Studytime – Submission 2

Roman Püngüntzky
David Schibl

Implementierung

Datenstrukturen

Szene

Alle Objekte in unserer Szene sind in Listen gespeichert. Beim Updaten wird für jedes einzelne Szenen-Objekt eine eigene update()-Methode ausgeführt. Das Rendern erfolgt über eine gemeinsame render()-Methode.

Models

Im Gegensatz zur 1. Submission werden die Models nun alle durch einen eigenen VAO gezeichnet. Sie werden mit Assimp importiert, auf etwaige Texturen und Animationen überprüft und die Daten direkt in entsprechende Buffer geschrieben. Die zugehörigen Texturen und Bones sind dabei ebenfalls in Listen abgelegt.
Die **Texturen** werden bei uns mit Hilfe de devIL Library geladen und gespeichert.
Unser **komplexes Objekt** ist das Player-Model. Es wird mittels GPU Vertex Skinning beim gehen (drücken von W oder S) animiert. Durch den rekursiven Durchlauf der Bones und die Gewichtung derselben geschieht dies natürlich auch **hierarchisch**.

View Frustum Culling

View Frustum Culling basiert auf einer geometrischen Herangehensweise über Sphären. Jedes Szenen-Objekt besitzt einen gewissen Sphären-Radius, über welchen ein Schnitt mit den einzelnen View Frustum Planes ermittelt wird.

Physik

Die Collision-Detection wird über die Physik-Simulation Bullet realisiert. Dabei wird eine Welt aus Physik-Objekten parallel zur sichtbaren Welt gerendert. Diese ist durch die Klasse **Physic** repräsentiert, in der die Initialisierung der Physik-Welt, das Hinzufügen/Entfernen einzelner Objekte sowie die Collision-Detection behandelt wird.
Die Map selbst wird mit Assimp in die Physikwelt geladen, damit das zugehörige Collision-Shape übereinstimmt. Für die anderen Models triviale Collision-Shapes (Quader, Sphären) verwendet.

Die Kollisionen der einzelnen Objekte im Physik-Raum werden über Collision-Masks realisiert. Für jede Art von Objekt ist in einer Maske definiert, mit welchen anderen Objekten es zu kollidieren hat und mit welchen nicht.
Die Gegner haben eine hohe Masse und können damit nicht vom Player aufgehalten werden.
Bücher werden bei einer Kollision mit anderen Objekten in der Szene gelöscht. Kollidieren sie mit einem Gegner, so verlieren diese Lebenspunkte (momentan: 1 Lebenspunkt pro Schuss).

Die Jump-Funktion des Spielers wird über Raycasting ermittelt. Es wird vorwiegend nach vorne und unten gecastet. Sollte das Modell keinen Kontakt mit einem anderen Objekt haben, so kann es nicht springen. Dies verhindert u.A. Double-Jumps, anzumerken ist, dass der Player nur vom definierten Boden der Map springen kann. Um die Fallen zu aktivieren wurde als Trigger eine 2. Collsion-Shape (in Form einer Sphäre) implementiert.

Enemy Path Following

Die gespawnten Wellen von Gegnern folgen vordefinierten Pfaden, die über Arrays von 3D Vektoren vordefiniert sind. Die Abarbeitung dieser Pfade wird über ein Checkpoint-System gelöst.

Wave Management

Es gibt insgesamt 10 Wellen von Gegnern, die sich aus zufälligen Gegnertypen zusammensetzen. Im Laufe des Spiels werden die Gegner stärker, und alle 5 Wellen gibt es eine Boss-Welle, in der ein mächtiger, einzelner Gegner besiegt werden muss.
Prinzipiell gibt es 3 unterschiedliche States, zwischen denen unterschieden wird: Pause zwischen Wellen, Welle im Gange und Warten bis Welle fertig ist.

Game States

Es gibt insgesamt 4 Game States: Loading, Play, Game Over und Win. In Loading wird ein Ladescreen mit der Steuerung des Spieles angezeigt. Nach dem Drücken von Space gelangt man in den Play State. Wenn das Hirn 0 Lebenspunkte erreicht, so ist das Spiel vorbei und wechselt in den Game Over State, aus dem heraus mit ESC das Spiel beendet werden kann. Bei einem gewonnenen Spiel wird eine Studytime-Testscore angezeigt, die sich aus der Win-Condition und den übrigen Lebenspunkten des Hirns berechnet ;-).

Experimente mit OpenGL

VBOs werden für alle Meshes sowie Particles verwendet. Unsere Meshes werden ausschließlich über **VAO**s gezeichnet. Ein **FBO** wird beim Rendern der Shadow-Map verwendet.
Die geforderten **experimentellen OpenGL Funktionen** liegen auf den gewünschten F-Tasten. Als Feedback wurde eine Konsolen-Ausgabe gewählt, in der ersichtlich ist, was gerade eingestellt bzw. aktiviert wurde.

