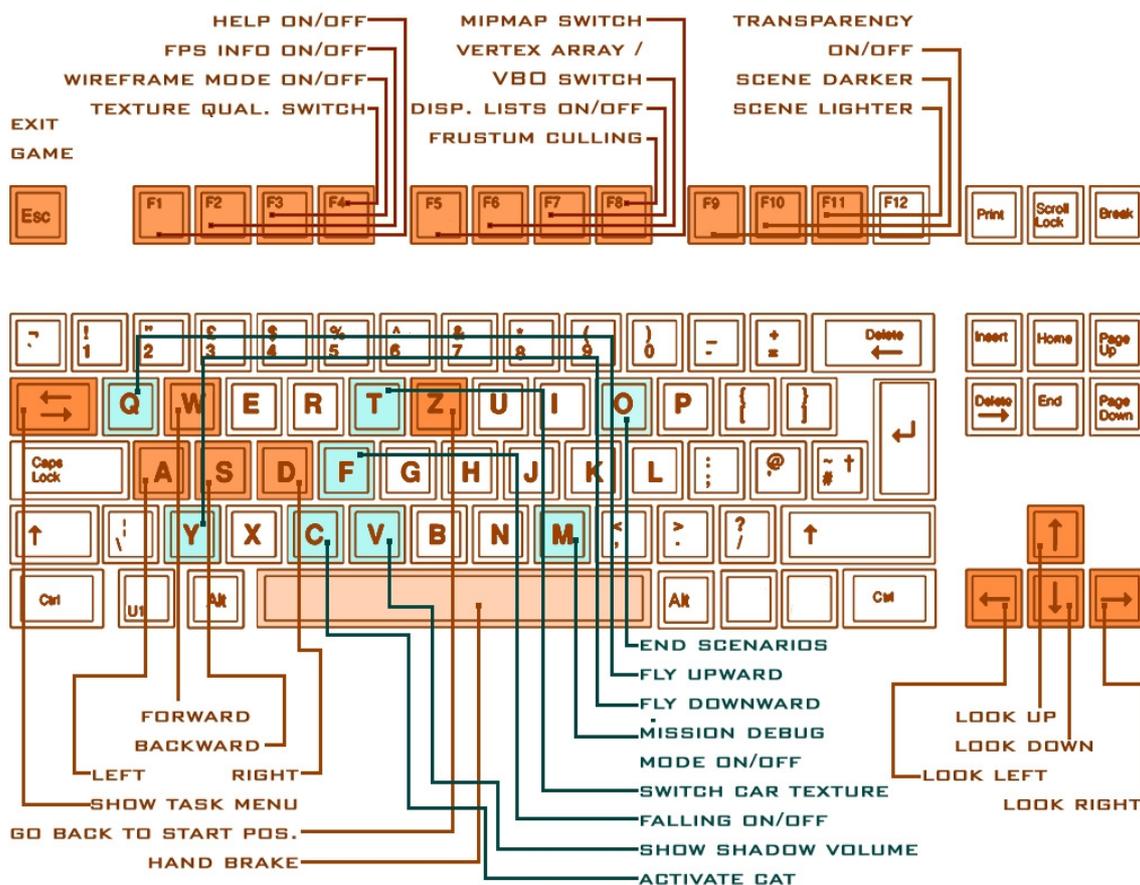


Anmerkung zur Abgabe:

Es wurden 4 ausführbare Dateien abgegeben. Fullscreen und Window Mode für XP und Vista. In Vista war es uns leider nicht möglich VBO's genug zu testen (sie sollten in Vista nicht funktionieren), deshalb sind in der Vista Version nicht aktiviert. Wenn es möglich ist würde wir jedoch doch lieber auf Vista präsentieren (falls die XP Präsentationen wieder später sind), da wir beide im Anschluss SEPM Prüfung haben.

A Workaholic's Nightmare

Steuerung:



Steuerung: orange Standardbelegung, blau extra Funktionalität für Debugging

Gameplay:

Es gibt 9 Missionen und 4 Bonusaufgaben. Das Ziel ist es, alle Missionen abzuschließen, bevor die Zeit ausläuft.

