

# Abgabe 3

## Computergraphik 2, 3

### *Teammitglieder*

---

<b>Nr.</b>	<b>Name</b>	<b>Matr-Nr.</b>	<b>E-Mail</b>
1.	Murat Sari	126198 534	<a href="mailto:murat_sari@gmx.at">murat_sari@gmx.at</a>
2.	Stefan Tomitsch	126195 534	<a href="mailto:tec@gmx.net">tec@gmx.net</a>

Gedruckt am: 20.06.2006 02:45:00

---

## Grundlegende Ogre-Features

### 1.1.) Overlays

Wir haben Overlays an folgenden Stellen verwendet:

- SceneFader (Diese Klasse kümmert sich um das weiche ein/aus -Faden einer Scene.
- CineFader (Diese Klasse Fadet 16:9 Balken ein und aus diese dienen für die zwischensequenzen).
- IntroFader (Diese Klasse ist eine allgemeine Klasse der man zur Laufzeit Materialien zuweisen kann und die dann vom einen Material zum anderen Fadet. Wurde beim Intro du Outro verwendet.)
- Crosshair(Wir blenden ein Kleines Fadenkreuz ein wenn der Player sich in der Ego-Camera befindet.)
- FPS anzeige

### 1.2.) Animation

- Spline Animation (Für diverse Cutscenes wurden Splines für die Camera fahrt verwendet die wir im Leveleditor aufgezeichnet haben).
- Entity Animation (Für ganzen Charaktere(2 Stück) und die Monster(> 20) verwenden wir Entity Animation wo wir auch Animation überblendungen durchführen.)

### 1.3.) OIS – Object oriented Input System

- InputManager (Diese Library verwenden wir für den Input wir mussten sie quasi verwenden weil wir CVS-Head von Ogre verwenden und das neu dazugekommen ist und es wurde nur ins Example Framework eingebaut welches wir nicht verwenden. Also im Ogre Core wird jetzt kein Input-System mehr drin.)

## **1.4.) CeGUI**

- InGame (Wird im Spiel verwendet um auf gestellte fragen antworten zu können und um die Energie anzeige darzustellen.)
- Menu (Hier wird es verwendet dem User einstellungs möglichkeiten zu bieten.)
- Leveleditor (Im Level editor wird stark gebraucht davon gemacht.)

## **1.5.) RaySceneQuery**

- Collision Detection (Wir mussten leider in letzter Sekunde ein eigens Kollisions- System Programmieren welches Ray strahlen verwendet.)
- Leveleditor (Für das Picking und Dragging von Objekten.)
- Bullets (Für die Feuerbälle im Spiel.)

---

# **Ogre Spezifische Effekten**

## **1.6.) Fresnel Wasser**

- Meer (Als Meer der die Insel umgibt.)
- Teich (Ein kleiner Teich bei den Spinnen.)

## **1.7.) Schatten**

- InGame (Genau wird TEXTURE\_MODULATIVE verwendet.)

## **1.8.) Partikelsysteme**

- Bullets (Also die Feuerbälle bestehen aus einem Partikelsystem.)
- Leveleditor (Mittels den Leveleditor kann man diverse andere Partikelsysteme ins Spiel bringen ohne neu zu kompilieren.)

## **1.9.) Compositor**

- Damage (Hier wird ein Compoitor Effekt verwendet wenn der Spieler schaden erleidet.)
- PostfilterManager (Mittels unserem Manager kann man Effekte wie Bloom Tonemapping, Motionblur und diverse andere Effekte ein und ausschalten.)

## **1.10.) Eigener Effekt**

Als eigener Effekt wurde Specular Bumpmapping unter der Verwendung einer Direktionalen Lichtquelle implementiert mittels CG verwendet wird zurzeit nur auf einem Felsen.

---

# **Performance-Optimierung**

## **1.11.) Static Geometry**

Im Leveleditor kann man für diverse meshes (die man rein platziert hat) Typen zuweisen wenn ein Mesh den Typ „Static“ bekommt werden alle Meshes mit dem Selben Material(die natürlich auch Static sein müssen) in einen StaticGeometry Batch zusammen zugefasst.

## **1.12.) Level of detail**

- Terrain (Das verwendet Geomipmapping LOD welches nicht wir implementiert haben.)
- Material Skripte (In diversen Material sind LOD fallbacks drinnen aber wurde nicht sooft verwendet.)

## **1.13.) Billboards**

Wurde nur einmal verwendet für die Sonne im Skydome.

