CG 2/3 Übungen, SS 2005

Abschnitt Nr.: 3

Mitsch, Matthias, 935, 7604717, muftimufti@gmx.at
Wagner, Christian, 532, 0026014, e0026014@student.tuwien.ac.at
Wagner, Herfried, 935, 0025861, hkwmvp@web.de

„The Krunscher“ – Technik

Das Spiel verwendet OpenGL zur Darstellung der Szene. Die Audioausgabe wurde mit der Bibliothek FMOD realisiert. Weiters wird GLUT verwendet. Weitere Bibliotheken kommen nicht zum Einsatz.

Implemetierung

Datenstruktur

Als Datenstruktur zur Repräsentation der Szene kommt eine Baumstruktur (Klasse `CTree`) zum Einsatz. Dadurch wird es ermöglicht, Elemente der Szene hierarchisch zu strukturieren. Das Hinzufügen der Elemente zur Szene erfolgt in der Klasse `CWorld`, in der die gesamte Welt inklusive Klängen und Musik verwaltet wird. In der Klasse `CEngine` werden das rekursive Zeichnen der gesamten Szene, die Animation der Objekte sowie die Erkennung von Kollisionen angestoßen. Gemeinsame Basisklasse aller Szenenobjekte ist die Klasse `COb`-ject, von der unterschiedliche weitere Basisklassen abgeleitet werden. Beim Entwurf wurde darauf geachtet, die Integration zusätzlicher Objektklassen möglichst einfach zu machen.

Die Erkennung von Kollisionen erfolgt in den meisten Fällen durch Umschreiben der Objekte mit einer Kugel. Dies erwies sich für Hindernisse wie Bäume und andere Spielfiguren als ausreichend.

Spielfiguren können sich in unterschiedlichen Zuständen befinden. Sie werden dem Zustand entsprechend animiert und positioniert. So werden bspw. eingesammelte Mädchen und Gifthühner neben der Statusanzeige platziert. Gesammelte Objekte werden in Listenstrukturen verwaltet.

Einige statische Szenenelemente wie die Skybox (Klasse `CSkybox`), das Terrain (Klasse `CTerrain`) und die Mauern (Klasse `CWall`) werden optional mit Hilfe von Displaylisten gezeichnet. Dadurch kann die Performanz und damit die Bildfrequenz erhöht werden. Bildfrequenzanzeiger und die Übersichtskarte sind nicht als eigene Objekte realisiert, sondern werden direkt nach Abschluss des Durchmusterns des Szenenbaums gezeichnet.

Kameramodell

Die Kamera (Klasse `CCamera`) befindet sich auf Augenhöhe des Spielers, der in die Welt des Spiels blickt. Über die Tastatur und die Maus kann der Benutzer seine Position in der Welt und damit jene der Kamera verändern. Die Kamera besitzt, wie auch die animierten Modelle, eine Position, eine Geschwindigkeit und einen Rotationswinkel. In der Hauptschleife des Spiels wird der Status der Steuerungstasten abgefragt und Geschwindigkeit und Winkel der Kamera in Abhängigkeit von der vergangenen Zeit gesetzt. In jedem Durchlauf der Spielschleife wird die Position der Kamera neu berechnet und die Kamera an die Position bewegt. Zudem kann eine Bewegung der Kamera optional unterbunden werden – das ist bspw. dann der Fall, wenn die Kamera sich gegen ein Hindernis bewegt.

