UiO : Department of Informatics
University of Oslo

OPVQ and OpenVQ

Creating free software tools for video quality assessment

Kristian Skarseth and Henrik Bjørlo

Master's Thesis Spring 2015

[simula . research laboratory]

OPVQ and OpenVQ

Creating free software tools for video quality assessment

Kristian Skarseth and Henrik Bjørlo

May 4, 2015

Abstract

PSNR is to this day the most common metric used to measure video quality, despite having been shown for decades to do so with very limited accuracy. Better metrics have been standardised, but their implementations are proprietary and expensive to license, which means that they are unavailable to most researchers.

The primary goal of this master's thesis is to develop an implementation of a standardised video quality metric that performs significantly better than PSNR, and make it available as Free Software. Subsequently, we want to provide useful abstractions to make it easy for researchers to implement their own metrics within the framework of this software.

An introduction to video quality measurement and its history is presented together with an examination of existing metrics that may be used as foundation for our implementation. We show how the metric called PEVQ standardised in ITU-T Recommendation J.247 is chosen as a basis from which we derive our own metric called OPVQ. The differences between the base and the derived metric are described in detail before we evaluate the performance of OPVQ.

OPVQ is implemented, not as a stand-alone program, but as part of a more general toolkit that is called OpenVQ. We explain the considerations that went into the design of this software, and describe the implementation in detail. We also give an introduction to how to use the program, both to assess video quality and to implement a video quality metric.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost we would like to sincerely thank Carsten Griwodz, who has tirelessly engaged in discussions and always been available to answer questions. Also, thanks to Pål Halvorsen for following our work. Thanks to Simula Research Laboratory for providing an excellent working environment, and the MPG Premium Coffee Club — for your existence we are forever grateful. Thanks to Espen Utne Landgraff for notes and corrections.

We want to thank Mads, Peder, Bård, Andreas, Kenneth and Bendik, our fellow master students and lunch companions at Simula Research Laboratory. Every day here has been a blast, engaging in interesting (and less interesting) discussions on a wide array of topics.

I, Kristian Skarseth, would personally like to thank Henrik Bjørlo, my co-worker on this project, who has been an absolute joy to work with. His endless energy and knowledge is an inspiration which I'm sure will be recognised in his coming career. I also want to thank my family for their support while pursuing my masters degree, and all the talented and lovely people I have worked with during my studies.

I, Henrik Bjørlo, would like to thank my partner in crime, Kristian Skarseth. You make a fantastic collaborator by putting in a lot of hard work (really, alot), keeping in touch and up to date, and always being open for suggestions and discussions. I have no doubt that these qualities will serve you well in the future. Thanks to my parents, Mona and Kåre, for continuous support during my studies. Finally, I want to extend my sincerest thanks to my loved ones, Kine — words cannot describe your amazing support — and our wonderful baby girl Sofia. You mean the world to me.

Sincerely,
Kristian Skarseth and Henrik Bjørlo
Fornebu, April 30, 2015

Part I

Introduction

Chapter 1

Introduction

Video streaming is responsible for a large amount of the Internet traffic generated today, through popular providers such as YouTube and Netflix. Digital video is bandwidth intensive, and video streaming over the Internet would not be feasible without efficient compression techniques. Such compression introduces loss of information, which can impact the quality of the displayed video. We are interested in the video quality as it is perceived by human viewers. Video quality assessment has traditionally been carried out by letting a set of humans give their subjective evaluation of the quality. Because of the time and human resources involved, this method of quality assessment is expensive, and in many research applications prohibitively so.

Signals like pictures and audio are represented digitally as a sequence of numbers, and we can develop metrics to measure different mathematical properties of them. We can combine such metrics to create a quality assessment model that, hopefully, correlates well with the human perception of quality.

In this thesis we expand on where video quality assessment research stands today, what issues there are, and present our own research in the field. Our research has yielded two main results. The first result is a video quality metric called *Open Perceptual Video Quality*, or OPVQ for short, which is based on a standardised metric called PEVQ. The work we have done regarding PEVQ and OPVQ is described in chapters 3–4. The other result is a video quality software toolkit called *OpenVQ* which facilitates implementation of video quality metrics. OPVQ is implemented as part of OpenVQ. All of this is released as Free Software.[1] OpenVQ is described in chapters 5–7.

[1]*What is free software?* https://www.gnu.org/philosophy/free-sw.en.html (*visited 27/4/2015*)

1.1 Background

Researchers working with video may need to measure the video quality to validate their work. Performing large scale subjective tests are currently the only way to truly know the quality of a video sequence as it is perceived by humans. Arranging such tests is however expensive and time consuming.

Digital video is fundamentally nothing else than a sequence of consecutive still image frames displayed in quick succession. Because we have digital representations of these frames available, we can quite easily perform mathematical calculations on them – they are after all nothing else than a set of numbers. Digital video is almost always stored in highly compressed formats, which in most cases means that a lot of the originally recorded data is lost. If we have available both the original data and a lossy compressed version of the same video sequence, we can compare the two signals objectively.

However, a mathematical difference between an original and a compressed signal alone is not enough to conclude that quality is degraded. What is important is how humans perceive the quality when watching the video being displayed to them, which is covered by the umbrella term *Quality of Experience* (Kuipers et al. 2010), abbreviated as QoE. This is a crucial realisation when assessing video quality. We often distinguish between *subjective* and *objective* video quality assessment; the former being assessments made by human test subjects and the latter by some sort of algorithmic approach. The performance a video quality metric is defined by its ability to accurately predict quality ratings given by human test subjects.

One of the earlier video quality metrics is known as *peak signal-to-noise ratio* (PSNR), and is nothing more than a pixel-by-pixel *mean square error* (MSE) between an original and a compressed image. Due to its simplicity, this model is still today the most widely used measure of video quality. However, PSNR can only approximate video quality as a human perceives it, and has been shown to do so with limited precision (Winkler and Mohandas 2008).

Efforts to develop quality metrics for digital video that correlate better with subjective opinion than PSNR has been ongoing since the early 1990s. The Alliance for Telecommunications Industry Solutions (ATIS) performed the first large scale validation test of such metrics in 1994–95, and since the late 1990s the Video Quality Experts Group (VQEG) has performed several similar tests resulting in various standards (M. H. Pinson et al. 2013).

In later years the best candidates from these validation tests are all developed by companies as commercially licensed proprietary software. Most researchers cannot afford to purchase expensive licenses to such metrics, and are forced to make do with outdated and less precise metrics such as PSNR.

1.2 Objectives

As indicated in the introduction to this chapter, there is a lack of good video quality metrics that researchers can use freely in their work. Our primary goal is to improve this situation by developing an adapted implementation of a modern video quality metric, and make it available as Free Software. This adaptation is to be based on one of the metrics standardised as a result of VQEG's validation efforts.

Subsequently, we want to provide a software toolkit with useful abstractions that can help researchers in implementing their own metrics.

1.3 Research Method

The research presented in this thesis is the result of a process that to a large degree conforms to the *design* paradigm for the computer science discipline, as defined by the ACM Education Board (Comer et al. 1989). The design paradigm has roots in engineering, and consists of stating the requirements and specifications, designing and implementing a system accordingly and testing the system.

We are applying this design paradigm to two different, but related systems; 1. a video quality metric; and 2. a general software toolkit for video quality assessment, where the latter is intended to facilitate the implementation of the former.

1.4 Significance of Study

A free and open source metric for measuring video quality that correlates well with human perception of quality, can be of use to anyone working with video quality. It can be used within QoE research, hereunder video streaming services and video codecs. Better tools and metrics to assess video quality without requiring significant funds for commercial licenses represents an improvement to the situation for researchers in the field.

1.5 Limitations and Scope

Video quality metrics are evaluated using subjective test scores. The results of such evaluation is therefore only conclusive within the limits of the subjective test data used. Test video sequences are generally around 10 seconds long (ITU-R 2012; ITU-T 2008b), and subjecitve scores are collected for a limited set of degradation types. In addition, each metric will have its own limitations which we will be forced to follow as well. We cover the specific limitations of our chosen metric in section 3.1.1.

As we discuss in section 4.1.5, there is a lack of freely available datasets with subjective test scores and corresponding test video sequences. Such datasets are required to test and to validate our metric, and the lack of data limits our ability to fully validate our implementation.

1.6 Outline

In chapter 2 we give a brief introduction to objective video quality assessment and the history of the field. We cover the current status of video quality research, and describe how and why we singled out PEVQ as the inspiration for our own metric OPVQ.

In chapter 3 we cover the design of the OPVQ metric and how it is derived from PEVQ, and the metric is tested and evaluated in chapter 4.

Chapters 5, 6 and 7 cover details about the design, implementation and usage of the OpenVQ toolkit.

We give conclusions about the achieved results in chapter 8, along with some pointers for further work. In the final chapter we discuss some of our subjective experiences with the project.

1.7 Summary

Video QoE is important when working with video services. Assessing the quality of video provided through a service using subjective testing is too expensive and time consuming. On the other hand, free objective metrics do not provide satisfactory results. Objective models producing results that correlate well with human perception are subject to expensive commercial licensing terms, and most researchers cannot afford them. As an effort to provide researchers with a modern tool that provides satisfactory performance we develop a free, open source, full reference objective video quality assessment metric that we call OPVQ. It is inspired by the PEVQ metric standardised through VQEG's validation tests. In addition, our program, called OpenVQ, will work as a toolkit where additional video quality metrics can be implemented with ease.

Chapter 2

Related Work

This chapter will provide a short introduction to how video quality measurement works, as well as a brief historical introduction to efforts for standardising objective video quality metrics. We then look at currently avilable objective video quality metrics. Based on an evaluation of these metrics we select one that we feel is best suited for a free and open source implementation.

2.1 Understanding Video Quality Measurement

The first and most obvious distinction one has to make within the field of video qality measurement is the difference between subjective and objective testing. Subjective quality measurement means that human test subjects view a series of video sequences and give their opinion of the quality of the video. Configuration of the room where the testing is performed is crucial, and everything from viewing distance, monitor size and colour calibration must be perfectly tuned. Jiménez Bermejo (2012) summarises different methods for subjective assessment as follows:

> "Subjective testing for visual quality assessment has been formalised in ITU-T Rec. P.910 (ITU-T 2008c) and ITU-R Rec. BT.500 (ITU-R 2012)…
>
> - **Double Stimulus Continous Quality Scale (DSCQS)**, where subjects rate short sequence pairs, consisting of a test and corresponding reference video.
> - **Double Stimulus IMpairment Scale (DSIS)**, also referred to as Degradation category Rating (DCR), where subjects rate the amount of impariment in the test video with respect to the known reference video.

- **Single Stimulus Continous Quality Evaluation (SSCQE)**, where subjects watch a program of typically 20-30 minutes duration and continously rate the instantaneously preceived quality on a slide.
- **Absolute Category Rating (ACR)**, a single-stimulus method, where subjects rate each test video individually without comparison to an explicit reference.
- **Pair Comparison (PC)**, where the test videos from the same scene but different conditions are paired in many possible combinations and subjects make a preference judgement for each pair.

. . . ”

Methodology for subjective assessment of video quality has been formalised in ITU-T Rec. P.910 (ITU-T 2008c) and ITU-R Rec. BT.500 (ITU-R 2012).

Objective measurement is performed solely by a computer. Objective metrics are categorised as either *full reference* (FR), *reduced reference* (RR) or *no reference* (NR), depending on the amount of reference data needed. The reference data in this context is the original source sequence from which the sequence under assessment is derived. As implied by the name, FR metrics require access to the entire reference sequence, RR metrics require a subset of the data from the reference sequence, and NR metrics do not need any reference data. Not surprisingly, FR models are able to achieve the highest correlation with subjective test results. For this reason we have chosen to limit our reserach to FR metrics.

We also wish to group each of the three objective approaches into two additional groups; perceptual and non-perceptual. In later years, researchers have found that simple metrics, such as PSNR, can predict subjective quality ratings to a limited degree, but in order to get truly good correlation with subjective scores it is necessary to develop metrics that attempt to view the video the same way the human visual system (HVS) does. Perceptual metrics are developed using our understanding of the HVS, while non-perceptual metrics are only mathematical models with no consideration of human perception.

In the next section we give an overview of validation tests of objective video quality metrics performed the past 20 years.

2.2 Validation of Video Quality Metrics

The 1990's saw the rise of digital video codecs. This quickly created a need for models that could say something useful about the quality of digitally encoded video. A number of efforts was made to create better objective video quality metrics. Furthermore, these new metrics needed to be validated to determine their correlation with human perception of quality.

The first large scale validation test was performed by the Alliance for Telecommunications Industry Solutions (ATIS) in 1994-95, which resulted in the two American National Standards Institute (ANSI) standards T1.801.01 and T1.801.03.[1] The test did not standardise any video quality metrics, but it did result in, among other things, a set of publicly available source video sequences that can be used for further testing.

In the late 1990's participants from the International Telecommunications Union (ITU) were drawn together to form the Video Quality Experts Group (VQEG), with a goal to advance the field of video quality research (M. H. Pinson et al. 2013). This group has since made a number of efforts to validate the performance of new objective video quality metrics for the purpose of standardisation, and in 1997 the first VQEG meeting found place in Turin.

To date, VQEG has performed several large testing phases. From 1999 to 2000 their first phase, the full reference television (FRTV) Phase I, was conducted by the Independent Lab Group (ILG). It was designed for testing full- and no-reference standard definition television quality, however, none of the NR models made it to the testing phase. The conclusion from the test was that none of the submitted models were statistically better than PSNR.

Following the FRTV Phase I came FRTV Phase II (2002-2003), the multimedia Phase I (2007-2008), reduced reference/no reference television (RRNR-TV) Phase I (2008-2009) and the high definition television (HDTV) test (2009-2010). All tests were conducted by ILG with some proponents involved in certain cases. Eight FR models were published in a first rendition of ITU-T Rec. J-144 following FRTV Phase I (2001). FRTV Phase II published a revised version of ITU-T Rec. J.144 as well as ITU-T Rec. BT.1683, where four FR models were standardised. In both phases all NR models were withdrawn.

[1]Institute for Telecommunication Sciences — National and International Standards http://www.its.bldrdoc.gov/resources/video-quality-research/standards/national-and-international-standards.aspx (*visited 6/11/2014*)

Test-phase name	Org.	Date	Resolutions	Standards documents
T1A1	ATIS	1994–1995	NTSC	T1.801.03 & T1.801.01
FRTV Phase I	VQEG	1999–2000	NTSC, PAL	ITU-T Rec. J.144
FRTV Phase II	VQEG	2002–2003	NTSC, PAL	ITU-T Rec. J.144 & ITU-R Rec. BT.1683
Multimedia	VQEG	2007–2008	VGA, CIF, QCIF	ITU-T Rec. J.247, ITU-R BT.1866, ITU-T Rec. J.246 & ITU-R BT.1867
RRNR-TV Phase I	VQEG	2008–2009	NTSC, PAL	ITU-T Rec. J.249
HDTV	VQEG	2009–2010	1080i, 1080p	ITU-T Rec. J.341 & ITU-T Rec. J.242

Table 2.1: Overview of research efforts into objective video quality metrics

Following the Multimedia Phase I, FR models from Nippon Telegraph and Telephone Corporation (NTT), OPTICOM, Psytechnics and Yonsei University were standardised in ITU-T Rec. J.247 and ITU-R BT.1866. One RR model, from Yonsei University, was standardised in ITU-T Rec. J.246 and ITU-R BT.1867. Again no NR models were standardised. The RRNR-TV Phase I test standardised 3 RR models in ITU-T Rec. J.249, and the HDTV test in 2009-2010 standardised two FR models in J.341 and one RR model in J.242. Two NR models were mentioned in VQEG's final report for the HDTV test, but neither were standardised.

For more information on the history of VQEG's validation tests, see M. Pinson, Staelens, et al. (2013). Table 2.1 is an overview of the test phases described in this section. It includes the test phase name, organisation that performed the test, date, resolutions tested and which standards were produced from the test phases. In the following sections we discuss various metrics that are available today. We review the standardised metrics and select one as basis for our own implementation.

2.3 Non-Perceptual FR Metrics

Early video quality metrics did not take the human visual system into account. The two most recognised early metrics are PSNR and SSIM. In the following sections we briefly describe these two metrics. While neither play a big part in our work, they serve as a useful benchmark and we use them in our results evaluation in chapter 4.

2.3.1 Peak signal-to-noise ratio

Peak signal-to-noise ratio (PSNR) is probably the most used objective video quality assessment metric today, even though its correlation with subjective tests is limited as explained by Huynh-Thu and Ghanbari (2008). The reason so many still use it, despite its shortcomings, can be explained by its simplicity. PSNR is nothing more than a mean squared error (MSE) as shown in equation 2.1.

$$MSE = \frac{1}{WH} \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} |SRC(i,j) - PVS(i,j)|^2 \tag{2.1}$$

The score is then represented in dB as defined in equation 2.2, where MAX_I^2 is the highest possible value of each pixel squared (typically $2^{\# bits\ per\ pixel - 1}$).

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \tag{2.2}$$

- W and H represent picture width and height.
- SRC and PVS is the source and processed video sequences respectively

Small spatial, colour or temporal shifts do not necessarily affect a subjective viewer noticeably, but it can have a large affect on the PSNR score, which is based on pixel-by-pixel comparison with the reference. The PSNR correlation to subjective score can therefore be improved by performing alignment before the MSE calculation is done. We discuss sequence alignment in detail in section 3.3. VQEG use a slightly modified version of PSNR, standardised in ITU-T Rec. J.340 (ITU-T 2010), where values for constant shifts in the spatial, temporal and luminance domain are calculated, as a minimum acceptable performance benchmark.

2.3.2 Structural Similarity

Structural similarity index is not much more complex than PSNR, but as explained in Z. Wang et al. (2005) it attempts to extract structural information from a visual scene. This is also a feature of the human visual system (HVS). Unlike PSNR, which estimates perceived errors, SSIM estimates perceived change in structural information. While the creators of SSIM claim it is a significant improvement over PSNR, Dosselmann and Yang (2011) explain how SSIM in fact is not so different from PSNR, and state that SSIM does not fill “the enormous gap that continues to exist between an automated measure of quality and that of the human mind".

SSIM is calculated by applying equation 2.3 to windows of an image. A window is a subset of pixels within the image and a typical window size can be 8×8 pixels. The resulting value from the equation is in the range $[-1, 1]$, and is averaged over the number of windows in the image. The values x and y represents a window from SRC and PVS respectively.

$$SSIM(x,y) = \frac{\left(2\mu_x\mu_y + c_1\right)\left(2\sigma_{xy} + c_2\right)}{\left(\mu_x^2 + \mu_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)} \tag{2.3}$$

where

- μ_x and μ_y are the averages of x and y
- σ_x^2 and σ_y^2 are the variances of x and y
- σ_{xy} is the covariance of x and y
- c_1 and c_2 are constants

For a video sequence the final value indicating the quality of the video is generated by averaging each frame value over the number of frames. More detailed information on the mathematics behind SSIM can be found in (Z. Wang et al. 2005) and (Wang Z., Bovik, A. C. and Sheikh, H. R. and Simoncelli, E. P. 2004).

Like with PSNR, spatial, temporal and colour misalignment between the SRC and PVS can affect the results, and therefore such alignment should be performed before the SSIM equation is calulated.

2.4 Perceptual FR Metrics

In order to improve the performance of objective metrics it is necessary to take the HVS into account. Since VQEG's FRTV Test Phase II, the top performing metrics have been perceptual metrics in that they, at least to some degree, perform calculations based on knowledge about the HVS. In this section we present the perceptual models we have considered, most of whom have been standardised as a result of VQEG's validation efforts.

2.4.1 Models from ITU-T Rec. J.144

ITU-T Rec. J.144 was first published in 2001 as a result of the FRTV Phase I, and then rebulished in 2004 after the FRTV Phase II. It was in the 2004 edition where four models were standardised. VQM, developed by a branch of the National Telecommunications and Information Administration (NTIA), was the top performing metric in the test phase, and it has been freely available for anyone to use since then, both commercially and non-commercially. VQEG hosts the VQM website, where the software is freely available for download.[2]

The VQM model is much more complex than PSNR and SSIM, and includes spatial, temporal and colour alignment steps before what they call the *General Model Parameters* are calculated. The parameters consist of 7 independent values, four from the spatial gradients of the luminance component, two from features extracted from the vector formed by the two chrominance components and one based on the product of features that measure contrast and motion. From these parameters the *General Model* calculates a final score where the parameters are linearly weighted (M. Pinson and Wolf 2004).

2.4.2 Models from ITU-T Rec. J.247

The ITU-T Rec. J.247 standardises four separate full reference models that all significantly outperform PSNR (ITU-T 2008a). The models were evaluated for VGA, CIF and QCIF resolutions.

All four models have a detailed description attached with the standardisation document which makes it possible to deduce if any of the metrics are feasible to implement within the time constraints of a master thesis.

Table 2.2 shows the Pearson correlation for each of the four metrics as presented in the standard, with PSNR added for comparison. The OPTICOM metric, named PEVQ, and Psytechnics metrics clearly stand out as the two best metrics. PEVQ has the best average and minimum correlation for both VGA and QCIF resolutions, while Psytechnics stand out with the highest amount of rank 1 occurences, as well as the best results for CIF resolution. Judging from these results alone either of them seems like the best choice among the four.

