



ADAM

Daniel Rind

0227292

e532

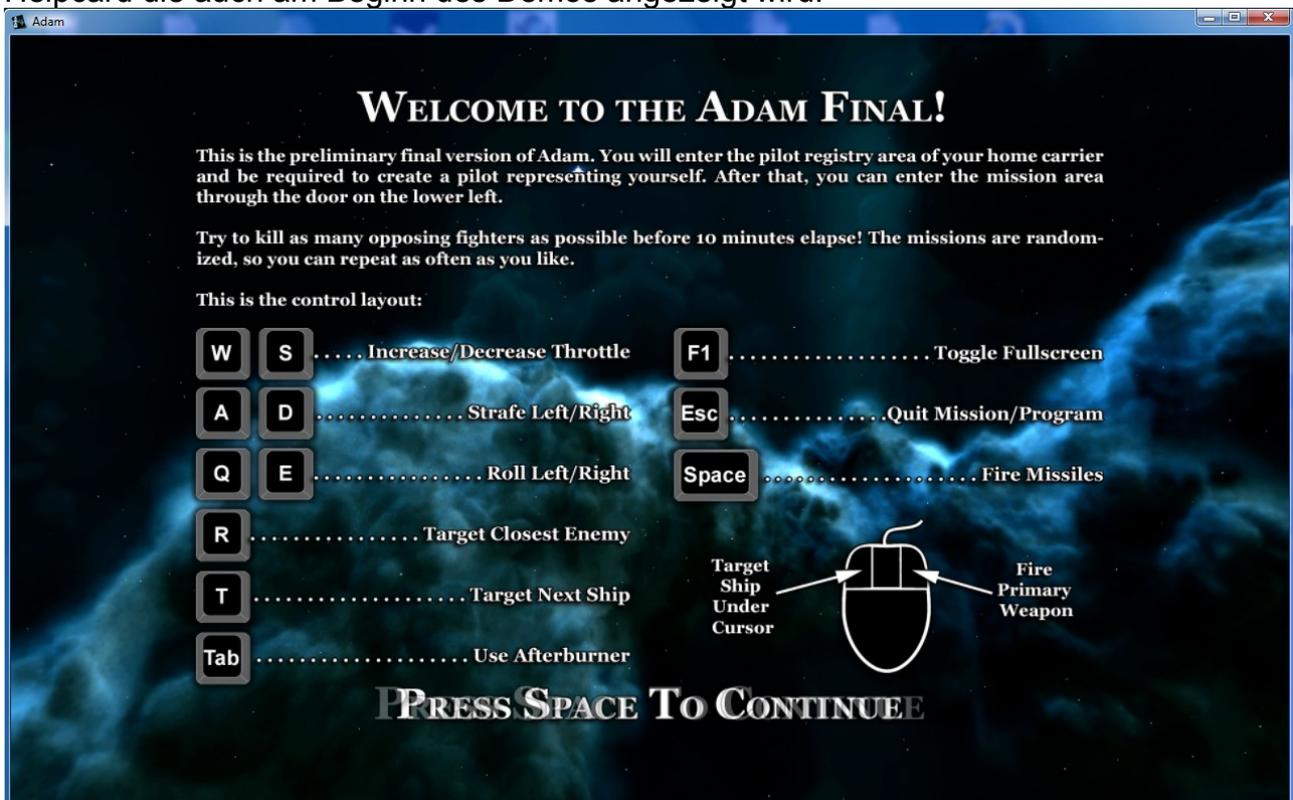
daniel.rind@chello.at

KURZBESCHREIBUNG

Dies ist die Dokumentation für die finale Abgabe des CG2/3 Projekts „Adam“. Im wesentlichen habe ich alle Ziele erreicht, wenn auch etliche Features (Shadow Mapping, komplexere Missionen, mehr Schiffe sowie ein etwas ausgefeilteres Menü- und Upgradesystem) dem Zeit- und Platzmangel zum Opfer fielen.

