Computergraphik 2/3

SS2006

Endabgabe

unsign

Ausgearbeitet von:

Koren Thomas	0426003	532	e0426003@student.tuwien.ac.at
Osebitz Christian	0426358	532	e0426358@student.tuwien.ac.at

1 Vorwort

Um eine möglichst umfangreiche Dokumentation zu bekommen, ohne aber die Konzentration der Tutoren unnötig zu strapazieren, wurden die Dokumentationsteile der Zwischenabgabe im Dokument gelassen. Diese wurden allerdings leicht ausgegraut. Jeglicher, tiefschwarzer Text in diesem Dokument beschreibt also die Änderungen seit der letzten Abgabe

Diese Übung war bislang mehr als eine Herausforderung für uns. Wir wussten, dass wir uns für diese Übung einen Programmiersprachenwechsel von JAVA auf C++ unterziehen müssen und mit diesem Mehraufwand hatten wir auch gerechnet. Unvorbereitet traf uns allerdings die Tatsache, dass das Wissen aus CG1 nur sehr wenig in die Programmierung von OpenGL einfließen konnte. Dies soll keineswegs eine Ausrede sein. Wir sind stolz auf das was wir bisher geleistet haben. Aber unser eigener Ehrgeiz stichelt uns immer wieder und flüstert uns zu „Ihr seid zu mehr fähig“.
Wir hoffen auch Sie verspüren dennoch ein Gefühl der Zufriedenheit, beim Kontrollieren unserer Arbeit.

Bereits beim ersten Spieleevent haben wir festgestellt, dass unser Spiel auch bis zum Ende des Bearbeitungszeitraumes nicht das effektreichste und somit schönste wird. Aus diesem Grund haben wir uns für die Endabgabe mehr auf Interaktion und Gameplay konzentriert. Die Anforderungen von Seiten der Übungsleitung wurden aber selbstverständlich eingehalten.
Am Ende dieses Dokuments befindet sich unsere verbleibende ToDo – Liste um zu zeigen, was wir uns für diese Abgabe noch alles vorgenommen hatten (und um uns selbst daran zu erinnern, sollten wir uns im Sommer an die Vervollständigung des Spiels wagen)

2 Zur Erinnerung

Eine imaginäre Kleinstadt fernab des Mainstreams.
Hungersnot, Arbeitslosigkeit, Terroranschläge… Begriffe, die den Bewohnern dieser Kleinstadt höchstens aus unrealistisch wirkenden Berichterstattungen aus unerreichbaren Ländern bekannt sind. Aber selbst in dieser Idylle sind Probleme bekannt. Die größte Nennbare wird gemeinhin als „Langeweile“ bezeichnet. Sie treibt die Bewohner dazu nackt in zugefrorenen Seen zu baden, sich unzählige Haustiere zu halten und auch dazu skurrile Gegenstände zu sammeln. Die Königsdisziplin dieser Sammler ist das traditionelle „Ortstafel abmontieren“.

Der Spieler schlüpft in die Rolle eines Szeneeinsteigers. Als solcher muss er zunächst auf eigene Faust Erfahrungen sammeln und sich in der Szene einen Namen machen. Je weiter er in diese Sucht eintaucht umso beliebter wird er in der Gemeinschaft, aber auch bei der Polizei.
Ziel ist es, sich nicht erwischen zu lassen.

3 Status

3.1 Starten

Im „`bin`“ Verzeichnis befinden sich mehrere Batchdateien, die die `exe` mit den nötigen Kommandozeilenparametern ausführt. Derzeit wird nur der erste Kommandozeilenparameter direkt als `glutGameModeString` übergeben. Fehlt dieser, startet das Programm im Fenstermodus.
Zum Starten des Programms, werden (neben Standarddateien wie der `opengl32.dll` oder einem installierten .NET v1.1 Framework) nur die Dateien aus dem `bin` Verzeichnis verwendet.

