CG 2/3 LU SS 2006

Duckrazor

Dokumentation

3. Abgabe

Stefan Hepp, 0026640, E535

Michael Preisel, 0326209, E700

Storyboard - Intro

Die Vorbereitungen für das legendäre Entenrennen sind im vollen Gange. Auf einer eigens angelegten Teststrecken gilt es, möglichst schnell ins Ziel zu kommen, und dabei Hindernissen auszuweichen und Strömungen auszunutzen.

Wer gute Zeiten fahren will, darf nichts dem Zufall überlassen. Aus diesem Grund hast du - um jeden Zweifel an deinem Sieg auszuschalten - über die ganze Teststrecke diverse Hilfsmittel versteckt, die dich mit ausreichend Power versorgen sollen.

Installation

Alle für das Spiel benötigten Dateien befinden sich im Ordner \bin. Um das Spiel zu starten muss die Datei duckrazor.exe gestartet werden. Das Spiel kann jederzeit mit ESC beendet werden (Achtung: CAPS LOCK darf nicht aktiv sein).

Um das Spiel von den Sourcen zu kompilieren, muss die Projektdatei Duckrazor.sln in \src mit Visual Studio 2003 geöffnet werden.

Kurzbeschreibung

Derzeit ist ein einfaches Level implementiert, das durchfahren werden kann. Über eine eigene Implementierung der Fahr-Physik kann man am Wasser schwimmen, untertauchen und springen, sowie mit dem Terrain kollidieren.

Ziel ist es, möglichst schnell ins Ziel zu kommen. Wenn man mit diesem Ziel nichts anfangen kann, kann man es sich auch zum Ziel machen, möglichst viele Bananen einzuheimsen, bevor man im Ziel angelangt ist. Es sind ein paar Dutzend im ganzen Gelände versteckt.

Weiters gibt es noch ein paar andere Spezial-Items, die weiter unten beschrieben werden.

Steuerung

Taste	***Funktion***	***Bemerkung***
WS	Vorwärts/Rückwärts	
AD	Seitlich	
MOUSE MOVE	Drehung	Um senkrechte Achse
SPACE	Springen	
LEFT SHIFT	Tauchen	Nach dem Loslassen taucht die Ente wieder auf
R	Zurück zum Start	Der Timer wird nicht angehalten, Items bleiben erhalten.
Q	Vorheriges Item	Kann auch mit Mausrad gewechselt werden
E	Nächstes Item	Kann auch mit Mausrad gewechselt werden
MOUSE LEFT	Item verwenden	Auch zum Spiel starten am Beginn

Taste	*Funktion*	*Bemerkung*
ESC	Spiel beenden	
C	Zum Ende springen	Cheat-Key um in den Zieleinlauf zu springen (Nur für Test-Zwecke)
T	Fullscreen/Window	
U	Mouse Grab ein/aus	Nützlich zum Debuggen
F2	FPS Counter	Derzeit nur auf der Konsole
F3	Toggle Shading	Point/Wireframe/Flat Shaded/Smooth Shaded

Spielablauf

Hilfsmittel, mit denen man das Level bewältigen kann:

Health:

Da Duckrazing ein sehr gefährlicher Sport ist, liegen im Level Erste-Hilfe-Koffer verteilt. Zu leicht stößt man in einem unaufmerksamen Moment am Ufer an und verletzt sich dabei auch noch! Die Koffer können, nachdem sie aufgesammelt wurden, jeweils 25% der Lebensenergie zurückgeben.

Oxygen Bottle:

Mit dem Sauerstoffvorrat kann man längere Zeit unter Wasser aushalten. Wenn man auftaucht, regeneriert sich der Sauerstoffvorrat jedoch immer wieder automatisch. Lange Tauchphasen sind aber wohl nötig, wenn man die eine oder andere Banane verspeisen will.

Pulso Drive:

Ein sehr mächtiges Werkzeug, mit dem durch geschicktes Springen oder durch Auffahren auf den Rand auch kurze Flugstrecken zurückgelegt werden können.
Es ist sogar möglich, das Level abzukürzen, allerdings ist Vorsicht geboten, da man auch mitten am Terrain landen kann und sich dann schwere Verletzungen zuführt. Wenn man am Terrain liegen bleibt, muss man mittels der 'R' Taste zurück zum Start. Dabei wird nichts an den Health-Punkten oder dem Item-Bestand geändert. Will man diese Höllenmaschine bändigen, so ist es am besten, das Triebwerk unter Wasser zu starten. Durch den höheren Widerstand hält man es so besser unter Kontrolle. Mit mehreren Pulso-Anwendungen hintereinander läßt sich übrigens eine ordentliche Flughöhe erreichen. Das ist zwar ziemlich lustig, führt aber nicht schneller zum Ziel.