Hinweise zum Kompilieren in VS.NET

Visual Studio speichert leider die Working Directory im .user File. Da diese standardmäßig auf $(ProjectDir) gesetzt ist, wird das Kompilieren unseres Spieles nicht auf anhieb funktionieren.

Damit es kompiliert werden kann, muss lediglich unter **Configuration Properties -> Debugging** die Working Directory auf **$(SolutionDir)** gesetzt werden.

Features

- Physik und Collision Detection
- Gegner folgen verschiedenen Pfaden
- Gamma Korrektur
- HUD
- Custom Map
- Mehrere Typen von Fallen und Gegnern, inkl. Bosse
- Beschränkte Fallen Positionierung und Reichweitenmarkierung beim Platzieren
- Geometrisches View Frustum Culling über Sphären
- Particles
- Phong Beleuchtungsmodell
- Schatten
- Animiertes Player-Model

Beleuchtung und Texturierung

Als **Beleuchtungsmodell** wird das Phong Modell für alle Objekte der Szene (mit Ausnahme der Skybox) verwendet. Dabei gibt es eine Punktlichtquelle, die aus der Richtung des Mondes die Objekte der Szene beleuchtet. Neben dem Phong Modell haben wir auch Attenuation, also die Abnahme der Intensität über die Distanz zur Lichtquelle, sowie eine Gamma-Korrektur implementiert. Die Gamma-Korrektur lässt sich über die Tasten F10 und F11 dynamisch zur Laufzeit ändern.
In dieses Beleuchtungsmodell wurde weiters die gerenderte Shadow-Map integriert, sowohl für nicht-animierte als auch für animierte Objekte unserer Szene.

Transparenz wird in unserem Spiel, neben den HUD-Elementen, beim Platzieren von Fallen genutzt. Transparente Sphären in Blau bzw. Rot zeigen an, ob eine Falle gesetzt werden kann oder nicht. Diese Sphäre wird nicht beim Shadow-Mapping berücksichtigt, da sie nur zur Anzeige der Reichweite dient und kein eigentliches Szenenobjekt darstellt.

Die **Texturen** des Players und des HUDs sind in Photoshop von uns selbst gemacht worden. Bei den restlichen verwendeten Models waren die Texturen bereits integriert.
Die **Skybox** wird als Cube-Map Textur auf eine Sphäre gemappt und verwendet selbst einen eigenen, einfachen Shader, der die Fragmentfarbe von der Textur 1:1 übernimmt.
Die Elemente des **HUD**s werden über Pixelangaben (x, y) mit Quads orthogonal auf den Bildschirm gerendert und mit einfachen UV-Koordinaten versehen. Der Quad der Healthbar des Brains wird mit den aktuellen Lebenspunkten gestaucht bzw. gestreckt. Die **Fonts** werden auch über Quads orthogonal gerendert und über die ASCII-Codes der Buchstaben mittels UV-Koordinaten aus einer 16x16 Bitmap-Textur ausgelesen. Zu allen Elementen ist anzumerken, dass in einem eigenen Vertex-Shader die (x, y) Positionen in homogene Koordinaten umgerechnet werden, damit die Darstellung mit unterschiedlichen Auflösungen korrekt funktioniert.

Für eine Liste der Models und Quellen siehe Abschnitt Models. Folgende Texturen wurden aus dem Internet bezogen:

- Skybox: https://93i.de/products/media/skybox-texture-set-1
- Map: https://lva.cg.tuwien.ac.at/textures/

- Brain Icon: http://www.1stwebdesigner.com/wp-content/uploads/2010/07/How-to-Illustrate-a-Brain-Icon-for-OSX-and-Vista.png
- Book Icon: http://game-icons.net/lorc/originals/book-cover.html
- Snail Icon: http://game-icons.net/lorc/originals/snail.html
- Gun Icon: http://game-icons.net/lorc/originals/gunshot.html

Effekte

- **(1.5 Punkte) Shadow Mapping mit PCF**
  http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/
  Wir sind im Wesentlichen nach dem genannten Tutorial vorgegangen und haben PCF mit Poisson-Sampling ergänzt, um die harten Schatten etwas weicher zu machen. Wir tasten die Schattentextur 4-Mal ab. Die Schatten wurden dann erfolgreich für nicht-animierte und animierte Objekte in unseren Phong-Shader integriert.
- **(1.0 Punkte) GPU Particle System über Transform Feedback**
  http://ogldev.atspace.co.uk/www/tutorial28/tutorial28.html
  Dieser Effekt wurde nach dem genannten Tutorial implementiert, an unsere Bedürfnisse angepasst und mit eigenen Fragment-Shadern ergänzt. Die Particles sind beim Treffen von Gegnern und beim Verlust von Lebenspunkten bemerkbar (Explosionen, Rauch).
- **(2.0 Punkte) GPU Vertex Skinning**
  http://ogldev.atspace.co.uk/www/tutorial38/tutorial38.html
  Die Extraktion der Animationen aus den Model-Files wurde eng nach dem genannten Tutorial vorgenommen. Die Animation wurde weiters mit den Bewegungen des Player-Models verbunden und unterstützt sowohl ein Vorwärts-Abspielen sowie ein Rückwärts-Abspielen der Animationen. Das Skinning wurde erfolgreich in unseren Phong-Vertexshader integriert. Rigging und erstellen der Animationen wurde in Maya selbst durchgeführt.