3.2 Wo bin ich?

3.2.1 Demoszene für die Zwischenabgabe

Ein lauer Sommerabend mitten in der Wüste. Ein karges Trainingscamp offenbart sich dem Auge. Das Chalet des Trainingscampleiters ist neben dem entspannenden Sonnenuntergang der einzige Blickfang. Hier wird der Wahlkärntner auf sein Hobby in der Heimat vorbereitet.
Das Training verläuft folgendermaßen. Auf dem Trainingsareal befinden sich drei Ortstafeln, die „geerntet“ werden wollen. Also was liegt näher als hinzufahren und sie abzuschrauben? Beim Verlassen des Startareals (WireSphere) startet die Uhr. Der Aspirant muss zu den drei Tafeln hinfahren, sie abschrauben und danach wieder zum Domizil des Campbesitzers zurückkehren. Dort stoppt die Uhr.
Ein geborener Kärntner bleibt dabei unter einer Minute ;-)

3.2.2 „Finale“ Spielszene

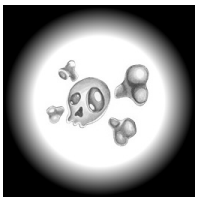
Raus aus dem kargen Trainingscamp und rein in eine sanft geschwungene Hügellandschaft. Der Spieler startet im Zentrum dieser Idylle an seinem Wochenendhäuschen. Der Abend dämmert. Die perfekte Zeit um seinem Hobby nachzugehen. Im Westen (in Richtung der untergehenden Sonne) und auch in den anderen drei Haupthimmelsrichtungen befinden sich die Ortsgrenzen der anliegenden Dörfer und die dazugehörigen Ortstafeln (markiert durch blaue Aura). Ziel ist es nun sein Können aus dem Trainingscamp unter Beweis zu stellen. Also hin zu den Tafeln und sie unbemerkt entwenden.
Aber Vorsicht! Das ist die Realität. Gesetzeshüter (rote Aura) wurden durch die Geschehnisse der vergangenen Nächte aufgescheucht und suchen nach Tafeldemonteuren wie dem Spieler. Als nützlich erweisen sich bei dem Vorhaben diverse Alltagsgegenstände, die in der Botanik verstreut sind:

„fuchzena Schließl“
DAS Werkzeug für den Tafelsammler.
Kenner schätzen ihn, da er die Demontagezeit glatt halbiert.
Wieder ein Beweis, dass man mit dem richtigen Werkzeug schneller an sein Ziel kommt.

„Grüße aus Reifnitz“
Die Tunigszene rund um den Wörthersee beschert auch dem Heimatpatrioten irrsinnige Beschleunigungsvorteile.
Mit dem richtigen Tuning leistet der Wagen kurzzeitig 50% mehr.

„A Kanister Marülln“
In Bauernkreisen wird Marillenschnaps grundsätzlich nur noch kanisterweise eingekauft bzw. konsumiert. Dass sich solche Unmengen an hochprozentigem Alkohol negativ auf die Fahrtauglichkeit auswirken verwundert also nicht weiter.

„Zecknimpfung“
Unausstehlich wie sie mit ihren Waffen auf der Lauer liegen um ihren ahnungslosen Opfern das Blut aus den Adern zu saugen. Polizisten mit Radarpistolen und Zecken haben erschreckende Gemeinsamkeiten.
Glücklicherweise kann man sich gegen beide Plagen kurzzeitig immunisieren.



„Was is grün und innen hohl? Schnittlauch! Schnittlauch!“
Selbst der toleranteste Polizist wird bei solchen versteckten Hassgesängen zum Verteidiger seines Berufsstandes. Seine Freunde eilen ihm natürlich prompt zur Hilfe. Wenn man sich also diese Dreistigkeit erlaubt, hat man sofort alle verfügbaren Polizisten am Hals.

Sollte es ihm gelingen alle Tafeln abzuernten, muss er in die Sicherheit seines Heimes zurückkehren (gelbe Aura) und den Tag so glücklich und zufrieden ausklingen zu lassen.

