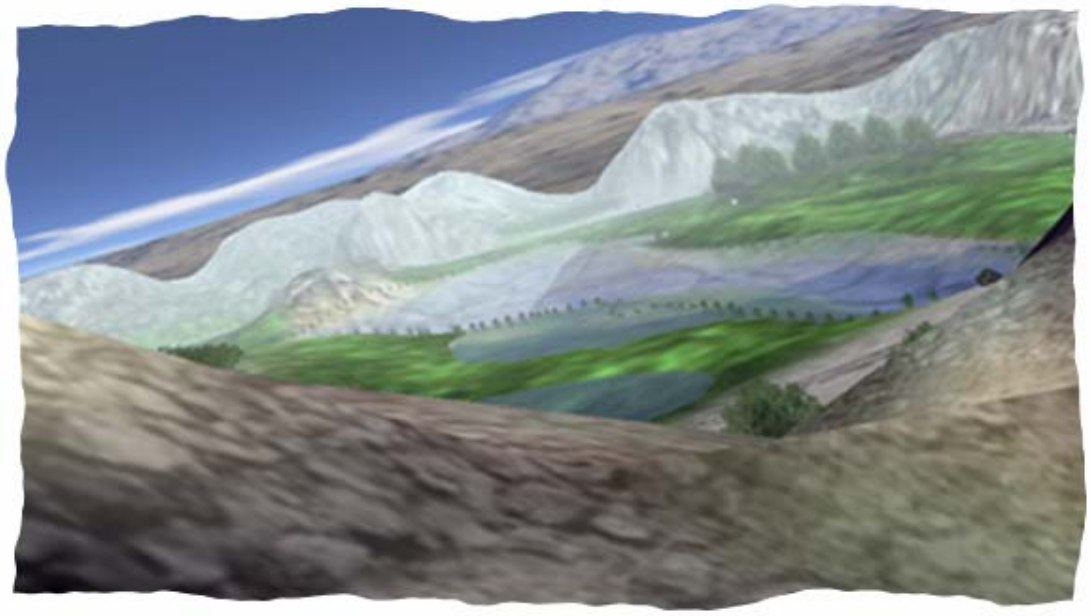
CrazyWheels
3. Abgabe



Harald Meyer - 0225448
Tobias Schleser - 0225349

1 Einleitung

CrazyWheels ist ein Rennspiel, bei dem Fahrzeuge, deren Fahrer Tiere/Kunstlebewesen sind, gegeneinander fahren.
Die Fahrzeuge können abgeschossen, und Punkte und Munition aufgesammelt werden. Es gibt diverse Gamemodes wie z.B. Trainingsfahrten, Rundenfahrten (z.B. 6- Runden) oder Timetrials.

2 Starten

Das Spiel wird über „bin/CrazyWheelsRel.exe“ gestartet. Unter „data/gamedata.xml“ können diverse Einstellungen vorgenommen werden (wie z.B. die Auflösung.). Die meisten Dinge können jedoch direkt im Spiel geändert werden.

3 Steuerung

Die Steuerung erfolgt über folgende Tasten (Standardeinstellung):

- c: Cameraview ändern
- s: schießen
- b: Pause
- v: 2D-Controlls ausschalten (währen dem Spiel)
- UP: beschleunigen
- DOWN: bremsen
- LEFT/RIGHT: lenken
- „SPACE“: Handbremse
- F1: Screenshot
- F2: Framerate
- F3: Wireframe
- F4: Texturqualität
- F5: Mipmapping
- F6: Vertex Arrays, Immediate Mode, VBOs
- F7: Display Lists
- F8: Frustum culling
- F9: Transparenz

Generell ist die Tastaturbelegung über die XML-Steuerdatei bzw. über das „Optionen“-Menü beliebig anpassbar.

4 Nichttriviale Objekte

Jedes Level hat eine globale Lichtquelle. Untexturierte und nur mit Material versehene Objekte sind z.B. die Bremslichter der Fahrzeuge, sowie einige aufsammelbare Objekte.
Das Terrain ist mit einer Colormap und einer Detailmap multitextured. Die Kombination erfolgt mit den OpenGL Combiner Funktionen.
Alle Objekte und das Terrain besitzen ein Material, Normal-, Tangential und Binormalvektoren.
Als Modelformat wurde das propiertäres Format „LR3D“ basierend auf dem Milkshape Binary Format entworfen. Dieses Format hat gegenüber Milkshape den Vorteil, dass die

diversen Daten für z.B. Bumpmapping bereits berechnet sind und dass keine zeitaufwendige Konvertierung der Struktur für Vertex Arrays notwendig ist. Zusätzliche Daten sind Tangenten, Binormale eine Normalmap und eine Heightmap.