When looking closer at the description we see that the PEVQ model is explained well with both text and mathematical equations. This makes

[2]VQEG — Video Quality Metric (VQM) http://www.its.bldrdoc.gov/resources/video-quality-research/software.aspx (*visited 17/2/2015*)

VGA	NTT	OPTICOM	Psytechnics	Yonsei	PSNR
Avg. correlation	0.786	0.825	0.822	0.805	0.713
Min. correlation	0.598	0.685	0.565	0.612	0.499
Occurences at rank 1	8	10	11	10	3
CIF					
Avg. correlation	0.777	0.808	0.836	0.785	0.656
Min. correlation	0.675	0.695	0.769	0.712	0.440
Occurences at rank 1	8	13	14	10	0
QCIF					
Avg. correlation	0.819	0.841	0.830	0.756	0.662
Min. correlation	0.711	0.724	0.664	0.587	0.540
Occurences at rank 1	9	11	12	4	1

Table 2.2: J.247 model performance overview

it possible to relatively easily get a general overview of the amount of code that has to be written. Our initial analysis suggested it would be relatively doable to create an implementation of the metric within our time constraints.

PEVQ's high stability with good correlation, paired with both text and equations to describe the metric, makes it our preferred choice from the ITU-T Rec. J.247 standard.

2.4.3 Models from ITU-T Rec. J.341

The resolution known as Full HD[3] has become the de facto standard for delivering digital video[4,5], and it is the native resolution for most of today's television panels. One could argue that any effort to provide an implementation of a video quality model should have support for at least Full HD resoltion.

ITU-T Rec. J.341 (ITU-T 2011) standardises one new full reference model. This model was evaluated against PSNR for 1080i and 1080p

[3]Full HD: 1920x1080

[4]Broadcasting standards ATSC (America), DVB (Europe), Optical standard Blu-ray Disc, Internet content from YouTube and Netflix all use 1080p or 1080i resolutions

[5]1080p — Wikipedia, The Free Encyclopedia http://en.wikipedia.org/wiki/1080p (*visited 10/11/2014*)

Metric	PSNR	SwissQual
Superset RMSE	0.71	0.56
Top performing group total	1	5
Better than PSNR total	-	4
Better than superset PSNR	-	Yes
Superset correlation	0.78	0.87

Table 2.3: J.341 model performance

television signals[6] with two different framerates; 25 fps and 29.97 fps.[7] However, the model in J.341 does not come with a formal description in the recommendation. The author, SwissQual AG, have instead released reference source code attached to the standard. This reference source code is protected under copyright, and it is subject to strict licensing from the owner. Any potential user, including researchers, must apply to the owner for a license to reproduce, modify and/or use the software, the conditions of which is not presented in the reference code's copyright notice.

Software copyright of this kind is a problematic intellectual property issue, but this is outside the scope of this thesis. Any implementation of this model will need to be carried out by at least two separate persons; one of which has to analyse the reference code and describe it formally, while the other in turn can use this formal description to create a new implementation. This new implementation would constitute original source code, licensable at the author's discretion.

In addition to being challenging and most likely time consuming to implement, the only information we get on the metrics performance is summarised in table 2.3. While it is clearly stated in the standard document that the metric outperforms PSNR, the limited information on the performance and the fact that it is not compared to any other perceptual models makes it difficult to make judgements about its performance.

2.4.4 Other Metrics

There are perceptual metrics available today that has not been standardised, but due to time constraints we have been forced to focus on the

[6]1080i: 1920x1080 interlaced, 1080p: 1920x1080 progressive

[7]*fps*: frames per second

standardised metrics in our research. We would however like to briefly mention the MOtion-based Video Integrity Evaluation index (MOVIE). It is a FR metric developed by K. Seshadrinathan and A. C. Bovik (Seshadrinathan and Bovik 2010). The metric has not been submitted to any of VQEG's test phases and has therefore not been standardised. The authors ran their own correlation testing and compared results with multiple variants of PSNR, SSIM and VQM on the LIVE Video Quality Database. In these tests the metric outperformed all other metrics (Seshadrinathan, Soundararajan, et al. 2010). Even though we have not had the time to look into this metric in more detail, it could be interesting to compare it to OPVQ in the future.

ITU-T Rec. J.343 was approved in November 2014 and is the result of VQEG-JEGs hybrid-FR test phase. The test plans are available, but the standardisation documents are currently restricted to TIES users[8] and we do therefore not have access to them. Because we do not have access to the standardisation documents, and the fact that the standard was not approved when we started our project, J.343 was never a possible candidate for us. The results may however contain useful and relevant information and should be examined when made available.

2.5 Review of candidates

VQEG's FRTV Phase II resulted in VQM, the first standardised metric to conclusively outperform PSNR. Since then, the top performing, and consequently standardised metrics, are licensed commercially as proprietary software. Descriptions of the metrics are however published with the standardisation documents. In this section we review the metrics published as a result of VQEG's Multimedia Phase I and HDTV Phase I in ITU-T Rec. J.247 and J.341, and we choose one of these metrics as the basis for our own implementation.

We have been unable to find any surveys where all the metrics we have talked about in this chapter are compared. Seshadrinathan, Soundararajan, et al. (2010), Y. Wang (2006) and Martinez-Rach et al. (2006) are all surveys where open and free metrics, most commonly PSNR, SSIM and VQM, are tested, but none of them include metrics from J.341 or J.247. In addition, OPTICOMs PEVQ metric which we concluded was the best choice from J.247 is not developed for the same resolutions as SwissQual AGs metric from J.341. This means the two

[8]TIES is a set of networked information resources and services offered to ITU members http://www.itu.int/TIES/ (*visited 20/4/2015*)

Feature	**OPTICOM** (J.247, *PEVQ*)	**SwissQual** (J.341)
Better than PSNR	Yes	Yes
Validated for HD	No	Yes
Formal description	Yes	No
Source code	No	Yes (copyrighted)
Patented	Partially	?
Implementable	Single stage	Multi stage

Table 2.4: Comparison of candidate models

metrics have not been tested against the same video sequences and that the little performance information we have on SwissQual AGs metrtic cannot be compared directly to the PEVQ metric performance results.

The absolute lack of independent tests outside of VQEG comparing the J.341 and J.247 metrics with other free metrics is a testament to the problem that researchers working with improving video quality measurement are unable to afford the latest standardised metrics.

The model from J.341 has the clear advantage that it has been validated for high definition resolution. However, the disadvantage is that implementing it comes at the cost of a laborious and potentially error prone approach. The more formal description of PEVQ in J.247 makes it more compelling from an implementation standpoint. Another observation is that just because PEVQ has not been validated for HD resolutions, doesn't mean that it can't provide useful and even good results. Adding support for HD could be a potential direction for further research if the implementation is a success, subject to validation against subjective test data.

We can see that there are pros and cons associated with both candidates, as summarised in table 2.4. For the resources we have available, the required process of implementing SwissQual's model in two separate phases is unfortunately a deal breaker, and we are left with PEVQ as the only option.

2.6 Availability of subjective test data

The method used for validation testing of the proposed models in VQEG's Multimedia Phase I is well documented in the ITU-T Rec. J.247. However, the actual source sequences used are not freely available, due to licensing restriction, but also due to multi party non-disclosure agreement signed by the proponents and the research institutions that helped conduct the validation tests, referred to as VQEG's Independent Lab Group (ILG).[9,10] We do not own a PEVQ license from OPTICOM, so comparing the results directly is not a viable option either. This limits the extent to which we have been able to validate our own implementation against PEVQ. Details about this and performance evaluation in general can be found in chapter 4.

[9]VQEG — Multimedia Phase I http://www.its.bldrdoc.gov/vqeg/projects/multimedia-phase-i/multimedia-phase-i.aspx (*visited 10/11/2015*)

[10]VQEG — Independent Lab Group (ILG) http://www.its.bldrdoc.gov/vqeg/projects/ilg.aspx (*visited 10/11/2015*)

Part II

OPVQ – The Open Perceptual Video Quality metric

Chapter 3

From PEVQ to OPVQ

As we have discussed previously we are lacking an open, free to use, video quality metric that attempts to mimic human perceptive mechanisms. We have decided to use the PEVQ metric from ITU-T Rec. J.247 as the basis for our implementation. In this chapter we provide a brief description of the PEVQ model as a whole, and present our final design and how it is derived from PEVQ.

3.1 PEVQ

PEVQ (*Perceptual Evaluation of Video Quality*) as described in J.247 has five main steps. The first step is a simple pre-processing step consisting of some predefined cropping based on the video resolution. Next, six statistical aspects are calculated over the source and processed video sequences, based on which the sequences are coarsly aligned temporally. The luma[1] levels are also corrected at this point, using histogram correction. In the third step, fine alignment is done both in the spatial and temporal domain, i.e. the sequences should at this point be aligned from start to finish. Chroma correction is also performed using histogram correction. The fourth step is the distortion analysis which generates five separate indicators that in the sixth and last step are weighted using parameters specific to the resolution, and mapped to a single mean opinion score (MOS).

3.1.1 Limitations with PEVQ

At present there are patents registered by OPTICOM GmbH regarding temporal alignment of video sequences. Temporal alignment is a major

[1]PEVQ's internal working format is $Y'C_BC_R$ 4:4:4 (ITU-T 2008a)

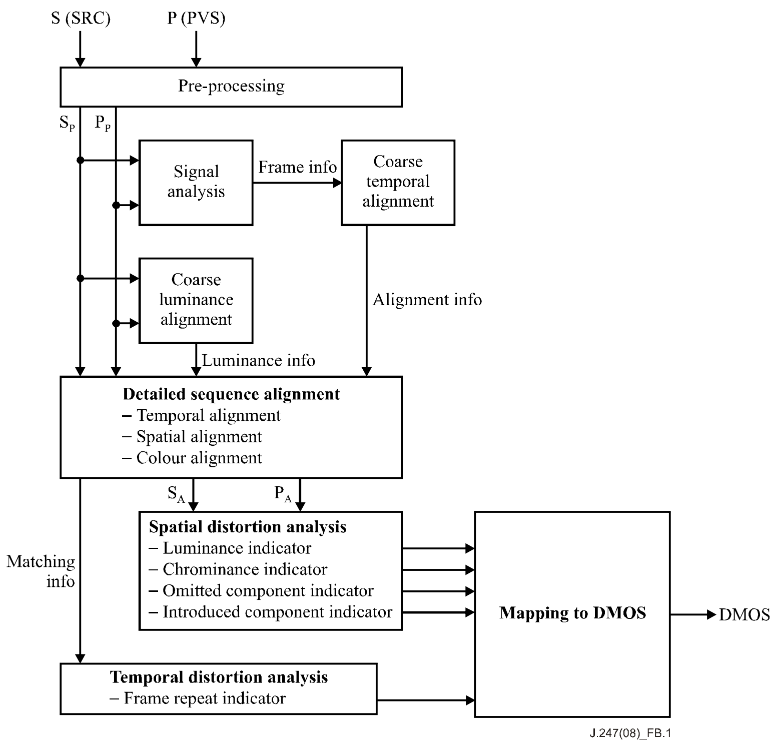


Figure 3.1: Overview of the PEVQ model

part of PEVQ as described in J.247, so parts of the model may need to be substituted with an original mechanism, to avoid legal issues when publishing our implementation.

PEVQ as described in J.247 provides support for only a limited set of spatial resolutions (VGA, CIF and QCIF). Adding support for other resolutions is possible, but may require significant effort. Due to the limited availability of testing data, as discussed in section 2.6, we focus on verifying OPVQ with VGA resolution video sequences (chapter 4).

In addition, any objective video quality metric based on digital signal processing will only be able to reliably detect and account for a limited set of errors or degradations. Limitations as to what type of errors PEVQ and the other metrics evaluated in J.247 were validated for is defined in the VQEG Multimedia Phase 1 Final Report (VQEG 2008), and further specified in the ITU-T J.247 recommendation (ITU-T 2008a). These limitations include video framerate, minimum and maximum bitrates and frame freeze or skip lengths. These properties limit the amount of relevant test sequences we can use to validate OPVQ, but it also explains what type of degradations we should expect our implementation to handle.

3.2 Pre-processing

The purpose of pre-processing is to crop each frame of the PVS and SRC sequences as the outer edges of the frames are generally not noticed by a human viewer. To crop the three resolutions officially supported by PEVQ we use equations (3.1-3.2).

$$S_p[i,j,t] = S[i+c, y+c, t] \forall i \in [0..W-2c\rangle, j \in [0..H-2c\rangle \qquad (3.1)$$

$$P_p[i,j,t] = P[i+c, y+c, t] \forall i \in [0..W-2c\rangle, j \in [0..H-2c\rangle \qquad (3.2)$$

The constant c is the crop size which is defined explicitly for each resolution.

$$c_{VGA} = 12 \qquad c_{CIF} = 6 \qquad c_{QCIF} = 3 \qquad (3.3)$$

3.3 Sequence alignment

Full reference algorithms require a source and a processed video sequence. Compression may give rise to small changes in the resulting video that do not represent quality degradations in the eyes of human viewers. For instance, imagine that every frame in a PVS is shifted 1 pixel to the right relative to SRC. A human viewer will not detect any degradation in quality, but a FR algorithm making a pixel-by-pixel comparison will see lots of errors. To avoid such a scenario, we need to perform sequence alignment. Figure 3.2 illustrates a right shift of 1 pixel in the left image which is barely noticeable without magnifying and comparing the left edges of the two pictures.

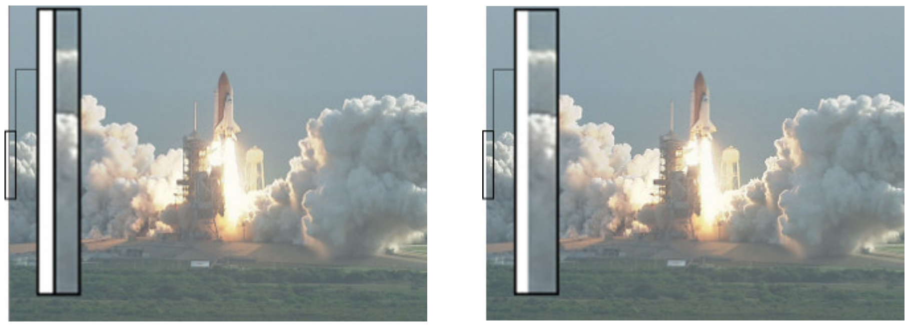


Figure 3.2: Unnoticeable black edge on left image (PVS) as a result of a shift by 1 pixel. Right image act as the SRC

Misalignment may need to be eliminated in various domains. Maybe the most obvious example is temporal misalignment. If our PVS for example misses the first frame of the source, we can't compare its first frame with the first frame in SRC. Compression techniques such as downampling and downscaling can lead to spatial misalignment when upscaled to display resolution. Colour will also be subject to slight change as an effect of lossy compression. In this section, we expand on the different sequence alignment procedures employed in OPVQ.

3.3.1 Temporal Alignment

A significant part of the PEVQ algorithm deals with temporal alignment, and OPTICOM holds patents pertaining to their temporal alignment mechanism, which prevents us from including it in our free, open source implementation. This leaves us with the option to *a*) design our own temporal alignment mechanism; or *b*) leave temporal alignment out all together. In order to make the decision, we analysed the scope of the temporal alignment algorithm.

J.247 lists "transmission errors with packet loss" and "Temporal errors (pausing with skipping) of maximum 2 seconds" as test factors (ITU-T 2008a). However, modern digital video delivery systems generally give priority to continuous playback by pre-buffering and proactively adapting the bitrate to the current network conditions. More often than not, this runs on top of reliable transport layer protocols such as TCP. Furthermore, the video codecs themselves encode timing information into the video. Our understanding is that these factors to a large degree mitigate temporal errors of this kind.

Temporal errors still occur in modern streaming scenarios in the case of buffer underrun events. In these cases, the playback will freeze as the video is re-buffered, but it will not skip ahead when playback is resumed. Such errors are common, and the subject of much research, as they can severely impact the viewer's QoE. However, given the nature of VQEG's test factors in J.247, PEVQ's temporal alignment and distortion analysis mechanisms do not efficiently handle such errors.

Based on this analysis, we decided to leave out the parts of PEVQ dealing with temporal alignment and errors. These parts are the steps named Signal Analysis, Coarse Temporal Alignment, Fine Temporal Alignment and Temporal Distortion Analysis in figure 3.1. Seeing as this is a significant part of the algorithm, leaving these parts out reduces the total number of steps and simplifies the flow of the algorithm, in turn giving rise to potentially reduced execution time.

3.3.2 Spatial Alignment

Spatial alignment is performed on a frame-by-frame basis to detect spatial shifts in the degraded video. In the PEVQ description in J.247 these potential offsets are found using a mean square error (MSE) approach as shown in equation 3.4.

$$f(\delta_x, \delta_y, t) = \sqrt{\frac{1}{Norm} \sum_{i=max(0,\delta_x)}^{min(W,W+\delta_x)-1} \sum_{j=max(0,\delta_y)}^{min(H,H+\delta_y)-1} |P_t(i+\delta_x, j+\delta_y, t) - S_t(i,j,t)|^2} \tag{3.4}$$

where

$$Norm = (\min(W, W+\delta_x) - \max(0,\delta_x))(\min(H, H+\delta_y) - \max(0,\delta_y)) \tag{3.5}$$

The goal is to find the (δ_x, δ_y) pair that minimises f, i.e. has the lowest MSE, and therefore probably the most accurate alignment (3.6).

$$f(\delta_x[t], \delta_y[t], t) \rightarrow min \tag{3.6}$$

$$\delta_x \in \{-1, 0, 1\}, \quad \delta_y \in \{-1, 0, 1\} \tag{3.7}$$

These minimum error offsets $(\delta_{min,x}, \delta_{min,y})$ are used to correct PVS frames, while the SRC frames remain unchanged (3.8 and 3.9).

$$S_{s,\mu}[i,j,t] = S_{p,\mu}[i,j,t] \tag{3.8}$$

$$P_{s,\mu} = \begin{cases} P_{t,\mu}[i+\delta_{min,x}[t], j+\delta_{min,y}[t], t] & \text{if } \begin{array}{l} 0 \leq (i+\delta_{min,x}[t]) < W \text{ and} \\ 0 \leq (j+\delta_{min,y}[t]) < H \end{array} \\ S_{s,\mu}[i,j,t] & \text{otherwise} \end{cases} \tag{3.9}$$

We found this description to be overly complicated. As the results of f (3.4) are only used comparatively, taking the square root is not necessary and represents waste of execution time. Also, there is a lot of bounds checking going on. As we have cropped the frames at least 3 pixels (sec. 3.2), and we only need to move 1 pixel outside the cropped bounds(eq. 3.7), we know that we never leave the bounds of the existing data. Because W and H are no longer affected by the δ offsets, the summation ranges can be simplified. In addition, it eliminates the need to take the mean, because $Norm$ will always be equal to WH. This leaves us with the following modified f and aligned PVS frame $P_{s,\mu}$:

$$f(\delta_x, \delta_y, t) = \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} |P_t(i+\delta_x, j+\delta_y, t) - S_t(i,j,t)|^2 \tag{3.10}$$

$$P_{s,\mu} = P_{t,\mu}[i+\delta_{min,x}[t], j+\delta_{min,y}[t], t] \tag{3.11}$$

3.3.3 Colour Alignment

Colour alignment attempts to find any changes in the luma and chroma channels of PVS relative to SRC and correct them. In short, the algorithm analyses the colour distribution of both sequences, and tries to correct any shifts present in PVS. For instance, there may be a peak at the value 100 in the luma channel of SRC. The same peak should be present in PVS as well, but it may be slightly shifted up or down. The colour correction algorithm creates a map, or *correction curve*, that is applied to PVS, after which the colour distribution of PVS should be more similar to that of SRC. This process is known as *histogram matching* (Shapira et al. 2013).

It's worth noting that PEVQ performs luma alignment twice, the first time as part of the temporal alignment procedure. Since temporal alignment is not part of OPVQ, the luma and chroma alignment steps are merged into a single colour alignment step that is performed after spatial alignment.

Histogram matching

An image frame is a countable set of pixel values. A histogram is created by counting how many pixels hold each value — or the *distribution* of the pixel values. We end up with what's often described as a set of *bins* or *buckets*, one for each possible value. If we have 8-bit values, we get $2^8 = 256$ different bins, regardless of the size of the image. The sum of all the bins will be the same as the number of pixels in the image, i.e. the spatial resolution, because each pixel fall into exactly one bin.

We can normalise a histogram by dividing by the spatial resolution. The sum of the normalised histogram is 1. The normalised histogram defines the empirical probability density function ($epdf$) of the frame's pixel values. We can cumulate the normalised histogram to get the empirical cumulative distribution function ($ecdf$).

The colour alignment process in PEVQ finds a transformation that matches the $ecdf$ of PVS as closely as possible to the $ecdf$ of SRC. Conceptually, the process transforms the x-axis of the cumulative histogram from PVS in a non-linear fashion so that it matches the cumulative histogram of SRC as closely as possible.

First, we traverse the sequences to produce histograms, non-cumulative and cumulative, of the luma and chroma channels. Next, we use these histograms to generate a correction curve which is applied to PVS. There are slight differences between the correction curve calculations for the luma and chroma channels.

