h_da - Computer Graphics

AM3D
Ein interaktives Spiel für Laserpointer

Paper

Marco Münch, Alexander Bartels

WS 08/09

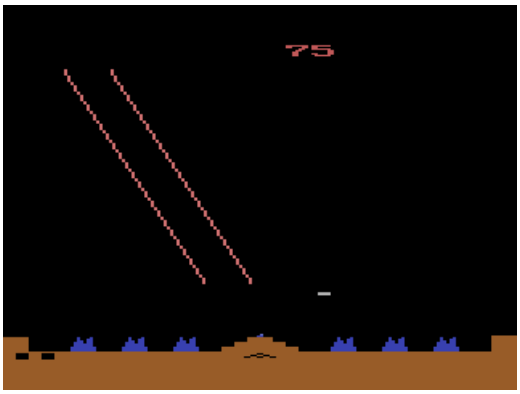

Abbildung 1: Missile Command ([Geo02])

1 Abstract

Das Projekt **AM3D** wurde im Rahmen der Veranstaltung Computer Graphics entwickelt. Es verbindet die Interaktion des Projektes Laser Interaction System 3D (**LIS3D**), welches im Sommersemester 2007 und Wintersemester 2007/2008 an der Hochschule Darmstadt entwickelt wurde, mit dem zu Beginn der 80er Jahre erschienenen Computerspiel **Missile Command**.

Bei LIS3D wird mit Hilfe von fünf symmetrisch angeordneten Laserpointern eine dreidimensionale Interaktion an einer Rückprojektionswand ermöglicht. Aus diesem Projekt wurden schließlich die Erkenntnisse der Laserpunkterkennung und des Laserpunkt-Tracking entnommen und in AM3D integriert. Das Spiel selbst wurde in der freien 3D-Grafikengine Ogre programmiert und schließlich für die Leinwand als Stereo-Sicht-Applikation umgesetzt.

Dieses Paper betrachtet somit hauptsächlich die Grundlagen des Erkennens und Trackens von Laserpointern auf einer Rückprojektionswand, des Weiteren die Funktionsweise einer LIS3D Anwendung aus Sicht des Anwenders und schließlich im speziellen den Entwicklungsprozess von AM3D.

Schlüsselwörter: AM3D, LIS3D, Laser-Tracking, Laser-Interaktion, Ogre, Stereoprojektion

2 Einleitung

Im Wintersemester 2008/2009 wurde für das Fach Computer Graphics eine interaktive Anwendung entwickelt. Diese Anwendung, im weiteren Verlauf als *AM3D* bezeichnet, verwendet dabei das frühere Projekt LIS3D [BA08], indem es das Tracken des Laserpointers auf einer Rückprojektionsleinwand zur Spielsteuerung übernimmt. Die Grundlage für das Programm stellt das Spiel Missile Command [Kun08] dar, welches in Abbildung 1 mit einem Screenshot dargestellt wird.
Ziel des Spieles ist es mit Hilfe von Raketen auf die Erde stürzende Meteoriten zu zerstören, bevor sie dort aufschlagen. Die Raketen werden dabei vom Benutzer selbst gestartet und fliegen anschließend zu ihrem Zielpunkt, explodieren und zerstören alle Meteoriten, die sich in einem bestimmten Radius um den Explosionsursprung befinden.

Normalerweise werden als Zielkoordinaten für die Raketen die Positionsdaten der Maus verwendet. In AM3D sollte dies nun stattdessen unter Verwendung des auf der Projektionswand auftreffenden Laserpunktes umgesetzt werden. Zusätzlich soll die Anwendung statt in 2D im dreidimensionalen Raum spielen und mit der freien 3D-Grafikengine Ogre[1] erstellt werden. Eine weitere Anforderung war die Verwendung von Stereoprojektion, um mit Hilfe der an der Hochschule Darmstadt vorhandenen Materialien einen spürbaren dreidimensionalen Effekt zu erzeugen.

3 Grundlagen

In diesem Kapitel sollen nun die benötigten Kenntnisse über die verwendete Stereoskopie und das Projekt LIS3D vermittelt werden.

3.1 Stereoskopie

Für die LIS3D Anwendung wird eine polarisationserhaltende, dunkle Projektionsleinwand benötigt. An der Hochschule Darmstadt wird das Bild - wie abgebildet in der Abbildung 2 dargestellt - durch einen Beamer von hinten auf die Rückseite der Leinwand projiziert, daher der Name ***Rück***projektionsleinwand.
Wenn auf dieser Leinwand (Powerwall) nun eine stereoskopische Anwendung gezeigt werden soll, so befinden sich hinter ihr nicht ein, sondern zwei Beamer. Damit zudem ein dreidimensionaler Eindruck entstehen kann, benötigt man außer einer Szene, die aus zwei verschiedenen Blickrichtungen aufgenommen wurde (siehe Abbildung 3), auch noch die Möglichkeit diese beiden Bilder für jedes Auge getrennt aufzuzeigen.

An der Hochschule Darmstadt wird die Stereoskopie mit einer Lösung der Firma Infitec realisiert. Diese besteht aus zwei Teilen. Zum einen einer Brille, welche vom Benutzer getragen werden muss und den Beamern. Dabei wird durch die Infitec-Brille für jedes Auge eine bestimmte Wel-

[1] http://www.ogre3d.org

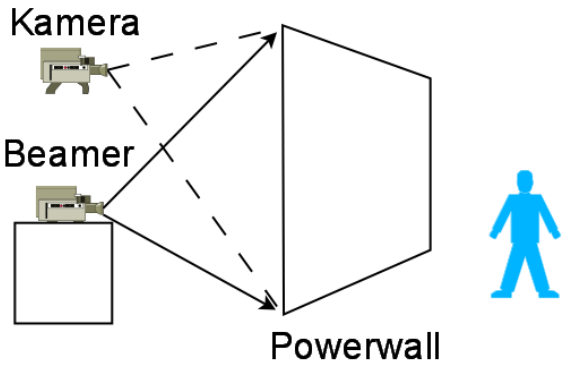

Abbildung 2: Rückprojektionswand

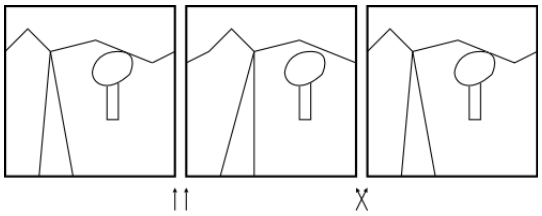

Abbildung 3: Stereobild [Haw08]

lenlänge Licht durch einen Interferenzfilter herausgefiltert. Das Spektrum dabei liegt in etwa bei den folgenden Werten:

