|  |  |
| --- | --- |
|  | |
| Projektarbeit | |
| Realisierung des Spiels Breakout mit C++ und OpenGL | |
| im Studiengang Informatik  an der Fakultät Informatik  der Hochschule Karlsruhe  vorgelegt von  Marco Pattke | |
|  |  |
| Referent: | Prof. Dr. Pape |
|  |  |
|  |  |
| Eingereicht am: | 26.02.2015 |

Inhaltsverzeichnis

[1 Einleitung 5](#_Toc412718633)

[2 Grundlagen 7](#_Toc412718634)

[2.1 Was ist OpenGL? 7](#_Toc412718635)

[2.2 Was leistet OpenGL? 7](#_Toc412718636)

[2.3 Game-Engine 8](#_Toc412718637)

[3 Umsetzung 9](#_Toc412718638)

[3.1 Erste Schritte 9](#_Toc412718639)

[3.2 Render-Ablauf 11](#_Toc412718640)

[3.2.1 PreRender - TODO 11](#_Toc412718641)

[3.2.2 Rendern aller Objekte – TODO 11](#_Toc412718642)

[3.2.3 PostRender – TODO 11](#_Toc412718643)

[3.3 Aufteilung in Klassen - TODO 12](#_Toc412718644)

[4 Verbesserungen 14](#_Toc412718645)

# Einleitung

Das Computerspiel Beakout wurde ursprünglich 1976 von Steve Wozniak als Arcade-Spiel rein in Hardware realisiert. Das Spiel wurde jedoch seit dem auf Atari und Apple portiert und es gibt zahlreiche Klone, die direkt Online, Auf dem Smartphone etc. spielbar sind.

Ziel des Spiels ist es, mit Hilfe des vom Spieler kontrollierten Schlägers den Ball so abzulenken, dass der Ball "Mauersteine" trifft um diese zu zerstören. Erst nach dem alle Steine zerstört wurden, gilt das Level als gewonnen. Sollte der Ball nicht vom Spieler abgelenkt werden können und er berührt den unteren Bildschirm-Rand, ist das Spiel verloren.

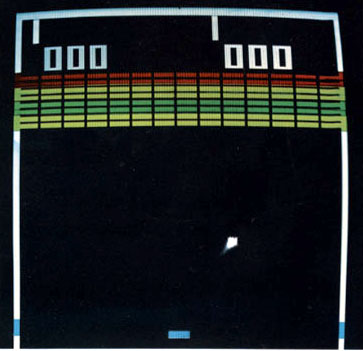


Abbildung 1 - Bild vom ursprünglichen Breakout auf dem Atari

Ziel dieser Projektarbeit ist es, dieses Spiel, angelehnt an das originale, mit der Programmiersprache C++ und der Grafikbibliothek OpenGL zu realisieren. Auf Tonausgabe wird keinen Wert gelegt.

# Grundlagen

## Was ist OpenGL?

OpenGL (Open Graphics Library) ist eine plattformunabhängige Grafikbibliothek, die inzwischen zu einem weltweiten Standard für 3D-Grafikprogrammierung und zu einer der meist genutzten 3D APIs geworden ist. Sie wird sehr vielfältig verwendet: In der Industrie, in Forschungszentren, in der Spieleprogrammierung und in der Lehre.

Die Plattformunabhängige wird dadurch erreicht, dass OpenGL tatsächlich nur die Spezifikation einer Bibliothek ist, für die verschiedenste Implantierungen existieren, teilweise direkt von Graphikkarten unterstutzt. Es gibt OpenGL Bibliotheken in fast allen erdenkbaren Programmiersprachen, z.B. für C, C++, Java, C# und sogar für Google GO.

OpenGL wurde 1992 von Silicon Graphics entwickelt, wird jedoch seit Juli 2006 von der Khronos Group verwaltet. Die Khronos Group ist eine Vereinigung mehrerer Unternehmen, die sich für die Erstellung und Verwaltung offener Standards im Multimediabereich auf einer Vielzahl von Plattformen und Geraten einsetzt. Die Aktuelle Version von OpenGL ist die Version 4.5 (seit 11. August 2014). Für OpenGL wird keine Lizenz benötigt, da seit 2009 alle Teile von OpenGL als freie Software lizensiert wurde..

