

2016

Verbesserung des HEVC Codes mit Hilfe von Gesichtserkennung

MASTER PROJEKTARBEIT
VIRGEL DEVIL

Inhaltsverzeichnis

Inhaltsverzeichnis.....	1
Abbildungsverzeichnis.....	3
I. Einleitung.....	5
1. Fernseh-Technologie im Zeitraum von 10 Jahren	5
2. Größenproblem	6
3. Gliederung.....	6
II. OpenCV.....	7
1. Installation OpenCV für C#	8
2. Beispiele Kompilieren	9
3. Gesicht Erkennung (Cascade Classifier Training)	12
4. Gesicht Erkennung mit Video	13
III. HEVC/H.265 Codec.....	14
1. GOP	15
2. Slice und Coding Unit:.....	15
3. Profile	16
4. Level.....	17
5. Installation	19
6. HEVC Parameter	20
Input File	20
Encoder Parameter.....	20
Decoder Parameter.....	21
7. HEVC Vergleich.....	22
Quantisierung	22
Videoformat	23
Vergleich zur anderen Codec Formate	24
IV. Verbesserung von HEVC mit Hilfe von OpenCV.....	27
1. Schnittstelle:	27
2. HEVC Code Struktur:.....	27
3. Kode für Schnittstelle:.....	28
4. Quantisierungsdatei:	30
5. Gesichtserkennungsprogramm:	31
6. Die Neue Parameter:	32

7. Lambdavariable	32
8. Vergleich.....	33
Zusammenfassung	34
Quellenverzeichnis.....	39
Anhang.....	40
HEVC Klassendiagramm	40
AUFRUF DIAGRAMM	43

I. Abbildungsverzeichnis

Abbildung 1 Fernseh-Technologie im Zeitraum von 10 Jahren	5
Abbildung 2 OpenCV Logo	7
Abbildung 3 Emgu Download	8
Abbildung 4 Emgu Download	9
Abbildung 5 Camera Capture	10
Abbildung 6 MotionDetection	10
Abbildung 7 Simple3DReconstruction	10
Abbildung 8 Gesicht Erkennung mit Lena	11
Abbildung 9 Grund Klassifizierter	12
Abbildung 10 Training.....	12
Abbildung 11 Face Detection + WebCam	13
Abbildung 12 HEVC Schema	14
Abbildung 13 GOP Struktur.....	15
Abbildung 14 Slice und CTU Beispielverteilung	15
Abbildung 15 Blockgrößenverteilung	16
Abbildung 16 Video Encoding (Ausschnitt).....	19
Abbildung 17 YUV Speicherung	20
Abbildung 18 YUV RAW	22
Abbildung 19 Quantisierung: 0	22
Abbildung 20 Quantisierung: 51	22
Abbildung 21 1080p Komprimiert	23
Abbildung 22 480p Komprimiert	23
Abbildung 23 Original 480p.....	23
Abbildung 24 HEVC 1080p	23
Abbildung 25 HEVC 480p	23
Abbildung 26 H.265.....	24
Abbildung 27 H.264.....	24
Abbildung 28 MPEG2	24
Abbildung 29 Xvid.....	24
Abbildung 30 H.265.....	25
Abbildung 31 H.264.....	25
Abbildung 32 MPEG2	25
Abbildung 33 Xvid.....	25
Abbildung 34 H.265.....	26
Abbildung 35 H.264.....	26
Abbildung 36 MPEG2	26
Abbildung 37 Xvid.....	26
Abbildung 38 HEVC Klassendiagramm (Gesamtübersicht, Großes Bild im Anhang)	27
Abbildung 39 Gesichtserkennungsprogramm	31
Abbildung 40 HEVC mit Gesichtserkennung	33
Abbildung 42 Quantisierung: 51	33

Abbildung 41 Quantisierung: 0	33
Abbildung 43 Gesichtserkennungsprogramm	37
Abbildung 44 Erstellung von YUV und Einstellungsdatei in dem Gesichtserkennungsprogramm	38
Abbildung 45 Speicherung des Bereiches in die Datei	38

II. Einleitung

1. Fernseh-Technologie im Zeitraum von 10 Jahren

In den letzten 10 Jahren hatten wir einen starken Technologieaufstieg (Abbildung 1). Früher hatte man nur einen großen Fernseher mit kleinem Bild mit der QVGA (320×240) Auflösung.

Heutzutage haben wir schon Flachbildschirme, wo die Fernseher fast so groß sind wie das eigene Bild in ihm und haben dabei eine 4K (4800×7680) Auflösung.

Außer der Auflösung der Bildqualität zum Fernseher wurden neue Funktionen eingefügt. Die Farbe als Option wurde erst im 60 – 70er Jahren eingeführt damals dachte man dass es schon die Grenze von dem was man mit dem Fernseher machen könnte.

Heute gibt es schon 3D Fernseher die mit der Hilfe von moderner Technik dem Zuschauer das Gefühl geben, dass er sich in der Scene befindet.

Außerdem haben moderne Fernseher ein Internetzugang, was ermöglicht dem User z.B. die Skype Kommunikation auf dem Fernseher.

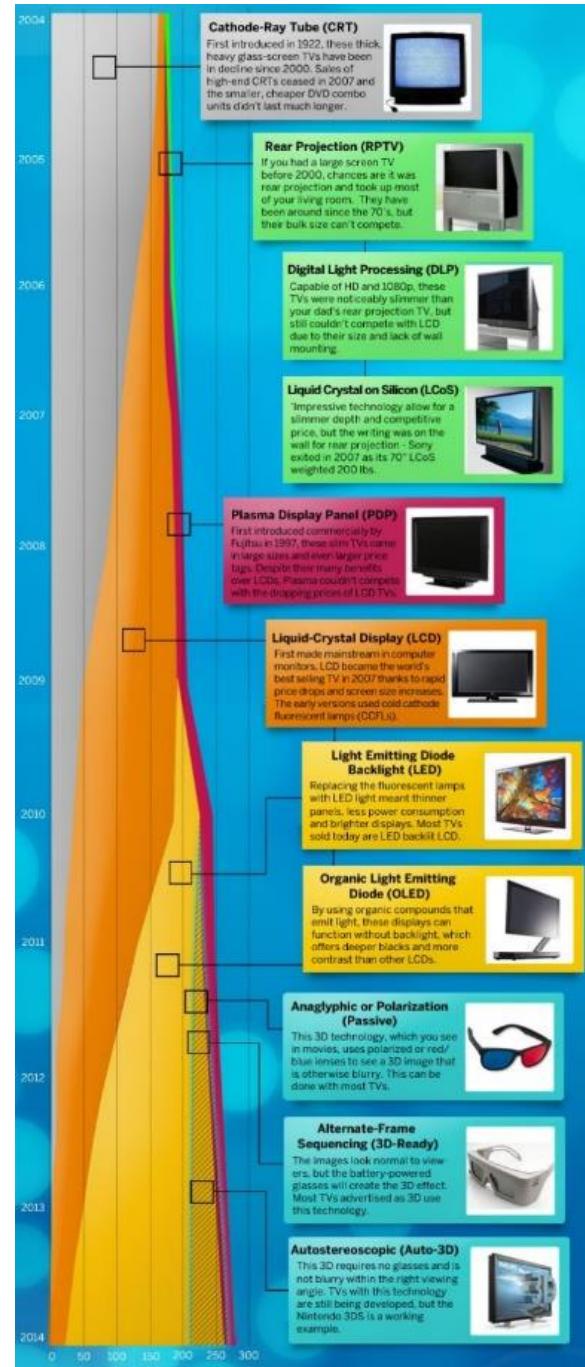


Abbildung 1 Fernseh-Technologie im Zeitraum von 10 Jahren

2. Größenproblem

Das alles ist aber noch keine Technologiegrenze und es kommen noch viel mehr Neuheiten auf den Markt. Leider vergrößert sich durch diese Änderungen auch die Größe von dem Video selbst. Dadurch kann man das Video im alten Format nicht mehr speichern oder auch das Video übertragen.

Um das Problem zu umgehen muss man neue Komprimierungsformate entwickeln. Das Neuste davon ist der HEVC Codec. Er ermöglicht die Komprimierung von bis zu 70% im Vergleich zur MPEG2 Codec. In dieser Arbeit wird versucht die Komprimierung mit Hilfe von Bildverbreitungsbibliothek OpenCV zu verbessern.

3. Gliederung

Das folgende Kapitel beschreibt die OpenCV Bibliothek. Hier wird erklärt wie man die Bibliothek installiert und an die C# Sprache anbindet. Dann werden auch Beispiele erklärt. Anschließend wird die Gesicht Erkennung beschrieben.

Im Drittem Kapitel wird der das HEVC Codec beschrieben. Es werden die wichtigsten Begriffe erklärt und Unterschiede zwischen den Profilen und Level gezeigt. Außerdem wird hier das Insolationsvorgehen erklärt und die Parameter veranschaulich. Am Ende wird das HEVC mit anderen Codecs verglichen.

Das Letzte Kapitel behandelt Zusammenbildung von HEVC und OPENCV. Durch die Erklärung von dem Code wird besseres Überblick über das Vorgehen erschafft. Es wird das Vorgehen bei der Programmierung erklärt und die Einstellungen für die neue Funktion gezeigt. Schließlich wird das neues Codec mit dem vorhandenen HEVC verglichen.

III. OpenCV

Open Source Computer Vision Library (OpenCV) (Abbildung 2) ist eine freie Programmbibliothek mit Algorithmen für die Bildverarbeitung und maschinelles Sehen, die von Intel und Willow Garage entwickelt wurde. Die Bibliothek wurde in C und C++ entwickelt, es gibt aber andere Implementierungen für C#, Python, Java, Ruby, Matlab, Lua und andere Sprachen. Die Bibliothek umfasst die Algorithmen für Gesichtsdetektion, 3D-Funktionalität, Haar-Klassifikatoren, verschiedene sehr schnelle Filter (z. B. Sobel, Canny, Gauß) und Funktionen für die Kamerakalibrierung.



Abbildung 2
OpenCV Logo

OpenCV besteht aus Modulen für verschiedene Anwendungsfelder:

- 2D- und 3D-Merkmale (z. B. Interest-Operator oder Deskriptoren)
- Eigenbewegungsschätzung, siehe Photogrammetrie: Rückwärtsschnitt
- Gesichtserkennung
- Gestenerkennung
- Mensch-Computer-Interaktion (HCI)
- mobile Roboter
- Objekterkennung
- Segmentierung und Erkennung
- Stereoskopisches Sehen (Stereopsis), ergibt Tiefenbilder
- Structure from motion (SfM), siehe Photogrammetrie
- optisches Tracking, Motion Compensation und Optischer Fluss
- Kalman-Filter zum Tracking

Ferner beinhaltet OpenCV eine Bibliothek für Maschinelles Lernen mit folgendem Funktionsumfang:

- Boosting (automatische Klassifizierung)
- Lernen eines Entscheidungsbaumes
- EM-Algorithmus (Expectation-Maximization)
- Nächste-Nachbarn-Klassifikation
- Bayes-Klassifikator
- Künstliche neuronale Netze
- Random Forest
- Support Vector Machine (SVM)

1. Installation OpenCV für C#

Für C# Gibt es eine Bibliothek namens Emgu CV. Die verbindet das OpenCV mit C#.

Auf der Seite http://www.emgu.com/wiki/index.php/Download_And_Installation findet man Einleitung zur Installation der Bibliothek.

Die Installation findet man auf der Seite:

http://www.emgu.com/wiki/index.php/Main_Page

Installation auf Windows (x64):

Auf der Hauptseite (Abbildung 3) steht immer das neuste Relise von der Emgu. Man muss einfach auf den neusten Sourceforge link klicken, dann wird man auf die Seite mit dem Link <https://sourceforge.net/projects/emgucv/> weitergeleitet.

The screenshot shows the Emgu CV Main Page on sourceforge.net. The page has a header with the Emgu logo, a navigation bar with links like 'Main Page', 'Tutorial', 'API Documentation', 'Version History', 'Download and Installation', 'Support and Services', 'Discussion Forum', 'Bug Tracking', 'Code Gallery', 'Licensing', 'Others', 'GIT', 'SourceForge', and 'Recent Changes'. On the left, there's a sidebar with 'Tools' and 'What links here'. The main content area has a 'Contents [hide]' sidebar with sections like 'Latest News', 'Platform Features' (with sub-sections for Windows, Mobile Devices, OS X, Linux, Unix), 'Advantage of Emgu CV' (with sub-sections for Cross Platform, Cross Language, and Other Advantages), and 'Architecture Overview'. Below this is a 'Latest News' section with a list of items. A large red arrow points from the word 'Download' in the text above to the green 'Download' button in the sidebar.

Abbildung 3 Emgu Download

Auf sourceforge Seite (Abbildung 4) muss man den grünen Knopf drücken um den Download zu starten.

Nach dem die Installationsdatei geladen wurde, installiert man diese mit einem Doppelklick. Bei der Installation muss man meistens auf weiter drücken.

Die Installationsdatei entpackt den Ordner „emgucv-windows-universal“ auf das Laufwerk. Der Ordner enthält die Beispieldaten, OpenCV Bibliothek, Quellcode für Beispieldaten und den Bibliotheken.



Abbildung 4 Emgu Download

2. Beispiele Kompilieren

Im Ordner „emgucv-windows-universal 3.0.0.2157\bin“ liegen schon die vorkompilierten Beispiele. Es enthält aber auch für viele Projekte wichtige DLL's, Daten und Beispielbilder wie z.B. ein meist genutztes Foto von Lena. Es gibt eine Übersicht zu den Beispielprojekten.

Im Ordner „emgucv-windows-universal 3.0.0.2157\Solution\VS2010-2015“ liegt die Solution für Visual Studio. In dieser Solution liegen alle Beispielprojekte die als sehr gute einstig Quelle dienen.

Man kann aber auch die Projekte aus dem Ordner „emgucv-windows-universal 3.0.0.2157\Emgu.CV.Example“ in eigene Projekt kopieren. Es hilft vor allem wenn das Beispiel fast das macht was man für ein eigenes Projekt braucht. Man muss aber beachten, dass für manche Funktionen brauchen bestimmte DLL's ins Projekt zu kopieren.