- Linkes Auge: Rot 629 nm, Grün 532 nm, Blau 446 nm
- Rechtes Auge: Rot 615 nm, Grün 518 nm, Blau 432 nm

Damit für jedes Auge auch nur ein entsprechendes Bild mit einem Beamer projiziert werden kann, müssen diese beiden Beamer auch nur die ensprechenden Farben darstellen. Dafür wurden von der Firma Infitec spezielle Filter entwickelt, die über ein bestimmtes Farbspektrum verfügen, welches dem der beiden Augen entspricht. Dies wird in Abbildung 4 für die drei Farbanteile des RGB-Wertes genauer dargestellt.

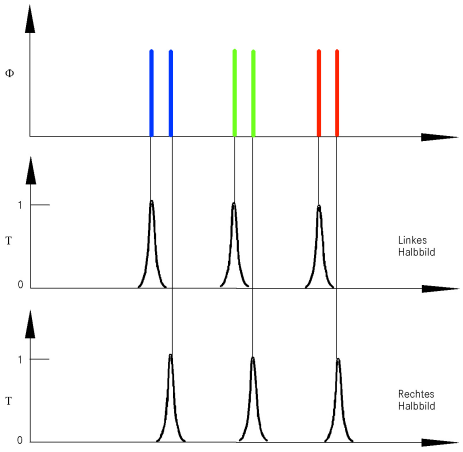

Abbildung 4: Infitec Wellenlänge [Jor08]

Dadurch ergibt sich für die zu entwickelnde Anwendung leider das Problem, dass der Punkt des Laserpointers auf dem rechten Auge nicht zu sehen ist, da die Wellenlänge etwa bei etwa 630 nm bis 650 nm liegt. Damit filtert die Infitec-Brille den Laserpunkt ungünstigerweise mit heraus. Dies beeinträchtigt jedoch nur den Anwender, nicht das eigentliche Tracken des Laserpunktes durch LIS3D.

3.2 LIS3D

Für das Erkennen des Laserpunktes gibt es im Prinzip nur eine Möglichkeit: Die Aufnahme des Laserpunktes mit einer Kamera, wobei diese entweder vor oder hinter der Rückprojektionswand steht. Da bei der Verwendung der ersten Methode der Anwender unerwünschterweise im Bild der Kamera stehen kann und somit den Laserpunkt verdecken könnte, steht die Kamera für LIS3D ebenso wie die Beamer hinter der Leinwand.

Die Kamera selbst ist eine *IDS uEye UI-1545LE-M Kamera* mit einem eingesetzten Objektiv *Pentax H612A2*. Beide Teile sind in Abbildung 5 zu sehen. Das Objektiv besitzt eine hohe Brennweite, damit die gesamte Leinwand und somit die komplette Rückprojektion abgefilmt werden kann.

Abbildung 5: *IDS uEye UI-1545LE-M* und das eingesetzte Objektiv *Pentax H612A* [BA08]

Beim Abfilmen der rückseitigen Wand mit einem Weitwinkelobjektiv - wie dem *Pentax H612A2* - ergeben sich aber auch Probleme. Zum einen stimmt die abgefilmte Fläche nicht mit der benötigten Fläche überein und zum anderen sind die aufgenommenen Bilder verzerrt. Die Ursache des letzten Punktes liegt in der Optik der Kamera. Die Schwere dieser sogenannten radialen Verzerrung ist von Kamera zu Kamera unterschiedlich. Daher muss die radiale Entzerrung für jede Kamera neu eingestellt werden. Ein Beispiel dieser Verzerrung aus der LIS3D Dokumentation findet sich in Abbildung 6.

Abbildung 6: Bild mit und ohne radialer Verzerrung [BA08]

Nachdem die radiale Entzerrung von LIS3D durchgeführt wurde, erfolgt noch die perspektivische Verzerrung. Diese dient dazu, den Filmausschnitt dem genauen Bild anzupassen. Abbildung 7 aus der LIS3D Dokumentation

erläutert das beispielhaft an einem Schachbrettmuster. Diese Vorgänge müssen nach jeder Bewegung der Kamera erneut durchgeführt werden.

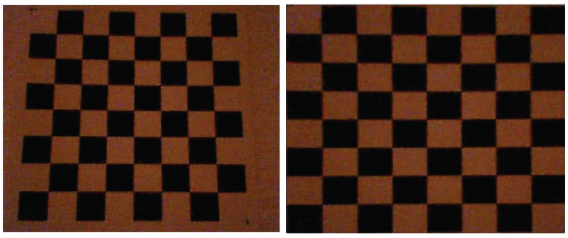

Abbildung 7: Bild mit und ohne perspektivischer Verzerrung [BA08]

Durch diese beiden Vorgänge erreicht man ein radial und perspektivisch entzerrtes Bild, welches nahezu der originalen Projektion entspricht. Mit diesem Ergebnis ist es nun möglich einen Laserpunkt in dem aufgenommenen Bild zu erkennen. Die eigentliche Durchführung dieser Kalibrierung wird zu einem späteren Zeitpunkt erläutert. Allerdings müssen vorher noch weitere Fehlerquellen ausgeschaltet werden, welche durch Beamer und Projektionsfläche hervorgerufen werden können.

Ein schwerwiegendes Problem ist zum Beispiel der in Abbildung 8 dargestellte Hotspot auf der Leinwand, welcher natürlich auch von der Kamera aufgenommen wird. Somit wäre innerhalb dieses Spots keinerlei Punkterkennung möglich. Dieses Problem kann man jedoch mit Hilfe zweier Verfahren abschwächen beziehungsweise komplett beseitigen: Zum einen mit Verwendung eines Infrarot-Filters oder aber andererseits durch die Polarisation des Lichtes.

