Furtwangen University Simulation and Entertainment Engine
Verwendung der Oculus Rift zur Darstellung von 3D-Inhalten

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

***Zusammenfassung* — Dieses Paper befasst sich mit der Verwendung der Oculus Rift als Möglichkeit der Darstellung von 3D-Inhalten und den dafür notwendigen Voraussetzungen sowie mit der Integration in eigene Engines am Beispiel von Fusee.**

***Keywords* — Fusee, 3D-Engine, C#, .NET, Oculus Rift**

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

I. Einleitung

Die Oculus Rift ist eine *VR-Brille*, die aktuell von Oculus VR entwickelt wird. Die Oculus Rift besteht aus einem 7 Zoll großen Display mit einer Auflösung von 1280x800 Pixel (640x800 Pixel pro Auge) und diversen Sensoren, um die Bewegungen des Kopfes tracken zu können. Die erste Auslieferung in Form einer Developer-Version erfolgte im 1. und 2. Quartal 2013, nachdem die Finanzierung über die Kickstarter-Plattform etwa 2,5 Millionen Dollar einspielte.

Einige Engines, unter anderem Unity und die von Valve entwickelte Source Engine, unterstützten bereits die Oculus Rift als Ausgabegerät. In der Open-Source-Engine Fusee war die Ausgabe von Inhalten in Stereo-3D bisher schwierig, da es keine dafür geeigneten Klassen gab. Daher ist dieses Paper entstanden, das sich mit der Oculus Rift als eine neue Möglichkeit der Darstellung von Stereo-3D-Inhalten in Fusee beschäftigt.

II. Darstellung auf dem Display

Zwischen Display der Oculus Rift und den Augen befinden sich zwei Linsen (eine pro Auge), die zur Vergrößerung des Bildes verwendet werden. Dadurch ergibt sich aber auch eine unerwünschte Verzerrung des Bildes, weshalb schon das gerenderte Bild eine (gegenteilige) tonnenförmige Wölbung aufweisen muss, die durch die Linsen wieder entzerrt wird. Dabei wird ein großer Teil des Displays allerdings nicht verwendet (fast 50%, siehe auch Abbildung 1 auf Seite 2).

Diese Verzerrung des Bildes kann mit Hilfe entsprechender Shader erreicht werden. Einer der ersten inoffiziellen Shader war der von Luke Groeninger entwickelte und auf GitHub veröffentlichte glslRiftDistort-Shader [Luke Groeninger 2013] für OpenGL, der neben den standardmäßigen Vertex- und Pixelshadern auch einen Geometrieshader zur Berechnung diverser Variablen benötigt. Damit setzt glslRiftDistort allerdings mindestens OpenGL 3.2 und GLSL 1.50 voraus.

Groeninger merkt an, dass auf den Geometrieshader verzichtet werden kann, sofern die Berechnung der entsprechenden Variablen außerhalb von OpenGL vorgenommen wird (siehe Quellcode 3 im Anhang). Da die Werte teilweise nur von Konstanten abhängig sind, können auch direkt feste Werte gesetzt werden. Die Werte, die Fusee verwendet, lassen sich Quelltext 1 auf Seite 2 entnehmen. Sie basieren auf auf Angaben des offiziellen Oculus-Rift-SDKs und auf eigener Erfahrung durch die Erstellung dieses Papers.

Ein kurzer Überblick über die Bedeutung der Variablen: `LensCenter` und `ScreenCenter` geben abhängig vom Auge die Mitte des Displays an und können daher als konstante Faktoren der Auflösung angegeben werden. `Scale` und `ScaleIn` entsprechen einer Art Vergrößerung und können ebenfalls fix gesetzt werden. `HmdWarpParam` ist für die Form der Verzerrung verantwortlich, wobei auch hier feste Werte vergeben werden können.

III. Head-Tracking

Eine weitere Besonderheit der Oculus Rift ist das fast verzögerungsfreie Head-Tracking. Mit Hilfe von 3-Achsen-Gyrometern und Beschleunigungssensoren sowie einem Magnetometer zur korrekten Ausrichtung werden die Bewegungen (in der Developer-Version lediglich Rotationen, keine Translationen) des Kopfes registriert und über den Treiber der Oculus Rift verarbeitet.

