Furtwangen University Simulation and Entertainment Engine

Fusee-Projektgruppe*unter Leitung von Professor Christoph Müller
Hochschule Furtwangen University

***Zusammenfassung* — Fusee („Furtwangen University Simulation and Entertainment Engine“) ist ein Projekt zur Entwicklung einer echtzeitfähigen und plattformunabhängigen 3D-Grafik-Engine. Dabei soll der Schwerpunkt und die Unique Selling Proposition dieser Engine – durch Unterstützung unterschiedlichster Plattformen, beispielsweise WebGL, Android, iOS oder XNA – in der Flexibilität liegen.**

***Keywords* — 3D-, Spiele-, Multiplattform-Engine, C#, .NET, JavaScript, WebGL**

***Links* — `http://www.fusee3d.org/`**

I. Einleitung

Das Unternehmen Incentive Software entwickelte 1987 mit „Freescape“ eine der ersten 3D-Engines für Videospiele. Die Möglichkeiten dieser Engine waren zwar noch stark beschränkt, Freescape war aber aufgrund einer Vielzahl verschiedener Primitive, einer ersten Scripting-Language und der Unterstützung mehrerer Plattformen ein wichtiger Schritt in der Entwicklung von Grafik-Engines. [Fahs 2008]

Durch stetig steigende Rechenleistung konnte die Grafik in Videospielen und 3D-Anwendungen bis heute immer weiter verbessert und die Engines um Funktionen wie Sound und Physik erweitert werden. Drei der bekanntesten 3D-Gaming-Engines, die „Unreal Engine 3“ (Epic Games), die „CryEngine 3“ (Crytek) und die „Source Engine“ (Valve), sind dabei mehr oder weniger multiplattformfähig. Eine Portierung von Spielen zwischen den Plattformen ist aber immer noch sehr aufwendig.

Eine weitere 3D-Engine ist „Unity“[1] (Unity Technologies), die einen umfangreichen Editor zur Entwicklung eigener Spiele und 3D-Grafik-Anwendungen für eine Vielzahl von Plattformen (neben den gängigen Betriebssystemen auch Spielekonsolen und mobile Systeme) bietet. Von den zuvor genannten Engines ist Unity die Einzige, die dem Entwickler die Möglichkeit bietet, seine Anwendungen auch für den Browser zu exportieren. Allerdings wird zur Darstellung der Inhalte im Browser das Unity Web Plugin benötigt. Unterstützung für HTML5 ist auch in der neusten Version 4 vom November 2012 nicht verfügbar. [Unity3D 2012]

Da HTML5 in naher Zukunft immer stärkeren Einfluss auf die Inhalte im Web nehmen wird und das W3C die Spezifikationen zu HTML5 Ende [2012] fertiggestellt hat, wäre es wünschenswert eine 3D-Grafik-Engine einsetzen zu können, mit der ohne große Vorkenntnisse Anwendungen für unterschiedliche Plattformen entwickelt werden können und die diese Inhalte auf Basis von WebGl im Browser darstellen kann. Hier geht Fusee den entscheidenden Schritt.

II. Fusee

Abbildung 1: *Das Logo von Fusee*

Das Projekt „Furtwangen University Simulation and Entertainment Engine“ wird seit 2012 im Rahmen des sogenannten Projektstudiums von Studenten des 4. und 5. Semesters an der Fakultät Digitale Medien der Hochschule Furtwangen University über einen Zeitraum von zwei Semestern bearbeitet und weiterentwickelt. Studenten anderer Semester entwickeln Plugins und Tests für Fusee oder verwenden Fusee bereits als Engine in ihren Projekten.

Fusee wird in C# und JavaScript entwickelt. Bei C# handelt es sich, auch für unerfahrene Programmierer, um eine relativ einfach zu erlernende Programmiersprache. Hierbei sind die Entwickler nicht an Microsoft Windows (.NET) gebunden, da mit Monodevelop eine von Fusee unterstützte Entwicklungsumgebung für Linux existiert. Die Engine ist von Grund auf modular aufgebaut, sodass es für Nutzer von Fusee unerheblich ist, ob sie OpenGL, WebGL oder auch zukünftig DirectX verwendet, oder welche Physikengine bevorzugt wird. Als offene Plattform wird es ohne Weiteres möglich sein, eigene Module zu entwickeln und beizutragen.