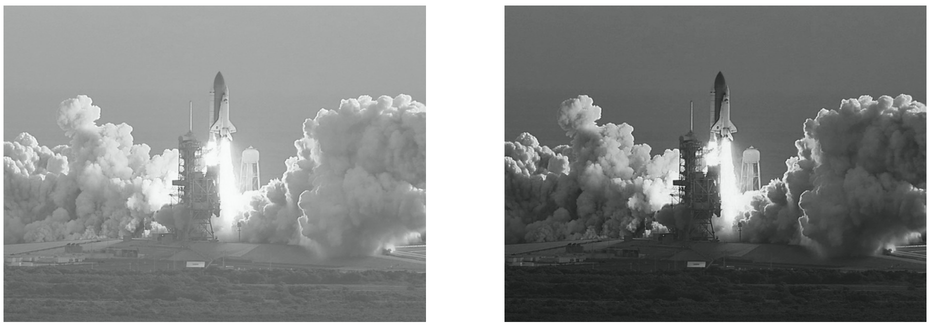


Figure 3.3: Luma of source (left) and processed frame (right)

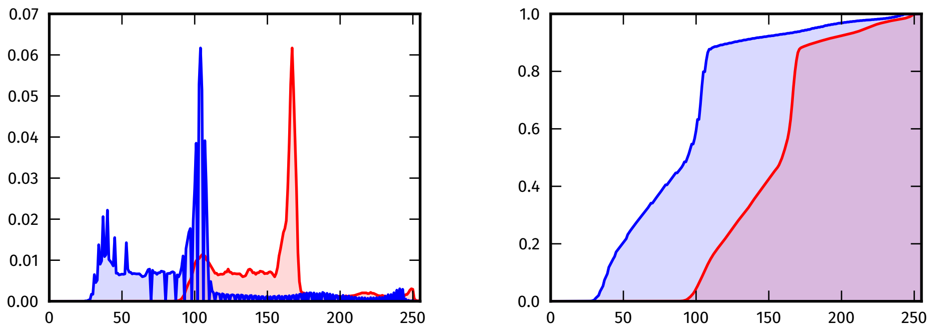


Figure 3.4: Normalised histograms and cumulative histograms

The histogram calculations is described in PEVQ as follows:

$$h_{s,\mu}[k] = \frac{1}{N \cdot W \cdot H} \sum_{t=0}^{N-1} \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} \delta[k, S_s[i,j,t]] \tag{3.12}$$

$$h_{p,\mu}[k] = \frac{1}{N \cdot W \cdot H} \sum_{t=0}^{N-1} \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} \delta[k, P_s[i,j,t]] \tag{3.13}$$

$$\delta[a,b] = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

The cumulation into cumulative histograms is defined as follows:

$$HC_{s,\mu}[\lambda] = \sum_{k=0}^{\lambda} h_{s,\mu}[k] \tag{3.15}$$

$$HC_{p,\mu}[\lambda] = \sum_{k=0}^{\lambda} h_{p,\mu}[k] \tag{3.16}$$

Correction curves

From the histograms created by (3.12-3.16), correction curves are calculated for each channel using histogram matching. The PEVQ description provides pseudo code for this process. These correction curves serve as mapping tables, according to which the pixel values of the spatially aligned PVS are corrected (3.18). SRC remains unchanged as shown in (3.17).

$$S_{A,\mu}[i,j,t] = S_{s,\mu}[i,j,t] \tag{3.17}$$

$$P_{A,\mu}[i,j,t] = CorrectionCurve_{\mu}[P_{s,\mu}[i,j,t]] \tag{3.18}$$

3.4 Distortion Analysis

At this point, we have a spatially aligned and colour corrected pair of SRC and PVS, and are ready to perform the actual analysis that leads to the assessment of the quality of PVS.

Index	*Name*	*Distortion type*	
1	Luma Indicator	Intra-frame	Spatial distortion
2	Chroma Indicator		
3	Introduced Component Indicator	Inter-frame	
4	Omitted Component Indicator		
5	Frame Repeat Indicator	Temporal distortion	

Table 3.1: PEVQ distortion indicators

During the distortion analysis step, PEVQ calculates 5 different indicators that independently contribute to the final score. The first two indicators measure intra-frame distortion for the luma and chroma channels respectively. Distortion is measured as introduction or loss of edges in a specific frame. Indicators three and four measure inter-frame distortion, i.e. distortion in the transition from one frame to the next. The fifth and last indicator takes anomalous frame skips/repeats into account, based on information from the temporal alignment routine.

3.4.1 Intra-frame spatial distortion

The Luma and Chroma Indicators follow the same overall procedure to measure distortion, with subtle differences in some of the steps. The general algorithm is as follows:

```
FOR EACH FRAME
    Step 1. Create edginess image for SRC and PVS
        1.1 $L^2$ norm of horizontal and vertical 1D filtering operations
        1.2 3x3 dilation filter
    Step 2. Calculate change in edginess for each frame
        2.1 Subtract SRC edginess from PVS edginess
        2.2 Weight by local colour and local edginess
        2.3 Clip to minimum/maximum impact range
    Step 3. Aggregate over space using $L^p$ norm weighted by pixel position
FOR ALL FRAMES
    Step 4. Average frame-wise distortions over time
```

Figure 3.5: Intra-frame spatial distortion analysis procedure

Step 2.2 is maybe the most interesting, explicitly displaying the intent to mimic the human perception. The theory is that loss or introduction of edginess in an area where there is already lots of edges, is less disturbing to the human eye than the loss or introduction of edginess in an area with little or noe edges. The authors of PEVQ also postulate that the local colour (luminance and chrominance) affects the importance of the visual qualities. We expand on this last part in the sections about the corresponding indicators.

Step 3 is also a perceptual one, giving more weight to changes appearing close the center of the frame than ones appearing further away.

Edginess Image

The first step is to create the edginess images needed by the luma and chroma indicator calculations. We perform the same operations as in the PEVQ description, but we have done some renaming to improve readability. This is done by applying the filtering operation $EdginessFilter$ to all channels $\mu \in \{Y, C_B, C_R\}$ in all frames t of the aligned sequences S_A and P_A.

$$
\begin{aligned}
S_{edge,\mu}[t] &= EdginessFilter\left(S_{A,\mu}[t]\right) \\
P_{edge,\mu}[t] &= EdginessFilter\left(P_{A,\mu}[t]\right)
\end{aligned}
\tag{3.19}
$$

The $EdginessFilter$ operation on an image I involves two steps. First, the image is convolved with two separate 1 dimensional filtering kernels, one vertical and one horizontal, and aggregate the two results using an L^2 norm. We then run a binary dilation operation on this result, using a 3×3 square as the structuring element. The dilation step is what gives rise to the notion of an *edginess image* as opposed to a regular edge image. In other words, if there is an edge at a specific position, the whole neighbouring area is defined to have edginess.

$$\begin{gathered}
EdginessFilter(I) = dilate_{3\times 3}\left(\sqrt{(I \ast K_v)^2 + (I \ast K_v)^2} \right) \\
K_v = \begin{bmatrix} 0.5 \\ 0.5 \\ 0 \\ -0.5 \\ -0.5 \end{bmatrix} \qquad K_h = \begin{bmatrix} 0.5 & 0.5 & 0 & -0.5 & -0.5 \end{bmatrix}
\end{gathered} \tag{3.20}$$

For an image I with 8 bit color depth, the edginess image E has the following properties:

$$\begin{gathered}
I_{x,y} \in [0, 256] \\
\Downarrow \\
I'_\delta = I \ast K_\delta, \quad \delta \in \{v, h\} \longrightarrow I'_{\delta,x,y} \in [-256, 256] \\
I'' = \sqrt{I_v'^2 + I_h'^2} \longrightarrow I''_{x,y} \in [0, \sqrt{2} \cdot 256] \\
\Downarrow \\
E = EdginessFilter(I) \longrightarrow E_{x,y} \in [0, \sqrt{2} \cdot 256] \approx [0, 362]
\end{gathered} \tag{3.21}$$

3.4.2 Luma Indicator

The luma indicator implements steps 2 and 3 from the procedure outlined in figure 3.5 for the luma channels of the sequences.

In PEVQ, the change in edginess from SRC to PVS e_Y is defined as follows:

$$e_Y[i,j,t] = \frac{P_{edge,Y}[i,j,t] - S_{edge,Y}[i,j,t]}{S_{edge,Y}[i,j,t] + 80 + dev[i,j,t]} \tag{3.22}$$

where

$$dev[i,j,t] = max(|S_{A,Y}[i,j,t] - 100|, |P_{A,Y}[i,j,t] - 100|) \tag{3.23}$$

The result is clipped to the impact range $[-40, 40]$

$$e_{clippedY}[i,j,t] = \begin{cases} -40 & \text{if } e_Y[i,j,t] \leq -40 \\ 40 & \text{if } e_Y[i,j,t] \geq 40 \\ e_Y[i,j,t] & \text{otherwise} \end{cases} \tag{3.24}$$

Dividing by $S_{edge,Y}$ means weighting by local edginess, and dividing by the so-called deviation signal dev corresponds to weighting by local colour, as described in section 3.4.1. The authors of PEVQ write that if introduction or loss of edginess appears in dark or bright areas, it is less disturbing than if it appears in an area with mid-range brightness, which is why we weight by the maximum absolute difference from 100 in local brightness. It is unclear why 100 is chosen as the middle point, being around 21% darker than the middle of the range at 127.

The weighting is made more obscure by the addition of the constant 80 in the denominator, but we can only assume that this is an expression of the relative importance of the other contributors to the weighting. I.e. the presence of a positive dev or $S_{edge,Y}$ value, will impact the absolute value of e_Y less if the constant 80 is already there.

A problem arises when arriving at the clipping performed in (3.24). If we examine the input values to e_Y, we find the following properties:

$$\begin{gathered} D = P_{edge,Y}[i,j,t] - S_{edge,Y}[i,j,t] \longrightarrow D \in \{-362, 362\} \\ W = S_{edge,Y}[i,j,t] + 80 + dev[i,j,t] \longrightarrow W \in \{80, 597\} \\ \Downarrow \\ e_Y[i,j,t] \in [-4.525, 4.525] \end{gathered} \tag{3.25a}$$

Given the shared component $S_{edge,Y}$ in the difference and the weighting factor, the actual range of e_Y is

$$e_Y[i,j,t] \in [-0.819, 4.525] \tag{3.25b}$$

Knowing this, it obviously doesn't make sense to clip e_Y to $[-40, 40]$ as described. Still, one has to assume that the clipping is there for a reason, namely to limit the impact of the change in edginess at a single point. Thus, to skip the clipping altogether is not a good solution. This leaves us with two options; 1. to upscale e_Y by some factor to activate the clipping for high values, and 2. to downscale the clipping range. While it may seem easier to change the clipping range, this has the effect that it changes the output range of the final aggregated indicator. This is unfortunate, because our score mapping function also depends on clipping to an impact range (see section 3.5). In both cases, the actual scaling factor cannot be much more than a best guess.

We mentioned the constant 80 present in the denominator of (3.22), controlling the relative importance of the weighting factors $S_{edge,Y}$ and dev. It of course also has the effect of yielding a smaller absolute value as the result. If we scale e_Y by the same constant 80, this effect is compensated for.

$$e_{Y'}[i,j,t] = 80\left(\frac{P_{edge,Y'}[i,j,t] - S_{edge,Y'}[i,j,t]}{S_{edge,Y'}[i,j,t] + 80 + dev[i,j,t]}\right) \tag{3.26}$$

In this case, we get an output range $[-65.52, 362]$, which will activate the clipping. The clipping may seem to be activated very easily, but it makes sense that an introduced edge with absolute difference of 40 in edginess in a mid-brightness area without any edges in SRC is about as seriously disturbing to a human viewer as any spatial distortion.

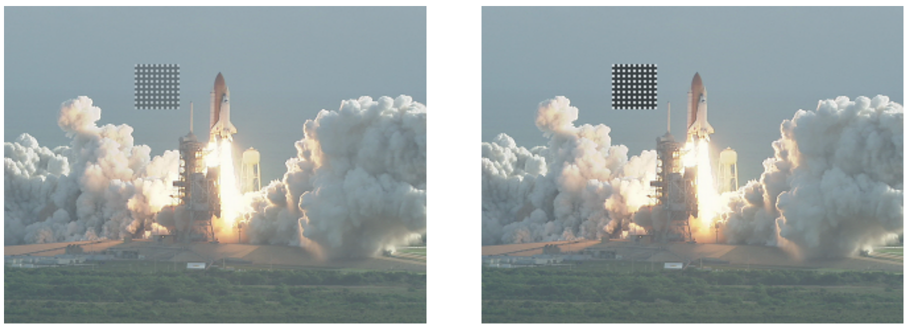


Figure 3.6: More pronounced error, but equal disturbance

The clipped results are aggregated using a normalised weighted L^5 norm as (3.27).

$$e_Y[t] = \sqrt[5]{\frac{\sum_{i=0}^{W-1}\sum_{j=0}^{H-1} \left|e_{clippedY}[i,j,t]\right|^5 w[i,j]}{\sum_{i=0}^{W-1}\sum_{j=0}^{H-1} w[i,j]}} \tag{3.27}$$

$$w[i,j] = \left|\sin\left(\pi\frac{i}{w}\right)\cdot\sin\left(\pi\frac{j}{h}\right)\right| \tag{3.28}$$

The factor $w[i,j]$ works to give more weight to errors close to the center of the frame than ones further away.

Finally these $e_Y[t]$ values are averaged over the sequence length N to produce the final indicator (3.29).

$$LumaIndicator = \frac{1}{N}\sum_{t=0}^{N-1} e_Y[t] \tag{3.29}$$

3.4.3 Chroma Indicator

The chroma indicator is the second intra-frame indicator, following the same general procedure as the luma indicator. The chroma indicator takes both chroma channels into account.

Again, we look at the change in edginess from SRC to PVS. These are the definitions from the PEVQ description:

$$e_{cb}[i,j,t] = \frac{P_{edge,cb}[i,j,t] - S_{edge,cr}[i,j,t]}{S_{edge,cb}[i,j,t] + 40 + 0.8 \cdot dev[i,j,t]} \tag{3.30}$$

$$e_{cr}[i,j,t] = \frac{P_{edge,cr}[i,j,t] - S_{edge,cr}[i,j,t]}{S_{edge,cr}[i,j,t] + 40 + 0.8 \cdot dev_{cb;cr}[i,j,t]} \tag{3.31}$$

where

$$dev_{cb;cr}[i,j,t] = max(Mx[i,j,t], My[i,j,t]) \tag{3.32}$$

$$Mx[i,j,t] = \sqrt{\left(S_{A,cb}[i,j,t] - 128\right)^2 + \left(S_{A,cr}[i,j,t] - 128\right)^2} \tag{3.33}$$

$$My[i,j,t] = \sqrt{\left(P_{A,cb}[i,j,t] - 128\right)^2 + \left(P_{A,cr}[i,j,t] - 128\right)^2} \tag{3.34}$$

There are a few obvious typographical errors in this description. The first can be found in (3.30), where we subtract the C_R channel of P_{edge} from the C_B channel of S_{edge} in the calculation of the change of edginess for the C_B channel. Also in (3.30), there is no channel index specified for the deviation signal dev in the denominator, which may introduce confusion with the deviation signal defined in section 3.4.2 for the Y' channel. By correcting these two typos, e_{cb} and e_{cr} are defined equally but using the edginess data from the respective channels only, see (3.39–3.41).

Just as with the luma indicator, the change in edginess is clipped to a minimum/maximum impact range.

$$e_{clippedCb}[i,j,t] = \begin{cases} -40 & \text{if } e_{Cb}[i,j,t] \leq -40 \\ 40 & \text{if } e_{Cb}[i,j,t] \geq 40 \\ e_{Cb}[i,j,t] & \text{otherwise} \end{cases} \tag{3.35}$$

$$e_{clippedCr}[i,j,t] = \begin{cases} -40 & \text{if } e_{Cr}[i,j,t] \leq -40 \\ 40 & \text{if } e_{Cr}[i,j,t] \geq 40 \\ e_{Cr}[i,j,t] & \text{otherwise} \end{cases} \tag{3.36}$$

The same discrepancy as with the luma indicator is present here, as the output range of e_μ is approximately $[-0.9, 9]$. We solve this the same way, by scaling up by the constant 40, see section 3.4.2 for details.

The change in edginess is aggregated using a weighted L^1 norm — or simply weighted average — as opposed to the L^5 norm used for the luma indicator. This reflects that luma degradations are considered more severe due to the fact that the HVS is much more sensitive to brightness than colour.

$$e_{Cb}[t] = \frac{\sum_{i=0}^{W-1} \sum_{j=0}^{H-1} e_{clippedCb}[i, j, t] w[i, j]}{\sum_{i=0}^{W-1} \sum_{j=0}^{H-1} w[i, j]} \tag{3.37}$$

$$e_{Cr}[t] = \frac{\sum_{i=0}^{W-1} \sum_{j=0}^{H-1} e_{clippedCr}[i, j, t] w[i, j]}{\sum_{i=0}^{W-1} \sum_{j=0}^{H-1} w[i, j]} \tag{3.38}$$

This definition of the aggregation, however, is also flawed. $e_{clippedCb}$ and $e_{clippedCr}$ range from −40 to 40, so aggregation of the non-absolute values is susceptible to cancellation. It cannot be the case that lots of introduced edges *and* loss of edges is better than just introduced edginess. Equally, it cannot be the case that introduced edges *and* loss of edges — as long as they are proportional in abundance — yields the same quality as no introduction nor loss of edges at all, i.e. no degradation. We therefore have to conclude that we must aggregate the absolute values of $e_{clippedCb}$ and $e_{clippedCb}$. This change is reflected in (3.43).

The last averaging steps over the two chroma channels and over the length of the sequence remain unchanged, see (3.44).

In addition to eliminating the flaws found in the PEVQ description, we wanted to generalise the operation to make it clear that we are performing the exact same calculations for both chroma channels. The definitions have been rewritten for OPVQ as follows:

$$\mu \in \{\mathrm{C_B}, \mathrm{C_B}\}$$

$$e_\mu[i,j,t] = 40\left(\frac{P_{edge,\mu}[i,j,t] - S_{edge,\mu}[i,j,t]}{S_{edge,\mu}[i,j,t] + 40 + 0.8 \cdot dev_{\mathrm{C_B};\mathrm{C_B}}[i,j,t]}\right) \tag{3.39}$$

$$dev_{\mathrm{C_B};\mathrm{C_B}}[i,j,t] = \max(M(S_A[i,j,t]), M(P_A[i,j,t])) \tag{3.40}$$

$$M(V) = \sqrt{\left(V_{\mathrm{C_B}} - 128\right)^2 - \left(V_{\mathrm{C_R}} - 128\right)^2} \tag{3.41}$$

$$e_{clipped,\mu}[i,j,t] = \begin{cases} -40 & \text{if } e_\mu[i,j,t] \le -40 \\ 40 & \text{if } e_\mu[i,j,t] \ge 40 \\ e_\mu[i,j,t] & \text{otherwise} \end{cases} \tag{3.42}$$

$$e_\mu[t] = \frac{\sum_{i=0}^{W-1}\sum_{j=0}^{H-1} \left|e_{clipped,\mu}[i,j,t]\right| w[i,j]}{\sum_{i=0}^{W-1}\sum_{j=0}^{H-1} w[i,j]}$$

$$w[i,j] = \left|\sin\left(\pi\frac{i}{W}\right)\cdot\sin\left(\pi\frac{j}{H}\right)\right| \tag{3.43}$$

$$ChromaIndicator = \frac{\sum_{t=0}^{N-1}\left(\frac{e_{C_B}[t] + e_{C_R}[t]}{2}\right)}{N} \tag{3.44}$$

3.4.4 Inter-frame spatial distortion

As opposed to the intra-frame luma and chroma indicators, the omitted and introduced component indicators are inter-frame indicators. A component is in this sense the amount of change at a specific position between two adjacent frames. The indicators are based on the difference of the components at the same spatial position in the corresponding pair of adjacent frames from SRC and PVS. The difference d is defined as the PVS component subtracted from the SRC component (3.45) in PEVQ.

$$d[i,j,t] = \left|S_{A,Y}[i,j,t] - S_{A,Y}[i,j,t-1]\right| - \left|P_{A,Y}[i,j,t] - P_{A,Y}[i,j,t-1]\right| \tag{3.45}$$

3.4.5 Introduced and Omitted Component Indicator

If a component is more pronounced in PVS than in SRC, we have an *introduced* component. Vice-versa, an *omitted* component is less pronounced in PVS than in SRC. Change in components are aggregated over space and time. Introduced components are treated as more severe degradations, to mimic human perception. This is reflected in the aggregation, as introduced components are aggregated over space by a normalised L^2 norm, and over time by a normalised L^5 norm. Omitted components are simply averaged (L^1 norm) over both space and time.

This is the definition of the Omitted Component Indicator from PEVQ:

$$d_{omitted}[i,j,t] = \begin{cases} 0 & \text{if } d[i,j,t] \leq 0 \\ -d[i,j,t] & \text{otherwise} \end{cases} \tag{3.46}$$

$$d_{omitted}[t] = \frac{1}{H \cdot W} \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} d_{omitted}[i,j,t] \tag{3.47}$$

$$OmittedComponentIndicator = \frac{1}{N} \sum_{t=0}^{N-1} d_{omitted}[t] \tag{3.48}$$

As in the description, we have an omitted component if the difference d is positive, because the value of the component was lower in PVS. It's unclear, however, why the positive value d should be negated in the other case (3.46). The aggregations in (3.47) and (3.48) are simple averages over space and time, which means that the resulting indicator will remain negative when the values in $d_{omitted}[i,j,t]$ are negative. The expected output range of the indicator is not defined in the indicator's section in the PEVQ description, but it is defined in the section about mapping the indicators to a final score as a positive range. Based on this, our conclusion is that the negation in the second case of (3.46) is an unintended typographical error. This has been corrected in the OPVQ definition, see (3.53).

The Introduced Component Indicator is defined in PEVQ as follows:

$$d_{introduced}[i,j,t] = \begin{cases} 0 & \text{if } d[i,j,t] \geq 0 \\ d[i,j,t] & \text{otherwise} \end{cases} \tag{3.49}$$

$$d_{introduced}[t] = \sqrt[5]{\frac{1}{W \cdot H} \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} d^5_{introduced}[i,j,t]} \tag{3.50}$$

$$IntroducedComponentIndocator = \sqrt{\frac{1}{N} \sum_{t=0}^{N-1} d^2_{introduced}[t]} \tag{3.51}$$

Here, we encounter a problem related to the error in the definition of Omitted Component Indicator. As $d_{introduced}[i, j, t]$ consists of strictly negative numbers, the contents of the 5th root in (3.50) will be negative as well. While negative numbers have real odd roots, it's not a concept well supported by the numerical math routines of computers. Because, the L^2 norm in (3.51) yields a positive result for whatever signedness resulting from (3.50), the suspicion is that the negation from (3.46) should really be placed in (3.49)

In addition to the flaws we have already pointed out, we find the negative definitions of the different component types in (3.46) and (3.49), i.e. *excluding* the irrelevant values instead of only *including* the relevant ones, to be unnecessarily implicit in nature.

