universität wien

Masterarbeit

Titel der Arbeit

Cloud Gaming – technische Machbarkeit und Auswirkungen von Streaming moderner Computerspiele auf das Benutzererlebnis

Verfasser

Alexander Franiak, BSc

Angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2013

Studienkennzahl: 066 935

Studienrichtung: Master Medieninformatik

Betreuer: Univ.-Prof.Dipl-Ing. Dr. Helmut Hlavacs

Inhaltsverzeichnis

1 Einleitung

Die vorliegende Masterarbeit beschäftigt sich mit dem Thema „Cloud Gaming“, den Möglichkeiten der technischen Umsetzung sowie den Einflüssen der technologiebedingten Einschränkungen auf die erlebte Qualität eines gestreamten Spiels für einen Benutzer.

Zu Beginn wird das Thema „Cloud Gaming“ im Allgemeinen erörtert und das damit verbundene Spielerlebnis beschrieben und evaluiert. Des Weiteren wird das im Zuge der Masterarbeit entstandene Framework sowie dessen technische Umsetzung dokumentiert, kritisch betrachtet und analysiert.

Ziel ist es mit Hilfe der Software als Hilfsmittel die derzeitige technische Realisierbarkeit und Qualitätsauswirkungen auf den Benutzer zu erheben und auszuwerten.

Um ein State-Of-The-Art-Computerspiel rendern zu können benötigt ein PC/Laptop heutzutage nahezu immer eine dedizierte Spielegrafikkarte. Diese kann je nach Qualität, sehr kostspielig ausfallen. Für Gelegenheitsspieler, die lediglich ein Arbeits-Netbook besitzen und bezüglich Preis, Gewicht und Akkukapazität keine Kompromisse eingehen wollen, würde sich Cloud Gaming als Lösung anbieten. So könnten Internetanbieter beispielsweise Rechnerkapazitäten und Spiele für das Streamen zu Kundenrechnern anbieten. In den letzten Jahren nahm die Weiterentwicklung von Computerspielen und den PC-Systemen, auf denen sie laufen rapide zu. Dies führte dazu, dass die kostspielige Hardware (im speziellen die Grafikkarte) in immer kürzeren Abständen erneuert werden musste um die benötigte Rechenleistung zu erbringen und neue Algorithmen ausführen zu können.

Die gleichzeitige Weiterentwicklung des Internets und der Bandbreite mit der Daten übertragen werden können zeigt nun eine völlig neue Möglichkeit, Computerspiele erleben zu können.

Unter dem Begriff „Cloud Gaming“ versteht man, dass ein Computerspiel nicht mehr länger von einem PC oder einer Spielekonsole, sondern von einem physisch entfernten Server berechnet wird. Die Bildschirmbewegung wird bildweise von einem Codec

enkodiert und über die Breitbandleitung zum Client-Computer gesendet. Dieser nimmt lediglich die Dekodierung vor um anschließend den Spielinhalt auf dem Monitor des Clients anzuzeigen.

Die Clientmaschine kann hierbei mit sehr kostengünstiger Hardware ausgestattet sein, da der Dekodierungsprozess im Vergleich zur Berechnung sowie Enkodierung eines Spiels keine hohe Hardwareanforderung stellt, ein Netbook aber auch Tablet oder sogar ein Smartphone ist völlig ausreichend. Der Nachteil ist hierbei die Bandbreite der verfügbaren Internetverbindung. Cloud Gaming Technologie ist noch in den Anfängen und sowohl Bildqualität als auch Geschwindigkeit lassen noch sehr zu wünschen übrig. Die derzeit verfügbaren Clients bieten nahezu keinerlei Kontrolle bezüglich der Bildqualität oder Übertragungseinstellungen.

Der Ist-Zustand soll erhoben und mögliche Lösungsansätze gefunden werden. Dazu wird ein Framework erstellt, das eine Cloud Gaming Architektur mit einem beliebigen DirectX-Spiel simuliert, indem das Bild des Spiels abgefangen und komprimiert wird. Die so erstellten Daten werden über das Netzwerk zum Benutzer übermittelt, so dass dieser dem Spielverlauf folgen kann. Jegliche Eingaben des Benutzers werden wiederum über das Netzwerk an den Server übermittelt, wo sie für das Spiel als reale Benutzereingabe simuliert werden. Es soll eine Benutzeroberfläche erstellt werden mit der Qualitätsparameter des Streamings wie z.B. Bitrate und Bilder pro Sekunde für den jeweiligen Anwendungsfall und Spieltyp angepasst und evaluiert werden können.

Das angefertigte Framework wird benutzt um den Einfluss verschiedener Parameter wie z.B. Framerate und Enkodierungs-Bitrate auf den Benutzer festzustellen bzw. zu messen.

1.1 Anforderungen and eine Cloud Gaming-Applikation

Das Netzwerk sollte sehr geringe Dekodierungs- und Übertragungsverzögerungen sowie einen geringen Jitter und Paketverlust aufweisen. Provider solcher Dienste müssen ihre Datacenter an strategisch günstigen Netzwerkknotenpunkten in physischer Nähe des Benutzers aufbauen.

Aus diesem Grund ergeben sich auch völlig neue Anforderungen an moderne Videocodecs, deren bisherige Spezialisierung auf Inhalten lag, die von einer digitalen Filmkamera erstellt wurden. Der zu kodierende (und zu dekodierende) Inhalt ist hierbei von einer Maschine berechnet, was neue Optimierungsmöglichkeiten, aber auch Herausforderungen darstellt. Beispielsweise ist das Abgreifen (oder auch Hijacken) der Bilddaten aus einem Computerspiel bei Zugriff auf den Programmcode des Spiels (OpenSource oder selbst programmiert) trivialer als ein Spiel aus dem Handel eines großen Publishers zu streamen. Dies kann nur durch das Eindringen auf der Ebene des 3D-Renderers (DirectX oder OpenGL) und einer vom ursprünglichen Hersteller nicht vorgesehenen Manipulation der Ausführung des Spiels erreicht werden.

Dem Nachteil der fehlenden Pufferung durch Echtzeitsteuerung und –veränderung des Spiels stehen jedoch auch neue Optimierungsmöglichkeiten gegenüber. Verschiedene Parameter eines Spiels sowie das oft repetitive Verhalten des Benutzers und der Spielmechanik (sich meist wiederholende Abläufe, die das Spiel dem Spieler zur Verfügung stellt um sein Ziel zu erreichen, die die Dynamik und den Ablauf des Spiels repräsentieren) bieten Raum für Verringerung der Datenmenge durch geschicktes Kodieren der für den Spielablauf wichtigeren Elemente des Bildes sowie dem Versuch, gewisse Abläufe vorauszusehen.

1.2 Problemstellung und Lösungsansatz dieser Arbeit

Im Rahmen dieser Masterarbeit soll eine Cloud Gaming-Umgebung simuliert und messbare Parameter gefunden werden, die ausschlaggebend für ein solches Spielergebnis sind. Dazu wird eine modulare, leicht adaptierbare und effiziente Softwarelösung einer Cloud Gaming-Umgebung vorgestellt. Diese verwendet einen DLL-Wrapper um ein Windows-DirectX9 Spiel beliebigen Genres, Herstellerfirma oder Spiel/Render-Engine (z.B. Source, Unreal Engine 3) zu streamen.

Nach eingehender Beleuchtung der Grundlagen und Technologien, die für eine solche Implementierung notwendig sind, werden die tatsächlich verwendeten Technologiebausteine vorgestellt und deren Auswahl begründet.

Es wird detailliert beschrieben wie dieser Wrapper funktioniert, in das Spiel eindringt und dort die Bilddaten abfängt, kodiert und zum Spieler auf der Clientseite schickt. Außerdem wird beschrieben wie die vom Spieler durchgeführten Eingaben vom Client abgefangen, über die Leitung gesendet und von der auf dem Host laufenden Software in das Spiel injiziert wird. Diese Funktionsweise ist keineswegs trivial und erfordert eine tiefere Einsicht in Computerspiel, 3D-Renderer, Netzwerkschicht sowie in den verwendeten Videocodec und die Windows API für Input, welche die Programmierschnittstelle von Windows ist, die es ermöglicht, Benutzereingaben zu generieren und zu simulieren).

Mit Hilfe dieser Softwarelösung werden subjektive Benutzerexperimente durchgeführt, die mit Hilfe des kürzlich von Google offen gestellten VP8-Codec erreicht wurden. Dieser unter der Googles WebM-Spezifikation stehende Codec ist nun unter Open Source-Lizenz und vermeidet die Einschränkungen und Patentbeschränkungen des bisher hauptsächlich eingesetzten und weit verbreiteten H.264-Codecs [1].

Ein Netzwerkexperiment soll die Praktikabilität von Cloud Gaming mit dem derzeitigen Stand der Leitungstechnik und unter Berücksichtigung einer reinen Softwarelösung kritisch evaluieren.

Zum Abschluss erfolgt ein Blick in die Zukunft des Cloud Gaming sowohl in Bezug auf periphere Faktoren wie Netzwerktechnik als auch die der eingebauten Softwarebausteine wie z.B. der derzeit verfügbaren Kodierungstechnologie und direkte Möglichkeiten und Ansätze zur Weiterentwicklung der bestehenden Testumgebung einer Cloud Gaming-Lösung.

1.3 Vorhandene Cloud Gaming-Lösungen

Die Idee von Cloud Gaming wurde/wird von mehreren kommerziellen Organisationen aufgegriffen, wobei derzeit keine Open Source Lösung zu existieren scheint und keine der Lösungen bisher vollends ausgereift bzw. auf dem Markt etabliert ist. Ein Kritikpunkt an dezentralisiertem Spielehosting ist, dass theoretisch die Möglichkeit bestünde, detaillierte Persönlichkeitsprofile der Spieler anzulegen, da alle spielerischen Interaktionen inklusive beispielsweise Community-Interaktionen wie Chats über die Onlive-Server laufen, was ein Risiko im Sinne des Datenschutzes darstellt.

1.3.1 Onlive[1]

Nach 7 Jahren Entwicklung hinter verschlossenen Türen wurde Onlive als Cloud Gaming- und Game-On-Gemand-Service unter der Leitung des CEO Steve Perlman 2011 in Europa gestartet. Ein Desktop-Client stellte hierbei das Fenster in das Onlive-Spielangebot dar (selbst die GUI-Menüs wurden bereits auf den Onlive-Servern dargestellt und gestreamt). Partnerschaften mit Namhaften Publishern wie Electronic Arts, Square Enix, 2K Games, Rockstar Games uvm. versichern ein breites Angebot. Leistungsreiche Serverfarmen rendern die Spieldaten und streamen die Bildinhalte zum Client. Das Verkaufsmodell sieht in seiner uneingeschränkten Variante den Verkaufspreis für ein Spiel gleichwertig wie im Handel. Dafür ist es quasi uneingeschränkt nutzbar (mindestens 3 Jahre Support sind garantiert). Es existiert außerdem die App „Onlive Desktop“ für den iTunes App Store sowie für den Google Play Store, mit der man von überall auf seinen PC zugreifen und arbeiten kann. Dies ähnelt stark Remotedesktop-Software, wird jedoch durch die Streamingtechnologie verbessert. Außerdem wurden Set-Top-Boxen und Gaming-Eingabegeräte angeboten.

Vor kurzem jedoch musste Onlive Insolvenz anmelden, da die monatlichen Kosten von rund 5 Millionen Dollar nicht amortisiert werden konnten. Im Zuge der Insolvenz wurde Onlive an ein neues Unternehmen verkauft wobei nur ein Teil des ursprünglichen Mitarbeiterstabs wieder eingestellt wurde, um den Betrieb aufrecht zu erhalten [2] [3].

[1] http://www.onlive.com/

1.3.2 Gaikai[2]

Gaikai (japanisch für „offener Ozean“) spezialisert seinen Cloud Gaming-Service auf die Einbettung in Websites sowie Social Networking-Sites wie z.B. Facebook. Zum Anzeigen wird jedoch ein Flash- oder Java-Plugin benötigt. Eine Einbindung in das Wikipad[3] sowie Googles Native Client (NaCl)[4] wurde bekanntgegeben. Ursprünglich enthielt Gaikai außerdem Werbe-Kauf-Angebote am Ende einer Spielsitzung. Derzeit ist Gaikai nicht in Betrieb, da es für 380 Millionen Dollar von Sony aufgekauft wurde, um es als Streaming-Service für die Play Station 4 einzusetzen [4] [5] [6] [7].

1.3.3 StreamMyGame[5]

Diese reine Softwarelösung bietet einen Client zum kostenlosen Download an. Die Gratisversion des Dienstes, der MPEG-4 zu verwenden scheint ist hierbei auf Auflösungen bis 1024x768 Pixel beschränkt. Die Premium-Version für 10 Dollar pro Jahr ermöglicht bis zu 1280x720 Pixel, die „Unlimited“ Version für 20 Dollar bis zu 3200x2400. StreamMyGame bietet außerdem eine Community-Funktion mit Chat und das Aufnehmen von Spielen und Teilen der Videos auf Youtube [8] [9] [10].

[2] http://www.gaikai.com/
[3] http://www.wikipad.com/
[4] http://code.google.com/p/nativeclient/
[5] http://www.streammygame.com/smg/index.php

1.3.4 Agawi, Cloudunion, CyberCloud networks, Playcast, Ubitus

Agawi, Cloudunion, CyberCloud networks, Playcast und Ubitus sind weitere CloudGaming-Anbieter, von denen jedoch außer des Versprechens von CloudGaming und teilweise Partnerschaften mit NVIDIA Grid[6] weder technische Details noch aktuell funktionierende Clients zur Verfügung stehen. Dies ist jedoch auch bei Onlive und Gaikai derzeit der Fall.

Die Technologie ist eindeutig in einem sehr frühen Stadium, da es derzeit mehr Promotion als tatsächliche Lösungen zu geben scheint. Als einzige Ausnahme könnte StreamMyGame genannt werden, obwohl dieser Anbieter seine Versprechen bezüglich Latenzzeiten derzeit noch nicht ausnahmslos halten kann sowie den Anschein erweckt länger nicht upgedatet worden zu sein.

[6] http://www.nvidia.de/object/nvidia-grid-cloud-gaming-de.html

2 Grundlagen der Videokodierung

Zum Verständnis der Materie des Cloud Gaming gilt es verschiedene große Gebiete der Informatik zu beleuchten. Einer der Hauptbausteine ist hierbei das En- und Dekodieren von Einzelbildern bzw. digitalen Videosignalen. Im Detail auf die Geschichte der verschiedenen Video- und Bildformate sowie die unterschiedlichen Kompressionstechniken einzugehen ist jedoch nicht Bestandteil dieser Arbeit. Diese behandelt lediglich die Grundprinzipien der Kompression und Kodierung sowie wichtige Erkennungsmechanismen, die für das Verständnis der behandelten Cloud Gaming-Thematik relevant sind. Diese sind unter anderem Darstellungsformen von Bildern, Bild- und Videokodierung, Latenzen und Bewegungssuche und –kompensation. Hierbei liegt der Fokus auf Verfahren und Techniken die vom tatsächlich verwendeten Codec der Cloud Gaming-Lösung eingesetzt werden.

2.1 Visuelle Wahrnehmung

Ab einer Einzelbildrate von 14-16 Einzelbildern pro Sekunde (FPS) kann das menschliche Auge einen Bildstrom als kontinuierliches Video erkennen. Um hingegen ein Video als „flüssige“ Darstellung bezeichnen zu können, erfordert es 20 FPS oder mehr. Der derzeitige Standard der Filmindustrie beträgt 24 FPS, wobei sich auch hier bereits ein Aufwärtstrend erkennen lässt. James Cameron plant z.B. den Kinofilm „Avatar 2“ mit 48 oder gar 60 Bildern pro Sekunde zu produzieren. Unter 50 FPS nimmt das menschliche Auge ein Flimmern war, als ob das Licht im Bild ein- und ausgeschaltet werden würde. Diesem Effekt kann mit Interlacing entgegengewirkt werden. Mittels Interlacing wie es das analoge TV-Übertragunsformat PAL (Phase-Alternation-Line) verwendet kann die Bildwechselrate verdoppelt werden, indem ein vollständiges Bild aus zwei Halbbildern aufgebaut wird, wobei zuerst die ungeraden und anschließend die geraden Zeilen eines Bildes dargestellt werden und somit die 24 Bilder pro Sekunde zu 48 Hz werden.

So wird ein Flackern, das bei 24 Hz auftreten würde vermieden und die Übertragungsbandbreite gleichzeitig verringert. Ein Hinweis zu dem verwendeten Interlacing-Modus kann aus z.B. der Angabe „1080i“ abgeleitet werden, wobei die Zahl für die horizontale Auflösung in Pixeln und das „i“ für „interlaced“ steht. Die verwendete Auflösung von PAL beträgt 720x576i mit 50 Halbbildern/s = 25 FPS (Bilder pro Sekunde) bei 24 Bit Farbtiefe (dazu später mehr im Kapitel 2.3) während z.B. NTSC (National Television Systems Comittee) eine Auflösung von 720x480i mit 60x1000/1001 HB/s und 29,97 FPS wiederum mit 24 Bit Farbtiefe bietet.

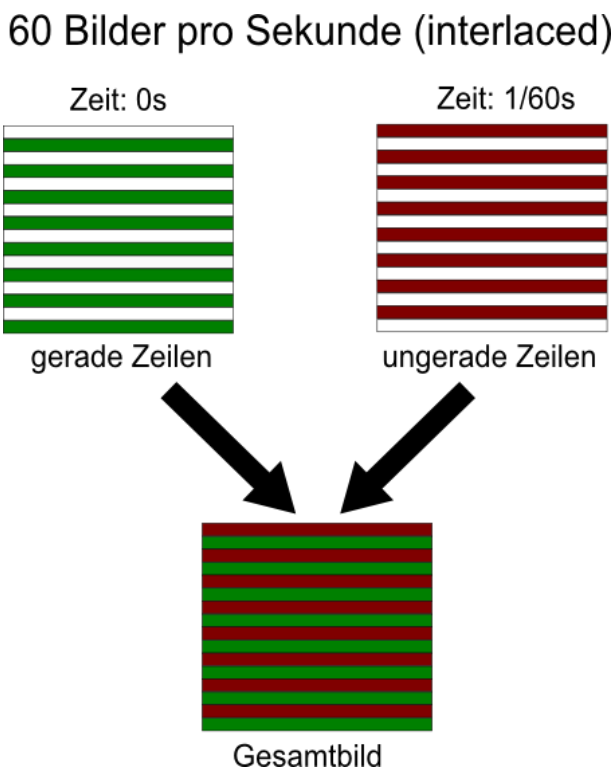

ABBILDUNG 1 – INTERLACING

2.2 Speicherplatz

Bilder benötigen besonders ab einer gewissen Größe, Qualität und Detailgrad um ein vielfaches mehr an Daten zur Speicherung und/oder Übertragung. Um Beispielsweise ein Monitorbild mit Textzeichen anzeigen zu können benötigt man für ein einzelnes Zeichen 8x8 Pixel (2 Byte), somit ergibt sich für eine FullHD-Anzeige 1920 x 1080 x 8 x 8 = 32400 Speicherplatz pro Monitorbild = 32400 x 2 Byte = 64800 Byte ~ 63,3 KByte.

Ein Pixelbild hingegen benötigt mit 256 Farben (= 2^8 = 8 Bit = 1 Byte per Pixel) 1920 x 1080 x 1 Byte ~ 2 MByte.

Geht man jetzt von einem unkomprimierten Video bestehend aus Vollbildern aus, so ergibt sich bei bereits 5 Minuten und 30 Bildern pro Sekunde eine Datenmenge von 2 x 30 x 60 x 5 ~ 17,6 GByte.