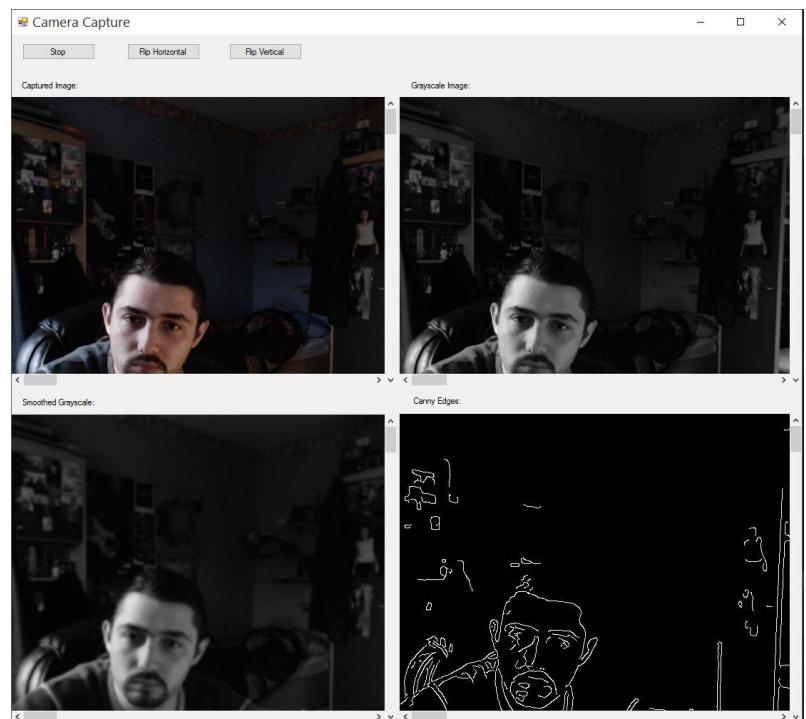
Hier werden die für Projekt interessante Beispiele aufgelistet:

Camera Capture:

Dieses Beispiel (Abbildung 5) zeigt wie man mit OpenCV die Kamera ansteuert. Es zeigt auch wie man auf das aufgenommenes Bild Filter ansetzt.

Dieses Beispiel benutzt die DLL's aus dem Ordner „emgucv-windows-universal 3.0.0.2157\bin“, deswegen muss man diese beiden Ordner unbedingt in das debug Verzeichnis kopieren.

Abbildung 5 Camera Capture



Motion Detection:

Dieses Beispiel (Abbildung 6) unterteilt sich in zwei OpenCV Funktionen. Als erstes wird die Ansteuerung der Kamera was auch im obigen Beispiel zu erkennen und das zweite ist Bewegungserkennung.

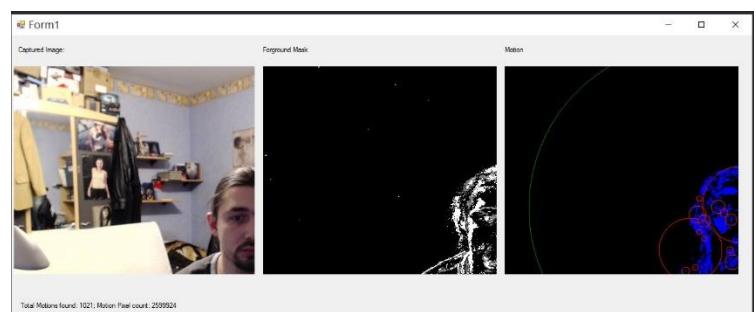


Abbildung 6 MotionDetection

Simple3DReconstruction:

Dieses Beispiel (Abbildung 7) zeigt 2 Funktionen Rechts wird ein rekonstruiertes 3D Model gezeigt, dass aus 2 entstand Bilder bestehen. Es Benutzt das gleiche Prinzip wie 3D für menschliche Augen.

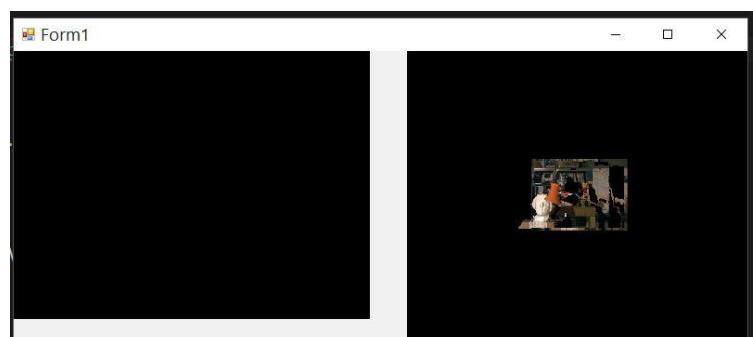


Abbildung 7 Simple3DReconstruction

In das linke Fenster kann man ein Bild laden. Die rechte Maustaste gibt eine Liste mit Filtern und Effekten, die man auf das geladene Bild anwenden könnte.

Gesicht Erkennung:

In dem Beispiel Ordner liegt auch ein Beispiel für die Gesicht und Augen Erkennung (Abbildung 8). Es Zeigt ein Beispiel mit dem Bild von Lena.

Es benutzt den Cascade Classifier um den Gesicht zu finden.

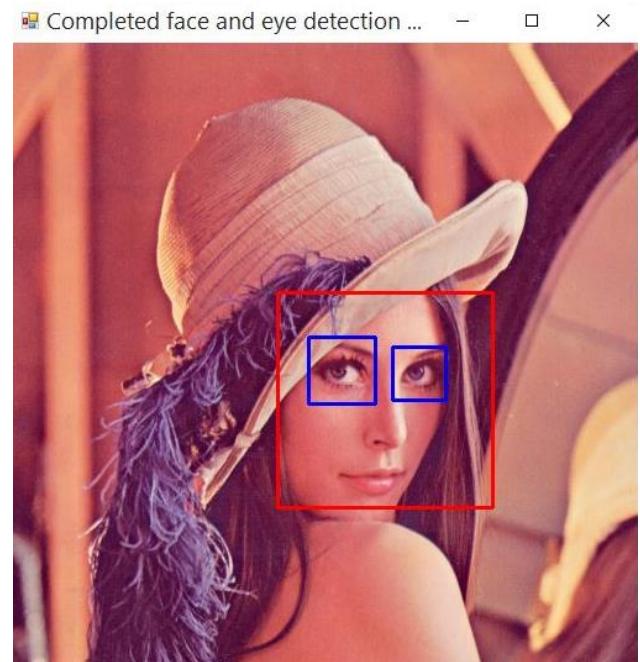


Abbildung 8 Gesicht Erkennung mit Lena

3. Gesicht Erkennung (Cascade Classifier Training)

OpenCV benutzt Haar Detektion (Abbildung 9) um Gesicht zu erkennen. Am Anfang wird die Haar Datei durch Training (Abbildung 10) hergestellt.

Die Datei wird mit 1000 Bildern von dem gesuchten Objekt und 1000 Bildern ohne Objekt hergestellt.

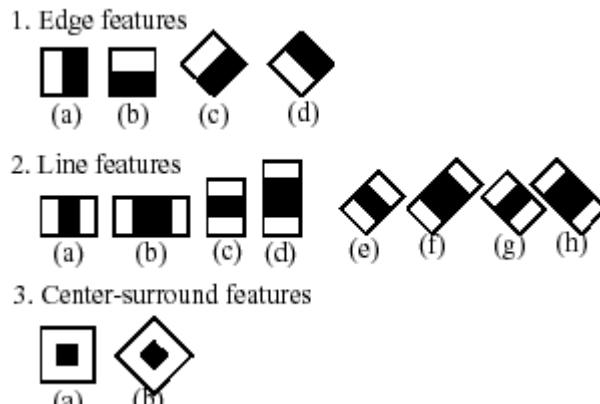


Abbildung 9 Grund Klassifizierter

Nachdem ein Klassifizierer trainiert ist, kann es zu einem in einem Eingabebild angewendet werden.

Der Klassifizierer gibt eine 1 bei einem wahrscheinlicher Region, und eine 0 sonst. Um den Objekt zu finden wird das Suchfenster über das gesamte Bild bewegt und nach Klassifizierern gesucht (wie in Abbildung 10). Bei der Suche nach einem Objekt mit unbekannter Größe muss man den Scan-Vorgang mehrmals wiederholen.

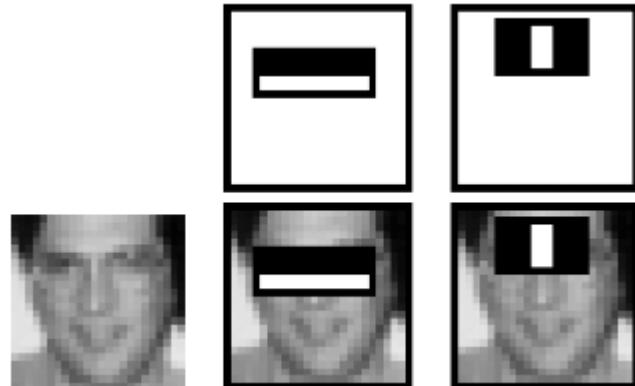


Abbildung 10 Training

Weil die resultierende Klassifizierer bestehen aus mehreren einfacher Klassifizierer (Stufen), die bis zu einem bestimmten Zeitpunkt die Kandidaten zurückgewiesen oder alle Stufen geleitet werden anschließend zu einer Region von Interesse angewendet werden wird es als Cascaden Funktion bezeichnet.

4. Gesicht Erkennung mit Video

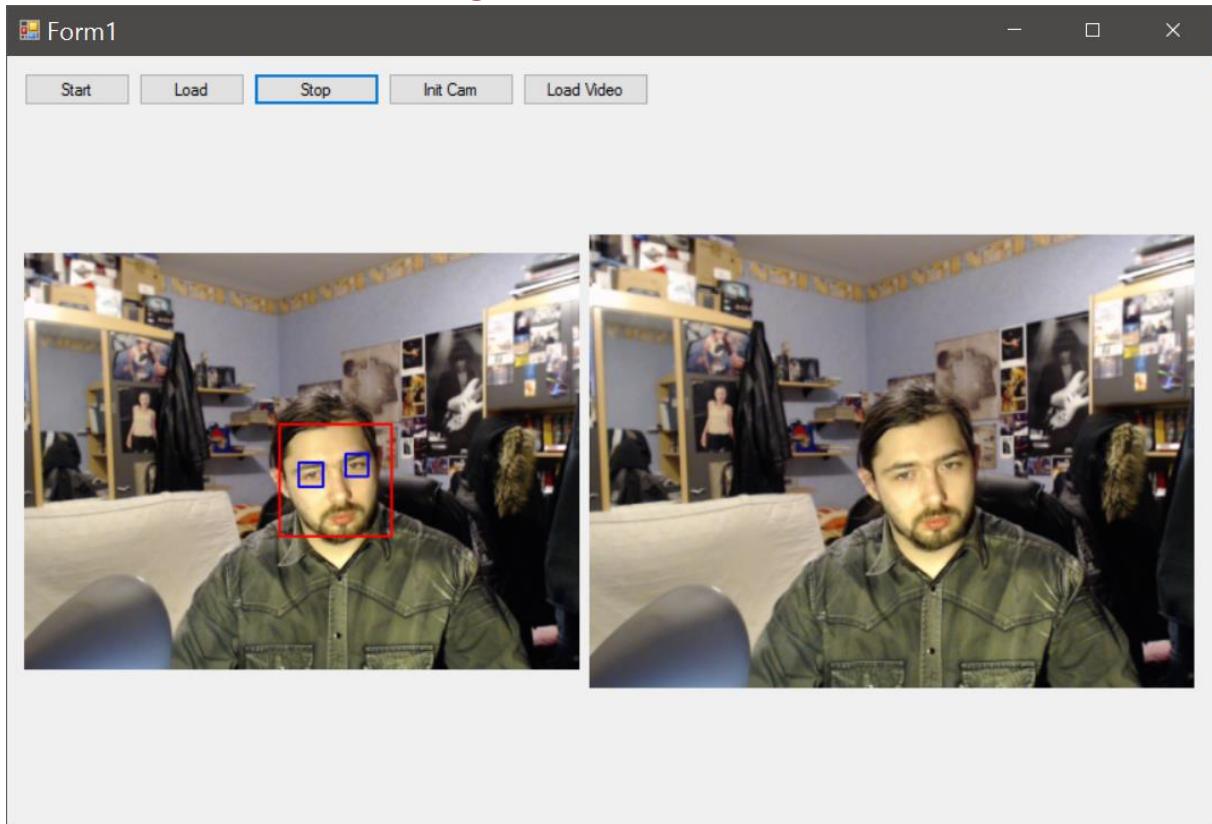


Abbildung 11 Face Detection + WebCam

Für unseres Programm (Abbildung 11) brauchen wir die Haar Datei. Wir können die durch die Cascade Classifier Training herstellen oder wir benutzen die fertige Datei aus dem Beispielprojekt „FaceDetection“ (mit Lena).

OpenCV hat die Capture Klasse, die ermöglicht die Arbeit mit einem Video, sei es ein Videofile oder ein Videostream von der Webcam. Leider arbeitet es nur mit FFmpeg Codec-Pack, was aber noch kein h265 und kein YUV Kodierung enthält, deswegen muss für diese beiden Formate eine eigene Capture Klasse programmiert werden. Mit dem „ImageGrabbed“ Event geht es zu nächstem Frame über. Die Funktion „Retrieve“ speichert den Frame in die Mat (Matrix) Variable.

Für die Gesichts Erkennung wird das CudaCascadeClassifier benutzt. Als Klassifizierer benutzen wir die Haar Datei die wir vorher erstellt haben. Dann setzen wir die Parameter. Um das Vorgehen zu beschleunigen konvertieren wir das Farbbild in das Grau Bild. Die Funktion „DetectMultiScale“ gibt und den gefundenen Region als eine Matrix. Mit der „Convert“ Funktion kann man die gefundene Matrix ins Rectangle Array Konvertieren, welche uns die Position und die Größe jeder gefundenen Region gibt.

IV. HEVC/H.265 Codec

High Efficiency Video Coding kurz HEVC oder auch bekannt unter dem Namen H.265 ist der neuste Codec, dass dafür gedacht ist, die 4K Videos zu komprimieren. Dieses Format entstand durch eine gemeinsame Entwicklung der ISO/IEC Moving Picture Experts Group (MPEG) und der ITU-T Video Coding Experts Group (VCEG).

Das Ziel war eine doppelte Kompression in Vergleich zur H.264/MPEG-4 bei gleicher Qualität zu schaffen. Zusätzlich kann H.265/HEVC von 320×240 Pixel bis zu 8192×4320 Pixel (4320p) skalieren. Ein Anwendungsbereich ist z.B. Übertragung von ultra-hochauflösenden Fernsehprogrammen, Blu-ray mit 4K oder Streaming-Angebote.

