CG 2/3 Übungen SS 2003

Egele, Manuel; 881, 0025546; e0025546@stud3.tuwien.ac.at
Szydlowski, Martin; 881, 0025313; e0025313@stud3.tuwien.ac.at

Spiel mir … Das Spiel vom TOD!

Ein Action-Jump&Slay-Adventure der besonderen Art.

I. Spielbarkeit - status quo

Leider ist auch bei der finalen Abgabe die Spiellogik nicht vollständig. Obwohl ein grosser Teil schon implementiert ist, wurden manche Routinen aus Zeitmangel nicht in den ausführbaren Teil des Codes eingebunden.
Die Animation und Steuerung des Spielercharakters sind schon beinahe vollständig, es müssen noch kleine Ungereimtheiten ausgemerzt werden. Der TOD kann nun die Sense auf unterschiedliche Arten schwingen, hat Geh-, Lauf- und Idleanimationen und kann auch Springen (allerdings muss hier noch etwas an der Animation gefeilt werden). Es besteht auch die Möglichkeit unterschiedliche Waffen zu verwenden und sie auszuwechseln, allerdings existiert bisher nur die Sense. Die Collision Detection mit der Sense ist noch nicht implementiert, deshalb kann auch keine „Interaktion“ mit den Gegnern stattfinden.
Die Gegner bewegen sich auf durch Waypoints bestimmten Pfaden, die zufallsbestimmt sind aber bestimmten Grundsätzen folgen, z.B. dass die Gegner einem Weg folgen wenn sie auf einen stossen. Dem Zeitmangel zum Opfer gefallen sind Routinen, die die Gegner weglaufen lassen wenn der TOD näher kommt, sowie die gesamte Gegnerverwaltung (Auswahl der Zielpersonen, Hinzufügen und Entfernen von Gegnern zur Laufzeit). Das bei der Abgabe verwendete Model ist ein aus einem Tutorial entnommenes Quake 3 Model.
Die Objekte im Level sind weiterhin nur Bäume, allerdings in verschiedenen Formen und mit verschiedenen Texturen.
Das Gelände wird nun zur Laufzeit für jedes Level dynamisch erzeugt, sowie die Wege und die Geländetextur. Auch die Positionierung der Objekte erfolg nun nicht mehr komplett zufällig, sondern wird durch den Levelgenerator festgelegt, dem man bestimmte Parameter übergeben kann.
Eine weitere wichtige Neuerung ist die Möglichkeit, bestimmte Parameter über ein .cfg – file festzulegen, vor allem die Steuerung sowie Anzahl und aussehen der Levels.

II. Start & Steuerung

Nach aufrufen des Executables kommt man in den Startbildschirm. Wenn man der Aufforderung „Press Entert to begin“ nachgeht, wird der erste Level geladen. Backspace ladet den nächsten im cfg-file definierten Level, so lange es noch welche gibt. Die Spielsteuerung ist im cfg-file festgelegt, die Defaultbelegung (also wenn für eine bestimmte Aktion kein Eintrag im cfh-file existiert) ist wie folgt:

- Vorwärts: PFEIL OBEN
- Rückwärts: PFEIL UNTEN
- Links drehen: PFEIL LINKS
- Rechts drehen : PFEIL RECHTS
- Links strafen: S
- Rechts Strafen: F
- Laufen (halten): A
- Spiel verlassen: ESC

Weiters gibt es spezielle Befehle, die Teilweise zum debuggen dienen:

- Umschalten Fullscreen/Windowed: F1
- Auflösung ändern (in beiden modi): 1-6
- Wireframe Modus: F2
- Umschalten (Spiel/Freeflight): F3
- FPS an/ausblenden: F4
- Karte rendern ja/nein: F5
- Spieler rendern ja/nein: F6
- Objekte rendenr ja/nein: F7
- Gelände zeichnen ja/nein: F8
- Nebel ein/ausschalten: F9
- View Frustum Culling an/aus: F11
- Screenshot: F12
- Lichtrichtung ändern: NUMPAD 2,4,6,8,5
- Kamera rauf (nur Freeflight): E
- Kamera runter (nur Freeflight): D
- Kamera yaw up (nur Freeflight): PAGE UP
- Kamera yaw down (nur Freeflight): PAGE DOWN
- Info Bildschirm & Spielauflösung: 0