Das Menü (siehe Abbildung unten) wird mittels der TAB-Taste erreicht. Ein Mausklick auf der „Show“- Fläche blendet animierte Pfeile ein, die einem die Position der für die jeweilige Mission relevanten Objekte zeigt. Die Pfeile sind nur wenige Sekunden sichtbar. Jeder solcher Mausklick wird mit Punkteabzug bestraft.

Die Punkteanzeige befindet sich im linken unteren Eck des Bildschirm.



Missionsübersicht

- Einschalten der Lampe: neben dem Startpunkt links ist eine Stehlampe; durch navigieren auf den Einschaltknopf (Position zwischen den Vorderrädern ist auslösend) kann man sie einschalten.
- Finde die zerrissene Telefonnummer und mache einen Anruf: die Telefonnummer ist in drei Stücke zerrissen. Die Teile müssen gefunden und zum Telefon gebracht werden. Die Reihenfolge ist hier irrelevant. Wenn alle Teile gesammelt wurden, wird ein Anruf getätigt, der einem ein bisschen Aufschluss geben kann, warum man sich in der derzeitigen Situation befindet. (*Vorsicht Spoiler: die Positionen der 3 Teile sind: auf dem Stuhlsitz, im dunklen Eck auf dem Tisch hinter dem Fernseher, im Regal oberhalb vom Telefon – um dorthin zu gelangen, muss man über die Kabel fahren!*)
- Finde die richtige Radiostation: Mann muss zum Radio oberhalb vom Telefon fahren. Wenn man die Position erreicht hat, wird man in den „Missionsmodus“ versetzt, wo die Belegung der WASD Tasten geändert wird, damit man die Aufgabe leichter erledigen kann. Die Änderung wird angezeigt. Sobald der richtige Sender gefunden wurde, kann man ein Bericht hören, der einem auch ein Teil der Geschichte erzählt.
- Klappe das Buch um: Der Weg um Plattenspieler auf dem kleinen ovalen Tisch ist durch ein halboffenes Buch gesperrt. Man muss nur dagegen fahren.

- Einsammeln der Tasten und zum Laptop bringen: im Level sind 4 verschiedene Tasten versteckt. Diese müssen der richtigen Reihenfolge nach zum Laptop gebracht werden. (*Vorsicht Spoiler: H: links neben dem Laptop, E: auf den Radio links neben dem Laptop, L: auf dem Ecktisch links neben dem Laptop, P: oberhalb des Dreieckigen Lineals rechts vom Laptop*)
- Stoße das Glas vom Tisch: dies ist eine der leichtesten und lustigsten Aufgaben im Spiel. Ein Glas befindet sich auf dem ersten Tisch kurz nach der Startposition. Man muss mehrfach dagegen fahren, bis es über die Tischkante kippt. (*Vorsicht Spoiler: ein zweites Glas ist unterhalb vom Ersten zu finden, auf dem großen Anatomieatlas, ein drittes Glas ist auf der linken Armlehne des Schaukelstuhls positioniert, wo sie hinter der Lehne abbiegt. Das sind **Bonusaufgaben**, die zusätzliche Punkte bringen.*)
- Schalte den Fernseher ein: Der Schalter befindet sich oben auf dem Fernseher. Es gibt zwei Zufahrtsmöglichkeiten. Die erste ist über die Lineale, wo gute Steuerungsfähigkeiten gefragt sind. Die zweite ist vom Radio ausgehen über die Kabel zum Ventilator zu fahren und von dort aus hinunter zum Fernseher. Dies ist die leichtere und spektakulärere Route. Sobald der Fernseher eingeschaltet wurde, wird die vierte Bonusaufgabe aktiviert. (*Vorsicht Spoiler: **Die vierte Bonusaufgabe** besteht darin, zum Schaukelstuhl zu fahren, über die mit Lichtpunkten gekennzeichnete „Sprungschanze“ hochzufahren und in den Fernseher zu springen. Man wird dann in eine andere Welt versetzt, wo man Zeit zurückgewinnen kann, indem man ein Herz aufhebt. Damit ist ein echtes Herz gemeint. Sobald man das Herz hat, wird man ins Zimmer zurückversetzt.*)
- Tasse zerbrechen: die Tasse ist vor dem Notebook zu finden. Man muss nur dagegen fahren. Wenn man die richtige Richtung erwischt, gewinnt man mehr Punkte.
- Spiele die Schallplatte ab: um die Mission zu starten, muss man gegen die Nadel des Plattenspielers stoßen. Dann wird man in den „Missionsmodus“ versetzt. Die WASD-Tasten werden wieder kurz umbelegt. Die Schallplatte muss ein paar Sekunden lang in der richtigen Geschwindigkeit abgespielt werden. Damit ist die Mission erledigt. *Hinweis: Die Anfahrtsrichtung ist wichtig um die Mission auszulösen. Sie sollte in etwa der Richtung des Pfeils entsprechen, den man sich mittels show anzeigen lassen kann.*