## Besonderheiten

### 1.14.) In game Leveleditor

Der Leveleditor ist quasi nur die eine GUI die unsern selbst geschriebenen Sceneloader und seine Klassen verwendet. Unser Sceneloader kann eine Scene aus einem XML File laden (zum Parsen des XML Files haben wir Expat verwendet). Wir haben es so gehandelt das wir ein Interface XMLDescription erzeugt haben welche die grundlegenden Features für so ein Gameobject bietet(z.B.: addToOgre,removeFromOgre...). Danach haben wir eine reihe von Descriptions programmiert die wir im LevelEditor zu Verfügung haben.

- EntityDescription (Hat die Möglichkeit eine Ogre::Entity zu kapseln und kann diveres Attribute ins XML speichern.)
- ShadowDescription (Jedes Level kann im Leveleditor sein schatten verhalten bestimmen und speichern.)
- LightDescription (Hier kann man Lichtquellen in die Scene setzten und diverse Werte ändern. Kann zu einer EntityDescription hinzugefügt werden.)
- ParticelDescription (Hier kann man einen Particel effekt aus der GUI wählen und es ins Game rein setzten und diverse Werte ändern. Kann zu einer EntityDescription hinzugefügt werden.)
- TriggerDescription (Hier haben wir die Möglichkeit visuell Trigger zu setzten und deren Wirkungsradius zu bestimmen man kann auch Trigger zu einer EntityDescription attachen wo man dann beweglich Trigger hat. Und man kann Trigger zu einem Entitytype hinzufügen. (So haben wir die Möglichkeit so viele Gegner wie wir wollen ins game zu setzten uns müssen nicht neu Kompilieren weil die „AI“ automatisch zugewiesen wird.))
- SoundDescription (Bietet die Möglichkeit Sounds ins Game zusetzten (3D/2D sounds) und zu EntityDescriptions zu attachen. Man hat auch diverse Möglichkeiten Volume und Min/Max Distance zu verändern.)
- ExternDescription (Hier kann man in einem XML File ein anderes referenzieren. Man muss bedenken das die Externen Level Files erst geladen wird wenn das Aktuelle fertig geparst wurde und im anderem XML File ExternDescriptions ignoriert da es sonst zu Rekursionen kommen könnte.)

Der oben beschriebene SceneLoader wird verwendet um das Game zu laden im Playstate.

### 1.15.) Init. Von Ogre

Hier übernimmt alles der OgreDirector der von OgreSingleton abgeleitet wurde wir haben ein eigenes config File und verwenden nicht den „Buildin loading Dialog“ alles wird von hand gemacht.

### 1.16.) GameStateManager

Also wir haben einen GameStateManager geschrieben der die einzelnen Gamestates verwaltet (z.B.: MenuState, PlayState, CreditState, LeveleditorState) diese GameStates werden von der Klasse GameState abgeleitet durch dieses System wäre es möglich einzelne GameStates als Plugins auszulagern leider hat und dafür die Zeit gefehlt.

### 1.17.) ScriptManager

Wir hatten geplant die ganze Logik via Lua scripts zu Programmieren doch angesichts des Zeitmangels sind die Scripte als C++ Klassen „hard gecodet „ und der ScriptManager verwaltet alle hinzugefügten Scripts und updatet diese.

---

## Walkthrough

### 1.18.) Allgemein

Wenn man das Spiel Startet sieht man das Hauptmenu mittels der ESC taste kann man das Spiel wieder verlassen.

Hier kann man diverse Grafik Optionen einstellen man sollte bedenken wenn man auf OK klickt wird das Programm herunter gefahren da ich anfangs versuch hatte Ogre neu zustarten und das nicht wirklich funktioniert hat ist der alte code noch drin der er abschaltet☺.

Wenn man im PlayState ist dann kann man mit den WASD tasten den Spieler steuern und mit der maus die Kamera bewegen für genauere Beschreibung siehe oben „Implementierung der Steuerung“ mittels der Taste TAB-Taste wechselt man in den EGO Modus.

Wenn man Space gedrückt hält kann man Laufen. Mittels der Maustaste 1 kann man Feuerbälle werfen.

Auflistung der anderen Funktionen:

- Taste 1 wird verwendet um die FPS anzeige ein und auszublenden.
- Taste 2 WireFrame Modus ein/aus
- Taste 3 Screenshot

### **1.19.) Durchspielen**

Man muss zuerst mit dem alten Mann reden dann muss man die Spinne töten danach wieder mit dem alten Mann reden dann die Zombies Töten und dann an die andere Bucht begeben.