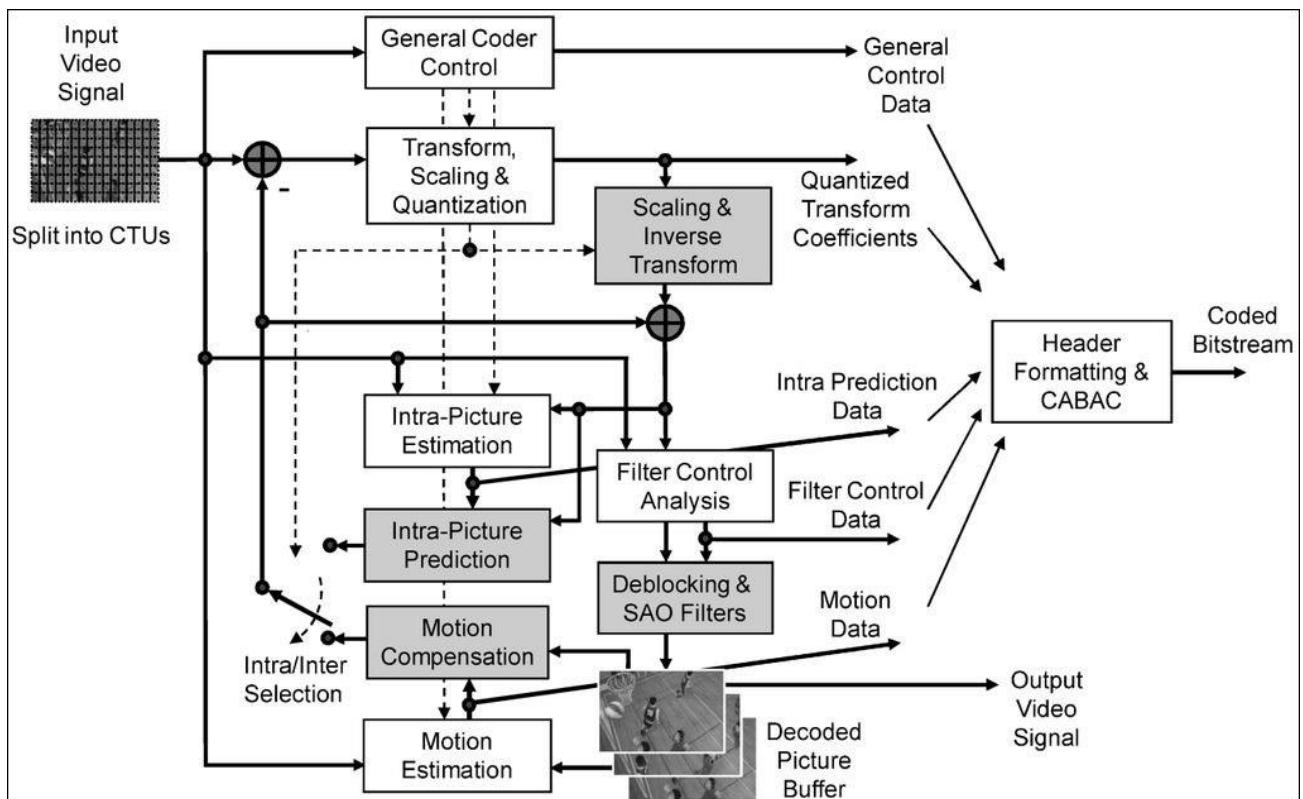


Abbildung 12 HEVC Schema

1. GOP

Das HEVC arbeitet mit einer Group of pictures (GOP) Struktur (Abbildung 13). GOP enthält folgende Bildtypen:

I Frame (intra coded picture) ist ein mit JPG ähnlichem verfahren codiertes volles Bild. Dieses Bild ist in jedem GOP enthalten.

P Frames (predictive coded picture) ist ein durch motion prediction komprimiertes Bild. Als Referenzbild wird das I Frame benutzt.

B Frame (bipredictive coded picture) ist ähnlich wie das P Frame, entsteht aber aus vorherigen und zukünftigen Bildern.

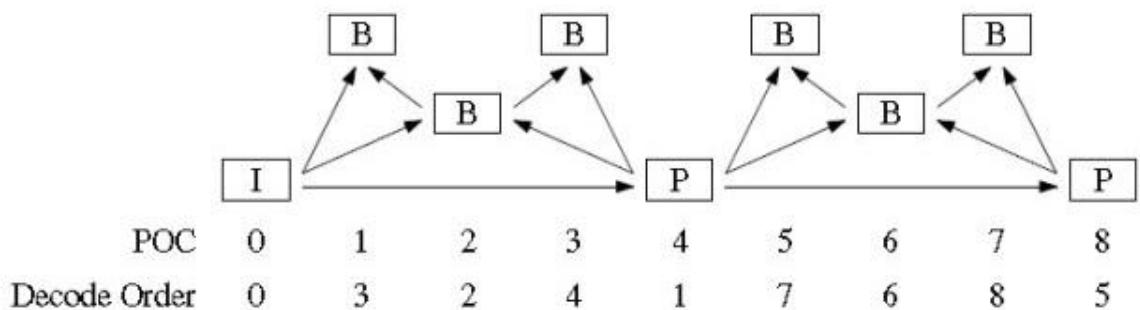


Abbildung 13 GOP Struktur

2. Slice und Coding Unit:

Ein Bild besteht aus mehrere Bildteile(Slice). Jedes Bildteil wird dann in mehrere kleine Teilchen aufgeteilt, die dann zur Prädiktion oder Quantisierung benutzt werden. Diese Teilchen heißen Coding Tree Unit (CTU) und Coding Unit (CU). Das CTU besteht aus 3 Coding Tree Block (CTB) wobei jedes CTB aus mehrere Coding Blocks (CB) bestehen kann. Für die inter-picture oder intra-picture Vorhersage wird das Coding Unit (CU) benutzt. CU ist eine Gruppe von 3 CB Teilchen, die für die 3 Farbkanäle(Y, Cb, Cr) stehen.

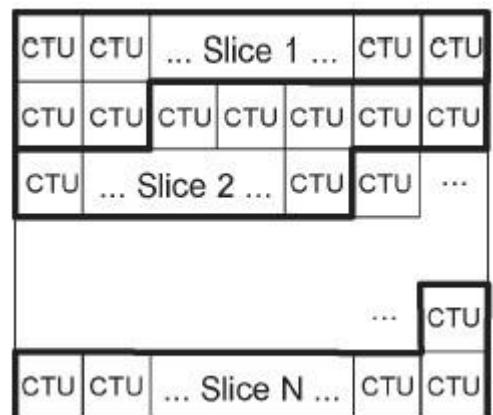


Abbildung 14 Slice und CTU Beispielverteilung

3. Profile

Im Vergleich zur AVC hat das H.265 nur drei Profile: Main, Main 10 und MainStillPicture. Das Main Profil ist vergleichbar mit dem „Progressive High Profil“ des H.264/MPEG-4 AVC Codecs.

Zusätzlich wurde es die Entwicklung von zukünftigen Erweiterungen für HEVC gestartet, wie etwa die Scalable-Video-Coding- (SHVC)[11] und die Multiview-Video-Coding-Erweiterung (MV-HEVC).

Das Main Profil arbeitet mit 8bit Farbpalette. Es arbeitet mit der 4:2:0 Farbunterabtastung. Des Decoder Buffer ist auf 6 Frames für Helle Komponente begrenzt.

Die Blockgröße kann von 64x64 bis 16x16 eingestellt werden. Die Prädiktionsblockgröße

liegt zwischen 64x64 bis 8x8 symmetrische Prädiktion (Abbildung 15). Es hat 35 Modi und verbesserte Prädiktion.

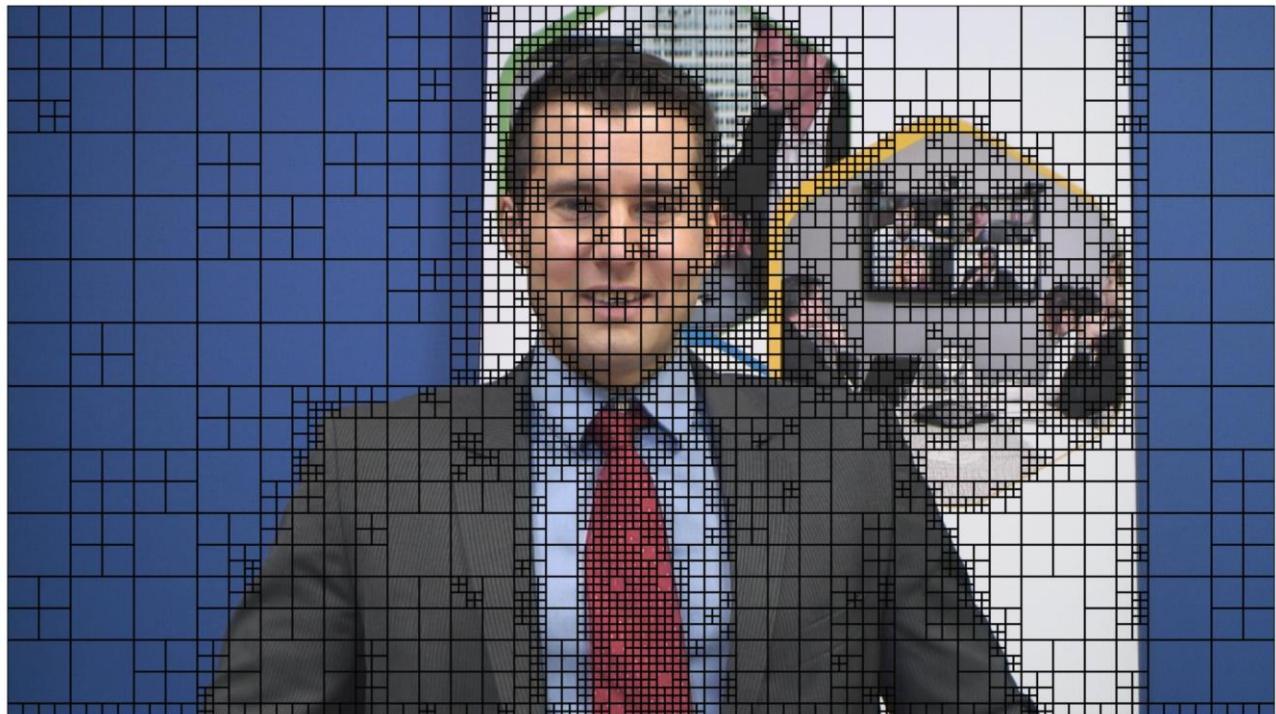


Abbildung 15 Blockgrößenverteilung

Das Main 10 hat ähnliches Struktur wie das Main Profil, aber mit 10bit farbtiefe mehr Blockgrößen und Asymmetrisches Prädiktion.

Main Still Picture ist für bewegungslose Bilder gedacht und hat einige Begrenzungen, die dem Main Profil entsprächen.

4. Level

HEVC hat genauso wie seine Vorgänger mehrere Level, die für spezielle Anwendungen geeignet sind. Dieses Codec ist in 13 Level aufgeteilt die stellen einen Satz von Beschränkungen für die mit der Rechenleistung des Decoders und Speichernutzung zugeordneten Datenstroms. Die Höhe wird auf der Grundlage der maximalen Abtastrate, die maximale Bildgröße, maximale Durchflussrate, minimale Kompressionsverhältnis und verfügt über einen Frame-Buffer der Decoder und Encoder.

Tabelle 1 Level

Level	Max. Frequenz (Hz)	Max Größe des Frames (Pixel)	Max. Durchflussmenge für Main и Main 10 Profile (kbit/s)		Beispiel Auflösung @ max. Freq. (Max. Größe Framebuffer)	Min. Kompre ssion
			Main Level	High Level		
1	552 960	36 864	128	-	128×96@33.7 (6) 176×144@15.0 (6)	2
2	3 686 400	122 880	1 500	-	176×144@100.0 (16) 352×288@30.0 (6)	2
2.1	7 372 800	245 760	3 000	-	352×288@60.0 (12) 640×360@30.0 (6)	2
3	16 588 800	552 960	6 000	-	640×360@67.5 (12) 720×480@42.1 (8) 720×576@37.5 (8) 960×544@30.0 (6)	2
3.1	33 177 600	983,040	10 000	-	720×480@84.3 (12) 720×576@75.0 (12) 960×544@60.0 (8) 1280×720@33.7 (6)	2

4	66 846 720		12 000	30 000	1280×720@68.0 (12) 1920×1080@32.0 (6) 2048×1080@30.0 (6)	4
4.1	133 693 440	2 228 224	20 000	50 000	1280×720@136.0 (12) 1920×1080@64.0 (6) 2048×1080@60.0 (6)	4
5	267 386 880		25 000	100 000	1920×1080@128.0 (16) 3840×2160@32.0 (6) 4096×2160@30.0 (6)	6
5.1	534 773 760	8 912 896	40 000	160 000	1920×1080@256.0 (16) 3840×2160@64.0 (6) 4096×2160@60.0 (6)	6
5.2	1 069 547 520		60 000	240 000	1920×1080@300.0 (16) 3840×2160@128.0 (6) 4096×2160@120.0 (6)	8
6	1 069 547 520		60 000	240 000	3840×2160@128.0 (16) 4096×2160@120.0 (16) 4096×2304@113.3 (12) 7680×4320@32.0 (6) 8192×4320@30.0 (6)	8
6.1	2 139 095 040	35 651 584	120 000	480 000	3840×2160@256.0 (16) 4096×2160@240.0 (16) 4096×2304@226.6 (12) 7680×4320@64.0 (6) 8192×4320@60.0 (6)	8
6.2	4 278 190 080		240 000	800 000	3840×2160@300.0 (16) 4096×2160@300.0 (16) 4096×2304@300.0 (12) 7680×4320@128.0 (6) 8192×4320@120.0 (6)	6

5. Installation

Auf der Seite <https://hevc.hhi.fraunhofer.de/> stehen 3 Links zur HEVC SVN Repository mit dem Quellcode. Die Ersten 2 Links sind für die SVN Programmen. Im drittem Link kann man das Zip Ordner mit dem Main Branche direkt von der Seite runterladen, was aber keine für diesen Projekt wichtige scc Daten enthält. Diese Änderungen liegen in einem anderem Branche liegt.

Im Ordner „hevc\build“ Öffnet man das Projekt für VS2010, die Einstellungen sollen nicht verändert werden. Um volle RAM Speicher auszunutzen muss man es im x64 kompilieren. Es erzeugt im Ordner „hevc\bin\vc10\x64\Debug“ die Daten TAppEncoder.exe (Abbildung 16) und TAppDecoder.exe (die TAppDecoderAnalyser.exe ist nur für die Videoanalyse da).