Das Spiel ist zu Ende, wenn alle Missionen geschafft wurden, oder wenn die Zeit ausläuft. Je nach Aussehen der EKG-Kurve gibt es vier unterschiedliche Endszenarios. (*Vorsicht Spoiler: Über die O – Taste können sie momentan noch während des Spiels angezeigt werden.*)

Des weiteren wird man von der Hauskatze verfolgt. Reagiert man nicht auf ihren Angriff, wird man an die Startposition zurückgeworfen und die Zeit, die noch übrig bleibt, wird knapper. Ausweichen kann man, indem man kräftig zurück lenkt. (*Vorsicht Spoiler: Ein Angriff kann jederzeit über die C- Taste ausgelöst werden.*)

Achten Sie auf die EKG-Kurve im rechten unteren Eck des Bildschirms. Sie zeigt an, wie weit Sie von Ende entfernt sind.

Technisches:

Datenstrukturen für das Laden von Objekten

- Modelle: werden als *.obj eingelesen. Das Auslesen erfolgt in der Model.cpp Klasse.
- Level: befindet sich in der Level.txt Datei. Wird in der Loader Klasse ausgelesen.
- Texturen: sind als *.tga gespeichert und werden auch in der Loader Klasse gelesen.
- Ausnahmen: Kollisionsobjekte, Auto, Katze und Schattenobjekte werden nicht in der Loader Klasse geladen.
- Opake, Transparente und glühende Objekte werden in dieser Reihenfolge geladen und in der renderScene() – Methode unterschiedlich behandelt.

Modelle werden mithilfe von Vertex Arrays oder als Vertex Buffer Objects ausgegeben und besitzen Texturen. Die Windows XP Version verwendet per default VBO's, die Vista Version Vertex Arrays. Der Wechsel zu VBO's in Vista ist derzeit leider nicht möglich.

Bewegte/animierte Objekte

Bewegung ist von der Framerate unabhängig. Dazu wird am Anfang des Spieles und beim Wechseln eines Anzeigemodus eine durchschnittliche (Median) FPS-Rate berechnet. Sollten die FPS zu schnell werden, wird die Schleife (Ende der Game Klasse) verlangsamt. Die Bewegungen werden mittels der Variable *accelMod* an die Framerate angepasst.

Die Katze ist hierarchisch animiert. Ihre Bewegung ist an der Autobewegung gekoppelt. (siehe Methode drawCat() in Car Klasse). Motion Blur wird hier angewandt.

Weitere animierte Objekte sind die Schallplatte und die Pfeile, die die Positionen der einzelnen Missionen anzeigen. Diese wurden als normales Objekt umgesetzt. Bewegung wird dadurch erzielt ihre Positions- oder Rotationswerte zu modifizieren. Beim Zeichnen aller Objekte werden diese Berücksichtigt.

Andere Bewegungen werden durch den Spieler ausgelöst – die Drehung der Radiotaste, das Umkippen der Gläser und des Buchs. Bewegung erfolgt ebenfalls als normales Objekt wie oben beschrieben.

Bewegungen die durch den Spieler ausgelöst werden und weiter laufen benutzen unterstützend die Klasse DynamicObjects. Die komplexere Bewegungen mit einer Referenz zum jeweiligen Objekt durchführt.

Beschleunigung der Sichtbarkeitsberechnung/Frustum Culling

Frustum Culling wurde zusätzlich implementiert, ist jedoch per default deaktiviert. Es sind leichte Geschwindigkeitsgewinne zu erkennen. Jedoch konnten wir es nicht lang genug testen um standardmäßig aktiviert zu lassen.

