Seifenkistenrennen

Andreas **Zweng** e0525927

Roman **Gurbat** e0525731

Gameplay

Das Prinzip des Spiels ist es die Straße entlang bis zum Ziel zu fahren. Gefahren wird mit „W-A-S-D“, gebremst mit der Leertaste. Die Fahrphysik ist mit der AGEIA PhysX Library realisiert und die Aktoren sind zum einen die Straße mit den Leitplanken und die Seifenkiste. Die Berge usw. wurden aus Performancegründen nicht als PhysX Objekte gespeichert.

Nicht triviale Objekte

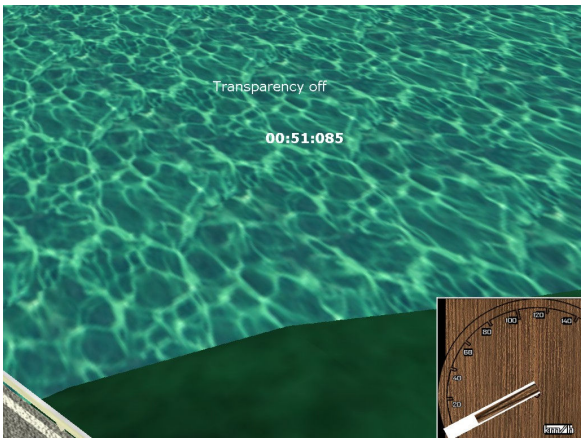
Die Objekte sind fast alle (das Wasser nicht) mit Maya modelliert und werden mit FBX eingelesen. Anschließend werden die Vertices, die Normalen und die UV-Koordinaten neu indiziert um mit einem Indexarray auszukommen. Beim ersten mal laden aus dem FBX-File werden die Daten in unser eigenes Dateiformat geschrieben, wobei wir beim einlesen die neu-Indizierung sparen und bei großen Meshes ca. 20 mal so schnell laden. Das speichern und lesen in unser eigenes Modellformat befindet sich in der Klasse „CG23FBXLoader“ in den Methoden „loadWorld“ und „saveWorld“.

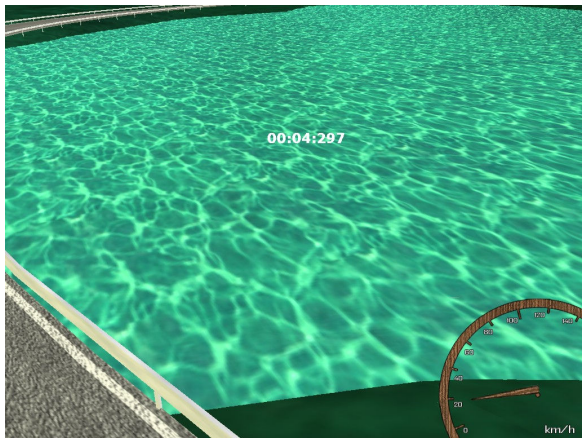
Beschleunigung der Sichtbarkeitsberechnung

Unser Terrain haben wir extra in 8x8 Teile gespalten um ein gutes Frustum Culling zu ermöglichen. Für jedes Mesh wird dann eine Boundingbox und eine Boundingsphere beim laden berechnet. Die Methode „BuildBoundingBoxes“ in der Klasse „CG23Game“ ruft die Funktionen „calcBoundingBox“ und „calcSphere“ aus der Klasse „CG23AABox“ auf um aus den gegebenen Vertices Boundingboxes und Boundingspheres zu berechnen. Beim rendern wird für jedes Objekt abgefragt, ob die Boundingsphere des aktuellen Objekts im Frustum ist, oder nicht und dementsprechend gerendert. Ein Beispiel ist in der Methode „RenderImmediate“ in der Klasse „CG23Game“ zu finden.

Transparenz-Effekte

Transparenz wurde beim Wasser, beim Tacho und beim Lensflare-Effekt verwendet.





Bei beiden Transparenzen wurde ein Alpha-Test verwendet.

Beim Wasser: `glBlendFunc(GL_ONE, GL_SRC_ALPHA);`

Beim Tacho: `glBlendFunc(GL_SRC_ALPHA , GL_ONE_MINUS_SRC_ALPHA);`

Experimentieren mit OpenGL

Bei jeder Modus-Änderung (F1, F2, F3, etc.) wird ein Text für einige Sekunden dargestellt, um die aktuelle Änderung der Einstellungen anzuzeigen.

Mit „F1“ kommt man zur Spielhilfe, wo nochmals die Tastenbelegung erklärt wird.

Mit „F2“ wird die Framerate angezeigt, bzw. weggeschaltet.

Mit „F3“ wird der Wireframe-Modus aktiviert, bzw. deaktiviert.