3.3 Steuerung

Folgende Tasten sind belegt:

W, S	Vor- bzw Rückwärtsbewegung des Wagens
A, D	Lenkung
Linke Maustaste + Bewegung	Freie Kameradrehung den Wagen
Rechte Maustaste	Rücksetzen der Kamera
F1	Übersicht der Funktionstasten anzeigen
F2	FPS Anzeige togglen (bremst das Programm wegen Aufruf von glFinish()
F3	Wireframe Modus ein/aus
F4	Texturfiltermodi wechseln
F5	Mipmapping ein/aus
F6	Zeichenmodi wechseln
F7	Displaylists ein/aus
F8	Visibilityculling ein/aus
F9	Alphablending ein/aus
F10	Texturen ein/aus
1	Eventspheres und ihre Beschriftungen ein/ausblenden
2	Boundingboxen für Sichtbarkeitsberechnung ein/ausblenden
3	Boundingboxen für jedes einzelne Mesh anzeigen („2“ muss aktiviert und Displaylisten deaktiviert sein)
4	Physikalische Wireframewelt rendern (SEHR LANGSAM!)
5	Achsen des Spielobjektes ein/ausblenden
6	Szenegraphinformationen auf Konsole schreiben (Objekthierarchie wird nicht angezeigt)
R	Quickreload der Szene. Mit Vorsicht zu genießen. Es wurde nicht getestet ob diese Funktion in JEDER Situation fehlerfrei arbeitet.
ESC	Beendet das Programm

3.4 Spielverlauf

Mit W, A, S, D wird der Wagen zu den Ortstafeln bewegt. Dort wird dann automatisch in den „Demontagemodus“ gewechselt. Dort muss die angezeigte Tastenkombination eingegeben werden um die Tafel abzumontieren und zurück in den „Drivemodus“ zu wechseln. Sind alle Tafeln geerntet, gilt es so schnell wie möglich zur Basis zurückzukehren.

4 Technische Details

4.1 Optisches

4.1.1 Demoszene der Zwischenabgabe

Die kleine Demowelt beinhaltet zwar noch wenige Objekte, aber jedes Objekt dient dazu einen gewissen Implementierungsaspekt zu veranschaulichen.
In der Szene sind zwei Lichtquellen und ein `LightModel` aktiv. Hinter der Sonne auf dem Skydome befindet sich eine direktionale Lichtquelle, die die Szene beleuchtet. Zusätzlich strahlt ein schwaches rotes Spotlight vorne aus dem Fahrzeug.
Grundsätzlich jedes Objekt verfügt über eine Materialdefinition und Normalvektoren.
Boden, Himmel, Ortstafeln und das Chalet verfügen zusätzlich über Texturen.
Boden und Himmel haben eine leichte Eigenbeleuchtung (EmissiveColor).
Himmel, Chalet, Ortstafeln und Wagen werden über einen einfachen ASCII Model-Loader eingebunden und automatisch mit Material versehen und Texturen nachgeladen.
Über dem Startareal schwebt ein kleines animiertes Solarsystem. Es wurde zu Beginn der Übung langsam aufgebaut um einzelne Teile der Engine zu testen. Man kann unterschiedliche Materialeigenschaften erkennen (Farben und Reflexionseigenschaften). Des Weiteren zeigt sich auch die Vorteile des hierarchischen Szenegraphen. Die „Erde“ (blau) rotiert um die „Sonne“ (gelb). Es ist dabei egal wo sich die Sonne befindet. Die Erde rotiert immer im selben Abstand und Geschwindigkeit um die Sonne. Dasselbe Prinzip verfolgen die beiden Monde. Sie nutzen allerdings die Erde als ihr Referenzobjekt.
Alle Animationen sind natürlich nur von der Zeit und nicht von der Rechenleitung des PCs abhängig.
Entfernt man sich zu weit von den Szeneobjekten wird am Ende des Viewfrustums langsam in die Hintergrundfarbe überblendet (Fog).