Die Modelle wurden teilweise in Milkshape und 3D Studio Max 7 designed bzw. verändert, aber größtenteils aus dem Internet geladen.

5 Beschleunigung der Sichtbarkeitsberechnung

Es wurde simples View-Frustum-Culling implementiert. Jedes Objekt besitzt eine Boundingbox, die mit dem Frustum geschnitten wird. Liegt die Box außerhalb des Frustums, so wird das Objekt nicht gezeichnet. Das Abschalten des View-Frustum-Cullings führt zu einem deutlichen Performanceabfall.
Weiters wurde Hardware Occlusion Culling (teilweise) mit GL_ARB_occlusion_query implementiert. Da unsere Modelle jedoch nur sehr wenige Polygone haben, führt das Occlusionquery zu einem deutlichen Performanceverlust, weshalb dieses standardmäßig deaktiviert ist.

6 Transparenz-Effekt

Wir verwenden Transparenz für das Wasser, für die Particleengine und fürs Billboarding. Letzteres benötigt weiters Tiefensortierung die wir bis auf Modellebene umgesetzt haben.

7 Experimentieren mit OpenGL

Alle Anforderungen wurden entsprechend der 3. Aufgabenstellung implementiert. Tastenbelegung sieht im Punkt „Tastenbelegung“.

8 Effekte

Folgende Effekte wurden implementiert:

- Particle Engine: Die Fahrzeuge erzeugen Staubwolken. Weiters ist ein Feuer beim Abfeuern zu sehen. Außerdem ist die Explosion des Geschosses mit Particles umgesetzt.
- Reifenspuren: Jedes Fahrzeug hinterlässt Reifenspuren im Terrain.
- Transparenz: die Menüs und die Particles verwenden Alpha-Blending.
- Wasser
- Billboarding: die Particle Engine verwendet Billboarding
- Bump Mapping: die Steine sind gebump-mapped
- Rauch: abgeschossene Fahrzeuge rauchen
- Shadow Mapping
- Projective Shadows
- Gaussian Blur

Die genauen Links zum jeweiligen Effekt sind unter „Quellen“ aufgelistet.

8.1 Particle Engine

Die Particle Engine ist die Basis für diverse Effekte. Ein Partikel ist eine Textur, die auf einen Quad projeziert wird, wobei schwarze Bereiche transparent sind. Jeder Particle hat eine Lebensdauer, eine Geschwindigkeit (x,y,z), eine Position (x,y,z) und eine zeitabhängige Größe.
Die Farbe der Partikel hängt vom Effekt ab.

8.2 Rauch

Der Rauch basiert auf der Particle Engine. Die Partikel haben einen fixen Grauwert.

Abbildung 1 Rauch

Rauch tritt auf, wenn ein Fahrzeug mit weniger als 2 Lebenspunkten von einem Geschoß getroffen wird.

8.3 Feuer

Das Feuer basiert auf der Particle Engine. Die Farbe hängt von der Zeit (und somit von der Entfernung zum Abfeuerungszentrum) ab. Innen ist die Farbe gelb und je weiter man nach außen kommt, desto größer wird er rot-Anteil.

Abbildung 2 Feuer

Feuer tritt beim Abfeuern von Geschoßen auf.

8.4 Explosion von Geschoßen

Der Effekt basiert auf der Particle Engine. Ein Geschoß explodiert, wenn es auf ein Objekt bzw. das Terrain auftrifft, oder wenn die Lebensdauer des Geschoßes überschritten wird. Trifft ein Geschoß auf das Terrain auf, so hinterlässt es einen Fleck.

Abbildung 3 Explosion eines Geschoßes

8.5 Staub hinter den Reifen

Auch dieser Effekt beruht auf der Particle Engine. Wenn sich ein Fahrzeug bewegt, so hinterlässt es eine Rauchspur. Die Bewegung dieses Rauchs ist entgegengesetzt der Fahrtrichtung. Die Farbe der Partikel wird durch einen Texture-Lookup in die Terraintextur ermittelt.

Abbildung 4 Reifenstaub

8.6 Reifenspuren

Die Räder der Fahrzeuge hinterlassen Spuren. Diese werden durch Multiplikation der Farbe in der Terrain-Colormap mit einem Faktor zwischen 0 und 1 erreicht.

Abbildung 5 Reifenspuren

8.7 Wasser

Das Wasser funktioniert nur auf NVIDIA-Karten. Der Wassereffekt wurde nahezu 1-1 von der angebenden Quelle kopiert. Lediglich kleine Optimierungen wurden vorgenommen (Transparenz; kein Depth-of-Field).