Mit „F4“ und „F5“ wird die Textur und die Mipmapping Qualität verändert. Dazu haben wir einen Texturmanager implementiert, der alle ID’s hällt und jedes Objekt merkt sich die ID aus dem Texturmanager, von der Textur, welche das Objekt trägt. Beim umschalten wird in der Methode „setTextureQuality“ in der Klasse „CG23Texture“ die aktuelle Qualität für alle Texturen der Map eingestellt. Die aktuellen Einstellungen für Textuqualität und Mipmappingqualität werden in Laufvariablen in der Klasse „CG23Game“ gespeichert.

Mit „F6“ wird zwischen den Modi „Immediate Mode“, „Vertex Arrays“ und „Vertex Buffer Objects“ geschaltet. Zusätzlich wird mit „F7“ entschieden, ob zusätzlich Display-Lists verwendet werden, oder nicht. Display-Lists werden am Anfang des Programmes in der Methode „BuildLists“ in der Klasse CG23Game“ für den Immediate Mode und für Vertex Arrays erstellt. Für Vertex Buffer Objects wurden keine DL’s erstellt, da diese sowieso die Daten im Grafikkartenspeicher halten.

Mit „F8“ wird das Frustum Culling ein und ausgeschalten, wobei dieses schon weiter oben erklärt wurde.

Mit „F9“ werden Transparenzen ein bzw. ausgeschaltet. Sichtbar beim Tacho, beim Wasser und beim Lensflare Effekt.

Effekte

Lens Flare (1 Punkt):

Der Lensflare Effekt wurde mit der Hilfe eines Tutorials von NeHe verwendet (http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=44). Der Effekt wurde gleich in die Klasse „CG23Frustum“ eingebunden, da man die Sichtbarkeitsberechnung gleich an dieser Stelle durchführen kann. Anfangs haben wir den Effekt so verändert, daß ein bestimmter Radius des Lensflares verdeckt werden muß, bis der ganze Effekt verdeckt wird, aber da dies nicht richtig ist, haben wir das wieder rausgenommen. Die Lensflare Texturen werden kleiner, je größer die Distanz ist. Dahingehend haben wir das ganze geändert, sodaß die Texturen immer gleich groß bleiben, indem man die größe der Textur mit der Distanz multipliziert. Die Distanz wird nur einmal für den Lensflare-Mittelpunkt berechnet und auf alle Texturen des Effektes angewandt. Der Lensflare verschwindet, wenn im Zentrum des Lensflares (Ein Pixel) der Wert des Z-Buffers an der gleichen Stelle kleiner ist als die Position des Effektes, denn in diesem Fall befindet sich ein Objekt vor dem Lensflare. Die Texturen werden dann nacheinander entlang eines Vektors mit verschiedenen Farben gezeichnet.

Motion Blur (2 Punkte):

Der Effekt wurde nach dem Tutorial von http://www.codeproject.com/opengl/MotionBlur.asp implementiert. Der Effekt wird nicht wie in den meisten Motion Blur Implementationen mit Shadern gerendert, sondern mit einer Textur, welche über das aktuelle Frame geblendet wird. Der Effekt funktioniert so, daß das aktuelle Frame gezeichnet wird und darüber das vergangene Frame geblendet wird und somit ein verschwommener Effekt entsteht. Der Vorteil daran liegt an der Ausführungsgeschwindigkeit, da die Szene nur ein mal gerendert werden muß. In unserem Spiel wird der Faktor mit dem das vergangene Frame (als Textur) über das aktuelle Frame geblendet wird mit der Geschwindigkeit erhöht, damit der Effekt bei höherer Geschwindigkeit verstärkt wird. Dabei wird der Effekt aber auf einen gewissen Prozentsatz limitiert, da sonst durch die Akkumulation der Textur bei einem Blendingfaktor von 1.0 keine Bewegung mehr erkannt wird, sondern nur noch die Textur gezeichnet wird.