Fusee stellt eine Reihe von Beispielanwendungen zur Verfügung. Entwickler können basierend auf diesen Beispielen oder auf Basis der zur Verfügung gestellten Templates ihre eigenen 3D-Spiele und -Anwendungen entwickeln und mittels Dropdown-Menü in Visual Studio die Ausgabe auf eine .NET-Framework-Applikation (lauffähig auf allen gängigen Betriebssystem) beschränken oder zusätzlich die Ordnerstruktur und die entsprechenden Dateien für eine Web-Anwendung (lauffähig in Google Chrome und Mozilla Firefox) erzeugen lassen. Diese ist ohne weitere Serverkonfiguration aufrufbar und kann über die beigelegten Konfigurationsdateien auch im Nachhinein jederzeit angepasst werden.

Die Portierung der eigenen Anwendung auf Apple iOS oder Google Android erfordert lediglich Anpassungen, die sich aus den unterschiedlichen Bedienkonzepten ergeben. HTML5 wird zwar von den meisten mobilen Browser unterstützt, sodass die Web-Anwendung bereits lauffähig wäre, doch ist die Stabilität und Performance auf mobilen Geräten eingeschränkt. Hier besteht deshalb auch die Möglichkeit mit Hilfe von Cross-Compilern der Firma Xamarin[2] native Apps für Smartphones zu exportieren. Grundlage ist auch hier der mit Fusee in .NET entwickelte Sourcecode.

*Team: `http://fusee3d.org/team/`
[1]Unity: `http://unity3d.com/`
[2]Mono for Android (`http://android.xamarin.com/`) und MonoTouch (`http://ios.xamarin.com/`)