Abbildung 6 Wasser, Bump mapping und Reflexionen

8.8 Billboarding + Transparenz

Beim Billboarding wird bei uns auf einen Quad eine Textur mit transparenten Bereichen projiziert. Der Quad orientiert sich immer so, dass seine Normale zur Kamera schaut. Billboarding wurde für die Particle Engine, und für die Pflanzen angewendet. Außerdem wurde Tiefensortierung benötigt.

8.9 Bump Mapping

Die Steine sind gebumpmapped.

Beim Bump Mapping wird versucht, Oberflächen von Objekten interessanter (detaillierter) zu machen, ohne dabei die Komplexität (Anzahl der Polygone) des Objekts vergrößern zu müssen.
Dazu wird – entsprechend einer „Normalmap“ – an jedem Oberflächenpunkt eines Polygons nicht der Normalvektor genommen, welchen man aus den Normalvektoren aller Vertices des Polygons errechnet, sondern ein anderer. Die Lichtberechnung wird dann per Pixel durchgeführt und so lassen sich raue bzw. Oberflächen mit Struktur darstellen.

8.10 Shadow mapping

Objekte können einen Schatten werfen. Ob ein Objekt einen Schatten wirft, wird im Scenen-XML definiert.

Zum Shadow mapping wird die Szene zuerst aus der Lichtposition in einen FBO gerendert (Tiefenwerte). Diese Tiefenwerte werden dann beim Rendern aus der Cameraposition projiziert.
Der Effekt benötigt sehr viel Performance. Daher wurden auch weitere Verbesserungen (Soft Shadows, PCF) weggelassen.

8.11 Projective Shadows

Die hier angewendeten Projective Shadows sind eigentlich ein 2D Effekt bzw. ein Schatteneffekt für eine Ebene.
Da er jedoch wesentlich schneller als Shadow mapping ist, und auch sehr gut aussieht, haben wir ihn verwendet.
Es ergeben sich jedoch einige Probleme, die aber selten auftreten:

- Der Schatten verlässt Objekte, die stark von einer Ebene abweichen (z.B. Kante an einem Hügel)
- Auf manchen Grafikkarten ist der Schatten unterschiedlich stark, d.h. mehrere Objekte addieren sich. Wir konnten hierfür keine Lösung finden.
- Manchmal scheint der Schatten durch Objekte hindurch (wegen Stencil Buffer-Wert).

8.12 Gausian Blur

Nachdem die ganze Scene gerendert wurde, besteht die Möglichkeit diese mit einem 3x3 Gaußfilter zu glätten. Dadurch werden unschöne Kanten geglättet. Der Filter verbraucht auf einer GeForce 6800 kaum Performance.

9 Physik

Die Physik der Fahrzeuge beruht auf diversen Tutorials aus dem Internet (das Haupttutorial ist leider seit ein paar Monaten offline!). Sie umfasst die Reifenbewegung und das gesamte Handling der Fahrzeuge. Auch die durch „AI“ gesteuerten gegnerischen Fahrzeuge besitzen Physik.
Jedes Objekt besitzt eine orientierte BoundingBox. Kollisionsberechnungen mit dieser werden programmintern berechnet. Eine genauere Berechnung von Kollisionen ist durch die Coldet-Library möglich (tritt eine Kollision mit der BoundingBox ein, so wird bei Bedarf eine Coldet-Kollisionsberechnung durchgeführt).
Die BoundingBoxen werden auch fürs View-Frustum-Culling verwendet.

10 Kameramodell

Für die Kameras wurde die Funktion gluLookAt() verwendet. Die Kameraklasse selber besitzt einen View-, Right-, Up- und Positionvektor.
Es gibt zwei Kameramodi: der eine folgt dem Fahrzeug und beim anderen befindet sich die Kamera im Fahrzeug.

11 Progammiertechnisches

Das Spiel ist auf Basis eines simplen Szenegraphen erstellt. Alle Objekte (Terrain, Objekte, Carcontrols, Menüscreens, …) sind Nodes.

Levels, Fahrzeuge und Fahrer können über XML verändert bzw. hinzugefügt werden. Auch die Einstellungen werden über XML gesteuert.

Hier eine kurze Auflistung der Struktur:
- data/gamedata.xml: gamespezifische Einstellungen (Tastaturbelegung, Auflösung, usw.)
- data/vehicles/*.xml: Fahrzeuge
- data/levels/*.xml: Levels
- data/animals/*.xml: Fahrer

12 Sound

Die Fahrzeuge besitzen einen Motorsound. Dieser passt sich an die Drehzahl an (über Frequenz und Amplitude). Im Hintergrund läuft eine Hintergrundmusik.
Weitere Sounds treten beim Anklicken von Steuerelementen, beim Schießen, bei Kollisionen und beim Driften des Fahrzeugs auf.
Zum Abspielen der Sounds wird FMOD verwendet.

