

# TOWER

# NETWORK

## Dokumentation Abgabe 3



Manuel Metzger, e0727354, [e0727354@student.tuwien.ac.at](mailto:e0727354@student.tuwien.ac.at)

David Körner, e0725733, [e0725733@student.tuwien.ac.at](mailto:e0725733@student.tuwien.ac.at)

Computergraphik 2 LU



TECHNISCHE  
UNIVERSITÄT  
WIEN  
Vienna University of Technology

## Projektmitglieder

Manuel Metzger, E033 532, e0727354, [e0727354@student.tuwien.ac.at](mailto:e0727354@student.tuwien.ac.at)

David Körner, E033 532, e0725733, [e0725733@student.tuwien.ac.at](mailto:e0725733@student.tuwien.ac.at)

## Allgemeines

Bei der 3. Abgabe wurde großer Wert auf die Implementierung der Effekte und die Erweiterung des Gameplays gelegt. Es wurde ein Glow-Effekt und Shadow Mapping implementiert. Zur Erweiterung des Gameplays wurde eine Superwaffe, sowie mehrere Gegner- und Turmtypen hinzugefügt. Weitere Kleinigkeiten im Gameplay sind ebenfalls hinzugefügt worden, Upgrade, Verkauf, Selektion sowie primitives Interface. Aufgrund des Zeitmangels wurde jedoch weniger Wert auf Performance und gutes Balancing gelegt.

Aufbauend auf der 2. Abgabe wurde diese Abgabe auf technischer Ebene durch die Effekte und der Ebene des Gameplays durch gesteigerte Interaktion mit dem User verbessert.

## Features (Zusammenfassung)

- RAIL-Konzept, "echter" ResourceManager bei MD2 Models
- Vollständiges Kamerasystem
- Level mit skalierbarer Größe (Map + Datenpakete)
- Bewegte Gegner mit simpler Wegfindung
- Tower platzieren mit Maus
- 3 Towertypen
- 4 Gegnertypen
- 1 Superwaffe
- Tower auswählen, verkaufen und upgraden
- Phong Beleuchtungsmodell
- TextureObjects für Texturierung
- Einfache Texturen für alle Objekte
- Simpler Shader, GLOW Shader, Shadow Mapping Shader
- WIN und LOSE Situationen
- Materialeigenschaften für alle Objekte
- Intuitive Steuerung
- Framerateunabhängige Gamelogik
- MD2 Model Loader mit Animationen und 2-Part Models

## Gameplay

Wie schon bei der 2ten Abgabe ist unser Spiel auch hier vollständig spielbar. In dieser Abgabe ist der vollständige erste Level mit verschiedenen Gegnern und Towertypen enthalten. Ausserdem die Superwaffe, die nach einer bestimmten Zeit freigeschalten wird.

## Nichttriviale Objekte

Unser Tower Modell ist ein nichttriviales Objekt (MD2 Model mit Texturierung) und wird durch unseren MD2 Model Loader geladen. Es wurde in Milkshape3d 1.8.4 erstellt und dann mit dem Quake2 MD2 Exporter exportiert (der in Milkshape allerdings keine Vertices ohne Bone erfordert sonst werden keine Animationen exportiert).

## Animierte Objekte

Unser Tower ist ein MD2 Model mit Animation und diese funktioniert mit Keyframes zwischen denen mit linearer Interpolation gearbeitet wird. Die ganze Animation ist 30 Frames lang und enthält jede 5 Frames einen Keyframe. Unser Implementierung vom MD2 Model Format sieht eine statische Liste von Animationen vor die jede 30 Frames hat.

## Beschleunigung der Sichtbarkeitsberechnung

Für die Beschleunigung der der Sichtbarkeit wurden für alle SceneObjects Bounding-Boxen berechnet. Die Bounding-Boxen werden direkt aus den Daten des Models oder der hardgecodeten Vertices errechnet. View-Frustum-Culling wird für alle Objekte durchgeführt und kann per Tastendruck ein- und ausgeschaltet werden. Die Steigerung der Performance durch das Culling lässt sich beim Anzeigen des FPS-Counters erkennen.