Vorgehensweise ist folgende: Beim Laden werden bounding spheres berechnet. In einer getrennten Klasse (*FrustumCulling.cpp*) wird geprüft ob sich die bounding spheres innerhalb des View-Frustums befinden. Dazu werden die aktuellen Projection- und Modelview-Matrizen ausgelesen.

Kameramodell

Kamera folgt dem Auto, das einen Spieler repräsentiert. Man kann jederzeit stehen bleiben und sich umsehen. Dies geschieht durch Drehung um die X- und Y- Achsen , wobei die Position des Autos als Zentrum der Drehung dient. Die Drehung um die X- Achse ist beschränkt, damit die Kamera nicht unter dem Boden verschwinden, oder sich ganz um das Auto drehen kann. Bewegung der Kamera wird auch in der Car Klasse behandelt. Da die Kamera zurückgesetzt werden muss und sich auch je nach Autobewegung auch bewegt ist keine konkrete Methode dafür zu nennen.

Texture Mapping

Texturen werden auf allen Objekten in der Szene angewandt. Sie werden vorher einmal geladen und können auch während des Spiels wechseln (zb. Monitor, Lichtkegel der Lampe, Hud-Ekgkurve).

Materialien, Transparenz, Beleuchtung

Es wird nur eine Lichtquelle verwendet. Dazu wird allen Objekten ein einfaches Material zugewiesen, das keinen spekularen Anteil aufweist. Das Auto hat ein anderes Material – mit roten diffusen und ambienten und weißen spekularen Anteilen.

Transparente Objekte werden mittels Alphawerten zwischen 0 und 1 und unterschiedlichen Blendfunktionen implementiert. Die Lampen haben zusätzliche eine weiße Emission – Komponente in ihrem Material. Die Materialzuweisung geschieht in der Methode `applyMaterial()` in der Klasse `ShadowVolume`.

Die Lichtquelle muss am Anfang des Spiels als allererste Aufgabe eingeschaltet werden. Dies geschieht durch Erweiterung des Lichtkegelwinkels und Umschaltung der Texturen für die betroffenen Objekten, d.h. es findet keine Zustandsänderung statt.

Die Normalvektoren aller Objekte werden, wie oben beschrieben, gleich mit den Vertex- und Textur- Koordinaten geladen und normalisiert. Es kommen sowohl flache als auch gekrümmte Flächen vor.

Experimentieren mit OpenGL

F1: Hilfe Bildschirm

F2: FPS Anzeige: dazu wird am Anfang und am Ende der Schleife die Zeit gemessen und die zugehörigen FPS im linken oberen Bildschirm-Eck dargestellt. Messung erfolgt nicht jedes Frame.

F3: Wireframe: einfaches wechseln von `glPolygonMode`

F4: Texturqualität: zum Wechseln mussten die Texturen neu geladen werden, durch die Verwendung von Lightmaps verwenden wir jedoch relativ viele Texturen, was das Wechseln sehr verlangsamt. Leichte Qualitätsunterschiede sind merkbar.

F5: MipMapping-Qualität: siehe Texturqualität.

F6: Vertex Arrays/Vertex Buffer Objects: Unser erster Ansatz war die Verwendung von Vertex Arrays, jedoch haben wir im Zuge des Experimentierens auch VBO's umgesetzt. Der

Wechsel war jedoch nicht allzu einfach zu bewerkstelligen, nachdem schon ziemlich viel Code auf Vertex Arrays basiert war. Nach beheben der durch VBO's entstandenen Fehler und Treiberupdates konnten wir eine Verbesserung der Framerate erzielen. Momentan scheint diese aber nicht sehr viel schneller als Vertex Arrays. Man muss anmerken, dass nur ein Großteil der Szene (alle gameObjects) als VBO's gezeichnet wird. Auto, Katze und Schatten nicht.

Wir haben uns gegen eine zusätzliche Implementierung von Immediate-Mode entschieden (da einerseits der Code dadurch noch unübersichtlicher geworden wäre und andererseits kein Performance Gewinn zu erwarten war), in der Schatten Klasse, bei der Debug-Ausgabe von Missionsbereichen und im Hud werden jedoch Immediate Mode Befehle verwendet.

F7: Display Lists: bringen einen Merkbaren Performance Gewinn, beeinflussen aber die Beweglichkeit der Objekte. Testweise implementiert, jedoch müsste zur Verwendung zwischen völlig statischen Objekten und beweglichen Objekten unterschieden werden.

