

MicroRC

Michael May, 0126194 (066 932), mike@ull.at
Jürgen Weinberger, 0527616 (532), gamsta@gmx.net

Die Idee	0
Spiel Details	2
UseCase	2
Spielmodi	3
Ablauf	3
3D Objekte	3
Steuerung	4
Allgemein	4
Fahrzeug	4
Kamera	4
Interfaces	4
GUI	4
Ingame GUI	6
Implementierung	6
Effekte	7
Besonderheiten	7
Loader	7
Physx	7
Level und Ingame-Objekt Definitionen	7
Graphik Engine	7
Fremdarbeit	8
Libraries	8
Tools	8
Texturen und Meshes	8
Referenzen	9

Die Idee

MicroRC ist ein kleines, schnelles und kurzweiliges Rennspiel nach dem Vorbild von GeneRally (<http://generally.rscsites.org/download.shtml>). Hier können bis zu 6 Spieler, menschlich- oder computergesteuert, in unterschiedlichen Spielmodi gegeneinander antreten. Um die Welt lebendig erscheinen zu lassen kommt zusätzlich zu Keyframe- u. Texturanimationen eine Physik Engine zum Einsatz. Die Fahrphysik wird dabei allerdings nur sehr vereinfacht "simuliert", um unter anderem den Spielspaß und das "Arcade" Gefühl zu behalten. Es geht mehr um Strecken- und zerstörbare Objekte die eben die Welt beleben und Hindernisse für die Spieler darstellen. Um eine Vielzahl an unterschiedlichen Strecken und Fahrzeugen zu ermöglichen, macht ein benutzerfreundlicher Strecken- und Fahrzeugseditor es sehr leicht neuen Content hinzuzufügen (vielleicht sogar von GeneRally zu importieren).