## Transparenz-Effekte

Transparenz wird im Spiel nur an einer Stelle im Spiel verwendet. Beim Glow-Effekt wird über die normal gerenderte Szene, die Glow-Textur mit der Glow-Szene geblendet. Dies wird durch glBlend und gleiche Beteiligung der beiden Komponenten bewerkstelligt.

## Experimentieren mit OpenGL

Entsprechend der Anforderungen wurden im Spiel VBOs implementiert die sich über den entsprechenden Hotkey aktivieren lassen. VBOs werden für die Darstellung der Map, Files und Enemies eingesetzt. Die Auswirkungen auf die Performance sind durch das aktivieren der FPS-Anzeige ersichtlich.

Im Zuge der Implementierung der Effekte haben wir intensiv mit FBOs experimentiert und setzen diese auch aktiv im Spiel ein. So wird die Szene mit Glow-Texturen in eine eigene Textur gerendert und auch das Ergebnis der Glow-Szene sowie die Tiefenwerte für das Shadow-Mapping werden mit einer entsprechenden Textur und einem FBO zwischengespeichert.

Desweiteren lässt sich die Texturqualität und auch die Mip-Mapping-Qualität durch die entsprechenden Hotkeys verändern. Veränderungen der Darstellung sind sofort sichtbar wenn die Qualität verändert wird.

Es ist wie bereits erwähnt möglich eine FPS-Anzeige zu aktivieren, aber auch in den Wireframe-Modus zu wechseln. Auch View-Frustum-Culling kann per Knopfdruck aktiviert und deaktiviert werden. Die Hotkeys für diese Funktionalitäten sind entsprechend der Angabe implementiert und ihre Funktion wird auch durch eine kurze Meldung links oben am Bildschirm angezeigt.

## Restriktionen

Es wurde gegen keine der genannten Restriktionen von der Computergraphik 2 LU Website verstoßen. Jeglicher Code für Anzeige und Rendering wurde selbst geschrieben. Wir verwenden weder fertige Sourcecodes / Libraries zum Laden von 3D Modellen oder eine Game Engine.

## Effekte

### Shadow Mapping

Dieser Effekt wurde im Prinzip nach den CG2 Tutoriumsfolien implementiert. Es ist ein sogenannter Zweipass Algorithmus und wird in 2 Schritten durchgeführt. Im First Pass werden die Tiefenwerte in ein FBO (also offscreen) gerendert (aus Sicht des Lichtes) und beim 2nd Pass werden diese Tiefenwerte in der Shadow Mapping Textur verglichen. Wenn der Tiefenwert zur Lichtquelle größer als der Tiefenwert in der Shadow Map ist dann liegt das Objekt im Schatten. Man muss bei diesem Effekt mit 3 Koordinatensystemen rechnen und für das Light auch eine View und Perspective Matrix anlegen.

Link zu den Folien: [http://www.cg.tuwien.ac.at/courses/CG23/slides/tutorium/CG2LU\\_Tutorium.pdf](http://www.cg.tuwien.ac.at/courses/CG23/slides/tutorium/CG2LU_Tutorium.pdf)

### Glow

Der Glow Effekt wurde entsprechend der Folien aus dem Tutorium und unter Verwendung des Artikels über Real-Time Glow in den Nvidias GPU Gems realisiert. Zur Berechnung wurden eigene Glow-Texturen verwendet, um auch eventuelle Transparenzeffekte mit dem Alphakanal der normalen Texturen zu ermöglichen. Die Glow-Szene wird in eine verkleinerte Textur gerendert und dann zunächst in horizontaler dann in vertikaler Richtung mit einem Weichzeichner gefiltert. Anschließend wird die Glow-Szene über die normal gerenderte Szene geblendet.

Link (Nvidia Developer Zone): [http://http.developer.nvidia.com/GPUGems/gpugems\\_ch21.html](http://http.developer.nvidia.com/GPUGems/gpugems_ch21.html)

### Besonderheiten

Die für View-Frustum-Culling berechneten Bounding-Boxen wurden ebenfalls für die Ray-Box-Intersection beim Auswählen eines Tower verwendet. Diese Intersection Berechnung wurde entsprechend der Verbesserung der Methode von Smit implementiert.