4.1.2 „Finale“ Spielszene

Das Terrain wird bei jedem Start von einer Heightmap generiert, in 8x8 Quadranten eingeteilt und mit 64 Texturen belegt.
Newton Game Physics übernimmt Physikberechnungen.
Das Abmontieren der Tafeln erfolgt noch immer über die Eingabe von Tastenkombinationen. Die optische Anzeige dieser wurde allerdings durch 2D Overlays ersetzt. Ebenso auch die Anzeige von Fehlern/Ereignissen am Spielverlauf.
Weitere Änderungen betreffen den Bereich „Spezialeffekte“ und werden dort näher erläutert.

4.1.3 Spezialeffekte

Bewegter Himmel:
Dafür war keine Informationsquelle von Nöten. Ein Quad wird ebenso wie die Skybox über dem Spieler platziert. Die Texturkoordinaten, werden über die Zeit verändert.
Die Wolkentextur wird mittels der Informationen aus dem Alphakanal mit dem Hintergrund überblendet.
Lens Flare:
Die Basisinformationen zur Realisierung stammen aus dem Nehe Tutorial 44 (Link is grad nicht verfügbar).
Die Umsetzung der Positionierungsberechnung musste allersings an unseren Szenegraphen angepasst werden. Somit befinden sich nur noch minimale Anteile des Tutorialcodes in dem Programm.

Die Texturen wurden in höherer Auflösung neu erstellt.
Transparenz:
Zunächst nur für Himmel, Lensflare und Wagenscheiben. Es stellte sich aber heraus, dass auch Grafikoverlays sehr gut aussehen. Damit wurden dann Spielnachrichten und der Loadingscreen realisiert.
Die Objekte werden grob sortiert. D.h. nicht auch Trianglebasis, sondern nur auf Objektbasis. Dadurch kommt es bei größeren Objekten zu Fehlern in der Zeichenreihenfolge. Besonders der Übergang von Grafikoverlay zu Himmel ist störend, da sich der Himmel nur ein Quad ist und dieser sich aber über einen großen Bereich erstreckt.
Basiscode stammt aus dem Red Book.
Billboarding:
Für die Auras um die wichtigsten Spielobjekte.
Es wurde zunächst echtes Billboarding implementiert. Jedoch stellte sich heraus, dass diese Variante zu „unruhig“ war. Die Texturen drehten sich natürlich mit jeder Kamerabewegung mit und lenkten das Auge so vom eigentlichen Spielgeschehen etwas ab. Aus diesem Grund werden die Billboards nun immer normal auf die Hauptblickrichtung der Kamera gezeichnet. Dies führt zwar auch zu sichtbaren Bewegungen der Textur. Allerdings beschränkt sich diese Bewegung nun auf die äußeren Ränder des Zeichenbereiches, der meist weitestgehend unbeachtet bleibt.

4.2 Spielerisches

4.2.1 Demoszene der Zwischenabgabe

Um für die Zwischenabgabe eine vereinfachte Spielvariante einzubauen, mussten kleine Interaktionen mit den Objekten implementiert werden.
Beim Verlassen des Startareals (provisorische Collision Detection), wird die Stoppuhr gestartet. An den Ortstafeln (wieder CD) wechselt das Spiel in den Demontagemodus. Dabei werden die Steuertasten des Wagens neu belegt. Sie dienen nun dazu die Sequenz zum Abschrauben der Tafel einzugeben. „Verdrückt“ man sich dabei, muss die Sequenz von neuem eingegeben werden. Ist die eingegebene Sequenz korrekt, werden die Tasten wieder zum steuern des Wagens verwendet und die Tafel als „demontiert“ markiert. Dies zeigt sich, wenn man versucht eine Tafel zweimal zu bearbeiten. Versucht man ohne alle drei Tafeln in das Zielareal zu fahren, läuft die Uhr weiter und ein Hinweis wird ausgegeben. Erst mit allen Tafeln wird die Uhr im Zielbereich gestoppt und die Mission als „erledigt“ markiert.