Texture Mapping

Die meisten Elemente in der Welt des Spiels sind texturiert. Folgende Liste bietet eine Übersicht über die im Spiel eingesetzten Objekte und Charakteristika von deren Texturierung:

- Das Terrain wird mittels Multitexturing (ARB-Erweiterung) mit zwei übereinander gelegten Texturen versehen. Die Grobtextur wird mittels eines Algorithmus erstellt, der auf Basis der Höhe an einem bestimmten Punkt und der Steigung unter Einbeziehung des Zufalls die Textur an jedem Punkt bestimmt. Die Grobtextur überdeckt das gesamte Terrain, d.h., es werden immer entsprechende Ausschnitte aus der Grobtextur auf einzelne Vierecke abgebildet. Jedes der Vierecke wird weiters mit der Feintextur versehen, die eine Erd- bzw. grasähnliche Oberflächenstruktur bewirkt. Die beiden Texturen werden transparent übereinander gelegt.
- Zum räumlichen Abschluss der Welt nach außen hin wird eine Skybox verwendet, die aus 6 Seitenflächen besteht, welche einen Quader darstellen. Die Skybox ist an der Innenseite mit einer Himmel- bzw. Bergtextur versehen, Kantenübergänge der aneinandergrenzenden Flächen werden mittels `GL_CLAMP_TO_EDGE` schwerer sichtbar gemacht. Auf die Wand hinter dem Krunscher ist eine Steintextur aufgebracht.
- Bäume sind ebenfalls texturiert.
- Hinter dem Krunscher befindet sich eine Steinwand, die mit einer Steintextur versehen ist. Zusätzlich ist dazu auch noch einem NormalMap aufgebraucht um Tiefeninformation aus einem Flachen Bild generieren zu können und so der Wand einen realeren Eindruck zu vermitteln.
- Der Pool besteht ebenfalls aus texturierten Objekten.
- Die Modelle der animierten Spielfiguren (Krunscher, Mädchen und kleine Monster) sind ebenfalls texturiert.
- Nicht texturiert sind auch die Kugeln, welche der Spieler aufsammeln kann.
- Die Übersichtskarte wird ebenfalls texturiert – als Textur kommt dabei die Grobtextur des Terrains zum Einsatz.
- Die Balken werden ja nach Stand des Spieles unterschiedlich texturiert.
- Die Gewonnen- und Verloren-Screens sind ebenfalls Texturen.
- Nicht texturiert werden die „Anzahlbalken“ im HUD.

Modelle

Die Modelle der Figuren wie die verschiedenen Monster, Hühner oder Mädchen sind aus diversen Seiten vom Internet geladen und etwas umgebaut so dass sie von den Bewegungen und Texturen zu unserem Spiel passen.

Alle anderen Modelle wurden mit Maya und 3DSMax kreiert, als 3DS-File exportiert und mit einem geeigneten Importer ins Spiel geladen. Dazugehörige Texturen wurden ebenfalls per Hand erstellt.

Beleuchtung / Materialien

Über der gesamten Szene liegt ein leichter Nebel, der die Landschaft realistischer aussehen lässt, insbesondere dann, wenn der Spieler einen Berg erklommen hat. Nebel und Beleuchtung werden in der Klasse `CWorld` festgelegt.

Eine ambiente und diffuse Lichtquelle wurde definiert, wobei das Terrain so beleuchtet wird, dass Berge nicht vollständig mit voller Intensität angestrahlt werden.

Für einige Objekte werden explizit Materialeigenschaften festgelegt. Die Oberfläche der Bäume weist bspw. eine geringe Reflexion auf. Kugeln, die aufgesammelt werden können, werden zeitweise transparent dargestellt, ebenso ist die Wasseroberfläche transparent.

Features

Hintergrundsound wird während des gesamten Spieles dargeboten. Sammelt man Dinge auf, werden jeweils zu den verschiedenen Objekten auch verschiedene Sounds wiedergegeben. Sammelt man ein Gifthuhn auf, wird ein anderes Stück gespielt, als wenn man Opfer einsammelt.