Banana:

Wer viel strampelt, will auch verpflegt werden. Offensichtlich hat ein Luftfrachter eine ganze Fuhre Bananen verloren, die jetzt überall im Gelände herumkugelt. Bananen werden umgehend aufgefuttert und bringen jeweils 5 fette Bonuspunkte.

Aufblasbare Tore:

Im Streckenverlauf gibt es drei Gabelungen, damit man sich auf Anhieb zurechtfindet, wurde an den jeweils richtigen Abzweigungen Tore aufgestellt.

Unmotivierte Enten:

Es befinden sich einige Enten im Level, denen das Training offenbar zu sehr zugesetzt hat. Diese sind jedoch recht freundlich und quaken einfach vor sich hin.

Tipp:

Auch wenn es für einen Anfänger so erscheinen mag, man ist der Strömung nicht hilflos ausgeliefert! Oft sind enge Kurven leichter zu bewältigen, wenn man zusätzlich rückwärts watschelt oder gerade ein Pulso-Triebwerk laufen hat. Manche Passagen sind evtl. auch mit einem kurzen Tauchgang leichter zu bewältigen.

Implementierung

Für die Implementierung wurde der Code in drei Teile aufgeteilt:

Library:

In einer Library wurden verschiedene, oft verwendete Hilfsklassen gesammelt, wie verschiedene Shared-Pointer, Matrix- und Vektor-Templateklassen oder Konstanten.

Game-Engine:

Die Engine stellt alle Klassen zur Verfügung, um einen Scenegraph aufzubauen und zu rendern, allerdings auch darüber hinausgehende Funktionalität wie Input-Events-Generierung oder Loader um Texturen und Geometrien zu laden. Außerdem werden Sounds verwaltet und Spezial-Scenegraph Nodes wie Terrain oder Wasser sowie eine Partikel-Engine zur Verfügung gestellt.

Duckrazor:

Duckrazor bedient sich der Engine, um ein Level als Scenegraph aufzubauen und mittels Modifier zu animieren. Ein Mini-Windowmanager kümmert sich dabei um das Weiterreichen der Inputevents an das Level oder fängt Spezialtasten wie Framecounter oder Quit-Events ab.

Umgesetzte Features

Physik, Bewegungen

Für die Bewegung wird ein Kräftemodell für die Ente berechnet. Verschiedene Kräfte wie Auftriebskraft, Antriebskraft, Gravitation oder Wasserwiderstand werden verwendet, um die Bewegungen der Ente zu simulieren. Anhand der Position der Ente wird vom Wasser eine Strömungsrichtung sowie ein Normalvektor abgefragt, die zusätzlich in die Berechnung einfließt.

Um Probleme bei der Berechnung bei langen Framezeiten zu vermeiden, wurde ein einfacher iterativer Solver implementiert. Bei Framezeiten von mehr als 0.1s werden die Berechnungen mehrfach mit kürzeren Zeiten durchgeführt um starke Sprünge zu vermeiden.

Alle Animationen in der Szene (Ente, Kamera, Items) erfolgen über Modifier, die die Transformationen der Knoten im Scenegraph pro Frame verändern können. Für die Items wird z.b. ein einfacher Modifier verwendet, der die Rotation um die Y-Achse kontiuierlich (abhängig von der Zeit pro Frame) erhöht. Für die Bewegung der Kamera kommt ein Modifier zum Einsatz, der stets versucht, die Kamera mit einem gewissen Abstand und einer Verzögerung hinter die Ente zu bewegen.

Für Kollisionen mit dem Terrain wird ebenfalls eine eigene Implementierung verwendet und in die Animation der Ente eingerechnet. Collision-Tests mit Items und dem Start und Ziel werden über einen Boundingbox-Test-Traverser durchgeführt. Um eine Kollisionsbehandlung mit anderen Objekten zu ermöglichen, sollte eine bestehende Physik-Library eingebunden werden, konnte aus Zeitgründen aber leider nicht mehr

umgesetzt werden.

Für die Höhenberechnung der Wasserhöhe werden die irregulären Vierecke mit einer beim Laden vorberechneten Transformationsmatrix auf ein einfacheres Quadrat abgebildet, um die Interpolierung einfacher berechnen zu können.