Dieses einfache Rechenbeispiel zeigt auf wie enorm wichtig Kompression von Einzelbildern bzw. ganzen Videos ist, um diese mit der heutigen Speicher- und Übertragungstechnik einer breiten Masse zugänglich zu machen. Dabei macht man sich die menschliche Wahrnehmung zu Nutze.

2.3 Helligkeit und Farbraum

Videos liegen normalerweise im RGB (Rot-Grün-Blau)-Farbraum vor. Hierbei wird aus jeder der drei Farben (im Wertebereich 0 bis 255) die gewünschte gemischt. Reines Magenta ist zum Beispiel 0 Grün, 255 Rot und 255 Blau. Dieser Wert ist in einem Pixel als Farbinformation gespeichert, wohingegen ein Bild aus einer gewissen Anzahl aus Pixeln besteht.

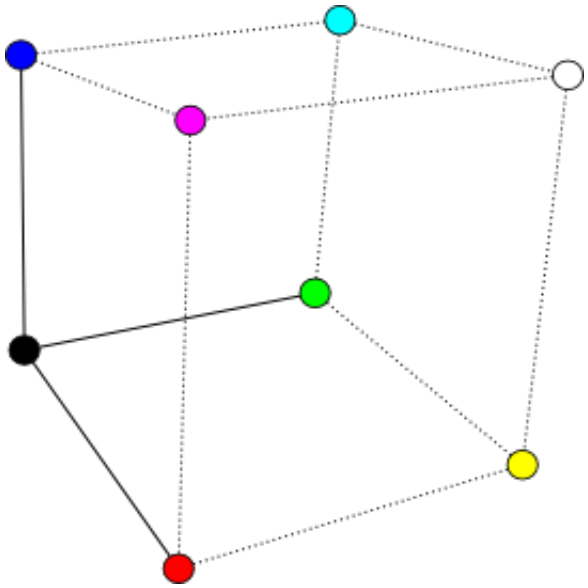

ABBILDUNG 2 - RGB FARBRAUM

Bei einem digitalen Video wird jedoch meist Y 'UV verwendet. Dieser Begriff bezieht sich auf den konkreten Y 'CbCr Farbraum. Das besondere hierbei ist, dass die Helligkeit und die Farbe in verschiedenen Komponenten gespeichert werden, um sich so die menschliche Wahrnehmung zunutze zu machen. Das Auge reagiert auf Veränderungen der Helligkeit empfindlicher als auf Farbänderungen, somit kann man Farbinformationen niedriger aufgelöst speichern, um auf diese Weise Speicherplatz zu sparen. Außerdem wurde damals so die Rückwärtskompatibilität des Fernsehsignals zu Schwarzweiß-Fernsehern gewährleistet, indem man durch die Helligkeitsinformation ein Grau-Bild erhielt. Y ist hierbei dir Helligkeitskomponente, Cb ist *color blue* während Cr für *color red* steht. Das Symbol (') markiert dabei, dass es sich bei Y (genau Y ') um die Größe *Helligkeit (engl. luma)* handelt und nicht die *Leuchtdichte (engl. Luminance)* [11].

Alle drei Komponenten lassen sich aus den RGB-Komponenten berechnen:

$$Y' = 0.212R + 0.7154G + 0.0721B$$

Diese Formel wurde an die moderne HDTV-Technologie angepasst. Die alte Berechnung enthielt 0.299 Rot-, 0.587 Grün- und 0.114 Blauanteile, die mehr an die Wahrnehmung des menschlichen Auges an Bilder einer Kathodenstrahlröhre älterer Fernsehgeräte angepasst war.

Die Chrominanz- oder Farbsättingungswerte lassen sich wie folgt berechnen:

$$U = B - Y'$$

$$V = R - Y'$$

Hierbei sind jedoch die Wertebereiche der einzelnen Komponenten an den Anwendungsbereich der digitalen Videoverarbeitung angepasst und lauten wie folgt:

$$Y' : [\ 16 – 235\ ]$$

Cb/Cr : [16 – 240], wobei der Wert 128 den Null-Wert repräsentiert.

2.3.1 Subsampling

Das bereits besprochene Verringern der Farbauflösung bezeichnet man als *Chroma Supsampling*. Drei Ziffern beschreiben die Art der Abtastung:

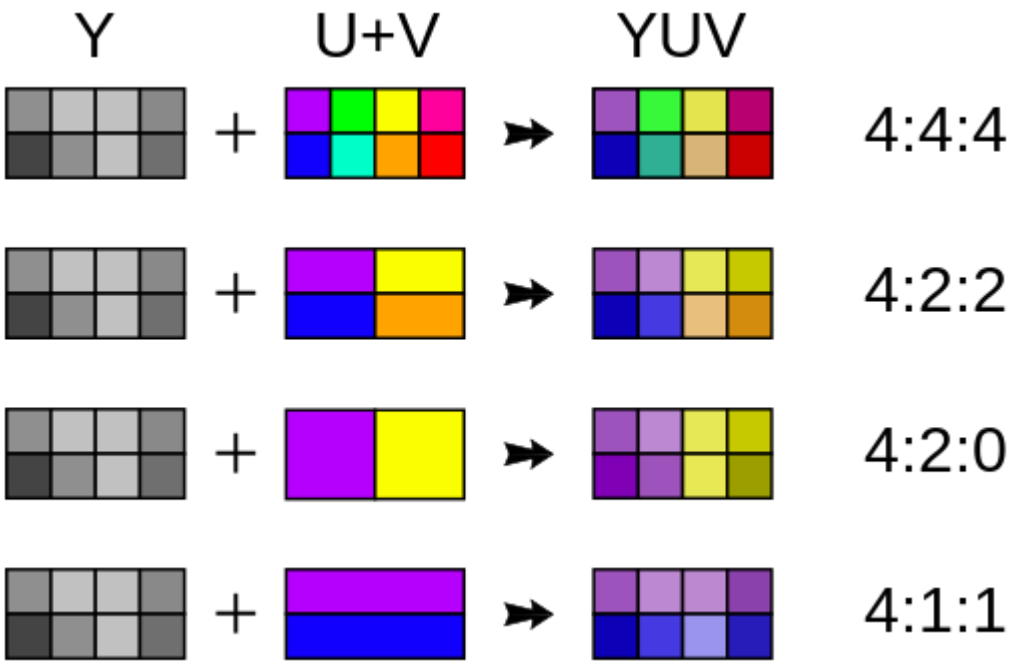

ABBILDUNG 3 - CHROMA SUBSAMPLING [12]

Der erste Wert beschreibt in zwei Reihen dargestellt, wie viele Pixel beim Subsampling für die Abtastung berücksichtigt werden. Meistens beträgt diese Zahl 4. Der zweite Wert beschreibt wie viele Farbwerte für 4 Pixel in der ersten Reihe verwendet werden, der dritte wie viele in der zweiten Reihe. Sind weniger Farbwerte als Pixel vorhanden, so müssen sich die Pixel die Farbwerte „teilen“. Auf diese Art und Weise kann die Datenmenge verringert werden, jedoch erhält man auch ein ungenaueres Bild/Bildränder [13]. Durch die Tatsache, dass mehrere Pixel die gleiche Farbe besitzen entsteht die so genannte Blockbildung (Artefakte):

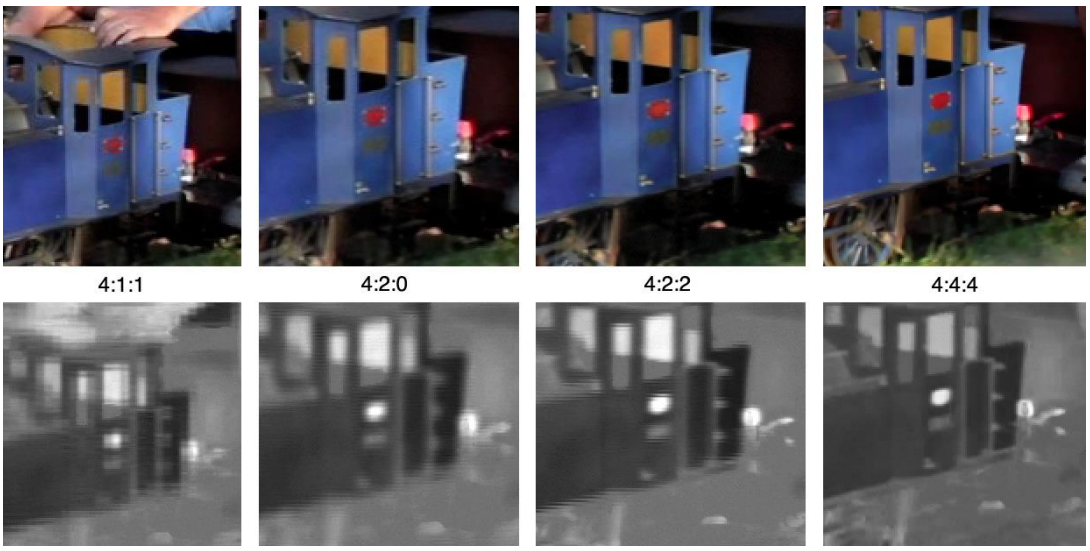

ABBILDUNG 4 - BEISPIEL SUBSAMPLING [14]

Die Darstellung der System-Grundfarben im RGB- und YUV- bzw. Y'CbCr-Farbsystem erfolgt mit folgenden Komponentenwerten:

Color	R	G	B	Y'	Cb	Cr
Schwarz	0	0	0	16	128	128
Rot	255	0	0	81	90	240
Grün	0	255	0	145	54	34
Blau	0	0	255	41	240	110
Cyan	0	255	255	170	166	16
Magenta	255	0	255	106	202	222
Gelb	255	255	0	210	16	146
Weiß	255	255	255	235	128	128

TABELLE 1 - DARSTELLUNG DER GRUNDFARBEN [13]

Auf diese Weise lassen sich Bilder bis zu 50% komprimieren, ohne dass ein bedeutender Unterschied ersichtlich ist:

4:4:4:

$$\mathrm{Y}' + Cb + Cr = 3\ \mathrm{Y}'\mathrm{CbCr}$$

4:2:0:

$$\mathrm{Y}' + \frac{1}{4}Cb + \frac{1}{4}Cr = 1.5\ \mathrm{Y}'\mathrm{CbCr}$$

Die bei heute modernen Videocodecs verwendeten Subsampling-Mechanismen verwenden 4:2:0 Subsampling. Im Falle des in dieser Arbeit behandelten VP8 Codecs wird ein 8-Bit YUV 4:2:0 Bildformat verwendet.

Subsampling zählt zu den *lossy (verlustbehafteten)* Kompressionsmethoden auf Bilder. Weitere Kopressionsmethoden werden im nächsten Kapitel aufgezeigt.

2.3.2 Weitere Kompressionstechniken

Es gibt weitere verlustlose Kompressionsalgorithmen wie *Lauflängenkodierung, Huffman Coding, Entropiekodierung* oder *Ziv-Lempel (LZW)*, die im Wesentlichen darauf basieren, sich öfter wiederholende Bildelemente nur durch die Unterschiede zu benachbarten Bildbereichen abzuspeichern. Durch verlustfreie Kompression können jedoch nur geringe Kompressionsraten erreicht werden.

Zu verlustbehafteten Verfahren gehören außer Subsampling die Reduktion der Farbinformation, Transformationskodierung wie die DCT (diskrete Cosinus-Transformation) und Quantisierung. Grundsätzlich werden hierbei die Bildinformationen vom Orts- in den Frequenzbereich übermittelt. Es wird die Eigenschaft des menschlichen Auges ausgenutzt, welches feine, detailreiche und hochfrequentierte Bereiche eines Bildes schlechter auflöst als detailärmere, niederfrenquierte Bereiche. Diese hohen Frequenzen werden „weggelassen“ indem man für jeden Pixel einen DCT-Koeffizienten beeinflusst durch benachbarte Pixel ermittelt und mittels Transformation und Quantisierung einen Weg findet diese Information platzsparender abzubilden. Diese Informationen werden noch zusätzlich mit Lauflängen- und Huffmankodierung verlustlos komprimiert [15].

2.4 WebM/VP8

Der VP8 Codec wurde von On2 Technologies entwickelt und im Mai 2010 von Google zusammen mit On2 übernommen. Bis 2008 war der VP8 Codec als geschützter Codec bereitgestellt. Das Ziel des Codec ist es, einen hochwertigen Codec für die breite Öffentlichkeit frei zur Verfügung zu stellen.

Darüber hinaus ist dies eine Initiative von Google, die Verwendung eines frei verfügbaren Videocodecs auf mobilen Endgeräten und im Browser zu verstärken.

Aus diesem Grund wird der Codec vor allem für die Verwendung im World Wide Web optimiert, während gleichzeitig eine ähnliche Leistung wie die des derzeit weit verbreiteten jedoch geschützten H.264 Codecs angestrebt wird. Gleichzeitig wurde das Containerformat für Einzelbilder namens WebP eingeführt, welches die Einzelbildkompression (siehe Kapitel 2.3 Helligkeit und Farbraum) verwendet. VP8 wurde für eine breitbandsparenden Einsatz entwickelt und soll eine Bildqualität zwischen „noch schaubar“ und „augenscheinlich verlustfrei“ liefern (ca. zwischen 30 und 45 dB auf der PSNR[7] Skala).

2.5 Intra- und Interframe Prediction

Die Kompression von Einzelbildern gewinnt eine neue Bedeutung, wenn diese in einem Kontext bzw. einer Bildfolge abgespeichert oder übertragen werden. Videos verursachen heute einen erheblichen Teil des gesamten Internetverkehrs. Hierbei ergeben sich durch den visuellen und thematischen Zusammenhang bzw. Ähnlichkeiten der dargestellten Einzelbilder völlig neue Möglichkeiten zur Kompression. Die in diesem Kapitel behandelten Algorithmen und strukturellen Eigenschaften sind hauptsächlich die des für die Softwarelösung verwendeten WebM VP8 Codec von Google, welcher jedoch sehr starke Ähnlichkeit zu den Codecs der MPEG-Familie wie z.B. des H.264 aufweist.

Ein komprimierter digitaler Videostrom besteht nicht aus einzelnen Vollbildern, sondern aus mehreren Teilbildern, die entweder aus vorherigen oder aus nachfolgenden ganzen Bildern errechnet werden können. Auf diese Weise wird zusätzlich Speicherplatz gespart, indem nicht alle vorliegenden Bilder als Gesamtbilder abgespeichert werden müssen.

[7] http://multimedia-technology.googlecode.com/svn-history/r7/trunk/lab2_jpeg/info/PSNR.pdf

Dabei wird ein Video in mehrere Groups Of Pictures aufgeteilt, bestehend aus drei verschiedenen Frametypen VP8 verwendet keine B-Frames, diese werden dennoch erklärt, dazu gleich mehr:

I-Frames: Auch Interframes genannt. Diese Bilder sind als ganze Bilder abgespeichert auf die lediglich Kompressionstechniken für Einzelbilder angewendet werden. Sie sind Schlüsselstellen aus denen weitere B- oder P-Frames errechnet werden sie dienen bei einer benutzergesteuerten Wiedergabe außerdem dem schnell Rück- bzw. Vorlauf. Das kodieren dieser Bilder benötigt die geringste Rechenzeit im Vergleich zu den anderen Bildtypen, benötigt jedoch den meisten Speicherplatz.

P-Frames: Auch Intraframes genannt. Bei P-Frames werden nur die Änderungen zum vorherigen Frame abgespeichert, um Speicherplatz zu sparen. Diese Bilder benötigen vorherige P-Frames oder I-Frames, um das vollständige Bild wiederherstellen zu können. Das Kodieren dieser Frames ist bereits rechen- und zeitintensiver.

B-Frames (nur MPEG): Hier besteht eine Abhängigkeit in beide Richtungen. Das bedeutet es werden zur vollständigen Dekodierung sowohl vorhergehende als auch nachfolgende P- und I-Frames benötigt. Diese Frames verbrauchen den wenigsten Speicherplatz, nehmen jedoch den meisten rechen- und Zeitaufwand in Anspruch.

Im VP8-Codec werden jedoch keine B-Frames also keine Berechnung aus vorherigen und nachfolgenden Einzelbildern verwendet [16].

Golden Frames (nur VP8): Auch *alternate prediction frames* genannt. Diese helfen zusätzlich die Vorhersage zu verbessern. P-Frames können sowohl aus vorherigen als auch aus dem letzten Golden Frame errechnet werden.

Jedes I-Frame ist automatisch ein Golden Frame und ein P-Frame kann optional durch das letzte Golden Frame ersetzt werden, wodurch die Toleranz gegenüber verlorenen Frames erhöht werden kann. Diese Frames können im Laufe des Kodierungsprozesses jederzeit aktualisiert werden um die Genauigkeit zu erhöhen. Ein Golden Frame kann aus dem letzten vorherigen I-Frame (Key Frame) wiederhergestellt werden, auch wenn ein, einige oder alle vorhergehenden P-Frames verlorengehen. Auf diese Weise soll Bildausfällen eines Streams durch Paketverlust entgegengewirkt werden. Es wurde außerdem festgestellt, dass ein Golden Frame von einem beliebig in der Vergangenheit liegenden Zeitpunkt (jedoch > 3 Frames) oft ein besseres Ergebnis bei der Dekodierung liefern kann, da es den Fall abdeckt, dass ein Objekt im Bild, das als verschwunden galt, wieder auftaucht. Ein Flag im Frameheader kann ein Bild zu einem Golden Frame machen. Zusätzlich zur Ablage an der Speicherposition der letzten Bilder (im Puffer), wird es auch noch an einer anderen Stelle speziell für Golden Frames abgelegt [17].

Jeder erste Datenteil besitzt einen Frame Header, der wichtige Informationen beinhaltet:

Frame Header		
Feld	**Größe (Bit)**	**Bedeutung/Wert**
frame type	1	0 für I-Frame, 1 für P-Frame
version number	3	0-3 definiert eines der vier Encoding-Profile
show_frame	1	0 für ein unsichtbares Frame, sonst 1
size	19	Größe der Daten

TABELLE 2 - VP8 FRAME HEADER [18]

Alternate Frames (nur VP8): Dieser Frame ist ähnlich dem Golden Frame, jedoch nicht zwingend ein Teil des Videos. Manchmal ist es nicht sichtbar und dient lediglich zur Referenzierung. Diese Frames dienen dazu den Nachteil auszugleichen, dass VP8 aus patentrechtlichen Gründen keine Frames referenziert werden, die nach ihnen im Video vorliegen. Der Verzicht auf B-Frames bedeutet zwar eine Komplexitätsverringerung, jedoch auch eine signifikante Erhöhung des Speicherverbrauchs. Alternate Frames sind jedoch nicht Teil des Videos. [15]

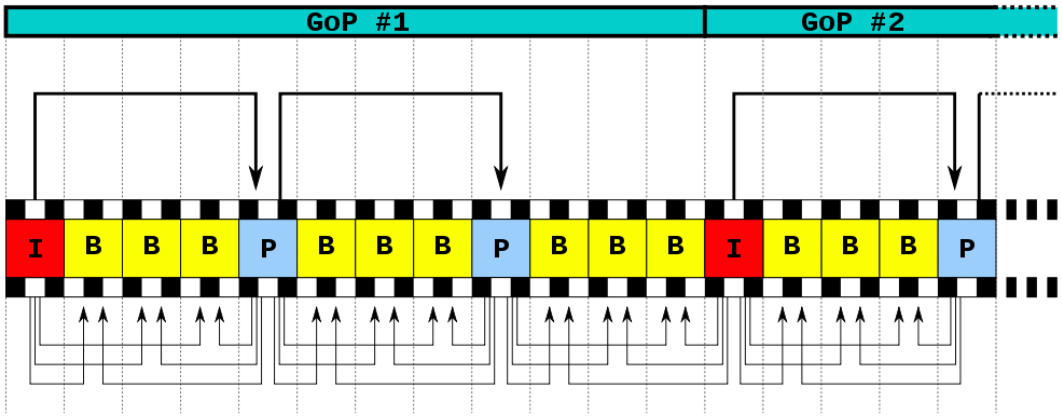

ABBILDUNG 5 - GROUP OF PICTURES (GOP) [19]

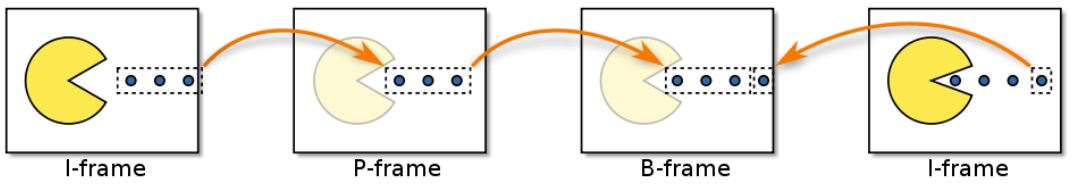

ABBILDUNG 6 - IPB BEISPIELSEQUENZ [20]

Ein Encoder muss die Frames nach dem enkodieren umsortieren, damit sie in der richtig dekodierbaren Reihenfolge im Decoder verwendet werden können.