## Was leistet OpenGL?

OpenGL stellt eine universelle Methode dar, relativ unkomplizierte komplexe 3D-Grafiken zu erzeugen. Durch die gezielte Anordnung einfacher Grafikelemente werden komplexe Körper gebildet, welche 3D-Szenen bilden. Primitiven sind Punkte, Linien und Flächen (Polygone). Aufbauend auf diesen Grundgeometrien hält OpenGL auch Grundkörper wie Kugeln, Zylinder oder gekrümmte Oberflächen in gesonderten Bibliotheken bereit.

OpenGL wurde als Zustandsmaschine entworfen. Attribute eines Objektes, wie z.B. Farben beim Zeichnen, bleiben bis zur Änderung erhalten. Anbieter wie z.B. Grafikkartenhersteller können diese Zustandsmaschine um eigene Zustände erweitern und Funktionen ausbauen.

## Game-Engine

Eine Game-Engine ist eine Sammlung von Technologien und Hilfsmitteln, die die Erstellung von Spielen in großem Maße unterstützt und erleichtert. Eine Engine umfasst dazu eine große Menge von Funktionen, die von nahezu allen Spielen verwendet werden, und keinen spezifischen Spielinhalt. Eine Game-Engine wird auch des Öfteren als „Betriebssystem für Spiele“ bezeichnet. Es existieren Engines, die sich nur für die Erstellung von Spielen eines bestimmten Genres eignen und dafür optimiert sind, wie z.B. einige Engines für Spiele aus der Ego-Perspektive, die nur in Gebäudekomplexen angesiedelte Szenarien und keine weitläufigen Außenlandschaften zulassen, dafür aber über sehr ausgefeilte Beleuchtungsmodelle verfügen. Andere Engines hingegen versuchen eine möglichst breite Bandbreite an Funktionalität anzubieten, um die auf ihrer Basis erstellten Spiele so wenig wie möglich im Genre und in den Möglichkeiten einzuschränken.

TODO: nochmal etwas anderster schreiben, erweitern

Als Aspekt einer Game-Engine habe ich in meiner Arbeit ist Austauschbarkeit der Render-Engine implantiert. Dadurch kann man zwischen OpenGL und DirectX wählen.

Was kommt rein:

Wie genau ist es aufgebaut, Klassen, Interfaces, kein quellcode (nurvergleich renderer )

Klassendiagramme

GAmeloop beschreiben, rechteck

Aktivitätsdiagramm für Rendering

Verwendeter Compiler, tools etc.

Blocksatz

Zitate etc. richtig

# Umsetzung

OpenGL ist eine reine Grafikbibliothek und kümmert sich daher nicht um die Verwaltung von Oberflächen, Puffern oder Rendercontexten. Um dies Bewerkstelligen zu können wird eine weitere Bibliothek benötigt, um OpenGL mit dem darunter liegenden Betriebssystem zu verbinden. Die ältestes Lösung dafür ist GLUT (OpenGL Utility Toolkit), diese Lösung wird jedoch seit 1998 nicht mehr weiterentwickelt und ist daher veraltet. Daher wurde das Breakout Projekt mit einer neueren, ebenfalls kostenlosen Bibliothek GLFW realisiert.

GLFW ist OpenSource, unterstützt mehrere Plattformen, Bildschirme, verschiedene Eingaben wie Tastatur, Controller und wird aktuell weiterentwickelt.

Als Entwicklungsumgebnung verwende ich Visual Studio C++ 2010 Express unter dem Betriebssystem Windows 7 64 Bit.

## Erste Schritte

Nach dem via Visual Studio eine Konsolenanwendung entworfen wurde habe ich GLFW 3 heruntergeladen und in das Projekt miteingebunden. Die die aktuelle Version von GLFW (3.1) nur mit Visual Studio 2012 funktioniert verwende ich die Version 3.0. Damit ich die Entwicklung des Spiels sowohl auf meinem Computer, als auch auf meinem Laptop vornehmen kann, wurden die kompletten benötigten Bibliotheken und Includes ins Projekt kopiert und in Visual Studio als Include- und Bibliotheksverzeichnis eingebunden, damit der Compiler und Linker diese Später findet und verwenden kann.