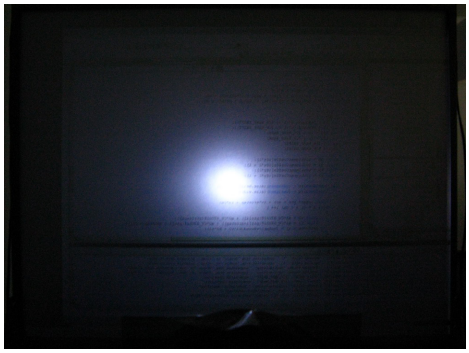

Abbildung 8: Hotspot auf der Projektionswand [BA08]

Ein Infrarot-Filter lässt nur das Licht im infraroten Bereich passieren. Da sich der Laserpointer nahe diesem Bereich befindet, wird dieses Licht auch durchgelassen. Diese Methode wurde im LIS3D Projekt getestet. Dabei stellte sich schließlich allerdings heraus, dass viele günstigere Kameras den infraroten Bereich nicht aufnehmen können, da in den meisten Modellen ein Schutzfilter installiert ist.

Die andere Möglichkeit ist eine Polarisation des Lichtes der Beamer. Dazu werden vor den Beamern und der Kamera entgegengesetzte Polarisationsfilter befestigt. Dadurch strahlen die Beamer Licht nur in eine Richtung aus und durch die polarisationserhaltende Oberfläche der Rückprojektionswand und dem entgegengesetzten Polarisationsfilter vor der Kamera ist das Licht von den Beamern für die Kamera nahezu unsichtbar. Zu sehen ist dies in Abbildung 9. Der leicht polarisierte Laser wird mit dieser Methode zwar ebenfalls leicht abgeschwächt, ist aber immer noch gut genug sichtbar, um seine Erkennung nicht zu behindern.

Dieser Aufbau wurde daher auch am Ende als beste Lösung angesehen und somit für das LIS3D Projekt verwendet.

Ein weiteres Problem ergibt sich aus dem Eintreffwinkel des Lasers auf der Leinwand. Wenn der Laser orthogonal auf die Leinwand zielt, kann man ihn sehr gut und deutlich erkennen. Je kleiner der Winkel wird, desto schlechter ist die Punkterkennung. Eine andere Oberfläche oder auch ein stärkerer Laser können dieses Problem vermindern.

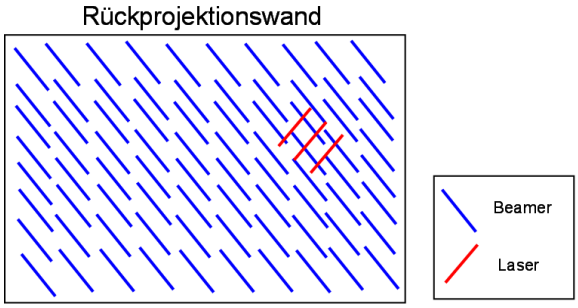

Abbildung 9: Polarisiertes Licht der Beamer und Laser auf der Rückprojektionswand

Damit wären die Grundlagen erklärt und der nächste Abschnitt befasst sich daher mit der praktischen Anwendung des LIS3D Projektes.

4 Verwendung einer LIS3D Anwendung

Dieses Kapitel beinhaltet Aufbau, Kalibrierung und Anwendung der Hard- und Software für eine LIS3D Anwendung.

4.1 Hardware Aufbau

Wie im vorhergehenden Kapitel besprochen, müssen zuerst die Polarisationsfilter an den Beamern und vor der Kamera angebracht werden. Zum Aufbau werden drei Polarisationsfilter mit den ungefähren Maßen 100 mm x 100 mm benötigt. Um die Richtung der Filter zu bestimmen, lässt sich ein einfacher Trick benutzen. Man legt zwei Polarisationsfilter übereinander. Wenn man nun durch die beiden Filter durchschauen kann sind diese somit gleichgerichtet. Sollte man einen dritten darüberlegen, muss die gesamte „Filterkonstruktion“ undurchlässig werden. Dieser letzte Filter ist somit das Gegenstück zu den anderen und wird für die Kamera benötigt. Einer der beiden gleichgerichteten Filter wird nun mit diesem Filter mit Hilfe eines Klebestreifen verbunden. Abbildung 10 verdeutlicht diese Konstruktion. Wenn die beiden Filter zusammengeklebt wurden, erfolgt das Anbringen an den Beamern. Dazu werden die Filter

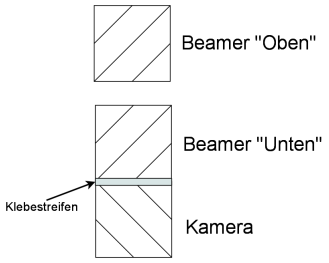

Abbildung 10: Aufbau

mit weiteren Klebestreifen vorsichtig an den Beamern befestigt, so dass der einzelne Filter über dem oberen Beamerobjektiv und die beiden zusammengeklebten Filter über dem unteren Objektiv und über der Kamera plaziert wurden. Abbildung 11 stellt den fertigen Aufbau im Labor der Hochschule Darmstadt dar.

Abbildung 11: Aufbau der Kamera und der Beamer

Das einzige Problem, welches sich hierbei ergeben könnte, wäre das Anbringen falschgerichteter Filter. Da die unteren Filter jedoch sowieso umgekehrt zueinander stehen müssen, ließe sich das ganze durch ein schnelles Umdrehen der Filter lösen.

4.2 Kalibrierung

Nachdem die Hardware erfolgreich installiert wurde, muss die Kamera kalibriert werden. Dieser Vorgang sollte möglichst nur von zwei Personen durchgeführt werden. Die radiale Entzerrung muss in diesem Fall nicht mehr vorgenommen werden, da sie bereits für die *IDS uEye* Kamera mit dem Pentax Objektiv vorhanden ist. Allerdings muss noch die perspektivische Entzerrung vorgenommen werden.

