvisieren)
Starke Attacke	2 (nach dem Anvisieren)
Flaschen Attacke (Partikel Effekt)	3
Hilfe Text	F1
FPS Limiter deaktivieren/aktivieren	F2
Wireframe deaktivieren/aktivieren	F3
Lightmap deaktivieren/aktivieren	F4
View Frustum Culling ON/OFF	F8
Backface Culling ON/OFF	F9
Spiel beenden	Esc

Die Voreinstellung der diversen gewünschten Parameter erfolgt in Textform über das Config-File "settings.ini". Folgende Parameter können geändert werden:

- Auflösung
- Fenstertitel
- Vollbildmodus
- Erneuerungsrate
- Helligkeit
- Kameraparameter
- FPS

Zusätzliche Bibliotheken

Nvidia PhysX:

Mit PhysX wird die allgemeine Kollisionsdetektion, sowie die Bewegung von Spielfigur und NPCs innerhalb der Spielwelt umgesetzt.

Es handelt sich um eine Physik-Engine des Unternehmens Nvidia. PhysX dient zur Berechnung physikalischer Effekte in Computerspielen und Simulationssoftware.

URL: <https://developer.nvidia.com/gameworks-physx-overview>

Mithilfe von PhysX werden der Charakter sowie die NPCs bewegt da sie sich auf einer unebenen Fläche bewegen welche zusätzlich mit Wänden beschränkt ist. Dazu wird der Spielcharakter als Zylinder dargestellt und die Gegner als Boxen. Die Koordinaten und die Größe werden anhand des Meshes berechnet.

Die Bewegung erfolgt somit über die move funktion des Controllers. Da man diese Actor gezielt bewegen möchte und nicht mit Forces.

Der Nvidia PhysX Visual Debugger ist ein Tool, um die Actors in einer PhysX Scene darzustellen. Damit kann die Positionierung und vieles mehr überprüft werden.

Mit PhysX wird auch ein Ray-Cast durchgeführt, um den Gegner zu bestimmen auf den der Spieler oder die Spielerin geklickt hat. Dazu werden zuerst die Bildschirmkoordinaten mit der umgekehrten Projektionsmatrix umgewandelt und anschließend mit der (umgekehrten) Transformation der Kamera positioniert. Somit können mit Position und Richtung des Rays die Auftreffenden Objekte bestimmt werden.

Assimp:

<http://www.assimp.org/>

Mit assimp können eine Vielzahl von 3D Objektdateien importiert werden. Assimp speichert geladene Objekte in einer eigenen Datenstruktur, diese wurde in die eigene umgewandelt damit die Meshes mit dem eigenen Shader gerendert werden können.

Das Grundkonzept des Imports wurde von folgendem Tutorial übernommen.

<https://learnopengl.com/Model-Loading/Assimp>

FreeImage:

<http://freeimage.sourceforge.net/>

Damit ist es möglich eine große Auswahl an Dateiformaten zu lesen und schlussendlich in einem Array zu speichern, um sie mit OpenGL auf die GPU zu laden. Dadurch ist es einfacher, Modelle mit existierenden Texturen mit assimp zu Importieren da die Texturen nicht manuell umgewandelt werden müssen.

Als Hilfe diene dieser Beitrag.

<https://r3dux.org/2014/10/how-to-load-an-opengl-texture-using-the-freeimage-library-or-freeimageplus-technically/>

FreeType:

<https://www.freetype.org/>

Innerhalb des Spiels wird FreeType zum Anzeigen des HUD und etwaiger Text-Rückmeldungen in 2D genutzt.

FreeType ist eine freie Programm-Bibliothek, die in der Lage ist Schriftarten zu laden und sie in Bitmaps zu rendern. Die Bibliothek ist besonders gut geeignet, weil sie TrueType-Schriften laden kann. Diese Schriftarten sind eine Sammlung von Glyphen, die nicht durch Pixel oder ähnliche nicht skalierbare Ansätze definiert sind, sondern durch mathematische Gleichungen. Somit können die gerasterten Schriftbilder prozedural generiert werden, basierend auf der bevorzugten Schrifthöhe. Somit können Glyphen verschiedener Größe ohne Qualitätsverlust dargestellt werden. *Tutorial:* <https://learnopengl.com/In-Practice/Text-Rendering>

Um ein HUD jedoch anzuzeigen musste ein neuer Shader geschrieben werden, welcher eine orthogonale Perspektive übergeben bekommt. Mithilfe eines HUDs ist es auch möglich einen FPS Counter direkt im Spiel anzuzeigen.

Implementierte Effekte

Light Mapping + In-Game-Calculation

Der Light Mapping Algorithmus basiert grundlegend auf dem Pseudocode von http://www.flipcode.com/archives/Light_Mapping_Theory_and_Implementation.shtml. Nach einem Test des Algorithmus auf der CPU wurde schnell klar, dass das zu lange dauert. Daher werden die Light Maps nun in einem Compute Shader berechnet, da jedes Pixel parallel berechnet werden kann. Im Anschluss werden MipMaps mit OpenGL generiert und als Textur im Shader gebunden.

GPU-Particle System (Compute Shader)

Auch das Partikel System wurde mittels eines Compute Shaders realisiert. Dazu wurde nach dem Tutorial der Lehrveranstaltung vorgegangen und Modifikationen durchgeführt, damit die Partikel in einem Ring "explodieren". Zusätzlich wurde eine "Restart" Funktion eingebaut, um den Effekt sowie die SSBOs wiederzuverwenden. Die Pipeline um die Partikel zu rendern wurde auch aus dem Tutorial bzw. von <http://www.geeks3d.com/20140815/particle-billboarding-with-the-geometry-shader-gles/> großteils übernommen.

Heads-Up Display

Hauptsächlich besteht das HUD aus Text. Dazu wird jedes Zeichen mit FreeType aus einer Schriftart gelesen und als Textur auf die GPU geladen. Um das HUD zu rendern muss zunächst Blending aktiviert werden und einem eigenen Shader ein Quad übergeben werden, welcher die Textur im Vordergrund ohne Verzerrung rendert. Der Code aus dem Tutorial von <https://learnopengl.com/In-Practice/Text-Rendering> wurde dahingehend erweitert, dass der Shader auch mit Bildern umgehen kann, um normale Bilder als HUD darstellen zu können.

Physics Engine

Mit der Physik Engine von Nvidia werden jeweils der Charakter als auch die Gegner gesteuert, um Kollisionen mit der Umgebung zu erreichen damit z.B. die Modelle

nicht durch Wände gehen können. Zusätzlich wird durch PhysX die Win- bzw. Losecondition eruiert. Kommt die Spielerin bzw. der Spieler bis ans ende des Levels mittels einer Kollision erkannt, dass das Ende erreicht wurde. So werden auch Kollisionen mit Gegnern überprüft, um Schadenspunkte zuzufügen.

Siehe

Animation mit Keyframes

Die Animation der Hauptfigur entsteht durch interpolation von 3 Models. Das Modell wurde aus dem Internet bezogen und nach Blender importiert. Dort wurde ein Skelett erstellt, um mit diesem die Bewegung der gliedmaßen durchzuführen. Es wurden 3 Posen erstellt und als obj exportiert. Im Spiel wird mithilfe eines Compute Shaders zwischen dem Input und Output Modell alle Vertex Positionen und Normalen interpoliert und als ausgabe gespeichert. Die daraus entstandenen Positionen und Normalen werden in einem Vertex Array gebunden und schlussendlich gezeichnet.