HANDHABUNG

Die Kontrollen orientieren sich stark an Spielen wie Freelancer oder Freespace. Hier die Helpcard die auch am Beginn des Demos angezeigt wird:



Pitch und Yaw, also rotation um die lokalen X und Y Achsen, werden durch die Position des Mausursors am Bildschirm gesteuert. Waffen feuern außerdem automatisch in die Richtung des Mausursors, und nicht entlang der Schiffs-Z-Achse, man muß also nur mit dem Mauscursor die gewünschte Feuerrichtung angeben, und nicht versuchen mit der Bildschirmmitte den Gegner anzuvisieren.

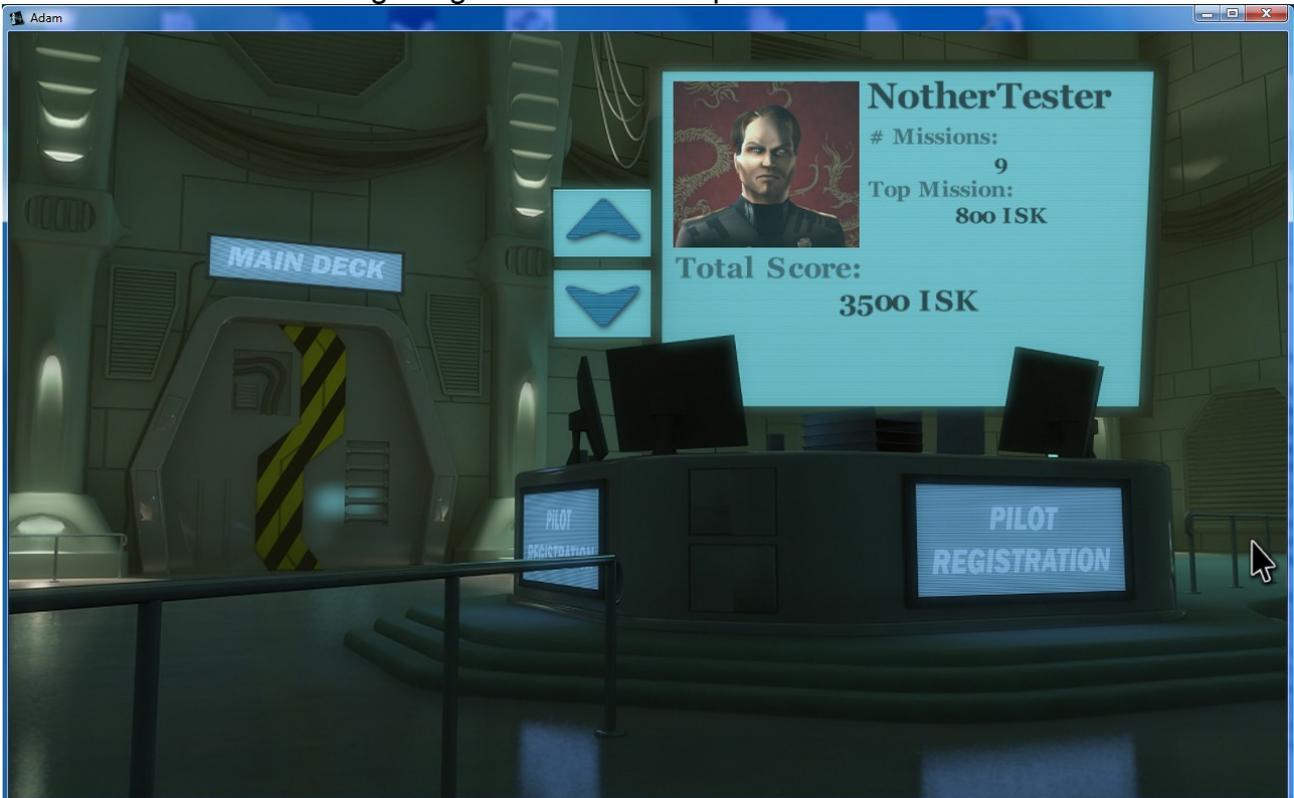
Zusätzlich zu den auf der Helpcard beschriebenen Kontrollen kann man weiters die „Pause“ Taste verwenden, um das Spiel einzufrieren und den Windows Mauscursor sichtbar zu machen. Dies ist besonders praktisch wenn man die Fenstergröße verändern will.