Einführung

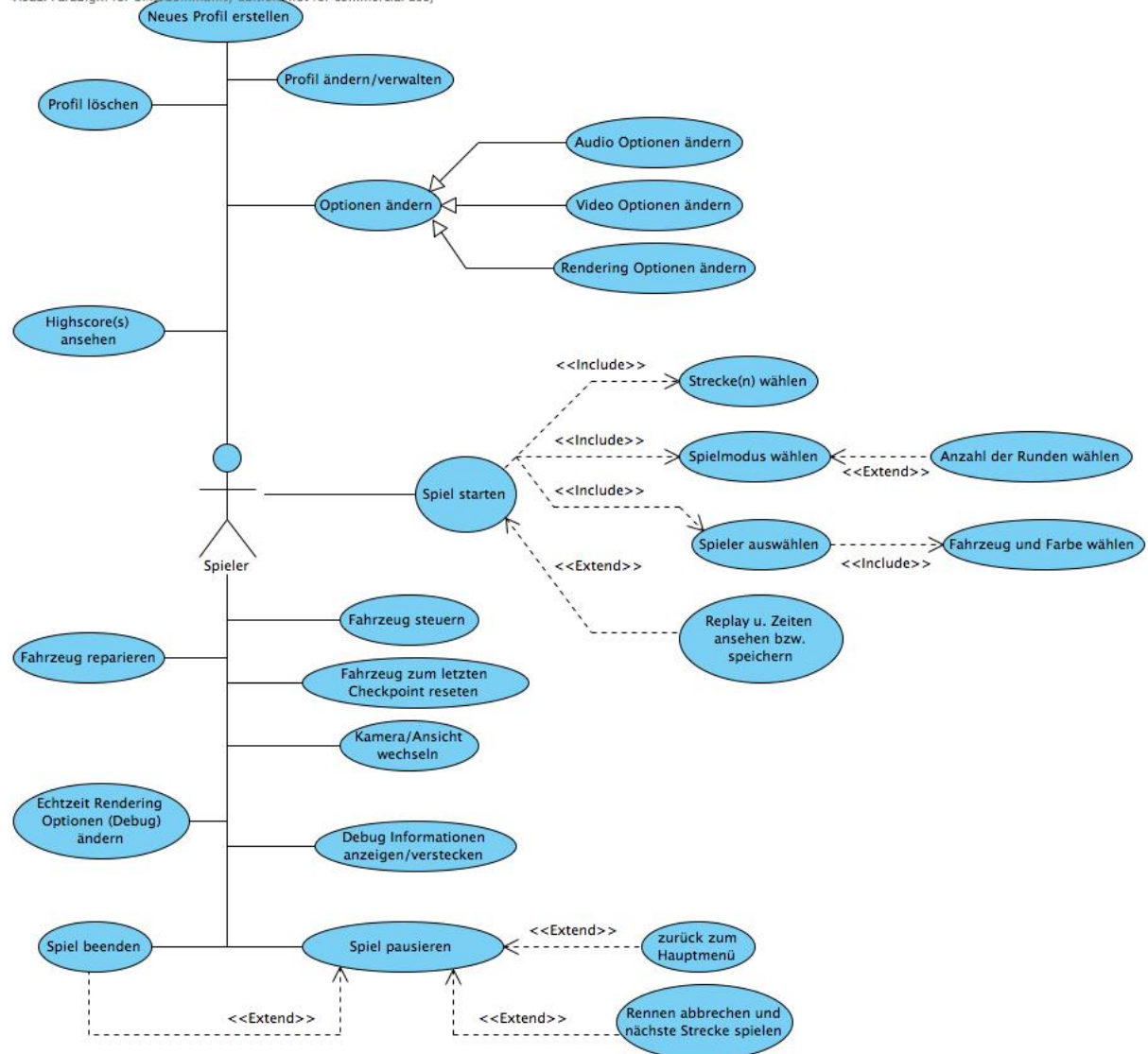
Nach Start des Spieles wird ein Menü dargestellt, in dem man derzeit Grundlegende Einstellungen vornehmen kann (Resolution, ColorDepth und Fullscreen). Zuerst muss man einen Spieler anlegen und die Steuerung festlegen. Um ein Spiel zu starten wählt man den Spieler und die Farbe für das Fahrzeug aus und fügt ihn zu der Spielersliste hinzu. Nachdem man unter "New Game" den "Race!" Button drückt wird das Spiel geladen. Die Zahlen markieren derzeit die Tore, die abgefahren werden müssen. Die Rundenzeiten sowie die Runden werden im oberen Eck angezeigt.

Anmerkung: Die Fahrphysik ist schwer unter Kontrolle zu bekommen und noch nicht zur vollständigen Zufriedenheit gelöst.

Spiel Details

UseCase

Visual Paradigm for UML Community Edition (not for commercial use)



Spielmodi

Name	Spieleranzahl	Strecken	Runden	Highscore
Training	beliebig	1	beliebig	keine Highscore

Ablauf

Nachdem das Spiel installiert und gestartet wurde öffnet sich die MFC GUI in welcher Profile erstellt und bearbeitet werden können, die Optionen verändert, die Highscore angesehen und ein neues Spiel gestartet werden kann.

Zuerst muss der Spieler ein oder mehrere Profile anlegen damit er ein Spiel starten kann. Die Profile unterscheiden da wir die AI Profile nicht implementieren konnten muss die Steuerung für jeden Spieler eingestellt werden. Um ein Spiel zu starten wählt der Spieler zuerst einen Spielmodus, eine Strecke und dann die Mitstreiter inkl. Farbe.

