TU WIEN

186.124 LU Computergraphik 2/3 SS 2006

CUBOPHOBIA

Spiele-Dokumentation

Heinrich Fink
0425503 532 heinrich.fink@chello.at

Anton Frühstück
0126452 932 a.fruehstueck@gmx.at

Inhaltsverzeichnis

1. Anforderungsanalyse

Im Folgenden wird auf die Spiel-Anforderungen von Seiten der Übungsleitung und die konkrete Implementierung in unserem Spiel eingegangen.

1.1. Spieldesign

Ein wichtiger Teil eines jeden Computerspieles ist das *Gameplay* und aus diesem Grund wurde ein großes Augenmerk auf diese Anforderung gelegt.

Nachdem der Spieler mit *Cubophobia* über die Steuerung des Balles interagiert, haben wir viel Zeit in das Steuerungs- und Kamera-Tuning gesteckt.

Die Kamera bewegt sich in einem Orbit um den Ball. Die Orientierung des Orbits kann mit 6 Freiheitsgraden verändert werden. Sämtliche Kamerabewegungen werden interpoliert und resultieren in flüssigen Bewegungen. Es wurde darauf geachtet, dass die Kamera von festen Objekten abgestoßen wird und dadurch keine *Occlusions* auftreten. Der Abstand zum Ball wird bei Kontakt mit anderen Objekten dynamisch verändert.

Um die Steuerung des Balles zu vereinfachen und an die verschiedenen Situationen im Level anzupassen wurde zwei verschieden Bewegungsarten implementiert: normale Bewegung und ein *Running Mode* bei dem der Ball stärker auf die Benutzereingaben reagiert. Insgesamt muss zur Steuerung angemerkt werden, dass unser Spiel sicherlich einige Gewöhnungszeit benötigt. Eine Vereinfachung der Steuerung würde aber einerseits tief in unser Spieledesign eingreifen und wäre vor allem auch gar nicht von uns gewünscht, weil es unseren physikalischen Spiel-Prinzipien widersprechen würde (ein rollender Ball reagiert nicht sofort).

Speziell die Physik-Engine hat uns erlaubt verschiedene Effekte in das Spiel zu programmieren, die entscheidend für den „Spiel-Spaß“ sind. So wurden eine Vielzahl an Triggern eingebaut, welche die Objekte im Spiel beschleunigen, Sounds auslösen, andere Trigger aktivieren oder deaktivieren oder die Schwerkraft ändern können. Mit diesen Möglichkeiten konnten wir Level designen, die nicht nur unsere Effekte präsentieren, sondern vor allem Spaß machen!

2. Renderengine-Features

2.1. Omnidirectional Shadows

Schatten sind ein sehr wichtiges Schlüsselelement realistischer Echtzeitgrafik. Wir haben uns für eine Variante von Shadow Maps entschieden. Virtual Shadow Depth Cube Textures

Diese Implementiereung ermöglich das schnelle Rendern in GL_DEPTH_COMPONENT24 – Texturformat, und ist daher für ein Omnidirektionales Shadow-Modell relativ performant.

Die Implementierung haben wir zum größten Teil aus dem Artikel „Efficient Omnidirectional Shadow Maps, G. King, W. Newhall, Shader X3“ übernommen.

Während wir zwar Shadow-Cube-Faces cullen, kann in Situationen, wo 5 oder alle 6 Cube-Seiten gerendert werden müssen, ein Vertex-Bottleneck entstehen, was auf die nicht implementierte Object-Culling Methode zurückzuführen ist. Hierbei hätte simples Light Frustum Culling nicht gereicht.

Zudem wird 4 tap-dithered PCF-Filter (siehe GPU Gems) eingesetzt.

