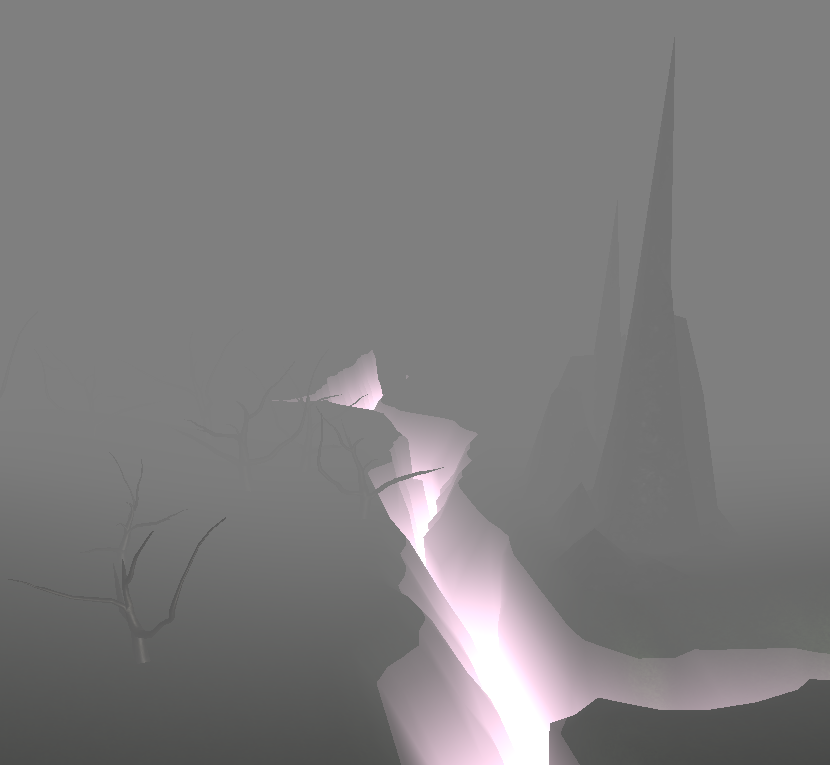


*CG2LU, SS 2010 Dokumentation zu Abgabe 3*



Andreas Gegendorfer, MatrNr.: 0301584, Kennz.: e066922, [gegendorfer@hotmail.com](mailto:gegendorfer@hotmail.com)

Nicholas Stifter, MatrNr.: 0306914, Kennz.: e066922, [nostifter@hotmail.com](mailto:nostifter@hotmail.com)

**Beschreibung:**

Wir haben uns mit dem Spiel „Sweet Dreams“ das Ziel gesetzt, grundlegende Methoden der Spiele und Grafikprogrammierung zu erlernen. Hierbei war es uns wichtig nicht durch das Szenario bei der Implementierung von Effekten und Inhalten stark eingeschränkt zu werden. Daraus resultierte ein hoher Aufwand für die Implementierung des Gameplays.

Weiteres haben wir komplett auf Open- Source Lösungen bei der Entwicklung gesetzt.

Anstelle der vorgeschlagenen Entwicklungsumgebung Visual Studio haben wir Eclipse CDT mit MinGW als Compiler verwendet. Als Modellierungsprogramm wurde Blender, als Bildbearbeitungsprogramm Gimp und als Physikengine Bullet verwendet. Die Soundausgabe erfolgt über OpenAL Die Motivation hierfür ist die Möglichkeit lizenzfrei Spiele entwickeln zu können.

Es folgt eine Aufzählung der implementierten Anforderungen.

**Gameplay:**

Sweet Dreams stellt eine Alptraumwelt dar, aus der es zu entkommen gilt. Unser Ziel war es, ein kleines „Adventure“ Spiel zu implementieren in dem man verschiedene Rätsel lösen muss um aus der Alptraumwelt wieder in das heimische Kinderzimmer zu gelangen. Leider dauerte die Implementierung diverser Effekte und technischer Elemente sehr lange und so hat sich unser Spielkonzept auf das Sammeln gewisser Objekte reduziert die einem das Entkommen letztendlich ermöglichen.

**Nichttriviale Objekte:**

Eines unserer Anliegen war es einen sinnvollen Workflow bei der Erstellung der Levels zu erhalten. Wir haben uns deshalb für die Verwendung des Collada Formats entschieden. Hierbei werden nicht nur einzelne Objekte, sondern ganze Szenen samt Physik über eine Collada Datei importiert.