1	2	3	4	5	6	7	8	9	10	11	12	13
I	B	B	B	P	B	B	B	P	B	B	B	I

TABELLE 3 - ENCODER EINGANG [21]

1	5	2	3	4	9	6	7	8	13	10	11	12
I	P	B	B	B	P	B	B	B	I	B	B	B

TABELLE 4 - ENCODER AUSGANG/DECODER EINGANG [21]

Zu Beginn werden alle I- und P-Frames in den Puffer gelesen, gefolgt vom ersten P- oder I-Frame, das nach einer Gruppe von B-Frames folgt, da zusätzlich zu den eingelesenen I- und P-Frames zu Beginn auch das erste nachfolgende I- oder P-Frame benötigt wird, um die B-Frames zu dekodieren. Dieser Prozess wiederholt sich für jede von Gruppe von B-Frames.

Die behandelten Kompressionsmethoden werden angewendet, indem ein Bild einzeln und bezüglich seiner zeitlichen Abfolge betrachtet wird. Ein Einzelbild wird dabei in Macroblöcke aufgeteilt und von links oben nach rechts unten abgearbeitet.

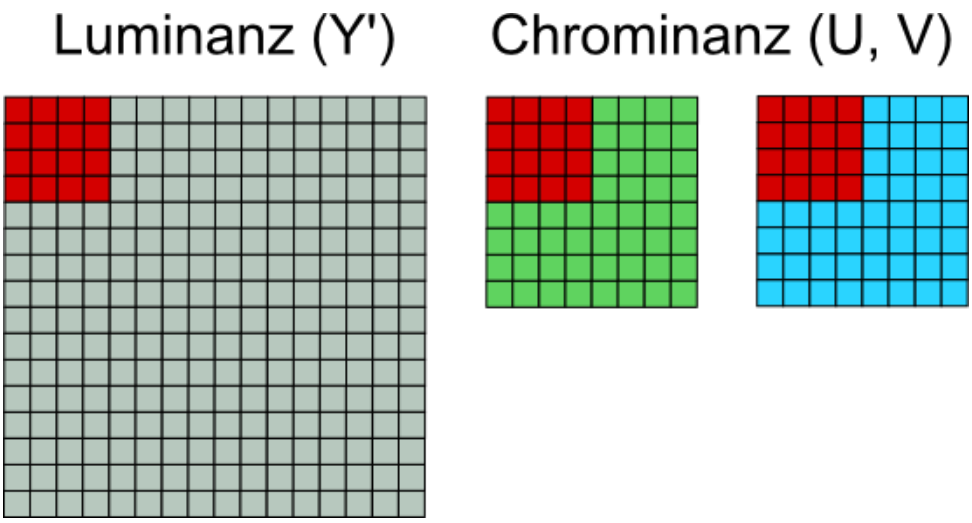

ABBILDUNG 7 - LUMINANZ- CHROMINANZ-BLÖCKE

Ein solcher Block besteht dabei aus einem 16x16 Pixel großen Luminanz- und zwei 8x8 Pixel großen Chrominanzblöcken wobei alle drei wiederum in 4x4 Pixel große Subblöcke aufgeteilt werden. So können Analyse- und Kompressionsfunktionen auf alle Blöcke ohne Rücksicht auf Skalierung oder Unterscheidung über die Art der Daten angewendet werden. VP8 ermöglicht Auflösungen von bis zu 16383x16383 Pixeln pro Bild. [15]

Loop Filtering dient dazu die durch Quantisierungsfehler entstandenen Artefakte (Kapitel 2.3.1, 0) [15] zu verringern, indem die entstandenen harten Kanten durch Angleichen angrenzender Farbwerte weicher werden. Während der *simple filter* nur die Luminanzebene verarbeitet, bezieht der *normal filter* auch die Farbwerte ein und bietet mehr Einstellungsmöglichkeiten (vergleiche [18])

2.5.1 Bewegungskompensation (Motion Compensation)

Hierbei wird versucht, die durch Kamerabewegungen oder Änderungen des Blickwinkels entstehenden Bildunterschiede vorauszuberechnen und auszugleichen. Dazu müssen die Objekte im Bild analysiert werden um Hintergrund- von Vordergrundobjekten unterscheiden zu können.

3 Game-Hijacking durch Hooking

Ein Feature, das Cloud Gaming erst praktikabel und attraktiv macht, ist das Streamen von Computerspielen auf deren Codebasis man keinen Zugriff besitzt, sozusagen auf die Inhalte von *Third-Party*-Spielen zugegriffen und in eine Cloud Gaming-Lösung integriert wird. Im Falle von Windows-Spielen, die auf DirectX basieren (welche den Großteil der heutigen AAA-Titel[8]) darstellen, ist dies die unterste Renderebene: die Schnittstelle von DirectX. Der Zugriff auf dieser Ebene ermöglicht größtmögliche Performance und Universalität. Den technischen Prozess der hierbei zum Einsatz kommt, nennt man *Hooking* oder *Hijacking*. Ein solcher Eingriff ist zwar nicht trivial weil vom Hersteller eines Computerspiels nicht vorgesehen, jedoch rechtlich unbedenklich, da keine dauerhafte substantielle Änderung am Spiel selbst vorgenommen wird. Hooking-Techniken sind aufgrund der Spezialität des Einsatzgebiets deshalb oft auch nur spärlich und inoffiziell im Internet dokumentiert vor allem weil das Thema auch stark mit Malware im Zusammenhang steht, obwohl Windows durchaus offiziell dokumentierte wenn auch nicht sehr bekannte Funktionen und sogar eine eigene Bibliothek für die Durchführung aller möglicher Arten von Hooks anbietet.

[8] http://www.wissenswertes.at/index.php?id=spiele-aaa

Anwendungsgebiete sind:

- **Monitoring:** Hierbei sollen Programme während ihrer Ausführung non-invasiv überwacht werden um beispielsweise Performancetests durchzuführen. Dabei sollen diese zwar in ihrer Ausführung, jedoch nicht in ihrer Codebasis verändert werden.

- **OS Debugging:** Programmierer können auf diese Art und Weise schlecht dokumentierte APIs von Betriebssystemen debuggen und verstehen lernen.

- **Reverse Engineering:** Um ein Programm, auf dessen Quellcode man keinen Zugriff hat, verstehen und/oder verändern zu können, benötigt man Hooking, um die Funktionsweise der APIs innerhalb des Programms in Aktion zu sehen und zu analysieren. Die Methode wird jedoch auch oft verwendet um beispielsweise Kopierschutzmechanismen auszuspionieren um diese später zu umgehen.

- **Originalfunkionalität erweitern:** Diese Anforderung wird in der behandelten Cloud Gaming-Arbeit gestellt. Die Grundfunktionalität einer Softwarekomponente kann durch Hooking um Pre- und/oder Postprocessing erweitert werden, um so neue funktionelle Eigenschaften zu erreichen, die vom Hersteller der API nicht vorgesehen wurden.

3.1 Funktionsweise

Je nachdem welche Anforderungen an die Software gestellt werden und welchen Aufwand bei der Implementierung betrieben werden soll stehen verschiedene Arten von Hooking zur Verfügung. Das Hooking von Funktionen zur Laufzeit ist ein geeigneter Einstiegspunkt in die Thematik.

Beginnend mit dem Funktion Override sind alle anderen Hooking Methoden vom Prinzip her ableitbar.

Eine Befehlsgröße im Speicher eines PC-Systems der x86-Architektur hat üblicherweise eine Länge zwischen 1 und 16 Byte(s). Um nun eine Funktion zu „hooken“, muss diese im Speicher gefunden und ein bedingungsloser Sprungbefehl (*engl. unconditional jump*) in ihren Adressbereich geschrieben werden (die Größe des Jump-Befehls beträgt 5 Byte).

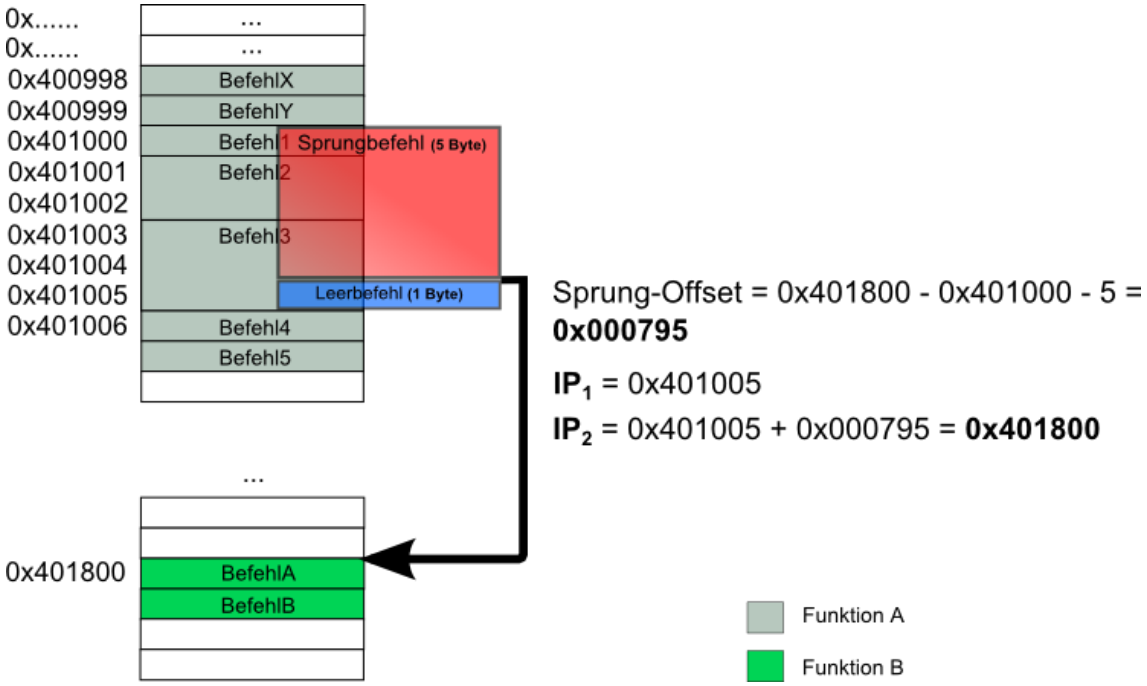

ABBILDUNG 8 - PREPROCESSING DURCH HOOKING

Befindet sich die Funktion A an der Speicheradresse 0x401000 und die Funktion B and bei 0x401800 so muss nun der Abstand (Offset) berechnet werden um von A nach B zu springen (der Einfachheit halber soll von A nach B gesprungen werden um keinen negativen Offset zu erhalten), welcher 0x400800 beträgt. Davon muss jedoch noch die Größe des Sprungbefehls (JMP) von 5 Byte subtrahiert werden da der Instruction Pointer des Systems (Adressposition des aktuell ausgeführten Befehls) nach dem Starten des Sprungbefehls und kurz vor dem eigentlichen Sprung zusätzlich die 5 Byte des Sprungbefehls enthält. Somit ist der Offset (Abstand zur angesprungenen Funktion) 0x000795.

Da die Sprungfunktion lediglich 4 Byte beträgt, wird bis zur nächsten Instruktion noch ein NOP-Feld (*engl. no operation*) ausgefüllt.

Die folgenden Funktionszeilen sind in der Maschinensprache Assembler[9] geschrieben und sollen die Maschinennähe ausdrücken (einzelne Anweisungen höherer Programmiersprachen, bestehen aus mehrerer solcher Befehle) und dienen hier lediglich als Beispiel zum Verständnis. Diese Abläufe werden im weiteren Verlauf durch standisierte Mechanismen abstrahiert [22].

```
function_A:
0x401000: push ebp
0x401001: mov ebp, esp
0x401003: sub esp, 0x40
0x401006: push ebx
0x401007: mov ebx, dword [esp+0x0c]
```

LISTING 1 - FUNKTION VOR DEM SPRUNGBEFEHL

```
function_A:
0x401000: jmp function_B
0x401005: nop
0x401006: push ebx
0x401007: mov ebx, dword [esp+0x0c]
```

LISTING 2 - MODIFIZIERTE FUNKTION

[9] http://www.cs.virginia.edu/~evans/cs216/guides/x86.html

3.2 Trampolin-Funktion:

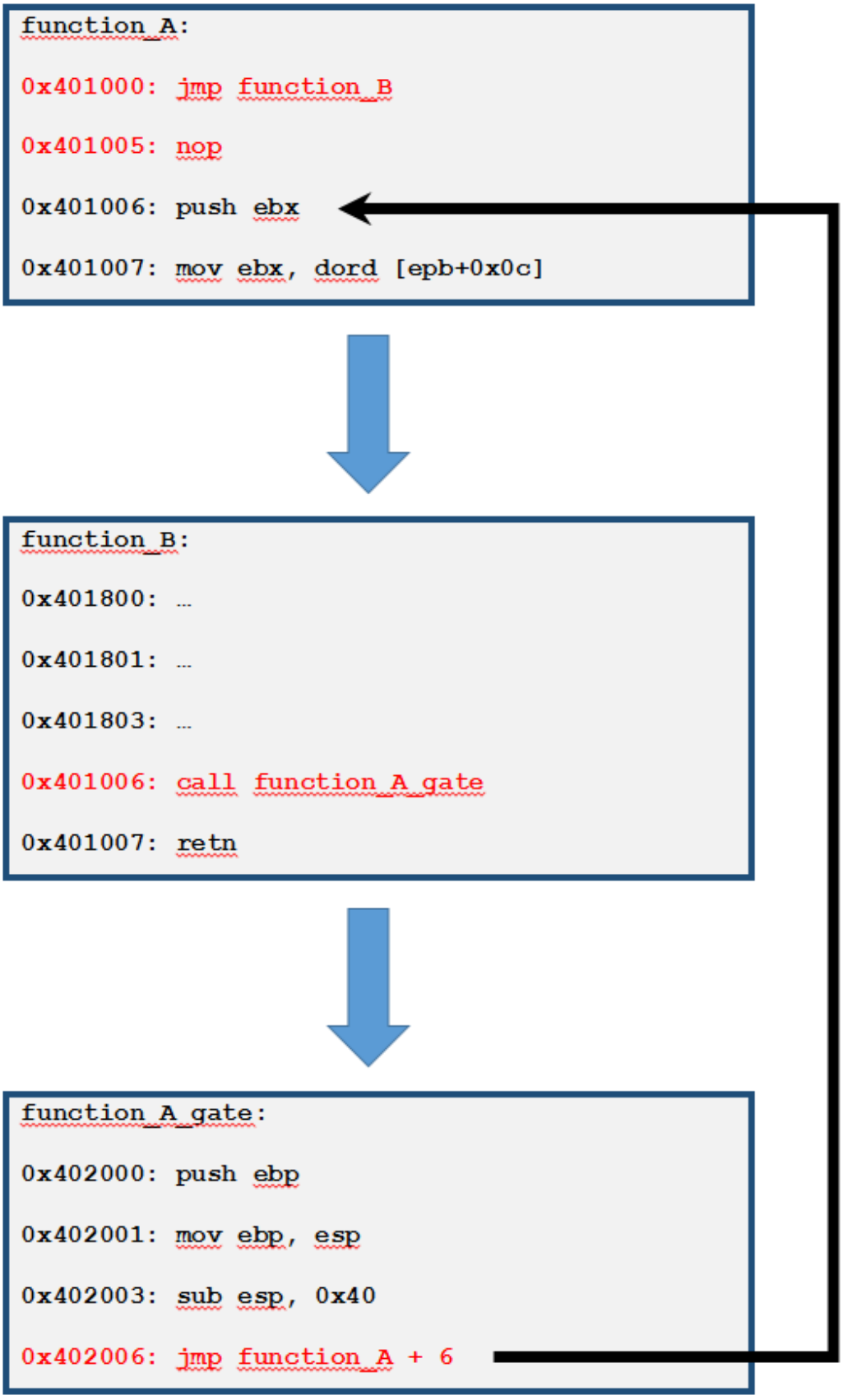

ABBILDUNG 9 – TRAMPOLIN-FUNKTION

Im vorhergehenden Beispiel wurde ein Teil der Ursprungsfunktion durch einen Sprungbefehl auf eine eigene Funktion ersetzt. Dieser Teil der Ursprungsfunktion wird auch *engl. stolen bytes* genannt. Es soll lediglich die eigene Funktion zwischen die Aufrufe der Ursprungsfunktion gesetzt werden, den Programmfluss nach dem Sprung also wieder fortsetzen. Dazu benötigt man eine Trampolinfunktion. Diese stellt sicher, dass nach der Ausführung der eigenen Funktion (function_A) eine Trampolin-Funktion (function_A_gate) aufgerufen wird, die die durch den Sprungbefehl überschrieben Teile der Ursprungsfunktion wie ursprünglich vorgesehen ausführt und dann wieder in die Ursprungsfunktion zurückspringt (und zwar an die Stelle nach dem Teil mit dem Sprungbefehl). [22]

3.3 Injektion

Als (*engl. injection*) bezeichnet man den Prozess, eine Bibliothek, die so genannte Dll-Datei (*engl. Dynamic Link Library*) in ein Programm zu „injizieren“ sodass diese vom Programm ausgeführt wird wenn dieses startet. Die bereits beschriebene Umleitung von Programmfunktionalität kann erst erreicht werden indem modifizierter Programmcode in ein Hostprogramm hinzugefügt werden wird. Das Hostprogramm muss dazu gebracht werden, den Einsprungspunkt für den fremden Code überhaupt erst auszuführen.

Eine Dll ist ein Modul, das Funktionen und Daten enthält, in diesem Fall die für das Hooking verantwortlichen, die von einer Applikation aus verwendet werden können, ohne deren genauen Aufbau zu kennen. Vorteile von Dlls sind:

- Mehrere laufende Prozesse können ein und dieselbe Dll vom selben physischen Speicherbereich laden, sodass Speicherplatz gespart wird

- Wenn sich die Inhalte der Dll-Funktionen ändern, so müssen die von ihr abhängigen Applikationen nicht neu kompiliert werden, solange die Aufrufkonventionen gleich bleiben

- Mittels Dlls können Nachrüstungen für bereits ausgelieferte Software getätigt werden (*engl. after market support*), so kann z.B. ein Grafiktreiber durch einen neuen, früher nicht verfügbaren Treiber ersetzt werden [23].