Nach drücken des Race-Buttons startet sich die eigentliche Renderengine und OpenGL, es wird ein Ladescreen inkl. Ladebalken sichtbar. Nachdem das Level mit sämtlichen Ressourcen geladen wurde, blendet der Ladescreen aus. Das Rennen bzw. Training startet sofort. Die Ansichten können mittels Tastendruck geändert werden. Die Spieler können nun mittels Keyboard ihr Fahrzeug beschleunigen, abbremsen, lenken und zum letzten Checkpoint zurücksetzen. Ziel ist es prinzipiell sein Fahrzeug schnellstmöglich von einem Checkpoint zum nächsten zu bewegen. Diese müssen in einer strikten Reihenfolge durchfahren werden und dürfen nicht ausgelassen werden. Kommt der Spieler vom letzten Checkpoint wieder zum Ersten (Start/Ziel) hat er eine Runde absolviert. Sollte die Streckenzeit in die Highscore fallen wird dies gespeichert und angezeigt. Absolviert ein Spieler alle Runden wird die Steuerung deaktiviert. Das Rennen beendet wenn alle Spieler die letzte Runde absolviert haben.

Der Spieler muss hauptsächlich darauf achten, dass er möglichst schnell seine Runden absolviert, dies gelingt ihm nur wenn er möglichst ideal auf der Strecke fährt und dabei mit möglichst wenig Gegenständen und auch anderen Fahrzeugen kollidiert.

Im Singleplayer kann das Fahrzeug aus der Ego Perspektive gesteuert werden. Im Multiplayer empfiehlt sich eher die Vogelperspektive.

3D Objekte

Generell gibt es 4 Arten von 3D Spielobjekten, das Terrain inkl. Wasser, die Fahrzeuge, statische und physikalische Objekte. Der Unterschied zwischen statischen Objekten und physikalischen ist nur das falls eine Kollision statt findet die Physikalischen von der Physikengine verwaltet werden und sich frei im Level bewegen können. Bei statischen Objekten kann z.B. eine Animation oder Partikelsystem ausgelöst werden und sie können sich nicht vom Platz bewegen. Es wurde zwar nicht mehr implementiert aber bei statischen Objekten können auch Transformationsanimationen definiert werden. Drehung bei Windmühlen z.B.

Name	Details
------	---------

Terrain	<ul style="list-style-type: none"> • Größe • wird dynamisch über eine Heightmap erzeugt
Checkpoints	<ul style="list-style-type: none"> • Start-/Ziel-/Strecken- Checkpoints
Fahrzeuge	
Bäume	<ul style="list-style-type: none"> • Statisch
Kisten	<ul style="list-style-type: none"> • Physikalisch
Haus	<ul style="list-style-type: none"> • Statisch

Steuerung

Allgemein

- 'ESC' Spiel Beenden
- 'Pause' Um das Spiel zu stoppen
- 'F1' Zeigt die Hilfe, FPS, sowie Spielinformationen.
- 'F2' Schaltet in den Wireframe Modus
- 'F3' Texturen Filter Bi/Trilinear Filterung
- 'F4' MipMapping ein/aus
- 'F5' Render Mode VertexArrays/VBOs
- 'F6' Frustrum culling ein/aus

Fahrzeug

Die Steuerung wird individuell per Spieler eingestellt. Prinzipiell kann das Fahrzeug gelenkt, beschleunigt, gebremst und zum letzten Checkpoint zurückgesetzt werden.