Die unterstützen Auflösungen (Hotkey):

- 640x480 (1)
- 800x600 (2)
- 1024x768 (3)
- 1152x864 (4)
- 1280x1024 (5)
- 1600x1200 (6)

Lichtsteuerung:

- NUMPAD 5: Sonne auf 12 Uhr mittags am Äquator am 21.März (also direkt von oben)
- NUMPAD 2: Sonne zum Horizont neigen
- NUMPAD 8: Sonne zur Mittagsposition neigen
- NUMPAD 4: Sonne gegen Uhrzeigersinn rotieren
- NUMPAD 6: Sonne mit Uhrzeigersinn rotieren

III. Implementierung

Datenstrukturen:

1. Gelände:

Die Vertices zur Darstellung des Geländes werden in einem Vertex Array gespeichert, die dazugehörigen Normals und Texture Coordinates in einem Normal- bzw. Texture Coordinate Array. Die Vertices sind in einem Gitter angeordnet und haben gleiche Abstände in x und in z Richtung, der y-Wert ergibt sich aus der durch den Geländegenerator erzeugten NURBS Surface. Das Gelände wird in kleinere Geländeeinheiten (Patches) unterteilt, wobei jedes Patch ein eigenes Indexarray hat, mit dem sich alle in diesem Patch befindlichen Dreiecke mit glRenderElements() als ein einziger GL_TRIANGLE_STRIP rendern lassen. Dazu müssen natürlich die GL-Clientstates GL_VERTEX_ARRAY, GL_NORMAL_ARRAY und GL_TEXTURE_COORD_ARRAY aktiviert werden, sowie die gl*Pointer() auf die entsprechenden Adressen gesetzt werden. Für weitere Optimierung sorgt die GL_EXT_compiled_vertex_array Extension, die aktiviert wird, falls sie vom Treiber zur Verfügung gestellt wird.

2. Models:

Als Modelfomat wird nun das von Quake 3 Arena bekannte MD3 Format verwendet. Die Loader und Animations Funktionen wurden alle selber geschrieben, ausgehend von den in einem Tutorial enthaltenen Informationen (Animation) und der MD3 Dateiformatbeschreibung (Loader). Alle sich im Spiel befindenden Objekte (ausser dem Gelände und der provisorischen Gegnermodel) wurden von uns in 3D Studio MAX 5.1 erstellt und dann mittels Pop'N'Fresh MD3 Export Plugin ins MD3 Format gebracht und dann mit Nphernos MD3 Compiler nachbearbeitet (der MD3 Exporter versaut die Normals).
Ein Komplettes MD3 Model besteht aus 4 Teilmodels (lower, upper, head, weapon) die in Separaten Files abgelegt sind. Unser Loader geht über die MD3 Spezifikation hinaus und gestattet auch das Laden und Rendern von Models mit weniger Teilen (z.B. die Bäume bestehen nur aus einem Teil). Es wäre auch möglich Models mit mehr Teilen zu laden, allerdings sahen wir keine Notwendigkeit dafür.
Zu den Files mit Geometriedaten gibt es auch noch Text Files, die Informationen über die Texturen und die Animation enthalten. Unser Loader ermöglicht es für ein Model mehrere Sätze von Texturen zu laden um Speicherplatz zu sparen (Die Geometriedaten liegen nur einmal im Speicher). Allerdings wird dazu ein extra File benötigt, das die Namen der zugehörigen .skin Files enthält.
Die verwendeten Datenstrukturen zur Speicherung und zum Rendering schauen wie folgt aus:

- **MD3Model:** Diese Klasse enthält die Funktionen zum Rendern eines aus mehreren Teilen bestehenden Models und enthält 4 Objekte vom Typ **MD3ModelPart** (lower, upper, head, weapon). Beim Rendern wird die DrawLink funktion mit dem lower ModelPart ausgeführt. Diese rendert die im ModelPart enthaltene Geometrie (Keyframe Interpolation, deshalb Immediate Mode), und dann iteriert sie durch alle mit diesem Teil verbundenen Teile des Models und ruft sich rekursiv auf. Die Translation und Rotation des nächsten Teils wird aus dem dazugehörigen Tag entnommen und zwischen den Keyframes interpoliert (Translation linear, Rotation mittels Spherical Linear Interpolation unter verwendung von Quaternionen). Ausserdem gibt es hier Funktionen um die aktuelle Animation festzulegen und während einer Animation zu berechnen (auf Grund der verflossenen Zeit) zwischen welchen Keyframes interpoliert werden soll. Zu guter Letzt gibt es noch die CloneModel funktion, die ein neues MD3Model Objekt zurückliefert, das allerdings Zeiger auf dieselben Geometrie und Texturdaten enthält, und einen Parser für das Animation.cfg file, welches Informationen über die Animation enthält (welche Keyframes gehören zur welchen Animation)
- **MD3ModelPart:** Enthält die Statusvariablen die die aktuelle Animation festlegen sowie einen Zeiger auf ein **MD3ModelData** Objekt, welches die Geometrie enthält. Ausserdem enthält es noch ein Array mit Zeigern auf andere **MD3ModelPart** Objekte die mit diesem Verbunden sind.
- **MD3ModelData:** Hier sind die Funktionen zum Laden der Geometriedaten aus einem MD3 File sowie zum Parsen des .skin Files welche Texturinformationen enthalten. Dieses Objekt enthält Arrays von Mesh, tTag und tAnimInfo Objekten.
  Ein Mesh Objekt enthält die eigentliche Geometrie (als Array von tVertex für vertexkoordinaten, Array von tNormal für Normals, Array von tTexCoord für Texturkoordinaten sowie ein Array von tFace, welches jeweils die Indizes der zu einem Triangle gehörenden Vertices enthält) sowie Texturinformationen.
  Ein tTag Objekt enthält die die Translation und Rotation (als 3x3 Orthogonale Rotationsmatrix) eines Tags, welche als Bezugspunkt für ein an dieses Model anchliessende Models dienen.
  Ein tAnimInfo Objekt enthält die Information zu einer Animationsphase (Startframe, Endframe, Frames per Second, …)

3. Texturen:

Die Texture Klasse beherbergt alle notwendigen Methoden, um ein Bitmap im Format BMP (24Bit RGB), PCX (8Bit Palette), TGA (24 oder 32Bit RGBA) oder JPG aus einer Datei zu laden und sie als OpenGL Texture Object zur Verfügung zu stellen, dabei werden Mipmaps automatisch mittels gluBuild2DMipmaps generiert.

4. Objekte

Alle 3D Objekte im Spiel sind Subklassen der Object Klasse. In dieser Klasse werden Position und Ausrichtung des Objekts gespeichert. Für statische Objekte gibt es die SimpleObject Klasse, sie speichert die Nummer der mit dem Objekt assoziierten Displaylist, und führt diese beim Aufruf der draw() Methode aus. Für animierte Objekte gibt es die ComplexObject Klasse, die einen Pointer auf ihre eigene Instanz der MD3Model Klasse hat, beim Aufruf von draw() wird die rendermethode des MD3s aufgerufen. Der Player ist eine Subklasse von ComplexObject, erweitert um die Steuerung des Spielerzustands (Angreifen, Idle, etc.)