Dies erfolgt durch den Aufruf der Datei *Calibration.bat* aus dem LIS3D Source Verzeichnis. Falls diese nicht mehr existiert kann man mit dem folgenden Befehl die Kalibrierung starten.

```
LIS3Dsrc\bin\release\UFAConfigLoader.exe
          -c Perspectivewarp_ueye.config
```

Nach dem Start des Programms öffnet sich ein Fenster mit der aktuellen Kamera Aufnahme, ansonsten ist die Kamera nicht an den Rechner angeschlossen. Wenn man nun orthogonal mit einem Laserpointer von vorne auf die Rückprojektionswand leuchtet, muss auf dem ansonsten schwarzen Bildschirm ein verschwommener weißer Punkt erscheinen. Ist dies nicht der Fall, liegt eventuell ein Fehler mit den Polarisationsfiltern vor. Diese sollten erneut auf ihre richtige Anordnung beziehungsweise Ausrichtung überprüft werden.

Wenn alles funktioniert und man den erkannten Laserpunkt irgendwo auf der Projektionswand sehen kann, muss man zuerst überprüfen, ob die Kamera richtig eingestellt ist und die gesamte Leinwand abdeckt. Dazu zeigt man mit dem Laserpointer in jede Ecke und überprüft, ob der Laserstrahl von der Kamera erfasst und angezeigt wird. Bei Erfolg sollte man nun irgendwo auf der Leinwand einen verschwommenen weißen Punkt sehen. Falls nichts zu sehen ist, muss man eventuell die Kamera etwas drehen, verschieben oder in der Höhe verstellen, um den Laser an allen vier Ecken erfassen und die gesamte Leinwand abfilmen zu können.

Nachdem dies geschehen ist, erfolgt die Softwarekalibrierung. Dazu drückt man im gleichen Programm die Taste **R** und betritt damit den Kalibrierungsmodus. Eine Person beleuchtet nun mit dem Laserpointer nacheinander alle vier Ecken. Währenddessen zieht die andere Person die jeweiligen Eckpunkte, welche im Programm als weiße Kästchen dargestellt sind, auf den erkannten Laserpunkt an der Leinwand. Abbildung 12 erläutert diesen Vorgang.

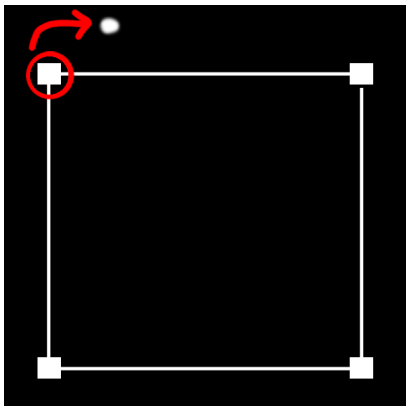

Abbildung 12: Verwenden der Kalibrierung

Der nächste Schritt ist zwar nicht unbedingt notwendig, wird aber von den Entwicklern von LIS3D empfohlen. Man beendet dazu das Programm noch nicht, sondern öffnet einen Datei-Explorer und wechselt in den folgenden Pfad:

```
\LIS3D\LIS3Dsrc\bin\release\
```

Anschließend löscht man dort die nachfolgende Datei:

```
perspectivewarp.yml
```

Danach kehrt man wieder in das Kalibrierungsprogramm zurück. Hier drückt man nun die Taste **S** zum Speichern. Wenn man sich nicht sicher ist, ob das Speichern geklappt hat kann man daraufhin im Datei-Explorer in dem Verzeichnis nachschauen, ob eine neue *perspectivewarp.yml* Datei mit einer Größe von 0 KB entstanden ist. Im Kalibrierungstool drückt man nun die ESC-Taste um das Kalibrierungsfenster zu beenden und schließt dann die Konsolenanwendung. Nun kann man die Datei mit einem Texteditor öffnen. Sie sollte etwa diese Form haben:

```
%YAML:1.0
calibrationMatrix: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 1.23, -0.01, -59.79,
           0.06,  1.31, -110.08,
           3.93,  7.88, 1. ]
```

Bei der Kalibrierung kam es leider häufiger vor, dass der Speichervorgang fehlerhaft verlief. So wurden zum Beispiel mehr Daten in die Datei geschrieben als nötig. Danach konnte man die Kalibrierung nicht erneut starten. Vor dem Start musste die *perspectivewarp.yml* gelöscht werden, um das Problem zu beheben.

Wenn die Datei *perspectivewarp.yml* jedoch die weiter oben genannte Form aufweist, so hat das Speichern funktioniert. Die Datei muss nun in das Verzeichnis

```
LIS3D\LIS3Dsrc\config\perspectivewarp\
```

kopiert und gegebenenfalls überschrieben werden. Damit steht LIS3D nun komplett konfiguriert für die 2D Punkterkennung beziehungsweise das 2D Punkttracking zur Verfügung.

4.3 Integration von LIS3D in Ogre

Diese Punkt behandelt die Verwendung der 2D Punkterkennungsmethode des LIS3D Projektes in einer eigenen Anwendung auf Basis der Ogre-Engine. Dabei ist zu beachten, dass in diesem Paper nur auf die Verwendung mit Windows eingegangen wird. Unter Linux sollte es jedoch, abgesehen von linuxspezifischen Definitionen und Funktionsaufrufen kaum Abweichungen geben. Diese Anleitung basiert zudem auf dem LIS3D-Projekt *Point Detection*, welches zum weiteren Verständnis und als Programmierbeispiel herangezogen werden kann. Dieses Beispiel diente auch als Vorlage für die LIS3D Integration in das AM3D Projekt und ist eine minimalistische und somit leicht verständliche Anwendung.

Um die eigene Ogreanwendung für den Gebrauch von LIS3D einzurichten sind eigentlich nur wenige Schritte erforderlich. Der wichtigste dabei ist das Einrichten eines Threads, welcher mit Hilfe der LIS3D Bibliothek die Kamerabilder analysiert und versucht, Laserpunkte auf der Leinwand zu erkennen. Dieser Thread sollte während der Initialisierung von Ogre erstellt und gestartet werden. Anschließend lassen sich diese Punktdaten am Besten in einem Ogre Framelistener in der Funktion *frameStarted* für jeden Frame abfragen. Dies geschieht, indem der LIS3D-Thread mit Hilfe von Mutexen gesperrt wird, um Speicherzugriffsfehler zu vermeiden.

```
_mutex.lock();
LIS3DCore::TrackedPoint::TrackedPointList list = _pointList;
_mutex.unlock();
```

Nach der Sperrung kann die Liste, welche die Punktdaten enthält in den Ogre-Thread kopiert und der LIS3D-Thread wieder entsperrt werden. Nun lassen sich die kopierten Daten, welche die Punkte in einer eigenen Vektorklasse beinhalten, weiterverarbeiten. Ein Zugriff auf die Position eines Elementes könnte dann wie folgt aussehen, wobei *point* ein Element der weiter oben erstellten Liste *list* wäre und *it* ein Iterator auf diese Liste.

```
const LIS3DCore::TrackedPoint& point = (*it);
const UFACore::Vector3d& pos = point.getPositionVector();
```

Der Vektor *pos* und seine XYZ Komponenten lassen sich nun weiter zur Berechnung von Bewegungen, Animation oder ähnlichem einsetzen.