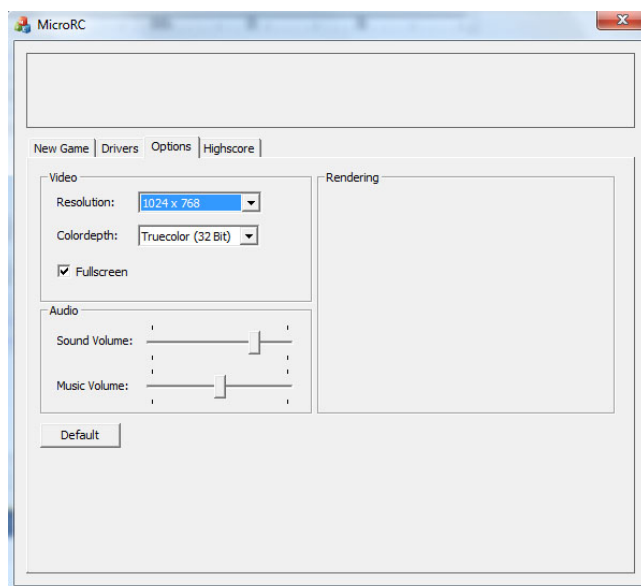
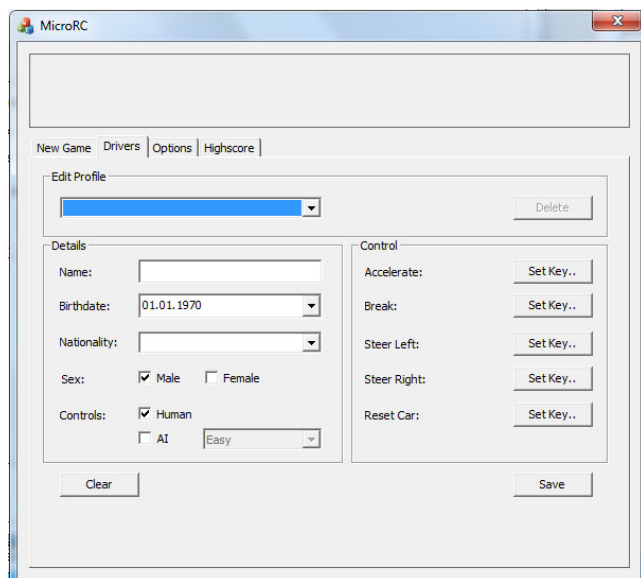
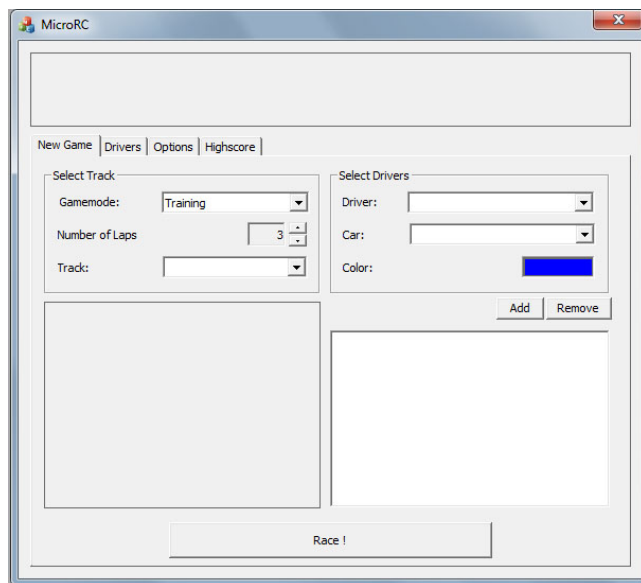
Kamera

- Pfeiltasten
- 'Pos1' & 'Ende' Können die Kameraansichten durchgeschalten werden.
- Rechte Maustaste zum rotieren der Kamera
- Pfeiltasten zum Bewegen der Kamera

Interfaces

GUI

Um möglichst wenig Aufwand in die GUI stecken zu müssen und trotzdem volle Funktionalität zu erhalten, haben wir uns für MFC entschieden und nur ganz wenige einfache Dialoge wie den Debug- und den Pausendialog dann In-Game mittels OpenGL realisiert.



Ingame GUI

Für die Ingame gui wurde ebenfalls ein kleines framework nach dem vorbild der CEGUI implementiert um die Arbeit zu erleichtern. Es existiert eine konsistente Hierarchie von Fenstern in denen andere Fenster oder Text eingebettet werden können.