Anfangs haben wir uns hierfür einen eigenen Collada importer basierend auf Collada DOM programmiert, der die Objektgeometrien exportieren konnte. Dieser wurde auch für die zweite Abgabe verwendet. Aus Komplexitäts- und Zeitgründen mussten wir uns jedoch leider von einer kompletten selbstständigen Implementierung des Collada Importers verabschieden. Konkret wäre eine sinnvolle Bearbeitung und Importierung der Materialien zu aufwändig gewesen. Stattdessen haben wir uns bei dieser Abgabe für ASSIMP (<http://assimp.sourceforge.net/lib_html/index.html>) als Modelloader entschieden. Um dennoch die in der Collada Datei vorhandenen Physikeigenschaften verwenden zu können muss die Collada Datei nochmals mit einem, eigens aus dem von Bullet bereitgestellten ColladaLoader abgeleiteteten, Importer, importiert werden.

Es ist so jedoch möglich eine komplette Szene inklusive Physik aus Blender zu exportieren und diese in unsere Engine zu importieren. Leider sind uns hier einige Limitierungen sowohl bei Blender, als auch ASSIMP aufgefallen. Diverse Materialeigenschaften können über den in Blender verwendeten Collada Exporter nur unzulänglich exportiert werden, oder werden von ASSIMP nicht richtig erkannt. Werden zu viele Materialen gleichzeitig exportiert, kann es bei ASSIMP zu Problemen kommen. Limitierungen bei dem Komplexitätsgrad der Modelle oder Szenen sind uns nicht aufgefallen.

Unsere Welt besteht aus mehr als 500.000 Dreiecken. Das Exportieren solch großer Dateien stellt ein echtes Problem für Blender dar. Für das Abgabelevel brauchte der Exporter ca. 7 Stunden. Eigentlich sieht unser Endlevel noch ein wenig anders aus, als das dass im Moment im Spiel zu sehen ist: Die Gegenstände, die man einsammeln muss, sind Sachen aus einem Kinderzimmer (Ball, Teddy, Spielzeugauto, …). Leider stürzte der Blender-Exporter bei dieser Datei nach etwa 5 Stunden ab und so fehlte nun die Zeit einen neuen Versuch zu wagen.

Die Modelle werden trianguliert aus Blender exportiert, ASSIMP besitzt ebenfalls einen Triangulationsmodus den wir ebenso aktiviert haben. Sämtliche Modelle wurden selbst erstellt.

**Animierte Objekte:**

Wir verwenden hierarchische keyframe Animationen die framerate-unabhängig sind. Es wurde hierbei die aus dem ASSIMP-viewer integrierte Animationsklasse modifiziert so dass sie eine Transformationsmatrix für jedes GameModel der Animation für den jeweiligen Frame erzeugt (SceneAnimator). Animationen werden als separate Collada Dateien eingelesen und in einem Wrapper (AnimatedObject) verpackt. Momentan werden die Physikeigenschaften für solche Objekte nicht importiert. Skelettanimationen werden ebenso importiert, eine Implementierung ist sich aber zeitlich nicht mehr ausgegangen.

Weiteres besitzt jedes Objekt eine Liste an GameModelMoves. Diese Moves liefern ebenso eine Transformationsmatrix zurück und können auch auf „normale“ Modelle angewandt werden. Über Moves sind prinzipiell sehr komplexe Bewegungsabläufe definierbar. Jegliche bewegten Objekte der zweiten Abgabe wurden über solche Moves gesteuert. Der animierte Hund dieser Abgabe wird über Keyframes animiert, verfolgt den Spieler aber mit einem „PersueMove„ der auch auf jegliche Objekte der Spielwelt anwendbar wäre.

Die Moves-Liste löscht selbstständig fertige Moves und aktualisiert jeden Move mit der neuen Systemzeit. Moves können wiederum Moves beinhalten was beispielsweise die Klasse ComplexMove realisiert. Die von uns implementierten Moves sind framerate unabhängig programmiert worden.

Die potentiellen Möglichkeiten dieses Systems wurden nicht ausgeschöpft da uns schlichtweg die Zeit ausgegangen ist.

