Karlsruher Institut für Technologie

aifb

3D-SPIELEENTWICKLUNG MIT JAVA

Seminararbeit von
JULIAN WADEPHUL UND FLORIAN ROTTACH

an der KIT-Fakultät für Wirtschaftswissenschaften

eingereicht am : 20. Februar 2017
Studiengang : Wirtschaftsingenieurwesen

Institut für Angewandte Informatik und Formale Beschreibungsverfahren
KIT - Die Forschungsuniversität in der Helmholtz-Gemeinschaft

ZUSAMMENFASSUNG

In dieser Seminararbeit wird die 3D-Programmierung in Java behandelt. Zur Vereinfachung wird eine Game - Engine namens jMonkey 3 verwendet. Auf Basis dieser wurde ein kleines Spiel programmiert, anhand welchem die fundamentalen Ideen und Umsetzungen von 3D-Programmierung geschildert werden.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

ABKÜRZUNGEN

1 EINLEITUNG

Im Rahmen des Seminars "Programmieren 3: XXXXXXXXXXXXXXß-oll in dieser Seminararbeit die Thematik der 3D-Spieleentwicklung dargestellt werden. Die Thematik soll jedoch nicht nur in einer schriftlichen Ausarbeitung, sondern auch in einem sogenannten Workshop mit den Seminarteilnehmern behandelt werden. In dem Workshop gilt es die grundsätzlichen theoretischen Hintergrundinformationen darzustellen, und in einem praktischen Teil anhand von Aufgaben selbst etwas zu programmieren. Um den Seminarteilnehmern einige Inhalte zu vermitteln, haben wir uns entschlossen ein eigenes 3D-Spiel zu programmieren, welches im Workshop vervollständigt werden soll. Neben der veranschaulichten Darstellung der zu lernenden Inhalte, zeigt unser eigenes 3D-Spiel auf, was mit Java in der 3D-Spieleprogrammierung möglich ist. In dieser Seminararbeit werden wir die Grundlagen der 3D-Spieleprogrammierung behandeln, und dies anhand unseres 3D-Spiels veranschaulichen. Im folgenden möchten wir unser Spiel kurz vorstellen.

Das Spiel "Progmanïst eine Anlehnung an das bekannte Horror-Spiel SSlenderman". In diesem Spiel geht es darum, dass der Spielende sich in einem Wald befindet und dort die 9 Bücher über die 9 anderen Themen in dem Seminar finden muss. Dabei wird er jedoch von einer Figur verfolgt, dem Progman, welcher versucht den Spielenden zu fangen. Ganz wesentlich hierbei ist die gruselige Stimmung, die durch Licht, Sound und Modelle in der Welt generiert wird. Im Laufe des Spiels nähert sich der Progman immer mehr an, bis er den Spielenden gefunden hat. Dabei spielen Faktoren wie die Anzahl der bereits eingesammelten Bücher und die häufige Benutzung der Taschenlampe in die Geschwindigkeit des Progman ein.

Für die Programmierung eines 3D-Spiels in Java stehen verschiedene Engines zur Verfügung. Diese Engines beinhalten vorgefertigte Klassen und Methoden, welche das Programmieren eines 3D-Spiels deutlich vereinfachen. Wir haben uns für die jMonkeyEngine entschieden, doch darauf möchten wir später noch genauer eingehen. Außerdem werden wir verschiedene Themen der Umsetzung im Programmcode untersuchen. Am Ende möchten wir noch die Optimierung eines 3D-Spiels beschreiben, da ein nicht-optimiertes 3D-Spiel sehr schnell zu aufwendig werden kann.