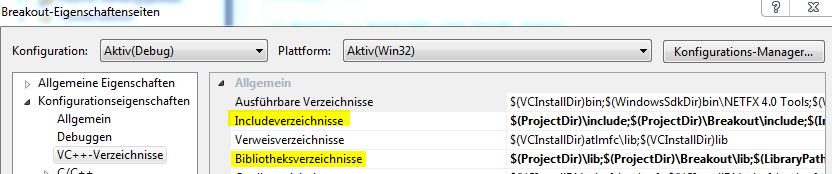


Abbildung 2 - Projekt Einstellungen in Visual Studio

Um GLFW nutzen zu können muss der Header der Bibliothek via include beigefügt werden:

#include <GLFW/glfw3.h>

Um ein simples OpenGL GLFW Fenster zu erzeugen reicht folgender Code:

glfwInit();

GLFWindow \*window = glfwCreateWindow(640, 480, "Simple example", NULL, NULL);

glfwMakeContextCurrent(window);

glfwSwapInterval(1);

Hierbei wird die GLFW Bibliothek initialisiert und ein Fenster erzeugt. Damit ein Rendering (mit Bewegungen usw.) stattfinden kann, muss danach eine (Endlos-) Schleife gestartet werden und alle Objekte in jedem Frame gerendert werden, was auch dem Prinzip einer Game-Engine entspricht (solange das Spiel läuft wird alles gezeigt, kann interagiert werden usw.).

Zum Rendern eines Primitives (in diesem Fall in Dreieck) muss OpenGL ein Beginn und ein Ende des Objekts, optional die Farbe und die Eckpunkte des Objekts mitgeteilt werden.

glBegin(GL\_TRIANGLES);

glColor3f(1.f, 0.f, 0.f);

glVertex3f(-0.6f, -0.4f, 0.f);

glColor3f(0.f, 1.f, 0.f);

glVertex3f(0.6f, -0.4f, 0.f);

glColor3f(0.f, 0.f, 1.f);

glVertex3f(0.f, 0.6f, 0.f);

glEnd();

Dieser Code rendert ein Dreieck mit drei verschiedenen Farben je Ecke.

Tastatureingabe können mit GLFW via Input-Events oder manueller Abfrage via

glfwGetKey(mWindow, glfwKey)

realisiert werden. Da bei Input-Events kein Zugriff auf die aktuelle Klasse via *this* möglich ist und ich keine globalen nicht Klassenvariablen verwenden wollte, habe ich mir für letzteres entschieden, was auch die Anbindung an DirectX einfacher macht, da hier eine ähnliche Abfrage werden kann.

## Render-Ablauf

### PreRender - TODO

### Rendern aller Objekte – TODO

Bei jedem Prozessor, Betriebssystem und Computer laufen die Render-Zeiten nicht genau gleich ab. Damit dennoch die Geschwinigkeit überall gleich ist, muss mit der Delta-Zeit, gerechnet werden, also die Differenz aus der aktuellen und der zuletzt bekannten Zeit. Damit können Unterschiedlichkeiten ausgleichen werden, da der Wert damit auch anders wird. Dieser Wert muss nun mit der gewünschten Beschleunigung bzw. Positionsänderung multipliziert werden. Damit bewegen sich die Objekte auf eine Feste Zeit gesehen immer gleich schnell, langsamere Prozessoren können mit einer größeren Delta-Zeit und somit einer größeren Beschleunigung die längere Zeit ausgleichen, sodass am Ende auf allen Computern der Ball nach Zeit x an derselben Stelle ist.

### PostRender – TODO

## Aufteilung in Klassen - TODO

Da viele Objekte des Spiels Breakout ähnliche Eigenschaften haben (Ball, Paddle und Bricks sind alles ähnliche Geometrische Figuren, benötigen alle einen Ursprung usw.) habe ich alle Ähnlichkeiten in eine Klasse gepackt. Für das Spiel werden lediglich feste und dynamische Vierecke benötigt, wobei die dynamischen viele Attribute der festen erben, den Ursprungspunkt, die Breite und Höhe und die Farbe. Alle gezeichneten Objekte im Spiel sind oder erben daher von der Klasse StaticBox. Diese realisiert alle Vierecke, welche gerendert und dargestellt werden können. Die Klasse DynamicBox eine eine besondere Form der StaticBox, welche zu dem Ursprungsvektor noch einen Beschleunigungsvektor hat und mit der Funktion void move(Vector2 moveVector) bewegt werden kann. Das Spiel ruft für jede DynamicBox die Funktion void Update(float deltaTime) auf, wodurch diese mit Hilfe ihrer aktuellen Beschleunigung ihre Position aktualisiert.   
Eine Besonderheit liegt noch bei den Bricks, da diese zerstört werden können. Dafür ist hier eine Extra Brick-Klasse notwendig, welche das zerstören realisieren kann. Dafür besitzt jeder Brick das Attribut int life und die Funktion void decLife(), welche das Leben um einen Punkt reduziert. Aktuelle haben zwar alle Bricks nur ein Leben, dadurch sind aber Bricks möglich, die öfters getroffen werden müssen.