Terrain

Das Terrain wird über eine Heightmap berechnet. Die Auflösung des Meshes des Terrains ist von der Auflösung der Heightmap unabhängig. Für die Positionen als auch für die Kollisionsabfrage werden die Höhenwerte zwischen zwei Pixel linear interpoliert.

In Hinblick auf Frustrum Culling erlaubt die Terrain-Klasse auch das Aufteilen des Meshes in mehrere quadratische Meshes. Dieses Feature wird derzeit allerdings nicht verwendet, weil ohne LOD die Anzahl der zu rendernden TriangleStrips stark steigt, wodurch die Framerate stark abhängig von der Blickrichtung ist.

Beleuchtung

Die Szene wird mit einer unendlich weit entfernten Lichtquelle beleuchtet, wobei das ambiente Licht verhältnismäßig hell ist, um eine helle Außenszene zu simulieren. Weitere Lichter können durch Einhängen in den Scenegraph eingebracht werden, werden aber momenan nicht benötigt. Der Skydome wird unbeleuchtet gerendert.

Alpha-Test

Für die Darstellung der Banner bei Start und Ziel, sowie für das Auffangnetz wird der Alpha-Kanal verwendet, um durchsichtige Stellen zu realisieren. Der Alpha-Kanal wird bei diesen Objekten allerdings nur für eine Screendoor-Transparenz verwendet, Pixel mit zu geringem Alpha werden von OpenGL über einen Filter ausgeblendet, eine Sortierung der Objekte ist in diesem Fall somit noch nicht notwendig.

Buffer

Alle Geometriedaten sowie alle Index-Arrays werden mittels OpenGL Vertex Buffer Objects im Speicher gehalten. Ebenso werden Texturen als Texture Objects angelegt und nur bei der Initialisierung oder bei Änderungen die Texturdaten in den Speicher übertragen.

Frustrum Culling

Für Frustrum Culling werden für alle Objekte Axis Aligned Bounding Boxen beim Laden erstellt und im Scenegraph verwendet. Bei Änderungen im Scenegraph werden die betroffenen Bounding-Boxen während des Update-Vorgangs neu erstellt. Beim Rendern werden die Bounding Boxen mit den Frustrum Planes getestet. Als einfache Anleitung dazu diente vor allem Mark Morley's Frustrum Culling Tutorial (Der Original-Link konnte leider nicht gefunden werden, der Artikel wurde aus dem Google-Cache geladen).

Texturen

Für Texturen wurde meistens Linear Filtering verwendet. Für manche Texturen wie z.B. beim Terrain werden zusätzlich MipMaps verwendet. Die MipMaps werden über GLU bei Bedarf automatisch generiert.

Für Multitexturing und Blending-Effekte wurden verschiedene Combiner-Funktionen wie Modulate, Decal, oder eine Modulation mit einer konstanten Farbe für schnelle Fading-Effekte implementiert.

Overlays

Für die Overlays wurden einfache Vierecke mit Texturen versehen. Die Texturen werden teilweise über Multitexturing zusammengebaut, teilweise liegen mehrere Vierecke übereinander.

Das Overlay ist ein eigener Scenegraph, der mit einer Ortogonal-Projektion abgebildet wird. Vor dem Rendern wird der Depth-Buffer zurückgesetzt, damit es zu keiner 'Interaktion' mit der eigentlichen Szene kommt.

Spezialeffekte

Alpha Blending

Zusätzlich zum Alpha Test wurde eine echte Transparenz mit Alpha Blending und z-Order Sortierung implementiert. Dazu werden beim Rendern zuerst alle undurchsichtigen Objekte (auch Objekte mit Alpha-Test aber keinem Blending) normal gerendert. Danach wird der zBuffer readonly gesetzt und die transparenten Objekte gerendert, wobei sie vorher aufsteigend nach dem Abstand zur Kamera sortiert werden.

Der Abstand entsteht als Nebenprodukt des Frustrum Cullings. Der Test mit den Boundingboxen erfolgt, indem für alle Eckpunkte der Abstand zu den Ebenen des Viewing-Frustrums berechnet wird. Ist der Abstand für alle Planes > 0, so befindet sich der Punkt im Frustrum. Wenn als letztes mit der Near-Plane getestet wird, erhält man dadurch näherungsweise den Abstand zur Kamera (der Abstand entspricht nicht der Entfernung vom Punkt zum Augpunkt, sondern dem Normalabstand zur Ebene).