2 3D SPIELEENTWICKLUNG MIT JAVA

2.1 ALLGEMEINER AUFBAU VON 3D SPIELEN

Bei 3-dimensionalen Spielen wird das Spielgeschehen in einen Raum transferiert und dem Spieler die Möglichkeit gegeben sich dort frei bewegen zu können.
Im Gegensatz zu 2-dimensionalen Spielen sind hier deutlich mehr Berechnungen auf vektorieller Ebene notwendig, was sich auf die Laufzeit niederschlägt. Daher ist es besonders wichtig ein effizientes Programm zu generieren. Diesbezüglich ist Java nicht die optimale Programmiersprache, da durch den Interpreter viel Zeit verloren geht.
Im Allgemeinen kann man sagen, dass die meisten 3D Spiele folgende Elemente enthalten:

- 3D-Modelle und eine räumliche Spielumgebung
- Eine sog. "Kamera", welche nur den relevanten Teil des Bildes abbildet
- Möglichkeiten der Interaktion

https://de.wikipedia.org/wiki/Spiel-Engine

2.2 FUNKTIONSWEISE UND AUSWAHL VON GAME ENGINES

Um nicht sämtliche mathematischen Berechnungen auf der Grafikkarte selber programmieren, oder beispielsweise die Lautsprecher für Audio-Effekte ansprechen zu müssen, erhält der Entwickler Unterstützung durch sogenannte "Game Engines". Diese beinhalten die Basisfunktionen von Spielen und ermöglichen dem Spiele-Programmierer eine gezieltere Entwicklung. Zu den Basisfunktionalitäten gehören im Allgemeinen die folgenden:

1. Grafik-Engine
2. Physiksystem
3. Soundsystem
4. Zustandsspeicherung
5. Steuerung
6. Datenverwaltung

GAMEENGINE	VORTEILE	NACHTEILE
Unity	Sehr bekannt, beliebt	zu oberflächlich, kein Java
jMonkeyEngine	Sehr entwicklungsnahe, Java	Schlechte Dokumentation
Wurfel Engine	Sehr benutzerfreundlich	keine Physikunterstützung

Tabelle 1: Vor und Nachteile einiger Game Engines

Zur Auswahl stehen eine Vielzahl von verschiedenen Engines, welche jeweils vor und Nachteile mit sich bringen. Da wir auf jeden Fall lernen wollten, wie die grundlegenden Dinge funktionieren, haben wir nach Engines gesucht, welche nur die Basisfunktionalitäten unterstützen, jedoch keine automatische Codegenerierung, Drag and Drop oder Editoren beinhalten. Im folgenden eine Übersicht einiger Engines:

Letztendlich fiel die Entscheidung auf die jMonkeyEngine, welche häufig von Java Entwicklern verwendet wird.
https://de.wikipedia.org/wiki/Spiel-Engine https://de.wikipedia.org/wiki/Liste-von-Spiel-Engines

2.2.1 *Die jMonkeyEngine*

Die jMonkeyEngine (jME) ist komplett in Java geschrieben und basiert auf dem Buch "3D Game Engine Design"von David Eberly. Durch eine Abstraktionsschicht kann jedes beliebige Rendering System verwendet werden, beispielsweise die Lightweight Java Game Library (LWJGL) oder die Open Graphics Library (OpenGL). Die neuste Version ist jME3, welche einige hilfreiche Funktionen mit sich bringt, wie z.B. ein Partikelsystem, Frustum Culling oder 3D Sound Unterstützung.
https://de.wikipedia.org/wiki/JMonkeyEngine

2.3 UMSETZUNG IN PROGRAMMCODE

Im folgenden wird beschrieben wie einzelne Elemente in der jMonkeyEngine programmiert werden können und was dabei zu beachten ist.
https://docs.jmonkeyengine.org/beginner/beginner-intro.html

2.3.1 *Erzeugung der Application-Klasse: SimpleApplication*

Die Main-Klasse jedes jME3 Spieles erbt von der Klasse SimpleApplication, welche ein Spiel darstellt. In der main-Methode wird dann eine neue Instanz erstellt und anschließend gestartet.

Jede Unterklasse der SimpleApplication beinhaltet diese Methoden:

1. simpleInitApp(): Sorgt für das Laden von Modellen, der Erstellung einer räumlichen Umgebung sowie jegliche Initiierungen.
2. simpleUpdate(float tpf): Wird jedes frame per second (fps) ausgeführt und kümmert sich um gegebenenfalls neue Zustände.
3. simpleRender(RenderManager rm): Wird stets nach simpleUpdate aufgerufen und zeichnet das Sichtbild des Spielers neu. Dazu bekommt die Methode einen RenderManager übergeben, welcher Präferenzen beim Zeichnen berücksichtigt (z.B. welche Ebene vorne oder hinten ist).