4.2.2 „Finale“ Spielszene

Das GameEvent Framework das für die Zwischenabgabe als „Hack“ eingeführt wurde,wurde weiter ausgebaut, da es sich als äußerst einfach und schnell herausstellte. Über die Taste „1“ kann währende des Spiels jede Eventsphere angezeigt werden. Wenn für zwei Objekte ein Event in der Pipeline vorhanden ist, wird dieses beim Betreten/Verlassen sowie beim Aufhalten innerhalb oder Außerhalb der Kugel ausgelöst.
Besonders erwähnenswert sind noch die Auswirkungen der Upgrades. Durch unser Eventframework, kann so ziemlich jede Veränderung des Spielverhaltens in wenigen Minuten implementiert werden (wir hatten ursprünglich Ideen für 15 upgrade items).
Es befindet sich eine einfache „Fang mich wenn du kannst“ AI in jedem Polizeiwagen. Dieser Jagdmodus wird durch Events wie beispielsweise „Spieler in Sichtweite“ ausgelöst.

4.3 Code

4.3.1 Hierarchischer Szenegraph

Zu Beginn gleich ein Teil, das uns sehr viele Nerven gekostet hat, aber auch gleichzeitig unser ganzer Stolz.
Der Szenegraph wurde als Implementierung des Compund-Patterns ausgeführt. In der Basisklasse der Hierarchie (`ScenegraphNode`) werden die Basisfunktionen aller Nodes des Baumes definiert. Dies beinhaltet die Ausgabe von Debuginformationen im Teilbaum oder aller Objekte, das Zeichnen aller Objekte im Baum (oder nur eines Teilbaumes), das Zeichnen von lokalen Koordinatenachsen sowie die Erstellung und Verwaltung von Displaylists. Jedes Element im Baum kann Operationen in eine Displaylist auslagern. Die Operationen dafür werden von jeder Tochterklasse in `executeDisplayList` definiert.

Durch diesen Aufbau kann jede der Tochterklassen als Wurzel eines Szenegraphen dienen. Um Einige Rendereinstellungen besser gekapselt definieren zu können wurde die `RootNode` Klasse eingeführt. In ihr kann eine Trennung zwischen Kameraobjekten (Objekte, die sich mit der Kamera bewegen) und Weltobjekten vorgenommen werden. Alle Objekte die im Baum unter `RootNode` gehängt werden, zählen zu Weltobjekten. Da `CameraNode` wiederum Wurzeleigenschaften hat, dient dieses Objekt als Wurzel für alle Kameraobjekte. Zudem kann hier auch noch das Beleuchtungsmodel für alle Objekte im aktuellen Baum definiert werden.

`CameraNode` kapselt in Kombination mit `RootNode` alle Einstellungen, die die Kamera betreffen

`TransformationNode` kapselt eine Transformationsmatrix in einem Baumknoten. Mit dieser ist es möglich hierarchische Animationen zu implementieren, wenn die Matrix durch eine Instanz von Animation verändert wird (siehe Planetenanimation in Demoszene).

`GeometryNode` dient als Basisklasse für alle zeichenbaren Objekte. Sie stellt die Zeichenroutinen für alle Tochterklassen zur Verfügung. Zudem auch noch Funktionen um die Modelview Matrix vom Stack zu speichern, die für jede Node gerade aktiv ist (interessant zur Bestimmung von Kamerakoordinaten von Objekten).

`BoxNode` und `SphereNode` sind von `GeometryNode` angeleitet und zeichnen nur einfache Körper. `RawGeometryNode` ist da schon eleganter. Mit ihr ist es möglich direkt eine Displaylist an jeder stelle im Code „aufzunehmen“ und diese dann auch im Szenegraphenbaum zu hängen. Als alternative kann diese Klasse auch „Polygonsuppen“ abspeichern und kapseln.
`MilkshapeNode` kapselt den Modelloader für Milkshape Dateien und erledigt auch alle damit verbundenen Aufgaben (Meshes/Materialien/Texturen des Objekts verwalten).