3.3.1 Injektionstechniken

Je nach Einsatzgebiet, gibt es verschiedene Injektionsarten. Es ist vor allem zu unterscheiden, ob man mehrere Applikationen (zur Laufzeit dargestellt durch laufende Prozesse) hooken möchte, die auf dieselbe Dll zugreifen oder ob lediglich eine spezielle Applikation das Ziel ist.

3.3.1.1 Systemweiter Windows-Hook

Sollen alle Prozesse übernommen werden, die eine bestimmte Bibliothek verwenden, so kann man dies erreichen indem man einen Hook in alle Windows-Prozesse installiert.

Windows bietet hierbei eine API mit eigens dafür entwickelten Funktionen. Die Funktion `SetWindowsHookEx()` mit den entsprechenden Parametern. Die Hook-Dll wird dann von Windows in den Adressraum geschrieben, den alle Prozesse anschließend verwenden. Alle Variablen, die zwischen den Prozessen ausgetauscht werden sollen, müssen in ein so genanntes geteiltes Segment (*engl. shared data section*) gesetzt werden, dies ist über eine entsprechende `#pragma`-Direktive (konkret mittels `#pragma data_seq`) der Programmiersprache C möglich. [23]

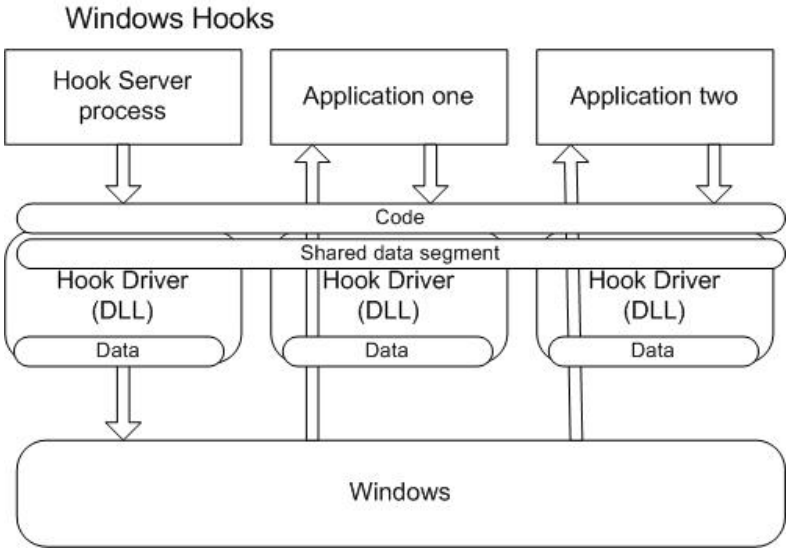

ABBILDUNG 10 - WINDOWS HOOK [23]

Ein Hook kann erst entfernt werden, wenn der Hook-Server (Code, der die Injektion für das Hooking durchführt `UnhookWindowsHookEx()`) aufruft oder die gehookte Applikation ihre Ausführung stoppt.

Vorteile:

- Verwendung der Abstraktionsebene von Windows (für das Anwendungsgebiet vergleichsweise triviale Verwendung)
- Effektive Möglichkeit, alle Prozesse im Betriebssystem zu erreichen/analysieren/modifizieren

Nachteile:

- Erschwertes Debugging durch Ausführung mehrerer Prozesse gleichzeitig
- Performance wird im Vergleich zum Hooking eines einzelnen Prozesses stärker eingeschränkt, außerdem kann es notwendig sein, den Rechner neu zu starten, um die Wirkung des Hooks aufzuheben

3.3.1.2 Injektion über CreateRemoteThread()

Diese Methode wird lediglich von NT-basierte Windowssystemen unterstützt, was jedoch seit der Einführung von Windows XP und später Windows 7 und 8 relativiert worden ist, da diese nun alle auf dem NT-Kern basieren bzw. diese Hooking-Methode anbieten.

Jeder Prozess kann zur Laufzeit eine Dll laden, indem er die Funktion `LoadLibrary()` aufruft. Auf diese Weise kann auch die eigenen Hook-Dll geladen werden. Es muss jedoch dafür gesorgt werden, dass ein fremder Prozess diese Funktion aufruft. Dafür verwendet der Injektor die Funktion `CreateRemoteThread()` mit der Adresse der Funktion, die mittels Aufruf der Funktion `GetProcAdress()` ermittelt wird (für den technisch genauen Ablauf gibt es im Web detaillierte Dokumentation inklusive Source Code Listings)[10].

[10] http://www.codeproject.com/Articles/2082/API-hooking-revealed

3.3.1.3 Proxy-Dll

Diese Methode eignet sich wenn man

a. Lediglich einen Prozess hooken will
b. Über die Funktionen und Aufrufe der Applikation (nicht deren Inhalt) bescheid weiß, sei es durch Dokumentation oder durch reverse Engineering

Es wird eine Dll-Kopie des gesamten Interfaces einer bekannten Original-Dll angelegt und diese in den Prozess injiziert. Die dann vom gehookten Prozess durchgeführten Aufrufe werden dann an die erstellte Proxy-Dll weitergegeben. Um die Hauptapplikation nicht zum Absturz zu bringen, muss oft der Aufruf nach dem Ausführen der eigenen Operationen, an die Originalfunktionen weitergegeben werden (siehe Kapitel 3.2 Trampolinfunktion).

Dieses Konzept wird im folgenden Kapitel genauer erläutert, da es sich um die gewählte Hooking-Option für die im Rahmen dieser Arbeit umgesetzte Software handelt.

4 Desktop-Client und Cloud Gaming Framework

4.1 Konzept

Cloud Gaming befindet sich noch in einem sehr frühen Entwicklungsstadium und es gibt praktisch keine offengelegten, mit offener Software implementierten Applikationen. Darüber hinaus wird dem User keinerlei Kontrolle über die Streamingparameter und Übertragungsoptionen geboten, obwohl dieser mit Hilfe vereinfachter Steuerelemente je nach Präferenz und Spieltyp maßgeblich zur Qualität des Ergebnisses beitragen könnte (nicht jedes Spiel benötigt dieselben Streaming-Verhältnisse).

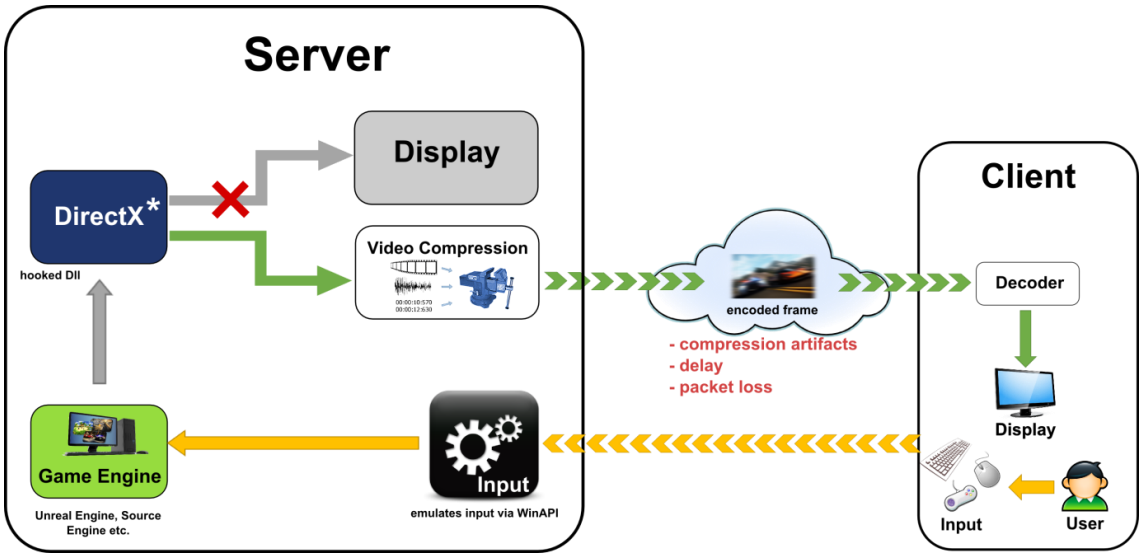

ABBILDUNG 11 - KONZEPT - CLOUD GAMING FRAMEWORK [1]

Das im Rahmen dieser Arbeit implementierte Cloud Gaming-Framework wurde speziell in Bezug auf folgende Gesichtspunkte entwickelt:

- Server
    - Modularer Aufbau der einzelnen Komponenten, sodass Wartung, Erweiterung und gegebenenfalls Austausch einzelner Module so einfach wie möglich gehalten wird
    - Kompatibilität zu den meisten gängigen DirectX-Windows-Computerspielen und gängigen Game-Engines (z.B. Source Engine, Unreal Engine 2, 3)

- Client
    - Plattformunabhängig
    - Leicht austauschbar bzw. andockbar an den Server

4.2 Systemarchitektur

Das System besteht aus den Hauptkomponenten:

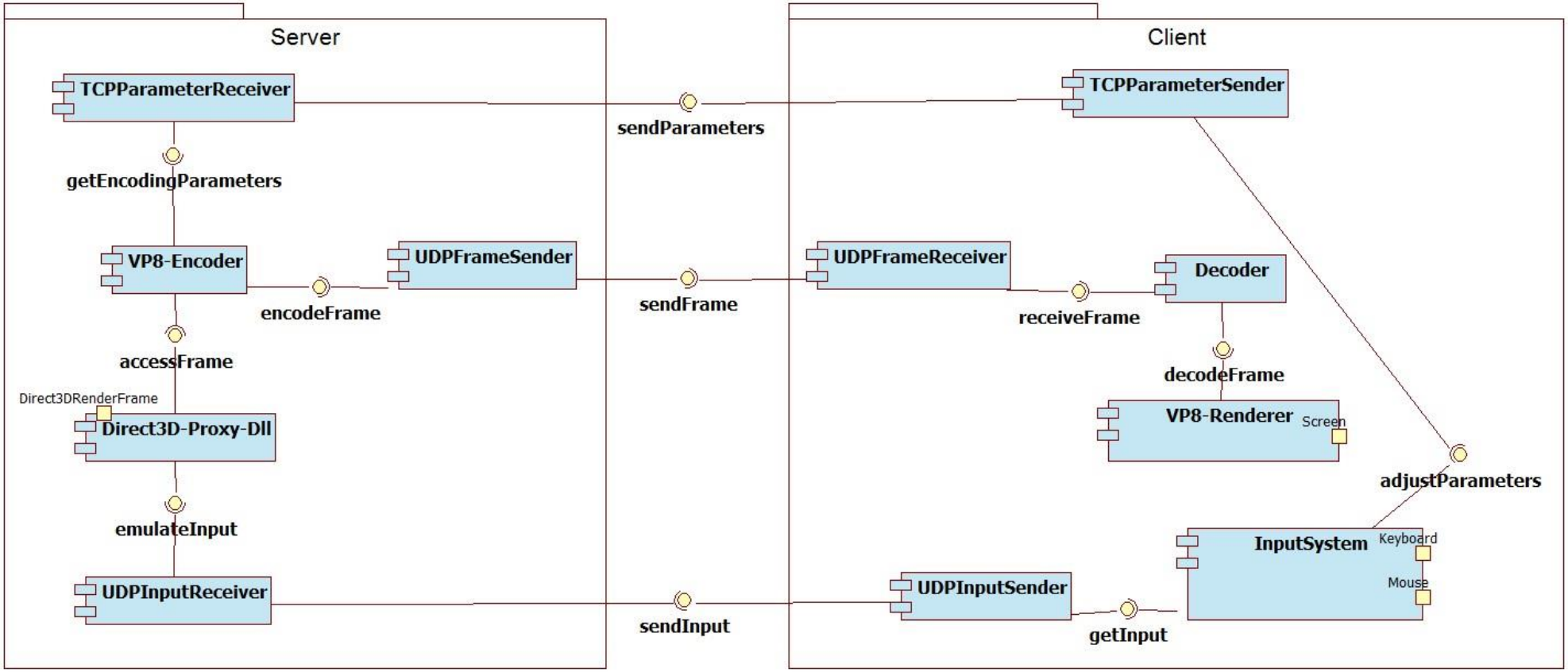

ABBILDUNG 12 - KOMPONENTENDIAGRAMM

	Komponente	Beschreibung
Server	Direct3D-ProxyDll	Hooken des Spielinhaltes als Bild pro Sekunde (Frame) und Emulation von Benutzereingaben
	VP8-Encoder	Enkodieren des extrahierten Bildes pro Sekunde (Frame)
	UDPFrameSender	Versenden des enkodierten Frames
	UDPInputReceiver	Empfangen von Benutzereingaben, die dann in das Spiel injiziert werden
	TCPParameterReceiver	Empfang von clientseitig bestimmten Initialisierungsparametern für den Encoder sowie Parameteränderung während des laufenden Betriebs (*engl. on the fly*)
Client	VP8-Decoder	Dekodieren des empfangenen Bildes pro Sekunde (Frame), damit es für die Anzeige auf dem Clientbildschirm bereit ist
	Renderer	Anzeigen des erhaltenen Spielbildes pro Sekunde (Frame)
	Input-System	Liest Maus- und

		Tastatureingaben des Benutzers und bereitet diese für das Versenden zum Server vor
	UDPFrameReceiver	Empfangen der Frames
	UDPInputSender	Senden der vom Benutzer getätigten Maus- und Tastatureingaben an den Server
	TCPParameterSender	Senden der Parameter zur Initialisierung und/oder Manipulation von Enkodierungsparametern

TABELLE 5 - SYSTEMKOMPONENTEN

In den folgenden Kapiteln wird die Funktionsweise der einzelnen Komponenten aus technischer Sicht beschrieben. Die dafür verwendeten Programmiertechniken umfassen dabei die Programmiersprache C (Proxy-Dll, Encoder, Input-Emulation, Netzwerk) und C++ (Client-User-Interface) sowie Grundkenntnisse über die Programmierschnittstellen (APIs *engl. Application Programming Interface*) von DirectX/Direct3D[11] (derzeit Version 9, Erweiterung auf 10 und 11 möglich), Windows Input API[12], Windows Hooking-API[13] und die der Videoencoding-Bibliothek FFMPEG[14].

[11] http://msdn.microsoft.com/en-us/library/windows/desktop/bb219837%28v=vs.85%29.aspx
[12] http://msdn.microsoft.com/en-us/library/windows/desktop/jj151916%28v=vs.85%29.aspx
[13] http://msdn.microsoft.com/en-us/library/windows/desktop/ms632589%28v=vs.85%29.aspx
[14] http://ffmpeg.org/developer.html

4.3 Direct3D-Proxy-Dll Hooking

Im Zuge des für diese Arbeit entwickelten Cloud Gaming entwickelten Cloud Gaming-Frameworks wird die Methode des ProxyDll-Hookings angewendet. Die im Rahmen dieser Masterarbeit erläuterte und verwendete Funktionsweise bezieht sich auf die Direct3D-Komponente von DirectX9. Direct3D wird von nahezu jedem modernen Windows-Computerspiel verwendet und befindet sich als Schicht zwischen der Game-Engine und dem Grafikkartentreiber. Es ist möglich, Spiele direkt mit Direct3D-Befehlen zu erstellen, die meisten Entwickler und großen Produktionsstudios verwenden jedoch eine Rendering-Engine, die den Prozess des Aufbaus von 3D-Szenen abstrahiert. Am häufigsten wird eine vollständige Game-Engine eingesetzt, die zusätzlich noch weitere fertige Bausteine für Netzwerk-Code für die Implementierung eines Multiplayer-Modus, Physik z.B. für Kollisionserkennung, Steuerungsmodul, das bereits vorbestimmtes Verhalten der Spielfigur und der Kameraperspektive auf Maus- und Tastatureingaben beinhaltet, ein Soundsystem für das Laden und Abspielen von Musik und Geräuscheffekten sowie einer Skripting-Unterstützung, mit der Abläufe mit wenig Programmieraufwand in das bestehende Spiel eingebettet werden können.

Eine Proxy-Dll ist ein Gerüst, das die Funktionsdeklarationen (Funktionsheader) der `d3d9.dll`-Bibliothek vollständig nach außen hin (für aufrufende Applikationen/Spiele) vollständig abbildet. Bis auf die Funktionen, die für das Cloud Gaming-Framework von Interesse sind, bestehen die Funktionsdefinitionen (Funktionskörper) jedoch lediglich aus einer Weiterleitung an die Original-Funktionen. Diese Weiterleitung besitzen auch die modifizierten Funktionen, jedoch werden vor der Weiterleitung spezielle Funktionen ausgeführt, die für das Cloud Gaming notwendig sind. Die vollständigen Funktionsheader für Direct3D9 befinden sich in der C++-Headerdatei `d3d9.h`, welche sich im Verzeichnis <Laufwerk>\<Windows-Ordner>\Microsoft DirectX SDK <Datum z.B. June 2010>\Include

befindet. Dazu muss das von Microsoft kostenlos bereitgestellte DirectX9-SDK (*engl. System Developer Kit*) heruntergeladen und installiert werden.

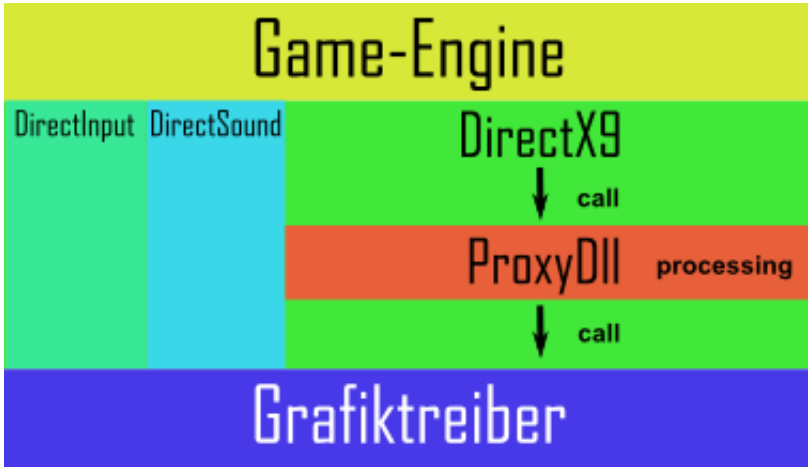

ABBILDUNG 13 - DIRECT3D-SCHICHTEN

Im Falle von Computerspielen ist keine dedizierte Injektion nötig, da bei Computerspielen, eine lokale `d3d9.dll` im Spielverzeichnis geladen wird. Somit wird die modifizierte Dll automatisch beim Spielstart geladen.

Bevor Direct3D die gehookten Funktionen aufrufen kann, muss der Hook beim Laden der lokalen `d3d9.dll` installiert werden:

```
1.  BOOL APIENTRY DllMain( HANDLE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved)

2.  {
3.      switch (ul_reason_for_call)
4.      {
5.          case DLL_PROCESS_ATTACH:
6.              InitInstance(hModule);
7.          break;
8.          case DLL_PROCESS_DETACH: ExitInstance(); break;
9.
10.         case DLL_THREAD_ATTACH:  break;
11.         case DLL_THREAD_DETACH:  break;
12.     }
13.     return TRUE;
14. }
```

LISTING 3 - DLL-EINSPRUNGSPUNKT

Beim Erstellen eines Prozesses durch das Betriebssystem, wird der Einsprungspunkt `DllMain` mit dem Status-Code `DLL_PROCESS_ATTACH` aufgerufen, auf diesen wird reagiert und die Funktion `InitInstance` aufgerufen, dort wird dann die aufrufende Programminstanz einer Variable im *shared data segment* (siehe Kapitel 3.3.1.1 Systemweiter Windows-Hook) für die spätere Verwendung gespeichert.