Die erste der drei Methoden wird stets zu Beginn ausgeführt um alle benötigten Elemente bereit zu stellen.

2.3.2 *Funktionsweise von Nodes*

Um Elemente zum Renderingprozess hinzuzufügen, um sie also sichtbar zu machen, müssen diese an ensprechende "Nodes"(Knoten) angehängt werden. Hierbei gibt es je nach Verwendungszweck verschiedene Arten zum Beispiel audioNode für Soundobjekte oder guiNode für Elemente auf der Benutzeroberfläche. Zuletzt werden alle Nodes an die rootNode, also die Wurzel, angehängt. Im Programmcode funktioniert dies mit der Methode attach() bzw. detach() zum entfernen. Selbstverständlich können Objekte auch direkt an die rootNode angehängt werden. Allerdings ist es empfehlenswert eine geeignete Baumstruktur zu erstellen um so bestimmte Elemente in Gruppen anzusprechen. Beispiel: Erzeugung einer eigenen Node durch:

```
Node myNode = new Node();
```

Wird nun beispielsweise die folgende Funktion auf dem Konten ausgeführt,

```
myNode.doSomething();
```

so wird diese auch für sämtliche Kinder des Knotens ausgeführt.

2.3.3 *Modelle und Assets*

Sämtliche externe Gegenstände des Spiels werden im assets Ordner im jME3 Projekt gesammelt. Dies sind multi-media Dateien wie 3D-Modelle, Soundfiles, Texturen, Shader und was sonst noch benötigt wird. Um diese dann aus dem Ordner ins Spiel zu laden wird der sogenannte AssetManager benötigt, welcher einfach eine Instanz der Klasse mit entsprechenden Funktionalitäten ist. Modelle sind dreidimensionale Gebilde welche verschiedenste Elemente in einem Spiel sein können. Hierbei verwendet jME3 die Klasse Spatial (engl. für "räumlich").

Zum Laden eines Objektes wird die entsprechende Funktion loadTexture() aufgerufen und der entsprechende Pfad zum Modell übergeben.

```
Spatial baum = assetManager.loadTexture("Models/Baum.j30");
```

Für Modelle gibt es viele verschiedene Datentypen. Neben dem jMonkey-eigenen Dateiformat j30 existieren unter anderem folgende wichtige Vertreter:

1. XML-Dateien: Aus mesh.xml Daten wird ein Objekt erzeugt.
2. OBJ-Dateien: Die allgemeine Darstellung für Objekte.
3. Blender-Dateien: Dies sind Dateien aus der Blender-Software, mit welcher Modelle erzeugt werden können.

Um Modelle sichtbar zu machen müssen diese nun nur noch zu einer Node z.B. der rootNode hinzugefügt werden.

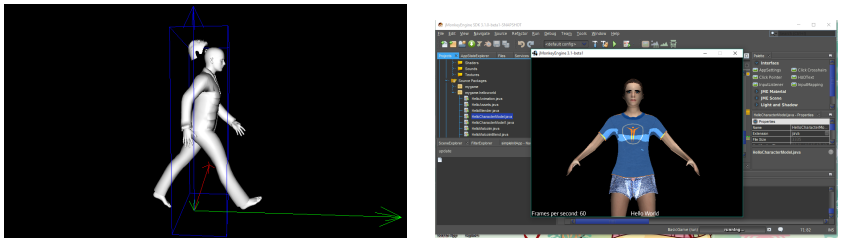
```
rootNode.attachChild(baum);
```

Neben dem Polygonzug und dem Material gibt es noch das Skelett. Dieses kann ebenfalls in einem entsprechenden Programm wie Blender erzeugt werden und daraus bestimmte Bewegungsabläufe in Spielen bestimmt werden. Das Skelett ist notwendig für Animationen von Modellen wie beispielsweise Gehen, Springen oder Ähnliches. In unserem Spiel haben wir uns für eine First-Person Perspektive entschieden, wodurch keine Animationen für den Spieler notwendig waren.