Beschleunigende Features seitens CPU

- Vertex Buffer Objects
- View Frustum Culling
- Shadow Cube Face Culling

2.2. Depth Of Field

Um einen speziellen räumlichen Eindruck zu gewinnen, haben wir Depth of Field als zusätzlichen Multi-Pass Effekt eingebaut (siehe ShaderX3 , Advanced Depth of Field, T. Scheuermann). Zusätzlich wird über Raycasten mit PhysX ein Autofocus gesetzt, der die Parameter für die Shader dynamisch setzt. Als Spieler erhält man so einen ungewöhnlich starken Eindruck von Raum-Dimensionen.

Als zusätzlicher Gewinn können wir mit dem DOF – Effekt nicht nur die Schattenkanten zusätzlich zu dem PCF-Filter weicher erscheinen lassen, sondern auch Fehler im Filtern über Cube-Face-Edges einfach verschwimmen lassen.

2.3. Per Pixel Lighting, Normal Mapping

Jedes Objekt wird mit vollständigem Per Pixel Lighting und Normal Mapping beleuchtet. Für die Berechnung der TBN-Koordinaten verwenden wir NVMeshmender.

2.4. Culling Varianten

Für die Erstellung der Shadow Cube Maps wird ein Cube-Face-Culling ALgorithmus angewandt, der auch auf Object-Culling für Cube-Maps zu erweitern wäre (woraus der momentane Vertex Bottleneck resultiert, da pro Cubemap alle Vertices die Pipeline durchlaufen, im schlimmsten Fall bis zu 6 mal.

Für den Renderpass wird ein ViewFrustum-Culling Algorithmus angewandt.

2.5. Referenzen und Links

Folgende Papers wurden für die Implementierung der Effekt einbezogen, bzw. implementiert:

- http://www.ati.com/developer/gdc/Scheuermann_DepthOfField.pdf
- ShaderX3:

  Advanced Depth of Field, T. Scheuermann

  Efficient Omnidirectional Shadow Maps, G. King
- GPU Gems 1:

  Shadow Map Antialiasing, M. Bunnell, F. Pellacini

  Omnidirectional Shadow Mapping, P. Gerasimov

3. Besonderheiten

Trennung zwischen Code und Content

Unser Spiel zeichnet sich dadurch aus, dass sämtlich Content unabhängig vom eigentlichen Programmcode ist! Alle Objekteigenschaften (Physik, Licht- und Materialeigenschaften, …) und die gesamte Gamelogic (Schwerkraftänderung, Lichtveränderung, Beschleunigungen, Level-Ende, …) sind in FBX gescripted und können jederzeit in Maya verändert werden, ohne den Programmcode ändern zu müssen. Das ermöglichte uns ein sehr freies und einfaches Leveldesign.

Game-Engine:

.) Physics-Simulation der ganzen Szene und Kollisionen mit beliebigen Meshes

.) Arbiträre Meshes werden 100% aus Maya übernommen

.) Intuitive Steuerung durch WASD

Graphical Features:

- Depth Of Field w. Autofocus
- Omnidirectional Shadows
- Normal Mapping in TBN-Space
- Per Pixel Lighting
- Texturing w. Trilinear Filters (Mip Maps)
- Vertex Buffers (switchable to Vertex Arrays, oder Immediate Mode)
- Framebuffer Objects (for multipasses)
- CGfx - Shadernetwork

4. Zusatztools

Im folgenden Abschnitt werden alle externen Libraries und Zusatztools aufgelistet und deren Funktion in unserem Spiel erklärt.

4.1. Cg

Die Renderengine basiert komplett auf NVIDIAs CGFX. Obwohl für die 2. Abgabe nur ein Shader die gesamte Beleuchtung / Texturierung übernimmt, ermöglicht uns das eine hervorragende Basis um für die 3. Abgabe fortgeschrittenere Effekt wie Normal Mapping / Environment Mapping uvm zu implementieren. Wir verwenden das neueste 1.4 Stable-Toolkit.

Praktischerweise werden nahezu alle OpenGL - State von CGFX verwaltet und werden pro pass in jeder Technique und jedem Effekt einzeln belegt. So können je nach Wunsch übersichtlich die State - Variablen geändert werden.