Als nächstes wird vom Spiel die Funktion `Direct3DCreate9` aufgerufen, um eine Direct3DDevice-Instanz zu erstellen. Dieser Aufruf wird in der Proxy-Dll abgefangen, ein eigenes Device-Objekt erstellt und das Spiel auf dieses umgeleitet:

```
1.  // Entry point in Direct3D
2.  IDirect3D9* WINAPI Direct3DCreate9()
3.  {
4.
5.      // Hooking IDirect3D Object from Original Library
6.      typedef IDirect3D9 *(WINAPI* D3D9_Type)(UINT SDKVersion);
7.      D3D9_Type D3DCreate9_fn = (D3D9_Type) GetProcAddress( gl_hOriginalDll, "Direct3D
    Create9");
8.
9.      // Request pointer from Original Dll.
10.     IDirect3D9 *pIDirect3D9_orig = D3DCreate9_fn(SDKVersion);
11.
12.     // Create my IDirect3D9 object and store pointer to original object there
13.     gl_pmyIDirect3D9 = new myIDirect3D9(pIDirect3D9_orig,iniPath);
14.
15.     // Return pointer to hooking Object instead of "real one"
16.     return (gl_pmyIDirect3D9);
17. }
```

LISTING 4 - DIRECT3D9 CREATE-FUNKTION

Ab diesem Zeitpunkt wird jede Funktion des Direct3D-Device-Objekts, die vom Spiel aufgerufen wird in Wirklichkeit auf das neu erstellte Proxy-Objekt durchgeführt. So auch die „Present“-Methode. Diese ist die einzige, deren Aufruf nicht einfach nur weitergegeben wird, denn an dieser Stelle findet der Prozess des Abgreifens des Spielinhaltes und die Enkodierung sowie Versendung über das Netzwerk zum Client statt.

```
1.  HRESULT myIDirect3DDevice9::Present(CONST RECT* pSourceRect,CONST RECT* pDestRect,HW
    ND hDestWindowOverride,CONST RGNDATA* pDirtyRegion)
2.  {
3.      // eigene Funktion
4.      GrabEncodeSend();
5.
6.     // call original routine
7.      HRESULT hres = m_pIDirect3DDevice9-
    >Present( pSourceRect, pDestRect, hDestWindowOverride, pDirtyRegion);
8.
9.      return (hres);
10. }
```

LISTING 5 - DIRECT3D9 PRESENT-FUNKTION

4.4 VP8 Encoder

4.4.1 FFMPEG

Das FFMPEG-Projekt ist eine Ansammlung von Open-Source Bibliotheken und Programmen für die En- und Dekodierung von Videos. Sie wird oft auch das Schweizer Taschenmesser der Video- und Audioverarbeitung genannt, unterstützt eine große Sammlung verschiedenster Codecs und wird von einer großen und lebhaften Community untermauert. Das Paket besteht aus einem Player zum Abspielen von Videos, Konverter für die Formatumwandlung, Server zum HTTP- und RTP-Streaming und Analyzer zur Informationsgewinnung, ist von der Kommandozeile ausführbar und die Grundlage des VLC Players[15]. Bei der Auswahl des verwendeten Codecs muss man (insbesondere bei kommerziellem Einsatz) die verschiedenen Patentsituationen per Codec beachten, da dabei eventuell Gebühren an die Patentinhaber anfallen könnten, auch wenn FFMPEG freie Software ist. Das Zickzack-Logo von FFMPEG symbolisiert den algorithmischen Ablauf bei der Entropiekodierung bei der Kompression eines Videos bzw. Einzelbildes (engl. Frame) [24] [25].

Das in der vorliegenden Masterarbeit implementierte Cloud Gaming-Framework verwendet die FFMPEG-Bibliothek für die Programmiersprache C und wird direkt in die Applikation im VP8 Encoder-Modul eingebunden.

[15] http://www.videolan.org/

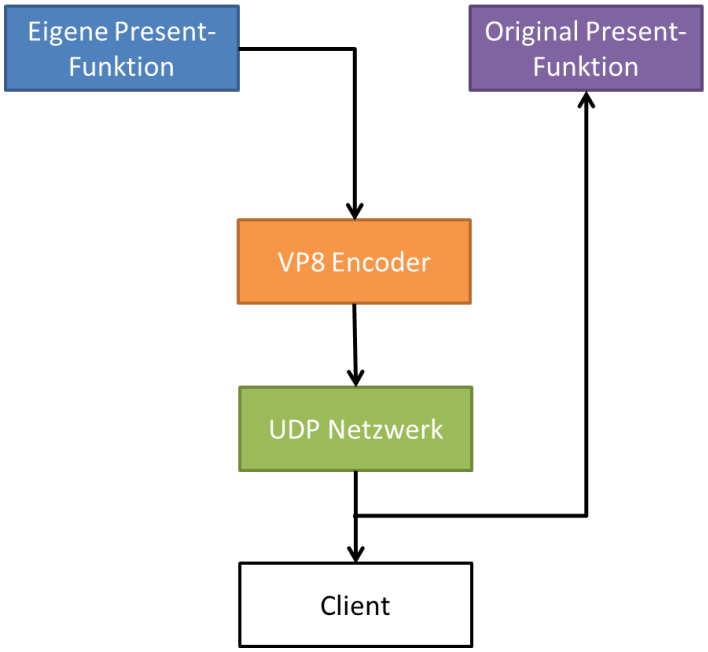

ABBILDUNG 14 – ENKODIERUNGSABLAUF

4.4.2 Technische Details

Die implementierte Software unterstützt vollständiges on-the-fly Verändern der Enkodierungsparameter: Qualitätsfaktor (bestimmt die Bitrate eines Einzelbildes), FPS, Profil, Auflösung und maximale Bandbreite in KBPS (engl. Kilobits per Second), wobei diese Einheit zur Steigerung der Benutzerfreundlichkeit dient und im Encoder wieder auf Bit pro Sekunde (was der Standard-Eingabeeinheit für die Bitrate des FFMPEG-Encoders entspricht) umgerechnet. FFMPEG unterstützt seit Version 0.7 außerdem standardmäßig Multithreading, was die Verwendung mehrerer CPU-Kernen ermöglicht und die Performance im Zeitalter der Mehrkern-Prozessoren enorm steigern kann. [24]

Die Eigenschaften der Parameter wirken sich sowohl auf die QoE (engl. Quality Of Experience) also auf das Spielerlebnis für den Benutzer auf der Client-Maschine als auch auf die Übertragungsgeschwindigkeit und Bandbreitenanforderungen (Datenmenge) aus.

FPS:

Je mehr Bilder pro Sekunde vom Spiel gesendet werden, desto flüssiger erscheint das Spiel. Dies hängt sehr stark vom Spieltyp ab. Ein Point-And-Click-Adventure kann beispielsweise auch mit 20 Bildern pro Sekunde betrieben werden, da meistens eine Spielfigur in geringem Tempo durch eine relativ statische Hintergrundlandschaft oder einen Raum bewegt wird. Ein First Person Shooter hingegen erfordert schnelle, ruckartige Reaktionsbewegungen und enthält meist mehrere stark bewegte Szenen (Gegner, aufwändigere Animationen). Meist jedoch reichen 30 FPS für ein flüssiges Bilderlebnis aus. Diese Bildwiederholrate ist derzeit Standard der aktuellen Konsolengeneration (XBOX 360[16], PS3[17], Wii[18]). Während jedoch das Bilderlebnis nur geringfügige Verbesserungen über dieser Grenze erhält (weshalb Filme meist in 24 FPS gedreht werden, wobei sich auch hier ein Trend nach oben erkennen lässt wie zum Beispiel bei „Avatar 2“ mit 60 Bildern pro Sekunde oder „Der Hobbit: Eine unerwartete Reise“ mit 48), so ist bei einem Computerspiel jede Erhöhung von Vorteil, da hier der Faktor der Eingabelatenz hinzukommt. Bei einem Rennspiel als Beispiel legt ein Fahrzeug im Spiel bei 216 km/h bei 24 Bildern pro Sekunde eine Strecke von 2,5 Metern zurück, bei 48 FPS sind dies nur noch 1,25 Meter, bei 120 FPS 0,313 Meter, nur zwischen diesen Abständen reagiert das Spiel auf Tastatureingaben, da pro Renderzyklus Eingaben verarbeitet werden. Ein Renderzyklus bei 60 FPS beträgt ca. 17 Millisekunden (1000ms/60), bei mehr Bildern pro Sekunde verringert sich diese Dauer.

Auf diese Weise reagiert ein Fahrzeug bei einer höheren Bildwiederholrate in kürzeren Abständen, somit präziser. Diese Latenzveränderung macht sich in reaktionsschnellen Spielen deutlich bemerkbar [26].

[16] http://www.xbox.com/de-AT/Xbox360
[17] http://at.playstation.com/ps3/
[18] http://www.nintendo.de/Wii/Wii-94559.html

AUFLÖSUNG/BITRATE/PROFIL/BANDBREITE:

Liefert die wohl deutlichste Qualitätsveränderung für den Benutzer. Die Bitrate beschreibt, wie viele Farbinformationen ein Pixel enthält und je höher die Bitrate, desto weniger Blockbildung (zu wenig Informationen werden für größere Bereiche interpoliert, dadurch erhalten größere Bereiche die gleiche Farbinformationen und werden als Blöcke sichtbar. Je nach verwendeter Auflösung (Größe des Bildes) ist eine höhere Bitrate notwendig, um die gewünschte Qualität zu erreichen. Eine geeignete Formel zur Berechnung der benötigten Streaming-Bitrate ist die Kush Gauge:

Sie basiert auf drei Enkodierungs- und Streaming-Parametern:

- Anzahl der Pixel pro Einzelbild (Auflösung) = Breite * Höhe des Bildes = „Pixel Count“
- Anzahl der Einzelbilder pro Sekunde (FPS)
- Bewegungsintensität der gezeigten Szene (low/mid/high) = „Motion Rank“. Dabei steht low für die Ziffer 0, mid für 2 und high für 4
- Hinzu kommt eine Konstante von 0.07, die mit Experimenten ermittelt wurde und realistische Bitrate-Werte hervorbringt

Dies führt zur Formel:

$$\frac{Pixel\ Count * FPS * Motion\ Rank * 0.07}{1000} = x\ kilobytes\ per\ second$$

Dies bedeutet z.B. bei HD-Auflösung mit 24 Bildern/Sekunde (1280x720 @24fps) und Motion Rank von 2:

$$\frac{1280 * 720 * 24 * 2 * 0.07}{1000} \sim 3000 kbps$$

Diesen Wert muss die Bandbreite minimal aufweisen. Die berechnete Größe geht hierbei von einem unkomprimierten Einzelbild aus, weshalb die eigentliche Bitrate teilweise weit darunterliegen kann, da I-Frames mit JPEG-Kompression verkleinert werden und P-Frames, lediglich Teile eines Gesamtbildes ausmachen und wiederum erheblich kleiner sind.

GOP:

Ein weiterer Faktor, der das Streaming eines Videos (in diesem Fall Einzelbilder des Spiels, die einen Videostream ergeben) stark beeinflussen kann ist die *Group Of Pictures* (siehe Kapitel 2.5 Intra- und Interframe Prediction). Eine Bitrate wird stark davon beeinflusst, wie die GOP gewählt wird. Je größer die GOP, desto mehr P-Frames werden verwendet, wodurch die übertragene Datenmenge stark sinkt. Dies hat jedoch den Nachteil, dass bei Paketverlusten die Wahrscheinlichkeit von Bildstörungen ansteigt, denn verliert man z.B. ein I-Frame, so sind auch alle P-Frames, die von diesem I-Frame abhängen wertlos. Die Anpassung dieses Parameters ist somit abhängig von der vorhandenen Netzwerkinfrastruktur.

4.4.3 Weitere Parameter

Abseits der Grundlegenden Parameter, die die Übertragung und Datenmenge beeinflussen, gibt es noch Parameter, die die Performance des Servers im speziellen die CPU-Auslastung beeinflussen.

CPU_USED: Ein Wert zwischen -16 und +16 der je nach Maschine experimentell gewählt werden muss, ermöglicht ein schnelleres Enkodieren, indem die CPU stärker ausgelastet wird, muss der Server keine GUI oder sonstige Applikationen betreiben, kann 16 als Wert gewählt werden.

THREADS: Anzahl der Threads, die für das Enkodieren verwendet werden. Durch diese Einstellung werden mehrere parallele Prozesse von FFMPEG für das Enkodieren gestartet und verwaltet. FFMPEG unterstützt Threads offiziell seit der Release-Version 0.6.2. Experimente mit der in dieser Masterarbeit entwickelten Software haben gezeigt, dass die Anzahl der Threads gleich der Anzahl der CPU-Kerne der Servermaschine entsprechen muss, um die besten Ergebnisse zu erzielen. [24]

HYPERTHREADING[19]: Hyperthreading wird von Intel-CPUs unterstützt und ermöglich eine Verdoppelung der verwendbaren CPU-Kerne durch virtuelle Threads. Diese Methode hat auch Einfluss auf die Cloud Gaming-Software, die die Anzahl der CPU-Kerne automatisch erkennt. Hyperthreading wird jedoch oft als Marketinginstrument der Firma Intel angesehen, da viele Benchmarks zeigen, dass es nahezu keine Verbesserung bringt.

4.5 Netzwerkmodul

Das Netzwerkmodul stellt die Kommunikationsschnittstelle zwischen Client und Server dar. Es ist das Werkzeug, um das enkodierte Bild an die Client-Maschine zu senden und Eingaben des Benutzers zu empfangen. Außerdem werden von der Netzwerkschicht Konfigurationsinformationen zum Verändern von Enkodierungsparametern angenommen. Diese Einstellungen kann der Benutzer clientseitig über die bereitgestellte Benutzeroberfläche (engl. GUI = Graphic User Interface) beeinflussen.

[19] http://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html

4.5.1 UDP (User Datagram Protocol)

Für das Senden von Bildern zum Client wird aus Performancegründen das Netzwerkprotokoll UDP verwendet. Dieses besitzt im Gegensatz zu TCP keinerlei Mechanismen um die Ankunft und Reihenfolge der Pakete zu gewährleisten. Dadurch kann es je nach Eigenschaften der Verbindungsleitung zu Paketverlusten kommen. UDP gilt somit als verbindungslos und unzuverlässig. Dies wird jedoch in Kauf genommen, da es bei einem Videostrom tolerierbar ist, einige Pakete zu verlieren, da der Strom dadurch dennoch intakt bleibt. Der Vorteil von UDP ist hierbei, dass durch den Wegfall der gerade genannten Kontrollmechanismen viel Protokoll-Overhead (Zusatzinformationen und deren Verarbeitung) wegfällt und somit die Performance drastisch anhebt. Deshalb wird UDP vornehmlich für Internettelefonie und Videokonferenzen verwendet, da hier eine möglichst geringe End-zu-End Latenz von größter Bedeutung ist [27].

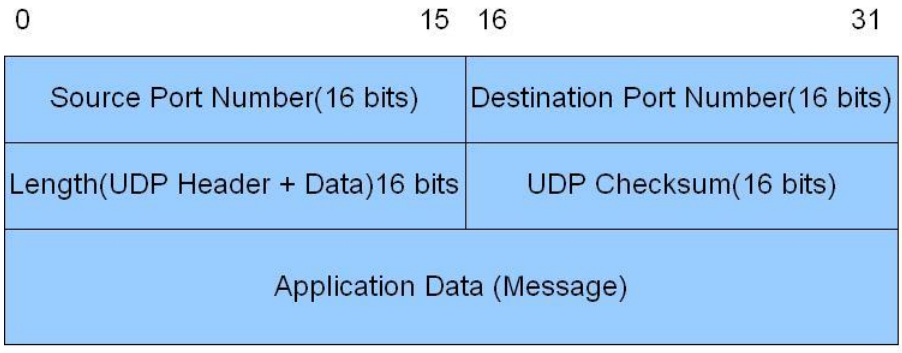

ABBILDUNG 15 - UDP HEADER AUFBAU [27]

Die im Rahmen der vorliegenden Masterarbeit angefertigte Applikation verwendet UDP außerdem noch für das Versenden von Maus- und Tastatureingaben vom Client und Server verwendet. Auch hier ist es wichtiger schnell zu übertragen als jedes einzelne Tastendruck-Ereignis (Key-Press-Event) oder jede einzelne Mausbewegung (Mouse-Move-Event) zu übertragen, da bei einem kurzfristigen Ausfall die Bewegungsinformation durch nachfolgend gesendete Koordinaten kompensiert wird.

4.5.2 TCP (Transport Control Protocol)

TCP arbeitet im Gegensatz zu UDP zuverlässig und stellt sicher, dass verloren gegangene Pakete nachgesendet werden und die Reihenfolge eingehalten wird. Für die Kommunikation zwischen zwei Anwendungen wird für die Dauer der Kommunikation eine zuverlässige Verbindung auf- und nach Beendigung dieser wieder abgebaut. Darüber hinaus bietet TCP einen Puffer zur Flusskontrolle (falls Pakete nicht in einer gleichmäßigen Geschwindigkeit ankommen), dieser wird *Window* genannt [28].

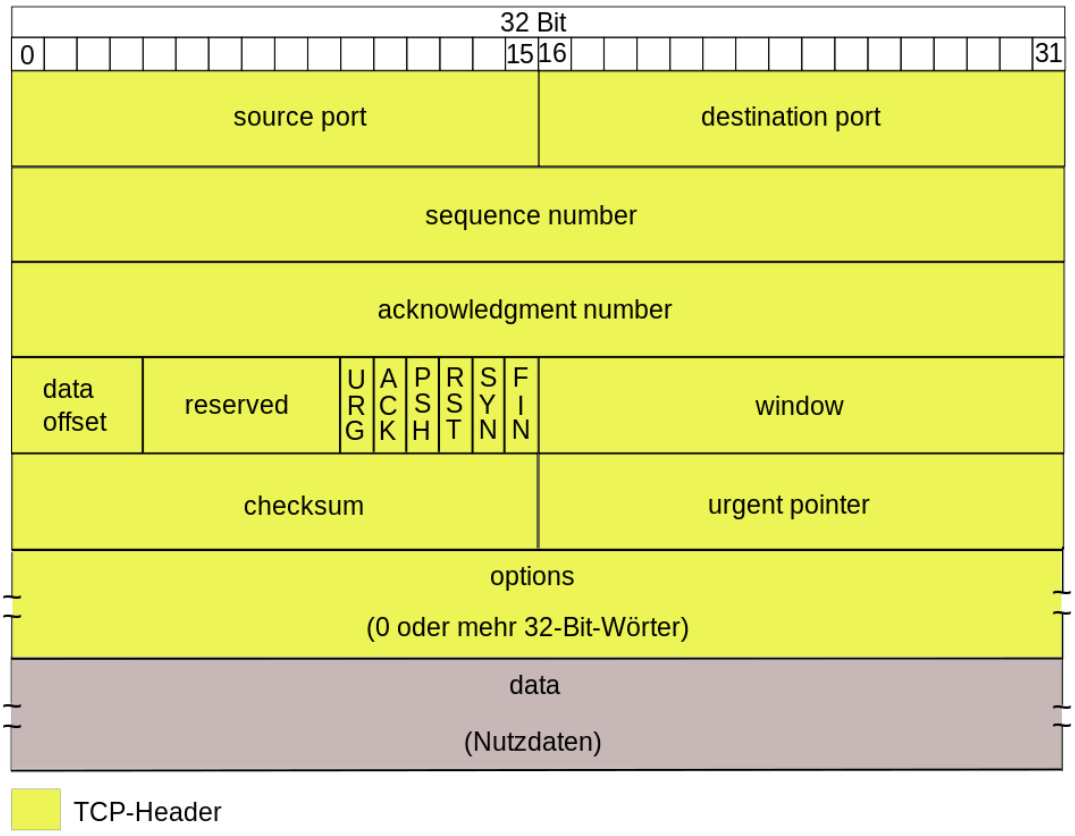

ABBILDUNG 16 - TCP HEADER AUFBAU [28]

Durch die zusätzlichen Felder *sequence number* und *acknowledgment number* versichern eine geordnete und zuverlässige Verbindung.