Des weiteren sind folgende Tasten belegt:

- ALT + ENTER.....Fullscreen (genauso wie F1)
- F2.....Framerate
- F3.....Wireframe
- F4.....Mipmapping aus/nearest/linear
- F5.....Texel filtering nearest/linear
- F6.....Adaptive HDR an/aus
- F7.....Immediate/Display List/VBO
- F8.....View Frustum Culling an/aus
- F9.....Transparenz an/aus

GAMEPLAY

Nach dem Titelbildschirm gelangt man in den Hauptbereich des Mutterschiffs:



Hier muß man einen Piloten anlegen bevor man eine Mission fliegen kann. Missionen können durch die Tür links betreten werden. Die Pfeiltasten links des Hauptmonitors

ermöglichen das Durchschalten der Piloten. Die Registrierung öffnet man durch klicken auf die „Pilot Registration“ Schirme:



Man kann einen kurzen Namen eingeben, und durch klicken auf das Portrait ein Bild auswählen. Hat man einen Piloten erstellt/ausgewählt, betritt man durch die Tür eine Mission.



Links unten wird auf Wunsch die Framerateanzeige eingeblendet (F2).

Die Kontrollen und das HUD haben sich gegenüber der Demoversion nicht wesentlich geändert. Links oben wird die Aktuelle Missionszeit eingeblendet (jede Mission dauert max. 10 Minuten), in der Mitte oben werden sowohl die Punkte für die aktuelle Mission, als auch die insgesamt gesammelten Punkte angezeigt. Man erhält für alle Abschüsse der eigenen Partei Punkte, doch eigene Abschüsse bringen wesentlich mehr als Abschüsse der anderen Fighter.

LEVEL

Der Level ist nach wie vor recht einfach gehalten, wird jedoch durch einen Zufallsgenerator optisch (und auch im Schwierigkeitsgrad) durchaus verändert. Besondere Vorsicht ist geboten, wenn ein zweites feindliches Trägerschiff unterhalb des ersten Schiffs spawnnt, da man dann einer 2:1 Übermacht gegenübersteht.

Der Level endet nach 10 Minuten, wenn man abgeschossen wird, oder wenn man mit ESC aus dem Level aussteigt.

KONFIGURATION/SAVEGAMES

Die Konfigurationsdatei sowie alle Piloten werden in <My Documents>/Adam abgelegt. Die Zeilen der Settings.ini sollten selbsterklärend sein.

ANFORDERUNGSLISTE

Anforderungen an die 3. Abgabe:

- **Gameplay:** Das Spiel ist durchaus spielbar, und die KI kann eine gewisse Herausforderung bieten, obwohl sie nicht wirklich stellar ist. Als Spielziel kann das Sammeln von möglichst vielen Punkten bzw. von einer möglichst hohen Punktezahl in einer einzelnen Mission angesehen werden.
- **Nichttriviale Objekte:** Der .obj-Loader wurde von mir selbst geschrieben und versteht im Prinzip beliebige .obj Dateien (es dürfen jedoch nur Triangles verwendet werden). Zusammen mit einer extra Datei in der Shader und Texturen für die Materialien festgelegt werden sind denke ich alle Anforderungen für nichttriviale Objekte erfüllt. Tangenten/Binormalen werden beim Laden dynamisch berechnet.
- **Animierte Objekte:** Etwas schwierig bei starren Raumschiffen, doch ich hoffe, daß Explosionen sowie einige der Spezialeffekte (Funken beim Aufschlag von Geschossen, etc.) hier genügen.
- **View Frustum Culling** wurde implementiert, der Geschwindigkeitsgewinn ist jedoch, außer beim Immediate Mode Rendering, kaum zu merken.
- **Transparenzeffekte** kommen zahlreiche vor - Abgasstrahlen, Explosionen, Projektile, Lens Flare, etc... Depth-Sorting mußte ich nicht implementieren, da eigentlich alle Sprites, bei denen es zu Verdeckungsproblemen kommen könnte, additives Blending verwenden.
- **Experimentieren mit OpenGL** wurde ebenfalls umgesetzt, ich hoffe das paßt so wie es implementiert wurde.