Daraus ergibt sich allerdings auch das Problem für die Nutzung des Head-Trackings in Fusee. Die Treiber, Teil des Oculus-Rift-SDKs, sind (bisher) nicht für .NET verfügbar. Eine Lösung dafür wären die beiden C#-Wrapper VRapper[1] und RiftDotNet[2]. Während das offizielle Oculus-SDK allerdings schon die Versionsnummer 0.2.4 trägt, beziehen sich beide Wrapper noch auf die Version 0.1.5 und sind damit nicht mehr auf dem neusten Stand. Es gibt bereits Ansätze, minimale C-Libraries als Alternative zum Oculus-Rift-SDK zu veröffentlichen[3], aus denen sich wiederum neue Wrapper für C#.NET entwickeln könnten. Aber auch hier scheint sich in letzter Zeit kaum noch etwas zu tun.

Alternativ wäre ein Blick auf HID-Programmierung interessant. Die eben angesprochenen C-Libraries basieren teilweise auf HIDAPI, einer plattformunabhängigen C-Library, die die Nutzung von USB-Interfaces vereinfacht. Für C#.NET gibt es im Bereich der HID-Programmierung wiederum wenig aktuelle Libraries. Die Simple HID Library[4] (Stand: März 2012) wäre eventuell eine mögliche Grundlage. Da dies allerdings noch nicht weitergehend getestet wurde, ist das Head-Tracking kein Teil dieses Papers.

*Team: `http://fusee3d.org/team/`
[1]VRapper: `https://github.com/Entroper/VRapper`
[2]RiftDotNet: `https://github.com/SiS-Shadowman/RiftDotNet`
[3]libovr_nsb: `https://github.com/ultranbrown/libovr_nsb`
[4]Simple HID Library: `http://simplehidlibrary.codeplex.com/`

```
uniform sampler2D vTexture;
varying vec2 vUV;

uniform vec2 LensCenter;     // linkes Auge: (0.3125, 0.5) – rechtes Auge: (0.6875, 0.5)
uniform vec2 ScreenCenter;   // linkes Auge: (0.25, 0.5)   – rechtes Auge: (0.75, 0.5)
uniform vec2 Scale;          // (0.1469278, 0.2350845)
uniform vec2 ScaleIn;        // (4, 2.5)
uniform vec4 HmdWarpParam;   // (1.0, 0.22, 0.24, 0.0)

vec2 HmdWarp(vec2 texIn) {
    vec2 theta = (texIn – LensCenter) * ScaleIn;
    float rSq = theta.x * theta.x + theta.y * theta.y;
    vec2 theta1 = theta * (HmdWarpParam.x + HmdWarpParam.y * rSq + HmdWarpParam.z * rSq * rSq +
        HmdWarpParam.w * rSq * rSq * rSq);
    return LensCenter + Scale * theta1;
}

void main() {
    vec2 tc = HmdWarp(vUV.xy);
    if (any(bvec2(clamp(tc,ScreenCenter–vec2(0.25,0.5), ScreenCenter+vec2(0.25,0.5)) – tc)))
      gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0);
    else
      gl_FragColor = texture(vTexture, tc);
}
```

Quellcode 1: *Ausschnitt aus dem Pixelshader*

IV. Umsetzung in Fusee

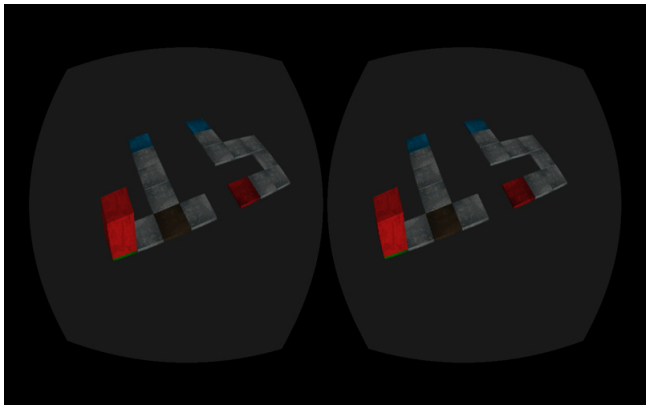

Abbildung 1: *Beispielhafte Ausgabe für die Oculus Rift in Fusee*

Quellcode 1 zeigt den in Fusee verwendeten Pixelshader für die Darstellung der Inhalte auf der Oculus Rift. Die Uniform-Variablen sind in der Regel vom Typ `vec2` (`HmdWarpParam` vom Typ `vec4`) zu deklarieren. Sowohl der Vertexshader als auch die Variablen `vTexture` und `vUV` unterscheiden sich nicht von den bisher in Fusee verwendeten Shadern bzw. Variablen. Die Werte werden den Variablen nach Aktivierung des Shaders zugewiesen.