Die Cloud-Gaming-Applikation verwendet TCP zur Absprache über den Beginn der Streaming-Session indem der Client eine Anfrage startet und auf eine Bestätigung wartet. Ist dieser Handshake erfolgt, sendet der Client eine Anfrage mit seinen gewünschten Streaming-Parametern:

```
1.  void CommandSystem::sendConfig(
2.  int width,
3.  int height,
4.  int bitrate,
5.  int fps,
6.  int maxbitrate,
7.  int profile) {
8.      sprintf(buf,"%s;
    bitrate=%i;fps=%i;width=%i;height=%i;maxbitrate=%i;profile=%i;", "K", bitrate, fps,
    width, height, maxbitrate, profile);
9.      tcpSender.send(buf,strlen(buf));
10. }
```

LISTING 6 - CLIENT SENDCONFIG-FUNKTION

4.6 Client

In diesem Modul geschehen zwei grundlegende Aufgaben: das Empfangen und Anzeigen der Frames und das Entgegennehmen von Maus- und Tastatureingaben und Versenden dieser zum Server, um damit den Spielablauf zu steuern.

Die Einzelbilder werden vom Server empfangen und dekodiert, um diese anschließend auf dem Bildschirm anzeigen zu können. Die dazu verwendeten Parameter werden, wie bereits beschrieben, vom Client zur Serverseite gesendet, wo diese dann eingestellt werden, clientseitig wird nur noch der Decoder initialisiert. Dafür wird vom Server lediglich die Anzahl der benötigten Channels übermittelt (meist 4 für RGBA, rot, grün, blau und der Alphakanal der die Transparenz eines Pixels bestimmt).

Die Clientsoftware verwendet das C++-Framework Qt[20]. Dadurch wird ein möglichst großer Abstrahierungsgrad durch objektorientierte Programmierung bei gleichzeitig möglichst großer Kompatibilität zur serverseitigen C/C++-Softwarelösung gewährleistet.

Ein großer Vorteil ist hierbei, dass Qt auf einer großen Anzahl von Endgeräten z.B. Windows, Linux, embedded Linux zum Einsatz kommen kann.[21]

Der Client kann auch andere Eingabegeräte wie z.B. Gamepads unterstützen. Sofern die Clientlösung Netzwerkfunktionen und eine FFMPEG-Bibliotheksimplementierung bereitstellt, kann dies auch in einer völlig anderen Programmiersprache auf unterschiedlichsten Betriebssystemen realisiert werden, denkbar sind dabei unter anderem Android, iOS oder

[20] http://qt-project.org/
[21] http://qt-project.org/doc/qt-4.8/supported-platforms.html

eine Implementierung im Webbrowser, da die Hardware lediglich die Voraussetzungen zur Dekodierung eines Videos erfüllen muss.

Von Vorteil sind die vielfältigen Qt-Module oder Bibliotheken, mit deren Hilfe Entwickler schnell und komfortabel Funktionalität programmieren können. Die wichtigsten abgedeckten Bereiche sind hierbei:

- GUI-Programmierung (inkl. WYSIWYG-Editor „Qt Creator“)
- Netzwerk (QSockets)
- 3D-Programmierung (für die im Rahmen der vorliegenden Masterarbeit entwickelte Software wird das OpenGL-Widget verwendet, um die dekodierten Frames möglichst schnell und effizient anzeigen zu können)

Weitere Module sind z.B. QtScript für die Ablauferstellung, QtMultimedia u.v.m.[22]

[22] http://qt-project.org/doc/qt-4.8/modules.html

4.6.1 Input Injection

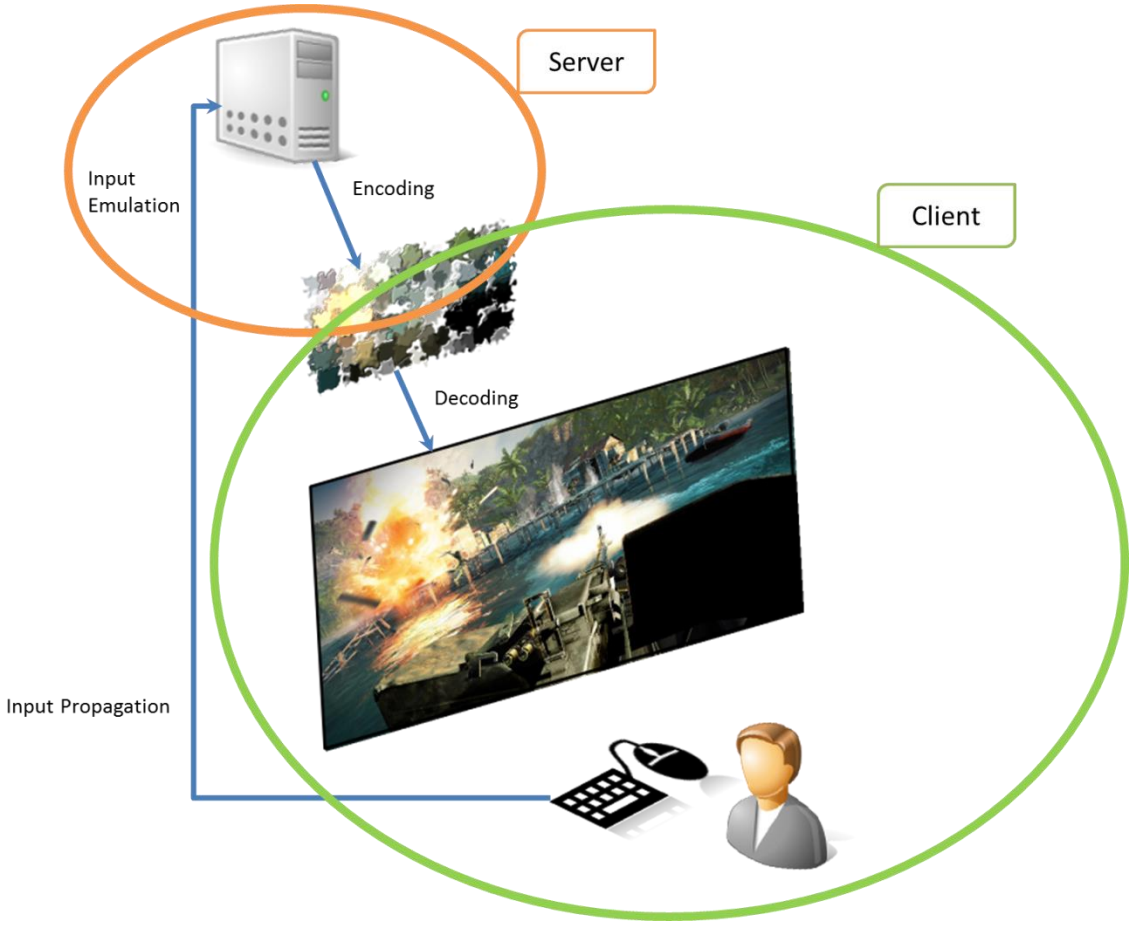

ABBILDUNG 17 - CLIENT ARCHITEKTUR

Die vom Benutzer getätigten Eingaben, sei es Maus oder Tastatur werden per UDP (siehe Kapitel 4.5.1 UDP (User Datagram Protocol) an den Server gesendet, um damit das Computerspiel fernsteuern.

Die Herausforderungen dieses Systems sind - außerhalb des Übertragungsweges über das Netzwerk – die Erkennung des Zustands des Spiels in Bezug auf die Art und Weise wie Maus- und Tastatureingaben vom Spiel zu welchem Zeitpunkt den Spielablauf beeinflussen.

4.6.1.1 Mouse-Locking

Der Zustand des Mauscursors lässt sich grundsätzlich in zwei verschiedene Zustände einteilen: Standard (*engl. default*) oder gesperrt (*engl. locked*). Die entspricht den beiden Zuständen absolute und relative Cursorbewegung: Im ersten Zustand verhält sich der Mauscursor wie man es vom Betriebssystem gewohnt ist. Durch jede Bewegung wird der Cursor am Bildschirm in Bewegung versetzt, erreicht dieser den Bildschirmrand so bleibt der Cursor stehen. Dies wird bei Adventures oder Strategiespielen verwendet, da hier eine Begrenzung der Spielwelt durch die Front- oder Vogelperspektive gegeben ist. Außerdem werden Spielobjekte wie z.B. Kampfeinheiten bei Strategiespielen oder Rätselobjekte bei Adventure-Spielen direkt ausgewählt, weshalb der absolut positionierte Cursor auch auf dem Bildschirm angezeigt wird.

In vielen Computerspielen, die eine Steuerung eines Fahrzeugs/Raumschiffs etc. aber auch Spielfiguren aus der Ego- oder Third-Person-Perspektive ermöglichen, wird der Cursor ausgeblendet und die Mausbewegungen vom Computerspiel relativ interpretiert. Jede Bewegung steuert das Raumschiff bzw. die Kamera so weit in die gewünschte Richtung, wie die Maus bewegt wird, der Zeiger wird dann vom Spiel wieder in die Mitte gesetzt, da die Bewegung endlos fortgesetzt werden kann (z.B. eine Mausbewegung nach rechts dreht die Spielfigur nach rechts, dies kann unendlich oft um die eigene Achse durchgeführt werden).

Dieser Umstand ist für die Cloud-Gaming-Applikation von besonderer Bedeutung, da sich der Mauscursor – je nach Spielzustand - auch auf dem Client entsprechend korrekt verhalten muss, um eine Fernsteuerung des Spielgeschehens zu ermöglichen.

Dafür gibt es zwei Möglichkeiten: die erste ist, dass der Server eine Änderung des Cursorzustands erkennt und per TCP sendet, sobald diese eintritt.

Die Verwendung von TCP ist möglich und notwendig, da die Zustandswechsel einerseits in den meisten Spielen selten vorkommen (Wechsel von einem Autorennen ins Hauptmenü) somit keine große Belastung für den Netzwerkverkehr darstellen und andererseits die Zustandsänderung unbedingt zuverlässig beim Client ankommen muss, da ansonsten das Spiel nicht mehr zu steuern wäre. Der Client wiederum blendet den Cursor aus und sperrt ihn in der Mitte falls der Cursor serverseitig gesperrt ist und übermittelt nur noch relative Mausbewegungen d.h. *Cursor hat sich um zwischen Position x und y um z Pixel bewegt* anstatt lediglich die absolute Position zu senden. Der Server interpretiert den Input je nach Zustand absolut oder relativ.

Eine weitere Möglichkeit ist, dass der Cursor clientseitig immer gesperrt ist das heißt, dass dieser ausgeblendet und nur relative Bewegung übertragen wird. Der Server wiederum interpretiert jede Bewegung relativ, jedoch wird immer wenn ein sichtbarer Cursor benötigt wird, serverseitig ein Cursor auf das zu übertragende Bild gezeichnet, bevor dieses enkodiert und zu Client übertragen wird.

Die zweite Variante wird in der im Rahmen dieser Masterarbeit erstellten Applikation verwendet und ist zu bevorzugen, da der Synchronisationsaufwand weit geringer und die Syntax der Maus-Input-Übertragung zu jedem Zeitpunkt einheitlich ist (kein Wechsel zwischen absoluter und relativer Bewegung). Diese Variante löst zudem ein weiteres Problem, das durch entsteht, dass Computerspiele eine große Vielfalt an Engines und Technologien verwenden und zwar die Unterscheidung zwischen Hardware- und Softwarecursor:

4.6.1.2 Hardware- und Softwarecursor

Der Server kann absoluten oder relativen Zustand lediglich dadurch feststellen, dass die DirectX-Proxy-Dll überprüft, ob der Cursor sichtbar ist oder nicht. Dazu wird die Windows-Funktion *GetCursorInfo* verwendet. Diese liefert den Zugriff auf die Datenstruktur CURSORINFO, welche Informationen über die Sichtbarkeit des Mousecursors enthält. Die Konstante CURSOR_SHOWING steht hierbei für den Hex-Wert 0x00000001. Ist der Cursor im Spiel auf dem Server sichtbar, so wird der Client-Cursor eingeblendet und der Modus auf absolut umgestellt und vice versa.

Viele Spiele verwenden jedoch einen Hardwarecursor, es wird nicht der Windows-oder Softwarecursor verwendet (dieser lässt sich durch Icon-Wechsel auch ans Spiel anpassen), sondern ein Hardwarecursor direkt auf das Frame im Spiel gezeichnet und jede Bewegung trotz absolutem Modus, relativ interpretiert. Passiert dies, ist das Spiel nicht mehr steuerbar, da der Client das Mausverhalten nicht zuverlässig bestimmen und sich darauf einstellen kann. Deshalb bietet die bereits erwähnte zweite Variante mit dauerhaft relativer Steuerung die zuverlässigste Art der Bestimmung des Cursormodus. Durch das serverseitige Zeichnen eines kleinen Rechtecks an der Position des Cursors als Mauszeigerersatz wird außerdem immer sichergestellt, dass die Cursorpositionen genau übereinstimmen und der Synchronisierte Clientcursor nicht abweicht.[23]

Eine weitere Möglichkeit wäre, die Cursorposition immer punktgenau an den Client zu übertragen, wodurch jedoch die gerade genannten Abweichungen durch verlorene Pakete entstehen können, da man für einen Strom an Positionsinformationen UDP verwenden müsste, um Performanceproblemen vorzubeugen.

[23] http://msdn.microsoft.com/en-us/library/windows/apps/hh994925.aspx

ABBILDUNG 18 - ADVENTURE-SPIEL MIT ABSOLUTEM CURSOR

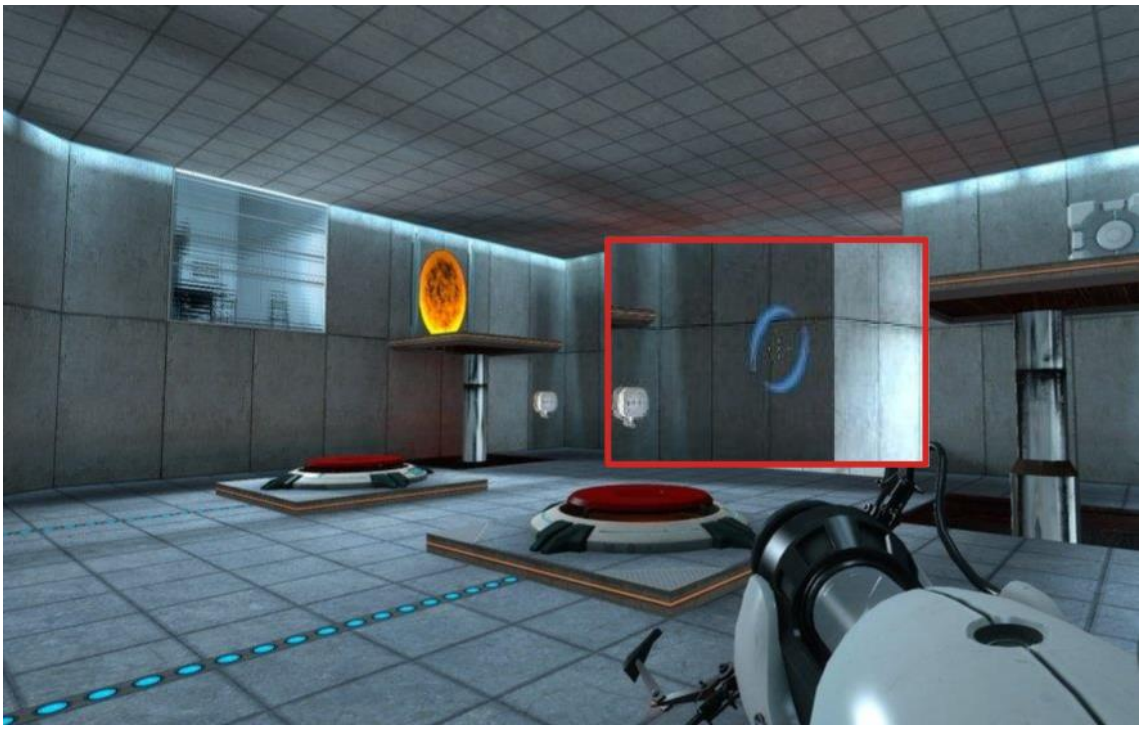

ABBILDUNG 19 - FIRST-PERSON-SHOOTER MIT RELATIVEM CURSOR

4.6.2 Input Emulation mittels WinAPI

Um eine serverseitig empfangenen Maus- oder Tastatureingabe in ein Spiel zu injizieren gibt es drei Grundlegende Möglichkeiten:

- Hooking der DirectInput-Bibliothek
- Hooking von XInput
- Input-Emulation mittels WinAPI

Während DirectInput auf Maus und Tastatur als Eingabegeräte spezialisiert ist, wird XInput für den XBOX-Controller verwendet. Da nicht jedes Spiel einheitlich bei der Verwendung von Input-Bibliotheken ist, wird die dritte Variante von der im Rahmen dieser Masterarbeit erstellten Applikation bevorzugt.

Dabei werden dem gesamten Betriebssystem über WinAPI Eingaben emuliert. Der Nachteil an dieser Methode ist, dass das gesamte System beeinflusst wird und man dadurch z.B. Client und Server nicht auf demselben System starten kann, da sich der Client sonst selbst steuert.

Der große Vorteil ist jedoch, dass damit jedes Spiel unabhängig von der internen Input-Implementierung ferngesteuert werden kann.

Dazu wird die Funktion *SendInput* der WinAPI verwendet. Diese ruft man mit einer INPUT-Struktur auf, in der sich alle auszuführenden (auch multiplen) Input-Befehle befinden, die dann vom Betriebssystem abgearbeitet werden.

```
1.  UINT WINAPI SendInput(
2.    _In_  UINT nInputs,
3.    _In_  LPINPUT pInputs,
4.    _In_  int cbSize
5.  );
```

LISTING 7 - WINAPI SENDINPUT-FUNKTION

nInputs: Anzahl der zu verarbeitenden Inputs.

pInputs: Adresse eines Arrays befüllt mit INPUT-Strukturen (siehe unten).

cbSize: Größe des INPUT-Struktur-Arrays in Bytes.

Eine Input-Struktur enthält hierbei einen UNION-Datentyp (auch bezeichnet als „Entweder-Oder-Verbund“), somit die Informationen über eine Maus- oder Tastatureingabe (Hardware steht dabei für ein Eingabegerät, das nicht eine Maus oder Tastatur ist).

```
1.  typedef struct tagINPUT {
2.    DWORD type;
3.    union {
4.      MOUSEINPUT    mi;
5.      KEYBDINPUT    ki;
6.      HARDWAREINPUT hi;
7.    };
8.  } INPUT, *PINPUT;
```

LISTING 8 - INPUT-DATENSTRUKTUR

tagKEYBDINPUT stellt einen einzelnen Tastaturinput dar:

```
1.  typedef struct tagKEYBDINPUT {
2.    WORD      wVk;
3.    WORD      wScan;
4.    DWORD     dwFlags;
5.    DWORD     time;
6.    ULONG_PTR dwExtraInfo;
7.  } KEYBDINPUT, *PKEYBDINPUT;
```

LISTING 9 - KEYBOARD-INPUT-DATENSTRUKTUR

wVk: stellt eine Taste als (*engl. Virtual Key Code*) wie z.B. 0x0D (Konstante VK_RETURN) für die Enter-Taste dar.[24]

wScan: Falls in **dwFlags** das Flag **KEYEVENTF_UNICODE** gesetzt ist, so stellt **wScan** ein Unicode-Zeichen dar.