4.2. NV-DDS Textur Loader

Eine kleine Library die dds files liest und für das Uploading als OpenGL-Texture Object vorbereitet -> In Kombination NVIDIAs DDS Plugin für Photoshop können sehr schnell komprimierte Texturen mit vorher erstellten (und auch in PS individuell veränderbaren!) MipMaps erstellt werden.

4.3. PhysX

Sämtliche *Collision Detection*, Objektbewegungen und Physik-Eigenschaften in unserem Spiel werden von der Physik-Engine von Ageia (siehe http://www.ageia.com/physx) berechnet.

4.4. FBX

Diese Library ermöglicht eine reibungslose Kombination mit Autodesk Maya. Maya exportiert die Szene in ein *.fbx - Format, das von einem Loader des SDKs eingelesen wird. Wir importieren Lights, Meshes (die allerdings noch in Vertex-Buffer - kompatible Arrays umgeschrieben werden), UV-Koordinaten, Normals, Materials und deren Farb-Texturen. Diese Informationen werden zwar vom FBX SDK angeboten, müssen aber natürlich für unsere Render Engine angepasst werden.

Zudem ist es möglich in Maya eigen definierte Attribute pro Objekt zu erstellen, die dann diverse andere Parameter pro Objekt erlauben. Diese Attribute und das entsprechende Programmierdesign erlaubt uns, sämtliche Spiel-Logik in Maya zu scripten ohne im Quelltext etwas ändern zu müssen. Überdies hinaus werden die zum Beispiel auch die PhysX-Eigenschaften über diese Custom Attributes gesteuert.

4.5. fmod

Das *fmod music & sound effects system* wurde in unserem Spiel verwendet, um Sounddateien laden und abspielen zu können.

5. Game-Anleitung

Die folgenden Kapitel dienen dazu, einerseits Cubophobia starten zu können, andererseits mit der Steuerung vertraut zu werden und dadurch die verschiedenen Level erfolgreich absolvieren zu können.

5.1. Anleitung zum Spielstart

Nach starten des Rechners und entpacken des rar-Archives, findet man im Ordner `bin` die Start-Datei des Spieles: `Cubophobia.exe`. Einstellungen bezüglich der Auflösung können in der Datei `resources\gamesettings.ini` in der Zeile 17 nach *glutParam* getätigt werden.

Empfohlen wird eine Auflösung von 1024x768.

5.2. Steuerung / Tastenbelegung, Tips

Im Menü:

w / Pfeil nach oben	vorige Auswahl
s / Pfeil nach unten	nächste Auswahl
Leertaste / Enter	Auswahl auswählen

Im Spiel:

w	vorwärts bewegen
s	rückwärts bewegen
a	nach links bewegen
d	nach rechts bewegen
c	bremsen
<	schneller bewegen
Leertaste	springen
r	Schwerkraft zurücksetzen
p	Pause
+	Kamera näher zum Ball bewegen
-	Kamera weiter vom Ball wegbewegen
#	Kamera Position zurücksetzen
m	Stumm
Linke Maustaste	Boost nach vorne
Esc	Level/Spiel beenden
F1	Hilfe anzeigen/verbergen
F2	Framerate anzeigen/verbergen
F3	Wireframe Modus anzeigen/verbergen
F4	Linear/Nearest bei der Color Texture umschalten
F5	MipMaps de/aktivieren
F6	Umschalten zw. Immediate und VBO/VA
F7	VBO und VA umschalten
F8	View Frustum Culling de/aktivieren
F9	VSDCT Culling de/aktivieren
F10	Shadows an- bzw. nicht-anzeigen
F11	RenderInfo ein-/ausblenden

Steuerungstips:

Der normale Steuerungsmodus des Balles eignet sich gut, um den Ball in einer Umgebung zu bewegen, in der z.B. auf Boxen oder Brettern relativ genau gesteuert werden muss und der Ball keine hohe Geschwindigkeit erhält.