Link (Ray-Box Intersection Algorithm): <http://www.cs.utah.edu/~awilliam/box/box.pdf>

## Steuerung [Quick Reference]

### Gameplay & Allgemein

Taste	Wirkung
<b>ESC</b>	Spiel beenden
<b>MAUS LINKS</b>	Tower bauen

### Kamera (Normal –Bewegung auf fester Ebene)

Taste	Wirkung
<b>PFEILTASTE LINKS</b>	Nach links bewegen
<b>PFEILTASTE RECHTS</b>	Nach rechts bewegen
<b>PFEILTASTE OBEN</b>	Nach vorne bewegen
<b>PFEILTASTE UNTEN</b>	Nach hinten bewegen
<b>BILD UP</b>	Nach oben bewegen
<b>BILD DOWN</b>	Nach unten bewegen
<b>MAUS RECHTS und BEWEGUNG</b>	Kamera schwenken in die jeweilige Richtung
<b>C</b>	Wechseln in den Spectator Modus
<b>MAUS am BILDSCHIRMRAND</b>	Bewegung in die jeweilige Richtung
<b>MAUSRAD</b>	Zoomen

### Kamera (Spectator-Mode, Freie Bewegung)

Taste	Wirkung
<b>PFEILTASTE LINKS</b>	Nach links bewegen
<b>PFEILTASTE RECHTS</b>	Nach rechts bewegen
<b>PFEILTASTE OBEN</b>	Nach vorne bewegen
<b>PFEILTASTE UNTEN</b>	Nach hinten bewegen
<b>BILD UP</b>	Nach oben bewegen
<b>BILD DOWN</b>	Nach unten bewegen
<b>C</b>	Wechseln in den Normalen Modus
<b>MAUS am BILDSCHIRMRAND</b>	Kamera schwenken in die jeweilige Richtung
<b>MAUSRAD</b>	Zoomen

## Libraries

### GLFW v2.7 (SVN Build 2010-03-16)

#### *Allgemein*

GLFW wird als Window Manager verwendet und zwar ein SVN-Build der mit OpenGL 3.2 umgehen kann. Leider ist die letzte Stable Version von GLFW schon etwas älter somit mussten wir eine unstable, aber vom Institut zur Verfügung gestellte Version verwenden. (Danke!)

#### *Funktionalität*

- Zeitmessung (framerateunabhängige Bewegung)
- Window Manager (erstellen / resizen von Windows)
- Input Handling (Callbacks für Mouse und Keyboard Inputs)
- Texture / Image Loading

### GLEW v1.5.3

Da Windows nativ nur OpenGL 1.1 unterstützt verwenden wir GLEW um die Extensions zu laden und OpenGL 3.2 Funktionalität zu bekommen.

### GLM v0.8.4.4



Da unter OpenGL 3.2 die Mathematik Funktionen alle deprecated sind verwenden wir GLM um unsere Berechnungen für Matrizen oder Vektoren nicht selbst implementieren zu müssen.

#### *Funktionalität*

- Objekte zum Speichern von Matrizen / Vektoren etc...
- Mathematik für Transformationen (Translationen, Rotationen etc...)
- Unproject (Window X,Y Umwandlung in OpenGL X,Y,Z für Ray-Plane Intersection)

### FMOD v4.30.04



Wir benutzen FMOD um Hintergrundmusik und Soundeffekte im Game (Laserschüsse bzw. Superwaffe benutzen) abzuspielen. Dabei werden auf mehreren Channels gleichzeitig Audiodateien abgespielt mit jeweils eigener Lautstärke. (Dateien sind im \*.xm, \*.mod und \*.mp3 Format)

## Sonstige Anforderungen

- An dieser Stelle sein noch angemerkt, dass das Updaten der Kamera und der Objekte im Spiel stets unabhängig von der Frame Rate des Spiel passiert. Die Bewegungen der Gegner sind also auf verschiedenen Computern immer gleich schnell.
- Die Steuerung wurde ebenfalls so konzipiert, dass eine intuitive Bedienung möglich sein sollte und die konventionellen Erwartungen so gut es geht erfüllt werden.