Und zu guter letzt noch `LightNode`, die es erlaubt Lichtquellen an beliebiger stelle in der Welt zu platzieren und ihre Beeinflussung durch Kameraveränderung zu bestimmen.

4.3.2 Zeitbasis/Animationen

Alle Bewegungen sind unabhängig von der Rechenleistung. Dafür sorgen die Klassen `TimeBase` und `Animation` (und ihre Ableitungen `TimedRotation` und `CarMovement`).
`Animator` übernimmt die Verwaltung aller Animationen. Sie implementiert das Observer Pattern und macht Hinzufügen, Entfernen und Ausführen von Animationen sehr komfortabel.

4.3.3 Materialien/Texturen

Jedes `GeometryNode` Objekt verfügt über ein `Material` Objekt. Vor dem Zeichnen wird dieses Material geladen.
Texturen können optional angefügt werden. Das übernehmen `Texture` bzw. `Texture2D`. Diese Klassen laden zunächst Bilder von der Platte in den Hauptspeicher und leiten diese dann weiter in ein TextureObject auf der Grafikkarte. Danach wird das Bild defaultmäßig aus dem Hauptspeicher entfernt. Von da an dient die Klasse nur noch zur Verwaltung des TextureObjects. Bei Bedarf, kann das Bild aber erneut von der Platte nachgeladen werden.

4.3.4 Extensions

Bislang wurde musste nur auf die Verfügbarkeit von `glWindowPos2i()` geachtet werden. Aber die Klasse `ExtensionManager` überprüft schon jetzt eventuell nicht vorhandene Funktionen und ersetzt sie gegebenenfalls durch Workarounds.
Um die Sichtbarkeitsberechnung auch mit der Hardwarevisibility Tests zu vergleichen wurde mit `GL_ARB_occlusion_query` experimentiert. Da es sich als im Großteil der Fälle schneller als View Fustrum Culling erwies, wurde es als Standardsichtbarkeitsberechnung eingeführt.
`GL_ARB_vertex_buffer_object`. Erklärt sich wohl von selbst.

4.3.5 Text Overlay

`FPSCounter` und `StopWatch` arbeiten für die Anzeige von Text nach demselben Prinzip. Die beinhalten eine eigene `TimeBase` und bieten das Ergebnis als Memberfield an. Um dieses auch auf den Bildschirm zu zeichnen, dient `TextPainter`. Da die Textverwaltung im 3D bzw. 2D Raum unnötig umständlich sind, werden wir demnächst auf die „glf“ library zurückgreifen.

4.3.6 Grafikoverlays

Spielstatusänderungen und Hinweise werden über Texturen mit Alphakanal angezeigt. Die meisten Textausgaben der Vorgängerversion wurden dadurch ersetzt

4.3.7 Physik/Collision Detection/Gamelogic

Die Collision detection (Spieler bei Ortstafel?) für die vereinfachte Spielvariante, wollten wir als Übergangslösung mit der „`coldet`“ library umsetzen. Dann hatten wir bemerkt, dass eine einfache euklidische Distanzberechnung zwischen boundingspheres hier gute Dienste leistet. So sind zwar noch Codefragmente, die die library verwenden vohanden (`Collider::colides()` case 1 bis 3), aber die werden nicht mehr verwendet. Nur noch „case 4“ ist aktiv.
Spätestens beim Umstieg auf die AGEIA Bibliothek, wird dieser Teil gelöscht.
Dann verliert auch die Gamelogic ihr derzeit katastrophales Layout.
Als Physikengine wird jetzt Newton verwendet. Einfach, schnell, zuverlässig und vor allem muss kein eigener Treiber installiert werden um das Spiel auf einem fremden Rechner zu starten. Alles in allem sehr zu empfehlen.
Spiellogik ist nun sehr modular und erweiterbar.
Als Event-Kollisionserkennung wird nun (wie schon zuvor erwähnt) ein eigenes framework verwendet (Im Ordner „Events“ des Sourcetrees).