2.3.4 *Materialien*

Ein Modell besteht im Allgemeinen aus drei wichtigen Elementen. Die Hauptstruktur (Polygonzug), das Skelett (Bewegungsapparat) und dann noch das Material. Letzteres bestimmt wie das Modell aussieht, d.h. welche Farben und Strukturen das Modell an den entsprechenden Stellen bekommt. Dateien für das Material sind meist

test.material aber auch Bilddateien wie test.jpg oder test.png sind gängig.



(a) Modell ohne Material (b) Modell mit Material

Abbildung 1: Modell mit und ohne Material

https://hub.jmonkeyengine.org/t/changes-to-animations-loading-in-blender-importer-important-for-importer-users/28304/10

2.3.5 *User-Input*

Aus einem 3-dimensionalen Gebilde wird erst dann ein Spiel, wenn Interaktion mit dem Spieler stattfindet. Dazu müssen Tasteneingaben, Mauseingaben oder gegebenfalls auch Toucheingaben abgefangen und verarbeitet werden. Hierbei werden die bekannten Listener-Klassen verwendet. TODO: Welche möglichkeiten zur Spiel/Shoot/Jump usw Bewegung abgefangen werden sollten. Dann natürlich Listener Klassen und wie FirstPerson gesteuert werden kann.

2.3.6 *Kollisionserkennung*

Allgemeine Beschreibung mit mesh als Form für die Kollision. Dann natürlich wie die Kollision funktioniert (aufruf von Methoden overlap, dann nicht weiter...) Wird intern von jme3 übernommen und kann auch direkt im scene composer verwendet werden.

2.3.7 *Erzeugung einer Spielumgebung*

Terrain oder SceneComposer, sky, Funktionalität und allgemeines vorgehen.

2.3.8 *Hinzufügen von Audio*

2.3.9 *Physikalische Modellierung*

Allgemeines zu Gravity, usw... wird von jme3 unterstützt.

2.3.10 *Effekte und Details*

2.3.10.1 *Nebel*

$$\xi = \frac{2\pi z^2 e^4 N_{Av} Z \rho \delta x}{m_e \beta^2 c^2 A} = 153.4 \frac{z^2}{\beta^2} \frac{Z}{A} \rho \delta x \quad \text{keV},$$

$$\kappa = \frac{\xi}{E_{max}} \tag{1}$$

$$E_{max} = \frac{2 m_e \beta^2 \gamma^2}{1 + 2\gamma m_e / m_x + (m_e/m_x)^2} \ ,$$

2.3.10.2 *Partikeleffekte*

Feuer, Regen... usw...

2.4 OPTIMIERUNG DES PROGRAMMS

Wenn man wie oben beschrieben einige Modelle und Effekte in seine Spielumgebung einbindet, kann das sehr schnell für ein Laptop zu aufwendig werden, dieses Spiel zu rendern. Dabei spielt die Anzahl der Vertexes bzw. Triangles eine zentrale Rolle. Jedes Modell hat unter Umständen einige tausend Triangles, sodass sich das in einem Spiel sehr leicht addieren kann. In unserem Spiel gibt es zum Beispiel knapp 2000 Bäume, welche alle gerendert werden müssen: Vereinfacht man dort das Modell des Baumes, hat dies viel Potenzial, das gesamte Spiel zu beschleunigen. Mit Hilfe von F5 kann man in jMonkey während des Spiels anzeigen lassen, wie viele Triangles und Vertexes gerade zu rendern sind. Es versteht sich von selbst, dass ein Spiel mit einigen Millionen Vertexes viel zu aufwendig wird zu rendern, weshalb die Framerate auf nahezu 0 sinken kann. Um dies zu verhindern, muss man also die Anzahl an Triangles und Vertexes verringern. Dies ist grundsätzlich durch die Minimierung der Anzahl von Modellen oder durh die Minimierung der Anzahl an Triangles und Vertexes innerhalb eines Modells möglich. Diese haben damit ein sogenanntes LevelOfDetail (kurz LOD). Interessant zu beobachten ist zudem, dass ein Terrain selbst (also der Ground) sehr viele Triangles besitzen kann. Das liegt daran, dass diese Terrains dafür ausgelegt sind, dass sie aufwendige Umgebungen darstellen müssen. So können Gebirge oder sonstige Unebenheiten sehr fein dargestellt werden. Der Nachteil dabei ist jedoch, dass es dadurch sehr aufwendig wird die Texture selbst ohne Models zu rendern, obwohl diese wie in unserem Beispiel einfach nur gerade sein kann. Es ist also unbedingt notwendig bei der Programmierung eines 3D-Spiels auf die Framerate und Komplexität der Welt zu achten.