5. Level

Die Level Klasse beherbergt die Geländedaten, sowie Methoden zum Rendern des Geländes sowie darin enthaltener Objekte. Der integrale Bestandteil der Level Klasse ist der Quadtree, der das Gelände in kleine Quadratische Geländeeinheiten (Patches) zerlegt, und worin die IndexArrays für die Darstellung dieser Patches enthalten sind, sowie Pointer auf die sich im Level befindenden Objekte. Weiters wird mit dem Quadtree view frustum culling und collision detection realisiert Beim Aufruf der draw() Funktion wird der Quadtree rekursiv durchlaufen, ausgehend von Wurzelknoten, der das gesamte Level repräsentiert. Dabei wird mittels eines Clipping-Algorithmus (Sutherland-Hodgeman) festgestellt, ob sich der aktuelle Knoten im Sichtfeld befindet, falls nicht werden die Kinder nicht mehr betrachtet. An einer bestimmten Tiefe (die sich ergibt aus ld(Levelgrösse/Patchgrösse)) enthalten die Knoten einen Pointer auf das zum rendern des Geländes notwendige Indexarray. Wenn dieser Pointer nicht NULL ist, wird er zu einem Pointerarray hinzugefügt, wenn der Baum durchlaufen worden ist werden die so gesammelten Index Arrays mittels glMultiDrawElementsEXT auf einmal gerendert. In den Ebenen darunter werden nur solche Knoten betrachtet, deren Flag Variable nicht 0x0 ist, denn das weist darauf hin, dass in der untersten Ebene der Knoten einen Pointer auf ein Objekt hat, dass ich an diesen Koordinaten im Level befindet. Ist dieser Pointer nicht NULL wird die ein Flag im ObjecIndexArray gesetzt und anschliessend, nachdem das Gelände gezeichnet worden ist, wird das Index Array durchlaufen und Objekte mit gesetzter Flage gerendert. Dann wird noch das Wasser gerendert, als ein grosser Quad mit bewegter Textur und Alpha-Blending und zuletzt die Map (Alpha-geblendet) welche aus

einem Quad mit der Leveltextur besteht, sowie mit Punkten, die die Position des Spielers und der Gegner darstellen.
Mit Hilfe des Quadtrees wird auch collision detection bewerkstelligt. Sie beruht auf dem simplen Prinzip, dass wenn ein Knoten in der untersten Ebene (=kleinste Geländeeinheit) durch ein Objekt belegt ist (Object Pointer != NULL), kein anderes Objekt diesen Knoten betreten kann. Somit wird verhindert, dass Bäume im Wasser wachsen, dass der Spieler und die Gegner nicht durch Bäume oder über Wasser laufen können, sowie dass sich zwei bewegte Objekte nicht durchdringen. Bewegte Objekte überprüfen jedes Mal, ob sie noch im „alten Knoten“ sind, und falls nicht dann entfernen sie sich aus dem alten und „hängen“ sich in den neuen rein.

6. GameEngine

Die GameEngine Klasse erzeugt beim Start ein Level Objekt und zerstört es wenn das Level beendet worden ist, um ein neues erzeugen zu können. Sie enthält ein ResourceServer Objekt, welches alle aus Files geladenen Objekte (Models, Texturen) verwaltet. Zur Laufzeit bestimmt die GameEngine in welchem Zustand sich das Spiel gerade befindet (zur Zeit nur STARTSCREEN, LOADSCREEN, GAMESCREEN) und führt dann die entsprechende draw() Funktion aus. Im Zustand GAMESCREEN sammelt die Funktion zuerst die Eingabe, berechnet dann die Position des Spielers und der Kamera. Dann wird die draw() Funktion des Levels aufgerufen, sowie eventuell die FPS Einblendung (die auch die Information enthält, wie viele Patches und Objekte im Moment gerendert werden) gerendert.

7. ResourceServer

Der RS verwaltet alle Objekte (Models und Texturen), in SDL vectors. Er hat Funktionen um die Objekte zu laden und sie wieder aus dem Speicher zu entfernen, und ist hauptsächlich dafür da, dass kein Model und keine Textur unnötigerweise mehr als einmal geladen werden. Falls ein Load befehl kommt, überprüft der Server zuerst ob das Objekt nicht schon im Speicher liegt und wenn ja, dann wird ein Zeiger auf dieses Objekt zurückgeliefert. Falls das Objekt noch nicht geladen worden ist, so wird das nachgeholt.