Für den Z-Wert wird der am weitesten entfernte Punkt der Bounding Box herangezogen. Es könnte auch der näheste Punkt genommen werden, allerdings führt das zu Problemen mit dem Wasser, da die Bounding Box prinzipiell die Near-Plane schneidet und somit immer als letztes gerendert werden würde.

Partikel

Als weiterer Effekt wurden einfache Partikel implementiert. Für die Items werden Partikel erzeugt, deren Geometrie von allen Items geshared werden, um Rechenzeit zu sparen.

Die Partikel werden von einem Emitter in einer Kugel zufällig erzeugt und über eine Update-Klasse abhängig von der Geschwindigkeit der Partikel um den Ursprung gedreht.

Für das Pulso-Triebwerk wird für den Rauch ein anderes Update-Modell verwendet, welches die Partikel abhängig von ihrer Geschwindigkeit weiterbewegt und kleine zufällige Kräfte pro Frame zusätzlich pro Partikel wirken lässt. Anders als bei den Items haben diese Partikel auch eine Lebensdauer, nach deren Ablauf sie gelöscht werden.

Für die Texturierung der Partikel mit Billboards wurde die GL_POINT_SPRITE_ARB Erweiterung von OpenGL 2.0 verwendet. Über die glPointParameter wird zusätzlich abhängig von der Entfernung die Größe der Partikel geändert.

Als einfache Anleitung für die Implementierung der Point Sprites diente der Point-Sprite Democode von http://www.codesampler.com/oglsrc/oglsrc_6.htm.

Terrain Multitexturing

Für die Texturierung des Terrains wurden verschiedenste Texturen in unterschiedlichen Auflösungen verwendet. In Summe wurden 7 Texturen in 4 Texture-Units untergebracht, um den gewünschten Effekt zu erzielen. Über die Textur-Combine Methoden wurden die verschiedenen Überblendungen realisiert.

Als Basis dient eine Textur mit Farbwerten, die in den Vertex Colors untergebracht wurde. In der ersten Unit befindet sich eine gräuliche Detail-Textur, die mit den Vertex-Farben moduliert wird. Damit werden unterschiedliche Gesteinsfärbungen erzeugt.

In der zweiten Unit befindet sich eine weitere Detail-Map, die für das Gras verwendet wird. Die Textur wird abhängig von einer Coverage-Map aufgebracht. Dazu wurde der Grau-Wert der entsprechenden Map in den Alpha-Kanal der Vertex-Colors geladen und mit der GL_INTERPOLATE Funktion mit bereits berechneten Textur je nach Alpha-Wert kombiniert.

Die dritte Unit enthält ebenfalls eine Detail-Map, die für den Boden des Flussbetts verwendet wird. Die Berechnung erfolgt wie vorher, es wird allerdings der Alpha-Wert der Textur in der vierten Unit verwendet.

In der vierten Unit wird eine beim Laden erzeugte Textur verwendet, die im Alpha-Kanal den Grauwert der Coverage-Map für die dritte Unit gesetzt bekommt. Die Farbwerte werden als Lightmap verwendet. Dazu wird die Farbe der vorhergehenden Unit mit den Farbwerten der Textur multipliziert. Als Alphawert des Ergebnisses dieser Unit wird konstant 1.0f verwendet.

Die Lightmap wird ebenfalls dynamisch generiert, kann aber auch durch eine beliebige Textur ersetzt werden. Für die Textur werden die Normals des Terrains berechnet und abhängig von der Neigung ein Farbwert gesetzt. Je höher die Neigung ist, desto dünkler ist die Map. Dadurch wird simuliert, dass stärkere Neigungen auf der der Lichtquelle abgewandten Seite im Schatten liegen. Die Simulation berücksichtigt allerdings nicht die Richtung der Lichtquelle, weshalb die Schatten auf allen Seiten gleich sind.

Als Alternative kann man auch die Heightmap als Lightmap verwenden, dadurch wird simuliert, dass es im Tal finsterer ist. Allerdings wird hier der Schattenwurf nicht berücksichtigt.

Als Anregung für das Slope-Mapping diente 'Focus on 3D Terrain Programming' von Trent Polack, als Anregung für das Kombinieren von Farb-Maps und Coverage-Maps der Artikel unter http://www.cs.auckland.ac.nz/~jvan006/multitex/multitex.html. Die dort beschriebene Methode um mittels der DOT3 Combiner-Funktion die Texturen zu modulieren, wird allerdings nicht verwendet.