Im gewohnten Renderprozess in Fusee kann dieser Shader aber so nicht eingesetzt werden, da sonst nicht das Bild als ganzes, sondern jegliche Objekte verzerrt wären. Es musste eine Möglichkeit gefunden werden, den Shader auf das Endergebnis anzuwenden, da Fusee bisher keine Möglichkeit für mehrere Renderdurchgänge hat. Die Lösung lag in der Zwischenspeicherung der Ergebnisse (pro Auge ein Ergebnis). OpenGL kennt dafür die Funktion `GetBufferContent`, die nun auch in Fusee zur Verfügung steht. Sie speichert den Pufferinhalt in eine als Parameter übergebene Textur.

Der komplette Render-Prozess läuft nun folgendermaßen ab: Zunächst wird der Viewport auf die Hälfte des Bildes eingeschränkt. Auf diese Hälfte wird wie gewohnt (durch Verschiebung der Kamera) das Bild für das linke Auge gerendert. Nach dem Zwischenspeichern mit `GetBufferContent` und dem Leeren des Puffers wird der Vorgang für das Bild des rechten Auges wiederholt.

Nun stehen zwei Texturen zur Verfügung, die mit Hilfe des Pixelshaders und den zugewiesenen Werten verzerrt auf zwei planare Objekte gerendert werden können. Die Kamera ist entsprechend nach hinten verschoben, so dass sich letztlich pro Frame die in Abbildung 1 dargestellte Ausgabe ergibt.

V. Die neue Klasse `Stereo3D`

Die Art und Weise wie die Darstellung von Inhalten auf der Oculus Rift in Fusee umgesetzt wurde, bietet sich auch für andere Arten der Darstellung von Stereo-3D-Inhalten an. So kann beispielsweise auch ein Anaglyphenbild (zur Nutzung mit einer Rot-Grün-Brille) dadurch umgesetzt werden, dass zunächst das Bild für das linke Auge und anschließend das Bild für das rechte Auge gerendert und zwischengespeichert wird, bevor dann die eigentliche Berechnung der Darstellung (side-by-side wie bei der Oculus Rift oder rot-grün durch Setzen der Farbkanäle) erfolgt.

Daher wurde im Rahmen dieser Entwicklung auch eine neue Klasse `Stereo3D` in Fusee implementiert. Mit dieser ist es mit geringem Aufwand möglich, verschiedene Arten der 3D-Darstellung umzusetzen. Entsprechende, aber weitaus komplexere Funktionen zur Darstellung der Inhalte als Anaglyphenbild waren bisher nur in einem der Beispielprojekte verfügbar.

Diese Klasse verwaltet neben der Speicherung der Zwischenbilder auch die entsprechenden Shader. Dadurch ist es völlig unerheblich, welche Art von Shadern für die eigentlichen Inhalte verwendet werden. Die Nutzer von Fusee müssen daher die eigenen bzw. die vorgegebenen Shader nicht speziell anpassen, um Stereo-3D zu aktivieren, sondern können dies automatisch von der Engine übernehmen lassen. So bleibt lediglich die Verschiebung der Kamera (über Funktionen wie `LookAt`) noch als manuelle Aufgabe.

```
1  // Initialisierung:
2  _stereo3D = new Stereo3D(RContext, Stereo3DMode.Oculus, 1280, 800);
3
4  // linkes Auge:
5  _stereo3D.Prepare(Stereo3DEye.Left);
6  // [...]
7  _stereo3D.Save();
8
9  // rechtes Auge:
10 _stereo3D.Prepare(Stereo3DEye.Right);
11 // [...]
12 _stereo3D.Save();
13
14 // Ausgabe:
15 _stereo3D.Display();
```

Quellcode 2: *Beispielcode*

VI. Beispielcode

Die Verwendung der Oculus Rift als neue Möglichkeit der Darstellung wurde durch die eben vorgestellte neue Klasse `Stereo3D` für die Nutzer von Fusee stark vereinfacht. Im Folgenden soll ein kurzer Überblick über den in Quellcode 2 dargestellten Beispielquelltext für die Ausgabe des Bildes auf der Oculus Rift unter Verwendung dieser neuen Klasse gegeben werden. Hierfür muss außerdem (sinnvollerweise bereits zur Initialisierungszeit) der Vollbildmodus über die Fusee-Variable `Fullscreen` aktiviert werden.

Zeile 2: Zunächst wird ein neues Objekt der Klasse `Stereo3D` benötigt. Der Konstruktor hierfür nimmt als ersten Parameter eine Referenz auf den `RenderContext` an. Der zweite Parameter definiert die Art der Ausgabe, wobei hier aktuell die beiden Werte `Stereo3DMode.Oculus` und `Stereo3DMode.Anaglyph` zur Verfügung stehen. Die Auflösung (dritter und vierter Parameter) muss mit der Auflösung des Ausgabefensters übereinstimmen. Hier bietet es sich an, die entsprechenden Fusee-Variablen `Width` und `Height` zu nutzen.