Um die Anwendung abschließend kompilieren zu können, müssen die folgenden Includes des LIS3D Projektes eingebunden werden.

```
#include <UFACore/ConfigLoader.h>
#include <UFACore/TinyXMLParser.h>
#include <UFACore/exceptions/Exception.h>
```

Außerdem wird zur Laufzeit eine große Anzahl von Dynamic Link Libraries (dll) benötigt. Da diese Liste zu lang ist, um alle Dateien namentlich zu nennen, wird empfohlen den gesamten Inhalt des Ordners *LIS3D/LIS3Dsrc/bin* in das ausführende Verzeichnis der eigenen Ogreanwendung zu kopieren. Je nach Ogreversion kann es zudem vorkommen, dass LIS3D gegen den Quellcode von Ogre neu kompiliert werden muss. So wurde AM3D mit der Ogre Version 1.6, Codename *Shoggoth* erstellt, während das Team des LIS3D Projektes noch eine ältere Version der Engine benutzte.

5 AM3D

Nachdem LIS3D nun zur Genüge abgehandelt worden ist, widmet sich das folgende Kapitel dem Entwicklungsprozess der eigentlichen Anwendung AM3D. Dabei sollen wichtige Meilensteine in der Anwendung näher erläutert werden und auf Probleme während der Entwicklung eingegangen werden.

5.1 Überblick

Die Spielidee von AM3D basiert wie bereits erwähnt auf dem aus der 2D-Welt stammenden *Missile Command* der 80er Jahre. Genau wie damals muss man auch heute in AM3D mit Raketen eine Flut von heranfliegenden Meteoriten bekämpfen welche die Erde zu zerstören drohen. Abbildung 13 zeigt einen Screenshot der fertigen AM3D Anwendung mit einem Viewport und mit Mauscursor.
Im Gegensatz zur alten 2D Version spielt AM3D jedoch in einer 3D Welt. Während man somit früher einfach irgendwo auf den Bildschirm klickte und sich um die dritte Achse des Koordinatensystems keine Sorgen machen musste, ist dies in AM3D anders. Da die Raketen entlang der Z-Achse in die Spielwelt hineinfliegen, muss der Spieler auch die räumliche Tiefe der ankommenden Meteoriten abschätzen, um diese zu treffen und damit zu zerstören.

Abbildung 13: Screenshot von AM3D mit Mauscursor (rotes Kreuz rechts im Bild)

5.2 Aufbau & Ablauf

Die Spielwelt besteht aus verschiedenen Teilen. Zum einen ist das Terrain zu nennen, welches auf Basis einer Heightmap aufgebaut und mit Texturen überzogen wird. Zum anderen wird ein Skydome um die Welt gelegt, um einen blauen bewölkten Himmel zu simulieren. Die Meteoren werden zur Laufzeit dynamisch erzeugt, mit Geschwindigkeiten versehen und in Richtung eines zufälligen Punktes am Boden bewegt. Die Raketen starten ebenso von den Punkten, an denen sie erstelllt wurden. Das entspricht, abgesehen von der Y-Koordinate, der Kameraposition. Die Y-Koordinate ist etwas niedriger als der echte Wert der Kamera, um beim Benutzer den Eindruck zu erwecken, dass die Rakete vor der Leinwand aus dem Boden startet und anschließend in Richtung des Zieles fliegt. Die eigentliche Zielkoordinate, die von der Rakete angesteuert wird, ist dagegen vom Spieler festgelegt und die zu erreichende Raumtiefe ist auf einen festen Wert gesetzt. Sie entspricht der in Abbildung 14 angezeigten Ebene, welche im folgenden Punkt näher erläutert wird.

Der Ablauf des Spiels wird in Abbildung 15 in einem leicht vereinfachten Aktivitätsdiagramm dargestellt. Mit dem Start der Anwendung und dem Einstieg in die Spielschleife werden die dort aufgezeigten Funktionen jeden Frame durch Aufruf in der Funktion *frameStarted* des Ogre *FrameListeners* ausgeführt. So werden nacheinander alle Eingabegeräte (Tastatur, Maus, Laser) auf Eingaben abgefragt und darauf reagiert. Somit werden entweder Optionen geändert (Tastatur) oder Raketen abgeschossen (Maus, Laser). Darauffolgend werden alle Partikelsystem aktualisiert und gegebenenfalls angehalten und entfernt. Anschließend wird überprüft, ob neue Meteoren erzeugt werden müssen. Diese werden - falls nötig - zufallsmäßig aus einigen möglichen Modellen ausgewählt und zudem mit einer Fluggeschwindigkeit und einer Richtung versehen. Nach der Bewegung erfolgt ebenfalls für alle Meteoren eine Überprüfung, ob sie bereits mit dem Terrain kollidiert sind. Falls dies der Fall sein sollte, werden sie gelöscht und an ihrer Aufschlagsposition erscheint eine Explosion. Dieselbe Bewegung mit Überprüfung wird darauffolgend bei den Raketen durchgeführt. Hierbei wird jedoch auf Kollisionen mit Meteoren überprüft. Zudem erfolgt eine Überprüfung, ob die Rakete bereits am Ziel angekommen ist. Bei einem positiven Test wird die Rakete ebenso gelöscht und eine Explosion eingeführt. Dies verhindert eine riesige Liste an Geschossen und damit eine Menge Rechenaufwand. Zuletzt wird noch die Informationsanzeige am oberen Bildschirmrand aktualisiert, bevor die Spielschleife wieder von vorne beginnt. Das Programmende schließlich erfolgt, sobald der Benutzer das Programm durch Drücken der Taste Escape beendet.