8. TerrainGenerator

Der Terrain Generator zeichnet sich dadurch aus dass jedes mal wenn ein Gelände erzeugt wird das Ergebnis unterschiedlich aussieht, was durch den massiven Einsatz von Pseudozufallszahlen erreicht wird und den Spielspass dadurch erhöht, dass keine 2 Levels gleich sind.
Das Gelände basiert Grundsätzlich auf einer NURBS – Oberfläche, Den Wegen liegen B-Splines zugrunde. Das Ergebnis kann als 24BPP - BMP exportiert werden.

Zuerst werden zufällig eine vorgegebene Anzahl von Stützpunkten (bzw deren Z - Werte) auf einem Rechteckigen regelmässigen Grid das der Level-Map zugrundeliegt erzeugt. (z.B. 8x8 Stützpunkte für eine 256x256 map).
Dann wird über diesem Grid durch diese Stützpunkte eine NURBS - Surface gelegt. Dadurch wird die Oberfläche des Levels bestimmt. Es können dabei die minimalen und maximalen Höhenwerte angegeben, sowie eine Höhe die in weiterer Folge den Wasserspiegel (der im ganzen Level konsistent ist) festgelegt werden.
Über das so erzeugte Gelände wird dann eine oder Mehrere B-Splines gelegt die die Wege im Gelände wiederspiegeln. Hierbei werden die Stützpunkte wiederum zufällig generiert. Nachdem die Grundform des Weges festgelegt wurde, wird der Verlauf dahingehend überprüft, ob sich teile unterhalb des Wasserspiegels befinden, sollte dies der Fall sein so wird eine Anpassung des Weges entlang der Uferlinie des Betreffenden Gewässers vorgenommen. Teilweise werden zusätzlich die Wasserwege durch Brücken Überquert.

9. PeopleEngine

Die Aufgabe der PeopleEngine besteht darin die im Spiel vorhandenen Gegener zu erzeugen und zu verwalten. Die aktiven Charaktere werden in 3 Gruppen unterteilt. Solche die ihr Leben gewöhnlich führen (ALIVE), solche deren Aufgabe des Spielers es ist ihre Seelen einzukassieren (TOKILL) oder solche deren Seelen hätten sollen heimgeführt werden was der Spieler allerdings versäumt hat (SURVIVED).
Auch hier kommen Pseudozufallszahlen verstärkt zum Einsatz.
Über Parameter kann gesteuert werden wie viele Charaktere maximal gleichzeitig als Gegner im Spiel existent sein sollen. Dann wird in regelmässigen Abständen eine Aktualisierung der Zustände der einzelnen Charaktere vorgenommen. Ein weiterer Parameter legt fest wie viele Gegner pro Aktualisierung ihren zustand wechseln. Eine Lebensspanne eines Gegners beginnt immer mit ALIVE geht mit der Zeit über in TOKILL und sollte dann der Tod noch nicht zugeschlagen haben in SURVIVED.

10. Windows-Framework

Das Windows-Framework ist in der GameWindow Klasse, sowie in den globalen Funktionen WinMain und WndProc realisiert. WinMain ist der Einsprungspunkt für die Applikation und enthält die berüchtigte Windows Main Loop, von hier wird die draw() Funktion der GameEngine aufgerufen. WndProc behandelt die eingehenden Windows Messages, wobei hier hauptsächlich die Tastatureingaben behandelt werden, die der Steuerung dienen. Die GameWindow Klasse beherbergt alle notwendigen Funktionsaufrufe, um ein Fenster zu erstellen, zu zerstören, die Grösse oder Art des Fensters zu ändern und die Bildschirmauflösung zu ändern. Dabei wird stets die optimale (sprich höchste verfügbare) Bildwiederholfrequenz

gewählt. Alle Änderungen am Fenster können zu jedem Zeitpunkt im Spiel durchgeführt werden (Hotkeys F1, 1-6).