Noch ein wichtiger Ordner ist „hevc\cfg“. Es enthält die Beispieleinstellungen für Video Encoding.

```
PS C:\Users\xma1\Source\Repos\HEVCandFaceDetect\hevc\bin\vc10\win32\Debug> .\TAppEncoder.exe -c test.cfg -i mobile_cif.yuv
HM software: Encoder Version [16.6] (including RExt)[Windows][Vs 1600][32 bit]
*****
** WARNING: --SEIDecodedPictureHash is now disabled by default. **
** automatic verification of decoded pictures by a decoder requires this option to be enabled. **
*****
** WARNING: For conforming bitstreams a valid Profile value must be set! **
*****
** WARNING: For conforming bitstreams a valid Level value must be set! **
*****
Input File : mobile_cif.yuv
Bitstream File : mobile_hevc
Reconstruction File : mobile_out testrk.yuv
Real Format : 352x288 24Hz
Internal Format : 352x288 24Hz
Sequence PSNR output : Linear average only
Sequence MSE output : Disabled
Frame MSE output : Enabled
Cabac-zero-word-padding : FaceDatei.txt
ObjectQP File : Frame based coding
Frame/Field : 0 - 9 (10 frames)
Frame_index : none
Profile : 64 / 4 / 4
CU size / depth / total-depth : 64 / 32
RQT trans. size (min / max) : 3
Max RQT depth inter : 3
Max RQT depth intra : 3
Min PCM size : 8
Motion search range : 64

log2_sao_offset_scale_luma : 0
log2_sao_offset_scale_chroma : 0
Cost function : lossy coding (default)
RateControl : 0
Max Num Merge Candidates : 5

TOOL CFG: IBD:0 HAD:1 RDQ:1 RDQTS:1 RDpenalty:0 SQP:0 ASR:0 FEN:1 ECU:0 FDM:1 CFM:0 ESD:0 RQT:1 TransformSkip:1 TransformSkipFast:1 TransformSkipLog2MaxSize:2 Slice: M=0 SliceSegment: M=0 CIP:0 SAO:1 PCM:0 TransQuantBypassEnabled:0 WPP:0 WPB:0 PME:2 WaveFrontSyncro:0 WavefrontSubstreams:1 ScalingList:0 TMVPMode:1 AQPs:0 SignBitHidingFlag:1 RecalQP:0

Non-environment-variable-controlled macros set as follows:
REXT_DECODER_DEBUG_BIT_STATISTICS = 0
REXT_HIGH_BIT_DEPTH_SUPPORT = 0
REXT_HIGH_PRECISION_FORWARD_TRANSFORM = 0
00043_BEST EFFORT DECODING = 0

Input ChromaFormatIDC = 4:2:0
output (internal) ChromaFormatIDC = 4:2:0

POC 0 TId: 0 ( I-SLICE, nQP 0 QP 0 ) 339136 bits [Y 23.8026 dB U 32.2257 dB V 31.9309 dB] [ET 21 ] [L0 ] [L1 ]
POC 1 TId: 0 ( B-SLICE, nQP 3 QP 3 ) 225096 bits [Y 25.1742 dB U 32.5916 dB V 32.3676 dB] [ET 54 ] [L0 0 ] [L1 ]
```

Abbildung 16 Video Encoding (Ausschnitt)

6. HEVC Parameter

Input File

Das HEVC Codec arbeitet nur mit YUV RAW Daten (Abbildung 17). Das sind die nichtkomprimierten Daten die in YUV gespeichert sind. Das Testvideo das ich als Test verwendet hatte war in 8bit YUV 420 gespeichert und hatte kein Header. Die Auflösung war 352X288@24Hz.

Da das Video in 420 Format gespeichert wurde für das Y Kanal 101.376bits und für U und V nur jeweils 25.344bits sponsert.

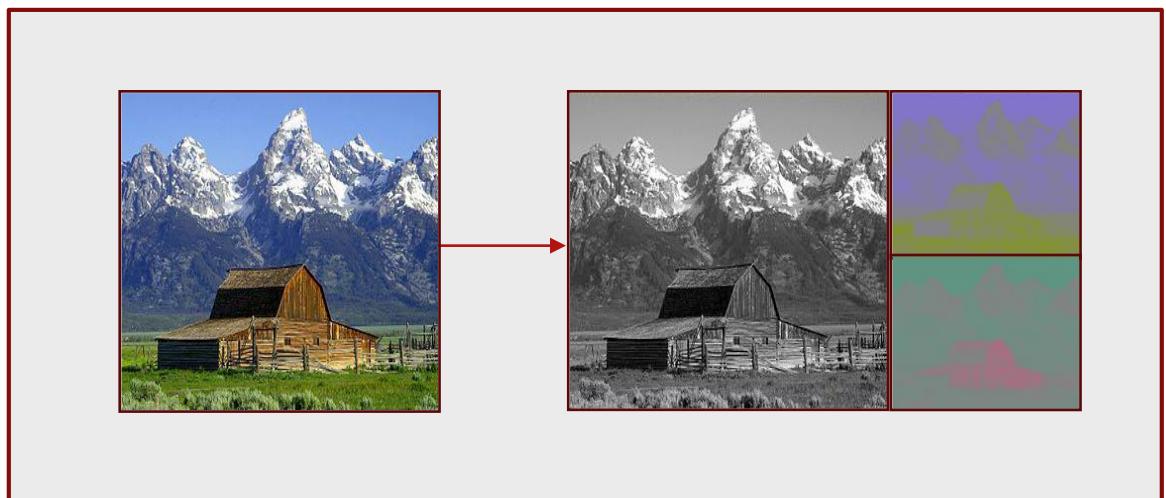


Abbildung 17 YUV Speicherung

Encoder Parameter

Um den Videofile zu codieren gibt man in CMD den Befehl:

“TAppEncoder.exe -c settings.cfg -i RAW.yuv”

Im Setting File stehen die Einstellungen für die Komprimierung des Videos.

-c steht für „Config“ hinter der **-c** steht Pfad zu der Konfigurationsdatei.

-i steht für „Inputfile“ hinter der **-i** steht Pfad zur yuv Videofile.

Im Ordner „hevc\cfg“ Liegen die Beispieldaten für die Konfigurationsdatei. Um viel Arbeit zu sparen nimmt man am besten eine Datei aus dem Ordner und verändert es so, dass es zu der Aufgabe passt.

Hier sind die wichtigsten Einstellungen:

- **BitstreamFile** – Das Ausgangsfile (Komprimiertes Video). Muss immer eingegeben werden, sonst wird nicht gespeichert.
- **ReconFile** – Das decodiertes File im yuv Format.
- **FrameSkip** und **FramesToBeEncoded** – Welches Bereich soll komprimiert werden.
- **FrameRate**, **SourceWidth** und **SourceHeight** – Information über yuv Video. Da das Video kein Header enthält muss man es per Hand eingeben.
- **Profile** – Welches Profil soll benutzt werden. Wenn es weggelassen wird, dann zahlt es als „none“.
- **Level** – Welches Level benutzt wird. Wie mit dem Profil als Default steht „none“
- **QP** – Quantisierungsparameter. 0 ist Beste Qualität 51 ist Maximale Quantisierung.

Den restlichen Parameter kann man aus der Beispieldatei kopieren. Im Ordner „hevc\doc“ liegt die Beschreibung zu den meisten Einstellungen, die restlichen Einstellungen findet man in der „source/Lib/TLibCommon/TypeDef.h“ Datei

Decoder Parameter

Um den Videofile zu decodieren gibt man in die CMD den Befehl:

„TAppDecoder.exe -b CodiertersFile.hevc -o DecodiertesFile.yuv“, ein.

-b steht für den HEVC Kodiertes Videofile.

-o steht für dem Output File im YUV Format.

Die restlichen Einstellungen stehen im „hevc\doc“ am Ende der PDF. Für den einfachen Fall braucht man nur die oben beschriebenen Einstellungen.

7. HEVC Vergleich



Abbildung 18 YUV RAW

Quantisierung

Für den ersten Test wurden die Standard Einstellungen benutzt. Das Profil wurde nicht ausgewählt (Profil und Level = „none“). Wegen lange Kompressionszeiten wurden nur 10 Frames komprimiert.



Abbildung 19 Quantisierung: 0



Abbildung 20 Quantisierung: 51

Tabelle 2 Subjektives Vergleich der Quantisierung

Quantisierung	RAW	Lossless (0)	Full (51)
Größe	1,45 MB	688 KB	1,81 KB
Bitrate	12,17 Mbps	13,213824 Kbps	34,7520 bps
PSNR YUV Ø	--	68,2514 dB	20,0957 dB
PSNR Y Ø	--	71,2398 dB	18,5913 dB
PSNR U Ø	--	70,8383 dB	28,4320 dB
PSNR V Ø	--	70,9036 dB	27,2496 dB
I- Frames	10	1	1
P-Frames	0	0	0
B-Frames	0	9	9
Objektiv	4	4	1

Videoformat

Hier wird die Videogröße verglichen. Es wurde das Mainprofil und 32 als Quantisierungsparameter benutzt. Die GOP Size ist 8 Frames.



Abbildung 22 480p Komprimiert



Abbildung 21 1080p Komprimiert

Tabelle 3 Subjektive Videoperformance Vergleich Größen

Bildgröße	480p	720p	1080p	4K UHD
Größe	15 KB	27 KB	47 KB	111 KB
Bitrate	267,2968 bps	489,6056 bps	865,0024 bps	2089,8904
PSNR YUV Ø	38,8395 dB	39,7183 dB	40,7244 dB	42,8896
PSNR Y Ø	41,6867 dB	42,4305 dB	43,4271 dB	41,8960
PSNR U Ø	43,5993 dB	44,2758 dB	45,2500 dB	47,0821
PSNR V Ø	37,7594 dB	38,6658 dB	39,6766 dB	45,4291
I- Frames	1	1	1	1
P-Frames	0	0	0	0
B-Frames	9	9	9	9
Objektiv	4	5	5	5

Man sieht dass der Codec mit einem größerem Bild viel besser arbeitet als mit einem Kleinerem. Ein 4K Bild hat ein Größeres PSNR als 480p und dabei auch besseres Qualität als 480p. 480p ist verschwommen und hat viele Artefakte. Ab 720p ist das Bild scharf und hat keine Artefakte.



Abbildung 23 Original 480p



Abbildung 25 HEVC 480p



Abbildung 24 HEVC 1080p

Vergleich zur anderen Codec Formate

Zum Codieren wurde das Programm MediaCoder Benutzt, da dieses Programm die Daten in fast jedes Format komprimieren kann. Überall wurden dieselben Parameter eingegeben. Alle Videos Wurden mit gleicher Bitrate Komprimiert. Zum Test wurde ein 3 Sekunden langes FullHD Video genommen.

1000 Bitrate



Abbildung 27 H.264



Abbildung 26 H.265



Abbildung 29 Xvid



Abbildung 28 MPEG2

Tabelle 4 Vergleich Differenzgröße / -PSNR in Prozent mit Bitrate = 1000

Codec	MPEG2	Xvid	H.264	H.265	PSNR Ø
Größe/KB	24,803	25,129	23,940	23,907	
MPEG2	-	1,3% / 8,33%	3,6% / 3,76%	3,75% / 23,63%	31,054
Xvid	-	-	4,97% / 11,16%	5,11% / 29,50%	28,667
H.264	-	-	-	0,14% / 20,65%	32,268
H.265	-	-	-	-	40,665
Subjektive	2	4	5	5	

Gerechnet wurde $|MPEG2 - Xvid| / MPEG2$. Die H.264/265 sind in diesem Vergleich fast gleich. Die Bildqualität in Beiden Fällen ist super obwohl es gibt ein PSNR Unterschied von 8.397 dB. Das Xvid in diesem Fall ist verschwommen und das MPEG2 ist stark Verzerrt. Von der Größe das H.265 ist das kleinste, das H.264 ist aber nicht viel größer.

512 Bitrate



Abbildung 31 H.264



Abbildung 30 H.265



Abbildung 33 Xvid



Abbildung 32 MPEG2

Tabelle 5 Vergleich Differenzgröße / -PSNR in Prozent mit Bitrate = 512

Codec	MPEG2	Xvid	H.264	H.265	PSNR Ø
Größe/KB	21,363	15,129	12,250	12,444	
MPEG2	-	41,21% / 10,80%	74,39% / 34,48%	71,67% / 2,85%	31,500
Xvid	-	-	23,5% /	23,5% / 12,32%	28,430
H.264	-	-	-	1,56% / 14%	32,423
H.265	-	-	-	-	37,700
Subjektive	2	2	4	4	

Die MPEG2 geht nicht weiter als 816kBit/s. Xvid steht auch ein bisschen über der 512kBit/s Soll Grenze, mit der Bitrate von 578kBit/s ist es aber noch akzeptabel für den Vergleich. Die H.264/5 Formate haben hier trotzdem ein besseres Bildqualität und wiegen auch weniger. Von der Größe gewinnt das H.264 von der Qualität H.265.

128 Bitrate



Abbildung 35 H.264



Abbildung 34 H.265



Abbildung 37 Xvid



Abbildung 36 MPEG2

Tabelle 6 Vergleich Differenzgröße / -PSNR in Prozent mit Bitrate = 128

Codec	MPEG2	Xvid	H.264	H.265	PSNR Ø
Größe/KB	21,248	14,518	3,107	3,169	
MPEG2	-	46,36% / 13,04%	583,88% / 2,41%	570,5% / 1,17%	31,485
Xvid	-	-	367% / 9,41%	358,13% / 12,58%	27,852
H.264	-	-	-	1,96% / 3,48%	30,745
H.265	-	-	-	-	31,859
Subjektive	2	2	1	3	

Die MPEG2 und Xvid haben die minimale Bitrategrenze erreicht. Die MPEG2 hat in Wirklichkeit 812kBit/s und Xvid 554kBit/s. Obwohl diese beiden Formate mehr Bitrate als die H.265 haben ist H.265 immer noch viel besser in der Bildqualität als beide anderen. Das H.264 Hat zwar unter gleicher Bedingung kleinere Größe, hat aber eine mangelhafte Bildqualität.

V. Verbesserung von HEVC mit Hilfe von OpenCV

1. Schnittstelle:

HEVC hat leider keine Möglichkeit die Quantisierung für einzelne Bereiche in jedem Frame zu ändern. Deswegen müsste man es im Quellcode von dem eingefügen. Die Übergabe erfolgt durch eine Textdatei. So kann man die Gesicht Erkennung und auch andere Funktionen unabhängig von dem HEVC behandeln. Das einzige Problem, das bei diesem Verfahren auftritt, ist das die Gesicht Erkennung vor der Komprimierung gemacht werden muss und es in eine Datei speichern muss.