Falls der Spieler will, dass der Ball schneller beschleunigt und auch bei höherer Geschwindigkeit gut reagiert, kann die Taste '<' verwendet werden (gedrückt, in Kombination mit den wasd-Tasten).

Eine dritte Möglichkeit zur Ballsteuerung besteht darin, die wasd-Tasten bei gedrücktem 'c' zu verwenden. Dabei reagiert der Ball nur sehr wenig und kann dadurch sehr genau gesteuert werden. Das ist nützlich, wenn z.B. Hindernisse, wie die dünnen Holzäste im Medieval Cube gemeister werden müssen.

Beim Springen ist zu beachten, dass die Sprunghöhe durch mehr oder weniger langes drücken der Space-Taste variiert werden kann.

5.3. Walkthrough – Endless Cube

In diesem Level muss der Ball aus einem hochtechnischen Cube entkommen. Aber Vorsicht: es können merkwürdige Dinge passieren. Falls auf Grund der Schwerkraft-Änderungen ein weiterkommen nicht mehr möglich ist, kann der `r` Button benutzt werden, um die Schwerkraft zurückzusetzen!

Der Cube ist zu meistern, indem man die Bretter entlangfährt. Die Schwerkraft wird sich so ändern, dass immer ein Weg weitergeht.

5.4. Walkthrough – Timeless Cube

In diesem Level muss sicher der Ball durch einen mysteriösen Cube kämpfen. Das Level zeichnet sich durch eine hohen Grad an Komplexität der Objekte aus.

Walkthrough:

- Den Steinhaufen am Boden zerschmettern
- Suche die passenden Symbole für die passenden Schilder an der Wand
- Du kannst die Steine auf die Rampen bewegen, durch die sie beschleunigt werden und fast immer auch gleich die Schilder treffen
- Hierbei ist die richtige Reihenfolge von Bedeutung, die Reihenfolge ist auf einem Schild an der Wand hinter zwei großen Steinen dargestellt
- Nachdem die Steine in der richtigen Reihenfolge an die passenden Schilder katapultiert wurden, zeigt der Spot des Lichts auf eine Ecke, in der ein loser Stein (leicht bläulich gekennzeichnet) weggestoßen werden muss.
- Du kommst nun in die Secret Church, stoße gegen das Schild, das die Schwerkraft dreht.
- „Falle“ nun wieder aus der Secrect Church auf die andere Wand.
- Nun kannst du auf den Looping springen und den entlangfahren(TIP: im Looping solltest du nicht nach vor drücken, du wirst von selbst beschleunigt….
- Die nächste Hürde ist nun auf den drei Stangen (nimm am besten die linke…) entlangzurollen (TIP: drücke C und gleichzeitig eine der Richtungstasten, dann bist du im „Schleich-Modus“, den du für diese Hürde brauchst.)
- Schließlich und endlich musst du über die schwebenden Steine in den Schlund des Cube-Gottes springen. Aber Achtung, bleib nicht zu lange auf einem Stein, sonst drückst du ihn runter und kannst nicht mehr entfliehen!!
- Gratuliere, du bist aus dem Cube in die weiße Erleuchtung geflohen!

6. Programmstruktur

Main

In der *main* Methode werden nach der Überprüfen des Pfades, ein *GameSettings* Object, die *SoundEngine* und das erste *RenderObject* erstellt. Diese erste *RenderObject* repräsentiert den *MainScreen*. Nach der Initialisierung von GLUT wird mit der *glutMainLoop* das Spiel gestartet. Etwaige Fehler während des Spieles, werden gefangen und ausgegeben. Bei einem Fehler werden sämtliche erzeugte Objekte kontrolliert zerstört. Danach wird das Programm beendet.

Bei dem Laden eines neuen Levels oder des *MainScreens* werden in der *renderCallback* Methode in *main* die entsprechenden Objekte neu erzeugt.

