PacSheep

ENDBERICHT

(3. Abgabe)

Ausgearbeitet von:
Cornelia Brenner, 0404399 (532), conni_b@gmx.at
Matthias Zeintlinger,0355400 (935), matthias.zeintlinger@liwest.at

Für die Lehrveranstaltung:
Computergraphik 2 – Laborübung, SS2007

Betreuung:
Michael Wimmer

Pac Sheep – Endbericht

Besonderheiten

Als Besonderheit bleibt der FBX-Loader zu erwähnen, der gänzlich selbst geschrieben wurde. Wir können auch komplexere Objekte problemlos in das Spiel hineinladen. Die Objekte wurden alle selbst mit Maya modelliert. Sogar Lichtquellen (bis max. 7), die in Maya positioniert wurden, werden in unsere Datenstruktur übernommen. Was auch viel Zeit und Arbeit gekostet hat war die selbst implementierte Kollisionserkennung mit Achsenparallelen BoundingBoxen.

Natürlich wurden alle Empfehlungen nach der 2. Abgabe (gefilterte Texturen, restliche Beleuchtung, Schaf bewegt sich am Boden) umgesetzt.

Alphablending wurde bei den Effekten und bei Nachrichten verwendet. Debugausgaben oder Spielstandinformationen werden auf einer halbtransparenten Hintergrundfläche ausgegeben. Debugnachrichten verschwinden ein oder zwei Sekunden nach der letzten Debugausgabe automatisch wieder.

Gameplay

Die Steuerung des Schafes funktioniert über die Tasten `w`, `a` und `d`. Mit `m` kann man das Labyrinth von oben betrachten. Wenn das Schaf über Grasbüschel fährt, so werden diese aufgesammelt und das gibt Punkte. Wenn alle Grasbüschel aufgesammelt wurden, ist das Level geschafft und das nächste Level wird geladen. Wenn das Schaf eine Karotte aufsammelt kann man mit `e` eine Tretmine legen.

Die Gegner (Wölfe) werden vom Computer gesteuert. Wenn ein Wolf mit einer Wand kollidiert, dreht er sich solange in eine Richtung bis er wieder weitergehen kann und geht dort solange weiter bis er wieder mit einer Wand kollidiert usw. Fährt der Wolf über eine zuvor gelegte Tretmine, so explodiert er. Wenn das Schaf mit einem Wolf kollidiert, verliert das Schaf ein Leben (von insgesamt fünf). In diesem Fall werden Schaf und Wolf auf ihre Ausgangsposition zurückgesetzt. Wenn das Schaf alle Leben aufgebraucht hat, ist das Spiel verloren.

Effekte

Als Effekte wurde in der ersten Abgabe ein Partikelsystem, Lensflares und Schatten vorgeschlagen. Folgende wurden tatsächlich realisiert.

Die Explosion des Wolfes wurde als Partikelsystem umgesetzt. Da wir vorher noch nie ein Partikelsystem programmiert haben, haben wir uns an das NeHe – Tutorial #19 (http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=19) angelehnt. Jedes Partikel bekommt einen zufälligen Richtungsvektor, dessen Länge der Geschwindigkeit des Partikels entspricht, eine Startposition, eine Farbe und einen zufälligen Wert, um den die Lebenszeit des Partikels (beim Start der Explosion auf 1.0 gesetzt) nach jedem Renderdurchgang verringert wird. Wenn diese Lebenszeit den Wert 0.0 unterschreitet, wird das Partikel inaktiv gesetzt. Diese Lebenszeit ist auch gleichzeitig der Alphawert des Partikels, also je länger es lebt desto mehr verblasst es. Die Partikel wurden als

Billboards realisiert, also als 2D-Fläche mit einer transparenten Textur, die immer so ausgerichtet wurde, dass die Vorderseite genau in die Kamera zeigt (also normal auf den LookAt – Vektor steht). Bei jedem Renderdurchgang wird also zuvor berechnet, um wieviel Grad sich die Fläche um die X- u. Y-Achse drehen muss, damit es genau in die Kamera sieht, außerdem wird die Position des Partikels beim nächsten Renderdurchgang berechnet. Je länger das Partikel lebt, desto langsamer wird es auch, um es realistisch aussehen zu lassen. Vom Tutorial konnten allerdings viele Dinge vereinfacht werden, da wir nur eine ganz einfache Explosion brauchten.

Jede vorhandene Lichtquelle im Spiel ist mit dem 3D-Lensflare-Effekt ausgestattet. Dieser Effekt wurde ebenfalls von einem NeHe – Tutorial #44 (http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=44) angeregt, da zuvor noch nie einen programmiert haben. Der Effekt wird nur dann gerendert, wenn die Lichtquelle im ViewFrustum ist, und sie durch nichts verdeckt wird. Dieser Test auf Verdeckung wurde über den z-Buffer realisiert. Genau wie beim Partikelsystem wurden stets zur Kamera ausgerichtete Billboards für die Flares verwendet. Die Lichtquelle selbst besteht aus einem Kreuz und einem großen Schimmer rund herum. Um einen realistischeren Lensflare-Effekt zu erhalten werden folgende Berechnungen durchgeführt, um das ganze 3D wirken zu lassen: in Kamerablickrichtung wird im selben Abstand, wie die Kamera von der Lichtquelle entfernt ist, ein Schnittpunkt berechnet. Auf der Line zwischen Lichtquelle und diesem Schnittpunkt werden nun in bestimmten Abständen mehrere kleinere Flares (im Wesentlichen Kreise) gezeichnet. Diese Texturen haben ebenfalls einen Alphakanal. Die Abstände und Größen dieser Flares wurden ursprünglich nur zum Testen vom Tutorial übernommen. Da diese Werte aber dann auch gut in unserem Spiel gepasst und funktioniert haben, wurden sie unverändert so belassen. Sollte man sie doch ändern wollen müsste man einfach nur 4 Werte ändern, also nichts tragisches.