Effekte

Für die dritte Abgabe wurden die folgenden Effekte implementiert:

Wunderkerze

Die Wunderkerze wurde mit CG implementiert, sie besteht aus drei Partikelsystemen, die Partikel werden durch Point Sprites (ARB-Erweiterung) dargestellt und nach Art einer Wunderkerze versprüht. Dazu wird ein Array von Partikeln erzeugt, denen Parameter wie Geschwindigkeit, Entstehungszeit und Position mitgegeben werden. In jedem Rendering Durchgang wird eine zufällige Anzahl dieser Partikel neu erzeugt, um einen Effekt wie beim unregelmäßigen Abbrennen einer Wunderkerze zu imitieren. Im CG Programm wird dann die Flugbahn aus Geschwindigkeit, Beschleunigung und Zeit errechnet. Der Effekt erscheint, um eine kleine Feier anzudeuten, wenn der Spieler drei Mädchen gefangen hat direkt vor dem Spieler, der Spieler ist gezwungen, die Feier anzusehen, er kann sich bis zu deren Ende nicht wegbewegen.

Wasser

Mit CG implementiert. Dargestellt wird eine transparente, mit CG animierte Wolkentextur, die Spiegelungen in der Wasseroberfläche simulieren soll. Die Wasseroberfläche besteht aus einem QUAD Gitter, Die y- Werte werden interpoliert in einer Sinuswelle angehoben.

Wir haben kurz vor der Abgabe bemerkt, dass falls das Spiel von der EXE gestartet wird, der Wassereffekt nicht sichtbar ist. Um den Effekt zu sehen bitte einfach den Sourcecode kompilieren. Egal ob als Release oder als Debug. Im Ordner bin gibt es zusätzlich auch noch eine Datei namens „Krunscher_Debug.exe“ mit der die Debug-Version des Spieles gestartet werden kann mit der das Wasser korrekt dargestellt wird. Allerdings ist dadurch das gesamte Spiel etwas langsamer.

Schielen

Bei Kollision mit dem Hindernis beginnt der Spieler zu schielen und ist dadurch in seiner Aktionsfähigkeit behindert. Das Hindernis besteht aus einer transparenten gluSphere, texturiert mit dem Bild eines schielenden Monsters. Der Effekt selbst wird durch verdoppeltes Rendern des Terrains und der nahe liegenden Bäume erreicht, wobei das Terrain transparent dargestellt wird.

Bump Mapping

Mit CG implementiert. Die gemauerten Wände im Spiel werden mittels Bump Mapping optisch aufgewertet. Zu diesem Zweck werden Normal Maps der aufgebrachten Texturen benutzt und mittels Multitexturing zusammen mit der Ziegeltextur gerendert.

Dazu wurde im Hauptprogramm eine Matrix für den tangent space errechnet und in einem vertex Programm der Lichtvektor ausgerechnet und in den tanget space transformiert. Die endgültige Farbe ergibt sich aus dem dot Produkt von Lichtvektor und Normale x ursprüngliche Texturfarbe.

Die Normalmaps wurde ursprünglich (in aufwendigen Prozeduren) händisch generiert. Erst später entdeckten wir das auch ein Photoshop Plug-in auf der Nvidia-Site (developer.nvidia.com) zur Verfügung steht mit dem Normal Maps um einiges einfacher und effizienter kreiert werden konnten. Die zuletzt im Spiel verwendeten Normal Maps wurden bereits mit diesem Plug-in kreiert.

Um die Bump Maps deutlicher erscheinen zu lassen, wird die Lichtquelle in einer Schleife um die Mauer bewegt.

Bump mapping wird im Moment nur für die Mauer hinter dem Pool verwendet.

Feuereffekt am Vulkan

Beim Feuereffekt auf dem Vulkan handelt es sich um eine Texturanimation. Das Feuer wird aufgebaut aus mehreren texturierten Rechtecken, die im Raum derart angeordnet werden, dass sie immer dem Spieler zugewandt sind. Die einzelnen Rechecke werden dabei so angeordnet, dass sie im Abstand leicht vom Benutzer weg verschoben werden. Dies erleichtert ein abstandsabhängiges Darstellen der einzelnen Feuerrechtecke, also die Ausgabe anhand des *Z*-Abstands des jeweiligen Rechtecks vom Benutzer.