13 Levelleditor

Zum einfacheren Assemblieren der Landschaft wurde ein simpler Levelleditor entwickelt.

14 Spielbarkeit

Nach dem Aufrufen des Spiels erscheint ein Intro-Video welche per Tastendruck übersprungen werden kann. Das Video ist ein DVD-Videodatenstrom, der auf einen Quad projeziert wird.

Danach kommt man zum Hauptmenü. Hier stehen die Punkte „Options“, „Credits“, „Car choser“, „Track choser“, „Animal choser“, „Race choser“ und „Start race“ zur Verfügung.

Nachdem Ende einer Fahrt erscheint eine Hall-Of-Fame mit den besten Fahrzeiten (realer) Player.

14.1 Options

Hier kann die Tastaturbelegung geändert werden, sowie diverse Spezialeffekte ein/ausgeschaltet werden.
„End blur“ bedeutet, dass die gesamte Szene vor der Ausgabe noch einmal Gausgefiltert wird. Dadurch werden besonderst die Kanten schöner.
„Slipping-sound“ bezeichnet das Geräusch, welches beim Driften in Kurven auftritt. Da die Fahrzeuge sehr zum driften neigen, ist es oft angenehmer, wenn man den Sound ausschaltet☺.

14.2 Credits

Beschreibung der Macher.

14.3 Car choser, Animal Choser, Track Choser

In den einzelnen Menüs können Fahrzeuge, Fahrer und die Strecke ausgewählt werden. Alle Daten werden aus XML-Files geladen. Um neue Einträge hinzuzufügen, muss man nur ein neues XML-File erstellen.

14.4 Race choser

Hier kann aus diversen Rennen gewählt werden:

- 2/6 bzw. 12-Runden Rennen. Man fährt gegen Gegner, die man zerstören kann. Der schnellste gewinnt.
- Training: hier kann man das Fahren auf Tracks und das Zerstören von Gegnern üben.
- Time Trial: Hier fährt man 10 Runden alleine gegen die Uhr.
- „Opponent Destroyer“: Ziel ist es hier, möglichst schnell alle Gegner zu eliminieren.

14.5 Start race

Startet das Rennen.

15 Performance

Die Framerate ist je nach Grafikkarte, Effekten und Spielmodus sehr unterschiedlich. Auf einer GeForce 6800 läuft das Spiel mit allen Effekten im Time-Trial Modus (also ohne Gegner) bei rund 150 Frames/s. Bei Modi mit Gegnern sinkt die Framerate drastisch ab (auf ca. 30-60 FPS).
Der Grund dafür ist nicht die Grafik, sondern die Logik. Jedes Auto führt diverse Berechnungen durch. Besonders die Kollisionsdetektion je Auto kostet viel Performance (obwohl nur Objekte in einem bestimmten Radius betrachtet werden).

16 Sonstiges

Auf der Rapunzel muss für jedes Spiel das gesamte Spiel neu-gestartet werden. Auf anderen (Windows-XP-Systemen) konnten wir diesen Fehler nicht feststellen.
Außerdem muss auf der Rapunzel beim Intro 3x F6 (VA - Immediate - VBO) gedrückt werden - dann sind die FPS bei 60. Auf anderen Systemen ist dies nicht notwendig.

17 Quellen

17.1 Physik:

- http://home01.wxs.nl/~monstrous/ (DOWN!!!)
- http://www.racer.nl/reference/carphys.htm
- http://www.experts-exchange.com/Programming/Game_Development/AI_Physics/Q_21070059.html

17.2 Particle Engine:

- http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=19
- http://www.mysticgd.com/misc/AdvancedParticleSystems.pdf
- http://www.codeproject.com/opengl/ParticleEngine.asp

17.3 Terrain

- http://www.vterrain.org/

17.4 Wasser

- http://www.bonzaisoftware.com/water_tut.html

17.5 Bump-Mapping

- http://www.paulsprojects.net/opengl/bumpmap/bumpmap.html

17.6 Projective Shadows

- http://www.devmaster.net/articles/projectiveshadows/

17.7 Shadow mapping

- http://www.paulsprojects.net/tutorials/smt/smt.html

17.8 Billboarding

- http://www.lighthouse3d.com/opengl/billboarding/

17.9 Sound

- http://www.fmod.de

17.10 Sonstiges

- GLIntercept zum Debuggen

17.11 Modelle

- http://www.turbosquid.com

17.12 Video in OpenGL

- http://www.devlib-central.org/mambo/

18 Libraries

Folgende Libraries wurden verwendet:

- GLUT
- GLEW (Management von OpenGL Extensions)
- DEVIL (Einlesen von Bildern)

- FMODex (Sound)
- ColDet (Collisiondetection): http://www.photoneffect.com/coldet/