FEATURES

Folgende Features wurden umgesetzt:

- **Komplexer Shader** für die Oberfläche von Raumschiffen, der Normal Maps, Glow Maps und Diffuse Texturen kombiniert. Außerdem wird das aktuelle Ziel per Shading-Effekt angezeigt (pulsiert in rot / blau), und die Zerstörung eines Raumschiffes per animierter Noise Texturemap umgesetzt. Alle Raumschiffe verwenden denselben Shader, unterschiedliches Aussehen wird über Parameter sowie Texturen erreicht. Das Shader Loading sowie die korrekte Initialisierung befinden sich in der CModel Klasse, der Shader in /shaders/Ship
- **Partikelsysteme** für alles Mögliche – Raketenabgase, Explosionen, Treffereffekte, etc... Die Klassen CParticleManager sowie CParticle sind dafür verantwortlich, sowie CExplosion für die Explosionseffekte.
- **HDR Rendering mit Bloom Effekt.** Es wird in einen 16 bit Floating Point Buffer gerendert, und mit Hilfe einer 11 Pixel Gauss Blurs werden helle Bereiche verschwommen dargestellt. COGIWnd->BuildFBO() sowie COGIWnd->DoHDRMagic() sind dafür verantwortlich, zusammen mit den HDRclip, HDRblurH, HDRblurV und HDRblend Shadern.
- **Adaptives HDR.** Es werden die minimale, maximale, sowie durchschnittliche Helligkeit eines herunterskalierten Bildes ermittelt, und mit den gewonnenen Werten werden Helligkeit, Kontrast und Gamma des finalen Bildes verändert um eine halbwegs realistische hell/dunkel Adaption zu erreichen, die auch bei extremen Helligkeitswerten nicht „ausflippt“. Die adaptive Berechnung findet in COGIWnd->DoHDRMagic() sowie im /shaders/HDRblend Shader statt.
- **Physik mit Hilfe von PhysX.** Große Raumschiffe werden als Trimeshes geladen und als kinematische Actors realisiert, kleinere Fighter als eine Ansammlung konvexer Hüllen und als dynamische Actors. Das Baking ist über die zwei Batch-Dateien im Root Verzeichnis zugänglich.
- Objekte werden als .obj Files und Texturen als .tga Files geladen. Alle zusätzlichen Informationen sind in einfachen Textdateien abgespeichert und werden durch die Loader bei Bedarf dazugeladen. Der massive Speicherplatzbedarf limitiert die Anzahl der Raumschiffe, Texturen und Menübildschirme extrem, für einen Umstieg auf .dds oder ähnliches hatte ich leider keine Zeit mehr. Texturen werden in CTexture und .obj Files in CModel geladen.
- **Oren-Nayar Shader** für Planeten, wobei die Rauheit über eine Texturemap festgelegt werden kann, was für das unterschiedliche Aussehen von Land und Wasserbereichen verwendet. Der Planetenshader verwendet drei 4-channel TGA texturen, wobei alle Kanäle belegt sind (Shader ist zu finden in /shaders/Planet).
- **Lens Flare** mit korrekter Verdeckung. Der Lensflare wird in CEnvironment::DrawSunFlare() berechnet.

ZUSATZTOOLS/LIBRARIES/EFFEKTSQUELLEN

Bis auf GLEW und PhysX wurden keine externen Libraries verwendet. Um die Modelle aus EVE Online zu extrahieren wurde der TriExporter verwendet (keine offizielle URL, ist in den EVE Online Foren zu finden). Die Modelle wurden in Softimage XSI erstellt / nachbearbeitet, Texturen und Graphiken wurden mit Photoshop erstellt. Die Eingangshalle wurde ebenfalls mit XSI modelliert und gerendert.

Wirkliche Quellen für die implementierten Effekte habe ich keine, die Implementierungen sind alle „auf meinem Mist gewachsen“. Das Formelwerk für den Oren-Nayar Shader ist dem Originalpaper entnommen:

http://www1.cs.columbia.edu/CAVE/publications/pdfs/Nayar_IJCV95.pdf