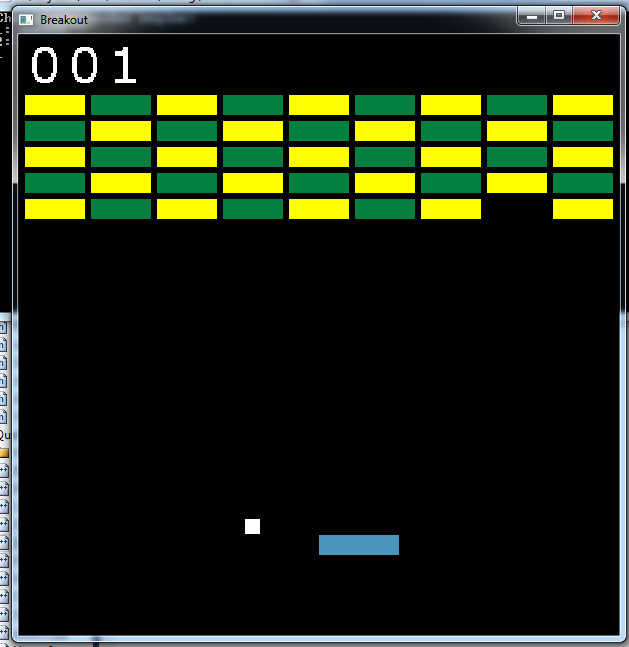


Abbildung 3 - Das fertige Breakout Spiel mit OpenGL

# Verbesserungen

Da ich durch dieses Projekt zum ersten Mal mit OpenGL oder DirectX in Berührung gekommen bin, sind einige Funktionen nicht Optimal gelöst und könnten verbessert werden.

Bei OpenGL besteht die Möglichkeit, einen Vertex-Buffer zu verwenden. Dieser reduziert die Funktionsaufraufe beim Erstellen von primitiven und behält sie in der Grafikkarte, damit sie nicht bei jedem Frame erneut gezeichnet werden müssen.

OpenGL kann von sich aus keine Zahlen oder Text Rendern, auch GLFW stellt hierfür keine Bibliotheken zur Verfügung. Es gibt verschiedene freie Bibliotheken, die dazu genutzt werden können, FreeTyp ist beispielsweise eine OpenSource Bibliothek dafür. Ich habe es jedoch mit keiner einzigen Bibliothek geschafft, Text bzw. die Zahlen vernünftig zu rendern und habe daher etwas kleines Eigenes gemacht, dass nur Zahlen von 1-9 als Textur auf einem Quad rendern kann. Dies ist nicht optimal gelöst, funktioniert aber. In DirectX ist es viel einfacher, da hier von Werk aus schon Funktionen geliefert werden um Text zu rendern. Hier sollte in OpenGL noch eine andere Möglichkeit gefunden werden, damit das Rendering des Texts auf beiden Grafik-Apis ähnlich abläuft und auch ähnlich aussieht und konfigurierbar ist, da aktuell unter OpenGL die Grafiken für die Nummern einmal erzeugt wurden und nur pixelweise skaliert werden können, was bei größeren Skalierung komplett pixelig wird.

Die Steuerung in dem Direct3D renderer ist leider noch etwas hackelig. Hier müsste eine bessere Methode gefunden werden, die Tastatur-Eingabe zu verarbeiten. In DirectX kann dazu auch Direct-Input verwendet werden, eine Windows Bibliothek für Tastatur eingaben, jedoch hatte diese bei den ersten Tests nicht richtig funktioniert und es wurde bei der Windows Variante, wie sie aktuelle im Projekt ist, geblieben.