2. HEVC Code Struktur:

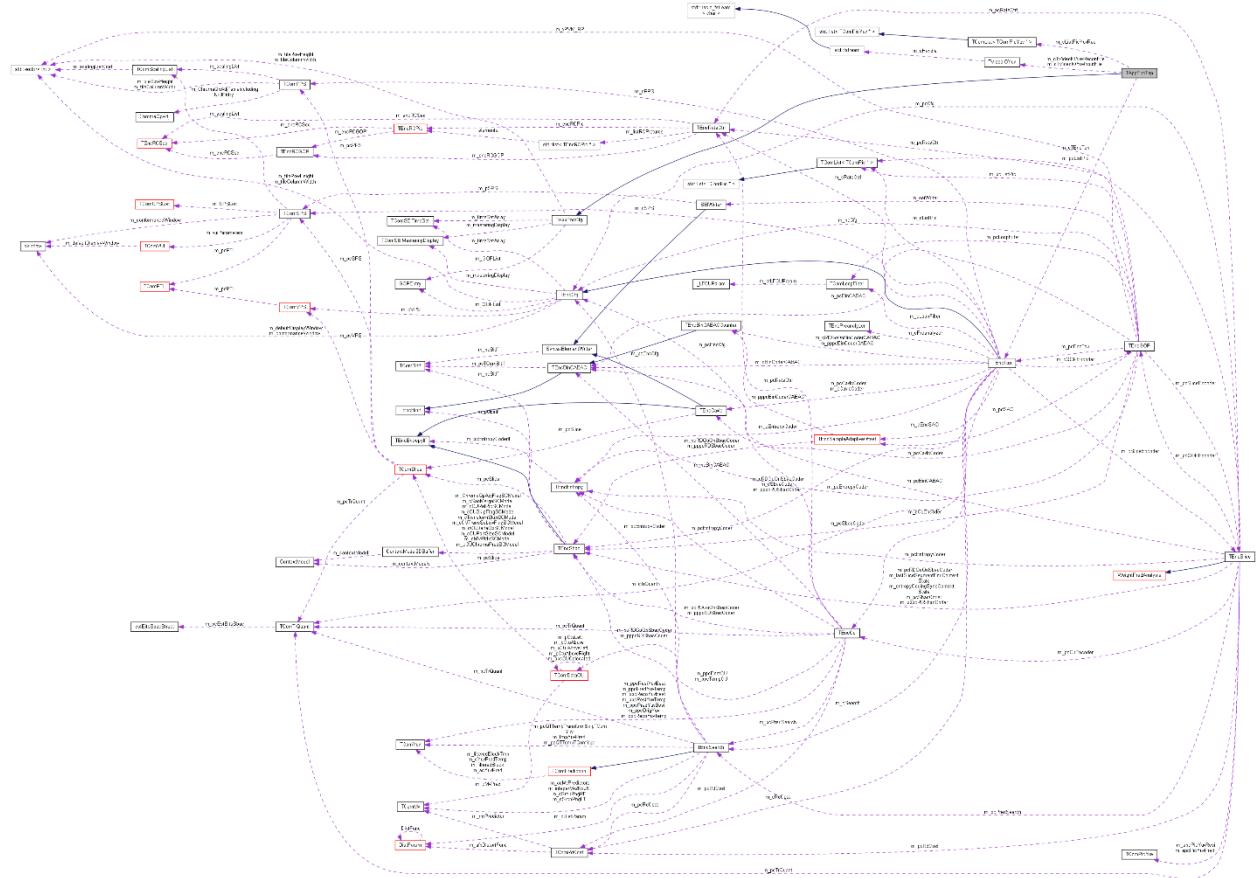


Abbildung 38 HEVC Klassendiagramm (Gesamtübersicht, Großes Bild im Anhang)

Wie man auf der Abbildung 38 sieht, ist die Struktur von HEVC sehr kompliziert und unübersichtlich aufgebaut. Für die Schnittstelle brauchen wir aber nur die CU, Slice und Picture (Pic) Klassen zu betrachten, da in diesen Klassen die Quantisierung durchgeführt wird.

3. Kode für Schnittstelle:

Der Hauptcode wurde in die Funktion `xComputeQP` aus der Klasse `TEncCu` geschrieben. Diese Funktion berechnet die Quantisierung für die einzelne Code Unit. Es lädt die Quantisierung aus für das in der Videdatei angegebenen Wert und addiert zu dem den Image QP (Quantisation Parameter) Offset, das es mit Hilfe von dem vorgerechnetem Adaptivem QP. Wenn es kein Adaptives QP ausgerechnet wurde wird das Angegebene QP ausgegeben. Mit der Funktion `clip3` wird der Wert an das min/max Bereich angepasst.

```
Int TEncCu::xComputeQP(TComDataCU* pcCU, UInt uiDepth)
{
    TComSlice* slice = pcCU->getSlice();
    Int iBaseQp = slice->getSliceQp();
    Int iQpOffset = 0;

    if (_pcEncCfg->getObjectTypePath() != "")
    {
        UInt PosX = pcCU->getCUPelX();
        UInt PosY = pcCU->getCUPelY();
        UChar Width = pcCU->getWidth(0);
        UChar Height = pcCU->getHeight(0);

        int ObX1 = 0;
        int ObX2 = 0;
        int ObY1 = 0;
        int ObY2 = 0;
        int ObQP = 0;
        bool invert = false;
        int frame = ObjectInFrame::iFrame;

        for (int i = 0; i < getObjectQP(frame).parameter.size(); i++)
        {
            ObX1 = getObjectQP(frame).parameter[i].X;
            ObX2 = getObjectQP(frame).parameter[i].X + getObjectQP(frame).parameter[i].Width;
            ObY1 = getObjectQP(frame).parameter[i].Y;
            ObY2 = getObjectQP(frame).parameter[i].Y + getObjectQP(frame).parameter[i].Height;
            ObQP = getObjectQP(frame).parameter[i].QP;
            invert = getObjectQP(frame).parameter[i].Invert;

            ObX1 = ((int)(ObX1 / 64)) * 64;
            ObX2 = ((int)(ObX2 / 64)) * 64;
            ObY1 = ((int)(ObY1 / 64)) * 64;
            ObY2 = ((int)(ObY2 / 64)) * 64;

            if (invert != (PosX >= ObX1 && PosX <= ObX2 && PosY >= ObY1 && PosY <= ObY2))
            {
                iBaseQp = ObQP;

                //Calculate Lambda:
                Double qp_temp = (Double)iBaseQp - 12;
                double NumberBFrames = 3;
                Double dLambda_scale = 1.0 - Clip3(0.0, 0.5, 0.05*NumberBFrames);
                double dQPFactor = 0.57*dLambda_scale;
                double dLambda = dQPFactor*pow(2.0, qp_temp / 3.0);

                Double dLambda[MAX_NUM_COMPONENT] = { dLambda };
                for (UInt compIdx = 1; compIdx < MAX_NUM_COMPONENT; compIdx++)
                {
                    const ComponentID compID = ComponentID(compIdx);

```

```

        Int chromaQPOffset = slice->getPPS()->getQpOffset(compID) + slice-
>getSliceChromaQpDelta(compID);
        Int qpc = iBaseQp - chromaQPOffset;
        Double tmpWeight = pow(2.0, (iBaseQp - qpc) / 3.0); // takes into
account of the chroma qp mapping and chroma qp Offset
        m_pcRdCost->setDistortionWeight(compID, tmpWeight);
        dLambdas[compIdx] = dLambda / tmpWeight;
    }

    m_pcTrQuant->setLambdas(dLambdas);
}
}

if (m_pcEncCfg->getUseAdaptiveQP())
{
    TEncPic* pcEPic = dynamic_cast<TEncPic*>(pcCU->getPic());
    UInt uiAQDepth = min(uiDepth, pcEPic->getMaxAQDepth() - 1);
    TEncPicQPAadaptationLayer* pcAQLayer = pcEPic->getAQLayer(uiAQDepth);
    UInt uiAQUPosX = pcCU->getCUPelX() / pcAQLayer->getAQPartWidth();
    UInt uiAQUPosY = pcCU->getCUPelY() / pcAQLayer->getAQPartHeight();
    UInt uiAQUStride = pcAQLayer->getAQPartStride();
    TEncQPAadaptationUnit* acAQU = pcAQLayer->getQPAadaptationUnit();

    Double dMaxQScale = pow(2.0, m_pcEncCfg->getQPAadaptationRange() / 6.0);
    Double dAvgAct = pcAQLayer->getAvgActivity();
    Double dCUAct = acAQU[uiAQUPosY * uiAQUStride + uiAQUPosX].getActivity();
    Double dNormAct = (dMaxQScale*dCUAct + dAvgAct) / (dCUAct + dMaxQScale*dAvgAct);
    Double dQpOffset = log(dNormAct) / log(2.0) * 6.0;
    iQpOffset = Int(floor(dQpOffset + 0.49999));
}

return Clip3(-pcCU->getSlice()->getSPS()->getQpBDOffset(CHANNEL_TYPE_LUMA), MAX_QP, iBaseQp +
iQpOffset);
}

```

Mit der Funktion `m_pcEncCfg->getObjectQPPath() != ""` wird geprüft ob die neue Funktion im Einstellungen definiert wurde. So kann man vor Komprimieren dem Programm sagen ob die Schnittstelle benutzt werden soll oder nicht.

Mit der Funktion `getCUPelX/Y` wird die Position von dem CU ermittelt. Das Bildnummer aus der Variable `ObjectInFrame::iFrame`, das bei jedem Aufruf von neuem Bild um eins vergrößert wird, genommen.

Die Daten werden mit Hilfe von `getObjectQP` Funktion aus der Datei genommen. Der Kode ist so aufgebaut, dass wenn die Datei mehrere Objekte enthält wird es nach mehrere Bereiche geprüft. Aus der Datei werden die 4 Ecken, Qp und die Investierungsvariable geladen.

- **4 Ecken:** stehen als grenzen von dem Bereich da.
- **QP:** ist der Quantisierung Parameter, dass an den Bereich Angewenden soll
- **Invert:** hilft wenn man alles außerhalb von dem Bereich invertieren will.

Wenn die CU sich in dem (oder außerhalb, wenn invertiert) Bereich sich befindet, dann wird die voreingestellte QP auf den neuen Wert ersetzt. Mit der

Aufrufposition von dem Bereich kann man für den Fall wenn sich mehrere Bereiche überlappen die Priorität des Bereiches festlegen.

Nach dem man die QP geändert wurde muss man die Lambda an den neuen Wert anpassen. Es wird nach dem Kommentar „Calculate Lambda“ gemacht. Dieses Teil wurde aber nur für die Einstellungen angepasst, die bei der Programmierung benutzt wurden.

4. Quantisierungsdatei:

Die Datei mit Quantisierungsinformation soll 7 Spalten enthalten. Jede Zeile gehört zu dem Bereich wo die Quotisierung geändert werden soll. Die Trennung erfolgt durch ein Leerzeichen. Die Zahlen sollen als integer eingegeben werden

Die einzelne Spalten stehen für:

Framenummer | X-Position | Y-Position | Breite | Höhe | QP | Inverse

B.S:

...

3 90 0 200 200 51 0

4 30 80 200 200 0 0

5 30 40 200 200 0 0

5 20 10 400 300 30 0

6 0 20 200 200 40 1

7 20 0 200 200 51 1

...

Framenummer wird von dem ersten kodierten Frame gezählt. Wenn bei der Komprimierung die Frames ausgelassen wurden, dann als erstes wird der erste Frame nach dem letztem ausgelassenem Frame gezählt. Die Frames werden von 1 gezählt.

X und Y Position stehen für die Position des Bereiches im Bild.

Breite und **Höhe** stehen für Breite und Höhe von dem Bereich.

QP steht für die Quantisierungsparameter, die in dem Bereich angewendet werden soll.

Inverse sagt ob es in dem Bereich oder außerhalb des Bereichs die QP angewendet werden soll.

5. Gesichtserkennungsprogramm:

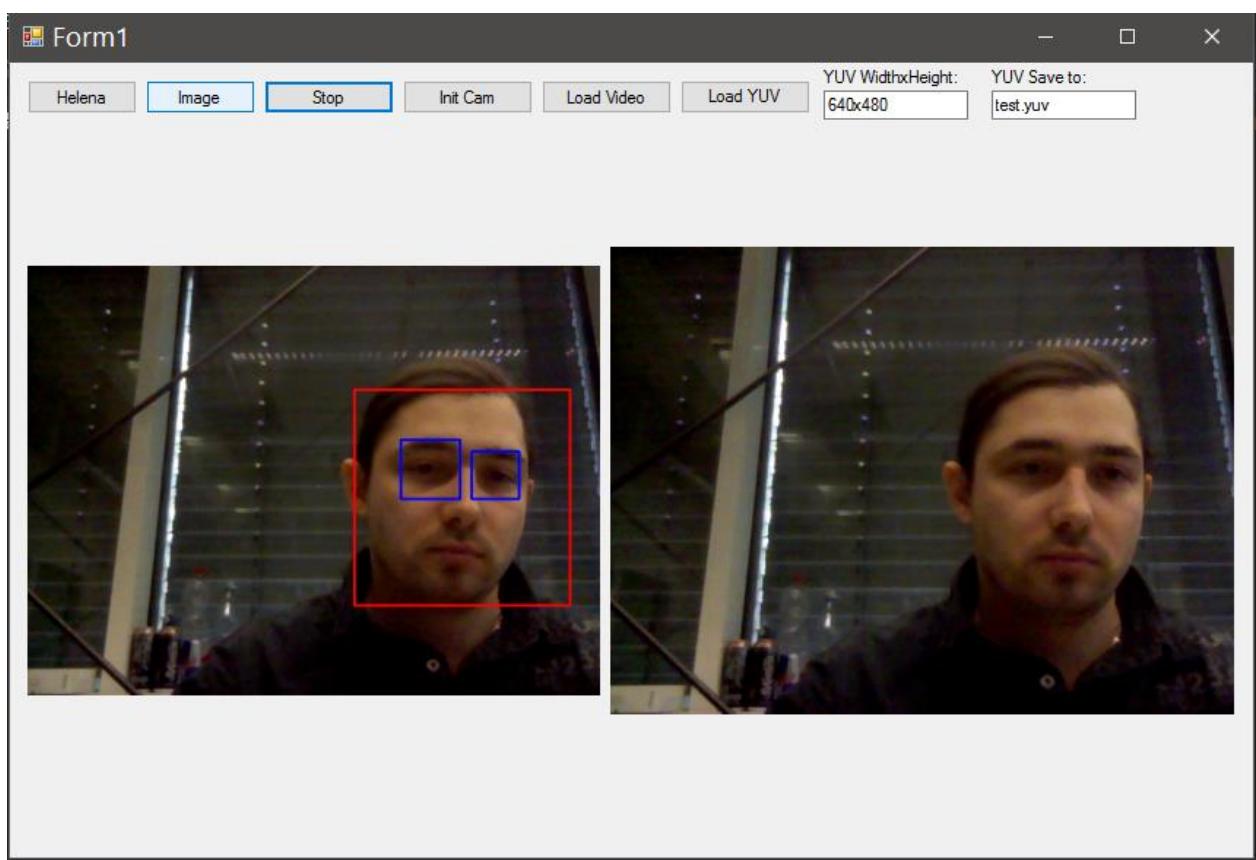


Abbildung 39 Gesichtserkennungsprogramm

Um die Gesichtserkennung mit dem HEVC Codec zu verbinden wurde das im erstem Schritt geschriebenes Programm modifiziert (Abbildung 39). Als erstes es wurden die Funktionen speichere YUV und lade YUV eingefügt. Als zweites es wurde automatisch Herstellung von Textdatei eingefügt.