F8: Viewfrustumculling: bringt ebenfalls Performance Gewinn, jedoch war nicht genügend Zeit vorhanden es genügend zu testen.

F9: Transparenz ein/aus: transparente Objekte werden am Ende der Render Methode gezeichnet. Davor wird Transparenz aktiviert, danach wieder deaktiviert.

Generell muss man anmerken, dass sicher noch Performance Gewinn zu erlangen wäre, wenn man versucht state-changes soweit als möglich zu vermeiden. Das Zeichnen wäre sicher an vielen Stellen noch zu vereinfachen, oder einheitlicher zu gestalten.

Vor allem durch die Umsetzung von Missionen war viel laufende Änderung notwendig und es war nicht von Anfang an klar, welche Objekte bewegt werden müssen und welche nur statisch sind, oder was Objekte sonst noch alles können müssen. Es war zwar von Anfang an geplant zwischen unterschiedlichen Objekten schon im Level File unterscheiden zu können, jedoch ging sich eine möglicherweise sinnvolle Vererbungsstruktur verschiedener Objekte nicht mehr aus.

Effekte

Volume Shadow:

Das Auto wirft Schatten auf alle Objekte der Szene und auf sich selbst. Dazu wird ein vereinfachtes Schattenobjekt geladen und seine Polygone immer in Richtung des Lichtstrahls extrudiert. Dazu werden die das Auto und die Kamera betreffende Transformationen anhand eigen implementierter Matrizen- Klasse der Schatten-berechnenden Methode durchgereicht.

Der so entstandene Schattenvolumen wird im Stencil- buffer gerendert.

Das Rendern der Szene (in Game.cpp) geschieht in 3 schritten: Initialisierung des Depth-buffers, Rendern des verschatteten Teils, Rendern des belichteten Teils.

Light Maps:

Die Modelle wurden mit 3dsmax v7.0 erstellt. Die Light Maps wurden im Modell berechnet und in TGA- Format dem Objekt- Loader weitergegeben.

Motion Blur:

Wird nur bei der Katze angewandt. Es wird die Überlagerung von 4 Images mit steigenden Transparenzwerten, die durch einen jitter – Value gegeneinander versetzt wurden, im Accumulation Buffer gebildet.

Collision detection:

Zur Kollisionsberechnung wurde keine vorgefertigte Library verwendet. Daraus resultierender Vorteil wäre, dass so keinerlei Lizenzgebühren entstehen würde.

Prinzipiell werden Kollisionen auf low-poly Kollisionsobjekten berechnet. Die Kollisionen werden getrennt für alle 4 Räder entlang der z-Achse berechnet, um die Höhe zu bestimmen. Polygone werden aus der Draufsicht mit bounding rectangles approximiert. Danach wird genau überprüft ob der Punkt innerhalb des Polygons liegt und die Höhe auf der Ebene berechnet.

Entlang der z-Achse wird geprüft welche Polygone am nächsten oberhalb und unterhalb des momentanen Rad-Punktes liegen. Liegt der Punkt weiter unterhalb wird der Radpunkt nach unten verschoben. Andernfalls folgt eine Ablenkung nach oben. Es kann jedoch nur eine bestimmte Höhe nach oben hin befahren werden.

Bei jeder Ablenkung nach unten die eine gewisse Schwelle überschreitet wird geprüft ob mit einer Wand kollidiert wurde. Dazu wird geprüft in welche Richtung der Normalvektor (beim initialisieren berechnet) des darüber liegenden Faces zeigt. Zeigt der Normalvektor nach oben wurde eine Kollision mit einer Wand festgestellt. (Anm.: Diese Vorgehensweise ist nur möglich, da eine Wand und ein Abgrund in jedem Fall ein Loch in einer Kollisionsebene bedeuten, wenn man nur ein Kollisionsobjekt benutzt dass keine überschneidenden Faces besitzt und dessen Normalvektoren alle nach außen zeigen.)

Zusätzlich wurden 2 weitere Punkte eingefügt die zwischen den Rädern Kollisionen abfragen und das Auto bei bedarf nach links oder rechts verschieben. Somit wurde behoben, dass man wenn man eine Kurve gegen ein Eck fährt leicht stecken bleiben konnte. (Kann aber ein leichtes verschieben des Autos an wenigen Stellen verursachen die eigentlich unerwünscht wären.)