**Transparenz-Effekte:**

Die implementierten Transparenz-Effekte erlauben es Modelle teilweise durchsichtig anzuzeigen. Jedes GameModel kann hierbei separat bestimmen, ob eine Überprüfung des Alphakanals durchgeführt wird. Diesen Effekt kann man bei dem wehenden Weizen in unserer Welt sehen und mittels Debug-Ausgabe an und abschalten. Schattenberechnungen werden ebenfalls durch den Alphawert beeinflusst und werfen so realistische Schatten und keine einfachen Flächen.

**Experimentieren mit OpenGL:**

In Sweet Dreams haben wir darauf geachtet ausschließlich im OpenGL 3.2 Kontext zu programmieren. Wir verwenden für alle GameModels VBOs, wobei diese in GameAssets erstellt werden. Diese weitere Unterteilung der GameModels rührt von den Eigenheiten des ASSIMP Importers her und erlaubt so mehrere Texturen und Materialien per Modell. Ein GameModel besteht demnach aus mehreren GameAssets die jeweils ihre eigene Geometrie und Materialien besitzen und ihre eigenen VBOs verwalten. Jedes GameAsset und jedes GameModel besitzt prinzipiell die Fähigkeit sich selbst in den Backbuffer zu zeichnen und zum Rendern einer Szene wird ein draw() Befehl an alle zu zeichnenden Objekte gegeben.

Zusätzlich verwenden wir FBOs um Shadowmapping und den postprocessing Effekt „Bloom“ zu realisieren.

Momentan gibt es leider massive Probleme mit dem Mipmapping, welches uns schon bei der 2. Abgabe Sorgen bereitet hat. Wir entwickeln beide mit ATI Grafikkarten und haben auch im näheren Umfeld keinen Zugang zu Nvidia Grafikkarten. Auf unseren Rechnern funktioniert der Befehl glGenerateMipmaps anstandslos, dieser hat aber auf dem Abgaberechner zu extremen Problemen geführt.

**Framerate**

Es wird eine durchschnittliche Framerate über 60 Frames hinweg ausgegeben.

**Effekte**

*Normal Mapping:*

Wir haben normal mapping auf einige Oberflächen angewandt (z.B. Boden in der ersten Szene).

*Beleuchtung/Schatten:*

Sweet Dreams besitzt Shadow Mapping mit self-shadows. Die “Lichtquelle” ist direktional. Die Implementierung lehnt sich stark an die in der CG2LU vorgestellten Methodik an. Um Artefakte zu vermeiden verwenden wir Front Face culling und überprüfen zusätzlich im Shader die Ausrichtung der Faces in Relation zur Lichtquelle.

*Screen Space-Effekte:*

Wir verwenden einen “Bloom” Effekt der mittels FBOs implementiert wurde. Der Effekt ist gut im Sandwüsten-Level und beim Weizenfeld beobachtbar.

*Andere Effekte:*

Weizen:

In unserem Spiel haben wir „wehenden„ Weizen implementiert. Hierbei werden im Shader ausgehend von der Ausrichtung der Texturkoordinaten die „oberen“ Vertices mittels einer Cosinusfunktion verschoben. Eine Zufallskomponente verhindert ein synchrones Wehen.

Nebel:

Ein weiterer Effekt, der die düstere Atmosphäre unterstreicht wurde Nebel implementiert. Dieser verändert sich immer anhand der Entfernung zum Spieler.

Bodennebel:

Ähnlich dem normalen Nebel, nur dass er sich nicht durch die Distanz zum Spieler verändert.

Verwendete Werkzeuge:

Als Entwicklungsumgebung haben wir Eclipse CDT mit MinGW verwendet. Die Modelle wurden in Blender erstellt und exportiert. Als Windowmanager kommt GLFW zum Einsatz. Unser PhysicsLoader basiert auf einem Collada Converter aus den Bullet Demos. Den Import der Geometrie und Materialeigenschaften übernimmt ASSIMP wobei wir die importierte Szene parsen und hauptsächlich in unsere eigenen Datenstrukturen überführen (bis auf Bones und einige Animationsstrukturen). Das Laden der Texturen wird mittels DevIL vorgenommen und in einer globalen <map> verwaltet. Als Physikengine verwenden wir Bullet. Sound wird mittels OpenAL realisiert.