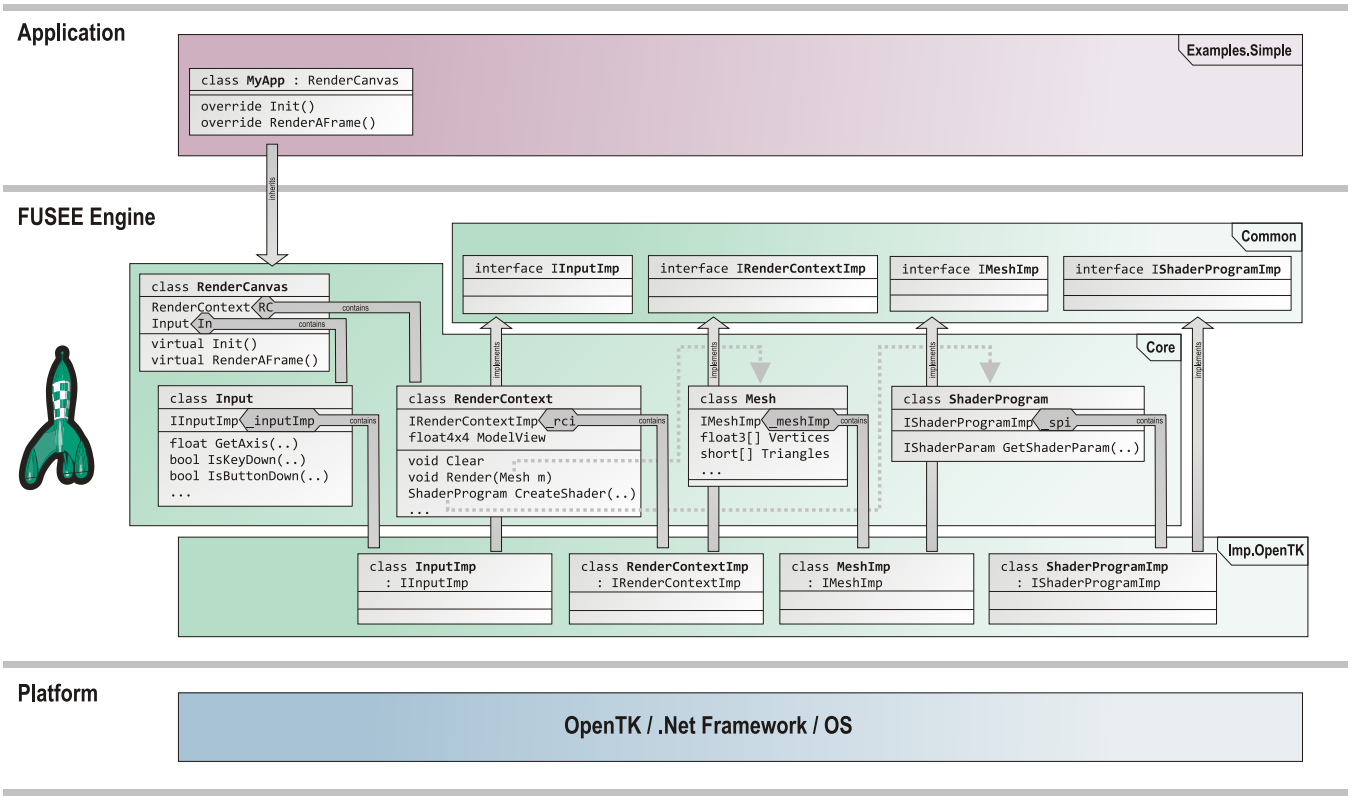

Abbildung 2: *Die Architektur von Fusee*

III. Technische Umsetzung

Fusee ist, wie in Abbildung 2 dargestellt, modular aufgebaut. Jedes interne oder externe Modul (quelloffene C#-Projekte oder .dll-Dateien) wird über eine Abstraktionsschicht (hier, Abbildung 2, Imp.OpenTK) eingebunden, die im Grunde eine interne API-Schnittstelle darstellt. Fusee bietet an diesem Punkt standardisierte Funktionen an, um die jeweiligen Module zu nutzen. Diese Abstraktionsschicht ist der einzige Teil der Codebasis, der für die unterschiedlichen Grafikplattformen separat entwickelt und gepflegt wird.

Eigene Anwendungen (oberste Ebene der Abbildung 2) leiten sich von der Klasse `RenderCanvas` ab und müssen die beiden Methoden `Init()` und `RenderAFrame()` implementieren. Erstere ist für die Initialisierung bei Programmstart zuständig, letztere wird einmal pro Frame ausgeführt. Die zu rendernden Objekte (Meshes) werden über eine Instanz der Klasse `RenderContext` gesteuert. Dazu gehören auch die Manipulation der Kamera (`ModelView`) und das Setzen der gewünschten Shader. In Kapitel V ist hierzu ein kurzer Beispielcode abgebildet. Die Verarbeitung von Eingaben über die Tastatur oder Maus wird über eine Instanz der Klasse `Input` möglich. Abhängigkeiten zwischen Objekten, Materialien und Hierarchien können über das implementierte Szenenmanagement verwaltet werden.

Zur Darstellung verschiedener Materialien und Lichtquellen stehen bereits mehrere Shader zur Verfügung, wobei es dem Entwickler allerdings auch frei steht, selbst definierte Vertex- und Pixelshader zu verwenden. Hier kann jedoch vorerst nur der Befehlssatz der niedrigsten einzusetzenden Version der OpenGL Shading Language verwendet werden.

IV. Übersetzung in JavaScript

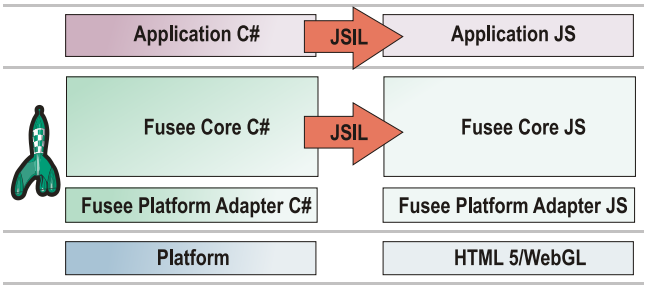

Abbildung 3: *Übersetzung mit JSIL*

Zur Übersetzung des C#-Bytecodes in JavaScript wird der Open-Source-Cross-Compiler „JSIL“[3] von Kevin Gadd verwendet. JSIL ist stark in Richtung Microsoft XNA ausgelegt, wodurch dieser Compiler bereits viele für eine 3D-Engine wichtige Funktionen implementiert hat und die mit Fusee erstellen Anwendungen daher ohne großen Erweiterungen in JavaScript übersetzen kann.

Übersetzt werden, wie in Abbildung 3 zu sehen, die Applikation und der Kern von Fusee. Lediglich der JavaScript-Anteil, der das Interface zu WebGL implementiert, wird separat entwickelt. JSIL verwendet zur effizienten Verarbeitung aller Assets der Anwendung eine Manifest-Datei. Diese und eine grundlegende HTML-Datei werden über ein weiteres Fusee-Tool automatisch generiert, sodass die Anwendung sofort mit Google Chrome, Mozilla Firefox und Opera gestartet werden kann.

[3] `http://www.jsil.org/`

```
1  // [...] - RenderInit():
2  Mesh Teapot = MeshReader.LoadMesh(@"Assets/Teapot.obj");
3
4  // [...] - RenderOnFrame():
5  RC.SetShaderParam(VColorParam, new float4(0.8f, 0.5f, 0, 1));
6
7  float4x4 mtxRot = float4x4.CreateRotationY(_angleH) * float4x4.CreateRotationX(_angleV);
8  float4x4 mtxCam = float4x4.LookAt(0, 200, 400, 0, 50, 0, 0, 1, 0);
9  RC.ModelView = mtxRot * float4x4.CreateTranslation(100, 0, 0) * mtxCam;
10
11 RC.Render(Teapot);
12
13 Present();
```

Quellcode 1: *Beispielcode*

V. Beispielcode

Zum Importieren und Darstellen von 3D-Objekten sind mit Fusee nur wenige Schritte nötig. Im Folgenden soll ein kurzer Überblick über den in Quellcode 1 dargestellten Beispielquelltext gegeben werden. Die allgemeine Struktur, inklusive der zu importierenden Klassen, kann dem bei Fusee beigelegten Template entnommen werden.