These are the rewritten definitions for OPVQ:

$$d[i,j,t] = \left|S_{A,Y'}[i,j,t] - S_{A,Y'}[i,j,t-1]\right| - \left|P_{A,Y'}[i,j,t] - P_{A,Y'}[i,j,t-1]\right| \tag{3.52}$$

$$d_{omitted}[i,j,t] = \begin{cases} d[i,j,t] & \text{if } d[i,j,t] > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.53}$$

$$d_{omitted}[t] = \frac{\sum_{i=0}^{W-1} \sum_{j=0}^{H-1} d_{omitted}[i,j,t]}{W \cdot H} \tag{3.54}$$

$$OmittedComponentIndicator = \frac{\sum_{t=0}^{N-1} d_{omitted}[t]}{N} \tag{3.55}$$

$$d_{introduced}[i,j,t] = \begin{cases} \left|d[i,j,t]\right| & \text{if } d[i,j,t] < 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.56}$$

$$d_{introduced}[t] = \sqrt[5]{\frac{\sum_{i=0}^{W-1} \sum_{j=0}^{H-1} d_{introduced}^5[i,j,t]}{W \cdot H}} \tag{3.57}$$

$$IntroducedComponentIndicator = \sqrt{\frac{\sum_{t=0}^{N-1} d_{introduced}[t]}{N}} \tag{3.58}$$

3.5 Mapping to DMOS

The PEVQ description in J.247 uses the following formula for mapping the indicator values $I[i]$ to a final score:

$$Score = LinearOffset + \sum_{i=0}^{4} \frac{w[i]}{1+e^{\alpha \cdot I_{lim}[i]+\beta[i]}} \tag{3.59}$$

where

$$I_{lim} = \begin{cases} I_{min}[i] & \text{if } I[i] < I_{min}[i] \\ I_{max}[i] & \text{if } I[i] > I_{max}[i] \\ I[i] & \text{otherwise} \end{cases} \tag{3.60}$$

A matrix of coefficients I_{min}, I_{min}, w, α and β corresponding to each indicator, as well as a $LinearOffset$, is given for each supported resolution (fig. 3.7). I_{min} and I_{min} are called *input scaling factors*, which is inaccurate as they are not used to scale the indicator values I, but rather clip them. w, α and β are reported to be *output scaling factors*, which is accurate for w, but not for α and β. The summation body from equation 3.59 is a logistic function, whose curve has a sigmoid or S-like shape. α controls the acuteness of the curve, while β offsets the curve along the x-axis.

Mapping coefficients used for VGA resolution:

i	**Indicator ($I[i]$)**	$I_{min}[i]$	$I_{max}[i]$	$w[i]$	$\alpha[i]$	$\beta[i]$
0	LumIndicator	0.0000000	26.3458920	5.5178358	0.1982675	-1.9184154
1	ChromIndicator	0.0888870	11.9341383	-61.9967023	0.8956342	-14.5877780
2	OmittedComponent Indicator	0.0000000	1603.3526610	-12.8507869	0.0026048	2.3705606
3	IntroducedComponent Indicator	0.0000000	44.0389137	-0.2219432	0.7256163	-15.7681800
4	FrameRepeatIndicator	0.0000000	3.3093989	27700.0404630	2.4068676	11.2761009
$LinearOffset$		63.1413711				

Figure 3.7: Example of mapping coefficient matrix from PEVQ.

These degrees of freedom make it possible to very finely tune the impact of the individual indicators that produce the final score. Given an initial guess and subjective test results, a numerical optimisation algorithm can be employed to optimise the coefficients to minimise the error and maximise the correlation of the objective scores. However, the parameters can be combined in multiple ways to create the same curve. This represents a problem for machine learning algorithms that depend on finding minima or maxima. This can be remedied with additional

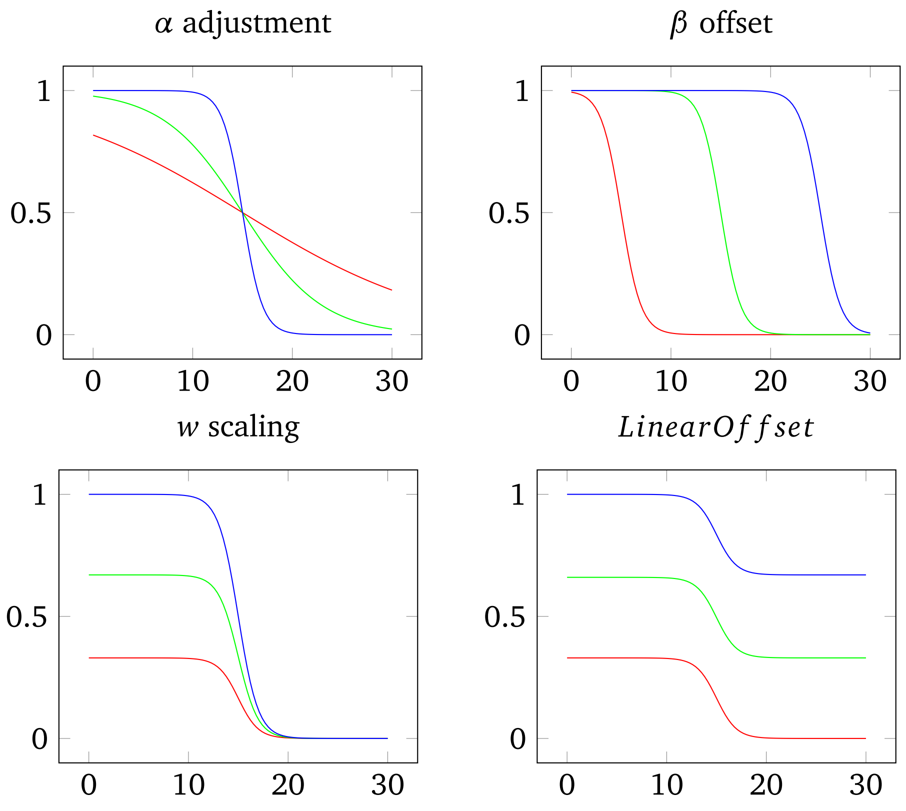


Figure 3.8: Demonstration of the curve parameters

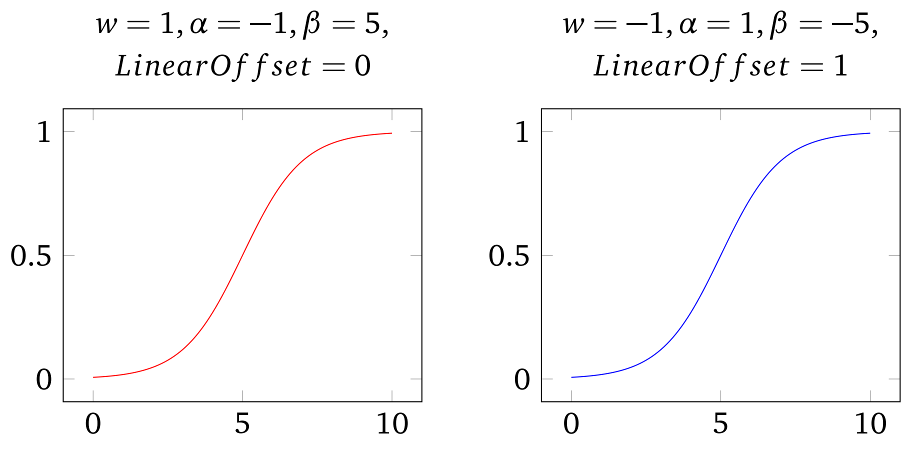


Figure 3.9: Equal curves with different parameters

constraints. If we look at the denominator of (3.59) separately we find the following:

$$\begin{aligned} d(I) &= 1 + e^{\alpha I + \beta} \\ d'(I) &= \alpha e^{\alpha I + \beta} \end{aligned} \quad \Rightarrow \quad \begin{cases} d \nearrow & \text{if } \alpha > 0 \\ d \searrow & \text{if } \alpha < 0 \end{cases} \tag{3.61}$$

This means that both w (the nominator) and α influence the direction of the curve. If we constrain $w \in \mathbb{R}_{>0}$, α alone controls the direction.

Further, we observe that for all of our indicators, the value increases with the number of degradations encountered. Thus, we expect our mapping curves to be decreasing, i.e. contributing less to the total score if the indicator value is high. As w is a strictly positive constant, this is achieved by having an increasing denominator. We can therefore apply the constraint $\alpha \in \mathbb{R}_{>0}$.

We define our final mapping function as follows:

$$Score = \epsilon + \sum_{i=0}^{3} \frac{\omega[i]}{1 + e^{\alpha[i] I_{lim}[i] + \beta[i]}} \qquad \begin{matrix} \alpha, \omega \in \mathbb{R}_{>0} \\ \beta, \epsilon \in \mathbb{R} \end{matrix} \tag{3.62}$$

$$DMOS_P = \begin{cases} 1 & \text{if } Score < 1 \\ 5 & \text{if } Score > 5 \\ Score & \text{otherwise} \end{cases} \tag{3.63}$$

3.5.1 Mapping coefficients

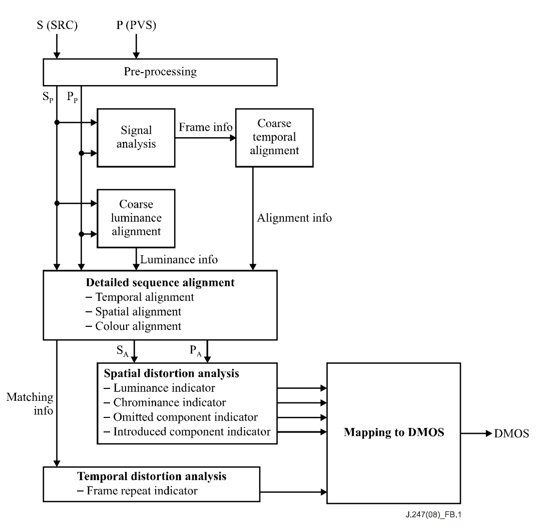
Because of the changes to the indicator calculations documented previously in this section, we expect that the mapping coefficients have to be recalculated to produce a meaningful score that correlates well with human subjective assessment. Furthermore, we notice that many of the coefficients supplied in the PEVQ description produce curves that weight their corresponding indicators oppositely from what they were intended to. For instance, the coefficients for the Chroma Indicator for VGA resolution makes the weighting sigmoid curve increasing, which in turn makes the Chroma Indicator contribute positively to the final score if the indicator value — i.e. number of errors — is high. While one can only make guesses as to why this is so, our assertion is that the most probable reason is that the authors have employed an unbounded numerical optimisation algorithm to optimise the coefficients to correlate well with a validation data set. Because the mapping function is additive in nature, such unbounded optimisation may well yield a result that correlates very well, but that doesn't follow the intention of the model designer, i.e. that more errors found contributes negatively to the final score, resulting in an overfitted mapping. This problem will be mitigated by constraining the model as described in section 3.5. Details about optimising the coefficients for OPVQ and evaluation of the improvements outlined in this section are presented in chapter 4.

3.6 Summary

In this chapter we have provided a full description of the OPVQ algorithm, and how its components are derived from PEVQ. Most of the components have been altered in some fashion compared to the original description, and some have been left out altogether. The resulting OPVQ algorithm is significantly less complex than PEVQ. All changes and corrections described in this chapter are summarised in table 3.2, and figure 3.10 shows schematic representation of both PEVQ and OPVQ.

PEVQ component	Detailed	**OPVQ status**
Preliminary steps	Pre-processing	*Identical*
	Signal analysis	*Removed*
	Coarse temporal alignment	*Removed*
	Coarse luminance alignment	*Removed*
Detailed sequence alignment	Fine temporal alignment	*Removed*
	Spatial alignment	*Altered*
	Colour alignment	*Identical*
Spatial distortion analysis	Luma indicator	*Altered*
	Chroma indicator	*Altered*
	Omitted component indicator	*Altered*
	Introduced component indicator	*Altered*
Temporal distortion analysis	Frame repeat indicator	*Removed*
Score mapping	Mapping to DMOS	*Altered*

Table 3.2: Overview of changes from PEVQ to OPVQ



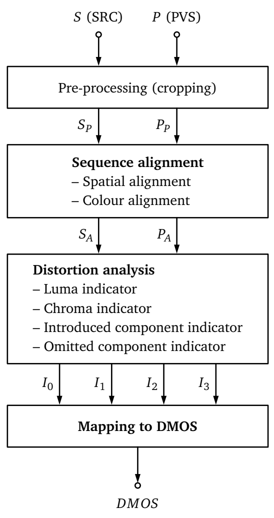


Figure 3.10: Original PEVQ model (left) and the derived OPVQ model (right)

Chapter 4

Results

In chapter 1 we described the goal of this project as implementing a video quality metric that represents a significant improvement over PSNR and releasing it as Free and Open Source Software for the general public as well as the research community to use and modify. In this chapter we evaluate the performance of the metric and conclude whether or not we have reached the goal of creating a metric that performs better than PSNR.

4.1 About performance evaluation

Evaluating the metric's performance is important if we're going to assert that it represents an improvement over PSNR. In addition to the immediate goal of outperforming PSNR, we expect OPVQ to at least equal the performance of PEVQ for non temporally distorted sequences. We follow VQEG's guidelines for measuring the performance of video quality metrics against subjective test data. VQEG's guidelines for performance evaluation are defined in their Final Report for the Multimedia Phase I validation effort (VQEG 2008), and a condensed version of the same guidelines are included as an appendix to the ITU-T Rec. J.247 (Appendix II) that we have already made multiple references to in the context of PEVQ. In this section we present details about these guidelines and the subjective test data we have been able to source.

4.1.1 Subjective vs. objective scores

When comparing subjective and objective data, it is important to take into account the inherent characteristics of the subjective data. For instance, subjective video quality assessment scores are often compressed at the ends of the rating scales, according to VQEG. Such characteristics are regarded as weaknesses that objective models are not required to mimic (VQEG 2008). Because of this, a non-linear mapping function that provides the best fit to the subjective data, should be applied to the predicted scores when validating objective metrics. Whenever we talk about predicted scores in this chapter, we differentiate between *fitted* and *unfitted* scores. In keeping with VQEG's notation in the Multimedia Phase I report, fitted scores are denoted $DMOS_P$, while the unfitted scores are denoted VQR, short for Video Quality Rating. The data fitting process is described in section 4.1.3.

4.1.2 Evaluation metrics

Three separate statistical measures are employed to assess the performance of objective models againt subjective test data, according to VQEG's Multimedia Phase I report:

1. Pearson correlation coefficient (Pearson's r)
2. Root-mean-square error ($RMSE$)
3. Outlier ratio (OR)

The first two, Pearson's R and $RMSE$, are well known statistical measures. The Outlier Ratio OR is not a common statistical measure, but is described in detail in the Multimedia Phase I report. The following paragraphs describe each of these evaluation metrics in detail.

Pearson correlation coefficient

The Pearson correlation coefficient (Pearson's r) is a measure of linear correlation between two variables. It is defined as the covariance of the two variables divided by the product of their standard deviations. This yields values in the range $(-1, 1)$, 1 being total positive correlation, 0 no correlation and 1 total negative correlation. In statistical terms we're dealing with samples in our work, so the following definition is employed:

$$r = \frac{\sum_{n=1}^{N}(x_n - \bar{x})(y_n - \bar{y})}{\sqrt{\sum_{n=1}^{N}(x_n - \bar{x})^2}\sqrt{\sum_{n=1}^{N}(y_n - \bar{y})^2}} \tag{4.1}$$

where

$$x = \{x_1, \ldots, x_N\}$$
$$y = \{y_1, \ldots, y_N\}$$

The variables x and y represent the observed and predicted values. In our context the observed values are subjective $DMOS$ scores, while the predicted values are the scores predicted by OPVQ or other objective models, either as unfitted VQR scores or fitted $DMOS_P$ scores.

Root-mean-square error

$RMSE$ is a commonplace statistic that measures the difference between predicted/estimated values and actually observed values, and is regarded as a good measure of the accuracy of the predictive model in question. As the name suggests, $RMSE$ is the square root of the mean of the squared differences between the predicted and observed values. In other words, it is the standard deviation of the differences between the values. We define $RMSE$ mathematically as

$$RMSE = \sqrt{\frac{\sum\limits_{n=1}^{N} (x_n - y_n)^2}{N - d}} \tag{4.2}$$

where x represents the observed values and y represents the predicted values. The degrees of freedom of the fitting model d is taken into account to remove bias (VQEG 2008). E.g., if no data fitting is applied, $d = 0$. In the case of cubic polynomial data fitting as applied by VQEG, $d = 4$, see details in section 4.1.3.

Outlier ratio

In the Multimedia Phase I Final Report, VQEG defines a simple evaluation metric *Outlier Ratio* which is simply the number of outlier measurements divided by total number of measurements.

$$OR = \frac{NumOutliers}{N} \tag{4.3}$$

In statistics, there is no hard definition as to what constitutes an outlier, so a definition is provided in the report which states that a measurement is an outlier if the following is satisfied:

$$|x_n - y_n| > K2 \cdot \frac{\sigma_{x_n}}{\sqrt{Nsubjs}} \tag{4.4}$$

The standard deviation σ_{x_n} is the standard deviation of the subjective scores for test case n. $K2$ is the absolute value of the 95% confidence interval of the distribution in question. The appropriate distribution is the Student's T distribution with degrees of freedom $df = Nsubjs - 1$. It is stated in VQEG's report (VQEG 2008, p. 44), however, that it can be substituted for a Gaussian distribution if $Nsubjs \geq 30$, for which $K2 = 1.96$. Why the limit is set to 30 is not clear, as Student's T's $K2$ does not approach 1.96 before $df = 100$ approximately.

4.1.3 Data fitting

As mentioned in section 4.1.1, data fitting is applied to VQR scores before evaluation with Pearson's r, $RMSE$ and OR. The function used is the cubic polynomial

$$DMOS_P = ax^3 + bx^2 + cx + d \tag{4.5}$$

where the result $DMOS_P$ is the predicted score to be compared with the subjective $DMOS$, and the variable x corresponds to a VQR. The coefficients $\{a, b, c, d\}$ are then calculated so that the resulting set of $DMOS_P$ values fits the set of subjective $DMOS$ scores as well as possible.

Data fitting is performed in two steps. First, correlation is maximised by determining $\{a', b', c'\}$ for the function

$$DMOS_p = k(a'x^3 + b'x^2 + c'x) + d \tag{4.6}$$

with $k = 1$ and $d = 0$ kept constant. In this step the maximisation is constrained by the requirement that the function is monotonic within the range of possible values, defined by the scale of the subjective scores.

Next, the RMSE is minimised by determining $\{k, d\}$ for (4.6). With $\{k, d\}$ determined, we expand (4.6) so that

$$a = ka' \qquad b = kb' \qquad c = kc' \tag{4.7}$$

and obtain the final coefficients $\{a, b, c, d\}$ for (4.5).

4.1.4 Comparative evaluation of OPVQ and PEVQ

As OPVQ is based on PEVQ, it would be interesting to compare the performance of OPVQ to the performance of PEVQ as reported in J.247. The standard presents some numbers describing the performance achieved by the model during the validation testing, but the datasets are not available to the general public, as discussed in section 2.6. This means that the performance of OPVQ presented in this thesis may not be comparable with the performance of PEVQ from J.247, because the subjective baseline for the evaluation is different.

4.1.5 Datasets

To evaluate the performance of our objective metric, we need access to subjective scores. Not many sources of subjective data are freely available to the research community and the general public. We also face the further limitation that OPVQ only supports a small set of resolutions, namely VGA, CIF and QCIF. Due to time/resource limitations, we decided to validate OPVQ's performance only for VGA resolution data, and not the lower CIF and QCIF resolutions. We have located the following datasets that provide subjective scores for VGA resolution sequences:

Institution	*Dataset*	*Types of distortion*	*# sequences ref/dist*
IRCCyN	Influence Content[1]	H.264/SVC coding	60/240
IRCCyN	H264 AVC vs. SVC[2]	H.264 and H.264/SVC coding	4/48
IRCCyN	SVC4QoE Replace Slice[3,4]	H.264 and H.264/SVC coding, transmission errors	9/131
IRCCyN	SVC4QoE QP0 QP1[3]	H.264 and H.264/SVC coding	11/313
IRCCyN	SVC4QoE Temporal Switch[5,3]	H.264 and H.264/SVC coding, switching between SVC base layer and SVC enhanced layer	11/379
ETFOS	VGA Video Quality (EVVQ) database[6,7]	H.264/AVC compression and MPEG-4 Visual compression	8/90

Table 4.1: Subjective test data for VGA resolution sequences

IRCCyN/IVC Datasets

IRCCyN is the Institute for Research into Communications and Cybernetics at the École Centrale de Nantes, France. The institute's Images and Video-communications team (*IVC*) make all of their subjective test data available to the general public on their website[8], which has proven to be

[1]Pitrey, Barkowsky, Pépion, et al. (2012)
[2]Pitrey et al. (2010b)
[3]Pitrey, Engelke, Barkowsky, et al. (2011)
[4]Pitrey et al. (2010a)
[5]Pitrey, Engelke, Le Callet, et al. (2011)
[6]Rimac-Drlje et al. (2010)
[7]Vranješ et al. (2013)
[8]http://ivc.univ-nantes.fr/en/databases/filter/Quality%2520Assessment/

a rich source of test data. At present, the IVC team has made five separate datasets for VGA video available, two of which contain two separate data series.