Informationen über die weiteren Parameter finden sich in Microsofts MSDN-Dokumentation.[25]

tagMOUSEINPUT repräsentiert Mauseingaben:

```
1.  typedef struct tagMOUSEINPUT {
2.    LONG      dx;
3.    LONG      dy;
4.    DWORD     mouseData;
5.    DWORD     dwFlags;
6.    DWORD     time;
7.    ULONG_PTR dwExtraInfo;
8.  } MOUSEINPUT, *PMOUSEINPUT;
```

LISTING 10 - MAUS-INPUT-DATENSTRUKTUR

dx: Ist die absolute x-Koordinate des Mauscursors ODER die Anzahl der Pixel, um die sich der Cursor seit der letzten Eingabe bewegt hat. Dies ist abhängig wie **dwFlags** gesetzt wird (dazu gleich mehr).

dy: Analog zu **dx** wird hier die Position bzw. die relative Bewegung auf der vertikalen Bildschirmachse beschrieben.

[24] http://msdn.microsoft.com/en-us/library/windows/desktop/dd375731(v=vs.85).aspx
[25] http://msdn.microsoft.com/en-us/library/windows/desktop/ms646271(v=vs.85).aspx

mouseData: Wenn in **dwFlags** das Flag **MOUSEEVENTF_WHEEL** gesetzt ist, so beschreibt **mouseData** die Größe der Veränderung des Mausrads. Ein positiver Wert bedeutet, dass das Rad sich nach oben/vorne bewegt hat und vice versa. Ein Klick mit dem Mausrad wird durch die Konstante **WHEEL_DELTA** beschrieben, welche dem numerischen Wert 120 entspricht. Enthält **dwFlags** keine von den Konstanten **MOUSEEVENTF_WHEEL, MOUSEEVENTF_XDOWN** oder **MOUSEEVENTF_XUP**, dann muss **mouseData** auf 0 gesetzt sein.

Im Falle von **MOUSEEVENTF_XDOWN oder MOUSEEVENTF_XUP** spezifiziert **mouseData** jedoch, welche Maustaste gedrückt wurde (bzw. da SendInput Eingaben generiert, welche Taste gedrückt werden soll). Dafür werden die beiden Konstanten **XBUTTON1 (0x0001)** und **XBUTTON2 (0x0002)** für die beiden Maustasten verwendet.

dwFlags: Hier ist für die in dieser Arbeit beschriebene Applikation **MOUSEEVENTF_ABSOLUTE** wichtig, welches NICHT gesetzt werden darf, damit alle Mausbewegungen als relativ interpretiert werden. Weitere Parameterinformationen sind wiederum in der MSDN-Dokumentation zu finden[26]

Anmerkung: Eine von SendInput erzeugte Eingabe wird immer an das aktuell aktive Fenster gesendet.

[26] http://msdn.microsoft.com/en-us/library/windows/desktop/ms646273(v=vs.85).aspx

4.6.2.1 Tastatur- und Mouse-Hook

Möchte man in der Server-Applikation (=gehooktes Computerspiel) darauf reagieren, wenn serverseitig Maus- und Tastatureingaben entgegengenommen werden so bietet Windows dafür einen speziellen Maus- und Tastatur-Hook, der ohne größere Mühe implementiert werden kann. Dies geschieht in 3 Schritten:

Registrieren des Hooks:

```
1.  g_hKeyboardHook = SetWindowsHookEx(WH_KEYBOARD, KeyboardHook, gl_hThisInstanc
    e, 0);
```

LISTING 11 - KEYBOARDHOOK -FUNKTION

für Tastatur und

```
2.  g_hmousehook = setwindowshookex(wh_mouse, mousehook, gl_hthisinstance, 0);
```

LISTING 12 - MOUSEHOOK -FUNKTION

für Maus-Hooks. Die Parameter *KeyboardHook* und *MouseHook* sind hierbei Callback-Funktionen, die registriert werden. Diese werden dann jedes Mal aufgerufen, wenn eine Maus- oder Tastatureingabe auftritt, sodass diese verarbeitet werden kann:

```
3.  LRESULT CALLBACK KeyboardHook(int Code, WPARAM wParam, LPARAM lParam)
4.  LRESULT CALLBACK MouseHook(int Code, WPARAM wParam, LPARAM lParam)
```

LISTING 13 - MAUS- UND TASTATURHOOK: CALLBACK-FUNKTIONEN

5 Experiment: Quality Of Experience I

Um eine Cloud Gaming-Erfahrung sinnvoll evaluieren zu können, müssen Tests mit menschlichen Benutzern durchgeführt werden, da durch die Kompression und Übertragung, Teile des ursprünglichen Spiels für den Vorteil der Portabilität verloren gehen.

Die Herausforderung beim Design einer Cloud Gaming-Lösung liegt darin, die Kompressionsparameter je nach Gegebenheit, Spieltyp und Benutzer-Preferenzen so anzupassen, dass die Nachteile der Kompression möglichst gering ausfallen.

Deshalb wurde ein Experiment erstellt und durchgeführt, welches dies ermöglichen sollte.

5.1 Parameter

Zunächst wurden die wichtigsten Parameter als Bitrate und FPS (Bilder pro Sekunde) festgestellt, da sich diese maßgeblich auf das Spielerlebnis und gleichzeitig die Bandbreite auswirken: FPS bestimmt wie „ruckartig“ sich ein Spiel bewegt, weniger zu übertragende Bilder pro Sekunde spielen jedoch eine große Rolle bei der zu übertragenden Datenmenge. Die Bitrate hingegen beeinflusst die Datenmenge wiederum stark und bestimmt, wie viel Information pro Pixel beim nach dem En- und Dekodieren verbleibt. Sind zu wenige Informationen enthalten, so müssen sich immer mehr benachbarte Pixel dieselbe Information teilen und es kommt zur Blockbildung:

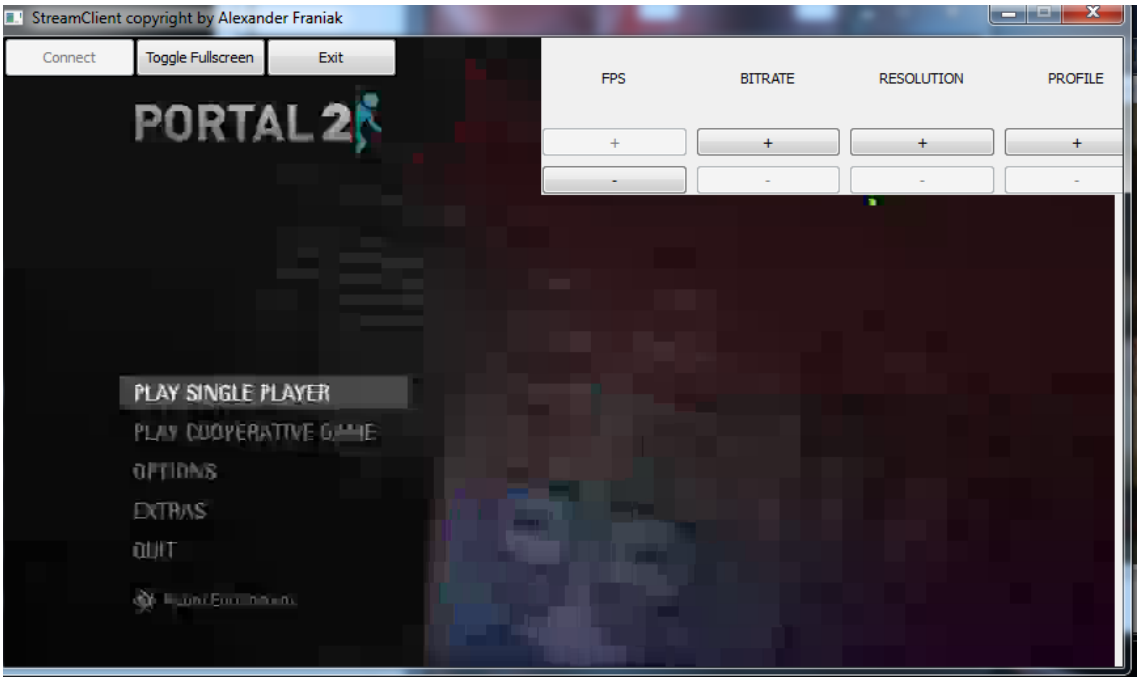

ABBILDUNG 20 - SPIELSZENE MIT SEHR NIEDRIGER BITRATE

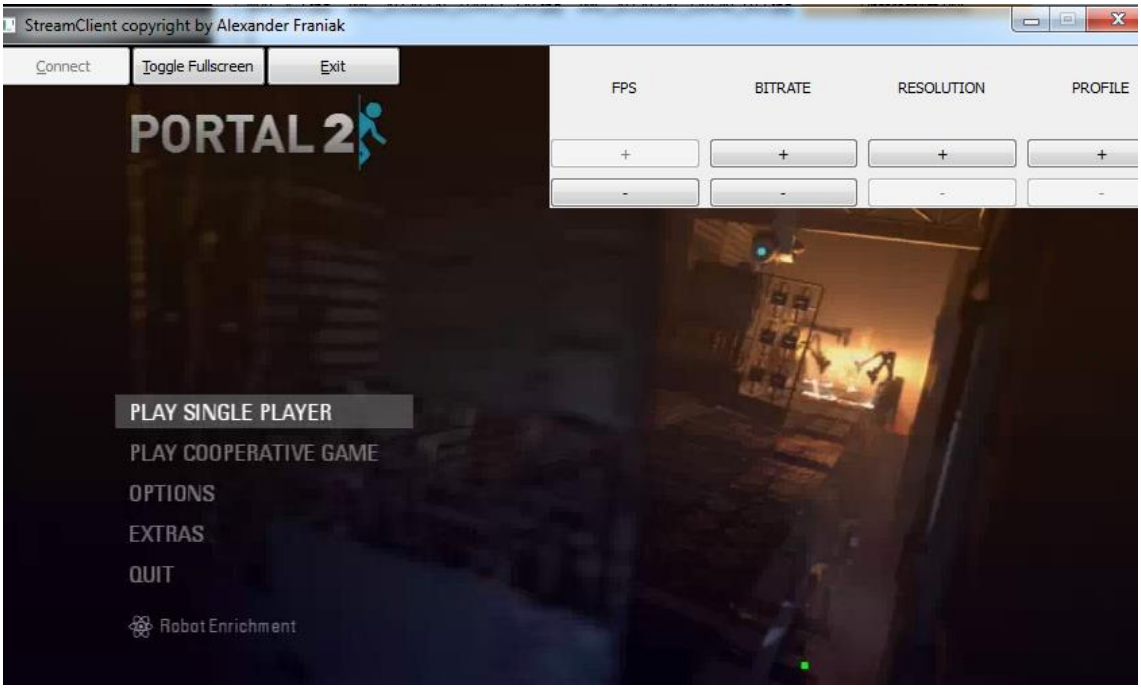

ABBILDUNG 21 - SPIELSZENE MIT HOHER BITRATE

5.2 Spieltyp

Außerdem wurden zwei sehr unterschiedliche Spieltypen getestet: First-Person-Shooter (wird in dieser Arbeit als „Shooter“ anstatt FPS genannt um Verwechslungen zu vermeiden) und Point-And-Click-Adventure („Adventure“). Während man beim First Person Shooter weniger auf Szenendetails achten muss, besitzt das Spiel einen sehr schnellen, flüssigen Ablauf mit viel Bewegungssteuerung und Kameraschwenks. Beim Adventure hingegen wird meist eine Spielfigur über eine vornehmlich statische Szene gesteuert indem man ein Ziel für die Figur markiert und diese sich dann auf dieses zubewegt. Dabei bleibt die Sicht meist statisch, jedoch muss oft der gesamte Bildschirm nach Rätselobjekten und wichtigen Details abgesucht werden.

5.3 Ablauf

Für das Experiment wurden 5 Testpersonen mit unterschiedlicher Erfahrung bei Computerspielen gebeten, nach einer Eingewöhnungsphase von 5 Minuten im jeweiligen Spiel, festzustellen, welche Einstellungen von FPS und Bitrate für welches Spiel die beiden Grenzen die Übergänge von „schlecht“ bzw. „unspielbar“ zu „akzeptabel“ und von „akzeptabel“ zu „gut“ bzw. „ohne Probleme spielbar“ darstellen. Dazu sollen die beiden Parameter abwechselnd zufällig pro Testperson einmal (FPS oder Bitrate) vom höchsten Wert zum niedrigsten heruntergestellt bzw. vom niedrigsten Wert zum höchsten von einem Testaufseher verändert werden. Dazu wurde jeder Testperson pro Spiel vor Beginn zunächst die schlechteste und beste Variante von Bitrate und FPS gezeigt. Als dritter Schritt wurde Bitrate und FPS kombiniert wiederum je nach Testperson zufällig, von unten nach oben oder umgekehrt schrittweise angepasst. Die Anpassungen wurden hierbei immer vom Testleiter verändert und von den Personen Feedback erbeten, um die Person nicht durch die Parameterbedienung vom Spielfluss abzulenken.

Die Wertebereiche waren hierbei für FPS von 5 - 40 Hz, die Bitrate von 160 – 1600 kbps. [1]

5.4 Ergebnisse und Interpretation

Für die Auswertung des Experiments wurden anhand der gewählten Werte aller Teilnehmer der Mittelwert als Diagrammpunkte für die Schwellenwerte „akzeptabel“ und „gut“ sowie die Standardabweichung als Balkenhöhe für FPS und Bitrate berechnet und grafisch veranschaulicht.

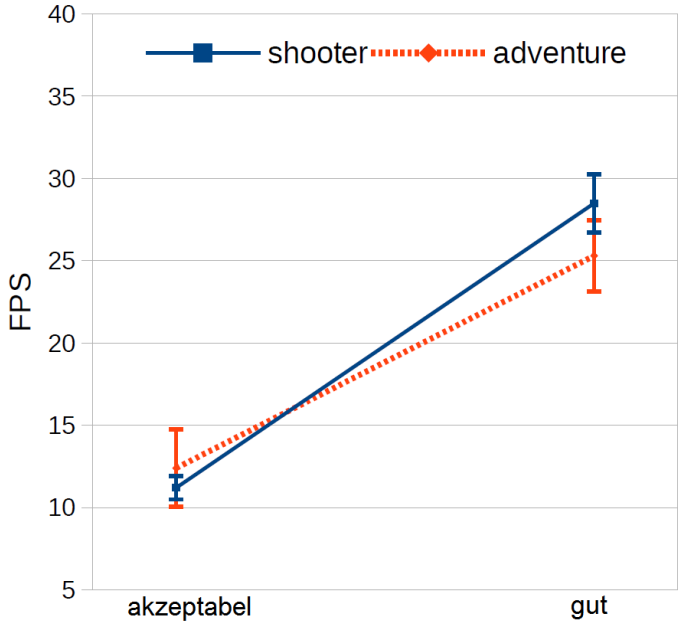

ABBILDUNG 22 - ERGEBNISSE FPS

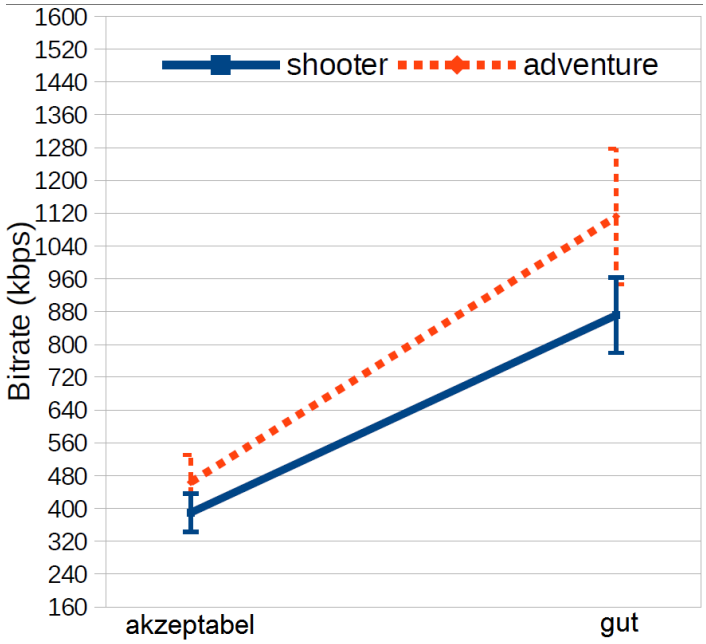

ABBILDUNG 23 - ERGEBNISSE BITRATE

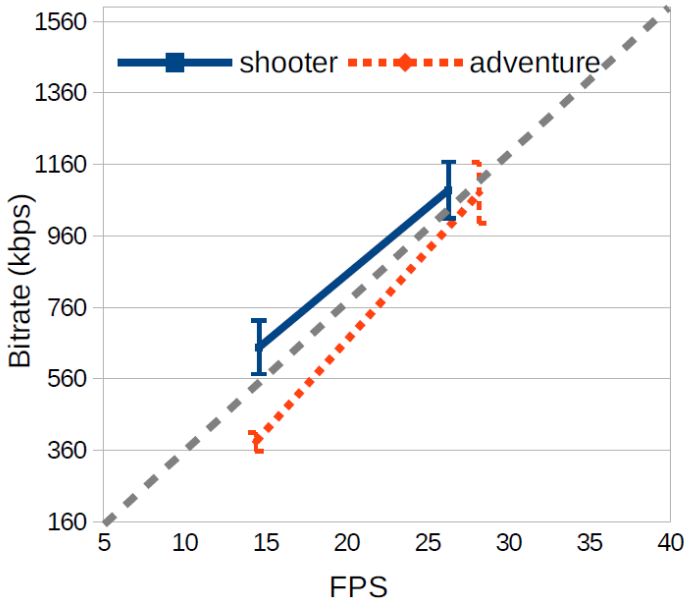

ABBILDUNG 24 - ERGEBNISSE FPS UND BITRATE KOMBINIERT

Wie man unschwer erkennen kann besitzen die beiden Spieltypen sehr unterschiedliche Ansprüche an die Kodierungsparameter. Während beim Shooter die FPS weit höher sein müssen als beim Adventure, verhält es sich bei der Bitrate genau umgekehrt. Adventure-Spiele benötigen eine weit höhere Bitrate. Ist ein Adventure rund um 10 FPS noch spielbar, so trifft das auf den Shooter kaum mehr zu. Beim Shooter liegt die Untergrenze der Bitrate-Toleranz unter der des Adventures. Dies liegt daran, dass bei einem Shooter auch bei sehr „verwaschenem“ Bild bedingt durch die niedrige Bitrate, das Spielerlebnis durchwegs noch passabel sein kann, solange die Framerate in einem höheren Bereich liegt während beim Adventure Rätseldetails bei einer solch niedrigen Bitrate kaum mehr erkennbar sind und das Spiel auch bei einer höheren Bildwiederholrate unspielbar wird. Dafür zeigt sich das Adventure tolerant gegenüber den FPS. Ist ein passabler Wert erreicht so bleibt das Spielerlebnis stabil.

Es ist für eine Cloud Gaming-Lösung also unumgänglich, Wissen über den angebotenen Spieltyp zu erlangen. Dies ist programmatisch nur schwer möglich, da derzeit kein Computerspiel ein „Spieltyp-Tag“ besitzt (dies wäre zukünftig wünschenswert), sodass diese Information vom Benutzer eingegeben werden sollte. Eine weitere Lösung wäre es, eine Spieledatenbank abzufragen, da online der Spieltyp für einen Titel oft verfügbar ist, jedoch fehlt es hier an Standardisierung einer Einzelquelle.

6 Experiment: Quality of Experience II

6.1 Ablauf

Dieses Experiment wurde mit einer höher entwickelten Version der Software durchgeführt. Der Fokus liegt nun eindeutig auf der Handlungsinitiative des Benutzers. Parameter, die das Streaming- und somit das Spielerlebnis beeinflussen, werden nun über eine Benutzeroberfläche/GUI (*engl. Graphical User Interface*) direkt vom Spieler manipuliert solange das Experiment andauert. Der Testaufseher besitzt nun lediglich eine erklärende, beratende und auswertende Rolle.