Das Programm kann leider nur YUV mit 4:2:0 Einstellung behandeln. Da die RAW Datei kein Header hat muss man beim Laden die Bildgröße von dem Video per

Hand eingaben. Wenn das Feld „YUV Save to“ leer ist, dann wird es keine Datei hergestellt.

Mit den Funktionen „Init Cam“ und „Load Video“ kann man die Videoquelle auswählen. Aufnahme erfolgt mit dem drückt auf den Knopf „Start Capture“.

6. Die Neue Parameter:

Um die Funktion zu Benutzen muss man in die Konfigurationsdatei den Parameter ObjectQT Pfad zur Quantisierungsdatei einfügen.

Ausschnitt aus der „test.cfg“:

```
...
#===== Quantization =====
QP      : 0      # Quantization parameter(0-51)
MaxDeltaQP   : 0      # CU-based multi-QP optimization
MaxCuDQPDepth : 0      # Max depth of a minimum CuDQP for sub-LCU-level delta QP
DeltaQpRD    : 0      # Slice-based multi-QP optimization
RDOQ      : 1      # RDOQ
RDOQTS     : 1      # RDOQ for transform skip
```

ObjectQT : FaceDatei.txt

Das ist der neue Parameter

Durch die ObjectQT wird die Information von der Textdatei ausgelesen und in den Speicher eingefügt. Dann bei der Komprimierung wird die Nummer des komprimierten Bildes verglichen und die Parameter aus der Datei an das Bild angewendet.

7. Lambdavariable

Im HEVC wurde neues Quantisierungs Model Names Rate Distortion (RD) eingefügt. Dies benutzt 2 Variable Lambda und QP um das Bild besser zu Komprimieren. Lambda wird aus QP berechnet und ist bei QP gleich 51 über 3000 und Bei QP gleich 0 ungefähr bei 0.11.

8. Vergleich



Es wurden die Parameter aus dem Kapitel IV.6 verwendet von Vorherigen benutzt.

Die Mischung von dem HEVC und Gesichtserkennung verbesserte die Komprimierung in dem das mit besserer Qualität gelassen hatte und Hintergrund mit schlechterer Qualität komprimierte.

Abbildung 40 HEVC mit Gesichtserkennung



Abbildung 42 Quantisierung: 0

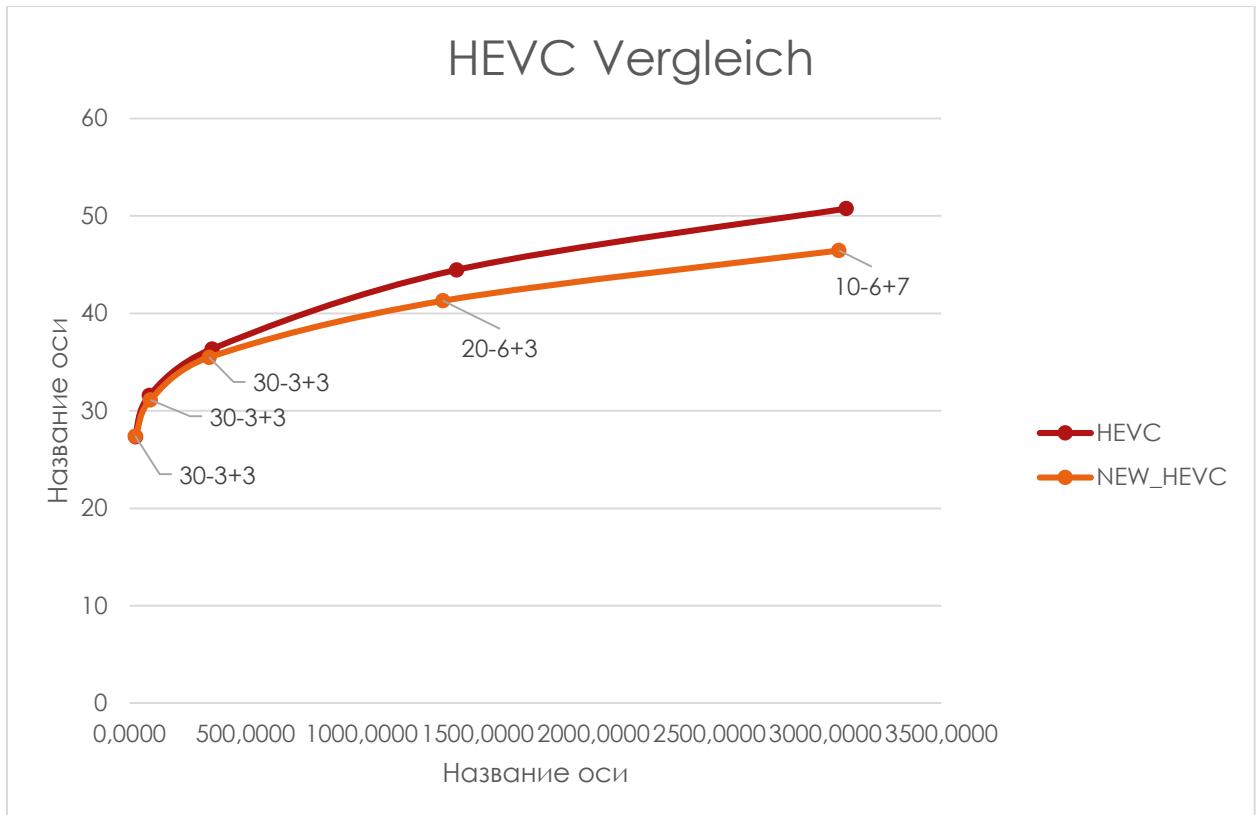


Abbildung 41 Quantisierung: 51

Tabelle 7 Subjektives Vergleich der Quantisierung

Quantisierung	Lossles (0)	Mix	Full (51)
Größe	404 KB	18,1 KB	1,49 KB
Bitrate	7927,8720 kbps	354,9504 kbps	28.3776 kbps
PSNR YUV Ø	71,5264 dB	27,7737 dB	34.2651 dB
PSNR Y Ø	74,9585 dB	26,4636 dB	26.2181 dB
PSNR U Ø	79,5902 dB	34,7819 dB	34.4612 dB
PSNR V Ø	78,3930 dB	34,4129 dB	35,2222 dB
I- Frames	1	1	1
P-Frames	0	0	0
B-Frames	9	9	9
Objektiv	5	3	1

Für den Test wurde das gleiche Video mit mehreren Einstellungen Komprimiert. Es wurde 5 verschiedene Komprimierungsstufen betrachtet. Am Anfang wurde mit einem Standartverfahren komprimiert und dann wurde von jeder QP Einstellung entweder 3 oder 6 im Gesichtsbereich subtrahiert und durch verschlechtern von dem Hintergrund wurde es probiert den gleichen Bitrate zu gewinnen.



Das Bitrate Nähe 0 hat eine Abweichung von 89bps und Nähe 51 1bps. Das PSNR ist dabei schlechter als von dem originalem Verfahren.

Das Bild selbst sieht viel besser, als das von dem originalen Verfahren. Es liegt daran, dass das Vordergrund, in unserem Fahl ist es das Gesicht, wird mit besserem Qualität dargestellt und Hintergrund erscheint verschwommen. Diesen Effekt sieht man auch teilweise im realem leben, wo die Auge sich an das fordere Objekt fokussiert, das Hintergrund dadurch verschwommen.



Abbildung 43 44/51 vs 50

In der Abbildung 43 mit dem neuem Verfahren wird das Gesicht besser erkannt als mit dem altem.



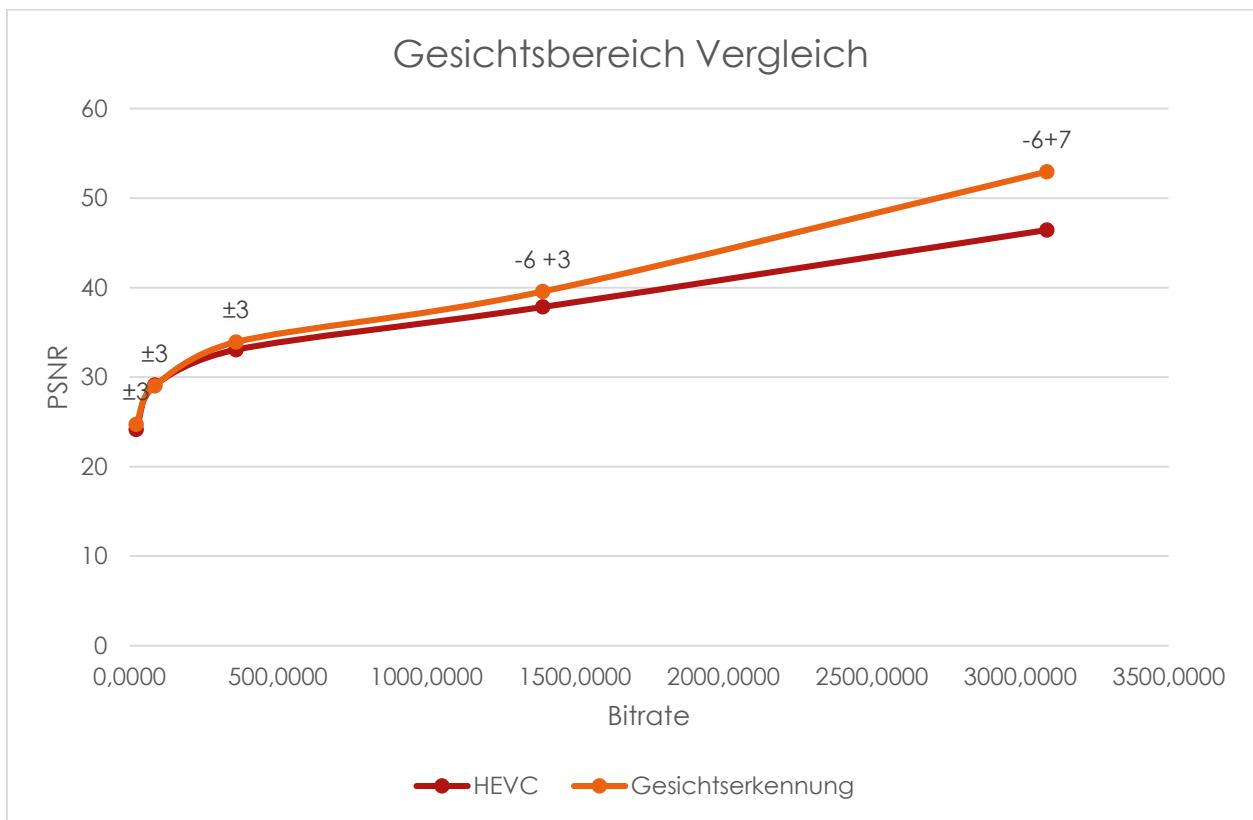
Abbildung 44 34/46 vs 40

Bei der Abbildung 44 sieht man den Unterschied besser. Das Gesicht in dem Linken Bild Sieht viel besser aus als in dem Rechten. Obwohl ist das Hintergrund Schlechter ist als im rechtem Bild ist das linke Bild viel angenehmer als das rechte.



Abbildung 45 24/36 vs 30

Bei der Abbildung 45 sieht das Gesicht schärfer aus und Hintergrund ist verschwommen. So hat man das Gefühl, dass Gesicht näher steht als Hintergrund.



In dieser Diagramm erkennt man, dass die PSNR im Bereich des Gesichtes viel besser ist als in der originaler HEVC Komprimierung.

Zusammenfassung

In dieser Arbeit wurde versucht die HEVC Videokomprimierung mit Hilfe von der Gesicht Erkennung zu verbessern. Die Gesicht Erkennung wurde mit Hilfe von der OpenCV Bibliothek programmiert.

Im ersten Arbeitsschritt wurde die Information über OpenCV und HEVC gesammelt. Dann wurde die OpenCV Bibliothek für C# namens Emgu CV installiert und die vorhandene Beispiele getestet. Aus diesen Beispielen wurden für das Projekt passende Grundlagen ausgewählt. Aus diesen Unterlagen wurde das Gesichtserkennungsprogramm (Abbildung 46) programmiert.