Unfortunately, some of the datasets contain quality degradations that are outside the scope of the distortion detection mechanisms of OPVQ. The *SVC4QoE Replace Slice* test considers errors that are limited to a few slices[9] of the H.264 frames. Because of the spatial averaging present in the OPVQ distortion indicators[10], the impact of highly local errors is diluted, which does not correspond to human perception. The *SVC4QoE Temporal Switch* test features a mid-sequence quality switch, which similarly represents an error local to only part of the sequence, however in the temporal domain instead of the spatial domain. OPVQ also does temporal averaging of errors, which makes it incapable of correcly assessing temporal switching of quality in the PVS. We did not include data from these two tests in our performance evaluation.

Out of the remaining datasets, all tests were conducted according to the ACR-HR testing methodology (VQEG 2008), except the *H.264 AVC vs. SVC* test, which was conducted according to the SAMVIQ testing methodology (Péchard et al. 2008).

ETFOS Dataset

ETFOS is the Faculty of Electrical Engineering of the University of Osijek, Croatia. Their Video Quality Group (VQG) has made available one dataset of subjective test data for VGA video sequences. The subjective test was conducted according to the SAMVIQ testing methodology. The source and processed video sequences were made available as raw planar $Y'C_BC_R$ 4:2:0 video files, at 25 fps. For convenience, we wrapped these video streams in AVI containers[11], using the Libav tool `avconv`[12]:

```
user@host: ~ $ find . -name "*.yuv" | xargs -I{} basename "{}" .yuv
     | xargs -I{} avconv -y -r 25 -f rawvideo -s 640x480 -pix_fmt
    yuv420p -i "{}".yuv -vcodec rawvideo -pix_fmt yuv420p -f avi -r
     25 -s 640x480 "{}".avi
```

[9]*Slice*: row of macroblocks

[10]see Step 3 of figure 3.5, p.31

[11]Microsoft Developer Network — AVI RIFF File Reference https://msdn.microsoft.com/en-us/library/ms779636.aspx (*visited 7/4/2015*)

[12]Libav https://libav.org/about.html (*visited 20/1/2015*)

4.2 Score mapper evaluation

A significant part of the OPVQ algorithm deals with the mapping of the four separate distortion indicators that are produced, into a single VQR. As described in section 4.1.1, VQR scores should not necessarily mimic the non-linear characteristics of subjective scores, without being subject to data fitting after the fact. Nontheless, OPVQ's score mapper is intended to generate VQR scores that closely resemble those reported by human test subjects, and inherently employs non-linear weighting of the degradation indicators to achieve this. This means that if the OPVQ score mapper performs well on its own, additional polynomial fitting will not be necessary. Thus, a good measure of the performance of the OPVQ score mapper and its coefficients will be the linearity of the best-fit polynomial $DMOS_P$. The linearity can be measured by calculating the Pearson correlation coefficient for the fitted and unfitted data.

In this section we look at the changes made to the score mapper in isolation from the rest of the algorithm, and evaluate their impact on the result.

4.2.1 Constraining the score mapper

The PEVQ description comes with a set of mapping coefficients to be used in the score mapping function ($\{\omega, \alpha, \beta, \epsilon\}$ from eq. 3.62, p. 42). In order to adapt these mapping coefficients to perform well with our altered degradation indicators, we want to employ convergence based numerical optimisation algorithms. A problem is that the PEVQ mapping function makes it possible to map a set of indicators to the same score with wildly different mapping coefficients. In section 3.5 we proposed to constrain the mapping function to amend this issue. With such constraints applied, we expect different optimisation algorithms to produce similar and predictable results. Another aspect is that constraining the mapper prevents overfitting the model to the training data supplied, thereby potentially improving the model's performance directly.

To test this, we ran numerical optimisation with and without the proposed constraints. Some of the original coefficients from PEVQ fall outside the constrained bounds, which meant that we could not use the original coefficients directly as the initial guess for the constrained optimisation. By only flipping the signs of the offending coefficients, we kept the scale and shape of the individual mapping curves intact, but changed the direction to adhere to the constraints and follow the intended mapping model.

i	$Indicator[i]$	$\omega[i]$	$\alpha[i]$	$\beta[i]$
0	LumIndicator	5.5178358	0.1982675	-1.9184154
1	ChromIndicator	-61.9967023	0.8956342	-14.5877780
2	OmittedComponent Indicator	-0.2219432	0.7256163	-15.7681800
3	IntroducedComponent Indicator	-12.8507869	0.0026048	2.3705606
ϵ		63.1413711		

Table 4.2: Inital guess for unconstrained optimisation (original VGA coefficients)

i	$Indicator[i]$	$\omega[i]$	$\alpha[i]$	$\beta[i]$
0	LumIndicator	5.5178358	0.1982675	-1.9184154
1	ChromIndicator	**61.9967023**	0.8956342	-14.5877780
2	OmittedComponent Indicator	**0.2219432**	0.7256163	-15.7681800
3	IntroducedComponent Indicator	**12.8507869**	0.0026048	2.3705606
ϵ		**-63.1413711**		

Table 4.3: Inital guess for constrained optimisation (changed values in bold)

The Python library `scipy.optimize` provides several numerical minimisation algorithms for multivariate functions.[13] We ran all the applicable algorithms[14] with the *IRCCyN Influence Content* dataset as input data, and used the resulting mapping coefficients on the smaller *IRCCyN H.264 AVC vs. SVC* dataset, with and without constraints.

	Average standard deviation			
	ω	α	β	ϵ
Unconstrained	4.672	1392.633	6951.061	9.367
Constrained	1.652	5.548	35.323	1.703

Table 4.4: Variability of optimised coefficients

It's obvious from table 4.4 that the constraints help us get less variable coefficients. From the results in table 4.5 we make the observation that even though the unconstrained optimisation produces slightly

[13]SciPy Reference Guide — Optimization (scipy.optimize) http://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html (*visited 5/2/2015*)

[14]Algorithms used: Nelder-Mead, Powell, CG, BFGS, L-BFGS-B, TNC to minimise RMSE. The first three do not support constraints, so the constraints were implicitly enforced by the provided optimizable function. The latter two algorithms take explicit bounds for each variable as additional input.

	Correlation		**RMS error**		**Outlier ratio**	
	avg.	*min*	*avg.*	*max*	*avg.*	*max*
Unconstrained						
Influence content	0.909	0.908	0.452	0.455	0.524	0.542
H.264 AVC vs. SVC	0.957	0.955	0.359	0.394	0.326	0.417
Constrained						
Influence content	0.906	0.898	0.460	0.477	0.526	0.571
H.264 AVC vs. SVC	0.961	0.957	0.342	0.413	0.243	0.417

Table 4.5: Performance of DMOS mapping after optimisation

better mapping during training, the coefficients from the constrained optimisation performs better on the test data. We interpret this as the constraints guarding against overfitting as expected.

4.2.2 Optimising mapping coefficients

OPVQ produces four different degradation indicators that are mapped into a single VQR. The calculation of these indicators, as well as the mapping function itself, has been changed from the description in PEVQ. We suspected that the coefficients needed for mapping the OPVQ indicators to a VQR had to be re-calculated to reflect this change. To investigate this, we started by evaluating the performance of OPVQ using the original coefficients from PEVQ.

Table 4.6 shows better correlation for OPVQ on the IRCCyN and ETFOS datasets than PEVQ on VQEG's unavailable datasets from the Multimedia Phase I tests. On the other hand, the error measures are worse. This indicated to us that the coefficients indeed needed to be re-calculated.

To test if we could tune the coefficients with a numerical method using training data from one or more subjective datasets to predict scores from a different dataset, we ran a simple cross-validation scheme using the different available datasets as a partition of a larger superset. For every dataset, we optimised the coefficients using all the other available datasets as training data and the predicted scores for the dataset that was left out from training. To optimise, we used the L-BFGS-B bounded multivariate minimisation algorithm from the Python library `scipy.optimize` to minimise average RMSE for the training data.

The results show that correlation has been improved. There is also

	Dataset					Averages	
	AVC/SVC	*Infl.cont.*	*QP0 QP1 (A)*	*QP0 QP1 (B)*	*EVVQ*	OPVQ	PEVQ[15]
Pearson R	0.939	0.898	0.856	0.856	0.888	**0.887**	0.825
RMS error	0.338	0.722	0.695	0.695	0.583	0.606	**0.571**
Outlier ratio	0.375	0.829	0.790	0.790	0.556	0.668	**0.502**

Table 4.6: OPVQ performance with unaltered PEVQ mapping coefficients

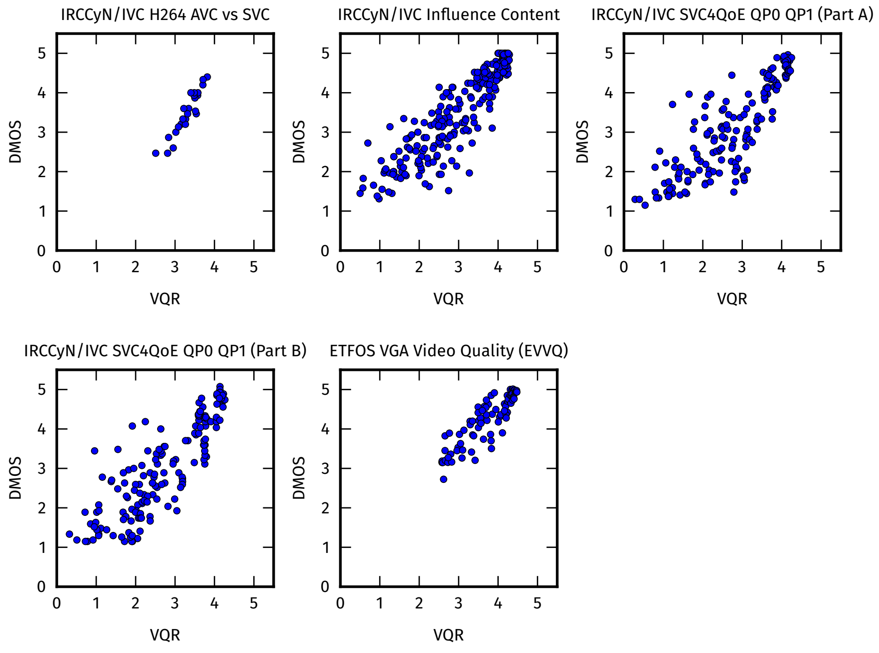


Figure 4.1: VQRs from OPVQ with unaltered PEVQ mapping coefficients

siginficant improvement in precision, accounted for by less error and fewer outliers. This shows us that training the mapping coefficients using subjective data is a sound method.

We wanted to investigate if the testing methodology used in the subjective test impacted the degree to which a dataset is suitable as training data. Of the five datasets used, two — AVC/SVC and EVVQ — were conducted using the SAMVIQ test methodology, while ACR-HR was used for the remaining three — Influence Content and QP0 QP1 Parts A and B. To determine if this impacted the trained coefficients, we re-ran the cross-validation scheme, but this time only training with datasets produced using the same methodology. This meant that the coefficients used for testing AVC/SVC were trained using only EVVQ etc.

[15]Averages for PEVQ are taken from J.247, and are *not* computed using the same test data.

	Dataset					Averages	
	AVC/SVC	*Infl.cont.*	*QP0 QP1 (A)*	*QP0 QP1 (B)*	*EVVQ*	OPVQ	PEVQ
Pearson R	0.942	0.907	0.861	0.870	0.880	**0.892**	0.825
RMS error	0.291	0.477	0.589	0.598	0.409	**0.473**	0.571
Outlier ratio	0.167	0.592	0.577	0.677	0.222	**0.447**	0.502

Table 4.7: OPVQ performance with optimised mapping coefficients using the other datasets as training data

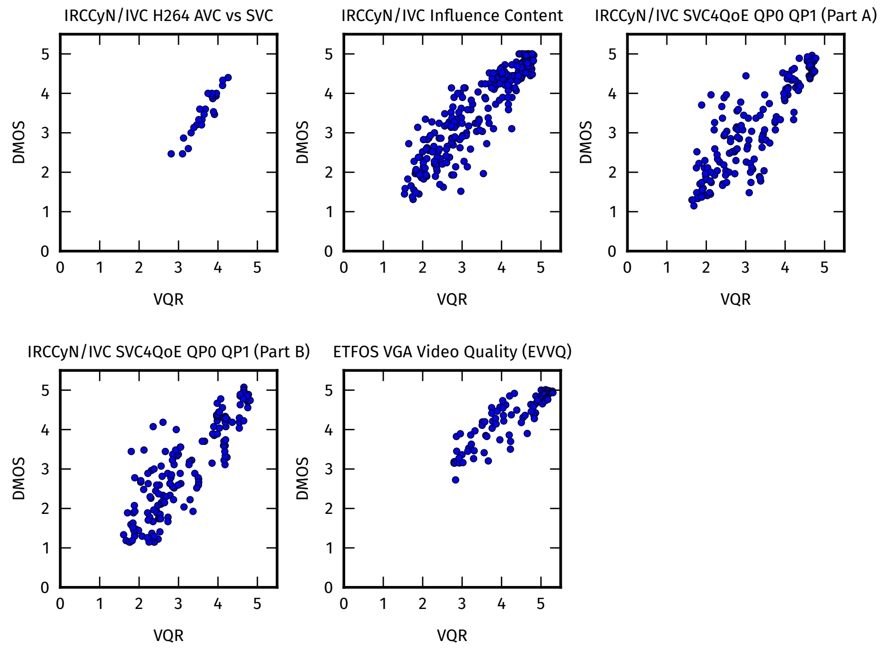


Figure 4.2: VQR from OPVQ with mapping coefficients optimised using the other datasets as training data

The results do not show improvement compared to training with all the other available subjective data. The lack of sufficient training data does seem to play a significant role, because the two SAMVIQ datasets (AVC/SVC and EVVQ) that are left with only each other as training data suffer the greatest decline. Incidentally, these are also the two datasets with the lowest number of samples, 48 and 90 respectively. This makes it difficult to draw any conclusions from this result, and further experimentation with more subjective data available would be preferable.

	Dataset					Averages	
	AVC/SVC	*Infl.cont.*	*QP0 QP1 (A)*	*QP0 QP1 (B)*	*EVVQ*	Trained with	
	SAMVIQ	ACR-HR	ACR-HR	ACR-HR	SAMVIQ	Same method	All data
Pearson R	0.937	0.905	0.864	0.870	0.880	0.891	**0.892**
RMS error	0.498	0.491	0.581	0.595	0.675	0.568	**0.473**
Outlier ratio	0.542	0.537	0.595	0.695	0.744	0.623	**0.447**

Table 4.8: OPVQ performance with optimised mapping coefficients using only the other datasets based on the same test methodology as training data

4.2.3 Evaluation

In section 4.1.1 we described issues with mapping an objective video quality rating to a subjective scale. In this section we evaluate the OPVQ score mapper's ability to produce subjective scale results.

We performed data fitting as described in section 4.1.1 for VQR scores produced with the original coefficients from PEVQ and optimised coefficients using the same scheme as in section 4.2.2. Figures 4.3 and 4.4 show the subjective $DMOS$ scores plotted against objective VQR along with the best-fit cubic polynomials.

Table 4.9 shows that training the coefficients beforehand results in less fitting necessary after the fact, and that OPVQ's built in DMOS Mapper is capable of mimicking nonlinear characteristics of subjective data.

	Linearity	
	Unaltered	Optimised
AVC/SVC	0.995	**0.999**
Infl.cont.	0.990	**1.000**
QP0 QP1 (A)	0.973	**0.991**
QP0 QP1 (B)	0.980	**0.998**
EVVQ	**0.998**	0.989
Average	0.987	**0.996**

Table 4.9: OPVQ DMOS Mapper performance measured by the linearity of the fitted and unfitted data

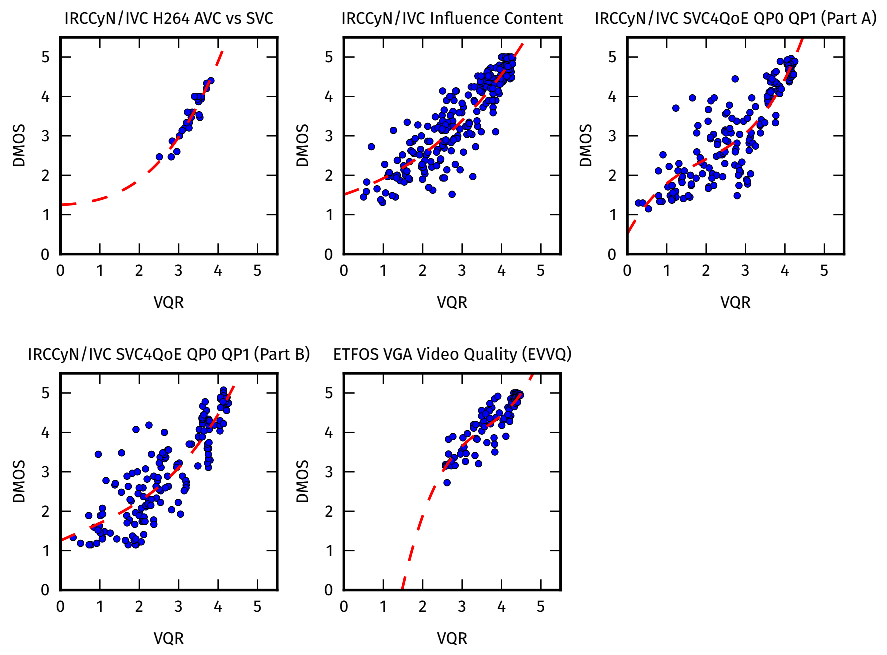


Figure 4.3: Best-fit cubic polynomials for OPVQ using unaltered PEVQ mapping coefficients

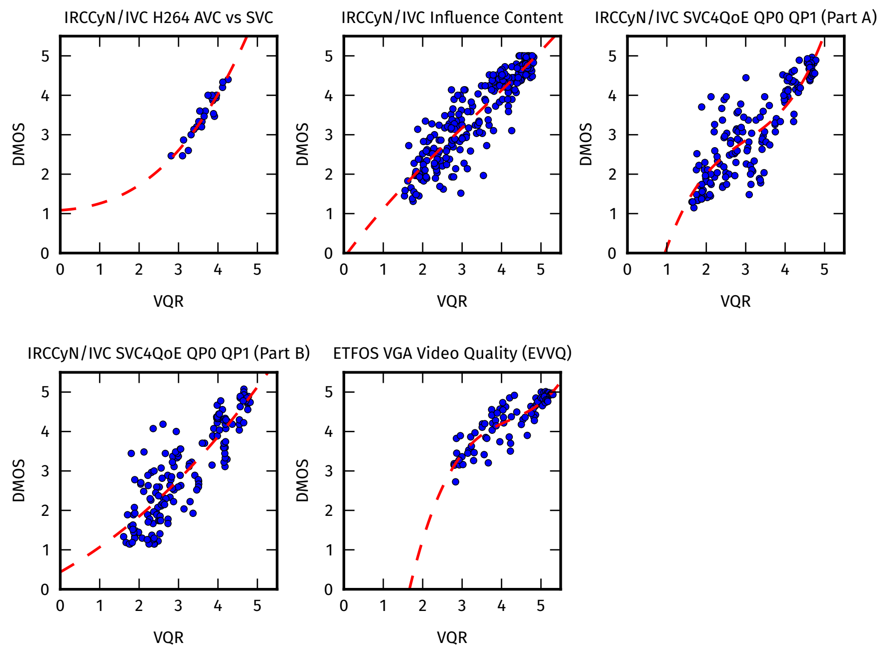


Figure 4.4: Best-fit cubic polynomials for OPVQ rating using optimised mapping coefficients

4.3 Overall performance evaluation

Part of the primary goal of this project was to implement a video quality metric that performs significantly better than PSNR. Subsequently, we expect our implementation to be on par with OPTICOM's PEVQ, the metric that it's derived from. As mentioned in section 4.1.4 however, we do not have the resources to make a direct comparison with PEVQ.

To investigate if OPVQ performs significantly better than PSNR, we calculated PSNR ratings for the applicable datasets, and used cubic polynomial fitting to best fit the scores to the subjective scale of each dataset. We used our own implementation of PSNR with spatial alignment enabled (see section 7.2), to ensure optimal conditions for PSNR. We also calculated the final performance evaluation data for OPVQ with cubic polynomial fitting, using unaltered PEVQ coefficients as well as optimised coefficients using both schemes described in section 4.2.2.

The results in table 4.10 show a clear difference in performance between PSNR and OPVQ in all configurations, and strongly indicate that we have achieved our goal of implementing a video quality metric that significantly outperforms PSNR.

An interesting observation is that the best performance for OPVQ after polynomial fitting is achieved when using the unaltered PEVQ coefficients — if only slightly better than with trained coefficients. As we discussed in section 3.5.1 the coefficients from the PEVQ description are not logically sound for some indicators, as they reward increase in amount of errors with higher contribution to the final score. Because of this, and the fact that the difference recorded is only minimal, our recommendation is to use optimised coefficients adhering to the constraints defined for OPVQ in section 3.5.

As we do not own a PEVQ license and the subjective data used for the validation of PEVQ in ITU-T Rec. J.247 is unavailable, we do not have the sufficient means to investigate the subsequent goal of keeping OPVQ's performance on par with PEVQ. The performance data we have been able to produce shows better performance for OPVQ than what's reported for PEVQ in J.247, but as these figures are not produced using the same test data they are not reliable for comparison.