Verwendete Programme

Programmierung:

Für die Programmierung wurde vorranging Visual Studio 2003 .Net Professional eingesetzt. Teilweise wurde außerdem Eclipse mit C++ Plugin verwendet, vor allem zur Verwaltung des Quellcodes und der Spieldaten mit Subversion, da die Visual-Studio Plugins nicht besonders ausgereift sind.

Heightmap:

Die Heightmap wurde mit Terragen erstellt.

Terragen hat sich für unsere Anwendungen als hervorragendes Tool zum schnellen und intuitiven Erstellen von Heightmaps erwiesen. Besonders die übersichtliche Bedienung half mit, schnell ein ästhetisch ansprechendes Gelände modellieren zu können.

Die Möglichkeit, das Wasserniveau zu definieren, in Echtzeit zu rendern und das Gelände im Renderer durchfahren zu können war für unsere Art von Spiel besonders hilfreich.

Während es in Grafikprogrammen (Photoshop, Gimp, ...) nicht leicht möglich ist, ein Werkzeug so zu benutzen, daß es die Helligkeit jeweils um konstante Werte ändert, formt Terragen mit einer speziellen Brush ansprechende Kuppen und Senken, die natürlichen Geländeformen entsprechen. Mit jeder Änderung werden weiters die Grauwerte auf den nutzbaren Wertebereich (8 Bit) aktualisiert.

Modelle:

Die Modelle wurden teilweise mit in Blender, teilweise mit Milkshape 3D erstellt bzw. vorhandene editiert und abgewandelt. Die Nachbearbeitung und Texturierung erfolgte in Milkshape 3D. Um die Modelle von Blender in Milkshape3D zu importieren wurden sie als Wavefront Obj bzw. 3DS zwischengespeichert.

Die Texturen wurden mit Adobe Photoshop erstellt und als PNG oder JPG exportiert.

Sound:

Die Sounds wurden mit Cool Edit teilweise nachbearbeitet oder selbst aufgenommen und gemischt.
Der Tusch beim Finish wurde als Auftragskomposition extra für dieses Spiel von einem Profi erstellt, herzlichen Dank an Klemens Löwenstein.

Verwendete Libraries

SDL:

Für die Fensterverwaltung und das Input-Polling wurde SDL verwendet. Mithilfe der SDL OpenGL Funktionen wurde ein OpenGL Fenster mit Doublebuffering erzeugt. Das Buffer switchen erfolgt über die SDL Funktionen.

Ebenso wurde das Abfragen von Window-Events und Input-Events, sowie das Grabben des Mouse-Cursors über SDL implementiert.

Url: http://www.libsdl.org/

DevIL:

Für das Laden von Texturen (sowie der Heightmap) kam DevIL zum Einsatz. Es werden nur die Methoden zum Laden von Bildern verwendet. Weiterführende Funktionen wie das Zusammensetzen von Texturen für die Overlays oder das Terrain, sowie das Laden in OpenGL wurden selbst implementiert.

Url: http://openil.sourceforge.net/

Glew:

Um auf die verschiedenen OpenGL-Extensions zugreifen zu können, wurde Glew verwendet. Dadurch kann in Laufzeit auf vorhandene OpenGL-Erweiterungen überprüft werden.

Url: http://glew.sourceforge.net/

Ms3d File Loader:

Für das Laden der Milkshape3D Modelle wurde ein eigener Model-Loader implementiert, welcher auf dem NeHe-Gamedev Tutorial Lession 31 basiert. Der in dem Tutorial beschriebene C-Quellcode wurde in einen eigenen Loader eingebunden und mit den ausgelesenen Daten Engine-eignene Geometrie-Klassen befüllt.

Url: http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=31

FMOD Ex Sound:

Für den Sound wurde FMOD Ex verwendet. Die Sounds werden als Streams geladen und über die Software-3D Funktionalität von FMOD ausgegeben.

Url: http://fmod.org/

Known Bugs

Beim Kompilieren des Programms im Release-Mode entsteht reproduzierbar eine 'Unknown Win Exception' beim Starten, wenn das Programm nicht aus der IDE heraus gestartet wird. Deswegen wurde das Programm im Debug-Mode kompiliert. Um dennoch keine zu starken Performance-Verluste zu bekommen, wurden im Debug-Mode die Realtime-Checks abgeschalten und die Code-Optimierung angeschalten. Daher werden die Variableninhalte und die Zeilenpositionen beim Debuggen teilweise falsch angezeigt.