Zeile 4-7: Über die Funktion `Prepare` mit Übergabe von `Stereo3DEye.Left` als Parameter kann der Rendervorgang des Inhalts für das linke Auge vorbereitet werden. Danach folgen die gewohnten Funktionen für das Rendern von Objekten in Fusee. Der Abschnitt wird anschließend über die Funktion `Save` abgeschlossen. Hierfür ist kein Parameter mehr nötig.

Zeile 9-12: Der selbe Vorgang wird für den Inhalt für das rechte Auge wiederholt. Welches Bild zuerst gerendert wird ist allerdings nicht ausschlaggebend. Es kann ebenso gut auch zuerst das Bild für das rechte und anschließend das Bild für das linke Auge gerendert werden. Jeder weitere Aufruf von `Prepare` und `Save` überschreibt die bisher gespeicherten Daten.

Zeile 15: Die Funktion `Display` ersetzt das bisherige verwendete `Present` und übernimmt basierend auf den zuvor gespeicherten Zwischenbildern die Berechnung für eine optimale Darstellung in Stereo-3D und setzt die entsprechenden Shader. Im Falle der Oculus Rift ergibt sich so die auf Seite 2 in Abbildung 1 gezeigte Ausgabe.

VII. Fazit und Ausblick

Die Verwendung der Oculus Rift als Display zur Darstellung von 3D-Inhalten konnte erfolgreich mit Fusee umgesetzt werden. Gleichzeitig ergaben sich durch eine verbesserte Organisationsstruktur im Quellcode (unter anderem die neue Klasse `Stereo3D`) neue Möglichkeiten, um weitere Arten der Stereo-3D-Darstellung implementieren zu können.

Dennoch wird das Potential der Stereo-3D-Brille noch nicht voll und ganz ausgenutzt, da — wie bereits in Kapitel 3 erwähnt — das Head-Tracking in Fusee bisher nicht genutzt werden kann. Bis zur Veröffentlichung der Consumer-Version der Oculus Rift (voraussichtlich 2014 [EDGE 2013]) kann sich das Projektteam aber noch umfassend mit der Implementierung der Head-Tracking-Funktion und weiteren Features sowohl im Zusammenhang mit der Oculus Rift (beispielsweise Möglichkeiten zur Kalibrierung) als auch ganz allgemein im Bereich der Stereo-3D-Darstellung (Shutter, 3D-Beamer, etc.) befassen. Interessant wäre hier auch ein Blick auf die neue GamepadAPI[5], um herauszufinden, ob damit die Oculus Rift sogar im Browser benutzt werden könnte.

VIII. Danksagung

Wir danken Herrn Professor Dell'Oro-Friedl von der Fakultät Digitale Medien der Hochschule Furtwangen für die vorübergehende Bereitstellung der Oculus Rift (Developer-Version) zu Testzwecken.

Literatur

EDGE, 2013 Oculus rift aiming for 2014 release, ceo touts potential on 'next gen cellphones'
`http://www.edge-online.com/news/oculus-rift-aiming-for-2014-release-consoles-not-a-focus-but-mobile/`
Abgerufen am 24.09.2013.

Luke Groeninger, 2013 Opengl barrel distortion shader for the oculus rift
`https://github.com/dghost/glslRiftDistort`
Abgerufen am 24.09.2013.

[5] GamepadAPI: https://wiki.mozilla.org/GamepadAPI

A ANHANG

```
// h = 1, w = 0.5
// x = 0 oder 0.5, y = 1

const float K0 = 1.0f;
const float K1 = 0.22f;
const float K2 = 0.24f;
const float K3 = 0.0f;

float aspectRatio = (float)(Width/2.0f) / Height;
const float scaleFactor = 0.5877112f;
float distortionXCenterOffset = (eye == Stereo3DEye.Left) ? 0.25f : -0.25;

float lensCenterLocation = new float2(x + (w + distortionXCenterOffset * 0.5f)*0.5f, y + h*0.5f);
float screenCenterLocation = new float2(x + w*0.5f, y + h*0.5f);

float scale = new float2(w/2.0f) * scaleFactor, (h/2.0f) * scaleFactor * aspectRatio);
float scaleIn = new float2((2.0f/w), (2.0f/h) / aspectRatio);

float hdmWarp = new float4(K0, K1, K2, K3);
```

Quellcode 3: *Berechnung der Shadervariablen*