CG2LU, SS 2009

Artillery Wars

Dokumentation der Endabgabe

Martin Gobber
532 0625799
e0625799@student.tuwien.ac.at

Alois Wollersberger
532 0625597
wollersberger@gmx.net

Kurzbeschreibung:

Gameplay:

Grundlegendes Ziel des Spieles ist es die gegnerische Kanone zu zerstören bevor der Gegner die eigene Kanone zerstört.

Das Spiel gliedert sich ein vier Gamestates:

- **Aim (3rd Person):** In diesem State wird die Kanone des aktuellen Spielers sowie UI und Fadenkreuz angezeigt. Die Kamera folgt den Zielbewegungen der Kanone. Dies ist die Standardansicht.
- **Roam (isometric):** Im durch Betätigung des Mausrads kann aus dem Aim-State heraus gezoomed werden. Drücken der linken Maustaste und ziehen der Maus ermöglicht dann eine Rotation des Spielfeldes um sich eine Übersicht zu verschaffen. Nach Loslassen der linken Maustaste bewegt sich die Kamera wieder zum Ausgangspunkt.
- **Shot:** wird die Leertaste gedrückt feuert die Kanone ein Projektil ab. Die Kamera folgt dem Projektil und der Effekt „Motion Blur“ wird aktiviert. Trifft das Projektil auf Sonne oder Planeten ist der andere Spieler an der Reihe. Trifft man eine der Kanonen, geht das Spiel in den Status Game Over.
- **Game Over:** in diesem Status wurde eine Kanone getroffen. Die Kamera verbleibt in dem Zustand den sie beim Auftreffen des Projektils hatte. Es wird die Explosion der Kanone angezeigt. Durch drücken von SPACE wird ein neues Spiel gestartet.

Effekte:

Es wurden folgende Effekte implementiert (Referenzen sowie Implementierungsdetails siehe unten):

- **Motion Blur:** Während des Projektilflugs wird erscheint die umgebung in Bewegungsrichtung verschwommen.
- **Lens Flare:** Beim Blick in die Sonne werden Reflexionen auf der Kameralinse simuliert.
- **Environment Mapping:** Ursprüngliche Planung sah vor den Effekt auf Teilen des zufallsgenerierten Spielfeldes anzuwenden um ein dem Planeten ein fremdartiges Feeling zu geben. Allerdings mussten wir die Feldgenerierung aus Zeitmangel streichen. Das Environment mapping ist jedoch implementiert und für unseren Spielfeldplaneten mittels F11 zuschaltbar.

Animierte Objekte:

Geplant war eine Funktionalität die dem beschossenen Spieler eine Möglichkeit gibt sich mit einem Schild vor einem ankommenden Schuss zu schützen (das Schild wäre nach Benutzung eine Zeit lang nicht einsetzbar). Leider wurde die Funktionalität dieser Funktion aus Zeitmangel nicht mehr implementiert. Die Animation der Schildfunktion befindet bereits im Spiel und ist mit F10 abrufbar.

Des weiteren rotiert das Rohr der Kanone mit in Abschussrichtung (sichtbar im Roam State) was allerdings keine keine tatsächliche Animation sondern lediglich eine Rotation des Rohrmodels ist.

Zusätzlich wird nach Abschuss des Gegners eine „Death Animation“ abgespielt, kombiniert mit einer 2D Textur Animation (Explosion).

Frustum Culling:

Das Frustum Culling wird so berechnet, dass für jede Seite (Oben, Unten, Rechts, usw …) des Pyramidenstumpfes – mit Hilfe der Projektions- und Modelviewmatrix – die jeweiligen Ebenenkoeffizienten (A,B,C,D) berechnet werden, so dass dann in der Methode „checkFrustum“ die Ebenengleichung „Ax + By + Cz + D“ bestimmt, ob ein Objekt (eingehüllt in einer Kugel) im Sichtbereich liegt, oder nicht. Dabei wird der Radius der umgebenden Objektkugel verglichen mit dem Wert der Ebenengleichung, um zu entscheiden, ob das Objekt im Sichtbereich ist, oder nicht.

Experimentieren mit OpenGL: Immidiate Mode / VBO / Display Lists:

Der Benutzer hat die Möglichkeit die Szene mit Immediate Mode Triangles, oder Display Lists zu rendern. Das Rendern der Szene mit VBO's is theoretisch implementiert, aber da graphische Fehler entstanden sind, wurde diese Funktionalität weggelassen bzw. auskommentiert.

Debug Ausgaben:

Unter dem Punkt „*Funktionierende Steuerung*“ sind alle Funktionalitäten aufgelistet. Es wurden alle Anforderungen implementiert (außer das Rendern mit VBO's).