	Dataset					
	AVC/SVC	*Infl.cont.*	*QP0 QP1 (A)*	*QP0 QP1 (B)*	*EVVQ*	Average
OPVQ w/coeffs unaltered from PEVQ						
Pearson R	**0.943**	**0.907**	**0.872**	0.873	**0.890**	**0.897**
RMS error	**0.201**	**0.454**	**0.567**	0.578	**0.283**	**0.416**
Outlier ratio	**0.000**	**0.521**	0.643	0.665	**0.078**	**0.381**
OPVQ w/coeffs optimised using all other datasets						
Pearson R	**0.943**	**0.907**	0.869	0.871	**0.890**	0.896
RMS error	**0.201**	**0.454**	0.574	0.583	**0.283**	0.419
Outlier ratio	**0.000**	0.529	0.649	0.677	**0.078**	0.386
OPVQ w/coeffs optimised using datasets of same test methodology						
Pearson R	**0.943**	**0.907**	0.867	0.871	0.886	0.895
RMS error	**0.201**	**0.454**	0.576	0.583	0.288	0.420
Outlier ratio	**0.000**	0.525	0.655	0.671	0.111	0.392
PSNR w/spatial alignment						
Pearson R	0.515	0.813	0.807	0.807	0.715	0.731
RMS error	0.516	0.629	0.684	0.700	0.434	0.593
Outlier ratio	0.542	0.642	0.631	0.701	0.200	0.543
SSIM w/spatial alignment						
Pearson R	0.584	0.859	0.864	**0.878**	0.859	0.809
RMS error	0.489	0.552	0.583	**0.567**	0.317	0.502
Outlier ratio	0.458	0.604	**0.589**	**0.623**	0.100	0.475
PEVQ as reported in J.247 – not tested with the same data						
Pearson R	-	-	-	-	-	0.825
RMS error	-	-	-	-	-	0.571
Outlier ratio	-	-	-	-	-	0.502

Table 4.10: Final performance data after cubic fitting (best results in bold)

4.4 Summary

In this chapter we have described our evaluation of the OPVQ full reference video quality metric. We have seen that for the subjective data available to us and within an expected range of quality degradations, OPVQ performs significantly better than PSNR. Because the amount of subjective test data available is small, we encourage further validation of the metrics performance using other datasets. We have not been able to conclusively compare the performance of OPVQ with PEVQ.

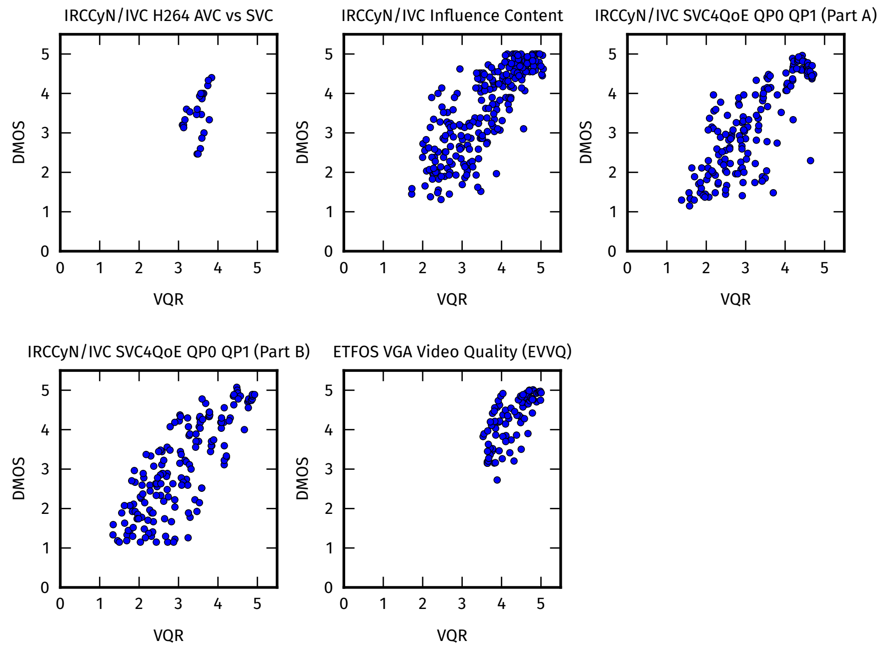


Figure 4.5: Cubic fitted PSNR scores plotted against subjective DMOS scores

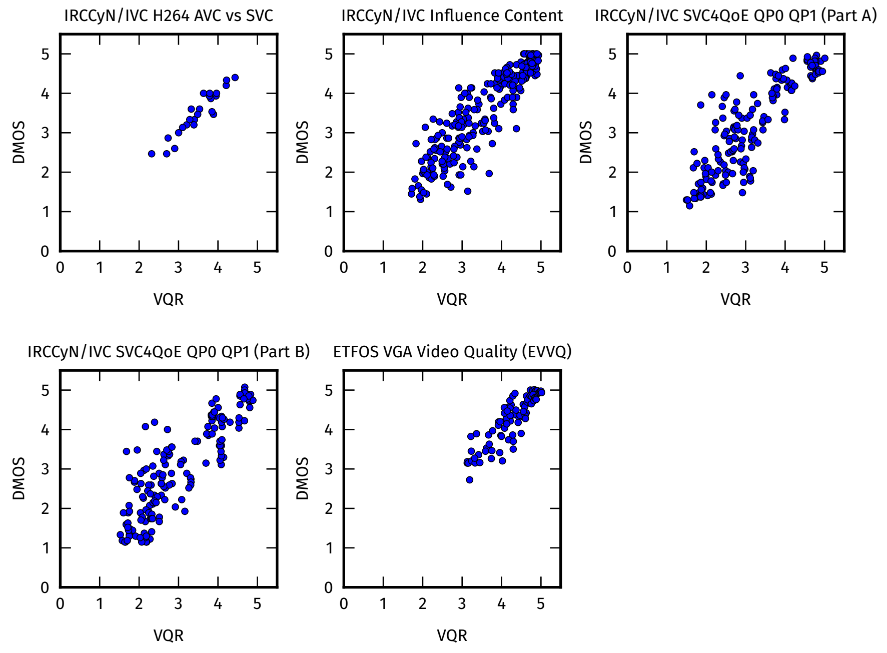


Figure 4.6: Cubic fitted OPVQ scores plotted against subjective DMOS scores. OPVQ mapping coefficients optimised with the other available datasets

Part III

OpenVQ – The Open Video Quality Toolkit

Chapter 5

Designing a Video Quality Toolkit

When we first started this project, the goal was to pick a standardised video quality metric and create a stand-alone implementation. After deciding on PEVQ as the base metric, we started implementing a program according to the standard. The PEVQ description in ITU-T J.247 consists mainly of mathematical descriptions of the individual steps that make up the complete metric, which require many layers of abstraction to be implemented in a computer program before they are directly applicable. To provide these abstractions, we had to familiarise ourselves with several problems regarding programmatic handling of video data. Examples of such problems are decoding compressed video residing on disk into raw frames in memory, encoding of color spaces, chroma subsampling and performing mathematical operations on frames/pixels.

Implementing all of the steps necessary before getting to perform even the first operations defined in the OPVQ description made us realise that these problems had nothing to do with the specific metric we were trying to implement. Different video quality metrics share a common modus operandi. They have a set of digital signal processing (DSP) steps that they perform on each frame, or a few consecutive frames, from the first to the last frame. In other words, if we were setting out to implement even a primitive metric such as PSNR, the steps of, for instance, decoding video and defining a representation of the frames in memory would still need to be tackled first. Furthermore, even integral parts of OPVQ itself, such as the alignment steps, could be implemented as modules to be plugged into other metric implementations in a dynamic fashion. This gave us the incentive to generalise parts of the implementation to develop a toolkit that provides the abstractions necessary to implement *any* video qualty metric — not just OPVQ — instead of a specialised OPVQ program. We call this the Open Video Quality toolkit, or OpenVQ for short. In this chapter we describe considerations that went into the design of OpenVQ.

5.1 Background

The first revision of our implementation was a stand alone implementation of PEVQ. Development of this first revision progressed from January through September of 2014, at which point we had a complete implementation of the PEVQ model as described in J.247. This revision did not yield good results, which made us analyse the PEVQ description in more detail. This analysis resulted in the closely related metric OPVQ as detailed in chapter 3. Subsequently, development continued until January 2015 when our implementation was working to produce results as reported in chapter 4.

In programming terms, the design idea revolved around creating separate classes for each stage in the metric — now OPVQ — with a manager function that called these separate classes into action accordingly for each stage. The results would be stored in a separate container class which the manager controlled and passed on to any later stage requiring access to the results. Even though we had an object oriented implementation with separate modules performing different tasks, all parts of our code including video decoding, scaling, video frame structures and alignment processing, were tailored to the specific needs of OPVQ.

The OpenVQ toolkit we present was created by identifying the areas that were suited for generalisation to provide a programming interface for a user who wants to implement another video quality metric. This part represented the last iteration of our development process.

5.2 Considerations

To develop our implementation from a stand-alone OPVQ imeplementation to a toolkit for any metric, we needed to consider how we could divide our code into general utilities and metric specifc parts.

Table 5.1 summarises the basic building blocks used in OPVQ. We have split the table in two parts where the top part includes the modules we believe can be generalised and used for any video quality metric. Notice that even though signal processing steps such as cropping and

Component	**Status**
Program options handling	Generaliseable
Video decoder	
Colourspace conversion	
Video sequence structure	
Frame structure	
Video processing system	
Cropping	
Alignment procedures	
Indicator calculations	OPVQ specific
Score Mapping	

Table 5.1: Component applicability in the general case

alignment are explicitly described in OPVQ, they are categorised as generaliseable. The idea is that such steps can be implemented as plug in modules or common functions, that users can add to their own metrics if needed, potentially saving resources compared to having to implement the same steps from scratch. The bottom part consists of OPVQ specific calculations and includes indicator calculation and Score mapping. These parts are most likely not suited for direct reusability in other metrics.

5.2.1 Video handling and processing

It seems favorable to handle video access to the video frames as transparently as possible in our toolkit. To come up with a design that works well in the general case, we make a few assertions about the general modus operandi of video quality metrics:

- All metrics will need access to the frames from a sequence.
- Metrics perform the same processing operations on all frames.
- Frames will be accessed in sequential order, because that's the way a human viewer sees the frames.

This is typically implemented by placing the processing operations in a loop that iterates over the video sequence frame by frame. This is known as a *processing pass*, demonstrated in listing 5.1.

In some cases, multiple passes may be needed. An example from OPVQ is the colour alignment, where the correction curves are based on

Listing 5.1: Pseudocode for a processing pass

```
**Data**: sequence : Video Sequence
**foreach** frame **in** sequence **do**
    some operation on frame;
    some other operation on frame;
    ;
**end**
```

histograms for the full sequences. Histograms are calculated in the first pass, after which the correction curves are calculated, and the correction curves are applied to the PVS frames in the second pass (sec. 3.3.3, p. 28).

We want to provide the notion of a processing pass as an abstraction called *makePass*. This way, a metric implementation can contain as many passes necessary. In programming terms, this can be implemented by having a function for every pass called a *function body*, in which the processing steps are collected (listing 5.3). These function bodies can be given as references to the *makePass* interface (listing 5.2).

Listing 5.2: Pseudocode for *makePass* interface (toolkit)

```
input: body : Function Reference;
Data: sequence : Video Sequence;
foreach frame in sequence do
    call body with reference to frame;
end
```

Listing 5.3: Pseudocode for function body (metric specific)

```
input: body : Function Reference;
some operation on frame;
some other operation on frame;
;
```

This separation lets the metric implementation be agnostic to all the details about video handling — it simply receives a reference to a frame. In the next section we talk about video files and memory handling, which because of this abstraction is now nicely contained as part of the toolkit, and none of the metric implementation's concern.

5.2.2 Memory and I/O

In the first OPVQ implementation we stored all video frames in memory during the analysis. This is fine for OPVQ as it is designed for short sequences of no higher than VGA resolution. However, other metrics that may want to analyse longer sequences or higher resolutions will run out of available memory on most platforms when using this design. An alternative design is to read video frames from disk when they are needed and freeing the memory afterwards. This means that each frame will be read from disk multiple times if they are needed during different stages of the analysis, which can cause the program to become I/O bound [1], but it elimitates the memory issue.

As we demonstrate in section 6.4.1, we decided to implement a variant of the second design where the memory issue is solved. Because video processing is extremely demanding for the CPU, the execution speed of most VQMs will be limited by the CPU and not I/O. This solution is designed with CPU processing in mind, and it should be noted that it may not be the best solution if a GPU is used for the computation. A GPU is a device with its own physical memory and I/O between the main memory and the device memory will have to be taken into account. We have not had the time to tackle the issues of GPU support and our solutions in chapter 6 will in certain cases reflect this.

5.3 Concept

Based on the separation shown in table 5.1, we came up with the following conceptual description of OpenVQ:

- A metric is implemented as a plug-in module, conforming to a common interface. OpenVQ can contain many different metric implementations, and the user may specify which one to use at run time.
- Common signal processing steps may also be implemented as plug-in modules. Such a module may be integral to the metric's algorithm, or it may be possible for the user to switch it on and off. For instance, if spatial alignment is a plug-in processing module part of a metric implementation, the user may wish to switch it off if he knows that SRC and PVS are already spatially aligned.

[1] I/O bound: The execution speed of a program is limited because of input-output operations.

- The toolkit should provide a facility for such run time configuration of the metric through user specified options.
- The toolkit contains a video decoding module that provides a high level representation of a video sequence as a sequence of image frames.

Figure 5.1 shows a high level schematic representation of the OpenVQ concept.

5.4 Summary

In this chapter we have described the basis for a general video quality toolkit, and our proposed concept for the interfaces that such a toolkit should provide the user.

In the next chapter we expand on the details of the implementation of OpenVQ.

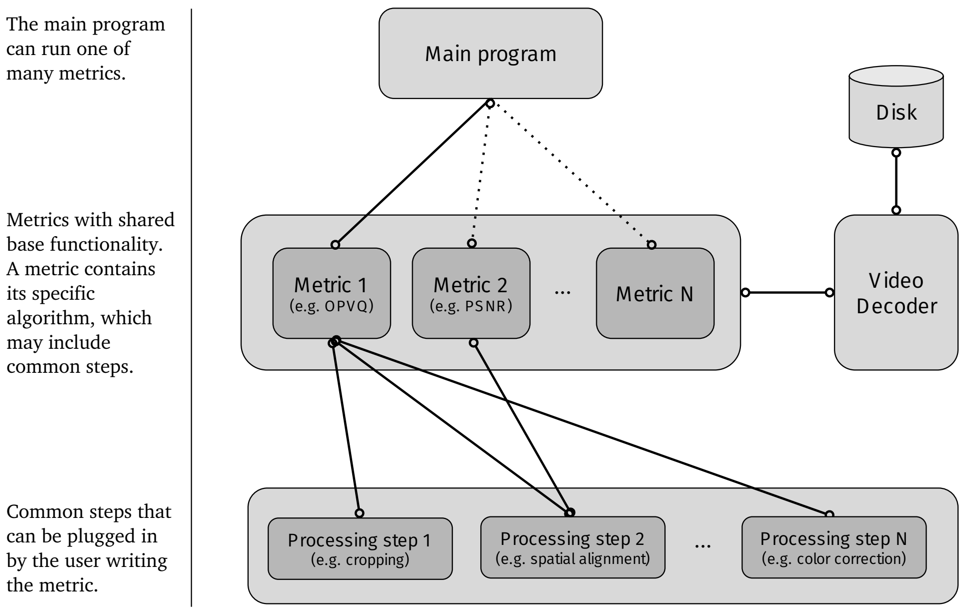

The main program can run one of many metrics.
Metrics with shared base functionality. A metric contains its specific algorithm, which may include common steps.
Common steps that can be plugged in by the user writing the metric.
Main program
Disk
Metric 1 (e.g. OPVQ)
Metric 2 (e.g. PSNR)
...
Metric N
Video Decoder
Processing step 1 (e.g. cropping)
Processing step 2 (e.g. spatial alignment)
...
Processing step N (e.g. color correction)

Figure 5.1: Conceptual overview of a Video Quality toolkit application

Chapter 6

Implementation

The considerations and requirements we proposed in the previous chapter serve as a conceptual foundation of the OpenVQ toolkit and its design. This chapter covers the implementation details of the toolkit. We describe the most important aspects of OpenVQ's implementation in detail, with particular weight given to the abstractions useful for implementing new metrics inside the framework of OpenVQ. A quick introduction to how a new metric may be implemented is summarised towards the end of this chapter.

6.1 Overview

In section 5.2 we provided an overview of components that we regard as common for all video quality metrics. These are the components that make up the bulk of OpenVQ. Figure 6.1 shows the most important building blocks in the OpenVQ implementation.

Some of the components from table 5.1 are self contained in one part of OpenVQ, while some are distributed across different parts. For instance Video decoding is nicely contained in the `VideoSequence` class. Implementing a metric involves inheritance from the `Algorithm` base class, which provides some useful abstractions to process the video sequence(s). In the following sections we describe the most important parts of the implementation in detail.

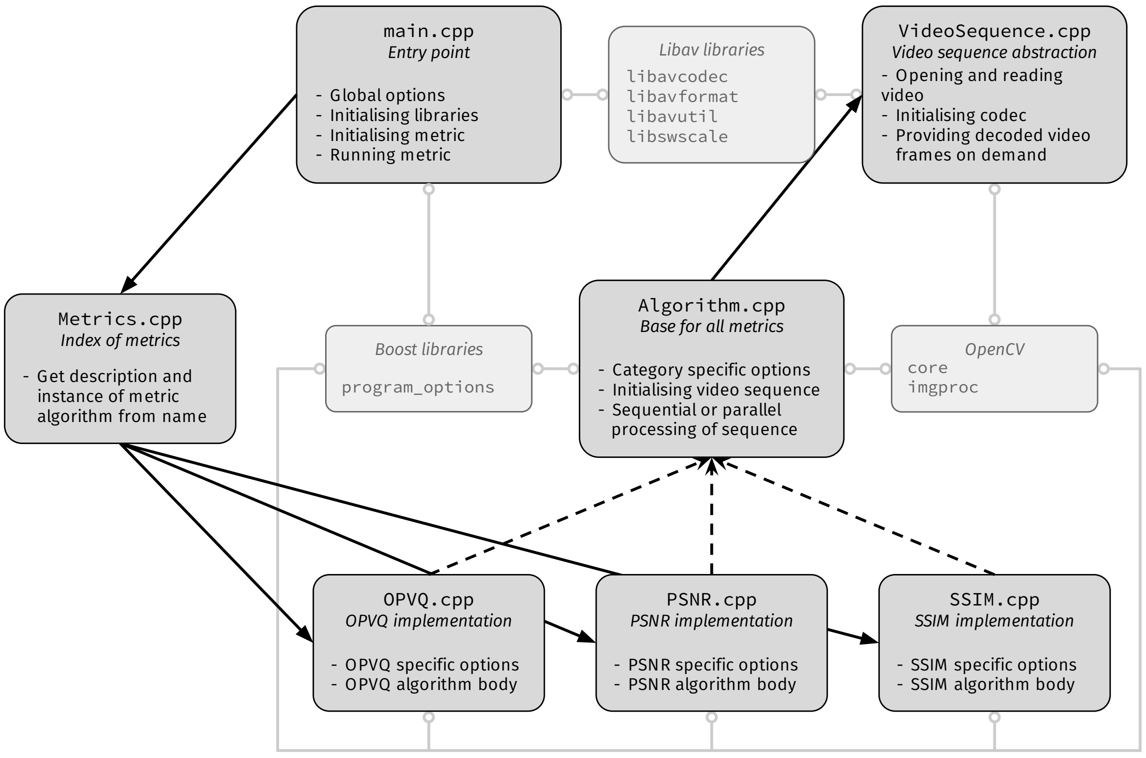


Figure 6.1: Schematic overview of the concrete OpenVQ implementation

6.2 Decoding video files

Providing decoded video frames is maybe the most fundamental abstraction in OpenVQ. Video is most often delivered together with audio. However, in most cases video and audio are encoded as separate media streams. Because of this, most multimedia file formats are so called *container formats*, that is files that contain multiple media streams that are supposed to be played back together. The container also includes the metadata about the streams that is necessary to know how to decode them. For a video stream, such information may include the video encoding format, pixel format and resolution and frame rate. Some containers — such as AVI — are agnostic to the encoding formats of the media streams, while others — such as MP4 — are more specialised to support only one or a few formats.

We wanted to be able to take virtually any multimedia file containing a video stream as input to our program. Libav is a free and open-source library to interact with multimedia formats and protocols, written in portable C.[1,2] Libav is widely used and has an active community, and was a good choice for us to handle interaction with different multimedia formats. Libav is licensed under the GNU LGPL ver. 2.1, with optional parts that, if enabled, apply the more restrictive GNU GPL ver. 2.[3]

In this section we describe the steps necessary to read an arbitrary video file into memory as a sequence of decompressed video frames using the Libav library suite.

6.2.1 Opening a video file

The first step when opening a multimedia file for reading is to determine the container format. The Libav library that deals with file formats is called `libavformat`. Libav recognises container formats based on the file extension appended to the filename — e.g. `.avi` or `.mp4` — so for well known containers, the user only needs to supply the path of the file to be opened.

Listing 6.1: Opening video files with `libavformat`

```
AVFormatContext *formatContext = avformat_alloc_context();
avformat_open_input(&formatContext, "movie.avi", NULL, NULL);
```

Listing 6.2: Finding video stream and initialising codec

```
avformat_find_stream_info(formatContext, NULL);
int videoStream = -1;
for (int i = 0; i < formatContext->nb_streams; i++) {
    AVStream *stream = formatContext->streams[i];
    if (stream->codec->codec_type == AVMEDIA_TYPE_VIDEO) {
        videoStream = i;
        break;
    }
}
AVCodecContext *codecContext = formatContext->streams[i]->codec;
AVCodec *decoder = avcodec_find_decoder(codecContext->codec_id);
avcodec_open2(codecContext, decoder, NULL);
```

Next, we must examine the container and determine the video stream that we want to decode. The container can contain multiple streams, but more often than not there will be only one video stream. In OpenVQ we use only the first video stream in the container's index. When the stream

[1]Libav https://libav.org/about.html (*visited 20/1/2015*)
[2]Libav is a fork of the FFmpeg project https://www.ffmpeg.org/about.html
[3]Details about the GNU licenses can be found at https://www.gnu.org/licenses/

is identified, we determine the encoding format of the stream, and we initialise the appropriate codec[4] if the format is recognised. This is done by a few simple calls to Libav, which allocates buffers and initialises internal structures as needed behind the scenes. In return we get a *codec context*, a data structure that contains all information necessary to decode the stream. This context will be passed back to Libav during the subsequent calls. To interface with the available codecs, we use the library `libavcodec`.