Implementierung

- Spielmechanik
 - *Fahrzeug* - Das Fahrzeug kann vom Spieler gelenkt werden und wird durch PhysX beeinträchtigt.
 - *Laps* - Tore müssen nach einer vorgegebenen Reihenfolge durchfahren werden.
- Menü MFC
 - *Graphikkonfiguration*
 - *Spieler Erstellung*
 - *Highscore*
 - *Level und Fahrzeug auswahl*
- Graphik
 - *Meshes* - Jedes Mesh wird als VBO geladen und hat Materialeigenschaften zur richtigen Beleuchtung und Texturierung.
 - *Texturen* - Unterschiedlichste Texturformat können geladen werden und optional mit mipmaps an die Graphikkarte geschickt werden. Registriert werden sie per Namen, mit welchem sie dann auch im Shader zu adressieren sind.
 - *Fonts* - Es wird eine Fontmap geladen und für jeden Buchstaben ein einfaches Mesh erstellt, die nach Wunsch zusammengefügt werden. Der Font besitzt einen Alphakanal.
 - *Beleuchtung* - Alle Standardbeleuchtungen (Directional, Spot, Point) sind möglich. [4]
 - *Frustum Culling* - Jedes Objekt wird vor dem Zeichnen kontrolliert, ob es im Camera Frustum liegt. Dies kann immer in Anspruch genommen werden, wie z.B. auch bei Erzeugung der Shadowmap. Durchgeführt wird dass ganze in ClipKoordinaten. [1]
 - *Environment Mapping* - Statische Reflexionen werden mit cube maps umgesetzt. Dabei wird auf ein Image zugegriffen, dass die Einzelbilder sequentiell beinhaltet und im Speicher getrennt und ausgerichtet werden. Reflexions Berechnungen werden in Welt-Koordinaten durchgeführt. [2],[3],[4],[5]
 - *Shadow Mapping* - Werden mittels FBO erzeugt und in einer Depth Textur gespeichert. Bei der Darstellung wird ein 4-sample AntiAliasing verwendet. Dabei werden auch Alpha Werte mit voller Transparenz berücksichtigt. [4],[5]
- Sonstiges
 - *Scenegraph* - Alle Positionierungen wie Camera, Fahrzeug bis hin zur Schrift basiert auf einem Scenegraph. Jedes Objekt ist bewegbar und es lassen sich auch Positionsabhängigkeiten modellieren (zB. 3rd person camera).

Alle Spiele Objekte sowie das Level werden über Descriptor geladen. Diese sind zwar im Moment statisch im Programm gesetzt, können aber ohne Probleme von Textfiles geladen werden um so das Level zu beschreiben. Ziel war es eben das ganze Spiel leicht erweiterbar zu gestalten.

Effekte

- Shadow Maps - Ist im ganzen Level zu sehen, insbesondere bei den dynamischen Elementen, wie den Fahrzeugen. [4],[5]
- Environment Mapping - Derzeit zur Darstellung der Seen. [2],[3],[4],[5]
- Transparenz - Derzeit nur mit dem dynamischen Fontrendering verwendet.
- Simple projizierte Textur - Eine bewegte WolkenTextur wird auf die Szene projiziert. [6],[7]
- Einfaches Wasser - Eine Bump Map wird zur Verzerrung der Oberflächenspiegelung, Schatten und Lichtreflexionen verwendet. Es findet keine Berechnung im Tangent Space statt, weil sich dies Zeitlich nicht mehr ausgegangen ist. Daher funktioniert das ganze nur für orientierte Planes.

Besonderheiten

Loader

Die Meshs werden allgemein über die AssImp geladen. Dies ermöglicht uns eine weite Anzahl an unterschiedlichen Fileformaten zu laden. Wir bevorzugten Kollada. Der Loader allgemein ist nach einem Factory Pattern aufgebaut und verwaltet seine Ressourcen selbständig über Smartpointer. Die Gefahr eines Memoryleaks wird somit verringert. Außerdem verwaltet er die Ressourcen so das diese zum Beispiel nur ein mal geladen werden.