Scene

Im *Scene* Objekt werden die Geometrie und Textur-Eigenschaften aus dem FBX-File ausgelesen und überarbeitet. Außerdem wird ein *PhysicsEngine* Objekt erzeugt und mit dem den Objekten aus dem FBX-Scene-Baum „gefüllt“. Die zurückgegeben *PhysXObjekte* werden in den entsprechenden FBX *nodes* als *UserData* abgespeichert.

GameData

Jedes Objekt im FBX-Scenegraphen enthält als *Userproperty* ein *GameData*-Objekt, welches neben einem PhysX Objekt sämtliche Objekt-bezogenen Daten speichert, die nicht aus dem FBX-Baum direkt gewonnen werden können und für das Rendering notwendig sind.

GameSettings

In der *GameSettings* Klasse wird das *.ini-File geladen und die Parameter, die aus diesem File ausgelesen werden, den anderen Objekte zugänglich gemacht. Das ermöglicht Parameteränderungen ohne Neukompilieren des Spieles.

PhysicsEngine

In der *PhysicsEngine* wird sämtliche Physik-Eigenschaften der Szene generiert. Pro Frame kann die Position und andere Eigenschaften der Objekte für das *Rendering* abgefragt werden. Als Basis-Engine wird Ageia-PhysX verwendet. Die Physik Engine läuft mit einer konstanten (vordefinierten) Framerate – im Gegensatz zur *RenderEngine*.

MemoryWriteBuffer / MemoryReadBuffer

Diese beiden Klassen werden von der *PhysicsEngine* für die Speicherverwaltung benötigt.

RenderInterface

Die abstrakte Klasse *RenderInterface* stellt eine Basis für Objekte dar, die am eigentlichen Render-Prozess teilnehmen können.

MainScreen

MainScreen enthält die Methoden zum Rendern des Eingangsbildschirmes. In diesem kann der Benutzer das Spiel beenden, sowie die Level starten.

RenderEngine

In der RenderEngine wird sämtliches *Rendering* der Szene ausgeführt. Dazu wird über das *Scene* Objekt die FBX-Szene traversiert und die PhysX-Objekte extrahiert. Über diese Objekte kann wiederum die Geometrie-Information aus dem FBX-Baum positioniert und gerendert werden.

Sämtliche Benutzereingaben werden auch in dieser Klasse abgearbeitet. Ebenso wird die Kamera in verschiedenen Methoden berechnet.

ShadowFrustum
Diese Klasse bietet die notwendigen Datenstrukturen für View Frustum (besonders auch für Frustum-Frustum-Tests, die beim VSDCT-CUlling notwendig sind)

SoundEngine
Über die *SoundEngine* Klasse wird einerseits das Sound-System initialisiert, anderseits werden hier verschiedenen Sounds abgespielt und kontrolliert.

MsgHandler
Der *MsgHandler* ist eine kleine Klasse, über die sämtliche Console-Ausgaben laufen. Damit ist es möglich, diese Ausgaben einerseits zu unterdrücken, andererseits ev. in eine andere Ausgabeform umzuleiten (z.B. File-Ausgabe).

ContactCallback / TriggerCallback
ContactCallback und *TriggerCallback* sind Callback-Klassen für die *PhysicsEngine*, die Informationen über Objekt-Kontakte bzw. die Kollisionen mit Triggern zurückliefert. In diesen Klassen werden Aktionen nach einem Objektkontakt (z.B. Abspielen eines Sounds) oder dem Auslösen eines Triggers (z.B. Änderung der Schwerkraft) ausgeführt.

FpsCounter
In der Klasse *FpsCounter* werden im Spiel die aktuellen *frames per seconds* berechnet. Darüber hinaus kann in jedem *frame* die zum letzten *frame* vergangene Zeit abgefragt werden.

7. Known Issues

Write Adress Access Violation
Auf einigen Rechnern tritt bei Exit des Spiels eine Access Violation auf. Wir wissen leider noch nicht den genauen Usprung und ob unseren Anwendung oder ein Treiber daran schuld ist.