Drei bekannte Probleme die vielleicht noch teilweise bis zur Präsentation gerichtet werden können sind: in wenigen Situationen kann es vorkommen, dass die Vorder- oder Hinterräder durch den Boden gefallen scheinen; die Höhe des Autos wird nicht berücksichtigt; die Neigung nach links/rechts des Autos wird nicht berücksichtigt.

Zusätzliches

Sound

Es wurde OpenAL verwendet. Die Sounds wurden Großteils von Youtube geladen und bearbeitet. Die Auto Geräusche benötigten etwas mehr Bearbeitung unter der Verwendung von Cubase. Leider haben wir keine Möglichkeit gefunden störende Knackgeräusche beim Abschneiden der Sounds durch OpenAL zu beheben. Die Sounds sind im 3d Raum platziert und OpenAL kümmert sich um das korrekte Abspielen.

Missionen

Wurden in der Klasse Missions umgesetzt und benötigen alle möglichen Referenzen zu Objekten, Sounds, Hud, ect. Dort wurde die Logik der Missionen umgesetzt. Jeweils als eigene Methode für jede Mission.

Video (noch nicht vorhanden)

Für das bei dieser Abgabe noch nicht vorhandene Video wurde Direct Show verwendet. Wir hoffen das Video noch bis zur Präsentation fertig stellen zu können.

Klassenliste

Game.cpp: main-Methode, Game Loop, Initialisierungen starten

Car.cpp: behandelt das Auto und seine Bewegung sowie die Katze. Kann über Tastatureingabe über Game gelenkt werden und reagiert auf unterschiedliche Einflüsse. (z.B: „Gravitation“, Missionen, Katze, ...)

Missionen:

Missions.cpp: enthält alle Missionen und behandelt deren Ablauf.

GameTask.cpp: definiert Missions-Auslöserbereiche die einen String liefern sobald das Auto diese erreicht. Können mehrfach oder einzeln feuern.

DynamicObject.cpp: Hilfsklasse für erweiterte Bewegung von Objekten. (z.B: Glas, LP)

Zeichnen:

HudLoader.cpp: das Menü inklusive Endbildschirme

ShadowVolume.cpp: verwaltet Beleuchtung, Materialien und Schattenberechnung

FrustumCulling.cpp: Umsetzung der FrustumCullings

Laden:

Loader.cpp: liest Level File

Model.cpp: zum Speichern und einlesen der Modelle

GameObject.cpp: alle Objekte die gezeichnet werden sind GameObjects. Beinhaltete Information zu Modell, Textur, Status, Sichtbarkeit, Position, Rotation, Skalierung, ect.

FacelIndex.cpp: speichert Vertex-Information unterstützend beim Einlesen

Kollision:

Collision.cpp: übergeordnete Klasse für Kollisionen

CollisionDetection.cpp: berechnet einzelne Kollisionen

DynamicCollision.cpp: berechnet Kollisionen für bewegliche Objekte (zb Glas)

Sonstige Hilfsklassen:

Sound.cpp: kümmert sich um die Sounds

PlayVideo.cpp: zum Abspielen des Videos wenn vorhanden.

CG2Matrix4x4.cpp: enthält Hilfsfunktionen zur Vektor- und Matrizen- rechnen

Verwendete Quellen:

[1] OpenGL Programming Guide 6th Edition

[2] verschiedenes: <http://www.cg.tuwien.ac.at/courses/CG23/Tipps.html>

[3] Collision: <http://2000clicks.com/MathHelp/GeometryPointAndTriangle2.htm>

[4] Volume Shadow (Autor: Mark J Kilgard): www.scribd.com/doc/4838089/Stencil-Buffer-in-OpenGL

[5] Vertex Buffer Objects: http://www.songho.ca/opengl/gl_vbo.html

[6] Frustumculling: <http://robertmarkmorley.com/2008/11/16/frustum-culling-in-opengl/>

[7] OpenAL: <http://www.devmaster.net/articles/openal-tutorials/lesson1.php>

[8] DirectShow: [http://msdn.microsoft.com/en-us/library/dd375454\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/dd375454(VS.85).aspx)