JMONKEYENGINE	VORTEILE	NACHTEILE
Dokumentation	viele Beispiele	oft deprecated
Noch ein Punkt	Julian	Florian

Tabelle 2: Beispiel für eine Tabelle

2.4.0.1 *Minimierung der Anzahl von Modellen*

2.4.0.2 *Level of Detail (LOD)*

In der jMonkeyEngine ist es grundsätzlich vorgesehen, dass man seinen Modellen eigene LODs gibt. Damit kann man anhand dieser LODs die Komplexität des Models im Laufe des Spiels bzw. vor dem Spiel verändern. Dabei gilt es noch zu beachten, dass die Modelle selbst keine LODs haben, sondern die Geometrys, welchen sie zugrunde liegene. Man kann innerhalb des SceneExplorers die einzelnen Modelle und deren Geometries auswählen, und eigene LODs generieren:

GRAFIK generateLOD.png

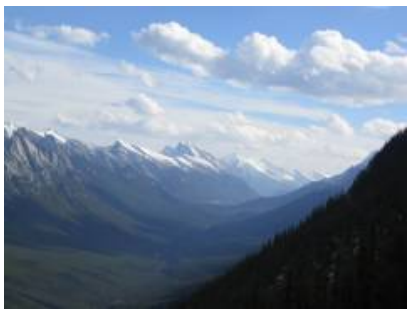
Ein alternativer Weg wäre, dass man der Geometry eines vorhandenen Spatials während der Laufzeit einen neuen LOD zuweist.

Das ist ein Zitat. [Wa14b].

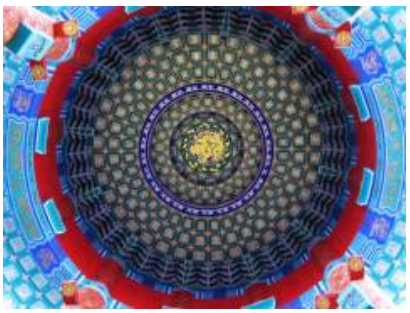
BEGRIFF A: Und so funktioniert eine Begriffsbeschreibung.

(a) Asia personas duo.

(b) Pan ma signo.



(c) Methodicamente o uno.



(d) Titulo debitas.

Abbildung 2: Beispielbilder von jMonkey.

ANHANG A

LITERATURVERZEICHNIS

[ABoo] Abel, K.; Bibel, U.: Formatierungsrichtlinien für Tagungsbände. Format-Verlag, Bonn, 2000.

[ABC01] Abraham, N.; Bibel, U.; Corleone, P.: Formatting Contributions for Proceedings. In (Glück, H.I., Hrsg.): Proc. 7th Int. Conf. on Formatting of Workshop-Proceedings, New York 1999. Noah & Sons, San Francisco, S. 46–53, 2001.

[An14] Anteil an Frauen in der Informatik. Statistics Worldwide, 2014.

[Az09] Azubi, L. et al.: Die Fußnote in LNI-Bänden. In (Glück, H.I., Hrsg.): Formatierung 2009. LNI 999, Format-Verlag, Bonn, S. 135–162, 2009.

[Ez10] Ezgarani, O.: The Magic Format - Your Way to Pretty Books. Noah & Sons, 2010.

[GI14] GI, Gesellschaft für Informatik e.V., www.gi-ev.at, Stand: 24.12.2014.

[Gl09] Glück, H.I.: Formatierung leicht gemacht. Formatierungsjournal 11/09, S. 23–27, 2009.

[Wa14a] Wasser, K.; Feuer, H.; Erde, R.; Licht, H.: Essenzen der Informatik. Verlag Formvoll, 2014.

[Wa14b] Wasser, K.; Feuer, H.; Erde, R.; Licht, H.: Ganz neue Essenzen der Informatik im selben Jahr. Format-Verlag, 2014.

ERKLÄRUNG

Karlsruhe, 20. Februar 2017

Julian Wadephul und Florian Rottach