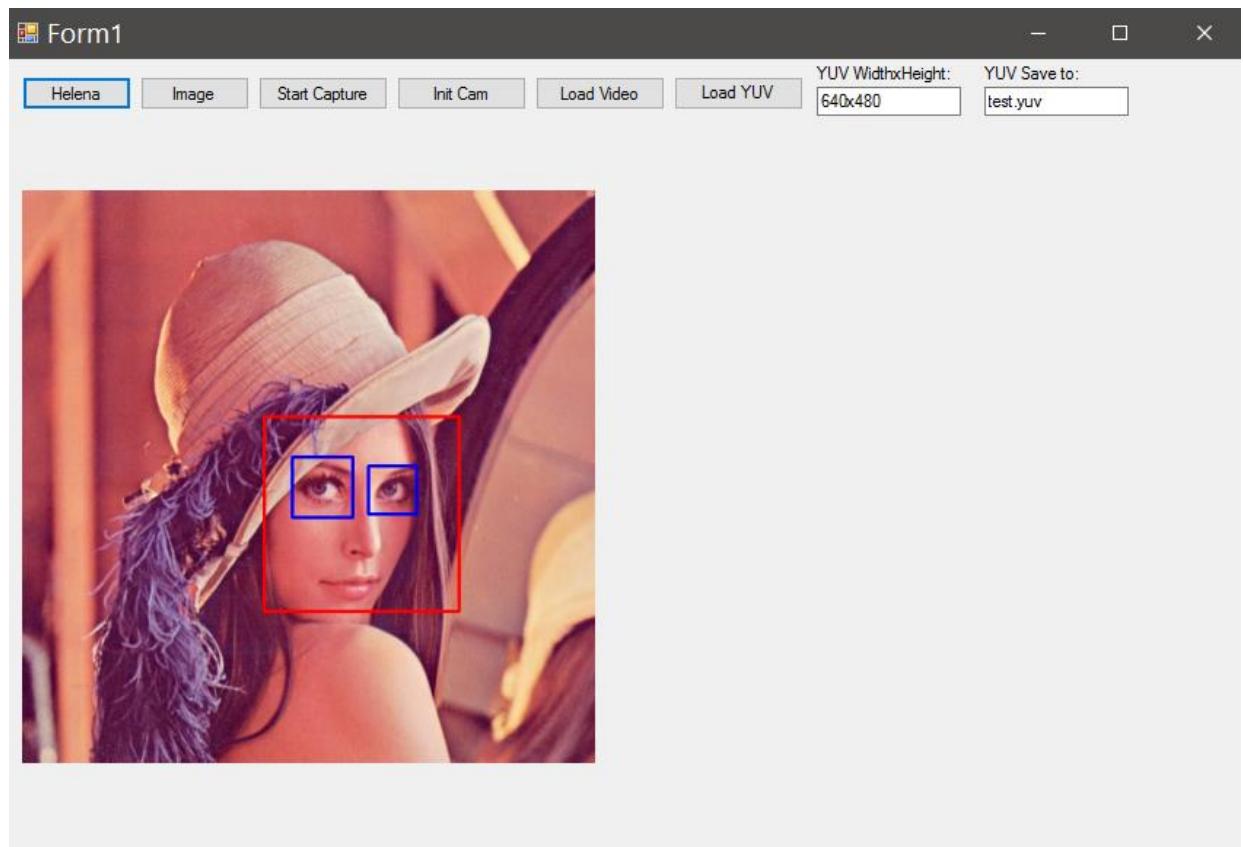


Abbildung 46 Gesichtserkennungsprogramm

Dann wurde der HEVC Quellcode von der Fraunhofer Seite geladen und kompiliert. Nach dem der Kode erstellt wurde, folgte als nächstes der Vergleich zwischen den Einstellungen und älteren Kodeks. Dafür wurde die RAW Daten hergestellt (Abbildung 47).

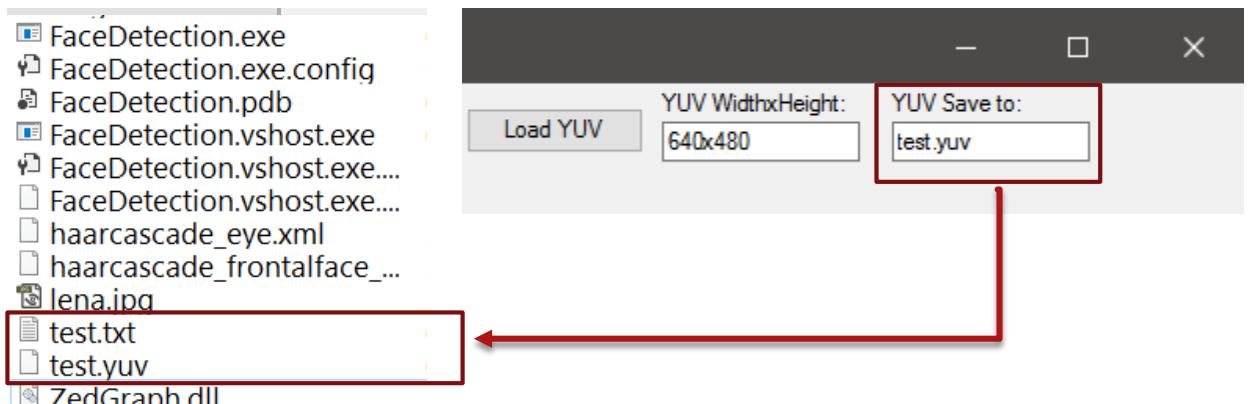


Abbildung 47 Erstellung von YUV und Einstellungsdatei in dem Gesichtserkennungsprogramm

Anschließend wurde die Schnittstelle entwickelt, die es ermöglicht die Daten von dem Gesichtserkennungsprogramm an den HEVC zu übergeben. Als zwischen Datei wurde der einfache Text Datei benutzt, das gibt die Möglichkeit die Gesichtserkennung und HEVC separat voneinander zu benutzen. Somit kann mehrere Gesicht Daten erstellen und es später bei der Komprimierung zu benutzen. In dem Gesichtserkennungsprogramm wurden die Funktionen zum Herstellen von dieser Datei und RAW Datei aus der Webkamera programmiert (Abbildung 48).

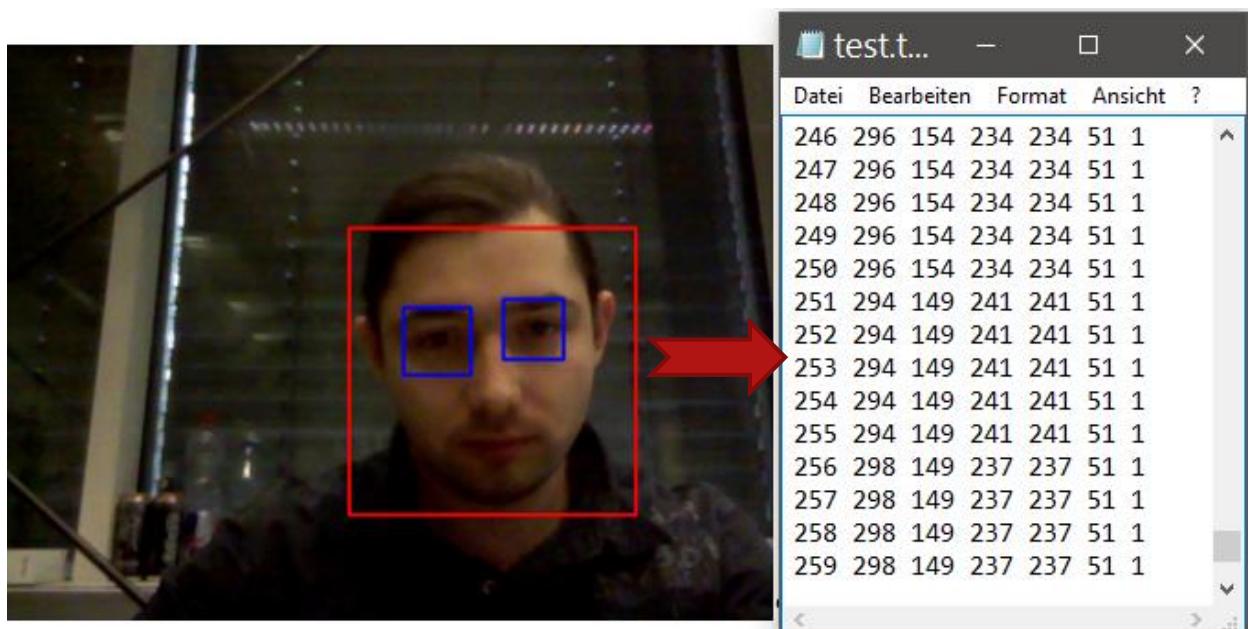


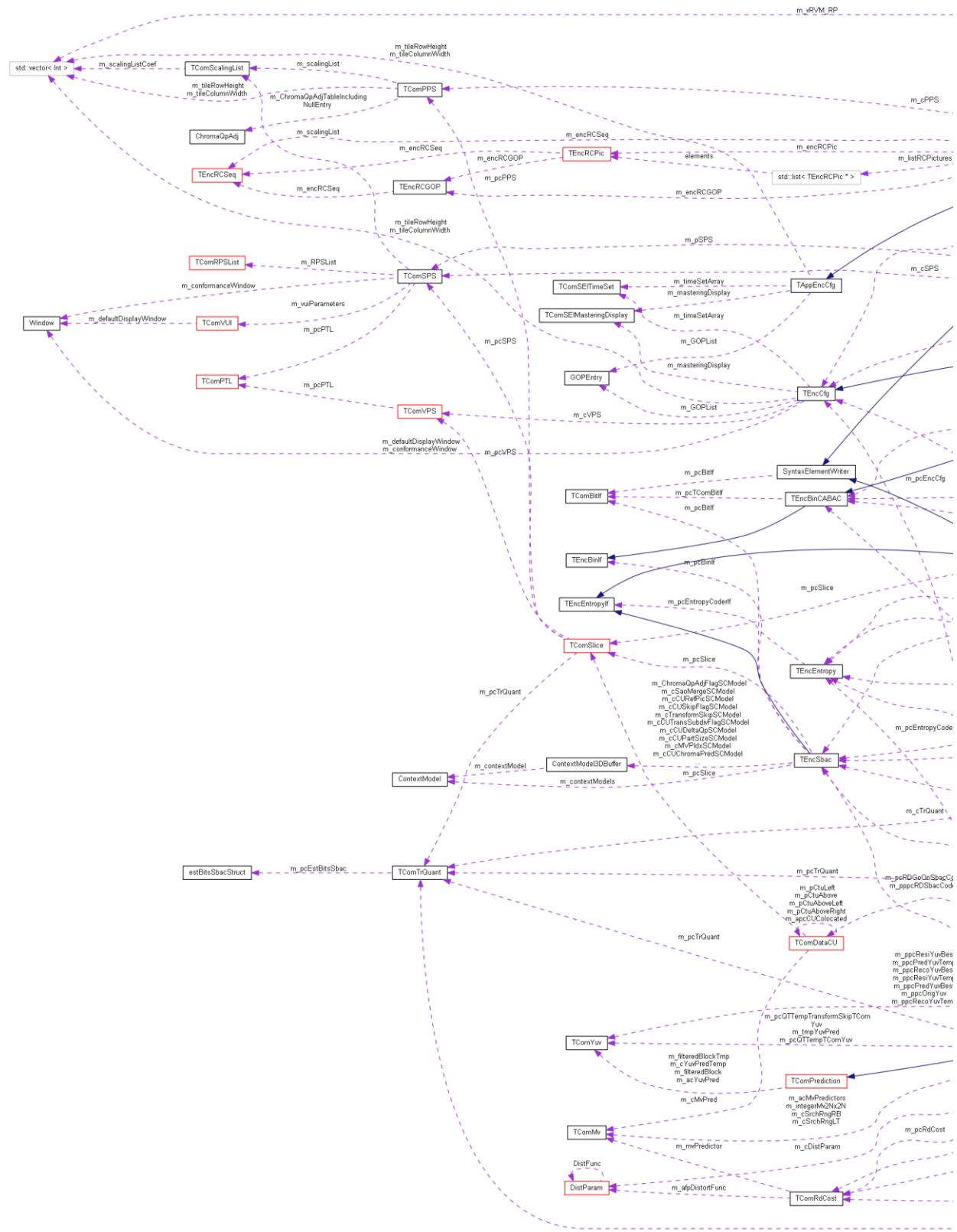
Abbildung 48 Speicherung des Bereiches in die Datei

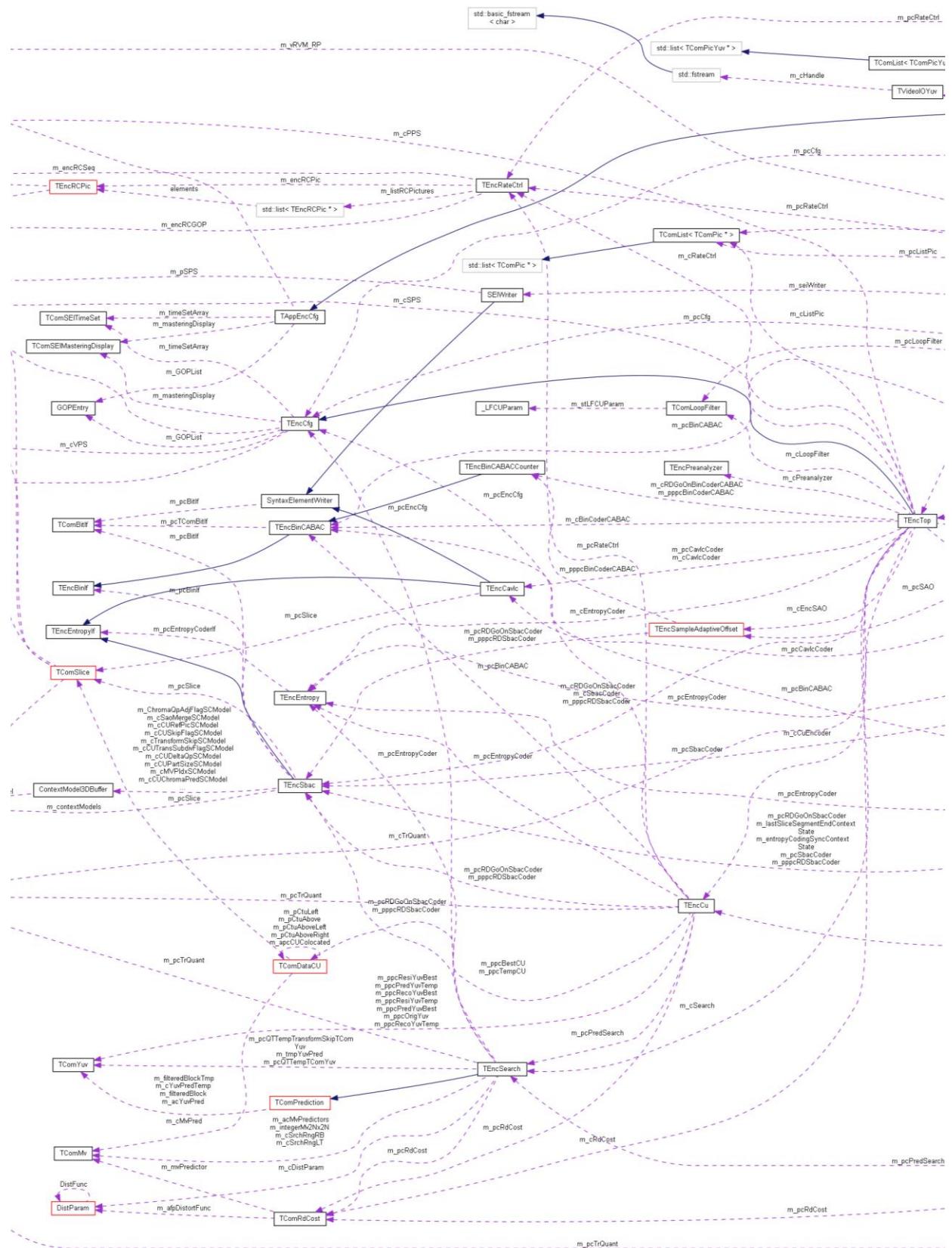
Außer dem Gesicht kann man andere Bereiche (wie z.B. Tafel mit dem Wichtigem Text) in diese Datei einfügen. Das kann die Qualität Komprimierung noch stärker verbessern.