Eine Aufspaltung der Rechecke ist nicht notwendig, da sich die Rechecke nicht überschneiden, sondern hintereinander angeordnet werden. Die einzelnen Rechtecke werden mit transparenten (Alphakanal) Texturen versehen und mittels Blending übereinander gelegt. Durch Wahl eines geeigneten Verknüpfungsmodus wird ein feuerähnliches Aussehen erwirkt.

Jedes Element besitzt eine bestimmte Lebenszeit und einen Ort, an dem es dargestellt wird. Läuft die Lebenszeit eines der Elemente ab oder bewegt sich dieses zu weit vom Feuerursprung weg, wird es zerstört und durch ein neues Element ersetzt. Es stehen drei verschiedene Flammentexturen bereit, die zufällig auf die Rechtecke aufgebracht werden.

Erdbebeneffekt am Vulkan

Der Effekt wurde auf einfache Weise durch zufällige Auf- und Abbewegungen und kleinere zufällige Seitenbewegungen der Kamera simuliert.

Stolpern über die kleinen Kampfmonster

Bei Begegnung mit den kleinen, in geordneter Formation durch die Welt laufenden, sonst relativ harmlosen Kampfmonstern stolpert der Spieler und wird an einen zufälligen Punkt geschleudert. Dort bleibt er kurz mit dem Gesicht nach unten liegen. Zur Implementierung gibt’s nicht viel zu sagen, sie ist trivial.

Statische Levels of Detail und Billboards

Bäume werden in der Landschaft plaziert und in Abhängigkeit von der Entfernung vom Betrachter in unterschiedlichen Detailstufen gezeichnet. Dazu ist es erforderlich, bei jedem Durchlauf der Szenenbeschreibung für entsprechende Objekte die Entfernung vom Betrachter zu ermitteln und anschließend das passende Modell zu generieren.

In den feineren Detailstufen werden Bäume als Billboards dargestellt, die aus mehreren Rechtecken bestehen, die im Winkel 180 / *n* (bei *n* Rechtecken) zueinander stehen. *n* ist dabei direkt proportional zum Abstand zwischen Baum und Betrachter. Bei Bäumen wird darauf geachtet, dass der Stamm aus mehreren der Rechtecksteilbilder besteht. Die aufgebrachten

Texturen besitzen binäre Transparenz, weshalb keine Sortierung nach dem *Z*-Abstand vom Betrachter vorgenommen werden muss. Die Ebenen in diesen Detailstufen werden nicht automatisch dem Betrachter zugewandt, sondern stehen in ihrem Drehungswinkel konstant in der Landschaft.

Auf der groben Detailebene wird lediglich ein mit einer binär transparenten Textur versehenes Rechteck dargestellt. Dieses wird immer so mit dem Benutzer mitgedreht, dass es normal auf die Sichtlinie zwischen Benutzer und horizontalem Objektmittelpunkt steht. Bei Texturen und Modellen musste darauf geachtet sein, dass deren Aussehen nach Möglichkeit rotationsinvariant ist, also unabhängig vom Blickwinkel ein annähernd gleiches Erscheinungsbild sichergestellt ist. Dies bewirkt, dass das Umschalten zwischen zwei unterschiedlich komplexen Objektrepräsentationen nicht so gut sichtbar ist.

Vorberechnung von Beleuchtungsinformation und Multitexturing

Zur Texturierung des Terrains wird Multitexturing benutzt, d.h., es werden mehrere Texturen übereinander gelegt. Neben der Detailtextur wird eine weitere Textur generiert, in die auch Beleuchtungsinformationen eingerechnet werden. Dies bewirkt, dass das „kantige“ Aussehen der Texturbeleuchtung pro Rechteck nicht sichtbar ist.