5.3 Steuerung

Die Anwendung bietet zwei verschiedene Möglichkeiten zur Steuerung. Zum einen eine reguläre Maussteuerung, zum anderen die mit Hilfe von LIS3D implementierte Steuerung mit Laserpointern.

Um innerhalb der Spielwelt jegliche Spielerinteraktion bearbeiten zu können, wird für beide Eingabemethoden auf Rays zurückgegriffen. Hierbei wird eine Ebene (Plane) entlang der X und Y Achsen in die Spielwelt eingebaut. Diese Ebene wird zudem auf der Z-Achse in Richtung der Farclippingplane der Kamera bewegt. Wenn der Benutzer nun innerhalb seiner View in die Welt klickt beziehungsweise mit dem Laser beleuchtet wird ein unsichtbarer Strahl (Ray) aus der Kamera in Richtung der Ebene gezeichnet. Sollte dieser Strahl ohne Hindernisse auf die Ebene treffen, ist der Klick gültig und liefert Koordinaten zurück. Diese Koordinaten entsprechen dem Schnittpunkt des Rays mit der aufgespannten Ebene. Abbildung 14 stellt dieses Prinzip genauer dar. Die Grafik zeigt die Spielwelt aus Sicht der Kamera, wobei zu beachten ist, dass nur der rotgefärbte Bereich des Bildes zum Frustum der Kamera gehört. Zu sehen sind weiterhin die Koordinatenachsen, wobei die Z-Achse direkt aus dem Bild heraus auf den Betrachter zukommt. Außerdem sind das Terrain (braun) und die Ebene (blau) im Hintergrund, welche als Schnittpunkt der Koordinaten dient, sichtbar. Der schwarze Halbkreis in der unteren Mitte des Bildes repräsentiert die Kameraposition.

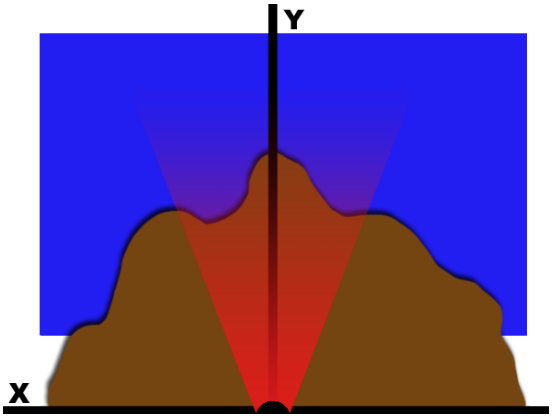

Abbildung 14: pseudohafte Darstellung der Spielwelt aus Sicht der Kamera

Nur dort wo die blaue Schnittebene zu sehen ist werden somit Klicks akzeptiert. Dies verhindert Abschüsse von Raketen auf die Erde und ermöglicht dennoch ein Zielen im Raum.

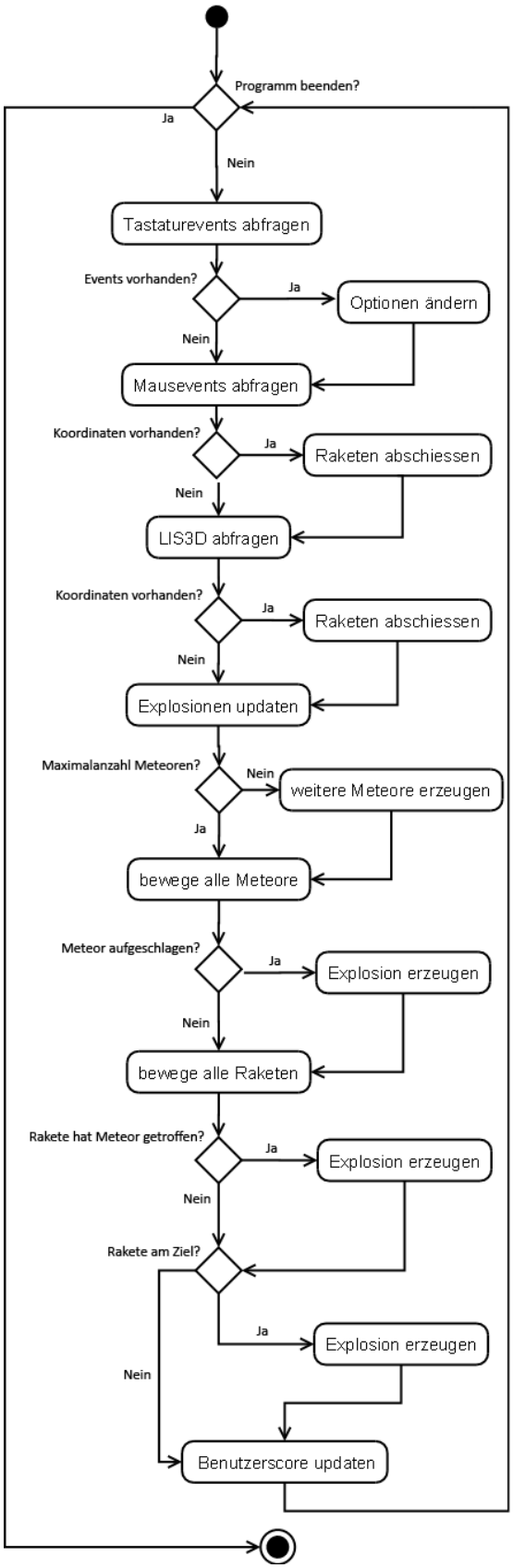

Abbildung 15: Aktivitätsdiagramm

5.4 Kollisionsabfragen

Kollisionen werden mit sogenannten Axis-Aligned Bounding Boxen (AAB) überprüft. Das sind Boxen, die ihr zugehöriges Objekt umschließen und zudem an den Koordinatenachsen ausgerichtet sind. Da Ogre Funktionen zum Erzeugen und Überprüfen dieser Boxen von Haus aus mitbringt, ist eine Implementierung sehr einfach. Zudem ist der Vorgang des Zielens und Treffens in AM3D schwierig genug, so dass etwas größere Ziele wie Bounding Boxen die Schwierigkeit im Rahmen halten. Zwei Boxen können so gegeneinander auf Überschneidungen getestet werden. Der Rückgabewert dieses Tests ist wiederum eine AAB. Solange ihr Wert größer als null ist, hat eine Überschneidung beider Boxen und somit eine Kollision stattgefunden. Abbildung 16 erläutert den Unterschied zwischen einer normalen orientierten Bounding Box auf der rechten Seite und einer AAB auf der linken Seite der Grafik.