ViewFrustum Culling

Schon für den Lensflare-Effekt wurde ein ViewFrustum – Test benötigt. Als dieses implementiert war haben wir es gleich zu ViewFrustum Culling weiterentwickelt. Und zwar wird um das ViewFrustum herum ebenfalls eine Achsenparallele BoundingBox gelegt. Vor dem Rendern wird mit der schon bestehenden Kollisionserkennung überprüft, ob der zu rendernde Node im Frustum liegt oder nicht. Nur wenn er innerhalb der BoundingBox des ViewFrustum liegt, wird er auch gerendert.

Da unser Spiel allerdings nur relativ wenig Polygone enthält, kann es durchaus sein, dass aktiviertes ViewFrustum Culling keine wesentlichen Verbesserungen bringt.

Experimentieren mit OpenGL

Natürlich haben wir auch versucht, mit OpenGL ein bisschen zu experimentieren. Diese Experimente sind mit den Funktionstasten aktivierbar bzw. deaktivierbar und als kurze Bestätigung wird eine Debugmessage am oberen linken Bildschirmrand ausgegeben, die nach ein oder zwei Sekunden wieder verschwindet. Im Folgenden werden die Funktionstasten und ihre Implementierung beschrieben:

F2 – Framerate anzeigen ein/aus
F3 – Wireframe Modus ein/aus (in diesem Fall werden natürlich auch die Polygone der Debugmessages als Drahtmodell ausgegeben)
F4 – Texturqualität verändern (Nearest Neighbour/Bilinear)
F8 – ViewFrustum Culling ein/aus
F9 – Blending/Transparenzen ein/aus

Animierte Objekte

Schaf und Wolf wurden in Maya animiert. Wir konnten die Animation bloß nicht in unsere Datenstruktur reinladen (siehe unten).

Probleme

Wir haben auch versucht, unsere Objekte zu animieren (bzw. diese Animationen aus dem FBX-Format herauszulesen). Wir konnten uns allerdings nur an den Beispielprogrammen des FBX-SDKs orientieren und dort wurden Animationen direkt aus dem FBX-File herausgelesen. Wir mussten allerdings die Animationen in unsere eigene Datenstruktur laden und dafür gab es weder Beispiele beim SDK, Tutorials oder sonstige Referenzen, wo wir nachsehen hätten können. Die SDK-Dokumentation gab darüber keinen Aufschluss und im Tutorial wusste auch niemand Bescheid.

Der dritte geplante Effekt hat bis zur Abgabe nicht funktioniert und uns außerdem die Framerate aber sowas von in den Keller getrieben, dass wir den Code wieder rausgenommen haben.

Natürlich wurde auch versucht, die Geschwindigkeit mit VertexArrays bzw. VertexBuffer Objects aufzubessern. Im Tutorial wurde uns bei der zweiten Abgabe gesagt, wir sollen unsere Datenstruktur mal aufbauen ohne dies (und andere Dinge) zu berücksichtigen, weil wir darauf achten sollten, dass „unser Hochhaus nicht einstürzt“. Als wir diese Funktionalität dann im Nachhinein einbauen wollten kamen wir drauf, dass dies mit unsere Datenstruktur nicht vereinbar war und wir das komplette Rendering umstellen hätten müssen, wofür uns dann die Zeit nicht mehr gereicht hatte (außerdem hatten wir Angst, alles bisher funktionsfähige kaputt zu machen).

Plädoyer

Wir haben zum ersten mal ein Spiel unter OpenGL programmiert. Bei dieser Übung haben wir einiges gelernt, unter anderem auch uns selbst zu helfen. Daher haben wir auch sehr viel Zeit in diese Übung investiert, mehr, als wir uns am Anfang eigentlich vorgenommen hatten. Wir haben auch viel experimentiert und waren über jeden noch so kleinen Schritt, der endlich funktioniert hat, froh. Leider ist uns am Ende die Zeit zu knapp geworden und wir mussten auf einige Dinge wie Animation, Schatteneffekte und flüssige Graphik verzichten. Wir hoffen aber, dass sich unsere Mühe (vor allem auch in Hinblick auf die Note) gelohnt hat und wir uns die ein oder andere Nächte nicht umsonst (schlaflos) um die Ohren geschlagen haben.

Bibliotheken

Es werden die OpenGL Utilities GLUT für Win32 verwendet (http://www.xmission.com/~nate/glut.html). Zum Laden von Bildern (für Texturen) wird das DevIL-SDK (http://openil.sourceforge.net/) verwendet, zum Laden der Models aus FBX das FBX-SDK (http://www.autodesk.com).

Das FBX-SDK kann nur über das mitgelieferte Setupprogramm installiert werden. Im Projekt müssen lediglich die Pfade bekannt gemacht werden. Eingebunden wird es mittels `#include <fbxsdk.h>`. Rendermethoden werden keine daraus verwendet (werden auch nicht vom SDK zur Verfügung gestellt). Externer Code bleibt unverändert.

Als Programmierumgebung wurde Visual C++ (Teil von Visual Studio .NET 2005) verwendet, und daraus natürlich auch die includierten Bibliotheken für vorgefertigte Datenstrukturen wie Hashmap, LinkedList o.ä.

Für die Objektmodellierung wurde Maya verwendet, mit dem wir die Modelle als FBX-Datei exportiert und über das FBX-SDK mit selbst implementiertem Modelloader in unsere Datenstruktur eingelesen haben.