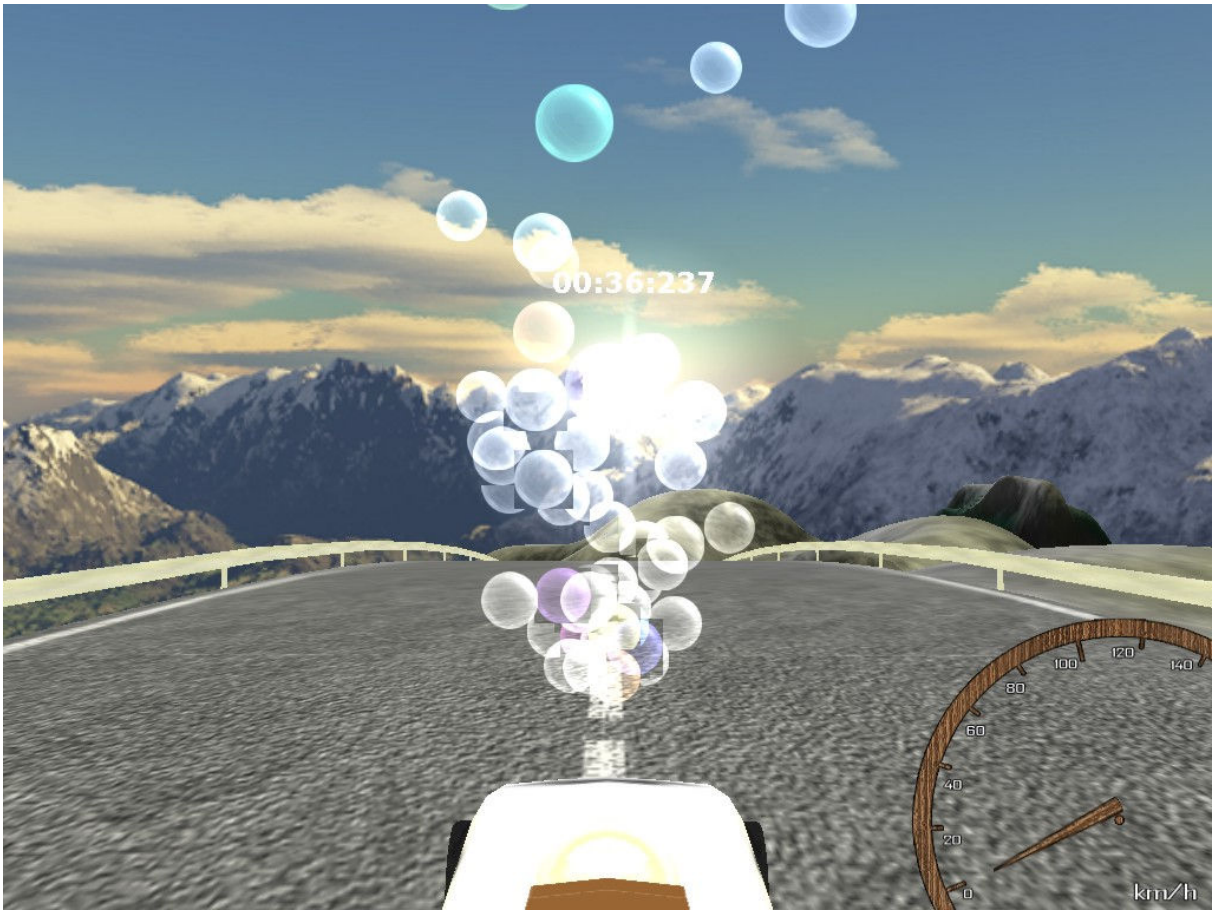
Wasser (1 Punkt):

Das Wasser, welches oben schon bei den Transparenz-Effekten gezeigt wurde, wurde aus dem NeHe Tutorial von einem Flag Effekt implementiert (http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=11). Das Wasser ist im Prinzip eine Plane, welche Punkte beinhaltet, die sich dynamisch ändern. Diese Punkte bilden ein Mesh und dadurch, daß sich die Punkte unterschiedlich, aber kontinuierlich und wiederholend ändern (sinus), entsteht ein Flag-Effekt. Der Effekt wurde dahingehend verändert, daß das Wasser rotationsinvariant ist und eine Textur auf das Mesh gemappt wird, welche mit einer Transparenz versehen wird.

Partikel (1 Punkt):

Die Partikel im Seifenkistenrennen sollten aus dem Auto kommen und Seifenblasen darstellen. Der Effekt wurde ebenfalls nach einem Tutorial von NeHe implementiert (http://nehe.gamedev.net/data/lessons/lesson.asp?lesson=19). Die Partikel haben alle eine Lebensdauer, eine Farbe und werden mit einem Alphawert gezeichnet, welcher ihre verbleibende Lebensdauer signalisiert. In unserer Implementation wird jedes Partikel mit einem Alphawert von 1.0 gezeichnet um ein Platzen der

Seifenblasen bei Ablauf der Lebenszeit zu simulieren. Wenn die Lebensdauer eines Partikels abgelaufen ist, wird es wiederbelebt mit voller Lebensdauer.