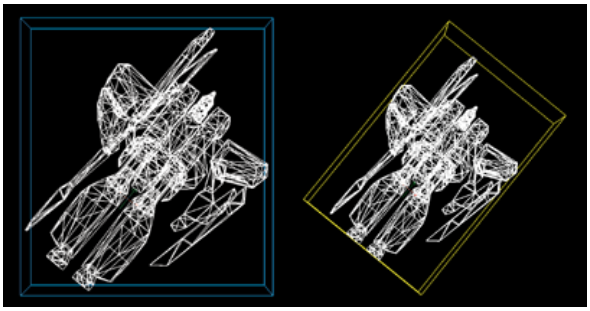

Abbildung 16: Links: Axis-Aligned Bounding Box, Rechts: Oriented Bounding Box [Lan08]

5.5 Partikeleffekte, Modelle & Grafiken

AM3D verwendet verschiedene Partikeleffekte, um den Anwendern ein wenig Realismus wie Explosionen zu bieten und das Spiel etwas grafiklastiger zu gestalten. Zudem dienen diese Effekte als Rückmeldung, ob ein Objekt getroffen und zerstört wurde oder nicht. Zur Anwendung kommt dabei das Ogreplugin *ParticleUniverse* [vM08], welches die direkt in Ogre implementierten Partikelfunktionen kapselt und in AM3D für alle verwendeten Partikeleffekte zuständig ist.

Die einzelnen Partikeleffekte betreffen dabei - wie bereits erwähnt - die Explosionen sowohl der Meteore wie auch der Raketen. Dabei explodieren Meteore als Feuerball, während Raketen sich mit einem Feuerwerk zerstören. Weitere Effekte betreffen die Feuer- beziehungsweise Rauchschweife, welche sowohl Raketen als Meteore hinter sich herziehen. Zudem werden beim Auftreffen von Meteoren auf der Erde an diesen Stellen für gewisse Zeit Feuereffekte erstellt.

Für die 3D-Modelle wurden in ersten Versuchen eigene Objekte mit Hilfe von Milkshape 3D[2] und Deled[3] erstellt. Diese Modelle waren jedoch zu unglaubwürdig, so dass nach einigen Recherchen verschiedene freie Modelle gefunden werden konnten. Die Modelle der Meteore stammen so

[2] http://chumbalum.swissquake.ch
[3] http://www.delgine.com

von der Homepage von *Scott Hudson* [Hud08], Professor für Electrical Engineering an der Washington State University. Abbildung 17 zeigt einen Screenshot des Modells *Toutatis*.

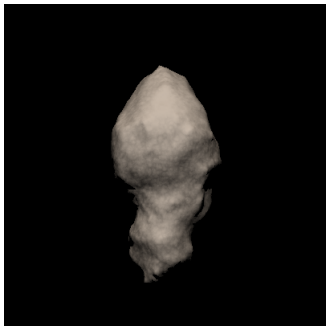

Abbildung 17: Darstellung des Asteroidmodells Toutatis [Hud08]

Das Raketenmodell [vir08] basiert auf der Saturn V Rakete, welche auch für die Apollomission zum Einsatz kam. Der Screenshot in Abbildung 18 liefert einen Eindruck dieses Modells.

Abbildung 18: Darstellung des Raketenmodells Saturn V [vir08]

5.6 Stereosicht

Die Stereosicht wurde als letzter Schritt in das Programm integriert. Die Ogre Engine bietet dazu die Erstellung mehrerer Kameras und Viewports mit unterschiedlichen Größen und Seitenverhältnissen. So konnte die Stereosicht innerhalb kürzester Zeit implementiert werden. Abbildung 19 zeigt einen Screenshot der AM3D Anwendung mit zwei Viewports für den oberen und unteren Beamer beziehungsweise für das rechte und linke Auge.

5.7 Probleme

Im Verlauf der Entwicklung von AM3D kam es zu verschiedensten Problemen, die überwunden werden mussten. Die wichtigsten werden hier nun im weiteren Verlauf kurz vorgestellt.

Eines der ersten Probleme hing mit der Dokumentation von Ogre zusammen. Es existiert zwar eine Variablen- und Methodenübersicht in Form einer HTML-Dokumentation,

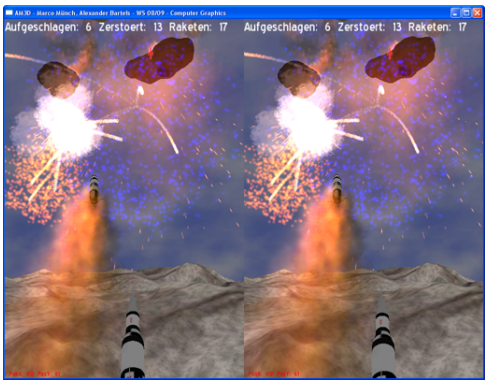

Abbildung 19: Screenshot von AM3D mit zwei Viewports

jedoch ist die vorhandene Anzahl zu hoch, um einen schnellen Überblick von Ogre zu gewährleisten. So wurde während der Entwicklung der Kollisionen längere Zeit gerätselt, warum keinerlei Treffer erkannt wurden. Es stellte sich heraus, dass Ogre zwei Funktionen zur Rückgabe einer Bounding Box enthält. Die Funktion *getBoundingBox* gibt die Bounding Box relativ zum lokalen Koordinatensystem des Objektes zurück, während die Funktion *getWorldBoundingBox* die Bounding Box relativ zum Weltkoordinatensystem zurückgibt. Mit Hilfe der zweiten Funktion konnte das Problem der Kollisionen schließlich gelöst werden ohne die Boxen eigenstandig transformieren zu müssen. Die wenigen Beispiele im Wiki der Engine schneiden zudem nur einen kleinen Teil des riesigen Gesamtkomplexes Ogre an. Weitere kleinere Quellcodebeispiele sind zudem teilweise fehlerhaft oder unvollständig und auch das Forum ist relativ unübersichtlich und die Suchfunktion unbefriedigend. So wird der Einstieg in Ogre erschwert.