**Bewegung in der Welt:**

Ein wichtiger Hinweis zuerst: Die Ladezeiten zwischen den Levels können beträchtlich (bis zu mehreren Minuten) sein. Das liegt hauptsächlich an den detaillierten Modellen, großen Texturen und – wesentlich - an den Berechnungen der „Bounding Boxes“ durch die Physik-Engine.

Das erste Level ist lediglich ein Intro. Zum Wechseln ins nächste Level: bitte ENTER drücken.

Im zweiten Level findet man sich im Kinderzimmer wieder und man sieht einige Gegenstände, die man in den darauffolgenden Levels suchen muss (Teddy, Ball, Spielzeugauto, Hund). Nun „schläft“ man ein und findet sich im Alptraum wieder (zum Szenenwechseln wiederum ENTER drücken).

Das dritte Level entspricht Großteils der Welt aus der Zwischenabgabe (diesmal mit korrekter Physikberechnung, Schatten, usw.). Man muss den Teddy finden (sollte nicht allzu schwierig sein - er befindet sich in der Baumgruppe). Hat man ihn gefunden erscheint eine Brücke und man kann den Graben gefahrlos überwinden. Auf der anderen Seite wird man von einem wahnsinnig gefährlichen (animierten) Hund verfolgt. Frisst er den Spieler, so wird man zum Anfang teleportiert. Es gilt die Treppe und weiter die draufstehende Tür zu erreichen. Sobald man sich auf der Treppe befindet ist die Gefahr durch den Hund gebannt.

Sobald man durch die vermeintliche Tür ins Kinderzimmer schreitet wacht man noch nicht auf, sondern kommt in ein weiteres Level.

Hier gilt es vier verschiedene Gegenstände zu finden. Einen Teddy, ein Spielzeugauto, einen Ball und einen Hund.

3 von den 4 Sachen befinden sich in den Ecken der Welt. Ein Gegenstand auf dem Leuchtturm, einer auf einem Berg (erreichbar durch eine rieeesen Treppe) und einer in einem weiteren Eck am Boden (in der Nähe der Pyramide, in einer Baumgruppe). Der Ball befindet sich in der Pyramide. Hinweise zum Weg durch das Labyrinth finden sich auf Zetteln die am Boden der Pyramide liegen. In diesen ist jeweils eine Zahl zu finden. Welche Zahl welcher Richtung entspricht kann man auf einem Zettel, der sich am Eingang der Pyramide befindet, erkennen. Der richtige Weg in die Kammer ist: RECHTS, RECHTS, LINKS, GERADE, RECHTS (Tip: wenn man nicht mehr rausfindet einfach „R“ drücken).

Jedes Mal wenn man einen Gegenstand einsammelt ändert sich die Farbe und Beleuchtung der Welt. Hat man alle vier Gegenstände eingesammelt so fängt die Welt an zu strahlen und eine Tür erscheint auf einem Hügel vor dem Eingang zur Pyramide.

Geht man nun durch diese Tür findet man sich immer noch nicht im Kinderzimmer wieder, aber man ist dem Ziel schon recht nahe.

Der Traum verändert sich im letzten Level zum Positiven und es bleibt nur noch eins zu tun: Durch das wogende Weizenfeld in Richtung der Sonne und somit der Tür zum Zimmer, zu gehen.

Jetzt endlich „wacht man auf“ und findet sich in dem Zimmer wieder in dem Alles begonnen hat.

Gratulation!

**Tastaturbelegung:**

W – vorwärts

W + SHIFT – vorwärts rennen

S - zurück

A - links

D - rechts

R – Spieler an den Anfang zurücksetzen

SPACE „springen“ (funktioniert nicht korrekt – es ist möglich beliebig hoch zu springen. Hier fehlt eine Implementation seitens der Bullet Engine), debug camera rauf

Pfeil runter – debug camera nach unten

F2 - framerate anzeigen lassen (Konsole)

F3 – wireframe modus

F4 – Texturen umschalten (linear/nearest) - kann auf dem Abgaberechner zum Absturz führen

F7 – bloom mode

F9 – alpha modes: on , off, default

F10 - shadow modes, on , off, default

F12 – debug camera

L – Schatten aus Kamera Sicht setzen

ENTER – Nächste Szene

ESC – Szene zurück bzw. Spiel beenden