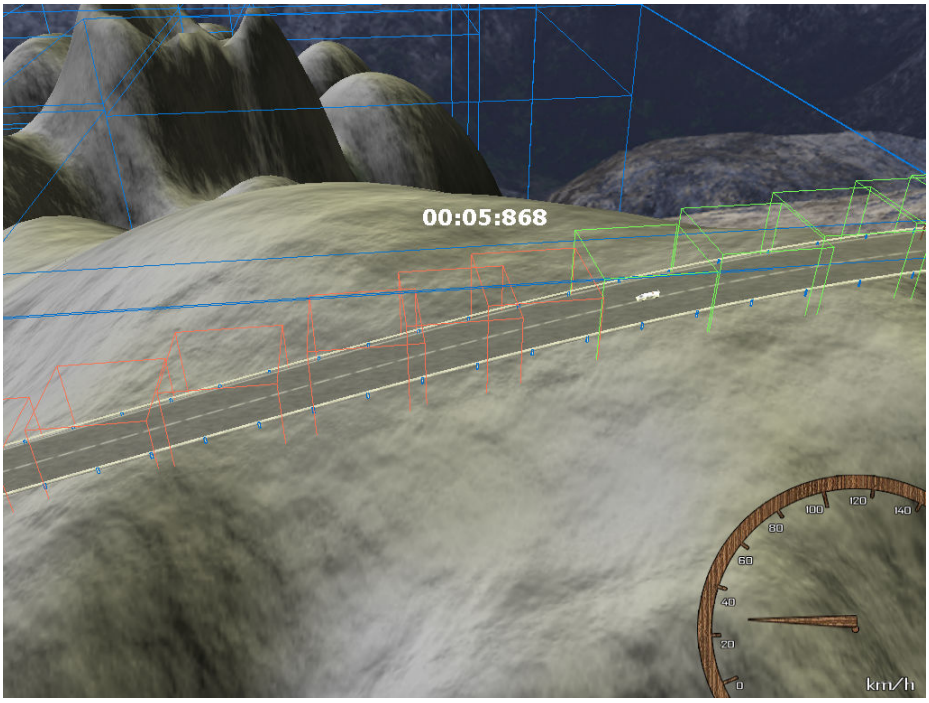


Besonderheiten des Spiels:

Wir haben ein eigenes Modell-format implementiert, in dem nur die wichtigsten Daten gespeichert werden und dadurch das Laden der Objekte um ein vielfaches schneller geht. Zusätzlich erspart man sich dadurch auch das neu Indizieren der Geometrie-Arrays. Die Geometrie wird beim ersten mal laden des Spiels vom FBX-File geladen, falls aber schon das File mit dem eigenen Modellformat vorhanden ist, wird von diesem geladen. Die Lade- und Speichermethoden befinden sich in der Klasse „CG23FBXLoader“: „loadWorld“ und „saveWorld“.

Das Spiel hat eine eigene Nebelklasse „CG23Fog“, welche bewerkstelligt, daß sich der Nebel Zufällig verstärkt oder verschwächt. Dabei finden diese Verstärkungen bzw. Verschwächungen in Perioden statt, sodaß es Auflockerungsphasen bzw. Nebelphasen gibt. Der Nebel kann sich stärker oder schwächer verändern und wird mit der Taste „N“ aktiviert bzw. deaktiviert.

Zum Rücksetzen des Autos wurde eine Checkpoint-Klasse implementiert „CG23Checkpoints“, wobei jeder Checkpoint eine Boundingbox, die Orientierung des Auto’s und die Position des Autos gespeichert hat um das Auto korrekt zurückzusetzen. Die Boundingboxen sind aus der Klasse „CG23AABox“ und werden aktiviert, sobald das Auto durch diese durchfährt. Die Abfrage beim durchfahren ist deswegen performant, da nur der vorderste nicht gecheckte Checkpoint abgefragt wird aufs aktivieren. Dadurch wird auch verhindert, daß man Abkürzungen fahren kann. Die blauen Boxen im Bild sind nur die Boundingboxen der Weltobjekte. Alle Boundingboxes können mit der Taste „Q“ sichtbar gemacht werden.



Tastaturbelegung:

Funktionstasten wie oben beschrieben.

„Q“: Alle Boundingboxen anzeigen/ausblenden
„W“: Gas
„M“: Musik an/aus
„+“: Musik lauter
„-„: Musik leiser
„B“: Sound, Pause
„N“: Dynamischen Nebel an/aus
„C“: Freie Kamera / Kamera an Auto gebunden
„X“: Wechsel zwischen EGO-Perspektive und Verfolgungskamera beim Auto
„Y“: Partikel an/aus
„S“: Rückwärts
„SPACE“: Bremsen
„A“: Links lenken
„D“: Rechts lenken
„R“: Auto bei Unfall zurücksetzen
„UP“: Kamera bei freier Kameramodus aufwärts
„DOWN“: Kamera bei freier Kameramodus abwärts

Zusatztools

Physik: AGEIA PhysX (http://www.ageia.com)

Musik: FMod (http://www.fmod.de)

Models: FBX (http://usa.autodesk.com/adsk/servlet/index?id=6837478&siteID=123112)

Texturen: DevIL (http://openil.sourceforge.net/)

Modelling: Autodesk Maya (http://www.autodesk.de/adsk/servlet/index?siteID=403786&id=8800342)