Weitere Probleme traten mit verschiedenen Abhängigkeiten auf. So benutzt AM3D eine Bibliothek für die Partikeleffekte mit dem Namen *ParticleUniverse*. Dieses Projekt für Visual Studio 2008 muss selbstständig kompiliert werden. AM3D wurde jedoch mit der Version 2003 des Visual Studio erstellt und somit konnte das vorgefertigte Projekt nicht verwendet werden, da die erstellten Bibliotheken inkompatibel mit AM3D waren. Es wurde daher anhand der vorhandenen Projektdatei eine gleichwertige Datei für Version 2003 erstellt, mit welcher das Ogreplugin schließlich erfolgreich kompiliert werden konnte.

Auch bei der Erstellung der Bibliotheken des LIS3D Projektes kam es zu ähnlichen Problemen. Bei ersten Versuchen das LIS3D Projekt zu kompilieren, fiel auf, dass verschiedene abhängige Bibliotheken nicht vorhanden waren. Leider waren diese wichtigen Dateien nicht auf der LIS3D Website zu finden. Erst eine Kopie des auf den Laborrechnern verwendeten LIS3D schaffte hier Abhilfe. Zusätzlich musste später auch noch das gesamte LIS3D Projekt gegen die von AM3D verwendete Ogreversion neu kompiliert werden.

Weitere Probleme betrafen das Auftreten eines Trojaners, welcher sich auf dem verwendeten Rechner eingeschlichen hatte. Dieser wurde bei jeder Erstellung der Anwendung als

Aufruf mit eingebunden womit das Programm beim ersten Tastendruck sofort abstürzte. Dieses Problem konnte kurz darauf durch den Gebrauch eines Virenscanners gelöst werden.

6 Ausblick

Die Spielmechanik selbst betreffend gibt es sicherlich unendlich viele Möglichkeiten zur Erweiterung von AM3D, angefangen bei einer Highscore über hübschere Grafikeffekte bis zu verschiedenen Gegnertypen wie zum Beispiel UFO's.

Eine wirklich interessante Erweiterung wäre jedoch der Ausbau der Methoden zur 2D Lasererkennung, bis die gesamte Anwendung nur per Laser gesteuert werden könnte. In der aktuellen, finalen Version von AM3D gibt es viele Shortcuts, um unter anderem den Meteoritenspawn zu stoppen, Bounding Boxen anzeigen zu lassen oder Partikeleffekte zu deaktivieren. All diese Shortcuts könnte man über „Lasergesten" realisieren, um somit den Laser als echten Maus- oder auch Tastaturersatz in die Anwendung einzubringen. Auch die Steuerung einer grafischen Benutzeroberfläche zum Ändern von Einstellungen, Betrachten der Highscore oder ähnlichem wäre damit möglich.

Da die Anwendung nicht alle Möglichkeiten des LIS3D Projektes ausschöpft, wäre es denkbar sie in diesem Bereich zu erweiteren. Beispielsweise wäre es möglich mit dem 3D Punkttracking den Zielpunkt der Raketen direkt im 3D Raum zu bestimmen, um somit noch genauer zielen zu können. Allerdings ist die dafür benötigte Laserapperatur sehr unhandlich und wohl eher weniger zum Bedienen eines Spieles geeignet, bei dem es auf Geschwindigkeit ankommt.

7 Zusammenfassung

Im Rahmen dieses Praktikums wurde sich ausgiebig mit den Verfahren der Laserpunktinteraktion auseinandergesetzt. Die dabei verwendete Bibliothek LIS3D erwies sich schlußendlich als sehr einfach integrierbar. Dafür war die notwendige Einrichtung und Kalibrierung der Hardware ohne vorherige Kenntnis relativ kompliziert. Leider ist diese auch mit der Kenntnis der Projektdokumentation von LIS3D nicht direkt verständlich. Durch Nachfrage bei den zuständigen Personen des LIS3D Projektes konnten jedoch auch diese Vorgänge relativ unkompliziert verstanden und umgesetzt werden.

Zudem erwies sich auch Ogre trotz einigen Einstiegsschwierigkeiten und Problemen als mächtige, zuverlässige und dennoch einfach handhabbare Engine mit großem Potential.

Auch wenn nicht alle geplanten Features, wie unter anderem zerstörbare Gebäude auf dem Terrain, aus Zeitgründen ins Spiel eingebaut werden konnten, entstand dennoch eine funktionsfähige, durch Laser steuerbare, Anwendung, die weiteren Projekten auf LIS3D Basis eine Grundlage bieten kann.

Literatur

[BA08] Bernd Amend, Ronny Giera, Philip Hammer Hendrik Schmedt: *Laser Interaction System 3D*. http://dl.sharesource.org/lis3d/lis3d_dokumentation_v1.0.1.pdf, 2008.

[Geo02] George, Gregory D.: *The Top 20 Greatest Games For Your Atari*. http://www.ataritimes.com/article.php?showarticle=272, 2002.

[Haw08] Hawlisch, Martin: *Stereoskopie*. http://de.wikipedia.org/wiki/Stereoskopie#Stereobildpaar, 2008.

[Hud08] Hudson, Scott: *3D asteroid models, courtesy of Scott Hudson, Washington State University*. http://www.tricity.wsu.edu/~hudson/Research/Asteroids/index.htm, 2008.

[Jor08] Jorke, Helmut: *INFITEC - Die Technologie der Wellenlängenmultiplex Visualisierungssysteme*. http://www.infitec.net/INFITEC.pdf, 2008.

[Kun08] Kungfuman, Don Magnifico: *Missile Command*. http://de.wikipedia.org/wiki/Missile_Command, 2008.

[Lan08] Lander, Jeff: *When Two Hearts Collide: Axis-Aligned Bounding Boxes*. http://www.gamasutra.com/features/20000203/lander_01.htm, 2008.

[vir08] virez: *Apollo 13 Rocket*. http://www.turbosquid.com/FullPreview/Index.cfm/ID/282579, 2008.

[vM08] Merode, Henry van: *Particle Universe*. http://www.fxpression.com, 2008.