Controls

Die Tastatur- und Mauseingaben werden mittels Polling und GLFW ermittelt.

Taste	Effekt
W, A, S, D	Bewegen des Players (W=Vorne, A=Links, S=Hinten, D=Rechts)
Maus	Drehen des Players
Linke Maustaste	Schießen
Rechte Maustaste	Falle platzieren (sofern ausgewählt)
1, 2, 3	Auswahl der Waffen und Fallen (1=Angriff, 2=Verlangsamungsfalle, 3=Schadensfalle)
Shift	Sprinten
Space	Springen
ESC	Beendet das Spiel
G	Toggle Cheat Mode (Off, On)
M	Toggle Sound (On, Off)
F1	Hilfe (pausiert das Spiel)
F2	Toggle FPS/Frametime/Triangles
F3	Toggle Wire Frame Mode
F4	Toggle Texture Sampling Quality (Nearest, Linear)

F5	Toggle Mip Mapping Quality (Off, Nearest, Linear)
F8	Toggle View Frustum Culling (Off, On)
F9	Toggle Transparenz (Off, On)
F10	Gamma erhöhen
F11	Gamma erniedrigen
F12	Toggle Physik Debug Mode (Off, On)

Tools

Das Map-Model wurde von uns in **Autodesk Maya**[1] erstellt und texturiert. Für die Animation des Player-Models und diverse Korrekturen der heruntergeladenen Models wurde ebenfalls Maya verwendet.

Models

Alle verwendeten Models in unserem Spiel (mit Ausnahme der Map) wurden aus dem Internet bezogen und teilweise bearbeitet. Die folgende Liste zeigt die Quellen aller Models:

- Player: http://tf3dm.com/3d-model/toon-scientist-69583.html
- Enemy TV: http://tf3dm.com/3d-model/tv-26000.html
- Enemy Phone: http://tf3dm.com/3d-model/iphone-5-53069.html
- Enemy Ball: http://tf3dm.com/3d-model/soccer-ball-82644.html
- Buch: http://tf3dm.com/3d-model/antique-book-28979.html
- Core (Brain): http://www.3dmodel777.com/Download-3d-model/Character-and-role/Human-brain-3d-model.html (von uns weiter bearbeitet)
- Münze: http://tf3dm.com/3d-model/coin-14340.html (von uns weiter bearbeitet)
- Skull Powerup: http://tf3dm.com/3d-model/pirate-skull-14144.html (von uns weiter bearbeitet)
- Herz Powerup: http://tf3dm.com/3d-model/valentine39s-day-heart-69073.html
- Bossmodels: http://tf3dm.com/3d-model/lcd-television-169-37070.html

Sounds

Alle verwendeten Sounds wurden aus dem Internet bezogen und verfügen über eine Creative Commons Lizenz.

- BG - Normal http://opengameart.org/content/oves-essential-game-audio-pack-collection-160-files-updated
- BG - Bossfight http://opengameart.org/content/last-minute
- PU - Health http://www.flashkit.com/soundfx/People/Breathing/Female_B-LadyIT-2321/index.php
- PU - Kill http://www.flashkit.com/soundfx/Cartoon/Explosions/idg-expl-intermed-778/index.php
- PU - Money http://opengameart.org/content/coins-sounds
- Coin collected http://opengameart.org/content/coins-sounds

[1] http://www.autodesk.de/products/autodesk-maya/overview

- Trap placed http://www.flashkit.com/soundfx/Cartoon/Clanks/thud-Aaron_Sc-9026/index.php
- Brain hit http://opengameart.org/content/collaboration-sound-effects-fx-005

Libraries

Es wurden ausschließlich Libraries verwendet, die im CG Wiki vorgeschlagen wurden:

- GLFW: http://www.glfw.org/ Window Manager
- GLEW: http://glew.sourceforge.net/ OpenGL Extension
- GLM: http://glm.g-truc.net/ Mathematics Library
- Assimp: http://assimp.sourceforge.net/ Model Loader
- DevIL: http://openil.sourceforge.net/ Image Loader
- Bullet Physics: http://bulletphysics.org/ Physics Simulator
- FMOD: http://www.fmod.org/ Sound Loader