Zeile 2: Alle 3D-Objekte sind vom Datentyp `Mesh`. Die Klasse `MeshReader` dient zum Laden und Verarbeiten dieser Objekte. Die hier angegebene .obj-Datei ist ein in Cinema4D erstelltes und im Wavefront-Format exportiertes Objekt. Durch die Verarbeitungsmöglichkeit dieses quelloffenen Formates ergibt sich eine vereinfachte Migration von bereits bestehenden Projekten. In der Regel werden alle Objekte im Konstruktor einer Klasse oder in der `Init`-Funktion zu Beginn der Applikation geladen.

Zeile 5: Je nach eingesetztem Shader ist es möglich, verschiedene Eigenschaften der Meshes, zum Beispiel die Farbe oder die Textur, vor dem Rendern anzupassen.

Zeile 7-9: Translationen und Rotationen werden über die Klasse `float4x4` berechnet. Rückgabe ist jeweils eine Transformationsmatrix, die über Matrixmultiplikationen mit weiteren Matrizen verknüpft werden kann. Neben diesem klassischen Weg der Transformation über Eulerwinkel sind auch Berechnungen mit Hilfe von Quaternionen möglich. `LookAt` berechnet über die Koordinaten dreier Parameter (Position, Ziel, Drehung) die Ausrichtung der Kamera. Die Berechnungen werden schließlich dem `ModelView` des `RenderContexts` zugewiesen und bewirken damit eine relative Verschiebung des zu rendernden Objektes.

Zeile 11: Die Funktion `Render` mit einem Mesh als Parameter rendert dieses Mesh und verwendet dabei die Funktionen der verfügbaren Grafikbibliothek (beispielsweise OpenGL oder WebGL).

Zeile 13: Um ein Flackern bei der Darstellung zu vermeiden, ist es in modernen Engines üblich, zwischen Front- und Backbuffer zu unterscheiden, wobei alle Rendervorgänge im Backbuffer durchgeführt und schließlich über den Frontbuffer dargestellt werden. Daher werden über die Funktion `Present` als letzter Schritt in `RenderOnFrame` die Buffer vertauscht.

VI. Fazit und Ausblick

Fusee hat das Potenzial eine umfangreiche 3D-Multiplattform-Engine zu werden. Dazu gehören die Entwicklung und Implementierung von Sound-, Physik- und Netzwerkfunktionen, wobei auch weiterhin viel Wert auf die Modularisierung gelegt wird, um den Anforderungen unterschiedlicher Plattformen gerecht zu werden. Hier wäre es denkbar, auf bereits vorhandene, quelloffene C#-Bibliotheken anderer Entwickler zurückzugreifen. Mit Hilfe von JSIL wäre es so sogar möglich, diese Bibliotheken nach JavaScript zu übersetzen und so auch in der Web-Applikation einzusetzen.

Zur Veröffentlichung des Quelltexts greift Fusee auf die Versionsverwaltung Git[4] zurück. Dadurch ist es jedem interessierten Entwickler möglich, die Entwicklung von Fusee mitzuverfolgen, die jeweils aktuellste Version als Sourcecode herunterzuladen oder sogar mitzuarbeiten. Dadurch ist die Laufzeit des Projektes nicht an den einjährigen Rahmen des Projektstudiums gebunden. Darüber hinaus kann Fusee als Grundlage für Forschungs- und Abschlussarbeiten in allen Bereichen der Informatik dienen.

VII. Danksagung

Wir danken der Fakultät Digitale Medien der Hochschule Furtwangen University für die Bereitstellung der Ressourcen und Kevin Gadd für die Entwicklung des JSIL-Compilers.

Literatur

Fahs, T., 2008 Exploring the Freescape – How Incentive Software blazed a trail into 3D
`http://uk.ign.com/articles/2008/10/22/exploring-the-freescape/`
Abgerufen am 11.02.2013.

Unity3D, 2012 Unleash your game with effortless deployment to 10 global platforms
`http://unity3d.com/unity/multiplatform/`
Abgerufen am 11.02.2013.

World Wide Web Consortium, 2012 HTML5 Definition Complete, W3C Moves to Interoperability Testing and Performance
`http://www.w3.org/2012/12/html5-cr.html.en`
Abgerufen am 11.02.2013.

[4] Projekt auf GitHub: `https://github.com/FUSEEProjectTeam/`