Physx

Alle Spiele Objekte werden von Nvidias PhysX gemanaged. Es war etwas mühsam sich einzulernen aber die Hardware Unterstützung und die Handhabung haben uns dann doch überzeugt.

Level und Ingame-Objekt Definitionen

Das Level und dessen Objekte werden ebenfalls über eine Factory erzeugt und gemanaged. Diese erzeugt mit Hilfe eines Descriptors das jeweilige Objekt mit seinen Attributen. Es existiert eine konsistente Hierarchie zwischen den Objekten und auch diese können leicht durch Vererbung erweitert werden. Die Descriptoren für die Objekte übrigen ebenfalls. Die Descriptoren erfüllen den Zweck die Objekte und das Level aus unterschiedlichen Datenquellen (bevorzugt files) zu laden. Dies macht es sehr leicht möglich das Spiel zu erweitern. Diese Descriptoren sind über die boost serialization implementiert und können so in alle Formen von Files (verschlüsselt, xml, plain text, bin) usw. gespeichert und geladen werden.

Graphik Engine

Setzt sich aus vier Teilen zusammen:

- GL Resource Mgmt - Managed Meshes und Texturen im Graphikspeicher.
- Shader Mgmt - Managed das Laden, Verwalten und Updaten von Shadern im Graphikspeicher.
- RenderTargets - Hiermit lässt sich bestimmen, ob in eine Textur oder auf den Schirm gezeichnet wird.

- RenderKernel - Hier laufen alle Fäden zusammen. Ein Singleton das die Verwaltung von Graphikaufgaben verwaltet. Beinhaltet und Arbeitet mit den Instanzen der obig vorgestellten Teile. zB. das Laden von Meshes in den Graphikspeicher während des zeichnens oder auch der Zusammenhang von Texturnamen in Shadern und der OpenGL-Nummerierung.

Fremdarbeit

Libraries

boost	für SmartPointer und vieles andere	www.boost.org
glfw	OpenGL Context	www.glfw.org
glew	OpenGL Extensions	glew.sourceforge.net
AssImp	Model Loader	assimp.sourceforge.net
FreeImagePlus	Image Loader	freeimage.sourceforge.net
PhysX	Physik Library	www.nvidia.com/object/physx_new.html
glm	Math Library	glm.g-truc.net

Game.dll ist unsere Spiellogik. Die Idee war, das Spiel möglichst getrennt von MFC zu entwickeln. Nachdem es aber Probleme gab MFC mit der zuerst statischen Game.lib zu verlinken, sind wir auf eine dynamische umgestiegen.

Tools

Blender	3d modeller	www.blender.org
XSI	3d modeller	www.softimage.com
Photoshop	Image Bearbeitung	www.adobe.com/products/photoshop/photoshop/
Bitmap Font Builder	erzeugen der Font Maps	www.lmnopc.com/bitmapfontbuilder/

Texturen und Meshes

Cubical Environment Maps	www.codemonsters.de/home/content.php?show=cubemaps

Referenzen

- [1] View Frustum Culling Tutorial <http://www.lighthouse3d.com/opengl/viewfrustum/>
- [2] OpenGL Cube Map Texturing http://developer.nvidia.com/object/cube_map_ogl_tutorial.html
- [3] 4.3 Cube Env. Mapping [ttp://www.ozone3d.net/tutorials/glsl_texturing_p04.php#part_43](http://www.ozone3d.net/tutorials/glsl_texturing_p04.php#part_43)
- [4] OGLS 2nd edition 'orange book'
- [5] Real-time rendering 2nd edition
- [6] Creating Clouds in Photoshop <http://www.n-sane.net/effects/fluffy-realistic-clouds/index.php>
- [7] Crating Seamless Textures <http://www.photoshoptextures.com/texture-tutorials/seamless-textures.htm>
- [8] Lake water <http://habib.wikidot.com/lake-water-with-bigger-waves>