Funktionierende Steuerung:

- UP/DOWN: Veränderung des Abschusswinkels (0-90)
- LEFT/RIGHT: Veränderung der Abschussrichtung (0-359)
- PAGE_UP/PAGE_DOWN: Veränderung der Abschussgeschwindigkeit(40-100)
- SPACE: Schuss abfeuern / Spiel neu Starten
- LEFT_MOUSE_DOWN: Rotation um den Planeten. Nach Release, Rückkehr zum Ausgangspunkt
- MOUSE_WHEEL: Zoomfunktion
- F2: Framerate ein/aus
- F3: WireFrame ein/aus
- F4: Texturqualität verändern: Nearest Neighbor/Bilinea
- F5: MipMapping-Qualität ändern: Aus/Nearest Neighbor/Linear
- F6: Vertex Arrays vs. Immediate Mode
- F7: Display Lists ein/aus
- F8: Viewfrustumculling ein/aus
- F9: Transparenz ein/aus
- F10: Environment Mapping am Planeten ein/aus
- F11: Animation abspielen ein/aus

Spezialeffekte

Motion Blur (motionblur.cpp):

Beim erstellen der Motion Blur Klasse wird eine Textur von der Größe des Bildschirms erstellt. Nach der Aktivierung des Effektes wird die Szene normal gezeichnet und die Textur speichert den aktuellen Bildschirminhalt. Im nächsten Durchlauf wird dann die gespeicherte Textur mit voreingestelltem Alpha Wert (konkret 0.9) über die neue Szene gelegt, und das Ergebnis erneut gespeichert.

Also Code-Beispiel diente uns ein Projekt von Stanciu Vlad „Full Scene Motion Blur“

http://www.codeproject.com/KB/openGL/MotionBlur.aspx

Lens Flare (Flare.cpp) :

Im Konstruktor der Flare Klasse wird mittels glFeedbackBuffer() Vektor als ziel des Feedback Rendermodus gesetzt. Beim aufrufen der drawFlares() Funktion wird nun der Rendermodus auf Feedback gesetzt um zu überprüfen ob die übergebe Sonnenposition innerhalb des Bildschirms gerendert wird. Ist dies der Fall so wird aus dem Abstand von Sonne und Mittelpunkt des Bildschirms ein Vektor errechnet der die Richtung in die die Flares gezeichnet werden müssen anzeigt. Der Alpha Wert der Flares verändert sich unter Abhängigkeit der länge des Directionvektors. Skalierung und Position auf dem Directionvektors wurden durch ausprobieren in ansprechender Form gewählt.

Theoretische Grundlage bildete ein Artikel von Alan Gordie unter:

http://www.gamedev.net/reference/articles/article813.asp

Code-Beispiel (skybox + lensflare):

http://www.alsprogrammingresource.com/skybox_lens_flare.html

Environment Mapping:

Hier wird eine refkletierende Textur erstellt mit dem OpenGL Parameter „GL_REFLECTION_MAP" bei der Texturgenerierung. Dann wird durch alle sechs Betrachtungsrichtungen iteriert und alle zu reflektierenden Objekte werden "aufgenommen" und in eine Textur gespeichert. Schlussendlich muss dann wieder die Projektionsmatrix und „Viewport" richtig gestellt werden.

Known Bugs:

- Der Motion Blur nimmt zwar Rücksicht auf die Fenstergröße allerdings gibt es ein Problem wenn die Fenstergröße während des eines aktiven Blurs verändert wird. In diesem Fall kommt es beim neu erstellen der Textur zu einer Spiecherverletzung.
- Lens Flares erscheinen auch wenn die Sonne vom Planeten verdeckt wird.

Modellierungstools:

Zur Modellierung der Panzer, Kanonenrohre und dem Schild, wurde zum Teil Maya, als auch Blender benutzt. Die Panzer wurden in Maya (30 Tage Testversion) modelliert und als .obj Files exportiert, so dass diese mit Blender importiert werden konnten. In Blender wurden die Modelle animiert, texturiert und schlussendlich als MD2 Modelle exportiert. Für das „Schild" wurde nur Blender benutzt. Anfangs war geplant die MD2 Animationen mit Milkshape zu erstellen, aber da dieses Programm auch nur 30 Tage lang gültig war und wir viele Probleme damit hatten, haben wir uns doch noch für Blender entschieden, obwohl das Erstellen von Animationen für MD2 Modelle doch sehr mühsam war.

Texturen wurden mit Photoshop erstellt und mit GIMP als TGAs für unseren Imageloader abgespeichert.

Externe Quellen:

Wir benutzen die externe Bibliothek GLFW [1] und haben uns beim erstellen der MD2 Model Loader Klasse und Image Loader Klasse sehr an den Code vom Buch „Beginning OpenGL Game Programming“ [2] gehalten. Unsere MD2Loader und Image Loader Klasse unterscheidet sich aber vom Buch dahingehend, dass wir mehr Funktionalität (rotate, scale, loadTexture, render (selbst implementiert aber das Buch war auch eine kleine Hilfe ;-)), etc ...) bereitstellen und die Datenstrukturen sich sehr von denen im Buch vorgeschlagenen unterscheiden. Zum Beispiel verwenden wir die eigene Template Vektorklasse „vectorN“, um Vertex Werte abzuspeichern. Unsere Klassen unterscheiden sich daher doch sehr von denen im Buch – einzig und allein die im Buch verwendeten Routinen zum Laden von Modellen (das impliziert das Laden von Bildern) und Animieren dieser wurden übernommen.

[1] GLFW – OpenGL Framework: http://glfw.sourceforge.net/

[2] Beginning OpenGL Game Programming – ISBN: 159863528X