Während der Benutzer das Spiel steuert und die Parameter beeinflusst, befindet sich der Testaufseher außerhalb des direkten Blickfeldes des Spielers und überwacht den Test auf dem Serverbildschirm.

6.2 Parameter

ABBILDUNG 25 - CLIENT-BILDSCHIRM MIT PARAMETER-GUI

Dieses Mal wird zunächst wieder die Bildwiederholrate beobachtet. Die Testperson kann hierbei mittels Plus- und Minus-Schaltflächen die Bildwiederholrate in einem zuvor durch Experimente fest abgesteckten Bereich verändern. Die Werte betragen 5, 8, 12, 20 und 30 Bilder pro Sekunde (Parameter A).

Der zweite Parameter ist der Qualitätsfaktor. Dieser beeinflusst, wie hoch die Bitrate für ein Einzelbild beträgt. Je geringer die Bitrate desto weniger Farbinformation enthält ein Einzelbild und desto leichter kommt es zur Blockbildung (siehe 2.3.1 Subsampling). Der Wertebereich beträgt hier 63, 40 und 16, wobei 63 für niedrige und 16 für sehr hohe Qualität steht (Parameter B).

Beim dritten Parameter handelt es sich um die Auflösung. Diese ist im Verhältnis 16:9 (Breitbild, *engl. Widescreen*) und kann 854x480 (480p), 960x540 (540p) sowie 1280x720 (HD oder 720p) betragen (Parameter C).

Da das Spiel im Vollbild dargestellt wird, stellt die Veränderung der Auflösung anstatt der Veränderung der Bildgröße, lediglich einen Skalierungsfaktor für das Gesamtbild dar.

Die Bildwiederholrate hat definitiv den größten Einflussfaktor auf die Datenrate da bei mehr Bildern pro Sekunde der Datenfluss sofort um ein Vielfaches ansteigt. Für Spiele mit schneller Reaktion ist eine hohe Bildwiederholrate unumgänglich, ein Minimum von 20 Bildern pro Sekunde ist angebracht. Bei Spielen mit wenig Bewegung und vielen statischen Objekten wiederum kann die Bildwiederholrate geringer gehalten werden, Werte um die 15 Bilder pro Sekunde sind denkbar.

Gleichzeitig ist ein großer Vorteil für das Cloud Gaming-Szenario ersichtlich: Spiele mit viel Bewegung benötigen bei hoher Bildwiederholrate einen vergleichsweise niedrigeren Qualitätsfaktor, da gerade durch die schnellen Bewegungen die Aufmerksamkeit des Spielers auf Umgebungsdetails erheblich verringert wird, somit Bandbreite bei der Bitrate pro Einzelbild eingespart werden kann. Gleichzeitig wird bei einem Adventure eine hohe Bitrate für ein Einzelbild benötigt, um Umgebungsdetails und Rätselobjekte besser erkennen zu können. Durch die geringe Geschwindigkeit des Spielflusses ist es hier als Kompensation jedoch möglich, Bilder pro Sekunde und somit wiederum Bandbreite einzusparen.

6.3 Szenarien

Wie beim letzten Experiment werden zwei verschieden Arten von Spielen getestet. Ein Adventure und ein Shooter (siehe 5.2 Spieltyp). Jedoch wurde diesmal ein weiterer Faktor zum Experiment hinzugefügt: Als Grenze für jegliche Parameterkombination wird eine maximale Bitrate angesetzt. Diese dient als Spielraum bzw. Grenze für die gewählte Qualität, da die Erhöhung der Parameter die zu übertragende Menge an Daten erhöht, diese jedoch hat in der Realität bei tatsächlich vorhandener Datenleitung Kapazitätsgrenzen.

Es wurden zwei Grenzen bestimmt. Die höhere Grenze beträgt 2000kbps = 2MBit/s („HIGH“) während die niedrigere Grenze bei 750 kbps liegt („LOW“), was ungefähr einer besseren und schlechteren ADSL-Leitung entspricht.

Der Spieler soll hierbei auf ein Kontrolllicht-Symbol achten, das bei zu hohen Einstellungen rot wird bzw. rot blinkt oder bei viel zu hohen Anforderungen während des Spielablaufs rot bleibt. Ziel ist es, dass der Benutzer selbsttätig eine Kombination von Parametern findet, bei der einerseits sein Spielerlebnis das möglichst subjektiv Beste ist während gleichzeitig das Kontrolllicht grün bleibt bzw. nicht länger als eine halbe Sekunde auf die Farbe Rot wechselt und dies nicht öfter als alle 20 Sekunden passiert (siehe Abb. 25).

6.4 Ergebnisse und Interpretation

Die Mittelwerte der gewählten Parameter aller Teilnehmer werden in Form der folgenden beiden Diagramme veranschaulicht. Die Werte stellen jeweils den Anteil eines Parameterwerts am Maximalwert dar.

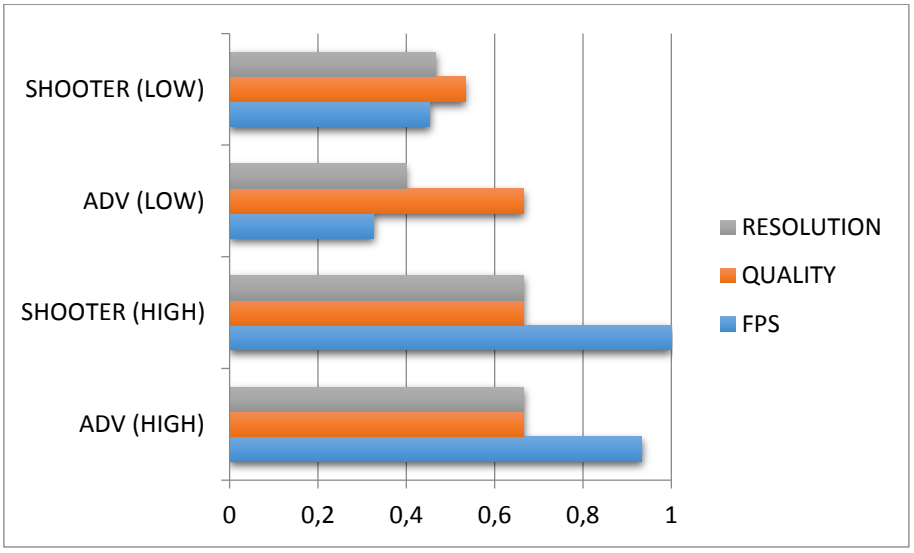

ABBILDUNG 26 - EINZELBETRACHTUNG ALS SÄULENDIAGRAMM

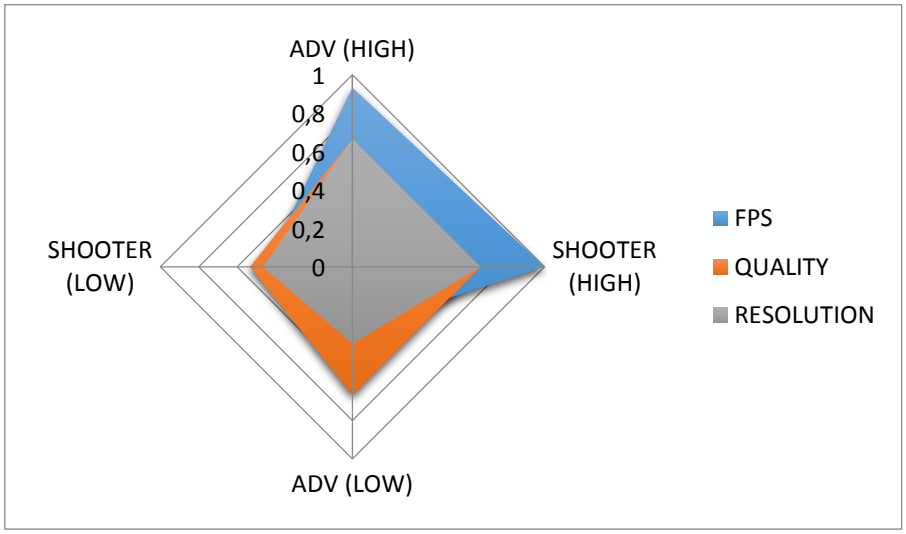

ABBILDUNG 27 - BETRACHTUNG ALS SPIDERWEB IM GESAMTKONTEXT

Zunächst ist als Auffälligkeit zu bemerken, dass nahezu keine der Testpersonen, bei keinem der beiden Spiele oder Szenarios bereit war, den Qualitätsfaktor mehr als eine Stufe unter dem Limit einzustellen (der Wert im Diagramm ist für jedes Szenario gleich und beträgt eine Stufe unter dem Maximum). Das absolute Maximum hingegen war nicht notwendig. Jedoch wurde aus den nachfolgenden Interviews mit den Testpersonen ersichtlich, dass ein niedrigerer Wert sofort einen Enttäuschungseffekt beim Spieler auslöst, da direkt und schnell eine deutliche Qualitätsverringerung bemerkbar war, während eine geringere Bildwiederholrate erst nach dem Verändern des Parameters beim Weiterspielen bemerkbar wurde. Lediglich bei geringer Bandbreite und der sich daraus ergebenden höheren Kompromissbereitschaft (die Testpersonen wurden in den Ablauf soweit eingeführt, dass sie erkannten, dass sie Abstriche bei gewissen Parametern machen mussten, um das gegebene Ziel zu erreichen, nämlich ein grünes Kontrolllicht) konnte der Shooter auch mit etwas niedrigerem Qualitätsfaktor jedoch bei vergleichsweise hoher Bildwiederholrate zur Zufriedenheit des Testers gespielt werden.

Es zeichnet sich außerdem sehr deutlich ab, dass die Bildwiederholrate beim Spielen eines Shooters um ein Vielfaches höher liegen muss, um ein flüssiges Spielerlebnis zu ermöglichen während die Auflösung nahezu nicht von Bedeutung ist, da sie im Vollbildmodus eine geringe Skalierung des Bildes darstellt.

7 Literaturverzeichnis

[1] A. Franiak, H. Hlavacs, Y. Pitrey und C. Czepa, „Streaming DirectX-Based Games on Windows," in *Streaming DirectX-Based Games on Windows*, Kathmandu, 2012.

[2] M. Venables, „http://www.wired.com," 6 Februar 2011. [Online]. Available: http://www.wired.com/geekdad/2011/06/review-onlive-the-what-why-and-who-of-gaming-in-the-cloud/. [Zugriff am 12 April 2013].

[3] E. Caoili, „Gamasutra," 17 August 2012. [Online]. Available: http://www.gamasutra.com/view/news/176180/OnLive_lays_off_all_employees.php. [Zugriff am 14 März 2013].

[4] S. Hollister, „http://www.theverge.com," 30 Juni 2012. [Online]. Available: http://www.theverge.com/2012/6/30/3127602/gaikai-google-nacl-native-client. [Zugriff am 12 April 2013].

[5] A. Gallegos, „http://www.ign.com," 20 Februar 2013. [Online]. Available: http://www.ign.com/articles/2013/02/20/playstation-cloud-revealed. [Zugriff am 5 April 2013].

[6] T. Geron, „http://www.forbes.com," 7 Februar 2012. [Online]. Available: http://www.forbes.com/sites/tomiogeron/2012/07/02/sony-to-acquire-cloud-gaming-startup-gaikai-for-380-million/. [Zugriff am 12 April 2013].

[7] J. Fingas, „http://www.engadget.com," 3 Mai 2012. [Online]. Available: http://www.engadget.com/2012/05/03/wikipad-android-gaming-tablet-adds-gaikai/. [Zugriff am 12 April 2013].

[8] D. Gibbon, „http://www.digitalspy.co.uk," 26 Oktober 2007. [Online]. Available: http://www.digitalspy.co.uk/gaming/news/a78532/ps3-to-allow-pc-games-to-run-on-it.html. [Zugriff am 12 April 2013].

[9] C. Klaß, „http://www.golem.de," 26 Oktober 2007. [Online]. Available: http://www.golem.de/0710/55653.html. [Zugriff am 12 April 2013].

[10] L. Chan, „http://www.neoseeker.com,“ 14 Jänner 2008. [Online]. Available: http://www.neoseeker.com/news/7525-play-directx10-pc-games-on-ps3-over-network-or-internetr/. [Zugriff am 12 April 2013].

[11] C. Poynton, Digital Video and HD, Waltham MA 02451, USA: Morgan Kaufmann, 2012.

[12] „en.wikipedia.org,“ 18 Juli 2010. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/thumb/f/f2/Common_chroma_subsampling_ratios.svg/520px-Common_chroma_subsampling_ratios.svg.png. [Zugriff am 7 Mai 2013].

[13] „msdn.microsoft.com,“ 28 November 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/bb530104%28v=vs.85%29.aspx. [Zugriff am 7 Mai 2013].

[14] „en.wikipedia.org,“ 19 Oktober 2010. [Online]. Available: http://upload.wikimedia.org/wikipedia/en/0/06/Colorcomp.jpg. [Zugriff am 7 Mai 2013].

[15] P. W. Y. X. Jim Bankoski, „TECHNICAL OVERVIEW OF VP8, AN OPEN SOURCE VIDEO CODEC FOR THE WEB,“ Google Inc. 1600 Amphitheatre Parkway, Mountain View, CA, USA.

[16] G. J. S. G. B. A. L. Thomas Wiegand, „Overview of the H.264/AVC Video Coding Standard,“ *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY,* pp. Vol. 13, Issue 7, 7 Juli 2003.

[17] http://www.on2.com, 22 Oktober 2008. [Online]. Available: http://web.archive.org/web/20081022154547/http://www.on2.com/index.php?602. [Zugriff am 10 Mai 2013].

[18] J. K. L. Q. J. S. P. W. Y. X. J. Bankoski, „VP8 Data Format and Decoding Guide,“ Google Inc. 1600 Amphitheatre Parkway, Mountain View, CA, USA, November 2011.

[19] „en.wikipedia.org,“ 6 Oktober 2010. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/thumb/e/e2/GOP_2.svg/1000px-GOP_2.svg.png. [Zugriff am 8 Mai 2013].

[20] „en.wikipedia.org,“ 30 September 2009. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/thumb/6/64/I_P_and_B_frames.svg/1000px-I_P_and_B_frames.svg.png. [Zugriff am 8 Mai 2013].

[21] B. Müller, „www.netzwelt.de,“ 5 Juni 2005. [Online]. Available: http://www.netzwelt.de/news/71398_4-know-how-mpeg-grundlagen.html. [Zugriff am 10 Mai 2013].

[22] J. Bremer, „http://jbremer.org,“ 2 Juli 2012. [Online]. Available: http://jbremer.org/x86-api-hooking-demystified/. [Zugriff am 10 Mai 2013].

[23] I. Ivanov, „www.codeproject.com,“ 2 Dezember 2002. [Online]. Available: http://www.codeproject.com/Articles/2082/API-hooking-revealed. [Zugriff am 16 Mai 2013].

[24] S. Grüner, „www.golem.de,“ 24 März 2011. [Online]. Available: http://www.golem.de/1103/82321.html. [Zugriff am 23 Juni 2013].

[25] „www.ffmpeg.org,“ [Online]. Available: http://www.ffmpeg.org/about.html. [Zugriff am 23 Juni 2013].

[26] A. L. Raffael Vötter, „www.pcgameshardware.de,“ 9 November 2012. [Online]. Available: http://www.pcgameshardware.de/Spiele-Thema-239104/Specials/Wann-laufen-Spiele-fluessig-1034704/. [Zugriff am 23 Juni 2013].

[27] D. R. Gerhard Krüger, Telematik, Netze-Dienste-Protokolle, Deutschland, Leipzig: Fachbuchverlag Leipzig, 2002.

[28] „en.wikipedia.org,“ 16 November 2006. [Online]. Available: http://upload.wikimedia.org/wikibooks/en/d/d9/Header_of_UDP.jpg. [Zugriff am 23 Juni 2013].

[29] en.wikipedia.org, „en.wikipedia.org,“ 6 Juli 2007. [Online]. Available: http://upload.wikimedia.org/wikipedia/de/thumb/f/fd/TCP_Header.svg/1000px-TCP_Header.svg.png. [Zugriff am 23 Juni 2013].

[30] A. Müller, „http://andremueller.gmxhome.de,“ 12 August 2003. [Online]. Available:

0] http://andremueller.gmxhome.de/toc.html. [Zugriff am 11 Mai 2013].

[31] A. B. M. L. A. J. D. E. Adam Ferrari, „http://www.cs.virginia.edu,“ University of Virginia Computer Science, 13 Februar 2006-2013. [Online]. Available: http://www.cs.virginia.edu/~evans/cs216/guides/x86.html. [Zugriff am 16 Mai 2013].

[32] K. Amerasinghe, „H.264 FOR THE REST OF US,“ 2009.

7.1 Abbildungsverzeichnis

7.2 Tabellenverzeichnis

7.3 Listingverzeichnis

8 Anhang

8.1 Zusammenfassung

In der vorliegenden Masterarbeit wurde eine Einführung in das Gebiet „Cloud Gaming“ der Informatik gegeben. Die Herausforderungen und Problemstellungen dieser neuartigen Technologie, die zum Zeitpunkt der Erstellung dieser Arbeit nirgendwo marktreif kommerziell angeboten wird, wurden beschrieben und ein Lösungsvorschlag für die Implementierung eines Cloud Gaming-Frameworks angeboten, wobei die Gewichtung auf Portabilität, Modularität und die Verwendung von ausschließlich freier Software gelegt wurde.

Die vielfältigen Mechanismen, welche für eine solche Lösung zum Einsatz kommen wurden umfassend beschrieben und ihren Fachgebieten der Informatik bestehen aus Netzwerk, Videokodierung, Dll-Hooking, Eingabeverarbeitung und Rendering zugeordnet.

Die Implementierung der im Rahmen der vorliegenden Masterarbeit erstellten Softwarelösung wurde durch eine Beschreibung aller Module und deren Innenleben beschrieben. Dabei wurde im speziellen darauf geachtet, alle kritischen Herausforderungen durch Listings zu unterlegen und Verbindungen zu den Themengruppen, die diese betreffen, herzustellen.

Die Thematik wurde anschließend einem Praxistest in Form zweier Benutzerexperimente unterzogen und die Ergebnisse in ansprechbarer Form dargestellt und interpretiert.

Eine Cloud Gaming Lösung ist somit machbar, jedoch müssen Kompromisse hinsichtlich Bandbreite und Enkodierungsqualität getroffen werden. Auf diese Art Computerspiele zu erleben stellt im Zeitalter der digitalen Distribution und des On-Demand-Entertainments mit Sicherheit eine zukunftsweisende Perspektive dar.

8.2 Akademischer Lebenslauf

Persönliche Angaben:

Name: Alexander Franiak

Geburtsdatum: 30.09.1985

Geburtsort: Wien

Familienstand: ledig

Ausbildung:

1992-1994: VS 28 Linz

1994-1996: Volksschule Kirchberg-Thening

1996-2000: Bundesrealgymnasium Landwiedstraße Linz

2000-2005: HTL Leonding

2006-2009: Bachelor Medieninformatik – Universität Wien

2010-*: Master Medieninformatik - Universität Wien

Berufslaufbahn:

2004: KEBA AG, Systementwicklung

2006: Landesnervenklinik Wagner-Jauregg, Zivildienst

2007: Voestalpine Stahl Service Center, Produktion

2008-2011: webfreetv.com Multimedia Dienstleistungs AG, Java Softwareentwickler

2011-*: T-Mobile Austria GmbH, Java Softwareentwickler