6.2.2 Decoding the video stream

When our codec context is initialised, we can start reading the video stream into memory. Even though different streams are encoded separately, the data is often interleaved to ensure sequential access when reading from disk. `libavcodec` has the notion of a *packet*, which is a data structure containing a sequence of bytes from the data stream along some info about the data, such as the stream index to which the data corresponds. By reading the data stream piece by piece into such packets, we can de-interleave the stream by assigning input tagged with a specific stream index to a corresponding handler. In our case, we're only interested in the input for the previously determined `videoStream` index, and are free to discard data from the other streams. Listing 6.3 shows the code necessary to decode a video stream and place it in an AVFrame.

Listing 6.3: Reading and decoding a frame from the video stream

```
AVPacket packet;
av_init_packet(&packet);
int currentStream = -1;
while(currentStream != videoStream) {
    if (av_read_frame(formatContext, &packet) < 0) {
        // Reached end of file; return
    }
    currentStream = packet->stream_index;
}
AVFrame *frame = avcodec_alloc_frame();
avcodec_decode_video2(codecContext, frame, NULL, &packet);
```

The AVFrame structure contains pointers to the decoded frame in memory. A video frame generally consists of multiple channels to make up a colour image, either in a RGB-type colour space or as $Y'C_BC_R$.

[4]*Codec*: Software that encodes and decodes a specific media encoding format (*coder-decoder*).

Channels are not necessarily of equal size due to chroma subsampling.[5] Libav can convert a frame's pixels to a specific format. The user must be aware of the fact that some conversions are lossy and can introduce distortion of its own, which is a problem for video quality assessment. Other conversions, such as upscaling without interpolation does not affect the data. Listing 6.4 is an example of converting a frame from planar $Y'C_BC_R$ 4:2:0 to planar $Y'C_BC_R$ 4:4:4, using the Libav library `libswscale`.

Listing 6.4: Converting between pixel formats using `libswscale`

```
// Allocate space for 4:4:4 data
size_t sz = frame->width * frame->height;
uint8_t *out[3] = {new uint8_t[sz], new uint8_t[sz], new uint8_t[sz
    ]};
int lineSizes[3] = {frame->width, frame->width, frame->width};

SwsContext *swsContext = sws_getContext(
        frame->width, frame->height, AV_PIX_FMT_YUV420P,
        frame->width, frame->height, AV_PIX_FMT_YUV444P,
        SWS_X, NULL, NULL, NULL
);
sws_scale(swsContext, frame->data, frame->linesize, 0,
          frame->height, out, lineSizes);
```

At this point, we have a raw video frame stored in memory on the format we want to work with.

6.3 Structures and representation

6.3.1 Frame representation

The uncompressed video frames decoded with `libavcodec` are stored as arrays of pixel values. However, mathematically we think of a video frame in terms of a matrix of pixel values, and we can apply matrix operations on whole frames at a time. We went in search for a library that implemented this abstraction, and found the Open Source Computer Vision (OpenCV) library to be a good fit. OpenCV is based on such a matrix representation of image frames, and has defined a lot of mathematical operations on these structures. These include basic arithmetic operations and statistical calculations as well as more complex operations like image filtering, feature detection, motion analysis and

[5]More about chroma subsampling in Poynton (2012, pp. 121–128)

object tracking.[6] OpenCV is very widely used and has a large and active community.

By not having to implement all the matrix operations ourselves, we eliminated possible sources of error and ended up with leaner and more readable code. All this made it an easy decision to use OpenCV to handle the image processing and to hold our video frame data in its matrix container.

Listing 6.5: Adding a scalar to all pixels in a frame, with and without OpenCV

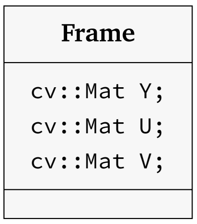
```
// Manually
uint8_t *out_array = new uint8_t[f->width * f->height];
for (int i = 0; i < f->width * f->height; i++) {
    out_array[i] = f->data[0][i] + 80;
}

// Using OpenCV
cv::Mat out_matrix(f->height, f->width, CV_8UC1, f->data[0]);
out_matrix += cv::Scalar(80);
```

Listing 6.6: Applying an arbitrary filter kernel using OpenCV

```
cv::Mat in(f->height, f->width, CV_64FC1, f->data[0]);
cv::Mat out(in.rows, in.cols, in.type())
cv::Mat kernel = (cv::Mat_<double>(1,5) << -1, -1, 0, 1, 1);
cv::filter2D(in, out, in.depth(), kernel);
```

6.3.2 The `Frame` class

Because a colour frame consists of three channels, we need three `cv::Mat` objects to hold the data for a single frame. A `cv::Mat` can actually hold multiple channels itself, but as OpenCV always treat multi channel matrices as RGB frames, this was not a viable option as we needed to support $Y'C_BC_R$. We solved this by creating a small data structure that wraps three `cv::Mat` objects, one for each of the channels.



```
cv::Mat Y;
cv::Mat U;
cv::Mat V;
```

Figure 6.2: The `Frame` struct in OpenVQ

[6] Open Source Computer Vision http://opencv.org/about.html (*visited 19/1/2015*)

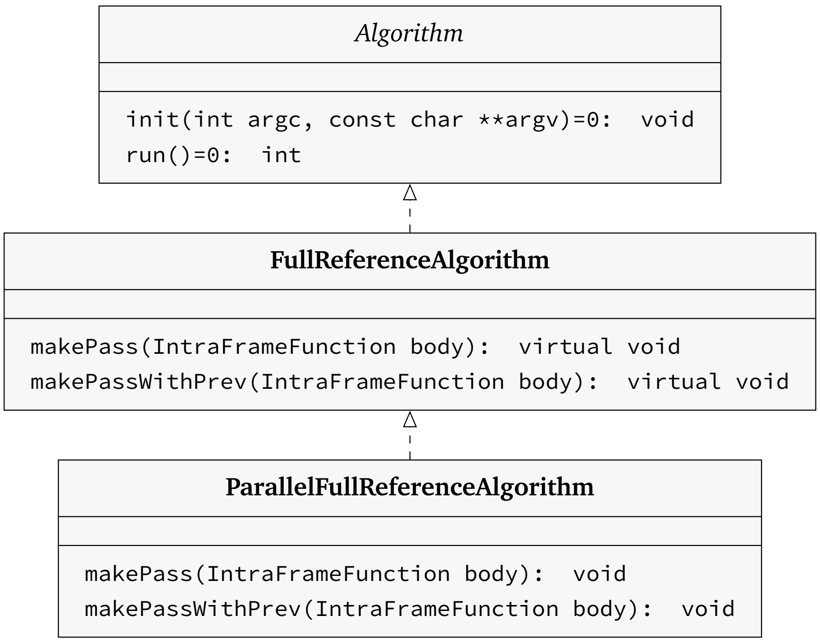

Algorithm
init(int argc, const char **argv)=0: void
run()=0: int
FullReferenceAlgorithm
makePass(IntraFrameFunction body): virtual void
makePassWithPrev(IntraFrameFunction body): virtual void
ParallelFullReferenceAlgorithm
makePass(IntraFrameFunction body): void
makePassWithPrev(IntraFrameFunction body): void

Figure 6.3: OpenVQ algorithm Interface.

6.4 The `Algorithm` interface

Abstracting the video decoding step goes a long way to make it easy for a user to implement his or her own metric. Still, there is room for further generalisation. For instance for full reference metrics, we know that we will require two video sequences, SRC and PVS. Also, many metrics will access one frame or maybe two consecutive frames at a time to perform a processing step. Doing this for all the frames in a video sequence is known as a *processing pass*. Some algorithms may be performed in a single pass, while other may require a few passes to do their job. These tasks are implemented in the base classes in the `Algorithm` class hierarchy in OpenVQ.

Our algorithm interface was written as generic as possible to fit any video quality metric. The actual interface is an abstract class with a constructor and two purely virtual functions, `init` and `run`. Any metric using this interface must override these functions. We wanted to limit the extent of `Algorithm` to this simple interface. On the one hand, we provide as much freedom as possible to the author of a new metric implementation. On the other hand we make it as simple as possible to plug an algorithm into the function that wants to run it — two simple function calls.

From `Algorithm` we derive another class `FullReferenceAlgorithm`. This class implements the two functions `makePass` and `makePassWith-`

Prev that facilitate making a processing pass over the SRC and PVS sequences. Because video processing is CPU intensive work, we derive yet another class ParallelFullReferenceAlgorithm that provides a parallel variant of the processing pass functions.

We have limited the current implementation to provide abstractions for Full Reference metrics only. A natural continuation of the development is to provide equivalent derivations for Reduced Reference and No Reference metrics. Figure 6.3 presents the current class hierarchy.

6.4.1 Processing passes

We wanted to make it easy to implement a metric that operates by performing a series of processing steps on SRC and PVS, frame by frame. As discussed in section 5.2.1, one or a set of consecutive processing steps can be implemented as a function, called a *body*, that receives a reference to the frame at a given position t. If only one frame is handled at a time, we define it as an *intra-frame function* body. If the steps process two frames at a time from each sequence, we call the function an *inter-frame function* body. This is analogous to the discussion in 3.4.

We make use of the C++11 standard library functional to define these two function types:

Listing 6.7: Definitions of IntraFrameFunction and InterFrameFunction

```
typedef std::function<void(std::shared_ptr<Frame> srcFrame,
                           std::shared_ptr<Frame> pvsFrame,
                           int t)>
    IntraFrameFunction;

typedef std::function<void(std::shared_ptr<Frame> srcCurr,
                           std::shared_ptr<Frame> pvsCurr,
                           std::shared_ptr<Frame> srcPrev,
                           std::shared_ptr<Frame> pvsPrev,
                           int tCurr)>
    InterFrameFunction;
```

The functions makePass and makePassWithPrev respectively take references to an IntraFrameFunction and an InterFrameFunction as arguments. The body is applied sequentially to the set of sequences, from $t = 0$ all the way to $t = N - 1$, N being the length of the sequences. The processing pass functions in ParallelFullReferenceAlgorithm provide the additional feature of applying the body on j adjacent frames in parallel, using threads. Metric implementations derived from this base class must ensure that the function body passed to `makePass` or `makePassWithPrev` is thread safe.

Listing 6.8 is a pseudocode representation of the `makePass` function from `FullReferenceAlgorithm`. This function operates on a frame by frame basis without any knowledge of any previous or coming frames. The input to the function is, as mentioned, a function body that performs the the metric specific calculations. Our `makePass` function then runs a loop until the entire video is processed, preparing a frame from each sequence in each iteration before calling the function body with references to the SRC and PVS frames as arguments.

Listing 6.8: Pseudocode for `makePass`

```
input: body : IntraFrameFunction
while not eof do
    srcCurr := next SRC frame;
    pvsCurr := next PVS frame;
    body(srcCurr, pvsCurr);
end
```

Listing 6.9 represents `makePassWithPrev` and is an extension of listing 6.8. It stores the previous frame for each pass such that it is possible to compare two frames.

Listing 6.9: Pseudocode for `makePassWithPrev`

```
input: body : InterFrameFunction
while not eof do
    srcCurr := next SRC frame;
    pvsCurr := next PVS frame;
    body(srcCurr, pvsCurr, srcPrev, pvsPrev);
    srcPrev := srcCurr;
    pvsPrev := pvsCurr;
end
```

Listings 6.10 & 6.12 serve the same purpose as listings 6.8 & 6.9, but have been modified to work on multiple frames in parallel. The parallelised variants use worker threads that run jobs stored in a job queue. A job in this context is a function body and the according arguments.

This is an instance of the classic *Producer-Consumer problem*[7], with 1 producer — the main thread — and J consumers — the worker threads.

[7]https://en.wikipedia.org/wiki/Producer-consumer_problem

J is defined in the variable `jFactor`. This number is automatically determined to match the number of hardware threads supported by the platform, but it can be overridden by the user on the command line. `jFactor` also defines the capacity of the job queue. The main thread will try to keep this filled as long as there are jobs left. This ensures that even if all J threads want to remove a job from the queue at the same time, none of them need to wait for I/O.

Listing 6.10: Pseudocode for parallel `makePass`

```
input: body : IntraFrameFunction;
initialise jobQueue;
launch jFactor worker threads;
while not eof do
    srcCurr := next SRC frame;
    pvsCurr := next PVS frame;
    jobQueue.push(Job(body, srcCur, pvsCurr));
end
jobQueue.close();
foreach worker thread do join();
```

Listing 6.11: Pseudocode for worker thread

```
input: jobQueue;
repeat success := jobQueue.pop().run();
until not success;
```

Listing 6.12: Pseudocode for parallel `makePassWithPrev`

```
input: body : InterFrameFunction;
initialise jobQueue;
launch jFactor worker threads;
while not eof do
    srcCurr := next SRC frame;
    pvsCurr := next PVS frame;
    jobQueue.push(Job(body, srcCurr, pvsCurr, srcPrev, pvsPrev));
    srcPrev := srcCurr;
    pvsPrev := pvsCurr;
end
jobQueue.close();
foreach worker thread do join();
```

6.5 User options

The main use case of OpenVQ is that a user wants to run a specific quality metric on a source and a processed video sequence. These video sequences must somehow be passed to the program at run time. The individual metric may itself be further configurable or provide additional features on demand, so the user needs to be able to pass along metric specific options as well. Lastly, it may be preferable to provide the user with some options that specify the behavior of the program itself, such as the amount of information to be output to screen. In this section we describe how this complex system of user specifiable options is handled in all parts of our program.

6.5.1 Command line syntax

The command line syntax for OpenVQ is defined as follows:

```
openvq [global options] command [command specific options]
```

The `global options` and `command specific options` are either keyword arguments or flags. Keywords and flags always have one or two leading dashes. These options can be either required or optional. The `command specific option` are passed along to the metric implementation, which gives the implementation all the flexibility it needs to be configured at run time. The `command` itself is interpreted as a string, and specifies which metric OpenVQ invokes. For example, the following command line will run OPVQ with default settings:

```
openvq opvq --src <path to SRC> --pvs <path to PVS>
```

For an in depth view of the possible options, see section 7.1.3.

6.5.2 Options handling

Our program options parsing system is based on the `boost.program_options` library[8], which provides all the facilities necessary to parse command line arguments. The syntax itself is defined using one or more so called *description* objects, which are passed to a parser that parses the options and stores the result in a map structure.

[8]Boost Program_options http://www.boost.org/doc/libs/1_51_0/doc/html/program_options.html (*visited 19/1/2015*)

Listing 6.13: Minimal `boost.program_options` example

```
namespace po = boost::program_options;

int main(int argc, const char **argv) {
    po::options_description opts("Options");
    opts.add_options()("message,m", "Print message");
    po::variables_map vars;
    po::store(po::command_line_parser(argc, argv).
              options(opts).run(), vars);
    if (!vars.count("message")) {
        std::cout << "Hello, World!" << std::endl;
    } else {
        std::cerr << opts;
    }
}
```

The minimal program from listing 6.13 yields the following behaviour:

```
user@host: ~ $ program_options_example
Options:
  -m [ --message ]      Print message
user@host: ~ $ program_options_example -m
Hello, World!
user@host: ~ $ program_options_example --message
Hello, World!
```

A `command_line_parser` supports two types of description objects; `options_description` and `positional_options_description`. Description objects of both types can be added to the same parser object. In this case, the parser identifies keyword arguments and flags by leading dashes, and separates them from the rest. Any remaining arguments are parsed as positional arguments. In the OpenVQ case, the `global options` are defined in an `options_description` object, and the `command` is defined as the only argument in a `positional_options_description`.

The `command_line_parser` lets the user specify that unknown options should not raise errors. We take advantage of this to ignore any command specific options in the main program. The full argument vector (`argv`) and the number of elements (`argc`) are passed on to the `init` function of the metric implementation, which contains its own set of description and parser objects. This structure makes it easy for the authors of a new metric implementation to add all the user specifiable options they want.

6.6 Implementing a metric

In the previous sections we have outlined the most important parts of the OpenVQ toolkit. Figure 6.4 gives a step by step summary of how to implement a video quality metric in OpenVQ.

Step 1. Derive a new class from one of the `Algorithm` base classes
Step 2. Implement functions
    2.1 Constructor (optional): Set up metric specific options.
    2.2 `init`: Call base class init. If any, handle metric specific options.
    2.3 `run`: Perform the algorithm body, using `makePass` if applicable.
Step 3. Register the class in the `Metrics` index in the OpenVQ main program

Figure 6.4: Procedure to implement a metric in OpenVQ

6.7 Access and licensing

OpenVQ is released as Free Software. The source code is maintained under version control, and the repository is available at:

https://bitbucket.org/mpg_code/openvq

OpenVQ is distributed under the GNU Affero General Public License version 3.[9] We chose the Affero variant of the General Public License because it covers the case where a modified version of the software is not distributed, but provided as a service accessible over the Internet. Providing a service of this kind is not counted as distribution as defined in the GNU GPL, and modified versions of a program licensed under GNU GPL can be provided as an Internet service without the service provider having to publish modifications made to the source code. The GNU Affero GPL variant remedies this, by requiring the service provider to make the modified source code available.

Even though OpenVQ in its current incarnation is not designed as a network service, it's not hard to imagine video quality assessment service that accepts a pair of *SRC* and *PVS* sequences, and gives back a *VQR*. Creating such a service based on OpenVQ is in no way prohibited, and neither is charging for it, but a service provider would have to make the modified source code freely available under equal licensing terms as OpenVQ. We consider this a fair requirement.

[9]GNU AGPL https://www.gnu.org/licenses/agpl-3.0.html (*visited 29/4/2015*)

6.8 Summary

In this chapter we have provided details about the implementation of OpenVQ, and the libraries OpenVQ employs for certain parts of the processing. This can be used as a helpful guide to the structure of the program and serves as a context when reading and using the source code.

Chapter 7

Using OpenVQ

In this chapter we provide an introduction to general use of OpenVQ and OPVQ. We also demonstrate how to implement a metric for the toolkit, using our PSNR implementation as an example.

7.1 Installation and usage

7.1.1 Prerequisites

Before building OpenVQ, some dependencies have to be in place. Table 7.1 lists the required programs or libraries. All the dependencies

Dependency	*Version*	*Usage*
`CMake`	3.0	Generating a project for the desired target platform. Can also be used for building.
`Libav`	9	All video I/O operations.
`OpenCV`	2.4	Digital signal processing, mathematical operations on images.
`Boost`	1.54	Handling and parsing of program options.

Table 7.1: Dependency listing for OpenVQ

are well known and freely available, and can easily be installed using a package manager. For example, the following command will install all the dependencies on a Debian based Linux distribution:

```
user@host: ~ $ apt-get install cmake libavcodec-dev libavutil-dev
    libavformat-dev libswscale-dev libopencv-dev libboost-program-
    options-dev
```

7.1.2 Setup

OpenVQ is distributed as a CMake project. CMake is an extensible, open-source system that manages the build process in an operating system and in a compiler-independent manner.[1] In principle, this means that OpenVQ can be built and run on any platform for which CMake generates project files. We have successfully built and run OpenVQ on the UNIX-based platforms Linux and Mac OS X.

For UNIX systems, CMake can be invoked from the command line:

```
user@host: ~/openvq-build-dir $ cmake <path-to-openvq-source-dir>
-- The CXX compiler identification is GNU 4.8.2
-- Check for working CXX compiler: /usr/bin/c++ -- works
[...]
-- Found OpenCV version 2.4.8
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user/openvq-build-dir
```

If no other project type is specified, CMake will generate a Makefile that can be invoked with Make.

```
user@host: ~/openvq-build-dir $ make
 make
Scanning dependencies of target openvq
[  5%] Building CXX object src/CMakeFiles/openvq.dir/metrics/common
    /alignment/SpatialAlignment.cpp.o
[ 11%] Building CXX object src/CMakeFiles/openvq.dir/metrics/common
    /alignment/ColourAlignment.cpp.o
[...]
Linking CXX executable openvq
[100%] Built target openvq
```

The program is now built, and is ready to be installed into the system path.

```
user@host: ~/openvq-build-dir $ make install
[100%] Built target openvq
Install the project...
-- Install configuration: ""
-- Installing: /usr/local/bin/openvq
```

The default install prefix is `/usr/local`. A different prefix can be changed when CMake is invoked to generate the build files.