4.3.8 Rendering

`Renderer` dient derzeit als kleine Hilfsklasse zur Verwaltung von Fenstern, dem Viewfustrum und Renderparametern. In der finalen Version wird hier aber jegliche Interaktion zwischen Spieler und Grafik stattfinden. D.h. die unschönen prozeduralen Funktionen in `main.cpp` werden größtenteils in `Renderer` wandern.

Schon jetzt kümmert sich die Klasse beispielsweise um die Fogüberblendungen am Ende des Viewfustrums.

4.3.9 Und da waren noch…

Auch die kleinen Hilfsklassen `Matrix`, `Vector` und `Utilities` haben zumindest eine Erwähnung verdient. Wobei letztere hauptsächlich zur komfortablen Erstellung von Arrays dient.

4.3.10 Unschönes

Nobody is perfect. Deswegen hier eine kleine Stellungnahme zu Codesünden.
Besonders die `main.cpp` ähnelt mehr einem Schlachtfeld als schöner Programmierung. In ihr wurden immer wieder „schnelle Testfunktionen“ implementiert um zu „schaun obs geht“. Die Flut an statischen Variablen und prozeduralen Funktionen, wird demnächst einer dringend benötigten Schönheitskur unterzogen. Das nur als Anmerkung, dass wir solche „Vergewaltigungen von OOP Paradigmen“ keinesfalls gutheißen.

4.4 Libraries

4.4.1 Boost 1.33.1

(http://www.boost.org/)
Eine irrsinnig große Library mit allen möglichen Schnickschnack um den Code und/oder den Hauptspeicher sauber zu halten.
Sie macht entpackt den größten Teil unserer Abgabe aus. Umso schlimmer, dass wir nur die beiden smart pointer Implementierungen `boost::shared_ptr<>` und `boost::shared_array<>` davon verwenden.

4.4.2 GLUT 3.7.6

Die modifizierte Version von Alexander Wilkie.
Ich glaube ein Link dazu ist wohl überflüssig :)

4.4.3 GLEW 1.3.4

(http://glew.sourceforge.net/)
Wird verwendet um die Verfügbarkeit von fragwürdigen Funktionen zu überprüfen. Neben den gezwungenen Einbindungen von `glew.h`, werden nur in `ExtensionManager` Funktionen dieser Library aufgerufen.

4.4.4 devIL 1.3.7

(http://openil.sourceforge.net/)
Zum Laden von Grafikdateien. `Texture` und `Texture2D` beinhalten die Funktionsaufrufe dieser library.

4.4.5 Newton Game Dynamics 1.52

http://www.newtondynamics.com/
Für Kollisionserkennung der „physikalischen“ Objekte und ihrer Interaktion miteinander.

4.4.6 FMOD 4.03.06

http://www.fmod.org/
Sounduntermalung

4.4.7 (Milkman3D)

(http://sourceforge.net/projects/milkman3d/)
Hier scheiden sich die Geister. Ursprünglich sollte diese Library Milkshape ASCII files für uns laden. Nachdem wir allerdings bemerkt hatten, was für Vereinfachungen und Denkfehler im Loader vorhanden sind, haben wir den Importcode kopiert und die entsprechenden Stellen korrigiert. Stellenweise den Loader auch für unsere Zwecke erweitert.
Und obwohl keine einzige Funktion aus diesem Package mehr verwendet wird, weigert sich der Linker das Projekt ohne diese Library zu Vervollständigen.
(Für Lösungsvorschläge sind wir offen)

4.4.8 (Coldet 1.1)

(http://photoneffect.com/coldet/)
Ursprünglich als Übergangslösung geplant, bis die AGEIA Library vollständig eingebunden ist und um bei der Zwischenabgabe eine leichte Kollisionserkennung zu haben.
Wurden aber durch unsere eigene Minikollisionserkennung ersetzt. Codefragmente befinden sich noch in `Collider`, werden aber nicht mehr verwendet und fliegen demnächst raus.