Das Kameramodell

Das Spiel ist als 3rd Person Spiel ausgelegt, demzufolge orientiert sich die Kameraführung an anderen Spielen aus diesem Genre. Die Kamera ist stets auf den Spieler ausgerichtet (ausser im Freeflight Modus), schwebt ihm sozusagen nach. Der Hinterkopf des Avatars ist dabei meistens in der Mitte, während ihn die Kamera von einer leicht erhöhten Position betrachtet. Beim Wenden dreht sich die Kamera um die Spielerposition. Die Neigung (Yaw) der Kamera passt sich dem Gelände an, desshalb ist es (zumindest für den Moment) besonders wichtig, dass das Gelände „smooth“ ist, um eine angenehme Kameraführung zu gewährleisten. Geht der Spieler bergauf, so schaut die Kamera nach oben, geht er nach bergab so schaut sie hinunter. Im Freeflight Modus ist die Kamera 1st Person, rotiert also um sich selber, weiters kann man die Höhe und die Neigung selber bestimmen. Die Bewegungsrichtung ist aber nicht von der Neigung abhängig.

Texturing

Das Gelände wird mittels Single-Pass Mutlitexturing texturiert, unter Verwendung von 2 Texture Units, wobei die erste eine grosse (von Levelgrösse abhängig, für ein 128x128 Level ist die Textur 1024x1024) beim generieren des Levels erzeugte Leveltextur enthält auf der die Wege, Seegrund und Schatten eingezeichnet wurden. Die zweite eine Detailmap (aus einer Datei geladen) um dem Boden mehr Struktur zu verleihen. Die Texturkoordinaten werden beim generieren des Levels berechnet, und zwar jeweils ein Satz für die Hautptextur und eine Satz für die Detailtextur.
Die Objekte enthalten alle schon die Texturkoordinaten, jedes Objekt kann mehrere Sätze von Texturen haben, die mittels MD3Model.currentTexture ausgewählt werden. Bei einem Renderdurchgang kann jeweils nur ein Satz texturen verwendet werden. Die Anzahl der Texturen pro Satz wird durch das MD3 Model Format auf eine pro Mesh limitiert, wobei ein MD3File bis zu 32 Meshes enthalten darf.

Beleuchtung/Materialien

Alle Objekte und das Gelände werden mittels einer direktionalen Lichtquelle („Sonne“) beleuchtet, welche von Spieler gesteuert werden kann. Es wird das GL_SMOOTH Shading Model Verwendet und es wird nur die sichtbare Seite beleuchtet. Bei der Erstellung des Geländes werden per Vertex Normals berechnet, somit wird das Gelände auch smooth geshaded. Bei den Objekten werden die mitgelieferten (codiert abgespeicherten) per Vertex Normals verwendet, um auch

die Objekte smooth zu shaden. Materialien werden nicht verwendet, es lässt sich nur die Farbe eines Objekts festlegen.

IV. Special Effects

Leider ist sich aus Zeitgründen die Implementierung von besonderen Special Effects nicht ausgegangen. Geplant wären:

- Reflexion an der Wasseroberfläche (Mittels Sencil Buffer/Render to Texure)
- Environment Mapping für die Klinge der Sense
- Partikeleffekte (himmlische bzw. höllische Flammen)
- Schatten
- Eventuell LoD

V. Features

Hier noch mal alle wichtigen Features auf einen Blick:

- Zufallsgeneriertes Gelände mit Wegen, Brücken, Vegetationszonen
- Einfache Konfiguration mittels cfg-file
- Steuerbare direktionale Lichtquelle
- Smooth shading
- Animierte und texturierte Models
- Animation mittels Keyframeinterpolation
- View frustum culling
- Vertex Arrays
- Display Lists
- Collision detection
- Weniger Nebel als beim letzten Mal ;-)

Anhang a: das CFG-FILE

Das cfg-file muss im selben Verzeichnis liegen wie das Executable. Es kann folgende Sektionen enthalten:

- Eine [General] Sektion: Auflösung beim start sowie die Pfade zu den Resourcenverzeichnissen
- Eine [Controls] Sektion: Steuerung, frei konfigurierbar, beschränkt sich aber im Moment nur auf die Buchstaben auf der Tastatur (ohne Umlaute oder Sonderzeichen) sowie Space und die Cursor tasten.
- Sowie beliebig viele (aber mindestens eine) [Level] Sektionen: Sie legen die Parameter für den Levelgenerator fest.