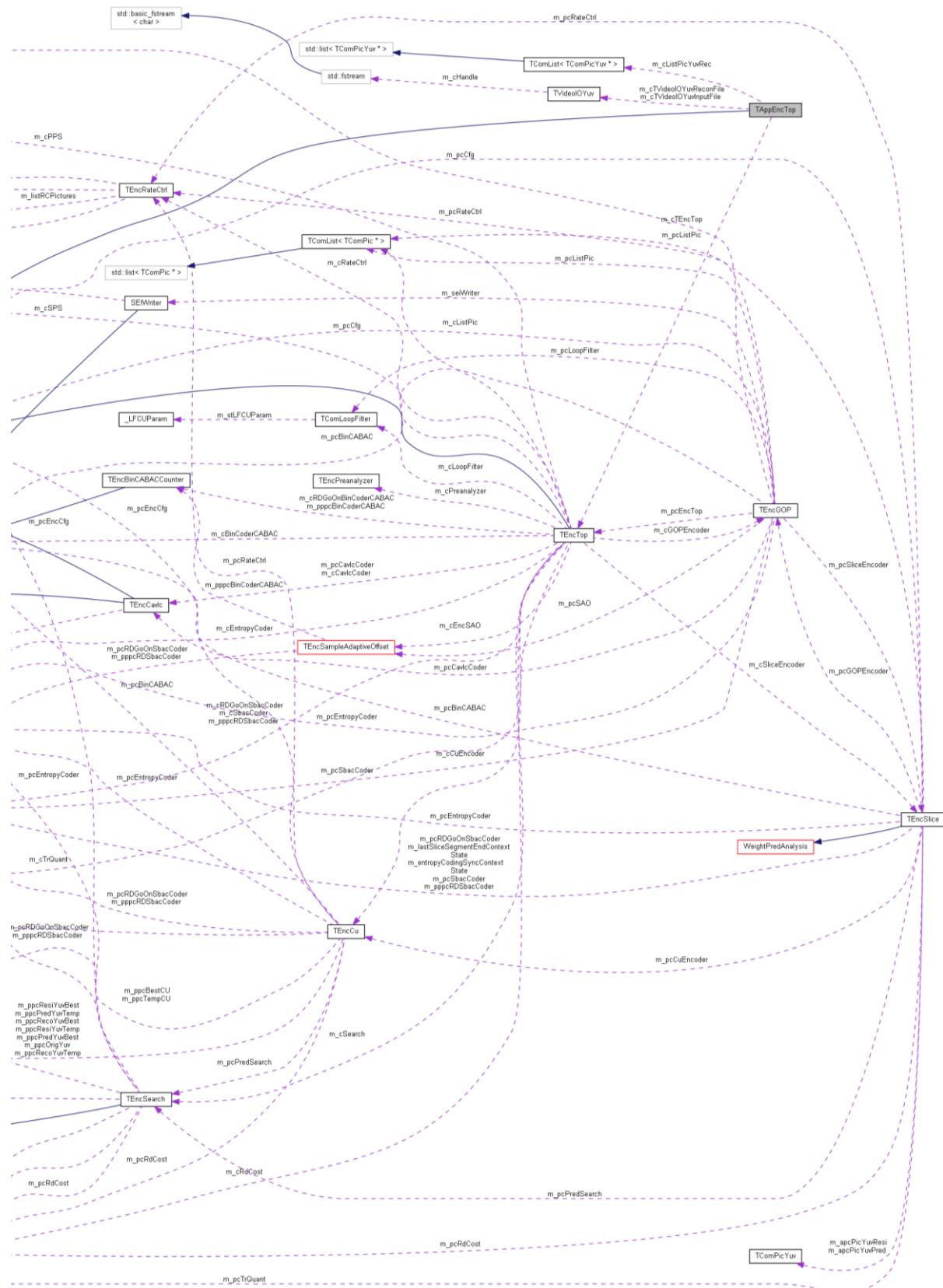
Quellenverzeichnis

1. (24. 10 2015). Von docs.opencv:
http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html abgerufen
2. Die Entwicklung der Fernseh-Technologie im Zeitraum von 10 Jahren. (25. 02 2016). Von T3N: <http://t3n.de/news/entwicklung-fernseh-technologie-letzten-10-jahren-329337/> abgerufen
3. Download And Installation. (18. 10 2015). Von Emgu CV:
http://www.emgu.com/wiki/index.php/Download_And_Installation abgerufen
4. Face Detection using Haar Cascades. (24. 10 2015). Von docs.opencv:
http://docs.opencv.org/master/d7/d8b/tutorial_py_face_detection.html#gsc.tab=0 abgerufen
5. H.265. (8. 11 2015). Von Wikipedia: <https://ru.wikipedia.org/wiki/H.265> abgerufen
6. HEVC – What are CTU, CU, CTB, CB, PB, and TB? (3. 12 2015). Von codesequoia.wordpress:
<https://codesequoia.wordpress.com/2012/10/28/hevc-ctu-cu-ctb-cb-pb-and-tb/> abgerufen
7. High Efficiency Video Coding (HEVC). (15. 11 2015). Von Fraunhofer HHI: <https://hevc.hhi.fraunhofer.de/> abgerufen
8. High Efficiency Video Coding. (8. 11 2015). Von Wikipedia:
https://en.wikipedia.org/wiki/High_Efficiency_Video_Coding abgerufen
9. Main Page. (18. 10 2015). Von emgu CV:
http://www.emgu.com/wiki/index.php/Main_Page abgerufen
10. OpenCV. (12. 10 2015). Von wikipedia:
<https://de.wikipedia.org/wiki/OpenCV> abgerufen
11. OpenCV. (12. 10 2015). Von wikipedia:
<https://ru.wikipedia.org/wiki/OpenCV> abgerufen
12. OpenCV. (12. 10 2015). Von wikipedia:
<https://en.wikipedia.org/wiki/OpenCV> abgerufen
13. Overview of the High Efficiency Video Coding. (26. 10 2015). Von Fraunhofer HHI: http://iphome.hhi.de/wiegand/assets/pdfs/2012_12_IEEE-HEVC-Overview.pdf abgerufen
14. pcs2013. (08. 03 2016). Von HEVC Encoder Optimization Based on a New RD: <http://www.pcs2013.org/GCC/Deng.pdf> abgerufen

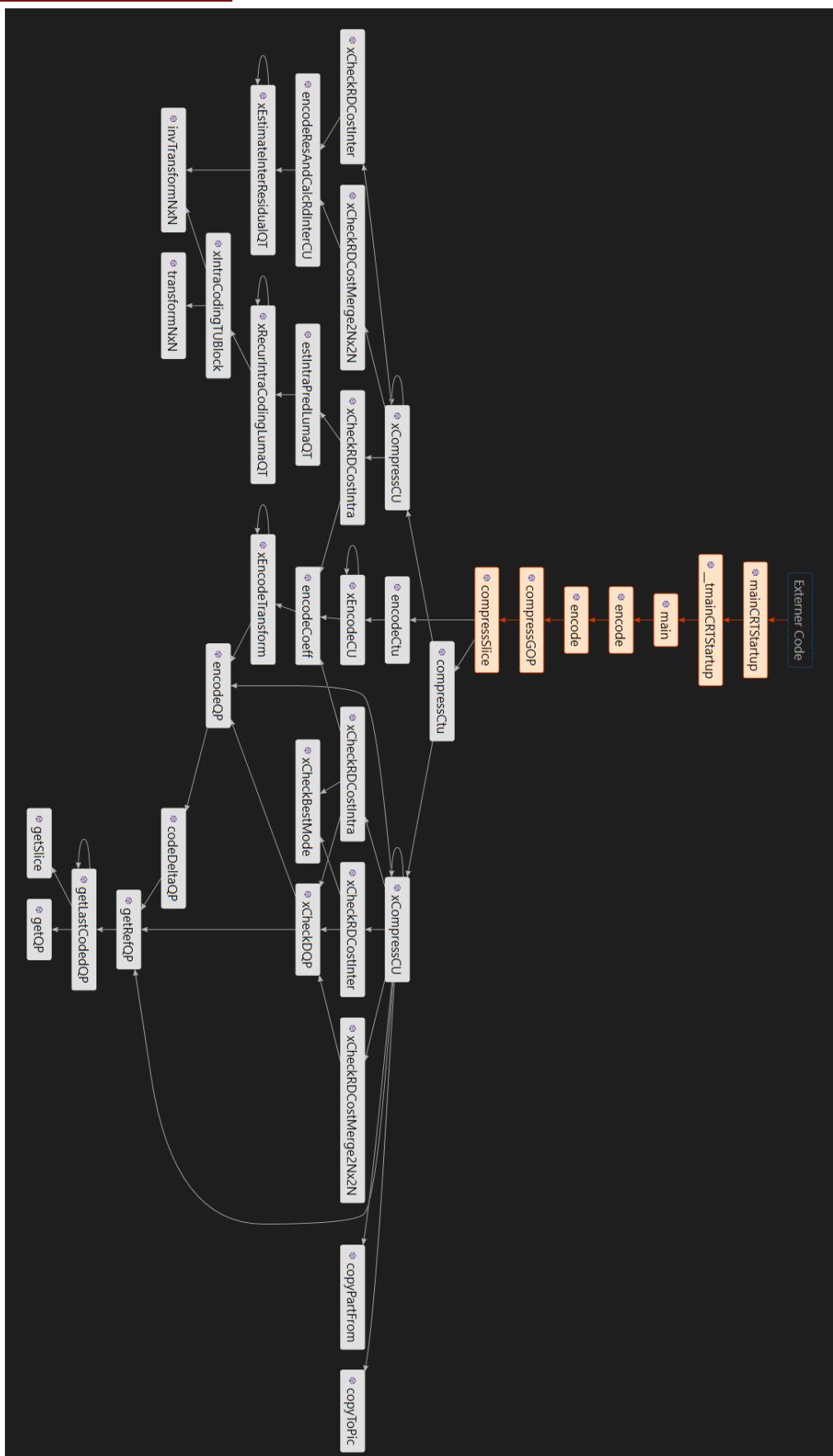
HEVC Klassendiagramm







AUFRUF DIAGRAMM



OBJEKTQPPARAMETER.CPP

```
#include "ObjectQPPParameter.h"

namespace ObjectInFrame
{
    int iFrame = 0;
    vector<ObjectQPFrame> ObjectFrames;
}

ObjectQPPParameter::ObjectQPPParameter()
{
}

ObjectQPPParameter::~ObjectQPPParameter()
{
}

ObjectQPFrame::ObjectQPFrame()
{
}

ObjectQPFrame::~ObjectQPFrame()
{
}

vector< ObjectQPFrame > readObjectQPFile(std::string filePath)
{
    vector< ObjectQPFrame > ObArr;
    vector< ObjectQPPParameter *> PaArr = new vector< ObjectQPPParameter >();
    ObjectQPFrame *object = NULL;
    ObjectQPPParameter* parameter;

    ifstream myfile(filePath);
    if (myfile.is_open())
    {
        string line;
        while (getline(myfile, line))
        {
            int tempID;
            stringstream ss(line);
            ss >> tempID;
            if (object == NULL)
            {
                object = new ObjectQPFrame;
                object->frameID = tempID;
                object->length = new int(1);
            }
            if (object->frameID != tempID)
            {
                object->parameter = *PaArr;
                PaArr = new vector< ObjectQPPParameter >();
                ObArr.push_back(*object);
                object = new ObjectQPFrame;
                object->frameID = tempID;
                object->length = ObArr[0].length;
            }

            parameter = new ObjectQPPParameter;
            ss >> parameter->X;
            ss >> parameter->Y;
        }
    }
}
```

```

        ss >> parameter->Width;
        ss >> parameter->Height;
        ss >> parameter->QP;
        ss >> parameter->Invert;
        PaArr->push_back(*parameter);
    }
    object->parameter = *PaArr;
    *(object->length) = ObArr.size() + 1;
    ObArr.push_back(*object);

    myfile.close();
}

return ObArr;
}

int findObjectFrame(int i, vector< ObjectQPFrame > Fr)
{
    for (int j = 0; j < Fr.size(); j++)
    {
        if (Fr[j].frameID == i)
            return j;
    }
    return 0;
}

void setObjectQP(vector< ObjectQPFrame > QP)
{
    ObjectInFrame::ObjectFrames = QP;
}

ObjectQPFrame getObjectQP(int i)
{
    int address = findObjectFrame(i, ObjectInFrame::ObjectFrames);
    ObjectQPFrame QP = ObjectInFrame::ObjectFrames[address];
    return QP;
}

bool isEmpty()
{
    return ObjectInFrame::ObjectFrames.size() == 0;
}

```

Geändertes Kode

FACEDETECTION (FACEDETECTION.SLN):

- Form1.cs (GUI + Aufruffunktionen)
- YUVLoad.cs (Laden von yuv Daten)
- YUVSave.cs (Speichern von yuv Daten)

HEVC (BUILD/HM_vc10.SLN)

- TAppEncoder:
 - ObjectQPParameter.cpp/.h (neue Klassen)
 - TAppEncCfg.cpp/.h (Initialisierung und Aufruf von ObjectQT Klassen, lesen von dem Parameter aus der Konfiguration Datei)
- TLibEncoder:
 - TEncCu.cpp (das Hauptkode wurde in die xComputeQP Funktion geschrieben das auf der Zeile 906 sich befindet)
 - TEncSlice.cpp (Zeile 642, lädt die Information von der Gesichtsdatei ins Memory)

CODEK TESTER (CODEK TESTER.SLN):

Es wurde erstellt um die Tests und Vergleiche einfacher durchzuführen

- Main.cs (GUI + Aufruffunktionen)
- PSNR.cs (PSNR Ausgabe mit Diagrammen)
- YUVLoad/Save... (wurden im FaceDetection benutzt)

CD