[1]CMake — the cross-platform, open-source build system http://www.cmake.org/overview/ (*visited 20/1/2015*)

```
user@host: ~/openvq-build-dir $ cmake <path-to-openvq-source-dir> -
    DCMAKE_INSTALL_PREFIX=<prefix>
```

7.1.3 Running the program

OpenVQ provides a simple command line interface. Running the program with `--help` or without any arguments displays the help screen.

```
user@host: ~ $ openvq
[INFO] OpenVQ:
Usage:
  openvq [global options] command [command specific options]

Available commands:
  opvq                    Open Perceptual Video Quality metric
  psnr                    Peak Signal-to-Noise Ratio
  ssim                    Structural Similarity Index

Global options:
  -h [ --help ]         Print help message
  --log-level arg       Set log level threshold {trace,debug,info,
      warn,error}

To get additional help for the individual commands, run:
  openvq <command> --help
```

The individual metrics can specify their own specific options, in which case they provide their own additional help screen.

```
user@host: ~ $ openvq opvq --help
[INFO] OpenVQ: Allowed options for algorithm OPVQ:
  -t [ --max-frames ] arg      Set frame limit
  -h [ --help ]                Print help message
  --csv arg                    Path to csv file to which the indi-
                               cators will be added as a new line
  -s [ --src ] arg             Path to source video sequence
                               (required)
  -p [ --pvs ] arg             Path to processed video sequence
                               (required)
  -j [ --num_threads ] arg     Number of threads wanted. If no
                               value is given, the algorithm tries
                               to determine the number of threads
                               supported by the hardware.
  --disable-spatial-alignment  Disable spatial alignment
  --disable-colour-correction  Disable colour correction
```

An example run of the OPVQ algorithm may look like the following.

```
user@host: ~ $ openvq opvq -s src01_hrc00.avi -p src01_hrc09.avi
07:48:29 [INFO] SRC is src01_hrc00.avi
07:48:29 [INFO] PVS is src01_hrc09.avi
07:48:29 [INFO] Pass 1
[========================================================>] 100%
07:48:35 [INFO] Pass 2
[========================================================>] 100%
LINEAR OFFSET: 63.1414
INDICATOR      VALUE          MIN         MAX         CONTRIBUTION
Luma           7.118528       0.000000    26.345892   3.443753
Chroma         1.021960       0.088887    11.934138   -61.996631
Omitted        6.186605       0.000000    1603.352661 -1.081965
Introduced     1.238975       0.000000    44.038914   -0.221943
Final score: 3.284585 (will be clipped to [1, 5])
07:48:44 [INFO] Aggregated final score: 3.28459
```

7.2 Implementing PSNR with OpenVQ

Up to this point we have provided an explanation of the each intricate part of OpenVQ, how they were designed and how they can be used. To better demonstrate for potential users how other metrics may be implemented with OpenVQ we have added a PSNR and SSIM implementation with it. As previously described (2.3.1) PSNR is a simple metric that by itself is nothing more than a MSE calculation over each frame. Using OpenVQ we are however able to make use of our spatial alignment module to potentially achieve better subjective correlation. In this section we explain how we implemented PSNR with OpenVQ and utilised our spatial alignment module.

Implementing a new metric in OpenVQ can be very simple. In the case of PSNR we can utilise several of the modules included in OpenVQ such that we only need to write some setup-code next to the MSE calculation. The listed steps below summarise our PSNR implementation.

- Create a new class, `PSNR`, inheriting one of the algorithms.
- Implement the following functions in the new class:
  - `constructor`: Add program-option for spatial alignment and initialise accumulation values.
  - `init`: Call parent class init and enable/disable spatial alignment
  - `run`: Write the PSNR code in lambda fuction passed to `makePass`.
- Add the PSNR class to the metrics list in metrics.cpp.

The first step in implementing a new metric in OpenVQ is choosing the base class. PSNR is a full reference metric so we can choose between FullReferenceAlgorithm and ParallelFullReferenceAlgorithm.

While PSNR can easily be paralleised we chose to use the FullReferenceAlgorithm to make the implementation as simple as possible. We can now make a PSNR class that is derived from FullReferenceAlgorithm. Our new class only needs to implement a constructor, the two abstract functions from the parent class (init and run) and one function that performs the PSNR calculation itself.

The constructor (listing 7.1) has the job of calling the parent constructor, providing a name for the metric. In the case of our PSNR implementation we are passing the string PSNR as an argument to the parent class and initialising two variables used for the PSNR calculation, psnrAccum and framesCalculated, to zero. In addition, the constructor add any metric specific program options to a list of available options. For our PSNR implementation we add an option to disable or enable the spatial alignment module.

Listing 7.1: The PSNR constructor

```
PSNR::PSNR()
        : FullReferenceAlgorithm("PSNR"),
          psnrAccum(0), framesCalculated(0) {
    options.add_options()("disable-spatial-alignment",
            "Disable spatial alignment");
}
```

OpenVQ passes along the command line arguments to the init function (listing 7.2). These are passed through to the parent class' init function which will handle any options inherited through the algorithm hierarchy. After that is done our init function can parse the same array for the options we added in the constructor.

The next function to implement is run. This function will by called by OpenVQ after init has completed, and it is from this function we perform all the calculations. We can choose to implement the entire metric here, including accessing video frames through the parent class, or we can use one of the makePass functions available from the parent class which does this work for us. Since we do not need access to more than one SRC and PVS frame at a time we use the intra-frame algorithm, named makePass (listing 6.8). The makePass function takes a function body as parameter to which we can write the code required to calculate the PSNR value of one video frame and store the result locally in the PSNR class instance.

Finally, after the call to `makePass` has terminated, we can calculate the final PSNR value by finding the mean of the accumulated variable used in the function body. Listing 7.3 shows our `run` code. The functions `addFrame` and `calcPsnr` are simply functions wrapping the functionality for calculating the PSNR of a single frame and the final mean value respectively.

Listing 7.2: The PSNR `init` function

```
void PSNR::init(int argc, const char **argv) {
    FullReferenceAlgorithm::init(argc, argv);
    opts::variables_map vm;
    opts::parsed_options parsed =
        opts::command_line_parser(argc, argv).
        options(options).
        allow_unregistered().
        run();
    opts::store(parsed, vm);
    enableSpatialAlignment = !static_cast<bool>(
        vm.count("disable-spatial-alignment"));
}
```

Listing 7.3: The PSNR `run` function

```
int PSNR::run() {
    SpatialAlignment spatialAlignment;
    makePass([&](std::shared_ptr<Frame> srcCurr, std::shared_ptr<
        Frame> pvsCurr, int tCurr) {
        if (enableSpatialAlignment) {
            int crop = 1;
            cv::Point2i sptialOffset = spatialAlignment.
                spatialOffsetDetermination(srcCurr, pvsCurr, crop);
            spatialAlignment.cropAndAlign(srcCurr, pvsCurr, crop,
                sptialOffset);
        }
        addFrame(srcCurr, pvsCurr);
    });
    double psnr = calcPsnr();
    std::vector<double> values = {psnr};
    writeCSV(pvsURL, values);
    return 0;
}
```

7.3 Summary

In this chapter we have shown how to build, install and run the OpenVQ toolkit and OPVQ. We have also shown how to implement new metrics inside the framework of OpenVQ, using our implementation of PSNR as an example.

Part IV

Conclusions

Chapter 8

Conclusions

8.1 Contributions

Working with this project has been a learning experience from start to finish. These lessons will hopefully have been conveyed and be of interest to the readers of this thesis. As the project consists of two separate parts, the matter of contributions is two fold as well.

In terms of the work pertaining to the metric itself, what was ostensibly a simple matter of implementing a well defined metric for video quality turned out to require significant analytical effort. This analysis is a significant contribution for anyone interested in learning about the workings of the standardised PEVQ metric in particular, and objective video quality metrics in general.

The other significant contribution is the OpenVQ toolkit released as Free and Open Source Software. The project will live on and enjoy continued development and maintenance under the ownership of the Media group at Simula Research Laboratory in Oslo. We view this as a step in the right direction to bring more openness and transparency to the video quality community, which from the outside appears very much closed and opaque. A planned research paper submission about the toolkit will, if accepted, hopefully aid in gaining publicity and possibly brining in contributors from different parts of the community.

8.2 Further Research and Development

As all research projects, this project has been subject to time and resource constraints. During the course of the research, we have thought of many possible options to explore that we have not had the time to pursue. In this section we briefly mention some of the possibilities for further work that we think will prove the most rewarding if explored.

8.2.1 Support for higher resolutions

OPVQ has currently been tested and validated for VGA resolution only. Video content today is to a large extent delivered at resolutions higher than VGA. Given this current situation, finding a way to support higher resolutions — specifically Full HD or higher — would be very relevant.

The amount of data in a frame of Full HD video is much larger than in a VGA frame, but is not necessarily displayed on a larger display. This means that the density of information is higher. An interesting question is whether or not this affects OPVQ's ability to assess quality. To investigate this, a good place to start would be to try to determine a set of mapping coefficients that correlate well with subjective tests for higher resolutions. If this proves difficult, the indicator calculations may need to be changed, or altogether new indicators may be added and mapped into the final score. As mentioned in chapter 2, the model from SwissQual AG standardised in ITU-T J.341 supports Full HD resolution. Elements from this metric may be applicable — or at least of interest — if changes to the model itself turn out to be necessary.

8.2.2 Support for hybrid models

In the introduction of this thesis, we state that video streaming is responsible for a large amount of the internet traffic generated today. Streaming video over networks introduces issues such as loss of quality due to bitrate adaptation and buffer underrun events. There is ongoing research into how such events affect the QoE over time.

To assess the quality of streamed video, it may be of interest to take into account not just the decoded video frames, but other information such as the bitstream itself and information carried in the protocol headers of the packets that deliver the video. Such models are called *hybrid models*, and VQEG have recently published a recommendation on hybrid models through ITU-T, as mentioned in section 2.4.4.

To facilitate research into streaming related issues, abstractions that deal with streaming video should probably be implemented as part of the OpenVQ toolkit. The ITU-T Rec. J.343 will be of great interest in this context when published, as it may contain metric descriptions that can serve as a basis for such an extension.

We have considered a general design on our own that would allow OpenVQ to support streaming input of arbitrarily long video sequences by using a sliding window mechanism. When receiving streamed video, the frames will be buffered until the window is filled, at which point a VQR can be calculated for the buffered window. When a new frame arrives it is buffered and the oldest frame in the buffer is pushed out. The VQR can then be re-calculated for the updated buffer. This setup would enable a more real time visualisation of the predicted quality.

For OPVQ, there are some challenges when it comes to the sliding window solution, as recalculating the score for the entire window whenever a new frame enters would prevent it from running in real-time. However, figuring out a way to reuse for instance partially calculated values from previous windows could make it possible to achieve real-time performance.

8.2.3 Temporal Alignment

As we discussed in section 3.3.1 we chose to leave temporal alignment out of OPVQ. Even so, temporal alignment may be useful to have as a part of the metric, but it would need to be implemented based on something else than the PEVQ description. A paper written by Barowsky, M., Bialkowski, J., Bitto, R., and Kaup, A. Barkowsky et al. (2007) mentions the temporal alignment used in PEVQ, and proposes an improvement on this model. This model may be possible to use for a temporal alignment implementation in OPVQ, but as both Barowsky, M. and Bitto, R. are affiliated with OPTICOM the patent situation must be examined.

8.2.4 GPU utilisation

We briefly mention in section 5.2.2 that we have not considered GPU interaction when implementing OpenVQ. There is no doubt, however, that facilitating for use of GPU devices would be a relevant development for OpenVQ. Specifically, creating `makePass` abstractions that load Frame objects into device memory instead of main memory could be a good place to start. The latest major revision of OpenCV (version 3) contains a new API, called *Transparent API*, that is supposed to make use of hardware devices through OpenCL in a transparent fashion. It should be investigated if this API has application in OpenVQ.

Chapter 9

Reflections

Throughout our development of OPVQ and OpenVQ we have learned a lot about video quality measurement and software engineering. We want to summarise some of the key experiences and perhaps provide some pointers for others who wish to continue our work with a masters or doctoral thesis. The contents of this section will not be directly relevant to the scientific findings presented in this thesis, but rather act as an informal summary of our own subjective opinions and experiences gathered during the work.

While we did not have any good estimate on the time necessary to develop an implementation of the standardised PEVQ model when we started, we were determined to at least achieve the goal of proividing a working metric implementation. In addition, the inital plan was that at some point when we had a working version of PEVQ we would diverge into different directions. One of us would then most likely research possibilities for development of additional indicators and/or resolution support, while the other would work on implementing these features.

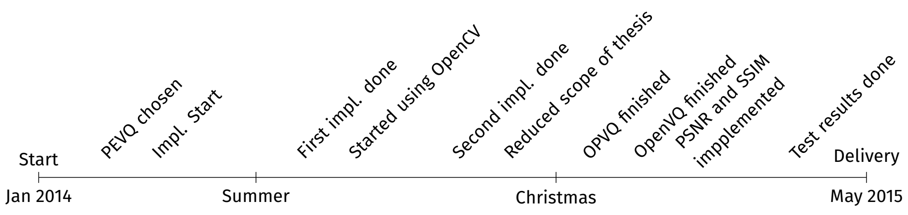
As soon as we started work on the implementation we identified some of the inconsistencies in the PEVQ description, and suspected that the time required would be longer than we first had hoped. Around the middle of September 2014 we had finished the first implementation of PEVQ. At this stage the only library we had used was Libav for video decoding, the rest of the image processing was written by us from sctrach. We were not able to produce good results with this version of the software, We made an effort to improve results, but after a few weeks worth of debugging and fixing small errors, we decided to make some big changes.

These changes were two fold. First of all, the changes involved simplifying our solution as much as possibly by using tried and tested libraries for as much as possible. This was perhaps the most important choice we made from a software engineering point of view. Secondly, as we progressed step by step through the implementation, we analysed in detail the mathemtical description in PEVQ. This also turned out to be a crucial part of the work. Within a few months of hard work we had rewritten the entire implementation using OpenCV for all the image processing. This automatically removed some bugs we had not discovered and simplified the solution by making it shorter and more understandable.

In hindsight we would probably have saved time by implementing OPVQ using OpenCV from the beginning. At the same time we were forced to thoroughly analyse every step of the PEVQ description when we wrote the OpenCV implementation, which further improved our understanding of every aspect of the metric. It is, however, a testament to the fact that with all the open source libraries available for all kinds of applications, spending a little time to find helpful libraries is a worth while exercise.

The analysis of the PEVQ description turned out to be a crucial part of the work, as it uncovered a large number of errors and inconsistencies. We found it remarkable that a description published as part of a standards document contained such an amount of flaws. In our opinion, this raises questions about the soundness of the standardisation procedure applied in this case.

The timeline below shows the key points of our development in chronological order. We estimate that we have spent roughly 2 900 man-hours on this project, which translates to 2.1 man-years by Norwegian standards.[1]



Timeline of key development steps

[1]Average annual hours actually worked per worker, as reported by OECD: https://stats.oecd.org/Index.aspx?DataSetCode=ANHRS

Chapter 10

References

Barkowsky, M. et al. (2007). “Temporal registration using 3D phase correlation and a maximum likelihood approach in the perceptual evaluation of video quality”. In: *Multimedia Signal Processing, 2007. MMSP 2007. IEEE 9th Workshop on*, pp. 195–198. doi: 10.1109/MMSP.2007.4412851.

Comer, D. E. et al. (1989). “Computing As a Discipline”. In: *Commun. ACM* 32.1. Ed. by P. J. Denning, pp. 9–23. issn: 0001-0782. url: http://doi.acm.org/10.1145/63238.63239.

Dosselmann, R. and X. D. Yang (2011). “A comprehensive assessment of the structural similarity index”. In: *Signal, Image and Video Processing* 5.1, pp. 81–91.

Huynh-Thu, Q. and M. Ghanbari (2008). “Scope of validity of PSNR in image/video quality assessment”. In: *Electronics Letters* 44.13, pp. 800–801. issn: 0013-5194. doi: 10.1049/el:20080522.

ITU-R (2012). *Methodology for the subjective assessment of the quality of television pictures*. url: http://www.itu.int/rec/R-REC-BT.500-13-201201-I.

ITU-T (2008a). *J.247: Objective perceptual multimedia video quality measurement in the presence of a full reference*. url: http://www.itu.int/rec/T-REC-J.247-200808-I/en.

ITU-T (2008b). *P.900: Telephone transmission quality, telephone installations, local line networks (audiovisual quality in multimedia services)*. url: https://www.itu.int/rec/T-REC-P.911-199812-I/en.

ITU-T (2008c). *Subjective video quality assessment methods for multimedia applications*. url: https://www.itu.int/rec/T-REC-P.910-200804-I/en.

ITU-T (2010). *J.340: Reference algorithm for computing peak signal to noise ratio of a processsed video sequence with compensation for* *constant spatial shifts, constant temporal shift and constant luminance gain and offset*. url: http://www.itu.int/rec/T-REC-J.340-201006-I.

ITU-T (2011). *J.341: Objective perceptual multimedia video quality measurement of HDTV for digital cable television in the presence of a full reference*. url: http://www.itu.int/rec/T-REC-J.341-201101-I/en.

Jiménez Bermejo, D. (2012). "High definition video quality assessment metric built upon full reference ratios". PhD thesis. Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid.

Kuipers, F. et al. (2010). "Techniques for Measuring Quality of Experience". English. In: *Wired/Wireless Internet Communications*, pp. 216–227. isbn: 978-3-642-13314-5. doi: 10.1007/978-3-642-13315-2_18.

Martinez-Rach, M. et al. (2006). "A study of objective quality assessment metrics for video codec design and evaluation". In: *Multimedia, 2006. ISM'06. Eighth IEEE International Symposium on*. IEEE, pp. 517–524.

Péchard, S., R. Pépion, and P. Le Callet (2008). "Suitable methodology in subjective video quality assessment: a resolution dependent paradigm". In: *International Workshop on Image Media Quality and its Applications, IMQA2008*. Kyoto, Japan, p. 6.

Pinson, M. H., N. Staelens, and A. Webster (2013). "The history of video quality model validation". English. In: *2013 IEEE 15th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, pp. 458–463. isbn: 978-1-4799-0125-8. doi: 10.1109/MMSP.2013.6659332.

Pinson, M., N. Staelens, and A. Webster (2013). "The history of video quality model validation". In: *Multimedia Signal Processing (MMSP), 2013 IEEE 15th International Workshop on*, pp. 458–463. doi: 10.1109/MMSP.2013.6659332.

Pinson, M. and S. Wolf (2004). "A new standardized method for objectively measuring video quality". In: *Broadcasting, IEEE Transactions on* 50.3, pp. 312–322. issn: 0018-9316. doi: 10.1109/TBC.2004.834028.

Pitrey, Y., M. Barkowsky, R. Pépion, et al. (2012). "Influence of the source content and encoding configuration on the perceived quality for scalable video coding". In: *SPIE Human Vision and Electronic Imaging XVII*. Vol. 8291. 54. San franscisco, United States, pp. 1–6. url: https://hal.archives-ouvertes.fr/hal-00665993.

Pitrey, Y., U. Engelke, M. Barkowsky, et al. (2011). "Aligning subjective tests using a low cost common set". In: *Euro ITV*. Lisbonne, Portugal, irccyn contribution. url: https://hal.archives-ouvertes.fr/hal-00608310.

Pitrey, Y., U. Engelke, P. Le Callet, et al. (2011). “Subjective quality of SVC-coded videos with different error-patterns concealed using spatial scalability”. In: *Third European Workshop on Visual Information Processing (EUVIP)*. Paris, France, paper number 67. url: https://hal.archives-ouvertes.fr/hal-00608300.

Pitrey, Y. et al. (2010a). “Evaluation of MPEG4-SVC for QoE protection in the context of transmission errors”. In: *SPIE Optical Engineering*. San Diego, United States. url: https://hal.archives-ouvertes.fr/hal-00608337.

Pitrey, Y. et al. (2010b). “Subjective quality assessment of MPEG-4 scalable video coding in a mobile scneario”. In: *Second European Workshop on Visual Information Processing*. Paris, France, paper 72. url: https://hal.archives-ouvertes.fr/hal-00608333.

Poynton, C. (2012). *Digital Video and HD: Algorithms and Interfaces*. 2nd ed. Morgan Kaufman.

Rimac-Drlje, S., M. Vranješ, and D. Žagar (2010). “Foveated Mean Squared Error–a Novel Video Quality Metric”. In: *Multimedia Tools Appl.* 49.3, pp. 425–445. issn: 1380-7501. doi: 10.1007/s11042-009-0442-1. url: http://dx.doi.org/10.1007/s11042-009-0442-1.

Seshadrinathan, K. and A. C. Bovik (2010). “Motion tuned spatio-temporal quality assessment of natural videos”. In: *Image Processing, IEEE Transactions on* 19.2, pp. 335–350.

Seshadrinathan, K., R. Soundararajan, et al. (2010). “A subjective study to evaluate video quality assessment algorithms”. In: *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, 75270H–75270H.

Shapira, D., S. Avidan, and Y. Hel-Or (2013). “Multiple histogram matching”. English. In: *2013 IEEE International Conference on Image Processing*. IEEE, pp. 2269–2273. url: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6738468.

VQEG (2008). *Final Report from the Video Quality Experts Group om the Validation of Objective Models of Multimedia Quality Assessment, Phase I*. Tech. rep. url: ftp://vqeg.its.bldrdoc.gov/Documents/Projects/multimedia/MM_Final_Report/VQEG_MM_Report_Final_v2.6.pdf.

Vranješ, M., S. Rimac-Drlje, and K. Grgić (2013). “Review of objective video quality metrics and performance comparison using different databases”. In: *Signal Processing: Image Communication* 28.1, pp. 1–19. issn: 0923-5965. doi: http://dx.doi.org/10.1016/j.image.2012.10.003. url: http://www.sciencedirect.com/science/article/pii/S0923596512001919.

Wang Z., Bovik, A. C. and Sheikh, H. R. and Simoncelli, E. P. (2004). “Wavelets for Image Image quality assessment: From error visibility to structural similarity”. In: *IEEE Transactions on Image Processing* 13.4, pp. 600–612. url: http://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf.

Wang, Y. (2006). “Survey of objective video quality measurements”. In: *Computer Science Faculty Publications*. url: http://digitalcommons.wpi.edu/computerscience-pubs/42/.

Wang, Z., A. C. Bovik, and E. P. Simoncelli (2005). “Structural Approaches to Image Quality Assessment”. In: *Handbook of Image and Video Processing*. 2nd. Academic Press. Chap. 8.3. url: https://ece.uwaterloo.ca/~z70wang/publications/HIVP_chapter83.pdf.

Winkler, S. and P. Mohandas (2008). “The Evolution of Video Quality Measurement: From PSNR to Hybrid Metrics”. In: *IEEE Trans. on Broadcasting* 54.3, pp. 660–668. doi: 10.1109/TBC.2008.2000733.